IRIT
Programmers' Manual

Version 13

A Solid modeling Program

EMail: gershon@cs.technion.ac.il

Join *IRIT* mailing list: gershon@cs.technion.ac.il
Mailing list: irit-mail@cs.technion.ac.il
Bug reports: irit-bugs@cs.technion.ac.il
WWW Page: http://gershon.cs.technion.ac.il/irit

Contents

Chapter 1

Introduction

This manual describes the different libraries of the IRIT solid modeling environment. Quite a few libraries can be found to manipulate geometry in general, freeform curves and surfaces, symbolic computation, trimmed surfaces, triangular patches, freeform trivariate functions, Boolean operations, input output data file parsing, and miscelleneous.

All interface to the libraries should be made via the appropriate header files that can be found in the include subdirectory. Most libraries have a single header file that is named the same as the library. Functions and constants that are visible to the users of the libraries are prefixed with a unique prefix, usually derived from the library name itself. External definitions that start with an underscore should not be used, even if found in header files.

The header file **include/irit_sm.h** must be sourced by every source file in the solid modeller. In most cases, this file is sourced indirectly via local header files.

The following libraries are avaliable in IRIT:

Name of Library	Tasks
bool	Boolean operations on polygonal models.
cagd	Low level freeform curves and surfaces.
geom	General geometry functions.
grap	General graphics/display functions.
mdl	Model's processing functions.
misc	Memory allocation, configuration files, attributes, etc.
mvar	Multi variate functions.
prsr	Input and output for file/sockets of objects of IRIT.
rndr	Scan conversion rendering functions.
symb	Symbolic manipulation of curves and surfaces.
trim	Trimmed surfaces support.
triv	Freeform trivariate functions.
trng	Triangular patches support.
user	General high level user inteface functions.
xtra	Public domain code that is not part of IRIT.

1.0.1 Tools and Programs

The IRIT package includes several complete programs such as poly3d-h (hidden line removal), irender (scan conversion tool), and irit2ps (a filter to Postscript). Somewhat different than most other programs is the kernel interpreter, also called irit. The irit program is nothing more than an interpreter (written in C) that invokes numerous functions in the several libraries provided. In order to add a new function to the irit interpreter, the following sequence of operations must be followed:

1. Write a C function that accepts only one or more of the following type of parameters. All parameters, including the IrtRType, *must* be transferred by address:
   - IrtRType (see irit_sm.h).
   - IrtVecType (see irit_sm.h).
   - IrtPtType (see irit_sm.h).
   - CagdCtlPtStruct (see cagd_lib.h).
   - IrtPlnType (see irit_sm.h).
   - StringType (char *).
   - IPObjectStruct (see iritprsr.h). This includes all object types in IPObjectStruct other than the above.

2. The written C function can return one of:
   - IrtRType by value.
   - IPObjectStruct by address.
   - Nothing (a procedure).
3. According to the returned type by the new C function, go to file **inptprsl.h**, and add a new enum NEW_FUNC for the new function in enum RealValueFuncType (for IrtRType returned value), in enum ObjValueFuncType (for IPObjectStruct * returned value), or in enum GenValueFuncType (for no returned value).
4. In **inptevl1.c**, add one new line in one of NumFuncTable, ObjFuncTable, or GenFuncTable (depends upon the return value). The line will have the form of:

```
{ ‘‘FuncName’’, NEW_FUNC, CFunctionName,  N, { Param1Type,
                                        Param2Type, ... ParamNType } }
```

   for procedures with no return values, and of the form of:

```
{ ‘‘FuncName’’, NEW_FUNC, CFunctionName,  N, { Param1Type,
                             Param2Type, ... ParamNType }, RetValType }
```

   otherwise. `N` is the number of parameters, `‘FuncName’’` is the (unique) name that will be used in the interpreter, CFunctionName is the name of the C function you have written. This C function must be declared in one of the header files that inptevl1.c includes. ParamIType, for I between 1 and N, and RetValType are of type IritExprType (see inptprsl.h).
5. Thats it!

For example, to add the C function declared as

```
IPObjectStruct *PolyhedraMoments(IPObjectStruct *IPObjectStruct,
                                 IrtRType *m);
```

one has to add the following line to ObjFuncTable in iritevl1.c

```
{ ‘‘PMOMENT’’, PMOMENT, { POLY_EXPR, REAL_EXPR }, VECTOR_EXPR },
```

where PMOMENT needs to be added to ObjValueFuncType in iritprsl.h.

While all objects in the interpreted space are of type IPObjectStruct, the functions' interface unfolds many types to simplify matter. IrtRType, IrtVecType, Strings, etc. are all extracted from the given (IPObjectStruct) parameters and passed directly by address. For returned values, only numeric real data is allowed where everything else must be returned wrapped in an IPObjectStruct.

The following chapters reference the different functions of the different libraries. Chapter 18 provides several examples of writing C code using the IRIT libraries.

Chapter 2

Boolean Library, bool_lib

2.1 General Information

On of the crucial operation in any solid modeling environment is the ability to perform Boolean operations among different geometric objects. The interface of the library is defined in *include/bool_lib.h*. This library supports only Boolean operations of polygonal objects. The Boolean operations of OR, AND, SUBtract, NEGate, CUT, and MERGE are supported via the **BoolOperType** typedef:

BoolOperType	Meaning
BOOL_OPER_OR	Union of two geometrical objects
BOOL_OPER_AND	Intersection of two geometrical objects
BOOL_OPER_SUB	The difference of two geometrical objects
BOOL_OPER_NEG	Unary Inside-out of one geometrical object
BOOL_OPER_CUT	Boundary of one object outside the other
BOOL_OPER_MERGE	Simple merge without any computation

The **BoolOperType** typedef is used in two dimensional Boolean operations invoked via **Boolean2D**. Three dimensional Boolean operations are invoked via **BooleanXXX** where XXX is one of OR, AND, SUB, NEG, CUT or MERGE, denoting the same as in the table above.

In addition several state functions are available to control the way the Boolean operations are conducted. Functions to enable the dump of (only) the intersection curves, to handle coplanar polygons, and to set the axis along which the sweep is performed are provided.

All globals in this library have a profix of **Bool**.

2.2 Algorithmic Hints Behind the Boolean Operations

Let $\mathcal{M}_i,\ i = 1, 2$ be two polygonal models to compute a Boolean operation between and let P_m denote the m'th polygon of $\mathcal{M}_i$. Denote by $InterList(P_m),\ P_m \in \mathcal{M}_i$, the list of line segments representing the intersections of polygon P_m with polygons in $\mathcal{M}_j$. A naive first, intersection, step in a Boolean operation between two polyhedra models, $\mathcal{M}_1$ and $\mathcal{M}_2$, will consist of the following stages:

```
For each polygon P_m in M_1
begin
    InterList(P_m) ⇐ ∅
end
For each polygon P_n in M_2
begin
    InterList(P_n) ⇐ ∅
end

For each polygon P_m in M_1
begin
    For each polygon P_n in M_2
    begin
        If ( P_m ∩ P_n )
        begin
```

```
            L ⇐ P_m ∩ P_n;
            InterList(P_m) ⇐ InterList(P_m) ∪ L;
            InterList(P_n) ⇐ InterList(P_n) ∪ L;
        end
    end
end
```

At the end of the first, intersection stage, each polygon, P_m in either $\mathcal{M}_1$ (or $P_n \in \mathcal{M}_2$) contains in its $InterList(P_m)$ the set of line segments of the intersection(s) of P_m with all polygons in the other model. In order to figure out orientation, each intersection line segment also contains an orientation hint as to the inside direction of the two polygons that intersect at that line segment. This list of intersection line segments can now be processed:

```
For each polygon P_m in M_1
begin
    Polys ⇐ InterList(P_m) sorted into polylines;
    OpenList(P_m) ⇐ Open polylines of Polys;
    ClosedList(P_m) ⇐ Closed polylines of Polys;
end
For each polygon P_n in M_2
begin
    Polys ⇐ InterList(P_n) sorted into polylines;
    OpenList(P_n) ⇐ Open polylines of Polys;
    ClosedList(P_n) ⇐ Closed polylines of Polys;
end
```

Open polylines, $OpenList(P_m)$, are intersection polylines that are open. These polylines starts on the boundary of polygon P_m and also ends on the boundary, at a different boundary location though. In contrast, the set of closed polylines, $ClosedList(P_n)$, contains closed loops that are completely contained in P_n.

The polygons are now ready for *trimming*. Depending upon the Boolean operation that is requested, the inside or the outside of $\mathcal{M}_1$ or $\mathcal{M}_2$ is selected for the output result. Denote by $\mathcal{M}_i\ Out\ \mathcal{M}_j$ ($\mathcal{M}_i\ In\ \mathcal{M}_j$) the region of $\mathcal{M}_i$ that is outside (inside) $\mathcal{M}_j$. Then,

Boolean Operation	implementation as
$\mathcal{M}_1 \cup \mathcal{M}_1$	$\{\mathcal{M}_1\ Out\ \mathcal{M}_2\} \cup \{\mathcal{M}_2\ Out\ \mathcal{M}_1\}$
$\mathcal{M}_1 \cap \mathcal{M}_1$	$\{\mathcal{M}_1\ In\ \mathcal{M}_2\} \cup \{\mathcal{M}_2\ In\ \mathcal{M}_1\}$
$\mathcal{M}_1 - \mathcal{M}_1$	$\{\mathcal{M}_1\ Out\ \mathcal{M}_2\} \cup Inverse\{\mathcal{M}_2\ In\ \mathcal{M}_1\}$

Note the $Inverse$ operation that reverses the orientation inside out.

At this point, all Boolean operations are translated into two complementary type of operations, In and Out. The computation of $\mathcal{M}_i\ Out\ \mathcal{M}_j$ is conducted by tracing the outside regions of the polygons of $\mathcal{M}_i$ that *intersect* $\mathcal{M}_j$ and *propagating* this inside–outside information to the non intersecting polygons:

```
For each polygon P_m in M_1 with non empty OpenList(P_m)
                                or non empty ClosedList(P_m)
begin
    P_m^in ⇐ Inside region of P_m;
    P_m^out ⇐ Outside region of P_m;

    For each original edge, e ∈ P_m^in
    begin
        Push e onto InsideStack;
    end
    For each original edge, e ∈ P_m^out
    begin
        Push e onto OutSideStack;
    end
end

While InsideStack not empty
```

```
begin
    e ⇐ edge poped from top of InsideStack;
    P_m, P_l ⇐ two polygons sharing e in M_1;

    If ( P_m has empty OpenList(P_m) and empty ClosedList(P_m),
                                                and P_m is not classified yet )
    begin
        Classify P_m as inside;
        push all edges of P_m, but e, onto InsideStack;
    end
    If ( P_l has empty OpenList(P_l) and empty ClosedList(P_l),
                                                and P_l is not classified yet )
    begin
        Classify P_l as inside;
        push all edges of P_l, but e, onto InsideStack;
    end
end

While OutsideStack not empty
begin
    e ⇐ edge poped from top of outsideStack;
    P_m, P_l ⇐ two polygons sharing e in M_1;

    If ( P_m has empty OpenList(P_m) and empty ClosedList(P_m),
                                                and P_m is not classified yet )
    begin
        Classify P_m as outside;
        push all edges of P_m, but e, onto OutsideStack;
    end
    If ( P_l has empty OpenList(P_l) and empty ClosedList(P_l),
                                                and P_l is not classified yet )
    begin
        Classify P_l as outside;
        push all edges of P_l, but e, onto OutsideStack;
    end
end
```

At the end of this process, M_1 is completely split into two sets of polygons, classified as either inside M_2 or outside M_2. An identical spliting process can be applied to $\mathcal{M}_2$. The proper inside/outside lists of $\mathcal{M}_1$ can now be combined with the proper list of classified polygons of $\mathcal{M}_2$ to yield the proper result of the requested Boolean operation.

While this completes the Boolean operation, vast room for improvements can be found, with some improvements implemented in IRIT. For example, the first step of the intersection computation necessitates an $O(n^2)$ polygon–polygon intersection tests, where n is in the order of the number of polygons in $\mathcal{M}_i$. Clearly, one can hope for a better time complexity. Bounding boxes, hierarchical bounding boxes, or three dimensional space sweep techniques can all be applied to reduce the overhead invested in these $O(n^2)$ tests. Efficient sorting of line segments within a polygon, when forming the open and closed loops is another optimization consideration.

2.3 Library Functions

2.3.1 BoolClnAdjacencies (adjacncy.c:695)

```
void BoolClnAdjacencies(IPObjectStruct *PObj)
```

PObj: Polygon object to clean adjacency information.

Returns: void

Description: Clean the adjacency pointers in the given polygonal model.
See also: BoolGenAdjacencies,

2.3.2 BoolCutPolygonAtRay (bool2low.c:430)

Booleans

```
IPVertexStruct *BoolCutPolygonAtRay(IPPolygonStruct *Pl, IrtPtType Pt)
```

Pl: The polygon to compute the ray intersection with.

Pt: The origin of the ray point.

Returns: The added vertex on Pl where the intersection with the ray occurred.

Description: Finds the intersection of the ray fired from Pt to +X direction with the given polygon. Note Pt MUST be in the polygon. Two vertices equal to ray/polygon intersection point are added to polygon vertex list, and a pointer to the first one is also returned. The polygon is NOT assumed to be convex and we look for the minimum X intersection. The polygon might not be convex as a result of combining some other closed loop before we got to do this test.

2.3.3 BoolDebugPrintAdjacencies (adjacncy.c:908)

```
void BoolDebugPrintAdjacencies(IPObjectStruct *PObj)
```

PObj: To debug print its adjacency information.

Returns: void

Description: Prints the adjacency information of an object.

2.3.4 BoolDescribeError (bool_err.c:62)

error handling

```
const char *BoolDescribeError(BoolFatalErrorType ErrorNum)
```

ErrorNum: Type of the error that was raised.

Returns: A string describing the error type.

Description: Returns a string describing a the given error. Errors can be raised by any member of this bool library as well as other users. Raised error will cause an invocation of BoolFatalError function which decides how to handle this error. BoolFatalError can for example, invoke this routine with the error type, print the appropriate message and quit the program.

2.3.5 BoolDfltFatalError (bool-hi.c:1482)

error handling

```
void BoolDfltFatalError(BoolFatalErrorType ErrID)
```

ErrID: Error type that was raised.

Returns: void

Description: Defalu fatal error trapping rotuine of the Boolean library.
See also: BoolSetFatalErrorFunc,

2.3.6 BoolExtractPolygons (bool2low.c:962)

Booleans

```
IPObjectStruct *BoolExtractPolygons(IPObjectStruct *PObj, int AinB)
```

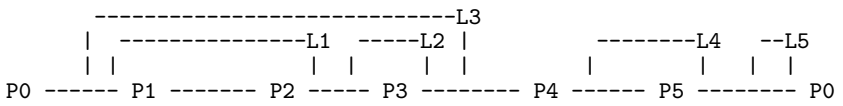
PObj: Object that need to be rebuilt according to the intersection curves that were found, both closed and open loops.

AinB: Type of inclusion/exclusion requested.

Returns: The newly created clipped object.

Description: This routine coordinates all the extraction of the polygons from the intersecting lists. Does it in the following steps:

1. Mark all polygons with no intersection at all as complete polygons. (this is because this polygon will be totally in or out, according to inter-polygon adjacencies propagation...) Also mark them as undefined (if in output or undefined) yet. Uses IPPolygonStruct Tags to save these bits.
2. do
   2.1. Convert the unordered segment list of each polygon to closed loops (if create a hole in polygon) or open (if crosses its boundary).
   2.2. Order the open loops along the perimeter of the polygon (note these loops cannot intersect. For example (5 vertices polygon):



   Note L1, L2 are enclosed in L3 loop, and that the order is circular.
   2.3. "Break" the polygon at each open loop that has no enclosed loops in it. For example we can start at L1, L2, L4, L5 and then L3. "Break" means - replace the vertex chain between the two loop end points, with the loops itself. Depends upon the relation required we may need to output a new polygon form from the deleted chain and that loop. In addition we may form a new polygon from last loop and was was left from the original polygon For each formed polygon, for each complete edge of it (i.e. edge which was originally in the polygon) test the adjacent polygon if it is complete (as marked in 1.) and if in or undefined (marked undefined in 1.) is still undefined:
      2.3.1. set it to be in.
      2.3.2. push it on adjacency stack.
   2.4. For each closed loop - find in which polygon (from polygons created in 2.3.) it is enclosed, and decompose it.
3. While adjacencies stack not empty do:
   3.1. pop top polygon from stack and output it.
   3.2. For each of its edges (which obviousely must be complete edges) if adjacent polygon is complete and undefined:
      3.3.1. set it to be in.
      3.3.2. push it on adjacency stack.
   3.3 go back to 3.

The above algorithm defines in as in output, but dont be confused with the required inter-object AinB (or AoutB if FALSE), which used to determine which side of the trimming loop should be output. Note this routine may return non-convex polygons (but marked as so) even though the input for the booleans must be convex polygons only! In order to keep the given object unchanged, a whole new copy off the polygon list is made. The polygons of the list that are not in the output are freed: a global list of all polygons (pointers) is used to scan them in the end and free the unused ones (list PolysPtr).

2.3.7 BoolFilterCollinearities (bool-2d.c:273)

```
int BoolFilterCollinearities(IPPolygonStruct *Pl)
```

Pl: To filter, in place. The polygon is assumed to have a circular vertex list.

Returns: TRUE if the polygon has been modified, FALSE otherwise.

Description: Filters out collinear edges and duplicated vertices, in the given polygon.

2.3.8 BoolGenAdjacencies (adjacncy.c:102)

adjacency

topology

```
int BoolGenAdjacencies(IPObjectStruct *PObj)
```

PObj: The polygonal object to compute the adjacency information for.

Returns: TRUE if all adjacencies were resolved, or the object is completely closed.

Description: Routine to generate adjacencies to the given object. Note an edge might be only partially adjacent to another edge, and a second attempt is made to find (again only part of - see below) them. Any case, FALSE will be returned as there is no way we can say the object is perfectly closed! This is the only routine to generate the adjacencies of a geometric object. These adjacencies are needed for the Boolean operations on them. Algorithm: for each edge, for each polygon in the object, the edges are sorted according to the key defined by EdgeKey routine (sort in hash tbl). A second path on the table is made to match common keys edges and set the pointers from one to another. Note that each edge is common to exactly 2 faces if it is internal, or exactly 1 face if it is on the border (if the object is open).
See also: BoolGetAdjEdge, BoolClnAdjacencies,

2.3.9 BoolGetAdjEdge (adjacncy.c:852)

```
IPVertexStruct *BoolGetAdjEdge(IPVertexStruct *V)
```

V: Vertex of a mesh defining edge (V, V->Pnext) to extract its adjacent edge, if any.

Returns: First vertex of adjacent edge to edge (V, V->Pnext) or NULL if none.

Description: Given a polygonal mesh with adjacent information, find the adjacent edge (VAdj, VAdj->Pnext) that is adjacent to edge (V, V->Pnext), if any.
See also: BoolGenAdjacencies,

2.3.10 BoolGetDisjointPart (adjacncy.c:816)

```
IPPolygonStruct *BoolGetDisjointPart(IPObjectStruct *PObj, int Index)
```

PObj: Object to extract part number Index from.

Index: Index of part to fetch from PObj.

Returns: Extract polygonal list with disjoint number Index.

Description: Get the disjoint part number Index from object PObj.
See also: BoolMarkDisjointParts,

2.3.11 BoolInterPolyPoly (bool1low.c:918)

```
IPPolygonStruct *BoolInterPolyPoly(IPPolygonStruct *Pl1, IPPolygonStruct *Pl2)
```

Pl1: First polygon to compute intersection for.

Pl2: Second polygon to compute intersection for.

Returns: The intersection segment, if any, NULL otherwise.

Description: Routine to intersect polygon Pl1, with polygon Pl2. If found common intersection, that segment will be added to the InterSegmentStruct list saved in Pl1 PAux list. Note that as the two polygons convex, at most one line segment can result from such intersection (of two non coplanar polygons). Algorithm: intersect all Pl2 edges with Pl1 plane. If found that (exactly) two vertices (one segment) of Pl2 do intersect Pl1 plane then: Perform clipping of the segment against Pl1. If result is not empty, add the result segment to Pl1 InterSegmentStruct list (saved at PAux of polygon - see IPPolygonStruct).

2.3.12 BoolLoopsFromInterList (bool1low.c:1310)

```
int BoolLoopsFromInterList(IPPolygonStruct *Pl,
                           InterSegListStruct **PClosed,
                           InterSegListStruct **POpen)
```

Pl: Polygon with intersection information in its PAux slot.

PClosed: To be updated with the closed loops found in Pl.

POpen: To be updated with the open loops found in Pl.

Returns: TRUE if has loops.

Description: Given a polygon with the intersection list, creates the polylines loop(s) out of it, which can be one of the two:

1. Closed loop - all the intersections create a loop in one polygon.
2. Open polyline - if the intersections cross the polygon boundary. In this case the two end point of the polyline, must lay on polygon boundary.

In both cases, the polyline will be as follows: First point at first list element at PtSeg[0] (see InterSegmentStruct). Second point at first list element at PtSeg[1] (see InterSegmentStruct). Point i at list element (i-1) at PtSeg[0] (PtSeg[1] is not used!). In the closed loop case the last point is equal to first. Both cases returns NULL terminated list.

2.3.13 BoolMarkDisjointParts (adjacncy.c:731)

```
int BoolMarkDisjointParts(IPObjectStruct *PObj)
```

PObj: To anaylize for different disjoint parts.

Returns: Number of disjoint parts.

Description: Mark polygons in the given object based on their association with different disjoint object parts.
See also: BoolGetDisjointPart,

2.3.14 BoolSetFatalErrorFunc (bool-hi.c:1453)

error handling

```
BoolFatalErrorFuncType BoolSetFatalErrorFunc(BoolFatalErrorFuncType ErrFunc)
```

ErrFunc: New error trapping function to use.

Returns: Old error trapping function.

Description: Sets the fatal error trapping routine of the boolean library.
See also: BoolDfltFatalError,

2.3.15 BoolSetHandleCoplanarPoly (bool-hi.c:1393)

Booleans

```
int BoolSetHandleCoplanarPoly(int HandleCoplanarPoly)
```

HandleCoplanarPoly: If TRUE, coplanar polygons are handled.

Returns: Old value.

Description: Controls if coplanar polygons should be handled or not.
See also: BooleanOR, BooleanAND, BooleanSUB, BooleanCUT, BooleanMERGE, BooleanNEG, , Boolean2D, BoolSetOutputInterCurve, BoolSetPolySortAxis, , BoolSetParamSurfaceUVVals, BoolSetPerturbAmount,

2.3.16 BoolSetOutputInterCurve (bool-hi.c:1330)

Booleans

```
int BoolSetOutputInterCurve(int OutputInterCurve)
```

OutputInterCurve: If TRUE only intersection curves are computed, If false, full blown Boolean is applied.

Returns: Old value.

Description: Controls if intersection curves or full Boolean operation is to be performed.
See also: BooleanOR, BooleanAND, BooleanSUB, BooleanCUT, BooleanMERGE, BooleanNEG, , Boolean2D, BoolSetHandleCoplanarPoly, BoolSetPolySortAxis, , BoolSetParamSurfaceUVVals, BoolSetPerturbAmount,

2.3.17 BoolSetParamSurfaceUVVals (bool-hi.c:1424)

Booleans

```
int BoolSetParamSurfaceUVVals(int HandleBoolParamSrfUVVals)
```

HandleBoolParamSrfUVVals: If TRUE, UV values are to be returned.

Returns: Old value.

Description: Controls if UV paprameter values of original surface should be returned.
See also: BooleanOR, BooleanAND, BooleanSUB, BooleanCUT, BooleanMERGE, BooleanNEG, , Boolean2D, BoolSetOutputInterCurve, BoolSetHandleCoplanarPoly, , BoolSetPolySortAxis, BoolSetPerturbAmount,

2.3.18 BoolSetPerturbAmount (bool-hi.c:1363)

Booleans

```
IrtRType BoolSetPerturbAmount(IrtRType PerturbAmount)
```

PerturbAmount: Perturbation amount of objects before reattempting Booleans that ends up empty.

Returns: Old value.

Description: Controls the perturbation amount, if any, of the second object to improve the success likelihood. Perturbations are applied once the Booleans return with an empty intersection.
See also: BooleanOR, BooleanAND, BooleanSUB, BooleanCUT, BooleanMERGE, BooleanNEG, , Boolean2D, BoolSetHandleCoplanarPoly, BoolSetPolySortAxis, , BoolSetParamSurfaceUVVals, BoolSetOutputInterCurve,

2.3.19 BoolSetPolySortAxis (bool1low.c:688)

Booleans

```
int BoolSetPolySortAxis(int PolySortAxis)
```

PolySortAxis: Sorting axis. Either 0(x), 1 (y), or 2 (z).

Returns: Old value.

Description: Routine to set polygonal sorting axis.

2.3.20 BoolSortOpenInterList (bool1low.c:1574)

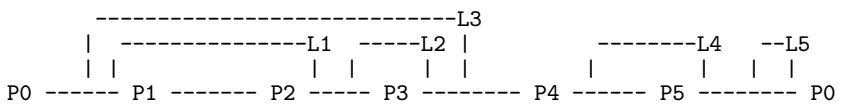
```
void BoolSortOpenInterList(IPPolygonStruct *Pl, InterSegListStruct **POpen)
```

Pl: To sort the loops for.

POpen: The set of open loops. Updated in place.

Returns: void

Description: Sorts the open loops of given polygon to an order that can be used in subdividing into sub polygons later (see comment of BoolExtractPolygons). This order is such that each loops will have no other loop between its end points, if we walk along the polygon in the (linked list direction) perimeter from one end to the other, before it. For example:



In this case, any order such that L1, L2 are before L3 will do. Obviously this is not a total order, and they are few correct ways to sort it. Algorithm: For each open loop, for each of its two end, evaluate a IrtRType key for the end point P between segment P(i) .. P(i+1) to be i + t, where: t is the ratio (P - P(i)) / (P(i+1) - P(i)) . This maps the all perimeter of the polygon onto 0..N-1, where N is number of vertices of that polygon. Sort the keys, and while they are keys in data structure, search and remove a consecutive pair of keys associated with same loop, and output it. Note that each open loop point sequence is tested to be such that it starts on the first point (first and second along vertex list) on polygon perimeter, and the sequence end is on the second point, and the sequence is reversed if not so. This order will make the replacement of the perimeter from first to second points by the open loop much easier. This may be real problem if there are two intersection points almost identical - floating point errors may cause it to loop forever. We use some reordering heuristics in this case, and return fatal error if fail!

2.3.21 Boolean2D (bool-2d.c:70)

Booleans

```
IPPolygonStruct *Boolean2D(IPPolygonStruct *Pl1,
                           IPPolygonStruct *Pl2,
                           BoolOperType BoolOper)
```

Pl1: First convex polygon to compute 2D Boolean for.

Pl2: Second convex polygon to compute 2D Boolean for.

BoolOper: Boolean operation requested (and, or, etc.)

Returns: The resulting Boolean operation.

Description: Given two convex polygons assumed to be in the same plane, compute their 2D Boolean operation BoolOper and return it as a new polygon(s). NULL is returned if an error occur (No intersection or invalid BoolOper).
See also: BooleanOR, BooleanAND, BooleanSUB, BooleanCUT, BooleanMERGE, BooleanNEG, , BoolSetHandleCoplanarPoly, BoolSetOutputInterCurve, , Boolean2DComputeInters,

2.3.22 Boolean2DComputeInters (bool-2d.c:563)

Booleans

```
Bool2DInterStruct *Boolean2DComputeInters(IPPolygonStruct *Poly1,
                                          IPPolygonStruct *Poly2,
                                          int HandlePolygons,
                                          int DetectIntr)
```

Poly1, Poly2: The two polygons/lines to intersect.

HandlePolygons: If polygons, needs to handle normals etc.

DetectIntr: If TRUE, return non NULL dummy pointer if the two polys do indeed intersect. For detection of intersection!

Returns: Intersection information.

Description: Given two polygons/lines, Detect all edges in Pl1 that intersect with edges in Pl2. Returned is the information about all intersections as a Bool2DInter structure list.
See also: Boolean2D,

2.3.23 BooleanAND (bool-hi.c:477)

Booleans

```
IPObjectStruct *BooleanAND(const IPObjectStruct *PObjIn1,
                           const IPObjectStruct *PObjIn2)
```

PObjIn1: First object to perform the Boolean operation on.

PObjIn2: Second object to perform the Boolean operation on.

Returns: The result of the Boolean operation.

Description: Performs a Boolean AND between two objects.
See also: BooleanOR, BooleanSUB, BooleanCUT, BooleanMERGE, BooleanNEG, , Boolean2D, BoolSetOutputInterCurve, BoolSetHandleCoplanarPoly, , BoolSetPolySortAxis, BoolSetParamSurfaceUVVals, BooleanSELF, , BooleanCONTOUR, BoolSetPerturbAmount,

2.3.24 BooleanCONTOUR (bool-hi.c:807)

Booleans

```
IPObjectStruct *BooleanCONTOUR(const IPObjectStruct *PObjIn, IrtPlnType Plane)
```

PObjIn: Object to perform the contouring operation on.

Plane: Plane to use in the contouring.

Returns: The result of the contouring operation.

Description: Performs a contouring of the given poly object with the given plane.
See also: BooleanMultiCONTOUR, BooleanOR, BooleanAND, BooleanSUB, BooleanMERGE, , BooleanNEG, Boolean2D, BoolSetOutputInterCurve, , BoolSetHandleCoplanarPoly, BoolSetPolySortAxis, , BoolSetParamSurfaceUVVals, BooleanSELF, BooleanCUT, BoolSetPerturbAmount,

2.3.25 BooleanCUT (bool-hi.c:720)

Booleans

```
IPObjectStruct *BooleanCUT(const IPObjectStruct *PObjIn1,
                           const IPObjectStruct *PObjIn2)
```

PObjIn1: First object to perform the Boolean operation on.

PObjIn2: Second object to perform the Boolean operation on.

Returns: The result of the Boolean operation.

Description: Performs a Boolean CUT between two objects.
See also: BooleanOR, BooleanAND, BooleanSUB, BooleanMERGE, BooleanNEG, , Boolean2D, BoolSetOutputInterCurve, BoolSetHandleCoplanarPoly, , BoolSetPolySortAxis, BoolSetParamSurfaceUVVals, BooleanSELF, , BooleanCONTOUR, BoolSetPerturbAmount,

2.3.26 BooleanComputeRotatedPolys (bool-hi.c:1244)

Booleans

```
IPPolygonStruct *BooleanComputeRotatedPolys(IPPolygonStruct *Pl,
                                            int CopyOnePl,
                                            IrtHmgnMatType RotMat)
```

Pl: Polygon(s) to transform. Assumed to be convex.

CopyOnePl: Should we copy? Also if FALSE Pl might be non convex!

RotMat: Transformation matrix.

Returns: Transformed polygon(s).

Description: Routine to optionally copy (if CpolyOnePl) a single polygon and rotate according to the rotation matrix provided. If, however, CopyOnePl is False all polygons in list are converted.

2.3.27 BooleanLow1In2 (bool1low.c:224)

Booleans

```
IPObjectStruct *BooleanLow1In2(IPObjectStruct *PObj1, IPObjectStruct *PObj2)
```

PObj1: First object of Boolean operation.

PObj2: Second object of Boolean operation.

Returns: Result of one in two.

Description: Finds the part of PObj1 which is in of PObj2:
See also: BooleanLow1Out2, BooleanLowSelfInOut,

2.3.28 BooleanLow1Out2 (bool1low.c:159)

Booleans

```
IPObjectStruct *BooleanLow1Out2(IPObjectStruct *PObj1, IPObjectStruct *PObj2)
```

PObj1: First object of Boolean operation.

PObj2: Second object of Boolean operation.

Returns: Result of one out two.

Description: Finds the part of PObj1 which is out of PObj2:
See also: BooleanLow1In2, BooleanLowSelfInOut,

2.3.29 BooleanLowSelfInOut (bool1low.c:276)

self intersection

```
IPObjectStruct *BooleanLowSelfInOut(IPObjectStruct *PObj, int InOut)
```

Booleans

PObj: Object of Boolean operation with itself (self intersection).

InOut: What are we looking for? in or out.

Returns: Result of Boolean in/out with self.

Description: Finds the part of PObj which is in/out of itself:
See also: BooleanLow1Out2, BooleanLow1In2,

2.3.30 BooleanMERGE (bool-hi.c:870)

Booleans

```
IPObjectStruct *BooleanMERGE(const IPObjectStruct *PObjIn1,
                             const IPObjectStruct *PObjIn2)
```

PObjIn1: First object to perform the Boolean operation on.

PObjIn2: Second object to perform the Boolean operation on.

Returns: The result of the Boolean operation.

Description: Performs a Boolean MERGE between two objects.
See also: BooleanOR, BooleanAND, BooleanSUB, BooleanCUT, BooleanNEG, , Boolean2D, BoolSetOutputInterCurve, BoolSetHandleCoplanarPoly, , BoolSetPolySortAxis, BoolSetParamSurfaceUVVals, BooleanSELF, , BooleanCONTOUR, BoolSetPerturbAmount,

2.3.31 BooleanMultiCONTOUR (bool_multi_cntr.c:147)

Booleans

```
IPObjectStruct *BooleanMultiCONTOUR(IrtRType CntrLevel,
                                    BooleanMultiCntrGenInfoStruct *GI)
```

CntrLevel: Level to contour at.

GI: The local data for contouring as initialized by the init function.

Returns: The result of the contouring operation.

Description: Performs multiple contouring of the given poly object with parallel planes. Planes must be X/Y/Z orthogonal planes.
See also: BooleanCONTOUR, BooleanOR, BooleanAND, BooleanSUB, BooleanMERGE, , BooleanNEG, Boolean2D, BoolSetOutputInterCurve, , BoolSetHandleCoplanarPoly, BoolSetPolySortAxis, , BoolSetParamSurfaceUVVals, BooleanSELF, BooleanCUT, BoolSetPerturbAmount,

2.3.32 BooleanMultiCONTOURFree (bool_multi_cntr.c:117)

Booleans

```
void BooleanMultiCONTOURFree(BooleanMultiCntrGenInfoStruct *GI)
```

GI: The local data for contouring to ve freed.

Returns: void

Description: Initializes the multiple contouring of the given poly object with parallel planes. Planes must be X/Y/Z constant.
See also: BooleanMultiCONTOUR, BooleanMultiCONTOURInit, , BooleanCONTOUR, BooleanOR, BooleanAND, BooleanSUB, BooleanMERGE, , BooleanNEG, Boolean2D, BoolSetOutputInterCurve, , BoolSetHandleCoplanarPoly, BoolSetPolySortAxis, , BoolSetParamSurfaceUVVals, BooleanSELF, BooleanCUT, BoolSetPerturbAmount,

2.3.33 BooleanMultiCONTOURInit (bool_multi_cntr.c:53)

Booleans

```
BooleanMultiCntrGenInfoStruct *BooleanMultiCONTOURInit(
                                                const IPObjectStruct *PObj,
                                                int Axis)
```

PObj: Poly object to perform the multi-contouring operation on.

Axis: 0, 1, 2 for X/Y/Z orthogonal planes to use.

Returns: Local data initialized for contouring.

Description: Initializes the multiple contouring of the given poly object with parallel planes. Planes must be X/Y/Z constant.
See also: BooleanMultiCONTOUR, BooleanMultiCONTOURFree, , BooleanCONTOUR, BooleanOR, BooleanAND, BooleanSUB, BooleanMERGE, , BooleanNEG, Boolean2D, BoolSetOutputInterCurve, , BoolSetHandleCoplanarPoly, BoolSetPolySortAxis, , BoolSetParamSurfaceUVVals, BooleanSELF, BooleanCUT, BoolSetPerturbAmount,

2.3.34 BooleanNEG (bool-hi.c:925)

Booleans

```
IPObjectStruct *BooleanNEG(const IPObjectStruct *PObjIn)
```

PObjIn: Object to negate.

Returns: The result of the Boolean operation.

Description: Performs a Boolean NEG of an objects. Negation is simply reversing the direction of the plane equation of each polygon - the simplest Boolean operation...
See also: BooleanOR, BooleanAND, BooleanSUB, BooleanCUT, BooleanMERGE, , Boolean2D, BoolSetOutputInterCurve, BoolSetHandleCoplanarPoly, , BoolSetPolySortAxis, BoolSetParamSurfaceUVVals, BooleanSELF, , BooleanCONTOUR, BoolSetPerturbAmount,

2.3.35 BooleanOR (bool-hi.c:339)

Booleans

```
IPObjectStruct *BooleanOR(const IPObjectStruct *PObjIn1,
                          const IPObjectStruct *PObjIn2)
```

PObjIn1: First object to perform the Boolean operation on.

PObjIn2: Second object to perform the Boolean operation on.

Returns: The result of the Boolean operation.

Description: Performs a Boolean OR between two objects.
See also: BooleanAND, BooleanSUB, BooleanCUT, BooleanMERGE, BooleanNEG, Boolean2D, , BoolSetOutputInterCurve, BoolSetHandleCoplanarPoly, BoolSetPolySortAxis, , BoolSetParamSurfaceUVVals, BooleanSELF, , BooleanCONTOUR, BoolSetPerturbAmount,

2.3.36 BooleanPrepObject (bool1low.c:571)

```
int BooleanPrepObject(IPObjectStruct *PObj)
```

PObj: To prepare, in place.

Returns: TRUE if successful, FALSE otherwise.

Description: Routine to prepare the polygons before the Booleans: verify the plane equations, and compute BBox for all polygons in provided object. Also, the polygons are sorted in the list with according to their minimal BBox value in GlblPolySortAxis axis.

2.3.37 BooleanSELF (bool-hi.c:987)

Booleans

```
IPObjectStruct *BooleanSELF(const IPObjectStruct *PObjIn)
```

PObjIn: Object to perform the self intersecting Boolean operation on.

Returns: The result of the Boolean operation.

Description: Performs a Boolean Self intersection operation.
See also: BooleanOR, BooleanAND, BooleanSUB, BooleanMERGE, BooleanNEG, , Boolean2D, BoolSetOutputInterCurve, BoolSetHandleCoplanarPoly, , BoolSetPolySortAxis, BoolSetParamSurfaceUVVals, BooleanCUT, , BooleanCONTOUR, BoolSetPerturbAmount,

2.3.38 BooleanSUB (bool-hi.c:589)

Booleans

```
IPObjectStruct *BooleanSUB(const IPObjectStruct *PObjIn1,
                           const IPObjectStruct *PObjIn2)
```

PObjIn1: First object to perform the Boolean operation on.

PObjIn2: Second object to perform the Boolean operation on.

Returns: The result of the Boolean operation.

Description: Performs a Boolean SUBtracion between two objects.
See also: BooleanOR, BooleanAND, BooleanCUT, BooleanMERGE, BooleanNEG, , Boolean2D, BoolSetOutputInterCurve, BoolSetHandleCoplanarPoly, , BoolSetPolySortAxis, BoolSetParamSurfaceUVVals, BooleanSELF, , BooleanCONTOUR, BoolSetPerturbAmount,

Chapter 3

CAGD Library, cagd_lib

3.1 General Information

This library provides a rich set of function to create, convert, display and process freeform Bezier and NURBs curves and surfaces. The interface of the library is defined in *include/cagd_lib.h*. This library mainly supports low level freeform curve and surface operations. Supported are curves and surfaces from scalars to five dimensions as E1/P1 to E5/P5 using the **CagdPointType**. Pi is a rational (projective) version of Ei, with an additional W coefficient. Polynomial in the power basis have some very limited support as well. Different data structures to hold UV parameter values, control points, vectors, planes, bounding boxes, polylines and polygons are defined as well as the data strcutures to hold the curves and surfaces themselves,

```
typedef struct CagdCrvStruct {
    struct CagdCrvStruct *Pnext;
    struct IPAttributeStruct *Attr;
    CagdGeomType GType;
    CagdPointType PType;
    int Length;            /* Number of control points (== order in Bezier). */
    int Order;     /* Order of curve (only for Bspline, ignored in Bezier). */
    CagdBType Periodic;    /* Valid only for Bspline curves. */
    CagdRType *Points[CAGD_MAX_PT_SIZE];     /* Pointer on each axis vector. */
    CagdRType *KnotVector;
} CagdCrvStruct;

typedef struct CagdSrfStruct {
    struct CagdSrfStruct *Pnext;
    struct IPAttributeStruct *Attr;
    CagdGeomType GType;
    CagdPointType PType;
    int ULength, VLength;  /* Mesh size in the tensor product surface. */
    int UOrder, VOrder;   /* Order in tensor product surface (Bspline only). */
    CagdBType UPeriodic, VPeriodic;      /* Valid only for Bspline surfaces. */
    CagdRType *Points[CAGD_MAX_PT_SIZE];     /* Pointer on each axis vector. */
    CagdRType *UKnotVector, *VKnotVector;
} CagdSrfStruct;
```

Curves and surfaces have a geometric type **GType** to prescribe the type of entity (such as CAGD_SBEZIER_TYPE for Bezier surface) and a point type **PType** to prescribe the point type of the entity (such as CAGD_PT_E3_TYPE for three dimensional Euclidean control points). **Length** and **Order** slots are used to hold the number of control points in the mesh and or control polygon and the order(s) of the basis functions. **Periodic** flag(s) are used to denote periodic end conditions. In addition, **KnotVector** slot(s) are used if the entity exploits Bspline basis functions, or NULL otherwise.

The control polygon and/or mesh itself is organized in the **Points** slot as a vector of size **CAGD_MAX_PT_SIZE** of vectors of **CagdRType**s. For surfaces, the mesh is ordered U first and the macros of **CAGD_NEXT_U** **CAGD_NEXT_V**, and **CAGD_MESH_UV** can be used to determine indices in the mesh.

All structures in the cagd library can be allocated using New constrcutures (i.e. **CagdUVNew** or **CagdCrfNew**, freed using Free destructores (i.e. **CagdSrfFree** or CagdBBoxFree, linked list free using FreeList destructores (i.e. **CagdPolylineFreeList**), and copied using copy constructores i.e. **CagdPtCopy** or **CagdCtlPtCopyList**).

This library has its own error handler, which by default prints an error message and exit the program called **CagdFatalError**.

Most globals in this library have a prefix of **Cagd** for general cagd routines. Prefix of **Bzr** is used for Bezier routines, prefix of **Bsp** for Bspline specific routines, prefix of **Cnvrt** for conversion routines, and **Afd** for adaptive forward differencing routines.

3.2 Library Functions

3.2.1 AfdApplyAntiLStep (afd_cube.c:230)

forward differencing

```
void AfdApplyAntiLStep(CagdRType Coef[4])
```

Coef: Four coefficients of the AFD basis functions.

Returns: void

Description: Given four coefficents of a cubic afd polynomial, apply the anti L step, in place.
See also: AfdApplyLStep, AfdApplyEStep, AfdApplyLn, , AfdCnvrtCubicBzrToAfd, AfdComputePolyline, AfdBzrCrvEvalToPolyline,

3.2.2 AfdApplyEStep (afd_cube.c:180)

forward differencing

```
void AfdApplyEStep(CagdRType Coef[4])
```

Coef: Four coefficients of the AFD basis functions.

Returns: void

Description: Given four coefficents of a cubic afd polynomial, apply the E (step 1) in place.
See also: AfdApplyAntiLStep, AfdApplyLStep, AfdApplyLn, , AfdCnvrtCubicBzrToAfd, AfdComputePolyline, AfdBzrCrvEvalToPolyline,

3.2.3 AfdApplyLStep (afd_cube.c:205)

forward differencing

```
void AfdApplyLStep(CagdRType Coef[4])
```

Coef: Four coefficients of the AFD basis functions.

Returns: void

Description: Given four coefficents of a cubic afd polynomial, apply the L step, in place.
See also: AfdApplyAntiLStep, AfdApplyEStep, AfdApplyLn, , AfdCnvrtCubicBzrToAfd, AfdComputePolyline, AfdBzrCrvEvalToPolyline,

3.2.4 AfdApplyLn (afd_cube.c:81)

forward differencing

```
void AfdApplyLn(CagdRType Coef[4], int n)
```

Coef: Four coefficients of the AFD basis functions.

n: How many times to compute the L transform.

Returns: void

Description: Given four coefficents of a cubic afd polynomial, apply the L (half the step size) n times to them, in place. We basically precomputed L^n and apply it here once. Every instance of L half the domain and so L^n divides the domain by 2^n.
See also: AfdApplyAntiLStep, AfdApplyLStep, AfdApplyEStep, , AfdCnvrtCubicBzrToAfd, AfdComputePolyline, AfdBzrCrvEvalToPolyline,

3.2.5 AfdBzrCrvEvalToPolyline (afd_cube.c:313)

forward differencing

```
void AfdBzrCrvEvalToPolyline(const CagdCrvStruct *Crv,
                             int FineNess,
                             CagdRType *Points[])
```

Crv: A cubic Bezier curve to piecewise linear sample using AFD.

FineNess: Of samples.

Points: Where to place the piecewise linear approximation. Assumed to be valid with respect to the dimension of Crv.

Returns: void

Description: Samples the curves at FineNess location equally spaced in the Bezier parametric domain [0..1]. If Cache is enabled, and FineNess is power of two, up to or equal to CacheFineNess, the cache is used, otherwise the points are evaluated manually for each of the samples. Data is saved at the Points array of vectors (according to Curve PType), each vector is assumed to be allocated for FineNess CagdRType points. Bezier curve must be cubic.
See also: AfdApplyAntiLStep, AfdApplyLStep, AfdApplyEStep, AfdApplyLn, , AfdCnvrtCubicBzrToAfd, AfdComputePolyline,

3.2.6 AfdCnvrtCubicBzrToAfd (afd_cube.c:49)

forward differencing

```
void AfdCnvrtCubicBzrToAfd(CagdRType Coef[4])
```

Coef: Converts, in place, cubic Bezier Coef to AFD Coef.

Returns: void

Description: Given four coefficents of a cubic Bezier curve, computes the four coefficients of the cubic afd basis functions, in place.
See also: AfdApplyAntiLStep, AfdApplyLStep, AfdApplyEStep, AfdApplyLn, , AfdComputePolyline, AfdBzrCrvEvalToPolyline,

3.2.7 AfdComputePolyline (afd_cube.c:265)

forward differencing

```
void AfdComputePolyline(CagdRType Coef[4],
                        CagdRType *Poly,
                        int Log2Step,
                        CagdBType NonAdaptive)
```

Coef: Four coefficients of a cubic Bezier curve.

Poly: Where to put the polyline computed.

Log2Step: How many steps to take (2 to the power of this).

NonAdaptive: if TRUE, ignore the adaptive option.

Returns: void

Description: Given four coefficents of a cubic Bezier curve, computes the four coefficients of the cubic afd basis functions and step along them to create a piecewise polynomial approximating the curve. If NonAdaptive is TRUE then 2^Log2Step constant steps are taken, creating 2^Log2Step + 1 points along the curve. Otherwise the full blown adaptive algorithm is used.
See also: AfdApplyAntiLStep, AfdApplyLStep, AfdApplyEStep, AfdApplyLn, , AfdCnvrtCubicBzrToAfd, AfdBzrCrvEvalToPolyline,

3.2.8 BBoxDiagonalInitCrvCalculator (cbsp_fit.c:2035)

InitFittingCrvCalculatorFuncType

bounding box diagonal

```
static CagdCrvStruct *BBoxDiagonalInitCrvCalculator(CagdPType *PtList,
                                                    int NumOfPoints,
                                                    int Length,
                                                    int Order,
                                                    CagdBType Periodic)
```

PtList: Points cloud.

NumOfPoints: Number of points in PtList.

Length: The desired length of the output b-spline curve.

Order: The desired order of the output b-spline curve.

Periodic: TRUE for periodic output curve, FALSE for open end.

Returns: B-spline curve which is a bounding box diagonal [/] of the points cloud.

Description: Computes an initial b-spline fitting curve which is a diagonal of the points cloud bounding box.

3.2.9 BBoxPerimeterInitCrvCalculator (cbsp_fit.c:1504)

InitFittingCrvCalculatorFuncType

bounding box diagonal

```
static CagdCrvStruct *BBoxPerimeterInitCrvCalculator(CagdPType *PtList,
                                                     int NumOfPoints,
                                                     int Length,
                                                     int Order,
                                                     CagdBType Periodic)
```

PtList: Points cloud.

NumOfPoints: Number of points in PtList.

Length: The desired length of the output b-spline curve.

Order: The desired order of the output b-spline curve.

Periodic: TRUE for periodic output curve, FALSE for open end.

Returns: B-spline curve with control points equally spread on a perimeter of the points cloud bounding box.

Description: Computes an initial b-spline fitting curve which control points lies on a perimeter of the points cloud bounding box

3.2.10 BspBasisFuncMultEval (bspcoxdb.c:330)

```
CagdBspBasisFuncEvalStruct *BspBasisFuncMultEval(const CagdRType *KnotVector,
                                                 int KVLength,
                                                 int Order,
                                                 CagdBType Periodic,
                                                 CagdRType *Params,
                                                 int NumOfParams,
                                                 CagdBspBasisFuncMultEvalType
                                                                    EvalType)
```

KnotVector: Knot sequence defining the spline space.

KVLength: Length of KnotVector.

Order: Of the spline space.

Periodic: TRUE if space is periodic.

Params: At which to evaluate and compute the spline functions.

NumOfParams: Size of Params vector.

EvalType: Type of evaluation requested: value (position), 1st derivative, or 2nd derivative.

Returns: A vector of size NumOfParams of evaluation results, each holding the index of the first non zero basis function and the (at most) Order non zero basis function values.

Description: Computes multiple evaluations of the given spline space basis functions, as prescribed by KnotVector and Order, at the requested NumOfParams parameter values, Params.

See also:

3.2.11 BspBasisFuncMultEvalFree (bspcoxdb.c:473)

```
void BspBasisFuncMultEvalFree(CagdBspBasisFuncEvalStruct *Evals,
                              int NumOfParams)
```

Evals: Structure to free.

NumOfParams: Size of Evals - number of parameter evaluations we have.

Returns: void

Description: Frees the allocated structure for multiple evaluations.
See also:

3.2.12 BspBasisFuncMultEvalPrint (bspcoxdb.c:441)

```
void BspBasisFuncMultEvalPrint(const CagdBspBasisFuncEvalStruct *Evals,
                               int Order,
                               CagdRType *Params,
                               int NumOfParams)
```

Evals: Structure to print.

Order: Of evaluated basis functions.

Params: Parameters the basis functions were evaluated at.

NumOfParams: Size of Evals/Params - number of parameters/evaluations.

Returns: void

Description: Prints to stdout the allocated structure for multiple evaluations.
See also:

3.2.13 BspC1Srf2PolygonsSamples (bsp2poly.c:431)

polygonization

surface approximation

```
CagdBType BspC1Srf2PolygonsSamples(const CagdSrfStruct *Srf,
                                   int FineNess,
                                   CagdBType ComputeNormals,
                                   CagdBType ComputeUV,
                                   CagdRType **PtWeights,
                                   CagdPtStruct **PtMesh,
                                   CagdVecStruct **PtNrml,
                                   CagdUVStruct **UVMesh,
                                   int *FineNessU,
                                   int *FineNessV)
```

Srf: To sample in a grid.

FineNess: Control on accuracy, the higher the finer.

ComputeNormals: If TRUE, normal information is also computed.

ComputeUV: If TRUE, UV values are stored and returned as well.

PtWeights: Weights of the evaluations, if rational, to detect poles. NULL if surface nor rational.

PtMesh: Evaluted positions of grid of samples.

PtNrml: Evaluted normals of grid of samples or NULL if none.

UVMesh: Evaluted UV vals of grid of samples or NULL if none.

FineNessU, FineNessV: Actual size of PtMesh, PtNrml, UVMesh.

Returns: FALSE is returned in case of an error, TRUE otherwise.

Description: Routine to uniformly sample a single C1 continuous Bspline srf as grid. FineNess is a fineness control on the result and the larger it is more samples may result. A value of 10 is a good starting value. FALSE is returned in case of an error, TRUE otherwise.
See also: BzrSrf2Polygons, IritSurface2Polygons, IritTrimSrf2Polygons, , CagdSrf2Polygons, TrimSrf2Polygons, BspSrf2PolygonsSamplesNuNv,

3.2.14 BspCrv2Polyline (bsp2poly.c:928)

piecewise linear approximation

polyline

```
CagdPolylineStruct *BspCrv2Polyline(const CagdCrvStruct *Crv,
                                    int SamplesPerCurve,
                                    BspKnotAlphaCoeffStruct *A,
                                    CagdBType OptiLin)
```

Crv: To approximate as a polyline.

SamplesPerCurve: Number of samples to approximate with.

A: Alpha matrix (Oslo algorithm) if precomputed.

OptiLin: If TRUE, optimize linear curves.

Returns: A polyline representing the piecewise linear approximation from, or NULL in case of an error.

Description: Routine to approx. a single B-spline curve as a polyline with SamplesPerCurve samples. Polyline is always E3 CagdPolylineStruct type. Curve is refined equally spaced in parametric space, unless the curve is linear in which the control polygon is simply being copied. If A is specified, it is used to refine the curve. NULL is returned in case of an error, otherwise CagdPolylineStruct.
See also: BzrCrv2Polyline, BspSrf2Polylines, IritCurve2Polylines, , SymbCrv2Polyline,

3.2.15 BspCrvAllEuclideanC1Discont (cagd_aux.c:2208)

```
CagdPtStruct *BspCrvAllEuclideanC1Discont(const CagdCrvStruct *Crv,
                                          IrtBType EuclideanC1Discont,
                                          IrtRType Tolerance)
```

Crv: To subdivide at all C^1 discontinuity locations.

EuclideanC1Discont: TRUE to compute the C1 discontinuities and verify them in the Euclidean space. FALSE to only consider C1 discontinuities by knot multiplicity in parametric space.

Tolerance: Of parametric C1 discontinuity that is also a Euclidean discontinuity - deviation from inner product of unit tangents before and after discontinuity by less than Tolerance (1.0 if tangent identical, 0.0 if orthogonal, -1.0 if opposite). Ignored if < -1.0 or EuclideanC1Discont is FALSE.

Returns: Locations of C1 discontinuities.

Description: Compute, for the given curve, all C^1 potential discontinuity locations.
See also: CagdCrvSubdivAtParams, BspKnotAllC1Discont, , BspCrvsSubdivAtAllDetectedLocations, CagdCrvSubdivAtAllC0Discont, , CagdCrvSubdivAtAllC1Discont,

3.2.16 BspCrvBiNormalMalloc (cbsp_aux.c:794)

binormal

```
CagdVecStruct *BspCrvBiNormalMalloc(const CagdCrvStruct *Crv,
                                    CagdRType t,
                                    CagdBType Normalize)
```

Crv: Crv for which to compute a (unit) binormal.

t: The parameter at which to compute the unit binormal.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, length is a function of given parametrization.

Returns: A pointer to a vector holding the binormal information. Always an E3 vector.

Description: Returns a (unit) vector, equal to the binormal to Crv at parameter value t. Algorithm: insert (order - 1) knots and using 3 consecutive control points at the refined location (p1, p2, p3), compute to binormal to be the cross product of the two vectors (p1 - p2) and (p2 - p3). Since a curve may have not BiNormal at inflection points or if the 3 points are collinear, NULL will be returned at such cases.

3.2.17 BspCrvBiNormalToData (cbsp_aux.c:748)

binormal

```
CagdVecStruct *BspCrvBiNormalToData(const CagdCrvStruct *Crv,
                                    CagdRType t,
                                    CagdBType Normalize,
                                    CagdVecStruct *Vec)
```

Crv: Crv for which to compute a (unit) binormal.

t: The parameter at which to compute the unit binormal.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, length is a function of given parametrization.

Vec: A pointer to a vector holding the binormal information

Returns: A pointer to a vector holding the binormal information. Always an E3 vector.

Description: Returns a (unit) vector, equal to the binormal to Crv at parameter value t. Algorithm: insert (order - 1) knots and using 3 consecutive control points at the refined location (p1, p2, p3), compute to binormal to be the cross product of the two vectors (p1 - p2) and (p2 - p3). Since a curve may have not BiNormal at inflection points or if the 3 points are collinear, NULL will be returned at such cases.

3.2.18 BspCrvCoxDeBoorBasis (bspcoxdb.c:169)

evaluation

B-splines

```
CagdRType *BspCrvCoxDeBoorBasis(const CagdRType *KnotVector,
                                int Order,
                                int Len,
                                CagdBType Periodic,
                                CagdRType t,
                                int *IndexFirst,
                                CagdRType *Basis)
```

KnotVector: To evaluate the B-spline Basis functions for this space.

Order: Of the geometry.

Len: Number of control points in the geometry. The length of KnotVector is equal to Len + Order (+(Order-1) if periodic).

Periodic: TRUE if freeform is periodic.

t: At which the B-spline basis functions are to be evaluated.

IndexFirst: Index of the first B-spline basis function that might be non zero.

Basis: Where evaluated basis functions are going to be saved.

Returns: A vector of length Order thats holds the values of the B-spline basis functions for the given t. A B-spline of order Order might have at most Order non zero basis functions that will hence start at IndexFirst and up to (*IndexFirst + Order - 1). Same as Vec.

Description: Returns a pointer to a vector of size Order, holding values of the non zero basis functions of a given curve at given parametric location t. This vector SHOULD NOT BE FREED. Although it is dynamically allocated, the returned pointer does not point to the beginning of this memory and it it be maintained by this routine (i.e. it might be freed next time this routine is being called). IndexFirst returns the index of first non zero basis function for the given parameter value t. Uses the recursive Cox de Boor algorithm, to evaluate the B-spline basis functions. Algorithm: Use the following recursion relation with B(i,0) == 1.

$$B(i,k) = \frac{t - t(i)}{t(i+k-1) - t(i)} B(i,k-1) + \frac{t(i+k) - t}{t(i+k) - t(i+1)} B(i+1,k-1)$$

Starting with constant B-spline (k == 0) only one basis function is non zero and is equal to one. This is the constant B-spline spanning interval t(i)...t(i+1) such that t(i) <= t < t(i+1). We then raise this constant B-spline to the prescribed Order and find in this process all the basis functions that are non zero in t for order Order. Sound simple hah!?

3.2.19 BspCrvCoxDeBoorIndexFirst (bspcoxdb.c:282)

evaluation

B-splines

```
int BspCrvCoxDeBoorIndexFirst(const CagdRType *KnotVector,
                              int Order,
                              int Len,
                              CagdRType t)
```

KnotVector: To evaluate the B-spline Basis functions for this space.

Order: Of the geometry.

Len: Number of control points in the geometry. The length of KnotVector is equal to Len + Order.

t: At which the B-spline basis functions are to be evaluated.

Returns: The index.

Description: Computes the index of the first non zero basis function as returned by the BspCrvCoxDeBoorBasis function.

3.2.20 BspCrvCreateApproxHelix (cagd_arc.c:838)

```
CagdCrvStruct *BspCrvCreateApproxHelix(CagdRType NumOfLoops,
                                       CagdRType Pitch,
                                       CagdRType Radius,
                                       int Sampling,
                                       int CtlPtsPerLoop)
```

NumOfLoops: Number of loops in the helix - can be fractional.

Pitch: Essentially the size of the helix. A Pitch of one will step one unit in Z for one full circle.

Radius: Radius of helix. If radius is negative, the radius will change monotonically from zero to abs(Radius) at the end.

Sampling: Number of samples to compute on the helix. Should be several hundreds for a reasonable result.

CtlPtsPerLoop: Number of control points to use per loop. Use at least 5 for a reasonable approximation.

Returns: A helix B-spline curve approximation.

Description: Constructs an approximated polynomial helix curve, along the +Z axis.
See also: BspCrvCreateCircle, BspCrvCreateApproxSine, BspCrvCreateApproxSpiral,

3.2.21 BspCrvCreateApproxSine (cagd_arc.c:893)

```
CagdCrvStruct *BspCrvCreateApproxSine(CagdRType NumOfCycles,
                                      int Sampling,
                                      int CtlPtsPerCycle)
```

NumOfCycles: Number of cycles in the sine - can be fractional.

Sampling: Number of samples to compute on the sine. Should be several hundreds for a reasonable result.

CtlPtsPerCycle: Number of control points to use per cycle. Use at least 5 for a reasonable approximation.

Returns: A sine wave Bspline curve approximation.

Description: Constructs an approximated polynomial sine curve.
See also: BspCrvCreateCircle, BspCrvCreateApproxSine, BspCrvCreateApproxHelix,

3.2.22 BspCrvCreateApproxSpiral (cagd_arc.c:780)

```
CagdCrvStruct *BspCrvCreateApproxSpiral(CagdRType NumOfLoops,
                                        CagdRType Pitch,
                                        int Sampling,
                                        int CtlPtsPerLoop)
```

NumOfLoops: Number of loops in the spiral - can be fractional.

Pitch: Essentially the size of the spiral. A Pitch of one will construct a roughly size-one spiral curve.

Sampling: Number of samples to compute on the spiral. Should be several hundreds for a reasonable result.

CtlPtsPerLoop: Number of control points to use per loop. Use at least 5 for a reasonable approximation.

Returns: A spiral B-spline curve approximation.

Description: Constructs an approximated spiral curve (not rational!)
See also: BspCrvCreateCircle, BspCrvCreateApproxSine, BspCrvCreateApproxHelix,

3.2.23 BspCrvCreateCircle (cagd_arc.c:364)

circle

```
CagdCrvStruct *BspCrvCreateCircle(const CagdPtStruct *Center, CagdRType Radius)
```

Center: Of circle to be created, NULL for origin.

Radius: Of circle to be created.

Returns: A circle centered at Center and radius Radius that is parallel to the XY plane represented as a rational quadratic B-spline curve.

Description: Creates a circle at the specified position as a rational quadratic B-spline curve. Circle is always parallel to the XY plane.
See also: BspCrvCreateUnitCircle, BspCrvCreatePCircle, BspCrvCreateUnitPCircle, , BspCrvCreateUnitPCircleQuadTol, BspCrvCreateUnitPCircleCubicTol, , CagdCreateConicCurve, CagdCrvCreateArc, BzrCrvCreateArc,

3.2.24 BspCrvCreatePCircle (cagd_arc.c:479)

circle

```
CagdCrvStruct *BspCrvCreatePCircle(const CagdPtStruct *Center,
                                   CagdRType Radius)
```

Center: Of circle to be created, NULL for origin.

Radius: Of circle to be created.

Returns: A circle approximation centered at Center and radius Radius that is parallel to the XY plane represented as a polynomial cubic B-spline curve.

Description: Approximates a circle as a cubic polynomial B-spline curve at the specified position and radius. Construct the circle as four 90 degrees arcs of polynomial cubic Bezier segments using predefined constants. See Faux & Pratt "Computational Geometry for Design and Manufacturing" for a polynomial approximation to a circle.
See also: BspCrvCreateCircle, BspCrvCreateUnitCircle, BspCrvCreateUnitPCircle, , BspCrvCreateUnitPCircleQuadTol, BspCrvCreateUnitPCircleCubicTol, , CagdCreateConicCurve, CagdCrvCreateArc, BzrCrvCreateArc,

3.2.25 BspCrvCreatePCircleTol (cagd_arc.c:731)

circle

```
CagdCrvStruct *BspCrvCreatePCircleTol(const CagdPtStruct *Center,
                                      CagdRType Radius,
                                      int Order,
                                      int Cont,
                                      CagdRType Tol)
```

Center: Of circle to be created.

Radius: Of circle to be created.

Order: Of approximating polynomial - can be either 3 (quadratic) or 4 (cubic).

Cont: Desired continuity: 0 for C^0 between the arc segments, 1 for C^1 continuity between the arc segments.

Tol: Tolerance of approximation - no less than 1e-15 for a unit size circle.

Returns: A circle approximation centered at Center and radius Radius that is parallel to the XY plane represented as a polynomial cubic B-spline curve.

Description: Approximates a circle as a quadratic or cubic polynomial B-spline curve at the specified position and radius, and of required Order and continuity. Construct the circle as n arcs of polynomial quadratic/cubic Bezier segments using predefined constants. See Faux & Pratt "Computational Geometry for Design and Manufacturing" for a polynomial approximation to a circle.

See also: BspCrvCreateCircle, BspCrvCreateUnitCircle, BspCrvCreateUnitPCircle, , BspCrvCreateUnitPCircleQuadTol, BspCrvCreateUnitPCircleCubicTol, , CagdCreateConicCurve, CagdCrvCreateArc, BzrCrvCreateArc,

3.2.26 BspCrvCreateUnitCircle (cagd_arc.c:311)

circle

```
CagdCrvStruct *BspCrvCreateUnitCircle(void)
```

Returns: A rational quadratic B-spline curve representing a unit circle.

Description: Creates a circle at the specified position as a rational quadratic B-spline curve. Constructs a unit circle as 4 90 degrees arcs of rational quadratic Bezier segments using a predefined constants.

See also: BspCrvCreateCircle, BspCrvCreatePCircle, BspCrvCreateUnitPCircle, , BspCrvCreateUnitPCircleQuadTol, BspCrvCreateUnitPCircleCubicTol, , CagdCreateConicCurve, CagdCrvCreateArc, BzrCrvCreateArc,

3.2.27 BspCrvCreateUnitPCircle (cagd_arc.c:403)

circle

```
CagdCrvStruct *BspCrvCreateUnitPCircle(void)
```

Returns: A cubic polynomial B-spline curve approximating a unit circle

Description: Approximates a unit circle as a cubic polynomial B-spline curve. Construct a circle as four 90 degrees arcs of polynomial cubic Bezier segments using predefined constants. See Faux & Pratt "Computational Geometry for Design and Manufacturing" for a polynomial approximation to a circle.

See also: BspCrvCreateCircle, BspCrvCreateUnitCircle, BspCrvCreatePCircle, , BspCrvCreateUnitPCircleQuadTol, BspCrvCreateUnitPCircleCubicTol, , CagdCreateConicCurve, CagdCrvCreateArc, BzrCrvCreateArc, BspCrvCreateApproxSpiral,

3.2.28 BspCrvCreateUnitPCircleCubicTol (cagd_arc.c:634)

circle

```
CagdCrvStruct *BspCrvCreateUnitPCircleCubicTol(CagdRType Tol, int Cont)
```

Tol: Tolerance of approximation, Tol > 1e-13. Sets the maximally allowed deviation from the exact unit square.

Cont: 0 or 1 for C^0 or C^1 continuity. Ignored as result is C^1 & G^2.

Returns: A rational quadratic B-spline curve representing a unit circle, to within Tol.

Description: Creates a cubic polynomial B-spline curve that approximates a unit circle to a specified tolerance. Implements method 2 from: "Good Approximation of Circles by-curvature-continuous Bezier curves" by Dokken, Lyche, and Morken, CAGD, June 1990, pp 33-41.

See also: BspCrvCreateCircle, BspCrvCreatePCircle, BspCrvCreateUnitPCircle, , CagdCreateConicCurve, CagdCrvCreateArc, BzrCrvCreateArc, , BspCrvCreateUnitPCircleQuadTol,

3.2.29 BspCrvCreateUnitPCircleQuadTol (cagd_arc.c:521)

circle

```
CagdCrvStruct *BspCrvCreateUnitPCircleQuadTol(CagdRType Tol, int Cont)
```

Tol: Tolerance of approximation, Tol > 1e-13. Sets the maximally allowed deviation from the exact unit square.

Cont: 0 or 1 for C^0 or C^1 continuity.

Returns: A rational quadratic B-spline curve representing a unit circle, to within Tol. int: Desired continuity: 0 for C^0 for some better accuracy or 1 for C^1 continuity requiring more control points.

Description: Creates a quadratic polynomial B-spline curve that approximates a uni circle to a specified tolerance. Implements methos 2 and 5 from: "Planar curve offset based on circle approximation", CAD Vol 28, No 8, pp 617-630, 1996, by Lee, Kim and Elber.

See also: BspCrvCreateCircle, BspCrvCreatePCircle, BspCrvCreateUnitPCircle, , CagdCreateConicCurve, CagdCrvCreateArc, BzrCrvCreateArc, , BspCrvCreateUnitPCircleCubicTol,

3.2.30 BspCrvDegreeRaise (cbsp_aux.c:464)

degree raising

```
CagdCrvStruct *BspCrvDegreeRaise(const CagdCrvStruct *Crv)
```

Crv: To raise it degree by one.

Returns: A curve with one degree higher representing the same geometry as Crv.

Description: Returns a new curve, identical to the original but with one degree higher.

3.2.31 BspCrvDegreeRaiseN (cbsp_aux.c:407)

degree raising

```
CagdCrvStruct *BspCrvDegreeRaiseN(const CagdCrvStruct *Crv, int NewOrder)
```

Crv: To raise its degree to a NewOrder.

NewOrder: NewOrder for Crv.

Returns: A curve of order NewOrder representing the same geometry as Crv.

Description: Returns a new curve, identical to the original but with order N. Degree raise is computed by multiplying by a constant 1 curve of order

3.2.32 BspCrvDerive (cbsp_aux.c:993)

derivatives

```
CagdCrvStruct *BspCrvDerive(const CagdCrvStruct *Crv, CagdBType DeriveScalar)
```

Crv: To differentiate.

DeriveScalar: TRUE to differentiate each channel independently. FALSE to treat geometry as a rational differentiation.

Returns: Differentiated curve.

Description: Returns a new curve, equal to the given curve, differentiated once. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one then:

```
Q(i) = Degree * (P(i+1) - P(i)) / (Kv(i + k) - Kv(i + 1)), i = 0 to k-2.
```

See also: BzrCrvDerive, CagdCrvDerive, SymbCrvDeriveRational, , BspCrvDeriveScalar,

3.2.33 BspCrvDeriveScalar (cbsp_aux.c:1073)

derivatives

```
CagdCrvStruct *BspCrvDeriveScalar(const CagdCrvStruct *Crv)
```

Crv: To differentiate.

Returns: Differentiated curve.

Description: Returns a new curve, equal to the given curve, differentiated once. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one then:

```
Q(i) = (k - 1) * (P(i+1) - P(i)) / (Kv(i + k) - Kv(i + 1)), i = 0 to k-2.
```

For a Euclidean curve this is the same as CagdCrvDerive but for a rational curve the returned curve is not the vector field but simply the derivatives of all the curve's coefficients, including the weights.

See also: BzrCrvDerive, CagdCrvDerive, SymbCrvDeriveRational, , BspCrvDerive, BzrCrvDeriveScalar, CagdCrvDeriveScalar,

3.2.34 BspCrvDomain (bsp_gen.c:218)

domain

```
void BspCrvDomain(const CagdCrvStruct *Crv, CagdRType *TMin, CagdRType *TMax)
```

parametric domain

Crv: To get its parametric domain.

TMin: Where to put the minimal domain's boundary.

TMax: Where to put the maximal domain's boundary.

Returns: void

Description: Returns the parametric domain of a Bspline curve.
See also: CagdCrvDomain,

3.2.35 BspCrvEvalAtParamMalloc (cbspeval.c:134)

evaluation

```
CagdRType *BspCrvEvalAtParamMalloc(const CagdCrvStruct *Crv, CagdRType t)
```

Crv: To evaluate at the given parametric location t.

t: The parameter value at which the curve Crv is to be evaluated.

Returns: A vector holding all the coefficients of all components of curve Crv's point type. If for example the curve's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1). This vector is allocated dynamically.

Description: Returns a pointer to a dynamically allocated data, holding the value of the curve at given parametric location t. The curve is assumed to be B-spline. Uses the Cox de Boor recursive algorithm.
See also: CagdCrvEvalToData, BzrCrvEvalAtParamToData, BzrCrvEvalVecAtParam, , BspCrvEvalVecAtParam, BspCrvEvalCoxDeBoorToData, CagdCrvEvalToPolyline, BspCrvEvalAtParamToData,

3.2.36 BspCrvEvalAtParamToData (cbspeval.c:101)

evaluation

```
void BspCrvEvalAtParamToData(const CagdCrvStruct *Crv,
                             CagdRType t,
                             CagdRType *R)
```

Crv: To evaluate at the given parametric location t.

t: The parameter value at which the curve Crv is to be evaluated.

R: vector holding all the coefficients of all components of curve Crv's point type. If for example the curve's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Returns: void

Description: Computes the value of the curve position at parameter t, into R. The curve is assumed to be B-spline. Uses the Cox de Boor recursive algorithm.
See also: CagdCrvEvalToData, BzrCrvEvalAtParamToData, BzrCrvEvalVecAtParam, , BspCrvEvalVecAtParam, BspCrvEvalCoxDeBoorToData, CagdCrvEvalToPolyline, , BspCrvEvalAtParamToData,

3.2.37 BspCrvEvalCoxDeBoorMalloc (bspcoxdb.c:106)

evaluation

```
CagdRType *BspCrvEvalCoxDeBoorMalloc(const CagdCrvStruct *Crv, CagdRType t)
```

B-splines

Crv: To evaluate at the given parametric location t.

t: The parameter value at which the curve Crv is to be evaluated.

Returns: A vector holding all the coefficients of all components of curve Crv's point type. If for example the curve's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Description: Returns a pointer to a dynamically allocated data, holding the value of the curve at the prescribed parametric location t. Uses the recursive Cox de-Boor algorithm, to evaluate the spline, which is not very efficient if many evaluations of the same curve are necessary Use knot insertion when multiple evaluations are to be performed.

3.2.38 BspCrvEvalCoxDeBoorToData (bspcoxdb.c:43)

evaluation

B-splines

```
void BspCrvEvalCoxDeBoorToData(const CagdCrvStruct *Crv,
                               CagdRType t,
                               CagdRType *Pt)
```

Crv: To evaluate at the given parametric location t.

t: The parameter value at which the curve Crv is to be evaluated.

Pt: A vector holding all the coefficients of all components of curve Crv's point type. If for example the curve's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Returns: void

Description: Evaluates into a pointer of data, holding the value of the curve at the prescribed parametric location t. Uses the recursive Cox de-Boor algorithm, to evaluate the spline, which is not very efficient if many evaluations of the same curve are necessary Use knot insertion when multiple evaluations are to be performed.

3.2.39 BspCrvEvalVecAtParam (cbspeval.c:42)

evaluation

```
CagdRType BspCrvEvalVecAtParam(const CagdRType *Vec,
                               int VecInc,
                               const CagdRType *KnotVector,
                               int Order,
                               int Len,
                               CagdBType Periodic,
                               CagdRType t)
```

Vec: Coefficents of a scalar Bspline univariate function.

VecInc: Step to move along Vec.

KnotVector: Knot vector of associated geoemtry.

Order: Order of associated geometry.

Len: Length of control vector.

Periodic: If this geometry is Periodic.

t: Parameter value where to evaluate the curve.

Returns: Geometry's value at parameter value t.

Description: Assumes Vec holds control points for scalar Bspline curve of order Order length Len and knot vector KnotVector. Evaluates and returns that curve value at parameter value t. Vec is incremented by VecInc (usually by 1) after each iteration.

3.2.40 BspCrvExtension (bsp_gen.c:758)

```
CagdCrvStruct *BspCrvExtension(const CagdCrvStruct *OrigCrv,
                               const CagdBType *ExtDirs,
                               CagdRType Epsilon,
                               CagdBType RemoveExtraKnots)
```

OrigCrv: The curve to be extended.

ExtDirs: A boolean array of size 2 to determine the required directions of extension MinDmn, MaxDmn. A NULL here means (TRUE, TRUE).

Epsilon: The length of the requested extension, in the param. domain.

RemoveExtraKnots: If FALSE, the resulting curve will not have minimal multiplicity at the first internal knot on the extension side.

Returns: The new extended curve.

Description: Extends a B-spline curve. The domain of Crv is extended, such that the trace coincides with the input curve's trace over the original domain. An interface function for the one-sided curve extension function. OrigCrv can have "ExtntScl" real attributes to scale the extension in Euclidean space (for same Epsilon parametric extension.

See also: BspCrvExtraKnotRmv, BspSrfExtension, BspCrvExtensionOneSide,

3.2.41 BspCrvExtensionOneSide (bsp_gen.c:822)

```
CagdCrvStruct *BspCrvExtensionOneSide(const CagdCrvStruct *OrigCrv,
                                      CagdBType MinDmn,
                                      CagdRType Epsilon,
                                      CagdRType ExtntScl,
                                      CagdBType RemoveExtraKnots)
```

OrigCrv: The curve to be extended.

MinDmn: TRUE for min domain extension, FALSE for max domain extension.

Epsilon: The length of the extension in the domain.

ExtntScl: A scaling factor on the extension (for the same epsilon in the parametric domain).

RemoveExtraKnots: If FALSE, the resulting curve will not have minimal multiplicity at the first internal knot on the extension side.

Returns: The new extended curve.

Description: Extends a B-spline curve, at the min/max end. The domain of Crv is extended, such that the trace coincides with the original trace over the original domain. Assumes Crv has open end conditions. If ExtntScl != 1, the curve might also undergo refinements.

See also: BspCrvExtraKnotRmv, BspSrfExtension,

3.2.42 BspCrvExtraKnotRmv (bsp_gen.c:1017)

```
CagdCrvStruct *BspCrvExtraKnotRmv(const CagdCrvStruct *Crv,
                                  int RmvIndex)
```

Crv: The curve to be updated.

RmvIndex: The index in the knot vector of the knot to be removed.

Returns: The new updated curve.

Description: Reverse operation of the knot insertion, assuming it is guaranteed that the knot currently does not have its minimal multiplicity, that is: it is possible to remove it and maintain the exact trace. Indices and conventions follow Boehm's Knot Insertion algo, as in E. Cohen, R.F. Riesenfeld, G. Elber, "Geometric Modeling with Splines: an Introduction", CH 07.

See also: BspCrvExtensionOneSide, BspSrfExtension,

3.2.43 BspCrvFitLstSqr (cbsp_int.c:1021)

```
CagdCrvStruct *BspCrvFitLstSqr(const CagdCrvStruct *Crv,
                               int Order,
                               int Size,
                               CagdBType Periodic,
                               CagdParametrizationType ParamType,
                               CagdBType EndPtInterp,
                               CagdBType EvalPts,
                               CagdRType *Err)
```

Crv: Curve to fit a new curve to.

Order: Of the to be created curve.

Size: Control polygon size of the to be created curve.

Periodic: Constructed curve should be Periodic.

ParamType: Type of parametrization.

EndPtInterp: TRUE to force Crv's end point interpolation. Has affect only if has open end-conditions.

EvalPts: TRUE to evaluate the samples points at equal parametric interval, FALSE to simply copy the control points.

Err: The maximum error is updated into here

Returns: Fitted curve.

Description: Fits a curve to the give curve by sampling points on Crv and fitting a curve of orders Order and Size control points. Error is measured by the difference between the original and the fitted surface, as maximum error norm.

See also: BspCrvInterpPts,

3.2.44 BspCrvHasBezierKV (bsp_knot.c:42)

conversion

```
CagdBType BspCrvHasBezierKV(const CagdCrvStruct *Crv)
```

Crv: To check for KV that mimics Bezier polynomial curve.

Returns: TRUE if same as Bezier curve, FALSE otherwise.

Description: Returns TRUE iff the given curve has no interior knot open end KV.

3.2.45 BspCrvHasOpenEC (bsp_knot.c:105)

open end conditions

```
CagdBType BspCrvHasOpenEC(const CagdCrvStruct *Crv)
```

Crv: To check for open end conditions.

Returns: TRUE, if curve has open end conditions, FALSE otherwise.

Description: Returns TRUE iff the given B-spline curve has open end conditions.
See also: BspSrfHasOpenEC, ,

3.2.46 BspCrvIntegrate (cbsp_aux.c:1115)

integrals

```
CagdCrvStruct *BspCrvIntegrate(const CagdCrvStruct *Crv)
```

Crv: Curve to integrate.

Returns: Integrated curve.

Description: Returns a new Bspline curve, equal to the integral of the given Bspline crv. The given Bspline curve should be nonrational.

$$C(t) = \sum_{i=0}^{l} P_i B_{i,n}(t) = \sum_{i=0}^{l} P_i \int B_{i,n}(t) = \sum_{i=0}^{l} \frac{P_i}{n+1-i} \sum_{j=i+1}^{l+1} (t^{j+n} - t^{j+n}) B_{j,j}(t) =$$

$$\frac{1}{n+1} \sum_{j=1}^{l+1} \sum_{i=0}^{j-1} P_i (t^{j+n} - t^{j+n}) B_{j,j}(t)$$

See also: BzrCrvIntegrate, BspSrfIntegrate, CagdCrvIntegrate,

3.2.47 BspCrvInterpBuildKVs (cbsp_int.c:328)

```
void BspCrvInterpBuildKVs(const CagdCtlPtStruct *PtList,
                          int Order,
                          int CrvSize,
                          CagdParametrizationType ParamType,
                          CagdBType Periodic,
                          CagdRType **RetPtKnots,
                          CagdRType **RetKV)
```

PtList: List of point to interpolate.

Order: Order of interpolating curve.

CrvSize: Number of control points in interpolating curve.

ParamType: Parametrization type: Uniform, chord length, etc.

Periodic: TRUE for a periodic interpolating curve

RetPtKnots: Parameter values assigned to the interpolation.

RetKV: Knot sequence built for the interpolation, unless CAGD_KV_NODAL_PARAM in which case the KV is assumed given.

Returns: void

Description: Build knot sequence and sampling parameter for the given data to interpolate, curve type and the parameterization type desired.

3.2.48 BspCrvInterpPts (cbsp_int.c:142)

interpolation

least square approximation

```
CagdCrvStruct *BspCrvInterpPts(const CagdPtStruct *PtList,
                               int Order,
                               int CrvSize,
                               CagdParametrizationType ParamType,
                               CagdBType Periodic)
```

PtList: List of points to interpolate/least square approximate.

Order: Of interpolating/approximating curve.

CrvSize: Number of degrees of freedom (control points) of the interpolating/approximating curve.

ParamType: Type of parametrization.

Periodic: Constructed curve should be Periodic. Periodic necessitates uniform knot sequence in ParamType.

Returns: Constructed interpolating/approximating curve.

Description: Given a set of points, PtList, computes a Bspline curve of order Order that interpolates or least square approximates the set of points. The size of the control polygon of the resulting Bspline curve defaults to the number of points in PtList (if CrvSize = 0). However, this number is can smaller to yield a least square approximation. The created curve can be parametrized as specified by ParamType.
See also: BspCrvInterpolate, BspCrvInterpPts2, MvarBspCrvInterpVecs,

3.2.49 BspCrvInterpPts2 (cbsp_int.c:215)

interpolation

least square approximation

```
CagdCrvStruct *BspCrvInterpPts2(const CagdCtlPtStruct *PtList,
                                int Order,
                                int CrvSize,
                                CagdParametrizationType ParamType,
                                CagdBType Periodic,
                                CagdBType EndPtInterp)
```

PtList: List of points to interpolate/least square approximate.

Order: Of interpolating/approximating curve.

CrvSize: Number of degrees of freedom (control points) of the interpolating/approximating curve.

ParamType: Type of parametrization.

Periodic: Constructed curve should be Periodic. Periodic necessitates uniform knot sequence in ParamType.

EndPtInterp: TRUE to force Crv's end point interpolation. Has affect only if has open end-conditions.

Returns: Constructed interpolating/approximating curve.

Description: Given a set of points, PtList, computes a Bspline curve of order Order that interpolates or least square approximates the set of points. The size of the control polygon of the resulting Bspline curve defaults to the number of points in PtList (if CrvSize = 0). However, this number is can smaller to yield a least square approximation. The created curve can be parametrized as specified by ParamType.
See also: BspCrvInterpolate, BspCrvInterpPts,

3.2.50 BspCrvInterpPts3 (cbsp_int.c:280)

interpolation

least square approximation

```
CagdCrvStruct *BspCrvInterpPts3(const CagdCtlPtStruct *PtList,
                                const CagdRType *Params,
                                const CagdRType *KV,
                                int Length,
                                int Order,
                                CagdBType Periodic)
```

PtList: List of points to interpolate/least square approximate.

Params: At which to interpolate the points in PtList.

KV: Computed knot vector for the constructed curve.

Length: Number of degrees of freedom (control points) of the interpolating/approximating curve.

Order: Of interpolating/approximating curve.

Periodic: Constructed curve should be Periodic.

Returns: Constructed interpolating/approximating curve, NULL if singular.

Description: Given a set of points, PtList, and parameter values the curve should interpolate or approximate these points, Params, and the expected knot vector, KV, length Length and order Order of the B-spline curve, computes the B-spline curve's coefficients. All points in PtList are assumed of the same type. If Periodic, Order - 1 more constraints (and DOF's) are added so that the first Order - 1 points are the same as the last Order - 1 points. Same as BspCrvInterpolate but forces end points to be interpolated.

3.2.51 BspCrvInterpPtsError (cbsp_int.c:1397)

error estimation

interpolation

least square approximation

```
CagdRType BspCrvInterpPtsError(const CagdCrvStruct *Crv,
                               const CagdPtStruct *PtList,
                               CagdParametrizationType ParamType,
                               CagdBType Periodic)
```

Crv: Curve that was fitted to the data set.

PtList: The data set.

ParamType: Parameter values at with curve should interpolate PtList.

Periodic: Constructed curve should be Periodic. Periodic necessitates uniform knot sequence in ParamType.

Returns: Error measured in the L1 norm.

Description: Given a set of points, and a curve least square fitting them using the BspCrvInterpPts function, computes an error measure as a the maximal distance between the curve and points (L1 norm).

3.2.52 BspCrvInterpolate (cbsp_int.c:581)

interpolation

least square approximation

```
CagdCrvStruct *BspCrvInterpolate(const CagdCtlPtStruct *PtList,
                                 const CagdRType *Params,
                                 const CagdRType *KV,
                                 int Length,
                                 int Order,
                                 CagdBType Periodic)
```

PtList: List of points to interpolate/least square approximate.

Params: At which to interpolate the points in PtList.

KV: Computed knot vector for the constructed curve.

Length: Number of degrees of freedom (control points) of the interpolating/approximating curve.

Order: Of interpolating/approximating curve.

Periodic: Constructed curve should be Periodic.

Returns: Constructed interpolating/approximating curve, NULL if singular.

Description: Given a set of points, PtList, and parameter values the curve should interpolate* or approximate these points, Params, and the expected knot vector, KV, length Length and order Order of the Bspline curve, computes he Bspline* curve's coefficients. All points in PtList are assumed of the same type. If Periodic, Order - 1 more constraints (and DOF's) are added so that the first Order - 1 points are the same as the last Order - 1 points.

3.2.53 BspCrvIsC1DiscontAt (cbspeval.c:295)

```
CagdBType BspCrvIsC1DiscontAt(const CagdCrvStruct *Crv, CagdRType t)
```

Crv: Curves to examine for a C1 discontinuity.

t: Parameter value to examine at.

Returns: TRUE if Crv has a C1 discontinuity at parameter t, FALSE otherwise.

Description: Examines the control polygon at given parametric location for a real C1 discontinuity, in Euclidean space.
See also: BspSrfIsC1DiscontAt,

3.2.54 BspCrvKnotC0Discont (bsp_gen.c:370)

knot vectors
continuity
discontinuity

```
CagdBType BspCrvKnotC0Discont(const CagdCrvStruct *Crv, CagdRType *t)
```

Crv: To examine its potential discontinuity.

t: Where to put the parameter value (knot) that can be C0 discontinuous.

Returns: TRUE if found a C0 discontinuity, FALSE otherwise.

Description: Scans the given knot vector of the given curve for a potential C0 discontinuity. Looks for multiplicities in the knot sequence and then examine the mesh if indeed the mesh is discontinuous at that location. Assumes knot vectors has open end condition.
See also: BspCrvKnotC1Discont, BspCrvKnotC2Discont, BspKnotC0Discont, , BspCrvMeshC1Continuous, BspCrvKnotC1Discont,

3.2.55 BspCrvKnotC1Discont (bsp_gen.c:404)

knot vectors
continuity
discontinuity

```
CagdBType BspCrvKnotC1Discont(const CagdCrvStruct *Crv, CagdRType *t)
```

Crv: To examine its potential discontinuity.

t: Where to put the parameter value (knot) that can be C1 discontinuous.

Returns: TRUE if found a C1 discontinuity, FALSE otherwise.

Description: Scans the given knot vector of the given curve for a potential C1 discontinuity. Looks for multiplicities in the knot sequence and then examine the mesh if indeed the mesh is discontinuous at that location. Assumes knot vectors has open end condition.
See also: BspCrvKnotC0Discont, BspCrvKnotC2Discont, BspKnotC1Discont, , BspCrvMeshC1Continuous,

3.2.56 BspCrvKnotC2Discont (bsp_gen.c:438)

knot vectors
continuity
discontinuity

```
CagdBType BspCrvKnotC2Discont(const CagdCrvStruct *Crv, CagdRType *t)
```

Crv: To examine its potential discontinuity.

t: Where to put the parameter value (knot) that can be C1 discontinuous.

Returns: TRUE if found a C2 discontinuity, FALSE otherwise.

Description: Scans the given knot vector of the given curve for a potential C2 discontinuity. Looks for multiplicities in the knot sequence and then examine the mesh if indeed the mesh is discontinuous at that location. Assumes knot vectors has open end condition.
See also: BspCrvKnotC0Discont, BspCrvKnotC1Discont, BspKnotC1Discont, , BspCrvMeshC1Continuous,

3.2.57 BspCrvKnotInsert (bspboehm.c:60)

refinement

knot insertion

```
CagdCrvStruct *BspCrvKnotInsert(const CagdCrvStruct *Crv, CagdRType t)
```

Crv: To refine by adding a new knot with value equal to t. If Crv is a periodic curve, it is first unwrapped to a float end condition curve.

t: New knot to insert into Crv.

Returns: The refined curve.

Description: Returns a new curve refined at t (t is inserted as a new knot in Crv). If however the multiplicity of t in the current knot vector is equal (or greater!?) to the degree or t is not in the curve's parametric domain, no new knot is insert and NULL is returned instead. Control mesh is updated as follows (P is old ctl polygon, Q is new): Let Index be the last knot in old knot vector less than t and let j be j = Index - order + 1. Also let k be the curve order. Then,

```
Case 1: Q(i) = P(i), i <= j
```

$$\text{case 2: } Q(i) = \frac{t - t(i)}{t(i+k-1) - t(i)} P(i) + \frac{t(i+k-1) - t}{t(i+k-1) - t(i)} P(i-1),\ j<i<=Index$$

```
case 3: Q(i) = P(i-1), Index < i
```

Note: Altough works, this is not the optimal way to insert many knot! See also the BspKnotEvalAlpha set of routines.

For more see: "Recursive proof of Boehm's knot insertion technique", by Phillip J Barry Ronald N Goldman, CAD, Volume 20 number 4 1988, pp 181-182. Which also references the original 1980 paper by Boehm.

See also: BspCrvKnotInsertNSame, BspCrvKnotInsertNDiff, BspSrfKnotInsert, BspKnotEvalAlphaCoef,

3.2.58 BspCrvKnotInsertNDiff (cbsp_aux.c:303)

refinement

subdivision

```
CagdCrvStruct *BspCrvKnotInsertNDiff(const CagdCrvStruct *Crv,
                                     CagdBType Replace,
                                     CagdRType *t,
                                     int n)
```

Crv: To refine by insertion (upto) n knot of value t.

Replace: if TRUE, the n knots in t should replace the knot vector of size n of Crv. Sizes must match. If False, n new knots as defined by t will be introduced into Crv.

t: New knots to introduce/replace knot vector of Crv.

n: Size of t.

Returns: Refined Crv with n new knots.

Description: Inserts n knot with different values as defined by the vector t. If, however, Replace is TRUE, the knot are simply replacing the current knot vector.

3.2.59 BspCrvKnotInsertNSame (cbsp_aux.c:254)

refinement

subdivision

```
CagdCrvStruct *BspCrvKnotInsertNSame(const CagdCrvStruct *Crv,
                                     CagdRType t,
                                     int n)
```

Crv: To refine by insertion (upto) n knot of value t.

t: Parameter value of new knot to insert.

n: Maximum number of times t should be inserted.

Returns: Refined Crv with n knots of value t.

Description: Inserts n knot, all with the value t. In no case will the multiplicity of a knot be greater or equal to the curve order.

3.2.60 BspCrvMaxCoefParam (bsp_knot.c:1821)

extremum

```
CagdRType BspCrvMaxCoefParam(const CagdCrvStruct *Crv,
                             int Axis,
                             CagdRType *MaxVal)
```

Crv: To compute the parameter node value of the largest coefficient.

Axis: Which axis should we search for maximal coefficient? 1 for X, 2 for Y, etc.

MaxVal: The coefficient itself will be place herein.

Returns: The node parameter value of the detected maximal coefficient.

Description: Finds the parameter value with the largest coefficient of the curve using nodes values to estimate the coefficients' parameters.

3.2.61 BspCrvMeshC0Continuous (bsp_gen.c:472)

```
CagdBType BspCrvMeshC0Continuous(const CagdCrvStruct *Crv,
                                 int Idx,
                                 CagdRType Tol)
```

Crv: To examine its potential discontinuity.

Idx: Index of control point where to examine the discontinuity. if 0, curve is assumed closed and beginning/end is examined.

Tol: Tolerance allowed for distance deviation.

Returns: TRUE if continuous there, FALSE otherwise.

Description: Examine the control polygon of the given curve at index Idx for a real C0 discontinuity in the mesh. This index will typically be for a knot multiplicity potential discont.
See also: BspKnotC1Discont, BspCrvMeshC1Continuous,

3.2.62 BspCrvMeshC1Continuous (bsp_gen.c:509)

```
CagdBType BspCrvMeshC1Continuous(const CagdCrvStruct *Crv,
                                 int Idx,
                                 CagdRType *CosAngle)
```

Crv: To examine its potential discontinuity.

Idx: Index of control point where to examine the discontinuity. if 0, curve is assumed closed and beginning/end is examined.

CosAngle: If not NULL, updated with the cosine of the deviation angle.

Returns: TRUE if continuous there, FALSE otherwise.

Description: Examine the control polygon of the given curve at index Idx for a real C1 discontinuity in the mesh. This index will typically be for a knot multiplicity potential discont.
See also: BspKnotC1Discont, BspCrvMeshC0Continuous,

3.2.63 BspCrvMoebiusTransform (cbsp_aux.c:1183)

```
CagdCrvStruct *BspCrvMoebiusTransform(const CagdCrvStruct *CCrv, CagdRType c)
```

CCrv: Curve to apply the Moebius transformation to.

c: The scaling coefficient - c^n is the ratio between the first and last weight of the curve. If c == 0, the first and last weights are made equal.

Returns: The modified curve with the same shape but different speed.

Description: Apply the Moebius transformation to a rational Bspline curve. See "Moebius reparametrization of rational Bsplines", by Lee & Lucian, CAGD 8 (1991) pp 213-215.
See also: BzrCrvMoebiusTransform, BspSrfMoebiusTransform,

3.2.64 BspCrvNew (bsp_gen.c:133)

allocation

```
CagdCrvStruct *BspCrvNew(int Length, int Order, CagdPointType PType)
```

Length: Number of control points

Order: The order of the curve

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform Bspline curve.

Description: Allocates the memory required for a new Bspline curve.
See also: BzrCrvNew, BspPeriodicCrvNew, CagdCrvNew, CagdPeriodicCrvNew, TrimCrvNew,

3.2.65 BspCrvNormalMalloc (cbsp_aux.c:964)

normal

```
CagdVecStruct *BspCrvNormalMalloc(const CagdCrvStruct *Crv,
                                  CagdRType t,
                                  CagdBType Normalize)
```

Crv: Crv for which to compute a (unit) normal.

t: The parameter at which to compute the unit normal.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, length is a function of given parametrization.

Returns: A pointer to a vector holding the normal information. Always an E3 vector.

Description: Returns a (unit) vector, equal to the normal of Crv at parameter value t. Algorithm: returns the cross product of the curve tangent and binormal.

3.2.66 BspCrvNormalToData (cbsp_aux.c:928)

normal

```
CagdVecStruct *BspCrvNormalToData(const CagdCrvStruct *Crv,
                                  CagdRType t,
                                  CagdBType Normalize,
                                  CagdVecStruct *N)
```

Crv: Crv for which to compute a (unit) normal.

t: The parameter at which to compute the unit normal.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, length is a function of given parametrization.

N: A pointer to a vector holding the normal information.

Returns: A pointer to a vector holding the normal information. Always an E3 vector.

Description: Returns a (unit) vector, equal to the normal of Crv at parameter value t. Algorithm: returns the cross product of the curve tangent and binormal.

3.2.67 BspCrvOpenEnd (bsp_gen.c:283)

open end conditions

```
CagdCrvStruct *BspCrvOpenEnd(const CagdCrvStruct *Crv)
```

Crv: To convert to a new curve with open end conditions.

Returns: Same curve as Crv but with open end conditions.

Description: Returns a curve with open end conditions, similar to given curve. Open end curve is computed by extracting a subregion from Crv that is the entire curve's parametric domain, by inserting multiple knots at the domain's boundary.
See also: BspSrfOpenEnd,

3.2.68 BspCrvSubdivAtParam (cbsp_aux.c:162)

subdivision

refinement

```
CagdCrvStruct *BspCrvSubdivAtParam(const CagdCrvStruct *Crv, CagdRType t)
```

Crv: To subdivide at parametr value t.

t: Parameter value to subdivide Crv at.

Returns: A list of the two subdivided curves.

Description: Given a Bspline curve - subdivides it into two sub-curves at the given parametric value. Returns pointer to first curve in a list of two subdivided curves. The subdivision is achieved by inserting (order-1) knot at the given parameter value t and spliting the control polygon and knot vector at that location.

3.2.69 BspCrvSubdivCtlPoly (cbsp_aux.c:54)

```
void BspCrvSubdivCtlPoly(const CagdCrvStruct *Crv,
                         CagdRType **LPoints,
                         CagdRType **RPoints,
                         int LLength,
                         int RLength,
                         CagdRType t,
                         int Mult)
```

Crv: To subdivide at parameter value t.

LPoints, RPoints: Where the results are kept.

LLength, RLength: Lengths of respective vectors.

t: Parameter value to subdivide Crv at.

Mult: Current multiplicity of t in the knot sequence.

Returns: void

Description: Apply B-spline subdivision to the given curve Crv at parameter value t, and save the result in curves' LPoints/RPoints.
See also: BzrCrvSubdivCtlPoly, BspCrvSubdivAtParam,

3.2.70 BspCrvTangentToData (cbsp_aux.c:545)

tangent

```
CagdVecStruct *BspCrvTangentToData(const CagdCrvStruct *Crv,
                                   CagdRType t,
                                   CagdBType Normalize,
                                   CagdVecStruct *Tan)
```

Crv: Crv for which to compute a (unit) tangent.

t: The parameter at which to compute the unit tangent.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, returned is an unnormalized vector in the right direction of the tangent.

Tan: A pointer to a vector holding the tangent information.

Returns: A pointer to a vector holding the tangent information. An Ek, k >= 3 vector, depending on dimension k of Crv.

Description: Returns a (unit) vector, equal to the tangent to Crv at parameter value t. Algorithm: insert (order - 1) knots and return control polygon tangent. The unnormalized normal does not equal dC/dt in its magnitude, only in its direction.

3.2.71 BspCrvsSubdivAtAllDetectedLocations (cagd_aux.c:2074)

```
CagdCrvStruct *BspCrvsSubdivAtAllDetectedLocations(const CagdCrvStruct *Crvs,
                                                   CagdCrvTestingFuncType
                                                     CrvTestFunc)
```

Crvs: Curves to subdivide at all detected locations by CrvTestFunc.

CrvTestFunc: Curve testing function, like BspCrvKnotC0Discont.

Returns: Set of subdivided surfaces, or NULL if nothing was detected.

Description: Subdivides the given curve Crv at all locations CrvTesTFunc detects. Examples for CrvTesTFunc can be BspCrvKnotC0Discont or BspCrvKnotC1Discont.
See also: BspSrfsSubdivAtAllDetectedLocations, CagdCrvSubdivAtAllC1Discont, , BspCrvKnotC0Discont, BspCrvKnotC1Discont,

3.2.72 BspGenBasisFuncsAsCurves (bsp_gen.c:1391)

```
CagdCrvStruct *BspGenBasisFuncsAsCurves(int Order,
                                        int Length,
                                        const CagdRType *KV)
```

Order: Of space to create basis functions for.

Length: Number of control points in this space.

KV: Knot sequence of this space, of length (Order + Length).

Returns: Length curves representing the basis functions.

Description: Creates a list of curves representing the B-spline basis functions of the given space (order and knot sequence). All basis functions will be scaled to fit into the unit square [0, 1]^2. If the space is periodic, Length should reflect this in the input (i.e. length should be enlarged by order - 1).
See also: BspGenKnotsGeometryAsCurves,

3.2.73 BspGenKnotsGeometryAsCurves (bsp_gen.c:1478)

```
CagdCrvStruct *BspGenKnotsGeometryAsCurves(int Order,
                                           int Length,
                                           const CagdRType *KV,
                                           CagdRType SizeOfKnot)
```

Order: Of space to create the geometry of the knots for.

Length: Number of control points in this space.

KV: Knot sequence of this space, of length (Order + Length).

SizeOfKnot: The size of the plot knot. Knots are created as triangles

Returns: Length curves representing the Length knots.

Description: Creates a list of linear B-spline curves representing the B-spline knots in the given space (order and knot sequence). All knots will be scaled to fit just below the unit square [0, 1]^2. If the space is periodic, Length should reflect this in the input (i.e. length should be enlarged by order - 1).
See also: BspGenBasisFuncsAsCurves,

3.2.74 BspIsKnotDiscontUniform (bsp_knot.c:714)

knot vectors

end conditions

```
CagdEndConditionType BspIsKnotDiscontUniform(int Len,
                                             int Order,
                                             const CagdRType *KnotVector)
```

Len: Of control polygon/mesh of curve/surface that is using this knot vector.

Order: Of the curve/surface that is using this knot vector.

KnotVector: The knot vector to verify.

Returns: CAGD_END_COND_GENERAL if general knot vector, or CAGD_END_COND_OPEN/FLOAT/PERIODIC if knot vector is uniform with open/float/periodic end conditions.

Description: Tests the given knot vector for discontinuous uniformity and open/float end conditions. That is all interior knots are of multiplicity Order-1 and are uniformly spaced.

3.2.75 BspIsKnotUniform (bsp_knot.c:646)

knot vectors

end conditions

```
CagdEndConditionType BspIsKnotUniform(int Len,
                                      int Order,
                                      const CagdRType *KnotVector)
```

Len: Of control polygon/mesh of curve/surface that is using this knot vector.

Order: Of the curve/surface that is using this knot vector.

KnotVector: The knot vector to verify.

Returns: CAGD_END_COND_GENERAL if general knot vector, or CAGD_END_COND_OPEN/FLOAT/PERIODIC if knot vector is uniform with open/float/periodic end conditions.

Description: Tests the given knot vector for uniformity and open/float end conditions.

3.2.76 BspKnotAffineTrans (bsp_knot.c:864)

knot vectors

affine transformation

```
void BspKnotAffineTrans(CagdRType *KnotVector,
                        int Len,
                        CagdRType Translate,
                        CagdRType Scale)
```

KnotVector: To affinely transform.

Len: Of knot vector. This is not the length of the curve or surface using this knot vector.

Translate: Amount to translate the knot vector.

Scale: Amount to scale the knot vector.

Returns: void

Description: Applies an affine transformation to the given knot vector. Note affine transformation on the knot vector does not change the Bspline curve. Knot vector is translated by Translate amount and scaled by Scale as

```
KV[i] = (KV[i] - KV[0]) * Scale + (KV[0] + Translate).
```

All transformation as taken place in place.
See also: BspKnotScale, BspKnotAffineTrans2, BspKnotAffineTransOrder,

3.2.77 BspKnotAffineTrans2 (bsp_knot.c:908)

knot vectors

affine transformation

```
void BspKnotAffineTrans2(CagdRType *KnotVector,
                         int Len,
                         CagdRType MinVal,
                         CagdRType MaxVal)
```

KnotVector: To affinely transform.

Len: Of knot vector. This is not the length of the curve or surface using this knot vector.

MinVal, MaxVal: New parametric domain of knot vector.

Returns: void

Description: Applies an affine transformation to the given knot vector. Note affine transformation on the knot vector does not change the B-spline curve. Knot vector is translated and scaled so as to span the domain from MinVal top MaxVal. This works for open end condition curves only.

```
KV[i] = (KV[i] - KV[0]) * Scale + MinVal,
```

where Scale = (MaxVal - MinVal) / (KV[Len - 1] - KV[0]). All transformation as taken place in place.
See also: BspKnotScale, BspKnotAffineTrans, BspKnotAffineTransOrder2,

3.2.78 BspKnotAffineTransOrder (bsp_knot.c:955)

knot vectors

affine transformation

```
void BspKnotAffineTransOrder(CagdRType *KnotVector,
                             int Order,
                             int Len,
                             CagdRType Translate,
                             CagdRType Scale)
```

KnotVector: To affinely transform.

Order: Order of the space using this knot vector.

Len: Of knot vector. This is not the length of the curve or surface using this knot vector.

Translate: Amount to translate the knot vector.

Scale: Amount to scale the knot vector.

Returns: void

Description: Applies an affine transformation to the given knot vector. Note affine transformation on the knot vector does not change the Bspline curve. Knot vector is translated by Translate amount and scaled by Scale as

```
KV[i] = (KV[i] - KV[Order-1]) * Scale + (KV[Order-1] + Translate).
```

All transformation as taken place in place.
See also: BspKnotScale, BspKnotAffineTrans2,

3.2.79 BspKnotAffineTransOrder2 (bsp_knot.c:1001)

knot vectors

affine transformation

```
void BspKnotAffineTransOrder2(CagdRType *KnotVector,
                              int Order,
                              int Len,
                              CagdRType MinVal,
                              CagdRType MaxVal)
```

KnotVector: To affinely transform.

Order: Order of the space using this knot vector.

Len: Of knot vector. This is not the length of the curve or surface using this knot vector.

MinVal, MaxVal: New parametric domain of knot vector.

Returns: void

Description: Applies an affine transformation to the given knot vector. Note affine transformation on the knot vector does not change the Bspline curve. Knot vector is translated and scaled so as to span the domain from MinVal to MaxVal.

```
KV[i] = (KV[i] - KV[Order - 1]) * Scale + MinVal,
```

where Scale = (MaxVal - MinVal) / (KV[Len - Order] - KV[Order - 1]). All transformation as taken place in place.
See also: BspKnotScale, BspKnotAffineTrans2,

3.2.80 BspKnotAllC0Discont (bsp_knot.c:2566)

knot vectors

continuity

discontinuity

```
CagdRType *BspKnotAllC0Discont(const CagdRType *KnotVector,
                               int Order,
                               int Length,
                               int *n)
```

KnotVector: To test for potential C0 discontinuities.

Order: Of geometry that exploits KnotVector.

Length: Length of curve/surface using KnotVector. This is NOT the length of KnotVector which is equal to (Length + Order).

n: Length of returned vector - number of potential C0 discontinuities found.

Returns: Vector holding all parametr values with potential C0 discontinuities.

Description: Scans the given knot vector for all potential C0 discontinuity. Returns a vector holding the parameter values of the potential C0 discontinuities, NULL of none found. Sets n to length of returned vector. Assumes knot vector has open end condition. A knot vector with multiplicity of a knot of Order can be C0 discontinuous at that knot. However, this is only a necessary condition for C0 discontinuity in the geometry.

3.2.81 BspKnotAllC1Discont (bsp_knot.c:2632)

knot vectors

continuity

discontinuity

```
CagdRType *BspKnotAllC1Discont(const CagdRType *KnotVector,
                               int Order,
                               int Length,
                               int *n)
```

KnotVector: To test for potential C1 discontinuities.

Order: Of geometry that exploits KnotVector.

Length: Length of curve/surface using KnotVector. This is NOT the length of KnotVector which is equal to (Length + Order).

n: Length of returned vector - number of potential C1 discontinuities found, zero if none.

Returns: Vector holding all parameter values with potential C1 discontinuities, allocated dynamically, NULL if none.

Description: Scans the given knot vector for all potential C1 discontinuity. Returns a vector holding the parameter values of the potential C1 discontinuities, NULL of none found. Sets n to length of returned vector. Assumes knot vector has open end condition. A knot vector with multiplicity of a knot of (Order - 1) can be C1 discontinuous at that knot. However, this is only a necessary condition for C1 discontinuity in the geometry.

3.2.82 BspKnotAlphaLoopBlendNotPeriodic (cagdoslo.c:812)

```
void BspKnotAlphaLoopBlendNotPeriodic(const BspKnotAlphaCoeffStruct *A,
                                      int IMin,
                                      int IMax,
                                      const CagdRType *OrigPts,
                                      CagdRType *RefPts)
```

A: Alpha matrix to use.

IMin, IMax: Domain of refined controls points to blend.

OrigPts: original coefficients.

RefPts: Refined (returned) coefficients.

Returns: void

Description: Blend the input control points using the given Alpha matrix. A non periodic case is assumed.
See also: BspKnotEvalAlphaCoef, BspKnotAlphaLoopBlendPeriodic, BspKnotAlphaLoopBlendStep,

3.2.83 BspKnotAlphaLoopBlendPeriodic (cagdoslo.c:891)

```
void BspKnotAlphaLoopBlendPeriodic(const BspKnotAlphaCoeffStruct *A,
                                   int IMin,
                                   int IMax,
                                   const CagdRType *OrigPts,
                                   int OrigLen,
                                   CagdRType *RefPts)
```

A: Alpha matrix to use.

IMin, IMax: Domain of refined controls points to blend.

OrigPts: original coefficients.

OrigLen: Original length of OrigPts.

RefPts: Refined (returned) coefficients.

Returns: void

Description: Blend the input control points using the given Alpha matrix. A non periodic case is assumed.
See also: BspKnotEvalAlphaCoef, BspKnotAlphaLoopBlendNotPeriodic, , BspKnotAlphaLoopBlendStep,

3.2.84 BspKnotAlphaLoopBlendStep (cagdoslo.c:999)

```
void BspKnotAlphaLoopBlendStep(const BspKnotAlphaCoeffStruct *A,
                               int IMin,
                               int IMax,
                               const CagdRType *OrigPts,
                               int OrigPtsStep,
                               int OrigLen,
                               CagdRType *RefPts,
                               int RefPtsStep)
```

A: Alpha matrix to use.

IMin, IMax: Domain of refined controls points to blend.

OrigPts: original coefficients.

OrigPtsStep: Steps between adjacent coefficients, in multi-dim. arrays.

OrigLen: Original length of OrigPts.

RefPts: Refined (returned) coefficients.

RefPtsStep: Steps between adjacent refined coefficients, in multi-dim. arrays.

Returns: void

Description: Blend the input control points using the given Alpha matrix. A non periodic case is assumed.
See also: BspKnotEvalAlphaCoef, BspKnotAlphaLoopBlendNotPeriodic, , BspKnotAlphaLoopBlendPeriodic,

3.2.85 BspKnotAverage (bsp_knot.c:1470)

knot vectors

node values

```
CagdRType *BspKnotAverage(const CagdRType *KnotVector, int Len, int Ave)
```

KnotVector: To average out.

Len: Length of KnotVector. This is not the length of the curve or surface using this knot vector.

Ave: How many knots to average each time.

Returns: The averaged knot vector of length (Len - Ave + 1).

Description: Creates a new knot vector from the given KnotVector that averages Ave consequetive knots. Resulting vector will have (Len - Ave + 1) elements.
See also: BspKnotNodes,

3.2.86 BspKnotC0Discont (bsp_knot.c:2249)

knot vectors

continuity

discontinuity

```
CagdBType BspKnotC0Discont(const CagdRType *KnotVector,
                           int Order,
                           int Length,
                           CagdRType *t)
```

KnotVector: To test for potential C0 dicontinuities.

Order: Of geometry that exploits KnotVector.

Length: Length of curve/surface using KnotVector. This is NOT the length of KnotVector which is equal to (Length + Order).

t: Where to put the parameter value (knot) that can be C0 discontinuous.

Returns: TRUE if found a potential C0 discontinuity, FALSE otherwise.

Description: Scans the given knot vector to a potential C0 discontinuity. Returns TRUE if found one and set t to its parameter value. Assumes knot vector has open end condition. A knot vector with multiplicity of a knot of (Order) can be C0 discontinuous at that knot. However, this is only a necessary condition for C0 discontinuity in the geometry.
See also: BspSrfKnotC1Discont, BspKnotC1Discont, BspKnotC2Discont, , BspKnotAllC1Discont,

3.2.87 BspKnotC1Discont (bsp_knot.c:2329)

knot vectors
continuity
discontinuity

```
CagdBType BspKnotC1Discont(const CagdRType *KnotVector,
                           int Order,
                           int Length,
                           CagdRType *t)
```

KnotVector: To test for potential C1 discontinuities.

Order: Of geometry that exploits KnotVector.

Length: Length of curve/surface using KnotVector. This is NOT the length of KnotVector which is equal to (Length + Order).

t: Where to put the parameter value (knot) that can be C1 discontinuous.

Returns: TRUE if found a potential C1 discontinuity, FALSE otherwise.

Description: Scans the given knot vector to a potential C1 discontinuity. Returns TRUE if found one and set t to its parameter value. Assumes knot vector has open end condition. A knot vector with multiplicity of a knot of (Order - 1) can be C1 discontinuous at that knot. However, this is only a necessary condition for C1 discontinuity in the geometry.
See also: BspSrfKnotC1Discont, BspKnotC0Discont, BspKnotC2Discont, BspKnotAllC1Discont,

3.2.88 BspKnotC2Discont (bsp_knot.c:2408)

knot vectors
continuity
discontinuity

```
CagdBType BspKnotC2Discont(const CagdRType *KnotVector,
                           int Order,
                           int Length,
                           CagdRType *t)
```

KnotVector: To test for potential C1 discontinuities.

Order: Of geometry that exploits KnotVector.

Length: Length of curve/surface using KnotVector. This is NOT the length of KnotVector which is equal to (Length + Order).

t: Where to put the parameter value (knot) that can be C1 discontinuous.

Returns: TRUE if found a potential C2 discontinuity, FALSE otherwise.

Description: Scans the given knot vector to a potential C2 discontinuity. Returns TRUE if found one and set t to its parameter value. Assumes knot vector has open end condition. A knot vector with multiplicity of a knot of (Order - 1) can be C2 discontinuous at that knot. However, this is only a necessary condition for C2 discontinuity in the geometry.
See also: BspSrfKnotC1Discont, BspKnotC0Discont, BspKnotC1Discont, BspKnotAllC1Discont,

3.2.89 BspKnotCnDiscont (bsp_knot.c:2489)

knot vectors
continuity
discontinuity

```
CagdBType BspKnotCnDiscont(const CagdRType *KnotVector,
                           int Order,
                           int Length,
                           int n,
                           CagdRType *t)
```

KnotVector: To test for potential C-n discontinuities.

Order: Of geometry that exploits KnotVector.

Length: Length of curve/surface using KnotVector. This is NOT the length of KnotVector which is equal to (Length + Order).

n: The degree of the C-n discontinuity.

t: Where to put the parameter value (knot) that can be C-n discontinuous.

Returns: TRUE if found a potential C-n discontinuity, FALSE otherwise.

Description: Scans the given knot vector to a potential C-n discontinuity, for a prescribed n. Returns TRUE if found one and set t to its parameter value. Assumes knot vector has open end condition. A knot vector with multiplicity of a knot of (Order - n) can be C-n discontinuous at that knot. However, this is only a necessary condition for C-n discontinuity in the geometry.
See also: BspSrfKnotC1Discont, BspKnotC0Discont, BspKnotC2Discont,

3.2.90 BspKnotContinuityMergeTwo (bsp_knot.c:1335)

knot vectors
compatibility
refinement

```
CagdRType *BspKnotContinuityMergeTwo(const CagdRType *KnotVector1,
                                     int Len1,
                                     int Order1,
                                     const CagdRType *KnotVector2,
                                     int Len2,
                                     int Order2,
                                     int ResOrder,
                                     int *NewLen)
```

KnotVector1: First knot vector.

Len1: Length of KnotVector1. This is not the length of the curve or surface using this knot vector.

Order1: Order of first knot vector's geometry.

KnotVector2: Second knot vector.

Len2: Length of KnotVector2. This is not the length of the curve or surface using this knot vector.

Order2: Order of second knot vector's geometry.

ResOrder: Expected order of geometry that will use the merged knot vector.

NewLen: To save the size of the knot vector that contains the merged knot vectors.

Returns: The merged knot verctor (KnotVector1 U KnotVector2).

Description: Merges two knot vector KnotVector1 and KnotVector2 of length Len1 and Len2 respectively into one, from geometries of orders Order1 and Order2. Merged knot vector is for order ResOrder so that the resulting curve can represent the discontinuities in both geometries. Assumes both knot vectors are open end spanning the same domain.

3.2.91 BspKnotCopy (bsp_knot.c:1035)

allocation
knot vectors

```
CagdRType *BspKnotCopy(CagdRType *DstKV, const CagdRType *SrcKV, int Len)
```

DstKV: Destination address or NULL for a whole new copy.

SrcKV: Knot vector to duplicate

Len: Length of knot vector. This is not the length of the curve or surface using this knot vector.

Returns: The duplicated (destination) knot vector.

Description: Creates an identical copy of a given knot vector KnotVector of length Len.

3.2.92 BspKnotCopyAlphaCoef (cagdoslo.c:548)

alpha matrix
refinement

```
BspKnotAlphaCoeffStruct *BspKnotCopyAlphaCoef(const BspKnotAlphaCoeffStruct *A)
```

A: Alpha matrix to copy.

Returns: Copied matrix.

Description: Copies the BspKnotAlphaCoeffStruct data structure.
See also: BspKnotEvalAlphaCoef, BspKnotEvalAlphaCoefMerge, BspKnotFreeAlphaCoef, , BspCrvKnotInsert, BspSrfKnotInsert,

3.2.93 BspKnotDegreeRaisedKV (bsp_knot.c:1114)

```
CagdRType *BspKnotDegreeRaisedKV(const CagdRType *KV,
                                 int Len,
                                 int Order,
                                 int NewOrder,
                                 int *NewLen)
```

KV: Current knot vector of freeform.

Len: Length of the freeform - number of control points.

Order: Order of the freeform.

NewOrder: New order of the freeform.

NewLen: New length of (dynamically) allocated knot vector.

Returns: A new knot vector, allocated dynamically, that would fit this same freeform if degree raised.

Description: Computes a knot vector for a freeform that will fit the freeform if it was degree raised to NewOrder.

3.2.94 BspKnotDiscontUniformOpen (bsp_knot.c:598)

knot vectors

open end conditions

d conditions

```
CagdRType *BspKnotDiscontUniformOpen(int Len, int Order, CagdRType *KnotVector)
```

Len: Of control polygon/mesh of curve/surface that is to use this knot vector.

Order: Of the curve/surface that is to use this knot vector.

KnotVector: If new knot vector is to be saved here, otherwise a new space is allocated.

Returns: The created uniform open knot vector, either newly allocated or given in Knotvector and just initialized.

Description: Returns a discontinuous uniform open knot vector for Len Control points and order Order. The actual length of the KV is Len + Order. The generated sequence is of the form "0 0 0 0 1 1 1 2 2 2 ... n n n n" and Hence Len + Order must equal 2 * Order + 3 * x * (Order - 1). If KnotVector is NULL it is being allocated dynamically.

3.2.95 BspKnotDoubleKnots (bsp_knot.c:1429)

knot vectors

node values

```
CagdRType *BspKnotDoubleKnots(const CagdRType *KnotVector, int *Len, int Order)
```

KnotVector: To average out.

Len: Length of KnotVector. This is not the length of the curve or surface using this knot vector. Len is updated in the end to the length of the returned vector.

Order: Order of freeform geometry using this knot sequence.

Returns: The averaged knot vector of length (Len - Ave + 1).

Description: Creates a new knot vector from the given KnotVector that includes knot values in the middle of any two adjacent knots that are different in value.

See also: BspKnotNodes,

3.2.96 BspKnotEvalAlphaCoef (cagdoslo.c:88)

alpha matrix

refinement

```
BspKnotAlphaCoeffStruct *BspKnotEvalAlphaCoef(int k,
                                              CagdRType *KVT,
                                              int LengthKVT,
                                              CagdRType *KVt,
                                              int LengthKVt,
                                              int Periodic)
```

k: Order of geometry.

KVT: Original knot vector.

LengthKVT: Length of original control polygon with KVT knot vector.

KVt: Refined knot vector. Must contain all knots of KVT.

LengthKVt: Length of refined control polygon with KVt knot vector.

Periodic: If the refinement is for a periodic entity.

Returns: A matrix to multiply the coefficients of the geometry using KVT, in order to get the coefficients under the space defined using KVt that represent the same geometry.

Description: Computes the values of the alpha coefficients, Ai,k(j) of order k:

$$C(t) = \sum_{i=0}^{n} Pi\ Bi,k(t) = \sum_{i=0}^{n} Pi \sum_{j=0}^{m} Ai,k(j)\ Nj,k(t) = \sum_{j=0}^{m} \left(\sum_{i=0}^{n} Pi\ Ai,k(j) \right) Nj,k(t)$$

Let T be the original knot vector and let t be the refined one, i.e. T is a subset of t. The Ai,k(j) are computed from the following recursive definition:

$$Ai,1(j) = \begin{cases} 1, & T(i) <= t(i) < T(i+1) \\ 0, & \text{otherwise.} \end{cases}$$

$$Ai,k(j) = \frac{T(j+k-1) - T(i)}{T(i+k-1) - T(i)} Ai,k-1(j) + \frac{T(i+k) - T(j+k-1)}{T(i+k) - T(i+1)} Ai+1,k-1(j)$$

LengthKVT + k is the length of KVT and similarly LengthKVt + k is the length of KVt. In other words, LengthKVT and LengthKVt are the control points len...
The output matrix has LengthKVT rows and LengthKVt columns (#cols > #rows) ColIndex/Length hold LengthKVt pairs of first non zero scalar and length of non zero values in that column, so not all LengthKVT scalars are blended.

See also: BspKnotFreeAlphaCoef, BspKnotEvalAlphaCoefMerge, BspCrvKnotInsert, , BspSrfKnotInsert, BspKnotAlphaLoopBlendPeriodic, , BspKnotAlphaLoopBlendNotPeriodic, BspKnotAlphaLoopBlendStep,

3.2.97 BspKnotEvalAlphaCoefIdentity (cagdoslo.c:432)

alpha matrix

refinement

```
static BspKnotAlphaCoeffStruct *BspKnotEvalAlphaCoefIdentity(int k,
                                                             CagdRType *KVT,
                                                             int LengthKVT,
                                                             int Periodic)
```

k: Order of geometry.

KVT: knot vector. Both in and out.

LengthKVT: Length of control polygon with the KVT knot vector.

Periodic: If the refinement is for a periodic entity.

Returns: A matrix to multiply the coefficients of the geometry using KVT, in order to get the coefficients under the space defined using KVt that represent the same geometry.

Description: Computes an identity (diagonal) Alpha matrix. In cases it is used, for efficiency reasons...
See also: BspKnotFreeAlphaCoef, BspKnotEvalAlphaCoefMerge, BspCrvKnotInsert, , BspSrfKnotInsert, BspKnotAlphaLoopBlendPeriodic, , BspKnotAlphaLoopBlendNotPeriodic, BspKnotAlphaLoopBlendStep, , BspKnotEvalAlphaCoef,

3.2.98 BspKnotEvalAlphaCoefMerge (cagdoslo.c:708)

alpha matrix

refinement

```
BspKnotAlphaCoeffStruct *BspKnotEvalAlphaCoefMerge(int k,
                                                   CagdRType *KVT,
                                                   int LengthKVT,
                                                   CagdRType *NewKV,
                                                   int LengthNewKV,
                                                   int Periodic)
```

k: Order of geometry.

KVT: Original knot vector.

LengthKVT: Length of original knot vector.

NewKV: A sequence of new knots to introduce into KVT.

LengthNewKV: Length of new knot sequence.

Periodic: If the refinement is for a periodic entity.

Returns: A matrix to multiply the coefficients of the geometry using KVT, in order to get the coefficients under the space defined using KVt that represent the same geometry.

Description: Same as EvalAlphaCoef but the new knot set NewKV is merged with KVT to form the new knot vector KVt.

See also: BspKnotFreeAlphaCoef, BspKnotEvalAlphaCoef, BspCrvKnotInsert, , BspSrfKnotInsert,

3.2.99 BspKnotFindMult (bsp_knot.c:2053)

knot vectors

```
int BspKnotFindMult(const CagdRType *KnotVector,
                    int Order,
                    int Len,
                    CagdRType t)
```

KnotVector: To test multiplicity of knot value t at.

Order: Of geometry that exploits KnotVector.

Len: Length of curve/surface using KnotVector. This is NOT the length of KnotVector which is equal to (Len + Order).

t: The knot to verify the multiplicity of.

Returns: Multiplicity of t in KnotVector.

Description: Returns the multiplicity of knot t in knot vector KnotVector, zero if none.

3.2.100 BspKnotFirstIndexG (bsp_knot.c:396)

knot vectors

```
int BspKnotFirstIndexG(const CagdRType *KnotVector,
                       int Len,
                       CagdRType t,
                       CagdRType Tol)
```

KnotVector: To search for a knot with the G relation to t.

Len: Of knot vector. This is not the length of the curve/surface using this KnotVector.

t: The parameter value to search for the G relation.

Tol: Tolerance of comparisons.

Returns: Index of first knot in KnotVector that is greater than t or Len if t is greater than last knot in KnotVector.

Description: Returns the index of the first knot which is greater than t in the given knot vector KnotVector of length Len. IRIT_APX_EQ_EPS is used for equality. Parameter t is assumed to be in the parametric domain for the knot vector.

3.2.101 BspKnotFreeAlphaCoef (cagdoslo.c:647)

alpha matrix
refinement

```
void BspKnotFreeAlphaCoef(BspKnotAlphaCoeffStruct *A)
```

A: Alpha matrix to free.

Returns: void

Description: Frees the BspKnotAlphaCoeffStruct data structure.
See also: BspKnotEvalAlphaCoef, BspKnotEvalAlphaCoefMerge, BspKnotCopyAlphaCoef, , BspCrvKnotInsert, BspSrfKnotInsert,

3.2.102 BspKnotHasBezierKV (bsp_knot.c:84)

knot vectors
conversion

```
CagdBType BspKnotHasBezierKV(const CagdRType *KnotVector, int Len, int Order)
```

KnotVector: To check for open end and no interior knots conditions.

Len: Of control mesh of this knot vector.

Order: Of curve/surface the exploits this knot vector.

Returns: TRUE if has open end conditions and no interior knots, FALSE otherwise.

Description: Returns TRUE iff the given knot vector of length (Len + Order) has no interior knots and it has an open end conditions.

3.2.103 BspKnotHasOpenEC (bsp_knot.c:186)

knot vectors
open end conditions

```
CagdBType BspKnotHasOpenEC(const CagdRType *KnotVector, int Len, int Order)
```

KnotVector: To check for open end condition.

Len: Of control mesh of this knot vector.

Order: Of curve/surface the exploits this knot vector.

Returns: TRUE if KV has open end conditions.

Description: Returns TRUE iff the given knot vector of length (Len + Order) has open end conditions.

3.2.104 BspKnotInsertMult (bsp_knot.c:1997)

knot vectors
knot insertion
refinement

```
CagdRType *BspKnotInsertMult(const CagdRType *KnotVector,
                             int Order,
                             int *Len,
                             CagdRType t,
                             int Mult)
```

KnotVector: To insert new knot t in.

Order: Of geometry that exploits KnotVector.

Len: Length of curve/surface using KnotVector. This is NOT the length of KnotVector which is equal to (Length + Order).

t: The new knot t to insert.

Mult: The multiplicity that this knot should have in resulting knot vector.

Returns: A new knot vector derived from KnotVector that has a mltiplicity of exacly Mult at the knot t.

Description: Inserts Mult knots with value t into the knot vector KnotVector. Attempt is made to make sure t in knot vector domain. If a knot equal to t (up to IRIT_APX_EQ) already exists with multiplicity i only (Mult - i) knot are being inserted into the new knot vector. Len is updated to the resulting knot vector. It is possible to DELETE a knot using this routine by specifying multiplicity less then current multiplicity! This function only constructs a refined knot vector and does not compute the actual refined coefficients.

3.2.105 BspKnotInsertOne (bsp_knot.c:1957)

knot vectors
knot insertion
refinement

```
CagdRType *BspKnotInsertOne(const CagdRType *KnotVector,
                            int Order,
                            int Len,
                            CagdRType t)
```

KnotVector: To insert new knot t in.

Order: Of geometry that exploits KnotVector.

Len: Length of curve/surface using KnotVector. This is NOT the length of KnotVector which is equal to (Length + Order).

t: The new knot t to insert.

Returns: A new knot vector larger by one than KnotVector that contains t.

Description: Creates a new vector with t inserted as a new knot. Attempt is made to make sure t is in the knot vector domain. No test is made for the current multiplicity of knot t in KnotVector. This function only constructs a refined knot vector and does not compute the actual refined coefficients.

3.2.106 BspKnotLastIndexL (bsp_knot.c:345)

knot vectors

```
int BspKnotLastIndexL(const CagdRType *KnotVector,
                      int Len,
                      CagdRType t,
                      CagdRType Tol)
```

KnotVector: To search for a knot with the L relation to t.

Len: Of knot vector. This is not the length of the curve/surface using this KnotVector.

t: The parameter value to search for the L relation.

Tol: Tolerance of comparisons.

Returns: Index of last knot in KnotVector that is less than t or -1 if t is below the first knot.

Description: Returns the index of the last knot which is less t in the given knot vector KnotVector of length Len. IRIT_APX_EQ_EPS is used for equality. Parameter t is assumed to be in the parametric domain for the knot vector.

3.2.107 BspKnotLastIndexLE (bsp_knot.c:294)

knot vectors

```
int BspKnotLastIndexLE(const CagdRType *KnotVector,
                       int Len,
                       CagdRType t,
                       CagdRType Tol)
```

KnotVector: To search for a knot with the LE relation to t.

Len: Of knot vector. This is not the length of the curve/surface using this KnotVector.

t: The parameter value to search for the LE relation.

Tol: Tolerance of comparisons.

Returns: Index of last knot in KnotVector that is LE t, or -1 if t is below the first knot.

Description: Returns the index of the last knot which is less than or equal to t in the given knot vector KnotVector of length Len. IRIT_APX_EQ_UEPS is used in equality. Parameter t is assumed to be in the parametric domain for the knot vector.

3.2.108 BspKnotMakeRobustKV (bsp_knot.c:2794)

knot vectors

```
CagdBType BspKnotMakeRobustKV(CagdRType *KV, int Len)
```

KV: Knot vector to make robust, in place.

Len: Length of knot vector KV.

Returns: TRUE if the knot sequence has been modified.

Description: Given a knot vector, make sure adjacent knots that are close "enough" are actually identical. Important for robustness of subdiv/refinement algs.
See also: BspKnotVerifyKVValidity,

3.2.109 BspKnotMergeTwo (bsp_knot.c:1247)

knot vectors
compatibility
refinement

```
CagdRType *BspKnotMergeTwo(const CagdRType *KnotVector1,
                           int Len1,
                           const CagdRType *KnotVector2,
                           int Len2,
                           int Mult,
                           int *NewLen)
```

KnotVector1: First knot vector.

Len1: Length of KnotVector1. This is not the length of the curve or surface using this knot vector.

KnotVector2: Second knot vector.

Len2: Length of KnotVector2. This is not the length of the curve or surface using this knot vector.

Mult: Maximum multiplicity to allow in merged knot vector.

NewLen: To save the size of the knot vector that contains the merged knot vectors.

Returns: The merged knot verctor (KnotVector1 U KnotVector2).

Description: Merges two knot vector KnotVector1 and KnotVector2 of length Len1 and Len2 respectively into one. If Mult is not zero then knot multiplicity is tested not to be larger than Mult value. NewLen is set to new KnotVector length.

3.2.110 BspKnotMinDmnBzrIdx (bsp_knot.c:2154)

```
int BspKnotMinDmnBzrIdx(const CagdRType *KnotVector,
                        int Len,
                        int Order,
                        CagdRType MinDmn,
                        CagdRType Eps)
```

KnotVector: To derive a specific Bezier domain for.

Len: Length of the KnotVector.

Order: The order of the basis functions using this KnotVector.

MinDmn: The minimum of the Bezier domain we seek its index.

Eps: Tolerance of approximated equal knot.

Returns: The computed index, or -1 if error.

Description: Compute the index (starting from zero) of the Bezier polynomial domain that starts at MinDmn.
See also: BspKnotsMultiplicityVector,

3.2.111 BspKnotMultiplicity (bsp_knot.c:440)

```
int BspKnotMultiplicity(const CagdRType *KnotVector, int Len, int Idx)
```

KnotVector: To compute the multiplicity of knot index Idx.

Len: Of knot vector. This is not the length of the curve/surface using this KnotVector.

Idx: Index of knot to compute its multiplicity.

Returns: Multiplicity of the knot. At least one.

Description: Computes the multiplicity of given knot index Idx in KnotVector,

3.2.112 BspKnotNode (bsp_knot.c:1599)

knot vectors

node value

```
CagdRType BspKnotNode(const CagdRType *KnotVector, int i, int Order)
```

KnotVector: To average out as nodes. Can be NULL in which case a Bezier case is assumed.

i: Index of control points. First index is zero.

Order: Of curve or surface that exploits this knot vector.

Returns: The node value as a parameter value estimation for the i'th control point.

Description: Computes a parameter value estimation (as a node value) for the i'th control point. The parameter value (the node) is defined as (k = Order - 1 or degree):

```
          i+k
           -
           \
           / KnotVector(j)
           -
         j=i+1
N(i) = -----------------
                k
```

See also: BspKnotAverage, BspKnotNodes, BspKnotPeriodicNodes,

3.2.113 BspKnotNodes (bsp_knot.c:1533)

knot vectors

node values

```
CagdRType *BspKnotNodes(const CagdRType *KnotVector, int Len, int Order)
```

KnotVector: To average out as nodes.

Len: Length of KnotVector. This is not the length of the curve or surface using this knot vector.

Order: Of curve or surface that exploits this knot vector.

Returns: The nodes computed for the given knot vector.

Description: Creates a new vector with the given KnotVector Node values. The given knot vector is assumed to have open end conditions. The nodes are the approximated parametric value associated with the each control point. Therefore for a knot vector of length Len and order Order there are Len - Order control points and therefore nodes. Nodes are defined as (k = Order - 1 or degree):

```
          i+k
           -                              First Node N(i = 0)
           \
           / KnotVector(j)                Last Node  N(i = Len - k - 2)
           -
         j=i+1
N(i) = -----------------
                k
```

See also: BspKnotAverage, BspKnotPeriodicNodes,

3.2.114 BspKnotParamInDomain (bsp_knot.c:253)

parametric domain

knot vectors

```
CagdBType BspKnotParamInDomain(const CagdRType *KnotVector,
                               int Len,
                               int Order,
                               CagdBType Periodic,
                               CagdRType t)
```

KnotVector: To verify t is indeed in.

Len: Length of curve/surface using KnotVector. This is NOT the length of KnotVector which is equal to (Length + Order).

Order: Order of curve/surface using KnotVector.

Periodic: TRUE if this KnotVector is periodic.

t: Parametric value to verify.

Returns: TRUE, if t is contained in the parametric domain, FALSE otherwise.

Description: Returns TRUE iff t is in the parametric domain as define by the knot vector KnotVector, its length Len, and the order Order.

3.2.115 BspKnotParamValues (bsp_knot.c:2695)

piecewise linear approximation

knot vectors

```
CagdRType *BspKnotParamValues(CagdRType PMin,
                              CagdRType PMax,
                              int NumSamples,
                              CagdRType *C1Disconts,
                              int NumC1Disconts)
```

PMin: Minimum of parametric domain.

PMax: Maximum of parametric domain.

NumSamples: To allocate for the vector of samples.

C1Disconts: A vector of potential C1 discontinuities in the (PMin, PMax) domain. This vector is freed by this routine, if it is not NULL.

NumC1Disconts: Length of C1Discont. if zero then C1Discont == NULL.

Returns: A vector of the suggested set of sampling locations.

Description: Routine to determine where to sample along the provided paramtric domain, given the C1 discontinuities along it. Returns a vector of length NumSamples. If C1Disconts != NULL (NumC1Disconts > 0), C1Discont is being freed.

3.2.116 BspKnotPeriodicNodes (bsp_knot.c:1650)

knot vectors

node values

```
CagdRType *BspKnotPeriodicNodes(const CagdRType *KnotVector,
                                int Len,
                                int Order)
```

KnotVector: To average out as nodes.

Len: Length of periodic KnotVector. This is not the length of the curve or surface using this knot vector.

Order: Of curve or surface that exploits this knot vector.

Returns: The nodes computed for the given knot vector.

Description: Creates a new vector with the given KnotVector Node values. The given knot vector is assumed to have periodic end conditions. The nodes are the approximated parametric value associated with the each control point. Therefore for a knot vector of length Len and order Order there are Len - Order control points and therefore nodes. Nodes are defined as (k = Order - 1 or degree):

$$N(i) = \frac{\sum_{j=i+1}^{i+k} KnotVector(j)}{k}$$

First Node N(i = 0)

Last Node N(i = Len - k - 2)

See also: BspKnotAverage, BspKnotNodes,

3.2.117 BspKnotPrepEquallySpaced (cagdoslo.c:756)

refinement

```
CagdRType *BspKnotPrepEquallySpaced(const CagdRType *KV,
                                    int KVLen,
                                    int KVOrder,
                                    int *n,
                                    CagdRType Tmin,
                                    CagdRType Tmax)
```

KV: The knot vector to refine.

KVLen: The total length of the KV (Control polygon Length + Order).

KVOrder: The order of the space.

n: Number of knots to introduce. Returns the actual introduced which can be a bit less if a knot already found in proximity.

Tmin: Minimum domain to introduce knots.

Tmax: Maximum domain to introduce knots.

Returns: A vector of n knots uniformly spaced between TMin and TMax.

Description: Prepares a refinement vector for the given knot vector domain with n inserted knots equally spaced.

3.2.118 BspKnotReverse (bsp_knot.c:1072)

knot vectors

reverse

```
CagdRType *BspKnotReverse(const CagdRType *KnotVector, int Len)
```

KnotVector: Knot vector to be reversed.

Len: Length of knot vector. This is not the length of the curve or surface using this knot vector.

Returns: The reversed knot vector.

Description: Reverse a knot vector of length Len. Reversing of knot vector keeps the knots monotonically non-decreasing as well as the parametric domain. Only the spaces between the knots are being flipped. For example the knot vector:

```
[0 0 0 0 1 2 2 6 6 6 6]
```

is reversed to be:

```
[0 0 0 0 4 4 5 6 6 6 6]
```

3.2.119 BspKnotScale (bsp_knot.c:788)

knot vectors

affine transformation

```
void BspKnotScale(CagdRType *KnotVector, int Len, CagdRType Scale)
```

KnotVector: To affinely transform.

Len: Of knot vector. This is not the length of the curve or surface using this knot vector.

Scale: Amount to scale the knot vector.

Returns: void

Description: Applies a scale transformation to the given knot vector. Note scale transformation on the knot vector does not change the Bspline curve. Knot vector is scaled by Scale as KV[i] = KV[i] * Scale. Scaling is taken place in place.

See also: BspKnotTranslate, BspKnotAffineTrans, BspKnotAffineTrans2,

3.2.120 BspKnotSortKVMonotone (bsp_knot.c:3073)

```
void BspKnotSortKVMonotone(CagdRType *KV, int Len)
```

KV: To verify and ensure it is monotone, in place.

Len: Length of KV.

Returns: void

Description: Verify and sort the given knot sequence, in place, so it is monotone.

See also: BspKnotVerifyKVValidity,

3.2.121 BspKnotSubtrTwo (bsp_knot.c:1176)

knot vectors
compatibility
refinement

```
CagdRType *BspKnotSubtrTwo(const CagdRType *KnotVector1,
                           int Len1,
                           const CagdRType *KnotVector2,
                           int Len2,
                           int *NewLen)
```

KnotVector1: First knot vector.

Len1: Length of KnotVector1. This is not the length of the curve or surface using this knot vector.

KnotVector2: Second knot vector.

Len2: Length of KnotVector2. This is not the length of the curve or surface using this knot vector.

NewLen: To save the size of the knot vector that contains the computed subset of KnotVector1 / KnotVector2.

Returns: The subset of knot in KnotVector1 that are not in KnotVector2 (KnotVector1 / KnotVector2).

Description: Returns a knot vector that contains all the knots in KnotVector1 that are not in KnotVector2. NewLen is set to new KnotVector length.

3.2.122 BspKnotTranslate (bsp_knot.c:824)

knot vectors
affine transformation

```
void BspKnotTranslate(CagdRType *KnotVector, int Len, CagdRType Trans)
```

KnotVector: To translate.

Len: Of knot vector. This is not the length of the curve or surface using this knot vector.

Trans: Amount to translate the knot vector.

Returns: void

Description: Applies a translation transformation to the given knot vector. Note translation transformation on the knot vector does not change the Bspline curve. Knot vector is translated by Trans as KV[i] = KV[i] + Trans. Translation is taken place in place.

See also: BspKnotScale, BspKnotAffineTrans, BspKnotAffineTrans2,

3.2.123 BspKnotUniformFloat (bsp_knot.c:513)

knot vectors
floating end conditions
end conditions

```
CagdRType *BspKnotUniformFloat(int Len, int Order, CagdRType *KnotVector)
```

Len: Of control polygon/mesh of curve/surface that is to use this knot vector.

Order: Of the curve/surface that is to use this knot vector.

KnotVector: If new knot vector is to be saved here, otherwise a new space is allocated.

Returns: The created uniform floating knot vector, either newly allocated or given in Knotvector and just initialized.

Description: Returns a uniform floating knot vector for Len Control points and order Order. The actual length of the KV is Len + Order. If KnotVector is NULL it is being allocated dynamically.

3.2.124 BspKnotUniformOpen (bsp_knot.c:552)

knot vectors
open end conditions
end conditions

```
CagdRType *BspKnotUniformOpen(int Len, int Order, CagdRType *KnotVector)
```

Len: Of control polygon/mesh of curve/surface that is to use this knot vector.

Order: Of the curve/surface that is to use this knot vector.

KnotVector: If new knot vector is to be saved here, otherwise a new space is allocated.

Returns: The created uniform open knot vector, either newly allocated or given in Knotvector and just initialized.

Description: Returns a uniform open knot vector for Len Control points and order Order. The actual length of the KV is Len + Order. If KnotVector is NULL it is being allocated dynamically.

3.2.125 BspKnotUniformPeriodic (bsp_knot.c:474)

knot vectors

periodic end conditions

end conditions

```
CagdRType *BspKnotUniformPeriodic(int Len, int Order, CagdRType *KnotVector)
```

Len: Of control polygon/mesh of curve/surface that is to use this knot vector.

Order: Of the curve/surface that is to use this knot vector.

KnotVector: If new knot vector is to be saved here, otherwise a new space is allocated.

Returns: The created uniform periodic knot vector, either newly allocated or given in Knotvector and just initialized.

Description: Returns a uniform periodic knot vector for Len Control points and order Order. The actual length of the KV is Len + Order + Order - 1. If KnotVector is NULL it is being allocated dynamically.

3.2.126 BspKnotVectorsSame (bsp_knot.c:2840)

```
CagdBType BspKnotVectorsSame(const CagdRType *KV1,
                             const CagdRType *KV2,
                             int Len,
                             CagdRType Eps)
```

KV1, KV2: The two knot vectors to compare.

Len: Length of knot vectors.

Eps: Tolerance of equality.

Returns: TRUE if knot vectors are the same, FALSE otherwise.

Description: Compare the two knot vectors for similarity.
See also: CagdCtlPointsSame, CagdCrvsSame, CagdSrfsSame,

3.2.127 BspKnotVerifyKVValidity (bsp_knot.c:2917)

```
int BspKnotVerifyKVValidity(CagdRType *KV, int Order, int Len, CagdRType Tol)
```

KV: To validate its knots in, place.

Order: Of the knot sequence.

Len: Such that Len + Order equals the number of knots in KV.

Tol: Tolerance to consider two knots the same.

Returns: 1 if valid, -1 if was validated in place and is valid now, 0 if failed to validate.

Description: Verify that the given knot sequence is a valid one up to tolerance Tol. That is no more than order knots are similar up to Tol and that the knot sequence is monotone. Updated in place, if can.
See also: BspKnotSortKVMonotone,

3.2.128 BspKnotVerifyPeriodicKV (bsp_knot.c:2872)

```
void BspKnotVerifyPeriodicKV(CagdRType *KV, int Order, int Len)
```

KV: To update its knots outside the entity's domain.

Order: Of the entity.

Len: Such that Len + Order equals the number of knots in KV.

Returns: void

Description: Update the two ends (knots outside the curve's domain) to match the same spacing as the inner knots on the other end... Updates KV in place.

3.2.129 BspKnotsGetIntervals (bsp_knot.c:2202)

knot vectors

```
void BspKnotsGetIntervals(const CagdRType *KV,
                          int KVLen,
                          CagdRType **KnotIntervals,
                          int *IntervalCount)
```

KV: To derive its knot intervals.

KVLen: Length of the KnotVector.

KnotIntervals: (Out parameter) Vector of the knot intervals (unique values found in KV). Allocated dynamically & returned.

IntervalCount: (Out parameter) The resulting size of KnotIntervals - number of actual intervals found.

Returns: void

Description: Computes the knot intervals of the given knot sequence (i.e. a sequence of unique knot values). For example: (0, 0, 0, 1, 2, 2, 3, 3, 3) will be converted to KnotIntervals: (0, 1, 2, 3) and KnotCount of 4. The knot sequence is returned in KnotIntervals (allocated inside the function) and its length in IntervalCount.
See also: BspKnotsMultiplicityVector,

3.2.130 BspKnotsMultiplicityVector (bsp_knot.c:2104)

knot vectors

```
int BspKnotsMultiplicityVector(const CagdRType *KnotVector,
                               int Len,
                               CagdRType *KnotValues,
                               int *KnotMultiplicities,
                               CagdRType Eps)
```

KnotVector: To derive its multiplicity/value vectors.

Len: Length of the KnotVector.

KnotValues: Vector of the unique values found in KnotVector.

KnotMultiplicities: Multiplicities of unique values found in Knotvector. Can be NULL to ignore.

Eps: Tolerance of approximated equal knot.

Returns: Size of vectors KnotValues/KnotMultiplicities.

Description: Computes the multiplicity/value vectors of the given knot sequence. For example: (0, 0, 0, 1, 2, 2, 3, 3, 3) will be converted to: KnotValues of (0, 1, 2, 3) and KnotMultiplicities of (3, 1, 2, 3). KnotValues and KnotMultiplicities are assumed big enough vectors to hold the result.
See also: BspKnotsGetIntervals,

3.2.131 BspMakeReparamCurve (cbsp_int.c:1719)

```
CagdCrvStruct *BspMakeReparamCurve(const CagdPtStruct *PtsList,
                                   int Order,
                                   int DegOfFreedom)
```

PtsList: List of points on the reparametrization curve.

Order: of reparametrization curve.

DegOfFreedom: of reparametrization curve (== number of coefficients).

Returns: Result of reparametrization curve, computed using list squares fit.

Description: Computes a reparametrization scalar Bspline curve, y(x), so that each at each point in PtsList, the curve parameter value of X is evaluated into Y.
See also: BspCrvInterpolate,

3.2.132 BspMeshC1PtsCollinear (bsp_gen.c:629)

```
CagdBType BspMeshC1PtsCollinear(const CagdPType Pt0,
                                const CagdPType Pt1,
                                const CagdPType Pt2,
                                CagdRType *LenRatio)
```

Pt0, Pt1, Pt2: Three points to check for collinearity along possible C^1 discont.

LenRatio: To verify (if !IRIT_INFNTY) or compute (if IRIT_INFNTY).

Returns: TRUE if collinear, FALSE otherwise.

Description: Examines the collinearity of the given three points and also verify (if First FALSE) or compute the lengths ratios.
See also: BspSrfMeshC1Continuous, TrivTVMeshC1Continuous, BspSrfMeshC1Continuous,

3.2.133 BspPeriodicCrvNew (bsp_gen.c:181)

allocation

```
CagdCrvStruct *BspPeriodicCrvNew(int Length,
                                 int Order,
                                 CagdBType Periodic,
                                 CagdPointType PType)
```

Length: Number of control points

Order: The order of the curve

Periodic: Is this curve periodic?

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform Bspline curve. If Periodic is FALSE, this function is identical to BspCrvNew.

Description: Allocates the memory required for a new, possibly periodic, Bspline curve.
See also: BzrCrvNew, BspCrvNew, CagdCrvNew, CagdPeriodicCrvNew, TrimCrvNew,

3.2.134 BspPeriodicSrfNew (bsp_gen.c:92)

allocation

```
CagdSrfStruct *BspPeriodicSrfNew(int ULength,
                                 int VLength,
                                 int UOrder,
                                 int VOrder,
                                 CagdBType UPeriodic,
                                 CagdBType VPeriodic,
                                 CagdPointType PType)
```

ULength: Number of control points in the U direction.

VLength: Number of control points in the V direction.

UOrder: The order of the surface in the U direction.

VOrder: The order of the surface in the V direction.

UPeriodic: Is this surface periodic in the U direction?

VPeriodic: Is this surface periodic in the V direction?

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform Bspline surface. If both UPeriodic and VPeriodic are FALSE, this function is identical to BspSrfNew.

Description: Allocates the memory required for a new, possibly periodic, Bspline surface.
See also: BspSrfNew, BzrSrfNew, CagdSrfNew, CagdPeriodicSrfNew, TrimSrfNew,

3.2.135 BspPtSamplesToKV (cbsp_int.c:517)

```
CagdRType *BspPtSamplesToKV(const CagdRType *PtsSamples,
                            int NumPts,
                            int CrvOrder,
                            int CrvLength)
```

PtsSamples: The parameter values of the data.

NumPts: Number of parameters in PtsSamples.

CrvOrder: Order of curve that will employ the constructed knot vector.

CrvLength: Length of curve that will employ constructed knot vector.

Returns: Constructed knot vector.

Description: Given NumPts parameter values (in PtsSamples), construct a knot vector for a curve of order CrvOrder and CrvLength control points to fit to this data set.
See also: BspKnotAverage,

3.2.136 BspReparameterizeCrv (cbsp_aux.c:1734)

```
void BspReparameterizeCrv(CagdCrvStruct *Crv,
                          CagdParametrizationType ParamType)
```

Crv: The curve to update its paraametrization.

ParamType: The desired parametrization type: uniform, chord len., etc.

Returns: void

Description: Reparameterize a curve to follow a desired parametrization.
See also: BspCrvInterpBuildKVs, BspReparameterizeSrf,

3.2.137 BspReparameterizeSrf (sbsp_aux.c:1994)

```
void BspReparameterizeSrf(CagdSrfStruct *Srf,
                          CagdSrfDirType Dir,
                          CagdParametrizationType ParamType)
```

Srf: The surface to update its paraametrization.

Dir: Parametric direction to reparameterize.

ParamType: The desired parametrization type: uniform, chord len., etc.

Returns: void

Description: Reparameterize a surface to follow a desired parametrization.
See also: BspCrvInterpBuildKVs, BspReparameterizeCrv,

3.2.138 BspSrf2Curves (bsp2poly.c:835)

curves

isoparametric curves

```
CagdCrvStruct *BspSrf2Curves(const CagdSrfStruct *Srf,
                             int NumOfIsocurves[2])
```

Srf: To extract isoparametric curves from.

NumOfIsocurves: In each (U or V) direction.

Returns: List of extracted isoparametric curves. These curves inherit the order and continuity of the original Srf. NULL is returned in case of an error.

Description: Routine to extract from a B-spline surface NumOfIsoline isocurve list in each param. direction. Iso parametric curves are sampled equally spaced in parametric space. NULL is returned in case of an error, otherwise list of CagdCrvStruct.
See also: BspSrf22Polylines, BzrSrf2PCurves, SymbSrf2Curves,

3.2.139 BspSrf2PolygonSetErrFunc (bsp2poly.c:105)

```
CagdSrfErrorFuncType BspSrf2PolygonSetErrFunc(CagdSrfErrorFuncType Func,
                                              void *Data)
```

Func: New function to use, NULL to disable.

Data: Optional reference to data to transfer to the function.

Returns: Old value of function.

Description: Sets the surface approximation error function. The error function will return a negative value if this patch must be purged or otherwise a non negative error measure.
See also: BspSrf2Polygons,

3.2.140 BspSrf2Polygons (bsp2poly.c:146)

polygonization

surface approximation

```
IPPolygonStruct *BspSrf2Polygons(const CagdSrfStruct *Srf,
                                 const CagdSrf2PlsInfoStrct *TessInfo)
```

Srf: To approximate into triangles.

TessInfo: Auxiliary tessellation information.

Returns: A list of polygons with optional normal and/or UV parametric information. NULL is returned in case of an error.

Description: Routine to convert a single B-spline surface to set of triangles approximating it. FineNess is a fineness control on result and the larger is more triangles may result. A value of 10 is a good start value. NULL is returned in case of an error, otherwise list of IPPolygonStruct. This routine looks for C1 discontinuities in the surface and splits it into C1 continuous patches to invoke BspC1Srf2Polygons to gen. polygons.
See also: BspSrf2PolygonSetErrFunc, BzrSrf2Polygons, IritSurface2Polygons, , IritTrimSrf2Polygons, CagdSrf2Polygons, TrimSrf2Polygons, , BspC1Srf2Polygons, CagdSrf2Polygons,

3.2.141 BspSrf2PolygonsN (bsp2poly.c:252)

polygonization

surface approximation

```
IPPolygonStruct *BspSrf2PolygonsN(const CagdSrfStruct *Srf,
                                  int Nu,
                                  int Nv,
                                  CagdBType ComputeNormals,
                                  CagdBType FourPerFlat,
                                  CagdBType ComputeUV)
```

Srf: To approximate into triangles.

Nu, Nv: The number of uniform samples in U and V of surface.

ComputeNormals: If TRUE, normal information is also computed.

FourPerFlat: If TRUE, four triangles are created per flat surface. If FALSE, only 2 triangles are created.

ComputeUV: If TRUE, UV values are stored and returned as well.

Returns: A list of polygons with optional normal and/or UV parametric information. NULL is returned in case of an error.

Description: Routine to convert a single B-spline surface to set of triangles approximating it. FineNess is a fineness control on result and the larger is more triangles may result. A value of 10 is a good start value. NULL is returned in case of an error, otherwise list of IPPolygonStruct. This routine looks for C1 discontinuities in the surface and splits it into C1 continuous patches to invoke BspC1Srf2Polygons to gen. polygons.
See also: BspSrf2PolygonSetErrFunc, BzrSrf2Polygons, IritSurface2Polygons, , IritTrimSrf2Polygons, CagdSrf2Polygons, TrimSrf2Polygons, , BspC1Srf2Polygons, CagdSrf2Polygons, BspSrf2Polygons, , BzrSrf2PolygonsN, CagdSrf2PolygonsN,

3.2.142 BspSrf2PolygonsSamplesNuNv (bsp2poly.c:312)

```
CagdBType BspSrf2PolygonsSamplesNuNv(const CagdSrfStruct *Srf,
                                     int Nu,
                                     int Nv,
                                     CagdBType ComputeNormals,
                                     CagdBType ComputeUV,
                                     CagdRType **PtWeights,
                                     CagdPtStruct **PtMesh,
                                     CagdVecStruct **PtNrml,
                                     CagdUVStruct **UVMesh)
```

Srf: To sample in a grid.

Nu, Nv: The number of uniform samples in U and V of surface.

ComputeNormals: If TRUE, normal information is also computed.

ComputeUV: If TRUE, UV values are stored and returned as well.

PtWeights: Weights of the evaluations, if rational, to detect poles. NULL if surface nor rational.

PtMesh: Evaluted positions of grid of samples.

PtNrml: Evaluted normals of grid of samples or NULL if none.

UVMesh: Evaluted UV vals of grid of samples or NULL if none.

Returns: FALSE is returned in case of an error, TRUE otherwise.

Description: Routine to uniformly sample a single Bspline srf as a grid. Nu and Nv fix the grid's sizes. FALSE is returned in case of an error, TRUE otherwise.
See also: BspSrf2Polygons, IritSurface2Polygons, IritTrimSrf2Polygons, , CagdSrf2Polygons, TrimSrf2Polygons, BzrSrf2PolygonsSamples, , BspC1Srf2PolygonsSamples,

3.2.143 BspSrf2Polylines (bsp2poly.c:682)

polylines

isoparametric curves

```
CagdPolylineStruct *BspSrf2Polylines(const CagdSrfStruct *Srf,
                                     int NumOfIsocurves[2],
                                     int SamplesPerCurve)
```

Srf: Srf to extract isoparametric curves from.

NumOfIsocurves: To extarct from Srf in each (U or V) direction.

SamplesPerCurve: Fineness control on piecewise linear curve approximation.

Returns: List of polygons representing a piecewise linear approximation of the extracted isoparamteric curves or NULL is case of an error.

Description: Routine to convert a single Bspline surface to NumOfIsolines polylines in each parametric direction with SamplesPerCurve in each isoparametric curve. Polyline are always E3 of CagdPolylineStruct type. Iso parametric curves are sampled equally spaced in parametric space. NULL is returned in case of an error, otherwise list of CagdPolylineStruct. Attempt is made to extract isolines along C1 discontinuities first.
See also: BspCrv2Polyline, BzrSrf2Polylines, IritSurface2Polylines, , IritTrimSrf2Polylines, SymbSrf2Polylines, TrimSrf2Polylines, CagdSrf2Polylines,

3.2.144 BspSrfC1DiscontCrvs (sbspeval.c:468)

```
CagdCrvStruct *BspSrfC1DiscontCrvs(const CagdSrfStruct *Srf)
```

Srf: To extract its C1 discontinuity curves.

Returns: The C1 discontinuities as a list of isoaprametric curves.

Description: Extracts the C1 discontinuity curves from the given Bspline surface. This routine detects potential discontinuities in the control mesh by seeking knots of Order-1 multiplicity. Potential discontinuities that materialize as real (by examining the mesh itself along that line) are extracted as isoparametric curves.
See also: BspSrfCrvFromSrf, BspKnotAllC1Discont, BspSrfIsC1DiscontAt, , BspSrfHasC1Discont,

3.2.145 BspSrfCrvFromMesh (sbspeval.c:388)

isoparametric curves

curve from mesh

```
CagdCrvStruct *BspSrfCrvFromMesh(const CagdSrfStruct *Srf,
                                 int Index,
                                 CagdSrfDirType Dir)
```

Srf: To extract a curve from.

Index: Index along the mesh of Srf to extract the curve from.

Dir: Direction of extracted curve. Either U or V.

Returns: A curve from Srf. This curve inherit the order and continuity of surface Srf in direction Dir. However, thiscurve is not on surface Srf, in general.

Description: Extracts a curve from the mesh of a tensor product Bspline surface Srf in direction Dir at index Index.

See also: CagdCrvFromSrf, BzrSrfCrvFromSrf, BspSrfCrvFromSrf, , CagdCrvFromMesh, BzrSrfCrvFromMesh,

3.2.146 BspSrfCrvFromSrf (sbspeval.c:295)

isoparametric curves

curve from surface

```
CagdCrvStruct *BspSrfCrvFromSrf(const CagdSrfStruct *Srf,
                                CagdRType t,
                                CagdSrfDirType dir)
```

Srf: To extract an isoparametric curve from.

t: Parameter value of extracted isoparametric curve.

dir: Direction of the isocurve on the surface. Either U or V.

Returns: An isoparametric curve of Srf. This curve inherits the order and continuity of surface Srf in direction Dir.

Description: Extracts an isoparametric curve out of the given tensor product B-spline surface in direction Dir at the parameter value of t. Operations should prefer the CONST_U_DIR, in which the extraction is somewhat faster if that is possible.

See also: CagdCrvFromSrf, BzrSrfCrvFromSrf, CagdCrvFromMesh, BzrSrfCrvFromMesh, , BspSrfCrvFromMesh,

3.2.147 BspSrfDegreeRaise (sbsp_aux.c:620)

degree raising

```
CagdSrfStruct *BspSrfDegreeRaise(const CagdSrfStruct *Srf, CagdSrfDirType Dir)
```

Srf: To raise it degree by one.

Dir: Direction of degree raising. Either U or V.

Returns: A surface with one degree higher in direction Dir, representing the same geometry as Srf.

Description: Returns a new Bspline surface, identical to the original but with one degree higher, in the requested direction Dir.

See also: CagdSrfDegreeRaise, BzrSrfDegreeRaise, TrimSrfDegreeRaise, BspSrfDegreeRaiseN,

3.2.148 BspSrfDegreeRaiseN (sbsp_aux.c:789)

degree raising

```
CagdSrfStruct *BspSrfDegreeRaiseN(const CagdSrfStruct *Srf,
                                  int NewUOrder,
                                  int NewVOrder)
```

Srf: To raise its degree.

NewUOrder: New U order of Srf.

NewVOrder: New V order of Srf.

Returns: A surface with higher degrees as prescribed by NewUOrder/NewVOrder.

Description: Returns a new Bspline surface, identical to the original but with higher degrees, as prescribed by NewUOrder, NewVOrder.

See also: CagdSrfDegreeRaise, BzrSrfDegreeRaise, TrimSrfDegreeRaise, BspSrfDegreeRaise, BzrSrfDegreeRaiseN, CagdSrfDegreeRaiseN,

3.2.149 BspSrfDerive (sbsp_aux.c:863)

derivatives

partial derivatives

```
CagdSrfStruct *BspSrfDerive(const CagdSrfStruct *Srf,
                            CagdSrfDirType Dir,
                            CagdBType DeriveScalar)
```

Srf: To differentiate.

Dir: Direction of differentiation. Either U or V.

DeriveScalar: TRUE to differentiate each channel independently. FALSE to treat geometry as a rational differentiation.

Returns: Differentiated surface.

Description: Returns a new surface equal to the given surface, differentiated once in the direction Dir. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one then:

```
Q(i) = (k - 1) * (P(i+1) - P(i)) / (Kv(i + k) - Kv(i + 1)), i = 0 to k-2.
```

This is applied to all rows/cols of the surface.
See also: CagdSrfDerive, BzrSrfDerive, SymbSrfDeriveRational, , BspSrfDeriveScalar,

3.2.150 BspSrfDeriveScalar (sbsp_aux.c:985)

derivatives

```
CagdSrfStruct *BspSrfDeriveScalar(const CagdSrfStruct *Srf, CagdSrfDirType Dir)
```

Srf: To differentiate.

Dir: Direction of differentiation. Either U or V.

Returns: Differentiated curve.

Description: Returns a new surface equal to the given surface, differentiated once in the direction Dir. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one then:

```
Q(i) = (k - 1) * (P(i+1) - P(i)), i = 0 to k-2.
```

This is applied to all rows/cols of the surface. For a Euclidean surface this is the same as BspSrfDerive but for a rational surface the returned surface is not the vector field but simply the derivatives of all the surface's coefficients, including the weights.
See also: BzrSrfDerive, CagdSrfDerive, SymbSrfDeriveRational, , BspSrfDerive, BzrSrfDeriveScalar, CagdSrfDeriveScalar,

3.2.151 BspSrfDomain (bsp_gen.c:251)

domain

parametric domain

```
void BspSrfDomain(const CagdSrfStruct *Srf,
                  CagdRType *UMin,
                  CagdRType *UMax,
                  CagdRType *VMin,
                  CagdRType *VMax)
```

Srf: To get its parametric domain.

UMin: Where to put the minimal U domain's boundary.

UMax: Where to put the maximal U domain's boundary.

VMin: Where to put the minimal V domain's boundary.

VMax: Where to put the maximal V domain's boundary.

Returns: void

Description: Returns the parametric domain of a Bspline surface.
See also: CagdSrfDomain, TrimSrfDomain,

3.2.152 BspSrfEvalAtParamMalloc (sbspeval.c:260)

evaluation

B-splines

```
CagdRType *BspSrfEvalAtParamMalloc(const CagdSrfStruct *Srf,
                                   CagdRType u,
                                   CagdRType v,
                                   void **Cache)
```

Srf: Surface to evaluate at the given (u, v) location.

u, v: Location where to evaluate the surface.

Cache: Optional cache for faster repeated evaluations of same U values.

Returns: A vector holding all the coefficients of all components of curve Crv's point type. If for example the curve's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1). This vector is allocated dynamically.

Description: Evaluates the given tensor product B-spline surface at a given point, by extracting an isoparametric curve along u from the surface and evaluating the curve at parameter v.

```
          u -->
  +----------------------+
  |P0                Pi-1|
V |Pi               P2i-1|  Parametric space orientation - control mesh.
| |                      |
v |Pn-i               Pn-1|
  +----------------------+
```

See also: CagdSrfEvalMalloc, BzrSrfEvalAtParamToData, BspSrfEvalAtParamToData, , BspSrfEvalAtParamToData, TrimSrfEvalToData,

3.2.153 BspSrfEvalAtParamToData (sbspeval.c:44)

evaluation

B-splines

```
void BspSrfEvalAtParamToData(const CagdSrfStruct *Srf,
                             CagdRType u,
                             CagdRType v,
                             CagdRType *R)
```

Srf: Surface to evaluate at the given (u, v) location.

u, v: Location where to evaluate the surface.

R: Where evaluated point should be saved.

Returns: void

Description: Evaluates the given tensor product B-spline surface at a given point, by extracting the basis functions in U/V and blending them with the 2D matrix of the relevant control points.

```
          u -->
  +----------------------+
  |P0                Pi-1|
V |Pi               P2i-1|  Parametric space orientation - control mesh.
| |                      |
v |Pn-i               Pn-1|
  +----------------------+
```

See also: CagdSrfEvalToData, BzrSrfEvalAtParamToData, BspSrfEvalAtParamToData, , BspSrfEvalAtParamMalloc, TrimSrfEvalToData,

3.2.154 BspSrfEvalAtParamToDataOld (sbspeval.c:148)

evaluation

B-splines

```
void BspSrfEvalAtParamToDataOld(const CagdSrfStruct *Srf,
                                CagdRType u,
                                CagdRType v,
                                CagdRType *R)
```

Srf: Surface to evaluate at the given (u, v) location.

u, v: Location where to evaluate the surface.

R: Where evaluated point should be saved.

Returns: void

Description: Evaluates the given tensor product B-spline surface at a given point, by extracting an isoparametric curve along u from the surface and evaluating the curve at parameter v.

```
           u -->
   +----------------------+
   |P0                Pi-1|
V  |Pi               P2i-1|  Parametric space orientation - control mesh.
|  |                      |
v  |Pn-i               Pn-1|
   +----------------------+
```

See also: CagdSrfEvalToData, BzrSrfEvalAtParamToData, BspSrfEvalAtParamToData, , BspSrfEvalAtParamMalloc, TrimSrfEvalToData,

3.2.155 BspSrfExtension (bsp_gen.c:1211)

```
CagdSrfStruct *BspSrfExtension(const CagdSrfStruct *OrigSrf,
                               const CagdBType *ExtDirs,
                               CagdRType EpsilonU,
                               CagdRType EpsilonV,
                               CagdBType RemoveExtraKnots)
```

OrigSrf: The surface to be extended.

ExtDirs: A vector of four boolean values to set the extension directions. The convention is MinU, MinV, MaxU, MaxV. if NULL, all four directions are extended.

EpsilonU: The length of the extension in the u direction.

EpsilonV: The length of the extension in the v direction.

RemoveExtraKnots: If FALSE, the resulting surface will not have minimal multiplicity at the first internal knot on the extension side. This is boolean controls all extensions that were performed, one decision for all of them.

Returns: The new extended surface.

Description: Extension of a B-spline surface, in any (or more than one) of the four optional directions of the 2D domain. The domain is extended, such that the trace coincides with the original trace over the original domain. Assumes open end conditions (in both knot vectors u and v). OrigSrf can have "ExtntSclU*" and/or "ExtntSclV*" real attrs. to scale the extension in Euclidean space (for same Epsilon parametric extension. If ExtntSclU/V != 1, the surface might also undergo refinements.

See also: BspCrvExtensionOneSide, BspCrvExtraKnotRmv,

3.2.156 BspSrfFitLstSqr (sbsp_int.c:367)

```
CagdSrfStruct *BspSrfFitLstSqr(const CagdSrfStruct *Srf,
                               int UOrder,
                               int VOrder,
                               int USize,
                               int VSize,
                               CagdParametrizationType ParamType,
                               CagdRType *Err)
```

Srf: Surface to fit a new surface to.

UOrder: Of the to be created surface.

VOrder: Of the to be created surface.

USize: U size of the to be created surface.

VSize: V size of the to be created surface.

ParamType: Type of parametrization.

Err: The maximum error is updated into here

Returns: Fitted surface.

Description: Fits a surface to the give surface by sampling points on Srf and fitting a surface of orders U/Order and U/VSize control points. Error is measured by the difference between the original and the fitted surface, as maximum error norm.
See also: BspSrfInterpPts,

3.2.157 BspSrfHasBezierKVs (bsp_knot.c:60)

conversion

```
CagdBType BspSrfHasBezierKVs(const CagdSrfStruct *Srf)
```

Srf: To check for KVs that mimics Bezier polynomial surface.

Returns: TRUE if same as Bezier surface, FALSE otherwise.

Description: Returns TRUE iff the given surface has no interior knot open end KVs.

3.2.158 BspSrfHasC1Discont (sbspeval.c:541)

```
CagdBType BspSrfHasC1Discont(const CagdSrfStruct *Srf, int E3C1Discont)
```

Srf: To examine for C^1 discontinuity curves.

E3C1Discont: If TRUE examine if real Euclidean C1 discont.

Returns: True if Srf has C^1 discontinuities, false otherwise. curves.

Description: Examines if the given Bspline surface has C^1 discontinuities. This routine detects potential discontinuities in the control mesh by seeking knots of Order-1 multiplicity. if E3C1Discont is TRUE, only parametric discontinuities that materialize as real (by examining the mesh itself along that line) are reported as true.
See also: BspSrfC1DiscontCrvs, BspKnotAllC1Discont, BspSrfIsC1DiscontAt,

3.2.159 BspSrfHasOpenEC (bsp_knot.c:128)

open end conditions

```
CagdBType BspSrfHasOpenEC(const CagdSrfStruct *Srf)
```

Srf: To check for open end conditions.

Returns: TRUE, if surface has open end conditions, FALSE otherwise.

Description: Returns TRUE iff the given B-spline surface has open end conditions.
See also: BspCrvHasOpenEC,

3.2.160 BspSrfHasOpenECDir (bsp_knot.c:153)

open end conditions

```
CagdBType BspSrfHasOpenECDir(const CagdSrfStruct *Srf, CagdSrfDirType Dir)
```

Srf: To check for open end conditions.

Dir: Either the U or the V parametric direction.

Returns: TRUE, if surface has open end conditions, FALSE otherwise.

Description: Returns TRUE iff the given B-spline surface has open end conditions in the specified direction.

3.2.161 BspSrfIntegrate (sbsp_aux.c:1013)

integrals

```
CagdSrfStruct *BspSrfIntegrate(const CagdSrfStruct *Srf, CagdSrfDirType Dir)
```

Srf: Surface to integrate.

Dir: Direction of integration. Either U or V.

Returns: Integrated surface.

Description: Returns a new Bspline surface, equal to the integral of the given Bspline srf. The given Bspline surface should be nonrational.

See also: BzrSrfIntegrate, BspCrvIntegrate, CagdSrfIntegrate,

3.2.162 BspSrfInterpPts (sbsp_int.c:106)

interpolation

least square approximation

```
CagdSrfStruct *BspSrfInterpPts(const CagdPtStruct **PtList,
                               int UOrder,
                               int VOrder,
                               int SrfUSize,
                               int SrfVSize,
                               CagdParametrizationType ParamType)
```

PtList: A NULL terminating array of linked list of points.

UOrder: Of the to be created surface.

VOrder: Of the to be created surface.

SrfUSize: U size of the to be created surface. Must be at least as large as the array PtList.

SrfVSize: V size of the to be created surface. Must be at least as large as the length of each list in PtList.

ParamType: Type of parametrization.

Returns: Constructed interpolating/approximating surface.

Description: Given a set of points, PtList, computes a Bspline surface of order UOrder by VOrder that interpolates or least square approximates the given set of points. PtList is a NULL terminated array of linked lists of CagdPtStruct structs. All linked lists in PtList must have the same length. U direction of surface is associated with array, V with the linked lists. The size of the control mesh of the resulting Bspline surface defaults to the number of points in PtList (if SrfUSize = SrfVSize = 0). However, either numbers can smaller to yield a least square approximation of the gievn data set. The created surface can be parametrized as specified by ParamType.

3.2.163 BspSrfInterpScatPts (sbsp_int.c:467)

interpolation

least square approximation

```
CagdSrfStruct *BspSrfInterpScatPts(const CagdCtlPtStruct *PtList,
                                   int UOrder,
                                   int VOrder,
                                   int USize,
                                   int VSize,
                                   CagdRType *UKV,
                                   CagdRType *VKV)
```

PtList: A NULL terminating array of linked list of points.

UOrder: Of the to be created surface.

VOrder: Of the to be created surface.

USize: U size of the to be created surface.

VSize: V size of the to be created surface.

UKV: Expected knot vector in U direction, NULL for uniform open.

VKV: Expected knot vector in V direction, NULL for uniform open.

Returns: Constructed interpolating/approximating surface.

Description: Given a set of scattered points, PtList, computes a Bspline surface of order UOrder by VOrder that interpolates or least square approximates the given set of scattered points. PtList is a NULL terminated lists of CagdPtStruct structs, with each point holding (u, v, x [, y[, z]]). That is, E3 points create an E1 scalar surface and E5 points create an E3 surface,

See also: BspSrfInterpScatPtsC0Bndry,

3.2.164 BspSrfInterpScatPts2 (sbsp_int.c:688)

interpolation

least square approximation

```
CagdSrfStruct *BspSrfInterpScatPts2(const CagdCtlPtStruct *PtList,
                                    int UOrder,
                                    int VOrder,
                                    int USize,
                                    int VSize,
                                    CagdRType *UKV,
                                    CagdRType *VKV,
                                    CagdRType *MatrixCondition)
```

PtList: A NULL terminating array of linked list of points.

UOrder: Of the to be created surface.

VOrder: Of the to be created surface.

USize: U size of the to be created surface.

VSize: V size of the to be created surface.

UKV: Expected knot vector in U direction, NULL for uniform open.

VKV: Expected knot vector in V direction, NULL for uniform open.

MatrixCondition: Address of a IrtRType to return SVD matrix condition number to. if NULL, this option is ignored

Returns: Constructed interpolating/approximating surface.

Description: This function is a variation BspSrfInterpScatPts function that is less accurate/stable but is faster. The difference is that we solve a LSQ problem as A'*A*Vertices=A'*points where A' is the transpose matrix of A. This method is also refered to as pseudo inverse. The SVD decomposition is still used to calculate the above equation set. Given a set of scattered points, PtList, the function computes a Bspline surface of order UOrder by VOrder that interpolates or least square approximates the M given set of scattered points. PtList is a NULL terminated lists of CagdPtStruct structs, with each point holding (u, v, x [, y[, z]]). That is, E3 points create an E1 scalar surface and E5 points create an E3 surface,

3.2.165 BspSrfInterpScatPtsC0Bndry (sbsp_int.c:602)

interpolation

least square approximation

```
CagdSrfStruct *BspSrfInterpScatPtsC0Bndry(const CagdCtlPtStruct *PtList,
                                          const CagdCrvStruct *UMinCrv,
                                          const CagdCrvStruct *UMaxCrv,
                                          const CagdCrvStruct *VMinCrv,
                                          const CagdCrvStruct *VMaxCrv)
```

PtList: A NULL terminating array of linked list of points.

UMinCrv, UMaxCrv, VMinCrv, VMaxCrv: The four boundary curves. UMin and UMax curves must be in the same function space as are VMin and VMax. This function space is also defining the function space of the fitted surface,

Returns: Constructed interpolating/approximating surface.

Description: Solves a least squares problem as in BspSrfInterpScatPts but forces the four boundary curves to be interpolated.

See also: BspSrfInterpScatPts,

3.2.166 BspSrfInterpolate (sbsp_int.c:243)

interpolation

least square approximation

```
CagdSrfStruct *BspSrfInterpolate(const CagdCtlPtStruct *PtList,
                                 int NumUPts,
                                 int NumVPts,
                                 const CagdRType *UParams,
                                 const CagdRType *VParams,
                                 const CagdRType *UKV,
                                 const CagdRType *VKV,
                                 int ULength,
                                 int VLength,
                                 int UOrder,
                                 int VOrder)
```

PtList: A long linked list (NumUPts * NumVPts) of points to interpolated or least square approximate.

NumUPts: Number of points in PtList in the U direction.

NumVPts: Number of points in PtList in the V direction.

UParams: Parameter at which surface should interpolate or approximate PtList in the U direction.

VParams: Parameter at which surface should interpolate or approximate PtList in the V direction.

UKV: Requested knot vector form the surface in the U direction.

VKV: Requested knot vector form the surface in the V direction.

ULength: Requested length of control mesh of surface in U direction.

VLength: Requested length of control mesh of surface in V direction.

UOrder: Requested order of surface in U direction.

VOrder: Requested order of surface in V direction.

Returns: Constructed interpolating/approximating surface.

Description: Given a set of points on a rectangular grid, PtList, parameter values the surface should interpolate or approximate these grid points, U/VParams, the expected two knot vectors of the surface, U/VKV, the expected lengths U/VLength and orders U/VOrder of the B-spline surface, computes the B-spline surface's coefficients. All points in PtList are assumed of the same type.

3.2.167 BspSrfIsC1DiscontAt (sbspeval.c:606)

```
CagdBType BspSrfIsC1DiscontAt(const CagdSrfStruct *Srf,
                              CagdSrfDirType Dir,
                              CagdRType t)
```

Srf: Surface to examine for C1 discontinuity.

Dir: Parametric direction to examine at.

t: Parameter value to examine at.

Returns: TRUE if Srf has a C1 discontinuity at parameter t in direction Dir, FALSE otherwise.

Description: Examines the mesh at given parametric location for a C1 discontinuity.
See also: BspSrfC1DiscontCrvs, BspKnotAllC1Discont, BspSrfMeshC1Continuous, , BspSrfHasC1Discont,

3.2.168 BspSrfKnotC0Discont (bsp_gen.c:560)

knot vectors
continuity
discontinuity

```
CagdBType BspSrfKnotC0Discont(const CagdSrfStruct *Srf,
                              CagdSrfDirType Dir,
                              CagdRType *t)
```

Srf: To examine its potential discontinuity across Dir.

Dir: Direction to examine the discontinuity across.

t: Where to put the parameter value (knot) that can be C0 discontinuous.

Returns: TRUE if found a C0 discontinuity, FALSE otherwise.

Description: Scans the given knot vector of the given surface for a potential C0 discontinuity. Looks for multiplicities in the knot sequence and then examine the mesh if indeed the mesh is discontinuous at that location. Assumes knot vectors has open end condition.
See also: BspSrfKnotC1Discont, BspKnotC1Discont, BspSrfMeshC1Continuous,

3.2.169 BspSrfKnotC1Discont (bsp_gen.c:596)

knot vectors
continuity
discontinuity

```
CagdBType BspSrfKnotC1Discont(const CagdSrfStruct *Srf,
                              CagdSrfDirType Dir,
                              CagdRType *t)
```

Srf: To examine its potential discontinuity across Dir.

Dir: Direction to examine the discontinuity across.

t: Where to put the parameter value (knot) that can be C1 discontinuous.

Returns: TRUE if found a C1 discontinuity, FALSE otherwise.

Description: Scans the given knot vector of the given surface for a potential C1 discontinuity. Looks for multiplicities in the knot sequence and then examine the mesh if indeed the mesh is discontinuous at that location. Assumes knot vectors has open end condition.
See also: BspSrfKnotC0Discont, BspKnotC1Discont, BspSrfMeshC1Continuous,

3.2.170 BspSrfKnotInsert (bspboehm.c:143)

refinement

knot insertion

```
CagdSrfStruct *BspSrfKnotInsert(const CagdSrfStruct *Srf,
                                CagdSrfDirType Dir,
                                CagdRType t)
```

Srf: To refine by adding a new knot with value equal to t. If Srf is a periodic curve, it is first unwrapped to a float end condition curve.

Dir: Of refinement, either U or V.

t: New knot to insert into Srf.

Returns: The refined surface.

Description: Returns a new surface refined at t (t is inserted as a new knot in Srf) in parametric directipn Dir. See BspCrvKnotInsert for the mathematical background of this knot insertion algorithm.
See also: BspSrfKnotInsertNSame, BspSrfKnotInsertNDiff, BspCrvKnotInsert, BspKnotEvalAlphaCoef,

3.2.171 BspSrfKnotInsertNDiff (sbsp_aux.c:423)

refinement

subdivision

```
CagdSrfStruct *BspSrfKnotInsertNDiff(const CagdSrfStruct *Srf,
                                     CagdSrfDirType Dir,
                                     CagdBType Replace,
                                     CagdRType *t,
                                     int n)
```

Srf: To refine by insertion (upto) n knot of value t.

Dir: Direction of refinement. Either U or V.

Replace: if TRUE, the n knots in t should replace the knot vector of size n of Srf. Sizes must match. If False, n new knots as defined by t will be introduced into Srf.

t: New knots to introduce/replace knot vector of Srf.

n: Size of t.

Returns: Refined Srf with n new knots in direction Dir.

Description: Inserts n knot with different values as defined by the vector t. If, however, Replace is TRUE, the knot are simply replacing the current knot vector.

3.2.172 BspSrfKnotInsertNSame (sbsp_aux.c:352)

refinement

subdivision

```
CagdSrfStruct *BspSrfKnotInsertNSame(const CagdSrfStruct *Srf,
                                     CagdSrfDirType Dir,
                                     CagdRType t,
                                     int n)
```

Srf: To refine by insertion (upto) n knot of value t.

Dir: Direction of refinement. Either U or V.

t: Parameter value of new knot to insert.

n: Maximum number of times t should be inserted.

Returns: Refined Srf with n knots of value t in direction Dir.

Description: Inserts n knot, all with the value t in direction Dir. In no case will the multiplicity of a knot be greater or equal to the curve order.

3.2.173 BspSrfMaxCoefParamMalloc (bsp_knot.c:1924)

extremum

```
CagdRType *BspSrfMaxCoefParamMalloc(const CagdSrfStruct *Srf,
                                    int Axis,
                                    CagdRType *MaxVal)
```

Srf: To compute the parameter node value of the largest coefficient.

Axis: Which axis should we search for maximal coefficient? 1 for X, 2 for Y, etc.

MaxVal: The coefficient itself will be place herein.

Returns: The node UV parameter values of the detected maximal coefficient.

Description: Finds the parameter value with the largest coefficient of the surface using nodes values to estimate the coefficients' parameters. Returns a pointer to an array of two elements holding U and V.

3.2.174 BspSrfMaxCoefParamToData (bsp_knot.c:1871)

extremum

```
CagdRType *BspSrfMaxCoefParamToData(const CagdSrfStruct *Srf,
                                    int Axis,
                                    CagdRType *MaxVal,
                                    CagdRType *UV)
```

Srf: To compute the parameter node value of the largest coefficient.

Axis: Which axis should we search for maximal coefficient? 1 for X, 2 for Y, etc.

MaxVal: The coefficient itself will be place herein.

UV: The node UV parameter values of the detected maximal coefficient.

Returns: The node UV parameter values of the detected maximal coefficient.

Description: Finds the parameter value with the largest coefficient of the surface using nodes values to estimate the coefficients' parameters. Returns a pointer to an array of two elements holding U and V.

3.2.175 BspSrfMeshC1Continuous (bsp_gen.c:684)

```
CagdBType BspSrfMeshC1Continuous(const CagdSrfStruct *Srf,
                                 CagdSrfDirType Dir,
                                 int Idx)
```

Srf: To examine its potential discontinuity across Dir.

Dir: Direction to examine the discontinuity across.

Idx: Index where to examine the discontinuity.

Returns: TRUE if continuous there, FALSE otherwise.

Description: Examine the mesh of the given surface across direction Dir in index of mesh Index for a real discontinuity in the mesh. This index will typically be for a knot multiplicity potential discont.

See also: BspKnotC1Discont, BspSrfIsC1DiscontAt, BspMeshC1PtsCollinear,

3.2.176 BspSrfMeshNormals (sbsp_aux.c:1298)

normal

```
CagdVecStruct *BspSrfMeshNormals(const CagdSrfStruct *Srf,
                                 int UFineNess,
                                 int VFineNess)
```

Srf: To compute normals on a grid of its parametric domain.

UFineNess: U Fineness of imposed grid on Srf's parametric domain.

VFineNess: V Fineness of imposed grid on Srf's parametric domain.

Returns: An vector of unit normals (u increments first).

Description: Evaluates the unit normals of a surface at a mesh defined by subdividing the parametric space into a grid of size UFineNess by VFineNess. The normals are saved in a linear CagdVecStruct vector which is allocated dynamically. Data is saved u inc. first. This routine is much faster than evaluating normal for each point, individually.

See also: CagdSrfNormal, BspSrfNormal, SymbSrfNormalSrf, BzrSrfMeshNormals, , BspSrfMeshNormalsSymb,

3.2.177 BspSrfMeshNormalsSymb (sbsp_aux.c:1600)

normal

```
CagdVecStruct *BspSrfMeshNormalsSymb(CagdSrfStruct *Srf,
                                     int UFineNess,
                                     int VFineNess)
```

Srf: To compute normals on a grid of its parametric domain.

UFineNess: U Fineness of imposed grid on Srf's parametric domain.

VFineNess: V Fineness of imposed grid on Srf's parametric domain.

Returns: An vector of unit normals (u increments first).

Description: Evaluates the unit normals of a surface at a mesh defined by subdividing the parametric space into a grid of size UFineNess by VFineNess. The normals are saved in a linear CagdVecStruct vector which is allocated dynamically. Data is saved u inc. first. This routine is much faster than evaluating normal for each point, individually.
See also: CagdSrfNormal, BspSrfNormal, SymbSrfNormalSrf, BspSrfMeshNormals,

3.2.178 BspSrfMoebiusTransform (sbsp_aux.c:1674)

```
CagdSrfStruct *BspSrfMoebiusTransform(const CagdSrfStruct *CSrf,
                                      CagdRType c,
                                      CagdSrfDirType Dir)
```

CSrf: Surface to apply the Moebius transformation to.

c: The scaling coefficient - c^n is the ratio between the first and last weight of the surface, along each row or column. If c == 0, the first and last weights are made equal, in the first row/column.

Dir: Direction to apply the Moebius transformation, row or col. If Dir == CAGD_BOTH_DIR, the transformation is applied to both the row and column directions, in this order.

Returns: The modified surface with the same shape but different speeds.

Description: Apply the Moebius transformation to a ration Bspline surface. See "Moebius reparametrization of rational Bsplines", by Lee & Lucian, CAGD 8 (1991) pp 213-215.
See also: BspCrvMoebiusTransform, BzrSrfMoebiusTransform,

3.2.179 BspSrfNew (bsp_gen.c:39)

allocation

```
CagdSrfStruct *BspSrfNew(int ULength,
                         int VLength,
                         int UOrder,
                         int VOrder,
                         CagdPointType PType)
```

ULength: Number of control points in the U direction.

VLength: Number of control points in the V direction.

UOrder: The order of the surface in the U direction.

VOrder: The order of the surface in the V direction.

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform Bspline surface.

Description: Allocates the memory required for a new B-spline surface.
See also: BzrSrfNew, BspPeriodicSrfNew, CagdSrfNew, CagdPeriodicSrfNew, TrimSrfNew,

3.2.180 BspSrfNormalMalloc (sbsp_aux.c:1264)

normal

```
CagdVecStruct *BspSrfNormalMalloc(const CagdSrfStruct *Srf,
                                  CagdRType u,
                                  CagdRType v,
                                  CagdBType Normalize)
```

Srf: Bspline surface to evaluate (unit) normal vector for.

u, v: Parametric location of required (unit) normal.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, length is a function of given parametrization.

Returns: A pointer to a vector holding the (unit) normal information, allocated dynamically.

Description: Evaluate the (unit) normal of a surface at a given parametric location. If we fail to compute the normal at given location we retry by moving a tad.
See also: CagdSrfNormal, BzrSrfNormal, BspSrfMeshNormals, SymbSrfNormalSrf, , BspSrfNormalToData,

3.2.181 BspSrfNormalToData (sbsp_aux.c:1199)

normal

```
CagdVecStruct *BspSrfNormalToData(const CagdSrfStruct *Srf,
                                  CagdRType u,
                                  CagdRType v,
                                  CagdBType Normalize,
                                  CagdVecStruct *Normal)
```

Srf: B-spline surface to evaluate (unit) normal vector for.

u, v: Parametric location of required (unit) normal.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, length is a function of given parametrization.

Normal: A pointer to a vector holding the (unit) normal information.

Returns: A pointer to a vector holding the (unit) normal information (the Normal Parameter).

Description: Evaluate the (unit) normal of a surface at a given parametric location. If we fail to compute the normal at given location we retry by moving a tad.
See also: CagdSrfNormal, BzrSrfNormal, BspSrfMeshNormals, SymbSrfNormalSrf, , BspSrfNormalMalloc,

3.2.182 BspSrfOpenEnd (bsp_gen.c:323)

open end conditions

```
CagdSrfStruct *BspSrfOpenEnd(const CagdSrfStruct *Srf)
```

Srf: To convert to a new surface with open end conditions. Input can also be periodic.

Returns: Same surface as Srf but with open end conditions.

Description: Returns a surface with open end conditions, similar to given surface. Open end surface is computed by extracting a subregion from Srf that is the entire surface's parametric domain, by inserting multiple knots at the domain's boundary.
See also: BspCrvOpenEnd,

3.2.183 BspSrfSubdivAtParam (sbsp_aux.c:64)

subdivision

refinement

```
CagdSrfStruct *BspSrfSubdivAtParam(const CagdSrfStruct *Srf,
                                   CagdRType t,
                                   CagdSrfDirType Dir)
```

Srf: To subdivide at parameter value t.

t: Parameter value to subdivide Srf at.

Dir: Direction of subdivision. Either U or V.

Returns: A list of the two subdivided surfaces.

Description: Given a Bspline surface - subdivides it into two sub-surfaces at the given parametric value. Returns pointer to first surface in a list of two subdivided surfaces.
See also: CagdSrfSubdivAtParam, BzrSrfSubdivAtParam, TrimSrfSubdivAtParam,

3.2.184 BspSrfTangentToData (sbsp_aux.c:1145)

tangent

```
CagdVecStruct *BspSrfTangentToData(const CagdSrfStruct *Srf,
                                   CagdRType u,
                                   CagdRType v,
                                   CagdSrfDirType Dir,
                                   CagdBType Normalize,
                                   CagdVecStruct *Tan)
```

Srf: B-spline surface to evaluate (unit) tangent vector for.

u, v: Parametric location of required (unit) tangent.

Dir: The OTHER direction (for historic reasons...) of tangent vector. Either U or V.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, length is a function of given parametrization.

Tan: A pointer to a vector holding the (unit) tangent information.

Returns: A pointer to a vector holding the (unit) tangent information. An Ek, k >= 3 vector, depending on dimension k of Srf.

Description: Evaluates the (unit) tangent to a surface at a given parametric location (u, v) and given direction Dir.

See also: CagdSrfTangent, BzrSrfTangent,

3.2.185 BspSrfsSubdivAtAllDetectedLocations (cagd_aux.c:2412)

```
CagdSrfStruct *BspSrfsSubdivAtAllDetectedLocations(const CagdSrfStruct *Srf,
                                                   CagdSrfTestingFuncType
                                                       SrfTestFunc)
```

Srf: Surface to subdivide at all detected locations by SrfTestFunc.

SrfTestFunc: Surface testing function, like BspSrfKnotC0Discont.

Returns: Set of subdivided surfaces, or NULL if nothing was detected.

Description: Subdivides the given surface Srf at all locations SrfTesTFunc detects. Examples for SrfTesTFunc can be BspSrfKnotC0Discont or BspSrfKnotC1Discont.

See also: BspSrfKnotC0Discont, BspSrfKnotC1Discont, , BspCrvsSubdivAtAllDetectedLocations,

3.2.186 BspVecSpreadEqualItems (bsp_knot.c:3096)

```
CagdBType BspVecSpreadEqualItems(CagdRType *Vec, int Len, CagdRType MinDist)
```

Vec: Vector of reals to spread (almost) equal items in. Modified in place.

Len: Length of vector Vec.

MinDist: The minimal distance two adjacent item should have.

Returns: TRUE if successful, FALSE if cannot be done (MinDist too large).

Description: Given a monotone vector of reals, spread (almost) equal items so they are MinDist apart while keeping the minimal and maximal values the same.

3.2.187 BzrCrv2Polyline (bzr2poly.c:1169)

piecewise linear approximation

polyline

```
CagdPolylineStruct *BzrCrv2Polyline(const CagdCrvStruct *Crv,
                                    int SamplesPerCurve)
```

Crv: To approximate as a polyline.

SamplesPerCurve: Number of samples to approximate with.

Returns: A polyline representing the piecewise linear approximation from, or NULL in case of an error.

Description: Routine to approx. a single Bezier curve as a polyline with SamplesPerCurve samples. Polyline is always E3 CagdPolylineStruct type. Curve is sampled equally spaced in parametric space. NULL is returned in case of an error, otherwise CagdPolylineStruct.

See also: BspCrv2Polyline, BzrSrf2Polylines, IritCurve2Polylines, , SymbCrv2Polyline,

3.2.188 BzrCrvBiNormalMalloc (cbzr_aux.c:672)

binormal

```
CagdVecStruct *BzrCrvBiNormalMalloc(const CagdCrvStruct *Crv,
                                    CagdRType t,
                                    CagdBType Normalize)
```

Crv: Crv for which to compute a (unit) binormal.

t: The parameter at which to compute the unit binormal.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, length is a function of given parametrization.

Returns: A pointer to a vector holding the binormal information. Always an E3 vector.

Description: Returns a (unit) vector, equal to the binormal to Crv at parameter value t. Algorithm: insert (order - 1) knots and using 3 consecutive control points at the refined location (p1, p2, p3), compute to binormal to be the cross product of the two vectors (p1 - p2) and (p2 - p3). Since a curve may have not BiNormal at inflection points or if the 3 points are collinear, NULL will be returned at such cases.

3.2.189 BzrCrvBiNormalToData (cbzr_aux.c:567)

binormal

```
CagdVecStruct *BzrCrvBiNormalToData(const CagdCrvStruct *Crv,
                                    CagdRType t,
                                    CagdBType Normalize,
                                    CagdVecStruct *Vec)
```

Crv: Crv for which to compute a (unit) binormal.

t: The parameter at which to compute the unit binormal.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, length is a function of given parametrization.

Vec: A pointer to a vector holding the binormal information.

Returns: A pointer to a vector holding the binormal information. Always an E3 vector.

Description: Returns a (unit) vector, equal to the binormal to Crv at parameter value t. Algorithm: insert (order - 1) knots and using 3 consecutive control points at the refined location (p1, p2, p3), compute to binormal to be the cross product of the two vectors (p1 - p2) and (p2 - p3). Since a curve may have not BiNormal at inflection points or if the 3 points are collinear, NULL will be returned at such cases.

3.2.190 BzrCrvCreateArc (cagd_arc.c:50)

circle

arc

```
CagdCrvStruct *BzrCrvCreateArc(const CagdPtStruct *Start,
                               const CagdPtStruct *Center,
                               const CagdPtStruct *End)
```

Start: Point of beginning of arc.

Center: Point of arc.

End: Point of end of arc.

Returns: A rational quadratic Bezier curve representing the arc.

Description: Creates an arc at the specified position as a rational quadratic Bezier curve. The arc is assumed to be less than 180 degrees from Start to End in the shorter path as arc where Center as arc center.

See also: BspCrvCreateCircle, BspCrvCreateUnitCircle, BspCrvCreatePCircle, , CagdCreateConicCurve, CagdCrvCreateArc, BspCrvCreateUnitPCircle, , CagdCrvCreateArcCCW, CagdCrvCreateArcCW,

3.2.191 BzrCrvDegreeRaise (cbzr_aux.c:251)

degree raising

```
CagdCrvStruct *BzrCrvDegreeRaise(const CagdCrvStruct *Crv)
```

Crv: To raise its degree by one.

Returns: A curve of one order higher representing the same geometry as Crv.

Description: Returns a new curve, identical to the original but with one degree higher. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one. Then:

$$Q(0) = P(0),\ Q(i) = \frac{i}{k} P(i-1) + \left(1 - \frac{i}{k}\right) P(i),\ Q(k) = P(k-1).$$

See also: BzrCrvDegreeReduce, BzrCrvDegreeRaiseN, PwrCrvDegreeRaise,

3.2.192 BzrCrvDegreeRaiseN (cbzr_aux.c:203)

degree raising

```
CagdCrvStruct *BzrCrvDegreeRaiseN(const CagdCrvStruct *Crv, int NewOrder)
```

Crv: To raise its degree to a NewOrder.

NewOrder: NewOrder for Crv.

Returns: A curve of order NewOrder representing the same geometry as Crv.

Description: Returns a new curve, identical to the original but with order NewOrder. Degree raise is computed by multiplying by a constant 1 curve of order

See also: BzrCrvDegreeRaise, PwrCrvDegreeRaiseN,

3.2.193 BzrCrvDegreeReduce (cbzr_aux.c:322)

degree reduction

degree raising.

```
CagdCrvStruct *BzrCrvDegreeReduce(const CagdCrvStruct *Crv)
```

Crv: To reduce its degree by one.

Returns: A curve of one order lower representing a similar geometry to Crv. The result is optimal in the infinity norm and will be identical to the given curve if the original curve was degree raised.

Description: Returns a new curve, usually similar to the original but with one degree smaller. Let old control polygon be P(i), i = 0 to n, and Q(i) be new one. Then:

$$Q_r(i) = \frac{n\,P(i) - i\,Q_r(i-1)}{n - i}, \quad i = 0, 1, \ldots, n-1.$$

$$Q_l(i-1) = \frac{n\,P(i) - (n-i)\,Q_l(i)}{i}, \quad i = n, n-1, \ldots, 1.$$

and

$$g(i) = \frac{1}{2^{(2n-1)}} \sum_{j=0}^{i} \binom{2n}{2j}$$

yielding,

$$Q(i) = (1 - g(i))\,Q_r(i) + g(i)\,Q_l(i).$$

See also "Curves and Surfaces for Computer Aided Geometric Design" Gerald Farin. Academic Press, Inc. Third Edition.

See also: BzrCrvDegreeRaise, SymbBzrDegReduce,

3.2.194 BzrCrvDerive (cbzr_aux.c:768)

derivatives

```
CagdCrvStruct *BzrCrvDerive(const CagdCrvStruct *Crv, CagdBType DeriveScalar)
```

Crv: To differentiate.

DeriveScalar: TRUE to differentiate each channel independently. FALSE to treat geometry as a rational differentiation.

Returns: Differentiated curve.

Description: Returns a new curve, equal to the given curve, differentiated once. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one then: Q(i) = (k - 1) * (P(i+1) - P(i)), i = 0 to k-2.

See also: CagdCrvDerive, BspCrvDerive, SymbCrvDeriveRational, , BzrCrvDeriveScalar,

3.2.195 BzrCrvDeriveScalar (cbzr_aux.c:818)

derivatives

```
CagdCrvStruct *BzrCrvDeriveScalar(const CagdCrvStruct *Crv)
```

Crv: To differentiate.

Returns: Differentiated curve.

Description: Returns a new curve, equal to the given curve, differentiated once. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one then:

```
Q(i) = (k - 1) * (P(i+1) - P(i)) / (Kv(i + k) - Kv(i + 1)), i = 0 to k-2.
```

For a Euclidean curve this is the same as CagdCrvDerive but for a rational curve the returned curve is not the vector field but simply the derivatives of all the curve's coefficients, including the weights.

See also: BzrCrvDerive, CagdCrvDerive, SymbCrvDeriveRational., BspCrvDerive, BzrCrvDeriveScalar, CagdCrvDeriveScalar,

3.2.196 BzrCrvEvalAtParamMalloc (cbzreval.c:220)

evaluation

```
CagdRType *BzrCrvEvalAtParamMalloc(const CagdCrvStruct *Crv, CagdRType t)
```

Crv: To evaluate at the given parametric location t.

t: The parameter value at which the curve Crv is to be evaluated.

Returns: A vector holding all the coefficients of all components of curve Crv's point type. If for example the curve's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1). This vector is allocated dynamically.

Description: Returns a pointer to a dynamically allocated data, holding the value of the curve at given parametric location t. The curve is assumed to be Bezier.

See also: CagdCrvEvalToData, BspCrvEvalAtParamToData, BspCrvEvalVecAtParam, , BzrCrvEvalVecAtParam, BspCrvEvalCoxDeBoorToData, CagdCrvEvalToPolyline, , BzrCrvEvalAtParam,

3.2.197 BzrCrvEvalAtParamToData (cbzreval.c:259)

evaluation

```
void BzrCrvEvalAtParamToData(const CagdCrvStruct *Crv,
                             CagdRType t,
                             CagdRType *Pt)
```

Crv: To evaluate at the given parametric location t.

t: The parameter value at which the curve Crv is to be evaluated.

Pt: A vector holding all the coefficients of all components of curve Crv's point type. If for example the curve's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Returns: void

Description: Returns a pointer to a dynamic data, holding the value of the curve at given parametric location t. The curve is assumed to be Bezier.

See also: CagdCrvEvalToData, BspCrvEvalAtParamToData, BspCrvEvalVecAtParam, , BzrCrvEvalVecAtParam, BspCrvEvalCoxDeBoorToData, CagdCrvEvalToPolyline, , BzrCrvEvalAtParamToData,

3.2.198 BzrCrvEvalBasisFunc (cbzreval.c:47)

```
CagdRType BzrCrvEvalBasisFunc(int i, int k, CagdRType t)
```

i: I'th basis function.

k: Order of the basis function.

t: Parameter value at which to evaluate the Bezier basis function.

Returns: Value of basis function.

Description: Evaluates the i'th Bezier basis function of order k, at parametric value t (t in [0..1]).

```
The functions is:                i     i          k - i - 1
                    Bi,k-1(t) = ( ) * t  * (1 - t)
                                k-1                                    *
```

See also: BzrCrvEvalBasisFuncs,

3.2.199 BzrCrvEvalBasisFuncs (cbzreval.c:357)

```
CagdRType *BzrCrvEvalBasisFuncs(int k, CagdRType t, CagdRType *Vec)
```

k: Order of the basis function.

t: Parameter value at which to evaluate the Bezier basis function.

Vec: Vector to hold the result, of length Order k.

Returns: Value of basis function's vector . Same as Vec.

Description: Evaluates the vector of Bezier basis functions of order k, at parametric value t (t in [0..1]).

```
The functions are:               i     i          k - i - 1
                    Bi,k-1(t) = ( ) * t  * (1 - t)
                                k-1                                    *
```

See also: BzrCrvEvalBasisFunc,

3.2.200 BzrCrvEvalToPolyline (cbzreval.c:306)

conversion

refinement

evaluation

```
void BzrCrvEvalToPolyline(const CagdCrvStruct *Crv,
                          int FineNess,
                          CagdRType *Points[])
```

Crv: To approximate as a polyline.

FineNess: Control over number of samples.

Points: Where to put the resulting polyline. Assumed to be valid with respect to the dimension of Crv.

Returns: void

Description: Samples the curve at FineNess location equally spaced in the curve's parametric domain.
See also: CagdCrvEvalToData, BspCrvEvalAtParamToData, BzrCrvEvalVecAtParam, , BspCrvEvalVecAtParam, BspCrvEvalCoxDeBoorToData, CagdCrvEvalToPolyline,

3.2.201 BzrCrvEvalVecAtParam (cbzreval.c:168)

evaluation

```
CagdRType BzrCrvEvalVecAtParam(const CagdRType *Vec,
                               int VecInc,
                               int Order,
                               CagdRType t,
                               CagdRType *BasisFuncs)
```

Vec: Coefficients of a scalar B-spline univariate function.

VecInc: Step to move along Vec.

Order: Order of associated geometry.

t: Parameter value where to evaluate the curve.

BasisFuncs: Optional basic functions, if not NULL, in which case t is ignored.

Returns: Geometry's value at parameter value t.

Description: Assumes Vec holds control points for scalar Bezier curve of order Order, and evaluates and returns that curve value at parameter value t. Vec is incremented by VecInc (usually by 1) after each iteration.

See also: CagdCrvEvalToData, BspCrvEvalAtParamToData, BzrCrvEvalAtParamToData, , BspCrvEvalVecAtParam, BspCrvEvalCoxDeBoorToData, CagdCrvEvalToPolyline,

3.2.202 BzrCrvIntegrate (cbzr_aux.c:861)

integrals

```
CagdCrvStruct *BzrCrvIntegrate(const CagdCrvStruct *Crv)
```

Crv: Curve to integrate.

Returns: Integrated curve.

Description: Returns a new Bezier curve, equal to the integral of the given Bezier crv. The given Bezier curve should be nonrational.

$$
\begin{vmatrix} C(t) = \sum_{i=0}^{n} P_i B_i^n(t) \end{vmatrix} = \sum_{i=0}^{n} P_i \int B_i^n(t) = \sum_{i=0}^{n} P_i \frac{1}{n+1-i} B_{i+1}^{n+1}(t) =
$$

$$
= \sum_{j=1}^{n+1} \frac{1}{n+1} \sum_{i=0}^{j-1} P_i B_j^{n+1}(t)
$$

See also: BspCrvIntegrate, BzrSrfIntegrate, CagdCrvIntegrate,

3.2.203 BzrCrvInterp2 (bzr_intr.c:453)

interpolation

```
CagdBType BzrCrvInterp2(IrtRType *Result, const IrtRType *Input, int Size)
```

Result: Where the interpolated control points will be placed.

Input: Points to interpolate at node parameter values.

Size: Of control polygon. Same as the Bezier order.

Returns: TRUE if successful, FALSE otherwise.

Description: Interpolates the given Input data sets at node points and place the Bezier coefficients in Result.

3.2.204 BzrCrvMoebiusTransform (cbzr_aux.c:1029)

```
CagdCrvStruct *BzrCrvMoebiusTransform(const CagdCrvStruct *CCrv, CagdRType c)
```

CCrv: Curve to apply the Moebius transformation to.

c: The scaling coefficient - c^n is the ratio between the first and last weight of the curve. If c == 0, the first and last weights are made equal.

Returns: The modified curve with the same shape but different speed.

Description: Apply the Moebius transformation to a rational Bezier curve.
See also: BspCrvMoebiusTransform, BzrSrfMoebiusTransform,

3.2.205 BzrCrvNew (bzr_gen.c:61)

allocation

```
CagdCrvStruct *BzrCrvNew(int Length, CagdPointType PType)
```

Length: Number of control points

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform Bezier curve.

Description: Allocates the memory required for a new Bezier curve.
See also: BspCrvNew, BspPeriodicCrvNew, CagdCrvNew, CagdPeriodicCrvNew, TrimCrvNew, PwrCrvNew,

3.2.206 BzrCrvNormalMalloc (cbzr_aux.c:739)

normal

```
CagdVecStruct *BzrCrvNormalMalloc(const CagdCrvStruct *Crv,
                                  CagdRType t,
                                  CagdBType Normalize)
```

Crv: Crv for which to compute a (unit) normal.

t: The parameter at which to compute the unit normal.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, length is a function of given parametrization.

Returns: A pointer to a vector holding the normal information. Always an E3 vector.

Description: Returns a (unit) vector, equal to the normal of Crv at parameter value t. Algorithm: returns the cross product of the curve tangent and binormal.

3.2.207 BzrCrvNormalToData (cbzr_aux.c:703)

normal

```
CagdVecStruct *BzrCrvNormalToData(const CagdCrvStruct *Crv,
                                  CagdRType t,
                                  CagdBType Normalize,
                                  CagdVecStruct *N)
```

Crv: Crv for which to compute a (unit) normal.

t: The parameter at which to compute the unit normal.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, length is a function of given parametrization.

N: A pointer to a vector holding the normal information.

Returns: A pointer to a vector holding the normal information. Always an E3 vector.

Description: Returns a (unit) vector, equal to the normal of Crv at parameter value t. Algorithm: returns the cross product of the curve tangent and binormal.

3.2.208 BzrCrvSetCache (cbzreval.c:99)

evaluation

caching

```
void BzrCrvSetCache(int FineNess, CagdBType EnableCache)
```

FineNess: Number of samples to support.

EnableCache: Are we really planning on using this thing?

Returns: void

Description: Sets the Bezier sampling cache - if enabled, a Bezier can be evaluated directly from presampled basis function.

3.2.209 BzrCrvSubdivAtParam (cbzr_aux.c:166)

subdivision

refinement

```
CagdCrvStruct *BzrCrvSubdivAtParam(const CagdCrvStruct *Crv, CagdRType t)
```

Crv: To subdivide at parametr value t.

t: Parameter value to subdivide Crv at.

Returns: A list of the two subdivided curves.

Description: Given a Bezier curve - subdivides it into two sub-curves at the given parametric value. Returns pointer to first curve in a list of two subdivided curves.
See also: BzrSubdivCtlPoly, BspCrvSubdivAtParam,

3.2.210 BzrCrvSubdivCtlPoly (cbzr_aux.c:46)

```
void BzrCrvSubdivCtlPoly(CagdRType * const *Points,
                         CagdRType **LPoints,
                         CagdRType **RPoints,
                         int Length,
                         CagdPointType PType,
                         CagdRType t)
```

Points: To subdivide at parametr value t.

LPoints, RPoints: Where the results are kept.

Length: Of this Bezier curve.

PType: Points types we have here.

t: Parameter value to subdivide curve at.

Returns: void

Description: Apply Bezier subdivision to the given curve at parameter value t, and save the result in data LPoints/RPoints. Note this function could also be called from a B-spline curve with a Bezier knot sequence.
See also: BzrCrvSubdivAtParam, BspCrvSubdivCtlPoly, BzrCrvSubdivCtlPolyStep,

3.2.211 BzrCrvSubdivCtlPolyStep (cbzr_aux.c:106)

```
void BzrCrvSubdivCtlPolyStep(CagdRType * const *Points,
                             CagdRType **LPoints,
                             CagdRType **RPoints,
                             int Length,
                             CagdPointType PType,
                             CagdRType t,
                             int Step)
```

Points: To subdivide at parametr value t.

LPoints, RPoints: Where the results are kept.

Length: Of this Bezier curve.

PType: Points types we have here.

t: Parameter value to subdivide data at.

Step: Stride along the data, 1 for curves, ULength for a surface subdivision along V.

Returns: void

Description: Apply Bezier subdivision to the given data at parameter value t, and save the result in data LPoints/RPoints. Note this function could also be called from a B-spline curve with a Bezier knot sequence. This function is used to Bezier subdivide surfaces (See Step size!).
See also: BzrCrvSubdivAtParam, BspCrvSubdivCtlPoly, BzrCrvSubdivCtlPoly,

3.2.212 BzrCrvTangentToData (cbzr_aux.c:398)

tangent

```
CagdVecStruct *BzrCrvTangentToData(const CagdCrvStruct *Crv,
                                   CagdRType t,
                                   CagdBType Normalize,
                                   CagdVecStruct *Tan)
```

Crv: Crv for which to compute a (unit) tangent.

t: The parameter at which to compute the unit tangent.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, returned is an unnormalized vector in the right direction of the tangent.

Tan: A pointer to a vector holding the tangent information.

Returns: A pointer to a vector holding the tangent information. An Ek, k >= 3 vector, depending on dimension k of Crv.

Description: Returns a (unit) vector, equal to the tangent to Crv at parameter value t. Algorithm: pseudo subdivide Crv at t and using control point of subdivided curve find the tangent as the difference of the 2 end points.

3.2.213 BzrSrf2Curves (bzr2poly.c:1103)

curves

isoparametric curves

```
CagdCrvStruct *BzrSrf2Curves(const CagdSrfStruct *Srf, int NumOfIsocurves[2])
```

Srf: To extract isoparametric curves from.

NumOfIsocurves: In reach (U or V) direction

Returns: List of extracted isoparametric curves. These curves inherit the order and continuity of the original Srf. NULL is returned in case of an error.

Description: Routine to extract from a bezier surface NumOfIsoline isocurve list in each param. direction. Iso parametric curves are sampled equally spaced in parametric space. NULL is returned in case of an error, otherwise list of CagdCrvStruct.
See also: BzrSrf22Polylines, BspSrf2PCurves, SymbSrf2Curves,

3.2.214 BzrSrf2Polygons (bzr2poly.c:140)

polygonization

surface approximation

```
IPPolygonStruct *BzrSrf2Polygons(const CagdSrfStruct *Srf,
                                 const CagdSrf2PlsInfoStrct *TessInfo)
```

Srf: To approximate into triangles.

TessInfo: All auxiliary information/state to tessellate.

Returns: A list of polygons with optional normal and/or UV parametric information. NULL is returned in case of an error.

Description: Routine to convert a single Bezier surface to set of triangles approximating it. FineNess is a fineness control on result and the larger is more triangles may result. A value of 10 is a good start value. NULL is returned in case of an error, otherwise list of IPPolygonStruct.
See also: BspSrf2Polygons, IritSurface2Polygons, IritTrimSrf2Polygons, , CagdSrf2Polygons, TrimSrf2Polygons, CagdSrf2Polygons,

3.2.215 BzrSrf2PolygonsN (bzr2poly.c:192)

evaluation

polygonal approximation

```
IPPolygonStruct *BzrSrf2PolygonsN(const CagdSrfStruct *Srf,
                                  int Nu,
                                  int Nv,
                                  CagdBType ComputeNormals,
                                  CagdBType FourPerFlat,
                                  CagdBType ComputeUV)
```

Srf: To approximate into triangles.

Nu, Nv: The number of uniform samples in U and V of surface.

ComputeNormals: If TRUE, normal information is also computed.

FourPerFlat: If TRUE, four triangles are created per flat surface. If FALSE, only 2 triangles are created.

ComputeUV: If TRUE, UV values are stored and returned as well.

Returns: A list of polygons with optional normal and/or UV parametric information. NULL is returned in case of an error.

Description: Routine to convert a single freeform surface to set of triangles approximating it using a uniform fixed resolution of Nu x Nv. NULL is returned in case of an error, otherwise list of IPPolygonStruct.

See also: BzrSrf2Polygons, BspSrf2Polygons, CagdCrv2Polyline, CagdSrf2Polylines, , CagdSrf2PolygonStrip, CagdSrf2Polygons,

3.2.216 BzrSrf2PolygonsSamples (bzr2poly.c:850)

polygonization

surface approximation

```
CagdBType BzrSrf2PolygonsSamples(const CagdSrfStruct *Srf,
                                 int FineNess,
                                 CagdBType ComputeNormals,
                                 CagdBType ComputeUV,
                                 CagdRType **PtWeights,
                                 CagdPtStruct **PtMesh,
                                 CagdVecStruct **PtNrml,
                                 CagdUVStruct **UVMesh,
                                 int *FineNessU,
                                 int *FineNessV)
```

Srf: To sample in a grid.

FineNess: Control on accuracy, the higher the finer.

ComputeNormals: If TRUE, normal information is also computed.

ComputeUV: If TRUE, UV values are stored and returned as well.

PtWeights: Weights of the evaluations, if rational, to detect poles. NULL if surface not rational.

PtMesh: Evaluated positions of grid of samples.

PtNrml: Evaluated normals of grid of samples or NULL if none.

UVMesh: Evaluated UV vals of grid of samples or NULL if none.

FineNessU, FineNessV: Actual size of PtMesh, PtNrml, UVMesh.

Returns: FALSE is returned in case of an error, TRUE otherwise.

Description: Routine to uniformly sample a single Bezier srf as a grid. FineNess is a fineness control on the result and the larger it is, more samples may result. A value of 10 is a good starting value. FALSE is returned in case of an error, TRUE otherwise.

See also: BspSrf2Polygons, IritSurface2Polygons, IritTrimSrf2Polygons, , CagdSrf2Polygons, TrimSrf2Polygons, BzrSrf2PolygonsSamplesNuNv,

3.2.217 BzrSrf2PolygonsSamplesNuNv (bzr2poly.c:924)

```
CagdBType BzrSrf2PolygonsSamplesNuNv(const CagdSrfStruct *Srf,
                                     int Nu,
                                     int Nv,
                                     CagdBType ComputeNormals,
                                     CagdBType ComputeUV,
                                     CagdRType **PtWeights,
                                     CagdPtStruct **PtMesh,
                                     CagdVecStruct **PtNrml,
                                     CagdUVStruct **UVMesh)
```

Srf: To sample in a grid.

Nu, Nv: The number of uniform samples in U and V of surface.

ComputeNormals: If TRUE, normal information is also computed.

ComputeUV: If TRUE, UV values are stored and returned as well.

PtWeights: Weights of the evaluations, if rational, to detect poles. NULL if surface not rational.

PtMesh: Evaluted positions of grid of samples.

PtNrml: Evaluted normals of grid of samples or NULL if none.

UVMesh: Evaluted UV vals of grid of samples or NULL if none.

Returns: FALSE is returned in case of an error, TRUE otherwise.

Description: Routine to uniformly sample a single Bezier srf as a grid. Nu and Nv fix the grid's sizes. FALSE is returned in case of an error, TRUE otherwise.

See also: BspSrf2Polygons, IritSurface2Polygons, IritTrimSrf2Polygons, , CagdSrf2Polygons, TrimSrf2Polygons, BzrSrf2PolygonsSamples,

3.2.218 BzrSrf2Polylines (bzr2poly.c:1030)

polylines

isoparametric curves

```
CagdPolylineStruct *BzrSrf2Polylines(const CagdSrfStruct *Srf,
                                     int NumOfIsocurves[2],
                                     int SamplesPerCurve)
```

Srf: Srf to extract isoparametric curves from.

NumOfIsocurves: To extarct from Srf in each (U or V) direction.

SamplesPerCurve: Fineness control on piecewise linear curve approximation.

Returns: List of polygons representing a piecewise linear approximation of the extracted isoparamteric curves or NULL is case of an error.

Description: Routine to convert a single Bezier surface to NumOfIsolines polylines in each parametric direction with SamplesPerCurve in each isoparametric curve. Polyline are always E3 of CagdPolylineStruct type. Iso parametric curves are sampled equally spaced in parametric space. NULL is returned in case of an error, otherwise list of CagdPolylineStruct.

See also: BzrCrv2Polyline, BspSrf2Polylines, IritSurface2Polylines, , IritTrimSrf2Polylines, SymbSrf2Polylines, TrimSrf2Polylines, CagdSrf2Polylines,

3.2.219 BzrSrfCrvFromMesh (sbzreval.c:212)

isoparametric curves

curve from mesh

```
CagdCrvStruct *BzrSrfCrvFromMesh(const CagdSrfStruct *Srf,
                                 int Index,
                                 CagdSrfDirType Dir)
```

Srf: To extract a curve from.

Index: Index along the mesh of Srf to extract the curve from.

Dir: Direction of extracted curve. Either U or V.

Returns: A curve from Srf. This curve inherit the order and continuity of surface Srf in direction Dir. However, thiscurve is not on surface Srf, in general.

Description: Extracts a curve from the mesh of a tensor product Bezier surface Srf in direction Dir at index Index.

See also: CagdCrvFromSrf, BzrSrfCrvFromSrf, BspSrfCrvFromSrf, , CagdCrvFromMesh, BspSrfCrvFromMesh,

3.2.220 BzrSrfCrvFromSrf (sbzreval.c:130)

isoparametric curves

curve from surface

```
CagdCrvStruct *BzrSrfCrvFromSrf(const CagdSrfStruct *Srf,
                                CagdRType t,
                                CagdSrfDirType Dir)
```

Srf: To extract an isoparametric curve from.

t: Parameter value of extracted isoparametric curve.

Dir: Direction of the isocurve on the surface. Either U or V.

Returns: An isoparametric curve of Srf. This curve inherits the order and continuity of surface Srf in direction Dir.

Description: Extracts an isoparametric curve out of the given tensor product Bezier surface in direction Dir at the parameter value of t. Operations should prefer the CONST_U_DIR, in which the extraction is somewhat faster if that is possible.

See also: CagdCrvFromSrf, BspSrfCrvFromSrf, CagdCrvFromMesh, BzrSrfCrvFromMesh, , BspSrfCrvFromMesh,

3.2.221 BzrSrfDegreeRaise (sbzr_aux.c:162)

degree raising

```
CagdSrfStruct *BzrSrfDegreeRaise(const CagdSrfStruct *Srf, CagdSrfDirType Dir)
```

Srf: To raise it degree by one.

Dir: Direction to degree raise. Either U or V.

Returns: A surface with one degree higher in direction Dir, representing the same geometry as Srf.

Description: Returns a new Bezier surface, identical to the original but with one degree higher, in the requested direction Dir. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one then:

$$Q(0) = P(0),\ Q(i) = \frac{i}{k} P(i-1) + \left(1 - \frac{i}{k}\right) P(i),\ Q(k) = P(k-1).$$

This is applied to all rows/cols of the surface.

See also: CagdSrfDegreeRaise, BspSrfDegreeRaise, TrimSrfDegreeRaise, , PwrSrfDegreeRaise, PwrSrfDegreeRaiseN,

3.2.222 BzrSrfDegreeRaiseN (sbzr_aux.c:250)

degree raising

```
CagdSrfStruct *BzrSrfDegreeRaiseN(const CagdSrfStruct *Srf,
                                  int NewUOrder,
                                  int NewVOrder)
```

Srf: To raise its degrees.

NewUOrder: New U order of Srf.

NewVOrder: New V order of Srf.

Returns: A surface with higher degrees as prescribed by NewUOrder/NewVOrder.

Description: Returns a new Bezier surface, identical to the original but with higher degrees, as prescribed by NewUOrder, NewVOrder.

See also: CagdSrfDegreeRaise, BzrSrfDegreeRaise, TrimSrfDegreeRaise, , BspSrfDegreeRaise, BzrSrfDegreeRaiseN, CagdSrfDegreeRaiseN, , PwrSrfDegreeRaise, PwrSrfDegreeRaiseN,

3.2.223 BzrSrfDerive (sbzr_aux.c:415)

derivatives

partial derivatives

```
CagdSrfStruct *BzrSrfDerive(const CagdSrfStruct *Srf,
                            CagdSrfDirType Dir,
                            CagdBType DeriveScalar)
```

Srf: To differentiate.

Dir: Direction of differentiation. Either U or V.

DeriveScalar: TRUE to differentiate each channel independently. FALSE to treat geometry as a rational differentiation.

Returns: Differentiated surface.

Description: Returns a new surface equal to the given surface, differentiated once in the direction Dir. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one then:

```
Q(i) = (k - 1) * (P(i+1) - P(i)), i = 0 to k-2.
```

This is applied to all rows/cols of the surface.
See also: CagdSrfDerive, BspSrfDerive, SymbSrfDeriveRational, , BzrSrfDeriveScalar,

3.2.224 BzrSrfDeriveScalar (sbzr_aux.c:489)

derivatives

```
CagdSrfStruct *BzrSrfDeriveScalar(const CagdSrfStruct *Srf, CagdSrfDirType Dir)
```

Srf: To differentiate.

Dir: Direction of tangent vector. Either U or V.

Returns: Differentiated curve.

Description: Returns a new surface equal to the given surface, differentiated once in the direction Dir. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one then:

```
Q(i) = (k - 1) * (P(i+1) - P(i)), i = 0 to k-2.
```

This is applied to all rows/cols of the surface. For a Euclidean surface this is the same as CagdCrvDerive but for a rational surface the returned surface is not the vector field but simply the derivatives of all the surface's coefficients, including the weights.
See also: BzrSrfDerive, CagdSrfDerive, SymbSrfDeriveRational, , BspSrfDerive, BspSrfDeriveScalar, CagdSrfDeriveScalar,

3.2.225 BzrSrfEvalAtParamMalloc (sbzreval.c:95)

evaluation

Bezier

```
CagdRType *BzrSrfEvalAtParamMalloc(const CagdSrfStruct *Srf,
                                   CagdRType u,
                                   CagdRType v)
```

Srf: Surface to evaluate at the given (u, v) location.

u, v: Location where to evaluate the surface.

Returns: A vector holding all the coefficients of all components of curve Crv's point type. If for example the curve's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Description: Evaluates the given tensor product Bezier surface at a given point, by extracting an isoparametric curve along u from the surface and evaluating the curve at parameter v.

```
          u -->
   +----------------------+
   |P0                Pi-1|
V  |Pi               P2i-1|  Parametric space orientation - control mesh.
 ||                       |
 v|Pn-i               Pn-1|
   +----------------------+
```

See also: CagdSrfEval, BspSrfEvalAtParam, BspSrfEvalAtParamToData, TrimSrfEval, , BzrSrfEvalAtParamToData,

3.2.226 BzrSrfEvalAtParamToData (sbzreval.c:49)

evaluation

Bezier

```
void BzrSrfEvalAtParamToData(const CagdSrfStruct *Srf,
                             CagdRType u,
                             CagdRType v,
                             CagdRType *R)
```

Srf: Surface to evaluate at the given (u, v) location.

u, v: Location where to evaluate the surface.

R: A vector holding all the coefficients of all components of curve Crv's point type. If for example the curve's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Returns: void

Description: Evaluates the given tensor product Bezier surface at a given point, by extracting an isoparametric curve along u from the surface and evaluating the curve at parameter v.

```
           u -->
   +----------------------+
   |P0                Pi-1|
 V |Pi               P2i-1|  Parametric space orientation - control mesh.
  ||                     |
  v|Pn-i              Pn-1|
   +----------------------+
```

See also: CagdSrfEval, BspSrfEvalAtParam, BspSrfEvalAtParamToData, TrimSrfEval, BzrSrfEvalAtParam,

3.2.227 BzrSrfIntegrate (sbzr_aux.c:517)

integrals

```
CagdSrfStruct *BzrSrfIntegrate(const CagdSrfStruct *Srf, CagdSrfDirType Dir)
```

Srf: Surface to integrate.

Dir: Direction of integration. Either U or V.

Returns: Integrated surface.

Description: Returns a new Bezier surface, equal to the integral of the given Bezier srf. The given Bezier surface should be nonrational.

See also: BspSrfIntegrate, BzrCrvIntegrate, CagdSrfIntegrate,

3.2.228 BzrSrfMeshNormals (sbzr_aux.c:750)

normal

```
CagdVecStruct *BzrSrfMeshNormals(const CagdSrfStruct *Srf,
                                 int UFineNess,
                                 int VFineNess)
```

Srf: To compute normals on a grid of its parametric domain.

UFineNess: U Fineness of imposed grid on Srf's parametric domain.

VFineNess: V Fineness of imposed grid on Srf's parametric domain.

Returns: An vector of unit normals (u increments first).

Description: Evaluates the unit normals of a surface at a mesh defined by subdividing the parametric space into a grid of size UFineNess by VFineNess. The normals are saved in a linear CagdVecStruct vector which is allocated dynamically. Data is saved u inc. first. This routine is much faster than evaluating normal for each point, individually.

See also: CagdSrfNormal, BspSrfNormal, SymbSrfNormalSrf, BspSrfMeshNormals,

3.2.229 BzrSrfMoebiusTransform (sbzr_aux.c:954)

```
CagdSrfStruct *BzrSrfMoebiusTransform(const CagdSrfStruct *CSrf,
                                      CagdRType c,
                                      CagdSrfDirType Dir)
```

CSrf: Surface to apply the Moebius transformation to.

c: The scaling coefficient - c^n is the ratio between the first and last weight of the surface, along each row or column. If c == 0, the first and last weights are made equal, in the first row/column.

Dir: Direction to apply the Moebius transformation, row or col. If Dir == CAGD_BOTH_DIR, the transformation is applied to both the row and column directions, in this order.

Returns: The modified surface with the same shape but different speeds.

Description: Apply the Moebius transformation to a ration Bezier surface.
See also: BzrCrvMoebiusTransform, BspSrfMoebiusTransform,

3.2.230 BzrSrfNew (bzr_gen.c:30)

allocation

```
CagdSrfStruct *BzrSrfNew(int ULength, int VLength, CagdPointType PType)
```

ULength: Number of control points in the U direction.

VLength: Number of control points in the V direction.

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform Bezier surface.

Description: Allocates the memory required for a new Bezier surface.
See also: BspSrfNew, BspPeriodicSrfNew, CagdSrfNew, CagdPeriodicSrfNew, TrimSrfNew, PwrSrfNew,

3.2.231 BzrSrfNormalMalloc (sbzr_aux.c:717)

normal

```
CagdVecStruct *BzrSrfNormalMalloc(const CagdSrfStruct *Srf,
                                  CagdRType u,
                                  CagdRType v,
                                  CagdBType Normalize)
```

Srf: Bezier surface to evaluate (unit) normal vector for.

u, v: Parametric location of required (unit) normal.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, length is a function of given parametrization.

Returns: A pointer to a vector holding the (unit) normal information.

Description: Evaluate the (unit) normal of a surface at a given parametric location. If we fail to compute the normal at given location we retry by moving a tad.
See also: CagdSrfNormal, BspSrfNormal, SymbSrfNormalSrf,

3.2.232 BzrSrfNormalToData (sbzr_aux.c:657)

normal

```
CagdVecStruct *BzrSrfNormalToData(const CagdSrfStruct *Srf,
                                  CagdRType u,
                                  CagdRType v,
                                  CagdBType Normalize,
                                  CagdVecStruct *Normal)
```

Srf: Bezier surface to evaluate (unit) normal vector for.

u, v: Parametric location of required (unit) normal.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, length is a function of given parametrization.

Normal: A pointer to a vector holding the (unit) normal information.

Returns: A pointer to a vector holding the (unit) normal information (the Normal Parameter).

Description: Evaluate the (unit) normal of a surface at a given parametric location. If we fail to compute the normal at given location we retry by moving a tad.
See also: CagdSrfNormal, BspSrfNormal, SymbSrfNormalSrf,

3.2.233 BzrSrfSubdivAtParam (sbzr_aux.c:118)

subdivision

refinement

```
CagdSrfStruct *BzrSrfSubdivAtParam(const CagdSrfStruct *Srf,
                                   CagdRType t,
                                   CagdSrfDirType Dir)
```

Srf: To subdivide at parameter value t.

t: Parameter value to subdivide Srf at.

Dir: Direction of subdivision. Either U or V.

Returns: A list of the two subdivided surfaces.

Description: Given a Bezier surface - subdivides it into two sub-surfaces at the given parametric value. Returns pointer to first surface in a list of two subdivided surfaces.
See also: CagdSrfSubdivAtParam, BspSrfSubdivAtParam, TrimSrfSubdivAtParam,

3.2.234 BzrSrfSubdivCtlMesh (sbzr_aux.c:53)

```
void BzrSrfSubdivCtlMesh(CagdRType * const *Points,
                         CagdRType **LPoints,
                         CagdRType **RPoints,
                         int ULength,
                         int VLength,
                         CagdPointType PType,
                         CagdRType t,
                         CagdSrfDirType Dir)
```

Points: To subdivide at parametr value t.

LPoints, RPoints: Where the results are kept.

ULength, VLength: Of this Bezier surface, dimensions of Points.

PType: Points types we have here.

t: Parameter value to subdivide curve at.

Dir: Direction of subdivision.

Returns: void

Description: Apply Bezier subdivision to the given curve at parameter value t, and save the result in data LPoints/RPoints. Note this function could also be called from a B-spline curve with a Bezier knot sequence.
See also: BzrCrvSubdivAtParam, BspCrvSubdivCtlPoly, BzrCrvSubdivCtlPolyStep,

3.2.235 BzrSrfTangentToData (sbzr_aux.c:606)

tangent

```
CagdVecStruct *BzrSrfTangentToData(const CagdSrfStruct *Srf,
                                   CagdRType u,
                                   CagdRType v,
                                   CagdSrfDirType Dir,
                                   CagdBType Normalize,
                                   CagdVecStruct *Tangent)
```

Srf: Bezier surface to evaluate (unit) tangent vector for.

u, v: Parametric location of required (unit) tangent.

Dir: The OTHER direction (for historic reasons...) of tangent vector. Either U or V.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, length is a function of given parametrization.

Tangent: A pointer to a vector holding the (unit) tangent information.

Returns: A pointer to a vector holding the (unit) tangent information.

Description: Evaluates the (unit) tangent to a surface at a given parametric location u, v) and given direction Dir.
See also: CagdSrfTangent, BspSrfTangent,

3.2.236 BzrZeroSetNoSubdiv (bzrzrfct.c:1166)

```
CagdPtStruct *BzrZeroSetNoSubdiv(const CagdCrvStruct *Crv,
                                 int Axis,
                                 CagdRType NumericTol,
                                 CagdRType SubdivTol)
```

Crv: A curve to find zeros of.

Axis: The axis along which to find roots.

NumericTol: The numeric tolerance up to which solution is required.

SubdivTol: The tolerance up to which subdivision is to be done.

Returns: List of solutions as tuples of the form (root, multiplicity).

Description: Finds the zeros of a given curve assuming it is a scalar. Employs factoring out of roots without subdivision.
See also: MvarCrvZeroSet, CagdCrvCrvInter, SymbCrvZeroSet, CagdCrvZeroSet, , SymbCrvCrvInter, SymbScalarCrvLowDegZeroSet.,

3.2.237 Cagd2PolyClipPolysAtPoles (bzr2poly.c:102)

```
CagdBType Cagd2PolyClipPolysAtPoles(CagdBType ClipPolysAtPoles)
```

ClipPolysAtPoles: New setting to use or IRIT_QUERY_INT_PROP to query.

Returns: Old value.

Description: Sets the option of clipping polylines and polygon at poles (when the rational curves/surface goes to infinity due to division by zero.). If ClipPolysAtPoles == IRIT_QUERY_INT_PROP, current state is only queried.
See also: CagdSrf2Polygons, CagdSrf2PolygonFast, CagdSrf2PolygonStrip, CagdSrf2PolygonMergeCoplanar,

3.2.238 CagdAllWeightsNegative (cagd2gen.c:2426)

```
CagdBType CagdAllWeightsNegative(CagdRType * const *Points,
                                 CagdPointType PType,
                                 int Len,
                                 CagdBType Flip)
```

Points: Control points to consider and possibly modify in place.

PType: Input point type, as given in Points.

Len: Number of points in Points.

Flip: If TRUE, flips all weights (and points coefficients) in place, so we end up with positive weights only.

Returns: TRUE if original has negative weights, FALSE otherwise.

Description: Returns TRUE if the given control points have negative weights.
See also: CagdPointsHasPoles, CagdAllWeightsSame,

3.2.239 CagdAllWeightsSame (cagd2gen.c:2520)

```
CagdBType CagdAllWeightsSame(CagdRType * const *Points, int Len)
```

Points: Control points to consider.

Len: Number of points in Points.

Returns: TRUE if all weights are the same, FALSE otherwise.

Description: Returns TRUE if given control points has identical weights throughout.
See also: CagdCrvBBox, CagdSrfBBox, GMBBSetBBoxPrecise, SymbCrvPosNegWeights, CagdPointsHasPoles, CagdAllWeightsNegative,

3.2.240 CagdAreClosedCrvs (cagd1gen.c:1496)

```
CagdBType CagdAreClosedCrvs(const CagdCrvStruct *Crvs,
                            const CagdSrfStruct *Srf)
```

Crvs: To test if form a closed loop.

Srf: If not NULL, Crvs are assumed in the parameteric space of Srf, and crossing a boundary of Srf can still be valid loop. Otherwise, if NULL, curves end points are compared directly.

Returns: TRUE if closed, FALSE otherwise.

Description: Returns TRUE if the curves form a closed loop.
See also: CagdIsClosedCrv, CagdIsClosedSrf, CagdIsZeroLenCrv,

3.2.241 CagdBBoxArrayFree (cagd2gen.c:761)

free

```
void CagdBBoxArrayFree(CagdBBoxStruct *BBoxArray, int Size)
```

BBoxArray: To be deallocated.

Size: Of the deallocated array.

Returns: void

Description: Deallocates and frees an array of BBox structure.

3.2.242 CagdBBoxArrayNew (cagd1gen.c:541)

allocation

```
CagdBBoxStruct *CagdBBoxArrayNew(int Size)
```

Size: Size of BBox array to allocate.

Returns: An array of BBox structures of size Size.

Description: Allocates and resets all slots of an array of BBox structures.

3.2.243 CagdBBoxCopy (cagd1gen.c:1074)

copy

```
CagdBBoxStruct *CagdBBoxCopy(const CagdBBoxStruct *BBox)
```

BBox: To be copied.

Returns: A duplicate of BBox.

Description: Allocates and copies all slots of a BBox structure.

3.2.244 CagdBBoxFree (cagd2gen.c:739)

free

```
void CagdBBoxFree(CagdBBoxStruct *BBox)
```

BBox: To be deallocated.

Returns: void

Description: Deallocates and frees all slots of a BBox structure.

3.2.245 CagdBBoxNew (cagd1gen.c:563)

allocation

```
CagdBBoxStruct *CagdBBoxNew(void)
```

Returns: A BBox structure.

Description: Allocates and resets all slots of a BBox structure.

3.2.246 CagdBilinearSrf (cagdruld.c:127)

Bilinear surface

surface constructors

```
CagdSrfStruct *CagdBilinearSrf(const CagdPtStruct *Pt00,
                               const CagdPtStruct *Pt01,
                               const CagdPtStruct *Pt10,
                               const CagdPtStruct *Pt11,
                               CagdPointType PType)
```

Pt00, Pt01, Pt10, Pt11: The four points to construct a bilinear between.

PType: The point time and hence dimension of these four points.

Returns: A bilinear surface with four corners at Ptij.

Description: Constructs a bilinear surface between the four provided points.

3.2.247 CagdBlendTwoSurfaces (hermite.c:417)

```
CagdSrfStruct *CagdBlendTwoSurfaces(const CagdSrfStruct *Srf1,
                                    const CagdSrfStruct *Srf2,
                                    int BlendDegree,
                                    CagdRType TanScale)
```

Srf1: First surface to blend.

Srf2: Second surface to blend.

BlendDegree: Degree of the blending function in U. 2 for C^0, 4 for C^1, 6 for C^2.

TanScale: If C^1, sets the tangency scale factor.

Returns: Resulting blended surface.

Description: Blends the two given surfaces along the u (first) parameter value. Returned surface will start and Srf1(UMin) and terminate at Srf2(UMax). Continuity is governed by the blending degree. 2 for C^0, 4 for C^1. Odd degree will be rounded up to the next even degree. Uses Hermite interpolation for the C^1 case.
See also:

3.2.248 CagdBlossomDegreeRaiseMat (blossom.c:937)

```
CagdBlsmAlphaCoeffStruct *CagdBlossomDegreeRaiseMat(const CagdRType *KV,
                                                    int Order,
                                                    int Len)
```

KV: Of space to degree raise.

Order: Of space to degree raise.

Len: Of control poly/mesh.

Returns: Degree raising matrix.

Description: Computes a new degree raising matrix to degree raise once the following function space, defined using the Order, and knot vector KV of length Len + Order (Len is the length of the control poly/mesh using KV).
See also: CagdCrvBlossomDegreeRaise, CagdSrfBlossomDegreeRaise, CagdCrvDegreeRaise, CagdBlossomDegreeRaiseNMat, CagdDegreeRaiseMatProd,

3.2.249 CagdBlossomDegreeRaiseNMat (blossom.c:1034)

```
CagdBlsmAlphaCoeffStruct *CagdBlossomDegreeRaiseNMat(const CagdRType *KV,
                                                     int Order,
                                                     int NewOrder,
                                                     int Len)
```

KV: Of space to degree raise.

Order: Of space to degree raise.

NewOrder: Destination order to raise to.

Len: Of control poly/mesh.

Returns: Degree raising matrix.

Description: Computes a degree raising matrix to degree raise the following function space to order NewOrder. The space is defined using the Order, and knot ector KV of length Len + Order (Len is the length of the control poly/mesh using KV).
See also: CagdCrvBlossomDegreeRaise, CagdSrfBlossomDegreeRaise, CagdCrvDegreeRaise, CagdBlossomDegreeRaiseMat, CagdDegreeRaiseMatProd,

3.2.250 CagdBlossomEval (blossom.c:438)

```
CagdRType CagdBlossomEval(const CagdRType *Pts,
                          int PtsStep,
                          int Order,
                          const CagdRType *Knots,
                          int KnotsLen,
                          const CagdRType *BlsmVals,
                          int BlsmLen)
```

Pts: Coefficeints or scalar control points to blossom.

PtsStep: Step size between coefficients, typically one.

Order: Order of the freeform geometry,

Knots: Knots of the freeform geometry. If NULL assumed Bezier.

KnotsLen: Length of Knots knot vectors.

BlsmVals: Blossoming values to consider.

BlsmLen: Length of BlsmVals vector.

Returns: Evaluated Blossom

Description: Computes the Blossom over the given points, Pts, with knot sequence Knots, and Blossoming factors BlsmVals. Evaluation is conducted via the Cox - De Boor algorithm with a possibly different parameter at each iteration as prescribed via the Blossoming factors. Note that the Bezier case is supported via the case for which the Knots are NULL. This function assumes no Order multiplicity of knots in interior of KV.
See also: CagdCrvBlossomDegreeRaise, CagdBlossomEvalSymb,

3.2.251 CagdBlsmAddRowAlphaCoef (blossom.c:1648)

```
void CagdBlsmAddRowAlphaCoef(CagdBlsmAlphaCoeffStruct *A,
                             CagdRType *Coefs,
                             int ARow,
                             int ColIndex,
                             int ColLength)
```

A: The current blossome alpha matrix to update.

Coefs: The coefficients to add to the alpha matrix in row ARow.

ARow: The row in A to update.

ColIndex: Starting index in column Col to update.

ColLength: Number of coefficients to update in column Col.

Returns: void

Description: Updates one row, ARow, in the blossom alpha matrix. New coefficients are being added to the current values from ColIndex to ColIndex+ColLength-1.
See also: CagdBlsmEvalSymb, CagdBlsmAllocAlphaCoef, CagdBlsmCopyAlphaCoef, , CagdBlsmFreeAlphaCoef, CagdBlsmScaleAlphaCoef, , CagdBlsmSetDomainAlphaCoef,

3.2.252 CagdBlsmAllocAlphaCoef (blossom.c:1433)

alpha matrix

blossom

```
CagdBlsmAlphaCoeffStruct *CagdBlsmAllocAlphaCoef(int Order,
                                                 int Length,
                                                 int NewOrder,
                                                 int NewLength,
                                                 CagdBType Periodic)
```

Order, Length: Current Order and Length of current function space.

NewOrder, NewLength: New function space, after the blossom.

Periodic: TRUE, if periodic.

Returns: Allocated blossom Alpha matrix.

Description: Allocates the CagdBlsmAlphaCoeffStruct data structrure.
See also: CagdCrvBlossomDegreeRaise, CagdBlossomEvalSymb, CagdBlsmAddRowAlphaCoef, CagdBlsmCopyAlphaCoef, CagdBlsmFreeAlphaCoef, CagdBlsmScaleAlphaCoef, , CagdBlsmSetDomainAlphaCoef,

3.2.253 CagdBlsmCopyAlphaCoef (blossom.c:1518)

alpha matrix

refinement

```
CagdBlsmAlphaCoeffStruct *CagdBlsmCopyAlphaCoef(const CagdBlsmAlphaCoeffStruct
                                                                        *A)
```

A: Blossom alpha matrix to copy.

Returns: Copied matrix.

Description: Copies the CagdBlsmAlphaCoeffStruct data structrure.
See also: CagdCrvBlossomDegreeRaise, CagdBlossomEvalSymb, CagdBlsmAddRowAlphaCoef, CagdBlsmAllocAlphaCoef, CagdBlsmFreeAlphaCoef, CagdBlsmScaleAlphaCoef, , CagdBlsmSetDomainAlphaCoef,

3.2.254 CagdBlsmEvalSymb (blossom.c:83)

```
CagdRType *CagdBlsmEvalSymb(int Order,
                            const CagdRType *Knots,
                            int KnotsLen,
                            const CagdRType *BlsmVals,
                            int BlsmLen,
                            int *RetIdxFirst,
                            int *RetLength,
                            CagdRType *RetVec,
                            void *Cache)
```

Order: Order of the freeform geometry,

Knots: Knots of the freeform geometry. If NULL assumed Bezier.

KnotsLen: Length of Knots knot vectors.

BlsmVals: Blossoming values to consider.

BlsmLen: Length of BlsmVals vector.

RetIdxFirst: Index of first input coefficient to blend returned vector with.

RetLength: Length of returned blend vector.

RetVec: Vector of blending values of the input coefficients for this blossom evaluation. This vector is maintained by this function and should not be freed by the caller of this function. No need to provide both Cache and RetVec.

Cache: Optional Cache (can be NULL) to use as created by CagdBlsmEvalSymbAllocCache. No need to provide both Cache and RetVec.

Returns: Same as RetVec.

Description: Same as CagdBlossomEval, but computes the result symbolically. That is, get the contribution of each input coefficients to this blossom. This function assumes no Order multiplicity of knots in interior of KV.
See also: CagdBlsmEvalSymbFreeCache, CagdBlsmEvalSymbAllocCache, , CagdCrvBlossomDegreeRaise, CagdBlossomEval, CagdBlsmAddRowAlphaCoef, CagdBlsmAllocAlphaCoef, CagdBlsmCopyAlphaCoef, , CagdBlsmFreeAlphaCoef, CagdBlsmScaleAlphaCoef, , CagdBlsmSetDomainAlphaCoef,

3.2.255 CagdBlsmEvalSymbAllocCache (blossom.c:364)

```
void *CagdBlsmEvalSymbAllocCache()
```

Returns: Allocated new cache.

Description: Auxiliary function of CagdBlsmEvalSymb to allocate a cache for the blossom evaluations.
See also: CagdBlsmEvalSymb, CagdBlsmEvalSymbFreeCache,

3.2.256 CagdBlsmEvalSymbFreeCache (blossom.c:392)

```
void CagdBlsmEvalSymbFreeCache(void *Cache)
```

Cache: Blossom cache to free.

Returns: void

Description: Auxiliary function of CagdBlsmEvalSymb to free a cache for the blossom evaluations.
See also: CagdBlsmEvalSymb, CagdBlsmEvalSymbAllocCache,

3.2.257 CagdBlsmFreeAlphaCoef (blossom.c:1606)

alpha matrix

blossom

```
void CagdBlsmFreeAlphaCoef(CagdBlsmAlphaCoeffStruct *A)
```

A: Blossom Alpha matrix to free.

Returns: void

Description: Frees the CagdBlsmAlphaCoeffStruct data structrure.
See also: CagdCrvBlossomDegreeRaise, CagdBlossomEvalSymb, CagdBlsmAddRowAlphaCoef, CagdBlsmAllocAlphaCoef, CagdBlsmCopyAlphaCoef, CagdBlsmScaleAlphaCoef, , CagdBlsmSetDomainAlphaCoef,

3.2.258 CagdBlsmScaleAlphaCoef (blossom.c:1691)

```
void CagdBlsmScaleAlphaCoef(CagdBlsmAlphaCoeffStruct *A, CagdRType Scl)
```

A: Blossom alpha matrix to scale all it coefficients.

Scl: Scaling factor.

Returns: void

Description: Scale all the coefficients in the given blossom alpha matrix by Scl.
See also: CagdBlsmEvalSymb, CagdBlsmAllocAlphaCoef, CagdBlsmCopyAlphaCoef, , CagdBlsmFreeAlphaCoef, CagdBlsmSetDomainAlphaCoef, , CagdBlsmAddRowAlphaCoef,

3.2.259 CagdBlsmSetDomainAlphaCoef (blossom.c:1718)

```
void CagdBlsmSetDomainAlphaCoef(CagdBlsmAlphaCoeffStruct *A)
```

A: Blossom alpha matrix to update its ColIndex/ColLength settings, in place.

Returns: void

Description: Update domain bounds ColIndex/ColLength in the blossom alpha matrix A.
See also: CagdBlsmEvalSymb, CagdBlsmAllocAlphaCoef, CagdBlsmCopyAlphaCoef, , CagdBlsmFreeAlphaCoef, CagdBlsmScaleAlphaCoef, CagdBlsmAddRowAlphaCoef,

3.2.260 CagdBndryAsOneCrvFromSrf (cagd_aux.c:2859)

curve from surface

```
CagdCrvStruct *CagdBndryAsOneCrvFromSrf(const CagdSrfStruct *Srf)
```

Srf: To extract the boundary from.

Returns: A closed curve, formed from the boundaries of Srf.

Description: Extracts the four boundaries of a given surface, as a single closed curve.
See also: CagdBndryCrvsFromSrf, CagdBndryCrvFromSrf,

3.2.261 CagdBndryCrvFromSrf (cagd_aux.c:2790)

isoparametric curves

curve from surface

```
CagdCrvStruct *CagdBndryCrvFromSrf(const CagdSrfStruct *Srf,
                                   CagdSrfBndryType Bndry)
```

Srf: To extract the boundary from.

Bndry: The boundary to extract.

Returns: The extracted boundary curve.

Description: Extracts one boundary curve of the given surface.

3.2.262 CagdBndryCrvsFromSrf (cagd_aux.c:2829)

isoparametric curves

curve from surface

```
CagdCrvStruct **CagdBndryCrvsFromSrf(const CagdSrfStruct *Srf,
                                     CagdCrvStruct *Crvs[4])
```

Srf: To extract the boundary from.

Crvs: A pointer to a vector of four curve pointers, representing the four boundaries of surface Srf in order of UMin, UMax, VMin, VMax.

Returns: A pointer to a vector of four curve pointers, representing the four boundaries of surface Srf in order of UMin, UMax, VMin, VMax.

Description: Extracts the four boundary curves of the given surface.

3.2.263 CagdBoolSumSrf (cagdbsum.c:182)

Boolean sum

surface constructors

```
CagdSrfStruct *CagdBoolSumSrf(const CagdCrvStruct *CCrvLeft,
                              const CagdCrvStruct *CCrvRight,
                              const CagdCrvStruct *CCrvTop,
                              const CagdCrvStruct *CCrvBottom)
```

CCrvLeft: Left boundary curve of Boolean sum surface to be created.

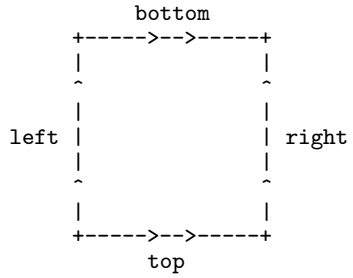
CCrvRight: Right boundary curve of Boolean sum surface to be created.

CCrvTop: Top boundary curve of Boolean sum surface to be created.

CCrvBottom: Bottom boundary curve of Boolean sum surface to be created.

Returns: A Boolean sum surface constructed using given four curves.

Description: Constructs a Boolean sum surface using the four provided boundary curves. Curve's end points must meet at the four surface corners if surface boundary are to be identical to the four given curves.



See also: MvarTrivarBoolSum, CagdBoolSumSrf2, CagdBoolSumSrfRtnl,

3.2.264 CagdBoolSumSrf2 (cagdbsum.c:63)

Boolean sum

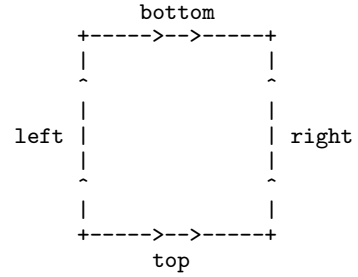
surface constructors

```
CagdSrfStruct *CagdBoolSumSrf2(const CagdCrvStruct *Crv1,
                               const CagdCrvStruct *Crv2,
                               const CagdCrvStruct *Crv3,
                               const CagdCrvStruct *Crv4)
```

Crv1, Crv2, Crv3, Crv4: Four boundary curves forming a freefrom rectangle in the XY plane. Crv2, Crv3, and Crv4 will be properly oriented, following Crv1.

Returns: A Boolean sum surface constructed using given four curves.

Description: Constructs a Boolean sum surface using the four provided boundary curves. Curve's end points must meet at the four surface corners if surface boundary are to be identical to the four given curves.



See also: MvarTrivarBoolSum3, CagdBoolSumSrf, CagdBoolSumSrfRtnl,

3.2.265 CagdBoolSumSrfRtnl (cagdbsum.c:754)

Boolean sum

surface constructors

```
CagdSrfStruct *CagdBoolSumSrfRtnl(const CagdCrvStruct *CrvLeft,
                                  const CagdCrvStruct *CrvRight,
                                  const CagdCrvStruct *CrvTop,
                                  const CagdCrvStruct *CrvBottom)
```

CrvLeft: Left boundary curve of Boolean sum surface to be created.

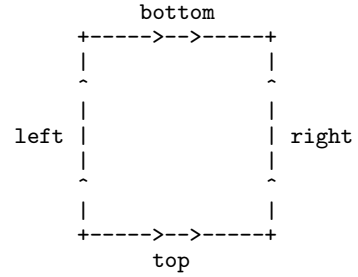
CrvRight: Right boundary curve of Boolean sum surface to be created.

CrvTop: Top boundary curve of Boolean sum surface to be created.

CrvBottom: Bottom boundary curve of Boolean sum surface to be created.

Returns: A Boolean sum surface constructed using given four curves.

Description: Constructs a Boolean sum surface using the four provided boundary curves. Curve's end points must meet at the four surface corners if surface boundary are to be identical to the four given curves.



This function is similar to CagdBoolSumSrf but if rational it will not raise the degrees at the cost of changing the internal parametrization

See also: MvarTrivarBoolSumRtnl, CagdBoolSumSrf,

3.2.266 CagdBspCrvPDMFitting (cbsp_fit.c:1316)

points cloud approximization

```
static CagdCrvStruct *CagdBspCrvPDMFitting(
                                    CagdPType *PtList,
                                    int NumOfPoints,
                                    CagdCrvStruct *InitCrv,
                                    RegTermCalculatorFuncType CalcRegularization,
                                    RegMatrixCalculatorFuncType CalcRegMatrix,
                                    int MaxIterations,
                                    CagdRType ErrorLimit,
                                    CagdRType ErrorChangeLimit,
                                    CagdRType Lambda)
```

PtList: Points cloud we want to approximate.

NumOfPoints: Number of points in PtList.

InitCrv: Pointer to an initial fitting curve. Its order and length will be taken as order and length of the desired fitting curve.

CalcRegularization: Pointer to a function that should be used for calculating regularization term value.

CalcRegMatrix: Pointer to a function that should be used for calculating regularization term minimization matrix.

MaxIterations: Maximum iterations to perform (stop condition).

ErrorLimit: Minimum error (stop condition).

ErrorChangeLimit: Minimum error change (stop condition).

Lambda: Weight of regularization term.

Returns: B-spline curve that fits the input points cloud.

Description: Calculates b-spline curve that fits (approximates) the given points Using input init curve, PDM fitting method and specified regularization term.
See also: CagdBspCrvSDMFitting,

3.2.267 CagdBspCrvSDMFitting (cbsp_fit.c:1141)

points cloud approximization

```
static CagdCrvStruct *CagdBspCrvSDMFitting(
                                    CagdPType *PtList,
                                    int NumOfPoints,
                                    CagdCrvStruct *InitCrv,
                                    RegTermCalculatorFuncType CalcRegularization,
                                    RegMatrixCalculatorFuncType CalcRegMatrix,
                                    int MaxIterations,
                                    CagdRType ErrorLimit,
                                    CagdRType ErrorChangeLimit,
                                    CagdRType Lambda)
```

PtList: Points cloud we want to approximate.

NumOfPoints: Number of points in PtList.

InitCrv: Pointer to an initial fitting curve. Its order and length will be taken as order and length of the desired fitting curve.

CalcRegularization: Pointer to a function that should be used for calculating regularization term value.

CalcRegMatrix: Pointer to a function that should be used for calculating regularization term minimization matrix.

MaxIterations: Maximum iterations to perform (stop condition).

ErrorLimit: Minimum error (stop condition).

ErrorChangeLimit: Minimum error change (stop condition).

Lambda: Weight of regularization term.

Returns: B-spline curve that fits the input points cloud.

Description: Calculates b-spline curve that fits (approximates) the given points Using input init curve, SDM fitting method and specified regularization term.
See also: CagdBspCrvPDMFitting,

3.2.268 CagdBsplineCrvFitting (cbsp_fit.c:234)

```
CagdCrvStruct *CagdBsplineCrvFitting(CagdPType *PtList,
                                     int NumOfPoints,
                                     int Length,
                                     int Order,
                                     CagdBType IsPeriodic,
                                     CagdBspFittingType AgorithmType,
                                     int MaxIter,
                                     CagdRType ErrorLimit,
                                     CagdRType ErrorChangeLimit,
                                     CagdRType Lambda)
```

PtList: Points cloud we want to approximate.

NumOfPoints: Number of points in PtList.

Length: The desired length of the output b-spline curve.

Order: The desired order of the output b-spline curve.

IsPeriodic: TRUE for periodic output curve, FALSE for open end.

AgorithmType: Fitting algorithm type (CAGD_PDM_FITTING, CAGD_SDM_FITTING, etc).

MaxIter: Maximum iterations to perform (stop condition).

ErrorLimit: Minimum error (stop condition).

ErrorChangeLimit: Minimum error change (stop condition).

Lambda: Weight of regularization term.

Returns: Output b-spline curve.

Description: Calculates b-spline curve that fits (approximates) the given points cloud using SD error function and Energy2 regulation function. There are three stop conditions: 1) 100 iterations 2) Error = 0.1 3) Error change = 0.005 The initial curve is a least square approximating b-spline.

3.2.269 CagdBsplineCrvFittingWithInitCrv (cbsp_fit.c:166)

```
CagdCrvStruct *CagdBsplineCrvFittingWithInitCrv(CagdPType *PtList,
                                                int NumOfPoints,
                                                CagdCrvStruct *InitCrv,
                                                CagdBspFittingType AgorithmType,
                                                int MaxIter,
                                                CagdRType ErrorLimit,
                                                CagdRType ErrorChangeLimit,
                                                CagdRType Lambda)
```

PtList: Points cloud we want to approximate.

NumOfPoints: Number of points in PtList.

InitCrv: Initial fitting curve.

AgorithmType: Fitting algorithm type (CAGD_PDM_FITTING, CAGD_SDM_FITTING, etc).

MaxIter: Maximum iterations to perform (stop condition).

ErrorLimit: Minimum error (stop condition).

ErrorChangeLimit: Minimum error change (stop condition).

Lambda: Weight of regularization term.

Returns: Output b-spline curve.

Description: Calculates b-spline curve that fits (approximates) the given points cloud using SD error function and Energy2 regulation function. There are three stop conditions: 1) Maximum iterations. 2) Error = 0 (i.e. curve passes through all the points) 3) Error change = 0 (the iteration gives no improvement)

3.2.270 CagdCnvrtBsp2BzrCrv (cbzr_aux.c:945)

conversion

```
CagdCrvStruct *CagdCnvrtBsp2BzrCrv(const CagdCrvStruct *CCrv)
```

CCrv: A B-spline curve to convert to a Bezier curve.

Returns: A list of Bezier curves representing the B-spline curve Crv.

Description: Converts a B-spline curve into a set of Bezier curves by subdividing the B-spline curve at all its internal knots. Returned is a list of Bezier curves.
See also: CagdCnvrtBzr2BspCrv, CagdCnvrtBzr2PwrCrv, CagdCnvrtPwr2BzrCrv,

3.2.271 CagdCnvrtBsp2BzrSrf (sbzr_aux.c:858)

conversion

```
CagdSrfStruct *CagdCnvrtBsp2BzrSrf(const CagdSrfStruct *CSrf)
```

CSrf: B-spline surface to convert to a Bezier surface.

Returns: A list of Bezier surfaces representing same geometry as Srf.

Description: Convert a B-spline surface into a set of Bezier surfaces by subdividing the B-spline surface at all its internal knots. Returned is a list of Bezier surface.
See also: CagdCnvrtBzr2BspSrf,

3.2.272 CagdCnvrtBsp2OpenCrv (cbsp_aux.c:1631)

conversion

```
CagdCrvStruct *CagdCnvrtBsp2OpenCrv(const CagdCrvStruct *Crv)
```

Crv: Bspline curve to convert to open end conditions.

Returns: A Bspline curve with open end conditions, representing the same geometry as Crv.

Description: Converts a Bspline curve to a Bspline curve with open end conditions.

3.2.273 CagdCnvrtBsp2OpenSrf (sbsp_aux.c:1937)

conversion

```
CagdSrfStruct *CagdCnvrtBsp2OpenSrf(const CagdSrfStruct *Srf)
```

Srf: Bspline surface to convert to open end conditions.

Returns: A Bspline surface with open end conditions, representing the same geometry as Srf.

Description: Converts a Bspline surface to a Bspline surface with open end conditions.

3.2.274 CagdCnvrtBzr2BspCrv (cbzr_aux.c:905)

conversion

```
CagdCrvStruct *CagdCnvrtBzr2BspCrv(const CagdCrvStruct *Crv)
```

Crv: A Bezier curve to convert to a Bspline curve.

Returns: A Bspline curve representing Bezier curve Crv.

Description: Converts a Bezier curve into Bspline curve by adding an open knot vector.
See also: CagdCnvrtBsp2BzrCrv, CagdCnvrtBzr2PwrCrv, CagdCnvrtPwr2BzrCrv,

3.2.275 CagdCnvrtBzr2BspSrf (sbzr_aux.c:815)

conversion

```
CagdSrfStruct *CagdCnvrtBzr2BspSrf(const CagdSrfStruct *Srf)
```

Srf: Bezier surface to convert to a B-spline surface.

Returns: A Bspline surface representing same geometry as Srf.

Description: Converts a Bezier surface into a B-spline surface by adding open end knot vector with no interior knots.

See also: CagdCnvrtBsp2BzrSrf,

3.2.276 CagdCnvrtBzr2PwrCrv (bzr_pwr.c:57)

power basis

conversion

```
CagdCrvStruct *CagdCnvrtBzr2PwrCrv(const CagdCrvStruct *Crv)
```

Crv: To convert into Power basis function representation.

Returns: Same geometry, but in the Power basis.

Description: Converts the given curve from Bezier basis functions to a Power basis functions. Using:

$$B_i^n(t) = \sum_{j=i}^{n} (-1)^{j-i} \binom{n}{j} \binom{j}{i} t^j$$

Which can be derived by expanding the $(1-t)^{n-i}$ term in bezier basis function definition as:

$$(1-t)^{n-i} = \sum_{j=0}^{n-i} \binom{n-i}{j} (-t)^j$$ using binomial expansion.

This routine simply take the weight of each Bezier basis function B(t) and spread it into the different power basis t^j function scaled by:

$$(-1)^{j-i} \binom{n}{j} \binom{j}{i}$$

See also: CagdCnvrtBzr2BspCrv, CagdCnvrtBsp2BzrCrv, CagdCnvrtPwr2BzrCrv,

3.2.277 CagdCnvrtBzr2PwrSrf (bzr_pwr.c:195)

power basis

conversion

```
CagdSrfStruct *CagdCnvrtBzr2PwrSrf(const CagdSrfStruct *Srf)
```

Srf: To convert into Power basis function representation.

Returns: Same geometry, but in the Power basis.

Description: Converts the given surface from Bezier basis functions to a Power basis functions. Using:

$$B_i^n(t) = \sum_{p=i}^{n} (-1)^{p-i} \binom{n}{p} \binom{p}{i} t^p$$

or

$$B_i^n(u)\, B_j^m(v) = \sum_{p=i}^{n} \sum_{q=j}^{m} (-1)^{p-i+q-j} \binom{n}{p} \binom{p}{i} \binom{m}{q} \binom{q}{j} u^p v^q$$

This routine simply take the weight of each product of two Bezier basis functions Bi(u) Bj(v) and spread it into the different power basis u^j v^k functions scaled by:

$$(-1)^{p-i+q-j} \binom{n}{p} \binom{p}{i} \binom{m}{q} \binom{q}{j}$$

3.2.278 CagdCnvrtCrvToCtlPts (cbsp_aux.c:1690)

```
CagdCtlPtStruct *CagdCnvrtCrvToCtlPts(const CagdCrvStruct *Crv)
```

Crv: To the curve to convert to list of control points.

Returns: List of control points of curve Crv.

Description: Convert a curve into a list of control points.

3.2.279 CagdCnvrtCrvToSrf (cagdruld.c:161)

```
CagdSrfStruct *CagdCnvrtCrvToSrf(const CagdCrvStruct *Crv, CagdSrfDirType Dir)
```

Crv: A Crv to promote into a surface

Dir: Direction of Crv (of degree of Crv). Either U or V.

Returns: The surface promoted from Crv.

Description: Promotes a curve to a surface by creating a constant surface in the new direction. Dir controls if the curve should be U or V surface direction. The resulting surface is degenerate in that its speed is zero in the ruled direction and hence the surface is not regular.

3.2.280 CagdCnvrtFloat2OpenCrv (cbsp_aux.c:1576)

conversion

```
CagdCrvStruct *CagdCnvrtFloat2OpenCrv(const CagdCrvStruct *Crv)
```

Crv: Bspline curve to convert to open end conditions.

Returns: A Bspline curve with open end conditions, representing the same geometry as Crv.

Description: Converts a float Bspline curve to a Bspline curve with open end conditions.

3.2.281 CagdCnvrtFloat2OpenSrf (sbsp_aux.c:1902)

conversion

```
CagdSrfStruct *CagdCnvrtFloat2OpenSrf(const CagdSrfStruct *Srf)
```

Srf: Bspline surface to convert to open end conditions.

Returns: A Bspline surface with open end conditions, representing the same geometry as Srf.

Description: Converts a float Bspline surface to a Bspline surface with open end conditions.
See also: CagdCnvrtPeriodic2FloatSrf,

3.2.282 CagdCnvrtLinBspCrv2Polyline (cbsp_aux.c:1471)

linear curves

conversion

```
CagdPolylineStruct *CagdCnvrtLinBspCrv2Polyline(const CagdCrvStruct *Crv,
                                                CagdBType FilterIdentical)
```

Crv: A linear B-spline curve to convert to a polyline.

FilterIdentical: TRUE to filter almost identical adjacent vertices.

Returns: A polyline same as linear curve Crv.

Description: Returns a new polyline in E3 representing same geometry as the given linear B-spline curve.
See also: UserPolylines2LinBsplineCrvs, UserPolyline2LinBsplineCrv, , CagdCnvrtPolyline2LinBspCrv, CagdCnvrtPtList2Polyline,

3.2.283 CagdCnvrtPeriodic2FloatCrv (cbsp_aux.c:1523)

conversion

```
CagdCrvStruct *CagdCnvrtPeriodic2FloatCrv(const CagdCrvStruct *Crv)
```

Crv: Bspline curve to convert to floating end conditions. Assume Crv is either periodic or has floating end condition.

Returns: A Bspline curve with floating end conditions, representing the same geometry as Crv.

Description: Converts a Bspline curve to a Bspline curve with floating end conditions.

3.2.284 CagdCnvrtPeriodic2FloatSrf (sbsp_aux.c:1827)

conversion

```
CagdSrfStruct *CagdCnvrtPeriodic2FloatSrf(const CagdSrfStruct *Srf)
```

Srf: Bspline surface to convert to floating end conditions. Assume Srf is either periodic or has floating end condition.

Returns: A Bspline surface with floating end conditions, representing the same geometry as Srf.

Description: Converts a Bspline surface into a Bspline surface with floating end conditions.
See also: CagdCnvrtFloat2OpenSrf,

3.2.285 CagdCnvrtPolyline2LinBspCrv (cbsp_aux.c:1446)

linear curves

conversion

```
CagdCrvStruct *CagdCnvrtPolyline2LinBspCrv(const CagdPolylineStruct *Poly,
                                           CagdPointType PType)
```

Poly: To convert to a linear B-spline curve.

PType: Typically CAGD_PT_E3_TYPE but can be any other type.

Returns: A linear B-spline curve representing Poly.

Description: Returns a new linear B-spline curve constructed from the given polyline. consecutive Identical points in the polyline are skipped.
See also: UserPolylines2LinBsplineCrvs, CagdCnvrtPolyline2LinBspCrv, , CagdCnvrtLinBspCrv2Polyline, CagdCnvrtPtList2Polyline, CagdCnvrtIritPolyline2CagdPolyline, CagdCnvrtPolyline2LinBspCrv2,

3.2.286 CagdCnvrtPolyline2LinBspCrv2 (cbsp_aux.c:1366)

linear curves

conversion

```
CagdCrvStruct *CagdCnvrtPolyline2LinBspCrv2(const CagdPolylineStruct *Poly,
                                            CagdPointType PType,
                                            CagdBType SkipIdenticalPoints)
```

Poly: To convert to a linear B-spline curve.

PType: Typically CAGD_PT_E3_TYPE but can be any other type.

SkipIdenticalPoints: TRUE iff consecutive points are to be skipped.

Returns: A linear B-spline curve representing Poly.

Description: Returns a new linear B-spline curve constructed from the given polyline. consecutive Identical points in the polyline are skipped if specified only.

See also: UserPolylines2LinBsplineCrvs, CagdCnvrtPolyline2LinBspCrv, , CagdCnvrtLinBspCrv2Polyline, CagdCnvrtPtList2Polyline, CagdCnvrtIritPolyline2CagdPolyline, CagdCnvrtPolyline2LinBspCrv,

3.2.287 CagdCnvrtPolyline2PtList (cbsp_aux.c:1324)

```
CagdPtStruct *CagdCnvrtPolyline2PtList(const CagdPolylineStruct *Poly)
```

Poly: Input polyline to convert into a point list.

Returns: Converted point list.

Description: Converts a polyline into a list of points.

See also: CagdCnvrtLinBspCrv2Polyline, CagdCnvrtPolyline2LinBspCrv, , CagdCnvrtPtList2Polyline,

3.2.288 CagdCnvrtPtList2Polyline (cbsp_aux.c:1271)

```
CagdPolylineStruct *CagdCnvrtPtList2Polyline(const CagdPtStruct *Pts,
                                             CagdPolylineStruct **Params)
```

Pts: Input point list to convert into a polyline.

Params: Optional polylines of parameters if found is saved here.

Returns: Converted polyline.

Description: Converts a list of points into a polyline in E3.

See also: CagdCnvrtLinBspCrv2Polyline, CagdCnvrtPolyline2LinBspCrv, , CagdCnvrtPolyline2PtList,

3.2.289 CagdCnvrtPwr2BzrCrv (bzr_pwr.c:124)

power basis

conversion

```
CagdCrvStruct *CagdCnvrtPwr2BzrCrv(const CagdCrvStruct *Crv)
```

Crv: To convert to Bezier basis functions.

Returns: Same geometry, in the Bezier basis functions.

Description: Converts the given curve from Power basis functions to Bezier basis functions. Using:

$$t^i = \sum_{j=i}^{n} \frac{\binom{j}{i}}{\binom{n}{i}} B_j^n(t)$$

This routine simply take the weight of each Power basis function t^i and spread it into the different basis basis function B(t) scaled by:

$$\frac{\binom{j}{i}}{\binom{n}{i}}$$

See also: CagdCnvrtBzr2BspCrv, CagdCnvrtBsp2BzrCrv, CagdCnvrtBzr2PwrCrv,

3.2.290 CagdCnvrtPwr2BzrSrf (bzr_pwr.c:276)

power basis

conversion

```
CagdSrfStruct *CagdCnvrtPwr2BzrSrf(const CagdSrfStruct *Srf)
```

Srf: To convert into Bezier basis function representation.

Returns: Same geometry, but in the Bezier basis.

Description: Converts the given surface from Power basis functions to Bezier basis functions. Using:

```
           u -->
   +----------------------+
   |P0                Pi-1|
 v |Pi               P2i-1|  Parametric space orientation - control mesh.
 | |                      |
 v |Pn-i               Pn-1|
   +----------------------+
```

$$u^p v^q = \sum_{i=p}^{n} \sum_{j=q}^{m} \frac{\binom{i}{p}\binom{j}{q}}{\binom{n}{p}\binom{m}{q}} B_i^n(u) B_j^m(v)$$

$$C_{pq} = \sum_{i=p}^{n} \sum_{j=q}^{m} \frac{\binom{i}{p}\binom{j}{q}}{\binom{n}{p}\binom{m}{q}} A_{ij}$$

3.2.291 CagdCoerceCrvTo (cagdcoer.c:747)

coercion

```
CagdCrvStruct *CagdCoerceCrvTo(const CagdCrvStruct *Crv,
                               CagdPointType PType,
                               CagdBType AddParametrization)
```

Crv: To be coerced to a new point type PType.

PType: New point type for Crv.

AddParametrization: If TRUE, the input is a scalar curve and the requested output is 2D, add a parametrization to newly added axis.

Returns: The new, coerced to PType, curve.

Description: Coerces a curve to a new point type PType. If given curve is E1 or P1 and requested new type is E2 or P2 the Y coefficients are updated to hold the parametric domain of the curve, if AddParametrization.

3.2.292 CagdCoerceCrvsTo (cagdcoer.c:713)

coercion

```
CagdCrvStruct *CagdCoerceCrvsTo(const CagdCrvStruct *Crv,
                                CagdPointType PType,
                                CagdBType AddParametrization)
```

Crv: To be coerced to a new point type PType.

PType: New point type for Crv.

AddParametrization: If TRUE, the input is a scalar curve and the requested output is 2D, add a parametrization to newly added axis.

Returns: The new, coerced to PType, curves.

Description: Coerces a list of curves to a new point type PType. If given curves are E1 or P1 and requested new type is E2 or P2 the Y coefficients are updated to hold the parametric domain of the curve, if AddParametrization.

3.2.293 CagdCoercePointTo (cagdcoer.c:251)

coercion

```
void CagdCoercePointTo(CagdRType *NewPoint,
                       CagdPointType NewPType,
                       CagdRType * const *Points,
                       int Index,
                       CagdPointType OldPType)
```

NewPoint: Where the coerced information is to be saved.

NewPType: Point type of the coerced new point.

Points: Array of vectors if Index >= 0, a single point if Index < 0.

Index: Index into the vectors of Points.

OldPType: Point type to be expected from Points.

Returns: void

Description: Coerces Srf/Crv Point from index Index of Points array of Type PType to a new type NewPType. If however Index < 0 Points is considered single point.

3.2.294 CagdCoercePointsTo (cagdcoer.c:535)

coercion

```
void CagdCoercePointsTo(CagdRType *Points[],
                        int Len,
                        CagdPointType OldPType,
                        CagdPointType NewPType)
```

Points: Where the old and new points are placed.

Len: Length of vectors in the array of vectors, Points.

OldPType: Point type to be expected from Points.

NewPType: Point type of the coerced new point.

Returns: void

Description: Coerces an array of vectors of points of point type OldPType to point type NewPType, in place.

3.2.295 CagdCoerceSrfTo (cagdcoer.c:854)

coercion

```
CagdSrfStruct *CagdCoerceSrfTo(const CagdSrfStruct *Srf,
                               CagdPointType PType,
                               CagdBType AddParametrization)
```

Srf: To be coerced to a new point type PType.

PType: New point type for Srf.

AddParametrization: If TRUE, the input is a scalar surface and the requested output is 3D, add a parametrization to newly added axis.

Returns: The new, coerced to PType, surface.

Description: Coerces a surface to a new point type PType. If given surface is E1 or P1 and requested new type is E3 or P3 the Y and Z coefficients are updated to hold the parametric domain of the surface, if AddParametrization.

3.2.296 CagdCoerceSrfsTo (cagdcoer.c:820)

coercion

```
CagdSrfStruct *CagdCoerceSrfsTo(const CagdSrfStruct *Srf,
                                CagdPointType PType,
                                CagdBType AddParametrization)
```

Srf: To be coerced to a new point type PType.

PType: New point type for Srf.

AddParametrization: If TRUE, the input is a scalar surface and the requested output is 2D, add a parametrization to newly added axis.

Returns: The new, coerced to PType, surfaces.

Description: Coerces a list of surfaces to a new point type PType. If given surfaces are E1 or P1 and requested new type is E2 or P2 the Y coefficients are updated to hold the parametric domain of surface, if AddParametrization.

3.2.297 CagdCoerceToE2 (cagdcoer.c:32)

coercion

```
void CagdCoerceToE2(CagdRType *E2Point,
                    CagdRType * const Points[CAGD_MAX_PT_SIZE],
                    int Index,
                    CagdPointType PType)
```

E2Point: Where the coerced information is to be saved.

Points: Array of vectors if Index >= 0, a single point if Index < 0.

Index: Index into the vectors of Points.

PType: Point type to be expected from Points.

Returns: void

Description: Coerce Srf/Crv Point from index Index of Points array of Type PType to a point type E2. If however Index < 0 Points is considered single point.

3.2.298 CagdCoerceToE3 (cagdcoer.c:93)

coercion

```
void CagdCoerceToE3(CagdRType *E3Point,
                    CagdRType * const Points[CAGD_MAX_PT_SIZE],
                    int Index,
                    CagdPointType PType)
```

E3Point: Where the coerced information is to be saved.

Points: Array of vectors if Index >= 0, a single point if Index < 0.

Index: Index into the vectors of Points.

PType: Point type to be expected from Points.

Returns: void

Description: Coerce Srf/Crv Point from index Index of Points array of Type PType to a point type E3. If however Index < 0 Points is considered single point.

3.2.299 CagdCoerceToP2 (cagdcoer.c:154)

coercion

```
void CagdCoerceToP2(CagdRType *P2Point,
                    CagdRType * const Points[CAGD_MAX_PT_SIZE],
                    int Index,
                    CagdPointType PType)
```

P2Point: Where the coerced information is to be saved.

Points: Array of vectors if Index >= 0, a single point if Index < 0.

Index: Index into the vectors of Points.

PType: Point type to be expected from Points.

Returns: void

Description: Coerce Srf/Crv Point from index Index of Points array of Type PType to a point type P2. If however Index < 0 Points is considered single point.

3.2.300 CagdCoerceToP3 (cagdcoer.c:202)

coercion

```
void CagdCoerceToP3(CagdRType *P3Point,
                    CagdRType * const Points[CAGD_MAX_PT_SIZE],
                    int Index,
                    CagdPointType PType)
```

P3Point: Where the coerced information is to be saved.

Points: Array of vectors if Index >= 0, a single point if Index < 0.

Index: Index into the vectors of Points.

PType: Point type to be expected from Points.

Returns: void

Description: Coerce Srf/Crv Point from index Index of Points array of Type PType to a point type P3. If however Index < 0 Points is considered single point.

3.2.301 CagdConic2Quadric (cagd_cnc.c:1272)

```
CagdBType CagdConic2Quadric(CagdRType *A,
                            CagdRType *B,
                            CagdRType *C,
                            CagdRType *D,
                            CagdRType *E,
                            CagdRType *F,
                            CagdRType *G,
                            CagdRType *H,
                            CagdRType *I,
                            CagdRType *J)
```

A, B, C, D, E, F, G, H, I, J: Input - in A-F the conic and in J the Z height. Output - the new 10 coefficients of the quadric.

Returns: TRUE if successful, FALSE otherwise.

Description: Construct rational quadric surface above (and below) the given conic in the XY plane of Z height. The conic is given in the A-F coefficients as A x^2 + B xy + C y^2 + D x + E y + F = 0 and the quadric is returned in the A-J coefficients as: A x^2 + B y^2 + C z^2 + D xy + E xz + F yz + G x + H y + I z + J = 0.
See also: CagdCreateQuadricSrf, CagdEllipse3Points, CagdEllipse4Points, , CagdEllipseOffset,

3.2.302 CagdConicMatTransform (cagd_cnc.c:1123)

implicit

```
CagdBType CagdConicMatTransform(CagdRType *A,
                                CagdRType *B,
                                CagdRType *C,
                                CagdRType *D,
                                CagdRType *E,
                                CagdRType *F,
                                CagdMType Mat)
```

A, B, C, D, E, F: The six coefficients of the conic. Updated in place.

Mat: Transformation matrix in the XY plane.

Returns: TRUE if successful, FALSE otherwise.

Description: Transform given conic form A x^2 + B xy + C y^2 + D x + E y + F = 0, using Mat, in the XY plane. Algorithm:

```
1. Convert the implicit conic to a matrix form as:
               [ A  B/2  0  D/2] [x]
```

```
[x, y, 0, 1] [B/2  C   0  E/2] [y] = P^T M P = 0

             [ 0   0   1   0 ] [0]

             [D/2 E/2  0   F ] [1]
```

2. Compute N = Mat^{-1} the inverse of the desired transformation.

3. Compute K = N M N^T and decompose K back to A-F coefficients.

See also: CagdQuadricMatTransform, CagdSrfTransfrom, CagdCrvTransform,

3.2.303 CagdCreateConicCurve (cagd_cnc.c:57)

```
CagdCrvStruct *CagdCreateConicCurve(CagdRType A,
                                    CagdRType B,
                                    CagdRType C,
                                    CagdRType D,
                                    CagdRType E,
                                    CagdRType F,
                                    CagdRType ZLevel,
                                    CagdBType RationalEllipses)
```

A, B, C, D, E, F: The six coefficients of the conic curve.

ZLevel: Sets the Z level of this XY parallel conic curve.

RationalEllipses: TRUE for ration ellipses, FALSE for a polynomial approximation.

Returns: A quadratic curve representing the conic.

Description: Construct rational quadratic curve out of the 6 coefficients of the conic section: A x^2 + B xy + C y^2 + D x + E y + F = 0. Based on: Bezier Curves and Surface Patches on Quadrics, by Josef Hoschek, Mathematical methods in Computer aided Geometric Design II, Tom Lyche and Larry L. Schumaker (eds.), pp 331-342, 1992.

See also: BspCrvCreateCircle, BspCrvCreateUnitCircle, CagdCrvCreateArc, , BzrCrvCreateArc, CagdCreateConicCurve2, CagdCreateQuadricSrf,

3.2.304 CagdCreateConicCurve2 (cagd_cnc.c:226)

```
CagdCrvStruct *CagdCreateConicCurve2(CagdRType A,
                                     CagdRType B,
                                     CagdRType C,
                                     CagdRType D,
                                     CagdRType E,
                                     CagdRType F,
                                     CagdRType ZLevel,
                                     const CagdRType *PStartXY,
                                     const CagdRType *PEndXY,
                                     CagdBType RationalEllipses)
```

A, B, C, D, E, F: The six coefficients of the conic curve.

ZLevel: Sets the Z level of this XY parallel conic curve.

PStartXY, PEndXY: Domain of conic section - starting/end points, in the XY plane. If NULL, the most complete conic possible is created.

RationalEllipses: TRUE for rational ellipses (if full ellipse), FALSE for a polynomial approximation.

Returns: A quadratic curve representing the conic.

Description: Construct rational quadratic curve out of the 6 coefficients of the conic section: A x^2 + B xy + C y^2 + D x + E y + F = 0.

See also: BspCrvCreateCircle, BspCrvCreateUnitCircle, CagdCrvCreateArc, , BzrCrvCreateArc, CagdCreateConicCurve, CagdCreateQuadricSrf,

3.2.305 CagdCreateConicCurveSingular (cagd_cnc.c:466)

```
CagdCrvStruct *CagdCreateConicCurveSingular(CagdRType A,
                                            CagdRType B,
                                            CagdRType C,
                                            CagdRType D,
                                            CagdRType E,
                                            CagdRType F,
                                            CagdRType ZLevel)
```

A, B, C, D, E, F: The six coefficients of the singular conic curve.

ZLevel: Sets the Z level of this XY parallel conic curve.

Returns: A line/list of lines representing the singular conic.

Description: Handles construction of singular conics, when the conic degenerates into line/2lines out of the 6 coefficients: A x^2 + B xy + C y^2 + D x + E y + F = 0. The conic is singular if the following 3x3 determinant vanishes:

```
|   A      0.5*B   0.5*D  |
| 0.5*B      C     0.5*E  |
| 0.5*D    0.5*E    F     |
```

See also http://mathworld.wolfram.com/QuadraticCurve.html.
See also: CagdCreateConicCurve,

3.2.306 CagdCreateQuadricSrf (cagd_cnc.c:1339)

```
CagdSrfStruct *CagdCreateQuadricSrf(CagdRType A,
                                    CagdRType B,
                                    CagdRType C,
                                    CagdRType D,
                                    CagdRType E,
                                    CagdRType F,
                                    CagdRType G,
                                    CagdRType H,
                                    CagdRType I,
                                    CagdRType J)
```

A, B, C, D, E, F, G, H, I, J: The ten coefficients of the quadric.

Returns: A quadric surface representing the given form.

Description: Construct rational quadric surface out of the 9 coefficients of: A x^2 + B y^2 + C z^2 + D xy + E xz + F yz + G x + H y + I z + J = 0. Based on: Bezier Curves and Surface Patches on Quadrics, by Josef Hoschek, Mathematical methods in Computer aided Geometric Design II, Tom Lyche and Larry L. Schumaker (eds.), pp 331-342, 1992.
See also: BspCrvCreateCircle, BspCrvCreateUnitCircle, CagdCrvCreateArc, , BzrCrvCreateArc, CagdCreate-ConicCurve, CagdCreateConicCurve2, , CagdConic2Quadric,

3.2.307 CagdCrv2CtrlPoly (cagdmesh.c:24)

control polygon

```
CagdPolylineStruct *CagdCrv2CtrlPoly(const CagdCrvStruct *Crv)
```

Crv: To extract a control polygon from.

Returns: The control polygon of Crv.

Description: Extracts the control polygon of a curve as a polyline in E3.

3.2.308 CagdCrv2DNormalField (cagd_aux.c:908)

derivatives

normal field

```
CagdCrvStruct *CagdCrv2DNormalField(const CagdCrvStruct *Crv)
```

Crv: To compute a normal field for. This normal field is well defined at inflection points and is not flipped there.

Returns: Resulting normal field.

Description: Given a curve assumed to be planar, computes a normal field for the curve by rotating the tangent field 90 degrees.
See also: CagdCrvDerive, CagdCrvDeriveScalar,

3.2.309 CagdCrv2Polyline (bsp2poly.c:1016)

piecewise linear approximation

polyline

```
CagdPolylineStruct *CagdCrv2Polyline(const CagdCrvStruct *Crv,
                                     int SamplesPerCurve,
                                     CagdBType OptiLin)
```

Crv: To approximate as a polyline.

SamplesPerCurve: Number of samples to compute on polyline.

OptiLin: If TRUE, optimize linear curves.

Returns: A polyline representing the piecewise linear approximation from, or NULL in case of an error.

Description: Routine to approx. a single curve as a polyline with TolSamples samples/tolerance. Polyline is always E3 CagdPolylineStruct type. NULL is returned in case of an error, otherwise CagdPolylineStruct.
See also: BspCrv2Polyline, BzrCrv2Polyline, IritCurve2Polylines,

3.2.310 CagdCrvArcLenPoly (cagdcmrg.c:1530)

arc length

```
CagdRType CagdCrvArcLenPoly(const CagdCrvStruct *Crv)
```

Crv: To bound its length.

Returns: An upper bound on the curve Crv length as the length of Crv's control polygon.

Description: Computes a bound on the arc length of a curve by computing the length of its control polygon.
See also: CagdLimitCrvArcLen, CagdSrfAvgArgLenMesh, CagdCrvAreaPoly,

3.2.311 CagdCrvAreaPoly (cagdcmrg.c:1613)

```
CagdRType CagdCrvAreaPoly(const CagdCrvStruct *Crv)
```

Crv: Curve to compute the XY area below (int Y) the control polygon.

Returns: The signed area.

Description: Given a curve, returns the signed XY area below (in Y) the control polygon of the curve. For a closed control polygon, it will equal the XY area in the control polygon.
See also: CagdCrvArcLenPoly,

3.2.312 CagdCrvAverageValue (cagdbbox.c:864)

bbox

bounding box

minimum

maximum

```
CagdRType CagdCrvAverageValue(const CagdCrvStruct *Crv, int Axis)
```

Crv: To compute an average value of its control mesh, in some Axis.

Axis: 1 for X, 2 for Y etc.

Returns: Average value.

Description: Computes an average value of all control points of given curve in a given axis. Rational values are taken into account (projected into Euclidean space first).
See also: CagdCrvMinMax, CagdPointsBBox, CagdSrfAverageValue,

3.2.313 CagdCrvBBox (cagdbbox.c:67)

bbox

bounding box

```
CagdBBoxStruct *CagdCrvBBox(const CagdCrvStruct *Crv, CagdBBoxStruct *BBox)
```

Crv: To compute a bounding box for.

BBox: Where bounding information is to be saved.

Returns: BBox.

Description: Computes a bounding box for a freeform curve.
See also: CagdCrvListBBox, CagdSrfBBox, GMBBSetBBoxPrecise, , CagdIgnoreNonPosWeightBBox, CagdPolygonBBox, CagdCrvIsConstant,

3.2.314 CagdCrvBiNormalMalloc (cagd_aux.c:3302)

binormal

```
CagdVecStruct *CagdCrvBiNormalMalloc(const CagdCrvStruct *Crv,
                                     CagdRType t,
                                     CagdBType Normalize)
```

Crv: To compute (unit) binormal vector for.

t: Location where to evaluate the binormal of Crv.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, length is a function of given parametrization.

Returns: A pointer to a vector holding the unit binormal information.

Description: Given a curve Crv and a parameter value t, returns the (unit) binormal direction of Crv at t.

3.2.315 CagdCrvBiNormalToData (cagd_aux.c:3266)

binormal

```
CagdVecStruct *CagdCrvBiNormalToData(const CagdCrvStruct *Crv,
                                     CagdRType t,
                                     CagdBType Normalize,
                                     CagdVecStruct *Vec)
```

Crv: To compute (unit) binormal vector for.

t: Location where to evaluate the binormal of Crv.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, length is a function of given parametrization.

Vec: A pointer to a vector holding the unit binormal information.

Returns: A pointer to a vector holding the unit binormal information.

Description: Given a curve Crv and a parameter value t, returns the (unit) binormal direction of Crv at t.

3.2.316 CagdCrvBlossomDegreeRaise (blossom.c:1198)

```
CagdCrvStruct *CagdCrvBlossomDegreeRaise(const CagdCrvStruct *Crv)
```

Crv: Curve to degree raise.

Returns: Degree raised curve, or NULL if error.

Description: Computes a new curve with its degree raised once, given curve Crv, using blossoming.
See also: CagdCrvBlossomDegreeRaiseN, BspCrvDegreeRaise, BzrCrvDegreeRaise, CagdCrvDegreeRaise, CagdSrfBlossomDegreeRaise,

3.2.317 CagdCrvBlossomDegreeRaiseN (blossom.c:1087)

```
CagdCrvStruct *CagdCrvBlossomDegreeRaiseN(const CagdCrvStruct *Crv,
                                          int NewOrder)
```

Crv: Curve to degree raise.

NewOrder: New desired order of curve Crv.

Returns: Degree raised curve, or NULL if error.

Description: Computes a new curve with its degree raised to NewOrder, given curve Crv, using blossoming.
See also: CagdCrvBlossomDegreeRaise, BspCrvDegreeRaise, BzrCrvDegreeRaise, CagdCrvDegreeRaise, CagdSrfBlossomDegreeRaiseN,

3.2.318 CagdCrvBlossomEvalMalloc (blossom.c:633)

```
CagdRType *CagdCrvBlossomEvalMalloc(const CagdCrvStruct *Crv,
                                    const CagdRType *BlsmVals,
                                    int BlsmLen)
```

Crv: Curve to blossom.

BlsmVals: Blossoming values to consider.

BlsmLen: Length of BlsmVals vector; assumed less than curve order!

Returns: Evaluated Blossom.

Description: Computes the Blossom over the given curve, Crv, and Blossoming factors BlsmVals.
See also: CagdSrfBlossomEval, CagdSrfBlossomEvalU, CagdSrfBlossomEvalToData,

3.2.319 CagdCrvBlossomEvalToData (blossom.c:580)

```
CagdRType *CagdCrvBlossomEvalToData(const CagdCrvStruct *Crv,
                                    const CagdRType *BlsmVals,
                                    int BlsmLen,
                                    CagdRType *BlossomVals,
                                    void *BlsmCache)
```

Crv: Curve to blossom.

BlsmVals: Blossoming values to consider.

BlsmLen: Length of BlsmVals vector; assumed less than curve order!

BlossomVals: Evaluated Blossom will be saved here.

BlsmCache: Optional cache (can be NULL) to use, as allocated by CagdBlsmEvalSymbAllocCache and freed by CagdBlsmEvalSymbFreeCache.

Returns: Evaluated Blossom.

Description: Computes the Blossom over the given curve, Crv, and Blossoming factors BlsmVals.
See also: CagdSrfBlossomEval, CagdSrfBlossomEvalU,

3.2.320 CagdCrvCopy (cagd1gen.c:747)

copy

```
CagdCrvStruct *CagdCrvCopy(const CagdCrvStruct *Crv)
```

Crv: To be copied.

Returns: A duplicate of Crv.

Description: Allocates and copies all slots of a curve structure.

3.2.321 CagdCrvCopyList (cagd1gen.c:1170)

copy

```
CagdCrvStruct *CagdCrvCopyList(const CagdCrvStruct *CrvList)
```

CrvList: To be copied.

Returns: A duplicated list of curves.

Description: Allocates and copies a list of curve structures.

3.2.322 CagdCrvCreateArc (cagd_arc.c:137)

circle

arc

```
CagdCrvStruct *CagdCrvCreateArc(const CagdPtStruct *Center,
                                CagdRType Radius,
                                CagdRType StartAngle,
                                CagdRType EndAngle)
```

Center: Point of arc.

Radius: Of arc.

StartAngle: Starting angle of arc, in degrees.

EndAngle: End angle of arc, in degrees.

Returns: A rational quadratic Bezier (or Bspline) curve representing the arc.

Description: Creates an arc at the specified position as a rational quadratic Bspline curve, with upto two Bezier pieces. The arc is defined from StartAngle to EndAngle counter clockwise, and is assumed to be less than 360 degrees from Start to End.

See also: BspCrvCreateCircle, BspCrvCreateUnitCircle, BspCrvCreatePCircle, , CagdCreateConicCurve, BzrCrvCreateArc, BspCrvCreateUnitPCircle, , CagdCrvCreateArcCCW, CagdCrvCreateArcCW,

3.2.323 CagdCrvCreateArcCCW (cagd_arc.c:203)

circle

arc

```
CagdCrvStruct *CagdCrvCreateArcCCW(const CagdPtStruct *Start,
                                   const CagdPtStruct *Center,
                                   const CagdPtStruct *End)
```

Start: Starting position of arc.

Center: Center point of arc.

End: End position of arc.

Returns: A rational quadratic Bezier (or Bspline) curve representing the arc, or NULL if error.

Description: Creates a counter clockwise arc at the specified position as a rational quadratic Bspline curve, with up to two Bezier pieces. The arc is defined from Start to End, and is assumed to be less than or eual to 360 degrees (full circle, if Start == End).

See also: BspCrvCreateCircle, BspCrvCreateUnitCircle, BspCrvCreatePCircle, , CagdCreateConicCurve, BzrCrvCreateArc, BspCrvCreateUnitPCircle, , CagdCrvCreateArc, CagdCrvCreateArcCW,

3.2.324 CagdCrvCreateArcCW (cagd_arc.c:275)

circle

arc

```
CagdCrvStruct *CagdCrvCreateArcCW(const CagdPtStruct *Start,
                                  const CagdPtStruct *Center,
                                  const CagdPtStruct *End)
```

Start: Starting position of arc.

Center: Center point of arc.

End: End position of arc.

Returns: A rational quadratic Bezier (or B-spline) curve representing the arc, or NULL if error.

Description: Creates a counter clockwise arc at the specified position as a rational quadratic B-spline curve, with up to two Bezier pieces. The arc is defined from Start to End, and is assumed to be less than or equal to 360 degrees (full circle, if Start == End).

See also: BspCrvCreateCircle, BspCrvCreateUnitCircle, BspCrvCreatePCircle, , CagdCreateConicCurve, BzrCrvCreateArc, BspCrvCreateUnitPCircle, , CagdCrvCreateArc, CagdCrvCreateArcCCW,

3.2.325 CagdCrvCrvInter (cagd_cci.c:397)

cci

```
CagdPtStruct *CagdCrvCrvInter(const CagdCrvStruct *Crv1,
                              const CagdCrvStruct *Crv2,
                              CagdRType Eps)
```

Crv1, Crv2: Two curves to compute their intersection points.

Eps: Accuracy of computation.

Returns: List of intersection points. Each points would contain (u1, u2, 0.0).

Description: Computes the intersection points, if any, of the two given curves.
See also: CagdCrvTanAngularSpan, SymbCrvCrvInter, UserSrfSrfInter, , CagdCrvCrvInterArrangment,

3.2.326 CagdCrvCrvInterArrangment (cagd_cci.c:838)

cci

```
CagdCrvStruct *CagdCrvCrvInterArrangment(const CagdCrvStruct *ArngCrvs,
                                         CagdBType SplitCrvs,
                                         CagdRType Eps)
```

ArngCrvs: Curves to intersect.

SplitCrvs: TRUE, to also split the curves at detected intersections.

Eps: Tolerance of CCI computations.

Returns: (Sub) curves as split due to cci's, or identical curves with "InterPts" attributes.

Description: Computes the intersections in the plane between all given curves and optionally split the curves at those parameters. If the curves are not split, a list of intersection parameters is returned in an attribute called "InterPts", holding the parameter in X coordinate of pts.
See also: CagdCrvCrvInter, SymbCrvCrvInter, CagdCrvsLowerEnvelop,

3.2.327 CagdCrvCrvMakeJoinMatch (cagdcmrg.c:661)

match

```
void CagdCrvCrvMakeJoinMatch(CagdCrvStruct **Crv1,
                             CagdCrvStruct **Crv2,
                             IrtRType Tolerance,
                             CagdBType G1Continuity,
                             CagdBType ClosedLoop)
```

Crv1: To modify its end to meet Crv2's starting location, in place.

Crv2: To modify its end to meet Crv2's starting location, in place.

Tolerance: To consider two end points the same.

G1Continuity: If TRUE, ensure G1 continuity.

ClosedLoop: If TRUE, try matching the curves at both endpoints.

Returns: void

Description: If Crv1 and Crv2 share an endpoint (up to Tolerance), then the shared endpoint of each curve is updated to the average of both points, so the two curves will meet precisecely. Crv1 and Crv2 will be reversed as needed so the last point of Crv1 will be the same as the first point of Crv2. If G1Continuity is TRUE, then the one but last and second endpoints will be also modified to ensure G1 continuity. If ClosedLoop is TRUE and the curves share two endpoints, then they will be matched at both of these endpoints, making them a closed loop.
See also: CagdCrvListMakeJoinMatch,

3.2.328 CagdCrvDegreeRaise (cagd_aux.c:2312)

degree raising

```
CagdCrvStruct *CagdCrvDegreeRaise(const CagdCrvStruct *Crv)
```

Crv: To raise its degree.

Returns: A curve with same geometry as Crv but with one degree higher.

Description: Returns a new curve representing the same curve as Crv but with its degree raised by one.

3.2.329 CagdCrvDegreeRaiseN (cagd_aux.c:2374)

degree raising

```
CagdCrvStruct *CagdCrvDegreeRaiseN(const CagdCrvStruct *Crv, int NewOrder)
```

Crv: To raise its degree.

NewOrder: Expected new order of the raised curve.

Returns: A curve with same geometry as Crv but with order that is equal to NewOrder.

Description: Returns a new curve representing the same curve as Crv but with its degree raised to NewOrder

3.2.330 CagdCrvDegreeReduce (cagd_aux.c:2342)

degree raising

```
CagdCrvStruct *CagdCrvDegreeReduce(const CagdCrvStruct *Crv)
```

Crv: To raise its degree.

Returns: A curve with same geometry as Crv but with one degree higher.

Description: Returns a new curve representing the same curve as Crv but with its degree raised by one.

3.2.331 CagdCrvDeletePoint (cagdedit.c:150)

```
CagdCrvStruct *CagdCrvDeletePoint(const CagdCrvStruct *Crv, int Index)
```

Crv: Input curve to delete a point from.

Index: Index of control point to delete from Crv.

Returns: A new curve of length smaller by one than Crv.

Description: Delete a point at Index from curve Crv. Returned curve's length is smaller by one than the length of Crv. Knot vector is updated (if Bspline curve) to a uniform open. Order of curve is reduced if greater than new number of control points.
See also: CagdCrvInsertPoint,

3.2.332 CagdCrvDerive (cagd_aux.c:738)

derivatives

Hodograph

```
CagdCrvStruct *CagdCrvDerive(const CagdCrvStruct *Crv)
```

Crv: To compute its Hodograph curve.

Returns: Resulting hodograph.

Description: Given a curve, computes its derivative curve (Hodograph).
See also: BzrCrvDerive, BspCrvDerive, SymbCrvDeriveRational, , CagdCrvDeriveScalar, CagdCrvScalarCrvSlopeBounds,

3.2.333 CagdCrvDeriveScalar (cagd_aux.c:773)

derivatives

Hodograph

```
CagdCrvStruct *CagdCrvDeriveScalar(const CagdCrvStruct *Crv)
```

Crv: To compute derivatives of all its components.

Returns: Resulting derivative.

Description: Given a curve, computes the derivative of all is scalar components. For a Euclidean curve this is the same as CagdCrvDerive but for a rational curve the returned curves is not the vector field but simply the derivatives of all the curve's coefficients, including the weights.
See also: BzrCrvDerive, BspCrvDerive, SymbCrvDeriveRational, , CagdCrvDerive, BzrCrvDeriveScalar, BspCrvDeriveScalar,

3.2.334 CagdCrvDomain (cagd_aux.c:42)

domain

parametric domain

```
void CagdCrvDomain(const CagdCrvStruct *Crv, CagdRType *TMin, CagdRType *TMax)
```

Crv: To get its parametric domain.

TMin: Where to put the minimal domain's boundary.

TMax: Where to put the maximal domain's boundary.

Returns: void

Description: Returns the parametric domain of a curve.
See also: BspCrvDomain,

3.2.335 CagdCrvEvalEndPtsE3 (cagd_aux.c:206)

evaluation

```
void CagdCrvEvalEndPtsE3(const CagdCrvStruct *Crv,
                         CagdPType Start,
                         CagdPType End)
```

Crv: To evaluate at the end points.

Start: To be updated with the starting point of the curve.

End: To be updated with the end point of the curve.

Returns: void

Description: Given a curve, evaluate its two end points and coerce them to E3.
See also: CagdCrvEval2Data,

3.2.336 CagdCrvEvalMalloc (cagd_aux.c:176)

evaluation

```
CagdRType *CagdCrvEvalMalloc(const CagdCrvStruct *Crv, CagdRType t)
```

Crv: To evaluate at the given parametric location t.

t: The parameter value at which the curve Crv is to be evaluated.

Returns: A vector holding all the coefficients of all components of curve Crv's point type. If for example the curve's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1). This vector is allocated dynamically.

Description: Given a curve and parameter value t, evaluate the curve at t.
See also: CagdCrvEval2Data,

3.2.337 CagdCrvEvalToData (cagd_aux.c:134)

evaluation

```
void CagdCrvEvalToData(const CagdCrvStruct *Crv, CagdRType t, CagdRType *R)
```

Crv: To evaluate at the given parametric location t.

t: The parameter value at which the curve Crv is to be evaluated.

R: A vector holding all the coefficients of all components of curve Crv's point type. If for example the curve's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Returns: void

Description: Given a curve and parameter value t, evaluate the curve at t.
See also: BzrCrvEvalAtParamMalloc, BspCrvEvalAtParamMalloc, , BzrCrvEvalVecAtParamMalloc, BspCrvEvalVecAtParamMalloc, , BspCrvEvalCoxDeBoorMalloc, CagdCrvEvalToPolylineMalloc,

3.2.338 CagdCrvEvalToPolyline (cbspeval.c:176)

conversion
refinement
evaluation

```
int CagdCrvEvalToPolyline(const CagdCrvStruct *Crv,
                          int FineNess,
                          CagdRType *Points[],
                          BspKnotAlphaCoeffStruct *A,
                          CagdBType OptiLin)
```

Crv: To approximate as a polyline.

FineNess: Control on number of samples.

Points: Where to put the resulting polyline. Assumed to be valid with respect to the dimension of Crv.

A: Optional alpha matrix for refinement.

OptiLin: If TRUE, optimize linear curves.

Returns: The actual number of samples placed in Points. Always less than or equal to FineNess.

Description: Samples the curve at FineNess location equally spaced in the curve's parametric domain. Computes a refinement alpha matrix (If FineNess > 0), refines the curve and uses refined control polygon as the approximation to the curve. If FineNess == 0, Alpha matrix A is used instead. Returns the actual number of points in polyline (<= FineNess). Note this routine may be invoked with Bezier curves as well as B-spline.
See also: BzrCrvEvalToPolyline, AfdBzrCrvEvalToPolyline, CagdCrvEval,

3.2.339 CagdCrvFirstMoments (cbsp_int.c:1651)

```
void CagdCrvFirstMoments(const CagdCrvStruct *Crv,
                         int n,
                         CagdPType Loc,
                         CagdVType Dir)
```

Crv: To compute zero and first moment.

n: Number of samples the curve should be sampled at.

Loc: Center of curve as zero moment.

Dir: Main direction of curve as first moment.

Returns: void

Description: Computes zero and first moments of a curve.

3.2.340 CagdCrvFree (cagd2gen.c:186)

free

```
void CagdCrvFree(CagdCrvStruct *Crv)
```

Crv: To be deallocated.

Returns: void

Description: Deallocates and frees all slots of a curve structure.

3.2.341 CagdCrvFreeList (cagd2gen.c:237)

free

```
void CagdCrvFreeList(CagdCrvStruct *CrvList)
```

CrvList: To be deallocated.

Returns: void

Description: Deallocates and frees a curve structure list:

3.2.342 CagdCrvFromMesh (cagd_aux.c:2910)

isoparametric curves

curve from mesh

```
CagdCrvStruct *CagdCrvFromMesh(const CagdSrfStruct *Srf,
                               int Index,
                               CagdSrfDirType Dir)
```

Srf: To extract a curve from.

Index: Index along the mesh of Srf to extract the curve from.

Dir: Direction of extracted curve. Either U or V.

Returns: A curve from Srf. This curve inherit the order and continuity of surface Srf in direction Dir. However, this curve is not on surface Srf, in general.

Description: Extracts a curve from the mesh of surface Srf in direction Dir at index Index.
See also: CagdCrvFromSrf, BzrSrfCrvFromSrf, BspSrfCrvFromSrf, BzrSrfCrvFromMesh, , BspSrfCrvFromMesh, CagdCrvToMesh,

3.2.343 CagdCrvFromSrf (cagd_aux.c:2759)

isoparametric curves

curve from surface

```
CagdCrvStruct *CagdCrvFromSrf(const CagdSrfStruct *Srf,
                              CagdRType t,
                              CagdSrfDirType Dir)
```

Srf: To extract an isoparametric curve from.

t: Parameter value of extracted isoparametric curve.

Dir: Direction of extracted isoparametric curve. Either U or V.

Returns: An isoparametric curve of Srf. This curve inherit the order and continuity of surface Srf in direction Dir.

Description: Extracts an isoparametric curve from the surface Srf in direction Dir at the parameter value of t.
See also: BzrSrfCrvFromSrf, BspSrfCrvFromSrf, CagdCrvFromMesh, BzrSrfCrvFromMesh, , BspSrfCrvFromMesh, TrngCrvFromTriSrf,

3.2.344 CagdCrvIncCnstrct (crv_inc_cnstrct.c:662)

```
CagdCrvStruct *CagdCrvIncCnstrct(CagdCrvIncCnstrctType Op,
                                 CagdRType *Params,
                                 CagdCrvIncCnstrctStruct **CrvIncCnstrct)
```

Op: Controls the new incremental operation. If new information is relative to previous location and no previous location was given, zero is assumed. All angles are in degrees. The different operations are CAGD_CRV_INC_CNSTRCT_INIT - Initialized the incremental construction. No parameters are expected. CAGD_CRV_INC_CNSTRCT_XY - Appends a new XY location. Expects (X, Y) as Params. CAGD_CRV_INC_CNSTRCT_X - Appends a new XY location. Expects (X) as Params (Y is same as last point). CAGD_CRV_INC_CNSTRCT_Y - Appends a new XY location. Expects (Y) as Params (X is same as last point). CAGD_CRV_INC_CNSTRCT_DELTA_XY - Appends a new XY location. Expects (DeltaX, DeltaY) as Params, as relative locations to previous location. CAGD_CRV_INC_CNSTRCT_DELTA_X - Appends a new XY location. Expects (DeltaX) as Params, as relative location to previous X location. Y is same as last point. CAGD_CRV_INC_CNSTRCT_DELTA_Y - Appends a new XY location. Expects (DeltaY) as Params, as relative location to previous Y location. X is same as last point. CAGD_CRV_INC_CNSTRCT_DELTA_X_AND_Y - Appends a new XY location. Expects (DeltaX, Y) as Params. A new Y location and a relative X location to previous X. CAGD_CRV_INC_CNSTRCT_DELTA_Y_AND_X - Appends a new XY location. Expects (X, DeltaY) as Params. A new X location and a relative Y location to previous Y. CAGD_CRV_INC_CNSTRCT_DIST_ALPHA - Appends a new XY location. Expects (d, Alpha) as Params. A new XY location relative to last location in polar coordinates as (d cos(Alpha), d sin(Alpha)). CAGD_CRV_INC_CNSTRCT_DIST_DELTA_ALPHA - Appends a new XY location. Expects (d, DeltaAlpha) as Params. A new XY location relative to last location in polar coordinates as (d cos(LastAlpha + DeltaAlpha), d sin(LastAlpha + DeltaAlpha). CAGD_CRV_INC_CNSTRCT_DELTA_X_DELTA_ALPH - Appends a new XY location. Expects (DeltaX, DeltaAlpha) as Params. Seeks a new XY location in direction LastAlpha + DeltaAlpha until it intersects the line X = LastX + DeltaX. CAGD_CRV_INC_CNSTRCT_DELTA_Y_D - Appends a new XY location. Expects (DeltaY, DeltaAlpha) as Params. Seeks a new XY location in direction LastAlpha + DeltaAlpha until it intersects the line Y = LastY + DeltaY. CAGD_CRV_INC_CNSTRCT_ALPHA_LIN

- Appends a new XY location. Expects (Alpha, X2, Y2, Alpha2) as Params. Seeks a new XY location in direction Alpha from current location at the intersection with the line defined by (X, Y) in direction Alpha2. CAGD_CRV_INC_CNSTRCT_ARC - Modifies the last XY location. Expects (R, F) as Params. Replaces the last segment with an arc that ends at the two end points of the segment. Positive R for an arc on the left, Negative on the right. If F is non zero, the other left/right arc is provided (note there are four ways to connect two points with C/CW arcs). CAGD_CRV_INC_CNSTRCT_ADD_INTERMEDIATE - Modifies last XY location. Expects (A, D, W) as Params. Adds interior (control) points to the last segment. A is between zero and one setting the distance along the segment (A = 0 starting point, 1 end point). D is the signed distance (positive for the left size) orthogonal to the line segment. Finally W sets the weight of the new (control) points. This feature enable the creation of arbitrary degree spline curves. if all W are 1.0, a polynomial curve is created. CAGD_CRV_INC_CNSTRCT_ROUND_LAST - Rounds the last entered point. Expects (R) as the parameter, the radius of the round. CAGD_CRV_INC_CNSTRCT_CHAMFER_LAST - Chamfers the last entered point. Expects (C) as the parameter, the size of the chamfer, that linearly connects the two tangency points of a rounding of size C. CAGD_CRV_INC_CNSTRCT_CURVE_PARAMS - Sets default curve parameters. Expects (d, KV) as Params. The parameter d sets the degree of all fore coming constructed non-linear curves (has internal control point) and KV sets the knot sequence to use which is one of uniform open (0), uniform float (1), or uniform periodic (2). CAGD_CRV_INC_CNSTRCT_CLOSE - Terminate the constriction. Expects (P, d) as Params. If P is negative the shape is closed to the initial location. If P is positive an open curve will be created. If P = +/-2 all sharp corners are going to be chamfered as controlled by d. If P = +/-3 all sharp corners will be rounded by radius d.

Params: See descriptions under Op.

CrvIncCnstrct: A handle that is used locally to hold intermediate data.

Returns: NULL, unless Op = CAGD_CRV_INC_CNSTRCT_CLOSE in which case, the created curve is returned.

Description: An incremental constructor for planar curves, inserting one entity (typically a point) at a time, as governed by Op.
See also: CagdCrvIncCnstrctPrint, CagdCrvIncCnstructSize, CagdCrvIncCnstrctList,

3.2.345 CagdCrvIncCnstrctError (crv_inc_cnstrct.c:1011)

```
const char *CagdCrvIncCnstrctError(CagdCrvIncCnstrctStruct *CrvIncCnstrct)
```

CrvIncCnstrct: An array of incremental curve construction commands.

Returns: The error string description or NULL if no error.

Description: In interface function to function CagdCrvIncCnstrct, that returns an error description if was an error. Should be called after CagdCrvIncCnstrct is invoked.
See also: CagdCrvIncCnstrctPrint, CagdCrvIncCnstructSize, CagdCrvIncCnstrct, , CagdCrvIncCnstrctList,

3.2.346 CagdCrvIncCnstrctList (crv_inc_cnstrct.c:1040)

```
CagdCrvStruct *CagdCrvIncCnstrctList(CagdCrvIncCnstrctInputStruct *CrvIncInput,
                                     char **ErrorStr)
```

CrvIncInput: An array of incremental curve construction commands.

ErrorStr: Updated to NULL if no error, or an error description if was an error (allocated dynamically).

Returns: The constructed curve.

Description: In interface function to function CagdCrvIncCnstrct, that accepts am array of incremental curve construction commands. The array must start INIT command and must terminate with a CLOSE command.
See also: CagdCrvIncCnstrctPrint, CagdCrvIncCnstructSize, CagdCrvIncCnstrct,

3.2.347 CagdCrvIncCnstrctPrint (crv_inc_cnstrct.c:511)

```
void CagdCrvIncCnstrctPrint(CagdCrvIncCnstrctStruct *CrvIncCnstrct)
```

CrvIncCnstrct: Accumulated data to print.

Returns: void

Description: Prints the current accumulated data in the incremental build.
See also: CagdCrvIncCnstrct, CagdCrvIncCnstrctList, CagdCrvIncCnstrctSize,

3.2.348 CagdCrvIncCnstrctSize (crv_inc_cnstrct.c:557)

```
int CagdCrvIncCnstrctSize(CagdCrvIncCnstrctStruct *CrvIncCnstrct)
```

CrvIncCnstrct: Current state of incrementally build curve.

Returns: Number of inserted points so far.

Description: Returns number of points inserted so far.
See also: CagdCrvIncCnstrct, CagdCrvIncCnstrctList, CagdCrvIncCnstrctPrint,

3.2.349 CagdCrvInsertPoint (cagdedit.c:94)

```
CagdCrvStruct *CagdCrvInsertPoint(const CagdCrvStruct *Crv,
                                  int Index,
                                  const CagdPType Pt)
```

Crv: Input curve to insert a new point into.

Index: Index of control point to insert into Crv. Zero inserts at first location in Crv.

Pt: New point to insert that will be coerced to Crv point type.

Returns: A new curve of length larger by one than Crv.

Description: Inserts a new point Pt at Index into curve Crv. Returned curve's length is larger by one than the length of Crv. Knot vector is updated (if Bspline curve) to a uniform open.
See also: CagdCrvDeletePoint,

3.2.350 CagdCrvIntegrate (cagd_aux.c:875)

integrals

```
CagdCrvStruct *CagdCrvIntegrate(const CagdCrvStruct *Crv)
```

Crv: To compute its integral curve.

Returns: Resulting integral curve.

Description: Given a curve, compute its integral curve.
See also: BzrCrvIntegrate, BspCrvIntegrate,

3.2.351 CagdCrvIsConstant (cagdbbox.c:138)

bbox

bounding box

```
CagdBType CagdCrvIsConstant(const CagdCrvStruct *Crv, IrtRType Eps)
```

Crv: To check if constant or not.

Eps: Tolerance of equality allowed.

Returns: TRUE if indeed a constant(s) valued curve.

Description: Checks if this curve is a constant(s) curve.
See also: CagdCrvBBox, CagdSrfIsConstant,

3.2.352 CagdCrvIsCtlPolyMonotone (cagd_aux.c:988)

```
int CagdCrvIsCtlPolyMonotone(const CagdCrvStruct *Crv, int Axis, CagdRType Eps)
```

Crv: To examine if monotone in axis Axis.

Axis: 1 for X, 2 for Y, etc.

Eps: For monotonicity epsilon test. Zero for strictly monotone.

Returns: 1 if increasingly monotone in axis Axis, -1 if decreasingly monotone and 0 otherwise.

Description: Examines if the given curve's control polygon is monotone in given Axis.

3.2.353 CagdCrvListBBox (cagdbbox.c:107)

bbox

bounding box

```
CagdBBoxStruct *CagdCrvListBBox(const CagdCrvStruct *Crvs, CagdBBoxStruct *BBox)
```

Crvs: To compute a bounding box for.

BBox: Where bounding information is to be saved.

Returns: BBox.

Description: Computes a bounding box for a list of freeform curves.
See also: CagdCrvBBox, CagdSrfBBox, GMBBSetBBoxPrecise,

3.2.354 CagdCrvListMakeJoinMatch (cagdcmrg.c:818)

match

```
CagdCrvStruct *CagdCrvListMakeJoinMatch(const CagdCrvStruct *CrvList,
                                        IrtRType Tolerance,
                                        CagdBType G1Continuity,
                                        CagdBType ClosedLoop)
```

CrvList: To match their endpoints.

Tolerance: To consider two end points the same.

G1Continuity: If TRUE, ensure G1 continuity.

ClosedLoop: If TRUE, match the last curve with the first one.

Returns: The matched curves.

Description: Modifies the given curves so each curve will share a common endpoint with its consecutive curve in the given list, given that these curves already have a pair of endpoints which are close up to Tolerance. If G1Contniuity is TRUE, then the curves are also modified so they share a tangent at their endpoints If ClosedLoop is TRUE, the last curve is also matched with the first one. The result is returned as a newly allocated list of curves.
See also: CagdCrvCrvMakeJoinMatch,

3.2.355 CagdCrvListMatTransform (cagd2gen.c:1862)

scaling

rotation

translation

transformations

```
CagdCrvStruct *CagdCrvListMatTransform(const CagdCrvStruct *Crvs,
                                       CagdMType Mat)
```

Crvs: To be transformed.

Mat: Defining the transformation.

Returns: Returned transformed curves.

Description: Applies an homogeneous transformation, to the given list of curves Crvs as specified by homogeneous transformation Mat.
See also: CagdTransform, CagdSrfMatTransform, CagdMatTransform, CagdCrvRotateToXY, , CagdCrvMatTransform,

3.2.356 CagdCrvMatTransCenter (cagd2gen.c:1313)

transformations

```
void CagdCrvMatTransCenter(CagdCrvStruct *Crv, IrtHmgnMatType Mat)
```

Crv: To be transformed around the center of the curve.

Mat: Transformation to apply.

Returns: void

Description: Applies a transform, in place, to given curve Crv as specified by Mat, around the center of the curve.
See also: CagdSrfTransform, CagdTransform, CagdCrvMatTransform, CagdCrvRotateToXY, CagdCrvTransform, CagdSrfMatTransCenter,

3.2.357 CagdCrvMatTransform (cagd2gen.c:1799)

scaling
rotation
translation
transformations

```
CagdCrvStruct *CagdCrvMatTransform(const CagdCrvStruct *Crv,
                                   CagdMType Mat)
```

Crv: To be transformed.

Mat: Defining the transformation.

Returns: Returned transformed curve.

Description: Applies an homogeneous transformation, to the given curve Crv as specified by homogeneous transformation Mat.
See also: CagdTransform, CagdSrfMatTransform, CagdMatTransform, CagdCrvRotateToXY, CagdCrvListMatTransform,

3.2.358 CagdCrvMinMax (cagdbbox.c:821)

bbox
bounding box
minimum
maximum

```
void CagdCrvMinMax(const CagdCrvStruct *Crv,
                   int Axis,
                   CagdRType *Min,
                   CagdRType *Max)
```

Crv: To test for minimum/maximum.

Axis: 0 for W, 1 for X, 2 for Y etc.

Min: Where minimum found value should be place.

Max: Where maximum found value should be place.

Returns: void

Description: Computes a min max bound on a curve in a given axis. The curve is not coerced to anything and the given axis is tested directly where 0 is the W axis and 1, 2, 3 are the X, Y, Z etc.
See also: CagdCrvAverageValue, CagdPointsBBox, CagdSrfMinMax,

3.2.359 CagdCrvMoebiusTransform (cagd_aux.c:950)

moebius transformation

```
CagdCrvStruct *CagdCrvMoebiusTransform(const CagdCrvStruct *Crv, CagdRType c)
```

Crv: To compute its moebius transformation.

c: The scaling coefficient - c^n is the ratio between the first and last weight of the curve. If c == 0, the first and last weights are made equal.

Returns: Resulting curve after the moebius transformation.

Description: Given a curve, compute its moebius transformation.
See also: BzrCrvMoebiusTransform, BspCrvMoebiusTransform,

3.2.360 CagdCrvNew (cagd1gen.c:59)

allocation

```
CagdCrvStruct *CagdCrvNew(CagdGeomType GType, CagdPointType PType, int Length)
```

GType: Type of geometry the curve should be - B-spline, Bezier etc.

PType: Type of control points (E2, P3, etc.).

Length: Number of control points

Returns: An uninitialized freeform curve.

Description: Allocates the memory required for a new curve.
See also: BzrCrvNew, BspPeriodicCrvNew, bspCrvNew, CagdPeriodicCrvNew, TrimCrvNew,

3.2.361 CagdCrvNodes (bsp_knot.c:1702)

node values

```
CagdRType *CagdCrvNodes(const CagdCrvStruct *Crv)
```

Crv: To compute node values for.

Returns: Node values of the given curve.

Description: Returns the nodes of a freeform curve.

3.2.362 CagdCrvNormalMalloc (cagd_aux.c:3369)

normal

```
CagdVecStruct *CagdCrvNormalMalloc(const CagdCrvStruct *Crv,
                                   CagdRType t,
                                   CagdBType Normalize)
```

Crv: To compute (unit) normal vector for.

t: Location where to evaluate the normal of Crv.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, length is a function of given parametrization.

Returns: A pointer to a vector holding the unit normal information.

Description: Given a curve Crv and a parameter value t, returns the (unit) normal direction of Crv at t.

3.2.363 CagdCrvNormalToData (cagd_aux.c:3333)

normal

```
CagdVecStruct *CagdCrvNormalToData(const CagdCrvStruct *Crv,
                                   CagdRType t,
                                   CagdBType Normalize,
                                   CagdVecStruct *N)
```

Crv: To compute (unit) normal vector for.

t: Location where to evaluate the normal of Crv.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, length is a function of given parametrization.

N: A pointer to a vector holding the unit normal information.

Returns: A pointer to a vector holding the unit normal information.

Description: Given a curve Crv and a parameter value t, returns the (unit) normal direction of Crv at t.

3.2.364 CagdCrvNormalXYToData (cagd_aux.c:3402)

normal

```
CagdVecStruct *CagdCrvNormalXYToData(const CagdCrvStruct *Crv,
                                     CagdRType t,
                                     CagdBType Normalize,
                                     CagdVecStruct *Vec)
```

Crv: To compute (unit) normal vector for.

t: Location where to evaluate the normal of Crv.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, length is a function of given parametrization.

Vec: A pointer to a vector holding the unit normal information.

Returns: A pointer to a vector holding the unit normal information.

Description: Given a curve Crv and a parameter value t, returns the (unit) normal direction of Crv at t, that is consistent over inflection points. That is, this normal is not flipped over inflection points and is always 90 rotation from the tangent vector. Needless to say, this function is for two dimensional palanr curves.

3.2.365 CagdCrvOnOneSideOfLine (cagd_aux.c:3925)

```
CagdBType CagdCrvOnOneSideOfLine(const CagdCrvStruct *Crv,
                                 CagdRType X1,
                                 CagdRType Y1,
                                 CagdRType X2,
                                 CagdRType Y2)
```

Crv: Curve to examine if totally on one side of the line, in XY plane.

X1, Y1: First points defining the line in the XY plane.

X2, Y2: First points defining the line in the XY plane.

Returns: TRUE if Crv totlally on one side of the line between (X1, Y1) and (X2, Y2), in the XY plane, or FALSE otherwise.

Description: Examines if the given curve is totally on one side of the line prescribed by points (X1, Y1) and (X2, Y2), in the XY plane.

3.2.366 CagdCrvOrientationFrame (cagdswep.c:556)

orientation frame

sweep

surface constructors

```
CagdBType CagdCrvOrientationFrame(const CagdCrvStruct *Crv,
                                  CagdRType CrntT,
                                  CagdVecStruct *Tangent,
                                  CagdVecStruct *Normal,
                                  CagdVecStruct **MallocedNrmls,
                                  int *MallocedNrmlsVecSize,
                                  CagdBType FirstTime)
```

Crv: Curve to evaluate orientation frame for.

CrntT: Parameter value where to evaluate.

Tangent: Of curve at parameter value t.

Normal: Of curve at parameter value t.

MallocedNrmls: Vectors of malloced samples will be created and saved here if first time and will be used if !FirstTime.

MallocedNrmlsVecSize: Size of malloced vector will be saved here if first time and will be used if !FirstTime.

FirstTime: TRUE if first time to compute a frame for this curve.

Returns: TRUE if computed, FALSE if error/failed.

Description: Estimates an orientation frame (tangent and normal) for the given curve at the given parameter t.

3.2.367 CagdCrvQuadDirectInterp (cbsp_int.c:2301)

```
CagdCrvStruct *CagdCrvQuadDirectInterp(const CagdCrvStruct *Crv,
                                       CagdRType InflectStretch)
```

Crv: To interpolate its control points. Assumed to have at least 3 control points, with adjacent control points that are not collinear.

InflectStretch: Factor to affect the stretch in inflection points. 1.0 is a good start.

Returns: Interpolating curve, or NULL if error.

Description: Constructs a quadratic B-spline curve that interpolates the given control points of the input planar (in XY) curve. The quadratic B-spline interpolating curve is constructed directly by adjusting knot spacing. No linear system is solved!

See also:

3.2.368 CagdCrvQuadTileAssumeSrf (crv2quad.c:401)

curve tiling

```
CagdCrvQuadTileStruct *CagdCrvQuadTileAssumeSrf(CagdSrfStruct *Srf)
```

Srf: The surface from which to create the quad tile.

Returns: A quad tile structure containing the surface.

Description: Creates a quad tile structure from an existing surface. In this function, the surface pointer is taken into the quad tile structure, and therefore SHOULD NOT BE FREED EXTERNALLY.
See also: CagdCrvQuadTileFromSrf,

3.2.369 CagdCrvQuadTileCopy (crv2quad.c:171)

copy

```
CagdCrvQuadTileStruct *CagdCrvQuadTileCopy(const CagdCrvQuadTileStruct *Tile)
```

Tile: To be copied.

Returns: A duplicate of Tile.

Description: Allocates and copies a quad tile structure.

3.2.370 CagdCrvQuadTileCopyList (crv2quad.c:201)

copy

```
CagdCrvQuadTileStruct *CagdCrvQuadTileCopyList(
                                    const CagdCrvQuadTileStruct *Tiles)
```

Tiles: To be copied.

Returns: A duplicated list of quad tiles.

Description: Allocates and copies a list of quad tile structures.

3.2.371 CagdCrvQuadTileFromSrf (crv2quad.c:462)

curve tiling

```
CagdCrvQuadTileStruct *CagdCrvQuadTileFromSrf(const CagdSrfStruct *Srf)
```

Srf: The surface from which to create the quad tile.

Returns: A quad tile structure containing the surface.

Description: Creates a quad tile structure from an existing surface. In this function, the surface is copied into the quad tile structure.
See also: CagdCrvQuadTileAssumeSrf,

3.2.372 CagdCrvRefineAtParams (cagd_aux.c:1789)

refinement

subdivision

```
CagdCrvStruct *CagdCrvRefineAtParams(const CagdCrvStruct *Crv,
                                     CagdBType Replace,
                                     CagdRType *t,
                                     int n)
```

Crv: To refine.

Replace: If TRUE, t holds knots in exactly the same length as the length of the knot vector of Crv and t simply replaces the knot vector.

t: Vector of knots with length of n.

n: Length of vector t.

Returns: A refined curve of Crv after insertion of all the knots as specified by vector t of length n.

Description: Given a curve - refines it at the given n knots as defined by vector t. If Replace is TRUE, the values in t replaces current knot vector. Returns pointer to refined surface (Note a Bezier curve will be converted into a B-spline curve).

3.2.373 CagdCrvRefineUniformly (cagd_aux.c:1828)

```
CagdCrvStruct *CagdCrvRefineUniformly(const CagdCrvStruct *Crv, int RefLevel)
```

Crv: To refine.

RefLevel: Refinement level - how many knots to insert in each interval.

Returns: Refined curve.

Description: Refine the given curve by adding RefLevel knots in the middle of any interior knot interval.
See also: CagdCrvRefineAtParams, CagdCrvRefineUniformly2,

3.2.374 CagdCrvRefineUniformly2 (cagd_aux.c:1891)

```
CagdCrvStruct *CagdCrvRefineUniformly2(const CagdCrvStruct *Crv, int n)
```

Crv: To refine.

n: How many knots to insert.

Returns: Refined curve.

Description: Refine the given curve by adding RefLevel knots in the middle of any interior knot interval.
See also: CagdCrvRefineAtParams, CagdCrvRefineUniformly,

3.2.375 CagdCrvRegionFromCrv (cagd_aux.c:1638)

regions

subdivision

```
CagdCrvStruct *CagdCrvRegionFromCrv(const CagdCrvStruct *Crv,
                                    CagdRType t1,
                                    CagdRType t2)
```

Crv: To extract a sub-region from.

t1, t2: Parametric domain boundaries of sub-region.

Returns: Sub-region extracted from Crv from t1 to t2.

Description: Given a curve - extracts a sub-region within the domain specified by t1 and t2.
See also: CagdCrvRegionFromCrvWrap,

3.2.376 CagdCrvRegionFromCrvWrap (cagd_aux.c:1734)

regions

subdivision

```
CagdCrvStruct *CagdCrvRegionFromCrvWrap(const CagdCrvStruct *Crv,
                                        CagdRType t1,
                                        CagdRType t2)
```

Crv: To extract a sub-region from.

t1, t2: Parametric domain boundaries of sub-region.

Returns: Sub-region extracted from Crv from t1 to t2.

Description: Given a periodic curve, extracts a sub-region within the domain specified by t1 and t2. If t1 is greater than t2, the resulting curve subregion id from t1 to the end of the curve and then from the start of the curve to t2.
See also: CagdCrvRegionFromCrv,

3.2.377 CagdCrvReverse (cagd_aux.c:1961)

reverse

```
CagdCrvStruct *CagdCrvReverse(const CagdCrvStruct *Crv)
```

Crv: To be reversed.

Returns: Reversed curve of Crv.

Description: Returns a new curve that is the reversed curve of Crv by reversing the control polygon and the knot vector of Crv is a B-spline curve. See also BspKnotReverse.
See also: CagdCrvReverseUV,

3.2.378 CagdCrvReverseUV (cagd_aux.c:2036)

reverse

```
CagdCrvStruct *CagdCrvReverseUV(const CagdCrvStruct *Crv)
```

Crv: To be UV reversed.

Returns: UV reversed curve of Crv.

Description: Returns a new curve in which the first two (UV or XY)) coordinates are reversed from the input Crv.
See also: CagdCrvReverse,

3.2.379 CagdCrvRotateToXY (cagd2gen.c:2337)

```
CagdCrvStruct *CagdCrvRotateToXY(const CagdCrvStruct *Crv)
```

Crv: To rotate, to the XY plane.

Returns: Rotated Crv if reasonably successfull, NULL if failed.

Description: Rotates the given (hopefully planar) curve to the XY plane. If the curve is not planar, the rotation is heuristic and is not optimal in any sense.
See also: CagdCrvTransform, CagdCrvMatTransform, CagdCrvRotateToXYMat,

3.2.380 CagdCrvRotateToXYMat (cagd2gen.c:2259)

```
CagdBType CagdCrvRotateToXYMat(const CagdCrvStruct *Crv, IrtHmgnMatType Mat)
```

Crv: To compute a matrix that rotate (and possibly translate) Crv to the XY plane.

Mat: Defining the transformation.

Returns: TRUE if reasonably successfully, FALSE if failed.

Description: Computes a rotation matrix to rotate the given (hopefully planar) curve to the XY plane, in place. If the curve is not planar, the rotation is heuristic and is not optimal in any sense.
See also: CagdCrvTransform, CagdCrvMatTransform, CagdCrvRotateToXY,

3.2.381 CagdCrvScalarCrvSlopeBounds (cagd_aux.c:808)

```
void CagdCrvScalarCrvSlopeBounds(const CagdCrvStruct *Crv,
                                 CagdRType *MinSlope,
                                 CagdRType *MaxSlope)
```

Crv: Scalar curve to estimate its extreme slopes.

MinSlope: Minimal slope detected.

MaxSlope: Maximal slope detected.

Returns: void

Description: Compute slopes' bounds to a scalar curve.
See also: CagdCrvDerive,

3.2.382 CagdCrvScale (cagd2gen.c:1239)

scaling

transformations

```
void CagdCrvScale(CagdCrvStruct *Crv, const CagdRType *Scale)
```

Crv: To be non-uniformly scaled.

Scale: Scaling amount.

Returns: void

Description: Applies a nonuniform scaling transform, in place, to given curve Crv as specified by Scale.
See also: CagdSrfTransform, CagdTransform, CagdCrvMatTransform, CagdCrvRotateToXY, CagdCrvTransform,

3.2.383 CagdCrvScaleCenter (cagd2gen.c:1276)

scaling

transformations

```
void CagdCrvScaleCenter(CagdCrvStruct *Crv, const CagdRType *Scale)
```

Crv: To be non-uniformly scaled.

Scale: Scaling amount.

Returns: void

Description: Applies a nonuniform scaling transform, in place, to given curve Crv as specified by Scale, around the center of the curve.
See also: CagdSrfTransform, CagdTransform, CagdCrvMatTransform, CagdCrvRotateToXY, CagdCrvTransform,

3.2.384 CagdCrvSetDomain (cagd_aux.c:79)

domain

parametric domain

```
CagdCrvStruct *CagdCrvSetDomain(CagdCrvStruct *Crv,
                                CagdRType TMin,
                                CagdRType TMax)
```

Crv: To reset its parametric domain.

TMin: Minimal domain's new boundary.

TMax: Maximal domain's new boundary.

Returns: Modified curve, in place.

Description: Affinely reset the parametric domain of a curve, in place.
See also: BspCrvDomain, BspKnotAffineTrans2, CagdSrfSetDomain,

3.2.385 CagdCrvSubdivAtAllC0Discont (cagd_aux.c:2137)

```
CagdCrvStruct *CagdCrvSubdivAtAllC0Discont(const CagdCrvStruct *Crv,
                                           IrtBType EuclideanC0Discont,
                                           IrtRType Tolerance)
```

Crv: To subdivide at all C^0 discontinuity locations.

EuclideanC0Discont: TRUE to compute the C0 discontinuities and verify them in the Euclidean space. FALSE to only consider C0 discontinuities by knot multiplicity in parametric space.

Tolerance: Of parametric C0 discontinuity that is also a Euclidean discontinuity - th eallowed distance.

Returns: Curve segments result from the subdivision.

Description: Subdivides the given curve at all C^0 potential discontinuity locations.
See also: CagdCrvSubdivAtParams, BspKnotAllC1Discont, , BspCrvsSubdivAtAllDetectedLocations, CagdCrvSubdivAtAllC1Discont,

3.2.386 CagdCrvSubdivAtAllC1Discont (cagd_aux.c:2281)

```
CagdCrvStruct *CagdCrvSubdivAtAllC1Discont(const CagdCrvStruct *Crv,
                                           IrtBType EuclideanC1Discont,
                                           IrtRType Tolerance)
```

Crv: To subdivide at all C^1 discontinuity locations.

EuclideanC1Discont: TRUE to compute the C1 discontinuities and verify them in the Euclidean space. FALSE to only consider C1 discontinuities by knot multiplicity in parametric space.

Tolerance: Of parametric C1 discontinuity that is also a Euclidean discontinuity - deviation from inner product of unit tangents before and after discontinuity by less than Tolerance (1.0 if tangent identical, 0.0 if orthogonal, -1.0 if opposite). Ignored if < -1.0 or EuclideanC1Discont is FALSE.

Returns: Curve segments result from the subdivision.

Description: Subdivides the given curve at all C^1 potential discontinuity locations.
See also: CagdCrvSubdivAtParams, BspKnotAllC1Discont, , BspCrvsSubdivAtAllDetectedLocations, CagdCrvSubdivAtAllC0Discont, , BspCrvAllEuclideanC1Discont,

3.2.387 CagdCrvSubdivAtParam (cagd_aux.c:1343)

subdivision

```
CagdCrvStruct *CagdCrvSubdivAtParam(const CagdCrvStruct *Crv, CagdRType t)
```

Crv: To subdivide at the prescibed parameter value t.

t: The parameter to subdivide the curve Crv at.

Returns: A list of the two curves resulting from the process of subdivision.

Description: Given a curve - subdivides it into two curves at the given parameter value t. Returns pointer to first curve in a list of two subdivided curves.
See also: CagdCrvSubdivAtParams,

3.2.388 CagdCrvSubdivAtParams (cagd_aux.c:1550)

```
CagdCrvStruct *CagdCrvSubdivAtParams(const CagdCrvStruct *CCrv,
                                     const CagdPtStruct *Pts,
                                     CagdRType Eps,
                                     CagdBType PurgeTooSmallSegs,
                                     int *Proximity)
```

CCrv: Curve to split at all parameter values as prescribed by Pts. Bezier curves are promoted to B-spline curves in this function.

Pts: Ordered list of parameter values (first coordinate of point) to split curve Crv at.

Eps: parameter closer than Eps to boundary or other parameters are ignored.

PurgeTooSmallSegs: TRUE to purge too small curve segments in result.

Proximity: A 3 bits marker to return if the first (last) parameter was too close to the boundary and/or two middle parameters were too close and one of them was ignored as follows, 0x01 - first parameter to split at was too close to the boundary. 0x02 - last parameter to split at was too close to the boundary. 0x04 - a middle parameter was too close to another parameter.

Returns: List of splitted curves, in order.

Description: Given a curve - subdivides it into curves at all the given parameter values Pts. Pts is assumed to hold the parameters in order in the first point coordinate. Returns pointer to first curve in a list of subdivided curves.
See also: CagdCrvSubdivAtParam, CagdCrvSubdivAtParams2, CagdCrvSubdivAtParams3,

3.2.389 CagdCrvSubdivAtParams2 (cagd_aux.c:1410)

```
CagdCrvStruct *CagdCrvSubdivAtParams2(const CagdCrvStruct *CCrv,
                                      const CagdPtStruct *Pts,
                                      int Idx,
                                      CagdRType Eps,
                                      CagdBType PurgeTooSmallSegs,
                                      int *Proximity)
```

CCrv: Curve to split at all parameter values as prescribed by Pts. Bezier curves are promoted to B-spline curves in this function.

Pts: Unordered list of parameter values (first coordinate of point) to split curve Crv at.

Idx: Index of parameter in Pts points: 0 for X, 1 for Y, etc.

Eps: parameter closer than Eps and/or closer to boundary than Eps are ignored.

PurgeTooSmallSegs: TRUE to purge too small curve segments in result.

Proximity: A 3 bits marker to return if the first (last) parameter was too close to the boundary and/or two middle parameters were too close and one of them was ignored as follows 0x01 - first parameter to split at was too close to the boundary. 0x02 - last parameter to split at was too close to the boundary. 0x04 - a middle parameter was too close to another parameter.

Returns: List of splitted curves, in order.

Description: Given a curve - subdivides it into curves at all the given parameter values Pts. Pts can hold the parameters in any order in the Idx point coordinate. Returns pointer to first curve in a list of subdivided curves.
See also: CagdCrvSubdivAtParam, CagdCrvSubdivAtParams, CagdCrvSubdivAtParams3,

3.2.390 CagdCrvSubdivAtParams3 (cagd_aux.c:1488)

```
CagdCrvStruct *CagdCrvSubdivAtParams3(const CagdCrvStruct *CCrv,
                                      CagdRType *Prms,
                                      int PrmsLen,
                                      CagdRType Eps,
                                      CagdBType PurgeTooSmallSegs,
                                      int *Proximity)
```

CCrv: Curve to split at all parameter values as prescribed by Pts. Bezier curves are promoted to B-spline curves in this function.

Prms: Unordered list of parameter values (first coordinate of point) to split curve Crv at. Sorted in place.

PrmsLen: Length pf Prms vector.

Eps: parameter closer than Eps and/or closer to boundary than Eps are ignored.

PurgeTooSmallSegs: TRUE to purge too small curve segments in result.

Proximity: A 3 bits marker to return if the first (last) parameter was too close to the boundary and/or two middle parameters were too close and one of them was ignored as follows 0x01 - first parameter to split at was too close to the boundary. 0x02 - last parameter to split at was too close to the boundary. 0x04 - a middle parameter was too close to another parameter.

Returns: List of splitted curves, in order.

Description: Given a curve - subdivides it into curves at all the given parameter values in Prms. Prms can hold the parameters in any order. Returns pointer to first curve in a list of subdivided curves.
See also: CagdCrvSubdivAtParam, CagdCrvSubdivAtParams, CagdCrvSubdivAtParams2,

3.2.391 CagdCrvTanAngularSpan (cagd_cci.c:67)

```
CagdBType CagdCrvTanAngularSpan(const CagdCrvStruct *Crv,
                                CagdVType ConeDir,
                                CagdRType *AngularSpan)
```

Crv: Curve to consider

ConeDir: General, median, direction of tangent field, in XY plane.

AngularSpan: Maximal deviation of tangent field from Dir, in radians.

Returns: TRUE if angular span of curve is less than 180 degrees, FALSE otherwise. In the later case, Dir and Angle are invalid.

Description: Given a curve, computes the angular span of its tangent field, in XY plane.
See also: CagdCrvCrvInter,

3.2.392 CagdCrvTangentToData (cagd_aux.c:3228)

tangent

```
CagdVecStruct *CagdCrvTangentToData(const CagdCrvStruct *Crv,
                                    CagdRType t,
                                    CagdBType Normalize,
                                    CagdVecStruct *Tan)
```

Crv: To compute (unit) tangent vector for.

t: Location where to evaluate the tangent of Crv.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, returned is an unnormalized vector in the right direction of the tangent.

Tan: A pointer to a vector holding the unit tanegnt information.

Returns: A pointer to a vector holding the unit tanegnt information.

Description: Given a curve Crv and a parameter value t, returns the (unit) tangent direction of Crv at t. The unnormalized normal does not equal dC/dt in its magnitude, only in its direction.

3.2.393 CagdCrvToMesh (cagd_aux.c:2950)

curve from mesh

```
void CagdCrvToMesh(const CagdCrvStruct *Crv,
                   int Index,
                   CagdSrfDirType Dir,
                   CagdSrfStruct *Srf)
```

Crv: To substitute into the surface Srf.

Index: Of mesh where the curve Crv should be substituted in.

Dir: Either U or V.

Srf: That a row or a column of should be replaced by Crv.

Returns: void

Description: Substitutes a row/column of surface Srf from the given curve Crv at surface direction Dir and mesh index Index. Curve must have the same PtType/Length as the surface in the selected direction.
See also: CagdCrvFromSrf, CagdCrvFromMesh,

3.2.394 CagdCrvTransform (cagd2gen.c:1199)

scaling

translation

transformations

```
void CagdCrvTransform(CagdCrvStruct *Crv,
                      const CagdRType *Translate,
                      CagdRType Scale)
```

Crv: To be affinely transformed.

Translate: Translation amount, NULL for non.

Scale: Scaling amount.

Returns: void

Description: Applies an affine transform, in place, to given curve Crv as specified by Translate and Scale. Each control point is first translated by Translate and then scaled by Scale.
See also: CagdSrfTransform, CagdTransform, CagdCrvMatTransform, CagdCrvRotateToXY, CagdCrvScale,

3.2.395 CagdCrvTwoCrvsOrient (crvmatch.c:1204)

```
CagdBType CagdCrvTwoCrvsOrient(CagdCrvStruct *Crv1, CagdCrvStruct *Crv2, int n)
```

Crv1, Crv2: The two curvs to consider.

n: Number of samples to take on the curves.

Returns: TRUE if needs to reverse, FALSE if as is is better.

Description: Check if macthed points on the given two curves are closer as is or when one of the curves is reversed.

3.2.396 CagdCrvUnitMaxCoef (cagd2gen.c:2128)

```
CagdCrvStruct *CagdCrvUnitMaxCoef(CagdCrvStruct *Crv)
```

Crv: Curve to normalize in place its coefficients.

Returns: Normalized curve, in place

Description: Normalize in place the given curve so its maximal coefficient is of unit size. Each axis is treated independently, including W.
See also: CagdSrfUnitMaxCoef,

3.2.397 CagdCrvUpdateLength (cagd1gen.c:3104)

```
CagdCrvStruct *CagdCrvUpdateLength(CagdCrvStruct *Crv, int NewLength)
```

Crv: Curve to update its length.

NewLength: New length to reallocate for the curve.

Returns: Resized curve, in place.

Description: Resize the length of the curve, in place. The new curve is not the same as the original while a minimal effort is invested to keep it similar.
See also: CagdSrfUpdateLength,

3.2.398 CagdCrvZeroNumericStep (bzrzrfct.c:820)

```
CagdBType CagdCrvZeroNumericStep(const CagdCrvStruct *Crv,
                                 CagdCrvStruct *DCrv,
                                 CagdRType Seed,
                                 CagdBType LeftOut,
                                 CagdBType RightOut,
                                 CagdBType CheckEndPts,
                                 CagdRType *Solution,
                                 CagdRType NumericTol)
```

Crv: A scalar curve to find zero of. Either a Bezier or B-spline.

DCrv: The hodograph of Crv. Optional and can be NULL to be computed on the fly.

Seed: Initial guess to start the NR iterations from.

LeftOut: If TRUE, let the method stray outside the left boundary of the domain of search.

RightOut: If TRUE, let the method stray outside the right boundary of the domain of search.

CheckEndPts: If TRUE, check for roots at end-points of domain.

Solution: The solution point to return.

NumericTol: Numeric tolerance within which solution is required.

Returns: TRUE if solution was found, otherwise FALSE.

Description: Newton-Raphson method for a Bezier or a B-spline curve.
See also: MvarCrvZeroSet, CagdCrvCrvInter, SymbCrvZeroSet, , SymbCrvCrvInter, SymbScalarCrvLowDegZeroSet, CagdCrvZeroSet.,

3.2.399 CagdCrvZeroSet (bzrzrfct.c:84)

```
CagdPtStruct *CagdCrvZeroSet(const CagdCrvStruct *Crv,
                             int Axis,
                             int NRInit,
                             CagdRType NumericTol,
                             CagdRType SubdivTol)
```

Crv: A curve to find zeros on prescribed axis.

Axis: The axis for which zeros are to be found - 1 for X etc.

NRInit: Indicates how to initialize Newton-Raphson method. A value of 0 indicates seed value 0.5, values 1 and 2 indicate seeds computed by intersecting the control polygon of Crv with the domain axis, with 1 implying an intersection close to the middle of the domain and 2 implying the first intersection.

NumericTol: The numeric tolerance up to which solution is required.

SubdivTol: The tolerance up to which subdivision is to be done.

Returns: List of solutions as tuples of the form (root, multiplicity).

Description: Finds the zeros of a given curve along the required axis along with multiplicities of zeros. Employs factoring out of (t) and (1-t) terms after subdivision at roots already discovered.
See also: MvarCrvZeroSet, CagdCrvCrvInter, SymbCrvZeroSet, , SymbCrvCrvInter, SymbScalarCrvLowDegZeroSet, CagdCrvZeroSetC0,

3.2.400 CagdCrvZeroSetC0 (bzrzrfct.c:135)

```
CagdPtStruct *CagdCrvZeroSetC0(const CagdCrvStruct *Crv,
                               int Axis,
                               int NRInit,
                               CagdRType NumericTol,
                               CagdRType SubdivTol)
```

Crv: A curve to find zeros on prescribed axis.

Axis: The axis for which zeros are to be found - 1 for X etc.

NRInit: Indicates how to initialize Newton-Raphson method. A value of 0 indicates seed value 0.5, values 1 and 2 indicate seeds computed by intersecting the control polygon of Crv with the domain axis, with 1 implying an intersection close to the middle of the domain and 2 implying the first intersection.

NumericTol: The numeric tolerance up to which solution is required.

SubdivTol: The tolerance up to which subdivision is to be done.

Returns: List of solutions as tuples of the form (root, multiplicity). Note multiplicity will be IRIT_MAX_INT if a whole region was detected as zero.

Description: Finds the zeros of a given curve along the required axis along with multiplicities of zeros. Employs factoring out of (t) and (1-t) terms after subdivision at roots already discovered. Curve is assumed to be C0 continuous.
See also: MvarCrvZeroSet, CagdCrvCrvInter, SymbCrvZeroSet, , SymbCrvCrvInter, SymbScalarCrvLowDegZeroSet, CagdCrvZeroSet.,

3.2.401 CagdCrvonSrfBndry (cagd1gen.c:3298)

```
CagdSrfBndryType CagdCrvonSrfBndry(const CagdCrvStruct *Crv,
                                   const CagdSrfStruct *Srf)
```

Crv: UV Curve to check if on the boundary of Srf.

Srf: Surface to check if Crv is on its boundary.

Returns: Boundary that Crv is on or CAGD_NO_BNDRY otherwise.

Description: Checks if given UV curve is on the boundary of Srf.
See also: CAGD_PT_ON_BNDRY,

3.2.402 CagdCrvsRelation (cagd1gen.c:1967)

```
int CagdCrvsRelation(const CagdCrvStruct *Crv1,
                     const CagdCrvStruct *Crv2,
                     CagdRType ParialOverlapRatio,
                     int *Crv1StartOn2,
                     int *Crv1EndOn2,
                     int *Crv2StartOn1,
                     int *Crv2EndOn1,
                     CagdRType *Crv1StartOn2Prm2,
                     CagdRType *Crv1EndOn2Prm2,
                     CagdRType *Crv2StartOn1Prm1,
                     CagdRType *Crv2EndOn1Prm1,
                     CagdRType *RelOverlap,
                     CagdRType Eps)
```

Crv1, Crv2: The two curves to consider their relation.

ParialOverlapRatio: Ratio (between zero and one) to consider the result as partial overlap (both directions).

Crv1StartOn2: Set to TRUE if start pt of Crv1 is on Crv2 (within Eps).

Crv1EndOn2: Set to TRUE if end pt of Crv1 is on Crv2 (within Eps).

Crv2StartOn1: Set to TRUE if start pt of Crv2 is on Crv1 (within Eps).

Crv2EndOn1: Set to TRUE if end pt of Crv2 is on Crv1 (within Eps).

Crv1StartOn2Prm2: Parameter on Crv2 at which Crv1StartOn2 event occurs.

Crv1EndOn2Prm2: Parameter on Crv2 at which Crv1EndOn2 event occurs.

Crv2StartOn1Prm1: Parameter on Crv1 at which Crv2StartOn1 event occurs

Crv2EndOn1Prm1: Parameter on Crv1 at which Crv2EndOn1 event occurs.

RelOverlap: A vector of size 3 to be updated with the: (OverPortionCrv1, OverlapPortionCrv2, CommonArcLen). Only maximal relative overlap is computed.

Eps: Tolerance to consider the two curves as related.

Returns: 0 if the curves are not related or, 1 if Crv1 contains Crv2, -1 if Crv1 contains Crv2 (flipped), 2 if Crv2 contains Crv1, -2 if Crv2 contains Crv1 (flipped), 3 if Crv1 and curve Crv2 have the same start/end location, -3 if Crv1 and curve Crv2 have flipped start/end location, 4 if the two curves partially overlap. -4 if the two curves partially overlap (flipped).

Description: See if the given two curves are related to each other (within Eps). Related means sharing some common region (to within Eps). Tested by examining the distances of the end points of one curve to the other.
See also: CagdCrvsSame,

3.2.403 CagdCrvsSame (cagd1gen.c:1793)

```
CagdBType CagdCrvsSame(const CagdCrvStruct *Crv1,
                       const CagdCrvStruct *Crv2,
                       CagdRType Eps)
```

Crv1, Crv2: The two curves to compare.

Eps: Tolerance of equality.

Returns: TRUE if curves are the same, FALSE otherwise.

Description: Compare the two lists of curves for similarity.
See also: CagdCtlMeshsSame, BspKnotVectorsSame, CagdSrfsSame, CagdCrvsSame2, , CagdCrvsSameUptoRigidScl2D, CagdSrfsSameUptoRigidScl2D, , CagdCrvsSameFuncSpace,

3.2.404 CagdCrvsSame2 (cagd1gen.c:1839)

```
CagdBType CagdCrvsSame2(const CagdCrvStruct *Crv1,
                        const CagdCrvStruct *Crv2,
                        CagdRType Eps)
```

Crv1, Crv2: The two curves to compare.

Eps: Tolerance of equality.

Returns: TRUE if curves are the same, FALSE otherwise.

Description: Compare the two curves for similarity, after bringing them to a common function space, by degree raising and refinement.
See also: CagdCtlMeshsSame, BspKnotVectorsSame, CagdCrvsSame, , CagdCrvsSameUptoRigidScl2D, CagdSrfsSameUptoRigidScl2D,

3.2.405 CagdCrvsSame3 (cagd1gen.c:1886)

```
CagdBType CagdCrvsSame3(const CagdCrvStruct *Crv1,
                        const CagdCrvStruct *Crv2,
                        CagdRType Eps,
                        CagdBType *Reversed)
```

Crv1, Crv2: The two curves to compare.

Eps: Tolerance of equality.

Reversed: Optional parameter to be set if the curves are reversed. Can be NULL to ignore.

Returns: TRUE if curves are the same, FALSE otherwise.

Description: Compare the two curves for similarity, as is and in reverse.
See also: CagdCtlMeshsSame, BspKnotVectorsSame, CagdCrvsSame, , CagdCrvsSameUptoRigidScl2D, CagdSrfsSameUptoRigidScl2D,

3.2.406 CagdCrvsSameFuncSpace (cagd1gen.c:1755)

```
CagdBType CagdCrvsSameFuncSpace(const CagdCrvStruct *Crv1,
                                const CagdCrvStruct *Crv2,
                                CagdRType Eps)
```

Crv1, Crv2: The two curves to compare.

Eps: Tolerance of equality.

Returns: TRUE if curves are in same function space, FALSE otherwise.

Description: Compare the two curves for similarity of their function space.
See also: CagdCtlMeshsSame, BspKnotVectorsSame, CagdSrfsSame, CagdCrvsSame2, , CagdCrvsSameUptoRigidScl2D, CagdSrfsSameUptoRigidScl2D, CagdCrvsSame,

3.2.407 CagdCrvsSameUptoRigidScl2D (cagd1gen.c:1707)

```
CagdBType CagdCrvsSameUptoRigidScl2D(const CagdCrvStruct *Crv1,
                                     const CagdCrvStruct *Crv2,
                                     IrtPtType Trans,
                                     CagdRType *Rot,
                                     CagdRType *Scl,
                                     CagdRType Eps)
```

Crv1, Crv2: The two curves to compare.

Trans: Translation amount to apply to Crv1 to bring to Crv2 (after rotation/scale).

Rot, Scl: Rotation and scale amounts to apply to Crv1 to bring to Crv2 (before translation). Rot is specified in degrees.

Eps: Tolerance of equality.

Returns: TRUE if curves are the same, FALSE otehrwise. Trans & Rot are valid only if this function returns TRUE.

Description: Compare the two planar curves for similarity up to rigid motion and scale in the XY plane.
See also: CagdCtlMeshsSame, BspKnotVectorsSame, CagdSrfsSame, CagdCrvsSame, CagdSrfsSameUptoRigidScl2D,

3.2.408 CagdCtlMeshAverageValue (cagdbbox.c:780)

bbox

bounding box

minimum

maximum

```
CagdRType CagdCtlMeshAverageValue(CagdRType * const *Pts,
                                  int Length,
                                  int Axis)
```

Pts: To compute an average value of its control mesh, in some Axis.

Length: The number of control points in the control mesh.

Axis: 1 for X, 2 for Y etc.

Returns: Average value.

Description: Computes an average value of all control points of given control mesh in a given axis. Rational values are taken into account (projected into Euclidean space first).
See also: CagdCrvMinMax, CagdPointsBBox, CagdSrfAverageValue, CagdSrfAverageValue,

3.2.409 CagdCtlMeshsSame (cagdcoer.c:312)

```
CagdBType CagdCtlMeshsSame(CagdRType * const Mesh1[],
                           CagdRType * const Mesh2[],
                           int Len,
                           CagdRType Eps)
```

Mesh1, Mesh2: Two control meshes to compare.

Len: Length of control meshes.

Eps: Tolerance of equality.

Returns: TRUE if control meshes are the same, FALSE otherwise.

Description: Compare the two control meshes for similarity.
See also: BspKnotVectorsSame, CagdRealVecSame, CagdCrvsSame, CagdSrfsSame,

3.2.410 CagdCtlMeshsSameUptoRigidScl2D (cagdcoer.c:422)

```
CagdBType CagdCtlMeshsSameUptoRigidScl2D(CagdRType * const Mesh1[],
                                         CagdRType * const Mesh2[],
                                         int Len,
                                         IrtPtType Trans,
                                         CagdRType *Rot,
                                         CagdRType *Scl,
                                         CagdRType Eps)
```

Mesh1, Mesh2: Two control meshes to compare.

Len: Length of control meshes.

Trans: Translation amount to apply second to Mesh1 to bring to Mesh2.

Rot, Scl: Rotation and scale amounts to apply first to Mesh1 to bring to Mesh2. Rot is specified in degrees.

Eps: Tolerance of equality.

Returns: TRUE if control meshes are the same, FALSE otherwise.

Description: Compare the two control meshes for similarity up to rigid motion and scale. Comparison is conducted in the XY plane and only X and Y (and W) are considered.
See also: BspKnotVectorsSame, CagdCrvsSame, CagdSrfsSame,

3.2.411 CagdCtlPtArrayFree (cagd2gen.c:574)

free

```
void CagdCtlPtArrayFree(CagdCtlPtStruct *CtlPtArray, int Size)
```

CtlPtArray: To be deallocated.

Size: Of the deallocated array.

Returns: void

Description: Deallocates and frees an array of CtlPt structure.

3.2.412 CagdCtlPtArrayNew (cagd1gen.c:384)

allocation

```
CagdCtlPtStruct *CagdCtlPtArrayNew(CagdPointType PtType, int Size)
```

PtType: Point type of control point.

Size: Size of CtlPt array to allocate.

Returns: An array of CtlPt structures of size Size.

Description: Allocates and resets all slots of an array of CtlPt structures.

3.2.413 CagdCtlPtBBox (cagdbbox.c:469)

bbox

bounding box

```
void CagdCtlPtBBox(const CagdCtlPtStruct *CtlPt, CagdBBoxStruct *BBox)
```

CtlPt: To compute bounding box for.

BBox: Where bounding information is to be saved.

Returns: void

Description: Computes a bounding box for a control point.
See also: CagdPolygonListBBox, CagdPointsBBox, CagdPointsBBox2,

3.2.414 CagdCtlPtCopy (cagd1gen.c:999)

copy

```
CagdCtlPtStruct *CagdCtlPtCopy(const CagdCtlPtStruct *CtlPt)
```

CtlPt: To be copied.

Returns: A duplicate of CtlPt.

Description: Allocates and copies all slots of a CtlPt structure.

3.2.415 CagdCtlPtCopyList (cagd1gen.c:1353)

copy

```
CagdCtlPtStruct *CagdCtlPtCopyList(const CagdCtlPtStruct *CtlPtList)
```

CtlPtList: To be copied.

Returns: A duplicated list of CtlPt's.

Description: Allocates and copies a list of CtlPt structures.

3.2.416 CagdCtlPtFree (cagd2gen.c:526)

free

```
void CagdCtlPtFree(CagdCtlPtStruct *CtlPt)
```

CtlPt: To be deallocated.

Returns: void

Description: Deallocates and frees all slots of a CtlPt structure.

3.2.417 CagdCtlPtFreeList (cagd2gen.c:549)

free

```
void CagdCtlPtFreeList(CagdCtlPtStruct *CtlPtList)
```

CtlPtList: To be deallocated.

Returns: void

Description: Deallocates and frees a CtlPt structure list:

3.2.418 CagdCtlPtListBBox (cagdbbox.c:505)

```
CagdBType CagdCtlPtListBBox(const CagdCtlPtStruct *CtlPts,
                            CagdBBoxStruct *BBox)
```

CtlPts: To computes its bbox. All ctlpts should be of the same type.

BBox: Where bounding information is to be saved.

Returns: TRUE if successful, FALSE if error.

Description: Computes a bounding box for a list of control points.
See also: CagdCtlPtBBox, CagdPointsBBox, CagdPointsBBox2,

3.2.419 CagdCtlPtNew (cagd1gen.c:413)

allocation

```
CagdCtlPtStruct *CagdCtlPtNew(CagdPointType PtType)
```

PtType: Point type of control point.

Returns: A CtlPt structure.

Description: Allocates and resets all slots of a CtlPt structure.

3.2.420 CagdCubicCrvFit (hermite.c:111)

Hermite

```
CagdCrvStruct *CagdCubicCrvFit(const CagdCrvStruct *Crv)
```

Crv: To approximate as a cubic.

Returns: A cubic Bezier curve, fitting the end position and tangent values.

Description: Construct a cubic Bezier curve that fits the input curve end points and end derivatives.
See also: CagdCubicHermiteCrv, CagdQuaraticCrvFit, SymbApproxCrvAsBzrCubics,

3.2.421 CagdCubicHermiteCrv (hermite.c:34)

Hermite

```
CagdCrvStruct *CagdCubicHermiteCrv(const CagdPType Pt1,
                                   const CagdPType Pt2,
                                   const CagdVType Dir1,
                                   const CagdVType Dir2)
```

Pt1, Pt2: Starting and end points of curve.

Dir1, Dir2: Starting and end vectors of curve.

Returns: A cubic Bezier curve, satisfying the four constrants.

Description: Construct a cubic Bezier curve using the Hermite constraints - two positions and two tangents.
See also: CagdCubicCrvFit, CagdCubicHermiteCrv2,

3.2.422 CagdCubicHermiteCrv2 (hermite.c:75)

Hermite

```
CagdCrvStruct *CagdCubicHermiteCrv2(const CagdRType *Pt1,
                                    const CagdRType *Pt2,
                                    const CagdRType *Dir1,
                                    const CagdRType *Dir2,
                                    int Dim)
```

Pt1, Pt2: Starting and end points of curve.

Dir1, Dir2: Starting and end vectors of curve.

Dim: Dimension of points (and hence also curve).

Returns: A cubic Bezier curve, satisfying the four constrants.

Description: Construct a cubic Bezier curve using the Hermite constraints - two positions and two tangents.
See also: CagdCubicCrvFit, CagdCubicHermiteCrv,

3.2.423 CagdCubicHermiteSrf (hermite.c:225)

Hermite

```
CagdSrfStruct *CagdCubicHermiteSrf(const CagdCrvStruct *CPos1Crv,
                                   const CagdCrvStruct *CPos2Crv,
                                   const CagdCrvStruct *CDir1Crv,
                                   const CagdCrvStruct *CDir2Crv)
```

CPos1Crv, CPos2Crv: Starting and end curves of surface.

CDir1Crv, CDir2Crv: Starting and end tangent fields surface.

Returns: A cubic by something Bezier surface, satisfying the four constraints. The other something degree is the largest of the four given curves.

Description: Construct a cubic surface using the Hermite constraints - two positions and two tangents. Other direction's degree depends on input.

3.2.424 CagdCubicSrfFit (hermite.c:528)

Hermite

```
CagdSrfStruct *CagdCubicSrfFit(const CagdSrfStruct *Srf)
```

Srf: To approximate as a bicubic.

Returns: A bicubic Bezier surface, fitting the corners positions and corners tangents.

Description: Construct a bicubic Bezier surface that fits the input surface corner points and corner derivatives.
See also: CagdCubicHermiteSrf, CagdQuaraticSrfFit, CagdCubicCrvFit,

3.2.425 CagdDbg (cagd_dbg.c:32)

debugging

```
void CagdDbg(const void *Obj)
```

Obj: Either a list of curves or a surfaces - to be printed to stderr.

Returns: void

Description: Prints curves and surfaces to stderr. Should be linked to programs for debugging purposes, so curves and surfaces may be inspected from the debugger.
See also: CagdDbg1,

3.2.426 CagdDbg1 (cagd_dbg.c:83)

debugging

```
void CagdDbg1(const void *Obj)
```

Obj: Either a curve or a surface - to be printed to stderr.

Returns: void

Description: Prints one curve or surface to stderr. Should be linked to programs for debugging purposes, so curves and surfaces may be inspected from the debugger.
See also: CagdDbg,

3.2.427 CagdDbgDsp (cagd_dbg.c:142)

debugging

```
void CagdDbgDsp(const void *Obj)
```

Obj: Either a curve or a surface - to be displayed.

Returns: void

Description: Views curves and surfaces in a display device. Should be linked to programs for debugging purposes, so curves and surfaces may be inspected from the debugger.

3.2.428 CagdDegreeRaiseMatProd (blossom.c:864)

```
CagdBlsmAlphaCoeffStruct *CagdDegreeRaiseMatProd(CagdBlsmAlphaCoeffStruct *A1,
                                                 CagdBlsmAlphaCoeffStruct *A2)
```

A1, A2: matrices to multiply.

Returns: Resulting product matrix.

Description: Computes the product of two adjacent degree raising matrices. Matrices are adjacent if A1 raises from Order to Order+k and A2 from Order+k to Order+n. Returned matrix is a degree raising matrix from Order to Order+n.
See also: CagdBlossomDegreeRaiseMat, CagdCrvDegreeRaise,

3.2.429 CagdDistCrvLine (cagd_cci.c:153)

curve line distance

```
CagdRType CagdDistCrvLine(const CagdCrvStruct *Crv, CagdLType Line)
```

Crv: Planar curve to compute its signed distance to the line. Assumed to be a Bezier or a Bspline curve.

Line: Line Equations, in the XY plane.

Returns: Zero if might intersect. Otherwise, a bound on the minimal possible distance, signed.

Description: Given a curve and a line in the XY Plane, finds a bound on the minimal signed distance between the two. Returns positive/negative minimal expected distance if curve on either side of the line or zero if might intersect. Computation is performed by measuring the signed distance between the line and all control points of the curve.

See also: SymbDistCrvLine, SymbLclDistCrvLine,

3.2.430 CagdDistPtPlane (mshplanr.c:143)

point plane distance

```
CagdRType CagdDistPtPlane(CagdPlaneStruct const *Plane,
                          CagdRType * const *Points,
                          int Index,
                          int MaxDim)
```

Plane: To compute the distance to.

Points: To compute the distance from.

Index: Index in Points for the point to consider.

MaxDim: Number of dimensions to consider. Less or equal to three.

Returns: Resulting distance.

Description: Computes and returns distance between point Index and given plane which is assumed to be normalized, so that the A B C plane;s normal has a unit length. Also assumes the Points are non rational with MaxDim dimension.

3.2.431 CagdDistTwoCtlPt (cagdcoer.c:981)

```
CagdRType CagdDistTwoCtlPt(CagdRType * const *Pt1,
                           int Index1,
                           CagdRType * const *Pt2,
                           int Index2,
                           CagdPointType PType)
```

Pt1, Index1, Pt2, Index2: Two Control points to compute distance between, and indices into the vectors of Points, Pt1 and Pt2. If, however, Index? < 0, Pt? is a single point.

PType: Type of points Pt?.

Returns: The distance between Pt1 and Pt2

Description: Computes the L2 distance between two arbitrary control points.

3.2.432 CagdDistTwoCtlPt2 (cagdcoer.c:1018)

```
CagdRType CagdDistTwoCtlPt2(CagdRType * const *Points,
                            int Index1,
                            int Index2,
                            CagdPointType PType)
```

Points: The vector of Control points of the freeform.

Index1, Index2: The two indices of the two control points.

PType: Type of points Pt?.

Returns: The distance between Pt1 and Pt2

Description: Computes the L2 distance between two arbitrary control points of some freeform.

3.2.433 CagdEditSingleCrvPt (cagdedit.c:33)

curve editing

```
CagdCrvStruct *CagdEditSingleCrvPt(const CagdCrvStruct *Crv,
                                   CagdCtlPtStruct *CtlPt,
                                   int Index,
                                   CagdBType Write)
```

Crv: Curve to be modified/query.

CtlPt: New control point to be substituted into Crv. Must carry the same PType as Crv if to be written to Crv.

Index: In curve CRV's control polygon to substitute/query CtlPt.

Write: If TRUE CtlPt is copied into Crv, if FALSE the point is copied from Crv to CtlPt.

Returns: If Write is TRUE, the new modified curve, if WRITE is FALSE, NULL.

Description: Provides the way to modify/get a single control point into/from the curve.

3.2.434 CagdEditSingleSrfPt (cagdedit.c:213)

surface editing

```
CagdSrfStruct *CagdEditSingleSrfPt(const CagdSrfStruct *Srf,
                                   CagdCtlPtStruct *CtlPt,
                                   int UIndex,
                                   int VIndex,
                                   CagdBType Write)
```

Srf: Surface to be modified/query.

CtlPt: New control point to be substituted into Srf. Must carry the same PType as Srf if to be written to Srf.

UIndex, VIndex: In surface Srf's control mesh to substitute/query CtlPt.

Write: If TRUE CtlPt is copied into Srf, if FALSE the point is copied from Srf to CtlPt.

Returns: If Write is TRUE, the new modified curve, if WRITE is FALSE, NULL.

Description: Provides the way to modify/get a single control point into/from a surface.

3.2.435 CagdEllipse3Points (cagd_cnc.c:692)

ellipse

```
CagdBType CagdEllipse3Points(CagdPType Pt1,
                             CagdPType Pt2,
                             CagdPType Pt3,
                             CagdRType *A,
                             CagdRType *B,
                             CagdRType *C,
                             CagdRType *D,
                             CagdRType *E,
                             CagdRType *F)
```

Pt1, Pt2, Pt3: The 3 input points. Assumed non-colnear.

A, B, C, D, E, F: Coefficients of the computed bounding ellipse.

Returns: TRUE if succesful, FALSE otherwise.

Description: Constructs an ellipse in the XY plane through the given 3 points of minimal area. The A,B,C,D,E,F coefficients of the bounding ellipse as in A x^2 + B xy + C y^2 + D x + E y + F = 0. are returned.

```
  Algorithm:
1. Compute center,  C := (Pt1 + Pt2 + Pt3) / 3
```

$$2.\ \text{Computer a 2x2 matrix}\quad N = \frac{1}{3}\sum_{i=1}^{3} (Pti - C)\,(Pti - C)^T$$

```
3. M = N^{-1}
```

```
4. The ellipse E: (P - C)^T M (P - C) - Z = 0,    Z constant, P = (x, y).
```

See also: "Exact Primitives for Smallest Enclosing Ellipses", by Bernd Gartner and Sven Schonherr, Proceedings of the 13th annual symposium on Computational geometry, 1997.

See also: CagdEllipseOffset, CagdCreateConicCurve, CagdCreateConicCurve2, , CagdEllipse4Points,

3.2.436 CagdEllipse4Points (cagd_cnc.c:859)

ellipse

```
CagdBType CagdEllipse4Points(CagdPType Pt1,
                             CagdPType Pt2,
                             CagdPType Pt3,
                             CagdPType Pt4,
                             CagdRType *A,
                             CagdRType *B,
                             CagdRType *C,
                             CagdRType *D,
                             CagdRType *E,
                             CagdRType *F)
```

Pt1, Pt2, Pt3, Pt4: The 4 input points. Assumed in general position.

A, B, C, D, E, F: Coefficients of the computed bounding ellipse.

Returns: TRUE if successful, FALSE otherwise.

Description: Constructs an ellipse in the XY plane through the given 4 points of minimal area. The A,B,C,D,E,F coefficients of the bounding ellipse as in A x^2 + B xy + C y^2 + D x + E y + F = 0. are returned.

```
Algorithm:
```

1. Using the four points (x1,y1)..(x4,y4), the following matrices:

| x1^2 y1^2 2*x1*y1 2*x1 2*y1 1 | ACBDEF = | x2^2 y2^2 2*x2*y2 2*x2 2*y2 1 | | x3^2 y3^2 2*x3*y3 2*x3 2*y3 1 | | x4^2 y4^2 2*x4*y4 2*x4 2*y4 1 |
| 2*x1*y1 2*x1 2*y1 1 | matBDEF = | 2*x2*y2 2*x2 2*y2 1 | | 2*x3*y3 2*x3 2*y3 1 | | 2*x4*y4 2*x4 2*y4 1 |
and the following vectors:
x = | A C B D E F |
Zeros = | 0 0 0 0 |
are defined, and the 4 point interpolation constraints can be written as:
ACBDEF * x = Zeros

2. Using this, the following can be written:

BDEFfromAC = inv(matBDEF)*ACBDEF
which, by construction, is a matrix in the form:
| AB CB 1 0 0 0 | BDEFfromAC = | AD CD 0 1 0 0 | | AE CE 0 0 1 0 | | AF CF 0 0 0 1 |
Where AB, AD, AE, AF, CB, CD, CE, CF are functions of (x1,y1)..(x4,y4).
Using the previous equations, it is possible to write:
BDEFfromAC * x = Zeros
And conclude the following equations:
AB * A + CB * C + B = 0 AD * A + CD * C + D = 0 AE * A + CE * C + E = 0 AF * A + CF * C + F = 0

3. The A..F ellipse coefficients can be scaled so that C = 1 - A, allowing the following equations to be written, using the previous step:

C(A) = (-1) * A + 1 B(A) = (CB - AB) * A - CB D(A) = (CD - AD) * A - CD E(A) = (CE - AE) * A - CE F(A) = (CF - AF) * A - CF

4. This allows writing the ellipse area function:

area(A) = f(A)^3 / g(A)^2
where:
f(A) = A*C(A)-B(A)^2 g(A) = -d(A)^2*C(A)+2*d(A)*E(A)*B(A)-E(A)^2*A+F(A)*A*C(A)-F(A)*B(A)^2
as a function of A alone. The extreme (minimum) area values can be found by locating where its derivative with respect to A equals zero, by solving the cubic polynomial:
(-3*diff(f(A),A)*g(A) + 2*f(A)*diff(g(A),A))
which coefficients are defined using AB, AD, AE, AF, CB, CD, CE, CF mentioned above.
If solutions for A exist, they define the other ellipse coefficients (B, C, D, E, F) as mentioned above. If these coefficients define an ellipse, they are returned to the user. Otherwise, the function returns false.
See also: "Exact Primitives for Smallest Enclosing Ellipses", by Bernd Gartner and Sven Schonherr, Proceedings of the 13th annual symposium on Computational geometry, 1997.

See also: CagdEllipseOffset, CagdCreateConicCurve, CagdCreateConicCurve2,

3.2.437 CagdEllipseOffset (cagd_cnc.c:1029)

ellipse

```
CagdBType CagdEllipseOffset(CagdRType *A,
                            CagdRType *B,
                            CagdRType *C,
                            CagdRType *D,
                            CagdRType *E,
                            CagdRType *F,
                            CagdRType Offset)
```

A, B, C, D, E, F: The six coefficients of the ellipse.

Offset: Offset amount.

Returns: TRUE if successful, FALSE otherwise.

Description: Update the implicit form of the given ellipse with some offset Offset.

See also: CagdEllipse3Points, CagdEllipse4Points, CagdEllipse4Points, , CagdCreateConicCurve, CagdCreateConicCurve2,

3.2.438 CagdEstimateCrvCollinearity (mshplanr.c:213)

conversion

collinearity

```
CagdRType CagdEstimateCrvCollinearity(const CagdCrvStruct *Crv)
```

Crv: To measure its collinearity.

Returns: Collinearity relative measure.

Description: Tests polygonal collinearity by testing the distance of interior control points from the line connecting the two control polygon end points. Returns a relative ratio of deviation from line relative to its length. Zero means all points are collinear. If two end points are same (no line can be fit) IRIT_INFNTY is returned.

3.2.439 CagdEstimateSrfPlanarity (mshplanr.c:301)

conversion

coplanarity

```
CagdRType CagdEstimateSrfPlanarity(const CagdSrfStruct *Srf)
```

Srf: To measure its coplanarity.

Returns: Coplanarity measure.

Description: ests mesh collinearity by testing the distance of interior points from the plane thru 3 corner points. Returns a relative ratio of deviation from plane relative to its size. Zero means all points are coplanar. If end points are same (no plane can be fit) IRIT_INFNTY is returned.

3.2.440 CagdEvaluateSurfaceVecField (cagd_aux.c:696)

normal

vector field

```
void CagdEvaluateSurfaceVecField(CagdVType Vec,
                                 CagdSrfStruct *VecFieldSrf,
                                 CagdRType U,
                                 CagdRType V)
```

Vec: Where resulting unit length vector is to be saved.

VecFieldSrf: A surface representing a vector field.

U, V: Parameter locations.

Returns: void

Description: Evaluates a vector field surface to a unit size vector. If fails, moves a tad until success. Useful for normal field evaluations.

3.2.441 CagdExtrudeSrf (cagdextr.c:32)

surface constructors

```
CagdSrfStruct *CagdExtrudeSrf(const CagdCrvStruct *CCrv,
                              const CagdVecStruct *Vec)
```

CCrv: To extrude in direction specified by Vec.

Vec: Direction as well as magnitude of extrusion.

Returns: An extrusion surface with Orders of the original Crv order and 2 in the extrusion direction.

Description: Constructs an extrusion surface in the Vector direction for the given profile curve. Input curve can be either a B-spline or a Bezier curve and the resulting output surface will be of the same type.
See also: CagdZTwistExtrudeSrf, CagdExtrudeSrfList,

3.2.442 CagdExtrudeSrfList (cagdextr.c:125)

surface constructors

```
CagdSrfStruct *CagdExtrudeSrfList(const CagdCrvStruct *Crvs,
                                  const CagdVecStruct *Vec)
```

Crvs: To extrude in direction specified by Vec.

Vec: Direction as well as magnitude of extrusion.

Returns: An extrusion surface with Orders of the original Crv order and 2 in the extrusion direction.

Description: Constructs an extrusion surface in the Vector direction for the given profile curve. Input curve can be either a B-spline or a Bezier curve and the resulting output surface will be of the same type.
See also: CagdZTwistExtrudeSrf.,

3.2.443 CagdFitPlaneThruCtlPts (mshplanr.c:37)

plane fit

```
CagdRType CagdFitPlaneThruCtlPts(CagdPlaneStruct *Plane,
                                 CagdPointType PType,
                                 CagdRType * const *Points,
                                 int Index1,
                                 int Index2,
                                 int Index3,
                                 int Index4)
```

Plane: To compute and save here.

PType: Point type expected of four points. Must be E2 or E3.

Points: Point array where to look for the four points.

Index1, Index2, Index3, Index4: Four indices of the points.

Returns: Measure the distance between the data points, Negative value if fitting failed.

Description: Fits a plane through the four points from Points indices Index?. Points may be either E2 or E3 only. Returns 0.0 if failed to fit a plane, otherwise a measure on the size of the mesh data (distance between points) is returned.

3.2.444 CagdForceClosedCrv (cagd1gen.c:1461)

```
void CagdForceClosedCrv(CagdCrvStruct *Crv)
```

Crv: To force into a closed loop, in place.

Returns: void

Description: Forces the curve to be a closed loop, by making the first and last point exactly the same.
See also: CagdAreClosedCrvs, CagdIsClosedSrf, CagdIsZeroLenCrv, CagdIsClosedCrv,

3.2.445 CagdGetCrvsCommonPt (cagdbsum.c:470)

```
int CagdGetCrvsCommonPt(const CagdCrvStruct *Crv1,
                        const CagdCrvStruct *Crv2,
                        CagdPType Pt)
```

Crv1: First curve.

Crv2: Second curve.

Pt: A pointer to Euclidean point, the meeting point will be stored here.

Returns: 0b00 - 0 if the point is the beginning of Crv1/2. 0b01 - 1 if the point is the end of Crv1 and the beginning of Crv2. 0b10 - 2 if the point is the end of Crv2 and the beginning of Crv1. 0b11 - 3 if the point is the end of Crv1/2. -1 if the two curves didn't meet.

Description: The function takes two curves and extract the common point between these two curves and store it in Pt, if any.
See also: MvarGetSrfsCommonCrvs, CagdOneSidedBoolSumSrf,

3.2.446 CagdGetPlnrSrfJacobian (crv_quad.c:152)

```
MvarMVStruct *CagdGetPlnrSrfJacobian(const CagdSrfStruct *Srf)
```

Srf: The given parametric surface.

Returns: The surface Jacobian univariate function.

Description: Computes the Jacobian univariate function to the given surface.
See also:

3.2.447 CagdGetPlnrSrfJacobianMinMax (crv_quad.c:193)

```
void CagdGetPlnrSrfJacobianMinMax(const CagdSrfStruct *Srf,
                                  CagdRType *JMin,
                                  CagdRType *JMax,
                                  int ComputePrecisely)
```

Srf: The given surface.

JMin, JMax: Pointers to the desired extreme values. (Not computed if NULL).

ComputePrecisely: If True computes the exact Jacobian extrema. Otherwise, approximate the extrema based on the bounding box of the Jacobian's control mesh

Returns: void

Description: Computes extreme values of the determinant of the Jacobian of a given planar surface.
See also: CagdQuadSrfJacobianWeight.,

3.2.448 CagdIChooseK (cbzreval.c:402)

evaluation

combinatorics

```
CagdRType CagdIChooseK(int i, int k)
```

i, k: Coefficients of i choose k.

Returns: Result of i choose k, in floating point, to prevent from overflows.

Description: Evaluates the following (in floating point arithmetic):

$$\binom{k}{i} = \frac{k!}{i! * (k - i)!}$$

3.2.449 CagdIcKJcMIJcKM (cbzreval.c:445)

evaluation

combinatorics

```
CagdRType CagdIcKJcMIJcKM(int i, int j, int k, int m)
```

i, j, k, m: Coefficients of i choose k and j choose m.

Returns: Result of the above expression, in floating point, to prevent from overflows.

Description: Evaluates the following (in floating point arithmetic):

$$\binom{k}{i} * \binom{m}{j} / \binom{k + m}{i + j}$$

3.2.450 CagdIgnoreNonPosWeightBBox (cagdbbox.c:35)

bbox

bounding box

```
CagdBType CagdIgnoreNonPosWeightBBox(CagdBType IgnoreNonPosWeightBBox)
```

IgnoreNonPosWeightBBox: TRUE to ignore negative and zero weight control points in the bounding box computation.

Returns: old value.

Description: Computes a bounding box for a freeform curve.
See also: CagdCrvBBox, CagdSrfBBox, GMBBSetBBoxPrecise, SymbCrvPosNegWeights,

3.2.451 CagdInsertInterPoints (cagd_cci.c:774)

```
CagdPtStruct *CagdInsertInterPoints(CagdRType t1,
                                    CagdRType t2,
                                    CagdRType Eps,
                                    CagdPtStruct **InterList)
```

t1, t2: New parameter values to insert to InterList list.

Eps: Accuracy of insertion computation.

InterList: List to update, in place.

Returns: A reference to the constructed and inserted point.

Description: Insert t1/t2 values into InterList, provided no equal t1/t2 value exists already in the list. List is in ascending order with respect to t1.
See also: CagdInsertInterPointReset,

3.2.452 CagdIsClosedCrv (cagd1gen.c:1410)

```
CagdBType CagdIsClosedCrv(const CagdCrvStruct *Crv)
```

Crv: To test for a closed loop.

Returns: TRUE if closed, FALSE otherwise.

Description: Returns TRUE if the curve is a closed loop.
See also: CagdAreClosedCrvs, CagdIsClosedSrf, CagdIsZeroLenCrv, CagdForceClosedCrv,

3.2.453 CagdIsClosedSrf (cagd1gen.c:1647)

```
CagdBType CagdIsClosedSrf(const CagdSrfStruct *Srf, CagdSrfDirType Dir)
```

Srf: To test for a closed boundary.

Dir: Direction to test if surface is closed. Either U or V.

Returns: TRUE if closed, FALSE otherwise.

Description: Returns TRUE if the surface is closed in the given direction. That is, if the min curve boundary equal the max curve boundary
See also: CagdIsClosedCrv,

3.2.454 CagdIsCrvInsideCH (cagdbbox.c:1016)

```
int CagdIsCrvInsideCH(const CagdCrvStruct *Crv,
                      const IrtE2PtStruct *CHPts,
                      int NumCHPts)
```

Crv: The input curve.

CHPts: Points in convex hull in GMR2Struct form.

NumCHPts: Number of Points in the convex hull.

Returns: TRUE if successful, FALSE otherwise.

Description: Identifying whether the given curve is inside the convex hull. Test is conducted by verifying that all the control points of Crv are inside the convex hull.
See also: CagdIsCrvInsideCirc, GMConvexHull,

3.2.455 CagdIsCrvInsideCirc (cagdbbox.c:972)

```
CagdBType CagdIsCrvInsideCirc(const CagdCrvStruct *Crv,
                              const CagdRType Center[2],
                              CagdRType Radius)
```

Crv: Curve to test for containment in the circle.

Center: Center of the circle to test against.

Radius: Radius of the circle to test against.

Returns: TRUE if Crv is indeed inside the circle, FALSE otherwise.

Description: Tests if a Crv is contained in the given prescribed circle. Test is conducted by verifying that all the control points of Crv are inside the circle.
See also: CagdIsCrvInsideCH, MvarIsCrvInsideCirc,

3.2.456 CagdIsZeroLenCrv (cagd1gen.c:1386)

```
CagdBType CagdIsZeroLenCrv(const CagdCrvStruct *Crv, CagdRType Eps)
```

Crv: Curve to examine.

Eps: Epsilon to consider the curve degenerate below this length.

Returns: TRUE if zero length, FALSE otherwise.

Description: Checks if the given curve is degenerate and have almost zero length.
See also: CagdIsClosedCrv, CagdIsZeroLenSrfBndry,

3.2.457 CagdIsZeroLenSrfBndry (cagd1gen.c:1597)

```
CagdBType CagdIsZeroLenSrfBndry(const CagdSrfStruct *Srf,
                                CagdSrfBndryType Bndry,
                                CagdRType Eps)
```

Srf: Surface to examine its boundary.

Bndry: The boundary, out of the four of Srf, to examine.

Eps: Epsilon to consider the curve degenerate below this length.

Returns: TRUE if zero length, FALSE otherwise.

Description: Checks if the prescribed boundary of the given surface is degenerate and has an almost zero length.
See also: CagdIsClosedCrv, CagdIsZeroLenCrv,

3.2.458 CagdLimitCrvArcLen (cagdcmrg.c:1569)

arc length

```
CagdCrvStruct *CagdLimitCrvArcLen(const CagdCrvStruct *Crv, CagdRType MaxLen)
```

Crv: To subdivide into curves, each with control polygon length less than MaxLen.

MaxLen: Maximum length of control polygon to allow.

Returns: List of subdivided curves from Crv, each with control polygon size of less than MaxLen.

Description: Subdivides the given curves to curves, each with size of control polygon less than or equal to MaxLen. Returned is a list of curves.
See also: CagdCrvArcLenPoly, CagdSrfAvgArgLenMesh, CagdCrvAreaPoly,

3.2.459 CagdLinCrvLinCrvInter (cagd_cci.c:228)

```
CagdPtStruct *CagdLinCrvLinCrvInter(const CagdCrvStruct *Crv1,
                                    const CagdCrvStruct *Crv2)
```

Crv1, Crv2: Two piecewise linear curves to compute their intersection points.

Returns: List of intersection points. Each points would contain (u1, u2, 0.0).

Description: Computes all the intersection points of two piecewise liner curves.
See also:

3.2.460 CagdLineFitToPts (cbsp_int.c:1873)

interpolation

least square approximation

```
CagdRType CagdLineFitToPts(CagdPtStruct *PtList,
                           CagdVType LineDir,
                           CagdPType LinePos)
```

PtList: List of points to interpolate/least square approximate.

LineDir: A unit vector of the line.

LinePos: A point on the computed line.

Returns: Average distance between a point and the fitted line, or IRIT_INFNTY if failed.

Description: Given set of points, PtList, fits a line using least squares fit to them.
See also: MvarLineFitToPts,

3.2.461 CagdListAppend (cagd2gen.c:992)

```
VoidPtr CagdListAppend(VoidPtr List1, VoidPtr List2)
```

List1, List2: Two lists of cagd objects to append, in place.

Returns: Appended list.

Description: Appends two lists, in place.

3.2.462 CagdListDelNth (cagd2gen.c:1092)

```
VoidPtr CagdListDelNth(VoidPtr *List, int n)
```

List: Reference to list to delete the nth item from the list.

n: The item to delete from List.

Returns: Deleted item from the list.

Description: Delete the nth item in the given list, or NULL if error. First item is item n = 0.
See also: CagdListLast, CagdListFind, CagdListNth,

3.2.463 CagdListFind (cagd2gen.c:1023)

```
IrtBType CagdListFind(const VoidPtr List, const VoidPtr Item)
```

List: The list of elements.

Item: Item to find.

Returns: TRUE if found, FALSE otherwise.

Description: Search for an element in a given list. A pointer comparison is used.

3.2.464 CagdListLast (cagd2gen.c:915)

```
VoidPtr CagdListLast(const VoidPtr List)
```

List: To return its last element.

Returns: Last element.

Description: Returns the last element of given list of cagd library objects.

3.2.465 CagdListLength (cagd2gen.c:968)

```
int CagdListLength(const VoidPtr List)
```

List: List of cagd objects.

Returns: Length of list.

Description: Computes the length of a list.

3.2.466 CagdListNth (cagd2gen.c:1055)

```
VoidPtr CagdListNth(const VoidPtr List, int n)
```

List: List to fetch the nth item.

n: The item to fetch from List.

Returns: Fetched item or NULL of error.

Description: Gets the nth item in the given list, or NULL if error. First item is item n = 0.
See also: CagdListLast, CagdListFind, CagdListDelNth,

3.2.467 CagdListPrev (cagd2gen.c:943)

```
VoidPtr CagdListPrev(const VoidPtr List, const VoidPtr Item)
```

List: To seek the previous element to Item.

Item: Item to seek its prev.

Returns: Previous item to Item or NULL if not found (or Item is the first item in List).

Description: Returns the element previous to given Item in List of cagd library objs.

3.2.468 CagdListReverse (cagd2gen.c:880)

reverse

```
VoidPtr CagdListReverse(VoidPtr List)
```

List: To be reversed.

Returns: Reversed list.

Description: Reverses a list of cagd library objects, in place.

3.2.469 CagdListSort (cagd2gen.c:1138)

```
VoidPtr CagdListSort(VoidPtr List,
                     CagdBType Ascending,
                     CagdCompFuncType SortCmprFunc)
```

List: The linked list of objects with attributes to sort.

Ascending: Ascending (TRUE) or descending (FALSE) order.

SortCmprFunc: Sorting function.

Returns: Head of the sorted list.

Description: Sort a given linked list in place according to some integer attribute value.

3.2.470 CagdMakeCrvsCompatible (cagdcmpt.c:40)

compatibility

```
CagdBType CagdMakeCrvsCompatible(CagdCrvStruct **Crv1,
                                 CagdCrvStruct **Crv2,
                                 CagdBType SameOrder,
                                 CagdBType SameKV)
```

Crv1, Crv2: Two curves to be made compatible, in place.

SameOrder: If TRUE, this routine make sure they share the same order.

SameKV: If TRUE, this routine make sure they share the same knot vector and hence continuity. *

Returns: TRUE if successful, FALSE otherwise.

Description: Given two curves, makes them compatible by:

1. Coercing their point type to be the same.
2. Making them have the same curve type.
3. Raising the degree of the lower one to be the same as the higher.
4. Refining them to a common knot vector (If Bspline and SameOrder).

Note 3 is performed if SameOrder TRUE, 4 if SameKV TRUE. Both curves are modified IN PLACE.
See also: CagdMakeCrvsCompatible2,

3.2.471 CagdMakeCrvsCompatible2 (cagdcmpt.c:94)

compatibility

```
CagdBType CagdMakeCrvsCompatible2(CagdCrvStruct **Crv1,
                                  CagdCrvStruct **Crv2,
                                  CagdBType SameOrder,
                                  CagdBType SameKV)
```

Crv1, Crv2: Two curves to be made compatible, in place.

SameOrder: If TRUE, this routine make sure they share the same order.

SameKV: If TRUE, this routine make sure they share the same knot vector and hence continuity. *

Returns: TRUE if successful, FALSE otherwise.

Description: Given two curves, makes them compatible by:

1. Making them have the same curve type.
2. Raising the degree of the lower one to be the same as the higher.
3. Refining them to a common knot vector (If Bspline and SameOrder).

Note 2 is performed if SameOrder TRUE, 3 if SameKV TRUE. Both curves are modified IN PLACE. Note here point type can be different and will stay different.

See also: CagdMakeCrvsCompatible,

3.2.472 CagdMakeSrfsCompatible (cagdcmpt.c:240)

compatibility

```
CagdBType CagdMakeSrfsCompatible(CagdSrfStruct **Srf1,
                                 CagdSrfStruct **Srf2,
                                 CagdBType SameUOrder,
                                 CagdBType SameVOrder,
                                 CagdBType SameUKV,
                                 CagdBType SameVKV)
```

Srf1, Srf2: Two surfaces to be made compatible, in place.

SameUOrder: If TRUE, this routine make sure they share the same U order.

SameVOrder: If TRUE, this routine make sure they share the same V order.

SameUKV: If TRUE, this routine make sure they share the same U knot vector and hence continuity. *

SameVKV: If TRUE, this routine make sure they share the same V knot vector and hence continuity.

Returns: TRUE if successful, FALSE otherwise.

Description: Given two surfaces, makes them compatible by:

1. Coercing their point type to be the same.
2. Making them have the same curve type.
3. Raising the degree of the lower one to be the same as the higher.
4. Refining them to a common knot vector (If Bspline and SameOrder).

Note 3 is performed if SameOrder TRUE, 4 if SameKV TRUE. Both surface are modified IN PLACE.

See also: CagdMakeSrfsCompatible2,

3.2.473 CagdMakeSrfsCompatible2 (cagdcmpt.c:302)

compatibility

```
CagdBType CagdMakeSrfsCompatible2(CagdSrfStruct **Srf1,
                                  CagdSrfStruct **Srf2,
                                  CagdBType SameUOrder,
                                  CagdBType SameVOrder,
                                  CagdBType SameUKV,
                                  CagdBType SameVKV)
```

Srf1, Srf2: Two surfaces to be made compatible, in place.

SameUOrder: If TRUE, this routine make sure they share the same U order.

SameVOrder: If TRUE, this routine make sure they share the same V order.

SameUKV: If TRUE, this routine make sure they share the same U knot vector and hence continuity. *

SameVKV: If TRUE, this routine make sure they share the same V knot vector and hence continuity.

Returns: TRUE if successful, FALSE otherwise.

Description: Given two surfaces, makes them compatible by:

1. Making them have the same curve type.
2. Raising the degree of the lower one to be the same as the higher.
3. Refining them to a common knot vector (If Bspline and SameOrder).

Note 2 is performed if SameOrder TRUE, 3 if SameKV TRUE. Both surface are modified IN PLACE. Note here point type can be different and will stay different.

See also: CagdMakeSrfsCompatible,

3.2.474 CagdMatTransCenter (cagd2gen.c:1755)

transformations

```
void CagdMatTransCenter(CagdRType **Points,
                        int Len,
                        int MaxCoord,
                        int IsNotRational,
                        IrtHmgnMatType Mat)
```

Points: To be transformed. Array of vectors.

Len: Of vectors of Points.

MaxCoord: Maximum number of coordinates to be found in Points.

IsNotRational: TRUE if input has no weights (is not rational).

Mat: Transformation.

Returns: void

Description: Applies a transform, in place, to given set of points Points which is given as array of vectors, each vector of length Len. The transform is applied around the center of the geometry at hand. Array Points optionally contains (if !IsNotRational) in Points[0] the weights coefficients and in Points[i] the coefficients of axis i, up to and include MaxCoord (X = 1, Y = 2, etc.). Points are transformed as prescribed by Mat.

See also: CagdSrfScale, CagdCrvScale, CagdTransform, CagdScaleCenter,

3.2.475 CagdMatTransform (cagd2gen.c:2004)

scaling

rotation

translation

transformations

```
void CagdMatTransform(CagdRType **Points,
                      int Len,
                      int MaxCoord,
                      CagdBType IsNotRational,
                      CagdMType Mat)
```

Points: To be affinely transformed. Array of vectors.

Len: Of vectors of Points.

MaxCoord: Maximum number of coordinates to be found in Points.

IsNotRational: Do we have weights as vector Points[0]?

Mat: Defining the transformation.

Returns: void

Description: Applies an homogeneous transformation, in place, to given set of points Points which as array of vectors, each vector of length Len. Array Points optionally contains (if !IsNotRational) in Points[0] the weights coefficients and in Points[i] the coefficients of axis i, up to and include MaxCoord (X = 1, Y = 2, etc.).

See also: CagdTransform, CagdSrfMatTransform, CagdCrvMatTransform,

3.2.476 CagdMatTransform2 (cagd2gen.c:2076)

scaling
rotation
translation
transformations

```
void CagdMatTransform2(CagdRType **NewPoints,
                       const CagdRType **OldPoints,
                       int Len,
                       int MaxCoord,
                       CagdBType IsNotRational,
                       CagdMType Mat)
```

NewPoints: To be affinely transformed. Array of E3/P3 vectors.

OldPoints: Original set of E3/P3 points to be transformed.

Len: Of vectors of Points.

MaxCoord: Maximal coordinate to transform.

IsNotRational: Do we have weights as vector Points[0]?

Mat: Defining the transformation.

Returns: void

Description: Applies an homogeneous transformation, in place, to given set of points Points which in array of E3/P3 vectors, each vector of length Len. Array Points optionally contains (if !IsNotRational) in Points[0] the weights coefficients and in Points[i] the coefficients of axis i, up to and include MaxCoord (X = 1, Y = 2, etc.).
See also: CagdTransform, CagdMatTransform,

3.2.477 CagdMatchBisectorNorm (crvmatch.c:355)

correspondence
matching

```
CagdRType CagdMatchBisectorNorm(const CagdVType T1,
                                const CagdVType T2,
                                const CagdVType P1,
                                const CagdVType P2)
```

T1: A pointer to unit tangent to the first curve at i-th point.

T2: A pointer to unit tangent to the second curve at j-th point.

P1: A pointer to value of the first curve at i-th point.

P2: A pointer to value of the second curve at j-th point.

Returns: A numeric matching value, the smaller the better.

Description: Computes the bisector norm to the matching.
See also: CagdMatchDistNorm, CagdMatchRuledNorm, CagdMatchMorphNorm, , CagdMatchingTwoCurves,

3.2.478 CagdMatchDistNorm (crvmatch.c:322)

correspondance
matching

```
CagdRType CagdMatchDistNorm(const CagdVType T1,
                            const CagdVType T2,
                            const CagdVType P1,
                            const CagdVType P2)
```

T1: A pointer to unit tangent to the first curve at i-th point.

T2: A pointer to unit tangent to the second curve at j-th point.

P1: A pointer to value of the first curve at i-th point.

P2: A pointer to value of the second curve at j-th point.

Returns: A numeric matching value, the smaller the better.

Description: Computes the distance norm to the matching, || P1 - P2 ||.
See also: CagdMatchBisectorNorm, CagdMatchRuledNorm, CagdMatchMorphNorm, , CagdMatchingTwoCurves,

3.2.479 CagdMatchMorphNorm (crvmatch.c:407)

correspondence

matching

```
CagdRType CagdMatchMorphNorm(const CagdVType T1,
                             const CagdVType T2,
                             const CagdVType P1,
                             const CagdVType P2)
```

T1: A pointer to unit tangent to the first curve at i-th point.

T2: A pointer to unit tangent to the second curve at j-th point.

P1: A pointer to value of the first curve at i-th point.

P2: A pointer to value of the second curve at j-th point.

Returns: -1 for no matching or the cost of the matching for the point between zero and one.

Description: Computes the default morphing norm to the matching, 1.0 - < T1, T2 >.
See also: CagdMatchDistNorm, CagdMatchBisectorNorm, CagdMatchRuledNorm, , CagdMatchingTwoCurves,

3.2.480 CagdMatchRuled2Norm (crvmatch.c:490)

correspondence

matching

```
CagdRType CagdMatchRuled2Norm(const CagdVType T1,
                              const CagdVType T2,
                              const CagdVType P1,
                              const CagdVType P2)
```

T1: A pointer to unit tangent to the first curve at i-th point.

T2: A pointer to unit tangent to the second curve at j-th point.

P1: A pointer to value of the first curve at i-th point.

P2: A pointer to value of the second curve at j-th point.

Returns: -1 for no matching or the cost of the matching for the point between zero and one, the minimum (non negative) the better.

Description: Computes the default ruled norm to the matching, < T1 x (P2 - P1), T2 x (P2 - P1) > must be non negative and then the norm is 1.0 - MIN(|| T1 x (P2 - P1) ||^2, || T2 x (P2 - P1) ||^2) / || P2 - P1 ||^2.
See also: CagdMatchDistNorm, CagdMatchBisectorNorm, CagdMatchMorphNorm, CagdMatchingTwoCurves, CagdMatchRuledNorm,

3.2.481 CagdMatchRuledNorm (crvmatch.c:445)

correspondence

matching

```
CagdRType CagdMatchRuledNorm(const CagdVType T1,
                             const CagdVType T2,
                             const CagdVType P1,
                             const CagdVType P2)
```

T1: A pointer to unit tangent to the first curve at i-th point.

T2: A pointer to unit tangent to the second curve at j-th point.

P1: A pointer to value of the first curve at i-th point.

P2: A pointer to value of the second curve at j-th point.

Returns: -1 for no matching or the cost of the matching for the point between zero and one, the minimum (non negative) the better.

Description: Computes the default ruled norm to the matching, < T1 x (P2 - P1), T2 x (P2 - P1) > must be non negative and then the norm is 1.0 - < T1, T2 >.
See also: CagdMatchDistNorm, CagdMatchBisectorNorm, CagdMatchMorphNorm, CagdMatchingTwoCurves, CagdMatchRuledNorm,

3.2.482 CagdMatchingFixCrv (crvmatch.c:900)

correspondance
matching

```
void CagdMatchingFixCrv(CagdCrvStruct *Crv)
```

Crv: The input curve to be fixed.

Returns: void

Description: Fix the input curve to be monotone.

3.2.483 CagdMatchingFixVector (crvmatch.c:849)

correspondance
matching

```
void CagdMatchingFixVector(int *OldVec, CagdRType *NewVec, int Len)
```

OldVec: The input vector.

NewVec: The output (fixed) vector

Len: The length of the vector.

Returns: void

Description: Fix the input integer vector, so that NewVec is increasingly monotone.

3.2.484 CagdMatchingPolyTransform (crvmatch.c:934)

correspondance
matching

```
void CagdMatchingPolyTransform(CagdRType **Poly,
                               int Len,
                               CagdRType NewBegin,
                               CagdRType NewEnd)
```

Poly: A pointer to points to change.

Len: The length of the input poly.

NewBegin: The new begin.

NewEnd: The new end.

Returns: void

Description: Affine transform a set of points, so its new end locations are NewBegin and NewEnd, respectively.

3.2.485 CagdMatchingTwoCurves (crvmatch.c:1023)

correspondence
matching

```
CagdCrvStruct *CagdMatchingTwoCurves(const CagdCrvStruct *Crv1,
                                     const CagdCrvStruct *Crv2,
                                     int Reduce,
                                     int SampleSet,
                                     int ReparamOrder,
                                     int RotateFlag,
                                     int AllowNegativeNorm,
                                     int ReturnReparamFunc,
                                     CagdBType MinimizeMaxError,
                                     CagdMatchNormFuncType MatchNormFunc)
```

Crv1: The first curve.

Crv2: The second curve.

Reduce: The degrees of freedom of the reparametrization curve. The larger this number is, the better the reparametrization will be at the cost of more computation. Must be less than SampleSet

SampleSet: Number of samples the two curves are sampled at. The larger this number is, the better the reparametrization will be

ReparamOrder: Order of reparametrization curve.

RotateFlag: use or not use rotation in finding best matching

AllowNegativeNorm: If TRUE, negative norms are locally allowed.

ReturnReparamFunc: If TRUE, return the reparamterization function instead of Crv2 reparametrized.

MinimizeMaxError: TRUE for minimizing maximal error, FALSE to minimize the error's sum over the entire domain.

MatchNormFunc: A pointer to the matching norm.

Returns: The second curve, Crv2, after reparametrization that matches the first curve. Error of returned result is saved in an "_Error" attr.

Description: Gets two freeform curves, Crv1, nd Crv2, computes a new parametrization to Crv2 using composition between Crv2 and a computed reparametrization that establishes a matching correspondance between Crv1 and Crv2.

See also: CagdMatchDistNorm, CagdMatchBisectorNorm, CagdMatchMorphNorm, CagdMatchRuledNorm,

3.2.486 CagdMatchingVectorTransform (crvmatch.c:966)

correspondance

matching

```
void CagdMatchingVectorTransform(CagdRType *Vec,
                                 CagdRType NewBegin,
                                 CagdRType NewEnd,
                                 int Len)
```

Vec: The input and the output vector.

NewBegin: The new begin.

NewEnd: The new end.

Len: The length of the input vector.

Returns: void

Description: Affine transform a set of vectors, so its new end locations are NewBegin and NewEnd, respectively.

3.2.487 CagdMergeCrvCrv (cagdcmrg.c:71)

merge

```
CagdCrvStruct *CagdMergeCrvCrv(const CagdCrvStruct *CCrv1,
                               const CagdCrvStruct *CCrv2,
                               CagdBType InterpolateDiscont,
                               CagdRType MergeEps)
```

CCrv1: To connect to Crv1's starting location at its end.

CCrv2: To connect to Crv2's end location at its start.

InterpolateDiscont: If TRUE, linearly interpolate discontinuity.

MergeEps: Epsilon to consider end points the same.

Returns: The merged curve.

Description: Merges two curves by connecting the end of Crv1 to the beginning of Crv2. If the end of Crv1 is identical to the beginning of Crv2 then the result is as expected. However, if the curves do not meet, their end points are linearly interpolated if InterpolateDiscont is TRUE or simply blended out in a freeform shape if InterpolateDiscont is FALSE. If one of the curves is NULL, a copy of other curve is simply returned.

See also: CagdMergeCtlPtCtlPt, CagdMergePtPt, CagdMergePtCrv, CagdMergeCrvPt, , CagdMergeCrvList, CagdMergeCrvCtlPt, CagdMergeCtlPtCrv,

3.2.488 CagdMergeCrvCtlPt (cagdcmrg.c:1122)

merge

```
CagdCrvStruct *CagdMergeCrvCtlPt(const CagdCrvStruct *Crv,
                                 const CagdCtlPtStruct *CtlPt)
```

Crv: To connect to CtlPt its end.

CtlPt: To connect to Crv's end point.

Returns: The merged curve.

Description: Merges a curve and a point by connecting the end of Crv to CtlPt, using a linear segment.

See also: CagdMergeCtlPtCtlPt, CagdMergePtPt, CagdMergePtCrv, CagdMergeCrvCrv, , CagdMergeCrvPt, CagdMergePtCrv,

3.2.489 CagdMergeCrvList (cagdcmrg.c:190)

merge

```
CagdCrvStruct *CagdMergeCrvList(const CagdCrvStruct *CrvList,
                                CagdBType InterpDiscont,
                                CagdRType MergeEps)
```

CrvList: To connect into one curve.

InterpDiscont: If TRUE, linearly interpolate discontinuity.

MergeEps: Epsilon to consider end points the same.

Returns: The merged curve.

Description: Merges a list of curves by connecting the end of one curve to the beginning of the next. I.e. curves are assumed in order.
See also: CagdMergeCtlPtCtlPt, CagdMergePtPt, CagdMergePtCrv, CagdMergeCrvCrv, CagdMergeCrvCrv, CagdMergeCrvList2, CagdMergeCrvList3, , CagdMergeCrvCtlPt, CagdMergeCtlPtCrv,

3.2.490 CagdMergeCrvList1 (cagdcmrg.c:235)

merge

```
CagdCrvStruct *CagdMergeCrvList1(CagdCrvStruct *CrvList,
                                 IrtRType Tolerance,
                                 CagdBType InterpDiscont)
```

CrvList: To connect into larger curves.

Tolerance: To consider two end points the same.

InterpDiscont: If TRUE, linearly interpolate discontinuity.

Returns: The merged curves.

Description: Merges a list of curves by connecting end points that are (almost) the same, in place.
See also: CagdMergeCtlPtCtlPt, CagdMergePtPt, CagdMergePtCrv, CagdMergeCrvCrv, CagdMergeCrvCrv, CagdMergeCrvList, , CagdMergeCrvCtlPt, CagdMergeCtlPtCrv,

3.2.491 CagdMergeCrvList2 (cagdcmrg.c:335)

merge

```
CagdCrvStruct *CagdMergeCrvList2(CagdCrvStruct *CrvList,
                                 IrtRType Tolerance,
                                 CagdBType InterpDiscont)
```

CrvList: To connect into larger curves, and free in place.

Tolerance: To consider two end points the same.

InterpDiscont: If TRUE, linearly interpolate discontinuity.

Returns: The merged curves.

Description: Merges a list of curves by connecting end points that are (almost) the same, in place.
See also: CagdMergeCtlPtCtlPt, CagdMergePtPt, CagdMergePtCrv, CagdMergeCrvCrv, CagdMergeCrvCrv, CagdMergeCrvList, CagdMergeCrvList3, , CagdMergeCrvCtlPt, CagdMergeCtlPtCrv,

3.2.492 CagdMergeCrvList3 (cagdcmrg.c:583)

merge

```
CagdCrvStruct *CagdMergeCrvList3(CagdCrvStruct *CrvList,
                                 IrtRType Tolerance,
                                 CagdBType InterpDiscont)
```

CrvList: To connect into larger curves. Expected to be open ended.

Tolerance: To consider two end points the same.

InterpDiscont: If TRUE, linearly interpolate discontinuity.

Returns: The merged curves, or NULL if error.

Description: Merges a list of curves by connecting end points that are (almost) the same, in place. Uses the generic merging package in geom_lib.
See also: CagdMergeCtlPtCtlPt, CagdMergePtPt, CagdMergePtCrv, CagdMergeCrvCrv, CagdMergeCrvCrv, CagdMergeCrvList, CagdMergeCrvList2, , CagdMergeCrvCtlPt, CagdMergeCtlPtCrv, GMMergeSameGeometry,

3.2.493 CagdMergeCrvPt (cagdcmrg.c:974)

merge

```
CagdCrvStruct *CagdMergeCrvPt(const CagdCrvStruct *Crv, const CagdPtStruct *Pt)
```

Crv: To connect to Pt its end.

Pt: To connect to Crv's end point.

Returns: The merged curve.

Description: Merges a curve and a point by connecting the end of Crv to Pt, using a linear segment.
See also: CagdMergeCtlPtCtlPt, CagdMergePtPt, CagdMergePtCrv, CagdMergeCrvCrv, , CagdMergeCrvCtlPt, CagdMergeCtlPtCrv,

3.2.494 CagdMergeCtlPtCrv (cagdcmrg.c:1147)

merge

```
CagdCrvStruct *CagdMergeCtlPtCrv(const CagdCtlPtStruct *CtlPt,
                                 const CagdCrvStruct *Crv)
```

CtlPt: To connect to Crv's starting point.

Crv: To connect to CtlPt its starting point.

Returns: The merged curve.

Description: Merges a point and a curve by connecting CtlPt to starting point of Crv, using a linear segment.
See also: CagdMergeCtlPtCtlPt, CagdMergePtPt, CagdMergeCrvPt, CagdMergeCrvCrv, , CagdMergeCrvPt, CagdMergePtCrv,

3.2.495 CagdMergeCtlPtCtlPt (cagdcmrg.c:1314)

merge

```
CagdCrvStruct *CagdMergeCtlPtCtlPt(const CagdCtlPtStruct *Pt1,
                                   const CagdCtlPtStruct *Pt2,
                                   int MinDim)
```

Pt1, Pt2: Two control points to connect using a linear segment.

MinDim: Minimal ctlpts dimension to build the curve with, 2 for E2 or P2, etc.

Returns: The merged curve.

Description: Merges two control points by connecting Pt1 to Pt2, using linear segment.
See also: CagdMergePtPt, CagdMergePtCrv, CagdMergeCrvPt, CagdMergeCrvCrv, CagdMergePtPt2, CagdMergeCrvCtlPt, CagdMergeCtlPtCrv,

3.2.496 CagdMergePointTypes (cagdcoer.c:950)

coercion

```
CagdPointType CagdMergePointTypes(CagdPointType PType1, CagdPointType PType2)
```

PType1, PType2: To point types to find the point type of their union.

Returns: A point type of the union of the spaces of PType1 and PType2.

Description: Returns a point type which spans the spaces of both two given point types.

3.2.497 CagdMergePtCrv (cagdcmrg.c:998)

merge

```
CagdCrvStruct *CagdMergePtCrv(const CagdPtStruct *Pt, const CagdCrvStruct *Crv)
```

Pt: To connect to Crv's starting point.

Crv: To connect to Pt its starting point.

Returns: The merged curve.

Description: Merges a point and a curve by connecting Pt to the starting point of Crv, using a linear segment.
See also: CagdMergeCtlPtCtlPt, CagdMergePtPt, CagdMergeCrvPt, CagdMergeCrvCrv, , CagdMergeCrvCtlPt, CagdMergeCtlPtCrv,

3.2.498 CagdMergePtPt (cagdcmrg.c:1170)

merge

```
CagdCrvStruct *CagdMergePtPt(const CagdPtStruct *Pt1, const CagdPtStruct *Pt2)
```

Pt1, Pt2: Two points to connect using a linear segment.

Returns: The merged curve.

Description: Merges two points by connecting Pt1 to Pt2, using a linear segment.
See also: CagdMergeCtlPtCtlPt, CagdMergePtCrv, CagdMergeCrvPt, CagdMergeCrvCrv, CagdMergePtPt2, CagdMergePtPtLen, CagdMergeUvUv, , CagdMergeCrvCtlPt, CagdMergeCtlPtCrv,

3.2.499 CagdMergePtPt2 (cagdcmrg.c:1257)

merge

```
CagdCrvStruct *CagdMergePtPt2(const CagdPType Pt1, const CagdPType Pt2)
```

Pt1, Pt2: Two points to connect using a linear segment.

Returns: The merged curve.

Description: Merges two points by connecting Pt1 to Pt2, using a linear segment.
See also: CagdMergeCtlPtCtlPt, CagdMergePtCrv, CagdMergeCrvPt, CagdMergeCrvCrv, CagdMergePtPt, CagdMergePtPtLen, CagdMergeUvUv, , CagdMergeCrvCtlPt, CagdMergeCtlPtCrv,

3.2.500 CagdMergePtPtLen (cagdcmrg.c:1221)

merge

```
CagdCrvStruct *CagdMergePtPtLen(const CagdPtStruct *Pt1,
                                const CagdPtStruct *Pt2,
                                int Len)
```

Pt1, Pt2: Two points to connect using a linear segment.

Len: The dimension of the points.

Returns: The merged curve.

Description: Merges two points by connecting Pt1 to Pt2, using a linear segment.
See also: CagdMergeCtlPtCtlPt, CagdMergePtCrv, CagdMergeCrvPt, CagdMergeCrvCrv, CagdMergePtPt, CagdMergePtPt2, CagdMergeUvUv,

3.2.501 CagdMergeSrfList (cagdsmrg.c:407)

merge

```
CagdSrfStruct *CagdMergeSrfList(const CagdSrfStruct *SrfList,
                                CagdSrfDirType Dir,
                                CagdBType SameEdge,
                                CagdBType InterpolateDiscont)
```

SrfList: To connect into one surface.

Dir: Direction the merge should take place. Either U or V.

SameEdge: If the two surfaces share a common edge.

InterpolateDiscont: If TRUE, linearly interpolate discontinuity.

Returns: The merged surface.

Description: Merges a list of surfaces by connecting the end of one surface to the beginning of the next. See also CagdMergeSrfSrf.

3.2.502 CagdMergeSrfList2 (cagdsmrg.c:784)

merge

```
CagdSrfStruct *CagdMergeSrfList2(CagdSrfStruct *SrfList,
                                 CagdSrfDirType Dir,
                                 IrtRType Tolerance,
                                 CagdBType InterpDiscont)
```

SrfList: To connect into larger surfaces.

Dir: Merger along U or V.

Tolerance: To consider two boundary edges the same.

InterpDiscont: If TRUE, linearly interpolate discontinuity.

Returns: The merged surfaces.

Description: Merges a list of surfaces by connecting boundary edges that are (almost) the same, in place.
See also: CagdMergeSrfSrf, CagdMergeSrfList, , CagdMergeSrfList3U, CagdMergeSrfList3V,

3.2.503 CagdMergeSrfList3U (cagdsmrg.c:1140)

merge

```
CagdSrfStruct *CagdMergeSrfList3U(CagdSrfStruct *SrfList,
                                  IrtRType Tolerance,
                                  CagdBType InterpDiscont)
```

SrfList: To connect into larger surfaces. Expected to be open ended.

Tolerance: To consider two shared boundaries the same.

InterpDiscont: If TRUE, linearly interpolate discontinuity.

Returns: The merged surfaces, or NULL if error.

Description: Merges a list of surfaces by connecting shared boundaries that are (almost) the same, in place. Uses the generic merging package in geom_lib.
See also: CagdMergeSrfSrf, CagdMergeSrfList, CagdMergeSrfList2, , GMMergeSameGeometry, CagdMergeSrfList3V,

3.2.504 CagdMergeSrfList3V (cagdsmrg.c:1369)

merge

```
CagdSrfStruct *CagdMergeSrfList3V(CagdSrfStruct *SrfList,
                                  IrtRType Tolerance,
                                  CagdBType InterpDiscont)
```

SrfList: To connect into larger surfaces. Expected to be open ended.

Tolerance: To consider two shared boundaries the same.

InterpDiscont: If TRUE, linearly interpolate discontinuity.

Returns: The merged surfaces, or NULL if error.

Description: Merges a list of surfaces by connecting shared boundaries that are (almost) the same, in place. Uses the generic merging package in geom_lib.
See also: CagdMergeSrfSrf, CagdMergeSrfList, CagdMergeSrfList2, , GMMergeSameGeometry, CagdMergeSrfList3U,

3.2.505 CagdMergeSrfSrf (cagdsmrg.c:110)

merge

```
CagdSrfStruct *CagdMergeSrfSrf(const CagdSrfStruct *CSrf1,
                               const CagdSrfStruct *CSrf2,
                               CagdSrfDirType Dir,
                               CagdBType SameEdge,
                               CagdBType InterpolateDiscont)
```

CSrf1: To connect to Srf2's starting boundary at its end.

CSrf2: To connect to Srf1's end boundary at its start.

Dir: Direction the merge should take place. Either U or V.

SameEdge: If the two surfaces share a common edge - If considered same edge, SameEdge can be - 0. Edge is not similar. 1. Copy row of 1st surface as the shared row/col. 2. Copy row of 2nd surface as the shared row/col. This is also the default behavior for other SameEdge values. 3. Equally blend respective row/col as shared.

InterpolateDiscont: If TRUE, linearly interpolate discontinuity.

Returns: The merged surface.

Description: Merges two surfaces in the requested direction Dir. If SameEdge, it is assumed last edge of Srf1 is identical to first edge of Srf2 and one row is dropped from new mesh. Otherwise a ruled surface is fit between the two edges.

3.2.506 CagdMergeUvUv (cagdcmrg.c:1283)

merge

```
CagdCrvStruct *CagdMergeUvUv(const CagdUVType UV1, const CagdUVType UV2)
```

UV1, UV2: Two UV coordinates to connect using a linear segment.

Returns: The merged curve.

Description: Merges two UV coordinates by connecting UV1, UV2, using a linear segment.
See also: CagdMergeCtlPtCtlPt, CagdMergePtCrv, CagdMergeCrvPt, CagdMergeCrvCrv, CagdMergePtPt, CagdMergePtPtLen, CagdMergePtPt2,

3.2.507 CagdOneBoolSumSrf (cagdbsum.c:824)

Boolean sum

surface constructors

```
CagdSrfStruct *CagdOneBoolSumSrf(const CagdCrvStruct *BndryCrv)
```

BndryCrv: To be subdivided into four curves for a Boolean sum construction.

Returns: A Boolean sum surface constructed using given curve

Description: Constructs a Boolean sum surface using the single boundary curve. The curve is subdivided into four, equally spaced in parameter space, sub-regions which are used as the four curves to the Boolean sum constructor. See CagdBoolSumSrf.

3.2.508 CagdOneSidedBoolSumSrf (cagdbsum.c:536)

CagdBoolSumSrf

Boolean sum

surface constructor

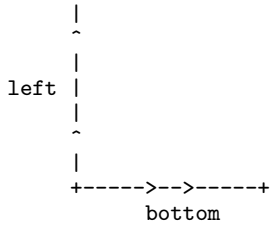
```
CagdSrfStruct *CagdOneSidedBoolSumSrf(const CagdCrvStruct *CCrvLeft,
                                      const CagdCrvStruct *CCrvBottom)
```

CCrvLeft: Left boundary curve of Sided Boolean sum surface to be created.

CCrvBottom: Bottom boundary curve of Sided Boolean sum surface to be created.

Returns: A one-sided Boolean sum surface constructed using the given two curves.

Description: Constructs a one-sided Boolean sum surface using the two provided boundary curve. Curve's must meet at a corner.



See also: MvarTrivarBoolSum, CagdBoolSumSrfRtnl, TrivarSidedBoolSumTwoSrf, , TrivarOneSidedBoolSumThree-Srf,

3.2.509 CagdOrientCrvAsCrv2EndPts (cagd1gen.c:2493)

```
CagdCrvStruct *CagdOrientCrvAsCrv2EndPts(const CagdCrvStruct *OrntCrv,
                                         const CagdCrvStruct *Crv)
```

OrntCrv: Reorient this curve by flipping, so it matches the two end points of Crv.

Crv: Original curve to possibly reverse OrntCrv to fit its end pts.

Returns: New version of OrntCrv or NULL if failed.

Description: Reorient curve OrntCrv so it two end points match Crv.
See also: CagdOrientSrfAsSrf4Corners,

3.2.510 CagdOrientSrfAsSrf4Corners (cagd1gen.c:2569)

```
CagdSrfStruct *CagdOrientSrfAsSrf4Corners(const CagdSrfStruct *OrntSrf,
                                          const CagdSrfStruct *Srf)
```

OrntSrf: Reorient this surface by flipping and reverseing, so it matches the four corners of Srf.

Srf: Original surface to reverse/flip OrntSrf to fit its 4 corners.

Returns: New version of OrntSrf or NULL if failed.

Description: Reorient a new surface OrntSrf so it four corners match Srf.
See also: CagdOrientCrvAsCrv2EndPts,

3.2.511 CagdPDError (cbsp_fit.c:441)

```
static CagdRType CagdPDError(const CagdPType Point, const CagdPType FootPoint)
```

Point: (Xk) Points to calculate the error for.

FootPoint: (P(tk)) Footpoint (the closest point to Xk on the curve)

Returns: PD error.

Description: Computes PD (Point Distance) error for a single point, which is:

$$e_{PD,k} = ||P(tk)-Xk||^2$$

3.2.512 CagdPeriodicCrvNew (cagd1gen.c:131)

allocation

```
CagdCrvStruct *CagdPeriodicCrvNew(CagdGeomType GType,
                                  CagdPointType PType,
                                  int Length,
                                  CagdBType Periodic)
```

GType: Type of geometry the curve should be - Bspline, Bezier etc.

PType: Type of control points (E2, P3, etc.).

Length: Number of control points

Periodic: Is this curve periodic?

Returns: An uninitialized freeform curve.

Description: Allocates the memory required for a new, possibly periodic, curve.
See also: BzrCrvNew, BspCrvNew, BspPeriodicCrvNew, CagdCrvNew, TrimCrvNew,

3.2.513 CagdPeriodicSrfNew (cagd1gen.c:233)

allocation

```
CagdSrfStruct *CagdPeriodicSrfNew(CagdGeomType GType,
                                  CagdPointType PType,
                                  int ULength,
                                  int VLength,
                                  CagdBType UPeriodic,
                                  CagdBType VPeriodic)
```

GType: Type of geometry the surface should be - Bspline, Bezier etc.

PType: Type of control points (E2, P3, etc.).

ULength: Number of control points in the U direction.

VLength: Number of control points in the V direction.

UPeriodic: Is this surface periodic in the U direction?

VPeriodic: Is this surface periodic in the V direction?

Returns: An uninitialized freeform surface.

Description: Allocates the memory required for a new, possibly periodic, surface.
See also: BzrSrfNew, BspSrfNew, BspPeriodicSrfNew, CagdSrfNew, TrimSrfNew,

3.2.514 CagdPlaneArrayFree (cagd2gen.c:716)

free

```
void CagdPlaneArrayFree(CagdPlaneStruct *PlaneArray, int Size)
```

PlaneArray: To be deallocated.

Size: Of the deallocated array.

Returns: void

Description: Deallocates and frees an array of plane structure.

3.2.515 CagdPlaneArrayNew (cagd1gen.c:489)

allocation

```
CagdPlaneStruct *CagdPlaneArrayNew(int Size)
```

Size: Size of Plane array to allocate.

Returns: An array of Plane structures of size Size.

Description: Allocates and resets all slots of an array of Plane structures.

3.2.516 CagdPlaneCopy (cagd1gen.c:1049)

copy

```
CagdPlaneStruct *CagdPlaneCopy(const CagdPlaneStruct *Plane)
```

Plane: To be copied.

Returns: A duplicate of Plane.

Description: Allocates and copies all slots of a Plane structure.

3.2.517 CagdPlaneCopyList (cagd2gen.c:99)

copy

```
CagdPlaneStruct *CagdPlaneCopyList(const CagdPlaneStruct *PlaneList)
```

PlaneList: To be copied.

Returns: A duplicated list of planes.

Description: Allocates and copies a list of plane structures.

3.2.518 CagdPlaneFitToPts (cbsp_int.c:1965)

approximation

```
CagdRType CagdPlaneFitToPts(const CagdPtStruct *PtList,
                            IrtPlnType Pln,
                            IrtVecType MVec,
                            IrtPtType Cntr,
                            IrtRType *CN)
```

PtList: List of points to fit an approximated plane.

Pln: The fitted plane's coefficients.

MVec: Direction in the plane with the major change.

Cntr: The centroid of the data.

CN: Condition number of the fitting. The smaller this number is the more planar the input is.

Returns: Average distance between a point and the fitted plane, or IRIT_INFNTY if failed.

Description: Given set of points, PtList, fits an (approximated) plane fit to them.

3.2.519 CagdPlaneFitToPts2 (cbsp_int.c:2014)

approximation

```
CagdRType CagdPlaneFitToPts2(CagdRType * const *Points,
                             int NumPts,
                             CagdPointType PType,
                             IrtPlnType Pln,
                             IrtVecType MVec,
                             IrtPtType Cntr,
                             IrtRType *CN)
```

Points: List of points to fit an approximated plane, in format as in CagdCrvStruct or CagdSrfStruct.

NumPts: Length of the vector of Points.

PType: Type of points in Points.

Pln: The fitted plane's coefficients.

MVec: Direction in the plane with the major change.

Cntr: The centroid of the data.

CN: Condition number of the fitting. The smaller this number is the more planar the input is.

Returns: Average distance between a point and the fitted plane, or IRIT_INFNTY if failed.

Description: Given set of points, PtList, fits an (approximated) plane fit to them.

3.2.520 CagdPlaneFitToPts3 (cbsp_int.c:2058)

approximation

```
CagdRType CagdPlaneFitToPts3(CagdPType * const Points,
                             int NumPts,
                             IrtPlnType Pln,
                             IrtVecType MVec,
                             IrtPtType Cntr,
                             IrtRType *CN)
```

Points: List of points to fit an approximated plane,

NumPts: Length of the vector of Points.

Pln: The fitted plane's coefficients.

MVec: Direction in the plane with the major change.

Cntr: The centroid of the data.

CN: Condition number of the fitting. The smaller this number is the more planar the input is.

Returns: Average distance between a point and the fitted plane, or IRIT_INFNTY if failed.

Description: Given set of points, PtList, fits an (approximated) plane fit to them.

3.2.521 CagdPlaneFree (cagd2gen.c:668)

free

```
void CagdPlaneFree(CagdPlaneStruct *Plane)
```

Plane: To be deallocated.

Returns: void

Description: Deallocates and frees all slots of a plane structure.

3.2.522 CagdPlaneFreeList (cagd2gen.c:691)

free

```
void CagdPlaneFreeList(CagdPlaneStruct *PlaneList)
```

PlaneList: To be deallocated.

Returns: void

Description: Deallocates and frees a plane structure list:

3.2.523 CagdPlaneNew (cagd1gen.c:517)

allocation

```
CagdPlaneStruct *CagdPlaneNew(void)
```

Returns: A Plane structure.

Description: Allocates and resets all slots of a Plane structure.

3.2.524 CagdPointsBBox (cagdbbox.c:627)

bbox

bounding box

```
void CagdPointsBBox(CagdRType * const *Points,
                    int Length,
                    int Dim,
                    CagdRType *BBoxMin,
                    CagdRType *BBoxMax)
```

Points: To compute bounding box for.

Length: Length of vectors of Points array.

Dim: Dimensions of points, typically 3 for R^3.

BBoxMin, BBoxMax: Where bounding information is to be saved.

Returns: void

Description: Computes a bounding box for a set of control points, in any dimension.
See also: CagdPointsBBox2, CagdCtlPtBBox,

3.2.525 CagdPointsBBox2 (cagdbbox.c:556)

bbox

bounding box

```
void CagdPointsBBox2(CagdRType * const *Points,
                     int Min,
                     int Max,
                     int Dim,
                     CagdRType *BBoxMin,
                     CagdRType *BBoxMax)
```

Points: To compute bounding box for.

Min, Max: Range to examine in the points array, from Min to Max, inclusively.

Dim: Dimensions of points, typically 3 for R^3. A negative value means to compute only -Dim dimension.

BBoxMin, BBoxMax: Where bounding information is to be saved.

Returns: void

Description: Computes a bounding box for a set of E1/E2/E3 control points.
See also: CagdPointsBBox, CagdCtlPtBBox,

3.2.526 CagdPointsHasPoles (cagd2gen.c:2367)

```
int CagdPointsHasPoles(CagdRType * const *Points, int Len, int *Sign)
```

Points: Control points to consider.

Len: Number of points in Points.

Sign: If no poles, the sign of the weights. Can be NULL to ignore.

Returns: TRUE if has poles, FALSE otherwise.

Description: Returns TRUE if the given control points have poles - that is have both negative and positive weights. N

See also: CagdAllWeightsNegative, CagdAllWeightsSame,

3.2.527 CagdPolyApproxErrEstimate (poly_err.c:43)

polygonization

error estimate

```
int CagdPolyApproxErrEstimate(CagdPolyErrEstimateType Method, int Samples)
```

Method: 1. Samples one distance at the center of each polygon. 2. Samples Samples samples uniformly distributed in the parametric area of each polygon and selects the maximum. 3. Samples Samples samples uniformly distributed in the parametric area of each polygon and selects the average.

Samples: Number of samples to sample in the parametric domain of each polygon.

Returns: Old sampling method + Old sampling rate << 8

Description: Sets the methods of sampling the error of a polygonal approximation.

3.2.528 CagdPolyApproxErrs (poly_err.c:116)

polygonization

error estimate

```
CagdRType *CagdPolyApproxErrs(const CagdSrfStruct *Srf,
                              const CagdPolygonStruct *Polys)
```

Srf: Approximated surface.

Polys: The given polygonal approximation. Assumes UV slots are computed and updated in Polys.

Returns: Errors between surface and its polygons. A vector of size (number of polygons + 1) holding the maximal error of each polygon. The last element of the vector will be negative.

Description: Returns the errors between the surface and its polygonal approx.

3.2.529 CagdPolyApproxMaxErr (poly_err.c:81)

polygonization

error estimate

```
CagdRType CagdPolyApproxMaxErr(const CagdSrfStruct *Srf,
                               const CagdPolygonStruct *Polys)
```

Srf: Approximated surface.

Polys: The given polygonal approximation. Assumes UV slots are computed and updated in Polys.

Returns: Maximal error between surface and polygonal approximation

Description: Returns the maximal error between the surface and its polygonal approx.

3.2.530 CagdPolygonArrayNew (cagd1gen.c:584)

allocation

```
CagdPolygonStruct *CagdPolygonArrayNew(int Size)
```

Size: Size of Polygon array to allocate.

Returns: An array of Polygon structures of size Size.

Description: Allocates and resets all slots of an array of Polygon structures.

3.2.531 CagdPolygonBBox (cagdbbox.c:364)

```
void CagdPolygonBBox(const CagdPolygonStruct *Poly, CagdBBoxStruct *BBox)
```

Poly: To computes its bbox.

BBox: Where bounding information is to be saved.

Returns: void

Description: Computes a bounding box for a cagd polygon.
See also: CagdPolygonListBBox, CagdCrvBBox, CagdSrfBBox,

3.2.532 CagdPolygonCopy (cagd1gen.c:1097)

copy

```
CagdPolygonStruct *CagdPolygonCopy(const CagdPolygonStruct *Poly)
```

Poly: To be copied.

Returns: A duplicate of Polygon.

Description: Allocates and copies all slots of a Polygon structure.

3.2.533 CagdPolygonCopyList (cagd2gen.c:157)

copy

```
CagdPolygonStruct *CagdPolygonCopyList(const CagdPolygonStruct *PolyList)
```

PolyList: To be copied.

Returns: A duplicated list of polygons.

Description: Allocates and copies a list of polygon structures.

3.2.534 CagdPolygonFree (cagd2gen.c:827)

free

```
void CagdPolygonFree(CagdPolygonStruct *Poly)
```

Poly: To be deallocated.

Returns: void

Description: Deallocates and frees all slots of a polygon structure.

3.2.535 CagdPolygonFreeList (cagd2gen.c:856)

free

```
void CagdPolygonFreeList(CagdPolygonStruct *PolyList)
```

PolyList: To be deallocated.

Returns: void

Description: Deallocates and frees a polygon structure list:

3.2.536 CagdPolygonListBBox (cagdbbox.c:415)

```
void CagdPolygonListBBox(const CagdPolygonStruct *Polys, CagdBBoxStruct *BBox)
```

Polys: To computes its bbox.

BBox: Where bounding information is to be saved.

Returns: void

Description: Computes a bounding box for a list of cagd polygons.
See also: CagdPolygonBBox, CagdCrvListBBox, CagdSrfListBBox,

3.2.537 CagdPolygonNew (cagd1gen.c:617)

allocation

```
CagdPolygonStruct *CagdPolygonNew(int Len)
```

Len: Number of vertices

Returns: A Polygon structure.

Description: Allocates and resets all slots of a Polygon structure.

3.2.538 CagdPolygonSetErrFunc (cagd2gen.c:2628)

```
CagdPlgErrorFuncType CagdPolygonSetErrFunc(CagdPlgErrorFuncType Func)
```

Func: New function to use, NULL to disable.

Returns: Old value of function.

Description: Sets the polygon approximation error function. The error function will return a negative value if this triangle must be purged or otherwise a non negative error measure.

3.2.539 CagdPolygonStripNew (cagd1gen.c:647)

allocation

```
CagdPolygonStruct *CagdPolygonStripNew(int Len)
```

Len: Number of polygons in strip.

Returns: A Polygon structure.

Description: Allocates and resets all slots of a Polygon structure as a strip.

3.2.540 CagdPolylineArrayNew (cagd1gen.c:680)

allocation

```
CagdPolylineStruct *CagdPolylineArrayNew(int Length, int Size)
```

Length: Length of each polyline in the polyline array.

Size: Size of Polyline array to allocate.

Returns: An array of Polyline structures of size Size.

Description: Allocates and resets all slots of an array of Polyline structures.

3.2.541 CagdPolylineCopy (cagd1gen.c:1141)

copy

```
CagdPolylineStruct *CagdPolylineCopy(const CagdPolylineStruct *Poly)
```

Poly: To be copied.

Returns: A duplicate of Polyline.

Description: Allocates and copies all slots of a Polyline structure.

3.2.542 CagdPolylineCopyList (cagd2gen.c:128)

copy

```
CagdPolylineStruct *CagdPolylineCopyList(const CagdPolylineStruct *PolyList)
```

PolyList: To be copied.

Returns: A duplicated list of polylines.

Description: Allocates and copies a list of polyline structures.

3.2.543 CagdPolylineFree (cagd2gen.c:779)

free

```
void CagdPolylineFree(CagdPolylineStruct *Poly)
```

Poly: To be deallocated.

Returns: void

Description: Deallocates and frees all slots of a polyline structure.

3.2.544 CagdPolylineFreeList (cagd2gen.c:803)

free

```
void CagdPolylineFreeList(CagdPolylineStruct *PolyList)
```

PolyList: To be deallocated.

Returns: void

Description: Deallocates and frees a polyline structure list:

3.2.545 CagdPolylineNew (cagd1gen.c:711)

allocation

```
CagdPolylineStruct *CagdPolylineNew(int Length, int PtDim)
```

Length: Length of polyline.

PtDim: Dimension of polyline, typically 3.

Returns: A Polyline structure.

Description: Allocates and resets all slots of a Polyline structure.

3.2.546 CagdPrimBoxSrf (cagdprim.c:614)

```
CagdSrfStruct *CagdPrimBoxSrf(CagdRType MinX,
                              CagdRType MinY,
                              CagdRType MinZ,
                              CagdRType MaxX,
                              CagdRType MaxY,
                              CagdRType MaxZ)
```

MinX, MinY, MinZ: Minimum range of box model.

MaxX, MaxY, MaxZ: Maximum range of box model.

Returns: Constructed box model, as a set of six bilinear srfs.

Description: A surface constructor of a box, parallel to main axes.

3.2.547 CagdPrimCone2Srf (cagdprim.c:861)

```
CagdSrfStruct *CagdPrimCone2Srf(const CagdVType Center,
                                CagdRType MajorRadius,
                                CagdRType MinorRadius,
                                CagdRType Height,
                                CagdBType Rational,
                                CagdPrimCapsType Caps)
```

Center: of constructed cone (center of its base).

MajorRadius: of constructed cone.

MinorRadius: of constructed cone.

Height: of constructed cone.

Rational: If TRUE exact ration sphere is created. If FALSE an approximated integral surface is created.

Caps: Do we want caps (top and/or bottom) for the cone?

Returns: A Bspline surface representing a cone.

Description: A surface constructor of a truncated cone, centered at Center and radii of MajorRadius and MinorRadius. A MinorRadius of zero would construct a regular cone. Otherwise, a truncated cone. Axis of cone is Z axis.

See also: CagdPrimPlaneSrf, CagdPrimSphereSrf, CagdPrimTorusSrf, CagdPrimConeSrf, , CagdPrimCylinderSrf,

3.2.548 CagdPrimConeSrf (cagdprim.c:937)

```
CagdSrfStruct *CagdPrimConeSrf(const CagdVType Center,
                               CagdRType Radius,
                               CagdRType Height,
                               CagdBType Rational,
                               CagdPrimCapsType Caps)
```

Center: of constructed cone (center of its base).

Radius: of constructed cone's base.

Height: of constructed cone.

Rational: If TRUE exact ration sphere is created. If FALSE an approximated integral surface is created.

Caps: Do we want caps (top and/or bottom) for the cone?

Returns: A Bspline surface representing a cone.

Description: A surface constructor of a cone, centered at Center, radii of Radius, and height of Height. Axis of cone is Z axis.

See also: CagdPrimPlaneSrf, CagdPrimSphereSrf, CagdPrimTorusSrf, CagdPrimCone2Srf, , CagdPrimCylinderSrf,

3.2.549 CagdPrimCubeSphereSrf (cagdprim.c:713)

```
CagdSrfStruct *CagdPrimCubeSphereSrf(const CagdVType Center,
                                     CagdRType Radius,
                                     CagdBType Rational)
```

Center: Of constructed sphere.

Radius: Of constructed sphere.

Rational: If TRUE exact ration sphere is created. If FALSE an approximated integral surface is created.

Returns: A list of six B-spline surfaces representing a sphere.

Description: A surface constructor of a sphere, centered at Center and radius Radius. Constructs the rational sphere out of six patches in a cube topology. See also: "Tiling the Sphere with Rational Bezier Patches" by Jim Cobb.

See also: CagdPrimPlaneSrf, CagdPrimTorusSrf, CagdPrimCone2Srf, CagdPrimConeSrf, , CagdPrimCylinderSrf, CagdPrimSphereSrf,

3.2.550 CagdPrimCylinderSrf (cagdprim.c:969)

```
CagdSrfStruct *CagdPrimCylinderSrf(const CagdVType Center,
                                   CagdRType Radius,
                                   CagdRType Height,
                                   CagdBType Rational,
                                   CagdPrimCapsType Caps)
```

Center: of constructed Cylinder (center of its base).

Radius: of constructed Cylinder.

Height: of constructed Cylinder.

Rational: If TRUE exact ration sphere is created. If FALSE an approximated integral surface is created.

Caps: Do we want caps (top and/or bottom) for the cone?

Returns: A Bspline surface representing a cylinder.

Description: A surface constructor of a Cylinder, centered at Center, radii of Radius, and height of Height. Axis of cylinder is Z axis.
See also: CagdPrimPlaneSrf, CagdPrimSphereSrf, CagdPrimTorusSrf, CagdPrimCone2Srf, , CagdPrimConeSrf,

3.2.551 CagdPrimLinCrvFrom4Pts (cagdprim.c:142)

```
CagdCrvStruct *CagdPrimLinCrvFrom4Pts(const CagdPType P1,
                                      const CagdPType P2,
                                      const CagdPType P3,
                                      const CagdPType P4,
                                      CagdBType Closed)
```

P1, P2, P3, P4: The four corner points of the curve, in 3-space.

Closed: TRUE to close the curve. That is, last, 5th, point will be duplicated and equal to first.

Returns: Constructed linear B-spline curve.

Description: Creates a piecewise linear closed B-spline curve from 4 points in general position. Note points need not be co-planar.
See also: CagdPrimPlaneSrf, CagdPrimRectangleCrv,

3.2.552 CagdPrimPlaneFromE3Crv (cagdprim.c:339)

```
CagdSrfStruct *CagdPrimPlaneFromE3Crv(const CagdCrvStruct *Crv,
                                      const IrtPlnType Plane)
```

Crv: Planar curve in general E3 position.

Plane: Optional plane equation of the planar surface. If not given (can be NULL) computed from Crv directly.

Returns: Constructed plane, as a bilinear surface.

Description: Build a planar srf, large enough to accommodate the given curve in it. The curve is assumed planar and in general E3 position.
See also: CagdPrimSphereSrf, CagdPrimTorusSrf, CagdPrimCone2Srf, CagdPrimConeSrf, , CagdPrimCylinderSrf, CagdPrimRectangleCrv, CagdPrimPlaneSrf, , CagdPrimPlaneSrf2, CagdPrimPlanePlaneSpanBBox,

3.2.553 CagdPrimPlanePlaneSpanBBox (cagdprim.c:553)

```
CagdSrfStruct *CagdPrimPlanePlaneSpanBBox(const CagdBBoxStruct *BBox,
                                          const IrtPlnType Pln)
```

BBox: To make sure Plane spans beyond it in all dimensions.

Pln: The plane to build specifications.

Returns: The constructed plane.

Description: Builds a planar surface, large enough to span all of BBox as the plane specified by Pln.
See also: CagdPrimSphereSrf, CagdPrimTorusSrf, CagdPrimCone2Srf, CagdPrimConeSrf, , CagdPrimCylinderSrf, CagdPrimRectangleCrv, CagdPrimPlaneSrf, , CagdPrimPlaneSrf2, CagdPrimPlaneFromE3Crv, CagdPrimPlaneYZSrf,

3.2.554 CagdPrimPlaneSrf (cagdprim.c:187)

```
CagdSrfStruct *CagdPrimPlaneSrf(CagdRType MinX,
                                CagdRType MinY,
                                CagdRType MaxX,
                                CagdRType MaxY,
                                CagdRType ZLevel)
```

MinX, MinY: Minimum X coordinates of plane.

MaxX, MaxY: Maximum Y coordinates of plane.

ZLevel: Z level of plane, parallel to the XY plane.

Returns: Constructed plane, as a bilinear surface.

Description: A surface constructor of a plane, parallel to XY plane at level Zlevel.
See also: CagdPrimSphereSrf, CagdPrimTorusSrf, CagdPrimCone2Srf, CagdPrimConeSrf, , CagdPrimCylinderSrf, CagdPrimRectangleCrv, CagdPrimPlaneSrf1, , CagdPrimPlaneSrf2, CagdPrimPlaneFromE3Crv, CagdPrimPlanePlaneSpanBBox,

3.2.555 CagdPrimPlaneSrf1 (cagdprim.c:244)

```
CagdSrfStruct *CagdPrimPlaneSrf1(CagdRType P00X,
                                 CagdRType P00Y,
                                 CagdRType P01X,
                                 CagdRType P01Y,
                                 CagdRType P10X,
                                 CagdRType P10Y,
                                 CagdRType P11X,
                                 CagdRType P11Y,
                                 CagdRType ZLevel)
```

P00X, P00Y: Location of first corner planar surface.

P01X, P01Y: Location of second dorner planar surface.

P10X, P10Y: Location of third corner planar surface.

P11X, P11Y: Location of fourth corner planar surface.

ZLevel: Z level of plane, parallel to the XY plane.

Returns: Constructed planar surface, as a bilinear.

Description: A surface constructor of a planar bilinear, parallel to XY plane at level Zlevel.
See also: CagdPrimSphereSrf, CagdPrimTorusSrf, CagdPrimCone2Srf, CagdPrimConeSrf, , CagdPrimCylinderSrf, CagdPrimRectangleCrv, CagdPrimPlaneSrf, , CagdPrimPlaneSrf2, CagdPrimPlaneFromE3Crv, CagdPrimPlanePlaneSpanBBox,

3.2.556 CagdPrimPlaneSrf2 (cagdprim.c:292)

```
CagdSrfStruct *CagdPrimPlaneSrf2(CagdPType Cntr,
                                 CagdVType Vec1,
                                 CagdVType Vec2)
```

Cntr: The center of the constructed plane.

Vec1, Vec2: Two vectors spanning the plane (also sets the plane's size).

Returns: Constructed plane, as a bilinear surface.

Description: A surface constructor of a plane, in general position: centered around Center and spanned by Vec1 and Vec2 that are two independent vectors.
See also: CagdPrimSphereSrf, CagdPrimTorusSrf, CagdPrimCone2Srf, CagdPrimConeSrf, , CagdPrimCylinderSrf, CagdPrimRectangleCrv, CagdPrimPlaneSrf, , CagdPrimPlaneSrf1, CagdPrimPlaneFromE3Crv,

3.2.557 CagdPrimPlaneSrfOrderLen (cagdprim.c:404)

```
CagdSrfStruct *CagdPrimPlaneSrfOrderLen(CagdRType MinX,
                                        CagdRType MinY,
                                        CagdRType MaxX,
                                        CagdRType MaxY,
                                        CagdRType ZLevel,
                                        int Order,
                                        int Len)
```

MinX, MinY: Minimum XY coordinates of plane.

MaxX, MaxY: Maximum XY coordinates of plane.

ZLevel: Z level of plane, parallel to the XY plane.

Order: Order of plane surface that is requested.

Len: Number of control points (via refinement).

Returns: Constructed surface.

Description: A surface constructor of a plane, parallel to XY plane at level Zlevel.
See also: CagdPrimSphereSrf, CagdPrimTorusSrf, CagdPrimCone2Srf, CagdPrimConeSrf, , CagdPrimCylinderSrf, CagdPrimPlaneSrf, CagdPrimPlaneSrf2, , CagdPrimPlaneSrf2, CagdPrimPlaneFromE3Crv, CagdPrimPlanePlaneSpanBBox,

3.2.558 CagdPrimPlaneXZSrf (cagdprim.c:455)

```
CagdSrfStruct *CagdPrimPlaneXZSrf(CagdRType MinX,
                                  CagdRType MinZ,
                                  CagdRType MaxX,
                                  CagdRType MaxZ,
                                  CagdRType YLevel)
```

MinX, MinZ: Minimum X coordinates of plane.

MaxX, MaxZ: Maximum Z coordinates of plane.

YLevel: Y level of plane, parallel to the XZ plane.

Returns: Constructed plane, as a bilinear surface.

Description: A surface constructor of an XZ plane, parallel to XZ plane at level Ylevel.
See also: CagdPrimSphereSrf, CagdPrimTorusSrf, CagdPrimCone2Srf, CagdPrimConeSrf, , CagdPrimCylinderSrf, CagdPrimRectangleCrv, CagdPrimPlaneSrf, , CagdPrimPlaneSrf2, CagdPrimPlaneFromE3Crv, CagdPrimPlanePlaneSpanBBox,

3.2.559 CagdPrimPlaneYZSrf (cagdprim.c:506)

```
CagdSrfStruct *CagdPrimPlaneYZSrf(CagdRType MinY,
                                  CagdRType MinZ,
                                  CagdRType MaxY,
                                  CagdRType MaxZ,
                                  CagdRType XLevel)
```

MinY, MinZ: Minimum Y coordinates of plane.

MaxY, MaxZ: Maximum Z coordinates of plane.

XLevel: X level of plane, parallel to the YZ plane.

Returns: Constructed plane, as a bilinear surface.

Description: A surface constructor of an YZ plane, parallel to YZ plane at level Ylevel.
See also: CagdPrimSphereSrf, CagdPrimTorusSrf, CagdPrimCone2Srf, CagdPrimConeSrf, , CagdPrimCylinderSrf, CagdPrimRectangleCrv, CagdPrimPlaneSrf, , CagdPrimPlaneSrf2, CagdPrimPlaneFromE3Crv, CagdPrimPlanePlaneSpanBBox,

3.2.560 CagdPrimRectangleCrv (cagdprim.c:90)

```
CagdCrvStruct *CagdPrimRectangleCrv(CagdRType MinX,
                                    CagdRType MinY,
                                    CagdRType MaxX,
                                    CagdRType MaxY,
                                    CagdRType ZLevel)
```

MinX, MinY: Minimum XY coordinates of rectangle.

MaxX, MaxY: Maximum XY coordinates of rectangle.

ZLevel: Z level of rectangle, parallel to the XY plane.

Returns: Constructed curve.

Description: A curve constructor of a rectangle, parallel to XY plane at level Zlevel.
See also: CagdPrimPlaneSrf, CagdPrimLinCrvFrom4Pts,

3.2.561 CagdPrimSphereSrf (cagdprim.c:655)

```
CagdSrfStruct *CagdPrimSphereSrf(const CagdVType Center,
                                 CagdRType Radius,
                                 CagdBType Rational)
```

Center: Of constructed sphere.

Radius: Of constructed sphere.

Rational: If TRUE exact ration sphere is created. If FALSE an approximated integral surface is created.

Returns: A Bspline surface representing a sphere.

Description: A surface constructor of a sphere, centered at Center and radius Radius.
See also: CagdPrimPlaneSrf, CagdPrimTorusSrf, CagdPrimCone2Srf, CagdPrimConeSrf, , CagdPrimCylinderSrf, CagdPrimCubeSphereSrf,

3.2.562 CagdPrimTorusSrf (cagdprim.c:798)

```
CagdSrfStruct *CagdPrimTorusSrf(const CagdVType Center,
                                CagdRType MajorRadius,
                                CagdRType MinorRadius,
                                CagdBType Rational)
```

Center: of constructed torus.

MajorRadius: of constructed torus.

MinorRadius: of constructed torus.

Rational: If TRUE exact ration sphere is created. If FALSE an approximated integral surface is created.

Returns: A Bspline surface representing a torus.

Description: A surface constructor of a torus, centered at Center and radii of MajorRadius and MinorRadius.
See also: CagdPrimPlaneSrf, CagdPrimSphereSrf, CagdPrimCone2Srf, CagdPrimConeSrf, , CagdPrimCylinderSrf,

3.2.563 CagdPtArrayFree (cagd2gen.c:503)

free

```
void CagdPtArrayFree(CagdPtStruct *PtArray, int Size)
```

PtArray: To be deallocated.

Size: Of the deallocated array.

Returns: void

Description: Deallocates and frees an array of Pt structure.

3.2.564 CagdPtArrayNew (cagd1gen.c:308)

allocation

```
CagdPtStruct *CagdPtArrayNew(int Size)
```

Size: Size of Pt array to allocate.

Returns: An array of Pt structures of size Size.

Description: Allocates and resets all slots of an array of Pt structures.

3.2.565 CagdPtCopy (cagd1gen.c:949)

copy

```
CagdPtStruct *CagdPtCopy(const CagdPtStruct *Pt)
```

Pt: To be copied.

Returns: A duplicate of Pt.

Description: Allocates and copies all slots of a Pt structure.

3.2.566 CagdPtCopyList (cagd1gen.c:1295)

copy

```
CagdPtStruct *CagdPtCopyList(const CagdPtStruct *PtList)
```

PtList: To be copied.

Returns: A duplicated list of points.

Description: Allocates and copies a list of point structures.

3.2.567 CagdPtFree (cagd2gen.c:408)

free

```
void CagdPtFree(CagdPtStruct *Pt)
```

Pt: To be deallocated.

Returns: void

Description: Deallocates and frees all slots of a point structure.

3.2.568 CagdPtFreeList (cagd2gen.c:431)

free

```
void CagdPtFreeList(CagdPtStruct *PtList)
```

PtList: To be deallocated.

Returns: void

Description: Deallocates and frees a point structure list:

3.2.569 CagdPtNew (cagd1gen.c:335)

allocation

```
CagdPtStruct *CagdPtNew(void)
```

Returns: A Pt structure.

Description: Allocates and resets all slots of a Pt structure.

3.2.570 CagdPtPolyline2EkPolyline (bzr2poly.c:1232)

piecewise linear approximation

polyline

```
CagdPolylineStruct *CagdPtPolyline2EkPolyline(
                                CagdRType * const Polyline[CAGD_MAX_PT_SIZE],
                                int n,
                                int MaxCoord,
                                CagdBType IsRational)
```

Polyline: A vector of evaluations with MaxCoord each evaluation.

n: Number of evaluation (length of Polyline).

MaxCoord: Number of coordinates in the polyline Polyline.

IsRational: TRUE, if original curve was rational, FALSE otherwise.

Returns: A list of E3 polylines representing the piecewise linear approximation. Typically, only one polyline, unless the (rational) curve has poles.

Description: Routine to convert a possibly rational polyline to a CagdPolylineStruct Ek, k >= 3, format.
See also: BzrCrv2Polyline, BspSrf2Polylines, IritCurve2Polylines, , SymbCrv2Polyline,

3.2.571 CagdPtsSortAxis (cbsp_int.c:1818)

```
CagdPtStruct *CagdPtsSortAxis(CagdPtStruct *PtList, int Axis)
```

PtList: List of points to sort.

Axis: Axis to sort along: 1,2,3 for X,Y,Z.

Returns: Sorted list of points, in place.

Description: Sorts given list of points based on their increasing order in axis Axis. Sorting is done in place.

3.2.572 CagdQuadCrvToQuadsLineSweep (crv2quad.c:288)

untrimming

```
CagdCrvQuadTileStruct *CagdQuadCrvToQuadsLineSweep(const CagdCrvStruct *Crv,
                                                   CagdBType OutputSrfs)
```

Crv: The curve to cover in tiles.

OutputSrfs: A flag for choosing whether to output ruled surfaces (if TRUE) or pairs of curves (forming the limits of the ruled surfaces).

Returns: A list of either ruled surfaces, or pairs of curves (forming the bounadry of the ruled surfaces.

Description: Computes the covering of a closed planar curve in quadrilateral tiles using the line sweep algorithm. This function includes preprocessing, and error handling by attempting multiple rotations.
See also: CagdCrvPartitionIntoBoolSums,

3.2.573 CagdQuadCurveListWeightedQuadrangulation (crv_quad.c:1951)

```
CagdSrfStruct *CagdQuadCurveListWeightedQuadrangulation(
                                                const CagdCrvStruct *CrvList,
                                                CagdQuadSrfWeightFuncType
                                                                WeightFunc,
                                                int ApproxOrder)
```

CrvList: curve list to quadrangulate.

WeightFunc: Quad surface weight function to optimize for the quadrangulation algorithm.

ApproxOrder: If not zero, the results surfaces will be approximated to surfaces with of this given order. The approximation is done before the composition.

Returns: List of tensor product surfaces tiling Crv.

Description: Tiles list of curves with tensor product surfaces using minimal weighted quadrangulation algorithm.

3.2.574 CagdQuadCurveWeightedQuadrangulation (crv_quad.c:1726)

```
CagdSrfStruct *CagdQuadCurveWeightedQuadrangulation(const CagdCrvStruct *Crv,
                                                    CagdQuadSrfWeightFuncType
                                                                  WeightFunc,
                                                    int ApproxOrder)
```

Crv: A curve to quadrangulate.

WeightFunc: Quad surface weight function to optimize for the quadrangulation algorithm.

ApproxOrder: If not zero, the results surfaces will be approximated to surfaces with of this given order. The approximation is done before the composition.

Returns: List of tensor product surfaces tiling Crv.

Description: Tiles a curve with tensor product surfaces using minimal weighted quadrangulation algorithm.

3.2.575 CagdQuadGetQuadSrfCombinedWeightFactors (crv_quad.c:540)

```
void CagdQuadGetQuadSrfCombinedWeightFactors(IrtRType *JacobianFactor,
                                             IrtRType *ConformFactor,
                                             IrtRType *RegularityFactor)
```

JacobianFactor: Factor of QuadSrfJacobianWeight weight function.

ConformFactor: Factor of QuadSrfConformWeight weight function.

RegularityFactor: Factor of QuadSrfRegularPolyWeight weight function.

Returns: void

Description: Gets the factor multiplied by each weight function in the combined weight function QuadSrfCombinedWeight.
See also: CagdQuadSrfCombinedWeight, CagdQuadSetQuadSrfCombinedWeightFactors.,

3.2.576 CagdQuadSetQuadSrfCombinedWeightFactors (crv_quad.c:512)

```
void CagdQuadSetQuadSrfCombinedWeightFactors(IrtRType JacobianFactor,
                                             IrtRType ConformFactor,
                                             IrtRType RegularityFactor)
```

JacobianFactor: Factor of QuadSrfJacobianWeight weight function.

ConformFactor: Factor of QuadSrfConformWeight weight function.

RegularityFactor: Factor of QuadSrfRegularPolyWeight weight function.

Returns: void

Description: Sets the factor multiplied by each weight function in the combined weight function QuadSrfCombinedWeight.
See also: CagdQuadSrfCombinedWeight, CagdQuadGetQuadSrfCombinedWeightFactors.,

3.2.577 CagdQuadSrfCombinedWeight (crv_quad.c:464)

```
IrtRType CagdQuadSrfCombinedWeight(const CagdSrfStruct *QuadSrf,
                                   const CagdCrvStruct *BoundaryCrv,
                                   const CagdPolylineStruct *SampledPolygon,
                                   const int *VIndices,
                                   int numV)
```

QuadSrf: The given surface, built from four curve pieces of BoundaryCrv.

BoundaryCrv: The curve being quadratized.

SampledPolygon: The representative polygon of BoundaryCrv.

VIndices: An array of vertices indices from SampledPolygon that forms a quad.

numV: Number of vertices of the quad (length of VIndices).

Returns: The weight of the given quad surface.

Description: Computes weight function based on a combination of Jacobian, conformality and the regularity of a given surface. The factor multiplied by each weight function can be set using SetQuadSrfCombinedWeightFactors function.

See also: CagdQuadSrfJacobianWeight, CagdQuadSrfConformWeight, , CagdQuadSrfRegularPolyWeight, GMQuadAreaPerimterRatioWeightFunc, , CagdQuadSetQuadSrfCombinedWeightFactors.,

3.2.578 CagdQuadSrfConformWeight (crv_quad.c:361)

```
IrtRType CagdQuadSrfConformWeight(const CagdSrfStruct *QuadSrf,
                                  const CagdCrvStruct *BoundaryCrv,
                                  const CagdPolylineStruct *SampledPolygon,
                                  const int *VIndices,
                                  int numV)
```

QuadSrf: The given surface, built from four curve pieces of BoundaryCrv.

BoundaryCrv: The curve being quadratized.

SampledPolygon: The representative polygon of BoundaryCrv.

VIndices: An array of vertices indices from SampledPolygon that forms a quad.

numV: Number of vertices of the quad (length of VIndices).

Returns: The weight of the given quad surface.

Description: Computes weight function based on orthogonality of iso-curves of a given surface. More orthogonal iso curve are assigned lower weight. A large weight is assigned for triangular surfaces.

See also: CagdQuadSrfJacobianWeight, CagdQuadSrfRegularPolyWeight, , CagdQuadSrfCombinedWeight.,

3.2.579 CagdQuadSrfJacobianWeight (crv_quad.c:320)

```
IrtRType CagdQuadSrfJacobianWeight(const CagdSrfStruct *QuadSrf,
                                   const CagdCrvStruct *BoundaryCrv,
                                   const CagdPolylineStruct *SampledPolygon,
                                   const int *VIndices,
                                   int numV)
```

QuadSrf: The given surface, built from four curve pieces of BoundaryCrv.

BoundaryCrv: The curve being quadratized.

SampledPolygon: The representative polygon of BoundaryCrv.

VIndices: An array of vertices indices from SampledPolygon that forms a quad.

numV: Number of vertices of the quad (length of VIndices).

Returns: The weight of the given quad surface.

Description: Computes weight function based on Jacobian of a given planar surface. If the Jacobian changes sign, an infinite weight is returned, otherwise JMax/JMin is returned, where JMin and JMax are the minimal and maximal values of the Jacobian. A large weight is assigned for triangular surfaces.

See also: CagdQuadSrfConformWeight, CagdQuadSrfRegularPolyWeight, , CagdQuadSrfCombinedWeight.,

3.2.580 CagdQuadSrfRegularPolyWeight (crv_quad.c:428)

```
IrtRType CagdQuadSrfRegularPolyWeight(const CagdSrfStruct *QuadSrf,
                                      const CagdCrvStruct *BoundaryCrv,
                                      const CagdPolylineStruct *SampledPolygon,
                                      const int *VIndices,
                                      int numV)
```

QuadSrf: The given surface, built from four curve pieces of BoundaryCrv.

BoundaryCrv: The curve being quadratized.

SampledPolygon: The representative polygon of BoundaryCrv.

VIndices: An array of vertices indices from SampledPolygon that forms a quad.

numV: Number of vertices of the quad (length of VIndices).

Returns: The weight of the given quad surface.

Description: Computes weight function based on regularity of a given surface. Regularity is approximated by the representative polygonal quad as: (0.75 * QuadArea + 0.05 * QuadPerimeter + 0.10 * EdgesMaxMinRatio). A large weight is assigned for triangular surfaces.
See also: CagdQuadSrfJacobianWeight, CagdQuadSrfConformWeight, , CagdQuadSrfCombinedWeight.,

3.2.581 CagdQuadraticCrvFit (hermite.c:151)

Hermite

```
CagdCrvStruct *CagdQuadraticCrvFit(const CagdCrvStruct *Crv)
```

Crv: To approximate as a quadratic.

Returns: A quadratic Bezier curve, fitting the end position tangent directions if planar, and not singular.

Description: Construct a quadratic Bezier curve that fits the input curve end points and aims for Same tangent directions. If Crv is planar and not singular i. e. parallel tangents) G1 continuity is ensured.
See also: CagdCubicHermiteCrv, CagdCubicCrvFit, SymbApproxCrvAsBzrQuadratics,

3.2.582 CagdQuadraticSrfFit (hermite.c:578)

Hermite

```
CagdSrfStruct *CagdQuadraticSrfFit(const CagdSrfStruct *Srf)
```

Srf: To approximate as a quadratic.

Returns: A biquadratic Bezier surface, fitting end positions and tangent directions if planar, and not singular.

Description: Construct a biquadratic Bezier surface that fits the input surface corner points and aims for same corner tangent directions. If Srf planar and not singular (i. e. parallel tangents) G1 continuity is ensured.
See also: CagdCubicHermiteSrf, CagdCubicSrfFit, CagdQuadraticCrvFit,

3.2.583 CagdQuadricMatTransform (cagd_cnc.c:1198)

implicit

```
CagdBType CagdQuadricMatTransform(CagdRType *A,
                                  CagdRType *B,
                                  CagdRType *C,
                                  CagdRType *D,
                                  CagdRType *E,
                                  CagdRType *F,
                                  CagdRType *G,
                                  CagdRType *H,
                                  CagdRType *I,
                                  CagdRType *J,
                                  CagdMType Mat)
```

A, B, C, D, E, F, G, H, I, J: The ten coefficients of the quadric. Updated in place.

Mat: Transformation matrix.

Returns: TRUE if successful, FALSE otherwise.

Description: Transform given quadric form A x^2 + B y^2 + C z^2 + D xy + E xz + F yz + G x + H y + I z + J = 0, using Mat. Algorithm:

```
1. Convert the implicit quadric to a matrix form as:
               [ A  D/2 E/2 G/2] [x]
```

```
[x, y, z, 1] [D/2  B  F/2 H/2] [y] = P^T M P = 0

             [E/2 F/2  C  I/2] [z]

             [G/2 H/2 I/2  J ] [1]
```

```
2. Compute N = Mat^{-1} the inverse of the desired transformation.
```

```
3. Compute K = N M N^T and decompose K back to A-J coefficients.
```

See also: CagdConicMatTransform, CagdSrfTransfrom, CagdCrvTransform,

3.2.584 CagdQuinticHermiteSrf (hermite.c:310)

Hermite

```
CagdSrfStruct *CagdQuinticHermiteSrf(const CagdCrvStruct *CPos1Crv,
                                     const CagdCrvStruct *CPos2Crv,
                                     const CagdCrvStruct *CDir1Crv,
                                     const CagdCrvStruct *CDir2Crv,
                                     const CagdCrvStruct *C2Dir1Crv,
                                     const CagdCrvStruct *C2Dir2Crv)
```

CPos1Crv, CPos2Crv: Starting and end curves of surface.

CDir1Crv, CDir2Crv: Starting and end tangent fields surface.

C2Dir1Crv, C2Dir2Crv: Starting and end 2nd derivative fields surface.

Returns: A cubic by something Bezier surface, satisfying the four constrants. The other something degree is the largest of the four given curves.

Description: Construct a cubic surface using the Hermite constraints - two positions and two tangents. Other direction's degree depends on input.

3.2.585 CagdRayTraceBzrSrf (bez_clip.c:111)

```
CagdBType CagdRayTraceBzrSrf(CagdPType StPt,
                             CagdVType Dir,
                             const CagdSrfStruct *BzrSrf,
                             CagdUVStruct **IntrPrm,
                             CagdPtStruct **IntrPt)
```

StPt: Start point of the ray.

Dir: Direction of the ray.

BzrSrf: A rational Bezier surface to be ray-traced.

IntrPrm: Ray/Bezier surface intersection parameter (of the surface).

IntrPt: Ray/Bezier surface intersection points.

Returns: FALSE if no intersection, TRUE for intersection(s).

Description: Computing the intersection of a rational Bezier surface with a ray. Based on algorithm described in Computer Graphics, Volume 24, Number 4, August 1990: "Ray Tracing Trimmed Rational Surface Patches",written by Tomoyuki Nishita, Thomas W.Sederberg and Masanori Kakimoto.

3.2.586 CagdRealVecSame (cagdcoer.c:372)

```
CagdBType CagdRealVecSame(CagdRType const *Vec1,
                          CagdRType const *Vec2,
                          int Len,
                          CagdRType Eps)
```

Vec1, Vec2: Two vectors to compare.

Len: Length of the two vectors.

Eps: Tolerance of equality.

Returns: TRUE if vectors are the same, FALSE otherwise.

Description: Compare the two real vectors for similarity.
See also: BspKnotVectorsSame, CagdCtlMeshsSame, CagdCrvsSame, CagdSrfsSame,

3.2.587 CagdReorderCurvesInLoop (cagdbsum.c:893)

```
CagdCrvStruct *CagdReorderCurvesInLoop(CagdCrvStruct *UVCrvs)
```

UVCrvs: Curves forming one loop to reorder so end point of one curve is the beginning of the next curve.

Returns: Curves properly reordered in loop.

Description: Properly reorder given curves of one closed loops, in place. That is, input is assumed to define a complete loop where one curve ends where another begins. Compare end points in R^3 (XYZ). Input curves can be in arbitrary order and even partially reversed. Orientation of loop will be following first curve in the list.
See also: CagdSrfFromNBndryCrvs,

3.2.588 CagdRuledSrf (cagdruld.c:33)

ruled surface

surface constructors

```
CagdSrfStruct *CagdRuledSrf(const CagdCrvStruct *CCrv1,
                            const CagdCrvStruct *CCrv2,
                            int OtherOrder,
                            int OtherLen)
```

CCrv1, CCrv2: The two curves to form a ruled surface in between.

OtherOrder: Usually two, but one can specify higher orders in the ruled direction. OtherOrder must never be larger than OrderLen.

OtherLen: Usually two control points in the ruled direction which necesitates a linear interpolation.

Returns: The ruled surface.

Description: Constructs a ruled surface between the two provided curves. OtherOrder and OtherLen (equal for Bezier) specifies the desired order and refineness level (if Bspline) of the other ruled direction.

3.2.589 CagdSDError (cbsp_fit.c:482)

```
static CagdRType CagdSDError(const CagdPType Point,
                             const CagdPType FootPoint,
                             const CagdRType Distance,
                             const CagdPType Tangent,
                             const CagdPType Normal,
                             const CagdRType Curvature)
```

Point: (Xk) Points to calculate the error for.

FootPoint: (P(tk)) Footpoint (the closest point to Xk on the curve)

Distance: (dk) Distance between Point and FootPoint

Tangent: (Tk) Curve tangent at Footpoint.

Normal: (Nk) Curve normal at Footpoint.

Curvature: (pk) Curve curvature at Footpoint.

Returns: SD error.

Description: Computes SD (Squared Distance) error for a single point, which is:

```
           /   dk              T     2                T       2
           |  -----[(P(tk)-Xk) * Tk]  +  [(P(tk)-Xk)  *  Nk]  ,if dk<0
e      =  <   dk-pk
 SD,k      |                 T     2
           |  [(P(tk) - Xk] * Ni]                          ,if 0=<dk<pk
           \
```

3.2.590 CagdScale (cagd2gen.c:1672)

scaling

transformations

```
void CagdScale(CagdRType **Points,
               int Len,
               int MaxCoord,
               const CagdRType *Scale)
```

Points: To be affinely transformed. Array of vectors.

Len: Of vectors of Points.

MaxCoord: Maximum number of coordinates to be found in Points.

Scale: Scaling amount.

Returns: void

Description: Applies a scale transform, in place, to given set of points Points which as array of vectors, each vector of length Len. Array Points optionally contains (if !IsNotRational) in Points[0] the weights coefficients and in Points[i] the coefficients of axis i, up to and include MaxCoord (X = 1, Y = 2, etc.). Points are scaled as prescribed by Scale.

See also: CagdSrfScale, CagdCrvScale, CagdTransform,

3.2.591 CagdScaleCenter (cagd2gen.c:1709)

scaling

transformations

```
void CagdScaleCenter(CagdRType **Points,
                     int Len,
                     int MaxCoord,
                     const CagdRType *Scale)
```

Points: To be scaled. Array of vectors.

Len: Of vectors of Points.

MaxCoord: Maximum number of coordinates to be found in Points.

Scale: Scaling amount.

Returns: void

Description: Applies a scale transform, in place, to given set of points Points which as array of vectors, each vector of length Len. The scale is applied around the center of the geometry at hand. Array Points optionally contains (if !IsNotRational) in Points[0] the weights coefficients and in Points[i] the coefficients of axis i, up to and include MaxCoord (X = 1, Y = 2, etc.). Points are scaled as prescribed by Scale.

See also: CagdSrfScale, CagdCrvScale, CagdTransform, CagdRotateCenter,

3.2.592 CagdScaleWeights (cagd2gen.c:2485)

```
CagdBType CagdScaleWeights(CagdRType * const *Points,
                           CagdPointType PType,
                           int Len,
                           CagdRType WScale)
```

Points: Control points to scale, in place.

PType: Input point type, as given in Points.

Len: Number of points in Points.

WScale: The scaling factor of the knots.

Returns: TRUE if original has negative weights, FALSE otherwise.

Description: Scale all weights, in place, (and control points as needed) by WScale. This operation has no affect on the shape/geometry aside from normalization.
See also: CagdPointsHasPoles, CagdAllWeightsSame,

3.2.593 CagdSetLinear2Poly (cagd2gen.c:2750)

polygonization

polygonal approximation

```
void CagdSetLinear2Poly(CagdLin2PolyType Lin2Poly)
```

Lin2Poly: Specification.

Returns: void

Description: Sets the way (co)linear surfaces are converted into polygons.

3.2.594 CagdSparseMatFree (sbsp_int.c:926)

Sparse matrix

allocation

```
void CagdSparseMatFree(CagdSparseMatStruct *Mat)
```

Mat: Pointer to the sparse matrix.

Returns: void

Description: Free the memory of the sparse matrix (including all the memory allocated for the cells and for the indicator array.
See also: CagdSparseMatNew,

3.2.595 CagdSparseMatMultNonSparseResult (sbsp_int.c:1130)

Sparse matrix

Multiplication

```
IrtRType *CagdSparseMatMultNonSparseResult(CagdSparseMatStruct *Mat1,
                                           CagdSparseMatStruct *Mat2)
```

Mat1: Pointer to first sparse matrix.

Mat2: Pointer to second sparse matrix.

Returns: An array containing the member values of the non sparse matrix. The array size will be (Mat1->RowNum * Mat2->ColNum) The matrix member should be accessed as Result[Row*Mat2->ColNum +Col]

Description: Multiply two sparse matrices (the result is non-sparse matrix), Result = Mat1 * Mat2. NULL will be returned if the input matrix sizes' are incompatible.

3.2.596 CagdSparseMatNew (sbsp_int.c:874)

Sparse matrix

allocation

```
CagdSparseMatStruct *CagdSparseMatNew(int RowNum, int ColNum, int AddIndicator)
```

RowNum: Number of rows in the matrix.

ColNum: Number of columns inthe matrix.

AddIndicator: A Boolean indicating whether to allocate memory and initialize the indicator of cell existance.

Returns: Pointer to allocated sparse matrix.

Description: Allocates memory and initialize a sparse matrix of size RowNum, ColNum.
See also: CagdSparseMatFree,

3.2.597 CagdSparseMatNewCell (sbsp_int.c:1021)

Sparse matrix

allocation

```
void CagdSparseMatNewCell(CagdSparseMatStruct *Mat,
                          int CellRow,
                          int CellCol,
                          IrtRType CellValue)
```

Mat: Pointer to the sparse matrix.

CellRow: The new cell row index.

CellCol: The new cell col index.

CellValue: The Value stored in the cell.

Returns: void

Description: Add a cell to the sparse matrix in the appropriate row and column.
See also: ,

3.2.598 CagdSparseMatTranspose (sbsp_int.c:1189)

Sparse matrix

Transpose

```
CagdSparseMatStruct *CagdSparseMatTranspose(CagdSparseMatStruct *Mat,
                                            CagdBType AddIndicator)
```

Mat: Pointer to the sparse matrix we want to transpose.

AddIndicator: If TRUE, the returned matrix will contain a bit indicator.

Returns: Newly allocated sparse matrix which is the transpose of the input matrix.

Description: Returns the tranpose of the matrix in the input. The newly allocated matrix is returned in sparse format (with or without an indicator as pointed in the AddIndicator parameter).

3.2.599 CagdSrf2CtrlMesh (cagdmesh.c:60)

control mesh

```
CagdPolylineStruct *CagdSrf2CtrlMesh(const CagdSrfStruct *Srf)
```

Srf: To extract a control mesh from.

Returns: The control mesh of Srf.

Description: Extracts the control mesh of a surface as a list of polylines in E3.
See also: CagdSrf2KnotLines, CagdSrf2KnotCurves,

3.2.600 CagdSrf2Curves (cagd_aux.c:656)

curves

isoparametric curves

```
CagdCrvStruct *CagdSrf2Curves(const CagdSrfStruct *Srf, int NumOfIsocurves[2])
```

Srf: To extract isoparametric curves from.

NumOfIsocurves: In each (U or V) direction.

Returns: List of extracted isoparametric curves. These curves inherit the order and continuity of the original Srf. NULL is returned in case of an error.

Description: Routine to extract from a surface NumOfIsoline isocurve list in each param. direction. Iso parametric curves are sampled equally spaced in parametric space. NULL is returned in case of an error, otherwise list of CagdCrvStruct.
See also: BspSrf2PCurves, BzrSrf2Curves, SymbSrf2Curves,

3.2.601 CagdSrf2KnotCurves (cagdmesh.c:200)

control mesh
knots
knot lines
knot curves

```
int CagdSrf2KnotCurves(const CagdSrfStruct *Srf,
                       CagdCrvStruct **UKnotLines,
                       CagdCrvStruct **VKnotLines)
```

Srf: To extract a knot curves from.

UKnotLines: The extracted U knot lines.

VKnotLines: The extracted V knot lines.

Returns: TRUE if successful, FALSE if error.

Description: Extracts a list of knot curves along all knots in U and V of Srf.
See also: CagdSrf2CtrlMesh, CagdSrf2KnotLines, CagdSrf2KnotPolylines, , TrimSrf2KnotCurves,

3.2.602 CagdSrf2KnotLines (cagdmesh.c:119)

control mesh
knots
knot lines

```
CagdPolylineStruct *CagdSrf2KnotLines(const CagdSrfStruct *Srf)
```

Srf: To extract the knot liens in E3 from.

Returns: The control mesh of Srf.

Description: Extracts a polyline grid along all knot in U and V as a list of polylines in E3.
See also: CagdSrf2CtrlMesh, CagdSrf2KnotCurves,

3.2.603 CagdSrf2KnotPolylines (cagd_aux.c:594)

polylines
isoparametric curves

```
CagdPolylineStruct *CagdSrf2KnotPolylines(const CagdSrfStruct *Srf,
                                          int SamplesPerCurve,
                                          BspKnotAlphaCoeffStruct *A)
```

Srf: Srf to extract isoparametric curves from.

SamplesPerCurve: Fineness control on piecewise linear curve approximation.

A: Alpha matrix (Oslo algorithm) if precomputed.

Returns: List of polygons representing a piecewise linear approximation of the extracted isoparametric knot curves or NULL if none or in case of an error.

Description: Routine to extract from a single B-spline surface all internal knot line curves, in each parametric direction with SamplesPerCurve in each curve. Polyline are always E3 of CagdPolylineStruct type. NULL is returned in case of an error, otherwise list of CagdPolylineStruct.
See also: BspCrv2Polyline, BzrSrf2Polylines, IritSurface2Polylines, , IritTrimSrf2Polylines, SymbSrf2Polylines, TrimSrf2Polylines, CagdSrf2Polylines, BspSrf2Polylines, CagdSrf2KnotCurves,

3.2.604 CagdSrf2PolyAdapSetAuxDataFunc (cagd2ply.c:185)

```
CagdSrfAdapAuxDataFuncType
            CagdSrf2PolyAdapSetAuxDataFunc(CagdSrfAdapAuxDataFuncType Func)
```

Func: New function to use, NULL to disable.

Returns: Old value of function.

Description: Sets the surface approximation auxiliary function. This function will be invoked on each subdivision step during the approximation process, for auxiliary processing that are application specific.
See also: CagdSrfAdap2Polygons, CagdSrf2PolyAdapSetPolyGenFunc, , CagdSrf2PolyAdapSetErrFunc,

3.2.605 CagdSrf2PolyAdapSetErrFunc (cagd2ply.c:151)

```
CagdSrfErrorFuncType CagdSrf2PolyAdapSetErrFunc(CagdSrfErrorFuncType Func,
                                                void *Data)
```

Func: New function to use, NULL to disable.

Data: New function data to use, NULL to disable.

Returns: Old value of function.

Description: Sets the surface approximation error function. The error function will return a negative value if this patch is flat enough, and positive value if flat enough. Either case, the magnitude will equal to the actual error.
See also: CagdSrfAdap2Polygons, CagdSrf2PolyAdapSetAuxDataFunc, , CagdSrf2PolyAdapSetPolyGenFunc,

3.2.606 CagdSrf2PolyAdapSetPolyGenFunc (cagd2ply.c:217)

```
CagdSrfAdapPolyGenFuncType
             CagdSrf2PolyAdapSetPolyGenFunc(CagdSrfAdapPolyGenFuncType Func)
```

Func: New function to use, NULL to disable.

Returns: Old value of function.

Description: Sets the function to convert flat surface rectangle domains into polygons.
See also: CagdSrfAdap2Polygons, CagdSrf2PolyAdapSetAuxDataFunc, , CagdSrf2PolyAdapSetErrFunc,

3.2.607 CagdSrf2PolygonFast (bsp2poly.c:72)

```
CagdBType CagdSrf2PolygonFast(CagdBType PolygonFast)
```

PolygonFast: New setting to use.

Returns: Old value.

Description: Sets the polygonal approximation of surfaces to create polygonal data ast and approximated (if TRUE) or slowly and exact (if FALSE).
See also: CagdSrf2Polygons, CagdSrf2PolygonStrip, CagdSrf2PolygonMergeCoplanar,

3.2.608 CagdSrf2PolygonMergeCoplanar (bzr2poly.c:68)

```
CagdBType CagdSrf2PolygonMergeCoplanar(CagdBType MergeCoplanarPolys)
```

MergeCoplanarPolys: New setting to use.

Returns: Old value.

Description: Sets the polygonal approximation of surfaces to mrge or not adjacent coplanar polygons into one. The default is to apply this optimization but in cases where a uniform mesh is need, it should be disabled.
See also: CagdSrf2Polygons, CagdSrf2PolygonFast, CagdSrf2PolygonStrip, Cagd2PolyClipPolysAtPoles,

3.2.609 CagdSrf2PolygonStrip (bsp2poly.c:43)

```
CagdBType CagdSrf2PolygonStrip(CagdBType PolygonStrip)
```

PolygonStrip: New setting to use.

Returns: Old value.

Description: Sets the polygonal approximation of surfaces to create polygonal strips (if TRUE) or regular individual polygons (if FALSE). If TRUE this hints the ability and desire to use polygonal strips but it does not guarantee that only polygonal strips would indeed be returned. Regular polygonal data should always be handled as well.
See also: CagdSrf2Polygons, CagdSrf2PolygonFast, CagdSrf2PolygonMergeCoplanar,

3.2.610 CagdSrf2Polygons (cagd_aux.c:432)

evaluation

polygonal approximation

```
IPPolygonStruct *CagdSrf2Polygons(const CagdSrfStruct *Srf,
                                  const CagdSrf2PlsInfoStrct *TessInfo)
```

Srf: To approximate into triangles.

TessInfo: All auxiliary information/state to tessellate.

Returns: A list of polygons with optional normal and/or UV parametric information. NULL is returned in case of an error.

Description: Routine to convert a single freeform surface to set of triangles approximating it. FineNess is a fineness control on result and the larger is more triangles may result. A value of 10 is a good start value. NULL is returned in case of an error, otherwise list of IPPolygonStruct.

See also: BzrSrf2Polygons, BspSrf2Polygons, CagdCrv2Polyline, CagdSrf2Polylines, , CagdSrf2PolygonStrip, CagdSrf2PolygonsN,

3.2.611 CagdSrf2PolygonsGenPolys (bzr2poly.c:254)

polygonization

surface approximation

```
IPPolygonStruct *CagdSrf2PolygonsGenPolys(const CagdSrfStruct *Srf,
                                          CagdRType *PtWeights,
                                          CagdPtStruct *PtMesh,
                                          CagdVecStruct *PtNrml,
                                          CagdUVStruct *UVMesh,
                                          int FineNessU,
                                          int FineNessV,
                                          const CagdSrf2PlsInfoStrct *TessInfo)
```

Srf: To approximate into triangles.

PtWeights: Weights of the evaluations, if rational, to detect poles. NULL if surface not rational. Freed by this function.

PtMesh: Evaluated positions of grid of samples. Freed by this function.

PtNrml: Evaluated normals of grid of samples or NULL if none. Freed by this function.

UVMesh: Evaluated UV vals of grid of samples or NULL if none. Freed by this function.

FineNessU, FineNessV: Actual size of PtMesh, PtNrml, UVMesh.

TessInfo: All auxiliary information/state to tessellate.

Returns: A list of polygons with optional normal and/or UV parametric information. NULL is returned in case of an error.

Description: Routine to convert uniform grid samples of a freeform srf to a set of triangles/rectangles/polystrips approximating it.

See also: BzrSrf2Polygons, BspSrf2Polygons, IritSurface2Polygons, , IritTrimSrf2Polygons, CagdSrf2Polygons, TrimSrf2Polygons,

3.2.612 CagdSrf2PolygonsN (cagd_aux.c:484)

evaluation

polygonal approximation

```
IPPolygonStruct *CagdSrf2PolygonsN(const CagdSrfStruct *Srf,
                                   int Nu,
                                   int Nv,
                                   CagdBType ComputeNormals,
                                   CagdBType FourPerFlat,
                                   CagdBType ComputeUV)
```

Srf: To approximate into triangles.

Nu, Nv: The number of uniform samples in U and V of surface.

ComputeNormals: If TRUE, normal information is also computed.

FourPerFlat: If TRUE, four triangles are created per flat surface. If FALSE, only 2 triangles are created.

ComputeUV: If TRUE, UV values are stored and returned as well.

Returns: A list of polygons with optional normal and/or UV parametric information. NULL is returned in case of an error.

Description: Routine to convert a single freeform surface to set of triangles approximating it using a uniform fixed resolution of Nu x Nv. NULL is returned in case of an error, otherwise list of IPPolygonStruct.

See also: BzrSrf2Polygons, BspSrf2Polygons, CagdCrv2Polyline, CagdSrf2Polylines, , CagdSrf2PolygonStrip, CagdSrf2Polygons,

3.2.613 CagdSrf2Polylines (cagd_aux.c:544)

polylines

isoparametric curves

```
CagdPolylineStruct *CagdSrf2Polylines(const CagdSrfStruct *Srf,
                                      int NumOfIsocurves[2],
                                      int SamplesPerCurve)
```

Srf: Srf to extract isoparametric curves from.

NumOfIsocurves: To extarct from Srf in each (U or V) direction.

SamplesPerCurve: Fineness control on piecewise linear curve approximation.

Returns: List of polygons representing a piecewise linear approximation of the extracted isoparamteric curves or NULL is case of an error.

Description: Routine to convert a single B-spline surface to NumOfIsolines polylines in each parametric direction with SamplesPerCurve in each isoparametric curve. Polyline are always E3 of CagdPolylineStruct type. Iso parametric curves are sampled equally spaced in parametric space. NULL is returned in case of an error, otherwise list of CagdPolylineStruct. Attempt is made to extract isolines along C1 discontinuities first.
See also: BspCrv2Polyline, BzrSrf2Polylines, IritSurface2Polylines, , IritTrimSrf2Polylines, SymbSrf2Polylines, TrimSrf2Polylines,

3.2.614 CagdSrfA2PGridFetchPts (cagd2pl2.c:567)

```
CagdSrfPtStruct *CagdSrfA2PGridFetchPts(struct CagdA2PGridStruct *A2PGrid,
                                        CagdSrfDirType Dir,
                                        int StartIndex,
                                        int EndIndex,
                                        int OtherDirIndex,
                                        CagdSrfPtStruct **LastPt,
                                        CagdBType Reversed)
```

A2PGrid: This grid data structure.

Dir: Are we to fetch sampled points along U or V direction?

StartIndex, EndIndex: Limit indices along this direction to fetch.

OtherDirIndex: Index along the other direction to fetch points.

LastPt: Will be set to last point in linked list.

Reversed: TRUE to fetch the linked list reversed.

Returns: Sampled points.

Description: Fetch an interval of sampled points along isoparametric direction.
See also: CagdSrfA2PGridInit,

3.2.615 CagdSrfA2PGridFetchRect (cagd2pl2.c:490)

```
CagdSrfPtStruct *CagdSrfA2PGridFetchRect(struct CagdA2PGridStruct *A2PGrid,
                                         int UIndex1,
                                         int VIndex1,
                                         int UIndex2,
                                         int VIndex2)
```

A2PGrid: This grid data structure.

UIndex1, VIndex1: Start point of rectangle domain.

UIndex2, VIndex2: End point of rectangle domain.

Returns: List of points found around the rectangle, or NULL if error.

Description: Fetch a all sampled points in the rectangle region defined from [UIndex1, VIndex1] to [UIndex2, VIndex2].
See also: CagdSrfA2PGridInit,

3.2.616 CagdSrfA2PGridFree (cagd2pl2.c:112)

```
void CagdSrfA2PGridFree(struct CagdA2PGridStruct *A2PGrid)
```

A2PGrid: The data structure to free.

Returns: void

Description: Free the grid data structure.
See also: CagdSrfA2PGridInit,

3.2.617 CagdSrfA2PGridInit (cagd2pl2.c:75)

```
struct CagdA2PGridStruct *CagdSrfA2PGridInit(const CagdSrfStruct *Srf)
```

Srf: Surface to prepare the grid structure point sampling support.

Returns: The structure if successful, NULL otherwise.

Description: Initializes a data structure to efficiently save UV sample locations on a surface and allow fast fetching of them as well.
See also: CagdSrfAdap2Polygons,

3.2.618 CagdSrfA2PGridInsertUV (cagd2pl2.c:156)

```
void CagdSrfA2PGridInsertUV(struct CagdA2PGridStruct *A2PGrid,
                            int UIndex,
                            int VIndex,
                            CagdRType u,
                            CagdRType v)
```

A2PGrid: This grid data structure.

UIndex, VIndex: Indices of these U / V parameter values.

u, v: The parameter values.

Returns: void

Description: Insert one UV location to sample the surfaces, into the data grid.
See also: CagdSrfA2PGridInit,

3.2.619 CagdSrfA2PGridProcessUV (cagd2pl2.c:290)

```
CagdBType CagdSrfA2PGridProcessUV(struct CagdA2PGridStruct *A2PGrid)
```

A2PGrid: This grid data structure.

Returns: TRUE if successful, FALSE otherwise.

Description: Once all surface sampled points are insert, this function is invoked to process the data for fast fetch. Two vectors, UGridVec and VGridVec are processed and updated to hold evaluate surface locations. Each entry in UGridVec (respectively in VGridVec) will hold a linked list of surface evaluated locations sort in V for that particular U value.
See also: CagdSrfA2PGridInit,

3.2.620 CagdSrfAdap2PolyDefErrFunc (cagd2ply.c:254)

```
CagdRType CagdSrfAdap2PolyDefErrFunc(const CagdSrfStruct *Srf,
                                     CagdRType Tolerance,
                                     void *AuxData)
```

Srf: Surface to test for flatness.

Tolerance: As to be used by the flatness testing.

AuxData: Optional call back data. Ignored here.

Returns: Negative value if flat enough, positive if not flat. Either case, magnitude will equal to the actual error.

Description: Tolerance evaluation of flatness for given surface. Constructs a plane from the four corner points, if possible, and measure distance to rest of control points.
See also: CagdSrfIsCoplanarCtlMesh, CagdSrfIsLinearBndryCtlMesh, , CagdSrfAdap2Polygons,

3.2.621 CagdSrfAdap2PolyEvalNrmlBlendedUV (cagd2ply.c:1567)

```
CagdRType *CagdSrfAdap2PolyEvalNrmlBlendedUV(const CagdRType *UV1,
                                             const CagdRType *UV2,
                                             const CagdRType *UV3,
                                             void *EvalNrmlCache,
                                             CagdRType *Nrml)
```

UV1: To compute the surface normal close to.

UV2, UV3: Two other UV"s of the triangle.

EvalNrmlCache: Cache to speed up normal evaluation.

Nrml: Where result will be saved.

Returns: Computed normal. Same as Nrml.

Description: Compute the normal to the surface very close to UV1 in the triangle defined by UV1/UV2/UV3.
See also: CagdSrfAdap2Polygons,

3.2.622 CagdSrfAdap2Polygons (cagd2ply.c:618)

polygonization

surface approximation

```
IPPolygonStruct *CagdSrfAdap2Polygons(const CagdSrfStruct *Srf,
                                      VoidPtr AuxSrfData,
                                      const CagdSrf2PlsInfoStrct *TessInfo)
```

Srf: To approximate into triangles.

AuxSrfData: Optional data structure that will be passed to all subdivided sub-surfaces, or NULL if not needed. See also CagdSrf2PolyAdapSetAuxDataFunc.

TessInfo: Aux. information for the srf to polys conv.

Returns: A list of polygons with optional normal and/or UV parametric information. NULL is returned in case of an error or if use of call back function to collect the polygons.

Description: Routine to convert a single surface to set of polygons approximating it. Tolerance is a tolerance control on result, typically related to the the accuracy of the approximation. A value of 0.1 is a good rough start. NULL is returned in case of an error or use of call back function to get a hold over the created polygons, otherwise list of IPPolygonStruct.
See also: CagdSrf2PolygonSetErrFunc, CagdSrfAdap2PolyDefErrFunc, , CagdSrf2PolyAdapSetErrFunc, CagdSrf2PolyAdapSe
CagdSrf2PolyAdapSetPolyGenFunc, BzrSrf2Polygons, CagdSrf2Polygons, , CagdSrf2Polygons, TrimSrf2Polygons,
BspC1Srf2Polygons, CagdSrf2Polygons,

3.2.623 CagdSrfAdapGetE3Pt (cagd2ply.c:293)

coercion

```
static void CagdSrfAdapGetE3Pt(CagdRType *E3Point,
                               CagdRType * const Points[CAGD_MAX_PT_SIZE],
                               int Index,
                               CagdPointType PType)
```

E3Point: Where the coerced information is to be saved.

Points: The control points vector of the surface.

Index: Index into the vectors of Points.

PType: Point type of Srf.

Returns: void

Description: Fetches an E3 point at given Index. Input control points are assumed E3 or P3 only.

3.2.624 CagdSrfAdapRectPolyGen (cagd2ply.c:1329)

polygonization

surface approximation

```
IPPolygonStruct *CagdSrfAdapRectPolyGen(const CagdSrfStruct *Srf,
                                        CagdSrfPtStruct *SrfPtList,
                                        const CagdSrfAdapRectStruct *Rect,
                                        const CagdSrf2PlsInfoStrct *TessInfo)
```

Srf: B-spline surface with no discontinuities to approximate into triangles.

SrfPtList: Circular list of a convex surface domain to convert to triangles.

Rect: The rectangular domain to convert to polygons.

TessInfo: All auxiliary information/state to tessellate.

Returns: List of polygons out of the closed srf pt list; Could be NULL if polygons are call back created.

Description: Converts the given circular list of surface points into polygons. The list is assumed a convex parametric domain (which ease the process of decomposition).

See also: CagdSrfAdap2Polygons, CagdSrf2PolygonSetErrFunc, , CagdSrfAdap2PolyDefErrFunc, CagdSrf2PolyAdapSetErrFu , CagdSrf2PolyAdapSetAuxDataFunc,

3.2.625 CagdSrfAre2SrfsSharingBndry (cagdsmrg.c:62)

```
CagdBType CagdSrfAre2SrfsSharingBndry(CagdSrfStruct *Srf1,
                                      CagdSrfBndryType Bndry1,
                                      CagdSrfStruct *Srf2,
                                      CagdSrfBndryType Bndry2,
                                      IrtRType Tolerance)
```

Srf1: First surface to consider.

Bndry1: First boundary of first surface to consider.

Srf2: Second surface to consider.

Bndry2: Second boundary of first surface to consider.

Tolerance: Tolerance of similarity.

Returns: TRUE if similar, FALSE otherwise.

Description: Compare designated two boundaries of given two surfaces for similarity.
See also: MvarAre2MVsSharingBndry,

3.2.626 CagdSrfArea (cagd2ply.c:1723)

```
CagdRType CagdSrfArea(const CagdSrfStruct *Srf, CagdRType Tol)
```

Srf: Surface to approximate its surface area.

Tol: Of approximation.

Returns: Approximated surface area.

Description: Compute an approximated surface area by polygonalizing Srf...
See also: TrivSrfArea,

3.2.627 CagdSrfAverageValue (cagdbbox.c:939)

bbox

bounding box

minimum

maximum

```
CagdRType CagdSrfAverageValue(const CagdSrfStruct *Srf, int Axis)
```

Srf: To compute an average value of its control mesh, in some Axis.

Axis: 1 for X, 2 for Y etc.

Returns: Average value.

Description: Computes an average value of all control points of given surface in a given axis. Rational values are taken into account (projected into Euclidean space first).
See also: CagdSrfMinMax, CagdPointsBBox, CagdCrvAverageValue,

3.2.628 CagdSrfAvgArgLenMesh (cagdsmrg.c:484)

arc length

```
CagdRType CagdSrfAvgArgLenMesh(const CagdSrfStruct *Srf,
                               CagdRType *AvgULen,
                               CagdRType *AvgVLen)
```

Srf: To computer average mesh edge length in the U and the V directions.

AvgULen, AvgVLen: Average length of edges of the control mesh of Srf in the U and V mesh directions.

Returns: The ratio of AvgULen / AvgVLen.

Description: Computes an average of edge lengths of edges of the control mesh of the given surface in the U direction and in the V direction.
See also: CagdCrvArcLenPoly, CagdLimitCrvArcLen,

3.2.629 CagdSrfBBox (cagdbbox.c:177)

bbox

bounding box

```
CagdBBoxStruct *CagdSrfBBox(const CagdSrfStruct *Srf, CagdBBoxStruct *BBox)
```

Srf: To compute a bounding box for.

BBox: Where bounding information is to be saved.

Returns: BBox.

Description: Computes a bounding box for a freeform surface.
See also: CagdCrvBBox, CagdSrfListBBox, GMBBSetBBoxPrecise, , CagdIgnoreNonPosWeightBBox, CagdPolygonBBox,

3.2.630 CagdSrfBelowPlane (cagdbbox.c:288)

```
CagdBType CagdSrfBelowPlane(const CagdSrfStruct *Srf, const CagdRType Pln[4])
```

Srf: The surface examine if below the plane Pln.

Pln: First points defining the line in the XY plane.

Returns: TRUE if Srf is entirely below Pln, or FALSE otherwise.

Description: Examines if the given surface is entirely below a given plane.

3.2.631 CagdSrfBlossomDegreeRaise (blossom.c:1388)

```
CagdSrfStruct *CagdSrfBlossomDegreeRaise(const CagdSrfStruct *Srf,
                                         CagdSrfDirType Dir)
```

Srf: Surface to degree raise.

Dir: Direction of degree raising. Either U or V.

Returns: Degree raised surface, or NULL if error.

Description: Computes a new surface with its degree raised once, given surface Srf, using blossoming.
See also: CagdSrfBlossomDegreeRaiseN, BspSrfDegreeRaise, BzrSrfDegreeRaise, CagdSrfDegreeRaise, CagdCrvBlossomDegreeRaise,

3.2.632 CagdSrfBlossomDegreeRaiseN (blossom.c:1225)

```
CagdSrfStruct *CagdSrfBlossomDegreeRaiseN(const CagdSrfStruct *Srf,
                                          int NewUOrder,
                                          int NewVOrder)
```

Srf: Surface to degree raise.

NewUOrder: New U order of Srf.

NewVOrder: New V order of Srf.

Returns: Degree raised surface, or NULL if error.

Description: Computes a new surface with its degree raised to NewOrder, given surface Srf, using blossoming.
See also: CagdSrfBlossomDegreeRaise, BspSrfDegreeRaise, BzrSrfDegreeRaise, CagdSrfDegreeRaise, CagdCrvBlossomDegreeRaiseN,

3.2.633 CagdSrfBlossomEvalMalloc (blossom.c:759)

```
CagdRType *CagdSrfBlossomEvalMalloc(const CagdSrfStruct *Srf,
                                    const CagdRType *BlsmUVals,
                                    int BlsmULen,
                                    const CagdRType *BlsmVVals,
                                    int BlsmVLen)
```

Srf: Surface to blossom.

BlsmUVals: U Blossoming values to consider.

BlsmULen: Length of BlsmUVals vector; assumed less than Srf U order!

BlsmVVals: V Blossoming values to consider.

BlsmVLen: Length of BlsmVVals vector; assumed less than Srf V order!

Returns: Evaluated Blossom.

Description: Computes the Blossom over the given surface, Srf, and Blossoming factors BlsmU/VVals.
See also: CagdSrfBlossomEvalU, CagdCrvBlossomEval,

3.2.634 CagdSrfBlossomEvalToData (blossom.c:673)

```
CagdRType *CagdSrfBlossomEvalToData(const CagdSrfStruct *Srf,
                                    const CagdRType *BlsmUVals,
                                    int BlsmULen,
                                    const CagdRType *BlsmVVals,
                                    int BlsmVLen,
                                    CagdRType *BlossomVals,
                                    void *BlsmCache)
```

Srf: Surface to blossom.

BlsmUVals: U Blossoming values to consider.

BlsmULen: Length of BlsmUVals vector; assumed less than Srf U order!

BlsmVVals: V Blossoming values to consider.

BlsmVLen: Length of BlsmVVals vector; assumed less than Srf V order!

BlossomVals: Evaluated Blossom will be placed here.

BlsmCache: Optional cache (can be NULL) to use, as allocated by CagdBlsmEvalSymbAllocCache and freed by CagdBlsmEvalSymbFreeCache.

Returns: Evaluated Blossom.

Description: Computes the Blossom over the given surface, Srf, and Blossoming factors BlsmU/VVals.
See also: CagdSrfBlossomEvalU, CagdCrvBlossomEval, CagdCrvBlossomEvalMalloc,

3.2.635 CagdSrfBlossomEvalU (blossom.c:796)

```
CagdCrvStruct *CagdSrfBlossomEvalU(const CagdSrfStruct *Srf,
                                   const CagdRType *BlsmUVals,
                                   int BlsmULen,
                                   void *BlsmCache)
```

Srf: Surface to blossom.

BlsmUVals: U Blossoming values to consider.

BlsmULen: Length of BlsmUVals vector; assumed less than Srf U order!

BlsmCache: Optional cache (can be NULL) to use, as allocated by CagdBlsmEvalSymbAllocCache and freed by CagdBlsmEvalSymbFreeCache.

Returns: Evaluated Blossom in U as a curve in V. This curve holds as many control points as Srf has in the V direction.

Description: Computes the Blossom over the given surface, Srf, and Blossoming factors BlsmUVals, in the U direction only. Returned is a curve in V.
See also: CagdSrfBlossomEval, CagdCrvBlossomEval,

3.2.636 CagdSrfCopy (cagd1gen.c:836)

copy

```
CagdSrfStruct *CagdSrfCopy(const CagdSrfStruct *Srf)
```

Srf: To be copied.

Returns: A duplicate of Srf.

Description: Allocates and copies all slots of a surface structure.

3.2.637 CagdSrfCopyList (cagd1gen.c:1218)

copy

```
CagdSrfStruct *CagdSrfCopyList(const CagdSrfStruct *SrfList)
```

SrfList: To be copied.

Returns: A duplicated list of surfaces.

Description: Allocates and copies a list of surface structures.

3.2.638 CagdSrfDegreeRaise (cagd_aux.c:2683)

degree raising

```
CagdSrfStruct *CagdSrfDegreeRaise(const CagdSrfStruct *Srf,
                                  CagdSrfDirType Dir)
```

Srf: To raise its degree.

Dir: Direction of degree raising. Either U or V.

Returns: A surface with same geometry as Srf but with one degree higher.

Description: Returns a new surface representing the same surface as Srf but with its degree raised by one.
See also: BzrSrfDegreeRaise, BspSrfDegreeRaise, TrimSrfDegreeRaise,

3.2.639 CagdSrfDegreeRaiseN (cagd_aux.c:2721)

degree raising

```
CagdSrfStruct *CagdSrfDegreeRaiseN(const CagdSrfStruct *Srf,
                                   int NewUOrder,
                                   int NewVOrder)
```

Srf: To raise its degree.

NewUOrder: New U order of Srf.

NewVOrder: New V order of Srf.

Returns: A surface with higher degrees as prescribed by NewUOrder/NewVOrder.

Description: Returns a new surface, identical to the original but with higher degrees, as prescribed by NewUOrder, NewVOrder.
See also: CagdSrfDegreeRaise, BzrSrfDegreeRaise, TrimSrfDegreeRaise, BspSrfDegreeRaise, BzrSrfDegreeRaiseN, BspSrfDegreeRaiseN, ,

3.2.640 CagdSrfDerive (cagd_aux.c:1054)

derivatives

partial derivatives

```
CagdSrfStruct *CagdSrfDerive(const CagdSrfStruct *Srf, CagdSrfDirType Dir)
```

Srf: To compute its derivative surface in direction Dir.

Dir: Direction of differentiation. Either U or V.

Returns: Resulting partial derivative surface.

Description: Given a surface, computes its partial derivative in the prescribed direction Dir.
See also: BzrSrfDerive, BspSrfDerive, SymbSrfDeriveRational, , CagdSrfDeriveScalar,

3.2.641 CagdSrfDeriveScalar (cagd_aux.c:1099)

derivatives

Hodograph

```
CagdSrfStruct *CagdSrfDeriveScalar(const CagdSrfStruct *Srf,
                                   CagdSrfDirType Dir)
```

Srf: To compute derivatives of all its components.

Dir: Direction of differentiation. Either U or V.

Returns: Resulting derivative.

Description: Given a surface, computes its partial derivative in the prescibed direction Dir of all is scalar components. For a Euclidean surface this is the same as CagdSrfDerive but for a rational surface the returned surfaces is not the vector field but simply the derivatives of all the surface's coefficients, including the weights.
See also: BzrSrfDerive, BspSrfDerive, CagdSrfIsSingular, SymbSrfDeriveRational, , CagdSrfDerive, BzrSrfDeriveScalar, BspSrfDeriveScalar,

3.2.642 CagdSrfDomain (cagd_aux.c:240)

domain

parametric domain

```
void CagdSrfDomain(const CagdSrfStruct *Srf,
                   CagdRType *UMin,
                   CagdRType *UMax,
                   CagdRType *VMin,
                   CagdRType *VMax)
```

Srf: To get its parametric domain.

UMin: Where to put the minimal U domain's boundary.

UMax: Where to put the maximal U domain's boundary.

VMin: Where to put the minimal V domain's boundary.

VMax: Where to put the maximal V domain's boundary.

Returns: void

Description: Returns the parametric domain of a surface.
See also: BspSrfDomain, CagdSrfSetDomain,

3.2.643 CagdSrfEffiNrmlEval (nrmleval.c:82)

```
CagdVecStruct *CagdSrfEffiNrmlEval(CagdRType u,
                                   CagdRType v,
                                   CagdBType Normalize,
                                   CagdVecStruct *RetVec,
                                   void *EvalCache)
```

u, v: Parameter values of the location on the surface to compute the normal for. For efficiency, no test is made as for the validity of the (u, v) position.

Normalize: If TRUE, the normal is normalized into a unit length.

RetVec: Where the result is going to be saved.

EvalCache: Evaluation cache computed by CagdSrfEffiNrmlPrelude.

Returns: A pointer to RetVec. An all zero vector is returned if failed to compute.

Description: Evaluate the surface normal at the given (u, v) surface location. The normal is normalized if Normalize is TRUE. For best performance normal locations with the same U values should be invoking this function in a sequence before moving on to a different U value.
See also: CagdSrfNormal, CagdSrfEffiNrmlPrelude, CagdSrfEffiNrmlPostlude,

3.2.644 CagdSrfEffiNrmlPostlude (nrmleval.c:165)

```
void CagdSrfEffiNrmlPostlude(void *EvalCache)
```

EvalCache: Evaluation cache computed by CagdSrfEffiNrmlPrelude to free.

Returns: void

Description: Released all data structures allocated by this efficient normal evaluation routines.
See also: CagdSrfNormal, CagdSrfEffiNrmlEval, CagdSrfEffiNrmlPrelude,

3.2.645 CagdSrfEffiNrmlPrelude (nrmleval.c:35)

```
void *CagdSrfEffiNrmlPrelude(const CagdSrfStruct *Srf)
```

Srf: To preprocess for fast normal evaluations.

Returns: The computed cache to use.

Description: Do the necessary preprocessing so we can efficiently evaluate normal on Srf. For best efficiency normals with same U values should be evaluated in a sequence, before moving to the next U.
See also: CagdSrfNormal, CagdSrfEffiNrmlEval, CagdSrfEffiNrmlPostlude,

3.2.646 CagdSrfEstimateCurveness (bzr2poly.c:1306)

```
void CagdSrfEstimateCurveness(const CagdSrfStruct *Srf,
                              CagdRType *UCurveness,
                              CagdRType *VCurveness)
```

Srf: To consider.

UCurveness: The surface curveness in the U direction.

VCurveness: The surface curveness in the V direction.

Returns: void

Description: Estimate a relative surface curveness measure in U and V (no twist consideration). A flat surface (or a bilinear) would return two zeros. A highly curved surface would return values near one.

3.2.647 CagdSrfEval4Corners (cagd1gen.c:2539)

```
void CagdSrfEval4Corners(const CagdSrfStruct *Srf,
                         CagdPType P00,
                         CagdPType P01,
                         CagdPType P10,
                         CagdPType P11)
```

Srf: Surface to evaluate at its four corners.

P00, P01, P10, P11: The four evaluated corners in E3.

Returns: void

Description: Simple function to evaluate a surface at its four corners.
See also:

3.2.648 CagdSrfEvalMalloc (cagd_aux.c:398)

evaluation

```
CagdRType *CagdSrfEvalMalloc(const CagdSrfStruct *Srf,
                             CagdRType u,
                             CagdRType v)
```

Srf: To evaluate at the given parametric location (u, v).

u, v: The parameter values at which the curve Crv is to be evaluated.

Returns: A vector holding all the coefficients of all components of surface Srf's point type. If for example the surface's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1). This vector is allocated dynamically.

Description: Given a surface and parameter values u, v, evaluate the surface at (u, v).
See also: CagdCrvEvalToData, BspSrfEvalAtParamToData, BzrSrfEvalAtParamToData, , CagdCrvEvalMalloc, CagdSrfEvalToData, TrimSrfEvalToData,

3.2.649 CagdSrfEvalToData (cagd_aux.c:351)

evaluation

```
void CagdSrfEvalToData(const CagdSrfStruct *Srf,
                       CagdRType u,
                       CagdRType v,
                       CagdRType *R)
```

Srf: To evaluate at the given parametric location (u, v).

u, v: The parameter values at which the curve Crv is to be evaluated.

R: A vector holding all the coefficients of all components of surface Srf's point type. If for example the surface's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Returns: void

Description: Given a surface and parameter values u, v, evaluate the surface at (u, v).
See also: CagdCrvEvalToData, BspSrfEvalAtParamToData, BzrSrfEvalAtParamToData, , BspSrfEvalAtParamToData, BzrSrfEvalAtParamToData, TrimSrfEvalToData,

3.2.650 CagdSrfExtensionDup (cagd2gen.c:2558)

```
CagdSrfStruct *CagdSrfExtensionDup(const CagdSrfStruct *Srf,
                                   CagdSrfBndryType Bndry)
```

Srf: Surface to extend.

Bndry: One (or two - UMin/UMax or VMin/VMax together) of the four boundaries of the surface.

Returns: Extended surface or NULL if failed (i.e. not closed).

Description: Extends a closed surface by duplicating it, in desired boundary dir.
See also: BspSrfExtension,

3.2.651 CagdSrfFree (cagd2gen.c:261)

free

```
void CagdSrfFree(CagdSrfStruct *Srf)
```

Srf: To be deallocated.

Returns: void

Description: Deallocates and frees all slots of a surface sstructure.

3.2.652 CagdSrfFreeList (cagd2gen.c:313)

free

```
void CagdSrfFreeList(CagdSrfStruct *SrfList)
```

SrfList: To be deallocated.

Returns: void

Description: Deallocates and frees a surface structure list:

3.2.653 CagdSrfFromCrvs (cagdcsrf.c:142)

surface constructors

```
CagdSrfStruct *CagdSrfFromCrvs(const CagdCrvStruct *CrvList,
                               int OtherOrder,
                               CagdEndConditionType OtherEC,
                               IrtRType *OtherParamVals)
```

CrvList: List of curves to consturct a surface with.

OtherOrder: Other order of surface.

OtherEC: End condition in the other (non CrvList) srf direction.

OtherParamVals: If not NULL, updated with other direction set parameters of the curves in the new surfaces.

Returns: Constructed surface from curves.

Description: Constructs a surface using a set of curves. Curves are made to be compatible and then each is substituted into the new surface's mesh as a row. If the OtherOrder is less than the number of curves, number of curves is used. If OtherOrder is negative, the absolute value is employed and a periodic surface is constructed in the other direction. A knot vector is formed with OtherEC end conditions for the other direction Note, however, that only the first and the last curves are interpolated if open end conditions are selected and OtherOrder is greater than 2.

See also: CagdSrfInterpolateCrvs,

3.2.654 CagdSrfFromNBndryCrvs (cagdbsum.c:1040)

NGon

NSided

```
CagdSrfStruct *CagdSrfFromNBndryCrvs(const CagdCrvStruct *Crvs,
                                     CagdBType MinimizeSize)
```

Crvs: To build 1/2 surfaces with these curves as boundaries.

MinimizeSize: If true, minimize the size of the output, on expense of accuracy. N

Returns: One or two planar surfaces spanning the curves.

Description: Builds tensor product surfaces that spans the given list of surface boundary curves in one closed loop. Can be 1 to 6 input boundary curves, and 1 to 2 surfaces are returned.

See also: CagdReorderCurvesInLoop,

3.2.655 CagdSrfIntegrate (cagd_aux.c:1138)

integrals

```
CagdSrfStruct *CagdSrfIntegrate(const CagdSrfStruct *Srf, CagdSrfDirType Dir)
```

Srf: To compute its integral surface.

Dir: Direction of integration. Either U or V.

Returns: Resulting integral surface.

Description: Given a surface, compute its integral surface.
See also: BzrSrfIntegrate, BspSrfIntegrate,

3.2.656 CagdSrfInterpolateCrvs (cagdcsrf.c:289)

surface constructors

interpolation

```
CagdSrfStruct *CagdSrfInterpolateCrvs(const CagdCrvStruct *CrvList,
                                      int OtherOrder,
                                      CagdEndConditionType OtherEC,
                                      CagdParametrizationType OtherParam,
                                      IrtRType *OtherParamVals)
```

CrvList: List of curves to consturct a surface with.

OtherOrder: Other order of surface.

OtherEC: End condition in the other (non CrvList) srf direction.

OtherParam: Currently only Chord length and uniform are supported.

OtherParamVals: If not NULL, updated with other direction set parameters of the curves in the new surfaces.

Returns: Constructed surface from curves.

Description: Constructs a surface using a set of curves. Curves are made to be compatible and then interpolated by the created surfaces. If the OtherOrder is less than the number of curves, number of curves is used. If OtherOrder is negative, the absolute value is employed and a periodic surface is constructed in the other direction. A knot vector is formed with OtherEC end conditions for the other direction.
See also: CagdSrfFromCrvs, CagdSrfInterpolateCrvsChordLenParams,

3.2.657 CagdSrfInterpolateCrvsChordLenParams (cagdcsrf.c:214)

surface constructors

```
CagdRType *CagdSrfInterpolateCrvsChordLenParams(const CagdCrvStruct *CrvList)
```

CrvList: List of curves to consturct a surface with.

Returns: Vectors of parameters normalized to [0, 1] of parameters, of size of number of curves, allocated dynamically.

Description: Computes parameters to interpolate the given curves at, as a surface. Estimate a middle point from each curve and set parameters based on chord length from each middle point to the next.
See also: CagdSrfFromCrvs, CagdSrfInterpolateCrvs, CagdSrfInterpolateCrvs,

3.2.658 CagdSrfIsConstant (cagdbbox.c:326)

bbox

bounding box

```
CagdBType CagdSrfIsConstant(const CagdSrfStruct *Srf, IrtRType Eps)
```

Srf: To check if constant or not.

Eps: Tolerance of equality allowed.

Returns: TRUE if indeed a constant(s) valued surface.

Description: Checks if this surface is a constant surface.
See also: CagdSrfBBox, CagdCrvIsConstant,

3.2.659 CagdSrfIsCoplanarCtlMesh (cagd2ply.c:494)

```
CagdRType CagdSrfIsCoplanarCtlMesh(const CagdSrfStruct *Srf)
```

Srf: Surface to test for flatness of its control mesh.

Returns: A bound on the distance between the control points and the plane fitted to the four corners.

Description: evaluate the coplanarity of the control mesh for a given surface. Constructs a plane from the four corner points, if possible, and measure distance to rest of control points.
See also: CagdSrfAdap2Polygons, CagdSrfAdap2PolyDefErrFunc,

3.2.660 CagdSrfIsLinearBndryCtlMesh (cagd2ply.c:458)

```
CagdRType CagdSrfIsLinearBndryCtlMesh(const CagdSrfStruct *Srf)
```

Srf: Surface to test for linearity of of its control mesh's boundary.

Returns: A bound on the distance between the control mesh boundary and a linear rectangle connecting the four corners.

Description: Evaluate the linearity of the boundary of the control mesh for a given surface. Constructs a line for each boundary, if possible, and measure distance to rest of control points on that boundary.
See also: CagdSrfAdap2Polygons, CagdSrfAdap2PolyDefErrFunc,

3.2.661 CagdSrfIsLinearCtlMesh (cagd2ply.c:417)

```
CagdRType CagdSrfIsLinearCtlMesh(const CagdSrfStruct *Srf, CagdBType Interior)
```

Srf: Surface to test for linearity of its control mesh's boundary.

Interior: TRUE to handle interior rows/columns only. FALSE to check Boundary as well.

Returns: A bound on the distance between the control mesh boundary and a linear rectangle connecting the four corners.

Description: Evaluate the linearity of the control mesh for a given surface. Constructs a line for each row/col, if possible, and measure distance to rest of control points on that row/col.
See also: CagdSrfAdap2Polygons, CagdSrfAdap2PolyDefErrFunc, CagdSrfIsLinearBndryCtlMesh, CagdSrfIsLinearCtlMeshOneRowCol,

3.2.662 CagdSrfIsLinearCtlMeshOneRowCol (cagd2ply.c:335)

```
CagdRType CagdSrfIsLinearCtlMeshOneRowCol(const CagdSrfStruct *Srf,
                                          int Idx,
                                          CagdSrfDirType Dir)
```

Srf: Surface to test for linearity of its control mesh's row/col.

Idx: Of row/column.

Dir: A row or column specification.

Returns: A bound on the distance between the control points on the row/col and the line through end points of that row/column.

Description: Evaluate the linearity of the control mesh for a given surface, along ne row or column. Constructs a line through the two end points and measure distance to rest of control points on that row/col.
See also: CagdSrfAdap2Polygons, CagdSrfAdap2PolyDefErrFunc, CagdSrfIsLinearBndryCtlMesh, CagdSrfIsLinearCtlMesh,

3.2.663 CagdSrfIsPtIndexBoundary (cagd1gen.c:3260)

```
CagdSrfBndryType CagdSrfIsPtIndexBoundary(CagdSrfStruct *Srf, int PtIdx)
```

Srf: Surface to examine if PtIdx is of a boundary control point.

PtIdx: Index into Points vectors to examine if of a boundary point.

Returns: Boundary type, or CAGD_NO_BNDRY if interior.

Description: Checks if the given control point index (in the Points vector of control points in CagdSrfStruct) is of a boundary control point.

3.2.664 CagdSrfIsSingular (cagd_aux.c:1180)

derivatives

```
CagdBType CagdSrfIsSingular(const CagdSrfStruct *Srf)
```

Hodograph

Srf: To check if it is singular.

Returns: TRUE if singular, FALSE if not.

Description: Given a surface, checks if it is whole singular by:

1. Examine if its derivative with respect to U is identically zero.
2. Examine if its derivative with respect to V is identically zero.

See also: BzrSrfDerive, BspSrfDerive, CagdSrfDeriveScalar, SymbSrfDeriveRational, , CagdSrfDerive, BzrSrfDeriveScalar, BspSrfDeriveScalar, , CagdSrfsFilterSingular,

3.2.665 CagdSrfListBBox (cagdbbox.c:260)

bbox

```
CagdBBoxStruct *CagdSrfListBBox(const CagdSrfStruct *Srfs, CagdBBoxStruct *BBox)
```

bounding box

Srfs: To compute a bounding box for.

BBox: Where bounding information is to be saved.

Returns: BBox

Description: Computes a bounding box for a list of freeform surfaces.
See also: CagdCrvBBox, CagdSrfBBox, CagdPolygonListBBox, GMBBSetBBoxPrecise,

3.2.666 CagdSrfListMatTransform (cagd2gen.c:1961)

scaling

rotation

translation

```
CagdSrfStruct *CagdSrfListMatTransform(const CagdSrfStruct *Srfs,
                                       CagdMType Mat)
```

transformations

Srfs: To be transformed.

Mat: Defining the transformation.

Returns: Returned transformed surfaces.

Description: Applies an homogeneous transformation, to the given list of surfaces Srfs as specified by homogeneous transformation Mat.
See also: CagdTransform, CagdSrfMatTransform, CagdMatTransform, , CagdSrfMatTransform,

3.2.667 CagdSrfMakeBoundryIndexUMin (cagd_aux.c:3883)

```
CagdSrfStruct *CagdSrfMakeBoundryIndexUMin(const CagdSrfStruct *Srf,
                                           int BndryIdx)
```

Srf: Surface to reverse.

BndryIdx: Boundary index, between 0 to 3 as (UMin, UMax, VMin, VMax)

Returns: Properly reversed surface, NULL if error.

Description: Reverse the surface so the boundary index (0,..,3) for (UMin, UMax, VMin VMax) will become the UMin boundary.
See also: CagdSrfReverse, CagdSrfReverse2, CagdSrfReverseDir,

3.2.668 CagdSrfMatTransCenter (cagd2gen.c:1469)

transformations

```
void CagdSrfMatTransCenter(CagdSrfStruct *Srf, IrtHmgnMatType Mat)
```

Srf: To be transformed around the center of the surface.

Mat: Transformation to apply.

Returns: void

Description: Applies a transform, in place, to given surface Srf as specified by Mat, around the center of the surface.
See also: CagdSrfTransform, CagdTransform, CagdCrvMatTransform, CagdCrvRotateToXY, CagdCrvTransform, CagdCrvMatTransCenter,

3.2.669 CagdSrfMatTransform (cagd2gen.c:1897)

scaling
rotation
translation
transformations

```
CagdSrfStruct *CagdSrfMatTransform(const CagdSrfStruct *Srf,
                                   CagdMType Mat)
```

Srf: To be transformed.

Mat: Defining the transformation.

Returns: Returned transformed surface.

Description: Applies an homogeneous transformation, to the given surface Srf as specified by homogeneous transformation Mat.
See also: CagdTransform, CagdCrvMatTransform, CagdMatTransform, , CagdSrfListMatTransform,

3.2.670 CagdSrfMinMax (cagdbbox.c:896)

bbox
bounding box
minimum
maximum

```
void CagdSrfMinMax(const CagdSrfStruct *Srf,
                   int Axis,
                   CagdRType *Min,
                   CagdRType *Max)
```

Srf: To test for minimum/maximum.

Axis: 0 for W, 1 for X, 2 for Y etc.

Min: Where minimum found value should be place.

Max: Where maximum found value should be place.

Returns: void

Description: Computes a min max bound on a surface in a given axis. The surface is not coerced to anything and the given axis is tested directly where 0 is the W axis and 1, 2, 3 are the X, Y, Z etc.
See also: CagdSrfAverageValue, CagdPointsBBox, CagdCrvMinMax,

3.2.671 CagdSrfMoebiusTransform (cagd_aux.c:1265)

moebius transformation

```
CagdSrfStruct *CagdSrfMoebiusTransform(const CagdSrfStruct *Srf,
                                       CagdRType c,
                                       CagdSrfDirType Dir)
```

Srf: Surface to apply the Moebius transformation to.

c: The scaling coefficient - c^n is the ratio between the first and last weight of the surface, along each row or column. If c == 0, the first and last weights are made equal, in the first row/column.

Dir: Direction to apply the Moebius transformation, row or col. If Dir == CAGD_BOTH_DIR, the transformation is applied to both the row and column directions, in this order.

Returns: Resulting surface after the moebius transformation.

Description: Given a surface, compute its moebius transformation.
See also: BzrSrfMoebiusTransform, BspSrfMoebiusTransform,

3.2.672 CagdSrfNew (cagd1gen.c:163)

allocation

```
CagdSrfStruct *CagdSrfNew(CagdGeomType GType,
                          CagdPointType PType,
                          int ULength,
                          int VLength)
```

GType: Type of geometry the surface should be - Bspline, Bezier etc.

PType: Type of control points (E2, P3, etc.).

ULength: Number of control points in the U direction.

VLength: Number of control points in the V direction.

Returns: An uninitialized freeform surface.

Description: Allocates the memory required for a new surface.
See also: BzrSrfNew, BspPeriodicSrfNew, BspSrfNew, CagdPeriodicSrfNew, TrimSrfNew,

3.2.673 CagdSrfNodes (bsp_knot.c:1748)

node values

```
CagdRType *CagdSrfNodes(const CagdSrfStruct *Srf, CagdSrfDirType Dir)
```

Srf: To compute node values for.

Dir: Either the U or the V parametric direction.

Returns: Node values of the given surface and given parametric direction.

Description: Returns the nodes of a freeform surface.

3.2.674 CagdSrfNormalMalloc (cagd_aux.c:3546)

normal

```
CagdVecStruct *CagdSrfNormalMalloc(const CagdSrfStruct *Srf,
                                   CagdRType u,
                                   CagdRType v,
                                   CagdBType Normalize)
```

Srf: To compute (unit) normal vector for.

u, v: Location where to evaluate the normal of Srf.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, length is a function of given parametrization.

Returns: A pointer to a vector holding the unit normal information.

Description: Given a surface Srf and a parameter values u, v, returns the (unit) normal vector of Srf.
See also: BzrSrfNormal, BspSrfNormal, SymbSrfNormalSrf, TrngTriSrfNrml, , pSrfMeshNormals,

3.2.675 CagdSrfNormalToData (cagd_aux.c:3505)

normal

```
CagdVecStruct *CagdSrfNormalToData(const CagdSrfStruct *Srf,
                                   CagdRType u,
                                   CagdRType v,
                                   CagdBType Normalize,
                                   CagdVecStruct *Normal)
```

Srf: To compute (unit) normal vector for.

u, v: Location where to evaluate the normal of Srf.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, length is a function of given parametrization.

Normal: A pointer to a vector holding the unit normal information.

Returns: A pointer to a vector holding the unit normal information.

Description: Given a surface Srf and a parameter values u, v, returns the (unit) normal vector of Srf.
See also: BzrSrfNormal, BspSrfNormal, SymbSrfNormalSrf, TrngTriSrfNrml, , pSrfMeshNormals,

3.2.676 CagdSrfPtCopy (cagd1gen.c:974)

copy

```
CagdSrfPtStruct *CagdSrfPtCopy(const CagdSrfPtStruct *Pt)
```

Pt: To be copied.

Returns: A duplicate of SrfPt.

Description: Allocates and copies all slots of a surface Pt structure.

3.2.677 CagdSrfPtCopyList (cagd1gen.c:1324)

copy

```
CagdSrfPtStruct *CagdSrfPtCopyList(const CagdSrfPtStruct *SrfPtList)
```

SrfPtList: To be copied.

Returns: A duplicated list of points.

Description: Allocates and copies a list of surface point structures.

3.2.678 CagdSrfPtFree (cagd2gen.c:455)

free

```
void CagdSrfPtFree(CagdSrfPtStruct *SrfPt)
```

SrfPt: To be deallocated.

Returns: void

Description: Deallocates and frees all slots of a point structure.

3.2.679 CagdSrfPtFreeList (cagd2gen.c:478)

free

```
void CagdSrfPtFreeList(CagdSrfPtStruct *SrfPtList)
```

SrfPtList: To be deallocated.

Returns: void

Description: Deallocates and frees a point structure list:

3.2.680 CagdSrfPtNew (cagd1gen.c:359)

allocation

```
CagdSrfPtStruct *CagdSrfPtNew(void)
```

Returns: A surface Pt structure.

Description: Allocates and resets all slots of a Surface Pt structure.

3.2.681 CagdSrfRefineAtParams (cagd_aux.c:3181)

refinement

subdivision

```
CagdSrfStruct *CagdSrfRefineAtParams(const CagdSrfStruct *Srf,
                                     CagdSrfDirType Dir,
                                     CagdBType Replace,
                                     CagdRType *t,
                                     int n)
```

Srf: To refine.

Dir: Direction of refinement. Either U or V.

Replace: If TRUE, t holds knots in exactly the same length as the length of the knot vector of Srf and t simply replaces the knot vector.

t: Vector of knots with length of n.

n: Length of vector t.

Returns: A refined curve of Srf after insertion of all the knots as specified by vector t of length n.

Description: Given a surface - refines it at the given n knots as defined by vector t. If Replace is TRUE, the values in t replaces current knot vector. Returns pointer to refined surface (Note a Bezier surface will be converted into a B-spline surface).

3.2.682 CagdSrfRegionFromSrf (cagd_aux.c:3053)

regions

subdivision

```
CagdSrfStruct *CagdSrfRegionFromSrf(const CagdSrfStruct *Srf,
                                    CagdRType t1,
                                    CagdRType t2,
                                    CagdSrfDirType Dir)
```

Srf: To extract a sub-region from.

t1, t2: Parametric domain boundaries of sub-region.

Dir: Direction of region extraction. Either U or V.

Returns: Sub-region extracted from Srf from t1 to t2.

Description: Given a surface - extracts a sub-region within the domain specified by t1 and t2, in the direction Dir.

3.2.683 CagdSrfRegionFromSrf2 (cagd_aux.c:3140)

regions

subdivision

```
CagdSrfStruct *CagdSrfRegionFromSrf2(const CagdSrfStruct *Srf,
                                     CagdRType UMin,
                                     CagdRType UMax,
                                     CagdRType VMin,
                                     CagdRType VMax)
```

Srf: To extract a sub-region from.

UMin, UMax, VMin, VMax: Parametric domain boundaries of sub-region.

Returns: Sub-region extracted from Srf from t1 to t2.

Description: Given a surface - extracts a sub-region within the domain specified by t1 and t2, in the direction Dir.

3.2.684 CagdSrfReverse (cagd_aux.c:3701)

reverse

```
CagdSrfStruct *CagdSrfReverse(const CagdSrfStruct *Srf)
```

Srf: To be reversed.

Returns: Reversed surface of Srf.

Description: Returns a new surface that is the reversed surface of Srf by reversing the control mesh and the knot vector (if B-spline surface) of Srf in the U direction. See also BspKnotReverse.
See also: CagdSrfReverse2, CagdSrfReverseDir,

3.2.685 CagdSrfReverse2 (cagd_aux.c:3816)

reverse

```
CagdSrfStruct *CagdSrfReverse2(const CagdSrfStruct *Srf)
```

Srf: To be reversed.

Returns: Reversed surface of Srf.

Description: Returns a new surface that is the reversed surface of Srf by flipping the U and the V directions of the surface. See also BspKnotReverse.
See also: CagdSrfReverse, CagdSrfReverseDir, CagdSrfMakeBoundryIndexUMin,

3.2.686 CagdSrfReverseDir (cagd_aux.c:3725)

reverse

```
CagdSrfStruct *CagdSrfReverseDir(const CagdSrfStruct *Srf, CagdSrfDirType Dir)
```

Srf: To be reversed.

Dir: Direction to reverse the Mesh along. Either U or V.

Returns: Reversed surface of Srf.

Description: Returns a new surface that is the reversed surface of Srf by reversing the control mesh and the knot vector (if B-spline surface) of Srf in the Dir direction. See also BspKnotReverse.
See also: CagdSrfReverse, CagdSrfReverse2, CagdSrfMakeBoundryIndexUMin,

3.2.687 CagdSrfScale (cagd2gen.c:1394)

scaling

transformations

```
void CagdSrfScale(CagdSrfStruct *Srf, const CagdRType *Scale)
```

Srf: To be non-uniformly scaled.

Scale: Scaling amount.

Returns: void

Description: Applies a nonuniform scaling transform, in place, to given Srf as specified by Scale.
See also: CagdCrvTransform, CagdTransform, CagdSrfMatTransform, , CagdSrfTransform,

3.2.688 CagdSrfScaleCenter (cagd2gen.c:1431)

scaling

transformations

```
void CagdSrfScaleCenter(CagdSrfStruct *Srf, const CagdRType *Scale)
```

Srf: To be non-uniformly scaled.

Scale: Scaling amount.

Returns: void

Description: Applies a nonuniform scaling transform, in place, to given Srf as specified by Scale.
See also: CagdCrvTransform, CagdTransform, CagdSrfMatTransform, , CagdSrfTransform, CagdSrfScale,

3.2.689 CagdSrfSetDomain (cagd_aux.c:283)

domain

parametric domain

```
CagdSrfStruct *CagdSrfSetDomain(CagdSrfStruct *Srf,
                                CagdRType UMin,
                                CagdRType UMax,
                                CagdRType VMin,
                                CagdRType VMax)
```

Srf: To set its parametric domain.

UMin: Minimal domain's new U boundary.

UMax: Maximal domain's new U boundary.

VMin: Minimal domain's new V boundary.

VMax: Maximal domain's new V boundary.

Returns: Modified surface, in place.

Description: Affinely set the parametric domain of a surface, in place.
See also: BspSrfDomain, BspKnotAffineTrans2, CagdCrvSetDomain, CagdSrfDomain,

3.2.690 CagdSrfSetMakeOnlyTri (cagd2gen.c:2724)

```
CagdBType CagdSrfSetMakeOnlyTri(CagdBType OnlyTri)
```

OnlyTri: TRUE for triangles only, FALSE otherwise.

Returns: Old value of flag.

Description: Sets a flag to control if only triangles are to be generated from the tesselation code. If TRUE only triangular polygons will be in the output set
See also: CagdSrfSetMakeRectFunc, CagdSrf2Polygons, CagdSrfAdap2Polygons, , CagdMakeTriangle, CagdSrfSetMakeTriFunc,

3.2.691 CagdSrfSetMakeRectFunc (cagd2gen.c:2692)

```
CagdSrfMakeRectFuncType CagdSrfSetMakeRectFunc(CagdSrfMakeRectFuncType Func)
```

Func: New function to use, NULL to disable.

Returns: Old value of function.

Description: Sets the call back function to generate rectangles. The function will be invoked with each rectangle in the polygonal approximation. Default call back function used is IPSetRectangle.
See also: CagdSrfSetMakeTriFunc, CagdSrf2Polygons, CagdSrfAdap2Polygons, , IPSetRectangle, CagdSrfSetMakeOnlyTri,

3.2.692 CagdSrfSetMakeTriFunc (cagd2gen.c:2660)

```
CagdSrfMakeTriFuncType CagdSrfSetMakeTriFunc(CagdSrfMakeTriFuncType Func)
```

Func: New function to use, NULL to disable.

Returns: Old value of function.

Description: Sets the call back function to generate triangles. The function will be invoked with each triangle in the polygonal approximation. Default call back function used is CagdMakeTriangle.
See also: CagdSrfSetMakeRectFunc, CagdSrf2Polygons, CagdSrfAdap2Polygons, , CagdMakeTriangle, CagdSrfSetMakeOnlyTri,

3.2.693 CagdSrfSrfMakeJoinMatch (cagdsmrg.c:722)

match

```
CagdBType CagdSrfSrfMakeJoinMatch(CagdSrfStruct **Srf1,
                                  CagdSrfStruct **Srf2,
                                  CagdSrfDirType Dir,
                                  CagdRType Tolerance,
                                  CagdBType PreserveOrientation)
```

Srf1: To modify its end to meet Srf2's starting location, in place.

Srf2: To modify its end to meet Srf1's starting location, in place.

Dir: The direction of the boundary curves to be matched.

Tolerance: To consider two points the same.

PreserveOrientation: If false, Srf1 and Srf2 will be reversed as needed to ensure that the end curve of Srf1 is the same as the initial curve of Srf2 in the given direction. Otherwise, the surfaces' won't be reversed.

Returns: TRUE if matching has been done succesfully.

Description: If Srf1 and Srf2 (approximately) share a boundary in the given direction, then the shared boundary curve of each surface is updated to the average of both boundary curves, so the two surfaces will meet precisecely. This function assumes Srf1 and Srf2 have the same lengthesand orders.

3.2.694 CagdSrfSubdivAtAllC0Discont (cagd_aux.c:2473)

```
CagdSrfStruct *CagdSrfSubdivAtAllC0Discont(const CagdSrfStruct *Srf)
```

Srf: To subdivide at all C^0 discontinuity locations.

Returns: Surfaces that result from the subdivision.

Description: Subdivides the surface at all C^0 potential discontinuity locations.
See also: CagdSrfSubdivAtParams, BspKnotAllC1Discont, , CagdSrfsSubdivAtAllC0Discont, BspSrfsSubdivAtAllDetectedLocations, , CagdSrfsSubdivAtAllC1Discont, CagdSrfsSubdivAtAllC0Discont,

3.2.695 CagdSrfSubdivAtAllC1Discont (cagd_aux.c:2529)

```
CagdSrfStruct *CagdSrfSubdivAtAllC1Discont(const CagdSrfStruct *Srf)
```

Srf: To subdivide at all C^1 discontinuity locations.

Returns: Surfaces that result from the subdivision.

Description: Subdivides the surface at all C^1 potential discontinuity locations.
See also: CagdSrfSubdivAtParams, BspKnotAllC1Discont, , CagdSrfSubdivAtAllC0Discont, BspSrfsSubdivAtAllDetectedLocations, , CagdSrfsSubdivAtAllC1Discont,

3.2.696 CagdSrfSubdivAtAllCnDiscont (cagd_aux.c:2584)

```
CagdSrfStruct *CagdSrfSubdivAtAllCnDiscont(const CagdSrfStruct *Srf, int n)
```

Srf: To subdivide at all C^n discontinuity locations.

n: Level of discontinuity to consider.

Returns: A list Surfaces that result from the subdivision.

Description: Subdivides a surface at all C^n potential discontinuity locations.
See also: CagdSrfsSubdivAtAllC0Discont, CagdSrfsSubdivAtAllC1Discont,

3.2.697 CagdSrfSubdivAtParam (cagd_aux.c:3018)

subdivision

```
CagdSrfStruct *CagdSrfSubdivAtParam(const CagdSrfStruct *Srf,
                                    CagdRType t,
                                    CagdSrfDirType Dir)
```

Srf: To subdivide at the prescibed parameter value t.

t: The parameter to subdivide the curve Crv at.

Dir: Direction of subdivision. Either U or V.

Returns: A list of the two surfaces resulting from the process of subdivision.

Description: Given a surface - subdivides it into two sub-surfaces at given parametric value t in the given direction Dir. Returns pointer to first surface in a list of two subdivided surfaces.

3.2.698 CagdSrfSubdivAtPoles (cagd_aux.c:2630)

subdivision

```
CagdSrfStruct *CagdSrfSubdivAtPoles(const CagdSrfStruct *Srf, CagdRType Tol)
```

Srf: To subdivide at all poles.

Tol: Domain size to subdive to this tolerance.

Returns: A list of surface patches with only positive weights.

Description: Given a surface - subdivides it at domain with poles (zero deniminator). Subdivision will terminate either when the domain size is smaller than Tol or the current patch has only positive (or only negative weights). Patches with whole negative weights are scaled by -1, and an attribute "NegPoles" is attached to them.
See also: CagdCrvSubdivAtParams, CagdCrvSubdivAtPoles,

3.2.699 CagdSrfTangentToData (cagd_aux.c:3461)

tangent

```
CagdVecStruct *CagdSrfTangentToData(const CagdSrfStruct *Srf,
                                    CagdRType u,
                                    CagdRType v,
                                    CagdSrfDirType Dir,
                                    CagdBType Normalize,
                                    CagdVecStruct *Tan)
```

Srf: To compute (unit) tangent vector for.

u, v: Location where to evaluate the tangent of Srf.

Dir: The OTHER direction (for historic reasons...) of tangent vector. Either U or V.

Normalize: If TRUE, attempt is made to normalize the returned vector. If FALSE, length is a function of given parametrization.

Tan: A pointer to a vector holding the unit tangent information.

Returns: A pointer to a vector holding the unit tangent information.

Description: Given a surface Srf and a parameter values u, v, returns the (unit) tangent vector of Srf in direction Dir.
See also: BzrSrfTangent, BspSrfTangent,

3.2.700 CagdSrfTransInnerCtlPts2Pt (cagd2gen.c:1508)

```
void CagdSrfTransInnerCtlPts2Pt(CagdSrfStruct *Srf,
                                const IrtPtType Pt,
                                IrtRType Ratio)
```

Srf: The modified surface, in place.

Pt: The point to translate all inner control points to.

Ratio: The translation ratio from zero to one.

Returns: void

Description: Translates all inner control points by (Ratio * |Pt - Cp|), where Cp is the location of the current control point.
See also:

3.2.701 CagdSrfTransform (cagd2gen.c:1354)

scaling

translation

transformations

```
void CagdSrfTransform(CagdSrfStruct *Srf,
                      const CagdRType *Translate,
                      CagdRType Scale)
```

Srf: To be affinely transformed.

Translate: Translation amount, NULL for non.

Scale: Scaling amount.

Returns: void

Description: Applies an affine transform, in place, to given surface Srf as specified by Translate and Scale. Each control point is first translated by Translate and then scaled by Scale.
See also: CagdCrvTransform, CagdTransform, CagdSrfMatTransform,

3.2.702 CagdSrfUVDirOrthoE3Malloc (cagd_aux.c:3675)

```
CagdUVType *CagdSrfUVDirOrthoE3Malloc(const CagdSrfStruct *Srf,
                                      const CagdUVType *UV,
                                      const CagdUVType *UVDir)
```

Srf: Surface to compute orthogonal direction to, in its tangent plane.

UV: Location on Srf where to compute the orthogonal direction.

UVDir: Direction to compute its orthogonal direction in Euclidean space.

Returns: UV direction that is orthogonal to UVDir, in Euclidean space, allocated dynamically. NULL, if error.

Description: Computes a new parametric direction OrthoUVDir that is orthogonal, in Euclidean space, to given parametric UVDir at parametric position UV of Srf. Clearly both Srf(OrthoUVDir) and Srf(UVDir) are in the tangent space.

3.2.703 CagdSrfUVDirOrthoE3ToData (cagd_aux.c:3578)

```
CagdUVType *CagdSrfUVDirOrthoE3ToData(const CagdSrfStruct *Srf,
                                      const CagdUVType *UV,
                                      const CagdUVType *UVDir,
                                      CagdUVType *OrthoUVDir)
```

Srf: Surface to compute orthogonal direction to, in its tangent plane.

UV: Location on Srf where to compute the orthogonal direction.

UVDir: Direction to compute its orthogonal direction in Euclidean space.

OrthoUVDir: UV direction that is orthogonal to UVDir, in Euclidean space.

Returns: UV direction that is orthogonal to UVDir, in Euclidean space. NULL, if error.

Description: Computes a new parametric direction OrthoUVDir that is orthogonal, in Eculidean space, to given parametric UVDir at parametric position UV of Srf. Clearly both Srf(OrthoUVDir) and Srf(UVDir) are in the tangent space.

3.2.704 CagdSrfUnitMaxCoef (cagd2gen.c:2192)

```
CagdSrfStruct *CagdSrfUnitMaxCoef(CagdSrfStruct *Srf)
```

Srf: Surface to normalize in place its coefficients.

Returns: Normalized surface, in place.

Description: Normalize in place the given surface so its maximal coefficient is of unit size. Each axis is treated independently, including W.
See also: CagdCrvUnitMaxCoef,

3.2.705 CagdSrfUpdateLength (cagd1gen.c:3167)

```
CagdSrfStruct *CagdSrfUpdateLength(CagdSrfStruct *Srf,
                                   int NewLength,
                                   CagdSrfDirType Dir)
```

Srf: Surface to update its mesh length.

NewLength: New length to reallocate for the surface.

Dir: Direction to resize the mesh length, U or V.

Returns: Resized surface, in place.

Description: Resize the mesh length of the surface, in place. The new surface is not the same as the original while a minimal effort is invested to keep the surface similar.
See also: CagdCrvUpdateLength,

3.2.706 CagdSrfsAddAdjAttributes (cagd1gen.c:2133)

```
int CagdSrfsAddAdjAttributes(CagdSrfStruct *Srfs,
                             CagdCrvAdjCmpFuncType CrvCmpFuncPtr,
                             CagdRType Eps)
```

Srfs: A list of surfaces to add adjacency attributes.

CrvCmpFuncPtr: A pointer to the function to compare two curves. Can be NULL for a default comparison function.

Eps: A tolerance value, of boundary curve comparison.

Returns: 0, if the surfaces have a non manifold topology. 1/2, if the surfaces have a manifold topology and forms an open (1) / closed (2) solid.

Description: Given a list of surfaces, we will find for each surface it's adjacent surfaces. Each surface can have up to four neighbors surfaces, each of which is adjacent to the UMin/UMax/VMin/VMax boundary curve of the given surface. If the given surfaces have a non 2-manifold topology, then the function returns zero and do not change the input. If the surfaces have a 2=manifold topology, the function adds to each surface an adjacency attribute "SrfAdjInfo" of type CagdSrfAdjInfoStruct.

See also:

3.2.707 CagdSrfsFilterDuplicated (cagd1gen.c:2979)

```
CagdSrfStruct *CagdSrfsFilterDuplicated(CagdSrfStruct *Srfs, CagdRType Eps)
```

Srfs: The surface list.

Eps: Epsilon value for duplicate determination. If positive, similar surfaces, up to Eps will be considered identical and will be purged from the output. If negative, the absolute value of Eps is used in the determination of duplicated surfaces and a "used" attributes will be placed in the duplicated surfaces with a value of zero.

Returns: The list of filtered surfaces.

Description: Detect duplicated surfaces from a list of surface and either mark or delete the duplicated surfaces based on the sign of Eps.

3.2.708 CagdSrfsFilterSingular (cagd_aux.c:1220)

derivatives

Hodograph

```
CagdSrfStruct *CagdSrfsFilterSingular(CagdSrfStruct *Srfs)
```

Srfs: To filter out singular surfaces, in place.

Returns: Filtered list.

Description: Given a list of surfaces, filters out whole singular surfaces, in place:

1. Examine if its derivative with respect to U is identically zero.
2. Examine if its derivative with respect to V is identically zero.

See also: BzrSrfDerive, BspSrfDerive, CagdSrfDeriveScalar, SymbSrfDeriveRational, , CagdSrfDerive, BzrSrfDeriveScalar, BspSrfDeriveScalar, CagdSrfIsSingular,

3.2.709 CagdSrfsFreeAdjAttributes (cagd1gen.c:2363)

```
void CagdSrfsFreeAdjAttributes(CagdSrfStruct *Srfs)
```

Srfs: A list of surfaces.

Returns: void

Description: This function Takes a list of surfaces and deallocate foreach surface its adjacency attribute "SrfAdjInfo" of type CagdSrfAdjInfoStruct if exists.

See also:

3.2.710 CagdSrfsSame (cagd1gen.c:2687)

```
CagdBType CagdSrfsSame(const CagdSrfStruct *Srf1,
                       const CagdSrfStruct *Srf2,
                       CagdRType Eps)
```

Srf1, Srf2: The two surfaces to compare.

Eps: Tolerance of equality.

Returns: TRUE if surfaces are the same, FALSE otherwise.

Description: Compare the two lists of surfaces for similarity.
See also: CagdCtlMeshsSame, BspKnotVectorsSame, CagdCrvsSame, CagdSrfsSame2, , CagdSrfsSame3, CagdSrfsSame4, CagdCrvsSameUptoRigidScl2D, , CagdSrfsSameUptoRigidScl2D, CagdSrfsSameCorners, CagdSrfsSameFuncSpace,

3.2.711 CagdSrfsSame2 (cagd1gen.c:2754)

```
CagdBType CagdSrfsSame2(const CagdSrfStruct *Srf1,
                        const CagdSrfStruct *Srf2,
                        CagdRType Eps,
                        int *Modified)
```

Srf1, Srf2: The two surfaces to compare.

Eps: Tolerance of equality.

Modified: 0 if no surface was refined/degree raised, 1 if Srf1 was refined/degree raised, 2 if Srf2 was refined/degree raised, 3 if both Srf1 and Srf2 were refined/degree raised. This parameter is optional and can be NULL.

Returns: TRUE if surfaces are the same, FALSE otherwise.

Description: Compare the two surfaces for similarity, after bringing them to a common function space, by degree raising and refinement.
See also: CagdCtlMeshsSame, BspKnotVectorsSame, CagdSrfsSame, CagdSrfsSame3, , CagdSrfsSame4, CagdSrfsSameUptoRigidScl2D, CagdSrfsSameUptoRigidScl2D, , CagdSrfsSameCorners,

3.2.712 CagdSrfsSame3 (cagd1gen.c:2810)

```
CagdBType CagdSrfsSame3(const CagdSrfStruct *Srf1,
                        const CagdSrfStruct *Srf2,
                        CagdRType Eps,
                        int *Modified)
```

Srf1, Srf2: The two surfaces to compare.

Eps: Tolerance of equality.

Modified: 0x00 if no surface was refined/degree raised, 0x01 if Srf1 was refined/degree raised, 0x02 if Srf2 was refined/degree raised, 0x03 if both Srf1 and Srf2 were refined/degree raised. 0x10 if Srf2 was reversed in U 0x20 if Srf2 was reversed in V 0x30 if Srf2 was reversed in U and in V 0x80 if Srf2 was flipped in U with V This parameter is optional and can be NULL.

Returns: TRUE if surfaces are the same, FALSE otherwise.

Description: Compare the two surfaces for similarity, after bringing them to a common function space, by degree raising and refinement, and UV reversing.
See also: CagdCtlMeshsSame, BspKnotVectorsSame, CagdSrfsSame, CagdSrfsSame2, , CagdSrfsSame4, CagdSrfsSameUptoRigidScl2D, CagdSrfsSameUptoRigidScl2D, , CagdSrfsSameCorners,

3.2.713 CagdSrfsSame4 (cagd1gen.c:2866)

```
CagdBType CagdSrfsSame4(const CagdSrfStruct *Srf1,
                        const CagdSrfStruct *Srf2,
                        CagdRType Eps,
                        int *Modified)
```

Srf1, Srf2: The two surfaces to compare.

Eps: Tolerance of equality.

Modified: 0x00 if no surface was refined/degree raised, 0x01 if Srf1 was refined/degree raised, 0x02 if Srf2 was refined/degree raised, 0x03 if both Srf1 and Srf2 were refined/degree raised. 0x10 if Srf2 was reversed in U 0x20 if Srf2 was reversed in V 0x30 if Srf2 was reversed in U and in V 0x80 if Srf2 was flipped in U with V This parameter is optional and can be NULL.

Returns: TRUE if surfaces are the same, FALSE otherwise.

Description: Compare the two surfaces for similarity, after bringing them to a common function space, by degree raising and refinement, and UV reversing.
See also: CagdCtlMeshsSame, BspKnotVectorsSame, CagdSrfsSame, CagdSrfsSame2, , CagdSrfsSame3, CagdSrfsSameUptoRigidScl2D, CagdSrfsSameUptoRigidScl2D, , CagdSrfsSameCorners,

3.2.714 CagdSrfsSameCorners (cagd1gen.c:2441)

```
CagdBType CagdSrfsSameCorners(const CagdSrfStruct *Srf1,
                              const CagdSrfStruct *Srf2,
                              CagdRType Eps)
```

Srf1, Srf2: The two surfaces to compare their four corners.

Eps: Tolerance of equality.

Returns: TRUE if surfaces share the same corners, FALSE otherwise.

Description: Compare the four corners of the given two surfaces for similarity.
See also: CagdCtlMeshsSame, BspKnotVectorsSame, CagdCrvsSame, CagdSrfsSame2, CagdCrvsSameUptoRigidScl2D, CagdSrfsSameUptoRigidScl2D, CagdSrfsSame,

3.2.715 CagdSrfsSameFuncSpace (cagd1gen.c:2640)

```
CagdBType CagdSrfsSameFuncSpace(const CagdSrfStruct *Srf1,
                                const CagdSrfStruct *Srf2,
                                CagdRType Eps)
```

Srf1, Srf2: The two surfaces to compare.

Eps: Tolerance of equality.

Returns: TRUE if surfaces are in same function space, FALSE otherwise.

Description: Compare the two surfaces for similarity of their function space.
See also: CagdCtlMeshsSame, BspKnotVectorsSame, CagdCrvsSame, CagdSrfsSame2, , CagdCrvsSameUptoRigidScl2D, CagdSrfsSameUptoRigidScl2D, , CagdSrfsSameCorners, CagdSrfsSame,

3.2.716 CagdSrfsSameUptoRigidScl2D (cagd1gen.c:2401)

```
CagdBType CagdSrfsSameUptoRigidScl2D(const CagdSrfStruct *Srf1,
                                     const CagdSrfStruct *Srf2,
                                     IrtPtType Trans,
                                     CagdRType *Rot,
                                     CagdRType *Scl,
                                     CagdRType Eps)
```

Srf1, Srf2: The two surfaces to compare.

Trans: Translation amount to apply to Srf1 to bring to Srf2 (after rotation/scale).

Rot, Scl: Rotation and scale amounts to apply to Srf1 to bring to Srf2 (before translation). Rot is specified in degrees.

Eps: Tolerance of equality.

Returns: TRUE if surfaces are the same, FALSE otherwise.

Description: Compare the two lists of surfaces for similarity up to rigid motion and scale in the XY plane.
See also: CagdCtlMeshsSame, BspKnotVectorsSame, CagdCrvsSame, CagdCrvsSameUptoRigidScl2D,

3.2.717 CagdSrfsSubdivAtAllC0Discont (cagd_aux.c:2506)

```
CagdSrfStruct *CagdSrfsSubdivAtAllC0Discont(const CagdSrfStruct *Srfs)
```

Srfs: To subdivide at all C^0 discontinuity locations.

Returns: Surfaces that result from the subdivision.

Description: Subdivides the given list of surfaces at all C^0 potential discontinuity locations.
See also: CagdSrfSubdivAtParams, BspKnotAllC1Discont, , CagdSrfsSubdivAtAllC1Discont, BspSrfsSubdivAtAllDetectedLocations, , CagdSrfSubdivAtAllC0Discont,

3.2.718 CagdSrfsSubdivAtAllC1Discont (cagd_aux.c:2562)

```
CagdSrfStruct *CagdSrfsSubdivAtAllC1Discont(const CagdSrfStruct *Srfs)
```

Srfs: To subdivide at all C^1 discontinuity locations.

Returns: Surfaces that result from the subdivision.

Description: Subdivides the given list of surfaces at all C^1 potential discontinuity locations.
See also: CagdSrfSubdivAtParams, BspKnotAllC1Discont, , CagdSrfsSubdivAtAllC0Discont, BspSrfsSubdivAtAllDetectedLocations, , CagdSrfSubdivAtAllC1Discont,

3.2.719 CagdStructOnceCoercePointsTo (cagdcoer.c:621)

coercion

```
VoidPtr CagdStructOnceCoercePointsTo(CagdRType * const *OldPoints,
                                     const VoidPtr OldStruct,
                                     int OldStructLen,
                                     int ExtraMem,
                                     int PtsLen,
                                     CagdPointType OldPType,
                                     CagdPointType NewPType)
```

OldPoints: Where the old points in OldStruct are placed.

OldStruct: A pointer to the original structure hold Points.

OldStructLen: Sizeof OldStruct structure.

ExtraMem: Do we seek to allocate extra memory at the end?

PtsLen: Length of vectors in the array of vectors, Points.

OldPType: Point type to be expected from Points.

NewPType: Point type of the coerced new point.

Returns: A duplicated parent structure with new point types.

Description: Coerces an array of vectors of points of point type OldPType to point type NewPType, while duplicating the parent's structure.

3.2.720 CagdSurfaceRev (cagdsrev.c:47)

surface of revolution

surface constructors

```
CagdSrfStruct *CagdSurfaceRev(const CagdCrvStruct *CCrv)
```

CCrv: To create surface of revolution around Z with.

Returns: Surface of revolution.

Description: Constructs a surface of revolution around the Z axis of the given profile curve. Resulting surface will be a Bspline surface, while input may be either a Bspline or a Bezier curve.
See also: CagdSurfaceRev2, CagdSurfaceRevAxis, CagdSurfaceRev2Axis, , CagdSurfaceRevPolynomialApprox,

3.2.721 CagdSurfaceRev2 (cagdsrev.c:218)

surface of revolution

surface constructors

```
CagdSrfStruct *CagdSurfaceRev2(const CagdCrvStruct *Crv,
                               CagdBType PolyApprox,
                               CagdRType StartAngle,
                               CagdRType EndAngle)
```

Crv: To create surface of revolution around Z with.

PolyApprox: TRUE for a polynomial approximation, FALSE for a precise rational construction.

StartAngle: Starting Angle to consider rotating Crv from, in degrees.

EndAngle: Terminating Angle to consider rotating Crv from, in degrees.

Returns: Surface of revolution.

Description: Constructs a surface of revolution around the Z axis of the given profile curve from StartAngle to EndAngle. Resulting surface will be a Bspline surface, while input may be either a Bspline or a Bezier curve.
See also: CagdSurfaceRev, CagdSurfaceRevAxis, CagdSurfaceRev2Axis, , CagdSurfaceRevPolynomialApprox,

3.2.722 CagdSurfaceRev2Axis (cagdsrev.c:301)

surface of revolution

surface constructors

```
CagdSrfStruct *CagdSurfaceRev2Axis(const CagdCrvStruct *Crv,
                                   CagdBType PolyApprox,
                                   CagdRType StartAngle,
                                   CagdRType EndAngle,
                                   const CagdVType Axis)
```

Crv: To create surface of revolution around Axis.

PolyApprox: TRUE for a polynomial approximation, FALSE for a precise rational construction.

StartAngle: Starting Angle to consider rotating Crv from, in degrees.

EndAngle: Terminating Angle to consider rotating Crv from, in degrees.

Axis: Of rotation of Crv. This axis is always through the origin.

Returns: Surface of revolution.

Description: Constructs a surface of revolution around vector Axis of the given profile curve from StartAngle to EndAngle. Resulting surface will be a Bspline surface, while input may be either a Bspline or a Bezier curve.
See also: CagdSurfaceRev, CagdSurfaceRev2, CagdSurfaceRev2Axis, , CagdSurfaceRevPolynomialApprox,

3.2.723 CagdSurfaceRevAxis (cagdsrev.c:166)

surface of revolution

surface constructors

```
CagdSrfStruct *CagdSurfaceRevAxis(const CagdCrvStruct *Crv, CagdVType Axis)
```

Crv: To create surface of revolution around Axis.

Axis: Of rotation of Crv. This axis is always through the origin.

Returns: Surface of revolution.

Description: Constructs a surface of revolution around vector Axis of the given profile curve. Resulting surface will be a B-spline surface, while input may be either a B-spline or a Bezier curve.
See also: CagdSurfaceRev, CagdSurfaceRev2, CagdSurfaceRev2Axis, , CagdSurfaceRevPolynomialApprox,

3.2.724 CagdSurfaceRevPolynomialApprox (cagdsrev.c:350)

surface of revolution

surface constructors

```
CagdSrfStruct *CagdSurfaceRevPolynomialApprox(const CagdCrvStruct *Crv)
```

Crv: To approximate a surface of revolution around Z with. Crv is assumed planar in a plane holding the Z axis.

Returns: Surface of revolution approximation.

Description: Constructs a surface of revolution around the Z axis of the given profile curve. Resulting surface will be a Bspline surface, while input may be either a Bspline or a Bezier curve. Resulting surface will be a polynomial Bspline surface, approximating a surface of revolution using a polynomial circle approx. (See Faux & Pratt "Computational Geometry for Design and Manufacturing").

See also: CagdSurfaceRev, CagdSurfaceRev2, CagdSurfaceRevAxis, CagdSurfaceRev2Axis,

3.2.725 CagdSweepAxisRefine (cagdswep.c:827)

sweep

refinement

```
CagdCrvStruct *CagdSweepAxisRefine(const CagdCrvStruct *Axis,
                                   const CagdCrvStruct *ScalingCrv,
                                   int RefLevel)
```

Axis: Axis to be used in future sweep operation with the associated ScalingCrv.

ScalingCrv: If sweep is to have one, NULL otherwise.

RefLevel: Some refinement control. Keep it low like 2 or 3.

Returns: Refined Axis curve.

Description: Routine to refine the axis curve, according to the scaling curve to better approximate the requested sweep operation.

See also: CagdSweepSrfError, CagdSweepSrf,

3.2.726 CagdSweepComputeNormalOrientation (cagdswep.c:429)

orientation frame

sweep

```
CagdBType CagdSweepComputeNormalOrientation(const CagdCrvStruct *Axis,
                                            const CagdVType FrameVec,
                                            CagdCrvStruct *FrameCrv,
                                            CagdRType *PrevT,
                                            CagdRType CrntT,
                                            const CagdVecStruct *Tangent,
                                            CagdVecStruct *Normal,
                                            CagdBType FirstTime)
```

Axis: Axis curve of sweep.

FrameVec: Binormal constant of orientation frame, if not NULL.

FrameCrv: Binormal vector field of orientation frame, if not NULL.

PrevT: Previous parameter value where to evaluate.

CrntT: Parameter value where to evaluate.

Tangent: of Axis curve at parameter value t.

Normal: An estimated normal for the orientation frame.

FirstTime: TRUE if first time to compute the normal.

Returns: TRUE if computed, FALSE otherwise.

Description: Estimates a normal for the orientation frame at the given parameter t.

3.2.727 CagdSweepCosineHalfAngle (cagdswep.c:659)

orientation frame
sweep
surface constructors

```
CagdRType CagdSweepCosineHalfAngle(CagdRType **Points, int Index)
```

Points: Points to consider.

Index: at indices Index-1, Index, and Index+1.

Returns: Cosine of half the angle between the given three points

Description: Computes the cosine of half the angle between the two vectors between the three consecutive control points Index-1, Index, and Index+1.

3.2.728 CagdSweepGenTransformMatrix (cagdswep.c:782)

orientation frame
sweep
surface constructors

```
void CagdSweepGenTransformMatrix(CagdMType Mat,
                                 const CagdRType *Trans,
                                 const CagdVecStruct *Normal,
                                 const CagdVecStruct *Tangent,
                                 const CagdRType *Scale,
                                 CagdRType NormalScale)
```

Mat: To place the newly computed transformation.

Trans: Translation factor.

Normal: Normal direction to prescribe orientation.

Tangent: Tangent direction to prescribe orientation.

Scale: Scale factor in X&Y. If the Y scaling factor is zero, the X scale factor is also used for the Y axis. Can be NULL of no scale.

NormalScale: Scale factor in the normal vector direction.

Returns: void

Description: Routine to prepare a transformation matrix to do the following (in this order): scale by Scale, rotate such that X axis is in Normal dir and Y is collinear with the BiNormal and then translate by Trans. Algorithm: given the Trans vector, it forms the 4th line of Mat. Dir is used to form the second line (the first 3 lines set the rotation), and finally Scale is used to scale first 3 lines/columns to the needed scale: | Nx Ny Nz 0 | A transformation which takes the coord | Bx By Bz 0 | system into T, N & B as required and [X Y Z 1] * | Tx Ty Tz 0 | then translate it to C. T, N, B are | Cx Cy Cz 1 | scaled by Scale. T is exactly Tangent (unit vec). N is set to be Normal and B their cross product. All argument vectors are assumed to be normalized to a unit length.
See also: CagdSweepSrfError, CagdSweepSrf,

3.2.729 CagdSweepSrf (cagdswep.c:103)

sweep
surface constructors

```
CagdSrfStruct *CagdSweepSrf(const CagdCrvStruct *CrossSection,
                            const CagdCrvStruct *Axis,
                            const CagdCrvStruct *ScalingCrv,
                            CagdRType Scale,
                            const VoidPtr Frame,
                            int FrameOption,
                            CagdRType *LastBiNormal)
```

CrossSection: Of the constructed sweep surface. If more than one curve is given as a linked list of curves, the cross sections are modified as we progresses along the sweep, blending between the cross sections so that last cross section is used in the last parameter value of the Axis.

Axis: Of the constructed sweep surface.

ScalingCrv: Optional scale or profile curve. NULL if none. The Y axis of this curve is using as scaling as the function of the parametrization. X axis can be used to define a 2D parametric curve so this curve can be displayed but X is ignored by this function.

Scale: If no Scaling Crv, Scale is used to apply a fixed scale on the CrossSection curve.

Frame: An optional vector or a curve to specify the binormal orientation. Otherwise, Frame must be NULL.

FrameOption: If 2, Frame is a curve, and if -2, -1, 0, 1 a vector is used to control the frame, as follows, If 0, Frame vector is used just to init the first frame. If -1, use Frame vector to init first and last frame. If 1, use Frame vector for all frames. If -2, Frame holds two vectors (6 real values) to init both first and last frame.

LastBiNormal: Optional, if not NULL, vector to return the last normal used in the frame.

Returns: Constructed sweep surface.

Description: Constructs a sweep surface using the following curves:

1. CrossSection - defines the basic cross section of the sweep. Must be in the XY plane. Can be several curves to be blended along the Axis.
2. Axis - a 3D curve the CrossSection will be swept along such that the Axis normal aligns with the Y axis of the cross section. If Axis is linear (i.e. no normal), the normal is picked randomly or to fit the non linear part of the Axis (if any).
3. Scale - a scaling curve for the sweep, If NULL a scale of Scale is used.
4. Frame - a curve or a vector that specifies the orientation of the sweep by specifying the axes curve's binormal. If Frame is a vector, it is a constant binormal. If Frame is a curve (FrameOption == 2), it is assumed to be a vector field binormal. If NULL, it is computed from the Axis curve's pseudo Frenet frame, that minimizes rotation.

This operation is only an approximation. See CagdSweepAxisRefine for a tool to refine the Axis curve and improve accuracy.

See also: CagdSweepSrfError, CagdSweepAxisRefine, TrivSweepTV,

3.2.730 CagdSweepSrfC1 (cagdswep.c:1028)

sweep

surface constructors

```
CagdSrfStruct *CagdSweepSrfC1(const CagdCrvStruct *CrossSection,
                              const CagdCrvStruct *Axis,
                              const CagdCrvStruct *ScalingCrv,
                              CagdRType Scale,
                              const VoidPtr Frame,
                              int FrameOption,
                              CagdCrvCornerType CornerType,
                              CagdRType C1DiscontCropTol)
```

CrossSection: Of the constructed sweep surface. If more than one curve is given as a linked list of curves, the cross sections are modified as we progresses along the sweep, blending between the cross sections so that last cross section is used in the last parameter value of the Axis.

Axis: Of the constructed sweep surface. Can be C1 discont.

ScalingCrv: Optional scale or profile curve. NULL if none. The Y axis of this curve is using as scaling as the function of the parametrization. X axis can be used to define a 2D parametric curve so this curve can be displayed but X is ignored by this function.

Scale: If no Scaling Crv, Scale is used to apply a fixed scale on the CrossSection curve.

Frame: An optional vector or a curve to specified the binormal orientation. Otherwise Frame must be NULL.

FrameOption: If 2 Frame is a curve, if 0 or 1 a vector (if Frame is not NULL). If 0, the vector is used just to init the first frame.

CornerType: C^1 discont. joint corner type - 1. For splitting and computing sweeps of individual C^1 cont. axes. 2. For rounded joints (circular if input is rational). 3. For chamfered joints. 4. For mitered joints.

C1DiscontCropTol: Additional epsilon to crop surfaces at intersection.

Returns: Constructed sweep surface.

Description: Constructs a sweep surface using the following curves:

1. CrossSection - defines the basic cross section of the sweep. Must be in the XY plane. Can be several curves to be blended along the Axis.
2. Axis - a 3D curve the CrossSection will be swept along such that the Axis normal aligns with the Y axis of the cross section. If Axis is linear (i.e. no normal), the normal is picked randomly or to fit the non linear part of the Axis (if any). Can be C^1 discontinuous.
3. Scale - a scaling curve for the sweep, If NULL a scale of Scale is used.
4. Frame - a curve or a vector that specifies the orientation of the sweep by specifying the axes curve's binormal. If Frame is a vector, it is a constant binormal. If Frame is a curve (FrameOption == 2), it is assumed to be a vector field binormal. If NULL, it is computed from the Axis curve's pseudo Frenet frame, that minimizes rotation.

This operation is only an approximation. See CagdSweepAxisRefine for a tool to refine the Axis curve and improve accuracy.

See also: CagdSweepSrfError, CagdSweepAxisRefine, CagdSweepSrf,

3.2.731 CagdSweepSrfC1AdjSrfsInterDmn (cagdswep.c:1203)

```
CagdBType CagdSweepSrfC1AdjSrfsInterDmn(const CagdSrfStruct *PrevSrf,
                                        const CagdSrfStruct *NextSrf,
                                        CagdRType C1DiscontCropTol,
                                        CagdRType *PrevSrfVMax,
                                        CagdRType *NextSrfVMin)
```

PrevSrf: Previous surface before the C^1 discontinuity.

NextSrf: Next surface after the C^1 discontinuity.

C1DiscontCropTol: Additional epsilon to crop surfaces at intersection.

PrevSrfVMax: VMax to clip PrevSrf at, if the two srfs intersect.

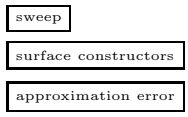
NextSrfVMin: VMin to clip NextSrf at, if the two srfs intersect.

Returns: TRUE if surfaces intersects, FALSE otherwise.

Description: Check if two adjacent surfaces at a C^1 discontinuity intersect. If they do intersect, returns their (V, the sweep parametric direction) domain of intersection.

See also: CagdSweepSrfC1,

3.2.732 CagdSweepSrfError (cagdswep.c:1498)




```
CagdSrfStruct *CagdSweepSrfError(const CagdSrfStruct *SweepSrf,
                                 const CagdCrvStruct *CrossSection,
                                 const CagdCrvStruct *Axis,
                                 const CagdCrvStruct *ScalingCrv,
                                 CagdRType Scale)
```

SweepSrf: A computed approximated sweep surface, given the params.

CrossSection: Of the constructed sweep surface. If more than one curve is given as a linked list of curves, the cross sections are modified as we progresses along the sweep, blending between the cross sections so that last cross section is used in the last parameter value of the Axis.

Axis: Of the constructed sweep surface.

ScalingCrv: Optional scale or profile curve. NULL if none. The Y axis of this curve is using as scaling as the function of the parametrization. X axis can be used to define a 2D parametric curve so this curve can be displayed but X is ignored by this function.

Scale: If no Scaling Crv, Scale is used to apply a fixed scale on the CrossSection curve.

Returns: A scalar field represented the error as L2 distance square from the precise sweep surface.

Description: Computes an upper bound on the error as a scalar function over the sweep surface approximation, SweepSrf, as a field over the domain of SweepSrf, given the sweep surface construction parameters. This operation is only an approximation. See CagdSweepAxisRefine for a tool to refine the Axis curve and improve accuracy. Error is computed as:

```
SweepErr(u, v) = || CrossSection(u) * Scale(v) || ^ 2  -
                 || SweepSrf(u, v) - Axis(v) || ^ 2
```

where scale(v) is constant Scale if no scaling crv.

See also: CagdSweepSrf, CagdSweepAxisRefine,

3.2.733 CagdTransform (cagd2gen.c:1582)

scaling
translation
transformations

```
void CagdTransform(CagdRType **Points,
                   int Len,
                   int MaxCoord,
                   CagdBType IsNotRational,
                   const CagdRType *Translate,
                   CagdRType Scale)
```

Points: To be affinely transformed. Array of vectors.

Len: Of vectors of Points.

MaxCoord: Maximum number of coordinates to be found in Points. At most 3 for R^3.

IsNotRational: Do we have weights as vector Points[0]?

Translate: Translation amount, NULL for non.

Scale: Scaling amount.

Returns: void

Description: Applies an affine transform, in place, to given set of points Points which as array of vectors, each vector of length Len. Array Points optionally contains (if !IsNotRational) in Points[0] the weights coefficients and in Points[i] the coefficients of axis i, up to and include MaxCoord (X = 1, Y = 2, etc.). Points are translated and scaled as prescribed by Translate and Scale. Each control point is first translated by Translate and then scaled by Scale.
See also: CagdSrfTransform, CagdCrvTransform, CagdTransform2,

3.2.734 CagdTransform2 (cagd2gen.c:1623)

scaling
translation
transformations

```
void CagdTransform2(CagdRType **Points,
                    int Len,
                    int MaxCoord,
                    CagdBType IsNotRational,
                    const CagdRType *Translate,
                    CagdRType Scale)
```

Points: To be affinely transformed. Array of vectors.

Len: Of vectors of Points.

MaxCoord: Maximum number of coordinates to be found in Points.

IsNotRational: Do we have weights as vector Points[0]?

Translate: Translation amount, NULL for non.

Scale: Scaling amount.

Returns: void

Description: Applies an affine transform, in place, to given set of points Points which as array of vectors, each vector of length Len. Array Points optionally contains (if !IsNotRational) in Points[0] the weights coefficients and in Points[i] the coefficients of axis i, up to and include MaxCoord (X = 1, Y = 2, etc.). Points are translated and scaled as prescribed by Translate and Scale. Each control point is first translated by Translate and then scaled by Scale.
See also: CagdSrfTransform, CagdCrvTransform, CagdTransform,

3.2.735 CagdUVArrayFree (cagd2gen.c:385)

free

```
void CagdUVArrayFree(CagdUVStruct *UVArray, int Size)
```

UVArray: To be deallocated.

Size: Of the deallocated array.

Returns: void

Description: Deallocates and frees an array of UV structure.

3.2.736 CagdUVArrayNew (cagd1gen.c:257)

allocation

```
CagdUVStruct *CagdUVArrayNew(int Size)
```

Size: Size of UV array to allocate.

Returns: An array of UV structures of size Size.

Description: Allocates and resets all slots of an array of UV structures.

3.2.737 CagdUVCopy (cagd1gen.c:923)

copy

```
CagdUVStruct *CagdUVCopy(const CagdUVStruct *UV)
```

UV: To be copied.

Returns: A duplicate of UV.

Description: Allocates and copies all slots of a UV structure.

3.2.738 CagdUVCopyList (cagd1gen.c:1266)

copy

```
CagdUVStruct *CagdUVCopyList(const CagdUVStruct *UVList)
```

UVList: To be copied.

Returns: A duplicated list of UV's.

Description: Allocates and copies a list of UV structures.

3.2.739 CagdUVFree (cagd2gen.c:337)

free

```
void CagdUVFree(CagdUVStruct *UV)
```

UV: To be deallocated.

Returns: void

Description: Deallocates and frees all slots of a UV structure.

3.2.740 CagdUVFreeList (cagd2gen.c:360)

free

```
void CagdUVFreeList(CagdUVStruct *UVList)
```

UVList: To be deallocated.

Returns: void

Description: Deallocates and frees a UV structure list:

3.2.741 CagdUVNew (cagd1gen.c:284)

allocation

```
CagdUVStruct *CagdUVNew(void)
```

Returns: A UV structure.

Description: Allocates and resets all slots of a UV structure.

3.2.742 CagdUpdateBndryCrvInSrf (cagdbsum.c:608)

```
int CagdUpdateBndryCrvInSrf(CagdSrfStruct *Srf,
                            const CagdCrvStruct *Crv,
                            CagdSrfBndryType SrfBndry)
```

Srf: Surface to update one of its boundaries.

Crv: Curve information to update with the selected boundary of Srf.

SrfBndry: The boundary in Srf to update with Crv.

Returns: TRUE, if successful, FALSE otherwise.

Description: Update one boundary of surface Srf using the given curve Crv.
See also: ,

3.2.743 CagdVecArrayFree (cagd2gen.c:645)

free

```
void CagdVecArrayFree(CagdVecStruct *VecArray, int Size)
```

VecArray: To be deallocated.

Size: Of the deallocated array.

Returns: void

Description: Deallocates and frees an array of vector structure.

3.2.744 CagdVecArrayNew (cagd1gen.c:438)

allocation

```
CagdVecStruct *CagdVecArrayNew(int Size)
```

Size: Size of Vec array to allocate.

Returns: An array of Vec structures of size Size.

Description: Allocates and resets all slots of an array of Vec structures.

3.2.745 CagdVecCopy (cagd1gen.c:1024)

copy

```
CagdVecStruct *CagdVecCopy(const CagdVecStruct *Vec)
```

Vec: To be copied.

Returns: A duplicate of Vec.

Description: Allocates and copies all slots of a Vec structure.

3.2.746 CagdVecCopyList (cagd2gen.c:70)

copy

```
CagdVecStruct *CagdVecCopyList(const CagdVecStruct *VecList)
```

VecList: To be copied.

Returns: A duplicated list of vectors.

Description: Allocates and copies a list of vector structures.

3.2.747 CagdVecFree (cagd2gen.c:597)

free

```
void CagdVecFree(CagdVecStruct *Vec)
```

Vec: To be deallocated.

Returns: void

Description: Deallocates and frees all slots of a vector structure.

3.2.748 CagdVecFreeList (cagd2gen.c:620)

free

```
void CagdVecFreeList(CagdVecStruct *VecList)
```

VecList: To be deallocated.

Returns: void

Description: Deallocates and frees a vector structure list:

3.2.749 CagdVecNew (cagd1gen.c:465)

allocation

```
CagdVecStruct *CagdVecNew(void)
```

Returns: A Vec structure.

Description: Allocates and resets all slots of a Vec structure.

3.2.750 CagdZTwistExtrudeSrf (cagdextr.c:165)

surface constructors

```
CagdSrfStruct *CagdZTwistExtrudeSrf(const CagdCrvStruct *CCrv,
                                    CagdBType Rational,
                                    CagdRType ZPitch)
```

CCrv: To twist and extrude in the +Z direction.

Rational: TRUE to construct a rational (and precise) twist, FALSE to approximate using polynomials.

ZPitch: The +Z amount for full 360 degrees. If zero, the result will be a planar (degenerated) surface. A negative value will reverse the twist.

Returns: A twisted extrusion surface.

Description: Constructs a full circular twisted/rotated extrusion surface in the +Z direction for the given profile curve. Input curve can be either a B-spline or a Bezier curve.
See also: CagdExtrudeSrf,

3.2.751 Energy1Calc (cbsp_fit.c:1865)

RegTermCalculatorFuncType

```
static CagdRType Energy1Calc(CagdCrvStruct *Crv)
```

Crv: A curve to derive and integrate.

Returns: The calculated integral value.

Description: Calculates the curve first derivative energy integral:

$$\int \| Crv'(t) \|^2 \, dt$$

3.2.752 Energy1MatrixCalc (cbsp_fit.c:1917)

RegMatrixCalculatorFuncType

```
static void Energy1MatrixCalc(CagdCrvStruct *Crv,
                              CagdRType *A,
                              CagdRType *b,
                              CagdRType Lambda)
```

Crv: Input b-spline curve.

A: Input/output initialized matrix, which size = 2 * Crv -> Length x 2 * Crv -> Length.

b: Input/output initialized offset vector, which size = 2 * Crv -> Length x 1.

Lambda: Weight.

Returns: void

Description: Calculates the curve first derivative energy (see Energy1Calc) minimization matrix. The calculated coefficients are ADDED to A and b.

3.2.753 Energy2Calc (cbsp_fit.c:1672)

RegTermCalculatorFuncType

```
static CagdRType Energy2Calc(CagdCrvStruct *Crv)
```

Crv: A curve to derive and integrate.

Returns: The calculated integral value.

Description: Calculates the curve second derivative energy integral:

```
/             2
| ||Crv''(t)||  dt
/
```

3.2.754 Energy2MatrixCalc (cbsp_fit.c:1722)

RegMatrixCalculatorFuncType

```
static void Energy2MatrixCalc(CagdCrvStruct *Crv,
                              CagdRType *A,
                              CagdRType *b,
                              CagdRType Lambda)
```

Crv: Input b-spline curve.

A: Input/output initialized matrix, which size = 2 * Crv -> Length x 2 * Crv -> Length.

b: Input/output initialized offset vector, which size = 2 * Crv -> Length x 1.

Lambda: Weight.

Returns: void

Description: Calculates the curve second derivative energy (see Energy2Calc) minimization matrix. The calculated coefficients are ADDED to A and b.

3.2.755 IritCagdDescribeError (cagd_err.c:114)

error handling

```
const char *IritCagdDescribeError(IritCagdFatalErrorType ErrorNum)
```

ErrorNum: Type of the error that was raised.

Returns: A string describing the error type.

Description: Returns a string describing a the given error. Errors can be raised by any member of this cagd library as well as other users. Raised error will cause an invocation of IritCagdFatalError function which decides how to handle this error. IritCagdFatalError can for example, invoke this routine with the error type, print the appropriate message and quit the program.

3.2.756 IritCagdFatalError (cagd_ftl.c:56)

error handling

```
void IritCagdFatalError(IritCagdFatalErrorType ErrID)
```

ErrID: Error type that was raised.

Returns: void

Description: Trap Cagd_lib errors right here. Provides a default error handler for the cagd library. Gets an error description using IritCagdDescribeError, prints it and exit the program using exit.

3.2.757 IritCagdSetFatalErrorFunc (cagd_ftl.c:28)

error handling

```
CagdSetErrorFuncType IritCagdSetFatalErrorFunc(CagdSetErrorFuncType ErrorFunc)
```

ErrorFunc: New error function to use.

Returns: Old error function reference.

Description: Sets the error function to be used by Cagd_lib.

3.2.758 LeastSquareInitCrvCalculator (cbsp_fit.c:1459)

InitFittingCrvCalculatorFuncType

```
static CagdCrvStruct *LeastSquareInitCrvCalculator(CagdPType *PtList,
                                                   int NumOfPoints,
                                                   int Length,
                                                   int Order,
                                                   CagdBType Periodic)
```

PtList: Points cloud.

NumOfPoints: Number of points in PtList.

Length: The desired length of the output b-spline curve.

Order: The desired order of the output b-spline curve.

Periodic: TRUE for periodic output curve, FALSE for open end.

Returns: The calculated b-spline curve.

Description: Computes an initial b-spline fitting curve that least square approximates the input points.

3.2.759 PDMErrorCalc (cbsp_fit.c:533)

```
static CagdRType PDMErrorCalc(int NumOfPoints,
                              CagdPType *Points,
                              CagdPType *FootPoints)
```

NumOfPoints: Number of points in the points cloud.

Points: (X) Points cloud. Array of points with size = NumOfPoints.

FootPoints: (P(t)) Footpoints (the closest points on the curve) array. Array size must be 'NumOfPoints'.

Returns: PD error.

Description: Computes PD Minimization method error, which is:

```
 NumOfPoints
    ----
 1  \
 -  /     e
 2  ----   PD,k
     k=1
                          2
 e      =  ||P(tk)-Xk||
  PD,k
```

See also: CagdSDError,

3.2.760 PDMatrixCalc (cbsp_fit.c:1617)

PD error

```
static void PDMatrixCalc(int Length,
                         int NumOfPoints,
                         CagdPType *Points,
                         CagdRType *Basis,
                         CagdPType *FootPoints,
                         CagdRType *A, /* MATRIX */
                         CagdRType *b)
```

Length: Fitting curve Length.

NumOfPoints: Number of points in the points cloud.

Points: Points cloud. Array of points with size = NumOfPoints.

Basis: Array of size (NumOfPoints * Length) containing basis function coefficients at the foot points

FootPoints: Footpoints (the closest points on the curve) array of size NumOfPoints.

A: Input/output matrix (2Length * 2Length).

b: Input/output offset vector (2Length * 1).

Returns: void

Description: Calculates the PD error minimization equation Matrix and offset vector, i.e. A and b of the Ax=b equation. The caluclated values are ADDED to the A and b parameters The order of variables in x is assumed to be: (D1_x,D2_x,...,DLength_x,D1_y,D2_y,...,DLength_y)

3.2.761 PwrCrvDegreeRaise (cpwr_aux.c:286)

degree raising

```
CagdCrvStruct *PwrCrvDegreeRaise(const CagdCrvStruct *Crv)
```

Crv: To raise its degree by one.

Returns: A curve of one order higher representing the same geometry as Crv.

Description: Returns a new curve, identical to the original but with one degree higher. Adds one more, highest degree coefficient, that is identically zero.
See also: BzrCrvDegreeRaiseN, BzrCrvDegreeRaise, PwrCrvDegreeRaiseN,

3.2.762 PwrCrvDegreeRaiseN (cpwr_aux.c:239)

degree raising

```
CagdCrvStruct *PwrCrvDegreeRaiseN(const CagdCrvStruct *Crv, int NewOrder)
```

Crv: To raise its degree to a NewOrder.

NewOrder: NewOrder for Crv.

Returns: A curve of order NewOrder representing the same geometry as Crv.

Description: Returns a new curve, identical to the original but with order NewOrder. Degree raise is computed by adding zeros at high order coefs.
See also: BzrCrvDegreeRaise, BzrCrvDegreeRaiseN, PwrCrvDegreeRaise,

3.2.763 PwrCrvDerive (cpwr_aux.c:116)

derivatives

```
CagdCrvStruct *PwrCrvDerive(const CagdCrvStruct *Crv, CagdBType DeriveScalar)
```

Crv: To differentiate.

DeriveScalar: TRUE to differentiate each channel independently. FALSE to treat geometry as a rational differentiation.

Returns: Differentiated curve.

Description: Returns a new curve, equal to the given curve, differentiated once.
See also: CagdCrvDerive, BspCrvDerive, SymbCrvDeriveRational,

3.2.764 PwrCrvDeriveScalar (cpwr_aux.c:170)

derivatives

```
CagdCrvStruct *PwrCrvDeriveScalar(const CagdCrvStruct *Crv)
```

Crv: To differentiate.

Returns: Differentiated curve.

Description: Returns a new curve, equal to the given curve, differentiated once. For a Euclidean curve this is the same as CagdCrvDerive but for a rational curve the returned curve is not the vector field but simply the derivatives of all the curve's coefficients, including the weights.
See also: PwrCrvDerive, CagdCrvDerive, SymbCrvDeriveRational, , BspCrvDerive, PwrCrvDeriveScalar, CagdCrvDeriveScalar,

3.2.765 PwrCrvEvalAtParamMalloc (cpwr_aux.c:41)

evaluation

```
CagdRType *PwrCrvEvalAtParamMalloc(const CagdCrvStruct *Crv, CagdRType t)
```

Crv: To evaluate at the given parametric location t.

t: The parameter value at which the curve Crv is to be evaluated.

Returns: A vector holding all the coefficients of all components of curve Crv's point type. If for example the curve's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Description: Returns a pointer to a dynamically allocated data, holding the value of the curve at given parametric location t. The curve is assumed to be a power basis. Evaluation is conducted using the Horner rule.
See also: CagdCrvEvalToData, BspCrvEvalAtParamToData, BzrCrvEvalAtParamToData, , PwrCrvEvalAtParamToData, BspCrvEvalAtParamMalloc,

3.2.766 PwrCrvEvalAtParamToData (cpwr_aux.c:80)

evaluation

```
void PwrCrvEvalAtParamToData(const CagdCrvStruct *Crv,
                             CagdRType t,
                             CagdRType *Buf)
```

Crv: To evaluate at the given parametric location t.

t: The parameter value at which the curve Crv is to be evaluated.

Buf: A vector holding all the coefficients of all components of curve Crv's point type. If for example the curve's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Returns: void

Description: Updates the given data (Buf) to hold the value of the curve at given parametric location t. The curve is assumed to be a power basis. Evaluation is conducted using the Horner rule.
See also: CagdCrvEvalToData, BspCrvEvalAtParamToData, BzrCrvEvalAtParamToData, , PwrCrvEvalAtParamToData,

3.2.767 PwrCrvIntegrate (cpwr_aux.c:195)

integrals

```
CagdCrvStruct *PwrCrvIntegrate(const CagdCrvStruct *Crv)
```

Crv: Curve to integrate.

Returns: Integrated curve.

Description: Returns a new Bezier curve, equal to the integral of the given power
See also: BspCrvIntegrate, BzrSrfIntegrate, CagdCrvIntegrate,

3.2.768 PwrCrvNew (bzr_gen.c:128)

allocation

```
CagdCrvStruct *PwrCrvNew(int Length, CagdPointType PType)
```

Length: Number of control points

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform Power basis curve.

Description: Allocates the memory required for a new Power basis curve.
See also: BspCrvNew, BspPeriodicCrvNew, CagdCrvNew, CagdPeriodicCrvNew, TrimCrvNew, BzrCrvNew,

3.2.769 PwrSrfDegreeRaise (sbzr_aux.c:306)

degree raising

```
CagdSrfStruct *PwrSrfDegreeRaise(const CagdSrfStruct *Srf, CagdSrfDirType Dir)
```

Srf: To raise it degree by one.

Dir: Direction to degree raise. Either U or V.

Returns: A surface with one degree higher in direction Dir, representing the same geometry as Srf.

Description: Returns a new power basis surface, identical to the original but with one degree higher, in the requested direction Dir.
See also: CagdSrfDegreeRaise, BzrSrfDegreeRaise, TrimSrfDegreeRaise, , BspSrfDegreeRaise, BzrSrfDegreeRaiseN, CagdSrfDegreeRaiseN, , PwrSrfDegreeRaiseN,

3.2.770 PwrSrfDegreeRaiseN (sbzr_aux.c:345)

degree raising

```
CagdSrfStruct *PwrSrfDegreeRaiseN(const CagdSrfStruct *Srf,
                                  int NewUOrder,
                                  int NewVOrder)
```

Srf: To raise its degrees.

NewUOrder: New U order of Srf.

NewVOrder: New V order of Srf.

Returns: A surface with higher degrees as prescribed by NewUOrder/NewVOrder.

Description: Returns a new power basis surface, identical to the original but with higher degrees, as prescribed by NewUOrder, NewVOrder.
See also: CagdSrfDegreeRaise, BzrSrfDegreeRaise, TrimSrfDegreeRaise, , BspSrfDegreeRaise, BzrSrfDegreeRaiseN, CagdSrfDegreeRaiseN, , PwrSrfDegreeRaise,

3.2.771 PwrSrfNew (bzr_gen.c:97)

allocation

```
CagdSrfStruct *PwrSrfNew(int ULength, int VLength, CagdPointType PType)
```

ULength: Number of control points in the U direction.

VLength: Number of control points in the V direction.

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform Power basis surface.

Description: Allocates the memory required for a new Power basis surface.
See also: BspSrfNew, BspPeriodicSrfNew, CagdSrfNew, CagdPeriodicSrfNew, TrimSrfNew, BzrSrfNew,

3.2.772 SDMErrorCalc (cbsp_fit.c:592)

```
static CagdRType SDMErrorCalc(int NumOfPoints,
                              CagdPType *Points,
                              CagdPType *FootPoints,
                              CagdRType *Distances,
                              CagdPType *Tangents,
                              CagdPType *Normals,
                              CagdRType *Curvatures,
                              CagdBType *IsOuter)
```

NumOfPoints: Number of points in the points cloud.

Points: (X) Points cloud. Array of points with size = NumOfPoints.

FootPoints: (P(t)) Footpoints (the closest points on the curve) array.

Distances: (d) Array of distances between each point to the corresponding footpoint.

Tangents: (T) Array of curve tangents at each footpoint.

Normals: (N) Array of curve normals at each footpoint.

Curvatures: (p) Array of curve curvature radiuses at each footpoint.

IsOuter: Array of booleans that indicates outer points.

Returns: SD error.

Description: Computes SD Minimization method error, which is:

```
 NumOfPoints
    ----
1   \
-   /     e
2   ----   SD,k
     k=1
             /   dk              T     2               T       2
             |  -----[(P(tk)-Xk) * Tk]  +  [(P(tk)-Xk)  *  Nk]  ,if dk<0
e      =   <   dk-pk
 SD,k        |                T     2
             |  [(P(tk) - Xk] * Ni]                          ,if 0=<dk<pk
             \
```

NOTE: The size of each array must be 'NumOfPoints'.

See also: CagdSDError,

3.2.773 SDMatrixCalc (cbsp_fit.c:320)

SD error

```
static void SDMatrixCalc(int Length,
                         int NumOfPoints,
                         CagdPType *Points,
                         CagdRType *Basis,
                         CagdPType *FootPoints,
                         CagdRType *Distances,
                         CagdPType *Tangents,
                         CagdPType *Normals,
                         CagdRType *Curvatures,
                         CagdBType *IsOuter,
                         CagdRType *A,
                         CagdRType *b)
```

Length: Fitting curve Length.

NumOfPoints: Number of points in the points cloud.

Points: Points cloud. Array of points with size = NumOfPoints.

Basis: Array of size (NumOfPoints * Length) containing basis function coefficients at the foot points

FootPoints: Footpoints (the closest points on the curve) array.

Distances: Array of distances between each point to the corresponding footpoint.

Tangents: Array of curve tangents at each footpoint.

Normals: Array of curve normals at each footpoint.

Curvatures: Array of curve curvature radiuses at each footpoint.

IsOuter: Array of booleans that indicates outer points.

A: Input/output matrix (2Length * 2Length).

b: Input/output offset vector (2Length * 1).

Returns: void

Description: Calculates the SD error minimization equation Matrix and offset vector, i.e. A and b of the Ax=b equation. The caluclated values are ADDED to the A and b parameters The order of variables in x is assumed to be: (D1_x,D2_x,...,DLength_x,D1_y,D2_y,...,DLength_y) NOTE: The size of each array except Basis must be 'NumOfPoints'.

Chapter 4

Geometry Library, geom_lib

4.1 General Information

This library handles general computational geometry algorithms and geometric queries such as a distance between two lines, bounding boxes, convexity and convex hull of polygons, polygonal constructors of primitives (cylinders, spheres, etc.), basic scan conversion routines, etc.

4.2 Library Functions

4.2.1 GM2BiTansFromCircCirc (geom_bsc.c:3207)

circle circle tangenties

```
int GM2BiTansFromCircCirc(const IrtPtType Center1,
                          IrtRType Radius1,
                          const IrtPtType Center2,
                          IrtRType Radius2,
                          int OuterTans,
                          IrtPtType TanPts[2][2])
```

Center1, Radius1: Geometry of first circle.

Center2, Radius2: Geometry of second circle.

OuterTans: TRUE for outer two tangents, FALSE for inner two.

TanPts: The two tangents designated by the end points of the Segments.

Returns: TRUE for successful computation, FALSE for failure or no such bitangents exist for the current configuration.

Description: Finds the two pairs of tangent points of the given two planar circles.
See also: GM2PointsFromLineLine, GM2PointsFromCircCirc3D, GMCircleFrom3Points, , GMCircleFrom2Pts2Tans, GM2PointsFromCircCirc, GM2TanLinesFromCircCirc, , GM2IsPtInsideCirc,

4.2.2 GM2PointsFromCircCirc (geom_bsc.c:2784)

circle circle intersection

```
int GM2PointsFromCircCirc(const IrtPtType Center1,
                          IrtRType Radius1,
                          const IrtPtType Center2,
                          IrtRType Radius2,
                          IrtPtType Inter1,
                          IrtPtType Inter2)
```

Center1, Radius1: Geometry of first circle.

Center2, Radius2: Geometry of second circle.

Inter1, Inter2: Where the two intersection locations will be placed.

Returns: TRUE for successful computation, FALSE for failure.

Description: Finds the two intersection points of the given two planar circles.
See also: GM2PointsFromLineLine, GM2PointsFromCircCirc3D, GMCircleFrom3Points, , GMCircleFrom2Pts2Tans, GM2BiTansFromCircCirc, GM2TanLinesFromCircCirc, , GM2IsPtInsideCirc,

4.2.3 GM2PointsFromCircCirc3D (geom_bsc.c:2868)

```
int GM2PointsFromCircCirc3D(const IrtPtType Cntr1,
                            const IrtVecType Nrml1,
                            IrtRType Rad1,
                            const IrtPtType Cntr2,
                            const IrtVecType Nrml2,
                            IrtRType Rad2,
                            IrtPtType Inter1,
                            IrtPtType Inter2)
```

Cntr1, Nrml1, Rad1: Center, normal and radius of first circle.

Cntr2, Nrml2, Rad2: Center, normal and radius of second circle.

Inter1: First intersection location in E3.

Inter2: Second intersection location in E3.

Returns: Number of intersections found - 0, 1, or 2.

Description: Compute the intersection of two circles in general position in R^3. The circles are centered at Cntr1/2 in a plane normal to Nrml1/2 and have a radius of Rad1/2. The upto two intersections are returned in Inter1/2.

See also: GM2PointsFromLineLine, GM2PointsFromCircCirc, GMCircleFrom3Points, , GMCircleFrom2Pts2Tans, GM2BiTansFromCircCirc, GM2TanLinesFromCircCirc, , GM2IsPtInsideCirc,

4.2.4 GM2PointsFromLineLine (geom_bsc.c:1223)

line line distance

line line intersection

```
int GM2PointsFromLineLine(const IrtPtType Pl1,
                          const IrtPtType Vl1,
                          const IrtPtType Pl2,
                          const IrtPtType Vl2,
                          IrtPtType Pt1,
                          IrtRType *t1,
                          IrtPtType Pt2,
                          IrtRType *t2)
```

Pl1, Vl1: Position and direction defining the first line.

Pl2, Vl2: Position and direction defining the second line.

Pt1: Point on Pt1 that is closest to line 2.

t1: Parameter value of Pt1 as (Pl1 + Vl1 * t1).

Pt2: Point on Pt2 that is closest to line 1.

t2: Parameter value of Pt2 as (Pl2 + Vl2 * t2).

Returns: TRUE, if successful.

Description: Routine to find the two points Pti on the lines (Pli, Vli), i = 1, 2 with the minimal Euclidean distance between them. In other words, the distance between Pt1 and Pt2 is defined as distance between the two lines. The two points are calculated using the fact that if V = (Vl1 cross Vl2) then these two points are the intersection point between the following: Point 1 - a plane (defined by V and line1) and the line line2. Point 2 - a plane (defined by V and line2) and the line line1. This function returns TRUE iff the two lines are not parallel! This function is also valid for the case of coplanar lines.

See also: GGMPointFromPlanarLineLine,

4.2.5 GM2TanLinesFromCircCirc (geom_bsc.c:3256)

circle circle tangenties

```
int GM2TanLinesFromCircCirc(const IrtPtType Center1,
                            IrtRType Radius1,
                            const IrtPtType Center2,
                            IrtRType Radius2,
                            int OuterTans,
                            IrtLnType Tans[2])
```

Center1, Radius1: Geometry of first circle.

Center2, Radius2: Geometry of second circle.

OuterTans: TRUE for outer two tangents, FALSE for inner two.

Tans: The two tangent lines designated by line equatios.

Returns: TRUE for successful computation, FALSE for failure or no such bitangents exist for the current configuration.

Description: Finds the two tangent lines to the given two planar circles.
See also: GM2PointsFromLineLine, GM2PointsFromCircCirc3D, GMCircleFrom3Points, , GMCircleFrom2Pts2Tans, GM2PointsFromCircCirc, GM2BiTansFromCircCirc, , GM2IsPtInsideCirc,

4.2.6 GM3Pts2EqltrlTriMat (geomat3d.c:1317)

```
int GM3Pts2EqltrlTriMat(const IrtPtType Pt1Orig,
                        const IrtPtType Pt2Orig,
                        const IrtPtType Pt3Orig,
                        IrtHmgnMatType Mat)
```

Pt1Orig, Pt2Orig, Pt3Orig: The three vertices of the input triangle.

Mat: The computed transform.

Returns: TRUE if successful, FALSE otherwise.

Description: Compute the linear transform that maps the given planar triangle Pt1Pt2Pt3 to an equilateral triangle around the origin so that edge Pt1Pt2 is horizontal and remains of the same size.
See also: GMGenMatrix3Pts2EqltrlTri,

4.2.7 GMAffineTransUVVals (poly_pts.c:1853)

```
void GMAffineTransUVVals(IPObjectStruct *PObj,
                         const IrtRType Scale[2],
                         const IrtRType Trans[2])
```

PObj: A polygonal object to affine transform the UV vals.

Scale: UV scale factors.

Trans: UV translational factors.

Returns: void

Description: Affine transform the given UV coordinates in polygonal object PObj, in place.
See also: GMGenUVValsForPolys,

4.2.8 GMAllIntersLinePolygon2D (polysmth.c:618)

```
int GMAllIntersLinePolygon2D(const IPVertexStruct *VS,
                             const IrtPtType V1,
                             const IrtPtType V2,
                             IrtRType *AllPrms)
```

VS: Cyclic List of vertices of the Polygon.

V1, V2: The end points of the line.

AllPrms: All the blending values from V1 to V2 where the intersections have occurred will be placed here as - t*V1 + (1-t)*V2. Can be NULL to ignore. The vector, if exists, will be sorted in increasing values.

Returns: Number of intersections that detected, zero to ignore.

Description: Check if a line V1V2 and polygon VS interest, in 2D.
See also: GMIsInterLinePolygon2D, GMIsInterLineLine2D,

4.2.9 GMAngleSphericalTriangle (geom_bsc.c:2073)

```
IrtRType GMAngleSphericalTriangle(const IrtVecType Dir,
                                  const IrtVecType ODir1,
                                  const IrtVecType ODir2)
```

Dir: Spherical vertex to compute its angle with respect to ODir1/2.

ODir1, ODir2: Other two vertices of spherical triangle.

Returns: Spherical angle at Dir.

Description: Computes the angle at Dir, with respect to ODir1 and ODir2.
See also: GMAreaSphericalTriangle,

4.2.10 GMAnimAffineTransAnimation (animate.c:271)

animation

```
int GMAnimAffineTransAnimation(const IPObjectStruct *PObjs,
                               IrtRType Trans,
                               IrtRType Scale)
```

PObjs: Objects to update animation domain.

Trans: Translation amount.

Scale: Scale amount.

Returns: TRUE if there are animation attributes. FALSE otherwise.

Description: Affine transform the animation domain in the given object(s), in place. Domain "D" is mapped to domain "D * Scale + Trans".
See also: GMAnimHasAnimationOne,

4.2.11 GMAnimAffineTransAnimation2 (animate.c:458)

animation

```
int GMAnimAffineTransAnimation2(const IPObjectStruct *PObjs,
                                IrtRType Min,
                                IrtRType Max)
```

PObjs: Objects to update animation domain.

Min, Max: Desired time domain of animation.

Returns: TRUE if there are animation attributes. FALSE otherwise.

Description: Affine transform the animation domain in the given object(s), in place. Animation domain is mapped to "Min, Max".
See also: GMAnimHasAnimationOne,

4.2.12 GMAnimAffineTransAnimationOne (animate.c:304)

animation

```
int GMAnimAffineTransAnimationOne(const IPObjectStruct *PObj,
                                  IrtRType Trans,
                                  IrtRType Scale)
```

PObj: Object to update animation domain.

Trans: Translation amount.

Scale: Scale amount.

Returns: TRUE if there are animation attributes. FALSE otherwise.

Description: Affine transform the animation domain in the given object, in place. Domain "D" is mapped to domain "D * Scale + Trans".
See also: GMAnimHasAnimation,

4.2.13 GMAnimAffineTransAnimationOne2 (animate.c:493)

animation

```
int GMAnimAffineTransAnimationOne2(const IPObjectStruct *PObj,
                                   IrtRType Min,
                                   IrtRType Max)
```

PObj: Object to update animation domain.

Min, Max: Desired time domain of animation.

Returns: TRUE if there are animation attributes. FALSE otherwise.

Description: Affine transform the animation domain in the given object, in place. Animation domain is mapped to "Min, Max".
See also: GMAnimHasAnimation,

4.2.14 GMAnimCheckInterrupt (anim_aux.c:44)

```
int GMAnimCheckInterrupt(GMAnimationStruct *Anim)
```

Anim: The animation to abort.

Returns: TRUE if we need to abort, FALSE otherwise.

Description: Should we stop this animation. Senses the event queue of X11.

4.2.15 GMAnimDoAnimation (animate.c:916)

animation

```
void GMAnimDoAnimation(GMAnimationStruct *Anim, IPObjectStruct *PObjs)
```

Anim: Animation structure.

PObjs: Objects to render.

Returns: void.

Description: Routine to run a sequence of objects through an animation according to animation attributes of matrices and curves that are attached to them.
See also: GMAnimEvalAnimation,

4.2.16 GMAnimDoSingleStep (animate.c:1223)

animation

```
void GMAnimDoSingleStep(GMAnimationStruct *Anim, IPObjectStruct *PObjs)
```

Anim: Animation structure.

PObjs: Objects to render.

Returns: void.

Description: Routine to exectue a single step the animation, at current time.
See also: GMAnimDoAnimation, GMAnimEvalAnimation, GMAnimSetAnimInternalNodes,

4.2.17 GMAnimEvalAnimation (animate.c:1112)

animation

```
void GMAnimEvalAnimation(IrtRType t, IPObjectStruct *PObj)
```

t: Time to evaluate the animation at.

PObj: To evaluate their animation curves.

Returns: void

Description: Evaluate the animation curves at the given time, setting the proper animation attributes ("_animation_mat" and "_isvisible"), in place.
See also: GMAnimDoAnimation, GMAnimEvalAnimationList, GMAnimSetAnimInternalNodes, , GMAnimEvalObjAtTime,

4.2.18 GMAnimEvalAnimationList (animate.c:1198)

animation

```
void GMAnimEvalAnimationList(IrtRType t, IPObjectStruct *PObjList)
```

t: Time to evaluate the animation at.

PObjList: A list of objects to evaluate their animation curves.

Returns: void

Description: Evaluate the animation curves at the given time, setting the proper animation attributes ("_animation_mat" and "_isvisible"), in place.
See also: GMAnimDoAnimation, GMAnimEvalAnimation, GMAnimSetAnimInternalNodes,

4.2.19 GMAnimEvalObjAtTime (animate.c:1146)

animation

```
IPObjectStruct *GMAnimEvalObjAtTime(IrtRType t, IPObjectStruct *PObj)
```

t: Time to evaluate the animation at.

PObj: To evaluate their animation curves.

Returns: Input object positioned at time t.

Description: Evaluate the animation curves at the given time, and creating the object in the proper place in time.
See also: GMAnimDoAnimation, GMAnimEvalAnimationList, GMAnimSetAnimInternalNodes, GMAnimEvalAnimation,

4.2.20 GMAnimFindAnimationTime (animate.c:526)

animation

```
void GMAnimFindAnimationTime(GMAnimationStruct *Anim,
                             const IPObjectStruct *PObjs)
```

Anim: Animation structure to update.

PObjs: Objects to scan for animation attributes.

Returns: void

Description: Computes the time span for which the animation executes.
See also: GMAnimFindAnimationTimeOne,

4.2.21 GMAnimFindAnimationTimeOne (animate.c:569)

animation

```
void GMAnimFindAnimationTimeOne(GMAnimationStruct *Anim,
                                const IPObjectStruct *PObj)
```

Anim: Animation structure to update.

PObj: One object to scan for animation attributes.

Returns: void

Description: Computes the time span for which the animation executes.
See also: GMAnimFindAnimationTime,

4.2.22 GMAnimGetAnimInfoText (animate.c:91)

animation

```
void GMAnimGetAnimInfoText(GMAnimationStruct *Anim)
```

Anim: The animation state to update.

Returns: void

Description: Getting input parameters of animation from user using textual user interface.

4.2.23 GMAnimHasAnimation (animate.c:204)

animation

```
int GMAnimHasAnimation(const IPObjectStruct *PObjs)
```

PObjs: Objects to scan for animation attributes.

Returns: TRUE if there are animation attributes. FALSE otherwise.

Description: Scan the given geometry for possible animation attributes.
See also: GMAnimHasAnimationOne,

4.2.24 GMAnimHasAnimationOne (animate.c:232)

animation

```
int GMAnimHasAnimationOne(const IPObjectStruct *PObj)
```

PObj: One object to scan for animation attributes.

Returns: TRUE if there are animation attributes. FALSE otherwise.

Description: Scan the given geometry for possible animation attributes.
See also: GMAnimHasAnimation,

4.2.25 GMAnimHasDispPropAttrib (animate.c:991)

animation

```
int GMAnimHasDispPropAttrib(const IPObjectStruct *PObj)
```

PObj: Object to examine.

Returns: TRUE if require re-polygonization/polylining, FALSE otherwise.

Description: Routine to test if the given object has animation of display property (that will require rebuild of poly approximation for it).
See also: GMAnimEvalAnimation, GMAnimDoAnimation,

4.2.26 GMAnimResetAnimStruct (animate.c:58)

animation

```
void GMAnimResetAnimStruct(GMAnimationStruct *Anim)
```

Anim: The animation state to reset.

Returns: void

Description: Resets the slots of an animation structure.

4.2.27 GMAnimSaveIterationsAsImages (anim_aux.c:64)

animation

```
void GMAnimSaveIterationsAsImages(GMAnimationStruct *Anim,
                                  IPObjectStruct *PObjs)
```

Anim: Animation structure.

PObjs: Objects to render.

Returns: void

Description: Saves one iteration of the animation sequence as an image.

4.2.28 GMAnimSaveIterationsToFiles (animate.c:1265)

animation

```
void GMAnimSaveIterationsToFiles(GMAnimationStruct *Anim,
                                 IPObjectStruct *PObjs)
```

Anim: Animation structure.

PObjs: Objects to render.

Returns: void

Description: Saves one iteration of the animation sequence as IRIT data (*.itd). The objects that are saved are those that are visibled on the current time frame as set via current animation_mat attribute.

4.2.29 GMAnimSetAnimInternalNodes (animate.c:1080)

```
int GMAnimSetAnimInternalNodes(int AnimInternalNodes)
```

AnimInternalNodes: New setting for internal animation nodes.

Returns: Old settings.

Description: Allows animation transformations to be saved at internal nodes.
See also: GMAnimEvalAnimation,

4.2.30 GMAnimSetReverseHierarchyMatProd (animate.c:1051)

```
int GMAnimSetReverseHierarchyMatProd(int ReverseHierarchyMatProd)
```

ReverseHierarchyMatProd: TRUE to reverse default order.

Returns: Old settings.

Description: Controls the order in which animation matrices are multiplied in the hierarchy.
See also: GMAnimEvalAnimation,

4.2.31 GMApproxGeodesicDist (poly_dist.c:80)

```
int GMApproxGeodesicDist(IPPolygonStruct *PlMesh,
                         const IPPolygonStruct *HitPl,
                         const IrtPtType HitPt,
                         IrtRType MaxDist)
```

PlMesh: The mesh to update with the geodesic distance approximations.

HitPl: The polygon in PlMesh to start the distance search.

HitPt: The point in HitPl with the origin distance.

MaxDist: Positive value to limit the search for up to this distance. Negative value absolute value same as positive but also color the geometry red-blue-green from hit point and up to this amount distance. Zero to disable MaxDist.

Returns: TRUE if successful, FALSE otherwise.

Description: Approximates geodesic distances from a given location in the mesh, over the given mesh. Distance results, per vertex, are stored as "GeodesicDist" attribute.

4.2.32 GMAreaOfTriangle (geom_bsc.c:3395)

```
IrtRType GMAreaOfTriangle(const IrtRType *Pt1,
                          const IrtRType *Pt2,
                          const IrtRType *Pt3)
```

Pt1, Pt2, Pt3: Points to compute the area of a triangle formed by Pt1, Pt2, and Pt3, in the XY plane.

Returns: Resulting area.

Description: Computing the Area of the triangle formed by points Pt1, Pt2 and Pt3. This can be used as a test to identify whether point Pt3 lies to the left, right, or on the line (vector) formed by Pt1 and Pt2 using the this signed area of the triangle's computation.

4.2.33 GMAreaSphericalTriangle (geom_bsc.c:2037)

```
IrtRType GMAreaSphericalTriangle(const IrtVecType Dir1,
                                 const IrtVecType Dir2,
                                 const IrtVecType Dir3)
```

Dir1, Dir2, Dir3: Vertices of the spherical triangle. unit vectors.

Returns: Computed area.

Description: Computes the area of a spherical triangle over the unit sphere with given three (unit vector) vertices, Dir1, Dir2, Dir3. Area is equal to (Alpha1 + Alpha2 + Alpha3 - Pi) where Alphai is the angle at vertex Diri with the other two vertices.
See also: GMAngleSphericalTriangle,

4.2.34 GMBBComputeBboxObject (bbox.c:92)

bounding box

```
GMBBBboxStruct *GMBBComputeBboxObject(const IPObjectStruct *PObj,
                                      GMBBBboxStruct *Bbox,
                                      int HandleInvisibelObjs)
```

PObj: To compute a bounding box for.

Bbox: A bounding box of PObj is saved here.

HandleInvisibelObjs: TRUE to also include invisible objects in the bbox computation.

Returns: A bounding box of PObj - reference to Bbox.

Description: Computes a bounding box of a given object of any type.
See also: ,

4.2.35 GMBBComputeBboxObjectList (bbox.c:291)

bounding box

```
GMBBBboxStruct *GMBBComputeBboxObjectList(const IPObjectStruct *PObj,
                                          GMBBBboxStruct *Bbox,
                                          int HandleInvisibelObjs)
```

PObj: To compute a bounding box for.

Bbox: A pointer to a bounding box holding bounding box information on objects PObj.

HandleInvisibelObjs: TRUE to also include invisible objects in the bbox computation.

Returns: A pointer to a bounding box holding bounding box information on objects PObj.

Description: Computes a bounding box of a list of objects of any type.

4.2.36 GMBBComputeOnePolyBbox (bbox.c:354)

bounding box

```
GMBBBboxStruct *GMBBComputeOnePolyBbox(const IPPolygonStruct *PPoly,
                                       GMBBBboxStruct *Bbox)
```

PPoly: To compute a bounding box for.

Bbox: A pointer to a bounding box holding bounding box information on PPoly.

Returns: A pointer to a bounding box holding bounding box information on PPoly.

Description: Computes a bounding box of a polygon/polyline/pointlist object.

4.2.37 GMBBComputePointBbox (bbox.c:425)

bounding box

```
GMBBBboxStruct *GMBBComputePointBbox(const IrtRType *Pt, GMBBBboxStruct *Bbox)
```

Pt: To compute a bounding box for.

Bbox: A pointer to a bounding box holding bounding box information on Pt.

Returns: A pointer to a bounding box holding bounding box information on Pt.

Description: Computes a bounding box of a point object.

4.2.38 GMBBComputePolyListBbox (bbox.c:388)

bounding box

```
GMBBBboxStruct *GMBBComputePolyListBbox(const IPPolygonStruct *PPoly,
                                        GMBBBboxStruct *Bbox)
```

PPoly: To compute a bounding box for.

Bbox: A pointer to a bounding box holding bounding box information on PPoly list.

Returns: A pointer to a bounding box holding bounding box information on PPoly list.

Description: Computes a bounding box for a list of polygon/polyline/pointlist objects.

4.2.39 GMBBMergeBbox (bbox.c:503)

bounding box

```
GMBBBboxStruct *GMBBMergeBbox(GMBBBboxStruct *MergedBbox,
                              const GMBBBboxStruct *Bbox)
```

MergedBbox: A first bounding box, to merge Bbox into.

Bbox: A second bounding box to merge with MergedBBox.

Returns: A merged bounding box. Same as MergedBbox.

Description: Merges (union) given two bounding boxes into one. Note Dim can be different in both Bboxes in which case higher dims are simply copied from the Bbox that has them.

4.2.40 GMBBMergeBboxTo (bbox.c:455)

bounding box

```
GMBBBboxStruct *GMBBMergeBboxTo(const GMBBBboxStruct *Bbox1,
                                const GMBBBboxStruct *Bbox2,
                                GMBBBboxStruct *MergedBbox)
```

Bbox1: First bounding box to union up.

Bbox2: Second bounding box to union up.

MergedBbox: A unioned bounding box that contains both BBox1 and BBox2.

Returns: A unioned bounding box that contains both BBox1 and BBox2. Same as MergedBBox.

Description: Merges (union) given two bounding boxes into one. Either Bbox1 or Bbox2 can be pointing to the same address as MergedBBox. Note Dim can be different in both Bboxes in which case higher dims are simply copied from the Bbox that has them.

4.2.41 GMBBSetBBoxPrecise (bbox.c:59)

bbox

bounding box

```
int GMBBSetBBoxPrecise(int Precise)
```

Precise: 0 for, simpler, looser bbox that is derived using the control poly/mesh. 1 for a better bbox on freeforms, by refining geometry first. 2 for a precise bbox on freeforms, by computing extrema.

Returns: old value.

Description: Enforce the computation of a precise bounding box for a freeform.
See also: GMBBComputeBboxObject,

4.2.42 GMBBSetGlblBBInstncObjList (bbox.c:324)

bounding box

```
const IPObjectStruct *GMBBSetGlblBBInstncObjList(const IPObjectStruct
                                                 *BBInstObjList)
```

BBInstObjList: Global object list to search instances at.

Returns: Old global object list.

Description: Sets the global list object to search instances at.

4.2.43 GMBaryCentric3Pts2DToData (geom_bsc.c:2647)

```
IrtRType *GMBaryCentric3Pts2DToData(const IrtPtType Pt1,
                                    const IrtPtType Pt2,
                                    const IrtPtType Pt3,
                                    const IrtPtType Pt,
                                    IrtVecType RetVal)
```

Pt1, Pt2, Pt3: Three points forming a triangular in general position.

Pt: A point for which the barycentric coordinates are to be computed.

RetVal: A pointer to a space holding the three Barycentric coefficients, or NULL if point Pt is outside the triangle Pt1 Pt2 Pt3.

Returns: A pointer to a space holding the three Barycentric coefficients, or NULL if point Pt is outside the triangle Pt1 Pt2 Pt3.

Description: Computes the Barycentric coordinates of given point Pt with respect to given Triangle Pt1 Pt2 Pt3. All points are assumed to be in the XY plane.
See also: GMBaryCentric3PtsToData,

4.2.44 GMBaryCentric3PtsToData (geom_bsc.c:2708)

```
IrtRType *GMBaryCentric3PtsToData(const IrtPtType Pt1,
                                  const IrtPtType Pt2,
                                  const IrtPtType Pt3,
                                  const IrtPtType Pt,
                                  IrtVecType RetVal)
```

Pt1, Pt2, Pt3: Three points forming a triangular in general position.

Pt: A point for which the barycentric coordinates are to be computed.

RetVal: A pointer to a space holding the three Barycentric coefficients, or NULL if point Pt is outside the triangle Pt1 Pt2 Pt3.

Returns: A pointer to a space holding the three Barycentric coefficients, or NULL if point Pt is outside the triangle Pt1 Pt2 Pt3.

Description: Computes the Barycentric coordinates of given point Pt with respect to given Triangle Pt1 Pt2 Pt3. All points are assumed to be coplanar.
See also: GMBaryCentric3Pts2DToData,

4.2.45 GMBasicSetEps (geom_bsc.c:60)

```
IrtRType GMBasicSetEps(IrtRType Eps)
```

Eps: New epsilon to use.

Returns: Old epsilon value.

Description: Sets the epsilon to use in basic geometry processing.

4.2.46 GMBlendNormalsToVertices (intrnrml.c:740)

```
void GMBlendNormalsToVertices(IPPolygonStruct *PlList,
                              IrtRType MaxAngle)
```

PlList: List of polygons to blend the normals of their vertices

MaxAngle: Between approximated normal at vertex and polygon normal of the vertex to allow averaging. In degrees. If negative, all vertices normals are cleared and all polygon normals reevaluated (and no vertices normals are evaluated).

Returns: void

Description: Approximate normals to all vertices of the given geometry by blending the normals of the faces that share the vertex. Assumes polygons are properly oriented. Places on each vertex an "_CosNrmlMaxDev" attribute with the maximal deviation of this normal from an adjacent polygon plane (cosine of the maximal angle).
See also: GMUpdateVerticesByInterp, GMFixNormalsOfPolyModel,

4.2.47 GMBoxBVHConstructGeneralAlignedFrustum (box2BVH.c:300)

```
void GMBoxBVHConstructGeneralAlignedFrustum(const IrtRType Center[3],
                                            IrtRType SizeX,
                                            IrtRType SizeY,
                                            const IrtRType XAxis[3],
                                            const IrtRType HeightAxis[3],
                                            IrtRType Height,
                                            IrtRType XAngle,
                                            IrtRType YAngle,
                                            GMGeneralFrustumInfoStruct *Frustum)
```

Center: The center of the narrow base of the frustum.

SizeX, SizeY: The size of the narrow base of the frustum.

XAxis: The X direction of the bases of the frustum.

HeightAxis: The height (Z) axis of the frustum.

Height: The height of the frustum.

XAngle, YAngle: The angles of the frustum in the X and Y directions, respectively.

Frustum: The resulting Frustum.

Returns: void

Description: Construct a frustum with the given parameters. The frustum consists of six quadrilateral sides, each with an inward-pointing normal.

4.2.48 GMBoxBVHCreate (box_BVH.c:433)

```
GMBoxBVHStruct *GMBoxBVHCreate(const GMBoxBVHInfoStruct **Boxes,
                               int Num,
                               int Size)
```

Boxes: The boxes to arrange in the BVH.

Num: The number of boxes.

Size: The size of the leaf nodes.

Returns: The returned BVH.

Description: construct a box BVH with the given boxes.
See also:

4.2.49 GMBoxBVHDbgConstructGeneralAlignedFrustum (box2BVH.c:497)

```
IPPolygonStruct *GMBoxBVHDbgConstructGeneralAlignedFrustum(
                                            const IrtRType Center[3],
                                            IrtRType SizeX,
                                            IrtRType SizeY,
                                            const IrtRType XAxis[3],
                                            const IrtRType HeightAxis[3],
                                            IrtRType Height,
                                            IrtRType XAngle,
                                            IrtRType YAngle)
```

Center: The center of the narrow base of the frustum.

SizeX, SizeY: The size of the narrow base of the frustum.

XAxis: The X direction of the bases of the frustum.

HeightAxis: The height (Z) axis of the frustum.

Height: The height of the frustum.

XAngle, YAngle: The angles of the frustum in the X and Y directions, respectively.

Returns: a 3D visualization of the frustum.

Description: Construct a 3D visualization of the frustum, based on its parameters. The frustum is represented by a list of six IPPolygonStruct, one for side of the frustum.
See also: ,

4.2.50 GMBoxBVHDbgConstructGeneralAlignedFrustum2 (box2BVH.c:567)

```
IPObjectStruct *GMBoxBVHDbgConstructGeneralAlignedFrustum2(
                                            const IrtRType Center[3],
                                            IrtRType SizeX,
                                            IrtRType SizeY,
                                            const IrtRType XAxis[3],
                                            const IrtRType HeightAxis[3],
                                            IrtRType Height,
                                            IrtRType XAngle,
                                            IrtRType YAngle)
```

Center: The center of the narrow base of the frustum.

SizeX, SizeY: The size of the narrow base of the frustum.

XAxis: The X direction of the bases of the frustum.

HeightAxis: The height (Z) axis of the frustum.

Height: The height of the frustum.

XAngle, YAngle: The angles of the frustum in the X and Y directions, respectively.

Returns: a 3D visualization of the frustum.

Description: Construct a 3D visualization of the frustum, based on its parameters. The frustum is represented by a IPObjectStruct, construed as a "ruled" object of two bases of the frustum.
See also: ,

4.2.51 GMBoxBVHFree (box_BVH.c:464)

```
void GMBoxBVHFree(GMBoxBVHStruct *BVH)
```

BVH: The BVH to free.

Returns: void

Description: frees a box BVH .
See also:

4.2.52 GMBoxBVHGetBoxIntersection (box_BVH.c:513)

```
int GMBoxBVHGetBoxIntersection(const GMBoxBVHStruct *BVH,
                               const IrtRType *Max,
                               const IrtRType *Min,
                               int *Res)
```

BVH: The BVH.

Max: The maximum coordinates of the box.

Min: The minimum coordinates of the box.

Res: The array to write the result into.

Returns: The number of boxes in the result.

Description: Get the Id for all boxes intersecting the given box.
See also:

4.2.53 GMBoxBVHGetBoxNum (box_BVH.c:486)

```
int GMBoxBVHGetBoxNum(const GMBoxBVHStruct *BVH)
```

BVH: The BVH.

Returns: the number of boxes in the BVH.

Description: Get the number of boxes in a box BVH.
See also:

4.2.54 GMBoxBVHGetFrustumIntersection (box_BVH.c:545)

```
int GMBoxBVHGetFrustumIntersection(const GMBoxBVHStruct *BVH,
                                   const IrtRType *Max,
                                   const IrtRType *Min,
                                   IrtRType Angle,
                                   int *Res)
```

BVH: The BVH.

Max: The maximum coordinates of the frustum (z is the frustum top, x, y are the maximum coordinates of the lower base).

Min: The minimum coordinates of the frustum (z is the frustum bottom, x, y are the minimum coordinates of the lower base).

Angle: The angle of the frustum.

Res: The array to write the result into.

Returns: the number of boxes in the result.

Description: Get the Id for all boxes intersecting the given frustum. The frustum is z aligned wider towards the top.
See also:

4.2.55 GMBoxBVHGetGeneralAlignedFrustumIntersection (box2BVH.c:423)

```
int GMBoxBVHGetGeneralAlignedFrustumIntersection(const GMBoxBVHStruct *BVH,
                                                 const IrtRType Center[3],
                                                 IrtRType SizeX,
                                                 IrtRType SizeY,
                                                 const IrtRType XAxis[3],
                                                 const IrtRType HeightAxis[3],
                                                 IrtRType Height,
                                                 IrtRType XAngle,
                                                 IrtRType YAngle,
                                                 int *Res)
```

BVH: The BVH.

Center: The center of the narrow base of the frustum.

SizeX, SizeY: The size of the narrow base of the frustum.

XAxis: The X direction of the bases of the frustum.

HeightAxis: The height (Z) axis of the frustum.

Height: The height of the frustum.

XAngle, YAngle: The angles of the frustum in the X and Y directions, respectively.

Res: The array to write the result into.

Returns: the number of boxes in the result.

Description: Get the Id for all boxes overlapping the given frustum. The frustum can have a general-aligned axis and orientation. It is defined by a center point, x and y sizes of the NARROW base, x and height axis directions, height and angle.

See also:

4.2.56 GMBoxBVHGetGeneralAlignedFrustumIntersection2 (box2BVH.c:460)

```
int GMBoxBVHGetGeneralAlignedFrustumIntersection2(
                                    const GMBoxBVHStruct *BVH,
                                    const GMGeneralFrustumInfoStruct *Frustum,
                                    int *Res)
```

BVH: The BVH.

Frustum: The frustum with to interse

Res: The array to write the result into.

Returns: the number of boxes in the result.

Description: Get the Id for all boxes overlapping the given frustum. The frustum can have a general-aligned axis and orientation. The frustum can be constructed with the function GMBoxBVHConstructGeneralAlignedFrustum.

See also: GMBoxBVHConstructGeneralAlignedFrustum,

4.2.57 GMBoxBVHTestFrustumIntersection (box_BVH.c:584)

```
int GMBoxBVHTestFrustumIntersection(const GMBoxBVHStruct *BVH,
                                    const IrtRType *Max,
                                    const IrtRType *Min,
                                    IrtRType Angle)
```

BVH: The BVH.

Max: The maximum coordinates of the frustum (z is the frustum top, x, y are the maximum coordinates of the lower base).

Min: The minimum coordinates of the frustum (z is the frustum bottom, x, y are the minimum coordinates of the lower base).

Angle: The angle of the frustum.

Returns: 0 if no BVH data intersects the frustum.

Description: Check if any Boxes are in the frustum aligned along the Z axis.

See also:

4.2.58 GMBspPtsAddSlices (bsp_pts.c:507)

```
void GMBspPtsAddSlices(GMBspPtsNodeStruct *Head,
                       GMBspPtsSliceInfoStruct *AddedSlices)
```

Head: The Head of a Bsp tree.

AddedSlices: The Slices to divide with.

Returns: void

Description: The function take a Head of a Bsp tree as an input, then splits its leafs iteratively using the given slices's hyper-planes.

See also:

4.2.59 GMBspPtsAllocSliceStruct (bsp_pts.c:137)

```
GMBspPtsSliceInfoStruct *GMBspPtsAllocSliceStruct()
```

Returns: The allocated slice.

Description: Allocate a new GMBspPtsSliceInfoStruct.
See also:

4.2.60 GMBspPtsCreateTree (bsp_pts.c:222)

```
GMBspPtsNodeStruct *GMBspPtsCreateTree(IPVertexStruct *PC,
                                       GMBspPtsSliceInfoStruct *Slices,
                                       int StartIdx,
                                       int EndIdx)
```

PC: A list of vertices to place in the tree.

Slices: A list of ordered slices for space division.

StartIdx: The index of the first slice to divide with.

EndIdx: The index of the last slice to divide with.

Returns: The Head Node of the constructed tree.

Description: The function takes as an input a point cloud of points, then creates a Binary space partition tree by dividing the space according to the given slices' hyper-planes.
See also:

4.2.61 GMBspPtsFreeSliceStruct (bsp_pts.c:162)

```
void GMBspPtsFreeSliceStruct(GMBspPtsSliceInfoStruct *Slice)
```

Slice: A pointer to a GMBspPtsSliceInfoStruct.

Returns: void

Description: Deallocate the given GMBspPtsSliceInfoStruct pointer.
See also:

4.2.62 GMBspPtsFreeSliceStructList (bsp_pts.c:187)

```
void GMBspPtsFreeSliceStructList(GMBspPtsSliceInfoStruct *SliceList)
```

SliceList: A pointer to GMBspPtsSliceInfoStruct list object.

Returns: void

Description: Deallocate a list of a GMBspPtsSliceInfoStruct.
See also:

4.2.63 GMBspPtsFreeTree (bsp_pts.c:287)

```
void GMBspPtsFreeTree(GMBspPtsNodeStruct *Head)
```

Head: The Head of the Bsp tree.

Returns: void

Description: Deallocate the Bsp tree that spanned from the given Node.
See also:

4.2.64 GMBspPtsGetVertices (bsp_pts.c:370)

```
IPVertexStruct *GMBspPtsGetVertices(GMBspPtsNodeStruct *Head,
                                    GMBspPtsSliceInfoStruct *LeftSlc,
                                    GMBspPtsSliceInfoStruct *RightSlc)
```

Head: The Head of the Bsp tree.

LeftSlc: Slice contains the left hyper-plane of the search interval.

RightSlc: Slice contains the right hyper-plane of the search interval.

Returns: A list of points that lies between the given slices.

Description: The function returns a copy of all the vertices that lies between the two given slices.
See also:

4.2.65 GMCircleFrom2Pts2Tans (geom_bsc.c:3043)

circle

```
int GMCircleFrom2Pts2Tans(IrtPtType Center,
                          IrtRType *Radius,
                          const IrtPtType Pt1,
                          const IrtPtType Pt2,
                          const IrtVecType Tan1,
                          const IrtVecType Tan2)
```

Center: Of computed circle.

Radius: Of computed circle.

Pt1, Pt2: Two points to fit a circle through.

Tan1, Tan2: Two tangents to the circle at Pt1, Pt2.

Returns: TRUE if successful, FALSE otherwise.

Description: Routine to construct a circle through given 3 points. If two of the points are the same or the three points are collinear it returns FALSE, otherwise (successful), it returns TRUE.
See also: GM2PointsFromCircCirc3D, GM2PointsFromLineLine, GM2PointsFromCircCirc, , GMCircleFrom3Points, GM2BiTansFromCircCirc, GM2TanLinesFromCircCirc, , GM2IsPtInsideCirc,

4.2.66 GMCircleFrom3Points (geom_bsc.c:2976)

circle

```
int GMCircleFrom3Points(IrtPtType Center,
                        IrtRType *Radius,
                        const IrtPtType Pt1,
                        const IrtPtType Pt2,
                        const IrtPtType Pt3)
```

Center: Of computed circle.

Radius: Of computed circle.

Pt1, Pt2, Pt3: Three points to fit a circle through.

Returns: TRUE if successful, FALSE otherwise.

Description: Routine to construct a circle through given 3 points. If two of the points are the same or the three points are collinear it returns FALSE, otherwise (successful), it returns TRUE.
See also: GM2PointsFromCircCirc3D, GM2PointsFromLineLine, GM2PointsFromCircCirc, , GMCircleFrom2Pts2Tans, GM2BiTansFromCircCirc, GM2TanLinesFromCircCirc, , GM2IsPtInsideCirc,

4.2.67 GMCircleFromLstSqrPts (geom_bsc.c:3106)

```
int GMCircleFromLstSqrPts(IrtPtType Center,
                          IrtRType *Radius,
                          const IrtPtType *Pts,
                          int PtsSize)
```

Center: The circle's center is returned in this param.

Radius: The circle's radius is returned in this param.

Pts: Vector of points to fit a circle to, least squares.

PtsSize: Size of the PTs vector.

Returns: TRUE if successful, FALSE otherwise.

Description: Compute a least-sqaures fitted circle to a set of points, in the XY plane. Returns the center and radius of the computed circle. Solving the problem in a new (U,V) apace by minimizing P^2, P=(U-Uc)^2+(V-Vc)^2-R^2, where (Uc,Vc) the circle center. For the full solution, see http://www.dtcenter.org/met/users/docs/write_ups/circle_fit.pdf.
See also:

4.2.68 GMCleanUpDupPolys (poly_cln.c:139)

duplicated polygons

cleaning

```
IPPolygonStruct *GMCleanUpDupPolys(IPPolygonStruct **PPolygons, IrtRType Eps)
```

PPolygons: List of polygons to clean, in place.

Eps: Tolerance of vertices equality, etc.

Returns: Reference to the filtered polygons.

Description: Routine to search and remove duplicated identical polygons in the input model, in place.
See also: GMCleanUpPolylineList, GMVrtxListToCircOrLin, GMFilterInteriorVertices, GMCleanUpPolygonList, GMTwoPolySameGeom,

4.2.69 GMCleanUpPolygonList (poly_cln.c:184)

zero length edges

cleaning

```
IPPolygonStruct *GMCleanUpPolygonList(IPPolygonStruct **PPolygons,
                                      IrtRType Eps)
```

PPolygons: List of polygons to clean, in place.

Eps: Tolerance of vertices equality, etc.

Returns: Reference to the filtered polygons.

Description: Routine to clean up polygons - delete zero length edges, and polygons with less than 3 vertices, in place.
See also: GMCleanUpPolylineList, GMVrtxListToCircOrLin, GMFilterInteriorVertices, GMCleanUpDupPolys,

4.2.70 GMCleanUpPolylineList (poly_cln.c:274)

zero length polyline

cleaning

```
IPPolygonStruct *GMCleanUpPolylineList(IPPolygonStruct **PPolylines,
                                       IrtRType Eps)
```

PPolylines: List of polylines to clean, in place.

Eps: Tolerance of vertices equality, etc.

Returns: Reference to the filtered polylines.

Description: Routine to clean up polylines of zero length, in place. Valid polylines are assumed to hold at least 3 points, by this function.
See also: GMCleanUpPolygonList, GMVrtxListToCircOrLin, GMFilterInteriorVertices, GMCleanUpPolylineList2,

4.2.71 GMCleanUpPolylineList2 (poly_cln.c:343)

colinear points

cleaning

```
IPPolygonStruct *GMCleanUpPolylineList2(IPPolygonStruct *PPolylines)
```

PPolylines: List of polylines to clean, in place.

Returns: Reference to the filtered polylines.

Description: Routine to clean up colinear points in polylines, in place.
See also: GMCleanUpPolygonList, GMVrtxListToCircOrLin, GMFilterInteriorVertices, GMCleanUpPolylineList,

4.2.72 GMClipPolysAgainstPlane (poly_cln.c:673)

```
IPPolygonStruct *GMClipPolysAgainstPlane(IPPolygonStruct *PHead,
                                         IPPolygonStruct **PClipped,
                                         IPPolygonStruct **PInter,
                                         IrtPlnType Plane)
```

PHead: Pointer to head of a NULL terminated list of polygons.

PClipped: List of clipped polygons on the negative side of Plane, if any.

PInter: List of polygons that intersects Plane, if any.

Plane: Plane to clip against.

Returns: List of polygons in the positive domain of the Plane.

Description: Clips polygons that are at the negative side of the plane foreach $Ax + By + Cz + D < 0$. Clipped polygons are returned in PClipped list whereas polygons that intersects the plane Plane are returned in PInter.

4.2.73 GMCollinear3Pts (geom_bsc.c:358)

collinearity

```
int GMCollinear3Pts(const IrtPtType Pt1,
                    const IrtPtType Pt2,
                    const IrtPtType Pt3)
```

Pt1, Pt2, Pt3: Three points to verify for collinearity.

Returns: TRUE if collinear, FALSE otherwise.

Description: Verifies the collinearity of three points.
See also: GMCollinear3PtsInside,

4.2.74 GMCollinear3PtsInside (geom_bsc.c:439)

collinearity

```
int GMCollinear3PtsInside(const IrtPtType Pt1,
                          const IrtPtType Pt2,
                          const IrtPtType Pt3)
```

Pt1, Pt2, Pt3: Three points to verify for collinearity.

Returns: TRUE if collinear and inside segment, FALSE otherwise (including the case of Pt1 == Pt2 or Pt3 == Pt2).

Description: Verifies the collinearity of three points and that Pt2 is inside (up to GMBasicColinEps) the line segment (Pt1, Pt3).
See also: GMCollinear3Pts,

4.2.75 GMCollinear3Vertices (intrnrml.c:200)

collinearity

```
int GMCollinear3Vertices(const IPVertexStruct *V1,
                         const IPVertexStruct *V2,
                         const IPVertexStruct *V3)
```

V1, V2, V3: Vertices to test for collinearity.

Returns: TRUE if collinear, FALSE otherwise.

Description: Verify the collinearity of the given three vertices.

4.2.76 GMComplexRoot (geom_bsc.c:3965)

```
void GMComplexRoot(IrtRType RealVal,
                   IrtRType ImageVal,
                   IrtRType *RealRoot,
                   IrtRType *ImageRoot)
```

RealVal, ImageVal: The number to compute the root for.

RealRoot, ImageRoot: The computed root.

Returns: void

Description: Computes one root of an imagniary number.

4.2.77 GMComputeAverageVertex (polysmth.c:546)

```
int GMComputeAverageVertex(const IPVertexStruct *VS,
                           IrtPtType CenterPoint,
                           IrtRType BlendFactor)
```

VS: List of vertices of the Polygon.

CenterPoint: Input center location into which average is to be blended.

BlendFactor: 1.0 to move the vertex all the way to the average position otherwise (less than 1.0) to the proper fraction.

Returns: TRUE if computed average is valid, FALSE otherwise.

Description: Computes the average location of the vertices in poly VS and blend this average with CenterPoint with ratio BlendFactor (== CenterPoint if 0).

See also: GMPolyCentroid, GMComputeAverageVertex2, GMPolygonGetCentroid,

4.2.78 GMComputeAverageVertex2 (polysmth.c:482)

```
int GMComputeAverageVertex2(const int *NS,
                            const IPPolyVrtxArrayStruct *PVIdx,
                            IrtPtType CenterPoint,
                            int CenterIndex,
                            IrtRType BlendFactor,
                            IrtRType DesiredRadius)
```

NS: List of indices of vertices to average in PVIdx.

PVIdx: Vertex array data structure.

CenterPoint: Input center location into which average is to be blended.

CenterIndex: Input center location index in PVIdx structure.

BlendFactor: 1.0 to move the vertex all the way to the average position otherwise (less than 1.0) to the proper fraction.

DesiredRadius: If positive and curvature attributes (K1Curv and K2Curv) are provided, verify we do not smooth this vertex beyond DesiredRadius (examining max. principle curvature value).

Returns: TRUE if computed average is valid, FALSE otherwise.

Description: Computes the average location of the vertices in vertex indices NS and blend this average with CenterPoint with ratio BlendFactor (== CenterPoint if 0).

See also: GMPolyCentroid, GMComputeAverageVertex, GMPolygonGetCentroid,

4.2.79 GMConvertPolyToTriangles (poly_pts.c:295)

```
IPPolygonStruct *GMConvertPolyToTriangles(IPPolygonStruct *Pl)
```

Pl: A convex polygon to convert to triangles. If Pl contains collinear edges, singular polygons are likely to be generated as a result.

Returns: A list of triangles, to which Pl also points.

Description: Converts a single convex polygon to a set of triangles, in place.
See also: GMConvertPolysToTriangles2, GMConvertPolysToTriangles,

4.2.80 GMConvertPolysToNGons (poly_pts.c:180)

```
IPObjectStruct *GMConvertPolysToNGons(const IPObjectStruct *PlObj,
                                      int n,
                                      int HandleNormals)
```

PlObj: Polygonal object to split into up to n-gons.

n: Maximal number of vertices.

HandleNormals: TRUE to also process normals of vertices.

Returns: A polygonal object containing polygons with upto n vertices, representing the same model as PolyObj.

Description: Creates a new polygonal objects out of given one, that contains only polygons of up to n vertices. Non convex polygons are split to convex one so the result will contain convex data only.
See also: ConvexPolyObjectN, GMConvertPolysToTriangles, GMConvertPolysToRectangles,

4.2.81 GMConvertPolysToRectangles (poly_pts.c:801)

```
IPObjectStruct *GMConvertPolysToRectangles(IPObjectStruct *PolyObj)
```

PolyObj: Polygonal object to split into rectangles, in place.

Returns: Return list of rectangular polygons.

Description: Creates a new polygonal object out of given one, that contains only rectangles. Non convex polygons with an empty kernel will generate self- intersecting results. By selecting a centroid location in the kernel of each polygon and connecting that centroid location to all the middle two adjacent edges, for all edges, n rectangular regions are defined for each n-gon, about half the (edge) size.
See also: ConvexPolyObjectN, GMConvertPolysToNGons, GMLimitTrianglesEdgeLen, GMConvertPolysToTriangles, GMConvertPolysToTriangles2,

4.2.82 GMConvertPolysToTriangles (poly_pts.c:562)

```
IPObjectStruct *GMConvertPolysToTriangles(const IPObjectStruct *PObj)
```

PObj: Polygonal object to split into triangles.

Returns: A polygonal object containing only triangles representing the same model as PolyObj.

Description: Creates a new polygonal objects out of given one, that contains only triangles. Non convex polygons are split to convex one which, in turn, converted to triangles. Collinear points are purged away.
See also: ConvexPolyObjectN, GMConvertPolysToNGons, GMLimitTrianglesEdgeLen, , GMConvertPolysToTriangles2, GMConvertPolysToRectangles, , GMConvertPolyToTriangles,

4.2.83 GMConvertPolysToTriangles2 (poly_pts.c:602)

```
IPObjectStruct *GMConvertPolysToTriangles2(const IPObjectStruct *PObj)
```

PObj: Polygonal object to split into triangles.

Returns: A polygonal object containing only triangles representing the same model as PolyObj.

Description: Creates a new polygonal objects out of given one, that contains only triangles. Non convex polygons are split to convex one which, in turn, converted to triangles. Collinear points are used and split at.
See also: ConvexPolyObjectN, GMConvertPolysToNGons, GMLimitTrianglesEdgeLen, GMConvertPolysToTriangles, GMConvertPolysToRectangles,

4.2.84 GMConvertPolysToTrianglesIntrrPt (poly_pts.c:708)

```
IPObjectStruct *GMConvertPolysToTrianglesIntrrPt(IPObjectStruct *PolyObj)
```

PolyObj: Polygonal object to split into triangles, in place.

Returns: A polygonal object containing only triangles representing the same model as PolyObj.

Description: Creates a new polygonal objects out of given one, that contains only triangles. Non convex polygons are split to convex one which, in turn, converted to triangles. This version triangulates convex polygons by connecting each vertex to the geometric centroid, hence co-linear pts are not purged, and do not create degenerate triangles. All other fields are not handled (normals, etc').
See also: ConvexPolyObjectN, GMConvertPolysToNGons, GMLimitTrianglesEdgeLen, GMConvertPolysToTriangles, GMConvertPolysToRectangles,

4.2.85 GMConvexHull (cnvxhull.c:61)

```
int GMConvexHull(IrtE2PtStruct *DTPts, int *NumOfPoints)
```

DTPts: The set of point to compute their convex hull.

NumOfPoints: Number of points in set DTPts.

Returns: TRUE if successful, FALSE otherwise.

Description: Convex Hull computation of a set of points. The Convex Hull is returned in place, updating NumOfPoints. Algorithm is based on two articles:

```
1. An Efficient Algorithm For Determining The convex Hull of a Finite Set.
   By R.L. Graham, Information processing letters (1972) 132-133.
2. A Reevolution of an Efficient Algorithm For Determining The Convex Hull
   of a Finite Planar Set. By K.R. Anderson, Information Processing
   Letters, January 1978, Vol. 7, Num. 1, 53-55.
```

See also: GMMonotonePolyConvex,

4.2.86 GMConvexNormalizeNormal (convex.c:200)

convexity

convex polygon

```
int GMConvexNormalizeNormal(int NormalizeNormals)
```

NormalizeNormals: TRUE to normalize after blend, FALSE just to blend.

Returns: Old value.

Description: Routine to set how normals are blended - with or without normalizations.
See also: GMConvexPolyObjectN, GMConvexPolyObject,

4.2.87 GMConvexPolyObject (convex.c:266)

convexity

convex polygon

```
void GMConvexPolyObject(IPObjectStruct *PObj, int HandleNormals)
```

PObj: To test for convexity of its polygons, and split into convex polygons non convex polygons found, in place. Either a polygonal object or a list of polygonal objects.

HandleNormals: TRUE to also handle normals of vertices.

Returns: void

Description: Routine to test all polygons in a given object for convexity, and split non convex ones, in place. This function will introduce new vertices to the split polygons.
See also: GMConvertPolysToTriangles, GMConvexPolyObjectN, GMIsConvexPolygon, , SplitNonConvexPoly,

4.2.88 GMConvexPolyObjectN (convex.c:233)

convexity

convex polygon

```
IPObjectStruct *GMConvexPolyObjectN(const IPObjectStruct *PObj,
                                    int HandleNormals)
```

PObj: To test for convexity of its polygons.

HandleNormals: TRUE to also handle normals of vertices.

Returns: A duplicate of PObj, but with convex polygons only.

Description: Routine to test all polygons in a given object for convexity, and split non convex ones, non destructively - the original object is not modified. This function will introduce new vertices to the split polygons.
See also: GMConvertPolysToTriangles, GMConvexPolyObject, GMConvexPolygon, SplitNonConvexPoly,

4.2.89 GMConvexRaysToVertices (convex.c:172)

```
int GMConvexRaysToVertices(int RaysToVertices)
```

RaysToVertices: TRUE for rays to vertices, FALSE bisector rays.

Returns: Previous state of raay casting.

Description: If TRUE, ray will be fired to vertices. If FALSE, ray as angle bisectors will be used.
See also: GMConvexPolyObjectN, GMConvertPolysToTriangles, ConvexPolyObject, , nvexPolygon, SplitNonConvexPoly, GMConvexPolyNormals,

4.2.90 GMCoplanar4Pts (geom_bsc.c:479)

coplanarity

```
int GMCoplanar4Pts(const IrtPtType Pt1,
                   const IrtPtType Pt2,
                   const IrtPtType Pt3,
                   const IrtPtType Pt4)
```

Pt1, Pt2, Pt3, Pt4: Four points to verify for coplanarity.

Returns: TRUE if coplanar, FALSE otherwise.

Description: Verifies the coplanarity of four points.
See also: GMCollinear3Pts,

4.2.91 GMDistLineLine (geom_bsc.c:1393)

line line distance

```
IrtRType GMDistLineLine(const IrtPtType Pl1,
                        const IrtPtType Vl1,
                        const IrtPtType Pl2,
                        const IrtPtType Vl2)
```

Pl1, Vl1: Position and direction defining the first line.

Pl2, Vl2: Position and direction defining the second line.

Returns: Distance between the two lines.

Description: Routine to find the distance between two lines (Pli, Vli) , i = 1, 2.

4.2.92 GMDistPointLine (geom_bsc.c:910)

point line distance

```
IrtRType GMDistPointLine(const IrtPtType Point,
                         const IrtPtType Pl,
                         const IrtPtType Vl)
```

Point: To find the distance to on the line.

Pl, Vl: Position and direction that defines the line.

Returns: The computed distance.

Description: Routine to compute the disstance between a 3d point and a 3d line. The line is prescribed using a point on it (Pl) and vector (Vl).
See also: MvarDistPointLine,

4.2.93 GMDistPointPlane (geom_bsc.c:947)

point plane distance

```
IrtRType GMDistPointPlane(const IrtPtType Point, const IrtPlnType Plane)
```

Point: To find the distance to on the plane.

Plane: To find the distance to on the point.

Returns: The computed distance.

Description: Routine to compute the distance between a Point and a Plane. The Plane is prescribed using its four coefficients : Ax + By + Cz + D = 0 given as four elements vector.

4.2.94 GMDistPointPoint (geom_bsc.c:586)

point point distance

```
IrtRType GMDistPointPoint(const IrtPtType P1, const IrtPtType P2)
```

P1, P2: Two points to compute the distance between.

Returns: Computed distance.

Description: Routine to compute the distance between two 3d points.

4.2.95 GMDistPolyPoly (geom_bsc.c:1607)

```
IrtRType GMDistPolyPoly(const IPPolygonStruct *Pl1,
                        const IPPolygonStruct *Pl2,
                        IPVertexStruct **V1,
                        IPVertexStruct **V2,
                        int TagIgnoreV)
```

Pl1, Pl2: Two polys we seek the closest vertices of the two.

V1, V2: Two vertices on Pl1 and Pl2, respectively that are closest.

TagIgnoreV: If non zero, vertices that own this tag are ignored.

Returns: Distance between V1 and V2.

Description: Finds the closest vertices of Pl1 and Pl2.
See also: GMDistPolyPt,

4.2.96 GMDistPolyPt (geom_bsc.c:1438)

```
IrtRType GMDistPolyPt(const IPPolygonStruct *Pl,
                      IrtPtType Pt,
                      const IPVertexStruct **ExtremeV,
                      int MaxDist)
```

Pl: Poly to examine its extreme distance from point Pt.

Pt: Point to examine its extreme distance from poly Pl.

ExtremeV: Will be set to the vertex at the extreme distance. If NULL it is ignored.

MaxDist: TRUE to compute the maximum distance, FALSE for the minimum.

Returns: Extreme distance, -1.0 if error.

Description: Computes the extreme distance between point Pt and polygons/lines Poly. The extreme distance is computed on the vertices of Pl only.
See also: GMDistPolyPoly, GMDistPolyPt2, UserMinDistPointPolylineList,

4.2.97 GMDistPolyPt2 (geom_bsc.c:1507)

```
IrtRType GMDistPolyPt2(const IPPolygonStruct *Pl,
                       int IsPolyline,
                       IrtPtType Pt,
                       IPVertexStruct **ExtremeV,
                       int MaxDist)
```

Pl: Poly to examine its extreme distance from point Pt.

IsPolyline: TRUE if Pl holds polylines, FALSE if polygons.

Pt: Point to examine its extreme distance from poly Pl.

ExtremeV: Will be set to the location at the vertex with the extreme. If happens between this and next vertex, will be set to this and this vertex with have a "Param" real attribute with the blend factor in [0, 1]. Will be allocated dynamically.

MaxDist: TRUE to compute the maximum distance, FALSE for the minimum.

Returns: Extreme distance, -1.0 if error.

Description: Computes the extreme distance between point Pt and polygons/lines Poly. The extreme distance is computed on the vertices and edges of Pl only.
See also: GMDistPolyPoly, GMDistPolyPt, UserMinDistPointPolylineList,

4.2.98 GMEvalWeightsVFromPl (intrnrml.c:250)

barycentric coordinates

```
int GMEvalWeightsVFromPl(const IrtRType *Coord,
                         const IPPolygonStruct *Pl,
                         IrtRType *Wgt)
```

Coord: Of vertex that its weights we seek.

Pl: Polygon including V.

Wgt: Vector to update with two weights. Must be of length larger or equal to the number of vertices in Pl.

Returns: TRUE if successful, FALSE if failed (V outside Pl).

Description: Compues blending weights for a vertex inside a polygon. Computes the barycentric coordinates of the triangle in Pl, V is in.
See also: GMUpdateVerticesByInterp, GMInterpVrtxNrmlBetweenTwo, , GMInterpVrtxNrmlBetweenTwo2, GMInterpVrtxNrmlFromPl, , GMInterpVrtxRGBBetweenTwo, GMInterpVrtxRGBFromPl, , GMInterpVrtxUVBetweenTwo, GMInterpVrtxUVFromPl,

4.2.99 GMExecuteAnimationEvalMat (animate.c:796)

```
IrtRType GMExecuteAnimationEvalMat(IPObjectStruct *AnimationP,
                                   IrtRType Time,
                                   IrtHmgnMatType ObjMat)
```

AnimationP: A (list of) animation curve(s) to evaluate at time t.

Time: Time of animation.

ObjMat: A matrix to chain the new animation matrices into.

Returns: Positive if visible (between zero and one hints on opacity), 0.0 otherwise, -1.0 if no visible curve found.

Description: Auxiliary function of ExecuteAnimation. Processes a linked list of objects.

4.2.100 GMFilterInteriorVertices (poly_cln.c:593)

```
IPVertexStruct *GMFilterInteriorVertices(IPVertexStruct *VHead,
                                         IrtRType MinTol,
                                         int n)
```

VHead: Pointer to head of NULL terminated list of vertices.

MinTol: Vertices that the inner product of previous edge direction and next edge direction is more than MinTol are purged.

n: Number of interior vertices to keep.

Returns: Similar list modified in place with only n interior vertices.

Description: Filters out interior vertices to upto n interior vertices, in place. Computes the angle between adjacent edges and purge the almost collinear ones until we have n interior vertices.
See also: GMCleanUpPolylineList, GMCleanUpPolygonList, GMVrtxListToCircOrLin,

4.2.101 GMFindLinComb2Vecs (geom_bsc.c:626)

```
int GMFindLinComb2Vecs(const IrtVecType V1,
                       const IrtVecType V2,
                       const IrtVecType V,
                       IrtRType w[2])
```

V1, V2: The two vectors that span the plane containing V.

V: To compute lin. comb. "w[0] * V1 + w[1] * V2" for.

w: The two scalar coefficients to be computed.

Returns: TRUE if successful, FALSE otherwise.

Description: Computes the linear combination of V1 and V2 that yields V. It is assumed that the three vectors are coplanar.

4.2.102 GMFindPtInsidePoly (polysmth.c:759)

```
int GMFindPtInsidePoly(const IPPolygonStruct *Pl, IrtPtType PtInside)
```

Pl: Polygon to find a point inside.

PtInside: Computed interior point.

Returns: TRUE if kernel is not empty, FALSE otherwise.

Description: Computes a point inside the given polygon, in the XY plane.

4.2.103 GMFindPtInsidePolyKernel (polysmth.c:819)

```
int GMFindPtInsidePolyKernel(const IPVertexStruct *VE, IrtPtType KrnlPt)
```

VE: Cyclic list of vertices of the Polygon.

KrnlPt: Computed interior kernel point.

Returns: TRUE if kernel is not empty, FALSE otherwise.

Description: Computes a point inside the kernel of the given polygon, if has any.

4.2.104 GMFindThirdPointInTriangle (poly_cln.c:737)

```
IPVertexStruct *GMFindThirdPointInTriangle(const IPPolygonStruct *Pl,
                                           const IPVertexStruct *V,
                                           const IPVertexStruct *VNext)
```

Pl: Triangle containing edge (V, VNext).

V, VNext: Two given points in triangle.

Returns: Pointer to the third vertex, or NULL if error.

Description: Given two points in triangle, find the third (other) point (vertex) in the triangle.

4.2.105 GMFindUnConvexPolygonNormal (polysmth.c:927)

```
void GMFindUnConvexPolygonNormal(const IPVertexStruct *VL, IrtVecType Nrml)
```

VL: List of vertices of the Polygon.

Nrml: Computed normal of the polygon. Not normalized.

Returns: void

Description: Finds the normal of polygon that can be non-convex.

4.2.106 GMFitData (fit1pts.c:83)

```
IrtRType GMFitData(IrtRType** PointData,
                   unsigned int NumberOfPointsToFit,
                   GMFittingModelType FittingModel,
                   IrtRType ModelExtParams[],
                   IrtRType Tolerance)
```

PointData: List of data points.

NumberOfPointsToFit: The length of the list of points.

FittingModel: An enumerator indicating which shape type to fit.

ModelExtParams: The resulting external params of the shape.

Tolerance: If the error is smaller then Tolerance return.

Returns: The avereage squared error.

Description: This function finds the best model params in the least-squares sense.
See also:

4.2.107 GMFitDataWithOutliers (fit1pts.c:394)

Outliers

```
IrtRType GMFitDataWithOutliers(IrtRType **PointData,
                               unsigned int NumberOfPointsToFit,
                               GMFittingModelType FittingModel,
                               IrtRType ModelExtParams[],
                               IrtRType Tolerance,
                               unsigned int NumOfChecks)
```

PointData: List of data points.

NumberOfPointsToFit: The length of the list of points.

FittingModel: An enumerator indicating which shape type to fit.

ModelExtParams: The resulting params of the shape.

Tolerance: If the error is smaller then Tolerance return.

NumOfChecks: The number of attempts to calculate best sigma.

Returns: The median squared error.

Description: This function finds the best model params in the minimal median least squares sense.
See also: GMFitData,

4.2.108 GMFitEstimateRotationAxis (fit1pts.c:810)

```
IrtRType GMFitEstimateRotationAxis(IrtPtType *PointsOnObject,
                                   IrtVecType *Normals,
                                   unsigned int NumberOfPoints,
                                   IrtPtType PointOnRotationAxis,
                                   IrtVecType RotationAxisDirection)
```

PointsOnObject: Points on the surface.

Normals: Corresponding normals.

NumberOfPoints: The number of points/normals.

PointOnRotationAxis: The result.

RotationAxisDirection: The result.

Returns: The error.

Description: This function estimates a rotation axis of a surface of revolution.
See also:

4.2.109 GMFitObjectWithOutliers (fit1pts.c:666)

```
IrtRType GMFitObjectWithOutliers(IPPolygonStruct *PPoly,
                                 GMFittingModelType FittingModel,
                                 IrtRType ModelExtParams[],
                                 IrtRType Tolerance,
                                 unsigned int NumOfChecks)
```

PPoly: Pointer the object to estimate.

FittingModel: An enumerator indicating which shape type to fit.

ModelExtParams: The resulting params of the shape. if FittingModel is GM_FIT_PLANE - A, B, C, D of plane equation. GM_FIT_SPHERE - Xcntr, Ycntr, Zcntr, Radius. GM_FIT_CYLINDER - Xcntr, Ycntr, Zcntr, Xdir, Ydir, Zdir, Radius. GM_FIT_CIRCLE - Xcntr, Ycntr, Radius. GM_FIT_CONE - Xapex, Yapex, Zapex, apex semi angle, Xdir, Ydir, Zdir. GM_FIT_TORUS - Xpnt, Ypnt, Zpnt, DiscRad, ExtRad, Xdir, Ydir, Zdir.

Tolerance: If the error is smaller than Tolerance return.

NumOfChecks: The number of attempts to calculate best sigma, 100 is a good start.

Returns: The median squared error.

Description: This function finds the best model params in the minimal median least squares sense. Warning: This function is NOT thread-safe.
See also: GMFitDataWithOutliers,

4.2.110 GMFixNormalsOfPolyModel (intrnrml.c:1001)

```
void GMFixNormalsOfPolyModel(IPPolygonStruct *PlList, int TrustFixedPt)
```

PlList: Polygonal object to correct normals.

TrustFixedPt: 0 to trust the vertices' normal, 1 to trust the orientation of polygons' normals, 2 to reorient the polygons so all plane normals point outside or all inside (based on first poly).

Returns: void

Description: Fix orientation discrepancy between polygon normals and vertices normals.
See also: GMBlendNormalsToVertices, GMFixOrientationOfPolyModel,

4.2.111 GMFixOrientationOfPolyModel (intrnrml.c:880)

```
void GMFixOrientationOfPolyModel(IPPolygonStruct *Pls)
```

Pls: Polygons to reorient them all based on first polygon.

Returns: void

Description: Using computed adjacency information, propagate the orientation of first polygon in the given poly object until all polygons are processed. Disjoint poly-meshes will be marked with an _OrientDisjoint attribute on the first polygon of each disjoint part.
See also: ,

4.2.112 GMFixPolyNormals (intrnrml.c:1068)

```
void GMFixPolyNormals(IPObjectStruct *PObj, int TrustFixPt)
```

PObj: Polygonal object to correct normals, in place.

TrustFixPt: 0 to trust the vertices' normal, 1 to trust the orientation of polygons' normals, 2 to reorient the polygons so all plane normals point outside or all inside (based on first poly). 3 same as 2 but splits disjoint parts in the input object into different objects.

Returns: void

Description: Correct the (polygonal and/or vertices) normals of a polygonal object via adjacency propagations. As a side effect also allow the split of a polygonal models with disjoint parts, into the disjoint parts. That is, the connected components in the input will be organized as a list object, in place.
See also: GMBlendNormalsToVertices, GMFixNormalsOfPolyModel,

4.2.113 GMGenMatObjectRotVec (geomat3d.c:175)

rotation
transformations

```
IPObjectStruct *GMGenMatObjectRotVec(const IrtVecType Vec,
                                     const IrtRType *Degree)
```

Vec: Vector to rotate along its axis.

Degree: Amount of rotation, in degrees.

Returns: A matrix object.

Description: Routine to generate rotation object around the vector Vec in Degree degs:
See also: GMGenMatrixZ2Dir, GMGenMatrixZ2Dir2, GMGenMatObjectZ2Dir2, GMGenMatrixRotVec,

4.2.114 GMGenMatObjectRotX (geomat3d.c:46)

rotation
transformations

```
IPObjectStruct *GMGenMatObjectRotX(const IrtRType *Degree)
```

Degree: Amount of rotation, in degrees.

Returns: A matrix object.

Description: Routine to generate rotation object around the X axis in Degree degrees:

4.2.115 GMGenMatObjectRotY (geomat3d.c:68)

rotation

transformations

```
IPObjectStruct *GMGenMatObjectRotY(const IrtRType *Degree)
```

Degree: Amount of rotation, in degrees.

Returns: A matrix object.

Description: Routine to generate rotation object around the Y axis in Degree degrees:

4.2.116 GMGenMatObjectRotZ (geomat3d.c:90)

rotation

transformations

```
IPObjectStruct *GMGenMatObjectRotZ(const IrtRType *Degree)
```

Degree: Amount of rotation, in degrees.

Returns: A matrix object.

Description: Routine to generate rotation object around the Z axis in Degree degrees:

4.2.117 GMGenMatObjectScale (geomat3d.c:226)

scaling

transformations

```
IPObjectStruct *GMGenMatObjectScale(const IrtVecType Vec)
```

Vec: Amount of scaling, in X, Y, and Z.

Returns: A matrix object.

Description: Routine to generate a scaling object.
See also: MatGenMatScale,

4.2.118 GMGenMatObjectTrans (geomat3d.c:200)

translation

transformations

```
IPObjectStruct *GMGenMatObjectTrans(const IrtVecType Vec)
```

Vec: Amount of translation, in X, Y, and Z.

Returns: A matrix object.

Description: Routine to generate a translation object.
See also: MatGenMatTrans,

4.2.119 GMGenMatObjectV2V (geomat3d.c:1286)

rotation

transformations

```
IPObjectStruct *GMGenMatObjectV2V(const IrtVecType V1, const IrtVecType V2)
```

V1, V2: To compute the rotation from (V1) to (V2).

Returns: A matrix object.

Description: Generates a transformation matrix that rotates V1 to V2.
See also: GMGenMatrixZ2Dir, GMGenMatrixZ2Dir2, GMGenMatObjectZ2Dir2, GMGenMatrixRotVec, GMGenMatrixRotV2V,

4.2.120 GMGenMatObjectZ2Dir (geomat3d.c:116)

rotation

transformations

```
IPObjectStruct *GMGenMatObjectZ2Dir(const IrtVecType Dir)
```

Dir: Vector to rotate Z axis to it.

Returns: A matrix object.

Description: Routine to generate rotation object to rotate Z to Dir:
See also: GMGenMatrixZ2Dir, GMGenMatrixZ2Dir2, GMGenMatObjectZ2Dir2, GMGenMatrixRotV2V, GMGenMatrixRotVec,

4.2.121 GMGenMatObjectZ2Dir2 (geomat3d.c:147)

rotation

transformations

```
IPObjectStruct *GMGenMatObjectZ2Dir2(const IrtVecType Dir,
                                     const IrtVecType Dir2)
```

Dir: Vector to rotate Z axis to it.

Dir2: Vector to rotate X axis to it.

Returns: A matrix object.

Description: Routine to generate rotation object around the vector Dir in Degree degs:
See also: GMGenMatrixZ2Dir2, E, GMGenMatrixZ2Dir2, GMGenMatrixZ2Dir, GMGenMatObjectZ2Dir, GMGenMatrixRotV2V, GMGenMatrixRotVec,

4.2.122 GMGenMatrix3Pts2EqltrlTri (geomat3d.c:1411)

transformations

shear

```
IPObjectStruct *GMGenMatrix3Pts2EqltrlTri(const IrtPtType Pt1,
                                          const IrtPtType Pt2,
                                          const IrtPtType Pt3)
```

Pt1, Pt2, Pt3: The three points to compute the mapping for.

Returns: A matrix object.

Description: Compute the linear transform that maps the given planar triangle Pt1Pt2Pt3 to an equilateral triangle around the origin so that edge Pt1Pt2 is horizontal and remains of the same size.
See also: GM3Pts2EqltrlTriMat,

4.2.123 GMGenMatrix4Pts2Affine4Pts (geomat3d.c:1243)

transformations

rotation

```
int GMGenMatrix4Pts2Affine4Pts(const IrtPtType P0,
                               const IrtPtType P1,
                               const IrtPtType P2,
                               const IrtPtType P3,
                               const IrtPtType Q0,
                               const IrtPtType Q1,
                               const IrtPtType Q2,
                               const IrtPtType Q3,
                               IrtHmgnMatType Trans)
```

P0, P1, P2, P3: The four source points.

Q0, Q1, Q2, Q3: The four image points.

Trans: The computed affine transform. This is a return argument.

Returns: TRUE if four source points are non-coplanar, FALSE otherwise.

Description: Computes an affine transform from R^3 to R^3 using four points in R^3 and their images under the transform. Such a transform is well defined if and only if the four source points are not coplanar.
See also: GMGenMatrixZ2Dir, GMGenMatrixZ2Dir2, GMGenMatObjectZ2Dir2, , GMGenMatrixRotVec, GMGenMatrixRotV2V,

4.2.124 GMGenMatrixRotV2V (geomat3d.c:1171)

transformations

rotation

```
void GMGenMatrixRotV2V(IrtHmgnMatType Mat,
                       const IrtVecType V1,
                       const IrtVecType V2)
```

Mat: To place the computed transformation.

V1, V2: Vector to rotate from (V1) to (V2).

Returns: void

Description: Generates a transformation matrix that rotates V1 to V2.
See also: GMGenMatrixZ2Dir, GMGenMatrixZ2Dir2, GMGenMatObjectZ2Dir2, , GMGenMatrixRotVec, GMGenMatrix4Pts2Affine4Pts,

4.2.125 GMGenMatrixRotVec (geomat3d.c:1443)

transformations

rotation

```
void GMGenMatrixRotVec(IrtHmgnMatType Mat,
                       const IrtVecType Vec,
                       IrtRType Degrees)
```

Mat: To place the computed transformation.

Vec: Vector to rotate along its axis.

Degrees: Amount of rotation, in degrees.

Returns: void

Description: Generates a transformation matrix that rotates the object around Vec, Angle degrees.
See also: GMGenMatrixZ2Dir, GMGenMatrixZ2Dir2, GMGenMatObjectZ2Dir2, GMGenMatrixRotV2V,

4.2.126 GMGenMatrixX2Dir (geomat3d.c:971)

rotation

transformations

```
void GMGenMatrixX2Dir(IrtHmgnMatType Mat, const IrtVecType Dir)
```

Mat: To place the computed transformation.

Dir: Direction to take X axis to.

Returns: void

Description: Routine to generate rotation matrix to rotate X to Dir:

4.2.127 GMGenMatrixY2Dir (geomat3d.c:994)

rotation

transformations

```
void GMGenMatrixY2Dir(IrtHmgnMatType Mat, const IrtVecType Dir)
```

Mat: To place the computed transformation.

Dir: Direction to take Y axis to.

Returns: void

Description: Routine to generate rotation matrix to rotate Y to Dir:

4.2.128 GMGenMatrixZ2Dir (geomat3d.c:1017)

transformations

rotation

```
void GMGenMatrixZ2Dir(IrtHmgnMatType Mat, const IrtVecType Dir)
```

Mat: To place the computed transformation.

Dir: Direction to take Z axis to.

Returns: void

Description: Same as GMGenTransMatrixZ2Dir but with no scaling and/or translation.

4.2.129 GMGenMatrixZ2Dir2 (geomat3d.c:1099)

transformations

rotation

```
void GMGenMatrixZ2Dir2(IrtHmgnMatType Mat,
                       const IrtVecType Dir,
                       const IrtVecType Dir2)
```

Mat: To place the computed transformation.

Dir: Direction to take Z axis to.

Dir2: Direction to take X axis to.

Returns: void

Description: Same as GMGenTransMatrixZ2Dir2 but with no scaling and/or translation.

4.2.130 GMGenPolyline2Vrtx (polyprop.c:397)

```
IPPolygonStruct *GMGenPolyline2Vrtx(IrtVecType V1,
                                    IrtVecType V2,
                                    IPPolygonStruct *Pnext)
```

V1, V2: Two vertices of the constructed polyline.

Pnext: Next is chain of polylines, in linked list.

Returns: The constructed polyline.

Description: Routine to create a polyline out of a list of 2 vertices V1/2. No test is made to make sure the 2 points are not the same... The points are placed in order.
See also: PrimGenPolygon3Vrtx, PrimGenPolygon4Vrtx,

4.2.131 GMGenProjectionMat (geomat3d.c:1479)

```
void GMGenProjectionMat(const IrtPlnType ProjPlane,
                        const IrtRType EyePos[4],
                        IrtHmgnMatType Mat)
```

ProjPlane: The plane to project the objects onto.

EyePos: The position of the eye.

Mat: Matrix to update.

Returns: void

Description: Constructs a matrix that projects 3D objects to the Projection Plane ProjPlane, having the eye at EyePos. This solution is derived by solving for the intersection point of the line through the eye and the projected point and the given plane.
See also: GMGenReflectionMat,

4.2.132 GMGenReflectionMat (geomat3d.c:1527)

```
void GMGenReflectionMat(const IrtPlnType ReflectPlane, IrtHmgnMatType Mat)
```

ReflectPlane: The plane to computed a reflection matrix for.

Mat: Matrix to update.

Returns: void

Description: Constructs a matrix that reflects 3D objects based upon the prescribed reflection plane, ReflectPlane. Uses a Householder reflection to immediately construct the refelction matrix (around the origin).
See also: GMGenProjectionMat, GMGenMatrixZ2Dir,

4.2.133 GMGenRotateMatrix (convex.c:126)

transformations

```
void GMGenRotateMatrix(IrtHmgnMatType Mat, const IrtVecType Dir)
```

Mat: To place the constructed homogeneous transformation.

Dir: To derive a transformation such that Dir goes to Z axis.

Returns: void

Description: Routine to prepare a transformation martix to rotate such that Dir is parallel to the Z axes. Used by the convex decomposition to rotate the polygons to be XY plane parallel. Algorithm: form a 4 by 4 matrix from Dir as follows:

```
                | Tx  Ty  Tz  0 |   A transformation which takes the coord
                | Bx  By  Bz  0 |  system into T, N & B as required.
 [X  Y  Z  1] * | Nx  Ny  Nz  0 |
                | 0   0   0   1 |
```

N is exactly Dir, but we got freedom on T & B which must be on a plane perpendicular to N and perpendicular between them but thats all! T is therefore selected using this (heuristic ?) algorithm: Let P be the axis of which the absolute N coefficient is the smallest. Let B be (N cross P) and T be (B cross N).

4.2.134 GMGenTransMatrixZ2Dir (geomat3d.c:923)

transformations

rotation

```
void GMGenTransMatrixZ2Dir(IrtHmgnMatType Mat,
                           const IrtVecType Trans,
                           const IrtVecType Dir,
                           IrtRType Scale)
```

Mat: To place the computed transformation.

Trans: Translation factor.

Dir: Direction to take Z axis to.

Scale: Scaling factor.

Returns: void

Description: Routine to prepare a transformation martix to do the following (in this order): scale by Scale, rotate such that the Z axis is in direction Dir and then translate by Trans. Algorithm: given the Trans vector, it forms the 4th line of Mat. Dir is used to form the second line (the first 3 lines set the rotation), and finally Scale is used to scale first 3 lines/columns to the needed scale:

```
                |  Tx  Ty  Tz  0 |   A transformation which takes the coord
                |  Bx  By  Bz  0 |  system into T, N & B as required and
 [X  Y  Z  1] * |  Nx  Ny  Nz  0 |  then translate it to C. T, N, B are
                |  Cx  Cy  Cz  1 |  scaled by Scale.
```

N is exactly Dir (unit vec) but we got freedom on T & B which must be on a plane perpendicular to N and perpendicular between them but thats all! T is therefore selected using this (heuristic ?) algorithm: Let P be the axis of which the absolute N coefficient is the smallest. Let B be (N cross P) and T be (B cross N).

4.2.135 GMGenTransMatrixZ2Dir2 (geomat3d.c:1056)

transformations

rotation

```
void GMGenTransMatrixZ2Dir2(IrtHmgnMatType Mat,
                            const IrtVecType Trans,
                            const IrtVecType Dir,
                            const IrtVecType Dir2,
                            IrtRType Scale)
```

Mat: To place the computed transformation.

Trans: Translation factor.

Dir: Direction to take Z axis to.

Dir2: Direction to take X axis to.

Scale: Scaling factor.

Returns: void

Description: Routine to prepare a transformation matrix to do the following (in this order): scale by Scale, rotate such that the Z axis is in direction Dir and X axis is direction Dir2 and then translate by Trans. Algorithm: given the Trans vector, it forms the 4th line of Mat. Dir is used to form the second line (the first 3 lines set the rotation), and finally Scale is used to scale first 3 lines/columns to the needed scale:

```
                |  Tx  Ty  Tz  0 |   A transformation which takes the coord
                |  Bx  By  Bz  0 |  system into T, N & B as required and
 [X  Y  Z  1] * |  Nx  Ny  Nz  0 |  then translate it to C. T, N, B are
                |  Cx  Cy  Cz  1 |  scaled by Scale.
```

N is exactly Dir (unit vec) and T is exactly Dir2.

4.2.136 GMGenUVValsForPolys (poly_pts.c:1897)

```
void GMGenUVValsForPolys(const IPObjectStruct *PObj,
                         IrtRType UTextureRepeat,
                         IrtRType VTextureRepeat,
                         IrtRType WTextureRepeat,
                         int HasXYZScale)
```

PObj: A polygonal object to update UV vals for.

UTextureRepeat, VTextureRepeat, WTextureRepeat: Repeat texture factors.

HasXYZScale: If TRUE, WTextureRepeat is also valid - use XYZ coords.

Returns: void

Description: Generates UV values for polygonal geometry, based on the XY(Z) coordinates of the geometry. Will NOT overwrite existing "uvvals", if any.
See also: GMAffineTransUVVals,

4.2.137 GMGetGeneralAlignedFrustumPtsIntersection (box2BVH.c:132)

```
CagdBType GMGetGeneralAlignedFrustumPtsIntersection(
                                              const IrtRType Center[3],
                                              IrtRType SizeX,
                                              IrtRType SizeY,
                                              const IrtRType XAxis[3],
                                              const IrtRType HeightAxis[3],
                                              IrtRType Height,
                                              IrtRType XAngle,
                                              IrtRType YAngle,
                                              const IrtRType **Pts,
                                              int NumPts,
                                              CagdBType RationalPts,
                                              IrtRType Tol)
```

Center: The center of the narrow base of the frustum.

SizeX, SizeY: The size of the narrow base of the frustum.

XAxis: The X direction of the bases of the frustum.

HeightAxis: The height (Z) axis of the frustum.

Height: The height of the frustum.

XAngle, YAngle: The angles of the frustum in the X and Y directions, respectively.

Pts: An array of points to be tested.

NumPts: The number of points in the array.

RationalPts: A flag indicating if the points are in projective (rational) coordinates.

Tol: The allowed tolerance.

Returns: Non-zero if the frustum and box overlap.

Description: Checks if a set of points is outside of a frustum, within a given tolerance. The frustum is constructed from its parameters. The points are treated as a solid convex body. The frustum can have a general alignment.
See also: GMGetGeneralAlignedFrustumPtsIntersection2,

4.2.138 GMGetGeneralAlignedFrustumPtsIntersection2 (box2BVH.c:42)

```
IrtBType GMGetGeneralAlignedFrustumPtsIntersection2(
                                    const GMGeneralFrustumInfoStruct *Frustum,
                                    const IrtRType **Pts,
                                    int NumPts,
                                    IrtBType RationalPts,
                                    IrtRType Tolerance)
```

Frustum: The frustum info.

Pts: An array of points to be tested.

NumPts: The number of points in the array.

RationalPts: A flag indicating if the points are in projective (rational) coordinates.

Tolerance: The allowed tolerance.

Returns: Non-zero if the frustum and box overlap.

Description: Checks if a set of points is outside of a frustum, within a given tolerance. The points are treated as a solid convex body. The frustum can have a general alignment.
See also: GMGetGeneralAlignedFrustumPtsIntersection,

4.2.139 GMGetMatTransPortion (geomat3d.c:253)

translation

transformations

```
IPObjectStruct *GMGetMatTransPortion(const IPObjectStruct *MatObj,
                                     int TransPortion)
```

MatObj: To operate on.

TransPortion: TRUE to extract translational portion out of Mat, FALSE to dump the translational portion from Mat.

Returns: A matrix object hold either the translational portion of Mat or anything but the translational part.

Description: Routine to extract the translational part of a matrix or dump it.

4.2.140 GMGetMaxNumVrtcsPoly (poly_pts.c:107)

```
int GMGetMaxNumVrtcsPoly(IPObjectStruct *PolyObj)
```

PolyObj: Polygonal object to look for its largest poly.

Returns: Maximal number of vertices found in one poly.

Description: Go over all polygons in given object and return the maximal number of vertices found in a single poly.

4.2.141 GMGetObjNumOfPolys (poly_pts.c:138)

```
int GMGetObjNumOfPolys(IPObjectStruct *PObj)
```

PObj: Object to accumulate number of polygons in all.

Returns: Maximal number of polygons found in PObj.

Description: Go over all objects and accumulate the number of polygons in all.
See also: IPPolyListLen,

4.2.142 GMIdentifyTJunctions (poly_pts.c:2201)

```
int GMIdentifyTJunctions(IPObjectStruct *PObj,
                         GMIdentifyTJunctionFuncType TJuncCB,
                         IrtRType Eps)
```

PObj: Polygonal mesh to search for T junctions in.

TJuncCB: Call back function to invoke with every detected T junction. If E0 = (V0, V0 -> Pnext) is adjacent to both E1 = (V1, V1 -> Pnext) and E2 = (V2, V2 -> Pnext) so E2 is a T junction vertex on edge E0, the TJuncCB will be invoked as TJuncCB(V0, V1, V2, Pl0, Pl1, Pl2).

Eps: Tolerance of testing.

Returns: Number of detected T junctions, zero if none.

Description: Function to search for T junctions in given polygonal mesh Polys. The call back function TJunCB is invoked on every such T junction. This functions offers a naive and non-optimal (N^2) solution.
See also: GMIsTJunction,

4.2.143 GMInterpVrtxNrmlBetweenTwo (intrnrml.c:303)

normals

```
void GMInterpVrtxNrmlBetweenTwo(IPVertexStruct *V,
                                const IPVertexStruct *V1,
                                const IPVertexStruct *V2)
```

V: Vertex that its normal is to be updated.

V1, V2: Edge V is assumed to be on so that the two normals of V1 and V2 can be blended to form the normal of V.

Returns: void

Description: Update Normal of the middle vertex V, assumed to be between V1 and V2.
See also: GMUpdateVerticesByInterp, GMInterpVrtxNrmlBetweenTwo2, , GMInterpVrtxNrmlFromPl, GMInterpVrtxRGBBetweenTwo, GMInterpVrtxRGBFromPl, , GMInterpVrtxUVBetweenTwo, GMInterpVrtxUVFromPl,

4.2.144 GMInterpVrtxNrmlBetweenTwo2 (intrnrml.c:354)

normals

```
void GMInterpVrtxNrmlBetweenTwo2(IrtPtType Pt,
                                 IrtVecType Normal,
                                 const IPVertexStruct *V1,
                                 const IPVertexStruct *V2,
                                 int Normalize)
```

Pt: Middle position at which a normal is to be computed.

Normal: Where resulting vector is to be placed.

V1, V2: Edge V is assumed to be on so that the two normals of V1 and V2 can be blended to form the normal of V.

Normalize: TRUE to return a unit size vector.

Returns: void

Description: Update normal of middle position Pt, assumed to be between V1 and V2.
See also: GMUpdateVerticesByInterp, GMInterpVrtxNrmlBetweenTwo, , GMInterpVrtxNrmlFromPl, GMInterpVrtxRGBBetweenTwo, GMInterpVrtxRGBFromPl, , GMInterpVrtxUVBetweenTwo, GMInterpVrtxUVFromPl,

4.2.145 GMInterpVrtxNrmlFromPl (intrnrml.c:402)

normals

```
int GMInterpVrtxNrmlFromPl(IPVertexStruct *Vrtx, const IPPolygonStruct *Pl)
```

Vrtx: Vertex that its normal is to be updated.

Pl: Polygon surrounding V to interpolate normal from.

Returns: TRUE if point is inside polygon, FALSE otherwise.

Description: Update Normal of vertex V, based on surrounding polygon Pl.
See also: GMUpdateVerticesByInterp, GMInterpVrtxNrmlBetweenTwo, , GMInterpVrtxNrmlBetweenTwo2, GMInterpVrtxRGBBetweenTwo, , GMInterpVrtxRGBFromPl, GMInterpVrtxUVBetweenTwo, GMInterpVrtxUVFromPl,

4.2.146 GMInterpVrtxRGBBetweenTwo (intrnrml.c:456)

rgb color

```
int GMInterpVrtxRGBBetweenTwo(IPVertexStruct *V,
                              const IPVertexStruct *V1,
                              const IPVertexStruct *V2)
```

V: Vertex that its rgb color is to be updated.

V1, V2: Edge V is assumed to be on so that the two rgb colors of V1 and V2 can be blended to form the rgb color of V.

Returns: TRUE if successful, FALSE if failed.

Description: Update RGB of the middle vertex V, assumed to be between V1 and V2.
See also: GMUpdateVerticesByInterp, GMInterpVrtxNrmlBetweenTwo, , GMInterpVrtxNrmlBetweenTwo2, GMInterpVrtxNrmlFromPl, , GMInterpVrtxRGBFromPl, GMInterpVrtxUVBetweenTwo, GMInterpVrtxUVFromPl,

4.2.147 GMInterpVrtxRGBFromPl (intrnrml.c:502)

rgb color

```
int GMInterpVrtxRGBFromPl(IPVertexStruct *Vrtx, const IPPolygonStruct *Pl)
```

Vrtx: Vertex that its rgb color is to be updated.

Pl: Polygon surrounding V to interpolate rgb color from.

Returns: TRUE if point is inside polygon, FALSE otherwise.

Description: Update rgb color of vertex V, based on surrounding polygon Pl.
See also: GMUpdateVerticesByInterp, GMInterpVrtxNrmlBetweenTwo, , GMInterpVrtxNrmlBetweenTwo2, GMInterpVrtxNrmlFromPl, , GMInterpVrtxRGBBetweenTwo, GMInterpVrtxUVBetweenTwo, , GMInterpVrtxUVFromPl,

4.2.148 GMInterpVrtxUVBetweenTwo (intrnrml.c:567)

uv coordinates

```
int GMInterpVrtxUVBetweenTwo(IPVertexStruct *V,
                             const IPVertexStruct *V1,
                             const IPVertexStruct *V2)
```

V: Vertex that its UV coordinates are to be updated.

V1, V2: Edge V is assumed to be on so that the two UV coords of V1 and V2 can be blended to form the UV coords of V.

Returns: TRUE if successful, FALSE if failed.

Description: Update UV of the middle vertex V, assumed to be between V1 and V2.
See also: GMUpdateVerticesByInterp, GMInterpVrtxNrmlBetweenTwo, , GMInterpVrtxNrmlBetweenTwo2, GMInterpVrtxNrmlFromPl, , GMInterpVrtxRGBBetweenTwo, GMInterpVrtxRGBFromPl, , GMInterpVrtxUVFromPl,

4.2.149 GMInterpVrtxUVFromPl (intrnrml.c:613)

uv coordinates

```
int GMInterpVrtxUVFromPl(IPVertexStruct *Vrtx, const IPPolygonStruct *Pl)
```

Vrtx: Vertex that its UV coordinate is to be updated.

Pl: Polygon surrounding V to interpolate UV coordinate from.

Returns: TRUE if successful, FALSE if failed.

Description: Update UV coordinate of vertex V, based on surrounding polygon Pl.
See also: GMUpdateVerticesByInterp, GMInterpVrtxNrmlBetweenTwo, , GMInterpVrtxNrmlBetweenTwo2, GMInterpVrtxNrmlFromPl, , GMInterpVrtxRGBBetweenTwo, GMInterpVrtxRGBFromPl, , GMInterpVrtxUVBetweenTwo,

4.2.150 GMInverseBilinearMap (geomvals.c:441)

```
int GMInverseBilinearMap(const IrtPtType P00,
                         const IrtPtType P01,
                         const IrtPtType P10,
                         const IrtPtType P11,
                         const IrtPtType PtInt,
                         IrtPtType UV[2])
```

P00, P01, P10, P11: The four points defining the bilinear.

PtInt: A 5th point in the plane, to computes its UV values.

UV: The computed solutions (up to two solutions are possible.

Returns: 0 if error. Otherwise number of solution in UV.

Description: Computes the inverse of the bilinear mapping. Given four points, Pij, defining a bilinear mapping in the plane, and a point PtInt, inside it, computes the UV coordinates of PtInt in this bilinear mapping:

`B(u, v) = (1-u) * ((1- v) P00 + v P10) + u * ((1-v) P01 + V P11).`

Solution follows the following steps:

1. Isolate v as a function of the 5 points and u. Two equations for x and y.
2. Equate the two functions from 1, eliminating v. We have only terms of u and (1-u) or get a polynomial quadratic Bezier, once having a common denominator and examining only the numerator.
3. Solve the quadratic equation in u and then substitute to one of the two equations of 1 to get v.

4.2.151 GMIsConvexPolygon (convex.c:479)

convexity

convex polygon

```
int GMIsConvexPolygon(IPPolygonStruct *Pl)
```

Pl: To test its convexity condition. Reverses vertices in place if needed.

Returns: TRUE if convex, FALSE otherwise.

Description: Routine to test if the given polygon is convex or not. Algorithm: The polygon is convex iff the normals generated from cross products of two consecutive edges points to the same direction. Note a 5 star polygon satisfies this constraint but it is self intersecting and we assume given polygon is not self intersecting. The computed direction is also verified against the polygon's plane normal. The routine returns TRUE iff the polygon is convex. In addition the polygon CONVEX tag (see IPPolygonStruct) is also updated. If the polygon is already marked as convex, nothing is tested!
See also: GMConvertPolysToTriangles, GMConvexPolyObject, GMConvexPolyObjectN, , SplitNonConvexPoly, GMIsConvexPolygon2,

4.2.152 GMIsConvexPolygon2 (convex.c:411)

convexity

files

parser

```
int GMIsConvexPolygon2(const IPPolygonStruct *Pl)
```

Pl: To test for convexity.

Returns: TRUE if PL convex, FALSE otherwise.

Description: Routine to test if the given polygon is convex (by IRIT definition) or not. For both closed and open vertex lists. Algorithm: The polygon is convex iff the normals generated from cross products of two consecutive edges points to the same direction. The same direction is tested by a positive dot product.
See also: GMIsConvexPolygon,

4.2.153 GMIsInterLineLine2D (polysmth.c:725)

```
static int GMIsInterLineLine2D(const IrtPtType A1,
                               const IrtPtType A2,
                               const IrtPtType B1,
                               const IrtPtType B2,
                               IrtRType *t)
```

A1, A2: The two points of first line.

B1, B2: The two points of second line.

t: The blending value from B1 to B2 where the intersection has occurred. Can be NULL to ignore.

Returns: TRUE if the two lines intersect, FALSE otherwise.

Description: Check if the two lines A1A2 and B1B2 interest in the XY plane. End points intersections, up to IRIT_EPS, are ignored.
See also: GMAllIntersLinePolygon2D, GMIsInterLinePolygon2D,

4.2.154 GMIsInterLinePolygon2D (polysmth.c:680)

```
int GMIsInterLinePolygon2D(const IPVertexStruct *VS,
                           const IrtPtType V1,
                           const IrtPtType V2,
                           IrtRType *t)
```

VS: Cyclic List of vertices of the Polygon.

V1, V2: The end points of the line.

t: The blending value from V1 to V2 where the intersection has occurred. Can be NULL to ignore.

Returns: TRUE if line and polygon do interest, FALSE otherwise.

Description: Check if a line V1V2 and polygon VS interest, in 2D.
See also: GMAllIntersLinePolygon2D, GMIsInterLineLine2D,

4.2.155 GMIsPolygonPlanar (poly_cln.c:387)

```
int GMIsPolygonPlanar(const IPPolygonStruct *Pl, IrtRType Tol)
```

Pl: Polygons to examine for planairy up to tolerance Tol.

Tol: Of planarify verification.

Returns: TRUE if poly planar, FALSE if not.

Description: Examines the planarity of the given object.

4.2.156 GMIsPtInsideCirc (geom_bsc.c:3341)

```
int GMIsPtInsideCirc(const IrtRType *Point,
                     const IrtRType *Center,
                     IrtRType Radius)
```

Point: Point to test for containment in the circle in XY plane.

Center: Center of the circle to test against.

Radius: Radius of the circle to test against.

Returns: TRUE if Point is indeed inside the circle, FALSE otherwise.

Description: Tests if a point is contained in the given prescribed circle in XY plane. Points on the circle are not considered inside (open domain).
See also: GM2PointsFromLineLine, GM2PointsFromCircCirc3D, GMCircleFrom3Points, , GMCircleFrom2Pts2Tans, GM2BiTansFromCircCirc, GM2TanLinesFromCircCirc, , GM2PointsFromCircCirc, GMIsPtOnCirc,

4.2.157 GMIsPtOnCirc (geom_bsc.c:3370)

```
int GMIsPtOnCirc(const IrtRType *Point,
                 const IrtRType *Center,
                 IrtRType Radius)
```

Point: Point to test for inclusing in circle boundary in XY plane.

Center: Center of the circle to test against.

Radius: Radius of the circle to test against.

Returns: TRUE if Point is indeed on the circle, FALSE otherwise.

Description: Tests if a point is on the given prescribed circle's boundary, in the N XY plane.
See also: GM2PointsFromLineLine, GM2PointsFromCircCirc3D, GMCircleFrom3Points, , GMCircleFrom2Pts2Tans, GM2BiTansFromCircCirc, GM2TanLinesFromCircCirc, , GM2PointsFromCircCirc, GMIsPtInsideCirc,

4.2.158 GMLimitTrianglesEdgeLen (poly_pts.c:1697)

```
IPPolygonStruct *GMLimitTrianglesEdgeLen(const IPPolygonStruct *OrigPls,
                                         IrtRType MaxLen)
```

OrigPls: List of triangles.

MaxLen: Maximum allowed length of an edge of a triangle.

Returns: Splitted polygons with edges smaller/equal to MaxLen.

Description: Splits all triangles that has edge length larger than MaxLen. The output will have no edge in no triangle with length larger than MaxLen.
See also: ConvexPolyObjectN, GMConvertPolysToNGons, GMConvertPolysToTriangles,

4.2.159 GMLineFrom2Planes (geom_bsc.c:1331)

```
int GMLineFrom2Planes(const IrtPlnType Pl1,
                      const IrtPlnType Pl2,
                      IrtPtType Pt,
                      IrtVecType Dir)
```

Pl1, Pl2: Two planes to consider.

Pt, Dir: Intersection line found (if any).

Returns: TRUE if exists an intersection point, FALSE otherwise.

Description: Find the intersection line (if exists) of two planes.

4.2.160 GMLineFrom2Points (geom_bsc.c:684)

line

```
int GMLineFrom2Points(IrtLnType Line, const IrtPtType Pt1, const IrtPtType Pt2)
```

Line: To compute as Ax + By + C, such that A^2 + B^2 = 1.

Pt1, Pt2: Two points to fit a line through. Only XY coordinates are considered.

Returns: TRUE if successful, FALSE otherwise.

Description: Routine to construct a line through 2 points, in the plane. If the points are the same it returns FALSE, otherwise (successful), TRUE.

4.2.161 GMLineSweep (ln_sweep.c:45)

line sweep

line line intersections

```
void GMLineSweep(GMLsLineSegStruct **Lines)
```

Lines: To compute all intersections against each other, in the plane.

Returns: void

Description: Computes all intersections between all given lines, in the plane. The Lines segments are updated so the Inters slot holds the list of intersections with the other segments, NULL if none. Returned is a list with possibly a different order than that is given.

4.2.162 GMLoadTextFont (text.c:42)

```
int GMLoadTextFont(const char *FName)
```

FName: Name of IRIT font file to load.

Returns: TRUE if successful, FALSE otherwise.

Description: Loads the IRIT font file that is specified by the FName file name. An IRIT font file contains the geometries of the characters as a list ordered according to the ASCII table starting from 32 (space). Chars can, alternatively, be prescribed by their names in the list as ASCII???

See also: GMMakeTextGeometry,

4.2.163 GMMakeTextGeometry (text.c:100)

```
IPObjectStruct *GMMakeTextGeometry(const char *Str,
                                   const IrtVecType Spacing,
                                   const IrtRType *Scaling)
```

Str: The text to convert to geometry.

Spacing: Between individual characters, in X, Y, Z.

Scaling: Relative, with scaling of one generates unit size chars.

Returns: Geometry representing the given text.

Description: Constructs a geometry representing the given text in Str, with Spacing space between character and scaling factor of Scale. If no font is loaded, this functions also loads the default font in iritfont.itd in the directory that is prescribed by the IRIT_PATH environment variable.

See also: GMLoadTextFont,

4.2.164 GMMatFromPosDir (geomat3d.c:1129)

transformations

```
int GMMatFromPosDir(const IrtPtType Pos,
                    const IrtVecType Dir,
                    const IrtVecType UpDir,
                    IrtHmgnMatType Mat)
```

Pos: The location of the viewer.

Dir: The viewing direction.

UpDir: The direction of up. Must be different than Dir.

Mat: Matrix to update. N

Returns: TRUE if succesful, FALSE otherwise.

Description: Routine to create a viewing transformation matrix, standing at Pos, and looking in direction Dir, with UpDir being the upper viewing direction Creates a transformation matrix that takes Pos to DEFAULT_VIEW_POS and rotate Dir into the Z axis, and UpDir into the Y axis.

4.2.165 GMMatchPointListIntoPolylines (poly_pts.c:1014)

```
IPPolygonStruct *GMMatchPointListIntoPolylines(IPObjectStruct *PtsList,
                                               IrtRType MaxTol)
```

PtsList: Point list to connect into polylines.

MaxTol: Maximum distance allowed to connect to points.

Returns: Connected polylines, upto MaxTol tolerance.

Description: Connect the list of points into polylines by connecting the closest point pairs, until the distances between adjacent points/polylines is more than MaxTol. Points are assumed to be in E3.

4.2.166 GMMatrixToTransform (quatrnn.c:898)

```
void GMMatrixToTransform(IrtHmgnMatType Mat,
                         IrtVecType S,
                         GMQuatType R,
                         IrtVecType T)
```

Mat: Source matrix to decompose.

S: Scaling components.

R: Rotation Components.

T: Translation components.

Returns: void

Description: Decompose the affine transformation matrix into uniform scaling, rotation and translation components. Rotation is defined as normal and angle.

4.2.167 GMMergeClosedLoopHoles (merge.c:663)

merge

polygons

holes

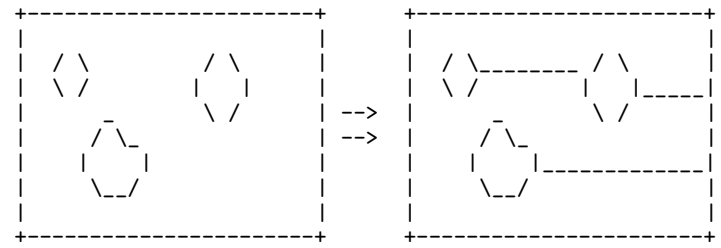
```
IPPolygonStruct *GMMergeClosedLoopHoles(IPPolygonStruct *PlMain,
                                        IPPolygonStruct *PClosedPls)
```

PlMain: Main polygon to combine with the given closed loops, in place.

PClosedPls: Closed loops inside polygon PlMain, to merge with, in place.

Returns: Merged polygon, based on PlMain, or NULL if error.

Description: This routines combines a given polygon with its interior closed loops by adding an edge from the polygon boundary to each of its closed loops For example:



Algorithm:

1. Transform the polygon and the closed loops to a plane parallel to XY plane (Z = Const plane).
2. For each loop find its MaxX while keeping the information on the vertex on that extremum.
3. For the loop with the biggest MaxX:
   3.1. based on the sign of the area decide to reverse (or not) the inner loop.
   3.2. Fire a ray from the extremum vertex, to the +X direction outside of the loop till it intersect the polygon, break the polygon at that point and add two edges to beginning of loop from breaking point and from end of loop to breaking point (beginning/end point of loop is the extrema vertex point).
4. Repeat step 3, with all loops.
5. Transform the new polygon back (using the inverse matrix of step 1) to its original orientation.

See also: MvarPolyMergePolylines, GMMergePolylines, GMMergeSameGeometry,

4.2.168 GMMergeFindSimilarPoints (merge.c:894)

merge

```
int *GMMergeFindSimilarPoints(IrtRType *VecPts,
                              int PtLen,
                              int VecLen,
                              IrtRType Tolerance)
```

VecPts: Vector of points to seek similarities.

PtLen: Number of real valued coordinates in point (2 for E2, 4 for P3, etc.).

VecLen: Length of VecPts. VecPts should have PtLen * VecLen reals.

Tolerance: To consider two points the same, in L^2 norm.

Returns: The dynamically allocated vector of indices, so that VecIdx[i] = j means i and j are the same.

Description: Finds out similar (to Tolerance) points in the given vector of points of length VecLen. Returned is a newly allocated vector VecIdx of length VecLen so VecIdx[i] = j means point i is the same as point j.

See also: GMMergeSameGeometry,

4.2.169 GMMergeGeometry (merge.c:474)

merge
points
polylines

```
int GMMergeGeometry(void **GeomEntities,
                    int NumOfGEntities,
                    IrtRType Eps,
                    IrtRType IdenticalEps,
                    VoidPtr GenericData,
                    GMMergeGeomInitFuncType InitFunc,
                    GMMergeGeomDistFuncType DistSqrFunc,
                    GMMergeGeomKeyFuncType *KeyFuncs,
                    GMMergeGeomMergeFuncType MergeFunc)
```

GeomEntities: Geometry to merge, typically points or polylines, as a vector of reference pointers to the entities. Returned, merged, data will also be stored here.

NumOfGEntities: Number of geometric entities. Also length of GeomEntities.

Eps: Epsilon of similarity to merge entities at. Entities farther than Eps will not be merged.

IdenticalEps: Epsilon to consider two entities' distance as zero.

GenericData: Optional (can be NULL) data to transfer to all call back functions.

InitFunc: A function to initialize all geometry. NULL for the initialization of IPPolygonStruct polylines.

DistSqrFunc: A distance computation function. NULL for two IPPolygonStruct polylines compare. This function computes the minimal distance squared between two entities.

KeyFuncs: A NULL terminated version of functions to return a Key to sort the entities so that similar/adjacent entities will get a similar Key. A polyline, for example, can have two Key functions of its two end points (to be merged).

MergeFunc: A merge function. NULL to merge two IPPolygonStructs polylines merge. This function returns a merged entity while destroying the two input entities.

Returns: Number of merged entities in the end or 0 if error.

Description: Merges separated geometric entities into longer ones, in place, as much as possible. Given a list of entities (typically points or polylines), find and merge closest ones as possible and in place.
See also: MvarPolyMergePolylines, GMMergePolylines, GMMergeSameGeometry,

4.2.170 GMMergePolylines (poly_pts.c:946)

merge

polyline

```
IPPolygonStruct *GMMergePolylines(IPPolygonStruct *Polys, IrtRType Eps)
```

Polys: Polylines to merge, in place.

Eps: Epsilon of similarity to merge points at.

Returns: Merged as possible polylines.

Description: Merges separated polylines into longer ones, in place, as possible. Given a list of polylines, matches end points and merged as possible polylines with common end points, in place.
See also: GMMergeGeometry, MvarPolyMergePolylines, GMMergePolylines2,

4.2.171 GMMergeSameGeometry (merge.c:308)

```
int GMMergeSameGeometry(void **GeomEntities,
                        int NumOfGEntities,
                        IrtRType IdenticalEps,
                        VoidPtr GenericData,
                        GMMergeGeomInitFuncType InitFunc,
                        GMMergeGeomDistFuncType DistSqrFunc,
                        GMMergeGeomKeyFuncType *KeyFuncs,
                        GMMergeGeomMergeFuncType MergeFunc)
```

GeomEntities: Geometry to merge, typically points or polylines, as a vector of reference pointers to the entities. Returned, merged, data will also be stored here.

NumOfGEntities: Number of geometric entities. Also length of GeomEntities.

IdenticalEps: Epsilon to consider two entities' distance as zero. This epsilon is typically very small, i.e., IRIT_UEPS.

GenericData: Optional (can be NULL) data to transfer to all call back functions.

InitFunc: A function to initialize all geometry. NULL for the initialization of IPPolygonStruct polylines.

DistSqrFunc: A distance computation function. NULL for two IPPolygonStruct polylines compare. This function computes the minimal distance squared between two entities.

KeyFuncs: Two function to return two Keys to sort the entities so that similar/adjacent entities will get similar a Key. For example for a polyline, they will return a key related to the two end points of the polyline.

MergeFunc: A merge function. NULL to merge two IPPolygonStructs polylines merge. This function destroys the two input entities and returns the merged entity in the first reference, setting the second reference to NULL.

Returns: Number of merged entities in the end or 0 if error.

Description: Merges separated geometric entities into longer ones, in place, as much as possible. Given a list of entities (typically points or polylines), find and merge closest ones as possible and in place. This function should only be used to merge geometry that is very precise and adjacent entities distance is very small.
See also: GMMergeGeometry,

4.2.172 GMMinSpanCirc (ms_circ.c:65)

minimum spanning circle

```
int GMMinSpanCirc(IrtE2PtStruct *DTPts,
                  int NumOfPoints,
                  IrtE2PtStruct *Center,
                  IrtRType *Radius)
```

DTPts: The set of point to compute their MSC.

NumOfPoints: Number of points in set DTPts.

Center: Of computed MSC.

Radius: Of computed MSC.

Returns: TRUE if successful, FALSE otherwise.

Description: Minimum spanning circle (MSC) computation of a set of points. Algorithm is based on Section 4.7 of "Computational Geometry, Algorithms and Applications" by M. de Berg et. al.
See also: GMMinSpanCone,

4.2.173 GMMinSpanCone (ms_circ.c:284)

```
int GMMinSpanCone(IrtVecType *DTVecs,
                  int VecsNormalized,
                  int NumOfVecs,
                  IrtVecType ConeAxis,
                  IrtRType *ConeAngle)
```

DTVecs: The set of vectors to compute their MSC.

VecsNormalized: TRUE if vectors are normalized, FALSE otherwise.

NumOfVecs: Number of vectors in set DTVecs.

ConeAxis: Of computed MSC.

ConeAngle: Of computed MSC, in radians.

Returns: TRUE if successful, FALSE otherwise.

Description: Minimum spanning cone (MSC) computation of a set of vectors. Algorithm is based on the Minimum Spanning Circle in Section 4.7 of "Computational Geometry, Algorithms and Applications" by M. de Berg et. al.
See also: GMMinSpanCirc, GMMinSpanConeAvg,

4.2.174 GMMinSpanConeAvg (ms_circ.c:215)

```
int GMMinSpanConeAvg(IrtVecType *DTVecs,
                     int VecsNormalized,
                     int NumOfVecs,
                     IrtVecType ConeAxis,
                     IrtRType *ConeAngle)
```

DTVecs: The set of vectors to compute their MSC.

VecsNormalized: TRUE if vectors are normalized, FALSE otherwise.

NumOfVecs: Number of vectors in set DTVecs.

ConeAxis: Of computed MSC.

ConeAngle: Of computed MSC, in radians.

Returns: TRUE if successful, FALSE otherwise.

Description: Minimum spanning cone (MSC) computation of a set of vectors. Find a central vector as the average of all given vectors and find the vector with maximal angular distance from it.
See also: GMMinSpanCirc, GMMinSpanCone,

4.2.175 GMMinSpanSphere (ms_spher.c:60)

minimum spanning sphere

```
int GMMinSpanSphere(IrtE3PtStruct *DTPts,
                    int NumOfPoints,
                    IrtE3PtStruct *Center,
                    IrtRType *Radius)
```

DTPts: The set of point to compute their MSS.

NumOfPoints: Number of points in set DTPts.

Center: Of computed MSS.

Radius: Of computed MSS.

Returns: TRUE if successful, FALSE otherwise.

Description: Minimum spanning sphere (MSS) computation of a set of points.
See also: GMMinSpanCirc,

4.2.176 GMMonotonePolyConvex (cnvxhull.c:347)

```
int GMMonotonePolyConvex(IPVertexStruct *VHead, int Cnvx)
```

VHead: An X-monotone polyline to make into a convex (concave) one. Note the first and last vertices will always be in the output sequence.

Cnvx: TRUE to make the vertex sequence Vhead convex, FALSE to make it concave, in the functional sense.

Returns: TRUE if succesful, FALSE otherwise.

Description: Compute the convex (concave) envelope of an X-monotone shape, in the XY plane, by purging all concave (convex) vertices in the sequence, in place.
See also: GMConvexHull,

4.2.177 GMObjExplodeObject (geomat3d.c:1795)

```
IPObjectStruct *GMObjExplodeObject(const IPObjectStruct *PObj,
                                   int ExplosionType,
                                   const IrtVecType CenterOfExplosion,
                                   IrtRType ExplosionAmount,
                                   const IrtVecType LineExplodeDir,
                                   int AnimCrvs)
```

PObj: Object to explode view, including its sub-objects, in place.

ExplosionType: Type of explosion - FALSE to centralize around Center, TRUE to explode along the direction of Center vector.

CenterOfExplosion: Center of exploded view. If directional (ExplosionType is TRUE), sets a point on the base plane orthogonal to direction.

ExplosionAmount: Amount to explode-move the objects. Zero to effectively disable.

LineExplodeDir: If directional explosion, sets the direction.

AnimCrvs: TRUE to update anim curves, FALSE for ObjectMatrix update.

Returns: Exploded object.

Description: A function to create an exploded view of the given object hierarchy.

4.2.178 GMObjectNumCoordinates (geomat3d.c:824)

```
int GMObjectNumCoordinates(const IPObjectStruct *Obj)
```

Obj: Object to figure its highest dimension.

Returns: The computed (highest) dimension, or 0 if no geometry was found.

Description: Computes the number of dimensions the given object has, based on the sub-object found with the hiest dimension.
See also:

4.2.179 GMOglZPeel (ogl_depth_peel.c:196)

```
GMOglZPeelBufferStruct *GMOglZPeel(const IPObjectStruct *PObj,
                                   GMOglZPeelParamStruct *Params)
```

PObj: Object to render.

Params: The peel parameters.

Returns: A list of all the peeled z levels.

Description: Peel an object's depth using an off-screen openGL renderer.
See also: GMZBufferOGLInit, GMZBufferInit, GMOglZPeelTesselate,

4.2.180 GMOglZPeelTesselate (ogl_depth_peel.c:1056)

```
IPObjectStruct *GMOglZPeelTesselate(struct GMOglZPeelBufferStruct *Peel,
                                    const GMBBBboxStruct *Domain,
                                    int LayerIndex)
```

Peel: The peel z-buffer object.

Domain: World space domain to map to from z-buffer space.

LayerIndex: The index of the specific peel layer to tessellate.

Returns: A polygonal object that approximates the z-buffer.

Description: tessellates the given OGL z-buffer peel structure, into triangles.
See also: GMZBufferOGLInit, GMZBufferInit, GMOglZPeel,

4.2.181 GMOrthogonalVector (geom_bsc.c:391)

```
int GMOrthogonalVector(const IrtVecType V, IrtVecType OV, int UnitLen)
```

V: Input vector to find an orthogonal vector for.

OV: Output newly computed orthogonal (possibly unit) vector to V, in R^3.

UnitLen: If TRUE, normalize the outpt vector.

Returns: TRUE if successful, FALSE otherwise.

Description: Computes an orthogonal vector in R^3 to the given vector.

4.2.182 GMPlCrvtrSetCurvatureAttr (plycrvtr.c:55)

```
void GMPlCrvtrSetCurvatureAttr(IPPolygonStruct *PolyList,
                               int NumOfRings,
                               int EstimateNrmls)
```

PolyList: The triangular two-manifold mesh data.

NumOfRings: Number of rings around a vertex in the paraboloid fitting.

EstimateNrmls: If TRUE estimate normals to the vertices on the fly. This functions needs these normals for its proper work.

Returns: void

Description: Estimates the Gaussian and Mean curvature values for the given triangular regular mesh and initializes the corresponding attributes: for each vertex: "K1Curv", "K2Curv", "D1", "D2" and "K" and "H". Uses a least sqaures osculating quadratic function in the estimate. Mesh is assumed to be a triangular regular mesh.
See also: GMPlCrvtrSetFitDegree, SymbEvalSrfCurvPrep, SymbEvalSrfCurvature,

4.2.183 GMPlCrvtrSetFitDegree (plycrvtr.c:305)

```
int GMPlCrvtrSetFitDegree(int UseCubic)
```

UseCubic: TRUE to use cubic fit, FALSE for a quadratic

Returns: Old value of fitting degree.

Description: Sets the degree for the continuous function we fit at the vertex.
See also: GMPlCrvtrSetCurvatureAttr,

4.2.184 GMPlSilImportanceAttr (plyimprt.c:34)

```
void GMPlSilImportanceAttr(IPPolygonStruct *PolyList)
```

PolyList: The triangular two-manifold mesh data.

Returns: void

Description: Estimates the importance of vertices of a polygonal mesh based on the probability of their adjacent edges to posses silhouettes. Mesh is assumed to be a triangular regular mesh. Each vertex in the mesh is assigned a new "SilImp" attribute which is a positive value.
See also: GMPlCrvtrSetCurvatureAttr,

4.2.185 GMPlSilImportanceRange (plyimprt.c:235)

```
IPPolygonStruct *GMPlSilImportanceRange(IPPolygonStruct *PolyList)
```

PolyList: The triangular two-manifold mesh data.

Returns: the extracted geometry.

Description: Extract the silhouette importance range of a polygonal mesh with "SilImp" attribute. See also GMPlSilImportanceAttr. Mesh is assumed to be a triangular regular mesh.
See also: GMPlSilImportanceAttr,

4.2.186 GMPlanarVecVecAngle (geom_bsc.c:237)

```
IrtRType GMPlanarVecVecAngle(const IrtVecType V1,
                             const IrtVecType V2,
                             int Normalize)
```

V1, V2: Planar vectors to compute their relative angle, in degrees.

Normalize: TRUE if vectors need normalization first, FALSE if unit size.

Returns: Angle between V1 and V2, in degree.

Description: Compute the angle between two planar vectors V1 and V2. Angle is zero if V2 is in the exact same direction as V1, negative if V2 turns right and positive if V2 turns left. Angle is returned in degrees in the domain of (-180, +180]. Only the XY coefficients of V1 and V2 are considered.

4.2.187 GMPlaneFrom3Points (geom_bsc.c:755)

plane

```
int GMPlaneFrom3Points(IrtPlnType Plane,
                       const IrtPtType Pt1,
                       const IrtPtType Pt2,
                       const IrtPtType Pt3)
```

Plane: To compute.

Pt1, Pt2, Pt3: Three points to fit a plane through.

Returns: TRUE if successful, FALSE otherwise.

Description: Routine to construct the plane from given 3 points. If two of the points are the same or the three points are collinear it returns FALSE, otherwise (successful), it returns TRUE.

4.2.188 GMPointCoverOfPolyObj (poly_cvr.c:42)

```
IPObjectStruct *GMPointCoverOfPolyObj(IPObjectStruct *PolyObj,
                                      int n,
                                      IrtRType *Dir,
                                      char *PlAttr)
```

PolyObj: Object to compute a uniform point distribution on.

n: Number of points to distribute (estimate).

Dir: If given - use it as view dependent uniform distribution. Note that if Dir != NULL less than n points will be generated.

PlAttr: If not NULL, the created points are placed as attributes named PlAttr in each polygon in the return (copied) model.

Returns: A point list object.

Description: Computes a uniform distribution of points on a polygonal object. If an "Imprt" attribute is found in a polygon then it is used to weigh the importance of this polygon and hence the number of points that will be allocated to (on) it.

4.2.189 GMPointCoverOfUnitHemiSphere (sph_pts.c:30)

```
IPObjectStruct *GMPointCoverOfUnitHemiSphere(IrtRType HoneyCombSize)
```

HoneyCombSize: Size of honey comb, on the unit sphere. A fraction.

Returns: A point list object.

Description: Computes a honey comb distribution of points on a sphere. Result is an approximation.

4.2.190 GMPointFrom3Planes (geom_bsc.c:1292)

```
int GMPointFrom3Planes(const IrtPlnType Pl1,
                       const IrtPlnType Pl2,
                       const IrtPlnType Pl3,
                       IrtPtType Pt)
```

Pl1, Pl2, Pl3: Three planes to consider.

Pt: Intersection point found (if any).

Returns: TRUE if exists an intersection point, FALSE otherwise.

Description: Find the intersection point (if exists) of three planes.

4.2.191 GMPointFromLinePlane (geom_bsc.c:1037)

line plane intersection

```
int GMPointFromLinePlane(const IrtPtType Pl,
                         const IrtPtType Vl,
                         const IrtPlnType Plane,
                         IrtPtType InterPoint,
                         IrtRType *t)
```

Pl, Vl: Position and direction that defines the line.

Plane: To find the intersection with the line.

InterPoint: Where the intersection occurred.

t: Parameter along the line of the intersection location (as Pl + Vl * t).

Returns: TRUE, if successful.

Description: Routine to find the intersection point of a line and a plane (if any). The Plane is prescribed using four coefficients : Ax + By + Cz + D = 0 given as four elements vector. The line is define via a point on it Pl and a direction vector Vl. Returns TRUE only if such point exists.

4.2.192 GMPointFromLinePlane01 (geom_bsc.c:1090)

line plane intersection

```
int GMPointFromLinePlane01(const IrtPtType Pl,
                           const IrtPtType Vl,
                           const IrtPlnType Plane,
                           IrtPtType InterPoint,
                           IrtRType *t)
```

Pl, Vl: Position and direction that defines the line.

Plane: To find the intersection with the line.

InterPoint: Where the intersection occurred.

t: Parameter along the line of the intersection location (as Pl + Vl * t).

Returns: TRUE, if successful and t between zero and one.

Description: Routine to find the intersection point of a line and a plane (if any). The Plane is prescribed using four coefficients : Ax + By + Cz + D = 0 given as four elements vector. The line is define via a point on it Pl and a direction vector Vl. Returns TRUE only if such point exists. this routine accepts solutions only for t between zero and one.

4.2.193 GMPointFromPlanarLineLine (geom_bsc.c:1153)

line line intersection

```
int GMPointFromPlanarLineLine(const IrtPtType Pl1,
                              const IrtPtType Vl1,
                              const IrtPtType Pl2,
                              const IrtPtType Vl2,
                              IrtPtType Pi,
                              IrtRType *t1,
                              IrtRType *t2)
```

Pl1, Vl1: Position and direction defining the first line.

Pl2, Vl2: Position and direction defining the second line.

Pi: Intersection point if lines are not parallel.

t1: Parameter value of Pi as (Pl1 + Vl1 * t1).

t2: Parameter value of Pi as (Pl2 + Vl2 * t2).

Returns: TRUE, if successful.

Description: Routine to find the intersection point Pi on the two coplanar lines (Pli, Vli), i = 1, 2. This function returns TRUE iff the two lines are not parallel! Solved as the solution of the following two linear equations:

```
Vl1(x) t1 - Vl2(x) t2 = Pl2(x) - Pl1(x)
Vl1(y) t1 - Vl2(y) t2 = Pl2(y) - Pl1(y)
```

See also: GM2PointsFromLineLine,

4.2.194 GMPointFromPointLine (geom_bsc.c:808)

point line distance

```
IrtRType GMPointFromPointLine(const IrtPtType Point,
                              const IrtPtType Pl,
                              const IrtPtType Vl,
                              IrtPtType ClosestPoint)
```

Point: To find the closest to on the line.

Pl, Vl: Position and direction that defines the line. Vl need not be a unit length vector.

ClosestPoint: Where closest point found on the line is to be saved.

Returns: Parameter along the line where if the closest point is Pl equal zero and if closest point is Pl + Vl equal one. In other words, the closest point is on the finite line segment for return value in [0, 1].

Description: Routine to compute the closest point on a given 3d line to a given 3d point. The line is prescribed using a point on it (Pl) and vector (Vl).

See also: MvarPointFromPointLine,

4.2.195 GMPointFromPointPlane (geom_bsc.c:986)

point plane distance

```
int GMPointFromPointPlane(const IrtPtType Pt,
                          const IrtPlnType Plane,
                          IrtPtType ClosestPoint)
```

Pt: Point to find closest point on Plane to.

Plane: To find the closest point on to Pt.

ClosestPoint: Where the closest point on Plane to Pt is.

Returns: TRUE, if successful.

Description: Routine to compute the closest point on a given plane to a given 3d point. The Plane is prescribed using four coefficients : Ax + By + Cz + D = 0 given as four elements vector.

4.2.196 GMPointInsideCnvxPolygon (geomat3d.c:1575)

```
int GMPointInsideCnvxPolygon(const IrtPtType Pt, const IPPolygonStruct *Pl)
```

Pt: Point to test for inclusion in convex polygon Pl. Pt is assumed to be on the plane holding polygon Pl.

Pl: Convex polygon to test for inclusion of Pt.

Returns: TRUE if Pt is inside Polygon Pl, FALSE otherwise.

Description: Test if a given points is contained in the given convex polygon.
See also: GMPointOnPolygonBndry,

4.2.197 GMPointOnPolygonBndry (geomat3d.c:1628)

```
int GMPointOnPolygonBndry(const IrtPtType Pt,
                          const IPPolygonStruct *Pl,
                          IrtRType Eps)
```

Pt: Point to test for inclusion in the boundary of polygon Pl. Pt is assumed to be on the plane holding polygon Pl.

Pl: Polygon to test for inclusion of Pt.

Eps: Maximal distance from the boundary to be considered on boundary.

Returns: TRUE if Pt is on the boundary of Polygon Pl, FALSE otherwise.

Description: Test if a given points is on the boundary of the given polygon.
See also: GMPointInsideCnvxPolygon,

4.2.198 GMPointVecFromLine (geom_bsc.c:719)

```
void GMPointVecFromLine(const IrtLnType Line, IrtPtType Pt, IrtVecType Dir)
```

Line: To extract and point and a direction for.

Pt: A point on line Line.

Dir: The direction of line Line.

Returns: void

Description: Computes a point on and the direction of the given line

4.2.199 GMPolyAdjacncyFree (plystrct.c:393)

```
void GMPolyAdjacncyFree(VoidPtr PolyAdj)
```

PolyAdj: Data struct to free, as constructed by GMPolyAdjacncyGen.

Returns: void

Description: Free the adjacency data structure associated with a polygonal model.
See also: GMPolyAdjacncyGen, GMPolyAdjacncyVertex,

4.2.200 GMPolyAdjacncyGen (plystrct.c:175)

```
VoidPtr GMPolyAdjacncyGen(IPObjectStruct *PObj, IrtRType EqlEps)
```

PObj: A polygonal mesh to compute adjacency information for.

EqlEps: Epsilon for point equality.

Returns: A reference to the data structure holding the adjacency info.

Description: Constructs an adjacency information data structure for the given polygonal mesh in GMPolyAdjStruct: VList: A list of vertices and for each all edges using the vertex. EList: A list of edges, each referencing the two vertices using it. PObj: A reference to the original model.
See also: GMPolyAdjacncyVertex, GMPolyAdjacncyFree,

4.2.201 GMPolyAdjacncyVertex (plystrct.c:336)

```
void GMPolyAdjacncyVertex(IPVertexStruct *V,
                          VoidPtr PolyAdj,
                          GMPolyAdjacncyVertexFuncType AdjVertexFunc)
```

V: Vertex to find all edges that share it.

PolyAdj: Data struct to use, as constructed by GMPolyAdjacncyGen.

AdjVertexFunc: Call be function to invoke on every edge.

Returns: void

Description: Get the adjacency information of a vertex - all the edges that share it. Invokes the given call back function on all edges.
See also: GMPolyAdjacncyGen, GMPolyAdjacncyFree,

4.2.202 GMPolyBVHBoxPolyInter (bvh_poly.c:1318)

```
int GMPolyBVHBoxPolyInter(const GMBBBboxStruct *BBox,
                          const struct GMPolyBVHStruct *PolyBVH)
```

BBox: The domain of the box in question.

PolyBVH: The BVH of the mesh.

Returns: TRUE if BBox intersects with the mesh, FALSE otherwise.

Description: Test whether a box intersects with the polygonal mesh using the BVH of the mesh.
See also:

4.2.203 GMPolyBVHCreate (bvh_poly.c:84)

```
struct GMPolyBVHStruct *GMPolyBVHCreate(const IPPolygonStruct *Pl)
```

Pl: The polygonal mesh to build the BVH.

Returns: The returned BVH.

Description: construct a box BVH of the given polygonal mesh.
See also: GMPolyBVHFree,

4.2.204 GMPolyBVHCrvsPolyInter (bvh_poly.c:1281)

GMPolyBVHCrvPolyInter

```
int GMPolyBVHCrvsPolyInter(const struct IPObjectStruct *PCrvs,
                           const struct GMPolyBVHStruct *PolyBVH)
```

PCrvs: A list of curves to test intersection with the mesh.

PolyBVH: The BVH of the mesh.

Returns: TRUE if one of PCrvs intersects with the mesh, FALSE otherwise.

Description: Test whether a list of curves intersects with the polygonal mesh using the bounding volume hierarchy of the mesh. Return true if one of the curves intersects with the mesh.
See also:

4.2.205 GMPolyBVHFree (bvh_poly.c:173)

```
void GMPolyBVHFree(struct GMPolyBVHStruct *PolyBVH)
```

PolyBVH: The BVH to free.

Returns: void

Description: frees a box BVH for the mesh.
See also: GMPolyBVHCreate,

4.2.206 GMPolyBVHGetClosestPoint (bvh_poly.c:510)

```
int GMPolyBVHGetClosestPoint(const IrtPtType QueryPt,
                             const struct GMPolyBVHStruct *PolyBVH,
                             IrtPtType ClosestPt,
                             int *ClosestPlIdx)
```

QueryPt: The point to search the closest point on the mesh.

PolyBVH: The BVH of the polygonal mesh.

ClosestPt: The closest point on the mesh.

ClosestPlIdx: The index of triangle where ClosestPt is on.

Returns: TRUE if the closest point is found, FALSE otherwise.

Description: For the given query point, finds the closest point on the polygonal mesh. It first compute the minimum distances between the input point and the two child BV nodes. Among two child nodes, pick the node having the smaller distance from the input point and continue to compute the minimum distance using the child BV nodes of the picked node. Continue comparisons until a leaf BV node is encountered. After the leaf is encounter, then the minimum distance is computed between the actual triangle and the input point.
See also: GMPolyBVHGetClosestPoint2,

4.2.207 GMPolyBVHGetClosestPoint2 (bvh_poly.c:443)

GMPolyBVHMinDistEnqueue

```
int GMPolyBVHGetClosestPoint2(const IrtPtType QueryPt,
                              const struct GMPolyBVHStruct *PolyBVH,
                              IrtPtType ClosestPt,
                              int *ClosestPlIdx)
```

QueryPt: The point to search the closest point on the mesh.

PolyBVH: The BVH of the polygonal mesh.

ClosestPt: The closest point on the mesh.

ClosestPlIdx: The index of triangle where ClosestPt is on.

Returns: TRUE if the closest point is found, FALSE otherwise.

Description: For the given query point, finds the nearest point on the polygonal mesh. The nearest point is computed while traversing the BVH of the mesh in the breadth-first way: It first enqueues the root BV to the priority queue. While the queue is not empty, the first node of the queue (which has the smallest minimum distance as a key) is deleted and the two child nodes are enqueued to the queue. If one (or more) of the child nodes as the larger minimum distance than the known minimum distance, then the node(s) is(are) not enqueued and all triangles in the subtree of the node is purged from the search. The known distance is initialized with a very large number and updated whenever we can guarantee the minimum distance between the node and the input point (i.e. when the leaf BV has been encountered). This routine computes the global minimum distance between the input point and the mesh model.
See also: GMPolyBVHGetClosestPoint,

4.2.208 GMPolyBVHGetRayBVHIntersectionPt (bvh_poly.c:782)

```
int GMPolyBVHGetRayBVHIntersectionPt(const IrtPtType RayPt,
                                     const IrtVecType RayDir,
                                     const struct GMPolyBVHStruct *PolyBVH,
                                     IrtPtType *ResPt,
                                     const IPPolygonStruct **ResPl)
```

RayPt: A point whether the ray is being shot from.

RayDir: The direction of the ray.

PolyBVH: The BVH of the mesh.

ResPt: Array of the intersection points between the ray and the mesh. Allocated by the caller to be large enough to hold all inters.

ResPl: Array of the intersection polys between the ray and the mesh. Allocated by the caller to be large enough to hold all inters. Can be NULL to ignore.

Returns: The number of intersection points.

Description: Find the intersection points between the polygonal mesh and the ray starting from the given point and shooting along the given direction. This routine first collects a list of boxes that bound the triangles of the mesh and intersect with the given ray using the BVH of the mesh. For each box that intersects with the ray, it further tests whether the ray intersects with the actual triangle in the box or not. If the ray intersects with the triangle, then the intersection point on the triangle is calculated.
See also: GMPolyBVHPointInsidePolys,

4.2.209 GMPolyBVHPointInsidePolys (bvh_poly.c:839)

GMPolyBVHGetRayBVHIntersectionPt

```
int GMPolyBVHPointInsidePolys(const IrtPtType Pt,
                              const struct GMPolyBVHStruct *PolyBVH)
```

Pt: A point to test whether to be inside the mesh or not.

PolyBVH: The BVH of the mesh.

Returns: TRUE if Pt is inside the mesh, and FALSE otherwise.

Description: Determines whether the given point is inside the mesh model or not. This routine creates a set of rays in random directions and shoot the rays from the query point. For each ray, it computes the number of intersections between the ray and the mesh model using the BVH of the mesh. If the number of intersections is odd, the ray is considered being shot from the inside (outside if the number of intersections is even). The final inclusion is determined from votes of the inclusion results of all rays.
See also:

4.2.210 GMPolyBVHPolygonPtNormalEstimation (bvh_poly.c:954)

```
int GMPolyBVHPolygonPtNormalEstimation(const IrtPtType Pt,
                                       const struct GMPolyBVHStruct *PolyBVH,
                                       IrtVecType RES)
```

Pt: A point to estimate the surface normal at.

PolyBVH: The BVH of the mesh.

RES: The estimated surface normal.

Returns: TRUE if the surface normal is computed successfully.

Description: Estimate the surface normal of the polygonal mesh for the given point. If the point is not on the mesh, then return FALSE. If the point is on one of the triangle of the mesh, then it further checks whether the point is on one of the vertices of the mesh. If the point is on the vertex, then the normal is calculated by averaging the face normals of the triangles adjacent to the vertex. If the point is not on the vertex, but on one of the edges of the mesh, then the normal is calculated by blending the face normals of the triangles adjacent to the edge. Otherwise, the face normal of the triangle where the point is on is returned.
See also:

4.2.211 GMPolyBVHSrfsPolyInter (bvh_poly.c:1170)


```
int GMPolyBVHSrfsPolyInter(const struct IPObjectStruct *PSrfs,
                           const struct GMPolyBVHStruct *PolyBVH)
```

PSrfs: A list of surfaces to test intersection with the mesh.

PolyBVH: The BVH of the mesh.

Returns: TRUE if one of PSrfs intersects with the mesh, FALSE otherwise.

Description: Test whether a list of surfaces intersects with the polygonal mesh using the bounding volume hierarchy of the mesh. Return true if one of the surfaces intersects with the mesh.
See also:

4.2.212 GMPolyCentroid (geomvals.c:66)

```
int GMPolyCentroid(const IPPolygonStruct *Pl, IrtPtType Centroid)
```

Pl: The poly to compute its centroid.

Centroid: Computed center point. Note it can be outside Pl!

Returns: TRUE if at least one vertex in input Pl, FALSE otherwise.

Description: Computes the centroid of a poly, as an average of all input vertices.
See also: GMComputeAverageVertex, GMPolygonGetCentroid,

4.2.213 GMPolyHasCollinearEdges (poly_pts.c:256)

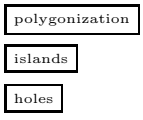
```
int GMPolyHasCollinearEdges(const IPPolygonStruct *Pl)
```

Pl: Polygon to examine for collinear edges.

Returns: TRUE if Pl has collinear edges, FALSE otherwise.

Description: Examines if the given polygons has collinear edges (three consecutive vertices that are collinear).
See also:

4.2.214 GMPolyHierarchy2SimplePoly (geom_bsc.c:2502)




```
IPPolygonStruct *GMPolyHierarchy2SimplePoly(IPPolygonStruct *Root,
                                            IPPolygonStruct *Islands)
```

Root: The top most, outer polygon.

Islands: Interior islands to connected with root into one simply poly.

Returns: Merged poly.

Description: Converts a root polygons with islands into a closed, simple, polygon. Interior islands are all connected into the root, outer, polygon. The outer, root, loop is assumed to be oriented in opposite direction to the islands.

4.2.215 GMPolyLength (geomvals.c:33)

```
IrtRType GMPolyLength(const IPPolygonStruct *Pl)
```

Pl: The poly to compute its length.

Returns: The length of the poly(line).

Description: Computes the lengths of a poly, first vertex to last vertex.

4.2.216 GMPolyMeshSmoothing (polysmth.c:84)

```
IPObjectStruct *GMPolyMeshSmoothing(IPObjectStruct *PolyObj,
                                    const IPPolygonStruct *VerticesToRound,
                                    int AllowBndryMove,
                                    IrtRType RoundingRadius,
                                    int NumIters,
                                    IrtRType BlendFactor,
                                    int CurvatureLimits)
```

PolyObj: Polygonal object to smooth, in place.

VerticesToRound: If not NULL, only vertices found in VerticesToRound are allowed to move and other vertices are kept stationary.

AllowBndryMove: 0 to keep boundary vertices fixed, 1 to allow the boundary vertices to move only in the plane containing the boundary, -1 to allow the boundary vertices to freely move.

RoundingRadius: If we have a restriction on the movable vertices any other vertex that is at a Euclidean distance of less than RoundingRadius from a restricted vertex will also be tagged as such.

NumIters: Number of times to perform this smoothing algorithm.

BlendFactor: 1.0 to move the vertex all the way to average position. Otherwise, (less than 1.0) to the proper fraction.

CurvatureLimits: If TRUE, estimate the curvature and make sure we are within the desired rounding radius. Available only for meshes with triangles only.

Returns: The smoothed out polygons, in place. Same as PolyObj.

Description: Move designated Vertices (that are typically not on the boundary) of the polygons in PolyObj to new (averages of their 1-rings) positions, smoothing the shape of the mesh.

4.2.217 GMPolyObjectArea (geomvals.c:121)

area

```
IrtRType GMPolyObjectArea(const IPObjectStruct *PObj, int SignedArea)
```

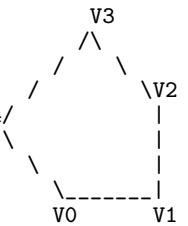
PObj: A polyhedra object to compute its surface area.

SignedArea: True for sign area, FALSE always positive.

Returns: The area of object PObj.

Description: Routine to evaluate the Area of the given geom. object, in object unit.

```
Algorithm (for each polygon):
1. Set Polygon Area to be zero.
   Make a copy of the original polygon
   and transform it to a XY parallel plane.
   Find the minimum Y value of the polygon
   in the XY plane.
2. Let V(0) be the first vertex, V(n) the last one.
   For i goes from 0 to n-1 add to Area the area
   below edge V(i), V(i+1):
   PolygonArea += (V(i+1)x - V(i)x) * (V(i+1)y' - V(i)y') / 2
   where V(i)y' is V(i)y - MinY, where MinY is polygon minimum Y value.
3. The result of step 2 is the area of the polygon itself.
   However, it might be negative, so take the absolute result of step 2
   and add it to the global ObjectArea.
```



Note step 2 is performed by another auxiliary routine: GMPolyOnePolyXYArea.

See also: GMPolyOnePolyArea,

4.2.218 GMPolyObjectVolume (geomvals.c:284)

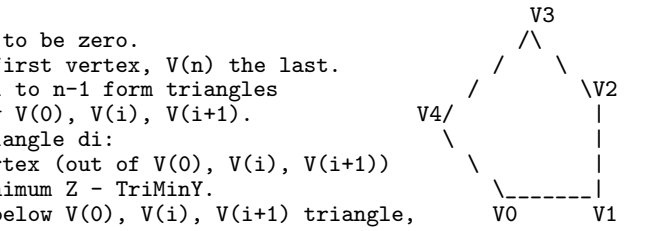
volume

```
IrtRType GMPolyObjectVolume(const IPObjectStruct *PObj)
```

PObj: To compute volume for.

Returns: Computed volume.

Description: Routine to evaluate the Volume of the given geom object, in object unit. This routine has a side effect that all non-convex polygons will be splitted to convex ones. Algorithm (for each polygon, and let ObjMinY be the minimum OBJECT Y):




```
1. Set Polygon Area to be zero.
   Let V(0) be the first vertex, V(n) the last.
   For i goes from 1 to n-1 form triangles
                 by V(0), V(i), V(i+1).
   For each such triangle di:
   1.1. Find the vertex (out of V(0), V(i), V(i+1))
        with the minimum Z - TriMinY.
   1.2. The volume below V(0), V(i), V(i+1) triangle,
        relative to ObjMinZ level, is the sum of:
        1.2.1. volume of V'(0), V'(i), V'(i+1) - the
               area of projection of V(0), V(i), V(i+1) on XY parallel
               plane, times (TriMinZ - ObjMinZ).
        1.2.2. Assume V(0) is the one with the PolyMinZ. Let V"(i) and
               V"(i+1) be the projections of V(i) and V(i+1) on the plane
               Z = PolyZMin. The volume above 1.2.1. and below the polygon
               (triangle!) will be: the area of quadraliteral V(i), V(i+1),
               V"(i+1), V"(i), times distance of V(0) for quadraliteral
               plane divided by 3.
   1.3. If Z component of polygon normal is negative add 1.2. result to
        ObjectVolume, else subtract it.
```

4.2.219 GMPolyOffset (polyofst.c:83)

```
IPPolygonStruct *GMPolyOffset(const IPPolygonStruct *Poly,
                              int IsPolygon,
                              IrtRType Ofst,
                              GMPolyOffsetAmountFuncType AmountFunc)
```

Poly: To compute its offset in the XY plane.

IsPolygon: TRUE for a polygon, FALSE for a polyline.

Ofst: Amount of offset.

AmountFunc: Scale the offset amount according to this function. A NULL here will use a constant scaling factor of one.

Returns: Offset of Poly by Ofst amount.

Description: Computes the offset of a given polygon/line in the XY plane by Ofst.
See also: GMPolyOffsetAmountDepth, GMPolyOffset3D,

4.2.220 GMPolyOffset3D (polyofst.c:182)

```
IPPolygonStruct *GMPolyOffset3D(const IPPolygonStruct *Poly,
                                IrtRType Ofst,
                                int ForceSmoothing,
                                IrtRType MiterEdges,
                                GMPolyOffsetAmountFuncType AmountFunc)
```

Poly: To compute its offset in R^3.

Ofst: Amount of offset.

ForceSmoothing: True to force normal smoothing.

MiterEdges: Positive to properly handle dihedral angle, in which case sets the maximal scale allowed. Zero to disable.

AmountFunc: Scale the offset amount according to this function. A NULL here will use a constant scaling factor of one.

Returns: Offset of Poly by Ofst amount.

Description: Computes the offset of a given polygon object in R^3 by Ofst. Computation is done by moving all vertices in the nomal direction by ofst amount. If Poly does not contain vertex normals, they are estimated.
See also: GMPolyOffsetAmountDepth, GMPolyOffset,

4.2.221 GMPolyOffsetAmountDepth (polyofst.c:55)

```
IrtRType GMPolyOffsetAmountDepth(const IrtRType *Coord)
```

Coord: Of point as XYZ values.

Returns: scaling factor of offset amount.

Description: Sets the offset amount to be a function of the depth Z value by scaling with 1/Z.
See also: GMPolyOffset,

4.2.222 GMPolyOnePolyArea (geomvals.c:157)

```
IrtRType GMPolyOnePolyArea(const IPPolygonStruct *Pl, int SignedArea)
```

Pl: To compute its area.

SignedArea: True for sign area, FALSE always positive.

Returns: Area of polygon Pl.

Description: Computes the area of a single polygon.
See also: GMPolyObjectArea, GMPolyOnePolyXYArea,

4.2.223 GMPolyOnePolyXYArea (geomvals.c:213)

```
IrtRType GMPolyOnePolyXYArea(const IPVertexStruct *VHead, int SignedArea)
```

VHead: Of vertex list to compute area for.

SignedArea: True for sign area, FALSE always positive.

Returns: Computed area.

Description: Routine to evaluate the area of the given polygon projected on the XY plane. The polygon does not have to be on a XY parallel plane, as only its XY projection is considered (Z is ignored). Returned the area of its XY parallel projection. See GMObjectArea above for algorithm.
See also: GMPolyObjectArea,

4.2.224 GMPolyPlaneClassify (geom_bsc.c:1886)

```
IrtRType GMPolyPlaneClassify(const IPPolygonStruct *Pl, const IrtPlnType Pln)
```

Pl: Polygon to consider and classify.

Pln: Plane to claassify against.

Returns: Positive or negative, classifying the polygon's side.

Description: Classify a plane as (mostly) on the positive side of the plane, returning a positive value, or (mostly) on the negative side of the plane, returning a negative value.

4.2.225 GMPolyPropFetch (polyprop.c:304)

```
IPPolygonStruct *GMPolyPropFetch(IPPolygonStruct *Pls,
                                 GMFetchVertexPropertyFuncType VertexProperty,
                                 IrtRType ConstVal,
                                 void *AuxData)
```

Pls: The polygons to process the constant value property for. Assumed to hold triangles only.

VertexProperty: A call back function to return the desired property for the invoked vertex.

ConstVal: Constant value of property sought.

AuxData: to optionally pass to the VertexProperty call back func.

Returns: List of polylines on the polygons for which the property assumes the value ConstVal.

Description: Computes the constant value over a polygonal model by processing each polygons for edges that the values of the property are begin crossed. The input polygonal model is assumed to hold triangles only.
See also: GMPolyPropFetchIsophotes, GMPolyPropFetchCurvature, , GMFetchVertexPropertyFuncType,

4.2.226 GMPolyPropFetchAttribute (polyprop.c:95)

```
IPPolygonStruct *GMPolyPropFetchAttribute(IPPolygonStruct *Pls,
                                          IPAttrIDType AttrID,
                                          IrtRType Value)
```

Pls: The polygons to process the constant value property for. Assumed to hold triangles only.

AttrID: ID of attribute to extract.

Value: Value of property to seek.

Returns: List of polylines on the polygons along the requested property value.

Description: A function to derive curves over a polygonal model, Pls, based on given prescribed attribute PropAttr.
See also: GMPolyPropFetchIsophotes, GMPolyPropFetchCurvature,

4.2.227 GMPolyPropFetchCurvature (polyprop.c:227)

```
IPPolygonStruct *GMPolyPropFetchCurvature(IPPolygonStruct *Pls,
                                          int CurvatureProperty,
                                          int NumOfRings,
                                          IrtRType CrvtrVal)
```

Pls: The polygons to process the constant value property for. Assumed to hold triangles only.

CurvatureProperty: 0 for Gaussian curvature, 1 for Mean curvature.

NumOfRings: In the paraboloid fit estimation, usually 1-2.

CrvtrVal: Value of curvature property to seek.

Returns: List of polylines on the polygons along the requested curvature lines.

Description: A function to derive curvature properties for a polygonal model, Pls.
See also: GMPolyPropFetchIsophotes, GMPolyPropFetchAttribute,

4.2.228 GMPolyPropFetchIsophotes (polyprop.c:156)

```
IPPolygonStruct *GMPolyPropFetchIsophotes(IPPolygonStruct *Pls,
                                          const IrtVecType ViewDir,
                                          IrtRType InclinationAngle)
```

Pls: The polygons to process the constant value property for. Assumed to hold triangles only.

ViewDir: Direction of view.

InclinationAngle: In degrees to consider the isophotes for. 90 degrees would yield regular silhouettes viewed from LightDir.

Returns: List of polylines on the polygons along the requested isophotes.

Description: A function to derive isophotes for a polygonal model, Pls.
See also: GMPolyPropFetchCurvature, GMPolyPropFetchAttribute,

4.2.229 GMPolygonGetCentroid (geom_bsc.c:1753)

```
void GMPolygonGetCentroid(const IPPolygonStruct *Plgn, IrtPtType *Pt)
```

Plgn: A closed polygon.

Pt: Resulting center point.

Returns: void

Description: Given a closed polygon, compute its centroid point, as a weighted average of its edges. Valid only for convex polys.
See also: GMPolyCentroid, GMComputeAverageVertex, GMComputeAverageVertex2,

4.2.230 GMPolygonListPlaneInter (geom_bsc.c:1722)

```
int GMPolygonListPlaneInter(const IPPolygonStruct *Plgns,
                            const IrtPlnType Pln)
```

Plgns: A list of 3D polygons.

Pln: Plane to examine if intersecting with the polygons.

Returns: TRUE if intersects, FALSE otherwise.

Description: Given a list of 3D polygons, test if one (or more) intersects the given plane.
See also: GMPolygonPlaneInter,

4.2.231 GMPolygonPlaneInter (geom_bsc.c:1665)

```
int GMPolygonPlaneInter(const IPPolygonStruct *Pl,
                        const IrtPlnType Pln,
                        IrtRType *MinDist)
```

Pl: 3D polygon to test if interestes plane Pln.

Pln: Plane to examine if polygon Pl intersects. Assumed normalized normal vector in Pln.

MinDist: Returns closest vertex in Pl to Pln.

Returns: TRUE if intersects, FALSE otherwise.

Description: Test if the given 3D polygon intersects the given plane.
See also: GMPolygonRayInter, GMPolygonListPlaneInter, GMSplitPolygonAtPlane,

4.2.232 GMPolygonPointInclusion (geom_bsc.c:1967)

ray polygon intersection

point inclusion

```
int GMPolygonPointInclusion(const IPPolygonStruct *Pl,
                            const IrtPtType Pt,
                            IrtRType OnBndryEps)
```

Pl: Polygon to test if Pt is in it.

Pt: Point to test for inclusion in Pl.

OnBndryEps: If not zero, will also check if Pt is on boundary to within this epsilon.

Returns: 1 if Pt inside Pl, 0 if on boundary (if OnBndryEps > 0), -1 if Pt outside.

Description: Routine to check if a point is inside a polygon, in the XY plane. Uses winding number accumulation in the computation.
See also: GMPolygonPointInclusion3D, GMPolygonPlaneInter, GMPolygonXYRayInter,

4.2.233 GMPolygonPointInclusion3D (geom_bsc.c:2111)

ray polygon intersection

point inclusion

```
int GMPolygonPointInclusion3D(const IPPolygonStruct *Pl, const IrtPtType Pt)
```

Pl: Polyhedra in 3D to test if Pt is in it.

Pt: Point to test for inclusion in Pl.

Returns: TRUE if Pt inside Pl, FALSE otherwise.

Description: Routine to check if a point is inside a polyhedra in 3D. Computes the accumulated Gaussian sphere area deviation of Pt-vertices over all triangles in the given Pl model, with respect to given point Pt. Each angular deviation of each triangle is measured as area of spherical triangle over the Gaussian sphere, formed from its three Pt-vertices dirs. A point is inside a simple closed polyhedra if the sum is +/-4 Pi, zero if the point is outside.
See also: GMPolygonPointInclusion, GMPolygonPlaneInter, GMPolygonXYRayInter, GMAreaSphericalTriangle,

4.2.234 GMPolygonXYRayInter (geom_bsc.c:2164)

ray polygon intersection

Jordan theorem

```
int GMPolygonXYRayInter(const IPPolygonStruct *Pl,
                        const IrtPtType PtRay,
                        int RayAxes)
```

Pl: To compute "Jordan Theorem" for the given ray. Can be either a polygon or a closed polyline (first and last points of polyline are equal).

PtRay: Origin of ray.

RayAxes: Direction of ray. 0 for X, 1 for Y, etc.

Returns: Number of intersections of ray with the polygon.

Description: Routine that implements "Jordan Theorem". Same as GMPolygonXYRayInter2 but does not return the first intersection info.
See also: GMPolygonPlaneInter, GMPolygonPointInclusion, GMSplitPolygonAtPlane,

4.2.235 GMPolygonXYRayInter2 (geom_bsc.c:2236)

ray polygon intersection

Jordan theorem

```
int GMPolygonXYRayInter2(const IPPolygonStruct *Pl,
                         const IrtPtType PtRay,
                         int RayAxes,
                         IPVertexStruct **FirstInterV,
                         IrtRType *FirstInterP,
                         IrtRType *AllInters)
```

Pl: To compute "Jordan Theorem" for the given ray. Can be either a polygon or a closed polyline (first and last points of polyline are equal).

PtRay: Origin of ray.

RayAxes: Direction of ray. 0 for X, 1 for Y, etc.

FirstInterV: OUT - First intersection edge, between V and VNext.

FirstInterP: OUT - First intersection location, blending V and VNext.

AllInters: OUT - if not NULL, will be updated with a vector of all intersection locations. First entry must hold number of elements in this vector.

Returns: Number of intersections of ray with the polygon.

Description: Routine that implements "Jordan Theorem": Fire a ray from a given point and find the number of intersections of a ray with the polygon, excluding the given point Pt (start of ray) itself, if on polygon boundary. The ray is fired in +X (Axes == 0) or +Y if (Axes == 1). Only the X/Y coordinates of the polygon are taken into account, i.e. the orthogonal projection of the polygon on an X/Y parallel plane (equal to polygon itself if on X/Y parallel plane...). Note that if the point is on polygon boundary, the ray should not be in its edge direction.
Algorithm:

1. 1.1. Set NumOfIntersection = 0;

   1.2. Find vertex V not on Ray level and set AlgState to its level (below or above the ray level). If none goto 3;

1.3. Mark VStart = V;

2. Do

2.1. While State(V) == AlgState do

2.1.1. V = V -> Pnext;

2.1.2. If V == VStart goto 3;

2.2. IntersectionMinX = IRIT_INFNTY;

2.3. While State(V) == ON_RAY do

2.3.1. IntersectionMin = IRIT_MIN(IntersectionMin, V -> Coord[Axes]);

2.3.2. V = V -> Pnext;

2.4. If State(V) != AlgState do

2.4.1. Find the intersection point between polygon edge VLast, V and the Ray and update IntersectionMin if lower than it.

2.4.2. If IntersectionMin is greater than Pt[Axes] increase the NumOfIntersection counter by 1.

2.5. AlgState = State(V);

2.6. goto 2.2.

3. Return NumOfIntersection;

See also: GMPolygonPlaneInter, GMSplitPolygonAtPlane,

4.2.236 GMPolygonXYRayInter3D (geom_bsc.c:2451)

ray polygon intersection

Jordan theorem

```
int GMPolygonXYRayInter3D(const IPPolygonStruct *Pl,
                          const IrtPtType PtRay,
                          int RayAxes)
```

Pl: To compute "Jordan Theorem" for the given ray.

PtRay: Origin of ray.

RayAxes: Direction of ray. 0 for X, 1 for Y, etc.

Returns: Number of intersections of ray with the polygon.

Description: Same as GMPolygonXYRayInter but for arbitrary oriented polygon. The polygon is transformed into the XY plane and then GMPolygonXYRayInter is invoked on it.

4.2.237 GMPolygonalMorphosis (pt_morph.c:32)

```
IPPolygonStruct *GMPolygonalMorphosis(const IPPolygonStruct *Pl1,
                                      const IPPolygonStruct *Pl2,
                                      IrtRType t)
```

Pl1, Pl2: The two polygonal object to meta-morph.

t: Linear blending factor: (1-t) * Pl1 + t * Pl2.

Returns: Blended polyhedra.

Description: Computes a blend of the given two polyhedra as 't Pl1 + (1-t) Pl2'. The two polyhedra are assumed to be of the same topology: same number of (ordered) polygons and same number of vertices in each corresponding polygon.

4.2.238 GMPolylineRayInter3D (geom_bsc.c:2392)

```
int GMPolylineRayInter3D(const IPPolygonStruct *Polyline,
                         const IrtPtType Pt,
                         const IrtVecType Dir,
                         IrtPtType InterPt)
```

Polyline: A polyline to check intersection with.

Pt: The ray source point.

Dir: The ray direction.

InterPt: The resulting intersection point, if any.

Returns: TRUE if intersect, FALSE otherwise.

Description: Examine if the given ray intersects the given polyline. The polyline and ray must lie in the same plane.

See also: GMSegmentRayInter,

4.2.239 GMQuadAreaPerimeterRatioWeightFunc (polyquad.c:466)

```
IrtRType GMQuadAreaPerimeterRatioWeightFunc(const CagdPolylineStruct *P,
                                            const int *VIndices,
                                            int numV)
```

P: The polygon the quad is formed from its vertices.

VIndices: Array of indices if the vertices in Pl that forms the quad.

numV: Length of VIndices array.

Returns: Value of the weight function described above.

Description: Returns the following quad weight function: function WF. * The used weight function is: (0.75 * QuadArea + 0.05 * QuadPerimeter + 0.10 * EdgesMaxMinRatio).

4.2.240 GMQuadrangulatePolygon (polyquad.c:541)

```
struct IPPolygonStruct *GMQuadrangulatePolygon(const CagdPolylineStruct *Pl,
                                               GMQuadWeightFuncType WF,
                                               void *UserData)
```

Pl: A polygon to quadrangulate.

WF: Weight function of a quad. Invalid/undesired quads should have a negative weight, where in this case the weight of the quad is infinity.

UserData: User data that can be used for the weight function.

Returns: List of quads (can contains triangles as well).

Description: Quadrangulates a closed polygon, by minimizing a given quad weight function WF. * The used weight function is: WF * (0.75 * QuadArea + 0.05 * QuadPerimeter +

0.10 * EdgesMaxMinRatio).

See also: GMQuadrangulatePolygon2,

4.2.241 GMQuadrangulatePolygon2 (polyquad.c:614)

```
struct IPPolygonStruct *GMQuadrangulatePolygon2(const struct IPPolygonStruct
                                                                    *Pl,
                                                GMQuadWeightFuncType WF,
                                                void *UserData)
```

Pl: A polygon to quadrangulate.

WF: Weight function of a quad. Invalid/undesired quads should have a negative weight, where in this case the weight of the quad is infinity.

UserData: User data that can be used for the weight function.

Returns: List of quads (can contains triangles as well).

Description: Quadrangulates a closed polygon, by minimizing a given quad weight function WF. * The used weight function is: WF * (0.75 * QuadArea + 0.05 * QuadPerimeter +

0.10 * EdgesMaxMinRatio).

See also: GMQuadrangulatePolygon,

4.2.242 GMQuadrangulatePolygonList (polyquad.c:650)

```
struct IPPolygonStruct *GMQuadrangulatePolygonList(const struct
                                                     IPPolygonStruct *PlgnList,
                                                   GMQuadWeightFuncType WF,
                                                   void *UserData)
```

PlgnList: A polygon list to quadrangulate.

WF: The weight function.

UserData: User data that can be used for the weight function.

Returns: List of Quads (might contain triangles also).

Description: Quadrangulates a closed polygon, by minimizing a given quad weight function WF. The used weight function is: WF * (0.75 * QuadArea + 0.05 * QuadPerimeter + 0.10 * EdgesMaxMinRatio)
See also: GMQuadrangulatePolygon2,

4.2.243 GMQuatAdd (quatrnn.c:231)

Quaternion

```
void GMQuatAdd(GMQuatType q1, GMQuatType q2, GMQuatType QRes)
```

q1: Left quaternion.

q2: Right quaternion.

QRes: Result quaternion.

Returns: void

Description: Adds two quatenrions.
See also: GMQuatMatToQuat,

4.2.244 GMQuatExp (quatrnn.c:418)

Quaternion

```
void GMQuatExp(IrtVecType SrcVec, GMQuatType DstQ)
```

SrcVec: Source vector.

DstQ: Detination (result) quaternion.

Returns: void

Description: Calculates the exponent quaternion of a vector.
See also: GMQuatLog,

4.2.245 GMQuatInverse (quatrnn.c:315)

Quaternion

```
void GMQuatInverse(GMQuatType SrcQ, GMQuatType DstQ)
```

SrcQ: Source quaternion.

DstQ: Destination inversed quaternion.

Returns: void

Description: Creates q^(-1) from a quaternion.
See also: GMQuatNormalize, GMQuatMatToQuat, GMQuatIsUnitQuat,

4.2.246 GMQuatIsUnitQuat (quatrnn.c:257)

Quaternion

```
int GMQuatIsUnitQuat(GMQuatType q)
```

q: Tested quaternion.

Returns: Non-zero if TRUE.

Description: Checks if a given quaternion is of unit magnitude.
See also: GMQuatMatToQuat, GMQuatNormalize,

4.2.247 GMQuatLog (quatrnn.c:382)

Quaternion

```
void GMQuatLog(GMQuatType SrcQ, IrtVecType DstVec)
```

SrcQ: Source quaternion.

DstVec: Detination (result) vector.

Returns: void

Description: Calculates the logarithm of a quaternion.
See also: GMQuatExp,

4.2.248 GMQuatMatToQuat (quatrnn.c:85)

Quaternion

```
void GMQuatMatToQuat(IrtHmgnMatType Mat, GMQuatType q)
```

Mat: Source matrix.

q: Destination quaternion.

Returns: void

Description: Transforms a matrix to a quaternion.
See also: GMQuatToMat,

4.2.249 GMQuatMatrixToAngles (quatrnn.c:561)

```
int GMQuatMatrixToAngles(IrtHmgnMatType Mat, IrtVecType *Vec)
```

Mat: Source roation matrix.

Vec: Destination angles vectors (up to 8).

Returns: The number of possible solutions (0 = no solution).

Description: Calculates the angle of rotation in each axis, from a given rotation matrix.
The rotation, being axis-dependant, must be performed in a predefined order: rotate by X, then by Y and finally by Z.
A rotation angle in the X-Y-Z order looks like this:

```
        c2*c3                 c2*s3                 -s2             0
        s1*s2*c3 - c1*s3      s1*s2*s3 + c1*c3      s1*c2           0
        c1*s2*c3 + s1*s3      c1*s2*s3 - s1*c3      c1*c2           0
        0                     0                     0               1



where c1 = cos x, c2 = cos y, c3 = cos z
      s1 = sin x, s2 = sin y, s3 = sin z



This is compared to our matrix:
        a                     b                     c               0
        d                     e                     f               0
        g                     h                     i               0
        0                     0                     0               1
```

See also: GMQuatMatrixToScale, GMQuatMatrixToTranslation,

4.2.250 GMQuatMatrixToScale (quatrnn.c:653)

```
IrtRType GMQuatMatrixToScale(IrtHmgnMatType Mat)
```

Mat: Source transformation matrix.

Returns: The uniform scale factor result.

Description: Extract the uniform scale factor from a given transformation matrix.
See also: GMQuatMatrixToAngles, GMQuatMatrixToTranslation,

4.2.251 GMQuatMatrixToTranslation (quatrnn.c:630)

Transformation

Quaternion

```
void GMQuatMatrixToTranslation(IrtHmgnMatType Mat, IrtVecType Vec)
```

Mat: Source transformation matrix.

Vec: Destination translation vector.

Returns: void

Description: Extract the translation vector from a given transformation matrix.
See also: GMQuatMatrixToScale, GMQuatRotMatrixToAngles,

4.2.252 GMQuatMatrixToVector (quatrnn.c:696)

Transformation

Quaternion

```
int GMQuatMatrixToVector(IrtHmgnMatType Mat, GMQuatTransVecType TransVec)
```

Mat: Source transformation matrix.

TransVec: The result transformation parameters vector.

Returns: TRUE only if the input matrix is a transformation matrix.

Description: Extract the transformation parameters vector from a given transformation matrix.
A transformation vector contains the rotation angles in all 3 axis, the translation in all 3 axis and a uniform scaling factor.
Since there are many ways to combine these parameters, we chose the following order: To create a transformation matrix out of a transformation vector - first create a rotation matrix (rotate by X then by Y and then by Z), then create a uniform scaling matrix and finally create a translation matrix.
Apply roation, then scale and finally translation to obtain the original transformation matrix.
To create a transformation vector out of a transformation matrix, we simply do it all in reverse : extract and cancel transformation effects, extract and cancel scaling effects and then extract roation effects.
See also: GMQuatMatrixToScale, GMQuatMatrixToTranslation, GMQuatMatrixToAngles,

4.2.253 GMQuatMul (quatrnn.c:202)

Quaternion

```
void GMQuatMul(GMQuatType q1, GMQuatType q2, GMQuatType QRes)
```

q1: Left quaternion.

q2: Right quaternion.

QRes: Result quaternion.

Returns: void

Description: Mutliplies two quatenrions. Order of arguments counts.
See also: GMQuatMatToQuat,

4.2.254 GMQuatNormalize (quatrnn.c:279)

Quaternion

```
void GMQuatNormalize(GMQuatType q)
```

q: quaternion.

Returns: void

Description: Normalizes a quaternion into a unit size quaternion (as a 4 vector)
See also: GMQuatIsUnitQuat, GMQuatInverse, GMQuatIsUnitQuat,

4.2.255 GMQuatPow (quatrnn.c:455)

Quaternion

```
void GMQuatPow(GMQuatType MantisQ, IrtRType Expon, GMQuatType DstQ)
```

MantisQ: Mantisa quaternion.

Expon: Real exponent.

DstQ: Detination (result) quaternion.

Returns: void

Description: Calculates the quaternion to the power of a real exponenet.
See also: GMQuatLog, GMQuatExp,

4.2.256 GMQuatRotateVec (quatrnn.c:345)

Quaternion

```
void GMQuatRotateVec(IrtVecType OrigVec, GMQuatType RotQ, IrtVecType DestVec)
```

OrigVec: Original (source) vector.

RotQ: Rotation quaternion.

DestVec: Detination (rotated) vector.

Returns: void

Description: Rotates a vector using a rotation quaternion.
See also: GMQuatMatToQuat,

4.2.257 GMQuatRotationToQuat (quatrnn.c:143)

Quaternion

```
void GMQuatRotationToQuat(IrtRType Xangle,
                          IrtRType Yangle,
                          IrtRType Zangle,
                          GMQuatType q)
```

Xangle: Roation angle around X axis

Yangle: Roation angle around Y axis

Zangle: Roation angle around Z axis

q: Destination quaternion.

Returns: void

Description: Creates a quaternion from an arbitrary rotation in X-Y-Z order.
See also: GMQuatToMat, GMQuatToRotation,

4.2.258 GMQuatToMat (quatrnn.c:34)

Quaternion

```
void GMQuatToMat(GMQuatType q, IrtHmgnMatType Mat)
```

q: Source quaternion.

Mat: Destination matrix.

Returns: void

Description: Transforms a quaternion to a matrix.
See also: GMQuatMatToQuat, GMQuatNormalize,

4.2.259 GMQuatToRotation (quatrnn.c:175)

Quaternion

```
void GMQuatToRotation(GMQuatType q, IrtVecType *Angles, int *NumSolutions)
```

q: Rotation quaternion.

Angles: All possible rotation angles (up to 8).

NumSolutions: Pointer to buffer that holds number of solutions found.

Returns: void

Description: Finds rotation angles in X-Y-Z order from a given quaternion representation.
See also: GMQuatToMat, GMQuatRotationToQuat,

4.2.260 GMQuatVecToRotMatrix (quatrnn.c:827)

Transformation

Quaternion

```
void GMQuatVecToRotMatrix(GMQuatTransVecType TransVec,
                          IrtHmgnMatType RotMatrix)
```

TransVec: The source transformation parameters vector.

RotMatrix: The result rotation matrix.

Returns: void

Description: Extracts a rotation matrix from a transformation parameters vector. The rotation, being axis-dependant, must be performed in a predefined order: rotate by X, then by Y and finally by Z.
See also: GMQuatVecToTransMatrix, GMQuatVecToScaleMatrix,

4.2.261 GMQuatVecToScaleMatrix (quatrnn.c:798)

Transformation

Quaternion

```
void GMQuatVecToScaleMatrix(GMQuatTransVecType TransVec,
                            IrtHmgnMatType ScaleMatrix)
```

TransVec: The source transformation parameters vector.

ScaleMatrix: The result scale matrix.

Returns: void

Description: Extracts a scale matrix from a transformation parameters vector.
See also: GMQuatVecToRotMatrix, GMQuatVecToTransMatrix,

4.2.262 GMQuatVecToTransMatrix (quatrnn.c:868)

Transformation

Quaternion

```
void GMQuatVecToTransMatrix(GMQuatTransVecType TransVec,
                            IrtHmgnMatType TransMatrix)
```

TransVec: The source transformation parameters vector.

TransMatrix: The result translation matrix.

Returns: void

Description: Extracts a translation matrix from a transformation parameters vector.
See also: GMQuatVecToRotMatrix, GMQuatVecToScaleMatrix,

4.2.263 GMQuatVectorToMatrix (quatrnn.c:765)

Transformation

Quaternion

```
void GMQuatVectorToMatrix(GMQuatTransVecType TransVec, IrtHmgnMatType Mat)
```

TransVec: The source transformation parameters vector.

Mat: The result transformation matrix.

Returns: void

Description: Converts a transformation parameters vector to a transformation matrix.
A transformation vector contains the rotation angles in all 3 axis, the translation in all 3 axis and a uniform scaling factor.
Since there are many ways to combine these parameters, we chose the following order: To create a transformation matrix out of a transformation vector - first create a rotation matrix (rotate by X, then by Y, and then by Z), then create a uniform scaling matrix and finally create a translation matrix.
Apply rotation, then scale and finally translation to obtain the original transformation matrix.
To create a transformation vector out of a transformation matrix, we simply do it all in reverse: extract and cancel transformation effects, extract and cancel scaling effects and then extract rotation effects.
See also: GMQuatVecToScaleMatrix, GMQuatVecToRotMatrix, GMQuatVecToTransMatrix,

4.2.264 GMRayCnvxPolygonInter (geomat3d.c:1682)

```
int GMRayCnvxPolygonInter(const IrtPtType RayOrigin,
                          const IrtVecType RayDir,
                          const IPPolygonStruct *Pl,
                          IrtPtType InterPoint)
```

RayOrigin: Starting point of ray.

RayDir: Direction of ray.

Pl: Convex polygon to test against ray for intersection.

InterPoint: Resulting intersection point.

Returns: TRUE if successful, FALSE otherwise.

Description: Tests if the given ray intersects the given convex polygon.
See also: GMPointFromLinePlane, GMPointInsideCnvxPolygon,

4.2.265 GMRayCnvxPolygonListInter (geomat3d.c:1714)

```
int GMRayCnvxPolygonListInter(const IrtPtType RayOrigin,
                              const IrtVecType RayDir,
                              const IPPolygonStruct *Plgns,
                              IrtPtType InterPoint)
```

RayOrigin: Starting point of the ray.

RayDir: Direction of the ray.

Plgns: Polygon list to test against ray for intersection.

InterPoint: Resulting intersection point.

Returns: TRUE if successful, FALSE otherwise.

Description: Tests if the given ray intersects the given convex polygon list, Plgns.
See also: GMRayCnvxPolygonInterm,

4.2.266 GMRayCnvxPolygonListInter2 (geomat3d.c:1752)

```
int GMRayCnvxPolygonListInter2(const IrtPtType RayOrigin,
                               const IrtVecType RayDir,
                               const IPPolygonStruct *Plgns,
                               IrtPtType *AllInterPts,
                               const IPPolygonStruct **AllInterPls)
```

RayOrigin: Starting point of the ray.

RayDir: Direction of the ray.

Plgns: Polygon list to test against ray for intersection.

AllInterPts: All intersection points. A vector large enough to hold all intersections.

AllInterPls: All intersection polygons. A vector large enough to hold all intersecting polygons. Can be NULL to ignore.

Returns: Number of intersections, zero for none.

Description: Tests if the given ray intersects the given convex polygon list, Plgns.
See also: GMRayCnvxPolygonInter, GMRayCnvxPolygonListInter,

4.2.267 GMRefineDeformedTriangle (poly_pts.c:2500)

```
int GMRefineDeformedTriangle(IPPolygonStruct *Pl,
                             GMPointDeformVrtxFctrFuncType DeformVrtxFctrFunc,
                             GMPointDeformVrtxDirFuncType DeformVrtxDirFunc,
                             void *UserData,
                             IrtRType DeviationTol,
                             IrtRType MaxEdgeLen)
```

Pl: Triangle to refine if necessary.

DeformVrtxFctrFunc: Function to evaluate the deformation amount factor of a given vertex.

DeformVrtxDirFunc: Function to evaluate the deformation vector (direction and amount) of a given vertex.

UserData: Optional auxiliary data reference to be transfered to DeformVrtxFctrFunc and DeformVrtxDirFunc.

DeviationTol: Of deformation approximation.

MaxEdgeLen: to allow in a triangle.

Returns: TRUE if triangle underwent refinement, FALSE otherwise.

Description: Given a triangle in some deformation function and a function to evaluate deformation amount per vertex, refine the triangle as necessary to make the deformation be accurate within tolerance Tol. Only the vertices and mid-edge points on triangle are examined. Newly created (refined) triangles are appended in place between Pl and Pl -> Pnext. Note Pl will also be modified in place if refinement occurs.
See also: GMRefineDeformedTriangle2,

4.2.268 GMRefineDeformedTriangle2 (poly_pts.c:2285)

```
int GMRefineDeformedTriangle2(IPPolygonStruct *Pl,
                              GMPointDeformVrtxFctrFuncType DeformVrtxFctrFunc,
                              void *UserData,
                              IrtBType Ref12,
                              IrtBType Ref23,
                              IrtBType Ref31)
```

Pl: Triangle to refine if necessary.

DeformVrtxFctrFunc: Function to evaluate the deformation amount factor of a given vertex.

UserData: Optional auxiliary data reference to be transfered to DeformVrtxFctrFunc and DeformVrtxDirFunc.

Ref12, Ref23, Ref31: Booleans to set which edge must be refined.

Returns: TRUE if triangle underwent refinement, FALSE otherwise.

Description: Given a triangle Pl, divide and refine it along the edges as set by Refij. The result can be between one (no refinement) and four (all edges are defined) triangles that are substituted in place in Pl and before Pl -> pnext.
See also: GMRefineDeformedTriangle,

4.2.269 GMRegularizePolyModel (poly_pts.c:1151)

```
IPObjectStruct *GMRegularizePolyModel(const IPObjectStruct *PObj,
                                      int SplitCollinear,
                                      IrtRType MinRefineDist)
```

PObj: A polygonal object to regularize.

SplitCollinear: TRUE to also split polygons at collinear edges.

MinRefineDist: new vertices closer than this distance to their neighbors will NOT be added.

Returns: Regularized object.

Description: Regularize a polygonal model by eliminating all T junction in the polygonal mesh.

4.2.270 GMReparamTexture (texture_manage.c:257)

```
int GMReparamTexture(IPPolygonStruct *Pls,
                     const IrtRType UVCrntDmn[4],
                     const char *UVNewDmnStr)
```

Pls: Polygons to update, in place.

UVCrntDmn: The current domain of freeform, from which the polygons were created, as (UMin, VMin, UMax, VMax).

UVNewDmnStr: The new domain to map the parametrization to, as a string with "UMin VMin UMax VMax".

Returns: TRUE if successful.

Description: Reparameterize (affine transform) the uv coordinates in the polygons to match, the domain defined in UVNewDmnStr as (UMin, VMin, UMax, VMax).
See also: GMUniteTextures,

4.2.271 GMReparamTexture2 (texture_manage.c:291)

```
int GMReparamTexture2(IPPolygonStruct *Pls,
                      const IrtRType UVCrntDmn[4],
                      const IrtRType UVNewDmn[4])
```

Pls: Polygons to update, in place.

UVCrntDmn: The current domain of freeform, from which the polygons were created, as (UMin, VMin, UMax, VMax).

UVNewDmn: The new domain to map the parametrization to, as (UMin, VMin, UMax, VMax).

Returns: TRUE if successful.

Description: Reparametrize (affine transform) the uv coordinates in the polygons to match, the domain defined in UVNewDmn as (UMin, VMin, UMax, VMax).
See also: GMUniteTextures,

4.2.272 GMSLerp (geom_bsc.c:324)

```
void GMSLerp(const IrtVecType V1,
             const IrtVecType V2,
             IrtRType t,
             IrtVecType VOut)
```

V1, V2: The two vectors to interpolate.

t: The amount to interpolate (in the range [0,1]).

VOut: Output parameter. The resulting interpolated vector.

Returns: void

Description: Compute an angle-linear interpolation between two unit vectors (as if rotating vectors on a unit sphere). The Slerp function is based on: Animating rotation with quaternion curves, K. Shoemake, 1985.

4.2.273 GMScanConvertTriangle (scancnvt.c:33)

```
void GMScanConvertTriangle(int Pt1[2],
                           int Pt2[2],
                           int Pt3[2],
                           GMScanConvertApplyFuncType ApplyFunc)
```

Pt1, Pt2, Pt3: The three coordinates of the triangle.

ApplyFunc: The function that will be invoked on every pixel that is visited in this triangle.

Returns: void

Description: Visits all pixels of the given triangle and invokes ApplyFunc on each such pixel.

4.2.274 GMSegmentRayInter (geom_bsc.c:2352)

```
int GMSegmentRayInter(const IrtPtType v1,
                      const IrtPtType v2,
                      const IrtPtType Pt,
                      const IrtPtType Dir,
                      IrtPtType InterPt,
                      IrtRType Eps)
```

v1: The source point of the line segment.

v2: The end point of the line segemnt.

Pt: The source point of the ray.

Dir: The ray direction.

InterPt: Resulting intersection point.

Eps: A epsilon tolerance.

Returns: TRUE if intersect, FALSE otherwise.

Description: Assuming the given line segment and the ray are co-planar, the function returns the intersection point between them. Otherwise it will return a point on the line segment that is the closed to the ray up to the specified Eps tolerance.
See also: GMPolylineGeneralRayInter,

4.2.275 GMSilExtractBndry (poly_sil.c:618)

boundary

silhouette.

```
IPObjectStruct *GMSilExtractBndry(IPObjectStruct *PObjReg)
```

PObjReg: Object to extract the Boundary.

Returns: The Boundary Object.

Description: Generates the boundary of the polyhdral object PObjReg, assumed already regularized.
See also: GMSilExtractSil, GMSilExtractSilDirect, BoolGenAdjacencies, GMSilExtractDiscont,

4.2.276 GMSilExtractDiscont (poly_sil.c:543)

discontinuities.

```
IPObjectStruct *GMSilExtractDiscont(IPObjectStruct *PObjReg,
                                    IrtRType MinAngle)
```

PObjReg: Polyhedral Object to generate the silhouette for.

MinAngle: Minimal dihedral angle between adjacent polygons to consider as discontinuity. In radians.

Returns: Discontinuities Object.

Description: Generates edges along adjacent polygons with a dihedral angle of more than MinAngle degrees.
See also: GMSilExtractBndry, GMSilExtractSil, GMSilExtractSilDirect2, GMSilExtractSilDirect,

4.2.277 GMSilExtractSil (poly_sil.c:671)

silhouette.

```
IPObjectStruct *GMSilExtractSil(VoidPtr PrepSils, IrtHmgnMatType ViewMat)
```

PrepSils: Associated silhouette processing data structure of a polygonal object to generate its silhouettes.

ViewMat: View Matrix.

Returns: Silhouette Object.

Description: Generates the silhouette of an object which has been already preprocessed and is associated with a grid structure.
See also: GMSilPreprocessPolys, GMSilOrigObjAlive, GMSilExtractSilDirect, , GMSilExtractSilDirect2,

4.2.278 GMSilExtractSilDirect (poly_sil.c:335)

silhouette.

```
IPObjectStruct *GMSilExtractSilDirect(IPObjectStruct *PObjReg,
                                      IrtHmgnMatType ViewMat)
```

PObjReg: Polyhedral Object to generate the silhouette for.

ViewMat: View Matrix.

Returns: Silhouette Object.

Description: Generates the silhouette from a polyhedral object, assumed already regularized with the straight forward method.
See also: GMSilExtractBndry, GMSilExtractSil, GMSilExtractSilDirect2, GMSilExtractDiscont,

4.2.279 GMSilExtractSilDirect2 (poly_sil.c:491)

silhouette.

```
IPObjectStruct *GMSilExtractSilDirect2(IPObjectStruct *PObjReg,
                                       IrtHmgnMatType ViewMat)
```

PObjReg: Polyhedral Object to generate the silhouette for. Assumed to be regular and hold triangles only.

ViewMat: View Matrix.

Returns: Silhouette Object.

Description: Generates the silhouette from a polyhedral object consisting of only triangles, and further assumed already regularized. The silhouettes generated by this function are interior to the triangles and are computed for each triangle individually, based on its vertices' normals.
See also: GMSilExtractBndry, GMSilExtractSil, GMSilExtractSilDirect,

4.2.280 GMSilOrigObjAlive (poly_sil.c:772)

```
int GMSilOrigObjAlive(int ObjAlive)
```

ObjAlive: If TRUE, assumes original object remains valid throughout.

Returns: Original vaolue of original object alive.

Description: If TRUE, this module is allowed to assume that the original polygonal object is alive while silhouette queries are conducted. Default to FALSE. Setting it to TRUE would allow certain optimization as well as the propagation of attributes of vertices from the original object to the detected silhouette edges.
See also: GMSilExtractSil,

4.2.281 GMSilPreprocessPolys (poly_sil.c:222)

silhouette

boundary.

```
VoidPtr GMSilPreprocessPolys(IPObjectStruct *PObjReg, int n)
```

PObjReg: Regular polyhedral Object.

n: Subdivision resolution of the Grid (n by n).

Returns: Grid Structure of preprocessing data structure.

Description: Generates the Grid Structure of a polyhedral object. This is the preprocessing stage to the silhouette extraction method. Polyhedra is assumed regular, and has adjacency information.
See also: GMSilExtractBndry, GMSilExtractSil, BoolGenAdjacencies,

4.2.282 GMSilPreprocessRefine (poly_sil.c:294)

silhouette

boundary.

```
int GMSilPreprocessRefine(VoidPtr PrepSils, int n)
```

PrepSils: Preprocessing data structure of silhouettes to refine to a new resolution n.

n: New subdivision resolution of the grid.

Returns: TRUE if the grid was updated, FALSE otherwise.

Description: Compute a new Grid if the subdivision resolution has been changed.

4.2.283 GMSilProprocessFree (poly_sil.c:731)

allocation

```
void GMSilProprocessFree(VoidPtr PrepSils)
```

PrepSils: To free.

Returns: void

Description: Frees the proprocessing data structure of silhouettes.

4.2.284 GMSolveCubicEqn (geom_bsc.c:3571)

```
int GMSolveCubicEqn(IrtRType A, IrtRType B, IrtRType C, IrtRType *Sols)
```

A, B, C: The equation's coefficients as x^3 + A x^2 + B x + C = 0.

Sols: Where to place the solutions. At most three.

Returns: Number of real solutions.

Description: Computes the solutions, if any, of the given cubic equation. Only real solutions are considered.
See also: GMSolveQuadraticEqn, GMSolveQuadraticEqn2, GMSolveCubicEqn2, , GMSolveQuarticEqn,

4.2.285 GMSolveCubicEqn2 (geom_bsc.c:3667)

```
int GMSolveCubicEqn2(IrtRType A,
                     IrtRType B,
                     IrtRType C,
                     IrtRType *RSols,
                     IrtRType *ISols)
```

A, B, C: The coefficients of the cubic polynomial.

RSols, ISols: Each pair (RSols[i], ISols[i]) is the complex root(i)

Returns: The number of REAL solutions of the cubic polynomial.

Description: Calculates the three roots (complex & real) of the cubic equation x^3 + Ax^2 + Bx + C = 0 Note: Cubic equations have at least one real root; this function always calculates the real root first, and the other two (possibly complex) later. This order of filling RSols & ISols is CRUCIAL for GMSolveQuarticEqn()
See also: GMSolveCubicEqn, GMSolveQuadraticEqn, GMSolveQuadraticEqn2, , GMSolveQuarticEqn,

4.2.286 GMSolveQuadraticEqn (geom_bsc.c:3420)

```
int GMSolveQuadraticEqn(IrtRType A, IrtRType B, IrtRType *Sols)
```

A, B: The equation's coefficients as x^2 + A x + B = 0.

Sols: Where to place the solutions. At most two.

Returns: Number of real solutions.

Description: Computes the solutions, if any, of the given quadratic equation. Only real solutions are considered.
See also: GMSolveCubicEqn, GMSolveCubicEqn2, GMSolveQuadraticEqn2, GMSolveQuarticEqn,

4.2.287 GMSolveQuadraticEqn2 (geom_bsc.c:3463)

```
int GMSolveQuadraticEqn2(IrtRType B,
                         IrtRType C,
                         IrtRType *RSols,
                         IrtRType *ISols)
```

B, C: The coefficients of the quadratic polynomial.

RSols, ISols: Solutions such that each pair RSols[i], ISols[i] is the complex root(i).

Returns: The number of REAL solutions of the polynomial.

Description: Calculates the two square roots of the quadratic equation: x^2 + Bx + C = 0
See also: GMSolveCubicEqn, GMSolveCubicEqn2, GMSolveQuadraticEqn, GMSolveQuarticEqn,

4.2.288 GMSolveQuarticEqn (geom_bsc.c:3833)

```
int GMSolveQuarticEqn(IrtRType a,
                      IrtRType b,
                      IrtRType c,
                      IrtRType d,
                      IrtRType *Sols)
```

a, b, c, d: The coefficients of the quartic polynomial.

Sols: The real roots of the polynomial.

Returns: The number of REAL solutions of the polynomial.

Description: Computes the (upto) four real roots of the quartic equation

```
x^4 + Ax^3 + Bx^2 + Cx + D = 0
```

Note 1 —— In order to avoid building a library for complex numbers arithmetics, two arrays are used ISols[] and RSols[], where each RSols[i] and ISols[i], represent a complex number, and so calculation where made on the fly; Anyway, some of these calculations where performed in a specific way to reduce errors of double-precision nature. (especially calculating square roots of complex numbers).
Note 2. ———- In the case of Cubic and Quadratic equations, the number of real solutions is determined via the value of D (the descrimenant); As such, the number of real solutions is easier to depict. However, Euler's solution for quartic equations manipulates all the three solutions of the cubic (real and complex), hoping to find some real roots by eliminating the imaginary part of the complex solutions. This, however, is a weakness of dependency upon the accuracy of the numbers' representation as a double-precision floating point.
See also: GMSolveCubicEqn, GMSolveCubicEqn2, , GMSolveQuadraticEqn, GMSolveQuadraticEqn2,

4.2.289 GMSphConeGetPtsDensity (sph_cone.c:455)

```
const IrtVecType *GMSphConeGetPtsDensity(int *n)
```

n: Number of pts to be distributed on the sphere (approximately). n will be updated to the actual size of the returned vector of unit vectors.

Returns: A vector of n unit vectors equally spead over the unit sphere.

Description: Returns unit vectors on the unit sphere spread uniformly, in a number that closely approximate given n.
See also: GMSphConeSetConeDensity,

4.2.290 GMSphConeQuery2GetVectors (sph_cone.c:688)

```
void GMSphConeQuery2GetVectors(VoidPtr SphConePtr,
                               GMSphConeQueryDirFuncType SQQuery,
                               GMSphConeQueryCallBackFuncType SQFunc)
```

SphConePtr: Processed data struct for efficient Direction querying.

SQQuery: Query function to invoke.

SQFunc: Function to invoke on detected elements.

Returns: void

Description: Invokes SQFunc with all vectors in the preprocessed data set that the cone containing them satisfy the query function SQQuery. Each such vector is guaranteed to be invoked once only.
See also: GMSphConeQueryInit, GMSphConeQueryFree, GMSphConeSetConeDensity, GMSphConeQuery2GetVectors,

4.2.291 GMSphConeQueryFree (sph_cone.c:589)

```
void GMSphConeQueryFree(VoidPtr SphConePtr)
```

SphConePtr: Cone data structure to free.

Returns: void

Description: Release all data allocated by GMSphConeQueryInit function.
See also: GMSphConeQueryInit, GMSphConeQueryGetVectors, GMSphConeSetConeDensity,

4.2.292 GMSphConeQueryGetVectors (sph_cone.c:628)

```
void GMSphConeQueryGetVectors(VoidPtr SphConePtr,
                              IrtVecType Dir,
                              IrtRType Angle,
                              GMSphConeQueryCallBackFuncType SQFunc)
```

SphConePtr: Processed data struct for efficient Direction querying.

Dir: Direction to query.

Angle: Angular span to query.

SQFunc: Function to invoke on detected elements.

Returns: void

Description: Invokes SQFunc with all vectors in the preprocessed data set that are at most Angle degrees for the prescibed Dir. Each such vector is guaranteed to be invoked once only.
See also: GMSphConeQueryInit, GMSphConeQueryFree, GMSphConeSetConeDensity, GMSphConeQuery2GetVectors,

4.2.293 GMSphConeQueryInit (sph_cone.c:490)

```
VoidPtr GMSphConeQueryInit(IPObjectStruct *PObj, int n)
```

PObj: A point list object to preprocess.

n: Number of cones to distribute on the sphere (approximately).

Returns: Processed data structure for fast cone queries.

Description: Proeprocess the given set of points into the different bounding cones of the unit sphere.
See also: GMSphConeQueryFree, GMSphConeQueryGetVectors, GMSphConeSetConeDensity,

4.2.294 GMSphereWith3Pts (ms_spher.c:300)

```
int GMSphereWith3Pts(IrtE3PtStruct *Pts, IrtRType *Center, IrtRType *RadiusSqr)
```

Pts: The set of point to compute their sphere.

Center: Of computed sphere.

RadiusSqr: Of computed Sphere.

Returns: TRUE if successful, FALSE otherwise.

Description: Given three points, compute the sphere through the set of points. Initially, the three points are rotated to a plane paraellel to XY-plane and then using GMCircleFrom3Points, the circle through them is computed. The center is then rotated back to form the center of the sphere in 3D.
See also: GMCircleFrom3Points,

4.2.295 GMSphereWith4Pts (ms_spher.c:359)

```
int GMSphereWith4Pts(IrtE3PtStruct *Pts, IrtRType *Center, IrtRType *RadiusSqr)
```

Pts: The set of point to compute their sphere.

Center: Of computed sphere.

RadiusSqr: Of computed Sphere.

Returns: TRUE if successful, FALSE otherwise.

Description: Given four points, compute the sphere through the set of points. It is identified by solving the following equidistant conditions.

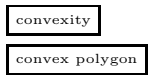
```
< P - P1, P - P1 > = < P - P2, P - P2 >,
< P - P1, P - P1 > = < P - P3, P - P3 >,
< P - P1, P - P1 > = < P - P4, P - P4 >,
```

or,

```
2(P2 - P1) P = P2^2 - P1^2,
2(P3 - P1) P = P3^2 - P1^2,
2(P4 - P1) P = P4^2 - P1^2.
```

We can solve for P (the sphere's center) using the Cramer's rule.
See also: GMMinSpanSphere,

4.2.296 GMSplitNonConvexPoly (convex.c:573)




```
IPPolygonStruct *GMSplitNonConvexPoly(IPPolygonStruct *Pl,
                                      int HandleNormals)
```

Pl: Non convex polygon to split into convex ones.

HandleNormals: TRUE to also handle normals.

Returns: A list of convex polygons resulting from splitting up Pl.

Description: Routine to split non convex polygon into a list of convex ones.

1. Search for non convex corner. If not found stop - polygon is convex. Otherwise let the non convex corner found be V(i).
2. Fire a ray from V(i) in the opposite direction to V(i-1). Find the closest intersection of the ray with polygon boundary P.
3. Split the polygon into two at V(i)-P edge and push the two new polygons on the GlblList.
4. Goto 1.

See also: GMConvertPolysToTriangles, GMConvexPolyObject, GMConvexPolyObjectN, , GMIsConvexPolygon,

4.2.297 GMSplitPolyInPlaceAt2Vertices (poly_pts.c:1558)

```
IPPolygonStruct *GMSplitPolyInPlaceAt2Vertices(IPPolygonStruct *Pl,
                                               IPVertexStruct *V1,
                                               IPVertexStruct *V2)
```

Pl: Convex polygon to split into two, in place.

V1: First Vertex to split Pl at.

V2: Second Vertex to split Pl at.

Returns: The second half of the splitted polygon (first half is returned, in place, in Pl). This function returns a NULL if split failed due to the fact that the polygon degenerated into a line. Pl is not affected if NULL is returned. The second polygon is added as next to the first polygon.

Description: Splits the given convex polygon, in place, into two, returning second half of the polygon while updating Pl to hold the first half. The second returned half is also chained after Pl (as Pl -> Pnext).
See also: GMSplitPolyInPlaceAtVertex,

4.2.298 GMSplitPolyInPlaceAtVertex (poly_pts.c:1481)

```
IPPolygonStruct *GMSplitPolyInPlaceAtVertex(IPPolygonStruct *Pl,
                                            IPVertexStruct *VHead)
```

Pl: Convex polygon to split into two.

VHead: Vertex to split Pl at.

Returns: The second half of the splitted polygon (first half is returned, in place, in Pl). This function returns a NULL if split failed due to the fact that the polygon degenerated into a line. Pl is not affected if NULL is returned.

Description: Splits the given convex polygon, in place, into two, returning second half of the polygon while updating Pl to hold the first half. Polygon is split so that VHead is on border between the two polygons.
See also: GMSplitPolyInPlaceAt2Vertices,

4.2.299 GMSplitPolygonAtPlane (geom_bsc.c:1806)

```
int GMSplitPolygonAtPlane(IPPolygonStruct *Pl, const IrtPlnType Pln)
```

Pl: Polygon to split if interestes plane Pln.

Pln: Plane to split polygon Pl at. Assumed normalized normal vector in Pln.

Returns: TRUE if intersects, FALSE otherwise.

Description: Split the given convex polygon where it intersects the given plane. Pl is updated to in inside potion (where the normal is point into) and Pl -> Pnext will hold the second half.
See also: GMPolygonRayInter, GMPolygonPlaneInter,

4.2.300 GMSplitPolysAtCollinearVertices (poly_pts.c:1407)

```
IPPolygonStruct *GMSplitPolysAtCollinearVertices(IPPolygonStruct *Pls)
```

Pls: List of polygons to split at collinear edges.

Returns: New list of polygons with no collinear adjacent edges.

Description: Splits the given polygons in vertices that connect two adjacent collinear edges. Polygons are assumed convex other than this collinearity conditions.

4.2.301 GMSrfBilinearFit (analyfit.c:48)

```
IrtPtType *GMSrfBilinearFit(IrtPtType *ParamDomainPts,
                            IrtPtType *EuclideanPts,
                            int FirstAtOrigin,
                            int NumPts,
                            IrtPtType *FitPts)
```

ParamDomainPts: Array of UV points prescribing the parametric values.

EuclideanPts: Array of XYZ points defining the Euclidean values of the ParamDomainPts with obviously the same order.

FirstAtOrigin: If TRUE, the first points is set to be at U = V = 0.

NumPts: Number of points in ParamDomainPts and EuclideanPts.

FitPts: Array of four points values, A, B, C, D. Will hold the returned value.

Returns: Array of four points values, A, B, C, D. Same as Fit Pts.

Description: Fits a bilinear surface to the set of given points as F(u,v) = A + B * u + C * v + D * u * v, A,B,C,D points in R^3.

See also: GMSrfQuadricFit,

4.2.302 GMSrfCubicQuadOnly (analyfit.c:345)

```
IrtPtType *GMSrfCubicQuadOnly(IrtPtType *ParamDomainPts,
                              IrtPtType *EuclideanPts,
                              int FirstAtOrigin,
                              int NumEucDim,
                              int NumPts,
                              IrtPtType *CubicData)
```

ParamDomainPts: Array of UV points prescribing the parametric values.

EuclideanPts: Array of XYZ points defining the Euclidean values of the ParamDomainPts with obviously the same order.

FirstAtOrigin: If TRUE, the first points is set to be at U = V = 0.

NumEucDim: Number of Euclidean dimension. 1 for scalar surface and up to 3 for parametric surface in R^3.

NumPts: Number of points in ParamDomainPts and EuclideanPts.

CubicData: The fitted cubic data. Should be given as a vector of 10 reals.

Returns: Array of 10 point values, A,B,C,D,E,F,G,H,I,J in order, where A = B = C = 0 always. Same as CubicData.

Description: Fits a cubic surface (cubic and quad terms only) to the set of given points as F(u,v) = A + B * u + C * v + D * u^2 + E * u * v + F * v^2 + G * u^3 + H * u^2 * v + I * u * v^2 + J * v^3, A,B,C,D,E,F,G,H,I,J points in R^3.

See also: GMSrfBilinearFit, GMSrfQuadricFit, GMSrfQuadricQuadOnly,

4.2.303 GMSrfQuadricFit (analyfit.c:142)

```
IrtPtType *GMSrfQuadricFit(IrtPtType *ParamDomainPts,
                           IrtPtType *EuclideanPts,
                           int FirstAtOrigin,
                           int NumPts,
                           IrtPtType *FitPts)
```

ParamDomainPts: Array of UV points prescribing the parametric values.

EuclideanPts: Array of XYZ points defining the Euclidean values of the ParamDomainPts with obviously the same order.

FirstAtOrigin: If TRUE, the first points is set to be at U = V = 0.

NumPts: Number of points in ParamDomainPts and EuclideanPts.

FitPts: Array of six point values, A,B,C,D,E,F in order. Will hold the returned value.

Returns: Array of six point values, A,B,C,D,E,F in order. Same as Fit Pts.

Description: Fits a quadric surface to the set of given points as F(u,v) = A + B * u + C * v + D * u * u + E * u * v + F * v * v, A,B,C,D,E,F points in R^3.

See also: GMSrfBilinearFit, GMSrfQuadricQuadOnly, GMSrfCubicQuadOnly,

4.2.304 GMSrfQuadricQuadOnly (analyfit.c:242)

```
IrtPtType *GMSrfQuadricQuadOnly(IrtPtType *ParamDomainPts,
                                IrtPtType *EuclideanPts,
                                int FirstAtOrigin,
                                int NumEucDim,
                                int NumPts,
                                IrtPtType *QuadData)
```

ParamDomainPts: Array of UV points prescribing the parametric values.

EuclideanPts: Array of XYZ points defining the Euclidean values of the ParamDomainPts with obviously the same order.

FirstAtOrigin: If TRUE, the first points is set to be at U = V = 0.

NumEucDim: Number of Euclidean dimension. 1 for scalar surface and up to 3 for parametric surface in R^3.

NumPts: Number of points in ParamDomainPts and EuclideanPts.

QuadData: The returned fitted quadratic data. Should be given as a vector of 6 reals.

Returns: Array of 6 point values, A,B,C,D,E,F in order, where A = B = C = 0 always. Same as QuadData.

Description: Fits a quadric surface (quad terms only) to the set of given points as F(u,v) = A + B * u + C * v + D * u * u + E * u * v + F * v * v, A,B,C,D,E,F points in R^3.
See also: GMSrfBilinearFit, GMSrfQuadricFit, GMSrfCubicQuadOnly,

4.2.305 GMSubButterfly (sbdv_srf.c:156)

```
IPObjectStruct *GMSubButterfly(const IPObjectStruct *OriginalObj,
                               IrtRType ButterflyWCoef)
```

OriginalObj: A pointer to the original polygonal object.

ButterflyWCoef: The scalar butterfly blending coefficient.

Returns: pointer to refined polygonal object after subdivision.

Description: Refines a polygonal object according to Butterfly subdivision rules. One iteration is performed.

4.2.306 GMSubCatmullClark (sbdv_srf.c:102)

```
IPObjectStruct *GMSubCatmullClark(const IPObjectStruct *OriginalObj)
```

OriginalObj: A pointer to the original polygonal object.

Returns: Pointer to refined polygonal object after subdivision.

Description: Refines a polygonal object according to Catmull-Clark subdivision rules. One iteration is performed.

4.2.307 GMSubLoop (sbdv_srf.c:128)

```
IPObjectStruct *GMSubLoop(const IPObjectStruct *OriginalObj)
```

OriginalObj: A pointer to the original polygonal object.

Returns: Pointer to refined polygonal object after subdivision.

Description: Refines a polygonal object according to Loop subdivision rules. One iteration is performed.

4.2.308 GMTransObjSetAnimCrvUpdateFunc (geomat3d.c:412)

```
GMTransObjUpdateAnimCrvsFuncType GMTransObjSetAnimCrvUpdateFunc(
                                 GMTransObjUpdateAnimCrvsFuncType AnimUpdateFunc)
```

AnimUpdateFunc: New animation crvs update function for obj transform.

Returns: Old function

Description: Sets the function to update the animation curves to work properly after the applied transformation Mat to the parent object whose PAnim are his.
See also: GMTransformObject, GMTransObjSetUpdateFunc, GMTransObjUpdateAnimCrvs,

4.2.309 GMTransObjSetUpdateFunc (geomat3d.c:381)

```
GMTransObjUpdateFuncType GMTransObjSetUpdateFunc(GMTransObjUpdateFuncType
                                                                  UpdateFunc)
```

UpdateFunc: New call back function for GMTransformObject

Returns: Old value of call back function.

Description: Set the update transform call back function to a new function. This call back function is invoked with the original object, the transformed object and the transformation matrix, just before GMTransformObject is returned.
See also: GMTransformObject, GMTransObjSetAnimCrvUpdateFunc,

4.2.310 GMTransObjUpdateAnimCrvs (geomat3d.c:719)

transformations

```
IPObjectStruct *GMTransObjUpdateAnimCrvs(IPObjectStruct *PAnim,
                                         IrtHmgnMatType Mat)
```

PAnim: Animation curves to update following transformation matrix Mat.

Mat: The transformation matrix.

Returns: The updated animation curves' list.

Description: Update the animation curves to work properly after the applied transformation Mat to the parent object whose PAnim are his.
See also: GMTransformObject,

4.2.311 GMTransformObj2UnitSize (geomat3d.c:441)

```
IPObjectStruct *GMTransformObj2UnitSize(const IPObjectStruct *PObj)
```

PObj: Object to map to [0, 1]^n, n = 2,3.

Returns: Input object, mapped to [0, 1]^2, n = 2,3.

Description: Maps the given object to nicely fit the unit square/cube.
See also: GMTransformObject,

4.2.312 GMTransformObject (geomat3d.c:513)

transformations

```
IPObjectStruct *GMTransformObject(const IPObjectStruct *PObj,
                                  IrtHmgnMatType Mat)
```

PObj: Object to be transformed.

Mat: Transformation matrix.

Returns: Transformed object.

Description: Routine to transform an object according to the transformation matrix.
See also: GMTransObjUpdateAnimCrvs, GMTransObjSetUpdateFunc, GMTransformPolyList, GMTransformObjectInPlace,

4.2.313 GMTransformObjectInPlace (geomat3d.c:484)

transformations

```
IPObjectStruct *GMTransformObjectInPlace(IPObjectStruct *PObj,
                                         IrtHmgnMatType Mat)
```

PObj: Object to be transformed.

Mat: Transformation matrix.

Returns: Transformed object.

Description: Routine to transform an object according to the transformation matrix. Input object, PObj, is freed.
See also: GMTransObjUpdateAnimCrvs, GMTransObjSetUpdateFunc, GMTransformPolyList, GMTransformObject,

4.2.314 GMTransformObjectList (geomat3d.c:789)

transformations

```
IPObjectStruct *GMTransformObjectList(const IPObjectStruct *PObj,
                                      IrtHmgnMatType Mat)
```

PObj: Object list to transform.

Mat: Transformation matrix.

Returns: Transformed object list.

Description: Routine to transform an list of objects according to a transformation matrix.

4.2.315 GMTransformPolyList (geomat3d.c:294)

```
IPPolygonStruct *GMTransformPolyList(const IPPolygonStruct *Pls,
                                     IrtHmgnMatType Mat,
                                     int IsPolygon)
```

Pls: List of polygons to transform.

Mat: Transformation matrix.

IsPolygon: TRUE for polygons, for for polylines/points.

Returns: A list of transformed polygons.

Description: Routine to transform a list of polygons according to the prescribed transformation matrix.
See also: GMTransformObject,

4.2.316 GMTrianglePointInclusion (geom_bsc.c:1924)

point inclusion

```
int GMTrianglePointInclusion(const IrtRType *V1,
                             const IrtRType *V2,
                             const IrtRType *V3,
                             const IrtPtType Pt)
```

V1, V2, V3: Triangle to test if Pt is in it.

Pt: Point to test for inclusion in triangle.

Returns: TRUE if Pt inside triangle, FALSE otherwise.

Description: Routine to check if a point is inside a triangle, in the XY plane.
See also: GMPolygonPointInclusion, GMPolygonPlaneInter, GMPolygonXYRayInter,

4.2.317 GMTriangulatePolygon (poly_tri.c:385)

```
IPPolygonStruct *GMTriangulatePolygon(const CagdPolylineStruct *Pl)
```

Pl: A polygon to triangulate.

Returns: List of triangles.

Description: Triangulates a closed polygon. The triangulation based on the paper: Klincsek, G.T. (1980), Minimal triangulations of polygonal domains, Annals of Discrete Mathematics 9, 121-123 The used weight function is:

0.75 * TriangleArea + 0.05 * TrianglePerimeter +

0.10 * EdgesMaxMinRatio + 0.10 * AnglesMaxMinRatio

See also: GMTriangulatePolygon2,

4.2.318 GMTriangulatePolygon2 (poly_tri.c:469)

```
IPPolygonStruct *GMTriangulatePolygon2(const IPPolygonStruct *Pl)
```

Pl: A polygon to triangulate.

Returns: List of triangles.

Description: Triangulates a closed polygon. The triangulation based on minimum weight triangulation(https://en.wikipedia.or
weight_triangulation) The used weight function is:

0.75 * TriangleArea + 0.05 * TrianglePerimeter +

0.10 * EdgesMaxMinRatio + 0.10 * AnglesMaxMinRatio

See also: GMTriangulatePolygon, GMConvertPolysToTriangles,

4.2.319 GMTriangulatePolygonList (poly_tri.c:501)

```
IPPolygonStruct *GMTriangulatePolygonList(const IPPolygonStruct *PlgnList)
```

PlgnList: A polygon list to triangulate.

Returns: List of triangles.

Description: Triangulates a closed polygon. The triangulation based on minimum weight triangulation(https://en.wikipedia.or
weight_triangulation) The used weight function is:

0.75 * TriangleArea + 0.05 * TrianglePerimeter +

0.10 * EdgesMaxMinRatio + 0.10 * AnglesMaxMinRatio

See also: GMConvertPolysToTriangles, GMTriangulatePolygon2,

4.2.320 GMTwoPolySameGeom (poly_cln.c:42)

```
int GMTwoPolySameGeom(const IPPolygonStruct *Pl1,
                      const IPPolygonStruct *Pl2,
                      IrtRType Eps)
```

Pl1, Pl2: Two polygons to compare.

Eps: Tolerance of vertices equality, etc.

Returns: TRUE if two polygons posses same geometry, FALSE otherwise.

Description: Compare two polygons if share the same geometry. Two polygons are considered same if the share the same vertices in order (or in reverse).

See also: GMCleanUpDupPolys,

4.2.321 GMUniteTextures (texture_manage.c:125)

```
IPObjectStruct *GMUniteTextures(const IPObjectStruct *PObj,
                                const char *UnitedTextureFName)
```

PObj: Object to unite all texture images detected into one.

UnitedTextureFName: Name of new united texture image to save to.

Returns: A similar object with merged textures into one image.

Description: Given some input geometry with "ptexture" texture image attributes, unite all textures into one image named UnitedTextureFName, redirect the different "ptexture" attributes to point to it and also place "PTextureZone" attribute next to the "ptexture" to set the current domain within the United image. A single object is assumed to hold no more than one geometry (surface, trimmed surface, etc.)
See also: GMReparamTexture,

4.2.322 GMUpdateVerticesByInterp (intrnrml.c:77)

normals

uv coords

rgb color

```
void GMUpdateVerticesByInterp(IPPolygonStruct *PlList,
                              const IPPolygonStruct *OriginalPl)
```

PlList: List of polygons to update normal for.

OriginalPl: Original polygons PlList was derived from, probably using Boolean operations.

Returns: void

Description: For each polygon in PlList update any vertex with a proper normal, uv uv coord, rgb color, etc. if available in the Original polygon vertex list OriginalPl. All the new vertices are enclosed within the original polygon which must be convex as well.
See also: GMBlendNormalsToVertices, GMInterpVrtxNrmlBetweenTwo, , GMInterpVrtxNrmlBetweenTwo2, GMInterpVrtxNrmlFromPl, , GMInterpVrtxRGBBetweenTwo, GMInterpVrtxRGBFromPl, , GMInterpVrtxUVBetweenTwo, GMInterpVrtxUVFromPl,

4.2.323 GMVecCopy (geom_bsc.c:88)

copy

```
void GMVecCopy(IrtVecType Vdst, const IrtVecType Vsrc)
```

Vdst: Destination vector.

Vsrc: Source vector.

Returns: void

Description: Routine to copy one vector to another:

4.2.324 GMVecCrossProd (geom_bsc.c:207)

cross prod

```
void GMVecCrossProd(IrtVecType Vres,
                    const IrtVecType V1,
                    const IrtVecType V2)
```

Vres: Result of cross product

V1, V2: Two vectors of the cross product.

Returns: void

Description: Routine to compute the cross product of two vectors. Note Vres may be the same as V1 or V2.

4.2.325 GMVecDotProd (geom_bsc.c:515)

GMVecDotProdLen

dot product

```
IrtRType GMVecDotProd(const IrtVecType V1, const IrtVecType V2)
```

V1, V2: Two vector to compute dot product of.

Returns: Resulting dot product.

Description: Routine to compute the dot product of two vectors.

4.2.326 GMVecDotProdLen (geom_bsc.c:534)

GMVecDotProd

dot product

```
IrtRType GMVecDotProdLen(const IrtRType *V1, const IrtRType *V2, int Len)
```

V1, V2: Two vector to compute dot product of.

Len: Length of vectors.

Returns: Resulting dot product.

Description: Routine to compute the dot product of two vectors of length Len.

4.2.327 GMVecLength (geom_bsc.c:131)

magnitude

```
IrtRType GMVecLength(const IrtVecType V)
```

V: To compute its magnitude.

Returns: Magnitude of V.

Description: Routine to compute the magnitude (length) of a given 3D vector:

4.2.328 GMVecMaxAbsValueIndex (geom_bsc.c:179)

magnitude

```
int GMVecMaxAbsValueIndex(const IrtVecType V)
```

V: To compute its maximal axis.

Returns: Computed axis.

Description: Routine to compute the axis in vector with the largest absolute value.
See also: GMVecMinAbsValueIndex,

4.2.329 GMVecMinAbsValueIndex (geom_bsc.c:152)

magnitude

```
int GMVecMinAbsValueIndex(const IrtVecType V)
```

V: To compute its minimal axis.

Returns: Computed axis.

Description: Routine to compute the axis in vector with the smallest absolute value.
See also: GMVecMaxAbsValueIndex,

4.2.330 GMVecNormalize (geom_bsc.c:106)

normalize

```
void GMVecNormalize(IrtVecType V)
```

V: To normalize.

Returns: void

Description: Routine to normalize the vector length to a unit size.

4.2.331 GMVecReflectPlane (geom_bsc.c:561)

```
void GMVecReflectPlane(IrtVecType Dst, IrtVecType Src, IrtVecType PlaneNormal)
```

Dst: Reflected vector.

Src: Input Vector to reflect.

PlaneNormal: Normal of plane (through origin) to reflect Src through.

Returns: void

Description: Reflects a 3D vector through a plane with normal PlaneNormal

4.2.332 GMVecVecAngle (geom_bsc.c:285)

```
IrtRType GMVecVecAngle(const IrtVecType V1, const IrtVecType V2, int Normalize)
```

V1, V2: Vectors to compute their relative angle, in radians.

Normalize: TRUE if vectors need normalization first, FALSE if unit size.

Returns: Angle.

Description: Computes the angle between two space vectors. Angle is returned in radians in the domain of [-Pi, +Pi].

See also: GMAreaSphericalTriangle,

4.2.333 GMVectorFromVectorPlane (geom_bsc.c:861)

```
int GMVectorFromVectorPlane(const IrtVecType Vec,
                            const IrtVecType PlaneN,
                            IrtPtType ProjVec)
```

Vec: Vector for which find projection onto Plane.

PlaneN: On which to project Vec.

ProjVec: The projection of Vec onto Plane, as a normalized vector.

Returns: TRUE, if successful.

Description: Compute the normalized projection of a vector onto a plane. Only the normal direction of the plane id used. This function is invariant to translation of the plane in Euclidean space.

See also: GMPointFromPointPlane,

4.2.334 GMVerifyPolygonsPlanarity (poly_cln.c:423)

```
IPPolygonStruct *GMVerifyPolygonsPlanarity(IPPolygonStruct *Pls, IrtRType Tol)
```

Pls: Polygons to scan and modify, if necessary, in place.

Tol: Of planarity verification.

Returns: Refined list with no coplanar polygons.

Description: Scans all input polygons and split any non coplanar polygons into triangles.

4.2.335 GMVrtxListToCircOrLin (poly_cln.c:474)

circular lists

```
void GMVrtxListToCircOrLin(IPPolygonStruct *Pls, int DoCirc)
```

Pls: List of polys to make sure are circular/linear, in place.

DoCirc: If TRUE, list are made circular. If FALSE, vertices are NULL terminated.

Returns: void

Description: Routine to make sure all polys given are circular/linear. Update in place.

See also: GMCleanUpPolylineList, GMCleanUpPolygonList, GMFilterInteriorVertices, GMVrtxListToCircOrLinDup, IPOpenPolysToClosed, IPClosedPolysToOpen, , IPSetPolyListCirc,

4.2.336 GMVrtxListToCircOrLinDup (poly_cln.c:513)

circular lists

```
void GMVrtxListToCircOrLinDup(IPPolygonStruct *Pls, int DoCirc)
```

Pls: List of polys to make sure are circular/linear, in place.

DoCirc: If TRUE, list are made circular. If FALSE, vertices are NULL terminated.

Returns: void

Description: Routine to make sure all polys given are circular/linear. Update in place. If circular and made linear, first vertex is duplicated as last and same when linear is made circular.
See also: GMCleanUpPolylineList, GMCleanUpPolygonList, GMFilterInteriorVertices, GMVrtxListToCircOrLin, IPOpenPolysToClosed, IPClosedPolysToOpen,

4.2.337 GMZBufferClear (zbuffer.c:129)

```
void GMZBufferClear(VoidPtr ZbufferID)
```

ZbufferID: ID of the zbuffer to use.

Returns: void

Description: Clears the Z buffer to initialization state.
See also: GMZBufferClearSet,

4.2.338 GMZBufferClearSet (zbuffer.c:171)

```
void GMZBufferClearSet(VoidPtr ZbufferID, IrtRType Depth)
```

ZbufferID: ID of the zbuffer to use.

Depth: Initial depth to use.

Returns: void

Description: Clears the Z buffer to initialization state of depth value Depth.
See also: GMZBufferClear,

4.2.339 GMZBufferFree (zbuffer.c:97)

```
void GMZBufferFree(VoidPtr ZbufferID)
```

ZbufferID: ID of the zbuffer to free.

Returns: void

Description: Free the given Zbuffer.

4.2.340 GMZBufferInit (zbuffer.c:60)

```
VoidPtr GMZBufferInit(int Width, int Height)
```

Width, Height: Width and Height of the Z buffer.

Returns: An I.D. of the constructed Z buffer.

Description: Sets up the Zbuffer software implementation.

4.2.341 GMZBufferInvert (zbuffer.c:296)

```
VoidPtr GMZBufferInvert(VoidPtr ZbufferID)
```

ZbufferID: ID of the zbuffer to use.

Returns: An I.D. of the constructed Z buffer.

Description: Invert the depth values. That is z -> -z.

4.2.342 GMZBufferLaplacian (zbuffer.c:365)

```
VoidPtr GMZBufferLaplacian(VoidPtr ZbufferID)
```

ZbufferID: ID of the zbuffer to use.

Returns: An I.D. of the constructed Z buffer.

Description: Apply a Laplacian operator to the Z buffer.

4.2.343 GMZBufferOGLClear (zbuf_ogl.c:805)

```
void GMZBufferOGLClear(void)
```

Returns: void

Description: Clears the Z buffer to initialization state.

4.2.344 GMZBufferOGLFlush (zbuf_ogl.c:972)

```
void GMZBufferOGLFlush(void)
```

Returns: void

Description: Make sure all drawing commands are flushed and we are in sync.

4.2.345 GMZBufferOGLInit (zbuf_ogl.c:603)

```
IritIntPtrSizeType GMZBufferOGLInit(int Width,
                                    int Height,
                                    IrtRType ZMin,
                                    IrtRType ZMax,
                                    int OffScreen)
```

Width, Height: Width and Height of the Z buffer.

ZMin, ZMax: Z domain that the Z buffer will have to support.

OffScreen: Z buffer should be hidden (TRUE) or displayed (FALSE).

Returns: An I.D. of the constructed Z buffer.

Description: Sets up the Zbuffer implementation. This one is employing Open GL.

4.2.346 GMZBufferOGLMakeActive (zbuf_ogl.c:860)

```
void GMZBufferOGLMakeActive(IritIntPtrSizeType Id)
```

Id: I.D. of the Zbuffer to activate.

Returns: void

Description: Make active context. Once this function called drawing commands may be issued. Here, the drawing commands are in Open GL. Necessary only if other Open GL applications are active simultaneously.

4.2.347 GMZBufferOGLQueryColor (zbuf_ogl.c:943)

```
void GMZBufferOGLQueryColor(IrtRType x,
                            IrtRType y,
                            int *Red,
                            int *Green,
                            int *Blue)
```

x, y: The XY coordinates of the point to consider for color.

Red, Green, Blue: The color specifications.

Returns: void

Description: Returns the color at the given location in the Zbuffer.
See also: GMZBufferOGLQueryZ,

4.2.348 GMZBufferOGLQueryZ (zbuf_ogl.c:891)

```
IrtRType GMZBufferOGLQueryZ(IrtRType x, IrtRType y)
```

x, y: The XY coordinates of the point to consider for visibility.

Returns: The depth found at that XY location.

Description: Returns depth at the given location in the Zbuffer.
See also: GMZBufferOGLQueryColor,

4.2.349 GMZBufferOGLSetColor (zbuf_ogl.c:836)

```
void GMZBufferOGLSetColor(int Red, int Green, int Blue)
```

Red, Green, Blue: The color specifications, each between 0 and 255.

Returns: void

Description: Sets the colors of all drawing operations to come.

4.2.350 GMZBufferQueryInfo (zbuffer.c:442)

```
VoidPtr GMZBufferQueryInfo(VoidPtr ZbufferID, int x, int y)
```

ZbufferID: ID of the zbuffer to use.

x, y: The XY coordinates of the point to consider its info.

Returns: The pointer to the user infromation at that XY location.

Description: Returns the user information at the given location in the Zbuffer.

4.2.351 GMZBufferQueryZ (zbuffer.c:417)

```
IrtRType GMZBufferQueryZ(VoidPtr ZbufferID, int x, int y)
```

ZbufferID: ID of the zbuffer to use.

x, y: The XY coordinates of the point to consider its depth.

Returns: The depth found at that XY location.

Description: Returns depth at the given location in the Zbuffer.

4.2.352 GMZBufferRoberts (zbuffer.c:326)

```
VoidPtr GMZBufferRoberts(VoidPtr ZbufferID)
```

ZbufferID: ID of the zbuffer to use.

Returns: An I.D. of the constructed Z buffer.

Description: Apply a Roberts Edge detection operator to the Z buffer.

4.2.353 GMZBufferSetUpdateFunc (zbuffer.c:271)

```
GMZBufferUpdateFuncType GMZBufferSetUpdateFunc(VoidPtr ZbufferID,
                                               GMZBufferUpdateFuncType
                                                                  UpdateFunc)
```

ZbufferID: ID of the zbuffer to use.

UpdateFunc: The call back function to invote for each pixel that is updated in the Z buffer.

Returns: Old call back function

Description: Sets a call back function for each pixel update in the Z buffer.

4.2.354 GMZBufferSetZTest (zbuffer.c:242)

```
GMZTestsType GMZBufferSetZTest(VoidPtr ZbufferID, GMZTestsType ZTest)
```

ZbufferID: ID of the zbuffer to use.

ZTest: The new Z test to consider.

Returns: The old Z test used.

Description: Sets the Z testing option with the assumption that largers Z values mean closers to the viewer.

4.2.355 GMZBufferUpdateHLn (zbuffer.c:548)

```
void GMZBufferUpdateHLn(VoidPtr ZbufferID,
                        int x1,
                        int x2,
                        int y,
                        IrtRType z1,
                        IrtRType z2)
```

ZbufferID: ID of the zbuffer to use.

x1, x2, y: The XY coordinates of the points on the horizontal line.

z1, z2: The new z's to set into the z buffer, if larger (==closer).

Returns: void

Description: Set the depth for all points on a given horizontal line in the Zbuffer.
See also: GMZBufferUpdatePt, GMZBufferUpdateTri, GMZBufferUpdateLine,

4.2.356 GMZBufferUpdateInfo (zbuffer.c:510)

```
VoidPtr GMZBufferUpdateInfo(VoidPtr ZbufferID, int x, int y, VoidPtr Info)
```

ZbufferID: ID of the zbuffer to use.

x, y: The XY coordinates of the point to set its user information.

Info: The new user information to set into the z buffer.

Returns: Old user information.

Description: Set new user information at the given location in the Zbuffer. This update always affects the Z buffer regardless of the depth.
See also: GMZBufferUpdatePt, GMZBufferUpdateLn, GMZBufferUpdateTri,

4.2.357 GMZBufferUpdateLine (zbuffer.c:608)

```
void GMZBufferUpdateLine(VoidPtr ZbufferID,
                         int x1,
                         int y1,
                         int x2,
                         int y2,
                         IrtRType z1,
                         IrtRType z2)
```

ZbufferID: ID of the zbuffer to use.

x1, y1, x2, y2: The XY coordinates of the end points of the line.

z1, z2: The new z's to set into the z buffer, if larger (==closer).

Returns: void

Description: Set the depth for all points on a givenline in the Zbuffer.
See also: GMZBufferUpdatePt, GMZBufferUpdateHLn, GMZBufferUpdateTri,

4.2.358 GMZBufferUpdatePt (zbuffer.c:474)

```
IrtRType GMZBufferUpdatePt(VoidPtr ZbufferID, int x, int y, IrtRType z)
```

ZbufferID: ID of the zbuffer to use.

x, y: The XY coordinates of the point to set its depth. No clipping/validity test is conducted to make sure that the point is inside the Z buffer!

z: The new z to set into the z buffer, if z test succeeds.

Returns: Old Z value.

Description: Set the depth at the given location in the Zbuffer. This update affects the Z buffer only if the z test succeeds.
See also: GMZBufferUpdateInfo, GMZBufferUpdateLn, GMZBufferUpdateTri,

4.2.359 GMZBufferUpdateTri (zbuffer.c:709)

```
void GMZBufferUpdateTri(VoidPtr ZbufferID,
                        int x1,
                        int y1,
                        IrtRType z1,
                        int x2,
                        int y2,
                        IrtRType z2,
                        int x3,
                        int y3,
                        IrtRType z3)
```

ZbufferID: ID of the zbuffer to use.

x1, y1, z1: First point of triangle.

x2, y2, z2: Second point of triangle.

x3, y3, z3: Third point of triangle.

Returns: void

Description: Set the depth for all points in a given triangular line in the Zbuffer.
See also: GMZBufferUpdatePt, GMZBufferUpdateeHLn, GMZBufferUpdateLine,

4.2.360 IGRedrawViewWindow (anim_aux.c:27)

```
void IGRedrawViewWindow(void)
```

Returns: void

Description: Redraws the view window.

4.2.361 IritGeomDescribeError (geom_err.c:82)

error handling

```
const char *IritGeomDescribeError(IritGeomFatalErrorType ErrorNum)
```

ErrorNum: Type of the error that was raised.

Returns: A string describing the error type.

Description: Returns a string describing a the given error. Errors can be raised by any member of this geom library as well as other users. Raised error will cause an invocation of IritGeomFatalError function which decides how to handle this error. IritGeomFatalError can for example, invoke this routine with the error type, print the appropriate message and quit the program.

4.2.362 IritGeomFatalError (geom_ftl.c:57)

error handling

```
void IritGeomFatalError(IritGeomFatalErrorType ErrID)
```

ErrID: Error type that was raised.

Returns: void

Description: Trap Geom_lib errors right here. Provides a default error handler for the geom library. Gets an error description using IritGeomDescribeError, prints it and exit the program using exit.

4.2.363 IritGeomSetFatalErrorFunc (geom_ftl.c:29)

error handling

```
IritGeomSetErrorFuncType IritGeomSetFatalErrorFunc(IritGeomSetErrorFuncType
                                                                    ErrorFunc)
```

ErrorFunc: New error function to use.

Returns: Old error function reference.

Description: Sets the error function to be used by Geom_lib.

4.2.364 PrimGenBOXObject (primitv1.c:149)

box

primitives

```
IPObjectStruct *PrimGenBOXObject(const IrtVecType Pt,
                                 IrtRType WidthX,
                                 IrtRType WidthY,
                                 IrtRType WidthZ)
```

Pt: Low end corner of BOX.

WidthX: Width of BOX (X axis).

WidthY: Depth of BOX(Y axis).

WidthZ: Height of BOX(Z axis).

Returns: A BOX primitive

Description:

```
  Routine to create a BOX geometric object defined by Pt - the minimum
3d point, and Width - Dx Dy & Dz vector.              4
  Order of vertices is as                        5       7
follows in the picture:                          |   6   |
                                                 |   |   |
(Note vertex 0 is hidden behind edge 2-6)        |   |   |
                                                 1   |   3
                                                     2
```

All dimensions can be negative, denoting the reversed direction.

See also: PrimSetGeneratePrimType, PrimGenGBOXObject, PrimGenCONEObject, , PrimGenCONE2Object, PrimGenCYLINObject, PrimGenSPHEREObject, , PrimGenTORUSObject, PrimGenBOXWIREObject,

4.2.365 PrimGenBOXWIREObject (primitv1.c:240)

box

primitives

```
IPObjectStruct *PrimGenBOXWIREObject(const IrtVecType Pt,
                                     IrtRType WidthX,
                                     IrtRType WidthY,
                                     IrtRType WidthZ)
```

Pt: Low end corner of BOX.

WidthX: Width of BOX (X axis).

WidthY: Depth of BOX(Y axis).

WidthZ: Height of BOX(Z axis).

Returns: A BOX primitive

Description:

```
  Routine to create a BOX Wireframe geometric object defined by Pt - the
minimum 3d point, and Width - Dx Dy & Dz vector.      4
  Order of vertices is as                          5       7
follows in the picture:                            |   6   |
                                                   |   |   |
(Note vertex 0 is hidden behind edge 2-6)          |   |   |
                                                   1   |   3
                                                       2
```

All dimensions can be negative, denoting the reversed direction.

See also: PrimSetGeneratePrimType, PrimGenGBOXObject, PrimGenCONEObject, , PrimGenCONE2Object, PrimGenCYLINObject, PrimGenSPHEREObject, , PrimGenTORUSObject, PrimGenBOXObject,

4.2.366 PrimGenCONE2Object (primitv1.c:582)

cone

primitives

```
IPObjectStruct *PrimGenCONE2Object(const IrtVecType Pt,
                                   const IrtVecType Dir,
                                   IrtRType R1,
                                   IrtRType R2,
                                   int Bases)
```

Pt: Center location of Base of CON2.

Dir: Direction and distance from Pt to center of other base of CON2.

R1, R2: Two base radii of the truncated CON2

Bases: 0 for none, 1 for bottom, 2 for top, 3 for both.

Returns: A CON2 Primitive.

Description: Routine to create a truncated CONE, CON2, geometric object defined by Pt - the base 3d center point, Dir - the cone direction and height, and two base radii R1 and R2. See also PrimSetResolution on fineness control of approximation of the primitive using flat faces.

See also: PrimSetGeneratePrimType, PrimGenBOXObject, PrimGenGBOXObject, , PrimGenCONEObject, PrimGenCYLINObject, PrimGenSPHEREObject, , PrimGenTORUSObject,

4.2.367 PrimGenCONEObject (primitv1.c:392)

cone

primitives

```
IPObjectStruct *PrimGenCONEObject(const IrtVecType Pt,
                                  const IrtVecType Dir,
                                  IrtRType R,
                                  int Bases)
```

Pt: Center location of Base of CONE.

Dir: Direction and distance from Pt to apex of CONE.

R: Radius of Base of the cone.

Bases: 0 for none, 1 for bottom, 2 for top, 3 for both.

Returns: A CONE Primitive.

Description: Routine to create a CONE geometric object defined by Pt - the base 3d center point, Dir - the cone direction and height, and base radius R. See also PrimSetResolution on fineness control of approximation of the primitive using flat faces.

See also: PrimSetGeneratePrimType, PrimGenBOXObject, PrimGenGBOXObject, , PrimGenCONE2Object, PrimGenCYLINObject, PrimGenSPHEREObject, , PrimGenTORUSObject,

4.2.368 PrimGenCYLINObject (primitv1.c:797)

cylinder

primitives

```
IPObjectStruct *PrimGenCYLINObject(const IrtVecType Pt,
                                   const IrtVecType Dir,
                                   IrtRType R,
                                   int Bases)
```

Pt: Center location of Base of CYLINder.

Dir: Direction and distance from Pt to other base of cylinder.

R: Radius of Base of the cylinder.

Bases: 0 for none, 1 for bottom, 2 for top, 3 for both.

Returns: A CYLINDER Primitive.

Description: Routine to create a CYLINder geometric object defined by Pt - the base 3d center point, Dir - the cylinder direction and height, and radius R. See also PrimSetResolution on fineness control of approximation of the primitive using flat faces.
See also: PrimSetGeneratePrimType, PrimGenBOXObject, PrimGenGBOXObject, , PrimGenCONEObject, PrimGenCONE2Object, PrimGenSPHEREObject, , PrimGenTORUSObject,

4.2.369 PrimGenEXTRUDEObject (primitv2.c:441)

extrusion surface

primitives

```
IPObjectStruct *PrimGenEXTRUDEObject(const IPObjectStruct *Cross,
                                     const IrtVecType Dir,
                                     int Bases)
```

Cross: To extrude in direction Dir.

Dir: Direction and magnitude of extrusion.

Bases: 0 for none, 1 for bottom, 2 for top, 3 for both.

Returns: An extrusion surface.

Description: Routine to a create an extrusion surface out of the given cross section and the given direction. Input can either be a polygon/line or a freefrom curve object. If input is a polyline/gon, it must never be coplanar with Dir. See also PrimSetResolution on fineness control of approximation of the primitive using flat faces.
See also: PrimSetResolution, PrimGenSURFREVObject, PrimGenSURFREVAxisObject, , PrimGenSURFREV2Object, PrimGenSURFREV2AxisObject, PrimGenRULEDObject,

4.2.370 PrimGenFrameController (primitv3.c:919)

```
IPObjectStruct *PrimGenFrameController(IrtRType BBoxLen,
                                       IrtRType NLeverLen,
                                       IrtRType TLeverLen,
                                       const char *HandleName)
```

BBoxLen: The length along the X axis of the deformed object.

NLeverLen: The length of the normal Levers.

TLeverLen: The length of the tangent Lever.

HandleName: The name of the handle, required for naming sub-objects.

Returns: A list object with the proper sub-objects named as <HandleName>O for the sphere Object in the origin, <HandleName><T/N/B><H/C> for objects of T/N/B levers.

Description: Constructs a frame handle with axes parallel to XYZ, at the origin.
See also: PrimGenTransformController, PrimGenTransformController2DCrvs,

4.2.371 PrimGenGBOXObject (primitv1.c:302)

general box
box
primitives

```
IPObjectStruct *PrimGenGBOXObject(const IrtVecType Pt,
                                  const IrtVecType Dir1,
                                  const IrtVecType Dir2,
                                  const IrtVecType Dir3)
```

Pt: Low end corner of GBOX.

Dir1, Dir2, Dir3: Three independent directional vectors to define GBOX.

Returns: A GBOX primitive.

Description: Routine to create a GBOX geometric object defined by Pt - the minimum 3d point, and 3 direction Vectors Dir1, Dir2, Dir3. If two of the direction vectors are parallel the GBOX degenerates to a zero volume object. A NULL pointer is returned in that case.

```
                                          4
Order of vertices is as                 5       7
follows in the picture:                 |   6   |
                                        |   |   |
(Note vertex 0 is hidden behind edge 2-6)|   |   |
                                        1   |   3
                                            2
```

See also: PrimSetGeneratePrimType, PrimGenBOXObject, PrimGenCONEObject, , PrimGenCONE2Object, PrimGenCYLINObject, PrimGenSPHEREObject, , PrimGenTORUSObject,

4.2.372 PrimGenObjectFromPolyList (primitv1.c:1612)

primitives

```
IPObjectStruct *PrimGenObjectFromPolyList(IPObjectStruct *PObjList)
```

PObjList: List of polygonal objects.

Returns: A single object containing all polygons in all provided objects, by a simple union.

Description: Routine to create an OBJECT directly from set of specified polys. No test is made for the validity of the model in any sense.

4.2.373 PrimGenPOLYDISKObject (primitv1.c:1405)

disk
primitives

```
IPObjectStruct *PrimGenPOLYDISKObject(const IrtVecType Nrml,
                                      const IrtVecType Trns,
                                      IrtRType R)
```

Nrml: Normal to the plane this disk included in.

Trns: A translation factor of the center of the disk.

R: Radius of the disk.

Returns: A single polygon object - a disk.

Description: Routine to create a POLYDISK geometric object defined by the normal N and the translation vector T. The object is a planar disk (a circle of _PrimGlblResolution points in it...) and its radius is equal to R. The normal direction is assumed to point to the inside of the object. See also PrimSetResolution on fineness control of approximation of the primitive using flat faces.

4.2.374 PrimGenPOLYGONObject (primitv1.c:1480)

polygon
polyline
primitives

```
IPObjectStruct *PrimGenPOLYGONObject(IPObjectStruct *PObjList, int IsPolyline)
```

PObjList: List of vertices/points to construct as a polygon/line.

IsPolyline: If TRUE, make a polyline, otherwise a polygon.

Returns: A polygon/line constructed from PObjList.

Description: Routine to create a POLYGON/LINE directly from its specified vertices. The validity of the elements in the provided list is tested to make sure they are vectors or points. No test is made to make sure all vertices are on one plane, and that no two vertices are similar.

4.2.375 PrimGenPolygon3Vrtx (primitv1.c:1850)

```
IPPolygonStruct *PrimGenPolygon3Vrtx(const IrtVecType V1,
                                     const IrtVecType V2,
                                     const IrtVecType V3,
                                     const IrtVecType Vin,
                                     int *VrtcsRvrsd,
                                     IPPolygonStruct *Pnext)
```

V1, V2, V3: Three vertices of the constructed polygon.

Vin: A vertex that can be assumed to be inside the object for normal evaluation of the plane of polygon. Can be NULL in which case vrtcs order is kept.

VrtcsRvrsd: set to TRUE if has Vin and order of vertices is oriented in reverse. Can be NULL to be ignored. M

Pnext: Next is chain of polygons, in linked list.

Returns: The constructed polygon.

Description: Routine to create a polygon out of a list of 3 vertices V1/2/3. The fourth vertex is inside (actually, this is not true, as this point will be in the positive part of the plane, which only locally in the object...) the object, so the polygon's normal direction can be evaluated uniquely. No test is made to make sure the 3 points are not co-linear... The points are placed in order.

See also: PrimGenPolygon4Vrtx, GMGenPolyline2Vrtx,

4.2.376 PrimGenPolygon4Vrtx (primitv1.c:1715)

```
IPPolygonStruct *PrimGenPolygon4Vrtx(const IrtVecType V1,
                                     const IrtVecType V2,
                                     const IrtVecType V3,
                                     const IrtVecType V4,
                                     const IrtVecType Vin,
                                     int *VrtcsRvrsd,
                                     IPPolygonStruct *Pnext)
```

V1, V2, V3, V4: Four vertices of the constructed polygon.

Vin: A vertex that can be assumed to be inside the object for normal evaluation of the plane of polygon. Can be NULL in which case vrtcs order is kept.

VrtcsRvrsd: Set to TRUE if has Vin and order of vertices is oriented in reverse. Can be NULL to be ignored.

Pnext: Next is chain of polygons, in linked list.

Returns: The constructed polygon.

Description: Routine to create a polygon out of a list of 4 vertices V1/2/3/4. The fifth vertex is inside (actually, this is not true, as this point will be in the positive part of the plane, which only locally in the object...) the object, so the polygon's normal direction can be evaluated uniquely. No test is made to make sure the 4 points are co-planar... The points are placed in order.

See also: PrimGenPolygon3Vrtx, PrimGenPolygon4Vrtx2, GMGenPolyline2Vrtx,

4.2.377 PrimGenPolygon4Vrtx2 (primitv1.c:1760)

```
IPPolygonStruct *PrimGenPolygon4Vrtx2(const IrtVecType V1,
                                      const IrtVecType V2,
                                      const IrtVecType V3,
                                      const IrtVecType V4,
                                      const IrtVecType Vin,
                                      int *VrtcsRvrsd,
                                      int *Singular,
                                      IPPolygonStruct *Pnext)
```

V1, V2, V3, V4: Four vertices of the constructed polygon.

Vin: A vertex that can be assumed to be inside the object for normal evaluation of the plane of polygon. Can be NULL in which case vrtcs order is kept.

VrtcsRvrsd: Set to TRUE if has Vin and order of vertices is oriented in reverse. Can be NULL to be ignored.

Singular: -1 if regular, index of singular vertex if this rectangle is singular and a triangle is returned.

Pnext: Next is chain of polygons, in linked list.

Returns: The constructed polygon.

Description: Routine to create a polygon out of a list of 4 vertices V1/2/3/4. The fifth vertex is inside (actually, this is not true, as this point will be in the positive part of the plane, which only locally in the object...) the object, so the polygon's normal direction can be evaluated uniquely. No test is made to make sure the 4 points are co-planar... The points are placed in order.
See also: PrimGenPolygon3Vrtx, PrimGenPolygon4Vrtx, GMGenPolyline2Vrtx,

4.2.378 PrimGenPolyline4Vrtx (primitv3.c:997)

```
IPPolygonStruct *PrimGenPolyline4Vrtx(const IrtVecType V1,
                                      const IrtVecType V2,
                                      const IrtVecType V3,
                                      const IrtVecType V4,
                                      IPPolygonStruct *Pnext)
```

V1, V2, V3, V4: Four vertices of the constructed polyline. V1 is duplicated as fifth last point as well.

Pnext: Next is chain of polyline, in linked list.

Returns: The constructed polygon.

Description: Routine to create a polyline out of a list of 4 vertices V1/2/3/4. No test is made to make sure the 4 points are co-planar... The points are placed in order.
See also: PrimGenPolyline4Vrtx, PrimGenPolygon3Vrtx, GMGenPolyline2Vrtx,

4.2.379 PrimGenRULEDObject (primitv2.c:694)

ruled surface

primitives

```
IPObjectStruct *PrimGenRULEDObject(const IPObjectStruct *Cross1,
                                   const IPObjectStruct *Cross2)
```

Cross1, Cross2: Polylines to rule a surface between. If both cross sections are in the XY plane, a single planar polygon is constructed. Otherwise, the number of vertices in Cross1 and Cross2 must be equal and a rectangular polygon is constructed for each edge.

Returns: A single polygon representing the ruled surface.

Description: Routine to a create a ruled surface out of the given two cross sections.
See also: PrimSetResolution, PrimGenSURFREVObject, PrimGenSURFREVAxisObject, , PrimGenSURFREV2Object, PrimGenSURFREV2AxisObject, PrimGenEXTRUDEObject,

4.2.380 PrimGenSPHEREObject (primitv1.c:1014)

sphere

primitives

```
IPObjectStruct *PrimGenSPHEREObject(const IrtVecType Center, IrtRType R)
```

Center: Center location of SPHERE.

R: Radius of sphere.

Returns: A SPHERE Primitive.

Description: Routine to create a SPHERE geometric object defined by Center, the center of the sphere and R, its radius. See also PrimSetResolution on fineness control of approximation of the primitive using flat faces.
See also: PrimSetGeneratePrimType, PrimGenBOXObject, PrimGenGBOXObject, , PrimGenCONEObject, PrimGenCONE2Object, PrimGenCYLINObject, , PrimGenTORUSObject,

4.2.381 PrimGenSURFREV2AxisObject (primitv2.c:391)

surface of revolution

primitives

```
IPObjectStruct *PrimGenSURFREV2AxisObject(IPObjectStruct *Cross,
                                          IrtRType StartAngle,
                                          IrtRType EndAngle,
                                          const IrtVecType Axis)
```

Cross: To rotate around Axis axis forming a surface of revolution.

StartAngle, EndAngle: angles of portion of surface of revolution, in degrees, between 0 and 360.

Axis: Axis of rotation.

Returns: A (portion of) a surface of revolution.

Description: Routine to a create surface of revolution by rotating the given cross section along the Axis axis. Input can either be a polygon/line or a freefrom curve object.
See also: PrimSetResolution, PrimGenSURFREVObject, PrimGenSURFREVAxisObject, , PrimGenSURFREV2Object, PrimGenEXTRUDEObject, PrimGenRULEDObject,

4.2.382 PrimGenSURFREV2Object (primitv2.c:234)

surface of revolution

primitives

```
IPObjectStruct *PrimGenSURFREV2Object(const IPObjectStruct *Cross,
                                      IrtRType StartAngle,
                                      IrtRType EndAngle)
```

Cross: To rotate around the Z axis forming a surface of revolution.

StartAngle, EndAngle: angles of portion of surface of revolution, in degrees, between 0 and 360.

Returns: A (portion of) a surface of revolution.

Description: Routine to a create surface of revolution by rotating the given cross section along the Z axis, from StartAngle to EndAngle. Input can either be a polygon/line or a freefrom curve object. If input is a polyline/gon, it must never be coplanar with the Z axis.
See also: PrimSetResolution, PrimGenSURFREVObject, PrimGenSURFREVAxisObject, , PrimGenSURFREV2AxisObject, PrimGenEXTRUDEObject, PrimGenRULEDObject,

4.2.383 PrimGenSURFREVAxisObject (primitv2.c:186)

surface of revolution

primitives

```
IPObjectStruct *PrimGenSURFREVAxisObject(IPObjectStruct *Cross,
                                         const IrtVecType Axis)
```

Cross: To rotate around Axis axis forming a surface of revolution.

Axis: Axis of rotation.

Returns: A surface of revolution.

Description: Routine to a create surface of revolution by rotating the given cross section along the Axis axis. Input can either be a polygon/line or a freefrom curve object.
See also: PrimSetResolution, PrimGenSURFREVObject, PrimGenSURFREV2Object, , PrimGenSURFREV2AxisObject, PrimGenEXTRUDEObject, PrimGenRULEDObject,

4.2.384 PrimGenSURFREVObject (primitv2.c:44)

surface of revolution

primitives

```
IPObjectStruct *PrimGenSURFREVObject(const IPObjectStruct *Cross)
```

Cross: To rotate around the Z axis forming a surface of revolution.

Returns: A surface of revolution.

Description: Routine to a create surface of revolution by rotating the given cross section along the Z axis. Input can either be a polygon/line or a freeform curve object. If input is a polyline/gon, it must never be coplanar with the Z axis.
See also: PrimSetResolution, PrimGenSURFREVAxisObject, PrimGenSURFREV2Object, , PrimGenSURFREV2AxisObject PrimGenEXTRUDEObject, PrimGenRULEDObject,

4.2.385 PrimGenTORUSObject (primitv1.c:1233)

torus

primitives

```
IPObjectStruct *PrimGenTORUSObject(const IrtVecType Center,
                                   const IrtVecType Normal,
                                   IrtRType Rmajor,
                                   IrtRType Rminor)
```

Center: Center location of the TORUS primitive.

Normal: Normal to the major plane of the torus.

Rmajor: Major radius of torus.

Rminor: Minor radius of torus.

Returns: A TOURS Primitive.

Description: Routine to create a TORUS geometric object defined by Center - torus 3d center point, the main torus plane normal Normal, major radius Rmajor and minor radius Rminor (Tube radius). Teta runs on the major circle, Fee on the minor one. Then

```
X = (Rmajor + Rminor * cos(Fee)) * cos(Teta)
Y = (Rmajor + Rminor * cos(Fee)) * sin(Teta)
Z = Rminor * sin(Fee)
```

See also PrimSetResolution on fineness control of approximation of the primitive using flat faces.

See also: PrimSetGeneratePrimType, PrimGenBOXObject, PrimGenGBOXObject, , PrimGenCONEObject, PrimGenCONE2Object, PrimGenCYLINObject, , PrimGenSPHEREObject,

4.2.386 PrimGenTransformController2D (primitv3.c:120)

```
IPObjectStruct *PrimGenTransformController2D(const GMBBBboxStruct *BBox,
                                             int HasRotation,
                                             int HasTranslation,
                                             int HasScale)
```

BBox: Dimensions of constructed box.

HasRotation: Should we add rotational handles?

HasTranslation: Should we add translation handles?

HasScale: Should we add scale handles?

Returns: A list object with the proper sub-objects named as R for (Z) rotations, {X,Y} for translations.

Description: Constructs a 2D bounding box with transformation handles to control the way an object is transformed in the plane.

See also: PrimGenTransformController, PrimGenTransformController2DCrvs,

4.2.387 PrimGenTransformController2DCrvs (primitv3.c:260)

```
IPObjectStruct *PrimGenTransformController2DCrvs(const GMBBBboxStruct *BBox)
```

BBox: Dimensions of constructed box (only XY).

Returns: A list object with the proper sub-objects named as R for (Z) rotations, {X,Y} for translations

Description: Constructs a 2D bounding box with transformation handles to control the way an object is transformed in the plane. This transform is formed out of curves only.

See also: PrimGenTransformController, PrimGenTransformController2D,

4.2.388 PrimGenTransformControllerBox (primitv3.c:617)

```
IPObjectStruct *PrimGenTransformControllerBox(const GMBBBboxStruct *BBox,
                                              int HasRotation,
                                              int HasTranslation,
                                              int HasUniformScale,
                                              IrtRType BoxOpacity,
                                              IrtRType RelTesalate)
```

BBox: Dimensions of constructed box.

HasRotation: Should we add rotational handles?

HasTranslation: Should we add translation handles?

HasUniformScale: Should we add uniform scale handles?

BoxOpacity: Opacity of the bbox itself, between zero and one. If one, no bbox geometry is created.

RelTesalate: Relative tessallations of elements of the handles. 1.0 for default values.

Returns: A list object with the proper sub-objects named as R{X,Y,Z} for rotations, T{X,Y,Z} for translations, and SXYZ for scaling.

Description: Constructs a bounding box with transformation handles to control the way an object is transformed.
See also: PrimGenTransformController2D, PrimGenTransformController2DCrvs, PrimGenTransformControllerSphere,

4.2.389 PrimGenTransformControllerSphere (primitv3.c:435)

```
IPObjectStruct *PrimGenTransformControllerSphere(const GMBBBboxStruct *BBox,
                                                 int HasRotation,
                                                 int HasTranslation,
                                                 int HasUniformScale,
                                                 IrtRType BoxOpacity,
                                                 IrtRType RelTesalate)
```

BBox: Dimensions of constructed box.

HasRotation: Should we add rotational handles?

HasTranslation: Should we add translation handles?

HasUniformScale: Should we add uniform scale handles?

BoxOpacity: Opacity of the bbox itself, between zero and one. If one, no bbox geometry is created.

RelTesalate: Relative tessallations of elements of the handles. 1.0 for default values.

Returns: A list object with the proper sub-objects named as R{X,Y,Z} for rotations, T{X,Y,Z} for translations, and SXYZ for scaling.

Description: Constructs a bounding box with transformation handles to control the way an object is transformed.
See also: PrimGenTransformController2D, PrimGenTransformController2DCrvs, PrimGenTransformControllerBox,

4.2.390 PrimSetGeneratePrimType (primitv1.c:73)

```
int PrimSetGeneratePrimType(int SetGeneratePrimitive)
```

SetGeneratePrimitive: 0 - polygonal primitive. 1 - surface primitive. 2 - model primitive. 3 - trivariate volumetric primitive. 4 - VModel primitive. 5 - Singular (having locations with vanishing Jacobian) primitive.

Returns: Old value of PolygonalPrimitive flag.

Description: Sets the way primitives are constructed - as polygons, as a freeform surface, or as a model surface.
See also: PrimSetSurfacePrimitiveRational, PrimGenBOXObject, PrimGenGBOXObject, , PrimGenCONEObject, PrimGenCONE2Object, PrimGenCYLINObject, , PrimGenSPHEREObject, PrimGenTORUSObject,

4.2.391 PrimSetResolution (primitv1.c:1946)

primitives

```
int PrimSetResolution(int Resolution)
```

resolution

Resolution: To set as new resolution for all primitve constructors.

Returns: Old resolution value.

Description: Routine to set the polygonal resolution (fineness). Resolution sroutghly the number of edges a circular primitive will have along the entire circle.

4.2.392 PrimSetSurfacePrimitiveRational (primitv1.c:105)

```
int PrimSetSurfacePrimitiveRational(int SurfaceRational)
```

SurfaceRational: TRUE for rational, FALSE for integral form.

Returns: Old value of PolygonalPrimitive flag.

Description: Sets the way surface primitives are constructed - as exact rational form or approximated polynomial (integral) form.
See also: PrimSetGeneratePrimType, PrimGenBOXObject, PrimGenGBOXObject, , PrimGenCONEObject, PrimGenCONE2Object, PrimGenCYLINObject, , PrimGenSPHEREObject, PrimGenTORUSObject,

4.2.393 TriangulatePolygonIndices (poly_tri.c:272)

```
static int *TriangulatePolygonIndices(const CagdPolylineStruct *P,
                                      const IPPolygonStruct *IrtP,
                                      CagdBType EnsureNoLeakingTrs,
                                      int *NumTriangles)
```

P, IrtP: A polygon to triangulate. (in two representations - for efficiency)

EnsureNoLeakingTrs: If TRUE, the weight function gives a weight of infinity to triangles that leak of the polygon.

NumTriangles: An output parameter that stores the number of generated Triangles.

Returns: List of triangle indices orginazied in tuples of three. Each index refers to a vertex in the input polygon. Each Three consequence indices forms a triangle.

Description: Triangulates a closed polygon. The triangulation based on the paper: Klincsek, G.T. (1980), Minimal triangulations of polygonal domains, Annals of Discrete Mathematics 9, 121-123 The used weight function is:

0.75 * TriangleArea + 0.05 * TrianglePerimeter +

0.10 * EdgesMaxMinRatio + 0.10 * AnglesMaxMinRatio

See also: TriangulatePolygon,

4.2.394 _GMFitGetFittingModel (fit2pts.c:265)

```
const GMFitFittingShapeStruct *_GMFitGetFittingModel(GMFittingModelType
                                                     FittingModel)
```

FittingModel: The enum of the needed fitting model.

Returns: A pointer to a staticly allocated fitting struct.

Description: This function returns the matching GMFitFittingShapeStruct to the given enum, or NULL if none exist. THIS POINTER MUST NOT BE FREED!!!
See also: FitData, FitDataWithOutliers,

4.2.395 main (geom_bsc.c:4017)

```
void main(void)
```

Returns: void

Description: Test routines for the polynomial equation solvers.

Chapter 5

Graphics Library, grap_lib

5.1 General Information

This library handles general drawing and display algorithms, including tesselation of all geometric objects, such as curves and surfaces, into displayable primitives, i.e. polygons and polylines.

5.2 Library Functions

5.2.1 IGActiveFreeAttrByID (grap_gen.c:379)

```
void IGActiveFreeAttrByID(IPObjectStruct *PObj, IPAttrIDType AttrID)
```

PObj: Objects to remove Named attribute from.
AttrID: ID of attribute to remove.
Returns: void
Description: Free attribute ID AttrID from PObj in all object's hierarchy.

5.2.2 IGActiveFreePolyIsoAttribute (grap_gen.c:321)

```
void IGActiveFreePolyIsoAttribute(IPObjectStruct *PObj,
                                  int FreePolygons,
                                  int FreeIsolines,
                                  int FreeSketches,
                                  int FreeCtlMesh)
```

PObj: Object to remove Named attribute from.
FreePolygons: If TRUE free all polygonal approximations of freeforms.
FreeIsolines: If TRUE free all isocurve approximations of freeforms.
FreeSketches: If TRUE free all sketching strokes of freeforms.
FreeCtlMesh: If TRUE free all control meshes of freeforms.
Returns: void
Description: Free attribute named Name from all objects in PObj object's hierarchy.

5.2.3 IGActiveListFreeAttrByID (grap_gen.c:290)

```
void IGActiveListFreeAttrByID(IPObjectStruct *PObjs, IPAttrIDType AttrID)
```

PObjs: Objects to remove Named attribute from.
AttrID: ID of attribute to remove.
Returns: void
Description: Free attribute named Name from all objects in PObjs object's hierarchy.

5.2.4 IGActiveListFreePolyIsoAttribute (grap_gen.c:265)

```
void IGActiveListFreePolyIsoAttribute(IPObjectStruct *PObjs,
                                      int FreePolygons,
                                      int FreeIsolines,
                                      int FreeSketches,
                                      int FreeCtlMesh)
```

PObjs: Objects to remove Named attribute from.

FreePolygons: If TRUE free all polygonal approximations of freeforms.

FreeIsolines: If TRUE free all isocurve approximations of freeforms.

FreeSketches: If TRUE free all sketching strokes of freeforms.

FreeCtlMesh: If TRUE free all control meshes of freeforms.

Returns: void

Description: Free attribute named Name from all objects in PObjs object's hierarchy and linked list.

5.2.5 IGClearObjTextureMovieAttr (grap_gen.c:1604)

```
void IGClearObjTextureMovieAttr(const IPObjectStruct *PObj)
```

PObj: Object to free all texture/movie attributes.

Returns: void

Description: Clears all texture/movie attributes, if any from PObj.
See also: IGInitSrfTexture, IGInitSrfMovie,

5.2.6 IGCnvrtChar2Raster (draw_str.c:271)

```
unsigned char *IGCnvrtChar2Raster(const char c)
```

c: Character to convert.

Returns: The bit map rasterized

Description: Converts a given text, using a fixed raster font, to a form suitable for drawing in a raster display. In Open GL this can amount to:

```
glColor3d(r, g, b);
glRasterPos2d(x, y);
glBitmap(8, 8, 0, 0, 10, 0, IGCnvrtChar2Raster('x'));
```

See also: IGDrawString,

5.2.7 IGConfirmConvexPolys (grap_gen.c:91)

```
void IGConfirmConvexPolys(IPObjectStruct *PObj, int Depth)
```

PObj: Objects to make sure all its polygons are convex.

Depth: Of object hierarchy.

Returns: void

Description: Make sure all polygons are convex.

5.2.8 IGDSetDrawPolyFunc (drawpoly.c:33)

```
IGDrawObjectFuncType IGDSetDrawPolyFunc(IGDrawObjectFuncType DrawPolyFunc)
```

DrawPolyFunc: New poly drawing routine or NULL to disable.

Returns: Old poly drawing routine.

Description: Sets the polygon drawing call back routine.

5.2.9 IGDecomposeMdlDue2Graphics (draw_mdl.c:740)

```
IPObjectStruct *IGDecomposeMdlDue2Graphics(const IPObjectStruct *PMdls)
```

PMdls: To check, and possibly decompose.

Returns: NULL, if can be rendered as a monolithic object, or a list of trimmed surface object if required splitting.

Description: Checks if the Model can be rendered as a monolithic objects or needs to split into trimmed surfaces (as each trimmed surface has different graphics attributes).
See also: IGDecomposeVMdlDue2Graphics,

5.2.10 IGDecomposeVMdlDue2Graphics (drawvmdl.c:733)

```
IPObjectStruct *IGDecomposeVMdlDue2Graphics(const IPObjectStruct *PVMdls)
```

PVMdls: To check, and possibly decompose.

Returns: NULL, if can be rendered as a monolithic object, or a list of trimmed surface object if required splitting.

Description: Checks if the VModel can be rendered as a monolithic objects or needs to split into trimmed surfaces (as each trimmed surface has different graphics attributes).
See also: IGDecomposeMdlDue2Graphics,

5.2.11 IGDefaultProcessEvent (grap_gen.c:1646)

```
int IGDefaultProcessEvent(IGGraphicEventType Event, IrtRType *ChangeFactor)
```

Event: Event to process.

ChangeFactor: A continuous scale between -1 and 1 to quantify the change to apply according to the event type. For composed operation contains both X and Y information.

Returns: TRUE if refresh is needed.

Description: Processes the given event. Returns TRUE if redraw of view window is needed.
See also: IGHandleContinuousMotion,

5.2.12 IGDefaultStateHandler (grap_gen.c:1773)

```
int IGDefaultStateHandler(int State, int StateStatus, int Refresh)
```

State: State event type to handle.

StateStatus: IG_STATE_OFF, IG_STATE_ON, IG_STATE_TGL for turning off, on or toggling current value. IG_STATE_DEC and IG_STATE_INC serves as dec./inc. factors.

Refresh: Not used.

Returns: TRUE if needs to refresh.

Description: Handle the event of a pop up window. This is the default handler which can be invoked by other specific handlers for event they do not care about.

5.2.13 IGDeleteTexture (grap_gen.c:1442)

```
int IGDeleteTexture(const IPObjectStruct *PObj)
```

PObj: Object to delete texture from. Attributes might be affected.

Returns: TRUE if was texture, FALSE otherwise.

Description: Deletes the texture mapping function of an object.

5.2.14 IGDrawCurve (draw_crv.c:36)

```
void IGDrawCurve(IPObjectStruct *PObj)
```

PObj: A curve object to draw.

Returns: void

Description: Draw a single Curve object using current modes and transformations. Curve must be with either E3 or P3 point type and must be a NURB curve. Piecewise linear approximation is cashed under "_isoline" and "_ctlpoly" attributes of PObj.

5.2.15 IGDrawCurveGenPolylines (draw_crv.c:101)

```
void IGDrawCurveGenPolylines(const IPObjectStruct *PObj)
```

PObj: A curve(s) object. Its attributes are likely to be affected.

Returns: void

Description: Generates the polyline approximation of the curve on the fly if needed and display it.

5.2.16 IGDrawModel (draw_mdl.c:52)

```
void IGDrawModel(IPObjectStruct *PObj, IGDrawObjectFuncType DrawTSrfsFuncPtr)
```

PObj: A model object to draw.

DrawTSrfsFuncPtr: Optional pointer to a function that will draw the trimmed surfaces, if model is decomposed into trimmed surfaces.

Returns: void

Description: Draw a single model object using current modes and transformations. Piecewise linear approximation is cashed under "_isoline" and "_ctlmesh" attributes of PObj. Polygonal approximation is saved under "_polygons".

5.2.17 IGDrawModelGenPolygons (draw_mdl.c:168)

```
void IGDrawModelGenPolygons(const IPObjectStruct *PObj)
```

PObj: A model(s) object.

Returns: void

Description: Generates the polygonal approximation of the model on the fly if needed and display it.

5.2.18 IGDrawPolyBoundary (grap_gen.c:797)

```
void IGDrawPolyBoundary(IPObjectStruct *PObj)
```

PObj: A polygonal object.

Returns: void

Description: Draw the boundary edges of a polygonal object.

5.2.19 IGDrawPolyContours (grap_gen.c:862)

```
void IGDrawPolyContours(IPObjectStruct *PObj)
```

PObj: A polygonal object.

Returns: void

Description: Draw XY parallel contours to the given model.

5.2.20 IGDrawPolyContoursSetup (grap_gen.c:835)

```
int IGDrawPolyContoursSetup(IrtRType x, IrtRType y, IrtRType z, int n)
```

x, y, z: Direction of contouring.

n: Number of desired contours.

Returns: Old number of contours.

Description: Setup information for contouring.

5.2.21 IGDrawPolyDiscontinuities (grap_gen.c:1078)

```
void IGDrawPolyDiscontinuities(IPObjectStruct *PObj)
```

PObj: A polyline object.

Returns: void

Description: Draw the discont edges of an object, if any. PObj can have a "DiscontAngle" attribute with the dihedral angle to consider discontinuous (in degrees).

5.2.22 IGDrawPolyIsophotes (grap_gen.c:999)

```
void IGDrawPolyIsophotes(IPObjectStruct *PObj)
```

PObj: A polygonal object.

Returns: void

Description: Draw Isophotes from the Z direction to the given model.

5.2.23 IGDrawPolyIsophotesSetup (grap_gen.c:972)

```
int IGDrawPolyIsophotesSetup(IrtRType x, IrtRType y, IrtRType z, int n)
```

x, y, z: Direction of isophotes' view.

n: Number of desired isophotes.

Returns: Old number of isophotes.

Description: Setup information for isophotes' drawings.

5.2.24 IGDrawPolyKnotLines (grap_gen.c:1120)

```
void IGDrawPolyKnotLines(IPObjectStruct *PObj)
```

PObj: A polyline object.

Returns: void

Description: Draw the knot lines of an object, if any.

5.2.25 IGDrawPolySilhBndry (grap_gen.c:736)

```
void IGDrawPolySilhBndry(IPObjectStruct *PObj)
```

PObj: A polygonal object.

Returns: void

Description: Draw the boundary and silhouette edges of a polygonal object.

5.2.26 IGDrawPolySilhouette (grap_gen.c:755)

```
void IGDrawPolySilhouette(IPObjectStruct *PObj)
```

PObj: A polygonal object.

Returns: void

Description: Draw the silhouette edges of a polygonal object.

5.2.27 IGDrawPolygonSketches (grap_gen.c:660)

```
void IGDrawPolygonSketches(IPObjectStruct *PObj)
```

PObj: A surface(s) object.

Returns: void

Description: Generates a sketch like drawing of the surface on the fly if needed and display it.

5.2.28 IGDrawString (draw_str.c:165)

```
void IGDrawString(IPObjectStruct *PObj)
```

PObj: A string object to draw.

Returns: void

Description: Draw a single string object using current modes and transformations.
See also: IGCnvrtChar2Raster,

5.2.29 IGDrawSurface (draw_srf.c:56)

```
void IGDrawSurface(IPObjectStruct *PObj)
```

PObj: A surface object to draw.

Returns: void

Description: Draw a single surface object using current modes and transformations. Piecewise linear approximation is cashed under "_Isoline??Res" and "_ctlmesh" attributes of PObj. Polygonal approximation is saved under "_Polygons??Res".

5.2.30 IGDrawSurfaceGenPolygons (draw_srf.c:186)

```
void IGDrawSurfaceGenPolygons(const IPObjectStruct *PObj)
```

PObj: A surface(s) object. Note its attributes might be affected.

Returns: void

Description: Generates the polygonal approximation of the surface on the fly if needed and display it.

5.2.31 IGDrawTriangGenSrfPolygons (drawtris.c:126)

```
void IGDrawTriangGenSrfPolygons(const IPObjectStruct *PObj)
```

PObj: A triangular surface(s) object.

Returns: void

Description: Generates the polygonal approximation of the triangular surface on the fly if needed and display it.

5.2.32 IGDrawTriangSrf (drawtris.c:46)

```
void IGDrawTriangSrf(IPObjectStruct *PObj)
```

PObj: A triangular surface object to draw.

Returns: void

Description: Draw a single triangular surface object using current modes and transformations. Piecewise linear approximation is cashed under "_isoline" and "_ctlmesh" attributes of PObj. Polygonal approximation is saved under "_polygons".

5.2.33 IGDrawTrimSrf (drawtsrf.c:51)

```
void IGDrawTrimSrf(IPObjectStruct *PObj)
```

PObj: A trimmed surface object to draw.

Returns: void

Description: Draw a single trimmed surface object using current modes and transformations. Piecewise linear approximation is cashed under "_isoline" and "_ctlmesh" attributes of PObj. Polygonal approximation is saved under "_polygons".

5.2.34 IGDrawTrimSrfGenPolygons (drawtsrf.c:150)

```
void IGDrawTrimSrfGenPolygons(const IPObjectStruct *PObj)
```

PObj: A trimmed surface(s) object.

Returns: void

Description: Generates the polygonal approximation of the trimmed surface on the fly if needed and display it.

5.2.35 IGDrawTrivar (drawtriv.c:47)

```
void IGDrawTrivar(IPObjectStruct *PObj)
```

PObj: A trivariate function object to draw.

Returns: void

Description: Draw a single trivariate function object using current modes and transformations. Piecewise linear approximation is cashed under "_isoline" and "_ctlmesh" attributes of PObj. Polygonal approximation is saved under "_polygons".

5.2.36 IGDrawTrivarGenSrfPolygons (drawtriv.c:147)

```
void IGDrawTrivarGenSrfPolygons(const IPObjectStruct *PObj)
```

PObj: A trivariate(s) object.

Returns: void

Description: Generates the polygonal approximation of the trivariate on the fly if needed and display it.

5.2.37 IGDrawVModel (drawvmdl.c:53)

```
void IGDrawVModel(IPObjectStruct *PObj, IGDrawObjectFuncType DrawTSrfsFuncPtr)
```

PObj: A model object to draw.

DrawTSrfsFuncPtr: Optional pointer to a function that will draw the trimmed surfaces, if model is decomposed into trimmed surfaces.

Returns: void

Description: Draw a single model object using current modes and transformations. Piecewise linear approximation is cashed under "_isoline" and "_ctlmesh" attributes of PObj. Polygonal approximation is saved under "_polygons".

5.2.38 IGDrawVModelGenPolygons (drawvmdl.c:168)

```
void IGDrawVModelGenPolygons(const IPObjectStruct *PObj)
```

PObj: A model(s) object.

Returns: void

Description: Generates the polygonal approximation of the model on the fly if needed and display it.

5.2.39 IGFindMinimalDist (grap_gen.c:500)

```
IrtRType IGFindMinimalDist(IPObjectStruct *PObj,
                           IPPolygonStruct **MinPl,
                           IrtPtType MinPt,
                           int *MinPlIsPolyline,
                           const IrtPtType LinePos,
                           const IrtVecType LineDir,
                           IrtRType *HitDepth)
```

PObj: List of objects to search for a minimal distance.

MinPl: Poly with the minimal distance in PObj.

MinPt: Closest point on the picked object.

MinPlIsPolyline: TRUE if MinPl is a polyline, FALSE if a polygon.

LinePos: A point on the line to test against.

LineDir: The direction of the line.

HitDepth: In case of zero distance (the ray hits a polygon, update the closest depth.

Returns: The minimal distance found to the line.

Description: Finds the minimal distance in PObj to the line defined by LinePos and LineDir.

5.2.40 IGGenPolygonSketches (grap_gen.c:694)

```
IPObjectStruct *IGGenPolygonSketches(IPObjectStruct *PObj, IrtRType FineNess)
```

PObj: A surface(s) object.

FineNess: Relative fineness to approximate PObj with.

Returns: The sketch data.

Description: Generates a sketch like drawing for the given object with the prescribed fineness.

5.2.41 IGGetObjIsoLines (grap_gen.c:1150)

```
IPObjectStruct *IGGetObjIsoLines(const IPObjectStruct *PObj)
```

PObj: To get the iso curves' approximation

Returns: The iso curve's approximation.

Description: Get the proper isoparametric curve approximation of the object.

5.2.42 IGGetObjPolygons (grap_gen.c:1181)

```
IPObjectStruct *IGGetObjPolygons(const IPObjectStruct *PObj)
```

PObj: To get the polygonal approximation

Returns: The polygonal approximation.

Description: Get the proper polygonal approximation of the object.

5.2.43 IGGraphicsStateGet (grap_gen.c:2430)

```
void IGGraphicsStateGet(IGGraphicStateStruct *IGS)
```

IGS: Graphics state to fetch.

Returns: void

Description: Get the Irit graphics state.
See also: IritGraphStateStruct, IGGraphicsStateReset, IGGraphicsStateGet,

5.2.44 IGGraphicsStateReset (grap_gen.c:2472)

```
void IGGraphicsStateReset(IGGraphicStateStruct *IGS)
```

IGS: Graphics state to initialize.

Returns: void

Description: Initializes the Irit graphics state
See also: IritGraphStateStruct, IGGraphicsStateGet, IGGraphicsStateSet,

5.2.45 IGGraphicsStateResetDefault (grap_gen.c:2451)

```
void IGGraphicsStateResetDefault(void)
```

Returns: void

Description: Reste the default global graphics state.
See also:

5.2.46 IGGraphicsStateSet (grap_gen.c:2409)

```
void IGGraphicsStateSet(const IGGraphicStateStruct *IGS)
```

IGS: Graphics state to set.

Returns: void

Description: Set an Irit graphics state.
See also: IritGraphStateStruct, IGGraphicsStateReset, IGGraphicsStateGet,

5.2.47 IGInitSrfMovie (grap_gen.c:1481)

```
int IGInitSrfMovie(const IPObjectStruct *PObj)
```

PObj: Object to extract its texture mapping movie if has one.

Returns: TRUE if object has texture mapping movie, FALSE otherwise.

Description: Reads in a texture mapping movie if object has "pmovie" attribute and set up the texture movie for further processing.
See also: IGInitSrfTexture, IGClearObjTextureMovieAttr,

5.2.48 IGInitSrfTexture (grap_gen.c:1217)

```
int IGInitSrfTexture(const IPObjectStruct *PObj)
```

PObj: Object to extract its texture mapping function if has one. Note its attributes might be affected.

Returns: TRUE if object has texture mapping, FALSE otherwise.

Description: Reads in a texture mapping image if object has "ptexture" attribute and set up the texture for further processing.
See also: IGInitSrfMovie, IGClearObjTextureMovieAttr, IGInitSrfTextureFromImage,

5.2.49 IGInitSrfTextureFromImage (grap_gen.c:1371)

```
int IGInitSrfTextureFromImage(const IPObjectStruct *PObj,
                              int SrfIdx,
                              void *Image,
                              int ImageWidth,
                              int ImageHeight,
                              int ImageAlpha)
```

PObj: Object to apply texture mapping image to. Note its attributes might be affected though.

SrfIdx: If PObj is an object with multiple surfaces (e.g. a MODEL), will apply the texture to that surface (First is 1). If -1 will apply to all surfaces in PObj.

Image: Image to apply as texture, either IrtImgRGBAPxlStruct or IrtImgPixelStruct, depending on value of ImageAlpha.

ImageWidth: Width of image to apply.

ImageHeight: Height of image to apply.

ImageAlpha: TRUE if image to apply has Alpha, FALSE otherwise.

Returns: TRUE if successful, FALSE otherwise.

Description: Apply a texture mapping image to object PObj and set up the texture for further processing.
See also: IGInitSrfTexture,

5.2.50 IGLoadGeometry (grap_gen.c:143)

```
IPObjectStruct *IGLoadGeometry(const char **FileNames, int NumFiles)
```

FileNames: Names of files from which to read the geometry.

NumFiles: Number of files in array of names FileNames.

Returns: Read geometry, NULL if error or no geometry.

Description: Get data. This function also updates the following global variables: + IPViewMat, IPPrspMat are update with the respective matrices, if found. + IPWasViewMat, IPWasPrspMat are set to TRUE if above matrices were found. + IGState.LastProcessMat is updated with the most current matrix value. + IGState.ViewMode is updated with view mode - perspective or orthographic.
See also: IPGetDataFiles,

5.2.51 IGSaveCurrentMat (grap_gen.c:194)

```
void IGSaveCurrentMat(int ViewMode, char *Name)
```

ViewMode: Either perspective or orthographic.

Name: File name to save current view at.

Returns: void

Description: Saves the current viewing matrices.
See also: IGSubmitCurrentMat,

5.2.52 IGSetCtlMeshPostFunc (grap_gen.c:2383)

```
IGDrawUpdateFuncType IGSetCtlMeshPostFunc(IGDrawUpdateFuncType Func)
```

Func: New callback function to use, or NULL to disable.

Returns: Old callback function used.

Description: Sets a call back function to handle setting after control mesh draw.

5.2.53 IGSetCtlMeshPreFunc (grap_gen.c:2360)

```
IGDrawUpdateFuncType IGSetCtlMeshPreFunc(IGDrawUpdateFuncType Func)
```

Func: New callback function to use, or NULL to disable.

Returns: Old callback function used.

Description: Sets a call back function to handle setting before control mesh draw.

5.2.54 IGSetSketchPostFunc (grap_gen.c:2337)

```
IGDrawUpdateFuncType IGSetSketchPostFunc(IGDrawUpdateFuncType Func)
```

Func: New callback function to use, or NULL to disable.

Returns: Old callback function used.

Description: Sets a call back function to handle setting after sketch style draw.

5.2.55 IGSetSketchPreFunc (grap_gen.c:2314)

```
IGDrawUpdateFuncType IGSetSketchPreFunc(IGDrawUpdateFuncType Func)
```

Func: New callback function to use, or NULL to disable.

Returns: Old callback function used.

Description: Sets a call back function to handle setting before sketch style draw.

5.2.56 IGSetSrfPolysPostFunc (grap_gen.c:2245)

```
IGDrawUpdateFuncType IGSetSrfPolysPostFunc(IGDrawUpdateFuncType Func)
```

Func: New callback function to use, or NULL to disable.

Returns: Old callback function used.

Description: Sets a call back function to handle setting after polygons are drawn.

5.2.57 IGSetSrfPolysPreFunc (grap_gen.c:2221)

```
IGDrawUpdateFuncType IGSetSrfPolysPreFunc(IGDrawUpdateFuncType Func)
```

Func: New callback function to use, or NULL to disable.

Returns: Old callback function used.

Description: Sets a call back function to handle setting before polygons are drawn.

5.2.58 IGSetSrfWirePostFunc (grap_gen.c:2291)

```
IGDrawUpdateFuncType IGSetSrfWirePostFunc(IGDrawUpdateFuncType Func)
```

Func: New callback function to use, or NULL to disable.

Returns: Old callback function used.

Description: Sets a call back function to handle setting after wireframe draw.

5.2.59 IGSetSrfWirePreFunc (grap_gen.c:2268)

```
IGDrawUpdateFuncType IGSetSrfWirePreFunc(IGDrawUpdateFuncType Func)
```

Func: New callback function to use, or NULL to disable.

Returns: Old callback function used.

Description: Sets a call back function to handle setting before wireframe draw.

5.2.60 IGSketchDrawPolygons (sketches.c:791)

```
void IGSketchDrawPolygons(IPObjectStruct *PObjSketches)
```

PObjSketches: A sketches object.

Returns: void

Description: Generates a sketch like drawing of the polygonal object, on the fly if needed, and display it.

5.2.61 IGSketchDrawSurface (sketches.c:168)

```
void IGSketchDrawSurface(const IPObjectStruct *PObjSketches)
```

PObjSketches: A sketches object.

Returns: void

Description: Generates a sketch like drawing of a surface.

5.2.62 IGSketchGenPolyImportanceSketches (sketches.c:1146)

```
IPObjectStruct *IGSketchGenPolyImportanceSketches(IPObjectStruct *PObj,
                                                  IGSketchParamStruct *SketchParams,
                                                  IrtRType FineNess)
```

PObj: Polygonal model to process for importance.

SketchParams: NPR sketch parameters' info structure.

FineNess: Of marching steps on the surface.

Returns: Traced strokes as piecewise linear approximations.

Description: Precompute the strokes that emphasize important features in the geometry using an importance measure that looks at the diherdal angle of adjacent polygons.
See also: IGSketchGenPolySketches,

5.2.63 IGSketchGenPolySketches (sketches.c:821)

```
IPObjectStruct *IGSketchGenPolySketches(IPObjectStruct *PlObj,
                                        IrtRType FineNess,
                                        int Importance)
```

PlObj: The polygonal object to process.

FineNess: Relative fineness to approximate the sketchs of PlObj with.

Importance: If TRUE, we should also compute importance for each point.

Returns: The sketch data.

Description: Generates a sketch like drawing for the given polygonal object.
See also: IGSketchGenPolyImportanceSketches,

5.2.64 IGSketchGenSrfSketches (sketches.c:415)

```
IPObjectStruct *IGSketchGenSrfSketches(const CagdSrfStruct *InSrf,
                                       IrtRType FineNess,
                                       const IPObjectStruct *PObj,
                                       int Importance)
```

InSrf: Input surface to gene sketch geometry for.

FineNess: Relative fineness to approximate the sketches with.

PObj: Object Srf originated from.

Importance: If TRUE, we should also compute importance for each point.

Returns: The sketch data.

Description: Generates a sketch like drawing for the given surface with the fineness prescribed.

5.2.65 IGUpdateFPS (grap_gen.c:2185)

```
void IGUpdateFPS(int Start)
```

Start: TRUE to start timing, FALSE to end it.

Returns: void

Description: Update the current frame per second.

5.2.66 IGUpdateObjectBBox (grap_gen.c:420)

```
void IGUpdateObjectBBox(IPObjectStruct *PObj)
```

PObj: Objects to update its BBOX.

Returns: void

Description: Update the BBox of the given object if has none.

5.2.67 IGUpdateViewConsideringScale (grap_gen.c:457)

```
void IGUpdateViewConsideringScale(IrtHmgnMatType Mat)
```

Mat: N.S.F.I.

Returns: void

Description: Updates the global view matrix IPViewMat with given matrix Mat while preserving the scaling factor in the original global view.

Chapter 6

Model Library, mdl_lib

6.1 General Information

This library provides the necessary tools to represent and process models. Models are sets of trimmed surfaces forming a closed 2-manifold shell. Models are typically the result of Boolean operations over freeform (trimmed) NURBs geometry but can also be constructed directly or via other schemes.

6.2 Library Functions

6.2.1 IritMdlDescribeError (mdl_err.c:50)

error handling

```
const char *IritMdlDescribeError(IritMdlFatalErrorType ErrorNum)
```

ErrorNum: Type of the error that was raised.

Returns: A string describing the error type.

Description: Returns a string describing a the given error. Errors can be raised by any member of this mdl library as well as other users. Raised error will cause an invocation of IritMdlFatalError function which decides how to handle this error. IritMdlFatalError can for example, invoke this routine with the error type, print the appropriate message and quit the program.

6.2.2 IritMdlFatalError (mdl_ftl.c:56)

error handling

```
void IritMdlFatalError(IritMdlFatalErrorType ErrID)
```

ErrID: Error type that was raised.

Returns: void

Description: Trap Mdl_lib errors right here. Provides a default error handler for the mdl library. Gets an error description using IritMdlDescribeError, prints it and exit the program using exit.

6.2.3 IritMdlSetFatalErrorFunc (mdl_ftl.c:28)

error handling

```
MdlSetErrorFuncType IritMdlSetFatalErrorFunc(MdlSetErrorFuncType ErrorFunc)
```

ErrorFunc: New error function to use.

Returns: Old error function reference.

Description: Sets the error function to be used by Mdl_lib.

6.2.4 MdlAddSrf2Mdl (mdlcnvrt.c:795)

```
MdlModelStruct *MdlAddSrf2Mdl(const MdlModelStruct *Mdl,
                              const CagdSrfStruct *Srf,
                              int StitchModel)
```

Mdl: Model to add Srf to.

Srf: A surface to add to a model. The four boundary curves of Srf are created and added as trimming curves.

StitchModel: TRUE to also stitch model (detect and fuse common edges).

Returns: New model with Srf merged into Mdl. NULL if error.

Description: Adds the given surface into a model.
See also: MdlCnvrtMdl2TrimmedSrfs, MdlCnvrtTrimmedSrf2Mdl, MdlCnvrtSrf2Mdl,

6.2.5 MdlAddTrimmedSrf2Mdl (mdlcnvrt.c:832)

```
MdlModelStruct *MdlAddTrimmedSrf2Mdl(const MdlModelStruct *Mdl,
                                     const TrimSrfStruct *TSrf,
                                     int MergeTCrvsIntoLoops,
                                     int StitchModel)
```

Mdl: Model to add Srf to.

TSrf: A trimmed surface to add to a model.

MergeTCrvsIntoLoops: TRUE to also merge trimming curves in loops into one large loop curve.

StitchModel: TRUE to also stitch model (detect and fuse common edges).

Returns: New model with TSrf merged into Mdl. NULL if error.

Description: Adds the given trimmed surface into a model.
See also: MdlCnvrtMdl2TrimmedSrfs, MdlCnvrtSrf2Mdl, MdlCnvrtTrimmedSrf2Mdl,

6.2.6 MdlBoolClassifyNonInterIslands (mdl2bool.c:1312)

```
int MdlBoolClassifyNonInterIslands(MdlModelStruct *Model,
                                   MdlBooleanType BType)
```

Model: Model in which to classify the unclassified trimming loop.

BType: The Boolean operation applied, resulting in model.

Returns: TRUE if some trimming loops was updated, FALSE otherwise.

Description: Classify loops that are isolated from the intersection and hence the inside-propagation of MdlBoolClassifyNonInterTrimSrfs cannot reach them. For each trimming surface do:

1. Go over all unclassified loops and check if inside or outside the classified loops of the model.
2. If inside - include in the model. if outside - purge the loop.

6.2.7 MdlBoolClassifyNonInterTrimSrfs (mdl2bool.c:1045)

```
int MdlBoolClassifyNonInterTrimSrfs(MdlModelStruct *Model)
```

Model: Model in which to classify the non-intersecting trimming curves.

Returns: TRUE if some trimming curve segment was updated, FALSE otherwise.

Description: Classify references of trimming curves in trimmed surfaces that are not interested in this Boolean operation.

1. Go over all refs and search for a reference of an old intersection that is classified on one srf side only and propagate to the second srf side.
2. Go over all trimming references in the second srf side and propagated the information.
3. While there are unclassified references go to 1. Note this function will not propagate to (valid) trimming curves that intersection curves are not connected to (island loops inside valid trimming loops - See MdlBoolClassifyNonInterIslands for that).

6.2.8 MdlBoolClassifyTrimSrfLoops (mdl2bool.c:957)

```
int MdlBoolClassifyTrimSrfLoops(MdlTrimSrfStruct *TSrf,
                                CagdRType Tol,
                                CagdBType InsideOtherModel)
```

TSrf: A trimmed surface we need to update its trimming loops.

Tol: Tolerance to match end points of trimming curves.

InsideOtherModel: TRUE if we seek the inside of the other model (i.e. intersection operations), FALSE for outside (i.e. subtraction).

Returns: TRUE if successful, FALSE otherwise.

Description: Based on the new trimming curves, recreate the trimming loops of this trimmed surface. At this stage we assume the new intersecting curves only exists in the interior domain of trimmed surface (outside intersections were purged away). Further, the old existing trimming curves were split at the locations they intersect with the new intersection curves. The returned TSrf will have the trimming curves organized in loops.

6.2.9 MdlBoolCleanInterCrvs (mdl3bool.c:44)

```
void MdlBoolCleanInterCrvs(CagdCrvStruct *Crv, const CagdSrfStruct *Srf)
```

Crv: Intersection curve to clean up.

Srf: where Crv is an intersection (trimming) curve in its domain.

Returns: void

Description: Make sure the intersection curve is fully consistent with the surface. Either precisely closed or precisely on the boundary of the surface.

See also:

6.2.10 MdlBoolCleanRefsToTSrf (mdl3bool.c:390)

```
int MdlBoolCleanRefsToTSrf(MdlModelStruct *Model, MdlTrimSrfStruct *TSrf)
```

Model: To clean references to TSrf in, in place.

TSrf: Reference to clean, in Model.

Returns: TRUE if found and cleaned, FALSE otherwise.

Description: Cleans references to TSrf in Model. TSrf, if in Model, is NOT affected. Note associated trimming curves of TSrf are removed anf freed.

6.2.11 MdlBoolCleanUnusedTrimCrvRefs (mdl3bool.c:434)

```
int MdlBoolCleanUnusedTrimCrvRefs(MdlModelStruct *Model,
                                  CagdBType OutsideTCrvRefs,
                                  CagdBType UnclassifiedTCrvRefs)
```

Model: To process and clean unused trimming crvs in, in place.

OutsideTCrvRefs: Trimming curve that are tagged as !Inside are purged.

UnclassifiedTCrvRefs: Trimming curve that are unclassified are purged.

Returns: Number of trimming curve references that were purged.

Description: Clean unused trimming curves in the model - trimming curves that are used by no trimming surface, etc.

6.2.12 MdlBoolCleanUnusedTrimCrvsSrfs (mdl3bool.c:522)

```
int MdlBoolCleanUnusedTrimCrvsSrfs(MdlModelStruct *Model)
```

Model: To process and clean unused trimming crvs/srfs in, in place.

Returns: Number of trimming curves that were purged.

Description: Clean unused trimming curves/surfaces in the model - trimming curves that are used by no trimming surface.

6.2.13 MdlBoolCleanUnusedTrimCrvsSrfs2 (mdl3bool.c:575)

```
int MdlBoolCleanUnusedTrimCrvsSrfs2(MdlModelStruct *Model)
```

Model: To process and clean unused trimming crvs in, in place.

Returns: Number of trimming curves that were purged.

Description: Clean unused trimming curves in the model - trimming curves that are used by no trimming surface.

6.2.14 MdlBoolClipTSrfs2TrimDomain (mdl3bool.c:678)

```
void MdlBoolClipTSrfs2TrimDomain(MdlModelStruct *Model,
                                 CagdBType ExtendSrfDomain)
```

Model: To clip its surfaces to their trimmed domain, in place.

ExtendSrfDomain: Whether to extend the models's surfaces in the UV domain.

Returns: void

Description: Clip the surfaces in the given model to the domain of the trimming curves (+ some epsilon).

6.2.15 MdlBoolConcatLoops2OneSegRefsList (mdl2bool.c:793)

```
MdlTrimSegRefStruct *MdlBoolConcatLoops2OneSegRefsList(MdlLoopStruct *Loops)
```

Loops: To concatenate into one long list of trimming curve references. Concatenation is done in place, destroying Loops.

Returns: The new trimming curve references ;ist.

Description: Concatenate all trimming curve references in given loops into one linear list of trimming curve references.

6.2.16 MdlBoolGenTrimSegs (mdl3bool.c:333)

```
MdlTrimSegStruct *MdlBoolGenTrimSegs(const MvarPolylineStruct *Inters12,
                                     MdlTrimSrfStruct *Srf1,
                                     MdlTrimSrfStruct *Srf2)
```

Inters12: The intersection result as polylines in R^4 (the space of both surfaces' domains).

Srf1, Srf2: The two surfaces participated in this intersection computation.

Returns: List of model trimming segments.

Description: Converts the result of two surfaces intersection into MdlTrimSegStructs.
See also:

6.2.17 MdlBoolImprovedPointsAddPoints (mdl3bool.c:870)

```
void MdlBoolImprovedPointsAddPoints(MdlImprovedPointsCacheStruct **IPCache,
                                    int n,
                                    MdlImprovingPointStruct IPts[4],
                                    MdlImprovingPointStruct OrigIPts[4],
                                    CagdRType ImprovedErrorSqr)
```

IPCache: The improved points cache to update.

n: Number of UV locations to consider as candidate into the Cache. n is at most 4.

IPts: The n improved locations.

OrigIPts: The n original locations.

ImprovedErrorSqr: Error square after numeric improvement was aimed.

Returns: void

Description: Update the IPC cache with new (improved) locations, so these n TSrfs are evaluated to the same Euclidean location at these UV locations.

6.2.18 MdlBoolImprovedPointsCacheFree (mdl3bool.c:1062)

```
void MdlBoolImprovedPointsCacheFree(MdlImprovedPointsCacheStruct **IPCache)
```

IPCache: The improved points cache to free.

Returns: void

Description: Free the improved points cache.

6.2.19 MdlBoolImprovedPointsCacheInit (mdl3bool.c:842)

```
MdlImprovedPointsCacheStruct *MdlBoolImprovedPointsCacheInit()
```

Returns: New cache.

Description: init an improved points cache.

6.2.20 MdlBoolImprovedPointsUpdateOldPointsInMdl (mdl3bool.c:1018)

```
int MdlBoolImprovedPointsUpdateOldPointsInMdl(
                                    const MdlImprovedPointsCacheStruct *IPCache,
                                    MdlModelStruct *Mdl)
```

IPCache: The improved points cache.

Mdl: The model to update.

Returns: TRUE if successful, FALSE otherwise.

Description: Based on the global list of improved points, go over the entire model and update it with all the improved UV locations.

6.2.21 MdlBoolImprovedPointsUpdateOldPointsInSeg (mdl3bool.c:959)

```
int MdlBoolImprovedPointsUpdateOldPointsInSeg(
                                    const MdlImprovedPointsCacheStruct *IPCache,
                                    MdlTrimSegStruct *Seg)
```

IPCache: The improved points cache.

Seg: The trimming segment to update.

Returns: TRUE if successful, FALSE otherwise.

Description: Based on the global list of improved points, go over the entire model and update it with all the improved UV locations.

6.2.22 MdlBoolIsMdlContainsModel (mdl3bool.c:764)

```
CagdBType MdlBoolIsMdlContainsModel(const MdlModelStruct *Model1,
                                    const MdlModelStruct *Model2)
```

Model1, Model2: The first and second models.

Returns: True iff Model1 is contained inside Model2.

Description: Returns if first model is contained inside a second model. The function assumes that the two models do not intersect.

6.2.23 MdlBoolMergeSegRefsIntoLoops (mdl2bool.c:828)

```
MdlLoopStruct *MdlBoolMergeSegRefsIntoLoops(MdlTrimSrfStruct *TSrf,
                                            MdlBooleanType OrigBType,
                                            CagdRType Tol)
```

TSrf: Trimmed surface we now update its trimming loops.

OrigBType: Original type of Boolean operation involved.

Tol: Tolerance to match end points of trimming curves.

Returns: The newly formed loops of the given surface.

Description: Merges the given list of trimming segments back into loops, in place.

6.2.24 MdlBoolNumerImproveEndPt (mdl3bool.c:106)

```
int MdlBoolNumerImproveEndPt(MdlImprovedPointsCacheStruct **IPCache,
                             MdlTrimSegStruct *Seg1,
                             MdlTrimSegStruct *Seg2,
                             CagdBType Seg1Last,
                             CagdBType Seg2Last)
```

IPCache: The improved points cache to update.

Seg1, Seg2: The two intersecting segments that start/end at the same Euclidean location. Seg2 can be NULL.

Seg1Last, Seg2Last: TRUE for last point in Segment, FALSE for first.

Returns: TRUE if improved, FALSE otherwise.

Description: Improves numerically a given intersection start/end point in Seg1 and Seg2 that points at the same Euclidean location.

See also:

6.2.25 MdlBoolOpParamsAlloc (mdl_gen.c:1575)

```
MdlBopsParamsStruct *MdlBoolOpParamsAlloc(
                                    CagdBType PertubrateSecondModel,
                                    CagdBType ExtendUVSrfResult,
                                    IPAttrIDType IntersectedSurfacesAttrID,
                                    MdlIntersectionCBFunc InterCBFunc,
                                    MdlPreIntersectionCBFunc PreInterCBFunc,
                                    MdlPostIntersectionCBFunc PostInterCBFunc)
```

PertubrateSecondModel: Perturb second Model or not.

ExtendUVSrfResult: Extend the result model surfaces in the UV space.

IntersectedSurfacesAttrID: ID of integer attribute that will indicate in each surface if it is trimmed as result of the Boolean operation.

InterCBFunc: Call back function during the SSI.

PreInterCBFunc: Callback function before the SSI.

PostInterCBFunc: Callback function after the SSI.

Returns: New initialized Boolean operations struct.

Description: Allocates and initializes Model Boolean operations parameters struct. Memory management of User data in the sub-struct SSICBData -> InterCBData is the user's responsibility.

6.2.26 MdlBoolOpParamsFree (mdl_gen.c:1618)

```
void MdlBoolOpParamsFree(struct MdlBopsParamsStruct *BopsParams)
```

BopsParams: Params struct to free.

Returns: void

Description: Frees memory of Model Boolean operations parameters struct. Memory management of User data in the sub-struct SSICBData -> InterCBData is the user's responsibility.

6.2.27 MdlBoolResetAllTags (mdl3bool.c:482)

```
void MdlBoolResetAllTags(MdlModelStruct *Model)
```

Model: To process and clean, in place.

Returns: void

Description: Clean all tags from the model.

6.2.28 MdlBoolSetHandleInterDiscont (mdl_bool.c:2658)

Booleans

```
int MdlBoolSetHandleInterDiscont(int HandleInterDiscont)
```

HandleInterDiscont: TRUE to aim at handling intersections on boundaries and C^1 discontinuous edges.

Returns: Old OutputInterCurveType value.

Description: Controls aim for C^1 discontinuous edges/boundary edges intersections.
See also: MdlBooleanInterCrv, MdlBoolSetOutputInterCrv,

6.2.29 MdlBoolSetOutputInterCrv (mdl_bool.c:2599)

Booleans

```
int MdlBoolSetOutputInterCrv(int OutputInterCurve)
```

OutputInterCurve: If TRUE only intersection curves are computed. If FALSE, full blown Boolean is applied.

Returns: Old OutputInterCurve value.

Description: Controls if intersection curves or full Boolean operation is to be performed on input models.
See also: MdlBooleanInterCrv, MdlBoolSetOutputInterCrvType,

6.2.30 MdlBoolSetOutputInterCrvType (mdl_bool.c:2628)

Booleans

```
int MdlBoolSetOutputInterCrvType(int OutputInterCurveType)
```

OutputInterCurveType: 0 for Euclidean space, 1/2 for inter curves in Model1/2.

Returns: Old OutputInterCurveType value.

Description: Controls the type of intersection curves to return.
See also: MdlBooleanInterCrv, MdlBoolSetOutputInterCrv,

6.2.31 MdlBoolTrimSrfIntersects (mdl3bool.c:809)

```
int MdlBoolTrimSrfIntersects(const MdlTrimSrfStruct *TSrf)
```

TSrf: Trimmed surface to examine if it intersects the other model.

Returns: TRUE if intersects, FALSE otherwise

Description: Returns TRUE if this trimmed surface intersects the other model in this Boolean operation.

6.2.32 MdlBooleanCut (mdl_bool.c:2270)

```
IPObjectStruct *MdlBooleanCut(const MdlModelStruct *Model1,
                              const MdlModelStruct *Model2,
                              struct MvarSrfSrfInterCacheStruct *SSICache,
                              MdlBopsParamsStruct *BopsParams)
```

Model1, Model2: The two models to consider.

SSICache: SSI cache to be used and updated, if provided (if not NULL - optional).

BopsParams: Additional parameters and callback functions used in the Boolean operation process.

Returns: Resulting boolean.

Description: Computes the Boolean subtraction of two models.
See also: MdlBooleanSetTolerances, MdlBooleanUnion, MdlBooleanIntersection, MdlBooleanSubtractionm MdlBooleanDtctModelSrfInter, , MdlBooleanDtctModelSrfInter,

6.2.33 MdlBooleanDetectInter (mdl_bool.c:2123)

```
CagdBType MdlBooleanDetectInter(const MdlModelStruct *Model1,
                                const MdlModelStruct *Model2,
                                struct MvarSrfSrfInterCacheStruct *SSICache,
                                MdlBopsParamsStruct *BopsParams)
```

Model1, Model2: The two models to consider.

SSICache: SSI cache to be used and updated, if provided (if not NULL - optional).

BopsParams: Additional parameters and callback functions used in the Boolean operation process.

Returns: TRUE if two models intersect, FALSE otherwise.

Description: Check whether two models intersect with each other. The function only detects if two models intersect or not, and does not compute the intersecting curves or models.
See also: MdlBooleanIntersection, MdlBooleanUnion, MdlBooleanSubtraction, , MdlBooleanDtctModelSrfInter, MdlBooleanMerge, MdlBooleanSetTolerances,

6.2.34 MdlBooleanDtctModelSrfInter (mdl_bool.c:2034)

```
int MdlBooleanDtctModelSrfInter(MdlModelStruct *Model1,
                                const CagdSrfStruct *Srf2)
```

Model1: The first model to consider. This model might be updated in place with auxiliary information.

Srf2: The second surface to consider.

Returns: TRUE if intersect, FALSE otherwise.

Description: Detect if the given model and given surface intersect.
See also: MdlBooleanIntersection, MdlBooleanUnion, MdlBooleanSubtraction, , MdlBooleanDetectInter, MdlBooleanMerge, MdlBooleanSetTolerances, , MdlBooleanCut,

6.2.35 MdlBooleanInterCrv (mdl_bool.c:2455)

```
CagdCrvStruct *MdlBooleanInterCrv(const MdlModelStruct *Model1,
                                  const MdlModelStruct *Model2,
                                  int InterType,
                                  MdlModelStruct **InterModel,
                                  MdlBopsParamsStruct *BopsParams)
```

Model1, Model2: The two models to consider.

InterType: 0 for Euclidean space, 1/2 for inter curves in Model1/2. 10/11/12 as above but also aims to merge curves into loops.

InterModel: If not NULL, returns here the model so UV locations could be evaluated into E3.

BopsParams: Additional optional parameters and callback functions used in the Boolean operation process. Can be NULL.

Returns: Resulting intersection curves.

Description: Computes the intersection curve of two models. If InterModel != NULL, also saves an attribute "trimsrf" with the trimmed surface reference, on the returned UV curves.

6.2.36 MdlBooleanIntersection (mdl_bool.c:2192)

```
IPObjectStruct *MdlBooleanIntersection(
                                    const MdlModelStruct *Model1,
                                    const MdlModelStruct *Model2,
                                    struct MvarSrfSrfInterCacheStruct *SSICache,
                                    MdlBopsParamsStruct *BopsParams)
```

Model1, Model2: The two models to consider.

SSICache: SSI cache to be used and updated, if provided (if not NULL - optional).

BopsParams: Additional parameters and callback functions used in the Boolean operation process.

Returns: Resulting boolean.

Description: Computes the Boolean intersection of two models.
See also: MdlBooleanSetTolerances, MdlBooleanUnion, MdlBooleanDtctModelSrfInter, MdlBooleanDtctModelSrfInter,

6.2.37 MdlBooleanMerge (mdl_bool.c:2371)

```
IPObjectStruct *MdlBooleanMerge(const MdlModelStruct *Model1,
                                const MdlModelStruct *Model2,
                                CagdBType StitchBndries)
```

Model1, Model2: The two models to consider.

StitchBndries: TRUE to also stitch shared boundaries.

Returns: Resulting merged model.

Description: Computes the merge of two models. All surfaces and trimming curves are concatenated together and optionally shared boundaries are stitched together.
See also: MdlStitchModel, MdlBooleanSetTolerances, MdlBooleanUnion, MdlBooleanCut, , MdlBooleanIntersection, MdlBooleanSubtraction, MdlBooleanMerge2, MdlBooleanDtctModelSrfInter, MdlBooleanDtctModelSrfInter,

6.2.38 MdlBooleanMerge2 (mdl_bool.c:2405)

```
MdlModelStruct *MdlBooleanMerge2(const MdlModelStruct *Model1,
                                 const MdlModelStruct *Model2,
                                 CagdBType StitchBndries)
```

Model1, Model2: The two models to consider.

StitchBndries: TRUE to also stitch shared boundaries.

Returns: The merged model.

Description: Computes the merge of two models. All surfaces and trimming curves are concatenated together and optionally shared boundaries are stitched together.
See also: MdlStitchModel, MdlBooleanSetTolerances, MdlBooleanUnion, MdlBooleanCut, , MdlBooleanIntersection, MdlBooleanSubtraction, MdlBooleanMerge, MdlBooleanDtctModelSrfInter, MdlBooleanDtctModelSrfInter,

6.2.39 MdlBooleanSetClip2Trim (mdl_bool.c:178)

```
CagdBType MdlBooleanSetClip2Trim(CagdBType Clip2Trim)
```

Clip2Trim: New state of clip surface to trimming curves.

Returns: Old state value.

Description: Sets the state of allowing to clip the surface to the domain set by the trimming curves.
See also: MdlBoolClipTSrfs2TrimDomain,

6.2.40 MdlBooleanSetTolerances (mdl_bool.c:135)

```
CagdRType MdlBooleanSetTolerances(CagdRType SubdivTol,
                                  CagdRType NumerTol,
                                  CagdRType TraceTol)
```

SubdivTol, NumerTol, TraceTol: See MvarSrfSrfInter.

Returns: The last modified tolerance, between (Trace, Numer, Subdiv).

Description: Sets the tolerances to use in the boolean operations computations.
See also:

6.2.41 MdlBooleanSubtraction (mdl_bool.c:2231)

```
IPObjectStruct *MdlBooleanSubtraction(
                                   const MdlModelStruct *Model1,
                                   const MdlModelStruct *Model2,
                                   struct MvarSrfSrfInterCacheStruct *SSICache,
                                   MdlBopsParamsStruct *BopsParams)
```

Model1, Model2: The two models to consider.

SSICache: SSI cache to be used and updated, if provided (if not NULL - optional).

BopsParams: Additional parameters and callback functions used in the Boolean operation process.

Returns: Resulting boolean.

Description: Computes the Boolean subtraction of two models.
See also: MdlBooleanSetTolerances, MdlBooleanUnion, MdlBooleanIntersection, MdlBooleanDtctModelSrfInter, MdlBooleanDtctModelSrfInter, MdlBooleanCut,

6.2.42 MdlBooleanUnion (mdl_bool.c:2153)

```
IPObjectStruct *MdlBooleanUnion(const MdlModelStruct *Model1,
                                const MdlModelStruct *Model2,
                                struct MvarSrfSrfInterCacheStruct *SSICache,
                                MdlBopsParamsStruct *BopsParams)
```

Model1, Model2: The two models to consider.

SSICache: SSI cache to be used and updated, if provided (if not NULL - optional).

BopsParams: Additional parameters and callback functions used in the Boolean operation process.

Returns: Resulting boolean.

Description: Computes the Boolean union of two models.
See also: MdlBooleanIntersection, MdlBooleanSetTolerances, , MdlBooleanDtctModelSrfInter, MdlBooleanDtctModelSrfInter,

6.2.43 MdlClipModelByPlane (mdl_bool.c:3012)

```
MdlModelStruct *MdlClipModelByPlane(const MdlModelStruct *Mdl,
                                    const IrtPlnType Pln,
                                    MdlBooleanType BoolOp)
```

Mdl: Model to clip against plane Pln.

Pln: Clipping plane.

BoolOp: One of CUT, INTERSECTION, SUBTRACTION.

Returns: Portion of Mdl, as a new model, in the positive side of plane Pln. NULL if nothing.

Description: Clips the given model, Mdl, to the portion that is in the positive side of plane Pln.
See also: MdlClipSrfByPlane, MdlClipTrimmedSrfByPlane, MdlInterModelByPlane,

6.2.44 MdlClipSrfByPlane (mdl_bool.c:2804)

```
TrimSrfStruct *MdlClipSrfByPlane(const CagdSrfStruct *Srf,
                                 const IrtPlnType Pln)
```

Srf: Surface to clip against plane Pln.

Pln: Clipping plane.

Returns: Trimmed surface portion of Srf in the positive side of plane Pln. NULL if nothing.

Description: Clips the given surface, Srf, to the portion that is in the positive side of plane Pln. Intersection is assumed to be a closed loop.
See also: MdlClipModelByPlane, MdlClipTrimmedSrfByPlane, MdlClipModelByPlane, , MdlInterModelByPlane,

6.2.45 MdlClipTrimmedSrfByPlane (mdl_bool.c:2836)

```
TrimSrfStruct *MdlClipTrimmedSrfByPlane(const TrimSrfStruct *TSrf,
                                        const IrtPlnType Pln)
```

TSrf: Trimmed Surface to clip against plane Pln.

Pln: Clipping plane.

Returns: Trimmed surface portion of Srf in the positive side of plane Pln. NULL if nothing. Can be a list of such trimmed surfaces.

Description: Clips the given trimmed surface, TSrf, to the portion that is in the positive side of plane Pln. Intersection is assumed to be a closed loop.
See also: MdlClipSrfByPlane, MdlClipModelByPlane, MdlInterModelByPlane,

6.2.46 MdlCnvrtMdl2TrimmedSrfs (mdlcnvrt.c:118)

```
TrimSrfStruct *MdlCnvrtMdl2TrimmedSrfs(const MdlModelStruct *Model,
                                       CagdRType TrimCrvStitchTol)
```

Model: Model to convert to a list of trimmed surfaces.

TrimCrvStitchTol: If positive, process the Euclidean trimming curves as - I. Approximate the trimming curves to piecewise linear curves. II. Make sure the two curves of two surfaces sharing a common, stitching, trimming curves are the same in Euclidean space.

Returns: List of trimming surfaces.

Description: Converting the model into the list of trimming surfaces. Each trimmed surface is updated with the proper boundary/trimming curves ready for tessellation, so each trimming curve is shared by both surfaces it stitches, preventing black holes.
See also: TrimCrvSegNew, TrimCrvNew, TrimsrfNew, MdlExtractUVCrv, , MdlCnvrtSrf2Mdl, MdlCnvrtTrimmedSrf2Mdl, MdlCnvrtMdls2TrimmedSrfs,

6.2.47 MdlCnvrtMdls2TrimmedSrfs (mdlcnvrt.c:77)

```
TrimSrfStruct *MdlCnvrtMdls2TrimmedSrfs(const MdlModelStruct *Models,
                                        CagdRType TrimCrvStitchTol)
```

Models: list of models to convert to a list of trimmed surfaces.

TrimCrvStitchTol: If not zero, process the Euclidean trimming curves as - I. Approximate the trimming curves to piecewise linear curves. II. Make sure the two curves of two surfaces sharing a common, stitching, trimming curves are the same in Euclidean space.

Returns: List of trimming surfaces.

Description: Converting a list of models into the list of trimming surfaces. Each trimmed surface is updated with the proper boundary/trimming curves ready for tessellation, so each trimming curve is shared by both surfaces it stitches, preventing black holes.
See also: TrimCrvSegNew, TrimCrvNew, TrimsrfNew, MdlExtractUVCrv, , MdlCnvrtSrf2Mdl, MdlCnvrtTrimmedSrf2Mdl, MdlCnvrtMdl2TrimmedSrfs,

6.2.48 MdlCnvrtSrf2Mdl (mdlcnvrt.c:615)

```
MdlModelStruct *MdlCnvrtSrf2Mdl(const CagdSrfStruct *Srf, int StitchModel)
```

Srf: A surface to convert to a model.

StitchModel: TRUE to also stitch the model. This is need as a final step once all surfaces were added.

Returns: A model.

Description: Converting the given surface into a model.
See also: MdlCnvrtMdl2TrimmedSrfs, MdlCnvrtTrimmedSrf2Mdl, MdlCnvrtSrfs2Mdls,

6.2.49 MdlCnvrtSrfs2Mdls (mdlcnvrt.c:580)

```
MdlModelStruct *MdlCnvrtSrfs2Mdls(const CagdSrfStruct *Srfs, int StitchModel)
```

Srfs: A linked list of surfaces to convert to models.

StitchModel: TRUE to also stitch the model. This is need as a final step once all surfaces were added.

Returns: Models.

Description: Converting the given linked list of surfaces into models.
See also: MdlCnvrtMdl2TrimmedSrfs, MdlCnvrtTrimmedSrf2Mdl, MdlCnvrtSrf2Mdl,

6.2.50 MdlCnvrtTrimmedSrf2Mdl (mdlcnvrt.c:674)

```
MdlModelStruct *MdlCnvrtTrimmedSrf2Mdl(const TrimSrfStruct *TSrf,
                                       int MergeTCrvsIntoLoops)
```

TSrf: A trimmed surface to convert to a model.

MergeTCrvsIntoLoops: TRUE to also merge trimming curves in loops into one large loop curve.

Returns: A model.

Description: Converting the given trimmed surface into a model.
See also: MdlCnvrtMdl2TrimmedSrfs, MdlCnvrtSrf2Mdl, MdlCnvrtTrimmedSrfs2Mdls,

6.2.51 MdlCnvrtTrimmedSrfs2Mdl (mdlcnvrt.c:754)

```
MdlModelStruct *MdlCnvrtTrimmedSrfs2Mdl(const TrimSrfStruct *TSrfs,
                                        int MergeTCrvsIntoLoops,
                                        int StitchMdl)
```

TSrfs: A linked list of trimmed surfaces to convert to models.

MergeTCrvsIntoLoops: TRUE to also merge trimming curves in loops into one large loop curve.

StitchMdl: True to also apply stitching on the model, merging trimming curves that are the name (in Euclidean space).

Returns: Converted Model or NULL if error.

Description: Converting the given linked list of trimmed surfaces into one model.
See also: MdlCnvrtMdl2TrimmedSrfs, MdlCnvrtSrf2Mdl, MdlCnvrtTrimmedSrf2Mdl, , MdlCnvrtTrimmedSrfs2Mdls,

6.2.52 MdlCnvrtTrimmedSrfs2Mdls (mdlcnvrt.c:639)

```
MdlModelStruct *MdlCnvrtTrimmedSrfs2Mdls(const TrimSrfStruct *TSrfs,
                                         int MergeTCrvsIntoLoops)
```

TSrfs: A linked list of trimmed surfaces to convert to models.

MergeTCrvsIntoLoops: TRUE to also merge trimming curves in loops into one large loop curve.

Returns: Converted models.

Description: Converting the given linked list of n trimmed surfaces into n models.
See also: MdlCnvrtMdl2TrimmedSrfs, MdlCnvrtSrf2Mdl, MdlCnvrtTrimmedSrf2Mdl,

6.2.53 MdlCreateCubeSpherePrim (mdl_prim.c:1353)

```
int MdlCreateCubeSpherePrim(int CubeTopoSphere)
```

CubeTopoSphere: TRUE to construct spheres using cube-topology.

Returns: Old value setting.

Description: Select to create sphere using cube-topology, if TRUE.

6.2.54 MdlDbg (mdl_dbg.c:39)

debugging

```
void MdlDbg(void *Obj)
```

Obj: A model object - to be printed to stderr.

Returns: void

Description: Prints model objects to stderr. Should be linked to programs for debugging purposes, so model objects may be inspected from a debugger.
See also: MdlDbg1,

6.2.55 MdlDbg1 (mdl_dbg.c:68)

debugging

```
void MdlDbg1(void *Obj)
```

Obj: A model object - to be printed to stderr.

Returns: void

Description: Prints model objects to stderr. Should be linked to programs for debugging purposes, so model objects may be inspected from a debugger.
See also: MdlDbg,

6.2.56 MdlDbg2 (mdl_dbg.c:97)

debugging

```
void MdlDbg2(void *Obj)
```

Obj: A model object - to be printed to stderr.

Returns: void

Description: Prints model information to stderr. Should be linked to programs for debugging purposes, so model objects may be inspected from a debugger.

6.2.57 MdlDbgBoolImprovedPointsCacheReport (mdl3bool.c:922)

```
void MdlDbgBoolImprovedPointsCacheReport(
                                  const MdlImprovedPointsCacheStruct *IPCache)
```

IPCache: The improved points cache to report to stderr.

Returns: void

Description: Report on the improved points cache.

6.2.58 MdlDbgDsp (mdl_dbg.c:141)

debugging

```
void MdlDbgDsp(void *Obj, CagdRType Trans, int Clear)
```

Obj: A model object - to be displayed.

Trans: Amount to translate each surface in the surface center normal direction.

Clear: TRUE to first clear the display device.

Returns: void

Description: Display an entire model object by drawing all tensor product surfaces and their trimming curves. Should be linked to programs for debugging purposes, so model objects may be inspected from a debugger. Trimming curves will be colored as follows:

1. Green - for trimming curves that are in the output set.
2. Yellow - for undetermined trimming curves.
3. Magenta - for trimming curves that will be purged.

6.2.59 MdlDbgDsp2 (mdl_dbg.c:262)

debugging

```
void MdlDbgDsp2(void *Obj, CagdRType Trans, int Clear)
```

Obj: A model object - to be displayed.

Trans: Amount to translate each surface in the surface center normal direction.

Clear: TRUE to first clear the display device.

Returns: void

Description: Display an entire model object by drawing all trimming curves in Euclidean space after mapping them through their tensor product surfaces. N Should be linked to programs for debugging purposes, so model objects may be inspected from a debugger.

6.2.60 MdlDbgMC (mdl_dbg.c:408)

debugging.

```
int MdlDbgMC(const MdlModelStruct *Mdl, int Format)
```

Mdl: To dump to stderr/display all its trimming curves in UV space, for each surface.

Format: 0 to write it to stderr all data, 1 to write it to stderr but only end points of trimming curves. 9 to display the data in a display device. 10 same as 9 but to also clear first.

Returns: TRUE if successful, FALSE otherwise.

Description: Debug routine to dump to stderr/display the UV trimming curves of all surfaces in a given model.

6.2.61 MdlDbgRC (mdl_dbg.c:513)

debugging.

```
int MdlDbgRC(const MdlTrimSegRefStruct *Refs, int Format)
```

Refs: To dump its trimming curves in UV space.

Format: 0 to write it to stderr all data, 1 to write it to stderr but only end points of trimming curves. 9 to display the data in a display device. 10 same as 9 but to also clear first.

Returns: TRUE if successful, FALSE otherwise.

Description: Debug routine to dump the UV trimming curves in the given ref. list.

6.2.62 MdlDbgRC2 (mdl_dbg.c:549)

debugging.

```
int MdlDbgRC2(const MdlTrimSegRefStruct *Refs,
              const MdlTrimSrfStruct *TSrf,
              int Format)
```

Refs: To dump its trimming curves in UV space.

TSrf: Trimmed surface these references should list (and its trimming curves).

Format: 0 to write it to stderr all data, 1 to write it to stderr but only end points of trimming curves. 9 to display the data in a display device. 10 same as 9 but to also clear first.

Returns: TRUE if successful, FALSE otherwise.

Description: Debug routine to dump the UV trimming curves in the given ref. list.

6.2.63 MdlDbgSC (mdl_dbg.c:481)

debugging.

```
int MdlDbgSC(const MdlTrimSrfStruct *TSrf, int Format)
```

TSrf: To dump its trimming curves in UV space.

Format: 0 to write it to stderr all data, 1 to write it to stderr but only end points of trimming curves. 9 to display the data in a display device. 10 same as 9 but to also clear first.

Returns: TRUE if successful, FALSE otherwise.

Description: Debug routine to dump the UV trimming curves of a given surface.

6.2.64 MdlDbgTC (mdl_dbg.c:454)

debugging.

```
int MdlDbgTC(const MdlTrimSegStruct *TSegs,
             const MdlTrimSrfStruct *TSrf,
             int Format)
```

TSegs: To dump to stderr/display all its trimming curves in UV space.

TSrf: If not NULL - dump/display only its trimming curves.

Format: 0 to write it to stderr all data, 1 to write it to stderr but only end points of trimming curves. 9 to display the data in a display device. 10 same as 9 but to also clear first.

Returns: TRUE if successful, FALSE otherwise.

Description: Debug routine to dump to stderr/display the UV trimming curves of the list of trimming curves.

6.2.65 MdlDbgVsl (mdl_dbg.c:1648)

debugging.

```
void MdlDbgVsl(const MdlModelStruct *Model,
               CagdBType TCrvs,
               CagdBType TSrfs,
               CagdBType Srfs)
```

Model: To verify.

TCrvs: TRUE to dump the trimming curves. FALSE to ignore. will be colored green if shared by two different surfaces, and color red if shared by one.

TSrfs: TRUE to dump the trimmed surfaces. FALSE to ignore.

Srfs: TRUE to dump the tensor product surfaces. FALSE to ignore.

Returns: void

Description: Debug routine to verify consistency of a model, visually. Same as MdlDebugVisual but send the geometry directly to a display device.

6.2.66 MdlDebugHandleTCrvLoops (mdl_dbg.c:585)

debugging.

```
int MdlDebugHandleTCrvLoops(const MdlTrimSrfStruct *TSrf,
                            const MdlLoopStruct *Loops,
                            const CagdPType Trans,
                            int Display,
                            int TrimEndPts)
```

TSrf: To dump its trimming loops in UV space. Can be NULL.

Loops: To dump as trimming curves in UV space.

Trans: To translate all the curves.

Display: TRUE to display the data, FALSE to write it to stdout.

TrimEndPts: TRUE to dump only the trimming curves' end points.

Returns: TRUE if successful, FALSE otherwise.

Description: Debug routine to dump the UV trimming curve loops.

6.2.67 MdlDebugHandleTSrfCrvs (mdl_dbg.c:628)

debugging.

```
int MdlDebugHandleTSrfCrvs(const MdlTrimSegStruct *TCrvs,
                           const MdlTrimSrfStruct *TSrf,
                           const CagdPType Trans,
                           int Display,
                           int TrimEndPts)
```

TCrvs: List of trimming curves to dump.

TSrf: If not NULL - dump only its trimming curves.

Trans: To translate all the curves. Can be NULL.

Display: TRUE to display the data, FALSE to write it to stdout.

TrimEndPts: TRUE to dump only the trimming curves' end points.

Returns: TRUE if successful, FALSE otherwise.

Description: Debug routine to dump a list of trimming curves.

6.2.68 MdlDebugHandleTSrfRefCrvs (mdl_dbg.c:786)

debugging.

```
int MdlDebugHandleTSrfRefCrvs(const MdlTrimSegRefStruct *Refs,
                              const MdlTrimSrfStruct *TSrf,
                              const CagdPType Trans,
                              int Loop,
                              int Display,
                              int TrimEndPts)
```

Refs: List of references to trimming curves to dump.
TSrf: If not NULL - dump only its trimming curves.
Trans: To translate all the curves. Can be NULL.
Loop: Unique loop index.
Display: TRUE to display the data, FALSE to write it to stdout.
TrimEndPts: TRUE to dump only the trimming curves' end points.
Returns: TRUE if successful, FALSE otherwise.
Description: Debug routine to dump a list of refs to trimming curves.

6.2.69 MdlDebugVerify (mdl_dbg.c:1357)

debugging.

```
int MdlDebugVerify(const MdlModelStruct *Model, int Complete, int TestLoops)
```

Model: To verify.
Complete: TRUE if the model is indeed complete and all back pointers should be set at this time.
TestLoops: TRUE to also verify that each loop is indeed a loop.
Returns: TRUE if successful, FALSE otherwise.
Description: Debug routine to verify consistency of a model, using default tolerances.

6.2.70 MdlDebugVerifyEps (mdl_dbg.c:1391)

debugging.

```
int MdlDebugVerifyEps(const MdlModelStruct *Model,
                      int Complete,
                      int TestLoops,
                      CagdRType TCrvTol,
                      CagdRType UVVertexTol,
                      CagdRType VertexTol)
```

Model: To verify.
Complete: TRUE if the model is indeed complete and all back pointers should be set at this time.
TestLoops: TRUE to also verify that each loop is indeed a loop.
TCrvTol: The tolerance for comparing trimming curves.
UVVertexTol: The tolerance for comparing the end points of trimming curves in UV-space.
VertexTol: The tolerance for comparing the end points of trimming curves in Euclidean space.
Returns: TRUE if successful, FALSE otherwise.
Description: Debug routine to verify consistency of a model.

6.2.71 MdlDebugVerifyTrimSeg (mdl_dbg.c:1037)

debugging.

```
int MdlDebugVerifyTrimSeg(const MdlTrimSegStruct *TSeg, int VerifyBackPtrs)
```

TSeg: To verify.
VerifyBackPtrs: TRUE to also verify the two references to this segment and that they are referencing it in reverse order.
Returns: TRUE if successful, FALSE otherwise.
Description: Debug routine to verify consistency of a trimming curve segment, using default tolerances.

6.2.72 MdlDebugVerifyTrimSegEps (mdl_dbg.c:1061)

debugging.

```
int MdlDebugVerifyTrimSegEps(const MdlTrimSegStruct *TSeg,
                             int VerifyBackPtrs,
                             CagdRType TCrvTol)
```

TSeg: To verify.

VerifyBackPtrs: TRUE to also verify the two references to this segment and that they are referencing it in reverse order.

TCrvTol: The tolerance for comparing trimming curves.

Returns: TRUE if successful, FALSE otherwise.

Description: Debug routine to verify consistency of a trimming curve segment.

6.2.73 MdlDebugVerifyTrimSegsArcLen (mdl_dbg.c:1202)

debugging.

```
int MdlDebugVerifyTrimSegsArcLen(const MdlTrimSegStruct *TSegs)
```

TSegs: To verify.

Returns: TRUE if successful, FALSE otherwise.

Description: Debug routine to verify arc length of list of trimming curve segments.

6.2.74 MdlDebugVerifyTrimSrf (mdl_dbg.c:1241)

debugging.

```
int MdlDebugVerifyTrimSrf(const MdlTrimSrfStruct *MdlTrimSrf,
                          int Complete,
                          int TestLoops)
```

MdlTrimSrf: To verify.

Complete: TRUE if the model is indeed complete and all back pointers should be set at this time.

TestLoops: TRUE to also verify that each loop is indeed a loop.

Returns: TRUE if successful, FALSE otherwise.

Description: Debug routine to verify consistency of a model.

6.2.75 MdlDebugVerifyTrimSrfEps (mdl_dbg.c:1270)

debugging.

```
int MdlDebugVerifyTrimSrfEps(const MdlTrimSrfStruct *MdlTrimSrf,
                             int Complete,
                             int TestLoops,
                             CagdRType TCrvTol,
                             CagdRType UVVertexTol)
```

MdlTrimSrf: To verify.

Complete: TRUE if the model is indeed complete and all back pointers should be set at this time.

TestLoops: TRUE to also verify that each loop is indeed a loop.

TCrvTol: Tolerance to use on trimming curve segments.

UVVertexTol: Tolerance to use in merging curves into loops.

Returns: TRUE if successful, FALSE otherwise.

Description: Debug routine to verify consistency of a model.

6.2.76 MdlDebugVisual (mdl_dbg.c:1491)

debugging.

```
IPObjectStruct *MdlDebugVisual(const MdlModelStruct *Model,
                               CagdBType TCrvs,
                               CagdBType TSrfs,
                               CagdBType Srfs)
```

Model: To verify.

TCrvs: TRUE to dump the trimming curves. FALSE to ignore. will be colored green if shared by two different surfaces, and color red if shared by one.

TSrfs: TRUE to dump the trimmed surfaces. FALSE to ignore.

Srfs: TRUE to dump the tensor product surfaces. FALSE to ignore.

Returns: A list object holding the different extracted entities of the model.

Description: Debug routine to verify consistency of a model, visually.

6.2.77 MdlDebugWriteTrimSegs (mdl_dbg.c:984)

debugging.

```
int MdlDebugWriteTrimSegs(const MdlTrimSegStruct *TSegs,
                          const MdlTrimSrfStruct *TSrf,
                          const CagdPType Trans)
```

TSegs: To dump all its trimming curves in the list.

TSrf: If not NULL - dump only its trimming curves.

Trans: To translate all the curves. Can be NULL.

Returns: TRUE if successful, FALSE otherwise.

Description: Debug routine to dump a list of UV trimming curves model.

6.2.78 MdlDivideTrimCrv (mdl_aux.c:251)

allocation

```
MdlTrimSegStruct *MdlDivideTrimCrv(MdlTrimSegStruct *Seg,
                                   const CagdPtStruct *Pts,
                                   int Idx,
                                   CagdRType Eps,
                                   int *Proximity)
```

Seg: Trimming segment to divide. Seg will return, in place, the first curve in the divided set.

Pts: Parameters at which to split.

Idx: Index of parameter in Pts points: 0 for X, 1 for Y, etc.

Eps: parameter closer than Eps to boundary or other parameters are ignored.

Proximity: Proximity bit set to end points - see CagdCrvSubdivAtParams2.

Returns: The rest of the divided segments.

Description: Subdivides the given segment at the specified parameters values.
See also: CagdCrvSubdivAtParams2,

6.2.79 MdlEnsureMdlTrimCrvsPrecision (mdl_aux.c:505)

```
void MdlEnsureMdlTrimCrvsPrecision(MdlModelStruct *Mdl)
```

Mdl: To ensure the precision of its trimming curves.

Returns: void

Description: Ensure adjacent trimming curves have the precise same end points.
See also: MdlEnsureTSrfTrimCrvsPrecision, MdlEnsureTSrfTrimLoopPrecision,

6.2.80 MdlEnsureTSrfTrimCrvsPrecision (mdl_aux.c:532)

```
void MdlEnsureTSrfTrimCrvsPrecision(MdlTrimSrfStruct *MdlTrimSrf)
```

MdlTrimSrf: To ensure the precision of its trimming curves.

Returns: void

Description: Ensure adjacent trimming curves have the precise same end points.
See also: MdlEnsureMdlTrimCrvsPrecision, MdlEnsureTSrfTrimLoopPrecision,

6.2.81 MdlEnsureTSrfTrimLoopPrecision (mdl_aux.c:562)

```
int MdlEnsureTSrfTrimLoopPrecision(MdlLoopStruct *Loop,
                                   MdlTrimSrfStruct *MdlTrimSrf,
                                   CagdRType Tol)
```

Loop: To ensure the precision of its trimming curves.

MdlTrimSrf: To ensure the precision of its trimming curves in Loop.

Tol: Tolerance to require to validate this loop.

Returns: TRUE if precise, FALSE if imprecise beyond tolerance Tol.

Description: Ensure adjacent trimming curves have the precise same end points.
See also: MdlEnsureMdlTrimCrvsPrecision, MdlEnsureMdlTrimCrvsPrecision,

6.2.82 MdlExportTrimmingCurves (mdl_dbg.c:1704)

```
void MdlExportTrimmingCurves(MdlTrimSrfStruct *MdlTrimSrf,
                             MdlTrimSegStruct *NewSegs,
                             const char *Path)
```

MdlTrimSrf: The trimmed surface

NewSegs: New trimming segments.

Path: Output path.

Returns: void

Description: Exports old and new trimming curves with of a given trimmed surface in R3 to files in the specified path.

6.2.83 MdlExtractUVCrv (mdlcnvrt.c:862)

```
CagdCrvStruct *MdlExtractUVCrv(const MdlTrimSrfStruct *MdlSrf,
                               const MdlTrimSegStruct *MdlSeg)
```

MdlSrf: Model's trimming surface.

MdlSeg: Model's trimming curve segment.

Returns: Extracted UV curve.

Description: Extracting the UV curve from MdlTrimSegStruct depending on the position of the current model surface.

6.2.84 MdlFilterOutCrvs (mdl_aux.c:37)

```
MdlTrimSegStruct *MdlFilterOutCrvs(MdlTrimSegStruct *TSegs)
```

TSegs: Curve segments to examine if inside both its trimmed surfaces.

Returns: Filtered (inside) list of TSegs (in its trim. srfs).

Description: Examines the given list of curve segments in TSrf domain if inside or not, against its trimming surfaces.

6.2.85 MdlGetLoopSegIndex (mdl_ptch.c:134)

```
IritIntPtrSizeType MdlGetLoopSegIndex(const MdlTrimSegRefStruct *TrimSeg,
                                      const MdlTrimSegStruct *TrimSegList)
```

TrimSeg: To search in TrimSegList for its index.

TrimSegList: List of trimming curve segments.

Returns: Index in list, or zero if not found. This is a special integer of a size of a pointer.

Description: Returns the index of TrimSeg in TrimSegList, first index is 1. Index is going to be negative if the Reversed flag is on.
See also: MdlGetSrfIndex,

6.2.86 MdlGetModelTrimSegRef (mdl_prim.c:230)

```
int MdlGetModelTrimSegRef(const MdlModelStruct *Mdl,
                          const MdlTrimSegStruct *TSeg,
                          MdlTrimSegRefStruct **TSegRef1,
                          MdlTrimSrfStruct **TSrf1,
                          MdlTrimSegRefStruct **TSegRef2,
                          MdlTrimSrfStruct **TSrf2)
```

Mdl: In which to find to to two trim curve segments references to.

TSeg: Trimming curve segment to seek references to.

TSegRef1: Where to place the first reference found.

TSrf1: The trimmed surface of the first reference.

TSegRef2: Where to place the second reference found.

TSrf2: The trimmed surface of the second reference.

Returns: Number of references found.

Description: Fetches the up to two trim curve segments references to TSeg in Model.
See also: MdlGetSrfTrimSegRef,

6.2.87 MdlGetOtherSegRef (mdl_gen.c:1222)

```
MdlTrimSegRefStruct *MdlGetOtherSegRef(const MdlTrimSegRefStruct *SegRef,
                                       const MdlTrimSrfStruct *TSrf)
```

SegRef: Reference to trimming curve segment we seek the second reference, if exist.

TSrf: Trimmed surface SegRef belongs to.

Returns: The second reference ot NULL if none.

Description: Find the second reference to the given reference of a trimming curve, if any.
See also: MdlGetOtherSegRef2,

6.2.88 MdlGetOtherSegRef2 (mdl_gen.c:1254)

```
MdlTrimSegRefStruct *MdlGetOtherSegRef2(const MdlTrimSegRefStruct *SegRef,
                                        const MdlTrimSrfStruct *TSrf,
                                        MdlTrimSrfStruct **OtherTSrf,
                                        MdlLoopStruct **OtherLoop)
```

SegRef: Reference to trimming curve segment we seek the second reference ,if exist.

TSrf: Trimmed surface SegRef belongs to.

OtherTSrf: The other trimmed surface holding other ref., if any.

OtherLoop: The other loop holding other ref., if any.

Returns: The second reference ot NULL if none.

Description: Find the second reference to the given reference of a trimming curve if any. Same as MdlGetOtherSegRef but returns more info.
See also: MdlGetOtherSegRef,

6.2.89 MdlGetSrfIndex (mdl_ptch.c:168)

```
IritIntPtrSizeType MdlGetSrfIndex(const MdlTrimSrfStruct *Srf,
                                  const MdlTrimSrfStruct *TrimSrfList)
```

Srf: To search in TrimSrfList for its index.

TrimSrfList: List of surfaces.

Returns: Index in list, or zero if not found. This is a special integer of a size of a pointer.

Description: Returns the index of an Srf in TrimSrfList, first index is 1.
See also: MdlGetLoopSegIndex,

6.2.90 MdlGetSrfTrimSegRef (mdl_prim.c:274)

```
MdlTrimSegRefStruct *MdlGetSrfTrimSegRef(const MdlTrimSrfStruct *TSrf,
                                         const MdlTrimSegStruct *TSeg)
```

TSrf: Trimmed surface to search the trimming segment reference in.

TSeg: The segment to seek the trimmed curve reference for.

Returns: Reference if found, NULL otherwise.

Description: Find the reference to TSeg in TSrf.
See also: MdlGetModelTrimSegRef,

6.2.91 MdlGetTrimmingCurves (mdl_prim.c:317)

trimming curves

```
CagdCrvStruct *MdlGetTrimmingCurves(const MdlTrimSrfStruct *TrimSrf,
                                    CagdBType MergeLoops,
                                    CagdBType ParamSpace,
                                    CagdBType EvalEuclid)
```

TrimSrf: Trimmed surface to extract trimming curves from.

MergeLoops: TRUE to merge loops into a single curve.

ParamSpace: TRUE for curves in parametric space, FALSE for 3D Euclidean space.

EvalEuclid: If TRUE and ParamSpace is FALSE, evaluate Euclidean curve even if one exists.

Returns: List of trimming curves of TrimSrf.

Description: Extracts the trimming curves of the given model's trimmed surface.
See also: TrimGetTrimmingCurves, TrimGetTrimmingCurves2,

6.2.92 MdlGetTrimmingCurvesEndPts (mdl_aux.c:824)

```
CagdPType *MdlGetTrimmingCurvesEndPts(MdlModelStruct *Mdl, int *N)
```

Mdl: Model to process.

N: Will be updated with length of the return list (number of points where 3N is number of real values returned).

Returns: A vector of 3N real values holding the N points, allocated dynamically.

Description: Derives a list of all trimming curves start/terminating points. Aim is made to get unique points (so shared vertices by several trimming curves will show up once). As a side effect, all trimming curve segments in Mdl will be tagged with the index into the point list, as "StartVertex" and "TrmntVrtx" attributes.
See also:

6.2.93 MdlGetUVEpsInsideTSrf (mdl2bool.c:1193)

```
int MdlGetUVEpsInsideTSrf(MdlTrimSrfStruct *TSrf, CagdUVType UVInside)
```

TSrf: to find a point inside it.

UVInside: Will be updated by the computed inside location.

Returns: TRUE if successful, FALSE otherwise.

Description: Find a point epsilon inside the valid domain of TSrf by moving a small amount in a direction prescribed by an adjacent surface to TSrf.

6.2.94 MdlGetUVLocationInLoop (mdl_aux.c:416)

```
int MdlGetUVLocationInLoop(const MdlLoopStruct *L,
                           const MdlTrimSrfStruct *TSrf,
                           CagdUVType UV)
```

L: Loop to fetch UV value from.

TSrf: Surface to fetch UV value for.

UV: Computed UV value.

Returns: TRUE if successful, FALSE otherwise.

Description: Fetches one UV value from the give loop L in surface TSrf.

6.2.95 MdlInterModelByCurve (mdl_bool.c:2944)

```
CagdPtStruct *MdlInterModelByCurve(const MdlModelStruct *Mdl,
                                   const CagdCrvStruct *Crv,
                                   CagdRType Eps,
                                   int DtctInters)
```

Mdl: Model to intersect against plane Pln.

Crv: Intersecting curve.

Eps: Tolerance of computation.

DtctInters: TRUE to only detect intersection (and return non-NULL ptr).

Returns: Parameter values along Crv of intersection point or NULL if none. If DtctInters is TRUE, returns a NULL only if no intersection.

Description: Intersects the given model, Mdl, with curve Crv
See also: MdlClipSrfByPlane, MdlClipTrimmedSrfByPlane, MdlClipModelByPlane, , MvarSrfSrfInter, UserCntrSrfWithPlane, MdlInterSrfByPlane, , MdlInterModelByPlane,

6.2.96 MdlInterModelByPlane (mdl_bool.c:2898)

```
CagdCrvStruct *MdlInterModelByPlane(const MdlModelStruct *Mdl,
                                    const IrtPlnType Pln)
```

Mdl: Model to intersect against plane Pln.

Pln: Intersecting plane.

Returns: Intersection curves or NULL if none.

Description: Intersects the given model, Mdl, with plane Pln.
See also: MdlClipSrfByPlane, MdlClipTrimmedSrfByPlane, MdlClipModelByPlane, , MvarSrfSrfInter, UserCntrSrfWithPlane, MdlInterSrfByPlane, , MdlInterModelByCurve,

6.2.97 MdlInterSrfByPlane (mdl_bool.c:2748)

```
CagdCrvStruct *MdlInterSrfByPlane(const CagdSrfStruct *Srf,
                                  const IrtPlnType Pln)
```

Srf: Surface to intersect against plane Pln.

Pln: intersecting plane.

Returns: Intersection curves or NULL if none.

Description: Intersects the given surface, Srf, with plane Pln.
See also: MdlClipSrfByPlane, MdlClipTrimmedSrfByPlane, MdlClipModelByPlane, , MvarSrfSrfInter, UserCntrSrfWithPlane, MdlInterModelByPlane,

6.2.98 MdlIsLoopNested (mdl_aux.c:462)

```
int MdlIsLoopNested(const MdlLoopStruct *L, const MdlTrimSrfStruct *TSrf)
```

L: Loop to test if disjoint with respect to trimming loops in TSrf.

TSrf: Surface to examine its trimming loops against loop L.

Returns: 0 if disjoint, 1 if L contains a loop of TSrf, 2 if L is contained in a loop of TSrf, in a valid domain.

Description: Checks if the given loops is completely disjoint with respect to the other loops.
See also:

6.2.99 MdlIsPointInsideModel (mdl_aux.c:701)

```
CagdBType MdlIsPointInsideModel(const CagdVType Pnt, const MdlModelStruct *Mdl)
```

Pnt: Point to examined.

Mdl: The model.

Returns: TRUE if the point lies inside the model, and FALSE otherwise.

Description: Checks if a given points lies inside a given Model. Several rays starting from the point are fired, and the number of intersection points with the model is examined, if the number is odd then the point is outside the model, and inside otherwise.
See also: TrimSrfIntersectRay, MdlIsPointInsideTrimLoop, MdlIsPointInsideTrimSrf,

6.2.100 MdlIsPointInsideTrimLoop (mdl_aux.c:374)

inclusion

```
int MdlIsPointInsideTrimLoop(const MdlTrimSrfStruct *TSrf,
                             const MdlLoopStruct *Loop,
                             CagdUVType UV)
```

TSrf: The owner of these trimming curves.

Loop: Trimming loop to consider.

UV: Parametric location.

Returns: Number of intersection. Even means outside.

Description: Returns TRUE if the given UV value is inside the domain prescribed by the trimming loop.
See also: TrimIsPointInsideTrimUVCrv, TrimIsPointInsideTrimSrf, , TrimIsPointInsideTrimCrvs, MdlIsPointInsideTrimSrf,

6.2.101 MdlIsPointInsideTrimSrf (mdl_aux.c:340)

inclusion

```
CagdBType MdlIsPointInsideTrimSrf(const MdlTrimSrfStruct *TSrf,
                                  CagdUVType UV)
```

TSrf: Trimming curves to consider.

UV: Parametric location.

Returns: TRUE if inside, FALSE otherwise.

Description: Returns TRUE if the given UV value is inside the domain prescribed by the trimming curves of TSrf.

See also: TrimIsPointInsideTrimUVCrv, TrimIsPointInsideTrimSrf, , TrimIsPointInsideTrimCrvs, MdlIsPointInsideTrimLoop,

6.2.102 MdlLoopCopy (mdl_gen.c:454)

allocation

```
MdlLoopStruct *MdlLoopCopy(const MdlLoopStruct *MdlLoop,
                           const MdlTrimSegStruct *TrimSegList)
```

MdlLoop: A trimming surface to duplicate.

TrimSegList: The original trimmed segments.

Returns: A trimming surface structure.

Description: Duplicates a trimming surface structure.

6.2.103 MdlLoopCopyList (mdl_gen.c:482)

copy

```
MdlLoopStruct *MdlLoopCopyList(const MdlLoopStruct *MdlLoopList,
                               const MdlTrimSegStruct *TrimSegList)
```

MdlLoopList: To be copied.

TrimSegList: The original trimmed segments.

Returns: A duplicated list of Loops.

Description: Allocates and copies a list of Loops structures.

6.2.104 MdlLoopFree (mdl_gen.c:131)

allocation

```
void MdlLoopFree(MdlLoopStruct *MdlLoop)
```

MdlLoop: A loop to free.

Returns: void

Description: Deallocates a Model Loop structure.

6.2.105 MdlLoopFreeList (mdl_gen.c:151)

allocation

```
void MdlLoopFreeList(MdlLoopStruct *MdlLoopList)
```

MdlLoopList: A list of loops to free.

Returns: void

Description: Deallocates a Model Loop List structure.

6.2.106 MdlLoopNew (mdl_gen.c:922)

allocation

```
MdlLoopStruct *MdlLoopNew(MdlTrimSegRefStruct *MdlTrimSegRefList)
```

MdlTrimSegRefList: List of model loops forming the trimming loop.

Returns: A model loop.

Description: Allocates a model loop structure.

6.2.107 MdlModelBBox (mdl_bbox.c:30)

bbox

```
CagdBBoxStruct *MdlModelBBox(const MdlModelStruct *Mdl, CagdBBoxStruct *BBox)
```

bounding box

Mdl: A model to compute a bounding box for.

BBox: Where bounding information is to be saved.

Returns: BBox.

Description: Computes a bounding box for a freeform model.
See also: MdlModelListBBox, CagdCrvListBBox, CagdSrfBBox, GMBBSetBBoxPrecise, , MdlModelTSrfTCrvsBBox,

6.2.108 MdlModelCopy (mdl_gen.c:593)

allocation

```
MdlModelStruct *MdlModelCopy(const MdlModelStruct *Model)
```

Model: A Model to duplicate.

Returns: A Model structure.

Description: Duplicates a Model structure.

6.2.109 MdlModelCopyList (mdl_gen.c:628)

copy

```
MdlModelStruct *MdlModelCopyList(const MdlModelStruct *ModelList)
```

ModelList: To be copied.

Returns: A duplicated list of trimming surfaces.

Description: Allocates and copies a list of Model structures.

6.2.110 MdlModelFree (mdl_gen.c:220)

allocation

```
void MdlModelFree(MdlModelStruct *Model)
```

Model: A Model to free.

Returns: void

Description: Deallocates a Model structure.

6.2.111 MdlModelFreeList (mdl_gen.c:241)

allocation

```
void MdlModelFreeList(MdlModelStruct *Model)
```

Model: A list of trimmed surface to free.

Returns: void

Description: Deallocates a list of Model structures.

6.2.112 MdlModelIsClosed (mdl_aux.c:758)

```
int MdlModelIsClosed(const MdlModelStruct *Model)
```

Model: Model to example if it is closed.

Returns: TRUE if closed, FALSE otherwise.

Description: Checks if a model is closed. A model is closed if all its trimming curves are shared by two surfaces.

6.2.113 MdlModelListBBox (mdl_bbox.c:82)

bbox

bounding box

```
CagdBBoxStruct *MdlModelListBBox(const MdlModelStruct *Mdls,
                                 CagdBBoxStruct *BBox)
```

Mdls: List of models to compute a bounding box for.

BBox: Where bounding information is to be saved.

Returns: BBox.

Description: Computes a bounding box for freeform models.
See also: MdlModelBBox, CagdCrvListBBox, CagdSrfBBox, GMBBSetBBoxPrecise, , MdlModelTSrfTCrvsBBox,

6.2.114 MdlModelMatTransform (mdl_gen.c:1448)

models

```
void MdlModelMatTransform(MdlModelStruct *Model, CagdMType Mat)
```

Model: Model to transform.

Mat: Homogeneous transformation to apply to TV.

Returns: void

Description: Transforms, in place, the given model as specified by homogeneous matrix Mat.

6.2.115 MdlModelNegate (mdl_bool.c:2687)

```
MdlModelStruct *MdlModelNegate(const MdlModelStruct *Model)
```

Model: The model to negate.

Returns: Resulting negation.

Description: Computes the inside out (negation of the given model as a new model.
See also:

6.2.116 MdlModelNew (mdl_gen.c:1312)

Allocation

```
MdlModelStruct *MdlModelNew(CagdSrfStruct *Srf,
                            CagdCrvStruct **LoopList,
                            int NumLoops,
                            CagdBType HasTopLvlTrim)
```

Srf: The original surface to be made into a model. Used in place.

LoopList: A vector of trimming loops (lists of curves). Used in place.

NumLoops: Size of vector LoopList.

HasTopLvlTrim: If FALSE, add outer loops boundary.

Returns: The Model.

Description: Constructor for a Model.

6.2.117 MdlModelNew2 (mdl_gen.c:1381)

Allocation

```
MdlModelStruct *MdlModelNew2(MdlTrimSrfStruct *TrimSrfs,
                             MdlTrimSegStruct *TrimSegs)
```

TrimSrfs: Trimming surfaces. Used in place.

TrimSegs: A list of trimming segments. Used in place.

Returns: The construct Model.

Description: Constructor for a Model. No attempt is mode to verify the consistency of the given data and proper pointers between the different data strcutres.
See also: MdlModelNew, MdlPatchTrimmingSegPointers,

6.2.118 MdlModelTSrfTCrvsBBox (mdl_bbox.c:116)

bbox

bounding box

```
CagdBBoxStruct *MdlModelTSrfTCrvsBBox(const MdlTrimSrfStruct *TSrf,
                                      CagdBBoxStruct *BBox)
```

TSrf: A trimmed surfaces to compute a bbox for its trimming curves.

BBox: Where bounding information is to be saved.

Returns: BBox.

Description: Computes a bounding box for the trimming curves of one trimmed srf.
See also: MdlModelListBBox, CagdCrvListBBox, CagdSrfBBox, GMBBSetBBoxPrecise,

6.2.119 MdlModelTransform (mdl_gen.c:1413)

models

```
void MdlModelTransform(MdlModelStruct *Model,
                       const CagdRType *Translate,
                       CagdRType Scale)
```

Model: Model to transform.

Translate: Translation factor.

Scale: Scaling factor.

Returns: void

Description: Linearly transforms, in place, given model as specified by Translate and Scale.

6.2.120 MdlModelsSame (mdl_gen.c:1492)

```
CagdBType MdlModelsSame(const MdlModelStruct *Model1,
                        const MdlModelStruct *Model2,
                        CagdRType Eps)
```

Model1, Model2: The two trivariates to compare.

Eps: Tolerance of equality.

Returns: TRUE if trivariates are the same, FALSE otherwise.

Description: Compare the two models for similarity.
See also: CagdSrfsSame, CagdCrvsSame, MvarMVsSame,

6.2.121 MdlNewLoop (mdl_gen.c:1073)

Allocation

```
MdlLoopStruct *MdlNewLoop(CagdCrvStruct *LoopCrvs)
```

LoopCrvs: The given list of curves, used in place.

Returns: The created Loop.

Description: Creates a new loop from a list of curves forming a loop.
See also: MdlModelNew, MdlTrimSrfNew2,

6.2.122 MdlPatchTrimmingSegPointers (mdl_ptch.c:39)

```
void MdlPatchTrimmingSegPointers(MdlModelStruct *Model)
```

Model: To back patch its pointer.

Returns: void

Description: Patches a model to have proper (back) pointers between the different data structures. The assumtions are that all reference and back pointers are set as indices into the proper lists:

1. Every reference to a surface, is given as its index in the Model's TrimSrfList trimmed surfaces' list.
2. Every reference to a trimming segment, is given as its index in the Model's TrimSegList or trimming segments' list. All lists are indexed starting from 1, 0 denotes an error.

See also: MdlModelNew, MdlModelNew2, MdlReadModelFromFile,

6.2.123 MdlPrimBox (mdl_prim.c:992)

```
MdlModelStruct *MdlPrimBox(CagdRType MinX,
                           CagdRType MinY,
                           CagdRType MinZ,
                           CagdRType MaxX,
                           CagdRType MaxY,
                           CagdRType MaxZ)
```

MinX, MinY, MinZ: Minimum range of box model.

MaxX, MaxY, MaxZ: Maximum range of box model.

Returns: Constructed box model, as a set of bilinear surfaces.

Description: A model constructor of a box, parallel to main axes.
See also: agdPrimBoxSrf, ,

6.2.124 MdlPrimCone (mdl_prim.c:1164)

```
MdlModelStruct *MdlPrimCone(const CagdVType Center,
                            CagdRType Radius,
                            CagdRType Height,
                            CagdBType Rational,
                            CagdPrimCapsType Caps)
```

Center: Of constructed cone (center of its base).

Radius: Of constructed cone's base.

Height: Of constructed cone.

Rational: If TRUE exact ration cone is created. If FALSE an approximated integral cone is created.

Caps: Do we want caps (top and/or bottom) for the cone?

Returns: Constructed cone model.

Description: A model constructor of a cone, centered at Center, radii of Radius, and height of Height. Axis of cone is Z axis.
See also: CagdPrimConeSrf, MdlStitchSelfSrfPrims,

6.2.125 MdlPrimCone2 (mdl_prim.c:1132)

```
MdlModelStruct *MdlPrimCone2(const CagdVType Center,
                             CagdRType MajorRadius,
                             CagdRType MinorRadius,
                             CagdRType Height,
                             CagdBType Rational,
                             CagdPrimCapsType Caps)
```

Center: Of constructed cone (center of its base).

MajorRadius: Of constructed cone.

MinorRadius: Of constructed cone.

Height: Of constructed cone.

Rational: If TRUE exact ration cone is created. If FALSE an approximated integral cone is created.

Caps: Do we want caps (top and/or bottom) for the cone?

Returns: Constructed truncated cone model.

Description: A model constructor of a truncated cone, centered at Center and radii of MajorRadius and MinorRadius. A MinorRadius of zero would construct a regular cone. Otherwise, a truncated cone. Axis of cone is Z axis.
See also: CagdPrimCone2Srf, MdlStitchSelfSrfPrims,

6.2.126 MdlPrimCylinder (mdl_prim.c:1195)

```
MdlModelStruct *MdlPrimCylinder(const CagdVType Center,
                                CagdRType Radius,
                                CagdRType Height,
                                CagdBType Rational,
                                CagdPrimCapsType Caps)
```

Center: Of constructed Cylinder (center of its base).

Radius: Of constructed Cylinder.

Height: Of constructed Cylinder.

Rational: If TRUE exact ration sphere is created. If FALSE an approximated integral surface is created.

Caps: Do we want caps (top and/or bottom) for the cone?

Returns: Constructed cylinder model.

Description: A model constructor of a Cylinder, centered at Center, radii of Radius, and height of Height. Axis of cylinder is Z axis.
See also: CagdPrimCylinderSrf, MdlStitchSelfSrfPrims,

6.2.127 MdlPrimListOfSrfs2Model (mdl_prim.c:617)

```
MdlModelStruct *MdlPrimListOfSrfs2Model(CagdSrfStruct *Srfs,
                                        int *n,
                                        int StitchModel)
```

Srfs: List of surfaces to convert to one model, in place. If Srfs contains one surface only, it is tested for being closed in either U or V and divided into two accordingly, before forming the model.

n: Optional parameter to be set with number of boundary edges that were stitched. Can be NULL to be ignored.

StitchModel: TRUE to also aim to stitch adjacent surfaces, in the end.

Returns: Constructed model.

Description: A model constructor from a list of surfaces.
See also: MdlStitchSelfSrfPrims, MdlStitchSrfsInModel2,

6.2.128 MdlPrimListOfSrfs2Model2 (mdl_prim.c:897)

```
MdlModelStruct *MdlPrimListOfSrfs2Model2(const CagdSrfStruct *InSrfs,
                                         CagdCrvAdjCmpFuncType CrvCmpFuncPtr,
                                         int *Is2ManifoldP,
                                         int *StitchedCrvsP)
```

InSrfs: A list of surface.

CrvCmpFuncPtr: A pointer to the function to compare two curves. Can be NULL for a default comparison function.

Is2ManifoldP: A pointer to integer, will be set to 1/2 if the model is open/closed, respectively. It can be NULL to be ignored.

StitchedCrvsP: A pointer to integer, will be set with the number of boundary curves that were stitched. -1 for error. It can be NULL to be ignored.

Returns: Constructed Model.

Description: The function takes a list of surface and build a single Model from these surfaces. If the surfaces have a non 2-manifold topology, the function returns NULL. This function is similar to MdlPrimListOfSrfs2Model, only more efficient.
See also: CagdSrfsAddAdjAttributes, MdlPrimListOfSrfs2Model,

6.2.129 MdlPrimPlane (mdl_prim.c:546)

```
MdlModelStruct *MdlPrimPlane(CagdRType MinX,
                             CagdRType MinY,
                             CagdRType MaxX,
                             CagdRType MaxY,
                             CagdRType ZLevel)
```

MinX, MinY: Minimum XY coordinates of plane.

MaxX, MaxY: Maximum XY coordinates of plane.

ZLevel: Z level of plane, parallel to the XY plane.

Returns: Constructed plane model, as a bilinear surface.

Description: A model constructor of a plane, parallel to XY plane at level Zlevel.
See also: CagdPrimPlaneSrf, MdlPrimPlaneSrfOrderLen,

6.2.130 MdlPrimPlaneSrfOrderLen (mdl_prim.c:582)

```
MdlModelStruct *MdlPrimPlaneSrfOrderLen(CagdRType MinX,
                                        CagdRType MinY,
                                        CagdRType MaxX,
                                        CagdRType MaxY,
                                        CagdRType ZLevel,
                                        int Order,
                                        int Len)
```

MinX, MinY: Minimum XY coordinates of plane.

MaxX, MaxY: Maximum XY coordinates of plane.

ZLevel: Z level of plane, parallel to the XY plane.

Order: Order of plane surface that is requested.

Len: Number of control points (via refinement).

Returns: Constructed plane model, as a bi-Order surface.

Description: A model constructor of a plane, parallel to XY plane at level Zlevel.
See also: agdPrimPlaneSrfOrderLen, MdlPrimPlane,

6.2.131 MdlPrimSphere (mdl_prim.c:1037)

```
MdlModelStruct *MdlPrimSphere(const CagdVType Center,
                              CagdRType Radius,
                              CagdBType Rational)
```

Center: of constructed sphere.

Radius: of constructed sphere.

Rational: If TRUE exact ration sphere is created. If FALSE an approximated integral surface is created.

Returns: Constructed sphere model.

Description: A model constructor of a sphere, centered at Center and radius Radius.
See also: CagdPrimSphereSrf, MdlStitchSelfSrfPrims, MdlCreateCubeSpherePrim,

6.2.132 MdlPrimTorus (mdl_prim.c:1089)

```
MdlModelStruct *MdlPrimTorus(const CagdVType Center,
                             CagdRType MajorRadius,
                             CagdRType MinorRadius,
                             CagdBType Rational)
```

Center: Of constructed torus.

MajorRadius: Of constructed torus.

MinorRadius: Of constructed torus.

Rational: If TRUE exact ration sphere is created. If FALSE an approximated integral surface is created.

Returns: Constructed torus model.

Description: A model constructor of a torus, centered at Center and radii of MajorRadius and MinorRadius.
See also: CagdPrimTorusSrf, MdlStitchSelfSrfPrims,

6.2.133 MdlPrintLoop (mdl_dbg.c:1787)

```
void MdlPrintLoop(MdlLoopStruct *Loop, const char *FileName)
```

Loop: The trimming loop

FileName: Output file name.

Returns: void

Description: Saves a given trimming loop in R3 to a specified file.

6.2.134 MdlRemovEucTrimCrvs (mdlcnvrt.c:890)

```
void MdlRemovEucTrimCrvs(MdlModelStruct *Mdl)
```

Mdl: A model to remove all Euclidean trimming curves from.

Returns: void

Description: Remove the Euclidean curves from the trimming curves in the given data.
See also: TrimRemovEucTrimCrvs,

6.2.135 MdlSplitDisjointComponents (mdl_cc.c:172)

```
MdlModelStruct *MdlSplitDisjointComponents(const MdlModelStruct *Mdl)
```

Mdl: The model.

Returns: List of models, each model contains a component.

Description: Splits a given model into disconnected components, each component as a new model.

6.2.136 MdlSplitTrimCrv (mdl_aux.c:189)

allocation

```
int MdlSplitTrimCrv(MdlTrimSegStruct *Seg,
                    const CagdPtStruct *Pts,
                    int Idx,
                    CagdRType Eps,
                    int *Proximity)
```

Seg: Trimming segment to split, in place.

Pts: Parameters at which to split.

Idx: Index of parameter in Pts points: 0 for X, 1 for Y, etc.

Eps: parameter closer than Eps to boundary or other parameters are ignored.

Proximity: Proximity bit set to end points - see CagdCrvSubdivAtParams2.

Returns: TRUE if successful, FALSE otherwise.

Description: Subdivides the given segment at the specified parameter values. This amounts to:

1. Dividing all curves in Seg and chaining the pieces after Seg.
2. Updating the references in SrfFirst and SrfNext that points to Seg, to now point to all new pieces, in the right order, as can be reversed.

6.2.137 MdlStitchModel (mdl_prim.c:416)

```
int MdlStitchModel(MdlModelStruct *Mdl,
                   CagdBType BackProjTest,
                   CagdRType StitchTol)
```

Mdl: Model to seek trimming curves to stitch together, in place.

BackProjTest: If TRUE, then use back-projection to match the curve segments. If FALSE, compare the middle points of the curves to stitch.

StitchTol: Tolerance to use in stitching two trimmed curves as one.

Returns: Number of trimming curves stitched together, in place.

Description: Scans the given model for trimming curves that could be stitched together. A pair of trimming curves can be stitched together if they have the same Euclidean representation and they now have no neighbors. Two trimming curves are considered with "same Euclidean representation" if their end points are the same upto given tolerance (should probably do somewhat better here).

See also: MdlBooleanMerge, VMdlStitchMdlModel,

6.2.138 MdlStitchSelfSrfPrims (mdl_prim.c:1329)

```
int MdlStitchSelfSrfPrims(int Stitch)
```

Stitch: TRUE to stitch different boundaries of same surface.

Returns: Old value setting.

Description: Set if different boundaries on the same primitive surface are to be stitched as well or not.

6.2.139 MdlTrimSegCopy (mdl_gen.c:269)

allocation

```
MdlTrimSegStruct *MdlTrimSegCopy(const MdlTrimSegStruct *MdlTrimSeg,
                                 const MdlTrimSrfStruct *TrimSrfList)
```

MdlTrimSeg: A trimming segment to duplicate.

TrimSrfList: The original trimmed surfaces.

Returns: A trimming segment structure.

Description: Duplicates a trimming segments structure. The reference pointers to the (upto) two surfaces are replaced with the indices of the surfaces in TrimSrfList.

6.2.140 MdlTrimSegCopyList (mdl_gen.c:333)

copy

```
MdlTrimSegStruct *MdlTrimSegCopyList(const MdlTrimSegStruct *MdlTrimSegList,
                                     const MdlTrimSrfStruct *TrimSrfList)
```

MdlTrimSegList: To be copied.

TrimSrfList: The original trimmed surfaces.

Returns: A duplicated list of trimming segments.

Description: Allocates and copies a list of trimming segment structures. The reference pointers to the (upto) two surfaces are replaced with the indices of the surfaces in TrimSrfList.

6.2.141 MdlTrimSegFree (mdl_gen.c:42)

allocation

```
void MdlTrimSegFree(MdlTrimSegStruct *MTSeg)
```

MTSeg: A Trimming Segment to free.

Returns: void

Description: Deallocates a Model Trimming Segments structure.

6.2.142 MdlTrimSegFreeList (mdl_gen.c:64)

allocation

```
void MdlTrimSegFreeList(MdlTrimSegStruct *MTSegList)
```

MTSegList: A Trimming Segment List to free.

Returns: void

Description: Deallocates a Model Trimming Segments List structure.

6.2.143 MdlTrimSegNew (mdl_gen.c:671)

allocation

```
MdlTrimSegStruct *MdlTrimSegNew(CagdCrvStruct *UVCrv1,
                                CagdCrvStruct *UVCrv2,
                                CagdCrvStruct *EucCrv1,
                                MdlTrimSrfStruct *SrfFirst,
                                MdlTrimSrfStruct *SrfSecond)
```

UVCrv1: A UV curve for SrfFirst. Must be an E2 curve. Used in place.

UVCrv2: A UV curve for SrfSecond. Must be an E2 curve. Used in place. Can be NULL.

EucCrv1: Optional Euclidean curve for SrfFirst. Must be an E3 curve. Used in place. Can be NULL.

SrfFirst: First surface of the segment. Can be NULL.

SrfSecond: Second surface of the segment. Can be NULL.

Returns: A model trimming segment structure.

Description: Allocates a model trimming segment structure. Allows periodic and float end conditions - converts them to open end.

6.2.144 MdlTrimSegRefCopy (mdl_gen.c:387)

allocation

```
MdlTrimSegRefStruct *MdlTrimSegRefCopy(const MdlTrimSegRefStruct *MTSegRefList,
                                       const MdlTrimSegStruct *TrimSegList)
```

MTSegRefList: A trimming curve segment reference to duplicate.

TrimSegList: The original trimmed segments.

Returns: A trimming segment reference structure.

Description: Duplicates a trimming segment reference structure. The reference pointer to the trimming segment is replaced with the index of trimming segment in TrimSegList.

6.2.145 MdlTrimSegRefCopyList (mdl_gen.c:422)

copy

```
MdlTrimSegRefStruct *MdlTrimSegRefCopyList(const MdlTrimSegRefStruct
                                                              *MTSegRefList,
                                           const MdlTrimSegStruct *TrimSegList)
```

MTSegRefList: To be copied.

TrimSegList: The original trimmed segments.

Returns: A duplicated list of trimming segments.

Description: Allocates and copies a list of trimming segment reference structures. The reference pointer to the trimming segment is replaced with the index of trimming segment in TrimSegList.

6.2.146 MdlTrimSegRefFree (mdl_gen.c:88)

allocation

```
void MdlTrimSegRefFree(MdlTrimSegRefStruct *MTSegRef)
```

MTSegRef: A Segments Reference to free.

Returns: void

Description: Deallocates a Model Trimming Segment Reference structure.

6.2.147 MdlTrimSegRefFreeList (mdl_gen.c:107)

allocation

```
void MdlTrimSegRefFreeList(MdlTrimSegRefStruct *MTSegRefList)
```

MTSegRefList: A list of loops to free.

Returns: void

Description: Deallocates a Model Trimming Segment Reference List structure.

6.2.148 MdlTrimSegRefNew (mdl_gen.c:820)

allocation

```
MdlTrimSegRefStruct *MdlTrimSegRefNew(MdlTrimSegStruct *MdlTrimSeg)
```

MdlTrimSeg: List of model trimming segments forming the trimming curve.

Returns: A trimmig curve.

Description: Allocates a model trimming segment reference structure.

6.2.149 MdlTrimSegRefRemove (mdl_gen.c:852)

```
int MdlTrimSegRefRemove(const MdlTrimSegStruct *TSeg,
                        MdlTrimSegRefStruct **TSegRefList,
                        int FreeRef)
```

TSeg: Trimmed Segment to remove references to it from list.

TSegRefList: List to remove TSegRef From.

FreeRef: TRUE to also free this reference, if found.

Returns: TRUE if reference found and removed, FALSE otherwise.

Description: Remove TSegRef from TSegRefList. TSeg is not deleted.

6.2.150 MdlTrimSegRefRemove2 (mdl_gen.c:899)

```
int MdlTrimSegRefRemove2(const MdlTrimSegStruct *TSeg,
                         MdlLoopStruct *Loops,
                         int FreeRef)
```

TSeg: Trimmed Segment to remove references to it from list.

Loops: List of loops to find TSeg in and remove it.

FreeRef: TRUE to also free this reference, if found.

Returns: TRUE if reference found and removed, FALSE otherwise.

Description: Remove TSeg from Loops. TSeg is not deleted.

6.2.151 MdlTrimSegRemove (mdl_gen.c:755)

```
int MdlTrimSegRemove(const MdlTrimSegStruct *TSeg, MdlTrimSegStruct **TSegList)
```

TSeg: Trim segment to remove from list.

TSegList: List to remove TSeg from.

Returns: TRUE if removed, FALSE otherwise.

Description: Removes TSeg from SegList. TSeg is not freed.

6.2.152 MdlTrimSegRemove2 (mdl_gen.c:791)

```
int MdlTrimSegRemove2(MdlTrimSegStruct *TSeg, MdlModelStruct *Mdl)
```

TSeg: Trim segment to remove from Mdl.

Mdl: Model to remove TSegFrom.

Returns: TRUE if removed, FALSE otherwise.

Description: Removes TSeg from Mdl. TSeg is freed and all references to it are removed as well.

6.2.153 MdlTrimSrfChainTrimSegs (mdl_gen.c:1170)

allocation

```
MdlTrimSegStruct *MdlTrimSrfChainTrimSegs(MdlTrimSrfStruct *TSrfs)
```

TSrfs: The trimmed surfaces in the model. Used in place.

Returns: A pointer to a linked list of all trimming curve segmented in TSrfs, chained in place.

Description: Go over all trimmed surface in the model and chain all trimming curve segments into one large linked list, a pointer to which is returned. This function is typically invoked on a newly created model where the trimmed surfaces are already created but the TrimSegList of the model is not derived yet.

6.2.154 MdlTrimSrfCopy (mdl_gen.c:513)

allocation

```
MdlTrimSrfStruct *MdlTrimSrfCopy(const MdlTrimSrfStruct *MdlTrimSrf,
                                 const MdlTrimSegStruct *TrimSegList)
```

MdlTrimSrf: A trimming surface to duplicate.

TrimSegList: The original trimmed segments.

Returns: A trimming surface structure.

Description: Duplicates a trimming surface structure.

6.2.155 MdlTrimSrfCopyList (mdl_gen.c:543)

copy

```
MdlTrimSrfStruct *MdlTrimSrfCopyList(const MdlTrimSrfStruct *MdlTrimSrfList,
                                     const MdlTrimSegStruct *TrimSegList)
```

MdlTrimSrfList: To be copied.

TrimSegList: The original trimmed segments.

Returns: A duplicated list of trimming surfaces.

Description: Allocates and copies a list of trimming surface structures.

6.2.156 MdlTrimSrfFree (mdl_gen.c:175)

allocation

```
void MdlTrimSrfFree(MdlTrimSrfStruct *TrimSrf)
```

TrimSrf: A surface to free.

Returns: void

Description: Deallocates a Model Trimming Surface structure.

6.2.157 MdlTrimSrfFreeList (mdl_gen.c:196)

allocation

```
void MdlTrimSrfFreeList(MdlTrimSrfStruct *MdlTrimSrfList)
```

MdlTrimSrfList: A list of trimming curve to free.

Returns: void

Description: Deallocates a Model Trimming Surface List structure.

6.2.158 MdlTrimSrfNew (mdl_gen.c:954)

allocation

```
MdlTrimSrfStruct *MdlTrimSrfNew(CagdSrfStruct *Srf,
                                MdlLoopStruct *LoopList,
                                CagdBType HasTopLvlTrim,
                                CagdBType UpdateBackTSrfPtrs)
```

Srf: Surface to make into a trimmed surface. In place.

LoopList: An optional list of loops. Used in place.

HasTopLvlTrim: Do we have a top level outer most trimming curve?

UpdateBackTSrfPtrs: TRUE to also update back pointers from trimming curves to the trimmed surface.

Returns: The trimmed surface.

Description: Constructor for a model trimmed surface.

6.2.159 MdlTrimSrfNew2 (mdl_gen.c:1131)

allocation

```
MdlTrimSrfStruct *MdlTrimSrfNew2(CagdSrfStruct *Srf,
                                 CagdCrvStruct **LoopList,
                                 int NumLoops,
                                 CagdBType HasTopLvlTrim)
```

Srf: The original surface to be trimmed. Used in place.

LoopList: An array of trimming loops. Used in place.

NumLoops: The number of the loops (LoopList array length)

HasTopLvlTrim: If FALSE, add outer loops boundary.

Returns: The trimmed surface.

Description: Constructor for a model trimmed surface.

6.2.160 MdlTwoTrimSegsSameEndPts (mdl_prim.c:54)

```
int MdlTwoTrimSegsSameEndPts(const MdlTrimSegStruct *TSeg1,
                             const MdlTrimSegStruct *TSeg2,
                             CagdBType BackProjTest,
                             CagdRType Tol)
```

TSeg1, TSeg2: The two segments to compare.

BackProjTest: If TRUE, then use back-projection to match the curve segments. If FALSE, compare the middle points of the curves.

Tol: Tolerance of comparison of end points.

Returns: 0 if not the same Euclidean representation, 1 if same and start and end points match, -1 if same but reversed, -9 if singular.

Description: Compare two given trimming curve segments for similarity in Euclidean space. Comparison is made at the end pts and then mid pt or back proj.

Chapter 7

Miscelleneous Library, misc_lib

7.1 General Information

This library holds general miscelleneous functions such as reading configuration files, low level homogeneous matrices computation, low level attributes and general machine specific low level routines. Several header files can be found for this library:

Header (include/*.h)	Functionality
config.h	Manipulation of configuration files (*.cfg files)
dist_pts.h	Enegry based distribution of points.
gen_mat.h	Homogeneous matrices manipulation
getarg.h	Command line parsing for application tools
imalloc.h	Low level dynamic memory functions for IRIT
miscattr.h	Low level attribute related functions
priorque.h	An implementation of a priority queue
xgeneral.h	Low level, machine specific, routines

7.2 Library Functions

7.2.1 Attr2StringToData (miscattr.c:1343)

attributes

```
char *Attr2StringToData(const IPAttributeStruct *Attr, int DataFileFormat)
```

Attr: To convert to a string.

DataFileFormat: If TRUE, the attribute is formated in the IRIT data file format. Otherwise, just the attribute value is returned as a string.

Returns: A pointer to a string representing/describing the given attribute. Allocated dynamically.

Description: Routine to convert an attribute to a string.

7.2.2 AttrCmpTwoAttrByName (miscattr.c:1505)

attributes

```
int AttrCmpTwoAttrByName(const IPAttributeStruct *AttrList1,
                         const IPAttributeStruct *AttrList2,
                         const char *AttrName)
```

AttrList1: First attribute list to took at.

AttrList2: Second attribute list to took at.

AttrName: Attribute name to look for and compare.

Returns: -2 if no attribute was found in AttrList1. 2 if no attribute was found in AttrList2. -1, 0, 1 if attribute in AttrList1 is smaller/similar/greater than in AttrList2. 9 is returned if the found attributes are of different types.

Description: Routine to compare if both attribute list contain attribute named AttrName and same value in these attributes.

7.2.3 AttrCopyAttributes (miscattr.c:1888)

attributes

```
IPAttributeStruct *AttrCopyAttributes(const IPAttributeStruct *Src)
```

Src: Attribute list to duplicate.

Returns: Duplicated attribute list.

Description: Routine to copy an attribute list. Attributes prefixed with '_' are considered internal and are not copied.
See also: AttrCopyAttributes2, AttrCopyOneAttribute,

7.2.4 AttrCopyAttributes2 (miscattr.c:1912)

attributes

```
IPAttributeStruct *AttrCopyAttributes2(const IPAttributeStruct *Src,
                                       int AllAttrs)
```

Src: Attribute list to duplicate.

AllAttrs: If FALSE, attributes prefixed with '_' are not copied. If TRUE, all attributes are copied.

Returns: Duplicated attribute list.

Description: Routine to copy an attribute list.
See also: AttrCopyAttributes, AttrCopyOneAttribute2,

7.2.5 AttrCopyOneAttribute (miscatt3.c:35)

attributes

```
IPAttributeStruct *AttrCopyOneAttribute(const IPAttributeStruct *Src)
```

Src: Attribute to duplicate.

Returns: Duplicated attribute.

Description: Routine to copy one attribute. This routine also exists in attribut.c with object handling. It will be linked in iff no object handling is used. Attributes prefixed with '_' are considered internal and are not copied.
See also: AttrCopyAttributes2, AttrCopyOneAttribute2,

7.2.6 AttrCopyOneAttribute2 (miscatt3.c:61)

attributes

```
IPAttributeStruct *AttrCopyOneAttribute2(const IPAttributeStruct *Src,
                                         int AllAttr)
```

Src: Attribute to duplicate.

AllAttr: If FALSE, attributes prefixed with '_' are not copied. If TRUE, all attributes are copied.

Returns: Duplicated attribute.

Description: Routine to copy one attribute. This routine also exists in attribut.c with object handling. It will be linked in iff no object handling is used.
See also: AttrCopyAttributes2, AttrCopyOneAttribute,

7.2.7 AttrCopyValidAttrNameList (miscattr.c:1830)

attributes

```
const char **AttrCopyValidAttrNameList(const char **AttrNames)
```

AttrNames: Vector of attribute names to use in attr list copy.

Returns: Old list of valid attributes to copy.

Description: Sets the NULL terminated list of attribute's names to copy.

7.2.8 AttrFindAttribute (miscattr.c:1605)

attributes

```
IPAttributeStruct *AttrFindAttribute(const IPAttributeStruct *Attrs,
                                     const char *Name)
```

Attrs: Attribute list to search.

Name: Attribute to search by this name.

Returns: Attribute if found, otherwise NULL.

Description: Routine to search for an attribute by Name.
See also: AttrIDFindAttribute,

7.2.9 AttrFindAttributeHashNum (miscatt1.c:188)

```
IPAttributeStruct *AttrFindAttributeHashNum(const IPAttributeStruct *Attrs,
                                            IPAttrNumType AttrHashNum)
```

Attrs: Attribute list to search.

AttrHashNum: Attribute numeric index to search by this number.

Returns: Attribute if found, otherwise NULL.

Description: Routine to search for an attribute by hashed numeric index.

7.2.10 AttrFreeAttributes (miscattr.c:1792)

attributes

```
void AttrFreeAttributes(IPAttributeStruct **Attrs)
```

Attrs: To remove and delete.

Returns: void

Description: Routine to remove and delete all attributes of the given Attr list.

7.2.11 AttrFreeOneAttribute (miscattr.c:1742)

attributes

```
void AttrFreeOneAttribute(IPAttributeStruct **Attrs, const char *Name)
```

Attrs: To search for an attribute named Name and remove and delete it.

Name: Name of attribute to remove and delete.

Returns: void

Description: Routine to remove and delete the attribute named Name from the given Attr list.

7.2.12 AttrGetAttribHashNumber (miscatt1.c:132)

```
IPAttrNumType AttrGetAttribHashNumber(const char *AttribName)
```

AttribName: The Attribute name to seek its hashed number (or create if not found.)

Returns: The hashed attribute number.

Description: This function returns the matching attribute hash number to the specified attribute name. If the attribute name doesn't exist it is created.

7.2.13 AttrGetAttribName (miscatt1.c:54)

```
const char *AttrGetAttribName(const IPAttributeStruct *Attr)
```

Attr: The attribute structure.

Returns: The name of the attribute, "_undefined_" if don't exist.

Description: This function returns the name of the attribute.

7.2.14 AttrGetIndexColor (miscattr.c:75)

attributes

color

rgb

```
void AttrGetIndexColor(int Color, int *Red, int *Green, int *Blue)
```

Color: Index of color between 0 and 15.

Red, Green, Blue: Component of RGB color.

Returns: void

Description: Routine to fetch one of 16 basic colors based on its index.
See also: AttrSetIntAttrib, AttrSetObjectRGBColor,

7.2.15 AttrGetIntAttrib (miscattr.c:325)

attributes

```
int AttrGetIntAttrib(const IPAttributeStruct *Attrs, const char *Name)
```

Attrs: Attribute list to search for requested attribute.

Name: Name of requested attribute.

Returns: Found attribute, or IP_ATTR_BAD_INT if not found.

Description: Routine to get an integer attribute.
See also: AttrSetIntAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, AttrSetRealAttrib, , AttrSetRealPtrAttrib, AttrGetRealAttrib, AttrGetRealPtrAttrib, , AttrSetStrAttrib, AttrGetStrAttrib, AttrGetIntAttrib2,

7.2.16 AttrGetIntAttrib2 (miscattr.c:355)

attributes

```
int AttrGetIntAttrib2(const IPAttributeStruct *Attrs, IPAttrNumType AttrNum)
```

Attrs: Attribute list to search for requested attribute.

AttrNum: Unique ID derived from name of requested attribute.

Returns: Found attribute, or IP_ATTR_BAD_INT if not found.

Description: Routine to get an integer attribute.
See also: AttrSetIntAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, AttrSetRealAttrib, , AttrSetRealPtrAttrib, AttrGetRealAttrib, AttrGetRealPtrAttrib, , AttrSetStrAttrib, AttrGetStrAttrib, AttrGetIntAttrib,

7.2.17 AttrGetLastAttr (miscattr.c:2069)

linked lists

last element

```
IPAttributeStruct *AttrGetLastAttr(IPAttributeStruct *AList)
```

AList: A list of attributes.

Returns: Last attribute in AList.

Description: Returns a pointer to last attribute of a list.

7.2.18 AttrGetPrevAttr (miscattr.c:2090)

previous element

linked lists

```
IPAttributeStruct *AttrGetPrevAttr(IPAttributeStruct *AList,
                                   const IPAttributeStruct *A)
```

AList: A list of attributes.

A: For which the previous attribute in AList is pursuit.

Returns: Previous attribute to A in AList if found, NULL otherwise.

Description: Returns a pointer to previous attribute in AList to A.

7.2.19 AttrGetPtrAttrib (miscattr.c:472)

attributes

```
VoidPtr AttrGetPtrAttrib(const IPAttributeStruct *Attrs, const char *Name)
```

Attrs: Attribute list to search for requested attribute.

Name: Name of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a pointer attribute.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrSetPtrAttrib, AttrSetRealAttrib, , AttrSetRealPtrAttrib, AttrGetRealAttrib, AttrGetRealPtrAttrib, , AttrSetStrAttrib, AttrGetStrAttrib, AttrGetUVAttrib, , AttrSetUVAttrib, AttrSetRefPtrAttrib, AttrGetRefPtrAttrib, , AttrGetPtrAttrib2,

7.2.20 AttrGetPtrAttrib2 (miscattr.c:505)

attributes

```
VoidPtr AttrGetPtrAttrib2(const IPAttributeStruct *Attrs,
                          IPAttrNumType AttrNum)
```

Attrs: Attribute list to search for requested attribute.

AttrNum: Unique ID derived from name of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a pointer attribute.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrSetPtrAttrib, AttrSetRealAttrib, , AttrSetRealPtrAttrib, AttrGetRealAttrib, AttrGetRealPtrAttrib, , AttrSetStrAttrib, AttrGetStrAttrib, AttrGetUVAttrib, , AttrSetUVAttrib, AttrSetRefPtrAttrib, AttrGetRefPtrAttrib, , AttrGetPtrAttrib2,

7.2.21 AttrGetRGBColor2 (miscattr.c:109)

attributes

color

rgb

```
int AttrGetRGBColor2(const IPAttributeStruct *Attrs,
                     const char *Name,
                     int *Red,
                     int *Green,
                     int *Blue)
```

Attrs: For which we would like to know the RGB of.

Name: Name of the attribute, if NULL default is taken.

Red, Green, Blue: Component of RGB color to initialize.

Returns: TRUE if does have any color attribute, FALSE otherwise.

Description: Routine to return a RGB attribute or COLOR attribute converted to RGB. Beside, it can be used to get other color attributes: specular, etc.
See also: AttrSetIntAttrib, AttrGetObjectRGBColor, , AttrSetRGBDoubleColor, AttrGetRGBDoubleColor,

7.2.22 AttrGetRGBDoubleColor (miscattr.c:209)

attributes

color

rgb

```
int AttrGetRGBDoubleColor(const IPAttributeStruct *Attrs,
                          double *Red,
                          double *Green,
                          double *Blue)
```

Attrs: For which we would like to know the RGB of.

Red, Green, Blue: Component of RGB color to initialize.

Returns: TRUE if does have an RGB color attribute, FALSE otherwise.

Description: Routine to return a RGB attribute.
See also: AttrSetIntAttrib, AttrGetObjectRGBColor, AttrSetRGBDoubleColor,

7.2.23 AttrGetRealAttrib (miscattr.c:759)

attributes

```
IrtRType AttrGetRealAttrib(const IPAttributeStruct *Attrs, const char *Name)
```

Attrs: Attribute list to search for requested attribute.

Name: Name of requested attribute.

Returns: Found attribute, or IP_ATTR_BAD_REAL if not found.

Description: Routine to get a IrtRType attribute.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, , AttrSetRealAttrib, AttrSetRealPtrAttrib, AttrSetStrAttrib, , AttrGetStrAttrib, AttrGetUVAttrib, AttrSetUVAttrib, AttrGetRealAttrib2, , AttrGetRealPtrAttrib,

7.2.24 AttrGetRealAttrib2 (miscattr.c:791)

attributes

```
IrtRType AttrGetRealAttrib2(const IPAttributeStruct *Attrs,
                            IPAttrNumType AttrNum)
```

Attrs: Attribute list to search for requested attribute.

AttrNum: Unique ID derived from name of requested attribute.

Returns: Found attribute, or IP_ATTR_BAD_REAL if not found.

Description: Routine to get a IrtRType attribute.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, , AttrSetRealAttrib, AttrSetRealPtrAttrib, AttrSetStrAttrib, , AttrGetStrAttrib, AttrGetUVAttrib, AttrSetUVAttrib, AttrGetRealAttrib, AttrGetRealPtrAttrib,

7.2.25 AttrGetRealPtrAttrib (miscattr.c:928)

attributes

```
IrtRType *AttrGetRealPtrAttrib(const IPAttributeStruct *Attrs,
                               const char *Name)
```

Attrs: Attribute list to search for requested attribute.

Name: Name of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a IrtRType * attribute.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, , AttrSetRealAttrib, AttrSetRealPtrAttrib, AttrSetStrAttrib, , AttrGetStrAttrib, AttrGetUVAttrib, AttrSetUVAttrib, AttrGetRealAttrib2, , AttrGetRealPtrAttrib,

7.2.26 AttrGetRealPtrAttrib2 (miscattr.c:960)

attributes

```
IrtRType *AttrGetRealPtrAttrib2(const IPAttributeStruct *Attrs,
                                IPAttrNumType AttrNum)
```

Attrs: Attribute list to search for requested attribute.

AttrNum: Unique ID derived from name of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a IrtRType * attribute.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, , AttrSetRealAttrib, AttrSetRealPtrAttrib, AttrSetStrAttrib, , AttrGetStrAttrib, AttrGetUVAttrib, AttrSetUVAttrib, AttrGetRealAttrib, AttrGetRealPtrAttrib,

7.2.27 AttrGetRefPtrAttrib (miscattr.c:617)

attributes

```
VoidPtr AttrGetRefPtrAttrib(const IPAttributeStruct *Attrs, const char *Name)
```

Attrs: Attribute list to search for requested attribute.

Name: Name of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a pointer reference attribute.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrSetPtrAttrib, AttrSetRealAttrib, , AttrSetRealPtrAttrib, AttrGetRealAttrib, AttrGetRealPtrAttrib, , AttrSetStrAttrib, AttrGetStrAttrib, AttrGetUVAttrib, , AttrSetUVAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, AttrGetRefPtrAttrib2,

7.2.28 AttrGetRefPtrAttrib2 (miscattr.c:650)

attributes

```
VoidPtr AttrGetRefPtrAttrib2(const IPAttributeStruct *Attrs,
                             IPAttrNumType AttrNum)
```

Attrs: Attribute list to search for requested attribute.

AttrNum: Unique ID derived from name of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a pointer reference attribute.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrSetPtrAttrib, AttrSetRealAttrib, , AttrSetRealPtrAttrib, AttrGetRealAttrib, AttrGetRealPtrAttrib, , AttrSetStrAttrib, AttrGetStrAttrib, AttrGetUVAttrib, , AttrSetUVAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, AttrGetRefPtrAttrib,

7.2.29 AttrGetStrAttrib (miscattr.c:1224)

attributes

```
const char *AttrGetStrAttrib(const IPAttributeStruct *Attrs, const char *Name)
```

Attrs: Attribute list to search for requested attribute.

Name: Name of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a string attribute.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, , AttrSetRealAttrib, AttrGetRealAttrib, AttrSetRealPtrAttrib, , AttrSetStrAttrib, AttrGetUVAttrib, AttrSetUVAttrib, AttrGetStrAttrib2,

7.2.30 AttrGetStrAttrib2 (miscattr.c:1255)

attributes

```
const char *AttrGetStrAttrib2(const IPAttributeStruct *Attrs,
                              IPAttrNumType AttrNum)
```

Attrs: Attribute list to search for requested attribute.

AttrNum: Unique ID derived from name of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a string attribute.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, , AttrSetRealAttrib, AttrGetRealAttrib, AttrSetRealPtrAttrib, , AttrSetStrAttrib, AttrGetUVAttrib, AttrSetUVAttrib, AttrGetStrAttrib,

7.2.31 AttrGetUVAttrib (miscattr.c:1072)

attributes

```
float *AttrGetUVAttrib(const IPAttributeStruct *Attrs, const char *Name)
```

Attrs: Attribute list to search for requested attribute.

Name: Name of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a UV attribute.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, , AttrSetRealAttrib, AttrSetRealPtrAttrib, AttrSetStrAttrib, , AttrSetUVAttrib, AttrGetUVAttrib2, AttrGetStrAttrib, AttrGetUVAttrib,

7.2.32 AttrGetUVAttrib2 (miscattr.c:1102)

attributes

```
float *AttrGetUVAttrib2(const IPAttributeStruct *Attrs, IPAttrNumType AttrNum)
```

Attrs: Attribute list to search for requested attribute.

AttrNum: Unique ID derived from name of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a UV attribute.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, , AttrSetRealAttrib, AttrGetRealPtrAttrib, AttrSetStrAttrib, , AttrGetStrAttrib, AttrGetUVAttrib, AttrSetUVAttrib, AttrGetUVAttrib,

7.2.33 AttrIDCopyValidAttrIDList (miscattr.c:1857)

attributes

```
const IPAttrIDType *AttrIDCopyValidAttrIDList(const IPAttrIDType *AttrIDs,
                                              int *IsValidList)
```

AttrIDs: Vector of attribute IDs to use in attr list copy.

IsValidList: TRUE if AttrIDs are the list of valid attribute to copy, FALSE, if the list contains invalid attrs to purge. Updated in place with the old IsValidList value.

Returns: Old list of valid attribute IDs to copy.

Description: Sets the NULL terminated list of attribute's names to copy.

7.2.34 AttrIDFindAttribute (miscatt1.c:280)

```
IPAttributeStruct *AttrIDFindAttribute(const IPAttributeStruct *Attrs,
                                       IPAttrIDType AttrID)
```

Attrs: Attribute list to search.

AttrID: Attribute ID to search by this number.

Returns: Attribute if found, otherwise NULL.

Description: Routine to search for an attribute by ID numeric index.
See also: AttrFindAttribute,

7.2.35 AttrIDFreeOneAttribute (misc_attr_ids.c:1016)

attributes

```
void AttrIDFreeOneAttribute(IPAttributeStruct **Attrs, IPAttrIDType AttrID)
```

Attrs: To search for an attribute named Name and remove and delete it.

AttrID: ID of attribute to remove and delete.

Returns: void

Description: Routine to remove and delete the attribute named Name from the given Attr list.
See also: AttrIDRemoveOneAttribute,

7.2.36 AttrIDGetAttribID (miscatt1.c:235)

```
IPAttrIDType AttrIDGetAttribID(const char *AttribName)
```

AttribName: The Attribute name of the given AttribName or IRIT_ATTR_CREATE_ID(INVALID_ATTR) if error.

Returns: The attribute ID number, or INVALID if not found.

Description: This function returns the matching attribute ID to the specified attribute name. Does a slow traversal over all string attributes but is rarely called (only when saving files, etc.).

7.2.37 AttrIDGetColor (misc_attr_ids.c:80)

attributes

color

```
int AttrIDGetColor(const IPAttributeStruct *Attrs)
```

Attrs: For which we would like to know the color of.

Returns: Color or IP_ATTR_NO_COLOR if no color set.

Description: Routine to return a color attribute ID.
See also: AttrIDSetColor, AttrIDSetRGBColor, AttrIDGetRGBColor, AttrIDSetWidth, , AttrIDGetWidth, AttrIDGetIntAttrib, AttrIDGetObjectColor,

7.2.38 AttrIDGetIntAttrib (misc_attr_ids.c:471)

attributes

```
int AttrIDGetIntAttrib(const IPAttributeStruct *Attrs,
                       IPAttrIDType AttribID)
```

Attrs: Attribute list to search for requested attribute.

AttribID: Attribute unique ID of requested attribute.

Returns: Found attribute, or IP_ATTR_BAD_INT if not found.

Description: Routine to get an integer attribute ID.
See also: AttrIDSetIntAttrib, AttrIDSetPtrAttrib, AttrIDGetPtrAttrib, , AttrIDSetRealPtrAttrib, AttrIDGetRealAttrib, AttIDrGetRealPtrAttrib, , AttrIDSetStrAttrib, AttrIDGetStrAttrib, AttrIDGetIntAttrib, , AttrIDSetRealAttrib,

7.2.39 AttrIDGetPtrAttrib (misc_attr_ids.c:555)

attributes

```
VoidPtr AttrIDGetPtrAttrib(const IPAttributeStruct *Attrs,
                           IPAttrIDType AttribID)
```

Attrs: Attribute list to search for requested attribute.

AttribID: Attribute unique ID of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a pointer attribute.
See also: AttrIDSetIntAttrib, AttrIDGetIntAttrib, AttrIDSetPtrAttrib, , AttrIDSetRealPtrAttrib, AttrIDGetRealAttrib, AttrIDGetRealPtrAttrib, AttrIDSetStrAttrib, AttrIDGetStrAttrib, AttrIDGetUVAttrib, , AttrIDSetUVAttrib, AttrIDSetRefPtrAttrib, AttrIDGetRefPtrAttrib, , AttrIDSetRealAttrib, AttrIDGetPtrAttrib2,

7.2.40 AttrIDGetRGBColor (misc_attr_ids.c:162)

attributes

color

rgb

```
int AttrIDGetRGBColor(const IPAttributeStruct *Attrs,
                      int *Red,
                      int *Green,
                      int *Blue)
```

Attrs: For which we would like to know the RGB of.

Red, Green, Blue: Component of RGB color to initialize.

Returns: TRUE if does have an RGB color attribute, FALSE otherwise.

Description: Routine to return a RGB attribute ID.
See also: AttrIDSetColor, AttrIDGetColor, AttrIDSetRGBColor, AttrIDSetWidth, , AttrIDSetIntAttrib, AttrIDGetObjectRGBColor, , AttrIDSetRGBDoubleColor, AttrIDGetRGBDoubleColor, AttrIDGetWidth,

7.2.41 AttrIDGetRGBColor2 (misc_attr_ids.c:205)

attributes

color

rgb

```
int AttrIDGetRGBColor2(const IPAttributeStruct *Attrs,
                       IPAttrIDType AttrID,
                       int *Red,
                       int *Green,
                       int *Blue)
```

Attrs: For which we would like to know the RGB of.

AttrID: Attribute unique ID of requested attribute.

Red, Green, Blue: Component of RGB color to initialize.

Returns: TRUE if does have an RGB color attribute, FALSE otherwise.

Description: Routine to return a RGB attribute ID.
See also: AttrIDSetColor, AttrIDGetColor, AttrIDSetRGBColor, AttrIDSetWidth, , AttrIDSetIntAttrib, AttrIDGetObjectRGBColor, , AttrIDSetRGBDoubleColor, AttrIDGetRGBDoubleColor, AttrIDGetWidth,

7.2.42 AttrIDGetRGBDoubleColor (misc_attr_ids.c:305)

attributes

color

rgb

```
int AttrIDGetRGBDoubleColor(const IPAttributeStruct *Attrs,
                            double *Red,
                            double *Green,
                            double *Blue)
```

Attrs: For which we would like to know the RGB of.

Red, Green, Blue: Component of RGB color to initialize.

Returns: TRUE if does have an RGB color attribute, FALSE otherwise.

Description: Routine to return a RGB attribute ID.
See also: AttrIDSetColor, AttrIDGetColor, AttrIDSetRGBColor, AttrIDSetWidth, , AttrIDGetWidth, AttrIDSetIntAttrib, AttrIDGetObjectRGBColor, , AttrIDSetRGBDoubleColor,

7.2.43 AttrIDGetRealAttrib (misc_attr_ids.c:709)

attributes

```
IrtRType AttrIDGetRealAttrib(const IPAttributeStruct *Attrs,
                             IPAttrIDType AttribID)
```

Attrs: Attribute list to search for requested attribute.

AttribID: Attribute unique ID of requested attribute.

Returns: Found attribute, or IP_ATTR_BAD_REAL if not found.

Description: Routine to get a IrtRType attribute.
See also: AttrIDSetIntAttrib, AttrIDGetIntAttrib, AttrIDSetPtrAttrib, , AttrIDGetPtrAttrib, AttrIDSetRealAttrib, AttrIDSetRealPtrAttrib, , AttrIDSetStrAttrib, AttrIDGetStrAttrib, AttrIDGetUVAttrib, , AttrIDSetUVAttrib, AttrIDGetRealAttrib, AttrIDGetRealPtrAttrib,

7.2.44 AttrIDGetRealPtrAttrib (misc_attr_ids.c:808)

attributes

```
IrtRType *AttrIDGetRealPtrAttrib(const IPAttributeStruct *Attrs,
                                 IPAttrIDType AttribID)
```

Attrs: Attribute list to search for requested attribute.

AttribID: Attribute unique ID of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a IrtRType * attribute.
See also: AttrIDSetIntAttrib, AttrIDGetIntAttrib, AttrIDSetPtrAttrib, , AttrIDGetPtrAttrib, AttrIDSetRealAttrib, AttrIDSetRealPtrAttrib, , AttrIDSetStrAttrib, AttrIDGetStrAttrib, AttrIDGetUVAttrib, , AttrIDSetUVAttrib, AttrIDGetRealAttrib, AttrIDGetRealPtrAttrib,

7.2.45 AttrIDGetRefPtrAttrib (misc_attr_ids.c:633)

attributes

```
VoidPtr AttrIDGetRefPtrAttrib(const IPAttributeStruct *Attrs,
                              IPAttrIDType AttribID)
```

Attrs: Attribute list to search for requested attribute.

AttribID: Attribute unique ID of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a pointer reference attribute.
See also: AttrIDSetIntAttrib, AttrIDGetIntAttrib, AttrIDSetPtrAttrib, , AttrIDSetRealAttrib, AttrIDSetRealPtrAttrib, AttrIDGetRealAttrib, , AttrIDGetRealPtrAttrib, AttrIDSetStrAttrib, AttrIDGetStrAttrib, , AttrIDGetUVAttrib, AttrIDSetUVAttrib, AttrIDSetPtrAttrib, , AttrIDGetPtrAttrib, AttrIDGetRefPtrAttrib,

7.2.46 AttrIDGetStrAttrib (misc_attr_ids.c:985)

attributes

```
const char *AttrIDGetStrAttrib(const IPAttributeStruct *Attrs,
                               IPAttrIDType AttribID)
```

Attrs: Attribute list to search for requested attribute.

AttribID: Attribute unique ID of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a string attribute.
See also: AttrIDSetColor AttrIDGetColor AttrIDSetRGBColor AttrIDGetRGBColor, , AttrIDSetRGBDoubleColor AttrIDGetRGBDoubleColor AttrIDSetWidth, , AttrIDGetWidth AttrIDSetIntAttrib AttrIDGetIntAttrib, , AttrIDSetPtrAttrib, AttrIDGetRefPtrAttrib AttrIDSetRealAttrib AttrIDGetRealAttrib, , AttrIDSetRealPtrAttrib AttrIDGetRealPtrAttrib AttrIDSetUVAttrib, , AttrIDGetUVAttrib,

7.2.47 AttrIDGetUVAttrib (misc_attr_ids.c:889)

attributes

```
float *AttrIDGetUVAttrib(const IPAttributeStruct *Attrs,
                         IPAttrIDType AttribID)
```

Attrs: Attribute list to search for requested attribute.

AttribID: Attribute unique ID of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a UV attribute.
See also: AttrIDSetIntAttrib, AttrIDGetIntAttrib, AttrIDSetPtrAttrib, , AttIDrGetPtrAttrib, AttrIDSetRealAttrib, AttrIDGetRealPtrAttrib, AttrIDSetStrAttrib, AttrIDGetStrAttrib, AttrIDGetUVAttrib, AttrIDSetUVAttrib, AttrIDGetUVAttrib,

7.2.48 AttrIDGetWidth (misc_attr_ids.c:381)

attributes

width

```
IrtRType AttrIDGetWidth(const IPAttributeStruct *Attrs)
```

Attrs: For which we would like to know the width of.

Returns: Width or IP_ATTR_NO_WIDTH if no width set.

Description: Routine to return a width attribute ID.
See also: AttrIDSetColor, AttrIDGetColor, AttrIDSetRGBColor, AttrIDGetRGBColor, , AttrIDSetWidth, AttrIDGetRealAttrib, AttrIDGetObjectWidth,

7.2.49 AttrIDRemoveOneAttribute (misc_attr_ids.c:1066)

attributes

```
void AttrIDRemoveOneAttribute(IPAttributeStruct **Attrs, IPAttrIDType AttrID)
```

Attrs: To search for an attribute named Name and remove it.

AttrID: ID of attribute to remove.

Returns: void

Description: Routine to remove the attribute named Name from the given Attr list.
See also: AttrIDFreeOneAttribute,

7.2.50 AttrIDSetColor (misc_attr_ids.c:43)

attributes

color

```
void AttrIDSetColor(IPAttributeStruct **Attrs, int Color)
```

Attrs: Where to place the color attribute.

Color: New color.

Returns: void

Description: Routine to set a color attribute ID.
See also: AttrIDGetColor, AttrIDSetRGBColor, AttrIDGetRGBColor, AttrIDSetWidth, , AttrIDGetWidth, AttrIDSetIntAttrib, AttrIDSetObjectColor,

7.2.51 AttrIDSetIntAttrib (misc_attr_ids.c:431)

attributes

```
void AttrIDSetIntAttrib(IPAttributeStruct **Attrs,
                        IPAttrIDType AttribID,
                        int Data)
```

Attrs: Attribute list where to place new attribute.

AttribID: Attribute unique ID of requested attribute.

Data: Ingeter attribute to save.

Returns: void

Description: Routine to set an integer attribute ID.
See also: AttrIDGetIntAttrib, AttrIDSetPtrAttrib, AttrIDGetPtrAttrib, , AttrIDSetRealPtrAttrib, AttrIDGetRealAttrib, AttrIDGetRealPtrAttrib, , AttrIDSetStrAttrib, AttrIDGetStrAttrib, AttrIDGetUVAttrib, , AttrIDSetRealAttrib, AttrIDSetUVAttrib, AttrIDSetIntAttrib,

7.2.52 AttrIDSetPtrAttrib (misc_attr_ids.c:514)

attributes

```
void AttrIDSetPtrAttrib(IPAttributeStruct **Attrs,
                        IPAttrIDType AttribID,
                        VoidPtr Data)
```

Attrs: Attribute list where to place new attribute.

AttribID: Attribute unique ID of requested attribute.

Data: Pointer attribute to save.

Returns: void

Description: Routine to set a pointer attribute.
See also: AttrIDSetIntAttrib, AttrIDGetIntAttrib, AttrIDGetPtrAttrib, , AttrIDSetRealPtrAttrib, AttrIDGetRealAttrib, AttrIDGetRealPtrAttrib, , AttrIDSetStrAttrib, AttrIDGetStrAttrib, AttrIDGetUVAttrib, , AttrIDSetUVAttrib, AttrIDSetRefPtrAttrib, AttrIDGetRefPtrAttrib, , AttrIDSetRealAttrib, AttrIDSetPtrAttrib,

7.2.53 AttrIDSetRGBColor (misc_attr_ids.c:117)

attributes
color
rgb

```
void AttrIDSetRGBColor(IPAttributeStruct **Attrs, int Red, int Green, int Blue)
```

Attrs: Where to place the TGB color attribute.

Red, Green, Blue: Component of RGB color.

Returns: void

Description: Routine to set an RGB color attribute ID.
See also: AttrIDSetColor, AttrIDGetColor, AttrIDGetRGBColor, AttrIDSetWidth, , AttrIDSetIntAttrib, AttrIDSetObjectRGBColor, AttrIDGetWidth,

7.2.54 AttrIDSetRGBDoubleColor (misc_attr_ids.c:260)

attributes
color
rgb

```
void AttrIDSetRGBDoubleColor(IPAttributeStruct **Attrs,
                             double Red,
                             double Green,
                             double Blue)
```

Attrs: Where to place the TGB color attribute.

Red, Green, Blue: Component of RGB color.

Returns: void

Description: Routine to set a double floating point RGB color attribute ID.
See also: AttrIDSetColor, AttrIDGetColor, AttrIDGetRGBColor, AttrIDSetWidth, , AttrIDSetIntAttrib, AttrIDSetObjectRGBColor, AttrIDGetRGBDoubleColor, AttrIDGetWidth,

7.2.55 AttrIDSetRealAttrib (misc_attr_ids.c:669)

attributes

```
void AttrIDSetRealAttrib(IPAttributeStruct **Attrs,
                         IPAttrIDType AttribID,
                         IrtRType Data)
```

Attrs: Attribute list where to place new attribute.

AttribID: Attribute unique ID of requested attribute.

Data: IrtRType attribute to save.

Returns: void

Description: Routine to set a IrtRType attribute.
See also: AttrIDSetIntAttrib, AttrIDGetIntAttrib, AttrIDSetPtrAttrib, , AttrIDGetPtrAttrib, AttrIDGetRealAttrib, AttrIDGetRealPtrAttrib, , AttrIDSetStrAttrib, AttrIDGetStrAttrib, AttrIDGetUVAttrib, , AttrIDSetUVAttrib, AttrIDSetRealAttrib, AttrIDSetRealPtrAttrib,

7.2.56 AttrIDSetRealPtrAttrib (misc_attr_ids.c:758)

attributes

```
void AttrIDSetRealPtrAttrib(IPAttributeStruct **Attrs,
                            IPAttrIDType AttribID,
                            IrtRType *Data,
                            int DataLen)
```

Attrs: Attribute list where to place new attribute.

AttribID: Attribute unique ID of requested attribute.

Data: IrtRType * attribute to save.

DataLen: If positive allocates and copies that many reals from Data. Otherwise, Data (of length -DataLen) is used directly.

Returns: void

Description: Routine to set a IrtRType * attribute. Attribute will own Data.
See also: AttrIDSetIntAttrib, AttrIDGetIntAttrib, AttrIDSetPtrAttrib, , AttrIDGetPtrAttrib, AttrIDGetRealAttrib, AttrIDGetRealPtrAttrib, , AttrIDSetStrAttrib, AttrIDGetStrAttrib, AttrIDGetUVAttrib, , AttrIDSetUVAttrib, AttrIDSetRealAttrib, AttrIDSetRealPtrAttrib,

7.2.57 AttrIDSetRefPtrAttrib (misc_attr_ids.c:592)

attributes

```
void AttrIDSetRefPtrAttrib(IPAttributeStruct **Attrs,
                           IPAttrIDType AttribID,
                           VoidPtr Data)
```

Attrs: Attribute list where to place new attribute.

AttribID: Attribute unique ID of requested attribute.

Data: Pointer attribute to save.

Returns: void

Description: Routine to set a pointer reference attribute.
See also: AttrIDSetIntAttrib, AttrIDGetIntAttrib, AttrIDGetPtrAttrib, , AttrIDSetRealAttrib, AttrIDSetRealPtrAttrib, AttrIDGetRealAttrib, , AttrIDGetRealPtrAttrib, AttrIDSetStrAttrib, AttrIDGetStrAttrib, , AttrIDGetUVAttrib, AttrIDSetUVAttrib, AttrIDSetPtrAttrib, , AttrIDGetPtrAttrib, AttrIDSetRefPtrAttrib,

7.2.58 AttrIDSetStrAttrib (misc_attr_ids.c:941)

attributes

```
void AttrIDSetStrAttrib(IPAttributeStruct **Attrs,
                        IPAttrIDType AttribID,
                        const char *Data)
```

Attrs: Attribute list where to place new attribute.

AttribID: Attribute unique ID of requested attribute.

Data: String attribute to save.

Returns: void

Description: Routine to set a string attribute.
See also: AttrIDSetIntAttrib, AttrIDGetIntAttrib, AttrIDSetPtrAttrib, , AttrIDGetPtrAttrib, AttrIDSetRealAttrib, AttrIDGetRealAttrib, , AttrIDSetRealPtrAttrib, AttrIDGetStrAttrib, AttrIDGetUVAttrib, , AttrIDSetUVAttrib, AttrIDSetStrAttrib,

7.2.59 AttrIDSetUVAttrib (misc_attr_ids.c:848)

attributes

```
void AttrIDSetUVAttrib(IPAttributeStruct **Attrs,
                       IPAttrIDType AttribID,
                       IrtRType U,
                       IrtRType V)
```

Attrs: Attribute list where to place new attribute.

AttribID: Attribute unique ID of requested attribute.

U, V: UV attribute to save.

Returns: void

Description: Routine to set a UV attribute.
See also: AttrIDSetIntAttrib, AttrIDGetIntAttrib, AttrIDSetPtrAttrib, , AttrIDGetPtrAttrib, AttrIDSetRealAttrib, AttrIDSetRealPtrAttrib, , AttrIDGetRealAttrib, AttrIDGetStrAttrib, AttrIDSetUVAttrib, , AttrIDSetStrAttrib,

7.2.60 AttrIDSetWidth (misc_attr_ids.c:343)

attributes

```
void AttrIDSetWidth(IPAttributeStruct **Attrs, IrtRType Width)
```

width

Attrs: Where to place the width attribute.

Width: New width.

Returns: void

Description: Routine to set a width attribute ID.
See also: AttrIDSetColor, AttrIDGetColor, AttrIDSetRGBColor, AttrIDGetRGBColor, , AttrIDGetWidth, AttrIDSetRealAttrib, AttrIDSetRealPtrAttrib, , AttrIDSetObjectWidth,

7.2.61 AttrInitHashTbl (miscatt1.c:347)

```
void AttrInitHashTbl(void)
```

Returns: void

Description: Initialize the hash table for attributes names.

7.2.62 AttrMergeAttributes (miscattr.c:1985)

attributes

```
IPAttributeStruct *AttrMergeAttributes(IPAttributeStruct *Orig,
                                       const IPAttributeStruct *Src,
                                       int Replace)
```

Orig: Original list of attributes. Updated in place.

Src: Attribute list to duplicate and merge into orig.

Replace: If TRUE, original attributes with the same name are replaced by old ones. If FALSE, original attributes with same name are left intact.

Returns: Merged attribute list.

Description: Routine to copy an attribute list.

7.2.63 AttrReverseAttributes (miscattr.c:1567)

attributes

```
IPAttributeStruct *AttrReverseAttributes(IPAttributeStruct *Attr)
```

Attr: To reverse, in place.

Returns: The reversed list, in place.

Description: Routine to reverse the given Attr list.

7.2.64 AttrSetIntAttrib (miscattr.c:253)

attributes

```
void AttrSetIntAttrib(IPAttributeStruct **Attrs, const char *Name, int Data)
```

Attrs: Attribute list where to place new attribute.

Name: Name of the newly introduced attribute.

Data: Integer attribute to save.

Returns: void

Description: Routine to set an integer attribute.
See also: AttrGetIntAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, AttrSetRealAttrib, , AttrSetRealPtrAttrib, AttrGetRealAttrib, AttrGetRealPtrAttrib, , AttrSetStrAttrib, AttrGetStrAttrib, AttrGetUVAttrib, , AttrSetUVAttrib, AttrSetIntAttrib2,

7.2.65 AttrSetIntAttrib2 (miscattr.c:287)

attributes

```
void AttrSetIntAttrib2(IPAttributeStruct **Attrs,
                       IPAttrNumType AttribNum,
                       int Data)
```

Attrs: Attribute list where to place new attribute.

AttribNum: Unique ID derived from name of requested attribute.

Data: Ingeter attribute to save.

Returns: void

Description: Routine to set an integer attribute.
See also: AttrGetIntAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, AttrSetRealAttrib, , AttrSetRealPtrAttrib, AttrGetRealAttrib, AttrGetRealPtrAttrib, , AttrSetStrAttrib, AttrGetStrAttrib, AttrGetUVAttrib, , AttrSetUVAttrib, AttrSetIntAttrib,

7.2.66 AttrSetPtrAttrib (miscattr.c:397)

attributes

```
void AttrSetPtrAttrib(IPAttributeStruct **Attrs,
                      const char *Name,
                      VoidPtr Data)
```

Attrs: Attribute list where to place new attribute.

Name: Name of the newly introduced attribute.

Data: Pointer attribute to save.

Returns: void

Description: Routine to set a pointer attribute.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrGetPtrAttrib, AttrSetRealAttrib, , AttrSetRealPtrAttrib AttrGetRealAttrib, AttrGetRealPtrAttrib, , AttrSetStrAttrib, AttrGetStrAttrib, AttrGetUVAttrib, AttrSetUVAttrib, AttrSetRefPtrAttrib, AttrGetRefPtrAttrib, AttrSetPtrAttrib2,

7.2.67 AttrSetPtrAttrib2 (miscattr.c:432)

attributes

```
void AttrSetPtrAttrib2(IPAttributeStruct **Attrs,
                       IPAttrNumType AttribNum,
                       VoidPtr Data)
```

Attrs: Attribute list where to place new attribute.

AttribNum: Unique ID derived from name of requested attribute.

Data: Pointer attribute to save.

Returns: void

Description: Routine to set a pointer attribute.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrGetPtrAttrib, AttrSetRealAttrib, , AttrSetRealPtrAttrib, AttrGetRealAttrib, AttrGetRealPtrAttrib, , AttrSetStrAttrib, AttrGetStrAttrib, AttrGetUVAttrib, , AttrSetUVAttrib, AttrSetRefPtrAttrib, AttrGetRefPtrAttrib, , AttrSetPtrAttrib,

7.2.68 AttrSetRGBDoubleColor (miscattr.c:162)

attributes

color

rgb

```
void AttrSetRGBDoubleColor(IPAttributeStruct **Attrs,
                           double Red,
                           double Green,
                           double Blue)
```

Attrs: Where to place the TGB color attribute.

Red, Green, Blue: Component of RGB color.

Returns: void

Description: Routine to set a Floating point 64 RGB color attribute.
See also: AttrSetIntAttrib, AttrSetObjectRGBColor, AttrGetRGBDoubleColor,

7.2.69 AttrSetRealAttrib (miscattr.c:686)

attributes

```
void AttrSetRealAttrib(IPAttributeStruct **Attrs,
                       const char *Name,
                       IrtRType Data)
```

Attrs: Attribute list where to place new attribute.

Name: Name of the newly introduce dattribute.

Data: IrtRType attribute to save.

Returns: void

Description: Routine to set a IrtRType attribute.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, , AttrGetRealAttrib, AttrGetRealPtrAttrib, AttrSetStrAttrib, , AttrGetStrAttrib, AttrGetUVAttrib, AttrSetUVAttrib, AttrSetRealAttrib2, , AttrSetRealPtrAttrib,

7.2.70 AttrSetRealAttrib2 (miscattr.c:720)

attributes

```
void AttrSetRealAttrib2(IPAttributeStruct **Attrs,
                        IPAttrNumType AttribNum,
                        IrtRType Data)
```

Attrs: Attribute list where to place new attribute.

AttribNum: Unique ID derived from name of requested attribute.

Data: IrtRType attribute to save.

Returns: void

Description: Routine to set a IrtRType attribute.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, , AttrGetRealAttrib, AttrGetRealPtrAttrib, AttrSetStrAttrib, , AttrGetStrAttrib, AttrGetUVAttrib, AttrSetUVAttrib, AttrSetRealAttrib, , AttrSetRealPtrAttrib,

7.2.71 AttrSetRealPtrAttrib (miscattr.c:840)

attributes

```
void AttrSetRealPtrAttrib(IPAttributeStruct **Attrs,
                          const char *Name,
                          IrtRType *Data,
                          int DataLen)
```

Attrs: Attribute list where to place new attribute.

Name: Name of the newly introduce dattribute.

Data: IrtRType * attribute to save.

DataLen: If positive allocates and copies that many reals from Data. Otherwise, Data (of length -DataLen) is used directly.

Returns: void

Description: Routine to set a IrtRType * attribute. Attribute will own Data.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, , AttrGetRealAttrib, AttrGetRealPtrAttrib, AttrSetStrAttrib, , AttrGetStrAttrib, AttrGetUVAttrib, AttrSetUVAttrib, AttrSetRealAttrib2, , AttrSetRealPtrAttrib,

7.2.72 AttrSetRealPtrAttrib2 (miscattr.c:878)

attributes

```
void AttrSetRealPtrAttrib2(IPAttributeStruct **Attrs,
                           IPAttrNumType AttribNum,
                           IrtRType *Data,
                           int DataLen)
```

Attrs: Attribute list where to place new attribute.

AttribNum: Unique ID derived from name of requested attribute.

Data: IrtRType * attribute to save.

DataLen: If positive allocates and copies that many reals from Data. Otherwise, Data (of length -DataLen) is used directly.

Returns: void

Description: Routine to set a IrtRType * attribute. Attribute will own Data.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, , AttrGetRealAttrib, AttrGetRealPtrAttrib, AttrSetStrAttrib, , AttrGetStrAttrib, AttrGetUVAttrib, AttrSetUVAttrib, AttrSetRealAttrib, , AttrSetRealPtrAttrib,

7.2.73 AttrSetRefPtrAttrib (miscattr.c:542)

attributes

```
void AttrSetRefPtrAttrib(IPAttributeStruct **Attrs,
                         const char *Name,
                         VoidPtr Data)
```

Attrs: Attribute list where to place new attribute.

Name: Name of the newly introduced attribute.

Data: Pointer attribute to save.

Returns: void

Description: Routine to set a pointer reference attribute.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrGetPtrAttrib, AttrSetRealAttrib, , AttrSetRealPtrAttrib, AttrGetRealAttrib, AttrGetRealPtrAttrib, , AttrSetStrAttrib, AttrGetStrAttrib, AttrGetUVAttrib, , AttrSetUVAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, , AttrSetRefPtrAttrib2,

7.2.74 AttrSetRefPtrAttrib2 (miscattr.c:577)

attributes

```
void AttrSetRefPtrAttrib2(IPAttributeStruct **Attrs,
                          IPAttrNumType AttribNum,
                          VoidPtr Data)
```

Attrs: Attribute list where to place new attribute.

AttribNum: Unique ID derived from name of requested attribute.

Data: Pointer attribute to save.

Returns: void

Description: Routine to set a pointer reference attribute.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrGetPtrAttrib, AttrSetRealAttrib, , AttrSetRealPtrAttrib, AttrGetRealAttrib, AttrGetRealPtrAttrib, , AttrSetStrAttrib, AttrGetStrAttrib, AttrGetUVAttrib, , AttrSetUVAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, , AttrSetRefPtrAttrib,

7.2.75 AttrSetStrAttrib (miscattr.c:1152)

attributes

```
void AttrSetStrAttrib(IPAttributeStruct **Attrs,
                      const char *Name,
                      const char *Data)
```

Attrs: Attribute list where to place new attribute.

Name: Name of the newly introduced attribute.

Data: String attribute to save.

Returns: void

Description: Routine to set a string attribute.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, , AttrSetRealAttrib, AttrGetRealAttrib, AttrSetRealPtrAttrib, , AttrGetStrAttrib, AttrGetUVAttrib, AttrSetUVAttrib, AttrSetStrAttrib2,

7.2.76 AttrSetStrAttrib2 (miscattr.c:1185)

attributes

```
void AttrSetStrAttrib2(IPAttributeStruct **Attrs,
                       IPAttrNumType AttribNum,
                       const char *Data)
```

Attrs: Attribute list where to place new attribute.

AttribNum: Unique ID derived from name of requested attribute.

Data: String attribute to save.

Returns: void

Description: Routine to set a string attribute.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, , AttrSetRealAttrib, AttrGetRealAttrib, AttrSetRealPtrAttrib, , AttrGetStrAttrib, AttrGetUVAttrib, AttrSetUVAttrib, AttrSetStrAttrib,

7.2.77 AttrSetUVAttrib (miscattr.c:999)

attributes

```
void AttrSetUVAttrib(IPAttributeStruct **Attrs,
                     const char *Name,
                     IrtRType U,
                     IrtRType V)
```

Attrs: Attribute list where to place new attribute.

Name: Name of the newly introduced attribute.

U, V: UV attribute to save.

Returns: void

Description: Routine to set a UV attribute.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, , AttrSetRealAttrib, AttrSetRealPtrAttrib, AttrGetRealAttrib, , AttrSetStrAttrib, AttrGetStrAttrib, AttrSetUVAttrib2,

7.2.78 AttrSetUVAttrib2 (miscattr.c:1033)

attributes

```
void AttrSetUVAttrib2(IPAttributeStruct **Attrs,
                      IPAttrNumType AttribNum,
                      IrtRType U,
                      IrtRType V)
```

Attrs: Attribute list where to place new attribute.

AttribNum: Unique ID derived from name of requested attribute.

U, V: UV attribute to save.

Returns: void

Description: Routine to set a UV attribute.
See also: AttrSetIntAttrib, AttrGetIntAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, , AttrSetRealAttrib, AttrSetRealPtrAttrib, AttrGetRealAttrib, , AttrGetStrAttrib, AttrSetUVAttrib2, AttrSetStrAttrib,

7.2.79 AttrTraceAttributes (miscattr.c:1291)

attributes

```
const IPAttributeStruct *AttrTraceAttributes(
                                        const IPAttributeStruct *TraceAttrs,
                                        const IPAttributeStruct *FirstAttrs)
```

TraceAttrs: If not NULL, contains the previously returned attribute.

FirstAttrs: First attribute in list, usually NULL in all but the first invocation.

Returns: Next attribute in list.

Description: Routine to aid in scanning a list of attributes. If TraceAttrs != NULL, a ptr to its attribute list is saved and the next attribute is returned every call until the end of the list is reached, in which NULL is returned. FirstAttrs should be NULL in all but the first call in the sequence. Attributes with names starting with an underscore '_' are assumed to be temporary or internal and are skipped.

7.2.80 E2TFreeNodetree (exprtree.c:1830)

```
void E2TFreeNodetree(IritE2TExprNodeStruct *Root)
```

Root: Tree to release.

Returns: void

Description: Routine to free a tree - release all memory allocated by it.
See also:

7.2.81 GAGetArgs (getarg.c:201)

command line arguments

```
int GAGetArgs(int va_alist, ...)
```

va_alist: Do "man stdarg".

...: Rest of optional parameters

Returns: TRUE if command line was valid, FALSE otherwise.

Description: Routine to access command line arguments and interpret them, by getting access to the main routine's argc/argv interface and a control string that prescribes the expected options. Returns ARG_OK (0) is case of succesfull parsing, error code else...
Format of CtrlStr format: The control string passed to GAGetArgs controls the way argv (argc) are parsed. Each entry in this string must have no spaces in it. The First Entry is the name of the program which is usually ignored except when GAPrintHowTo is called. All the other entries (except the last one which will be discussed shortly) must have the following format:

1. One letter which sets the option letter (i.e. 'x' for option '-x').
2. '!' or '%' to determines if this option is really optional ('%') or it must be provided by the user ('!').
3. '-' always.
4. Alpha numeric string (and '|' to denote a space), usually ignored, but used by GAPrintHowTo to describe the meaning of this option.
5. Sequences that start with either '!' or '%'. Again if '!' then this sequence must exists (only if its option flag is given), and if '%' it is optional. Each sequence will be followed by one or two characters which defines the kind of the input:
   5.1. d, x, o, u - integer is expected (decimal, hex, octal base or unsigned).
   5.2. D, X, O, U - long integer is expected (same as above).
   5.3. f - float number is expected.
   5.4. F - double number is expected.
   5.5. s - string is expected.
   5.6. *? - any number of '?' kind (d, x, o, u, D, X, O, U, f, F, s) will match this one. If '?' is numeric, it scans until none numeric input is given. If '?' is 's' then it scans up to the next option or end of argv.

If the last parameter given in the CtrlStr, is not an option (i.e. the second char is not in ['!', '%'] and the third one is not '-'), all what remained from argv is hooked to it.
The variables passed to GAGetArgs (starting from 4th parameter) MUST match the order of options in the CtrlStr. For each option, an address of an integer must be passed. This integer must initialized by 0. If that option is given in the command line, it will be set to one. Otherwise, this integer will not be affected. In addition, the sequences that might follow an option require the following parameter(s) to be passed

1. d, x, o, u - pointer to an integer (int *).
2. D, X, O, U - pointer to a long (long *).
3. f - pointer to a float (float *).
4. F - pointer to a double (double *).
5. s - pointer to a char * (char **). NO pre-allocation is required.
6. *? - TWO variables are passed for each such wild character request. The first variable is an address of an integer, and it will return the number of parameters actually hooked to this sequence. The second variable is a pointer to a pointer to type ? (? **). It will return an address of a vector of pointers of type ?, terminated with a NULL pointer. NO pre-allocation is required. These two variables behaves very much like the argv/argc pair and are used the "trap" unused command line options.

Examples:

```
    "Example1  i%-OneInteger!d  s%-Strings!*s  j%-  k!-Double!F  Files!*s"
 Will match: Example1 -i 77 -s String1 String2 String3 -k 88.2 File1 File2
   or match: Example1 -s String1 -k 88.3 -i 999 -j
    but not: Example1 -i 77 78 (i expects one integer, k must be specified).
```

The option k must exists in the above example and if '-i' is prescribed one integer argument must follow it. In the first example, File1 & File2, will match Files in the control string. The order of the options in the command line is irrelevant. A call to GAPrintHowTo with this CtrlStr will print the info:

```
Example1 [-i OneIngeter] [-s Strings...] [-j] -k Float Files...
```

The parameters below are stdarg style and in fact are expecting the following:

```
GAGetArgs(argc, argv, CtrlStr, ...);
```

1. argc, argv: The usual C interface from the main routine of the program.
2. CtrlStr: Defining the types/options to expect in the command line.
3. ...: list of addreses of variables to initialize according to parsed command line.

See also: GAPrintErrMsg, GAPrintHowTo,

7.2.82 GAPrintErrMsg (getarg.c:793)

command line arguments

```
void GAPrintErrMsg(int Error)
```

Error: Error type as returned by GAGetArgs.

Returns: void

Description: Routine to print a description of an error, for this module:
See also: GAStringErrMsg, GAPrintHowTo, GAGetArgs,

7.2.83 GAPrintHowTo (getarg.c:922)

command line arguments

```
void GAPrintHowTo(const char *CtrlStr)
```

CtrlStr: Defining the types/options to expect in the command line.

Returns: void

Description: Routine to print the correct format of command line allowed. For example, for the following control string,

```
"Example1  i%-OneInteger!d  s%-Strings!*s  j%-  k!-Double!F  Files"
```

This routine will print

```
Example1 [-i OneIngeter] [-s Strings...] [-j] -k Double Files...
```

See also: GAPrintErrMsg, GAStringHowTo, GAGetArgs,

7.2.84 GAStringErrMsg (getarg.c:742)

command line arguments

```
char *GAStringErrMsg(int Error, char *OutStr)
```

Error: Error type as returned by GAGetArgs.

OutStr: Where to place the error message.

Returns: The error message string (OutStr).

Description: Routine to print a description of an error to a string.
See also: GAPrintErrMsg, GAStringHowTo, GAGetArgs,

7.2.85 GAStringHowTo (getarg.c:821)

command line arguments

```
char *GAStringHowTo(const char *CtrlStr, char *OutStr)
```

CtrlStr: Defining the types/options to expect in the command line.

OutStr: Where to place the how to message.

Returns: The how-to message string (OutStr).

Description: Routine to write the correct format of command line allowed, to a string. For example, for the following control string,

```
"Example1  i%-OneInteger!d  s%-Strings!*s  j%-  k!-Double!F  Files"
```

This routine will write

```
Example1 [-i OneIngeter] [-s Strings...] [-j] -k Double Files...
```

See also: GAPrintHowTo, GAStringErrMsg, GAGetArgs,

7.2.86 IritApproxStrStrMatch (xgeneral.c:240)

```
IrtRType IritApproxStrStrMatch(const char *Str1,
                               const char *Str2,
                               int IgnoreCase)
```

Str1, Str2: The two strings to compare.

IgnoreCase: TRUE to ignore case, FALSE to consider as different.

Returns: Perfect match returns 1.0, otherwise match estimation between 0.0 and 1.0.

Description: Provides an approximated comparison between two strings.

7.2.87 IritAscii2WChar (config.c:757)

```
wchar_t *IritAscii2WChar(const char *Str)
```

Str: Input multi byte Ascii string to convert to a wide string.

Returns: Converted string, allocated dynamically.

Description: Converts an Ascii (multi-byte) string to a wide (multi-short) string.
See also: IritWChar2Ascii, IritTextLocaleInit,

7.2.88 IritCPUTime (xgeneral.c:447)

time

```
IrtRType IritCPUTime(int Reset)
```

Reset: If TRUE, clock is reset back to zero.

Returns: CPU time since last reset or beginning of execution.

Description: Routine to compute the cpu time, in second, of the running process.

7.2.89 IritConfig (config.c:184)

configuration

cfg files

```
const char *IritConfig(const char *PrgmName,
                       const IritConfigStruct *SetUp,
                       int NumVar,
                       char *FoundName)
```

PrgmName: Name of program that uses this data base.

SetUp: Configuration data based.

NumVar: Number of entries on configuration data base.

FoundName: Name of config file read or NULL to disable.

Returns: Name of config file read, or NULL if error. Same as FoundName.

Description: Main routine of configuration file handling. Gets the program name, PrgmName, and the configuration data base that defines the acceptable variables, Setup, with Numvar entries.

7.2.90 IritConfigInitState (config.c:619)

```
void IritConfigInitState(void)
```

Returns: void

Description: An initialization function that is invoked at the very beginning of every main function in irit.

7.2.91 IritConfigPrint (config.c:69)

configuration

cfg files

```
void IritConfigPrint(const IritConfigStruct *SetUp, int NumVar)
```

SetUp: Configuration data based.

NumVar: Number of entries on configuration data base.

Returns: void

Description: Routine to print the current configuration data structure contents.

7.2.92 IritConfigSave (config.c:472)

configuration

cfg files

```
int IritConfigSave(const char *FileName,
                   const IritConfigStruct *SetUp,
                   int NumVar)
```

FileName: File to save configuration at.

SetUp: Configuration data based.

NumVar: Number of entries on configuration data base.

Returns: TRUE if saved successfully, FALSE otherwise.

Description: Saves the given configuration into a file.

7.2.93 IritDynMemoryDbgCheckMark (imalloc.c:177)

```
void IritDynMemoryDbgCheckMark(IrtRType *Start,
                               IrtRType *KeepStackStep,
                               IrtRType *TrackAllocID)
```

Start: 1 to initial check mark, 2 to also request stack trace info, 0 to terminate and dump result.

KeepStackStep: If Start = 2, sets stack-keeping every n'th alloc.

TrackAllocID: if non-zero, also track the unique allocation ID.

Returns: void

Description: Sets/clear dynamic memory check marks and reports. Initializes the dynamic memory testing routines, if not initialized until now.
See also: IritFree, IritMalloc, IritRealloc, IritDynMemoryDbgTestAll,

7.2.94 IritDynMemoryDbgInitTest (imalloc.c:255)

```
void IritDynMemoryDbgInitTest(void)
```

Returns: void

Description: Initialize dynamic memory testing routines, using envvars.
See also: IritFree, IritMalloc, IritRealloc, IritDynMemoryDbgCheckMark, IritDynMemoryDbgInitTest2,

7.2.95 IritDynMemoryDbgInitTest2 (imalloc.c:296)

```
void IritDynMemoryDbgInitTest2(int DebugMalloc, int DebugSearchAllocID)
```

DebugMalloc: Bitwise control over what to test: 0x02 - List possibly freed yet not malloced ptrs. 0x04 - Track unfreed memory blocks. 0x08 - Under windows use windows CrtDbg check. 0x10 - Under windows use windows CrtDbg check every 16 mallocs. 0x20 - Under windows, enable stack info trace. See IRIT_DEBUG_MALLOC_* at the beginning of file.

DebugSearchAllocID: Allocation ID to trace, 0 for no ID to trace.

Returns: void

Description: Initialize dynamic memory testing routines.
See also: IritFree, IritMalloc, IritRealloc, IritDynMemoryDbgCheckMark, IritDynMemoryDbgInitTest,

7.2.96 IritDynMemoryDbgMallocSearchID (imalloc.c:620)

```
void IritDynMemoryDbgMallocSearchID(int ID)
```

ID: Allocation ID to seek and abort at.

Returns: void

Description: Set the searched malloced ID. This function will take affect iff IritDebugMalloc is non zero as set via "IRIT_MALLOC" env.
See also: IritFree, IritMalloc, IritRealloc, IritDynMemoryDbgTestAll, , IritDebugMallocSearchPtr, IritDebugMallocAllocated, IritDynMemoryDbgInitTest, IritDynMemoryDbgCheckMark,

7.2.97 IritDynMemoryDbgNoReport (imalloc.c:782)

```
void *IritDynMemoryDbgNoReport(void *p)
```

p: Dynamically allocated pointer.

Returns: Exactly p, for composition of functions.

Description: remember this pointer as a special pointer we allocated statically once per entry so we will not report it as mem leak.
See also:

7.2.98 IritDynMemoryDbgTestAll (imalloc.c:119)

```
void IritDynMemoryDbgTestAll(void)
```

Returns: void

Description: Tests the content of the dynamic memory allocated.
See also: IritFree, IritMalloc, IritRealloc, IritDynMemoryDbgInitTest,

7.2.99 IritE2Expr2TreeDefaultFetchParamValue (exprtree.c:148)

```
IrtRType IritE2Expr2TreeDefaultFetchParamValue(MiscExprTreeGenInfoStruct *GI,
                                               const char *SData)
```

GI: General info. for the expression tree parsing functions.

SData: The parameter to fetch.

Returns: Fetched value of parameter.

Description: Default function to fetch the value of a parameter, during evaluation.

7.2.100 IritE2Expr2TreeSetFetchParamValueFunc (exprtree.c:124)

```
IritE2TExprNodeParamFuncType IritE2Expr2TreeSetFetchParamValueFunc(
                            MiscExprTreeGenInfoStruct *GI,
                            IritE2TExprNodeParamFuncType FetchParamValueFunc)
```

GI: General info. for the expression tree parsing functions.

FetchParamValueFunc: Function pointer to use to fetch a value of a parameter, given its name.

Returns: Old function pointer.

Description: Sets the function to use to fetch a parameter value, given its name, during evaluation.

7.2.101 IritE2TCmpTree (exprtree.c:1703)

```
int IritE2TCmpTree(const IritE2TExprNodeStruct *Root1,
                   const IritE2TExprNodeStruct *Root2)
```

Root1, Root2: The two trees to compare.

Returns: TRUE if equal, FALSE otherwise.

Description: Routine to compere two trees - for equality: The trees are compered to be symbolically equal i.e. A*B == B*A !
See also:

7.2.102 IritE2TCopyTree (exprtree.c:969)

```
IritE2TExprNodeStruct *IritE2TCopyTree(const IritE2TExprNodeStruct *Root)
```

Root: Tree to duplicate.

Returns: Duplicated tree.

Description: Routine to create a new copy of a given tree.
See also:

7.2.103 IritE2TDerivError (exprtree.c:1931)

```
int IritE2TDerivError(MiscExprTreeGenInfoStruct *GI)
```

GI: General info. for the expression tree parsing functions.

Returns: Error or 0 in none.

Description: Get a derivative error is was one or 0 in none.

7.2.104 IritE2TDerivTree (exprtree.c:1171)

```
IritE2TExprNodeStruct *IritE2TDerivTree(MiscExprTreeGenInfoStruct *GI,
                                        const IritE2TExprNodeStruct *Root,
                                        int Param)
```

GI: General info. for the expression tree parsing functions.

Root: To derive.

Param: The parameter to differentiate according to.

Returns: Derived tree.

Description: Routine to generate the tree represent the derivative of tree Root.
See also:

7.2.105 IritE2TEvalTree (exprtree.c:1036)

```
IrtRType IritE2TEvalTree(MiscExprTreeGenInfoStruct *GI,
                         const IritE2TExprNodeStruct *Root)
```

GI: General info. for the expression tree parsing functions.

Root: Tree to evaluate.

Returns: Evaluated result.

Description: Routine to evaluate a value of a given tree Root and set parameters.
See also:

7.2.106 IritE2TExpr2Tree (exprtree.c:248)

```
IritE2TExprNodeStruct *IritE2TExpr2Tree(MiscExprTreeGenInfoStruct *GI,
                                        const char s[])
```

GI: General info. for the expression tree parsing functions.

s: String expression to parse.

Returns: Built binary tree.

Description: Routine to convert the expression in string S into a binary tree. Algorithm: Using operator precedence with the following grammer: EXPR ::= EXPR | EXPR + EXPR | EXPR - EXPR EXPR ::= EXPR | EXPR * EXPR | EXPR / EXPR EXPR ::= EXPR | EXPR ^ EXPR EXPR ::= NUMBER | -EXPR | (EXPR) | FUNCTION FUCTION ::= SIN(EXPR) | COS(EXPR) | TAN(EXPR) ARCSIN(EXPR) | ARCCOS(EXPR) | ARCTAN(EXPR) SQRT(EXPR) | SQR(EXPR) | ABS(EXPR) LN(EXPR) | LOG(EXPR) | EXP(EXPR)
And left associativity for +, -, *, /, ^. Precedence of operators is as usual: <Highest> {unar minus} {^} {*,/} {+,-} <Lowest>
Returns NULL if an error was found, and error is in E2TParsingError
See also:

7.2.107 IritE2TExpr2TreeFree (exprtree.c:101)

```
void IritE2TExpr2TreeFree(MiscExprTreeGenInfoStruct *GI)
```

GI: N.S.F.I.

Returns: void

Description: Frees the local data and info needed by the expression tree module.

7.2.108 IritE2TExpr2TreeInit (exprtree.c:74)

```
MiscExprTreeGenInfoStruct *IritE2TExpr2TreeInit()
```

Returns: Local info. structure.

Description: Initializes local data and info needed by the expression tree module.

7.2.109 IritE2TFreeTree (exprtree.c:1112)

```
void IritE2TFreeTree(IritE2TExprNodeStruct *Root)
```

Root: Tree to free.

Returns: void

Description: Frees a given expression tree.
See also:

7.2.110 IritE2TParamInTree (exprtree.c:1769)

```
int IritE2TParamInTree(const IritE2TExprNodeStruct *Root,
                       const char *ParamName)
```

Root: Tree to examine for an existence of a parameter.

ParamName: Name of parameter to seek, or NULL to seek if any param.

Returns: TRUE if parameter exists, FALSE otherwise.

Description: Routine to test if the parameter is in the tree: If ParamName is NULL then any parameter return TRUE.
See also:

7.2.111 IritE2TParseError (exprtree.c:1909)

```
int IritE2TParseError(MiscExprTreeGenInfoStruct *GI)
```

GI: General info. for the expression tree parsing functions.

Returns: Error or 0 in none.

Description: Get a parsing error is was one or 0 in none.

7.2.112 IritE2TPrintTree (exprtree.c:795)

```
void IritE2TPrintTree(const IritE2TExprNodeStruct *Root, char *Str)
```

Root: Tree to print.

Str: Destination.

Returns: void

Description: Routine to print a content of Root (using inorder traversal): If *str = NULL print on stdout, else on given string str.

See also:

7.2.113 IritE2TSetParamValue (exprtree.c:1890)

```
void IritE2TSetParamValue(MiscExprTreeGenInfoStruct *GI,
                          IrtRType Value,
                          int Index)
```

GI: General info. for the expression tree parsing functions.

Value: New value to assign to a parameter.

Index: The index of the parameter.

Returns: void

Description: Routine to set the value of a Parameter before evaluating an expression.

7.2.114 IritEmulatePthreadMutexLock (xgeneral.c:888)

```
void IritEmulatePthreadMutexLock(IRIT_MUTEX *Mx)
```

Mx: The mutex to initialize/lock.

Returns: void

Description: Locking a mutex and also initializing it on the fly if so needed. Used by parallel code if needed.

7.2.115 IritEmulatePthreadMutexUnLock (xgeneral.c:917)

```
void IritEmulatePthreadMutexUnLock(IRIT_MUTEX Mx)
```

Mx: The mutex to unlock.

Returns: void

Description: Unlocking a mutex. Used by parallel code if needed.

7.2.116 IritFatalError (irit_ftl.c:60)

error trap

```
void IritFatalError(const char *Msg)
```

Msg: Error message to print.

Returns: void

Description: Default trap for IRIT programs for irit fatal errors. This function just prints the given error message and die.
See also: IritWarningMsg, IritInformationMsg, IritFatalErrorPrintf,

7.2.117 IritFatalErrorPrintf (irit2ftl.c:35)

error trap

```
void IritFatalErrorPrintf(const char *va_alist, ...)
```

va_alist: Do "man stdarg".

Returns: void

Description: Default trap for IRIT programs for irit fatal errors, printf style.
See also: IritFatalError, IritWarningMsgPrintf, IritInformationMsgPrintf,

7.2.118 IritFree (imalloc.c:644)

allocation

```
void IritFree(VoidPtr p)
```

p: Pointer to a block that needs to be freed.

Returns: void

Description: Routine to free dynamic memory for all IRIT program/tool/libraries. All requests to free dynamic memory should invoke this function.
See also: IritMalloc, IritRealloc, IritDynMemoryDbgCheckMark, , IritDynMemoryDbgInitTest,

7.2.119 IritFree2UnixFree (imalloc.c:150)

```
void IritFree2UnixFree(void *p)
```

p: Dynamically allocated pointer to free.

Returns: void

Description: For those who need to call IRIT insiead of C library functions.

7.2.120 IritGaussJordan (levenmar.c:575)

Gauss-Jordan elimination

```
int IritGaussJordan(IrtRType *A, IrtRType *B, unsigned N, unsigned M)
```

A: A matrix of N*N elements, is invalid on exit.

B: A matrix of N*M elements, contains the result on exit.

N: Described above.

M: Described above.

Returns: TRUE on success, FALSE if singular.

Description: This functions solves the linear equation Ax=B, using the Gauss-Jordan elimination algorithm.
See also:

7.2.121 IritHashTableCreate (hash_tbl.c:38)

```
IritHashTableStruct *IritHashTableCreate(IrtRType MinKeyVal,
                                         IrtRType MaxKeyVal,
                                         IrtRType KeyEps,
                                         int VecSize)
```

MinKeyVal: Minimum expected key value.

MaxKeyVal: Maximum expected key value.

KeyEps: Tolerance of two keys to be considered the same. Negative to never consider the same.

VecSize: Size of hash table to use.

Returns: Constructed has table.

Description: Constructs a simple hasing table.
See also: IritHashTableInsert, IritHashTableRemove, IritHashTableFree, , IritHashTableFind,

7.2.122 IritHashTableFind (hash_tbl.c:163)

```
VoidPtr IritHashTableFind(IritHashTableStruct *IHT,
                          VoidPtr Data,
                          IritHashCmpFuncType HashCmpFunc,
                          IrtRType Key)
```

IHT: IritHashTable structure.

Data: Element to compare against during the search.

HashCmpFunc: Test function to compare two data items. Returns -1,0,1 if first item is less, equal, greater than second item. If NULL, search is conducted by the Key only.

Key: Key with which to search in the table.

Returns: Found element, or NULL if none.

Description: Find an element in the hashing table. Search is conducted in two steps. First the search is performed by key and then by HashCmpFunc against Data.
See also: IritHashTableCreate, IritHashTableRemove, IritHashTableInsert, , IritHashTableFree,

7.2.123 IritHashTableFree (hash_tbl.c:277)

```
void IritHashTableFree(IritHashTableStruct *IHT)
```

IHT: IritHashTable structure to free.

Returns: void

Description: Free the entire hash table.
See also: IritHashTableCreate, IritHashTableInsert, IritHashTableFind, , IritHashTableRemove,

7.2.124 IritHashTableInsert (hash_tbl.c:89)

```
int IritHashTableInsert(IritHashTableStruct *IHT,
                        VoidPtr Data,
                        IritHashCmpFuncType HashCmpFunc,
                        IrtRType Key,
                        int RplcSame)
```

IHT: IritHashTable structure.

Data: Element to insert into the hash table.

HashCmpFunc: Test function to compare two data items. Returns -1,0,1 if first item is less, equal, greater than second item. If NULL, search is conducted by the Key only.

Key: Key with which to insert into the table.

RplcSame: TRUE, to replace a similar Data if detected, FALSE to skip.

Returns: TRUE if old element with the same key was found and replaced, FALSE if indeed a data with new key.

Description: Insert one element into the hashing table.
See also: IritHashTableCreate, IritHashTableFind, IritHashTableRemove, , IritHashTableFree,

7.2.125 IritHashTableRemove (hash_tbl.c:219)

```
int IritHashTableRemove(IritHashTableStruct *IHT,
                        VoidPtr Data,
                        IritHashCmpFuncType HashCmpFunc,
                        IrtRType Key)
```

IHT: IritHashTable structure.

Data: Element to compare against during the search.

HashCmpFunc: Test function to compare two data items. Returns -1,0,1 if first item is less, equal, greater than second item. If NULL, search is conducted by the Key only.

Key: Key with which to search in the table.

Returns: TRUE if element found and removed, FALSE if not found.

Description: Remove an element from the hashing table. Search is conducted in two steps. First the search is performed by key and then by HashCmpFunc against Data.

See also: IritHashTableCreate, IritHashTableFind, IritHashTableInsert, , IritHashTableFree,

7.2.126 IritImgPrcssAppFunOnLine (img_prcss.c:742)

```
void IritImgPrcssAppFunOnLine(IritImgPrcssImgStruct *Image,
                              int X0,
                              int Y0,
                              int X1,
                              int Y1,
                              IritImgPrcssFunctionOnPixel F,
                              void *Data)
```

Image: The image processing library image struct.

X0, Y0: The coordinates for the first image.

X1, Y1: The coordinates for the second image.

F: The function to call on the iterated pixels

Data: Data for the function

Returns: void

Description: Given an image and two points on the image, iterates over the pixels that lay on the line between the two points, and for each pixel, calls the function with the provided data and the pixel coordinates

7.2.127 IritImgPrcssBiSobel (img_prcss.c:406)

Convolution
filter
Sobel
Edges
Edge detection
Bi Directional

```
IritImgPrcssImgStruct *IritImgPrcssBiSobel(const IritImgPrcssImgStruct *Image)
```

Image: The image to apply the Bi Directional Sobel operator on.

Returns: A copy of the image after applying the Bi Directional Sobel operator.

Description: Given an image, applies the Sobel operator twice(vertical and horizontal) on it. it then calculates the final color value by: SQRT ((G1 ^ 2) + (G2 ^ 2)) where G1 and G2 are the gray scale intensities for the results of the vertical and horizontal runs. Equivalent to calling ApplyConvolution3Image with the following filters { -1, 0, 1, -2, 0, 2, -1, 0, 1 } { -1, -2, -1, 0, 0, 0, 1, 2, 1 }.

7.2.128 IritImgPrcssConv3 (img_prcss.c:301)

convolution
filter

```
IritImgPrcssImgStruct *IritImgPrcssConv3(const IritImgPrcssImgStruct *Image,
                                         float Filter[9],
                                         int Normalize)
```

Image: The image to apply the convolution on.

Filter: An array of floats containing the 3x3 filter data.

Normalize: TRUE to clip to [0, 1] the results.

Returns: The image after applying the filter by convolution.

Description: Given an image and a 3x3 filter, applies the filter on a zero padded version of the image. the function prepares the filter for convolution.

7.2.129 IritImgPrcssCopyImg (img_prcss.c:149)

Copy image

copy constructor

```
IritImgPrcssImgStruct *IritImgPrcssCopyImg(const IritImgPrcssImgStruct *Image)
```

Image: Image to copy.

Returns: A copy of the provided image.

Description: Allocates a new image filled with pixels from provided Image.

7.2.130 IritImgPrcssCreateBlkImg (img_prcss.c:92)

Create image

constructor

black

```
IritImgPrcssImgStruct *IritImgPrcssCreateBlkImg(unsigned int Height,
                                                unsigned int Width)
```

Height: Height of the image to be produced.

Width: Width of the image to be produced.

Returns: The black produced image.

Description: Allotcates a new image filled with black pixels of the dimensions Height and Width.

7.2.131 IritImgPrcssCreateImg (img_prcss.c:121)

create image

constructor

```
IritImgPrcssImgStruct *IritImgPrcssCreateImg(unsigned int Height,
                                             unsigned int Width,
                                             float *Data)
```

Height: Height of the image to be produced.

Width: Width of the image to be produced.

Data: Data to occupy the image with of size Height * Width * 4.

Returns: The newly allocated image.

Description: Allotcates a new image filled with pixels from provided data of the dimensions Height and Width.

7.2.132 IritImgPrcssCreateWhiteImg (img_prcss.c:66)

Create image

constructor

white

```
IritImgPrcssImgStruct *IritImgPrcssCreateWhiteImg(unsigned int Height,
                                                  unsigned int Width)
```

Height: Height of the image to be produced.

Width: Width of the image to be produced.

Returns: The white produced image.

Description: Allotcates a new image filled with white pixels of the dimensions Height and Width.

7.2.133 IritImgPrcssDeleteImg (img_prcss.c:273)

delete

free

```
void IritImgPrcssDeleteImg(IritImgPrcssImgStruct *Image)
```

Image: Pointer for the image to free.

Returns: void

Description: Deletes the allocated memory for the given image.

7.2.134 IritImgPrcssDetectEdges (img_prcss.c:586)

convolution
filter
Sharpening

```
IritImgPrcssImgStruct *IritImgPrcssDetectEdges(const IritImgPrcssImgStruct
                                                                    *Image,
                                               int BlurAmount)
```

Image: The image to Detect its edges.

BlurAmount: The amount of times to blur the image before applying the bi Sobel operator.

Returns: The edges in the image.

Description: Given an image, detects the edges in the image by applying gaussian blue BlurAmount times and then running the Bi-Sobel operator on the blurred.

7.2.135 IritImgPrcssGaussianBlur (img_prcss.c:464)

convolution
filter
Gaussian blur

```
IritImgPrcssImgStruct *IritImgPrcssGaussianBlur(const IritImgPrcssImgStruct
                                                                    *Image)
```

Image: The image to apply the convolution on.

Returns: A copy of the image after applying the Gaussian Blur operator.

Description: Given an image, applies the Gaussian Blur filter on it. Equivilant to calling ApplyConvolution3Image with the following filter { 1, 2, 1, 2, 4, 2, 1, 2, 1 }.

7.2.136 IritImgPrcssGaussianBlurMult (img_prcss.c:498)

convolution
filter
blur
gaussian blur

```
IritImgPrcssImgStruct *
                IritImgPrcssGaussianBlurMult(IritImgPrcssImgStruct *Image,
                                             int Amount)
```

Image: The image to apply the convolution on.

Amount: The amount of times to run the Gaussian blur on the given image.

Returns: A copy of the image after applying the Gaussian blur operator Amount times.

Description: Given an image, applies the Gaussian Blur filter on it multiple times. Equivalent to calling ApplyConvolution3Image with the following filter (1 / 16) * {1, 2, 1, 2, 4, 2, 1, 2, 1} multiple times.

7.2.137 IritImgPrcssGetPixelVal (img_prcss.c:176)

get pixel
read pixel

```
int IritImgPrcssGetPixelVal(const IritImgPrcssImgStruct *Image,
                            unsigned int i,
                            unsigned int j,
                            float Color[4])
```

Image: Image struct to read the pixel value from.

i: Column index for the pixel.

j: Row index for the pixel.

Color: Array of floats to save the pixel data in.

Returns: TRUE if successful, FALSE otherwise.

Description: Returns the values of the (j, i) pixel in the provided color array.

7.2.138 IritImgPrcssInvert (img_prcss.c:538)

filter

Invert

```
IritImgPrcssImgStruct *IritImgPrcssInvert(const IritImgPrcssImgStruct *Image)
```

Image: The image to invert.

Returns: A copy of the image after applying the sharpen operator.

Description: Given an image, inverts its colors.

7.2.139 IritImgPrcssReadImg (img_prcss.c:615)

```
IritImgPrcssImgStruct *IritImgPrcssReadImg(const char *Path)
```

Path: The path for the image to load.

Returns: Read image.

Description: Given a path, reads the image in that path and creates an image processing library image struct out of it.

7.2.140 IritImgPrcssSetPixelVal (img_prcss.c:204)

set pixel

write pixel

```
int IritImgPrcssSetPixelVal(IritImgPrcssImgStruct *Image,
                            unsigned int i,
                            unsigned int j,
                            const float color[4])
```

Image: Image struct to write the pixel value to.

i: Column index for the pixel.

j: Row index for the pixel.

color: Array of floats containing the pixel data.

Returns: TRUE if successful, FALSE otherwise.

Description: Puts the given pixel in the (j, i) pixel of the provided image.

7.2.141 IritImgPrcssSharpen (img_prcss.c:375)

Convolution

filter

Sharpe

```
IritImgPrcssImgStruct *IritImgPrcssSharpen(const IritImgPrcssImgStruct *Image)
```

Image: The image to apply the convolution on.

Returns: A copy of the image after applying the sharpen operator.

Description: Given an image, applies the sharpen filter on it. Equivalent to calling ApplyConvolution3Image with the following filter { 0, -1, 0, -1, 5, -1, 0, -1, 0 }.

7.2.142 IritImgPrcssToGrayScale (img_prcss.c:229)

gray scale

```
IritImgPrcssImgStruct *IritImgPrcssToGrayScale(const IritImgPrcssImgStruct
                                                                  *Image)
```

Image: Input color image to convert to gray scale.

Returns: Grayscale image of the provided image.

Description: Creates a gray scale image out of the provided image Gray intensity is calculated according to Y = 0.2126R + 0.7152G + 0.0722B following CIE 1931 (i.e. https://en.wikipedia.org/wiki/Grayscale)

7.2.143 IritImgPrcssWriteImg (img_prcss.c:682)

```
void IritImgPrcssWriteImg(IritImgPrcssImgStruct *Image, const char *SavePath)
```

Image: The image processing library image struct.

SavePath: The path to save the image to

Returns: void

Description: Given a path, write the image to that path
See also: IritImgWriteImg,

7.2.144 IritInformationMsg (irit_inf.c:121)

information messages

```
void IritInformationMsg(const char *Msg)
```

Msg: Error message to print.

Returns: void

Description: Default trap for IRIT programs for irit information. This function just prints the given message.
See also: IritInformationMsgPrintf, IritFatalError, IritWarningMsg, , IritInformationWMsg,

7.2.145 IritInformationMsgPrintf (irit2inf.c:35)

information messages

```
void IritInformationMsgPrintf(const char *va_alist, ...)
```

va_alist: Do "man stdarg".

Returns: void

Description: Default trap for IRIT programs for irit warning errors, printf style.
See also: IritInformationMsg, IritFatalErrorPrintf, IritWarningMsgPrintf,

7.2.146 IritInformationWMsg (irit_inf.c:181)

information messages

```
void IritInformationWMsg(const wchar_t *Msg)
```

Msg: Error message to print.

Returns: void

Description: Default trap for IRIT programs for irit information. This function just prints the given message.
See also: IritInformationMsgPrintf, IritFatalError, IritWarningMsg, IritInformationMsg,

7.2.147 IritIntrvlArithAbs (intrv_arit.c:234)

```
void IritIntrvlArithAbs(const IritIntrvlArithStruct *A,
                        IritIntrvlArithStruct *Result)
```

A: Interval A.

Result: The resulting interval.

Returns: void

Description: Calcualtes the absolute value of an interval.
See also:

7.2.148 IritIntrvlArithAdd (intrv_arit.c:32)

```
void IritIntrvlArithAdd(const IritIntrvlArithStruct *A,
                        const IritIntrvlArithStruct *B,
                        IritIntrvlArithStruct *Result)
```

A: Interval A.

B: Interval B.

Result: The resulting interval.

Returns: void

Description: Adds two intervals.
See also:

7.2.149 IritIntrvlArithDiv (intrv_arit.c:129)

```
void IritIntrvlArithDiv(const IritIntrvlArithStruct *A,
                        const IritIntrvlArithStruct *B,
                        IritIntrvlArithStruct *Result)
```

A: Interval A.

B: Interval B.

Result: The resulting interval.

Returns: void

Description: divides interval A by interval B.
See also:

7.2.150 IritIntrvlArithMult (intrv_arit.c:92)

```
void IritIntrvlArithMult(const IritIntrvlArithStruct *A,
                         const IritIntrvlArithStruct *B,
                         IritIntrvlArithStruct *Result)
```

A: Interval A.

B: Interval B.

Result: The resulting interval.

Returns: void

Description: Multiplies two intervals.
See also:

7.2.151 IritIntrvlArithMultScalar (intrv_arit.c:200)

```
void IritIntrvlArithMultScalar(const IritIntrvlArithStruct *A,
                               IrtRType Val,
                               IritIntrvlArithStruct *Result)
```

A: Interval A.

Val: A scalar value.

Result: The resulting interval.

Returns: void

Description: Multiplies an interval by a scalar value.
See also:

7.2.152 IritIntrvlArithSqrt (intrv_arit.c:174)

```
void IritIntrvlArithSqrt(const IritIntrvlArithStruct *A,
                         IritIntrvlArithStruct *Result)
```

A: Interval A.

Result: The resulting interval.

Returns: void

Description: Calculates the square root of an interval.
See also:

7.2.153 IritIntrvlArithSub (intrv_arit.c:62)

```
void IritIntrvlArithSub(const IritIntrvlArithStruct *A,
                        const IritIntrvlArithStruct *B,
                        IritIntrvlArithStruct *Result)
```

A: Interval A.

B: Interval B.

Result: The resulting interval.

Returns: void

Description: Subtracts interval B from Interval A.
See also:

7.2.154 IritIntrvlArithUnion (intrv_arit.c:275)

```
void IritIntrvlArithUnion(const IritIntrvlArithStruct *A,
                          const IritIntrvlArithStruct *B,
                          IritIntrvlArithStruct *Result)
```

A: Interval A.

B: Interval B.

Result: The resulting interval.

Returns: void

Description: Calcualtes the union of two intervals.
See also:

7.2.155 IritIntrvlArithVAdd (intrv_arit.c:301)

```
void IritIntrvlArithVAdd(const IritIntrvlArithStruct *A,
                         const IritIntrvlArithStruct *B,
                         IritIntrvlArithStruct *Result)
```

A: Interval vector A.

B: Interval vector B.

Result: The resulting interval vector.

Returns: void

Description: Adds two vector(3d) intervals.
See also:

7.2.156 IritIntrvlArithVCross (intrv_arit.c:364)

```
void IritIntrvlArithVCross(const IritIntrvlArithStruct *A,
                           const IritIntrvlArithStruct *B,
                           IritIntrvlArithStruct *Result)
```

A: Interval vector A.

B: Interval vector B.

Result: The resulting interval vector.

Returns: void

Description: The cross product of two vector(3d) intervals.
See also:

7.2.157 IritIntrvlArithVDot (intrv_arit.c:330)

```
void IritIntrvlArithVDot(const IritIntrvlArithStruct *A,
                         const IritIntrvlArithStruct *B,
                         IritIntrvlArithStruct *Result)
```

A: Interval vector A.

B: Interval vector B.

Result: The resulting interval vector.

Returns: void

Description: The dot product of two vector(3d) intervals.
See also:

7.2.158 IritIntrvlArithVFromCone (intrv_arit.c:401)

```
void IritIntrvlArithVFromCone(const IrtRType *Dir,
                              IrtRType Angle,
                              IritIntrvlArithStruct *Result)
```

Dir: Cone direction (axis).

Angle: Cone angle.

Result: The resulting interval vector.

Returns: void

Description: Generates an interval vector from a given cone (apex at origin).
See also:

7.2.159 IritIntrvlArithVMultScalar (intrv_arit.c:579)

```
void IritIntrvlArithVMultScalar(const IritIntrvlArithStruct *A,
                                IrtRType Val,
                                IritIntrvlArithStruct *Result)
```

A: The interval vector.

Val: A scalar value.

Result: The resulting interval vector.

Returns: void

Description: Multiplies an interval vector by a scalar value.
See also:

7.2.160 IritIntrvlArithVToCone (intrv_arit.c:436)

```
void IritIntrvlArithVToCone(const IritIntrvlArithStruct *IntervalV,
                            IrtRType *Dir,
                            IrtRType *Angle)
```

IntervalV: The interval vector.

Dir: Cone direction (axis).

Angle: Cone angle.

Returns: void

Description: Generates a cone (apex at origin) from a given interval vector.
See also:

7.2.161 IritIntrvlArithVToSphere (intrv_arit.c:541)

```
void IritIntrvlArithVToSphere(const IritIntrvlArithStruct *IntervalV,
                              IrtRType *Center,
                              IrtRType *Radius)
```

IntervalV: The interval vector.

Center: sphere center.

Radius: Sphere radius.

Returns: void

Description: Generates a sphere of an average radius around the interval vector.
See also:

7.2.162 IritLevenMarMin (levenmar.c:433)

```
IrtRType IritLevenMarMin(IrtRType **X,
                         IrtRType Y[],
                         IrtRType Sigma[],
                         unsigned NumberOfDataElements,
                         IrtRType ModelParams[],
                         IritLevenEvalFuncType *ShapeFunc,
                         IritLevenNumerProtectionFuncType *ProtectionFunc,
                         IritLevenIsModelValidFuncType *ModelValidatorFunc,
                         unsigned NumberOfModelParams,
                         IrtRType Tolerance)
```

X: Pointer to a list of data.

Y: Pointer to the list of expected results.

Sigma: Pointer to the expected variance vector.

NumberOfDataElements: The number of data elements in X, Y & Sigma.

ModelParams: The model params to be checked.

ShapeFunc: The shape function.

ProtectionFunc: The numerical protection function.

ModelValidatorFunc: The model validator.

NumberOfModelParams: The number of model params.

Tolerance: If the error is smaller then Tolerance return.

Returns: The squared sum of the error.

Description: This function calculates the levenberg-marquardt minimization of the Specified function.
See also: IritLevenMarMinSig1, IritLevenMarSetMaxIterations,

7.2.163 IritLevenMarMinSig1 (levenmar.c:510)

```
IrtRType IritLevenMarMinSig1(IrtRType **X,
                             IrtRType Y[],
                             unsigned NumberOfDataElements,
                             IrtRType ModelParams[],
                             IritLevenEvalFuncType *ShapeFunc,
                             IritLevenNumerProtectionFuncType *ProtectionFunc,
                             IritLevenIsModelValidFuncType *ModelValidatorFunc,
                             unsigned NumberOfModelParams,
                             IrtRType Tolerance)
```

X: Pointer to a list of data.

Y: Pointer to the list of expected results.

NumberOfDataElements: The number of data elements in X, Y & Sigma.

ModelParams: The model params to be checked.

ShapeFunc: The shape function.

ProtectionFunc: The numerical protection function.

ModelValidatorFunc: The model validator.

NumberOfModelParams: The number of model params.

Tolerance: If the error is smaller then Tolerance return.

Returns: The squared sum of the error.

Description: This function calculates the levenberg-marquardt minimization of the Specified function. This function is similar to LevenMin except that sigma is allways 1.
See also: IritLevenMarMin, IritLevenMarSetMaxIterations,

7.2.164 IritLevenMarSetMaxIterations (levenmar.c:88)

```
unsigned IritLevenMarSetMaxIterations(unsigned NewVal)
```

NewVal: The new maximum number of iterations.

Returns: The old maximum number of iterations.

Description: This function modifies the maximum number of iteration when calculating Levenberg-Marquardt.
See also: IritLevenMarMinSig1, IritLevenMarMin,

7.2.165 IritLineHasCntrlChar (irit_inf.c:91)

```
int IritLineHasCntrlChar(const char *Line)
```

Line: Line to examine for control char's existence.

Returns: TRUE if this line holds control chars, FALSE otherwise.

Description: Test if the given line contains control chars.

7.2.166 IritMalloc (imalloc.c:494)

allocation

```
VoidPtr IritMalloc(unsigned Size,
                   const char *ObjType,
                   const char *FileName,
                   int LineNum)
```

Size: Size of block to allocate, in bytes.

ObjType: This variable exists iff "#define DEBUG_IRIT_MALLOC". This holds the object descriptions.

FileName: This variable exists iff "#define DEBUG_IRIT_MALLOC". This holds the file name where the call is invoked from.

LineNum: This variable exists iff "#define DEBUG_IRIT_MALLOC". This holds the line number where the call is invoked from.

Returns: A pointer to the allocated block. A function calling this may assume return value will never be NULL, since no more memory cases are trapped locally.

Description: Routine to allocate dynamic memory for all IRIT program/tool/libraries. All requests for dynamic memory should invoke this function. If the environment variable "IRIT_MALLOC" is set when an IRIT program is executed, the consistency of the dynamic memory is tested on every invokation of this routine. See IritDynMemoryDbgTestAll function for more.
See also: IritFree, IritRealloc, IritDynMemoryDbgCheckMark, , IritDynMemoryDbgInitTest,

7.2.167 IritMallocCheckNULL (imalloc.c:841)

```
VoidPtr IritMallocCheckNULL(unsigned Size)
```

Size: Size to alloc.

Returns: Allocated memory.

Description: A version of malloc that also checks for malloc failure.
See also: IritMalloc,

7.2.168 IritMiscDescribeError (misc_err.c:43)

error handling

```
const char *IritMiscDescribeError(IritMiscFatalErrorType ErrorNum)
```

ErrorNum: Type of the error that was raised.

Returns: A string describing the error type.

Description: Returns a string describing a the given error. Errors can be raised by any member of this misc library as well as other users. Raised error will cause an invocation of IritMiscFatalError function which decides how to handle this error. IritMiscFatalError can for example, invoke this routine with the error type, print the appropriate message and quit the program.

7.2.169 IritMiscFatalError (misc_ftl.c:57)

error handling

```
void IritMiscFatalError(IritMiscFatalErrorType ErrID, char *ErrDesc)
```

ErrID: Error type that was raised.

ErrDesc: Possibly, an additional description on error.

Returns: void

Description: Trap Misc_lib errors right here. Provides a default error handler for the misc library. Gets an error description using IritMiscDescribeError, prints it and exit the program using exit.

7.2.170 IritMiscSetFatalErrorFunc (misc_ftl.c:28)

error handling

```
MiscSetErrorFuncType IritMiscSetFatalErrorFunc(MiscSetErrorFuncType ErrorFunc)
```

ErrorFunc: New error function to use.

Returns: Old error function reference.

Description: Sets the error function to be used by Misc_lib.

7.2.171 IritPQCompFunc (priorque.c:136)

priority queue

```
IritPQCompFuncType IritPQCompFunc(IritPriorQueStruct *PQ,
                                  IritPQCompFuncType NewCompFunc)
```

PQ: The priority queue to update.

NewCompFunc: A comparison function to used on item in the queue.

Returns: Old function used.

Description: Sets (a pointer to) the function that is used in comparing two items in the queue. This comparison function will get two item pointers, and should return >0, 0, <0 as comparison result for greater than, equal, or less than relation, respectively.

7.2.172 IritPQDelete (priorque.c:252)

priority queue

```
VoidPtr IritPQDelete(IritPriorQueStruct *PQ, VoidPtr OldItem)
```

PQ: To delete OldItem from.

OldItem: Old element in priority queue PQ to remove from.

Returns: Removed OldItem if found, NULL otherwise.

Description: Deletes an old item from the queue, using the PQ comparison function. Returns a pointer to Deleted item if was found and deleted from the queue, NULL otherwise.

7.2.173 IritPQEmpty (priorque.c:112)

priority queue

```
int IritPQEmpty(IritPriorQueStruct *PQ)
```

PQ: Priority queue to test for containment.

Returns: TRUE if not empty, FALSE otherwise.

Description: Returns TRUE iff PQ priority queue is empty.

7.2.174 IritPQFind (priorque.c:403)

priority queue

```
VoidPtr IritPQFind(IritPriorQueStruct *PQ, VoidPtr OldItem)
```

PQ: To search for OldItem at.

OldItem: Element to search in PQ.

Returns: Found element or otherwise NULL.

Description: Finds old item on the queue, PQ, using the comparison function CompFunc. Returns a pointer to item if was found, NULL otherwise.

7.2.175 IritPQFindByFunc (priorque.c:350)

priority queue

```
VoidPtr IritPQFindByFunc(IritPriorQueStruct *PQ,
                         IritPQCompFuncType CompFunc,
                         VoidPtr CompItem)
```

PQ: To find OldItem in.

CompFunc: Comparison function to use.

CompItem: Element in priority queue PQ to compare with and find.

Returns: Found OldItem if found, NULL otherwise.

Description: Find an old item from the queue, using a given comparison function CompFunc. Returns a pointer to item if was found, NULL otherwise. Note this search to delete an item is exhaustive and is not optimal.

7.2.176 IritPQFirst (priorque.c:161)

priority queue

```
VoidPtr IritPQFirst(IritPriorQueStruct *PQ, int Delete)
```

PQ: To examine/remove first element from.

Delete: If TRUE first element is being removed from the queue.

Returns: A pointer to the first element in the queue.

Description: Returns the first element in the given priority queue, and delete it from the queue if Delete is TRUE. returns NULL if empty queue.

7.2.177 IritPQFree (priorque.c:587)

priority queue

```
void IritPQFree(IritPriorQueStruct *PQ, int FreeItem)
```

PQ: Priority queue to release.

FreeItem: If TRUE, elements are being freed as well.

Returns: void

Description: Frees the given queue. The elements are also freed if FreeItems is TRUE.

7.2.178 IritPQFreeFunc (priorque.c:625)

priority queue

```
void IritPQFreeFunc(IritPriorQueStruct *PQ, void (*FreeFunc)(VoidPtr))
```

PQ: Priority queue to release.

FreeFunc: "Printing function".

Returns: void

Description: Frees the given queue. The elements are also freed by invoking FreeFunc on all of them as FreeFunc's only argument.

7.2.179 IritPQInit (priorque.c:91)

priority queue

```
void IritPQInit(IritPriorQueStruct **PQ, IritPQCompFuncType CompFunc)
```

PQ: To initialize.

CompFunc: A comparison function to used on item in the queue.

Returns: void

Description: Initializes the priority queue.

7.2.180 IritPQInsert (priorque.c:198)

priority queue

```
VoidPtr IritPQInsert(IritPriorQueStruct *PQ, VoidPtr NewItem)
```

PQ: To insert a new element to.

NewItem: The new element to insert.

Returns: An old element NewItem replaced, or NULL otherwise.

Description: Insert a new element into the queue (NewItem is a pointer to new element) using given compare function CompFunc (See IritPQCompFunc). Insert element will always be a leaf of the constructed tree. Returns a pointer to old element if was replaced or NULL if the element is new.

7.2.181 IritPQNext (priorque.c:459)

priority queue

```
VoidPtr IritPQNext(IritPriorQueStruct *PQ,
                   VoidPtr CmpItem,
                   VoidPtr LargerThan)
```

PQ: To examine.

CmpItem: To find the smallest item in PQ that is larger than it.

LargerThan: The item that is found larger so far.

Returns: The smallest item in PQ that is larger than CmpItem or NULL if no found.

Description: Returns the smallest element in PQ that is larger than given element CmpItem. PQ is not modified. Return NULL if none was found. LargerThan will always hold the smallest Item Larger than current one.

7.2.182 IritPQPrint (priorque.c:551)

priority queue

```
void IritPQPrint(IritPriorQueStruct *PQ, void (*PrintFunc)(VoidPtr))
```

PQ: Priority queue to traverse.

PrintFunc: "Printing function".

Returns: void

Description: Scans the priority queue in order and invokes the "printing" routine, PrintFunc on every item in the queue as its only argument.

7.2.183 IritPQSize (priorque.c:515)

```
int IritPQSize(IritPriorQueStruct *PQ)
```

PQ: Priority queue to traverse.

Returns: Number of nodes in tree == number of elements.

Description: Computes the size of the given tree - number of nodes/elements.

7.2.184 IritPseudoRandom (xgeneral.c:403)

```
IrtRType IritPseudoRandom()
```

Returns: A pseudo random number between zero and one.

Description: Computes a pseudo random number generator using the large Bezier evaluation table of the cagd library...

See also: CagdIChooseKTable, IritPseudoRandomInit, IritRandom,

7.2.185 IritPseudoRandomInit (xgeneral.c:381)

```
void IritPseudoRandomInit(unsigned int Seed)
```

Seed: New seed for the randomization process.

Returns: void

Description: A seed generator for a pseudo random number generator using the large Bezier evaluation table of the cagd library...

See also: CagdIChooseKTable, IritPseudoRandom, IritRandom,

7.2.186 IritQRFactorization (qrfactor.c:52)

QR factorization
linear systems
matrices

```
int IritQRFactorization(IrtRType *A,
                        int n,
                        int m,
                        IrtRType *Q,
                        IrtRType *R)
```

A: The matrix of size n by m (m <= n), must be preallocated dynamically by the user.

n, m: Dimensions of matrix A.

Q, R: The computed decomposition matrices, must be preallocated dynamically by the user. Q is n by m (like A), R is m by m.

Returns: TRUE if Singular, FALSE otherwise.

Description: Performs a QR factorization of matrix A.
See also: SvdLeastSqr,

7.2.187 IritQRUnderdetermined (qrfactor.c:285)

QR factorization
linear systems
matrices

```
int IritQRUnderdetermined(IrtRType *A,
                          IrtRType *x,
                          const IrtRType *b,
                          int m,
                          int n,
                          void **QRCache)
```

A: The matrix of size m by n (m <= n), must be preallocated dynamically by the user.

x: The solution vector of size n.

b: A vector of size m.

m, n: Dimensions of matrix A. Because A is under-determined m <= n.

QRCache: A reference to a cache used internally.

Returns: TRUE if Singular, FALSE otherwise.

Description: Solve for Ax = b when the system is under-determined (singular). The solution is a minimum 2-norm solution. See "matrix Computation", by Gene H. Golub and Charles F. Van Loan, 3rd edition, pp 271-272. Caches the QR factorizations so we can solve multiples b's as follows:

1. If A != NULL a QR factorization is computed for A and cached.
2. if A == NULL and X != NULL, a solution is computed for the given b.
3. If A == NULL and x == NULL the cache is freed.

See also: IritQRFactorization, IritSolveUpperDiagMatrix, SvdLeastSqr,

7.2.188 IritQRUnderdetermined2 (qrfactor.c:377)

QR factorization
linear systems
matrices

```
int IritQRUnderdetermined2(IrtRType *A,
                           IrtRType *x,
                           const IrtRType *b,
                           int m,
                           int n)
```

A: The matrix of size m by n (m <= n), must be preallocated dynamically by the user.

x: The solution vector of size n.

b: A vector of size m.

m, n: Dimensions of matrix A. Because A is under-determined m <= n.

Returns: TRUE if Singular, FALSE otherwise.

Description: Solve for Ax = b when the system is under-determined (singular). The solution is a minimum 2-norm solution. See "matrix Computation", by Gene H. Golub and Charles F. Van Loan, 3rd edition, pp 271-272. Same as IritQRUnderdetermined but handles one solution vector.
See also: IritQRFactorization, IritSolveUpperDiagMatrix, SvdLeastSqr, , IritQRUnderdetermined,

7.2.189 IritRLAdd (mincover.c:279)

```
void IritRLAdd(VoidPtr RLC, IrtRType l, IrtRType r, int Attr)
```

RLC: An existing list which will be added a new node.

l, r, Attr: Node details; Left, Right and Attribute.

Returns: void

Description: Creates and links a new range node to an existing list of nodes.
See also: IritRLNew, IritRLFindCyclicCover, IritRLDelete,

7.2.190 IritRLDelete (mincover.c:233)

```
void IritRLDelete(VoidPtr RLC)
```

RLC: A list to be deleted.

Returns: void

Description: Cleanup of RLStruct (a whole list)
See also: IritRLNew, IritRLAdd, IritRLFindCyclicCover,

7.2.191 IritRLFindCyclicCover (mincover.c:1103)

```
int *IritRLFindCyclicCover(VoidPtr RLC, IrtRType Tol)
```

RLC: A list (RL) of candidate ranges to cover [0,1].

Tol: Accuracy of merges of sets.

Returns: A vector of indices (attributes) of covering set, terminated by -1.

Description: Finds a cyclic cover from nodes in RL to cover [0,1] This function decides the type of the cover SignleRange or MultiRange and calls the suitable internal function to find a cover.
See also: IritRLNew, IritRLAdd, IritRLDelete,

7.2.192 IritRLNew (mincover.c:194)

```
VoidPtr IritRLNew(void)
```

Returns: The new allocated list structure.

Description: Initializes a new Range List (RL) - which is an empty double link list The list specifier is Plist*, which points, always, at the tail.
See also: IritRLAdd, IritRLFindCyclicCover, IritRLDelete,

7.2.193 IritRLSetGaurdiansNumber (mincover.c:109)

```
int IritRLSetGaurdiansNumber(int g)
```

g: Maximum number of gaurdians required.

Returns: The previously used number of gaurdians.

Description: Sets the maximum number of views allowed to generate a min cover. Actually this is the maximum number of subsets to be generated by the FindMRCyclicCover() function.
See also: IritRLAdd, IritRLFindCyclicCover, IritRLDelete,

7.2.194 IritRandom (xgeneral.c:339)

random numbers

```
IrtRType IritRandom(IrtRType Min, IrtRType Max)
```

Min: Minimum range of random number requested.

Max: Maximum range of random number requested.

Returns: A random number between Min andMax.

Description: Routine to compute a random number in a specified range. See also IritRandomInit.
See also: IritRandomInit, IritPseudoRandom,

7.2.195 IritRandomInit (xgeneral.c:304)

random numbers

```
void IritRandomInit(long Seed)
```

Seed: To initialize the random number generator with.

Returns: void

Description: Routine to initialize the random number generator.
See also: IritRandom, IritPseudoRandom,

7.2.196 IritRealTimeDate (xgeneral.c:524)

date

time

```
char *IritRealTimeDate(char *StrTime)
```

StrTime: Where to save the written date/time. Same as return value.

Returns: A string describing current date and time. Same as StrTime.

Description: Routine to create and return a string describing current date and time.

7.2.197 IritRealloc (imalloc.c:447)

allocation

```
VoidPtr IritRealloc(VoidPtr p, unsigned OldSize, unsigned NewSize)
```

p: Old pointer to reallocate. Freed at the end of this routine.

OldSize: Size of old block pointed by p, zero if unknown.

NewSize: Size of new block to allocate, in bytes. Must be larger than OldSize.

Returns: A pointer to the allocated block. A function calling this may assume return value will never be NULL, since no more memory cases are trapped locally.

Description: Routine to reallocate dynamic memory for all IRIT program/tool/libraries. All requests for dynamic memory should invoke this function. If the environment variable "IRIT_MALLOC" is set when an IRIT program is executed, the consistency of the dynamic memory is tested on every invokation of this routine. See IritDynMemoryDbgTestAll function for more.
See also: IritFree, IritMalloc, IritDynMemoryDbgCheckMark, , IritDynMemoryDbgInitTest,

7.2.198 IritSearch2DFindElem (search.c:210)

```
int IritSearch2DFindElem(VoidPtr S2D,
                         IrtRType XKey,
                         IrtRType YKey,
                         VoidPtr Data)
```

S2D: Internal data structure, aiding search.

XKey, YKey: The 2D coordinates to search for.

Data: A pointer to the location to copy the found data into.

Returns: TRUE if element was found, FALSE otherwise.

Description: Looks for an existing element in the data structure. If found an element within prescribed tolerance, Data is updated with the saved data
See also: IritSearch2DFree, IritSearch2DInsertElem, IritSearch2DInit,

7.2.199 IritSearch2DFree (search.c:119)

```
void IritSearch2DFree(VoidPtr S2D)
```

S2D: Internal data structure, aiding search, to be freed.

Returns: void

Description: Free auxiliary data structure aiding in 2D search.
See also: IritSearch2DInsertElem, IritSearch2DFindElem, IritSearch2DInit,

7.2.200 IritSearch2DInit (search.c:58)

```
VoidPtr IritSearch2DInit(IrtRType XMin,
                         IrtRType XMax,
                         IrtRType YMin,
                         IrtRType YMax,
                         IrtRType Tol,
                         int DataSize)
```

XMin, XMax, YMin, YMax: Dimensions of 2D domain.

Tol: Tolerance of expected search.

DataSize: Size, in bytes, of expected data elements to keep.

Returns: An internal auxiliary structure to be provided to the insertion and searching routines, NULL if error.

Description: Initialize the search data structure.
See also: IritSearch2DInsertElem, IritSearch2DFindElem, IritSearch2DFree,

7.2.201 IritSearch2DInsertElem (search.c:167)

```
void IritSearch2DInsertElem(VoidPtr S2D,
                            IrtRType XKey,
                            IrtRType YKey,
                            VoidPtr Data)
```

S2D: Internal data structure, aiding search.

XKey, YKey: The new 2D coordinates to insert.

Data: A pointer to the data to save with this 2D coordinate key.

Returns: void

Description: Insert a new element into the data structure. No test is made if a similar element already exists.
See also: IritSearch2DFree, IritSearch2DFindElem, IritSearch2DInit,

7.2.202 IritSetFatalErrorFunc (irit_ftl.c:30)

error handling

```
IritFatalMsgFuncType IritSetFatalErrorFunc(IritFatalMsgFuncType FatalMsgFunc)
```

FatalMsgFunc: New function to use.

Returns: Old function reference.

Description: Sets the warning function of irit.

7.2.203 IritSetInfoMsgFunc (irit_inf.c:36)

error handling

```
IritInfoMsgFuncType IritSetInfoMsgFunc(IritInfoMsgFuncType InfoMsgFunc)
```

InfoMsgFunc: New function to use.

Returns: Old function reference.

Description: Sets the warning function of irit.
See also: IritSetInfoWMsgFunc,

7.2.204 IritSetInfoWMsgFunc (irit_inf.c:65)

error handling

```
IritInfoWMsgFuncType IritSetInfoWMsgFunc(IritInfoWMsgFuncType InfoWMsgFunc)
```

InfoWMsgFunc: New function to use.

Returns: Old function reference.

Description: Sets the warning function of irit (for wide charachers).
See also: IritSetInfoMsgFunc,

7.2.205 IritSetIritParallelExec (xgeneral.c:938)

parallel computation

```
int IritSetIritParallelExec(int Prll)
```

Prll: If not zero enables parallel computation, while a positive value controls the maximal number of threads.

Returns: old value of Prll flag.

Description: Controls parallel execution - enabled and maximal number of threads.

7.2.206 IritSetWarningMsgFunc (irit_wrn.c:30)

error handling

```
IritWarningMsgFuncType IritSetWarningMsgFunc(IritWarningMsgFuncType WrnMsgFunc)
```

WrnMsgFunc: New function to use.

Returns: Old function reference.

Description: Sets the warning function of irit.

7.2.207 IritSleep (xgeneral.c:158)

sleep

```
void IritSleep(int MilliSeconds)
```

MilliSeconds: Sleeping time required, in miliseconds.

Returns: void

Description: Routine to force a process to sleep.

7.2.208 IritSolveLowerDiagMatrix (qrfactor.c:232)

QR factorization
linear systems
matrices

```
int IritSolveLowerDiagMatrix(const IrtRType *A,
                             int n,
                             const IrtRType *b,
                             IrtRType *x)
```

A: The diagonal matrix of size n by n, must be preallocated dynamically by the user.

n: Dimension of matrix A.

b: A vector of size n.

x: The solution vector of size n.

Returns: TRUE if Singular, FALSE otheriwse.

Description: Solve for Ax = b when R is lower diagonal using forward substitution.
See also: IritSolveUpperDiagMatrix, IritQRFactorization, IritQRUnderdetermined, , SvdLeastSqr,

7.2.209 IritSolveUpperDiagMatrix (qrfactor.c:188)

QR factorization
linear systems
matrices

```
int IritSolveUpperDiagMatrix(const IrtRType *A,
                             int n,
                             const IrtRType *b,
                             IrtRType *x)
```

A: The diagonal matrix of size n by n, must be preallocated dynamically by the user.

n: Dimension of matrix A.

b: A vector of size n.

x: The solution vector of size n.

Returns: TRUE if Singular, FALSE otheriwse.

Description: Solve for Ax = b when R is upper diagonal using back substitution.
See also: IritSolveLowerDiagMatrix, IritQRFactorization, IritQRUnderdetermined, , SvdLeastSqr,

7.2.210 IritStrIStr (xgeneral.c:697)

```
const char *IritStrIStr(const char *s, const char *Pattern)
```

s: To search for Pattern in.

Pattern: To search in s.

Returns: Address in s where Pattern was 1st found, NULL otherwise.

Description: Routine to search for a Pattern (no regular expression) in s. Returns address in s of first occurrence of Pattern, NULL if non found. Case insensitive.

7.2.211 IritStrLower (xgeneral.c:118)

lower case
upper case

```
char *IritStrLower(char *s)
```

s: String to convert to lower case.

Returns: Reference to s.

Description: Routine to convert all upper case chars into lower case, in place.

7.2.212 IritStrUpper (xgeneral.c:94)

upper case
lower case

```
char *IritStrUpper(char *s)
```

s: String to convert to upper case.

Returns: Reference to s.

Description: Routine to convert all lower case chars into upper case, in place.

7.2.213 IritStrdup (xgeneral.c:69)

strdup

```
char *IritStrdup(const char *s)
```

s: String to duplicate.

Returns: Duplicated string.

Description: Routine to duplicate a string. Exists in some computer environments.

7.2.214 IritSubstStr (xgeneral.c:733)

```
char *IritSubstStr(const char *s,
                   const char *Src,
                   const char *Dst,
                   int CaseInsensitive)
```

s: Input string.

Src: Pattern to look for in S and substitute with Dst.

Dst: Replacement patter for Src.

CaseInsensitive: TRUE for case insensitive, FALSE for case sensitive.

Returns: new string, allocated dynamically.

Description: A find (Src) and replace (into Dst) function.

7.2.215 IritTextLocaleInit (config.c:730)

```
const char *IritTextLocaleInit(void)
```

Returns: The locale initialized ot NULL if failed.

Description: Initializes the text locale to be used.
See also: IritWChar2Ascii, IritAscii2WChar,

7.2.216 IritWChar2Ascii (config.c:786)

```
char *IritWChar2Ascii(const wchar_t *Str)
```

Str: Input wide (multi-short) string to convert to a multi-byte Ascii string.

Returns: Converted string, allocated dynamically, or NULL if error.

Description: Converts a wide (multi-short) string to an Ascii (multi-byte) string.
See also: IritDspAscii2WChar, IritTextLocaleInit,

7.2.217 IritWarningMsg (irit_wrn.c:60)

warning trap

error trap

```
void IritWarningMsg(const char *Msg)
```

Msg: Error message to print.

Returns: void

Description: Default trap for IRIT programs for irit warnings. This function just prints the given warning message.
See also: IritWarningMsgPrintf, IritFatalError,

7.2.218 IritWarningMsgPrintf (irit2wrn.c:35)

warning trap

error trap

```
void IritWarningMsgPrintf(const char *va_alist, ...)
```

va_alist: Do "man stdarg".

Returns: void

Description: Default trap for IRIT programs for irit warning errors, printf style.
See also: IritWarningMsg, IritFatalErrorPrintf,

7.2.219 IrtDitherBurkes (dither2.c:427)

Dithering
Dither
Error Diffusion
Burkes

```
IrtImgRGBAPxlStruct *IrtDitherBurkes(IrtImgRGBAPxlStruct *Image,
                                     int XSize,
                                     int YSize,
                                     int NumColors,
                                     const IrtImgRGBAPxlStruct *DitherColors)
```

Image: The image to diffuse.

XSize: The width of the image.

YSize: The height of the image

NumColors: The number of colors in the palette.

DitherColors: The color palette to dither into.

Returns: The dithered output image.

Description: A function implementing serpentine Burkes error diffusion algorithm. PAPER: Presentation of the Burkes error filter for use in preparing continuous-tone images for presentation on bi-level devices (Burkes, 1988)
See also: IrtDitherStucki, IrtDitherFloydSteinberg, IrtDitherJarvisJudiceNinke, , IrtDitherSierraFirst, IrtDitherSierraSecond, , IrtDitherSierraThird, IrtDitherSerpentineSimpleAlgorithm, , IrtImgDitherImageClr,

7.2.220 IrtDitherFloydSteinberg (dither2.c:343)

Dithering
Dither
Error Diffusion
Floyd-Steinberg

```
IrtImgRGBAPxlStruct *IrtDitherFloydSteinberg(
                                     IrtImgRGBAPxlStruct *Image,
                                     int XSize,
                                     int YSize,
                                     int NumColors,
                                     const IrtImgRGBAPxlStruct *DitherColors)
```

Image: The image to diffuse.

XSize: The width of the image.

YSize: The height of the image

NumColors: The number of colors in the palette.

DitherColors: The color palette to dither into.

Returns: The dithered output image.

Description: A function implementing serpentine FloydSteinberg error diffusion algorithm. PAPER: An adaptive algorithm for spatial grey scale (Floyd & Steinberg, 1976)
See also: IrtDitherStucki, IrtDitherJarvisJudiceNinke, , IrtDitherBurkes, IrtDitherSierraFirst, IrtDitherSierraSecond, , IrtDitherSierraThird, IrtDitherSerpentineSimpleAlgorithm, , IrtImgDitherImageClr,

7.2.221 IrtDitherJarvisJudiceNinke (dither2.c:385)

Dithering
Dither
Error Diffusion
Jarvis-Judice-Ninke

```
IrtImgRGBAPxlStruct *IrtDitherJarvisJudiceNinke(
                                     IrtImgRGBAPxlStruct *Image,
                                     int XSize,
                                     int YSize,
                                     int NumColors,
                                     const IrtImgRGBAPxlStruct *DitherColors)
```

Image: The image to diffuse.

XSize: The width of the image.

YSize: The height of the image

NumColors: The number of colors in the palette.

DitherColors: The color palette to dither into.

Returns: The dithered output image.

Description: A function implementing serpentine Jarvis-Judice-Ninke ED algorithm. PAPER: A Survey of Techniques for the Display of Continuous Tone Pictures on Bi-level Displays (Jarvis, Judice and Ninke, 1976)
See also: IrtDitherStucki, IrtDitherFloydSteinberg, , IrtDitherBurkes, IrtDitherSierraFirst, IrtDitherSierraSecond, , IrtDitherSierraThird, IrtDitherSerpentineSimpleAlgorithm, , IrtImgDitherImageClr,

7.2.222 IrtDitherSierraFirst (dither2.c:467)

Dithering
Dither
Error Diffusion
Sierra

```
IrtImgRGBAPxlStruct *IrtDitherSierraFirst(
                                    IrtImgRGBAPxlStruct *Image,
                                    int XSize,
                                    int YSize,
                                    int NumColors,
                                    const IrtImgRGBAPxlStruct *DitherColors)
```

Image: The image to diffuse.

XSize: The width of the image.

YSize: The height of the image

NumColors: The number of colors in the palette.

DitherColors: The color palette to dither into.

Returns: The dithered output image.

Description: A function implementing serpentine Sierra First error diffusion algorithm. PAPER: Frankie Sierra, in LIB 17 (Developer's Den), CIS Graphics Support Forum, 1989.

See also: IrtDitherStucki, IrtDitherFloydSteinberg, IrtDitherJarvisJudiceNinke, , IrtDitherBurkes, IrtDitherSierraSecond, IrtDitherSierraThird, , IrtDitherSerpentineSimpleAlgorithm, IrtImgDitherImageClr,

7.2.223 IrtDitherSierraSecond (dither2.c:509)

Dithering
Dither
Error Diffusion
Sierra

```
IrtImgRGBAPxlStruct *IrtDitherSierraSecond(
                                    IrtImgRGBAPxlStruct *Image,
                                    int XSize,
                                    int YSize,
                                    int NumColors,
                                    const IrtImgRGBAPxlStruct *DitherColors)
```

Image: The image to diffuse.

XSize: The width of the image.

YSize: The height of the image

NumColors: The number of colors in the palette.

DitherColors: The color palette to dither into.

Returns: The dithered output image.

Description: A function implementing serpentine Sierra Second error diffusion algorithm. PAPER: Frankie Sierra, in LIB 17 (Developer's Den), CIS Graphics Support Forum, 1989.

See also: IrtDitherStucki, IrtDitherFloydSteinberg, IrtDitherJarvisJudiceNinke, , IrtDitherBurkes, IrtDitherSierraFirst, IrtDitherSierraThird, , IrtDitherSerpentineSimpleAlgorithm, IrtImgDitherImageClr,

7.2.224 IrtDitherSierraThird (dither2.c:549)

Dithering
Dither
Error Diffusion
Sierra

```
IrtImgRGBAPxlStruct *IrtDitherSierraThird(
                                    IrtImgRGBAPxlStruct *Image,
                                    int XSize,
                                    int YSize,
                                    int NumColors,
                                    const IrtImgRGBAPxlStruct *DitherColors)
```

Image: The image to diffuse.

XSize: The width of the image.

YSize: The height of the image

NumColors: The number of colors in the palette.

DitherColors: The color palette to dither into.

Returns: The dithered output image.

Description: A function implementing serpentine Sierra Third error diffusion algorithm. PAPER: Frankie Sierra, in LIB 17 (Developer's Den), CIS Graphics Support Forum, 1989

See also: IrtDitherStucki, IrtDitherFloydSteinberg, IrtDitherJarvisJudiceNinke, , IrtDitherBurkes, IrtDitherSierraFirst, IrtDitherSierraSecond, , IrtDitherSerpentineSimpleAlgorithm, IrtImgDitherImageClr,

7.2.225 IrtDitherStucki (dither2.c:300)

Dithering
Dither
Error Diffusion
Stucki.

```
IrtImgRGBAPxlStruct *IrtDitherStucki(IrtImgRGBAPxlStruct *Image,
                                     int XSize,
                                     int YSize,
                                     int NumColors,
                                     const IrtImgRGBAPxlStruct *DitherColors)
```

Image: The image to diffuse.

XSize: The width of the image.

YSize: The height of the image

NumColors: The number of colors in the palette.

DitherColors: The color palette to dither into.

Returns: The dithered output image.

Description: A function implementing serpentine Stucki error diffusion algorithm. PAPER: Mecca-a multiple-error correcting computation algorithm for bilevel image hardcopy reproduction (Stucki, 1981)
See also: IrtDitherFloydSteinberg, IrtDitherJarvisJudiceNinke, , IrtDitherBurkes, IrtDitherSierraFirst, IrtDitherSierraSecond, , IrtDitherSierraThird, IrtDitherSerpentineSimpleAlgorithm, , IrtImgDitherImageClr,

7.2.226 IrtImgCnvrtRGB2RGBA (writimag.c:406)

```
IrtImgRGBAPxlStruct *IrtImgCnvrtRGB2RGBA(const IrtImgPixelStruct *RGBImg,
                                         int XSize,
                                         int YSize)
```

RGBImg: Input RGB image.

XSize, YSize: Dimensions of input RGB image.

Returns: Mapped image into RGBA space.

Description: Maps an RGB image to RGBA image with a full alpha coverage.
See also: IrtImgCnvrtRGBA2RGB,

7.2.227 IrtImgCnvrtRGBA2RGB (writimag.c:446)

```
IrtImgPixelStruct *IrtImgCnvrtRGBA2RGB(const IrtImgRGBAPxlStruct *RGBAImg,
                                       int XSize,
                                       int YSize)
```

RGBAImg: Input RGBA image.

XSize, YSize: Dimensions of input RGB image.

Returns: Mapped image into RGB space.

Description: Maps an RGBA image to RGB image with no alpha coverage.
See also: IrtImgCnvrtRGB2RGBA,

7.2.228 IrtImgDitherImage (dither.c:522)

```
IrtImgRGBAPxlStruct *IrtImgDitherImage(IrtImgRGBAPxlStruct *Image,
                                       int XSize,
                                       int YSize,
                                       int DitherSize,
                                       IrtBType ErrorDiffusion,
                                       int NumColors,
                                       const IrtImgRGBAPxlStruct *DitherColors)
```

Image: To dither.

XSize, YSize: Size of the image to dither.

DitherSize: Dithering matrices size: 2, 3, or 4.

ErrorDiffusion: TRUE, to also diffuse the error in the image.

NumColors: Optional set of colors (in DitherColors) to color dither the image. 0 to ignore (and do B&W dithering).

DitherColors: Optional set of NumDitherColors colors to color-dither. NULL to ignore (and do B&W dithering).

Returns: Dithered image.

Description: A wrapper routine to read an image from a file, dither it and save into a file the dithered image.
See also: IrtImgDitherImage2, IrtImgDitherImageBW, , IrtImgDitherImageClr, IrtImgDitherImageBW2,

7.2.229 IrtImgDitherImage2 (dither.c:569)

```
int IrtImgDitherImage2(const char *InputImage,
                       const char *OutputImage,
                       int DitherSize,
                       IrtBType ErrorDiffusion,
                       int NumColors,
                       const IrtImgRGBAPxlStruct *DitherColors)
```

InputImage: To dither.

OutputImage: Dithered image.

DitherSize: Dithering matrices size: 2, 3, or 4.

ErrorDiffusion: TRUE, to also diffuse the error in the image.

NumColors: Optional set of colors (in DitherColors) to color dither the image. 0 to ignore (and do B&W dithering).

DitherColors: Optional set of NumDitherColors colors to color-dither. NULL to ignore (and do B&W dithering).

Returns: TRUE if successful, FALSE if failed.

Description: A wrapper routine to read an image from a file, dither it and save into a file the dithered image.
See also: IrtImgDitherImage, IrtImgDitherImageBW, IrtImgDitherImageBW2, , IrtImgDitherImageClr,

7.2.230 IrtImgDitherImage3 (dither2.c:653)

```
int IrtImgDitherImage3(const char *InputImage,
                       const char *OutputImage,
                       int NumColors,
                       const IrtImgRGBAPxlStruct *DitherColors,
                       IrtImgImageDitherType DitherMethod)
```

InputImage: To dither.

OutputImage: Dithered image.

NumColors: Optional set of colors (in DitherColors) to color dither the image. 0 to ignore (and do B&W dithering).

DitherColors: Optional set of NumDitherColors colors to color-dither. NULL to ignore (and do B&W dithering).

DitherMethod: Dithering matrix to use.

Returns: TRUE if successful, FALSE if failed.

Description: A wrapper routine to read an image from a file, dither it and save into a file the dithered image.
See also: IrtImgDitherImage, IrtImgDitherImageBW, IrtImgDitherImageBW2, , IrtImgDitherImageClr, IrtImgDitherImageClr3,

7.2.231 IrtImgDitherImageBW (dither.c:181)

```
IrtBType *IrtImgDitherImageBW(IrtImgRGBAPxlStruct *Image,
                              int XSize,
                              int YSize,
                              int DitherSize,
                              IrtBType ErrorDiffusion)
```

Image: To dither.

XSize, YSize: Size of the image to dither.

DitherSize: Dithering matrices size: 2, 3, or 4.

ErrorDiffusion: TRUE, to also diffuse the error in the image.

Returns: The dithered B&W image of size X/YSize * DitherSize.

Description: Routine to dither a given Image of RGBA pixels using dithering matrices of size DitherSize to B&W. Alpha is duplicated.
See also: IrtImgDitherImage, IrtImgDitherImage2, , IrtImgDitherImageClr, IrtImgDitherImageBW1, IrtImgDitherImageBW2,

7.2.232 IrtImgDitherImageBW1 (dither.c:251)

```
IrtBType *IrtImgDitherImageBW1(IrtImgPixelStruct *Image,
                               int XSize,
                               int YSize,
                               int DitherSize,
                               IrtBType ErrorDiffusion)
```

Image: To dither.

XSize, YSize: Size of the image to dither.

DitherSize: Dithering matrices size: 2, 3, or 4.

ErrorDiffusion: TRUE, to also diffuse the error in the image.

Returns: The dithered B&W image of size X/YSize * DitherSize.

Description: Routine to dither a given Image of RGB pixels using dithering matrices of size DitherSize to B&W.
See also: IrtImgDitherImage, IrtImgDitherImage2, IrtImgDitherImageBW, IrtImgDitherImageBW2, IrtImgDitherImageClr,

7.2.233 IrtImgDitherImageBW2 (dither.c:290)

```
IrtImgRGBAPxlStruct *IrtImgDitherImageBW2(IrtImgRGBAPxlStruct *Image,
                                          int XSize,
                                          int YSize,
                                          int DitherSize,
                                          IrtBType ErrorDiffusion)
```

Image: To dither.

XSize, YSize: Size of the image to dither.

DitherSize: Dithering matrices size: 2, 3, or 4.

ErrorDiffusion: TRUE, to also diffuse the error in the image.

Returns: Dithered B&W image of size X/YSize * DitherSize.

Description: Routine to dither a given Image of RGB pixels using dithering matrices of size DitherSize to B&W.
See also: IrtImgDitherImage, IrtImgDitherImage2, , IrtImgDitherImageClr, IrtImgDitherImageBW, IrtImgDitherImageBW1,

7.2.234 IrtImgDitherImageClr (dither.c:460)

```
IrtImgRGBAPxlStruct *IrtImgDitherImageClr(
                                        IrtImgRGBAPxlStruct *Image,
                                        int XSize,
                                        int YSize,
                                        int DitherSize,
                                        IrtBType ErrorDiffusion,
                                        int NumColors,
                                        const IrtImgRGBAPxlStruct *DitherColors)
```

Image: To dither.

XSize, YSize: Size of the image to dither.

DitherSize: Dithering matrices size: 2, 3, or 4.

ErrorDiffusion: TRUE, to also diffuse the error in the image.

NumColors: Optional set of colors (in DitherColors) to color dither the image.

DitherColors: Optional set of NumDitherColors colors to color-dither.

Returns: The dithered colored image of size X/YSize * DitherSize.

Description: Routine to dither a given Image of RGBA pixels using dithering matrices of size DitherSize to colors. Alpha is duplicated.

See also: IrtImgDitherImage, IrtImgDitherImage2, , IrtImgDitherImageBW, IrtImgDitherImageBW2,

7.2.235 IrtImgDitherImageClr3 (dither2.c:589)

Dithering

Dither

Error Diffusion

```
IrtImgRGBAPxlStruct *IrtImgDitherImageClr3(
                                        IrtImgRGBAPxlStruct *Image,
                                        int XSize,
                                        int YSize,
                                        int NumColors,
                                        const IrtImgRGBAPxlStruct *DitherColors,
                                        IrtImgImageDitherType DitherMethod)
```

Image: The image to diffuse.

XSize: The width of the image.

YSize: The height of the image

NumColors: The number of colors in the palette.

DitherColors: The color palette to dither into.

DitherMethod: Dithering matrix to use.

Returns: The dithered output image.

Description: A function implementing the chosen error diffusion algorithm, I chose to use Stucki algorithm.

See also: IrtDitherStucki, IrtDitherFloydSteinberg, IrtDitherJarvisJudiceNinke, , IrtDitherBurkes, IrtDitherSierraFirst, IrtDitherSierraSecond, , IrtDitherSierraThird, IritDitherSerpentineSimpleAlgorithm, , IrtImgDitherImage3,

7.2.236 IrtImgDitherImageClrBatch (dither2.c:715)

Dither

Batch

Parallel

Layered Dithering.

```
void IrtImgDitherImageClrBatch(const char **ImagePaths,
                               int NumImages,
                               const char *OutputFolder,
                               int NumColors,
                               const IrtImgRGBAPxlStruct *DitherColors,
                               IrtImgImageDitherType DitherMethod)
```

ImagePaths: an array of the image file paths that we want to dither.

NumImages: The number of the images we want to dither.

OutputFolder: The folder path to put the output images in.

NumColors: The number of colors in the palette.

DitherColors: The color palette to dither into.

DitherMethod: The dithering function to use.

Returns: void

Description: Applies dithering to a given set of images, in parallel. We assume all parameters to be valid, and the files and folders to exist. We also assume the images to have 4 channels per pixel.
See also: IrtImgDitherImageClr.,

7.2.237 IrtImgFlipHorizontallyImage (editimag.c:133)

```
IrtImgPixelStruct *IrtImgFlipHorizontallyImage(const IrtImgPixelStruct *Img,
                                               int MaxX,
                                               int MaxY,
                                               int Alpha)
```

Img: Image to flip its X and Y axes.

MaxX: Maximum X of Img image.

MaxY: Maximum Y of Img image.

Alpha: TRUE if this image has alpha and is actually RGBARGBA...

Returns: Flipped image.

Description: Flip the given images horizontally.
See also: IrtImgNegateImage, IrtImgScaleImage,

7.2.238 IrtImgFlipVerticallyImage (editimag.c:191)

```
IrtImgPixelStruct *IrtImgFlipVerticallyImage(const IrtImgPixelStruct *Img,
                                             int MaxX,
                                             int MaxY,
                                             int Alpha)
```

Img: Image to flip its X and Y axes.

MaxX: Maximum X of Img image.

MaxY: Maximum Y of Img image.

Alpha: TRUE if this image has alpha and is actually RGBARGBA...

Returns: Flipped image.

Description: Flip the given images Verticaally.
See also: IrtImgNegateImage, IrtImgScaleImage,

7.2.239 IrtImgFlipXYImage (editimag.c:36)

```
IrtImgPixelStruct *IrtImgFlipXYImage(const IrtImgPixelStruct *Img,
                                     int MaxX,
                                     int MaxY,
                                     int Alpha)
```

Img: Image to flip its X and Y axes.

MaxX: Maximum X of Img image.

MaxY: Maximum Y of Img image.

Alpha: TRUE if this image has alpha and is actually RGBARGBA...

Returns: Flipped image.

Description: Reads one image in from a file named ImageFileName. The image is returned as a vector of RGBRGB... of size (MaxX+1) * (MaxY+1) * 3.
See also: IrtImgNegateImage, IrtImgScaleImage,

7.2.240 IrtImgGetColorsInImage (editimag.c:251)

```
IrtImgPixelStruct *IrtImgGetColorsInImage(const IrtImgPixelStruct *Img,
                                          int MaxX,
                                          int MaxY,
                                          int Alpha,
                                          int *NumClrs)
```

Img: Image to flip its X and Y axes.

MaxX: Maximum X of Img image.

MaxY: Maximum Y of Img image.

Alpha: TRUE if this image has alpha and is actually RGBARGBA...

NumClrs: Will be set with the number off different colors found.

Returns: The NumClrs colors, allocated dynamically.

Description: Fetches all the different colors in the given image.
See also: IrtImgNegateImage, IrtImgScaleImage, IrtImgFlipVerticallyImage,

7.2.241 IrtImgGetImageSize (readimag.c:447)

```
int IrtImgGetImageSize(const char *ImageFileName,
                       int *Width,
                       int *Height)
```

ImageFileName: Name of image to read.

Width: The width of the image.

Height: The height of the image.

Returns: TRUE, if loaded successfully. Otherwise FALSE.

Description: Reads the dimensions of one image in from a file named ImageFileName.

7.2.242 IrtImgNegateImage (editimag.c:92)

```
IrtImgPixelStruct *IrtImgNegateImage(const IrtImgPixelStruct *InImage,
                                     int MaxX,
                                     int MaxY)
```

InImage: A vector of IrtImgPixelStruct of size (MaxX+1) * (MaxY+1).

MaxX: Maximum X of input image.

MaxY: Maximum Y of input image.

Returns: The scaled image as vector of RGBRGB (or RGBARGBA).

Description: Negate an image.
See also: IrtImgFlipXYImage, IrtImgScaleImage,

7.2.243 IrtImgParsePTextureString (readimag.c:1162)

```
int IrtImgParsePTextureString(const char *PTexture,
                              char *FName,
                              IrtRType *Scale,
                              int *Flip,
                              int *NewImage)
```

PTexture: The string of the "ptexture" attribute.

FName: The texture file name will be placed here.

Scale: The scaling vector in XYZ or just XY if Z = IRIT_INFNTY.

Flip: If Image flipping was requested.

NewImage: if a new image of same name - replace in cache.

Returns: TRUE if parsed successfully, FALSE otherwise.

Description: Parses the string of the "ptexture" attribute. "ImageName {, S X Y {Z}} {, F} {, N}" where

1. X, Y, and possibly Z are scaling factors of how many times the image should fit into the object,
2. 'F' optionally request to flip X and Y axes of the image.
3. 'N' optionally forces reload the image as a New image, even if an image by this exact same name was already loaded and cached.

7.2.244 IrtImgParsePTextureString2 (readimag.c:1079)

```
int IrtImgParsePTextureString2(const char *PTexture,
                               char *FName,
                               IrtRType *Scale,
                               int *Flip,
                               int *NewImage,
                               int *FlipHorizontally,
                               int *FlipVertically)
```

PTexture: The string of the "ptexture" attribute.

FName: The texture file name will be placed here. At least IRIT_LINE_LEN_LONG in length.

Scale: The scaling vector in XYZ or just XY if Z = IRIT_INFNTY.

Flip: If Image flipping was requested.

NewImage: if a new image of same name - replace in cache.

FlipHorizontally: If horizontal Image flipping was requested.

FlipVertically: If vertical Image flipping was requested.

Returns: TRUE if parsed succesfully, FALSE otherwise.

Description: Parses the string of the "ptexture" attribute. "ImageName {, S X Y {Z}} {, F} {, N}, {, H}, {, V}" where

1. X, Y, and possibly Z are scaling factors of how many times the image should fit into the object,
2. 'F' optionally request to flip X and Y axes of the image.
3. 'N' optionally forces reload the image as a New image, even if an image by this exact same name was already loaded and cached.
4. 'H' optionally request to filp the image pixels horizontally.
5. 'V' optionally request to flip the image pixels vertically

7.2.245 IrtImgReadClrCache (readimag.c:367)

```
void IrtImgReadClrCache(void)
```

Returns: void

Description: Clears the cache of read images.
See also: IrtImgReadImage2, IrtImgReadClrOneImage, IrtImgReadUpdateCache,

7.2.246 IrtImgReadClrOneImage (readimag.c:396)

```
void IrtImgReadClrOneImage(const char *ImageName)
```

ImageName: Name of image to remove.

Returns: void

Description: Removes one image, by name from the cache of images.
See also: IrtImgReadImage2, IrtImgReadClrCache,

7.2.247 IrtImgReadImage (readimag.c:131)

```
IrtImgPixelStruct *IrtImgReadImage(const char *ImageFileName,
                                   int *MaxX,
                                   int *MaxY,
                                   int *Alpha)
```

ImageFileName: Name of image to read.

MaxX: Maximum X of read image is saved here.

MaxY: Maximum Y of read image is saved here.

Alpha: If TRUE, will attempt to load alpha channel if has any and will return TRUE if successful in loading it.

Returns: A vector of RGBRGB... of size (MaxX+1) * (MaxY+1) * 3 or NULL if failed. If however, Alpha is requested and found RGBARGBA... is returned as IrtImgRGBAPxlStruct.

Description: Reads one image in from a file named ImageFileName. The image is returned as a vector of RGBRGB... of size (MaxX+1) * (MaxY+1) * 3.
See also: IrtImgReadImage2, IrtImgReadImage3, IrtImgReadImageXAlign,

7.2.248 IrtImgReadImage2 (readimag.c:205)

```
IrtImgPixelStruct *IrtImgReadImage2(const char *ImageFileName,
                                    int *MaxX,
                                    int *MaxY,
                                    int *Alpha)
```

ImageFileName: Name of image to read.

MaxX: Maximum X of read image is saved here.

MaxY: Maximum Y of read image is saved here.

Alpha: If TRUE, will attempt to load alpha channel if has any and will return TRUE if successful in loading it.

Returns: A vector of RGBRGB... of size (MaxX+1) * (MaxY+1) * 3 or NULL if failed. If however, Alpha is requested and found RGBARGBA... is returned as IrtImgRGBAPxlStruct.

Description: Same as IrtImgReadImage but if a name of an image repeats itself, the image is read only once.
See also: IrtImgReadImage, IrtImgReadImage3, IrtImgReadClrCache, , IrtImgReadUpdateCache,

7.2.249 IrtImgReadImage3 (readimag.c:268)

```
IrtImgPixelStruct *IrtImgReadImage3(const char *ImageFileName,
                                    int *MaxX,
                                    int *MaxY,
                                    int *Alpha)
```

ImageFileName: Name of image to read.

MaxX: Maximum X of read image is saved here.

MaxY: Maximum Y of read image is saved here.

Alpha: If TRUE, will attempt to load alpha channel if has any and will return TRUE if successful in loading it.

Returns: A vector of RGBRGB... of size (MaxX+1) * (MaxY+1) * 3 or NULL if failed. If however, Alpha is requested and found RGBARGBA... is returned as IrtImgRGBAPxlStruct.

Description: Same as IrtImgReadImage2 but if a name of an image repeats itself, the new image replaces the old one.
See also: IrtImgReadImage, IrtImgReadImage2, IrtImgReadClrCache, , IrtImgReadUpdateCache,

7.2.250 IrtImgReadImageXAlign (readimag.c:92)

```
int IrtImgReadImageXAlign(int Alignment)
```

Alignment: Word size alignment required.

Returns: old alignment.

Description: Sets an alignment of the width of the image. For example, OGL requires images to have width aligned on 4-bytes words (alignment 4).
See also: IrtImgReadImage, IritImgWriteImg,

7.2.251 IrtImgReadUpdateCache (readimag.c:330)

```
int IrtImgReadUpdateCache(const char *ImageFileName,
                          int MaxX,
                          int MaxY,
                          int Alpha,
                          IrtBType *Image)
```

ImageFileName: Name of image to update.

MaxX: Maximum X of updated image.

MaxY: Maximum Y of updated image.

Alpha: TRUE if image has alpha channel, in which case Image is a sequence of RGBA. FALSE if only RGB is given in Image.

Image: The new image to replace old one with.

Returns: TRUE if image was found and updated. FALSE otherwise.

Description: Update the cached image with the given one, in place, if found.
See also: IrtImgReadImage2, IrtImgReadImage3, IrtImgReadClrOneImage, , IrtImgReadClrCache,

7.2.252 IrtImgWriteCloseFile (writimag.c:369)

output image

```
void IrtImgWriteCloseFile(MiscWriteGenInfoStruct *GI)
```

GI: General info of the image write module.

Returns: void

Description: Closes output image file.

7.2.253 IrtImgWriteGetType (writimag.c:225)

```
IrtImgImageType IrtImgWriteGetType(const char *ImageType)
```

ImageType: A string describing the image type.

Returns: Returns the detected type.

Description: Gets image type.

7.2.254 IrtImgWriteOpenFile (writimag.c:279)

output image

```
MiscWriteGenInfoStruct *IrtImgWriteOpenFile(const char **argv,
                                            const char *FName,
                                            IrtImgImageType ImageType,
                                            int Alpha,
                                            int XSize,
                                            int YSize)
```

argv: Pointer to the name of this program.
FName: Filename to open, or NULL for stdout.
ImageType: Type of image to open.
Alpha: Do we have an alpha channel.
XSize: X dimension of the image.
YSize: Y dimension of the image.
Returns: A handle on local image info if successful, NULL otherwise.
Description: Opens output image file.

7.2.255 IrtImgWritePutLine (writimag.c:339)

output image

```
void IrtImgWritePutLine(MiscWriteGenInfoStruct *GI,
                        IrtBType *Alpha,
                        IrtImgPixelStruct *Pixels)
```

GI: General info of the image write module.
Alpha: Array of alpha values. Can be NULL.
Pixels: Array of color pixels.
Returns: void
Description: Outputs given line of color pixels and alpha correction values to the output image file.

7.2.256 IrtMovieParsePMovieString (readmovi.c:871)

```
int IrtMovieParsePMovieString(const char *PMovie,
                              char *FName,
                              IrtRType *Scale,
                              int *NewImage,
                              int *Flip,
                              int *Restart,
                              IrtRType *TimeSetup,
                              int *FlipHorizontally,
                              int *FlipVertically)
```

PMovie: The string of the "pmovie" attribute.
FName: The texture file name will be placed here.
Scale: The scaling vector in XYZ or just XY if Z = IRIT_INFNTY.
NewImage: Force read an image even if in cache as the file is new?
Flip: If Image flipping was requested.
Restart: TRUE to restart the movie onces it ends.
TimeSetup: A vector of 3 slots to hold (tmin, tmax, dt) if specified or all are zero if not specified.
FlipHorizontally: If horizontal Image flipping was requested.
FlipVertically: If vertical Image flipping was requested.
Returns: TRUE if parsed successfully, FALSE otherwise.
Description: Parses the string of the "pmovie" attribute. Can be "MovieName {, S X Y {Z}} {,F} {,R} {, T=tmin,tmax} where

1. X, Y, and possibly Z are scaling factors of how many times the image should fit into the object,
2. F optionally request to flip X and Y axes of the image.
3. R optionally request to restart the movie once it gets to the end.
4. T=tmin,tmax sets the time range of the movie.

7.2.257 IrtMovieReadClrCache (readmovi.c:366)

```
void IrtMovieReadClrCache(void)
```

Returns: void

Description: Clears the cache of read images.
See also: IrtMovieReadMovie2,

7.2.258 IrtMovieReadMovie (readmovi.c:115)

```
IrtImgPixelStruct **IrtMovieReadMovie(const char *MovieFileName,
                                      int *MaxX,
                                      int *MaxY,
                                      int *Alpha)
```

MovieFileName: Name of movie to read.

MaxX: Maximum X of read image is saved here.

MaxY: Maximum Y of read image is saved here.

Alpha: If TRUE, will attempt to load alpha channel if has any and will return TRUE if successful in loading it.

Returns: A NULL terminated vector of images. Each image is itself a vector of RGBRGB... of size (MaxX+1) * (MaxY+1) * 3 or NULL if failed. If however, Alpha is requested and found RGBARGBA... is returned as IrtImgRGBAPxlStruct.

Description: Reads a movie in from a file named MovieFileName. The movies are returned as a vector of images: vector of RGBRGB... of size (MaxX+1) * (MaxY+1) * 3.
See also: IrtMovieReadMovie2, IrtMovieReadMovie3, IrtImgReadMovieXAlign,

7.2.259 IrtMovieReadMovie2 (readmovi.c:181)

```
IrtImgPixelStruct **IrtMovieReadMovie2(const char *MovieFileName,
                                       int *MaxX,
                                       int *MaxY,
                                       int *Alpha)
```

MovieFileName: Name of movie to read.

MaxX: Maximum X of read image is saved here.

MaxY: Maximum Y of read image is saved here.

Alpha: If TRUE, will attempt to load alpha channel if has any and will return TRUE if successful in loading it.

Returns: A NULL terminated vector of images. Each image is itself a vector of RGBRGB... of size (MaxX+1) * (MaxY+1) * 3 or NULL if failed. If however, Alpha is requested and found RGBARGBA... is returned as IrtImgRGBAPxlStruct.

Description: Same as IrtMovieReadMovie but if a name of a movie repeats itself, the movie is read only once.
See also: IrtMovieReadMovie, IrtImgReadMovie3, IrtMovieReadClrCache,

7.2.260 IrtMovieReadMovie3 (readmovi.c:242)

```
IrtImgPixelStruct **IrtMovieReadMovie3(const char *MovieFileName,
                                       int *MaxX,
                                       int *MaxY,
                                       int *Alpha)
```

MovieFileName: Name of movie to read.

MaxX: Maximum X of read image is saved here.

MaxY: Maximum Y of read image is saved here.

Alpha: If TRUE, will attempt to load alpha channel if has any and will return TRUE if successful in loading it.

Returns: A NULL terminated vector of images. Each image is itself a vector of RGBRGB... of size (MaxX+1) * (MaxY+1) * 3 or NULL if failed. If however, Alpha is requested and found RGBARGBA... is returned as IrtImgRGBAPxlStruct.

Description: Same as IrtMovieReadMovie2 but if a name of a movie repeats itself, the new movie replaces the old one.
See also: IrtMovieReadMovie, IrtImgReadMovie2, IrtMovieReadClrCache,

7.2.261 Mat2x2Determinant (gnrl_mat.c:665)

```
IrtRType Mat2x2Determinant(IrtRType a11,
                           IrtRType a12,
                           IrtRType a21,
                           IrtRType a22)
```

a11, a12, a21, a22: Coefficients of the 2x2 matrix.

Returns: The determinant.

Description: Computes a 2x2 determinant.

7.2.262 Mat3x3Determinant (gnrl_mat.c:691)

```
IrtRType Mat3x3Determinant(IrtRType a11,
                           IrtRType a12,
                           IrtRType a13,
                           IrtRType a21,
                           IrtRType a22,
                           IrtRType a23,
                           IrtRType a31,
                           IrtRType a32,
                           IrtRType a33)
```

a11, a12, a13, a21, a22, a23, a31, a32, a33: Coefficients of 3x3 matrix.

Returns: The determinant.

Description: Computes 3x3 determinant.

7.2.263 MatAddTwo4by4 (hmgn_mat.c:371)

transformations

matrix addition

```
void MatAddTwo4by4(IrtHmgnMatType MatRes,
                   IrtHmgnMatType Mat1,
                   IrtHmgnMatType Mat2)
```

MatRes: Result of matrix addition.

Mat1, Mat2: The two operand of the matrix addition.

Returns: void

Description: Routine to add two 4by4 matrices. MatRes may be one of Mat1 or Mat2.
See also: MatGnrlAddTwo4by4, MatSubTwo4by4, MatMultTwo4by4,

7.2.264 MatDeterminantMatrix (hmgn_mat.c:601)

determinant

```
IrtRType MatDeterminantMatrix(IrtHmgnMatType Mat)
```

Mat: Input matrix.

Returns: Determinant of Mat.

Description: Computes the determinant of a 4 by 4 matrix.

7.2.265 MatGenMatRotX (hmgn_mat.c:209)

transformations

rotation

```
void MatGenMatRotX(IrtRType CosTeta, IrtRType SinTeta, IrtHmgnMatType Mat)
```

CosTeta, SinTeta: Amount of rotation, given as sine and cosine of Teta.

Mat: Matrix to initialize as a rotation matrix.

Returns: void

Description: Routine to generate a 4*4 matrix to Rotate around the X axis by Teta, given the sin and cosine of Teta

7.2.266 MatGenMatRotX1 (hmgn_mat.c:185)

transformations

rotation

```
void MatGenMatRotX1(IrtRType Teta, IrtHmgnMatType Mat)
```

Teta: Amount of rotation, in radians.

Mat: Matrix to initialize as a rotation matrix.

Returns: void

Description: Routine to generate a 4*4 matrix to Rotate around the X axis by Teta radians.

7.2.267 MatGenMatRotY (hmgn_mat.c:257)

transformations

rotation

```
void MatGenMatRotY(IrtRType CosTeta, IrtRType SinTeta, IrtHmgnMatType Mat)
```

CosTeta, SinTeta: Amount of rotation, given as sine and cosine of Teta.

Mat: Matrix to initialize as a rotation matrix.

Returns: void

Description: Routine to generate a 4*4 matrix to Rotate around the Y axis by Teta, given the sin and cosine of Teta

7.2.268 MatGenMatRotY1 (hmgn_mat.c:233)

transformations

rotation

```
void MatGenMatRotY1(IrtRType Teta, IrtHmgnMatType Mat)
```

Teta: Amount of rotation, in radians.

Mat: Matrix to initialize as a rotation matrix.

Returns: void

Description: Routine to generate a 4*4 matrix to Rotate around the Y axis by Teta radians.

7.2.269 MatGenMatRotZ (hmgn_mat.c:305)

transformations

rotation

```
void MatGenMatRotZ(IrtRType CosTeta, IrtRType SinTeta, IrtHmgnMatType Mat)
```

CosTeta, SinTeta: Amount of rotation, given as sine and cosine of Teta.

Mat: Matrix to initialize as a rotation matrix.

Returns: void

Description: Routine to generate a 4*4 matrix to Rotate around the Z axis by Teta, given the sin and cosine of Teta

7.2.270 MatGenMatRotZ1 (hmgn_mat.c:281)

transformations

rotation

```
void MatGenMatRotZ1(IrtRType Teta, IrtHmgnMatType Mat)
```

Teta: Amount of rotation, in radians.

Mat: Matrix to initialize as a rotation matrix.

Returns: void

Description: Routine to generate a 4*4 matrix to Rotate around the Z axis by Teta radians.

7.2.271 MatGenMatScale (hmgn_mat.c:143)

transformations

scaling

```
void MatGenMatScale(IrtRType Sx, IrtRType Sy, IrtRType Sz, IrtHmgnMatType Mat)
```

Sx, Sy, Sz: Scaling factors requested.

Mat: Matrix to initialize as a scaling matrix.

Returns: void

Description: Routine to generate a 4*4 matrix to Scale x, y, z in Sx, Sy, Sz amounts.

7.2.272 MatGenMatTrans (hmgn_mat.c:121)

transformations

translation

```
void MatGenMatTrans(IrtRType Tx, IrtRType Ty, IrtRType Tz, IrtHmgnMatType Mat)
```

Tx, Ty, Tz: Translational amounts requested.

Mat: Matrix to initialize as a translation matrix.

Returns: void

Description: Routine to generate a 4*4 matrix to translate in Tx, Ty, Tz amounts.

7.2.273 MatGenMatUnifScale (hmgn_mat.c:165)

transformations

scaling

```
void MatGenMatUnifScale(IrtRType Scale, IrtHmgnMatType Mat)
```

Scale: Uniform scaling factor requested.

Mat: Matrix to initialize as a scaling matrix.

Returns: void

Description: Routine to generate a 4*4 matrix to uniformly scale Scale amount.

7.2.274 MatGenUnitMat (hmgn_mat.c:31)

transformations

unit matrix

```
void MatGenUnitMat(IrtHmgnMatType Mat)
```

Mat: Matrix to initialize as a unit matrix.

Returns: void

Description: Routine to generate a 4*4 unit matrix:
See also: MatGnrlUnitMat,

7.2.275 MatGnrlAddTwoMat (gnrl_mat.c:164)

transformations

matrix addition

```
void MatGnrlAddTwoMat(IrtGnrlMatType MatRes,
                      IrtGnrlMatType Mat1,
                      IrtGnrlMatType Mat2,
                      int n)
```

MatRes: Result of matrix addition.

Mat1, Mat2: The two operand of the matrix addition.

n: Size of matrices MatRes/Mat1/Mat2.

Returns: void

Description: Routine to add two general matrices. MatRes may be one of Mat1 or Mat2.
See also: MatAddTwoMat, MatGntlMultTwomat, MatGnrlSubTwoMat, MatGnrlScaleMat,

7.2.276 MatGnrlCopy (gnrl_mat.c:29)

transformations

```
void MatGnrlCopy(IrtGnrlMatType Dst, IrtGnrlMatType Src, int n)
```

Dst: Matrix to copy to.

Src: Matrix to copy from.

n: Size of matrix Mat.

Returns: void

Description: Routine to copy a (n x n) matrix:

7.2.277 MatGnrlDetMatrix (gnrl_mat.c:480)

```
IrtRType MatGnrlDetMatrix(IrtGnrlMatType M, int n)
```

M: The square matrix of size (n x n) to compute its determinant.

n: Size of the Matrix.

Returns: Value of the determinant.

Description: Given a general matrix, computes its determinant, recorsively. Note this process is exponential as a function of n (as you might expect) so use it with care (for small matrices only).
See also: MatGnrlInverseMatrix, MatGnrlTranspMatrix,

7.2.278 MatGnrlInverseMatrix (gnrl_mat.c:340)

transformations

matrix inverse

```
int MatGnrlInverseMatrix(IrtGnrlMatType M, IrtGnrlMatType InvM, int n)
```

M: Original matrix to invert.

InvM: Inverted matrix will be placed here.

n: Sizes of matrices M/InvM.

Returns: TRUE if inverse exists, FALSE otherwise.

Description: Routine to compute the INVERSE of a given matrix M which is not modified. Return TRUE if inverted matrix (InvM) do exists.
See also: MatInverseMatrix, MatGnrlTranspMatrix,

7.2.279 MatGnrlIsUnitMatrix (gnrl_mat.c:83)

```
int MatGnrlIsUnitMatrix(IrtGnrlMatType Mat, IrtRType Eps, int n)
```

Mat: Matrix to test if a unit matrix.

Eps: Epsilon of test.

n: Size of matrix Mat.

Returns: TRUE if unit matrix to within epsilon, FALSE otherwise.

Description: Test if the given matrix is a unit matrix to within Eps.
See also: MatGnrlUnitMat, MatIsUnitMatrix,

7.2.280 MatGnrlMultTwoMat (gnrl_mat.c:119)

transformations

matrix product

```
void MatGnrlMultTwoMat(IrtGnrlMatType MatRes,
                       IrtGnrlMatType Mat1,
                       IrtGnrlMatType Mat2,
                       int n)
```

MatRes: Result of matrix product.

Mat1, Mat2: The two operand of the matrix product.

n: Size of matrices MatRes/Mat1/Mat2.

Returns: void

Description: Routine to multiply two general matrices. MatRes may be one of Mat1 or Mat2 - it is only updated in the end.
See also: MatMultTwoMat, MatGnrlAddTwoMat, MatGnrlSubTwoMat,

7.2.281 MatGnrlMultVecbyMat (gnrl_mat.c:256)

transformations

vector matrix product

```
void MatGnrlMultVecbyMat(IrtVecGnrlType VecRes,
                         IrtGnrlMatType Mat,
                         IrtVecGnrlType Vec,
                         int n)
```

VecRes: Result of vector - matrix product. Can be Vec.

Mat: General matrix.

Vec: Vector to transform using Matrix.

n: Sizes of vectors Vec/VecRes and of matrix Mat.

Returns: void

Description: Routine to mult a vector of len n by a general matrix of size (n x n), as VecRes = Mat * Vec.
See also: MatMultVecby4by4, MatGnrlMultVecbyMat2,

7.2.282 MatGnrlMultVecbyMat2 (gnrl_mat.c:299)

transformations

vector matrix product

```
void MatGnrlMultVecbyMat2(IrtVecGnrlType VecRes,
                          IrtVecGnrlType Vec,
                          IrtGnrlMatType Mat,
                          int n)
```

VecRes: Result of vector - matrix product. Can be Vec.

Vec: Vector to transform using Matrix.

Mat: General matrix.

n: Sizes of vectors Vec/VecRes and of matrix Mat.

Returns: void

Description: Routine to mult a vector of len n by a general matrix of size (n x n), as VecRes = Vec * Mat.
See also: MatMultVecby4by4, MatGnrlMultVecbyMat,

7.2.283 MatGnrlOrthogonalSubspace (gnrl_mat.c:529)

```
int MatGnrlOrthogonalSubspace(IrtGnrlMatType M, int n)
```

M: A matrix holding m (m < n) vectors as rows.

n: Size of marix M (and the space).

Returns: TRUE if successful, FALSE otherwise.

Description: Given m (m < n) vectors as rows in matrix M, and zeros at the rest of the rows, computes n-m new vectors of the orthogonal space to the space spanned by the input m vectors. Uses SVD in the computation and saves the new, orthogonal, vectors in the last n-m rows of M, in place.

7.2.284 MatGnrlPrintMatrix (gnrl_mat.c:637)

transformations

matrix transpose

```
void MatGnrlPrintMatrix(IrtGnrlMatType M, int n, FILE *F)
```

M: Matrix to print.

n: Size of matrix Mat.

F: What file to print to.

Returns: void

Description: Routine to print matrix M to file F.

7.2.285 MatGnrlScaleMat (gnrl_mat.c:225)

transformations

matrix scaling

```
void MatGnrlScaleMat(IrtGnrlMatType MatRes,
                     IrtGnrlMatType Mat,
                     IrtRType *Scale,
                     int n)
```

MatRes: Result of matrix scaling.

Mat: The two operand of the matrix scaling.

Scale: Scalar value to multiple matrix with.

n: Size of matrices MatRes/Mat.

Returns: void

Description: Routine to scale a general matrix, by a scalar value. MatRes may be Mat.
See also: MatScaleMat, MatGnrlAddTwoMat, MatGntlMultTwoMat,

7.2.286 MatGnrlSubTwoMat (gnrl_mat.c:194)

transformations

matrix subtraction

```
void MatGnrlSubTwoMat(IrtGnrlMatType MatRes,
                      IrtGnrlMatType Mat1,
                      IrtGnrlMatType Mat2,
                      int n)
```

MatRes: Result of matrix subtraction.

Mat1, Mat2: The two operand of the matrix subtraction.

n: Size of matrices MatRes/Mat1/Mat2.

Returns: void

Description: Routine to subtract two general matrices. MatRes may be one of Mat1 or Mat2.
See also: MatSubTwoMat, MatGnrlAddTwoMat, MatGntlMultTwoMat, MatGnrlScaleMat,

7.2.287 MatGnrlTranspMatrix (gnrl_mat.c:445)

transformations

matrix transpose

```
void MatGnrlTranspMatrix(IrtGnrlMatType M, IrtGnrlMatType TranspM, int n)
```

M: Original matrix to transpose.

TranspM: Transposed matrix will be placed here.

n: Sizes of matrices M/TranspM.

Returns: void

Description: Routine to compute the TRANSPOSE of matrix M which is not modified.
See also: MatTranspMatrix, MatGnrlTranspMatrix,

7.2.288 MatGnrlUnitMat (gnrl_mat.c:51)

transformations

unit matrix

```
void MatGnrlUnitMat(IrtGnrlMatType Mat, int n)
```

Mat: Matrix to initialize as a unit matrix.

n: Size of matrix Mat.

Returns: void

Description: Routine to generate a (n x n) unit matrix:
See also: MatGenUnitMat,

7.2.289 MatInverseMatrix (hmgn_mat.c:643)

transformations

matrix inverse

```
int MatInverseMatrix(IrtHmgnMatType M, IrtHmgnMatType InvM)
```

M: Original matrix to invert.

InvM: Inverted matrix will be placed here. Can be same as M.

Returns: TRUE if inverse exists, FALSE otherwise.

Description: Routine to compute the INVERSE of a given matrix M which is not modified. The matrix is assumed to be 4 by 4 (transformation matrix). Return TRUE if inverted matrix (InvM) do exists.
See also: MatTranspMatrix,

7.2.290 MatIsUnitMatrix (hmgn_mat.c:60)

```
int MatIsUnitMatrix(IrtHmgnMatType Mat, IrtRType Eps)
```

Mat: Matrix to test if a unit matrix.

Eps: Epsilon of test.

Returns: TRUE if unit matrix to within epsilon, FALSE otherwise.

Description: Test if the given matrix is a unit matrix to within Eps.
See also: MatGenUnitMat, MatGnrlIsUnitMatrix, MatIsWeightAffected,

7.2.291 MatIsWeightAffected (hmgn_mat.c:92)

```
int MatIsWeightAffected(IrtHmgnMatType Mat, IrtRType Eps)
```

Mat: Matrix to test if weight is affected by it.

Eps: Epsilon of test.

Returns: TRUE if matrix indeed affects the weight coefficient.

Description: Test if the given matrix affects the weight in any way (i.e. W = 1 will not be W = 1 after the transform).
See also: MatGenUnitMat, MatGnrlIsUnitMatrix, MatIsUnitMatrix,

7.2.292 MatMultPtby4by4 (hmgn_mat.c:524)

transformations

vector matrix product

```
void MatMultPtby4by4(IrtPtType PtRes,
                     const IrtPtType Pt,
                     IrtHmgnMatType Mat)
```

PtRes: Result of point - matrix product.

Pt: Point to transfrom using Matrix.

Mat: Transformation matrix.

Returns: void

Description: Routine to multiply a XYZ point by 4by4 matrix: The point has only 3 components (X, Y, Z) and it is assumed that W = 1. PtRes may be Pt as it is only updated in the end.
See also: MatMultVecby4by4, MatMultWVec2by4by4,

7.2.293 MatMultTwo4by4 (hmgn_mat.c:335)

transformations

matrix product

```
void MatMultTwo4by4(IrtHmgnMatType MatRes,
                    IrtHmgnMatType Mat1,
                    IrtHmgnMatType Mat2)
```

MatRes: Result of matrix product.

Mat1, Mat2: The two operand of the matrix product.

Returns: void

Description: Routine to multiply two 4by4 matrices. MatRes = Mat1 * Mat2 assumed Mat[i][j] refers to row i and column j. MatRes may be one of Mat1 or Mat2 - it is only updated in the end.
See also: MatGnrlMultTwo4by4, MatSubTwo4by4, MatAddTwo4by4,

7.2.294 MatMultVecby4by4 (hmgn_mat.c:491)

transformations

vector matrix product

```
void MatMultVecby4by4(IrtVecType VecRes,
                      const IrtVecType Vec,
                      IrtHmgnMatType Mat)
```

VecRes: Result of vector - matrix product.

Vec: Vector to transfrom using Matrix.

Mat: Transformation matrix.

Returns: void

Description: Routine to multiply an XYZ Vector by 4by4 matrix: The Vector has only 3 components (X, Y, Z) and it is assumed that W = 0. VRes may be Vec as it is only updated in the end.
See also: MatMultPtby4by4, MatMultWVec2by4by4,

7.2.295 MatMultWVecby4by4 (hmgn_mat.c:572)

transformations

vector matrix product

```
void MatMultWVecby4by4(IrtRType VRes[4],
                       const IrtRType Vec[4],
                       IrtHmgnMatType Mat)
```

VRes: Result of vector - matrix product.

Vec: Vector to transfrom using Matrix.

Mat: Transformation matrix.

Returns: void

Description: Routine to multiply a WXYZ Vector by 4by4 matrix: The Vector has only 4 components (X, Y, Z, W). VRes may be Vec as it is only updated in the end.
See also: MatMultPtby4by4, MatMultVec2by4by4,

7.2.296 MatRotSclFactorMatrix (hmgn_mat.c:883)

transformations

```
void MatRotSclFactorMatrix(IrtHmgnMatType M, IrtHmgnMatType RotSclMat)
```

M: Matrix to extract rotation and scale factors from.

RotSclMat: The rotation and scale factors of matrix M.

Returns: void

Description: Routine to estimate the rotation and scale factors in a matrix.
See also: MatScaleFactorMatrix, MatRotateFactorMatrix, MatTranslateFactorMatrix,

7.2.297 MatRotateFactorMatrix (hmgn_mat.c:842)

transformations

```
void MatRotateFactorMatrix(IrtHmgnMatType M, IrtHmgnMatType RotMat)
```

M: Matrix to extract rotation factors.

RotMat: The rotational factors of matrix M.

Returns: void

Description: Routine to estimate the rotation factor in a matrix.
See also: MatScaleFactorMatrix, MatTranslateFactorMatrix, MatRotSclFactorMatrix,

7.2.298 MatSameTwo4by4 (hmgn_mat.c:455)

```
int MatSameTwo4by4(IrtHmgnMatType Mat1,
                   IrtHmgnMatType Mat2,
                   IrtRType Eps)
```

Mat1, Mat2: Two homogeneous matrices to compare.

Eps: Tolerance of comparison.

Returns: TRUE if similar, FALSE otherwise.

Description: Compare two homogeneous matrices for approximate equality.

7.2.299 MatScale4by4 (hmgn_mat.c:430)

transformations

matrix scaling

```
void MatScale4by4(IrtHmgnMatType MatRes,
                  IrtHmgnMatType Mat,
                  const IrtRType *Scale)
```

MatRes: Result of matrix scaling.

Mat: The two operand of the matrix scaling.

Scale: Scalar value to multiple matrix with.

Returns: void

Description: Routine to scale a 4by4 matrix. MatRes may be Mat.
See also: MatGnrlScaleMat, MatSubTwo4by4, MatAddTwo4by4, MatMultTwo4by4,

7.2.300 MatScaleFactorMatrix (hmgn_mat.c:765)

transformations

```
IrtRType MatScaleFactorMatrix(IrtHmgnMatType M)
```

M: Matrix to estimate scaling factors (assume positive scales).

Returns: Estimated Scaling factor (returns positive scale values).

Description: Routine to estimate the scaling factor in a matrix by computing the SVD decomposition of the matrix and if fails, use average of the scale of the X, Y, and Z unit vectors.
See also: MatRotateFactorMatrix, MatTranslateFactorMatrix, MatRotSclFactorMatrix, MatScaleFactorMatrix2,

7.2.301 MatScaleFactorMatrix2 (hmgn_mat.c:795)

transformations

```
IrtRType *MatScaleFactorMatrix2(IrtHmgnMatType M, IrtVecType ScaleVec)
```

M: Matrix to estimate scaling factors (assume positive scales).

ScaleVec: To save the eEstimated Scaling factor in the XYZ directions.

Returns: Estimated Scaling factor in the XYZ directions. Same as ScaleVec.

Description: Routine to estimate the scaling factor in a matrix by computing the SVD decomposition of the matrix and if fails, maps and measures the scale of unit vectors in the X, Y, and Z direction.
See also: MatRotateFactorMatrix, MatTranslateFactorMatrix, MatRotSclFactorMatrix, MatScaleFactorMatrix,

7.2.302 MatSubTwo4by4 (hmgn_mat.c:400)

transformations

matrix subtraction

```
void MatSubTwo4by4(IrtHmgnMatType MatRes,
                   IrtHmgnMatType Mat1,
                   IrtHmgnMatType Mat2)
```

MatRes: Result of matrix subtraction.

Mat1, Mat2: The two operand of the matrix subtraction.

Returns: void

Description: Routine to subtract two 4by4 matrices. MatRes may be one of Mat1 or Mat2.
See also: MatGnrlSubTwo4by4, MatAddTwo4by4, MatMultTwo4by4,

7.2.303 MatTranslateFactorMatrix (hmgn_mat.c:908)

transformations

```
void MatTranslateFactorMatrix(IrtHmgnMatType M, IrtVecType Trans)
```

M: Matrix to extract rotation factors.

Trans: The translation factors of matrix M.

Returns: void

Description: Routine to estimate the translation factors in a matrix.
See also: MatScaleFactorMatrix, MatRotateFactorMatrix,

7.2.304 MatTranspMatrix (hmgn_mat.c:732)

transformations

matrix transpose

```
void MatTranspMatrix(IrtHmgnMatType M, IrtHmgnMatType TranspM)
```

M: Original matrix to transpose.

TranspM: Transposed matrix will be placed here. Can be same as M.

Returns: void

Description: Routine to compute the TRANSPOSE of a given matrix M which is not modified. The matrix is assumed to be 4 by 4 (transformation matrix).
See also: MatInverseMatrix,

7.2.305 MiscBiPrComputeMinCostRecMatching (bipartte2.c:102)

```
int *MiscBiPrComputeMinCostRecMatching(const IrtRType *CostMat,
                                       int NumRows,
                                       int NumCols)
```

CostMat: Cost matrix of size R x C. The costs are stored in a column- major, i.e., CostMat[r + NumRows * c]. Costs should be always nonnegative and invalid weights are represented as infinity.

NumRows: Number of rows in the cost matrix.

NumCols: Number of columns in the cost matrix.

Returns: A 1D array of size R. Stores the matched column index for each row when Match[i] = -1, row i does not match to any column.

Description: Implementation of Kunh-Munkres algorithm version of the Hungarian alg. for the rectangular (unbalanced) weighted bi-partite graph. The original Munkres algorithm finds the minimal assignment between N workers and N jobs given the N x N cost matrix, by starring and priming the cost matrix and covering and uncovering the rows and columns. This implementation is the modified version of Munkres algorithm to handle the rectangular M x N cost matrix. When k = min (M, N), this function finds at most k assignments. The cost matrix (CostMat) is a 1D array storing the costs(weights) in a column-major order: CostMat[r + NumRows * c] stores the cost at row r and column c from the cost matrix. Invalid costs are represented as infinity. The flow diagram of the algorithm is as follows:
Step 1(Preprocess) -> Step 2 -> Exit -> Step 3 -> Step 4 -> Step 2 -> Step 5 -> Step 3 Time complexity of Munkres algorithm is O(n^3).
[Reference]

1. J. Munkres. Algorithms for the Assignment and Transportation Problems. Journal of the Society for Industrial and Applied Mathematics, Vol. 5, No. 1, pp. 32–38, 1958.
2. R.A. Pilgrim, Munkres' Assignment Algorithm. Modified for Rectangular Matrices. (http://csclab.murraystate.edu/ bob.pilgrim/ Course notes, Murray State University.
3. M. Buehren. Functions for the rectangular assignment problem. (https://www.mathworks.com/matlabcentral/fileexchange/6543-functions-for-the-rectangular-assignment-problem), MATLAB Central File Exchange. 2020.

See also:

7.2.306 MiscBiPrWeightedMatchBipartite (bipartte.c:84)

```
int MiscBiPrWeightedMatchBipartite(const IrtRType **Weight,
                                   IritBiPrWeightedMatchStruct *Match,
                                   int n)
```

Weight: Cost matrix of size n*n, passed as an array of size n holding pointers to arrays of size n. Weight[i][j] is the cost of the edge between vertices i and j. Negative entry denotes that there is no edge (== edge cost is infinite).

Match: Array of size n allocated in advance. Returns with the optimal matching found. For every 0 <= i < n Match[i].m1 = i Match[i].m2 = vertex that is matched to i Match[i].m3 = unused

n: Number of vertices in each side of the bi-partite graph.

Returns: 0 if an optimal match was found - in such case the array Match is filled with the optimal matching found. Negative value if there is no match in the graph - in such case Match may be undefined.

Description: Implementation of Shortest-Path variation of the Hungarian algorithm, from "Assignment Problems" by R. Burkard, M. Dell'Amico, S. Martello Algorithm 4.10 Hungarian_SP, page 97. Complexity: O(n^3) The sacnning order of the graph's vertices is chosen at random.

7.2.307 MiscHashAddElement (hash2tbl.c:136)

```
int MiscHashAddElement(MiscHashPtrType Hash,
                       void *Elem,
                       unsigned long SizeInByte)
```

Hash: The hash table.

Elem: The element to hash.

SizeInByte: The second parameter to pass to the hash function.

Returns: -1 if unable to add, 0 otherwise.

Description: Hash the given element and insert it into the table. The added element is the returned value from the CopyFunc given to MiscHashNewHash. More than one identical element can exist in the table. The element's auxilary data is initialized to 0.

7.2.308 MiscHashFindElement (hash2tbl.c:233)

```
int MiscHashFindElement(MiscHashPtrType Hash,
                        void *Elem,
                        unsigned long SizeInByte)
```

Hash: The hash table.

Elem: The element to hash.

SizeInByte: The second parameter to pass to the hash function. If the second parameter is not equal to the one used to hash the element, the element may not be found even if it is in the hash table.

Returns: 0 if found, -1 otherwise.

Description: Looks for the given element in the hash table (using the CompFunc given to MiscHashNewHash). The Element and SizeInByte will be passed to the hash function.

7.2.309 MiscHashFreeHash (hash2tbl.c:288)

```
void MiscHashFreeHash(MiscHashPtrType Hash)
```

Hash: The hash table to free.

Returns: void

Description: Frees the given hash table.

7.2.310 MiscHashGetElementAuxData (hash2tbl.c:265)

```
long *MiscHashGetElementAuxData(MiscHashPtrType Hash,
                                void *Elem,
                                unsigned long SizeInByte)
```

Hash: The hash table.

Elem: The element to retrieve its auxilary data.

SizeInByte: The second parameter to pass to the hash function. If the second parameter is not equal to the one used to hash the element, the element may not be found even if it is in the hash table.

Returns: Pointer to Elem's auxilary data. NULL if Elem doesn't exist in the hash table.

Description: Find the first occurence of Elem in the hash table (using the CompFunc given to MiscHashNewHash) and return a pointer to its auxilary data (The auxilary data may be changed without influencing the hash table). The Element and SizeInByte will be passed to the hash function.

7.2.311 MiscHashNewHash (hash2tbl.c:75)

```
MiscHashPtrType MiscHashNewHash(unsigned long Size,
                                MiscHashFuncType HashFunc,
                                MiscHashCopyFuncType CopyFunc,
                                MiscHashFreeFuncType FreeFunc,
                                MiscHashCompFuncType CompFunc)
```

Size: The size of the hash table (the range of the hash function should be compatible).

HashFunc: The function used for hashing (the range should be compatible with HashSize). Receives 2 parameters to make it easy to hash arrays.

CopyFunc: The function that will be used to copy elements (for example when hashing a new element).

FreeFunc: The function that will be used to free elements (for example when freeing the hash table).

CompFunc: The function that will be used to compare elements (for example when looking for an element in the hash table).

Returns: The new hash table if successful, of NULL if failed.

Description: Creates a new Hash Table.

7.2.312 MiscISCAddPicture (imgstcvr.c:1648)

```
int MiscISCAddPicture(MiscISCCalculatorPtrType Calc, MiscISCPixelType *Picture)
```

Calc: The calculator to add the picture to.

Picture: The picture to add.

Returns: FALSE if error occurred.

Description: Adds a picture to the calculator. Copies the picture (will be freed when the calculator is freed).
See also:

7.2.313 MiscISCCalculateExact (imgstcvr.c:2685)

MiscISCCalculateExactAux

```
int MiscISCCalculateExact(MiscISCCalculatorPtrType Calc,
                          int SizeLimit,
                          int **SolutionByIndex,
                          int *SolutionSize,
                          IrtRType *CoverPart)
```

Calc: The calculator.

SizeLimit: IN, if this value is greater than 0 the algorithm will stop adding pictures when getting to that size of pictures combinations and revert to search in other branches. This will speed the algorithm though may not find any solution at all in which case the combination with the best cover will be returned.

SolutionByIndex: OUT, The solution as indices of pictures.

SolutionSize: OUT, The size of the solution (size of SolutionByIndex).

CoverPart: OUT, The part of the uprocessed picture covered by the solution.

Returns: FALSE if error occurred.

Description: Calculates the cover. Uses an exact exponential Algorithm. Each MiscISCCalculatorPtrType can be used only once with any search algorithm. Calling this funcion will move the calculator to compute phase.
See also: ,

7.2.314 MiscISCCalculateExhaustive (imgstcvr.c:2909)

MiscISCCalculateExact

```
int MiscISCCalculateExhaustive(MiscISCCalculatorPtrType Calc,
                               IrtRType CoverLimit,
                               int SizeLimit,
                               int **SolutionByIndex,
                               int *SolutionSize,
                               IrtRType *CoverPart)
```

Calc: The calculator.

CoverLimit: IN, if this value is in (0,1), the algorithm will consider a combination of pictures which covers CoverLimit part of the image as a valid solution.

SizeLimit: IN, if this value is greater than 0 the algorithm will stop adding pictures when getting to that size of pictures combinations and revert to search in other branches. This will speed the algorithm though may not find any solution at all in which case the combination with the best cover will be returned.

SolutionByIndex: OUT, The solution as indices of pictures.

SolutionSize: OUT, The size of the solution (size of SolutionByIndex).

CoverPart: OUT, The part of the unprocessed pictured covered by the solution.

Returns: FALSE if error occurred.

Description: Calculates the cover. Uses an exhaustive exponential Algorithm. Each MiscISCCalculatorPtrType can be used only once with any search algorithm. Calling this funcion will move the calculator to compute phase.
See also: ,

7.2.315 MiscISCCalculateGreedy (imgstcvr.c:3389)

MiscISCCalculateGreedyAux

```
int MiscISCCalculateGreedy(MiscISCCalculatorPtrType Calc,
                           int **SolutionByIndex,
                           int *SolutionSize,
                           IrtRType *CoverPart)
```

Calc: The calculator.

SolutionByIndex: OUT, The solution as indices of pictures by the order they were added to the solution.

SolutionSize: OUT, The size of the solution (size of SolutionByIndex).

CoverPart: OUT, The part of the unprocessed image covered by the solution.

Returns: FALSE if error occurred.

Description: Calculates the cover using quick and not optimal greedy algorithm. In each step the algorithm choose the next picture to be the picture which covers the maximum number of pixel out of the original unprocessed picture. If Calc -> UseCoverLimit is TRUE, the algorithm's search will stop when reaching cover the size of Calc -> CoverLimit and return the solution. Each MiscISCCalculatorPtrType can be used only once with any search algorithm. Calling this funcion will move the calculator to compute phase.

7.2.316 MiscISCFreeCalculator (imgstcvr.c:1560)

```
void MiscISCFreeCalculator(MiscISCCalculatorPtrType Calc)
```

Calc: The calculator.

Returns: void

Description: Frees memory allocated for the calculator.
See also:

7.2.317 MiscISCNewCalculator (imgstcvr.c:1422)

```
MiscISCCalculatorPtrType MiscISCNewCalculator(int MaxPictures,
                                              MiscISCImageSizeType ImageSize,
                                              MiscISCColorTypeEnum ColorType,
                                              MiscISCPrintFuncType Print)
```

MaxPictures: The maximal number of pictures the calculator will hold.

ImageSize: The number of pixels in each picture.

ColorType: Are the pictures in black & white or gray.

Print: Pointer to a printing routine where texture data should go to or NULL to ignore.

Returns: The new calculator or NULL if fails.

Description: Creates a new calculator.

7.2.318 MiscISCSetImageToCover (imgstcvr.c:1612)

```
int MiscISCSetImageToCover(MiscISCCalculatorPtrType Calc,
                           MiscISCPixelType *RequiredCover)
```

Calc: The calculator to add the picture to.

RequiredCover: The cover picture to add.

Returns: FALSE if error occurred.

Description: Set the image to be covered. A pixel with value 0 means that the pixel isn't required to be covered. A pixel with value 1 means that the pixel is required to be covered. If this function isn't called than all the pixel are required to be covered. The function makes an inner copy of RequiredCover (that copy will be freed when the calculator is freed). The user may free or change RequiredCover's memory without harming the process.
See also:

7.2.319 MiscListAddElement (list.c:91)

```
int MiscListAddElement(MiscListPtrType List,
                       void *Elem,
                       unsigned long SizeInByte)
```

List: The list where the element will be inserted.

Elem: The element to add to the list. Must be compatible with the functions given to MiscListNewEmptyList.

SizeInByte: Can be used to pass an array size, for the Copy Function.

Returns: -1 if unable to add, 0 otherwise.

Description: Adds a new element to the beginning of the list. The added element is the returned value from the CopyFunc given to MiscListNewEmptyList.

7.2.320 MiscListCompLists (list.c:158)

```
int MiscListCompLists(MiscListPtrType L1, MiscListPtrType L2)
```

L1: A lists to compare.

L2: A lists to compare.

Returns: 0 if the lists are equal.

Description: Compares the 2 given list element by element. The list are considered equal if they have the same number of elements, in the same order.

7.2.321 MiscListFindElementInList (list.c:126)

```
int MiscListFindElementInList(MiscListPtrType List,
                              void *Elem,
                              unsigned long SizeInByte)
```

List: The list where the element will be looked for.

Elem: The element to look for.

SizeInByte: The second parameter to pass to the CompFunc (CompFunc takes 2 parametes to make it easy to compare arrays).

Returns: -1 if not found, 0 otherwise.

Description: Looks for the give element in the list (using the CompFunc given to MiscListNewEmptyList).

7.2.322 MiscListFreeList (list.c:201)

```
void MiscListFreeList(MiscListPtrType List)
```

List: The list to free.

Returns: void

Description: Frees the List (and elements).

7.2.323 MiscListFreeListIterator (list.c:264)

```
void MiscListFreeListIterator(MiscListIteratorPtrType It)
```

It: The iterator to free.

Returns: void

Description: Frees the given iterator.

7.2.324 MiscListGetListIterator (list.c:236)

```
MiscListIteratorPtrType MiscListGetListIterator(MiscListPtrType List)
```

List: The list to iterate over.

Returns: The new iterator, of NULL if fails.

Description: Returns a new iterator to the given list. Before MiscListIteratorFirst is used the place where the iterator points to is undefined.

7.2.325 MiscListIteratorAtEnd (list.c:333)

```
int MiscListIteratorAtEnd(MiscListIteratorPtrType It)
```

It: The iterator to check.

Returns: 1 if at the end of the list, 0 otherwise.

Description: Returns 1 if at the end of the List.

7.2.326 MiscListIteratorFirst (list.c:286)

```
void *MiscListIteratorFirst(MiscListIteratorPtrType It)
```

It: The iterator to reset.

Returns: The element at the head of the list.

Description: Resets the iterator to the head of the list.
See also:

7.2.327 MiscListIteratorNext (list.c:309)

```
void *MiscListIteratorNext(MiscListIteratorPtrType It)
```

It: The iterator to move.

Returns: The element at the new location in the list (if the new location is the end of the list, the return value is NULL).

Description: Moves the iterator to the next element in the list. If used when the iterator is at the end of the list the result is undefined.

7.2.328 MiscListIteratorValue (list.c:355)

```
void *MiscListIteratorValue(MiscListIteratorPtrType It)
```

It: The iterator.

Returns: The data stored the the current element.

Description: Returns the data stored the the current element.

7.2.329 MiscListNewEmptyList (list.c:50)

```
MiscListPtrType MiscListNewEmptyList(MiscListCopyFuncType CopyFunc,
                                     MiscListFreeFuncType FreeFunc,
                                     MiscListCompFuncType CompFunc)
```

CopyFunc: The function that will be used to copy elements (for example when adding a new element to the list).

FreeFunc: The function that will be used to free elements (for example when freeing the list).

CompFunc: The function that will be used to compare elements (for example when looking for an element in the list).

Returns: The new list if successful, of NULL if fails.

Description: Creates a new List.

7.2.330 MiscPHashMapCreate (phashmap.c:56)

```
MiscPHashMap MiscPHashMapCreate(int HashSize)
```

HashSize: Size of the hash array.

Returns: New hash map.

Description: Creates a hash map.

7.2.331 MiscPHashMapDelete (phashmap.c:94)

```
void MiscPHashMapDelete(MiscPHashMap HMap)
```

HMap: Hash map to destroy.

Returns: void

Description: Destroys a hash map.

7.2.332 MiscPHashMapGet (phashmap.c:168)

```
MiscPHMResult MiscPHashMapGet(MiscPHashMap HMap,
                              MiscPHashMapKey Key,
                              MiscPHashMapValue *Value)
```

HMap: Hash map.

Key: Key to search for.

Value: Return parameter of the key value if found.

Returns: MISC_P_HMSUCCESS in success and MISC_P_MFAILURE otherwise.

Description: Gets the value of a given key in a hash map.

7.2.333 MiscPHashMapIterate (phashmap.c:247)

```
void MiscPHashMapIterate(MiscPHashMap HMap, MiscPHashMapCBFunc CBFunc)
```

HMap: Hash map.

CBFunc: Entries call back function.

Returns: void

Description: Iterates over the entries of a hash map and calls a call back function on them.

7.2.334 MiscPHashMapPrint (phashmap.c:274)

```
void MiscPHashMapPrint(MiscPHashMap HMap)
```

HMap: Hash map.

Returns: void

Description: Prints the entries and other information of a hash map.

7.2.335 MiscPHashMapRemove (phashmap.c:203)

```
MiscPHMResult MiscPHashMapRemove(MiscPHashMap HMap, MiscPHashMapKey Key)
```

HMap: Hash map.

Key: Key to search for.

Returns: MISC_P_HMSUCCESS in success and MISC_P_MFAILURE otherwise if the key is not found.

Description: Removes key and its value from the hash map.

7.2.336 MiscPHashMapSet (phashmap.c:127)

```
MiscPHMResult MiscPHashMapSet(MiscPHashMap HMap,
                              MiscPHashMapKey Key,
                              MiscPHashMapValue Value)
```

HMap: Hash map.

Key, Value: Pair of key,value to add.

Returns: MISC_P_HMSUCCESS in success and MISC_P_MFAILURE otherwise.

Description: Add a key and a value to a hash map. If the key exists, its value is replaced.

7.2.337 RLNewFromFile (mincover.c:1180)

RLNew

```
RLStruct *RLNewFromFile(const char *FileName)
```

FileName: The file name containing the required details

Returns: A new constrcuted RL

Description: Initializes an new RL using a file to describe the ranges.
See also: RLNew,

7.2.338 _AttrMallocAttributeHashNum (miscattr.c:1690)

```
IPAttributeStruct *_AttrMallocAttributeHashNum(IPAttrNumType AttribHashNum,
                                               IPAttributeType Type)
```

AttribHashNum: Hash number of newly created attribute.

Type: Type of newly created attribute.

Returns: The newly created attribute.

Description: Allocated a new attribute structure, by hash number.

7.2.339 _AttrMallocAttributeIDType (miscattr.c:1661)

```
IPAttributeStruct *_AttrMallocAttributeIDType(IPAttrNumType AttribID,
                                              IPAttributeType Type)
```

AttribID: ID of newly created attribute.

Type: Type of newly created attribute.

Returns: The newly created attribute.

Description: Allocated a new attribute structure, by hash number.
See also: _AttrMallocAttributeNameType,

7.2.340 _AttrMallocAttributeNameType (miscattr.c:1636)

```
IPAttributeStruct *_AttrMallocAttributeNameType(const char *Name,
                                                IPAttributeType Type)
```

Name: Name of newly created attribute.

Type: Type of newly created attribute.

Returns: The newly created attribute.

Description: Allocated a new attribute structure.
See also: _AttrMallocAttributeIDType,

7.2.341 _IrtImgVerifyAlignment (readimag.c:486)

```
IrtBType *_IrtImgVerifyAlignment(IrtBType *Data,
                                 int *MaxX,
                                 int *MaxY,
                                 int Alpha)
```

Data: Image data to verify alignment, in place.

MaxX: Current dimension of image. Might be changed after alignment.

MaxY: Current dimension of image.

Alpha: Do we have Alpha in the image?

Returns: The verified image data.

Description: Verifies the alignment of the image. Returned is either the input Data if aligned, or a new aligned copy (and Data is freed).

7.2.342 _wfopen (xgeneral.c:979)

```
FILE *_wfopen(const wchar_t *Filename, const wchar_t *Mode)
```

Filename: The file name.

Mode: The file mode.

Returns: Open file, or NULL if error.

Description: Emulate, under non Windows OSs, the function _wfopen, by coercing the wchar strings to ascii...
See also: _wpopen, _wgetenv,

7.2.343 _wgetenv (xgeneral.c:1042)

```
wchar_t *_wgetenv(const wchar_t *WName)
```

WName: Name of env. var. to fetch.

Returns: Env. var. value, or NULL if not found.

Description: Emulate, under non Windows OSs, the function _wgetenv, by coercing the wchar strings to ascii...
See also: _wfopen, _wpopen,

7.2.344 _wpopen (xgeneral.c:1011)

```
FILE *_wpopen(const wchar_t *Command, const wchar_t *Mode)
```

Command: The comamnd to execute.

Mode: The file mode.

Returns: Open file, or NULL if error.

Description: Emulate, under non Windows OSs, the function _wpopen, by coercing the wchar strings to ascii...
See also: _wfopen, _wgetenv,

7.2.345 getcwd (xgeneral.c:866)

```
char *getcwd(char *s, int Len)
```

s: Where to save current working direction.

Len: Length of s.

Returns: Same as s.

Description: Get current working directory - BSD4.3 style. *

7.2.346 movmem (xgeneral.c:558)

copy

```
void movmem(VoidPtr Src, VoidPtr Dest, int Len)
```

Src: Of block to copy.

Dest: Of block to copy.

Len: Of block to copy.

Returns: void

Description: Routine to move a block in memory. Unlike memcpy/bcopy, this routine should support overlaying blocks. This stupid implementation will copy it twice - to a temporary block and back again. The temporary block size will be allocated by demand.

7.2.347 searchpath (xgeneral.c:588)

```
const char *searchpath(const char *Name, char *FullPath)
```

Name: Of file to search for.

FullPath: Where returned path will be written. Same as returned value.

Returns: Complete file name of Name. Same as FullPath.

Description: Routine to search for a given file name.
See also: searchpathW,

7.2.348 searchpathW (xgeneral.c:639)

```
const wchar_t *searchpathW(const wchar_t *Name, wchar_t *FullPath)
```

Name: Of file to search for.

FullPath: Where returned path will be written. Same as returned value.

Returns: Complete file name of Name. Same as FullPath.

Description: Routine to search for a given file name.
See also: searchpath,

7.2.349 stricmp (xgeneral.c:803)

```
int stricmp(const char *s1, const char *s2)
```

s1, s2: The two strings to compare.

Returns: <0, 0, >0 according to the relation between s1 and s2.

Description: Routine to compare two strings, ignoring case.

7.2.350 strnicmp (xgeneral.c:775)

```
int strnicmp(const char *s1, const char *s2, int n)
```

s1, s2: The two strings to compare.

n: maximum number of characters to compare.

Returns: <0, 0, >0 according to the relation between s1 and s2.

Description: Routine to compare two strings, ignoring case, up to given length.

7.2.351 strstr (xgeneral.c:835)

```
char *strstr(const char *s, const char *Pattern)
```

s: To search for Pattern in.

Pattern: To search in s.

Returns: Address in s where Pattern was first found, NULL otherwise.

Description: Routine to search for a Pattern (no regular expression) in s. Returns address in s of first occurance of Pattern, NULL if non found.

Chapter 8

Multi variate functions, mvar_lib

8.1 General Information

This library holds functions to handle functions of arbitrary number of variables. In this context curves (univariate), surfaces (bivariate) and trivariates are special cases. This library provides a rich set of functions to manipulate freeform Bezier and/or NURBs multivariates. This library heavily depends on the cagd an symb libraries. Functions are provided to create, copy, and destruct multivariates, to extract isoparametric lower degree multivariates, to evaluate, refine and subdivide, to read and write multivariates, to differentiate, degree raise, make compatible and convert back and forth to/from curves, surfaces, and trivariates.

A multivariate has m orders, m Length prescriptions and, possibly, m knot vectors (if Bspline). In addition it contains an m dimensional volume of control points,

```
typedef struct MvarMVStruct {
    struct MvarMVStruct *Pnext;
    struct IPAttributeStruct *Attr;
    MvarGeomType GType;
    CagdPointType PType;
    int Dim;       /* Number of dimensions in this multi variate. */
    int *Lengths;              /* Dimensions of mesh size in multi-variate. */
    int *SubSpaces;    /* SubSpaces[i] = Prod(i = 0, i-1) of Lengths[i]. */
    int *Orders;               /* Orders of multi vararíate (Bspline only). */
    CagdBType *Periodic;           /* Periodicity - valid only for Bspline. */
    CagdRType *Points[CAGD_MAX_PT_SIZE];     /* Pointer on each axis vector. */
    CagdRType **KnotVectors;
} MvarMVStruct;
```

The interface of the library is defined in *include/mvar_lib.h*.

This library has its own error handler, which by default prints an error message and exit the program called **MvarFatalError**.

All globals in this library have a prefix of **Mvar**.

8.2 Library Functions

8.2.1 IritMvarDescribeError (mvar_err.c:110)

error handling

```
const char *IritMvarDescribeError(IritMvarFatalErrorType ErrorNum)
```

ErrorNum: Type of the error that was raised.

Returns: A string describing the error type.

Description: Returns a string describing a the given error. Errors can be raised by any member of this mvar library as well as other users. Raised error will cause an invocation of IritMvarFatalError function which decides how to handle this error. IritMvarFatalError can for example, invoke this routine with the error type, print the appropriate message and quit the program.

8.2.2 IritMvarFatalError (mvar_ftl.c:57)

error handling

```
void IritMvarFatalError(IritMvarFatalErrorType ErrID)
```

ErrID: Error type that was raised.

Returns: void

Description: Trap Mvar_lib errors right here. Provides a default error handler for the mvar library. Gets an error description using IritMvarDescribeError, prints it and exit the program using exit.

8.2.3 IritMvarSetFatalErrorFunc (mvar_ftl.c:29)

error handling

```
MvarSetErrorFuncType IritMvarSetFatalErrorFunc(MvarSetErrorFuncType ErrorFunc)
```

ErrorFunc: New error function to use.

Returns: Old error function reference.

Description: Sets the error function to be used by Mvar_lib.

8.2.4 MVHyperConeFromNPoints (mvcones.c:697)

```
int MVHyperConeFromNPoints(MvarNormalConeStruct *MVCone,
                           MvarVecStruct * const *Vecs,
                           int n)
```

MVCone: The result is to be placed here.

Vecs: Input vectors, prescribing n locations in R^n.

n: Size of array Vecs.

Returns: TRUE if successful, FALSE otherwise.

Description: Consturcts a hyper cone in R^n through n vectors specified by Vecs.
See also: MVHyperPlaneFromNPoints, MVHyperConeFromNPoints2, , MVHyperConeFromNPoints3,

8.2.5 MVHyperConeFromNPoints2 (mvcones.c:745)

```
int MVHyperConeFromNPoints2(MvarNormalConeStruct *MVCone,
                            MvarVecStruct * const *Vecs,
                            int m)
```

MVCone: The result is to be placed here.

Vecs: Input vectors, prescribing m locations in R^n.

m: Size of array Vecs, m < n.

Returns: TRUE if successful, FALSE otherwise.

Description: Consturcts a hyper cone in R^n through m (m < n) vectors specified by Vecs.
See also: MVHyperPlaneFromNPoints, MVHyperConeFromNPoints, MVHyperConeFromNPoints3,

8.2.6 MVHyperConeFromNPoints3 (mvcones.c:842)

```
int MVHyperConeFromNPoints3(MvarNormalConeStruct *MVCone,
                            MvarVecStruct * const *Vecs,
                            int m)
```

MVCone: The result is to be placed here.

Vecs: Input vectors, prescribing m locations in R^n.

m: Size of array Vecs, m < n.

Returns: TRUE if successful, FALSE otherwise.

Description: Consturcts a hyper cone in R^n through m (m < n) vectors specified by Vecs. Same functionality of MVHyperConeFromNPoints2 but more efficient, by solving for A A^T x = e, were e is [1, 1,..., 1], and having x being the linear combination of A's rows defining the cone axis.
See also: MVHyperPlaneFromNPoints, MVHyperConeFromNPoints, MVHyperConeFromNPoints2,

8.2.7 MVHyperPlaneFromNPoints (mvcones.c:643)

```
int MVHyperPlaneFromNPoints(MvarPlaneStruct *MVPlane,
                            MvarVecStruct * const *Vecs,
                            int n)
```

MVPlane: The result is to be placed here.

Vecs: Input vectors, prescribing n locations in R^n.

n: Size of array Vecs.

Returns: TRUE if successful, FALSE otherwise.

Description: Constructs a hyper plane in R^n through n locations specified by Vecs.
See also: MVHyperConeFromNPoints,

8.2.8 MVarBzrMVDivide (mvbzrsym.c:38)

product

```
void MVarBzrMVDivide(const MvarMVStruct *MV1,
                     const MvarMVStruct *MV2,
                     MvarMVStruct **Q,
                     MvarMVStruct **R)
```

MV1, MV2: The two multivariates to divide.

Q, R: Where to allocate the computed quotient and remainder.

Returns: void

Description: Given two scalar Bezier multivariates, MV1, MV2, calculates a quotient Bezier multivariate, Q, and a remainder Bezier multivarite, R, such that, MV1 = Q*MV2 + R. MV1 and MV2 are assumed to have the same dimension.

8.2.9 MVarExprTreeNormalCone (mvcones.c:2074)

```
MvarNormalConeStruct *MVarExprTreeNormalCone(MvarExprTreeStruct *Eqn)
```

Eqn: Multivariate to derive a cone bounding its normal space.

Returns: The resulting normal cone.

Description: Constructs a normal cone to a scalar multivariate expression tree. If the input multivariate is not scalar it is assumed to be of point type E(1+Dim), where Dim is the dimension of the MV. This scalar holds the gradient already in the Dim locations of the Points array in MV, in slots Points[2] to Points[Dim + 1], with Points[1] still holding the scalar field.
See also: MvarExprTreeConesOverlap,

8.2.10 MVarIsCrvInsideCirc (ms_circ.c:448)

```
int MVarIsCrvInsideCirc(const CagdCrvStruct *Crv,
                        CagdRType Center[2],
                        CagdRType Radius,
                        CagdRType Tolerance)
```

Crv: Curve to test for containment in the circle.

Center: Center of the circle to test against.

Radius: Radius of the circle to test against.

Tolerance: Of computation.

Returns: TRUE if Crv is indeed inside the circle, FALSE otherwise.

Description: Tests if a Crv is contained in the given prescribed circle. Computes the maximal distance of the curve from the circle center and compares that distance with circle radius.
See also: MvarMSCircOfThreeCurves, MvarMSCircOfThreeCurves, MvarMinSpanCirc,

8.2.11 MVarMVNormalCone (mvcones.c:1150)

```
MvarNormalConeStruct *MVarMVNormalCone(const MvarMVStruct *MV)
```

MV: Multivariate to derive a cone bounding its normal space.

Returns: A cone bounding the normal space of MV, or NULL if failed (i.e. angular span of normal space too large).

Description: Constructs a normal cone to a scalar multivariate MV. Note the normal space of the trivariate is assumed of dimension one, and the gradient of the multivariate is assumed to span the normal space. If the input multivariate is not scalar it is assumed to be of point type E(1+Dim), where Dim is the dimension of the MV. This scalar holds the gradient already in the Dim locations of the Points array in MV, in slots Points[2] to Points[Dim + 1], with Points[1] still holding the scalar field.
See also: MVarMVNormalCone2, MvarMVConesOverlap,

8.2.12 MVarMVNormalCone2 (mvcones.c:1219)

```
MvarNormalConeStruct *MVarMVNormalCone2(const MvarMVStruct *MV,
                                        CagdRType * const *GradPoints,
                                        int TotalLength,
                                        int *MaxDevIndex)
```

MV: Multivariate to derive a cone bounding its normal space.

GradPoints: Control vectors of gradient field.

TotalLength: Number of control vectors in gradient field.

MaxDevIndex: The index in GradPoints where maximal deviation occur will be kept here.

Returns: A cone bounding the normal space of MV, or NULL if failed (i.e. angular span of normal space too large).

Description: A second version of MVarMVNormalCone in which the control vectors are given directly. Note the normal space of the trivariate is assumed of dimension one, and the gradient of the multivariate is assumed to span the normal space. If the input multivariate is not scalar it is assumed to be of point type E(1+Dim), where Dim is the dimension of the MV. This scalar holds the gradient already in the Dim locations of the Points array in MV, in slots Points[2] to Points[Dim + 1], with Points[1] still holding the scalar field.
See also: MVarMVNormalCone, MvarMVConesOverlap,

8.2.13 MVarMVNormalConeMainAxis (mvcones.c:1294)

```
MvarNormalConeStruct *MVarMVNormalConeMainAxis(const MvarMVStruct *MV,
                                               MvarVecStruct **MainAxis)
```

MV: Multivariate to derive a cone bounding its normal space.

MainAxis: Main axis (principal component) of the normal cone's vectors distribution. Valid only if success.

Returns: A cone bounding the normal space of MV, or NULL if failed (i.e. angular span of normal space too large).

Description: Constructs a normal cone to a scalar multivariate MV. Note the normal space of the trivariate is assumed of dimension one, and the gradient of the multivariate is assumed to span the normal space. If the input multivariate is not scalar it is assumed to be of point type E(1+Dim), where Dim is the dimension of the MV. This scalar holds the gradient already in the Dim locations of the Points array in MV, in slots Points[2] to Points[Dim + 1], with Points[1] still holding the scalar field.
See also: MVarMVNormalConeMainAxis2, MVarMVNormalCone, MvarMVConesOverlap,

8.2.14 MVarMVNormalConeMainAxis2 (mvcones.c:1352)

```
MvarNormalConeStruct *MVarMVNormalConeMainAxis2(const MvarMVStruct *MV,
                                                CagdRType * const *GradPoints,
                                                int TotalLength,
                                                MvarVecStruct **MainAxis)
```

MV: Multivariate to derive a cone bounding its normal space.

GradPoints: Control vectors of gradient field.

TotalLength: Number of control vectors in gradient field.

MainAxis: Main axis (principal component) of the normal cone's vectors distribution. Allocated and valid only if success. Can be NULL in which case it is ignored.

Returns: A cone bounding the normal space of MV, or NULL if failed (i.e. angular span of normal space too large).

Description: A second version of MVarMVNormalConeMainAxis in which the control points are given directly.
See also: MVarMVNormalConeMainAxis, MVarMVNormalCone, MvarMVConesOverlap,

8.2.15 MVarProjNrmlPrmt2MVScl (mvaccess.c:418)

```
MvarMVStruct *MVarProjNrmlPrmt2MVScl(const CagdSrfStruct *Srf,
                                     const CagdSrfStruct *NrmlSrf,
                                     const MvarMVStruct *MVScl)
```

Srf: Surface to project promote and scale.

NrmlSrf: Normal field to project along.

MVScl: Scale field to scale with.

Returns: Resulting multivariate. *

Description: Performs the following steps in order:

1. Project Srf onto NrmlSrf by computing their inner product.
2. Promote the surface to a 4-variate with the surface the first 2 vars.
3. Scale the new 4-variate by a scalar product with MVScl.

8.2.16 MVarSmallestPrincipalDirection (mvcones.c:1028)

```
void MVarSmallestPrincipalDirection(MvarVecStruct *SPDVec,
                                    MvarVecStruct *ConeAxis,
                                    CagdRType * const *GradPoints,
                                    int TotalLength,
                                    int Dim)
```

SPDVec: Vector to be updated with the smallest principla component.

ConeAxis: Axis of cone bounding all the normal vectors in GradPoints. Assumed unit length.

GradPoints: The normal (Gradient) vectors to handle.

TotalLength: Number of normal vectors in GradPoints.

Dim: Dimension of Normal vectors. N

Returns: void

Description: Computes the smallest principal direction of a set of normal vectors.
See also: MVarMVMaximalDeviation,

8.2.17 Mvar2CntctCompute2CntctMotion (mv2cntct.c:3502)

2Contact Trace

```
MvarPolylineStruct *Mvar2CntctCompute2CntctMotion(const CagdCrvStruct *CCrvA,
                                                  const CagdCrvStruct *CCrvB,
                                                  CagdRType Step,
                                                  CagdRType SubdivTol,
                                                  CagdRType NumericTol)
```

CCrvA: Moving curve.

CCrvB: Obstacle Curves.

Step: Step size to use in the numeric tracing.

SubdivTol: Subdivision tolerance of Computation.

NumericTol: Numerical tolerance of computation.

Returns: Linked list of the solutions holding parameter value of two contact points and rotation in radian (u1, v1, u2, v2, theta).

Description: Computes the 2 Contact motion between two C^1 cont curves, CCrvA and CCrvB. The curves are assumed to be C1 periodic curve with open end condition. The algebraic conditions for 2contact are following (please refer the paper,"Precise Continuous Contact Motion for Planar Freeform Geometric Curves" for the details):

[CrvB(v1) -Rot(theta) * CrvA(u1)]_x = [CrvB(v2) - Rot(theta) * CrvA(u2)]_x [CrvB(v1) -Rot(theta) * CrvA(u1)]_y = [CrvB(v2) - Rot(theta) * CrvA(u2)]_y

```
det (Rot(theta) * CrvA'(u1), CrvB'(v1) = 0,
det (Rot(theta) * CrvA'(u2), CrvB'(v2) = 0.
```

See also:

8.2.18 Mvar2CtBuildBVH (mv2ctbvh.c:778)

```
Mvar2CtBVHStruct *Mvar2CtBuildBVH(CagdCrvStruct *Crv,
                                  CagdRType SubdivTol,
                                  CagdRType BvTol)
```

Crv: Crv to build a BVH.

SubdivTol: Subdivision tolerance for the construction.

BvTol: Bounding volume error tolerance for the construction.

Returns: Bounding volume hierarchy for a curve

Description: Build a bounding volume hierarchy for a curve.
See also:

8.2.19 Mvar2CtBuildCParamHierarchy (mv2ctaux.c:147)

```
void Mvar2CtBuildCParamHierarchy(CagdCrvStruct *Circle,
                                 Mvar2CtCParamStruct **Node,
                                 CagdRType Min,
                                 CagdRType Max,
                                 CagdRType Tol)
```

Circle: Unit circle.

Node: New node will be allocated herein.

Min, Max: Domain for the node.

Tol: Tolerance for building hierarchy.

Returns: void

Description: Build a hierarchy for rotation data structure.
See also:

8.2.20 Mvar2CtCheck2CtTrace (mv2ctaux.c:1056)

```
CagdBType Mvar2CtCheck2CtTrace(Mvar2CtBVNodeStruct *Nodes[4],
                               Mvar2CtCParamStruct *Cparam,
                               CagdRType Tol)
```

Nodes: Bounding volumes for curves.

Cparam: data structure for rotation.

Tol: Subdivision tolerance of checking.

Returns: FALSE if there is no solution in the domain, TRUE otherwise.

Description: Check if there exists 2 contact trace in the domain using BVH.
See also:

8.2.21 Mvar2CtConnectPeriodic (mv2ctaux.c:2292)

```
MvarPolylineStruct *Mvar2CtConnectPeriodic(MvarPolylineStruct *Polys,
                                           CagdRType Tol)
```

Polys: 2contact traces.

Tol: tolerance for the connection.

Returns: connected 2contact motion curves.

Description: Connect 2Contact traces that are not connected due to the parametrization of the periodic curve.
See also:

8.2.22 Mvar2CtCurvatureOverlap (mv2ctaux.c:90)

```
CagdBType Mvar2CtCurvatureOverlap(Mvar2CtBVNodeStruct *ANode,
                                  Mvar2CtBVNodeStruct *BNode,
                                  Mvar2CtCParamStruct *Cparam)
```

ANode: Bounding volume for a moving curve.

BNode: Bounding volume for a obstacle curve.

Cparam: Data structure for rotation.

Returns: FALSE if there is no curvature overlap TRUE otherwise.

Description: Check if two curves can have a same curvature by using BVH.
See also:

8.2.23 Mvar2CtExtractMVRegion (mv2ctaux.c:1507)

```
MvarMVStruct **Mvar2CtExtractMVRegion(MvarMVStruct **MVs,
                                      int MVNum,
                                      CagdRType *Min,
                                      CagdRType *Max)
```

MVs: Array of multivariates.

MVNum: Number of constraints (may be updated).

Min: Minimum domain values.

Max: Maximum domain values.

Returns: Extracted MVs.

Description: Extract sub-region of multivariate correspond to a given sub domain.
See also:

8.2.24 Mvar2CtFreeBVH (mv2ctbvh.c:853)

```
void Mvar2CtFreeBVH(Mvar2CtBVHStruct *Bvh)
```

Bvh: BVH to deallocate.

Returns: void

Description: Free bounding volume hierarchy.
See also:

8.2.25 Mvar2CtFreeCparam (mv2ctaux.c:234)

```
void Mvar2CtFreeCparam(Mvar2CtCParamStruct *Node)
```

Node: Mvar2CtCParamStruct node to free.

Returns: void

Description: Free Mvar2CtCParamStruct recursively.
See also:

8.2.26 Mvar2CtGetMiddlePt (mv2ctaux.c:1956)

```
MvarPtStruct *Mvar2CtGetMiddlePt(MvarPtStruct *PtList, int Length)
```

PtList: Linked list of the points.

Length: Length of the list.

Returns: Middle point of the list.

Description: Find a middle point of Point list.
See also:

8.2.27 Mvar2CtGetParentBVNode (mv2ctbvh.c:594)

```
void Mvar2CtGetParentBVNode(Mvar2CtBVNodeStruct *Node,
                            CagdRType Min,
                            CagdRType Max,
                            Mvar2CtBVNodeStruct **Parent)
```

Node: Data structure for rotation.

Min, Max: Domain range for testing.

Parent: Place for saving the result.

Returns: void

Description: Find smallest bounding volume includes domain range Min Max.
See also:

8.2.28 Mvar2CtGetParentCparam (mv2ctaux.c:207)

```
void Mvar2CtGetParentCparam(Mvar2CtCParamStruct *Node,
                            CagdRType Min,
                            CagdRType Max,
                            Mvar2CtCParamStruct **Parent)
```

Node: Data structure for rotation.

Min, Max: Domain range for testing.

Parent: Place for saving the result.

Returns: void

Description: Find smallest bounding volume includes the domain Min-Max.
See also:

8.2.29 Mvar2CtGetTheta (mv2ctaux.c:369)

```
CagdRType Mvar2CtGetTheta(CagdRType x, CagdRType y)
```

x, y: vector for computing rotation angle.

Returns: Rotation angle in radian.

Description: Compute a rotation angle in radian correspond (x, y) vector.
See also:

8.2.30 Mvar2CtInDomain (mv2ctaux.c:1555)

```
CagdBType Mvar2CtInDomain(CagdRType *Min,
                          CagdRType *Max,
                          MvarPtStruct *MPt)
```

Min: Minimum values of the domain.

Max: Maximum values of the domain.

MPt: Point to check.

Returns: TRUE if MPt is in the domain, FALSE otherwise.

Description: Check if a point is located in a given domain.
See also:

8.2.31 Mvar2CtIsConnectedNode (mv2ctaux.c:998)

```
CagdBType Mvar2CtIsConnectedNode(Mvar2CtBVNodeStruct *Node1,
                                 Mvar2CtBVNodeStruct *Node2)
```

Node1, Node2: Bounding volumes to test.

Returns: TRUE if connected, FALSE otherwise.

Description: Check if two bounding volumes are connected.
See also:

8.2.32 Mvar2CtIsPassing (mv2ctaux.c:1587)

```
CagdBType Mvar2CtIsPassing(CagdRType *Min,
                           CagdRType *Max,
                           MvarPolylineStruct *MPoly)
```

Min: Minimum values of the domain.

Max: Maximum values of the domain.

MPoly: MvarPolyline to check.

Returns: TRUE if MPoly passes the domain, FALSE otherwise.

Description: Check if a polyline passes a given domain.
See also:

8.2.33 Mvar2CtLineLineDist (mv2ctbvh.c:267)

```
CagdRType Mvar2CtLineLineDist(Mvar2CtLineStruct *A, Mvar2CtLineStruct *B)
```

A, B: Two line segments to compute distance.

Returns: Distance between A and B.

Description: Compute distance between two line segments.
See also:

8.2.34 Mvar2CtLinePointDist (mv2ctbvh.c:330)

```
CagdRType Mvar2CtLinePointDist(CagdPType P, CagdPType L[2])
```

P: Point to compute distance.

L: Two end points of line segment.

Returns: Minimum distance between a line segment and a point.

Description: Compute distance between point and line segment.
See also:

8.2.35 Mvar2CtNormalOverlap (mv2ctaux.c:410)

```
CagdBType Mvar2CtNormalOverlap(Mvar2CtBVNodeStruct *ANode,
                               Mvar2CtBVNodeStruct *BNode,
                               CagdRType RMin,
                               CagdRType RMax)
```

ANode, BNode: Bounding volumes to test.

RMin: Minimum rotation angle in radian.

RMax: Maximum rotation angle in radian.

Returns: FALSE if tangency condition does not satisfy TRUE otherwise.

Description: Check the tangency condition for 2 contact points using BVH.
See also:

8.2.36 Mvar2CtNormalOverlapBoth (mv2ctaux.c:571)

```
CagdBType Mvar2CtNormalOverlapBoth(Mvar2CtBVNodeStruct *ANode,
                                   Mvar2CtBVNodeStruct *BNode)
```

ANode, BNode: Nodes for normal overlapping test.

Returns: TRUE if normal cones overlap FALSE otherwise.

Description: Check normal overlap for both direction.
See also:

8.2.37 Mvar2CtPenetrationDepth (mv2ctbvh.c:1158)

```
CagdRType Mvar2CtPenetrationDepth(Mvar2CtBVHStruct *BvhA,
                                  Mvar2CtBVHStruct **BvhBs,
                                  int BSize,
                                  CagdRType Xtrans,
                                  CagdRType Ytrans,
                                  CagdRType Rot)
```

BvhA: Bvh for moving curve.

BvhBs: Bvh for obstacle curves.

BSize: Number of obstacle curves.

Xtrans: X translation.

Ytrans: Y translation.

Rot: Rotation angle in radian.

Returns: Maximum penetration depth, negative if moving curve penetrate obstacle curves positive, otherwise.

Description: Compute a maximum penetration depth of moving curve into obstacle curves using BVHs.
See also: MvarStewartPlatformSolve,

8.2.38 Mvar2CtReduce2CtDomain (mv2ctaux.c:1153)

```
void Mvar2CtReduce2CtDomain(Mvar2CtBVNodeStruct *Nodes[4],
                            Mvar2CtCParamStruct *Cparam,
                            CagdRType Min[5],
                            CagdRType Max[5],
                            int MinMax,
                            int FixedDir,
                            CagdRType Tol)
```

Nodes: Bounding volumes for curves.

Cparam: Data structure for rotation.

Min: Current minimum domain values.

Max: Current maximum domain values.

MinMax: 0 if fixed parameter is minimum parameter of the bounding volume, 1 if if fixed parameter is maximum parameter of the of the bounding volume.

FixedDir: Fixed dimension.

Tol: Subdivision tolerance of checking.

Returns: void

Description: Reduce the domain for solving 2 contact points at domain boundary using BVH.
See also:

8.2.39 Mvar2CtReduce3CtDomain (mv2ctaux.c:1308)

```
void Mvar2CtReduce3CtDomain(Mvar2CtBVNodeStruct *Nodes[6],
                            Mvar2CtCParamStruct *Cparam,
                            CagdRType Min[7],
                            CagdRType Max[7])
```

Nodes: Bounding volumes for curves.

Cparam: Data structure for rotation.

Min: Current minimum domain values.

Max: Current maximum domain values.

Returns: void

Description: Reduce the domain for solving 3 contact points using BVH.
See also:

8.2.40 Mvar2CtReduceRotExtremeDomain (mv2ctaux.c:1404)

```
void Mvar2CtReduceRotExtremeDomain(Mvar2CtBVNodeStruct *Nodes[4],
                                   Mvar2CtCParamStruct *Cparam,
                                   CagdRType Min[5],
                                   CagdRType Max[5],
                                   CagdRType Tol)
```

Nodes: Bounding volumes for curves.

Cparam: data structure for rotation.

Min: Current minimum domain values.

Max: Current maximum domain values.

Tol: Subdivision tolerance of checking.

Returns: void

Description: Reduce the domain for solving rotational extreme point for2 contact trace using BVH.
See also:

8.2.41 Mvar2CtRejectbyCurvature (mv2ctaux.c:960)

```
CagdBType Mvar2CtRejectbyCurvature(Mvar2CtBVNodeStruct *Node1,
                                   Mvar2CtBVNodeStruct *Node2)
```

Node1, Node2: Bounding volumes of curves to test.

Returns: TRUE if they don't satisfy the condition, FALSE otherwise.

Description: Check if pair of curves satisfy curvature condition for 2 contact.
See also:

8.2.42 Mvar2CtSetNodeId (mv2ctbvh.c:563)

```
void Mvar2CtSetNodeId(Mvar2CtBVNodeStruct *Node, int Id)
```

Node: Bounding volume to set id.

Id: id value to set.

Returns: void

Description: Set id of bounding volume recursively.
See also:

8.2.43 Mvar2CtSwapTrace (mv2ctaux.c:2039)

```
void Mvar2CtSwapTrace(MvarPolylineStruct *MPoly)
```

MPoly: Trace to swap.

Returns: void

Description: Swap the first and second parameter with third and fourth value .
See also:

8.2.44 Mvar2CtTraceBBox (mv2ctaux.c:2085)

```
void Mvar2CtTraceBBox(CagdRType *Min,
                      CagdRType *Max,
                      MvarPtStruct *SPt,
                      MvarPtStruct *EPt)
```

Min, Max: Minimum and maximum values of Bounding box.

SPt, EPt: Start and end point of the trace.

Returns: void

Description: Compute bounding box of a trace.
See also:

8.2.45 Mvar2CtTraceCollide (mv2ctaux.c:1987)

```
int Mvar2CtTraceCollide(MvarPolylineStruct *Poly1,
                        MvarPolylineStruct *Poly2)
```

Poly1, Poly2: Two traces to check.

Returns: 0 if there is no common contact 1 4 depending on the type of common contact point.

Description: Test whether two traces Poly1, Poly2 have common contact point.
See also:

8.2.46 Mvar2CtValidate2Ct (mv2ctaux.c:1627)

```
MvarPtStruct *Mvar2CtValidate2Ct(MvarPtStruct *MPts,
                                 Mvar2CtBVHStruct *BvhA,
                                 Mvar2CtBVHStruct **BvhBs,
                                 int BSize,
                                 CagdCrvStruct *Circle)
```

MPts: Linked list of curvature contacts.

BvhA: Bvh for moving curve.

BvhBs: Bvh for obstacle curves.

BSize: Number of obstacle curves.

Circle: Unit circle.

Returns: Linked list of curvature contact points having no penetration into obstacle curves.

Description: Check whether the 2 contact points cause inter-penetration into obstacle curves. The 2 contact points having inter-penetration are removed from the list.
See also:

8.2.47 Mvar2CtValidateCurvContact (mv2ctaux.c:1697)

```
MvarPtStruct *Mvar2CtValidateCurvContact(MvarPtStruct *MPts,
                                         Mvar2CtBVHStruct *BvhA,
                                         Mvar2CtBVHStruct **BvhBs,
                                         int BSize,
                                         int BIndex,
                                         CagdCrvStruct *Circle)
```

MPts: Linked list of curvature contacts.

BvhA: Bvh for moving curve.

BvhBs: Bvh for obstacle curves.

BSize: Number of obstacle curves.

BIndex: Index number of obstacle curve.

Circle: Unit circle.

Returns: Linked list of curvature contact points having no penetration into obstacle curves.

Description: Check whether the curvature contact points cause inter-penetration into obstacle curves. The curvature contact points having inter-penetration are removed from the list.
See also:

8.2.48 Mvar2CtValidateTraces (mv2ctaux.c:1759)

```
MvarPolylineStruct *Mvar2CtValidateTraces(MvarPolylineStruct *Polys,
                                          Mvar2CtBVHStruct *BvhA,
                                          Mvar2CtBVHStruct **BvhBs,
                                          int BSize,
                                          CagdCrvStruct *Circle)
```

Polys: 2contact traces.

BvhA: Bvh for moving curve.

BvhBs: Bvh for obstacle curves.

BSize: Number of obstacle curves.

Circle: Unit circle.

Returns: 2contact motion curves.

Description: Check if 2contact traces inter-penetrate the obstacle. The trace having inter-penetration is removed from the list.
See also:

8.2.49 Mvar3CircsInTriangles (mvarpack.c:50)

```
MvarPtStruct *Mvar3CircsInTriangles(const CagdPType Pts[3],
                                    CagdRType SubdivTol,
                                    CagdRType NumericTol)
```

Pts: 3 vertices of triangle in the plane (only XY coordinates).

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the surfaces.

NumericTol: Numeric tolerance of a possible numeric improvement stage. The numeric stage is employed if NumericTol < SubdivTol.

Returns: Points in R9 as (X1, Y1, R1, X2, Y2, R2, X3, Y3, R3). Each such solution is also tagged with "InTriangle" that is TRUE if all solution is inside the triangle.

Description: Given a triangles in the XY plane, specified by its 3 vertices Pts, Find the 3 circles packed inside the triangle (and tangent to it while also tangent to each other. This problem is also known as the (incorrect solution to the) "Malfatti Circles" problem.
See also:

8.2.50 Mvar6CircsInTriangles (mvarpck2.c:40)

```
MvarPtStruct *Mvar6CircsInTriangles(const CagdPType Pts[3],
                                    CagdRType SubdivTol,
                                    CagdRType NumericTol)
```

Pts: 3 vertices of triangle in the plane (only XY coordinates).

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the surfaces.

NumericTol: Numeric tolerance of a possible numeric improvement stage. The numeric stage is employed if NumericTol < SubdivTol.

Returns: Points in R9 as (x2, x4, x5, y1, y2, y3, y4, y5, y6).

Description: Given a triangles in the XY plane, specified by its 3 vertices Pts, Find the 6 circles packed inside the triangle (and tangent to it while also tangent to each other. The code to this function was synthesized automatically using the code below.
See also:

8.2.51 MvarAdjacentSrfSrfInter (selfintr.c:1401)

```
MvarPolylineStruct *MvarAdjacentSrfSrfInter(const CagdSrfStruct *Srf1,
                                            const CagdSrfStruct *Srf2,
                                            CagdSrfBndryType Srf1Bndry,
                                            CagdRType SubdivTol,
                                            CagdRType NumericTol)
```

Srf1, Srf2: The two adjacent surfaces to intersect.

Srf1Bndry: Boundary of Srf1 that is shared with Srf2. Srf2 Boundary Must be the reciprocal boundary. That is, if SrfBndry is UMin, Srf2's boundary will be UMax.

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the surfaces.

NumericTol: Numeric tolerance of a possible numeric improvement stage. The numeric stage is employed if NumericTol < SubdivTol.

Returns: List of intersection plins, as (u, v) parameter pairs into the two surfaces' domains. Points in R^4.

Description: Computes the intersection locations of two adjacent surfaces, that share an edge (a boundary curve). No intersections locations along the shared edge are returned. This case is common for adjacent patches in a B-spline surface.
See also: MvarBspSrfSelfInterDiagFactor, MvarBzrSrfSelfInterDiagFactor, , BzrSrfFactorExtremeRowCol,

8.2.52 MvarAre2MVsPossiblySharingBndry (mvar_aux.c:1774)

```
CagdBType MvarAre2MVsPossiblySharingBndry(const MvarMVStruct *MV1,
                                          const MvarMVStruct *MV2,
                                          int Dir,
                                          CagdBType *MV1Rev,
                                          CagdBType *MV2Rev,
                                          CagdRType Eps)
```

MV1, MV2: To detect if they possibly share a sub-MV in direction Dir.

Dir: To share for similary of a sub-MV along.

MV1Rev: TRUE if MV1 needs to be reversed in Dir to be merged with MV2.

MV2Rev: TRUE if MV2 needs to be reversed in Dir to be merged with MV1.

Eps: Tolerance of approximation.

Returns: TRUE if they are likely to share a common sub-MV.

Description: A fast detection if the given two multivariates possibly share a Sub-MV face, in Dir, as boundaries.
See also: MvarAre2MVsSharingBndry,

8.2.53 MvarAre2MVsSharingBndry (mvar_aux.c:1876)

```
int MvarAre2MVsSharingBndry(const MvarMVStruct *MV1,
                            int Dir1,
                            CagdBType MaxBndry1,
                            const MvarMVStruct *MV2,
                            int Dir2,
                            CagdBType MaxBndry2,
                            IrtRType Tolerance,
                            int *Modified)
```

MV1: First multivariate to consider.

Dir1: Axis to extract the sub-manifold boundary. Starts from 0.

MaxBndry1: TRUE for the maximal boundary in Dir1, FALSE for minimal.

MV2: Second multivariate to consider.

Dir2: Axis to extract the sub-manifold boundary. Starts from 0.

MaxBndry2: TRUE for the maximal boundary in Dir2, FALSE for minimal.

Tolerance: Tolerance of similarity.

Modified: If MVs requires degree raising, refinements, or reversing. See MvarMVSame3 for more. N

Returns: TRUE if similar, FALSE otherwise.

Description: Compare designated two boundaries of given two multivariates for similarity.
See also: CagdSrfAre2SrfsSharingBndry, MvarAre2MVsPossiblySharingBndry, MvarMVsSame3,

8.2.54 MvarBBoxOfCrossProd (mvarbbox.c:458)

```
void MvarBBoxOfCrossProd(const MvarBBoxStruct *BBox1,
                         const MvarBBoxStruct *BBox2,
                         MvarBBoxStruct *DCrossBBox)
```

BBox1, BBox2: Two bounding boxes to compute their cross product.

DCrossBBox: Where to place the returned result.

Returns: void

Description: Computes the cross product of two bounding boxes in R^3, fetching the possible values that could result from the cross product of the original multiariate data these bounding boxes bound. Returned bbox is a vector bbox with bounds on those possible cross product values. Computation is done by computing min/max value for each axis of the pair of bbox's cross product.
See also: MvarMVBBox, MvarBBoxOfDotProd,

8.2.55 MvarBBoxOfDotProd (mvarbbox.c:311)

```
void MvarBBoxOfDotProd(const MvarBBoxStruct *BBox1,
                       const MvarBBoxStruct *BBox2,
                       MvarBBoxStruct *DProdBBox)
```

BBox1, BBox2: Two bounding boxes to compute their inner product.

DProdBBox: Where to place the returned result.

Returns: void

Description: Computes the dot product of two bounding boxes in R^n, fetching the possible values that could result from the dot product of the original multiariate data these bounding boxes bound. Returned bbox is a scalar bbox with bounds on those possible dot product values. Computation is done by computing min/max value for each axis of the pair of bbox's and summing that up.

See also: MvarMVBBox, MvarBBoxOfCrossProd, MvarBBoxOfDotProd2,

8.2.56 MvarBBoxOfDotProd2 (mvarbbox.c:355)

```
void MvarBBoxOfDotProd2(const MvarBBoxStruct *BBox1,
                        const MvarBBoxStruct *BBox2,
                        MvarBBoxStruct *DProdBBox)
```

BBox1, BBox2: Two bounding boxes to compute their inner product.

DProdBBox: Where to place the returned result.

Returns: void

Description: Computes the dot product of two bounding boxes in R^n, fetching the possible values that could result from the dot product of the original multiariate data these bounding boxes bound. Returned bbox is a scalar bbox with bounds on those possible dot product values. Computation is done by enumerating all 2^n vertices of each bbox and computing their dot products. Slower than MvarBBoxOfDotProd.

See also: MvarMVBBox, MvarBBoxOfDotProd,

8.2.57 MvarBlendConvexMVMV (mvar_sym.c:1370)

product

symbolic computation

multivariates

```
MvarMVStruct *MvarBlendConvexMVMV(const MvarMVStruct *MV1,
                                  const MvarMVStruct *MV2,
                                  const MvarMVStruct *MVT)
```

MV1, MV2: Two vector multivariates to blend.

MVT: A multivariate indication the blend weight.

Returns: A multivariate representing the blend of MV1 and MV2 by MVT and 1 - MVT, respectively.

Description: Given two vector multivariates, MV1, and MV2, and a scalar multivariate, MVT, return a new multivariate equal to the convex blend function: MV1 * MVT + MV2 * (1 - MVT). The function does not verify that MVT is between zero and one.

See also: MvarBlendMVMV, MvarMVMultScalar, MvarMVMult,

8.2.58 MvarBlendMVMV (mvar_sym.c:1335)

product

symbolic computation

multivariates

```
MvarMVStruct *MvarBlendMVMV(const MvarMVStruct *MV1,
                            const MvarMVStruct *Scalar1,
                            const MvarMVStruct *MV2,
                            const MvarMVStruct *Scalar2)
```

MV1: 1st vector multivariates to blend.

Scalar1: 1st scalar multivariate, used as the blend MV1.

MV2: 2nd vector multivariates to blend.

Scalar2: 2nd scalar multivariate, used as the blend MV2.

Returns: A multivariate representing the blend of MV1 and MV2 by Scalar1 and Scalar2, respectively.

Description: Given two vector multivariates, MV1, and MV2, and two scalar multivariates, Scalar1, and Scalar2, return a new multivariate equal to the blend function: MV1 * Scalar1 + MV2 * Scalar2.

See also: MvarMVMultScalar, MvarMVMult,

8.2.59 MvarBndryMVsFromMV (mvarbbox.c:491)

multivariates

```
MvarMVStruct **MvarBndryMVsFromMV(const MvarMVStruct *MV)
```

MV: To extract the six boundary surfaces from.

Returns: A pointer to a dynamically allocated vector of 2N MVs representing the 2N boundaries of the multivariate MV, in order of U1Min, U1Max, U2Min, U2Max, ...

Description: Extracts the 2N boundary MVs of the given N dimensional tensor product MV.
See also: TrivBndrySrfsFromTV, CagdBndryCrvsFromSrf,

8.2.60 MvarBsctApplyCC (mvbiscon.c:246)

bisectors

```
int MvarBsctApplyCC(MvarVoronoiCrvStruct *Cv1,
                    MvarVoronoiCrvStruct **CCFreeCrvs)
```

Cv1: VoronoiCrvStruct

CCFreeCrvs: VoronoiCrvStruct for storing the resultant curves

Returns: TRUE or FALSE.

Description: Given the struct, this function says whether the bisector point satisfies the the following curvature constraints or not. Calculating the curvature constraints 1st constraint <P(t,r) - C1(t), k(t)N1(t)> - 1 < 0 2nd constraint <P(t,r) - C2(r), k(r)N2(r)> - 1 < 0
See also: MvarBsctApplyLL,

8.2.61 MvarBsctApplyLL (mvbiscon.c:124)

bisectors

```
MvarVoronoiCrvStruct *MvarBsctApplyLL(MvarVoronoiCrvStruct *Cv1)
```

Cv1: VoronoiCrvStruct

Returns: Returns the VoronoiCrvStruct after appying the LL constaint

Description: Given the struct, this function says whether the bisector point is to the left of the curve or not. Compute the constraints as dot products 1st constraint <P(t,r) - C1(t), N1(t)> > 0 2nd constraint <P(t,r) - C2(r), N2(r)> > 0
See also: MvarBsctIsCurveLL,

8.2.62 MvarBsctCheckFootPtEqualsMinDistPt (mvtrmpcr.c:709)

```
CagdBType MvarBsctCheckFootPtEqualsMinDistPt(CagdCrvStruct *Crv1,
                                             CagdRType *Pt,
                                             CagdPType BP)
```

Crv1: The input curve - CagdCrvStruct.

Pt: The input point - CagdRType.

BP: Bisector point - CagdPType.

Returns: Returns TRUE or FALSE.

Description: Given the Crv1, point Pt and the Bisector point (BP), this function says BP's footpoint is the minimum distance point to Crv1. Uses the function SymbDistCrvPoint from symb_lib.
See also: SymbDistCrvPoint,

8.2.63 MvarBsctComputeCrvPtBis (mvtrmpcr.c:223)

```
CagdPtStruct *MvarBsctComputeCrvPtBis(CagdCrvStruct *Crv,
                                      CagdRType *Pt,
                                      CagdRType t)
```

Crv: The input curve - CagdCrvStruct.

Pt: The input point - CagdRType.

t: Parameter on the curve Crv - CagdRType.

Returns: The identified bisector point.

Description: Given a Crv and a Pt, this function computes the point on the bisector using determinants for a particular parameter t on the curve.

See also: SymbCrvPtBisectorCrv2D,

8.2.64 MvarBsctComputeDenomOfP (mvtrmbis.c:49)

bisectors

skeleton

```
void MvarBsctComputeDenomOfP(CagdCrvStruct *Crv1Inp,
                             CagdCrvStruct *Crv2Inp,
                             CagdSrfStruct **DenomOut)
```

Crv1Inp, Crv2Inp: Two curves to compute bisectors for. Assumes E2 curves.

DenomOut: The resulting denominator surface is stored here.

Returns: void

Description: Computes the denominator of the bisector surface F3 of two given curves. Solve for the normal intersection surface in the plane and then substitute into (the bisector's correspondence is the zero set then).

```
          C1(s) + C2(t)
    < P - -------------, C1(t) - C2(s) > = 0.
                2
```

See also: SymbCrvCnvxHull, SymbCrvDiameter, SymbCrvBisectors, SymbCrvBisectorsSrf2, , SymbCrvBisectorsSrf, SymbCrvPtBisectorsSrf3D, SymbCrvCrvBisectorSrf3D.,

8.2.65 MvarBsctComputeF3 (mvtrmbis.c:142)

bisectors

skeleton

```
void MvarBsctComputeF3(CagdCrvStruct *Crv1Inp,
                       CagdCrvStruct *Crv2Inp,
                       CagdCrvStruct **Crv1Coerced,
                       CagdCrvStruct **Crv2Coerced,
                       CagdSrfStruct **F3,
                       CagdSrfStruct **L1,
                       CagdSrfStruct **L2,
                       CagdSrfStruct **CC1,
                       CagdSrfStruct **CC2)
```

Crv1Inp, Crv2Inp: Two curves to compute bisectors for. Assumes E2 curves.

Crv1Coerced: N.S.F.I.

Crv2Coerced: N.S.F.I.

F3: The resulting bisector surface is stored here.

L1: N.S.F.I.

L2: N.S.F.I.

CC1: N.S.F.I.

CC2: N.S.F.I.

Returns: void

Description: Computes the bisector surface definition of two curves. The result is a scalar surface whose zero set is the set of bisector(s) of the curves. Solve for the normal intersection surface in the plane and then substitute into (the bisector's correspondance is the zero set then).

```
          C1(s) + C2(t)
    < P - -------------, C1(t) - C2(s) > = 0.
                2
```

See also: SymbCrvCnvxHull, SymbCrvDiameter, SymbCrvBisectors, SymbCrvBisectorsSrf2, , SymbCrvBisectorsSrf, SymbCrvPtBisectorsSrf3D, SymbCrvCrvBisectorSrf3D,

8.2.66 MvarBsctComputeLowerEnvelope (mvlowenv.c:1045)

```
void MvarBsctComputeLowerEnvelope(MvarVoronoiCrvStruct *InputCurves,
                                  MvarVoronoiCrvStruct **LowerEnvelope)
```

InputCurves: A MvarVoronoiCrvStruct of monotone pieces.

LowerEnvelope: A MvarVoronoiCrvStruct of lower envelope.

Returns:

Description: Given the monotone curves, compute the lower envelope. This is the main calling function. The current implementation is an improved version that uses the auxiliary MvarLECrvStruct structure for efficiency and robustness (less solver calls).
See also: MvarBsctComputeLowerEnvelopeAux, MvarBsctComputeLowerEnvelopeOfOverlap, MvarBsctMergeLowerEnvelopes, MvarBsctSplitEnvelope1AtEnvelope2,

8.2.67 MvarBsctComputeXYFromBisTR (mvtrmbis.c:473)

```
CagdRType *MvarBsctComputeXYFromBisTR(CagdCrvStruct *Crv1,
                                      CagdRType t,
                                      CagdCrvStruct *Crv2,
                                      CagdRType r,
                                      CagdRType *InterPt)
```

Crv1: First curve.

t: Parameter value of first curve.

Crv2: Second curve.

r: Parameter value of second curve.

InterPt: Bisector point from given tr.

Returns: Bisector point from given tr. Same as InterPt

Description: Computes the bisector point given tr-values

8.2.68 MvarBsctCrvPtCurvature (mvtrmpcr.c:473)

```
CagdCrvStruct *MvarBsctCrvPtCurvature(CagdCrvStruct *Crv,
                                      CagdRType *Pt,
                                      CagdRType Alpha)
```

Crv: The input curve - CagdCrvStruct.

Pt: The input point - CagdRType.

Alpha: Parameter of the alpha-sector - CagdRType. 0.5 for bisector.

Returns: CagdCrvStruct of the resultant.

Description: Given the struct, this function says whether the bisector point satisfies the Curvature constraint.
See also: MvarBsctCrvPtLeft,

8.2.69 MvarBsctCrvPtLeft (mvtrmpcr.c:283)

```
CagdCrvStruct *MvarBsctCrvPtLeft(CagdCrvStruct *Crv,
                                 CagdRType *Pt,
                                 CagdRType Alpha)
```

Crv: The input curve - CagdCrvStruct.

Pt: The input point - CagdRType.

Alpha: Parameter of the alpha-sector - CagdRType. 0.5 for bisector.

Returns: CagdCrvStruct of the resultant.

Description: Given the struct, this function says whether the bisector point satisfies the Left constraint.
See also: MvarBsctCrvPtCurvature,

8.2.70 MvarBsctCurveLeft (mvvorcrv.c:258)

```
void MvarBsctCurveLeft(MvarVoronoiCrvStruct *Cv, MvarPtStruct *Res)
```

Cv: Input curve.

Res: Output stored here.

Returns: void

Description: A Geometric Primitive Function that returns the leftmost point of Cv.
See also: MvarBsctCurveRight,

8.2.71 MvarBsctCurveRight (mvvorcrv.c:356)

```
void MvarBsctCurveRight(MvarVoronoiCrvStruct *Cv, MvarPtStruct *Res)
```

Cv: input curve

Res: ouput stored here

Returns: void

Description: A Geometric Primitive Function that returns the rightmost point of Cv.
See also: MvarBsctCurveLeft,

8.2.72 MvarBsctCv1IsYSmallerAt (mvvorcrv.c:467)

```
int MvarBsctCv1IsYSmallerAt(MvarVoronoiCrvStruct *Cv1,
                            MvarVoronoiCrvStruct *Cv2,
                            MvarPtStruct *MidPoint)
```

Cv1, Cv2: Input curves.

MidPoint: Mid-point parameter.

Returns: Returns TRUE or FALSE.

Description: A Geometric Primitive Function that identifies whether Cv1 has Y min. at given mid-point parameter, assuming both curves are in range.

8.2.73 MvarBsctDenomPtCrvBis (mvtrmpcr.c:82)

bisector

```
CagdCrvStruct *MvarBsctDenomPtCrvBis(CagdCrvStruct *Crv,
                                     CagdPType Pt,
                                     CagdRType Alpha)
```

Crv: Planar curve to compute its bisector curve with Pt.

Pt: A point in the plane to compute its bisector with Crv.

Alpha: Alpha-sector ratio (0.5 for a bisector).

Returns: The bisector curve, in the XY plane.

Description: Computes the denominator of the alpha-/bi-sector curve of a planar curve and a point, all in the XY plane. The result is the denominator of the solution to the following two linear equations in alpha-/bi-sector's two unknowns, the x and y coefficients:

```
<C'(t),     B(t)> = <C'(t), C(t)>
<C(t) - Pt, B(t)> = <C(t) - Pt, a Pt + (1 - a) C(t)>
```

where a is the Alpha of the alpha-sector, 0.5 for a bisector, Pt is the point entity, C(t) is the curve entity and B(t) is the saught bisector.

See also: SymbCrvDiameter, SymbCrvCnvxHull, SymbCrvBisectorsSrf, , SymbCrvCrvBisectorSrf3D, SymbSrf-PtBisectorSrf3D, SymbCrvPtBisectorSrf3D,

8.2.74 MvarBsctGetAllIntersectionPoints (mvvorcrv.c:895)

```
void MvarBsctGetAllIntersectionPoints(MvarVoronoiCrvStruct *Cv1,
                                      MvarVoronoiCrvStruct *Cv2,
                                      MvarPtStruct **Points)
```

Cv1, Cv2: Input curves.

Points: The resultant points.

Returns: void

Description: A Geometric Primitive Function that computes the equidistant points of three curves Cv1, Cv2 and Cv2 (third one also the second curve). Currently implemented just by calling Skel2DEqPts3Crvs (and purges away solutions). ToDo: implement a more efficient version, based on the already calculated F3.
See also: MvarBsctSkel2DEqPts3Crvs,

8.2.75 MvarBsctIsCrvLeftToLine (mvtrmpcr.c:1069)

```
CagdRType MvarBsctIsCrvLeftToLine(CagdCrvStruct *Crv,
                                  CagdRType *Pt,
                                  CagdPType LeftNormal)
```

Crv: The input curve - CagdCrvStruct.

Pt: The discontinuity point on Crv.

LeftNormal: Left normal at the point Pt.

Returns: Return the value of the expression.

Description: This function returns the value of the cross product between two vectors. First vector is between the point Pt and the midpoint of Crv and the second one is between the point and the LeftNormal.
See also: MvarBsctTrimCrvPt,

8.2.76 MvarBsctIsCurveLL (mvbiscon.c:35)

bisectors

```
int MvarBsctIsCurveLL(MvarVoronoiCrvStruct *Cv)
```

Cv: MvarVoronoiCrvStruct

Returns: TRUE or FALSE.

Description: Given the struct, this function says whether the bisector point is to the left of the curve or not. Compute the constraints as dot products 1st constraint <P(t,r) - C1(t), N1(t)> > 0 2nd constraint <P(t,r) - C2(r), N2(r)> > 0
See also: MvarBsctApplyLL,

8.2.77 MvarBsctIsXSmaller (mvvorcrv.c:175)

```
int MvarBsctIsXSmaller(MvarPtStruct *P1, MvarPtStruct *P2)
```

P1, P2: Points as input.

Returns: Returns true or false.

Description: A Geometric Primitive Function that identifies which x-coord of the two points P1 and P2 is smaller.

8.2.78 MvarBsctNewFindZeroSetOfSrfAtParam (mvsplmon.c:206)

```
MvarPtStruct *MvarBsctNewFindZeroSetOfSrfAtParam(CagdSrfStruct *Srf,
                                                 CagdRType Param,
                                                 CagdSrfDirType Dir,
                                                 CagdRType MvarBsctSubdivTol,
                                                 CagdRType MvarBsctNumerTol,
                                                 CagdBType ShouldCheckEndPoints)
```

Srf: Implicit surface definition.

Param: Parametric value for computation.

Dir: U or V direction for computation

MvarBsctSubdivTol: Subdivision tolerance for the multivariate solver.

MvarBsctNumerTol: Numeric tolerance of a possible numeric improvement stage. The numeric stage is employed if MvarBsctNumerTol < MvarBsctSubdivTol.

ShouldCheckEndPoints: To include end points also for checking.

Returns: The computed values

Description: Find the zeroset value at the given paramter of the given surface

8.2.79 MvarBsctPurgeAwayLLAndCCConstraints (mvbiscon.c:358)

bisectors

```
MvarVoronoiCrvStruct *MvarBsctPurgeAwayLLAndCCConstraints(MvarVoronoiCrvStruct
                                                          *InputCrvs)
```

InputCrvs: A VoronoiCrvStruct of monotone pieces

Returns: VoronoiCrvStruct for storing the resultant

Description: Given the struct, this function says whether the bisector point satisfies the LL and Curvature constraints

See also: MvarBsctApplyLL, MvarBsctApplyCC,

8.2.80 MvarBsctSkel2DEqPts3Crvs (mvtrmbis.c:553)

```
MvarPtStruct *MvarBsctSkel2DEqPts3Crvs(CagdCrvStruct *Crv1Inp,
                                       CagdCrvStruct *Crv2Inp,
                                       CagdCrvStruct *Crv3Inp)
```

Crv1Inp, Crv2Inp, Crv3Inp: The three input primitives to consider.

Returns: A linked list of all equadistant points computed, or NULL if none found.

Description: Formulate multivariate constraints for the points that are at equal distance from the three primitives and solve for them. Assumes that the three primitives are lines or curves and that they do NOT intersect. Modified from from skel2d.c in mvar_lib to suit our needs

8.2.81 MvarBsctSplitCurve (mvvorcrv.c:984)

```
void MvarBsctSplitCurve(MvarVoronoiCrvStruct *Cv,
                        MvarPtStruct *SplitPt,
                        MvarVoronoiCrvStruct **CvLeft,
                        MvarVoronoiCrvStruct **CvRight)
```

Cv: Input curve.

SplitPt: The parameter at which to split.

CvLeft, CvRight: The resultant split curves.

Returns: void

Description: A Geometric Primitive Function that splits the given curve into two - left and right curve of a given parameter t.

8.2.82 MvarBsctSplitImplicitCrvToMonotonePieces (mvsplmon.c:1146)

```
void MvarBsctSplitImplicitCrvToMonotonePieces(CagdSrfStruct *Srf,
                                              CagdSrfStruct **OutLst,
                                              CagdRType MvarBsctSubdivTol,
                                              CagdRType MvarBsctNumerTol)
```

Srf: Implicit surface definition.

OutLst: Output list of uv-monotone surfaces.

MvarBsctSubdivTol: Subdivision tolerance for multivariate solver.

MvarBsctNumerTol: Numeric tolerance of a possible numeric improvment stage. The numeric stage is employed if MvarBsctNumerTol < MvarBsctSubdivTol.

Returns: void

Description: Splits the bivariate zero set Srf=0 (implicit form) into uv-monotone pieces. The recursive algorithm is based on the paper by Keyser et al.

8.2.83 MvarBsctTrimCrvPt (mvtrmpcr.c:1132)

bisectors

```
CagdCrvStruct *MvarBsctTrimCrvPt(CagdCrvStruct *Crv,
                                 CagdRType *Pt,
                                 CagdRType Alpha,
                                 CagdCrvStruct *BaseCrv)
```

Crv: The input curve - CagdCrvStruct.

Pt: The input point - CagdRType.

Alpha: Parameter of the alpha-sector - CagdRType. 0.5 for bisector.

BaseCrv: The Crv for which the trimming is sought along its discontinuity point.

Returns: Return the list of trimmed bisector curves.

Description: This function returns the radial lower envelope if the input is a (list of) crv (s) and a point or returns the trimmed bisector if the input is a pair of crv and a point.
See also: MvarBsctTrimCrvPtPair, MvarBsctCrvPtLeft, MvarBsctCrvPtCurvature, , SymbCrvsLowerEnvelop,

8.2.84 MvarBsctTrimCrvPtPair (mvtrmpcr.c:770)

```
CagdCrvStruct *MvarBsctTrimCrvPtPair(CagdCrvStruct *Crv,
                                     CagdRType *Pt,
                                     CagdRType Alpha)
```

Crv: The input curve - CagdCrvStruct.

Pt: The input point - CagdRType.

Alpha: Parameter of the alpha-sector - CagdRType. 0.5 for bisector.

Returns: Return the list of trimmed bisector curves.

Description: Given the struct, this function says whether the bisector point satisfies the LL and Curvature constraint for a crv and a point pair and trims it to the minimum.
See also: MvarBsctTrimCrvPt, MvarBsctCrvPtLeft, MvarBsctCrvPtCurvature,

8.2.85 MvarBsctTrimCurveBetween (mvlowenv.c:115)

```
void MvarBsctTrimCurveBetween(MvarVoronoiCrvStruct *Cv,
                              MvarPtStruct *Pt1,
                              MvarPtStruct *Pt2,
                              MvarVoronoiCrvStruct **TrimmedCurve)
```

Cv: Given a MvarVoronoiCrvStruct Cv.

Pt1: A MvarPtStruct.

Pt2: A MvarPtStruct.

TrimmedCurve: A MvarVoronoiCrvStruct of the resultant.

Returns: void

Description: Splitting the Cv at the given points Pt1 and Pt2.
See also: MvarBsctComputeLowerEnvelope, MvarBsctMergeLowerEnvelopes, MvarBsctComputeLowerEnvelopeAux, MvarBsctComputeLowerEnvelopeOfOverlap,

8.2.86 MvarBsctTrimSurfaceByUVBbox (mvsplmon.c:369)

```
CagdSrfStruct *MvarBsctTrimSurfaceByUVBbox(CagdSrfStruct *Srf,
                                           CagdBBoxStruct UVBbox)
```

Srf: Implicit surface definition.

UVBbox: Given box dimensions.

Returns: The trimmed surface

Description: Trims the given surface Srf using the given box dimensions. Uses CagdSrfRegionFromSrf twice.

8.2.87 MvarBspCrvInterpVecs (mvar_int.c:58)

interpolation

least square approximation

```
CagdCrvStruct *MvarBspCrvInterpVecs(const MvarVecStruct *vecList,
                                    int Order,
                                    int CrvSize,
                                    CagdParametrizationType ParamType,
                                    CagdBType Periodic)
```

vecList: List of points to interpolate/least square approximate. All points are assumed of same dimension (not tested.).

Order: Of interpolating/approximating curve.

CrvSize: Number of degrees of freedom (control points) of the interpolating/approximating curve.

ParamType: Type of parametrization.

Periodic: Constructed curve should be Periodic. Periodic necessitates uniform knot sequence in ParamType.

Returns: Constructed interpolating/approximating curve.

Description: Given a set of points, vecList, computes a Bspline curve of order Order that interpolates or least square approximates the set of points. The size of the control polygon of the resulting Bspline curve defaults to the number of points in PtList (if CrvSize = 0). However, this number is can smaller to yield a least square approximation. The created curve can be parametrized as specified by ParamType.
See also: BspCrvInterpPts, BspCrvInterpolate, BspCrvInterpPts2,

8.2.88 MvarBspMVDerive (mvar_der.c:176)

multi-variates

```
MvarMVStruct *MvarBspMVDerive(const MvarMVStruct *MV,
                              MvarMVDirType Dir,
                              CagdBType DeriveScalar)
```

MV: Multi-Variate to differentiate.

Dir: Direction of differentiation.

DeriveScalar: TRUE to differentiate each channel independently. FALSE to treat geometry as a rational differentiation.

Returns: Differentiated multi-variate in direction Dir. A B-spline multi-variate.

Description: Given a B-spline multi-variate, computes its partial derivative multi-variate in direction Dir. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one. Then:

```
Q(i) = (k - 1) * (P(i+1) - P(i)) / (Kv(i + k) - Kv(i + 1)), i = 0 to k-2.
```

This function computes the derivative of a rational function component- wise without taking into consideration the quotient rule.

See also: MvarMVDeriveBound, MvarBzrMVDerive, MvarMVDerive,

8.2.89 MvarBspMVDeriveAllBounds (mvar_der.c:583)

multi-variates

```
void MvarBspMVDeriveAllBounds(const MvarMVStruct *MV, IrtMinMaxType *MinMax)
```

MV: Multi-Variate to differentiate.

MinMax: Bounds on the derivative values of MV in all directions.

Returns: void

Description: Given a scalar B-spline multi-variate, computes bounds to its partial derivative in all directions. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one. Then:

```
Q(i) = (k - 1) * (P(i+1) - P(i)) / (Kv(i + k) - Kv(i + 1)), i = 0 to k-2.
```

See also: MvarMVDerive, MvarBzrMVDeriveBound, MvarMVDeriveBound,

8.2.90 MvarBspMVDeriveBound (mvar_der.c:410)

multi-variates

```
void MvarBspMVDeriveBound(const MvarMVStruct *MV,
                          MvarMVDirType Dir,
                          CagdRType MinMax[2])
```

MV: Multi-Variate to differentiate.

Dir: Direction of differentiation.

MinMax: Bounds on the derivative values of MV in direction Dir.

Returns: void

Description: Given a scalar B-spline multi-variate, computes bounds to its partial derivative in direction Dir. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one. Then:

```
Q(i) = (k - 1) * (P(i+1) - P(i)) / (Kv(i + k) - Kv(i + 1)), i = 0 to k-2.
```

See also: MvarMVDerive, MvarBzrMVDeriveBound, MvarMVDeriveBound,

8.2.91 MvarBspMVDeriveRational (mvbspsym.c:203)

derivatives

```
MvarMVStruct *MvarBspMVDeriveRational(const MvarMVStruct *MV,
                                      MvarMVDirType Dir)
```

MV: Rational B-spline multivariate to differentiate.

Dir: Direction of Differentiation.

Returns: Differentiated rational B-spline multivariate.

Description: Given a rational B-spline multivariates - computes its derivative surface in direction Dir, using the quotient rule for differentiation.
See also: MvarMVDerive, MvarBzrMVDerive, MvarBspMVDerive, MvarBzrMVDeriveRational,

8.2.92 MvarBspMVDeriveScalar (mvar_der.c:283)

derivatives

```
MvarMVStruct *MvarBspMVDeriveScalar(const MvarMVStruct *MV, MvarMVDirType Dir)
```

MV: To differentiate.

Dir: Direction of differentiation. Either U or V.

Returns: Differentiated multi-variate.

Description: Given a Bezier multi-variate, computes its partial derivative multi-variate in direction Dir. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one. Then:

```
Q(i) = (k - 1) * (P(i+1) - P(i)), i = 0 to k-2.
```

For a Euclidean surface this is the same as MvarBzrMVDerive but for a rational multivar the returned multivar is not the vector field but simply the derivatives of all the multivar's coefficients, including the weights.
See also: MvarMVDeriveBound, MvarMVDerive, MvarBspMVDerive, MvarBspMVDerive, MvarBzrMVDeriveScalar,

8.2.93 MvarBspMVHasOpenEC (mvar_aux.c:1393)

open end conditions

```
CagdBType MvarBspMVHasOpenEC(const MvarMVStruct *MV)
```

MV: To check for open end conditions.

Returns: TRUE, if MV has open end conditions in all directions, FALSE otherwise.

Description: Returns TRUE iff the given B-spline multivariate has open end conditions in all direction directions.
See also: BspCrvHasOpenEC, MvarBspMVHasOpenECInDir, MvarBspMVHasOpenECInDir,

8.2.94 MvarBspMVHasOpenECInDir (mvar_aux.c:1364)

open end conditions

```
CagdBType MvarBspMVHasOpenECInDir(const MvarMVStruct *MV, MvarMVDirType Dir)
```

MV: To check for open end conditions.

Dir: Direction to test for open end conditions.

Returns: TRUE, if MV has open end conditions in Dir, FALSE otherwise.

Description: Returns TRUE iff the given B-spline multivariate has open end conditions in the specified direction Dir.
See also: BspCrvHasOpenEC, MvarBspMVHasOpenEC, MvarBspMVIsPeriodic, , MvarBspMVIsPeriodicInDir,

8.2.95 MvarBspMVInteriorKnots (mvar_aux.c:1487)

Internal knots

Bezier

```
int MvarBspMVInteriorKnots(const MvarMVStruct *MV, CagdRType *Knot)
```

MV: To check for interior knots.

Knot: Where to return an interior knot if found one.

Returns: -1 if MV has no interior knots, Axis of interior knot otherwise.

Description: Returns -1 if the given B-spline multivariate has not interior knots in no direction. Otherwise, return the direction that has an interior knot and returns the knot value in Knot.
See also: BspCrvHasOpenEC, MvarBspMVHasOpenEC, MvarBspMVHasOpenECInDir, , MvarBspMVIsPeriodicInDir,

8.2.96 MvarBspMVIsPeriodic (mvar_aux.c:1455)

periodic end conditions

```
CagdBType MvarBspMVIsPeriodic(const MvarMVStruct *MV)
```

MV: To check for periodic end conditions.

Returns: TRUE, if MV has periodic end conditions in some Dir, FALSE otherwise.

Description: Returns TRUE iff the given B-spline multivariate has periodic end conditions in at least one direction.
See also: BspCrvHasOpenEC, MvarBspMVHasOpenEC, MvarBspMVHasOpenECInDir, , MvarBspMVIsPeriodicInDir, MvarBspMVInteriorKnots,

8.2.97 MvarBspMVIsPeriodicInDir (mvar_aux.c:1431)

periodic end conditions

```
CagdBType MvarBspMVIsPeriodicInDir(const MvarMVStruct *MV, MvarMVDirType Dir)
```

MV: To check for periodic end conditions.

Dir: Direction to test for periodic end conditions.

Returns: TRUE, if MV has periodic end conditions in Dir, FALSE otherwise.

Description: Returns TRUE iff the given B-spline multivariate has periodic end conditions in the specified direction Dir.
See also: BspCrvHasOpenEC, MvarBspMVHasOpenEC, MvarBspMVHasOpenECInDir, , MvarBspMVIsPeriodic,

8.2.98 MvarBspMVKnotInsertNDiff (mvar_ref.c:88)

multi-variates

```
MvarMVStruct *MvarBspMVKnotInsertNDiff(const MvarMVStruct *MV,
                                       MvarMVDirType Dir,
                                       int Replace,
                                       CagdRType *t,
                                       int n)
```

MV: Multi-variate to refine according to t in direction Dir.

Dir: Direction of refinement. Either U or V or W.

Replace: If TRUE t is a knot vector exactly in the length of the knot vector in direction Dir in MV and t simply replaces that knot vector. If FALSE, the knot vector in direction Dir in MV is refined by adding all the knots in t.

t: Knot vector to refine/replace the knot vector of MV in direction Dir.

n: Length of vector t.

Returns: The refined multi-variate. A Bspline multi-variate.

Description: Given a Bspline multi-variate, inserts n knots with different values as defined by t. If, however, Replace is TRUE, the knot are simply replacing the current knot vector in the prescribed direction.

8.2.99 MvarBspMVMult (mvbspsym.c:58)

product

```
MvarMVStruct *MvarBspMVMult(const MvarMVStruct *CMV1, const MvarMVStruct *CMV2)
```

CMV1, CMV2: The two multivariates to multiply.

Returns: The product MV1 * MV2 coordinate-wise.

Description: Given two B-spline multivariates - multiply them coordinate-wise. The two multivariates are promoted to same point type before multiplication can take place. See also BspMultComputationMethod.

8.2.100 MvarBspMVNew (mvar_gen.c:173)

multi-variates

allocation

```
MvarMVStruct *MvarBspMVNew(int Dim,
                           const int *Lengths,
                           const int *Orders,
                           MvarPointType PType)
```

Dim: Number of dimensions of this multivariate.

Lengths: Of control mesh in each of the dimensions. Vector of size Dim.

Orders: Of multi variate function in each of the dimensions.

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform multi-variate Bspline.

Description: Allocates the memory required for a new Bspline multi-variate.
See also: MvarMVFree, MvarBzrMVNew, MvarMVNew, MvarPwrMVNew,

8.2.101 MvarBspMVSubdivAtParam (mvar_sub.c:233)

multi-variates

```
MvarMVStruct *MvarBspMVSubdivAtParam(const MvarMVStruct *MV,
                                     CagdRType t,
                                     MvarMVDirType Dir)
```

MV: BsplineMulti-Variate to subdivide.

t: Parameter to subdivide at.

Dir: Direction of subdivision.

Returns: A list of two multi-variates, result of the subdivision.

Description: Given a Bspline multi-variate, subdivides it at parameter value t in direction Dir.
See also: MvarMVSubdivAtParam, MvarBzrMVSubdivAtParam,

8.2.102 MvarBspMVSubdivAtParamOneSide (mvar_sub.c:638)

multi-variates

```
MvarMVStruct *MvarBspMVSubdivAtParamOneSide(const MvarMVStruct *MV,
                                            CagdRType t,
                                            MvarMVDirType Dir,
                                            IrtBType LeftSide)
```

MV: BsplineMulti-Variate to subdivide.

t: Parameter to subdivide at.

Dir: Direction of subdivision.

LeftSide: TRUE to only fetch left half, FALSE to fetch right half.

Returns: A single multi-variate, result of the subdivision.

Description: Given a B-spline multi-variate, subdivides it at parameter value t in direction Dir.
See also: MvarMVSubdivAtParam, MvarBzrMVSubdivAtParam,

8.2.103 MvarBspMultComputationMethod (mvbspsym.c:37)

product

```
int MvarBspMultComputationMethod(int BspMultUsingInter)
```

BspMultUsingInter: If TRUE, B-spline product is computed by setting an interpolation problem. Otherwise, by decomposing the B-spline geometry to Bezier geometry.

Returns: Previous setting.

Description: Sets method of B-spline product computation.

8.2.104 MvarBspSrfSelfInterDiagFactor (selfintr.c:1650)

```
MvarPolylineStruct *MvarBspSrfSelfInterDiagFactor(const CagdSrfStruct *Srf,
                                                  CagdRType SubdivTol,
                                                  CagdRType NumericTol)
```

Srf: The surface to derive its self intersections.

SubdivTol: Tolerance of the first zero set finding subdivision stage.

NumericTol: Tolerance of the second zero set finding numeric stage.

Returns: List of self intersection pllns, as (u, v) parameter pairs into the surfaces' domain. Points in R^4.

Description: Given a B-spline surface, S, compute its self intersections, if any, by dividing it at all internal knots and examining all patches against all other patches. Diagonal patches should also be examined against themselves with the aid of MvarBzrSrfSelfInterDiagFactor. Adjacent patches are sharing an edge (a curve) and hence spacial care is taken in those cases to eliminate and ignore that shared curve, in MvarAdjacentSrfSrfInter.
See also: MvarAdjacentSrfSrfInter, MvarBzrSrfSelfInterDiagFactor,

8.2.105 MvarBuildParamMV (mvar_gen.c:270)

```
MvarMVStruct *MvarBuildParamMV(int Dim, int Dir, CagdRType Min, CagdRType Max)
```

Dim: Number of dimensions of the MV parameter function.

Dir: Direction of this parameter (between 0 and Dim-1).

Min, Max: The range of this parameter.

Returns: Constructed parameter MV function.

Description: Construct an MV that serves as a parameter function in direction Dir.
See also:

8.2.106 MvarBzrLinearInOneDir (mvar_aux.c:1526)

```
MvarMVStruct *MvarBzrLinearInOneDir(int Dim, int Dir, MvarPointType PType)
```

Dim: DImension f the sought multivariate Bezier.

Dir: The direction that is to be linear, between 0 and Dim-1.

PType: Type of points of this new multivariate.

Returns: The constructed multivariate.

Description: Creates a Bezier multivariate that is constant in all directions but one.
See also:

8.2.107 MvarBzrMVDerive (mvar_der.c:74)

multi-variates

```
MvarMVStruct *MvarBzrMVDerive(const MvarMVStruct *MV,
                              MvarMVDirType Dir,
                              CagdBType DeriveScalar)
```

MV: Multi-Variate to differentiate.

Dir: Direction of differentiation.

DeriveScalar: TRUE to differentiate each channel independently. FALSE to treat geometry as a rational differentiation.

Returns: Differentiated multi-variate in direction Dir. A Bezier multi-variate.

Description: Given a Bezier multi-variate, computes its partial derivative multi-variate in direction Dir. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one. Then:

```
Q(i) = (k - 1) * (P(i+1) - P(i)), i = 0 to k-2.
```

This function computes the derivative of a rational function component- wise without taking into consideration the quotient rule.

See also: MvarMVDeriveBound, MvarMVDerive, MvarBspMVDerive, MvarBzrMVDeriveScalar,

8.2.108 MvarBzrMVDeriveAllBounds (mvar_der.c:518)

multi-variates

```
void MvarBzrMVDeriveAllBounds(const MvarMVStruct *MV, CagdMinMaxType *MinMax)
```

MV: Multi-Variate to differentiate.

MinMax: Bounds on the derivative values of MV in all directions.

Returns: void

Description: Given a scalar Bezier multi-variate, computes bounds to its partial derivative in all directions. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one. Then:

```
Q(i) = (k - 1) * (P(i+1) - P(i)), i = 0 to k-2.
```

See also: MvarMVDerive, MvarMVDeriveBound, MvarBspMVDeriveBound,

8.2.109 MvarBzrMVDeriveBound (mvar_der.c:351)

multi-variates

```
void MvarBzrMVDeriveBound(const MvarMVStruct *MV,
                          MvarMVDirType Dir,
                          CagdRType MinMax[2])
```

MV: Multi-Variate to differentiate.

Dir: Direction of differentiation.

MinMax: Bounds on the derivative values of MV in direction Dir.

Returns: void

Description: Given a scalar Bezier multi-variate, computes bounds to its partial derivative in direction Dir. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one. Then:

```
Q(i) = (k - 1) * (P(i+1) - P(i)), i = 0 to k-2.
```

See also: MvarMVDerive, MvarMVDeriveBound, MvarBspMVDeriveBound,

8.2.110 MvarBzrMVDeriveRational (mvbzrsym.c:525)

derivatives

```
MvarMVStruct *MvarBzrMVDeriveRational(const MvarMVStruct *MV,
                                      MvarMVDirType Dir)
```

MV: Rational Bezier multivariate to differentiate.

Dir: Direction of Differentiation.

Returns: Differentiated rational Bezier multivariate.

Description: Given a rational Bezier multivariates - computes its derivative surface in direction Dir, using the quotient rule for differentiation.
See also: MvarMVDerive, MvarBzrMVDerive, MvarBzrMVDerive, MvarBzrMVDeriveRational,

8.2.111 MvarBzrMVDeriveScalar (mvar_der.c:139)

derivatives

```
MvarMVStruct *MvarBzrMVDeriveScalar(const MvarMVStruct *MV, MvarMVDirType Dir)
```

MV: To differentiate.

Dir: Direction of differentiation. Either U or V.

Returns: Differentiated multi-variate.

Description: Given a Bezier multi-variate, computes its partial derivative multi-variate in direction Dir. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one. Then:

```
Q(i) = (k - 1) * (P(i+1) - P(i)), i = 0 to k-2.
```

For a Euclidean surface this is the same as MvarBzrMVDerive but for a rational multivar the returned multivar is not the vector field but simply the derivatives of all the multivar's coefficients, including the weights.
See also: MvarMVDeriveBound, MvarMVDerive, MvarBspMVDerive, MvarBzrMVDerive, , MvarBspMVDeriveScalar,

8.2.112 MvarBzrMVMult (mvbzrsym.c:309)

product

```
MvarMVStruct *MvarBzrMVMult(const MvarMVStruct *MV1, const MvarMVStruct *MV2)
```

MV1, MV2: The two multivariates to multiply.

Returns: The product MV1 * MV2 coordinate-wise.

Description: Given two Bezier multivariates - multiply them coordinate-wise. The two multivariates are promoted to same point type before multiplication can take place. See also BzrMultInterpFlag.

8.2.113 MvarBzrMVNew (mvar_gen.c:214)

multi-variates

allocation

```
MvarMVStruct *MvarBzrMVNew(int Dim, const int *Lengths, MvarPointType PType)
```

Dim: Number of dimensions of this multivariate.

Lengths: Of control mesh in each of the dimensions. Vector of size Dim.

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform multi-variate Bezier.

Description: Allocates the memory required for a new Bezier multi-variate.
See also: MvarMVFree, MvarMVNew, MvarBspMVNew, MvarPwrMVNew,

8.2.114 MvarBzrMVRegionFromMV (mvar_aux.c:747)

```
MvarMVStruct *MvarBzrMVRegionFromMV(const MvarMVStruct *MV,
                                    CagdRType t1,
                                    CagdRType t2,
                                    MvarMVDirType Dir)
```

MV: The Bezier multivariate, values of which are required over a domain other than [0,1].

t1: The required min' domain.

t2: The required max' domain.

Dir: The direction to extract the domain.

Returns: The new MV with the required new domain.

Description: Changing the domain of a multivariate, in direction Dir, such that: The values of the output multivariate over [0,1] in direction Dir, are those of the input multivariate over [NewMinDmn,NewMaxDmn]. If required, the user of this function should map back point x from [0,1] to [NewMinDmn,NewMaxDmn] by the affine domain change: y[dir] = (1 - x[dir]) * NewMinDmn + x[dir] * NewMaxDmn. NOTE: when the new domain is always known to be contained in the old domain, if possible use MvarMVRegionFromMV, which suits B-splines as well and supports the same operation. This function is made for cases where a new region is not contained in [0,1].

See also: MvarMVRegionFromMV, MvarMVExtension,

8.2.115 MvarBzrMVSubdivAtParam (mvar_sub.c:179)

multi-variates

```
MvarMVStruct *MvarBzrMVSubdivAtParam(const MvarMVStruct *MV,
                                     CagdRType t,
                                     MvarMVDirType Dir)
```

MV: Bezier Multi-Variate to subdivide.

t: Parameter to subdivide at.

Dir: Direction of subdivision.

Returns: A list of two multi-variates, result of the subdivision.

Description: Given a Bezier multi-variate, subdivides it at parameter value t in direction Dir.

See also: MvarBspMVSubdivAtParam, MvarMVSubdivAtParam,

8.2.116 MvarBzrMVSubdivAtParamOneSide (mvar_sub.c:574)

multi-variates

```
MvarMVStruct *MvarBzrMVSubdivAtParamOneSide(const MvarMVStruct *MV,
                                            CagdRType t,
                                            MvarMVDirType Dir,
                                            IrtBType LeftSide)
```

MV: Bezier Multi-Variate to subdivide.

t: Parameter to subdivide at.

Dir: Direction of subdivision.

LeftSide: TRUE to only fetch left half, FALSE to fetch right half.

Returns: A single multi-variate, result of the subdivision.

Description: Given a Bezier multi-variate, subdivides it at parameter value t in direction Dir.

See also: MvarBspMVSubdivAtParamOneSide, MvarMVSubdivAtParam,

8.2.117 MvarBzrMultBrnBasis (mvbzrsym.c:216)

product

```
MvarMVStruct *MvarBzrMultBrnBasis(const MvarMVStruct *MV,
                                  int *MulOrders,
                                  int *MulIndices)
```

MV: A Bezier multivariate.

MulOrders: The orders of the Bernstein basis function to multiply with.

MulIndices: MulIndices[i] is the exponent of the ith variable of the given Bernstein basis function.

Returns: The product of MV with the given Bernstein basis monomial, coordinate-wise.

Description: Given a Bezier multivariate and a Bernstein basis function - multiply them coordinate-wise.

8.2.118 MvarBzrSelfInter4VarDecomp (selfintr.c:1042)

```
void MvarBzrSelfInter4VarDecomp(const CagdSrfStruct *Srf,
                                MvarMVStruct **U1MinusU3Factor,
                                MvarMVStruct **U2MinusU4Factor)
```

Srf: The 2-variate to subtract against itself and decompose.

U1MinusU3Factor: The G function above.

U2MinusU4Factor: The H function above.

Returns: void

Description: Given a 2-variate Bezier function, Srf, that is to be subtracted from itself as,

```
F(u1, u2, u3, u4) = Srf(u1, u2) - Srf(u3, u4),
```

computes a decomposition for F as

```
F(u1, u2, u3, u4) = (u1 - u3) G(u1, u2, u3, u4) +
                    (u2 - u4) H(u1, u2, u3, u4),
```

that is known to always exist.

See also:

8.2.119 MvarBzrSrfSelfInterDiagFactor (selfintr.c:1267)

```
MvarPolylineStruct *MvarBzrSrfSelfInterDiagFactor(const CagdSrfStruct *Srf,
                                                  CagdRType SubdivTol,
                                                  CagdRType NumericTol)
```

Srf: The surface to derive its self intersections.

SubdivTol: Tolerance of the first zero set finding subdivision stage.

NumericTol: Tolerance of the second zero set finding numeric stage.

Returns: List of self intersection pllns, as (u, v) parameter pairs into the surfaces' domain. Points in R^4.

Description: Given a Bezier surface, S, compute its self intersections, if any, by computing a (u1 - u3) G(u1, u2, u3, u4) + (u2 - u4) H(u1, u2, u3, u4) decomposition for the 4-variate function of S(u1, u2)-S(u3, u4). The self intersection is then derived as the solution of:

```
Gx Hy - Gy Hx = 0,
Gx Hz - Gz Hx = 0,
(u1 - u3) Gx + (u2 - u4) Hx = 0.
```

The first two equations ensure the G an H are parallel while the last guarantee a zero point in X (which is therefore a zero point in Y an Z.) Due to singularities in these equations, all six equations of the form:

```
Gx Hy - Gy Hx = 0,
Gx Hz - Gz Hx = 0,
Gy Hz - Gz Hy = 0,
(u1 - u3) Gx + (u2 - u4) Hx = 0,
(u1 - u3) Gy + (u2 - u4) Hy = 0,
(u1 - u3) Gz + (u2 - u4) Hz = 0,
```

are employed during the subdivision stage. Finally a 7th equations of the form "|| S - S ||^2 = 0 is added for faster pruning of far regions.

See also: MvarBzrSelfInter4VarDecomp,

8.2.120 MvarCalculateExtremePoints (mvarjimp.c:298)

```
MvarPtStruct *MvarCalculateExtremePoints(const MvarMVStruct *MV)
```

MV: Input scalar function, represented as a multivariate, to compute parametric locations of extreme values.

Returns: List of extreme multivariate points candidates returns NULL in case of invalid input.

Description: Calculates list of extreme points of a given scalar function. Supports up to dimension 3 (i.e. trivariates). Note that the result, the returned list of parameteric locations, is a super set of the extreme values MV can assume as we decompose MV into low dimensional entities (i.e. boundary surfaces and curves) and an extrema in a lower diemsnional entities does not mean it is an extreme in MV.

8.2.121 MvarCalculateTVJacobian (mvarjimp.c:632)

```
MvarMVStruct *MvarCalculateTVJacobian(const TrivTVStruct *TV)
```

TV: The input trivariate.

Returns: The Jacobian in a multi variate representation.

Description: Calculates the Jacobian of a given trivariate.
See also: Symb2DSrfJacobian, MvarTrivJacobianImprove, MvarCalculateExtremePoints, , TrivComputeJacobian,

8.2.122 MvarCircAtDirMax (mvarpck3.c:312)

```
MvarPtStruct *MvarCircAtDirMax(const CagdCrvStruct *Bndry,
                               CagdPType XBnd,
                               CagdPType YBnd,
                               CagdRType Radius,
                               const CagdPType Dir,
                               CagdRType NumericTol,
                               CagdRType SubdivTol)
```

Bndry: The curve to find the circles that is tangent to.

XBnd: The range of x-coordinate of bounding box of Bndry.

YBnd: The range of y-coordinate of bounding box of Bndry.

Radius: The radius of the circle.

Dir: The direction to project the circle center on.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol < SubdivTol.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

Returns: List of (x,y) solution points.

Description: Computes all circles which are tangent to a given cirve and have a locally maximum projection on a given direction. The following set of multivariates is solved:

```
< Bndry(u) - (x,y), Bndry(u) - (x, y) > = R^2,
< Bndry(u) - (x,y), Bndry'(u) > = 0,
< Bndry'(u),  Dir > = 0
```

See also: MvarCircTanAtTwoPts,

8.2.123 MvarCircOnLineTangToBdry (mvarpck3.c:452)

```
MvarPtStruct *MvarCircOnLineTangToBdry(const CagdCrvStruct *Bndry,
                                       const CagdCrvStruct *InNrml,
                                       CagdRType Radius,
                                       const CagdPType Dir,
                                       const CagdPType Pt,
                                       CagdRType NumericTol,
                                       CagdRType SubdivTol)
```

Bndry: The curve to find the circles that is tangent to.

InNrml: Normal pointing in the interior of Bndry, or NULL.

Radius: The radius of the circle.

Dir: The direction of the line on which circle center must lie.

Pt: A point through which the line passes.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol < SubdivTol.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

Returns: List of (c,t) solution points.

Description: Computes all circles which are tangent to a given curve and whose centers lie on a given line. Following system of multivariates is solved:

```
< Pt + c.Dir - Bndry(t), Bndry'(t) > = 0,
< Pt + c.Dir - Bndry(t), Pt + c.Dir - Bndry(t) > - Radius^2= 0.
```

8.2.124 MvarCircTanAtTwoPts (mvarpck3.c:578)

```
MvarPtStruct *MvarCircTanAtTwoPts(const CagdCrvStruct *Bndry,
                                  CagdPType XBnd,
                                  CagdPType YBnd,
                                  CagdRType Radius,
                                  CagdRType NumericTol,
                                  CagdRType SubdivTol)
```

Bndry: The curve to find the circles that is tangent to.

XBnd: The range of x-coordinate of bounding box of Bndry.

YBnd: The range of y-coordinate of bounding box of Bndry.

Radius: The radius of the circle.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol < SubdivTol.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

Returns: List of (x,y) solution points.

Description: Computes all circles which are tangent to a given curve at two distinct points. The following set of multivariates is solved:

```
< Bndry(u) - (x,y), Bndry(u) - (x, y) > = R^2,
< Bndry(u) - (x,y), Bndry'(u) > = 0,
< Bndry(v) - (x,y), Bndry(v) - (x, y) > = R^2,
< Bndry(v) - (x,y), Bndry'(v) > = 0,
u - v > Eps
```

See also: MvarCircAtDirMax,

8.2.125 MvarCircTanTo2Crvs (mvtangnt.c:415)

bi-tangent

```
MvarPtStruct *MvarCircTanTo2Crvs(const CagdCrvStruct *Crv1,
                                 const CagdCrvStruct *Crv2,
                                 CagdRType Radius,
                                 CagdRType Tol)
```

Crv1, Crv2: The two curves to find the circles that is tangent to both.

Radius: Of all the circle(s) that is tangent to Crv1/2.

Tol: Tolerance of approximation. Subdiv Tol of MV Zeros.

Returns: List of the 4-tuples as (t, r, x, y).

Description: Computes all circles of prescribed radius that are tangent to given two XY planar curves. Solves for circles' centers P(x, y), using the following four equations in four unknowns (t, r, x, y), and R is the desired circle radius:

```
||C1(t) - P||^2 = R^2,
||C2(r) - P||^2 = R^2,
< C1(t) - P, C1'(t) > = 0,
< C2(t) - P, C2'(t) > = 0.
```

See also: SymbCircTanTo2Crvs, MvarCircTanTo3Crvs,

8.2.126 MvarCircTanTo3Crvs (mvtangnt.c:562)

tri-tangent

```
MvarPtStruct *MvarCircTanTo3Crvs(const CagdCrvStruct *Crv1,
                                 const CagdCrvStruct *Crv2,
                                 const CagdCrvStruct *Crv3,
                                 CagdRType SubdivTol,
                                 CagdRType NumericTol,
                                 CagdBType OneSideOrientation)
```

Crv1, Crv2, Crv3: The two curves to find the circles that is tangent to both.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol < SubdivTol.

OneSideOrientation: TRUE to compute tri-tangencies on one side of the curves only.

Returns: List of (u, v, w) solution points.

Description: Computes all circles that are tangent to given three curves. Solves for circles' centers P(x, y), using the following process: Solve, symbolically for P in the following 2x2 system by Cremmer rule:

```
||C1(u) - P||^2 = ||C2(v) - P||^2,
||C1(u) - P||^2 = ||C3(w) - P||^2,
```

and substitute P into the following 3 euqations and solve 3 equations in (u, v, w):

```
< C1(u) - P, C1'(u) > = 0,
< C2(v) - P, C2'(v) > = 0,
< C3(w) - P, C2'(w) > = 0.
```

See also: MvarCircTanTo2Crvs,

8.2.127 MvarCircTanToCircCrv3By3 (mvarpck3.c:760)

```
MvarPtStruct *MvarCircTanToCircCrv3By3(const CagdCrvStruct *Bndry,
                                       const CagdCrvStruct *InNrml,
                                       CagdPType XBnd,
                                       CagdPType YBnd,
                                       const CagdPType Center,
                                       CagdRType Radius,
                                       CagdBType BndBndryPar,
                                       CagdRType BndryPar,
                                       CagdRType NumericTol,
                                       CagdRType SubdivTol)
```

Bndry: The curve to find the circles that is tangent to.

InNrml: Normal pointing in the interior of Bndry, or NULL.

XBnd: The range of x-coordinate of bounding box of Bndry.

YBnd: The range of y-coordinate of bounding box of Bndry.

Center: The center of circle to find circle tangent to.

Radius: The radius of the circle.

BndBndryPar: If TRUE, find contact points on boundary curve greater than the specified value.

BndryPar: The boundary curve parameter to serve as lower bound.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol < SubdivTol.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

Returns: List of (x, y) solution points.

Description: Computes all circles which are tangent to a given circle and also to the another curve. The following system of multivariats is solved:

```
< Bndry(u) - (x,y), Bndry(u) - (x,y) > = R^2,
< Center - (x,y), Center - (x,y) > = 4 * R^2,
< Bndry(u) - (x,y), Bndry'(u) > = 0
```

8.2.128 MvarCircTanToCrvXCoord (mvarpck3.c:179)

```
MvarPtStruct *MvarCircTanToCrvXCoord(const CagdCrvStruct *Bndry,
                                     const CagdCrvStruct *InNrml,
                                     CagdPType YBnd,
                                     CagdRType Radius,
                                     CagdRType XCoord,
                                     CagdRType NumericTol,
                                     CagdRType SubdivTol)
```

Bndry: The curve to find the circles that is tangent to.

InNrml: Normal pointing in the interior of Bndry, or NULL.

YBnd: The range of y-coordinate of bounding box of Bndry.

Radius: The radius of the circle.

XCoord: The X-coordinate of the center of the circle.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol < SubdivTol.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

Returns: List of (y) solution points.

Description: Computes all circles which are tangent to a given curve and have a given x-coordinate. Following system of bivariates is solved, where the variables are u and y, while X is fixed:

```
< Bndry(u) - (X,y), Bndry(u) - (X, y) > = R^2,
< Bndry(u) - (X,y), Bndry'(u) > = 0
```

See also: MvarCircTanToCrvYCoord,

8.2.129 MvarCircTanToCrvYCoord (mvarpck3.c:47)

```
MvarPtStruct *MvarCircTanToCrvYCoord(const CagdCrvStruct *Bndry,
                                     const CagdCrvStruct *InNrml,
                                     CagdPType XBnd,
                                     CagdRType Radius,
                                     CagdRType YCoord,
                                     CagdRType NumericTol,
                                     CagdRType SubdivTol)
```

Bndry: The curve to find the circles that is tangent to.

InNrml: Normal pointing in the interior of Bndry, or NULL.

XBnd: The range of x-coordinate of bounding box of Bndry.

Radius: The radius of the circle.

YCoord: The Y-coordinate of the center of the circle.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol < SubdivTol.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

Returns: List of (x) solution points.

Description: Computes all circles which are tangent to a given curve and have a given y-coordinate. Following system of bivariates is solved, where the variables are u and x, while Y is fixed:

```
< Bndry(u) - (x,Y), Bndry(u) - (x, Y) > = R^2,
< Bndry(u) - (x,Y), Bndry'(u) > = 0
```

See also: MvarCircTanToCrvXCoord,

8.2.130 MvarCntctTangentialCrvCrvC1 (contacts.c:41)

```
MvarPtStruct *MvarCntctTangentialCrvCrvC1(const CagdCrvStruct *Crv1,
                                          const CagdCrvStruct *Crv2,
                                          CagdRType Epsilon)
```

Crv1, Crv2: The two curves to solve for their antipodal locations.

Epsilon: Tolerance of computation.

Returns: List of pairs of parameters in the r & t coefficients.

Description: Computes tangential contact points of the given two curves by solving for Ci(t) = (xi(t), yi(t)), i = 1,2:

```
x1(t) - x2(r) = 0,
y1(t) - y2(r) = 0,
< C1(t) - C2O(r), C1'(t) > = 0,
< C1(t) - C2O(r), C2'(r) > = 0,
```

where C2O is an offset curve of C2 by amount larger than subdivision tol.
See also: MvarCrvAntipodalPoints,

8.2.131 MvarCnvrtBsp2BzrMV (mvar_gen.c:1645)

conversion

multi-variate

```
MvarMVStruct *MvarCnvrtBsp2BzrMV(const MvarMVStruct *MV)
```

MV: A B-spline multi-variate to convert to Bezier MVs.

Returns: A list of Bezier multi-variate representing the same geometry as the given B-spline MV.

Description: Converts a Bspline multi-variate into a Bezier multi-variate by splitting at all interior knots.

8.2.132 MvarCnvrtBzr2BspMV (mvar_gen.c:1606)

conversion

multi-variate

```
MvarMVStruct *MvarCnvrtBzr2BspMV(const MvarMVStruct *MV)
```

MV: A Bezier multi-variate to convert to a B-spline MV.

Returns: A Bspline multi-variate representing the same geometry as the given Bezier MV.

Description: Converts a Bezier multi-variate into a B-spline multi-variate by adding two open end uniform knot vectors to it.

8.2.133 MvarCnvrtBzr2PwrMV (mvbzrpwr.c:63)

power basis

conversion

```
MvarMVStruct *MvarCnvrtBzr2PwrMV(const MvarMVStruct *MV)
```

MV: To convert into Power basis function representation.

Returns: Same geometry, but in the Power basis.

Description: Converts the given multivariate from Bezier basis functions to a Power basis functions. Using:

$$B_i^n(t) = \sum_{p=i}^{n} (-1)^{p-i} \binom{n}{p} \binom{p}{i} t^p$$

or

```
 n0     n1          nm
B (u0) B (v1) ... B (um) =
 i0     i1          im
```

```
  n0        nm
  --        --
  \         \      p0-i0  n0   p0          pm-im  nm   pm       p0       pm
  /    ...  /   (-1)      (  ) (  ) ... (-1)      (  ) (  )    u0   ... um
  --        --             p0   i0                 pm   im
 p0=i0    pm=im
```

This routine simply take the weight of each product of m Bezier basis functions B0(u0)... Bm(u0) and spread it into the different power basis u0^p0 ...um^pm functions scaled by:

```
    p0-i0 n0   p0         pm-im  nm   pm
(-1)    (  ) (  ) ... (-1)    (  ) (  )
          p0   i0               pm   im
```

See also: MvarCnvrtPwr2BzrMV, CagdCnvrtBzr2PwrSrf, CagdCnvrtPwr2BzrSrf,

8.2.134 MvarCnvrtCagdPtsToMVPts (mvar_pll.c:814)

```
MvarPtStruct *MvarCnvrtCagdPtsToMVPts(const CagdPtStruct *Pts)
```

Pts: Cagd points to convert to MV points.

Returns: MV points.

Description: Converts a list of Cagd points to a list of MV points.
See also: MvarCnvrtMVPolysToIritPolys, MvarCnvrtMVPolysToMVPts,

8.2.135 MvarCnvrtCrvToMV (mvareval.c:982)

multi-variates

```
MvarMVStruct *MvarCnvrtCrvToMV(const CagdCrvStruct *Crv)
```

Crv: Curve to convert into the multi-variate MV.

Returns: A multi variate function representation to Crv.

Description: Converts a curve into a multivariate function.

8.2.136 MvarCnvrtFloat2OpenMV (mvar_aux.c:1311)

conversion

```
MvarMVStruct *MvarCnvrtFloat2OpenMV(const MvarMVStruct *MV)
```

MV: B-spline multivariate to convert to open end conditions.

Returns: A Bspline multivariate with open end conditions, representing the same geometry as MV.

Description: Converts a float B-spline multivariate to a B-spline multivariate with open end conditions.
See also: MvarCnvrtPeriodic2FloatMV, CnvrtFloat2OpenMV,

8.2.137 MvarCnvrtIritLinCrvsToMVPolys (mvar_pll.c:1115)

```
MvarPolylineStruct *MvarCnvrtIritLinCrvsToMVPolys(const CagdCrvStruct *Crvs)
```

Crvs: List of linear curves to convert to mvar polylines.

Returns: A list of polylines.

Description: Converts a list of mvar polylines into a list of irit curves. Assumes crvs at most in E3.
See also: MvarCnvrtMVPtsToCtlPts, MvarCnvrtMVPolysToIritPolys2, , MvarCnvrtMVPolysToIritCrvs,

8.2.138 MvarCnvrtMVPolysToCtlPts (mvar_pll.c:876)

```
struct IPObjectStruct *MvarCnvrtMVPolysToCtlPts(const MvarPolylineStruct
                                                                    *MVPlls)
```

MVPlls: MV polylines to convert to MV points.

Returns: MV control points.

Description: Converts a list of MV polylines to a list of MV control points.
See also: MvarCnvrtMVPolysToIritPolys,

8.2.139 MvarCnvrtMVPolysToIritCrvs (mvar_pll.c:1058)

```
CagdCrvStruct *MvarCnvrtMVPolysToIritCrvs(const MvarPolylineStruct *MVPlls,
                                          int Order)
```

MVPlls: List of multivariate polylines to convert to Irit curves.

Order: Order of constructed curves, typically 2 (linear).

Returns: A list of curves.

Description: Converts a list of mvar polylines into a list of irit curves. Assumes all points of same dimensions.
See also: MvarCnvrtMVPtsToCtlPts, MvarCnvrtMVPolysToIritPolys2, , MvarCnvrtIritLinCrvsToMVPolys,

8.2.140 MvarCnvrtMVPolysToIritPolys (mvar_pll.c:912)

```
IPObjectStruct *MvarCnvrtMVPolysToIritPolys(const MvarPolylineStruct *MVPlls)
```

MVPlls: List of multivariate polylines to convert to Irit polylines.

Returns: A list object of polylines (as lists of ctlpts).

Description: Converts a list of mvar polylines into a list of irit polylines. Assumes all points of same dimensions.
See also: MvarCnvrtMVPtsToCtlPts, MvarCnvrtMVPtsToPolys2, MvarCnvrtMVPtsToPolys2, MvarCnvrtMVPolysToIritCrvs,

8.2.141 MvarCnvrtMVPolysToIritPolys2 (mvar_pll.c:974)

```
IPObjectStruct *MvarCnvrtMVPolysToIritPolys2(const MvarPolylineStruct *MVPlls,
                                             int IgnoreIndividualPts)
```

MVPlls: List of multivariate polylines to convert to Irit polylines.

IgnoreIndividualPts: True to handle only polylines, ignoring points.

Returns: A polylines object, or if have individual points a list of two objects, as (PllnObjs, PntObjs).

Description: Converts a list of mvar polylines into a list of irit polylines. Assumes all points of same dimensions.
See also: MvarCnvrtMVPtsToCtlPts, MvarCnvrtMVPtsToPolys2, MvarCnvrtMVPtsToPolys, MvarCnvrtMVPolysToIritCrvs,

8.2.142 MvarCnvrtMVPolysToMVPts (mvar_pll.c:845)

```
MvarPtStruct *MvarCnvrtMVPolysToMVPts(const MvarPolylineStruct *MVPlls)
```

MVPlls: MV polylines to convert to MV points.

Returns: MV points.

Description: Converts a list of MV polylines to a list of MV points.
See also: MvarCnvrtMVPolysToIritPolys, MvarCnvrtCagdPtsToMVPts,

8.2.143 MvarCnvrtMVPtsToCagdPts (mvar_pll.c:502)

```
CagdPtStruct *MvarCnvrtMVPtsToCagdPts(const MvarPtStruct *MVPts)
```

MVPts: List of multivariate points to convert to list of =points.

Returns: A list of points.

Description: Converts a list of multivariate points into a list of cagd E3 points. Assumes all input points are E1, E2, or E3.
See also: MvarCnvrtMVPtsToCtlPts, MvarCnvrtMVPtsToPolys,

8.2.144 MvarCnvrtMVPtsToCtlPts (mvar_pll.c:537)

MvarCnvrtMVPtsToCagdPts

```
IPObjectStruct *MvarCnvrtMVPtsToCtlPts(const MvarPtStruct *MVPts,
                                       IrtRType MergeTol)
```

MVPts: List of multivariate points to convert to list of ctlpts.

MergeTol: If non negative, attempt to merge the data into polylines.

Returns: A list object of ctlpts.

Description: Converts a list of multivariate points into a list of control points. Assumes all points of same dimensions.
See also: MvarCnvrtMVPtsToPolys, MvarCnvrtMVPtsToCagdPts,

8.2.145 MvarCnvrtMVPtsToPolys (mvar_pll.c:605)

```
IPObjectStruct *MvarCnvrtMVPtsToPolys(const MvarPtStruct *MVPts,
                                      const MvarMVStruct *MV,
                                      IrtRType MergeTol)
```

MVPts: List of multivariate points to convert to polylines.

MV: A multivariate to evaluate through, if not NULL.

MergeTol: Tolerance to merge points into polylines.

Returns: A list object of polylines.

Description: Converts a list of multivariate points into a list of polylines. Assumes all points of same dimensions.
See also: MvarCnvrtMVPtsToCtlPts, MvarCnvrtMVPtsToPolys2,

8.2.146 MvarCnvrtMVPtsToPolys2 (mvar_pll.c:678)

```
IPPolygonStruct *MvarCnvrtMVPtsToPolys2(const MvarPtStruct *InPts,
                                        CagdRType FineNess,
                                        int Dim,
                                        IrtRType *ParamDomain)
```

InPts: A list of discrete points.

FineNess: Tolerance.

Dim: The dimension of discrete points, 1 to 3.

ParamDomain: The domain of the mvar points.

Returns: Connected list of polylines.

Description: Connect a list of discrete points into a polylines.
See also: MvarCnvrtMVPtsToCtlPts, MvarCnvrtMVPtsToPolys,

8.2.147 MvarCnvrtMVToCrv (mvareval.c:1038)

multi-variates

```
CagdCrvStruct *MvarCnvrtMVToCrv(const MvarMVStruct *MV)
```

MV: A multivariate of at least dimension one to convert to a curve.

Returns: A curve representation the given multivariate (or its lowest dimension if higher dim.).

Description: Converts a multivariate function into a curve. If the multivariate is of dimension higher than one, the lowest dimension is employed in the conversion.

8.2.148 MvarCnvrtMVToSrf (mvareval.c:1171)

multi-variates

```
CagdSrfStruct *MvarCnvrtMVToSrf(const MvarMVStruct *MV)
```

MV: A multivariate of dimension two or more to convert to a surface.

Returns: A surface representation the given multivariate (or its lowest two dimensions if higher dim.).

Description: Converts a multivariate function into a surface. If the multivariate is of dimension higher than two, the lowest two dimensions are employed in the conversion.

8.2.149 MvarCnvrtMVToTV (mvareval.c:1329)

multi-variates

```
TrivTVStruct *MvarCnvrtMVToTV(const MvarMVStruct *MV)
```

MV: A multivariate of dimension three or more to convert to a trivar.

Returns: A trivar representation the given multivariate (or its lowest three dimensions if higher dim.).

Description: Converts a multivariate function into a trivar. If the multivariate is of dimension higher than three, the lowest three dimensions are employed in the conversion.

8.2.150 MvarCnvrtMVTrsToIritPolygons (mvar_pll.c:1197)

```
IPObjectStruct *MvarCnvrtMVTrsToIritPolygons(const MvarTriangleStruct *MVTrs,
                                             int *Coords)
```

MVTrs: List of multivariate triangles to convert to Irit polygons.

Coords: The required coordinates, or NULL if the dimension is three.

Returns: A list object of polygons.

Description: Converts a list of mvar triangles into a list of irit polygons. If the list of triangles consists of points of dimension higher than three, the polygons list is created as the projection on R^3, using the coordinates specified in Coords. Assumes all points of same dimensions.
See also: MvarIrit2DTrTo2DMVTrs, MvarCnvrtMVPolysToIritPolys,

8.2.151 MvarCnvrtPeriodic2FloatMV (mvar_aux.c:1239)

conversion

```
MvarMVStruct *MvarCnvrtPeriodic2FloatMV(const MvarMVStruct *MV)
```

MV: B-spline multivariate to convert to floating end conditions. Assume MV is either periodic or has floating end condition.

Returns: A Bspline multivariate with floating end conditions, representing the same geometry as MV.

Description: Converts a B-spline multivariate into a B-spline multivariate with floating end conditions.
See also: CnvrtPeriodic2FloatMV, MvarCnvrtFloat2OpenMV,

8.2.152 MvarCnvrtPwr2BzrMV (mvbzrpwr.c:161)

power basis

conversion

```
MvarMVStruct *MvarCnvrtPwr2BzrMV(const MvarMVStruct *MV)
```

MV: To convert into Bezier basis function representation.

Returns: Same geometry, but in the Bezier basis.

Description: Converts the given multivariate from Power basis functions to Bezier basis functions. Using:

```
              n0       nm    i0       im
              --       --   (  ) ... (  )
  p0     pm   \        \     p0       pm    n0          nm
u0 ... um  =  /    ... /    -------------- B (u0) ... B (um)
              --       --    n0       nm    i0          im
             i0=p0    im=pm (  ) ... (  )
                            p0       pm
```

This routine simply take the weight of each product of m power basis functions u0^p0 ...um^pm and spread it into the different Bezier basis B0(u0)... Bm(u0) functions scaled by:

```
 i0       im
(  ) ... (  )
 p0       pm
--------------
 n0       nm
(  ) ... (  )
 p0       pm
```

See also: MvarCnvrtBzr2PwrMV, CagdCnvrtBzr2PwrMV, MvarCnvrtPwr2BzrMV,

8.2.153 MvarCnvrtSrfToMV (mvareval.c:1097)

multi-variates

```
MvarMVStruct *MvarCnvrtSrfToMV(const CagdSrfStruct *Srf)
```

Srf: Surface to convert into the multi-variate MV.

Returns: A multi variate function representation to Srf.

Description: Converts a surface into a multivariate function.

8.2.154 MvarCnvrtTVToMV (mvareval.c:1244)

multi-variates

```
MvarMVStruct *MvarCnvrtTVToMV(const TrivTVStruct *TV)
```

TV: Trivar to convert into the multi-variate MV.

Returns: A multi variate function representation to TV.

Description: Converts a trivar into a multivariate function.

8.2.155 MvarCoerceMVTo (mvarcoer.c:52)

coercion

```
MvarMVStruct *MvarCoerceMVTo(const MvarMVStruct *MV, MvarPointType PType)
```

MV: To coerce to a new point type PType.

PType: New point type for MV.

Returns: A new multi-variate with PType as its point type.

Description: Coerces a multi-variate to point type PType.

8.2.156 MvarCoerceMVsTo (mvarcoer.c:25)

coercion

```
MvarMVStruct *MvarCoerceMVsTo(const MvarMVStruct *MV, MvarPointType PType)
```

MV: To coerce to a new point type PType.

PType: New point type for MV.

Returns: New multivariates with PType as their point type.

Description: Coerces a list of multivariates to point type PType.

8.2.157 MvarComposeMVMdl (mvcomps2.c:736)

composition
trimming
multivariates

```
MdlModelStruct *MvarComposeMVMdl(const MvarMVStruct *MV,
                                 const MdlModelStruct *Models)
```

MV: The mapping multivariate.

Models: The models to compose into MV.

Returns: The composed models.

Description: Compose a list of B-rep models (Models) into a multivariate (MV), allowing the surfaces of the models to cross knot values of the mapping multivariate.
See also: MvarTrimComposeMVSrf, MvarUnTrimComposeMVSrf, MvarComposeMVVModel,

8.2.158 MvarComposeMVVModel (mvcomps2.c:813)

composition
trimming
multivariates
V-model

```
VMdlVModelStruct *MvarComposeMVVModel(const MvarMVStruct *MV,
                                      const VMdlVModelStruct *Vmdl)
```

MV: The mapping multivariate.

Vmdl: The V-models to compose into MV.

Returns: The composed V-models.

Description: Compose a list of V-Models (Vmdl) into a multivariate (MV), allowing the V-models to cross knot values of the mapping multivariate.
See also: MvarTrimComposeMVSrf, MvarUnTrimComposeMVSrf, MvarComposeMVMdl,

8.2.159 MvarComposedSrfAssumeSrf (mvcomps2.c:1051)

composition
trimming
multivariates

```
MvarComposedSrfStruct *MvarComposedSrfAssumeSrf(CagdSrfStruct *Srf)
```

Srf: The surface on which to base the MvarComposedSrfStruct.

Returns: A MvarComposedSrfStruct with the given surface.

Description: Creates a MvarComposedSrfStruct from a given surface. The surface is NOT copied.
See also: MvarTrimComposeMVSrf, MvarComposedSrfAssumeTSrf,

8.2.160 MvarComposedSrfAssumeTSrf (mvcomps2.c:1082)

composition
trimming
multivariates

```
MvarComposedSrfStruct *MvarComposedSrfAssumeTSrf(TrimSrfStruct *TSrf)
```

TSrf: The trimmed surface on which to base the MvarComposedSrfStruct.

Returns: A MvarComposedSrfStruct with the given trimmed surface.

Description: Creates a MvarComposedSrfStruct from a given trimmed surface. The trimmed surface is NOT copied.
See also: MvarTrimComposeMVSrf, MvarComposedSrfAssumeSrf,

8.2.161 MvarComposedSrfCopy (mvcomps2.c:1112)

composition
trimming
multivariates

```
MvarComposedSrfStruct *MvarComposedSrfCopy(const MvarComposedSrfStruct
                                                                *CompSrf)
```

CompSrf: The MvarComposedSrfStruct to copy.

Returns: A copy of the given MvarComposedSrfStruct.

Description: Copies a single MvarComposedSrfStruct.
See also: MvarTrimComposeMVSrf, MvarComposedSrfCopyList,

8.2.162 MvarComposedSrfCopyList (mvcomps2.c:1144)

composition
trimming
multivariates

```
MvarComposedSrfStruct *MvarComposedSrfCopyList(const MvarComposedSrfStruct
                                                               *CompSrfs)
```

CompSrfs: The list of MvarComposedSrfStruct to copy.

Returns: A copy of the given list of MvarComposedSrfStruct.

Description: Copies a list of MvarComposedSrfStruct.
See also: MvarTrimComposeMVSrf, MvarComposedSrfCopy,

8.2.163 MvarComposedTrivAssumeTV (mvcomps2.c:1237)

composition
V-model
multivariates

```
MvarComposedTrivStruct *MvarComposedTrivAssumeTV(TrivTVStruct *TV)
```

TV: The trivariate on which to base the MvarComposedTrivStruct.

Returns: A MvarComposedTrivStruct with the given trivariate.

Description: Creates a MvarComposedTrivStruct from a given trivariate. The trivariate is NOT copied.
See also: MvarTrimComposeMVTV, MvarComposedTrivAssumeVMdl,

8.2.164 MvarComposedTrivAssumeVMdl (mvcomps2.c:1268)

composition
V-model
multivariates

```
MvarComposedTrivStruct *MvarComposedTrivAssumeVMdl(VMdlVModelStruct *VMdl)
```

VMdl: The V-model on which to base the MvarComposedTrivStruct.

Returns: A MvarComposedTrivStruct with the given V-model.

Description: Creates a MvarComposedTrivStruct from a given V-model. The V-model is NOT copied.
See also: MvarTrimComposeMVTV, MvarComposedTrivAssumeTV,

8.2.165 MvarComposedTrivCopy (mvcomps2.c:1357)

composition
trimming
multivariates

```
MvarComposedTrivStruct *MvarComposedTrivCopy(const MvarComposedTrivStruct *CTV)
```

CTV: The MvarComposedTrivStruct to copy.

Returns: A copy of the given MvarComposedTrivStruct.

Description: Copies a single MvarComposedTrivStruct.
See also: MvarTrimComposeMVTV, MvarComposedTrivCopyList,

8.2.166 MvarComposedTrivCopyList (mvcomps2.c:1389)

composition
trimming
multivariates

```
MvarComposedTrivStruct *MvarComposedTrivCopyList(
                                        const MvarComposedTrivStruct *CTVs)
```

CTVs: The list of MvarComposedTrivStruct to copy.

Returns: A copy of the given list of MvarComposedTrivStruct.

Description: Copies a list of MvarComposedTrivStruct.
See also: MvarTrimComposeMVTV, MvarComposedTrivCopy,

8.2.167 MvarComputeInterMidPoint (mvtrmbis.c:363)

```
CagdRType *MvarComputeInterMidPoint(CagdCrvStruct *Crv1,
                                    CagdRType t1,
                                    CagdCrvStruct *Crv2,
                                    CagdRType t2,
                                    CagdRType *Inter)
```

Crv1: First curve of the matching mid point.

t1: Parameter value of first curve's mid point.

Crv2: Second curve of the matching mid point.

t2: Parameter value of second curve's mid point.

Inter: Point of intersection.

Returns: Point of intersection.

Description: Computes the intersection point of the normals of the given two points on the given two curves. Taken from crvskel.c - need to modify to avoid artifacts in near parallel normals.

8.2.168 MvarComputeMVPowers (mvcompos.c:1406)

```
MvarMVStruct **MvarComputeMVPowers(const MvarMVStruct *CMV, int Order)
```

CMV: Multivariate to compute factors for.

Order: Order is n + 1.

Returns: A vector of all possible factors for i equal to 0 to n, allocated dynamically.

Description: Computes the factors of the Bernstein polynomials where MV is a scalar-range multivariate, for i from 0 to n (degree):

```
 n                  n                     n-i                i
B (MV(t0,...tm)) = ( ) (1 - MV(t0,...tm))     (MV(t0,...tm))
 i                  i
```

The multivariate MV(t0,...tm) is a scalar, possibly a rational multivariate. If rational, the returned vector, index Order will contain w MV(t0,...tm)^n. See: "Freeform surface analysis using a hybrid of symbolic and numeric computation" by Gershon Elber, PhD thesis, University of Utah, 1992.
See also: MvarGenerateBspBasisMVs, SymbComputeCurvePowers, , SymbComputeSurfacePowers, MvarMVCompose,

8.2.169 MvarComputeRayTraps (ray-trap.c:70)

```
MvarPtStruct *MvarComputeRayTraps(const CagdCrvStruct *Crvs,
                                  int Orient,
                                  CagdRType SubdivTol,
                                  CagdRType NumerTol,
                                  CagdBType UseExprTree)
```

Crvs: List of curves to handle in order (cyclically).

Orient: Pick the proper orientation with respect to the normal if TRUE. May be faster at times.

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the curves.

NumerTol: Numeric tolerance of a possible numeric improvment stage. The numeric stage is employed if NumericTol < SubdivTol.

UseExprTree: TRUE to use expression trees in the computation, FALSE to use regular multivariate expressions.

Returns: Linked list of solutions, each holding the parameter values of the different Crvs.

Description: Computes solutions to locations on the given curves that would bounce rays from one curve to the next in a closed loop.

8.2.170 MvarComputeRayTraps3D (raytrp3d.c:74)

```
MvarPtStruct *MvarComputeRayTraps3D(const CagdSrfStruct *Srfs,
                                    int Orient,
                                    CagdRType SubdivTol,
                                    CagdRType NumerTol,
                                    CagdBType UseExprTree)
```

Srfs: List of surfaces to handle in order (cyclically).

Orient: Pick the proper orientation with respect to the normal if TRUE. May be faster at times.

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the surfaces.

NumerTol: Numeric tolerance of a possible numeric improvment stage. The numeric stage is employed if NumericTol < SubdivTol.

UseExprTree: TRUE to use expression trees in the computation, FALSE to use regular multivariate expressions.

Returns: Linked list of solutions, each holding the parameter values of the different Srfs.

Description: Computes solutions to locations on the given surfaces that would bounce rays from one surface to the next in a closed loop.

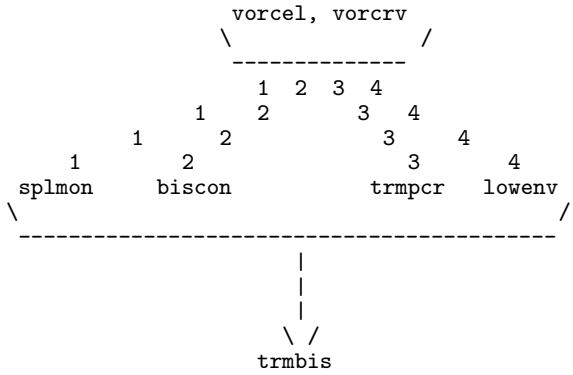
8.2.171 MvarComputeVoronoiCell (mvvorcel.c:143)

```
IPObjectStruct *MvarComputeVoronoiCell(CagdCrvStruct *Crv)
```

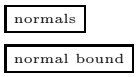
Crv: List of curves to compute the Voronoi cell.

Returns: Returns the Voronoi cell.

Description: Computes the Voronoi cell of the first curve in the given list of curves. For the details of the algorithm, see the following paper.
Precise Voronoi Cell Extraction of Free-form Rational Planar Closed Curves. Iddo Hanniel, Ramanathan Muthuganapathy, Gershon Elber, Myung-Soo Kim ACM Symposium on Solid and Physical Modeling, 2005. The following are the files used for computing the Voronoi cell mvvorcel.c - Main function that call other function. It takes the input curves, process them for discontinuities and calls other functions. Displaying the output is also done in this function. mvsplmon.c - The function that creates monotone segments. mvbiscon.c - The monotone segments obtained from mvsplmon.c are then subjected to orientation and curvature constraint functions written in this file. mvtrmbis.c - Auxillary functions required for trimming are written here. mvtrmpcr.c - This file does the trimming of point/crv bisector. mvlowenv.c - Functions in this file compute the lower envelope. mvvorcrv.c - Operations on the MvarVoronoiCrvStruct are available in this file. Dependency of each of the above file is depicted in the following diagram:



8.2.172 MvarConesOverlapAux (mvcones.c:1579)




```
CagdBType MvarConesOverlapAux(const MvarNormalConeStruct *ConesList)
```

ConesList: Cones in a list.

Returns: TRUE if overlap, FALSE if not.

Description: Computes the tangency anti-cones of the set of normal cones, and returns whether they overlap or not.

8.2.173 MvarCreateKrnlTV (krnl_based_cnstrct.c:2897)

```
TrivTVStruct *MvarCreateKrnlTV(MvarKrnlTVInptStruct *KrnlInpt,
                               int Op,
                               int AllowInvldTV)
```

KrnlInpt: The input surfaces and parameters.

Op: 1 - For Boolean sum operator. 2 - For one sided Boolean sum operator using two adjacent surfaces. 3 - For one sided Boolean sum operator using three adjacent surfaces. 4 - For ruling operator.

AllowInvldTV: Allow return of invalid trivariates, for debugging purposes.

Returns: The constructed kernel based trivariate.

Description: Constructs a kernel based trivariate.
See also:

8.2.174 MvarCreatePlnrKrnlSrf (krnl_based_cnstrct.c:1304)

```
CagdSrfStruct *MvarCreatePlnrKrnlSrf(MvarPlnrKrnlSrfInptStruct *KrnlInpt,
                                     int Op,
                                     int AllowInvldSrf)
```

KrnlInpt: The input curves and parameters.

Op: 1 - For Boolean sum operator. 2 - For one sided Boolean sum operator. 3 - For ruling operator.

AllowInvldSrf: Allow return of invalid surfaces, for debugging purposes.

Returns: The constructed kernel based surface.

Description: Constructs a planar kernel based surface.
See also: MvarPlnrKrnlBooleanSumSrf, MvarPlnrKrnlOneSidedBSumSrf, , MvarPlnrKrnlRuledSrf,

8.2.175 MvarCrv2DMAT (mv_mat2d.c:72)

```
CagdCrvStruct *MvarCrv2DMAT(const CagdCrvStruct *OCrv,
                            CagdRType SubdivTol,
                            CagdRType NumericTol,
                            CagdBType InvertOrientation)
```

OCrv: B-spline curve to compute its 2D planar MAT.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol < SubdivTol.

InvertOrientation: Flip what is considered inside and outside.

Returns: The MAT as a list of curves.

Description: Computes the (inside) 2D MAT (medial axis transform) of a given closed planar self-intersection-free oriented B-spline curve.

8.2.176 MvarCrvAntipodalPoints (selfintr.c:146)

```
MvarPtStruct *MvarCrvAntipodalPoints(const CagdCrvStruct *CCrv,
                                     CagdRType SubdivTol,
                                     CagdRType NumericTol)
```

CCrv: To detect its antipodal points.

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the surfaces.

NumericTol: Numeric tolerance of a possible numeric improvment stage. The numeric stage is employed if NumericTol < SubdivTol.

Returns: Antipodal points, as points in E2 (r, t).

Description: Computes antipodal points in the given curve - pairs of points C(t) and C(r) such that (t and r are two independent parameters of same curve):

```
< C(t) - C(r), dC(t)/dt > = 0,
< C(t) - C(r), dC(r)/dr > = 0.
```

Direct attempt to solve this set of constraints is bound to be slow as all points in Crv satisfy these equations when t == r. The key is in adding a third inequality constaint of the form

```
< C'(t), C'(r) > < 0.
```

Antipodal points must exists if Crv self intersect in a closed loop and hence can help in detecting self-intersections. Further, the diameter of Crv could be easily deduced from the antipodal points. Note this function also captures the self-intersection locations C(t) = C(r), for which the dot product of the tangents is negative.

See also: MvarCrvDiameter, SymbCrvDiameter, MvarSrfAntipodalPoints, , MvarHFDistAntipodalCrvCrvC1,

8.2.177 MvarCrvArtGalleryPoint (crv_krnl.c:1222)

```
IPObjectStruct *MvarCrvArtGalleryPoint(const CagdCrvStruct *Crv,
                                       CagdRType SubEps,
                                       int NumGuards,
                                       int BBoxSubdivDepth)
```

Crv: Simple closed curve to estimates art gallery solution for.

SubEps: Subdivision epsilon.

NumGuards: Number of guards to try and place in this art gallery corevage. NumGuards > 1.

BBoxSubdivDepth: Zero to use regular bounding boxes on constraints. Positive integer >= 2 for recursive subdivision of this depth for a more precise BBOx bounds.

Returns: XY locations of all Guards if found, as points in r^k, k = NumGuards * 2. NULL if none found.

Description: Estimates art gallery solution for a given freeform planar closed curve. The presented algebraic approach is NOT solving the art gallery (due to constraints being local and visibility is a global problem) but it shows a possible use of the partial positivity in constraints in the MV solver.
See also: MvarCrvKernel, MvarCrvAntipodalPoints, MvarCrvKernelPoint,

8.2.178 MvarCrvCrvBisector2D (mvbisect.c:1287)

```
CagdCrvStruct *MvarCrvCrvBisector2D(const CagdCrvStruct *Crv1,
                                    const CagdCrvStruct *Crv2,
                                    CagdRType Step,
                                    CagdRType SubdivTol,
                                    CagdRType NumericTol,
                                    CagdRType *BBoxMin,
                                    CagdRType *BBoxMax,
                                    CagdBType SupportPrms)
```

Crv1, Crv2: Planar curves to compute their bisector.

Step: Stepsize for curve tracing.

SubdivTol: The subdivision tolerance to use.

NumericTol: The numerical tolerance to use.

BBoxMin, BBoxMax: The bounding box, where the bisector is computed.

SupportPrms: TRUE to return a curve in E4 as (X, Y, y1, t2), FALSE to return a curve in E2 as (X, Y).

Returns: A (list of) piecewise linear curves that approximates the bisector curve, or NULL if none.

Description: Computes a bisector curve of two planar curves. Curves are assumed to to lie in z = 0 plane. The bisector curve is computed inside BBox.
See also: MvarCrvDtctSelfInterLocations,

8.2.179 MvarCrvCrvContact (mvarintr.c:594)

```
MvarPtStruct *MvarCrvCrvContact(const CagdCrvStruct *Crv1,
                                const CagdCrvStruct *Crv2,
                                const CagdCrvStruct *MotionCrv1,
                                const CagdCrvStruct *ScaleCrv1,
                                CagdRType SubdivTol,
                                CagdRType NumericTol,
                                CagdBType UseExprTree)
```

Crv1, Crv2: Two curves to compute contacts over time in R^3.

MotionCrv1: The motion over time Crv1 undergoes. Can be NULL.

ScaleCrv1: The scale over time Crv1 undergoes. Can either be a scalar function, or vector function in R^3. Can be NULL. If both MotionCrv1 and Crv1Scale are defined, they better share their domains.

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the curves.

NumericTol: Numeric tolerance of a possible numeric improvement stage. The numeric stage is employed if NumericTol < SubdivTol.

UseExprTree: TRUE to use expression trees in the computation, FALSE to use regular multivariate expressions.

Returns: List of intersection points, as parameter pairs into the two curves domains.

Description: Computes the contact locations of two C^1 curves in R^3, possibly with multivariate expression trees. Expression trees could be beneficial computationally when the geometry is complex (i.e. dozens of control points or more, in each directions).
See also: MvarCrvCrvInter, MvarSrfSrfSrfInter, MvarSrfSrfContact,

8.2.180 MvarCrvCrvInter (mvarintr.c:80)

```
MvarPtStruct *MvarCrvCrvInter(const CagdCrvStruct *Crv1,
                              const CagdCrvStruct *Crv2,
                              CagdRType SubdivTol,
                              CagdRType NumericTol,
                              CagdBType UseExprTree)
```

Crv1, Crv2: Two curves to intersect.

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the surfaces.

NumericTol: Numeric tolerance of a possible numeric improvement stage. The numeric stage is employed if NumericTol < SubdivTol.

UseExprTree: TRUE to use expression trees in the computation, FALSE to use regular multivariate expressions.

Returns: List of intersection points, as parameter pairs into the two curves domains.

Description: Computes the intersection locations of two planar curves, possibly with multivariate expression trees. Expression trees could be beneficial computationally when the geometry is complex (i.e. dozens of control points or more).

See also: CagdCrvCrvInter, SymbCrvCrvInter, SymbSrfSrfSrfInter, MvarSrfSrfContact,

8.2.181 MvarCrvCrvMinimalDist (hasdrf2d.c:1108)

```
MvarPtStruct *MvarCrvCrvMinimalDist(const CagdCrvStruct *Crv1,
                                    const CagdCrvStruct *Crv2,
                                    CagdRType *MinDist,
                                    CagdBType ComputeAntipodals,
                                    CagdRType Eps)
```

Crv1: To detect its minimal distance to Crv2.

Crv2: To detect its minimal distance to Crv1.

MinDist: Upon return, is set to the minimal distance detected.

ComputeAntipodals: TRUE to compute antipodal locations as well in the search for minimal distance. If FALSE, samples are made along both curves and closest locations to other curve are computed.

Eps: Numeric tolerance of the computation.

Returns: Pairs of parameters at the minimal distance, allocated dynamically.

Description: Computes the minimal distance between two given planar C1 curves. This minimal distance can occur:

1. At intersection locations, if any.
2. At end points vs. end points.
3. At the end points vs interior locations.
4. At antipodal interior locations.

See also: MvarCrvCrvAntipodalPoints, SymbDistCrvPoint, CagdCrvCrvInter, SymbSrfDistCrvCrv,

8.2.182 MvarCrvCrvtrByOneCtlPt (mv_crvtr.c:224)

curvature

```
MvarMVStruct *MvarCrvCrvtrByOneCtlPt(const CagdCrvStruct *Crv,
                                     int CtlPtIdx,
                                     CagdRType Min,
                                     CagdRType Max)
```

Crv: To compute its curvature behaviour (convex vs. concave) as a function of the curve parameter and the Euclidean coordiate of the CtlPtIdx's control point.

CtlPtIdx: Index of control point to make a parameter for the curvature.

Min, Max: Domain each coordinate of CtlPtIdx point should vary.

Returns: The computed curvature field of Crv.

Description: Given a parametric curve, Crv, and a control point index CtlPtIdx, compute the curvature sign field of the curve as function of the Euclidean locations of control point index CtlPtIdx. Returned is a multivariate of dimension "1 + Dim(Crv)", where Dim(Crv) is the dimension of the curve (E2, E3, etc.).

See also: SymbCrv2DCurvatureSign, MvarCrvMakeCtlPtParam, UserCrvCrvtrByOneCtlPt,

8.2.183 MvarCrvDiameter (crv_krnl.c:481)

```
IPObjectStruct *MvarCrvDiameter(const CagdCrvStruct *Crv,
                                CagdRType SubEps,
                                CagdRType NumEps)
```

Crv: Simple closed curve to compute the diameter for.

SubEps: Subdivision epsilon.

NumEps: Numeric marching tolerance.

Returns: List of pair of parameter values between which the local diameter could be found.

Description: Computes the minimal and maximal diameter of the given curve. Let the input curve be C(t) and let f(t,r) = < C(t)-C(r), C(t)-C(r) >. Then, df/dt = df/dr = 0 find all the finite set of line segments that connects two points on the curve orthogonally to the curve. The min/max diameter is part of this set.
See also: MvarCrvKernel, MvarCrvAntipodalPoints,

8.2.184 MvarCrvDtctSelfInterLocations (offset2.c:1124)

```
MvarPtStruct *MvarCrvDtctSelfInterLocations(const CagdCrvStruct *Crv,
                                            CagdRType OffsetDist,
                                            CagdRType SubdivTol,
                                            CagdRType NumericTol)
```

Crv: Curve to detects its self intersection locations, if any.

OffsetDist: The desired offset distance.

SubdivTol, NumericTol: Tolerances of MV solver.

Returns: Locations of self intersection in R^4 as (u, v, x, y), if any. Can be NULL if none.

Description: Compute the locations where an offset by distance OffsetDist of Crv will yield self intersections.
See also: MvarCrvCrvBisector2D, MvarCrvTrimSelfInterLocations, , SymbCrvTrimGlblOffsetSelfInter, SymbSrfTrimGlblOffsetSelfInter, MvarCrvTrimGlblOffsetSelfInter, MvarCrvTrimGlblOffsetSelfInter2,

8.2.185 MvarCrvGammaKernel (crv_krnl.c:157)

```
MvarMVStruct *MvarCrvGammaKernel(const CagdCrvStruct *Crv, CagdRType Gamma)
```

Crv: Simple closed curve to compute its gamma-kernel.

Gamma: Angular deviation of the gamma-kernel, in degrees.

Returns: The trivariate function F(u, v, t) whose zero set, projected on the XY plane, is the domain that is not in the gamma-kernel.

Description: Computes the gamma-kernel of the given curve, or the points in the plane that have a line of sight (rotated gamma degrees) with all the points of the curve. The curve is assumed to be closed and simple. Let the input curve be C(t) and let S(u, v) = (u, v), the XY plane. Further let D'(t) = Rot(Gamma)[C'(t)]. Then, let F(u, v, t) = (C(t) - S(u, v)) x D'(t) (only the Z component of crossprod).
The zero set of F projected over the XY plane defines all the domain that is NOT in the gamma-kernel of C(t).
See also: MvarCrvKernelSilhouette, MvarCrvKernel, MvarCrvGammaKernelSrf,

8.2.186 MvarCrvGammaKernelSrf (crv_krnl.c:271)

```
MvarMVStruct *MvarCrvGammaKernelSrf(const CagdCrvStruct *Crv,
                                    CagdRType ExtentScale,
                                    CagdRType GammaMax)
```

Crv: Simple curve to compute its gamma-kernel surface.

ExtentScale: To scale the constructed surface as function of Gamma.

GammaMax: Max gamma deviation of the gamma-kernel, in degrees, for this curve. If negative, does so to opposite direction.

Returns: The surface S(u, v) or trivariate function F(u, v, t) representing the gamma surface as a function of gamma.

Description: Constructs a gamma-kernel surfaces the given curve and as a function of gamma. Let the input curve be C(t). If C(t) is a linear curve then we compute the surface: Let P = (Px, Py) be the initial point and (Dx, Dy) the slope of the line. Then the constructed surface equals (ES == ExtentScale): X(r, Gamma) = Px + r * Dx * ES + r * Gamma * Dy * ES Y(r, Gamma) = Py - r * Gamma * Dx * ES + r * Dy * ES Z(r, Gamma) = Gamma If C(t) is a higher order, non linear, curve then we compute trivar: X(t, r, Gamma) = Cx(t) + r * Cx'(t) * ES + r * Gamma * Cy'(t) * ES Y(t, r, Gamma) = Cy(t) - r * Gamma * Cx'(t) * ES + r * Cy'(t) * ES Z(t, r, Gamma) = Gamma

See also: MvarCrvKernelSilhouette, MvarCrvKernel, MvarCrvGammaKernel,

8.2.187 MvarCrvKernel (crv_krnl.c:123)

```
MvarMVStruct *MvarCrvKernel(const CagdCrvStruct *Crv)
```

Crv: Simple closed curve to compute its kernel.

Returns: The trivariate function F(u, v, t) whose zero set, projected on the XY plane, is the domain that is not in the kernel.

Description: Computes the kernel of the given curve, or the points in the plane that have a line of sight with all the points of the curve. The curve is assumed to be closed and simple. Let the input curve be C(t) and let S(u, v) = (u, v), the XY plane. Then, let F(u, v, t) = (C(t) - S(u, v)) x C'(t) (only the Z component of cross prod). The zero set of F projected over the XY plane defines all the domain that is NOT in the kernel of C(t).

See also: MvarCrvKernelSilhouette, MvarCrvGammaKernel, SymbCrvDiameter, , MvarCrvKernelPoint, MvarCrvKernelPoint,

8.2.188 MvarCrvKernelPoint (crv_krnl.c:872)

```
IPObjectStruct *MvarCrvKernelPoint(const CagdCrvStruct *Crvs,
                                   CagdRType SubEps,
                                   int OneKernelPoint,
                                   IrtRType PreciseBBoundTol,
                                   CagdRType Gamma,
                                   int NumTanSamples,
                                   CagdBType CnvxHull)
```

Crvs: A set of curves to compute a kernel point for.

SubEps: Subdivision epsilon.

OneKernelPoint: TRUE to return a kernel point, FALSE to return all points found.

PreciseBBoundTol: Zero to use regular bounding boxes on constraints. Positive integer >= 1 for recursive refinements of this amount for more precise BBox bounds. A positive fraction (<1.0) for the tolerance to use in precise BBox computations.

Gamma: If Gamma > 0, then compute the points that belong to the gamma kernel of the surfaces. Angle must be specified in degrees.

NumTanSamples: If > 0, then NumTanSamples tangent lines are sampled along each curve to purge the domains that do not belong to the the kernel of the curves. If a xy-domain is outside one of the sampled tangent lines, then the domain is purged during subdivision.

CnvxHull: If TRUE, then output is a convex hull of kernel points found. If FALSE, then output is a list of the kernel points in R^2.

Returns: A convex hull of the kernel points if CnvxHull is TRUE. A list of kernel points in R^2, otherwise.

Description: Computes a kernel point, if any, for the given curves. Let the input curve be C_i(t) and let N_i(t) be the normal filed of C_i(t), where i = 1, ..., n. Then: f_i(t_i, x, y) = < C_i(t)-P(x, y), N_i(t) >. If f_i(t_i, x, y) > 0, for all t_i and all i, then (x, y) is a kernel point. Computed only with semi-algebraic (inequality) constraints.

See also: MvarCrvKernel, MvarCrvAntipodalPoints, MvarCrvArtGalleryPoint,

8.2.189 MvarCrvKernelSilhouette (crv_krnl.c:436)

```
MvarPolylineStruct *MvarCrvKernelSilhouette(const CagdCrvStruct *Crv,
                                            CagdRType Gamma,
                                            CagdRType SubEps,
                                            CagdRType NumEps)
```

Crv: Simple closed curve to compute the silhouette of the kernel.

Gamma: Angular deviation of the gamma-kernel, in degrees.

SubEps: Subdivision epsilon.

NumEps: Numeric marching tolerance.

Returns: The silhouettes along the third, t, parameter of the trivariate function f(u, v, t).

Description: Computes the kernel of the given curve, or the points in the plane that have a line of sight with all the points of the curve. The curve is assumed to be closed and simple. Let the input curve be C(t) and let S(u, v) = (u, v), the XY plane. Then, let f(u, v, t) = < C(t) - S(u, v), C'(t) >. The zero set of f projected over the XY plane defines all the domain that is NOT in the kernel of C(t).
See also: MvarCrvKernel, MvarCrvGammaKernel,

8.2.190 MvarCrvListPreciseBBox (mvarbbox.c:1351)

bbox

bounding box

```
void MvarCrvListPreciseBBox(const CagdCrvStruct *Crvs,
                            MvarBBoxStruct *BBox,
                            CagdRType Tol)
```

Crvs: To compute a precise bounding box for.

BBox: Where bounding information is to be saved.

Tol: Tolerance of computations.

Returns: void

Description: Computes the precise bounding box for a list of curves.
See also: CagdCrvListBBox, MvarCrvPreciseBBox,

8.2.191 MvarCrvMakeCtlPtParam (mvar_rev.c:464)

```
MvarMVStruct *MvarCrvMakeCtlPtParam(const CagdCrvStruct *Crv,
                                    int CtlPtIdx,
                                    CagdRType Min,
                                    CagdRType Max)
```

Crv: Curve to make its i'th control point a parameter.

CtlPtIdx: Index of control point to make a parameter.

Min, Max: Domain each coordinate of CtlPtIdx point should vary.

Returns: A multivariate parametrizing both the original curve and the CtlPtIdx's control points Euclidean values.

Description: Given a polynomal curve and an index of a control point, CtlPtIdx, construct a new multiariate where the i'th control points is mapped to 2nd and above parameters of the returned multivariate. The first parameter of the multivariate remains the input curves' parameter. For example, if the input curve is E2 (planar), a trivariate will be returned, M(t, x, y), where t is the original curve's parameter and x and y parametrize the Euclidean values of the CtlPtIdx control point.

8.2.192 MvarCrvMaxXYOriginDistance (lnsrfdst.c:32)

```
CagdRType MvarCrvMaxXYOriginDistance(const CagdCrvStruct *Crv,
                                     CagdRType Epsilon,
                                     CagdRType *Param)
```

Crv: To examine for its maximal XY distance for the origin.

Epsilon: Tolerance of computation.

Param: Will be set with the parameter location where this maximum occur.

Returns: The distance.

Description: Computes the maximal XY distance location of given curve from origin.

8.2.193 MvarCrvPreciseBBox (mvarbbox.c:1283)

bbox

bounding box

```
void MvarCrvPreciseBBox(const CagdCrvStruct *Crv,
                        MvarBBoxStruct *BBox,
                        CagdRType Tol)
```

Crv: To compute a precise bounding box for.

BBox: Where bounding information is to be saved.

Tol: Tolerance of computations.

Returns: void

Description: Computes the precise bounding box for a curve freeform.
See also: MvarMVBBox, MvarMVPreciseBBox, CagdCrvBBox,

8.2.194 MvarCrvSelfInterDiagFactor (selfintr.c:666)

```
MvarPtStruct *MvarCrvSelfInterDiagFactor(const CagdCrvStruct *Crv,
                                         CagdRType SubdivTol,
                                         CagdRType NumericTol)
```

Crv: To detect its self intersection points.

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the surfaces.

NumericTol: Numeric tolerance of a possible numeric improvment stage. The numeric stage is employed if NumericTol < SubdivTol.

Returns: Self intersection points, as points in E4 (r, t).

Description: Computes self intersection points for given curve using the following constraints r and t are two independent params of the same crv:

```
x(r) - x(t) = 0,
y(r) - y(t) = 0.
```

Direct attempt to solve this set of constraints is bound to be slow as all points in Crv satisfy these equations when r == t. The key here is to remove all (u - v) factors off diagonal Bezier patches of the above.
See also: MvarCrvAntipodalPoints, MvarSrfSelfInterNrmlDev, , MvarCrvSelfInterNrmlDev,

8.2.195 MvarCrvSelfInterNrmlDev (selfintr.c:483)

```
MvarPtStruct *MvarCrvSelfInterNrmlDev(const CagdCrvStruct *CCrv,
                                      CagdRType SubdivTol,
                                      CagdRType NumericTol,
                                      CagdRType MinNrmlDeviation)
```

CCrv: To detect its self intersection points.

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the surfaces.

NumericTol: Numeric tolerance of a possible numeric improvment stage. The numeric stage is employed if NumericTol < SubdivTol.

MinNrmlDeviation: At the intersection points. Zero for 90 degrees minimum deviation, positive for smaller minimal deviation and negative for a larger minimal deviation.

Returns: Self intersection points, as points in E4 (r, t).

Description: Computes self intersection points for given curve using the following constraints r and t are two independent params of the same crv):

```
x(r) - x(t) = 0,
y(r) - y(t) = 0.
```

Direct attempt to solve this set of constraints is bound to be slow as all points in Crv satisfy these equations when r == t. The key is in adding a third inequality constaint of the form

```
     < N(r), N(t) >
---------------------- < Cos(Angle),
|| N(r) ||  || N(t) ||
```

Where Cos(Angle) is a provided constant that prescribes the minimal angle the curve is expected to intersect at. The closer Cos(Angle) to one the more work this function will have to do in order to isolate the self intersection points. The above expression is not rational and so, we use a logical or of the following two expressions:

```
     < N(r), N(t) >^2
------------------------- < Cos^2(Angle),
|| N(r) ||^2 || N(t) ||^2
```

or

```
     < N(r), N(t) > < 0.
```

See also: MvarCrvAntipodalPoints, MvarSrfSelfInterNrmlDev,

8.2.196 MvarCrvSrfBisector (mvbisect.c:114)

bisectors

```
MvarMVStruct *MvarCrvSrfBisector(const MvarMVStruct *CMV1,
                                 const MvarMVStruct *CMV2)
```

CMV1: The univariate (curve) in R^4.

CMV2: The bivariate (surface) in R^4.

Returns: The resulting bisector.

Description: Computes the bisectors of a curve and a surface in R^4.
See also: MvarMVsBisector, MvarSrfSrfBisector, MvarCrvSrfBisectorApprox,

8.2.197 MvarCrvSrfBisectorApprox (mvbisect.c:705)

bisectors

```
MvarZeroSolutionStruct *MvarCrvSrfBisectorApprox(const MvarMVStruct *CMV1,
                                                 const MvarMVStruct *CMV2,
                                                 CagdRType SubdivTol,
                                                 CagdRType NumericTol)
```

CMV1: The univariate (curve) in R^3.

CMV2: The bivariate (surface) in R^3.

SubdivTol: Tolerance of the first zero set finding subdivision stage.

NumericTol: Tolerance of the second zero set finding numeric stage.

Returns: An approximation of the bisector surface, either by polyline loops or triangles.

Description: Computes an approximation to the bisector of a curve and a surface. Let C(t) be the parametric curve and T(t) its unnormalized tangent field. Let S(u,v) be the parametric surface and n(u, v) its unnormalized normal. Then:

```
< P - C(t), T(t) > = 0    defines the normal plane of T(t),
```

and the solution of

```
< S(u, v) + n(u, v) Alpha - C(t), T(t) > = 0
```

finds the intersection point of of the normal of the surface with the normal plane of the curve. Then,

$$\text{Alpha}(t, u, v) = \frac{\langle C(t) - S(u, v), T(t) \rangle}{\langle n(u, v), T(t) \rangle}$$

We now can define this intersection point, P, as
$P(u, v, t) = S(u, v) + \text{Alpha}(u, v, t)\ n(u, v)$
and end up with a single function we must extract its zero set

```
< C(t) - P(u, v, t), C(t) - P(u, v, t) > -
                 < S(u, v) - P(u, v, t), S(u, v) - P(u, v, t) > = 0
```

or

```
< C(t) - P(u, v, t), C(t) - P(u, v, t) > -
                 < Alpha(u, v, t) n(u, v), Alpha(u, v, t) n(u, v) > = 0
```

with simple algebraic manipulation, the following equivalent form is is obtained, and is a polynomial (not rational) function, assuming the the input is not rational:

```
< S(u, v) - C(t), S(u, v) - C(t) > < n(u, v), T(t) > +
          2 < C(t) - S(u, v), T(t) > <S(u, v) - C(t), n(u, v) > = 0
```

Finding the zero set of the last equation provides the solution in (t, u, v) space, and the bisector point is given by:

```
          (x, y, z) = S(u, v) + Alpha(t, u, v) n(u, v)
```

which is implemented as a post-process mapping.

See also: MvarMVsBisector, MvarSrfSrfBisector, MvarCrvSrfBisector, MvarSrfSrfBisectorApprox,

8.2.198 MvarCrvSrfBisectorApprox2 (mvbisect.c:463)

bisectors

```
VoidPtr MvarCrvSrfBisectorApprox2(const MvarMVStruct *CMV1,
                                  const MvarMVStruct *CMV2,
                                  int OutputType,
                                  CagdRType SubdivTol,
                                  CagdRType NumericTol)
```

CMV1: The univariate (curve) in R^3.

CMV2: The bivariate (surface) in R^3.

OutputType: Expected output type: 1. For the computed multivariate constraints. 2. For the computed point cloud on the bisector. 3. Points in a form of (u, v, x, y, z) where (u, v) are the parameter space of the surface.

SubdivTol: Tolerance of the first zero set finding subdivision stage.

NumericTol: Tolerance of the second zero set finding numeric stage.

Returns: Following OutputType, either a set of multivariates (as a linked list of MvarMVStruct), or a cloud of points on the bisector (as a linked list of MvarPtStruct).

Description: Computes an approximation to the bisector of a curve and a surface. Let C(t) be the parametric curve and T(t) its unnormalized tangent field. Let S(u,v) be the parametric surface and n(u, v) its unnormalized normal. Then:

```
< P - C(t), T(t) > = 0    defines the normal plane of T(t),
```

and the solution of

```
< S(u, v) + n(u, v) Alpha - C(t), T(t) > = 0
```

finds the intersection point of of the normal of the surface with the normal plane of the curve. Then,

$$\text{Alpha}(u, v, t) = \frac{\langle C(t) - S(u, v), T(t) \rangle}{\langle n(u, v), T(t) \rangle}$$

We now can define this intersection point, P, as
P(u, v, t) = S(u, v) + Alpha(u, v, t) n(u, v)
and end up with a single function we must extract its zero set

```
< C(t) - P(u, v, t), C(t) - P(u, v, t) > -
                 < S(u, v) - P(u, v, t), S(u, v) - P(u, v, t) > = 0
```

or

```
< C(t) - P(u, v, t), C(t) - P(u, v, t) > -
                 < Alpha(u, v, t) n(u, v), Alpha(u, v, t) n(u, v) > = 0
```

Finding the zero set of the last equation provides the correspondance between the (u, v) location and the surface and (t) locations on the curve that serve as mutual foot point for some bisector point.

See also: MvarMVsBisector, MvarSrfSrfBisector, MvarCrvSrfBisector, MvarSrfSrfBisectorApprox,

8.2.199 MvarCrvSrfInter (lnsrfdst.c:340)

```
MvarPtStruct *MvarCrvSrfInter(const CagdCrvStruct *Crv,
                              const CagdSrfStruct *Srf,
                              CagdRType SubdivTol,
                              CagdRType NumericTol)
```

Crv: Curve to intersect.

Srf: Surface to intersect.

SubdivTol: Subdivision tolerance of computation.

NumericTol: Numerical tolerance of computation.

Returns: List of intersection points as parameters (t,u,v).

Description: Computes intersection points of a surface and a curve.
See also: MvarLineSrfInter,

8.2.200 MvarCrvSrfMinimalDist (hasdrf3d.c:864)

```
MvarPtStruct *MvarCrvSrfMinimalDist(const CagdSrfStruct *Srf1,
                                    const CagdCrvStruct *Crv2,
                                    CagdRType *MinDist)
```

Srf1: To detect its minimal distance to Crv2.

Crv2: To detect its minimal distance to Srf1.

MinDist: Upon return, is set to the minimal distance detected.

Returns: Pairs of parameters at the minimal distance.

Description: Computes the minimal distance between a given C1 curve and a C1 surface. Shapes are assumed to not intersect. This minimal distance can occur:

1. At end points vs. end points.
2. At the end points vs interior locations.
3. At a boundary curve of the surface vs the other curves.
4. At antipodal interior locations.

See also: MvarSrfSrfAntipodalPoints, MvarDistSrfPoint, MvarSrfSrfMinimalDist,

8.2.201 MvarCrvSrfMinkowskiSum (mink_sum.c:286)

```
struct IPObjectStruct *MvarCrvSrfMinkowskiSum(const CagdCrvStruct *Crv,
                                              const CagdSrfStruct *Srf,
                                              CagdRType SubdivTol,
                                              CagdRType CrvTraceStep,
                                              CagdRType NumericTol,
                                              CagdRType OffsetTrimDist)
```

Crv, Srf: The curve and surface to compute their Minkowski sum.

SubdivTol: The subdivision tolerance for the solver call.

CrvTraceStep: Univariate curve tracing step size.

NumericTol: The numeric tolerance for the solver call.

OffsetTrimDist: When positive, indicates that the Minkowski sum is an offset problem, i.e. Srf is a sphere of radius OffsetTrimDist, and the value is used in the offset trimming. If negative, trimming is skipped (the problem is not an offset problem, or any other reason).

Returns: List of polygons or polylines (according to the setting of the bivariate solver), approximating the Minkowski sum.

Description: Computation of the Minkowski sum of a curve and a surface. The sum is defined as the locus of points Crv(t) + Srf(u,v) in R^3, for t and (u,v) in the parameter spaces, such that the corresponding tangent vector T(t) is orthogonal to the surface normal and N(u,v), i.e. We solve:
< Crv_t , Srf_u x Srf_v > = 0.
The solution triangles (or polylines) in (t, u, v) space, are then mapped to the Euclidean space using the post-process mapping Crv(t) + Srf(u,v).
See also: MvarSrfSrfMinkSumMVs,

8.2.202 MvarCrvTrimGlblOffsetSelfInter (offset2.c:119)

```
CagdCrvStruct *MvarCrvTrimGlblOffsetSelfInter(CagdCrvStruct *Crv,
                                              const CagdCrvStruct *OffCrv,
                                              CagdRType TrimAmount,
                                              CagdRType SubdivTol,
                                              CagdRType NumericTol)
```

Crv: Original curve, assumed to be C1 self intersection free.

OffCrv: The offset curve approximation.

TrimAmount: The trimming distance. A fraction smaller than the offset amount.

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the surfaces.

NumericTol: Numeric tolerance of a possible numeric improvement stage. The numeric stage is employed if NumericTol < SubdivTol.

Returns: A list of curve segments that are valid, after the trimming process took place.

Description: Trims regions in the offset curve OffCrv that are closer than TrimAmount to original Crv, bu compute all self intersections in the offset curve, splitting the offset curve at those intersections, and purges offset curve segments that are too close to original curve.
See also: SymbCrvTrimGlblOffsetSelfInter,

8.2.203 MvarCrvTrimGlblOffsetSelfInter2 (offset2.c:565)

```
CagdCrvStruct *MvarCrvTrimGlblOffsetSelfInter2(const CagdCrvStruct *Crv,
                                               CagdRType OffsetDist,
                                               int Operation,
                                               CagdRType SubdivTol,
                                               CagdRType NumericTol)
```

Crv: Original curve, assumed to be C1.

OffsetDist: The expected offset distance.

Operation: 0 - return valid curve segments in original curve. 1 - return valid curve segments as offset curves (using SymbCrvAdapOffset). 2 - merge the offset curve segments (as in Operation = 1) into one final offset curve.

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the surfaces.

NumericTol: Numeric tolerance of a possible numeric improvement stage. The numeric stage is employed if NumericTol < SubdivTol.

Returns: A list of curve segments that are valid, after the trimming process took place.

Description: Trims regions in the offset curve of planar curve Crv are closer than TrimAmount to original Crv, bu computing all self intersections in the ffset curve. Constraints are define ONLY over the original curve. Then, Split the offset curve at those intersections, and purge offset curve segments that are too close to original curve.
See also: SymbCrvTrimGlblOffsetSelfInter, SymbSrfTrimGlblOffsetSelfInter, MvarCrvTrimGlblOffsetSelfInter, MvarCrvDtctSelfInterOffsetLocCreateOneMV,

8.2.204 MvarCrvTrimSelfInterLocations (offset2.c:1236)

```
CagdCrvStruct *MvarCrvTrimSelfInterLocations(const CagdCrvStruct *Crv,
                                             CagdRType OffsetDist,
                                             CagdRType SubdivTol,
                                             CagdRType NumericTol,
                                             CagdBType PurgeSelfInters)
```

Crv: Curve to detects its self intersection locations, if any.

OffsetDist: The desired offset distance.

SubdivTol, NumericTol: Tolerances of MV solver.

PurgeSelfInters: TRUE to purge self intersecting regions in the offset. If FALSE, all curves segments are returned after splitting.

Returns: Offset curve, clipped to non self-intersecting locations.

Description: Clips the regions in an offset by distance OffsetDist of Crv and returns the proper offset.
See also: MvarCrvCrvBisector2D, MvarCrvDtctSelfInterLocations, SymbCrvTrimGlblOffsetSelfInter, SymbSrfTrimGlblOffsetSelfInter, MvarCrvTrimGlblOffsetSelfInter, MvarCrvTrimGlblOffsetSelfInter2,

8.2.205 MvarCrvZeroSet (zrmvaux0.c:1274)

```
CagdPtStruct *MvarCrvZeroSet(const CagdCrvStruct *Curve,
                             int Axis,
                             CagdRType SubdivTol,
                             CagdRType NumericTol,
                             CagdBType FilterTangencies)
```

Curve: To compute its zeros.

Axis: of Crv to seek its zeros: 0 for W, 1 for X, 2 for Y, etc.

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the curves.

NumericTol: Numeric tolerance of a possible numeric improvement stage. The numeric stage is employed if NumericTol < SubdivTol.

FilterTangencies: If TRUE, filter out tangencies at the zeros.

Returns: List of zeros (parametric locations on CUrve).

Description: Computes the zeros of the given curve in the given axis.
See also: MvarSrfZeroSet, MvarMVsZeros0D,

8.2.206 MvarCtrlComputeCrvNCycle (control.c:48)

```
MvarPtStruct *MvarCtrlComputeCrvNCycle(const CagdCrvStruct *Crv,
                                       int CycLen,
                                       CagdRType SubdivTol,
                                       CagdRType NumerTol)
```

Crv: A curve to compute a cycle of length CycLen for.

CycLen: Length of sought cycle (also k above).

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the curves.

NumerTol: Numeric tolerance of a possible numeric improvment stage. The numeric stage is employed if NumericTol < SubdivTol.

Returns: Linked list of solutions, each holding the parameter values of the different Cycles, if any.

Description: Computes a cycle of length CycLen of bouncing lines between the given scalar curve C(t) and the diagonal of the domain. A ray is bounced 'down' from the diagonal to C and from C, we bounce a horizontal ray to the diagonal. Let Di, i == 1,k be the diagonal points and Ci, i == 1,k the points on C. Then, the cycle of length k = CycLen is: (D1, C1, D2, C2, ... Dk, Ck). The following algebraic constraints can be imposed:

```
Ci(y) = Di+1(y), for all i, (horizontal move from curve to diagonal)
Di(x) = Ci(x),   for all i. (vertical move from diagonal to curve)
```

Over all, we have 2n equations and 2n dofs. We can reduce them to n equations and dofs (eliminating the diagonals):

```
Ci(y) = Ci+1(x),   for all i.
```

Over all, we have now n equations and n dofs (n different parameters of C).
See also: MvarCtrlComputeSrfNCycle,

8.2.207 MvarCtrlComputeSrfNCycle (control.c:186)

```
MvarPtStruct *MvarCtrlComputeSrfNCycle(const CagdSrfStruct *Srf,
                                       int CycLen,
                                       CagdRType SubdivTol,
                                       CagdRType NumerTol)
```

Srf: A surface to compute a cycle of length CycLen for.

CycLen: Length of sought cycle (also k above).

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the surface.

NumerTol: Numeric tolerance of a possible numeric improvment stage. The numeric stage is employed if NumericTol < SubdivTol.

Returns: Linked list of solutions, each holding the parameter values of the different Cycles, if any.

Description: Computes a cycle of length CycLen of bouncing lines between the given surface S(u, v) = (S1(u, v), S2(u, v)). The cycle of length k = CycLen is:

```
S(u1, v1) -> (u2, v)
S(ui, vi) -> (u(i+1), v(i+1))
S(uk, vk) -> (u1, v1)
```

Over all, we have 2n equations and 2n dofs.
See also: MvarCtrlComputeCrvNCycle,

8.2.208 MvarDbg (mvar_dbg.c:31)

debugging

```
void MvarDbg(const void *Obj)
```

Obj: A multi-variate - to be printed to stderr.

Returns: void

Description: Prints multi-variates to stderr. Should be linked to programs for debugging purposes, so multi-variates may be inspected from a debugger.
See also: MvarDbg1,

8.2.209 MvarDbg1 (mvar_dbg.c:73)

debugging

```
void MvarDbg1(const void *Obj)
```

Obj: A multi-variate - to be printed to stderr.

Returns: void

Description: Prints multi-variates to stderr. Should be linked to programs for debugging purposes, so multi-variates may be inspected from a debugger.
See also: MvarDbg,

8.2.210 MvarDbgDsp (mvar_dbg.c:104)

debugging

```
void MvarDbgDsp(const void *Obj)
```

Obj: A multivariate - to be displayed.

Returns: void

Description: Views multivariates in a display device. Should be linked to programs for debugging purposes, so multivariates may be inspected from the debugger.

8.2.211 MvarDbgInfo (mvar_dbg.c:198)

debugging

```
void MvarDbgInfo(const void **Objs, int Size)
```

Objs: A vector of multi-variate - to print statistics on to stderr.

Size: Size of vector.

Returns: void

Description: Prints multi-variates statistics to stderr. Should be linked to programs for debugging purposes, so multi-variates may be inspected from a debugger.

8.2.212 MvarDbgMVZR1DExamineSegmentBndry (zrmvaux1.c:403)

```
void MvarDbgMVZR1DExamineSegmentBndry(MvarPtStruct *PtList,
                                      const MvarMVStruct *MV)
```

PtList: One traced segment to examine...

MV: If PtList starting/ending on MV boundary or closed.

Returns: void

Description: Verify the given segment is closed or on MV's boundary.

8.2.213 MvarDbgMVZR1DPrintEndPtPlList (zrmvaux1.c:368)

```
void MvarDbgMVZR1DPrintEndPtPlList(MvarPolylineStruct *PolyList)
```

PolyList: List of polylines.

Returns: void

Description: Print the end points of a list of polylines.

8.2.214 MvarDevelopSrfFromCrvSrf (pdvl_alg.c:840)

```
IPObjectStruct *MvarDevelopSrfFromCrvSrf(const CagdSrfStruct *Srf,
                                         const CagdCrvStruct *Crv,
                                         const CagdCrvStruct *OrientField,
                                         int CrvSizeReduction,
                                         CagdRType SubdivTol,
                                         CagdRType NumericTol,
                                         CagdBType Euclidean)
```

Srf: A general surface to compute developable scroll from it to Crv.

Crv: A general curve to compute developable scroll from it to Srf.

OrientField: If exists, prescribes where to look for solutions of the next developable in the surface (u, v) space, along Crv params. Assumed to have the same parameterization as Crv.

CrvSizeReduction: A reduction in size of traced curve while ensuring the Tolerance, conservatively.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol < SubdivTol.

Euclidean: TRUE to return the construct developable scroll(s) in the Euclidean space, FALSE to return the result in (u, v, t) space.

Returns: A developable scroll surface in Euclidean space if Euclidean (or a linked list of such), or a (list of) curve in (u, v, t) parametric space of the scroll(s), or NULL if none.

Description: Computes a developable scroll between the given curve, Crv, and surface, Srf. The developable scroll will be tangent to both Crv and Srf and will be a ruled surface in between (with same surface normal along rulings). Computed by solving the following two equations with three unknowns:

```
< C(t) - S(u, v), n(u, v) > = 0,
< n(u, v), C'(t) > = 0.
```

See also: MvarDevelopSrFfromCrvSrfMakeSrfs,

8.2.215 MvarDevelopSrfFromCrvSrfMakeSrfs (pdvl_alg.c:1011)

```
CagdSrfStruct *MvarDevelopSrfFromCrvSrfMakeSrfs(const CagdCrvStruct *Crv,
                                                const CagdSrfStruct *Srf,
                                                const CagdCrvStruct *UVTCrvs,
                                                int CrvSizeReduction)
```

Crv: A general curve to compute developable scroll from it to Srf.

Srf: A general surface to compute developable scroll from it to Crv.

UVTCrvs: The solution as polylines in (u, v, t) space.

CrvSizeReduction: A reduction in size of traced curve while ensuring the Tolerance, conservatively.

Returns: Euclidean space developable surface.

Description: Evaluate the given solution to curve(s) in (u, v, t) space to surfaces between the Crv and (tangential point on) Srf.

See also: MvarDevelopSrfFromCrvSrf,

8.2.216 MvarDistCrvOfPtFromSrf (mvardist.c:2367)

```
CagdCrvStruct *MvarDistCrvOfPtFromSrf(const CagdRType *Pt,
                                      const CagdSrfStruct *Srf,
                                      CagdRType Dist,
                                      CagdRType Step,
                                      CagdRType SubdivTol,
                                      CagdRType NumerTol,
                                      int LstSqrFit)
```

Pt: Point in Euclidean space to compute the fixed distance curve(s) for.

Srf: Surface to compute the fixed distance curve(s) on.

Dist: Euclidean distance of the curves.

Step: Step size during the numeric tracing.

SubdivTol: The subdivision tolerance to use.

NumerTol: The numerical tolerance to use.

LstSqrFit: Number of control points to allow in fit, zero to disable.

Returns: Computed curves in UV space or NULL if not found.

Description: Computes curve(s) on Srf that are at a Euclidean distance Dist from Pt. Solves the following equation in two variables:

```
|| Srf(u, v) - Pt ||^2 = Dist^2.
```

8.2.217 MvarDistPointCrvC1 (hasdrf2d.c:57)

```
CagdRType MvarDistPointCrvC1(CagdPType P,
                             const CagdCrvStruct *Crv,
                             MvarHFDistParamStruct *Param,
                             CagdBType MinDist,
                             CagdRType Epsilon)
```

P: Point to measure its Hausdorff distance to curve Crv.

Crv: Crv to measure its hausdorff distance to point Pt.

Param: Where to return the parameter value with the maximal distance.

MinDist: TRUE for minimal distance, FALSE for maximal.

Epsilon: Tolerance of computation.

Returns: The Hausdorff distance.

Description: Computes the min or max distance between a point and a C^1 cont. curve. The curve and point are assumed to be either in R2 or R3. The extreme distance between a point and a curve could happen either at the end points of curve C or when

```
C'(t) <C(t) - P> = 0.
```

See also: SymbDistCrvPoint,

8.2.218 MvarDistPointLine (mvar_int.c:241)

point line distance

```
IrtRType MvarDistPointLine(const MvarVecStruct *Point,
                           const MvarVecStruct *Pl,
                           const MvarVecStruct *Vl)
```

Point: To find the closest to on the line.

Pl, Vl: Position and direction that defines the line. Vl is assumed a unit length vector.

Returns: The computed minimal distance.

Description: Routine to compute the minimal point distance to a given line in R^n. The line is prescribed using a point on it (Pl) and a unit vector (Vl).
See also: GMDistPointLine,

8.2.219 MvarDistSrfLine (mvardist.c:490)

surface line distance

```
CagdRType *MvarDistSrfLine(const CagdSrfStruct *Srf,
                           const CagdPType LnPt,
                           const CagdVType LnDir,
                           CagdBType MinDist,
                           CagdRType SubdivTol,
                           CagdRType NumericTol,
                           CagdUVType ExtremeDistUV)
```

Srf: The surface to find its nearest (farthest) point to Line.

LnPt: A point on the line to consider.

LnDir: The direction of the line to consider.

MinDist: If TRUE nearest points is needed, if FALSE farthest.

SubdivTol: Tolerance of the first zero set finding subdivision stage.

NumericTol: Tolerance of the second zero set finding numeric stage.

ExtremeDistUV: Parameter value in the parameter space of Srf of the nearest (farthest) point to line Line.

Returns: Parameter value in the parameter space of Srf of the nearest (farthest) point to line Line.

Description: Given a surface and a line, finds the nearest point (if MinDist) or the farthest location (if MinDist FALSE) from the surface to the given line. This function assumes the surface does not intersect the line. Returned is the parameter value of the surface. Only internal extrema are considered. Let S and N be the surface and its normal field. Then the extrema points are computed as the simultaneous solution of,

```
< (S - LnPt) x N, LnDir > = 0,
< N, LnDir > = 0.
```

See also: MvarLclDistSrfLine, MvarLclDistCrvLine, MvarDistCrvLine,

8.2.220 MvarDistSrfPoint (mvardist.c:104)

surface point distance

```
CagdRType *MvarDistSrfPoint(const CagdSrfStruct *Srf,
                            void *SrfPtPrepHandle,
                            const CagdPType Pt,
                            MvarPtStruct **InitialSol,
                            CagdBType MinDist,
                            CagdRType SubdivTol,
                            CagdRType NumericTol,
                            MvarPtStruct **ExtremePts,
                            CagdRType *ExtremeDistUV,
                            CagdPointType DistSpace)
```

Srf: The surface to find its nearest (farthest) point to Pt.

SrfPtPrepHandle: If not NULL, holds pre-processed data to speed up the surface - points distance computations, for multiple surface - point tests (same surface for all points). See MvarDistSrfPointPrep and MvarDistSrfPointFree.

Pt: The point to find the nearest (farthest) point on Srf to it.

InitialSol: Optional initial guess for a solution in which case we aim to improve numerically to within NumericTol. If successful, this solution is returned (in place here and the return value). Otherwise, the full solver is invoked.

MinDist: If TRUE nearest points is needed, if FALSE farthest.

SubdivTol: Tolerance of the first zero set finding subdivision stage.

NumericTol: Tolerance of the second zero set finding numeric stage.

ExtremePts: If not NULL, will be set to hold all dist solution, as can be more than one.

ExtremeDistUV: UV Parameter values in the parameter space of Srf of the nearest (farthest) point to point Pt.

DistSpace: The (point) space to compute the distance to surface. CAGD_PT_NONE to inherit from the input surface.

Returns: UV Parameter values in the parameter space of Srf of the nearest (farthest) point to point Pt.

Description: Given a surface and a point, finds the nearest point (if MinDist) or the farthest location (if MinDist FALSE) from the surface to the given point. Returned is the parameter value of the surface. Both internal as well as boundary extrema are considered. Computes the simultaneous zeros of:

```
(Srf(u, v) - Pt) . dSrf(u, v)/Du = 0,
(Srf(u, v) - Pt) . dSrf(u, v)/Dv = 0,
```

and also include all extrema on the boundaries.

8.2.221 MvarDistSrfPointFree (mvardist.c:310)

curve point distance

```
void MvarDistSrfPointFree(void *SrfPtPrepHandle)
```

SrfPtPrepHandle: Handle on the preprocessed surface for multiple point - curve test, to free.

Returns: void

Description: Free the pre-processed data structure, given using Handle.
See also: MvarLclDistSrfPoint, MvarDistSrfPointPrep,

8.2.222 MvarDistSrfPointPrep (mvardist.c:263)

curve point distance

```
void *MvarDistSrfPointPrep(const CagdSrfStruct *CSrf)
```

CSrf: The curve to pre-process for multi-point - curve tests.

Returns: A handle on a structure to boost multi-point - curve tests.

Description: Given a curve, pre-process it so many curve-distance test can be efficiently made.
See also: MvarDistSrfPointFree, MvarLclDistSrfPoint,

8.2.223 MvarETDbg (mvar_dbg.c:237)

debugging

```
void MvarETDbg(const MvarExprTreeStruct *ET)
```

ET: A multivariate expression tree - to be printed to stderr.

Returns: void

Description: Prints multivariate expression tree to stderr. Should be linked to programs for debugging purposes, so multi-variates may be inspected from a debugger.

8.2.224 MvarETDomain (mvarextr.c:1585)

```
int MvarETDomain(const MvarExprTreeStruct *ET,
                 CagdRType *Min,
                 CagdRType *Max,
                 int Axis)
```

ET: Expression tree to derive the domain of, in direction Axis.

Min, Max: Domain of ET in direction Axis.

Axis: The direction along with to compute the domain of ET, or -1 to evaluate the domain in all direction (Min and Max should be vectors of size ET -> Dim). m

Returns: TRUE for successful computation, FALSE otherwise.

Description: Computes the domain of the expression tree along direction Axis. The assumption is that any MV in ET with a non trivial degree in direction Axis possess the same domain.
See also: MvarExprTreeDomain,

8.2.225 MvarETUpdateConstDegDomains (mvarextr.c:1765)

```
int MvarETUpdateConstDegDomains(MvarExprTreeStruct **MVETs, int n)
```

MVETs: vectors of expression tree MVs to update.

n: Size of vector MVETs.

Returns: TRUE if successful, FALSE otherwise.

Description: Compute the shared domain of all given expression trees and update all expression trees leafs to that domain, in their constant directions. Only constant degree directions can be updated. Higher order degree of similar directions in two different (sub) expression trees must be sharing the same domain (or otherwise an error will be returned).
See also: MvarMVUpdateConstDegDomains,

8.2.226 MvarETsZeros0D (zrsolver.c:667)

```
MvarPtStruct *MvarETsZeros0D(MvarZeroPrblmSpecStruct *ZeroProblemSpec)
```

ZeroProblemSpec: Specification of the 0D ET problem, which includes the number of constraints, the multivariates constraints, constraints types, tolerances, call back functions, etc.

Returns: List of points on the solution set. Dimension of the points will be the same as the dimensions of all MVETs.

Description: Interface function for a MV expression trees solver, 1D solutions. Computes the simultaneous solution of the given set of NumOfMVs expression trees constraints. A constraint can be equality or inequality as prescribed by the Constraints vector. All multivariates are assumed to be in the same parametric domain size and dimension.
See also: MvarMVsZeros0D,

8.2.227 MvarEditSingleMVPt (mvaredit.c:34)

multi-variate editing

```
MvarMVStruct *MvarEditSingleMVPt(MvarMVStruct *MV,
                                 CagdCtlPtStruct *CtlPt,
                                 int *Indices,
                                 CagdBType Write)
```

MV: Multi-variate to be modified/query.

CtlPt: New control point to be substituted into MV. Must carry the same PType as MV if to be written to MV.

Indices: In multi-variate MV's control mesh to substitute/query CtlPt.

Write: If TRUE CtlPt is copied into MV, if FALSE the point is copied from MV to CtlPt.

Returns: If Write is TRUE, the new modified multi-variate, if WRITE is FALSE, NULL.

Description: Provides the way to modify/get a single control point into/from a multi-variate.

8.2.228 MvarEucCrvOffsetOnSrf (mvardist.c:1973)

```
CagdCrvStruct *MvarEucCrvOffsetOnSrf(const CagdCrvStruct *Crv,
                                     const CagdSrfStruct *Srf,
                                     CagdRType Dist,
                                     CagdRType Step,
                                     CagdRType SubdivTol,
                                     CagdRType NumerTol,
                                     int Orient,
                                     int LstSqrFit)
```

Crv: Curve in E3 to compute its Euclidean offset curves on Srf, assumed C^1.

Srf: Surface on which to seek parallel Euclidean offset curve(s) to Crv.

Dist: Euclidean distance of the offset.

Step: Step size during the numeric tracing.

SubdivTol: The subdivision tolerance to use.

NumerTol: The numerical tolerance to use.

Orient: +/-1 to request left/right orientation inequality filter, or 0 to disable.

LstSqrFit: Number of control points to allow in fit, zero to disable.

Returns: Computed curves in UVT space or NULL if not found.

Description: Computes curve(s) on Srf that are at a Euclidean distance Dist from Crv. Such curves can be consider Euclidean offset curves of Crv on Srf. Solves the following set of two equations in three variables:

```
|| Crv(t) - Srf(u, v) ||^2 = Dist^2.
< Crv(t) - Srf(u, v), Crv'(t) > = 0.
```

and optionally:

```
< (Crv'(t) x (Crv(t) - Srf(u, v)), N(u, v) > > 0,
```

where N(u, v) is the normal field of Srf.

See also: MvarSpiralCrvOnSrf,

8.2.229 MvarEvalCnvxHullRegion4Curves (crv_krnl.c:1403)

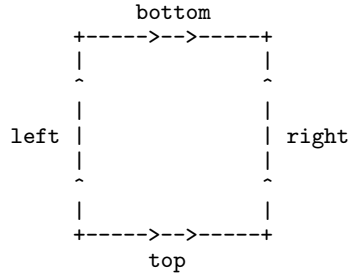
```
IPObjectStruct *MvarEvalCnvxHullRegion4Curves(const CagdCrvStruct *Left,
                                              const CagdCrvStruct *Right,
                                              const CagdCrvStruct *Top,
                                              const CagdCrvStruct *Bottom,
                                              IrtRType SubEps)
```

Left, Right, Top, Bottom: The four boundary curves of the closed domain.

SubEps: Subdivision epsilon. 0.01 is a reasonable start

Returns: The constructed convex hull.

Description: Computes the convex hull of the given four boundary curves. The orientation of the curves should be like the Boolean sum input.



See also:

8.2.230 MvarEvalKrnlPtsTV (krnl_based_cnstrct.c:1390)

```
IPObjectStruct *MvarEvalKrnlPtsTV(TrivTVStruct *TV, IrtRType SubEps)
```

TV: A trivariate.

SubEps: Subdivision epsilon. For example, 0.005 - For good kernel results, expensive computation time. 0.01 - For resonable kernel results, resonable computation time.

Returns: The kernel points list.

Description: The function computes the kernel points of the given trivarite, if exists.
See also:

8.2.231 MvarExprAuxDomainReset (mvarextr.c:1873)

```
void MvarExprAuxDomainReset(MvarExprTreeStruct *ET)
```

ET: Expression tree to reset all aux domain of Bezier MVs.

Returns: void

Description: Reset aux. domains of all bezier multivariates in the expression tree.
See also:

8.2.232 MvarExprTreeAdd (mvarextr.c:739)

```
MvarExprTreeStruct *MvarExprTreeAdd(MvarExprTreeStruct *Left,
                                    MvarExprTreeStruct *Right)
```

Left, Right: The two expression trees to add. These sub trees are used in place, so duplicate before calling this function if you like to keep the original Left/Right.

Returns: Added expression tree, "Left + Right".

Description: Adds two sub expression tree into one new expression tree.
See also:

8.2.233 MvarExprTreeBBox (mvarextr.c:1362)

```
const MvarBBoxStruct *MvarExprTreeBBox(MvarExprTreeStruct *ET)
```

ET: Expression Tree to compute the bounding box for.

Returns: Computed bounding box, in static memory.

Description: Computes the BBox of the given expression tree.
See also:

8.2.234 MvarExprTreeCnvrtBsp2BzrMV (mvarextr.c:2085)

```
int MvarExprTreeCnvrtBsp2BzrMV(MvarExprTreeStruct *ET,
                               MvarMinMaxType *Domain)
```

ET: Expression tree to convert Bspline MVs to Beziers, in place.

Domain: Optional domain to set the bezier ET as aux. domain. NULL for no setting of the aux. domain.

Returns: TRUE if successful, FALSE otherwise.

Description: Convert, in place, all Bspline MVs in expression tree ET to Beziers. All MVs are assumed to hold no interior knots. All created Bezier MVs are updated with their aux domain.
See also:

8.2.235 MvarExprTreeCnvrtBzr2BspMV (mvarextr.c:2149)

```
int MvarExprTreeCnvrtBzr2BspMV(MvarExprTreeStruct *ET)
```

ET: Expression tree to convert Bspline MVs to Beziers, in place.

Returns: TRUE if successful, FALSE otherwise.

Description: Convert, in place, all Bezier MVs in expression tree ET to Bspline.
See also:

8.2.236 MvarExprTreeCompositionDerivBBox (mvarextr.c:2756)

```
MvarBBoxStruct *MvarExprTreeCompositionDerivBBox(MvarExprTreeStruct *ET,
                                                 MvarBBoxStruct *BBox)
```

ET: To evaluate its gradient at given Params parametric location.

BBox: Where the result will be written to.

Returns: Returns the same result (for convinience).

Description: Bounds the derivative of the composition function. Currently only scalar composition allowed.
See also: MvarExprTreeBBox,

8.2.237 MvarExprTreeConesOverlap (mvcones.c:2171)

```
CagdBType MvarExprTreeConesOverlap(MvarExprTreeEqnsStruct *Eqns)
```

Eqns: The MVETs constraints formated into equations with * common expressions. *

Returns: TRUE if overlap, FALSE if not.

Description: Computes the tangency anti-cones of the set of multivariate constraints, and returns whether they overlap or not.
See also: MVarExprTreeNormalCone, MvarConesOverlapAux,

8.2.238 MvarExprTreeCopy (mvarextr.c:390)

```
MvarExprTreeStruct *MvarExprTreeCopy(const MvarExprTreeStruct *ET,
                                     CagdBType ThisNodeOnly,
                                     CagdBType DuplicateMVs)
```

ET: Expression tree to duplicate.

ThisNodeOnly: TRUE to duplicate just this node.

DuplicateMVs: If TRUE, Multivariates in leaf nodes are duplicated. Otherwise, references for them are kept instead.

Returns: Duplicated node or tree.

Description: A copy function to duplicate a node (ThisNodeOnly TRUE) or entire expression tree.
See also:

8.2.239 MvarExprTreeCos (mvarextr.c:1039)

```
MvarExprTreeStruct *MvarExprTreeCos(MvarExprTreeStruct *Left)
```

Left: The expression tree to take cosine of. The sub tree is used in place, so duplicate before calling this function if you like to keep the original Left.

Returns: Cosine expression tree, "Cos(Left)".

Description: Cosine of subexpression tree into a new expression tree.
See also:

8.2.240 MvarExprTreeCrossProd (mvarextr.c:947)

```
MvarExprTreeStruct *MvarExprTreeCrossProd(MvarExprTreeStruct *Left,
                                          MvarExprTreeStruct *Right)
```

Left, Right: The two expression trees to multiply. These sub trees are used in place, so duplicate before calling this function if you like to keep the original Left/Right.

Returns: Multiplied expression tree, "Left * Right".

Description: Cross Prod two sub expression tree into one new expression tree.
See also:

8.2.241 MvarExprTreeDotProd (mvarextr.c:912)

```
MvarExprTreeStruct *MvarExprTreeDotProd(MvarExprTreeStruct *Left,
                                        MvarExprTreeStruct *Right)
```

Left, Right: The two expression trees to multiply. These sub trees are used in place, so duplicate before calling this function if you like to keep the original Left/Right.

Returns: Multiplied expression tree, "Left * Right".

Description: Dot Prod two sub expression tree into one new expression tree.
See also:

8.2.242 MvarExprTreeEqnsFree (zret0d.c:354)

```
void MvarExprTreeEqnsFree(MvarExprTreeEqnsStruct *Eqns)
```

Eqns: Data structure to free.

Returns: void

Description: Free all data allocated in the expression tree equation's structure.
See also: MvarMVsZeros, MvarExprTreeEqnsMalloc,

8.2.243 MvarExprTreeEqnsMalloc (zret0d.c:310)

```
MvarExprTreeEqnsStruct *MvarExprTreeEqnsMalloc(int NumEqns,
                                               int MaxNumCommonExprs)
```

NumEqns: Number of equations we have.

MaxNumCommonExprs: Maximum number of common expression we can initially hold.

Returns: Allocated structure.

Description: Allocate a structure to hold NumEqns equations and at most MaxNumCommonExprs common expressions.
See also: MvarMVsZeros, MvarExprTreeEqnsFree,

8.2.244 MvarExprTreeEqnsReallocCommonExprs (zret0d.c:395)

```
void MvarExprTreeEqnsReallocCommonExprs(MvarExprTreeEqnsStruct *Eqns,
                                        int NewSize)
```

Eqns: Set of equations to increase, in place, the number of coomon expressions it can old.

NewSize: of vector of common expression, zero to double the size.

Returns: void

Description: Reallocate (increase) the number of common expressions give Eqns can hold, in place.
See also: MvarMVsZeros,

8.2.245 MvarExprTreeEval (mvarextr.c:2318)

```
CagdRType *MvarExprTreeEval(const MvarExprTreeStruct *ET,
                            CagdRType *Params,
                            CagdRType *Pt)
```

ET: Expression tree to evaluate.

Params: Parameter values to evaluate the expression at.

Pt: Evaluation result.

Returns: Evaluation result. Note entry zero is reserved to the rational (weight) value.

Description: Evaluate the expression tree at the given parametric location.
See also: MvarExprTreeGrad,

8.2.246 MvarExprTreeEvalTanPlane (mvarextr.c:2709)

```
MvarPlaneStruct *MvarExprTreeEvalTanPlane(const MvarExprTreeStruct *ET,
                                          CagdRType *Params)
```

ET: To evaluate its gradient at given Params parametric location.

Params: Parametric location to evaluate MV at.

Returns: A hyperplane, allocated dynamically. The tangent is normalized so that its last (independent coefficient is one: "A1 X1 + A2 X2 + ... + An Xn + 1". The size, n, is to the dimension of the multivariate.

Description: Evaluates the tangent hyperplane of the given ET at a given location, numerically.
See also: MvarMVsZeros,

8.2.247 MvarExprTreeExp (mvarextr.c:981)

```
MvarExprTreeStruct *MvarExprTreeExp(MvarExprTreeStruct *Left)
```

Left: The expression tree to exponent. The sub tree is used in place, so duplicate before calling this function if you like to keep the original Left.

Returns: Exponentiated expression tree, "e^Left".

Description: Exponent of subexpression tree into a new expression tree.
See also:

8.2.248 MvarExprTreeFree (mvarextr.c:554)

```
void MvarExprTreeFree(MvarExprTreeStruct *ET, CagdBType ThisNodeOnly)
```

ET: Expression tree to free.

ThisNodeOnly: TRUE to free just this node, FALSE for the entire tree.

Returns: void

Description: Free an expression tree node (ThisNodeOnly TRUE) or its entire tree.
See also: MvarExprTreeFreeSlots,

8.2.249 MvarExprTreeFreeSlots (mvarextr.c:482)

```
void MvarExprTreeFreeSlots(MvarExprTreeStruct *ET, CagdBType ThisNodeOnly)
```

ET: Expression tree to free.

ThisNodeOnly: TRUE to free just this node, FALSE for the entire tree.

Returns: void

Description: Free an expression tree node (ThisNodeOnly TRUE) or its entire tree.
See also: MvarExprTreeFree,

8.2.250 MvarExprTreeFromCrv (mvarextr.c:47)

```
MvarExprTreeStruct *MvarExprTreeFromCrv(const CagdCrvStruct *Crv,
                                        int NewDim,
                                        int StartAxis)
```

Crv: TO convert into an expression tree of NewDim dimension.

NewDim: The new multivariate dimension of this tree, to promote the curve from.

StartAxis: The starting axis (dimension) of the directions of Crv.

Returns: The build multivariate expression tree.

Description: Converts the input curve into a multivariate expression tree in newDim dimension so that the Crv is along the StartAxis dimension.
See also:

8.2.251 MvarExprTreeFromMV (mvarextr.c:112)

```
MvarExprTreeStruct *MvarExprTreeFromMV(const MvarMVStruct *MV,
                                       int NewDim,
                                       int StartAxis)
```

MV: To convert into an expression tree of NewDim dimension.

NewDim: The new multivariate dimension of this tree, to promote the surface from.

StartAxis: The starting axis (dimension) of the directions of MV.

Returns: The build multivariate expression tree.

Description: Converts the input multivariate into a multivariate expression tree in newDim dimension so that the MV is along the StartAxis dimension (and beyond).
See also: MvarExprTreeFromMV2,

8.2.252 MvarExprTreeFromMV2 (mvarextr.c:142)

```
MvarExprTreeStruct *MvarExprTreeFromMV2(const MvarMVStruct *MV)
```

MV: To convert into an expression tree of NewDim dimension.

Returns: The build multivariate expression tree.

Description: Converts the input multivariate into a multivariate expression tree in newDim dimension so that the MV is along the StartAxis dimension (and beyond).
See also: MvarExprTreeFromMV,

8.2.253 MvarExprTreeFromSrf (mvarextr.c:79)

```
MvarExprTreeStruct *MvarExprTreeFromSrf(const CagdSrfStruct *Srf,
                                        int NewDim,
                                        int StartAxis)
```

Srf: TO convert into an expression tree of NewDim dimension.

NewDim: The new multivariate dimension of this tree, to promote the surface from.

StartAxis: The starting axis (dimension) of the directions of Srf.

Returns: The build multivariate expression tree.

Description: Converts the input surface into a multivariate expression tree in newDim dimension so that the Srf is along the StartAxis dimension (and beyond).
See also:

8.2.254 MvarExprTreeGradient (mvarextr.c:2479)

```
CagdRType *MvarExprTreeGradient(const MvarExprTreeStruct *ET,
                                CagdRType *Params,
                                int *Dim,
                                CagdRType *GradData,
                                CagdRType *V)
```

ET: Expression tree to evaluate its gradient.

Params: Parameter values to evaluate the gradient of the expression.

Dim: Will be set with the dimension of the gradient.

GradData: Storage for gradient evaluation

V: Place to save auxiliary evaluated result. Of size MVAR_MAX_PT_COORD^2. If the returned gradient vector is of dimension n, it will be saved in entries [0] to [n-1].

Returns: Evaluation result. Same as V.

Description: Evaluate the gradient of the expression tree at the given parametric location.
See also: MvarExprTreeEval,

8.2.255 MvarExprTreeInteriorKnots (mvarextr.c:2209)

```
int MvarExprTreeInteriorKnots(const MvarExprTreeStruct *ET, CagdRType *Knot)
```

ET: Expression tree to examine for interior knots.

Knot: Interior knot value if non Bezier expression tree (valid only if a non negative value is returned).

Returns: Negative if all MVs are Bezier, index to dimension with interior knots, from which Knot is extracted.

Description: Tests if all MVs in expression tree ET are Bezier (no interior knots).
See also:

8.2.256 MvarExprTreeIntrnlNew (mvarextr.c:337)

```
MvarExprTreeStruct *MvarExprTreeIntrnlNew(MvarExprTreeNodeType NodeType,
                                          MvarExprTreeStruct *Left,
                                          MvarExprTreeStruct *Right,
                                          const MvarBBoxStruct *MVBBox)
```

NodeType: Type of internal node, addition, multiplication, etc.

Left, Right: The left and right sons of this node.

MVBBox: Bounding box, if any, of this node.

Returns: The expression tree constructed representation.

Description: An expression tree (ET) constructor for a tree internal node.
See also:

8.2.257 MvarExprTreeLeafDomain (mvarextr.c:1701)

```
int MvarExprTreeLeafDomain(MvarExprTreeStruct *ET,
                           CagdRType *Min,
                           CagdRType *Max,
                           int Axis)
```

ET: Expression tree to derive the domain of, in direction Axis.

Min, Max: Domain of ET in direction Axis.

Axis: The direction along with to compute the domain of ET.

Returns: TRUE for successful computation, FALSE otherwise.

Description: Computes the domain of the expression tree variable (leaf) along direction Axis. The assumption is that any MV in ET with a non trivial degree in direction Axis possess the same domain.
See also:

8.2.258 MvarExprTreeLeafNew (mvarextr.c:277)

```
MvarExprTreeStruct *MvarExprTreeLeafNew(CagdBType IsRef,
                                        MvarMVStruct *MV,
                                        int NewDim,
                                        int StartAxis,
                                        MvarNormalConeStruct *MVBCone,
                                        const MvarBBoxStruct *MVBBox)
```

IsRef: TRUE to just reference MV instead of copying it. Note that if NewDim is non zero, the MV will always be copied with the new expanded dimension NewDim.

MV: MV to occupy this leaf node. Can be NULL.

NewDim: The new multivariate dimension of this tree.

StartAxis: The starting axis of the directions of MV.

MVBCone: Bounding code, if any, of MV.

MVBBox: Bounding box, if any, of MV.

Returns: An expression tree representing MV.

Description: An expression tree (ET) constructor for a tree leaf node.
See also:

8.2.259 MvarExprTreeLog (mvarextr.c:1010)

```
MvarExprTreeStruct *MvarExprTreeLog(MvarExprTreeStruct *Left)
```

Left: The expression tree to take log of. The sub tree is used in place, so duplicate before calling this function if you like to keep the original Left.

Returns: Logarithm'ed expression tree, "Log(Left)".

Description: Logarithm of subexpression tree into a new expression tree.
See also:

8.2.260 MvarExprTreeMergeScalar (mvarextr.c:842)

```
MvarExprTreeStruct *MvarExprTreeMergeScalar(MvarExprTreeStruct *Left,
                                            MvarExprTreeStruct *Right)
```

Left, Right: The two expression trees to merge. These sub trees are used in place, so duplicate before calling this function if you like to keep the original Left/Right.

Returns: Multiplied expression tree, "Left * Right".

Description: Concatenates two scalar sub expression tree into one new expression tree. Left and Right are assumed to be scalar ET function.
See also:

8.2.261 MvarExprTreeMult (mvarextr.c:807)

```
MvarExprTreeStruct *MvarExprTreeMult(MvarExprTreeStruct *Left,
                                     MvarExprTreeStruct *Right)
```

Left, Right: The two expression trees to multiply. These sub trees are used in place, so duplicate before calling this function if you like to keep the original Left/Right.

Returns: Multiplied expression tree, "Left * Right".

Description: Multiplies two sub expression tree into one new expression tree.
See also:

8.2.262 MvarExprTreeMultScalar (mvarextr.c:878)

```
MvarExprTreeStruct *MvarExprTreeMultScalar(MvarExprTreeStruct *Left,
                                           MvarExprTreeStruct *Right)
```

Left, Right: The two expression trees to multiply. These sub trees are used in place, so duplicate before calling this function if you like to keep the original Left/Right.

Returns: Multiplied expression tree, "Left * Right".

Description: Multiplies two sub expression tree into one new expression tree. Right is assumed to be a scalar ET function.
See also:

8.2.263 MvarExprTreeNPow (mvarextr.c:1128)

```
MvarExprTreeStruct *MvarExprTreeNPow(MvarExprTreeStruct *Left, int Power)
```

Left: The expression tree to to raise to power. The sub tree is used in place, so duplicate before calling this function if you like to keep the original Left.

Power: The power to raise the expression to.

Returns: Square expression tree, "Left^power".

Description: Subexpression tree to a power of Power.
See also:

8.2.264 MvarExprTreeNormalConeMul (mvcones.c:1985)

```
MvarNormalConeStruct *MvarExprTreeNormalConeMul(
                                        const MvarNormalConeStruct *ConeF,
                                        const MvarNormalConeStruct *ConeG,
                                        const MvarBBoxStruct *BBoxF,
                                        const MvarBBoxStruct *BBoxG,
                                        int Dim)
```

ConeF: Normal cone of the first term.

ConeG: Normal cone of the second term.

BBoxF: Bounding box of the first term.

BBoxG: Bounding box of the second term.

Dim: Dimensions.

Returns: The resulting normal cone.

Description: Constructs a normal cone of a multiplication.
See also: MVarExprTreeNormalCone, MvarExprTreeNormalConeSum, MvarExprTreeNormalConeScale,

8.2.265 MvarExprTreeNormalConeScale (mvcones.c:2028)

```
MvarNormalConeStruct *MvarExprTreeNormalConeScale(
                                          const MvarNormalConeStruct *ConeF,
                                          const MvarBBoxStruct *BBoxGPrime,
                                          int Dim)
```

ConeF: Normal cone of the function.

BBoxGPrime: Bounding box of the scale factor.

Dim: Dimensions.

Returns: New scaled cone.

Description: Scales normal cone (for composition and multiplication).
See also: MVarExprTreeNormalCone, MvarExprTreeNormalConeMul,

8.2.266 MvarExprTreeNormalConeSub (mvcones.c:1946)

```
MvarNormalConeStruct *MvarExprTreeNormalConeSub(
                                        const MvarNormalConeStruct *ConeF,
                                        const MvarNormalConeStruct *ConeG,
                                        int Dim)
```

ConeF: Normal cone of the minuend.

ConeG: Normal cone of the subtrahend.

Dim: Dimensions.

Returns: The resulting normal cone.

Description: Constructs a normal cone of a difference.
See also: MVarExprTreeNormalCone, MvarExprTreeNormalConeSum,

8.2.267 MvarExprTreeNormalConeSum (mvcones.c:1758)

```
MvarNormalConeStruct *MvarExprTreeNormalConeSum(
                                           const MvarNormalConeStruct *ConeF,
                                           const MvarNormalConeStruct *ConeG,
                                           int Dim)
```

ConeF: Normal cone of the the first summand.

ConeG: Normal cone of the second summand.

Dim: Dimensions.

Returns: The resulting normal cone.

Description: Constructs a normal cone of a sum.
See also: MVarExprTreeNormalCone, HyperplaneOrthoSystem,

8.2.268 MvarExprTreePrintInfo (mvarextr.c:1928)

```
void MvarExprTreePrintInfo(const MvarExprTreeStruct *ET,
                           CagdBType CommonExprIdx,
                           CagdBType PrintMVInfo,
                           MvarExprTreePrintFuncType PrintFunc)
```

ET: To traverse and prints its Info/Node content.

CommonExprIdx: TRUE to only dump the common expression index, FALSE for the full common expression in place.

PrintMVInfo: TRUE to print information on MV leafs.

PrintFunc: Call back function to print a string.

Returns: void

Description: Traverses the ET in infix order and print the Info/Node in the ET.
See also:

8.2.269 MvarExprTreeRecip (mvarextr.c:1159)

```
MvarExprTreeStruct *MvarExprTreeRecip(MvarExprTreeStruct *Left)
```

Left: The expression tree to reciprocate. The sub tree is used in place, so duplicate before calling this function if you like to keep the original Left.

Returns: Reciprocation expression tree, "1.0/Left".

Description: Reciprocation of subexpression tree into a new expression tree.
See also:

8.2.270 MvarExprTreeSize (mvarextr.c:580)

```
int MvarExprTreeSize(MvarExprTreeStruct *ET)
```

ET: Expression tree to compute its size (number of nodes).

Returns: Number of nodes in ET.

Description: Returns the size (number of nodes) an expression tree has.
See also:

8.2.271 MvarExprTreeSqr (mvarextr.c:1097)

```
MvarExprTreeStruct *MvarExprTreeSqr(MvarExprTreeStruct *Left)
```

Left: The expression tree to take square of. The sub tree is used in place, so duplicate before calling this function if you like to keep the original Left.

Returns: Square expression tree, "Square(Left)".

Description: Square of subexpression tree into a new expression tree.
See also:

8.2.272 MvarExprTreeSqrt (mvarextr.c:1068)

```
MvarExprTreeStruct *MvarExprTreeSqrt(MvarExprTreeStruct *Left)
```

Left: The expression tree to take square root of. The sub tree is used in place, so duplicate before calling this function if you like to keep the original Left.

Returns: Sqrt expression tree, "Sqrt(Left)".

Description: Square root of subexpression tree into a new expression tree.
See also:

8.2.273 MvarExprTreeSub (mvarextr.c:773)

```
MvarExprTreeStruct *MvarExprTreeSub(MvarExprTreeStruct *Left,
                                    MvarExprTreeStruct *Right)
```

Left, Right: The two expression trees to subtract. These sub trees are used in place, so duplicate before calling this function if you like to keep the original Left/Right.

Returns: Subtracted expression tree, "Left - Right".

Description: Subtracts two sub expression tree into one new expression tree.
See also:

8.2.274 MvarExprTreeSubdivAtParam (mvarextr.c:1194)

```
int MvarExprTreeSubdivAtParam(const MvarExprTreeStruct *ET,
                              CagdRType t,
                              MvarMVDirType Dir,
                              MvarExprTreeStruct **Left,
                              MvarExprTreeStruct **Right)
```

ET: The expression tree to subdivide at parameter value t, in Dir.

t: The parameter value to subdivide at.

Dir: The direction of subdivision.

Left: First result of subdivision.

Right: Second result of subdivision.

Returns: TRUE if successful, FALSE otherwise.

Description: Subdivides the given multivariate expression tree at t in direction Dir.
See also:

8.2.275 MvarExprTreeToMV (mvarextr.c:164)

```
MvarMVStruct *MvarExprTreeToMV(const MvarExprTreeStruct *ET)
```

ET: Expression tree to convert to a multivariate.

Returns: Multivariate representing the expression tree.

Description: Converts an expression tree to a regular multivariate function.
See also:

8.2.276 MvarExprTreeZerosCnvrtBezier2MVs (zret0d.c:1162)

```
int MvarExprTreeZerosCnvrtBezier2MVs(int Bezier2MVs)
```

Bezier2MVs: TRUE to convert to MVs, FALSE to subdivide Bezier ETs.

Returns: Old setting for Bezier conversion setting.

Description: Sets the way expression trees of Bezier MVs are treated. If TRUE, the ETs are converted into MVs and the regular MV zero solver is invoked. If FALSE, the ETs are subdivided all the way to SUbdivTol.
See also: MvarExprTreesZeros, MvarExprTreeZerosUseCommonExpr,

8.2.277 MvarExprTreeZerosUseCommonExpr (zret0d.c:1132)

```
int MvarExprTreeZerosUseCommonExpr(int UseCommonExpr)
```

UseCommonExpr: TRUE to use common expressions, FALSE otherwise.

Returns: Old setting of common expressions' use.

Description: Sets the exploitation of common expression extraction in expression trees. If TRUE, the ETs are scanned for common expressions that are then processed once only, during the subdivision process.
See also: MvarExprTreesZeros, MvarExprTreeZerosCnvrtBezier2MVs,

8.2.278 MvarExprTreesSame (mvarextr.c:637)

```
CagdBType MvarExprTreesSame(const MvarExprTreeStruct *ET1,
                            const MvarExprTreeStruct *ET2,
                            CagdRType Eps)
```

ET1, ET2: Two expression trees to compare.

Eps: Tolerance of approximation.

Returns: TRUE if the same ETs (up to Eps), FALSE otherwise.

Description: A comparison function to test for similarity (up to Eps) two expression trees.
See also:

8.2.279 MvarExprTreesVerifyDomain (mvarextr.c:698)

```
int MvarExprTreesVerifyDomain(MvarExprTreeStruct *ET1,
                              MvarExprTreeStruct *ET2)
```

ET1, ET2: Two multivariate expression trees to verify that they share the same domains.

Returns: TRUE if Et1/2 shares the same domains, FALSE otherwise.

Description: Examine the given two multivariate's expression trees if they share the same domains.
See also: MvarExprTreeDomain,

8.2.280 MvarFindSrfMiterPoints (selfintr.c:1977)

Surface self-intersection
Miter points
singularities
Jacobian

```
MvarPtStruct *MvarFindSrfMiterPoints(const CagdSrfStruct *Srf,
                                     int Axis,
                                     CagdRType SubdivTol,
                                     CagdRType NumericTol,
                                     CagdBType Euclidean)
```

Srf: Surface to compute miter points.

Axis: System to compute the miter points. If Axis = 1, solve Nx(u, v) = Ny(u, v) = 0. If Axis = 2, solve Ny(u, v) = Nz(u, v) = 0. If Axis = 3, solve Nz(u, v) = Nx(u, v) = 0. If Axis = 0, find the miter points satisfying all three systems.

SubdivTol: Subdivion tolerance used in the solver.

NumericTol: Numeric tolerance used in the solver.

Euclidean: If FALSE, find the UV coordinates of miter points. Otherwise, compute the XYZ coordinates.

Returns: A list of miter points satisfying the constraint equations.

Description: Given a surface S(u,v), compute the singular or miter points, the points on the self-intersection curves of the surface that changes the normal direction abruptly. Specifically, it finds location where the normal (or Jacobian) is vanishing as the solutions of one of three systems of equations, using the multivariate solvers -

```
Nx(u, v) = Ny(u, v) = 0
```

or

```
Ny(u, v) = Nz(u, v) = 0
```

or

```
Nz(u, v) = Nx(u, v) = 0
```

where Ni(u, v) is i-th coordinate (x, y or z) of the unnormalized normal of the surface. Miter points satisfy all the three systems simultaneously.

See also: MvarFindSrfMiterPointsPartial, SymbSrfNormalSrf,

8.2.281 MvarFindSrfMiterPointsPartial (selfintr.c:1848)

Surface self-intersection
Miter points
singularities
Jacobian

```
MvarPtStruct *MvarFindSrfMiterPointsPartial(const CagdSrfStruct *Srf,
                                            const CagdSrfStruct *NrmlSrf,
                                            int Axis1,
                                            int Axis2,
                                            CagdRType SubdivTol,
                                            CagdRType NumericTol,
                                            CagdRType UVDiffTol,
                                            CagdBType Euclidean)
```

Srf: Surface to compute miter points.

NrmlSrf: Normal surface. If NULL, then compute the surface inside this routine.

Axis1: First axis of normal equation in the system (1-3).

Axis2: Second axis of normal equation in the system (1-3).

SubdivTol: Subdivion tolerance used in the solver.

NumericTol: Numeric tolerance used in the solver.

UVDiffTol: If positive, merge the solution points within the tolerance.

Euclidean: If FALSE, find the UV coordinates of miter points. Otherwise, compute the XYZ coordinates.

Returns: A list of miter points satisfying the constraint equations.

Description: Given a surface S(u,v), compute the singular or miter points, the points on the self-intersection curves of the surface that changes the normal direction abruptly. Specifically, it finds location where the normal (or Jacobian) is vanishing as the solutions of one of three systems of equations, using the multivariate solvers:

```
Nx(u, v) = Ny(u, v) = 0
```

or

```
Ny(u, v) = Nz(u, v) = 0
```

or

```
Nz(u, v) = Nx(u, v) = 0
```

where Ni(u, v) is i-th coordinate (x, y or z) of the unnormalized normal of the surface. Miter points satisfy all the three systems simultaneously.

See also: MVFindMiterPoints, SymbSrfNormalSrf,

8.2.282 MvarFlecnodalCrvsCreateMVCnstrnts (mvaccess.c:480)

```
MvarMVStruct **MvarFlecnodalCrvsCreateMVCnstrnts(const CagdSrfStruct *CSrf)
```

CSrf: To compute the (MV constraints of the) flecnodals for.

Returns: The set of 3 constraints in 4 unknowns to derive the flecnodal curves of CSrf.

Description: Computes the MV constraints of flecnodal curves on surface S(u, v). Denote by Suv the second order derivatives of S(u, v) and by Suvv the third order derivatives. Then, solution of the following 3 equations in 4 unknowns (u, v, a, b) provides the flecnodal curves:

```
              [ L M ] [ a ]
1. [ a  b ] [     ] [   ] =       (This location/direction is asymp.)
              [ M N ] [ b ]        (L, M, N are the coef. of SFF.)
```

```
                [ Suu Suv ]           [ a ]
   = [ a  b ] [         ] . n(u, v) [   ] = 0,
                [ Suv Svv ]           [ b ]
```

or

```
   < a^2 Suu + 2ab Suv + b^2 Svv, n(u, v) > = 0.
```

```
              (   [ Suuu Suuv ]       [ Svuu Svuv ] )            [ a ]
2. [ a  b ] ( a [             ]  +  b [           ] )  . n(u, v) [   ] = 0,
              (   [ Suuv Suvv ]       [ Svuv Svvv ] )            [ b ]
```

(A third derivative zero contact.)
or

```
   < a^3 Suuu + 3a^2b Suuv + 3ab^2 Suvv + b^3 Svvv, n(u, v) > = 0.
```

3. a * a + b * b = 1 (A normalization constraint.)

See also: SymbSrfGaussCurvature, UserSrfTopoAspectGraph, SymbEvalSrfAsympDir, , MvarSrfSilhInflections, MvarSrfFlecnodalCrvs,

8.2.283 MvarGetLastPt (mvar_pll.c:270)

```
MvarPtStruct *MvarGetLastPt(MvarPtStruct *Pts)
```

Pts: List of points to fetch last one.

Returns: Last element in list.

Description: Returns the last element in the list.

8.2.284 MvarGetMiniumIntnPar (mvtrmpcr.c:1543)

```
static CagdPtStruct *MvarGetMiniumIntnPar(CagdCrvStruct *TrimmedBis,
                                          CagdRType *Pt,
                                          CagdPtStruct *Inter,
                                          CagdPType LeftNormal)
```

TrimmedBis: The input curve in CagdCrvStruct.

Pt: The input point - CagdRType.

Inter: List of intersection parameters obtained using SymbLclDistCrvLine.

LeftNormal: Left Normal of TrimmedBis at the point Pt.

Returns: Return the TrimmedBis parameter.

Description: This function returns the minimum intersection parameter of the curve with the Line after confirming that the vector between the discontinuity point and the intersection parameter is in the same direction to the Left hand normal.
See also: SymbLclDistCrvLine,

8.2.285 MvarGetPointsMeshIndices (mvar_aux.c:1099)

```
int MvarGetPointsMeshIndices(const MvarMVStruct *MV, int *Indices)
```

MV: Whose indices are for.

Indices: To compute the exact point location in MV -> Points

Returns: Index of point whose indices are Indices in MV -> Points.

Description: Given indices into the control mesh, return the index in the vector representation Points of that single point.
See also: MvarMeshIndicesFromIndex, MvarGetPointsPeriodicMeshIndices,

8.2.286 MvarGetPointsPeriodicMeshIndices (mvar_aux.c:1151)

```
int MvarGetPointsPeriodicMeshIndices(const MvarMVStruct *MV, int *Indices)
```

MV: Whose indices are for.

Indices: To compute the exact point location in MV -> Points

Returns: Index of point whose indices are Indices in MV -> Points.

Description: Given indices into the control mesh, return the index in the vector representation Points of that single point. MV can be periodic.
See also: MvarMeshIndicesFromIndex, MvarGetPointsMeshIndices,

8.2.287 MvarGetSrfsCommonCrvs (mvtrivar.c:783)

```
int MvarGetSrfsCommonCrvs(const CagdSrfStruct *Srf1,
                          const CagdSrfStruct *Srf2,
                          CagdCrvStruct **CommonCrv)
```

Srf1, Srf2: The two input surfaces.

CommonCrv: A pointer of curve to return the common boundary, if any.

Returns: The returned number parsed according to this template - CagdSrfBndryType (According to Srf1) + CagdSrfBndryType (According to Srf2) * 0x01000. Otherwise -1.

Description: The function takes two surfaces and extract the common boundary curve of the two surfaces, if any, and makes a copy of it into CommonCrv.
See also: CagdGetCrvsCommonPt, MvarTrivarOneSidedBoolSum3Srfs,

8.2.288 MvarHFDistAntipodalCrvCrvC1 (hasdrf2d.c:156)

```
MvarPtStruct *MvarHFDistAntipodalCrvCrvC1(const CagdCrvStruct *Crv1,
                                          const CagdCrvStruct *Crv2,
                                          CagdRType Epsilon)
```

Crv1, Crv2: The two curves to solve for their antipodal locations.

Epsilon: Tolerance of computation.

Returns: List of pairs of parameters in the t & r coefficients.

Description: Computes the antipodal points of the given two curves by solving:

```
< C1(t) - C2(r), C1'(t) > = 0,
< C1(t) - C2(r), C2'(r) > = 0.
< C1(t) - C2O(r), C1'(t) > = 0,       (Only during subdivision step)
< C1(t) - C2O(r), C2'(r) > = 0,       (Only during subdivision step)
```

where C2O is an offset curve of C2 by amount larger than subdivison tol. The last two equations are used to purge intersection locations as they cause the first two equations to vanish.

See also: MvarCrvAntipodalPoints, MvarCntctTangentialCrvCrvC1,

8.2.289 MvarHFDistAntipodalCrvSrfC1 (hasdrf3d.c:139)

```
MvarPtStruct *MvarHFDistAntipodalCrvSrfC1(const CagdSrfStruct *Srf1,
                                          const CagdCrvStruct *Crv2)
```

Srf1, Crv2: A surface and a curve to solve for their antipodal locations.

Returns: List of pairs of parameters in the uv & t coefficients.

Description: Computes the antipodal points of the given surface and curve by solving:

```
< C2(t) - S1(u, v), dS1(u, v) / du > = 0,
< C2(t) - S1(u, v), dS1(u, v) / dv > = 0,
< C2(t) - S1(u, v), dC2(t) / dt > = 0.
```

See also: MvarSrfAntipodalPoints,

8.2.290 MvarHFDistAntipodalSrfSrfC1 (hasdrf3d.c:404)

```
MvarPtStruct *MvarHFDistAntipodalSrfSrfC1(const CagdSrfStruct *Srf1,
                                          const CagdSrfStruct *Srf2)
```

Srf1, Srf2: The two surfaces to solve for their antipodal locations.

Returns: List of pairs of parameters in the uv & st coefficients.

Description: Computes the antipodal points of the given two surfaces by solving:

```
< S2(r, t) - S1(u, v), dS1(u, v) / du > = 0,
< S2(r, t) - S1(u, v), dS1(u, v) / dv > = 0,
< S2(r, t) - S1(u, v), dS2(s, t) / ds > = 0,
< S2(r, t) - S1(u, v), dS2(s, t) / ds > = 0.
```

See also: ,

8.2.291 MvarHFDistBisectSrfSrfC1 (hasdrf3d.c:582)

```
MvarHFDistPairParamStruct *MvarHFDistBisectSrfSrfC1(const CagdSrfStruct *Srf1,
                                                    const CagdSrfStruct *Srf2)
```

Srf1: First surface to intersect its self-bisector with Srf2.

Srf2: Second surface to intersect against the self bisector of Srf1.

Returns: Linked list of all detected intersections. Note each detected intersection holds two parameter locations of Srf1.

Description: Compute the intersection locations of (C^1 cont) Srf2 with the self bisectors of (C^1 cont.) Srf1, if any. The solution is computed by solving the following set of contraints:

```
|| S2(s, t) - S1(a, b) ||^2 == || S2(s, t) - S1(c, d) ||^2,
< S2((s, t) - S1(a, b), dS1(a, b)/da > = 0,
< S2((s, t) - S1(a, b), dS1(a, b)/db > = 0,
< S2((s, t) - S1(c, d), dS1(c, d)/dc > = 0.
< S2((s, t) - S1(c, d), dS1(c, d)/dd > = 0.
```

augumented with

```
< (S1(a, b) - S(c, d)) x (S1(a, b) + S1(c, d) - 2S2(s, t)),
  NS2(s, t) > = 0,
```

where NS2 is a (non normalized) normnal field of S2.
The first equation above (equal distance to two different locations in C1) could be rewritten as:

```
< S1(a, b) + S1(c, d) - 2S2(s, t), S1(a, b) - S1(c, d) > = 0,
```

which hints to the fact that this equation vanish for ((a, b) == (c, d)). Hence, in the solution process, we eliminate the ((a, b) == (c, d)) factors from it.

See also:

8.2.292 MvarHFDistCrvCrvC1 (hasdrf2d.c:1052)

```
CagdRType MvarHFDistCrvCrvC1(const CagdCrvStruct *Crv1,
                             const CagdCrvStruct *Crv2,
                             MvarHFDistParamStruct *Param1,
                             MvarHFDistParamStruct *Param2,
                             CagdRType Epsilon)
```

Crv1: First Crv to measure its hausdorff distance to Crv2.

Crv2: Second Crv to measure its hausdorff distance to Crv1.

Param1: Where to return the parameter value(s) of Crv1 with the maximal Hausdorff distance. Can be more than one location!

Param2: Where to return the parameter value(s) of Crv2 with the maximal Hausdorff distance. Can be more than one location!

Epsilon: Tolerance of computation.

Returns: The Hausdorff distance.

Description: Computes the Hausdorff distance between two C^1 cont. curves, C1 and C2. The curves are assumed to be either in R2 or R3. The extreme distance between two curves could happen at: + The end points of the curves. + Antipodal locations where:

```
C1'(t) <C1(t) - C2(r)> = 0,
C2'(r) <C1(t) - C2(r)> = 0.
```

+ Locations where C1 crosses the self bisector of C2 (or vice versa): Let Bi(x,y) = 0 be self bisector of Ci(t), and Cj(r) = (xj(r), yj(r)). Then, solve for:

```
|| C2(r) - C1(s) ||^2 == || C2(r) - C1(t) ||^2,
< C2(r) - C1(s), C1'(s) > = 0,
< C2(r) - C1(t), C1'(t) > = 0.
```

All the above equations hold for R2 and for R3.

See also:

8.2.293 MvarHFDistFromCrvToCrvC1 (hasdrf2d.c:865)

```
CagdRType MvarHFDistFromCrvToCrvC1(const CagdCrvStruct *CCrv1,
                                   const CagdCrvStruct *CCrv2,
                                   MvarHFDistParamStruct *Param1,
                                   MvarHFDistParamStruct *Param2,
                                   CagdRType Epsilon)
```

CCrv1: First Crv to measure its hausdorff distance to CCrv2.

CCrv2: Second Crv to measure its hausdorff distance from CCrv1.

Param1: Where to return the parameter value(s) of Crv1 with the maximal Hausdorff distance. Can be more than one location!

Param2: Where to return the parameter value(s) of Crv2 with the maximal Hausdorff distance. Can be more than one location!

Epsilon: Tolerance of computation.

Returns: The Hausdorff distance.

Description: Computes the one sided Hausdorff distance between two C^1 cont. curves, from C1 to C2. The curves are assumed to be either in R2 or R3. The one sided extreme distance between two curves could happen at: + The end points of curve C1 or curve C2. + Antipodal locations or locations where:

```
C1'(t) <C1(t) - C2(r)> = 0,
C2'(r) <C1(t) - C2(r)> = 0,
```

that are also global distance minima from C1(t) to any point on C2. + Locations where C1 crosses the self bisector of C2 that are also local distance minima from C1 to any point on C2.
All the above equations hold for R2 and for R3.
See also:

8.2.294 MvarHFDistFromCrvToSrfC1 (hasdrf3d.c:293)

```
CagdRType MvarHFDistFromCrvToSrfC1(const CagdCrvStruct *Crv1,
                                   const CagdSrfStruct *Srf2,
                                   MvarHFDistParamStruct *Param1,
                                   MvarHFDistParamStruct *Param2)
```

Crv1: First crv to measure its hausdorff distance to Srf2.

Srf2: Second srf to measure its hausdorff distance from Crv1.

Param1: Where to return the parameter value(s) of Crv1 with the maximal Hausdorff distance. Can be more than one location!

Param2: Where to return the parameter value(s) of Srf2 with the maximal Hausdorff distance. Can be more than one location!

Returns: The Hausdorff distance.

Description: Computes the one sided Hausdorff distance between a C^1 cont. curve and a C^1 cont. surface, from C1 to S2. The shapes are assumed to be in R3 and non intersecting. The one sided extreme distance between the two shapes could happen at: + The corner/end points of curve C1 or surface S2. + Antipodal locations between the two shapes. + Locations where C1 crosses the self bisector of S2 that are also local distance minima from C1 to any point on S2.
See also: MvarHFDistFromSrfToSrfC1, MvarHFDistFromSrfToCrvC1,

8.2.295 MvarHFDistFromSrfToCrvC1 (hasdrf3d.c:331)

```
CagdRType MvarHFDistFromSrfToCrvC1(const CagdSrfStruct *Srf1,
                                   const CagdCrvStruct *Crv2,
                                   MvarHFDistParamStruct *Param1,
                                   MvarHFDistParamStruct *Param2)
```

Srf1: First srf to measure its hausdorff distance to Crv2.

Crv2: Second crv to measure its hausdorff distance from Srf1.

Param1: Where to return the parameter value(s) of Srf1 with the maximal Hausdorff distance. Can be more than one location!

Param2: Where to return the parameter value(s) of Crv2 with the maximal Hausdorff distance. Can be more than one location!

Returns: The Hausdorff distance.

Description: Computes the one sided Hausdorff distance between a C^1 cont. surface and a C^1 cont. curve, from S1 to C2 The shapes are assumed to be in R3 and non intersecting. The one sided extreme distance between the two shapes could happen at: + The corner/end points of surface S1 or curve C2. + Antipodal locations between the two shapes. + Locations where S1 crosses the self bisector of C2 that are also local distance minima from S1 to any point on C2.

See also: MvarHFDistFromSrfToSrfC1,

8.2.296 MvarHFDistFromSrfToSrfC1 (hasdrf3d.c:653)

```
CagdRType MvarHFDistFromSrfToSrfC1(const CagdSrfStruct *CSrf1,
                                   const CagdSrfStruct *CSrf2,
                                   MvarHFDistParamStruct *Param1,
                                   MvarHFDistParamStruct *Param2)
```

CSrf1: First Srf to measure its hausdorff distance to Srf2.

CSrf2: Second Srf to measure its hausdorff distance from Srf1.

Param1: Where to return the parameter value(s) of Srf1 with the maximal Hausdorff distance. Can be more than one location!

Param2: Where to return the parameter value(s) of Srf2 with the maximal Hausdorff distance. Can be more than one location!

Returns: The Hausdorff distance.

Description: Computes the one sided Hausdorff distance between two C^1 cont. surfaces from S1 to S2. The surfaces are assumed to be in R3 and non intersecting. The one sided extreme distance between two surfaces could happen at: + The corner points of surface S1 or surface S2. + Antipodal locations between the two surfaces. + Locations where S1 crosses the self bisector of S2 that are also local distance minima from S1 to any point on S2.

See also:

8.2.297 MvarHFDistInterBisectCrvCrvC1 (hasdrf2d.c:598)

```
MvarHFDistPairParamStruct *MvarHFDistInterBisectCrvCrvC1(
                                                const CagdCrvStruct *CCrv1,
                                                const CagdCrvStruct *CCrv2,
                                                CagdRType Epsilon)
```

CCrv1: First curve to intersect its self-bisector with Crv2.

CCrv2: Second curve to intersect against the self bisector of Crv1.

Epsilon: Tolerance of computation.

Returns: Linked list of all detected intersections. Note each detected intersection holds two parameters of Crv1.

Description: Compute the intersection locations of (C^1 cont) Crv2 with the self bisectors of (C^1 cont.) Crv1, if any. The solution is computed by solving the following cases:

1. The curve-curve self bisector of Crv1, intersected with Crv2:

   ```
   || C2(w) - C1(u) ||^2 == || C2(w) - C1(v) ||^2,
   < C2(w) - C1(u), C1'(u) > = 0,
   < C2(w) - C1(v), C1'(v) > = 0.
   ```

   The first equations above (equal distance to two different locations in C1) could be rewritten as:

   ```
   < C1(u) + C1(v) - 2C2(w), C1(u) - C1(v) > = 0,
   ```

   which hints to the fact that this equation vanish for (u == v). Hence, in the solution process, we eliminate the (u - v) factors from it.
2. Endpoint-curve self bisectors of Crv1, intersected with Crv2 (2 cases): Let B(t) be equal to the self bisector of Crv1 with one of its end points. Then, solve for B(t) = C2(w).
3. Endpoint-Endpoint self bisectors of Crv1, intersected with Crv2: solved as a line (Endpoint-Endpoint bisector) - Crv2 intersection.

See also:

8.2.298 MvarHFDistInterBisectCrvCrvC1Crvtr (hasdrf2d.c:351)

```
MvarHFDistPairParamStruct *MvarHFDistInterBisectCrvCrvC1Crvtr(
                                                    CagdCrvStruct *Crv1,
                                                    CagdCrvStruct *Crv2,
                                                    CagdRType Epsilon)
```

Crv1: First curve to intersect its self-bisector with Crv2.

Crv2: Second curve to intersect against the self bisector of Crv1.

Epsilon: Tolerance of computation.

Returns: Linked list of all detected intersections. Note each detected intersection holds two parameters of Crv1.

Description: Old version of MvarHFDistInterBisectCrvCrvC1 that splits the input curves at curvature max. locations whereas MvarHFDistInterBisectCrvCrvC1 eliminates the (u - v) terms directly. Computes the intersection locations of (C^1 cont) Crv2 with the self bisectors of (C^1 cont.) Crv1, if any. The solution is computed by solving the following cases:

1. The curve-curve self bisector of Crv1, intersected with Crv2:

```
|| C2(w) - C1(u) ||^2 == || C2(w) - C1(v) ||^2,
< C2(w) - C1(u), C1'(u) > = 0,
< C2(w) - C1(v), C1'(v) > = 0.
```

   The first equations above (equal distance to two different locations in C1) could be rewritten as:

```
< C1(u) + C1(v) - 2C2(w), C1(u) - C1(v) > = 0,
```

   which hints to the fact that this equation vanish for (u == v). To speed up the process we also add a constraint that the distance from C2 to its two foot point should be greater than the radius of curvature of C1.

2. Endpoint-curve self bisectors of Crv1, intersected with Crv2 (2 cases): Let B(t) be equal to the self bisector of Crv1 with one of its end points. Then, solve for B(t) = C2(w).

3. Endpoint-Endpoint self bisectors of Crv1, intersected with Crv2: solved as a line (Endpoint-Endpoint bisector) - Crv2 intersection.

See also: MvarHFDistInterBisectCrvCrvC1,

8.2.299 MvarHFDistInterBisectSrfSrfC1 (hasdrf3d.c:615)

```
MvarHFDistPairParamStruct *MvarHFDistInterBisectSrfSrfC1(
                                                  const CagdSrfStruct *Srf1,
                                                  const CagdSrfStruct *Srf2)
```

Srf1: First curve to intersect its self-bisector with Srf2.

Srf2: Second curve to intersect against the self bisector of Srf1.

Returns: Linked list of all detected intersections. Note each detected intersection holds two parameters locations of Srf1.

Description: Compute the intersection locations of (C^1 cont) Srf2 with the self bisectors of (C^1 cont.) Srf1, if any. The solution is computed by solving the following set of contraints:

1. The surface-surface self bisector of Srf1, intersected with Srf2

2. boundary-curve self bisectors of Srf1, intersected with Srf2

3. corner-point self bisectors of Srf1, intersected with Srf2

See also:

8.2.300 MvarHFDistPointSrfC1 (hasdrf3d.c:43)

```
CagdRType MvarHFDistPointSrfC1(const CagdPType P,
                               const CagdSrfStruct *Srf,
                               MvarHFDistParamStruct *Param,
                               CagdBType MinDist)
```

P: Point to measure its Hausdorff distance to surface Srf.

Srf: Srf to measure its hausdorff distance to point Pt.

Param: Where to return the parameter value with the maximal distance.

MinDist: TRUE for minimal distance, FALSE for maximal.

Returns: The Hausdorff distance.

Description: Computes the Hausdorff distance between a point and a C^1 cont. surface. The surface and point are assumed to be either in R3. The extreme distance between a point and a surface could happen either at the corner points of surface Srf, the boundary curves of Srf, or when

```
<S(u, v) - P> dS'(u, v)/du = 0,
<S(u, v) - P> dS'(u, v)/dv = 0.
```

See also: MvarDistSrfPoint,

8.2.301 MvarHFDistSrfCrvC1 (hasdrf3d.c:364)

```
CagdRType MvarHFDistSrfCrvC1(const CagdSrfStruct *Srf1,
                             const CagdCrvStruct *Crv2,
                             MvarHFDistParamStruct *Param1,
                             MvarHFDistParamStruct *Param2)
```

Srf1: First srf to measure its hausdorff distance to Crv2.

Crv2: Second crv to measure its hausdorff distance to Srf1.

Param1: Where to return the parameter value(s) of Srf1 with the maximal Hausdorff distance. Can be more than one location!

Param2: Where to return the parameter value(s) of Crv2 with the maximal Hausdorff distance. Can be more than one location!

Returns: The Hausdorff distance.

Description: Computes Hausdorff distance between a C^1 cont. surface and a C^1 cont. curve, S1 and C2. The shapes are assumed to be in R3 and non intersecting.
See also: MvarHFDistSrfSrfC1,

8.2.302 MvarHFDistSrfSrfC1 (hasdrf3d.c:819)

```
CagdRType MvarHFDistSrfSrfC1(const CagdSrfStruct *Srf1,
                             const CagdSrfStruct *Srf2,
                             MvarHFDistParamStruct *Param1,
                             MvarHFDistParamStruct *Param2)
```

Srf1: First Srf to measure its hausdorff distance to Srf2.

Srf2: Second Srf to measure its hausdorff distance to Srf1.

Param1: Where to return the parameter value(s) of Srf1 with the maximal Hausdorff distance. Can be more than one location!

Param2: Where to return the parameter value(s) of Srf2 with the maximal Hausdorff distance. Can be more than one location!

Returns: The Hausdorff distance.

Description: Computes Hausdorff distance between two C^1 cont. surfaces, S1 and S2. The surfaces are assumed to be in R3 and non intersecting.
See also:

8.2.303 MvarHFExtremeLclDistPointCrvC1 (hasdrf2d.c:98)

```
CagdRType MvarHFExtremeLclDistPointCrvC1(CagdPType P,
                                         const CagdCrvStruct *Crv1,
                                         const CagdCrvStruct *Crv2,
                                         MvarHFDistParamStruct *Param2,
                                         CagdRType Epsilon)
```

P: Point on Crv1 to measure its extreme distance to curve Crv2.

Crv1: First curve that contains P.

Crv2: Second curve to measure extreme distance to from P.

Param2: Where to return the parameter value of Crv2 is returned.

Epsilon: Tolerance of computation.

Returns: The local extreme distance found, 0.0 if none.

Description: Computes the local extreme distance between a point P of Crv1 and Crv2. At the local extreme distance location on Crv2, denoted Q, verify that Q is closest to P than to any other location on Crv1. Returns maximal local extreme distance found, 0.0 if none.
See also: SymbLclDistCrvPoint,

8.2.304 MvarHFExtremeLclDistPointSrfC1 (hasdrf3d.c:86)

```
CagdRType MvarHFExtremeLclDistPointSrfC1(const CagdPType P,
                                         const CagdSrfStruct *Srf1,
                                         const CagdSrfStruct *Srf2,
                                         MvarHFDistParamStruct *Param2)
```

P: Point on Srf1 to measure its extreme distance to curve Srf2.

Srf1: First curve that contains P.

Srf2: Second curve to measure extreme distance to from P.

Param2: Where to return the parameter value of Srf2 is returned.

Returns: The local extreme distance found, 0.0 if none.

Description: Computes the local extreme distance between a point P of Srf1 and Srf2. At the local extreme distance location on Srf2, denoted Q, verify that Q is closest to P than to any other location on Srf1. Returns maximal local extreme distance found, 0.0 if none.
See also: MvarLclDistSrfPoint,

8.2.305 MvarImplicitCrvExtreme (mvartopo.c:34)

```
MvarPtStruct *MvarImplicitCrvExtreme(const CagdSrfStruct *Srf,
                                     CagdSrfDirType Dir,
                                     CagdRType SubdivTol,
                                     CagdRType NumerTol)
```

Srf: Implicit surface definition.

Dir: U to compute U-extreme values in Srf = 0, V for V-extreme.

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the surfaces.

NumerTol: Numeric tolerance of a possible numeric improvment stage. The numeric stage is employed if NumericTol < SubdivTol.

Returns: The computed extreme values, in the parametric space of surface Srf.

Description: Computes U or V extreme values of the zero set (implicit form) of Srf = 0.

8.2.306 MvarInverseCrvOnSrfProj (mvardist.c:930)

distance
inverse
projection

```
CagdCrvStruct *MvarInverseCrvOnSrfProj(const CagdCrvStruct *E3Crv,
                                       const CagdSrfStruct *Srf,
                                       void *SrfPrepHandle,
                                       CagdRType ApproxTol,
                                       const CagdRType *PrevUVPt,
                                       int *RedundantSol,
                                       int IncrSol)
```

E3Crv: E3 Curve to compute its inverse projection onto Srf. If none linear, curve is approximated as a piecewise linear curve first using ApproxTol.

Srf: The Surface E3Crv is on.

SrfPrepHandle: Hold pre-processed data by MvarInverseCrvOnSrfProjPrep to speed up the crv on srf projections. See MvarInverseCrvOnSrfProjPrep/MvarInverseCrvOnSrfProjFree.

ApproxTol: Tolerance of piecewise linear approximation if so needed.

PrevUVPt: Optional parameter of previous UV location from the end location of last UV crv. NULL to ignore.

RedundantSol: Will be set to true if solution was redundant - I.e. on a shared boundary seam. NULL to ignore.

IncrSol: TRUE to try and march incrementally over the curves. FALSE to solve a full solution for all back projected points.

Returns: The UV curve representing E3Crv on Srf.

Description: Compute the inverse projection of a Euclidean curve on surface to the UV values of the surface.
See also: MvarInverseCrvOnSrfProjPrep, MvarInverseCrvOnSrfProjFree, , MvarLclDistSrfPoint, MvarDistSrfPoint, MvarMVOrthoCrvProjOnSrf, , MvarProjCrvOnSrf,

8.2.307 MvarInverseCrvOnSrfProjFree (mvardist.c:839)

distance
inverse
projection

```
void MvarInverseCrvOnSrfProjFree(void *SrfPrepHandle)
```

SrfPrepHandle: Prep data struct by MvarInverseCrvOnSrfProjPrep to free.

Returns: void

Description: Free the prepared data structure by MvarInverseCrvOnSrfProjPrep.
See also: MvarInverseCrvOnSrfProj, MvarInverseCrvOnSrfProjPrep,

8.2.308 MvarInverseCrvOnSrfProjPrep (mvardist.c:810)

distance
inverse
projection

```
void *MvarInverseCrvOnSrfProjPrep(const CagdSrfStruct *Srf)
```

Srf: The Surface to prepare curve projection onto.

Returns: A structure holding the prepared data.

Description: Do the necessary preparations for the inverse projection of a Euclidean curve on surface to the UV values of the surface.
See also: MvarInverseCrvOnSrfProj, MvarInverseCrvOnSrfProjFree,

8.2.309 MvarIrit2DTrTo2DMVTrs (mvar_pll.c:1280)

```
MvarTriangleStruct *MvarIrit2DTrTo2DMVTrs(IPObjectStruct *ObjTrs)
```

ObjTrs: Irit polygons.

Returns: A list triangles.

Description: Converts a list of irit planar triangles (given as polygon objects) into a list of Mvar planar (2D) triangle structures.
See also: MvarCnvrtMVTrsToIritPolygons, MvarCnvrtMVPolysToIritPolys,

8.2.310 MvarKrnlBooleanSumTV (krnl_based_cnstrct.c:2430)

```
MvarKrnlTVResStruct *MvarKrnlBooleanSumTV(MvarKrnlTVInptStruct *Inpt)
```

Inpt: The input of the kernel based Boolean sum operator.

Returns: The result will be holded in a MvarKrnlTVResStruct struct.

Description: Constructs a Kernel Boolean sum trivariate using the six provided boundary surfaces. First the trivariate is constructed by the regular Boolean sum. Second, re-ordering the trivariate's inner control points into a unifrom grid. Third, each inner control point is translated by DistRaio * |KernelPt - Cp|, where kernelPt is the centroid of the kernel loci of the trivariate, and Cp is the location of the control point. Fourth, re-center the inner control points grid around the kernel center. The function aims to build a trivariate with a valid parameterization. If the function failed, it will add dofs to the input surfaces by degree raising or uniform knot insertion.
See also: MvarKrnlTVOneSidedBSum2Srfs, MvarKrnlTVOneSidedBSum3Srfs, , MvarKrnlRuledTV, MvarKrnlTV-InptStructAlloc, MvarKrnlTVInptStructFree,

8.2.311 MvarKrnlRuledTV (krnl_based_cnstrct.c:2544)

```
MvarKrnlTVResStruct *MvarKrnlRuledTV(MvarKrnlTVInptStruct *Inpt)
```

Inpt: The input of the kernel based ruling operator.

Returns: The result will be holded in a MvarKrnlTVResStruct struct.

Description: Constructs a Kernel ruled trivariate which formed between the two provided boundary surfaces. First the trivariate is constructed by the regular ruling operator. Second, re-ordering the trivariate's inner control points into a unifrom grid. Third, each inner control point translated by DistRaio * |KernelPt - Cp|, where kernelPt is the centroid of the kernel loci of the trivariate, and Cp is the location of the control point. Fourth,re-center the inner control points grid around the kernel center. The function aims to build a trivariate with a valid parameterization. If the function failed, it will add dofs to the input surfaces by degree raising or uniform knot insertion.
See also: MvarKrnlTVOneSidedBSum2Srfs, MvarKrnlTVOneSidedBSum3Srfs, , MvarKrnlBooleanSumTV, MvarKrnlTV-InptStructAlloc, , MvarKrnlTVInptStructFree,

8.2.312 MvarKrnlTVInptStructAlloc (krnl_based_cnstrct.c:2824)

```
MvarKrnlTVInptStruct *MvarKrnlTVInptStructAlloc()
```

Returns: Return's a pointer to the allocated object.

Description: Allocates and initialize a new object of MvarKrnlTVInptStruct.
See also: MvarKrnlTVOneSidedBSum2Srfs, MvarKrnlTVOneSidedBSum3Srfs, , MvarKrnlBooleanSumTV, MvarKrnl-RuledTV, MvarKrnlTVInptStructFree,

8.2.313 MvarKrnlTVInptStructFree (krnl_based_cnstrct.c:2853)

```
void MvarKrnlTVInptStructFree(MvarKrnlTVInptStruct *KrnlInpt)
```

KrnlInpt: A pointer to a MvarKrnlTVInptStruct object.

Returns: void

Description: Free from memory the given MvarKrnlTVInptStruct object and its all interior variables.
See also: MvarKrnlTVOneSidedBSum2Srfs, MvarKrnlTVOneSidedBSum3Srfs, , MvarKrnlBooleanSumTV, MvarKrnl-RuledTV, MvarKrnlTVInptStructAlloc,

8.2.314 MvarKrnlTVOneSidedBSum2Srfs (krnl_based_cnstrct.c:2639)

```
MvarKrnlTVResStruct *MvarKrnlTVOneSidedBSum2Srfs(MvarKrnlTVInptStruct *Inpt)
```

Inpt: The input of the kernel based one sided Boolean sum operator upon two adjacent surfaces.

Returns: The result will be holded in a MvarKrnlTVResStruct struct.

Description: Constructs a Kernel one sided Boolean sum trivariate using the two provided boundary surfaces. First the trivariate is constructed by the regular sided Boolean sum operator. Second, re-ordering the trivariate's inner control points into a unifrom grid. Third, each inner control point is translated by DistRaio * |KernelPt - Cp|, where kernelPt is the centroid of the kernel loci of the trivariate, and Cp is the location of the control point. Fourth, re-center the inner control points grid around the kernel center. The function aims to build a trivariate with a valid parameterization. If the function failed, it will add dofs to the input surfaces by degree raising or uniform knot insertion.
See also: MvarKrnlBooleanSumTV, MvarKrnlTVOneSidedBSum3Srfs, , MvarKrnlRuledTV, MvarKrnlTVInptStructAlloc, MvarKrnlTVInptStructFree,

8.2.315 MvarKrnlTVOneSidedBSum3Srfs (krnl_based_cnstrct.c:2737)

```
MvarKrnlTVResStruct *MvarKrnlTVOneSidedBSum3Srfs(MvarKrnlTVInptStruct *Inpt)
```

Inpt: The input of the kernel based one sided Boolean sum operator upon three adjacent surfaces.

Returns: The result will be holded in a MvarKrnlTVResStruct struct.

Description: Constructs a Kernel sided Boolean sum trivariate using the three provided boundary surfaces. First the trivariate is constructed by the regular sided Boolean sum operator. Second, re-ordering the trivariate's inner control points into a unifrom grid. Third, each inner control point is translated by DistRaio * |KernelPt - Cp|, where kernelPt is the centroid of the kernel loci of the trivariate, and Cp is the location of the control point. Fourth, re-center the inner control points grid around the kernel center. The function aims to build a trivariate with a valid parameterization. If the function failed, it will add dofs to the input surfaces by degree raising or uniform knot insertion.
See also: MvarKrnlTVOneSidedBSum2Srfs, MvarKrnlBooleanSumTV, MvarKrnlRuledTV, , MvarKrnlTVInptStructAlloc, MvarKrnlTVInptStructFree,

8.2.316 MvarKrnlTVResStructFree (krnl_based_cnstrct.c:2343)

```
void MvarKrnlTVResStructFree(MvarKrnlTVResStruct *Res)
```

Res: A Object to free.

Returns: void

Description: Free the given MvarKrnlTVResStruct object.
See also:

8.2.317 MvarKrnlTVResStructNew (krnl_based_cnstrct.c:2316)

```
MvarKrnlTVResStruct *MvarKrnlTVResStructNew()
```

Returns: The allocated object.

Description: Allocate a new MvarKrnlTVResStruct object.
See also:

8.2.318 MvarLclDistSrfLine (mvardist.c:560)

surface line distance

```
MvarPtStruct *MvarLclDistSrfLine(const CagdSrfStruct *CSrf,
                                 const CagdPType LnPt,
                                 const CagdVType LnDir,
                                 CagdRType SubdivTol,
                                 CagdRType NumericTol)
```

CSrf: The surface to find its nearest (farthest) point to Line.

LnPt: A point on the line to consider.

LnDir: The direction of the line to consider.

SubdivTol: Tolerance of the first zero set finding subdivision stage.

NumericTol: Tolerance of the second zero set finding numeric stage.

Returns: A list of parameter values of extreme distance locations.

Description: Given a surface and a line, finds the nearest point (if MinDist) or the farthest location (if MinDist FALSE) from the surface to the given line. This function assumes the surface does not intersect the line. Returned is the parameter value of the surface. Only internal extrema are considered. Let S and N be the surface and its normal field. Then the extrema points are computed as the simultaneous solution of,

```
< (S - LnPt) x N, LnDir > = 0,
< N, LnDir > = 0.
```

See also: MvarDistSrfLine, MvarDistCrvLine,

8.2.319 MvarLclDistSrfPoint (mvardist.c:367)

surface point distance

```
MvarPtStruct *MvarLclDistSrfPoint(const CagdSrfStruct *CSrf,
                                  void *SrfPtPrepHandle,
                                  const CagdPType Pt,
                                  MvarPtStruct **InitialSol,
                                  CagdRType SubdivTol,
                                  CagdRType NumericTol)
```

CSrf: The surface to find its extreme distance locations to Pt.

SrfPtPrepHandle: If not NULL, holds pre-processed data to speed up the surface - points distance computations, for multiple surface - point tests (same surface for all points). See MvarDistSrfPointPrep and MvarDistSrfPointFree.

Pt: The point to find the extreme distance locations from Srf.

InitialSol: Optional initial guess for a solution in which case we aim to improve numerically to within NumericTol. If successful, this solution is returned (in place here and the return value). Otherwise, full solver is invoked and InitialSol is freed!

SubdivTol: Tolerance of the first zero set finding subdivision stage.

NumericTol: Tolerance of the second zero set finding numeric stage.

Returns: A list of parameter values of extreme distance locations.

Description: Given a surface and a point, find the local extremum distance points on the surface to the given point. Only interior extrema are considered. Returned is a list of parameter value with local extremum. Computes the simultaneous zeros of:

```
(Srf(u, v) - Pt) . dSrf(u, v)/Du = 0.
(Srf(u, v) - Pt) . dSrf(u, v)/Dv = 0.
```

See also: MvarDistSrfPointPrep, MvarDistSrfPointFree,

8.2.320 MvarLineFitToPts (mvar_int.c:280)

interpolation

least square approximation

```
CagdRType MvarLineFitToPts(const MvarVecStruct *VecList,
                           MvarVecStruct *LineDir,
                           MvarVecStruct *LinePos)
```

VecList: List of vectors to interpolate/least square approximate.

LineDir: A unit vector of the line.

LinePos: A point on the computed line.

Returns: Average distance between a vector and the fitted line, or IRIT_INFNTY if failed.

Description: Given set of vecs, VecList, fits a line using least squares fit to them.
See also: CagdLineFitToPts,

8.2.321 MvarLinePlaneInter (mvar_vec.c:713)

```
MvarVecStruct *MvarLinePlaneInter(const MvarVecStruct *P,
                                  const MvarVecStruct *V,
                                  const MvarPlaneStruct *Pln,
                                  CagdRType *Param)
```

P, V: Point and direction of line to intersect with hyperplane. Both P and V are of length Dim.

Pln: Hyperplane to intersect with line.

Param: Will be updated with the parameter along which the intersection has occurred, as Inter = P + V * t.

Returns: The intersection point.

Description: Compute the intersection of a line and a hyperplane, in R^Dim.

8.2.322 MvarLineSrfInter (lnsrfdst.c:283)

```
MvarPtStruct *MvarLineSrfInter(const CagdPType LinePt,
                               const CagdVType LineDir,
                               const CagdSrfStruct *Srf,
                               CagdRType SubdivTol,
                               CagdRType NumericTol)
```

LinePt, LineDir: Line specification.

Srf: Surface to intersect.

SubdivTol: Subdivision tolerance of computation.

NumericTol: Numerical tolerance of computation.

Returns: List of intersection points as surface parameters (u,v).

Description: Computes intersection points of a surface and a line specified as intersection of two implicitly defined planes.
See also: MvarCrvSrfInter,

8.2.323 MvarMSCircOfThreeCurves (ms_circ.c:232)

```
int MvarMSCircOfThreeCurves(const CagdCrvStruct *OrigCrv1,
                            const CagdCrvStruct *OrigCrv2,
                            const CagdCrvStruct *OrigCrv3,
                            CagdRType Center[2],
                            CagdRType *Radius,
                            CagdRType SubdivTol,
                            CagdRType NumerTol)
```

OrigCrv1, OrigCrv2, OrigCrv3: The three curves to consider. Curves could be identical.

Center: Center of the computed MSC.

Radius: Radius of the computed MSC.

SubdivTol, NumerTol: Of computation.

Returns: TRUE if successful, FALSE otherwise

Description: Computes the minimum spanning circle to three curves that are disjoint. Assumption is made that the MSC is tangent to all three curves.
See also: MvarMSCircOfTwoCurves, MvarMinSpanCirc, MVarIsCrvInsideCirc, MvarSkel2DInter3Prims,

8.2.324 MvarMSCircOfTwoCurves (ms_circ.c:64)

```
int MvarMSCircOfTwoCurves(const CagdCrvStruct *OrigCrv1,
                          const CagdCrvStruct *OrigCrv2,
                          CagdRType Center[2],
                          CagdRType *Radius,
                          CagdRType SubdivTol,
                          CagdRType NumerTol)
```

OrigCrv1, OrigCrv2: The two curves to consider.

Center: Center of the computed MSC.

Radius: Radius of the computed MSC.

SubdivTol, NumerTol: Of computation.

Returns: TRUE if successful, FALSE otherwise

Description: Computes the minimum spanning circle to two curves that are disjoint. Assumption is made that the MSC is tangent to both curves.
See also: MvarMSCircOfThreeCurves, MvarMinSpanCirc, MVarIsCrvInsideCirc,

8.2.325 MvarMVAdd (mvar_sym.c:38)

addition

symbolic computation

multivariates

```
MvarMVStruct *MvarMVAdd(const MvarMVStruct *MV1, const MvarMVStruct *MV2)
```

MV1, MV2: Two multivariate to add up coordinate-wise.

Returns: The summation of MV1 + MV2 coordinate-wise.

Description: Given two multivariates - add them coordinate-wise. The two multivariates are promoted to same point type before the operation can take place. Furthermore, order and continuity are matched as well.
See also: MvarMVSub, MvarMeshAddSub, MvarMVMult,

8.2.326 MvarMVAuxDomainSlotCopy (mvar_aux.c:284)

```
int MvarMVAuxDomainSlotCopy(MvarMVStruct *MVDst, const MvarMVStruct *MVSrc)
```

MVDst: Destination multivariate to copy domain slot to.

MVSrc: Source multivariate to copy domain slot from.

Returns: TRUE if successful, FALSE otherwise.

Description: Copies the optional aux. domain slot from MVSrc to MVDst.
See also: MvarMVAuxDomainSlotReset, MvarMVAuxDomainSlotSet, , MvarMVAuxDomainSlotSet, MvarMVAuxDomainSlotSetRel,

8.2.327 MvarMVAuxDomainSlotGet (mvar_aux.c:385)

```
int MvarMVAuxDomainSlotGet(const MvarMVStruct *MV,
                           CagdRType *Min,
                           CagdRType *Max,
                           int Dir)
```

MV: Multivariate to get its aux. domain slot.

Min, Max: The domain of this Bezier multivariate.

Dir: The direction to get the domain of MV.

Returns: TRUE if aux. domain exist and can get domain, FALSE otherwise.

Description: Gets one aux. domain range in the given Direction. The Min/Max arrays are assumed to be of sufficiently large enough space to hold all dimensions, if Axis == -1.
See also: MvarMVAuxDomainSlotReset, MvarMVAuxDomainSlotCopy, , MvarMVAuxDomainSlotSet, MvarMVAuxDomainSlotSetRel,

8.2.328 MvarMVAuxDomainSlotReset (mvar_aux.c:252)

```
void MvarMVAuxDomainSlotReset(MvarMVStruct *MV)
```

MV: Multivariate to reset domain slot.

Returns: void

Description: Resets an optional aux. domain slot inside MV for use in cases where MV has no explict domain (such as Bezier and/or Power basis).
See also: MvarMVAuxDomainSlotCopy, MvarMVAuxDomainSlotSet, , MvarMVAuxDomainSlotSet, MvarMVAuxDomainSlotSetRel,

8.2.329 MvarMVAuxDomainSlotSet (mvar_aux.c:322)

```
void MvarMVAuxDomainSlotSet(MvarMVStruct *MV,
                            CagdRType Min,
                            CagdRType Max,
                            int Dir)
```

MV: Multivariate to set its aux. domain slot.

Min, Max: The domain of this Bezier multivariate.

Dir: The direction where to set the MV domain.

Returns: void

Description: Sets one aux. domain range in the given direction.
See also: MvarMVAuxDomainSlotReset, MvarMVAuxDomainSlotCopy, , MvarMVAuxDomainSlotGet, MvarMVAuxDomainSlotSetRel,

8.2.330 MvarMVAuxDomainSlotSetRel (mvar_aux.c:352)

```
void MvarMVAuxDomainSlotSetRel(MvarMVStruct *MV,
                               CagdRType Min,
                               CagdRType Max,
                               int Dir)
```

MV: Multivariate to set its aux. domain slot.

Min, Max: The relative to current domain new interval of this Bezier multivariate.

Dir: The direction where to set the MV domain.

Returns: void

Description: Sets one aux. domain range in the given direction. Input Min/Max is relative to current domain.
See also: MvarMVAuxDomainSlotReset, MvarMVAuxDomainSlotCopy, , MvarMVAuxDomainSlotSet, MvarMVAuxDomainSlotGet,

8.2.331 MvarMVAvgArcLenMesh (mvar_aux.c:2266)

multi-variates

```
CagdRType MvarMVAvgArcLenMesh(const MvarMVStruct *MV, MvarMVDirType Dir)
```

MV: Multi-Variate to compute its mesh's arc-length in Dir.

Dir: Direction of arc length estimation.

Returns: Avarage arc-length of the mesh of MV in directuion Dir.

Description: Computes the average arc length of the mesh of MV in direction Dir.
See also: MvarMVDerive, CagdCrvArcLenPoly,

8.2.332 MvarMVBBox (mvarbbox.c:41)

bbox

bounding box

```
MvarBBoxStruct *MvarMVBBox(const MvarMVStruct *MV, MvarBBoxStruct *BBox)
```

MV: To compute a bounding box for.

BBox: Where bounding information is to be saved.

Returns: The computed BBox. Same as the BBox parameter.

Description: Computes a bounding box for a multi-variate freeform function.

8.2.333 MvarMVBiTangentLine (mvtangnt.c:1189)

bi-tangent

```
MvarPtStruct *MvarMVBiTangentLine(const CagdCrvStruct *Crv1,
                                  const CagdCrvStruct *Crv2,
                                  CagdRType SubdivTol,
                                  CagdRType NumericTol)
```

Crv1, Crv2: The 2 curves to compute their bi-tangent lines for. If Crv1 == Crv2, the self bi-tangent is computed.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol < SubdivTol.

Returns: Pairs of parameter values on both curves, each pair defines a single bi-tangent.

Description: Computes the bi-tangent line of two freeform curves. If Crv1 == Crv2, the self bi-tangent is computed.
See also: SymbTangentToCrvAtTwoPts, MvarMVBiTangents, MvarMVBiTangents2, MvarMVTriTangentLineCreateMVs, MvarMVTriTangentLine,

8.2.334 MvarMVBiTangents (mvtangnt.c:65)

bi-tangent

```
MvarPolylineStruct *MvarMVBiTangents(const MvarMVStruct *CMV1,
                                     const MvarMVStruct *CMV2,
                                     CagdRType SubdivTol,
                                     CagdRType NumericTol)
```

CMV1, CMV2: The two multivariates to compute the bi-tangents for.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol < SubdivTol.

Returns: Pllns on the bi-tangents of the two multivariates.

Description: Computes bi-tangents of freeform bivariate. Let,

```
DMV = MV1(u, v) - MV2(r, s)
```

then, computed the simultaneous solution of the following three equations:

```
  d MV1   d MV1  d MV2
< ----- x -----, ----- > = 0,
    du      dv     dr
```

```
  d MV1   d MV1  d MV2
< ----- x -----, ----- > = 0,
    du      dv     ds
```

```
  d MV1   d MV1
< ----- x -----, DMV > = 0.
    du      dv
```

See also: SymbTangentToCrvAtTwoPts, MvarMVBiTangents2, MvarMVTriTangents, MvarMVTriTangentLine,

8.2.335 MvarMVBoundGradient (mvar_der.c:902)

```
MvarMVGradientStruct *MvarMVBoundGradient(const MvarMVStruct *MV)
```

MV: Input MV function to compute bounds on its gradient.

Returns: Holding the set of vectors bounding the gradient of MV in the MVGrad slot.

Description: Provides a set of vectors that bounds the gradient function of given MV
See also: MvarMVDerive, MvarMVFreeGradient, MvarMVPrepGradient, , MvarMVEvalGradient, MvarMVEvalGradient2,

8.2.336 MvarMVCompose (mvcompos.c:138)

composition

```
MvarMVStruct *MvarMVCompose(const MvarMVStruct *TargetMV,
                            const MvarMVStruct *ReParamMV,
                            const int *DimMapping,
                            CagdBType DoMerge)
```

TargetMV: The target multivariate.

ReParamMV: The reparametrization multivariate.

DimMapping: The mapping from ReParamMV's range dimensions to TargetMV's domain dimensions. Of length of dimension of range of ReParamMV and holds indices into the domain of TargetMV. If NULLm assumes the trivial mapping - range of ReParamMV is mapped to the domain of TargetMV.

DoMerge: A flag indicating whether to merge the results into a single multivariate (applies if the reparametrization is B-spline, and was subdivided).

Returns: The composed multivariate TargetMV(ReParamMV(t0,...,tn)) or a list of composed multivariates.

Description: The main interface function for composition of multivariates. Given two multivariates TargetMV and ReParamMV, it returns their composition TargetMV(ReParamMV(t0,...,tn)) . The composition is possible if the range of the reparametrization multivariate is contained in a patch of the target multivariate that doesn't have internal knots. If the reparametrization multivariate is B-spline, it is subdivided at the knots, and the resulting composed multivariates are either merged into a single multivariate, or returned as a linked list, depending on the flag DoMerge.
See also: MvarComputeMVPowers, MvarGenerateBspBasisMVs, TrivComposeTVCrv, , TrivComposeTVCrv, SymbComposeCrvCrv, SymbComposeSrfCrv, , SymbComposeSrfSrf,

8.2.337 MvarMVCompose2 (mvcomps2.c:1581)

composition

multivariates

```
IPObjectStruct *MvarMVCompose2(const MvarMVStruct *MVMapping,
                               const MvarMVStruct *MVToCompose,
                               int HandleCrossKVLines)
```

MVMapping: The mapping multivariate function.

MVToCompose: The multivariate function to compose.

HandleCrossKVLines: Whenever possible, TRUE to divide first Obj at all locations it crosses knot lines in MVMapping, and treat the crossing of knot lines correctly.

Returns: The composed function.

Description: Compose two multivariate functions: MVMapping(MVToCompose). The function chooses the most general type of composition given the dimensions of the parameters.
See also: MvarMVCompose, MvarMVCompose3,

8.2.338 MvarMVCompose3 (mvcomps2.c:1431)

composition

multivariates

```
IPObjectStruct *MvarMVCompose3(const IPObjectStruct *MappingFunction,
                               const IPObjectStruct *Obj,
                               int HandleCrossKVLines)
```

MappingFunction: The geometry representing the mapping function.

Obj: The geometry to compose.

HandleCrossKVLines: Whenever possible, TRUE to divide first Obj at all locations it crosses knot lines in MVMapping, and treat the crossing of knot lines correctly.

Returns: The composed geometry.

Description: Compose two geometric structures: MappingFunction(Obj). The function chooses the most general type of composition (including knot-crossing), given the types of the parameters.
See also: MvarMVCompose, MvarMVCompose2,

8.2.339 MvarMVConesOverlap (mvcones.c:1703)

```
CagdBType MvarMVConesOverlap(MvarMVStruct **MVs, int NumOfZeroMVs)
```

MVs: Multivariates to derive their tangency anti-cones.

NumOfZeroMVs: Size of the vector MVs.

Returns: TRUE if overlap, FALSE if not.

Description: Computes the tangency anti-cones of the set of multivariate constraints, and returns whether they overlap or not.
See also: MVarMVNormalCone, MvarConesOverlapAux,

8.2.340 MvarMVCopy (mvar_gen.c:301)

multi-variates

```
MvarMVStruct *MvarMVCopy(const MvarMVStruct *MV)
```

MV: Multi-Variate to duplicate

Returns: Duplicated multi-variate.

Description: Allocates and duplicates all slots of a multi-variate structure.

8.2.341 MvarMVCopyList (mvar_gen.c:428)

multi-variates

```
MvarMVStruct *MvarMVCopyList(const MvarMVStruct *MVList)
```

MVList: List of multi-variates to duplicate.

Returns: Duplicated list of multi-variates.

Description: Duplicates a list of multi-variate structures.

8.2.342 MvarMVCornersMinMax (mvarbbox.c:124)

```
void MvarMVCornersMinMax(const MvarMVStruct *MV,
                         int Coord,
                         CagdRType *Min,
                         CagdRType *Max)
```

MV: In which to find the minimum and maximum at the corners of the control mesh.

Coord: The coordinate of the minimum/maximum values that need to be found, or -1 for all coordinates.

Min, Max: Output parameters. Pointers into which the minimum and maximum values are written. If more than one coordinate is requested, then Min and Max must be arrays of sufficient lengths for all the coordinates.

Returns: void

Description: Computes the minimum and maximum values of the control points at the corners of the control mesh of a multivariate. Can be applied to a specific coordinate or to all the coordinates of the multivariate.
See also: MvarMVMinMax,

8.2.343 MvarMVCrossProd (mvar_sym.c:647)

product

cross product

multivariates

mbolic computation

```
MvarMVStruct *MvarMVCrossProd(const MvarMVStruct *MV1,
                              const MvarMVStruct *MV2)
```

MV1, MV2: Two multivariate to multiply and compute a cross product for.

Returns: A vector multivariate representing the cross product of MV1 x MV2.

Description: Given two multivariates - computes their cross product. Returned multivariate is a vector multivariate representing the cross product of the two given multivariates.
See also: MvarMVDotProd, MvarMVVecDotProd, MvarMVScalarScale, MvarMVMultScalar, , MvarMVInvert, MvarMVCrossProd2D, MvarMVCrossProdZ,

8.2.344 MvarMVCrossProd2D (mvar_sym.c:744)

product

cross product

multivariates

mbolic computation

```
MvarMVStruct *MvarMVCrossProd2D(const MvarMVStruct *MV1X,
                                const MvarMVStruct *MV1Y,
                                const MvarMVStruct *MV2X,
                                const MvarMVStruct *MV2Y)
```

MV1X, MV1Y: First pair of scalar multivariates (X, Y) of first funcs.

MV2X, MV2Y: Second pair of scalar multivariates (X, Y) of second funcs.

Returns: A scalar multivariate representing the cross product of MV1X * MV2Y - MV2X * MV1Y.

Description: Given four multivariates - computes their 2D cross product. Returned multivariate is a scalar multivariate representing the cross product of the four given multivariates, as X1 * Y2 - X2 * Y1.
See also: MvarMVDotProd, MvarMVVecDotProd, MvarMVScalarScale, MvarMVMultScalar, , MvarMVInvert, MvarMVCrossProd, MvarMVCrossProdZ,

8.2.345 MvarMVCrossProdZ (mvar_sym.c:602)

product

cross product

multivariates

symbolic computation

```
MvarMVStruct *MvarMVCrossProdZ(const MvarMVStruct *MV1,
                               const MvarMVStruct *MV2)
```

MV1, MV2: Two multivariate to multiply and compute a Z component of cross product for.

Returns: A scalar multivariate representing the Z component of the cross product of MV1 x MV2.

Description: Given two multivariates - computes the Z component of the cross product. Returned multivariate is a scalar multivariate representing the Z component of the cross product of the two given multivariates.
See also: MvarMVDotProd, MvarMVVecDotProd, MvarMVScalarScale, MvarMVMultScalar, , MvarMVInvert, MvarMVCrossProd, MvarMVCrossProd2D,

8.2.346 MvarMVDSetDecompositionMode (zrdcm_main.c:2695)

MvarZeroSolverWithDecomposition

```
MvarMVDDecompositionModeType MvarMVDSetDecompositionMode(
                                          MvarMVDDecompositionModeType m)
```

m: The new decomposition mode.

Returns: The previous decomposition mode.

Description: Sets the decomposition mode used by the multivariate solver. The possible decomposition modes are: MVAR_MVD_NO_DECOMPOSITION - no decomposition MVAR_MVD_DECOMPOSITION_COMPOSITION - decomposition and propagation of the partial solutions using functional composition. MVAR_MVD_DECOMPOSITION_POINT_T - decomposition and point-by-point tracing of the complete solution from the solution to the first univariate problem. THIS MODE IS UNTESTES AND UNRECOMMENDED FOR USE

8.2.347 MvarMVDegreeRaise (mvarrais.c:323)

degree raising

```
MvarMVStruct *MvarMVDegreeRaise(const MvarMVStruct *MV, MvarMVDirType Dir)
```

MV: To raise its degree.

Dir: Direction of degree raising.

Returns: A multivariate with same geometry as MV but with one degree higher.

Description: Returns a new multivariate representing the same curve as MV but with its degree raised by one.
See also: MvarMVDegreeRaiseN, MvarMVDegreeRaise2, MvarMVPwrDegreeRaise,

8.2.348 MvarMVDegreeRaise2 (mvarrais.c:358)

degree raising

```
MvarMVStruct *MvarMVDegreeRaise2(MvarMVStruct *MV, MvarMVDirType Dir)
```

MV: To raise its degree.

Dir: Direction of degree raising.

Returns: A multivariate with same geometry as MV but with one degree higher.

Description: Returns a new multivariate representing the same curve as MV but with its degree raised by one.
See also: MvarMVDegreeRaise, , MvarMVDegreeRaise3, MvarMVDegreeRaiseN, , MvarMVPwrDegreeRaise,

8.2.349 MvarMVDegreeRaise3 (mvarrais.c:440)

degree raising

```
MvarMVStruct *MvarMVDegreeRaise3(MvarMVStruct *MV, MvarMVDirType Dir)
```

MV: To raise its degree.

Dir: Direction of degree raising.

Returns: A multivariate with same geometry as MV but with one degree higher.

Description: Returns a new multivariate representing the same curve as MV but with its degree raised by one.
See also: MvarMVDegreeRaiseN, MvarMVDegreeRaise, MvarMVDegreeRaise2, , MvarMVPwrDegreeRaise,

8.2.350 MvarMVDegreeRaiseN (mvarrais.c:34)

degree raising

```
MvarMVStruct *MvarMVDegreeRaiseN(const MvarMVStruct *MV, int *NewOrders)
```

MV: To raise its degree.

NewOrders: A vector prescribing the new orders of MV. Length of this vector is MV -> Dim.

Returns: A multivariate with same geometry as MV but with higher degrees.

Description: Returns a new multivariate representing the same curve as MV but with its degree raised by the NewOrders prescription.
See also: MvarMVDegreeRaise, MvarMVPwrDegreeRaise, MvarMVDegreeRaiseN2,

8.2.351 MvarMVDegreeRaiseN2 (mvarrais.c:228)

degree raising

```
MvarMVStruct *MvarMVDegreeRaiseN2(const MvarMVStruct *MV, int *NewOrders)
```

MV: To raise its degree.

NewOrders: A vector prescribing the new orders of MV. Length of this vector is MV -> Dim.

Returns: A multivariate with same geometry as MV but with higher degrees.

Description: Returns a new multivariate representing the same curve as MV but with its degree raised by the NewOrders prescription.
See also: MvarMVDegreeRaise, MvarMVPwrDegreeRaise, MvarMVDegreeRaiseN,

8.2.352 MvarMVDerive (mvar_der.c:33)

multi-variates

```
MvarMVStruct *MvarMVDerive(const MvarMVStruct *MV, MvarMVDirType Dir)
```

MV: Multi-Variate to differentiate.

Dir: Direction of differentiation.

Returns: Differentiated multi-variate in direction Dir.

Description: Given a multi-variate, computes its partial derivative multi-variate in direction Dir.
See also: MvarMVDeriveBound, MvarBzrMVDerive, MvarBspMVDerive,

8.2.353 MvarMVDeriveAllBounds (mvar_der.c:483)

multi-variates

```
void MvarMVDeriveAllBounds(const MvarMVStruct *MV, CagdMinMaxType *MinMax)
```

MV: Multi-Variate to differentiate.

MinMax: Bounds on the derivative values of MV in all directions.

Returns: void

Description: Given a multi-variate, computes its partial derivative multi-variate in all directions.
See also: MvarMVDerive, MvarBzrMVDeriveBound, MvarBspMVDeriveBound,

8.2.354 MvarMVDeriveBound (mvar_der.c:313)

multi-variates

```
void MvarMVDeriveBound(const MvarMVStruct *MV,
                       MvarMVDirType Dir,
                       CagdRType MinMax[2])
```

MV: Multi-Variate to differentiate.

Dir: Direction of differentiation.

MinMax: Bounds on the derivative values of MV in direction Dir.

Returns: void

Description: Given a multi-variate, computes its partial derivative multi-variate in direction Dir.
See also: MvarMVDerive, MvarBzrMVDeriveBound, MvarBspMVDeriveBound,

8.2.355 MvarMVDeterminant (mvar_det.c:296)

determinant

```
MvarMVStruct *MvarMVDeterminant(const MvarMVStruct * const *MVsMatrix,
                                int MatSize)
```

MVsMatrix: The matrix of MVs, as a linear array, with the following double index to linear index convention - [i][j] <—-> [(i - 1) * MatSize + (j - 1)], where i, j = 1, 2, ..., MatSize (double index as in the usual linear algebra convention).

MatSize: The size in each of the dimensions of the square MVs matrix.

Returns: A scalar field representing the determinant computation.

Description: Computes the expression of a determinant of MVs, recursively. Expansion is always using the first row. Does not check for a better option, such as a row with more zeros, etc. (Existing zeros means the zero function as a multivariate).
See also: MvarMVDeterminant2,

8.2.356 MvarMVDeterminant2 (mvar_det.c:42)

determinant

```
MvarMVStruct *MvarMVDeterminant2(const MvarMVStruct *MV11,
                                 const MvarMVStruct *MV12,
                                 const MvarMVStruct *MV21,
                                 const MvarMVStruct *MV22)
```

MV11, MV12, MV21, MV22: The four factors of the determinant.

Returns: A scalar field representing the determinant computation.

Description: Computes the expression of MV11 * MV22 - MV12 * MV21, which is a determinant of a 2 by 2 matrix.
See also: MvarMVDeterminant3, MvarSrfDeterminant2, MvarCrvDeterminant2,

8.2.357 MvarMVDeterminant3 (mvar_det.c:81)

determinant

```
MvarMVStruct *MvarMVDeterminant3(const MvarMVStruct *MV11,
                                 const MvarMVStruct *MV12,
                                 const MvarMVStruct *MV13,
                                 const MvarMVStruct *MV21,
                                 const MvarMVStruct *MV22,
                                 const MvarMVStruct *MV23,
                                 const MvarMVStruct *MV31,
                                 const MvarMVStruct *MV32,
                                 const MvarMVStruct *MV33)
```

MV11, MV12, MV13: The nine factors of the determinant.

MV21, MV22, MV23: ”

MV31, MV32, MV33: ”

Returns: A scalar field representing the determinant computation.

Description: Computes the expression of a 3 by 3 determinants.
See also: MvarMVDeterminant2, MvarMVDeterminant4, SymbSrfDeterminant3, SymbCrvDeterminant3,

8.2.358 MvarMVDeterminant4 (mvar_det.c:138)

determinant

```
MvarMVStruct *MvarMVDeterminant4(const MvarMVStruct *MV11,
                                 const MvarMVStruct *MV12,
                                 const MvarMVStruct *MV13,
                                 const MvarMVStruct *MV14,
                                 const MvarMVStruct *MV21,
                                 const MvarMVStruct *MV22,
```

```
                                const MvarMVStruct *MV23,
                                const MvarMVStruct *MV24,
                                const MvarMVStruct *MV31,
                                const MvarMVStruct *MV32,
                                const MvarMVStruct *MV33,
                                const MvarMVStruct *MV34,
                                const MvarMVStruct *MV41,
                                const MvarMVStruct *MV42,
                                const MvarMVStruct *MV43,
                                const MvarMVStruct *MV44)
```

MV11, MV12, MV13, MV14: The 16 factors of the determinant.

MV21, MV22, MV23, MV24: ”

MV31, MV32, MV33, MV34: ”

MV41, MV42, MV43, MV44: ”

Returns: A scalar field representing the determinant computation.

Description: Computes the expression of a 4 by 4 determinants.
See also: MvarMVDeterminant3, MvarMVDeterminant5,

8.2.359 MvarMVDeterminant5 (mvar_det.c:220)

determinant

```
MvarMVStruct *MvarMVDeterminant5(const MvarMVStruct *MV11,
                                 const MvarMVStruct *MV12,
                                 const MvarMVStruct *MV13,
                                 const MvarMVStruct *MV14,
                                 const MvarMVStruct *MV15,
                                 const MvarMVStruct *MV21,
                                 const MvarMVStruct *MV22,
                                 const MvarMVStruct *MV23,
                                 const MvarMVStruct *MV24,
                                 const MvarMVStruct *MV25,
                                 const MvarMVStruct *MV31,
                                 const MvarMVStruct *MV32,
                                 const MvarMVStruct *MV33,
                                 const MvarMVStruct *MV34,
                                 const MvarMVStruct *MV35,
                                 const MvarMVStruct *MV41,
                                 const MvarMVStruct *MV42,
                                 const MvarMVStruct *MV43,
                                 const MvarMVStruct *MV44,
                                 const MvarMVStruct *MV45,
                                 const MvarMVStruct *MV51,
                                 const MvarMVStruct *MV52,
                                 const MvarMVStruct *MV53,
                                 const MvarMVStruct *MV54,
                                 const MvarMVStruct *MV55)
```

MV11, MV12, MV13, MV14, MV15: The 25 factors of the determinant.

MV21, MV22, MV23, MV24, MV25: ”

MV31, MV32, MV33, MV34, MV35: ”

MV41, MV42, MV43, MV44, MV45: ”

MV51, MV52, MV53, MV54, MV55: ”

Returns: A scalar field representing the determinant computation.

Description: Computes the expression of a 5 by 5 determinants.
See also: MvarMVDeterminant4,

8.2.360 MvarMVDistCrvSrf (mvardist.c:628)

curve surface distance

```
MvarMVStruct *MvarMVDistCrvSrf(const CagdCrvStruct *Crv1,
                               const CagdSrfStruct *Srf2,
                               int DistType)
```

Crv1, Srf2: The two entities, Crv1(t) and Srf2(u, v), to form their distance function square between them as a multivariate function.

DistType: 0 for distance vector function, 1 for distance square scalar function, 2 for distance vector projected on the normal of Crv1, 3 for distance vector projected on the normal of Srf2. In cases 2 and 3 the normal field is not normalized.

Returns: The distance function square d2(t, u, v) of the distance from Crv1(t) to Srf2(u, v).

Description: Given a curve and a surface, creates a multivariate scalar field representing the distance function square, between them.
See also: SymbSrfDistCrvCrv, SymbCrvCrvInter, SymbSrfDistFindPoints, , MvarMVDistSrfSrf, MvarProjCrvOnSrf2,

8.2.361 MvarMVDistSrfSrf (mvardist.c:723)

surface surface distance

```
MvarMVStruct *MvarMVDistSrfSrf(const CagdSrfStruct *Srf1,
                               const CagdSrfStruct *Srf2,
                               int DistType)
```

Srf1, Srf2: The two surfaces, Srf1(u, v) and Srf2(r, t), to form their distance function square between them as a multivariate function.

DistType: 0 for distance vector function, 1 for distance square scalar function, 2 for distance vector projected on the normal of Srf1, 3 for distance vector projected on the normal of Srf2. In cases 2 and 3 the normal field is not normalized.

Returns: The distance function square d2(u, v, r, t) of the distance from Srf1(u, v) to Srf2(r, t).

Description: Given two surfaces, creates a multivariate scalar field representing the distance function square, between them.
See also: SymbSrfDistCrvCrv, SymbCrvCrvInter, SymbSrfDistFindPoints, , MvarMVDistCrvSrf,

8.2.362 MvarMVDomain (mvar_aux.c:49)

multi-variates

```
void MvarMVDomain(const MvarMVStruct *MV,
                  CagdRType *Min,
                  CagdRType *Max,
                  int Axis)
```

MV: Multivariate function to consider.

Min: Minimum domains of MV will be placed herein.

Max: Maximum domains of MV will be placed herein.

Axis: axis to extract or -1 for all axes.

Returns: void

Description: Given a multi-variate, returns its parametric domain. The Min/Max arrays are assumed to of sufficiently large enough space to hold all dimensions, if Axis == -1.
See also: varMVSetDomain, MvarMVSetAllDomains, MvarParamInDomain, , varParamsInDomain,

8.2.363 MvarMVDomainAlloc (mvar_aux.c:147)

```
void MvarMVDomainAlloc(const MvarMVStruct *MV,
                       CagdRType **MinDmn,
                       CagdRType **MaxDmn)
```

MV: Multivariate structures to receive its domain.

MinDmn: Array of maximal values.

MaxDmn: Array of minimal values.

Returns:

Description: Same as MvarMVDomain but also allocate the space to hold the domain.
See also: MvarMVDomain, MvarMVDomainFree,

8.2.364 MvarMVDomainFree (mvar_aux.c:174)

```
void MvarMVDomainFree(CagdRType *MinDmn, CagdRType *MaxDmn)
```

MinDmn: Array of maximal values.

MaxDmn: Array of minimal values.

Returns:

Description: Deallocates the memory of MV's domain, as allocated by MvarMVDomain2.
See also: MvarMVDomain, MvarMVDomainAlloc,

8.2.365 MvarMVDotProd (mvar_sym.c:494)

product

dot product

symbolic computation

multivariates

```
MvarMVStruct *MvarMVDotProd(const MvarMVStruct *MV1, const MvarMVStruct *MV2)
```

MV1, MV2: Two multivariate to multiply and compute a dot product for.

Returns: A scalar multivariate representing the dot product of MV1 . MV2.

Description: Given two multivariates - computes their dot product. Returned multivariate is a scalar multivariate representing the dot product of the two given multivariates. While typically in R3, the dot product can be computed for any dimension of MV1 and MV2.
See also: MvarMVMult, MvarMVVecDotProd, MvarMVScalarScale, MvarMVMultScalar, , MvarMVInvert, MvarMVCrossProd,

8.2.366 MvarMVEvalGradient (mvar_der.c:809)

```
CagdRType *MvarMVEvalGradient(const MvarMVGradientStruct *MVGrad,
                              const CagdRType *Params,
                              int Axis,
                              CagdRType *Grad)
```

MVGrad: Input gradient function to evaluate at.

Params: Parametric location to evaluate gradient at.

Axis: If the input function whose gradient we seek is scalar, Axis will always be zero. However we can also handle input vector functions in which case Axis specifies which function in the vector to compute the gradient for - 0 for X, 1 for Y, etc.

Grad: The gradient at Params.

Returns: The gradient at Params.

Description: Evaluates the gradient function at the given parametric location.
See also: MvarMVDerive, MvarMVFreeGradient, MvarMVPrepGradient, , MvarMVEvalGradient2, MvarMVBoundGradient,

8.2.367 MvarMVEvalGradient2 (mvareval.c:559)

evaluation
multi-variates
gradient

```
CagdRType *MvarMVEvalGradient2(const MvarMVStruct *MV,
                               const CagdRType *Params,
                               int *HasOrig,
                               CagdRType *Grad)
```

MV: To evaluate its gradient at given Params parametric location.

Params: Parametric location to evaluate MV at.

HasOrig: TRUE if the cached gradient also contains the original scalar field, as last, additional, coordinate.

Grad: A vector holding all the coefficients of all components of the multi-variate's point type.

Returns: A vector holding all the coefficients of all components of the multi-variate's point type. If for example multi-variate point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1). Same as Grad.

Description: Evaluates the gradient of the given multivariate function at a given location, numerically. Allowed for scalar multivariates only.
See also: MvarMVEval, MvarMVEvalTanPlane, MvarMVEvalGradient,

8.2.368 MvarMVEvalMalloc (mvareval.c:517)

evaluation
multi-variates

```
CagdRType *MvarMVEvalMalloc(const MvarMVStruct *MV, const CagdRType *Params)
```

MV: To evaluate at given Params parametric location.

Params: Parametric location to evaluate MV at.

Returns: A vector holding all the coefficients of all components of the multi-variate's point type. If for example multi-variate point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Description: Evaluates the given multivariate function at a given location. Same functionality as MvarMVEval but a different implementation.
See also: MvarMVEvalGradient, MvarMVEvalGradient2, MvarMVEvalToData,

8.2.369 MvarMVEvalTanPlane (mvareval.c:642)

evaluation
multi-variates
gradient

```
MvarPlaneStruct *MvarMVEvalTanPlane(const MvarMVStruct *MV,
                                    const CagdRType *Params)
```

MV: To evaluate its gradient at given Params parametric location.

Params: Parametric location to evaluate MV at.

Returns: A hyperplane, allocated dynammically. The tanegnt is normalized so that its last (independent coefficient is one: "A1 X1 + A2 X2 + ... + An Xn + 1". The size, n, is to the dimension of the multivariate.

Description: Evaluates the tangent hyperplane of the given multivariate function at a given location, numerically. Assumes a scalar multivariates of n parameters in a space of dimension n+1 (An explicit surface in E3).
See also: MvarMVEval, MvarMVEvalGradient2,

8.2.370 MvarMVEvalToData (mvareval.c:43)

evaluation

multi-variates

```
void MvarMVEvalToData(const MvarMVStruct *MV,
                      const CagdRType *Params,
                      CagdRType *Pt)
```

MV: To evaluate at given Params parametric location.

Params: Parametric location to evaluate MV at.

Pt: A vector holding all the coefficients of all components of the multi-variate's point type. If for example multi-variate point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Returns: void

Description: Evaluates the given multivariate function at a given location. Same functionality as MvarMVEval but a different implementation.
See also: MvarMVEvalGradient, MvarMVEvalGradient2, MvarMVEvalMalloc,

8.2.371 MvarMVExtension (mvar_gen.c:1891)

```
MvarMVStruct *MvarMVExtension(const MvarMVStruct *OrigMV,
                              const CagdBType *ExtMins,
                              const CagdBType *ExtMaxs,
                              const CagdRType *Epsilons)
```

OrigMV: The multivariate to be extended.

ExtMins: A vector of Dim Boolean values to set the extension directions in the Min side. Direction i is extended in its Min side if ExtMins[i] is TRUE, or if ExtMins is NULL.

ExtMaxs: A vector of Dim Boolean values to set the extension directions in the Max side. Direction i is extended in its Max side if ExtMaxs[i] is TRUE, or if ExtMaxs is NULL.

Epsilons: A vector of real numbers representing the length of the extension in each direction (both Min and Max).

Returns: The new extended MV.

Description: Extension of a B-spline muiltivariate, in any of the 2 * Dim optional directions of the Dim dimensional domain. The extension is such that the image coincides with the original image over the original domain. Assumes open end conditions (in all knot vectors). OrigNV can have "ExtntSclU", "ExtntSclV" and/or "ExtntSclW" (or for all axes "ExtndScl") real attributes to scale the extension in Euclidean space (for same Epsilon parametric extension).
See also: BspCrvExtensionOneSide, BspCrvExtraKnotRmv, , BspCrvExtension, BspSrfExtension,

8.2.372 MvarMVFactorRMinusT (mvar_sym.c:1156)

```
MvarMVStruct *MvarMVFactorRMinusT(const MvarMVStruct *MV, int RIdx, int TIdx)
```

MV: multivariate to factor out (r - t) term from.

RIdx, TIdx: Indices of the r and t axes inside MV.

Returns: Factored out multivariate.

Description: Removes a (r - t) factor from the given scalar multivariate which is assumed to be of dimension greater than 3. The r, t parameters are the RIdx and TIdx parameters of the multivariate.
See also: BspSrfFactorUMinusV, BzrSrfFactorUMinusV, MvarMVFactorUMinusV,

8.2.373 MvarMVFactorUMinusV (mvar_sym.c:1213)

```
MvarMVStruct *MvarMVFactorUMinusV(const MvarMVStruct *MV)
```

MV: Multivariate to factor out (u - v) term from.

Returns: Factored out multivariate.

Description: Removes a (u - v) factor from the given scalar multivariate which is assumed to be of dimension 2 or more. The u, v parameters are the first and second parameters of the multivariate.
See also: BspSrfFactorUMinusV, BzrSrfFactorUMinusV, MvarMVFactorRMinusT,

8.2.374 MvarMVFree (mvar_gen.c:460)

multi-variates

```
void MvarMVFree(MvarMVStruct *MV)
```

MV: Multi-Variate to free.

Returns: void

Description: Deallocates and frees all slots of a multi-variate structure.
See also: MvarMVNew,

8.2.375 MvarMVFreeGradient (mvar_der.c:763)

```
void MvarMVFreeGradient(MvarMVGradientStruct *MVGrad)
```

MVGrad: Gradient function to free.

Returns: void

Description: Free an gradient function.
See also: MvarMVDerive, MvarMVPrepGradient, MvarMVEvalGradient,

8.2.376 MvarMVFreeList (mvar_gen.c:523)

multi-variates

```
void MvarMVFreeList(MvarMVStruct *MVList)
```

MVList: Multi-Variate list to free.

Returns: void

Description: Deallocates and frees a list of multi-variate structures.

8.2.377 MvarMVFromMV (mvareval.c:695)

multi-variates

```
MvarMVStruct *MvarMVFromMV(const MvarMVStruct *MV,
                           CagdRType t,
                           MvarMVDirType Dir)
```

MV: To extract an isoparametric multi-variate from at parameter value t in direction Dir, or exapand its dimension by one.

t: Parameter value at which to extract the isosurface (if Dir >= 0).

Dir: Direction of isosurface extraction. If Dir is negative, however, its absolute value defines the order of a new axis added as last and new dimension to the given MV.

Returns: A multi - variate with one less (or more) dimensions.

Description: Extract an isoparametric sub multivariate out of the given tensor product multivariate, or expand its dimension by one.
See also: MvarMVReverse, MvarMVFromMesh, MvarPromoteMVToMV,

8.2.378 MvarMVFromMesh (mvareval.c:877)

multi-variates

```
MvarMVStruct *MvarMVFromMesh(const MvarMVStruct *MV,
                             int Index,
                             MvarMVDirType Dir)
```

MV: To extract an isoparametric multi-variate from as a sub-mesh in direction Dir, or exapand its dimension by one.

Index: Index of sub mesh of MV's mesh in direction Dir.

Dir: Direction of isosurface extraction. If Dir is negative, however, its absolute value defines the order of a new axis added as last and new dimension to the given MV.

Returns: A multi - variate with one less (or more) dimensions.

Description: Extract an isoparametric sub multi variate out of the given tensor product multi-variate, or expand its dimension by one.
See also: MvarMVReverse, MvarMVFromMVm, MvarPromoteMVToMV,

8.2.379 MvarMVIntersPtOnBndry (mvar_aux.c:591)

```
MvarPtStruct *MvarMVIntersPtOnBndry(MvarMVStruct *MV,
                                    MvarPtStruct *PointIns,
                                    MvarPtStruct *PointOuts)
```

MV: Multivariate structure.

PointIns: point inside the domain.

PointOuts: point outside the domain.

Returns: The intersection point.

Description: Computes the intersection point of line (PoinIns, PointOuts) with the boundary of the domain of multivariate MV.
See also: MvarParamsInDomain,

8.2.380 MvarMVInvert (mvar_sym.c:289)

division
symbolic computation
reciprocal value
ltivariates

```
MvarMVStruct *MvarMVInvert(const MvarMVStruct *MV)
```

MV: A scalar multivariate to compute a reciprocal value for.

Returns: A rational scalar multivariate that is equal to the reciprocal value of MV.

Description: Given a scalar multivariate, returns a scalar multivariate representing the reciprocal values, by making it rational (if was not one) and flipping the numerator and the denominator.
See also: MvarMVDotProd, MvarMVVecDotProd, MvarMVScalarScale, MvarMVMultScalar, , MvarMVMult, MvarMVCrossProd,

8.2.381 MvarMVIsConstant (mvarbbox.c:206)

bbox
bounding box

```
int MvarMVIsConstant(const MvarMVStruct *MV, IrtRType Eps)
```

MV: To verify if constant or not..

Eps: Tolerance of equality allowed.

Returns: TRUE if indeed a constant(s) valued multi-variate.

Description: Checks if this MV is a constant multi-variate freeform function.

8.2.382 MvarMVIsMeshC0DiscontAt (mvar_gen.c:2170)

```
CagdBType MvarMVIsMeshC0DiscontAt(const MvarMVStruct *MV,
                                  int Dir,
                                  CagdRType t)
```

MV: Multivariate to examine for C0 discontinuity.

Dir: Parametric direction to examine at.

t: Parameter value to examine at.

Returns: TRUE if MV has a C0 discontinuity at parameter t in direction Dir, FALSE otherwise.

Description: Examines the mesh at given parametric location for a C0 discontinuity.
See also: MvarMVMeshC0Continuous, MvarMVKnotHasC0Discont,

8.2.383 MvarMVIsMeshC1DiscontAt (mvar_gen.c:2313)

```
CagdBType MvarMVIsMeshC1DiscontAt(const MvarMVStruct *MV,
                                  int Dir,
                                  CagdRType t)
```

MV: Multivariate to examine for C1 discontinuity.

Dir: Parametric direction to examine at.

t: Parameter value to examine at.

Returns: TRUE if MV has a C1 discontinuity at parameter t in direction Dir, FALSE otherwise.

Description: Examines the mesh at given parametric location for a C1 discontinuity.
See also: MvarMVMeshC1Continuous, MvarMVKnotHasC1Discont,

8.2.384 MvarMVKnotHasC0Discont (mvar_gen.c:2067)

```
CagdBType MvarMVKnotHasC0Discont(const MvarMVStruct *MV,
                                 int *Dir,
                                 CagdRType *t)
```

MV: The multivariate.

Dir: If a non-negative value, only that direction will be checked for C0 discontinuity. If a negative value, all directions in the domain will be searched for a C0 discont, if has one. The detected direction will be returned here if found.

t: The parameter at the C0 discont. if has one.

Returns: TRUE if found a C0 discont., FALSE otherwise.

Description: Searches the given MV for parameter locations that are C0 discont, based on the given knot vector of the MV. If found update Dir and t to this finding and returns TRUE.
See also: MvarMVMeshC0Continuous, MvarMVIsMeshC0DiscontAt,

8.2.385 MvarMVKnotHasC1Discont (mvar_gen.c:2207)

```
CagdBType MvarMVKnotHasC1Discont(const MvarMVStruct *MV,
                                 int *Dir,
                                 CagdRType *t)
```

MV: The multivariate.

Dir: If a non-negative value, only that direction will be checked for C0 discontinuity. If a negative value, all directions in the domain will be searched for a C0 discont, if has one. The detected direction will be returned here if found.

t: The parameter at the C1 discont. if has one.

Returns: TRUE if found a C1 discont., FALSE otherwise.

Description: Searches the given MV for parameter locations that are C1 discont, based on the given knot vector of the MV. If found update Dir and t to this finding and returns TRUE.
See also: MvarMVMeshC1Continuous, MvarMVIsMeshC1DiscontAt,

8.2.386 MvarMVListBBox (mvarbbox.c:237)

bbox

bounding box

```
void MvarMVListBBox(const MvarMVStruct *MVs, MvarBBoxStruct *BBox)
```

MVs: To compute a bounding box for.

BBox: Where bounding information is to be saved.

Returns: void

Description: Computes a bounding box for a list of multi-variate freeform function.

8.2.387 MvarMVListMatTransform (mvar_gen.c:1537)

scaling
rotation
translation
transformations

```
MvarMVStruct *MvarMVListMatTransform(const MvarMVStruct *MVs, CagdMType Mat)
```

MVs: To be transformed.

Mat: Defining the transformation.

Returns: Transformed MVs.

Description: Transforms the given list of MVs as specified by homogeneous matrix Mat.
See also: MvarMVMatTransform, MvarMVMatTransform2, CagdSrfListMatTransform,

8.2.388 MvarMVListPreciseBBox (mvarbbox.c:821)

bbox
bounding box

```
void MvarMVListPreciseBBox(const MvarMVStruct *MVs,
                           MvarBBoxStruct *BBox,
                           CagdRType Tol)
```

MVs: To compute a precise bounding box for.

BBox: Where bounding information is to be saved.

Tol: Tolerance of computations.

Returns: void

Description: Computes the precise bounding box for a list of multi-variate freeform function.
See also: MvarMVListBBox, MvarMVPreciseBBox,

8.2.389 MvarMVMatTransform (mvar_gen.c:1490)

multi-variates

```
MvarMVStruct *MvarMVMatTransform(const MvarMVStruct *MV, CagdMType Mat)
```

MV: Multi-variate to transform.

Mat: Homogeneous transformation to apply to MV.

Returns: Transformed MV.

Description: Transforms the given MV as specified by homogeneous matrix Mat.
See also: MvarMVListMatTransform, MvarMVMatTransform2, CagdSrfListMatTransform,

8.2.390 MvarMVMatTransform2 (mvar_gen.c:1567)

multi-variates

```
void MvarMVMatTransform2(MvarMVStruct *MV, CagdMType Mat)
```

MV: Multi-variate to transform.

Mat: Homogeneous transformation to apply to MV.

Returns: void

Description: Transforms, in place, the given MV as specified by homogeneous matrix Mat.
See also: MvarMVMatTransform, MvarMVListMatTransform, CagdSrfListMatTransform,

8.2.391 MvarMVMergeScalar (mvar_sym.c:1068)

merge
multivariates
symbolic computation

```
MvarMVStruct *MvarMVMergeScalar(MvarMVStruct * const *ScalarMVs)
```

ScalarMVs: A vector of scalar MVs. Location 0 holds the W or NULL otherwise, Location 1 holds the X axis and so on. This vector is assumed to have MVAR_MAX_PT_COORD coordinates.

Returns: A new multivariates constructed from given scalar multivariates.

Description: Given a set of scalar multivariates, treat them as coordinates into a new multivariates Assumes at least X axis not NULL when a scalar multivariate is returned. Assumes all axes are either E1 or P1 in which the weights are assumed to be identical and can be ignored if W axis exists or copied otherwise.
See also: MvarMVSplitScalar,

8.2.392 MvarMVMeshC0Continuous (mvar_gen.c:2121)

```
CagdBType MvarMVMeshC0Continuous(const MvarMVStruct *MV,
                                 int Dir,
                                 int Idx)
```

MV: To examine its potential discontinuity across Dir.

Dir: Direction to examine the discontinuity across.

Idx: Index where to examine the discontinuity.

Returns: TRUE if continuous there, FALSE otherwise.

Description: Examine the mesh of the given surface across direction Dir in index of mesh Index for a real discontinuity in the mesh. This index will typically be for a knot multiplicity potential discont.
See also: MvarMVKnotHasC0Discont, MvarMVIsMeshC0DiscontAt,

8.2.393 MvarMVMeshC1Continuous (mvar_gen.c:2260)

```
CagdBType MvarMVMeshC1Continuous(const MvarMVStruct *MV,
                                 int Dir,
                                 int Idx)
```

MV: To examine its potential discontinuity across Dir.

Dir: Direction to examine the discontinuity across.

Idx: Index where to examine the discontinuity.

Returns: TRUE if continuous there, FALSE otherwise.

Description: Examine the mesh of the given surface across direction Dir in index of mesh Index for a real discontinuity in the mesh. This index will typically be for a knot multiplicity potential discont.
See also: MvarMVKnotHasC1Discont, MvarMVIsMeshC1DiscontAt,

8.2.394 MvarMVMinMax (mvarbbox.c:71)

bbox
bounding box
minimum
maximum

```
void MvarMVMinMax(const MvarMVStruct *MV,
                  int Axis,
                  CagdRType *Min,
                  CagdRType *Max)
```

MV: To test for minimum/maximum.

Axis: 0 for W, 1 for X, 2 for Y etc.

Min: Where minimum found value should be place.

Max: Where maximum found value should be place.

Returns: void

Description: Computes a min max bound on a multivariate in a given axis. The multivariate is not coerced to anything and the given axis is tested directly where 0 is the W axis and 1, 2, 3 are the X, Y, Z etc.

8.2.395 MvarMVMult (mvar_sym.c:214)

product
symbolic computation
multivariates

```
MvarMVStruct *MvarMVMult(const MvarMVStruct *MV1, const MvarMVStruct *MV2)
```

MV1, MV2: Two multivariate to multiply coordinate-wise.

Returns: The product of MV1 * MV2 coordinate wise.

Description: Given two multivariates - multiply them coordinate-wise. The two multivariates are promoted to same point type before the multiplication can take place.
See also: MvarMVDotProd, MvarMVVecDotProd, MvarMVScalarScale, MvarMVMultScalar, , MvarMVInvert,

8.2.396 MvarMVMultBlend (mvar_sym.c:827)

```
MvarMVStruct *MvarMVMultBlend(const MvarMVStruct *MV1,
                              const MvarMVStruct *MV2,
                              CagdRType Blend)
```

MV1, MV2: Two multivariates to blend as "MV1 * Blend + MV2 * (1-Blend)".

Blend: Blending factor.

Returns: Blended multivariate.

Description: Blend two multivariates assumed to share a domain and point type.
See also:

8.2.397 MvarMVMultScalar (mvar_sym.c:433)

product
symbolic computation
multivariates

```
MvarMVStruct *MvarMVMultScalar(const MvarMVStruct *MV1, const MvarMVStruct *MV2)
```

MV1, MV2: Two multivariates to multiply. Note MV2 is a scalar field.

Returns: A multivariate representing the product of MV1 and MV2.

Description: Given two multivariate - a vector multivariate MV1 and a scalar multivariate MV2, multiply all MV1's coordinates by the scalar multivariate MV2. Returned multivariate is a multivariate representing the product of the two given multivariates.
See also: MvarMVDotProd, MvarMVVecDotProd, MvarMVMult, MvarMVCrossProd, , MvarCrvMultScalar,

8.2.398 MvarMVMultiLinearMV (mvarprim.c:29)

```
MvarMVStruct *MvarMVMultiLinearMV(const IrtRType *Min,
                                  const IrtRType *Max,
                                  int Dim)
```

Min: Minimal values of the expected ranges.

Max: Maximal values of the expected ranges.

Dim: Dimension of expect multivariate.

Returns: Constructed multi-linear function.

Description: Constructs a multi-linear multivariates that spans the [Min, Max] ranges in all Dim dimenions.

8.2.399 MvarMVNew (mvar_gen.c:61)

multi-variates
allocation

```
MvarMVStruct *MvarMVNew(int Dim,
                        MvarGeomType GType,
                        MvarPointType PType,
                        const int *Lengths)
```

Dim: Number of dimensions of this multi-variate.

GType: Type of geometry the curve should be - Bspline, Bezier etc.

PType: Type of control points (E2, P3, etc.).

Lengths: Of control mesh in each of the dimensions. Vector of size Dim.

Returns: An uninitialized freeform multi-variate.

Description: Allocates the memory required for a new multi-variate.
See also: MvarMVFree, MvarBzrMVNew, MvarBspMVNew,

8.2.400 MvarMVNormal2Cones (mvcones.c:1440)

normals

normal bound

```
MvarNormalConeStruct *MvarMVNormal2Cones(const MvarMVStruct *MV,
                                         CagdRType ExpandingFactor,
                                         int NumOfZeroMVs,
                                         MvarNormalConeStruct **Cone1,
                                         MvarNormalConeStruct **Cone2)
```

MV: To compute the normal 2cones for.

ExpandingFactor: Factor to expand placement of 2cones axes locations.

NumOfZeroMVs: Number of zero type MVs in the problem we solve.

Cone1, Cone2: The two cones to compute or ConeAngle = M_PI if error. Can be NULL in which case no 2cones are computed - only the regular cone is computed.

Returns: Regular normal cone if successful, NULL otherwise.

Description: Computes a 2cones bound to the normal field of multivariate MV. The 2cones bounds the normal field in the common intersection space. The 2cones are computed using the regular normal cone by expanding in the direction orthogonal to the cone axis and its main principal component. The expansion is done an amount that is equal to regular cone radius times ExpandingFactor.
See also: SymbNormalConeForSrf, MvarNormalConeOverlap,

8.2.401 MvarMVOpenEnd (mvar_aux.c:803)

open end conditions

```
MvarMVStruct *MvarMVOpenEnd(const MvarMVStruct *MV)
```

MV: To convert to a new multivar with open end conditions. Input can also be periodic.

Returns: Same multivar as MV but with open end conditions.

Description: Returns a multivar with open end conditions, similar to given multivar. Open end multivar is computed by extracting a subregion from multivar that is the entire multivar's parametric domain, by inserting multiple knots at the domain's boundary.
See also: BspCrvOpenEnd, BspSrfOpenEnd,

8.2.402 MvarMVOrthoCrvProjOnModel (mvarproj.c:236)

projection

```
MvarPolylineStruct *MvarMVOrthoCrvProjOnModel(const CagdCrvStruct *Crv,
                                              const MdlModelStruct *Mdl,
                                              CagdRType Tol,
                                              TrimSrfStruct **TSrfs)
```

Crv: The curve to project on Srf, orthogonally.

Mdl: The model to project Crv on.

Tol: Tolerance of the computation.

TSrfs: Optional pointer to return the trimmed surfaces the computed UV curves are on, if not NULL.

Returns: The projections in UV space of Srf.

Description: Computes the orthogonal projection of a curve C(t) on a model. computes the tensor product projection and then clip to the trimming zone. if TSrfs is not NULL, because the curves are returned in the UV space of the (trimmed) surfaces in the model, each such UV plln will have a "trimsrf" attribute that references the corresponding trimmed surfaces, that are returned in TSrfs.
See also: MvarMVOrthoIsoCrvProjOnSrf, MvarInverseCrvOnSrfProj, MvarMVOrthoCrvProjOnSrf, MvarMVOrthoCrvProjOnTrimSrf,

8.2.403 MvarMVOrthoCrvProjOnSrf (mvarproj.c:49)

projection

```
MvarPolylineStruct *MvarMVOrthoCrvProjOnSrf(const CagdCrvStruct *Crv,
                                            const CagdSrfStruct *Srf,
                                            CagdRType Tol)
```

Crv: The curve to project on Srf, orthogonally.

Srf: The surface to project Crv on.

Tol: Tolerance of the computation.

Returns: The projections in UV space of Srf.

Description: Computes the orthogonal projection of a curve C(t) on a surface S(u, v). That is, the projection is along the normal lines of the surface S.
Computed as the univariate solution to the following two equations:

$$< C(t) - S(u, v), \frac{dS}{du} > = 0,$$

$$< C(t) - S(u, v), \frac{dS}{dv} > = 0.$$

See also: MvarMVOrthoIsoCrvProjOnSrf, MvarInverseCrvOnSrfProj, , MvarMVOrthoCrvProjOnModel, MvarMVOrthoCrvProjOnTrimSrf,

8.2.404 MvarMVOrthoCrvProjOnTrimSrf (mvarproj.c:179)

projection

```
MvarPolylineStruct *MvarMVOrthoCrvProjOnTrimSrf(const CagdCrvStruct *Crv,
                                                const TrimSrfStruct *TSrf,
                                                CagdRType Tol)
```

Crv: The curve to project on Srf, orthogonally.

TSrf: The trimmed surface to project Crv on.

Tol: Tolerance of the computation.

Returns: The projections in UV space of Srf.

Description: Computes the orthogonal projection of a curve C(t) on a trimmed surface. computes the tensor product projection and then clip to the trimming zone.
See also: MvarMVOrthoIsoCrvProjOnSrf, MvarInverseCrvOnSrfProj, , MvarMVOrthoCrvProjOnModel, MvarMVOrthoCrvProjOnSrf,

8.2.405 MvarMVOrthoIsoCrvProjOnSrf (mvarproj.c:306)

projection

```
MvarPolylineStruct *MvarMVOrthoIsoCrvProjOnSrf(const CagdSrfStruct *Srf1,
                                               const CagdRType RVal,
                                               const CagdRType CrvT0,
                                               const CagdRType CrvT1,
                                               CagdSrfDirType Dir,
                                               const CagdSrfStruct *Srf2,
                                               CagdRType Tol)
```

Srf1: The surface to project from, orthogonally.

RVal: The isoparametric value of the curve of Srf1 to project.

CrvT0: Minimal parametric value of the curve region.

CrvT1: Maximal parametric value of the curve region.

Dir: Direction of isoparametric curve of S1.

Srf2: The surface to project to.

Tol: Computation tolerance.

Returns: The projections in UV space of Srf2.

Description: Computes the orthogonal projection of a region of an isoparametric curve of surface S1(r, t) at a fixed parameter value, RVal, into surface S2(u, v). The projection is along the normal lines of S1, that contains the curve.
Computed as the univariate solution to the following two equations:

```
                          dS1
< S2(u, v) - S1(r, t), ----- > |          = 0,
                          dr     | (r=RVal)
```

```
                          dS1
< S2(u, v) - S1(r, t), ----- > |          = 0.
                          dt     | (r=RVal)
```

See also: MvarMVOrthoCrvProjOnSrf,

8.2.406 MvarMVParamShift (mvar_rev.c:265)

multi-variates

```
MvarMVStruct *MvarMVParamShift(const MvarMVStruct *MV, int AxisSrc, int AxisTar)
```

MV: Multi-Variate in which the axes are to be reordered.

AxisSrc: The axis to move.

AxisTar: The required new index of AxisSrc.

Returns: Reordered multi-variate.

Description: Reorders the axes (or variables) of the given MV, so that AxisSrc moves to the index AxisTar. The order of all the other axes is preserved.
See also: MvarMVShiftAxes, MvarMVReverse,

8.2.407 MvarMVPreciseBBox (mvarbbox.c:547)

bbox

bounding box

```
MvarPtStruct *MvarMVPreciseBBox(const MvarMVStruct *MV,
                                MvarBBoxStruct *BBox,
                                CagdRType Tol)
```

MV: To compute a precise bounding box for.

BBox: Where bounding information is to be saved. If NULL, all computed extrema are returned instead.

Tol: Tolerance of computations.

Returns: If BBox is NULL, all extrema locations identified are returned here. Otherwise a NULL is returned.

Description: Computes the precise bounding box for a multi-variate freeform function. The precise bbox is computed by:

1. Deriving the MV with respect to all parameters, in all direction and seeking all the interior local extrema.
2. Recursively extract the boundaries as one-less dimensional MV and call recursively, until it is a curve, in which case its two end points are considered.

See also: MvarMVBBox,

8.2.408 MvarMVPrepGradient (mvar_der.c:702)

```
MvarMVGradientStruct *MvarMVPrepGradient(const MvarMVStruct *MV,
                                         CagdBType Orig)
```

MV: Input scalar field to compute its gradient function.

Orig: If orig TRUE and input is polynomial, the original scalar MV is also placed as last, additional, dimension (for faster evaluation of MV and its gradient).

Returns: The gradient function of the input scalar field.

Description: Builds a gradient for the given scalar multivariate. If the input is rational, returned is a dynamically allocated vector of scalar multivariate functions each representing DMV/Dui, i from 0 to Dim. The returned partial derivative are differentiated directly without the quotient rule which must be applied manually. Otherwise, if the input is polynomial, the gradient is returned as one vector function.
See also: MvarMVDerive, MvarMVFreeGradient, MvarMVEvalGradient,

8.2.409 MvarMVPwrDegreeRaise (mvarrais.c:568)

```
MvarMVStruct *MvarMVPwrDegreeRaise(const MvarMVStruct *MV,
                                   int Dir,
                                   int IncOrder)
```

MV: Multivariate to increase its order in direction Dir.

Dir: Direction to increase the order.

IncOrder: By how much to increase the order, at least one.

Returns: New multivariate with higher order.

Description: Increase the order of the given power basis multivariate in direction Dir by IncOrder amount. IncOrder amount is at least one.
See also: MvarMVDegreeRaise, MvarMVDegreeRaiseN, MvarMVDegreeRaise2,

8.2.410 MvarMVRefineAtParams (mvar_ref.c:42)

multi-variates

```
MvarMVStruct *MvarMVRefineAtParams(const MvarMVStruct *MV,
                                   MvarMVDirType Dir,
                                   CagdBType Replace,
                                   CagdRType *t,
                                   int n)
```

MV: Multi-variate to refine according to t in direction Dir.

Dir: Direction of refinement. Either U or V or W.

Replace: If TRUE t is a knot vector exactly in the length of the knot vector in direction Dir in MV and t simply replaces than knot vector. If FALSE, the knot vector in direction Dir in MV is refined by adding all the knots in t.

t: Knot vector to refine/replace the knot vector of MV in direction Dir.

n: Length of vector t.

Returns: The refined multi-variate. Always a Bspline.

Description: Given a multi-variate, refines it at the given n knots as defined by the vector t. If Replace is TRUE, the values replace the current knot vector. Returns pointer to refined MV (Note a Bezier multi-variate will be converted into a Bspline multi-variate).

8.2.411 MvarMVRegionFromMV (mvar_aux.c:663)

multi-variates

```
MvarMVStruct *MvarMVRegionFromMV(const MvarMVStruct *MV,
                                 CagdRType t1,
                                 CagdRType t2,
                                 MvarMVDirType Dir)
```

MV: To extract a sub-region from.

t1, t2: Domain to extract from MV, in parametric direction Dir.

Dir: Direction to extract the sub-region. Either U or V or W.

Returns: A sub-region of MV from t1 to t2 in direction Dir.

Description: Given a multi-variate, returns a sub-region of it.
See also: MvarMVExtension, MvarBzrMVRegionFromMV,

8.2.412 MvarMVReverse (mvar_rev.c:32)

multi-variates

```
MvarMVStruct *MvarMVReverse(const MvarMVStruct *MV, int Axis1, int Axis2)
```

MV: Multi-Variate to reverse.

Axis1, Axis2: Two axis to flip over.

Returns: Reversed multi-variate.

Description: Reverse the role of the given two axis by flipping them out.
See also: MvarPromoteMVToMV, MvarMVShiftAxes,

8.2.413 MvarMVReverseDir (mvar_rev.c:101)

multi-variates

```
MvarMVStruct *MvarMVReverseDir(const MvarMVStruct *MV, int Axis)
```

MV: Multi-Variate to reverse.

Axis: Direction to reverse.

Returns: Reversed multi-variate.

Description: Reverses the direction of the mesh of the given MV in direction Axis (and also reverse the knot vector if B-spline.
See also: MvarPromoteMVToMV, MvarMVShiftAxes, MvarMVReverse,

8.2.414 MvarMVRtnlMult (mvar_sym.c:787)

product

multivariates

symbolic computation

```
MvarMVStruct *MvarMVRtnlMult(const MvarMVStruct *MV1X,
                             const MvarMVStruct *MV1W,
                             const MvarMVStruct *MV2X,
                             const MvarMVStruct *MV2W,
                             CagdBType OperationAdd)
```

MV1X: Numerator of first multivariate.

MV1W: Denominator of first multivariate. Can be NULL.

MV2X: Numerator of second multivariate.

MV2W: Denominator of second multivariate. Can be NULL.

OperationAdd: TRUE for addition, FALSE for subtraction.

Returns: The result of MV1X MV2W +/- MV2X MV1W.

Description: Given two multivariates - multiply them using the quotient product rule:

```
X = X1 W2 +/- X2 W1
```

All provided multivariates are assumed to be non rational scalar multivariates. Returned is a non rational scalar multivariate (CAGD_PT_E1_TYPE).
See also: MvarMVDotProd, MvarMVVecDotProd, MvarMVScalarScale, MvarMVMultScalar, , MvarMVInvert, MvarMVCrossProd2D,

8.2.415 MvarMVScalarScale (mvar_sym.c:351)

scaling
symbolic computation
multivariates

```
MvarMVStruct *MvarMVScalarScale(const MvarMVStruct *CMV, CagdRType Scale)
```

CMV: A multivariate to scale by magnitude Scale.

Scale: Scaling factor.

Returns: A multivariates scaled by Scale compared to MV.

Description: Given a multivariate, scale it by Scale.
See also: MvarMVDotProd, MvarMVVecDotProd, MvarMVMult, MvarMVMultScalar, , MvarMVInvert, MvarMVCrossProd,

8.2.416 MvarMVScalarScale2 (mvar_sym.c:391)

scaling
symbolic computation
multivariates

```
MvarMVStruct *MvarMVScalarScale2(const MvarMVStruct *CMV,
                                 const CagdRType *Scale)
```

CMV: A multivariate to scale by magnitude Scale.

Scale: The scaling factors.

Returns: A multivariates scaled by Scale compared to MV.

Description: Given a multivariate, scale each of its coordinates by a given factor.
See also: MvarMVDotProd, MvarMVVecDotProd, MvarMVMult, MvarMVMultScalar, , MvarMVInvert, MvarMVCrossProd, MvarMVScalarScale,

8.2.417 MvarMVSetAllDomains (mvar_aux.c:430)

multi-variates

```
MvarMVStruct *MvarMVSetAllDomains(MvarMVStruct *MV,
                                  CagdRType *Min,
                                  CagdRType *Max,
                                  int InPlace)
```

MV: Multivariate function to update its domain.

Min: New minimum domains of MV.

Max: New maximum domains of MV.

InPlace: If TRUE, updates domain in place, unless was a Bezier that was converted into a Bspline, in which case the Bezier is released.

Returns: Same multivariate with the updated domain in dir Axis.

Description: Given a multi-variate, sets its parametric domain in all directions to be between Min and Max. If the MV is a Bezier, it is coerced to a Bspline first (and if InPlace TRUE, the original Bezier is freed).
See also: varMVDomain, MvarMVSetAllDomains, MvarParamInDomain, MvarParamsInDomain,

8.2.418 MvarMVSetCompositionCheckDomains (mvcompos.c:2020)

```
CagdBType MvarMVSetCompositionCheckDomains(CagdBType NewValue)
```

NewValue: The new value to set to the flag.

Returns: The previous value of the flag.

Description: Set the flag which determines if domain checks are performed in multivariate composition. If set to TRUE, the composition algorithm will verify that all of the Bezier patches of the reparametrization multivariate are fully contained (by bounding box) in the parametric domain of the target multivariate. If FALSE, then these checks are skipped.
See also: MvarMVCompose, MvarMVComposeBzrBzr, MvarComposeExtractTargetSubdomain,

8.2.419 MvarMVSetCompositionPropagateHigherDims (mvcompos.c:2052)

```
CagdBType MvarMVSetCompositionPropagateHigherDims(CagdBType NewValue)
```

NewValue: The new value to set to the flag.

Returns: The previous value of the flag.

Description: Set the flag which determines if tiles with higher dimensions (more than the deformation function's domain) will propagate their higher dimensions to the output or not.
See also: MvarMVCompose, MvarMVComposeBzrBzr, MvarComposeExtractTargetSubdomain, , MvarMVSetCompositionPropagateHigherDims,

8.2.420 MvarMVSetDomain (mvar_aux.c:207)

multi-variates

```
MvarMVStruct *MvarMVSetDomain(MvarMVStruct *MV,
                              CagdRType Min,
                              CagdRType Max,
                              int Axis,
                              int InPlace)
```

MV: Multivariate function to update its domain.

Min: New minimum domain in Axis direction of MV.

Max: New maximum domain in Axis direction of MV.

Axis: Axis to set a new domain for.

InPlace: If TRUE, updates domain in place if possible. A Bezier can be converted into a B-spline, in which case the Bezier is released.

Returns: Same multivariate with the updated domain in dir Axis.

Description: Given a multi-variate, sets its parametric domain in direction Axis to be between Min and Max. If the MV is a Bezier and the domain requested is not [0, 1], it is coerced to a B-spline first.
See also: varMVDomain, MvarMVSetAllDomains, MvarParamInDomain, MvarParamsInDomain,

8.2.421 MvarMVShiftAxes (mvar_rev.c:170)

multi-variates

```
MvarMVStruct *MvarMVShiftAxes(const MvarMVStruct *MV, int Axis)
```

MV: Multi-Variate to shift axes.

Axis: From where to shift forward until last Axis and put last Axis Here instead.

Returns: Multi-variate, with shifted axes

Description: Shift the last index in, instead of index Axis. All axes after Axis are shifted forward one location as well.
See also: MvarMVReverse, MvarPromoteMVToMV,

8.2.422 MvarMVSplitScalar (mvar_sym.c:1014)

split

multivariates

symbolic computation

```
void MvarMVSplitScalar(const MvarMVStruct *MV, MvarMVStruct **MVs)
```

MV: Multivariate to split.

MVs: An array to hold the newly dynamically allocated scalar MVs. The zero entry would hold the W, or NULL otherwise. The first entry would hold X axis, etc. This vector should have MVAR_MAX_PT_COORD coordinates.

Returns: void

Description: Given a multivariate, splits it to its scalar component multivariates.
See also: MvarMVMergeScalar,

8.2.423 MvarMVSub (mvar_sym.c:116)

subtraction

symbolic computation

multivariates

```
MvarMVStruct *MvarMVSub(const MvarMVStruct *MV1, const MvarMVStruct *MV2)
```

MV1, MV2: Two multivariate to subtract coordinate-wise.

Returns: The difference of MV1 - MV2 coordinate-wise.

Description: Given two multivariates - subtract them coordinate-wise. The two multivariates are promoted to same point type before the operation can take place. Furthermore, order and continuity are matched as well.
See also: MvarMVAdd, MvarMeshAddSub, MvarMVMult,

8.2.424 MvarMVSubdivAtParam (mvar_sub.c:48)

multi-variates

```
MvarMVStruct *MvarMVSubdivAtParam(const MvarMVStruct *MV,
                                  CagdRType t,
                                  MvarMVDirType Dir)
```

MV: Multi-Variate to subdivide.

t: Parameter to subdivide at.

Dir: Direction of subdivision.

Returns: A list of two multi-variates, result of the subdivision.

Description: Given a multi-variate, subdivides it at parameter value t in direction Dir.
See also: MvarBspMVSubdivAtParam, MvarBzrMVSubdivAtParam,

8.2.425 MvarMVSubdivAtParamOneSide (mvar_sub.c:431)

multi-variates

```
MvarMVStruct *MvarMVSubdivAtParamOneSide(const MvarMVStruct *MV,
                                         CagdRType t,
                                         MvarMVDirType Dir,
                                         IrtBType LeftSide)
```

MV: Multi-Variate to subdivide.

t: Parameter to subdivide at.

Dir: Direction of subdivision.

LeftSide: TRUE to only fetch left half, FALSE to fetch right half.

Returns: A single multi-variate, result of the subdivision.

Description: Given a multi-variate, subdivides it at parameter value t in direction Dir.
See also: MvarBspMVSubdivAtParamOneSide, MvarBzrMVSubdivAtParamOneSide, , MvarMVSubdivAtParam,

8.2.426 MvarMVTransform (mvar_gen.c:1455)

multi-variates

```
void MvarMVTransform(MvarMVStruct *MV, CagdRType *Translate, CagdRType Scale)
```

MV: Multi-variate to transform.

Translate: Translation factor. Can be NULl for non.

Scale: Scaling factor.

Returns: void

Description: Linearly transforms, in place, given MV as specified by Translate and Scale.

8.2.427 MvarMVTriTangentLine (mvtangnt.c:1327)

tri-tangent

```
CagdCrvStruct *MvarMVTriTangentLine(const CagdSrfStruct *Srf1,
                                    const CagdSrfStruct *Srf2,
                                    const CagdSrfStruct *Srf3,
                                    CagdRType StepSize,
                                    CagdRType SubdivTol,
                                    CagdRType NumericTol,
                                    int Euclidean)
```

Srf1, Srf2, Srf3: The 3 bivariates to compute the tri-tangent lines for. Assumed Bsplines surfaces with Open End Cond.

StepSize: Tolerance of numeric tracing of univariate solution.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol < SubdivTol.

Euclidean: True to return the result in Euclidean space, FALSE to return it in parameteric space.

Returns: Triplets of univariate solutions (three curves on the three surfaces) of tri-tangent lines.

Description: Computes the tri-tangent line of three freeform bivariate. In other words, computes the tangent line at three different points to the three surface(s). The result is a univariate (describing the line tri-tangent to the surfaces as it slides over the three surfaces).

See also: SymbTangentToCrvAtTwoPts, MvarMVBiTangents, MvarMVBiTangents2, MvarMVTriTangentLineCreateMVs,

8.2.428 MvarMVTriTangentLineCreateETs (mvtangnt.c:1074)

tri-tangent

```
void MvarMVTriTangentLineCreateETs(const MvarMVStruct *CMV1,
                                   const MvarMVStruct *CMV2,
                                   const MvarMVStruct *CMV3,
                                   MvarExprTreeStruct **ETs,
                                   MvarConstraintType *Constraints)
```

CMV1, CMV2, CMV3: The 3 bivariates to compute the tri-tangent lines for.

ETs: To be populated with the computed ETs constraints.

Constraints: To be populated with the constraints' types.

Returns: void

Description: Computes the constraints to solve for the tri-tangent line of three freeform bivariate. In other words, to compute the tangent line at three different points to the three surface(s). Let,

```
DMV12 = MV1(u, v) - MV2(s, t)
DMV13 = MV1(u, v) - MV3(a, b)
DMV23 = MV2(s, t) - MV3(a, b)
```

Then, compute the simultaneous solution of the following five equations:

$$< \frac{dMV1}{du} \times \frac{dMV1}{dv}, DMV13 > = 0,$$

$$< \frac{dMV2}{dr} \times \frac{dMV2}{ds}, DMV13 > = 0,$$

$$\frac{d\,MV3}{da} \times \frac{d\,MV3}{db} < DMV13 > = 0,$$

$$\frac{d\,MV1}{du} \times \frac{d\,MV1}{dv} < DMV12 > = 0,$$

$$\frac{d\,MV3}{da} \times \frac{d\,MV3}{db} < DMV23 > = 0,$$

See also: SymbTangentToCrvAtTwoPts, MvarMVBiTangents, MvarMVBiTangents2, MvarMVTriTangentLine, MvarMVTriTangentLineCreateMVs,

8.2.429 MvarMVTriTangentLineCreateMVs (mvtangnt.c:934)

tri-tangent

```
void MvarMVTriTangentLineCreateMVs(const MvarMVStruct *CMV1,
                                   const MvarMVStruct *CMV2,
                                   const MvarMVStruct *CMV3,
                                   MvarMVStruct **MVs,
                                   MvarConstraintType *Constraints)
```

CMV1, CMV2, CMV3: The 3 bivariates to compute the tri-tangent lines for.

MVs: To be populated with the computed MVS constraints.

Constraints: To be populated with the constraints' types.

Returns: void

Description: Computes the constraints to solve for the tri-tangent line of three freeform bivariate. In other words, to compute the tangent line at three different points to the three surface(s). Let,

```
DMV12 = MV1(u, v) - MV2(s, t)
DMV13 = MV1(u, v) - MV3(a, b)
DMV23 = MV2(s, t) - MV3(a, b)
```

Then, compute the simultaneous solution of the following five equations:

$$\frac{d\,MV1}{du} \times \frac{d\,MV1}{dv} < DMV13 > = 0,$$

$$\frac{d\,MV2}{dr} \times \frac{d\,MV2}{ds} < DMV13 > = 0,$$

$$\frac{d\,MV3}{da} \times \frac{d\,MV3}{db} < DMV13 > = 0,$$

```
  d MV1   d MV1
< ----- x -----, DMV12 > = 0,
    du      dv
```

```
  d MV3   d MV3
< ----- x -----, DMV23 > = 0,
    dx      dy
```

See also: SymbTangentToCrvAtTwoPts, MvarMVBiTangents, MvarMVBiTangents2, MvarMVTriTangentLine,

8.2.430 MvarMVTriTangents (mvtangnt.c:218)

tri-tangent

```
MvarPtStruct *MvarMVTriTangents(const MvarMVStruct *CMV1,
                                const MvarMVStruct *CMV2,
                                const MvarMVStruct *CMV3,
                                int Orientation,
                                CagdRType SubdivTol,
                                CagdRType NumericTol)
```

CMV1, CMV2, CMV3: The 3 multivariates to compute the tri-tangents for. If MV2 == MV3 == NULL, the self tri-tangents of MV1 are computed.

Orientation: 0 for no effect, -1 or +1 for a request to get opposite or similar normal orientation bi tangencies only.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol < SubdivTol.

Returns: Points on the bi-tangents of the two multivariates.

Description: Computes tri-tangents of freeform bivariate. In other words, compute the tangent plane at three points to the surface(s). Let,

```
DMV12 = MV1(u, v) - MV2(r, s)
DMV13 = MV1(u, v) - MV3(x, y)
DMV23 = MV2(r, s) - MV3(x, y)
```

then, compute the simultaneous solution of the following six equations:

```
  d MV1   d MV1
< ----- x -----, DMV12 > = 0,
    du      dv
```

```
  d MV1   d MV1
< ----- x -----, DMV13 > = 0,
    du      dv
```

```
  d MV2   d MV2
< ----- x -----, DMV23 > = 0,
    dr      ds
```

```
  d MV2   d MV2
< ----- x -----, DMV12 > = 0,
    dr      ds
```

```
  d MV3   d MV3
< ----- x -----, DMV13 > = 0,
    dx      dy


  d MV3   d MV3
< ----- x -----, DMV23 > = 0,
    dx      dy
```

See also: SymbTangentToCrvAtTwoPts, MvarMVBiTangents, MvarMVBiTangents2, MvarMVTriTangentLine,

8.2.431 MvarMVUnitMaxCoef (mvar_gen.c:1394)

```
MvarMVStruct *MvarMVUnitMaxCoef(MvarMVStruct *MV)
```

MV: Multivariate to normalize in place its coefficients.

Returns: Normalized multivariate, in place.

Description: Normalize in place the given multivariate so its maximal coefficient is of unit size. Each axis is treated independently, including W.
See also: MvarSrfUnitMaxCoef, MvarCrvUnitMaxCoef,

8.2.432 MvarMVUpdateConstDegDomains (mvar_aux.c:549)

```
void MvarMVUpdateConstDegDomains(MvarMVStruct **MVs, int NumOfMVs)
```

MVs: To update their domains.

NumOfMVs: Size of MVs vector.

Returns: void

Description: Given a vector of MVs, some with constant degrees and invalid domain, update all MVs domains, exploiting MVs with non constant degrees' domains.
See also: MvarETUpdateConstDegDomains,

8.2.433 MvarMVVecDotProd (mvar_sym.c:548)

product

dot product

symbolic computation

ltivariates

```
MvarMVStruct *MvarMVVecDotProd(const MvarMVStruct *MV, const CagdRType *Vec)
```

MV: Multivarients to multiply and compute a dot product for.

Vec: Vector to project MV onto.

Returns: A scalar multivariate representing the dot product of MV . Vec.

Description: Given a multivariate and a vector - computes their dot product. Returned multivariate is a scalar multivariate representing the dot product. While typically in R3, the dot product can be computed for any dimension of MV, and Vec should be of the appropriate size.
See also: MvarMVDotProd, MvarMVMult, MvarMVScalarScale, MvarMVMultScalar, , MvarMVInvert, MvarMVCrossProd,

8.2.434 MvarMVVolumeOfDomain (mvar_aux.c:112)

```
IrtRType MvarMVVolumeOfDomain(MvarMVStruct * const MVs, int Dim)
```

MVs: The multivariate.

Dim: Number of dimensions.

Returns: Volume of domain.

Description: Compute the volume of a multivariate's domain.

8.2.435 MvarMVZR1DMergeTwoPoly (zrmvaux1.c:85)

```
MvarPolylineStruct *MvarMVZR1DMergeTwoPoly(MvarPolylineStruct *Poly1,
                                           MvarPolylineStruct *Poly2,
                                           CagdRType Tol)
```

Poly1: 1st polyline.

Poly2: 2nd polyline.

Tol: The tolerance under which two points are proclaimed to be identical.

Returns: If the start/endpoint of Poly1 coincides with the start/endpoint of Poly2, up to Tol, they are merged.

Description: Merges two neighboring polylines together, in place.
See also: MvarMVZR1DPrelimLink,

8.2.436 MvarMVZR1DPrelimLink (zrmvaux1.c:279)

```
MvarPolylineStruct *MvarMVZR1DPrelimLink(MvarPolylineStruct **PolyList,
                                         MvarPolylineStruct *Poly,
                                         CagdRType Tol,
                                         const MvarMVStruct *MV)
```

PolyList: List of polylines (found poly is removed from, in place).

Poly: Polyline, to be merged with the one from the list.

Tol: The tolerance under which two points are proclaimed to be identical.

MV: Optional parameter that, when not NULL, defines the domain where no merges close to boundary will take place.

Returns: Polyline which is suitable for Poly to be merged, NULL, if no such polyline exist.

Description: Finds a polyline in the list, which is to be merged to a given polyline and remove it from the input list.
See also: MvarMVZR1DMergeTwoPoly,

8.2.437 MvarMVsBisector (mvbisect.c:45)

bisectors

```
MvarMVStruct *MvarMVsBisector(const MvarMVStruct *MV1, const MvarMVStruct *MV2)
```

MV1, MV2: The two multivariates to compute the bisector for.

Returns: The result bisector.

Description: Compute bisector to two given multivariates.
See also: SymbSrfPtBisectorSrf3D, SymbCrvPtBisectorSrf3D, SymbCrvCrvBisectorSrf3D,

8.2.438 MvarMVsDetectZeros (zrsolver.c:559)

```
CagdBType MvarMVsDetectZeros(MvarMVStruct * const *MVs,
                             MvarConstraintType *Constraints,
                             int NumOfMVs,
                             CagdRType SubdivTol,
                             CagdRType NumericTol,
                             int HighDimBndry)
```

MVs: Vector of multivariate constraints.

Constraints: Either an equality or an inequality type of constraint. Can be NULL in which case all constraints are equality.

NumOfMVs: Size of the MVs and Constraints vector.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

NumericTol: Numeric tolerance of the numeric stage. Measured in the image space of the constraints function.

HighDimBndry: TRUE if invoked when solving boundary of a higher dimensional problem, FALSE otherwise.

Returns: Whether or not a solution to the zero problem exists.

Description: Interface function for determining whether or not a MV equation has a solution. Constructs the generic problem structure, calls the solver in the detection-only mode. The set of NumOfMVs constraints may consist of equality or inequality constraints, as prescribed by the constraints vector. All multivariates are assumed to be in the same parametric domain size and dimension.
See also: MvarMVsZeros0D, MvarMVsZerosNormalConeTest, MvarMVsZerosDomainReduction, , MvarMVsZerosVerifier, MvarETsZeros0D, MvarMVsZeros1D, MvarMVsZeros2D,

8.2.439 MvarMVsSame (mvar_gen.c:1765)

```
CagdBType MvarMVsSame(const MvarMVStruct *MV1,
                      const MvarMVStruct *MV2,
                      CagdRType Eps)
```

MV1, MV2: The two multivariates to compare.

Eps: Tolerance of equality.

Returns: TRUE if multivariates are the same, FALSE otherwise.

Description: Compare the two multivariates for similarity.
See also: CagdSrfsSame, CagdCrvsSame, TrivTVsSame, MvarMVsSame3, MvarMVsSameSpace, , VMdlVModelsSame,

8.2.440 MvarMVsSame3 (mvar_gen.c:1808)

```
CagdBType MvarMVsSame3(const MvarMVStruct *MV1,
                       const MvarMVStruct *MV2,
                       CagdRType Eps,
                       int *Modified)
```

MV1, MV2: The two multivariates to compare.

Eps: Tolerance of equality.

Modified: Tags how MVs are reflected (See CagdSrfsSame3 and CagdCrvsSame for sub-MVs that are curves or surfaces). Can be NULL in which case it is ignored.

Returns: TRUE if multivariates are the same, FALSE otherwise.

Description: Compare the two multivariates for similarity. If the MVs are curves or surfaces, performs all possible reflections, searching for similarity, and bring them to common function spaces.
See also: CagdSrfsSame3, CagdCrvsSame3, TrivTVsSame, MvarMVsSame, MvarMVsSameSpace,

8.2.441 MvarMVsSameSpace (mvar_gen.c:1713)

```
CagdBType MvarMVsSameSpace(const MvarMVStruct *MV1,
                           const MvarMVStruct *MV2,
                           CagdRType Eps)
```

MV1, MV2: The two multivariates to compare.

Eps: Tolerance of equality.

Returns: TRUE if multivariates are in the same sapce, FALSE otehrwise.

Description: Compare the two multivariates to be in the same function space.
See also: MvarMVsSame,

8.2.442 MvarMVsZeros0D (zrsolver.c:90)

```
MvarPtStruct *MvarMVsZeros0D(MvarZeroPrblmSpecStruct *ZeroProblemSpec)
```

ZeroProblemSpec: Specification of the 0D problem, which includes the number of constraints, the multivariates constraints, constraints types, tolerances, call back functions, etc.

Returns: List of points on the solution set. Dimension of the points will be the same as the dimensions of all MVs.

Description: Interface function for a generic MV equation solver, 0D solutions: constructs the generic problem structure, calls the solver and extracts the solution point list from the generic solution structure. The set of NumOfMVs constraints may consist of equality or inequality constraints, as prescribed by the constraints vector. All multivariates are assumed to be in the same parametric domain size and dimension.
See also: MvarMVsZerosNormalConeTest, MvarMVsZerosDomainReduction, , MvarMVsZerosVerifier, MvarETsZeros0D, MvarMVsZeros1D, MvarMVsZeros2D,

8.2.443 MvarMVsZeros0DNumeric (zrsolver.c:221)

```
MvarPtStruct *MvarMVsZeros0DNumeric(MvarMVStruct * const *MVs,
                                    int NumOfMVs,
                                    CagdRType NumericTol,
                                    int SuccessOnDmnErr,
                                    MvarPtStruct *ZeroPt)
```

MVs: Vector of multivariate constraints.

NumOfMVs: Size of the MVs and Constraints vector.

NumericTol: Numeric tolerance of the numeric stage. Measured in the image space of the constraints function.

SuccessOnDmnErr: TRUE to successfully terminate this process when domain is within tolerance. FALSE to terminate based on range value.

ZeroPt: An initial guess. The closer this guess to the precise solution the better chance it will converge.

Returns: Preicse solution if successful or NULL if failed (the improved solution is not within NumericTol).

Description: Interface function for a numeric solution, as part of the MV equation solver, for 0D solutions. An initial guess must be provided and note that solution is not guaranteed, even if one exists.
See also: MvarMVsZerosNormalConeTest, MvarMVsZerosDomainReduction, , MvarMVsZerosVerifier, MvarETsZeros0D, MvarMVsZeros1D, MvarMVsZeros2D, , MvarZero0DNumeric,

8.2.444 MvarMVsZeros1D (zrsolver.c:338)

```
MvarPolylineStruct *MvarMVsZeros1D(MvarZeroPrblmSpecStruct *ZeroProblemSpec)
```

ZeroProblemSpec: Specification of the 1D problem, which includes the number of constraints, the multivariates constraints, constraints types, tolerances, call back functions, etc.

Returns: The solution, either holding list of polylines which approximate the intersection curve, or a list of polylines of length 1, which represent the critical points of the problem. In the intersection polyline case, each polyline corresponds to the topologically isolated component of the curve and is in R^k, the unioned parametric spaces of all input MVs.

Description: Interface function for a generic MV equation solver, 1D solutions: constructs the generic problem structure, calls the solver and extracts the list of solution polylines from the generic solution structure. The set of NumOfMVs constraints may consist of equality or inequality constraints, as prescribed by the constraints vector. All multivariates are assumed to be in the same parametric domain size and dimension.
See also: MvarSrfSrfInter, MvarMVsZeros1DMergeSingularPts, MvarMVsZeros0D, MvarETsZeros0D, MvarMVsZeros2D, MvarMVsZeros1DTrace2Pts,

8.2.445 MvarMVsZeros1DMergeSingularPts (zrmv1d.c:981)

```
int MvarMVsZeros1DMergeSingularPts(int MergeSingularPts)
```

MergeSingularPts: Set the desired state of singular points mergers.

Returns: Old state.

Description: Sets the state of the singular points merger: If 0, singular locations are ignored (skipped). If 1, singular locations are merged using the subdivision tolerance which improves the changes of a complete long merged curves. If 2, singular locations are merged using the numeric tolerances (like every other case) which means most likely they will be left as isolated points.
See also: MvarSrfSrfInter, MvarMVsZeros1D,

8.2.446 MvarMVsZeros1DTrace2Pts (zrsolver.c:415)

```
MvarPolylineStruct *MvarMVsZeros1DTrace2Pts(MvarMVStruct * const *MVs,
                                            MvarConstraintType *Constraints,
                                            int NumOfMVs,
                                            MvarPtStruct *StartEndPts,
                                            CagdRType Step,
                                            CagdRType SubdivTol,
                                            CagdRType NumericTol)
```

MVs: Vector of multivariate constraints.

Constraints: Either an equality or an inequality type of constraint. Can be NULL in which case all constraints are equality.

NumOfMVs: Size of the MVs and Constraints vector.

StartEndPts: Start/end points for the polyline solution to trace.

Step: Step size to use in the numeric tracing.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

NumericTol: Numeric tolerance of the numeric stage. Measured in the image space of the constraints function.

Returns: The traced polyline which approximates solution, in R^k.

Description: Interface function similar to MvarMVsZeros1D but only for tracing a single univariate component between StartEndPts.
See also: MvarSrfSrfInter, MvarMVsZeros1DMergeSingularPts, MvarMVsZeros0D, MvarETsZeros0D, MvarMVsZeros2D, MvarMVsZeros1D,

8.2.447 MvarMVsZeros2D (zrsolver.c:492)

```
MvarZeroSolutionStruct *MvarMVsZeros2D(MvarZeroPrblmSpecStruct *ZeroProblemSpec)
```

ZeroProblemSpec: Specification of the 2D problem, which includes the number of constraints, the multivariates constraints, constraints types, tolerances, call back functions, etc.

Returns: The list of either triangles or polylines, according to the required output types, which approximate the surface.

Description: Interface function for a generic MV equation solver, 2D solutions: constructs the generic problem structure, calls the solver and extracts the list of either solution triangles or solution polylines from the generic solution structure. The set of NumOfMVs constraints must consist of equality constraints, prescribed by the constraints vector (inequalities not supported). All multivariates are assumed to be in the same parametric domain size and dimension.
See also: MvarSrfSrfInter, MvarMVsZeros0D, MvarETsZeros0D, MvarMVsZeros1D,

8.2.448 MvarMVsZeros2DBy0D (zrsolver.c:255)

```
MvarPtStruct *MvarMVsZeros2DBy0D(MvarZeroPrblmSpecStruct *ZeroProblemSpec)
```

ZeroProblemSpec: Specification of the 0D problem, which includes the number of constraints, the multivariates constraints, constraints types, tolerances, call back functions, etc.

Returns: List of points on the solution set. Dimension of the points will be the same as the dimensions of all MVs.

Description: Interface function for a generic MV equation solver, 2D solutions, in the temporary case where the problem is solved using the 0D solver and then surface fitting to the solution points. Constructs the generic problem structure, calls the solver and extracts the solution point list from the generic solution structure. The surface fitting step is done by the specific problem algorithm outside the solver environment. The set of NumOfMVs constraints may consist of equality or inequality constraints, as prescribed by the constraints vector. All multivariates are assumed to be in the same parametric domain size and dimension.
See also: MvarCrvSrfBisectorApprox,

8.2.449 MvarMVsZeros2DCornersOnly (zrmv2dTs.c:1535)

```
int MvarMVsZeros2DCornersOnly(int Is2DSolutionCornersOnly)
```

Is2DSolutionCornersOnly: New setting for the filter state.

Returns: Old setting for the filter state.

Description: Sets the filter state of the polyline loops, when tagging loop points for later triangulation. If set to TRUE, the only chosen points are domain corners and points added due to T-Junction handling. Otherwise, intermediate solution points will be selected as well. NOTE: if set to TRUE, then the MVAR_ZRMV2D_LIMIT_EDGE_LEN flag must be set to FALSE. Otherwise black holes can appear in the mesh.
See also: MvarZeroSolverOrganizeSol2DMVs, MvarZeroTagSegmentForTr,

8.2.450 MvarMVsZeros2DPolylines (zrmv2dTp.c:708)

```
int MvarMVsZeros2DPolylines(int IsPolyLines2DSolution)
```

IsPolyLines2DSolution: New setting for the output type.

Returns: Old setting for the output type.

Description: Sets the output type of the bivariate solver: either a collection of loops (polylines) or a collection of triangles.
See also: MvarZeroSolverInseparableProblem, MvarZeroSolverOne2OneProjMVs,

8.2.451 MvarMVsZerosCrtPts (zrsolver.c:3116)

```
int MvarMVsZerosCrtPts(int CrtPtsDetection)
```

CrtPtsDetection: New setting for the critical points detection.

Returns: Old setting of the critical points detection.

Description: Controls the critical points detection task of the solver. If FALSE, the main problem is solved, disregarding the question of critical points. Otherwise, the main problem is not solved, but is converted to its critical points detection problem, and this is the problem solved.
See also: MvarZeroSolverInseparableProblem,

8.2.452 MvarMVsZerosDmnExt (zrsolver.c:2406)

```
CagdRType MvarMVsZerosDmnExt(CagdRType DmnExt)
```

DmnExt: New setting for domain extension usage.

Returns: Old extensions tolerance for domain extension usage.

Description: Sets the tolerance (or zero to disable) of the domain extension inside the multivariate subdivisions' zero set solver.
See also: MvarZeroSolverInseparableProblem, MvarMVsZerosDomainReduction, , MvarMVsZerosGradPreconditioning,

8.2.453 MvarMVsZerosDomainReduction (zrmvaux0.c:118)

```
int MvarMVsZerosDomainReduction(int DomainReduction)
```

DomainReduction: New setting for the domain reduction option.

Returns: Old setting for normal cone testing usage.

Description: Sets the use (or not) of the domain reduction option - Bezier (and (B-spline) clipping in multivariate subdivisions' zero set solver.
See also: MvarMVsZeros, MvarMVsZerosNormalConeTest, , MvarMVsZerosGradPreconditioning,

8.2.454 MvarMVsZerosGradPreconditioning (zrmvaux0.c:87)

```
int MvarMVsZerosGradPreconditioning(int GradPreconditioning)
```

GradPreconditioning: New setting for the gradient orthogonalization.

Returns: Old setting of gradient orthogonalization.

Description: Sets the use (or not) of gradient precondition option - application of and orthogonalization process over the gradients in multivariate subdivisions' zero set solver.
See also: MvarMVsZeros, MvarMVsZerosNormalConeTest, MvarMVsZerosDomainReduction, ,

8.2.455 MvarMVsZerosKantorovichTest (zrmvkant.c:794)

```
int MvarMVsZerosKantorovichTest(int KantorovichTest)
```

KantorovichTest: New setting for normal cone testing usage.

Returns: Old setting for normal cone testing usage.

Description: Sets the use (or not) of the Kantorovich test inside the multivariate subdivisions' zero set solver.
See also: MvarMVsZeros, MvarMVsZerosDomainReduction, , MvarMVsZerosGradPreconditioning,

8.2.456 MvarMVsZerosNormalConeTest (zrmvaux0.c:56)

```
int MvarMVsZerosNormalConeTest(int NormalConeTest)
```

NormalConeTest: New setting for normal cone testing usage.

Returns: Old setting for normal cone testing usage.

Description: Sets the use (or not) of the normal cone tests inside the multivariate subdivisions' zero set solver.
See also: MvarMVsZeros, MvarMVsZerosDomainReduction, , MvarMVsZerosGradPreconditioning,

8.2.457 MvarMVsZerosNormalizeConstraints (zrsolver.c:3085)

```
int MvarMVsZerosNormalizeConstraints(int NormalizeConstraints)
```

NormalizeConstraints: New setting for the constraints normalization.

Returns: Old setting of the constraints normalization.

Description: Controls the option of normalizing the constraints on every recursive call. Might be useful for constraints that are very steep on one side and shallow in another.
See also: MvarZeroSolverInseparableProblem, MvarMVsZerosCrtPts,

8.2.458 MvarMVsZerosParallelHyperPlaneTest (zrmvaux0.c:148)

```
int MvarMVsZerosParallelHyperPlaneTest(int ParallelHPlaneTest)
```

ParallelHPlaneTest: New setting for the domain reduction option.

Returns: Old setting for normal cone testing usage.

Description: Sets the use (or not) of the parallel plane termination criteria in multivariate subdivisions' zero set solver.
See also: MvarMVsZeros, MvarMVsZerosNormalConeTest, , MvarMVsZerosGradPreconditioning, MvarMVsZerosDomainReduction, ,

8.2.459 MvarMVsZerosSameSpace (zrmvaux0.c:178)

```
CagdBType MvarMVsZerosSameSpace(MvarMVStruct **MVs, int NumOfMVs)
```

MVs: Vector of multivariate constraints.

NumOfMVs: Size of the MVs vector.

Returns: TRUE if in same function space, FALSE otherwise.

Description: Make sure all given MVs are in the same function space.
See also: MvarMVsSameSpace,

8.2.460 MvarMVsZerosVerifier (zrmvaux0.c:935)

```
void MvarMVsZerosVerifier(MvarMVStruct * const  *MVs,
                          int NumOfZeroMVs,
                          MvarPtStruct *Sols,
                          CagdRType NumerEps)
```

MVs: Input constraints.

NumOfZeroMVs: Number of (zero only) constraints.

Sols: Linked lists of solutions found.

NumerEps: Numeric tolerance used in the solution.

Returns: void

Description: A verification function to test the correctness of the solutions. For mostly development/debugging purposes.
See also: MvarZeroSolver,

8.2.461 MvarMakeCrvsOnSrfsMatchSpeed (mvardist.c:1816)

```
int MvarMakeCrvsOnSrfsMatchSpeed(const CagdSrfStruct *Srf1,
                                 const CagdSrfStruct *Srf2,
                                 CagdCrvStruct **UVCrv1,
                                 CagdCrvStruct **UVCrv2)
```

Srf1: First surface to consider.

Srf2: Second surface to consider.

UVCrv1: First curve in the domain of Srf1 to modify, in place.

UVCrv2: Second curve in the domain of Srf2 to modify, in place.

Returns: TRUE if successful, FALSE otherwise.

Description: Given surfaces Srf1/2 and curves UVCrv1/2 in the domain of the surfaces so that the trace the same shape, recreate UVCrv1 and UVCrv2 so the move along the trace at the same speed, or Srf1(UVCrv1) = Srf2(UVCrv2), and the two curve are in the same function space.
See also: MvarProjUVCrvOnE3CrvMatchSpeed, TrimMatch2ndCrvLenSpeedAs1stCrv,

8.2.462 MvarMakeMVsCompatible (mvarcmpt.c:40)

compatibility

```
CagdBType MvarMakeMVsCompatible(MvarMVStruct **MV1,
                                MvarMVStruct **MV2,
                                CagdBType SameOrders,
                                CagdBType SameKVs)
```

MV1, MV2: Two surfaces to be made compatible, in place.

SameOrders: If TRUE, this routine make sure they share the same orders.

SameKVs: If TRUE, this routine make sure they share the same KVs.

Returns: TRUE if successful, FALSE otherwise.

Description: Given two multi-variates, makes them compatible by:

1. Coercing their point type to be the same.
2. Making them have the same multi-variate type.
3. Raising the degree of the lower one to be the same as the higher (If SameOrder TRUE).
4. Refining them to a common knot vector (If Bspline and SameOrder TRUE).

Note 3 is performed if SameOrder TRUE, 4 if SameKV TRUE. Both multi-variates are modified IN PLACE.
See also: MvarMakeMVsCompatible2, CagdMakeCrvsCompatible, CagdMakeSrfsCompatible,

8.2.463 MvarMakeMVsCompatible2 (mvarcmpt.c:90)

compatibility

```
CagdBType MvarMakeMVsCompatible2(MvarMVStruct **MV1,
                                 MvarMVStruct **MV2,
                                 CagdBType SameOrders,
                                 CagdBType SameKVs)
```

MV1, MV2: Two surfaces to be made compatible, in place.

SameOrders: If TRUE, this routine make sure they share the same orders.

SameKVs: If TRUE, this routine make sure they share the same KVs.

Returns: TRUE if successful, FALSE otherwise.

Description: Given two multi-variates, makes them compatible by:

1. Making them have the same multi-variate type.
2. Raising the degree of the lower one to be the same as the higher (If SameOrder TRUE).
3. Refining them to a common knot vector (If Bspline and SameOrder TRUE).

Note 2 is performed if SameOrder TRUE, 3 if SameKV TRUE. Both multi-variates are modified IN PLACE. Note here point type can be different and will stay different.
See also: MvarMakeMVsCompatible, CagdMakeCrvsCompatible, CagdMakeSrfsCompatible,

8.2.464 MvarMakeMVsOneDimCompatible (mvarcmpt.c:234)

compatibility

```
CagdBType MvarMakeMVsOneDimCompatible(MvarMVStruct **MV1,
                                      MvarMVStruct **MV2,
                                      int Dim,
                                      CagdBType SameOrders,
                                      CagdBType SameKVs)
```

MV1, MV2: Two surfaces to be made compatible, in place.

Dim: The dimension in which MV1 and MV2 are to be made compatible.

SameOrders: If TRUE, this routine make sure they share the same orders.

SameKVs: If TRUE, this routine make sure they share the same KVs.

Returns: TRUE if successful, FALSE otherwise.

Description: Given two multi-variates, makes one of their directions compatible by:

1. Making them have the same multi-variate type (Bezier or B-spline).
2. Raising the degree of the lower one to be the same as the higher in Dim (If SameOrder TRUE).
3. Refining them to a common knot vector in Dim (If Bspline and SameOrder TRUE).

Note 2 is performed if SameOrder TRUE, 3 if SameKV TRUE. Both multi-variates are modified IN PLACE. Note here point type can be different and will stay different.

See also: MvarMakeMVsCompatible, CagdMakeCrvsCompatible, CagdMakeSrfsCompatible,

8.2.465 MvarMakeUniquePointsList (mvarjimp.c:240)

unique

```
void MvarMakeUniquePointsList(MvarPtStruct **PtList, CagdRType Tol)
```

PtList: Input point list to make unique in place.

Tol: Equality tolerance on the different coefficient of the points.

Returns: void

Description: Removes duplicated points in a given multivariate points list, in place.

8.2.466 MvarMatchPointListIntoPolylines (mvar_pll.c:470)

```
MvarPolylineStruct *MvarMatchPointListIntoPolylines(const MvarPtStruct *PtsList,
                                                    IrtRType MaxTol)
```

PtsList: Point list to connect into multivariate polylines.

MaxTol: Maximum distance allowed to connect multivariate points.

Returns: Connected multivariate polylines, upto MaxTol tolerance.

Description: Connect the list of multivariate points into multivariate polylines by connecting the closest multivariate point pairs, until the distances between adjacent multivariate points/polylines is more than MaxTol.

See also: GMMatchPointListIntoPolylines2,

8.2.467 MvarMdlTrimSrfListPreciseBBox (mvarbbox.c:1132)

bbox

bounding box

```
void MvarMdlTrimSrfListPreciseBBox(const MdlTrimSrfStruct *TSrfs,
                                   MvarBBoxStruct *BBox,
                                   CagdRType Tol)
```

TSrfs: To compute a precise bounding box for.

BBox: Where bounding information is to be saved.

Tol: Tolerance of computations.

Returns: void

Description: Computes the precise bounding box for a list of mdl trimmed surfaces.

See also: MdlSrfListPreciseBBox, MdlTrimSrfPreciseBBox, CagdSrfListBBox, , MdlSrfPreciseBBox,

8.2.468 MvarMdlTrimSrfPreciseBBox (mvarbbox.c:1074)

bbox

bounding box

```
void MvarMdlTrimSrfPreciseBBox(const MdlTrimSrfStruct *TSrf,
                               MvarBBoxStruct *BBox,
                               CagdRType Tol)
```

TSrf: To compute a precise bounding box for.

BBox: Where bounding information is to be saved.

Tol: Tolerance of computations.

Returns: void

Description: Computes the precise bounding box for a model trimmed surface.
See also: MvarSrfPreciseBBox, MvarTrimSrfPreciseBBox, MVarMdlrimSrfListPreciseBBox, MdlMVPreciseBBox, CagdSrfBBox,

8.2.469 MvarMergeBBox (mvarbbox.c:273)

bbox

bounding box

```
void MvarMergeBBox(MvarBBoxStruct *DestBBox, const MvarBBoxStruct *SrcBBox)
```

DestBBox: One BBox operand as well as the result.

SrcBBox: Second BBox operand.

Returns: void

Description: Merges (union) two bounding boxes into one, in place.

8.2.470 MvarMergeMVList (mvar_aux.c:2202)

merge

```
MvarMVStruct *MvarMergeMVList(MvarMVStruct *MVList,
                              int Dir,
                              IrtRType Tolerance,
                              int InterpDiscont)
```

MVList: To connect into larger multivariates. Expected to be open ended. Merge is performed in-place - MVList is freed.

Dir: Axis direction to merge. First axis is 0.

Tolerance: To consider two shared boundaries the same.

InterpDiscont: If TRUE, linearly interpolate discontinuity.

Returns: The merged multivariates, or NULL if error.

Description: Merges a list of multivariates by connecting shared boundaries that are (almost) the same, in place. Uses the generic merging package in geom_lib.
See also: CagdMergeMVList, CagdMergeMVList3, GMMergeSameGeometry,

8.2.471 MvarMergeMVMV (mvar_aux.c:1578)

merge

multivariate

```
MvarMVStruct *MvarMergeMVMV(const MvarMVStruct *MV1,
                            const MvarMVStruct *MV2,
                            MvarMVDirType Dir,
                            CagdBType Discont)
```

MV1: To connect to CMV2's starting boundary at its end.

MV2: To connect to CMV1's end boundary at its start.

Dir: Direction the merge should take place.

Discont: If TRUE, assumes the merged "edge" is discontinuous.

Returns: The merged multivariate.

Description: Merges two multivariates in the requested direction Dir. It is assumed that last Sub-MV of MV1 is identical to first Sub-MV of MV2 in Dir. It is assumed that both MVs have open end conditions and share the same orders and knot sequences in all axes, but the merged axes which can have different knots.
See also: CagdMergeSrfSrf,

8.2.472 MvarMergeMVMV2 (mvar_aux.c:1924)

merge

multivariate

```
MvarMVStruct *MvarMergeMVMV2(const MvarMVStruct *CMV1,
                             const MvarMVStruct *CMV2,
                             MvarMVDirType Dir,
                             CagdBType Discont)
```

CMV1: To connect to CMV2's starting boundary at its end.

CMV2: To connect to CMV1's end boundary at its start.

Dir: Direction the merge should take place.

Discont: If TRUE, assumes the merged "edge" is discontinuous.

Returns: The merged multivariate, or NULL if failed.

Description: Merges two multivariates in the requested direction Dir. It is assumed that first/last Sub-MV of MV1 is identical to first/last Sub-MV of MV2, in Dir, up to all possible reflections on other directions. It is assumed that both MVs have open end conditions and share the same orders and knot sequences in all axes, but the merged axes which can have different knots.

See also: CagdMergeSrfSrf, MvarMergeMVMV, MvarAre2MVsPossiblySharingBndry, , MvarAre2MVsSharingBndry,

8.2.473 MvarMergeTVList (mvtrivar.c:1221)

merge

```
TrivTVStruct *MvarMergeTVList(const TrivTVStruct *TVList,
                              int Dir,
                              IrtRType Tolerance,
                              int InterpDiscont)
```

TVList: To connect into larger trivariates. Expected to be open ended.

Dir: Axis direction to merge. First axis is 0.

Tolerance: To consider two shared boundaries the same.

InterpDiscont: If TRUE, linearly interpolate discontinuity.

Returns: The merged trivariates, or NULL if error.

Description: Merges a list of trivariates by connecting shared boundaries that are (almost) the same, in place. Uses the generic merging package in geom_lib.

See also: CagdMergeMVList, MvarMergeMVList, GMMergeSameGeometry, TrivMergeTVTV,

8.2.474 MvarMergeTwoPointTypes (mvarcoer.c:153)

coercion

```
MvarPointType MvarMergeTwoPointTypes(MvarPointType PType1, MvarPointType PType2)
```

PType1, PType2: To point types to find the point type of their union.

Returns: A point type of the union of the spaces of PType1 and PType2.

Description: Returns a point type which spans the spaces of both two given point types.

8.2.475 MvarMeshIndicesFromIndex (mvar_aux.c:1205)

```
int MvarMeshIndicesFromIndex(int Index, const MvarMVStruct *MV, int *Indices)
```

Index: To decompose into the different axes of the multivariate.

MV: Whose indices are for.

Indices: To compute the exact point location in MV -> Points

Returns: TRUE if Index in range, false otherwise.

Description: Given a linear Index into the vector of control points, compute the indices of the multivariates in all dimensions.

See also: MvarGetPointsMeshIndices,

8.2.476 MvarMinSpanCirc (ms_circ.c:565)

minimum spanning circle

```
int MvarMinSpanCirc(IPObjectStruct *Objs,
                    CagdRType *Center,
                    CagdRType *Radius,
                    CagdRType SubdivTol,
                    CagdRType NumerTol)
```

Objs: The geometry to compute the MSC for as a list object.

Center: Of computed MSC.

Radius: Of computed MSC.

SubdivTol, NumerTol: Of computation.

Returns: TRUE if successful, FALSE otherwise.

Description: Minimum spanning circle (MSC) computation of given Objs geometry. Geometry could be freeform C^1 curves.
See also: MvarMSCircOfThreeCurves, MvarMSCircOfThreeCurves, MvarMinSpanCirc, , MVarMSCircCurveInCirc, GMMinSpanCirc,

8.2.477 MvarMinSpanCone (mvcones.c:325)

```
int MvarMinSpanCone(MvarVecStruct *MVVecs,
                    int VecsNormalized,
                    int NumOfVecs,
                    MvarNormalConeStruct *MVCone)
```

MVVecs: The set of vectors to compute their MSC.

VecsNormalized: TRUE if vectors are normalized, FALSE otherwise.

NumOfVecs: Number of vectors in set MVVecs.

MVCone: Returns cone axis and cone cos angle of computed MSC.

Returns: TRUE if successful, FALSE otherwise.

Description: Minimum spanning cone (MSC) computation of a set of vectors. Algorithm is based on the Minimum Spanning Circle in Section 4.7 of "Computational Geometry, Algorithms and Applications" by M. de Berg et. al.
See also: GMMinSpanCone,

8.2.478 MvarMinSpanConeAvg (mvcones.c:254)

```
int MvarMinSpanConeAvg(MvarVecStruct *MVVecs,
                       int VecsNormalized,
                       int NumOfVecs,
                       MvarNormalConeStruct *MVCone)
```

MVVecs: The set of vectors to compute their MSC.

VecsNormalized: TRUE if vectors are normalized, FALSE otherwise.

NumOfVecs: Number of vectors in set MVVecs.

MVCone: Returns cone axis and cone cos angle of computed MSC.

Returns: TRUE if successful, FALSE otherwise.

Description: Minimum spanning cone (MSC) computation of a set of vectors. Find a central vector as the average of all given vectors and find the vector with maximal angular distance from it.
See also: GMMinSpanConeAvg,

8.2.479 MvarNormalConeCopy (mvcones.c:125)

```
MvarNormalConeStruct *MvarNormalConeCopy(const MvarNormalConeStruct
                                                                    *NormalCone)
```

NormalCone: Normal cone to copy.

Returns: Copied normal cone.

Description: Copy a multivariate normal cone structure.
See also: MvarNormalConeNew, MvarNormalConeCopyList, MvarNormalConeFree,

8.2.480 MvarNormalConeCopyList (mvcones.c:161)

multi-variates

```
MvarNormalConeStruct *MvarNormalConeCopyList(const MvarNormalConeStruct
                                                                    *NormalCones)
```

NormalCones: List of multivariate normal cones to duplicate.

Returns: Duplicated list of multivariate normal cones.

Description: Copy a list of multivariate normal cones' structures.
See also: MvarNormalConeNew, MvarNormalConeCopy, MvarNormalConeFree,

8.2.481 MvarNormalConeFree (mvcones.c:193)

```
void MvarNormalConeFree(MvarNormalConeStruct *NormalCone)
```

NormalCone: Normal cone to free.

Returns: void

Description: Free a multivariate normal cone structure.
See also: MvarNormalConeNew, MvarNormalConeFreeList, MvarNormalConeCopy,

8.2.482 MvarNormalConeFreeList (mvcones.c:219)

```
void MvarNormalConeFreeList(MvarNormalConeStruct *NormalConeList)
```

NormalConeList: Multi-Variate cone list to free.

Returns: void

Description: Deallocates and frees a list of multi-variate cone structures.
See also: MvarNormalConeFree, MvarNormalConeNew,

8.2.483 MvarNormalConeNew (mvcones.c:93)

```
MvarNormalConeStruct *MvarNormalConeNew(int Dim)
```

Dim: Dimension of the cone.

Returns: Constructed cone.

Description: Constructs a multivariate normal cone structure.
See also: MvarNormalConeFree, MvarNormalConeFreeList, MvarNormalConeCopy,

8.2.484 MvarNumericImporveSharedPoints (mvardist.c:1661)

```
int MvarNumericImporveSharedPoints(const CagdSrfStruct *Srf1,
                                   void *DistSrfPointPreps1,
                                   CagdRType *UV1,
                                   const CagdSrfStruct *Srf2,
                                   void *DistSrfPointPreps2,
                                   CagdRType *UV2)
```

Srf1: Surface holding UV1 in its domain.

DistSrfPointPreps1: Point projection on Srf1 prep struct.

UV1: First surface location (on Srf1).

Srf2: Surface holding UV2 in its domain.

DistSrfPointPreps2: Point projection on Srf1 prep struct.

UV2: Second surface location (on Srf2).

Returns: TRUE if improved, FALSE otherwise.

Description: Numerically improve UV1 and UV2, in Srf1 and Srf2 domains, respectively, so that Srf1(UV1) = Srf2(UV2).
See also:

8.2.485 MvarParamInDomain (mvar_aux.c:472)

multi-variates

```
CagdBType MvarParamInDomain(const MvarMVStruct *MV,
                            CagdRType t,
                            MvarMVDirType Dir)
```

MV: To make sure t is in its Dir domain.

t: Parameter value to verify.

Dir: Direction. Either U or V or W.

Returns: TRUE if in domain, FALSE otherwise.

Description: Given a multi-variate and a domain - validate it.
See also: varMVSetDomain, MvarMVSetAllDomains, MvarParamInDomain,

8.2.486 MvarParamsInDomain (mvar_aux.c:500)

multi-variates

```
CagdBType MvarParamsInDomain(const MvarMVStruct *MV, const CagdRType *Params)
```

MV: To make sure (u, v, w) is in its domain.

Params: Array of real valued parameters of size Dim to verify if this point is in MV's parametric domain.

Returns: TRUE if in domain, FALSE otherwise.

Description: Given a multi-variate and a domain - validate it.
See also: varMVSetDomain, MvarMVSetAllDomains, MvarParamsInDomain, varMVIntersPtOnBndry,

8.2.487 MvarPiecewiseDvlpAlgApproxLineAnalyze (pdvl_alg.c:108)

```
CagdSrfStruct *MvarPiecewiseDvlpAlgApproxLineAnalyze(
                                          const CagdSrfStruct *Srf,
                                          CagdRType Tolerance,
                                          CagdCrvStruct **StripBoundriesUV,
                                          int CrvSizeReduction,
                                          CagdRType SubdivTol,
                                          CagdRType NumericTol,
                                          CagdRType SrfExtent,
                                          int DvlpSteps)
```

Srf: To compute piecewise developable approximation for.

Tolerance: Maximal allowed distance from Srf to developables.

StripBoundriesUV: If points to a curve, used as first curve to develop from. This curve is considered a UV curve in Srf. Otherwise, will be initialized with VMin curve. Will be updated with the list of UV curves in the parametric domain of Srf that delineates the boundaries of the developable strips.

CrvSizeReduction: A reduction in size of traced curve while ensuring the Tolerance, conservatively.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol $<$ SubdivTol.

SrfExtent: Amount to extend the surface so the developable sheet will span it all. Zero to ignore.

DvlpSteps: Number of double-strip constructions to perform from the VMin boundary.

Returns: List of piecewise developables approximation of Srf to within Tolerance.

Description: Computes piecewise developable strips to Srf, within Tolerance. That is, every location on Srf is within tolerance to at least one developable surface position. Computation is performed starting from the VMin boundary of Srf.

See also: MvarPiecewiseDvlpAlgApproxBuildDvlpSrfs, SymbSrfDevelopableCrvOnSrf, MvarPiecewiseRuledAlgApproxBuildRuledSrfs, MvarPiecewiseRuledAlgApproxLineAnalyze,

8.2.488 MvarPiecewiseRuledAlgApproxBuildRuledSrfs (prld_alg.c:862)

```
IPObjectStruct *MvarPiecewiseRuledAlgApproxBuildRuledSrfs(
                                                const IPObjectStruct *Srf,
                                                const IPObjectStruct *UVRld)
```

Srf: Input surfaces. Can be either a regular or a trimmed surface.

UVRld: The computed ruled surfaces approximation for Srf, as UV curves.

Returns: A list of ruled surface depicting the result. If input Srf is trimmed, will be a set of trimmed surfaces as well.

Description: Build ruled surfaces from the extracted UV curves.

See also: MvarPiecewiseRuledAlgApproxLineAnalyze, MvarPiecewiseDvlpAlgApproxLineAnalyze, MvarPiecewiseDvlpAlgApproxBuildDvlpSrfs,

8.2.489 MvarPiecewiseRuledAlgApproxLineAnalyze (prld_alg.c:97)

```
CagdCrvStruct *MvarPiecewiseRuledAlgApproxLineAnalyze(
                                              const CagdSrfStruct *Srf,
                                              CagdRType Tolerance,
                                              CagdCrvStruct **StripBoundriesUV,
                                              int CrvSizeReduction,
                                              CagdRType SubdivTol,
                                              CagdRType NumericTol)
```

Srf: To compute piecewise ruling approximation for.

Tolerance: Maximal allowed distance between Srf and rulings.

StripBoundriesUV: List of UV curves in the parametric domain of Srf that delineates the boundaries of the strips.

CrvSizeReduction: A reduction in size of traced curve while ensuring the Tolerance conservatively.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol $<$ SubdivTol.

Returns: List of UV curves in the parametric domain of Srf that define the piecewise ruling approximation of Srf to within Tolerance.

Description: Computes piecewise ruled strips to Srf that can, for example be used for gouging-free flanked milling, within Tolerance. That is, every location on Srf is within tolerance to at least one ruled surface position. Note we assume the tool radius is zero for flanked milling application, as surface can be offset by tool radius, reducing tool to a line. Also computation is performed from VMin boundary of Srf.

See also: MvarPiecewiseRuledAlgApproxBuildRuledSrfs, MvarPiecewiseDvlpAlgApproxLineAnalyze, MvarPiecewiseDvlpAlgApproxBuildDvlpSrfs,

8.2.490 MvarPlaneCopy (mvar_gen.c:1284)

```
MvarPlaneStruct *MvarPlaneCopy(const MvarPlaneStruct *Pln)
```

Pln: Multi-Variate plane to duplicate.

Returns: Duplicated multi-variate plane.

Description: Allocates and duplicates all slots of a multi-variate plane structure.

8.2.491 MvarPlaneCopyList (mvar_gen.c:1310)

multi-variates

```
MvarPlaneStruct *MvarPlaneCopyList(const MvarPlaneStruct *PlnList)
```

PlnList: List of multi-variates to duplicate.

Returns: Duplicated list of multi-variates.

Description: Duplicates a list of multi-variate structures.

8.2.492 MvarPlaneFree (mvar_gen.c:1342)

```
void MvarPlaneFree(MvarPlaneStruct *Pln)
```

Pln: Multivariate plane to free.

Returns: void

Description: Deallocates and frees all slots of a multi-variate plane structure.
See also: MvarPlaneNew,

8.2.493 MvarPlaneFreeList (mvar_gen.c:1366)

```
void MvarPlaneFreeList(MvarPlaneStruct *PlnList)
```

PlnList: Multi-Variate plane list to free.

Returns: void

Description: Deallocates and frees a list of multi-variate plane structures.

8.2.494 MvarPlaneNew (mvar_gen.c:1250)

```
MvarPlaneStruct *MvarPlaneNew(int Dim)
```

Dim: Number of dimensions of this multi-variate.

Returns: An uninitialized freeform multi-variate plane.

Description: Allocates the memory required for a new multi-variate plane.
See also: MvarPlaneFree,

8.2.495 MvarPlaneNormalize (mvar_vec.c:670)

```
int MvarPlaneNormalize(MvarPlaneStruct *Pln)
```

Pln: Plane to normalize its normal direction.

Returns: TRUE if successful, FALSE if the input is the ZERO vector.

Description: Normalize a given multivariate plane's normal direction to a unit length, in place.
See also: MvarVecNormalize,

8.2.496 MvarPlnrKrnlBooleanSumSrf (krnl_based_cnstrct.c:928)

```
MvarPlnrKrnlSrfResStruct *MvarPlnrKrnlBooleanSumSrf(
                                          MvarPlnrKrnlSrfInptStruct *Inpt)
```

Inpt: The input of the kernel based Boolean sum operator.

Returns: The result will be holded in a MvarPlnrKrnlSrfResStruct struct.

Description: Constructs a planar Kernel Boolean sum surface using the four provided boundary curves. Curve's end points must meet at the four corners. First the surface is constructed by the regular Boolean sum. Second, re-ordering the surface's inner control points into a uniform grid. Third, each inner control point is translated by DistRaio * |KernelPt - Cp|, where kernelPt is the centroid of the kernel loci of the surface, and Cp is the location of the control point. Fourth, re-center the inner control points grid around the kernel center. The function aims to build a surface with a valid parameterization. If the function failed, it will add dofs to the input curves by degree raising or uniform knot insertion.
See also: MvarPlnrKrnlOneSidedBSumSrf, MvarPlnrKrnlRuledSrf,

8.2.497 MvarPlnrKrnlOneSidedBSumSrf (krnl_based_cnstrct.c:1143)

```
MvarPlnrKrnlSrfResStruct *MvarPlnrKrnlOneSidedBSumSrf(
                                       MvarPlnrKrnlSrfInptStruct *Inpt)
```

Inpt: The input of the kernel based one sided Boolean sum operator.

Returns: The result will be holded in a MvarPlnrKrnlSrfResStruct struct.

Description: Constructs a planar Kernel sided Boolean sum surface using the two provided boundary curves. The Curves must meet at a corner point. First the surface is constructed by the regular sided Boolean sum operator. Second, re-ordering the surface's inner control points into a unifrom grid. Third, each inner control point is translated by DistRaio * |KernelPt - Cp|, where kernelPt is the centroid of the kernel loci of the surface, and Cp is the location of the control point. Fourth, re-center the inner control points grid around the kernel center. The function aims to build a surface with a valid parameterization. If the function failed, it will add dofs to the input curves by degree raising or uniform knot insertion.
See also: MvarPlnrKrnlBooleanSumSrf, MvarPlnrKrnlRuledSrf,

8.2.498 MvarPlnrKrnlRuledSrf (krnl_based_cnstrct.c:1043)

```
MvarPlnrKrnlSrfResStruct *MvarPlnrKrnlRuledSrf(
                                       MvarPlnrKrnlSrfInptStruct *Inpt)
```

Inpt: The input of the kernel based ruling operator.

Returns: The result will be holded in a MvarPlnrKrnlSrfResStruct struct.

Description: Constructs a planar Kernel ruled surface which formed between the two provided boundary curves. First the surface is constructed by the regular ruling operator. Second, re-ordering the surface's inner control points into a unifrom grid. Third, each inner control point translated by DistRaio * |KernelPt - Cp|, where kernelPt is the centroid of the kernel loci of the surface, and Cp is the location of the control point. Fourth, re-center the inner control points grid around the kernel center. The function aims to build a surface with a valid parameterization. If the function failed, it will add dofs to the input curves by degree raising or uniform knot insertion.
See also: MvarPlnrKrnlBooleanSumSrf, MvarPlnrKrnlOneSidedBSumSrf,

8.2.499 MvarPlnrKrnlSrfInptStructAlloc (krnl_based_cnstrct.c:1234)

```
MvarPlnrKrnlSrfInptStruct *MvarPlnrKrnlSrfInptStructAlloc()
```

Returns: Pointer to the allocated object.

Description: Allocates and initialize a new object of MvarPlnrKrnlSrfInptStruct.
See also: MvarPlnrKrnlBooleanSumSrf, MvarPlnrKrnlOneSidedBSumSrf, , MvarPlnrKrnlRuledSrf, MvarPlnrKrnlSrfInptStructFree,

8.2.500 MvarPlnrKrnlSrfInptStructFree (krnl_based_cnstrct.c:1263)

```
void MvarPlnrKrnlSrfInptStructFree(
                            MvarPlnrKrnlSrfInptStruct *KrnlInpt)
```

KrnlInpt: A pointer to a MvarPlnrKrnlSrfInptStruct object.

Returns: void

Description: Free from memory the given MvarPlnrKrnlSrfInptStruct object and its all interior variables.
See also: MvarPlnrKrnlBooleanSumSrf, MvarPlnrKrnlOneSidedBSumSrf, , MvarPlnrKrnlRuledSrf, MvarPlnrKrnlSrfInptStructAlloc,

8.2.501 MvarPlnrKrnlSrfResStructFree (krnl_based_cnstrct.c:871)

```
void MvarPlnrKrnlSrfResStructFree(MvarPlnrKrnlSrfResStruct *Res)
```

Res: A object to free.

Returns: void

Description: Free the given MvarPlnrKrnlSrfResStruct object.
See also:

8.2.502 MvarPointFromPointLine (mvar_int.c:198)

point line distance

```
void MvarPointFromPointLine(const MvarVecStruct *Point,
                            const MvarVecStruct *Pl,
                            const MvarVecStruct *Vl,
                            MvarVecStruct *ClosestPoint)
```

Point: To find the closest to on the line.

Pl, Vl: Position and direction that defines the line. Vl is assumed a unit length vector.

ClosestPoint: Where closest point found on the line is to be saved.

Returns: void

Description: Routine to compute the closest point on a line to point, in R^n. The line is prescribed using a point on it (Pl) and a unit vector (Vl).
See also: GMPointFromPointLine, ,

8.2.503 MvarPolyMergePolylines (mvar_pll.c:220)

merge

polyline

```
MvarPolylineStruct *MvarPolyMergePolylines(MvarPolylineStruct *Polys,
                                           IrtRType Eps)
```

Polys: Multivariate polylines to merge, in place.

Eps: Epsilon of similarity to merge multivariate polylines at.

Returns: Merged as possible multivariate polylines.

Description: Merges separated multivariate polylines into longer ones, in place, as possible. Given a list of multivariate polylines, matches end points and merged as possible multivariate polylines with common end points, in place.
See also: GMMergePolylines,

8.2.504 MvarPolyReverseList (mvar_gen.c:741)

reverse

```
MvarPtStruct *MvarPolyReverseList(MvarPtStruct *Pts)
```

Pts: Multi-Variate point list to reverse.

Returns: Reversed list of Multi-Variate points, in place.

Description: Reverses a list of multivariate points, in place.

8.2.505 MvarPolylineCopy (mvar_gen.c:801)

```
MvarPolylineStruct *MvarPolylineCopy(const MvarPolylineStruct *Poly)
```

Poly: Multi-Variate polyline to duplicate.

Returns: Duplicated multi-variate polyline.

Description: Allocates and duplicates all slots of a multi-variate polyline structure.

8.2.506 MvarPolylineCopyList (mvar_gen.c:824)

multi-variates

```
MvarPolylineStruct *MvarPolylineCopyList(MvarPolylineStruct *PolyList)
```

PolyList: List of multi-variate polylines to duplicate.

Returns: Duplicated list of multi-variate polylines.

Description: Duplicates a list of multi-variate polyline structures.

8.2.507 MvarPolylineFree (mvar_gen.c:931)

```
void MvarPolylineFree(MvarPolylineStruct *Poly)
```

Poly: Multivariate polyline to free.

Returns: void

Description: Deallocates and frees all slots of a multi-variate polyline structure.
See also: MvarPolylineNew,

8.2.508 MvarPolylineFreeList (mvar_gen.c:951)

```
void MvarPolylineFreeList(MvarPolylineStruct *PolyList)
```

PolyList: Multi-Variate polyline list to free.

Returns: void

Description: Deallocates and frees a list of multi-variate polyline structures.

8.2.509 MvarPolylineNew (mvar_gen.c:775)

```
MvarPolylineStruct *MvarPolylineNew(MvarPtStruct *Pl)
```

Pl: List of points forming the polyline.

Returns: A new multi-variate polyline.

Description: Allocates the memory required for a new multi-variate polyline.
See also: MvarPolylineFree,

8.2.510 MvarProjCrvOnSrf (mvardist.c:1501)

```
CagdCrvStruct *MvarProjCrvOnSrf(const CagdCrvStruct *Crv,
                                const CagdSrfStruct *Srf,
                                CagdRType Step,
                                CagdRType SubdivTol,
                                CagdRType NumericTol,
                                void *SrfPrepHandle,
                                CagdBType GenUVCrv)
```

Crv: E3 Curve to check if precisely contained inside Srf.

Srf: Surface to check if Crv is precisely contained in it.

Step: Distance (step) in direction of tangent vec., while tracing.

SubdivTol: The subdivision tolerance to use.

NumericTol: The numerical tolerance to use.

SrfPrepHandle: Hold pre-processed data by MvarInverseCrvOnSrfProjPrep to speed up the crv on srf projections. See MvarInverseCrvOnSrfProjPrep/MvarInverseCrvOnSrfProjFree. Optional and can be NULL.

GenUVCrv: If TRUE, a UV curve on the surface is generated and returned, if precisely on Srf. If FALSE, and precisely on Srf, Crv is returned.

Returns: The (UV) curve representation of Crv that is found to be precisely on Srf or NULL otherwise.

Description: Checks if given curve is precisely contained in the given surface. returns a UV curve in Srf that represent it if indeed on Srf. Computation is done by sampling points on the curve and projecting these points on the surface.

See also: MvarMVDistCrvSrf, SymbSrfDistCrvCrv, SymbSrfDistFindPoints, , MvarMVDistSrfSrf, MvarMVDistCrvSrf, MvarProjCrvOnSrf1, , MvarProjCrvOnSrf2, MvarSrfSrfInter, MvarInverseCrvOnSrfProj,

8.2.511 MvarProjCrvOnSrf1 (mvardist.c:1345)

curve surface distance

```
MvarPolylineStruct *MvarProjCrvOnSrf1(const CagdCrvStruct *Crv1,
                                      const CagdSrfStruct *Srf2,
                                      int Axes,
                                      CagdRType Step,
                                      CagdRType SubdivTol,
                                      CagdRType NumericTol)
```

Crv1, Srf2: The two entities, Crv1(t) and Srf2(u, v), to find their common domains.

Axes: One of 12, 13, 23, 123 for XY, XZ, YZ, XYZ solution.

Step: Distance (step) in direction of tangent vec., while tracing.

SubdivTol: The subdivision tolerance to use.

NumericTol: The numerical tolerance to use.

Returns: Location where distance function square d2(t, u, v) of Crv1(t) and Srf2(u, v) is zero, as (t, u, v) tuples.

Description: Checks if given curve is precisely (partially) contained in the given surface and returns the portion that is indeed on Srf, if any. Computation is done by computing the distance field as a trivariate and finding the zeros of two out of three components of this distance field.

See also: MvarMVDistCrvSrf, SymbSrfDistCrvCrv, SymbSrfDistFindPoints, , MvarMVDistSrfSrf, MvarMVDistCrvSrf, MvarProjCrvOnSrf, MvarProjCrvOnSrf2, , MvarCrvSrfInter, MvarSrfSrfInter,

8.2.512 MvarProjCrvOnSrf2 (mvardist.c:1436)

curve surface distance

```
MvarPolylineStruct *MvarProjCrvOnSrf2(const CagdCrvStruct *Crv1,
                                      const CagdSrfStruct *Srf2,
                                      CagdRType Step,
                                      CagdRType SubdivTol,
                                      CagdRType NumericTol)
```

Crv1, Srf2: The two entities, Crv1(t) and Srf2(u, v), to find their common domains.

Step: Distance (step) in direction of tangent vec., while tracing.

SubdivTol: The subdivision tolerance to use.

NumericTol: The numerical tolerance to use.

Returns: Location where distance function square d2(t, u, v) of Crv1(t) and Srf2(u, v) is zero, as (t, u, v) tuples.

Description: Checks if given curve is precisely (partially) contained in the given surface and returns the portion that is indeed on Srf, if any. Computation is done by computing the distance field as a trivariate and finding the zeros of the derivatives of this distance field with respect to the two surface parameters.
See also: MvarMVDistCrvSrf, SymbSrfDistCrvCrv, SymbSrfDistFindPoints, , MvarMVDistSrfSrf, MvarMVDistCrvSrf, MvarProjCrvOnSrf, MvarProjCrvOnSrf1, , MvarCrvSrfInter, MvarSrfSrfInter,

8.2.513 MvarProjUVCrvOnE3CrvMatchSpeed (mvardist.c:1732)

```
CagdCrvStruct *MvarProjUVCrvOnE3CrvMatchSpeed(const CagdCrvStruct *UVLinCrv1,
                                              const CagdSrfStruct *Srf1,
                                              const CagdCrvStruct *UVCrv2,
                                              const CagdSrfStruct *Srf2)
```

UVLinCrv1: The linear curve in the UV domain of Srf1.

Srf1: Surface holding UVLinCrv1 in its domain.

UVCrv2: The UV curve in the UV domain of Srf2.

Srf2: Surface holding UVCrv2 in its domain.

Returns: Reconstructed matching linear UVCrv2 for Srf2.

Description: Projects all vertices of Srf1(UVLinCrv1) (UVLinCrv1 is in the UV space of Srf1) onto EucCrv2 = Sf2(UCrv2) and reconstruct another linear curve of same length as UVLinCrv1 for Srf2 that matches positions and speeds. That is, Srf2(NewUVLinCrv2) = Srf1(UVLinCrv1). The new returned curve is forced to interpolate end points of UVCrv2.
See also: TrimMatch2ndCrvLenSpeedAs1stCrv,

8.2.514 MvarPromoteMVToMV (mvar_rev.c:363)

multi-variates

```
MvarMVStruct *MvarPromoteMVToMV(const MvarMVStruct *MV, int Axis)
```

MV: Multi-Variate to promote.

Axis: Axis of promotion. Between zero and MV -> Dim.

Returns: Promoted multi-variate.

Description: Increase by one the dimensionality of the given multivariate, by introducing a new constant (degree zero) axis with one control point in direction Axis.
See also: MvarMVShiftAxes, MvarMVFromMV, MvarPromoteMVToMV2,

8.2.515 MvarPromoteMVToMV2 (mvar_rev.c:407)

multi-variates

```
MvarMVStruct *MvarPromoteMVToMV2(const MvarMVStruct *MV,
                                 int NewDim,
                                 int StartAxis)
```

MV: Multi-Variate to promote.

NewDim: New dimension of the promoted multivariate.

StartAxis: Original MV would span axes StartAxis to StartAxis+MV->Dim-1.

Returns: Promoted multi-variate.

Description: Increase by the dimensionality of the given multivariate to NewDim, by introducing new constant (degree zero) axes with one control points in all new directions. The Axis of the original MV will be starting at StartAxis.
See also: MvarMVShiftAxes, MvarMVFromMV, MvarPromoteMVToMV,

8.2.516 MvarPtCmpTwoPoints (mvar_pll.c:302)

```
int MvarPtCmpTwoPoints(const MvarPtStruct *P1,
                       const MvarPtStruct *P2,
                       CagdRType Eps)
```

P1, P2: Two multivariate points to compare.

Eps: The tolerance of the comparison.

Returns: 0 if identical, -1 or +1 if first point is less than/greater than second point, in lexicographic order over dimensions. 2 is returned if the dimensions are different.

Description: A comparison function to examine if the given two points are the same.
See also: MvarPtDistTwoPoints, MvarPtDistSqrTwoPoints, MvarVecCmpTwoVectors,

8.2.517 MvarPtCopy (mvar_gen.c:636)

```
MvarPtStruct *MvarPtCopy(const MvarPtStruct *Pt)
```

Pt: Multi-Variate point to duplicate.

Returns: Duplicated multi-variate point.

Description: Allocates and duplicates all slots of a multi-variate point structure.

8.2.518 MvarPtCopyList (mvar_gen.c:661)

multi-variates

```
MvarPtStruct *MvarPtCopyList(const MvarPtStruct *PtList)
```

PtList: List of multi-variate points to duplicate.

Returns: Duplicated list of multi-variate points.

Description: Duplicates a list of multi-variate point structures.

8.2.519 MvarPtDistSqrTwoPoints (mvar_pll.c:432)

```
CagdRType MvarPtDistSqrTwoPoints(const MvarPtStruct *P1, const MvarPtStruct *P2)
```

P1, P2: Two points to compute the distance between.

Returns: Distance computed.

Description: Compute the Eucildean distance between two multivariate points.
See also: MvarPtCmpTwoPoints, MvarPtDistTwoPoints,

8.2.520 MvarPtDistTwoPoints (mvar_pll.c:372)

```
CagdRType MvarPtDistTwoPoints(const MvarPtStruct *P1, const MvarPtStruct *P2)
```

P1, P2: Two points to compute the distance between.

Returns: Distance computed.

Description: Compute the Euclidean distance between two multivariate points.
See also: MvarPtCmpTwoPoints, MvarPtDistSqrTwoPoints,

8.2.521 MvarPtFree (mvar_gen.c:693)

```
void MvarPtFree(MvarPtStruct *Pt)
```

Pt: Multivariate point to free.

Returns: void

Description: Deallocates and frees all slots of a multi-variate point structure.
See also: MvarPtNew,

8.2.522 MvarPtFreeList (mvar_gen.c:717)

```
void MvarPtFreeList(MvarPtStruct *PtList)
```

PtList: Multi-Variate point list to free.

Returns: void

Description: Deallocates and frees a list of multi-variate point structures.

8.2.523 MvarPtInBetweenPoint (mvar_pll.c:397)

```
MvarPtStruct *MvarPtInBetweenPoint(const MvarPtStruct *Pt1,
                                   const MvarPtStruct *Pt2,
                                   CagdRType t)
```

Pt1, Pt2: Multivariate points.

t: Blending factor, 0 for Pt1, 0.5 for mid pt, 1 for Pt2.

Returns: The in-between point of Pt1 and Pt2.

Description: Computes an in-between point (middle if t = 0.5) of given two points.
See also: MvarPtCmpTwoPoints, MvarPtDistSqrTwoPoints,

8.2.524 MvarPtListGetCenter (krnl_based_cnstrct.c:2389)

```
void MvarPtListGetCenter(const IPObjectStruct *Pts, IrtPtType *Pt)
```

Pts: A Object list of points.

Pt: The computed center point.

Returns: void

Description: The function take a list of points as an input and computes the center point of the list.
See also:

8.2.525 MvarPtNew (mvar_gen.c:550)

```
MvarPtStruct *MvarPtNew(int Dim)
```

Dim: Number of dimensions of this multi-variate.

Returns: An uninitialized multi-variate point.

Description: Allocates the memory required for a new multi-variate point.
See also: MvarPtFree,

8.2.526 MvarPtRealloc (mvar_gen.c:590)

```
MvarPtStruct *MvarPtRealloc(MvarPtStruct *Pt, int NewDim)
```

Pt: Multi-Variate point to reallocate. Should not be used after this operation as it might be freed.

NewDim: Number of new dimensions of this multi-variate.

Returns: A reallocated point of dimension NewDim.

Description: Reallocates the memory that is required for a new dimension of a multi-variate point.
See also: MvarPtNew,

8.2.527 MvarPtSortListAxis (mvar_gen.c:881)

```
MvarPtStruct *MvarPtSortListAxis(MvarPtStruct *PtList, int Axis)
```

PtList: List of points to sort.

Axis: Axis to sort along: 1,2,3 for X,Y,Z.

Returns: Sorted list of points, in place.

Description: Sorts given list of points based on their increasing order in axis Axis. Sorting is done in place.

8.2.528 MvarPwrMVNew (mvar_gen.c:242)

multi-variates

```
MvarMVStruct *MvarPwrMVNew(int Dim, const int *Lengths, MvarPointType PType)
```

allocation

Dim: Number of dimensions of this multivariate.

Lengths: Of control mesh in each of the dimensions. Vector of size Dim.

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform multi-variate power basis.

Description: Allocates the memory required for a new power basis multi-variate.
See also: MvarMVFree, MvarMVNew, MvarBspMVNew, MvarBzrMVNew,

8.2.529 MvarRationalSrfsPoles (mvarpole.c:35)

```
MvarPolylineStruct *MvarRationalSrfsPoles(const CagdSrfStruct *Srf,
                                          CagdRType SubdivTol,
                                          CagdRType NumericTol)
```

Srf: Rational surface to extract its poles.

SubdivTol: The subdivision tolerance to use.

NumericTol: The numerical tolerance to use.

Returns: The poles, as piecewise linear approximations.

Description: Computes the poles of a rational surface, solving for the zeros of the surface's denominator.
See also: CagdPointsHasPoles, SymbCrvsSplitPoleParams,

8.2.530 MvarRoundChamferCrvAtC1Discont (mvtangnt.c:1523)

bi-tangent

```
CagdCrvStruct *MvarRoundChamferCrvAtC1Discont(const CagdCrvStruct *Crv,
                                              CagdCrvCornerType CornerType,
                                              CagdRType Radius)
```

Crv: The curve to round all its C1 discontinuities.

CornerType: Corner type desired.

Radius: The desired radius of rounding.

Returns: Rounded curve.

Description: Round all C1 discontinuities in the given curve Crv by an arc of radius Radius.
See also: MvarCircTanTo2Crvs, MvarRoundChamferCrvAtC1DiscontArc,

8.2.531 MvarRoundChamferCrvAtC1DiscontArc (mvtangnt.c:1428)

bi-tangent

```
CagdCrvStruct *MvarRoundChamferCrvAtC1DiscontArc(CagdCrvStruct **Crv1,
                                                 CagdCrvStruct **Crv2,
                                                 CagdCrvCornerType CornerType,
                                                 CagdRType Radius)
```

Crv1, Crv2: The two curves to bridge.

CornerType: Corner type desired.

Radius: The desired radius of rounding.

Returns: Constructed bridging arc.

Description: Build bridging arc in the C^1 discontinuity - i.e an arc or a chamfer. If a bridging arc between the end of Crv1 and the beginning of Crv2 is indeed found, Crv1/2 are also trimmed to the proper location of the arc.
See also: MvarCircTanTo2Crvs, MvarRoundChamferCrvAtC1Discont,

8.2.532 MvarSCvrBiNormals (mvar_cvr.c:1321)

bi-normal

```
MvarPtStruct *MvarSCvrBiNormals(const CagdCrvStruct *Crv1,
                                const CagdCrvStruct *Crv2,
                                CagdBType ElimDiagonals,
                                CagdRType RadiusLB,
                                CagdRType SubdivTol,
                                CagdRType NumericTol)
```

Crv1, Crv2: The input curves to which binormals are to be computed. Can be the same curve for self-bisectors (turn ElimDiagonals on then).

ElimDiagonals: If TRUE, the diagonal solutions are eliminated.

RadiusLB: Lower-bound on radius of circles - the diameter between the two intersecting points of the binormal and curves.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol < SubdivTol.

Returns: The list of solutions which consists of points of the form (u, v).

Description: Computes binormals to the given curves, i.e., the pair of points on curves C1 and C2 such that the line-segment joining these points is orthogonalto C1 and C2 at those points. This is done by solving the following system of two equations in two unknowns.

```
< C1(u) - C2(v), C1'(u) > = 0
< C1(u) - C2(v), C2'(v) > = 0
```

Diagonal solutions are eliminated, if requested, by factoring out the (u - v) term from the above multivariates.
See also: MvarCircTanTo2Crvs,

8.2.533 MvarSCvrCircTanTo2CrvsCntrOnCrv (mvar_cvr.c:186)

```
MvarPtStruct *MvarSCvrCircTanTo2CrvsCntrOnCrv(const CagdCrvStruct *Crv,
                                              const CagdCrvStruct *CntrOnCrv,
                                              CagdRType RadiusLB,
                                              CagdRType SubdivTol,
                                              CagdRType NumericTol)
```

Crv: The curves to find the circles that is tangent to at two points.

CntrOnCrv: The curve on which the center of circle should lie.

RadiusLB: Lower-bound on radius of circle.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol < SubdivTol.

Returns: List of (u, v, w) solution points.

Description: Computes all circles that are tangent to given curve at two points and whose centers lie on the given third curve. The following system of multivariats is solved. The last positivity constraint is added in order to eliminate the diagonal solutions.

```
< Crv(u) - CntrCrv(w), Crv'(u) > = 0,
< Crv(v) - CntrCrv(w), Crv'(v) > = 0,
|| Crv(u) - CntrCrv(w) || - || Crv(v) - CntrCrv(w) ||   = 0
u - v - SubdivTol*10 > 0
```

See also: MvarSCvrCircTanToCrvEndPtCntrOnCrv, MvarCircTanTo2Crvs,

8.2.534 MvarSCvrCircTanTo2CrvsEndPtNoDiag (mvar_cvr.c:711)

```
MvarPtStruct *MvarSCvrCircTanTo2CrvsEndPtNoDiag(const CagdCrvStruct *Crv1,
                                                const CagdCrvStruct *Crv2,
                                                const CagdPType Pt3,
                                                CagdBType ElimDiagonals,
                                                CagdRType RadiusLB,
                                                CagdRType SubdivTol,
                                                CagdRType NumericTol)
```

Crv1, Crv2: The two curves to find the circles that is tangent to both. Can be the same curve for self-bisectors (turn ElimDiagonals on then).

Pt3: The point through which circles should pass.

ElimDiagonals: Eliminate diagonal solutions if TRUE.

RadiusLB: Lower-bound on radius of circles.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol < SubdivTol.

Returns: List of (u, v) solution points.

Description: Computes all circles that are tangent to given two curves and pass through the given point. Solves for circles' Centers P(x, y), using the following process: Solve, symbolically for P in the following 2x2 system by Cremmer rule:

```
||C1(u) - P||^2 = ||C2(v) - P||^2,
||C1(u) - P||^2 = ||Pt3   - P||^2,
```

and substitute P into the following 2 euqations and solve 2 equations in (u, v):

```
< C1(u) - P, C1'(u) > = 0,
< C2(v) - P, C2'(v) > = 0,
```

See also: MvarSCvrCircTanTo3CrvsExprTreeNoDiagonal, MvarCircTanTo2Crvs,

8.2.535 MvarSCvrCircTanTo3CrvsExprTreeNoDiagonal (mvar_cvr.c:351)

tri-tangent

```
MvarPtStruct *MvarSCvrCircTanTo3CrvsExprTreeNoDiagonal(
                                                const CagdCrvStruct *Crv,
                                                CagdRType RadiusLB,
                                                CagdRType SubdivTol,
                                                CagdRType NumericTol)
```

Crv: The curves to find the circles that is tangent to at three points.

RadiusLB: Lower-bound on the radius of circles.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol < SubdivTol.

Returns: List of (u, v, w) solution points.

Description: Computes all circles that are tangent to given curve at three points. Diagonal solutions are eleminated. The computation is done using expression trees. Use this for big problems which otherwise require a lot of memory. Solves for circles' Centers P(x, y), using the following process: Solve, symbolically for P in the following 2x2 system by Crammer rule:

```
||C1(u) - P||^2 = ||C1(v) - P||^2,
||C1(u) - P||^2 = ||C1(w) - P||^2,
```

and substitute P into the following 3 euqations and solve 3 equations in (u, v, w):

```
< C1(u) - P, C1'(u) > = 0,
< C1(v) - P, C1'(v) > = 0,
< C1(w) - P, C1'(w) > = 0.
```

Diagonal solutions are eliminated by adding the following two constraints.

```
u - v - SubdivTol*10 > 0,
v - w - SubdivTol*10 > 0.
```

See also: MvarSCvrCircTanTo3Crvs, MvarCircTanTo3Crvs,

8.2.536 MvarSCvrCircTanToCrv2EndPt (mvar_cvr.c:1138)

```
MvarPtStruct *MvarSCvrCircTanToCrv2EndPt(const CagdCrvStruct *Crv1,
                                         const CagdPType Pt2,
                                         const CagdPType Pt3,
                                         CagdRType RadiusLB,
                                         CagdRType SubdivTol,
                                         CagdRType NumericTol)
```

Crv1: The curve to find the circles that is tangent to.

Pt2, Pt3: The two points through which the circles should pass.

RadiusLB: Lower-bound on radius of circles.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol < SubdivTol.

Returns: List of (u) solution points.

Description: Computes all circles that are tangent to the given curve and pass through the given two points.
See also: MvarCircTanTo2Crvs,

8.2.537 MvarSCvrCircTanToCrvEndPt (mvar_cvr.c:1047)

```
MvarPtStruct *MvarSCvrCircTanToCrvEndPt(const CagdCrvStruct *Crv1,
                                        const CagdPType Pt2,
                                        CagdRType RadiusLB,
                                        CagdRType SubdivTol,
                                        CagdRType NumericTol)
```

Crv1: The input curve to which the circles should be tangent.

Pt2: The point through which the circles should pass.

RadiusLB: Lower-bound on radius of circle.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol < SubdivTol.

Returns: The list of solutions which consists of points of the form (u).

Description: Computes all the circles tangent to a given curve and passing throuth the the given point. The line-segment joining the given point and the ponit on the curve passes through the center of the circle. This is done by solving the following univariate.

```
< C(u) - Pt2, C'(u) > = 0
```

See also: MvarSCvrCircTanTo2CrvEndpt, MvarSCvrCircTanToCrvEndPt, MvarSCvrBinormals,

8.2.538 MvarSCvrCircTanToCrvEndPtCntrOnCrv (mvar_cvr.c:59)

```
MvarPtStruct *MvarSCvrCircTanToCrvEndPtCntrOnCrv(
                                                const CagdCrvStruct *Crv1,
                                                const CagdPType Pt2,
                                                const CagdCrvStruct *CntrOnCrv,
                                                CagdRType RadiusLB,
                                                CagdRType SubdivTol,
                                                CagdRType NumericTol)
```

Crv1: The curve to find the circles that is tangent to.

Pt2: The point through which the circle should pass.

CntrOnCrv: The curve on which the center of circle should lie.

RadiusLB: Lower-bound on radius of circle.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol < SubdivTol.

Returns: List of (u, v) solution points.

Description: Computes all circles that are tangent to a given curve, are passing through a given point and whose centers lie on another curve. The following system of multivariates is solved.

```
< Crv1(u) - CntrCrv(v), Crv1'(u) > = 0,
|| Crv1(u) - CntrCrv(v) || = || Pt - CntrCrv(v) ||
```

See also: MvarSCvrCircTanTo2CrvsCntrOnCrv,

8.2.539 MvarSCvrPromoteCrvToSrfWithOtherCrv (mvar_cvr.c:1659)

bilinear

```
CagdSrfStruct *MvarSCvrPromoteCrvToSrfWithOtherCrv(
                                                const CagdCrvStruct *Crv,
                                                CagdSrfDirType Dir,
                                                const CagdCrvStruct *OtherCrv)
```

Crv: Curve to promote to a surfaces.

Dir: Direction in surface where Crv will be.

OtherCrv: Curve from which the other direction's parametrization is to be taken from.

Returns: Created surfaces or NULL if error.

Description: Creating a degenerate surface out of the given curve and parametrize the other direction of the surface to follow the parametrization of OtherCrv.
See also: MvarSCvrUMinusVSrf,

8.2.540 MvarSCvrUMinusVSrf (mvar_cvr.c:1599)

bilinear

```
CagdSrfStruct *MvarSCvrUMinusVSrf(const CagdRType *TMin, const CagdRType *TMax)
```

TMin: Array containing the two lower bounds of the domain.

TMax: Array containing the two upper bounds of the domain.

Returns: The surface.

Description: Creates a scalar surface in R^1 which is given by the following map (u, v) |-> (u - v).
See also: MvarSCvrPromoteCrvToSrfWithOtherCrv,

8.2.541 MvarSkel2DInter3Prims (skel2d.c:176)

skeleton
bisector

```
MvarSkel2DInter3PrimsStruct *MvarSkel2DInter3Prims(MvarSkel2DPrimStruct *Prim1,
                                                   MvarSkel2DPrimStruct *Prim2,
                                                   MvarSkel2DPrimStruct *Prim3)
```

Prim1, Prim2, Prim3: The three input primitives to consider.

Returns: A linked list of all equadistant points computed, or NULL if none found.

Description: Computes all points in R2 that are equadistant from the given three primitives. A primitive can be a point, a line, an arc, or a freeform curve. The end points of the line/arc/curve are NOT considered.
See also: MvarSkel2DSetEpsilon, MvarSkel2DSetFineness, MvarSkel2DSetOuterExtent,

8.2.542 MvarSkel2DInter3PrimsFree (skel2d.c:240)

free

```
void MvarSkel2DInter3PrimsFree(MvarSkel2DInter3PrimsStruct *SK2DInt)
```

SK2DInt: To be deallocated.

Returns: void

Description: Deallocates and frees all slots of 2d skeleton intersection structure.

8.2.543 MvarSkel2DInter3PrimsFreeList (skel2d.c:259)

free

```
void MvarSkel2DInter3PrimsFreeList(MvarSkel2DInter3PrimsStruct *SK2DIntList)
```

SK2DIntList: To be deallocated.

Returns: void

Description: Deallocates and frees a 2d skeleton intersection structure list:

8.2.544 MvarSkel2DSetEpsilon (skel2d.c:55)

skeleton
bisector

```
CagdRType MvarSkel2DSetEpsilon(CagdRType NewEps)
```

NewEps: New epsilon to use.

Returns: Old epsilon value.

Description: Sets the epsilon of the 2D skeleton computation.
See also: Skel2DInter3Primitives,

8.2.545 MvarSkel2DSetFineNess (skel2d.c:112)

skeleton
bisector

```
CagdRType MvarSkel2DSetFineNess(CagdRType NewFineNess)
```

NewFineNess: New fineness to use.

Returns: Old fineness value.

Description: Sets the fineness of the 2D skeleton computation.
See also: Skel2DInter3Primitives,

8.2.546 MvarSkel2DSetMZeroTols (skel2d.c:85)

```
CagdRType MvarSkel2DSetMZeroTols(CagdRType SubdivTol, CagdRType NumerTol)
```

SubdivTol, NumerTol: Subdivision and numeric tolerance of mvar solver.

Returns: Old SubdivTol.

Description: Sets the tolerances to be used in this module for the multivariate zero set solver.
See also: MvarMVsZeros, MvarSkel2DSetEpsilon, MvarSkel2DSetFineNess,

8.2.547 MvarSkel2DSetOuterExtent (skel2d.c:142)

skeleton

bisector

```
CagdRType MvarSkel2DSetOuterExtent(CagdRType NewOutExtent)
```

NewOutExtent: New outer extent to use.

Returns: Old outer extent value.

Description: Sets the outer extent of created (infinite) primitives in the 2D skeleton computation.
See also: Skel2DInter3Primitives,

8.2.548 MvarSpiralCrvOnSrf (mvardist.c:2224)

```
CagdCrvStruct *MvarSpiralCrvOnSrf(const CagdSrfStruct *Srf,
                                  CagdSrfDirType Dir,
                                  CagdRType Pitch,
                                  CagdRType Step,
                                  CagdRType SubdivTol,
                                  CagdRType NumerTol,
                                  int OutputType)
```

Srf: Surface on which to seek a Euclidean spiral curve.

Dir: The U or V direction of the spirals This Dir min and max should be the same (Srf is closed in Dir).

Pitch: Euclidean advancement in one spiral round surface.

Step: Step size during the numeric tracing.

SubdivTol: The subdivision tolerance to use.

NumerTol: The numerical tolerance to use.

OutputType: 0 - curves in UVT space, 1 - curves in Euclidean space, 2 - merged curves in Euclidean space.

Returns: Created curve on Srf.

Description: Construct a spiraling curve (mostly in Dir up to Pitch) on Srf. Srf is assumed to be closed/periodic in Dir.
See also: MvarEucCrvOffsetOnSrf,

8.2.549 MvarSrfAccessibility (mvaccess.c:47)

```
MvarPolylineStruct *MvarSrfAccessibility(const CagdSrfStruct *CPosSrf,
                                         const CagdSrfStruct *COrientSrf,
                                         const CagdSrfStruct *CCheckSrf,
                                         CagdRType SubdivTol,
                                         CagdRType NumerTol)
```

CPosSrf: The position surface to examine.

COrientSrf: The orientation field of the access to the position surface. Must be in same function space as CPosSrf. If NULL, the normal field of CPosSrf is employed.

CCheckSrf: The surface to check gouging against.

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the surfaces.

NumerTol: Numeric tolerance of a possible numeric improvement stage. The numeric stage is employed if NumericTol < SubdivTol.

Returns: The boundary of the accessible regions of CPosSrf.

Description: Derives the visibility domain of CPosSrf, when accessed from the direction that is specified by COrientSrf, not gouging into CCheckSrf. The boundary of the domain is prescribed by the following set of three constraints:

```
F1(u, v, s, t):  <O1(u, v), S(u, v) - K(s, t)> = 0
F2(u, v, s, t):  <O2(u, v), S(u, v) - K(s, t)> = 0
F3(u, v, s, t):  <Nk(s, t), S(u, v) - K(s, t)> = 0
```

where O1(u, v), O2(u, v) are two vector fields orthogonal to O(u, v), the orientation field. K(s, t) is the check surface and Nk(s, t) is its normal field, and S(u, v) is the position surface.

8.2.550 MvarSrfAntipodalPoints (selfintr.c:287)

```
MvarPtStruct *MvarSrfAntipodalPoints(const CagdSrfStruct *Srf,
                                     CagdRType SubdivTol,
                                     CagdRType NumericTol)
```

Srf: To detect its antipodal points.

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the surfaces.

NumericTol: Numeric tolerance of a possible numeric improvment stage. The numeric stage is employed if NumericTol < SubdivTol.

Returns: Antipodal points, as points in E4 (u, v, s, t).

Description: Computes antipodal points in the given surface - pairs of points S(u, v) and S(s, t) such that ((u, v) and (s, t) are two independent params of the same srf):

```
< S(u, v) - S(s, t), dS(u, v)/du > = 0,
< S(u, v) - S(s, t), dS(u, v)/dv > = 0,
< N(s, t), dS(u, v)/du > = 0,
< N(s, t), dS(u, v)/dv > = 0.
```

Direct attempt to solve this set of constraints is bound to be slow as all points in Srf satisfy these equations when (u, v) == (s, t). The key is in adding a fifth inequality constaint of the form

```
< N(u, v), N(s, t) > < 0.
```

Antipodal points must exists if Srf self intersect in a closed loop and hence can help in detecting sefl-intersections. Further, the diameter of Srf could be easily deduced from the antipodal points. Original version of this function was written by Diana Pekerman, Technion, Israel.

See also: MvarCrvAntipodalPoints, MvarHFDistAntipodalSrfSrfC1,

8.2.551 MvarSrfCalcAsympDirsCones (ln_access_hyper.c:163)

asymptotic directions

bounding volume

```
int MvarSrfCalcAsympDirsCones(const CagdSrfStruct *Srf,
                              SymbNormalConeStruct *DirsCones)
```

Srf: The surface for which to find the bounds of the asymptotic directions.

DirsCones: Out parameter: the array of bounding cones of the asymptotic directions of the surface. The array must be long enough to contain up to two cones.

Returns: The number of asymptotic directions on the surface.

Description: Computes bounding cones of the asymptotic directions of a surface. Up to two cones can be found.

See also: MvarSrfCalcAsympDirsVolume, MvarSrfCalcAsympDirsCoeffs,

8.2.552 MvarSrfCalcAsympDirsVolume (ln_access_hyper.c:39)

asymptotic directions

bounding volume

```
MvarMVStruct *MvarSrfCalcAsympDirsVolume(const CagdSrfStruct *Srf)
```

Srf: The surface for which to find the bounds of the asymptotic directions.

Returns: The multivariates which bound the asymptotic directions of the surface. A list of up to two multivariates can be returned.

Description: Computes bounding volumes of the asymptotic directions of a surface, as multivariates. The general form of the multivariates is: a(r) * Su(u,v) + b(r) * Sv(u,v), where Su, Sv, are the partial derivatives of the surface, and a(r), b(r) are the ranges of the coefficients of the asymptotic directions expressed as a function of parameter r. For more information on the computation of a(r), b(r), see: SymbSrfCalcAsympDirsCoeffs.
See also: MvarSrfCalcAsympDirsCoeffs,

8.2.553 MvarSrfFindOffsetSelfInter (self_inter_ofst.c:907)

```
IPObjectStruct *MvarSrfFindOffsetSelfInter(const CagdSrfStruct *Srf,
                                           CagdRType OffsetDist,
                                           CagdRType SubdivTol,
                                           CagdRType NumericTol,
                                           int OutputType)
```

Srf: Base surface to be offset.

OffsetDist: Distance between the offset surface and the base surface.

SubdivTol: The subdivision tolerance for the solver.

NumericTol: The numeric tolerance for the solver.

OutputType: 0 - return UVST coordinates of solution curves 1 - return UV coordinates of solution curves 2 - return XYZ coordinates of solution curves

Returns: A list of unfiltered global self intersection curves of the given offset surface.

Description: Main routine to find the unfiltered global self-intersection curves in offset surface by solving multivariate constraints equations with inequality constraints.
Example experiment setting used: SubdiviTol = 0.01, NumericTol = 1e-8 Currently StepSize and UVRTInequalityTol (eps used to remove trivial solutions) is 0.2 * SubdivTol and 5.0 * SubdivTol, resepctively.

8.2.554 MvarSrfFlecnodalCrvs (mvaccess.c:637)

```
MvarPolylineStruct *MvarSrfFlecnodalCrvs(const CagdSrfStruct *Srf,
                                         CagdRType Step,
                                         CagdRType SubdivTol,
                                         CagdRType NumerTol)
```

Srf: To compute the flecnodal curves for.

Step: Step size in the curve-tracing stage.

SubdivTol: Accuracy of the subdivision stage of the approximation.

NumerTol: Accuracy of numeric approx.

Returns: Flecnodal curves as polylines in (a, b, u, v) space.

Description: Computes the flecnodal curves on surface Srf = S(u, v). Uses the univariate MV solver.
See also: MvarFlecnodalCrvsCreateMVs, MvarSrfFlecnodalCrvs,

8.2.555 MvarSrfFlecnodalPts (mvaccess.c:717)

```
MvarPtStruct *MvarSrfFlecnodalPts(const CagdSrfStruct *CSrf,
                                  CagdRType SubdivTol,
                                  CagdRType NumerTol)
```

CSrf: To compute the silhouette higher orders' contact points.

SubdivTol: Accuracy of the subdivision stage of the approximation.

NumerTol: Accuracy of numeric approx.

Returns: Polylines on the unit sphere, depicting the flecnodal's partitioning lines.

Description: Computes the flecnodal points on surface Srf = S(u, v). Denote by Suv the second order derivatives of S(u, v) and by Suvv the third order derivatives. Then, solution of the following 4 equations in 4 unknowns (u, v, a, b) provides the flecnodal points:

```
            [ L M ] [ a ]
1. [ a  b ] [     ] [   ] =       (This location/direction is asymp.)
            [ M N ] [ b ]         (L, M, N are the coef. of SFF.)
```

```
              [ Suu Suv ]           [ a ]
    = [ a  b ] [         ] . n(u, v) [   ] = 0,
              [ Suv Svv ]           [ b ]
```

or

```
    < a^2 Suu + 2ab Suv + b^2 Svv, n(u, v) > = 0.
```

```
            (   [ Suuu Suuv ]       [ Svuu Svuv ] )            [ a ]
2. [ a  b ] ( a [           ]  +  b [           ] )  . n(u, v) [   ] = 0,
            (   [ Suuv Suvv ]       [ Svuv Svvv ] )            [ b ]
```

(A third derivative zero contact.)
or

```
    < a^3 Suuu + 3a^2b Suuv + 3ab^2 Suvv + b^3 Svvv, n(u, v) > = 0.
```

```
3. < a^4 Suuuu + 4 a^3b Suuuv + 6 a^2b^2 suuvv + 4ab^3 suvvv + b^4 Svvvv,
                                                        n(u, v) > = 0.
```

(A fourth derivative zero contact.)

```
4. a * a + b * b = 1             (A normalization nconstraint.)
```

See also: SymbSrfGaussCurvature, UserSrfTopoAspectGraph, SymbEvalSrfAsympDir, , MvarSrfSilhInflections,

8.2.556 MvarSrfHyperbolicLocallyAccessibleDirs (ln_access_hyper.c:227)

hyperbolic direction

line accessibility

```
CagdBType MvarSrfHyperbolicLocallyAccessibleDirs(
                                        const CagdSrfStruct *Srf,
                                        const CagdVType SrfVAxis,
                                        SymbNormalConeStruct *AccessibleCones)
```

Srf: The surface for which to find the locally accessible directions.

SrfVAxis: The (approximate) vertical axis of the surface. concave, hyperbolic, or mixed.

AccessibleCones: Output parameter. Two cones, representing ranges of accessible directions.

Returns: TRUE is accessible direction were found, FALSE otherwise.

Description: Compute a range of locally-accessible directions of a hyperbolic surface patch. The function computes cones containing all its asymptotic directions. These cones separate the tangent space of the surface patch into ranges of directions with negative normal curvature (accessible) and positive normal curvature (inaccessible). The locally-accessible directions are represented as cones.

See also: MvarSrfHyperbolicLocallyAccessibleDirs2,

8.2.557 MvarSrfHyperbolicLocallyAccessibleDirs2 (ln_access_hyper.c:410)

hyperbolic direction

line accessibility

```
CagdBType MvarSrfHyperbolicLocallyAccessibleDirs2(
                                        const CagdSrfStruct *Srf,
                                        SymbNormalConeStruct *AccessibleCones)
```

Srf: The surface for which to find the locally accessible directions.

AccessibleCones: Output parameter. Two cones, representing ranges of accessible directions.

Returns: TRUE is accessible direction were found, FALSE otherwise.

Description: Compute a range of locally-accessible directions of a hyperbolic surface patch, similarly to MvarSrfHyperbolicLocallyAccessibleDirs. This function does not require an approximate vertical axis for the surface, and instead computes it using SymbNormalConeForSrfToData.

See also: MvarSrfHyperbolicLocallyAccessibleDirs,

8.2.558 MvarSrfKernelPointIneql (srf_krnl_ineqls.c:251)

Surface Kernel

```
IPObjectStruct *MvarSrfKernelPointIneql(const CagdSrfStruct *Srfs,
                                        CagdRType SubEps,
                                        CagdRType ZeroEps,
                                        CagdRType BoxScale,
                                        CagdRType Gamma,
                                        int NumTanSamples,
                                        CagdBType DmnBoxOutput)
```

Srfs: A set of (regular) surfaces to compute a kernel point for.

SubEps: Subdivision epsilon.

ZeroEps: If > 0, then test f_i > -ZeroEps instead of f_i > 0. Points on the tangent planes of the surfaces are included in the kernel points, as a result.

BoxScale: Scale the bounding box of the surfaces to compute kernel points for. Set BoxScale > 1.0 to compute kernel points of open surfaces.

Gamma: If Gamma > 0, then compute the points that belong to the gamma kernel of the surfaces. Angle must be specified in degrees.

NumTanSamples: If > 0, then (NumTanSamples * NumTanSamples) tangent planes are sampled along each surface to purge the domains that do not belong to the kernel of the surfaces. If a xyz-domain is outside one of the sampled tangent planes, then the domain is purged during subdivision.

DmnBoxOutput: If TRUE, then output is a list of the kernel domain boxes in R^3. If FALSE, output is a list of the corner points of the kernel domain.

Returns: A list of the kernel domain boxes if DmnBoxOutput is TRUE, a list of the corner points in the kernel domain, otherwise.

Description: Computes a kernel point, if any, for the given surfaces. Let the input surface be S_i(u_i, v_i) and let N_i(u_i, v_i) be the normal filed of S_i(u_i, v_i), where i = 1, ..., n. Then: f_i(u_i, v_i, x, y, z) = < S_i(u_i, v_i)-P(x, y, z), N_i(u_i, v_i) >. If f_i(u_i, v_i, x, y, z) > 0, for all u_i, v_i and all i, then (x, y, z) is a kernel point. Computed only with only one semi-algebraic (inequality) constraint.

See also: MvarCrvKernelPoint, MvarCrvKernel, MvarCrvGammaKernel,

8.2.559 MvarSrfLineOneSidedMaxDist (lnsrfdst.c:117)

```
CagdRType MvarSrfLineOneSidedMaxDist(const CagdSrfStruct *Srf,
                                     const CagdUVType UV1,
                                     const CagdUVType UV2,
                                     CagdSrfDirType ClosedDir,
                                     CagdRType Epsilon)
```

Srf: Surface to compute the line distance to.

UV1, UV2: The two points on Srf the prescribes the line.

ClosedDir: If a valid direction the surface is to be treated as closed in that direction when examining the line segment.

Epsilon: Tolerance of computation.

Returns: The maximal distance from a point on the line to nearest location on the surfaces.

Description: Computes the maximal distance between a given line segment to a given surface. This is the one sided Hausdorff distance from the line to Srf. The line segment is assumed to be on the surface and is prescribed by two surface locations. The distance is bounded as follow. The composition of Srf(UV1UV2) is computed and the maximal distance to line is used as an upper bound, UB. Then, assuming the line is the Z axis, the farthest solution to: Nz(u, v) = 0, Nx(u, v) Y(u, v) - Ny(u, v) X(u, v) = 0,

```
where Srf = (X(u, v), Y(u, v), Z(u, v)), and
(Nx, Ny, Nz) is its normal field, that is smaller than UB is selected.
```

See also: MvarCrvMaxXYOriginDistance,

8.2.560 MvarSrfListPreciseBBox (mvarbbox.c:1241)

bbox

bounding box

```
void MvarSrfListPreciseBBox(const CagdSrfStruct *Srfs,
                            MvarBBoxStruct *BBox,
                            CagdRType Tol)
```

Srfs: To compute a precise bounding box for.

BBox: Where bounding information is to be saved.

Tol: Tolerance of computations.

Returns: void

Description: Computes the precise bounding box for a list of surfaces.
See also: CagdSrfListBBox, MvarSrfPreciseBBox,

8.2.561 MvarSrfPreciseBBox (mvarbbox.c:1174)

bbox

bounding box

```
void MvarSrfPreciseBBox(const CagdSrfStruct *Srf,
                        MvarBBoxStruct *BBox,
                        CagdRType Tol)
```

Srf: To compute a precise bounding box for.

BBox: Where bounding information is to be saved.

Tol: Tolerance of computations.

Returns: void

Description: Computes the precise bounding box for a surface.
See also: MvarMVBBox, MvarMVPreciseBBox, CagdSrfBBox,

8.2.562 MvarSrfRadialCurvature (mv_crvtr.c:51)

```
MvarPtStruct *MvarSrfRadialCurvature(const CagdSrfStruct *CSrf,
                                     const CagdVType ViewDir,
                                     CagdRType SubdivTol,
                                     CagdRType NumerTol)
```

CSrf: To compute the radial curvature lines for.

ViewDir: View direction to consider.

SubdivTol: Accuracy of the subdivision stage of the approximation.

NumerTol: Accuracy of numeric approx.

Returns: Polylines in the parameter space, depicting the radial curvature lines.

Description: Computes the radial curvature lines on surface Srf viewed from ViewDir. This amounts to finding the asymptotic directions of Srf that are in also the projection of ViewDir onto the tangent plane. Let a and b be two parameters that prescribe the tanget space. Then, the solution is derived as the solution of the following three equations:

```
      dS     dS
< ( a -- + b -- ) x N, V > = 0,  (the tangent direction is the projection
      du     dv                   of V onto the tangent plane.)
```

```
            [ L M ] [ a ]
  [ a  b ] [     ] [   ] = 0,    (This location/direction is asymp.)
            [ M N ] [ b ]         (L, M, N are the coef. of SFF.)
```

where a and b are normalized so

```
a^2 + b^2 = 1.
```

See also: SymbSrfGaussCurvature, UserSrfTopoAspectGraph, SymbEvalSrfAsympDir,

8.2.563 MvarSrfRayIntersect (ray-srf.c:155)

```
int MvarSrfRayIntersect(const CagdSrfStruct *Srf,
                        const CagdVType RayPt,
                        const CagdVType RayDir,
                        CagdUVStruct **InterPntsUV)
```

Srf: The surface.

RayPt: Starting point of the ray.

RayDir: The ray's direction.

InterPntsUV: Output parameters that contain all the intersection points in the UV parametric space of the surface, as linked list. Can be NULL if this information is not needed.

Returns: The number of intersection points.

Description: Calculates the intersection between a ray and a surface.

8.2.564 MvarSrfSelfInterDiagFactor (selfintr.c:1213)

```
MvarPolylineStruct *MvarSrfSelfInterDiagFactor(const CagdSrfStruct *Srf,
                                               CagdRType SubdivTol,
                                               CagdRType NumericTol)
```

Srf: To detect its self intersecting points.

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the surfaces.

NumericTol: Numeric tolerance of a possible numeric improvement stage. The numeric stage is employed if NumericTol < SubdivTol.

Returns: Self intersection pllns, as points in E4 (u, v, s, t).

Description: Computes self intersection points for a given surface S using the following constraints ((u, v) and (s, t) are two independent params of the same srf):

```
x(u, v) - x(s, t) = 0,
y(u, v) - y(s, t) = 0,
z(u, v) - z(s, t) = 0.
```

Direct attempt to solve this set of constraints is bound to be slow as all points in Srf satisfy these equations when u == v and s == t. The key here is to remove all (u - s) (and possibly (v - t)) factors off diagonal Bezier patches of the above, using a decomposition of the Bezier patches as (u1 - u3) G(u1, u2, u3, u4) + (u2 - u4) H(u1, u2, u3, u4).

See also: MvarSrfAntipodalPoints, MvarCrvSelfInterNrmlDev, MvarCrvSelfInterDiagFactor, MvarSrfSelfInterDiagFactor2,

8.2.565 MvarSrfSelfInterNrmlDev (selfintr.c:835)

```
MvarPolylineStruct *MvarSrfSelfInterNrmlDev(const CagdSrfStruct *Srf,
                                            CagdRType SubdivTol,
                                            CagdRType NumericTol,
                                            CagdRType MinNrmlDeviation)
```

Srf: To detect its self intersecting points.

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the surfaces.

NumericTol: Numeric tolerance of a possible numeric improvment stage. The numeric stage is employed if NumericTol < SubdivTol.

MinNrmlDeviation: The minimal angle teh surfaces are suppose to intersect at.

Returns: Self intersection pllns, as points in E4 (u, v, s, t).

Description: Computes self intersection points for given surface using the following constraints (u, v) and (s, t) are two independent params of the same srf:

```
x(u, v) - x(s, t) = 0,
y(u, v) - y(s, t) = 0,
z(u, v) - z(s, t) = 0.
```

Direct attempt to solve this set of constraints is bound to be slow as all points in Srf satisfy these equations when (u, v) == (s, t). The key is in adding a fourth inequality constaint of the form

```
     < N(u, v), N(s, t) >
--------------------------- < Cos(Angle),
|| N(u, v) ||  || N(s, t) ||
```

Where Cos(Angle) is a provided constant that prescribes the minimal angle the surface is expected to intersect at. The closer Cos(Angle) to one the more work this function will have to do in order to isolate the self intersection curve. The above expression is not rational and so, we use a logical or of the following two expressions:

```
     < N(u, v), N(s, t) >^2
------------------------------ < Cos^2(Angle),
|| N(u, v) ||^2 || N(s, t) ||^2
```

or

$< N(u, v), N(s, t) > < 0.$

See also: MvarSrfAntipodalPoints, MvarCrvSelfInterNrmlDev, MvarSrfSelfInterDiagFactor,

8.2.566 MvarSrfSilhInflections (mvaccess.c:229)

```
MvarPtStruct *MvarSrfSilhInflections(const CagdSrfStruct *Srf,
                                     const CagdVType ViewDir,
                                     CagdRType SubdivTol,
                                     CagdRType NumerTol)
```

Srf: To compute the silhouette higher orders' contact points.

ViewDir: View direction to consider.

SubdivTol: Accuracy of the subdivision stage of the approximation.

NumerTol: Accuracy of numeric approx.

Returns: Polylines on the unit sphere, depicting the flecnodal's partitioning lines.

Description: Computes the silhouette locations on surface Srf viewed from ViewDir that has inflection points or contact for second order or more. This amounts to finding the asymptotic directions of Srf that are in the direction of ViewDir. Let V(s, t) be a parametrization of all possible viewing directions. Solution is derived as the solution of the following three equations:

$$\left\langle \frac{\partial S}{\partial u} \times \frac{\partial S}{\partial v}, V \right\rangle = 0, \qquad \text{(View direction is in the tangent space)}$$

$$\begin{bmatrix} a & b \end{bmatrix} \begin{bmatrix} L & M \\ M & N \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = 0, \qquad \text{(This location/direction is asymp.)}$$

$$\text{(L, M, N are the coef. of SFF.)}$$

where a and b are the solutions of

$$a \frac{\partial S}{\partial u} + b \frac{\partial S}{\partial v} = V.$$

See also: SymbSrfGaussCurvature, UserSrfTopoAspectGraph, SymbEvalSrfAsympDir,

8.2.567 MvarSrfSilhouette (mvarsils.c:41)

```
IPObjectStruct *MvarSrfSilhouette(const CagdSrfStruct *Srf,
                                  const CagdVType VDir,
                                  CagdRType Step,
                                  CagdRType SubdivTol,
                                  CagdRType NumericTol,
                                  CagdBType Euclidean)
```

Srf: To compute its silhouette edges.

VDir: View direction vector (a unit vector).

Step: Step size for curve tracing.

SubdivTol: The subdivision tolerance to use.

NumericTol: The numerical tolerance to use.

Euclidean: If TRUE, returns the silhouettes in Euclidean space. Otherwise, the silhouette edges are returned in the Parametric domain.

Returns: The silhouettes as piecewise linear edges. Can include two object, 1st with points.

Description: Computes the silhouette edges of the given surfaces, orthographically seen from the given view direction VDir.

See also: SymbSrfOrthotomic, SymbSrfSilhouette,

8.2.568 MvarSrfSilhouetteThroughPoint (mvarsils.c:124)

```
IPObjectStruct *MvarSrfSilhouetteThroughPoint(const CagdSrfStruct *Srf,
                                              const CagdPType VPoint,
                                              CagdRType Step,
                                              CagdRType SubdivTol,
                                              CagdRType NumericTol,
                                              CagdBType Euclidean)
```

Srf: To compute its silhouette edges.

VPoint: A point location.

Step: Step size for curve tracing.

SubdivTol: The subdivision tolerance to use.

NumericTol: The numerical tolerance to use.

Euclidean: If TRUE, returns the silhouettes in Euclidean space. Otherwise, the silhouette edges are returned in the Parametric domain.

Returns: The silhouettes as piecewise linear edges. Can include two object, 1st with points.

Description: Computes the silhouette edges of the given surfaces, seen from the given view point VPoint.
See also: SymbSrfOrthotomic, SymbSrfSilhouette, MvarSrfSilhouetteThroughPoint2,

8.2.569 MvarSrfSilhouetteThroughPoint2 (mvarsils.c:188)

```
struct IPObjectStruct *MvarSrfSilhouetteThroughPoint2(
                                                  MvarMVStruct *SrfMv,
                                                  const MvarMVStruct *NrmlMv,
                                                  const CagdPType VPoint,
                                                  CagdRType Step,
                                                  CagdRType SubdivTol,
                                                  CagdRType NumericTol)
```

SrfMv: A MultiVariate problem for the surface. This variable will be translated by -VPoint, in place.

NrmlMv: A MultiVariate for the normals field of SrfMV.

VPoint: A point location.

Step: Step size for curve tracing.

SubdivTol: The subdivision tolerance to use.

NumericTol: The numerical tolerance to use.

Returns: The silhouettes as piecewise linear edges. Can include two object, 1st with points.

Description: Computes the silhouette edges of the given surfaces, seen from the given view point VPoint.
See also: SymbSrfOrthotomic, SymbSrfSilhouette, MvarSrfSilhouetteThroughPoint,

8.2.570 MvarSrfSplitPoleParams (mvarpole.c:91)

```
TrimSrfStruct *MvarSrfSplitPoleParams(const CagdSrfStruct *Srf,
                                      CagdRType SubdivTol,
                                      CagdRType NumericTol,
                                      CagdRType OutReach)
```

Srf: Rational surface to split at its poles.

SubdivTol: The subdivision tolerance to use.

NumericTol: The numerical tolerance to use.

OutReach: Small offset to clip poles regions at, zero to disable.

Returns: Trimmed surfaces, divided at all poles, or NULL if no poles.

Description: Split the given rational surface at its poles, if any, by solving for the zeros of the surface's denominator.
See also: CagdPointsHasPoles, SymbCrvsSplitPoleParams,

8.2.571 MvarSrfSrfBisector (mvbisect.c:254)

bisectors

```
MvarMVStruct *MvarSrfSrfBisector(const MvarMVStruct *CMV1,
                                 const MvarMVStruct *CMV2)
```

CMV1, CMV2: The two bivariates (surfaces) in R^5.

Returns: The resulting bisector.

Description: Computes the bisectors of two surfaces in R^5.
See also: MvarMVsBisector, MvarCrvSrfBisector, MvarSrfSrfBisectorApprox,

8.2.572 MvarSrfSrfBisectorApprox (mvbisect.c:860)

bisectors

```
MvarZeroSolutionStruct *MvarSrfSrfBisectorApprox(const MvarMVStruct *CMV1,
                                                 const MvarMVStruct *CMV2,
                                                 CagdRType SubdivTol,
                                                 CagdRType NumericTol)
```

CMV1, CMV2: The two bivariates (surfaces) in R^3.

SubdivTol: Tolerance of the first zero set finding subdivision stage.

NumericTol: Tolerance of the second zero set finding numeric stage.

Returns: Either a list of polyline loops or a list of triangles.

Description: Computes an approximation to the bisector of two surfaces. Let S1(u, v) and S2(r, s) be two parametric surfaces and let n1(u, v) and n2(r, s) be their unnormalized normal fields. Because the two normals of the two surfaces must be coplanar we introduce the following constraint, forcing the three vectors n1, n2, and S1 - S2 to all be in the same plane.

```
< ( S1(u, v) - S2(r, s) ) x n1(u, v), n2(r, s) > = 0.
```

To make sure the distance to the intersection point of the normals, from both surface's foot points we also coerces these three vectors to form a isosceles triangle:

```
|| n2(r, s) ||^2 < S1(u, v) - S2(r, s), n1(u, v) > ^ 2 -
                || n1(r, s) ||^2 < S1(u, v) - S2(r, s), n2(u, v) > ^ 2
```

Finding the zero set of the last equation provides the correspondence between the (u, v) locations on the first surface and (r, s) locations on the second surface that serve as mutual foot point for some bisector point.
See also: MvarMVsBisector, MvarCrvSrfBisector, MvarSrfSrfBisector, MvarCrvSrfBisectorApprox,

8.2.573 MvarSrfSrfBisectorApprox2 (mvbisect.c:1016)

bisectors

```
VoidPtr MvarSrfSrfBisectorApprox2(const MvarMVStruct *CMV1,
                                  const MvarMVStruct *CMV2,
                                  int OutputType,
                                  CagdRType SubdivTol,
                                  CagdRType NumericTol)
```

CMV1, CMV2: The two bivariates (surfaces) in R^3.

OutputType: Expected output type: 1. For the computed multivariate constraints. 2. For the computed point cloud on the bisector. 3. Points in a form of (u1, v2, x, y, z) where (u1, v1) are the parameter space of the first surface.

SubdivTol: Tolerance of the first zero set finding subdivision stage.

NumericTol: Tolerance of the second zero set finding numeric stage.

Returns: Following OutputType, either a set of multivariates (as a linked list of MvarMVStruct), or a cloud of points on the bisector (as a linked list of MvarPtStruct).

Description: Computes an approximation to the bisector of two surfaces - old version that does not use the 2D solver. Let S1(u, v) and S2(r, s) be two parametric surfaces and let n1(u, v) and n2(r, s) be their unnormalized normal fields. Because the two normals of the two surfaces must be coplanar we introduce the following constraint, forcing the three vectors n1, n2, and S1 - S2 to all be in the same plane.

```
< ( S1(u, v) - S2(r, s) ) x n1(u, v), n2(r, s) > = 0.
```

To make sure the distance to the intersection point of the normals, from both surface's foot points we also coerces these three vectors to form a isosceles triangle:

```
|| n2(r, s) ||^2 < S1(u, v) - S2(r, s), n1(u, v) > ^ 2 -
               || n1(r, s) ||^2 < S1(u, v) - S2(r, s), n2(u, v) > ^ 2
```

Finding the zero set of the last equation provides the correspondence between the (u, v) location and the first surface and (r, s) locations on the second surface that serve as mutual foot point for some bisector point.

See also: MvarMVsBisector, MvarCrvSrfBisector, MvarSrfSrfBisector, MvarCrvSrfBisectorApprox,

8.2.574 MvarSrfSrfCacheShouldAssignIds (ssi_cache.c:171)

```
CagdBType MvarSrfSrfCacheShouldAssignIds(const MvarSrfSrfInterCacheStruct
                                                                    *DataCache)
```

DataCache: The cache.

Returns: TRUE iff the cache is responsible for assigning surfaces Ids.

Description: Returns if it is the cache responsibility for assigning surface Ids.

8.2.575 MvarSrfSrfContact (mvarintr.c:901)

```
MvarPtStruct *MvarSrfSrfContact(const CagdSrfStruct *Srf1,
                                const CagdSrfStruct *Srf2,
                                const CagdCrvStruct *MotionSrf1,
                                const CagdCrvStruct *ScaleSrf1,
                                CagdRType SubdivTol,
                                CagdRType NumericTol,
                                CagdBType UseExprTree)
```

Srf1, Srf2: Two surface to compute contacts over time in R^3.

MotionSrf1: The motion over time Srf1 undergoes. Can be NULL.

ScaleSrf1: The scale over time Srf1 undergoes. Can either be a scalar function, or vector function in R^3. Can be NULL. If both MotionSrf1 and Srf1Scale are defined, they better share their domains.

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the surfaces.

NumericTol: Numeric tolerance of a possible numeric improvement stage. The numeric stage is employed if NumericTol < SubdivTol.

UseExprTree: TRUE to use expression trees in the computation, FALSE to use regular multivariate expressions.

Returns: List of intersection points, as parameter pairs into the two surfaces domains.

Description: Computes the contact locations of two C^1 surfaces in R^3, possibly with multivariate expression trees. Expression trees could be beneficial computationally when the geometry is complex (i.e. dozens of control points or more, in each directions).

See also: MvarCrvCrvInter, MvarSrfSrfSrfInter,

8.2.576 MvarSrfSrfInter (mvar_ssi.c:641)

```
MvarPolylineStruct *MvarSrfSrfInter(const CagdSrfStruct *Srf1,
                                    const CagdSrfStruct *Srf2,
                                    CagdRType Step,
                                    CagdRType SubdivTol,
                                    CagdRType NumericTol)
```

Srf1, Srf2: Two surfaces to be intersected.

Step: Distance (step) in direction of tangent vec., while tracing.

SubdivTol: The subdivision tolerance to use.

NumericTol: The numerical tolerance to use.

Returns: The list of polylines which approximate the curve. Each polyline corresponds to the topologically isolated component of the curve and is in R^4, the parametric spaces of both surfaces.

Description: Computes intersection curve of two surfaces.
See also: MvarMVsZeros1DMergeSingularPts, MvarMVsZeros1D, MvarSrfZeroSet, , MvarCrvSrfSingularInter, MvarSrfSrfInterDisc,

8.2.577 MvarSrfSrfInterCacheAddData (ssi_cache.c:198)

```
MvarSrfSrfInterCacheDataStruct *MvarSrfSrfInterCacheAddData(
                                        MvarSrfSrfInterCacheStruct *SSICache,
                                        CagdSrfStruct *Srf1,
                                        CagdSrfStruct *Srf2,
                                        MvarPolylineStruct *InterData)
```

SSICache: The cache.

Srf1: The first surface .

Srf2: The second surface .

InterData: Intersection result of Srf1 and Srf2.

Returns: Cached data of the intersection. The data is not added twice if already exists.

Description: Adds a copy of the result of intersection of two surfaces to the cache. The order of the surfaces is not important.

8.2.578 MvarSrfSrfInterCacheAlloc (ssi_cache.c:35)

```
MvarSrfSrfInterCacheStruct *MvarSrfSrfInterCacheAlloc(
                                                    IPAttrIDType AttribID,
                                                    CagdBType ShouldAssignIds)
```

AttribID: The surface Id attribute name.

ShouldAssignIds: If FALSE, the user is responsible for assigning Ids for the surfaces, otherwise it is the cache responsibility.

Returns: New cache structure.

Description: Allocated new SSI cache structure. Each surface in the cache should have a unique integer Id attribute. Since the attributes are copied when the surface is copied, it is not necessary to use the same surface for querying the cache, a copy can be used also.

8.2.579 MvarSrfSrfInterCacheClear (ssi_cache.c:274)

```
void MvarSrfSrfInterCacheClear(MvarSrfSrfInterCacheStruct *SSICache)
```

SSICache: The cache.

Returns: void.

Description: Empties a given SSI cache.

8.2.580 MvarSrfSrfInterCacheFree (ssi_cache.c:298)

```
void MvarSrfSrfInterCacheFree(MvarSrfSrfInterCacheStruct *SSICache)
```

SSICache: The cache.

Returns: void.

Description: Deallocated memory of a given SSI cache.

8.2.581 MvarSrfSrfInterCacheGetData (ssi_cache.c:95)

```
MvarSrfSrfInterCacheDataStruct *MvarSrfSrfInterCacheGetData(
                                    const MvarSrfSrfInterCacheStruct *SSICache,
                                    const CagdSrfStruct *Srf1,
                                    const CagdSrfStruct *Srf2)
```

SSICache: The cache.

Srf1: The first surface.

Srf2: The second surface.

Returns: Cached result of the intersection, or NULL if not found in the cache.

Description: Searches for a cached result of intersection of two given surfaces. The order between the surfaces is not important.

8.2.582 MvarSrfSrfInterCacheGetSrfId (ssi_cache.c:63)

```
int MvarSrfSrfInterCacheGetSrfId(const MvarSrfSrfInterCacheStruct *SSICache,
                                 const CagdSrfStruct *Srf)
```

SSICache: The cache.

Srf: The surface to get its Id.

Returns: integer Id of the surface, or IP_ATTR_BAD_INT if doesn't exist.

Description: Returns the Id of a given surface.

8.2.583 MvarSrfSrfInterDisc (mvar_ssi.c:322)

```
MvarPolylineStruct *MvarSrfSrfInterDisc(const CagdSrfStruct *Srf1,
                                        const CagdSrfStruct *Srf2,
                                        CagdRType Step,
                                        CagdRType SubdivTol,
                                        CagdRType NumericTol)
```

Srf1: First surface to be intersected.

Srf2: Second surface to be intersected.

Step: Step size for curve tracing.

SubdivTol: The subdivision tolerance to use.

NumericTol: The numerical tolerance to use.

Returns: The list of polylines which approximate the curve. Each polyline corresponds to the topologically isolated component of the curve and is in R^4, the parametric spaces of both surfaces.

Description: Computes intersection curve of two surfaces. This function detects special cases of C^1 discontinuity curves and treat them first. Then, the regular SSI cases are resolved by MvarSrfSrfIner.

See also: MvarMVsZeros1DMergeSingularPts, MvarMVsZeros1D, MvarSrfZeroSet, , MvarCrvSrfSingularInter, MvarSrfSrfInter,

8.2.584 MvarSrfSrfInterExamineBBoxes (mvar_ssi.c:1092)

```
IrtBType MvarSrfSrfInterExamineBBoxes(const CagdSrfStruct *Srf1,
                                      const CagdSrfStruct *Srf2)
```

Srf1, Srf2: Surfaces to be intersected.

Returns: TRUE if the surfaces BBoxes intersect, FALSE otherwise.

Description: Examine bounding boxes for no-overlap for quick pruning.

8.2.585 MvarSrfSrfInterNormalizeDomain (mvar_ssi.c:1065)

```
CagdRType MvarSrfSrfInterNormalizeDomain(CagdRType NewValue)
```

NewValue: The new value to set to the flag.

Returns: The previous value of the flag.

Description: Set a flag to control if domains of interesting surfaces are temporarily normalized to a canonical domain, before SSI takes place. This normalization have virtually no affect on the SSI output.

See also: MvarMVCompose, MvarMVComposeBzrBzr, MvarComposeExtractTargetSubdomain,

8.2.586 MvarSrfSrfMinimalDist (hasdrf3d.c:1023)

```
MvarPtStruct *MvarSrfSrfMinimalDist(const CagdSrfStruct *Srf1,
                                    const CagdSrfStruct *Srf2,
                                    CagdRType *MinDist)
```

Srf1: To detect its minimal distance to Srf2.

Srf2: To detect its minimal distance to Srf1.

MinDist: Upon return, is set to the minimal distance detected.

Returns: Pairs of parameters at the minimal distance.

Description: Computes the minimal distance between two given C1 surfaces. Surfaces are assumed to not intersect. This minimal distance can occur:

1. At end points vs. end points.
2. At the end points vs interior locations.
3. At a boundary curve vs the other surface.
4. At antipodal interior locations.

See also: MvarSrfSrfAntipodalPoints, MvarDistSrfPoint, MvarCrvSrfMinimalDist,

8.2.587 MvarSrfSrfMinkowskiSum (mink_sum.c:116)

```
struct IPObjectStruct *MvarSrfSrfMinkowskiSum(const CagdSrfStruct *Srf1,
                                              const CagdSrfStruct *Srf2,
                                              CagdRType SubdivTol,
                                              CagdRType CrvTraceStep,
                                              CagdRType NumericTol,
                                              int ParallelNrmls,
                                              CagdRType OffsetTrimDist)
```

Srf1, Srf2: The surfaces to compute their Minkowski sum.

SubdivTol: The subdivision tolerance for the solver call.

CrvTraceStep: Univariate curve tracing step size.

NumericTol: The numeric tolerance for the solver call.

ParallelNrmls: A flag determining if the problem solved shall be parallel normal vectors only (1), anti-parallel normals only (-1) or both (0).

OffsetTrimDist: When positive, indicates that the Minkowski sum is an offset problem, under the convention that the second surface, Srf2, is a sphere of radius OffsetTrimDist, and the value is used in the offset trimming. If negative, trimming is skipped (the problem is not an offset problem, or any other reason).

Returns: List of polygons or polylines (according to the setting of the bivariate solver), approximating the Minkowski sum.

Description: Computation of the Minkowski sum of two surfaces. The sum is defined as the locus of points Srf1(u,v) + Srf2(r,s) in R^3, for (u,v) and (r,s) pairs in the parameter spaces, such that the corresponding normal vectors N1(u,v) and N2(r,s) are parallel. In other words, we solve:
< Srf1_u x Srf1_v , Srf2_r > = 0, < Srf1_u x Srf1_v , Srf2_s > = 0.
The solution triangles (or polylines) in (u,v,r,s) space, are then mapped to R^3 using the post-process mapping Srf1(u,v) + Srf2(r,s).
See also: MvarSrfSrfMinkSumMVs,

8.2.588 MvarSrfSrfSrfInter (mvarintr.c:301)

```
MvarPtStruct *MvarSrfSrfSrfInter(const CagdSrfStruct *Srf1,
                                 const CagdSrfStruct *Srf2,
                                 const CagdSrfStruct *Srf3,
                                 CagdRType SubdivTol,
                                 CagdRType NumericTol,
                                 CagdBType UseExprTree)
```

Srf1, Srf2, Srf3: Three surface to intersect in R^3.

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the surfaces.

NumericTol: Numeric tolerance of a possible numeric improvement stage. The numeric stage is employed if NumericTol < SubdivTol.

UseExprTree: TRUE to use expression trees in the computation, FALSE to use regular multivariate expressions.

Returns: List of intersection points, as parameter pairs into the three surfaces domains.

Description: Computes the intersection locations of three surfaces in R^3, possibly with multivariate expression trees. Expression trees could be beneficial computationally when the geometry is complex (i.e. dozens of control points or more, in each directions).
See also: MvarCrvCrvInter, MvarSrfSrfContact,

8.2.589 MvarSrfSrfTestInter (mvar_ssi.c:862)

```
CagdBType MvarSrfSrfTestInter(const CagdSrfStruct *Srf1,
                              const CagdSrfStruct *Srf2,
                              CagdRType Step,
                              CagdRType SubdivTol,
                              CagdRType NumericTol)
```

Srf1, Srf2: The two surfaces to be tested for intersection.

Step: Distance (step) in direction of tangent vec., while tracing.

SubdivTol: The subdivision tolerance to use.

NumericTol: The numerical tolerance to use.

Returns: TRUE if the surfaces intersect, FALSE otherwise.

Description: Tests if the given two surfaces intersect.
See also: MvarSrfSrfInter, MvarMVsZeros1DMergeSingularPts, MvarMVsZeros1D, , MvarSrfZeroSet, MVarCrvSrfSingularInter, MvarSrfSrfInterC1Disc, , MvarSrfSrfInterNormalizeDomain,

8.2.590 MvarSrfTrimGlblOffsetSelfInter (offset2.c:300)

```
IPObjectStruct *MvarSrfTrimGlblOffsetSelfInter(CagdSrfStruct *Srf,
                                               const CagdSrfStruct *OffSrf,
                                               CagdRType TrimAmount,
                                               int Validate,
                                               int Euclidean,
                                               CagdRType SubdivTol,
                                               CagdRType NumerTol,
                                               CagdBType NumerImp)
```

Srf: Original surface.

OffSrf: The offset surface approximation.

TrimAmount: The trimming distance. A fraction smaller than the offset amount.

Validate: If TRUE, compute only points along the self intersections that are valid and serve to delineate between valid and invalid (to be purged) offset surface regions.

Euclidean: If TRUE, returned data is in the Euclidean space of OffSrf. FALSE for parametric space of OffSrf.

SubdivTol: Accuracy of computation.

NumerTol: If smaller that SubdivTol, a numerical improvement stage is applied by the solver.

NumerImp: If TRUE, a final stage of numerical marching over the surface is applied to improve the solution even further.

Returns: A list of self inter. curves, on the offset surface.

Description: trims regions in the offset surface OffSrf that are closer than TrimAmount to original Srf. TrimAmount should be a fraction smaller than the offset amount itself. See also: Joon-Kyung Seong, Gershon Elber, and Myung-Soo Kim. "Trimming Local and Global Self-intersections in Offset Curves and Surfaces using Distance Maps." Computer Aided Design, Vol 38, No 3, pp 183-193, March 2006.

8.2.591 MvarSrfTrimGlblOffsetSelfInterNI (offst2ni.c:88)

```
IPObjectStruct *MvarSrfTrimGlblOffsetSelfInterNI(IPPolygonStruct *Plls,
                                                 const CagdSrfStruct *OffSrf,
                                                 CagdRType SubdivTol,
                                                 CagdRType NumerTol,
                                                 int Euclidean,
                                                 CagdRType SameUVTol)
```

Plls: Connected list of polylines to be improved.

OffSrf: The offset surface approximation.

SubdivTol: Accuracy of computation.

NumerTol: NumerTol tolerance.

Euclidean: If TRUE, returned data is in the Euclidean space of OffSrf. FALSE for parametric space of OffSrf.

SameUVTol: Tolerance for termination condition around singular points.

Returns: A list of self inter. curves, on the offset surface.

Description: Improve self-intersection curves of offset surfaces using numerical methods.

8.2.592 MvarSrfZeroSet (mvar_ssi.c:1000)

```
MvarPolylineStruct *MvarSrfZeroSet(const CagdSrfStruct *Surface,
                                   int Axis,
                                   CagdRType Step,
                                   CagdRType SubdivTol,
                                   CagdRType NumericTol,
                                   MvarMVsZerosVerifyAllSolsCBFuncType
                                                        OutputPtsFilterCB,
                                   void *OutputPtsFilterCBData)
```

Surface: Cagd surface to compute its zeros.

Axis: Axis of Surface to consider its zeros.

Step: Distance (step) in direction of tangent vec., while tracing.

SubdivTol: The subdivision tolerance to use.

NumericTol: The numerical tolerance to use.

OutputPtsFilterCB: Optional filtering function for output. NULL to disable.

OutputPtsFilterCBData: Optional data to to pass to OutputPtsFilterCB.

Returns: The list of polylines which approximate the zeros. Each polyline corresponds to the topologically isolated component of the curve and is in R^2, the parametric spaces of the surface.

Description: Computes the zeros of some surface.
See also: MvarCrvZeroSet, MvarMVsZeros1D, MvarSrfSrfInter,

8.2.593 MvarStewartPlatform2Solve (mvarstpl.c:429)

```
MvarPtStruct *MvarStewartPlatform2Solve(const CagdPType BottomBasePoints[3],
                                        const CagdRType BotTopEdgeLengths[6],
                                        const CagdRType TopEdgeLengths[3],
                                        CagdBType Rational,
                                        CagdRType SubdivTol,
                                        CagdRType NumericTol)
```

BottomBasePoints: The three points of the bottom base.

BotTopEdgeLengths: The six edge lengths of rods connection bottom and top bases.

TopEdgeLengths: The three edge lengths of the top base.

Rational: TRUE to represent the circles precisely as rational, FALSE to approximate the circles as polynomials.

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the surfaces.

NumericTol: Numeric tolerance of a possible numeric improvement stage. The numeric stage is employed if NumericTol < SubdivTol.

Returns: Computed solution locations.

Description: A second solver for Stewart platform that is optimized. The original constraints are mapped to find a triangle (of TopEdgeLengths edges) whose vertices resides on three computed circles Ci, as follows,

1. Compute circles Ci, i = 0, 1, 2, as the locus of points of distance BotTopEdgeLengths[2i], and BotTopEdgeLengths[2i+1] from bottom base points BottomBasePoints[i], BottomBasePoints[(i+1)%3], respectively.
2. Solve for the three constraints and three unknowns || Ci - C[(i+1)%3] || = TopEdgeLengths[i].

See also: MvarStewartPlatformSolve, MvarStewartPlatformGenEqns,

8.2.594 MvarStewartPlatformGenEqns (mvarstpl.c:205)

```
MvarMVStruct **MvarStewartPlatformGenEqns(const CagdPType BottomBase[3],
                                          const CagdRType TopBaseEdgeLengths[6],
                                          const CagdPType WorkDomain[2])
```

BottomBase: Three vertices of triangle forming the base, denoted Pi.

TopBaseEdgeLengths: Three edge lengths between three vertices Qi, of top base.

WorkDomain: The Euclidean space working domain min./max. range.

Returns: Dynamically allocated vector of nine constraints, in nine unknowns (bottom base with given vertices Pi, top vertices with unknown vertices Qi) as follows: || Q1 - P1 || = L1, || Q1 - P2 || = L2, || Q2 - P2 || = L3, || Q2 - P3 || = L4, || Q3 - P3 || = L5, || Q3 - P1 || = L6, || Q1 - Q2 || = D1, || Q2 - Q3 || = D2, || Q3 - Q1 || = D3, where Li are the six BaseConnectLengths and Dj are lengths between adjacent edges of the top, moving, base. Note the Li are not given and so in this building process the are assumed all zero.

Description: Derive nine constraints for a Stewart platform, so that given the lengths of the six rods connection the bottom and top bases, we can solve the problem. The bottom base is specified and so are the lengths of the edges of the top base.
See also: MvarStewartPlatformSolve,

8.2.595 MvarStewartPlatformSolve (mvarstpl.c:289)

```
MvarPtStruct *MvarStewartPlatformSolve(const MvarMVStruct **AllCnstrnts,
                                       const CagdRType BaseConnectLengths[6],
                                       const CagdPType WorkDomain[2],
                                       CagdRType SubdivTol,
                                       CagdRType NumericTol)
```

AllCnstrnts: Nine constraints without base connecting length set.

BaseConnectLengths: Six specific edge-lengths of rods connection bases.

WorkDomain: The Euclidean space working domain min./max. range.

SubdivTol: Tolerance of the solution. This tolerance is measured in the parametric space of the surfaces.

NumericTol: Numeric tolerance of a possible numeric improvement stage. The numeric stage is employed if NumericTol < SubdivTol.

Returns: Solution(s) of the Qi locations, if any.

Description: Derive nine constraints for a Stewart platform, so that given the lengths of the six rods connection the bottom and top bases, we can solve the problem. The bottom base is specified and so are the lengths of the edges of the top base.
See also: MvarStewartPlatformGenEqns,

8.2.596 MvarSubDmnInfoStructFree (zrsolver.c:2895)

```
void MvarSubDmnInfoStructFree(MvarZeroSubDmnInfoStruct *InfoStruct,
                              int NumOfMVs)
```

InfoStruct: The info' structure to be freed.

NumOfMVs: Number of multi-variates held by the info' structure.

Returns: void.

Description: Deallocates a sub-domain info' structure.
See also: MvarSubDmnInfoStructNew,

8.2.597 MvarSubDmnInfoStructNew (zrsolver.c:2865)

```
MvarZeroSubDmnInfoStruct *MvarSubDmnInfoStructNew(MvarMVStruct **MVs,
                                                  MvarMVDirType ProjDir1,
                                                  MvarMVDirType ProjDir2)
```

MVs: Vector of multivariate constraints, defined in the required sub-domain.

ProjDir1: First coordinate direction of the two directions w.r.t. which the IPT succeeded.

ProjDir2: Second coordinate direction of the two directions w.r.t. which the IPT succeeded.

Returns: The new info' structure.

Description: Allocates the slots required for a sub-domain info' structure.
See also: MvarSubDmnInfoStructFree,

8.2.598 MvarSurfaceRayIntersection (ray-srf.c:47)

```
MvarPtStruct *MvarSurfaceRayIntersection(const CagdSrfStruct *Srf,
                                         const IrtPtType RayOrigin,
                                         const IrtVecType RayDir,
                                         IrtRType SubdivTol)
```

Srf: The surface to seek intersection(s) with a ray.

RayOrigin, RayDir: Parameters of ray.

SubdivTol: Subdivision tolerance of the solution.

Returns: List of all intersection locations of Srf (as UV coordinates in Srf) with ray.

Description: Computes a ray - surface intersection by finding two planes through the ray and solve for points on the surface that are on both plane (i.e. the ray). Let S(u, v) = (x(u, v), y(u, v), z(u, v)) and let the two planes be Aix + Biy + CiZ + Di = 0, i = 1,2. Then, solve for the following two algebraic constraints:

```
Aix(u, v) + Biy(u, v) + Ciz(u, v) + Di = 0, i = 1,2.
```

in two unknowns. This function returns ALL intersections of Srf and the infinite line holding the ray, from which the desired solution(s) must be filtered.
See also: MvarCrvAntipodalPoints,

8.2.599 MvarTVRglrIsNegJacobian (mvtrivar.c:1263)

```
int MvarTVRglrIsNegJacobian(const TrivTVStruct *TV)
```

TV: The trivariate to consider.

Returns: TRUE if Jacobian is inverted (negative), FALSE if positive.

Description: Test at a middle point of the given trivariate, assumed regular, is inverted and has a negative Jacobian.
See also: TrivTVEvalJacobian, MvarCalculateTVJacobian, TrivComputeJacobian,

8.2.600 MvarTVsRglrCorrectJacobian (mvtrivar.c:1295)

```
TrivTVStruct *MvarTVsRglrCorrectJacobian(TrivTVStruct *TVs)
```

TVs: List of trivariates to correct their Jacobian, in place.

Returns: An identical list of trivariates but with only positive Jacobians.

Description: Correct, in place, a given list of trivariates to have only positive Jacobians. Negative Jacobian surfaces are reversed via TrivTVReverseDir. Assumes all trivariates are regular in the interior.
See also: MvarTVRglrIsNegJacobian, TrivTVReverseDir,

8.2.601 MvarTanHyperSpheresofNManifolds (ms_sphr.c:49)

```
MvarPtStruct *MvarTanHyperSpheresofNManifolds(MvarMVStruct **MVs,
                                              int NumOfMVs,
                                              CagdRType SubdivTol,
                                              CagdRType NumericTol,
                                              CagdBType UseExprTree)
```

MVs: The manifolds to consider in R^d.

NumOfMVs: Number of multivariates we need to process.

SubdivTol, NumericTol: Of computation.

UseExprTree: TRUE to use expression trees in the computation, FALSE to use regular multivariate expressions.

Returns: List of tangent hyper-spheres, NULL if error.

Description: Computes all the hyper-spheres that are tangent to all the given set of manifolds. All manifolds should share the same range space, R^d, space in which the hyper-spheres are sought.
See also: MvarMSCircOfThreeCurves, MvarMSCircOfTwoCurves, MvarMinSpanCirc, MvarTanHyperSpheresofNManifoldsET,

8.2.602 MvarTriangleFree (zrmv2dTp.c:1224)

```
void MvarTriangleFree(MvarTriangleStruct *Tr)
```

Tr: Multivariate triangle to free.

Returns: void

Description: Deallocates and frees all slots of a multi-variate triangle structure.
See also: MvarTriangleNew, MvarTriangleFreeList,

8.2.603 MvarTriangleFreeList (zrmv2dTp.c:1245)

```
void MvarTriangleFreeList(MvarTriangleStruct *TrList)
```

TrList: Multivariate triangles list to free.

Returns: void

Description: Deallocates and frees a list of triangle structures.
See also: MvarTriangleNew, MvarTriangleFree,

8.2.604 MvarTriangleNew (zrmv2dTp.c:1185)

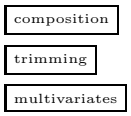
```
MvarTriangleStruct *MvarTriangleNew(int Dim)
```

Dim: Number of dimensions of each vertex.

Returns: An uninitialized triangle.

Description: Allocates the memory required for a new multi-variate triangle.
See also: MvarTriangleFree,

8.2.605 MvarTrimComposeMVSrf (mvcomps2.c:447)




```
MvarComposedSrfStruct *MvarTrimComposeMVSrf(const MvarMVStruct *MV,
                                            const CagdSrfStruct *Srfs)
```

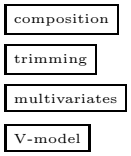
MV: The mapping multivariate.

Srfs: The surfaces to compose into MV.

Returns: The composed surfaces, represented as either trimmed surfaces, or, when possible tensor-product surfaces.

Description: Compose a list of surfaces (Srfs) into a multivariate (MV), such that the surfaces are allowed to cross knot values of the mapping multivariate. The results are returned as trimmed surfaces, except surface patches which result from only iso-parametric subdivisions, which are represented as tensor-product surfaces.
See also: MvarUntrimComposeMVSrf, MvarComposeMVMdl, MvarComposeMVVModel,

8.2.606 MvarTrimComposeMVTV (mvcomps2.c:860)




```
MvarComposedTrivStruct *MvarTrimComposeMVTV(const MvarMVStruct *MV,
                                            const TrivTVStruct *TV)
```

MV: The mapping multivariate.

TV: The trivariates to compose into MV.

Returns: The composed geometry, a mixed list of trivariates and V-models.

Description: Compose a list of trivariates (TV) into a multivariate (MV), allowing the surfaces of the models to cross knot values of the mapping multivariate.
See also: MvarComposeMVVModel,

8.2.607 MvarTrimSrfListPreciseBBox (mvarbbox.c:1031)

bbox

bounding box

```
void MvarTrimSrfListPreciseBBox(const TrimSrfStruct *TSrfs,
                                MvarBBoxStruct *BBox,
                                CagdRType Tol)
```

TSrfs: To compute a precise bounding box for.

BBox: Where bounding information is to be saved.

Tol: Tolerance of computations.

Returns: void

Description: Computes the precise bounding box for a list of trimmed surfaces.
See also: MvarSrfListPreciseBBox, MvarTrimSrfPreciseBBox, CagdSrfListBBox, , MvarSrfPreciseBBox,

8.2.608 MvarTrimSrfPreciseBBox (mvarbbox.c:973)

bbox

bounding box

```
void MvarTrimSrfPreciseBBox(const TrimSrfStruct *TSrf,
                            MvarBBoxStruct *BBox,
                            CagdRType Tol)
```

TSrf: To compute a precise bounding box for.

BBox: Where bounding information is to be saved.

Tol: Tolerance of computations.

Returns: void

Description: Computes the precise bounding box for a trimmed surface.
See also: MvarSrfPreciseBBox, MvarTrimSrfListPreciseBBox, MvarMVBBox, , MvarMVPreciseBBox, CagdSrfBBox,

8.2.609 MvarTrimSrfRayIntersect (ray-srf.c:215)

```
int MvarTrimSrfRayIntersect(const TrimSrfStruct *TrimSrf,
                            const CagdVType RayPt,
                            const CagdVType RayDir,
                            CagdUVStruct **InterPntsUV)
```

TrimSrf: The trimmed surface.

RayPt: Starting point of the ray.

RayDir: The ray's direction.

InterPntsUV: Output parameter that contains the intersection points in the UV parametric space of the surface. Can be NULL if this information is not needed.

Returns: The number of intersection points.

Description: Calculates the intersection between a ray and a trimmed surface.
See also: MvarSrfRayIntersect,

8.2.610 MvarTrisector3DCreateMVs (mvbisect.c:1373)

```
MvarMVStruct **MvarTrisector3DCreateMVs(VoidPtr FF1,
                                        VoidPtr FF2,
                                        VoidPtr FF3,
                                        CagdRType *BBoxMin,
                                        CagdRType *BBoxMax,
                                        int *Eqns)
```

FF1, FF2, FF3: Freeform objects (curve or surface) to compute their trisector.

BBoxMin, BBoxMax: The bounding box, where the trisector is computed.

Eqns: Will be updated with the number of MVs constraints.

Returns: The multivar system, its solution is the desired trisector curve in R3.

Description: Creates a set of MV constraints for the computation of a trisector curve of three objects (bspsline curves or surfaces) in R3. The trisector is considered to lie inside a spatial domain BBox.

8.2.611 MvarTrisectorCrvs (mvbisect.c:1578)

```
MvarPolylineStruct *MvarTrisectorCrvs(VoidPtr FF1,
                                      VoidPtr FF2,
                                      VoidPtr FF3,
                                      CagdRType Step,
                                      CagdRType SubdivTol,
                                      CagdRType NumericTol,
                                      CagdRType *BBoxMin,
                                      CagdRType *BBoxMax)
```

FF1, FF2, FF3: Objects to compute their trisector.

Step: Stepsize for curve tracing.

SubdivTol: The subdivision tolerance to use.

NumericTol: The numerical tolerance to use.

BBoxMin, BBoxMax: The bounding box, where the trisector is computed.

Returns: A (list of) polyline(s) that approximates the trisector curve.

Description: Computes a trisector curve of three objects (curves or surfaces). The trisector is considered to lie inside a spatial domain BBox.

8.2.612 MvarTrivJacobianImprove (mvarjimp.c:605)

```
void MvarTrivJacobianImprove(TrivTVStruct *TV,
                             CagdRType StepSize,
                             int NumIters)
```

TV: To try and improve (make more uniform) its Jacobian.

StepSize: Numerical step size to move along the gradient (at the Jacobian extreme values.)

NumIters: Number of numerical iterations to allow in the improvement process.

Returns: void

Description: Attempt the improve the Jacobian of the given trivariate by adjusting interior control-points of the TV.

See also: MvarCalculateTVJacobian, SymbSrfJacobianImprove,

8.2.613 MvarTrivarBoolOne (mvtrivar.c:73)

```
TrivTVStruct *MvarTrivarBoolOne(const CagdSrfStruct *Srf)
```

Srf: Surface to derive a trivariate volume for its interior.

Returns: A trivariate volume boolean sum of Srf.

Description: Computes the volumetric (trivariate) boolean sum as follows:

1. Divide Srf into 4 side patches, in U.
2. Compute the volumetric boolean sum for the 4 patches in 1 and 2. Two cap patches will be computed on the fly.

See also: MvarTrivarBoolSum,

8.2.614 MvarTrivarBoolSum (mvtrivar.c:168)

```
TrivTVStruct *MvarTrivarBoolSum(const CagdSrfStruct *Srf1,
                                const CagdSrfStruct *Srf2,
                                const CagdSrfStruct *Srf3,
                                const CagdSrfStruct *Srf4,
                                const CagdSrfStruct *Srf5,
                                const CagdSrfStruct *Srf6)
```

Srf1, Srf2, Srf3, Srf4, Srf5, Srf6: Surfaces to derive a trivariate volume for its interior. If Srf5 or Srf6 are NULL, they are generated as surface Boolean sum, from V boundaries of Srf1/2/3/4.

Returns: A trivariate volume boolean sum of Srf.

Description: Computes the volumetric (trivariate) boolean sum as follows: Srf1/2/3/4 are around the sweep surface and Srf5/6 are its two caps. The boolean sum is computed as follows:

```
    T0 = Trilinear(4Corners(Srf1), 4Corners(Srf3))
    T1 = RuledVolume(Srf1, Srf3)
    T2 = RuledVolume(Srf2, Srf4)
    T3 = RuledVolume(Srf5, Srf6)
    T4 = RuledVolume(RuledSrf(Bndry(Srf1, UMin), Bndry(Srf1, UMax)),
                     RuledSrf(Bndry(Srf3, UMin), Bndry(Srf3, UMax)),
    T5 = RuledVolume(RuledSrf(Bndry(Srf2, VMin), Bndry(Srf2, VMax)),
                     RuledSrf(Bndry(Srf4, VMin), Bndry(Srf4, VMax)),
    T6 = RuledVolume(RuledSrf(Bndry(Srf5, VMin), Bndry(Srf5, VMax)),
                     RuledSrf(Bndry(Srf6, VMin), Bndry(Srf6, VMax)),
And the final boolean sum equals:
   BoolSumVolume = T1 + t2 + t3 - (T4 + t5 + t6) + T0
```

See also: MvarTrivarBoolOne, MvarTrivarBoolSum2, MvarTrivarBoolSumRtnl,

8.2.615 MvarTrivarBoolSum2 (mvtrivar.c:474)

```
TrivTVStruct *MvarTrivarBoolSum2(const CagdSrfStruct *UMin,
                                 const CagdSrfStruct *UMax,
                                 const CagdSrfStruct *VMin,
                                 const CagdSrfStruct *VMax,
                                 const CagdSrfStruct *WMin,
                                 const CagdSrfStruct *WMax)
```

UMin, UMax, VMin, VMax, WMin, WMax: Surfaces to derive a trivariate volume for its interior.

Returns: A trivariate volume boolean sum of Srf.

Description: Computes the volumetric (trivariate) boolean sum as in MvarTrivarBoolSum but uses the 6 faces of a cube, as extracted from a face of an existing trivariate. In other words "TV = BoolSum2(TrivBndrySrfsFromTV(TV))".

See also: MvarTrivarBoolSum, MvarTrivarBoolOne, MvarTrivarBoolSumRtnl,

8.2.616 MvarTrivarBoolSum3 (mvtrivar.c:667)

```
TrivTVStruct *MvarTrivarBoolSum3(const CagdSrfStruct *Srf1,
                                 const CagdSrfStruct *Srf2,
                                 const CagdSrfStruct *Srf3,
                                 const CagdSrfStruct *Srf4,
                                 const CagdSrfStruct *Srf5,
                                 const CagdSrfStruct *Srf6)
```

Srf1, Srf2, Srf3, Srf4, Srf5, Srf6: Surfaces to derive a trivariate volume for its interior. If Srf5 or Srf6 are NULL, they are generated as surface Boolean sum, from V boundaries of Srf1/2/3/4. Srf5 and Srf6 are optional.

Returns: A trivariate volume boolean sum of Srf.

Description: Computes the volumetric (trivariate) boolean sum as in MvarTrivarBoolSum but reorient the surfaces automatically so they all align properly, following Srf1, that is not modified. Assume the Srf1/2/3/4 are orientable along constant U values.

See also: MvarTrivarBoolOne, MvarTrivarBoolSum2,

8.2.617 MvarTrivarBoolSumRtnl (mvtrivar.c:1448)

Boolean sum

surface constructors

```
TrivTVStruct *MvarTrivarBoolSumRtnl(const CagdSrfStruct *Srf1,
                                    const CagdSrfStruct *Srf2,
                                    const CagdSrfStruct *Srf3,
                                    const CagdSrfStruct *Srf4,
                                    const CagdSrfStruct *Srf5,
                                    const CagdSrfStruct *Srf6)
```

Srf1, Srf2, Srf3, Srf4, Srf5, Srf6: Surfaces to derive a trivariate volume for its interior. If Srf5 or Srf6 are NULL, they are generated as surface Boolean sum, from V boundaries of Srf1/2/3/4.

Returns: A trivariate volume boolean sum of Srf.

Description: Computes a volumetric (trivariate) boolean sum as in MvarTrivarBoolSum3. However, this function, for rational input srfs, will not raise the degrees at the cost of changing the internal parametrization.
See also: CagdBoolSumSrfRtnl, MvarTrivarBoolSum, MvarTrivarBoolSum3,

8.2.618 MvarTrivarCubicTVFit (mvtrivar.c:1089)

```
TrivTVStruct *MvarTrivarCubicTVFit(const TrivTVStruct *TV)
```

TV: To approximate as a tricubic.

Returns: A tricubic Bezier trivariate, fitting the corners positions and match corners tangents.

Description: Construct a tricubic Bezier trivariate that fits the input trivariate at corner points and match corner derivatives.
See also: MvarTrivarQuadraticTVFit, CagdCubicSrfFit, CagdCubicCrvFit,

8.2.619 MvarTrivarHalfBoolSum (mvtrivar.c:1674)

```
TrivTVStruct *MvarTrivarHalfBoolSum(const CagdSrfStruct *Srf1,
                                    const CagdSrfStruct *Srf2)
```

Srf1, Srf2: To compute half Boolean sum trivariate for.

Returns: A trivariate volume half Boolean sum of Srf1, Srf2.

Description: Computes a trivariate representing the half Boolean sum of the given surfaces, as follows: Srf1/2 are two surfaces having a shared boundary. Srf1/2 are reoriented so their shared boundary will be at VMin. The half Boolean sum is computed as follows:

```
C(u) = Bndry(Srf2, VMin),  (the shared boundary)
HalfBoolSumTV(u, v, w) = Srf1(u ,v) + Srf2(u, w) - C(u).
```

See also: MvarTrivarBoolSum, MvarTrivarBoolOne, MvarTrivarOneSidedBoolSum2Srfs,

8.2.620 MvarTrivarListPreciseBBox (mvarbbox.c:930)

bbox

bounding box

```
void MvarTrivarListPreciseBBox(const TrivTVStruct *Trivars,
                               MvarBBoxStruct *BBox,
                               CagdRType Tol)
```

Trivars: To compute a precise bounding box for.

BBox: Where bounding information is to be saved.

Tol: Tolerance of computations.

Returns: void

Description: Computes the precise bounding box for a list of trivariate freeform function.
See also: TrivTVListBBox, MvarTrivarPreciseBBox,

8.2.621 MvarTrivarOneSidedBoolSum2Srfs (mvtrivar.c:843)

```
TrivTVStruct *MvarTrivarOneSidedBoolSum2Srfs(const CagdSrfStruct *Srf1,
                                             const CagdSrfStruct *Srf2)
```

Srf1, Srf2: Surfaces to derive a trivariate volume that is extiror is Srf 1/2.

Returns: A trivariate volume one-sided boolean sum of Srf1/2, or NULL if error (no common curves etc.).

Description: Computes a volumetric (trivariate) one-sided Boolean sum of 2 surfaces. Srf1/2 are the faces of the generated trivariate. Srf1/2 are oriented around their meeting curve. The meeting curve will be in UMin direction in each one of them. The one-sided Boolean sum is computed as follows: T0 = TrivCnvrtSrfToTV(Srf1, TRIV_CONST_W_DIR), T1 = TrivCnvrtSrfToTV(Srf2, TRIV_CONST_U_DIR), T2 = TrivCnvrtCrvToTV(BndryCrv, TRIV_CONST_V_DIR), such that BndryCrv is the meeting curve between the two surfaces. The final sided Boolean sum equals: BoolSumVolume = T0 + T1 - T2
See also: MvarTrivarBoolSum, CagdOneSidedBoolSumSrf, , MvarTrivarOneSidedBoolSum3Srfs, MvarTrivarHalfBoolSum,

8.2.622 MvarTrivarOneSidedBoolSum3Srfs (mvtrivar.c:952)

```
TrivTVStruct *MvarTrivarOneSidedBoolSum3Srfs(const CagdSrfStruct *Srf1,
                                             const CagdSrfStruct *Srf2,
                                             const CagdSrfStruct *Srf3)
```

Srf1, Srf2, Srf3: Surfaces to derive a trivariate volume that three of its faces are Srf1/2/3.

Returns: A trivariate volume one-sided Boolean sum of Srf1/2/3, or NULL if error (no common curves etc.).

Description: Computes a volumetric (trivariate) one-sided Boolean sum of 3 surfaces. Srf1/2/3 are three faces of the generated trivariate. Srf 2/3 will be aligned properly, following Srf1, that is not modified. The common boundary curve between Srf2/3 with Srf1 will be in VMin direction. The one-sided Boolean sum is computed as follows: T0 = TrivCnvrtSrfToTV(Srf1, TRIV_CONST_W_DIR), T1 = TrivCnvrtSrfToTV(Srf2, TRIV_CONST_U_DIR), T2 = TrivCnvrtSrfToTV(Srf3, TRIV_CONST_V_DIR), T3 = TrivCnvrtCrvToTV(C1, TRIV_CONST_V_DIR), T4 = TrivCnvrtCrvToTV(C2, TRIV_CONST_U_DIR), T5 = TrivCnvrtCrvToTV(C3, TRIV_CONST_W_DIR),
such that C1 is the common boundary curve between Srf1 and Srf2, C2 is the common boudary curve between Srf1 and Srf3, and C3 is the common boundary curve between Srf2 and Srf3. C1, C2, and C3 meet at point P0. The final sided boolean sum equals: BoolSumVolume = T0 + T1 + T2 - (T3 + T4 + T5) - P0
See also: MvarTrivarSidedBoolSumTwoSrf, MvarTrivarBoolSum3, , CagdOneSidedBoolSumSrf, MvarTrivarOneSidedBoolSum2Srfs,

8.2.623 MvarTrivarPreciseBBox (mvarbbox.c:863)

bbox

bounding box

```
void MvarTrivarPreciseBBox(const TrivTVStruct *TV,
                           MvarBBoxStruct *BBox,
                           CagdRType Tol)
```

TV: To compute a precise bounding box for.

BBox: Where bounding information is to be saved.

Tol: Tolerance of computations.

Returns: void

Description: Computes the precise bounding box for a trivariate freeform function.
See also: MvarMVBBox, MvarMVPreciseBBox, TrivTVBBox,

8.2.624 MvarTrivarQuadraticTVFit (mvtrivar.c:1152)

```
TrivTVStruct *MvarTrivarQuadraticTVFit(const TrivTVStruct *TV)
```

TV: To approximate as a triquadratic.

Returns: A triquadratic Bezier trivariate, fitting the end position and approximate tangent directions.

Description: Construct a triquadratic Bezier trivariate that fits the input trivariate corners points and aims for same corners tangent directions.
See also: MvarTrivarCubicTVFit, CagdCubicSrfFit, CagdCubicCrvFit,

8.2.625 MvarTvTransInnerCtlPts2Pt (krnl_based_cnstrct.c:1614)

```
void MvarTvTransInnerCtlPts2Pt(TrivTVStruct *TV,
                               const IrtPtType Pt,
                               IrtRType Ratio)
```

TV: The modified trivariate, in place.

Pt: The point to translate all inner control points to.

Ratio: The translation ratio from zero to one.

Returns: void

Description: Translates all inner control points by (Ratio * |Pt - Cp|), where Cp is the location of the current control point.
See also:

8.2.626 MvarTwoMVsMorphing (mvarmrph.c:35)

morphing

```
MvarMVStruct *MvarTwoMVsMorphing(const MvarMVStruct *MV1,
                                 const MvarMVStruct *MV2,
                                 CagdRType Blend)
```

MV1, MV2: The two multi-variates to blend.

Blend: A parameter between zero and one.

Returns: MV2 * Blend + MV1 * (1 - Blend).

Description: Given two compatible multi-variates (See function MvarMakeMVsCompatible), computes a convex blend between them according to Blend which must be between zero and one. Returned is the new blended multi-variates.
See also: SymbTwoCrvsMorphing, SymbTwoCrvsMorphingCornerCut, , SymbTwoCrvsMorphingMultiRes, SymbTwoSrfsMorphing, TrivTwoTVsMorphing, , MvarMakeMVsCompatible,

8.2.627 MvarUniFuncsComputeLowerEnvelope (mvlowenv.c:1125)

```
void MvarUniFuncsComputeLowerEnvelope(CagdCrvStruct *InputCurves,
                                      CagdCrvStruct **LowerEnvelope)
```

InputCurves: A CagdCrvStruct of univariate (non-rational) curves in R^1

LowerEnvelope: A CagdCrvStruct of lower envelope

Returns: void

Description: Given monotone univariate function curves, and their domain, compute the lower envelope. This is the main calling function for such curves.

8.2.628 MvarUntrimComposeMVSrf (mvcomps2.c:558)

composition
untrimming
multivariates

```
CagdSrfStruct *MvarUntrimComposeMVSrf(
                                    const MvarMVStruct *MV,
                                    const CagdSrfStruct *Srfs,
                                    CagdQuadSrfWeightFuncType UntrimWeightFunc)
```

MV: The mapping multivariate.

Srfs: The surfaces to compose into MV.

UntrimWeightFunc: The weight function for the untrimming, or NULL, for applying the line-sweep algorithm.

Returns: The composed surfaces, represented as tensor-product surfaces.

Description: Compose a list of surfaces (Srfs) into a multivariate (MV), such that the surfaces are allowed to cross knot values of the mapping multivariate. The results are returned as untrimmed (tensor-product) surfaces, using the minimal-weight untrimming algorithm (with the weight provided by UntrimWeightFunc), or the line-sweep algorithm (if UntrimWeightFunc is NULL).
See also: MvarTrimComposeMVSrf, MvarComposeMVMdl, MvarComposeMVVModel,

8.2.629 MvarVecAdd (mvar_vec.c:35)

```
void MvarVecAdd(MvarVecStruct *VRes,
                const MvarVecStruct *V1,
                const MvarVecStruct *V2)
```

VRes: Result. Can be one of V1 or V2.

V1, V2: Two input vectors.

Returns: void

Description: Add two multivariate vectors.
See also: MvarVecDotProd, MvarVecSqrLength, MvarVecLength, MvarVecScale, MvarVecNormalize, MvarVecBlend,

8.2.630 MvarVecAddScale (mvar_vec.c:72)

```
void MvarVecAddScale(MvarVecStruct *VRes,
                     const MvarVecStruct *V1,
                     const MvarVecStruct *V2,
                     CagdRType Scale2)
```

VRes: Result. Can be one of V1 or V2.

V1, V2: Two input vectors.

Scale2: Scaling factor of V2.

Returns: void

Description: Add two multivariate vectors, second with scale: VRes = V1 + V2*Scale2.
See also: MvarVecDotProd, MvarVecSqrLength, MvarVecLength, MvarVecScale, MvarVecNormalize, MvarVecBlend,

8.2.631 MvarVecArrayFree (mvar_gen.c:1193)

free

```
void MvarVecArrayFree(MvarVecStruct *MVVecArray, int Size)
```

MVVecArray: To be deallocated.

Size: Of the deallocated array.

Returns: void

Description: Deallocates and frees a multi-variate vector array.
See also: MvarVecArrayNew, MvarVecFree, MvarVecNew, MvarVecFreeList,

8.2.632 MvarVecArrayNew (mvar_gen.c:1016)

```
MvarVecStruct *MvarVecArrayNew(int Size, int Dim)
```

Size: Size of multi-variate vector array to allocate.

Dim: Number of dimensions of this multi-variate.

Returns: An uninitialized multi-variate vector array.

Description: Allocates the memory required for a new multi-variate vector array.
See also: MvarVecArrayFree, MvarVecFree, MvarVecNew,

8.2.633 MvarVecBlend (mvar_vec.c:283)

```
void MvarVecBlend(MvarVecStruct *VRes,
                  const MvarVecStruct *V1,
                  const MvarVecStruct *V2,
                  CagdRType t)
```

VRes: Result. Can be one of V1 or V2.

V1, V2: Two input vectors to blend.

t: Blending factor.

Returns: void

Description: Compute the blend of the to given multivariate vectors as

```
V1 * t + V2 * (1-t).
```

See also: MvarVecAdd, MvarVecSqrLength, MvarVecLength, MvarVecScale, MvarVecNormalize, MvarVecDotProd,

8.2.634 MvarVecCmpTwoVectors (mvar_pll.c:344)

```
int MvarVecCmpTwoVectors(const CagdRType *V1,
                         const CagdRType *V2,
                         int Length,
                         CagdRType Eps)
```

V1, V2: Two vectors to compare.

Length: Of the two vectors.

Eps: The tolerance of the comparison.

Returns: 0 if identical, -1 or +1 if first point is less than/greater than second point, in lexicographic order over dimensions. 2 is returned if the dimensions are different.

Description: A comparison function to examine if the given two vectors are the same.
See also: MvarPtDistTwoPoints, MvarPtDistSqrTwoPoints, MvarPtCmpTwoPoints,

8.2.635 MvarVecCopy (mvar_gen.c:1108)

```
MvarVecStruct *MvarVecCopy(const MvarVecStruct *Vec)
```

Vec: Multi-Variate vector to duplicate.

Returns: Duplicated multi-variate vector.

Description: Allocates and duplicates all slots of a multi-variate vector structure.

8.2.636 MvarVecCopyList (mvar_gen.c:1133)

multi-variates

```
MvarVecStruct *MvarVecCopyList(const MvarVecStruct *VecList)
```

VecList: List of multi-variates to duplicate.

Returns: Duplicated list of multi-variates.

Description: Duplicates a list of multi-variate structures.

8.2.637 MvarVecDotProd (mvar_vec.c:139)

```
CagdRType MvarVecDotProd(const MvarVecStruct *V1, const MvarVecStruct *V2)
```

V1, V2: Two input vectors.

Returns: The dot product.

Description: Compute the dot product of two multivariate vectors.
See also: MvarVecAdd, MvarVecSqrLength, MvarVecLength, MvarVecScale, MvarVecNormalize, MvarVecBlend,

8.2.638 MvarVecFree (mvar_gen.c:1165)

```
void MvarVecFree(MvarVecStruct *Vec)
```

Vec: Multivariate vector to free.

Returns: void

Description: Deallocates and frees all slots of a multi-variate vector structure.
See also: MvarVecArrayNew, MvarVecArrayFree, MvarVecNew, MvarVecFreeList,

8.2.639 MvarVecFreeList (mvar_gen.c:1223)

```
void MvarVecFreeList(MvarVecStruct *VecList)
```

VecList: Multi-Variate vector list to free.

Returns: void

Description: Deallocates and frees a list of multi-variate vector structures.
See also: MvarVecArrayNew, MvarVecArrayFree, MvarVecNew, MvarVecFree,

8.2.640 MvarVecLength (mvar_vec.c:225)

```
CagdRType MvarVecLength(const MvarVecStruct *V)
```

V: Vector to compute its length.

Returns: Computed length.

Description: Compute the length of a multivariate vector.
See also: MvarVecAdd, MvarVecDotProd, MvarVecSqrLength, MvarVecScale, MvarVecNormalize, MvarVecBlend,

8.2.641 MvarVecNew (mvar_gen.c:978)

```
MvarVecStruct *MvarVecNew(int Dim)
```

Dim: Number of dimensions of this multi-variate.

Returns: An uninitialized multi-variate vector.

Description: Allocates the memory required for a new multi-variate vector.
See also: MvarVecArrayNew, MvarVecFree,

8.2.642 MvarVecNormalize (mvar_vec.c:316)

```
int MvarVecNormalize(MvarVecStruct *V)
```

V: Vector to normalize.

Returns: TRUE if successful, FALSE if the input is the ZERO vector.

Description: Normalize a given multivariate vector to a unit length, in place.
See also: MvarVecAdd, MvarVecDotProd, MvarVecSqrLength, MvarVecLength, , MvarVecScale,

8.2.643 MvarVecOrthogonal2 (mvar_vec.c:393)

```
int MvarVecOrthogonal2(MvarVecStruct *Dir,
                       const MvarVecStruct *Vec1,
                       const MvarVecStruct *Vec2)
```

Dir: Newly computed unit vector will be kept here.

Vec1, Vec2: Two vectors we must be orthogonal to. Assumed unit length.

Returns: TRUE if successful, FALSE otherwise.

Description: Derives a unit vector Dir that is orthogonal to both Vec1, and Vec2. Note that in R^2 there is no such vector, in R^3 only one such vector and in R^n, n > 3, there are infinitly many of which we find one.
See also: MvarVecOrthogonalize, MvarVecSetOrthogonalize, MvarVecWedgeProd,

8.2.644 MvarVecOrthogonalize (mvar_vec.c:347)

```
int MvarVecOrthogonalize(MvarVecStruct *Dir, const MvarVecStruct *Vec)
```

Dir: Vector to update in place so it will be orthgoonal to Vec.

Vec: Vector to make sure Dir is made orthogonal to.

Returns: TRUE if successful, FALSE otherwise.

Description: Updates Dir to be the closest vector to Dir that is orthogonal to Vec. In essence, apply a Graham Shmidt step. Vectors need not be unit size.
See also: MvarVecOrthogonal2, MvarVecSetOrthogonalize, MvarVecWedgeProd,

8.2.645 MvarVecRealloc (mvar_gen.c:1062)

```
MvarVecStruct *MvarVecRealloc(MvarVecStruct *Vec, int NewDim)
```

Vec: Multi-Variate vector to reallocate. Should not be used after this operation as it might be freed.

NewDim: Number of new dimensions of this multi-variate vector.

Returns: A reallocated multi-variate vector.

Description: Reallocates the memory that is required for a new dimension of a multi-variate vector.
See also: MvarVecNew,

8.2.646 MvarVecScale (mvar_vec.c:248)

```
MvarVecStruct *MvarVecScale(MvarVecStruct *V, CagdRType ScaleFactor)
```

V: Vector to scale, in place.

ScaleFactor: Scaling factor to use.

Returns: The input vector, scaled.

Description: Scale a given multivariate vector V by a scaling factor ScaleFactor.
See also: MvarVecAdd, MvarVecDotProd, MvarVecSqrLength, MvarVecLength, , MvarVecNormalize, MvarVecBlend,

8.2.647 MvarVecSetOrthogonalize (mvar_vec.c:473)

```
int MvarVecSetOrthogonalize(const MvarVecStruct **Vecs,
                            MvarVecStruct **OrthoVecs,
                            int Size)
```

Vecs: Input vectors to make orthonormal.

OrthoVecs: Output vectors that span the same (sub) space as vec but are orthogonal and unit length (orthonormal). Can be the same as Vecs.

Size: Number of vectors in Vecs and OrthoVecs vectors of vectors.

Returns: TRUE if successful, FALSE otherwise.

Description: Update the given set of vectors to be of unit size and orthogonal to each other. Vectors are all assumed of the same dimension. In essence, apply a Graham Shmidt to all vectors.
See also: MvarVecOrthogonal, MvarVecOrthogonal2, MvarVecWedgeProd,

8.2.648 MvarVecSortAxis (mvar_int.c:140)

```
MvarVecStruct *MvarVecSortAxis(MvarVecStruct *VecList, int Axis)
```

VecList: List of points to sort.

Axis: Axis to sort along: 1,2,3 for X,Y,Z.

Returns: Sorted list of points, in place.

Description: Sorts given list of points based on their increasing order in axis Axis. Sorting is done in place.

8.2.649 MvarVecSqrLength (mvar_vec.c:173)

```
CagdRType MvarVecSqrLength(const MvarVecStruct *V)
```

V: Vector to compute its length.

Returns: Computed length squared.

Description: Compute the length squared of a multivariate vector.
See also: MvarVecAdd, MvarVecDotProd, MvarVecLength, MvarVecScale, MvarVecNormalize, MvarVecBlend, MvarVecSqrLength2,

8.2.650 MvarVecSqrLength2 (mvar_vec.c:196)

```
CagdRType MvarVecSqrLength2(const CagdRType *v, int Dim)
```

v: Vector to compute its length.

Dim: Lentgth of vector v.

Returns: Computed length squared.

Description: Compute the length squared of a multivariate vector.
See also: MvarVecAdd, MvarVecDotProd, MvarVecLength, MvarVecScale, MvarVecNormalize, MvarVecBlend, MvarVecSqrLength,

8.2.651 MvarVecSub (mvar_vec.c:107)

```
void MvarVecSub(MvarVecStruct *VRes,
                const MvarVecStruct *V1,
                const MvarVecStruct *V2)
```

VRes: Result. Can be one of V1 or V2.

V1, V2: Two input vectors.

Returns: void

Description: Subtract two multivariate vectors.
See also: MvarVecDotProd, MvarVecSqrLength, MvarVecLength, MvarVecScale, MvarVecNormalize, MvarVecBlend, MvarVecAdd,

8.2.652 MvarVecWedgeProd (mvar_vec.c:573)

```
CagdBType MvarVecWedgeProd(MvarVecStruct **Vectors,
                           int Size,
                           MvarVecStruct **NewVecs,
                           int NewSize,
                           CagdBType CheckDet,
                           CagdRType *DetVal)
```

Vectors: The set of Size vectors, each of dimension larger than Size. This set of vectors is modified in place - it is made orthonormal.

Size: The size of Vectors set.

NewVecs: New vectors to allocate into here and update with orthogonal complement subspace of dimension newSize-Size. Should be able to hold NewSize-Size pointers to vectors.

NewSize: The new size of the set Vectors and NewVecs together.

CheckDet: When Size + NewSize == Dim, this flag indicates if the determinant of the Dim vectors is to be evaluated. Useful for orientation issues.

DetVal: Output value, the computed determinant if CheckDet is TRUE.

Returns: TRUE if successful, FALSE otherwise.

Description: Computes an orthogonal complement (wedge product) of dimension NewSize-Size to the given set of Size vectors. Vectors is set of vectors that are in a higher dimensional linear space, of at list dimension NewSize.

See also: MvarVecOrthogonal, MvarVecOrthogonal2, MvarVecSetOrthogonalize,

8.2.653 MvarVoronoiCrvCopy (mvvorcrv.c:53)

copy

```
MvarVoronoiCrvStruct *MvarVoronoiCrvCopy(MvarVoronoiCrvStruct *Crv)
```

Crv: To be copied.

Returns: A duplicate of Crv.

Description: Allocates and copies all slots of a MvarVoronoiCrvStruct structure.

8.2.654 MvarVoronoiCrvFree (mvvorcrv.c:93)

allocation

```
void MvarVoronoiCrvFree(MvarVoronoiCrvStruct *Crv)
```

Crv: Voronoi curve structure to free.

Returns: void

Description: Deallocates a VoronoiCrv structure.

8.2.655 MvarVoronoiCrvFreeList (mvvorcrv.c:117)

allocation

```
void MvarVoronoiCrvFreeList(MvarVoronoiCrvStruct *CrvList)
```

CrvList: Voronoi curve list to free.

Returns: void

Description: Deallocates a VoronoiCrv list structure.

8.2.656 MvarVoronoiCrvNew (mvvorcrv.c:24)

allocation

```
MvarVoronoiCrvStruct *MvarVoronoiCrvNew(void)
```

Returns: A VoronoiCrv structure.

Description: Allocates and resets all slots of a MvarVoronoiCrvStruct structure.

8.2.657 MvarVoronoiCrvReverse (mvvorcrv.c:141)

```
MvarVoronoiCrvStruct *MvarVoronoiCrvReverse(MvarVoronoiCrvStruct *Crv)
```

Crv: To be reversed.

Returns: A single duplicate of Crv that is reversed.

Description: Reverses the t and r parameters of an MvarVoronoiCrvStruct structure.

8.2.658 MvarZero0DNumeric (zrmv0d.c:1178)

```
MvarPtStruct *MvarZero0DNumeric(MvarPtStruct *ZeroPt,
                                const MvarExprTreeEqnsStruct *Eqns,
                                MvarMVStruct const * const *MVs,
                                int NumMVs,
                                CagdRType NumericTol,
                                int SuccessOnDmnErr,
                                const CagdRType *InputMinDmn,
                                const CagdRType *InputMaxDmn)
```

ZeroPt: Approximated solution, derived from a subdivision process, to improve in place.

Eqns: The constraints are given as Equations, if not NULL.

MVs: Alternatively, the constraints are given as MVS.

NumMVs: If MVs is not NULL, this specifies size of the MVs vector.

NumericTol: Tolerance of the numerical process. Tolerance is measured in the deviation of the scalar multivariates from their equality. Inequalities are ignored here. Points that fail to improve numerically are purged away.

SuccessOnDmnErr: TRUE to successfully terminate this process when domain is within tolerance. FALSE to terminate based on range values.

InputMinDmn, InputMaxDmn: Optional domain restriction. Can be NULL in which, the MVS/Eqns domains are used.

Returns: List of points on the solution set. Dimension of the points will be the same as the dimensions of all MVs. Points that failed to improve are purged away.

Description: Apply a numerical improvement stage, as a first order minimization procedure of gradient computation and marching, in place.

See also: MvarMVsZeros, MvarExprTreesZeros, MvarExprTreeEqnsZeros, , MvarMVsZeros0DNumeric,

8.2.659 MvarZeroC1DiscontSubdiv (zrsolver.c:1983)

```
static MvarZeroPrblmStruct **MvarZeroC1DiscontSubdiv(
                                                MvarZeroPrblmStruct *Problem)
```

Problem: The zero finding problem structure.

Returns: The array of two sub-problems.

Description: A preliminary recursive stage of the subdivision solver, invoked by the general solver - subdivides at the C1 discontinuity and solves recursively.

See also: MvarZeroSolverInseparableProblem,

8.2.660 MvarZeroEqnsHasMeshC1Discont (zrsolver.c:2062)

```
CagdBType MvarZeroEqnsHasMeshC1Discont(MvarMVStruct * const *MVs,
                                       int NumOfMVs,
                                       int *JLoc,
                                       CagdRType *t)
```

MVs: The system of equations.

NumOfMVs: The number of equations.

JLoc: The direction in the domain with C1 discont, if has one.

t: The parameter at the C1 discont. if has one.

Returns: TRUE if found a C1 discont., FALSE otherwise.

Description: Searches all MVs for parameter locations that are C1 discont. If found update JLoc and t to this finding and returns TRUE. Same functionality as MvarSSIHasC1Discont, but for any number of MVs, not necessarily Dim - 1.

8.2.661 MvarZeroFilterSolutionSet (zrmvaux0.c:998)

```
MvarPtStruct *MvarZeroFilterSolutionSet(MvarZeroPrblmStruct *Problem,
                                        MvarPtStruct *MVPts,
                                        const MvarMVStruct * const *MVs,
                                        const MvarConstraintType *Constraints,
                                        int NumOfMVs,
                                        IrtRType Tol,
                                        int CanHaveLoops,
                                        int SortSol,
                                        CagdBType InEqOnly,
                                        int SolDim)
```

Problem: Optional reference to the problem, in case we like to activate a call back verification function.

MVPts: Solution points to filter.

MVs: Constraints to test the solution points against.

Constraints: Type of constraints.

NumOfMVs: Also number of constraints.

Tol: Tolerance the solution point must satisfy.

CanHaveLoops: TRUE if point data can form loops in which case first and last point will be identical (and should not be purged).

SortSol: TRUE to sort solution set first, based on the 1st axis.

InEqOnly: TRUE if inequality constraints should be checked only, FALSE if both equality and inequality should be checked.

SolDim: Solutions' dimensions (0 for points, 1 for univariates, etc.).

Returns: Filtered points that satisfy the given constraints.

Description: Filters and purge solution points that are not satisfying some constraint in MVs. *
See also: MvarZeroSolver,

8.2.662 MvarZeroFirstSmoothUpdatesExpTr (zrsolver.c:2383)

```
CagdBType MvarZeroFirstSmoothUpdatesExpTr(MvarZeroPrblmStruct *Problem)
```

Problem: The zero finding problem structure.

Returns: TRUE if updates were successful, false otherwise.

Description: Completes the construction of the ETs problem structure with the actions that are performed only once: when attaining a C1 smooth problem for the first time. Adds the gradients and the current MVs, later to be used in numeric step, and solves the problem on the boundary of the domain.

8.2.663 MvarZeroFirstSmoothUpdatesMVs (zrsolver.c:2240)

```
CagdBType MvarZeroFirstSmoothUpdatesMVs(MvarZeroPrblmStruct *Problem)
```

Problem: The zero finding problem structure.

Returns: TRUE if updates were successful, FALSE otherwise.

Description: Completes the construction of the MVs problem structure with the actions that are performed only once: when attaining a C1 smooth problem for the first time, adds the gradients and the current MVs, later to be used in numeric step, and solves the problem on the boundary of the domain.

8.2.664 MvarZeroGenPtMidDmn (zrsolver.c:2438)

```
MvarPtStruct *MvarZeroGenPtMidDmn(const MvarZeroPrblmStruct *Problem,
                                  int SingleSol)
```

Problem: To construct a point in the middle of its domain.

SingleSol: If TRUE, this point is a single solution it MV domain.

Returns: The construct point in the middle of MV.

Description: Construct a point of the dimension as the given problem in the middle of its parametric domain.
See also: MvarMVsZeros,

8.2.665 MvarZeroGetRootsByKantorovich (zrmvkant.c:711)

```
MvarPtStruct *MvarZeroGetRootsByKantorovich(MvarMVStruct **MVs,
                                            MvarConstraintType *Constraints,
                                            int NumOfMVs,
                                            int NumOfZeroMVs,
                                            int ApplyNormalConeTest,
                                            CagdRType SubdivTol,
                                            int Depth,
                                            CagdBType SameSpace,
                                            CagdRType ParamPerturb)
```

MVs: Vector of multivariate constraints.

Constraints: Either an equality or an inequality type of constraint.

NumOfMVs: Size of the MVs and Constraints vector.

NumOfZeroMVs: Number of zero or equality constraints.

ApplyNormalConeTest: TRUE to apply normal cones' single intersection tests.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

Depth: Of subdivision recursion.

SameSpace: True if all MVs share the same function space.

ParamPerturb: The perturbation to applly to mid. subdivision location.

Returns: List of points on the solution set. Dimension of the points will be the same as the dimensions of all MVs. NULL is returned.

Description: Handle all aspects of root-uniqueness using Kantorovich; return a candidate point where a unique root exists, and find the rest of the roots recursively, using MvarZeroMVsSubdiv.
See also: MvarMVsZeros,

8.2.666 MvarZeroMVConstraintFail (zrmvaux0.c:1171)

```
CagdBType MvarZeroMVConstraintFail(const MvarMVStruct *MV,
                                   MvarConstraintType Constraint)
```

MV: Multivariate to examine.

Constraint: Type of constraint - zero, pos., neg.

Returns: TRUE if constraint cannot be satisfied, FALSE otherwise.

Description: Test if the given multivariate may satisfy the constraint. Examines the positivity/negativity of all coefficients in multivariate.
See also: MvarZeroSolver,

8.2.667 MvarZeroMVsSubdiv (zrmvkant.c:847)

```
static MvarPtStruct *MvarZeroMVsSubdiv(MvarMVStruct **MVs,
                                       MvarConstraintType *Constraints,
                                       int NumOfMVs,
                                       int NumOfZeroMVs,
                                       int ApplyNormalConeTest,
                                       CagdRType SubdivTol,
                                       int Depth,
                                       CagdBType SameSpace,
                                       CagdRType ParamPerturb)
```

MVs: Vector of multivariate constraints.

Constraints: Either an equality or an inequality type of constraint.

NumOfMVs: Size of the MVs and Constraints vector.

NumOfZeroMVs: Number of zero or equality constraints.

ApplyNormalConeTest: TRUE to apply normal cones' single intersection tests.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

Depth: Of subdivision recursion.

SameSpace: True if all MVs share the same function space.

ParamPerturb: The perturbation to applly to mid. subdivision location.

Returns: List of points on the solution set. Dimension of the points will be the same as the dimensions of all MVs.

Description: Approximate a solution to the set of constraints, if any, using the subdivision of the parametric domains of the MVs. Stops when the parametric domain is smaller than SubdivTol in all dimensions and returns a central point to that small multivariate patch. NOTE: The generic solver invokes this function only via the Kantorovich option (global flag set accordingly), otherwise this function is unused.
See also: MvarMVsZeros,

8.2.668 MvarZeroOrganizeETs0DProblem (zret0d.c:123)

```
MvarExprTreeEqnsStruct *MvarZeroOrganizeETs0DProblem(
                                       const MvarExprTreeStruct * const *MVETs,
                                       int NumOfMVETs)
```

MVETs: Input mvar expression tree equations.

NumOfMVETs: Number of input mvar expression tree equations, in MVETs.

Returns: Build set of equations with common exprs.

Description: Copy the given expression trees and process and fetch common expressions out to a separated common expressions' vector, all within the returned expression tree equations structure.
See also: MvarMVsZeros,

8.2.669 MvarZeroOrganizeMVs0DProblem (zrmv0d.c:105)

```
MvarMVStruct **MvarZeroOrganizeMVs0DProblem(const MvarMVStruct * const *MVs,
                                            MvarConstraintType *Constraints,
                                            int *NumOfMVs)
```

MVs: Array of multivariates.

Constraints: Equality or inequality constraints.

NumOfMVs: Number of constraints (may be updated).

Returns: The updated array of MVs to be used.

Description: Preliminary organization and scaling actions of the objects composing a zero finding problem. Relevant to the MVs representation only. This routine is invoked by the problem structure construction routine, and should not be invoked when extracting a sub-problem from an existing one.
See also: MvarZeroSolverSetCallbackFcns0DMVs,

8.2.670 MvarZeroOrganizeMVs1DProblem (zrmv1d.c:109)

```
MvarMVStruct **MvarZeroOrganizeMVs1DProblem(const MvarMVStruct * const *MVs,
                                            MvarConstraintType *Constraints,
                                            int *NumOfMVs)
```

MVs: Array of multivariates.

Constraints: Equality or inequality constraints.

NumOfMVs: Number of constraints (may be updated).

Returns: The updated array of MVs to be used.

Description: Some preliminary organization of the objects composing an MVs, 1D solution zero finding problem. This routine is invoked by the problem structure construction routine, and should not be invoked when extracting a sub-problem from an existing one.
See also: MvarZeroSolverSetCallbackFcns1DMVs,

8.2.671 MvarZeroOrganizeMVs2DProblem (zrmv2dTp.c:149)

```
MvarMVStruct **MvarZeroOrganizeMVs2DProblem(const MvarMVStruct * const *MVs,
                                            MvarConstraintType *Constraints,
                                            int *NumOfMVs)
```

MVs: Array of multivariates.

Constraints: Equality or inequality constraints.

NumOfMVs: Number of constraints (may be updated).

Returns: The updated array of MVs to be used.

Description: Some preliminary organization of the objects composing an MVs, 1D solution zero finding problem. This routine is invoked by the problem structure construction routine, and should not be invoked when extracting a sub-problem from an existing one.
See also: MvarZeroSolverSetCallbackFcns2DMVs,

8.2.672 MvarZeroSolveMatlabEqns (zrmatlab.c:52)

```
MvarZeroSolutionStruct *MvarZeroSolveMatlabEqns(
                                             MvarMatlabEqStruct **Eqns,
                                             int NumOfEqns,
                                             int MaxVarsNum,
                                             CagdRType *MinDmn,
                                             CagdRType *MaxDmn,
                                             CagdRType NumericTol,
                                             CagdRType SubdivTol,
                                             CagdRType StepTol,
                                             MvarConstraintType *Constraints)
```

Eqns: The constraints as recieved after the matlab parsing.

NumOfEqns: The number of constraints.

MaxVarsNum: The maximal number of unknowns appearing in the problem.

MinDmn: The min end point of the domain in all directions. If NULL, considered as all zeros.

MaxDmn: The max end point of the domain in all directions. If NULL, considered as all ones.

NumericTol: The required numeric tolerance of the solution.

SubdivTol: The subdivision tolerance.

StepTol: The step size for numeric tracing of curves.

Constraints: A vector of constraints specifying equality/inequality.

Returns: The solution to the problem.

Description: Convert a zero finding prolem from matlab form to the solver's problem and solve.
See also: MvarZeroSolver,

8.2.673 MvarZeroSolver (zrsolver.c:1814)

```
MvarZeroSolutionStruct *MvarZeroSolver(MvarZeroPrblmStruct *Problem)
```

Problem: The zero finding problem to be solved.

Returns: The point set (zero dimensional case) or piecewise linear approximation to the solution manifold (one/two dimensional case).

Description: This is a wrapper function for combining the zero set solver with the constraint system decomposition step. It tests the problem for criteria that determine that can't be decomposed, and based on the results it calls either MvarZeroSolverWithDecomposition (which uses constraint decomposition to solve the problem) or MvarZeroSolverInseparableProblem (which solves the problem as is).
See also: MvarZeroSolverInseparableProblem, MvarZeroSolverWithDecomposition,

8.2.674 MvarZeroSolverGetDmnDim (zrsolver.c:1951)

```
int MvarZeroSolverGetDmnDim(const MvarZeroPrblmStruct *Problem)
```

Problem: The zero finding problem structure.

Returns: The dimension of the domain of the problem.

Description: Extraction of the domain dimension of the MVs/ETs/other constraints in a zero finding problem.

8.2.675 MvarZeroSolverInseparableProblem (zrsolver.c:1481)

```
MvarZeroSolutionStruct *MvarZeroSolverInseparableProblem(MvarZeroPrblmStruct
                                                         *Problem)
```

Problem: The zero finding problem to be solved.

Returns: The point set (zero dimensional case) or piecewise linear approximation to the solution manifold (one/two dimensional case).

Description: The general zero finding problem solver. Invokes various zero finding algorithms according to the function representations/solution set dim. The solver follows as generally as possible the following paradigm:

0. Try to rule out the possibility of a solution by examining the sign of all coefficients. .
1. Recursively subdivide until the problem is C1 smooth.
2. If can't rule out, Check if the topology is guaranteed.
3. If topology is guaranteed, improve/reconstruct numerically.
4. If not, or if numeric step failed subdivide and solve recursively. This function is typically invoked when the constraints are inseparable meaning all constraints depends on all variables, and the problem cannot be split into two or more sub-problems, with less variables each.

See also: MvarMVDDecomposeProblem, MvarMVDMVSolvePlan, , MvarZeroSolverWithDecomposition,

8.2.676 MvarZeroSolverIsMVZero (zrsolver.c:2924)

```
CagdBType MvarZeroSolverIsMVZero(MvarMVStruct *MV,
                                 CagdRType NumericTol)
```

MV: The (scalar valued) multi-variate to be tested.

NumericTol: The tolerance under which a coefficient is considered zero.

Returns: TRUE if all zeros, FALSE otherwise.

Description: Tests if all coefficients of a multi-variate are zero, up to prescribed tolerance.
See also: MvarZeroMVConstraintFail,

8.2.677 MvarZeroSolverPolyProject (zrmv2dTp.c:1277)

```
MvarPolylineStruct *MvarZeroSolverPolyProject(MvarPolylineStruct *PolyList,
                                              int *Coords,
                                              int ProjDim)
```

PolyList: A list of polylines to project.

Coords: The required projection coordinates. Ordered vector.

ProjDim: The dimension of the required result. Also the length of Coords.

Returns: The new list of polylines.

Description: Projecting a given list of polylines on any of its required coordinates, as specified by the Coords vector of length projDim which is a subset of input dimension.

8.2.678 MvarZeroSolverPrblmFree (zrsolver.c:1261)

```
void MvarZeroSolverPrblmFree(MvarZeroPrblmStruct *Problem)
```

Problem: Problem structure to free.

Returns: void

Description: Deallocates and frees all slots of a problem structure of a zero finding problem. Constraints are never freed, since they are allocated externally. Note that some slots are freed only at depth zero, some are freed only at first C1 smooth instance.
See also: MvarZeroSolverPrblmNew, MvarZeroSolverSubProblem,

8.2.679 MvarZeroSolverPrblmNew (zrsolver.c:821)

```
MvarZeroPrblmStruct *MvarZeroSolverPrblmNew(const MvarMVStruct * const *MVs,
                                            const MvarExprTreeStruct * const
                                                                    *ETs,
                                            int NumOfConstraints,
                                            MvarConstraintType *Constraints,
                                            CagdRType SubdivTol,
                                            CagdRType NumericTol,
                                            CagdRType StepTol,
                                            CagdBType Solve2DBy0D)
```

MVs: Array of multivariates, NULL for ETs representations. MVs[0] can optionally hold a string "_NoSubdivDirs" attribute that sets the direction(s) not to subdivide in. I.e. '11' means the two least (UV) directions are not to be subdivided at.

ETs: Array of expression trees, NULL for MVs representations.

NumOfConstraints: Number of constraints in the problem (Total).

Constraints: Either an equality or an inequality type of constraint.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the constraints.

NumericTol: Tolerance of the numeric stage.

StepTol: In 1D numeric stages (curve tracing)- the step size used.

Solve2DBy0D: A temporary flag to indicate the use of the new 2D solver or not.

Returns: The new problem structure or NULL if error.

Description: Construction of a new zero finding problem structure. Allocates the memory and assigns slots that are already known upon construction. This routine should NOT be used when extracting a sub-problem from an existing problem (such as after domain reduction or subdivision): it should only be called at very first construction (SubdivDepth == 0). Scaling and organization takes place here that is not required for sub-problems.
See also: MvarZeroSolverPrblmFree,

8.2.680 MvarZeroSolverSetCallbackFcns0DExpTr (zret0d.c:93)

```
void MvarZeroSolverSetCallbackFcns0DExpTr(MvarZeroPrblmStruct *Problem)
```

Problem: The zero finding problem to be solved.

Returns: void

Description: Sets the callback functions for the Expression Trees representation, 0D solution case.
See also:

8.2.681 MvarZeroSolverSetCallbackFcns0DMVs (zrmv0d.c:73)

```
void MvarZeroSolverSetCallbackFcns0DMVs(MvarZeroPrblmStruct *Problem)
```

Problem: The zero finding problem to be solved.

Returns: void

Description: Sets the callback functions for the MVs representation, 0D solution case.
See also: MvarZeroOrganizeMVs0DProblem,

8.2.682 MvarZeroSolverSetCallbackFcns1DExpTr (zret1d.c:62)

```
void MvarZeroSolverSetCallbackFcns1DExpTr(MvarZeroPrblmStruct *Problem)
```

Problem: The zero finding problem to be solved.

Returns: void

Description: Sets the callback functions for the ETs representation, 1D solution case.
See also:

8.2.683 MvarZeroSolverSetCallbackFcns1DMVs (zrmv1d.c:77)

```
void MvarZeroSolverSetCallbackFcns1DMVs(MvarZeroPrblmStruct *Problem)
```

Problem: The zero finding problem to be solved.

Returns: void

Description: Sets the callback functions for the MVs representation, 1D solution case.
See also: MvarZeroOrganizeMVs1DProblem,

8.2.684 MvarZeroSolverSetCallbackFcns2DExpTr (zret2d.c:62)

```
void MvarZeroSolverSetCallbackFcns2DExpTr(MvarZeroPrblmStruct *Problem)
```

Problem: The zero finding problem to be solved.

Returns: void

Description: Sets the callback functions for the ETs representation, 2D solution case.
See also:

8.2.685 MvarZeroSolverSetCallbackFcns2DMVs (zrmv2dTp.c:117)

```
void MvarZeroSolverSetCallbackFcns2DMVs(MvarZeroPrblmStruct *Problem)
```

Problem: The zero finding problem to be solved.

Returns: void

Description: Sets the callback functions for the MVs representation, 2D solution case.

8.2.686 MvarZeroSolverSolutionFree (zrsolver.c:1421)

```
void MvarZeroSolverSolutionFree(MvarZeroSolutionStruct *Solution,
                                CagdBType FreeUnion)
```

Solution: Solution structure to free.

FreeUnion: If TRUE, the points/polylines stored at in the solution are freed, otherwise they are not. This is useful for the cases of uniting solutions in place.

Returns: void

Description: Deallocates and frees all slots of a solution structure of a zero finding problem. NOTE: the T-Junction list is not freed, as it is always freed as part of the problem deallocation.
See also: MvarZeroSolverSolutionNew,

8.2.687 MvarZeroSolverSolutionNew (zrsolver.c:1360)

```
MvarZeroSolutionStruct *MvarZeroSolverSolutionNew(
                                            MvarTriangleStruct *Tr,
                                            MvarPolylineStruct *Pl,
                                            MvarPtStruct *Pt,
                                            MvarZrSlvrRepresentationType Rep)
```

Tr: A list of triangles, or NULL if the representation is not such.

Pl: A list of polylines, or NULL if the representation is not such.

Pt: A list of points, or NULL if the representation is not such.

Rep: The active representation required for the new solution.

Returns: The new solution or NULL if error.

Description: Allocates the memory required for a new solution structure of a zero finding problem.
See also: MvarZeroSolverSolutionFree,

8.2.688 MvarZeroSolverSubProblem (zrsolver.c:1165)

```
MvarZeroPrblmStruct *MvarZeroSolverSubProblem(
                                        MvarZeroPrblmStruct const *Problem,
                                        MvarMVStruct **MVs,
                                        MvarExprTreeEqnsStruct *Eqns,
                                        MvarZeroSolutionStruct *BoundarySol)
```

Problem: The original problem, from which to extract the sub-problem.

MVs: Array of multivariates, NULL if not MVs representation.

Eqns: Struct of expression trees, NULL if not ETs representation.

BoundarySol: The solution to the corresponding 2 * Dim problems of one dimension less, defined on the boundary of the new, smaller problem.

Returns: The sub-problem.

Description: Construction of a new zero finding sub-problem structure from an existing problem, for the given constraints defined on the sub-domain (the result of subdivision or domain reduction). Note that there are slots that are only allocated once, while some are are allocated at all depths.
See also: MvarZeroSolverPrblmNew, MvarZeroOrganizeMVs0DProblem, , MvarZeroOrganizeMVs1DProblem,

8.2.689 MvarZeroSolverWithDecomposition (zrdcm_main.c:1177)

```
MvarZeroSolutionStruct *MvarZeroSolverWithDecomposition(
                                            MvarZeroPrblmStruct *Problem)
```

Problem: The constraint problem to be solved.

Returns: The solution to the problem.

Description: Given a constrain problem, the function determines if constraint system decomposition can be applied to it, and then attempts to solve it. If constraint system decomposition is not possible, it uses the regular MvarZeroSolverInseparableProblem. Currently, only well-constrained systems with no inequality constraints can be solved with the decomposition step.
See also: MvarMVDDecomposeProblem, MvarMVDMVSolvePlan, , MvarZeroSolverInseparableProblem,

8.2.690 MvarZeroTJCopy (zrmv2dTJ.c:93)

```
MvarZeroTJunctionStruct *MvarZeroTJCopy(const MvarZeroTJunctionStruct *TJ)
```

TJ: The T-Junction object to copy.

Returns: The T-Junction copy.

Description: Copies a multi-variate T-Junction object.
See also: MvarZeroTJFree,

8.2.691 MvarZeroTJCopyList (zrmv2dTJ.c:126)

```
MvarZeroTJunctionStruct *MvarZeroTJCopyList(
                                    const MvarZeroTJunctionStruct *TJList)
```

TJList: The T-Junction list to copy.

Returns: The T-Junctions list copy.

Description: Copies a list of multi-variate T-Junction objects.
See also: MvarZeroTJFree, MvarZeroTJCopy,

8.2.692 MvarZeroTJFree (zrmv2dTJ.c:158)

```
void MvarZeroTJFree(MvarZeroTJunctionStruct *TJ)
```

TJ: Multivariate T-Junction to free.

Returns: void

Description: Frees all slots of a multi-variate T-Junction structure.
See also: MvarZeroTJFreeList,

8.2.693 MvarZeroTJFreeList (zrmv2dTJ.c:188)

```
void MvarZeroTJFreeList(MvarZeroTJunctionStruct *TJList)
```

TJList: Multivariate T-Junctions list to free.

Returns: void

Description: Deallocates and frees a list of T-Junction structures.
See also: MvarZeroTJFree,

8.2.694 MvarZeroTJNew (zrmv2dTJ.c:61)

```
MvarZeroTJunctionStruct *MvarZeroTJNew(const MvarPtStruct *TJPrev,
                                       const MvarPtStruct *TJPt,
                                       const MvarPtStruct *TJNext,
                                       const MvarPtStruct *OrigSplit)
```

TJPrev: The point before the junction.

TJPt: The T-Junction point: A solution point.

TJNext: The point after the junction.

OrigSplit: The original curve split location, this is the TJ point, before it has been refined to the solution.

Returns: the T-Junction.

Description: Allocates the memory required for a new multi-variate T-Junction.
See also: MvarZeroTJFree, MvarZeroTJCopy,

8.2.695 MvarZeroUpdateProblemDmnExpTr (zrsolver.c:2168)

```
void MvarZeroUpdateProblemDmnExpTr(MvarZeroPrblmStruct *Problem)
```

Problem: The zero finding problem structure.

Returns: void

Description: Updates the problem with the domain and related data in the ETs case.

8.2.696 MvarZeroUpdateProblemDmnMVs (zrsolver.c:2113)

```
void MvarZeroUpdateProblemDmnMVs(MvarZeroPrblmStruct *Problem)
```

Problem: The zero finding problem structure.

Returns: void

Description: Updates the problem with the domain and related data in the MVs case.

8.2.697 MvarZerosSubdivTolAction (zrmv0d.c:961)

```
MvarZeroSubdivTolActionType MvarZerosSubdivTolAction(
                                MvarZeroSubdivTolActionType SubdivTolAction)
```

SubdivTolAction: New setting for the action type.

Returns: Old setting for the action type.

Description: Sets the action taken by the 0D solver when reaching a sub-domain of size less than subdivision tolerance.
See also: MvarZeroSolverInseparableProblem, MvarZeroSolverSinglrSol0DMVs,

8.2.698 MvarZrAlgAssignExpr (mvarzral.c:191)

```
int MvarZrAlgAssignExpr(void *MVZrAlg, const char *Name, const char *Expr)
```

MVZrAlg: Structure to add a new assignmen to.

Name: Of new assignment.

Expr: The expression.

Returns: TRUE if successful, FALSE otherwise.

Description: Insert an expression assignment into the structure.
See also:

8.2.699 MvarZrAlgAssignMVVar (mvarzral.c:291)

```
int MvarZrAlgAssignMVVar(void *MVZrAlg,
                         const char *Name,
                         CagdRType DmnMin,
                         CagdRType DmnMax,
                         const MvarMVStruct *MV)
```

MVZrAlg: Structure to add a new assignment to.

Name: Name of this variable.

DmnMin, DmnMax: Domain of this variable.

MV: ZrAlg Multivariate representing this variable. Optional.

Returns: TRUE if successful, FALSE otherwise.

Description: Insert an MV variable assignment into the structure.
See also:

8.2.700 MvarZrAlgAssignNumVar (mvarzral.c:241)

```
int MvarZrAlgAssignNumVar(void *MVZrAlg, const char *Name, CagdRType Val)
```

MVZrAlg: Structure to add a new assignmen to.

Name: Of new assignment.

Val: The numeric value.

Returns: TRUE if successful, FALSE otherwise.

Description: Insert a numeric variable into the structure.
See also:

8.2.701 MvarZrAlgCreate (mvarzral.c:90)

```
void *MvarZrAlgCreate()
```

Returns: The allocated algebraic expressions structure.

Description: Creates and allocate the structure to manipulate algebraic expressions.
See also:

8.2.702 MvarZrAlgDelete (mvarzral.c:127)

```
void MvarZrAlgDelete(void *MVZrAlg)
```

MVZrAlg: Structure to deallocate.

Returns: void

Description: Deallocates the structure to manipulate algebraic expressions.
See also:

8.2.703 MvarZrAlgGenMVCode (mvarzral.c:434)

```
int MvarZrAlgGenMVCode(void *MVZrAlg, const char *Expr, FILE *f)
```

MVZrAlg: Structure of variables and expressions.

Expr: To parse into a multivariate C Code.

f: Destination of synthesized C code.

Returns: TRUE if successful, FALSE otherwise.

Description: Generate multivariate C Code sequence into designated file that represent the given expression Expr:

1. Using MVZrAlg, perform all possible substitutions.
2. Parse the result into a binary tree and then synthesize the C code to build a multivar with variables at the leaves fetched from MVZrAlg.

See also:

8.2.704 _MvarIncBoundMeshIndices (mvar_aux.c:1054)

```
int _MvarIncBoundMeshIndices(const MvarMVStruct *MV,
                             int *Indices,
                             int *LowerBound,
                             int *UpperBound,
                             int *Index)
```

MV: To increment Indices to its control mesh.

Indices: To increment one step.

LowerBound: Minimal values to assume.

UpperBound: One above the maximal values to assume.

Index: Index to increment.

Returns: TRUE if Indices are in domain, FALSE if done.

Description: Increment the index of the control mesh of the multivariate function by one, with given lower and upper bounds: LowerBound <= Idx < UpperBound. Should only be called via the macro MVAR_INC_BOUND_MESH_INDICES.
See also: _MvarIncSkipMeshIndices, _MvarIncSkipMeshIndices1st, , _MvarIncrementMeshIndices, _MvarIncrementMeshOrderI

8.2.705 _MvarIncSkipMeshIndices (mvar_aux.c:980)

```
int _MvarIncSkipMeshIndices(const MvarMVStruct *MV,
                            int *Indices,
                            int Dir,
                            int *Index)
```

MV: To increment Indices to its control mesh.

Indices: To increment one step.

Dir: To skip in the incrementation.

Index: The total current index to be incremented as well, or zero if we wrapped around all incides.

Returns: Current non negative Index if Indices are in domain, zero (FALSE) if done - out of the domain.

Description: Increment the index of the control mesh of the multivariate function by one, skipping axis Dir. Should only be called via the macro MVAR_INC_SKIP_MESH_INDICES.
See also: _MvarIncBoundMeshIndices, _MvarIncSkipMeshIndices1st, , _MvarIncrementMeshIndices, _MvarIncrementMeshOrde

8.2.706 _MvarIncSkipMeshIndices1st (mvar_aux.c:937)

```
int _MvarIncSkipMeshIndices1st(const MvarMVStruct *MV, int *Indices)
```

MV: To increment Indices to its control mesh.

Indices: To increment one step.

Returns: TRUE if Indices are in domain, FALSE if done.

Description: Increment the index of the control mesh of the multivariate function by one, skipping axis Dir zero. Should only be called via the macro MVAR_INC_SKIP_MESH_INDICES_1ST.
See also: _MvarIncSkipMeshIndices, _MvarIncBoundMeshIndices, , _MvarIncrementMeshIndices, _MvarIncrementMeshOrderIn

8.2.707 _MvarIncrementMeshIndices (mvar_aux.c:854)

```
int _MvarIncrementMeshIndices(const MvarMVStruct *MV, int *Indices, int *Index)
```

MV: To increment Indices to its control mesh.

Indices: To increment one step.

Index: The total current index to be incremented as well, or zero if we wrapped around all indices.

Returns: The non zero advanced index if indices are in domain, zero if done (out of domain).

Description: Increment the index of the control mesh of the multivariate function by one. Should only be called via the macro MVAR_INCREMENT_MESH_INDICES.
See also: _MvarIncSkipMeshIndices, _MvarIncBoundMeshIndices, , _MvarIncSkipMeshIndices1st, _MvarIncrementMeshOrderIn

8.2.708 _MvarIncrementMeshOrderIndices (mvar_aux.c:897)

```
int _MvarIncrementMeshOrderIndices(const MvarMVStruct *MV,
                                   int *Indices,
                                   int *Index)
```

MV: To increment Indices to its control mesh.

Indices: To increment one step.

Index: The total current index to be incremented as well, or zero if we wrapped around all incides.

Returns: The non zero advanced index if indices are in domain, zero if done (out of domain).

Description: Increment the index of the control mesh of the multivariate function by one. Should only be called via the macro MVAR_INCREMENT_MESH_ORDER_INDICES. This macro is useful when traversing a Bspline mesh for evaluation, Orders[i] control points in i'th dimension.
See also: _MvarIncSkipMeshIndices, _MvarIncBoundMeshIndices, , _MvarIncSkipMeshIndices1st,

Chapter 9

Prsr Library, prsr_lib

9.1 General Information

This library provides the data file interface for IRIT. Functions are provided to read and write data files, both compressed (on unix only, using *compress*), and uncompressed, in binary and/or ascii text modes. This library is also used to exchange data between the IRIT server and the display devices' clients. Several header files can be found for this library:

Header (include/*.h)	Functionality
allocate.h	High level dynamic allocation of objects
attribut.h	High level attributes for objects
ip_cnvrt.h	Freeform to polygon and polyline high level conversion
iritprsr.h	Main interface to reading and writing data
irit_soc.h	Socket communication for data exchange

9.2 Library Functions

9.2.1 Attr2IritObject (attribut.c:1517)

attributes

```
IPObjectStruct *Attr2IritObject(const IPAttributeStruct *Attr)
```

Attr: To convert to an IRIT object.

Returns: Created irit object.

Description: Routine to convert an attribute to an IRIT object.

9.2.2 AttrCopyOneAttribute (attribut.c:1586)

attributes

```
IPAttributeStruct *AttrCopyOneAttribute(const IPAttributeStruct *Src)
```

Src: Attribute to duplicate.

Returns: Duplicated attribute.

Description: Routine to copy one attribute. This routine also exists in miscatt3.c without object handling. The routine in miscatt3.c will be linked in iff no object handling is used (i.e. this file is not linked in).

See also: AttrCopyAttributes2, AttrCopyOneAttribute2,

9.2.3 AttrCopyOneAttribute2 (attribut.c:1613)

attributes

```
IPAttributeStruct *AttrCopyOneAttribute2(const IPAttributeStruct *Src,
                                         int AllAttr)
```

Src: Attribute to duplicate.

AllAttr: If FALSE, attributes prefixed with '_' are not copied. If TRUE, all attributes are copied.

Returns: Duplicated attribute.

Description: Routine to copy one attribute. This routine also exists in miscatt3.c without object handling. The routine in miscatt3.c will be linked in iff no object handling is used (i.e. this file is not linked in).
See also: AttrCopyAttributes2, AttrCopyOneAttribute,

9.2.4 AttrDbg (iritprs2.c:792)

attributes

debugging.

```
void AttrDbg(const IPAttributeStruct *Attr)
```

Attr: Attributes to put out.

Returns: void

Description: Routine to dump to stderr the attributes of the attribute list, for debug.
See also: IPPutAttributes,

9.2.5 AttrFindObjsWithAttr (attribut.c:1958)

```
IPObjectStruct const * const *AttrFindObjsWithAttr(
                                        const IPObjectStruct *PObjs,
                                        const char *AttrName,
                                        const IPObjectStruct *AttrVal,
                                        int LeavesOnly,
                                        int Negate)
```

PObjs: A hierarchy of objects to scan and search for matched attributes. PObjs is modified in no way.

AttrName: Attribute name to search in the object of the hierarchy.

AttrVal: Optional (can be NULL) attribute value to verify for similarity as well.

LeavesOnly: TRUE to search only leaves, FALSE to also consider sub-lists.

Negate: If TRUE, returns all objects that do not posses the attributes (or the attribute value).

Returns: A Vector holding references on the original objects in PObjs that possess the sought attribute name (and possibly attribute value), or NULL if none found. Allocated dynamically.

Description: Searches the given hierarchy of objects, PObjs, for object that contains an attribute named AttrName. If AttrVal is not NULL, an additional test is made to verify the desired attribute value. Only support attribute value comparisons of numeric and string values.

9.2.6 AttrFreeObjectAttribute (attribut.c:1395)

```
void AttrFreeObjectAttribute(const IPObjectStruct *PObj, const char *Name)
```

PObj: Object to free attributes from.

Name: Name of attribute to delete, or all attributes if NULL.

Returns: void

Description: Free one or all attributes of an object PObj and its descendants.
See also: AttrFreeOneAttribute, AttrFreeAttributes,

9.2.7 AttrGetMAttribCount (iritvrml.c:377)

attributes

```
int AttrGetMAttribCount(IPAttributeStruct *Attr)
```

Attr: Attribute list to search for requested attribute.

Returns: Count of the values actually present in attribute.

Description: Routine to return a count of values in multi-attribute.
See also: AttrGetMRealAttrib, AttrGetMAttribCount,

9.2.8 AttrGetMIntAttrib (iritvrml.c:416)

attributes

```
int AttrGetMIntAttrib(IPAttributeStruct *Attrs, char *Name, int N, int **PV)
```

Attrs: Attribute list to search for requested attribute.

Name: Name of requested attribute.

N: Count of values in PV array, or zero.

PV: Address of the pointer to the first array value, or pointer itself.

Returns: Count of the values actually present in MAttribute.

Description: Routine to return a multi-integer attribute. Note, that when N is zero, PV is an address of a pointer that will receive allocated array, otherwise PV is regarded as just a pointer to the first value in the client provided array. If N is greater then actual count of values in the attribute, last value is replicated.
See also: AttrIDSetIntAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, , AttrGetRealAttrib, AttrIDSetStrAttrib, AttrGetStrAttrib), , AttrSetRealAttrib, AttrGetMRealAttrib, AttrGetMAttribCount,

9.2.9 AttrGetMRealAttrib (iritvrml.c:522)

attributes

```
int AttrGetMRealAttrib(IPAttributeStruct *Attrs,
                       IPAttrIDType AttrID,
                       int N,
                       IrtRType **PV)
```

Attrs: Attribute list to search for requested attribute.

AttrID: ID of requested attribute.

N: Count of values in PV array, or zero.

PV: Address of the pointer to the first array value, or pointer itself.

Returns: Count of the values actually present in MAttribute.

Description: Routine to return a multi-real attribute. Note, that when N is zero, PV is an address of a pointer that will recieve allocated array, otherwise PV is regarded as just a pointer to the first value in the client provided array. If N is greater then actual count of values in the attribute, last value is replicated.
See also: AttrIDSetIntAttrib, AttrSetPtrAttrib, AttrGetPtrAttrib, , AttrGetRealAttrib, AttrIDSetStrAttrib, AttrGetStrAttrib, , AttrSetRealAttrib, AttrGetMIntAttrib, AttrGetMAttribCount,

9.2.10 AttrGetObjAttrib (attribut.c:1333)

attributes

```
IPObjectStruct *AttrGetObjAttrib(const IPAttributeStruct *Attrs,
                                 const char *Name)
```

Attrs: Attribute list to search for requested attribute.

Name: Name of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get an object attribute.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrSetObjectStrAttrib, , AttrGetObjectStrAttrib, AttrSetObjectObjAttrib, AttrSetObjAttrib, , AttrGetObjectObjAttrib, AttrGetObjAttrib2,

9.2.11 AttrGetObjAttrib2 (attribut.c:1367)

attributes

```
IPObjectStruct *AttrGetObjAttrib2(const IPAttributeStruct *Attrs,
                                  IPAttrNumType AttribNum)
```

Attrs: Attribute list to search for requested attribute.

AttribNum: Unique ID derived from name of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get an object attribute.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrSetObjectStrAttrib, , AttrGetObjectStrAttrib, AttrSetObjectObjAttrib, AttrSetObjAttrib, , AttrGetObjectObjAttrib, AttrGetObjAttrib,

9.2.12 AttrGetObjectColor (attribut.c:90)

attributes

color

```
int AttrGetObjectColor(const IPObjectStruct *PObj)
```

PObj: For which we would like to know the color of.

Returns: Color of PObj or IP_ATTR_NO_COLOR if no color set.

Description: Routine to return the color of an object.
See also: AttrSetObjectColor, AttrSetObjectRGBColor, AttrGetObjectRGBColor, , AttrGetObjectWidth, AttrSetObjectWidth, AttrSetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, ,

9.2.13 AttrGetObjectGeomAttr (attribut.c:1472)

attributes

color

```
IPAttributeStruct *AttrGetObjectGeomAttr(const IPObjectStruct *PObj)
```

PObj: Object to get its geometric entity attribute slot.

Returns: The attribute slot, or NULL if not found/failed.

Description: Returns the attribute slot of the geometry entity in PObj, if any.
See also: AttrGetObjectColor, AttrSetObjectRGBColor, AttrGetObjectRGBColor, , AttrGetObjectWidth, AttrSetObjectWidth, AttrSetObjectIntAttrib, ,

9.2.14 AttrGetObjectIntAttrib (attribut.c:317)

attributes

```
int AttrGetObjectIntAttrib(const IPObjectStruct *PObj, const char *Name)
```

PObj: Object from which to get an ingeter attribute.

Name: Name of integer attribute.

Returns: Found attribute, or IP_ATTR_BAD_INT if not found.

Description: Routine to get an integer attribute from an object.
See also: AttrSetObjectIntAttrib, AttrSetObjectPtrAttrib, AttrGetObjectPtrAttrib, , AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrSetObjectStrAttrib, , AttrGetObjectStrAttrib, AttrSetObjectObjAttrib, AttrSetObjAttrib, AttrGetObjectObjAttrib, AttrGetObjAttrib, AttrGetObjectIntAttrib2,

9.2.15 AttrGetObjectIntAttrib2 (attribut.c:345)

attributes

```
int AttrGetObjectIntAttrib2(const IPObjectStruct *PObj,
                            IPAttrNumType AttribNum)
```

PObj: Object from which to get an ingeter attribute.

AttribNum: Unique ID derived from name of requested attribute.

Returns: Found attribute, or IP_ATTR_BAD_INT if not found.

Description: Routine to get an integer attribute from an object.
See also: AttrSetObjectIntAttrib, AttrSetObjectPtrAttrib, AttrGetObjectPtrAttrib, , AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrSetObjectStrAttrib, , AttrGetObjectStrAttrib, AttrSetObjectObjAttrib, AttrSetObjAttrib, AttrGetObjectObjAttrib, AttrGetObjAttrib, AttrGetObjectIntAttrib,

9.2.16 AttrGetObjectObjAttrib (attribut.c:1277)

attributes

```
IPObjectStruct *AttrGetObjectObjAttrib(const IPObjectStruct *PObj,
                                       const char *Name)
```

PObj: Object from which to get a object attribute.

Name: Name of object attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get an object attribute from an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrSetObjectStrAttrib, , AttrGetObjectStrAttrib, AttrSetObjectObjAttrib, AttrSetObjAttrib, AttrGetObjAttrib, AttrGetObjectObjAttrib2,

9.2.17 AttrGetObjectObjAttrib2 (attribut.c:1305)

attributes

```
IPObjectStruct *AttrGetObjectObjAttrib2(const IPObjectStruct *PObj,
                                        IPAttrNumType AttribNum)
```

PObj: Object from which to get a object attribute.

AttribNum: Unique ID derived from name of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get an object attribute from an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrSetObjectStrAttrib, , AttrGetObjectStrAttrib, AttrSetObjectObjAttrib, AttrSetObjAttrib, AttrGetObjAttrib, AttrGetObjectObjAttrib,

9.2.18 AttrGetObjectPtrAttrib (attribut.c:441)

attributes

```
VoidPtr AttrGetObjectPtrAttrib(const IPObjectStruct *PObj, const char *Name)
```

PObj: Object from which to get a pointer attribute.

Name: Name of pointer attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a pointer attribute from an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectStrAttrib, AttrGetObjectStrAttrib, AttrSetObjectObjAttrib, , AttrSetObjAttrib, AttrGetObjectObjAttrib, AttrGetObjAttrib, AttrSetObjectRefPtrAttrib, AttrGetObjectRefPtrAttrib, AttrGetObjectPtrAttrib2,

9.2.19 AttrGetObjectPtrAttrib2 (attribut.c:470)

attributes

```
VoidPtr AttrGetObjectPtrAttrib2(const IPObjectStruct *PObj,
                                IPAttrNumType AttribNum)
```

PObj: Object from which to get a pointer attribute.

AttribNum: Unique ID derived from name of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a pointer attribute from an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectStrAttrib, AttrGetObjectStrAttrib, AttrSetObjectObjAttrib, , AttrSetObjAttrib, AttrGetObjectObjAttrib, AttrGetObjAttrib, AttrSetObjectRefPtrAttrib, AttrGetObjectRefPtrAttrib, AttrGetObjectPtrAttrib,

9.2.20 AttrGetObjectRGBColor (attribut.c:147)

attributes
color
rgb

```
int AttrGetObjectRGBColor(const IPObjectStruct *PObj,
                          int *Red,
                          int *Green,
                          int *Blue)
```

PObj: Object to get its RGB color.

Red, Green, Blue: Component of color to initialize.

Returns: TRUE if PObj does have an RGB color attribute, FALSE otherwise.

Description: Routine to return the RGB color of an object.
See also: AttrSetObjectColor, AttrGetObjectColor, AttrSetObjectRGBColor, , AttrGetObjectWidth, AttrSetObjectWidth, AttrSetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRGBColor2,

9.2.21 AttrGetObjectRGBColor2 (attribut.c:176)

attributes
color
rgb

```
int AttrGetObjectRGBColor2(const IPObjectStruct *PObj,
                           const char *Name,
                           int *Red,
                           int *Green,
                           int *Blue)
```

PObj: Object to get its RGB color.

Name: Name of the attribute, if NULL default is taken.

Red, Green, Blue: Component of color to initialize.

Returns: TRUE if PObj does have an RGB color attribute, FALSE otherwise.

Description: Routine to return the RGB color of an object.
See also: AttrSetObjectColor, AttrGetObjectColor, AttrSetObjectRGBColor, , AttrGetObjectWidth, AttrSetObjectWidth, AttrSetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRGBColor,

9.2.22 AttrGetObjectRealAttrib (attribut.c:687)

attributes

```
IrtRType AttrGetObjectRealAttrib(const IPObjectStruct *PObj, const char *Name)
```

PObj: Object from which to get a real attribute.

Name: Name of real attribute.

Returns: Found attribute, or IP_ATTR_BAD_REAL if not found.

Description: Routine to get a real attribute from an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrSetObjectRealAttrib, AttrSetObjectStrAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrGetObjectStrAttrib, , AttrSetObjectObjAttrib, AttrSetObjAttrib, AttrGetObjectObjAttrib, AttrGetObjAttrib, AttrGetObjectRealAttrib2,

9.2.23 AttrGetObjectRealAttrib2 (attribut.c:715)

attributes

```
IrtRType AttrGetObjectRealAttrib2(const IPObjectStruct *PObj,
                                  IPAttrNumType AttribNum)
```

PObj: Object from which to get a real attribute.

AttribNum: Unique ID derived from name of requested attribute.

Returns: Found attribute, or IP_ATTR_BAD_REAL if not found.

Description: Routine to get a real attribute from an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrSetObjectRealAttrib, AttrSetObjectStrAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrGetObjectStrAttrib, , AttrSetObjectObjAttrib, AttrSetObjAttrib, AttrGetObjectObjAttrib, AttrGetObjAttrib, AttrGetObjectRealAttrib,

9.2.24 AttrGetObjectRealPtrAttrib (attribut.c:816)

attributes

```
IrtRType *AttrGetObjectRealPtrAttrib(const IPObjectStruct *PObj,
                                     const char *Name)
```

PObj: Object from which to get a real attribute.

Name: Name of real attribute.

Returns: Found attribute, or IP_ATTR_BAD_REAL if not found.

Description: Routine to get a real attribute from an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrSetObjectRealAttrib, AttrSetObjectStrAttrib, , AttrSetObjectRealPtrAttrib2, AttrGetObjectRealPtrAttrib2, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrGetObjectStrAttrib, , AttrSetObjectObjAttrib, AttrSetObjAttrib, AttrGetObjectObjAttrib, AttrGetObjAttrib, AttrGetObjectRealAttrib2,

9.2.25 AttrGetObjectRealPtrAttrib2 (attribut.c:844)

attributes

```
IrtRType *AttrGetObjectRealPtrAttrib2(const IPObjectStruct *PObj,
                                      IPAttrNumType AttribNum)
```

PObj: Object from which to get a real attribute.

AttribNum: Unique ID derived from name of requested attribute.

Returns: Found attribute, or IP_ATTR_BAD_REAL if not found.

Description: Routine to get a real attribute from an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrSetObjectRealAttrib, AttrSetObjectStrAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrGetObjectStrAttrib, , AttrSetObjectObjAttrib, AttrSetObjAttrib, AttrGetObjectObjAttrib, AttrGetObjAttrib, AttrGetObjectRealAttrib,

9.2.26 AttrGetObjectRefPtrAttrib (attribut.c:567)

attributes

```
VoidPtr AttrGetObjectRefPtrAttrib(const IPObjectStruct *PObj, const char *Name)
```

PObj: Object from which to get a pointer reference attribute.

Name: Name of pointer attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a pointer reference attribute from an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectStrAttrib, AttrGetObjectStrAttrib, AttrSetObjectObjAttrib, , AttrSetObjAttrib, AttrGetObjectObjAttrib, AttrGetObjAttrib, AttrSetObjectRefPtrAttrib, AttrGetObjectRefPtrAttrib, AttrGetObjectRefPtrAttrib2,

9.2.27 AttrGetObjectRefPtrAttrib2 (attribut.c:596)

attributes

```
VoidPtr AttrGetObjectRefPtrAttrib2(const IPObjectStruct *PObj,
                                   IPAttrNumType AttribNum)
```

PObj: Object from which to get a pointer reference attribute.

AttribNum: Unique ID derived from name of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a pointer reference attribute from an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectStrAttrib, AttrGetObjectStrAttrib, AttrSetObjectObjAttrib, , AttrSetObjAttrib, AttrGetObjectObjAttrib, AttrGetObjAttrib, AttrSetObjectRefPtrAttrib, AttrGetObjectRefPtrAttrib, AttrGetObjectRefPtrAttrib,

9.2.28 AttrGetObjectStrAttrib (attribut.c:1057)

attributes

```
const char *AttrGetObjectStrAttrib(const IPObjectStruct *PObj,
                                   const char *Name)
```

PObj: Object from which to get a string attribute.

Name: Name of string attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a string attribute from an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrSetObjectStrAttrib, , AttrSetObjectObjAttrib, AttrSetObjAttrib, AttrGetObjectObjAttrib, , AttrGetObjAttrib, AttrGetObjectStrAttrib2,

9.2.29 AttrGetObjectStrAttrib2 (attribut.c:1085)

attributes

```
const char *AttrGetObjectStrAttrib2(const IPObjectStruct *PObj,
                                    IPAttrNumType AttribNum)
```

PObj: Object from which to get a string attribute.

AttribNum: Unique ID derived from name of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a string attribute from an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrSetObjectStrAttrib, , AttrSetObjectObjAttrib, AttrSetObjAttrib, AttrGetObjectObjAttrib, , AttrGetObjAttrib, AttrGetObjectStrAttrib,

9.2.30 AttrGetObjectUVAttrib (attribut.c:937)

attributes

```
float *AttrGetObjectUVAttrib(const IPObjectStruct *PObj, const char *Name)
```

PObj: Object from which to get a real attribute.

Name: Name of real attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a UV attribute from an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrSetObjectRealAttrib, AttrSetObjectStrAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrGetObjectStrAttrib, AttrSetObjectObjAttrib, AttrSetObjAttrib, , AttrGetObjectObjAttrib, AttrGetObjAttrib, AttrSetObjectUVAttrib, , AttrGetObjectUVAttrib2,

9.2.31 AttrGetObjectUVAttrib2 (attribut.c:965)

attributes

```
float *AttrGetObjectUVAttrib2(const IPObjectStruct *PObj,
                              IPAttrNumType AttribNum)
```

PObj: Object from which to get a real attribute.

AttribNum: Unique ID derived from name of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a UV attribute from an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrSetObjectRealAttrib, AttrSetObjectStrAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrGetObjectStrAttrib, AttrSetObjectObjAttrib, AttrSetObjAttrib, , AttrGetObjectObjAttrib, AttrGetObjAttrib, AttrSetObjectUVAttrib, , AttrGetObjectUVAttrib,

9.2.32 AttrGetObjectWidth (attribut.c:226)

attributes

```
IrtRType AttrGetObjectWidth(const IPObjectStruct *PObj)
```

width

PObj: For which we would like to know the width of.

Returns: Width of PObj or IP_ATTR_NO_WIDTH if no width set.

Description: Routine to return the width of an object.
See also: AttrSetObjectColor, AttrGetObjectColor, AttrSetObjectRGBColor, , AttrGetObjectRGBColor, AttrSetObjectWidth, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, ,

9.2.33 AttrIDFreeObjectAttribute (attribute_id.c:828)

```
void AttrIDFreeObjectAttribute(const IPObjectStruct *PObj, IPAttrIDType ID)
```

PObj: Object to free attributes from. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

ID: ID of attribute to delete.

Returns: void

Description: Free one attributes of an object PObj and its descendants, by ID.
See also: AttrFreeOneAttribute, AttrFreeAttributes,

9.2.34 AttrIDGetIndexColor (attribute_id.c:60)

attributes

```
void AttrIDGetIndexColor(int Color, int *Red, int *Green, int *Blue)
```

color

rgb

Color: Index of color between 0 and 15.

Red, Green, Blue: Component of RGB color.

Returns: void

Description: Routine to fetch one of 16 basic colors based on its index.
See also: AttrIDSetColor, AttrIDGetColor, AttrSetIntAttrib, AttrSetObjectRGBColor,

9.2.35 AttrIDGetObjAttrib (attribute_id.c:798)

attributes

```
IPObjectStruct *AttrIDGetObjAttrib(const IPAttributeStruct *Attrs,
                                   IPAttrIDType ID)
```

Attrs: Attribute list to search for requested attribute.

ID: ID of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get an object attribute.
See also: AttrIDSetObjectIntAttrib, AttrIDGetObjectIntAttrib, , AttrIDGetObjectPtrAttrib, AttrIDSetObjectRealAttrib, , AttrIDSetObjectRealPtrAttrib, AttrIDGetObjectRealPtrAttrib, , AttrIDSetObjectUVAttrib, AttrIDGetObjectUVAttrib, , AttrIDGetObjectStrAttrib, AttrIDSetObjectObjAttrib, AttrIDSetObjAttrib, , AttrIDGetObjectObjAttrib, AttrIDSetObjectPtrAttrib, , AttrIDGetObjectRealAttrib, AttrIDSetObjectStrAttrib,

9.2.36 AttrIDGetObjectColor (attribute_id.c:87)

attributes

color

```
int AttrIDGetObjectColor(const IPObjectStruct *PObj)
```

PObj: For which we would like to know the color of.

Returns: Color of PObj or IP_ATTR_NO_COLOR if no color set.

Description: Routine to return the color of an object.
See also: AttrIDSetObjectColor, AttrIDSetObjectRGBColor, AttrIDGetObjectRGBColor, , AttrIDGetObjectWidth, AttrIDSetObjectWidth, AttrIDSetObjectRealAttrib, , AttrIDSetObjectRealPtrAttrib, AttrIDGetColor,

9.2.37 AttrIDGetObjectIntAttrib (attribute_id.c:283)

attributes

```
int AttrIDGetObjectIntAttrib(const IPObjectStruct *PObj, IPAttrIDType ID)
```

PObj: Object from which to get an ingeter attribute.

ID: ID of integer attribute.

Returns: Found attribute, or IP_ATTR_BAD_INT if not found.

Description: Routine to get an integer attribute from an object.
See also: AttrIDSetObjectIntAttrib, AttrIDSetObjectPtrAttrib, , AttrIDSetObjectRealAttrib, AttrIDGetObjectRealAttrib, , AttrIDSetObjectRealPtrAttrib, AttrIDGetObjectRealPtrAttrib, , AttrIDSetObjectUVAttrib, AttrIDGetObjectUVAttrib, , AttrIDGetObjectStrAttrib, AttrIDSetObjectObjAttrib, AttrIDSetObjAttrib, AttrIDGetObjectObjAttrib, AttrIDGetObjAttrib, AttrIDGetObjectPtrAttrib, , AttrIDSetObjectStrAttrib,

9.2.38 AttrIDGetObjectObjAttrib (attribute_id.c:769)

attributes

```
IPObjectStruct *AttrIDGetObjectObjAttrib(const IPObjectStruct *PObj,
                                         IPAttrIDType ID)
```

PObj: Object from which to get a object attribute.

ID: ID of object attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get an object attribute from an object.
See also: AttrIDSetObjectIntAttrib, AttrIDGetObjectIntAttrib, , AttrIDGetObjectPtrAttrib, AttrIDSetObjectRealAttrib, , AttrIDSetObjectRealPtrAttrib, AttrIDGetObjectRealPtrAttrib, , AttrIDSetObjectUVAttrib, AttrIDGetObjectUVAttrib, , AttrIDGetObjectStrAttrib, AttrIDSetObjectObjAttrib, , AttrIDSetObjectPtrAttrib, AttrIDGetObjectRealAttrib, , AttrIDSetObjectStrAttrib, AttrIDSetObjAttrib, AttrIDGetObjAttrib,

9.2.39 AttrIDGetObjectPtrAttrib (attribute_id.c:345)

attributes

```
VoidPtr AttrIDGetObjectPtrAttrib(const IPObjectStruct *PObj, IPAttrIDType ID)
```

PObj: Object from which to get a pointer attribute.

ID: ID of pointer attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a pointer attribute from an object.
See also: AttrIDSetObjectIntAttrib, AttrIDGetObjectIntAttrib, , AttrIDSetObjectRealAttrib, AttrIDGetObjectRealAttrib, , AttrIDSetObjectRealPtrAttrib, AttrIDGetObjectRealPtrAttrib, , AttrIDSetObjectStrAttrib, AttrIDGetObjectStrAttrib, , AttrIDSetObjAttrib, AttrIDGetObjectObjAttrib, AttrIDGetObjAttrib, , AttrIDSetObjectRefPtrAttrib, AttrIDGetObjectRefPtrAttrib, , AttrIDSetObjectPtrAttrib, AttrIDSetObjectObjAttrib,

9.2.40 AttrIDGetObjectRGBColor (attribute_id.c:144)

attributes

color

rgb

```
int AttrIDGetObjectRGBColor(const IPObjectStruct *PObj,
                            int *Red,
                            int *Green,
                            int *Blue)
```

PObj: Object to get its RGB color.

Red, Green, Blue: Component of color to initialize.

Returns: TRUE if PObj does have an RGB color attribute, FALSE otherwise.

Description: Routine to return the RGB color of an object. Examines "rgb" attributes.
See also: AttrIDSetObjectColor, AttrIDGetObjectColor, AttrIDSetObjectRGBColor, , AttrIDGetObjectWidth, AttrIDSetObjectWidth, AttrIDSetObjectRealAttrib, , AttrIDSetObjectRealPtrAttrib, AttrIDGetRGBColor,

9.2.41 AttrIDGetObjectRGBColor2 (attribute_id.c:172)

attributes

color

rgb

```
int AttrIDGetObjectRGBColor2(const IPObjectStruct *PObj,
                             int *Red,
                             int *Green,
                             int *Blue)
```

PObj: Object to get its RGB color.

Red, Green, Blue: Component of color to initialize.

Returns: TRUE if PObj does have an RGB color attribute, FALSE otherwise.

Description: Routine to return the RGB color of an object. Examines "rgb" and "color" attributes.
See also: AttrIDSetObjectColor, AttrIDGetObjectColor, AttrIDSetObjectRGBColor, , AttrIDGetObjectWidth, AttrIDSetObjectWidth, AttrIDSetObjectRealAttrib, , AttrIDSetObjectRealPtrAttrib, AttrIDGetRGBColor,

9.2.42 AttrIDGetObjectRealAttrib (attribute_id.c:470)

attributes

```
IrtRType AttrIDGetObjectRealAttrib(const IPObjectStruct *PObj,
                                   IPAttrIDType ID)
```

PObj: Object from which to get a real attribute.

ID: ID of real attribute.

Returns: Found attribute, or IP_ATTR_BAD_REAL if not found.

Description: Routine to get a real attribute from an object.
See also: AttrIDSetObjectIntAttrib, AttrIDGetObjectIntAttrib, , AttrIDGetObjectPtrAttrib, AttrIDSetObjectRealAttrib, , AttrIDSetObjectRealPtrAttrib, AttrIDGetObjectRealPtrAttrib, , AttrIDSetObjectUVAttrib, AttrIDGetObjectUVAttrib, , AttrIDSetObjectObjAttrib, AttrIDSetObjAttrib, AttrIDGetObjectObjAttrib, , AttrIDGetObjAttrib, AttrIDSetObjectPtrAttrib, AttrIDSetObjectStrAttrib, , AttrIDGetObjectStrAttrib,

9.2.43 AttrIDGetObjectRealPtrAttrib (attribute_id.c:534)

attributes

```
IrtRType *AttrIDGetObjectRealPtrAttrib(const IPObjectStruct *PObj,
                                       IPAttrIDType ID)
```

PObj: Object from which to get a real attribute.

ID: ID of real attribute.

Returns: Found attribute, or IP_ATTR_BAD_REAL if not found.

Description: Routine to get a real attribute from an object.
See also: AttrIDSetObjectIntAttrib, AttrIDGetObjectIntAttrib, , AttrIDGetObjectPtrAttrib, AttrIDSetObjectRealAttrib, , AttrIDSetObjectUVAttrib, AttrIDGetObjectUVAttrib, , AttrIDSetObjectObjAttrib, AttrIDSetObjAttrib, AttrIDGetObjectObjAttrib, AttrIDSetObjectPtrAttrib, AttrIDSetObjectStrAttrib, , AttrIDGetObjectStrAttrib,

9.2.44 AttrIDGetObjectRefPtrAttrib (attribute_id.c:408)

attributes

```
VoidPtr AttrIDGetObjectRefPtrAttrib(const IPObjectStruct *PObj,
                                    IPAttrIDType ID)
```

PObj: Object from which to get a pointer reference attribute.

ID: ID of pointer attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a pointer reference attribute from an object.
See also: AttrIDSetObjectIntAttrib, AttrIDGetObjectIntAttrib, , AttrIDSetObjectRealAttrib, AttrIDGetObjectRealAttrib, , AttrIDSetObjectRealPtrAttrib, AttrIDGetObjectRealPtrAttrib, , AttrIDSetObjectStrAttrib, AttrIDGetObjectStrAttrib, , AttrIDSetObjAttrib, AttrIDGetObjectObjAttrib, AttrIDGetObjAttrib, AttrIDSetObjectRefPtrAttrib, AttrIDGetObjectRefPtrAttrib, , AttrIDSetObjectPtrAttrib, AttrIDSetObjectObjAttrib,

9.2.45 AttrIDGetObjectStrAttrib (attribute_id.c:656)

attributes

```
const char *AttrIDGetObjectStrAttrib(const IPObjectStruct *PObj,
                                     IPAttrIDType ID)
```

PObj: Object from which to get a string attribute.

ID: ID of attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a string attribute from an object.
See also: AttrIDSetObjectIntAttrib, AttrIDGetObjectIntAttrib, , AttrIDGetObjectPtrAttrib, AttrIDSetObjectRealAttrib, , AttrIDSetObjectRealPtrAttrib, AttrIDGetObjectRealPtrAttrib, , AttrIDSetObjectUVAttrib, AttrIDGetObjectUVAttrib, , AttrIDSetObjectObjAttrib, AttrIDSetObjAttrib, AttrIDGetObjectObjAttrib, , AttrIDSetObjectPtrAttrib, AttrIDGetObjectRealAttrib, , AttrIDSetObjectStrAttrib, AttrIDGetObjAttrib,

9.2.46 AttrIDGetObjectUVAttrib (attribute_id.c:594)

attributes

```
float *AttrIDGetObjectUVAttrib(const IPObjectStruct *PObj, IPAttrIDType ID)
```

PObj: Object from which to get a real attribute.

ID: ID of real attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a UV attribute from an object.
See also: AttrIDSetObjectIntAttrib, AttrIDGetObjectIntAttrib, , AttrIDGetObjectPtrAttrib, AttrIDSetObjectRealAttrib, , AttrIDSetObjectRealPtrAttrib, AttrIDGetObjectRealPtrAttrib, , AttrIDGetObjectStrAttrib, AttrIDSetObjectObjAttrib, AttrIDSetObjAttrib, , AttrIDGetObjectObjAttrib, AttrIDGetObjAttrib, AttrIDSetObjectUVAttrib, , AttrIDSetObjectPtrAttrib, AttrIDSetObjectStrAttrib,

9.2.47 AttrIDGetObjectWidth (attribute_id.c:223)

attributes

width

```
IrtRType AttrIDGetObjectWidth(const IPObjectStruct *PObj)
```

PObj: For which we would like to know the width of.

Returns: Width of PObj or IP_ATTR_NO_WIDTH if no width set.

Description: Routine to return the width of an object.
See also: AttrIDSetObjectColor, AttrIDGetObjectColor, AttrIDSetObjectRGBColor, , AttrIDGetObjectRGBColor, AttrIDSetObjectWidth, AttrIDGetObjectRealAttrib, , AttrIDSetObjectRealPtrAttrib, AttrIDSetWidth,

9.2.48 AttrIDPropagateAttr (attribute_id.c:859)

attributes

```
void AttrIDPropagateAttr(const IPObjectStruct *PObj, IPAttrIDType AttrID)
```

files

parser

PObj: To propagate down Attr attributes.

AttrID: ID of attribute to propagate or IRIT_ATTR_INVALID_ATTR_ID to propagate all attributes.

Returns: void

Description: Propagate attributes from list objects down into their elements. Non propagatable attributes are accumulated or ignored as follows: "animation" is accumulated. "invisible" is ignored.

9.2.49 AttrIDSetObjAttrib (attribute_id.c:725)

attributes

```
void AttrIDSetObjAttrib(IPAttributeStruct **Attrs,
                        IPAttrIDType ID,
                        IPObjectStruct *Data,
                        int CopyData)
```

Attrs: Attribute list where to place new attribute.

ID: ID of the newly introduced attribute.

Data: Pointer attribute to save.

CopyData: If TRUE, object Data is duplicated first.

Returns: void

Description: Routine to set an object attribute.
See also: AttrIDSetObjectIntAttrib, AttrIDGetObjectIntAttrib, , AttrIDGetObjectPtrAttrib, AttrIDSetObjectRealAttrib, , AttrIDSetObjectRealPtrAttrib, AttrIDGetObjectRealPtrAttrib, , AttrIDSetObjectUVAttrib, AttrIDGetObjectUVAttrib, , AttrIDGetObjectStrAttrib, AttrIDSetObjectObjAttrib, , AttrIDSetObjectPtrAttrib, AttrIDGetObjectRealAttrib, , AttrIDSetObjectStrAttrib, AttrIDGetObjectObjAttrib,

9.2.50 AttrIDSetObjectColor (attribute_id.c:38)

attributes

```
void AttrIDSetObjectColor(const IPObjectStruct *PObj, int Color)
```

color

PObj: Object to set its color to Color. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

Color: New color for PObj.

Returns: void

Description: Routine to set the color of an object.
See also: AttrIDGetObjectColor, AttrIDSetObjectRGBColor, AttrIDGetObjectRGBColor, , AttrIDGetObjectWidth, AttrIDSetObjectWidth, AttrIDSetObjectIntAttrib, , AttrIDSetColor,

9.2.51 AttrIDSetObjectIntAttrib (attribute_id.c:255)

attributes

```
void AttrIDSetObjectIntAttrib(const IPObjectStruct *PObj,
                              IPAttrIDType ID,
                              int Data)
```

PObj: To add an integer attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

ID: ID of attribute.

Data: Content of attribute.

Returns: void

Description: Routine to set an integer attribute for an object.
See also: AttrIDGetObjectIntAttrib, AttrIDSetObjectPtrAttrib, , AttrIDSetObjectRealAttrib, AttrIDGetObjectRealAttrib, , AttrIDSetObjectRealPtrAttrib, AttrIDGetObjectRealPtrAttrib, , AttrIDSetObjectUVAttrib, AttrIDGetObjectUVAttrib, , AttrIDGetObjectStrAttrib, AttrIDSetObjectObjAttrib, AttrIDSetObjAttrib, AttrIDGetObjectObjAttrib, AttrIDGetObjAttrib, AttrIDSetObjectStrAttrib,

9.2.52 AttrIDSetObjectObjAttrib (attribute_id.c:691)

attributes

```
void AttrIDSetObjectObjAttrib(const IPObjectStruct *PObj,
                              IPAttrIDType ID,
                              IPObjectStruct *Data,
                              int CopyData)
```

PObj: To add an object attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

ID: ID of attribute.

Data: Content of attribute.

CopyData: If TRUE, Data object is duplicated first.

Returns: void

Description: Routine to set an object attribute for an object.
See also: AttrIDSetObjectIntAttrib, AttrIDGetObjectIntAttrib, , AttrIDGetObjectPtrAttrib, AttrIDSetObjectRealAttrib, , AttrIDSetObjectRealPtrAttrib, AttrIDGetObjectRealPtrAttrib, , AttrIDSetObjectUVAttrib, AttrIDGetObjectUVAttrib, , AttrIDGetObjectStrAttrib, AttrIDSetObjAttrib, AttrIDGetObjectObjAttrib, AttrIDSetObjectPtrAttrib, AttrIDGetObjectRealAttrib, , AttrIDSetObjectStrAttrib, AttrIDGetObjAttrib,

9.2.53 AttrIDSetObjectPtrAttrib (attribute_id.c:317)

attributes

```
void AttrIDSetObjectPtrAttrib(const IPObjectStruct *PObj,
                              IPAttrIDType ID,
                              VoidPtr Data)
```

PObj: To add a pointer attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

ID: ID of attribute.

Data: Content of attribute.

Returns: void

Description: Routine to set a pointer attribute for an object.
See also: AttrIDSetObjectIntAttrib, AttrIDGetObjectIntAttrib, , AttrIDSetObjectRealAttrib, AttrIDGetObjectRealAttrib, , AttrIDSetObjectRealPtrAttrib, AttrIDGetObjectRealPtrAttrib, , AttrIDSetObjectUVAttrib, AttrIDGetObjectUVAttrib, , AttrIDGetObjectStrAttrib, AttrIDSetObjectObjAttrib, AttrIDSetObjAttrib, , AttrIDGetObjectObjAttrib, AttrIDGetObjAttrib, AttrIDGetObjectPtrAttrib, , AttrIDSetObjectRefPtrAttrib, AttrIDGetObjectRefPtrAttrib, , AttrIDSetObjectStrAttrib,

9.2.54 AttrIDSetObjectRGBColor (attribute_id.c:117)

attributes

color

rgb

```
void AttrIDSetObjectRGBColor(const IPObjectStruct *PObj,
                             int Red,
                             int Green,
                             int Blue)
```

PObj: Object to set its RGB color. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

Red, Green, Blue: Component of color.

Returns: void

Description: Routine to set the RGB color of an object.
See also: AttrIDSetObjectColor, AttrIDGetObjectColor, AttrIDGetObjectRGBColor, , AttrIDGetObjectWidth, AttrIDSetObjectWidth, AttrIDSetObjectRealAttrib, , AttrIDSetObjectRealPtrAttrib, AttrIDSetRGBColor,

9.2.55 AttrIDSetObjectRealAttrib (attribute_id.c:441)

attributes

```
void AttrIDSetObjectRealAttrib(const IPObjectStruct *PObj,
                               IPAttrIDType ID,
                               IrtRType Data)
```

PObj: To add a real attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

ID: ID of attribute.

Data: Content of attribute.

Returns: void

Description: Routine to set a real attribute for an object.
See also: AttrIDSetObjectIntAttrib, AttrIDGetObjectIntAttrib, , AttrIDGetObjectPtrAttrib, AttrIDGetObjectRealAttrib, , AttrIDSetObjectRealPtrAttrib, AttrIDGetObjectRealPtrAttrib, , AttrIDSetObjectUVAttrib, AttrIDGetObjectUVAttrib, , AttrIDSetObjectObjAttrib, AttrIDSetObjAttrib, AttrIDGetObjectObjAttrib, , AttrIDGetObjAttrib, AttrIDSetObjectPtrAttrib AttrIDSetObjectStrAttrib, , AttrIDGetObjectStrAttrib,

9.2.56 AttrIDSetObjectRealPtrAttrib (attribute_id.c:505)

attributes

```
void AttrIDSetObjectRealPtrAttrib(const IPObjectStruct *PObj,
                                  IPAttrIDType ID,
                                  IrtRType *Data,
                                  int DataLen)
```

PObj: To add a real * attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

ID: ID of attribute.

Data: Content of attribute.

DataLen: If positive allocates and copies that many reals from Data. Otherwise, Data (of length -DataLen) is used directly.

Returns: void

Description: Routine to set a real * attribute for an object.
See also: AttrIDSetObjectIntAttrib, AttrIDGetObjectIntAttrib, , AttrIDGetObjectPtrAttrib, AttrIDGetObjectRealAttrib, , AttrIDSetObjectUVAttrib, AttrIDGetObjectUVAttrib, , AttrIDSetObjectObjAttrib, AttrIDSetObjAttrib, AttrIDGetObjectObjAttrib, , AttrIDSetObjectPtrAttrib, AttrIDSetObjectStrAttrib, , AttrIDGetObjectStrAttrib, AttrIDGetObjAttrib,

9.2.57 AttrIDSetObjectRefPtrAttrib (attribute_id.c:379)

attributes

```
void AttrIDSetObjectRefPtrAttrib(const IPObjectStruct *PObj,
                                 IPAttrIDType ID,
                                 VoidPtr Data)
```

PObj: To add a pointer reference attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

ID: ID of attribute.

Data: Content of attribute.

Returns: void

Description: Routine to set a pointer reference attribute for an object.
See also: AttrIDSetObjectIntAttrib, AttrIDGetObjectIntAttrib, , AttrIDSetObjectRealAttrib, AttrIDGetObjectRealAttrib, , AttrIDSetObjectRealPtrAttrib, AttrIDGetObjectRealPtrAttrib, , AttrIDSetObjectUVAttrib, AttrIDGetObjectUVAttrib, , AttrIDGetObjectStrAttrib, AttrIDSetObjectObjAttrib, AttrIDSetObjAttrib, , AttrIDGetObjectObjAttrib, AttrIDGetObjAttrib AttrIDGetObjectPtrAttrib, , AttrIDSetObjectRefPtrAttrib, AttrIDGetObjectRefPtrAttrib, , AttrIDSetObjectStrAttrib,

9.2.58 AttrIDSetObjectStrAttrib (attribute_id.c:627)

attributes

```
void AttrIDSetObjectStrAttrib(const IPObjectStruct *PObj,
                              IPAttrIDType ID,
                              const char *Data)
```

PObj: To add a string attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

ID: ID of attribute.

Data: Content of attribute.

Returns: void

Description: Routine to set a string attribute for an object.
See also: AttrIDSetObjectIntAttrib, AttrIDGetObjectIntAttrib, , AttrIDGetObjectPtrAttrib, AttrIDSetObjectRealAttrib, , AttrIDSetObjectRealPtrAttrib, AttrIDGetObjectRealPtrAttrib, , AttrIDSetObjectUVAttrib, AttrIDGetObjectUVAttrib, , AttrIDSetObjectObjAttrib, AttrIDSetObjAttrib, , AttrIDSetObjectPtrAttrib, AttrIDGetObjectStrAttrib, , AttrIDGetObjectObjAttrib, AttrIDGetObjectRealAttrib, AttrIDGetObjAttrib,

9.2.59 AttrIDSetObjectUVAttrib (attribute_id.c:567)

attributes

```
void AttrIDSetObjectUVAttrib(const IPObjectStruct *PObj,
                             IPAttrIDType ID,
                             IrtRType U,
                             IrtRType V)
```

PObj: To add a real attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

ID: ID of attribute.

U, V: Content of attribute.

Returns: void

Description: Routine to set a UV attribute for an object.
See also: AttrIDSetObjectIntAttrib, AttrIDGetObjectIntAttrib, , AttrIDGetObjectPtrAttrib, AttrIDGetObjectRealAttrib, , AttrIDSetObjectRealPtrAttrib, AttrIDGetObjectRealPtrAttrib, , AttrIDGetObjectStrAttrib, AttrIDSetObjectObjAttrib, AttrIDSetObjAttrib, , AttrIDGetObjectObjAttrib, AttrIDGetObjAttrib, AttrIDGetObjectUVAttrib, , AttrIDSetObjectPtrAttrib, AttrIDSetObjectStrAttrib,

9.2.60 AttrIDSetObjectWidth (attribute_id.c:199)

attributes

width

```
void AttrIDSetObjectWidth(const IPObjectStruct *PObj, IrtRType Width)
```

PObj: Object to set its width to Width. Note the semantics that the object itself is noty modified and is considered const - only its attributes are affected.

Width: New width for PObj.

Returns: void

Description: Routine to set the width of an object.
See also: AttrIDSetObjectColor, AttrIDGetObjectColor, AttrIDSetObjectRGBColor, , AttrIDGetObjectRGBColor, AttrIDGetObjectWidth, AttrIDSetObjectRealAttrib, , AttrIDSetObjectRealPtrAttrib, AttrIDSetWidth,

9.2.61 AttrMergeGeomSimilarAttrs (attribut.c:2113)

attributes

```
IPObjectStruct *AttrMergeGeomSimilarAttrs(const IPObjectStruct *PObjList,
                                          const char *AttrName,
                                          int MergeOptions)
```

PObjList: A list object to merge objects in the list with similar attrs.

AttrName: The name of the attribute to merge along.

MergeOptions: Merging options as follows: 1. Similar attribute objects are placed in one list object under the returned list. 2. Similar attribute objects are placed in one object, chaining the geometry itself (polygons, curves, etc.) in a linked list.

Returns: A new list where elements in the input with similar name (and value) were merged into one object.

Description: Routine to merge similar geometries (polygon, surfaces, etc.) into one IPObjectStruct, if they share the same designated attribute value.

9.2.62 AttrPropagateAttr (attribut.c:1685)

attributes

files

parser

```
void AttrPropagateAttr(const IPObjectStruct *PObj, const char *AttrName)
```

PObj: To propagate down Attr attributes.

AttrName: Name of attribute to propagate or NULL to propagate all attributes.

Returns: void

Description: Propagate attributes from list objects down into their elements. Non propagatable attributes are accumulated or ignored as follows: "animation" is accumulated. "invisible" is ignored.

9.2.63 AttrPropagateRGB2Vrtx (attribut.c:1881)

attributes

files

parser

```
void AttrPropagateRGB2Vrtx(const IPObjectStruct *PObj)
```

PObj: To propagate down "RGB" attributes.

Returns: void

Description: Propagates "RGB" attributes from a poly objects down into the vertices, in place.

9.2.64 AttrSetObjAttrib (attribut.c:1187)

attributes

```
void AttrSetObjAttrib(IPAttributeStruct **Attrs,
                      const char *Name,
                      IPObjectStruct *Data,
                      int CopyData)
```

Attrs: Attribute list where to place new attribute.

Name: Name of the newly introduced attribute.

Data: Pointer attribute to save.

CopyData: If TRUE, object Data is duplicated first.

Returns: void

Description: Routine to set an object attribute.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrSetObjectStrAttrib, , AttrGetObjectStrAttrib, AttrSetObjectObjAttrib, AttrGetObjectObjAttrib, AttrGetObjAttrib, AttrSetObjAttrib2,

9.2.65 AttrSetObjAttrib2 (attribut.c:1234)

attributes

```
void AttrSetObjAttrib2(IPAttributeStruct **Attrs,
                       IPAttrNumType AttribNum,
                       IPObjectStruct *Data,
                       int CopyData)
```

Attrs: Attribute list where to place new attribute.

AttribNum: Unique ID derived from name of requested attribute.

Data: Pointer attribute to save.

CopyData: If TRUE, object Data is duplicated first.

Returns: void

Description: Routine to set an object attribute.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrSetObjectStrAttrib, , AttrGetObjectStrAttrib, AttrSetObjectObjAttrib, AttrGetObjectObjAttrib, AttrGetObjAttrib, AttrSetObjAttrib,

9.2.66 AttrSetObjectColor (attribut.c:66)

attributes

color

```
void AttrSetObjectColor(const IPObjectStruct *PObj, int Color)
```

PObj: Object to set its color to Color. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

Color: New color for PObj.

Returns: void

Description: Routine to set the color of an object.
See also: AttrGetObjectColor, AttrSetObjectRGBColor, AttrGetObjectRGBColor, , AttrGetObjectWidth, AttrSetObjectWidth, AttrSetObjectIntAttrib, , AttrIDSetColor,

9.2.67 AttrSetObjectIntAttrib (attribut.c:258)

attributes

```
void AttrSetObjectIntAttrib(const IPObjectStruct *PObj,
                            const char *Name,
                            int Data)
```

PObj: To add an integer attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

Name: Name of attribute.

Data: Content of attribute.

Returns: void

Description: Routine to set an integer attribute for an object.
See also: AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, AttrGetObjectPtrAttrib, , AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrSetObjectStrAttrib, , AttrGetObjectStrAttrib, AttrSetObjectObjAttrib, AttrSetObjAttrib, AttrGetObjectObjAttrib, AttrGetObjAttrib, AttrSetObjectIntAttrib2,

9.2.68 AttrSetObjectIntAttrib2 (attribut.c:290)

attributes

```
void AttrSetObjectIntAttrib2(const IPObjectStruct *PObj,
                             IPAttrNumType AttribNum,
                             int Data)
```

PObj: To add an integer attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

AttribNum: Unique ID derived from name of requested attribute.

Data: Content of attribute.

Returns: void

Description: Routine to set an integer attribute for an object.
See also: AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, AttrGetObjectPtrAttrib, , AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrSetObjectStrAttrib, , AttrGetObjectStrAttrib, AttrSetObjectObjAttrib, AttrSetObjAttrib, AttrGetObjectObjAttrib, AttrGetObjAttrib, AttrSetObjectIntAttrib,

9.2.69 AttrSetObjectObjAttrib (attribut.c:1119)

attributes

```
void AttrSetObjectObjAttrib(const IPObjectStruct *PObj,
                            const char *Name,
                            IPObjectStruct *Data,
                            int CopyData)
```

PObj: To add an object attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

Name: Name of attribute.

Data: Content of attribute.

CopyData: If TRUE, Data object is duplicated first.

Returns: void

Description: Routine to set an object attribute for an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrSetObjectStrAttrib, , AttrGetObjectStrAttrib, AttrSetObjAttrib, AttrGetObjectObjAttrib, AttrGetObjAttrib, AttrSetObjectObjAttrib2,

9.2.70 AttrSetObjectObjAttrib2 (attribut.c:1154)

attributes

```
void AttrSetObjectObjAttrib2(const IPObjectStruct *PObj,
                             IPAttrNumType AttribNum,
                             IPObjectStruct *Data,
                             int CopyData)
```

PObj: To add an object attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

AttribNum: Unique ID derived from name of requested attribute.

Data: Content of attribute.

CopyData: If TRUE, Data object is duplicated first.

Returns: void

Description: Routine to set an object attribute for an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrSetObjectStrAttrib, , AttrGetObjectStrAttrib, AttrSetObjAttrib, AttrGetObjectObjAttrib, AttrGetObjAttrib, AttrSetObjectObjAttrib,

9.2.71 AttrSetObjectPtrAttrib (attribut.c:379)

attributes

```
void AttrSetObjectPtrAttrib(const IPObjectStruct *PObj,
                            const char *Name,
                            VoidPtr Data)
```

PObj: To add a pointer attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

Name: Name of attribute.

Data: Content of attribute.

Returns: void

Description: Routine to set a pointer attribute for an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrGetObjectPtrAttrib, , AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrSetObjectStrAttrib, , AttrGetObjectStrAttrib, AttrSetObjectObjAttrib, AttrSetObjAttrib, , AttrGetObjectObjAttrib, AttrGetObjAttrib, , AttrSetObjectRefPtrAttrib, AttrGetObjectRefPtrAttrib, AttrSetObjectPtrAttrib2,

9.2.72 AttrSetObjectPtrAttrib2 (attribut.c:413)

attributes

```
void AttrSetObjectPtrAttrib2(const IPObjectStruct *PObj,
                             IPAttrNumType AttribNum,
                             VoidPtr Data)
```

PObj: To add a pointer attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

AttribNum: Unique ID derived from name of requested attribute.

Data: Content of attribute.

Returns: void

Description: Routine to set a pointer attribute for an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrGetObjectPtrAttrib, , AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrSetObjectStrAttrib, , AttrGetObjectStrAttrib, AttrSetObjectObjAttrib, AttrSetObjAttrib, , AttrGetObjectObjAttrib, AttrGetObjAttrib, , AttrSetObjectRefPtrAttrib, AttrGetObjectRefPtrAttrib, , AttrSetObjectPtrAttrib,

9.2.73 AttrSetObjectRGBColor (attribut.c:120)

attributes

color

rgb

```
void AttrSetObjectRGBColor(const IPObjectStruct *PObj,
                           int Red,
                           int Green,
                           int Blue)
```

PObj: Object to set its RGB color. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

Red, Green, Blue: Component of color.

Returns: void

Description: Routine to set the RGB color of an object.
See also: AttrSetObjectColor, AttrGetObjectColor, AttrGetObjectRGBColor, , AttrGetObjectWidth, AttrSetObjectWidth, AttrSetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, ,

9.2.74 AttrSetObjectRealAttrib (attribut.c:628)

attributes

```
void AttrSetObjectRealAttrib(const IPObjectStruct *PObj,
                             const char *Name,
                             IrtRType Data)
```

PObj: To add a real attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

Name: Name of attribute.

Data: Content of attribute.

Returns: void

Description: Routine to set a real attribute for an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrGetObjectRealAttrib, AttrSetObjectStrAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrGetObjectStrAttrib, , AttrSetObjectObjAttrib, AttrSetObjAttrib, AttrGetObjectObjAttrib, , AttrGetObjAttrib, AttrSetObjectRealAttrib2,

9.2.75 AttrSetObjectRealAttrib2 (attribut.c:660)

attributes

```
void AttrSetObjectRealAttrib2(const IPObjectStruct *PObj,
                              IPAttrNumType AttribNum,
                              IrtRType Data)
```

PObj: To add a real attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

AttribNum: Unique ID derived from name of requested attribute.

Data: Content of attribute.

Returns: void

Description: Routine to set a real attribute for an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrGetObjectRealAttrib, AttrSetObjectStrAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrGetObjectStrAttrib, , AttrSetObjectObjAttrib, AttrSetObjAttrib, AttrGetObjectObjAttrib, , AttrGetObjAttrib, AttrSetObjectRealAttrib,

9.2.76 AttrSetObjectRealPtrAttrib (attribut.c:751)

attributes

```
void AttrSetObjectRealPtrAttrib(const IPObjectStruct *PObj,
                                const char *Name,
                                IrtRType *Data,
                                int DataLen)
```

PObj: To add a real * attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

Name: Name of attribute.

Data: Content of attribute.

DataLen: If positive allocates and copies that many reals from Data. Otherwise, Data (of length -DataLen) is used directly.

Returns: void

Description: Routine to set a real * attribute for an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrGetObjectRealAttrib, AttrSetObjectStrAttrib, , AttrSetObjectRealPtrAttrib2, AttrGetObjectRealPtrAttrib2, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrGetObjectStrAttrib, , AttrSetObjectObjAttrib, AttrSetObjAttrib, AttrGetObjectObjAttrib, , AttrGetObjAttrib, AttrSetObjectRealAttrib2,

9.2.77 AttrSetObjectRealPtrAttrib2 (attribut.c:787)

attributes

```
void AttrSetObjectRealPtrAttrib2(const IPObjectStruct *PObj,
                                 IPAttrNumType AttribNum,
                                 IrtRType *Data,
                                 int DataLen)
```

PObj: To add a real * attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

AttribNum: Unique ID derived from name of requested attribute.

Data: Content of attribute.

DataLen: If positive allocates and copies that many reals from Data. Otherwise, Data (of length -DataLen) is used directly.

Returns: void

Description: Routine to set a real * attribute for an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrGetObjectRealAttrib, AttrSetObjectStrAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrGetObjectStrAttrib, , AttrSetObjectObjAttrib, AttrSetObjAttrib, AttrGetObjectObjAttrib, , AttrGetObjAttrib, AttrSetObjectRealAttrib,

9.2.78 AttrSetObjectRefPtrAttrib (attribut.c:504)

attributes

```
void AttrSetObjectRefPtrAttrib(const IPObjectStruct *PObj,
                               const char *Name,
                               VoidPtr Data)
```

PObj: To add a pointer reference attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

Name: Name of attribute.

Data: Content of attribute.

Returns: void

Description: Routine to set a pointer reference attribute for an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrGetObjectPtrAttrib, , AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrSetObjectStrAttrib, , AttrGetObjectStrAttrib, AttrSetObjectObjAttrib, AttrSetObjAttrib, , AttrGetObjectObjAttrib, AttrGetObjAttrib, , AttrSetObjectRefPtrAttrib, AttrGetObjectRefPtrAttrib, AttrSetObjectRefPtrAttrib2,

9.2.79 AttrSetObjectRefPtrAttrib2 (attribut.c:538)

attributes

```
void AttrSetObjectRefPtrAttrib2(const IPObjectStruct *PObj,
                                IPAttrNumType AttribNum,
                                VoidPtr Data)
```

PObj: To add a pointer reference attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

AttribNum: Unique ID derived from name of requested attribute.

Data: Content of attribute.

Returns: void

Description: Routine to set a pointer reference attribute for an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrGetObjectPtrAttrib, , AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrSetObjectStrAttrib, , AttrGetObjectStrAttrib, AttrSetObjectObjAttrib, AttrSetObjAttrib, , AttrGetObjectObjAttrib, AttrGetObjAttrib, , AttrSetObjectRefPtrAttrib, AttrGetObjectRefPtrAttrib, , AttrSetObjectRefPtrAttrib,

9.2.80 AttrSetObjectStrAttrib (attribut.c:997)

attributes

```
void AttrSetObjectStrAttrib(const IPObjectStruct *PObj,
                            const char *Name,
                            const char *Data)
```

PObj: To add a string attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

Name: Name of attribute.

Data: Content of attribute.

Returns: void

Description: Routine to set a string attribute for an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrGetObjectStrAttrib, , AttrSetObjectObjAttrib, AttrSetObjAttrib, AttrGetObjectObjAttrib, , AttrGetObjAttrib, AttrSetObjectStrAttrib2,

9.2.81 AttrSetObjectStrAttrib2 (attribut.c:1029)

attributes

```
void AttrSetObjectStrAttrib2(const IPObjectStruct *PObj,
                             IPAttrNumType AttribNum,
                             const char *Data)
```

PObj: To add a string attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

AttribNum: Unique ID derived from name of requested attribute.

Data: Content of attribute.

Returns: void

Description: Routine to set a string attribute for an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrSetObjectRealAttrib, AttrGetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrSetObjectUVAttrib, AttrGetObjectUVAttrib, AttrGetObjectStrAttrib, , AttrSetObjectObjAttrib, AttrSetObjAttrib, AttrGetObjectObjAttrib, , AttrGetObjAttrib, AttrSetObjectStrAttrib,

9.2.82 AttrSetObjectUVAttrib (attribut.c:877)

attributes

```
void AttrSetObjectUVAttrib(const IPObjectStruct *PObj,
                           const char *Name,
                           IrtRType U,
                           IrtRType V)
```

PObj: To add a UV attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

Name: Name of attribute.

U, V: Content of attribute.

Returns: void

Description: Routine to set a UV attribute for an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrGetObjectRealAttrib, AttrSetObjectStrAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrGetObjectStrAttrib, AttrSetObjectObjAttrib, AttrSetObjAttrib, , AttrGetObjectObjAttrib, AttrGetObjAttrib, AttrGetObjectUVAttrib, , AttrSetObjectUVAttrib2,

9.2.83 AttrSetObjectUVAttrib2 (attribut.c:910)

attributes

```
void AttrSetObjectUVAttrib2(const IPObjectStruct *PObj,
                            IPAttrNumType AttribNum,
                            IrtRType U,
                            IrtRType V)
```

PObj: To add a UV attribute for. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

AttribNum: Unique ID derived from name of requested attribute.

U, V: Content of attribute.

Returns: void

Description: Routine to set a UV attribute for an object.
See also: AttrSetObjectIntAttrib, AttrGetObjectIntAttrib, AttrSetObjectPtrAttrib, , AttrGetObjectPtrAttrib, AttrGetObjectRealAttrib, AttrSetObjectStrAttrib, , AttrSetObjectRealPtrAttrib, AttrGetObjectRealPtrAttrib, , AttrGetObjectStrAttrib, AttrSetObjectObjAttrib, AttrSetObjAttrib, , AttrGetObjectObjAttrib, AttrGetObjAttrib, AttrGetObjectUVAttrib, , AttrSetObjectUVAttrib,

9.2.84 AttrSetObjectWidth (attribut.c:202)

attributes

width

```
void AttrSetObjectWidth(const IPObjectStruct *PObj, IrtRType Width)
```

PObj: Object to set its width to Width. Note the semantics that the object itself is not modified and is considered const - only its attributes are affected.

Width: New width for PObj.

Returns: void

Description: Routine to set the width of an object.
See also: AttrSetObjectColor, AttrGetObjectColor, AttrSetObjectRGBColor, , AttrGetObjectRGBColor, AttrGetObjectWidth, AttrSetObjectRealAttrib, , AttrSetObjectRealPtrAttrib, ,

9.2.85 BspCrvReadFromFile (bsp_read.c:34)

files

read

```
CagdCrvStruct *BspCrvReadFromFile(const char *FileName,
                                  char **ErrStr,
                                  int *ErrLine)
```

FileName: To read the B-spline curve from.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read curve, or NULL if an error occurred.

Description: Reads from a file B-spline curve(s). If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occurred in file. If no error is detected *ErrStr = NULL.

9.2.86 BspCrvReadFromFile2 (bsp_read.c:99)

files

read

stream

```
CagdCrvStruct *BspCrvReadFromFile2(int Handler,
                                   CagdBType NameWasRead,
                                   char **ErrStr,
                                   int *ErrLine)
```

Handler: A handler to the open stream.

NameWasRead: TRUE if "[CURVE BSPLINE" has been read, FALSE otherwise.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read curve, or NULL if an error occurred.

Description: Reads from a file a B-spline curve. If NameWasRead is TRUE, it is assumed prefix "[CURVE BSPLINE" has already been read. This is useful for a global parser which invokes this routine to read from a file several times as a parent controller. For exactly this reason, the given file descriptor is NOT closed in the end. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occurred in file relative to beginning of curve. If no error is detected *ErrStr is set to NULL.

9.2.87 BspCrvWriteToFile (bsp_wrt.c:39)

files

write

```
int BspCrvWriteToFile(const CagdCrvStruct *Crvs,
                      const char *FileName,
                      int Indent,
                      const char *Comment,
                      char **ErrStr)
```

Crvs: To write to file FileName.

FileName: Name of file to open so we can write Crvs in.

Indent: Primary indentation. All information will be written from the column specified by Indent.

Comment: Optional, to describe the geometry.

ErrStr: If an error occurs, to describe the error.

Returns: TRUE if succesful, FALSE otherwise.

Description: Writes Bspline curve(s) list into file. Returns TRUE if succesful, FALSE otherwise. If Comment is NULL, no comment is wrriten, if "" only internal comment is written.

9.2.88 BspCrvWriteToFile2 (bsp_wrt.c:83)

files

write

```
int BspCrvWriteToFile2(const CagdCrvStruct *Crvs,
                       int Handler,
                       int Indent,
                       const char *Comment,
                       char **ErrStr)
```

Crvs: To write to open stream.

Handler: A handler to the open stream.

Indent: Primary indentation. All information will be written from the column specified by Indent.

Comment: Optional, to describe the geometry.

ErrStr: If an error occurs, to describe the error.

Returns: TRUE if successful, FALSE otherwise.

Description: Writes B-spline curve(s) list into file. Returns TRUE if successful, FALSE otherwise. The file descriptor is not closed. If Comment is NULL, no comment is written, if "" only internal comment is written.

9.2.89 BspSrfReadFromFile (bsp_read.c:286)

files

read

```
CagdSrfStruct *BspSrfReadFromFile(const char *FileName,
                                  char **ErrStr,
                                  int *ErrLine)
```

FileName: To read the Bspline surface from.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read surface, or NULL if an error occurred.

Description: Reads from a file B-spline surface(s). If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occurred in file. If no error is detected *ErrStr = NULL.

9.2.90 BspSrfReadFromFile2 (bsp_read.c:351)

files
read
stream

```
CagdSrfStruct *BspSrfReadFromFile2(int Handler,
                                   CagdBType NameWasRead,
                                   char **ErrStr,
                                   int *ErrLine)
```

Handler: A handler to the open stream.

NameWasRead: TRUE if "[SURFACE BSPLINE" has been read, FALSE otherwise.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read surface, or NULL if an error occurred.

Description: Reads from a file a B-spline surface. If NameWasRead is TRUE, it is assumed prefix "[SURFACE BSPLINE" has already been read. This is useful for a global parser which invokes this routine to read from a file several times as a parent controller. For exactly this reason, the given file descriptor is NOT closed in the end. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occurred in file relative to beginning of surface. If no error is detected *ErrStr is set to NULL.

9.2.91 BspSrfWriteToFile (bsp_wrt.c:187)

files
write

```
int BspSrfWriteToFile(const CagdSrfStruct *Srfs,
                      const char *FileName,
                      int Indent,
                      const char *Comment,
                      char **ErrStr)
```

Srfs: To write to file FileName.

FileName: Name of file to open so we can write Srfs in.

Indent: Primary indentation. All information will be written from the column specified by Indent.

Comment: Optional, to describe the geometry.

ErrStr: If an error occurs, to describe the error.

Returns: TRUE if succesful, FALSE otherwise.

Description: Writes Bspline surface(s) list into file. Returns TRUE if succesful, FALSE otherwise. If Comment is NULL, no comment is wrriten, if "" only internal comment is written.

9.2.92 BspSrfWriteToFile2 (bsp_wrt.c:231)

files
write

```
int BspSrfWriteToFile2(const CagdSrfStruct *Srfs,
                       int Handler,
                       int Indent,
                       const char *Comment,
                       char **ErrStr)
```

Srfs: To write to open stream.

Handler: A handler to the open stream.

Indent: Primary indentation. All information will be written from the column specified by Indent.

Comment: Optional, to describe the geometry.

ErrStr: If an error occurs, to describe the error.

Returns: TRUE if successful, FALSE otherwise.

Description: Writes Bspline surface(s) list into file. Returns TRUE if succesful, FALSE otherwise. The file descriptor is not closed. If Comment is NULL, no comment is wrriten, if "" only internal comment is written.

9.2.93 BzrCrvReadFromFile (bzr_read.c:34)

files

read

```
CagdCrvStruct *BzrCrvReadFromFile(const char *FileName,
                                  char **ErrStr,
                                  int *ErrLine)
```

FileName: To read the Bezier curve from.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read curve, or NULL if an error occurred.

Description: Reads from a file Bezier curve(s). If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occurred in file. If no error is detected *ErrStr = NULL.

9.2.94 BzrCrvReadFromFile2 (bzr_read.c:99)

files

read

stream

```
CagdCrvStruct *BzrCrvReadFromFile2(int Handler,
                                   CagdBType NameWasRead,
                                   char **ErrStr,
                                   int *ErrLine)
```

Handler: A handler to the open stream.

NameWasRead: TRUE if "[CURVE BEZIER" has been read, FALSE otherwise.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read curve, or NULL if an error occurred.

Description: Reads from a file a Bezier curve. If NameWasRead is TRUE, it is assumed prefix "[CURVE BEZIER" has already been read. This is useful for a global parser which invokes this routine to read from a file several times as a parent controller. For exactly this reason, the given file descriptor is NOT closed in the end. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occurred in file relative to beginning of curve. If no error is detected *ErrStr is set to NULL.

9.2.95 BzrCrvWriteToFile (bzr_wrt.c:39)

files

write

```
int BzrCrvWriteToFile(const CagdCrvStruct *Crvs,
                      const char *FileName,
                      int Indent,
                      const char *Comment,
                      char **ErrStr)
```

Crvs: To write to file FileName.

FileName: Name of file to open so we can write Crvs in.

Indent: Primary indentation. All information will be written from the column specified by Indent.

Comment: Optional, to describe the geometry.

ErrStr: If an error occurs, to describe the error.

Returns: TRUE if succesful, FALSE otherwise.

Description: Writes Bezier curve(s) list into file. Returns TRUE if succesful, FALSE otherwise. If Comment is NULL, no comment is wrriten, if "" only internal comment is written.

9.2.96 BzrCrvWriteToFile2 (bzr_wrt.c:83)

files

write

```
int BzrCrvWriteToFile2(const CagdCrvStruct *Crvs,
                       int Handler,
                       int Indent,
                       const char *Comment,
                       char **ErrStr)
```

Crvs: To write to open stream.

Handler: A handler to the open stream.

Indent: Primary indentation. All information will be written from the column specified by Indent.

Comment: Optional, to describe the geometry.

ErrStr: If an error occurs, to describe the error.

Returns: TRUE if successful, FALSE otherwise.

Description: Writes Bezier curve(s) list into file. Returns TRUE if successful, FALSE otherwise. The file descriptor is not closed. If Comment is NULL, no comment is written, if "" only internal comment is written.

9.2.97 BzrSrfReadFromFile (bzr_read.c:233)

files

read

```
CagdSrfStruct *BzrSrfReadFromFile(const char *FileName,
                                  char **ErrStr,
                                  int *ErrLine)
```

FileName: To read the Bezier surface from.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read surface, or NULL if an error occurred.

Description: Reads from a file Bezier surface(s). If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occurred in file. If no error is detected *ErrStr = NULL.

9.2.98 BzrSrfReadFromFile2 (bzr_read.c:298)

files

read

stream

```
CagdSrfStruct *BzrSrfReadFromFile2(int Handler,
                                   CagdBType NameWasRead,
                                   char **ErrStr,
                                   int *ErrLine)
```

Handler: A handler to the open stream.

NameWasRead: TRUE if "[SURFACE BEZIER" has been read, FALSE otherwise.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read surface, or NULL if an error occurred.

Description: Reads from a file a Bezier surface. If NameWasRead is TRUE, it is assumed prefix "[SURFACE BEZIER" has already been read. This is useful for a global parser which invokes this routine to read from a file several times as a parent controller. For exactly this reason, the given file descriptor is NOT closed in the end. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occurred in file relative to beginning of surface. If no error is detected *ErrStr is set to NULL.

9.2.99 BzrSrfWriteToFile (bzr_wrt.c:180)

files

write

```
int BzrSrfWriteToFile(const CagdSrfStruct *Srfs,
                      const char *FileName,
                      int Indent,
                      const char *Comment,
                      char **ErrStr)
```

Srfs: To write to file FileName.

FileName: Name of file to open so we can write Srfs in.

Indent: Primary indentation. All information will be written from the column specified by Indent.

Comment: Optional, to describe the geometry.

ErrStr: If an error occurs, to describe the error.

Returns: TRUE if succesful, FALSE otherwise.

Description: Writes Bezier surface(s) list into file. Returns TRUE if succesful, FALSE otherwise. If Comment is NULL, no comment is wrriten, if "" only internal comment is written.

9.2.100 BzrSrfWriteToFile2 (bzr_wrt.c:224)

files

write

```
int BzrSrfWriteToFile2(const CagdSrfStruct *Srfs,
                       int Handler,
                       int Indent,
                       const char *Comment,
                       char **ErrStr)
```

Srfs: To write to open stream.

Handler: A handler to the open stream.

Indent: Primary indentation. All information will be written from the column specified by Indent.

Comment: Optional, to describe the geometry.

ErrStr: If an error occurs, to describe the error.

Returns: TRUE if successful, FALSE otherwise.

Description: Writes Bezier surface(s) list into file. Returns TRUE if successful, FALSE otherwise. The file descriptor is not closed. If Comment is NULL, no comment is written, if "" only internal comment is written.

9.2.101 CagdCrvReadFromFile (cagdread.c:31)

files

read

```
CagdCrvStruct *CagdCrvReadFromFile(const char *FileName,
                                   char **ErrStr,
                                   int *ErrLine)
```

FileName: To read the curve from.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read curve, or NULL if an error occurred.

Description: Reads from a file curve(s). If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occurred in file. If no error is detected *ErrStr = NULL.

9.2.102 CagdCrvReadFromFile2 (cagdread.c:170)

files
read
stream

```
CagdCrvStruct *CagdCrvReadFromFile2(int Handler, char **ErrStr, int *ErrLine)
```

Handler: A handler to the open stream.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in stream of the error, if occurred.

Returns: The read curve, or NULL if an error occurred.

Description: Reads from a stream a curve. It is assumed prefix "[CURVE" has already been read. This is useful for a global parser which invokes this routine to read from a stream several times as a parent controller. For exactly this reason, the given stream descriptor is NOT closed in the end. If error is found in reading the stream, ErrStr is set to a string describing it and ErrLine to line it occurred in stream relative to beginning of curve. If no error is detected *ErrStr is set to NULL.

9.2.103 CagdCrvWriteToFile (cagd_wrt.c:34)

files
write

```
int CagdCrvWriteToFile(const CagdCrvStruct *Crvs,
                       const char *FileName,
                       int Indent,
                       const char *Comment,
                       char **ErrStr)
```

Crvs: To be saved in file f.

FileName: File name where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if successful, FALSE otherwise.

Description: Generic routine to write curve(s) to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.104 CagdCrvWriteToFile2 (cagd_wrt.c:75)

files
write

```
int CagdCrvWriteToFile2(const CagdCrvStruct *Crvs,
                        int Handler,
                        int Indent,
                        const char *Comment,
                        char **ErrStr)
```

Crvs: To be saved in stream.

Handler: A handler to the open stream.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if successful, FALSE otherwise.

Description: Generic routine to write curve(s) to the given stream. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.105 CagdCrvWriteToFile3 (cagd_wrt.c:116)

files

write

```
int CagdCrvWriteToFile3(const CagdCrvStruct *Crvs,
                        FILE *f,
                        int Indent,
                        const char *Comment,
                        char **ErrStr)
```

Crvs: To be saved in file f.

f: File descriptor where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write curve(s) to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.106 CagdSrfReadFromFile (cagdread.c:99)

files

read

```
CagdSrfStruct *CagdSrfReadFromFile(const char *FileName,
                                   char **ErrStr,
                                   int *ErrLine)
```

FileName: To read the surface from.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read surface, or NULL if an error occurred.

Description: Reads from a file surface(s). If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occurred in file. If no error is detected *ErrStr = NULL.

9.2.107 CagdSrfReadFromFile2 (cagdread.c:231)

files

read

stream

```
CagdSrfStruct *CagdSrfReadFromFile2(int Handler, char **ErrStr, int *ErrLine)
```

Handler: A handler to the open stream.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in stream of the error, if occurred.

Returns: The read surface, or NULL if an error occurred.

Description: Reads from a stream a surface. It is assumed prefix "[SURFACE" has already been read. This is useful for a global parser which invokes this routine to read from a stream several times as a parent controller. For exactly this reason, the given stream descriptor is NOT closed in the end. If error is found in reading the stream, ErrStr is set to a string describing it and ErrLine to line it occurred in stream relative to beginning of surface. If no error is detected *ErrStr is set to NULL.

9.2.108 CagdSrfWriteToFile (cagd_wrt.c:148)

files

write

```
int CagdSrfWriteToFile(const CagdSrfStruct *Srfs,
                       const char *FileName,
                       int Indent,
                       const char *Comment,
                       char **ErrStr)
```

Srfs: To be saved in file f.

FileName: File name where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if successful, FALSE otherwise.

Description: Generic routine to write surface(s) to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.109 CagdSrfWriteToFile2 (cagd_wrt.c:189)

files

write

stream

```
int CagdSrfWriteToFile2(const CagdSrfStruct *Srfs,
                        int Handler,
                        int Indent,
                        const char *Comment,
                        char **ErrStr)
```

Srfs: To be saved in stream.

Handler: A handler to the open stream.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if successful, FALSE otherwise.

Description: Generic routine to write surface(s) to the given stream. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.110 CagdSrfWriteToFile3 (cagd_wrt.c:230)

files

write

```
int CagdSrfWriteToFile3(const CagdSrfStruct *Srfs,
                        FILE *f,
                        int Indent,
                        const char *Comment,
                        char **ErrStr)
```

Srfs: To be saved in file f.

f: File descriptor where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write surface(s) to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.111 IP3MFSaveFile (irit_3mf.c:123)

```
int IP3MFSaveFile(IPObjectStruct *PObj,
                  const char *OutFileName,
                  int WarningMsgs,
                  const char *Designer)
```

PObj: IritObject structure to convert into 3MF.

OutFileName: Name of output 3MF file.

WarningMsgs: Whether to display warning messages or not.

Designer: Model Designer name, used as 3MF metadata.

Returns: TRUE for success FALSE for failure.

Description: Convert IRIT data structure into a 3MF file.
See also: IP3MFLoadFile,

9.2.112 IPAllocObject (allocate.c:324)

allocation

```
IPObjectStruct *IPAllocObject(const char *Name,
                              IPObjStructType ObjType,
                              IPObjectStruct *Pnext)
```

Name: Name to assign to the newly allocated object.
ObjType: Object type of newly allocated object.
Pnext: Reference to initialize the Pnext slot of the allocated object.
Returns: A new allocated object structure.
Description: Allocates one Object Structure.

9.2.113 IPAllocPolygon (allocate.c:293)

allocation

```
IPPolygonStruct *IPAllocPolygon(IrtBType Tags,
                                IPVertexStruct *V,
                                IPPolygonStruct *Pnext)
```

Tags: Tags to initialize the Tags slot of allocated polygon.
V: Reference to initialize the PVertex slot of allocated polygon.
Pnext: Reference to initialize the Pnext slot of allocated polygon.
Returns: A new allocated polygon structure.
Description: Allocates one Polygon Structure.

9.2.114 IPAllocVertex (allocate.c:265)

allocation

```
IPVertexStruct *IPAllocVertex(IrtBType Tags,
                              IPPolygonStruct *PAdj,
                              IPVertexStruct *Pnext)
```

Tags: Tags to initialize the Tags slot of allocated vertex.
PAdj: PAdj to initialize the PAdj slot of allocated vertex.
Pnext: Reference to initialize the Pnext slot of the allocated vertex.
Returns: A new allocated vertex structure.
Description: Allocates one Vertex Structure.

9.2.115 IPAllocVertex2 (allocate.c:236)

allocation

```
IPVertexStruct *IPAllocVertex2(IPVertexStruct *Pnext)
```

Pnext: Reference to initialize the Pnext slot of the allocated vertex.
Returns: A new allocated vertex structure.
Description: Allocates one Vertex Structure.

9.2.116 IPAppendListObjects (linklist.c:515)

```
IPObjectStruct *IPAppendListObjects(IPObjectStruct *ListObj1,
                                    IPObjectStruct *ListObj2)
```

ListObj1, ListObj2: The two list objects to append.
Returns: A combined list.
Description: Appends two lists.
See also: IPAppendObjLists, IPObjLnkListToListObject,

9.2.117 IPAppendObjLists (linklist.c:482)

linked lists

```
IPObjectStruct *IPAppendObjLists(IPObjectStruct *OList1,
                                 IPObjectStruct *OList2)
```

OList1, OList2: Two lists to append. Used in place.

Returns: Appended list.

Description: Appends two object lists together, in place.
See also: IPAppendListObjects, IPObjLnkListToListObject,

9.2.118 IPAppendPolyLists (linklist.c:406)

linked lists

```
IPPolygonStruct *IPAppendPolyLists(IPPolygonStruct *PList1,
                                   IPPolygonStruct *PList2)
```

PList1, PList2: Two lists to append. Used in place.

Returns: Appended list.

Description: Appends two poly lists together, in place.

9.2.119 IPAppendVrtxLists (linklist.c:332)

linked lists

```
IPVertexStruct *IPAppendVrtxLists(IPVertexStruct *VList1,
                                  IPVertexStruct *VList2)
```

VList1, VList2: Two lists to append. Used in placed.

Returns: Appended list.

Description: Appends two vertex lists together, in place.

9.2.120 IPCagdPlgns2IritPlgns (ip_cnvrt.c:193)

```
IPPolygonStruct *IPCagdPlgns2IritPlgns(CagdPolygonStruct *Polys,
                                       CagdBType ComputeUV)
```

Polys: Polygons in cagd library format to convert.

ComputeUV: Do we have UV values as well, at the vertices?

Returns: Same polygons in IRIT format.

Description: Routine to convert a cagd polygon (triangle) into irit polygon. Old cagd polygons are freed!
See also: IPCagdPllns2IritPllns, IPIritPlgns2CagdPlgns,

9.2.121 IPCagdPllns2IritPllns (ip_cnvrt.c:52)

```
IPPolygonStruct *IPCagdPllns2IritPllns(CagdPolylineStruct *Polys)
```

Polys: Polygons in cagd_lib form to be converted in IRIT form, in place.

Returns: Same polylines in IRIT format.

Description: Routine to convert a cagd polyline to irit polyline. Old cagd polylines are freed!
See also: IPCagdPlgns2IritPlgns, CagdCnvrtPolyline2LinBspCrv,

9.2.122 IPCloseStream (iritprs1.c:573)

files

stream

parser

```
void IPCloseStream(int Handler, int Free)
```

Handler: A handler to the open stream.

Free: If TRUE, release content.

Returns: void

Description: Close a data file for read/write.

9.2.123 IPClosedPolysToOpen (ip_cnvrt.c:2392)

```
void IPClosedPolysToOpen(IPPolygonStruct *Pls)
```

Pls: Polygons to process, in place.

Returns: void

Description: Forces the given list of polygons to have open list of vertices
See also: IPOpenPolysToClosed, GMVrtxListToCircOrLin,

9.2.124 IPCnvDataToIrit (cnv2irit.c:123)

```
void IPCnvDataToIrit(const IPObjectStruct *PObjects)
```

PObjects: To convert to .irt style.

Returns: void

Description: Converts the given Objects to .irt style. Output goes to the function IPCnvPrintFunc which echos the lines, one at a time.
See also: IPCnvSetPrintFunc, IPCnvSetDelimitChar, IPCnvDataToIritOneObject,

9.2.125 IPCnvDataToIritAttribs (cnv2irit.c:593)

```
void IPCnvDataToIritAttribs(const char *Indent,
                            const char *ObjName,
                            const IPAttributeStruct *Attr)
```

Indent: Level of indentation, as white spaces string.

ObjName: Name of object these attributes are for.

Attr: Attributes to convert to irit scripting format.

Returns: void

Description: Converts attributes PObject to .irt style
See also: IPCnvDataToIrit, IPCnvSetPrintFunc,

9.2.126 IPCnvDataToIritOneObject (cnv2irit.c:333)

```
void IPCnvDataToIritOneObject(const char *Indent,
                              const IPObjectStruct *PObj,
                              int Level)
```

Indent: Level of indentation, as white spaces string

PObj: Object to convert to .irt style.

Level: Nesting level of this object.

Returns: void

Description: Converts one object PObject to .irt style
See also: IPCnvDataToIrit, IPCnvSetPrintFunc,

9.2.127 IPCnvEstimateBndryVrtxPlaneNrml (cnv2irit.c:1770)

```
int IPCnvEstimateBndryVrtxPlaneNrml(const IPPolyVrtxArrayStruct *PVIdx,
                                    int BndryVrtxIdx,
                                    IrtVecType PlaneNrml)
```

PVIdx: A mesh to search in.

BndryVrtxIdx: Boundary vertex index to estimate the boundary plane here.

PlaneNrml: Where to save the computed boundary plane normal.

Returns: TRUE if successful, FALSE otherwise.

Description: Estimate, for a boundry vertex, the plane of the boundary.
See also: IPCnvFindAdjacentPoly, IPCnvFindAdjacentEdge, IPCnvPolyVrtxNeighbors, , IPCnvrtIritPolyToPolyVrtxArray, IPCnvIsVertexBoundary,

9.2.128 IPCnvFindAdjacentEdge (cnv2irit.c:1865)

```
IPVertexStruct *IPCnvFindAdjacentEdge(const IPPolyVrtxArrayStruct *PVIdx,
                                      int ThisPolyIdx,
                                      int FirstVertexIndex,
                                      int SecondVertexIndex)
```

PVIdx: A mesh to search the adjacent edge in.

ThisPolyIdx: If negative ignored. Otherwise, allow unoriented meshes and researches for the edge in the adjacent poly.

FirstVertexIndex, SecondVertexIndex: The edge to search its adjacent edge.

Returns: Pointer to the adjacent edge, or NULL if error.

Description: Find out the adjacent edge of the given edge defined by vertices (FirstVertexIndex, SecondVertexIndex). The edge in the found polygon is assumed to be revised as (SecondVertexIndex, FirstVertexIndex).
See also: IPCnvFindAdjacentPoly, IPCnvPolyVrtxNeighbors, IPCnvIsVertexBoundary, , IPCnvrtIritPolyToPolyVrtxArray,

9.2.129 IPCnvFindAdjacentPoly (cnv2irit.c:1940)

```
IPPolygonStruct *IPCnvFindAdjacentPoly(const IPPolyVrtxArrayStruct *PVIdx,
                                       const IPVertexStruct *V,
                                       const IPVertexStruct *VNext)
```

PVIdx: Data structure of original mesh.

V, VNext: The edge is defined from V to VNext.

Returns: Pointer to the adjacent polygon.

Description: Find out the adjacent polygon that use the edge (V, V -> Pnext). The edge in this polygon will be (V -> Pnext, V).
See also: IPCnvFindAdjacentEdge, IPCnvPolyVrtxNeighbors, IPCnvIsVertexBoundary, IPCnvrtIritPolyToPolyVrtxArray,

9.2.130 IPCnvIsVertexBoundary (cnv2irit.c:1716)

```
int IPCnvIsVertexBoundary(const IPPolyVrtxArrayStruct *PVIdx, int VertexIndex)
```

PVIdx: A mesh to search the adjacent edge in.

VertexIndex: The vertex index to examine if a boundary vertex.

Returns: TRUE if examined vertex is a boundary vertex, FALSE otherwise.

Description: Find out if the given vertex is a boundary vertex. A boundary vertex is a vertex that does not have polygons completely around it, forming a closed ring.
See also: IPCnvFindAdjacentPoly, IPCnvFindAdjacentEdge, IPCnvPolyVrtxNeighbors, , IPCnvrtIritPolyToPolyVrtxArray, IPCnvEstimateBndryVrtxPlaneNrml,

9.2.131 IPCnvPolyVrtxNeighbors (cnv2irit.c:1563)

```
int *IPCnvPolyVrtxNeighbors(const IPPolyVrtxArrayStruct *PVIdx,
                            int VIdx,
                            int Ring)
```

PVIdx: The input mesh to look at. Assumed that was constructed using IPCnvrtIritPolyToPolyVrtxArray with CalcPPolys TRUE.

VIdx: The index of the source vertex, zero based.

Ring: maximal topological distance from VIdx.

Returns: A -1 terminated vector holding the indices of neighboring vertices to VIdx, with topological distance of up to Ring. This vector is allocated and memory managed by this function.

Description: Given a vertex and a mesh in IPPolyVrtxArrayStruct format, find neighbors up to the prescribed maximal distance/ring.
See also: IPCnvrtIritPolyToPolyVrtxArray, IPCnvFindAdjacentEdge, , IPCnvFindAdjacentEdge, IPCnvIsVertexBoundary,

9.2.132 IPCnvSetCompactList (cnv2irit.c:236)

files

```
int IPCnvSetCompactList(int CompactList)
```

CompactList: TRUE for compact list, FALSE for separate entities..

Returns: Old state of compact list dump.

Description: Sets the way list objects are dumped - TRUE for a single list, FALSE for separated objects that are grouped into a list at the end.
See also: IPCnvDataToIrit,

9.2.133 IPCnvSetDelimitChar (cnv2irit.c:206)

files

```
char IPCnvSetDelimitChar(char Delimit)
```

Delimit: The character to consider as an expression delimiting char.

Returns: Old delimiting character.

Description: Sets the delimiting chararacter. Typically ';' but can be ':' as well.
See also: IPCnvDataToIrit,

9.2.134 IPCnvSetDumpAssignName (cnv2irit.c:265)

files

```
int IPCnvSetDumpAssignName(int DumpAssignName)
```

DumpAssignName: TRUE to dump assignment, FALSE for no assignment.

Returns: Old state of dump assignments.

Description: If TRUE objects are dumped with an assignment to their own name. Otherwise, just the geometry is dumped with no assignment.
See also: IPCnvDataToIrit,

9.2.135 IPCnvSetLeastSquaresFit (cnv2irit.c:179)

```
int IPCnvSetLeastSquaresFit(int MinLenFit, int Percent, IrtRType MaxError)
```

MinLenFit: Minimum number of control point to attempt a fit.

Percent: Percent of number of control points to fit to.

MaxError: maximum allowed error (in maximum norm).

Returns: Old percent value.

Description: Fits using least squares, a new curve to the input curve with only Percent percents control points. A curve with be least squares fitted if it has more than MinLenFit control points.
See also: IPCnvDataToIrit,

9.2.136 IPCnvSetPrintFunc (cnv2irit.c:146)

files

```
IPPrintFuncType IPCnvSetPrintFunc(IPPrintFuncType CnvPrintFunc)
```

CnvPrintFunc: A function that gets a single string it should print.

Returns: Old value of this state.

Description: Sets the printing function to call if needs to redirect printing of dat to irt conversions. Called (indirectly) by IPCnvDataToIrit.
See also: IPCnvDataToIrit,

9.2.137 IPCnvrtIritPolyToPolyVrtxArray (cnv2irit.c:1995)

```
IPPolyVrtxArrayStruct *IPCnvrtIritPolyToPolyVrtxArray(const IPObjectStruct
                                                                    *PObj,
                                                      int CalcPPolys,
                                                      int AttribMask)
```

PObj: A polygonal mesh to convert to PolyIdx structure.

CalcPPolys: TRUE if a polygon pointer list is to be calculated, FALSE otherwise.

AttribMask: Sets what attributes to consider when comparing for identical vertices - Bit 0 - normals should be identical in identical vertices. Bit 1 - uvvals should be identical in identical vertices. Bit 2 - rgb should be identical in identical vertices.

Returns: The polygonal mesh as PolyIdx struct.

Description: Process a given polygonal model into a vertex list with each polygon having indices into the vertex list. All lists are zero based.
See also: IPCnvFindAdjacentEdge, IPCnvFindAdjacentPoly, IPCnvPolyVrtxNeighbors, , IPCnvIsVertexBoundary,

9.2.138 IPCoerceBezierToBspline (coerce.c:232)

```
IPObjectStruct *IPCoerceBezierToBspline(const IPObjectStruct *PObj)
```

PObj: Bezier geometry to convert to Bspline geometry.

Returns: Same geometry as PObj but as Bspline.

Description: Converts a Bezier freeform into a Bspline freeform.

9.2.139 IPCoerceBezierToPower (coerce.c:154)

```
IPObjectStruct *IPCoerceBezierToPower(const IPObjectStruct *PObj)
```

PObj: Bezier geometry to convert to power geometry.

Returns: Same geometry as PObj but in power basis.

Description: Converts a Bezier freeform into a power freeform.

9.2.140 IPCoerceBsplineToBezier (coerce.c:287)

```
IPObjectStruct *IPCoerceBsplineToBezier(const IPObjectStruct *PObj)
```

PObj: A Bspline geometry to convert to a Bezier geometry.

Returns: A Bezier geometry representing same geometry as PObj.

Description: Convert a Bspline freeform into list of Bezier freeforms.

9.2.141 IPCoerceCommonSpace (coerce.c:85)

coercion

```
CagdPointType IPCoerceCommonSpace(IPObjectStruct *PtObjList,
                                  CagdPointType Type)
```

PtObjList: List of points.

Type: Point type that we must span its space as well.

Returns: Point type that spans the space of point type Type as well as all points in PtObjList.

Description: Given a set of points, returns the list's common denominator that spans the space of all the points, taking into account type Type.

9.2.142 IPCoerceGregoryToBezier (coerce.c:127)

```
IPObjectStruct *IPCoerceGregoryToBezier(const IPObjectStruct *PObj)
```

PObj: Gregory geometry to convert to Bezier geometry.

Returns: Same geometry as PObj but in Gregory basis.

Description: Converts a Gregory freeform into a Bezier freeform.

9.2.143 IPCoerceObjectPtTypeTo (coerce.c:609)

coercion

```
IPObjectStruct *IPCoerceObjectPtTypeTo(const IPObjectStruct *PObj, int NewType)
```

PObj: Object to coerce.

NewType: New type which can be object type like IP_OBJ_VECTOR or point type like E2.

Returns: Newly coerced object.

Description: Coerces an object to a new object. Also about point types coercions. Points, vectors, control points and planes can always be coerced between themselves using this routine by specifying the new object type desired such as IP_OBJ_PLANE or control point type like CAGD_PT_E4_TYPE. Control points of curves and surfaces may be coerced to a new type by prescribing the needed point type as NewType, such as CAGD_PT_P2_TYPE.
See also: IPCoerceObjectTo,

9.2.144 IPCoerceObjectTo (coerce.c:859)

```
IPObjectStruct *IPCoerceObjectTo(const IPObjectStruct *PObj, int NewType)
```

PObj: Object to coerce.

NewType: New type for PObj.

Returns: The newly coerced object.

Description: Coerce an object to a new object.
See also: IPCoerceObjectPtTypeTo,

9.2.145 IPCoercePowerToBezier (coerce.c:193)

```
IPObjectStruct *IPCoercePowerToBezier(const IPObjectStruct *PObj)
```

PObj: Power geometry to convert to Bezier geometry.

Returns: Same geometry as PObj but in Bezier form.

Description: Converts a power freeform into a Bezier freeform.

9.2.146 IPCoercePtsListTo (coerce.c:560)

coercion

```
CagdPointType IPCoercePtsListTo(IPObjectStruct *PtObjList, CagdPointType Type)
```

PtObjList: Coerce points/vectors/control points in this list to Type.

Type: A minimum space type to coerce to in PtObjList.

Returns: The coercion type actually took place with in PtObjList.

Description: Coerces a list of objects to Type.

9.2.147 IPCoerceTrimmedSrfToTrimmedBezier (coerce.c:424)

```
IPObjectStruct *IPCoerceTrimmedSrfToTrimmedBezier(const IPObjectStruct *PObj)
```

PObj: A Bspline geometry to convert to a Bezier geometry.

Returns: A Bezier geometry representing same geometry as PObj.

Description: Convert a Bspline freeform into list of Bezier freeforms.

9.2.148 IPCoerceTrimmedSrfToUnTrimmedBezier (coerce.c:480)

```
IPObjectStruct *IPCoerceTrimmedSrfToUnTrimmedBezier(const IPObjectStruct *PObj,
                                                    int ComposeE3)
```

PObj: A B-spline geometry to convert to a Bezier geometry.

ComposeE3: TRUE to compose the tiles into TSrf, FALSE to return the surface tiles in the parametric domain of TSrf.

Returns: A Bezier geometry representing same geometry as PObj.

Description: Convert a trimmed B-spline freeform into untrimmed tensor product Bezier freeforms.

9.2.149 IPConcatFreeForm (iritprs1.c:3335)

conversion

```
IPObjectStruct *IPConcatFreeForm(IPFreeFormStruct *FreeForms)
```

FreeForms: Freeform geometry to process.

Returns: concatenated linked list.

Description: Concatenate all freeform objects in FreeForms into a single list.

9.2.150 IPConvertFreeForm (ff_cnvrt.c:1508)

conversion

```
IPObjectStruct *IPConvertFreeForm(IPObjectStruct *PObj,
                                  IPFreeformConvStateStruct *State)
```

PObj: A Crv/Srf/Trimmed Srf/Trivariate freeform geometry.

State: The way the freeform geometry should be converted.

Returns: Processed freeform geometry.

Description: Routine to convert a single freeform geometry to polylines/polygons, in place.
See also: IPConvertFreeFormHierachy,

9.2.151 IPConvertFreeFormHierachy (ff_cnvrt.c:1426)

conversion

```
IPObjectStruct *IPConvertFreeFormHierachy(IPObjectStruct *PObj,
                                          IPFreeformConvStateStruct *State,
                                          int TriangleOnly,
                                          int Regularize)
```

PObj: A Crv/Srf/Trimmed Srf/Trivar freeform geometry, or a list obj.

State: The way the freeform geometry should be converted.

TriangleOnly: TRUE to also ensure all polygons are triangles.

Regularize: TRUE to ensure no T-junctions in the output.

Returns: Processed freeform geometry.

Description: Routine to convert a hierarchy of freeform geometry to polylines/polygons, in place.
See also: IPConvertFreeForm,

9.2.152 IPCopyAllVerticesFromPolys (ip_cnvrt.c:2722)

```
IPVertexStruct *IPCopyAllVerticesFromPolys(const IPObjectStruct *PObj)
```

PObj: A polygonal object.

Returns: A list of all unique vertices in PObj.

Description: Get a unique copy of all the vertices in given polygons object, PObj.
See also:

9.2.153 IPCopyObject (allocate.c:2270)

copy

```
IPObjectStruct *IPCopyObject(IPObjectStruct *Dest,
                             const IPObjectStruct *Src,
                             int CopyAll)
```

Dest: Destination object, possibly NULL.

Src: Source object.

CopyAll: Do we want a complete identical copy?

Returns: Duplicate of Src, same as Dest if Dest != NULL.

Description: Routine to create a whole new copy of an object Src into Dest. If Dest is NULL, new object is allocated, otherwise Dest itself is updated to hold the new copy. If CopyAll then all the record is copied, otherwise, only its invariant elements arebeen copied (i.e. no Name/Pnext copying).
See also: IPSetCopyObjectReferenceCount, IPCopyObjectAuxInfo,

9.2.154 IPCopyObject2 (allocate.c:2301)

copy

```
IPObjectStruct *IPCopyObject2(IPObjectStruct *Dest,
                              const IPObjectStruct *Src,
                              int CopyNext,
                              int CopyName)
```

Dest: Destination object, possibly NULL.

Src: Source object.

CopyNext: Copy the Pnext slot?

CopyName: Copy the name of the object?

Returns: Duplicate of Src, same as Dest if Dest != NULL.

Description: Routine to create a whole new copy of an object Src into Dest. If Dest is NULL, new object is allocated, otherwise Dest itself is updated to hold the new copy. If CopyAll then all the record is copied, otherwise, only its invariant elements are been copied (i.e. no Name/Pnext copying).
See also: IPSetCopyObjectReferenceCount, IPCopyObjectAuxInfo, IPCopyObject,

9.2.155 IPCopyObjectAuxInfo (allocate.c:2228)

```
void IPCopyObjectAuxInfo(IPObjectStruct *Dest, const IPObjectStruct *Src)
```

Dest: Destination of copy process.

Src: Source of copy process.

Returns: void

Description: Copy to the destination object all object auxiliary information such as attributes, dependencies, and bbox.
See also: IPCopyObject,

9.2.156 IPCopyObjectGeomData (allocate.c:2351)

copy

```
IPObjectStruct *IPCopyObjectGeomData(IPObjectStruct *Dest,
                                     const IPObjectStruct *Src,
                                     int CopyNext,
                                     int CopyName)
```

Dest: Destination object.

Src: Source object.

CopyNext: Copy the Pnext slot?

CopyName: Copy the name of the object?

Returns: Reference to Dest.

Description: Routine to copy the geometry in Src object to Dest.
See also: IPSetCopyObjectReferenceCount, IPCopyObjectAuxInfo, IPCopyObject,

9.2.157 IPCopyObjectList (allocate.c:2480)

copy

```
IPObjectStruct *IPCopyObjectList(const IPObjectStruct *PObjs, int CopyAll)
```

PObjs: Source objects.

CopyAll: Do we want a complete identical copy?

Returns: Duplicated list of PObjs.

Description: Routine to create a new copy of an object list.

9.2.158 IPCopyPolygon (allocate.c:2515)

copy

```
IPPolygonStruct *IPCopyPolygon(const IPPolygonStruct *Src)
```

Src: A polygon to copy.

Returns: Duplicated polygon.

Description: Routine to create a new copy of one polygon.
See also: IPCopyPolygonList,

9.2.159 IPCopyPolygonList (allocate.c:2552)

copy

```
IPPolygonStruct *IPCopyPolygonList(const IPPolygonStruct *Src)
```

Src: A polygon list to copy.

Returns: Duplicated list of polygons.

Description: Routine to create a new copy of an object polygon list.
See also: IPCopyPolygon,

9.2.160 IPCopyVertex (allocate.c:2588)

copy

```
IPVertexStruct *IPCopyVertex(const IPVertexStruct *Src)
```

Src: A vertex to copy.

Returns: Duplicated vertex.

Description: Routine to create a new copy of a polygon vertex.
See also: IPCopyVertexList,

9.2.161 IPCopyVertexList (allocate.c:2620)

copy

```
IPVertexStruct *IPCopyVertexList(const IPVertexStruct *Src)
```

Src: A vertex list to copy.

Returns: Duplicated list of vertices.

Description: Routine to create a new copy of a polygon vertices list.
See also: IPCopyVertex,

9.2.162 IPCurve2CtlPoly (ip_cnvrt.c:468)

conversion

```
IPPolygonStruct *IPCurve2CtlPoly(const CagdCrvStruct *Crv)
```

Crv: To extract its control polygon as a polyline.

Returns: A polyline representing Crv's control polygon.

Description: Routine to convert a single curve's control polygon into a polyline.

9.2.163 IPCurve2Polylines (ip_cnvrt.c:336)

conversion

approximation

```
IPPolygonStruct *IPCurve2Polylines(const CagdCrvStruct *Crv,
                                   CagdRType TolSamples,
                                   SymbCrvApproxMethodType Method)
```

Crv: To approximate as a polyline.

TolSamples: Tolerance of approximation error (Method = 2) or Number of samples to compute on polyline (Method = 0, 1).

Method: 0 - TolSamples are set uniformly in parametric space, 1 - TolSamples sets the maximum error allowed between the piecewise linear approximation and original curve. 2 - TolSamples are set optimally, considering the curve's curvature.

Returns: A polyline approximating Crv. Can be more than one polyline if Crv is C0 discont.

Description: Routine to convert one curve into a polyline with TolSamples samples/tolerance.
See also: BspCrv2Polyline, BzrCrv2Polyline, SymbCrv2Polyline,

9.2.164 IPCurvesToCubicBzrCrvs (ip_cnvrt.c:1967)

conversion

approximation

```
CagdCrvStruct *IPCurvesToCubicBzrCrvs(CagdCrvStruct *Crvs,
                                      IPPolygonStruct **CtlPolys,
                                      CagdBType DrawCurve,
                                      CagdBType DrawCtlPoly,
                                      CagdRType MaxArcLen)
```

Crvs: To approximate as cubic Bezier curves.

CtlPolys: If we want control polygons as well (DrawCtlPoly == TRUE) they will be placed herein.

DrawCurve: Do we want to draw the curves?

DrawCtlPoly: Do we want to draw the control polygons?

MaxArcLen: Tolerance for cubic Bezier approximation. See function SymbApproxCrvAsBzrCubics.

Returns: The cubic Bezier approximation, or NULL if DrawCurve is FALSE.

Description: Approximates an arbitrary list of curves into cubic Beziers curves.

9.2.165 IPDescribeError (prsr_err.c:134)

error handling

```
const char *IPDescribeError(IritPrsrFatalErrorType ErrorNum)
```

ErrorNum: Type of the error that was raised.

Returns: A string describing the error type.

Description: Returns a string describing a the given error. Errors can be raised by any member of this user library as well as other users. Raised error will cause an invocation of IritUserFatalError function which decides how to handle this error. IritUserFatalError can for example, invoke this routine with the error type, print the appropriate message and quit the program.

9.2.166 IPEPSSaveFile (irit_ps.c:96)

```
int IPEPSSaveFile(IPObjectStruct *PObjects,
                  IrtHmgnMatType CrntMat,
                  IPEPSSaveDfltFileParamsStruct *EPSParams)
```

PObjects: List of Irit objects to dump as EPS file.

CrntMat: Transformation matrix to apply (if not NULL).

EPSParams: The rest of the ESP file saving parameters.

Returns: TRUE if successful, FALSE otherwise.

Description: Dumps IRIT object as a (encapsulated) post script file.

9.2.167 IPEvalFreeForms (iritprs1.c:1847)

```
IPObjectStruct *IPEvalFreeForms(IPObjectStruct *PObj)
```

PObj: Object(s) to freeform evaluate, in place.

Returns: Evaluated hierarchy, in place.

Description: Evaluates the freeforms in the given hierarchy - usually to convert into a polygonal approximation. This function invokes IPProcessFreeForm for the evaluation of the individual freeform entities.
See also: IPProcessFreeForm,

9.2.168 IPExportObjectToFile (iritprs2.c:220)

files

```
void IPExportObjectToFile(const char *FName,
                          const IPObjectStruct *PObj,
                          IPStreamFormatType FType)
```

FName: Output file name.

PObj: Object to put on output.

FType: File type desired. Can be IP_FILE_ANY, in which case type is derived from the type in FName. See IPStreamFormatType.

Returns: void

Description: Routine to export the data from given object into given file FName, in the designated file format.

9.2.169 IPFlattenForrest (iritprs1.c:1917)

```
IPObjectStruct *IPFlattenForrest(IPObjectStruct *PObjList, int ProcessFF)
```

PObjList: List of object(s) to flatten out.

ProcessFF: If TRUE also process freeforms using IPFlattenTreeprocessFF.

Returns: Flattened hierarchy; destroyed in place.

Description: Flattens out a list of trees' hierarchy (a forest) of objects into a linear list, in place.
See also: IPFlattenTree, IPFlattenTreeProcessFF,

9.2.170 IPFlattenForrest2 (iritprs1.c:1965)

```
IPObjectStruct *IPFlattenForrest2(IPObjectStruct *PObj, int ProcessFF)
```

PObj: List of object(s) to flatten out.

ProcessFF: If TRUE, also process freeforms using IPFlattenTreeprocessFF.

Returns: One object with all objects as one linked list of geometric objects.

Description: Flattens out a list of trees' hierarchy (a forest) of objects into a linear list, in place.

9.2.171 IPFlattenInvisibleObjects (iritprs1.c:1547)

files

parser

```
int IPFlattenInvisibleObjects(int FlattenInvisib)
```

FlattenInvisib: If TRUE, list objects will be flattened out to a long linear list. If FALSE, read object will be unchanged.

Returns: Old value of flatten state.

Description: Controls the hierarchy flattening of a read object.

9.2.172 IPFlattenTree (iritprs1.c:1637)

```
IPObjectStruct *IPFlattenTree(IPObjectStruct *PObj)
```

PObj: Object(s) to flatten out, in place.

Returns: Flattened hierarchy.

Description: Flattens out a tree hierarchy of objects into a linear list, in place.
See also: IPProcessFreeForm, IPEvalFreeForms, IPFlattenTreeProcessFF, , IPFlattenForrest,

9.2.173 IPFlattenTreeProcessFF (iritprs1.c:1687)

```
IPObjectStruct *IPFlattenTreeProcessFF(IPObjectStruct *PObj)
```

PObj: Object(s) to flatten out, in place.

Returns: Flattened hierarchy.

Description: Flattens out a tree hierarchy of objects into a linear list, in place. As a side effect freeform entities are processed by IPProcessFreeForm.
See also: IPProcessFreeForm, IPEvalFreeForms, IPFlattenTree, , IPFlattenForrest,

9.2.174 IPForEachObj2 (linklist.c:1385)

```
IPObjectStruct *IPForEachObj2(IPObjectStruct *OList,
                              IPForEachObjCallBack CallBack,
                              void *Param)
```

OList: The objects list to travel. OList might be destroyed during the process, therefore it shouldn't be used again. The returned list should be used instead.

CallBack: The function to use with each object of OList.

Param: Parameter which will be given to CallBack with every object.

Returns: The new list or NULL if the list is empty.

Description: Travel on OList (using Pnext) and use the given CallBack with every object. Each given object in OList is replaced with the returned value of CallBack. If CallBack returns NULL, the object is removed from OList (in that case, it's CallBack's responsibility to free the object's memory).

9.2.175 IPForEachPoly (linklist.c:1300)

```
void IPForEachPoly(IPObjectStruct *OList, void (*CallBack) (IPPolygonStruct *))
```

OList: Pointer to the Irit objects' linked list.

CallBack: Callback function.

Returns: void

Description: Iterates on Irit object list and calls CallBack function on every polygon found, passing it a pointer to the polygon object.

9.2.176 IPForEachPoly2 (linklist.c:1437)

```
IPPolygonStruct *IPForEachPoly2(IPPolygonStruct *PlList,
                                IPForEachPolyCallBack CallBack,
                                void *Param)
```

PlList: The polygons list to travel. PlList might be destroyed during the process, therefore it shouldn't be used again. The returned list should be used instead.

CallBack: The function to use with each polygon of PlList

Param: Parameter which will be given to CallBack with every polygon.

Returns: The new list or NULL if the list is empty.

Description: Travel on PlList (using Pnext) and use the given CallBack with every polygon. Each given polygon in PlList is replaced with the returned value of CallBack. If CallBack returns NULL, the polygon is removed from PlList (in that case, it's CallBack's responsibility to free the polygon's memory).

9.2.177 IPForEachVertex (linklist.c:1336)

```
void IPForEachVertex(IPObjectStruct *OList,
                     void (*CallBack) (IPVertexStruct *))
```

OList: Pointer to Irit objects' linked list.

CallBack: Callback function.

Returns: void

Description: Iterates on Irit object list and for each vertex in every polygon calls CallBack function passing it a pointer to the vertex object.

9.2.178 IPForEachVertex2 (linklist.c:1488)

```
IPVertexStruct *IPForEachVertex2(IPVertexStruct *VList,
                                 IPForEachVertexCallBack CallBack,
                                 void *Param)
```

VList: The vertex list to travel. VList might be destroyed during the process, therefore it shouldn't be used again. The returned list should be used instead.

CallBack: The function to use with each vertex of VList

Param: Parameter which will be given to CallBack with every polygon.

Returns: The new list or NULL if the list is empty.

Description: Travel on VList (using Pnext) and use the given CallBack with every vertex. Each given vertex in VList is replaced with the returned value of CallBack. If CallBack returns NULL, the vertex is removed from VList (in that case, it's CallBack's responsibility to free the vertex's memory.

9.2.179 IPFreeForm2CubicBzr (ff_cnvrt.c:646)

```
IPObjectStruct *IPFreeForm2CubicBzr(IPFreeFormStruct *FreeForms,
                                    IPFreeformConvStateStruct *State)
```

FreeForms: Crvs/Srfs/Trimmed Srfs/Trivariates read from a file by the irit parser.

State: State of conversion to cubic Beziers.

Returns: Polyline(s) representing the trivariate and its control mesh.

Description: Converts a whole set of geometry to cubic Bezier curves, in place.

9.2.180 IPFreeForm2Polygons (ff_cnvrt.c:961)

```
IPObjectStruct *IPFreeForm2Polygons(IPFreeFormStruct *FreeForms,
                                    IPFreeformConvStateStruct *State)
```

FreeForms: Crvs/Srfs/Trimmed Srfs/Trivariates read from a file by the irit parser.

State: The tesselation state desired.

Returns: Polygon/line(s) representing the geometry and its control mesh. Returns original geometries if not convertible.

Description: Converts a whole set of geometry to polygons, in place.
See also: IPFreeForm2PolysSetCState, IPFreeForm2Polylines,

9.2.181 IPFreeForm2Polylines (ff_cnvrt.c:321)

```
IPObjectStruct *IPFreeForm2Polylines(IPFreeFormStruct *FreeForms,
                                     IPFreeformConvStateStruct *State)
```

FreeForms: Crvs/Srfs/Trimmed Srfs/Trivariates read from a file by the irit parser.

State: The desired state of the tessellation.

Returns: Polyline(s) representing the trivariate and its control mesh.

Description: Converts a whole set of geometry to polylines, in place.
See also: IPFreeForm2PolysSetCState, IPFreeForm2Polygons,

9.2.182 IPFreeForm2PolysSetCState (ff_cnvrt.c:91)

```
IPFreeformConvStateStruct IPFreeForm2PolysSetCState(const
                                          IPFreeformConvStateStruct *CState)
```

CState: New tessellation state to set. Can be NULL to query current state.

Returns: original tessellation state.

Description: Sets the global tessellation state structure of freeform 2 polys.
See also: IPFreeForm2Polylines, IPFreeForm2Polygons,

9.2.183 IPFreeObject (allocate.c:413)

allocation

```
void IPFreeObject(IPObjectStruct *O)
```

O: To free.

Returns: void

Description: Frees one Object Structure.

9.2.184 IPFreeObjectBase (allocate.c:395)

allocation

```
void IPFreeObjectBase(IPObjectStruct *O)
```

O: To free.

Returns: void

Description: Frees one Object Structure.

9.2.185 IPFreeObjectGeomData (allocate.c:150)

```
void IPFreeObjectGeomData(IPObjectStruct *PObj)
```

PObj: To free all its slots, but the name.

Returns: void

Description: Routine to free all the slots of a given object, but the name.
See also: IPFreeObjectSlots,

9.2.186 IPFreeObjectList (allocate.c:507)

allocation

```
void IPFreeObjectList(IPObjectStruct *OFirst)
```

OFirst: To free.

Returns: void

Description: Free a list of Object structures.

9.2.187 IPFreeObjectSlots (allocate.c:62)

```
void IPFreeObjectSlots(IPObjectStruct *PObj)
```

PObj: To free all its slots, but the name.

Returns: void

Description: Routine to free all the slots of a given object, but the name.
See also: IPFreeObjectGeomData,

9.2.188 IPFreePolygon (allocate.c:374)

allocation

```
void IPFreePolygon(IPPolygonStruct *P)
```

P: To free.

Returns: void

Description: Frees one Polygon Structure.

9.2.189 IPFreePolygonList (allocate.c:475)

allocation

```
void IPFreePolygonList(IPPolygonStruct *PFirst)
```

PFirst: To free.

Returns: void

Description: Free a list of Polygon structures.

9.2.190 IPFreeVertex (allocate.c:353)

allocation

```
void IPFreeVertex(IPVertexStruct *V)
```

V: To free.

Returns: void

Description: Frees one Vertex Structure.

9.2.191 IPFreeVertexList (allocate.c:443)

allocation

```
void IPFreeVertexList(IPVertexStruct *VFirst)
```

VFirst: To free.

Returns: void

Description: Free a, possibly circular, list of Vertex structures.

9.2.192 IPGenCRVObject (allocate.c:1202)

allocation

```
IPObjectStruct *IPGenCRVObject(CagdCrvStruct *Crv)
```

Crv: Curves to place in object.

Returns: A newly created curve object.

Description: Creates one curve object.

9.2.193 IPGenCTLPTObject (allocate.c:1699)

allocation

```
IPObjectStruct *IPGenCTLPTObject(CagdPointType PtType,
                                 const IrtRType *Coords)
```

PtType: Point type of created control point (E2, P3, etc.).

Coords: Coefficients of new control point. Coords[0] is always W.

Returns: A newly created control point object.

Description: Creates one control point object. Only one of CagdCoords/Coords should be specified.

9.2.194 IPGenCrvObject (allocate.c:1176)

allocation

```
IPObjectStruct *IPGenCrvObject(const char *Name,
                               CagdCrvStruct *Crv,
                               IPObjectStruct *Pnext)
```

Name: Name of polygonal object.

Crv: Curves to place in object.

Pnext: Entry into the object structure.

Returns: A newly created curve object.

Description: Creates one curve object.

9.2.195 IPGenCtlPtObject (allocate.c:1664)

allocation

```
IPObjectStruct *IPGenCtlPtObject(const char *Name,
                                 CagdPointType PtType,
                                 const IrtRType *Coords,
                                 IPObjectStruct *Pnext)
```

Name: Name of polygonal object.

PtType: Point type of created control point (E2, P3, etc.).

Coords: Coefficients of new control point. Coords[0] is always W.

Pnext: Entry into the object structure.

Returns: A newly created control point object.

Description: Creates one control point object.

9.2.196 IPGenINSTNCObject (allocate.c:1640)

allocation

```
IPObjectStruct *IPGenINSTNCObject(const char *InstncName,
                                  const IrtHmgnMatType *Mat)
```

InstncName: Object name of original.

Mat: Instance matrix, or NULL if none.

Returns: A newly created instance object.

Description: Creates one instance object.

9.2.197 IPGenInstncObject (allocate.c:1610)

allocation

```
IPObjectStruct *IPGenInstncObject(const char *Name,
                                  const char *InstncName,
                                  const IrtHmgnMatType *Mat,
                                  IPObjectStruct *Pnext)
```

Name: Name of polygonal object.

InstncName: Object name of original.

Mat: Instance matrix, or NULL if none.

Pnext: Entry into the object structure.

Returns: A newly created instance object.

Description: Creates one instance object.

9.2.198 IPGenLISTObject (allocate.c:1959)

allocation

```
IPObjectStruct *IPGenLISTObject(IPObjectStruct *First)
```

First: First element in list, if any.

Returns: A newly created list object.

Description: Creates one list object.

9.2.199 IPGenListObject (allocate.c:1934)

allocation

```
IPObjectStruct *IPGenListObject(const char *Name,
                                IPObjectStruct *First,
                                IPObjectStruct *Pnext)
```

Name: Name of list object.

First: First element in list, if any.

Pnext: Entry into the object structure.

Returns: A newly created list object.

Description: Creates one list object.

9.2.200 IPGenMATObject (allocate.c:2063)

allocation

```
IPObjectStruct *IPGenMATObject(IrtHmgnMatType Mat)
```

Mat: Matrix to initialize with.

Returns: A newly created matrix object.

Description: Creates one matrix object.

9.2.201 IPGenMODELObject (allocate.c:1442)

allocation

```
IPObjectStruct *IPGenMODELObject(MdlModelStruct *Model)
```

Model: A model object.

Returns: A newly created triangular surface object.

Description: Creates one model object.

9.2.202 IPGenMULTIVARObject (allocate.c:1586)

allocation

```
IPObjectStruct *IPGenMULTIVARObject(MvarMVStruct *MultiVar)
```

MultiVar: A multivariate object.

Returns: A newly created triangular surface object.

Description: Creates one multivariate object.

9.2.203 IPGenMatObject (allocate.c:2036)

allocation

```
IPObjectStruct *IPGenMatObject(const char *Name,
                               IrtHmgnMatType Mat,
                               IPObjectStruct *Pnext)
```

Name: Name of polygonal object.

Mat: Matrix to initialize with.

Pnext: Entry into the object structure.

Returns: A newly created matrix object.

Description: Creates one matrix object.

9.2.204 IPGenModelObject (allocate.c:1416)

allocation

```
IPObjectStruct *IPGenModelObject(const char *Name,
                                 MdlModelStruct *Model,
                                 IPObjectStruct *Pnext)
```

Name: Name of polygonal object.

Model: A model object.

Pnext: Entry into the object structure.

Returns: A newly created triangular surface object.

Description: Creates one model object.

9.2.205 IPGenMultiVarObject (allocate.c:1560)

allocation

```
IPObjectStruct *IPGenMultiVarObject(const char *Name,
                                    MvarMVStruct *MultiVar,
                                    IPObjectStruct *Pnext)
```

Name: Name of polygonal object.

MultiVar: A multivariate object.

Pnext: Entry into the object structure.

Returns: A newly created triangular surface object.

Description: Creates one surface object.

9.2.206 IPGenNUMObject (allocate.c:1745)

allocation

```
IPObjectStruct *IPGenNUMObject(const IrtRType *R)
```

R: Numeric value to place in object.

Returns: A newly created numeric object.

Description: Creates one numeric object.

9.2.207 IPGenNUMValObject (allocate.c:1763)

allocation

```
IPObjectStruct *IPGenNUMValObject(IrtRType R)
```

R: Numeric value to place in object.

Returns: A newly created numeric object.

Description: Creates one numeric object.

9.2.208 IPGenNumObject (allocate.c:1721)

allocation

```
IPObjectStruct *IPGenNumObject(const char *Name,
                               const IrtRType *R,
                               IPObjectStruct *Pnext)
```

Name: Name of polygonal object.

R: Numeric value to place in object.

Pnext: Entry into the object structure.

Returns: A newly created numeric object.

Description: Creates one numeric object.

9.2.209 IPGenPLANEObject (allocate.c:2014)

allocation

```
IPObjectStruct *IPGenPLANEObject(const IrtRType *Plane0,
                                 const IrtRType *Plane1,
                                 const IrtRType *Plane2,
                                 const IrtRType *Plane3)
```

Plane0, Plane1, Plane2, Plane3: Coefficients of point.

Returns: A newly created plane object.

Description: Creates one plane object.

9.2.210 IPGenPOINTLISTObject (allocate.c:1154)

allocation

```
IPObjectStruct *IPGenPOINTLISTObject(IPPolygonStruct *Pl)
```

Pl: Pointlist(s) to place in object.

Returns: A newly created polygonal object.

Description: Creates one pointlist object.

9.2.211 IPGenPOLYLINEObject (allocate.c:1107)

allocation

```
IPObjectStruct *IPGenPOLYLINEObject(IPPolygonStruct *Pl)
```

Pl: Polyline(s) to place in object.

Returns: A newly created polygonal object.

Description: Creates one polyline object.

9.2.212 IPGenPOLYObject (allocate.c:1060)

allocation

```
IPObjectStruct *IPGenPOLYObject(IPPolygonStruct *Pl)
```

Pl: Polygon(s) to place in object.

Returns: A newly created polygonal object.

Description: Creates one polygonal object.

9.2.213 IPGenPTObject (allocate.c:1815)

allocation

```
IPObjectStruct *IPGenPTObject(const IrtRType *Pt0,
                              const IrtRType *Pt1,
                              const IrtRType *Pt2)
```

Pt0, Pt1, Pt2: Coefficients of point.

Returns: A newly created point object.

Description: Creates one point object.

9.2.214 IPGenPlaneObject (allocate.c:1984)

allocation

```
IPObjectStruct *IPGenPlaneObject(const char *Name,
                                 const IrtRType *Plane0,
                                 const IrtRType *Plane1,
                                 const IrtRType *Plane2,
                                 const IrtRType *Plane3,
                                 IPObjectStruct *Pnext)
```

Name: Name of polygonal object.

Plane0, Plane1, Plane2, Plane3: Coefficients of point.

Pnext: Entry into the object structure.

Returns: A newly created plane object.

Description: Creates one plane object.

9.2.215 IPGenPointListObject (allocate.c:1129)

allocation

```
IPObjectStruct *IPGenPointListObject(const char *Name,
                                     IPPolygonStruct *Pl,
                                     IPObjectStruct *Pnext)
```

Name: Name of pointlist object.

Pl: Pointlist(s) to place in object.

Pnext: Entry into the object structure.

Returns: A newly created pointlist object.

Description: Creates one pointlist object.

9.2.216 IPGenPolyObject (allocate.c:1035)

allocation

```
IPObjectStruct *IPGenPolyObject(const char *Name,
                                IPPolygonStruct *Pl,
                                IPObjectStruct *Pnext)
```

Name: Name of polygonal object.

Pl: Polygon(s) to place in object.

Pnext: Entry into the object structure.

Returns: A newly created polygonal object.

Description: Creates one polygonal object.

9.2.217 IPGenPolylineObject (allocate.c:1082)

allocation

```
IPObjectStruct *IPGenPolylineObject(const char *Name,
                                    IPPolygonStruct *Pl,
                                    IPObjectStruct *Pnext)
```

Name: Name of polyline object.

Pl: Polyline(s) to place in object.

Pnext: Entry into the object structure.

Returns: A newly created polyline object.

Description: Creates one polyline object.

9.2.218 IPGenPtObject (allocate.c:1787)

allocation

```
IPObjectStruct *IPGenPtObject(const char *Name,
                              const IrtRType *Pt0,
                              const IrtRType *Pt1,
                              const IrtRType *Pt2,
                              IPObjectStruct *Pnext)
```

Name: Name of polygonal object.

Pt0, Pt1, Pt2: Coefficients of point.

Pnext: Entry into the object structure.

Returns: A newly created point object.

Description: Creates one point object.

9.2.219 IPGenRectangle (ip_cnvrt.c:891)

```
IPPolygonStruct *IPGenRectangle(const CagdSrf2PlsInfoStrct *TessInfo,
                                const CagdRType *Pt1,
                                const CagdRType *Pt2,
                                const CagdRType *Pt3,
                                const CagdRType *Pt4,
                                const CagdRType *Nl1,
                                const CagdRType *Nl2,
                                const CagdRType *Nl3,
                                const CagdRType *Nl4,
                                const CagdRType *UV1,
                                const CagdRType *UV2,
                                const CagdRType *UV3,
                                const CagdRType *UV4,
                                CagdBType *GenPoly)
```

TessInfo: All auxiliary information/state to tessellate.

Pt1, Pt2, Pt3, Pt4: Euclidean locations of vertices.

Nl1, Nl2, Nl3, Nl4: Optional Normals of vertices (if ComputeNormals).

UV1, UV2, UV3, UV4: Optional UV parametric location of vertices (if ComputeUV).

GenPoly: Returns TRUE if a polygon was generated, FALSE otherwise.

Returns: The created polygon. Note if singular, might return a triangle.

Description: Call back routine to create one rectangular polygon, given its vertices, and, optionally, normals and uv coordinates.

See also: IPGenTriangle,

9.2.220 IPGenSRFObject (allocate.c:1250)

allocation

```
IPObjectStruct *IPGenSRFObject(CagdSrfStruct *Srf)
```

Srf: Surfaces to place in object.

Returns: A newly created surface object.

Description: Creates one surface object.

9.2.221 IPGenSTRObject (allocate.c:1912)

allocation

```
IPObjectStruct *IPGenSTRObject(const char *Str)
```

Str: The string.

Returns: A newly created strtor object.

Description: Creates one string object.

9.2.222 IPGenSrfObject (allocate.c:1224)

allocation

```
IPObjectStruct *IPGenSrfObject(const char *Name,
                               CagdSrfStruct *Srf,
                               IPObjectStruct *Pnext)
```

Name: Name of polygonal object.

Srf: Surfaces to place in object.

Pnext: Entry into the object structure.

Returns: A newly created surface object.

Description: Creates one surface object.

9.2.223 IPGenStrObject (allocate.c:1889)

allocation

```
IPObjectStruct *IPGenStrObject(const char *Name,
                               const char *Str,
                               IPObjectStruct *Pnext)
```

Name: Name of string object.

Str: The string.

Pnext: Entry into the object structure.

Returns: A newly created strtor object.

Description: Creates one string object.

9.2.224 IPGenTRIMSRFObject (allocate.c:1298)

allocation

```
IPObjectStruct *IPGenTRIMSRFObject(TrimSrfStruct *TrimSrf)
```

TrimSrf: Trimmed surfaces to place in object.

Returns: A newly created trimmed surface object.

Description: Creates one trimmed surface object.

9.2.225 IPGenTRISRFObject (allocate.c:1394)

allocation

```
IPObjectStruct *IPGenTRISRFObject(TrngTriangSrfStruct *TriSrf)
```

TriSrf: Triangular Surfaces to place in object.

Returns: A newly created triangular surface object.

Description: Creates one triangular surface object.

9.2.226 IPGenTRIVARObject (allocate.c:1346)

allocation

```
IPObjectStruct *IPGenTRIVARObject(TrivTVStruct *Triv)
```

Triv: Trivariates to place in object.

Returns: A newly created trivariate object.

Description: Creates one trivariate object.

9.2.227 IPGenTriSrfObject (allocate.c:1368)

allocation

```
IPObjectStruct *IPGenTriSrfObject(const char *Name,
                                  TrngTriangSrfStruct *TriSrf,
                                  IPObjectStruct *Pnext)
```

Name: Name of polygonal object.
TriSrf: Triangular Surfaces to place in object.
Pnext: Entry into the object structure.
Returns: A newly created triangular surface object.
Description: Creates one triangular surface object.

9.2.228 IPGenTriangle (ip_cnvrt.c:755)

```
IPPolygonStruct *IPGenTriangle(const CagdSrf2PlsInfoStrct *TessInfo,
                               const CagdRType *Pt1,
                               const CagdRType *Pt2,
                               const CagdRType *Pt3,
                               const CagdRType *Nl1,
                               const CagdRType *Nl2,
                               const CagdRType *Nl3,
                               const CagdRType *UV1,
                               const CagdRType *UV2,
                               const CagdRType *UV3,
                               CagdBType *GenPoly)
```

TessInfo: All auxiliary information/state to tessellate.
Pt1, Pt2, Pt3: Euclidean locations of vertices.
Nl1, Nl2, Nl3: Optional Normals of vertices (if ComputeNormals).
UV1, UV2, UV3: Optional UV parametric location of vertices (if ComputeUV).
GenPoly: Returns TRUE if a polygon was generated, FALSE otherwise.
Returns: The created triangle.
Description: Call back routine to create one triangular polygon, given its vertices, and, optionally, normals and uv coordinates. Places the constructed polygon in a global polygonal list.
See also: IPGenRectangle,

9.2.229 IPGenTrimSrfObject (allocate.c:1272)

allocation

```
IPObjectStruct *IPGenTrimSrfObject(const char *Name,
                                   TrimSrfStruct *TrimSrf,
                                   IPObjectStruct *Pnext)
```

Name: Name of polygonal object.
TrimSrf: Trimmed surfaces to place in object.
Pnext: Entry into the object structure.
Returns: A newly created trimmed surface object.
Description: Creates one trimmed surface object.

9.2.230 IPGenTrivarObject (allocate.c:1320)

allocation

```
IPObjectStruct *IPGenTrivarObject(const char *Name,
                                  TrivTVStruct *Triv,
                                  IPObjectStruct *Pnext)
```

Name: Name of polygonal object.
Triv: Trivariates to place in object.
Pnext: Entry into the object structure.
Returns: A newly created trivariate object.
Description: Creates one trivariate object.

9.2.231 IPGenVECObject (allocate.c:1867)

allocation

```
IPObjectStruct *IPGenVECObject(const IrtRType *Vec0,
                               const IrtRType *Vec1,
                               const IrtRType *Vec2)
```

Vec0, Vec1, Vec2: Coefficients of vector.

Returns: A newly created vector object.

Description: Creates one vector object.

9.2.232 IPGenVMODELObject (allocate.c:1490)

allocation

```
IPObjectStruct *IPGenVMODELObject(VMdlVModelStruct *VModel)
```

VModel: A volumetric model object.

Returns: A newly created triangular surface object.

Description: Creates one vmodel object.

9.2.233 IPGenVModelObject (allocate.c:1464)

allocation

```
IPObjectStruct *IPGenVModelObject(const char *Name,
                                  VMdlVModelStruct *VModel,
                                  IPObjectStruct *Pnext)
```

Name: Name of polygonal object.

VModel: A volumetric model object.

Pnext: Entry into the object structure.

Returns: A newly created triangular surface object.

Description: Creates one vmodel object.

9.2.234 IPGenVXLVMDLObject (allocate.c:1538)

allocation

```
IPObjectStruct *IPGenVXLVMDLObject(VMdlVoxelVModelStruct *VxlVModel)
```

VxlVModel: A volumetric model object.

Returns: A newly created triangular surface object.

Description: Creates one voxel vmodel object.

9.2.235 IPGenVecObject (allocate.c:1839)

allocation

```
IPObjectStruct *IPGenVecObject(const char *Name,
                               const IrtRType *Vec0,
                               const IrtRType *Vec1,
                               const IrtRType *Vec2,
                               IPObjectStruct *Pnext)
```

Name: Name of polygonal object.

Vec0, Vec1, Vec2: Coefficients of vector.

Pnext: Entry into the object structure.

Returns: A newly created vector object.

Description: Creates one vector object.

9.2.236 IPGenVxlVMdlObject (allocate.c:1512)

allocation

```
IPObjectStruct *IPGenVxlVMdlObject(const char *Name,
                                   VMdlVoxelVModelStruct *VxlVModel,
                                   IPObjectStruct *Pnext)
```

Name: Name of polygonal object.

VxlVModel: A volumetric model object.

Pnext: Entry into the object structure.

Returns: A newly created triangular surface object.

Description: Creates one voxel vmodel object.

9.2.237 IPGetBinObject (iritprsb.c:312)

files

parser

```
IPObjectStruct *IPGetBinObject(int Handler)
```

Handler: A handler to the open stream.

Returns: Read object.

Description: Routine to read one object from a given binary file, directly. Objects may be recursively defined (as lists), in which case all are read in this single call.

9.2.238 IPGetDataFileType (iritprs1.c:113)

```
const char *IPGetDataFileType(const char *FileName, int *Compressed)
```

FileName: Input file name to isolate its file type.

Compressed: TRUE if the file is compressed (.Z or .gz).

Returns: Detected file type, statically allocated, or "???" if not found.

Description: Isolates the file type associated with this file name.

9.2.239 IPGetDataFiles (iritprs1.c:851)

files

parser

```
IPObjectStruct *IPGetDataFiles(char const * const *DataFileNames,
                               int NumOfDataFiles,
                               int Messages,
                               int MoreMessages)
```

DataFileNames: Array of strings (file names) to process.

NumOfDataFiles: Number of elements in DataFileNames.

Messages: Do we want error messages?

MoreMessages: Do we want informative messages?

Returns: Objects read from all files, NULL if error.

Description: Reads data from a set of files specified by file names. Messages and MoreMessages controls the level of printout. Freeform geometry read in is handed out to a call back function named IPProcessFreeForm before it is returned from this routine. This is done so applications that do not want to deal with freeform shapes will be able to provide a call back that processes the freeform shapes into other geometry such as polygons.

See also: IPProcessFreeForm, IPGetDataWFiles,

9.2.240 IPGetDataFromFilehandles (iritprs1.c:982)

filehandles

parser

```
IPObjectStruct *IPGetDataFromFilehandles(FILE **Files,
                                         int NumOfFiles,
                                         char **Extensions,
                                         int Messages,
                                         int MoreMessages)
```

Files: Array of file handles to process.

NumOfFiles: Number of elements in Files.

Extensions: Array of file name extensions for the files; used to determine file formats.

Messages: Do we want error messages?

MoreMessages: Do we want informative messages?

Returns: Objects read from all filehandles.

Description: Convenience function for reading input from file handles instead of indirectly from filenames; caller must supply an array of filename extensions (as char * strings) for ascertaining the input files' formats.
See also: IPGetDataFiles, IPGetDataFromFilehandles2,

9.2.241 IPGetDataFromFilehandles2 (iritprs1.c:1033)

filehandles

parser

```
IPObjectStruct *IPGetDataFromFilehandles2(FILE **Files,
                                          int NumOfFiles,
                                          IPStreamFormatType *Formats,
                                          int *IsBinaryIndicators,
                                          int Messages,
                                          int MoreMessages)
```

Files: Array of file handles to process.

NumOfFiles: Number of elements in Files.

Formats: Array of file formats for the files

IsBinaryIndicators: Array of 'IsBinary' indications for the files

Messages: Do we want error messages?

MoreMessages: Do we want informative messages?

Returns: Objects read from all filehandles.

Description: Convenience function for reading input from file handles instead of indirectly from filenames; caller must supply an array of file formats and 'IsBinary' indicators.
See also: IPGetDataFiles, IPGetDataFromFilehandles,

9.2.242 IPGetDataWFiles (iritprs1.c:915)

files

parser

```
IPObjectStruct *IPGetDataWFiles(wchar_t const * const *DataFileNames,
                                int NumOfDataFiles,
                                int Messages,
                                int MoreMessages)
```

DataFileNames: Array of strings (file names) to process.

NumOfDataFiles: Number of elements in DataFileNames.

Messages: Do we want error messages?

MoreMessages: Do we want informative messages?

Returns: Objects read from all files, NULL if error.

Description: Reads data from a set of files specified by file names. Messages and MoreMessages controls the level of printout. Freeform geometry read in is handed out to a call back function named IPProcessFreeForm before it is returned from this routine. This is done so applications that do not want to deal with freeform shapes will be able to provide a call back that processes the freeform shapes into other geometry such as polygons.
See also: IPProcessFreeForm, IPGetDataFiles,

9.2.243 IPGetLastObj (linklist.c:435)

linked lists

last element

```
IPObjectStruct *IPGetLastObj(IPObjectStruct *OList)
```

OList: A list of objects.

Returns: Last object in list OList.

Description: Returns a pointer to last object of a list.

9.2.244 IPGetLastPoly (linklist.c:361)

linked lists

last element

```
IPPolygonStruct *IPGetLastPoly(IPPolygonStruct *PList)
```

PList: A list of polygons.

Returns: Last polygon in list PList.

Description: Returns a pointer to last polygon/line of a list.

9.2.245 IPGetLastVrtx (linklist.c:283)

linked lists

last element

```
IPVertexStruct *IPGetLastVrtx(IPVertexStruct *VList)
```

VList: A list of vertices

Returns: Last vertex in VList.

Description: Returns a pointer to last vertex of a list.

9.2.246 IPGetMatrixFile (iritprs1.c:3413)

```
int IPGetMatrixFile(const char *File,
                    IrtHmgnMatType ViewMat,
                    IrtHmgnMatType ProjMat,
                    int *HasProjMat)
```

File: File to read the matrix file from.

ViewMat, ProjMat: Matrices to get.

HasProjMat: TRUE if has a perspective matrix, FALSE otherwise.

Returns: TRUE if (at least) VIEW_MAT was found, FALSE otherwise.

Description: Gets an IRIT matrix file.
See also: IPPutMatrixFile,

9.2.247 IPGetObjectByName (linklist.c:1534)

```
IPObjectStruct *IPGetObjectByName(const char *Name,
                                  IPObjectStruct *PObjList,
                                  int TopLevel)
```

Name: Of object to find and return a reference of.

PObjList: List of objects to scan.

TopLevel: if TRUE, scan only the top level list.

Returns: A reference to found object, NULL otherwise.

Description: Searchs for an onbject in given PObjList, named Name.

9.2.248 IPGetObjectTypeAsString (iritprs2.c:1047)

```
const char *IPGetObjectTypeAsString(const IPObjectStruct *PObj)
```

PObj: Object to get a string description on.

Returns: A string describing PObj.

Description: Gets a string description of the object type.
See also: IPGetObjectTypeAsString2,

9.2.249 IPGetObjectTypeAsString2 (iritprs2.c:1071)

```
const char *IPGetObjectTypeAsString2(IPObjStructType ObjType)
```

ObjType: Object type to get a string description on.

Returns: A string describing ObjType.

Description: Gets a string description of the object type.
See also: IPGetObjectTypeAsString,

9.2.250 IPGetObjects (iritprs1.c:1184)

files

parser

```
IPObjectStruct *IPGetObjects(int Handler)
```

Handler: A handler to the open stream.

Returns: Read object, or NULL if failed.

Description: Routine to read the data from a given file. Returns NULL if EOF was reached or error occurred.
See also: IPSetPolyListCirc, IPSetFlattenObjects, IPSetReadOneObject, PTransformInstances.,

9.2.251 IPGetPrevObj (linklist.c:455)

previous element

linked lists

```
IPObjectStruct *IPGetPrevObj(IPObjectStruct *OList, IPObjectStruct *O)
```

OList: A list of objects.

O: For which the previous object in OList is pursuit.

Returns: Previous object to O in OList if found, NULL otherwise.

Description: Returns a pointer to previous object in OList to O.

9.2.252 IPGetPrevPoly (linklist.c:382)

previous element

linked lists

```
IPPolygonStruct *IPGetPrevPoly(IPPolygonStruct *PList,
                               IPPolygonStruct *P)
```

PList: A list of polygons.

P: For which the previous polygon in PList is pursuit.

Returns: Previous polygon to P in PList if found, NULL otherwise.

Description: Returns a pointer to previous polygon in PList to P.

9.2.253 IPGetPrevVrtx (linklist.c:303)

previous element

linked lists

```
IPVertexStruct *IPGetPrevVrtx(IPVertexStruct *VList, IPVertexStruct *V)
```

VList: A list of vertices.

V: For which the previous vertex in VList is pursuit.

Returns: Previous vertex to V in VList if found, NULL otherwise.

Description: Returns a pointer to previous vertex in VList to V.

9.2.254 IPGetPrspMat (prsrgeom.c:73)

```
IrtHmgnMatType *IPGetPrspMat(int *WasPrspMat)
```

WasPrspMat: TRUE if parser until now detected a perspective matrix.

Returns: A reference to the matrix.

Description: Fetches the current perspective matrix.
See also: IPGetViewMat,

9.2.255 IPGetRealNumber (iritprs1.c:3453)

files

```
int IPGetRealNumber(const char *StrNum, IrtRType *RealNum)
```

StrNum: A string representing a real number.

RealNum: The fetched real number.

Returns: TRUE if number was parsed correctly, FALSE otherwise.

Description: Fetchs a real number fro ma given string.

9.2.256 IPGetViewMat (prsrgeom.c:50)

```
IrtHmgnMatType *IPGetViewMat(int *WasViewMat)
```

WasViewMat: TRUE if parser until now detected a view matrix.

Returns: A reference to the matrix.

Description: Fetches the current view matrix.
See also: IPGetPrspMat,

9.2.257 IPHasError (prsr_err.c:191)

error handling

```
int IPHasError(const char **ErrorDesc)
```

ErrorDesc: Where to place the error description if was one.

Returns: TRUE if there was an error, FALSE otherwise.

Description: Returns TRUE if an error was signaled and set ErrorDesc to its value.

9.2.258 IPHierarchyObjToLinkedList (linklist.c:962)

```
void *IPHierarchyObjToLinkedList(const IPObjectStruct *HObj,
                                 IPObjStructType ObjType)
```

HObj: Hierarchy object to process.

ObjType: Type of objects to fetch.

Returns: Linked list of (copies) of input data of type ObjType (i.e. curves), NULL if error.

Description: Copy and collect all objects in HObj that are of type ObjType.
See also: IPObjLnkListToListObject, IPLnkListToListObject, IPListObjToLinkedList2, , IPListObjToLinkedList, IPHierarchyObjToVector,

9.2.259 IPHierarchyObjToVector (linklist.c:1098)

```
int IPHierarchyObjToVector(const IPObjectStruct *HObj,
                           IPObjStructType ObjType,
                           void ***Vec,
                           int ElmntSize)
```

HObj: Hierarchy object to process.

ObjType: Type of objects to fetch.

Vec: Vector of referenced types ObjType. Will be allocated/resized dynamically and on the fly as needed.

ElmntSize: One element size of ObjType element in Vec.

Returns: Number of elements referenced in resulting vector. If Zero, *Vec will be set to NULL.

Description: Referencing (not copying) all objects in HObj that are of type ObjType.
See also: IPObjLnkListToListObject, IPLnkListToListObject, IPListObjToLinkedList2, , IPListObjToLinkedList, IPHierarchyObjToLinkedList,

9.2.260 IPIgesLoadFile (igs_irit.c:285)

```
IPObjectStruct *IPIgesLoadFile(const char *IgesFileName,
                               const IPIgesLoadDfltFileParamsStruct *Params)
```

IgesFileName: Name of IGES file.

Params: Parameters to control the way the IGES file is loaded. ClipTrimmedSrf- TRUE to clip trimming surface to the size of the trimming curves. DumpAll - TRUE to dump all entities, including auxiliary entities used by other entities. IgnoreGrouping - TRUE to ignore any instance grouping. ApproxCoversion - TRUE to make an approximated conversion (of trimming curves with numerous samples. InverseProjCrvOnSrfs - TRUE to back project E3 trimming curves to the UV surface space. Disable for debugging only. Messages - 1 for error messages, 2 to include warning messages, 3 to include informative messages. 4 to include dump of IRIT objects.

Returns: Read IGES DATA or NULL if error.

Description: Read IGES 5.0 files into IRIT data.
See also: IPIgesSaveFile, IPIgesLoadFileSetDefaultParameters,

9.2.261 IPIgesLoadFileSetDefaultParameters (iritprs1.c:1310)

```
void IPIgesLoadFileSetDefaultParameters(int ClipTrimmedSrf,
                                        int DumpAll,
                                        int IgnoreGrouping,
                                        int ApproxCoversion,
                                        int InverseProjCrvOnSrfs,
                                        int Messages)
```

ClipTrimmedSrf: TRUE to clip trimming surface to their trimming curves.

DumpAll: TRUE to dump all entities, including auxiliary entities used by other entities.

IgnoreGrouping: TRUE to ignore any instance grouping.

ApproxCoversion: TRUE to support approximated geometry conversion (i.e. trimming curves).

InverseProjCrvOnSrfs: TRUE to back project Euclidean trimming curves on the UV domain of the surfaces. Slow but typically necessary. Disable for debugging mostly.

Messages: 1 for error messages, 2 to include warning messages, 3 to include informative messages. 4 to include dump of IRIT objects.

Returns: void

Description: Sets default loading parameters for Iges files.
See also: IPGetObjects, IPIgesLoadFile,

9.2.262 IPIgesSaveEucTrimCrvs (irit_igs.c:315)

```
int IPIgesSaveEucTrimCrvs(int SaveEucTrimCrvs)
```

SaveEucTrimCrvs: TRUE to save Euclidean trimming curves as well.

Returns: Old state of saving Euclidean trimming curves.

Description: Sets the state of the IGES file save.
See also: IPIgesSaveFile,

9.2.263 IPIgesSaveFile (irit_igs.c:173)

```
int IPIgesSaveFile(const IPObjectStruct *PObject,
                   IrtHmgnMatType CrntViewMat,
                   const char *IgesFileName,
                   int Messages)
```

PObject: IritObject to dump as IGES 5.0 file.

CrntViewMat: The current viewing matrix to apply to the object.

IgesFileName: Name of IGES file.

Messages: TRUE for warning messages.

Returns: TRUE if successful, FALSE otherwise.

Description: Dumps IRIT object as IGES 5.0 file.
See also: IPIgesLoadFile, IPIgesSaveEucTrimCrvs,

9.2.264 IPInputUnGetC (iritprs1.c:2548)

files

parser

```
void IPInputUnGetC(int Handler, char c)
```

Handler: A handler to the open stream.

c: Character to unget.

Returns: void

Description: Routine to unget a single character from input stream.

9.2.265 IPIritPlgns2CagdPlgns (ip_cnvrt.c:125)

```
CagdPolygonStruct *IPIritPlgns2CagdPlgns(IPPolygonStruct *Polys)
```

Polys: Polygons in irit form to be converted in cagd_lib form, in place.

Returns: Same polylines in CAGD format.

Description: Routine to convert a irit polygons to cagd polygons. Old irit polygons are freed!
See also: IPCagdPlgns2IritPlgns, CagdCnvrtPolyline2LinBspCrv,

9.2.266 IPLinkedListToObjList (linklist.c:672)

```
IPObjectStruct *IPLinkedListToObjList(const IPObjectStruct *LnkList)
```

LnkList: Linked list to process.

Returns: A list object holding the linked list as separated individual objects.

Description: Convert a linked list of similar objects to a list object.
See also: IPObjLnkListToListObject, IPLnkListToListObject, , IPHierarchyObjToLinkedList,

9.2.267 IPListObjToLinkedList (linklist.c:833)

```
void *IPListObjToLinkedList(const IPObjectStruct *LObjs)
```

LObjs: List object to process.

Returns: Linked list of (copies) of input data, NULL if error.

Description: Convert a list object of similar objects to a linked list of these objs.
See also: IPObjLnkListToListObject, IPLnkListToListObject, IPListObjToLinkedList2, IPHierarchyObjToLinkedList,

9.2.268 IPListObjToLinkedList2 (linklist.c:765)

```
IPObjectStruct *IPListObjToLinkedList2(const IPObjectStruct *LObjs)
```

LObjs: List object to process.

Returns: Linked list of (copies) of input data, NULL if error.

Description: Convert a list object of similar objects to a linked list of these objs.
See also: IPObjLnkListToListObject, IPLnkListToListObject, IPListObjToLinkedList, , IPHierarchyObjToLinkedList,

9.2.269 IPListObjectAppend (allocate.c:682)


```
void IPListObjectAppend(IPObjectStruct *PObj, IPObjectStruct *PObjItem)
```

PObj: A list object to insert PObjItem into.

PObjItem: Element to insert last into the list object PObj.

Returns: void

Description: Insert an object PObjItem as last into a list object, PObj.
See also: IPListObjectLength, IPListObjectFind, IPListObjectInsert, , IPListObjectDelete, IPListObjectGet, IPListObjectAppendList,

9.2.270 IPListObjectAppendList (allocate.c:711)


```
void IPListObjectAppendList(IPObjectStruct *PObj,
                            IPObjectStruct *PObjNew)
```

PObj: A list object to insert the objects in PObjItem into, in place.

PObjNew: A list object to move its objects into PObj. In place (PObjNew will be freed by this function once operation is complete).

Returns: void

Description: Insert the objects in list object PObjNewList as last into a list object, PObj.
See also: IPListObjectLength, IPListObjectFind, IPListObjectInsert, , IPListObjectDelete, IPListObjectGet, IPListObjectAppend,

9.2.271 IPListObjectDelete (allocate.c:745)

lists

insert

delete

```
IPObjectStruct *IPListObjectDelete(IPObjectStruct *PObj,
                                   int Index,
                                   int FreeItem)
```

PObj: A list of objects to delete item Index.

Index: Index where item should be deleted.

FreeItem: If TRUE, Item is also freed, if FALSE only deleted from list.

Returns: The removed object if not FreeItem, NULL otherwise.

Description: Delete an object at index Index from list of objects, PObj.
See also: IPListObjectLength, IPListObjectFind, IPListObjectInsert, , IPListObjectAppend, IPListObjectGet, IPListObjectDelete2,

9.2.272 IPListObjectDelete2 (allocate.c:796)

lists

insert

delete

```
void IPListObjectDelete2(IPObjectStruct *PObj,
                         IPObjectStruct *PObjToDel,
                         int FreeItem)
```

PObj: A list object to delete PObjToDel from.

PObjToDel: Object to delete from list object PObj.

FreeItem: If TRUE, Item is also freed, if FALSE only deleted from list.

Returns: void

Description: Delete an object at index Index from list of objects, PObj.
See also: IPListObjectLength, IPListObjectFind, IPListObjectInsert, , IPListObjectAppend, IPListObjectGet, IPListObjectDelete,

9.2.273 IPListObjectFind (allocate.c:571)

linked lists

find

```
int IPListObjectFind(const IPObjectStruct *PObjList,
                     const IPObjectStruct *PObj)
```

PObjList: To search for PObj in.

PObj: The element to search in PObjList.

Returns: TRUE if PObj was found in PObjList, FALSE otherwise.

Description: Returns TRUE if PObj is an object in list PObjList or in a sublist of PObjList, recursively.
See also: IPListObjectLength, IPListObjectInsert, IPListObjectAppend, , IPListObjectDelete, IPListObject-Get,

9.2.274 IPListObjectGet (allocate.c:829)

lists

find

```
IPObjectStruct *IPListObjectGet(const IPObjectStruct *PObj, int Index)
```

PObj: A list object to extract one object from.

Index: Index of object to extract from PObj.

Returns: Index object in list PObj, or NULL if no such thing.

Description: Returns the object number Index in list of PObjList object.
See also: IPListObjectLength, IPListObjectFind, IPListObjectInsert, , IPListObjectAppend, IPListObject-Delete,

9.2.275 IPListObjectInsert (allocate.c:612)

lists

insert

```
void IPListObjectInsert(IPObjectStruct *PObj,
                        int Index,
                        IPObjectStruct *PObjItem)
```

PObj: A list of objects to insert PObjItem into.

Index: Index where PObjItem should enter PObj.

PObjItem: Element to insert into the list PObj.

Returns: void

Description: Insert an object PObjItem at index Index into a list of objects, PObj. Overwriting existing item if was any.
See also: IPListObjectInsert2, IPListObjectLength, IPListObjectFind, , IPListObjectAppend, IPListObjectDelete, IPListObjectGet,

9.2.276 IPListObjectInsert2 (allocate.c:649)

lists

insert

```
void IPListObjectInsert2(IPObjectStruct *PObj,
                         int Index,
                         IPObjectStruct *PObjItem)
```

PObj: A list of objects to insert PObjItem into.

Index: Index where PObjItem should enter PObj.

PObjItem: Element to insert into the list PObj.

Returns: void

Description: Insert an object PObjItem at index Index into a list of objects, PObj. Same as IPListObjectInsert, but make sure to first make space in the list at index Index, by moving all items above or equal to index Index one position up, expanding the length of the list by one.
See also: IPListObjectInsert, IPListObjectLength, IPListObjectFind, , IPListObjectAppend, IPListObjectDelete, IPListObjectGet,

9.2.277 IPListObjectLength (allocate.c:535)

linked lists

length

```
int IPListObjectLength(const IPObjectStruct *PObj)
```

PObj: A list of objects to find its length.

Returns: Resulting length of list PObj.

Description: Returns the length of a list, given a list of objects.
See also: IPListObjectFind, IPListObjectInsert, IPListObjectAppend, , IPListObjectDelete, IPListObjectGet,

9.2.278 IPLnkListToListObject (linklist.c:599)

```
IPObjectStruct *IPLnkListToListObject(VoidPtr LnkList,
                                      IPObjStructType ObjType)
```

LnkList: Linked list to convert. We assume the Pnext pointer is the first entry in the structure.

ObjType: Type of objects we have in the linked list.

Returns: A list object holding all items in linked list.

Description: Converts a linked list into one list object, in place.
See also: IPAppendListObjects, IPObjLnkListToListObject, IPLinkedListToObjList, , IPHierarchyObjToLinkedList,

9.2.279 IPMapObjectInPlace (ff_cnvrt.c:1658)

```
void IPMapObjectInPlace(IPObjectStruct *PObj, IrtHmgnMatType Mat, void *AuxData)
```

PObj: Object to map, in place, according to Mat.

Mat: Transformation matrix.

AuxData: Optional auxiliary data - not used.

Returns: void

Description: Maps the object according to the given matrix, in place.

9.2.280 IPMshCnvrt2Bezier (irit_msh.c:3352)

```
void IPMshCnvrt2Bezier(IPObjectStruct *PObj,
                       IrtHmgnMatType Mat,
                       void *Data)
```

PObj: The PObject in question.

Mat: Placeholder, unused.

Data: Placeholder, unused.

Returns: void

Description: Convert a TV list into a Bezier TV list.
See also: IPMshCnvrt2Bezier,

9.2.281 IPMshSaveFile (irit_msh.c:3106)

```
int IPMshSaveFile(const IPObjectStruct *PObj,
                  const char *MSHFileName,
                  IPMSHSaveDfltFileParamsStruct *Params)
```

PObj: IritObject to dump as MSH file.

MSHFileName: Name of MSH file, "-" or NULL for stdout.

Params: The file parameters.

Returns: TRUE if successful, FALSE otherwise.

Description: Dumps IRIT object as a MSH file.

9.2.282 IPNCGCode2Geometry (irit_cnc.c:1181)

```
IPObjectStruct *IPNCGCode2Geometry(VoidPtr IPNCGCodes)
```

IPNCGCodes: Current GCodes' stream.

Returns: The toolpath in IRIT form.

Description: Convert the given stream of G Code toolpath to IRIT geometry. Every linear motion is considered one polyline as long as it is either G0 or G1. Free-air motion (G0) is marked with attribute "freemotion". "ToolNum", "SpindleSpeed" and "SpindleSpeed" attributes are saved on every vertex. As a side effect, computes the length (Len slot) of each G Code motion, and a curve representation (Crv slot).
See also: IPNCGCodeParserInit, IPNCGCodeParserParseLine, , IPNCGCodeParserSetStep, IPNCGCodeParserGetNext, , IPNCGCodeParserNumSteps, IPNCGCodeParserFree, , IPNCGCodeLength, IPNCGCodeBBox,

9.2.283 IPNCGCodeBBox (irit_cnc.c:1452)

```
GMBBBboxStruct *IPNCGCodeBBox(VoidPtr IPNCGCodes, int IgnoreG0Fast)
```

IPNCGCodes: Current GCodes' stream.

IgnoreG0Fast: If TRUE, ignore motions with G0's in bbox computaton.

Returns: The computed bbox returned in a static memory area.

Description: Computes a bounding box over the coordinates found in the G Code stream.
See also: IPNCGCodeParserInit, IPNCGCodeParserParseLine, , IPNCGCodeParserSetStep, IPNCGCodeParserGetNext, , IPNCGCodeLength, IPNCGCodeParserNumSteps, IPNCGCodeParserFree,

9.2.284 IPNCGCodeFastSpeedUpFactor (irit_cnc.c:2438)

```
IrtRType IPNCGCodeFastSpeedUpFactor(VoidPtr IPNCGCodes, IrtRType NewFactor)
```

IPNCGCodes: G Code sequence to traverse.

NewFactor: New Speed multiplicative factor to employ.

Returns: Old value of the simulation multiplicative time factor.

Description: Controls the speed of the simulation as a multiplicative time facotr.

9.2.285 IPNCGCodeGenToolGeom (irit_cnc.c:2071)

```
CagdSrfStruct *IPNCGCodeGenToolGeom(IPNCGCToolType ToolType,
                                    IrtRType Diameter,
                                    IrtRType Height,
                                    IrtRType TorusRadius,
                                    CagdCrvStruct **ToolProfile,
                                    CagdSrfStruct **ToolBottom)
```

ToolType: One of flat, ball, or Torus end.

Diameter: Of tool main cylinder.

Height: Of constructed tool entire geometry. Height must be larger than diameter.

TorusRadius: Only of a Torus end tool - minor radius of torus rounding. TorusRadius must be smaller than Diameter/2.

ToolProfile: A 2D profile cross section curve of the constructed tool in the XZ plane (+Z only). This profile is symmetric with respect to the Z axis, spanning both -X and +X sides and only holds the bottom visible part of the tool (to be used in Z-buffer further processing).

ToolBottom: The bottom part of the tool that will cut material, in the same canonical position.

Returns: Geometry of the constructed tool, NULL if failed.

Description: create tool geometry from tool's parameters, assuming not a general tool in which case this function returns NULL. Tool is created at a canonical position (origin) and orientation (+Z),
See also: IPNCGCodeTraverseStep,

9.2.286 IPNCGCodeHasABC (irit_cnc.c:2416)

```
int IPNCGCodeHasABC(VoidPtr IPNCGCodes)
```

IPNCGCodes: G Code sequence to traverse.

Returns: TRUE if has ABC, FALSE otehrwise.

Description: Returns if we found ABC orientation GCodes in teh stream.

9.2.287 IPNCGCodeLength (irit_cnc.c:1390)

```
IrtRType IPNCGCodeLength(VoidPtr IPNCGCodes, IrtRType *FastLength)
```

IPNCGCodes: Current GCodes' stream.

FastLength: Accumulate fast motion length here, if non NULL.

Returns: Computed arc length.

Description: Computes the accumulated arc length of the given stream of G Code toolpath, in cutting speed motion. Assumes IPNCGCode2Geometry was invoked on this stream to compute each G code individual arc length.
See also: IPNCGCodeParserInit, IPNCGCodeParserParseLine, , IPNCGCodeParserSetStep, IPNCGCodeParserGetNext, , IPNCGCodeBBox, IPNCGCode2Geometry, IPNCGCodeParserNumSteps, , IPNCGCodeParserFree,

9.2.288 IPNCGCodeLoadFile (irit_cnc.c:165)

```
IPObjectStruct *IPNCGCodeLoadFile(const char *NCGCODEFileName,
                                  const IPGcodeLoadDfltFileParamsStruct
                                                                  *Params)
```

NCGCODEFileName: G-Code file to read in.

Params: Loading parameters. ArcCentersRelative - TRUE for arc center in relative coordinates with respect to arc starting location, FALSE if in absolute coordinates. Messages - TRUE, for more messages.

Returns: Read data or NULL if error.

Description: Reads in a G-code CN file and return it as Irit geometry.
See also: IPNCGCodeLoadFileSetDefaultParameters,

9.2.289 IPNCGCodeLoadFileSetDefaultParameters (iritprs1.c:1406)

```
void IPNCGCodeLoadFileSetDefaultParameters(int ArcCentersRelative,
                                           int Messages)
```

ArcCentersRelative: TRUE for arc center in relative coordinates with respect to arc starting location, FALSE if in absolute coordinates.

Messages: TRUE, for more messages.

Returns: void

Description: Sets default loading parameters for GCode NC files.
See also: IPGetObjects, IPNCGCodeLoadFile,

9.2.290 IPNCGCodeParserDone (irit_cnc.c:742)

```
int IPNCGCodeParserDone(VoidPtr IPNCGCodes)
```

IPNCGCodes: Current GCodes' stream.

Returns: TRUE if successful, FALSE otherwise.

Description: Complete the reading of a new stream of G code.
See also: IPNCGCodeParserFree, IPNCGCodeParserParseLine, , IPNCGCodeParserInit, IPNCGCodeParserSetStep, , IPNCGCodeParserGetNext, IPNCGCodeParserNumSteps, , IPNCGCodeParserGetPrev,

9.2.291 IPNCGCodeParserFree (irit_cnc.c:975)

```
void IPNCGCodeParserFree(VoidPtr IPNCGCodes)
```

IPNCGCodes: G codes' stream to free.

Returns: void

Description: Free a processed stream of G codes.
See also: IPNCGCodeParserInit, IPNCGCodeParserParseLine, , IPNCGCodeParserSetStep, IPNCGCodeParserGetNext, , IPNCGCodeParserNumSteps, IPNCGCodeParserGetPrev,

9.2.292 IPNCGCodeParserGetNext (irit_cnc.c:916)

```
IPNCGCodeLineStruct *IPNCGCodeParserGetNext(VoidPtr IPNCGCodes)
```

IPNCGCodes: Current GCodes' stream.

Returns: The next GCode parsed state in the stream. NULL if end of stream.

Description: Returns the next GCode line in the parsed stream IPNCGCodes.
See also: IPNCGCodeParserInit, IPNCGCodeParserParseLine, , IPNCGCodeParserSetStep, IPNCGCodeParserFree, , IPNCGCodeParserNumSteps, IPNCGCodeParserGetPrev,

9.2.293 IPNCGCodeParserGetPrev (irit_cnc.c:946)

```
IPNCGCodeLineStruct *IPNCGCodeParserGetPrev(VoidPtr IPNCGCodes)
```

IPNCGCodes: Current GCodes' stream.

Returns: The previous GCode parsed state in the stream. NULL if beginning of stream.

Description: Returns the previous GCode line in the parsed stream IPNCGCodes.
See also: IPNCGCodeParserInit, IPNCGCodeParserParseLine, , IPNCGCodeParserSetStep, IPNCGCodeParserGetNext, , IPNCGCodeParserNumSteps, IPNCGCodeParserFree,

9.2.294 IPNCGCodeParserInit (irit_cnc.c:241)

```
VoidPtr IPNCGCodeParserInit(int ArcCentersRelative,
                            IrtRType DefFeedRate,
                            IrtRType DefSpindleSpeed,
                            int DefToolNumber,
                            int ReverseZDir,
                            int CoerceEndPoints,
                            IPNCGCodeParserErrorFuncType ErrorFunc)
```

ArcCentersRelative: TRUE if arc centers (G2/G3) are relative to arc starting location, FALSE if centers are absolute.

DefFeedRate: Default feed rate to use if none found. In units per second.

DefSpindleSpeed: Default spindle speed if none found, in RPM.

DefToolNumber: Default tool number to use if none found.

ReverseZDir: TRUE to reverse Z values. If FALSE, +Z points up and toward the machining tool.

CoerceEndPoints: TRUE to force visiting of precise G code end points or (if FALSE) traverse by arclen (may skip end pts).

ErrorFunc: Call back function in case of errors. Can be NULL to fully ignore errors.

Returns: A handle on the G code stream to process.

Description: Initialize a new stream of G code to process.
See also: IPNCGCodeParserDone, IPNCGCodeParserParseLine, , IPNCGCodeParserFree, IPNCGCodeParserSetStep, , IPNCGCodeParserGetNext, IPNCGCodeParserNumSteps, IPNCGCodeParserGetPrev,

9.2.295 IPNCGCodeParserNumSteps (irit_cnc.c:855)

```
int IPNCGCodeParserNumSteps(VoidPtr IPNCGCodes)
```

IPNCGCodes: Current GCodes' stream.

Returns: Number of GCode steps in IPNCGCodes.

Description: Returns the number of GCode steps in the parsed stream IPNCGCodes.
See also: IPNCGCodeParserInit, IPNCGCodeParserParseLine, , IPNCGCodeParserSetStep, IPNCGCodeParserGetNext, , IPNCGCodeParserFree, IPNCGCodeParserGetPrev,

9.2.296 IPNCGCodeParserParseLine (irit_cnc.c:466)

```
VoidPtr IPNCGCodeParserParseLine(VoidPtr IPNCGCodes,
                                 const char *NextLine,
                                 int LineNum)
```

IPNCGCodes: Current GCodes' stream.

NextLine: Next GCode line to process.

LineNum: Line number in the file this line was read from.

Returns: A handle on the G code stream to process

Description: Process one GCode line as specified by NextLine.
See also: IPNCGCodeParserInit, IPNCGCodeParserDone, , IPNCGCodeParserFree, IPNCGCodeParserNumSteps, , IPNCGCodeParserSetStep, IPNCGCodeParserGetNext, , IPNCGCodeParserGetPrev,

9.2.297 IPNCGCodeParserSetStep (irit_cnc.c:884)

```
IPNCGCodeLineStruct *IPNCGCodeParserSetStep(VoidPtr IPNCGCodes,
                                            int NewStep)
```

IPNCGCodes: Current GCodes' stream.

NewStep: The new current step to set to.

Returns: This step's GCode parsed state in the stream. NULL if outside the range of stream.

Description: Sets the current GCode step in the parsed stream IPNCGCodes.
See also: IPNCGCodeParserInit, IPNCGCodeParserParseLine, , IPNCGCodeParserFree, IPNCGCodeParserGetNext, , IPNCGCodeParserNumSteps, IPNCGCodeParserGetPrev,

9.2.298 IPNCGCodeResetFeedRates (irit_cnc.c:2389)

```
void IPNCGCodeResetFeedRates(VoidPtr IPNCGCodes)
```

IPNCGCodes: G Code sequence to traverse.

Returns: void

Description: Reset the feedrates to original inputfile values.

9.2.299 IPNCGCodeSave2File (irit_cnc.c:2316)

```
int IPNCGCodeSave2File(VoidPtr IPNCGCodes, const char *FName)
```

IPNCGCodes: Current GCodes' stream.

FName: File name to save the G Codes into, "-" for stdout.

Returns: TRUE if successful, false otherwise.

Description: Save the given G code sequence into a file.
See also: IPNCGCodeParserInit, IPNCGCodeParserParseLine, , IPNCGCodeParserSetStep, IPNCGCodeParserGetNext, , IPNCGCodeLength, IPNCGCodeParserNumSteps, IPNCGCodeParserFree,

9.2.300 IPNCGCodeSaveFile (iritwcnc.c:102)

```
int IPNCGCodeSaveFile(const IPObjectStruct *PObj,
                      IrtHmgnMatType CrntViewMat,
                      const char *NCGCODEFileName,
                      int Messages,
                      int Units,
                      int OutputType)
```

PObj: IritObject to dump as NCGCODE file.

CrntViewMat: The current viewing matrix to apply to the object.

NCGCODEFileName: Name of NCGCODE file, "-" or NULL for stdout.

Messages: TRUE for warning messages.

Units: 0 for inches, 1 for Milimeters.

OutputType: 0 for CNC machining, 1 for CNC laser, 2 for AM printer.

Returns: TRUE if successful, FALSE otherwise.

Description: Dumps IRIT object as an NCGCode file.
See also: IPNCGCodeSaveFileSetTol,

9.2.301 IPNCGCodeTraverseInit (irit_cnc.c:1532)

```
IrtRType IPNCGCodeTraverseInit(VoidPtr IPNCGCodes,
                               IrtRType InitTime,
                               IrtRType FastSpeedUpFactor,
                               IrtRType TriggerArcLen)
```

IPNCGCodes: G Code sequence to traverse.

InitTime: Initial time of animation.

FastSpeedUpFactor: Speedup multiplier for fast G0 motion.

TriggerArcLen: A minimal arc length to accumulate to create triggers every such arc length, or non-positive value to ignore.

Returns: Total arc-length of cutting speed motion

Description: Init a request to traverse this sequence of G Code.
See also: IPNCGCodeLength, IPNCGCodeTraverseTime, IPNCGCodeTraverseTriggerAAL,

9.2.302 IPNCGCodeTraverseLines (irit_cnc.c:2363)

```
const char *IPNCGCodeTraverseLines(VoidPtr IPNCGCodes, int Restart)
```

IPNCGCodes: Current GCodes' stream.

Restart: TRUE to init the traversal process.

Returns: Next line, NULL if done.

Description: Traverses the given G code sequence as strings.
See also: IPNCGCodeParserInit, IPNCGCodeParserParseLine, , IPNCGCodeParserSetStep, IPNCGCodeParserGetNext, , IPNCGCodeLength, IPNCGCodeParserNumSteps, IPNCGCodeParserFree,

9.2.303 IPNCGCodeTraverseStep (irit_cnc.c:1914)

```
IrtRType IPNCGCodeTraverseStep(VoidPtr IPNCGCodes,
                               IrtRType Step,
                               IrtRType *NewRealTime,
                               IrtPtType NewToolPosition,
                               IPNCGCodeLineStruct **NewGC)
```

IPNCGCodes: G Code sequence to traverse.

Step: Delta position step to add/subtract from current pos. Can be negative to subtract and go backward.

NewRealTime: New real time of new position is returned here with respect to starting time (that is 0.0).

NewToolPosition: The traversed tool position is saved here.

NewGC: GCode info at NewToolPosition.

Returns: New arc-length of cutting speed motion so far. -1.0 is returned if we completed the traversal, -2.0 is returned if we when back to the start point.

Description: Advance the animated tool motion along this sequence of G Code by Step.
See also: IPNCGCodeLength, IPNCGCodeTraverseInit, IPNCGCodeTraverseTime, IPNCGCodeTraverseTriggerAAL,

9.2.304 IPNCGCodeTraverseTime (irit_cnc.c:1725)

```
IrtRType IPNCGCodeTraverseTime(VoidPtr IPNCGCodes,
                               IrtRType Dt,
                               IrtRType *NewRealTime,
                               IrtPtType NewToolPosition,
                               IrtPtType NewToolOrientation,
                               IPNCGCodeLineStruct **NewGC)
```

IPNCGCodes: G Code sequence to traverse.

Dt: Delta time step to add/subtract from current time. Can be negative to subtract and go backward.

NewRealTime: New real time of new position is returned here with respect to starting time (that is 0.0).

NewToolPosition: The traversed tool position is saved here.

NewToolOrientation: If multi-axis tool path the orientation (ABC GCodes) will be saved here.

NewGC: GCode info at NewToolPosition/NewToolOrientation.

Returns: New arc-length of cutting speed motion so far. -1.0 is returned if we completed the traversal, -2.0 is returned if we when back to the start point. -3.0 is returned if we have an empty GCode file.

Description: Advance the animated tool motion along this sequence of G Code to Time.
See also: IPNCGCodeLength, IPNCGCodeTraverseInit, IPNCGCodeTraverseStep, IPNCGCodeTraverseTriggerAAL,

9.2.305 IPNCGCodeTraverseTriggerAAL (irit_cnc.c:1583)

```
int IPNCGCodeTraverseTriggerAAL(VoidPtr IPNCGCodes,
                                IPNCGCodeEvalMRRFuncType EvalMRR,
                                VoidPtr MRRData)
```

IPNCGCodes: G Code sequence to traverse.

EvalMRR: Function to invoke to compute the material removal rate in this arclen interval, or NULL to ignore. This function should return 1.0 if feedrate is fine or a multiplicative factor to modify the feedrate otherwise.

MRRData: A pointer to pass to EWvalMRR as its single parameter.

Returns: TRUE if generated a trigger, FALSE otherwise.

Description: Handle accumulations of arc-length (AAL). Tests and generates triggers every prescribed arc length. If a trigger is generated and EvalMRR is not NULL, this function is invoked to evaluate the material removal rate in this arc length interval. EvalMRR should return a 1.0 if the feed rate is appropriate and return a multiplicative factor to modify the feed rate if not.
See also: IPNCGCodeLength, IPNCGCodeTraverseInit, IPNCGCodeTraverseStep, IPNCGCodeTraverseTime,

9.2.306 IPNCGCodeUpdateGCodeIndexCBFunc (irit_cnc.c:1644)

```
IPNCGCodeIndexUpdateFuncType IPNCGCodeUpdateGCodeIndexCBFunc(
                                        VoidPtr IPNCGCodes,
                                        IPNCGCodeIndexUpdateFuncType Func,
                                        void *Data)
```

IPNCGCodes: G Code sequence to traverse.

Func: New call back function to use or NULL to disable.

Data: Optional call back data to transfer. Can be NULL.

Returns: Old call back function.

Description: Sets a call back function to be invoked on every G code index change in the simulation.
See also: IPNCGCodeTraverseInit,

9.2.307 IPOBJLoadFile (obj_irit.c:397)

```
IPObjectStruct *IPOBJLoadFile(const char *OBJFileName,
                              const IPOBJLoadDfltFileParamsStruct *Params)
```

OBJFileName: Name of OBJ file. NULL means standard input.

Params: Loading parameters. WarningMsgs - Weather to display warning messages or not. WhiteDiffuseTexture - When material has texture and RGB (0,0,0) give it RGB (1,1,1). IgnoreFullTransp - Full transparency of material is ignored. ForceSmoothing - If the s statement isn't given use smoothing for all the polygons.

Returns: Read OBJ DATA or NULL if error.

Description: Read an OBJ file into IRIT data structure.
See also: IPOBJSaveFile, IPOBJLoadFileSetDefaultParameters,

9.2.308 IPOBJLoadFileSetDefaultParameters (iritprs1.c:1378)

```
void IPOBJLoadFileSetDefaultParameters(int WarningMsgs,
                                       int WhiteDiffuseTexture,
                                       int IgnoreFullTransp,
                                       int ForceSmoothing)
```

WarningMsgs: Weather to display warning messages or not.

WhiteDiffuseTexture: When material has texture and RGB (0,0,0) give it RGB (1,1,1).

IgnoreFullTransp: Full transparency of material is ignored.

ForceSmoothing: If the s statement isn't given use smoothing for all the polygons.

Returns: void

Description: Sets default loading parameters for OBJ files.
See also: IPGetObjects, IPOBJLoadFile,

9.2.309 IPOBJSaveFile (irit_obj.c:154)

```
int IPOBJSaveFile(const IPObjectStruct *PObj,
                  const char *OBJFileName,
                  int WarningMsgs,
                  int CnvxTriangPolys,
                  int UniqueVertices)
```

PObj: IRIT data structure to convert.

OBJFileName: Name of OBJ file to write the result to.

WarningMsgs: Whether to display warning messages or not.

CnvxTriangPolys: 0 to ignore, 1 for convex polygons, 2 for triangles only. Employed only for input that is polygonal.

UniqueVertices: If true, don't gather identical vertices to one vector. Each polygon uses its own vertices.

Returns: TRUE if succeeded.

Description: Convert IRIT data structure to an OBJ file.
See also: IPOBJLoadFile,

9.2.310 IPODAddDependencyToObj (obj_dpnd.c:160)

```
void IPODAddDependencyToObj(IPODObjectDpndncyStruct **ObjDpnd, char *DpndName)
```

ObjDpnd: Object to update dependency on object named DpndName.

DpndName: Name of dependency object. May be NULL.

Returns: void

Description: Adds a new object name DpndName as a dependent on this object. This function will properly initialize a NULL ObjDpnd if found one.
See also: IPODAddParameterToObj,

9.2.311 IPODAddParameterToObj (obj_dpnd.c:124)

```
void IPODAddParameterToObj(IPODObjectDpndncyStruct **ObjDpnd, char *ParamName)
```

ObjDpnd: Object to update object named ParamName as a parameter of of this object. May be NULL

ParamName: Name of parameter object.

Returns: void

Description: Adds a new object name ParamName as a parameter of this object. This function will properly initialize a NULL ObjDpnd if found one.
See also: IPODAddDepaendencyToObj,

9.2.312 IPODCopyDependencies (obj_dpnd.c:341)

```
IPODObjectDpndncyStruct *IPODCopyDependencies(IPODObjectDpndncyStruct *Dpnds)
```

Dpnds: Structure to duplicate.

Returns: Duplicated structure.

Description: Copy the dependency structure Dpnds.
See also: IPODCopyParametersOfObj, IPODCopyDependenciesOfObj,

9.2.313 IPODCopyDependenciesOfObj (obj_dpnd.c:313)

```
IPODDependsStruct *IPODCopyDependenciesOfObj(IPODDependsStruct *ObjDepends)
```

ObjDepends: List of dependencies to copy.

Returns: Copies list of dependencies.

Description: Copy the dependency list of this objects (list of other objects depending on this one).
See also: IPODCopyDependencies, IPODCopyParametersOfObj,

9.2.314 IPODCopyParametersOfObj (obj_dpnd.c:284)

```
IPODParamsStruct *IPODCopyParametersOfObj(IPODParamsStruct *ObjParams)
```

ObjParams: List of parameters to copy.

Returns: Copies list of parameters.

Description: Copy the parameter list dependencies (list of objects this object depends upon).
See also: IPODCopyDependencies, IPODCopyDependenciesOfObj,

9.2.315 IPODDelDependencyFromObj (obj_dpnd.c:241)

```
void IPODDelDependencyFromObj(IPODObjectDpndncyStruct *ObjDpnd,
                              char *DpndName)
```

ObjDpnd: Dependency structure of this object.

DpndName: Dependency object name to remove.

Returns: void

Description: Removes dependency of object named DpndName on this object.
See also: IPODDelParameterFromObj,

9.2.316 IPODDelParameterFromObj (obj_dpnd.c:196)

```
void IPODDelParameterFromObj(IPODObjectDpndncyStruct *ObjDpnd,
                             char *ParamName)
```

ObjDpnd: Dependency structure of this object.

ParamName: Parameter object name to remove.

Returns: void

Description: Removes a parameter this object depends upon from this object's dependency structure.
See also: IPODDelDependencyFromObj,

9.2.317 IPODEvalOneObject (obj_dpnd.c:493)

```
void IPODEvalOneObject(IPObjectStruct *PObj)
```

PObj: To reevaluate.

Returns: void

Description: Reevaluate this object, based upon its dependency's EvalExpr.
See also: IPODUpdateAllDependencies,

9.2.318 IPODFreeDependencies (obj_dpnd.c:442)

```
void IPODFreeDependencies(IPODObjectDpndncyStruct *Dpnds)
```

Dpnds: Structure to free.

Returns: void

Description: Free the dependency structure Dpnds.
See also: IPODFreeParametersOfObj, IPODFreeDependenciesOfObj,

9.2.319 IPODFreeDependenciesOfObj (obj_dpnd.c:411)

```
void IPODFreeDependenciesOfObj(IPODDependsStruct *ObjDepends)
```

ObjDepends: List of dependencies to free.

Returns: void

Description: Free the dependency list of this objects (list of other objects depending on this one).
See also: IPODFreeParametersOfObj, IPODFreeDependencies,

9.2.320 IPODFreeParametersOfObj (obj_dpnd.c:379)

```
void IPODFreeParametersOfObj(IPODParamsStruct *ObjParams)
```

ObjParams: List of parameters to free.

Returns: void

Description: Free the parameter list dependencies (list of objects this object depends upon).
See also: IPODFreeDependenciesOfObj, IPODFreeDependencies,

9.2.321 IPODNewDependencies (obj_dpnd.c:94)

```
IPODObjectDpndncyStruct *IPODNewDependencies(void)
```

Returns: Allocated structure.

Description: Allocates new dependency structure to hold all dependencies and params.
See also: IPODNewParametersOfObj, IPODNewDependenciesOfObj,

9.2.322 IPODNewDependenciesOfObj (obj_dpnd.c:64)

```
IPODDependsStruct *IPODNewDependenciesOfObj(char *Name,
                                            IPODDependsStruct *Pnext)
```

Name: Name of dependency, NULl if none.

Pnext: Next dependency, NULL if none.

Returns: Allocated structure.

Description: Allocates new dependency on this object (of another object) structure.
See also: IPODNewDependencies, IPODNewParametersOfObj,

9.2.323 IPODNewParametersOfObj (obj_dpnd.c:32)

```
IPODParamsStruct *IPODNewParametersOfObj(char *Name, IPODParamsStruct *Pnext)
```

Name: Name of parameter, NULl if none.

Pnext: Next parameter, NULL if none.

Returns: Allocated structure.

Description: Allocates new parameter (of this object) dependency structure.
See also: IPODNewDependencies, IPODNewDependenciesOfObj,

9.2.324 IPODPrintDependencies (obj_dpnd.c:513)

```
void IPODPrintDependencies(IPObjectStruct *PObj)
```

PObj: To print its dependency structure.

Returns: void

Description: Debug function to print the content of the dependency structure.

9.2.325 IPODUpdateAllDependencies (obj_dpnd.c:472)

```
void IPODUpdateAllDependencies(IPODObjectDpndncyStruct *ObjDpnd)
```

ObjDpnd: To start this recursive visit.

Returns: void

Description: Recursively visit all objects this ObjDpnd affects (all other objects that depends on this one) and reevaluate them.
See also: IPODEvalOneObject,

9.2.326 IPObjListLen (linklist.c:1276)

length

linked lists

```
int IPObjListLen(const IPObjectStruct *O)
```

O: Object list to compute its length.

Returns: Number of elements in O list.

Description: Returns the length of a list of objects.

9.2.327 IPObjLnkListToListObject (linklist.c:560)

```
IPObjectStruct *IPObjLnkListToListObject(IPObjectStruct *ObjLnkList)
```

ObjLnkList: Linked list of object to convert.

Returns: A list object holding all items in linked list.

Description: Converts a linked list of objects into one list object, in place.
See also: IPAppendListObjects, IPLnkListToListObject, IPLinkedListToObjList, , IPHierarchyObjToLinkedList,

9.2.328 IPObjTypeAsString (allocate.c:87)

```
const char *IPObjTypeAsString(const IPObjectStruct *PObj)
```

PObj: To return its type as a string.

Returns: The string description of the object type.

Description: Returns a a string-description of the given object's type.

9.2.329 IPOpenDataFile (iritprs1.c:157)

files

parser

```
int IPOpenDataFile(const char *FileName, int Read, int Messages)
```

FileName: To try and open.

Read: If TRUE assume a read operation, otherwise write.

Messages: Do we want error/warning message?

Returns: A handler to the open file, -1 if error.

Description: Open a data file for read/write. Data file can be either Ascii IRIT data file or binary IRIT data file. A binary data file must have a ".ibd" (for Irit Binary Data) file type. Under unix, file names with the postfix ".Z" are assumed compressed and treated accordingly.
See also: IPGetObjects, IPSetPolyListCirc, IPSetFlattenObjects, , SetReadOneObject,

9.2.330 IPOpenDataWFile (iritprs1.c:298)

files

parser

```
int IPOpenDataWFile(const wchar_t *FileName, int Read, int Messages)
```

FileName: To try and open.

Read: If TRUE assume a read operation, otheriwse write.

Messages: Do we want error/warning message?

Returns: A handler to the open file, -1 if error.

Description: Open a data file for read/write. Data file can be either Ascii IRIT data file or binary IRIT data file. A binary data file must have a ".ibd" (for Irit Binary Data) file type. Under unix, file names with the postfix ".Z" are assumed compressed and treated accordingly.
See also: IPGetObjects, IPSetPolyListCirc, IPSetFlattenObjects, , SetReadOneObject,

9.2.331 IPOpenPolysToClosed (ip_cnvrt.c:2423)

```
void IPOpenPolysToClosed(IPPolygonStruct *Pls)
```

Pls: Polygons to process, in place.

Returns: void

Description: Forces the given list of polygons to have closed list of vertices
See also: IPClosedPolysToOpen, GMVrtxListToCircOrLin,

9.2.332 IPOpenStreamFromCallBackIO (iritprs1.c:640)

```
int IPOpenStreamFromCallBackIO(IPStreamReadCharFuncType ReadFunc,
                               IPStreamWriteBlockFuncType WriteFunc,
                               IPStreamFormatType Format,
                               int Read,
                               int IsBinary)
```

ReadFunc: A call back function to read a character (non blocking). will be ignored if Read == FALSE.

WriteFunc: A call back function to write a block of data. will be ignored if Read == TRUE.

Format: Format of call back.

Read: TRUE for reading from f, FALSE for writing to f.

IsBinary: Is it a binary file?

Returns: A handle on the constructed stream.

Description: Open a stream using direct call back function(s) to read/write data.
See also: IPOpenStreamFromFile, IPProcessFreeForm2,

9.2.333 IPOpenStreamFromFile (iritprs1.c:683)

```
int IPOpenStreamFromFile(FILE *f,
                         int Read,
                         int IsBinary,
                         int IsCompressed,
                         int IsPipe)
```

f: A handle to the open file.
Read: TRUE for reading from f, FALSE for writing to f.
IsBinary: Is it a binary file?
IsCompressed: Is it compressed file?
IsPipe: Is it a pipe?
Returns: A handle on the constructed stream.
Description: Converts an open file into a stream.
See also: IPOpenStreamFromCallBackIO, IPProcessFreeForm2,

9.2.334 IPOpenStreamFromFile2 (iritprs1.c:715)

IPOpenStreamFromFile

```
int IPOpenStreamFromFile2(FILE *f,
                          int Read,
                          IPStreamFormatType Format,
                          int IsBinary,
                          int IsCompressed,
                          int IsPipe)
```

f: A handle to the open file.
Read: TRUE for reading from f, FALSE for writing to f.
Format: IRIT Dat, VRML, etc.
IsBinary: Is it a binary file?
IsCompressed: Is it compressed file?
IsPipe: Is it a pipe?
Returns: A handle on the constructed stream.
Description: Converts an open file into a stream.
See also: IPOpenStreamFromFile, IPOpenStreamFromCallBackIO,

9.2.335 IPOpenStreamFromSocket (iritprs1.c:765)

```
int IPOpenStreamFromSocket(int Soc, int IsBinary)
```

Soc: A handle to the open socket.
IsBinary: Is it a binary file?
Returns: A handle on the constructed stream.
Description: Converts an open socket into a stream.

9.2.336 IPOpenStreamFromVrml (iritvrml.c:697)

```
int IPOpenStreamFromVrml(FILE *f, int Read, int IsBinary, int IsPipe)
```

f: A handle to the open file.
Read: TRUE for reading from f, FALSE for writing to f.
IsBinary: Is it a binary file? Currently only text vrml is supported.
IsPipe: Is it a pipe?
Returns: A handle on the constructed stream.
Description: Converts an open file into a stream.
See also: IPOpenStreamFromFile,

9.2.337 IPOpenVrmlFile (iritvrml.c:650)

files

parser

```
int IPOpenVrmlFile(const char *FileName,
                   int Messages,
                   IrtRType Resolution)
```

FileName: To try and open.

Messages: Do we want error/warning messages?

Resolution: Pass Irit interpreter state variable, due to the need of freeforms to polygon conversions.

Returns: A handler to the open file, -1 if error.

Description: Open a data file for write. Data file can be Ascii VRML 2.0 data file only.
See also: IPGetObjects, IPSetPolyListCirc, IPSetFlattenObjects, IPSetReadOneObject,

9.2.338 IPPolyListLen (linklist.c:1254)

length

linked lists

```
int IPPolyListLen(const IPPolygonStruct *P)
```

P: Polygon list to compute its length.

Returns: Number of elements in P list.

Description: Returns the length of a list of polygons.
See also: GMGetObjNumOfPolys,

9.2.339 IPPolyVrtxArrayFree (allocate.c:992)

```
void IPPolyVrtxArrayFree(IPPolyVrtxArrayStruct *PVIdx)
```

PVIdx: Data structure to free.

Returns: void

Description: Release a data structure of a polygonal mesh as vertex/polygon index structure with a linear vector of NumVrtcs vertices and NumPlys polygons.
See also: IPPolyVrtxArrayNew,

9.2.340 IPPolyVrtxArrayNew (allocate.c:956)

```
IPPolyVrtxArrayStruct *IPPolyVrtxArrayNew(int NumVrtcs, int NumPlys)
```

NumVrtcs: Number of different vertices in the mesh.

NumPlys: Number of polygons in the mesh.

Returns: The constructed data structure.

Description: Allocate a data structure of a polygonal mesh as vertex/polygon index structure with a linear vector of NumVrtcs vertices and NumPlys polygons.
See also: IPPolyVrtxArrayFree,

9.2.341 IPPolyVrtxArrayNew2 (allocate.c:924)

```
IPPolyVrtxArrayStruct *IPPolyVrtxArrayNew2(IPObjectStruct *PObj)
```

PObj: Polygonal mesh to convert to vertex/polygon index struct.

Returns: The constructed data structure.

Description: Allocate a data structure of a polygonal mesh as vertex/polygon index structure with a linear vector of NumVrtcs vertices and NumPlys polygons.
See also: IPPolyVrtxArrayFree, IPPolyVrtxArrayNew,

9.2.342 IPPolygonSetErrFunc (ip_cnvrt.c:1124)

```
CagdPlgErrorFuncType IPPolygonSetErrFunc(CagdPlgErrorFuncType Func)
```

Func: New function to use, NULL to disable.

Returns: Old value of function.

Description: Sets the polygon approximation error function. The error function will return a negative value if this triangle must be purged or otherwise a non negative error measure.

9.2.343 IPPolyline2Curve (ip_cnvrt.c:433)

conversion

approximation

```
CagdCrvStruct *IPPolyline2Curve(const IPPolygonStruct *Pl, int Order)
```

Pl: To convert to a curve.

Order: Typically 2 for linear, but can be higher order as well. In all cases ,the input polyline points serve as control points.

Returns: A curve.

Description: Routine to convert one polyline into a curve, typically linear.
See also: CagdCnvrtPolyline2LinBspCrv,

9.2.344 IPProcessFreeForm (ip_procs.c:32)

conversion

```
IPObjectStruct *IPProcessFreeForm(IPFreeFormStruct *FreeForms)
```

FreeForms: Freeform geometry to process, in place.

Returns: Processed freeform geometry. This function simply returns what it got.

Description: Default processor of read freeform geometry. This routine does not process the freeform geometry in any way. Other programs can, for example, convert the freeform shapes to polygons or polylines using the call back function or purge the freeform data if it is not supported. Processing should be done in place.

9.2.345 IPProcessModel2TrimSrfs (iritprs1.c:1778)

conversion

```
int IPProcessModel2TrimSrfs(IPFreeFormStruct *FreeForms)
```

FreeForms: Freeform model to process into trimmed srfs, in place.

Returns: TRUE if models were found and processed, FALSE otherwise.

Description: Process a (v)model freeform into trimmed surfaces freeform, in place.

9.2.346 IPProcessReadObject (iritprs1.c:1467)

files

parser

```
IPObjectStruct *IPProcessReadObject(IPObjectStruct *PObj)
```

PObj: Object to process.

Returns: Processed object, in place.

Description: Process a read object, in place, before returning it to the caller. List objects of zero or one elements are eliminated. Attributes are propagated throughout the hierarchy. If FlattenTree mode (see IPSetFlattenObjects) hierarchy is flattened out.

9.2.347 IPPropagateObjectName (allocate.c:2088)

copy

```
void IPPropagateObjectName(IPObjectStruct *Obj,
                           const char *ObjName,
                           int ForceRename)
```

Obj: To propagate object names from. Root of hierarchy.

ObjName: Object name to propagate, NULL if to be picked up from Obj.

ForceRename: Force subobject rename even if subobject has a valid name.

Returns: void

Description: Routine to propagate the object name down the object hierarchy. Objects with no name assign will inherit the name propagated to them from above. If input name is ObjName, the hierarchy will be assigned names of the form RootName{_%d}*.

9.2.348 IPPutAttributes (iritprs2.c:707)

```
void IPPutAttributes(int Handler, const IPAttributeStruct *Attr, int Indent)
```

Handler: A handler to the open stream.

Attr: Attributes to put out.

Indent: Indentation to put attributes at.

Returns: void

Description: Routine to print the attributes of given attribute list.
See also: AttrTraceAttributes,

9.2.349 IPPutBinObject (iritprsb.c:1724)

files

parser

```
void IPPutBinObject(int Handler, const IPObjectStruct *PObj)
```

Handler: A handler to the open stream. *

PObj: Object to write.

Returns: void

Description: Routine to write one object to a given binary file, directly. Objects may be recursively defined, as lists of objects.

9.2.350 IPPutMatrixFile (iritprs2.c:826)

```
int IPPutMatrixFile(const char *File,
                    IrtHmgnMatType ViewMat,
                    IrtHmgnMatType ProjMat,
                    int HasProjMat)
```

File: File to write the matrix file to.

ViewMat, ProjMat: Matrices to put.

HasProjMat: TRUE if has a perspective matrix, FALSE otherwise.

Returns: TRUE if successful, FALSE otherwise.

Description: Puts an IRIT matrix file.
See also: IPGetMatrixFile,

9.2.351 IPPutObjectToFile (iritprs2.c:106)

files

```
void IPPutObjectToFile(FILE *f, const IPObjectStruct *PObj, int IsBinary)
```

f: Output stream file handle.

PObj: Object to put on output stream.

IsBinary: Is this a binary file we should dump?

Returns: void

Description: Routine to print the data from given object into given file handle. See function IPSetPrintFunc, IPSetFloatFormat.

9.2.352 IPPutObjectToFile2 (iritprs2.c:150)

files

```
void IPPutObjectToFile2(FILE *f, const IPObjectStruct *PObj, int Indent)
```

f: Output stream.

PObj: Object to put on output stream.

Indent: File indentation (always a text file).

Returns: void

Description: Routine to print the data from given object into given file f. See function IPSetPrintFunc, IPSetFloatFormat.

9.2.353 IPPutObjectToFile3 (iritprs2.c:188)

files

```
void IPPutObjectToFile3(const char *FName,
                        const IPObjectStruct *PObj,
                        int Indent)
```

FName: Output file name.

PObj: Object to put on output.

Indent: File indentation (always a text file).

Returns: void

Description: Routine to print the data from given object into given file FName.

9.2.354 IPPutObjectToHandler (iritprs2.c:303)

files

```
void IPPutObjectToHandler(int Handler, const IPObjectStruct *PObj)
```

Handler: A handler to the open stream.

PObj: Object to put on output stream.

Returns: void

Description: Routine to print the data from given object into given file designated via Handler.
See also: IPSetPrintFunc, IPSetFloatFormat, IPPrintFunc,

9.2.355 IPPutVrmlViewPoint (iritvrml.c:2431)

```
void IPPutVrmlViewPoint(int Handler, IrtHmgnMatType *Mat, int Indent)
```

Handler: To put the generated VRML into.

Mat: To dump as the viewing matrix.

Indent: Level of indentation in VRML.

Returns: void

Description: Update a view point in VRML style based upon a VIEW_MAT matrix in the geometry stream.

9.2.356 IPReallocNewTypeObject (allocate.c:2167)

allocation

```
void IPReallocNewTypeObject(IPObjectStruct *PObj, IPObjStructType ObjType)
```

PObj: Object to reallocated as a new object of type ObjType.

ObjType: New type for object PObj.

Returns: void

Description: Routine to reallocate as necessary and object to a new object type.

9.2.357 IPResolveInstances (iritprs1.c:1082)

```
IPObjectStruct *IPResolveInstances(IPObjectStruct *PObjects)
```

PObjects: To eliminate instances from, in place.

Returns: Same geometry as in PObjects but without instances.

Description: Resolves, in place, all instances in the objects into their proper geometry. This functions hence eliminates all instances' objects while increasing the size of the data, and do so in place.

9.2.358 IPReverseListObj (linklist.c:90)

```
IPObjectStruct *IPReverseListObj(IPObjectStruct *ListObj)
```

ListObj: List object to reverses its entries, in place.

Returns: Reversed list object.

Description: Reverses a list. Original input list is used in place.
See also: IPReverseObjList,

9.2.359 IPReverseObjList (linklist.c:123)

reverse

files

parser

```
IPObjectStruct *IPReverseObjList(IPObjectStruct *PObj)
```

PObj: A list of objects to reverse.

Returns: Reverse list of objects, in place.

Description: Reverses a list of objects, in place.
See also: IPReverseListObj,

9.2.360 IPReverseObject (coerce.c:1484)

```
IPObjectStruct *IPReverseObject(const IPObjectStruct *PObj)
```

PObj: Input object to reverse.

Returns: Reversed object.

Description: Returns a similar hierarchy to given one but with reversed semantics.

9.2.361 IPReversePlList (linklist.c:154)

reverse

files

parser

```
IPPolygonStruct *IPReversePlList(IPPolygonStruct *PPl)
```

PPl: A list of polygons to reverse.

Returns: Reversed list of polygons, in place.

Description: Reverses a list of polygons, in place.

9.2.362 IPReverseVrtxList (linklist.c:187)

reverse
files
parser

```
void IPReverseVrtxList(IPPolygonStruct *Pl)
```

Pl: A polygon to reverse its vertex list, in place.

Returns: void

Description: Reverses the vertex list of a given polygon. This is used mainly to reverse polygons such that cross product of consecutive edges which form a convex corner will point in the polygon normal direction.

9.2.363 IPReverseVrtxList2 (linklist.c:252)

reverse
files
parser

```
IPVertexStruct *IPReverseVrtxList2(IPVertexStruct *PVertex)
```

PVertex: A list of vertices to reverse.

Returns: Reversed list of vertices, in place.

Description: Reverses a list of vertices of a polyline, in place. The list is assumed to be non circular and hence can be treated as a polyline.

9.2.364 IPSTLLoadFile (stl_irit.c:72)

```
IPObjectStruct *IPSTLLoadFile(const char *STLFileName,
                              const IPSTLLoadDfltFileParamsStruct *Params)
```

STLFileName: Name of STL file.

Params: Loading Parameters - BinarySTL - TRUE if input STL is a binary file. EndianSwap - Swap non char-long entities (little vs. big binaries endings). NormalFlip - Flip normal directions if TRUE. Messages - 1 for error messages, 2 to include warning messages, 3 to include informative messages. 4 to include dump of IRIT objects.

Returns: Read STL DATA or NULL if error.

Description: Read STL file into IRIT data.
See also: IPSTLSaveFile, IPSTLLoadFileSetDefaultParameters,

9.2.365 IPSTLLoadFileSetDefaultParameters (iritprs1.c:1346)

```
void IPSTLLoadFileSetDefaultParameters(int BinarySTL,
                                       int EndianSwap,
                                       int NormalFlip,
                                       int Messages)
```

BinarySTL: TRUE if input STL is a binary file.

EndianSwap: Swap non char-long entities (little vs. big binaries endings).

NormalFlip: Flip normal directions if TRUE.

Messages: 1 for error messages, 2 to include warning messages, 3 to include informative messages. 4 to include dump of IRIT objects.

Returns: void

Description: Sets default loading parameters for STL files.
See also: IPGetObjects, IPSTLLoadFile,

9.2.366 IPSTLSaveFile (irit_stl.c:96)

```
int IPSTLSaveFile(const IPObjectStruct *PObj,
                  IrtHmgnMatType CrntViewMat,
                  const char *STLFileName,
                  const IPSTLSaveDfltFileParamsStruct *Params)
```

PObj: IritObject to dump as STL file.

CrntViewMat: The current viewing matrix to apply to the object.

STLFileName: Name of STL file, "-" or NULL for stdout.

Params: Parameters of saved STL.

Returns: TRUE if successful, FALSE otherwise.

Description: Dumps IRIT object as an STL file. If the input PObj has a non zero "binary" attribute, it is saved as binary STL. If the binary attrib. is negative, no date/time is saved in header.
See also: IPSTLLoadFile, IPSTLSaveSetVrtxEps,

9.2.367 IPSTLSaveSetVrtxEps (irit_stl.c:59)

```
IrtRType IPSTLSaveSetVrtxEps(IrtRType SameVrtxEps)
```

SameVrtxEps: New epsilon.

Returns: Old epsilon.

Description: Sets the epsilon equality of two vertices, to be considered the same.
See also: IPSTLSaveFile,

9.2.368 IPSenseBinaryFile (iritprs1.c:501)

```
int IPSenseBinaryFile(const char *FileName)
```

FileName: File to sense.

Returns: TRUE if binary, FALSE if text.

Description: Senses if a given file (name) is a binary or a text file.
See also: IPSenseFileType,

9.2.369 IPSenseCompressedFile (iritprs1.c:537)

```
int IPSenseCompressedFile(const char *FileName)
```

FileName: File to sense.

Returns: TRUE if compressed, FALSE else.

Description: Senses if a given file (name) is a compressed file.
See also: IPSenseFileType,

9.2.370 IPSenseFileType (iritprs1.c:425)

```
IPStreamFormatType IPSenseFileType(const char *FileName)
```

FileName: The files' name to sense file type from.

Returns: Type of file detected.

Description: Attempts to detect the type of the file from its name.
See also: IPSenseBinaryFile,

9.2.371 IPSetCopyObjectReferenceCount (allocate.c:2201)

```
int IPSetCopyObjectReferenceCount(int RefCount)
```

RefCount: TRUE for reference count, FALSE for extensive copy.

Returns: Old value of reference count state.

Description: Controls of a copy is via reference counts or extensive (no ref. counts)
See also: IPCopyObject,

9.2.372 IPSetCurvesToCubicBzrTol (ip_cnvrt.c:1931)

approximation

```
IrtRType IPSetCurvesToCubicBzrTol(IrtRType Tolerance)
```

Tolerance: Of approximation to use.

Returns: Old value of cubic Bezier tolerance.

Description: Sets the tolerance that is used by the Bezier to Cubic Bezier conversion routines IritCurvesToCubicBzrCrvs and IritSurfacesToCubicBzrCrvs

9.2.373 IPSetFatalErrorFunc (prsr_ftl.c:29)

error handling

```
IPSetErrorFuncType IPSetFatalErrorFunc(IPSetErrorFuncType ErrorFunc)
```

ErrorFunc: New error function to use.

Returns: Old error function reference.

Description: Sets the error function to be used by Prsr_lib.

9.2.374 IPSetFilterDegen (iritprs2.c:965)

files

degeneracies

```
int IPSetFilterDegen(int FilterDegeneracies)
```

FilterDegeneracies: TRUE to filter, FALSE to load/dump anyway.

Returns: Old value of this state.

Description: Sets the filtering mode of degenerated geomerty while save/load.

9.2.375 IPSetFlattenObjects (iritprs1.c:1495)

files

parser

```
int IPSetFlattenObjects(int Flatten)
```

Flatten: If TRUE, list objects will be flattened out to a long linear list. If FALSE, read object will be unchanged.

Returns: Old value of flatten state.

Description: Controls the hierarchy flattening of a read object.

9.2.376 IPSetFloatFormat (iritprs2.c:991)

files

```
char *IPSetFloatFormat(const char *FloatFormat)
```

FloatFormat: A printf style floating point printing format string.

Returns: Old float format.

Description: Sets the floating point printing format.

9.2.377 IPSetPolyListCirc (prsrgeom.c:225)

files

parser

```
int IPSetPolyListCirc(int Circ)
```

Circ: If TRUE, vertex lists of polygons will be circular. If FALSE, the lists will be NULL terminated.

Returns: Old value of flag.

Description: Controls vertex list in polygons. Do we want it circular?
See also: IPGetObjects, GMVrtxListToCircOrLin,

9.2.378 IPSetPrintFunc (iritprs2.c:939)

files

```
IPPrintFuncType IPSetPrintFunc(IPPrintFuncType PrintFunc)
```

PrintFunc: A function that gets a single string it should print.

Returns: Old value of this state.

Description: Sets the printing function to call if needs to redirect printing.
See also: IPSetFloatFormat, IPPrintFunc,

9.2.379 IPSetProcessLeafFunc (iritprs1.c:1606)

files

parser

```
IPProcessLeafObjType IPSetProcessLeafFunc(IPProcessLeafObjType ProcessLeafFunc)
```

ProcessLeafFunc: A pointer to a call back function to be invoked on every leaf object read in.

Returns: Old call back pointer value.

Description: Sets a call back function on every leaf object read in.
See also: IPGetObjects,

9.2.380 IPSetPropagateAttrs (iritprs1.c:1521)

files

parser

```
int IPSetPropagateAttrs(int Propagate)
```

Propagate: If TRUE, attributes will be propagated from internal nodes to the leaves.

Returns: Old value of propagation state.

Description: Controls the propagation of attributes from internal nodes to the leaves.

9.2.381 IPSetReadOneObject (iritprs1.c:1578)

files

parser

```
int IPSetReadOneObject(int OneObject)
```

OneObject: If TRUE, only next object will be read by IPGetObjectst. If FALSE, objects will be read until EOF is detected and placed in a linked list.

Returns: Old value of read one object.

Description: Controls the way the Ascii/bin parser handle multiple objects in a file.
See also: IPGetObjects,

9.2.382 IPSetSubObjectName (linklist.c:1607)

```
void IPSetSubObjectName(IPObjectStruct *PListObj, int Index, const char *Name)
```

PListObj: A list object.

Index: Of object in PListObj to change its name.

Name: New name of sub object.

Returns: void

Description: Sets the name of object number Index in list object PListObj into Name.

9.2.383 IPSetVrmlExternalMode (iritvrml.c:616)

```
int IPSetVrmlExternalMode(int On)
```

On: SID mode is enabled if TRUE.

Returns: old value.

Description: Sets the mode of translating IRIT object tree into VRML graph. If TRUE then output file is suitable for External activation usage. Otherwise, it can be used for standalone viewing in VRML 2.0 browser. Default is TRUE.

9.2.384 IPSocClntInit (sockets.c:385)

```
int IPSocClntInit(void)
```

Returns: Handle to the socket stream, -1 if failed.

Description: Initialize the client's needs - builds the socket etc.

9.2.385 IPSocDisConnectAndKill (sockets.c:340)

```
int IPSocDisConnectAndKill(int Kill, int Handler)
```

Kill: If TRUE, send a KILL message to the client process.

Handler: The socket info handler. If IP_CLNT_BROADCAST_ALL_HANDLES do a broadcast write.

Returns: TRUE, if succesful, FALSE otherwise.

Description: Close, and optionally kill, io channels to another client process.

9.2.386 IPSocEchoInput (sockets.c:610)

ipc

```
void IPSocEchoInput(int Handler, int EchoInput)
```

Handler: The socket info handler index. If IP_CLNT_BROADCAST_ALL_HANDLES do a broadcast update.

EchoInput: TRUE to echo every character read in.

Returns: void

Description: Sets echo printing of read input.

9.2.387 IPSocExecAndConnect (sockets.c:267)

```
int IPSocExecAndConnect(const char *Program, int IsBinary)
```

Program: Name of program to execute. Name can be NULL, in which the user is prompt to execute the program manually.

IsBinary: If TRUE sets channels to binary, if FALSE channels are text. This is assuming no IRIT_BIN_IPC environment variable is set, when communication will always be binary.

Returns: Handle of client if succesful, -1 otherwise.

Description: Executes the given program and connect to its io ports. This function is typically called by a server that syncronically forks out a client.

9.2.388 IPSocHandleClientEvent (sock_aux.c:26)

```
void IPSocHandleClientEvent(int Handler, IPObjectStruct *PObj)
```

Handler: Client handler from which an event has been recieved.

PObj: NULL if a new client has connected, Object recieved from Client otherwise.

Returns: void

Description: Call back function of the server listening to clients.

9.2.389 IPSocReadCharNonBlock (sockets.c:644)

ipc

```
int IPSocReadCharNonBlock(int Handler)
```

Handler: The socket info handler index.

Returns: Read character or EOF if none found.

Description: Non blocking read of a single character. Returns EOF if no data is found.

9.2.390 IPSocReadLineNonBlock (sockets.c:786)

ipc

```
char *IPSocReadLineNonBlock(int Handler)
```

Handler: The socket info handler.

Returns: Read line, or NULL if unavailable.

Description: Non blocking read of a single line. Returns NULL if no line is available.

9.2.391 IPSocReadOneObject (sockets.c:835)

ipc

```
IPObjectStruct *IPSocReadOneObject(int Handler)
```

Handler: The socket info handler. *

Returns: An object if read one, NULL otherwise.

Description: Attempts to read (non blocking) an object from socket. If read is successful the object is returned, otherwise NULL is returned.

9.2.392 IPSocSrvrInit (sockets.c:94)

```
int IPSocSrvrInit(void)
```

Returns: TRUE if succesful, FALSE otherwise.

Description: Initialize the server's needs - builds the listening socket etc.

9.2.393 IPSocSrvrListen (sockets.c:210)

```
int IPSocSrvrListen(void)
```

Returns: FALSE if no new requests, TRUE otherwise.

Description: Listen to requests from clients. A non blocking function that samples all active clients for possible requests.

9.2.394 IPSocWriteBlock (sockets.c:515)

ipc

```
int IPSocWriteBlock(int Handler, void *Block, int BlockLen)
```

Handler: The socket info handler. If IP_CLNT_BROADCAST_ALL_HANDLES do a broadcast write.

Block: Block to write.

BlockLen: Length of block to write.

Returns: TRUE if write succesful, FALSE otherwise.

Description: Writes a single block of BlockLen bytes.

9.2.395 IPSocWriteOneObject (sockets.c:454)

ipc

```
void IPSocWriteOneObject(int Handler, IPObjectStruct *PObj)
```

Handler: The socket info handler. If IP_CLNT_BROADCAST_ALL_HANDLES do a broadcast write.

PObj: Object to write to the client's socket.

Returns: void

Description: Attempts to write an object to a socket.

9.2.396 IPStderrObject (iritprs2.c:85)

files

```
void IPStderrObject(const IPObjectStruct *PObj)
```

PObj: To be put out to stderr.

Returns: void

Description: Routine to print the data from given object into stderr.

9.2.397 IPStdoutObject (iritprs2.c:67)

files

```
void IPStdoutObject(const IPObjectStruct *PObj, int IsBinary)
```

PObj: To be put out to stdout.

IsBinary: Is this a binary file we should dump?

Returns: void

Description: Routine to print the data from given object into stdout.

9.2.398 IPSurface2CtlMesh (ip_cnvrt.c:713)

conversion

```
IPPolygonStruct *IPSurface2CtlMesh(const CagdSrfStruct *Srf)
```

Srf: To extract its control mesh as a polylines.

Returns: A polylines object representing Srf's control mesh.

Description: Routine to convert a single surface's control mesh into a polylines object.

9.2.399 IPSurface2KnotPolylines (ip_cnvrt.c:629)

conversion

approximation

```
IPPolygonStruct *IPSurface2KnotPolylines(const CagdSrfStruct *Srf,
                                         CagdRType TolSamples,
                                         SymbCrvApproxMethodType Method)
```

Srf: To approximate as a polyline.

TolSamples: Tolerance of approximation error (Method = 2) or Number of samples to compute on polyline (Method = 0, 1).

Method: 0 - TolSamples are set uniformly in parametric space, 1 - TolSamples are set optimally, considering the isocurve's curvature. 2 - TolSamples sets the maximum error allowed between the piecewise linear approximation and original curve.

Returns: A polyline object approximating Srf.

Description: Routine to convert a single surface into polylines along the surface knot lines with TolSamples samples or tolerance per isoline curve as a polyline object.
See also: IPCurve2Polylines,

9.2.400 IPSurface2Polygons (ip_cnvrt.c:1159)

approximation

conversion

```
IPPolygonStruct *IPSurface2Polygons(CagdSrfStruct *Srf,
                                    const CagdSrf2PlsInfoStrct *TessInfo)
```

Srf: To approximate using polygons.

TessInfo: Parameters to control the tessellation into polygons.

Returns: Resulting polygons that approximates Srf.

Description: Routine to approximate a single surface by polygons. The polygonal approximation routines have call back functions to invoke for each new polygon and if these call back functions are used, this function might return NULL. See IPGenTriangle and IPGenRectangle.
See also: CagdSrf2Polygons, CagdSrfAdap2Polygons, IPGenTriangle, IPGenRectangle, IPSurface2PolygonsGenTriOnly,

9.2.401 IPSurface2PolygonsGenDegenPolys (ip_cnvrt.c:1097)

triangles

rectangles

```
int IPSurface2PolygonsGenDegenPolys(int GenDegenPolys)
```

GenDegenPolys: TRUE for creating degenerated triangles and rectangles, FALSE otherwise.

Returns: Old value of flag.

Description: Controls the generation of degenerated triangles and rectangles, in the tessellation process.
See also: IPSurface2Polygons,

9.2.402 IPSurface2PolygonsGenTriOnly (ip_cnvrt.c:1068)

triangles

```
int IPSurface2PolygonsGenTriOnly(int OnlyTri)
```

OnlyTri: TRUE for triangles only, FALSE otherwise.

Returns: Old value of flag.

Description: Controls the generation of triangles only, in the tessellation process.
See also: IPSurface2Polygons,

9.2.403 IPSurface2Polylines (ip_cnvrt.c:506)

conversion

approximation

```
IPPolygonStruct *IPSurface2Polylines(const CagdSrfStruct *Srf,
                                     int NumOfIsolines[2],
                                     CagdRType TolSamples,
                                     SymbCrvApproxMethodType Method)
```

Srf: To approximate as a polyline.

NumOfIsolines: Number of isocurves to extract, in each direction.

TolSamples: Tolerance of approximation error (Method = 2) or Number of samples to compute on polyline (Method = 0, 1).

Method: 0 - TolSamples are set uniformly in parametric space, 1 - TolSamples are set optimally, considering the isocurve's curvature. 2 - TolSamples sets the maximum error allowed between the piecewise linear approximation and original curve.

Returns: A polyline object approximating Srf.

Description: Routine to convert a single surface into polylines with TolSamples samples or tolerance per isoline curve as a polyline object. If NumOfIsolines has negative value, its absolute value is heuristically used to derive a new NumOfIsolines number for it.
See also: IPCurve2Polylines,

9.2.404 IPSurfacesToCubicBzrCrvs (ip_cnvrt.c:2039)

conversion

approximation

```
CagdCrvStruct *IPSurfacesToCubicBzrCrvs(CagdSrfStruct *Srfs,
                                        IPPolygonStruct **CtlMeshes,
                                        CagdBType DrawSurface,
                                        CagdBType DrawMesh,
                                        int NumOfIsolines[2],
                                        CagdRType MaxArcLen)
```

Srfs: To approximate as cubic Bezier curves.

CtlMeshes: If we want control meshes as well (DrawMesh == TRUE) they will be placed herein.

DrawSurface: Do we want to draw the surfaces?

DrawMesh: Do we want to draw the control meshes?

NumOfIsolines: Number of isocurves to extract, in each direction.

MaxArcLen: Tolerance for cubic Bezier approximation. See function SymbApproxCrvAsBzrCubics.

Returns: The cubic Bezier approximation, or NULL if DrawSurface is FALSE.

Description: Approximates an arbitrary list of surfaces into cubic Beziers curves.
See also: IPSurfacesToCubicBzrSrfs,

9.2.405 IPSurfacesToCubicBzrSrfs (ip_cnvrt.c:2327)

conversion

approximation

```
CagdSrfStruct *IPSurfacesToCubicBzrSrfs(CagdSrfStruct *Srfs,
                                        CagdSrfStruct **NoConvertionSrfs)
```

Srfs: To approximate as cubic Bezier surfaces.

NoConvertionSrfs: List of surface that cannot be converted to bicubics.

Returns: The cubic Bezier approximation, or NULL if nothing has been converted.

Description: Converts an arbitrary list of surfaces into cubic integral Bezier surfaces, if possible. Any rational or surface with degrees higher than cubic cannot be converted and is left as is in the NoConvertionSrfs list.
See also: IPSurfacesToCubicBzrCrvs,

9.2.406 IPTSrf2PlyAuxSetPolyGenData (ip_cnvrt.c:2634)

```
CagdSrfAdapAuxDataFuncType
            IPTSrf2PlyAuxSetPolyGenData(const CagdSrf2PlsInfoStrct *TessInfo,
                                        CagdSrfAdapAuxDataFuncType Data)
```

TessInfo: Auxiliary struct which hold the parameters for the creation of the polygons.

Data: New data for used poly function, NULL use default.

Returns: Old value of function.

Description: Sets the call back data of the function to generate polygons.
See also: IPTSrf2PlyAuxSetRectFunc, IPTSrf2PlyAuxSetTriFunc,

9.2.407 IPTSrf2PlyAuxSetPolyGenFunc (ip_cnvrt.c:2669)

```
CagdSrfAdapPolyGenFuncType
           IPTSrf2PlyAuxSetPolyGenFunc(const CagdSrf2PlsInfoStrct *TessInfo,
                                       CagdSrfAdapPolyGenFuncType Func)
```

TessInfo: Auxiliary struct which hold the parameters for the creation of the polygons.

Func: New function to use, NULL use default.

Returns: Old value of function.

Description: Sets the call back function to generate polygons.
See also: IPTSrf2PlyAuxSetRectFunc, IPTSrf2PlyAuxSetTriFunc, , IPTSrf2PlyAuxSetPolyGenData,

9.2.408 IPTSrf2PlyAuxSetRectFunc (ip_cnvrt.c:2600)

```
CagdSrfMakeRectFuncType
               IPTSrf2PlyAuxSetRectFunc(const CagdSrf2PlsInfoStrct *TessInfo,
                                        CagdSrfMakeRectFuncType Func)
```

TessInfo: Auxiliary struct which hold the parameters for the creation of the polygons.

Func: New function to use, NULL use default.

Returns: Old value of function.

Description: Sets the call back function to generate rectangles. The function will be invoked with each rectangle in the polygonal approximation.
See also: IPTSrf2PlyAuxSetTriFunc,

9.2.409 IPTSrf2PlyAuxSetTriFunc (ip_cnvrt.c:2565)

```
CagdSrfMakeTriFuncType
                IPTSrf2PlyAuxSetTriFunc(const CagdSrf2PlsInfoStrct *TessInfo,
                                        CagdSrfMakeTriFuncType Func)
```

TessInfo: Auxiliary struct which hold the parameters for the creation of the polygons.

Func: New function to use, NULL use default.

Returns: Old value of function.

Description: Sets the call back function to generate triangles. The function will be invoked with each triangle in the polygonal approximation.
See also: IPTSrf2PlyAuxSetRectFunc,

9.2.410 IPTSrf2PlysFreeTessInfo (ip_cnvrt.c:2699)

```
void IPTSrf2PlysFreeTessInfo(CagdSrf2PlsInfoStrct *TessInfo)
```

TessInfo: The struct to free its content.

Returns: void

Description: Free the parameters of the srf2ply variables in struct TessInfo.
See also: IPTrimSrf2Polygons, TrimSrf2Polygons2, CagdSrf2PlsInfoStrct,

9.2.411 IPTSrf2PlysInitTessInfo (ip_cnvrt.c:2480)

```
CagdSrf2PlsInfoStrct *IPTSrf2PlysInitTessInfo(
                                        CagdSrf2PlsInfoStrct *TessInfo,
                                        int FourPerFlat,
                                        IrtRType FineNess,
                                        int ComputeUV,
                                        int ComputeNrml,
                                        int Optimal,
                                        void *EvalNrmlCache,
                                        CagdSrfMakeRectFuncType MakeRectangle,
                                        CagdSrfMakeTriFuncType MakeTriangle,
                                        CagdSrfAdapPolyGenFuncType
                                                          AdapPolyGenFunc,
                                        CagdSrfAdapAuxDataFuncType
                                                          AdapAuxDataFunc)
```

TessInfo: The struct to update,in place.

FourPerFlat: TRUE to generate four polys if surface flat enough, FALSE to generate two polys.

FineNess: Fineness of polygonal approximation to use.

ComputeUV: TRUE to also generate UV coordinates, FALSE to ignore.

ComputeNrml: TRUE to also generate normals, FALSE to ignore.

Optimal: TRUE to use optimal tessellation, FALSE for uniform samp.

EvalNrmlCache: Cache for the normal evaluations.

MakeRectangle: Generator function for rectangles. NULL for default.

MakeTriangle: Generator function for triangles. NULL for default.

AdapPolyGenFunc: Generating function for polygons. NULL for default.

AdapAuxDataFunc: Aux. processing func during subdiv. NULL for default.

Returns: contains the preferences as selected by the and user. A reference to given TessInfo.

Description: Sets the parameters of the srf2ply variables in struct TessInfo.
See also: IPTrimSrf2Polygons, TrimSrf2Polygons2, CagdSrf2PlsInfoStrct, , IPTSrf2PlysFreeTessInfo2,

9.2.412 IPTSrf2PlysInitTessInfo2 (ip_cnvrt.c:2537)

```
CagdSrf2PlsInfoStrct *IPTSrf2PlysInitTessInfo2(CagdSrf2PlsInfoStrct *TessInfo,
                                               int FourPerFlat,
                                               IrtRType FineNess,
                                               int ComputeUV,
                                               int ComputeNrml,
                                               int Optimal,
                                               void *EvalNrmlCache)
```

TessInfo: The struct to update,in place.

FourPerFlat: TRUE to generate four polys if surface flat enough, FALSE to generate two polys.

FineNess: Fineness of polygonal approximation to use.

ComputeUV: TRUE to also generate UV coordinates, FALSE to ignore.

ComputeNrml: TRUE to also generate normals, FALSE to ignore.

Optimal: TRUE to use optimal tessellation, FALSE for uniform samp.

EvalNrmlCache: Cache for the normal evaluations.

Returns: contains the preferences as selected by the and user. A reference to given TessInfo.

Description: Sets the parameters of the srf2ply variables in struct TessInfo.
See also: IPTrimSrf2Polygons, TrimSrf2Polygons2, CagdSrf2PlsInfoStrct, , IPTSrf2PlysFreeTessInfo,

9.2.413 IPTraverseObjHierarchy (linklist.c:1872)

hierarchy traversal

```
void IPTraverseObjHierarchy(IPObjectStruct *PObj,
                            IPObjectStruct *PObjList,
                            void *Data,
                            IPTraverseObjHierarchyStruct *TraversState)
```

PObj: To traverse and apply.

PObjList: DB to search instance of objects by name.

Data: Optional auxiliary data to be passed to all invocations.

TraversState: State of type/behvious of traversal desired.

Returns: void

Description: Auxiliary function of IPTraverseObjListHierarchy Instances are converted into their real objects on the fly. Objects that have an "Invisible" attribute are potentially ignored (they might be invoked indirectly as an instance).
See also: IPTraverseObjListHierarchy, IPTraverseObjectAll, IPTraverseObjectCopy, IPTraverseInvisibleObject,

9.2.414 IPTraverseObjHierarchy1 (linklist.c:1706)

hierarchy traversal

```
void IPTraverseObjHierarchy1(IPObjectStruct *PObj,
                             void *Data,
                             IPTraverseObjHierarchyStruct *TraversState)
```

PObj: To traverse and apply.

Data: Optional auxiliary data to be passed to all invocations.

TraversState: The state variables of the traversal.

Returns: void

Description: Auxiliary function of IPTraverseObjListHierarchy Instances are converted into their real objects on the fly. Objects that have an "Invisible" attribute are potentially ignored (they might be invoked indirectly as an instance).
See also: IPTraverseObjListHierarchy, IPTraverseObjectAll, IPTraverseObjectCopy, IPTraverseInvisibleObject,

9.2.415 IPTraverseObjHierarchyInitState (linklist.c:1797)

hierarchy traversal

```
void IPTraverseObjHierarchyInitState(IPTraverseObjHierarchyStruct
                                                          *TraversState)
```

TraversState: To initialize.

Returns: void

Description: Initializes the state of the given traversal struct, to default values.
See also: IPTraverseObjHierarchy, IPTraverseObjectAll, IPTraverseObjectCopy, IPTraverseInvisibleObject, IPTraverseObjListHierarchy, , IPTraverseObjListHierarchy1, IPTraverseObjListHierarchy2, , IPTraverseObjListHierarchy3,

9.2.416 IPTraverseObjListHierarchy (linklist.c:1640)

hierarchy traversal

```
void IPTraverseObjListHierarchy(IPObjectStruct *PObjList,
                                IPTraverseObjHierarchyStruct *TraversState)
```

PObjList: To traverse and apply.

TraversState: The state variables of the traversal.

Returns: void

Description: Traverses a hierarchy of objects and invokes ApplyObject to each leaf (non list) object in the given object list with an associated matrix. Instances are converted into their real objects on the fly. Objects that have an "Invisible" attribute are potentially ignored (they might be invoked indirectly as an instance).
See also: IPTraverseObjHierarchy, IPTraverseObjectAll, IPTraverseObjectCopy, IPTraverseInvisibleObject, IPTraverseObjListHierarchy2,

9.2.417 IPTraverseObjListHierarchy1 (linklist.c:1674)

hierarchy traversal

```
void IPTraverseObjListHierarchy1(IPObjectStruct *PObjList,
                                 void *Data,
                                 IPTraverseObjHierarchyStruct *TraversState)
```

PObjList: To traverse and apply.

Data: Optional auxiliary data to be passed to all invocations.

TraversState: The state variables of the traversal.

Returns: void

Description: Traverses a hierarchy of objects and invokes ApplyObject to each leaf (non list) object in the given object list with an associated matrix. Instances are converted into their real objects on the fly. Objects that have an "Invisible" attribute are potentially ignored (they might be invoked indirectly as an instance).
See also: IPTraverseObjHierarchy, IPTraverseObjectAll, IPTraverseObjectCopy, IPTraverseInvisibleObject, IPTraverseObjListHierarchy, , IPTraverseObjListHierarchy2, IPTraverseObjListHierarchy3,

9.2.418 IPTraverseObjListHierarchy2 (linklist.c:1737)

hierarchy traversal

```
void IPTraverseObjListHierarchy2(IPObjectStruct *PObjList,
                                 void *Data,
                                 IPTraverseObjHierarchyStruct *TraversState)
```

PObjList: To traverse and apply.

Data: Optional auxiliary data to be passed to all invocations.

TraversState: The state variables of the traversal.

Returns: void

Description: Traverses a hierarchy of objects and invokes ApplyObject to each leaf (non list) object in the given object list with an associated matrix. Instances are converted into their real objects on the fly. Objects that have an "Invisible" attribute are potentially ignored (they might be invoked indirectly as an instance).
See also: IPTraverseObjHierarchy, IPTraverseObjectAll, IPTraverseObjectCopy, IPTraverseInvisibleObject, IPTraverseObjListHierarchy, IPTraverseObjListHierarchy1, IPTraverseObjListHierarchy3,

9.2.419 IPTraverseObjListHierarchy3 (linklist.c:1826)

hierarchy traversal

```
IPObjectStruct **IPTraverseObjListHierarchy3(IPObjectStruct *PObjList)
```

PObjList: To traverse and create references for. Not modified.

Returns: The vector of references, allocated dynamically.

Description: Traverses a hierarchy of objects and convert to a vector of references over all leave (non list) objects in the given object list. Instances are converted into their real objects on the fly. Objects that have an "Invisible" attribute are potentially ignored (they might be invoked indirectly as an instance).
See also: IPTraverseObjHierarchy, IPTraverseObjectAll, IPTraverseObjectCopy, IPTraverseInvisibleObject, IPTraverseObjListHierarchy, , IPTraverseObjListHierarchy1, IPTraverseObjListHierarchy2,

9.2.420 IPTriSrf2CtlMesh (ip_cnvrt.c:1908)

conversion

```
IPPolygonStruct *IPTriSrf2CtlMesh(TrngTriangSrfStruct *TriSrf)
```

TriSrf: To extract its control mesh as a polylines.

Returns: A polylines object representing TriSrf's control mesh.

Description: Routine to convert a single triangular patch's control mesh into a polylines object.

9.2.421 IPTriSrf2Polygons (ip_cnvrt.c:1820)

approximation

conversion

```
IPPolygonStruct *IPTriSrf2Polygons(TrngTriangSrfStruct *TriSrf,
                                   const CagdSrf2PlsInfoStrct *TessInfo)
```

TriSrf: To approximate using polygons.

TessInfo: Parameters to control the tessellation into polygons.

Returns: Resulting polygons that approximates Srf.

Description: Routine to approximate a single triangular patch by polygons.

9.2.422 IPTriSrf2Polylines (ip_cnvrt.c:1868)

conversion

approximation

```
IPPolygonStruct *IPTriSrf2Polylines(TrngTriangSrfStruct *TriSrf,
                                    int NumOfIsolines[3],
                                    CagdRType TolSamples,
                                    SymbCrvApproxMethodType Method)
```

TriSrf: To approximate as a polyline.

NumOfIsolines: Number of isocurves to extract, in each direction.

TolSamples: Tolerance of approximation error (Method = 2) or Number of samples to compute on polyline (Method = 0, 1).

Method: 0 - TolSamples are set uniformly in parametric space, 1 - TolSamples are set optimally, considering the curve's curvature. 2 - TolSamples sets the maximum error allowed between the piecewise linear approximation and original curve.

Returns: A polylines object approximating TriSrf.

Description: Routine to convert a single triangular patch function into polylines with SamplesPerCurve samples, NumOfIsolines isolines into a polylines object. If NumOfIsolines has negative value, its absolute value is heuristically used to derive a new NumOfIsolines number for it.

9.2.423 IPTriSrfsToCubicBzrCrvs (ip_cnvrt.c:2271)

conversion

approximation

```
CagdCrvStruct *IPTriSrfsToCubicBzrCrvs(TrngTriangSrfStruct *TriSrfs,
                                       IPPolygonStruct **CtlMeshes,
                                       CagdBType DrawSurface,
                                       CagdBType DrawMesh,
                                       int NumOfIsolines[3],
                                       CagdRType MaxArcLen)
```

TriSrfs: To approximate as cubic Bezier curves.

CtlMeshes: If we want control meshes as well (DrawMesh == TRUE) they will be placed herein.

DrawSurface: Do we want to draw the surfaces?

DrawMesh: Do we want to draw the control meshes?

NumOfIsolines: Number of isocurves to extract, in each direction.

MaxArcLen: Tolerance for cubic Bezier approximation. See function SymbApproxCrvAsBzrCubics.

Returns: The cubic Bezier approximation, or NULL if DrawSurface is FALSE.

Description: Approximates an arbitrary list of triangular surfaces into cubic Beziers curves.

9.2.424 IPTrimSrf2CtlMesh (ip_cnvrt.c:1431)

conversion

```
IPPolygonStruct *IPTrimSrf2CtlMesh(TrimSrfStruct *TrimSrf)
```

TrimSrf: To extract its control mesh as a polylines.

Returns: A polylines object representing TrimSrf's control mesh.

Description: Routine to convert a single trimmed surface's control mesh into a polylines object.

9.2.425 IPTrimSrf2Polygons (ip_cnvrt.c:1237)

approximation

conversion

```
IPPolygonStruct *IPTrimSrf2Polygons(const TrimSrfStruct *TrimSrf,
                                    const CagdSrf2PlsInfoStrct *TessInfo)
```

TrimSrf: To approximate using polygons.

TessInfo: Parameters to control the tessellation into polygons.

Returns: Resulting polygons that approximates TrimSrf.

Description: Routine to approximate a single trimmed surface by polygons. The polygonal approximation routines have call back functions to invoke for each new polygon and if these call back functions are used, this function might return NULL. See IPGenTriangle and IPGenRectangle.
See also: TrimSrf2Polygons2, IPGenTriangle, IPGenRectangle,

9.2.426 IPTrimSrf2Polylines (ip_cnvrt.c:1357)

conversion

approximation

```
IPPolygonStruct *IPTrimSrf2Polylines(TrimSrfStruct *TrimSrf,
                                     int NumOfIsolines[2],
                                     CagdRType TolSamples,
                                     SymbCrvApproxMethodType Method,
                                     int TrimmingCurves,
                                     int IsoParamCurves)
```

TrimSrf: To approximate as a polyline.

NumOfIsolines: Number of isocurves to extract, in each direction.

TolSamples: Tolerance of approximation error (Method = 2) or Number of samples to compute on polyline (Method = 0, 1).

Method: 0 - TolSamples are set uniformly in parametric space, 1 - TolSamples are set optimally, considering the curve's curvature. 2 - TolSamples sets the maximum error allowed between the piecewise linear approximation and original curve.

TrimmingCurves: Do we want the trimming curves as well.

IsoParamCurves: Do we want trimmed isoparametric curves.

Returns: A polylines object approximating TrimSrf.

Description: Routine to convert a single trimmed surface into polylines with TolSamples samples/tolerance, NumOfIsolines isolines into a polylines object. If NumOfIsolines has negative value, its absolute value is heuristically used to derive a new NumOfIsolines number for it.

9.2.427 IPTrimSrfsToCubicBzrCrvs (ip_cnvrt.c:2100)

conversion

approximation

```
CagdCrvStruct *IPTrimSrfsToCubicBzrCrvs(TrimSrfStruct *TrimSrfs,
                                        IPPolygonStruct **CtlMeshes,
                                        CagdBType DrawTrimSrf,
                                        CagdBType DrawMesh,
                                        int NumOfIsolines[2],
                                        CagdRType MaxArcLen)
```

TrimSrfs: To approximate as cubic Bezier curves.

CtlMeshes: If we want control meshes as well (DrawMesh == TRUE) they will be placed herein.

DrawTrimSrf: Do we want to draw the surfaces?

DrawMesh: Do we want to draw the control meshes?

NumOfIsolines: Number of isocurves to extract, in each direction.

MaxArcLen: Tolerance for cubic Bezier approximation. See function SymbApproxCrvAsBzrCubics.

Returns: The cubic Bezier approximation, or NULL if DrawSurface is FALSE.

Description: Approximates an arbitrary list of trimmed surfaces into cubic Beziers curves. Only isoparametric curves are extarcted (no trimming curves).

9.2.428 IPTrivar2CtlMesh (ip_cnvrt.c:1797)

conversion

```
IPPolygonStruct *IPTrivar2CtlMesh(TrivTVStruct *Trivar)
```

Trivar: To extract its control mesh as a polylines.

Returns: A polylines object representing Trivar's control mesh.

Description: Routine to convert a single trivariate's control mesh into a polylines object.

9.2.429 IPTrivar2Polygons (ip_cnvrt.c:1456)

approximation

conversion

```
IPPolygonStruct *IPTrivar2Polygons(TrivTVStruct *Trivar,
                                   const CagdSrf2PlsInfoStrct *TessInfo)
```

Trivar: To approximate using polygons.

TessInfo: Parameters to control the tessellation into polygons.

Returns: Resulting polygons that approximates Srf.

Description: Routine to approximate a single trivariate by polygons. Six faces of the trivariate are extracted as six surfaces that are displayed. Trivariate can have C0 discontinuities at this time.

9.2.430 IPTrivar2Polylines (ip_cnvrt.c:1707)

conversion

approximation

```
IPPolygonStruct *IPTrivar2Polylines(TrivTVStruct *Trivar,
                                    int NumOfIsolines[3],
                                    CagdRType TolSamples,
                                    SymbCrvApproxMethodType Method,
                                    IrtBType SrfsOnly)
```

Trivar: To approximate as a polyline.

NumOfIsolines: Number of isocurves to extract, in each direction.

TolSamples: Tolerance of approximation error (Method = 2) or Number of samples to compute on polyline (Method = 0, 1).

Method: 0 - TolSamples are set uniformly in parametric space, 1 - TolSamples are set optimally, considering the curve's curvature. 2 - TolSamples sets the maximum error allowed between the piecewise linear approximation and original curve.

SrfsOnly: If TRUE, isolines of only the 6 surface faces of the trivariate are synthesized.

Returns: A polylines object approximating Trivar.

Description: Routine to convert a single trivariate function into polylines with TolSamples/Tolerance per isocurve, NumOfIsolines isolines into a polylines object. If NumOfIsolines has negative value, its absolute value is heuristically used to derive a new NumOfIsolines number for it.

9.2.431 IPTrivarToCubicBzrCrvs (ip_cnvrt.c:2161)

conversion

approximation

```
CagdCrvStruct *IPTrivarToCubicBzrCrvs(TrivTVStruct *Trivar,
                                      IPPolygonStruct **CtlMeshes,
                                      CagdBType DrawTrivar,
                                      CagdBType DrawMesh,
                                      int NumOfIsolines[2],
                                      CagdRType MaxArcLen)
```

Trivar: To approximate as a set of cubic bezier curves.

CtlMeshes: If we want control meshes as well (DrawMesh == TRUE) they will be placed herein.

DrawTrivar: Do we want to draw the trivariate function?

DrawMesh: Do we want to draw the control meshes?

NumOfIsolines: Number of isocurves to extract, in each direction.

MaxArcLen: Tolerance for cubic Bezier approximation. See function SymbApproxCrvAsBzrCubics.

Returns: A list of curves approximating Trivar.

Description: Routine to convert a single trivariate function into cubic Bezier curves.

9.2.432 IPUpdatePolyPlane (prsrgeom.c:94)

files

parser

```
int IPUpdatePolyPlane(IPPolygonStruct *PPoly)
```

PPoly: To update its normal/plane equation.

Returns: TRUE if successful, FALSE otherwise.

Description: Routine to update the Plane equation of the given polygon by the order of the most robust three vertices of that polygon to define the normal.

9.2.433 IPUpdatePolyPlane2 (prsrgeom.c:159)

files

parser

```
int IPUpdatePolyPlane2(IPPolygonStruct *PPoly, const IrtVecType Vin)
```

PPoly: To update its normal/plane equation.

Vin: A vertex to be considered in the inside, respective to PPoly.

Returns: 0 if failed, 1 if successful, -1 if successful but vertices reversed.

Description: Routine to update the Plane equation of the given polygon such that the Vin vertex will be in the positive side of it.

9.2.434 IPUpdateVrtxNrml (prsrgeom.c:193)

files

parser

```
void IPUpdateVrtxNrml(IPPolygonStruct *PPoly, IrtVecType DefNrml)
```

PPoly: Polygon to update normal information.

DefNrml: Normal tp use in update.

Returns: void

Description: Routine to update all vertices in polygon to hold a default normal if have none already.

9.2.435 IPVrtxListLen (linklist.c:1220)

length

linked lists

```
int IPVrtxListLen(const IPVertexStruct *V)
```

V: Vertex list to compute its length.

Returns: Number of elements in V list.

Description: Returns the length of a list of vertices.

9.2.436 Iges2IritWarning (igs_irit.c:5049)

```
void Iges2IritWarning(IgesInfoStruct *IgesInfo, int SeqNum, char *va_alist, ...)
```

IgesInfo: All information one ever needs to process IGES file.

SeqNum: Where the error occurred, 0 to specify line numbers.

va_alist: Do "man stdarg"

Returns: void

Description: Print warning messages.

9.2.437 IpcCompressObj (iritprsc.c:322)

files
parser
predictor
compress
quantization.

```
int IpcCompressObj(int Handler, const IPObjectStruct *PObj)
```

Handler: A handler to the open stream.

PObj: Irit format objects list.

Returns: Internal Error code.

Description: Compress a given IPObject to a file using Handler.
See also: IpcDecompressObj.,

9.2.438 IpcCompressObjToFile (iritprsc.c:285)

files
parser
predictor
compress
quantization.

```
int IpcCompressObjToFile(const char *FileName,
                         const IPObjectStruct *PObj,
                         float QntError)
```

FileName: Name of a compression file to save in.

PObj: Irit format objects list.

QntError: Quantization step between(0..1). Specifies maximum error for values. IPC_QUANTIZATION_NONE - no quanization is used.

Returns: Internal Error code.

Description: Compress a given IPObject to a file.
See also: IpcDecompressObjFromFile.,

9.2.439 IpcDecompressObj (iritprsd.c:246)

files
parser
uncompress

```
IPObjectStruct *IpcDecompressObj(int Handler)
```

Handler: A handler to the open stream.

Returns: A list of the Irit objects.

Description: Decompress the Irit objects from a compressed file using Handler to an IPObjectStructs list.
See also: IpcDecompressObjFromFile.,

9.2.440 IpcDecompressObjFromFile (iritprsd.c:222)

files
parser
uncompress

```
IPObjectStruct *IpcDecompressObjFromFile(const char *FileName)
```

FileName: A compression file to load from compressed data.

Returns: A list of the Irit objects.

Description: Decompress the Irit objects from a compressed file using Handler to an IPObjectStructs list.
See also: IpcDecompressObjFromFile.,

9.2.441 IpcSetQuantization (iritprsd.c:200)

Quantization
Handle.

```
void IpcSetQuantization(int Handler, float QntError)
```

Handler: A handler to the open stream.

QntError: Quantization error. Safe range is [0.00001 - 0.0000001]. IPC_QUANTIZATION_NONE = Quantization is not used.

Returns: void

Description: Set quantization error for specific Handler.
See also: IpcDecompressObjFromFile.,

9.2.442 Irit2WglAddPoly (irit_wgl.c:778)

Poly
PolyIndexArray
AddPoly

```
void Irit2WglAddPoly(Irit2WglSceneStruct *Scene,
                     Irit2WglModelObjectStruct *ModelObject,
                     int *PolyIndexArray)
```

Scene: Scene.

ModelObject: Model object.

PolyIndexArray: Array of vertex indices defining the poly.

Returns: void

Description: Create new poly to the given model object.

9.2.443 Irit2WglAddVertex (irit_wgl.c:733)

Vertex
Pos
Normal
U

```
void Irit2WglAddVertex(Irit2WglSceneStruct *Scene,
                       Irit2WglModelObjectStruct *ModelObject,
                       IrtPtType Pos,
                       IrtNrmlType Normal,
                       IrtUVType UV)
```

Scene: Scene.

ModelObject: Model object.

Pos: Vertex position.

Normal: Vertex normal.

UV: Vertex UV coordinates.

Returns: void

Description: Add new vertex to the given model object.

9.2.444 Irit2WglDumpCSS (irit_wgl.c:1455)

CSS

```
void Irit2WglDumpCSS(Irit2WglSceneStruct *Scene,
                     const char *OutputFileName)
```

Scene: Scene.

OutputFileName: Output file name.

Returns: void

Description: Dump CSS script related data.

9.2.445 Irit2WglDumpData (irit_wgl.c:875)

DumpData

```
void Irit2WglDumpData(Irit2WglSceneStruct *Scene, const char *OutputFileName)
```

Scene: Scene.

OutputFileName: Output file name.

Returns: void

Description: Dump WebGL HTML data.

9.2.446 Irit2WglDumpJS (irit_wgl.c:984)

JS

```
void Irit2WglDumpJS(Irit2WglSceneStruct *Scene, const char *OutputFileName)
```

Scene: Scene.

OutputFileName: Output file name.

Returns: void

Description: Dump JS script related data.

9.2.447 Irit2WglDumpJSInitCameraMatrices (irit_wgl.c:1212)

CameraMatrices

```
void Irit2WglDumpJSInitCameraMatrices(Irit2WglSceneStruct *Scene,
                                      FILE *JSOutputFile)
```

Scene: Scene.

JSOutputFile: JS Output file.

Returns: void

Description: Dump JS script related data - Camera matrices.

9.2.448 Irit2WglDumpJSSetControlBarParams (irit_wgl.c:1328)

ControlBar

```
void Irit2WglDumpJSSetControlBarParams(Irit2WglSceneStruct *Scene,
                                       FILE *JSOutputFile)
```

Scene: Scene.

JSOutputFile: JS Output file.

Returns: void

Description: Dump JS script related data - Control bar parameters.

9.2.449 Irit2WglDumpJSSetLightSources (irit_wgl.c:1252)

LightSources

```
void Irit2WglDumpJSSetLightSources(Irit2WglSceneStruct *Scene,
                                   FILE *JSOutputFile)
```

Scene: Scene.

JSOutputFile: JS Output file.

Returns: void

Description: Dump JS script related data - Light sources setting.

9.2.450 Irit2WglDumpJSSetMenuParams (irit_wgl.c:1356)

Menu

```
void Irit2WglDumpJSSetMenuParams(Irit2WglSceneStruct *Scene,
                                 FILE *JSOutputFile)
```

Scene: Scene.

JSOutputFile: JS Output file.

Returns: void

Description: Dump JS script related data - Menu parameters.

9.2.451 Irit2WglDumpJSSetModelData (irit_wgl.c:1101)

ModelData

```
void Irit2WglDumpJSSetModelData(Irit2WglSceneStruct *Scene,
                                FILE *JSOutputFile)
```

Scene: Scene.

JSOutputFile: JS Output file.

Returns: void

Description: Dump JS script related data - Model data.

9.2.452 Irit2WglDumpJSSetStatusLogParams (irit_wgl.c:1425)

StatusLog

```
void Irit2WglDumpJSSetStatusLogParams(Irit2WglSceneStruct *Scene,
                                      FILE *JSOutputFile)
```

Scene: Scene.

JSOutputFile: JS Output file.

Returns: void

Description: Dump JS script related data - Status log parameters.

9.2.453 Irit2WglDumpJSSetTextureData (irit_wgl.c:1067)

TextureData

```
void Irit2WglDumpJSSetTextureData(Irit2WglSceneStruct *Scene,
                                  FILE *JSOutputFile)
```

Scene: Scene.

JSOutputFile: JS Output file.

Returns: void

Description: Dump JS script related data - Texture data.

9.2.454 Irit2WglDumpScripts (irit_wgl.c:936)

Scripts

DumpSctipts

```
void Irit2WglDumpScripts(Irit2WglSceneStruct *Scene,
                         FILE* HtmlOutputFile,
                         const char *OutputFileName)
```

Scene: Scene.

HtmlOutputFile: HTML Output file.

OutputFileName: Output file name.

Returns: void

Description: Dump WebGL HTML scripts related data.

9.2.455 Irit2WglFatalError (irit_wgl.c:1588)

```
void Irit2WglFatalError(const char *FatalErrorMsg)
```

FatalErrorMsg: Fatal error message.

Returns: void

Description: Trap Irit2Wgl errors right here. Gets an error description, print it and exit the program using exit.

9.2.456 Irit2WglFreeScene (irit_wgl.c:562)

Free

```
void Irit2WglFreeScene(Irit2WglSceneStruct *Scene)
```

Scene: Scene.

Returns: void

Description: Free the given scene.

9.2.457 Irit2WglModelObjectTagType2Str (irit_wgl.c:1562)

ObjTag

```
const char *Irit2WglModelObjectTagType2Str(Irit2WglModelObjectTagType ModelObjectTag)
```

ModelObjectTag: Model object tag.

Returns: Object tag string.

Description: Convert model object tag enum to string.

9.2.458 Irit2WglNewModelObject (irit_wgl.c:635)

NewModelObject

```
Irit2WglModelObjectStruct *Irit2WglNewModelObject(Irit2WglSceneStruct *Scene,
                                                  const char *Name,
                                                  Irit2WglModelObjectTagType
                                                              ModelObjectTag,
                                                  IrtRType RGBA[4],
                                                  int MaxObjVertices,
                                                  const char *PTextureFileName,
                                                  IrtUVType SUV)
```

Scene: Scene.

Name: Model object's name.

ModelObjectTag: Model object tag.

RGBA: RGBA values.

MaxObjVertices: Maximum number of vertices in object.

PTextureFileName: Texture file name.

SUV: UV texture scaling.

Returns: New model object.

Description: Create new model object.

9.2.459 Irit2WglNewScene (irit_wgl.c:506)

NewScene

```
Irit2WglSceneStruct *Irit2WglNewScene(void)
```

Returns: New scene.

Description: Create new scene.

9.2.460 Irit2WglProjectionModeType2Str (irit_wgl.c:1537)

ProjMode

```
const char *Irit2WglProjectionModeType2Str(Irit2WglProjectionModeType ProjectionMode)
```

ProjectionMode: Projection mode.

Returns: Projection mode string.

Description: Convert projection mode enum to string.

9.2.461 Irit2WglSetLightSource (irit_wgl.c:836)

LightSource
LightPos
LightColor

```
void Irit2WglSetLightSource(Irit2WglSceneStruct *Scene,
                            IGLightType LightPos,
                            IrtVecType LightColor)
```

Scene: Scene.

LightPos: Light source position.

LightColor: Light source diffuse color.

Returns: void

Description: Set light source data for the given scene.

9.2.462 Irit2WglViewAngleType2Str (irit_wgl.c:1502)

ViewAngle

```
const char *Irit2WglViewAngleType2Str(Irit2WglViewAngleType ViewAngle)
```

ViewAngle: View angle.

Returns: View angle string.

Description: Convert view angle enum to string.

9.2.463 IritPrsrFatalError (prsr_ftl.c:57)

error handling

```
void IritPrsrFatalError(IritPrsrFatalErrorType ErrID)
```

ErrID: Error type that was raised.

Returns: void

Description: Trap Prsr_lib errors right here. Provides a default error handler for the prsr library. Gets an error description using PrsrDescribeError, prints it and exit the program using exit.

9.2.464 MdlReadModelFromFile (mdl_read.c:44)

files
read
Mdlmed surfaces

```
MdlModelStruct *MdlReadModelFromFile(const char *FileName,
                                     char **ErrStr,
                                     int *ErrLine)
```

FileName: To read the model from.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read model surface, or NULL if an error occurred.

Description: Reads from a file a model. If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occurred in file. If no error is detected *ErrStr = NULL.

See also: MdlReadModelSrfFromFile2,

9.2.465 MdlReadModelFromFile2 (mdl_read.c:107)

files
read
model

```
MdlModelStruct *MdlReadModelFromFile2(int Handler,
                                      CagdBType NameWasRead,
                                      char **ErrStr,
                                      int *ErrLine)
```

Handler: A handler to the open stream.

NameWasRead: If FALSE, also reads the MODEL prefix.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read model, or NULL if an error occurred.

Description: Reads from a file a model. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occurred in file relative to model. Assumes the [MODEL prefix was read if NameWasRead. If no error is detected *ErrStr is set to NULL.
See also: MdlReadModelSrfFromFile,

9.2.466 MdlWriteModelToFile (mdl_wrt.c:35)

files
write

```
int MdlWriteModelToFile(const MdlModelStruct *Models,
                        const char *FileName,
                        int Indent,
                        const char *Comment,
                        char **ErrStr)
```

Models: To be saved in stream.

FileName: File name where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write model(s) to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.467 MdlWriteModelToFile2 (mdl_wrt.c:76)

files
write
stream

```
int MdlWriteModelToFile2(const MdlModelStruct *Models,
                         int Handler,
                         int Indent,
                         const char *Comment,
                         char **ErrStr)
```

Models: To be saved in stream.

Handler: A handler to the open stream.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write models(s) to the given stream. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.468 MdlWriteModelToFile3 (mdl_wrt.c:201)

files

write

```
int MdlWriteModelToFile3(const MdlModelStruct *Models,
                         FILE *f,
                         int Indent,
                         const char *Comment,
                         char **ErrStr)
```

Models: To be saved in stream.

f: File descriptor where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write model(s) to the given stream. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.469 MvarBspMVReadFromFile (mvarread.c:366)

files

read

multi-variates

```
MvarMVStruct *MvarBspMVReadFromFile(const char *FileName,
                                    char **ErrStr,
                                    int *ErrLine)
```

FileName: To read the multi-variate from.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read multi-variate, or NULL if an error occurred.

Description: Reads from a file B-spline multi-variates. If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occurred in file. If no error is detected *ErrStr = NULL.

9.2.470 MvarBspMVReadFromFile2 (mvarread.c:431)

files

read

multi-variates

```
MvarMVStruct *MvarBspMVReadFromFile2(int Handler,
                                     CagdBType NameWasRead,
                                     char **ErrStr,
                                     int *ErrLine)
```

Handler: A handler to the open stream.

NameWasRead: If FALSE, also reads the MULTIVAR BEZIER prefix.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read multi-variate, or NULL if an error occurred.

Description: Reads from a file a B-spline multi-variate. If NameWasRead is TRUE, it is assumed prefix "[MULTIVAR BSPLINE" has already been read. This is useful for a global parser which invokes this routine to read from a file several times as a parent controller. For exactly this reason, the given file descriptor is NOT closed in the end. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occurred in file relative to beginning of multi-variate. If no error is detected *ErrStr is set to NULL.

9.2.471 MvarBspMVWriteToFile (mvar_wrt.c:269)

files
write
multi-variates

```
int MvarBspMVWriteToFile(const MvarMVStruct *MVs,
                         const char *FileName,
                         int Indent,
                         const char *Comment,
                         char **ErrStr)
```

MVs: To be saved in file f.

FileName: File name where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if successful, FALSE otherwise.

Description: Generic routine to write B-spline multi-variates to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.472 MvarBspMVWriteToFile2 (mvar_wrt.c:310)

files
write
multi-variates

```
int MvarBspMVWriteToFile2(const MvarMVStruct *MVs,
                          int Handler,
                          int Indent,
                          const char *Comment,
                          char **ErrStr)
```

MVs: To be saved in stream.

Handler: A handler to the open stream.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if successful, FALSE otherwise.

Description: Generic routine to write B-spline multi-variates to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.473 MvarBzrMVReadFromFile (mvarread.c:158)

files
read
multi-variates

```
MvarMVStruct *MvarBzrMVReadFromFile(const char *FileName,
                                    char **ErrStr,
                                    int *ErrLine)
```

FileName: To read the multi-variate from.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read multi-variate, or NULL if an error occurred.

Description: Reads from a file Bezier multi-variates. If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occurred in file. If no error is detected *ErrStr = NULL.

9.2.474 MvarBzrMVReadFromFile2 (mvarread.c:223)

files
read
multi-variates

```
MvarMVStruct *MvarBzrMVReadFromFile2(int Handler,
                                     CagdBType NameWasRead,
                                     char **ErrStr,
                                     int *ErrLine)
```

Handler: A handler to the open stream.

NameWasRead: If FALSE, also reads the MULTIVAR BEZIER prefix.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read multi-variate, or NULL if an error occurred.

Description: Reads from a file a Bezier multi-variate. If NameWasRead is TRUE, it is assumed prefix "[MULTIVAR BEZIER" has already been read. This is useful for a global parser which invokes this routine to read from a file several times as a parent controller. For exactly this reason, the given file descriptor is NOT closed in the end. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occurred in file relative to beginning of multi-variate. If no error is detected *ErrStr is set to NULL.

9.2.475 MvarBzrMVWriteToFile (mvar_wrt.c:142)

files
write
multi-variates

```
int MvarBzrMVWriteToFile(const MvarMVStruct *MVs,
                         const char *FileName,
                         int Indent,
                         const char *Comment,
                         char **ErrStr)
```

MVs: To be saved in file.

FileName: File name where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if successful, FALSE otherwise.

Description: Generic routine to write Bezier multi-variates to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.476 MvarBzrMVWriteToFile2 (mvar_wrt.c:183)

files
write
multi-variates

```
int MvarBzrMVWriteToFile2(const MvarMVStruct *MVs,
                          int Handler,
                          int Indent,
                          const char *Comment,
                          char **ErrStr)
```

MVs: To be saved in stream.

Handler: A handler to the open stream.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if successful, FALSE otherwise.

Description: Generic routine to write Bezier multi-variates to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.477 MvarMVReadFromFile (mvarread.c:34)

files
read
multi-variates

```
MvarMVStruct *MvarMVReadFromFile(const char *FileName,
                                 char **ErrStr,
                                 int *ErrLine)
```

FileName: To read the multi-variate from.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read multi-variate, or NULL if an error occurred.

Description: Reads from a file multi-variates. If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occurred in file. If no error is detected *ErrStr = NULL.

9.2.478 MvarMVReadFromFile2 (mvarread.c:101)

files
read
multi-variates

```
MvarMVStruct *MvarMVReadFromFile2(int Handler, char **ErrStr, int *ErrLine)
```

Handler: A handler to the open stream.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read multi-variate, or NULL if an error occurred.

Description: Reads from a file a multi-variate. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occurred in file relative to beginning of multi-variate. If no error is detected *ErrStr is set to NULL.

9.2.479 MvarMVWriteToFile (mvar_wrt.c:34)

files
write
multi-variates

```
int MvarMVWriteToFile(const MvarMVStruct *MVs,
                      const char *FileName,
                      int Indent,
                      const char *Comment,
                      char **ErrStr)
```

MVs: To be saved in file f.

FileName: File name where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if successful, FALSE otherwise.

Description: Generic routine to write multi-variates to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.480 MvarMVWriteToFile2 (mvar_wrt.c:72)

files
write
multi-variates

```
int MvarMVWriteToFile2(const MvarMVStruct *MVs,
                       int Handler,
                       int Indent,
                       const char *Comment,
                       char **ErrStr)
```

MVs: To be saved in stream.

Handler: A handler to the open stream.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if successful, FALSE otherwise.

Description: Generic routine to write multi-variates to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.481 MvarMVWriteToFile3 (mvar_wrt.c:110)

files

write

```
int MvarMVWriteToFile3(const MvarMVStruct *MVs,
                       FILE *f,
                       int Indent,
                       const char *Comment,
                       char **ErrStr)
```

MVs: To be saved in file f.

f: File descriptor where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if successful, FALSE otherwise.

Description: Generic routine to write multi-variate(s) to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.482 TrimReadTrimmedSrfFromFile (trimread.c:33)

files

read

trimmed surfaces

```
TrimSrfStruct *TrimReadTrimmedSrfFromFile(const char *FileName,
                                          char **ErrStr,
                                          int *ErrLine)
```

FileName: To read the trimmed surface from.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read trimmed surface, or NULL if an error occurred.

Description: Reads from a file trimmed surfaces. If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occurred in file. If no error is detected *ErrStr = NULL.

9.2.483 TrimReadTrimmedSrfFromFile2 (trimread.c:94)

files

read

trimmed surfaces

```
TrimSrfStruct *TrimReadTrimmedSrfFromFile2(int Handler,
                                           CagdBType NameWasRead,
                                           char **ErrStr,
                                           int *ErrLine)
```

Handler: A handler to the open stream.

NameWasRead: If FALSE, also reads the TRIMSRF BEZIER prefix.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read trimmed surface, or NULL if an error occurred.

Description: Reads from a file a trimmed surface. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occurred in file relative to beginning of trimmed surface. Assumes the [TRIMSRF prefix was read if NameWasRead. If no error is detected *ErrStr is set to NULL.

9.2.484 TrimWriteTrimmedSrfToFile (trim_wrt.c:34)

files

write

```
int TrimWriteTrimmedSrfToFile(const TrimSrfStruct *TrimSrfs,
                              const char *FileName,
                              int Indent,
                              const char *Comment,
                              char **ErrStr)
```

TrimSrfs: To be saved in stream.

FileName: File name where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write a trimmed surface to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.485 TrimWriteTrimmedSrfToFile2 (trim_wrt.c:75)

files

write

stream

```
int TrimWriteTrimmedSrfToFile2(const TrimSrfStruct *TrimSrfs,
                               int Handler,
                               int Indent,
                               const char *Comment,
                               char **ErrStr)
```

TrimSrfs: To be saved in stream.

Handler: A handler to the open stream.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write trimmed surface(s) to the given stream. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.486 TrimWriteTrimmedSrfToFile3 (trim_wrt.c:154)

files

write

```
int TrimWriteTrimmedSrfToFile3(const TrimSrfStruct *TrimSrfs,
                               FILE *f,
                               int Indent,
                               const char *Comment,
                               char **ErrStr)
```

TrimSrfs: To be saved in stream.

f: File descriptor where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write trimmed surface(s) to the given stream. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.487 TrivBspTVReadFromFile (trivread.c:341)

files
read
trivariates

```
TrivTVStruct *TrivBspTVReadFromFile(const char *FileName,
                                    char **ErrStr,
                                    int *ErrLine)
```

FileName: To read the trivariate from.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read trivariate, or NULL if an error occurred.

Description: Reads from a file B-spline trivariates. If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occurred in file. If no error is detected *ErrStr = NULL.

9.2.488 TrivBspTVReadFromFile2 (trivread.c:406)

files
read
trivariates

```
TrivTVStruct *TrivBspTVReadFromFile2(int Handler,
                                     CagdBType NameWasRead,
                                     char **ErrStr,
                                     int *ErrLine)
```

Handler: A handler to the open stream.

NameWasRead: If FALSE, also reads the TRIVAR BEZIER prefix.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read trivariate, or NULL if an error occurred.

Description: Reads from a file a B-spline trivariate. If NameWasRead is TRUE, it is assumed prefix "[TRIVAR BSPLINE" has already been read. This is useful for a global parser which invokes this routine to read from a file several times as a parent controller. For exactly this reason, the given file descriptor is NOT closed in the end. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occurred in file relative to beginning of trivariate. If no error is detected *ErrStr is set to NULL.

9.2.489 TrivBspTVWriteToFile (triv_wrt.c:273)

files
write
trivariates

```
int TrivBspTVWriteToFile(const TrivTVStruct *TVs,
                         const char *FileName,
                         int Indent,
                         const char *Comment,
                         char **ErrStr)
```

TVs: To be saved in file f.

FileName: File name where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write Bspline trivariates to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.490 TrivBspTVWriteToFile2 (triv_wrt.c:314)

files

write

trivariates

```
int TrivBspTVWriteToFile2(const TrivTVStruct *TVs,
                          int Handler,
                          int Indent,
                          const char *Comment,
                          char **ErrStr)
```

TVs: To be saved in stream.

Handler: A handler to the open stream.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if successful, FALSE otherwise.

Description: Generic routine to write B-spline trivariates to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.491 TrivBzrTVReadFromFile (trivread.c:148)

files

read

trivariates

```
TrivTVStruct *TrivBzrTVReadFromFile(const char *FileName,
                                    char **ErrStr,
                                    int *ErrLine)
```

FileName: To read the trivariate from.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read trivariate, or NULL if an error occurred.

Description: Reads from a file Bezier trivariates. If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occurred in file. If no error is detected *ErrStr = NULL.

9.2.492 TrivBzrTVReadFromFile2 (trivread.c:213)

files

read

trivariates

```
TrivTVStruct *TrivBzrTVReadFromFile2(int Handler,
                                     CagdBType NameWasRead,
                                     char **ErrStr,
                                     int *ErrLine)
```

Handler: A handler to the open stream.

NameWasRead: If FALSE, also reads the TRIVAR BEZIER prefix.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read trivariate, or NULL if an error occurred.

Description: Reads from a file a Bezier trivariate. If NameWasRead is TRUE, it is assumed prefix "[TRIVAR BEZIER" has already been read. This is useful for a global parser which invokes this routine to read from a file several times as a parent controller. For exactly this reason, the given file descriptor is NOT closed in the end. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occurred in file relative to beginning of trivariate. If no error is detected *ErrStr is set to NULL.

9.2.493 TrivBzrTVWriteToFile (triv_wrt.c:147)

files write trivariates

```
int TrivBzrTVWriteToFile(const TrivTVStruct *TVs,
                         const char *FileName,
                         int Indent,
                         const char *Comment,
                         char **ErrStr)
```

TVs: To be saved in file.

FileName: File name where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write Bezier trivariates to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.494 TrivBzrTVWriteToFile2 (triv_wrt.c:188)

files write trivariates

```
int TrivBzrTVWriteToFile2(const TrivTVStruct *TVs,
                          int Handler,
                          int Indent,
                          const char *Comment,
                          char **ErrStr)
```

TVs: To be saved in stream.

Handler: A handler to the open stream.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if successful, FALSE otherwise.

Description: Generic routine to write Bezier trivariates to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.495 TrivTVReadFromFile (trivread.c:33)

files read trivariates

```
TrivTVStruct *TrivTVReadFromFile(const char *FileName,
                                 char **ErrStr,
                                 int *ErrLine)
```

FileName: To read the trivariate from.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read trivariate, or NULL if an error occurred.

Description: Reads from a file trivariates. If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occurred in file. If no error is detected *ErrStr = NULL.

9.2.496 TrivTVReadFromFile2 (trivread.c:95)

files read trivariates

```
TrivTVStruct *TrivTVReadFromFile2(int Handler, char **ErrStr, int *ErrLine)
```

Handler: A handler to the open stream.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read trivariate, or NULL if an error occurred.

Description: Reads from a file a trivariate. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occurred in file relative to beginning of trivariate. If no error is detected *ErrStr is set to NULL.

9.2.497 TrivTVWriteToFile (triv_wrt.c:34)

files

write

trivariates

```
int TrivTVWriteToFile(const TrivTVStruct *TVs,
                      const char *FileName,
                      int Indent,
                      const char *Comment,
                      char **ErrStr)
```

TVs: To be saved in file f.

FileName: File name where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if successful, FALSE otherwise.

Description: Generic routine to write trivariates to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.498 TrivTVWriteToFile2 (triv_wrt.c:74)

files

write

trivariates

```
int TrivTVWriteToFile2(const TrivTVStruct *TVs,
                       int Handler,
                       int Indent,
                       const char *Comment,
                       char **ErrStr)
```

TVs: To be saved in stream.

Handler: A handler to the open stream.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if successful, FALSE otherwise.

Description: Generic routine to write trivariates to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.499 TrivTVWriteToFile3 (triv_wrt.c:114)

files

write

```
int TrivTVWriteToFile3(const TrivTVStruct *TVs,
                       FILE *f,
                       int Indent,
                       const char *Comment,
                       char **ErrStr)
```

TVs: To be saved in file f.

f: File descriptor where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write trivariate(s) to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.500 TrngBspTriSrfReadFromFile (trngread.c:349)

files
read
triangular surfaces

```
TrngTriangSrfStruct *TrngBspTriSrfReadFromFile(const char *FileName,
                                               char **ErrStr,
                                               int *ErrLine)
```

FileName: To read the triangular surface from.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read triangular surface, or NULL if an error occurred.

Description: Reads from a file B-spline triangular surfaces. If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occurred in file. If no error is detected *ErrStr = NULL.

9.2.501 TrngBspTriSrfReadFromFile2 (trngread.c:415)

files
read
triangular surfaces

```
TrngTriangSrfStruct *TrngBspTriSrfReadFromFile2(int Handler,
                                                CagdBType NameWasRead,
                                                char **ErrStr,
                                                int *ErrLine)
```

Handler: A handler to the open stream.

NameWasRead: If FALSE, also reads the TRISRF BEZIER prefix.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read triangular surface, or NULL if an error occurred.

Description: Reads from a file a B-spline triangular surface. If NameWasRead is TRUE, it is assumed prefix "[TRISRF BSPLINE" has already been read. This is useful for a global parser which invokes this routine to read from a file several times as a parent controller. For exactly this reason, the given file descriptor is NOT closed in the end. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occurred in file relative to beginning of triangular surface. If no error is detected *ErrStr is set to NULL.

9.2.502 TrngBspTriSrfWriteToFile (trng_wrt.c:265)

files
write
triangular surfaces

```
int TrngBspTriSrfWriteToFile(const TrngTriangSrfStruct *TriSrfs,
                             const char *FileName,
                             int Indent,
                             const char *Comment,
                             char **ErrStr)
```

TriSrfs: To be saved in file f.

FileName: File name where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write Bspline triangular surfaces to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.503 TrngBspTriSrfWriteToFile2 (trng_wrt.c:306)

files

write

triangular surfaces

```
int TrngBspTriSrfWriteToFile2(const TrngTriangSrfStruct *TriSrfs,
                              int Handler,
                              int Indent,
                              const char *Comment,
                              char **ErrStr)
```

TriSrfs: To be saved in stream.

Handler: A handler to the open stream.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if successful, FALSE otherwise.

Description: Generic routine to write B-spline triangular surfaces to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.504 TrngBzrTriSrfReadFromFile (trngread.c:158)

files

read

triangular surfaces

```
TrngTriangSrfStruct *TrngBzrTriSrfReadFromFile(const char *FileName,
                                               char **ErrStr,
                                               int *ErrLine)
```

FileName: To read the triangular surface from.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read triangular surface, or NULL if an error occurred.

Description: Reads from a file Bezier triangular surfaces. If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occurred in file. If no error is detected *ErrStr = NULL.

9.2.505 TrngBzrTriSrfReadFromFile2 (trngread.c:224)

files

read

triangular surfaces

```
TrngTriangSrfStruct *TrngBzrTriSrfReadFromFile2(int Handler,
                                                CagdBType NameWasRead,
                                                char **ErrStr,
                                                int *ErrLine)
```

Handler: A handler to the open stream.

NameWasRead: If FALSE, also reads the TRISRF BEZIER prefix.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read triangular surface, or NULL if an error occurred.

Description: Reads from a file a Bezier triangular surface. If NameWasRead is TRUE, it is assumed prefix "[TRISRF BEZIER" has already been read. This is useful for a global parser which invokes this routine to read from a file several times as a parent controller. For exactly this reason, the given file descriptor is NOT closed in the end. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occurred in file relative to beginning of triangular surface. If no error is detected *ErrStr is set to NULL.

9.2.506 TrngBzrTriSrfWriteToFile (trng_wrt.c:147)

files
write
triangular surfaces

```
int TrngBzrTriSrfWriteToFile(const TrngTriangSrfStruct *TriSrfs,
                             const char *FileName,
                             int Indent,
                             const char *Comment,
                             char **ErrStr)
```

TriSrfs: To be saved in file.

FileName: File name where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write Bezier triangular surfaces to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.507 TrngBzrTriSrfWriteToFile2 (trng_wrt.c:188)

files
write
triangular surfaces

```
int TrngBzrTriSrfWriteToFile2(const TrngTriangSrfStruct *TriSrfs,
                              int Handler,
                              int Indent,
                              const char *Comment,
                              char **ErrStr)
```

TriSrfs: To be saved in stream.

Handler: A handler to the open stream.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if successful, FALSE otherwise.

Description: Generic routine to write Bezier triangular surfaces to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.508 TrngGrgTriSrfReadFromFile (trngread.c:575)

files
read
triangular surfaces

```
TrngTriangSrfStruct *TrngGrgTriSrfReadFromFile(const char *FileName,
                                               char **ErrStr,
                                               int *ErrLine)
```

FileName: To read the triangular surface from.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read triangular surface, or NULL if an error occurred.

Description: Reads from a file Gregory triangular surfaces. If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occurred in file. If no error is detected *ErrStr = NULL.

9.2.509 TrngGrgTriSrfReadFromFile2 (trngread.c:641)

files
read
triangular surfaces

```
TrngTriangSrfStruct *TrngGrgTriSrfReadFromFile2(int Handler,
                                                CagdBType NameWasRead,
                                                char **ErrStr,
                                                int *ErrLine)
```

Handler: A handler to the open stream.

NameWasRead: If FALSE, also reads the TRISRF GREGORY prefix.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read triangular surface, or NULL if an error occurred.

Description: Reads from a file a Gregory triangular surface. If NameWasRead is TRUE, it is assumed prefix "[TRISRF GREGORY" has already been read. This is useful for a global parser which invokes this routine to read from a file several times as a parent controller. For exactly this reason, the given file descriptor is NOT closed in the end. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occurred in file relative to beginning of triangular surface. If no error is detected *ErrStr is set to NULL.

9.2.510 TrngGrgTriSrfWriteToFile (trng_wrt.c:400)

files
write
triangular surfaces

```
int TrngGrgTriSrfWriteToFile(const TrngTriangSrfStruct *TriSrfs,
                             const char *FileName,
                             int Indent,
                             const char *Comment,
                             char **ErrStr)
```

TriSrfs: To be saved in file.

FileName: File name where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write Gregory triangular surfaces to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.511 TrngGrgTriSrfWriteToFile2 (trng_wrt.c:441)

files
write
triangular surfaces

```
int TrngGrgTriSrfWriteToFile2(const TrngTriangSrfStruct *TriSrfs,
                              int Handler,
                              int Indent,
                              const char *Comment,
                              char **ErrStr)
```

TriSrfs: To be saved in stream.

Handler: A handler to the open stream.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if successful, FALSE otherwise.

Description: Generic routine to write Gregory triangular surfaces to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.512 TrngTriSrfReadFromFile (trngread.c:34)

files
read
triangular surfaces

```
TrngTriangSrfStruct *TrngTriSrfReadFromFile(const char *FileName,
                                            char **ErrStr,
                                            int *ErrLine)
```

FileName: To read the triangular surface from.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read triangular surface, or NULL if an error occurred.

Description: Reads from a file triangular surfaces. If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occurred in file. If no error is detected *ErrStr = NULL.

9.2.513 TrngTriSrfReadFromFile2 (trngread.c:101)

files
read
triangular surfaces

```
TrngTriangSrfStruct *TrngTriSrfReadFromFile2(int Handler,
                                             char **ErrStr,
                                             int *ErrLine)
```

Handler: A handler to the open stream.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: The read triangular surface, or NULL if an error occurred.

Description: Reads from a file a triangular surface. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occurred in file relative to beginning of triangular surface. If no error is detected *ErrStr is set to NULL.

9.2.514 TrngTriSrfWriteToFile (trng_wrt.c:34)

files
write
triangular surfaces

```
int TrngTriSrfWriteToFile(const TrngTriangSrfStruct *TriSrfs,
                          const char *FileName,
                          int Indent,
                          const char *Comment,
                          char **ErrStr)
```

TriSrfs: To be saved in file f.

FileName: File name where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write triangular surfaces to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.515 TrngTriSrfWriteToFile2 (trng_wrt.c:74)

files
write
triangular surfaces

```
int TrngTriSrfWriteToFile2(const TrngTriangSrfStruct *TriSrfs,
                           int Handler,
                           int Indent,
                           const char *Comment,
                           char **ErrStr)
```

TriSrfs: To be saved in stream.

Handler: A handler to the open stream.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write triangular surfaces to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.516 TrngTriSrfWriteToFile3 (trng_wrt.c:114)

files

write

```
int TrngTriSrfWriteToFile3(const TrngTriangSrfStruct *TriSrfs,
                           FILE *f,
                           int Indent,
                           const char *Comment,
                           char **ErrStr)
```

TriSrfs: To be saved in file f.

f: File descriptor where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write triangular surfaces(s) to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.517 VMdlReadModelFromFile (vmdlread.c:69)

files

read

Volume models

```
VMdlVModelStruct *VMdlReadModelFromFile(const char *FileName,
                                        char **ErrStr,
                                        int *ErrLine)
```

FileName: To read the volumetric model from.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: Read volumetric model, or NULL if an error occurred.

Description: Reads from a file a volumetric model. If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occurred in file. If no error is detected *ErrStr = NULL.

See also: VMdlReadModelSrfFromFile2,

9.2.518 VMdlReadModelFromFile2 (vmdlread.c:132)

files

read

model

```
VMdlVModelStruct *VMdlReadModelFromFile2(int Handler,
                                         CagdBType NameWasRead,
                                         char **ErrStr,
                                         int *ErrLine)
```

Handler: A handler to the open stream.

NameWasRead: If FALSE, also reads the VMODEL prefix.

ErrStr: Will be initialized if an error has occurred.

ErrLine: Line number in file FileName of the error, if occurred.

Returns: Read volumetric model, or NULL if an error occurred.

Description: Reads from a file a volumetric model. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occurred in file relative to model. Assumes the [MODEL prefix was read if NameWasRead. If no error is detected *ErrStr is set to NULL.

See also: VMdlReadModelSrfFromFile,

9.2.519 VMdlWriteVModelToFile (vmdl_wrt.c:62)

files

write

```
int VMdlWriteVModelToFile(const VMdlVModelStruct *VModels,
                          const char *FileName,
                          int Indent,
                          const char *Comment,
                          char **ErrStr)
```

VModels: To be saved in stream.

FileName: File name where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if successful, FALSE otherwise.

Description: Generic routine to write model(s) to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.520 VMdlWriteVModelToFile2 (vmdl_wrt.c:136)

files

write

stream

```
int VMdlWriteVModelToFile2(const VMdlVModelStruct *VModels,
                           int Handler,
                           int Indent,
                           const char *Comment,
                           char **ErrStr)
```

VModels: To be saved in stream.

Handler: A handler to the open stream.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if successful, FALSE otherwise.

Description: Generic routine to write models(s) to the given stream. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.521 VMdlWriteVModelToFile3 (vmdl_wrt.c:103)

files

write

```
int VMdlWriteVModelToFile3(const VMdlVModelStruct *VModels,
                           FILE *f,
                           int Indent,
                           const char *Comment,
                           char **ErrStr)
```

VModels: To be saved in stream.

f: File descriptor where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if successful, FALSE otherwise.

Description: Generic routine to write volumetric model(s) to the given stream. If Comment is NULL, no comment is printed, if "" only internal comment.

9.2.522 _IPFprintf (iritprs2.c:642)

files

```
void _IPFprintf(int Handler, int Indent, const char *va_alist, ...)
```

Handler: A handler to the open stream.

Indent: All printing will start at this column.

va_alist: Do "man stdarg"

Returns: void

Description: Same as fprintf but with indentation.
See also: IPSetPrintFunc, IPSetFloatFormat, IPPrintFunc,

9.2.523 _IPGetAllAttributes (iritprs1.c:3110)

```
int _IPGetAllAttributes(IPAttributeStruct **Attr, int Handler)
```

Attr: Where attributes should go to.

Handler: A handler to the open stream.

Returns: Number of attributes read.

Description: Routine to read from input file f the following [ATTR ...] [ATTR ...]. The first '[' was already read.
See also: _IPGetAllAttributes2,

9.2.524 _IPGetAllAttributes2 (iritprs1.c:3152)

```
int _IPGetAllAttributes2(IPAttributeStruct **Attr, int Handler)
```

Attr: Where attributes should go to.

Handler: A handler to the open stream.

Returns: Number of attributes read.

Description: Routine to read from input file f the following [ATTR ...] [ATTR ...]. Query if the next token is a '[', and fetch as many attrs as found.
See also: _IPGetAllAttributes,

9.2.525 _IPGetCloseParenToken (iritprs1.c:2099)

```
void _IPGetCloseParenToken(int Handler)
```

Handler: A handler to the open stream.

Returns: void

Description: Routine to get close paren token from FILE f. This function invokes the parser's abort routine, if no close paren.

9.2.526 _IPGetToken (iritprs1.c:2736)

```
IPTokenType _IPGetToken(int Handler, char *StringToken)
```

Handler: A handler to the open stream.

StringToken: String token will be placed herein.

Returns: Token as a numeral.

Description: Routine to get the next token out of the input file f as token number. StringToken must be allocated before calling this routine!

9.2.527 _IPGetTokenAttr (iritprs1.c:2940)

```
IPTokenType _IPGetTokenAttr(int Handler,
                            char *StringToken,
                            IPAttributeStruct **Attr)
```

Handler: A handler to the open stream.

StringToken: String token will be placed herein.

Attr: Attributes to fetch, if any.

Returns: Token as a numeral.

Description: Routine to get the next token out of the input file f as token number. StringToken must be allocated before calling this routine! Also senses if Attributes follows an open Paren and fetch them if so.

9.2.528 _IPParseResetError (prsr_err.c:238)

```
void _IPParseResetError()
```

Returns: void

Description: Clears the error records for a fresh start.

9.2.529 _IPSkipToCloseParenToken (iritprs1.c:2121)

```
int _IPSkipToCloseParenToken(int Handler)
```

Handler: A handler to the open stream.

Returns: TRUE, if found close paren.

Description: Routine to skip to the next closed parenthesis.

9.2.530 _IPThisLittleEndianHardware (iritprsb.c:123)

```
int _IPThisLittleEndianHardware(void)
```

Returns: TRUE/FALSE for little/big endian style.

Description: Routine to test little vs. big endian style of packing bytes. Test is done by placing a none zero byte into the first place of a zero integer.

9.2.531 _IPUnGetToken (iritprs1.c:2521)

```
void _IPUnGetToken(int Handler, char *StringToken)
```

Handler: A handler to the open stream.

StringToken: Token to unget

Returns: void

Description: Routine to unget one token (on stack of UNGET_STACK_SIZE levels!)

9.2.532 _IritPrsrFatalErrorEx (prsr_err.c:216)

error handling

```
void _IritPrsrFatalErrorEx(IritPrsrFatalErrorType ErrID,
                           int ErrLine,
                           const char *ErrDesc)
```

ErrID: Error type that was raised.

ErrLine: Line number of error in processed file or IP_ERR_NO_LINE_NUM. to ignore.

ErrDesc: Optional error description.

Returns: void

Description: Trap Prsr_lib errors right here. Provides a default error handler for the prsr library. Gets an error description using PrsrDescribeError, prints it and exit the program using exit.

Chapter 10

Rendering Library, rndr_lib

10.1 General Information

This library provides a powerful full screen scan conversion Z-buffer tool to process IRIT geometry and convert it into images. This library allows one to scan convert any IRIT geometry including polylines and curves that are converted to skinny polygons on the fly. The library offers regular scan conversion with flat, Gouraud, and Phong shading and several antialiasing filters along with advanced features such as transparency and animation support, and width depth cueing on polyline/curves rendering. The library also provide direct access to the depth Z-buffer as well as a stencil buffer.

10.2 Library Functions

10.2.1 FastAllocInit (fstalloc.c:76)

```
FastAllocType FastAllocInit(unsigned TypeSz,
                            unsigned BlkSz,
                            unsigned AllgnBits,
                            unsigned Verbose)
```

TypeSz: IN, size of allocated type (in bytes).
BlkSz: IN, size of blocks allocated (in bytes, >= TypeSz).
AllgnBits: IN, alignment of each allocation is 2^AllgnBits.
Verbose: IN, iff TRUE, FastAllocDestroy() prints statistics.
Returns: The FastAlloc instance.
Description: Initializes an instance of an FastAllocType.

10.2.2 FastAllocNew (fstalloc.c:114)

```
VoidPtr FastAllocNew(FastAllocType Alloc)
```

Alloc: IN, FastAllocType instance.
Returns: Void pointer to the new memory area.
Description: Allocates new area of memory of size TypeSz (see FastAllocInit()).

10.2.3 INCRndrBeginObject (nc_zbufr.c:223)

scan irit object
z-buffer

```
void INCRndrBeginObject(INCZBufferPtrType Rend, IPObjectStruct *Object)
```

Rend: IN, OUT, the rendering context.
Object: IN, the object to be scanned.
Returns: void
Description: Sets the Irit object to be scan converted.

10.2.4 INCRndrDestroy (nc_zbufr.c:198)

destroy
dispose
free
release

```
void INCRndrDestroy(INCZBufferPtrType Rend)
```

Rend: IN,OUT, the rendering context.

Returns: void

Description: Dispose of a the rendering context.

10.2.5 INCRndrEndObject (nc_zbufr.c:368)

scan
irit object
z-buffer

```
void INCRndrEndObject(INCZBufferPtrType Rend)
```

Rend: IN, OUT, the rendering context.

Returns: void

Description: Marks the end of the object scaning.

10.2.6 INCRndrGetActiveCells (nc_zbufr.c:597)

```
int INCRndrGetActiveCells(INCZBufferPtrType Rend,
                          int *MinCellX,
                          int *MinCellY,
                          int *MaxCellX,
                          int *MaxCellY,
                          IrtRType *ZPixelsRemoved)
```

Rend: IN,OUT, the rendering context.

MinCellX, MinCellY: OUT, Minimum indices of active cells.

MaxCellX, MaxCellY: OUT, Maximum indices of active cells.

ZPixelsRemoved: OUT, number of pixels removed since last call.

Returns: TRUE if we do have active cells.

Description: Gets the active cells - a rectangular (XMin, YMin) :; (XMax, YMax) indices of cells where the Z-buffer was updated in, since the last call. This function also resets all cells to inactive state

10.2.7 INCRndrGetLineDepth (nc_zbufr.c:447)

z-buffer
line Z information

```
void INCRndrGetLineDepth(INCZBufferPtrType Rend,
                         int x1,
                         int x2,
                         int y,
                         IrtRType *ZValues)
```

Rend: IN, OUT, the rendering context.

x1, x2: IN, the x-range to fetch along the line.

y: IN, the line number.

ZValues: OUT, a user allocated buffer to hold the result.

Returns: void

Description: Retrieve z-coordinate data from the z-buffer.

10.2.8 INCRndrGetPixelDepth (nc_zbufr.c:419)

z-buffer

line depth

```
void INCRndrGetPixelDepth(INCZBufferPtrType Rend,
                          int x,
                          int y,
                          IrtRType *Result)
```

Rend: IN,OUT, the rendering context.

x: IN, the column number.

y: IN, the line number.

Result: OUT, the user allocated buffer to hold the result.

Returns: void

Description: Retrieve a pixel's depth from the z-buffer.

10.2.9 INCRndrGetZbufferGridCell (nc_zbufr.c:507)

z-buffer

```
int INCRndrGetZbufferGridCell(INCZBufferPtrType Rend,
                              int GridCellX,
                              int GridCellY,
                              IrtRType *ZValues,
                              int *XMin,
                              int *YMin,
                              int *XMax,
                              int *YMax)
```

Rend: IN, OUT, the rendering context.

GridCellX, GridCellY: IN, the grid cell to fetch its Z values.

ZValues: OUT, a user allocated buffer to hold the result.

XMin, YMin: OUT, minimal dimensions of fetched grid cell.

XMax, YMax: OUT, maximal dimensions of fetched grid cell.

Returns: TRUE if succesful, FALSE otherwise (out of grid range).

Description: Fetches the z-buffer regions under the requested grid cell. Returned is a linear buffer of as many pixels in the requested grid cell, ordered by rows. See INCRndrGetZbufferGridCellMaxSize to get maximal grid cell size.

10.2.10 INCRndrGetZbufferGridCellMaxSize (nc_zbufr.c:474)

z-buffer

```
void INCRndrGetZbufferGridCellMaxSize(INCZBufferPtrType Rend,
                                      int *GridSizeX,
                                      int *GridSizeY,
                                      int *GridCellXSize,
                                      int *GridCellYSize)
```

Rend: IN, OUT, the rendering context.

GridSizeX, GridSizeY: OUT, the grid dimensions.

GridCellXSize, GridCellYSize: OUT, cell sizes.

Returns: void

Description: Gets the grid size and maximal cell size, in pixels of the z-buffer.

10.2.11 INCRndrInitialize (nc_zbufr.c:82)

create

initialize

z-buffer

```
INCZBufferPtrType INCRndrInitialize(int ZBufSizeX,
                                    int ZBufSizeY,
                                    int GridSizeX,
                                    int GridSizeY,
                                    IrtPtType XYZMin,
                                    IrtPtType XYZMax,
                                    int BottomMaxZ)
```

ZBufSizeX: IN, the width of the z-buffer, in pixels.

ZBufSizeY: IN, the height of the z-buffer, in pixels.

GridSizeX: IN, the grid X size to divide the z-buffer into.

GridSizeY: IN, the grid Y size to divide the z-buffer into.

XYZMin: IN, the minimum corner of volume to consider.

XYZMax: IN, the maximum corner of volume to consider.

BottomMaxZ: IN, TRUE if bottom is maximal Z value, FALSE if bottom should capture minimal Z values.

Returns: A handle to the newly created NC z-buffer.

Description: Creates a new NC (Numerically controled) Rendering context, and returns a handle to it. Specified are the required sizes of the ZBuffer, the grid size to impose over the XY ZBuffer, and the stock dimensions. The grid will be used to efficiently fetch ZBuffer data at the granularity of grid cells instead of the entire ZBuffer every time.

10.2.12 INCRndrMapPixelsToCells (nc_zbufr.c:555)

```
int INCRndrMapPixelsToCells(INCZBufferPtrType Rend, int *X, int *Y)
```

Rend: IN,OUT, the rendering context.

X, Y: IN,OUT, pixel coordinates to map to the cell indices they are in.

Returns: TRUE if we found the right cell.

Description: Maps the given pixel coordinates to indices of the cell that contains these pixel coordinates.

10.2.13 INCRndrPutMask (nc_zbufr.c:294)

scan mask

polygon z-buffer

```
IrtRType INCRndrPutMask(INCZBufferPtrType Rend,
                        int *PosXY,
                        IrtRType PosZ,
                        IrtRType *Mask,
                        int MaskXSize,
                        int MaskYSize)
```

Rend: IN, OUT, the rendering context.

PosXY: IN, XY location where to place the center of the mask in the Z buffer.

PosZ: IN, The depth to place the mask at.

Mask: IN, The 2D square array of depth values.

MaskXSize: IN, X size of Mask.

MaskYSize: IN, Y size of Mask.

Returns: Amount of material removed by this call in Pixels^2*Z volume units.

Description: Scan convert a depth Mask into the Zbuffer. The Mask is a 2D square of depth values that is centered around Pos.

10.2.14 INCRndrPutPixel (nc_zbufr.c:389)

put pixel

```
void INCRndrPutPixel(INCZBufferPtrType Rend, int x, int y, IrtRType z)
```

Rend: IN, OUT, the rendering context.

x: IN, the column number.

y: IN, the line number.

z: IN, the pixel's depth.

Returns: void

Description: Manually adds a single pixel.

10.2.15 INCRndrPutTriangle (nc_zbufr.c:254)

scan triangle

polygon z-buffer

```
void INCRndrPutTriangle(INCZBufferPtrType Rend,
                        IPPolygonStruct *Triangle)
```

Rend: IN, OUT, the rendering context.

Triangle: IN, the triangle to be scanned.

Returns: void

Description: Scan converts a triangle polygon.

10.2.16 INCRndrSetZCmp (nc_zbufr.c:173)

comparison

z-buffer

sort

```
IRndrZBufferCmpType INCRndrSetZCmp(INCZBufferPtrType Rend,
                                   IRndrZBufferCmpType ZCmp)
```

Rend: IN,OUT, the rendering context.

ZCmp: IN, the comparison method.

Returns: Old comparison method.

Description: Sets the NC z-buffer comparison method.

10.2.17 IRndr1DClearDepth (zbufr_1d.c:99)

comparison

z-buffer

sort

```
void IRndr1DClearDepth(IRndrZBuffer1DPtrType Rend, IrtRType ClearZ)
```

Rend: IN,OUT, the rendering context.

ClearZ: IN, the new depth to reset the zbuffer to.

Returns: void

Description: Resets the 1D z-buffer depth.

10.2.18 IRndr1DDestroy (zbufr_1d.c:145)

destroy

dispose

free

release

```
void IRndr1DDestroy(IRndrZBuffer1DPtrType Rend)
```

Rend: IN,OUT, the rendering context.

Returns: void

Description: Dispose of the rendering context.

10.2.19 IRndr1DFilterCollinearEdges (zbufr_1d.c:435)

z-buffer
line Z information

```
IPPolygonStruct *IRndr1DFilterCollinearEdges(IRndrZBuffer1DPtrType Rend,
                                             IPPolygonStruct *Pl,
                                             int MergeInters)
```

Rend: IN, the rendering context.

Pl: IN, the polyline to filer for collinear edges, in place.

MergeInters: If TRUE, adjacent linear segments as detected in the 1D Z buffer are merged whenever possible.
*

Returns: The filtered polyline.

Description: Filters the resulting polyline for collinear edges, in place.

10.2.20 IRndr1DGetLineDepth (zbufr_1d.c:340)

z-buffer
line Z information

```
void IRndr1DGetLineDepth(IRndrZBuffer1DPtrType Rend,
                         int x1,
                         int x2,
                         IrtRType *ZValues)
```

Rend: IN, OUT, the rendering context.

x1, x2: IN, the x-range to fetch along the line.

ZValues: OUT, a user allocated buffer to hold the result.

Returns: void

Description: Retrieve z-coordinate data from the z-buffer.

10.2.21 IRndr1DGetPixelDepth (zbufr_1d.c:315)

z-buffer
line depth

```
void IRndr1DGetPixelDepth(IRndrZBuffer1DPtrType Rend, int x, IrtRType *Result)
```

Rend: IN,OUT, the rendering context.

x: IN, the column number.

Result: OUT, the user allocated buffer to hold the result.

Returns: void

Description: Retrieve a pixel's depth from the z-buffer.

10.2.22 IRndr1DInitialize (zbufr_1d.c:52)

create
initialize
z-buffer

```
IRndrZBuffer1DPtrType IRndr1DInitialize(int ZBuf1DSize,
                                        IrtRType XMin,
                                        IrtRType XMax,
                                        IrtRType ZMin,
                                        IrtRType ZMax,
                                        int BottomMaxZ)
```

ZBuf1DSize: IN, the length of the 1D z-buffer, in pixels.

XMin, XMax: IN, the min/maximum real dimension to consider.

ZMin, ZMax: IN, the min/maximum depth to consider.

BottomMaxZ: IN, TRUE if bottom is maximal Z value, FALSE if bottom should capture minimal Z values.

Returns: A handle to the newly created 1D z-buffer.

Description: Creates a new 1D Zuffer, and returns a handle to it. Specified are the required length of the ZBuffer, and the real dimensions.

10.2.23 IRndr1DPutLine (zbufr_1d.c:199)

scan polyline
z-buffer

```
void IRndr1DPutLine(IRndrZBuffer1DPtrType Rend,
                    IrtRType x1,
                    IrtRType z1,
                    IrtRType x2,
                    IrtRType z2)
```

Rend: IN, OUT, the rendering context.

x1, z1, x2, z2: IN, the line to scan convert.

Returns: void

Description: Scan converts a line.

10.2.24 IRndr1DPutPixel (zbufr_1d.c:278)

put pixel

```
void IRndr1DPutPixel(IRndrZBuffer1DPtrType Rend, int x, IrtRType z)
```

Rend: IN, OUT, the rendering context.

x: IN, the column number.

z: IN, the pixel's depth.

Returns: void

Description: Manually adds a single pixel.

10.2.25 IRndr1DPutPolyline (zbufr_1d.c:168)

scan polyline
z-buffer

```
void IRndr1DPutPolyline(IRndrZBuffer1DPtrType Rend, IPPolygonStruct *Pl)
```

Rend: IN, OUT, the rendering context.

Pl: IN, the polyline to scan convert.

Returns: void

Description: Scan converts a polyline.

10.2.26 IRndr1DSetZCmp (zbufr_1d.c:122)

comparison
z-buffer
sort

```
IRndrZBufferCmpType IRndr1DSetZCmp(IRndrZBuffer1DPtrType Rend,
                                   IRndrZBufferCmpType ZCmp)
```

Rend: IN,OUT, the rendering context.

ZCmp: IN, the comparison method.

Returns: Old comparison method.

Description: Sets the NC z-buffer comparison method.

10.2.27 IRndr1DUpperEnvAsPolyline (zbufr_1d.c:366)

z-buffer
line Z information

```
IPPolygonStruct *IRndr1DUpperEnvAsPolyline(IRndrZBuffer1DPtrType Rend,
                                           int MergeInters)
```

Rend: IN, the rendering context.

MergeInters: If TRUE, adjacent linear segments as detected in the 1D Z buffer are merged whenever possible.
*

Returns: The retrieved envelope as a polyline.

Description: Retrieve the z-buffer envelope as one polyline from XMin to XMax.

10.2.28 IRndrAddLightSource (rndr_lib.c:200)

add light source

list

```
void IRndrAddLightSource(IRndrPtrType Rend,
                         IRndrLightType Type,
                         IrtPtType Where,
                         IRndrColorType Color)
```

Rend: IN, OUT the rendering context.

Type: IN, the light type (POINT, VECTOR)

Where: IN, the light position.

Color: IN, the light's color.

Returns: void

Description: Adds a new light source.

10.2.29 IRndrBeginObject (rndr_lib.c:587)

scan irit object

z-buffer

```
void IRndrBeginObject(IRndrPtrType Rend,
                      IPObjectStruct *Object,
                      int NoShading)
```

Rend: IN, OUT, the rendering context.

Object: IN, the object to be scanned.

NoShading: IN, if TRUE, ignore shading on this one.

Returns: void

Description: Sets the Irit object to be scan converted.

10.2.30 IRndrBeginPll (rndr_lib.c:697)

new line

start

```
void IRndrBeginPll(IRndrPtrType Rend)
```

Rend: IN, OUT, the rendering context.

Returns: void

Description: Begin drawing a line.

10.2.31 IRndrClearColor (rndr_lib.c:176)

clear

reset

background

color

```
void IRndrClearColor(IRndrPtrType Rend)
```

Rend: IN,OUT, the rendering context.

Returns: void

Description: Reset color information to the registered background color.

10.2.32 IRndrClearDepth (rndr_lib.c:140)

clear

reset

depth

Z coordinate

```
void IRndrClearDepth(IRndrPtrType Rend, IRndrZDepthType ClearZ)
```

Rend: IN,OUT, the rendering context.

ClearZ: IN, Depth to clear the ZBuffer to.

Returns: void

Description: Clear depth information in the rendering context.

10.2.33 IRndrClearStencil (rndr_lib.c:158)

clear
reset
stencil

```
void IRndrClearStencil(IRndrPtrType Rend)
```

Rend: IN,OUT, the rendering context.

Returns: void

Description: Clear stencil information in the rendering context.

10.2.34 IRndrDestroy (rndr_lib.c:115)

destroy
dispose
free
release

```
void IRndrDestroy(IRndrPtrType Rend)
```

Rend: IN,OUT, the rendering context.

Returns: void

Description: Dispose of a the rendering context.

10.2.35 IRndrEndObject (rndr_lib.c:679)

scan
irit object
z-buffer

```
void IRndrEndObject(IRndrPtrType Rend)
```

Rend: IN, OUT, the rendering context.

Returns: void

Description: Marks the end of the object scaning.

10.2.36 IRndrEndPll (rndr_lib.c:745)

line end

```
void IRndrEndPll(IRndrPtrType Rend)
```

Rend: IN, OUT, the rendering context.

Returns: void

Description: Marks the end of the line .

10.2.37 IRndrGetClippingPlanes (rndr_lib.c:327)

clipping
viewing frastrum

```
void IRndrGetClippingPlanes(IRndrPtrType Rend, IrtPlnType *ClipPlanes)
```

Rend: IN, the rendering context.

ClipPlanes: OUT, a pointer to the 6 user allocated planes.

Returns: void

Description: Retrives the 6 clipping planes, defining the viewing frastrum.

10.2.38 IRndrGetLineColorAlpha (rndr_lib.c:877)

z-buffer
line color information

```
void IRndrGetLineColorAlpha(IRndrPtrType Rend, int y, IRndrColorType *Result)
```

Rend: IN, OUT, the rendering context.

y: IN, the line number.

Result: OUT, the user allocated buffer to hold the result.

Returns: void

Description: Retrieve color (and alpha) data from the z-buffer.

10.2.39 IRndrGetLineDepth (rndr_lib.c:901)

z-buffer

line Z information

```
void IRndrGetLineDepth(IRndrPtrType Rend, int y, IrtRType *Result)
```

Rend: IN, OUT, the rendering context.

y: IN, the line number.

Result: OUT, the user allocated buffer to hold the result.

Returns: void

Description: Retrieve z-coordinate data from the z-buffer.

10.2.40 IRndrGetLineStencil (rndr_lib.c:925)

z-buffer

line stencil information

```
void IRndrGetLineStencil(IRndrPtrType Rend, int y, int *Result)
```

Rend: IN, OUT, the rendering context.

y: IN, the line number.

Result: OUT, the user allocated buffer to hold the result.

Returns: void

Description: Retrieve stencil data from the z-buffer.

10.2.41 IRndrGetPixelColorAlpha (rndr_lib.c:809)

z-buffer

line color information

```
void IRndrGetPixelColorAlpha(IRndrPtrType Rend,
                             int x,
                             int y,
                             IRndrColorType *Result)
```

Rend: IN, OUT, the rendering context.

x: IN, the column number.

y: IN, the line number.

Result: OUT, the user allocated buffer to hold the result.

Returns: void

Description: Retrieve a pixel's color (and alpha) from the z-buffer.

10.2.42 IRndrGetPixelDepth (rndr_lib.c:833)

z-buffer

line depth

```
void IRndrGetPixelDepth(IRndrPtrType Rend,
                        int x,
                        int y,
                        IrtRType *Result)
```

Rend: IN,OUT, the rendering context.

x: IN, the column number.

y: IN, the line number.

Result: OUT, the user allocated buffer to hold the result.

Returns: void

Description: Retrieve a pixel's depth from the z-buffer.

10.2.43 IRndrGetPixelStencil (rndr_lib.c:857)

z-buffer line stencil

```
void IRndrGetPixelStencil(IRndrPtrType Rend,
                          int x,
                          int y,
                          int *Result)
```

Rend: IN, OUT, the rendering context.

x: IN, the column number.

y: IN, the line number.

Result: OUT, the user allocated buffer to hold the result.

Returns: void

Description: Retrieve a pixel's stencil from the z-buffer.

10.2.44 IRndrGetScene (rndr_lib.c:1055)

```
struct IRndrSceneStruct *IRndrGetScene(IRndrPtrType Rend)
```

Rend: IN, the rendering context.

Returns: The scene struct.

Description: Get the scene struct.

10.2.45 IRndrGetViewPrsp (rndr_lib.c:300)

view

perspective

matrix context

```
void IRndrGetViewPrsp(IRndrPtrType Rend,
                      IrtHmgnMatType ViewMat,
                      IrtHmgnMatType PrspMat,
                      IrtHmgnMatType ScrnMat)
```

Rend: IN,OUT, the rendering context.

ViewMat: OUT, the view matrix. If NULL, it's ignored.

PrspMat: OUT, the perspective matrix. If NULL, it's ignored.

ScrnMat: OUT, the mapping to the screen. If NULL, it's ignored.

Returns: void

Description: Gets the view, perspective and screen matrices.

10.2.46 IRndrInitialize (rndr_lib.c:61)

create

initialize

z-buffer

```
IRndrPtrType IRndrInitialize(int SizeX,
                             int SizeY,
                             int SuperSampSize,
                             int ColorQuantization,
                             IrtBType UseTransparency,
                             IrtBType BackfaceCulling,
                             IRndrColorType BackgrCol,
                             IrtRType AmbientLight,
                             int VisMap)
```

SizeX: IN, the width of the z-buffer.

SizeY: IN, the height of the z-buffer.

SuperSampSize: IN, the super-sample size.

ColorQuantization: IN, non zero to quantize the generated colors to ColorQuantization levels of colors.

UseTransparency: IN, whether tarnsparency is on.

BackfaceCulling: IN, whether to use back-face culling.

BackgrCol: IN, the background color.

AmbientLight: IN, the abient light factor.

VisMap: IN, TRUE to create a visibility map image.

Returns: a handle to the newly created z-buffer.

Description: Creates a new Rendering context, and returns a handle to it.

10.2.47 IRndrPutPixel (rndr_lib.c:785)

put pixel

```
void IRndrPutPixel(IRndrPtrType Rend,
                   int x,
                   int y,
                   IrtRType z,
                   IrtRType Transparency,
                   IRndrColorType Color,
                   IPPolygonStruct *Triangle)
```

Rend: IN, OUT, the rendering context.

x: IN, the column number.

y: IN, the line number.

z: IN, the pixel's depth.

Transparency: IN, the pixel's transparency value.

Color: IN, the new color of pixel at (x, y).

Triangle: IN, The triangle which created the added point.

Returns: void

Description: Manually adds a single pixel.

10.2.48 IRndrPutPllVertex (rndr_lib.c:716)

line vertex lineto

```
void IRndrPutPllVertex(IRndrPtrType Rend, IPVertexStruct *Vertex)
```

Rend: IN, OUT, the rendering context.

Vertex: IN, the next vertex of the line.

Returns: void

Description: Sets the next vertex of the line.

10.2.49 IRndrPutTriangle (rndr_lib.c:616)

scan triangle

polygon z-buffer

```
void IRndrPutTriangle(IRndrPtrType Rend, IPPolygonStruct *Triangle)
```

Rend: IN, OUT, the rendering context.

Triangle: IN, the triangle to be scanned.

Returns: void

Description: Scan converts a triangle polygon.

10.2.50 IRndrSaveFile (rndr_lib.c:980)

z-buffer save dump

```
void IRndrSaveFile(IRndrPtrType Rend,
                   const char *BaseDirectory,
                   const char *OutFileName,
                   const char *Type)
```

Rend: IN, OUT, the rendering context.
BaseDirectory: IN, the directory to save the file in.
OutFileName: IN, the file name.
Type: IN, the file type.
Returns: void
Description: Save the color info of the z-buffer into a file .

10.2.51 IRndrSaveFileCB (rndr_lib.c:955)

z-buffer save dump

```
void IRndrSaveFileCB(IRndrPtrType Rend,
                     IRndrImgGetTypeFuncType ImgSetType,
                     IRndrImgOpenFuncType ImgOpen,
                     IRndrImgWriteLineFuncType ImgWriteLine,
                     IRndrImgCloseFuncType ImgClose)
```

Rend: IN, OUT, the rendering context.
ImgSetType: IN, Image setting file type function
ImgOpen: IN, Function to open an image file.
ImgWriteLine: IN, Function to write one row (Vec of RGB & vec of Alpha).
ImgClose: IN, Function to close an image file.
Returns: void
Description: Sets the call back functions to invoked when saving files.

10.2.52 IRndrSaveFileDepth (rndr_lib.c:1005)

z-buffer save dump

```
void IRndrSaveFileDepth(IRndrPtrType Rend,
                        const char *BaseDirectory,
                        const char *OutFileName,
                        const char *Type)
```

Rend: IN, OUT, the rendering context.
BaseDirectory: IN, the directory to save the file in.
OutFileName: IN, the file name.
Type: IN, the file type.
Returns: void
Description: Save the z coordinate values of the z-buffer into a file.

10.2.53 IRndrSaveFileStencil (rndr_lib.c:1036)

z-buffer save dump

```
void IRndrSaveFileStencil(IRndrPtrType Rend,
                          const char *BaseDirectory,
                          const char *OutFileName,
                          const char *Type)
```

Rend: IN, OUT, the rendering context.
BaseDirectory: IN, the directory to save the file in.
OutFileName: IN, the file name.
Type: IN, the file type.
Returns: void
Description: Save the context of the z-buffer into a file .

10.2.54 IRndrSaveFileVisMap (rndr_lib.c:1079)

z-buffer save dump

```
void IRndrSaveFileVisMap(IRndrPtrType Rend,
                         const char *BaseDirectory,
                         const char *OutFileName,
                         const char *Type)
```

Rend: IN, OUT, the rendering context.

BaseDirectory: IN, the directory to save the file in.

OutFileName: IN, the file name.

Type: IN, the file type.

Returns: void

Description: Save the context of the UV-map into a file .

10.2.55 IRndrSetFilter (rndr_lib.c:224)

Filter

anti-aliasing

```
void IRndrSetFilter(IRndrPtrType Rend, char *FilterName)
```

Rend: IN,OUT, the rendering context.

FilterName: IN, the filter's name.

Returns: void

Description: Changes the filter, used for antialiasing.

10.2.56 IRndrSetPllParams (rndr_lib.c:376)

polyline line depth cue

```
void IRndrSetPllParams(IRndrPtrType Rend,
                       IrtRType MinWidth,
                       IrtRType MaxWidth,
                       IrtRType ZNear,
                       IrtRType ZFar)
```

Rend: IN,OUT, the rendering context.

MinWidth: IN, the width of the line at Z = Znear.

MaxWidth: IN, the width of the line at Z = Zfar.

ZNear, ZFar: IN, as stated above.usually the expected scene width.

Returns: void

Description: Sets the Polyline parameters, used for line drawing.

10.2.57 IRndrSetPostZCmpClbk (rndr_lib.c:506)

comparison

z-buffer

post

```
void IRndrSetPostZCmpClbk(IRndrPtrType Rend,
                          IRndrPixelClbkFuncType ZPassClbk,
                          IRndrPixelClbkFuncType ZFailClbk)
```

Rend: IN,OUT, the rendering context.

ZPassClbk, ZFailClbk: IN, the callback functions called on success/failure.

Returns: void

Description: Sets the z-buffer post comparison function callback function.

10.2.58 IRndrSetPreZCmpClbk (rndr_lib.c:479)

comparison pre callback z-buffer

```
IRndrPixelClbkFuncType IRndrSetPreZCmpClbk(IRndrPtrType Rend,
                                           IRndrPixelClbkFuncType PixelClbk)
```

Rend: IN, OUT, the rendering context.
PixelClbk: IN, the callback function.
Returns: Old callbakc function.
Description: Sets the z-buffer pre comparison function callback function.

10.2.59 IRndrSetRawMode (rndr_lib.c:404)

antialising access mode raw

```
IrtBType IRndrSetRawMode(IRndrPtrType Rend, IrtBType UseRawMode)
```

Rend: IN, OUT, the rendering context.
UseRawMode: IN, whether the access mode is RAW (otherwise filtered).
Returns: Old raw mode.
Description: Sets the z-buffer access mode (original super smapled data or filtered).

10.2.60 IRndrSetShadeModel (rndr_lib.c:244)

shading
light model

```
IRndrShadingType IRndrSetShadeModel(IRndrPtrType Rend,
                                    IRndrShadingType ShadeModel)
```

Rend: IN, OUT, the rendering context.
ShadeModel: IN, the new shading model (FLAT,GOURAUD,PHONG,NONE).
Returns: Old shading model.
Description: Changes the shading model.

10.2.61 IRndrSetViewPrsp (rndr_lib.c:276)

view
perspective
matrix context

```
void IRndrSetViewPrsp(IRndrPtrType Rend,
                      IrtHmgnMatType ViewMat,
                      IrtHmgnMatType PrspMat,
                      IrtHmgnMatType ScrnMat)
```

Rend: IN,OUT, the rendering context.
ViewMat: IN, the view matrix.
PrspMat: IN, the perspective matrix, NULL if parallel projection.
ScrnMat: IN, the mapping to the screen or NULL if scale [-1,+1] to image size.
Returns: void
Description: Sets the view and perspective matrices.

10.2.62 IRndrSetZBounds (rndr_lib.c:351)

clipping
viewing frastrum

```
void IRndrSetZBounds(IRndrPtrType Rend, IrtRType ZNear, IrtRType ZFar)
```

Rend: IN, OUT, the rendering context.
ZNear: IN, the (negation of the) z-coordinate of the near clipping plane.
ZFar: IN, the (negation of the) z-coordinate of the far clipping plane.
Returns: void
Description: Sets the near and far XY clipping planes, defining the viewing frustum.

10.2.63 IRndrSetZCmp (rndr_lib.c:454)

comparison
z-buffer
sort

```
IRndrZBufferCmpType IRndrSetZCmp(IRndrPtrType Rend, IRndrZBufferCmpType ZCmp)
```

Rend: IN,OUT, the rendering context.

ZCmp: IN, the comparison method.

Returns: Old comparison method.

Description: Sets the z-buffer comparison method.

10.2.64 IRndrSetZCmpPolicy (rndr_lib.c:430)

comparison
z-buffer
sort

```
IRndrZCmpPolicyFuncType IRndrSetZCmpPolicy(IRndrPtrType Rend,
                                           IRndrZCmpPolicyFuncType ZCmpPol)
```

Rend: IN,OUT, the rendering context.

ZCmpPol: IN, the comparison function (linear order).

Returns: Old comparison function.

Description: Sets the z-buffer comparison function.

10.2.65 IRndrStencilCmpFunc (rndr_lib.c:531)

stencil buffer
OpenGL
glStencilFunc

```
void IRndrStencilCmpFunc(IRndrPtrType Rend,
                         IRndrStencilCmpType SCmp,
                         int Ref,
                         unsigned Mask)
```

Rend: IN, OUT, the rendering context.

SCmp: IN, stencil test comparison type.

Ref: IN, stencil test refernce value.

Mask: IN, stencil test mask.

Returns: void

Description: Sets the z-buffer stencil test function.

10.2.66 IRndrStencilOp (rndr_lib.c:560)

stencil buffer
OpenGL
glStencilOp

```
void IRndrStencilOp(IRndrPtrType Rend,
                    IRndrStencilOpType Fail,
                    IRndrStencilOpType ZFail,
                    IRndrStencilOpType ZPass)
```

Rend: IN, OUT, the rendering context.

Fail: IN, stencil test failure operation.

ZFail: IN, Z-test failure operation.

ZPass: IN, Z-test pass operation.

Returns: void

Description: Sets the z-buffer stencil operations.

10.2.67 IRndrVMClear (vis_maps.c:259)

clear visibility map.

```
void IRndrVMClear(IRndrVMStruct *VisMap)
```

VisMap: IN, OUT, Pointer to visibility map.

Returns: void

Description: Clears the context of visibility map (Sets all UV information to initial values).

10.2.68 IRndrVMGetLine (vis_maps.c:936)

Visible

access

```
int IRndrVMGetLine(IRndrVMStruct *VisMap,
                   int u0,
                   int u1,
                   int v,
                   IrtRType **FilterCoeff,
                   IrtRType *Result,
                   IRndrVisibleValidityType *Validity)
```

VisMap: IN, OUT, pointer to visibility map.

u0: IN, minimal U coordinate.

u1: IN, maximal U coordinate.

v: IN, line V number.

FilterCoeff: IN, the filter to use in the super sampling. If NULL, uses the same weight for all samples.

Result: OUT, the visibility values of the line. When super sampling is disabled 0 For not visible and 1 for visible. When super sampling is enable the values are in [0,1] and reflacts the amount of visible cell out of the total valid cells (invalid cells are ignored). If empty, returns negative number. If Validity isn't IRNDR_VISMAP_VALID_OK, Result is undefined.

Validity: OUT, The validity of the pixel.

Returns: TRUE if successful.

Description: Retrieves visibility information of a specific line in UV space. The line should be allocated by the caller.

10.2.69 IRndrVMGetObjDomain (vis_maps.c:1062)

```
int IRndrVMGetObjDomain(IPObjectStruct *PObj,
                        IrtRType *UMin,
                        IrtRType *UMax,
                        IrtRType *VMin,
                        IrtRType *VMax)
```

PObj: IN - The object to get its domain. Objects which aren't polygons are ignored.

UMin, UMax, VMin, VMax: OUT - The domain of the object.

Returns: FALSE if no uv value was found.

Description: Get the domain of the given object (Object which aren't polygons are ignored).

10.2.70 IRndrVMInit (vis_maps.c:97)

initialize

create visibility map.

```
int IRndrVMInit(IRndrVMStruct *VisMap,
                IRndrSceneStruct *Scene,
                int SuperSize,
                int UVBackfaceCulling)
```

VisMap: IN, OUT, Pointer to visibility map object.

Scene: IN, Pointer to the related scene object.

SuperSize: IN, Super sampling size.

UVBackfaceCulling: IN, use backface culling during the UV scan coversion.

Returns: 0 if successful.

Description: Initialize a newly created visibility map.

10.2.71 IRndrVMIsPointInTriangle (vis_maps.c:430)

```
int IRndrVMIsPointInTriangle(IPPolygonStruct *Triangle,
                             IrtPtType Pt,
                             int Perim,
                             IrtRType *z,
                             IrtRType *s,
                             IrtRType *t)
```

Triangle: IN, The triangle.

Pt: IN, The point.

Perim: IN, If TRUE, points on the perimeter are considered as being in the triangle. If FALSE they are not in the triangle.

z: OUT, The z value of Triangle at the xy values given by Pt. Ignored if it's NULL.

s, t: The paramters of Pt in the plane forumla. Ignored if NULL.

Returns: Whether Pt is inside Triangle

Description: Check whether Pt is inside Triangle. (The calculation are done using only the xy coordinates). z returns the z value of triangle at the xy coordinates given by Pt (it may return the z value at the continuation of Triangle if xy is outside Triangle). s and t are the parameters which create Pt in the plane formula: Triangle[0] + s*(Triangle[2]-Triangle[0]) + t*(Triangle[1]-Triangle[0])

10.2.72 IRndrVMPrepareUVValuesOfGeoObj (vis_maps.c:1121)

```
int IRndrVMPrepareUVValuesOfGeoObj(IPObjectStruct *PObj,
                                   int MapWidth,
                                   int MapHeight,
                                   IPObjectStruct *PObj2)
```

PObj: The geometric object.

MapWidth, MapHeight: The dimensions of the visibility map.

PObj2: If it isn't NULL, this object is a list of objects which contains at least as much elements as PObj. Each element i in PObj2 will go through the same Transformations and scaling as element i in PObj. Objects which aren't surfaces, trimmed surfaces or polygons are ignored.

Returns: FALSE if PObj doesn't contain any objects.

Description: If there is more than one object in PObj (using PNext) arrange the UV values domains to a one continuous non overlapping domain.

10.2.73 IRndrVMPutPixel (vis_maps.c:848)

put pixel

```
void IRndrVMPutPixel(IRndrVMStruct *VisMap,
                     int u,
                     int v,
                     IrtPtType xyzVals,
                     IRndrVisibleValidityType Validity,
                     int BackFaced,
                     IPPolygonStruct *Triangle)
```

VisMap: IN, OUT, pointer to visibility map.

u: IN, the column number.

v: IN, the line number.

xyzVals: IN, the pixel's view coordinates.

Validity: IN, validity of pixel.

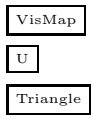
BackFaced: IN, if back face culling is on and pixel belongs to back facing triangle.

Triangle: IN, The triangle which created this UV point.

Returns: void

Description: Manually adds a single pixel to visibility map.

10.2.74 IRndrVMPutTriangle (vis_maps.c:334)




```
int IRndrVMPutTriangle(IRndrVMStruct *VisMap,
                       IRndrTriangleStruct *RendTri,
                       IRndrSceneStruct *Scene,
                       IRndrObjectStruct *Obj,
                       IPPolygonStruct *Triangle)
```

VisMap: IN,OUT, pointer to the visibility map.

RendTri: IN, renderer current rendered triangle.

Scene: IN, scene pointer.

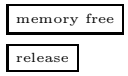
Obj: IN, current rendered object.

Triangle: IN, a polygon triangle represented which was scanned into Z-buffer, with UV values in 'uvvals' attribute, in each vertex. This object is changed during this function (So it can't use for other purposes after this function).

Returns: Whether the method finished successfully

Description: The method gets a scanned triangle polygon and scans it into visibility map using the regular scan conversion

10.2.75 IRndrVMRelease (vis_maps.c:294)




```
void IRndrVMRelease(IRndrVMStruct *VisMap)
```

VisMap: IN,OUT, pointer to the visibility map.

Returns: void

Description: Release the memory taken by the visibility map.

10.2.76 IRndrVMRelocatePtIntoTriangle (vis_maps.c:563)

```
void IRndrVMRelocatePtIntoTriangle(IPPolygonStruct *Triangle, IrtPtType Pt)
```

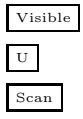
Triangle: IN, The triangle.

Pt: IN, The point.

Returns: void

Description: If Pt isn't inside Triangle, relcoation it to be the closest point inside triangle (The calculation are done only with the xy coordinates).

10.2.77 IRndrVMScan (vis_maps.c:739)




```
void IRndrVMScan(IRndrVMStruct *VisMap, IRndrZBufferStruct *Buff)
```

VisMap: IN,OUT, pointer to the visibility map.

Buff: IN, OUT, the zbuffer.

Returns: void

Description: Scan over all visibility map pixels, for each uv values, contains xyz values is compared to the triangles in the z-buffer in coordinate xy. coordinate can be visible, hidden(mapped) or unmapped.

10.2.78 IRndrVMSetCriticAR (vis_maps.c:212)

```
void IRndrVMSetCriticAR(IRndrVMStruct *VisMap, IrtRType CriticAR)
```

VisMap: IN, OUT, pointer to the visibility map.

CriticAR: IN, value of critic aspect ratio value, which is ratio between the largest and smallest edge of a triangle. use value 0 for skeeping AR check.

Returns: void

Description: Setting critic AR value.

10.2.79 IRndrVMSetDilation (vis_maps.c:236)

```
void IRndrVMSetDilation(IRndrVMStruct *VisMap, int Dilation)
```

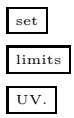
VisMap: IN, OUT, pointer to the visibility map.

Dilation: IN, the amount of iterations to do the dilation algorithm.

Returns: void

Description: Setting dilation iterations value, this will define how many times the dilation algorithm will be executed, expanding one pixel at a time.

10.2.80 IRndrVMSetLimits (vis_maps.c:160)




```
void IRndrVMSetLimits(IRndrVMStruct *VisMap, IPObjectStruct* Objects)
```

VisMap: IN, OUT, Pointer to visibility map object.

Objects: IN, Rendered object list.

Returns: void

Description: Set scene and visibility map limits according to the XY and UV maximum and minimum coordinates in all objects. Also, sets the scene's screen matrix accoridng to the XY limits so the entrie scene will be in the scene's window and centered. That means that the entire scene's x and y coordinates will be in the range [0,scene -> sizex/y] (z coordinate isn't changed).

10.2.81 IRndrVMSetScanOnUV (vis_maps.c:2108)

```
void IRndrVMSetScanOnUV(IRndrVMStruct *VisMap, int IsScanOnUV)
```

VisMap: Pointer to the visibility map.

IsScanOnUV: TRUE in order to scan on XYZ and UV spaces. FALSE to scan only on XYZ space.

Returns: void

Description: Set whether to scan the following triangles on UV space or just on XYZ space.

10.2.82 IRndrVMSetTanAngle (vis_maps.c:187)

```
void IRndrVMSetTanAngle(IRndrVMStruct *VisMap, IrtRType CosAng)
```

VisMap: IN, OUT, pointer to the visibility map.

CosAng: IN, value of critic cosines value of angle between normal and view vector.

Returns: void

Description: Setting critic cosines value of angle between view point to pixel and triangle normal.

10.2.83 IRndrVertexTransform (rndr_lib.c:1290)

```
void IRndrVertexTransform(IRndrPtrType Rend,
                          IPVertexStruct *Vertex,
                          IrtRType *Result)
```

Rend: The rendering context.

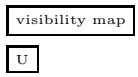
Vertex: IN, pointer to the Vertex.

Result: OUT, the result transformed homogenous coordinate.

Returns: void

Description: Wrapper function for calculating the transformed vertex coordinate.

10.2.84 IRndrVisMapEnable (rndr_lib.c:1105)




```
int IRndrVisMapEnable(IRndrPtrType Rend,
                      IPObjectStruct *Objects,
                      int SuperSize,
                      int UVBackfaceCulling)
```

Rend: IN, OUT, the rendering context.

Objects: IN, Rendered object list.

SuperSize: IN, filter sample super size.

UVBackfaceCulling: IN, use backface culling during the UV scan coversion.

Returns: TRUE if successful.

Description: Enabling visibility map in z-buffer.

10.2.85 IRndrVisMapGetObjDomain (rndr_lib.c:1216)

```
int IRndrVisMapGetObjDomain(IPObjectStruct *PObj,
                            IrtRType *UMin,
                            IrtRType *UMax,
                            IrtRType *VMin,
                            IrtRType *VMax)
```

PObj: IN - The object to get its domain. Objects which aren't polygons are ignored.

UMin, UMax, VMin, VMax: OUT - The domain of the object.

Returns: FALSE if no uv value was found.

Description: Wrapper function to get the domain of the given object (Object which aren't polygons are ignored.

10.2.86 IRndrVisMapPrepareUVValuesOfGeoObj (rndr_lib.c:1247)

```
int IRndrVisMapPrepareUVValuesOfGeoObj(IPObjectStruct *PObj,
                                       int MapWidth,
                                       int MapHeight,
                                       IPObjectStruct *PObj2)
```

PObj: The geometric object.

MapWidth, MapHeight: The dimensions of the visibility map.

PObj2: If it isn't NULL, this object is a list of objects which contains at least as much elements as PObj. Each element i in PObj2 will go through the same Transformations and scaling as element i in PObj. Objects which aren't surfaces, trimmed surfaces or polygons are ignored.

Returns: FALSE if PObj doesn't contain any objects.

Description: Wrapper function for spreading UV values domains.
See also: IRndrVMPrepareUVValuesOfGeoObj,

10.2.87 IRndrVisMapScan (rndr_lib.c:1131)

scan visibility map

```
void IRndrVisMapScan(IRndrPtrType Rend)
```

Rend: IN, OUT, the rendering context.

Returns: void

Description: Wrapper function for scaning visibility map.

10.2.88 IRndrVisMapSetCriticAR (rndr_lib.c:1171)

```
void IRndrVisMapSetCriticAR(IRndrPtrType Rend, IrtRType CriticAR)
```

Rend: IN, OUT, the rendering context.

CriticAR: IN, value of critic aspect ratio value, the ratio between largest and smallest edge of a triangle.

Returns: void

Description: wrapper function for setting critic aspect ratio.

10.2.89 IRndrVisMapSetDilation (rndr_lib.c:1191)

```
void IRndrVisMapSetDilation(IRndrPtrType Rend, int Dilation)
```

Rend: IN, OUT, the rendering context.

Dilation: IN, the amount of iterations to do the dilation algorithm.

Returns: void

Description: wrapper function for setting delation iterations number for white color hiding in visibility maps.

10.2.90 IRndrVisMapSetScanOnUV (rndr_lib.c:1268)

```
void IRndrVisMapSetScanOnUV(IRndrPtrType Rend, int IsScanOnUV)
```

Rend: The rendering context.

IsScanOnUV: TRUE in order to scan on XYZ and UV spaces. FALSE to scan only on XYZ space.

Returns: void

Description: Wrapper function for setting whether to scan the following triangles on both UV space and XYZ space or only on XYZ space.

10.2.91 IRndrVisMapSetTanAngle (rndr_lib.c:1151)

```
void IRndrVisMapSetTanAngle(IRndrPtrType Rend, IrtRType CosAng)
```

Rend: IN, OUT, the rendering context.

CosAng: IN, value of critic cosinus value of angle between normal and view vector.

Returns: void

Description: wrapper function for setting critic tangency angle.

10.2.92 InterpolCopy (interpol.c:29)

Interpol structure
interpolation

```
IRndrInterpolStruct *InterpolCopy(IRndrInterpolStruct *Dst,
                                  IRndrInterpolStruct *Src)
```

Dst: OUT, pointer to destination object.

Src: IN, pointer to source object.

Returns: Pointer to destination object.

Description: Performs COPY operation on object of Interpol type, which contains data to be interpolated during polygons scan converting. Interpol objects has some sort of linked structure, so copy is tricky.

10.2.93 InterpolDelta (interpol.c:64)

Interpol
interpolation

```
IRndrInterpolStruct *InterpolDelta(IRndrInterpolStruct *Dst,
                                   IRndrInterpolStruct *v1,
                                   IRndrInterpolStruct *v2,
                                   IrtRType d)
```

Dst: OUT, pointer to delta(increment) object to be initialized.

v1: IN, pointer to the first Interpol object.

v2: IN, pointer to the second Interpol object.

d: IN, scaling factor determined by dimension of the polygon on the current scan line. *

Returns: pointer to dst object.

Description: Initialize object of Interpol type to be an increment in interpolation between first and second Interpol objects in scan conversion process.

10.2.94 InterpolIncr (interpol.c:117)

Interpol
interpolation

```
IRndrInterpolStruct *InterpolIncr(IRndrInterpolStruct *Dst,
                                  IRndrInterpolStruct *d)
```

Dst: IN OUT, pointer to object to be incremented.

d: IN, pointer to delta object.

Returns: pointer to destination object.

Description: Increments destination Interpol object by delta Interpol object computed by "InterpolDelta" call. By that way we progress interpolation process.

10.2.95 LightIntensivity (color.c:64)

specular
shading

```
void LightIntensivity(IRndrLightStruct *l,
                      const IrtPtType p,
                      const IrtNrmlType n,
                      const IRndrObjectStruct *o,
                      IRndrSceneStruct *Scene,
                      IRndrIntensivityStruct *i)
```

l: IN, pointer to the light source object.

p: IN, point for which intensivity is computing.

n: IN, normal to the surface in the point "p".

o: IN, pointer to the object with surface characteristics.

Scene: IN, pointer to the scene the object belongs.

i: OUT, pointer to resulting intensivity object.

Returns: void

Description: Computes intensivity diffuse and specular values using specular model of illumination (see Foily, Van Dam). Differs between point and vector viewer, point and vector light sources, which is defined in Light object and by calling IS_VIEWER_POINT() function.

10.2.96 LightListAdd (lights.c:45)

```
void LightListAdd(IRndrLightListStruct *Lights, IRndrLightStruct *NewSrc)
```

Lights: OUT, pointer to LightList object which is initialized through.

NewSrc: IN, pointer to Light source.

Returns: void

Description: Adds a new light source to the list.

10.2.97 LightListInitEmpty (lights.c:25)

```
void LightListInitEmpty(IRndrLightListStruct *Lights)
```

Lights: OUT, pointer to LightList object which is initialized through.

Returns: void

Description: Creates an empty light sources list.

10.2.98 LineSegmentEnd (polyline.c:253)

line termination

```
void LineSegmentEnd(IRndrLineSegmentStruct *Seg)
```

Seg: IN, OUT, pointer to the line segment.

Returns: void

Description: Ends a line. Should be called when after the last point was added.

10.2.99 LineSegmentGetTri (polyline.c:278)

triangulation

```
IPPolygonStruct *LineSegmentGetTri(IRndrLineSegmentStruct *Seg, int NumTri)
```

Seg: IN, OUT, pointer to the line segment.

NumTri: IN, the number of triangle, should be < TrianglesNum.

Returns: The triangle.

Description: Retrieves the triangles compromising the current segment.

10.2.100 LineSegmentInit (polyline.c:28)

Initialize Options

```
void LineSegmentInit(IRndrLineSegmentStruct *Seg,
                     IRndrPolylineOptionsStruct *PolyOptions)
```

Seg: IN, OUT, pointer to the line segment.

PolyOptions: IN, the polyline options structure.

Returns: void

Description: Initialize the segment structure, using the PolylineOptions. Should be called when object is created.

10.2.101 LineSegmentRelease (polyline.c:95)

Release free Options

```
void LineSegmentRelease(IRndrLineSegmentStruct *Seg)
```

Seg: IN, OUT, pointer to the line segment.

Returns: void

Description: Frees the memory allocated by the object.

10.2.102 LineSegmentSet (polyline.c:133)

add point line-to

```
void LineSegmentSet(IRndrLineSegmentStruct *Seg, IrtPtType4 Vertex)
```

Seg: IN, OUT, pointer to the line segment.

Vertex: IN, the new point.

Returns: void

Description: Sets the next point for the line.

10.2.103 LineSegmentSetOptions (polyline.c:69)

Initialize Options

```
void LineSegmentSetOptions(IRndrLineSegmentStruct *Seg,
                           IRndrPolylineOptionsStruct *PolyOptions)
```

Seg: IN, OUT, pointer to the line segment.

PolyOptions: IN, the polyline options structure.

Returns: void

Description: Changes the PolyOptions.

10.2.104 LineSegmentStart (polyline.c:113)

begin start

```
void LineSegmentStart(IRndrLineSegmentStruct *Seg)
```

Seg: IN, OUT, pointer to the line segment.

Returns: void

Description: Begins a new line.

10.2.105 ObjectInit (object.c:321)

```
void ObjectInit(IRndrObjectStruct *PObject)
```

PObject: IN, OUT, pointer to the Object structure.

Returns: void

Description: Creates a new blank object. Should be called before the first time the object is used.

10.2.106 ObjectRelease (object.c:342)

```
void ObjectRelease(IRndrObjectStruct *PObject)
```

PObject: IN, OUT, pointer to the Object structure.

Returns: void

Description: Releases memory allocated by object.

10.2.107 ObjectSet (object.c:364)

```
IRndrObjectStruct *ObjectSet(IRndrObjectStruct *PObject,
                             IPObjectStruct *Obj,
                             IRndrSceneStruct *Scene)
```

PObject: IN, OUT, pointer to the Object structure.

Obj: IN, pointer to the Irit object, containing the attributes.

Scene: IN, pointer to the scene.

Returns: Created object.

Description: Wraps an Irit object by Initializing different attributes from the it: color, specularity, transparancy, texture image, volumetric texture.

10.2.108 SceneGetClippingPlane (scene.c:123)

clipping planes

```
void SceneGetClippingPlane(IRndrSceneStruct *Scene,
                           int Axis,
                           int Min,
                           IrtPlnType Result)
```

Scene: IN, pointer to the scene.

Axis: IN, the axis normal to the plane(X_AXIS, Y_AXIS, Z_AXIS).

Min: IN, whether the "near" or "far" clipping planes.

Result: OUT, the result plane.

Returns: void

Description: Retrives one of the clipping plane defining the view frustrum. The planes are built so that inside the view frustum the half planes are positive (and negative outside).

10.2.109 SceneGetMatrices (scene.c:235)

```
struct IRndrMatrixContextStruct *SceneGetMatrices(IRndrSceneStruct *Scene)
```

Scene: IN, pointer to the scene.

Returns: pointer to matrices struct.

Description: Get the scene matrices.

10.2.110 SceneRelease (scene.c:217)

memory free

release

```
void SceneRelease(IRndrSceneStruct *Scene)
```

Scene: IN, pointer to the scene.

Returns: void

Description: Free the memory of the scene.

10.2.111 SceneSetMatrices (scene.c:31)

matrix

view

perspective

```
void SceneSetMatrices(IRndrSceneStruct *Scene,
                      IrtHmgnMatType ViewMat,
                      IrtHmgnMatType PrspMat,
                      IrtHmgnMatType ScrnMat)
```

Scene: OUT, pointer to the scene.

ViewMat: IN, the view matrix.

PrspMat: IN, the perspective matrix, NULL if parallel projection.

ScrnMat: IN, the mapping to the screen or NULL if scale [-1,+1] to image size.

Returns: void

Description: Sets the view and perspective matrices for the scene.

10.2.112 SceneSetZClippingPlanes (scene.c:190)

clipping planes

```
void SceneSetZClippingPlanes(IRndrSceneStruct *Scene,
                             IrtRType ZNear,
                             IrtRType ZFar)
```

Scene: IN, pointer to the scene.

ZNear, ZFar: IN, the near and far z-coordinate of the planes.

Returns: void

Description: Sets the near and far XY clipping planes.

10.2.113 StencilOp (stencil.c:128)

stencil

stencil operation

opengl

```
void StencilOp(IRndrStencilCfgStruct *SCfg, int *SPtr, IRndrStencilOpType Op)
```

SCfg: IN, stencil configuration to be used.

SPtr: IN, pointer to target stencil buffer value.

Op: IN stencil operation to perform.

Returns: void

Description: Performs stencil operation.

10.2.114 StencilOpFail (stencil.c:68)

stencil

stencil operation

opengl

```
void StencilOpFail(IRndrStencilCfgStruct *StencilCfg, int *StencilPtr)
```

StencilCfg: IN, stencil configuration to be used.

StencilPtr: IN, pointer to target stencil buffer value.

Returns: void

Description: Performs stencil operation in case of stencil test failure.

10.2.115 StencilOpZFail (stencil.c:88)

stencil

stencil operation

opengl

```
void StencilOpZFail(IRndrStencilCfgStruct* StencilCfg, int *StencilPtr)
```

StencilCfg: IN, stencil configuration to be used.

StencilPtr: IN, pointer to target stencil buffer value.

Returns: void

Description: Performs stencil operation in case of stencil test success and Z test failure.

10.2.116 StencilOpZPass (stencil.c:108)

stencil

stencil operation

opengl

```
void StencilOpZPass(IRndrStencilCfgStruct* StencilCfg, int *StencilPtr)
```

StencilCfg: IN, stencil configuration to be used.

StencilPtr: IN, pointer to target stencil buffer value.

Returns: void

Description: Performs stencil operation in case of stencil test pass and Z test pass.

10.2.117 StencilTest (stencil.c:28)

stencil

stencil test

opengl

```
IrtBType StencilTest(IRndrStencilCfgStruct *StencilCfg, int Stencil)
```

StencilCfg: IN, stencil configuration to be used.

Stencil: IN, stecil buffer value to test.

Returns: comparison result.

Description: Performs stencil tests.

10.2.118 TextureBumpChocolate (texture.c:814)

```
void TextureBumpChocolate(IRndrTextureStruct *Txtr,
                          IrtPtType Point,
                          IrtNrmlType Normal,
                          IrtRType *Uv,
                          IRndrColorType Color)
```

Txtr: IN, pointer to the texture structure.

Point: IN, coordinate of the point in object space.

Normal: IN, OUT, normal at the point.

Uv: IN, uv parameteric domain's coordinates, if exists.

Color: IN, OUT, color value at the point.

Returns: void

Description: Evaluates pertubation on color value at given point in order to get "orange" bump texture.
See also: TextureBumpOrange,

10.2.119 TextureBumpOrange (texture.c:766)

```
void TextureBumpOrange(IRndrTextureStruct *Txtr,
                       IrtPtType Point,
                       IrtNrmlType Normal,
                       IrtRType *Uv,
                       IRndrColorType Color)
```

Txtr: IN, pointer to the texture structure.

Point: IN, coordinate of the point in object space.

Normal: IN, OUT, normal at the point.

Uv: IN, uv parameteric domain's coordinates, if exists.

Color: IN, OUT, color value at the point.

Returns: void

Description: Evaluates pertubation on color value at given point in order to get "orange" bump texture.
See also: TextureBumpChocolate,

10.2.120 TextureCamouf (texture.c:706)

```
void TextureCamouf(IRndrTextureStruct *Txtr, IrtPtType Point,
                   IrtNrmlType Normal,
                   IrtRType *Uv,
                   IRndrColorType Color)
```

Txtr: IN, pointer to the texture structure.

Point: IN, coordinate of the point in object space.

Normal: IN, OUT, normal at the point.

Uv: IN, uv parameteric domain's coordinates, if exists.

Color: IN, OUT, color value at the point.

Returns: void

Description: Evaluates pertubation on color value at given point in order to get "camouf" texture.

10.2.121 TextureChecker (texture.c:651)

```
void TextureChecker(IRndrTextureStruct *Txtr,
                    IrtPtType Point,
                    IrtNrmlType Normal,
                    IrtRType *Uv,
                    IRndrColorType Color)
```

Txtr: IN, pointer to the texture structure.

Point: IN, coordinate of the point in object space.

Normal: IN, OUT, normal at the point.

Uv: IN, uv parameteric domain's coordinates, if exists.

Color: IN, OUT, color value at the point.

Returns: void

Description: Evaluates pertubation on color value at given point in order to get "chekcer" texture.

10.2.122 TextureContour (texture.c:894)

texture

image warping

```
void TextureContour(IRndrTextureStruct *Txtr,
                    IrtPtType Point,
                    IrtNrmlType Normal,
                    IrtRType *Uv,
                    IRndrColorType Color)
```

Txtr: IN, pointer to the texture structure.

Point: IN, coordinate of the point in object space.

Normal: IN, OUT, normal at the point.

Uv: IN, uv parameteric domain's coordinates, if exists.

Color: IN, OUT, color value at the point.

Returns: void

Description: Evaluates pertubation on color value at given point in order to get "contour" texture.

10.2.123 TextureContourNormal (texture.c:939)

texture

image warping

```
void TextureContourNormal(IRndrTextureStruct *Txtr,
                          IrtPtType Point,
                          IrtNrmlType Normal,
                          IrtRType *Uv,
                          IRndrColorType Color)
```

Txtr: IN, pointer to the texture structure.

Point: IN, coordinate of the point in object space.

Normal: IN, normal at the point.

Uv: IN, uv parameteric domain's coordinates, if exists.

Color: IN, OUT, color value at the point.

Returns: void

Description: Evaluates pertubation on color value at given point in order to get "ncontour" texture.

10.2.124 TextureCurvature (texture.c:993)

texture

curvature

```
void TextureCurvature(IRndrTextureStruct *Txtr,
                      IrtPtType Point,
                      IrtNrmlType Normal,
                      IrtRType *Uv,
                      IRndrColorType Color)
```

Txtr: IN, pointer to the texture structure.

Point: IN, coordinate of the point in object space.

Normal: IN, normal at the point.

Uv: IN, uv parameteric domain's coordinates, if exists.

Color: IN, OUT, color value at the point.

Returns: void

Description: Evaluates a texture color that is a function of the curvature of the surface, getting the "curvature" texture.

10.2.125 TextureImageGetPixel (texture.c:83)

image

texture

```
IrtImgPixelStruct *TextureImageGetPixel(IRndrTextureStruct *Txtr,
                                        IRndrImageStruct *i,
                                        IrtPtType p,
                                        IrtRType v,
                                        IrtRType u,
                                        IPPolygonStruct *Poly)
```

Txtr: IN, general attributes and modifiers of the texture mapping.

i: IN, pointer to the Image data structure.

p: IN, location in Euclidean space of texture color to evaluate.

v: IN, real coordinate of the image pixel.

u: IN, real coordinate of the image pixel.

Poly: IN, pointer to the polygon.

Returns: value of the image pixel at (u,v) point.

Description: Gets image pixel by two real coordinates u and v from the Image data. Access function.

10.2.126 TextureInitParameters (texture.c:1227)

```
void TextureInitParameters(IRndrTextureStruct *Txtr, const char *pString)
```

Txtr: OUT, IRndrTextureStruct that contains the paramters.

pString: IN, paramters string, taken from the attribute.

Returns: void

Description: Initializes texture-specific paramters for volumetric textures.

10.2.127 TextureMarble (texture.c:513)

texture

image warping

```
void TextureMarble(IRndrTextureStruct *Txtr,
                   IrtPtType Point,
                   IrtNrmlType Normal,
                   IrtRType *Uv,
                   IRndrColorType Color)
```

Txtr: IN, pointer to the texture structure.

Point: IN, coordinate of the point in object space.

Normal: IN, OUT, normal at the point.

Uv: IN, uv parameteric domain's coordinates, if exists.

Color: IN, OUT, color value at the point.

Returns: void

Description: Evaluates pertubation on color value at given point in order to get "marble" texture.

10.2.128 TexturePunky (texture.c:1101)

texture

image warping

```
void TexturePunky(IRndrTextureStruct *Txtr,
                  IrtPtType Point,
                  IrtNrmlType Normal,
                  IrtRType *Uv,
                  IRndrColorType Color)
```

Txtr: IN, pointer to the texture structure.

Point: IN, coordinate of the point in object space.

Normal: IN, OUT, normal at the point.

Uv: IN, uv parameteric domain's coordinates, if exists.

Color: IN, OUT, color value at the point.

Returns: void

Description: Evaluates pertubation on color value at given point in order to get "punky" style texture.

10.2.129 TextureWood (texture.c:562)

```
void TextureWood(IRndrTextureStruct *Txtr,
                 IrtPtType Point,
                 IrtNrmlType Normal,
                 IrtRType *Uv,
                 IRndrColorType Color)
```

Txtr: IN, pointer to the texture structure.

Point: IN, coordinate of the point in object space.

Normal: IN, OUT, normal at the point.

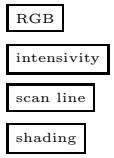
Uv: IN, uv parameteric domain's coordinates, if exists.

Color: IN OUT, color value at the point.

Returns: void

Description: Evaluates pertubation on color value at given point in order to get "wood" texture.

10.2.130 TriangleColorEval (color.c:166)




```
void TriangleColorEval(IPPolygonStruct *Poly,
                       int x,
                       int y,
                       IRndrObjectStruct *o,
                       IRndrSceneStruct *Scene,
                       IRndrInterpolStruct *Value,
                       IRndrColorType r)
```

Poly: IN, the polygon the pixel belongs.

x: IN, scan line pixel position.

y: IN, scan line number.

o: IN, the object of the triangle.

Scene: IN, the scene context.

Value: IN, OUT, interpolation value.

r: OUT, resulting color.

Returns: void

Description: For scan line "y" and pixel "x" in it computes color value in [0, 1] RGB format.

10.2.131 TriangleInit (triangle.c:126)

```
void TriangleInit(IRndrTriangleStruct *Tri)
```

Tri: IN, pointer to the Triangle object.

Returns: void

Description: Creates a new Triangle object and allocates memory for it's fields. Should be called before the first time the object is used.

10.2.132 TriangleRelease (triangle.c:159)

```
void TriangleRelease(IRndrTriangleStruct *Tri)
```

Tri: IN, pointer to the Triangle object.

Returns: void

Description: Free the memory of a the Triangle.

10.2.133 TriangleSet (triangle.c:335)

```
int TriangleSet(IRndrTriangleStruct *Tri,
                IPPolygonStruct *Poly,
                IRndrObjectStruct *o,
                IRndrSceneStruct *Scene)
```

Tri: OUT, pointer to the Triangle object.

Poly: IN, pointer to Irit polygon object.

o: IN, pointer to Object which contains a Triangle and stores various charactarisitics common to every polygon in the object.

Scene: IN, pointer to the scene context.

Returns: 1 in successful, 0 if polygon is not OK.

Description: Wraps an Irit polygon and initialize the Triangle structure from polygon and object data. That includes scan line and interpolation algorithm data initialization.

10.2.134 VertexTransform (triangle.c:48)

```
void VertexTransform(IPVertexStruct *Vertex,
                     IRndrMatrixContextStruct *Matrices,
                     IRndrObjectStruct *o,
                     IrtRType *Result)
```

Vertex: IN, pointer to the Vertex.

Matrices: IN, pointer to matrices context.

o: IN, pointer to Object which contains the Triangle that contains the vertex. If o is NULL it's being ignored.

Result: OUT, the result transformed homogenous coordinate.

Returns: void

Description: Calulate the transformed vertex coordinate.

10.2.135 VisMapCheckValidity (vis_maps.c:1565)

error

```
static int VisMapCheckValidity(IRndrVMStruct *VisMap,
                               IPPolygonStruct *Triangle,
                               IRndrVisibleValidityType *Validity)
```

VisMap: IN, pointer to the visibility map.

Triangle: IN, triangle polygon object.

Validity: OUT, validity of triangle.

Returns: TRUE if successful (All the given parameters weren't NULL, and the triangle has UV values).

Description: Check the validity of the triangle and set Validity accordingly. Assumes that XY values weren't yet switched with UVZ in the triangle.

10.2.136 VisMapFindLimits (vis_maps.c:1960)

scene fitting.

```
static void VisMapFindLimits(IRndrVMStruct *VisMap, IPObjectStruct *OList)
```

VisMap: IN, OUT, pointer to the visibility map.

OList: IN, list of all object to fit in scene.

Returns: void

Description: Enabling scene fitting for current rendering by setting the scene's XY limits and VisMap's UV limits accordig to all of OList's vertices (The scene is stored in VisMap). This function is recursive.

10.2.137 VisMapIsPoorAR (vis_maps.c:1885)

aspect ratio
triangle
AR.

```
static int VisMapIsPoorAR(IRndrVMStruct *VisMap, IrtPtType v1, IrtPtType v2)
```

VisMap: IN, pointer to the visibility map.

v1: IN, vector of one edge of triangle.

v2: IN, vector of second edge of triangle.

Returns: TRUE if aspect ratio is poor.

Description: Checks whether triangle has poor aspect ratio: the ratio between largest and smallest edge.

10.2.138 VisMapIsTangentZAxis (vis_maps.c:1930)

tangent
triangle.

```
static int VisMapIsTangentZAxis(IRndrVMStruct *VisMap, IrtPtType Norm)
```

VisMap: IN, pointer to the visibility map.

Norm: IN, vector normal of triangle.

Returns: TRUE triangle is close to tangent view.

Description: Checks whether triangle is close to tangent view vector. uses VisMap -> CosTanAng, as cos of angle between triangle normal and z axis.

10.2.139 VisMapMakeFittingMatrices (vis_maps.c:2055)

scene fitting.

```
static void VisMapMakeFittingMatrices(IRndrVMStruct *VisMap)
```

VisMap: IN, OUT, pointer to the visibility map.

Returns: void

Description: Setting scene fitting parameters after enabling. Uses the scene's XY limits in order to change the screen matrix so the entire scene will be in the scene's window and centered. That means that the entire scene's x and y coordinates will be in the range [0,scene -> sizex/y] (z coordinate isn't changed).

10.2.140 VisMapRestoreVector (vis_maps.c:1849)

restore
vector
point.

```
static void VisMapRestoreVector(IRndrVMStruct *VisMap, IrtPtType v)
```

VisMap: IN, pointer to the visibility map.

v: IN, OUT: point object to be restore.

Returns: void

Description: Restore vector's world coordinate from view coordinate system.

10.2.141 VisMapSetRefreshLimits (vis_maps.c:2007)

```
static void VisMapSetRefreshLimits(IRndrVMStruct *VisMap,
                                   IPVertexStruct *Vertex)
```

VisMap: IN, OUT, pointer to the visibility map.

Vertex: IN, vertex to update limits with.

Returns: void

Description: Adding to the scene the contribution of Vertex by updating the scene's XY limits values and VisMap's UV limits (The scene is stored in VisMap).

10.2.142 VisMapSwitchTriangleSpaces (vis_maps.c:1487)

space
switch
triangle
scan convention.

```
static int VisMapSwitchTriangleSpaces(IRndrVMStruct *VisMap,
                                      IRndrSceneStruct *Scene,
                                      IRndrObjectStruct *Obj,
                                      IPPolygonStruct *Triangle,
                                      int Reverse)
```

VisMap: IN, Pointer to the visibility map.

Scene: IN, a pointer to the scene struct.

Obj: IN, a pointer to the current rendered object.

Triangle: IN, OUT, triangle polygon object.

Reverse: IN, Whether to switch uv into xy or do the reverse operation.

Returns: TRUE if successful.

Description: Switch Polygon coordinates so it could be scanned with the z buffer scan convention into the visibility map. Simply set the coord field to contain the uv values while the x and y values are backuped in attributes VIS_MAP_X_ATTR_ID and VIS_MAP_Y_ATTR_ID\ At the same time, change the uv space to be [0..size of vismap - 1] (both in coord and in the uv attribute). If Reverse is TRUE, do the opposite operation. The coord values set to the x and y values which are restored from the attributes VIS_MAP_X_ATTR_ID and VIS_MAP_Y_ATTR_ID (the uv values which were in coord are dismissed). If Reverse is TRUE, no change of uv space is done.

10.2.143 VisMapTriangleDZ (vis_maps.c:1667)

error

estimation.

```
static int VisMapTriangleDZ(IRndrVMStruct *VisMap,
                            IPPolygonStruct *Triangle,
                            IRndrVisibleValidityType *Validity,
                            IrtRType *dz)
```

VisMap: IN, pointer to the visibility map.

Triangle: IN, triangle polygon object.

Validity: OUT, validity of triangle.

dz: OUT, z error estimation.

Returns: TRUE if successful.

Description: Estimating z error value in visibility map pixel. to make up on discretization errors. Estimating is made by the partial differentiating of z(x,y) ,x(u,v), and y(u,v) as follow: Dz = (dz/dx)Dx + (dz/dy)Dy Dy = (dy/du)Du + (dy/dv)Dv Dx = (dx/du)Du + (dx/dv)Dv Du = Dv = 1 Where d is partial differentiation, and D means delta value.

10.2.144 VisMapTriangleSet (vis_maps.c:1440)

```
static int VisMapTriangleSet(IRndrTriangleStruct *RendTri,
                             IPPolygonStruct *Triangle,
                             IRndrObjectStruct *Obj,
                             IRndrSceneStruct *Scene,
                             int UVBackfaceCulling)
```

RendTri: OUT, pointer to the Triangle object.

Triangle: IN, pointer to Irit polygon object.

Obj: IN, pointer to Object which contains a Triangle and stores various characteristics common to every polygon in the object.

Scene: IN, pointer to the scene context.

UVBackfaceCulling: IN, use backface culling during the UV scan coversion.

Returns: TRUE when successful.

Description: Sets a uv triangle using DoVisMapCalcs field in Object struct turned on while calling TriangleSet method.

10.2.145 ZBufferClear (zbuffer.c:134)

clear buffer

```
void ZBufferClear(IRndrZBufferStruct *Buffer)
```

Buffer: IN, OUT, Pointer to the z-buffer.

Returns: void

Description: Clears the context of the z-buffer.

10.2.146 ZBufferClearColor (zbuffer.c:1346)

clear

```
void ZBufferClearColor(IRndrZBufferStruct *Buffer)
```

background

color

reset

Buffer: Pointer to the z-buffer.

Returns: void

Description: Routine to clear the color information.

10.2.147 ZBufferClearDepth (zbuffer.c:1296)

Z coordinate

```
void ZBufferClearDepth(IRndrZBufferStruct *Buffer, IRndrZDepthType ClearZ)
```

depth

clear

Buffer: Pointer to the z-buffer.

ClearZ: Depth to clear the ZBuffer to.

Returns: void

Description: Routine to clear the z depth information

10.2.148 ZBufferClearStencil (zbuffer.c:1322)

clear

```
void ZBufferClearStencil(IRndrZBufferStruct *Buffer)
```

Stencil

Buffer: Pointer to the z-buffer.

Returns: void

Description: Routine to clear the Stencil information

10.2.149 ZBufferGetLineColorAlpha (zbuffer.c:1061)

color access

```
void ZBufferGetLineColorAlpha(IRndrZBufferStruct *Buffer,
                              int x0,
                              int x1,
                              int y,
                              IRndrColorType *Result)
```

Buffer: IN, OUT, pointer to the z-buffer

x0: IN, minimal x coordinate.

x1: IN, maximal x coordinate.

y: IN, line number.

Result: OUT, the color values of the line.

Returns: void

Description: Retrieves color information of a specific line. The line should be allocated by the caller.

10.2.150 ZBufferGetLineDepth (zbuffer.c:1132)

z depth access

```
int ZBufferGetLineDepth(IRndrZBufferStruct *Buffer,
                        int x0,
                        int x1,
                        int y,
                        IrtRType *Result)
```

Buffer: IN, OUT, pointer to the z-buffer

x0: IN, minimal x coordinate.

x1: IN, maximal x coordinate.

y: IN, line number.

Result: OUT, the z values of the line.

Returns: whether operation succeded.

Description: Retrives z information of a specific line. The line should be allocated by the caller.

10.2.151 ZBufferGetLineStencil (zbuffer.c:1190)

stencil access

```
int ZBufferGetLineStencil(IRndrZBufferStruct *Buffer,
                          int x0,
                          int x1,
                          int y,
                          int *Result)
```

Buffer: IN, OUT, pointer to the z-buffer.

x0: IN, minimal x coordinate.

x1: IN, maximal x coordinate.

y: IN, line number.

Result: OUT, the stencil values of the line.

Returns: whether operation succeded.

Description: Retrives stencil information of a specific line. The line should be allocated by the caller.

10.2.152 ZBufferInit (zbuffer.c:47)

initialize

create buffer

```
int ZBufferInit(IRndrZBufferStruct *Buffer,
                IRndrSceneStruct *Scene,
                int SuperSize,
                int ColorQuantization)
```

Buffer: IN, OUT, Pointer to the z-buffer.

Scene: IN, Pointer to the related scene object.

SuperSize: IN, Super sampling size.

ColorQuantization: IN, non zero to quantize the generated colors to ColorQuantization levels of colors.

Returns: 0 if successfull.

Description: Initialize a newly created z-buffer.

10.2.153 ZBufferPutPixel (zbuffer.c:185)

put pixel

```
void ZBufferPutPixel(IRndrZBufferStruct *Buffer,
                     int x,
                     int y,
                     IrtRType z,
                     IrtRType Transparency,
                     IRndrColorType Color,
                     IPPolygonStruct *Triangle,
                     VoidPtr ClbkData)
```

Buffer: IN, OUT, pointer to z-buffer.

x: IN, the column number.

y: IN, the line number.

z: IN, the pixel's depth.

Transparency: IN, the pixel's transparency value.

Color: IN, the new color of pixel at (x, y).

Triangle: IN, The triangle which created the added point. *

ClbkData: IN, data to be transfered to call back functions if any.

Returns: void

Description: Manually adds a single pixel.

10.2.154 ZBufferRelease (zbuffer.c:579)

memory free

release

```
void ZBufferRelease(IRndrZBufferStruct *Buffer)
```

Buffer: IN,OUT, pointer to the z-buffer.

Returns: void

Description: Release the memory taken by the z-buffer.

10.2.155 ZBufferSaveFile (zbuffer.c:737)

save

persist

```
void ZBufferSaveFile(IRndrZBufferStruct *Buffer,
                     const char *BaseDirectory,
                     const char *OutFileName,
                     const char *FileType,
                     IRndrZBufferDataType DataType)
```

Buffer: IN, OUT, pointer to the z-buffer.

BaseDirectory: IN, the directory where the file is to be saved.

OutFileName: IN, the file name.

FileType: IN, the file type.

DataType: IN, where to save color/z-depth/stencil data.

Returns: void

Description: Saves the context of the z-buffer into a file or to the functions pointed by ZBufferSaveFileCB.

10.2.156 ZBufferSaveFileCB (zbuffer.c:707)

save

persist

```
void ZBufferSaveFileCB(IRndrZBufferStruct *Buffer,
                       IRndrImgGetTypeFuncType ImgSetType,
                       IRndrImgOpenFuncType ImgOpen,
                       IRndrImgWriteLineFuncType ImgWriteLine,
                       IRndrImgCloseFuncType ImgClose)
```

Buffer: IN, OUT, pointer to the z-buffer.

ImgSetType: IN, Image setting file type function

ImgOpen: IN, Function to open an image file.

ImgWriteLine: IN, Function to write one row (Vec of RGB & vec of Alpha).

ImgClose: IN, Function to close an image file.

Returns: void

Description: Sets call back functions to set image type, open an image, save a row, and close the image, for ZBufferSaveFile.

10.2.157 ZBufferScanTri (zbuffer.c:282)

scan convert

```
void ZBufferScanTri(IRndrZBufferStruct *Buffer,
                    IRndrTriangleStruct *Tri,
                    VoidPtr ClbkData)
```

Buffer: IN, OUT, pointer to the z-buffer.

Tri: IN, pointer to the Triangle object.

ClbkData: IN, data to be transfered to call back functions if any.

Returns: void

Description: Scan converts a triangle object into the z-buffer.

10.2.158 ZBufferScanVMTri (zbuffer.c:544)

```
void ZBufferScanVMTri(IRndrZBufferStruct *Buffer,
                      IRndrTriangleStruct *Tri,
                      VoidPtr ClbkData)
```

Buffer: IN, OUT, pointer to the z-buffer.

Tri: IN, pointer to the Triangle object.

ClbkData: IN, data to be transfered to call back functions if any.

Returns: void

Description: Scan converts a triagle object into the z-buffer, for the visibility map.
See also: ZBufferScanTri,

10.2.159 ZBufferSetFilter (zbuffer.c:1239)

antialias

AntialiasLine

Utah filter package

```
void ZBufferSetFilter(IRndrZBufferStruct *Buffer, char *FilterName)
```

Buffer: Pointer to the z-buffer.

FilterName: String representing the filter name.

Returns: void

Description: Routine to set the filter before any antialias processing could be done.

10.2.160 _IRndrReportError (report.c:50)

report error

```
void _IRndrReportError(const char *Fmt, ...)
```

Fmt: IN, message format, like the "printf" format.

Returns: void

Description: Reports an error message.

10.2.161 _IRndrReportFatal (report.c:75)

report fatal halt

```
void _IRndrReportFatal(const char *Fmt, ...)
```

Fmt: IN, message format, like the "printf" format.

Returns: void

Description: Reports a fatal error and halts.

10.2.162 _IRndrReportWarning (report.c:25)

report warning

```
void _IRndrReportWarning(const char *Fmt, ...)
```

Fmt: IN, message format, like the "printf" format.

Returns: void

Description: Reports a warning message.

Chapter 11

Symbolic Library, symb_lib

11.1 General Information

This library provides a rich set of functions to symbolically manipulate freeform curves and surfaces. This library heavily depends on the cagd library. Functions are provided to low level add, subtract, and multiply freeform curves and surfaces, to compute fields such as curvature, and to extract singular points such as extremums, zeros, and inflections. High level tools to metamorph curves and surfaces, to compute layout (prisa) of freeform surfaces, to compute offset approximations of curves and surfaces, and to compose curves and surfaces are also provided.

The interface of the library is defined in *include/symb_lib.h*.

This library has its own error handler, which by default prints an error message and exit the program called **SymbFatalError**.

Globals in this library have a prefix of **Symb** for general symbolic routines. Prefix of **Bzr** is used for Bezier routines, and prefix of **Bsp** for Bspline specific routines.

11.2 Library Functions

11.2.1 BspCrvBlossomMult (bsp_sym.c:278)

product

```
CagdCrvStruct *BspCrvBlossomMult(const CagdCrvStruct *Crv1,
                                 const CagdCrvStruct *Crv2)
```

Crv1, Crv2: The two curves to multiply.

Returns: The product Crv1 * Crv2 coordinate-wise.

Description: Given two B-spline curves - multiply them coordinate-wise, using Blossoming. The two curves are assumed compatible (same point type and domain). See also:
"Multiplication as a General Operation for Splines", by Kenji Ueda, in curves and Surfaces in Geometric Design, Laurent, Mehaute and Shumaker (eds.), pp 475-482, A. K. Peters 1994.
"Sliding Windows Algorithm for B-spline Multiplication", by X. Chen, R. F. Riesenfeld, and E. Cohen, Solid and Physical Modeling 2007.

See also: BspCrvMult,

11.2.2 BspCrvMult (bsp_sym.c:133)

product

```
CagdCrvStruct *BspCrvMult(const CagdCrvStruct *CCrv1,
                          const CagdCrvStruct *CCrv2)
```

CCrv1, CCrv2: The two curves to multiply.

Returns: The product Crv1 * Crv2 coordinate-wise.

Description: Given two B-spline curves - multiply them coordinate-wise. The two curves are promoted to same point type before the multiplication can take place. The two curves are assumed to hold the same domain. See also BspMultComputationMethod.

11.2.3 BspMultComputationMethod (bsp_sym.c:102)

product

```
BspMultComputationMethodType BspMultComputationMethod(
                                    BspMultComputationMethodType BspMultMethod)
```

BspMultMethod: See BspMultComputationMethodType. B-spline product is computed by either Bezier decomposition, or setting an interpolation problem, or using Blossoming.

Returns: Previous setting.

Description: Sets method of B-spline product computation.

11.2.4 BspSrfBlossomMult (bsp_sym.c:1088)

product

```
CagdSrfStruct *BspSrfBlossomMult(const CagdSrfStruct *Srf1,
                                 const CagdSrfStruct *Srf2)
```

Srf1, Srf2: The two surfaces to multiply.

Returns: The product Srf1 * Srf2 coordinate-wise.

Description: Given two B-spline surfaces - multiply them coordinate-wise, using Blossoming. The two surfaces are assumed compatible (same point type & domain). See also:
"Multiplication as a General Operation for Splines", by Kenji Ueda, in surfaces and Surfaces in Geometric Design, Laurent, Mehaute and Shumaker (eds.), pp 475-482, A. K. Peters 1994.
"Sliding Windows Algorithm for B-spline Multiplication", by X. Chen, R. F. Riesenfeld, and E. Cohen, Solid and Physical Modeling 2007.
See also: BspSrfMult,

11.2.5 BspSrfFactorBilinear (bsp_sym.c:1489)

```
CagdSrfStruct *BspSrfFactorBilinear(const CagdSrfStruct *Srf,
                                    const CagdRType *A)
```

Srf: To factor out a bilinear factor from.

A: Four coeficients of the scalar bilinear.

Returns: Srf / {A[0] (1-u)(1-v) + A[1] u(1-v) + A[2] (1-u)v + A[3] uv}.

Description: Factors out a given bilinear term from a scalar surface, assuming it has this term.

```
S(u,v) / {A[0] (1-u)(1-v) + A[1] u(1-v) + A[2] (1-u)v + A[3] uv}
```

Note that typically a B-spline surface will not have this bilinear in all its patches so use this function with care - this function does not verify this existence.
See also: BzrSrfFactorUMinusV, BzrSrfFactorBilinear, MvarMVFactorUMinV,

11.2.6 BspSrfFactorUMinusV (bsp_sym.c:1634)

```
CagdSrfStruct *BspSrfFactorUMinusV(const CagdSrfStruct *Srf)
```

Srf: To factor out a (u - v) term from.

Returns: Srf / (u - v).

Description: Factors out a (u - v) term from a surface, assuming it has one. Note that typically a B-spline surface will not have (u, v) in all its patches so use this function with care - this function does not verify this existence. It is more common to have (u, v) only along symmetric diagonal patches of the B-spline surface, after symbolic operations like C1(u) - C2(v).
See also: BzrSrfFactorUMinusV, BzrSrfFactorBilinear, MvarMVFactorUMinV,

11.2.7 BspSrfMult (bsp_sym.c:891)

product

```
CagdSrfStruct *BspSrfMult(const CagdSrfStruct *CSrf1,
                          const CagdSrfStruct *CSrf2)
```

CSrf1, CSrf2: The two surfaces to multiply.

Returns: The product Srf1 * Srf2 coordinate-wise.

Description: Given two B-spline surfaces - multiply them coordinate-wise. The two surfaces are promoted to same point type before multiplication can take place. The two surfaces are assumed to hold the same domain. See also BspMultComputationMethod.

11.2.8 BzrComposeCrvCrv (composit.c:447)

composition

```
CagdCrvStruct *BzrComposeCrvCrv(const CagdCrvStruct *Crv1,
                                const CagdCrvStruct *Crv2)
```

Crv1, Crv2: The two curve to compose together.

Returns: The composed curve.

Description: Given two Bezier curves, Crv1 and Crv2, computes their composition Crv1(Crv2(t)). Crv2 must be a scalar curve completely contained in Crv1's parametric domain. See: "Freeform surface analysis using a hybrid of symbolic and numeric computation" by Gershon Elber, PhD thesis, University of Utah, 1992.
See also: SymbDecomposeCrvCrv, BzrComposeCrvCrv,

11.2.9 BzrComposeSrfCrv (composit.c:1412)

composition

```
CagdCrvStruct *BzrComposeSrfCrv(const CagdSrfStruct *Srf,
                                const CagdCrvStruct *Crv)
```

Srf, Crv: The curve and surface to compose.

Returns: The resulting composition.

Description: Given a Bezier curve Crv and a Bezier surface Srf, computes their composition Srf(Crv(t)). Crv must be a two dimensional curve completely contained in the parametric domain of Srf. See: "Freeform surface analysis using a hybrid of symbolic and numeric computation" by Gershon Elber, PhD thesis, University of Utah, 1992.

11.2.10 BzrComposeSrfSrf (compost2.c:213)

composition

```
CagdSrfStruct *BzrComposeSrfSrf(const CagdSrfStruct *Srf1,
                                const CagdSrfStruct *Srf2)
```

Srf1, Srf2: The two surfaces to compose.

Returns: The resulting composition.

Description: Given surfaces Srf1 and Srf2, computes their composition Srf1(Srf2(a, b)). Srf1 must be a Bezier. Srf2 must be a two dimensional surface completely contained in the parametric domain of Srf1. See: "Freeform surface analysis using a hybrid of symbolic and numeric computation" by Gershon Elber, PhD thesis, University of Utah, 1992. Compute the compositions by the products of:

```
S(u, v) = S(u(a, b), v(a, b))


           n   m
        = sum sum Pij Bi(u(a, b)) Bj(v(a, b))
          i=0 j=0
```

```
   n   m       n           i              n-i
= sum sum Pij ( ) (u(a, b))  (1 - u(a, b))
  i=0 j=0     i
                 m           j             m-j
                ( ) (v(a, b))  (1 - v(a, b))
                 j
```

or if Srf2 is rational:

```
S(u, v) = S(u(a, b), v(a, b))
```

```
   n   m
= sum sum Pij Bi(u(a, b)/w(a, b)) Bj(v(a, b)/w(a, b))
  i=0 j=0
```

```
   n   m       n           i                    n-i
= sum sum Pij ( ) (u(a, b))  (w(a, b) - u(a, b))
  i=0 j=0     i
                 m           j                    m-j
                ( ) (v(a, b))  (w(a, b) - v(a, b))    /
                 j
                                   n+m
                           (w(a, b))
```

See also: SymbComposeSrfSrf, MvarBzrComposeTVSrf,

11.2.11 BzrCrvFactor1MinusT (bzr_sym.c:535)

```
CagdCrvStruct *BzrCrvFactor1MinusT(const CagdCrvStruct *Crv)
```

Crv: To factor out a (1-t) term from.

Returns: Crv / (1-t).

Description: Factors out a (1-t) term from a scalar curve, assuming it has one.
See also: BzrCrvFactorT, BspSrfFactorUMinusV, BzrSrfFactorBilinear,

11.2.12 BzrCrvFactorT (bzr_sym.c:484)

```
CagdCrvStruct *BzrCrvFactorT(const CagdCrvStruct *Crv)
```

Crv: A bezier curve to factor out a (t) term from.

Returns: Crv / (t).

Description: Factors out a (t) term from a scalar curve, assuming it has one.
See also: BzrCrvFactor1MinusT, BspSrfFactorUMinusV, BzrSrfFactorBilinear,

11.2.13 BzrCrvMult (bzr_sym.c:59)

product

```
CagdCrvStruct *BzrCrvMult(const CagdCrvStruct *CCrv1,
                          const CagdCrvStruct *CCrv2)
```

CCrv1, CCrv2: The two curves to multiply.

Returns: The product Crv1 * Crv2 coordinate-wise.

Description: Given two Bezier curves - multiply them coordinate-wise. The two curves are promoted to same point type before the multiplication can take place.
See also: BzrCrvMultPtsVecs,

11.2.14 BzrCrvMultList (bzr_sym.c:247)

product

```
CagdCrvStruct *BzrCrvMultList(const CagdCrvStruct *Crv1Lst,
                              const CagdCrvStruct *Crv2Lst)
```

Crv1Lst: First list of Bezier curves to multiply.

Crv2Lst: Second list of Bezier curves to multiply.

Returns: A list of product curves

Description: Given two Bezier curve lists - multiply them one at a time. Return a Bezier curve lists representing their products.

11.2.15 BzrCrvMultPtsVecs (bzr_sym.c:180)

product

```
void BzrCrvMultPtsVecs(const CagdRType *Pts1,
                       int Order1,
                       const CagdRType *Pts2,
                       int Order2,
                       CagdRType *ProdPts)
```

Pts1: First vector of scalars of first Bezier curve.

Order1: Order of first Bezier curve.

Pts2: Second vector of scalars of second Bezier curve.

Order2: Order of second Bezier curve.

ProdPts: Result vector of scalars of product Bezier curve. Result vector is of length Order1+Order2-1.

Returns: void

Description: Given two Bezier scalar curves as vectors Ptsi of orders Orderi, multiply them.
See also: BzrCrvMult,

11.2.16 BzrSrfFactorBilinear (bzr_sym.c:594)

```
CagdSrfStruct *BzrSrfFactorBilinear(const CagdSrfStruct *Srf,
                                    const CagdRType *A)
```

Srf: To factor out a bilinear term from.

A: Four coefficients of the scalar bilinear.

Returns: Srf / (u - v).

Description: Factors out a given bilinear term from a scalar surface, assuming it has this term.

```
S(u,v) / {A[0] (1-u)(1-v) + A[1] u(1-v) + A[2] (1-u)v + A[3] uv}
```

```
If S(P) = Bilinear(A) * S(Q), then
A[0] (m-i) (n-j) Q[i][j]   + A[1] i (n-j) Q[i-1][j] +
A[2] (m-i) j     Q[i][j-1] + A[3] i j     Q[i-1][j-1] = m n P[i][j]
```

See also: BspSrfFactorUMinusV,

11.2.17 BzrSrfFactorExtremeRowCol (bzr_sym.c:906)

```
CagdSrfStruct *BzrSrfFactorExtremeRowCol(const CagdSrfStruct *Srf,
                                         CagdSrfBndryType Bndry)
```

Srf: Surface to compute a reduced form for.

Bndry: Boundary along which to reduce.

Returns: Reduced surface, of one degree less in one direction.

Description: Derived a reduced Bezier surface out of the given Bezier surface but performing the following:

1. Removing the row/cloumn near the specified surafce boundary, Bndry.
2. Scaling all other rows/columns by m/i where m is the original order in the direction we remove the row/colimn in 1, and i vanish for the row/column index of the removed row/column. For example, given the Bezier surface:

```
 m    n
sum  sum Pij Bi,m(u) Bj,n(v),
i=0  j=0
```

reducing Bndry = CAGD_U_MAX_BNDRY would yeild,

```
m-1   n    m
sum  sum  --- Pij Bi,m-1(u) Bj,n(v).
i=0  j=0  m-i
```

See also: MvarAdjacentSrfSrfInter,

11.2.18 BzrSrfFactorLowOrders (bzr_sym.c:995)

```
void BzrSrfFactorLowOrders(const CagdSrfStruct *Srf,
                           CagdSrfStruct **S11,
                           CagdSrfStruct **S12,
                           CagdSrfStruct **S21,
                           CagdSrfStruct **S22)
```

Srf: To factor out as four surfaces of lower order as, Srf = (1-u)(1-v) S11 + (1-u)v S12 + u(1-v) S21 + uv S22.

S11, S12, S21, S22: The four lower order surfaces to factor out.

Returns: void

Description: Factors out a given Bezier surface into four Bezier surfaces of one order smaller, as (1-u)(1-v) S11 + (1-u)v S12 + u(1-v) S21 + uv S22. Srf is assumed to be of orders larger than linear.
See also: BspSrfFactorUMinusV, BzrSrfFactorBilinear,

11.2.19 BzrSrfFactorUMinusV (bzr_sym.c:825)

```
CagdSrfStruct *BzrSrfFactorUMinusV(const CagdSrfStruct *Srf)
```

Srf: To factor out a (u - v) term from.

Returns: Srf / (u - v).

Description: Factors out a (u - v) term from a scalar surface, assuming it has one.
See also: BspSrfFactorUMinusV, BzrSrfFactorBilinear,

11.2.20 BzrSrfMult (bzr_sym.c:287)

product

```
CagdSrfStruct *BzrSrfMult(const CagdSrfStruct *CSrf1,
                          const CagdSrfStruct *CSrf2)
```

CSrf1, CSrf2: The two surfaces to multiply.

Returns: The product Srf1 * Srf2 coordinate-wise.

Description: Given two Bezier surfaces - multiply them coordinate-wise. The two surfaces are promoted to same point type before multiplication can take place.

11.2.21 BzrSrfSubdivAtCurve (compost2.c:486)

```
CagdSrfStruct *BzrSrfSubdivAtCurve(const CagdSrfStruct *Srf,
                                   const CagdCrvStruct *DivCrv)
```

Srf: To devide into two new tensor product surfaces along DivCurv. Must be a Bezier surface.

DivCrv: 2D curve in the UV space of Srf. Must split the UV domain of Srf into two and must start/end at opposite boundaries. That is, if DivCrv starts at UMin, it must end at UMax.

Returns: A list of two surfaces resulting from subdividing Srf along splitting curve DivCrv.

Description: Divides given surface Srf into two tensor product surfaces along a general curve DivCrv that splits Srf into two.
See also:

11.2.22 IritSymbDescribeError (symb_err.c:86)

error handling

```
const char *IritSymbDescribeError(IritSymbFatalErrorType ErrorNum)
```

ErrorNum: Type of the error that was raised.

Returns: A string describing the error type.

Description: Returns a string describing a the given error. Errors can be raised by any member of this symb library as well as other users. Raised error will cause an invocation of IritSymbFatalError function which decides how to handle this error. IritSymbFatalError can for example, invoke this routine with the error type, print the appropriate message and quit the program.

11.2.23 IritSymbFatalError (symb_ftl.c:56)

error handling

```
void IritSymbFatalError(IritSymbFatalErrorType ErrID)
```

ErrID: Error type that was raised.

Returns: void

Description: Trap Symb_lib errors right here. Provides a default error handler for the symb library. Gets an error description using IritSymbDescribeError, prints it and exit the program using exit.

11.2.24 IritSymbSetFatalErrorFunc (symb_ftl.c:28)

error handling

```
SymbSetErrorFuncType IritSymbSetFatalErrorFunc(SymbSetErrorFuncType ErrorFunc)
```

ErrorFunc: New error function to use.

Returns: Old error function reference.

Description: Sets the error function to be used by Symb_lib.

11.2.25 Symb2DCrvParamerize2Prms (prm_dmn.c:258)

```
CagdSrfStruct *Symb2DCrvParamerize2Prms(const CagdCrvStruct *Crv,
                                        CagdRType T1,
                                        CagdRType T2,
                                        int Dir,
                                        CagdRType *Error)
```

Crv: The curve to fit a bivariate to the area enclosed in.

T1, T2: Two parameters to split curve at and fit a match in between.

Dir: Direction where Crv1/2 were split (0 None, 1 for U, 2 for V).

Error: Error result in this fit. Error can only be compared to the error of similar invocations of this function.

Returns: Fitted bivariate, or NULL if failed.

Description: Fit a bivariate to the region enclosed by Crv by splitting at T1 and T2.
See also:

11.2.26 Symb2DCrvParameterizeDomain (prm_dmn.c:409)

```
CagdSrfStruct *Symb2DCrvParameterizeDomain(const CagdCrvStruct *UVCrv,
                                           CagdRType Eps)
```

UVCrv: Curve to examine the ability to parameterize its interior domain as a 2D surface.

Eps: Tolerance of the decomposition.

Returns: A planar surface parameterizing the domain enclosed. Typically one, possibly two surfaces, or NULL if failed.

Description: Examine curve Crv if its enclosed domain is bivariate parameterizable.

11.2.27 Symb2DCrvParameterizing2Crvs (prm_dmn.c:194)

```
CagdSrfStruct *Symb2DCrvParameterizing2Crvs(const CagdCrvStruct *Crv1,
                                            const CagdCrvStruct *Crv2,
                                            int Dir,
                                            CagdBType ForceMatch)
```

Crv1, Crv2: Two curves forming a loop 9and sharing starting and ending locations (i.e. they are roughly monotone with respect to each other.)

Dir: Direction where Crv1/2 were split (1 for U, 2 for V).

ForceMatch: Force match, even if the Jacobian might be negative.

Returns: 2D surface parameterizing region enclosed in Crv1/2, or NULL if failed.

Description: Parameterize the area enclosed by given pair of curves that form a closed planar loop.
See also: TrimDecompTrimSrf2Srfs, TrimCrvIsParameterizableDomain,

11.2.28 Symb2DSrfJacobian (symb_srf.c:1006)

normal

symbolic computation

```
CagdSrfStruct *Symb2DSrfJacobian(const CagdSrfStruct *Srf)
```

Srf: To compute the Jacobian scalar field for.

Returns: A scalar field representing the Jacobain of the given planar Srf.

Description: Given a planar surface - compute its Jacobian scalar field surface, as the cross product if its partial derivatives.
See also: SymbSrfNormalSrf, MvarCalculateTVJacobian,

11.2.29 SymbAdapIsoExtract (adap_iso.c:246)

adaptive isocurves

```
CagdCrvStruct *SymbAdapIsoExtract(SymbAdapIsoGenInfoStruct *GI,
                                  const CagdSrfStruct *Srf,
                                  const CagdSrfStruct *NSrf,
                                  SymbAdapIsoDistSqrFuncType AdapIsoDistFunc,
                                  CagdSrfDirType Dir,
                                  CagdRType Eps,
                                  CagdBType FullIso,
                                  CagdBType SinglePath)
```

GI: Local general information of the adap iso structure.

Srf: To compute adaptive isocurve coverage form

NSrf: Normal vector field defining the normals of Srf.

AdapIsoDistFunc: Optional function to invoke with the two adjacent isoparametric curves of the coverage to evaluate the distance between them.

Dir: Direction of adaptive isocurve extraction. Either U or V.

Eps: Tolerance of adaptive isocurve coverage. For every point P on Srf there will be a point Q in one of the extracted isocurves such the |P - Q| < Eps.

FullIso: Do we want all isocurves to span the entire domain?

SinglePath: Do we want a single curve through them all?

Returns: A list of curves representing the computed adaptive isocurve coverage for surface Srf. If normal field, NSrf, is prescribed, normal curves are concatenated alternatively in this list.

Description: Extracts a valid coverage set of isocurves from the given surface in the given direction and epsilon. If FullIso is TRUE, all extracted isocurves are spanning the entire parametric domain. If SinglePath is TRUE, the entire coverage is going to be a single curve. If NSrf != NULL, every second curve will be a vector field curve representing the unnormalized normal for the previous Euclidean curve. This mode disable the SinglePath mode. See also function SymbAdapIsoSetExtractMinLevel.
See also: SymbAdapIsoExtractRectRgns, SymbAdapIsoSetWeightPt, , SymbAdapIsoSkewDistSqr, SymbAdapIsoExtractInit, SymbAdapIsoExtractFree,

11.2.30 SymbAdapIsoExtractFree (adap_iso.c:194)

adaptive isocurves

```
void SymbAdapIsoExtractFree(SymbAdapIsoGenInfoStruct *GI)
```

GI: Local general information of the adap iso structure to free.

Returns: void

Description: Frees the local general info structures needed by the adap iso module.
See also: SymbAdapIsoExtractRectRgns, SymbAdapIsoSetWeightPt, , SymbAdapIsoSkewDistSqr, SymbAdapIsoExtractInit, ,

11.2.31 SymbAdapIsoExtractInit (adap_iso.c:161)

adaptive isocurves

```
SymbAdapIsoGenInfoStruct *SymbAdapIsoExtractInit(void)
```

Returns: Alloated dynamically structure.

Description: Initializes local general info structures needed by the adap iso module.
See also: SymbAdapIsoExtractRectRgns, SymbAdapIsoSetWeightPt, , SymbAdapIsoSkewDistSqr, SymbAdapIsoExtractFree,

11.2.32 SymbAdapIsoExtractRectRgns (adap_iso.c:1142)

adaptive isocurves

```
IPObjectStruct *SymbAdapIsoExtractRectRgns(SymbAdapIsoGenInfoStruct *GI,
                                           const CagdSrfStruct *Srf,
                                           CagdSrfDirType Dir,
                                           CagdRType Size,
                                           int Smoothing,
                                           int OutputType)
```

GI: Local general information of the adap iso structure.

Srf: To compute adaptive rectangular regions' tiles for.

Dir: Direction of adaptive isocurve extraction. Either U or V.

Size: Rough size of the edges of the generated rectangles.

Smoothing: Number of low pass smoothing steps to apply, 0 to disable.

OutputType: 0 for UV coordinates in original surfaces. 1 for polygonal rectangles in Euclidean space. 2 for surface patches in Euclidean space.

Returns: A list of rectangles, as OutputType sets.

Description: Extracts a valid coverage set of rectangular regions of roughly equal size to the given surface in the given direction Dir and size Size.
See also: SymbAdapIsoExtract, SymbAdapIsoSetWeightPt,

11.2.33 SymbAdapIsoSetExtractMinLevel (adap_iso.c:1348)

adaptive isocurves

```
int SymbAdapIsoSetExtractMinLevel(SymbAdapIsoGenInfoStruct *GI, int MinLevel)
```

GI: Local general information of the adap iso structure.

MinLevel: At least that many subdivision will occur.

Returns: Old value of Min subdivision level.

Description: Sets minimum level of subdivision forced in the adaptive iso extraction.

11.2.34 SymbAdapIsoSetWeightPt (adap_iso.c:1380)

```
void SymbAdapIsoSetWeightPt(SymbAdapIsoGenInfoStruct *GI,
                            CagdRType *Pt,
                            CagdRType Scale,
                            CagdRType Width)
```

GI: Local general information of the adap iso structure.

Pt: Point to consider or NULL to disable.

Scale: Scaling factor at Pt compared to a far location from Pt. Must be larger than 1.0 (1.0 has no effect).

Width: Control over the width of the effect.

Returns: void

Description: Sets the point location to consider using the adaptive iso generation.
See also: SymbAdapIsoDistWeightPt, SymbAdapIsoExtract,

11.2.35 SymbAdapIsoSkewDistSqr (adap_iso.c:123)

```
CagdCrvStruct *SymbAdapIsoSkewDistSqr(int Level,
                                      CagdCrvStruct *Crv1,
                                      CagdCrvStruct *NCrv1,
                                      CagdCrvStruct *Crv2,
                                      CagdCrvStruct *NCrv2)
```

Level: Maximum depth of recursion of adaptive iso algorithm.

Crv1: First curve to compute distance field from it to Crv2.

NCrv1: Normal vector field of Crv1.

Crv2: Second curve to compute distance field from it to Crv1.

NCrv2: Normal vector field of Crv2.

Returns: A scalar distance function field.

Description: Computes a distance square function between Crv1 and Crv2 as a scalar field, taking into account the skewing (non orthogonality/conformality) of the parametrization:

$$\text{SkewAdapDist}(t) = \frac{(\ d(Crv1(t) + Crv2(t))/dt \times (Crv1(t) - Crv2(2))\ )^2}{(\ d(Crv1(t) + Crv2(t))/dt\ )^2}$$

See also: SymbAdapIsoExtract,

11.2.36 SymbAlgebraicProdSrf (constrct.c:122)

```
CagdSrfStruct *SymbAlgebraicProdSrf(const CagdCrvStruct *Crv1,
                                    const CagdCrvStruct *Crv2)
```

Crv1, Crv2: Two curves to multiply algebraically.

Returns: A surface represent their product, algebraically.

Description: Multiply up algebraically the given two curves, C1(r) and C2(t). The result is a bivariate surface S(r, t) = C1(r) C2(t).

See also: SymbAlgebraicSumSrf, SymbSwungAlgSumSrf,

11.2.37 SymbAlgebraicSumSrf (constrct.c:86)

```
CagdSrfStruct *SymbAlgebraicSumSrf(const CagdCrvStruct *Crv1,
                                   const CagdCrvStruct *Crv2)
```

Crv1, Crv2: Two curves to sum algebraically.

Returns: A surface represent their sum, algebraically.

Description: Adds up algebraically the given two curves, C1(r) and C2(t). The result is a bivariate surface S(r, t) = C1(r) + C2(t).

See also: SymbSwungAlgSumSrf, SymbAlgebraicProdSrf,

11.2.38 SymbAllPrisaSrfs (prisa.c:58)

layout

prisa

```
CagdSrfStruct *SymbAllPrisaSrfs(const CagdSrfStruct *Srfs,
                                int SamplesPerCurve,
                                CagdRType Epsilon,
                                CagdSrfDirType Dir,
                                const CagdVType Space)
```

Srfs: To approximate and flatten out.

SamplesPerCurve: During the approximation of a ruled surface as a developable surface.

Epsilon: Accuracy control for the piecewise ruled surface approximation. if Epsilon us positive, the surfaces are laid down on the plane, otherwise they are return as 3-space ruled surfaces and form a piecewise ruled-surface approximation to Srfs.

Dir: Direction of ruled/developable surface approximation. Either U or V.

Space: A vector in the XY plane to denote the amount of translation from one flattened out surface to the next.

Returns: A list of planar surfaces denoting the layout (prisa) of the given Srfs to the accuracy requested.

Description: Computes a piecewise ruled surface approximation to a given set of surfaces with given Epsilon, and lay them out "nicely" onto the XY plane, by approximating each ruled surface as a developable surface with SamplesPerCurve samples. Dir controls the direction of ruled approximation, SpaceScale and Offset controls the placement of the different planar pieces. Prisa is the hebrew word for the process of flattening out a three dimensional surface. I have still to find an english word for it.

See also: SymbPiecewiseRuledSrfApprox, SymbPrisaRuledSrf, TrimAllPrisaSrfs, , SymbPrisaGetCrossSections,

11.2.39 SymbApproxCrvAsBzrCubics (bzr_sym.c:1063)

conversion

approximation

```
CagdCrvStruct *SymbApproxCrvAsBzrCubics(const CagdCrvStruct *Crv,
                                        CagdRType Tol,
                                        CagdRType MaxLen)
```

Crv: To approximate using cubic Bezier polynomials.

Tol: Accuracy control.

MaxLen: Maximum arc length of curve.

Returns: A list of cubic Bezier polynomials approximating Crv.

Description: Given a curve - convert it to (possibly) piecewise cubic polynomials. If the curve is

1. A cubic - a copy if it is returned.
2. Lower than cubic - a degree raised (to a cubic) curve is returned.
3. Higher than cubic - a C^1 continuous piecewise cubic polynomials approximation is computed for Crv.
4. Piecewise polynomial Bspline curve - split into polynomial segments.

In case 3, a list of polynomial cubic curves is returned. Tol is then used for the distance tolerance error measure for the approximation. If the total length of control polygon is (approximately) more than MaxLen, the curve is subdivided until this is not the case.

See also: SymbApproxCrvAsBzrQuadratics, SymbCrvCubicApprox, CagdCubicCrvFit,

11.2.40 SymbApproxCrvAsBzrQuadratics (bzr_sym.c:1277)

conversion

approximation

```
CagdCrvStruct *SymbApproxCrvAsBzrQuadratics(const CagdCrvStruct *CCrv,
                                            CagdRType Tol,
                                            CagdRType MaxLen)
```

CCrv: To approximate using quadratic Bezier polynomials.

Tol: Accuracy control.

MaxLen: Maximum arc length of curve.

Returns: A list of quadratic Bezier polynomials approximating Crv.

Description: Given a curve - convert it to (possibly) piecewise quadratic polynomials. If the curve is

1. A quadratic - a copy if it is returned.
2. Lower than quadratic - a degree raised (to a quadratic) curve is returned.
3. Higher than quadratic - a C^1 continuous piecewise quadratic polynomial approximation is computed for Crv.
4. Piecewise polynomial Bspline curve - split into polynomial segments.

In case 3, a list of polynomial quadratic curves is returned. Tol is then used for the distance tolerance error measure for the approximation. If the total length of control polygon is (approximately) more than MaxLen, the curve is subdivided until this is not the case.

See also: SymbApproxCrvAsBzrCubics, CagdQuadraticCrvFit,

11.2.41 SymbArcArrayFree (symb_gen.c:180)

free

```
void SymbArcArrayFree(SymbArcStruct *ArcArray, int Size)
```

ArcArray: To be deallocated.
Size: Of the deallocated array.
Returns: void
Description: Deallocates and frees an array of Arc structure.

11.2.42 SymbArcArrayNew (symb_gen.c:25)

allocation

```
SymbArcStruct *SymbArcArrayNew(int Size)
```

Size: Size of Arc array to allocate.
Returns: An array of Arc structures of size Size.
Description: Allocates and resets all slots of an array of Arc structures.

11.2.43 SymbArcCopy (symb_gen.c:79)

copy

```
SymbArcStruct *SymbArcCopy(SymbArcStruct *Arc)
```

Arc: To be copied.
Returns: A duplicate of Arc.
Description: Allocates and copies all slots of a Arc structure.

11.2.44 SymbArcCopyList (symb_gen.c:104)

copy

```
SymbArcStruct *SymbArcCopyList(SymbArcStruct *ArcList)
```

ArcList: To be copied.
Returns: A duplicated list of arcs.
Description: Allocates and copies a list of arc structures.

11.2.45 SymbArcFree (symb_gen.c:133)

free

```
void SymbArcFree(SymbArcStruct *Arc)
```

Arc: To be deallocated.
Returns: void
Description: Deallocates and frees all slots of an arc structure.

11.2.46 SymbArcFreeList (symb_gen.c:155)

free

```
void SymbArcFreeList(SymbArcStruct *ArcList)
```

ArcList: To be deallocated.
Returns: void
Description: Deallocates and frees an arc structure list:

11.2.47 SymbArcNew (symb_gen.c:53)

allocation

```
SymbArcStruct *SymbArcNew(int Arc)
```

Arc: TRUE for an arc, FALSE for degenerated-arc (a line...).

Returns: A Arc structure.

Description: Allocates and resets all slots of an Arc structure.

11.2.48 SymbArcs2Crvs (biarc.c:593)

```
CagdCrvStruct *SymbArcs2Crvs(const SymbArcStruct *Arcs)
```

Arcs: To convert to real curves' geometry.

Returns: Generated curves.

Description: Converts a list of arcs (and lines) into curves geometry.

11.2.49 SymbBspBasisInnerProd (bsp_sym.c:758)

```
CagdRType SymbBspBasisInnerProd(int Index1, int Index2,
                                CagdCrvStruct *InnerProdCrv1,
                                CagdCrvStruct *InnerProdCrv2)
```

Index1: Index of first basis function.

Index2: Index of second basis function.

InnerProdCrv1: Pointer to first basis function.

InnerProdCrv2: Pointer to second basis function.

Returns: The value of the inner product.

Description: Computes the inner product of two basis functions over a similar function space, as created via SymbBspBasisInnerProdPrep. The inner product is defined as "int(B1(t) * B2(t))" where "int (.)" denotes the integral of the function over all the domain.
See also: SymbBspBasisInnerProdPrep, SymbBspBasisInnerProdMat, SymbBspBasisInnerProd2,

11.2.50 SymbBspBasisInnerProd2 (bspiprod.c:67)

```
CagdRType SymbBspBasisInnerProd2(const CagdRType *KV,
                                 int Len,
                                 int Order1,
                                 int Order2,
                                 int Index1,
                                 int Index2)
```

KV: A common knot vector of the b-spline basis functions.

Len: Length of knot vector KV.

Order1: Order of first basis function.

Order2: Order of second basis function.

Index1: Index of first basis function.

Index2: Index of second basis function.

Returns: The value of the inner product.

Description: Computes the inner product of two B-spline basis functions over a similar function space. The inner product is defined as "int(B1(t) * B2(t))" where "int (.)" denotes the integral of the function over all the domain. The computation is conducted recursively over the orders until Order1/2 are constant.
See also: SymbBspBasisInnerProd,

11.2.51 SymbBspBasisInnerProdMat (bsp_sym.c:601)

```
CagdRType **SymbBspBasisInnerProdMat(const CagdRType *KV,
                                     int Len,
                                     int Order1,
                                     int Order2)
```

KV: Knot vector of the basis functions (of Order).

Len: Length of knot vector KV.

Order1: Order of first basis function.

Order2: Order of second basis function, <= Order1.

Returns: The allocated matrix and values of inner products.

Description: Computes a matrix of size (Len x (Len - (Order1 - Order2)) of inner products, SymbBspBasisInnerProd style. matrix is allocated dynamically.

See also: SymbBspBasisInnerProd, SymbBspBasisInnerProdPrep,

11.2.52 SymbBspBasisInnerProdPrep (bsp_sym.c:667)

```
void SymbBspBasisInnerProdPrep(const CagdRType *KV,
                               int Len,
                               int Order1,
                               int Order2,
                               CagdCrvStruct **InnerProdCrv1,
                               CagdCrvStruct **InnerProdCrv2)
```

KV: Knot vector of the basis functions (of Order).

Len: Length of knot vector KV.

Order1: Order of first basis function.

Order2: Order of second basis function, <= Order1.

InnerProdCrv1, InnerProdCrv2: Where to save the computed inner products.

Returns: void

Description: Prepares for the computation of the inner product of pair of basis functions over a similar function space. The inner product is defined as "int(B1(t) * B2(t))" where "int (.)" denotes the integral of the function over all the domain.

See also: SymbBspBasisInnerProd, SymbBspBasisInnerProdPrep2,

11.2.53 SymbBspBasisInnerProdPrep2 (bsp_sym.c:717)

```
void SymbBspBasisInnerProdPrep2(const CagdRType *KV1,
                                const CagdRType *KV2,
                                int Len1,
                                int Len2,
                                int Order1,
                                int Order2,
                                CagdCrvStruct **InnerProdCrv1,
                                CagdCrvStruct **InnerProdCrv2)
```

KV1: Knot vector of the first basis functions (of Order1).

KV2: Knot vector of the second basis functions (of Order2).

Len1: Length of knot vector KV1.

Len2: Length of knot vector KV2.

Order1: Order of first basis function.

Order2: Order of second basis function, <= Order1.

InnerProdCrv1: Pointer to first basis function.

InnerProdCrv2: Pointer to second basis function.

Returns: void

Description: Prepares for the computation of the inner product of pair of basis functions over a similar function space. The inner product is defined as "int(B1(t) * B2(t))" where "int (.)" denotes the integral of the function over all the domain.

See also: SymbBspBasisInnerProd2, SymbBspBasisInnerProdPrep,

11.2.54 SymbBzrDegReduce (cmp_crvs.c:136)

```
CagdCrvStruct *SymbBzrDegReduce(const CagdCrvStruct *Crv, CagdRType Eps)
```

Crv: A Bezier curve.

Eps: A threshold for degree reduction computations.

Returns: Maximal degree reduced curve. If no degree reduction was done, returns NULL.

Description: For given Bezier curve, performs maximal degree reduction of the curve. Maximal degree reduction is defined as a process of degree redusing the curve as long as the new curve represents the same curve.
See also: BzrCrvDegreeRaise, BzrCrvDegreeReduce,

11.2.55 SymbCanonicBzrCrv (cmp_crvs.c:72)

```
CagdCrvStruct *SymbCanonicBzrCrv(const CagdCrvStruct *Crv, CagdRType Eps)
```

Crv: A Bezier curve.

Eps: A threshold for "canonical representation" computations.

Returns: Canonically represented Bezier curve.

Description: For given Bezier curve, returns its irreducible version (its canonical representation), by reversing the processes of degree raising and composition.

11.2.56 SymbCircTanTo2Crvs (crv_tans.c:345)

bi-tangent

```
CagdPtStruct *SymbCircTanTo2Crvs(const CagdCrvStruct *Crv1,
                                 const CagdCrvStruct *Crv2,
                                 CagdRType Radius,
                                 CagdRType Tol)
```

Crv1, Crv2: The two curves to find the circle that is tangent to both.

Radius: Of the circle that is tangent to Crv1/2.

Tol: Tolerance of approximation.

Returns: List of the centers of bi-tangent circles. Each such point also contains a "Params" attribute with the two parameter values of the two curves.

Description: Computes all circles of prescribed radius that are tangent to given two curves. Compute the offset of +/-R to the two curves and intersect the pairs.
See also: SymbTangentToCrvAtTwoPts, MvarCircTanTo2Crvs,

11.2.57 SymbClipCrvToSrfDomain (blending.c:30)

```
CagdCrvStruct *SymbClipCrvToSrfDomain(const CagdSrfStruct *Srf,
                                      const CagdCrvStruct *UVCrv)
```

Srf: Surface to clip UVCrv to its parametric domain.

UVCrv: The curve to clip.

Returns: Clipped curve.

Description: Clips the given curve to the domain prescribed by Srf.

11.2.58 SymbComposeCrvCrv (composit.c:118)

composition

```
CagdCrvStruct *SymbComposeCrvCrv(const CagdCrvStruct *Crv1,
                                 const CagdCrvStruct *Crv2)
```

Crv1, Crv2: The two curves to compose together.

Returns: The composed curve.

Description: Given two curves, Crv1 and Crv2, computes the composition Crv1(Crv2(t)). Crv2 must be a scalar curve completely contained in Crv1's parametric domain.

See also: BzrComposeCrvCrv, SymbDecomposeCrvCrv, SymbComposePeriodicCrvCrv, SymbComposeSrfCrv, MvarComposeTVSrf,

11.2.59 SymbComposePeriodicCrvCrv (composit.c:543)

```
CagdCrvStruct *SymbComposePeriodicCrvCrv(const CagdCrvStruct *CCrv1,
                                         const CagdCrvStruct *CCrv2,
                                         CagdRType Epsilon)
```

CCrv1, CCrv2: The two curves to compose together.

Epsilon: Of subdivision at boundary crossing locations.

Returns: The composed curve.

Description: Given two curves, Crv1 and Crv2, computes the composition Crv1(Crv2(t)). Crv2 must be a scalar curve that is clipped and tiled modulus the domain of Crv1. As an example, if Crv1's domain is [1, 5), and Crv2's range is [-3, 11), then Crv2 will be clipped to the following pieces:

```
1. [-3, 1) that is remapped to [1, 5) and composed.
2. [ 1, 5) that is composed as is.
3. [ 5, 9) that is remapped to [1, 5) and composed.
4. [ 9, 11) that is remapped to [1, 3) and composed.
```

See also: SymbComposeCrvCrv,

11.2.60 SymbComposePeriodicSrfCrv (composit.c:1174)

composition

```
CagdCrvStruct *SymbComposePeriodicSrfCrv(const CagdSrfStruct *Srf,
                                         const CagdCrvStruct *Crv,
                                         CagdRType Epsilon)
```

Srf, Crv: The curve and periodic surface to compose together.

Epsilon: Of subdivision at boundary crossing locations.

Returns: The composed curve.

Description: Given a curve Crv and surface Srf, computes the composition Srf(Crv(t)). Crv must be a two dimensional curve that is clipped and tiled modulus the domain of Srf. As an example, if Surface V domain is [1, 5), and Crv Y range [-3, 11), then Crv will be V clipped to the following pieces:

```
1. [-3, 1) that is remapped to [1, 5) and composed.
2. [ 1, 5) that is composed as is.
3. [ 5, 9) that is remapped to [1, 5) and composed.
4. [ 9, 11) that is remapped to [1, 3) and composed.
```

See also: SymbComposeSrfCrv,

11.2.61 SymbComposeSrfClrCache (composit.c:662)

composition

```
void SymbComposeSrfClrCache(const CagdSrfStruct *Srf)
```

Srf: Surface to clean all its cached data for SymbComposeSrfCrv.

Returns: void

Description: Given a surface Srf that was used in a composition operation with some curve via SymbComposeSrfCrv while SymbComposeSrfSetCache was on, cleans all cached informaation on the surface.

See also: SymbComposeSrfCrv, SymbComposeSrfClrCache,

11.2.62 SymbComposeSrfCrv (composit.c:730)

composition

```
CagdCrvStruct *SymbComposeSrfCrv(const CagdSrfStruct *Srf,
                                 const CagdCrvStruct *Crv,
                                 int EvalLinCrvsDirectly,
                                 CagdRType FilterTinysegs)
```

Srf, Crv: The curve and surface to compose.

EvalLinCrvsDirectly: If positive, evaluate linear curves (that are not isoparametric curves) directly with no precise composition, with more control points than EvalLinCrvsDirectly

FilterTinysegs: Filters out composed curves smaller than this arc length. Zero to disable.

Returns: The resulting composition.

Description: Given a curve Crv and surface Srf, computes the composition Srf(Crv(t)). Crv must be a two dimensional curve completely contained in the parametric domain of Srf. If Crv is detected as an isoparametric curve in Srf, it is returned as such, which is far more efficient in size, typically.
See also: SymbDecomposeCrvCrv, SymbComposeSrfSetCache, SymbComposePeriodicSrfCrv, , MvarComposeTVSrf, SymbMapUVCrv2E3, SymbComposeSrfCrv2,

11.2.63 SymbComposeSrfCrv2 (composit.c:867)

composition

```
CagdCrvStruct *SymbComposeSrfCrv2(const CagdSrfStruct *Srf,
                                  const CagdCrvStruct *Crv,
                                  CagdRType FilterTinysegs)
```

Srf, Crv: The curve and surface to compose.

FilterTinysegs: Filters out composed curves smaller than this arc length. Zero to disable.

Returns: The resulting composition.

Description: Given a curve Crv and surface Srf, computes the composition Srf(Crv(t)). Crv must be a two dimensional curve completely contained in the parametric domain of Srf.
See also: SymbDecomposeCrvCrv, SymbComposeSrfSetCache, SymbComposePeriodicSrfCrv, , MvarComposeTVSrf, SymbMapUVCrv2E3, SymbComposeSrfCrv,

11.2.64 SymbComposeSrfPatch (compost2.c:601)

```
CagdSrfStruct *SymbComposeSrfPatch(const CagdSrfStruct *Srf,
                                   const CagdUVType UV00,
                                   const CagdUVType UV01,
                                   const CagdUVType UV10,
                                   const CagdUVType UV11)
```

Srf: In which the four UVij corners resides.

UV00, UV01, UV10, UV11: The four corners of the patch to extract.

Returns: A patch inside Srf that is a rectangle through UVij in the domain of Srf. Only the four boundaries will be precisely reconstructed and the resulting surface will be contained in Srf only if Srf is planar.

Description: Computes a surface patch that is a general rectangle in the domain of given surface.
See also: SymbComposeSrfCrv, SymbComposeSrfSrf,

11.2.65 SymbComposeSrfSetCache (composit.c:632)

composition

```
int SymbComposeSrfSetCache(int Cache)
```

Cache: TRUE to activate the cache, FALSE to disable it.

Returns: Old state of caching.

Description: Activate caching during surface-curve composition operations.
See also: SymbComposeSrfCrv, SymbComposeSrfClrCache,

11.2.66 SymbComposeSrfSrf (compost2.c:46)

composition

```
CagdSrfStruct *SymbComposeSrfSrf(const CagdSrfStruct *Srf1,
                                 const CagdSrfStruct *Srf2)
```

Srf1, Srf2: The surfaces to compose. Srf1 must be a Bezier.

Returns: The resulting composition.

Description: Given surfaces Srf2 and Srf1, computes the composition Srf1(Srf2(u, v)). Srf2 must be a two dimensional surface completely contained in the parametric domain of Srf1.
See also: SymbDecomposeCrvCrv, SymbComposeSrfCrv, MvarComposeTVSrf,

11.2.67 SymbComposeTileObjectInSrf (compost2.c:666)

```
IPObjectStruct *SymbComposeTileObjectInSrf(const IPObjectStruct *PTile,
                                           const CagdSrfStruct *DeformSrf,
                                           IrtRType UTimes,
                                           IrtRType VTimes,
                                           int FitObj)
```

PTile: The object to map through the bivariate. Can be a (list of) curve(s) or surface(s) only. If PTile is formed out of surface(s), DeformSrf must be a Bezier surface.

DeformSrf: The mapping/deformation function from R2 to R2.

UTimes, VTimes: Number of times to tile the object in each axis.

FitObj: 2 to rescale PTile tile to precisely fit the domain (UTimes x VTimes), 1 to assume PTile is in [0,1]^2 when fitting domain, 0 to apply no mapping at all.

Returns: (UTimes x VTimes) mapped and deformed objects.

Description: Tile an input object, in place, (UTimes x VTimes) in the given bivariate surface. Computation is made precise, using composition operations.
See also: SymbComposeSrfCrv, SymbComposeSrfSrf, TrivComposeTileObjectInTVBzr,

11.2.68 SymbConeConeBisect (smp_skel.c:1457)

bisectors

skeleton

```
CagdSrfStruct *SymbConeConeBisect(const CagdVType Cone1Dir,
                                  CagdRType Cone1Angle,
                                  const CagdVType Cone2Dir,
                                  CagdRType Cone2Angle,
                                  CagdRType Size)
```

Cone1Dir: Direction of first cone axes.

Cone1Angle: Spanning angle of the first cone, in degrees.

Cone2Dir: Direction of second cone axes.

Cone2Angle: Spanning angle of the second cone, in degrees.

Size: Portion of result as it is infinite.

Returns: Constructed bisector surface.

Description: Compute the bisector surface between two cones sharing an apex. The apex is assumed to be at the origin. The computation is reduced to that of a bisector between a line and a cone, that has a rational form.
See also: SymbPlanePointBisect, SymbCylinPointBisect, SymbConePointBisect, , SymbSpherePointBisect, SymbTorusPointBisect, , SymbConeLineBisect, SymbSphereLineBisect, SymbPlaneLineBisect, , SymbCylinPlaneBisect, SymbConePlaneBisect, SymbSpherePlaneBisect, , SymbCylinSphereBisect, SymbSphereSphereBisect, SymbTorusSphereBisect, , SymbTorusTorusBisect, SymbConeSphereBisect, SymbConeConeBisect2, , SymbConeCylinBisect, SymbCylinCylinBisect,

11.2.69 SymbConeConeBisect2 (smp_skel.c:1984)

bisectors

skeleton

```
CagdSrfStruct *SymbConeConeBisect2(const CagdVType Cone1Pos,
                                   const CagdVType Cone1Dir,
                                   CagdRType Cone1Angle,
                                   const CagdVType Cone2Pos,
                                   const CagdVType Cone2Dir,
                                   CagdRType Cone2Angle)
```

Cone1Pos: The apex point of the first cone.

Cone1Dir: The direction of the first cone.

Cone1Angle: Spanning angle of the first cone, in degrees.

Cone2Pos: The apex point of the the second cone.

Cone2Dir: The direction of the second cone.

Cone2Angle: Spanning angle of the second cone, in degrees.

Returns: Constructed bisector surface.

Description: Compute the bisector surface between two cones in general position. Let C1(u), T1(u), and N1(u) and C2(v), T2(v), and N2(v) be cones' cross sections, cross sections' unit tangent field and cross sections' unit normal field. Ci(u) can be derived as a transformed circle. Ti(u) are unit circles rotated to the proper orientation ConeiDir and Ni(u) is a circle on the unit sphere with the proper orientation. Finally, note that Ci(u), Ti(u), and Ni(u) are all rational. Then, the bisector is computed as the solution of the following three linear equations:

```
< B - C1(u), T1(u) > = 0
```

```
< B - C2(v), T2(v) > = 0
```

```
< B, N1(u) - N2(v) > = < C1(u), N1(u) > - < C2(v), N2(v) >
```

The first two constraints the bisector to be on the normal plane of the generators of the two cones that are fixed along the generator (the straight lines of the cone). The last constraint make sure the bisector is on the plane that bisects the two tangent planes of the two cones. This computation is following the bisectors of two developables, presented in, "Geometric Properties of Bisector Surfaces", by Martin Peternell, Graphical Models, Volume 62, No. 3, May 2000.

See also: SymbPlanePointBisect, SymbCylinPointBisect, SymbConePointBisect, , SymbSpherePointBisect, SymbTorusPointBisect, , SymbConeLineBisect, SymbSphereLineBisect, SymbPlaneLineBisect, , SymbCylinPlaneBisect, SymbConePlaneBisect, SymbSpherePlaneBisect, , SymbCylinSphereBisect, SymbSphereSphereBisect, SymbConeSphereBisect, , SymbConeConeBisect, SymbCylinCylinBisect, SymbConeCylinBisect, ,

11.2.70 SymbConeCylinBisect (smp_skel.c:2201)

bisectors

skeleton

```
CagdSrfStruct *SymbConeCylinBisect(const CagdVType Cone1Pos,
                                   const CagdVType Cone1Dir,
                                   CagdRType Cone1Angle,
                                   const CagdVType Cyl2Pos,
                                   const CagdVType Cyl2Dir,
                                   CagdRType Cyl2Rad)
```

Cone1Pos: The apex point of the first cone.

Cone1Dir: The direction of the first cone.

Cone1Angle: Spanning angle of the first cone, in degrees.

Cyl2Pos: A point on the axis of the second cylinder.

Cyl2Dir: The direction of the second cylinder.

Cyl2Rad: Radius of second cylinder.

Returns: Constructed bisector surface.

Description: Compute the bisector surface between two cones in general position. Let C1(u), T1(u), and N1(u) and C2(v), T2(v), and N2(v) be cones' cross sections, cross sections' unit tangent field and cross sections' unit normal field. Ci(u) can be derived as a transformed circle. Ti(u) are unit circles rotated to the proper orientation ConeiDir and Ni(u) is a circle on the unit sphere with the proper orientation. Finally, note that Ci(u), Ti(u), and Ni(u) are all rational. Then, the bisector is computed as the solution of the following three linear equations:

```
< B - C1(u), T1(u) > = 0
```

```
< B - C2(v), T2(v) > = 0
```

```
< B, N1(u) - N2(v) > = < C1(u), N1(u) > - < C2(v), N2(v) >
```

The first two constraints the bisector to be on the normal plane of the generators of the two cylinders that are fixed along the generator (the straight lines of the cylinder). The last constraint make sure the bisector is on the plane that bisects the two tangent planes of the two cylinders. This computation is following the bisectors of two developables, presented in, "Geometric Properties of Bisector Surfaces", by Martin Peternell, Graphical Models, Volume 62, No. 3, May 2000.

See also: SymbPlanePointBisect, SymbCylinPointBisect, SymbConePointBisect, , SymbSpherePointBisect, SymbTorusPointBisect, , SymbConeLineBisect, SymbSphereLineBisect, SymbPlaneLineBisect, , SymbCylinPlaneBisect, SymbConePlaneBisect, SymbSpherePlaneBisect, , SymbCylinSphereBisect, SymbSphereSphereBisect, SymbConeSphereBisect, , SymbConeConeBisect, SymbCylinCylinBisect, SymbConeConeBisect2, ,

11.2.71 SymbConeLineBisect (smp_skel.c:1027)

bisectors

skeleton

```
CagdSrfStruct *SymbConeLineBisect(const CagdVType ConeDir,
                                  CagdRType ConeAngle,
                                  const CagdVType LineDir,
                                  CagdRType Size)
```

ConeDir: Direction of cone axes. Must be in the northern hemisphere.

ConeAngle: Spanning angle of the cone, in degrees.

LineDir: Direction of line from the origin. Must be in the northern hemisphere.

Size: Portion of result as it is infinite.

Returns: Constructed bisector surface.

Description: Compute the bisector surface between a cone and a line through its apex. Assumes the cone's apex is at the origin.

See also: SymbPtCrvBisectOnSphere, , SymbPlanePointBisect, SymbCylinPointBisect, SymbConePointBisect, , SymbSpherePointBisect, SymbTorusPointBisect, , SymbSphereLineBisect, SymbPlaneLineBisect, , SymbConePlaneBisect, SymbCylinPlaneBisect, SymbSpherePlaneBisect, , SymbCylinSphereBisect, SymbSphereSphereBisect, SymbConeSphereBisect, , SymbTorusSphereBisect, SymbTorusTorusBisect, SymbConeConeBisect, , SymbConeConeBisect2, SymbConeCylinBisect, SymbCylinCylinBisect,

11.2.72 SymbConePlaneBisect (smp_skel.c:1254)

bisectors

skeleton

```
CagdSrfStruct *SymbConePlaneBisect(const CagdPType ConeApex,
                                   const CagdVType ConeDir,
                                   CagdRType ConeAngle,
                                   CagdRType Size)
```

ConeApex: Apex point of cone.

ConeDir: Direction of cylinder. Must be in the northern hemisphere.

ConeAngle: Angular span of cone, in degrees.

Size: Portion of result as it is infinite.

Returns: Constructed bisector surface.

Description: Compute the bisector surface between a cone and the XY plane. The computation is reduced to that of a bisector between a line and a cone, that has a rational form.

See also: SymbPlanePointBisect, SymbCylinPointBisect, SymbConePointBisect, , SymbSpherePointBisect, SymbTorusPointBisect, , SymbConeLineBisect, SymbSphereLineBisect, SymbPlaneLineBisect, , SymbCylinPlaneBisect, SymbSpherePlaneBisect, , SymbCylinSphereBisect, SymbSphereSphereBisect, SymbConeSphereBisect, , SymbTorusSphereBisect, SymbTorusTorusBisect, SymbConeConeBisect, , SymbConeConeBisect2, SymbConeCylinBisect, SymbCylinCylinBisect,

11.2.73 SymbConePointBisect (smp_skel.c:800)

bisectors

skeleton

```
CagdSrfStruct *SymbConePointBisect(const CagdPType ConeApex,
                                   const CagdVType ConeDir,
                                   CagdRType ConeAngle,
                                   const CagdPType Pt,
                                   CagdRType Size)
```

ConeApex: Apex point of cone.

ConeDir: Direction of cone axes.

ConeAngle: Spanning angle of the cone, in degrees.

Pt: Direction of line from origin.

Size: Portion of result as it is infinite.

Returns: Constructed bisector surface.

Description: Compute the bisector surface between a cone and a point.

See also: SymbPtCrvBisectOnSphere, , SymbPlanePointBisect, SymbCylinPointBisect, SymbSpherePointBisect, , SymbTorusPointBisect, , SymbConePlaneBisect, SymbCylinPlaneBisect, SymbSpherePlaneBisect, , SymbConeLineBisect, SymbSphereLineBisect, SymbCylinSphereBisect, SymbSphereSphereBisect, SymbConeSphereBisect, , SymbTorusSphereBisect, SymbTorusTorusBisect, SymbConeConeBisect, , SymbConeConeBisect2, SymbConeCylinBisect, SymbCylinCylinBisect,

11.2.74 SymbConeSphereBisect (smp_skel.c:1406)

bisectors

skeleton

```
CagdSrfStruct *SymbConeSphereBisect(const CagdPType ConeApex,
                                    const CagdVType ConeDir,
                                    CagdRType ConeAngle,
                                    const CagdPType SprCntr,
                                    CagdRType SprRad,
                                    CagdRType Size)
```

ConeApex: Apex point of cone.

ConeDir: Direction of cone axes.

ConeAngle: Spanning angle of the cone, in degrees.

SprCntr: Center location of the sphere.

SprRad: Radius of sphere.

Size: Portion of result as it is infinite.

Returns: Constructed bisector surface.

Description: Compute the bisector surface between a sphere and a cone. The computation is reduced to that of a bisector between a point and a cone, that has a rational form.

See also: SymbPlanePointBisect, SymbCylinPointBisect, SymbConePointBisect, , SymbSpherePointBisect, SymbTorusPointBisect, , SymbConeLineBisect, SymbSphereLineBisect, SymbPlaneLineBisect, , SymbCylinPlaneBisect, SymbConePlaneBisect, SymbSpherePlaneBisect, , SymbCylinSphereBisect, SymbSphereSphereBisect, SymbTorusSphereBisect, , SymbTorusTorusBisect, SymbConeConeBisect, SymbConeConeBisect2, , SymbConeCylinBisect, SymbCylinCylinBisect,

11.2.75 SymbConicDistCrvCrv (distance.c:978)

```
CagdSrfStruct *SymbConicDistCrvCrv(const CagdCrvStruct *CCrv1,
                                   const CagdCrvStruct *CCrv2,
                                   CagdRType Dist)
```

CCrv1, CCrv2: Two curves to compute elliptic/hyperbolic distance function to. Assumes E2 curves.

Dist: Distance to the two entities.

Returns: A scalar surface whose zero set provides the conic distance (both elliptic sum and hyperbolc difference).

Description: Computes the surface of equadistance to two univariate (curve) entities. The zero set of the computed surface equal to the locus of points whose sum or different of distances to Crv1 and Crv2 equal Dist. Let d1 and d2 be the two distances of a point on the conic to Crv1 and Crv2, respectively:

```
d1 +/- d2 = Dist
```

The distance to a curve is a local minimal distance to it occurs along the normal of the curves. Hence we first seek the solution to the following two equations in X and Y and two unknowns, A and B:

```
C1(t) + a N1(t) = C2(r) + b N2(r),      a, b > 0
```

where Ni is the unnormalized normal of Ci. Given a and b (rationals), d1 and d2 equals:
d1 = a || N1(t) ||) = a sqrt(<N1(t), N1(t)>), d2 = b || N2(s) ||) = b sqrt(<N2(s), N2(s)>),
and we need to solve for,

```
a sqrt( <N1(t), N1(t)> ) +/- b sqrt( <N2(s), N2(s)> ) = Dist
```

Because we are unable to represent square roots as rationals, we square:

```
a^2 <N1(t), N1(t)> + b^2 <N2(s), N2(s)> - Dist^2 =
        = -/+ 2 a b sqrt( <N1(t), N1(t)> ) sqrt( <N2(s), N2(s)> )
```

or

```
( a^2 <N1(t), N1(t)> + b^2 <N2(s), N2(s)> - Dist^2 )^2 -
                    - 4 (ab)^2 <N1(t), N1(t)> <N2(s), N2(s)> = 0
```

11.2.76 SymbCrv2DCurvatureSign (curvatur.c:452)

curvature

```
CagdCrvStruct *SymbCrv2DCurvatureSign(const CagdCrvStruct *Crv)
```

Crv: To compute the curvature sign field.

Returns: Computed curvature sign field.

Description: Computes a scalar curve representing the curvature sign of a planar curve. The given curve is assumed to be planar and only its x and y coordinates are considered. Then the curvature sign is equal to

$$s = \dot{X}\,\ddot{Y} - \dot{Y}\,\ddot{X}$$

See also: SymbCrv2DCurvatureSqr, SymbCrv3DCurvatureSqr, , SymbCrv3DRadiusNormal, SymbCrv3DCurvatureNormal, SymbCrv2DInflectionPts, , SymbCrvExtremCrvtrPts,

11.2.77 SymbCrv2DCurvatureSqr (curvatur.c:41)

curvature

```
CagdCrvStruct *SymbCrv2DCurvatureSqr(const CagdCrvStruct *Crv)
```

Crv: To compute the square of the curvature field for.

Returns: The square of the curvature field of Crv.

Description: Computes a scalar curve representing the curvature of a planar curve. The given curve is assumed to be planar and only its x and y coordinates are considered. Then the curvature k is equal to

$$k = \frac{\dot{X}\ddot{Y} - \dot{Y}\ddot{X}}{(\dot{X}^2 + \dot{Y}^2)^{3/2}}$$

Since we cannot represent k because of the square root, we compute and represent k^2.

See also: SymbCrv3DCurvatureSqr, SymbCrv3DRadiusNormal, , SymbCrv3DCurvatureNormal, SymbCrv2DCurvatureSign, , SymbCrv2DInflectionPts, SymbCrvExtremCrvtrPts,

11.2.78 SymbCrv2DInflectionPts (curvatur.c:601)

curvature

inflection points

```
CagdPtStruct *SymbCrv2DInflectionPts(const CagdCrvStruct *Crv,
                                     CagdRType Epsilon)
```

Crv: To find all its inflection points.

Epsilon: Accuracy control.

Returns: A list of parameter values on Crv that are inflection points.

Description: Given a planar curve, finds all its inflection points by finding the zero set of the sign of the curvature function of the curve.

See also: SymbCrv2DCurvatureSqr, SymbCrv3DCurvatureSqr, SymbCrv3DCurvatureSqr, , SymbCrv3DRadiusNormal, SymbCrv3DCurvatureNormal, SymbCrv2DCurvatureSign, , SymbCrvExtremCrvtrPts,

11.2.79 SymbCrv2DUnnormNormal (curvatur.c:303)

```
CagdCrvStruct *SymbCrv2DUnnormNormal(const CagdCrvStruct *Crv)
```

Crv: Planar curve to compute unnormalized normal field for.

Returns: The normal field.

Description: Computes the unnormalized normal of a planar 2D curve as a 90 rotation in the plane of the tangent field.

See also: SymbCrv3DRadiusNormal,

11.2.80 SymbCrv2Polyline (symbpoly.c:253)

piecewise linear approximation

polyline

```
CagdPolylineStruct *SymbCrv2Polyline(const CagdCrvStruct *Crv,
                                     CagdRType TolSamples,
                                     SymbCrvApproxMethodType Method,
                                     CagdBType OptiLin)
```

Crv: To approximate as a polyline.

TolSamples: Tolerance of approximation error (Method = 2) or Number of samples to compute on polyline (Method = 0, 1).

Method: 0 - TolSamples are set uniformly in parametric space, 1 - TolSamples sets the maximum error allowed between the piecewise linear approximation and original curve. 2 - TolSamples are set optimally, considering the isocurve's curvature.

OptiLin: If TRUE, optimize linear curves.

Returns: A polyline representing the piecewise linear approximation from, or NULL in case of an error.

Description: Routine to approx. a single curve as a polyline with TolSamples samples/tolerance. Polyline is always E3 CagdPolylineStruct type. NULL is returned in case of an error, otherwise CagdPolylineStruct.

See also: BspCrv2Polyline, BzrCrv2Polyline, IritCurve2Polylines, ,

11.2.81 SymbCrv2PolylineSetTlrncErrorFunc (symbpoly.c:401)

error handling

```
SymbCrv2PolylineTlrncErrorFuncType SymbCrv2PolylineSetTlrncErrorFunc(
                                   SymbCrv2PolylineTlrncErrorFuncType ErrorFunc)
```

ErrorFunc: New error function to use.

Returns: Old error function reference.

Description: Sets the error function to be used by optimal tolerance based approx. of curves into polylines. This function should return TRUE if tolerance is met, FALSE if the curve is to be further divided. This function, if defined, is invoked in addition to the tolerance testing.
See also: SymbCrv2Polyline,

11.2.82 SymbCrv3DCurvatureNormal (curvatur.c:354)

curvature

```
CagdCrvStruct *SymbCrv3DCurvatureNormal(const CagdCrvStruct *Crv)
```

Crv: To compute the normal curvature field.

Returns: Computed normal curvature field.

Description: Computes a vector field curve representing the curvature of a curve, in the normal direction, that is kN.

$$kN = kB \times T = \frac{\dot{C} \times \ddot{C}}{\dot{C}^3} \times \frac{\dot{C}}{\dot{C}^3} = \frac{(\dot{C} \times \ddot{C}) \times \dot{C}}{\dot{C}^4}$$

See also: SymbCrv2DCurvatureSqr, SymbCrv3DCurvatureSqr, SymbCrv3DCurvatureSqr, , SymbCrv3DRadiusNormal, SymbCrv2DCurvatureSign, SymbCrv2DInflectionPts, , SymbCrvExtremCrvtrPts,

11.2.83 SymbCrv3DCurvatureSqr (curvatur.c:150)

curvature

```
CagdCrvStruct *SymbCrv3DCurvatureSqr(const CagdCrvStruct *Crv)
```

Crv: To compute scalar field of curvatrue square for.

Returns: Computed scalar field of curvature square of Crv.

Description: Computes a scalar field curve representing the square of the curvature of a given 3D curve.
See also: SymbCrv2DCurvatureSqr, SymbCrv3DRadiusNormal, , SymbCrv3DCurvatureNormal, SymbCrv2DCurvatureSign, , SymbCrv2DInflectionPts, SymbCrvExtremCrvtrPts,

11.2.84 SymbCrv3DRadiusNormal (curvatur.c:232)

curvature

```
CagdCrvStruct *SymbCrv3DRadiusNormal(const CagdCrvStruct *Crv)
```

Crv: To compute the normal field with radius as magnitude.

Returns: Computed normal field with 1 / k as magnitude.

Description: Computes a vector field curve representing the radius (1/curvature) of a curve, in the normal direction, that is N / k:

$$N / k = \frac{k\,N}{k^2} = \frac{(\dot{C} \times \ddot{C}) \times \dot{C}}{|\dot{C}|^4} \cdot \frac{|\dot{C}|^6}{(\dot{C} \times \ddot{C})^2} = \frac{((\dot{C} \times \ddot{C}) \times \dot{C}) \, |\dot{C}|^2}{(\dot{C} \times \ddot{C})^2}$$

See also: SymbCrv2DCurvatureSqr, SymbCrv3DCurvatureSqr, SymbCrv3DCurvatureSqr, , SymbCrv3DCurvatureNormal, SymbCrv2DCurvatureSign, , SymbCrv2DInflectionPts, SymbCrvExtremCrvtrPts, SymbCrv2DUnnormNormal,

11.2.85 SymbCrvAdapOffset (offset.c:1008)

offset

```
CagdCrvStruct *SymbCrvAdapOffset(const CagdCrvStruct *OrigCrv,
                                 CagdRType OffsetDist,
                                 CagdRType OffsetError,
                                 SymbOffCrvFuncType OffsetAprxFunc,
                                 CagdBType BezInterp)
```

OrigCrv: To approximate its offset curve with distance OffsetDist.

OffsetDist: Amount of offset. Negative denotes other offset direction.

OffsetError: Tolerance control.

OffsetAprxFunc: A function that can be used to approximate an offset of a curve. If NULL SymbCrvOffset function is selected.

BezInterp: If TRUE, control points are interpolated when the curve is reduced to a Bezier form. Otherwise, control points are translated OffsetDist amount only, under estimating the Offset.

Returns: An approximation to the offset curve, to within OffsetError.

Description: Given a curve and an offset amount OffsetDist, returns an approximation to the offset curve by offseting the control polygon in the normal direction. This function computes an approximation to the offset using OffsetAprxFunc, measure the error and use it to refine and decrease the error adaptively. Bezier curves are promoted to Bsplines curves. See also: Gershon Elber and Elaine Cohen, "Error Bounded Variable Distance Offset Operator for Free Form Curves and Surfaces". International Journal of Computational Geometry & Applications, Vol. 1, Num. 1, March 1991, pp 67-78.

See also: SymbCrvOffset, SymbCrvSubdivOffset, SymbSrfOffset, SymbSrfSubdivOffset, , SymbCrvAdapOffset-Trim, SymbCrvLeastSquarOffset, SymbCrvMatchingOffset, , SymbCrvVarOffset, SymbCrvAdapVarOffset,

11.2.86 SymbCrvAdapOffsetTrim (offset.c:1297)

offset

```
CagdCrvStruct *SymbCrvAdapOffsetTrim(const CagdCrvStruct *OrigCrv,
                                     CagdRType OffsetDist,
                                     CagdRType OffsetError,
                                     SymbOffCrvFuncType OffsetAprxFunc,
                                     CagdBType BezInterp)
```

OrigCrv: To approximate its offset curve with distance OffsetDist.

OffsetDist: Amount of offset. Negative denotes other offset direction.

OffsetError: Tolerance control.

OffsetAprxFunc: A function that can be used to approximate an offset of a curve. If NULL SymbCrvOffset function is selected. Third parameter of SymbOffCrvFuncType is optional.

BezInterp: If TRUE, control points are interpolated when the curve is reduced to a Bezier form. Otherwise, control points are translated OffsetDist amount only, under estimating the Offset.

Returns: An approximation to the offset curve, to within OffsetError.

Description: Same function as CagdCrvAdapOffset, but trims the self intersection loops. See also: Gershon Elber and Elaine Cohen, "Error Bounded Variable Distance Offset Operator for Free Form Curves and Surfaces". International Journal of Computational Geometry & Applications, Vol. 1, Num. 1, March 1991, pp 67-78.

See also: SymbCrvOffset, SymbCrvSubdivOffset, SymbSrfOffset, SymbSrfSubdivOffset, , SymbCrvAdapOffset, SymbCrvLeastSquarOffset, SymbCrvMatchingOffset,

11.2.87 SymbCrvAdapVarOffset (offset.c:1156)

offset

```
CagdCrvStruct *SymbCrvAdapVarOffset(const CagdCrvStruct *OrigCrv,
                                    const CagdCrvStruct *VarOffsetDist,
                                    CagdRType OffsetError,
                                    SymbVarOffCrvFuncType VarOffsetAprxFunc,
                                    CagdBType BezInterp)
```

OrigCrv: To approximate its offset curve with distance OffsetDist.

VarOffsetDist: A scalar distance function of the variable offset. Must posses a parametric domain similar to OrigCrv.

OffsetError: Tolerance control.

VarOffsetAprxFunc: A function that can be used to approximate variable offset of a curve. If NULL SymbCrvVarOffset function is selected.

BezInterp: If TRUE, control points are interpolated when the curve is reduced to a Bezier form. Otherwise, control points are translated OffsetDist amount only, under estimating the Offset.

Returns: An approximation to the offset curve, to within OffsetError.

Description: Given a curve and a scalar offset function VarOffsetDist, returns an approximation to the variable offset curve by offseting the control polygon in the normal direction. This function computes an approximation to the offset using VarOffsetAprxFunc, measure the error and use it to refine and decrease the error adaptively. Bezier curves are promoted to Bsplines curves. See also: Gershon Elber and Elaine Cohen, "Error Bounded Variable Distance Offset Operator for Free Form Curves and Surfaces". International Journal of Computational Geometry & Applications, Vol. 1, Num. 1, March 1991, pp 67-78.
See also: SymbCrvOffset, SymbCrvVarOffset, SymbCrvSubdivOffset, SymbCrvAdapOffset, , SymbSrfOffset, SymbSrfSubdivOffset, SymbCrvAdapOffsetTrim, , SymbCrvLeastSquarOffset, SymbCrvMatchingOffset,

11.2.88 SymbCrvAdd (symb_crv.c:45)

addition

symbolic computation

```
CagdCrvStruct *SymbCrvAdd(const CagdCrvStruct *Crv1,
                          const CagdCrvStruct *Crv2)
```

Crv1, Crv2: Two curve to add up coordinate-wise.

Returns: The summation of Crv1 + Crv2 coordinate-wise.

Description: Given two curves - add them coordinate-wise. The two curves are promoted to same point type before the multiplication can take place. Furthermore, order and continuity are matched as well.
See also: SymbCrvSub, SymbCrvMult,

11.2.89 SymbCrvArcLen (arc_len.c:385)

arc length

```
CagdRType SymbCrvArcLen(const CagdCrvStruct *Crv, CagdRType Epsilon)
```

Crv: Curve to compute a tight approximation on arc length.

Epsilon: Accuracy control.

Returns: The approximated arc length of the given curve Crv.

Description: Computes a tight approximation to the arc length of a curve. Estimates the arc length scalar field of Crv using SymbCrvArcLenSclrCrv and evaluate the estimate on the curve's domain boundary.
See also: SymbCrvArcLen2,

11.2.90 SymbCrvArcLen2 (arc_len.c:730)

arc length

```
CagdRType SymbCrvArcLen2(const CagdCrvStruct *Crv, CagdRType Epsilon)
```

Crv: Curve to compute a tight approximation on arc length.

Epsilon: Accuracy control.

Returns: The approximated arc length of the given curve Crv.

Description: Computes a tight approximation to the arc length of a curve. Estimates arc length by bounding from above and from below the arc len: + From above, using the arc length control polygon of the curve. + From below, using the chord P0Pn, the line from first to last control point. If difference is greater than Epsilon, divide and conquer. If, however, the given curve is linear, its control polygon (also arc len) length is returned immediately.
See also: SymbCrvArcLen,

11.2.91 SymbCrvArcLenCrv (arc_len.c:538)

arc length

```
CagdCrvStruct *SymbCrvArcLenCrv(const CagdCrvStruct *Crv,
                                CagdRType Fineness,
                                int Order)
```

Crv: To approximate as an arc length curve.

Fineness: Tolerance to use is sampling the original curve.

Order: Order of least square fit curve.

Returns: A polynomial curve similar to input curve but with almost arc length parametrization.

Description: Computes an approximated arc length curve from a given curve. Approximation is achieved by least square fitting of points sampled along the curve that are arc length parameterized.

11.2.92 SymbCrvArcLenSclrCrv (arc_len.c:342)

arc length

```
CagdCrvStruct *SymbCrvArcLenSclrCrv(const CagdCrvStruct *Crv,
                                    CagdRType Epsilon)
```

Crv: To approximate its arc length scalar field.

Epsilon: Accuracy of approximating.

Returns: A scalar field approximating Crv arc length.

Description: Computes a scalar curve approximating the arc length of given curve Crv. Arc length is estimated by computing the square of Crv's first derivative approximating its square root and integrating symbolically.

11.2.93 SymbCrvArcLenSteps (arc_len.c:421)

arc length

```
CagdPtStruct *SymbCrvArcLenSteps(const CagdCrvStruct *Crv,
                                 CagdRType Length,
                                 CagdRType Epsilon)
```

Crv: Curve to compute constant arc Length steps.

Length: The step size.

Epsilon: Accuracy control.

Returns: List of parameter values to march along Crv with arc Length between them.

Description: Computes parameter values to move steps of Length at a time on curve Crv. Returned is a list of parameter values to move along.
See also: SymbCrvArcLenSteps2,

11.2.94 SymbCrvArcLenSteps2 (arc_len.c:471)

arc length

```
CagdRType *SymbCrvArcLenSteps2(const CagdCrvStruct *Crv,
                               int NumSteps,
                               CagdRType Epsilon)
```

Crv: Curve to compute NumSteps of equal arc-length.

NumSteps: Number of steps.

Epsilon: Accuracy control.

Returns: A vector of NumSteps+1 parameter values (including TMin and TMax) to step through along Crv with equal arc-length between them.

Description: Computes parameter values that divides the arc length of input Crv to NumSteps steps. Returned is a list of parameter values for the steps.
See also: SymbCrvArcLenSteps,

11.2.95 SymbCrvBiArcApprox (biarc.c:61)

```
SymbArcStruct *SymbCrvBiArcApprox(const CagdCrvStruct *Crv,
                                  CagdRType Tolerance,
                                  CagdRType MaxAngle)
```

Crv: 2D Curve to approximate using piecewise biarcs.

Tolerance: Of approximation.

MaxAngle: Of an arc in the output set. In no way it will be more than or equal to 180 degrees. In Degrees.

Returns: List of arcs approximating Crv to within Tolerance.

Description: Computes a piecewise biarc approximation to given freeform planar curve. The following steps are performed during this approximation process:

1. The curve is split at all C^1 discontinuities.
2. The curve is split at inflection points, if any.
3. Each convex/concave curve region: a. Fit the curve region with a G^1 continuous biarc that is tangent to the curve's region at the region's end points. b. If fit is good enough, we stop. Otherwise subdivide region into two and recursively invoke step 2 on both halves.

See also: SymbCrv2DInflectionPts, SymbCrvCubicApprox, SymbCrvBiArcApproxC1,

11.2.96 SymbCrvBiArcApproxC1 (biarc.c:188)

```
SymbArcStruct *SymbCrvBiArcApproxC1(const CagdCrvStruct *CCrv,
                                    CagdRType Tolerance,
                                    CagdRType MaxAngle)
```

CCrv: 2D Curve to approximate using piecewise biarcs.

Tolerance: Of approximation.

MaxAngle: Of an arc in the output set. In no way it will be more than or equal to 180 degrees. In Degrees.

Returns: List of arcs approximating Crv to within Tolerance.

Description: Computes a piecewise biarc approximation to given C^1 planar curve. The following steps are performed during this approximation process:

1. The curve is split at inflection points, if any.
2. Each convex/concave curve region: a. Fit the curve region with a G^1 continuous biarc that is tangent to the curve's region at the region's end points. b. If fit is good enough, we stop. Otherwise subdivide region into two and recursively invoke step 2 on both halves.

See also: SymbCrvBiArcApprox,

11.2.97 SymbCrvBisectors (crv_skel.c:96)

bisectors

skeleton

```
CagdCrvStruct *SymbCrvBisectors(const CagdCrvStruct *Crv,
                                int BisectFunc,
                                CagdRType SubdivTol,
                                CagdBType NumerImprove,
                                CagdBType SameNormal,
                                CagdBType SupportPrms)
```

Crv: Either one or two curves to compute bisectors for. Assumes E2 curves.

BisectFunc: If 1, normal fields are used to compute bisector surface. If 2, tangent fields instead of normals are used to compute the bisector surface function. If 3, using solution of normal intersection pt.

SubdivTol: Accuracy of computation. 0.001 is a good start.

NumerImprove: If TRUE, a numerical improvment stage is applied.

SameNormal: If TRUE, the bisector should be oriented for inner or outer side of the curves, with respect to their normals.

SupportPrms: If TRUE, return curve is of type E4 instead of E2 and the third and fourth coefficients holds the support parameters of the first and second curves, respectively.

Returns: A list of piecewise linear curves approximating the bisectors of Crv.

Description: Computes the skeleton curves (bisectors) of a given curve or two. If Crv contains a list of two curves the bisector between the two curves is computed. Otherwise, Crv self bisectors are computed. Employs the F1/F2/F34 functions from the paper: Gershon Elber and Myung Soo Kim. "Bisector Curves of Planar Rational Curves." CAD, Vol 30, No 14, pp 1089-1096, December 1998.
See also: SymbCrvCnvxHull, SymbCrvDiameter, SymbCrvBisectorsSrf,

11.2.98 SymbCrvBisectorsSrf (crv_skel.c:403)

bisectors

skeleton

```
CagdSrfStruct *SymbCrvBisectorsSrf(const CagdCrvStruct *Crv, int BisectFunc)
```

Crv: Either one or two curves to compute bisectors for. Assumes E2 curves.

BisectFunc: If 1, normal fields are used to compute bisector surface. If 2, tangent fields instead of tangents are used to compute the bisector surface function. If 3, using solution of normal intersection pt.

Returns: A scalar surface whose zero set provides matching bisecting points on Crv, if BisectFunc > 0.

Description: Computes the bisector surface definition of a given curve or two. If Crv contains a list of two curves the bisector between the two curves is computed. Otherwise, Crv self–bisectors are sought. The result is a scalar surface whose zero set is the set of bisector(s) of the curves.
See also: SymbCrvCnvxHull, SymbCrvDiameter, SymbCrvBisectors, , SymbCrvBisectorsSrf2, SymbCrvPtBisectorsSrf3D, SymbCrvCrvBisectorSrf3D,

11.2.99 SymbCrvBisectorsSrf2 (crv_skel.c:582)

bisectors

skeleton

```
CagdSrfStruct *SymbCrvBisectorsSrf2(const CagdCrvStruct *Crv)
```

Crv: Either one or two curves to compute bisectors for. Assumes E3 curves.

Returns: The real bisector surface for three space curves, if BisectFunc = 4.

Description: Computes the bisector surface definition of a given curve or two. If Crv contains a list of two curves the bisector between the two curves is computed. Otherwise, Crv self–bisectors are sought. The result is a scalar surface whose zero set is the set of bisector(s) of the curves. Solve for the normal intersection surface in the plane and then elevate in Z using the rational function of

```
||P - C1(s)||^2 - ||P - C2(t)||^2.
```

See also: SymbCrvCnvxHull, SymbCrvDiameter, SymbCrvBisectors, , SymbCrvBisectorsSrf, SymbCrvPtBisectorsSrf3D, SymbCrvCrvBisectorSrf3D, , SymbCrvBisectorsSrf3,

11.2.100 SymbCrvBisectorsSrf3 (crv_skel.c:747)

bisectors

skeleton

```
CagdSrfStruct *SymbCrvBisectorsSrf3(const CagdCrvStruct *Crv)
```

Crv: Either one or two curves to compute bisectors for. Assumes E2 curves.

Returns: A scalar surface whose zero set provides matching bisecting points on Crv, if BisectFunc = 3.

Description: Computes the bisector surface definition of a given curve or two. If Crv contains a list of two curves the bisector between the two curves is computed. Otherwise, Crv self–bisectors are sought. The result is a scalar surface whose zero set is the set of bisector(s) of the curves. Solve for the normal intersection surface in the plane and then substitute into (the bisector's correspondance is the zero set then).

```
                C1(s) + C2(t)
< P - ------------------------, C1(t) - C2(s) > = 0.
                      2
```

See also: SymbCrvCnvxHull, SymbCrvDiameter, SymbCrvBisectors, SymbCrvBisectorsSrf2, SymbCrvBisectorsSrf, SymbCrvPtBisectorsSrf3D, SymbCrvCrvBisectorSrf3D,

11.2.101 SymbCrvCnvxHull (ccnvhul.c:2518)

```
CagdCrvStruct *SymbCrvCnvxHull(const CagdCrvStruct *Crv, CagdRType SubdivTol)
```

Crv: To compute its convex hull.

SubdivTol: Of numeric search for the zero set (for surface subdivision). A positive value (10 is a good start).

Returns: A curve representing the convex hull of Crv.

Description: Computes the convex hull of a C1 freeform planar curve, in the XY plane. The convex hull is computed by symbolically isolating the non negative set (in t) of:

```
C'(t) x (C(r) - C(t)) >= 0,   for all r.
```

Note the above equation yields a scalar value since C(t) is planar. The resulting set in t contains all the sub-domain in C(t) that is on the convex hull of C(t). Connecting these pieces with straight lines yields the final convex hull curve.

See also: SymbCrvPtTangents, SymbCrvDiameter,

11.2.102 SymbCrvConstSet (symbzero.c:269)

constant set

zero set

symbolic computation

```
CagdPtStruct *SymbCrvConstSet(const CagdCrvStruct *Crv,
                              int Axis,
                              CagdRType Epsilon,
                              CagdRType ConstVal,
                              CagdBType NoSolsOnEndPts)
```

Crv: To compute its constant set.

Axis: The axis of Crv to compute constant set for, X = 1, Y = 2, etc.

Epsilon: Tolerance control.

ConstVal: The value at which to compute the constant set.

NoSolsOnEndPts: If TRUE, solutions at the end of the domain are purged.

Returns: List of parameter values form which Crv has an value of ConstVal in axis Axis.

Description: Computes the constant set of a given curve, in the given axis (1-3 for X-Z). Returned is a list of the constant set points holding the parameter values at Pt[0] of each point.

11.2.103 SymbCrvCrossProd (symb_crv.c:556)

product

cross product

symbolic computation

```
CagdCrvStruct *SymbCrvCrossProd(const CagdCrvStruct *CCrv1,
                                const CagdCrvStruct *CCrv2)
```

CCrv1, CCrv2: Two curve to multiply in R3 and compute cross product for.

Returns: A vector in R3 curve representing the cross product of Crv1 x Crv2.

Description: Given two curves - computes their cross product in R3. Returned curve is a curve representing the cross product of the two given curves.

See also: SymbCrvDotProd, SymbCrvVecDotProd, SymbCrvMult, SymbCrvMultScalar,

11.2.104 SymbCrvCrvBisectOnSphere (smp_skel.c:291)

bisectors

skeleton

```
CagdSrfStruct *SymbCrvCrvBisectOnSphere(const CagdCrvStruct *CCrv1,
                                        const CagdCrvStruct *CCrv2)
```

CCrv1, CCrv2: Two curves on the unit sphere.

Returns: The bisector surface bivariate function whose zero set provides the bisector correspondance in the parameteric space.

Description: Computes the bisector on a sphere between two curves, both are assumed to be on the unit sphere. The end result is a bivariate function NOT a curve on the sphere but rather a rational surface in the (s, t) parameter space of Crv1(s) and Crv2(t) whose zero set provides the correspondancing bisector in the parametric space. Let P be the bisector point. Then, the following must vanish:

```
< P, C1(t) > = < P, C2(t) > (Equality of angular distance)
< P - C1(t), C1'(t) > = 0   (orthogonality of distance measure)
< P - C2(t), C2'(t) > = 0   (orthogonality of distance measure)
```

Then the returned result is the determinant of these three equations that must vanish.
See also: SymbPtCrvBisectOnSphere2, SymbPtCrvBisectOnSphere3,

11.2.105 SymbCrvCrvBisectOnSphere2 (smp_skel.c:382)

bisectors

skeleton

```
CagdCrvStruct *SymbCrvCrvBisectOnSphere2(const CagdCrvStruct *Crv1,
                                         const CagdCrvStruct *Crv2,
                                         CagdRType SubdivTol)
```

Crv1, Crv2: Two curves on the unit sphere.

SubdivTol: Accuracy of computation.

Returns: A list of piecewise linear curves approximating the bisectors of Crv1 and Crv2 on the sphere.

Description: Computes the bisector on a sphere between two curves on the sphere. The returned result is a piecewise linear curve on the sphere.
See also: SymbPtCrvBisectOnSphere, SymbCrvCrvBisectOnSphere, , SymbCrvCrvBisectOnSphere3,

11.2.106 SymbCrvCrvBisectOnSphere3 (smp_skel.c:556)

bisectors

skeleton

```
CagdSrfStruct *SymbCrvCrvBisectOnSphere3(const CagdCrvStruct *CCrv1,
                                         const CagdCrvStruct *CCrv2)
```

CCrv1, CCrv2: two curves on the unit sphere.

Returns: The bisector surface function.

Description: Computes the bisector of two cone surfaces sharing an apex at the origin represented as their generating curves on a unit sphere. Let P be the bisector point. Then, the following must vanish:

```
< P, C1(t) > = < P, C2(t) > (Equality of angular distance)
< P - C1(t), C1'(t) > = 0   (orthogonality of distance measure)
< P - C2(t), C2'(t) > = 0   (orthogonality of distance measure)
```

Then, the returned is the solution of the above equations.
See also: SymbPtCrvBisectOnSphere, SymbPtCrvBisectOnSphere, , SymbPtCrvBisectOnSphere2,

11.2.107 SymbCrvCrvBisectorSrf3D (crv_skel.c:887)

bisectors

skeleton

```
CagdSrfStruct *SymbCrvCrvBisectorSrf3D(const CagdCrvStruct *CCrv1,
                                       const CagdCrvStruct *CCrv2,
                                       CagdRType Alpha)
```

CCrv1, CCrv2: Two three space curves to compute their bisector surface.

Alpha: Alpha-sector ratio (0.5 for a bisector).

Returns: The bisector surface.

Description: Computes the bisector surface of two curve in arbitrary general three space position.
See also: SymbCrvDiameter, SymbCrvCnvxHull, SymbCrvBisectorsSrf, , SymbCrvPtBisectorSrf3D,

11.2.108 SymbCrvCrvConvolution (moffset.c:116)

```
CagdCrvStruct *SymbCrvCrvConvolution(CagdCrvStruct *Crv1,
                                     CagdCrvStruct *Crv2,
                                     CagdRType OffsetDist,
                                     CagdRType Tolerance)
```

Crv1, Crv2: The two curves to convolve.

OffsetDist: Amount of offset, if Crv2 == NULL. Negative value denotes other offset/convolution direction.

Tolerance: Of angular discrepancy that is allowed.

Returns: The offset curve approximation.

Description: Computes the convolution of the given two curves by matching their tangents and reparametrizing Crv2. If Crv2 is NULL, an Arc of radius OffsetDist is used, resulting in an offset operation of Crv1. Both Crv1 and Crv2 are assumed to have no inflection points and to span the same angular domain. That is Crv1'(0) || Crv2'(0) and similarly Crv1'(1) || Crv2'(1), where || denotes parallel.
See also: SymbCrvOffset, SymbCrvSubdivOffset, SymbSrfOffset, SymbSrfSubdivOffset, , SymbCrvAdapOffset, SymbCrvAdapOffsetTrim, SymbCrvLeastSquarOffset, , SymbCrvMatchingOffset,

11.2.109 SymbCrvCrvInter (distance.c:802)

```
CagdPtStruct *SymbCrvCrvInter(const CagdCrvStruct *Crv1,
                              const CagdCrvStruct *Crv2,
                              CagdRType CCIEpsilon,
                              CagdBType SelfInter)
```

Crv1, Crv2: The two curves to intersect.

CCIEpsilon: Tolerance of computation.

SelfInter: If TRUE, needs to handle a curve against itself detecting self intersections in Crv1 (== Crv2).

Returns: List of intersection points. Each point holds the intersection location in Crv1 as first coefficient and the intersection location in Crv2 as second coefficient.

Description: Computes the intersection points of two planar curves, in the XY plane
See also: SymbSrfDistCrvCrv, SymbSrfDistFindPoints, CagdCrvCrvInter,

11.2.110 SymbCrvCrvtrTrim (offset.c:2144)

offset

```
CagdCrvStruct *SymbCrvCrvtrTrim(const CagdCrvStruct *Crv,
                                CagdRType Dist,
                                CagdRType Eps)
```

Crv: To clip at all locations the curvature of this planar (XY) curve is too high.

Dist: Amount of offset. Negative denotes other offset direction.

Eps: Tolerance of computation.

Returns: One of more subregions of Crv that have regions with curvature no larger than 1/Dist.

Description: Given a planar curve and an offset distance, a-priori subdivide and clip the given curve at all locations its curvature radius is smaller than the offset distance.
See also: SymbCrvOffset, SymbCrvSubdivOffset, SymbSrfOffset, SymbSrfSubdivOffset, , SymbCrvAdapOffset, SymbCrvLeastSquarOffset, SymbCrvMatchingOffset,

11.2.111 SymbCrvCubicApprox (cubcaprx.c:38)

```
CagdCrvStruct *SymbCrvCubicApprox(const CagdCrvStruct *Crv,
                                  CagdRType Tolerance)
```

Crv: 2D Curve to approximate using piecewise cubics.

Tolerance: Of approximation, in Hausdorff distance sense.

Returns: List of cubics approximating Crv to within Tolerance. List will be C^1.

Description: Computes a piecewise cubic approximation to given freeform planar curve. The following steps are performed during this approximation process: a. Fit the curve with a C^1 continuous cubic that is tangent to the curves's end points. b. If fit is good enough, we stop. Otherwise subdivide region into two and recursively invoke step 2 on both halfs.
See also: SymbCrvBiArcApprox, SymbApproxCrvAsBzrCubics,

11.2.112 SymbCrvDeriveRational (symb_crv.c:820)

derivatives

```
CagdCrvStruct *SymbCrvDeriveRational(const CagdCrvStruct *Crv)
```

Crv: A curve to differentiate.

Returns: Differentiated rational curve.

Description: Given a rational curve - computes its derivative curve (Hodograph) using the quotient rule for differentiation.
See also: BzrCrvDerive, BspCrvDerive, CagdCrvDerive, SymbSrfDeriveRational,

11.2.113 SymbCrvDeterminant2 (crv_skel.c:1770)

determinant

```
CagdCrvStruct *SymbCrvDeterminant2(const CagdCrvStruct *Crv11,
                                   const CagdCrvStruct *Crv12,
                                   const CagdCrvStruct *Crv21,
                                   const CagdCrvStruct *Crv22)
```

Crv11, Crv12, Crv21, Crv22: The four factors of the determinant.

Returns: A scalar field representing the determinant computation.

Description: Computes the expression of Crv11 * Crv22 - Crv12 * Crv21, which is a determinant of a 2 by 2 matrix.
See also: SymbCrvDeterminant3, SymbSrfDeterminant2,

11.2.114 SymbCrvDeterminant3 (crv_skel.c:1727)

determinant

```
CagdCrvStruct *SymbCrvDeterminant3(const CagdCrvStruct *Crv11,
                                   const CagdCrvStruct *Crv12,
                                   const CagdCrvStruct *Crv13,
                                   const CagdCrvStruct *Crv21,
                                   const CagdCrvStruct *Crv22,
                                   const CagdCrvStruct *Crv23,
                                   const CagdCrvStruct *Crv31,
                                   const CagdCrvStruct *Crv32,
                                   const CagdCrvStruct *Crv33)
```

Crv11, Crv12, Crv13: The nine factors of the determinant.

Crv21, Crv22, Crv23: ”

Crv31, Crv32, Crv33: ”

Returns: A scalar field representing the determinant computation.

Description: Computes the expression of a 3 by 3 determinants.
See also: SymbCrvDeterminant2, SymbSrfDeterminant3,

11.2.115 SymbCrvDiameter (ccnvhul.c:165)

```
IPPolygonStruct *SymbCrvDiameter(const CagdCrvStruct *Crv,
                                 CagdRType SubdivTol)
```

Crv: A curve to process its diameter function.

SubdivTol: Of numeric search for the zero set (for surface subdivision). A positive value (0.01 is a good start).

Returns: Contours of the matched parallel lines on Crv. Each vertex will hold two parameter values on Crv.

Description: Given a freeform curve, compute its diameter as a function. If the curve is a convex, probably as a result of a convex hull computation of an original curve, the matching will be one to one.
See also: SymbCrvCnvxHull, SymbCrvPtTangents, SymbCrvDiameterMinMax, MvarCrvDiameter,

11.2.116 SymbCrvDiameterMinMaxMalloc (ccnvhul.c:362)

```
CagdRType *SymbCrvDiameterMinMaxMalloc(const CagdCrvStruct *Crv,
                                       IPPolygonStruct *Cntrs,
                                       int Min)
```

Crv: To compute its diameter.

Cntrs: Output of SymbCrvDiameter - the matched paraller tangents.

Min: TRUE of minimum diameter, FALSE for maximum diameter.

Returns: Two parameter values on Crv of tangent lines extreme value. Returns an address to point.

Description: Computes the maximum or minimum diameter out of diameter matched list
See also: SymbCrvDiameter,

11.2.117 SymbCrvDiameterMinMaxToData (ccnvhul.c:305)

```
CagdRType *SymbCrvDiameterMinMaxToData(const CagdCrvStruct *Crv,
                                       IPPolygonStruct *Cntrs,
                                       int Min,
                                       CagdRType *Params)
```

Crv: To compute its diameter.

Cntrs: Output of SymbCrvDiameter - the matched paraller tangents.

Min: TRUE of minimum diameter, FALSE for maximum diameter.

Params: Two parameter values on Crv of tangent lines extreme value. Returns an address to point.

Returns: Two parameter values on Crv of tangent lines extreme value. Returns an address to point.

Description: Computes the maximum or minimum diameter out of diameter matched list
See also: SymbCrvDiameter,

11.2.118 SymbCrvDotProd (symb_crv.c:373)

product

dot product

symbolic computation

```
CagdCrvStruct *SymbCrvDotProd(const CagdCrvStruct *Crv1,
                              const CagdCrvStruct *Crv2)
```

Crv1, Crv2: Two curve to multiply and compute a dot product for.

Returns: A scalar curve representing the dot product of Crv1 . Crv2.

Description: Given two curves - computes their dot product. Returned curve is a scalar curve representing the dot product of the two given curves.
See also: SymbCrvScalarScale, SymbCrvVecDotProd, SymbCrvMult, SymbCrvMultScalar,

11.2.119 SymbCrvDual (duality.c:31)

```
CagdCrvStruct *SymbCrvDual(const CagdCrvStruct *Crv)
```

Crv: The curve to compute its dual.

Returns: The dual curve.

Description: Computes the dual of the given curve. The dual curve is a mapping of the tangent lines of Crv (for which Crv is the envelop of) to points in the dual space. Duality is derived by computing the tangent line "Ax + By + C = 0" to curve Crv and mapping this line to homogeneous point (A/C, B/C).
See also: SymbSrfDual,

11.2.120 SymbCrvEnclosedArea (symb_crv.c:1038)

area

symbolic computation

```
CagdCrvStruct *SymbCrvEnclosedArea(const CagdCrvStruct *Crv)
```

Crv: A curve to compute area field curve for.

Returns: The area field curve.

Description: Given a planar curve, compute its enclosed area field curve. This has little meaning unless Crv is closed, in which by evaluation the resulting area field curve at the end points, the area enclosed by Crv can be computed.
See also: SymbCrvEnclosedAreaEval,

11.2.121 SymbCrvEnclosedAreaEval (symb_crv.c:1109)

area

symbolic computation

```
CagdRType SymbCrvEnclosedAreaEval(const CagdCrvStruct *Crv)
```

Crv: A curve to compute signed area for.

Returns: The signed area.

Description: Given a planar curve, compute its enclosed signed area.
See also: SymbCrvEnclosedArea,

11.2.122 SymbCrvExtremCrvtrPts (curvatur.c:720)

curvature

```
CagdPtStruct *SymbCrvExtremCrvtrPts(const CagdCrvStruct *Crv,
                                    CagdRType Epsilon,
                                    CagdBType Crv2D)
```

Crv: To find all int extrem curvature locations.

Epsilon: Accuracy control.

Crv2D: TRUE if the curve is in the XY plane and treated as 2D, FALSE, for a full 3D curve.

Returns: A list of parameter values on Crv that have extreme curvature values. An int attribute named "ExtremType" is placed on each parameter value with a value of -1, 0, or 1 for minimum curvature location, zero curvature location and maximum curvature location, respectively.

Description: Given a planar curve, finds all its extreme curvature points by finding the set of extreme locations on the curvature function of Crv. Extreme curvature is computed as the zeros of <(kN)', kN> = k'k. Assumes the input curve is C^2 (preferably C^3).
See also: SymbCrv2DCurvatureSqr, SymbCrv3DCurvatureSqr, SymbCrv3DCurvatureSqr, , SymbCrv3DRadiusNormal, SymbCrv3DCurvatureNormal, SymbCrv2DCurvatureSign, , SymbCrv2DInflectionPts, SymbCrvExtremCrvtrPts2,

11.2.123 SymbCrvExtremCrvtrPts2 (curvatur.c:673)

curvature

```
CagdPtStruct *SymbCrvExtremCrvtrPts2(const CagdCrvStruct *Crv,
                                     CagdRType Epsilon,
                                     CagdBType Crv2D)
```

Crv: To find all int extrem curvature locations.

Epsilon: Accuracy control.

Crv2D: TRUE if the curve is in the XY plane and treated as 2D, FALSE for a full 3D curve.

Returns: A list of parameter values on Crv that have extrem curvature values. An int attribute named "ExtremType" is placed on each parameter value with a value of -1, 0, or 1 for minimum curvature location, zero curvature location and maximum curvature location, respectively.

Description: Given a planar curve, finds all its extreme curvature points by finding the set of extreme locations on the curvature function of Crv. Extreme curvature is computed as the zeros of <(kN)', kN> = k'k.
See also: SymbCrv2DCurvatureSqr, SymbCrv3DCurvatureSqr, SymbCrv3DCurvatureSqr, , SymbCrv3DRadiusNormal, SymbCrv3DCurvatureNormal, SymbCrv2DCurvatureSign, , SymbCrv2DInflectionPts, SymbCrvExtremCrvtrPts,

11.2.124 SymbCrvExtremSet (symbzero.c:142)

extrema set

symbolic computation

```
CagdPtStruct *SymbCrvExtremSet(const CagdCrvStruct *Crv,
                               int Axis,
                               CagdRType Epsilon,
                               CagdBType NoSolsOnEndPts)
```

Crv: To compute its extrema set.

Axis: The axis of Crv to compute extrema set for, W = 0, X = 1, etc.

Epsilon: Tolerance control.

NoSolsOnEndPts: If TRUE, solutions at the end of the domain are purged.

Returns: List of parameter values form which Crv has an extrema value in axis Axis.

Description: Computes the extrema set of a given curve, in given axis (0/1-3 for W/X-Z). Returned is a list of the extreme set points holding the parameter values at Pt[0] of each point. One could compute the derivative of the curve and find its zero set. However, for rational curves, this will double the degree and slow down the computation considerably.

11.2.125 SymbCrvGenSignedCrvtr (crvtrrec.c:37)

```
CagdCrvStruct *SymbCrvGenSignedCrvtr(const CagdCrvStruct *CCrv,
                                     int Samples,
                                     int Order,
                                     int ArcLen)
```

CCrv: Curve to compute signed curvature signature approximation for.

Samples: Number of samples to use in the approximation.

Order: Order of signed curvature approximating curve.

ArcLen: TRUE if Crv is to be assumed arc-length, FALSE if needs to compensate for a non arc-length parametrization.

Returns: Scalar polynomial curve of length Samples and of order Order that represents the signed curvature field of Crv.

Description: Computes a signed curvature signature scalar curve to the given planar curve. The returned signature is parameterized by the curve's arc-length even if the original curve is not arc length, if ArcLen is TRUE.
See also: SymbSignedCrvtrGenCrv, SymbCrv3DCurvatureNormal, SymbCrv2DCurvatureSign,

11.2.126 SymbCrvInvert (symb_crv.c:281)

division

symbolic computation

reciprocal value

```
CagdCrvStruct *SymbCrvInvert(const CagdCrvStruct *Crv)
```

Crv: A scalar curve to compute a reciprocal value for.

Returns: A rational scalar curve that is equal to the reciprocal value of Crv.

Description: Given a scalar curve, returns a scalar curve representing the reciprocal values, by making it rational (if was not one) and flipping the numerator and the denominator.
See also: SymbCrvDotProd, SymbCrvVecDotProd, SymbCrvMult, SymbCrvMultScalar,

11.2.127 SymbCrvLeastSquarOffset (offset.c:1470)

offset

```
CagdCrvStruct *SymbCrvLeastSquarOffset(const CagdCrvStruct *Crv,
                                       CagdRType OffsetDist,
                                       int NumOfSamples,
                                       int NumOfDOF,
                                       int Order,
                                       CagdRType *Tolerance)
```

Crv: To approximate its offset curve with distance OffsetDist.

OffsetDist: Amount of offset. Negative denotes other offset direction.

NumOfSamples: Number of samples to sample the offset curve at.

NumOfDOF: Number of degrees of freedom on the newly computed offset approximation. This is thesame as the number of control points the new curve will have.

Order: Of the newly constructed offset approximation. If equal to zero, the order of Crv will be used.

Tolerance: To return an error estimate in the L-infinity norm.

Returns: An approximation to the offset curve.

Description: Given a curve and an offset amount OffsetDist, returns an approximation to the offset curve by least square fitting a curve to samples taken on the offset curve. Resulting curve of order Order (degree of Crv if Order == 0) will have NumOfDOF control points that least sqaure fit NumOfSamples samples on the offset curve. Tolerance will be updated to hold an error distance measure.
See also: SymbCrvOffset, SymbCrvSubdivOffset, SymbSrfOffset, SymbSrfSubdivOffset, , SymbCrvAdapOffset, SymbCrvAdapOffsetTrim, SymbCrvMatchingOffset,

11.2.128 SymbCrvListCnvxHull (ccnvhul.c:2486)

```
CagdCrvStruct *SymbCrvListCnvxHull(CagdCrvStruct *Crvs, CagdRType SubdivTol)
```

Crvs: To compute their convex hull.

SubdivTol: Of numeric search for the zero set (for surface subdivision). A positive value (0.01 is a good start).

Returns: A curve representing the convex hull of Crvs.

Description: Computes the convex hull of C1 freeform planar curves, in the XY plane. The convex hull is computed by symbolically isolating the non negative set (in t) of:

```
C'(t) x (C(r) - C(t)) >= 0,   for all r.
```

Note the above equation yields a scalar value since C(t) is planar. The resulting set in t contains all the subdomain in C(t) that is on the convex hull of C(t). Connecting these pieces with straight lines yeilds the final convex hull curve.
See also: SymbCrvPtTangents, SymbCrvDiameter,

11.2.129 SymbCrvLstSqrAprxPlln (crvlsapx.c:1340)

least squares

```
CagdCrvStruct *SymbCrvLstSqrAprxPlln(const CagdCtlPtStruct *Polyline,
                                     CagdRType ExpectedError,
                                     int Order,
                                     CagdRType NumericTol,
                                     CagdBType ForceC1Continuity,
                                     CagdRType C1DiscontThresh,
                                     SymbCrvLSErrorMeasureType ErrMeasure)
```

Polyline: The polyline to be approximated (represented as a list of points).

ExpectedError: The expected square approximation error of the curve.

Order: The polynomial order of the required approximating curve.

NumericTol: The numeric tolerance for the zero solver (used in the curve-point distance computation).

ForceC1Continuity: A flag indicating that the algorithm should attempt to make the curve C1-continuous (except where C1-discontinuities in the original polyline are suspected).

C1DiscontThresh: The threshold for determining C1-discontinuity in the polyline, in terms of dot-product (i.e. the inverse cosine of this value is the threshold angle).

ErrMeasure: Which method of error measurement to use.

Returns: The approximating curve. SymbCrvApproxPolylineProcessSegment, SymbCrvApproxPolylineExtractNext

Description: Approximates a polyline segment by a curve within an estimates square error. This function repeatedly requests the next C1-continuous segment of the polyline and approximates it with a B-spline curve of a required order and approximation error, until the entire polyline is processed.

11.2.130 SymbCrvMatchingOffset (moffset.c:50)

```
CagdCrvStruct *SymbCrvMatchingOffset(CagdCrvStruct *Crv,
                                     CagdRType OffsetDist,
                                     CagdRType Tolerance)
```

Crv: To approximate its offset curve with distance OffsetDist.

OffsetDist: Amount of offset. Negative denotes other offset direction.

Tolerance: Of angular discrepancy that is allowed.

Returns: The offset curve approximation.

Description: Computes an offset to a freeform curve using matching of tangent fields. The given curve is split at all its inflection points, made sure it spans less than 90 degrees, and then is matched against an arc of the proper angular span of tangents. Unlike other offset methods, this method allways preserves the distance between the original curve ans its offset. The error in this methods can surface only in the non orthogonality of the offset direction.

See also: SymbCrvOffset, SymbCrvSubdivOffset, SymbSrfOffset, SymbSrfSubdivOffset, , SymbCrvAdapOffset, SymbCrvAdapOffsetTrim, SymbCrvLeastSquarOffset, , SymbCrvCrvConvolution,

11.2.131 SymbCrvMergeScalar (symb_crv.c:1488)

merge

symbolic computation

```
CagdCrvStruct *SymbCrvMergeScalar(const CagdCrvStruct *CrvW,
                                  const CagdCrvStruct *CrvX,
                                  const CagdCrvStruct *CrvY,
                                  const CagdCrvStruct *CrvZ)
```

CrvW: The weight component of new constructed curve, if have any.

CrvX: The X component of new constructed curve.

CrvY: The Y component of new constructed curve, if have any.

CrvZ: The Z component of new constructed curve, if have any.

Returns: A new curve constructed from given scalar curves.

Description: Given a set of scalar curves, treat them as coordinates into a new curve. Assumes at least CrvX is not NULL in which a scalar curve is returned. Assumes CrvX/Y/Z/W are either E1 or P1 in which the weights are assumed to be identical and can be ignored if CrvW exists or copied otheriwse.

See also: SymbSrfMergeScalar, SymbCrvSplitScalar,

11.2.132 SymbCrvMergeScalarN (symb_crv.c:1408)

merge

symbolic computation

```
CagdCrvStruct *SymbCrvMergeScalarN(CagdCrvStruct * const *CrvVec,
                                   int NumCrvs)
```

CrvVec: A vector of scalar curves.

NumCrvs: Number of curves in CrvVec.

Returns: A new curve constructed from given scalar curves.

Description: Given a vector of scalar curves, treat them as coordinates into a new vector curve. Assumes at least CrvVec[1] is not NULL in which case a scalar curve is returned. Assumes CrvVec[i] are either E1 or P1 in which case the weights are assumed to be identical and can be simply copied if exist.
See also: SymbSrfMergeScalar, SymbCrvSplitScalarN,

11.2.133 SymbCrvMonotoneCtlPt (composit.c:62)

```
int SymbCrvMonotoneCtlPt(const CagdCrvStruct *Crv, int Axis)
```

Crv: Curve to examine the the monotonicity in axis Axis.

Axis: The axis of Crv to examine the monotonicity for.

Returns: 0 if a completely zero curve (up to tight eps), +1/-1 for increasing/decreasing monotone, 9 if a non monotone curve.

Description: A function to examine if the given curve has a monotone control polygon in the prescribed axis.
See also: SymbCrvPosNegWeights,

11.2.134 SymbCrvMult (symb_crv.c:203)

product

symbolic computation

```
CagdCrvStruct *SymbCrvMult(const CagdCrvStruct *Crv1,
                           const CagdCrvStruct *Crv2)
```

Crv1, Crv2: Two curve to multiply coordinate-wise.

Returns: The product of Crv1 * Crv2 coordinate-wise.

Description: Given two curves - multiply them coordinate-wise. The two curves are promoted to same point type before the multiplication can take place.
See also: SymbCrvDotProd, SymbCrvVecDotProd, SymbCrvInvert, SymbCrvMultScalar,

11.2.135 SymbCrvMultScalar (symb_crv.c:487)

product

symbolic computation

```
CagdCrvStruct *SymbCrvMultScalar(const CagdCrvStruct *Crv1,
                                 const CagdCrvStruct *Crv2)
```

Crv1, Crv2: Two curve to multiply. Crv1 is assume at most of dimension 3.

Returns: A curve representing the product of Crv1 and Crv2.

Description: Given two curves - a vector curve Crv1 and a scalar curve Crv2, multiply all Crv1's coordinates by the scalar curve Crv2. Returned curve is a curve representing the product of the two given curves.
See also: SymbCrvDotProd, SymbCrvVecDotProd, SymbCrvMult, SymbCrvCrossProd, , SymbSrfMultScalar,

11.2.136 SymbCrvMultiResBWavelet (multires.c:860)

```
CagdCrvStruct *SymbCrvMultiResBWavelet(CagdRType *KV,
                                       int Order,
                                       int Len,
                                       int KnotIndex)
```

KV: Knot sequence of the space.

Order: Order of space.

Len: Length of knot sequence.

KnotIndex: Index of knot to compute the wavelet for.

Returns: A scalar curve representing the wavelet.

Description: Constructs the B-Wavelet of knot KV[KnotIndex] for the given knot sequence KV and order Order.
See also: SymbBspBasisInnerProd,

11.2.137 SymbCrvMultiResCompos (multires.c:537)

multi resolution

least square decomposition

```
CagdCrvStruct *SymbCrvMultiResCompos(const SymbMultiResCrvStruct *MRCrv)
```

MRCrv: A multi resolution decomposition of a curve.

Returns: A curve that adds up all components of the multi resolution decomposition MRCrv.

Description: Given a multi resolution decomposition of a Bspline curve, computes the regular Bspline curve out of it.

11.2.138 SymbCrvMultiResComposAtT (multires.c:566)

multi resolution

least square decomposition

```
CagdCrvStruct *SymbCrvMultiResComposAtT(const SymbMultiResCrvStruct *MRCrv,
                                        CagdRType T)
```

MRCrv: A multi resolution decomposition of a curve.

T: A mult resolution hierarcy level to compute curve for.

Returns: A curve that adds up all components of the multi resolution decomposition MRCrv up to and including level T.

Description: Given a multiresolution decomposition of a Bspline curve, computes a regular Bspline curve out of it representing the decomposed curve at the multi resolution hierarchy level of T. Although decomposition is discrete, T can be any real number between these discrete levels and a linear interpolation of adjacent levels is exploited.

11.2.139 SymbCrvMultiResCopy (multires.c:1136)

multi resolution

least square decomposition

```
SymbMultiResCrvStruct *SymbCrvMultiResCopy(const SymbMultiResCrvStruct
                                                              *MRCrvOrig)
```

MRCrvOrig: A multi resolution decomposition of a curve to copy.

Returns: A duplicated structure of MRCrv.

Description: Given a multi resolution decomposition of a Bspline curve, copy it.

11.2.140 SymbCrvMultiResDecomp (multires.c:180)

multi resolution

least square decomposition

```
SymbMultiResCrvStruct *SymbCrvMultiResDecomp(const CagdCrvStruct *Crv,
                                             CagdBType Discont)
```

Crv: To compute a least square multi resolution decomposition for.

Discont: Do we want to preserve discontinuities?

Returns: A multi resolution curve structure hold the multi resolution decomposition of Crv.

Description: Given a B-spline curve, computes a hierarchy of B-spline curves, each being represented using a subspace of the previous, up to a curve with no interior knots (i.e. a polynomial Bezier). However, if Discont == TRUE, then C1 discontinuities are preserved through out the hierarchy decomposition. Each level in hierarchy has approximately half the number of control points of the previous one. Least square curve fitting is used to build the hierarchy.

See also: SymbCrvMultiResDecomp2,

11.2.141 SymbCrvMultiResDecomp2 (multires.c:317)

multi resolution

B-Wavelet decomposition

```
SymbMultiResCrvStruct *SymbCrvMultiResDecomp2(const CagdCrvStruct *Crv,
                                              CagdBType Discont,
                                              CagdBType SameSpace)
```

Crv: To compute a B-Wavelet decomposition for.

Discont: Do we want to preserve discontinuities?

SameSpace: If this curve is in the same space as last curve, exploit this in optimizing the computation cost.

Returns: A B-Wavelet decomposition structure hold the multi resolution decomposition of Crv.

Description: Given a Bspline curve, computes a hierarch of Bspline curves, each being represented using a subspace of the previous, upto a curve with no interior knots (i.e. a polynomial Bezier). However, if Discont == TRUE, then C1 discontinuities are preserved through out the hierarchy decomposition. Each level in hierarchy has approximately half the number of control points of the previous one. B-Wavelet decomposition is used to build the hierarchy. See R. Kazinnik and G. Elber, "Orthogonal Decomposition of Non Uniform Bspline Spaces using Wavelets", Eurographics 1997.

See also: SymbCrvMultiResDecomp,

11.2.142 SymbCrvMultiResEdit (multires.c:624)

multi resolution

least square decomposition

```
void SymbCrvMultiResEdit(const SymbMultiResCrvStruct *MRCrv,
                         CagdRType t,
                         const CagdVType TransDir,
                         CagdRType Level,
                         CagdRType FracLevel)
```

MRCrv: A multi resolution decomposition of a curve to edit it in place.

t: Parameter value at which to modify MRCrv.

TransDir: Directional tranlation transformation to apply.

Level: Of multi resolution hierarchy to edit.

FracLevel: The fraction level to edit - will blend two neighboring levels.

Returns: void

Description: Given a multi resolution decomposition of a Bspline curve, edit it by modifying its Level'th Level according to the TransDir of Position at parametr t. Level can be a fraction number between the discrete levels of the decomposition denoting a linear blend of two neighboring discrete levels. Editing is performed in place.

11.2.143 SymbCrvMultiResFree (multires.c:1074)

multi resolution

least square decomposition

```
void SymbCrvMultiResFree(SymbMultiResCrvStruct *MRCrv)
```

MRCrv: A multi resolution decomposition of a curve to free.

Returns: void

Description: Given a multi resolution decomposition of a Bspline curve, free it.

11.2.144 SymbCrvMultiResKVBuild (multires.c:42)

```
CagdBType SymbCrvMultiResKVBuild(const CagdCrvStruct *Crv,
                                 CagdBType Discont,
                                 CagdRType ***KVList,
                                 int **KVListSizes,
                                 int *KVListSize)
```

Crv: Curve to construct a hierarchy of knot sequences.

Discont: Should we preserve discontinuities as much as we can?

KVList: A vector of pointers to knot sequences in the hierarchy.

KVListSizes: Length of each knot sequence in vector KVList.

KVListSize: Size of KVList - number of knot sequences in hierarchy.

Returns: TRUE if successful, FALSE otherwise.

Description: Constructs a hierarchy of knot sequence for the given B-spline curve and until no interior knot exists in the knot sequence.
See also: SymbCrvMultiResDecomp, SymbCrvMultiResDecomp2,

11.2.145 SymbCrvMultiResNew (multires.c:1104)

multi resolution

least square decomposition

```
SymbMultiResCrvStruct *SymbCrvMultiResNew(int Levels, CagdBType Periodic)
```

Levels: Number of levels to expect in the decomposition.

Periodic: Is the curve periodic?

Returns: A structure to hold a multi resolution decomposition of a curve of Levels levels.

Description: Allocates a data structure for multi resolution decomposition of a Bspline curve of Levels levels and possiblt periodic.

11.2.146 SymbCrvMultiResRefineLevelMalloc (multires.c:829)

multi resolution

least square decomposition

```
CagdRType *SymbCrvMultiResRefineLevelMalloc(SymbMultiResCrvStruct *MRCrv,
                                            CagdRType T,
                                            CagdBType SpanDiscont)
```

MRCrv: A multi resolution decomposition of a curve, to refine in place.

T: Parameter value at which to refine MRCrv.

SpanDiscont: Do we want to refine beyond discontinuities?

Returns: A pointer to an array of two real numbers holding the domain in MRCrv that was refined.

Description: Given a multi resolution decomposition of a Bspline curve, refine it at neighborhood of parameter value t, in place.

11.2.147 SymbCrvMultiResRefineLevelToData (multires.c:734)

multi resolution

least square decomposition

```
CagdRType *SymbCrvMultiResRefineLevelToData(SymbMultiResCrvStruct *MRCrv,
                                            CagdRType T,
                                            CagdBType SpanDiscont,
                                            CagdRType *Domain)
```

MRCrv: A multi resolution decomposition of a curve, to refine in place.

T: Parameter value at which to refine MRCrv.

SpanDiscont: Do we want to refine beyond discontinuities?

Domain: A pointer to an array of two real numbers holding the domain in MRCrv that was refined.

Returns: A pointer to an array of two real numbers holding the domain in MRCrv that was refined.

Description: Given a multi resolution decomposition of a Bspline curve, refine it at neighborhood of parameter value t, in place.

11.2.148 SymbCrvOffset (offset.c:72)

offset

```
CagdCrvStruct *SymbCrvOffset(const CagdCrvStruct *CCrv,
                             CagdRType OffsetDist,
                             CagdBType BezInterp)
```

CCrv: To approximate its offset curve with distance OffsetDist.

OffsetDist: Amount of offset. Negative denotes other offset direction.

BezInterp: If TRUE, control points are interpolated when the curve is reduced to a Bezier form. Otherwise, control points are translated OffsetDist amount only, under estimating the Offset.

Returns: An approximation to the offset curve.

Description: Given a curve and an offset amount OffsetDist, returns an approximation to the offset curve by offseting the control polygon in the normal direction.
See also: SymbCrvSubdivOffset, SymbSrfOffset, SymbSrfSubdivOffset, , SymbCrvAdapOffset, SymbCrvAdapOffsetTrim, SymbCrvLeastSquarOffset, , SymbCrvMatchingOffset, SymbCrvVarOffset,

11.2.149 SymbCrvOffset2CrvsJoint (offset.c:2038)

```
CagdCrvStruct *SymbCrvOffset2CrvsJoint(CagdCrvStruct *OrigCrv1,
                                       CagdCrvStruct *OrigCrv2,
                                       CagdCrvStruct **OffCrv1,
                                       CagdCrvStruct **OffCrv2)
```

OrigCrv1: To examine its end against Crv2, if the same.

OrigCrv2: To examine its start against Crv1, if the same.

OffCrv1: To properly update its end against Crv2, if needs to.

OffCrv2: To properly update its start against Crv1, if needs to.

Returns: A new round curve bnetween OffCrv1 and OffCrv2 if not NULL. Note also that OffCrv1/2 might be changed in place.

Description: Update the end of Crv1 and the beginning of Crv2 to connect after an offset operation that was applied to them. There are a few options:

1. End of Crv1 is the same of start of Crv (if the two curves are C^1 connected). Do nothing.
2. The two curves intersects. Trim away end of Crv1 and start of Crv2 up to the intersection location. This case is handled assuming small interaction between the two curves only, seeking a single intersection. Note old OfCrv1/2 are freed and being substituted in place with the new trimmed versions.
3. The two curves do not intersect. Add a joint round curve that is C^1 to both Crv1's end and Crv2's start. The new joint round curve is returned.

11.2.150 SymbCrvOrthotomic (orthotom.c:38)

```
CagdCrvStruct *SymbCrvOrthotomic(const CagdCrvStruct *Crv,
                                 const CagdPType P,
                                 CagdRType K)
```

Crv: To compute its K-orthotomic

P: The points to which the K-orthotomic is computed for Crv for.

K: The magnitude of the orthotomic function.

Returns: The K-orthotomic

Description: Computes the K-orthotomic of a curve with respect to point P:

```
P + K < (C(t) - P), N(t) > N(t)
```

See "Fundamentals of Computer Aided Geometric Design", by J. Hoschek and and D. Lasser.
See also: SymbSrfOrthotomic,

11.2.151 SymbCrvPointInclusion (distance.c:485)

curve point inclusion

```
int SymbCrvPointInclusion(const CagdCrvStruct *CCrv,
                          const CagdPType Pt,
                          CagdRType Epsilon)
```

CCrv: Closed planar curve to examine for the inclusion of Pt.

Pt: Point tp test for inclusion in Crv.

Epsilon: Accuracy of computation.

Returns: 1 if Pt inside Crv, -1 otherwise, 0 if on boundary to within Epsilon.

Description: Given a closed planar curve and a point, finds if point is inside curve. Uses winding number accumulation in the computation.
See also: SymbCrvPointInclusion, SymbDistCrvLine, SymbCrvRayInter,

11.2.152 SymbCrvPosNegWeights (symbzero.c:513)

symbolic computation

```
CagdBType SymbCrvPosNegWeights(const CagdCrvStruct *Crv)
```

Crv: To examine for same sign weights, if any.

Returns: TRUE if no weights or all of same sign.

Description: Returns TRUE iff the Crv is not rational or rational with weights that are entirely positive or entirely negative.
See also: CagdCrvBBox, CagdSrfBBox, GMBBSetBBoxPrecise, SymbCrvPosNegWeights, , CagdAllWeightsSame,

11.2.153 SymbCrvPtBisectorCrv2D (crv_skel.c:1214)

bisector

```
CagdCrvStruct *SymbCrvPtBisectorCrv2D(const CagdCrvStruct *CCrv,
                                      const CagdPType Pt,
                                      CagdRType Alpha)
```

CCrv: Planar curve to compute its bisector curve with Pt.

Pt: A point in the plane to compute its bisector with Crv.

Alpha: Alpha-sector ratio (0.5 for a bisector).

Returns: The bisector curve, in the XY plane.

Description: Computes the alpha-/bi-sector curve of a planar curve a point, all in the XY plane. The result is the solution to the following two linear equations in alpha-/bi-sector's two unknowns, the x and y coefficients:

```
<C'(t),     B(t)> = <C'(t), C(t)>
<C(t) - Pt, B(t)> = <C(t) - Pt, a Pt + (1 - a) C(t)>
```

where a is the Alpha of the alpha-sector, 0.5 for a bisector, Pt is the point entity, C(t) is the curve entity and B(t) is the sought bisector.

See also: SymbCrvDiameter, SymbCrvCnvxHull, SymbCrvBisectorsSrf, , SymbCrvCrvBisectorSrf3D, SymbSrfPtBisectorSrf3D, SymbCrvPtBisectorSrf3D,

11.2.154 SymbCrvPtBisectorSrf3D (crv_skel.c:1358)

bisector

```
CagdSrfStruct *SymbCrvPtBisectorSrf3D(const CagdCrvStruct *CCrv,
                                      const CagdPType Pt,
                                      CagdRType RulingScale)
```

CCrv: Three space curve to compute its bisector surface with Pt.

Pt: A point in three space to compute its bisector with Crv.

RulingScale: The scaling factor for the ruling direction.

Returns: The bisector surface.

Description: Computes the bisector surface of a curve in arbitrary general three space position and a point in three space.

See also: SymbCrvDiameter, SymbCrvCnvxHull, SymbCrvBisectorsSrf, , SymbCrvCrvBisectorSrf3D, SymbSrfPtBisectorSrf3D, SymbCrvPtBisectorCrv2D,

11.2.155 SymbCrvPtTangents (crv_tans.c:48)

```
CagdPtStruct *SymbCrvPtTangents(const CagdCrvStruct *CCrv,
                                const CagdPType Pt,
                                CagdRType Tolerance)
```

CCrv: To compute its tangent lines through Pt.

Pt: Point of origin, all tangents to Crv goes through.

Tolerance: Accuracy of computation.

Returns: A list of parameter location on Crv with tangent lines through Pt. Parameters are save in the X coordinate.

Description: Computes the points on a C1 freeform planar Bspline curve, Crv, that a line tangent to Crv there goes through point Pt. That is,

```
(C(t) - P) || C'(t),
```

where || denotes a parallel constraint.

See also: SymbCrvCnvxHull, SymbCircTanTo2Crvs, SymbTangentToCrvAtTwoPts, , mbCrvDiameter,

11.2.156 SymbCrvRayInter (distance.c:589)

curve ray intersection

```
CagdPtStruct *SymbCrvRayInter(const CagdCrvStruct *Crv,
                              const CagdPType RayPt,
                              const CagdVType RayDir,
                              CagdRType Epsilon)
```

Crv: The curve to find its intersections with the ray.

RayPt, RayDir: The ray prescription.

Epsilon: Accuracy of computation.

Returns: A list of parameter values of the ray-curve intersections.

Description: Given a curve and a ray, finds the intersection points on the curve where the ray pierces the curve. Returned is a list of parameter value with local extreme distances. Let Crv be (x(t), y(t)). By substituting x(t) and y(t) into the line equation of the ray, we derive the distance function whose zero set captures all curve-line intersections. They are then filtered to those on the half-line ray.

See also: SymbDistCrvLine, MvarDistSrfLine, SymbLclDistCrvLine, , GMPolygonXYRayInter,

11.2.157 SymbCrvRtnlMult (symb_crv.c:783)

product

symbolic computation

```
CagdCrvStruct *SymbCrvRtnlMult(const CagdCrvStruct *Crv1X,
                               const CagdCrvStruct *Crv1W,
                               const CagdCrvStruct *Crv2X,
                               const CagdCrvStruct *Crv2W,
                               CagdBType OperationAdd)
```

Crv1X: Numerator of first curve.

Crv1W: Denominator of first curve. Can be NULL.

Crv2X: Numerator of second curve.

Crv2W: Denominator of second curve. Can be NULL.

OperationAdd: TRUE for addition, FALSE for subtraction.

Returns: The result of Crv1X Crv2W +/- Crv2X Crv1W.

Description: Given two curves - multiply them using the quotient product rule:

```
X = X1 W2 +/- X2 W1
```

All provided curves are assumed to be non rational scalar curves. Returned is a non rational scalar curve (CAGD_PT_E1_TYPE).
See also: SymbCrvDotProd, SymbCrvVecDotProd, SymbCrvMult, SymbCrvMultScalar,

11.2.158 SymbCrvScalarScale (symb_crv.c:339)

scaling

symbolic computation

```
CagdCrvStruct *SymbCrvScalarScale(const CagdCrvStruct *Crv, CagdRType Scale)
```

Crv: A curve to scale by magnitude Scale.

Scale: Scaling factor.

Returns: A curves scaled by Scale compared to Crv.

Description: Given a curve, scale it by Scale.
See also: SymbCrvDotProd, SymbCrvVecDotProd, SymbCrvMult, SymbCrvMultScalar,

11.2.159 SymbCrvSliceCrvsByPrllLines (symbzero.c:339)

slices

contours.

```
CagdCrvStruct *SymbCrvSliceCrvsByPrllLines(const CagdCrvStruct *Crvs,
                                           int Axis,
                                           CagdRType Epsilon,
                                           CagdVType Range)
```

Crvs: A set of curves forming closed region(s) to slice and compute a set of parallel line segments in the region(s), in XY.

Axis: The axis of line compute the sliced line, X = 1, Y = 2.

Epsilon: Tolerance control.

Range: A vector setting the (min, max, step) of the parallel lines, as (XMin, XMax, XStep) if Axis = 1, (YMin, YMax, YStep) if Axis = 2.

Returns: List of parallel lines in the closed region, enclosed by Crvs.

Description: Computes a set of parallel line segments included inside the given set of Crvs, that is assumed to define closed region(s), in the XY plane.
See also: SymbCrvConstSet,

11.2.160 SymbCrvSplitPoleParams (symbzero.c:623)

```
CagdCrvStruct *SymbCrvSplitPoleParams(const CagdCrvStruct *CCrv,
                                      CagdRType Eps,
                                      CagdRType OutReach)
```

CCrv: Rational curve to split at all its poles.

Eps: Tolerance of computation.

OutReach: Clip end points of curves that goes to infinity at distance that is about OutReach from the origin.

Returns: List of splitted curves.

Description: Splits the given rational curve at all its poles. Returned is a list of curves each of which has weights of the same (positive) sign.
See also: CagdPointsHasPoles, SymbCrvsSplitPoleParams,

11.2.161 SymbCrvSplitScalar (symb_crv.c:1353)

split

symbolic computation

```
void SymbCrvSplitScalar(const CagdCrvStruct *Crv,
                        CagdCrvStruct **CrvW,
                        CagdCrvStruct **CrvX,
                        CagdCrvStruct **CrvY,
                        CagdCrvStruct **CrvZ)
```

Crv: Curve to split.

CrvW: The weight component of Crv, if have any.

CrvX: The X component of Crv.

CrvY: The Y component of Crv, if have any.

CrvZ: The Z component of Crv, if have any.

Returns: void

Description: Given a curve splits it to its scalar component curves. Ignores all dimensions beyond the third, Z, dimension.
See also: SymbSrfSplitScalar, SymbCrvMergeScalar,

11.2.162 SymbCrvSplitScalarN (symb_crv.c:1304)

split

symbolic computation

```
void SymbCrvSplitScalarN(const CagdCrvStruct *Crv, CagdCrvStruct **Crvs)
```

Crv: Curve to split.

Crvs: A vector of scalar curves - components of Crv. A vector of dynamically allocated scalar crvs.

Returns: void

Description: Given a curve, splits it to its scalar component curves.
See also: SymbSrfSplitScalar, SymbSrfSplitScalarN, SymbCrvMergeScalarN,

11.2.163 SymbCrvSqrtScalar (arc_len.c:174)

square root

```
CagdCrvStruct *SymbCrvSqrtScalar(const CagdCrvStruct *OrigCrv,
                                 CagdRType Epsilon)
```

OrigCrv: Scalar curve to approximate its square root function.

Epsilon: Accuracy of approximation.

Returns: A curve approximating the square root of OrigCrv.

Description: Computes the curve which is a square root approximation to a given scalar curve, to within epsilon.

11.2.164 SymbCrvSub (symb_crv.c:116)

subtraction

symbolic computation

```
CagdCrvStruct *SymbCrvSub(const CagdCrvStruct *Crv1, const CagdCrvStruct *Crv2)
```

Crv1, Crv2: Two curve to subtract coordinate-wise.

Returns: The difference of Crv1 - Crv2 coordinate-wise.

Description: Given two curves - subtract them coordinate-wise. The two curves are promoted to same point type before the multiplication can take place. Furthermore, order and continuity are matched as well.
See also: SymbCrvAdd, SymbCrvMult,

11.2.165 SymbCrvSubdivOffset (offset.c:424)

offset

```
CagdCrvStruct *SymbCrvSubdivOffset(const CagdCrvStruct *CCrv,
                                   CagdRType OffsetDist,
                                   CagdRType Tolerance,
                                   CagdBType BezInterp)
```

CCrv: To approximate its offset curve with distance OffsetDist.

OffsetDist: Amount of offset. Negative denotes other offset direction.

Tolerance: Accuracy control.

BezInterp: If TRUE, control points are interpolated when the curve is reduced to a Bezier form. Otherwise, control points are translated OffsetDist amount only, under estimating the Offset.

Returns: An approximation to the offset curve, to within Tolerance.

Description: Given a curve and an offset amount OffsetDist, returns an approximation to the offset curve by offseting the control polygon in the normal direction. If resulting offset is not satisfying the required tolerance the curve is subdivided and the algorithm recurses on both parts.
See also: SymbCrvOffset, SymbSrfOffset, SymbSrfSubdivOffset, SymbCrvAdapOffset, , SymbCrvAdapOffset-Trim, SymbCrvLeastSquarOffset, SymbCrvMatchingOffset,

11.2.166 SymbCrvTrimGlblOffsetSelfInter (offset.c:1555)

```
CagdCrvStruct *SymbCrvTrimGlblOffsetSelfInter(const CagdCrvStruct *Crv,
                                              const CagdCrvStruct *OffCrv,
                                              CagdRType SubdivTol,
                                              CagdRType TrimAmount,
                                              CagdRType NumerTol)
```

Crv: Original curve.

OffCrv: The offset curve approximation.

SubdivTol: Accuracy of computation. 0.001 will be a good start.

TrimAmount: The trimming distance. A fraction smaller than the offset amount.

NumerTol: If Positive, a numerical marching improvement step is applied with NumerTol tolerance to the derived intersection/clipped regions. Assumes derivative of offset curve exists.

Returns: A list of curve segments that are valid, after the trimming process took place.

Description: Trims regions in the offset curve OffCrv that are closer than TrimAmount to original Crv. TrimAmount should be a fraction smaller than the offset amount itself. See all: Gershon Elber. "Trimming Local and Global Self-intersections in Offset Curves using Distance Maps." The 10th IMA conference on the Mathematics of Surfaces, Leeds, UK, pp 213-222, September 2003, LLCS2768.
See also: MvarCrvTrimGlblOffsetSelfInter,

11.2.167 SymbCrvUnitLenCtlPts (arc_len.c:671)

unit vector field

```
CagdCrvStruct *SymbCrvUnitLenCtlPts(const CagdCrvStruct *CCrv)
```

CCrv: Curve to approximate a unit magnitude for.

Returns: Similar curve of Crv, but with unit length control points.

Description: Normalizes all the control points of the given (vector field) curve. This results in an approximated unit speed vector field.
See also: SymbCrvUnitLenScalar,

11.2.168 SymbCrvUnitLenScalar (arc_len.c:40)

unit vector field

```
CagdCrvStruct *SymbCrvUnitLenScalar(const CagdCrvStruct *OrigCrv,
                                    CagdBType Mult,
                                    CagdRType Epsilon)
```

OrigCrv: Curve to approximate a unit size for.

Mult: Do we want to multiply the computed scalar curve with Crv?

Epsilon: Accuracy required of this approximation.

Returns: A scalar curve to multiply OrigCrv so a unit size curve will result if Mult is FALSE, or the actual unit size vector field curve, if Mult.

Description: Normalizes the given vector field curve to be a unit length curve, by computing a scalar curve to multiply with this vector field curve. Returns the multiplied curve if Mult, or otherwise just the scalar curve.
See also: SymbCrvUnitLenCtlPts,

11.2.169 SymbCrvVarOffset (offset.c:243)

offset

```
CagdCrvStruct *SymbCrvVarOffset(const CagdCrvStruct *CCrv,
                                const CagdCrvStruct *VarOffsetDist,
                                CagdBType BezInterp)
```

CCrv: To approximate its variable offset curve.

VarOffsetDist: Scalar function prescribing the amount of offset. Must posses a parametric domain similar to CCrv.

BezInterp: If TRUE, control points are interpolated when the curve is reduced to a Bezier form. Otherwise, control points are translated OffsetDist amount only, under estimating the Offset.

Returns: An approximation to the varying offset amount curve.

Description: Given a curve and an offset amount function Var OffsetDist, returns an approximation to the offset curve by offseting the control polygon in the normal direction.
See also: SymbCrvSubdivOffset, SymbSrfOffset, SymbCrvMatchingOffset, , SymbCrvAdapOffset, SymbCrvAdapOffsetTrim, SymbCrvLeastSquarOffset,

11.2.170 SymbCrvVecCrossProd (symb_crv.c:669)

product

cross product

symbolic computation

```
CagdCrvStruct *SymbCrvVecCrossProd(const CagdCrvStruct *Crv,
                                   const CagdVType Vec)
```

Crv: Curve to multiply and compute a cross product for.

Vec: Vector to cross product Crv with.

Returns: A vector curve representing the cross product of Crv x Vec.

Description: Given a curve and a vector - computes their cross product. Returned curve is a scalar curve representing the cross product of the curve and vector.
See also: SymbCrvDotProd, SymbCrvVecDotProd, SymbCrvScalarScale, SymbCrvMultScalar, , SymbCrvInvert, SymbCrvCrossProd,

11.2.171 SymbCrvVecDotProd (symb_crv.c:425)

product

dot product

symbolic computation

```
CagdCrvStruct *SymbCrvVecDotProd(const CagdCrvStruct *Crv,
                                 const CagdRType *Vec)
```

Crv: Curve to multiply and compute a dot product for.

Vec: Vector to project Crv onto.

Returns: A scalar curve representing the dot product of Crv . Vec.

Description: Given a curve and a vector of any dimension - computes their dot product. Returned curve is a scalar curve representing the dot product.
See also: SymbCrvDotProd, SymbCrvMult, SymbCrvMultScalar, SymbCrvCrossProd,

11.2.172 SymbCrvZeroSet (symbzero.c:47)

zero set

symbolic computation

```
CagdPtStruct *SymbCrvZeroSet(const CagdCrvStruct *Crv,
                             int Axis,
                             CagdRType Epsilon,
                             CagdBType NoSolsOnEndPts)
```

Crv: To compute its zero set.

Axis: The axis of Crv to compute zero set for, W = 0, X = 1, etc.

Epsilon: Tolerance control.

NoSolsOnEndPts: If TRUE, solutions at the end of the domain are purged.

Returns: List of parameter values form which Crv is zero in axis Axis.

Description: Computes the zero set of a given curve, in given axis (0/1-3 for W/X-Z). Returned is a list of the zero set points holding the parameter values at Pt[0] of each point.

11.2.173 SymbCrvsCompare (cmp_crvs.c:231)

```
SymbCrvRelType SymbCrvsCompare(const CagdCrvStruct *Crv1,
                               const CagdCrvStruct *Crv2,
                               CagdRType Eps,
                               CagdRType *StartOvrlpPrmCrv1,
                               CagdRType *EndOvrlpPrmCrv1,
                               CagdRType *StartOvrlpPrmCrv2,
                               CagdRType *EndOvrlpPrmCrv2)
```

Crv1, Crv2: The two curves to be compared.

Eps: A threshold for numerical computations.

StartOvrlpPrmCrv1: If the curves are the same and overlapping, here the start of overlapping domain of Crv1 will be returned.

EndOvrlpPrmCrv1: If the curves are the same and overlapping, here the end of overlapping domain of Crv1 will be returned.

StartOvrlpPrmCrv2: If the curves are the same and overlapping, here the start of overlapping domain of Crv2 will be returned.

EndOvrlpPrmCrv2: If the curves are the same and overlapping, here the end of overlapping domain of Crv2 will be returned.

Returns: Either Same curves (1), overlapping curves (2), or distinct curves (3). If Overlapping, the Start/End overlapping parametric domain variables are set.

Description: Compares the given two curves. Each curve is converted, if necessary, into a set of Bezier curves, and the comparison is done by applying comparison algorithm for each Bezier curve.

11.2.174 SymbCrvsLowerEnvelop (crv_lenv.c:192)

cci

lower envelop

```
CagdCrvStruct *SymbCrvsLowerEnvelop(const CagdCrvStruct *CCrvs,
                                    CagdRType *Pt,
                                    CagdRType Eps)
```

CCrvs: Curves to derive the lower envelop for.

Pt: Point defining the center of the radial envelop, or NULL for linear, minimum Y, lower envelop.

Eps: Tolerance of computations.

Returns: A list of curves defining the lower envelop.

Description: Computes the lower envelop in the plane of all given curves. Returned is a list of (pieces of) curves that forms the lower envelop. If Pt is not NULL, radial lower envelop around Pt is computed. If Pt is NULL, regular, linear, lower envelop is computed seeking minimum Y values for the X domain that is spanned by the curves. Note the lower envelop might be discontinuous. The lower envelop is computed by splitting the input curves at all intersetion locations, sorting intersection and end point events and shooting rays in the middle of these intervals to determine lowest one.
See also: CagdCrvCrvInter, CagdCrvCrvInterArrangment,

11.2.175 SymbCrvsSplitPoleParams (symbzero.c:585)

```
CagdCrvStruct *SymbCrvsSplitPoleParams(const CagdCrvStruct *Crvs,
                                       CagdRType Eps,
                                       CagdRType OutReach)
```

Crvs: Rational curves to split at all its poles.

Eps: Tolerance of computation.

OutReach: Clip end points of curves that goes to infinity at distance that is about OutReach from the origin.

Returns: List of splitted curves.

Description: Splits the given rational curves at all their poles. Returned is a list of curves each of which has weights of the same (positive) sign.
See also: CagdPointsHasPoles, SymbCrvSplitPoleParams,

11.2.176 SymbCubicBspInjective (bsp3injc.c:107)

```
CagdBType SymbCubicBspInjective(CagdRType x[4][4], CagdRType y[4][4])
```

x: The 4 by 4 array of X coefficients of the cubic patch.

y: The 4 by 4 array of Y coefficients of the cubic patch.

Returns: TRUE if injective, FALSE otherwise.

Description: Examine if the given uniform cubic patch in injective. The patch is assumed to be a mapping from R2 to R2.

11.2.177 SymbCylinCylinBisect (smp_skel.c:1783)

bisectors

skeleton

```
CagdSrfStruct *SymbCylinCylinBisect(const CagdVType Cyl1Pos,
                                    const CagdVType Cyl1Dir,
                                    CagdRType Cyl1Rad,
                                    const CagdVType Cyl2Pos,
                                    const CagdVType Cyl2Dir,
                                    CagdRType Cyl2Rad)
```

Cyl1Pos: A point on the axis of the first cylinder.

Cyl1Dir: The direction of the first cylinder.

Cyl1Rad: Radius of first cylinder.

Cyl2Pos: A point on the axis of the second cylinder.

Cyl2Dir: The direction of the second cylinder.

Cyl2Rad: Radius of second cylinder.

Returns: Constructed bisector surface.

Description: Compute the bisector surface between two cylinders. Let C1(u), T1(u), and N1(u) and C2(v), T2(v), and N2(v) be the cylinders cross section, cross section tangent field and cross section unit normal field. Ci(u) can be derived as a transformed circle. Ti(u) and Ni(u) are unit circles rotated to the proper orientation CyliDir. Finally, note that Ci(u), Ti(u), and Ni(u) are all rational. Then, the bisector is computed as the solution of the following three linear equations:

```
< B - C1(u), T1(u) > = 0
```

```
< B - C2(v), T2(v) > = 0
```

```
< B, N1(u) - N2(v) > = < C1(u), N1(u) > - < C2(v), N2(v) >
```

The first two constraints the bisector to be on the normal plane of the generators of the two cylinders that are fixed along the generator (the straight lines of the cylinder). The last constraint make sure the bisector is on the plane that bisects the two tangent planes of the two cylinders. This computation is following the bisectors of two developables, presented in, "Geometric Properties of Bisector Surfaces", by Martin Peternell, Graphical Models, Volume 62, No. 3, May 2000.

See also: SymbPlanePointBisect, SymbCylinPointBisect, SymbConePointBisect, , SymbSpherePointBisect, SymbTorusPointBisect, , SymbConeLineBisect, SymbSphereLineBisect, SymbPlaneLineBisect, , SymbCylinPlaneBisect, SymbConePlaneBisect, SymbSpherePlaneBisect, , SymbCylinSphereBisect, SymbSphereSphereBisect, SymbConeSphereBisect, , SymbConeConeBisect, SymbConeConeBisect2, SymbConeCylinBisect, ,

11.2.178 SymbCylinPlaneBisect (smp_skel.c:1199)

bisectors

skeleton

```
CagdSrfStruct *SymbCylinPlaneBisect(const CagdPType CylPt,
                                    const CagdVType CylDir,
                                    CagdRType CylRad,
                                    CagdRType Size)
```

CylPt: Point on axis of cylinder.

CylDir: Direction of cylinder. Must be in the northern hemisphere (positive Z coefficient).

CylRad: Radius of cylinder.

Size: Portion of result as it is infinite.

Returns: Constructed bisector surface.

Description: Compute the bisector surface between a cylinder and the XY plane. The computation is reduced to that of a bisector between a line and a plane, that has a rational form. Only the portion for which $Z > 0$ should be considered in the output.

See also: SymbPlanePointBisect, SymbCylinPointBisect, SymbConePointBisect, , SymbSpherePointBisect, SymbTorusPointBisect, , SymbConeLineBisect, SymbSphereLineBisect, SymbPlaneLineBisect, , SymbConePlaneBisect, SymbSpherePlaneBisect, , SymbCylinSphereBisect, SymbSphereSphereBisect, SymbConeSphereBisect, , SymbTorusSphereBisect, SymbTorusTorusBisect, SymbConeConeBisect, , SymbConeConeBisect2, SymbConeCylinBisect, SymbCylinCylinBisect,

11.2.179 SymbCylinPointBisect (smp_skel.c:744)

bisectors

skeleton

```
CagdSrfStruct *SymbCylinPointBisect(const CagdPType CylPt,
                                    const CagdVType CylDir,
                                    CagdRType CylRad,
                                    const CagdPType Pt,
                                    CagdRType Size)
```

CylPt: Point on axis of cylinder.

CylDir: Direction of cylinder.

CylRad: Radius of cylinder.

Pt: Direction of line from origin.

Size: Portion of result as it is infinite.

Returns: Constructed bisector surface.

Description: Compute the bisector surface between a cylinder and a point.
See also: SymbPtCrvBisectOnSphere, , SymbPlanePointBisect, SymbConePointBisect, SymbSpherePointBisect, , SymbTorusPointBisect, , SymbConePlaneBisect, SymbCylinPlaneBisect, SymbSpherePlaneBisect, , SymbConeLineBisect, SymbSphereLineBisect, SymbCylinSphereBisect, SymbSphereSphereBisect, SymbConeSphereBisect, , SymbTorusSphereBisect, SymbTorusTorusBisect, SymbConeConeBisect,

11.2.180 SymbCylinSphereBisect (smp_skel.c:1328)

bisectors

skeleton

```
CagdSrfStruct *SymbCylinSphereBisect(const CagdPType CylPt,
                                     const CagdVType CylDir,
                                     CagdRType CylRad,
                                     const CagdPType SprCntr,
                                     CagdRType SprRad,
                                     CagdRType Size)
```

CylPt: Point on axis of cylinder.

CylDir: Direction of cylinder. Must be in the northern hemisphere (positive Z coefficient).

CylRad: Radius of cylinder.

SprCntr: Center location of the sphere. Must be in northern hemisphere (positive Z coefficient).

SprRad: Radius of sphere.

Size: Portion of result as it is infinite.

Returns: Constructed bisector surface.

Description: Compute the bisector surface between a cylinder and a sphere. The computation is reduced to that of a bisector between a point and a cylinder, that has a rational form.
See also: SymbPlanePointBisect, SymbCylinPointBisect, SymbConePointBisect, , SymbSpherePointBisect, SymbTorusPointBisect, , SymbConeLineBisect, SymbSphereLineBisect, SymbPlaneLineBisect, , SymbCylinPlaneBisect, SymbConePlaneBisect, SymbSpherePlaneBisect, , SymbSphereSphereBisect, SymbConeSphereBisect, SymbTorusSphereBisect, , SymbTorusTorusBisect, SymbConeConeBisect, SymbConeConeBisect2, , SymbConeCylinBisect, SymbCylinCylinBisect,

11.2.181 SymbDecomposeCrvCrv (decompos.c:171)

composition

```
CagdCrvStruct *SymbDecomposeCrvCrv(CagdCrvStruct *Crv)
```

Crv: A curve H(t) to try and decompose.

Returns: Pairs of curves F and G that are composable to the input curve, or NULL if nonreducable. F: [0, 1] -> R^3, G: [0, 1] -> [0, 1].

Description: Try to decompose a curve of arbitrary degree if possible. Among all possibilities, return one pair of two curves F and G which satisfy F(G(t)) = H(t). F can have arbitrary number of coordinates: F(x) = x^m + sum b_j x^j. G should be a scalar monotone curve having a range [0, 1]: G(x) = c_k x^k + ... + c_1 x.
See also: SymbComposeCrvCrv,

11.2.182 SymbDirectionsConeAvgToData (nrmlcone.c:974)

normals

normal bound

```
const SymbNormalConeStruct *SymbDirectionsConeAvgToData(
                                            const CagdRType * const *Pts,
                                            int NumPts,
                                            CagdPointType PType,
                                            SymbNormalConeStruct *NormalCone)
```

Pts: An array of vectors (represented as a control mesh). Supports E2/E3/P2/P3 point types.

NumPts: The number of vectors in the array.

PType: The point type of the vectors in Pts.

NormalCone: The computed directions cone, or NULL if failed.

Returns: The computed directions cone, or NULL if failed. Same as NormalCone.

Description: Computes a directions cone for a given vector field, represented as an array of points (or a control mesh). The angular span of this directions field is computed by averaging the normalized vectors, and finding the maximum deviation of the average direction.
See also: SymbNormalConeForSrfOpt, SymbNormalConeOverlap, SymbTangentConeForCrv, , SymbNormalConeForSrfDoOptimal, SymbNormalConeForSrf,

11.2.183 SymbDirectionsConeOptToData (nrmlcone.c:1064)

normals

normal bound

```
const SymbNormalConeStruct *SymbDirectionsConeOptToData(
                                             const CagdRType * const *Pts,
                                             int NumPts,
                                             CagdPointType PType,
                                             SymbNormalConeStruct *NormalCone)
```

Pts: An array of vectors (represented as a control mesh). Supports E2/E3/P2/P3 point types.

NumPts: The number of vectors in the array.

PType: The point type of the vectors in Pts.

NormalCone: The computed directions cone, or NULL if failed.

Returns: The computed directions cone, or NULL if failed. Same as NormalCone.

Description: Computes a directions cone for a given vector field, represented as an array of points (or a control mesh). The angular span of this directions field is computed by using linear programming.
See also: SymbNormalConeForSrfOpt, SymbNormalConeOverlap, SymbTangentConeForCrv, , SymbNormalConeForSrfDoOptimal, SymbNormalConeForSrf,

11.2.184 SymbDirectionsConeToData (nrmlcone.c:1122)

normals

normal bound

```
const SymbNormalConeStruct *SymbDirectionsConeToData(
                                             const CagdRType * const *Pts,
                                             int NumPts,
                                             CagdPointType PType,
                                             SymbNormalConeStruct *NormalCone)
```

Pts: An array of vectors (represented as a control mesh). Supports E2/E3/P2/P3 point types.

NumPts: The number of vectors in the array.

PType: The point type of the vectors in Pts.

NormalCone: The computed directions cone, or NULL if failed.

Returns: The computed directions cone, or NULL if failed. Same as NormalCone.

Description: Computes a directions cone for a given vector field, represented as an array of points (or a control mesh). The angular span of this directions field is computed either by the "averaging" algorithm or by the "optimal" linear programming algorithm, depending on the global flag .
See also: SymbNormalConeForSrfOpt, SymbNormalConeOverlap, SymbTangentConeForCrv, , SymbNormalConeForSrfDoOptimal, SymbNormalConeForSrf,

11.2.185 SymbDistBuildMapToCrv (distance.c:1157)

```
CagdRType SymbDistBuildMapToCrv(const CagdCrvStruct *Crv,
                                CagdRType Tolerance,
                                CagdRType *XDomain,
                                CagdRType *YDomain,
                                CagdRType **DiscMap,
                                CagdRType DiscMapXSize,
                                CagdRType DiscMapYSize)
```

Crv: The curve to approximate a discrete distance map for.

Tolerance: Tolerance of distance computation.

XDomain: X domain to sample R2 for distances.

YDomain: Y domain to sample R2 for distances.

DiscMap: Where output is saved as a real distance value.

DiscMapXSize: Horizontal resolution, 0 will be mapped to XDomain[0], (DiscMapXSize-1) to XDomain[0].

DiscMapYSize: Vertical resolution, 0 will be mapped to YDomain[0], (DiscMapYSize-1) to YDomain[0].

Returns: Maximal distance of points in prescribed domain to crv Crv.

Description: Compute a discrete distance map to a freeform curve by sampling the distance on a regular grid.
See also: SymbDistCrvPoint,

11.2.186 SymbDistCrvLine (distance.c:335)

curve line distance

```
CagdRType SymbDistCrvLine(const CagdCrvStruct *Crv,
                          const CagdLType Line,
                          CagdBType MinDist,
                          CagdRType Epsilon)
```

Crv: The curve to find its nearest (farthest) point to Line.

Line: The line to find the nearest (farthest) point on Crv to it.

MinDist: If TRUE nearest points is needed, if FALSE farthest.

Epsilon: Accuracy of computation.

Returns: Parameter value in the parameter space of Crv of the nearest (farthest) point to line Line.

Description: Given a curve and a line, finds the nearest point (if MinDist) or the farthest location (if MinDist FALSE) from the curve to the given line. Returned is the parameter value of the curve. Both internal as well as boundary extrema are considered. Let Crv be (x(t), y(t)). By substituting x(t) and y(t) into the line equation, we derive the distance function. Its zero set, combined with the zero set of its derivative provide the needed extreme distances.
See also: SymbLclDistCrvLine, MvarDistSrfLine, SymbCrvRayInter,

11.2.187 SymbDistCrvPoint (distance.c:75)

curve point distance

```
CagdRType SymbDistCrvPoint(const CagdCrvStruct *Crv,
                           void *CrvPtPrepHandle,
                           const CagdRType *Pt,
                           CagdBType MinDist,
                           CagdRType Epsilon,
                           CagdPointType DistSpace)
```

Crv: The curve to find its nearest (farthest) point to Pt.

CrvPtPrepHandle: If not NULL, holds pre-processed data to speed up the curve - points distance computations, for multiple curve - point tests (same curve for all points). See SymbDistCrvPointPrep and SymbDistCrvPointFree.

Pt: The point to find the nearest (farthest) point on Crv to it.

MinDist: If TRUE nearest points is needed, if FALSE farthest.

Epsilon: Accuracy of computation.

DistSpace: if not CAGD_PT_NONE, use this space to compute distances. Can be used, for example, to compute XY distances in a 3D curve. Otherwise, the point space of Crv is used.

Returns: Parameter value in the parameter space of Crv of the nearest (farthest) point to point Pt.

Description: Given a curve and a point, finds the nearest point (if MinDist) or the farthest location (if MinDist FALSE) from the curve to the given point. Returned is the parameter value of the curve. Both internal as well as boundary extrema are considered. Computes the zero set of (Crv(t) - Pt) . Crv'(t), and also look at the curves' end points, and all C^1 discontinuities.
See also: SymbLclDistCrvPoint, MvarDistSrfPoint, SymbDistCrvPointPrep, , SymbDistCrvPointFree,

11.2.188 SymbDistCrvPointFree (distance.c:223)

curve point distance

```
void SymbDistCrvPointFree(void *CrvPtPrepHandle)
```

CrvPtPrepHandle: Handle on the preprocessed curve for multiple point - curve test, to free.

Returns: void

Description: Free the pre-processed data structure, given using Handle.
See also: SymbLclDistCrvPoint, SymbDistCrvPointPrep,

11.2.189 SymbDistCrvPointPrep (distance.c:180)

curve point distance

```
void *SymbDistCrvPointPrep(const CagdCrvStruct *CCrv)
```

CCrv: The curve to pre-process for multi-point - curve tests.

Returns: A handle on a structure to boost multi-point - curve tests.

Description: Given a curve, pre-process it so many curve-distance test can be efficiently made.
See also: SymbDistCrvPointFree, SymbLclDistCrvPoint,

11.2.190 SymbDistPoint1DWithEnergy (ffptdist.c:359)

point distribution

```
IrtRType *SymbDistPoint1DWithEnergy(int N,
                                    IrtRType XMin,
                                    IrtRType XMax,
                                    CagdCrvStruct *CrvtrCrv,
                                    CagdCrvStruct *Deriv1MagSqrCrv,
                                    int Resolution,
                                    SymbDistEnergy1DFuncType EnergyFunc)
```

N: Number of points to distribute,

XMin: Minimum of domain to distribute points.

XMax: Minimum of domain to distribute points.

CrvtrCrv: The curvature field.

Deriv1MagSqrCrv: Will Contain the curvature square and arclength square.

Resolution: Fineness of integral calculation.

EnergyFunc: Energy function to use.

Returns: A vector of N points distributed as requested.

Description: Distributes N points with a given energy in the region in the X line that is bounded by XMin, XMax. Energy is specified via the EnergyFunc that recieves the X location. Resolution * N specifies how many samples to take from EnergyFunc. Returns an array of N distributed points. The solution to the distribution is analythic provided EnergyFunc can be integrated. Herein, this integral is computed nomerically.

11.2.191 SymbEnvOffsetFromCrv (moffset.c:339)

```
CagdSrfStruct *SymbEnvOffsetFromCrv(const CagdCrvStruct *Crv,
                                    CagdRType Height,
                                    CagdRType Tolerance)
```

Crv: The curve to process.

Height: The height of the elevated surface (also the width of the offset operation.

Tolerance: Accuracy of the elevated surface approximation.

Returns: A freeform surface approximating the elevated surface for open curve Crv, or two surfaces for the case of closed curve Crv.

Description: Computes an elevated surface emenating from the given C^1 continuous curve in all directions like a fire front. The surface gets away from Crv in a slope of 45 degrees. This elevated surface is an approximation of the real envelope only, as prescribed by Tolerance. If the given curve is closed, it is assume to be C^1 at the end point as well. For a close curve, two surfaces are actually returned - one for the inside and one for the outside firefront. This function employs SymbCrvSubdivOffset for the offset computations.

11.2.192 SymbEvalCrvCurvPrep (evalcurv.c:33)

curvature

```
void SymbEvalCrvCurvPrep(const CagdCrvStruct *Crv, CagdBType Init)
```

Crv: Curve to preprocess. Its attributes might be affected though.

Init: TRUE for initializing, FALSE for clearing out.

Returns: void

Description: Preprocess a given curve so we can evaluate curvature properties from it efficiently, at every point. See SymbEvalCurvature for actual curvature at curve point evaluations.
See also: SymbEvalCurvature, SymbEvalCrvCurvTN,

11.2.193 SymbEvalCrvCurvTN (evalcurv.c:143)

curvature

```
void SymbEvalCrvCurvTN(CagdVType Nrml,
                       CagdVType Tan,
                       CagdVType BiNrml,
                       int Normalize)
```

Nrml: The normal to return.

Tan: The tangent to return.

BiNrml: The bi-normal.

Normalize: TRUE to normalize the vectors.

Returns: void

Description: As bi-product, return the last tangent and normal of the curve evaluated last by SymbEvalCrvCurvature.
See also: SymbEvalCrvCurvPrep, SymbEvalCrvCurvature,

11.2.194 SymbEvalCrvCurvature (evalcurv.c:94)

curvature

```
CagdBType SymbEvalCrvCurvature(const CagdCrvStruct *Crv, CagdRType t,
                               CagdRType *k,
                               CagdVType Tan,
                               CagdVType BiNrml)
```

Crv: Curve to evaluate its curvature properties.

t: Location of evaluation.

k: The returned curvature at Crv(t).

Tan: The tangent to return.

BiNrml: The bi-normal to return.

Returns: TRUE if succesful, FALSE otherwise.

Description: Evaluate a given curve's curvature. Returns the curvature for the given curve location. This function must be invoked after SymbEvalCrvCurvPrep was called to initialize the proper data structures, for fast curvature evaluations at many points. SymbEvalCrvCurvPrep should be called at the end to release these data structures. As a bi-product, function SymbEvalCrvCurvTN could be invoked immediately after this function to fetch the tangent and the normal of the curve at the given location.
See also: SymbEvalCrvCurvPrep, SymbEvalCrvCurvTN,

11.2.195 SymbEvalSrfAsympDir (evalcurv.c:388)

curvature

```
int SymbEvalSrfAsympDir(const CagdSrfStruct *Srf,
                        CagdRType U,
                        CagdRType V,
                        CagdBType DirInUV,
                        CagdVType AsympDir1,
                        CagdVType AsympDir2)
```

Srf: To compute asymptotic directions for.

U, V: Location of evaluation.

DirInUV: If TRUE asymptotic direction is given in UV, otherwise in Euclidean 3-space.

AsympDir1, AsympDir2: Returned values.

Returns: Number of asymptotic directions found, zero for none.

Description: Computes the asymptotic directions of the surface in the given U, V location, if exists. If DirInUV, the returned asymptotic direction is in UV space, otherwise, in Euclidean surface tangent plane space. This function must be invoked after SymbEvalSrfCurvPrep was called to initialize the proper data structures, for fast curvature evaluations at many points. SymbEvalSrfCurvPrep should be called at the end to release these data structures. The asymptotic direction(s) is the direction for which the normal curvature vanish and hence can exist for hyperbolic regions. We solve:

```
[t  (1-t)] [ L  M ] [t  ]
           [      ] [   ]
           [ M  N ] [1-t]
```

and look for solution of t between zero and one as:

```
t = (N-M +/- sqrt(M^2-LN)) / (L-2M+N).
```

See also: SymbEvalSrfCurvature, SymbEvalSrfCurvPrep,

11.2.196 SymbEvalSrfCurvPrep (evalcurv.c:172)

curvature

```
void SymbEvalSrfCurvPrep(const CagdSrfStruct *Srf, CagdBType Init)
```

Srf: Surface to preprocess. Its attributes might be affected though.

Init: TRUE for initializing, FALSE for clearing out.

Returns: void

Description: Preprocess a given surface so we can evaluate curvature properties from it efficiently, at every point. See SymbEvalCurvature for actual curvature at surface point evaluations.
See also: SymbEvalCurvature,

11.2.197 SymbEvalSrfCurvature (evalcurv.c:239)

curvature

```
CagdBType SymbEvalSrfCurvature(const CagdSrfStruct *Srf,
                               CagdRType U,
                               CagdRType V,
                               CagdBType DirInUV,
                               CagdRType *K1,
                               CagdRType *K2,
                               CagdVType D1,
                               CagdVType D2)
```

Srf: Surface to evaluate its curvature properties.

U, V: Location of evaluation.

DirInUV: If TRUE principal directions are given in UV, otherwise in Euclidean 3-space.

K1, K2: Principal curvatures.

D1, D2: Principal directions.

Returns: TRUE if successful, FALSE otherwise.

Description: Evaluate a given surface's curvature properties. Returns the principal curvatures and directions for the given surface location. This function must be invoked after SymbEvalSrfCurvPrep was called to initialize the proper data structures, for fast curvature evaluations at many points. SymbEvalSrfCurvPrep should be called at the end to release these data structures.
See also: SymbEvalSrfCurvPrep,

11.2.198 SymbExtremumCntPtValsMalloc (symb_crv.c:1656)

extremum values

symbolic computation

```
CagdRType *SymbExtremumCntPtValsMalloc(CagdRType * const *Points,
                                       int Length,
                                       CagdBType FindMinimum)
```

Points: To scan for extremum values.

Length: Length of each vector in Points.

FindMinimum: TRUE for minimum, FALSE for maximum.

Returns: A vector holding PType point with the extremum values of each axis independently.

Description: Given a control polygon/mesh, computes the extremum values of them all.

11.2.199 SymbExtremumCntPtValsToData (symb_crv.c:1609)

extremum values

symbolic computation

```
CagdRType *SymbExtremumCntPtValsToData(CagdRType * const *Points,
                                       int Length,
                                       CagdBType FindMinimum,
                                       CagdRType *Extremum)
```

Points: To scan for extremum values.

Length: Length of each vector in Points.

FindMinimum: TRUE for minimum, FALSE for maximum.

Extremum: A vector holding PType point with the extremum values of each axis independently.

Returns: A vector holding PType point with the extremum values of each axis independently.

Description: Given a control polygon/mesh, computes the extremum values of them all.

11.2.200 SymbGet2CrvsInterDAreaDCtlPts (symb_cci.c:343)

```
int SymbGet2CrvsInterDAreaDCtlPts(CagdCrvStruct *Crv1,
                                  CagdCrvStruct *Crv2,
                                  CagdRType Eps,
                                  CagdRType **InterDomains,
                                  CagdRType **dAreadPts)
```

Crv1: First planar curve.

Crv2: Second planar curve.

Eps: Tolerance of computation.

InterDomains: Calculated domain as a list of the in the following 4-tuple structure, (u1_enter, u1_leave, u2_enter, u2_leave) sorted by u1 values.

dAreadPts: Will be updated to contain the change rate of the intersection areas relative to changes in each control points of both curves. The order in this array is as follows, (Crv1_ctpnt1_x, Crv1_ctpnt1_y, Crv1_ctpnt2_x, Crv1_ctpnt2_y, ... Crv2_ctpnt1_x, Crv2_ctpnt1_y, ... Crv2_ctpnt2_x, Crv2_ctpnt2_y, ...).

Returns: Number of intersection domains of the two curves (the length of InterDomains is 4-times this return value).

Description: Calculates the intersection domains of two curves, and the change rate of the total intersection area relative to control points change. The curves must be planar and have the same orientation. The intersections must be complete - that is, there are even number of intersections.
See also: SymbGet2CrvsIntersectionAreas,

11.2.201 SymbGet2CrvsIntersectionAreas (symb_cci.c:246)

```
CagdRType SymbGet2CrvsIntersectionAreas(const CagdCrvStruct *Crv1,
                                        const CagdCrvStruct *Crv2,
                                        CagdRType Eps)
```

Crv1: First planar curve.

Crv2: Second planar curve.

Eps: Tolerance of computation.

Returns: Total area of intersection domains of the two curves.

Description: Calculates the total area of the intersection domains of two planar curves. The curves must have the same orientation.
See also: SymbCrvEnclosedAreaEval, SymbGet2CrvsInterDAreaDCtlPts,

11.2.202 SymbGet2CrvsIntersectionRegions (symb_cci.c:169)

```
CagdCrvStruct *SymbGet2CrvsIntersectionRegions(const CagdCrvStruct *Crv1,
                                               const CagdCrvStruct *Crv2,
                                               CagdRType Eps)
```

Crv1, Crv2: The two curves to intersect and compute intersecting region(s).

Eps: Tolerance of computation.

Returns: List of closed intersecting region(s) or NULL if none.

Description: Computes the intersection region(s) of given two curves if any. curves. The curves must have the same orientation.
See also: CagdCrvCrvInter,

11.2.203 SymbGetCrvSubRegionAlphaMatrix (symb_cci.c:417)

```
CagdRType *SymbGetCrvSubRegionAlphaMatrix(const CagdCrvStruct *Crv,
                                          CagdRType t1,
                                          CagdRType t2,
                                          int *Dim)
```

Crv: Curve to compute its region extraction Alpha matrix. Assumed a non periodic curve.

t1, t2: Of extracted domain.

Dim: The two dimensions of the returned matrix will be placed here.

Returns: Linearized 2D matrix, row by row, of size (CrvLen, RgnLen), where CrvLen is the number of control points in curve Crv, and RgnLen is the size of the extracted curve region. Allocated dynamically.

Description: Computes the Alpha matrix that relates the control points of the curve region (t1, t2) to the control points of the input curve Crv of a larger domain. Compute the Alpha matrix by adding Order-1 knots at t1 and t2.
See also: BspKnotEvalAlphaCoef,

11.2.204 SymbHausDistBySamplesCrvCrv (distance.c:1366)

curve curve distance

```
CagdRType SymbHausDistBySamplesCrvCrv(const CagdCrvStruct *Crv1,
                                      const CagdCrvStruct *Crv2,
                                      int Samples,
                                      int HDistSide)
```

Crv1, Crv2: The two curves, Crv1(u) and Crv2(v), to estimate their Hausdorff distance.

Samples: Number of samples to take, uniformly in parametric space.

HDistSide: 1 for HD from V1 to V2, 2 for HD from V2 to V1, 3 for a symmetric HD.

Returns: An upper bound on the Hausdorff distance.

Description: Given two curves, compute an Hausdorff distance bound by sampling two dense sets of sampled points and estimating the Hausdorff distance between the two curves by the sampled points.
See also: SymbHausDistOfSamplefPts,

11.2.205 SymbHausDistBySamplesCrvSrf (distance.c:1406)

curve surface distance

```
CagdRType SymbHausDistBySamplesCrvSrf(const CagdCrvStruct *Crv1,
                                      const CagdSrfStruct *Srf2,
                                      int Samples,
                                      int HDistSide)
```

Crv1, Srf2: The two curves and surfaces, Crv(t) and Srf2(u, v), to estimate their Hausdorff distance.

Samples: Number of samples to take, uniformly in parametric space, as (Samples x Samples) samples.

HDistSide: 1 for HD from V1 to V2, 2 for HD from V2 to V1, 3 for a symmetric HD.

Returns: An upper bound on the Hausdorff distance.

Description: Given a curve and a surface, compute an Hausdorff distance bound by sampling two dense sets of sampled points and estimating the Hausdorff distance between the two shapes by the sampled points.
See also: SymbHausDistOfSamplefPts, SymbHausDistBySamplesSrfSrf,

11.2.206 SymbHausDistBySamplesSrfSrf (distance.c:1447)

surface surface distance

```
CagdRType SymbHausDistBySamplesSrfSrf(const CagdSrfStruct *Srf1,
                                      const CagdSrfStruct *Srf2,
                                      int Samples,
                                      int HDistSide)
```

Srf1, Srf2: The two surfaces, Srf1(u) and Srf2(v), to estimate their Hausdorff distance.

Samples: Number of samples to take, uniformly in parametric space, as (Samples x Samples) samples.

HDistSide: 1 for HD from V1 to V2, 2 for HD from V2 to V1, 3 for a symmetric HD.

Returns: An upper bound on the Hausdorff distance.

Description: Given two surfaces, compute an Hausdorff distance bound by sampling two dense sets of sampled points and estimating the Hausdorff distance between the two surfaces by the sampled points.
See also: SymbHausDistOfSamplefPts,

11.2.207 SymbHausDistOfSamplefPts (distance.c:1294)

```
CagdRType SymbHausDistOfSamplefPts(CagdPType * const V1,
                                   CagdPType * const V2,
                                   int V1Size,
                                   int V2Size,
                                   int HDistSide)
```

V1, V2: The two vectors to consider.

V1Size, V2Size: Lengths of vectors V1 and V2.

HDistSide: 1 for HD from V1 to V2, 2 for HD from V2 to V1, 3 for a symmetric HD.

Returns: Computed HD.

Description: Compute the Hausdorff Distance (HD) between two vectors of points.
See also: ,

11.2.208 SymbHighlightLnFree (rflct_ln.c:423)

```
void SymbHighlightLnFree(CagdSrfStruct *Srf)
```

Srf: Surface to free its internal data sets saved as attributes.

Returns: void

Description: Free the internal data sets, if any of the given surface, toward the computation of the highlight lines. as created by SymbHighlightLnPrepSrf.
See also: SymbHighlightLnPrepSrf, SymbHighlightLnGen,

11.2.209 SymbHighlightLnGen (rflct_ln.c:380)

```
CagdSrfStruct *SymbHighlightLnGen(CagdSrfStruct *Srf,
                                  const CagdPType LnPt,
                                  const CagdVType LnDir)
```

Srf: Surface to preprocess.

LnPt: Point on a highlight line.

LnDir: Direction of highlight line.

Returns: A scalar surface whose zero set is the highlight line sought on Srf.

Description: Compute the highlight line through LnPt on the given surface. The surface is assumed to have been preprocessed in SymbHighlightLnPrepSrf for the requested preprocessed named attribute, or otherwise SymbHighlightLnPrepSrf will be invoked on the fly.
See also: SymbHighlightLnPrepSrf, SymbHighlightLnFree,

11.2.210 SymbHighlightLnPrepSrf (rflct_ln.c:342)

```
void SymbHighlightLnPrepSrf(CagdSrfStruct *Srf, const CagdVType LnDir)
```

Srf: Surface to preprocess.

LnDir: Direction of highlight line.

Returns: void

Description: Precompute the necessary data set for as efficient as possible highlight lines' extractions. Data set is kept as an attribute on the surface. Note that only the direction of the highlight line is employed at this time, and exact location will be required by SymbHighlightLnGen only.
See also: SymbHighlightLnGen, SymbHighlightLnFree,

11.2.211 SymbHugeCrv2Polyline (symbpoly.c:337)

```
CagdPtStruct *SymbHugeCrv2Polyline(const CagdCrvStruct *Crv,
                                   int Samples,
                                   CagdBType AddFirstPt,
                                   CagdBType AddLastPt,
                                   CagdBType AddParamVals)
```

Crv: To select Samples points out of its control polygon.

Samples: Number of samples to grab.

AddFirstPt: If TRUE, first point on curve is also added.

AddLastPt: If TRUE, last point on curve is also added.

AddParamVals: TRUE to add parameter values, as attributes.

Returns: Taken samples.

Description: If the given curve is huge, simply select Samples points out if its control polygon.
See also:

11.2.212 SymbInsertNewParam2 (symbzero.c:715)

```
CagdPtStruct *SymbInsertNewParam2(CagdPtStruct *PtList, CagdRType t)
```

PtList: List to insert a new value t into.

t: New value to insert to PtList list.

Returns: Updated list, in place.

Description: Insert a single t value into given PtList, in place, provided no equal t value exists already in the list. List is ordered incrementally.
See also:

11.2.213 SymbIsCircularCrv (rvrs_eng.c:143)

```
CagdBType SymbIsCircularCrv(const CagdCrvStruct *Crv,
                            CagdPType Center,
                            CagdRType *Radius,
                            CagdRType Eps)
```

Crv: Curve to attempt and recognize as circular.

Center: If circular, the center of the circle, zero vector otherwise.

Radius: If circular, the radius of the circle, zero otherwise.

Eps: Tolarence of "same" value.

Returns: TRUE if circular curve, FALSE otherwise.

Description: Attempts to recongnize if the given curve Crv is indeed circular. If the curve is found to be circular, its center and radius are returned. Crv is tested for circularity in the XY plane. A curve is circular if its evolute curve is constant.
See also: SymbIsSphericalSrf,

11.2.214 SymbIsConstCrv (rvrs_eng.c:45)

```
CagdBType SymbIsConstCrv(const CagdCrvStruct *CCrv,
                         CagdCtlPtStruct *ConstVal,
                         CagdRType Eps)
```

CCrv: Curve to attempt and recognize as a constant curve.

ConstVal: Resulting constant value if indeed a constant curve. This value is always Euclidean, even for projective curves.

Eps: Tolarence of "same" value.

Returns: TRUE if a constant curve, FALSE otherwise.

Description: Attempts to recognize if the given curve Crv is a constant curve. If TRUE, ConstVal is reference to a static location holding the constant value.
See also: SymbIsConstSrf,

11.2.215 SymbIsConstSrf (rvrs_eng.c:273)

```
CagdBType SymbIsConstSrf(const CagdSrfStruct *CSrf,
                         CagdCtlPtStruct *ConstVal,
                         CagdRType Eps)
```

CSrf: Surface to attempt and recognize as a constant surface.

ConstVal: Resulting constant value if indeed a constant surface. This value is always Euclidean, even for projective surfaces.

Eps: Tolarence of "same" value.

Returns: TRUE if a constant surface, FALSE otherwise.

Description: Attempts to recognize if the given surface Srf is a constant surface.
See also: SymbIsConstCrv,

11.2.216 SymbIsDevelopSrf (rvrs_eng.c:435)

```
CagdBType SymbIsDevelopSrf(const CagdSrfStruct *Srf, CagdRType Eps)
```

Srf: Surface to attempt and recognize as a ruled surface.

Eps: Tolarence of "same" value.

Returns: TRUE if a ruled surface, FALSE otherwise.

Description: Attempts to recongnize if the given surface Srf is indeed a ruled surface. If the surface is found to be a ruled surface, it is decomposed into its two rail curves (two boundary curves essentially). A surface is a ruled surface if one of its partial derivatives is in the same direction for that parameter.
See also: SymbIsPlanarSrf, SymbIsRuledSrf, SymbIsSrfOfRevSrf, SymbIsSphericalSrf, SymbIsExtrusionSrf,

11.2.217 SymbIsExtrusionSrf (rvrs_eng.c:376)

```
int SymbIsExtrusionSrf(const CagdSrfStruct *Srf,
                       CagdCrvStruct **Crv,
                       CagdVType ExtDir,
                       CagdRType Eps)
```

Srf: Surface to attempt and recognize as an extrusion surface.

Crv: If an extrusion surface, the cross section curve, NULL otherwise.

ExtDir: The extrusion direction, if an extrusion surface, zero vector otherwise.

Eps: Tolarence of "same" value.

Returns: 1 if an extrusion surface along U, 2 if an extrusion surface along V, 0 otherwise.

Description: Attempts to recognize if the given surface Srf is indeed an extrusion surface. If the surface is found to be an extrusion, it is decomposed into a cross section curve Crv, and an extrusion direction ExtDir. A surface is an extrusion surface if one of its partial derivatives is constant.
See also: SymbIsPlanarSrf, SymbIsRuledSrf, SymbIsDevelopSrf, SymbIsSrfOfRevSrf, , SymbIsSphericalSrf,

11.2.218 SymbIsLineCrv (rvrs_eng.c:221)

```
CagdBType SymbIsLineCrv(const CagdCrvStruct *Crv,
                        CagdPType LnPos,
                        CagdVType LnDir,
                        CagdRType Eps)
```

Crv: Curve to attempt and recognize as circular.

LnPos: A point on the line, if the curve is indeed a line.

LnDir: A unit direction along the line, if the curve is a line.

Eps: Tolarence of "same" value.

Returns: TRUE if a line, FALSE otherwise.

Description: Attempts to recongnize if the given curve Crv is indeed circular. If the curve is found to be circular, its center and radius are returned. Crv is tested for circularity in the XY plane. A curve is circular if its evolute curve is constant.
See also: SymbIsPlanarSrf, SymbIsSphericalSrf, SymbIsCircularCrv,

11.2.219 SymbIsOffsetLclSelfInters (offset.c:1976)

```
CagdBType SymbIsOffsetLclSelfInters(CagdCrvStruct *Crv,
                                    CagdCrvStruct *OffCrv,
                                    CagdPtStruct **SIDmns)
```

Crv: Original curve.

OffCrv: Offset (approximation) curve. Assumed to share the same parametrization with Crv. I.e. Crv(t0) is offset to OffCrv(t0).

SIDmns: If not NULL set to domains that are in the self intersections. Each consecutive set of two points in this list defines one such domain.

Returns: TRUE if has local self intersection, FALSE otherwise.

Description: Reports if the given offset curve OffCrv to given curve Crv contains local self intersections. Solution is the zeros of <Crv', OffCrv'>.

11.2.220 SymbIsPlanarSrf (rvrs_eng.c:738)

```
CagdBType SymbIsPlanarSrf(const CagdSrfStruct *Srf,
                          IrtPlnType Plane,
                          CagdRType Eps)
```

Srf: Surface to attempt and recognize as circular.

Plane: The plane equation, if the surface is indeed planar.

Eps: Tolarence of "same" value.

Returns: TRUE if a line, FALSE otherwise.

Description: Attempts to recongnize if the given curve Crv is indeed circular. If the curve is found to be circular, its center and radius are returned. Crv is tested for circularity in the XY plane. A surface is plane if its gaussian and mean curvatures are zero.
See also: SymbIsRuledSrf, SymbIsDevelopSrf, SymbIsSrfOfRevSrf, SymbIsSphericalSrf, SymbIsExtrusionSrf, SymbIsLineCrv,

11.2.221 SymbIsRuledSrf (rvrs_eng.c:483)

```
int SymbIsRuledSrf(const CagdSrfStruct *Srf,
                   CagdCrvStruct **Crv1,
                   CagdCrvStruct **Crv2,
                   CagdRType Eps)
```

Srf: Surface to attempt and recognize as a ruled surface.

Crv1: If a ruled surface, the first curve, NULL otherwise.

Crv2: If a ruled surface, the second curve, NULL otherwise.

Eps: Tolarence of "same" value.

Returns: 1 if an extrusion surface along U, 2 if an extrusion surface long V, 0 otherwise.

Description: Attempts to recongnize if the given surface Srf is indeed a ruled surface. If the surface is found to be a ruled surface, it is decomposed into its two rail curves (two boundary curves essentially). A surface is a ruled surface if one of its partial derivatives is in the same direction for that parameter.
See also: SymbIsPlanarSrf, SymbIsDevelopSrf, SymbIsSrfOfRevSrf, SymbIsSphericalSrf, SymbIsExtrusionSrf,

11.2.222 SymbIsSphericalSrf (rvrs_eng.c:658)

```
CagdBType SymbIsSphericalSrf(const CagdSrfStruct *Srf,
                             CagdPType Center,
                             CagdRType *Radius,
                             CagdRType Eps)
```

Srf: Surface to attempt and recognize as a sphere.

Center: If a sphere, the center of the sphere, zero vector otherwise.

Radius: If a sphere, the radius of the sphere, zero otherwise.

Eps: Tolarence of "same" value.

Returns: TRUE if a sphere surface, FALSE otherwise.

Description: Attempts to recongnize if the given surface Srf is indeed a sphere. If the surface is found to be a sphere, its center and radius are returned. A surface is a sphere if its Gaussian and Mean sqaure surfaces are constant and equal. The center of the sphere is derived from the mean evolute surface.
See also: SymbIsPlanarSrf, SymbIsRuledSrf, SymbIsDevelopSrf, SymbIsSrfOfRevSrf, SymbIsExtrusionSrf, SymbIsCircularCrv,

11.2.223 SymbIsSrfOfRevSrf (rvrs_eng.c:566)

```
CagdBType SymbIsSrfOfRevSrf(const CagdSrfStruct *Srf,
                            CagdCrvStruct **CrossSec,
                            CagdPType AxisPos,
                            CagdVType AxisDir,
                            CagdRType Eps)
```

Srf: Surface to attempt and recognize as a ruled surface.

CrossSec: If a surface of revolution, the cross section curve, NULL otherwise.

AxisPos: If a surface of revolution, a point on axis of revolution, NULL otherwise.

AxisDir: If a surface of revolution, the direction of the axis of revolution, NULL otherwise.

Eps: Tolarence of "same" value.

Returns: TRUE if a surface of revolution, FALSE otherwise.

Description: Attempts to recongnize if the given surface Srf is indeed a surface of revolution. If the surface is found to be a surface of revolution, it is decomposed into its generator (cross section) curve, and the axis of revolution. A surface is a surface of revolution if the focal surface of one of the pseudo iso focal surfaces degenerates into a line.
See also: SymbIsPlanarSrf, SymbIsRuledSrf, SymbIsDevelopSrf, SymbIsSphericalSrf, SymbIsExtrusionSrf,

11.2.224 SymbIsZeroCrv (rvrs_eng.c:101)

```
CagdBType SymbIsZeroCrv(const CagdCrvStruct *Crv, CagdRType Eps)
```

Crv: Curve to attempt and recognize as a zero curve.

Eps: Tolarence of "same" value.

Returns: TRUE if a zero curve, FALSE otherwise.

Description: Attempts to recognize if the given curve Crv is an identically zero curve.
See also: SymbIsConstCrv, SymbIsZeroSrf,

11.2.225 SymbIsZeroSrf (rvrs_eng.c:329)

```
CagdBType SymbIsZeroSrf(const CagdSrfStruct *Srf, CagdRType Eps)
```

Srf: Surface to attempt and recognize as a zero surface.

Eps: Tolarence of "same" value.

Returns: TRUE if a zero surface, FALSE otherwise.

Description: Attempts to recognize if the given surface Srf is an identically zero surface.
See also: SymbIsConstSrf, SymbIsZeroCrv,

11.2.226 SymbLclDistCrvLine (distance.c:409)

curve line distance

```
CagdPtStruct *SymbLclDistCrvLine(const CagdCrvStruct *Crv,
                                 const CagdLType Line,
                                 CagdRType Epsilon,
                                 CagdBType InterPos,
                                 CagdBType ExtremPos)
```

Crv: The curve to find its nearest (farthest) point to Line.

Line: The line to find the nearest (farthest) point on Crv to it.

Epsilon: Accuracy of computation.

InterPos: Do we want the intersection locations?

ExtremPos: Do we want the extremum distance locations?

Returns: A list of parameter values of extreme distance locations.

Description: Given a curve and a line, finds the local extreme distance points on the curve to the given line. Only interior extrema are considered. Returned is a list of parameter value with local extreme distances. Let Crv be (x(t), y(t)). By substituting x(t) and y(t) into the line equation, we derive the distance function. Its zero set, possibly combined with the zero set of its derivative provide the needed extreme distances.
See also: SymbDistCrvLine, MvarDistSrfLine, SymbCrvRayInter,

11.2.227 SymbLclDistCrvPoint (distance.c:263)

curve point distance

```
CagdPtStruct *SymbLclDistCrvPoint(const CagdCrvStruct *CCrv,
                                  void *CrvPtPrepHandle,
                                  const CagdRType *Pt,
                                  CagdRType Epsilon)
```

CCrv: The curve to find its extreme distance locations to Pt.

CrvPtPrepHandle: If not NULL, holds pre-processed data to speed up the curve - points distance computations, for multiple curve - point tests (same curve for all points).

Pt: The point to find the extreme distance locations from Crv. Should be in the sapce range space as CCrv.

Epsilon: Accuracy of computation.

Returns: A list of parameter values of extreme distance locations.

Description: Given a curve and a point, find the local extremum distance points on the curve to the given point. Only interior extrema are considered. Returned is a list of parameter value with local extremum. Computes the zero set of (Crv(t) - Pt) . Crv'(t), in R^n - the space of Crv.
See also: SymbDistCrvPoint, MvarDistSrfPoint,

11.2.228 SymbMakePosCrvCtlPolyPos (curvatur.c:528)

refinement

```
CagdCrvStruct *SymbMakePosCrvCtlPolyPos(const CagdCrvStruct *OrigCrv)
```

OrigCrv: To refine until all its control points are non negative.

Returns: Refined positive curve with positive control points.

Description: Given a scalar curve that is positive, refine it until all its control points has positive coefficients. Always returns a Bspline curve.

11.2.229 SymbMapUVCrv2E3 (compost2.c:790)

```
CagdCrvStruct *SymbMapUVCrv2E3(const CagdCrvStruct *Crv,
                               const CagdSrfStruct *Srf,
                               CagdBType Compose)
```

Crv: In UV space of Srf to map to E3.

Srf: To map Crv over it.

Compose: TRUE to compose Srf(Crv) or FALSE to evaluate at ctl pts.

Returns: E3 curve of Srf(Crv).

Description: Computes Srf(Crv), by either evaluating Srf at all control points locations of Crv (useful if piecewise linear), or via composition.
See also: SymbComposeSrfCrv,

11.2.230 SymbMeshAddSub (symb_crv.c:1202)

addition

subtraction

symbolic computation

```
void SymbMeshAddSub(CagdRType **DestPoints,
                    CagdRType * const *Points1,
                    CagdRType * const *Points2,
                    CagdPointType PType,
                    int Size,
                    CagdBType OperationAdd)
```

DestPoints: Where addition or difference result should go to.

Points1: First control polygon/mesh.

Points2: Second control polygon/mesh.

PType: Type of points we are dealing with.

Size: Length of each vector in Points1/2.

OperationAdd: TRUE of addition, FALSE for subtraction.

Returns: void

Description: Given two control polygons/meshes - add them coordinate wise. If mesh is rational, weights are assumed identical and are just copied.
See also: SymbSrfSub, SymbSrfAdd, SymbMeshAddSubTo,

11.2.231 SymbMeshAddSubTo (symb_crv.c:1266)

addition

subtraction

symbolic computation

```
void SymbMeshAddSubTo(CagdRType **DestPoints,
                      CagdRType * const *Points2,
                      CagdPointType PType,
                      int Size,
                      CagdBType OperationAdd)
```

DestPoints: Where addition or difference result should go to.

Points2: Second control polygon/mesh.

PType: Type of points we are dealing with.

Size: Length of each vector in Points1/2.

OperationAdd: TRUE of addition, FALSE for subtraction.

Returns: void

Description: Given two control polygons/meshes - add/sub the second to the first coordinate wise. If mesh is rational, weights are assumed identical and are ignored.
See also: SymbSrfSub, SymbSrfAdd, SymbMeshAddSub,

11.2.232 SymbNormal2ConesForSrf (nrmlcone.c:541)

normals

normal bound

```
CagdBType SymbNormal2ConesForSrf(const CagdSrfStruct *Srf,
                                 CagdRType ExpandingFactor,
                                 SymbNormalConeStruct *Cone1,
                                 SymbNormalConeStruct *Cone2)
```

Srf: To compute the normal 2cones for.

ExpandingFactor: Factor to expand placement of 2cones axes locations.

Cone1, Cone2: The two cones to compute or ConeAngle == M_PI if error.

Returns: TRUE if successful, FALSE otherwise.

Description: Computes a 2cones bound to the normal field of surface Srf. The 2cones bound the normal field in the common intersection space. The 2cones are computed using the regular normal cone by expanding in the direction orthogonal to the cone axis and its main principal component. The expansion is done an amount that is equal to regular cone radius times ExpandingFactor.
See also: SymbNormalConeForSrf, SymbNormalConeOverlap, SymbTangentConeForCrv,

11.2.233 SymbNormalConeForSrfAvgToData (nrmlcone.c:284)

normals

normal bound

```
const SymbNormalConeStruct *SymbNormalConeForSrfAvgToData(const CagdSrfStruct
                                                                    *Srf,
                                                          SymbNormalConeStruct
                                                                  *NormalCone)
```

Srf: To compute a normal cone for.

NormalCone: The computed normal cone, or NULL if failed.

Returns: The computed normal cone, or NULL if failed.

Description: Computes a normal cone for a given surface, by computing the normal field of the surface and deriving the angular span of this normal field by testing the angular span of all control vector in the normal field. A normal field is searched for as "_NormalSrf" attribute in Srf or computed locally of no such attribute is found.
See also: SymbNormalConeForSrfOpt, SymbNormalConeOverlap, SymbTangentConeForCrv, , SymbNormal-ConeForSrfDoOptimal, SymbNormalConeForSrf,

11.2.234 SymbNormalConeForSrfDoOptimal (nrmlcone.c:210)

```
int SymbNormalConeForSrfDoOptimal(int Optimal)
```

Optimal: New setting.

Returns: Previous setting.

Description: Sets whether to use an optimal (but slower) algorithm to compue bounding cones for normals or use a simple vector averaging that is faster.
See also: SymbNormalConeForSrfAvg, SymbNormalConeForSrfOpt,

11.2.235 SymbNormalConeForSrfMainAxisToData (nrmlcone.c:462)

normals

normal bound

```
const SymbNormalConeStruct *SymbNormalConeForSrfMainAxisToData(
                                                const CagdSrfStruct *Srf,
                                                CagdVType MainAxis,
                                                SymbNormalConeStruct *Cone)
```

Srf: To compute normal cone and main axis of normal cone for.

MainAxis: Main axis (principal component) of the normal cone's vectors distribution.

Cone: The computed normal cone, or NULL if failed.

Returns: The computed normal cone, or NULL if failed.

Description: Same as SymbNormalConeForSrf but also estimates a main axis (principal component) for the cone A normal field is searched for as "_NormalSrf" attribute in Srf or computed locally of no such attribute is found.
See also: SymbNormalConeForSrf, SymbNormalConeOverlap, SymbTangentConeForCrv,

11.2.236 SymbNormalConeForSrfOptToData (nrmlcone.c:380)

normals

normal bound

```
const SymbNormalConeStruct *SymbNormalConeForSrfOptToData(const CagdSrfStruct
                                                                    *Srf,
                                                    SymbNormalConeStruct
                                                            *NormalCone)
```

Srf: To compute a normal cone for.

NormalCone: The computed normal cone, or NULL if failed.

Returns: The computed normal cone, or NULL if failed.

Description: Computes the optimal normal cone for a given surface, using linear prog,
See also: SymbNormalConeForSrfAvg, GMMinSpanCone, SymbNormalConeForSrfDoOptimal, SymbNormalConeForSrf,

11.2.237 SymbNormalConeForSrfToData (nrmlcone.c:249)

normals

normal bound

```
const SymbNormalConeStruct *SymbNormalConeForSrfToData(const CagdSrfStruct
                                                                  *Srf,
                                                  SymbNormalConeStruct
                                                          *NormalCone)
```

Srf: To compute a normal cone for.

NormalCone: The computed normal cone, or NULL if failed.

Returns: The computed normal cone, or NULL if failed.

Description: Computes a normal cone for a given surface, by computing the normal field of the surface and deriving the angular span of this normal field by testing the angular span of all control vector in the normal field. A normal field is searched for as "_NormalSrf" attribute in Srf or computed locally of no such attribute is found.
See also: SymbNormalConeForSrfOpt, SymbNormalConeForSrfAvg, SymbTangentConeForCrv, , SymbNormalConeForSrfDoOptimal,

11.2.238 SymbNormalConeOverlap (nrmlcone.c:618)

normals

normal bound

```
CagdBType SymbNormalConeOverlap(const SymbNormalConeStruct *NormalCone1,
                                const SymbNormalConeStruct *NormalCone2)
```

NormalCone1, NormalCone2: The two normal cones to test for angular overlap.

Returns: TRUE if overlap, FALSE otherwise.

Description: Tests if the given two normal cones overlap or not.
See also: SymbNormalConeOverlap,

11.2.239 SymbNormalConvexHullConeForSrf (nrmlcone.c:649)

```
SymbNormalConeStruct *SymbNormalConvexHullConeForSrf(const CagdSrfStruct *Srf,
                                                     CagdRType ***CH,
                                                     int *NPts)
```

Srf: To compute the conical hull for.

CH: The vectors of the convex conical hull to compute. CH[i][j] is the i'th coordinate of the j'th vector.

NPts: Contain the number of vectors in CH.

Returns: A circular normal cone as a by product.

Description: Computes the convex conical hull of the normal field of surface Srf. The result is returned as an array of subset of vectors from the field.
See also: SymbNormalConvexHullConeOverlap,

11.2.240 SymbNormalConvexHullConeOverlap (nrmlcone.c:799)

```
CagdBType SymbNormalConvexHullConeOverlap(const SymbNormalConeStruct
                                                        *NormalCone1,
                                          const CagdRType **CH1,
                                          int NPts1,
                                          const SymbNormalConeStruct
                                                        *NormalCone2,
                                          const CagdRType **CH2,
                                          int NPts2)
```

NormalCone1: The normal cone of the first surface.

CH1: The convex conical hull of the first surface.

NPts1: Number of points in the convex conical of the first surface.

NormalCone2: The normal cone of the second surface.

CH2: The convexc onical hull of the second surface.

NPts2: Number of point in the convex conical of the second surface.

Returns: TRUE if overlap, FALSE otherwise.

Description: Tests if the given two convex conical hulls overlap or not.
See also: SymbNormalConvexHullConeForSrf,

11.2.241 SymbOrthoNetSrf (symb_ortho.c:51)

```
CagdSrfStruct *SymbOrthoNetSrf(const CagdCrvStruct *Crv1,
                               const CagdCrvStruct *Crv2,
                               int FrontSamples,
                               int LayerSamples)
```

Crv1: 1st Bezier curve to fit a surface with an orthogonal network of isocurves through. Curve is assumed in XY plane.

Crv2: 2nd Bezier curve to fit a surface with an orthogonal network of isocurves through. Curve is assumed in XY plane.

FrontSamples: Number of samples to take along the curves, the higher the more accurate the result will be.

LayerSamples: Number of samples to take between the curves, the higher the more accurate the result will be.

Returns: A surface between Crv1 and Crv2 with approximately orthogonal network of U/V isoparametric curves. Returns NULL if error.

Description: A function to fit a planar surface between the given two planar Bezier curves, sharing the same domain. so the U/V isoparametric curves of the surface are approximately orthogonal. In other words, the surface will be approximately conformal.
See also:

11.2.242 SymbPiecewiseRuledSrfApprox (prisa.c:128)

layout

prisa

ruled surface approximation

```
CagdSrfStruct *SymbPiecewiseRuledSrfApprox(const CagdSrfStruct *CSrf,
                                           CagdBType ConsistentDir,
                                           CagdRType Epsilon,
                                           CagdSrfDirType Dir)
```

CSrf: To approximate using piecewise ruled surfaces.

ConsistentDir: Do we want parametrization to be the same as Srf?

Epsilon: Accuracy of piecewise ruled surface approximation.

Dir: Direction of piecewise ruled surface approximation. Either U or V.

Returns: A list of ruled surfaces approximating Srf to within Epsilon in direction Dir.

Description: Constructs a piecewise ruled surface approximation to the given surface, Srf, in the given direction, Dir, that is close to the surface to within Epsilon. If ConsitentDir then ruled surface parametrization is set to be the same as original surface Srf. Otherwise, ruling dir is always CAGD_CONST_V_DIR. Surface is assumed to have point types E3 or P3 only.

See also: SymbAllPrisaSrfs, SymbPrisaRuledSrf, TrimAllPrisaSrfs,

11.2.243 SymbPlaneLineBisect (smp_skel.c:962)

bisectors

skeleton

```
CagdSrfStruct *SymbPlaneLineBisect(const CagdVType LineDir, CagdRType Size)
```

LineDir: Direction of line from origin. Must be in northern hemisphere.

Size: Portion of result as it is infinite.

Returns: Constructed bisector surface.

Description: Compute the bisector surface between the XY plane and a line emanating from the origin in direction V.

See also: SymbPtCrvBisectOnSphere, , SymbPlanePointBisect, SymbCylinPointBisect, SymbConePointBisect, , SymbSpherePointBisect, SymbTorusPointBisect, , SymbConeLineBisect, SymbSphereLineBisect, , SymbConePlaneBisect, SymbCylinPlaneBisect, SymbSpherePlaneBisect, , SymbCylinSphereBisect, SymbSphereSphereBisect, SymbConeSphereBisect, , SymbTorusSphereBisect, SymbTorusTorusBisect, SymbConeConeBisect, , SymbConeConeBisect2, SymbConeCylinBisect, SymbCylinCylinBisect,

11.2.244 SymbPlanePointBisect (smp_skel.c:700)

bisectors

skeleton

```
CagdSrfStruct *SymbPlanePointBisect(const CagdPType Pt, CagdRType Size)
```

Pt: Direction of line from origin.

Size: Portion of result as it is infinite.

Returns: Constructed bisector surface.

Description: Compute the bisector surface between the XY plane and a point.

See also: SymbPtCrvBisectOnSphere, , SymbCylinPointBisect, SymbConePointBisect, SymbSpherePointBisect, , SymbTorusPointBisect, , SymbConePlaneBisect, SymbCylinPlaneBisect, SymbSpherePlaneBisect, , SymbConeLineBisect, SymbSphereLineBisect, SymbCylinSphereBisect, SymbSphereSphereBisect, SymbConeSphereBisect, , SymbTorusSphereBisect, SymbConeConeBisect, SymbConeConeBisect2, , SymbConeCylinBisect, SymbCylinCylinBisect, SymbTorusTorusBisect,

11.2.245 SymbPrisaGetCrossSections (prisa.c:503)

```
CagdCrvStruct *SymbPrisaGetCrossSections(const CagdSrfStruct *RSrfs,
                                         CagdSrfDirType Dir,
                                         const CagdVType Space)
```

RSrfs: A list of ruled surfaces to extract cross sections from.

Dir: Ruling direction of ruled/developable surface approximation. Typically CAGD_CONST_U_DIR.

Space: Increment on Y on the offset vector, after this cross section was placed in the XY plane.

Returns: A list of cross sections. The cross sections will be in the XY plane the cross section is indeed planar or approximately in the XY plane otherwise.

Description: Given a list of n ruled surface in 3-space, extract their cross sections and return a list of n+1 cross sections. The given ruled surfaces are assumed to be a layout decomposition of a freeform surface using the function SymbPiecewiseRuledSrfApprox.
See also: SymbPrisaRuledSrf, SymbPiecewiseRuledSrfApprox, SymbAllPrisaSrfs, , SymbPrisaGetOneCrossSection,

11.2.246 SymbPrisaGetOneCrossSection (prisa.c:585)

```
CagdCrvStruct *SymbPrisaGetOneCrossSection(const CagdSrfStruct *RSrf,
                                           CagdSrfDirType Dir,
                                           CagdBType Starting,
                                           CagdBType Ending)
```

RSrf: A ruled surface to extract cross section(s) from.

Dir: Ruling direction of ruled/developable surface approximation. Typically CAGD_CONST_U_DIR.

Starting: If TRUE, extracts the first cross section.

Ending: If TRUE, extracts the last cross section.

Returns: A list of one or two cross sections. The cross sections will be in the XY plane the cross section is indeed planar or approximately in the XY plane otherwise.

Description: Given a ruled surface in 3-space, extract its starting/ending cross sections and return a list of one or two cross sections. The given ruled surface is assumed to be a layout decomposition of a freeform surface using the function SymbPiecewiseRuledSrfApprox.
See also: SymbPrisaRuledSrf, SymbPiecewiseRuledSrfApprox, SymbAllPrisaSrfs, , SymbPrisaGetCrossSections,

11.2.247 SymbPrisaRuledSrf (prisa.c:359)

layout

prisa

```
CagdSrfStruct *SymbPrisaRuledSrf(const CagdSrfStruct *Srf,
                                 int SamplesPerCurve,
                                 CagdRType Space,
                                 CagdVType Offset)
```

Srf: A ruled surface to layout flat on the XY plane.

SamplesPerCurve: During the approximation of a ruled surface as a developable surface.

Space: Increment on Y on the offset vector, after this surface was placed in the XY plane.

Offset: A vector in the XY plane to denote the amount of translation for the flatten surface in the XY plane.

Returns: A planar surface in the XY plane approximating the falttening process of Srf.

Description: Layout a single ruled surface, by approximating it as a set of polygons. The given ruled surface might be non-developable, in which case approximation will be of a surface with no twist. The ruled surface is assumed to be constructed using CagdRuledSrf and that the ruled direction is consistent and is always CAGD_CONST_V_DIR.
See also: SymbPiecewiseRuledSrfApprox, SymbAllPrisaSrfs, TrimAllPrisaSrfs,

11.2.248 SymbPrmtSclrCrvTo2D (symb_crv.c:1564)

promotion
conversion
symbolic computation

```
CagdCrvStruct *SymbPrmtSclrCrvTo2D(const CagdCrvStruct *Crv,
                                   CagdRType Min,
                                   CagdRType Max)
```

Crv: Scalar curve to promose to a two dimensional one.

Min: Minimum of new monotone X axis.

Max: Maximum of new monotone X axis.

Returns: A two dimensional curve.

Description: Promote a scalar curve to two dimensions by moving the scalar axis to be the Y axis and adding monotone X axis.
See also: SymbPrmtSclrSrfTo3D,

11.2.249 SymbPrmtSclrSrfTo3D (symb_srf.c:1519)

promotion
conversion
symbolic computation

```
CagdSrfStruct *SymbPrmtSclrSrfTo3D(const CagdSrfStruct *Srf,
                                   CagdRType UMin,
                                   CagdRType UMax,
                                   CagdRType VMin,
                                   CagdRType VMax)
```

Srf: Surface to promote from one to three dimensions.

UMin: Minimum of new monotone X axis.

UMax: Maximum of new monotone X axis.

VMin: Minimum of new monotone Y axis.

VMax: Maximum of new monotone Y axis.

Returns: A three dimensional surface.

Description: Promote a scalar surface to three dimensions by moving the scalar axis to be the Z axis and adding monotone X and Y axes.
See also: SymbPrmtSclrCrvTo2D,

11.2.250 SymbPtCrvBisectOnSphere (smp_skel.c:69)

bisectors
skeleton

```
CagdCrvStruct *SymbPtCrvBisectOnSphere(const CagdPType Pt,
                                       const CagdCrvStruct *CCrv)
```

Pt: A point on the unit sphere, on the northern hemisphere.

CCrv: A curve on the unit sphere, on the northern hemisphere.

Returns: The bisector curve on Z = 1 plane to be centrally projected onto the unit sphere as the spherical bisector.

Description: Computes the bisector curve on a sphere of a point and a curve, both assumed to be on the unit sphere The following assumption are made and must be met for proper answer:

1. Both the curve and the point are indeed on the unit sphere.
2. Both the curve and the point are on the northern hemisphere. That is the Z coefficients of all points on both Pt and Crv are positive. The end result is NOT a curve on the sphere but rather a rational curve in the Z = 1 plane whose central projection onto the sphere (i.e. normalization) would yield the proper bisector on the unit sphere. Let P be the bisector point. Then, the following must be satisified:

```
< P, Pt > = < P, C(t) >   (Equality of angular distance)
< P - C(t), C'(t) > = 0   (orthogonality of distance measure)
< P, (0, 0, 1) > = 1      (containment in the Z = 1 plane, or Pz = 1)
```

Note we have only two unknowns (Px, Py) as Pz equals 1.

See also: SymbCrvCrvBisectOnSphere,

11.2.251 SymbPtCrvBisectOnSphere2 (smp_skel.c:188)

bisectors

skeleton

```
CagdCrvStruct *SymbPtCrvBisectOnSphere2(const CagdPType Pt,
                                        const CagdCrvStruct *CCrv,
                                        CagdRType SubdivTol)
```

Pt: A point on the unit sphere, on the northern hemisphere.

CCrv: A curve on the unit sphere, on the northern hemisphere.

SubdivTol: Accuracy of piecewise linear approximation.

Returns: A piecewise linear curve approximating the bisector of Crv1 and Crv2 on the sphere.

Description: Computes the bisector on a sphere between a point and a curve on the sphere. The returned result is a piecewise linear curve on the sphere.
See also: SymbPtCrvBisectOnSphere, SymbCrvCrvBisectOnSphere,

11.2.252 SymbRflctCircFree (rflct_ln.c:311)

```
void SymbRflctCircFree(CagdSrfStruct *Srf)
```

Srf: Surface to free its internal data sets saved as attributes.

Returns: void

Description: Free the internal data sets, if any of the given surface, toward the computation of the reflection circles. as created by SymbRflctCircPrepSrf.
See also: SymbRflctCircPrepSrf, SymbRflctCircGen,

11.2.253 SymbRflctCircGen (rflct_ln.c:272)

```
CagdSrfStruct *SymbRflctCircGen(CagdSrfStruct *Srf,
                                const CagdVType ViewDir,
                                const CagdPType SprCntr,
                                CagdRType ConeAngle)
```

Srf: Surface to preprocess.

ViewDir: Direction of view.

SprCntr: Center of sphere that reflection lines should be tangent to.

ConeAngle: Opening angle assumed for a cone holding the sphere with the apex of the cone at S(u, v), in degrees.

Returns: A scalar surface whose zero set is the reflection circles sought on Srf.

Description: Compute the reflection circles through a sphere centered at SprCntr off the given surface. The surface is assumed to have been preprocessed in SymbRflctCircPrepSrf for the requested preprocessed named attribute, or otherwise SymbRflctCircPrepSrf will be invoked on the fly.
See also: SymbRflctCircPrepSrf, SymbRflctCircFree,

11.2.254 SymbRflctCircPrepSrf (rflct_ln.c:208)

```
void SymbRflctCircPrepSrf(CagdSrfStruct *Srf,
                          const CagdVType ViewDir,
                          const CagdPType SprCntr)
```

Srf: Surface to preprocess.

ViewDir: Direction of view.

SprCntr: Center of sphere that reflection lines should be tangent to.

Returns: void

Description: Precompute the necessary data set for as efficient as possible reflection circles' extractions. Data set is kept as an attribute on the surface.
See also: SymbRflctCircGen, SymbRflctCircFree,

11.2.255 SymbRflctLnFree (rflct_ln.c:171)

```
void SymbRflctLnFree(CagdSrfStruct *Srf)
```

Srf: Surface to free its internal data sets saved as attributes.

Returns: void

Description: Free the internal data sets, if any of the given surface, toward the computation of the reflection lines. as created by SymbRflctLnPrepSrf. m
See also: SymbRflctLnPrepSrf, SymbRflctLnGen,

11.2.256 SymbRflctLnGen (rflct_ln.c:130)

```
CagdSrfStruct *SymbRflctLnGen(CagdSrfStruct *Srf,
                              const CagdVType ViewDir,
                              const CagdPType LnPt,
                              const CagdVType LnDir)
```

Srf: Surface to preprocess.

ViewDir: Direction of view.

LnPt: Point on a reflection line.

LnDir: Direction of reflection line.

Returns: A scalar surface whose zero set is the reflection line sought on Srf.

Description: Compute the reflection line through LnPt off the given surface. The surface is assumed to have been preprocessed in SymbRflctLnPrepSrf for the requested preprocessed named attribute, or otherwise SymbRflctLnPrepSrf will be invoked on the fly.
See also: SymbRflctLnPrepSrf, SymbRflctLnFree,

11.2.257 SymbRflctLnPrepSrf (rflct_ln.c:87)

```
void SymbRflctLnPrepSrf(CagdSrfStruct *Srf,
                        const CagdVType ViewDir,
                        const CagdVType LnDir)
```

Srf: Surface to preprocess.

ViewDir: Direction of view.

LnDir: Direction of reflection line.

Returns: void

Description: Precompute the necessary data set for as efficient as possible reflection lines' extractions. Data set is kept as an attribute on the surface. Note that only the direction of the reflection line is employed at this time, and exact location will be required by SymbRflctLnGen only.
See also: SymbRflctLnGen, SymbRflctLnFree,

11.2.258 SymbRingRingIntersection (rrinter.c:876)

```
CagdCrvStruct *SymbRingRingIntersection(CagdCrvStruct *C1,
                                        CagdCrvStruct *r1,
                                        CagdCrvStruct *C2,
                                        CagdCrvStruct *r2,
                                        CagdRType SubdivTol,
                                        CagdCrvStruct **PCrvs1,
                                        CagdCrvStruct **PCrvs2)
```

C1, r1: The two curves prescribing the first ring surface. Must be integral curves.

C2, r2: The two curves prescribing the second ring surface. Must be integral curves.

SubdivTol: Accuracy of zero set computation as part of the solution. Value of 0.01 is a good start.

PCrvs1, PCrvs2: The parametric domains of the intersection curves, in the two surfaces.

Returns: Intersection curves in Euclidean space.

Description: Computes the intersection curve of two ring surfaces:

```
S1(u, t) = C1(u) + Circ1(t) r1(u)
S2(v, s) = C2(v) + Circ2(s) r2(v)
```

where Circ1 and Circ2 are oriented to be in the normal plane of C1/C2. The intersection is derived from the zero set of the function that is computed by SymbRingRingZeroSetFunc.

See also: SymbRingRingZeroSetFunc,

11.2.259 SymbRingRingZeroSetFunc (rrinter.c:1037)

```
CagdSrfStruct *SymbRingRingZeroSetFunc(CagdCrvStruct *C1,
                                       CagdCrvStruct *r1,
                                       CagdCrvStruct *C2,
                                       CagdCrvStruct *r2)
```

C1, r1: The two curves prescribing the first ring surface. Must be integral curves.

C2, r2: The two curves prescribing the second ring surface. Must be integral curves.

Returns: F(u, v).

Description: Computes the intersection curve of two ring surfaces:

```
S1(u, t) = C1(u) + Circ1(t) r1(u)
S2(v, s) = C2(v) + Circ2(s) r2(v)
```

where Circ1 and Circ2 are oriented to be in the normal plane of C1/C2.
Let n1(u) and n2(v) be the normals of the normal plane of C1/C2, n1(u) = C1'(u), n2(v) = C2'(v). Then, solve for x(u, v), y(u, v), z(u, v)

```
|     n1(u)     | | x |     | < n1(u), C1(u) >                      |
|                | |   |     |                                       |
|     n2(v)     | | y |  =  | < n2(v), C2(v) >                      |
|                | |   |     |                                       |
| C1(u) - C2(v) | | z |     | ( < C1(u), C1(u) > - < C2(v), C2(v) > |
|                | |   |     |             + r2^2(v) - r1^2(u) ) / 2 |
```

Lets P(u, v) = (x(u, v), y(u, v), z(u, v)). Find the zero set of
F(u, v): < P(u, v) - C1(u), P(u, v) - C1(u) > - r1^2(u) = 0,
or, alternatively, the zero set of,
F(u, v): < P(u, v) - C2(v), P(u, v) - C2(v) > - r2^2(v) = 0.

See also: SymbRingRingIntersection,

11.2.260 SymbRmKntBspCrvCleanKnots (bspkntrm.c:261)

```
CagdCrvStruct *SymbRmKntBspCrvCleanKnots(const CagdCrvStruct *Crv)
```

Crv: Curve to remove knot from.

Returns: The new curve after removal.

Description: Remove only knots which do not change the given curve.

See also: SymbRmKntBspCrvRemoveKnots, SymbRmKntBspCrvRemoveKnotsError,

11.2.261 SymbRmKntBspCrvRemoveKnots (bspkntrm.c:216)

```
CagdCrvStruct *SymbRmKntBspCrvRemoveKnots(const CagdCrvStruct *CCrv,
                                          CagdRType Tolerance)
```

CCrv: Curve to remove knots from.

Tolerance: Desired accuracy to be kept, in L-infinity norm.

Returns: The new curve after removal.

Description: Remove knots while Tolerance is kept.
See also: SymbRmKntBspCrvCleanKnots,

11.2.262 SymbRmKntBspSrfCleanKnots (bspkntrm.c:489)

```
CagdSrfStruct *SymbRmKntBspSrfCleanKnots(const CagdSrfStruct *Srf)
```

Srf: Surface to remove knot from.

Returns: The new surface after removal.

Description: Remove only knots which do not change the given surface.
See also: SymbRmKntBspCrvRemoveKnots, SymbRmKntBspCrvRemoveKnotsError, , SymbRmKntBspCrvCleanKnots,

11.2.263 SymbRmKntBspSrfRemoveKnots (bspkntrm.c:286)

```
CagdSrfStruct *SymbRmKntBspSrfRemoveKnots(const CagdSrfStruct *CSrf,
                                          CagdRType Tolerance)
```

CSrf: Surface to remove knots from.

Tolerance: Desired accuracy to be kept, in L-infinity norm.

Returns: The new surface after removal.

Description: Remove knots while Tolerance is kept.
See also: SymbRmKntBspCrvCleanKnots, SymbRmKntBspCrvRemoveKnots, , ymbRmKntBspSrfRemoveKnotsDir, SymbRmKntBspSrfCleanKnots,

11.2.264 SymbRmKntBspSrfRemoveKnotsDir (bspkntrm.c:319)

```
CagdSrfStruct *SymbRmKntBspSrfRemoveKnotsDir(const CagdSrfStruct *CSrf,
                                             CagdSrfDirType Dir,
                                             CagdRType Tolerance)
```

CSrf: Surface to remove knots from.

Dir: Parametric direction of Srf to consider - row or column.

Tolerance: Desired accuracy to be kept, in L-infinity norm.

Returns: The new surface after removal.

Description: Remove knots while Tolerance is kept.
See also: SymbRmKntBspCrvCleanKnots, SymbRmKntBspCrvRemoveKnots,

11.2.265 SymbRuledRuledIntersection (rrinter.c:144)

```
CagdCrvStruct *SymbRuledRuledIntersection(CagdCrvStruct *C1,
                                          CagdCrvStruct *C2,
                                          CagdCrvStruct *D1,
                                          CagdCrvStruct *D2,
                                          CagdRType SubdivTol,
                                          CagdCrvStruct **PCrvs1,
                                          CagdCrvStruct **PCrvs2)
```

C1, C2: The two curves forming the first ruled surface.

D1, D2: The two curves forming the second ruled surface.

SubdivTol: Accuracy of zero set computation as part of the solution. Value of 0.01 is a good start. If SubdivTol is negative the ruled surfaces are assumed infinite (and absolute value of SubdivTol is employed). Otherwise, the ruled surface is bound between C1 and C2 and between D1 and D2 respectively.

PCrvs1, PCrvs2: The parametric domains of the intersection curves, in the two surfaces.

Returns: Intersection curves in Euclidean space.

Description: Computes the intersection curve of two ruled surfaces:

```
S1(u, t) = t C1(u) + (1-t) C2(u)
S2(v, s) = s D1(v) + (1-s) D2(s)
```

Then S1(u, t) = S2(v, s) yields,

```
C1(u) - D1(v) = s (C1(u) - C2(u)) + t (D2(v) - D1(v))
```

or solve for the zero set of the determinant of,

```
                | C1(u) - D1(u) |
Gamma(u, v) =   | C1(u) - C2(u) | = 0
                | D1(v) - D2(v) |
```

11.2.266 SymbRuledRuledZeroSetFunc (rrinter.c:80)

```
CagdSrfStruct *SymbRuledRuledZeroSetFunc(CagdCrvStruct *C1,
                                         CagdCrvStruct *C2,
                                         CagdCrvStruct *D1,
                                         CagdCrvStruct *D2)
```

C1, C2: The two curves forming the first ruled surface.

D1, D2: The two curves forming the second ruled surface.

Returns: Gamma(u, v).

Description: Computes the intersection curve of two ruled surfaces:

```
S1(u, t) = t C1(u) + (1-t) C2(u)
S2(v, s) = s D1(v) + (1-s) D2(s)
```

Then S1(u, t) = S2(v, s) yields,

```
C1(u) - D1(v) = s (C1(u) - C2(u)) + t (D2(v) - D1(v))
```

or solve for the zero set of the determinant of,

```
              | C1(u) - D1(u) |
Gamma(u, v) = | C1(u) - C2(u) | = 0
              | D1(v) - D2(v) |
```

11.2.267 SymbRuledSelfIntersection (rrinter.c:606)

```
CagdCrvStruct *SymbRuledSelfIntersection(CagdCrvStruct *C1,
                                         CagdCrvStruct *C2,
                                         CagdRType SubdivTol,
                                         CagdCrvStruct **PCrvs1,
                                         CagdCrvStruct **PCrvs2)
```

C1, C2: The two curves forming the ruled surface.

SubdivTol: Accuracy of zero set computation as part of the solution. Value of 0.01 is a good start. If SubdivTol is negative the ruled surfaces are assumed infinite (and absolute value of SubdivTol is employed). Otherwise, the ruled surface is bound between C1 and C2 and between D1 and D2 respectively.

PCrvs1, PCrvs2: The parametric domains of the intersection curves, in the two surfaces.

Returns: Intersection curves in Euclidean space.

Description: Computes the self intersection curve of a ruled surfaces:

11.2.268 SymbScalarCrvLowDegZeroSet (symbzero.c:425)

```
CagdPtStruct *SymbScalarCrvLowDegZeroSet(CagdCrvStruct *Crv)
```

Crv: Low degree polynomial to derive its roots analytically.

Returns: list of zeros.

Description: Computes the zeros of low degree polynomial, analytically.

11.2.269 SymbShapeBlendOnSrf (blending.c:154)

```
CagdSrfStruct *SymbShapeBlendOnSrf(CagdSrfStruct *Srf,
                                   CagdCrvStruct *UVCrv,
                                   const CagdCrvStruct *CrossSecShape,
                                   CagdRType TanScale,
                                   CagdRType Width,
                                   const CagdCrvStruct *WidthField,
                                   CagdRType Height,
                                   const CagdCrvStruct *HeightField)
```

Srf: Surface to construct the blended shape on, with C^1 continuity.

UVCrv: The curve along which to blend the formed shae, in the parametric domain of the surface. Assumed to be in Srf.

CrossSecShape: The cross section of this blended shape.

TanScale: Scale factor of derived tangent fields.

Width: Of swept shape, in parametric space units.

WidthField: If not NULL, a scaling field to modulate the width of the constructed blended surface.

Height: Of swept shape, in Euclidean space units.

HeightField: If not NULL, a scaling field to modulate the height of the constructed blended surface.

Returns: A newly form swept shape with CrossSecShape as approximated cross section, along UVCrv.

Description: Constructs a surface, C^1 tangent to given surface and has the prescribed cross section shape.
See also: SymbShapeBlendSrf,

11.2.270 SymbShapeBlendSrf (blending.c:302)

Hermite

```
CagdSrfStruct *SymbShapeBlendSrf(const CagdCrvStruct *Pos1Crv,
                                 const CagdCrvStruct *Pos2Crv,
                                 const CagdCrvStruct *CDir1Crv,
                                 const CagdCrvStruct *CDir2Crv,
                                 const CagdCrvStruct *CrossSecShape,
                                 const CagdCrvStruct *Normal)
```

Pos1Crv, Pos2Crv: Starting and end curves of surface.

CDir1Crv, CDir2Crv: Starting and end tangent fields surface.

CrossSecShape: The shape of the cross section of the blend.

Normal: A unit vector field orthogonal to Pos1Crv - Pos2Crv.

Returns: The blended surface. This surface will equal to - Let S(t) = (Pos1Crv(t) + Pos2Crv(t)) / 2 Let D(t) = (Pos2Crv(t) - Pos1Crv(t)) / 2 Then S(t, r) = H01(r) * Dir1Crv(t) + H11(r) * Dir2Crv(t) + S(t) + D(t) * CrossSecShape_x(r) Normal(t) * CrossSecShape_y(r) where H are the cubic Hermite functions for the two tangent fields.

Description: Construct a surface that interpolates Pos1Crv and Pos2Crv so that the surface is tangent to Dir1Crv and Dir2Crv there. CrossSecShape is a blending shaping curve that must satisfy the following (CrossSecShape is C(t), t in [0, 1]): C(0) = (-1, 0), C(1) = (1,0), C'(0) = (0, 0), C'(1) = (0, 0).

See also: CagdCubicHermiteSrf, SymbShapeBlendOnSrf,

11.2.271 SymbSignedCrvtrGenCrv (crvtrrec.c:137)

```
CagdCrvStruct *SymbSignedCrvtrGenCrv(const CagdCrvStruct *Crvtr,
                                     CagdRType Tol,
                                     int Order,
                                     CagdBType Periodic)
```

Crvtr: The signed curvature signature to reconstruct. Assumed to be a piecewise polynomial arc-length function.

Tol: Tolerance of approximations (arclen/subdivision) of curves.

Order: Order of output, approximated curve. At least quadratic.

Periodic: If TRUE, teh recostructed curve is periodic so shift the result to be closed.

Returns: Reconstructed planar curve, in the XY plane. This result is invariant to rotations and translations.

Description: Reconstructs a planar curve from the given arc-length curvature signature. The reconstruction is conducted as follows:

```
1. Tetha(s) = Integral(Crvtr(s)), the angular changes as function of s.
2. T(s) = Circ(Tetha(s)), the tangent field of the reconstructed curve.
3. C(s) = Integral(T(s)), the final curve.
```

See also: SymbCrvGenSignedCrvtr,

11.2.272 SymbSphereLineBisect (smp_skel.c:1103)

bisectors

skeleton

```
CagdSrfStruct *SymbSphereLineBisect(const CagdPType SprCntr,
                                    CagdRType SprRad,
                                    CagdRType Size)
```

SprCntr: Center location of the sphere.

SprRad: Radius of sphere.

Size: Portion of result as it is infinite.

Returns: Constructed bisector surface.

Description: Compute the bisector surface between a sphere and a line. The computation is reduced to that of a bisector between a point and a cylinder, that has a rational form. The line is assumed to be the Z axis.

See also: SymbPlanePointBisect, SymbCylinPointBisect, SymbConePointBisect, , SymbSpherePointBisect, SymbTorusPointBisect, , SymbConeLineBisect, SymbPlaneLineBisect, , SymbConePlaneBisect, SymbCylinPlaneBisect, SymbSpherePlaneBisect, , SymbCylinSphereBisect, SymbSphereSphereBisect, SymbConeSphereBisect, , SymbTorusSphereBisect, SymbTorusTorusBisect, SymbConeConeBisect, , SymbConeConeBisect2, SymbConeCylinBisect, SymbCylinCylinBisect,

11.2.273 SymbSpherePlaneBisect (smp_skel.c:1152)

bisectors

skeleton

```
CagdSrfStruct *SymbSpherePlaneBisect(const CagdPType SprCntr,
                                     CagdRType SprRad,
                                     CagdRType Size)
```

SprCntr: Center location of the sphere. Must be in northern hemisphere (positive Z coefficient).

SprRad: Radius of sphere.

Size: Portion of result as it is infinite.

Returns: Constructed bisector surface.

Description: Compute the bisector surface between a sphere and the XY plane. The computation is reduced to that of a bisector between a point and a plane, that has a rational form. Only the portion for which $Z > 0$ should be considered in the output.

See also: SymbPlanePointBisect, SymbCylinPointBisect, SymbConePointBisect, , SymbSpherePointBisect, SymbTorusPointBisect, , SymbConeLineBisect, SymbSphereLineBisect, SymbPlaneLineBisect, , SymbConePlaneBisect, SymbCylinPlaneBisect, , SymbCylinSphereBisect, SymbSphereSphereBisect, SymbConeSphereBisect, , SymbTorusSphereBisect, SymbTorusTorusBisect, SymbConeConeBisect, , SymbConeConeBisect2, SymbConeCylinBisect, SymbCylinCylinBisect,

11.2.274 SymbSpherePointBisect (smp_skel.c:859)

bisectors

skeleton

```
CagdSrfStruct *SymbSpherePointBisect(const CagdPType SprCntr,
                                     CagdRType SprRad,
                                     const CagdPType Pt)
```

SprCntr: Center location of the sphere.

SprRad: Radius of sphere.

Pt: Direction of line from origin.

Returns: Constructed bisector surface.

Description: Compute the bisector surface between a sphere and a point.

See also: SymbPtCrvBisectOnSphere, , SymbPlanePointBisect, SymbCylinPointBisect, SymbConePointBisect, , SymbTorusPointBisect, , SymbConePlaneBisect, SymbCylinPlaneBisect, SymbSpherePlaneBisect, , SymbConeLineBisect, SymbSphereLineBisect, SymbCylinSphereBisect, SymbSphereSphereBisect, SymbConeSphereBisect, , SymbTorusSphereBisect, SymbTorusTorusBisect, SymbConeConeBisect2, , SymbConeCylinBisect, SymbCylinCylinBisect,

11.2.275 SymbSphereSphereBisect (smp_skel.c:1364)

bisectors

skeleton

```
CagdSrfStruct *SymbSphereSphereBisect(const CagdVType SprCntr1,
                                      CagdRType SprRad1,
                                      const CagdVType SprCntr2,
                                      CagdRType SprRad2)
```

SprCntr1: Center location of the first sphere.

SprRad1: Radius of first sphere.

SprCntr2: Center location of the second sphere.

SprRad2: Radius of second sphere.

Returns: Constructed bisector surface.

Description: Compute the bisector surface between wto spheres. The computation is reduced to that of a bisector between a point and a sphere, that has a rational form.

See also: SymbPlanePointBisect, SymbCylinPointBisect, SymbConePointBisect, , SymbSpherePointBisect, SymbTorusPointBisect, , SymbConeLineBisect, SymbSphereLineBisect, SymbPlaneLineBisect, , SymbCylinPlaneBisect, SymbConePlaneBisect, SymbSpherePlaneBisect, , SymbCylinSphereBisect, SymbConeSphereBisect, SymbTorusSphereBisect, , SymbConeConeBisect, SymbConeConeBisect2, SymbConeCylinBisect, , SymbCylinCylinBisect,

11.2.276 SymbSplitCrvsAtExtremums (crv_lenv.c:93)

```
CagdCrvStruct *SymbSplitCrvsAtExtremums(const CagdCrvStruct *CCrvs,
                                        int Axis,
                                        const CagdPType Pt,
                                        CagdRType Eps)
```

CCrvs: Input list of curves to split at all extremum values the curves assumes.

Axis: Extremum to consider: 0 - Radials silhouette as viewed from Pt. 1,2 - Look for extremum (silhouette) in X,Y dir.

Pt: If radial silhouette are sought, use this as Eye location.

Eps: Tolerance of computations.

Returns: List of splitted curves.

Description: Split the given list of curves at the extremum values of each curve, if any.
See also:

11.2.277 SymbSplitRationalCrvsPoles (symbzero.c:550)

```
CagdPtStruct *SymbSplitRationalCrvsPoles(const CagdCrvStruct *Crv,
                                         CagdRType Epsilon)
```

Crv: Rational curves to extract its poles.

Epsilon: The numerical tolerance to use.

Returns: The poles, as piecewise linear approximations.

Description: Computes the poles of a rational surface, solving for the zeros of the surface's denominator.
See also: CagdPointsHasPoles, MvarRationalCrvsPoles,

11.2.278 SymbSrf2Curves (symbpoly.c:207)

curves

isoparametric curves

```
CagdCrvStruct *SymbSrf2Curves(const CagdSrfStruct *Srf, int NumOfIsocurves[2])
```

Srf: To extract isoparametric curves from.

NumOfIsocurves: In each (U or V) direction.

Returns: List of extracted isoparametric curves. These curves inherit the order and continuity of the original Srf. NULL is returned in case of an error.

Description: Routine to extract from a surface NumOfIsoline isocurve list in each param. direction. Iso parametric curves are sampled equally spaced in parametric space. NULL is returned in case of an error, otherwise list of CagdCrvStruct.
See also: BspSrf2PCurves, BzrSrf2Curves, CagdSrf2Curves,

11.2.279 SymbSrf2Polygons (symbpoly.c:72)

polygonization

surface approximation

```
IPPolygonStruct *SymbSrf2Polygons(const CagdSrfStruct *Srf,
                                  int FineNess,
                                  CagdBType ComputeNormals,
                                  CagdBType FourPerFlat,
                                  CagdBType ComputeUV)
```

Srf: To approximate into triangles.

FineNess: Control on accuracy, the higher the finer.

ComputeNormals: If TRUE, normal information is also computed.

FourPerFlat: If TRUE, four triangles are created per flat surface. If FALSE, only 2 triangles are created.

ComputeUV: If TRUE, UV values are stored and returned as well.

Returns: A list of polygons with optional normal and/or UV parametric information. NULL is returned in case of an error.

Description: Routine to convert a single surface to set of triangles approximating it. FineNess is a fineness control on result and the larger it is more triangles may result. A value of 10 is a good starting value. NULL is returned in case of an error, otherwise list of IPPolygonStruct.
See also: BzrSrf2Polygons, IritSurface2Polygons, IritTrimSrf2Polygons, , BspSrf2Polygons, TrimSrf2Polygons,

11.2.280 SymbSrf2Polylines (symbpoly.c:137)

polylines

isoparametric curves

```
CagdPolylineStruct *SymbSrf2Polylines(const CagdSrfStruct *Srf,
                                      int NumOfIsocurves[2],
                                      CagdRType TolSamples,
                                      SymbCrvApproxMethodType Method)
```

Srf: Srf to extract isoparametric curves from.

NumOfIsocurves: To extarct from Srf in each (U or V) direction.

TolSamples: Tolerance of approximation error (Method = 2) or Number of samples to compute on polyline (Method = 0, 1).

Method: 0 - TolSamples are set uniformly in parametric space, 1 - TolSamples are set optimally, considering the isocurve's curvature. 2 - TolSamples sets the maximum error allowed between the piecewise linear approximation and original curve.

Returns: List of polylines representing a piecewise linear approximation of the extracted isoparamteric curves or NULL is case of an error.

Description: Routine to convert a single surface to NumOfIsolines polylines in each parametric direction with SamplesPerCurve in each isoparametric curve. Polyline are always E3 of CagdPolylineStruct type. NULL is returned in case of an error, otherwise list of CagdPolylineStruct. Attempt is made to extract isolines along C1 discontinuities first.

See also: BspSrf2Polylines, BzrSrf2Polylines, IritSurface2Polylines, , IritTrimSrf2Polylines, TrimSrf2Polylines,

11.2.281 SymbSrfAdd (symb_srf.c:38)

addition

symbolic computation

```
CagdSrfStruct *SymbSrfAdd(const CagdSrfStruct *Srf1,
                          const CagdSrfStruct *Srf2)
```

Srf1, Srf2: Two surface to add up coordinate-wise.

Returns: The summation of Srf1 + Srf2 coordinate-wise.

Description: Given two surfaces - add them coordinate-wise. The two surfaces are promoted to same point type before the multiplication can take place. Furthermore, order and continuity are matched as well.

See also: SymbSrfSub, SymbMeshAddSub, SymbSrfMult,

11.2.282 SymbSrfCalcAsympDirsCoeffs (crvtr_bnds.c:548)

```
int SymbSrfCalcAsympDirsCoeffs(const CagdSrfStruct *SffMatrix[3],
                               IrtRType *AsympLimits,
                               SymbSrfAsympBoundsMethod *ChosenMethod)
```

SffMatrix: The second fundamental form matrix of the surface, as an array of the form - (L, M, N).

AsympLimits: Output parameter - the asymptotic directions coefficients.

ChosenMethod: Output parameter - the chosen computation method.

Returns: The number of asymptotic directions found, or -1, if failed.

Description: Finds the maximal and minimal bounds of the asymptotic directions of a surface, as coefficients of dS/du and dS/dv. The function finds the best method for computation out of the following three: [1 b] II [1 b]^2, [a 1] II [a 1]^T, and [t 1-t] II [t 1-t]^T, where II is the second fundamental form matrix. The function is chosen by a heuristic which makes sure the computation does not involve dividing by zero, and prefers a more stable division. The chosen computation method is returned as an output parameter.

See also:

11.2.283 SymbSrfCloseParallelSrfs2Shell (offset.c:883)

```
CagdSrfStruct *SymbSrfCloseParallelSrfs2Shell(const CagdSrfStruct *Srf1,
                                              const CagdSrfStruct *Srf2)
```

Srf1, Srf2: The two surfaces to fill in their gaps between their boundaries by new (ruled) surfaces.

Returns: Upto four surfaces that fill in the gaps between the boundaries of Srf1 and Srf2.

Description: Given two parallel surfaces (a surface and its offsets or two offsets of some surface, etc.) builds the (up to) four ruled surfaces that fills in the gaps on the Umin/max, Vmin/max boundaries. Note that if UMin == UMax (VMin == VMax) no new surfaces will be added.

11.2.284 SymbSrfCrossProd (symb_srf.c:584)

product

cross product

symbolic computation

```
CagdSrfStruct *SymbSrfCrossProd(const CagdSrfStruct *Srf1,
                                const CagdSrfStruct *Srf2)
```

Srf1, Srf2: Two surface to multiply in R3 and compute cross product for.

Returns: A vector surface representing the cross product of Srf1 x Srf2.

Description: Given two surfaces - computes their cross product in R3. Returned surface is a surface representing the cross product of the two given surfaces.
See also: SymbSrfDotProd, SymbSrfVecDotProd, SymbSrfScalarScale, SymbSrfMultScalar, , SymbSrfInvert, SymbSrfVecCrossProd,

11.2.285 SymbSrfCrvtrBndsCalcBnds (crvtr_bnds.c:300)

```
void SymbSrfCrvtrBndsCalcBnds(const SymbSrfCrvtrBndsInfoStructPtr Info,
                              IrtRType *K1Limits,
                              IrtRType *K2Limits)
```

Info: The SymbSrfCrvtrBndsInfoStruct.

K1Limits: Minimal and maximal K1 values (array of size 2).

K2Limits: Minimal and maximal K2 values (array of size 2).

Returns: void

Description: Calculate curvature limits for a given surface.
See also: SymbSrfCurvatureUpperBound, SymbSrfCurvatureUpperBound, , SymbSrfCrvtrBndsCalcBnds2,

11.2.286 SymbSrfCrvtrBndsCalcBnds2 (crvtr_bnds.c:410)

```
void SymbSrfCrvtrBndsCalcBnds2(const SymbSrfCrvtrBndsInfoStructPtr Info,
                               int Count,
                               IrtRType ValMin,
                               IrtRType ValMax,
                               IrtRType ValRes,
                               IrtRType *K1Limits,
                               IrtRType *K2Limits)
```

Info: The SymbSrfCrvtrBndsInfoStruct.

Count: Maximal number of recursion calls.

ValMin: Lower end of relevant curvature range.

ValMax: Upper end of relevant curvature range.

ValRes: Required gap between minimal and maximal value for K1, K2.

K1Limits: Minimal and maximal K1 values (array of size 2).

K2Limits: Minimal and maximal K2 values (array of size 2).

Returns: void

Description: Calculate curvature limits for a given surface, with given accuracy requirements. If the accuracy is not met bounds will be calculated recursively for subdivisions of the original surface. Required accuracy is defined by ValRes as the difference between the maximal and minimal curvature values. Required accuracy is only tested in the curvature range overlaps the range given by ValMin, ValMax.
See also: SymbSrfCurvatureUpperBound, SymbSrfCurvatureUpperBound, , SymbSrfCrvtrBndsCalcBnds2,

11.2.287 SymbSrfCrvtrBndsInfoClear (crvtr_bnds.c:488)

```
void SymbSrfCrvtrBndsInfoClear(SymbSrfCrvtrBndsInfoStruct *Info)
```

Info: The SymbSrfCrvtrBndsInfoStruct.

Returns: void

Description: Clears a given SymbSrfCrvtrBndsInfoStruct (frees internal data).
See also:

11.2.288 SymbSrfCrvtrBndsInfoCreate (crvtr_bnds.c:156)

```
SymbSrfCrvtrBndsInfoStruct *SymbSrfCrvtrBndsInfoCreate(const CagdSrfStruct
                                                                    *Srf)
```

Srf: The surface.

Returns: The handle for normal curvature bound functions.

Description: Creates the information needed for normal curvature limits (bounds) calculations on the given surface.
See also:

11.2.289 SymbSrfCrvtrBndsInfoFree (crvtr_bnds.c:514)

```
void SymbSrfCrvtrBndsInfoFree(SymbSrfCrvtrBndsInfoStruct *Info)
```

Info: The SymbSrfCrvtrBndsInfoStruct.

Returns: void

Description: Frees a given SymbSrfCrvtrBndsInfoStruct.
See also:

11.2.290 SymbSrfCrvtrBndsSplitInfo (crvtr_bnds.c:211)

```
void SymbSrfCrvtrBndsSplitInfo(const SymbSrfCrvtrBndsInfoStructPtr Info,
                               SymbSrfCrvtrBndsInfoStructPtr Ret,
                               CagdSrfDirType Dir)
```

Info: The SymbSrfCrvtrBndsInfoStruct.

Ret: The two resulting SymbSrfCrvtrBndsInfoStruct (array size 2).

Dir: The splitting direction.

Returns: void

Description: Splits a SymbSrfCrvtrBndsInfoStruct along the given parameter direction.
See also:

11.2.291 SymbSrfCrvtrBndsSubInfo (crvtr_bnds.c:262)

```
SymbSrfCrvtrBndsInfoStructPtr SymbSrfCrvtrBndsSubInfo(
                                    const SymbSrfCrvtrBndsInfoStructPtr Info,
                                    IrtRType UMin,
                                    IrtRType UMax,
                                    IrtRType VMin,
                                    IrtRType VMax)
```

Info: The SymbSrfCrvtrBndsInfoStruct.

UMin: Minimal U value.

UMax: Maximal U value.

VMin: Minimal V value.

VMax: Maximal V value.

Returns: The information for the subsurface.

Description: Gets SymbSrfCrvtrBndsInfoStruct for a subsurface of the original surface.
See also:

11.2.292 SymbSrfCurvatureUpperBound (curvatur.c:1439)

curvature

```
CagdSrfStruct *SymbSrfCurvatureUpperBound(const CagdSrfStruct *Srf)
```

Srf: Surface to compute curvature bound for.

Returns: A scalar field representing the curvature bound.

Description: Computes curvature upper bound as Xi = k1^2 + k2^2, where k1 and k2 are the principal curvatures. Gij are the coefficients of the first fundamental form and Lij are of the second, using non unit normal n,

$$Xi = \frac{(G11\ L22 + G22\ L11 - 2\ G12\ L12)^2 - 2\ |G|\ |L|}{|G|^2\ ||n||^2}$$

See: "Second Order Surface Analysis Using Hybrid of Symbolic and Numeric Operators", By Gershon Elber and Elaine Cohen, Transaction on graphics, Vol. 12, No. 2, pp 160-178, April 1993.
See also: SymbSrfFff, SymbSrfSff, SymbSrfDeterminant2, SymbSrfGaussCurvature, , SymbSrfMeanEvolute, SymbSrfMeanCurvatureSqr, SymbSrfMeanNumer, , SymbSrfIsoFocalSrf, SymbSrfIsoDirNormalCurvatureBound, , SymbSrfCrvtrBndsCalcBnds, SymbSrfCrvtrBndsCalcBnds2,

11.2.293 SymbSrfDeriveRational (symb_srf.c:851)

derivatives

```
CagdSrfStruct *SymbSrfDeriveRational(const CagdSrfStruct *Srf,
                                     CagdSrfDirType Dir)
```

Srf: A surface to differentiate.

Dir: Direction of differentiation.

Returns: Differentiated rational surface.

Description: Given a rational surface - computes its derivative surface in Dir, using the quotient rule for differentiation.
See also: BzrSrfDerive, BspSrfDerive, CagdSrfDerive, SymbCrvDeriveRational,

11.2.294 SymbSrfDeterminant2 (curvatur.c:971)

determinant

```
CagdSrfStruct *SymbSrfDeterminant2(const CagdSrfStruct *Srf11,
                                   const CagdSrfStruct *Srf12,
                                   const CagdSrfStruct *Srf21,
                                   const CagdSrfStruct *Srf22)
```

Srf11, Srf12, Srf21, Srf22: The four factors of the determinant.

Returns: A scalar field representing the determinant computation.

Description: Computes the expression of Srf11 * Srf22 - Srf12 * Srf21, which is a determinant of a 2 by 2 matrix.
See also: SymbSrfFff, SymbSrfSff, SymbSrfGaussCurvature, SymbSrfMeanEvolute, , SymbSrfMeanCurvatureSqr, SymbSrfIsoFocalSrf, SymbSrfCurvatureUpperBound, , SymbSrfIsoDirNormalCurvatureBound, SymbSrfDeterminant3, , SymbCrvDeterminant2,

11.2.295 SymbSrfDeterminant3 (crv_skel.c:983)

determinant

```
CagdSrfStruct *SymbSrfDeterminant3(const CagdSrfStruct *Srf11,
                                   const CagdSrfStruct *Srf12,
                                   const CagdSrfStruct *Srf13,
                                   const CagdSrfStruct *Srf21,
                                   const CagdSrfStruct *Srf22,
                                   const CagdSrfStruct *Srf23,
                                   const CagdSrfStruct *Srf31,
                                   const CagdSrfStruct *Srf32,
                                   const CagdSrfStruct *Srf33)
```

Srf11, Srf12, Srf13: The nine factors of the determinant.

Srf21, Srf22, Srf23: ”

Srf31, Srf32, Srf33: ”

Returns: A scalar field representing the determinant computation.

Description: Computes the expression of a 3 by 3 determinants.
See also: SymbSrfFff, SymbSrfSff, SymbSrfGaussCurvature, SymbSrfMeanEvolute, , SymbSrfMeanCurvatureSqr, SymbSrfIsoFocalSrf, SymbSrfCurvatureUpperBound, , SymbSrfIsoDirNormalCurvatureBound, SymbSrfDeterminant2, , SymbCrvDeterminant3,

11.2.296 SymbSrfDevelopableCrvOnSrf (dvlp_srf.c:59)

```
CagdSrfStruct *SymbSrfDevelopableCrvOnSrf(const CagdSrfStruct *Srf,
                                          const CagdCrvStruct *Crv,
                                          IrtRType Scale)
```

Srf: The surface Crv is on.

Crv: The curve in the parametric domain of Srf.

Scale: A scaling factor of the r (developing) parameter.

Returns: The computed developable sheet.

Description: Computes the developable surface of the swept tangent plane of surface, S, along a curve on the surface C. Let N be the normal field of S. Then:

```
D(r, t) = C(t) + r N(t) x N'(t).
```

See also: SymbSrfNormalSrf,

11.2.297 SymbSrfDevelopableSrfBetweenFrames (dvlp_srf.c:129)

```
CagdSrfStruct *SymbSrfDevelopableSrfBetweenFrames(const CagdVType Frame1Pos,
                                                  const CagdVType Frame1Tan,
                                                  const CagdVType Frame1Nrml,
                                                  const CagdVType Frame2Pos,
                                                  const CagdVType Frame2Tan,
                                                  const CagdVType Frame2Nrml,
                                                  CagdRType OtherScale,
                                                  CagdRType Tension)
```

Frame1Pos: Position of initial location.

Frame1Tan: Tangent of initial location.

Frame1Nrml: Normal of initial location (sets the boundary at init.).

Frame2Pos: Position of terminal location.

Frame2Tan: Tangent of terminal location.

Frame2Nrml: Normal of terminal location (sets the boundary at init.).

OtherScale: Other direction scale of constructed developable surface.

Tension: Controls the tension of the result. A positive value.

Returns: The constructed surface.

Description: Computes a developable surface between given two orientation frames. Solution is simply by building a ruled surface with the desired shape between the two frames only to extract the developable surface between the frame, using SymbSrfDevelopableCrvOnSrf.
See also: SymbSrfDevelopableCrvOnSrf, SymbSrfDevelopableSrfBetweenFrames2,

11.2.298 SymbSrfDevelopableSrfBetweenFrames2 (dvlp_srf.c:306)

```
CagdSrfStruct *SymbSrfDevelopableSrfBetweenFrames2(const CagdVType Frame1Pos,
                                                   const CagdVType Frame1Tan,
                                                   const CagdVType Frame1Nrml,
                                                   const CagdVType Frame2Pos,
                                                   const CagdVType Frame2Tan,
                                                   const CagdVType Frame2Nrml,
                                                   CagdRType OtherScale,
                                                   CagdRType Tension,
                                                   int DOFs)
```

Frame1Pos: Position of initial location.

Frame1Tan: Tangent of initial location.

Frame1Nrml: Normal of initial location (sets the boundary at init.).

Frame2Pos: Position of terminal location.

Frame2Tan: Tangent of terminal location.

Frame2Nrml: Normal of terminal location (sets the boundary at init.).

OtherScale: Other direction scale of constructed developable surface.

Tension: Controls the tension of the result. A positive value.

DOFs: Number of degrees of freedoms (control points to add to the base curve, that is Bezier (using refinement).

Returns: The constructed surface.

Description: Computes an approximated developable surface between given two orientation frames. Solution is based on building a curve with fixed Frenet frames at the end locations as prescribed and minimizing the torsion in the curve. This curve is then used to build an approximated developable sheet with a ruling along N, its normal.
See also: SymbSrfDevelopableCrvOnSrf, SymbSrfDevelopableSrfBetweenFrames,

11.2.299 SymbSrfDistCrvCrv (distance.c:650)

curve curve distance

```
CagdSrfStruct *SymbSrfDistCrvCrv(const CagdCrvStruct *Crv1,
                                 const CagdCrvStruct *Crv2,
                                 int DistType)
```

Crv1, Crv2: The two curves, Crv1(u) and Crv2(v), to form their distance function square between them as a bivariate function.

DistType: 0 for distance vector function, 1 for distance square scalar function, 2 for distance vector projected on the normal of Crv1, 3 for distance vector projected on the normal of Crv2. 4 for distance vector projected on the tangent of Crv1. 5 for distance vector projected on the tangent of Crv2. In cases 2 to 5 the vector field is not normalized.

Returns: The distance function square d2(u, v) of the distance from Crv1(u) to Crv2(v).

Description: Given two curves, creates a bivariate scalar surface representing the distance function square, between them.
See also: SymbCrvCrvInter, SymbSrfDistFindPoints, MvarCrvCrvMinimalDist,

11.2.300 SymbSrfDistFindPoints (distance.c:759)

curve curve distance

curve curve intersection

```
CagdPtStruct *SymbSrfDistFindPoints(const CagdSrfStruct *CSrf,
                                    CagdRType Epsilon,
                                    CagdBType SelfInter)
```

CSrf: A bivariate function that represent the distance square function between two curves.

Epsilon: Accuracy control.

SelfInter: Should we consider self intersection? That is, is Srf computed from a curve to itself!?

Returns: A list of parameter values of both curves, at all detected intersection locations.

Description: Given a scalar surface representing the distance function square between two curves, finds the zero set of the distance surface, if any, and returns it. The given surface is a non negative surface and zero set is its minima. The returned points will contain the two parameter values of the two curves that intersect in the detected zero set points.
See also: SymbSrfDistCrvCrv, SymvCrvCrvInter,

11.2.301 SymbSrfDotProd (symb_srf.c:461)

product

dot product

symbolic computation

```
CagdSrfStruct *SymbSrfDotProd(const CagdSrfStruct *Srf1,
                              const CagdSrfStruct *Srf2)
```

Srf1, Srf2: Two surface to multiply and compute a dot product for.

Returns: A scalar surface representing the dot product of Srf1 . Srf2.

Description: Given two surfaces - computes their dot product. Returned surface is a scalar surface representing the dot product of the two given surfaces.
See also: SymbSrfMult, SymbSrfVecDotProd, SymbSrfScalarScale, SymbSrfMultScalar, , SymbSrfInvert, SymbSrfCrossProd, SymbSrfVecCrossProd,

11.2.302 SymbSrfDual (duality.c:126)

```
CagdSrfStruct *SymbSrfDual(const CagdSrfStruct *Srf)
```

Srf: The surface to compute its dual.

Returns: The dual surface.

Description: Computes the dual of the given syrface. The dual curve is a mapping of the tangent planes of Srf (for which Srf is the envelop of) to points in the dual space. Duality is derived by computing the tangent plane "Ax + By + Cz + D = 0" to surface Srf and mapping this plane to homogeneous point (A/D, B/D, C/D).
See also: SymbCrvDual,

11.2.303 SymbSrfFff (curvatur.c:802)

first fundamental form

```
void SymbSrfFff(const CagdSrfStruct *Srf,
                CagdSrfStruct **DuSrf,
                CagdSrfStruct **DvSrf,
                CagdSrfStruct **FffG11,
                CagdSrfStruct **FffG12,
                CagdSrfStruct **FffG22)
```

Srf: Do compute the coefficients of the FFF for.

DuSrf: First derivative of Srf with respect to U goes to here.

DvSrf: First derivative of Srf with respect to V goes to here.

FffG11: FFF G11 scalar field.

FffG12: FFF G12 scalar field.

FffG22: FFF G22 scalar field.

Returns: void

Description: Computes coefficients of the first fundamental form of given surface Srf.
See also: SymbSrfSff, SymbSrfTff, SymbSrfDeterminant2, SymbSrfGaussCurvature, , SymbSrfMeanEvolute, SymbSrfMeanCurvatureSqr, SymbSrfIsoFocalSrf, , SymbSrfCurvatureUpperBound, SymbSrfIsoDirNormalCurvatureBound,

11.2.304 SymbSrfFirstMoment (moments.c:298)

```
CagdRType SymbSrfFirstMoment(const CagdSrfStruct *Srf, int Axis)
```

Srf: Surface to compute the first moment for.

Axis: 1 for X, 2 for Y, 3 for Z.

Returns: The computed moment.

Description: Computes the first moment of the given surface. The computed moment is for the (signed) volume between the surface and its projection onto the XY plane.
See also: SymbSrfFirstMomentSrf, SymbSrfFirstMomentSrf, SymbSrfVolume,

11.2.305 SymbSrfFirstMomentSrf (moments.c:231)

```
CagdSrfStruct *SymbSrfFirstMomentSrf(const CagdSrfStruct *Srf,
                                     int Axis,
                                     CagdBType Integrate)
```

Srf: Surface to compute the first moment for.

Axis: 1 for X, 2 for Y, 3 for Z.

Integrate: TRUE to also integrate the resulting surface.

Returns: The computed moment function.

Description: Computes the first moment function of the given surface. The computed moment is for the (signed) volume between the surface and its projection onto the XY plane.
See also: SymbSrfFirstMoment, SymbSrfFirstMomentSrf, SymbSrfVolume,

11.2.306 SymbSrfGaussCurvature (curvatur.c:1003)

curvature

```
CagdSrfStruct *SymbSrfGaussCurvature(const CagdSrfStruct *Srf,
                                     CagdBType NumerOnly)
```

Srf: Surface to compute Gaussian curvature for.

NumerOnly: If TRUE, only the numerator component of K is returned.

Returns: A surface representing the Gaussian curvature field.

Description: Computes the Gaussian curvature of a given surface.
See also: SymbSrfFff, SymbSrfSff, SymbSrfDeterminant2, SymbSrfMeanEvolute, , SymbSrfMeanCurvatureSqr, SymbSrfIsoFocalSrf, SymbSrfCurvatureUpperBound, , SymbSrfIsoDirNormalCurvatureBound,

11.2.307 SymbSrfInvert (symb_srf.c:284)

division

symbolic computation

reciprocal value

```
CagdSrfStruct *SymbSrfInvert(const CagdSrfStruct *Srf)
```

Srf: A scalar surface to compute a reciprocal value for.

Returns: A rational scalar surface that is equal to the reciprocal value of Srf.

Description: Given a scalar surface, returns a scalar surface representing the reciprocal values, by making it rational (if was not one) and flipping the numerator and the denominator.
See also: SymbSrfDotProd, SymbSrfVecDotProd, SymbSrfScalarScale, SymbSrfMultScalar, , SymbSrfMult, SymbSrfCrossProd,

11.2.308 SymbSrfIsoDirNormalCurvatureBound (curvatur.c:1523)

curvature

```
CagdSrfStruct *SymbSrfIsoDirNormalCurvatureBound(const CagdSrfStruct *Srf,
                                                 CagdSrfDirType Dir)
```

Srf: To compute normal curvature in an isoparametric direction Dir.

Dir: Direction to compute normal curvature. Either U or V.

Returns: A scalar field representing the normal curvature square of Srf in dirction Dir.

Description: Computes normal curvature bound in given isoparametric direction. This turns out to be (L11 . n) / G11 for u and (L22 . n) / G22 for v. Herein the square of these equations is computed symbolically and returned.
See also: SymbSrfFff, SymbSrfSff, SymbSrfDeterminant2, SymbSrfGaussCurvature, , SymbSrfMeanEvolute, SymbSrfMeanCurvatureSqr, SymbSrfMeanNumer, , SymbSrfIsoFocalSrf, SymbSrfCurvatureUpperBound, SymbSrfCrvtrBndsCalcBnds, SymbSrfCrvtrBndsCalcBnds2,

11.2.309 SymbSrfIsoFocalSrf (curvatur.c:1306)

curvature

focal surface

evolute

```
CagdSrfStruct *SymbSrfIsoFocalSrf(const CagdSrfStruct *Srf,
                                  CagdSrfDirType Dir)
```

Srf: Surface to compute iso focal surface.

Dir: Direction to compute iso focal surface. Either U or V.

Returns: A surface representing the iso focal surface.

Description: Computes a focal surface for a principal curvature in an isoparametric direction. For the u isoparametric direction,

$$F(u, v) = n(u, v) \frac{1}{k_n^u(u, v)} = n(u, v) \frac{G11}{L11}$$

Because Lii also has n(u,v) we can use the nonnormalized surface normal to compute F(u, v), which is therefore computable and representable.

See also: SymbSrfFff, SymbSrfSff, SymbSrfDeterminant2, SymbSrfGaussCurvature, , SymbSrfMeanEvolute, SymbSrfMeanCurvatureSqr, SymbSrfMeanNumer, , SymbSrfCurvatureUpperBound, SymbSrfIsoDirNormalCurvatureBound,

11.2.310 SymbSrfIsocline (orthotom.c:336)

```
IPPolygonStruct *SymbSrfIsocline(const CagdSrfStruct *Srf,
                                 const CagdVType VDir,
                                 CagdRType Theta,
                                 CagdRType SubdivTol,
                                 CagdBType Euclidean)
```

Srf: To compute its isocline edges.

VDir: View direction vector (a unit vector).

Theta: The fixed angle between the viewing direction and the surface normal, in degrees. An angle of 90 degrees yields the silhouettes.

SubdivTol: Accuracy of computation.

Euclidean: If TRUE, returns the isoclines in Euclidean space. Otherwise, the isocline edges are returned in the Parametric domain.

Returns: The isoclines as piecewise linear edges.

Description: Computes the isocline edges of the given surfaces, orthographically seen from the given view direction VDir, at an inclination angle of Theta degrees. The isocline is a curve with a fixed angle between the surface normal and the viewing direction. An angle of 90 degrees yields the silhouettes. Computed as the zero set of:

$$(\langle N, V \rangle)^2 - (\cos(\Theta))^2 \langle N, N \rangle$$

See also: SymbSrfOrthotomic, SymbSrfSilhouette, UserMoldReliefAngle2Srf,

11.2.311 SymbSrfJacobianImprove (prm_dmn.c:679)

```
CagdRType SymbSrfJacobianImprove(CagdSrfStruct *Srf,
                                 CagdRType StepSize,
                                 int MaxIter)
```

Srf: To try and improve its parameterizations, in place. Can have negative Jacobian to begin with (that hopefully will be corrected here). Only XY coordinates are considered, in the XY plane.

StepSize: Marching amount along the gradient. Set to a non positive value to automatically estimate the step.

MaxIter: The number of iterations to apply.

Returns: The final ratio between the minimal and maximal jacobian. A negative ratio clearly means the surface self intersects

Description: Numerically iterate and improve, in place, the ratio between the minimal and maximal Jacobians by shift around interior control points.

See also: Symb2DSrfJacobian,

11.2.312 SymbSrfMeanCurvatureSqr (curvatur.c:1261)

curvature

```
CagdSrfStruct *SymbSrfMeanCurvatureSqr(const CagdSrfStruct *Srf)
```

Srf: Surface to compute Mean curvature square for.

Returns: A surface representing the Mean curvature square field.

Description: Computes the Mean curvature square of a given surface H^2 = ((k1 + k2) / 2)^2.
See also: SymbSrfFff, SymbSrfSff, SymbSrfDeterminant2, SymbSrfGaussCurvature, , SymbSrfMeanEvolute, SymbSrfMeanNumer, SymbSrfIsoFocalSrf, , SymbSrfCurvatureUpperBound, SymbSrfIsoDirNormalCurvatureBound,

11.2.313 SymbSrfMeanEvolute (curvatur.c:1148)

curvature

```
CagdSrfStruct *SymbSrfMeanEvolute(const CagdSrfStruct *Srf)
```

evolute

Srf: Surface to compute mean evolute.

Returns: A surface representing the mean evolute surface.

Description: Computes an "evolute surface" to a given surface using twice the Mean curvature as magnitude.

```
                      1                              |G|
E(u, v) = n(u, v) --------- = n(u, v) ---------------------------------
                  2 H(u, v)           ( G11 L22 + G22 L11 - 2 G12 L12 )
```

Becuase H(u,v) also has n(u,v) we can use the nonnormalized surface normal to compute E(u, v), which is therefore computable and representable.
See also: SymbSrfFff, SymbSrfSff, SymbSrfDeterminant2, SymbSrfGaussCurvature, , SymbSrfMeanCurvatureSqr, SymbSrfMeanNumer, SymbSrfIsoFocalSrf, , SymbSrfCurvatureUpperBound, SymbSrfIsoDirNormalCurvatureBound,

11.2.314 SymbSrfMeanNumer (curvatur.c:1089)

curvature

```
CagdSrfStruct *SymbSrfMeanNumer(const CagdSrfStruct *Srf)
```

evolute

Srf: Surface to compute mean evolute.

Returns: A surface representing the mean evolute surface.

Description: Computes the numerator expression of the Mean as:

```
H(u, v) = G11 L22 + G22 L11 - 2 G12 L12
```

See also: SymbSrfFff, SymbSrfSff, SymbSrfDeterminant2, SymbSrfGaussCurvature, , SymbSrfMeanCurvatureSqr, SymbSrfMeanEvolute, SymbSrfIsoFocalSrf, , SymbSrfCurvatureUpperBound, SymbSrfIsoDirNormalCurvatureBound,

11.2.315 SymbSrfMergeScalar (symb_srf.c:1433)

merge

```
CagdSrfStruct *SymbSrfMergeScalar(const CagdSrfStruct *SrfW,
                                  const CagdSrfStruct *SrfX,
                                  const CagdSrfStruct *SrfY,
                                  const CagdSrfStruct *SrfZ)
```

symbolic computation

SrfW: The weight component of new constructed surface, if have any.

SrfX: The X component of new constructed surface.

SrfY: The Y component of new constructed surface, if have any.

SrfZ: The Z component of new constructed surface, if have any.

Returns: A new surface constructed from given scalar surfaces.

Description: Given a set of scalar surfaces, treat them as coordinates into a new surface. Assumes at least SrfX is not NULL in which a scalar surface is returned. Assumes SrfX/Y/Z/W are either E1 or P1 in which the weights are assumed to be identical and can be ignored if SrfW exists or copied otherwise.
See also: SymbSrfSplitScalar, SymbCrvMergeScalar,

11.2.316 SymbSrfMergeScalarN (symb_srf.c:1344)

merge

symbolic computation

```
CagdSrfStruct *SymbSrfMergeScalarN(CagdSrfStruct * const *SrfVec,
                                   int NumSrfs)
```

SrfVec: A vector of scalar surfaces, SrfVec[0] holds weights if any.

NumSrfs: Number of surfaces in CrvVec.

Returns: A new surface constructed from given scalar surfaces.

Description: Given a vector of scalar surfaces, treat them as coordinates into a new vector surface. Assumes at least SrfVec[1] is not NULL in which case a scalar surface is returned. Assumes SrfVec[i] are either E1 or P1 in which case the weights are assumed to be identical and can be simply copied if exist.
See also: SymbSrfMergeScalar, SymbSrfSplitScalarN,

11.2.317 SymbSrfMult (symb_srf.c:205)

product

symbolic computation

```
CagdSrfStruct *SymbSrfMult(const CagdSrfStruct *Srf1,
                           const CagdSrfStruct *Srf2)
```

Srf1, Srf2: Two surface to multiply coordinate-wise.

Returns: The product of Srf1 * Srf2 coordinate-wise.

Description: Given two surfaces - multiply them coordinate-wise. The two surfaces are promoted to same point type before the multiplication can take place.
See also: SymbSrfDotProd, SymbSrfVecDotProd, SymbSrfScalarScale, SymbSrfMultScalar, , SymbSrfInvert,

11.2.318 SymbSrfMultScalar (symb_srf.c:391)

product

symbolic computation

```
CagdSrfStruct *SymbSrfMultScalar(const CagdSrfStruct *Srf1,
                                 const CagdSrfStruct *Srf2)
```

Srf1, Srf2: Two surfaces to multiply.

Returns: A surface representing the product of Srf1 and Srf2.

Description: Given two surface - a vector curve Srf1 and a scalar curve Srf2, multiply all Srf1's coordinates by the scalar curve Srf2. Returned surface is a surface representing the product of the two given surfaces.
See also: SymbSrfDotProd, SymbSrfVecDotProd, SymbSrfMult, SymbSrfCrossProd, , SymbCrvMultScalar,

11.2.319 SymbSrfNormalSrf (symb_srf.c:942)

normal

symbolic computation

```
CagdSrfStruct *SymbSrfNormalSrf(const CagdSrfStruct *Srf)
```

Srf: To compute an unnormalized normal vector field for.

Returns: A vector field representing the unnormalized normal vector field of Srf.

Description: Given a surface - compute its unnormalized normal vector field surface, as the cross product if its partial derivatives.
See also: Symb2DSrfJacobian, SymbSrfNormalSrfReversed,

11.2.320 SymbSrfNormalSrfReversed (symb_srf.c:974)

normal

symbolic computation

```
CagdSrfStruct *SymbSrfNormalSrfReversed(const CagdSrfStruct *Srf)
```

Srf: To compute an unnormalized normal vector field for.

Returns: A vector field representing the unnormalized normal vector field of Srf.

Description: Given a surface - compute its unnormalized normal vector field surface, as the cross product if its partial derivatives. Here, the normal field is reversed, compared to SymbSrfNormalSrf.
See also: Symb2DSrfJacobian, SymbSrfNormalSrf,

11.2.321 SymbSrfOffset (offset.c:513)

offset

```
CagdSrfStruct *SymbSrfOffset(const CagdSrfStruct *CSrf, CagdRType OffsetDist)
```

CSrf: To approximate its offset surface with distance OffsetDist.

OffsetDist: Amount of offset. Negative denotes other offset direction.

Returns: An approximation to the offset surface.

Description: Given a surface and an offset amount OffsetDist, returns an approximation to the offset surface by offseting the control mesh in the normal direction.

See also: SymbCrvOffset, SymbCrvSubdivOffset, SymbSrfSubdivOffset, , SymbCrvAdapOffset, SymbCrvAdapOffsetTrim, SymbCrvLeastSquarOffset, , SymbCrvMatchingOffset,

11.2.322 SymbSrfOrthotomic (orthotom.c:117)

```
CagdSrfStruct *SymbSrfOrthotomic(const CagdSrfStruct *Srf,
                                 const CagdPType P,
                                 CagdRType K)
```

Srf: To compute its K-orthotomic

P: The points to which the K-orthotomic is computed for Srf for.

K: The magnitude of the orthotomic function.

Returns: The K-orthotomic

Description: Computes the K-orthotomic of a surface with respect to point P:

```
P + K < (S(u,v) - P), N(u,v) > N(u,v)
```

See "Fundamentals of Computer Aided Geometric Design, by J. Hoschek and and D. Lasser.

See also: SymbCrvOrthotomic, SymbSrfSilhouette,

11.2.323 SymbSrfPolarSilhouette (orthotom.c:263)

```
IPPolygonStruct *SymbSrfPolarSilhouette(const CagdSrfStruct *Srf,
                                        const CagdVType VDir,
                                        CagdRType SubdivTol,
                                        CagdBType Euclidean)
```

Srf: To compute its polar silhouette edges.

VDir: Axis of polar silhouette.

SubdivTol: Accuracy of computation.

Euclidean: If TRUE, returns the silhouettes in Euclidean space. Otherwise, the silhouette edges are returned in the Parametric domain.

Returns: The silhouettes as piecewise linear edges.

Description: Computes the polar silhouette edges of the given surfaces, along axis VDir. Equal to $<$ S(u, v) x N(u, v), VDir $> = 0$.

See also: SymbSrfOrthotomic, SymbSrfSilhouette, SymbSrfIsocline,

11.2.324 SymbSrfPtBisectorSrf3D (crv_skel.c:1556)

bisector

```
CagdSrfStruct *SymbSrfPtBisectorSrf3D(const CagdSrfStruct *CSrf,
                                      const CagdPType Pt)
```

CSrf: Three space surface too compute its bisector surface with Pt.

Pt: A point in three space to compute its bisector with Srf.

Returns: The bisector surface.

Description: Computes the bisector surface of a surface in arbitrary general three space position and a point in three space. Solution bisector surface R is derived by solving the three linear equations of (S for Srf, P for Pt):

```
< dS/du, R > = < dS/du, S >
< dS/dv, R > = < dS/dv, S >
< S - P, R > = (< S, S > - < P, P >) / 2
```

See also: SymbCrvDiameter, SymbCrvCnvxHull, SymbCrvBisectorsSrf, , SymbCrvCrvBisectorSrf3D,

11.2.325 SymbSrfRtnlMult (symb_srf.c:812)

product

symbolic computation

```
CagdSrfStruct *SymbSrfRtnlMult(const CagdSrfStruct *Srf1X,
                               const CagdSrfStruct *Srf1W,
                               const CagdSrfStruct *Srf2X,
                               const CagdSrfStruct *Srf2W,
                               CagdBType OperationAdd)
```

Srf1X: Numerator of first surface.

Srf1W: Denominator of first surface. Can be NULL.

Srf2X: Numerator of second surface.

Srf2W: Denominator of second surface. Can be NULL.

OperationAdd: TRUE for addition, FALSE for subtraction.

Returns: The result of Srf1X Srf2W +/- Srf2X Srf1W.

Description: Given two surfaces - multiply them using the quotient product rule:

```
X = X1 W2 +/- X2 W1
```

All provided surfaces are assumed to be non rational scalar surfaces. Returned is a non rational scalar surface (CAGD_PT_E1_TYPE).

See also: SymbSrfDotProd, SymbSrfVecDotProd, SymbSrfScalarScale, SymbSrfMultScalar, , SymbSrfInvert,

11.2.326 SymbSrfScalarScale (symb_srf.c:347)

scaling

symbolic computation

```
CagdSrfStruct *SymbSrfScalarScale(const CagdSrfStruct *Srf, CagdRType Scale)
```

Srf: A surface to scale by magnitude Scale.

Scale: Scaling factor.

Returns: A surfaces scaled by Scale compared to Srf.

Description: Given a surface, scale it by Scale.

See also: SymbSrfDotProd, SymbSrfVecDotProd, SymbSrfMult, SymbSrfMultScalar, , SymbSrfInvert, SymbSrfCrossProd,

11.2.327 SymbSrfSecondMoment (moments.c:403)

```
CagdRType SymbSrfSecondMoment(const CagdSrfStruct *Srf, int Axis1, int Axis2)
```

Srf: Surface to compute the second moment for.

Axis1, Axis2: 1 for X, 2 for Y, 3 for Z.

Returns: The computed moment.

Description: Computes the second moment of the given surface. The computed moment is for the (signed) volume between the surface and its projection onto the XY plane.

See also: SymbSrfSecondMomentSrf, SymbSrfFirstMoment, SymbSrfVolume,

11.2.328 SymbSrfSecondMomentSrf (moments.c:337)

```
CagdSrfStruct *SymbSrfSecondMomentSrf(const CagdSrfStruct *Srf,
                                      int Axis1,
                                      int Axis2,
                                      CagdBType Integrate)
```

Srf: Surface to compute the first moment for.

Axis1, Axis2: 1 for X, 2 for Y, 3 for Z.

Integrate: TRUE to also integrate the resulting surface.

Returns: The computed moment function.

Description: Computes the second moment function of the given surface. The computed moment is for the (signed) volume between the surface and its projection onto the XY plane.

See also: SymbSrfFirstMoment, SymbSrfFirstMomentSrf, SymbSrfVolumeSrf,

11.2.329 SymbSrfSff (curvatur.c:842)

second fundamental form

```
void SymbSrfSff(const CagdSrfStruct *DuSrf,
                const CagdSrfStruct *DvSrf,
                CagdSrfStruct **SffL11,
                CagdSrfStruct **SffL12,
                CagdSrfStruct **SffL22,
                CagdSrfStruct **SNormal)
```

DuSrf: First derivative of Srf with respect to U.

DvSrf: First derivative of Srf with respect to V.

SffL11: SFF L11 scalar field returned herein.

SffL12: SFF L12 scalar field returned herein.

SffL22: SFF L22 scalar field returned herein.

SNormal: Unnormalized normal vector field returned herein.

Returns: void

Description: Computes coefficients of the second fundamental form of given surface Srf that is prescribed via its two partial derivatives DuSrf and DvSrf. These coefficients are using non normalized normal that is also returned.
See also: SymbSrfFff, SymbSrfTff, SymbSrfDeterminant2, SymbSrfGaussCurvature, , SymbSrfMeanEvolute, SymbSrfMeanCurvatureSqr, SymbSrfIsoFocalSrf, , SymbSrfCurvatureUpperBound, SymbSrfIsoDirNormalCurvatureBound,

11.2.330 SymbSrfSilhouette (orthotom.c:194)

```
IPPolygonStruct *SymbSrfSilhouette(const CagdSrfStruct *Srf,
                                   const CagdVType VDir,
                                   CagdRType SubdivTol,
                                   CagdBType Euclidean)
```

Srf: To compute its silhouette edges.

VDir: View direction vector (a unit vector).

SubdivTol: Accuracy of computation. 0.001 will be a good start.

Euclidean: If TRUE, returns the silhouettes in Euclidean space. Otherwise, the silhouette edges are returned in the Parametric domain.

Returns: The silhouettes as piecewise linear edges.

Description: Computes the silhouette edges of the given surfaces, orthographically seen from the given view direction VDir.
See also: SymbSrfOrthotomic, SymbSrfIsocline, SymbSrfPolarSilhouette, MvarSrfSilhouette,

11.2.331 SymbSrfSmoothInternalCtlPts (prm_dmn.c:605)

```
void SymbSrfSmoothInternalCtlPts(CagdSrfStruct *Srf, CagdRType Weight)
```

Srf: To smooth its internal vcontrol points.

Weight: Of movement. Zero to move nothing. 1.0 to move to average pt.

Returns: void

Description: Mesh smoothing - move all internal points toward the average of their neigbors.
See also:

11.2.332 SymbSrfSplitScalar (symb_srf.c:1282)

split

symbolic computation

```
void SymbSrfSplitScalar(const CagdSrfStruct *Srf,
                        CagdSrfStruct **SrfW,
                        CagdSrfStruct **SrfX,
                        CagdSrfStruct **SrfY,
                        CagdSrfStruct **SrfZ)
```

Srf: Surface to split.

SrfW: The weight component of Srf, if have any.

SrfX: The X component of Srf.

SrfY: The Y component of Srf, if have any.

SrfZ: The Z component of Srf, if have any.

Returns: void

Description: Given a surface splits it to its scalar component surfaces. Ignores all dimensions beyond the third, Z, dimension.
See also: SymbSrfMergeScalar, SymbSrfSplitScalar, SymbSrfSplitScalarN,

11.2.333 SymbSrfSplitScalarN (symb_srf.c:1226)

split

symbolic computation

```
void SymbSrfSplitScalarN(const CagdSrfStruct *Srf,
                         CagdSrfStruct *Srfs[CAGD_MAX_PT_SIZE])
```

Srf: Surface to split.

Srfs: A vector of scalar surfaces - components of Srf. A vector of dynamically allocated scalar srfs.

Returns: void

Description: Given a surface, splits it to its scalar component surfaces.
See also: SymbSrfSplitScalar, SymbSrfMergeScalarN,

11.2.334 SymbSrfSub (symb_srf.c:110)

subtraction

symbolic computation

```
CagdSrfStruct *SymbSrfSub(const CagdSrfStruct *Srf1,
                          const CagdSrfStruct *Srf2)
```

Srf1, Srf2: Two surface to subtract coordinate-wise.

Returns: The difference of Srf1 - Srf2 coordinate-wise.

Description: Given two surfaces - subtract them coordinate-wise. The two surfaces are promoted to same point type before the multiplication can take place. Furthermore, order and continuity are matched as well.
See also: SymbSrfAdd, SymbMeshAddSub, SymbSrfMult,

11.2.335 SymbSrfSubdivOffset (offset.c:790)

offset

```
CagdSrfStruct *SymbSrfSubdivOffset(const CagdSrfStruct *CSrf,
                                   CagdRType OffsetDist,
                                   CagdRType Tolerance)
```

CSrf: To approximate its offset surface with distance OffsetDist.

OffsetDist: Amount of offset. Negative denotes other offset direction.

Tolerance: Accuracy control.

Returns: An approximation to the offset surface, to within Tolerance.

Description: Given a surface and an offset amount OffsetDist, returns an approximation to the offset surface by offseting the control mesh in the normal direction. If resulting offset is not satisfying the required tolerance the surface is subdivided and the algorithm recurses on both parts.
See also: SymbCrvOffset, SymbCrvSubdivOffset, SymbSrfOffset, SymbCrvAdapOffset, , SymbCrvAdapOffset-Trim, SymbCrvLeastSquarOffset, SymbCrvMatchingOffset,

11.2.336 SymbSrfTff (curvatur.c:892)

third fundamental form

```
void SymbSrfTff(const CagdSrfStruct *Srf,
                CagdSrfStruct **TffL11,
                CagdSrfStruct **TffL12,
                CagdSrfStruct **TffL22)
```

Srf: Surface to compute the coefficents of the TFF for.

TffL11: TFF L11 scalar field returned herein.

TffL12: TFF L12 scalar field returned herein.

TffL22: TFF L22 scalar field returned herein.

Returns: void

Description: Computes coefficients of the third fundamental form of given surface Srf. These coefficients are using non normalized normal that is also returned. The coefficients of the TFF equal:

$$Lij = \left\langle \frac{d\,n}{dui}, \frac{d\,n}{duj} \right\rangle = \frac{\left\langle \frac{d\,m}{dui}, \frac{d\,m}{duj} \right\rangle - \left\langle m, \frac{d\,m}{dui} \right\rangle \left\langle m, \frac{d\,m}{duj} \right\rangle}{\langle m, m \rangle \hat{} 2}$$

where n is the unit normal of Srf and m = dSrf/dui x dSrf/duj, the unnormalized normal field of Srf.
See also: SymbSrfFff, SymbSrfSff, SymbSrfDeterminant2, ,

11.2.337 SymbSrfVarOffset (offset.c:635)

offset

```
CagdSrfStruct *SymbSrfVarOffset(const CagdSrfStruct *CSrf,
                                const CagdSrfStruct *VarOffsetDist)
```

CSrf: To approximate its variable offset surface.

VarOffsetDist: Scalar function prescribing the amount of offset. Must posses a parametric domain similar to CSrf.

Returns: An approximation to the varying offset amount surface.

Description: Given a surface and an offset amount function Var OffsetDist, returns an approximation to the offset surface by offseting the control polygon in the normal direction.
See also: SymbSrfSubdivOffset, SymbSrfOffset,

11.2.338 SymbSrfVecCrossProd (symb_srf.c:698)

product

cross product

symbolic computation

```
CagdSrfStruct *SymbSrfVecCrossProd(const CagdSrfStruct *Srf,
                                   const CagdVType Vec)
```

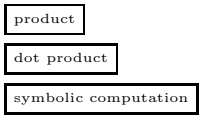
Srf: Surface to multiply and compute a cross product for.

Vec: Vector to cross product Srf with.

Returns: A vector surface representing the cross product of Srf x Vec.

Description: Given a surface and a vector - computes their cross product. Returned surface is a scalar surface representing the cross product of the surface and vector.
See also: SymbSrfDotProd, SymbSrfVecDotProd, SymbSrfScalarScale, SymbSrfMultScalar, , SymbSrfInvert, SymbSrfCrossProd,

11.2.339 SymbSrfVecDotProd (symb_srf.c:514)




```
CagdSrfStruct *SymbSrfVecDotProd(const CagdSrfStruct *Srf,
                                 const CagdVType Vec)
```

Srf: Surface to multiply and compute a dot product for.

Vec: Vector to project Srf onto.

Returns: A scalar surface representing the dot product of Srf . Vec.

Description: Given a surface and a vector - computes their dot product. Returned surface is a scalar surface representing the dot product.
See also: SymbSrfDotProd, SymbSrfMult, SymbSrfScalarScale, SymbSrfMultScalar, , SymbSrfInvert, SymbSrfCrossProd, SymbSrfVecCrossProd,

11.2.340 SymbSrfVolume1 (moments.c:99)

```
CagdRType SymbSrfVolume1(const CagdSrfStruct *Srf)
```

Srf: Surface to computes its enclosed volume.

Returns: The enclosed volume.

Description: A function to compute the enclosed volume by the given surface. The computed volume is the (signed) volume between the surface and its projection onto the XY plane.
See also: SymbSrfVolume1Srf, SymbSrfVolume2Srf, SymbSrfVolume2,

11.2.341 SymbSrfVolume1Srf (moments.c:30)

```
CagdSrfStruct *SymbSrfVolume1Srf(const CagdSrfStruct *Srf, CagdBType Integrate)
```

Srf: Surface to computes its enclosed volume.

Integrate: TRUE to also integrate the resulting surface.

Returns: Integral volume function.

Description: A function to compute the enclosed volume function of the given surface. The computed volume is the (signed) volume between the surface and its projection onto the XY plane.
See also: SymbSrfVolume1, SymbSrfVolume2Srf, SymbSrfVolume2,

11.2.342 SymbSrfVolume2 (moments.c:188)

```
CagdRType SymbSrfVolume2(const CagdSrfStruct *Srf)
```

Srf: Surface to computes its enclosed volume.

Returns: The enclosed volume.

Description: A function to compute the enclosed volume by the given surface. The computed volume is the (signed) volume occupied by all rays from the origin to the surface.
See also: SymbSrfVolume1Srf, SymbSrfVolume2Srf, SymbSrfVolume2,

11.2.343 SymbSrfVolume2Srf (moments.c:139)

```
CagdSrfStruct *SymbSrfVolume2Srf(const CagdSrfStruct *Srf,
                                 CagdBType Integrate)
```

Srf: Surface to computes its enclosed volume.

Integrate: TRUE to also integrate the resulting surface.

Returns: Integral volume function.

Description: A function to compute the enclosed volume function of the given surface. The computed volume is the (signed) volume occupied by all rays from the origin to the surface.
See also: SymbSrfVolume2, SymbSrfVolume1Srf, SymbSrfVolume1,

11.2.344 SymbSwungAlgSumSrf (constrct.c:160)

```
CagdSrfStruct *SymbSwungAlgSumSrf(const CagdCrvStruct *Crv1,
                                  const CagdCrvStruct *Crv2)
```

Crv1, Crv2: Two curves to sum algebraically, forming a swung surface.

Returns: A surface represent their swung algebraic sum.

Description: Adds up algebraically the given two curves, C1(r) and C2(t), as swung surfaces (The NURBs book', by Piegl and Tiller, pp 455):
S(r, t) = (x1(r) x2(t), x1(r) y2(t), y1(r))

See also: SymbAlgebraicProdSrf, SymbAlgebraicSumSrf,

11.2.345 SymbTangentConeForCrvMalloc (nrmlcone.c:136)

tangents

tangent bound

```
const SymbNormalConeStruct *SymbTangentConeForCrvMalloc(const CagdCrvStruct
                                                        *Crv,
                                                        CagdBType Planar)
```

Crv: To compute a tangent cone for.

Planar: If TRUE, only the X and Y coefficients are considered.

Returns: The computed tangent cone

Description: Computes a tangent cone for a given curve, by examine the control polygon of the curve and deriving its angular span.

See also: SymbNormalConeOverlap, SymbNormalConeForSrf,

11.2.346 SymbTangentConeForCrvToData (nrmlcone.c:45)

tangents

tangent bound

```
const SymbNormalConeStruct *SymbTangentConeForCrvToData(const CagdCrvStruct
                                                        *Crv,
                                                        CagdBType Planar,
                                                        SymbNormalConeStruct
                                                                *TangentCone)
```

Crv: To compute a tangent cone for.

Planar: If TRUE, only the X and Y coefficients are considered.

TangentCone: The computed tangent cone

Returns: The computed tangent cone

Description: Computes a tangent cone for a given curve, by examine the control polygon of the curve and deriving its angular span.

See also: SymbNormalConeOverlap, SymbNormalConeForSrf,

11.2.347 SymbTangentToCrvAtTwoPts (crv_tans.c:140)

```
CagdPtStruct *SymbTangentToCrvAtTwoPts(const CagdCrvStruct *CCrv,
                                       CagdRType SubdivTol)
```

CCrv: The curve to compute all tangent lines at two locations.

SubdivTol: Of numeric search for the zero set (for surface subdivision). A positive value (0.01 is a good start).

Returns: A list of parameter location on Crv with tangent lines through Pt. Parameters are save in the X & Y coordinate.

Description: Computes all the lines that are tangent to Crv at two locations. Returned is a list of parameter locations' pairs where the tangent is tangent to the curve. The tangents are computed as two sets of contours of the solution to the two equations of:

1. `[ C(t) - C(r) ] || C'(t)`
2. `[ C(t) - C(r) ] || C'(r)`

and computing all the intersection points between these two sets of contours. Note that since equations 1 and 2 are symmetric, one only needs to solve for once and flip the notation of r and t.

See also: SymbCrvPtTangents, SymbCircTanTo2Crvs, SymbCrvCnvxHull, SymbCrvDiameter, , MvarMVTriTangentLine,

11.2.348 SymbTorusPointBisect (smp_skel.c:915)

bisectors

skeleton

```
CagdSrfStruct *SymbTorusPointBisect(const CagdVType TrsCntr,
                                    const CagdVType TrsDir,
                                    CagdRType TrsMajorRad,
                                    CagdRType TrsMinorRad,
                                    const CagdPType Pt)
```

TrsCntr: Center of constructed torus.

TrsDir: Axis of symmetry of constructed torus.

TrsMajorRad: Major radius of constructed torus.

TrsMinorRad: Minor radius of constructed torus.

Pt: Direction of line from origin.

Returns: Constructed bisector surface.

Description: Compute the bisector surface between a torus and a point.
See also: SymbPtCrvBisectOnSphere, , SymbPlanePointBisect, SymbCylinPointBisect, SymbConePointBisect, , SymbSpherePointBisect, , SymbConePlaneBisect, SymbCylinPlaneBisect, SymbSpherePlaneBisect, , SymbConeLineBisect, SymbSphereLineBisect, SymbCylinSphereBisect, SymbSphereSphereBisect, SymbConeSphereBisect, , SymbTorusSphereBisect, SymbTorusTorusBisect, SymbConeConeBisect, , SymbConeConeBisect2, SymbConeCylinBisect, SymbCylinCylinBisect,

11.2.349 SymbTorusSphereBisect (smp_skel.c:1502)

bisectors

skeleton

```
CagdSrfStruct *SymbTorusSphereBisect(const CagdVType TrsCntr,
                                     const CagdVType TrsDir,
                                     CagdRType TrsMajorRad,
                                     CagdRType TrsMinorRad,
                                     const CagdVType SprCntr,
                                     CagdRType SprRad)
```

TrsCntr: Center of constructed torus.

TrsDir: Axis of symmetry of constructed torus.

TrsMajorRad: Major radius of constructed torus.

TrsMinorRad: Minor radius of constructed torus.

SprCntr: Center location of the sphere. Must be in northern hemisphere (positive Z coefficient).

SprRad: Radius of sphere.

Returns: Constructed bisector surface.

Description: Compute the bisector surface between a torus and a sphere. The computation is reduced to that of a bisector between a point and a torus, that has a rational form.
See also: SymbPlanePointBisect, SymbCylinPointBisect, SymbConePointBisect, , SymbSpherePointBisect, SymbTorusPointBisect, , SymbConeLineBisect, SymbSphereLineBisect, SymbPlaneLineBisect, , SymbCylinPlaneBisect, SymbConePlaneBisect, SymbSpherePlaneBisect, , SymbCylinSphereBisect, SymbSphereSphereBisect, SymbConeSphereBisect, , SymbConeConeBisect, SymbConeConeBisect2, SymbConeCylinBisect, , SymbCylinCylinBisect, SymbTorusTorusBisect,

11.2.350 SymbTorusTorusBisect (smp_skel.c:1555)

bisectors

skeleton

```
CagdSrfStruct *SymbTorusTorusBisect(const CagdVType Trs1Cntr,
                                    const CagdVType Trs1Dir,
                                    CagdRType Trs1MajorRad,
                                    const CagdVType Trs2Cntr,
                                    const CagdVType Trs2Dir,
                                    CagdRType Trs2MajorRad,
                                    CagdRType Alpha)
```

Trs1Cntr: Center of constructed torus1

Trs1Dir: Axis of symmetry of constructed torus1.

Trs1MajorRad: Major radius of constructed torus1.

Trs2Cntr: Center of constructed torus2.

Trs2Dir: Axis of symmetry of constructed torus2.

Trs2MajorRad: Major radius of constructed torus2.

Alpha: Some blend factor that allows one to shift the bisector a bit. Zero for no effect and +/- to shift the bisector toward one of the input surfaces.

Returns: Constructed bisector surface.

Description: Compute the bisector surface between two tori. The Bisector is computed as the solution of the following three linear equations:

```
< B - C1(u), C1'(u) > = 0,


< B - C2(v), C2'(V) > = 0,


< B, C2(u) - C1(v) > = (C2^2(u) - C1^2(v)) - Alpha) / 2,
```

where C1 and C2 are the circular main axes of the tori and R1 and R2 are their minor radii.

See also: SymbPlanePointBisect, SymbCylinPointBisect, SymbConePointBisect, , SymbSpherePointBisect, SymbTorusPointBisect, , SymbConeLineBisect, SymbSphereLineBisect, SymbPlaneLineBisect, , SymbCylinPlaneBisect, SymbConePlaneBisect, SymbSpherePlaneBisect, , SymbCylinSphereBisect, SymbSphereSphereBisect, SymbConeSphereBisect, , SymbConeConeBisect, SymbConeConeBisect2, SymbConeCylinBisect, , SymbCylinCylinBisect, SymbTorusSphereBisect,

11.2.351 SymbTwoCrvsMorphing (morphing.c:58)

morphing

```
CagdCrvStruct *SymbTwoCrvsMorphing(const CagdCrvStruct *Crv1,
                                   const CagdCrvStruct *Crv2,
                                   CagdRType Blend)
```

Crv1, Crv2: The two curves to blend.

Blend: A parameter between zero and one

Returns: Crv2 * Blend + Crv1 * (1 - Blend).

Description: Given two compatible curves (See function CagdMakeCrvsCompatible), computes a convex blend between them according to Blend which must be between zero and one. Returned is the new blended curve.

See also: SymbTwoCrvsMorphingCornerCut, SymbTwoCrvsMorphingMultiRes, SymbTwoSrfsMorphing, TrivTwoTVsMorphing,

11.2.352 SymbTwoCrvsMorphingCornerCut (morphing.c:128)

morphing

```
CagdCrvStruct *SymbTwoCrvsMorphingCornerCut(const CagdCrvStruct *Crv1,
                                            const CagdCrvStruct *Crv2,
                                            CagdRType MinDist,
                                            CagdBType SameLength,
                                            CagdBType FilterTangencies)
```

Crv1, Crv2: The two curves to blend.

MinDist: Minimal maximum distance between adjacent curves to make sure motion is visible. The curves will move at most twice that much in their maximal distance (roughly).

SameLength: If TRUE, length of curves is preserved, otherwise BBOX is.

FilterTangencies: If TRUE, attempt is made to eliminate the intermediate line representation.

Returns: The blended curve.

Description: Given two compatible curves (See function CagdMakeCrvsCompatible), computes a morph between them using corner cutting approach. Returned is the new blended curve.

See also: SymbTwoCrvsMorphing, SymbTwoCrvsMorphingMultiRes, , SymbTwoSrfsMorphing, TrivTwoTVsMorphing,

11.2.353 SymbTwoCrvsMorphingMultiRes (morphing.c:316)

morphing

```
CagdCrvStruct *SymbTwoCrvsMorphingMultiRes(const CagdCrvStruct *Crv1,
                                           const CagdCrvStruct *Crv2,
                                           CagdRType BlendStep)
```

Crv1, Crv2: The two curves to blend.

BlendStep: A step size of the blending.

Returns: A list of blended curves.

Description: Given two compatible curves (See function CagdMakeCrvsCompatible), computes a morph between them using multiresolution decomposition. Returned is a list of new blended curves.
See also: SymbTwoCrvsMorphing, SymbTwoCrvsMorphingCornerCut, , SymbTwoSrfsMorphing, TrivTwoTVsMorphing,

11.2.354 SymbTwoSrfsMorphing (morphing.c:760)

morphing

```
CagdSrfStruct *SymbTwoSrfsMorphing(const CagdSrfStruct *Srf1,
                                   const CagdSrfStruct *Srf2,
                                   CagdRType Blend)
```

Srf1, Srf2: The two surfaces to blend.

Blend: A parameter between zero and one

Returns: Srf2 * Blend + Srf1 * (1 - Blend).

Description: Given two compatible surfaces (See function CagdMakeSrfsCompatible), computes a convex blend between them according to Blend which must be between zero and one. Returned is the new blended surface.
See also: SymbTwoCrvsMorphing, SymbTwoCrvsMorphingCornerCut, , SymbTwoCrvsMorphingMultiRes, TrivTwoTVsMorphing,

11.2.355 SymbUniformAprxPtOnCrvDistrib (ffptdist.c:44)

uniform distribution

```
CagdRType *SymbUniformAprxPtOnCrvDistrib(const CagdCrvStruct *Crv,
                                         CagdBType ParamUniform,
                                         int n)
```

Crv: To place n points along, uniformly.

ParamUniform: If TRUE, produces a distribution uniform in parametric space. If FALSE, uniform in Euclidean space.

n: Number of points to distribute along Crv.

Returns: A dynamically allocated vector of size n, of the parameter values of the distributed points.

Description: Computes a stocastically uniform distribution of points on a curve. n points are placed at approximately equal distance from each other along Crv's arc length. This distribution converges to a uniform distribution as n approached infinity.
See also: SymbUniformAprxPtOnSrfDistrib,

11.2.356 SymbUniformAprxPtOnSrfDistrib (ffptdist.c:119)

uniform distribution

```
CagdUVType *SymbUniformAprxPtOnSrfDistrib(
                        const CagdSrfStruct *Srf,
                        CagdBType ParamUniform,
                        int n,
                        SymbUniformAprxSrfPtImportanceFuncType EvalImportance)
```

Srf: To place n points on, uniformly.

ParamUniform: If TRUE, produces a distribution uniform in parametric space. If FALSE, uniform in Euclidean space.

n: Number of points to distribute along Srf.

EvalImportance: Optional function to evaluate the importance of each selected points and if returning FALSE, that point is purged. NULL to disable.

Returns: A dynamically allocated vector of size n, of parameter values of the distributed points.

Description: Computes a stochastically uniform distribution of points on a surface. n points are placed at approximately equal distance from each other on Srf's surface. This distribution converges to a uniform distribution as n approached infinity.
See also: SymbUniformAprxPtOnCrvDistrib,

11.2.357 SymbUniformAprxPtOnSrfGetDistrib (ffptdist.c:260)

uniform distribution

```
CagdUVType *SymbUniformAprxPtOnSrfGetDistrib(const CagdSrfStruct *Srf,
                                             CagdUVType *DistUV,
                                             CagdRType *DistProb,
                                             int DistSize,
                                             int *n)
```

Srf: To place points on its parametric space, uniformly. This surface must have the same parameter domain as Srf in the last invokation of SymbUniformAprxPtOnSrfPrepDistrib.

DistUV: Distributed UV.

DistProb: Distributed Prob.

DistSize: Distributed Size.

n: Returns actual number of UV locations in the returned vector.

Returns: A dynamically allocated vector of at most n parameter values of the distributed points.

Description: Computes a uniform distribution of points on the surface Srf. The points are placed at approximately equal distance from each other on Srf's Euclidean space. A subset of the n points that were selected via the last invokation of SymbUniformAprxPtOnSrfPrepDistrib is returned, such that the points are at equal distance, approximately.
See also: SymbUniformAprxPtOnCrvDistrib, SymbUniformAprxPtOnSrfDistrib, , SymbUniformAprxPtOnSrfPrepDistrib,

11.2.358 SymbUniformAprxPtOnSrfPrepDistrib (ffptdist.c:205)

uniform distribution

```
void SymbUniformAprxPtOnSrfPrepDistrib(const CagdSrfStruct *Srf,
                                       CagdUVType **DistUV,
                                       CagdRType **DistProb,
                                       int *DistSize,
                                       int n)
```

Srf: To place n points on its parametric space, uniformly.

DistUV: Prepared distrib UV, allocated dynammically.

DistProb: Prepared distrib Prob, allocated dynammically.

DistSize: Prepared distrib Size (of DistUV and DistProb).

n: Number of points to distribute along Srf.

Returns: void

Description: Prepares a uniform distribution of points on surface Srf. This function is invoked in preparation of several calls to function SymbUniformAprxPtOnSrfGetDistrib that return a uniform Euclidean distributions that is consistent with the area differentials found.
See also: SymbUniformAprxPtOnCrvDistrib, SymbUniformAprxPtOnSrfDistrib, , SymbUniformAprxPtOnSrfGetDistrib,

Chapter 12

Trimmed surfaces Library, trim_lib

12.1 General Information

This library provides a set of functions to manipulate freeform trimmed Bezier and/or NURBs surfaces. This library heavily depends on the cagd library. Functions are provided to create, copy, and destruct trimmed surfaces to extract isoparametric curves, to evaluate, refine and subdivide, to read and write trimmed surfaces, degree raise, and approximate using polygonal representations. A trimming surface is defined out of a tensor product surface and a set of trimming loops that trims out portions of the parametric space of the surface,

```
typedef struct TrimSrfStruct {
    struct TrimSrfStruct *Pnext;
    IPAttributeStruct *Attr;
    int Tags;
    CagdSrfStruct *Srf;   /* Surface trimmed by TrimCrvList. */
    TrimCrvStruct *TrimCrvList;          /* List of trimming curves. */
} TrimSrfStruct;
```

Each trimming loop consists of a set of trimming curve segments:

```
typedef struct TrimCrvStruct {
    struct TrimCrvStruct *Pnext;
    IPAttributeStruct *Attr;
    TrimCrvSegStruct *TrimCrvSegList;    /* List of trimming curve segments. */
} TrimCrvStruct;
```

Each trimming curve segment contains a representation for the curve in the UV space of the surface as well as a representation in the Euclidean space,

```
typedef struct TrimCrvSegStruct {
    struct TrimCrvSegStruct *Pnext;
    IPAttributeStruct *Attr;
    CagdCrvStruct *UVCrv;    /* Trimming crv segment in srf's param. domain. */
    CagdCrvStruct *EucCrv;       /* Trimming curve as an E3 Euclidean curve. */
} TrimCrvSegStruct;
```

The interface of the library is defined in *include/trim_lib.h*.

This library has its own error handler, which by default prints an error message and exit the program called **TrimFatalError**.

All globals in this library have a prefix of **Trim**.

12.2 Library Functions

12.2.1 IritTrimDescribeError (trim_err.c:59)

error handling

```
const char *IritTrimDescribeError(IritTrimFatalErrorType ErrorNum)
```

ErrorNum: Type of the error that was raised.

Returns: A string describing the error type.

Description: Returns a string describing a the given error. Errors can be raised by any member of this trim library as well as other users. Raised error will cause an invocation of IritTrimFatalError function which decides how to handle this error. IritTrimFatalError can for example, invoke this routine with the error type, print the appropriate message and quit the program.

12.2.2 IritTrimFatalError (trim_ftl.c:56)

error handling

```
void IritTrimFatalError(IritTrimFatalErrorType ErrID)
```

ErrID: Error type that was raised.

Returns: void

Description: Trap Trim_lib errors right here. Provides a default error handler for the trim library. Gets an error description using IritTrimDescribeError, prints it and exit the program using exit.

12.2.3 IritTrimSetFatalErrorFunc (trim_ftl.c:28)

error handling

```
TrimSetErrorFuncType IritTrimSetFatalErrorFunc(TrimSetErrorFuncType ErrorFunc)
```

ErrorFunc: New error function to use.

Returns: Old error function reference.

Description: Sets the error function to be used by Trim_lib.

12.2.4 TrimAffineTransTrimCurves (trim_aux.c:1924)

trimming curves

```
void TrimAffineTransTrimCurves(TrimCrvStruct *TrimCrvList,
                               CagdRType OldUMin,
                               CagdRType OldUMax,
                               CagdRType OldVMin,
                               CagdRType OldVMax,
                               CagdRType NewUMin,
                               CagdRType NewUMax,
                               CagdRType NewVMin,
                               CagdRType NewVMax)
```

TrimCrvList: Trimming curves to affinely map.

OldUMin, OldUMax, OldVMin, OldVMax: Domain to map trimming curves from.

NewUMin, NewUMax, NewVMin, NewVMax: Domain to map trimming curves to.

Returns: void

Description: Map the given trimming curves into a new domain, in place.
See also: TrimAffineTransTrimSrf,

12.2.5 TrimAffineTransTrimSrf (trim_aux.c:1976)

trimming curves

```
TrimSrfStruct *TrimAffineTransTrimSrf(const TrimSrfStruct *CTrimSrf,
                                      CagdRType NewUMin,
                                      CagdRType NewUMax,
                                      CagdRType NewVMin,
                                      CagdRType NewVMax)
```

CTrimSrf: Trimmed surface to affinely map its parametric domain.

NewUMin, NewUMax, NewVMin, NewVMax: New parametric domain to map to.

Returns: A trimmed surface that is geometrically identical but with new different parametric domain.

Description: Maps the given trimmed surface into a new domain.
See also: BspKnotAffineTransOrder2, TrimAffineTransTrimCurves,

12.2.6 TrimAllPrisaSrfs (tr_prisa.c:59)

layout

prisa

```
TrimSrfStruct *TrimAllPrisaSrfs(const TrimSrfStruct *TSrfs,
                                int SamplesPerCurve,
                                CagdRType Epsilon,
                                CagdSrfDirType Dir,
                                CagdVType Space)
```

TSrfs: To approximate and flatten out.

SamplesPerCurve: During the approximation of a ruled surface as a developable surface.

Epsilon: Accuracy control for the piecewise ruled surface approximation. if Epsilon us positive, the surfaces are laid down on the plane, otherwise they are return as 3-space ruled surfaces and form a piecewise ruled-surface approximation to Srfs.

Dir: Direction of ruled/developable surface approximation. Either U or V.

Space: A vector in the XY plane to denote the amount of translation from one flattened out surface to the next.

Returns: A list of planar trimmed surfaces denoting the layout (prisa) of the given TSrfs to the accuracy requested.

Description: Computes a piecewise ruled surface approximation to a given set of trimmed surfaces with given Epsilon, and lay them out "nicely" onto the XY plane, by approximating each ruled surface as a developable surface with SamplesPerCurve samples. Dir controls the direction of ruled approximation, SpaceScale and Offset controls the placement of the different planar pieces. Prisa is the hebrew word for the process of flattening out a three dimensional surface. I have still to find an english word for it.
See also: TrimPiecewiseRuledSrfApprox, TrimPrisaRuledSrf, SymbAllPrisaSrfs,

12.2.7 TrimClassifyTrimCrvsOrientation (trim_aux.c:1432)

trimming curves

```
void TrimClassifyTrimCrvsOrientation(TrimCrvStruct *TCrvs, CagdRType Tol)
```

TCrvs: Trimming curve to classify their orientation, in place.

Tol: Tolerance of end points comparisons.

Returns: void

Description: Go over all given loops and classify the individual trimming curves as "reversed" if indeed the curve is reversed in the loop. Assumes the trimming curves were already linked into loops.
See also: TrimLinkTrimmingCurves2Loops, TrimLinkTrimmingCurves2Loops1, , TrimLinkTrimmingCurves2Loops2, TrimMergeTrimmingCurves2Loops, TrimClassifyTrimLoopOrient,

12.2.8 TrimClassifyTrimCurveOrient (trim2ply.c:2064)

```
CagdBType TrimClassifyTrimCurveOrient(const CagdCrvStruct *UVCrv)
```

UVCrv: Trimming curve to examine its orientation.

Returns: TRUE if the curve is clockwise, FALSE if counter clockwise.

Description: Given a closed, piecewise linear trimming curve, returns TRUE if the curve is clockwise, FALSE if counter clockwise. Orientation is determined by computing the signed area of the polygon and examining the sign of the result.
See also: CagdCrvAreaPoly,

12.2.9 TrimClassifyTrimLoopOrient (trim2ply.c:2089)

```
CagdBType TrimClassifyTrimLoopOrient(const TrimCrvSegStruct *TSegs)
```

TSegs: Trimming loop to examine its orientation.

Returns: TRUE if the curve is clockwise, FALSE if counter clockwise.

Description: Given a closed loop consisting of several, piecewise linear, trimming curves, returns TRUE if loop is clockwise, FALSE if counter clockwise. Orientation is determined by computing the signed area of the polygon and examining the sign of the result.
See also: TrimClassifyTrimmingLoops, TrimClassifyTrimCurveOrient, TrimClassifyTrimCrvsOrientation,

12.2.10 TrimClassifyTrimmingLoops (trim2ply.c:1844)

```
int TrimClassifyTrimmingLoops(TrimCrvStruct **TrimLoops)
```

TrimLoops: Input loops, reorganized in place.

Returns: TRUE if successful, FALSE otherwise.

Description: Classify the given trimming curve loops into a hierarchy. All curves with even nesting levels are considered outside loops and are oriented clockwise. All curves with odd nesting level are considered islands and are oriented counterclockwise. An island Ci of outside loop Cj will be placed in an "_subTrims" attribute under Ci.
See also: TrimCrvsHierarchy2Polys, TrimClassifyTrimCurveOrient, , TrimCrvFreeWithSubTrims, TrimCrvFreeListWithSubTrims,

12.2.11 TrimClipSrfToTrimCrvs (trim_aux.c:2810)

```
TrimSrfStruct *TrimClipSrfToTrimCrvs(TrimSrfStruct *TrimSrf)
```

TrimSrf: Input trimmed surface to extract a minimal valid region

Returns: Returns a possible smaller surface with similar geometry.

Description: Extract the minimal region of the tensor product surface that contains the domain that is prescribed by the trimming curves. The return surface represents the exact same geometry as the input surface but possible with a small size.
See also: TrimSrfTrimCrvSquareDomain, TrimSrfTrimCrvAllDomain,

12.2.12 TrimCnvrtBsp2BzrSrf (trim_sub.c:1126)

subdivision

```
TrimSrfStruct *TrimCnvrtBsp2BzrSrf(const TrimSrfStruct *TrimSrf)
```

TrimSrf: A trimmed Bezier or B-spline surface to subdivide into Bezier-like patches.

Returns: The subdivided surfaces, represented as a linked list.

Description: Subdivides a trimmed surface into patches in which all the surfaces are Bezier-like (that is, have no internal knots in either U or V direction). The trimming curves are also obviously subdivided. The result is returned as a list of trimmed Bezier-like surfaces.

12.2.13 TrimCoerceTrimUVCrv2Plane (trim_aux.c:1065)

trimming curves

```
int TrimCoerceTrimUVCrv2Plane(TrimCrvSegStruct *TSeg)
```

TSeg: Trimming curve segment to coerce to teh XY plane, in place.

Returns: TRUE if curve segment was modified.

Description: Make sure the UV curve in the trimming curve segment is planar (Z == 0). If not, the UV is projected to the XY plane, in place.
See also:

12.2.14 TrimCrv2Polyline (trim_aux.c:2099)

piecewise linear approximation

polyline

```
CagdPolylineStruct *TrimCrv2Polyline(const CagdCrvStruct *TrimCrv,
                                     CagdRType TolSamples,
                                     SymbCrvApproxMethodType Method,
                                     CagdBType OptiLin)
```

TrimCrv: To approximate as a polyline.

TolSamples: Tolerance of approximation error (Method = 2) or Number of samples to compute on polyline (Method = 0, 1).

Method: 0 - TolSamples are set uniformly in parametric space, 1 - TolSamples are set optimally, considering the isocurve's curvature. 2 - TolSamples sets the maximum error allowed between the piecewise linear approximation and original curve.

OptiLin: If TRUE, optimize linear curves.

Returns: A polyline representing the piecewise linear approximation from, or NULL in case of an error.

Description: Routine to approx. a single curve as a polyline with TolSamples samples/tolerance. Polyline is always E3 CagdPolylineStruct type. NULL is returned in case of an error, otherwise CagdPolylineStruct.
See also: BspCrv2Polyline, BzrCrv2Polyline, IritCurve2Polylines, SymbCrv2Polyline, , TrimCrvs2Polylines,

12.2.15 TrimCrvAgainstTrimCrvs (trim_iso.c:594)

```
CagdCrvStruct *TrimCrvAgainstTrimCrvs(CagdCrvStruct *UVCrv,
                                      const TrimSrfStruct *TrimSrf,
                                      CagdRType Eps)
```

UVCrv: A curve in the parametric space to trim. Used in place and Freed at the end. Can be a non isoparametric curve.

TrimSrf: A trimmed surface.

Eps: Tolerance of approximation.

Returns: A list of trimmed segments of UVCrv inside TrimSrf.

Description: Trim a given curve in UV space of a trimmed surface to the valid domain only. Returned is a list of segments of UV curve that is inside TrimSrf.
See also: TrimIntersectTrimCrvIsoVals,

12.2.16 TrimCrvBBox (trim_aux.c:142)

bbox

bounding box

```
int TrimCrvBBox(const TrimCrvStruct *TCrv, int UV, CagdBBoxStruct *BBox)
```

TCrv: To compute a bounding box for.

UV: TRUE to process the UV curve, FALSE for the Euclidean curve.

BBox: Where bounding information is to be saved.

Returns: TRUE if successful, FALSE otherwise

Description: Computes a bounding box for a freeform trimmed curve.ffl
See also: TrimSrfBBox, TrimCrvListBBox, CagdCrvBBox,

12.2.17 TrimCrvCopy (trim_gen.c:295)

allocation

```
TrimCrvStruct *TrimCrvCopy(const TrimCrvStruct *TrimCrv)
```

TrimCrv: A trimming curve to duplicate.

Returns: A trimming curve structure.

Description: Duplicates a trimming curve structure.

12.2.18 TrimCrvCopyList (trim_gen.c:323)

copy

```
TrimCrvStruct *TrimCrvCopyList(const TrimCrvStruct *TrimCrvList)
```

TrimCrvList: To be copied.

Returns: A duplicated list of trimming curves.

Description: Allocates and copies a list of trimming curve structures.

12.2.19 TrimCrvFree (trim_gen.c:352)

allocation

```
void TrimCrvFree(TrimCrvStruct *TrimCrv)
```

TrimCrv: A trimming curve to free.

Returns: void

Description: Deallocates a trimming curve structure.

12.2.20 TrimCrvFreeList (trim_gen.c:372)

allocation

```
void TrimCrvFreeList(TrimCrvStruct *TrimCrvList)
```

TrimCrvList: A list of trimming curve to free.

Returns: void

Description: Deallocates a list of trimming curve structures.

12.2.21 TrimCrvFreeListWithSubTrims (trim2ply.c:2034)

```
void TrimCrvFreeListWithSubTrims(TrimCrvStruct *TrimCrv)
```

TrimCrv: A list of trimming curves to free.

Returns: void

Description: Frees a list of trimming curve loops, which may or may not contain internal trimming curve loops. Each trimming curve in the list is freed by calling TrimCrvFreeRecursive.
See also: TrimCrvFreeWithSubTrims, TrimClassifyTrimmingLoops,

12.2.22 TrimCrvFreeWithSubTrims (trim2ply.c:2002)

```
void TrimCrvFreeWithSubTrims(TrimCrvStruct *TrimCrv)
```

TrimCrv: A trimming curve to free.

Returns: void

Description: Frees a trimming curve and all its internal trimming curve loops, recursively (by TrimCrvFreeListRecursive). The list of internal trimming curve loops are assumed to be in the _subTrims attribute.
See also: TrimCrvFreeListWithSubTrims, TrimClassifyTrimmingLoops,

12.2.23 TrimCrvListBBox (trim_aux.c:165)

bbox

bounding box

```
int TrimCrvListBBox(const TrimCrvStruct *TCrvs, int UV, CagdBBoxStruct *BBox)
```

TCrvs: To compute a bounding box for.

UV: TRUE to process the UV curve, FALSE for the Euclidean curve.

BBox: Where bounding information is to be saved.

Returns: TRUE if successful, FALSE otherwise

Description: Computes a bounding box for a list of freeform trimmed curves.
See also: TrimCrvBBox, CagdCrvBBox, GMBBSetBBoxPrecise,

12.2.24 TrimCrvNew (trim_gen.c:269)

allocation

```
TrimCrvStruct *TrimCrvNew(TrimCrvSegStruct *TrimCrvSegList)
```

TrimCrvSegList: List of trimming curve segments forming the trimming curve.

Returns: A trimmig curve.

Description: Allocates a trimming curve structure.

12.2.25 TrimCrvSegBBox (trim_aux.c:71)

bbox

bounding box

```
int TrimCrvSegBBox(const TrimCrvSegStruct *TCrvSeg,
                   int UV,
                   CagdBBoxStruct *BBox)
```

TCrvSeg: To compute a bounding box for.

UV: TRUE to process the UV curve, FALSE for the Euclidean curve.

BBox: Where bounding information is to be saved.

Returns: TRUE if successful, FALSE otherwise

Description: Computes a bounding box for a freeform trimmed curve.ffl
See also: TrimSrfBBox, TrimCrvListBBox, CagdCrvBBox, TrimCrvBBox, TrimCrvListBBox,

12.2.26 TrimCrvSegCopy (trim_gen.c:164)

allocation

```
TrimCrvSegStruct *TrimCrvSegCopy(const TrimCrvSegStruct *TrimCrvSeg)
```

TrimCrvSeg: A trimming curve segment to duplicate.

Returns: A trimming curve segment structure.

Description: Duplicates a trimming curve segment structure.

12.2.27 TrimCrvSegCopyList (trim_gen.c:194)

copy

```
TrimCrvSegStruct *TrimCrvSegCopyList(const TrimCrvSegStruct *TrimCrvSegList)
```

TrimCrvSegList: To be copied.

Returns: A duplicated list of trimming curve segments.

Description: Allocates and copies a list of trimming curve segment structures.

12.2.28 TrimCrvSegFree (trim_gen.c:223)

allocation

```
void TrimCrvSegFree(TrimCrvSegStruct *TrimCrvSeg)
```

TrimCrvSeg: A trimming curve segment to free.

Returns: void

Description: Deallocates a trimming curve segment structure.

12.2.29 TrimCrvSegFreeList (trim_gen.c:244)

allocation

```
void TrimCrvSegFreeList(TrimCrvSegStruct *TrimCrvSegList)
```

TrimCrvSegList: A list of trimming curve segments to free.

Returns: void

Description: Deallocates a list of trimming curve segment structures.

12.2.30 TrimCrvSegListBBox (trim_aux.c:108)

bbox

bounding box

```
int TrimCrvSegListBBox(const TrimCrvSegStruct *TCrvSegs,
                       int UV,
                       CagdBBoxStruct *BBox)
```

TCrvSegs: To compute a bounding box for.

UV: TRUE to process the UV curve, FALSE for the Euclidean curve.

BBox: Where bounding information is to be saved.

Returns: TRUE if successful, FALSE otherwise

Description: Computes a bounding box for a list of freeform trimmed curves.
See also: TrimCrvSegBBox, CagdCrvBBox, GMBBSetBBoxPrecise, TrimCrvBBox, , TrimCrvListBBox,

12.2.31 TrimCrvSegListReverse (trim2ply.c:1685)

```
TrimCrvSegStruct *TrimCrvSegListReverse(TrimCrvSegStruct *TSegs)
```

TSegs: Crv segments to reverse, in place.

Returns: Reversed list.

Description: Reverse all the curves in the given list of trimming curve segments, in place.
See also: TrimCrvSegReverse,

12.2.32 TrimCrvSegNew (trim_gen.c:55)

allocation

```
TrimCrvSegStruct *TrimCrvSegNew(CagdCrvStruct *UVCrv, CagdCrvStruct *EucCrv)
```

UVCrv: A UV curve. Only the E2/P2 potion of the curve is considered.

EucCrv: Optional Euclidean curve. Must be an E3 curve.

Returns: A trimming curve segment structure.

Description: Allocates a trimming curve segment structure. Allows periodic and float end conditions - converts them to open end. Input curves are used in place.

12.2.33 TrimCrvSegNewList (trim_gen.c:125)

allocation

```
TrimCrvSegStruct *TrimCrvSegNewList(CagdCrvStruct *UVCrvs,
                                    CagdCrvStruct *EucCrvs)
```

UVCrvs: A list of UV curves. Only the E2/P2 potion of the curve is considered.

EucCrvs: Optional list of Euclidean curves. Must be an E3 curve. If provided, list length must be the same as UVCrvs length.

Returns: A list of trimming curve segment structures.

Description: Allocates a list of trimming curve segment structure. Input curves are used in place.

12.2.34 TrimCrvSegReverse (trim2ply.c:1653)

```
void TrimCrvSegReverse(TrimCrvSegStruct *TSeg)
```

TSeg: Crv segment to reverse (its UV, and Euclidean curve, if exists).

Returns: void

Description: Reverse the given trimming curve segment, in place.
See also: TrimCrvSegListReverse,

12.2.35 TrimCrvTrimParamList (trim_iso.c:234)

curves

isoparametric curves

```
CagdCrvStruct *TrimCrvTrimParamList(CagdCrvStruct *Crv,
                                    TrimIsoInterStruct *InterList)
```

Crv: To trim out according to the prescribed intersections.

InterList: List of intersections, as parameters into Crv.

Returns: List of trimmed curves. May be empty (NULL).

Description: Trim Crv at the domains prescribed in the intersection list InterList Both Crv and InterList are FREED in this routine.

12.2.36 TrimCrvs2Polylines (trim_aux.c:2041)

trimming curves

```
CagdPolylineStruct *TrimCrvs2Polylines(TrimSrfStruct *TrimSrf,
                                       CagdBType ParamSpace,
                                       CagdRType TolSamples,
                                       SymbCrvApproxMethodType Method)
```

TrimSrf: To extract isoparametric curves from.

ParamSpace: TRUE for curves in parametric space, FALSE of 3D Euclidean space.

TolSamples: Tolerance of approximation error (Method = 2) or Number of samples to compute on polyline (Method = 0, 1).

Method: 0 - TolSamples are set uniformly in parametric space, 1 - TolSamples are set optimally, considering the isocurve's curvature. 2 - TolSamples sets the maximum error allowed between the piecewise linear approximation and original curve.

Returns: List of polylines representing a piecewise linear approximation of the extracted isoparamteric curves or NULL is case of an error.

Description: Routine to convert the trimming curves of a trimmed surface to polylines. Polyline are always E3 of CagdPolylineStruct type. NULL is returned in case of an error, otherwise list of CagdPolylineStruct.
See also: TrimCrv2Polyline, TrimEvalTrimCrvToEuclid,

12.2.37 TrimCrvsHierarchy2Polys (trim2ply.c:906)

polygonization

surface approximation

```
IPPolygonStruct *TrimCrvsHierarchy2Polys(TrimCrvStruct *TrimLoops)
```

TrimLoops: A linked list of trimming loops hierarchy (a 'forrest').

Returns: Piecewise linear polylines approximating the given trimming curves hierarchy.

Description: Converts a hierarchy of trimming curves/loops into closed, simple, polygons. The input trimming curves are destroyed by this function. A trimming curve inside another trimming curve are chained into one by adding a bidirection line segment between the two curves. A trimming loop will have its contained trimming loops in an attribute "_subTrims".
See also: TrimClassifyTrimmingLoops,

12.2.38 TrimDbg (trim_dbg.c:35)

debugging

```
void TrimDbg(const void *Obj)
```

Obj: A trimmed surface - to be printed to stderr.

Returns: void

Description: Prints trimmed surfaces to stderr. Should be linked to programs for debugging purposes, so trimmed surfaces may be inspected from a debugger.
See also: TrimvDbg,

12.2.39 TrimDbg1 (trim_dbg.c:67)

debugging

```
void TrimDbg1(const void *Obj)
```

Obj: A trimmed surface - to be printed to stderr.

Returns: void

Description: Prints trimmed surfaces to stderr. Should be linked to programs for debugging purposes, so trimmed surfaces may be inspected from a debugger.
See also: TrimDbg1,

12.2.40 TrimDbgTCrvSegs (trim_dbg.c:133)

debugging

```
void TrimDbgTCrvSegs(const TrimCrvSegStruct *TrimSegs)
```

TrimSegs: Trimming curve segments to print.

Returns: void

Description: Prints the trimming curves of a given trimmed surface..
See also: TrimDbg,

12.2.41 TrimDbgTCrvs (trim_dbg.c:93)

debugging

```
void TrimDbgTCrvs(const TrimCrvStruct *TrimCrv)
```

TrimCrv: Trimming curves to print.

Returns: void

Description: Prints the trimming curves of a given trimmed surface..
See also: TrimDbg,

12.2.42 TrimDbgVerifyTSeg (trim_dbg.c:161)

```
int TrimDbgVerifyTSeg(const TrimCrvSegStruct *TrimSeg)
```

TrimSeg: Trimming curve segment to verify.

Returns: TRUE if verified, FALSE otherwise.

Description: Verify the correctness of a trimming curve segment.
See also: TrimDbgVerifyUVCrv,

12.2.43 TrimDbgVerifyUVCrv (trim_dbg.c:187)

```
int TrimDbgVerifyUVCrv(const CagdCrvStruct *TrimCrv)
```

TrimCrv: Trimming curve to verify.

Returns: TRUE if verified, FALSE otherwise.

Description: Verify the correctness of a trimming curve.
See also: TrimDbgVerifyUVCrv,

12.2.44 TrimEnsureNoSingleTrimCrvLoops (trim2ply.c:1758)

```
int TrimEnsureNoSingleTrimCrvLoops(TrimCrvStruct **TrimLoops)
```

TrimLoops: Input loops, to possibly subdivided, in place.

Returns: TRUE if successful, FALSE otherwise.

Description: Divides closed trimming loops consisting of a single curve, into two. In other words, after this functions, all trimming loops will have at least two trimming curves.

12.2.45 TrimEvalTrimCrvToEuclid (trim_aux.c:2194)

```
CagdCrvStruct *TrimEvalTrimCrvToEuclid(const CagdSrfStruct *Srf,
                                       const CagdCrvStruct *UVCrv)
```

Srf: To compute the Euclidean UVCrv for.

UVCrv: A curve in the parametric space of Srf.

Returns: A Euclidean curve in Srf, following UVCrv.

Description: Computes the composed Euclidean curve of Srf(UVCrv). The resulting curve is either computed using a piecewise linear approximation or by symbolically composing it onto the surface. See TrimSetEuclidComposedFromUV for a way to control this computation.
See also: TrimCrvs2Polylines, TrimSetEuclidComposedFromUV, , TrimEvalTrimCrvToEuclid2,

12.2.46 TrimEvalTrimCrvToEuclid2 (trim_aux.c:2223)

```
CagdCrvStruct *TrimEvalTrimCrvToEuclid2(const CagdSrfStruct *Srf,
                                        const CagdCrvStruct *UVCrv,
                                        CagdCrvStruct **UVCrvLinear)
```

Srf: To compute the Euclidean UVCrv for.

UVCrv: A curve in the parametric space of TrimSrf.

UVCrvLinear: if not NULL, updated with the piecewise linear UV space curve, of the same length as the returned Euclidean curve.

Returns: A Euclidean curve in TrimSrf, following UVCrv.

Description: Computes the composed Euclidean curve of TrimSrf(UVCrv). The resulting curve is either computed using a piecewise linear approximation or by symbolically composing it onto the surface. See TrimSetEuclidComposedFromUV for a way to control this computation.
See also: TrimCrvs2Polylines, TrimSetEuclidComposedFromUV, TrimEvalTrimCrvToEuclid,

12.2.47 TrimExtendTrimmingDomain (trim_gen.c:1880)

```
CagdCrvStruct *TrimExtendTrimmingDomain(const CagdSrfStruct *Srf,
                                        CagdCrvStruct *TrimCrvs,
                                        CagdRType Extnt,
                                        CagdBType MergeCrvs)
```

Srf: Surfaces that owns these trimming curves.

TrimCrvs: Trimming curves to extend, in place.

Extnt: Extension amount.

MergeCrvs: If TRUE, aims to merge the given curves into loops, etc.

Returns: Extended trimming curves of extended domain.

Description: Given a trimming domain, extend it in all UV directions by Extnt amount interior to domain curves are not affected. Trimming curves are assuming not-merged. That is, the UMin and VMin boundary trimming curves are not merged into one curve, etc.
See also:

12.2.48 TrimFindClosestTrimCurve2UV (trim_gen.c:1733)

```
CagdRType TrimFindClosestTrimCurve2UV(TrimCrvStruct *TCrvs,
                                      const CagdUVType UV,
                                      TrimCrvSegStruct **ClosestTSeg)
```

TCrvs: Trimming curves to search for the closest location.

UV: The UV location to find a closest location on TCrvs.

ClosestTSeg: Will be updated with the closest TCrv in TCrvs.

Returns: Parameter value of the closest curve, ClosestTCrv.

Description: Find the closest location on the trimming curves TCrvs, to the given UV locations, in the parametric space.
See also:

12.2.49 TrimGetFullDomainTrimCrv (trim_gen.c:1683)

```
const TrimCrvSegStruct *TrimGetFullDomainTrimCrv(const TrimSrfStruct *TSrf)
```

TSrf: Trimmed surface to look for its outer full loop.

Returns: Return outer full loop if exists.

Description: Identifies the outer loop of the trimmed surfaces and returns a reference to it if it covers all the tensor product surface domain. A NULL is returned if outer loops is not entire tensor product domain.
See also: TrimGetLargestTrimmedSrf, TrimGetOuterTrimCrv,

12.2.50 TrimGetLargestTrimmedSrf (trim_gen.c:1583)

```
TrimSrfStruct *TrimGetLargestTrimmedSrf(TrimSrfStruct **TSrfs, int Extract)
```

TSrfs: List of trimmed usrface to find the 'largest'.

Extract: TRUE to extract and return the 'largest' trimmed surface from TSrfs, FALSE to only find it.

Returns: List of trimmed surface to search for 'largest'.

Description: Find the largest trimmed surface in the given list. While there notion of 'largest' is not really well define, we simply use here the heuristics of largest == trimmed surface with longest trimming curves.
See also: TrimGetOuterTrimCrv, TrimGetFullDomainTrimCrv,

12.2.51 TrimGetOuterTrimCrv (trim_gen.c:1636)

```
const TrimCrvSegStruct *TrimGetOuterTrimCrv(const TrimSrfStruct *TSrf)
```

TSrf: Trimmed surface to look for its outer loop.

Returns: Outer loop.

Description: Returns the outer trimming curve found in TSrf. If their is more than one outer curve, one of then is returned.
See also: TrimGetLargestTrimmedSrf, TrimGetFullDomainTrimCrv,

12.2.52 TrimGetTrimCrvLinearApprox (trim_iso.c:910)

```
CagdRType TrimGetTrimCrvLinearApprox(void)
```

Returns: Sought tolerance.

Description: Get the current tolerance used when approximating higher order trimming curves using piecewise linear approximation.
See also: TrimSetTrimCrvLinearApprox,

12.2.53 TrimGetTrimmingCurves (trim_aux.c:789)

trimming curves

```
CagdCrvStruct *TrimGetTrimmingCurves(const TrimSrfStruct *TrimSrf,
                                     CagdBType ParamSpace,
                                     CagdBType EvalEuclid)
```

TrimSrf: Trimmed surface to extract trimming curves from.

ParamSpace: TRUE for curves in parametric space, FALSE for 3D Euclidean space.

EvalEuclid: If TRUE and ParamSpace is FALSE, evaluate Euclidean curve even if one exists.

Returns: List of trimming curves of TrimSrf.

Description: Extracts the trimming curves of the given trimmed surface.
See also: TrimManageTrimmingCurvesDegrees, TrimGetTrimmingCurves2,

12.2.54 TrimGetTrimmingCurves2 (trim_aux.c:821)

trimming curves

```
CagdCrvStruct *TrimGetTrimmingCurves2(const TrimCrvStruct *TrimCrvList,
                                      const TrimSrfStruct *TrimSrf,
                                      CagdBType ParamSpace,
                                      CagdBType EvalEuclid)
```

TrimCrvList: Trimming curves to extract trimming curves as curves.

TrimSrf: Trimmed surface to extract trimming curves from. This parameter is optional and used only if EvalEuclid and/or !ParamSpace.

ParamSpace: TRUE for curves in parametric space, FALSE of 3D Euclidean space.

EvalEuclid: If TRUE and ParamSpace is FALSE, evaluate Euclidean curve even if one exists.

Returns: List of trimming curves as curves.

Description: Extracts the trimming curves as curves from the given trimming curves.
See also: TrimManageTrimmingCurvesDegrees, TrimGetTrimmingCurves,

12.2.55 TrimIntersectCrvsIsoVals (trim_iso.c:506)

isoparametric curves

```
TrimIsoInterStruct **TrimIntersectCrvsIsoVals(const CagdCrvStruct *UVCrvs,
                                              int Dir,
                                              CagdRType *IsoParams,
                                              int NumOfIsocurves)
```

UVCrvs: UV curves to intersect. Must be piecewise linear.

Dir: Either U or V.

IsoParams: Vector of isoparametric values in direction Dir.

NumOfIsocurves: Size of vector IsoParams.

Returns: A vector of size NumOfIsocurves, each slot contains a list of intersection parameter values.

Description: Computes the intersections of given UV curves with the ordered isoparametric values prescribed by IsoParams, in axis Axis.
See also: TrimIntersectTrimCrvIsoVals,

12.2.56 TrimIntersectTrimCrvIsoVals (trim_iso.c:303)

isoparametric curves

```
TrimIsoInterStruct **TrimIntersectTrimCrvIsoVals(const TrimSrfStruct *TrimSrf,
                                                 int Dir,
                                                 CagdRType *OrigIsoParams,
                                                 int NumOfIsocurves,
                                                 CagdBType Perturb)
```

TrimSrf: Trimmed surface to consider.

Dir: Either U or V.

OrigIsoParams: Vectors of isoparametric values in direction Dir.

NumOfIsocurves: Size of vector IsoParams.

Perturb: TRUE to epsilon-perturb the iso-param values.

Returns: A vector of size NumOfIsocurves, each contains a list of intersection parameter values.

Description: Computes the intersections of the trimming curves of TrimSrf with the ordered isoparametric values prescribed by OrigIsoParams, in axis Axis.
See also: TrimSrf2Polylines, TrimCrvAgainstTrimCrvs,

12.2.57 TrimIsPointInsideTrimCrvs (trim_aux.c:2523)

inclusion

```
CagdBType TrimIsPointInsideTrimCrvs(const TrimCrvStruct *TrimCrvs,
                                    CagdUVType UV)
```

TrimCrvs: Trimming curves to consider.

UV: Parametric location.

Returns: TRUE if inside, FALSE otherwise.

Description: Returns TRUE if the given UV value is inside the domain prescribed by the trimming curves.
See also: TrimIsPointInsideTrimUVCrv, TrimIsPointInsideTrimSrf, , MdlIsPointInsideTrimSrf,

12.2.58 TrimIsPointInsideTrimSrf (trim_aux.c:2498)

inclusion

```
CagdBType TrimIsPointInsideTrimSrf(const TrimSrfStruct *TrimSrf, CagdUVType UV)
```

TrimSrf: Trimmed surface to consider.

UV: Parametric location.

Returns: TRUE if inside, FALSE otherwise.

Description: Returns TRUE if the given UV value is inside the trimmed surface's parametric domain.
See also: TrimIsPointInsideTrimUVCrv, TrimIsPointInsideTrimCrvs,

12.2.59 TrimIsPointInsideTrimUVCrv (trim_aux.c:2592)

inclusion

```
int TrimIsPointInsideTrimUVCrv(const CagdCrvStruct *UVCrv, CagdUVType UV)
```

UVCrv: Trimming curve to consider.

UV: Parametric location.

Returns: Number of crossings of UVCrv by a ray from UV in -V dir.

Description: Returns the number of times a ray in the -V direction from UV crosses UVCrv.
See also: TrimIsPointInsideTrimCrvs, TrimIsPointInsideTrimSrf, , MdlIsPointInsideTrimSrf, TrimIsPointInsideTrimUVCrvs,

12.2.60 TrimIsPointInsideTrimUVCrvs (trim_aux.c:2562)

inclusion

```
int TrimIsPointInsideTrimUVCrvs(const CagdCrvStruct *UVCrvs, CagdUVType UV)
```

UVCrvs: Trimming curves to consider.

UV: Parametric location.

Returns: Number of crossings of UVCrvs by a ray from UV in -V dir.

Description: Returns the number of times a ray in the -V direction from UV crosses list of curves UVCrvs.
See also: TrimIsPointInsideTrimCrvs, TrimIsPointInsideTrimSrf, , MdlIsPointInsideTrimSrf, TrimIsPointInsideTrimUVCrv,

12.2.61 TrimLinkTrimmingCurves2Loops (trim_aux.c:1160)

trimming curves

```
TrimCrvStruct *TrimLinkTrimmingCurves2Loops(const TrimCrvStruct *TCrvs,
                                            CagdRType MaxTol,
                                            CagdBType *ClosedLoops)
```

TCrvs: Trimming curves to link into loops. Each, assumed to hold one or more trimming segments.

MaxTol: Maximal tolerance of end points comparisons.

ClosedLoops: Optional return parameter - if not NULL, Will be set to TRUE only if all linked loops are indeed closed to within tolerance.

Returns: Trimming curve segments linked into loops.

Description: Link all given curves into loops. Loops can be still open as they can start/end on the boundary, a curve someone else will have to complete... Aims to link the curves into loops from a very tight tolerance and until MaxTol or the loop was detected as closed.
See also: TrimLinkTrimmingCurves2Loops2, TrimLinkTrimmingCurves2Loops1, , TrimMergeTrimmingCurves2Loops,

12.2.62 TrimLinkTrimmingCurves2Loops1 (trim_aux.c:1110)

trimming curves

```
TrimCrvStruct *TrimLinkTrimmingCurves2Loops1(const TrimCrvSegStruct *TSegs,
                                             CagdRType MaxTol,
                                             CagdBType *ClosedLoops)
```

TSegs: Trimming curve segments to link into loops. Each, must hold one trimming segements.

MaxTol: Maximal tolerance of end points comparisons.

ClosedLoops: Optional return parameter - if not NULL, wWill be set to TRUE only if all linked loops are indeed closed to within tolerance.

Returns: Trimming curve segments linked into loops.

Description: Link all given curve segments into loops. Loops can be still open as they can start/end on the boundary, a curve someone else will have to complete... Aims to link the curves into loops from a very tight tolerance and until MaxTol or the loop was detected as closed.
See also: TrimLinkTrimmingCurves2Loops2, TrimMergeTrimmingCurves2Loops,

12.2.63 TrimLinkTrimmingCurves2Loops2 (trim_aux.c:1285)

trimming curves

```
TrimCrvStruct *TrimLinkTrimmingCurves2Loops2(TrimCrvStruct *TCrvs,
                                             CagdRType Tol,
                                             CagdBType *ClosedLoops)
```

TCrvs: Trimming curves to link into loops. Each can hold a list of trimming segments.

Tol: Tolerance of end points comparisons and link.

ClosedLoops: Will be set to TRUE only if all linked loops are indeed closed to within tolerance.

Returns: Trimming curve segments linked into loops.

Description: Link all given curves into loops. Loops can be still open as they can start/end on the boundary, a curve someone else will have to complete...
See also: TrimLinkTrimmingCurves2Loops, , TrimMergeTrimmingCurves2Loops, TrimLinkTrimmingCurves2Loops2Inc,

12.2.64 TrimLoopUV2Weight (trimcntr.c:1072)

```
CagdRType TrimLoopUV2Weight(const IrtRType *UV,
                            IrtRType *BndryUV,
                            CagdRType UMin,
                            CagdRType UMax,
                            CagdRType VMin,
                            CagdRType VMax,
                            CagdBType Last)
```

UV: UV boundary location To assign a weight.

BndryUV: If not NULL, updated with the closest point on the boundary to UV.

UMin, UMax, VMin, VMax: Domain of surface.

Last: Weight 0 and 4 are distinguished by UV being Last (Weight is 4) or not (Weight is 0).

Returns: Weight of UV.

Description: Computes a unique weight according to the location where the vertex meets the boundary: VMin is (0-1), UMax is (1-2), VMax is (2-3), UMin is (3-4).
See also: TrimSrfsFromContours, TrimSrfsFromTrimPlsHierarchy, , TrimLoopWeight2UVToData, TrimLoopWeightRelationInside,

12.2.65 TrimLoopWeight2UVToData (trimcntr.c:1140)

```
CagdRType *TrimLoopWeight2UVToData(IrtRType Wgt,
                                   CagdRType UMin,
                                   CagdRType UMax,
                                   CagdRType VMin,
                                   CagdRType VMax,
                                   CagdUVType UV)
```

Wgt: Weight to convert to a UV boundary location.

UMin, UMax, VMin, VMax: Domain of surface.

UV: UV of Wgt.

Returns: UV of Wgt.

Description: Computes the UV value associated with the given weight on the given boundary. Weight W must be in the range [0, 4]: VMin is (0-1), UMax is (1-2), VMax is (2-3), UMin is (3-4).
See also: TrimSrfsFromContours, TrimSrfsFromTrimPlsHierarchy, , TrimLoopUV2Weight, TrimLoopWeightRelationInside,

12.2.66 TrimLoopWeightRelationInside (trimcntr.c:1032)

```
int TrimLoopWeightRelationInside(CagdRType V1,
                                 CagdRType V2,
                                 CagdRType V)
```

V1, V2: The ordered end points to verify that V is inside.

V: The vertex to examine if between V1 and V2.

Returns: TRUE if inside (between V1 and V2).

Description: Returns TRUE iff V is between V1 and V2, going counter clockwise along the boundary of the parametric domain.
See also: TrimSrfsFromContours, TrimSrfsFromTrimPlsHierarchy, TrimLoopWeight2UVToData, TrimLoopUV2Weight,

12.2.67 TrimManageTrimmingCurvesDegrees (trim_aux.c:888)

trimming curves

```
TrimSrfStruct *TrimManageTrimmingCurvesDegrees(TrimSrfStruct *TrimSrf,
                                               int FitOrder,
                                               CagdBType EvalEuclid)
```

TrimSrf: Trimmed surface to extract trimming curves from, in place.

FitOrder: The order of the newly fitted trimming curves.

EvalEuclid: If TRUE reevaluate Euclidean curve as well.

Returns: The trimmed surface, modified in place, that holds the newly created trimming curves.

Description: Manage all trimming curves of a given surface as follows:

1. If FitOrder == 2, a piecewise linear (approximation) is computed for any higher order trimming curve using globals _TrimUVCrvApproxMethod and _TrimUVCrvApproxTolSamples approximation specs.
2. If FitOrder > 2, a single polynomial of FitOrder order is fitted for all trimming curves.
3. If FitOrder < 2, nothing is done.

See also: TrimGetTrimmingCurves, TrimSetTrimCrvLinearApprox, TrimCrv2Polyline,

12.2.68 TrimMatch2ndCrvLenSpeedAs1stCrv (trim2ply.c:1225)

```
void TrimMatch2ndCrvLenSpeedAs1stCrv(CagdCrvStruct **Crv1,
                                     CagdCrvStruct **Crv2,
                                     const CagdSrfStruct *Srf1,
                                     const CagdSrfStruct *Srf2)
```

Crv1: Base curve to force Crv2 to match its length. If Crv1 is an isoparametric curve it is forced to a uniform speed. Crv1 can also be refined in this function.

Crv2: 2nd curve to change its length to follow Crv1, in place.

Srf1, Srf2: If not NULL, Srf1 to compose with Crv1 (that is assumed a UV curve in Srf1), and we project the points of Crv1 onto the E3 curve of Crv2 or (Srf2(Crv2)). This, so we establish a precise match of the parametrization.

Returns: void

Description: Change the length of the given 2nd curve to be the same as the 1st Crv, by refinement. 1st Crv, Crv1, is assumed to be longer (or equal length).

See also: MvarProjUVCrvOnE3CrvMatchSpeed,

12.2.69 TrimMergePolylines (trim2ply.c:1433)

merge

polyline

```
IPPolygonStruct *TrimMergePolylines(IPPolygonStruct *Polys, IrtRType Eps)
```

Polys: Polylines to merge, in place.

Eps: Epsilon of similarity to merge points at.

Returns: Merged as possible polylines.

Description: Merges separated polylines into longer ones, in place, as possible. Given a list of polylines, matches end points and merged as possible polylines with common end points, in place.

See also: GMMergeGeometry, MvarPolyMergePolylines, GMMergePolylines, , GMMergePolylines2,

12.2.70 TrimMergeTrimmingCurves2Loops (trim_aux.c:1713)

trimming curves

```
TrimCrvStruct *TrimMergeTrimmingCurves2Loops(const TrimCrvStruct *TrimCrvs)
```

TrimCrvs: Trimming curves to merge into loops.

Returns: Trimming curves merged into loops.

Description: Merges all given trimming curves into closed loops, in place. Only UV curves are merged and Euclidean trimming curves are purged.

See also: TrimMergeTrimmingCurves2Loops2, TrimLinkTrimmingCurves2Loops,

12.2.71 TrimMergeTrimmingCurves2Loops2 (trim_aux.c:1798)

trimming curves

```
CagdCrvStruct *TrimMergeTrimmingCurves2Loops2(CagdCrvStruct *UVCrvs,
                                              CagdRType Tol)
```

UVCrvs: Curves to merge into loops, in place.

Tol: Tolerance of end points comparisons.

Returns: Curves merged into loops.

Description: Merges all given curves into closed loops, in place.
See also: TrimMergeTrimmingCurves2Loops, TrimLinkTrimmingCurves2Loops,

12.2.72 TrimOrderTrimCrvSegsInLoop (trim2ply.c:1712)

```
TrimCrvSegStruct *TrimOrderTrimCrvSegsInLoop(TrimCrvSegStruct *TSegs)
```

TSegs: List of curve segments to reorder, in place.

Returns: Ordered list.

Description: Make sure the given loop is ordered so the end of the i'th segment is the beginning of the i+1'th segment by comapring end points.
See also:

12.2.73 TrimOrientTrimingCrvs (trim2ply.c:1810)

```
CagdBType TrimOrientTrimingCrvs(TrimSrfStruct *TSrf)
```

TSrf: Trimmed surface to properly orient its trimming curves.

Returns: TRUE if successful, FALSE otherwise.

Description: Properly orient the trimming curves of the trimmed surface so walking along them, the valid portion of the trimmed surface will be on the same side, in the parametric domain.

12.2.74 TrimPiecewiseRuledSrfApprox (tr_prisa.c:130)

layout

prisa

ruled surface approximation

```
TrimSrfStruct *TrimPiecewiseRuledSrfApprox(const TrimSrfStruct *CTSrf,
                                           CagdBType ConsistentDir,
                                           CagdRType Epsilon,
                                           CagdSrfDirType Dir)
```

CTSrf: To approximate using piecewise ruled surfaces.

ConsistentDir: Do we want parametrization to be the same as TSrf?

Epsilon: Accuracy of piecewise ruled surface approximation.

Dir: Direction of piecewise ruled surface approximation. Either U or V.

Returns: A list of trimmed ruled surfaces approximating TSrf to within Epsilon in direction Dir.

Description: Constructs a piecewise ruled surface approximation to the given trimmed surface, TSrf, in the given direction, Dir, that is close to the surface to within Epsilon. If ConsitentDir then ruled surface parametrization is set to be the same as original surface TSrf. Otherwise, ruling dir is always CAGD_CONST_V_DIR. Surface is assumed to have point types E3 or P3 only.
See also: TrimAllPrisaSrfs, TrimPrisaRuledSrf, SymbAllPrisaSrfs,

12.2.75 TrimPointInsideTrimmedCrvsToData (trim_aux.c:549)

```
CagdRType *TrimPointInsideTrimmedCrvsToData(TrimCrvStruct *TrimCrvList,
                                            const TrimSrfStruct *TSrf,
                                            CagdUVType UVRetVal)
```

TrimCrvList: To find a location inside it.

TSrf: If provided, will attempt to find a point inside the trimmed curve from the surface boundary. If NULL, an interior point to the trimming curves will be selected.

UVRetVal: A location in the parametric space of the surface that is part of the valid trimmed surface domain.

Returns: A location in the parametric space of the surface that is part of the valid trimmed surface domain. NULl is returned if failed to compute.

Description: Finds a point inside a set of trimmed crvs. Returned is a UV location

12.2.76 TrimPolylines2LinTrimCrvs (trimcntr.c:637)

```
TrimCrvStruct *TrimPolylines2LinTrimCrvs(const IPPolygonStruct *Polys)
```

Polys: Input polylines to convert to lienar bspline curves.

Returns: Linear Bspline curves representing Polys.

Description: Returns a list of linear Bspline curves constructed from given polylines.

12.2.77 TrimPrisaRuledSrf (tr_prisa.c:381)

layout

prisa

```
TrimSrfStruct *TrimPrisaRuledSrf(const TrimSrfStruct *TSrf,
                                 int SamplesPerCurve,
                                 CagdRType Space,
                                 CagdVType Offset,
                                 CagdSrfDirType Dir)
```

TSrf: A trimmed ruled surface to layout flat on the XY plane.

SamplesPerCurve: During the approximation of a ruled surface as a developable surface.

Space: Increment on Y on the offset vector, after this surface was placed in the XY plane.

Offset: A vector in the XY plane to denote the amount of translation for the flatten surface in the XY plane.

Dir: Direction of piecewise ruled surface approximation. Either U or V.

Returns: A planar trimmed surface in the XY plane approximating the falttening process of TSrf.

Description: Layout a single trimmed ruled surface, by approximating it as a set of polygons. The given trimmed ruled surface might be non-developable, in which case approximation will be of a surface with no twist. The trimmed ruled surface is assumed to be constructed using CagdRuledSrf and that the ruled direction is consistent and is always CAGD_CONST_V_DIR.
See also: TrimPiecewiseRuledSrfApprox, TrimAllPrisaSrfs, SymbAllPrisaSrfs,

12.2.78 TrimRemovEucTrimCrvs (trim_aux.c:2461)

```
void TrimRemovEucTrimCrvs(TrimSrfStruct *TSrf)
```

TSrf: A trimmed surface to remove all trimming curves from.

Returns: void

Description: Remove the Euclidean curves from the trimming curves in the given data.
See also: MdlRemovEucTrimCrvs,

12.2.79 TrimRemoveCrvSegTrimCrvSegs (trim_sub.c:854)

```
int TrimRemoveCrvSegTrimCrvSegs(TrimCrvSegStruct *TrimCrvSeg,
                                TrimCrvSegStruct **TrimCrvSegs)
```

TrimCrvSeg: Segment to delete.

TrimCrvSegs: List of trimming curve segments to delete TrimCrvSeg from.

Returns: TRUE if found and removed, FALSE otherwise.

Description: Removes but not delete the given trimming crv segment from the list of trimming curve segments pointed by TrimCrvSegs.
See also: TrimRemoveCrvSegTrimCrvs,

12.2.80 TrimRemoveCrvSegTrimCrvs (trim_sub.c:801)

```
int TrimRemoveCrvSegTrimCrvs(TrimCrvSegStruct *TrimCrvSeg,
                             TrimCrvStruct **TrimCrvs)
```

TrimCrvSeg: Segment to delete.

TrimCrvs: List of trimming curves to delete TrimCrvSeg from.

Returns: TRUE if found and removed, FALSE otherwise.

Description: Removes but not delete the given trimming crv segment from the list of trimming curves point by TrimCrvs.
See also: TrimRemoveCrvSegTrimCrvSegs,

12.2.81 TrimSetEuclidComposedFromUV (trim_aux.c:2401)

```
int TrimSetEuclidComposedFromUV(int EuclidComposedFromUV)
```

EuclidComposedFromUV: Do we want symbolic composition for Euclidean curves, or should we piecewise linear sample the UV trimming curves.

Returns: Old value of way of Euclidean curve's computation

Description: Sets the way Euclidean trimming curves are computed from parametric trimming curves. Either by symbolic composition (TRUE) or by piecewise linear approximation of trimming curves (FALSE).
See also: TrimCrvs2Polylines, TrimSetEuclidLinearFromUV,

12.2.82 TrimSetEuclidLinearFromUV (trim_aux.c:2432)

```
int TrimSetEuclidLinearFromUV(int EuclidLinearFromUV)
```

EuclidLinearFromUV: TRUE to ensure Euclidean curves evaluated are made piecewise linear.

Returns: Old value of piecewise linear requirement.

Description: Sets the way Euclidean trimming curves are computed from parametric trimming curves. TRUE to ensure output is piecewise linear.
See also: TrimCrvs2Polylines, TrimSetEuclidComposedFromUV,

12.2.83 TrimSetNumTrimVrtcsInCell (trim2pl2.c:341)

```
int TrimSetNumTrimVrtcsInCell(int NumTrimVrtcsInCell)
```

NumTrimVrtcsInCell: Number of requested trimming vertices in a cell.

Returns: Old value of way of num of cells.

Description: Sets the way trimming curves contributes to each cell in the domain by setting the number of vertices trimming curves can contribute to each such cell.

12.2.84 TrimSetTrimCrvLinearApprox (trim_iso.c:882)

```
SymbCrvApproxMethodType TrimSetTrimCrvLinearApprox(CagdRType UVTolSamples,
                                                   SymbCrvApproxMethodType UVMethod)
```

UVTolSamples: Piecewise linear approximation of high order trimming curves - number of samples per curve or tolerance.

UVMethod: Method of sampling.

Returns: Old method of curve sampling.

Description: Sets the tolerances to use when approximating higher order trimming curves using piecewise linear approximation, for intersection computation.
See also: TrimGetTrimCrvLinearApprox, SymbCrv2Polyline, TrimCrv2Polyline,

12.2.85 TrimSrf2Curves (trim_iso.c:114)

curves

isoparametric curves

```
CagdCrvStruct *TrimSrf2Curves(TrimSrfStruct *TrimSrf,
                              int NumOfIsocurves[2])
```

TrimSrf: To extract isoparametric curves from.

NumOfIsocurves: In each (U or V) direction.

Returns: List of extracted isoparametric curves. These curves inherit the order and continuity of the original Srf. NULL is returned in case of an error.

Description: Routine to extract from a trimmed surface NumOfIsoline isocurve list in each param. direction. Iso parametric curves are sampled equally spaced in parametric space. NULL is returned in case of an error, otherwise list of CagdCrvStruct. As the isoparametric curves are trimmed according to the trimming curves the resulting number of curves is arbitrary.

12.2.86 TrimSrf2KnotCurves (trim_iso.c:686)

control mesh

knots

knot lines

knot curves

```
CagdCrvStruct *TrimSrf2KnotCurves(TrimSrfStruct *TrimSrf)
```

TrimSrf: To extract a knot curves from.

Returns: Extracted knot curves.

Description: Extracts a list of knot curves along all knots in U and V of TrimSrf.
See also: CagdSrf2CtrlMesh, CagdSrf2KnotLines, CagdSrf2KnotPolylines, , CagdSrf2KnotCurves,

12.2.87 TrimSrf2Polygons2 (trim2pl2.c:183)

polygonization

surface approximation

```
IPPolygonStruct *TrimSrf2Polygons2(const TrimSrfStruct *CTrimSrf,
                                   const CagdSrf2PlsInfoStrct *TessInfo)
```

CTrimSrf: To approximate into triangles.

TessInfo: All auxiliary information/state to tessellate.

Returns: A list of polygons with optional normal and/or UV parametric information. NULL is returned in case of an error.

Description: Routine to convert a single trimmed surface to set of triangles approximating it. FineNess is a fineness control on result and the larger is more triangles may result. A value of 10 is a good start value. NULL is returned in case of an error, otherwise list of IPPolygonStruct. This routine looks for C1 discontinuities in the surface and splits it into C1 continuous patches to invoke TrimC1Srf2Polygons to gen. polygons.
See also: BspSrf2PolygonSetErrFunc, BzrSrf2Polygons, IritSurface2Polygons, , IritTrimSrf2Polygons, CagdSrf2Polygons, BspSrf2Polygons, , BspC1Srf2Polygons, TrimSetNumTrimVrtcsInCell,

12.2.88 TrimSrf2Polylines (trim_iso.c:75)

isoparametric curves

```
CagdPolylineStruct *TrimSrf2Polylines(TrimSrfStruct *TrimSrf,
                                      int NumOfIsocurves[2],
                                      CagdRType TolSamples,
                                      SymbCrvApproxMethodType Method,
                                      CagdSrf2PlsInfoStrct *TessInfo)
```

TrimSrf: To extract isoparametric curves from.

NumOfIsocurves: In each (U or V) direction.

TolSamples: Tolerance of approximation error (Method = 2) or Number of samples to compute on polyline (Method = 0, 1).

Method: 0 - TolSamples are set uniformly in parametric space, 1 - TolSamples are set optimally, considering the isocurve's curvature. 2 - TolSamples sets the maximum error allowed between the piecewise linear approximation and original curve.

TessInfo: Auxiliary struct which hold the parameters for the creation of the polys. Can be NULL for default params.

Returns: List of polylines representing a piecewise linear approximation of the extracted isoparametric curves or NULL is case of an error.

Description: Routine to convert a single trimmed surface to NumOfIsolines polylines in each parametric direction with TolSamples samples/tolerance in each isoparametric curve. Polyline are always E3 of CagdPolylineStruct type. NULL is returned in case of an error, otherwise list of CagdPolylineStruct. Attempt is made to extract isolines along C1 discontinuities first.
See also: TrimCrv2Polyline,

12.2.89 TrimSrfAdap2Polygons (trim2ply.c:144)

polygonization

surface approximation

```
IPPolygonStruct *TrimSrfAdap2Polygons(const TrimSrfStruct *TrimSrf,
                                      const CagdSrf2PlsInfoStrct *TessInfo)
```

TrimSrf: To approximate into triangles.

TessInfo: All auxiliary information/state to tessellate.

Returns: A list of polygons with optional normal and/or UV parametric information. NULL is returned in case of an error or if use of call back function to collect the polygons.

Description: Routine to convert a single trimmed surface to set of polygons approximating it. Tolerance is a tolerance control on result, typically related to the the accuracy of the approximation. A value of 0.1 is a good rough start. NULL is returned in case of an error or use of call back function to get a hold over the created polygons, otherwise list of IPPolygonStruct. This routine looks for C1 discontinuities in the surface and splits it into C1 continuous patches first.
See also: CagdSrf2PolygonSetErrFunc, CagdSrfAdap2PolyDefErrFunc, , CagdSrf2PolyAdapSetErrFunc, CagdSrf2PolyAdapSe
CagdSrf2PolyAdapSetPolyGenFunc, CagdSrfAdap2Polygons, TrimSrf2Polygons,

12.2.90 TrimSrfBBox (trim_aux.c:285)

bbox

bounding box

```
CagdBBoxStruct *TrimSrfBBox(const TrimSrfStruct *TSrf, CagdBBoxStruct *BBox)
```

TSrf: To compute a bounding box for.

BBox: Where bounding information is to be saved.

Returns: BBox.

Description: Computes a bounding box for a freeform trimmed surface.
See also: TrimSrfListBBox, CagdSrfBBox, GMBBSetBBoxPrecise,

12.2.91 TrimSrfCnvrt2BzrRglrSrf (untrim.c:192)

subdivision

```
CagdSrfStruct *TrimSrfCnvrt2BzrRglrSrf(const TrimSrfStruct *TrimSrf)
```

TrimSrf: A trimmed Bezier or a Bspline surface to convert to non-trimmed Bezier surfaces.

Returns: The non trimmed regular Bezier surfaces.

Description: Given a trimmed surface - subdivides it into trimmed Bezier surfaces and convert the trimmed surface into regular Bezier/Bspline patches. Returns a list of non-trimmed Bezier surfaces, that together, are identical geometrically to the input trimmed surface.
See also: TrimSrfSubdivAtParam, TrimSrfCnvrt2BzrTrimSrf,

12.2.92 TrimSrfCnvrt2BzrRglrSrf2 (untrim.c:293)

```
CagdSrfStruct *TrimSrfCnvrt2BzrRglrSrf2(const TrimSrfStruct *TSrf,
                                        int ComposeE3,
                                        int OnlyBzrSrfs,
                                        CagdRType Eps)
```

TSrf: Trimmed surface to decompose into a list of tensor product (and no trimming) surfaces.

ComposeE3: TRUE to compose the tiles into TSrf, FALSE to return the surface tiles in the parametric domain of TSrf.

OnlyBzrSrfs: TRUE to force only Bezier tensor products in result.

Eps: Tolerance of the decomposition.

Returns: A list of regular tensor product surfaces that represents the same region as TSrf, to within machine precision.

Description: Divides the given trimmed surface TSrf to a list of tensor products. The result is tiling the original trimmed surface to within machine precision. The given TSrf surface is recursively divided until the trimming curves are simple, in which case the trimmed surface is converted into a regular tensor product surface. A trimmed surface is considered simple if it has a single trimming loop that is double monotone with U or V (That is from the minimal point to the maximal point in U or V we have monotone progress in two separated paths).
See also: TrimCrvIsParameterizableDomain, Trim2DSrfFromDoubleMonotoneTrimCrv, TrimSrfCnvrt2BzrRglrSrf,

12.2.93 TrimSrfCnvrt2BzrTrimSrf (untrim.c:62)

subdivision

```
TrimSrfStruct *TrimSrfCnvrt2BzrTrimSrf(const TrimSrfStruct *TrimSrf)
```

TrimSrf: A Bezier or a B-spline trimmed surface to convert to Bezier.

Returns: The subdivided Bezier trimmed surfaces.

Description: Given a trimmed surface - subdivide it into trimmed Bezier surfaces (each spanning domain [0, 1]^2). Returns a list of trimmed Bezier surfaces, that together, are identical geometrically to the input trimmed surface.
See also: TrimSrfSubdivAtParam,

12.2.94 TrimSrfCnvrt2TensorProdSrf (untrim.c:358)

```
CagdSrfStruct *TrimSrfCnvrt2TensorProdSrf(const TrimSrfStruct *TSrf,
                                          int ComposeE3,
                                          CagdRType Eps)
```

TSrf: Trimmed surface to decompose into a list of tensor product (and no trimming) surfaces.

ComposeE3: TRUE to compose the tiles into TSrf, FALSE to return the surface tiles in the parametric domain of TSrf.

Eps: Tolerance of the decomposition.

Returns: A list of regular tensor product surfaces that represents the same region as TSrf, to within machine precision.

Description: Divides the given trimmed surface TSrf to a list of tensor product srfs. The result is tiling the original trimmed surface to within machine precision. The given TSrf surface is recursively divided until the trimming curves are simple, in which case the trimmed surface is converted into a regular tensor product surface. A trimmed surface is considered simple if it has a single trimming loop that is double monotone with U or V (That is from the minimal point to the maximal point in U or V we have monotone progress in two separated paths).

See also: TrimCrvIsParameterizableDomain, Trim2DSrfFromDoubleMonotoneTrimCrv, TrimSrfCnvrt2BzrRglrSrf, TrimSrfCnvrt2BzrRglrSrf2,

12.2.95 TrimSrfCopy (trim_gen.c:1227)

allocation

```
TrimSrfStruct *TrimSrfCopy(const TrimSrfStruct *TrimSrf)
```

TrimSrf: A trimming surface to duplicate.

Returns: A trimming surface structure.

Description: Duplicates a trimming surface structure.

12.2.96 TrimSrfCopyList (trim_gen.c:1255)

copy

```
TrimSrfStruct *TrimSrfCopyList(const TrimSrfStruct *TrimSrfList)
```

TrimSrfList: To be copied.

Returns: A duplicated list of trimming surfaces.

Description: Allocates and copies a list of trimming surface structures.

12.2.97 TrimSrfDegreeRaise (trim_aux.c:405)

degree raising

```
TrimSrfStruct *TrimSrfDegreeRaise(const TrimSrfStruct *TrimSrf,
                                  CagdSrfDirType Dir)
```

TrimSrf: To raise its degree.

Dir: Direction of degree raising. Either U or V.

Returns: A surface with same geometry as Srf but with one degree higher.

Description: Returns a new trimmed surface representing the same surface as TrimSrf but with its degree raised by one.

See also: CagdSrfDegreeRaise,

12.2.98 TrimSrfDomain (trim_aux.c:263)

domain

parametric domain

```
void TrimSrfDomain(const TrimSrfStruct *TrimSrf,
                   CagdRType *UMin,
                   CagdRType *UMax,
                   CagdRType *VMin,
                   CagdRType *VMax)
```

TrimSrf: To get its parametric domain.

UMin: Where to put the minimal U domain's boundary.

UMax: Where to put the maximal U domain's boundary.

VMin: Where to put the minimal V domain's boundary.

VMax: Where to put the maximal V domain's boundary.

Returns: void

Description: Returns the parametric domain of a trimmed surface.

See also: CagdSrfDomain,

12.2.99 TrimSrfEvalMalloc (trim_aux.c:347)

evaluation

```
CagdRType *TrimSrfEvalMalloc(const TrimSrfStruct *TrimSrf,
                             CagdRType u,
                             CagdRType v)
```

TrimSrf: To evaluate at the given parametric location (u, v).

u, v: The parameter values at which TrimSrf is to be evaluated.

Returns: A vector holding all the coefficients of all components of surface TrimSrf's point type. If, for example, TrimSrf's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1). In dynamic memory.

Description: Given a trimmed surface and parameter values u, v, evaluate the surface at (u, v). No test is made to make sure (u, v) is in the untrimmed domain.
See also: CagdSrfEval, TrimSrfEvalToData,

12.2.100 TrimSrfEvalToData (trim_aux.c:380)

evaluation

```
void TrimSrfEvalToData(const TrimSrfStruct *TrimSrf,
                       CagdRType u,
                       CagdRType v,
                       CagdRType *Pt)
```

TrimSrf: To evaluate at the given parametric location (u, v).

u, v: The parameter values at which TrimSrf is to be evaluated.

Pt: A vector holding all the coefficients of all components of surface TrimSrf's point type. If, for example, TrimSrf's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Returns: void

Description: Given a trimmed surface and parameter values u, v, evaluate the surface at (u, v). No test is made to make sure (u, v) is in the untrimmed domain.
See also: CagdSrfEval, TrimSrfEvalMalloc,

12.2.101 TrimSrfFree (trim_gen.c:1284)

allocation

```
void TrimSrfFree(TrimSrfStruct *TrimSrf)
```

TrimSrf: A trimmed surface to free.

Returns: void

Description: Deallocates a trimmed surface structure.

12.2.102 TrimSrfFreeEuclideanTrimCrvs (trim_aux.c:426)

```
void TrimSrfFreeEuclideanTrimCrvs(TrimSrfStruct *TrimSrf)
```

TrimSrf: To free all its Euclidean trimming curves. UV curves are left as is.

Returns: void

Description: Frees all Euclidean trimming curves of trimmed surface TrimSrf.

12.2.103 TrimSrfFreeList (trim_gen.c:1308)

allocation

```
void TrimSrfFreeList(TrimSrfStruct *TrimSrfList)
```

TrimSrfList: A list of trimmed surface to free.

Returns: void

Description: Deallocates a list of trimmed surface structures.

12.2.104 TrimSrfFromE3TrimmingCurves (trim_gen.c:570)

```
TrimSrfStruct *TrimSrfFromE3TrimmingCurves(TrimCrvStruct *TCrvs,
                                           const IrtPlnType Plane)
```

TCrvs: Trimming curves to use, with only E3 information. Assumed first trimming curve is the outer largest one. Used in place and UV curves are added on the fly.

Plane: Optional place to consider. If NULL computed from TCrvs.

Returns: Constructed planar trimmed surface.

Description: Constructs a planar trimmed surface, large enough to accomodate the TCrvs as rimming curves. The trimming curves are assumed to hold E3 curves only and are used in place.
See also: TrimSrfNew, CagdPrimPlaneFromE3Crv, MvarInverseCrvOnSrfProj,

12.2.105 TrimSrfFromSrf (trimcntr.c:85)

```
TrimSrfStruct *TrimSrfFromSrf(CagdSrfStruct *Srf, int SingleTCrv)
```

Srf: Surface to convert into a trimmed surface, in place.

SingleTCrv: TRUE a single boundary trimming e loop. or FALSE to have four boundary trimming curves (for UMin/Max, VMin/Max). * *

Returns: Constructed trimmed surface.

Description: Build a trimmed surface from the given tensor product surface Srf, that spans all the domain of Srf.
See also: TrimSrfNew,

12.2.106 TrimSrfListBBox (trim_aux.c:307)

bbox

```
CagdBBoxStruct *TrimSrfListBBox(const TrimSrfStruct *TSrfs, CagdBBoxStruct *BBox)
```

bounding box

TSrfs: To compute a bounding box for.

BBox: Where bounding information is to be saved.

Returns: BBox.

Description: Computes a bounding box for a list of freeform trimmed surfaces.
See also: TrimSrfBBox, CagdSrfBBox, GMBBSetBBoxPrecise,

12.2.107 TrimSrfListMatTransform (trim_gen.c:1479)

scaling

rotation

translation

transformations

```
TrimSrfStruct *TrimSrfListMatTransform(const TrimSrfStruct *TrimSrfs,
                                       CagdMType Mat)
```

TrimSrfs: To be transformed.

Mat: Defining the transformation.

Returns: Returned transformed surfaces.

Description: Transforms, in place, the given trimmed surface as specified by a homogeneous matrix Mat. Applies an homogeneous transformation, to the given list of surfaces Srfs as specified by homogeneous transformation Mat.
See also: TrimSrfMatTransform, TrimSrfMatTransform2,

12.2.108 TrimSrfMatTransform (trim_gen.c:1425)

Trimmed surface

```
TrimSrfStruct *TrimSrfMatTransform(const TrimSrfStruct *TrimSrf,
                                   CagdMType Mat)
```

TrimSrf: Trimmed surface to transform.

Mat: Homogeneous transformation to apply to trimmed surface.

Returns: Transformed trimmed surface.

Description: Transforms the given trimmed surface as specified by a homogeneous matrix Mat.
See also: TrimSrfListMatTransform, TrimSrfMatTransform2,

12.2.109 TrimSrfMatTransform2 (trim_gen.c:1379)

Trimmed surface

```
void TrimSrfMatTransform2(TrimSrfStruct *TrimSrf, CagdMType Mat)
```

TrimSrf: Trimmed surface to transform.

Mat: Homogeneous transformation to apply to trimmed surface.

Returns: void

Description: Transforms, in place, the given trimmed surface as specified by a homogeneous matrix Mat.
See also: TrimSrfListMatTransform, TrimSrfMatTransform,

12.2.110 TrimSrfNew (trim_gen.c:403)

allocation

```
TrimSrfStruct *TrimSrfNew(CagdSrfStruct *Srf,
                          TrimCrvStruct *TrimCrvList,
                          CagdBType HasTopLvlTrim)
```

Srf: Surface to make into a trimmed surface. Used in place.

TrimCrvList: A list of trimming curves, used in place.

HasTopLvlTrim: Do we have a top level outer most trimming curve?

Returns: The trimmed surface.

Description: Constructor for a trimmed surface.
See also: TrimSrfNew2, TrimSrfNew3,

12.2.111 TrimSrfNew2 (trim_gen.c:472)

allocation

```
TrimSrfStruct *TrimSrfNew2(CagdSrfStruct *Srf,
                           CagdCrvStruct *TrimCrvList,
                           CagdBType HasTopLvlTrim)
```

Srf: Surface to make into a trimmed surface. Used in place.

TrimCrvList: A list of trimming curves, as regular curves, used in place.

HasTopLvlTrim: Do we have a top level outer most trimming curve?

Returns: The trimmed surface.

Description: Constructor for a trimmed surface.
See also: TrimSrfNew, TrimSrfNew3,

12.2.112 TrimSrfNew3 (trim_gen.c:512)

allocation

```
TrimSrfStruct *TrimSrfNew3(CagdSrfStruct *Srf,
                           CagdCrvStruct *TrimCrvList,
                           CagdBType HasTopLvlTrim)
```

Srf: Surface to make into a trimmed surface. Used in place.

TrimCrvList: A list of trimming curves, used in place.

HasTopLvlTrim: Do we have a top level outer most trimming curve?

Returns: The trimmed surface.

Description: Constructor for a trimmed surface. Same as TrimSrfNew2 but also verify and forces all trimming curves to be in the domain of Srf.
See also: TrimSrfNew, TrimSrfNew2,

12.2.113 TrimSrfNumOfTrimCrvSegs (trim_aux.c:224)

```
int TrimSrfNumOfTrimCrvSegs(const TrimSrfStruct *TSrf)
```

TSrf: The trimmed surface to consider.

Returns: Number of trimming curve segments.

Description: Counts how many trimming curves segments this trimmed surface has.
See also: TrimSrfNumOfTrimLoops,

12.2.114 TrimSrfNumOfTrimLoops (trim_aux.c:197)

```
int TrimSrfNumOfTrimLoops(const TrimSrfStruct *TSrf)
```

TSrf: The trimmed surface to consider.

Returns: Number of trimming loops.

Description: Counts how many trimming loops this trimmed surface has.
See also: TrimSrfNumOfTrimCrvSegs,

12.2.115 TrimSrfRefineAtParams (trim_aux.c:667)

refinement

subdivision

```
TrimSrfStruct *TrimSrfRefineAtParams(const TrimSrfStruct *TrimSrf,
                                     CagdSrfDirType Dir,
                                     CagdBType Replace,
                                     CagdRType *t,
                                     int n)
```

TrimSrf: To refine.

Dir: Direction of refinement. Either U or V.

Replace: If TRUE, t holds knots in exactly the same length as the length of the knot vector of Srf and t simply replaces the knot vector.

t: Vector of knots with length of n.

n: Length of vector t.

Returns: A refined surface of TrimSrf after insertion of all the knots as specified by vector t of length n.

Description: Given a trimmed surface - refines it at the given n knots as defined by vector t. If Replace is TRUE, the values in t replaces current knot vector. Returns pointer to refined surface (Note a Bezier surface will be converted into a Bspline surface).
See also: CagdSrfRefineAtParams,

12.2.116 TrimSrfRegionFromTrimSrf (trim_aux.c:464)

regions

subdivision

```
TrimSrfStruct *TrimSrfRegionFromTrimSrf(TrimSrfStruct *TrimSrf,
                                        CagdRType t1,
                                        CagdRType t2,
                                        CagdSrfDirType Dir)
```

TrimSrf: To extract a sub-region from.

t1, t2: Parametric domain boundaries of sub-region.

Dir: Direction of region extraction. Either U or V.

Returns: Sub-region extracted from TrimSrf from t1 to t2.

Description: Given a trimmed surface - extracts a sub-region within the domain specified by t1 and t2, in the direction Dir.
See also: TrimSrfSubdivAtParam, CagdSrfRegionFromSrf,

12.2.117 TrimSrfReverse (trim_aux.c:692)

reverse

```
TrimSrfStruct *TrimSrfReverse(const TrimSrfStruct *TrimSrf)
```

TrimSrf: To be reversed.

Returns: Reversed surface of TrimSrf.

Description: Returns a new trimmed surface that is the reversed surface of TrimSrf by reversing the control mesh and the knot vector (if Bspline surface) of TrimSrf in the U direction, as well as its trimming curves. See also CagdSrfReverse and BspKnotReverse.
See also: TrimSrfReverse2, CagdSrfReverse,

12.2.118 TrimSrfReverse2 (trim_aux.c:739)

reverse

```
TrimSrfStruct *TrimSrfReverse2(const TrimSrfStruct *TrimSrf)
```

TrimSrf: To be reversed.

Returns: Reversed surface of TrimSrf.

Description: Returns a new trimmed surface that is the reversed surface of Srf by flipping the U and the V directions of the surface, as well as flipping them in the trimming curves. See also BspKnotReverse.
See also: TrimSrfReverse, CagdSrfReverse2,

12.2.119 TrimSrfSetStateTrimCrvsManagement (trim_sub.c:76)

```
int TrimSrfSetStateTrimCrvsManagement(int TrimmingFitOrder)
```

TrimmingFitOrder: Order to fit trimming curves as defined above.

Returns: Old state of this flag.

Description: Controls the way future trimmed surfaces subdivisions are made: If <2, trimming curves are left as is. If 2, all trimming curves are made piecewise linear as a side effect. If 2>, each trimming curve is fitted using a single polynomial of that degree.
See also:

12.2.120 TrimSrfSubdivAtInnerLoops (trim_sub.c:1186)

```
TrimSrfStruct *TrimSrfSubdivAtInnerLoops(TrimSrfStruct *TSrf)
```

TSrf: The trimmed surface to subdivide.

Returns: A list of trimmed surfaces with no internal trimming loops.

Description: Subdivides a trimmed surface into sub-surfaces that have no interal trimming loops. This function works by trying to detect a single internal trimming loop, and subdivide the entire trimmed surface at the middle of the internal loop's bounding box. This step typically removes at least one internal trimming loop. The function then calls itself recursively, until none of the sub-surfaces have any internal trimming loops.
See also: TrimRemoveCrvSegTrimCrvs, TrimSrfSubdivValAtInnerLoop,

12.2.121 TrimSrfSubdivAtParam (trim_sub.c:114)

subdivision

```
TrimSrfStruct *TrimSrfSubdivAtParam(const TrimSrfStruct *TSrf,
                                    CagdRType t,
                                    CagdSrfDirType Dir)
```

TSrf: To subdivide at the prescribed parameter value t.

t: The parameter to subdivide the curve Crv at.

Dir: Direction of subdivision. Either U or V.

Returns: The subdivided surfaces. Usually two, but can have only one, if other is totally trimmed away.

Description: Given a trimmed surface - subdivides it into two sub-surfaces at given parametric value t in the given direction Dir. Returns pointer to a list of two trimmed surfaces, at most. It can very well may happen that the subdivided surface is completely trimmed out and hence nothing is returned for it.
See also: TrimSrfSubdivTrimmingCrvs,

12.2.122 TrimSrfSubdivTrimCrvsAtInnerLoops (trim_sub.c:1243)

```
TrimCrvStruct *TrimSrfSubdivTrimCrvsAtInnerLoops(const TrimCrvStruct *TCrvs)
```

TCrvs: The trimmed curves to subdivide.

Returns: A list of trimmed curves with no internal trimming loops.

Description: Subdivides the given trimmed curves into sub-curves that have no interal trimming loops. This function works by detecting internal trimming loops, and subdividing the given trimming curves at the middle of the internal loop. This step removes at least one internal trimming loop. The function then calls itself recursively, until none of the trimmings have any internal trimming loops.
See also: TrimSrfSubdivInnerLoops, TrimSrfSubdivValAtInnerLoop,

12.2.123 TrimSrfSubdivTrimmingCrvs (trim_sub.c:229)

subdivision

```
int TrimSrfSubdivTrimmingCrvs(const TrimCrvStruct *TrimCrvs,
                              CagdRType t,
                              CagdSrfDirType Dir,
                              TrimCrvStruct **TrimCrvs1,
                              TrimCrvStruct **TrimCrvs2)
```

TrimCrvs: To subdivide at the prescribed parameter value t.

t: The parameter to subdivide the curve Crv at.

Dir: Direction of subdivision. Either U or V.

TrimCrvs1: Returned first half of trimming curves, $< t$. Could be NULL.

TrimCrvs2: Returned second half of trimming curves, $> t$. Could be NULL.

Returns: TRUE if successful and have two halves. FALSE if failed or have only one half.

Description: Given a set of trimming curves - subdivides them into two groups below and above the subdividing line in direction Dir at parameter t.
See also: TrimSrfSubdivAtParam,

12.2.124 TrimSrfSubdivValAtInnerLoop (trim_sub.c:1294)

```
CagdSrfDirType TrimSrfSubdivValAtInnerLoop(const TrimCrvStruct *TCrvs,
                                           CagdRType *SubdivVal)
```

TCrvs: The trimmed curves to subdivide.

SubdivVal: Will be updated with the subdivision value if found one.

Returns: Subdivision direction to divide at or CAGD_NO_DIR if no internal loop was found.

Description: Given the trimmed curves, find an internal trimming loop, if any, and compute a location to divide the trimming curves (surface) so as to eliminate that internal loop.
See also: TrimSrfSubdivInnerLoops, TrimSrfSubdivTrimCrvsAtInnerLoops,

12.2.125 TrimSrfTransform (trim_gen.c:1337)

```
void TrimSrfTransform(TrimSrfStruct *TrimSrf,
                      const CagdRType *Translate,
                      CagdRType Scale)
```

TrimSrf: Trimmed surface to transform.

Translate: Translation factor. Can be NULL for non.

Scale: Scaling factor.

Returns: void

Description: Linearly transforms, in place, given trimmed surface as specified by Translate and Scale.

12.2.126 TrimSrfTrimCrvAllDomain (trim_aux.c:2771)

```
CagdBType TrimSrfTrimCrvAllDomain(const TrimSrfStruct *TrimSrf)
```

TrimSrf: Trimmed surface to examine.

Returns: TRUE if entire surface domain, FALSE otherwise.

Description: Examine the trimming curves of the given trimmed surface and returns TRUE iff the valid domain of trimming equals the entire surface domain.
See also: TrimSrfTrimCrvSquareDomain, TrimClipSrfToTrimCrvs,

12.2.127 TrimSrfTrimCrvSquareDomain (trim_aux.c:2672)

```
CagdBType TrimSrfTrimCrvSquareDomain(const TrimCrvStruct *TrimCrvList,
                                     CagdRType *UMin,
                                     CagdRType *UMax,
                                     CagdRType *VMin,
                                     CagdRType *VMax)
```

TrimCrvList: Trimming curves to examine.

UMin, UMax, VMin, VMax: Domain of square, if return TRUE

Returns: TRUE if a isoparametric square domain, FALSE otherwise.

Description: Examine the trimming curves of the given trimmed surface and returns TRUE iff the trimmed domain is a sub isoparametric square. In such a case the U/VMin/Max are set to the domain of the square.
See also: TrimSrfTrimCrvAllDomain, TrimClipSrfToTrimCrvs,

12.2.128 TrimSrfVerifyTrimCrvsValidity (trim_gen.c:664)

```
int TrimSrfVerifyTrimCrvsValidity(TrimSrfStruct *TrimSrf)
```

TrimSrf: To verify the validity of the trimming curves. This includes the verification of the continuity of the trimming loops and the inclusion in the domain of the trimming curves.

Returns: TRUE if valid, FALSE if cannot correct the trimming curves.

Description: Verify that all trimming curves are indeed in the parametric domain of the surface and that all of them matches neighboring curves.

12.2.129 TrimSrfsFromContours (trimcntr.c:169)

```
TrimSrfStruct *TrimSrfsFromContours(const CagdSrfStruct *Srf,
                                    const IPPolygonStruct *CCntrs)
```

Srf: To trim into pieces.

CCntrs: Polylines to use as separating edges.

Returns: List of trimmed surface pieces.

Description: Creates a set of trimmed surfaces as defined by given set of contours that can contain either closed or open contours. Open contours must terminate at the boundary of the parametric domain of the surface. Closed contours must be completely contained in the parametric domain with last point equals first.
See also: TrimSrfsFromContours2, TrimSrfsFromTrimPlsHierarchy, , UserDivideSrfAtInterCrvs, ,

12.2.130 TrimSrfsFromContours2 (trimcntr.c:395)

```
TrimSrfStruct *TrimSrfsFromContours2(const CagdSrfStruct *Srf,
                                     const CagdCrvStruct *CCntrs)
```

Srf: To trim into pieces.

CCntrs: Curves to use as separating trimming curves.

Returns: List of trimmed surface pieces.

Description: Same as TrimSrfsFromContours after converting the curves to polylines.
See also: TrimSrfsFromContours, TrimSrfsFromTrimPlsHierarchy,

12.2.131 TrimSrfsFromTrimPlsHierarchy (trimcntr.c:443)

```
TrimSrfStruct *TrimSrfsFromTrimPlsHierarchy(IPPolygonStruct *TopLevel,
                                            IPPolygonStruct *TrimPls,
                                            const CagdSrfStruct *Srf)
```

TopLevel: The top level outer loop or NULL if none.

TrimPls: Hierarchy of trimming polylines.

Srf: Surface to trim out.

Returns: List of trimmed surface out of the given counters.

Description: Construct trimmed surface from the given hierarchy of trimming polylines. If TopLevel is provided, it serves as the top level outer loop and both Odd and Even nested trimmed surfaces are extracted. If TopLevel is NULL only Odd nested trimmed surfaces are extracted.
See also: TrimSrfsFromContours,

12.2.132 TrimSrfsSame (trim_gen.c:1513)

```
CagdBType TrimSrfsSame(const TrimSrfStruct *TSrf1,
                       const TrimSrfStruct *TSrf2,
                       CagdRType Eps)
```

TSrf1, TSrf2: The two trimmed surfaces to compare.

Eps: Tolerance of equality.

Returns: TRUE if trimmed surfaces are the same, FALSE otehrwise.

Description: Compare the two trimmed surfaces for similarity.
See also: CagdSrfsSame,

12.2.133 TrimUntrimSetLineSweepOutputCrvPairs (untrim.c:1170)

untrimming.

```
CagdBType TrimUntrimSetLineSweepOutputCrvPairs(CagdBType NewValue)
```

NewValue: TRUE to output pairs of curves, FALSE for surfaces.

Returns: The normal at the middle of the surface patch.

Description: Configures whether the line-sweep algorithm should output surfaces or pairs of curves.
See also: CagdQuadCrvToQuadsLineSweep,

12.2.134 TrimUntrimmingResultFree (untrim.c:1057)

untrimming

```
void TrimUntrimmingResultFree(TrimUntrimResultStruct *Untrim)
```

Untrim: The untrimming result structure.

Returns: void

Description: Deallocates an untrimming result structure.

12.2.135 TrimUntrimmingResultFreeList (untrim.c:1078)

untrimming

```
void TrimUntrimmingResultFreeList(TrimUntrimResultStruct *Untrim)
```

Untrim: The list of untrimming result structures.

Returns: void

Description: Deallocates a list of untrimming result structures.

12.2.136 TrimUntrimmingResultToObj (untrim.c:1112)

untrimming

```
IPObjectStruct *TrimUntrimmingResultToObj(
                                    const TrimUntrimResultStruct *Untrimmed)
```

Untrimmed: A list of untrimming result structure.

Returns: An object containing the untrimming results (as described above).

Description: Converts a list of untrimming result structures to Irit objects. Each untrimming result is converted to a list with the following structure: if the untrimming result has untrimmed Euclidean-space surfaces, they are gathered into a list. If the untrimming result has only UV-space data (surface patches or boundary curve pairs), then the first item in the list will be the containing surface, and the second will be a list of UV-space untrimmed objects. All the untrimmed surfaces are themselves gathered into a single list, which is returned by this function.

12.2.137 TrimValidateNewTrimCntrs (trimcntr.c:333)

```
IPPolygonStruct *TrimValidateNewTrimCntrs(const CagdSrfStruct *Srf,
                                          const IPPolygonStruct *Cntrs)
```

Srf: To trim into pieces.

Cntrs: Polylines to coerce to be inside.

Returns: New set of polylines that is guaranteed to be in Srf.

Description: Make sure the given trimming contours are in the surface domain. Points on contours that are found outside are coerced to be inside.
See also: TrimSrfsFromContours, TrimSrfsFromContours2,

Chapter 13

Trivariate Library, triv_lib

13.1 General Information

This library provides a rich set of functions to manipulate freeform Bezier and/or NURBs trivariate. This library heavily depends on the cagd library. Functions are provided to create, copy, and destruct trivariates, to extract isoparametric surfaces, to evaluate, refine and subdivide, to read and write trivariates, to differentiate, degree raise, make compatible and approximate iso-surface at iso values using polygonal representations.

A trivariate has three orders, three Length prescriptions and, possibly, three knot vectors (if Bspline). In addition it contains a three dimensional volume of control points,

```
typedef struct TrivTVStruct {
    struct TrivTVStruct *Pnext;
    struct IPAttributeStruct *Attr;
    TrivGeomType GType;
    CagdPointType PType;
    int ULength, VLength, WLength;/* Mesh size in tri-variate tensor product.*/
    int UVPlane;   /* Should equal ULength * VLength for fast access. */
    int UOrder, VOrder, WOrder;       /* Order in trivariate (Bspline only). */
    CagdBType UPeriodic, VPeriodic, WPeriodic;    /* Valid only for Bspline. */
    CagdRType *Points[CAGD_MAX_PT_SIZE];     /* Pointer on each axis vector. */
    CagdRType *UKnotVector, *VKnotVector, *WKnotVector;
} TrivTVStruct;
```

The interface of the library is defined in *include/triv_lib.h*.

This library has its own error handler, which by default prints an error message and exit the program called **TrivFatalError**.

All globals in this library have a prefix of **Triv**.

13.2 Library Functions

13.2.1 IritTrivDescribeError (triv_err.c:80)

error handling

```
const char *IritTrivDescribeError(IritTrivFatalErrorType ErrorNum)
```

ErrorNum: Type of the error that was raised.

Returns: A string describing the error type.

Description: Returns a string describing the given error. Errors can be raised by any member of this triv library as well as other users. Raised error will cause an invocation of IritTrivFatalError function which decides how to handle this error. IritTrivFatalError can for example, invoke this routine with the error type, print the appropriate message and quit the program.

13.2.2 IritTrivFatalError (triv_ftl.c:56)

error handling

```
void IritTrivFatalError(IritTrivFatalErrorType ErrID)
```

ErrID: Error type that was raised.

Returns: void

Description: Trap Triv_lib errors right here. Provides a default error handler for the triv library. Gets an error description using IritTrivDescribeError, prints it and exit the program using exit.

13.2.3 IritTrivIGADescribeError (triv_iga.c:2609)

error handling

```
const char *IritTrivIGADescribeError(TrivIGAErrorType ErrorNum)
```

ErrorNum: Type of the error that was raised.

Returns: A string describing the error type.

Description: Returns a sting describing the given error.
See also: TrivIGAGetLastError, IritTrivDescribeError,

13.2.4 IritTrivSetFatalErrorFunc (triv_ftl.c:28)

error handling

```
TrivSetErrorFuncType IritTrivSetFatalErrorFunc(TrivSetErrorFuncType ErrorFunc)
```

ErrorFunc: New error function to use.

Returns: Old error function reference.

Description: Sets the error function to be used by Triv_lib.

13.2.5 MCExtractIsoSurface (mrch_run.c:299)

```
IPObjectStruct *MCExtractIsoSurface(const char *FileName,
                                    int DataType,
                                    IrtPtType CubeDim,
                                    int Width,
                                    int Height,
                                    int Depth,
                                    int SkipFactor,
                                    CagdRType IsoVal)
```

FileName: Containing the volumetric data.

DataType: Type of scalar value in volume. Can be one of: 1 - Regular float or int ascii (separated by white spaces). 2 - Two bytes short integer. 3 - Four bytes long integer. 4 - One byte (char) integer. 5 - Four bytes float. 6 - Eight bytes double.

CubeDim: Width, height, and depth of a single cube, in object space coordinates.

Width: Of volumetric data set.

Height: Of volumetric data set.

Depth: Of volumetric data set.

SkipFactor: Typically 1. For 2, only every second sample is considered and for i, only every i'th sample is considered, in all axes.

IsoVal: At which to extract the iso-surface.

Returns: A polygonal approximation of the iso-surface at IsoVal computed using the marching cubes algorithm.

Description: Extract a polygonal iso-surface out of volumetric data file.
See also: MCThresholdCube, MCExtractIsoSurface2, MCExtractIsoSurface3,

13.2.6 MCExtractIsoSurface2 (mrch_run.c:392)

```
IPObjectStruct *MCExtractIsoSurface2(const TrivTVStruct *CTV,
                                     int Axis,
                                     CagdBType TrivarNormals,
                                     IrtPtType CubeDim,
                                     int SkipFactor,
                                     CagdRType SamplingFactor,
                                     CagdRType IsoVal,
                                     int AddColors)
```

CTV: The trivariate to compute an iso surface approximation for,

Axis: Of the trivariate to handle, 1 for X, 2 for Y, etc.

TrivarNormals: If TRUE normal are computed using gradient of the trivariate, if FALSE no normal are estimated.

CubeDim: Width, height, and depth of a single cube, in object space coordinates.

SkipFactor: Typically 1. For 2, only every second sample is considered and for i, only every i'th sample is considered, in all axes.

SamplingFactor: Additional relative sampling to apply to TV. If SamplingFactor set to 1.0, the trivariate is sampled ULength * VLength * WLength. Otherwise, the samplings are (ULength * SamplingFactor) * (VLength * SamplingFactor) * (WLength * SamplingFactor).

IsoVal: At which to extract the iso-surface.

AddColors: If TRUE, the input CTV should be in E4 as (IsoVal, R, G, B) and Axis better be 1 so iso levels are sampled from the first dimension and the rest is setting the RGB colors.

Returns: A polygonal approximation of the iso-surface at IsoVal computed using the marching cubes algorithm.

Description: Extract a polygonal iso-surface out of a trivariate function.
See also: MCThresholdCube, MCExtractIsoSurface, MCExtractIsoSurface3,

13.2.7 MCExtractIsoSurface3 (mrch_run.c:515)

```
IPObjectStruct *MCExtractIsoSurface3(IPObjectStruct *ImageList,
                                     IrtPtType CubeDim,
                                     int SkipFactor,
                                     CagdRType IsoVal)
```

ImageList: List of image file names.

CubeDim: Width, height, and depth of a single cube, in object space coordinates.

SkipFactor: Typically 1. For 2, only every second sample is considered and for i, only every i'th sample is considered, in all axes.

IsoVal: At which to extract the iso-surface.

Returns: A polygonal approximation of the iso-surface at IsoVal computed using the marching cubes algorithm.

Description: Extract a polygonal iso-surface out of a stack of images.
See also: MCThresholdCube, MCExtractIsoSurface, MCExtractIsoSurface2,

13.2.8 MCExtractIsoSurface4 (mrch_run.c:594)

```
IPObjectStruct *MCExtractIsoSurface4(const void **ImageVector,
                                     const int *Size,
                                     TrivImagePixelType PixelType,
                                     IrtPtType CubeDim,
                                     int SkipFactor,
                                     CagdRType IsoVal)
```

ImageVector: Vector of images of size Size[2].

Size: Dimensions of list of images of Size[0] x Size[1]).

PixelType: Pixel type of in the images - byte, RGBA of float, etc.

CubeDim: Width, height, and depth of a single cube, in object space coordinates.

SkipFactor: Typically 1. For 2, only every second sample is considered and for i, only every i'th sample is considered, in all axes.

IsoVal: At which to extract the iso-surface.

Returns: A polygonal approximation of the iso-surface at IsoVal computed using the marching cubes algorithm.

Description: Extract a polygonal iso-surface out of a stack of images.
See also: MCThresholdCube, MCExtractIsoSurface, MCExtractIsoSurface2, , MCExtractIsoSurface3,

13.2.9 MCExtractIsoSurface5 (mrch_run.c:668)

```
IPObjectStruct *MCExtractIsoSurface5(const UserMicroRegImplctParamStruct *MRI,
                                     IrtPtType CubeDim,
                                     int SkipFactor,
                                     CagdRType SamplingFactor,
                                     CagdRType IsoVal)
```

MRI: The structure for regular implicit microstructures. The implicit function to march is defined in this structure.

CubeDim: Width, height, and depth of a single cube, in object space coordinates.

SkipFactor: Typically 1. For 2, only every second sample is considered and for i, only every i'th sample is considered, in all axes.

SamplingFactor: Samplings are (SamplingFactor) * (SamplingFactor) * (SamplingFactor).

IsoVal: At which to extract the iso-surface of the implicit function.

Returns: A polygonal approximation of the iso-surface at IsoVal computed using the marching cubes algorithm.

Description: Extract a polygonal iso-surface out of an implicit microstructure. The microstructure can be represented by a trivariate, an (reduced) expression parse tree, or a function pointer, which is defined in the regular implicit microstructure struct.
See also: MCThresholdCube, MCExtractIsoSurface2, ,

13.2.10 MCImprovePointOnIsoSrf (mrchtriv.c:192)

```
int MCImprovePointOnIsoSrf(MCImprovePointOnIsoSrfInfoStruct
                                                    *MCImprovePointOnIS,
                           IrtPtType Pt,
                           const IrtPtType CubeDim,
                           CagdRType IsoVal,
                           CagdRType Tolerance,
                           CagdRType AllowedError)
```

MCImprovePointOnIS: Pointer to struct that contains all auxiliary trivariate derivatives, for fast marching.

Pt: Position to improve.

CubeDim: Size of a single cell in the trivariate volume.

IsoVal: Of iso surface extracted from TV that Pt is approximately on.

Tolerance: Requested accuracy.

AllowedError: Maximally allowed error to be considered valid. If zero it is ignored.

Returns: TRUE if successful, FALSE otherwise.

Description: Improves a given Pt to "sit" exactly on top of the iso surface, by moving along the gradient of the trivariate. Assume TV has been preprocessed by MCImprovePointOnIsoSrfPrelude.
See also: MCImprovePointOnIsoSrfPrelude, MCImprovePointOnIsoSrfPostlude, , MCExtractIsoSurface2,

13.2.11 MCImprovePointOnIsoSrfPostlude (mrchtriv.c:86)

```
void MCImprovePointOnIsoSrfPostlude(MCImprovePointOnIsoSrfInfoStruct
                                                        *MCImprovePointOnIS)
```

MCImprovePointOnIS: Pointer to struct that contains all auxiliary trivariate derivatives, for fast marching.

Returns: void

Description: Release all allocated auxiliary trivariate derivatives, for fast marching.
See also: MCImprovePointOnIsoSrfPrelude, MCImprovePointOnIsoSrf,

13.2.12 MCImprovePointOnIsoSrfPrelude (mrchtriv.c:126)

```
MCImprovePointOnIsoSrfInfoStruct *MCImprovePointOnIsoSrfPrelude(
                                                    const TrivTVStruct *TV)
```

TV: to process and prepare for further iso surface marching.

Returns: Pointer to struct that contains all auxiliary trivariate derivatives, for fast marching. NULL if error.

Description: Prepare the necessary derivative vector fields of TV for fast marching on the trivariate, later on.
See also: MCImprovePointOnIsoSrfPostlude, MCImprovePointOnIsoSrf,

13.2.13 MCThresholdCube (mrchcube.c:94)

marching cubes

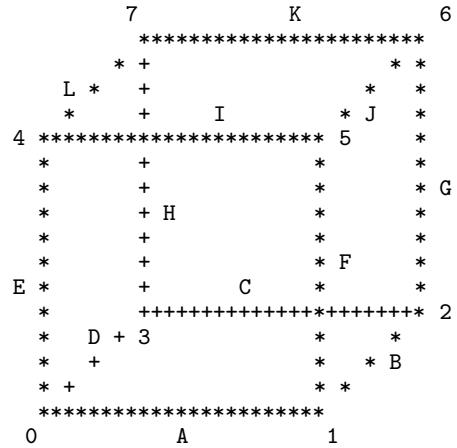
```
MCPolygonStruct *MCThresholdCube(MCCubeCornerScalarStruct *CCS,
                                 IrtRType Threshold)
```

CCS: The cube's dimensions/information.

Threshold: Iso surface level.

Returns: List of polygons (not necessarily triangles), or possibly NULL.

Description: Given 8 cube corner values (scalars), compute the polygon(s) in this cube along the isosurface at Threshold. if CCS has gradient information, it is used to approximate normals at the vertices.



Vertices 0 - 7
Edges A - L

See also: MCExtractIsoSurface, MCInitializeCube,

13.2.14 TrivAdapIsoExtractCrvs (adaptiso.c:679)

IsoSurfaces

IsoCurves

```
CagdCrvStruct *TrivAdapIsoExtractCrvs(const TrivTVStruct *Trivar,
                                      TrivTVDirType SrfDir,
                                      CagdRType Epsilon,
                                      int InitialDiv,
                                      CagdSrfDirType CrvDir,
                                      CagdRType CntrEps)
```

Trivar: The trivariate to deconstruct into curves.

SrfDir: Direction of adaptive isosurfaces extraction, as a first step for the adaptive isocurves extractions. Either U or V or W.

Epsilon: Maximum allowed distance from any point inside Trivar to some surface in the deconstruction.

InitialDiv: Number of initial divisions (regardless of Epsilon) as the depth of division tree to enforce. face equals UMax, etc.

CrvDir: Direction of curves, U, V or Both.

CntrEps: Tolerance of contouring surface geometry (computing trimming crvs).

Returns: List of adaptively extracted curves.

Description: This function returns a list of curves, adaptively extracted, corresponding to the provided isocurves extraction parameters.
See also: SymbAdapIsoExtract, TrivAdapIsoExtractSrfs,

13.2.15 TrivAdapIsoExtractSrfs (adaptiso.c:622)

IsoSurfaces

```
TrimSrfStruct *TrivAdapIsoExtractSrfs(const TrivTVStruct *Trivar,
                                      TrivTVDirType Dir,
                                      CagdRType Epsilon,
                                      int InitialDiv,
                                      CagdRType CntrEps)
```

Trivar: The trivariate to deconstruct into surfaces.

Dir: Direction of adaptive isosurfaces extraction. Either U or V or W.

Epsilon: Maximum allowed distance from any point inside Trivar to some surface in the deconstruction.

InitialDiv: Number of initial divisions (regardless of Epsilon) as the depth of division tree to enforce. Needed for cases such as periodic trivariates when UMin face equals UMax, etc.

CntrEps: Tolerance of contouring surface geometry (computing trimming crvs).

Returns: List of adaptively extracted surfaces.

Description: This function returns a list of surfaces, adaptively extracted, corresponding to the provided isosurfaces extraction parameters.
See also: SymbAdapIsoExtract, TrivAdapIsoExtractCrvs,

13.2.16 TrivAlgebraicProdTV (trivcnst.c:73)

```
TrivTVStruct *TrivAlgebraicProdTV(const CagdCrvStruct *Crv,
                                  const CagdSrfStruct *Srf)
```

Crv, Srf: A curve and a surface to multiple into a trivariate.

Returns: A trivariate represent their product, algebraically.

Description: Multiply algebraically the given curve and surface, C(r) and S(s,t). The result is a trivariate T(r, s, t) = C(r) S(s, t).
See also: TrivSwungAlgSumTV, TrivAlgebraicSumTV, SymbAlgebraicProdSrf, , SymbAlgebraicSumSrf, SymbSwungAlgSumSrf,

13.2.17 TrivAlgebraicSumTV (trivcnst.c:35)

```
TrivTVStruct *TrivAlgebraicSumTV(const CagdCrvStruct *Crv,
                                 const CagdSrfStruct *Srf)
```

Crv, Srf: A curve and a surface to add algebraically into a trivariate.

Returns: A trivariate represent their sum, algebraically.

Description: Adds up algebraically the given curve and surface, C(r) and S(s,t). The result is a trivariate T(r, s, t) = C(r) + S(s, t).
See also: TrivSwungAlgSumTV, TrivAlgebraicProdTV, SymbAlgebraicSumSrf, , SymbSwungAlgSumSrf, SymbAlgebraicProdSrf,

13.2.18 TrivBlendFilletProperties (triv_fillet.c:4288)

blending

filleting

```
void TrivBlendFilletProperties(TrivTVStruct **FilletTV,
                               const CagdSrfStruct *PrimSrf1,
                               const CagdSrfStruct *PrimSrf2)
```

FilletTV: To encode the properties into.

PrimSrf1, PrimSrf2: To blend their properties.

Returns: void

Description: Blends the material properties of the two given primary surfaes and encodes them into the given fillet trivariate. The given fillet trivariate is assumed to be constructed as a ruled volume.
See also: TrivTVFillet,

13.2.19 TrivBndryCrnrsFromTV (triveval.c:1066)

trivariates

```
CagdPtStruct *TrivBndryCrnrsFromTV(const TrivTVStruct *TV)
```

TV: To extract the eight boundary corners from.

Returns: A linked list of eight corners of trivariate TV.

Description: Extracts the eight boundary corners of the (topological cube of) given tensor product trivariate.
See also: TrivSrfFromTV, TrivBndryEdgesFromTV, TrivBndrySrfsFromTV,

13.2.20 TrivBndryEdgesFromTV (triveval.c:1007)

trivariates

```
CagdCrvStruct *TrivBndryEdgesFromTV(const TrivTVStruct *TV)
```

TV: To extract the twelve boundary curves from.

Returns: A linked list of twelve curves, representing the twelve boundary edges of trivariate TV.

Description: Extracts the twelve boundary curves of the (topological cube of) given tensor product trivariate.
See also: TrivSrfFromTV, TrivBndryEdgesFromTV, TrivBndryCrnrsFromTV,

13.2.21 TrivBndrySrfsFromTVToData (triveval.c:847)

trivariates

```
CagdSrfStruct **TrivBndrySrfsFromTVToData(const TrivTVStruct *TV,
                                          int OrientBoundary,
                                          int FilterSelfSimilar,
                                          CagdSrfStruct **Srfs)
```

TV: To extract the six boundary surfaces from.

OrientBoundary: Controls the orientations of extracted surfaces as, 0 - do no aim to examine/change the surfaces orientations. 1 - to reorient boundary surfaces to point with their normals into the trivariate. 2 - same as 1, but only tags the extract surface as "reversed" attribute, without reversing the surface.

FilterSelfSimilar: TRUE to filter self similar bndry surfaces (if TV is closed).

Srfs: A pointer to a vector of six surface pointers, representing the six boundaries of the trivariate TV in order of UMin, UMax, VMin, VMax, WMin, WMax.

Returns: A pointer to a vector of six surface pointers, representing the six boundaries of the trivariate TV in order of UMin, UMax, VMin, VMax, WMin, WMax.

Description: Extracts the six boundary surfaces of the given tensor product trivariate.
See also: TrivSrfFromTV, TrivBndryEdgesFromTV, TrivBndryCrnrsFromTV, TrivBndrySrfsFromTVs,

13.2.22 TrivBndrySrfsFromTVs (triveval.c:956)

```
CagdSrfStruct *TrivBndrySrfsFromTVs(const TrivTVStruct *Trivars,
                                    CagdRType Eps,
                                    int OrientBoundary,
                                    int FilterSelfSimilar,
                                    int FilterDupSrfs)
```

Trivars: The trivariate list.

Eps: Epsilon value for duplicate determination. Set to 0 (or negative) to disable duplicate removal. If positive, similar surfaces, up to Eps will be considered identical and will be purged or marked from the output, depending on FilterDupSrfs.

OrientBoundary: Controls the orientations of extracted surfaces as, 0 - do no aim to examine/change the surfaces orientations. 1 - to reorient boundary surfaces to point with their normals into the trivariate. 2 - same as 1, but only tags the extract surface as "reversed" attribute, without reversing the surface.

FilterSelfSimilar: TRUE to filter similar boundary surfaces, in same TV - if TV is closed and two boundaries are shared.

FilterDupSrfs: If TRUE, duplicated surfaces are removed. Otherwise, the duplicated surfaces are also saved with tagging the attribute "used" to 0.

Returns: The bounding surfaces of all trivariates in the list. The surfaces saved in the order of UMin/UMax/VMin/VMax/WMin/WMax from the first trivariate to the last.

Description: Get the bounding surfaces of all trivariate in a list. Duplicated surfaces (from different adjacent trivariates) are optionally removed according to the parameters given.
See also: TrivBndrySrfsFromTVToData,

13.2.23 TrivBspPeriodicTVNew (triv_gen.c:174)

allocation

```
TrivTVStruct *TrivBspPeriodicTVNew(int ULength,
                                   int VLength,
                                   int WLength,
                                   int UOrder,
                                   int VOrder,
                                   int WOrder,
                                   CagdBType UPeriodic,
                                   CagdBType VPeriodic,
                                   CagdBType WPeriodic,
                                   CagdPointType PType)
```

ULength: Number of control points in the U direction.

VLength: Number of control points in the V direction.

WLength: Number of control points in the W direction.

UOrder: The order of the surface in the U direction.

VOrder: The order of the surface in the V direction.

WOrder: The order of the surface in the W direction.

UPeriodic: Is this surface periodic in the U direction?

VPeriodic: Is this surface periodic in the V direction?

WPeriodic: Is this surface periodic in the W direction?

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform B-spline trivariate. If UPeriodic, VPeriodic and WPeriodic are FALSE, this function is identical to TrivTVNew.

Description: Allocates the memory required for a new, possibly periodic, B-spline trivariate.
See also: BspSrfNew, BzrSrfNew, CagdSrfNew, CagdPeriodicSrfNew, TrimSrfNew, , BspPeriodicSrfNew,

13.2.24 TrivBspTVDegreeRaise (trivrais.c:229)

degree raising

```
TrivTVStruct *TrivBspTVDegreeRaise(const TrivTVStruct *TV, TrivTVDirType Dir)
```

TV: To raise it degree by one.

Dir: Direction of degree raising. Either U, V or W.

Returns: A trivariate with one degree higher in direction Dir, representing the same geometry as TV.

Description: Returns a new Bspline trivariate, identical to the original but with one degree higher, in the requested direction Dir.
See also: TrivTVBlossomDegreeRaise, TrivBzrTVDegreeRaise, TrivTVDegreeRaise,

13.2.25 TrivBspTVDerive (triv_der.c:233)

trivariates

```
TrivTVStruct *TrivBspTVDerive(const TrivTVStruct *TV,
                              TrivTVDirType Dir,
                              CagdBType DeriveScalar)
```

TV: Trivariate to differentiate.

Dir: Direction of differentiation. Either U or V or W.

DeriveScalar: TRUE to differentiate each channel independently. FALSE to treat geometry as a rational differentiation.

Returns: Differentiated trivariate in direction Dir. A Bspline trivariate.

Description: Given a B-spline trivariate, computes its partial derivative trivariate in direction Dir. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one then:

```
Q(i) = (k - 1) * (P(i+1) - P(i)) / (Kv(i + k) - Kv(i + 1)), i = 0 to k-2.
```

See also: TrivBzrTVDerive, TrivBspTVDeriveScalar, TrivTVDerive,

13.2.26 TrivBspTVDeriveScalar (triv_der.c:382)

trivariates

```
TrivTVStruct *TrivBspTVDeriveScalar(const TrivTVStruct *TV, TrivTVDirType Dir)
```

TV: Trivariate to differentiate.

Dir: Direction of differentiation. Either U or V or W.

Returns: Differentiated trivariate in direction Dir. A Bspline trivariate.

Description: Given a B-spline trivariate, computes its partial derivative trivariate in direction Dir, each dimension on its own including the weights, if any. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one then:

```
Q(i) = (k - 1) * (P(i+1) - P(i)) / (Kv(i + k) - Kv(i + 1)), i = 0 to k-2.
```

See also: TrivBspTVDerive, TrivBzrTVDeriveScalar, TrivTVDeriveScalar,

13.2.27 TrivBspTVHasBezierKVs (triv_gen.c:1133)

conversion

```
CagdBType TrivBspTVHasBezierKVs(const TrivTVStruct *TV)
```

TV: To check for KVs that mimics Bezier polynomial surface.

Returns: TRUE if same as Bezier trivar, FALSE otherwise.

Description: Returns TRUE iff the given trivar has no interior knot open end KVs.

13.2.28 TrivBspTVHasOpenEC (triv_gen.c:1154)

open end conditions

```
CagdBType TrivBspTVHasOpenEC(const TrivTVStruct *TV)
```

TV: To check for open end conditions.

Returns: TRUE, if trivar has open end conditions, FALSE otherwise.

Description: Returns TRUE iff the given B-spline trivar has open end conditions.

13.2.29 TrivBspTVHasOpenECDir (triv_gen.c:1180)

open end conditions

```
CagdBType TrivBspTVHasOpenECDir(const TrivTVStruct *TV, TrivTVDirType Dir)
```

TV: To check for open end conditions.

Dir: Either the U, V or the W parametric direction.

Returns: TRUE, if trivariate has open end conditions, FALSE otherwise.

Description: Returns TRUE iff the given B-spline trivariate has open end conditions in the specified direction.

13.2.30 TrivBspTVKnotInsertNDiff (triv_ref.c:87)

trivariates

```
TrivTVStruct *TrivBspTVKnotInsertNDiff(const TrivTVStruct *TV,
                                       TrivTVDirType Dir,
                                       int Replace,
                                       const CagdRType *t,
                                       int n)
```

TV: Trivariate to refine according to t in direction Dir.

Dir: Direction of refinement. Either U or V or W.

Replace: If TRUE t is a knot vector exaclt in the length of the knot vector in direction Dir in TV and t simply replaces than knot vector. If FALSE, the knot vector in direction Dir in TV is refined by adding all the knots in t.

t: Knot vector to refine/replace the knot vector of TV in direction Dir.

n: Length of vector t.

Returns: The refined trivariate. A Bspline trivariate.

Description: Given a Bspline trivariate, inserts n knots with different values as defined by t. If, however, Replace is TRUE, the knot are simply replacing the current knot vector in the prescribed direction.

13.2.31 TrivBspTVNew (triv_gen.c:111)

trivariates

allocation

```
TrivTVStruct *TrivBspTVNew(int ULength,
                           int VLength,
                           int WLength,
                           int UOrder,
                           int VOrder,
                           int WOrder,
                           CagdPointType PType)
```

ULength: Number of control points in the U direction.

VLength: Number of control points in the V direction.

WLength: Number of control points in the W direction.

UOrder: Order of trivariate in the U direction.

VOrder: Order of trivariate in the V direction.

WOrder: Order of trivariate in the W direction.

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform trivariate B-spline.

Description: Allocates the memory required for a new B-spline trivariate.
See also: TrivBzrTVNew, TrivTVNew, TrivPwrTVNew,

13.2.32 TrivBzrComposeTVCrv (compost3.c:651)

composition

```
CagdCrvStruct *TrivBzrComposeTVCrv(const TrivTVStruct *TV,
                                   const CagdCrvStruct *Crv)
```

TV, Crv: The trivar and curve to compose. TV must be Bezier.

Returns: The resulting composition.

Description: Given curve Crv and Bezier trivariate TV, computes their composition TV(Crv(t)), Crv(t) = (u(t), v(t), w(t)). Crv must be a three dimensional curve completely contained in the parametric domain of TV. Compute the compositions by the products of:

```
TV(u, v, w) = TV(u(t), v(t), w(t))
```

$$= \sum_{i=0}^{n} \sum_{j=0}^{m} \sum_{k=0}^{l} P_{ijk}\, B_i(u(t))\, B_j(v(t))\, B_k(w(t))$$

$$= \sum_{i=0}^{n} \sum_{j=0}^{m} \sum_{k=0}^{l} P_{ijk} \binom{n}{i} (u(t))^{i} (1 - u(t))^{n-i} \binom{m}{j} (v(t))^{j} (1 - v(t))^{m-j} \binom{l}{k} (w(t))^{k} (1 - w(t))^{l-k}$$

See also: SymbComposeTVCrv, BzrComposeCrvCrv, TrivBzrComposeTVSrf,

13.2.33 TrivBzrComposeTVSrf (compost3.c:1067)

composition

```
CagdSrfStruct *TrivBzrComposeTVSrf(const TrivTVStruct *TV,
                                   const CagdSrfStruct *Srf)
```

TV, Srf: The trivar and surface to compose. TV must be Bezier.

Returns: The resulting composition.

Description: Given surface Srf and Bezier trivariate TV, computes their composition TV(Srf(a, b)), Srf(a, b) = (u(a, b), v(a, b), w(a, b)). Srf must be a three dimensional surface completely contained in the parametric domain of TV. Compute the compositions by the products of:

```
TV(u, v, w) = TV(u(a, b), v(a, b), w(a, b))


          n   m   l
        = sum sum sum Pijk Bi(u(a, b)) Bj(v(a, b)) Bk(w(a, b))
          i=0 j=0 k=0


          n   m   l        n           i              n-i
        = sum sum sum Pijk ( ) (u(a, b))  (1 - u(a, b))
          i=0 j=0 k=0      i
                             m           j             m-j
                            ( ) (v(a, b))  (1 - v(a, b))
                             j
                                l           k             l-k
                               ( ) (w(a, b))  (1 - w(a, b))
                                k
```

See also: SymbComposeTVSrf, BzrComposeSrfSrf, TrivBzrComposeTVCrv,

13.2.34 TrivBzrTVDegreeRaise (trivrais.c:120)

degree raising

```
TrivTVStruct *TrivBzrTVDegreeRaise(const TrivTVStruct *TV, TrivTVDirType Dir)
```

TV: To raise it degree by one.

Dir: Direction of degree raising. Either U, V or W.

Returns: A trivariate with one degree higher in direction Dir, representing the same geometry as TV.

Description: Returns a new Bezier trivariate, identical to the original but with one degree higher, in the requested direction Dir. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one then:

```
                    i           k-i
Q(0) = P(0), Q(i) = --- P(i-1) + (---) P(i), Q(k) = P(k-1).
                    k            k
```

This is applied to all rows/cols of the trivariate.

See also: TrivTVBlossomDegreeRaise, TrivBspTVDegreeRaise, TrivTVDegreeRaise,

13.2.35 TrivBzrTVDerive (triv_der.c:106)

trivariates

```
TrivTVStruct *TrivBzrTVDerive(const TrivTVStruct *TV,
                              TrivTVDirType Dir,
                              CagdBType DeriveScalar)
```

TV: Trivariate to differentiate.

Dir: Direction of differentiation. Either U or V or W.

DeriveScalar: TRUE to differentiate each channel independently. FALSE to treat geometry as a rational differentiation.

Returns: Differentiated trivariate in direction Dir. A Bezier trivariate.

Description: Given a Bezier trivariate, computes its partial derivative trivariate in direction Dir. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one then:

```
Q(i) = (k - 1) * (P(i+1) - P(i)), i = 0 to k-2.
```

See also: TrivBzrTVDeriveScalar, TrivBspTVDerive, TrivTVDerive,

13.2.36 TrivBzrTVDeriveScalar (triv_der.c:199)

trivariates

```
TrivTVStruct *TrivBzrTVDeriveScalar(const TrivTVStruct *TV, TrivTVDirType Dir)
```

TV: Trivariate to differentiate.

Dir: Direction of differentiation. Either U or V or W.

Returns: Differentiated trivariate in direction Dir. A Bezier trivariate.

Description: Given a Bezier trivariate, computes its scalar partial derivative trivariate in direction Dir, each dimension on its own including the weights, if any. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one then:

```
Q(i) = (k - 1) * (P(i+1) - P(i)), i = 0 to k-2.
```

See also: TrivBzrTVDerive, TrivBspTVDeriveScalar, TrivTVDeriveScalar,

13.2.37 TrivBzrTVNew (triv_gen.c:229)

trivariates

allocation

```
TrivTVStruct *TrivBzrTVNew(int ULength,
                           int VLength,
                           int WLength,
                           CagdPointType PType)
```

ULength: Number of control points in the U direction.

VLength: Number of control points in the V direction.

WLength: Number of control points in the W direction.

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform trivariate Bezier.

Description: Allocates the memory required for a new Bezier trivariate.
See also: TrivTVNew, TrivBspTVNew, TrivPwrTVNew,

13.2.38 TrivCnvrtBsp2BzrTV (triv_gen.c:793)

conversion

```
TrivTVStruct *TrivCnvrtBsp2BzrTV(const TrivTVStruct *TV)
```

TV: B-spline trivar to convert to a Bezier trivar.

Returns: A list of Bezier trivars representing same geometry as TV.

Description: Convert a B-spline trivar into a set of Bezier trivars by subdiving the B-spline trivar at all its internal knots. Returned is a list of Bezier trivars.
See also: TrivCnvrtBzr2BspeTV,

13.2.39 TrivCnvrtBsp2BzrTVList (triv_gen.c:886)

Convert

```
TrivTVStruct *TrivCnvrtBsp2BzrTVList(const TrivTVStruct *TVs)
```

TVs: The trivars to convert.

Returns: A new list made up of only Bezier TVs generated from the original TVs.

Description: Convert a list of B-spline TVs into a list of Bezier TVs.
See also: TrivCnvrtBsp2BzrTV,

13.2.40 TrivCnvrtBzr2BspTV (triv_gen.c:750)

conversion

trivariate

```
TrivTVStruct *TrivCnvrtBzr2BspTV(const TrivTVStruct *TV)
```

TV: A Bezier trivariate to convert to a B-spline TV.

Returns: A B-spline trivariate representing the same geometry as the given Bezier TV.

Description: Converts a Bezier trivariate into a B-spline trivariate by adding two open end uniform knot vectors to it.
See also: TrivCnvrtBsp2BzrTV,

13.2.41 TrivCnvrtCrvToTV (trivruld.c:178)

```
TrivTVStruct *TrivCnvrtCrvToTV(const CagdCrvStruct *Crv,
                               TrivTVDirType Dir)
```

Crv: A curve to promote into a trivariate.

Dir: Direction of curve in the promotion. Either U or V or W.

Returns: The trivariate promoted from Crv.

Description: Promotes a curve to a trivariate by creating a constant trivariate in two new direction. Dir sets the direction to use for the curve (U, V, or W). The resulting trivariate is degenerate in that its speed is zero in the two new direction and hence is not regular.
See also: TrivCnvrtSrfToTV,

13.2.42 TrivCnvrtFloat2OpenTV (triv_gen.c:1058)

conversion

```
TrivTVStruct *TrivCnvrtFloat2OpenTV(const TrivTVStruct *TV)
```

TV: B-spline trivariate to convert to open end conditions.

Returns: A B-spline trivariate with open end conditions, representing the same geometry as TV.

Description: Converts a float B-spline trivariate to a B-spline trivariate with open end conditions.
See also: TrivCnvrtPeriodic2FloatTV, TrivTVOpenEnd,

13.2.43 TrivCnvrtPeriodic2FloatTV (triv_gen.c:925)

conversion

```
TrivTVStruct *TrivCnvrtPeriodic2FloatTV(const TrivTVStruct *TV)
```

TV: B-spline trivariate to convert to floating end conditions. Assume TV is either periodic or has floating end condition.

Returns: A B-spline trivariate with floating end conditions, representing the same geometry as TV.

Description: Converts a B-spline trivariate into a B-spline trivariate with floating end conditions.
See also: TrivCnvrtFloat2OpenTV,

13.2.44 TrivCnvrtSrfToTV (trivruld.c:215)

```
TrivTVStruct *TrivCnvrtSrfToTV(const CagdSrfStruct *Srf, TrivTVDirType Dir)
```

Srf: A surface to promote into a trivariate.

Dir: Direction of promotion, of constant degree. Either U or V or W.

Returns: The trivariate promoted from Srf.

Description: Promotes a surface to a trivariate by creating a constant trivariate in the new direction. Dir sets the extended direction (U, V, or W). The resulting trivariate is degenerate in that its speed is zero in the new direction and hence is not regular.
See also: TrivCnvrtCrvToTV,

13.2.45 TrivCoerceTVTo (trivcoer.c:52)

coercion

```
TrivTVStruct *TrivCoerceTVTo(const TrivTVStruct *TV, CagdPointType PType)
```

TV: To coerce to a new point type PType.

PType: New point type for TV.

Returns: A new trivariate with PType as its point type.

Description: Coerces a trivariate to point type PType.

13.2.46 TrivCoerceTVsTo (trivcoer.c:25)

coercion

```
TrivTVStruct *TrivCoerceTVsTo(const TrivTVStruct *TV, CagdPointType PType)
```

TV: To coerce to a new point type PType.

PType: New point type for TV.

Returns: New trivariates with PType as their point type.

Description: Coerces a list of trivariates to point type PType.

13.2.47 TrivComposeOneObjectInTVBzr (compost3.c:355)

```
IPObjectStruct *TrivComposeOneObjectInTVBzr(const IPObjectStruct *PObj,
                                            const TrivTVStruct *DeformTV)
```

PObj: The object to map through the trivariate. Can be a (list of) curve(s), surface(s) or trivars only.

DeformTV: The mapping/deformation Bezier function from R3 to R3. Note domain might not be [0, 1]^3 - only no internal knots.

Returns: Mapped and deformed object.

Description: Compose (deform) input object in the given Bezier trivariate. Computation is made precise, using composition operations.

See also: TrivFFDTileObjectInTV, TrivFFDCtlMeshUsingTV, TrivFFDObjectTV, TrivComposeTileObjectInTV,

13.2.48 TrivComposeTVCrv (compost3.c:523)

composition

```
CagdCrvStruct *TrivComposeTVCrv(const TrivTVStruct *TV,
                                const CagdCrvStruct *Crv)
```

TV, Crv: The trivar and curve to compose. TV must be a Bezier.

Returns: The resulting composition.

Description: Given curve Crv and trivar TV, computes the composition TV(Crv). Crv must be a three dimensional curve completely contained in the parametric domain of TV.

See also: SymbComposeCrvCrv, SymbComposeSrfCrv, SymbComposeSrfSrf, , TrivComposeTVSrf, TrivComposeTVTV,

13.2.49 TrivComposeTVSrf (compost3.c:822)

composition

```
CagdSrfStruct *TrivComposeTVSrf(const TrivTVStruct *TV,
                                const CagdSrfStruct *Srf)
```

TV, Srf: The trivar and surface to compose. TV must be a Bezier.

Returns: The resulting composition.

Description: Given surface Srf and trivar TV, computes the composition TV(Srf). Srf must be a three dimensional surface completely contained in the parametric domain of TV.

See also: SymbComposeCrvCrv, SymbComposeSrfCrv, SymbComposeSrfSrf, , TrivComposeTVTV,

13.2.50 TrivComposeTVTV (compost3.c:890)

composition

```
TrivTVStruct *TrivComposeTVTV(const TrivTVStruct *TV1,
                              const TrivTVStruct *TV2)
```

TV1, TV2: The trivariates to compose. TV1 must be a Bezier.

Returns: The resulting composition.

Description: Given two trivariates, computes the composition TV1(TV2). TV2 must be a three dimensional trivariates completely contained in the parametric domain of TV1.
See also: SymbComposeCrvCrv, SymbComposeSrfCrv, SymbComposeSrfSrf, , TrivComposeTVCrv, TrivComposeTVSrf,

13.2.51 TrivComposeTileObjectInTV (compost3.c:61)

```
IPObjectStruct *TrivComposeTileObjectInTV(const IPObjectStruct *PObj,
                                          const TrivTVStruct *DeformTV,
                                          IrtRType UTimes,
                                          IrtRType VTimes,
                                          IrtRType WTimes,
                                          int FitObj,
                                          IrtRType CropBoundaries)
```

PObj: The object to map through the trivariate. Can be a (list of) curve(s) or (trimmed) surface(s) or trivariates only.

DeformTV: The mapping/deformation function from R3 to R3.

UTimes, VTimes, WTimes: Number of times to tile the object in each axis.

FitObj: 2 to rescale PObj tile to precisely fit the domain (UTimes x VTimes x WTimes), 1 to assume PObj is in [0,1]^3 when fitting/tiling domain. 0 to apply the TV mapping directly without transformations.

CropBoundaries: If positive, crop that amount from the tile, when the tile is assumed to be in the unit cube [0, 1]^3.

Returns: (UTimes x VTimes x WTimes) mapped and deformed objects.

Description: Tile an input object, in place, (UTimes x VTimes x WTimes) in the given trivariate. Computation is made precise, using composition operations.
See also: TrivFFDTileObjectInTV, TrivFFDCtlMeshUsingTV, TrivFFDObjectTV, TrivComposeTileObjectInTVBzr,

13.2.52 TrivComposeTileObjectInTVBzr (compost3.c:197)

```
IPObjectStruct *TrivComposeTileObjectInTVBzr(const IPObjectStruct *PObj,
                                             const TrivTVStruct *DeformTV,
                                             IrtRType UTimes,
                                             IrtRType VTimes,
                                             IrtRType WTimes,
                                             int FitObj)
```

PObj: The object to map through the trivariate. Can be a (list of) curve(s), surface(s) or trivars only.

DeformTV: The mapping/deformation Bezier function from R3 to R3. Note domain might not be [0, 1]^3 - only no internal knots.

UTimes, VTimes, WTimes: Number of times to tile the object in each axis.

FitObj: 2 to rescale PObj tile to precisely fit the domain (UTimes x VTimes x WTimes), 1 to assume PObj is in [0,1]^3 when fitting/tiling domain. 0 to apply TV mapping directly without any transformations.

Returns: (UTimes x VTimes x WTimes) mapped and deformed objects.

Description: Tile an input object, in place, (UTimes x VTimes x WTimes) in the given Bezier trivariate. Computation is made precise, using composition operations.
See also: TrivFFDTileObjectInTV, TrivFFDCtlMeshUsingTV, TrivFFDObjectTV, TrivComposeTileObjectInTV,

13.2.53 TrivCopyInverseQueries (trivinvs.c:284)

```
struct TrivInverseQueryStruct *TrivCopyInverseQueries(
                                    const struct TrivInverseQueryStruct *Handle)
```

Handle: The Handle tocopy its content.

Returns: The created copy.

Description: Copy the given the inverse queries handle.
See also: TrivPrepInverseQueries, TrivInverseQuery, TrivFreeInverseQueries,

13.2.54 TrivCoverIsoSurfaceUsingStrokes (mrchtriv.c:305)

```
CagdCrvStruct *TrivCoverIsoSurfaceUsingStrokes(TrivTVStruct *CTV,
                                               int NumStrokes,
                                               int StrokeType,
                                               CagdPType MinMaxPwrLen,
                                               CagdRType StepSize,
                                               CagdRType IsoVal,
                                               CagdVType ViewDir)
```

CTV: To cover with strokes along principal curvatures of iso surface of value IsoVal.

NumStrokes: Number of strokes to distribute on the implicit surface.

StrokeType: 1 - Draw strokes along minimal principal curvature. 2 - Draw strokes along maximal principal curvature. 3 - Draw strokes along both principal curvatures. 4 - Draw strokes along constant X planes. 5 - Draw strokes along constant Y planes. 6 - Draw strokes along constant Z planes. 7 - Draw strokes along silhouette lines. 8 - Draw strokes orthogonal to silhouette lines. 9 - Draw strokes along both silhouette lines and lines orthogonal to silhouette lines. StrokesType >= 10 equals StrokeType < 10 but also emphasizes the silhouette areas setting longer edges along silhouettes.

MinMaxPwrLen: Arc length of each stroke (randomized in between). a triplet of the form (Min, Max, Power) that determines the length of each stroke as Avg = (Max + Min) / 2, Dev = (Max - Min) / 2 Length = Avg + Dev * Random(0, 1)^ Pwr

StepSize: Steps to take in the piecewise linear approximation.

IsoVal: Of iso surface of TV that coverage is to be computed for.

ViewDir: Direction of view, used for silhouette computation.

Returns: A list of curves forming the coverage or NULL if error.

Description: Computes a coverage of an iso surface at IsoVal of the trivariate TV using curves along principal curvatures.
See also: MCmprovePointOnIsoSrf, MCExtractIsoSurface2,

13.2.55 TrivDbg (triv_dbg.c:31)

debugging

```
void TrivDbg(const void *Obj)
```

Obj: A trivariate - to be printed to stderr.

Returns: void

Description: Prints trivariates stderr. Should be linked to programs for debugging purposes, so trivariates may be inspected from a debugger.
See also: TrivDbg1,

13.2.56 TrivDbg1 (triv_dbg.c:72)

debugging

```
void TrivDbg1(const void *Obj)
```

Obj: A trivariate - to be printed to stderr.

Returns: void

Description: Prints trivariates stderr. Should be linked to programs for debugging purposes, so trivariates may be inspected from a debugger.
See also: TrivDbg,

13.2.57 TrivDbgDsp (triv_dbg.c:101)

debugging

```
void TrivDbgDsp(const void *Obj)
```

Obj: A trivariate - to be displayed.

Returns: void

Description: Views trivariates in a display device. Should be linked to programs for debugging purposes, so trivariates may be inspected from the debugger.

13.2.58 TrivEditSingleTVPt (trivedit.c:37)

trivar editing

```
TrivTVStruct *TrivEditSingleTVPt(const TrivTVStruct *TV,
                                 CagdCtlPtStruct *CtlPt,
                                 int UIndex,
                                 int VIndex,
                                 int WIndex,
                                 CagdBType Write)
```

TV: Trivar to be modified/query.

CtlPt: New control point to be substituted into TV. Must carry the same PType as TV if to be written to TV.

UIndex, VIndex, WIndex: In trivar TV's control mesh to substitute/query CtlPt.

Write: If TRUE CtlPt is copied into TV, if FALSE the point is copied from TV to CtlPt.

Returns: If Write is TRUE, the new modified TV, if WRITE is FALSE, NULL.

Description: Provides the way to modify/get a single control point into/from a trivariate.

13.2.59 TrivEvalCurvature (trivcurv.c:169)

```
CagdBType TrivEvalCurvature(TrivTVCurvEvalGenInfoStruct *TrivTVCurvature,
                            CagdPType Pos,
                            CagdRType *PCurv1,
                            CagdRType *PCurv2,
                            CagdVType PDir1,
                            CagdVType PDir2)
```

TrivTVCurvature: Pointer to struct that contains all auxiliary trivariate derivatives, for curvature analysis.

Pos: Location in the parametric space of the trivariate.

PCurv1, PCurv2: The two principal curvatures computed by this function.

PDir1, PDir2: The two principal directions computed by this function.

Returns: TRUE if successful, FALSE otherwise.

Description: Evaluates the principal curvatures and principal directions of the isosurface at location Pos in trivariate that was preprocessed by the TrivEvalCurvaturePredule function. Arbitrary number of invokations of this function are possible once TrivEvalCurvaturePredule is called. Also one should invoke TrivEvalCurvaturePostlude once done to release all auxiliary allocated data structures.

See also: TrivEvalCurvaturePrelude, TrivEvalCurvaturePostlude, TrivEvalHessian, , TrivEvalGradient,

13.2.60 TrivEvalGradient (trivcurv.c:263)

```
CagdBType TrivEvalGradient(TrivTVCurvEvalGenInfoStruct *TrivTVCurvature,
                           CagdPType Pos, CagdVType Gradient)
```

TrivTVCurvature: Pointer to struct that contains all auxiliary trivariate derivatives, for curvature analysis.

Pos: Location in the parametric space of the trivariate.

Gradient: The Gradient computed by this function.

Returns: TRUE if successful, FALSE otherwise.

Description: Evaluates the Gradient of the isosurface at location Pos in trivariate that was preprocessed by the TrivEvalCurvaturePredule function. Arbitrary number of invokations of this function are possible once TrivEvalCurvaturePredule is called. Also one should invoke TrivEvalCurvaturePostlude once done to release all auxiliary allocated data structures.

See also: TrivEvalCurvaturePrelude, TrivEvalCurvaturePostlude, TrivEvalCurvature, , TrivEvalHessian,

13.2.61 TrivEvalHessian (trivcurv.c:309)

```
CagdBType TrivEvalHessian(TrivTVCurvEvalGenInfoStruct
                              *TrivTVCurvature,
                          CagdPType Pos, CagdVType Hessian[3])
```

TrivTVCurvature: Pointer to struct that contains all auxiliary trivariate derivatives, for curvature analysis.

Pos: Location in the parametric space of the trivariate.

Hessian: The Hessian computed by this function.

Returns: TRUE if successful, FALSE otherwise.

Description: Evaluates the Hessian of the iso-surface at location Pos in trivariate that was preprocessed by the TrivEvalCurvaturePredule function. Arbitrary number of invocations of this function are possible once TrivEvalCurvaturePredule is called. Also one should invoke TrivEvalCurvaturePostlude once done to release all auxiliary allocated data structures.
See also: TrivEvalCurvaturePrelude, TrivEvalCurvaturePostlude, TrivEvalCurvature, , TrivEvalHessian,

13.2.62 TrivEvalTVCurvaturePostlude (trivcurv.c:48)

```
void TrivEvalTVCurvaturePostlude(TrivTVCurvEvalGenInfoStruct *TrivTVCurvature)
```

TrivTVCurvature: Pointer to struct that contains all auxiliary trivariate derivatives, for curvature analysis.

Returns: void

Description: Release all allocated auxiliary trivariate derivatives, for curvature analysis.
See also: TrivEvalTVCurvaturePrelude, TrivEvalTVCurvature, TrivEvalHessian, , TrivEvalGradient,

13.2.63 TrivEvalTVCurvaturePrelude (trivcurv.c:91)

```
TrivTVCurvEvalGenInfoStruct *TrivEvalTVCurvaturePrelude(const TrivTVStruct *TV)
```

TV: to process and prepare for further curvature evaluations.

Returns: Pointer to struct that contains all auxiliary trivariate derivatives, for curvature analysis. NULL if error.

Description: Prepare the necessary derivative vector fields of TV for curvature processing at prescribed locations, later on.
See also: TrivEvalTVCurvaturePostlude, TrivEvalTVCurvature, TrivEvalHessian, , TrivEvalGradient,

13.2.64 TrivExtractSleeveSrf (trivswep.c:727)

```
CagdSrfStruct *TrivExtractSleeveSrf(const TrivTVStruct *TV)
```

TV: The trivariate to extract the sleeve from.

Returns: The extracted sleeve,

Description: Extract a sleeve surface out of a trivariate, typically a sweep, by extracting the UMin, VMin, UMax, VMax boundaries of the trivar and merging them into one surface. I.e. A cylinder trivariate will be come a pipe.
See also:

13.2.65 TrivExtrudeTV (trivextr.c:32)

trivariate constructors

```
TrivTVStruct *TrivExtrudeTV(const CagdSrfStruct *Srf, const CagdVecStruct *Vec)
```

Srf: To exturde in direction specified by Vec.

Vec: Direction as well as magnitude of extursion.

Returns: An extrusion trivariate volume with Orders of the original Srf order and 2 in the extrusion direction.

Description: Constructs an extrusion trivariate in the Vector direction for the given surface. Input surface can be either a Bspline or a Bezier surface and the resulting output trivariate will be of the same type.
See also: CagdExtrudeSrf, TrivExtrudeTV2,

13.2.66 TrivExtrudeTV2 (trivextr.c:243)

trivariate constructors

```
TrivTVStruct *TrivExtrudeTV2(const CagdSrfStruct *Srf,
                             const CagdCrvStruct *Crv)
```

Srf: To exturde along the curve Crv.

Crv: Curve along which to move Srf. If Crv is a line, reduces to TrivExtrudeTV.

Returns: A trivariate volume with Orders of the original Srf/Crv orders.

Description: Constructs an extrusion trivariate along Crv, of the given surface: TV(u, v, t) = Srf(u, v) + Crv(t). Input curve/surface can be either a Bspline or a Bezier surface and the resulting output trivariate will be of the same type.

See also: TrivExtrudeTV2, CagdExtrudeSrf, SymbAlgebraicSumSrf,

13.2.67 TrivFFDCtlMeshUsingTV (triv_ffd.c:115)

```
void TrivFFDCtlMeshUsingTV(CagdRType **Points,
                           int Length,
                           CagdPointType PType,
                           const TrivTVStruct *DeformTV)
```

Points: The control mesh.

Length: The length of the vector of Points.

PType: The point type of Points.

DeformTV: The deformation mapping.

Returns: void

Description: Deform the given mesh in place, using the mapping that is defined by trivariate DeformTV. Input points that are outside the domain of DeformTV are coerced to the closest boundary. Computation is approximated by mapping (control) points only.

See also: TrivFFDObjectTV,

13.2.68 TrivFFDObjectTV (triv_ffd.c:181)

```
IPObjectStruct *TrivFFDObjectTV(IPObjectStruct *PObj,
                                const TrivTVStruct *DeformTV)
```

PObj: The object to map through the trivariate, in place.

DeformTV: The mapping/deformation function from R3 to R3.

Returns: PObj, mapped/deformed object, in place.

Description: Deform an input object, in place, through the given trivariate. Computation is approximated by mapping (control) points only.

See also: TrivFFDCtlMeshUsingTV, TrivFFDTileObjectInTV,

13.2.69 TrivFFDTileCropBndries (triv_ffd.c:599)

```
int TrivFFDTileCropBndries(IPObjectStruct *BndryTiles[3][3][3],
                           const IPObjectStruct *Tile,
                           IrtHmgnMatType Mat,
                           IrtRType CropBoundaries)
```

BndryTiles: The constructed boundary tiles will be kept here.

Tile: The interior tile, assumed to be in [0, 1]^3 volume.

Mat: Matrix to apply to all created tiles.

CropBoundaries: Small positive value less than one. The amount to crop from the tile, when the tile is assumed to be in the unit cube [0, 1]^3.

Returns: TRUE if successful, FALSE otherwise.

Description: Compute the 26 boundary tiles for the given tile, closing the tile at the relevant one or more directions of +/-X, +/-Y and +/-Z.

See also: TrivFFDTileObjectInTV, TrivComposeTileObjectInTV, TrivFFDTileFreeBndries,

13.2.70 TrivFFDTileFreeBndries (triv_ffd.c:689)

```
void TrivFFDTileFreeBndries(IPObjectStruct *BndryTiles[3][3][3])
```

BndryTiles: To free.

Returns: void

Description: Free the boundary tiles created by TrivFFDTileCropBndries
See also: TrivFFDTileCropBndries,

13.2.71 TrivFFDTileObjectInTV (triv_ffd.c:418)

```
IPObjectStruct *TrivFFDTileObjectInTV(const IPObjectStruct *PObj,
                                      const TrivTVStruct *DeformTV,
                                      IrtRType UTimes,
                                      IrtRType VTimes,
                                      IrtRType WTimes,
                                      int FitObj,
                                      IrtRType CropBoundaries,
                                      IrtRType MaxEdgeLen)
```

PObj: The object to map through the trivariate, in place.

DeformTV: The mapping/deformation function from R3 to R3.

UTimes, VTimes, WTimes: Number of times to tile the object in each axis.

FitObj: 2 to rescale PTile tile to precisely fit the domain (UTimes x VTimes), 1 to assume PTile is in [0,1]^2 when fitting domain, 0 to apply no mapping at all.

CropBoundaries: If positive, crop that amount from the tile, when the tile is assumed to be in the unit cube [0, 1]^3.

MaxEdgeLen: If Positive, makes sure tiles' polygonal edge lengths are not larger than MaxEdgeLen (Tile will be refined).

Returns: (UTimes x VTimes x WTimes) mapped and deformed objects.

Description: Tile an input object, in place, (UTimes x VTimes x WTimes) in the given trivariate. Computation is approximated using (control) points mapping.
See also: TrivFFDCtlMeshUsingTV, TrivFFDObjectTV, TrivComposeTileObjectInTV, , TrivFFDTileCropBndries,

13.2.72 TrivFitTV2PolyMesh (triv_fit.c:748)

```
TrivTVStruct *TrivFitTV2PolyMesh(IPObjectStruct *MeshObj,
                                 IPObjectStruct *MedialAxisObj,
                                 IrtRType MeanErrThr,
                                 IrtRType HausdorffErrThr)
```

MeshObj: The mesh polygon object.

MedialAxisObj: The mesh medial axis, given as polygon object. It is used in the initial step to construct a simple fitted trivariate.

MeanErrThr: The distance mean error threshold.

HausdorffErrThr: The Hausdorff error threshold.

Returns: The constructed B-Spline trivariate.

Description: The method fits a polygonal mesh that is a topological tube by B-Spline trivariate, so the boundaries of the trivariate approximates the mash. Aim is made to minimize the distance between the constructed trivariate and the mesh, and keep it under the given thresholds. A skeletal (medial) axis is also expected to aid the fitting.
See also:

13.2.73 TrivFreeInverseQueries (trivinvs.c:252)

```
void TrivFreeInverseQueries(struct TrivInverseQueryStruct *Handle)
```

Handle: The Handle to free its content.

Returns: void

Description: Frees the given the inverse queries handle.
See also: TrivPrepInverseQueries, TrivInverseQuery,

13.2.74 TrivIGAAddBoundaryFace (triv_iga.c:2727)

```
int TrivIGAAddBoundaryFace(TrivIGAArrangementID ArgmntID,
                           const TrivTVStruct *TV,
                           TrivTVBndryType Boundary,
                           TrivIGANodeBoundaryType NodeBoundary,
                           const char *BoundaryAxisConditions,
                           CagdRType Value)
```

ArgmntID: A handle on the IGA arrangement to process.

TV: Trivariate of the face.

Boundary: The boundary face on the trivariate (UMin, VMax, etc.).

NodeBoundary: IGA boundary condition type.

BoundaryAxisConditions: String represents the relevant axs, eg. "xy".

Value: Value of the boundary condition.

Returns: TRUE on success and FALSE on failure.

Description: Adds a boundary condition to a face in the arrangement.
See also: TrivIGAAddBoundaryFace2,

13.2.75 TrivIGAAddBoundaryFace2 (triv_iga2.c:1202)

```
int TrivIGAAddBoundaryFace2(TrivIGAArrangementID ArgmntID,
                            const TrivTVStruct *TV,
                            TrivTVBndryType BoundaryType,
                            TrivIGANodeBoundaryType NodeBoundaryType,
                            const char *BoundaryAxisConditions,
                            CagdRType Value)
```

ArgmntID: A handle on the IGA arrangement to process.

TV: Trivariate of the face.

BoundaryType: Type of the boundary face on the trivariate.

NodeBoundaryType: IGA boundary condition type.

BoundaryAxisConditions: String represents the relevant axes, eg. "xy".

Value: Value of the boundary condition.

Returns: TRUE on success and FALSE on failure

Description: Adds a boundary condition to a face in the arrangement.
See also: TrivIGAAddBoundaryFace,

13.2.76 TrivIGAAddBoundaryFaceByPt (triv_iga.c:2814)

```
int TrivIGAAddBoundaryFaceByPt(TrivIGAArrangementID ArgmntID,
                               const TrivTVStruct *TV,
                               const CagdPType Pt,
                               TrivIGANodeBoundaryType NodeBoundary,
                               const char *BoundaryAxisConditions,
                               CagdRType Value)
```

ArgmntID: A handle on the IGA arrangement to process.

TV: Trivariate owning the face. Can be NULL in which case we search all TVs in arrangment for the closest face.

Pt: A point in space to select the closest boundary.

NodeBoundary: IGA boundary condition type.

BoundaryAxisConditions: String represents the relevant axes, eg. "xy".

Value: Value of the boundary condition.

Returns: TRUE on success and FALSE on failure.

Description: Adds a boundary condition to a face in the arrangement.
See also: TrivIGAAddBoundaryFace2, TrivIGAGetBoundaryFaceByPt,

13.2.77 TrivIGAAddBoundaryNode (triv_iga2.c:1226)

```
int TrivIGAAddBoundaryNode(TrivIGAArrangementID ArgmntID,
                           TrivIGATVID TV,
                           int CtrlPointIndex)
```

ArgmntID: A handle on the IGA arrangement to process.

TV: N.S.F.I.

CtrlPointIndex: N.S.F.I.

Returns: N.S.F.I.

Description: Function is not supported.

13.2.78 TrivIGAAddMaterial (triv_iga.c:335)

```
TrivIGAMaterialID TrivIGAAddMaterial(TrivIGAArrangementID ArgmntID,
                                     TrivIGAMaterialStruct *Material)
```

ArgmntID: A handle on the IGA arrangement to process.

Material: The material

Returns: Material ID or invalid if failed.

Description: Adds the given material to the given arrangement if not exists, and updates it otherwise.

13.2.79 TrivIGAAddTrivar (triv_iga2.c:241)

```
TrivIGATVID TrivIGAAddTrivar(TrivIGAArrangementID ArgmntID,
                             const TrivTVStruct *TV,
                             int ID)
```

ArgmntID: A handle on the IGA arrangement to process.

TV: An existing trivariate to add (a copy thereof) to arrangement.

ID: ID to use or -1 to set a new ID.

Returns: INVALID_IGA_TV_ID if error, to TV ID if successful.

Description: Copy an existing new trivariate and add it to the give arrangement.

13.2.80 TrivIGAApplyDomainAndSeeding (triv_iga2.c:140)

```
static TrivTVStruct *TrivIGAApplyDomainAndSeeding(TrivIGAArrangementID ArgmntID,
                                                  TrivTVStruct *TV)
```

ArgmntID: A handle on the IGA arrangement to process.

TV: Trivariate to apply seeding to. Used in place.

Returns: Refined TV, after the seeding was applied.

Description: Apply, in place, seeding to the given trivariate, following the global arrangement seeding specifications.

13.2.81 TrivIGAArrangementComplete (triv_iga.c:1059)

```
int TrivIGAArrangementComplete(TrivIGAArrangementID ArgmntID)
```

ArgmntID: A handle on the IGA arrangement to process.

Returns: TRUE if successful, FALSE otherwise.

Description: A function to signal the end of the initialization process - insertion of trivariates into (the fields of) the IGA arrangement.

13.2.82 TrivIGADataManagerAddTrivariate (triv_iga.c:3144)

```
TrivIGATVID TrivIGADataManagerAddTrivariate(TrivIGAArrangementID ArgmntID,
                                            TrivTVStruct *TV,
                                            int ID)
```

ArgmntID: A handle on the IGA arrangement to process.

TV: Pointer to the trivariate.

ID: if non negative, use this ID as the unique ID of TV. Otherwise, assign a unique ID here.

Returns: ID of the trivariate TRUE if successful, and INVALID_IGA_TV_ID otherwise.

Description: Allocates an ID to a given trivariate in the IGA data manager, if the trivariate already exists, its associated ID is returned.

13.2.83 TrivIGADataManagerAllocateArrangement (triv_iga.c:3016)

```
TrivIGAArrangementID TrivIGADataManagerAllocateArrangement(
                                                    TrivIGADataManager *DM)
```

DM: IGA Data manager

Returns: ID of the new arrangement in success and INVALID_ARRANGEMENT_ID in failure

Description: Allocates an IGA arrangement and associates it with new ID.

13.2.84 TrivIGADataManagerFreeArrangement (triv_iga.c:3106)

```
int TrivIGADataManagerFreeArrangement(TrivIGAArrangementID ArrngmntID)
```

ArrngmntID: The arrangement ID to deallocate

Returns: TRUE if successful, FALSE otherwise.

Description: DeAllocate given arrangement from the data manager.

13.2.85 TrivIGADataManagerGetArrangement (triv_iga.c:3045)

```
TrivIGAArrangementStruct *TrivIGADataManagerGetArrangement(
                                          TrivIGAArrangementID ArrngmntID)
```

ArrngmntID: ID of the arrangement.

Returns: Pointer to the arrangement structure if success and NULL in failure.

Description: Maps between arrangement ID and its actual structure's address.

13.2.86 TrivIGADataManagerGetArrangementID (triv_iga.c:3073)

```
TrivIGAArrangementID TrivIGADataManagerGetArrangementID(
                                            TrivIGAArrangementStruct *H)
```

H: Pointer to the arrangement structure.

Returns: The arrangement id in success and NULL in failure

Description: Maps between arrangement structure address and its ID.

13.2.87 TrivIGADataManagerGetIGATrivariate (triv_iga.c:3256)

```
TrivIGATVStruct *TrivIGADataManagerGetIGATrivariate(TrivIGATVID TVID)
```

TVID: ID of the trivariate.

Returns: Pointer to IGA trivariate structue if successfull and NULL otherwise.

Description: Maps between trivariate ID and its structure's address in a given IGA data manager.

13.2.88 TrivIGADataManagerGetTrivID (triv_iga.c:3194)

```
TrivIGATVID TrivIGADataManagerGetTrivID(const TrivTVStruct *TV)
```

TV: Pointer to the trivariate.

Returns: ID of the trivariate TRUE if successful, and TRIV_IGA_INVALID_TV_ID otherwise.

Description: Returns the ID of a given trivariate from IGA data manager.

13.2.89 TrivIGADataManagerGetTrivariate (triv_iga.c:3230)

```
TrivTVStruct *TrivIGADataManagerGetTrivariate(TrivIGATVID TVID)
```

TVID: ID of the trivariate.

Returns: Pointer to the trivariate structue if successfull and NULL otherwise.

Description: Maps between trivariate ID and its structure's address in a given IGA data manager.

13.2.90 TrivIGAExportToXML (triv_iga_xml.c:678)

```
int TrivIGAExportToXML(TrivIGAArrangementID ArgmntID,
                       const char *FileName,
                       const char *TemplateFileName)
```

ArgmntID: A handle on the IGA arrangement to process.

FileName: Output XML file name.

TemplateFileName: Template XML file.

Returns: TRUE if successful, FALSE otherwise.

Description: Saves the input IGA arrangement as Febio XML file, loads template file and updates the material, geometry and boundary sections.

13.2.91 TrivIGAExtrudeTV (triv_iga2.c:271)

```
TrivIGATVID TrivIGAExtrudeTV(TrivIGAArrangementID ArgmntID,
                             const CagdSrfStruct *Srf,
                             const IrtVecType Vec,
                             int ID)
```

ArgmntID: A handle on the IGA arrangement to process.

Srf: Surface to extrude.

Vec: The extrusion vector.

ID: ID to use or -1 to set a new ID.

Returns: INVALID_IGA_TV_ID if error, to TV ID if successful.

Description: Constructs a new trivariate and add it to the give arrangement as an extrusion of the given surface.

13.2.92 TrivIGAExtrudeTV2 (triv_iga2.c:315)

```
TrivIGATVID TrivIGAExtrudeTV2(TrivIGAArrangementID ArgmntID,
                              const CagdSrfStruct *Srf,
                              const CagdCrvStruct *Crv,
                              int ID)
```

ArgmntID: A handle on the IGA arrangement to process.

Srf: Surface to extrude.

Crv: The extrusion curve.

ID: ID to use or -1 to set a new ID.

Returns: INVALID_IGA_TV_ID if error, to TV ID if successful.

Description: Constructs a new trivariate and add it to the give arrangement as an extrusion of the given surface along a curve.

13.2.93 TrivIGAFreeArrangement (triv_iga.c:2453)

```
int TrivIGAFreeArrangement(TrivIGAArrangementID ArgmntID)
```

ArgmntID: A handle on the IGA arrangement to free.

Returns: TRUE if successful, FALSE otherwise.

Description: Free all auxiliary data structure allocated in H.

13.2.94 TrivIGAGenNeighboringConstraints (triv_iga2.c:737)

```
void TrivIGAGenNeighboringConstraints(TrivIGAArrangementID ArgmntID,
                                      void *CallbackData,
                                      TrivIGANeighboringConstraintCallBackType
                                               NeighboringConstraintCallBack)
```

ArgmntID: A handle on the IGA arrangement to process.

CallbackData: Relevant data for the callback function (such as XML information).

NeighboringConstraintCallBack: Call back function to call with the constraint as a string.

Returns: void

Description: Construct linear constraints hooking adjacent faces that do not share a common function space.

13.2.95 TrivIGAGetAllTVs (triv_iga2.c:585)

```
TrivIGATVID *TrivIGAGetAllTVs(TrivIGAArrangementID ArgmntID)
```

ArgmntID: A handle on the IGA arrangement to process.

Returns: A dynamically allocated vector of IDs, terminated with INVALID_IGA_TV_ID.

Description: Returns a list of all trivariates in the given arrangement

13.2.96 TrivIGAGetBoundaryFaceByPtToData (triv_iga.c:2930)

```
int *TrivIGAGetBoundaryFaceByPtToData(TrivIGAArrangementID ArgmntID,
                                      const TrivTVStruct *TV,
                                      const CagdPType Pt,
                                      int *ReturnedIDs)
```

ArgmntID: A handle on the IGA arrangement to process.

TV: Trivariate owning the face. Can be NULl in which case we search all TVs in arrangement.

Pt: A point in space to select the closest boundary.

ReturnedIDs: A list of two integers, (TVID, FaceID).

Returns: A list of two integers, (TVID, FaceID).

Description: Find the face closest to Pt in the arrangement or specific TV.
See also: TrivIGAAddBoundaryFace2, TrivIGAAddBoundaryFaceByPt,

13.2.97 TrivIGAGetBzrElementCtrlPts (triv_iga.c:1379)

```
TrivIGACtrlPtStruct *TrivIGAGetBzrElementCtrlPts(
                                        TrivIGAArrangementID ArgmntID,
                                        const TrivTVStruct *TV,
                                        int IndexU,
                                        int IndexV,
                                        int IndexW)
```

ArgmntID: A handle on the IGA arrangement to process.

TV: The Trivar to fetch the control points of one of its Beziers.

IndexU: The Bezier trivariate index in U in TV, starting from zero.

IndexV: The Bezier trivariate index in V in TV, starting from zero.

IndexW: The Bezier trivariate index in W in TV, starting from zero.

Returns: A vector of control points of the Bezier trivar.

Description: Returns a dynamically allocated vector of control points of the Bezier trivariate at the designated indices in TV. Assumes TV has open end conditions. Control points will be returned in order, U changing first, W last.

13.2.98 TrivIGAGetCtlPt (triv_iga2.c:1017)

```
CagdCtlPtStruct *TrivIGAGetCtlPt(TrivIGAArrangementID ArgmntID,
                                 int CtlPtID,
                                 CagdCtlPtStruct *CtlPt)
```

ArgmntID: A handle on the IGA arrangement to process.

CtlPtID: ID of control point to get.

CtlPt: Fetched control point.

Returns: Fetched control point, same as CtlPt. NULL if error.

Description: Fetches one control point from the arrangement. Not efficient!

13.2.99 TrivIGAGetCtlPtIDRange (triv_iga.c:1267)

```
int *TrivIGAGetCtlPtIDRange(TrivIGAArrangementID ArgmntID,
                            const TrivTVStruct *TV,
                            int *IDs)
```

ArgmntID: A handle on the IGA arrangement to process.

TV: The Trivar to fetch its ID's range.

IDs: Vector of maximal IDs, (Max CtlPts ID, Max Trivars ID).

Returns: Same as IDs parameter - a vector of maximal IDs, as (Max CtlPts ID, Max Trivars ID) if successful, NULL otherwise.

Description: Returns the range of control point IDs used by this given TV.

13.2.100 TrivIGAGetEdgeNeighboringTVs (triv_iga.c:2254)

```
int *TrivIGAGetEdgeNeighboringTVs(TrivIGAArrangementID ArgmntID,
                                  const TrivTVStruct *TV)
```

ArgmntID: A handle on the IGA arrangement to process.

TV: Trivar to seek its edge neighbors.

Returns: A TRIV_IGA_INVALID_TV_ID terminated vector of edge neighboring TV IDS or NULL if error.

Description: Given a TV in some field, returns the edge-neighboring trivariates of TV, if exists.
See also: TrivIGAGetFaceNeighboringTVs, TrivIGAGetVrtxNeighboringTVs,

13.2.101 TrivIGAGetFaceNeighboringTVs (triv_iga.c:2180)

```
int TrivIGAGetFaceNeighboringTVs(TrivIGAArrangementID ArgmntID,
                                 const TrivTVStruct *TV,
                                 TrivIGAAdjacencyInfoStruct *AdjInfo)
```

ArgmntID: A handle on the IGA arrangement to process.

TV: Trivar to seek its (up to six) neighbors.

AdjInfo: Already allocated vector of six entries to be updated herein. Returns neighboring info for the six face-boundary surfaces of TV, if any. Six neighbors are updated in the following order, Umin, Umax, Vmin, Vmax, Wmin, Wmax.

Returns: TRUE if successful, FALSE otherwise.

Description: Given a TV in some field, returns the (up to) six facial neighboring trivariates of TV, if exists.
Notes:

1. The adjacent TV can be the same as TV if TV is closed in some dir(s).
2. The returned adjacency structure will also state if the adjacent TV needs to be reversed in U or in V or U and V should be reversed so the boundary can be detected as adjacent.

See also: TrivIGAGetVrtxNeighboringTVs, TrivIGAGetEdgeNeighboringTVs,

13.2.102 TrivIGAGetGlblMaxIDs (triv_iga.c:1230)

```
int *TrivIGAGetGlblMaxIDs(TrivIGAArrangementID ArgmntID, int *IDs)
```

ArgmntID: A handle on the IGA arrangement to process.

IDs: Vector to be updated with of maximal IDs.

Returns: Vector of maximal IDs same as IDs parameter, (Max CtlPts ID, Max Trivars ID, Max Arrangement ID) if successful, NULL otherwise.

Description: Returns the maximal IDS used in this arrangement.

13.2.103 TrivIGAGetKnotInterval (triv_iga.c:1452)

```
const CagdRType *TrivIGAGetKnotInterval(TrivIGAArrangementID ArgmntID,
                                        const TrivTVStruct *TV,
                                        TrivTVDirType Dir,
                                        int BzrIntervalIndex)
```

ArgmntID: A handle on the IGA arrangement to process.

TV: The Trivar to fetch its knot sequence.

Dir: The direction to fetch the knots: U, V, or W.

BzrIntervalIndex: Index of Bezier trivar, starting from zero.

Returns: The list of the (2*Order) knots, or NULL if error.

Description: Returns the knot sequence of TV that is used in the designated Bezier trivar index and direction.

13.2.104 TrivIGAGetLastError (triv_iga.c:2576)

error handling

```
TrivIGAErrorType TrivIGAGetLastError(TrivIGAArrangementID ArgmntID,
                                     int Reset)
```

ArgmntID: A handle on the IGA arrangement to process.

Reset: TRUE to also reset the last error as a side effect.

Returns: An ID with the error type, with TRIV_IGA_ERR_NO_ERROR if no error.

Description: Returns anID describing the given error.
See also: IritTrivIGADescribeError,

13.2.105 TrivIGAGetMaterial (triv_iga2.c:1078)

```
TrivIGAMaterialID TrivIGAGetMaterial(TrivIGAArrangementID ArgmntID,
                                     TrivIGATVID TVID)
```

ArgmntID: A handle on the IGA arrangement to process.

TVID: ID of TV to fetch its material ID.

Returns: The material fetched ID.

Description: Fetches the material ID of a given TV ID.

13.2.106 TrivIGAGetNumBzrElements (triv_iga.c:1315)

```
int TrivIGAGetNumBzrElements(TrivIGAArrangementID ArgmntID,
                             const TrivTVStruct *TV,
                             int *NumU,
                             int *NumV,
                             int *NumW)
```

ArgmntID: A handle on the IGA arrangement to process.

TV: The Trivar to fetch how many Bezier trivariates it has.

NumU: Number of Bezier trivariates in U.

NumV: Number of Bezier trivariates in V.

NumW: Number of Bezier trivariates in W.

Returns: TRUE if successful, FALSE otherwise.

Description: Returns the number of Bezier trivariates that exists along the three axes of (possibly B-spline) trivar TV in the IGA arrangement. Assumes TV has open end conditions.

13.2.107 TrivIGAGetTV (triv_iga2.c:633)

```
TrivTVStruct *TrivIGAGetTV(TrivIGAArrangementID ArgmntID, TrivIGATVID TVID)
```

ArgmntID: A handle on the IGA arrangement to process.
TVID: The TV ID to search with.
Returns: Found TV or NULL if not found.
Description: Get a pointer to the TV, given its TV ID in the arrangement.

13.2.108 TrivIGAGetTVCtlPtsIndices (triv_iga2.c:970)

```
int *TrivIGAGetTVCtlPtsIndices(TrivIGAArrangementID ArgmntID, TrivIGATVID TVID)
```

ArgmntID: A handle on the IGA arrangement to process.
TVID: ID of TV to get all its control points IDs.
Returns: A dynamically allocated vector of all IDS or NULL if error.
Description: Returns all the indices of all control points of the given TV.

13.2.109 TrivIGAGetTVFaceAsSrf (triv_iga2.c:920)

```
CagdSrfStruct *TrivIGAGetTVFaceAsSrf(TrivIGAArrangementID ArgmntID,
                                     TrivIGATVID TVID,
                                     int FaceID)
```

ArgmntID: A handle on the IGA arrangement to process.
TVID: The TV ID to seek its face.
FaceID: 0 to 5 for (UMin, UMax, VMin, VMax, WMin, WMax)
Returns: Sought source of NULL if error.
Description: Return indices of ALL control points in a face of designated trivariate.

13.2.110 TrivIGAGetTVFaceCtlPtsIDs (triv_iga2.c:655)

```
int *TrivIGAGetTVFaceCtlPtsIDs(TrivIGAArrangementID ArgmntID,
                               TrivIGATVID TVID,
                               int FaceID)
```

ArgmntID: A handle on the IGA arrangement to process.
TVID: The TV ID to seek its face.
FaceID: 0 to 5 for (UMin, UMax, VMin, VMax, WMin, WMax)
Returns: A vector of all IDs, terminated with -1 or NULL if error.
Description: Return indices of ALL control points in a face of designated trivariate.

13.2.111 TrivIGAGetVrtxNeighboringTVs (triv_iga.c:2365)

```
int *TrivIGAGetVrtxNeighboringTVs(TrivIGAArrangementID ArgmntID,
                                  const TrivTVStruct *TV)
```

ArgmntID: A handle on the IGA arrangement to process.
TV: Trivar to seek its vertex (corner) neighbors.
Returns: A TRIV_IGA_INVALID_TV_ID terminated vector of vertex neighboring TV IDS or NULL if error.
Description: Given a TV in some field, returns the vertex (corner) neighboring trivariates of TV, if exists.
See also: TrivIGAGetFaceNeighboringTVs, TrivIGAGetEdgeNeighboringTVs,

13.2.112 TrivIGALoadMaterialFromXML (triv_iga2.c:1154)

```
int TrivIGALoadMaterialFromXML(TrivIGAArrangementID ArgmntID,
                               const char *FileName)
```

ArgmntID: A handle on the IGA arrangement to process.

FileName: Name of material file to load.

Returns: Number of materials read.

Description: Load the materials defined in the given XML file.
See also: TrivIGALoadMaterialXML,

13.2.113 TrivIGALoadMaterialXML (triv_iga_xml.c:163)

```
int TrivIGALoadMaterialXML(const char *FileName,
                           TrivIGAMaterialStruct **Materials,
                           int *NumMaterials)
```

FileName: XML file name.

Materials: Array of loaded materials allocated dynamically.

NumMaterials: The number of the loaded materials.

Returns: TRUE on success, and FALSE in failure

Description: Load the materials defined in the given XML file.

13.2.114 TrivIGANewArrangement (triv_iga.c:535)

```
int TrivIGANewArrangement(TrivIGAArrangementID *NewArgmntID)
```

NewArgmntID: If successful, will hold the ID of the newly allocated arrangement.

Returns: 0 if error or arrangement ID if valid.

Description: Creates a new structure to hold the IGA analysis arrangement.

13.2.115 TrivIGANewField (triv_iga.c:647)

```
int TrivIGANewField(TrivIGAArrangementID ArgmntID,
                    const char *FieldAttributes)
```

ArgmntID: A handle on the IGA arrangement to process.

FieldAttributes: Geometry or scalar/vector data fields.

Returns: TRUE if successful, FALSE otherwise.

Description: Declares a new field. Every field can consists of several (adjacent or not) trivariates and all subsequent call of TrivIGAAddTrivar2Field will be placed in this new field.

13.2.116 TrivIGANewMaterial (triv_iga2.c:1123)

```
TrivIGAMaterialID TrivIGANewMaterial(TrivIGAArrangementID ArgmntID,
                                     const char *MaterialStr)
```

ArgmntID: A handle on the IGA arrangement to process.

MaterialStr: material description to parse.

Returns: New material ID or TRIV_IGA_INVALID_MATERIAL_ID if failed.

Description: Parses a new material description and create a new material.

13.2.117 TrivIGANewTV (triv_iga.c:757)

```
TrivIGATVStruct *TrivIGANewTV(TrivIGAArrangementID ArgmntID, TrivTVStruct *TV)
```

ArgmntID: A handle on the IGA arrangement to process.

TV: The Trivar to insert into the currently defined field, in place.

Returns: The allocated IGA TV, or NULL if error.

Description: Inserts a new trivariate into the IGA arrangement, in the current field, in place.

13.2.118 TrivIGAParseMaterial (triv_iga.c:256)

```
TrivIGAMaterialStruct *TrivIGAParseMaterial(const char *MaterialStr)
```

MaterialStr: string representing the material in the following format id = <id>, name = <name>, type = <type>, attrib1 = <attrib1>, ... where id, name, type attributes are must.

Returns: new created material

Description: Creates a new material and initializes it. Updates it otherwise.

13.2.119 TrivIGAPrintTVContent (triv_iga.c:1107)

```
int TrivIGAPrintTVContent(TrivIGAArrangementID ArgmntID,
                          const TrivTVStruct *TV)
```

ArgmntID: A handle on the IGA arrangement to process.

TV: The Trivar to print to stderr.

Returns: TRUE if successful, FALSE otherwise.

Description: A debugging function to dump to stderr, the content of the given TV.

13.2.120 TrivIGASetCtrlPtsPositions (triv_iga.c:1627)

```
int TrivIGASetCtrlPtsPositions(TrivIGAArrangementID ArgmntID,
                               int NumCtrlPts,
                               const TrivIGACtrlPtStruct *Vals)
```

ArgmntID: A handle on the IGA arrangement to process.

NumCtrlPts: Length of vector Vals.

Vals: Vector of control points to set the relevant control points of TV with. The control point type in Vals prescribes how many (Coord[i], ID[i]) pairs are in Vals which can be arbitrary, as a regular int value. The values are set by identified the ID in TV, for each (Coord[i], ID[i]) pair.

Returns: TRUE if successful, FALSE otherwise.

Description: A modifier function to replace control points' values in an arrangement, with the values as specified by Vals. The control points are identified by their unique IDs and must all be in some TV(s).

13.2.121 TrivIGASetDefaultDomain (triv_iga2.c:92)

```
int TrivIGASetDefaultDomain(TrivIGAArrangementID ArgmntID,
                            TrivTVDirType Dir,
                            CagdRType Min,
                            CagdRType Max)
```

ArgmntID: A handle on the IGA arrangement to process.

Dir: U, V, W direction to set the default domain to.

Min, Max: The domain to set.

Returns: TRUE if successful, FALSE otherwise.

Description: Sets the default domain of all to-be-entered trivariates into the arrangement in the given direction (u, V, or W). If TMax <= TMin, then this option is disabled (default) and the doamin is kept as it is inserted into the arrangement.

13.2.122 TrivIGASetDefaultSeeding (triv_iga2.c:34)

```
int TrivIGASetDefaultSeeding(TrivIGAArrangementID ArgmntID,
                             TrivTVDirType Dir,
                             CagdRType Alpha,
                             int NumIntervals)
```

ArgmntID: A handle on the IGA arrangement to process.

Dir: U, V, W direction to set the default seeding to.

Alpha: Ratio of last interval's length divided by first interval length, in set direction Dir.

NumIntervals: Number of interval to introduce.

Returns: The number of intervals that will be introduced in that axes.

Description: Sets the default seeding of all to-be-entered trivariates into the arrangement in the given direction (u, V, or W).

13.2.123 TrivIGATDegreeRaise (triv_iga2.c:546)

```
TrivIGATVStruct *TrivIGATDegreeRaise(TrivIGAArrangementID ArgmntID,
                                     TrivIGATVID TVID,
                                     TrivTVDirType Dir)
```

ArgmntID: A handle on the IGA arrangement to process.

TVID: ID of the TV to degree raise in the arrangement.

Dir: Degree raising direction - U, V or W.

Returns: Reference to the IGA TV degree raised, or NULL if error.

Description: Degree raise a given TV (ID) in the arrangement.

13.2.124 TrivIGATVEvalBasis (triv_iga.c:1974)

```
CagdRType *TrivIGATVEvalBasis(TrivIGAArrangementID ArgmntID,
                              const TrivTVStruct *TV,
                              TrivIGAEvalType EvalType,
                              TrivTVDirType Dir,
                              int Index,
                              CagdRType t,
                              CagdRType *Basis)
```

ArgmntID: A handle on the IGA arrangement to process.

TV: Trivar to evaluation a one location.

EvalType: What to evaluate: position, derivatives, normals, etc.

Dir: Direction to evaluate the basic function - U, V or W.

Index: The Bezier trivariate index in Dir in TV, starting from zero.

t: Parameter values to evaluate the designated Bezier trivar at, in Dir, in [0, 1].

Basis: Preallocated vector of size Order in Dir in TV to hold the resulting basis function values. Same as returned value.

Returns: A vector of Order values in Dir in TV holding the relevant basis function values, or NULL if error. Same as Basis parameter.

Description: Evaluate the basis function values of given TV at the given location. The evaluation location is identified by the relevant Bezier trivar (using IndexU/V/W) and the parameter values that are always normalized to be between zero and one.

13.2.125 TrivIGATVEvalToData (triv_iga.c:1681)

```
const TrivIGACtrlPtStruct *TrivIGATVEvalToData(TrivIGAArrangementID ArgmntID,
                                               const TrivTVStruct *TV,
                                               TrivIGAEvalType EvalType,
                                               int IndexU,
                                               int IndexV,
                                               int IndexW,
                                               CagdRType U,
                                               CagdRType V,
                                               CagdRType W,
                                               TrivIGACtrlPtStruct *CtlPt)
```

ArgmntID: A handle on the IGA arrangement to process.

TV: Trivar to evaluation a one location.

EvalType: What to evaluate: position, derivatives, normals, etc.

IndexU: The Bezier trivariate index in U in TV, starting from zero.

IndexV: The Bezier trivariate index in V in TV, starting from zero.

IndexW: The Bezier trivariate index in W in TV, starting from zero.

U, V, W: Parameters values to evaluate the designated Bezier trivar at. In [0, 1] only. For many evaluations, U varying first and W last will yield the best performance.

CtlPt: A control point to update with the evaluation.

Returns: A control point

Description: Evaluate the given TV at the given location. The evaluation location is identified by the relevant Bezier trivar (using IndexU/V/W) and the parameter values that are always normalized to be between zero and one.

13.2.126 TrivIGATVFromSurfaces (triv_iga2.c:418)

```
TrivIGATVID TrivIGATVFromSurfaces(TrivIGAArrangementID ArgmntID,
                                  const CagdSrfStruct *SrfList,
                                  int OtherOrder,
                                  CagdBType IsInterpolating,
                                  int ID)
```

ArgmntID: A handle on the IGA arrangement to process.

SrfList: List of surfaces to construct a trivariate with.

OtherOrder: Other, third, order of trivariate.

IsInterpolating: TRUE for the trivariate to interpolate the surfaces, FALSE to approximate them.

ID: ID to use or -1 to set a new ID.

Returns: INVALID_IGA_TV_ID if error, to TV ID if successful.

Description: Constructs a new trivariate and add it to the give arrangement as a trivariate through surfaces of the given ordered surfaces.

13.2.127 TrivIGATVFromSurfaces2 (triv_iga2.c:462)

```
TrivIGATVID TrivIGATVFromSurfaces2(TrivIGAArrangementID ArgmntID,
                                   const CagdSrfStruct *Srf,
                                   IrtHmgnMatType Transforms[],
                                   int NumTransforms,
                                   unsigned int OtherOrder,
                                   CagdBType IsInterpolating,
                                   int ID)
```

ArgmntID: A handle on the IGA arrangement to process.

Srf: Base surface to transform in space & skin a volume thru.

Transforms: Vector of transformation matrices to apply to Srf.

NumTransforms: Size of vector Transforms.

OtherOrder: Other order to use, in the skinning direction.

IsInterpolating: TRUE to interpolate surfaces, FALSE to approximate them.

ID: ID to use or -1 to set a new ID.

Returns: INVALID_IGA_TV_ID if error, to TV ID if successful.

Description: Constructs a new trivariate and add it to the give arrangement as a trivariate through surfaces of the given ordered transformations of Srf.

13.2.128 TrivIGATVRefine (triv_iga2.c:501)

```
TrivIGATVStruct *TrivIGATVRefine(TrivIGAArrangementID ArgmntID,
                                 TrivIGATVID TVID,
                                 TrivTVDirType Dir,
                                 CagdRType t)
```

ArgmntID: A handle on the IGA arrangement to process.

TVID: ID of the TV to refine in the arrangement.

Dir: Refinement direction - U, V or W.

t: Parameter at which to insert the knot.

Returns: Reference to the IGA TV refined, or NULL if error.

Description: Refines a given TV (ID) in the arrangement.

13.2.129 TrivIGATVofRevol (triv_iga2.c:365)

```
TrivIGATVID TrivIGATVofRevol(TrivIGAArrangementID ArgmntID,
                             const CagdSrfStruct *Srf,
                             const IrtPtType AxisPoint,
                             const IrtVecType AxisVector,
                             CagdRType StartAngle,
                             CagdRType EndAngle,
                             CagdBType IsRational,
                             int ID)
```

ArgmntID: A handle on the IGA arrangement to process.

Srf: Surface to rotate.

AxisPoint, AxisVector: Axis line of rotation's prescription.

StartAngle: Starting angle in degs., 0 for a full circle.

EndAngle: Terminating angle in degs., 360 for a full circle.

IsRational: TRUE to construct a rational precise volume of revolution, FALSE for polynomial approximation.

ID: ID to use or -1 to set a new ID.

Returns: INVALID_IGA_TV_ID if error, to TV ID if successful.

Description: Constructs a new trivariate and add it to the give arrangement as a volume of revolution of the given surface.

13.2.130 TrivIGAUpdateCtrlPtsPositions (triv_iga.c:1582)

```
int TrivIGAUpdateCtrlPtsPositions(TrivIGAArrangementID ArgmntID,
                                  int NumCtrlPts,
                                  const TrivIGACtrlPtStruct *DeltaVals)
```

ArgmntID: A handle on the IGA arrangement to process.

NumCtrlPts: Length of vector DeltaVals.

DeltaVals: Vector of control points with delta values to set the relevant control points of TV with. The control point type in Vals prescribes how many (Coord[i], ID[i]) pairs are in Vals which can be arbitrary, as a regular int value. The values are set by identified the ID in TV, for each (Coord[i], ID[i]) pair.

Returns: TRUE if successful, FALSE otherwise.

Description: A modifier function to update control points in an arrangement, with the delta values as specified by DeltaVals. The control points are identified by their unique IDs and must all be in some TV(s).

13.2.131 TrivIGAUpdateTV (triv_iga.c:883)

```
TrivIGATVStruct *TrivIGAUpdateTV(TrivIGAArrangementID ArgmntID,
                                 TrivTVStruct *ExistingTV,
                                 TrivTVStruct *NewTV)
```

ArgmntID: A handle on the IGA arrangement to process.

ExistingTV: Current TV in the arrangement to update. Will be freed.

NewTV: New TV to replace the existing TV.

Returns: Reference to the updated IGA TV, or NULL if error.

Description: Inserts a new trivariate into the IGA arrangement, in the current field.

13.2.132 TrivIgaGenOneFaceNeighboringConstraints (triv_iga2.c:804)

```
int TrivIgaGenOneFaceNeighboringConstraints(
                                    TrivIGAArrangementID ArgmntID,
                                    TrivIGANeighboringConstraintCallBackType
                                              NeighboringConstraintCallBack,
                                    const TrivTVStruct *TV1,
                                    int FaceID1,
                                    const TrivTVStruct *TV2,
                                    int FaceID2,
                                    void *CallbackData)
```

ArgmntID: A handle on the IGA arrangement to process.

NeighboringConstraintCallBack: Call back function to call with the constraint as a string.

TV1, FaceID1: Description of first face.

TV2, FaceID2: Description of second face.

CallbackData: Relevant data for the callback function (such as XML information).

Returns: TRUE if successful, FALSE otherwise.

Description: Construct linear constraints hooking one adjacent face that does not share a common function space with its neighbor while being the same.

13.2.133 TrivImplicitTVFromDistCrvs (trivcnst.c:251)

```
TrivTVStruct *TrivImplicitTVFromDistCrvs(int Order,
                                         int Length,
                                         const CagdCrvStruct *Crvs,
                                         CagdRType DistThreshold,
                                         const CagdRType CornerBias[2])
```

Order: Of desired trivariate in all three axes.

Length: Of desired trivariate in all three axes.

Crvs: List of curves to compute (minimal) distances to. Curves are considered only in region [-1, 1]^2

DistThreshold: How far from the curves should be considered 'inside'?

CornerBias: Two scalar values to control the span of extended support for implicit values at the eight corners. The first control how far from a corner wil have an effect and the second controls magnitude of the effect.

Returns: An implicit trivariate of domain [0, 1]^3, which its Zero set approximates the desired shape from curves.

Description: Builds an implicit trivariate of specified Order and Length (equal in all three axes) to approximate the shape defined as (minimal) distance field to the given set of curves.

See also:

13.2.134 TrivInterpTrivar (trinterp.c:30)

interpolation

```
TrivTVStruct *TrivInterpTrivar(const TrivTVStruct *TV)
```

TV: Trivariate to interpolate its control mesh.

Returns: The interpolating trivariate.

Description: Interpolates control points of given trivariate, preserving the order and continuity of the original trivariate.
See also: TrivTVInterpPts, TrivTVInterpolate,

13.2.135 TrivInverseQuery (trivinvs.c:140)

```
int TrivInverseQuery(struct TrivInverseQueryStruct *Handle,
                     const CagdRType *XYZPos,
                     CagdRType *UVWParams,
                     int InitialGuess)
```

Handle: The inverse query handle.

XYZPos: New xyz position to query.

UVWParams: u,v,w result. Contains an initial u,v,w guess when called.

InitialGuess: Non-zero if initial (u,v,w) guess is valid.

Returns: Number of solutions (0 or 1).

Description: Given an (x,y,z) position, compute the (u,v,w) for the trivar used to create the inverse query handle, if any.
See also: TrivPrepInverseQueries, TrivFreeInverseQueries, TrivInverseQueryPolys,

13.2.136 TrivInverseQueryPolys (trivinvs.c:327)

```
int TrivInverseQueryPolys(IPObjectStruct *PlObj, const TrivTVStruct *TV)
```

PlObj: Polygons to update their vertices with UVW params in TV.

TV: To compute the XYZ to UVW inverse transform.

Returns: Zero if successful, otherwise number of vertices failing.

Description: Computes UVW values of the given polygonal data that is assumed to be in the domain of TV. UVW values as saved as two attributes: "uvvals" and "wval".
See also: TrivInverseQuery,

13.2.137 TrivIsTVClosed (triv_gen.c:1001)

```
CagdBType TrivIsTVClosed(const TrivTVStruct *TV, int Dim)
```

TV: B-spline trivariate to examine if closed in given Dim.

Dim: 0, 1, 2 for U, V,W direction to check if closed.

Returns: TRUE if TV is closed in Dim, FALSE otherwise.

Description: Examines if the given trivariate is closed (i.e. minimal surface equals maximal surface)in the given dimension, Dim.
See also: TrivTVOpenEnd,

13.2.138 TrivLoadVolumeIntoTV (mrch_run.c:1297)

```
TrivTVStruct *TrivLoadVolumeIntoTV(const char *FileName,
                                   int DataType,
                                   IrtVecType VolSize,
                                   IrtVecType Orders)
```

FileName: To load the trivariate data set from.

DataType: Type of scalar value in volume data file. Can be one of: 1 - Regular ascii (separated by while spaces. 2 - Two bytes short integer. 3 - Four bytes long integer. 4 - One byte (char) integer. 5 - Four bytes float. 6 - Eight bytes double.

VolSize: Dimensions of trivariate volume.

Orders: Orders of the three axis of the volume (in U, V, and W).

Returns: Loaded trivariate, or NULL if error.

Description: Loads a volumetric data set as a trivariate function of prescribed orders. Uniform open end conditions are created for it.

13.2.139 TrivMakeTVArrngmntCompatible (triv_adj.c:927)

```
TrivTVStruct *TrivMakeTVArrngmntCompatible(const TrivTVStruct *TVList)
```

TVList: A List of trivariates in which each trivariate is to be made compatible with all of its adjacent neighbouring trivariates.

Returns: The compatible equivalent list.

Description: Given a list of trivariates with adjacency information and modification flags, make adjacent trivariates compatible with degree raise and refinement.
See also: TrivTVMarkAdjacencyInfo,

13.2.140 TrivMakeTVsCompatible (trivcmpt.c:93)

compatibility

```
CagdBType TrivMakeTVsCompatible(TrivTVStruct **TV1,
                                TrivTVStruct **TV2,
                                CagdBType SameUOrder,
                                CagdBType SameVOrder,
                                CagdBType SameWOrder,
                                CagdBType SameUKV,
                                CagdBType SameVKV,
                                CagdBType SameWKV)
```

TV1, TV2: Two surfaces to be made compatible, in place.

SameUOrder: If TRUE, this routine make sure they share the same U order.

SameVOrder: If TRUE, this routine make sure they share the same V order.

SameWOrder: If TRUE, this routine make sure they share the same W order.

SameUKV: If TRUE, this routine make sure they share the same U knot vector and hence continuity. *

SameVKV: If TRUE, this routine make sure they share the same V knot vector and hence continuity.

SameWKV: If TRUE, this routine make sure they share the same W knot vector and hence continuity.

Returns: TRUE if successful, FALSE otherwise.

Description: Given two trivariates, makes them compatible by:

1. Coercing their point type to be the same.
2. Making them have the same tri-variate type.
3. Raising the degree of the lower one to be the same as the higher.
4. Refining them to a common knot vector (If Bspline and SameOrder).

Note 3 is performed if SameOrder TRUE, 4 if SameKV TRUE. Both trivariates are modified IN PLACE.

13.2.141 TrivMakeTVsCompatibleDomain (trivcmpt.c:28)

```
int TrivMakeTVsCompatibleDomain(const TrivTVStruct *TV1, TrivTVStruct **TV2)
```

TV1: Trivariate to use its domain.

TV2: Trivariate to change its domain to be the same as TV1.

Returns: TRUE if successful.

Description: Maps the domain of the second trivariate TV2 to that of TV1.
See also:

13.2.142 TrivMergeTVTV (triv_aux.c:941)

```
TrivTVStruct *TrivMergeTVTV(const TrivTVStruct *CTV1,
                            const TrivTVStruct *CTV2,
                            TrivTVDirType Dir,
                            CagdBType Discont)
```

CTV1: To connect to CTV2's starting boundary at its end.

CTV2: To connect to CTV1's end boundary at its start.

Dir: Direction the merge should take place.

Discont: If TRUE, assumes the merged "edge" is discontinuous.

Returns: The merged trivariate.

Description: Merges two trivariates into along diretion Dir.
See also: MvarMergeTVList,

13.2.143 TrivNSPrimBox (trivprim.c:487)

```
TrivTVStruct *TrivNSPrimBox(CagdRType MinX,
                            CagdRType MinY,
                            CagdRType MinZ,
                            CagdRType MaxX,
                            CagdRType MaxY,
                            CagdRType MaxZ)
```

MinX, MinY, MinZ: Minimum range of box trivariate.

MaxX, MaxY, MaxZ: Maximum range of box trivariate.

Returns: A trivariate of a box model.

Description: A trivariate constructor of a box, parallel to main axes.
See also: TrivNSPrimGenBox, TrivNSPrimCone, TrivNSPrimCone2, TrivNSPrimCylinder, , TrivNSPrimSphere, TrivNSPrimTorus, TrivNSPrimGenBox2,

13.2.144 TrivNSPrimCone (trivprim.c:724)

```
TrivTVStruct *TrivNSPrimCone(const CagdVType Center,
                             CagdRType Radius,
                             CagdRType Height,
                             CagdBType Rational,
                             CagdRType InternalCubeSize)
```

Center: Of the cone.

Radius: Of the cone.

Height: Of the cone.

Rational: TRUE for precise rational, FALSE for polynomial approximation.

InternalCubeSize: Size of the internal cube trivariate edge.

Returns: A list of 5 trivariates, representing the cone.

Description: A trivariate constructor of a cone, centered at Center with radius Radius, and a Height height. The cone is built from 5 ruled trivariates. Note the cone will be (only) at the apex.
See also: TrivNSPrimBox, TrivNSPriCmone2, TrivNSPrimCylinder, TrivNSPrimSphere, , TrivNSPrimTorus,

13.2.145 TrivNSPrimCone2 (trivprim.c:759)

```
TrivTVStruct *TrivNSPrimCone2(const CagdVType Center,
                              CagdRType MajorRadius,
                              CagdRType MinorRadius,
                              CagdRType Height,
                              CagdBType Rational,
                              CagdRType InternalCubeSize)
```

Center: Of the cone.

MajorRadius: Of the cone.

MinorRadius: Of the cone.

Height: Of the cone.

Rational: TRUE for precise rational, FALSE for polynomial approximation.

InternalCubeSize: Size of the internal cube trivariate edge.

Returns: A list of 5 trivariates, representing the cone.

Description: A trivariate constructor of a truncated cone, centered at Center with major radius MajorRadius and minor radius MinorRadius, and a Height height. The cone is built from 5 non singular ruled trivariates.
See also: TrivNSPrimBox, TrivNSPrimCone, TrivNSPrimCylinder, TrivNSPrimSphere, , TrivNSPrimTorus,

13.2.146 TrivNSPrimCylinder (trivprim.c:666)

```
TrivTVStruct *TrivNSPrimCylinder(const CagdVType Center,
                                 CagdRType Radius,
                                 CagdRType Height,
                                 CagdBType Rational,
                                 CagdRType InternalCubeSize)
```

Center: Of the cylinder.

Radius: Of the cylinder.

Height: Of the cylinder.

Rational: TRUE for precise rational, FALSE for polynomial approximation.

InternalCubeSize: Size of the internal cube trivariate edge.

Returns: A list of 5 trivariates, representing the cylinder.

Description: A trivariate constructor of a cylinder, centered at Center with radius Radius, and a Height height. The cylinder is built from 5 non singular ruled trivariates.
See also: TrivNSPrimBox, TrivNSPrimCone, TrivNSPrimCone2, TrivNSPrimSphere, , TrivNSPrimTorus,

13.2.147 TrivNSPrimGenBox (trivprim.c:531)

```
TrivTVStruct *TrivNSPrimGenBox(const CagdPType P000,
                               const CagdPType P001,
                               const CagdPType P010,
                               const CagdPType P011,
                               const CagdPType P100,
                               const CagdPType P101,
                               const CagdPType P110,
                               const CagdPType P111)
```

P000, P001, P010, P011, P100, P101, P110, P111: 8 corners of the generalzied cube.

Returns: A trivariate of a generalized box model.

Description: A trivariate constructor of a general box, give 8 corners. No check is made that the Jacobian is valid here.
See also: TrivNSPrimBox, TrivNSPrimCone, TrivNSPrimCone2, TrivNSPrimCylinder, , TrivNSPrimSphere, TrivNSPrimTorus,

13.2.148 TrivNSPrimGenBox2 (trivprim.c:583)

```
TrivTVStruct *TrivNSPrimGenBox2(const CagdVType VecUMin,
                                const CagdVType VecUMax,
                                const CagdVType VecVMin,
                                const CagdVType VecVMax,
                                const CagdVType VecWMin,
                                const CagdVType VecWMax,
                                int Normalize)
```

VecUMin, VecUMax, VecVMin, VecVMax, VecWMin, VecWMax: The 6 normals. Note that the faces actual normals might be somewhat different and further, the faces will not necessarily be planar.

Normalize: TRUE to normalize the input vectors first.

Returns: A trivariate of a generalized box model. Result will be positioned centered at the origin.

Description: A trivariate constructor of a general box, give 6 vectors that the normal to the six faces of the trivariate.
See also: TrivNSPrimBox, TrivNSPrimCone, TrivNSPrimCone2, TrivNSPrimCylinder, , TrivNSPrimSphere, TrivNSPrimTorus, TrivNSPrimGenBox,

13.2.149 TrivNSPrimSphere (trivprim.c:824)

```
TrivTVStruct *TrivNSPrimSphere(const CagdVType Center,
                               CagdRType Radius,
                               CagdBType Rational,
                               CagdRType InternalCubeSize)
```

Center: Of the ball.

Radius: Of the ball.

Rational: TRUE for precise rational, FALSE for polynomial approximation.

InternalCubeSize: Size of the internal cube trivariate edge.

Returns: A list of 7 trivariates, representing the sphere.

Description: A trivariate constructor of a ball, centered at Center and radius Radius, built from 7 non singular trivariates, an inner cube and 6 ruled trivariate between patches of the ball's outer sphere surface and the cube faces.
See also: TrivNSPrimBox, TrivNSPrimCone, TrivNSPrimCone2, TrivNSPrimCylinder, , TrivNSPrimTorus, TrivNSPrimSphere,

13.2.150 TrivNSPrimTorus (trivprim.c:947)

```
TrivTVStruct *TrivNSPrimTorus(const CagdVType Center,
                              CagdRType MajorRadius,
                              CagdRType MinorRadius,
                              CagdBType Rational,
                              CagdRType InternalCubeSize)
```

Center: Of the torus.

MajorRadius: Of the torus.

MinorRadius: Of the torus.

Rational: TRUE for precise rational, FALSE for polynomial approximation.

InternalCubeSize: Size of the internal cube trivariate edge.

Returns: A list of 5 trivariates, representing the torus.

Description: A trivariate constructor of a torus, centered at Center with major radius MajorRadius and minor radius MinorRadius. The torus is built from 5 non singular trivariates of revolution.
See also: TrivNSPrimBox, TrivNSPrimCone, TrivNSPrimCone2, TrivNSPrimCylinder, , TrivNSPrimSphere,

13.2.151 TrivParamInDomain (triv_aux.c:252)

trivariates

```
CagdBType TrivParamInDomain(const TrivTVStruct *TV,
                            CagdRType t,
                            TrivTVDirType Dir)
```

TV: To make sure t is in its Dir domain.

t: Parameter value to verify.

Dir: Direction. Either U or V or W.

Returns: TRUE if in domain, FALSE otherwise.

Description: Given a tri-variate and a domain - validate it.

13.2.152 TrivParamsInDomain (triv_aux.c:288)

trivariates

```
CagdBType TrivParamsInDomain(const TrivTVStruct *TV,
                             CagdRType u,
                             CagdRType v,
                             CagdRType w)
```

TV: To make sure (u, v, w) is in its domain.

u, v, w: To verify if it is in TV's parametric domain.

Returns: TRUE if in domain, FALSE otherwise.

Description: Given a tri-variate and a domain - validate it.

13.2.153 TrivPlaneFrom4Points (geomat4d.c:44)

hyper plane

plane

```
int TrivPlaneFrom4Points(const TrivP4DType Pt1,
                         const TrivP4DType Pt2,
                         const TrivP4DType Pt3,
                         const TrivP4DType Pt4,
                         TrivPln4DType Plane)
```

Pt1, Pt2, Pt3, Pt4: The four points the plane should go through.

Plane: Where the result should be placed.

Returns: TRUE if successful, FALSE otherwise.

Description: Computes a hyperplane in four space through the given four points. Based on a direct solution in Maple of:

```
with(linalg);
readlib(C);



d := det( matrix( [ [x - x1, y - y1, z - z1, w - w1],
                    [x2 - x1, y2 - y1, z2 - z1, w2 - w1],
                    [x3 - x2, y3 - y2, z3 - z2, w3 - w2],
                    [x4 - x3, y4 - y3, z4 - z3, w4 - w3]] ) );
coeff( d, x );
coeff( d, y );
coeff( d, z );
coeff( d, w );
```

13.2.154 TrivPrepInverseQueries (trivinvs.c:81)

```
struct TrivInverseQueryStruct *TrivPrepInverseQueries(const TrivTVStruct
                                                      *Trivar)
```

Trivar: The trivar.

Returns: Handle for retrieval of results.

Description: Create a handle that can be used for efficient inreverse queries (given an (x,y,z) Euclidean position get (u,v,w)) in trivariate.
See also: TrivInverseQuery, TrivFreeInverseQueries,

13.2.155 TrivPrimCone (trivprim.c:268)

```
TrivTVStruct *TrivPrimCone(const CagdVType Center,
                           CagdRType Radius,
                           CagdRType Height,
                           CagdBType Rational)
```

Center: Of the cone.

Radius: Of the cone.

Height: Of the cone.

Rational: TRUE for precise rational, FALSE for polynomial approximation.

Returns: A trivariate, representing the cone.

Description: A trivariate constructor of a cone, centered at Center with radius Radius, and a Height height. The cone is built from a singular (on boundary) ruled trivariate. Note the cone will also be singular at the apex.
See also: TrivPrimBox, TrivPriCmone2, TrivPrimCylinder, TrivPrimSphere, , TrivPrimTorus, TrivNSPrimCone,

13.2.156 TrivPrimCone2 (trivprim.c:300)

```
TrivTVStruct *TrivPrimCone2(const CagdVType Center,
                            CagdRType MajorRadius,
                            CagdRType MinorRadius,
                            CagdRType Height,
                            CagdBType Rational)
```

Center: Of the cone.

MajorRadius: Of the cone.

MinorRadius: Of the cone.

Height: Of the cone.

Rational: TRUE for precise rational, FALSE for polynomial approximation.

Returns: A trivariate, representing the cone.

Description: A trivariate constructor of a truncated cone, centered at Center with major radius MajorRadius and minor radius MinorRadius, and a Height height. The cone is built from a singular (on boundary) ruled trivariate.
See also: TrivPrimBox, TrivPrimCone, TrivPrimCylinder, TrivPrimSphere, , TrivPrimTorus, TrivNSPrimCone2,

13.2.157 TrivPrimCylinder (trivprim.c:223)

```
TrivTVStruct *TrivPrimCylinder(const CagdVType Center,
                               CagdRType Radius,
                               CagdRType Height,
                               CagdBType Rational)
```

Center: Of the cylinder.

Radius: Of the cylinder.

Height: Of the cylinder.

Rational: TRUE for precise rational, FALSE for polynomial approximation.

Returns: A trivariate, representing the cylinder.

Description: A trivariate constructor of a cylinder, centered at Center with radius Radius, and a Height height. The cylinder is built from a singular (on boundary) ruled trivariate.
See also: TrivPrimBox, TrivPrimCone, TrivPrimCone2, TrivPrimSphere, , TrivPrimTorus, TrivPrimCylinder,

13.2.158 TrivPrimSphere (trivprim.c:350)

```
TrivTVStruct *TrivPrimSphere(const CagdVType Center,
                             CagdRType Radius,
                             CagdBType Rational)
```

Center: Of the ball.

Radius: Of the ball.

Rational: TRUE for precise rational, FALSE for polynomial approximation.

Returns: A trivariate, representing the sphere.

Description: A trivariate constructor of a ball, centered at Center and radius Radius, built from a singular in the interior (axes) trivariate.
See also: TrivPrimBox, TrivPrimCone, TrivPrimCone2, TrivPrimCylinder, , TrivPrimTorus, TrivNSPrimSphere,

13.2.159 TrivPrimSphere2 (trivprim.c:392)

```
TrivTVStruct *TrivPrimSphere2(const CagdVType Center,
                              CagdRType Radius,
                              CagdBType Rational)
```

Center: Of the ball.

Radius: Of the ball.

Rational: TRUE for precise rational, FALSE for polynomial approximation.

Returns: A single trivariate, representing the sphere.

Description: A trivariate constructor of a ball, centered at Center and radius Radius, built from a single non singular trivariate in the interior.
See also: TrivNSPrimBox, TrivNSPrimCone, TrivNSPrimCone2, TrivNSPrimCylinder, , TrivNSPrimTorus, TrivNSPrimSphere,

13.2.160 TrivPrimTorus (trivprim.c:438)

```
TrivTVStruct *TrivPrimTorus(const CagdVType Center,
                            CagdRType MajorRadius,
                            CagdRType MinorRadius,
                            CagdBType Rational)
```

Center: Of the torus.

MajorRadius: Of the torus.

MinorRadius: Of the torus.

Rational: TRUE for precise rational, FALSE for polynomial approximation.

Returns: A trivariate, representing the torus.

Description: A trivariate constructor of a torus, centered at Center with major radius MajorRadius and minor radius MinorRadius. The torus is built from a trivariate.
See also: TrivPrimBox, TrivPrimCone, TrivPrimCone2, TrivPrimCylinder, , TrivPrimSphere, TrivNSPrimTorus,

13.2.161 TrivPwrTVNew (triv_gen.c:265)

trivariates

allocation

```
TrivTVStruct *TrivPwrTVNew(int ULength,
                           int VLength,
                           int WLength,
                           CagdPointType PType)
```

ULength: Number of control points in the U direction.

VLength: Number of control points in the V direction.

WLength: Number of control points in the W direction.

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform trivariate power basis.

Description: Allocates the memory required for a new power basis trivariate.
See also: TrivBzrTVNew, TrivBspTVNew, TrivTVNew,

13.2.162 TrivRuledTV (trivruld.c:36)

ruled trivariate

trivariate constructors

```
TrivTVStruct *TrivRuledTV(const CagdSrfStruct *CSrf1,
                          const CagdSrfStruct *CSrf2,
                          int OtherOrder,
                          int OtherLen)
```

CSrf1, CSrf2: The two surfaces to form a ruled trivariate in between.

OtherOrder: Usually two, but one can specify higher orders in the ruled direction. OtherOrder must never be larger than OrderLen.

OtherLen: Usually two control points in the ruled direction which necessitates a linear interpolation.

Returns: The ruled trivariate.

Description: Constructs a ruled trivariate between the two provided surfaces. OtherOrder and OtherLen (equal for Bezier) specifies the desired order and refinement level (if B-spline) of the other ruled direction.
See also: CagdRuledSrf,

13.2.163 TrivSrfArea (triv_aux.c:794)

bbox

bounding box

```
CagdRType TrivSrfArea(const CagdSrfStruct *Srf, CagdBType VolType)
```

Srf: To compute its surface area.

VolType: TRUE to integrate the surfaces with respect to the XY plane, FALSE to integrate the surfaces with respect to the origin,

Returns: The computed area (can be negative if Srf reversed).

Description: Computes the surface area of the given surface by approximating it using a very thin shell and computing the volume...
See also: TrivTVVolume,

13.2.164 TrivSrfFromMesh (triveval.c:1123)

trivariates

```
CagdSrfStruct *TrivSrfFromMesh(const TrivTVStruct *TV,
                               int Index,
                               TrivTVDirType Dir)
```

TV: Trivariate to extract a bivariate surface out of its mesh.

Index: Index of row/column/level of TV's mesh in direction Dir.

Dir: Direction of iso-surface extraction. Either U or V or W.

Returns: A bivariate surface which was extracted from TV's Mesh. This surface is not necessarily on TV.

Description: Extract a bivariate surface out of the given trivariate's mesh. The provided (zero based) Index specifies which Index to extract.

13.2.165 TrivSrfFromTV (triveval.c:497)

trivariates

```
CagdSrfStruct *TrivSrfFromTV(const TrivTVStruct *TV,
                             CagdRType t,
                             TrivTVDirType Dir,
                             int OrientBoundary)
```

TV: To extract an parametrize surface from at parameter value t in direction Dir.

t: Parameter value at which to extract the iso-surface.

Dir: Direction of iso-surface extraction. Either U or V or W.

OrientBoundary: Controls the orientations of extracted surfaces as, 0 - do no aim to examine/change the surfaces orientations. 1 - to reorient boundary surfaces to point with their normals into the trivariate. 2 - same as 1, but only tags the extract surface as "reversed" attribute, without reversing the surface. IF OrientBoundary is not 0, t must be a boundary parameter value.

Returns: A bivariate surface which is an isosurface of TV.

Description: Extract an isoparametric surface out of the given tensor product trivariate. Operations should favor the CONST_W_DIR, in which the extraction is somewhat faster, if it is possible. surfaces that are on the boundary are reoriented so their normals are pointing into the trivariate, if OrientBoundary is TRUE.

See also: TrivBndrySrfsFromTV,

13.2.166 TrivSrfToMesh (triveval.c:1255)

trivariates

```
void TrivSrfToMesh(const CagdSrfStruct *Srf,
                   int Index,
                   TrivTVDirType Dir,
                   TrivTVStruct *TV)
```

Srf: Surface to substitute into the trivariate TV.

Index: Index of row/column/level of TV's mesh in direction Dir.

Dir: Direction of isosurface extraction. Either U or V or W.

TV: Trivariate to substitute a bivariate surface into its mesh.

Returns: void

Description: Substitute a bivariate surface into a given trivariate's mesh. The provided (zero based) Index specifies which Index to extract.

13.2.167 TrivSweepTV (trivswep.c:103)

sweep

trivariate constructors

```
TrivTVStruct *TrivSweepTV(const CagdSrfStruct *CrossSection,
                          const CagdCrvStruct *Axis,
                          const CagdCrvStruct *ScalingCrv,
                          CagdRType Scale,
                          const VoidPtr Frame,
                          int FrameOption)
```

CrossSection: Of the constructed sweep trivar. If more than one curve is given as a linked list of curves, the cross sections are modified as we progresses along the sweep, blending between the cross sections so that last cross section is used in the last parameter value of the Axis.

Axis: Of the constructed sweep trivariate.

ScalingCrv: Optional scale or profile curve. NULL if none. The Y axis of this curve is using as scaling as the function of the parametrization. X axis can be used to define a 2D parametric curve so this curve can be displayed but X is ignored by this function.

Scale: If no Scaling Crv, Scale is used to apply a fixed scale on the CrossSection curve.

Frame: An optional vector or a curve to specify the binormal orientation. Otherwise, Frame must be NULL.

FrameOption: If 2, Frame is a curve, and if -2, -1, 0, 1 a vector is used to control the frame as follows, If 0, Frame vector is used just to init the first frame. If -1, use Frame vector to init first and last frame. If 1, use Frame vector for all frames. If -2, Frame holds two vectors (6 real values) to init both first and last frame.

Returns: Constructed sweep trivariate.

Description: Constructs a sweep trivariate using the following surface/curves:

1. CrossSection - defines the basic cross section of the sweep. Must be in the XY plane. Can be several surfaces to be blended along the Axis.
2. Axis - a 3D curve the CrossSection will be swept along such that the Axis normal aligns with the Y axis of the cross section. If Axis is linear (i.e. no normal), the normal is picked randomly or to fit the non linear part of the Axis (if any).
3. Scale - a scaling curve for the sweep, If NULL a scale of Scale is used.
4. Frame - a curve or a vector that specifies the orientation of the sweep by specifying the axes curve's binormal. If Frame is a vector, it is a constant binormal. If Frame is a curve (FrameOption == 2), it is assumed to be a vector field binormal. If NULL, it is computed from the Axis curve's pseudo Frenet frame, that minimizes rotation.

This operation is only an approximation. See CagdSweepAxisRefine for a tool to refine the Axis curve and improve accuracy.

See also: CagdSweepSrf, CagdSweepSrfError, VMdlSweepTrimSrf, TrivSweepTVC1,

13.2.168 TrivSweepTVC1 (trivswep.c:532)

sweep

trivar constructors

```
TrivTVStruct *TrivSweepTVC1(const CagdSrfStruct *CrossSection,
                            const CagdCrvStruct *Axis,
                            const CagdCrvStruct *ScalingCrv,
                            CagdRType Scale,
                            const VoidPtr Frame,
                            int FrameOption,
                            CagdCrvCornerType CornerType,
                            CagdRType C1DiscontCropTol)
```

CrossSection: Of the constructed sweep surface. If more than one srf is given as a linked list of surfaces, the cross sections are modified as we progresses along the sweep, blending between the cross sections so that last cross section is used in the last parameter value of the Axis.

Axis: Of the constructed sweep surface. Can be C1 discont.

ScalingCrv: Optional scale or profile curve. NULL if none. The Y axis of this curve is using as scaling as the function of the parametrization. X axis can be used to define a 2D parametric curve so this curve can be displayed but X is ignored by this function.

Scale: If no Scaling Crv, Scale is used to apply a fixed scale on the CrossSection curve.

Frame: An optional vector or a curve to specified the binormal orientation. Otherwise Frame must be NULL.

FrameOption: If 2 Frame is a curve, if 0 or 1 a vector (if Frame is not NULL). If 0, the vector is used just to init the first frame.

CornerType: C^1 discont. joint corner type - 1. For splitting and computing sweeps of individual C^1 cont. axes. 2. For rounded joints (circular if input is rational). 3. For chamfered joints. 4. For mitered joints.

C1DiscontCropTol: Additional epsilon to crop surfaces at intersection.

Returns: Constructed sweep trivariate.

Description: Constructs a sweep trivariate using the following geometries:

1. CrossSection - defines the basic cross section of the sweep. Must be in the XY plane. Can be several surfaces to be blended along the Axis.
2. Axis - a 3D curve the CrossSection will be swept along such that the Axis normal aligns with the Y axis of the cross section. If Axis is linear (i.e. no normal), the normal is picked randomly or to fit the non linear part of the Axis (if any). Can be C^1 discontinuous.
3. Scale - a scaling curve for the sweep, If NULL a scale of Scale is used.
4. Frame - a curve or a vector that specifies the orientation of the sweep by specifying the axes curve's binormal. If Frame is a vector, it is a constant binormal. If Frame is a curve (FrameOption == 2), it is assumed to be a vector field binormal. If NULL, it is computed from the Axis curve's pseudo Frenet frame, that minimizes rotation.

This operation is only an approximation. See CagdSweepAxisRefine for a tool to refine the Axis curve and improve accuracy.

See also: CagdSweepSrfError, CagdSweepAxisRefine, CagdSweepSrf, CagdSweepSrfC1, , TrivSweepTV,

13.2.169 TrivSweepTVError (trivswep.c:1045)

sweep

trivariate constructors

approximation error

```
CagdSrfStruct *TrivSweepTVError(const TrivTVStruct *SweepTV,
                                const CagdSrfStruct *CrossSection,
                                const CagdCrvStruct *Axis,
                                const CagdCrvStruct *ScalingCrv,
                                CagdRType Scale)
```

SweepTV: Computed approximated sweep trivariate, given the params.

CrossSection: Of the constructed sweep trivar. If more than one curve is given as a linked list of srfs, the cross sections are modified as we progresses along the sweep, blending between the cross sections so that last cross section is used in the last parameter value of the Axis.

Axis: Of the constructed sweep surface.

ScalingCrv: Optional scale or profile curve. NULL if none. The Y axis of this curve is using as scaling as the function of the parametrization. X axis can be used to define a 2D parametric curve so this curve can be displayed but X is ignored by this function.

Scale: If no Scaling Crv, Scale is used to apply a fixed scale on the CrossSection curve.

Returns: A scalar field represented the error as L2 distance square from the (boundary of the) precise sweep trivariate.

Description: Computes an upper bound on the error as a scalar function over the sweep trivariate approximation, SweepTV, as a bivariate field over the domain of the boundary of SweepTV, given the sweep surface construction parameters. This operation is only an approximation. See CagdSweepAxisRefine for a tool to refine the Axis curve and improve accuracy. Error is computed as:

```
SweepErr(u, v) = || CrossSection(u) * Scale(v) || ^ 2  -
                 || SweepTVBndry(u, v) - Axis(v) || ^ 2
```

where scale(v) is constant Scale if no scaling <crv.

See also: TrivSweepTV, CagdSweepAxisRefine, CagdSweepSrfError,

13.2.170 TrivSwungAlgSumTV (trivcnst.c:113)

```
TrivTVStruct *TrivSwungAlgSumTV(const CagdCrvStruct *Crv,
                                const CagdSrfStruct *Srf)
```

Crv, Srf: A curve and a surface to swung algebraically into a trivariate.

Returns: A trivariate represent their swung algebraic sum.

Description: Adds up algebraically the given curve and surface , C1(r) and S2(s, t), as swung trivariate (The NURBs book', by Piegl and Tiller, pp 455):
T(r, s, t) = (x1(r) x2(s, t), y1(r) x2(s, t), y2(s, t))
See also: TrivAlgebraicProdTV, TrivAlgebraicSumTV, SymbSwungAlgSumSrf, , SymbAlgebraicProdSrf, SymbAlgebraicSumSrf,

13.2.171 TrivTV2CtrlMesh (trivmesh.c:24)

control mesh

```
CagdPolylineStruct *TrivTV2CtrlMesh(const TrivTVStruct *Trivar)
```

Trivar: To extract a control mesh from.

Returns: The control mesh of Srf.

Description: Extracts the control mesh of a surface as a list of polylines in E3.

13.2.172 TrivTVAdd (symb_tv.c:33)

addition

symbolic computation

```
TrivTVStruct *TrivTVAdd(const TrivTVStruct *TV1, const TrivTVStruct *TV2)
```

TV1, TV2: Two trivar to add up coordinate-wise.

Returns: The summation of TV1 + TV2 coordinate-wise.

Description: Given two trivars - add them coordinate-wise. The two trivars are promoted to same point type before the multiplication can take place. Furthermore, order and continuity are matched as well.
See also: MvarMVAdd, TrivTVSub, TrivTVMult,

13.2.173 TrivTVBBox (triv_aux.c:407)

bbox

bounding box

```
CagdBBoxStruct *TrivTVBBox(const TrivTVStruct *TV, CagdBBoxStruct *BBox)
```

TV: To compute a bounding box for.

BBox: Where bounding information is to be saved.

Returns: BBox.

Description: Computes a bounding box for a trivariate freeform function.

13.2.174 TrivTVBlockEvalDone (triveval.c:1835)

```
void TrivTVBlockEvalDone(TrivTVBlockEvalGenInfoStruct *TVBlock)
```

TVBlock: Pointer to struct that contains all parameters for block evaluation.

Returns: void

Description: Free all auxiliary memory used by this block evaluation procedure.
See also: TrivTVBlockEvalInit, TrivTVBlockEvalSetMesh, TrivTVBlockEvalOnce,

13.2.175 TrivTVBlockEvalInit (triveval.c:1566)

```
TrivTVBlockEvalGenInfoStruct *TrivTVBlockEvalInit(CagdRType *UKnotVector,
                                                  CagdRType *VKnotVector,
                                                  CagdRType *WKnotVector,
                                                  int Lengths[3],
                                                  int Orders[3],
                                                  int BlockSizes[3],
                                                  CagdPType *Params,
                                                  int NumOfParams[3])
```

UKnotVector: U Knot sequence defining the spline space in U.

VKnotVector: V Knot sequence defining the spline space in V.

WKnotVector: W Knot sequence defining the spline space in W.

Lengths: Lengths of Mesh in the U,V,W directions.

Orders: Of the spline space in the U,V,W Directions.

BlockSizes: Of the evaluation block sizes in U,V,W Directions.

Params: At which to evaluate and compute the volume functions. This vector is of size IRIT_MAX(NumOfparams[0],[1],[2]).

NumOfParams: Size of Params vector, in U, V, W.

Returns: Pointer to struct that contains all parameters for block evaluation.

Description: Initialize the computation of evaluations of blocks of parameter values for the given trivariate space, as prescribed by U/V/WKnotVectors and Orders, Lengths, and BlockSizes, at the requested NumOfParams parameter values, Params. All parameters below should stay valid for the duration of the block evaluation.
See also: TrivTVBlockEvalSetMesh, TrivTVBlockEvalOnce, TrivTVBlockEvalDone,

13.2.176 TrivTVBlockEvalOnce (triveval.c:1675)

```
TrivTVBlockEvalStruct *TrivTVBlockEvalOnce(
                                        TrivTVBlockEvalGenInfoStruct *TVBlock,
                                        int i,
                                        int j,
                                        int k)
```

TVBlock: Pointer to struct that contains all parameters for block evaluation.

i, j, k: Index of current block evaluation. This triple index is between 0 and NumOfParams[l]/BlockSize[l], in each dimension.

Returns: Evaluated block.

Description: Computes evaluations of one block of parameter values for the given trivariate space, as prescribed by i, j, k.
See also: TrivTVBlockEvalInit, TrivTVBlockEvalSetMesh, TrivTVBlockEvalDone,

13.2.177 TrivTVBlockEvalSetMesh (triveval.c:1646)

```
void TrivTVBlockEvalSetMesh(TrivTVBlockEvalGenInfoStruct *TVBlock,
                            CagdPType *Mesh)
```

TVBlock: Pointer to struct that contains all parameters for block evaluation.

Mesh: Provide the current mesh.

Returns: void

Description: Sets the current mesh of size Lengths[1] * Lengths[2] * Lengths[3] (see TrivTVBlockEvalInit) to use in the block evaluation.
See also: TrivTVBlockEvalInit, TrivTVBlockEvalOnce, TrivTVBlockEvalDone,

13.2.178 TrivTVBlossomDegreeRaise (trivrais.c:607)

```
TrivTVStruct *TrivTVBlossomDegreeRaise(const TrivTVStruct *TV,
                                       TrivTVDirType Dir)
```

TV: To raise it degree by one.

Dir: Direction of degree raising. Either U, V or W.

Returns: A trivariate with one degree higher in direction Dir, representing the same geometry as TV.

Description: Returns a new Bspline trivariate, identical to the original but with one degree higher, in the requested direction Dir.
See also: TrivBspTVDegreeRaise, TrivBzrTVDegreeRaise, TrivTVDegreeRaise,

13.2.179 TrivTVBlossomDegreeRaiseN (trivrais.c:406)

```
TrivTVStruct *TrivTVBlossomDegreeRaiseN(const TrivTVStruct *TV,
                                        int NewUOrder,
                                        int NewVOrder,
                                        int NewWOrder)
```

TV: To raise it degree by one.

NewUOrder: New degree raised order in U.

NewVOrder: New degree raised order in V.

NewWOrder: New degree raised order in W.

Returns: A trivariate with one degree higher in direction Dir, representing the same geometry as TV.

Description: Returns a new Bspline trivariate, identical to the original but with one degree higher, in the requested direction Dir.
See also: TrivBspTVDegreeRaise, TrivBzrTVDegreeRaise, TrivTVDegreeRaise,

13.2.180 TrivTVCopy (triv_gen.c:292)

trivariates

```
TrivTVStruct *TrivTVCopy(const TrivTVStruct *TV)
```

TV: Trivariate to duplicate

Returns: Duplicated trivariate.

Description: Allocates and duplicates all slots of a trivariate structure.

13.2.181 TrivTVCopyList (triv_gen.c:357)

trivariates

```
TrivTVStruct *TrivTVCopyList(const TrivTVStruct *TVList)
```

TVList: List of trivariates to duplicate.

Returns: Duplicated list of trivariates.

Description: Duplicates a list of trivariate structures.

13.2.182 TrivTVCrossProd (symb_tv.c:280)

product

cross product

symbolic computation

```
TrivTVStruct *TrivTVCrossProd(const TrivTVStruct *TV1, const TrivTVStruct *TV2)
```

TV1, TV2: Two trivar to multiply in R3 and compute cross product for.

Returns: A vector trivar representing the cross product of TV1 x TV2.

Description: Given two trivars - computes their cross product in R3. Returned trivar is a trivar representing the cross product of the two given trivars.
See also: MvarMVCrossProd, TrivTVDotProd, TrivTVVecDotProd, TrivTVMultScalar, , TrivTVInvert, TrivTVVecCrossProd,

13.2.183 TrivTVDegreeRaise (trivrais.c:38)

degree raising

```
TrivTVStruct *TrivTVDegreeRaise(const TrivTVStruct *TV, TrivTVDirType Dir)
```

TV: To raise its degree.

Dir: Direction of degree raising. Either U, V or W.

Returns: A trivariate with same geometry as TV but with one degree higher.

Description: Returns a new trivariate representing the same curve as TV but with its degree raised by one, in Dir direction.

13.2.184 TrivTVDegreeRaiseN (trivrais.c:78)

degree raising

```
TrivTVStruct *TrivTVDegreeRaiseN(const TrivTVStruct *TV,
                                 TrivTVDirType Dir,
                                 int NewOrder)
```

TV: To raise its degree.

Dir: Direction of degree raising. Either U, V or W.

NewOrder: New order to raise TV in direction Dir.

Returns: A trivariate with same geometry as TV but with one degree higher.

Description: Returns a new trivariate representing the same curve as TV but with its degree raised to a NewOrder in Dir direction.

13.2.185 TrivTVDerive (triv_der.c:37)

trivariates

```
TrivTVStruct *TrivTVDerive(const TrivTVStruct *TV, TrivTVDirType Dir)
```

TV: Trivariate to differentiate.

Dir: Direction of differentiation. Either U or V or W.

Returns: Differentiated trivariate in direction Dir.

Description: Given a trivariate, computes its partial derivative trivariate in direction Dir.
See also: TrivBzrTVDerive, TrivBspTVDerive, TrivTVDeriveScalar,

13.2.186 TrivTVDeriveScalar (triv_der.c:68)

trivariates

```
TrivTVStruct *TrivTVDeriveScalar(const TrivTVStruct *TV, TrivTVDirType Dir)
```

TV: Trivariate to differentiate.

Dir: Direction of differentiation. Either U or V or W.

Returns: Differentiated trivariate in direction Dir.

Description: Given a trivariate, computes its partial derivative trivariate in direction Dir.
See also: TrivBzrTVDeriveScalar, TrivBspTVDeriveScalar, TrivTVDerive,

13.2.187 TrivTVDomain (triv_aux.c:42)

trivariates

```
void TrivTVDomain(const TrivTVStruct *TV,
                  CagdRType *UMin,
                  CagdRType *UMax,
                  CagdRType *VMin,
                  CagdRType *VMax,
                  CagdRType *WMin,
                  CagdRType *WMax)
```

TV: Trivariate function to consider.

UMin, UMax: U Domain of TV will be placed herein.

VMin, VMax: V Domain of TV will be placed herein.

WMin, WMax: W Domain of TV will be placed herein.

Returns: void

Description: Given a tri-variate, returns its parametric domain.
See also: TrivTVSetDomain, TrivTVSetDomain2,

13.2.188 TrivTVDotProd (symb_tv.c:210)

product

dot product

symbolic computation

```
TrivTVStruct *TrivTVDotProd(const TrivTVStruct *TV1, const TrivTVStruct *TV2)
```

TV1, TV2: Two trivar to multiply and compute a dot product for.

Returns: A scalar trivar representing the dot product of TV1 . TV2.

Description: Given two trivars - computes their dot product. Returned trivar is a scalar trivar representing the dot product of the two given trivars.
See also: MvarMVDotProd, TrivTVMult, TrivTVVecDotProd, TrivTVMultScalar, , TrivTVInvert, TrivTVCrossProd, TrivTVVecCrossProd,

13.2.189 TrivTVEval2Malloc (triveval.c:445)

evaluation

trivariates

```
CagdRType *TrivTVEval2Malloc(const TrivTVStruct *TV,
                             CagdRType u,
                             CagdRType v,
                             CagdRType w)
```

TV: To evaluate at given (u, v, w) parametric location.

u, v, w: Parametric location to evaluate TV at.

Returns: A vector holding all the coefficients of all components of the trivariate's point type. If for example trivariate point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1). Returned value is allocated dynamically.

Description: This function is the same as TrivTVEval2Todata above, while the evaluated point is allocated in dynamic memory. Evaluates the given tensor product trivariate at a given point, by extracting an isoparametric surface along u and evaluating (v, w) in it.
See also: TrivTVEvalToData, TrivTVEval2ToData,

13.2.190 TrivTVEval2ToData (triveval.c:400)

evaluation

trivariates

```
void TrivTVEval2ToData(const TrivTVStruct *TV,
                       CagdRType u,
                       CagdRType v,
                       CagdRType w,
                       CagdRType *Pt)
```

TV: To evaluate at given (u, v, w) parametric location.

u, v, w: Parametric location to evaluate TV at.

Pt: A vector holding all the coefficients of all components of the trivariate's point type. If for example trivariate point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Returns: void

Description: This function is the same as TrivTVEvalToData above. Evaluates the given tensor product trivariate at a given point, by extracting an isoparamteric surface along u and evaluating (v, w) in it.
See also: TrivTVEvalToData, TrivTVEval2Malloc,

13.2.191 TrivTVEvalJacobian (triv_der.c:414)

```
CagdRType TrivTVEvalJacobian(const TrivTVStruct *TV,
                             CagdRType u,
                             CagdRType v,
                             CagdRType w)
```

TV: Trivariate to compute its Jacobian. Note rationals are not supported

u, v, w: The location to evaluate the Jacobian at.

Returns: Evaluated Jacobian of TV at (u, v, w).

Description: Evaluates the Jacobian of the given trivariates at the specified parametric location.
See also: MvarTVRglrIsNegJacobian, MvarCalculateTVJacobian, TrivComputeJacobian, , TrivTVEvalJacobianApprox,

13.2.192 TrivTVEvalJacobianApprox (triv_der.c:457)

```
CagdRType TrivTVEvalJacobianApprox(const TrivTVStruct *TV,
                                   CagdRType u,
                                   CagdRType v,
                                   CagdRType w,
                                   CagdRType Eps)
```

TV: Trivariate to compute its Jacobian.

u, v, w: The location to evaluate the Jacobian at.

Eps: Tolerance of approximation.

Returns: Evaluated Jacobian of TV at (u, v, w).

Description: Evaluates the Jacobian of the given trivariates at the specified parametric location.
See also: MvarTVRglrIsNegJacobian, MvarCalculateTVJacobian, TrivComputeJacobian,

13.2.193 TrivTVEvalMalloc (triveval.c:360)

evaluation

trivariates

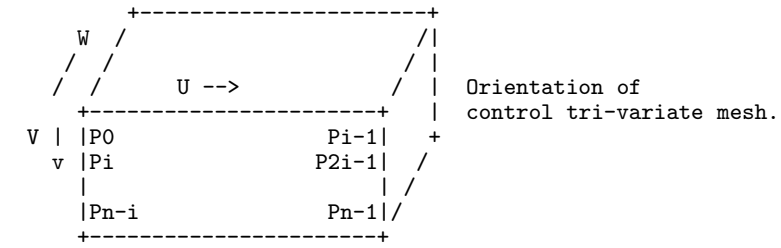
```
CagdRType *TrivTVEvalMalloc(const TrivTVStruct *TV,
                            CagdRType u,
                            CagdRType v,
                            CagdRType w)
```

TV: To evaluate at given (u, v, w) parametric location.

u, v, w: Parametric location to evaluate TV at.

Returns: A vector holding all the coefficients of all components of the trivariate's point type. If for example trivariate point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1). Returned value is allocated dynamically.

Description: Evaluates the given tensor product trivariate at a given point, by extracting an isoparamteric surface along w and evaluating (u,v) in it. Evaluated point is returned in allocated dyanmic memory.



See also: TrivTVEvalToData,

13.2.194 TrivTVEvalToData (triveval.c:84)

evaluation

trivariates

```
void TrivTVEvalToData(const TrivTVStruct *TV,
                      CagdRType u,
                      CagdRType v,
                      CagdRType w,
                      CagdRType *Pt)
```

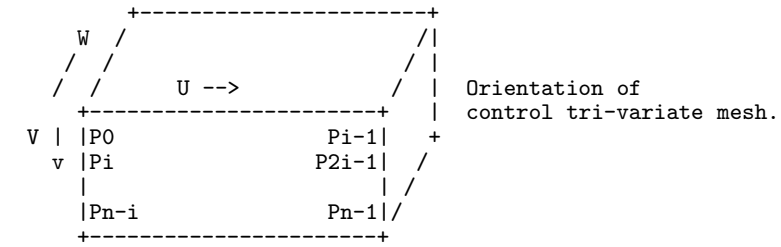
TV: To evaluate at given (u, v, w) parametric location.

u, v, w: Parametric location to evaluate TV at.

Pt: A vector holding all the coefficients of all components of the trivariate's point type. If for example trivariate point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Returns: void

Description: Evaluates the given tensor product trivariate at a given point, by computing the basis functions in U/V/W and blending them with the 3D matrix of relevant control points.



See also: TrivTVEvalMalloc,

13.2.195 TrivTVEvalToDataOld (triveval.c:231)

evaluation

trivariates

```
void TrivTVEvalToDataOld(const TrivTVStruct *TV,
                         CagdRType u,
                         CagdRType v,
                         CagdRType w,
                         CagdRType *Pt)
```

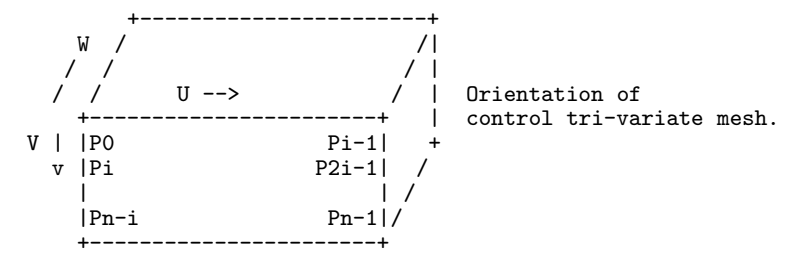
TV: To evaluate at given (u, v, w) parametric location.

u, v, w: Parametric location to evaluate TV at.

Pt: A vector holding all the coefficients of all components of the trivariate's point type. If for example trivariate point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Returns: void

Description: Evaluates the given tensor product trivariate at a given point, by extracting an isoparamteric surface along w and evaluating (u,v) in it. evaluated point is saved in Pt.



See also: TrivTVEvalMalloc,

13.2.196 TrivTVFillet (triv_fillet.c:4207)

trivariates
blending
filleting

```
TrivTVStruct *TrivTVFillet(const CagdSrfStruct *Srf1List,
                           const CagdSrfStruct *Srf2List,
                           CagdRType RailDist,
                           int R1Orient,
                           int R2Orient,
                           CagdRType TanScale,
                           int ApproxCrvsCtlPts,
                           CagdRType Tol,
                           CagdRType NumerTol,
                           TrivFilletingMethodType FilletingMethod,
                           CagdBType PreciseFillet,
                           CagdSrfStruct **PrimSrfs1,
                           CagdSrfStruct **PrimSrfs2)
```

Srf1List, Srf2List: Two list of surfaces to construct a fillet in between each pair of them. *

RailDist: The distance of the rail curves from the intersection curve of Srf1 and Srf2

R1Orient, R2Orient: +/-1 for left/right orientation of the corresponding rail curve. 0 to choose the option that results with a longer rail curve.

TanScale: The desired magnitude of the fillet's tangents that connect it with Srf1 and Srf2.

ApproxCrvsCtlPts: The number of control points used to approximate the curves used for fillet construction.

Tol: The tolerance used in the filleting algorithm comparisons.

NumerTol: The numeric tolerance used when using the multivariate solver.

FilletingMethod: The required filleting method.

PreciseFillet: If true, the primary surfaces of the fillet are computed precisely, using surface-surface composition. Otherwise, the primary surafces are approximated, using least square fitting.

PrimSrfs1: If not NULL, a list of the fillets' boundaries on Srf1List will be stored here.

PrimSrfs2: If not NULL, a list of the fillets' boundaries on Srf2List will be stored here.

Returns: The (list of) fillet trivariate(s).

Description: Constructs a (list of) fillet trivariate(s) between the two given surfaces. The fillet meets with the surfaces with G1 continuity, and its boundary curves are the intersection curve of Srf1 and Srf2, and two rail curves that are computed as an approximate Euclidean offest of the intersection curve on each of the surfaces.

See also: TrivBlendFilletProperties,

13.2.197 TrivTVFree (triv_gen.c:405)

trivariates

```
void TrivTVFree(TrivTVStruct *TV)
```

TV: Trivariate to free.

Returns: void

Description: Deallocates and frees all slots of a trivariate structure.

13.2.198 TrivTVFreeList (triv_gen.c:441)

trivariates

```
void TrivTVFreeList(TrivTVStruct *TVList)
```

TVList: Trivariate list to free.

Returns: void

Description: Deallocates and frees a list of trivariate structures.

13.2.199 TrivTVFromSrfs (trivstrv.c:161)

trivar constructors

```
TrivTVStruct *TrivTVFromSrfs(const CagdSrfStruct *SrfList,
                             int OtherOrder,
                             CagdEndConditionType OtherEC,
                             IrtRType *OtherParamVals)
```

SrfList: List of surfaces to construct a trivariate with.

OtherOrder: Other, third, order of trivariate.

OtherEC: End condition in the other, third, trivar direction.

OtherParamVals: If not NULL, updated with other direction set parameters of the surfaces in the new trivar.

Returns: Constructed trivariate from surfaces.

Description: Constructs a trivariate using a set of surfaces. Surfaces are made to be compatible and then each is substituted into the new trivariate's mesh as a row. If OtherOrder is less than the number of surfaces, number of surfaces is used. A knot vector is formed with uniform open end for the other direction, so created TV interpolates the first and last surfaces only, if OtherOrder is greater than 2.

See also: TrivTVInterpolateSrfs,

13.2.200 TrivTVGenAdjacencyInfo (triv_adj.c:166)

```
void TrivTVGenAdjacencyInfo(TrivTVStruct *TVList,
                            TrivTVAdjSrfCmpFuncType SrfCmpFuncPtr,
                            TrivTVAdjSrfCmpFuncType SrfCmpApxFuncPtr,
                            CagdRType SrfGapTol,
                            CagdBType MarkUVWBndry)
```

TVList: A List of trivariates to tag the adjacency information.

SrfCmpFuncPtr: A pointer to the function to compare two surfaces. Can be NULL for a default comparison function.

SrfCmpApxFuncPtr: A pointer to the function to approximately compare two surfaces. Can be NULL to ignore.

SrfGapTol: A tolerance value for the gap between two surfaces.

MarkUVWBndry: If TRUE, we mark if six MIN/MAX surfaces of trivariates are the boundary surfaces or not and save them in Bndry attribute. Mod value is set to 0x100 for the approximate adjacent surface. If all of the boundary surfaces are internal (adjacent to other surfaces), mark them as TRIV_NO_BNDRY.

Returns: void

Description: Given a list of trivariates, find all the adjacent trivariates of each trivariate. Each trivariate can have up to six neighbor trivariates, each of which is adjacent to UMin/UMax/VMin/VMax/WMin/WMax boundary surface of the given trivariate. A unique id is given to each trivariate and then the ids of the adjacent trivariate are saved in the 'TrivAdjID[UMin|UMax|VMin|VMax| WMin|WMax]' attribute in the given trivariate as follows: 'TrivAdjID[UMin|UMax|VMin|VMax|WMin|WMax] i, mod', where i denotes the id of the neighboring trivariate in the specified direction and mod represents the modification flag returning from the user-specified surface comparison function. The id of each trivariate is also saved in the attribute 'TrivID'. When SrfCmpApxFuncPtr is not NULL, simple comparison tests from SrfCmpApxFuncPtr are done prior to checking the compatibility between the boundary surfaces. Only the surfaces that approximately match are compared to see whether they are compatible or not. If two trivariates are only approximately matched, the id of adjacent trivariate is saved with the mod value = 0x100. When MarkUVWBndry is true, we mark whether UMin|UMax|VMin|VMax|WMin|WMax surface of each trivariate is a boundary surface or not and saved in Bndry attribute of the trivariate. If the trivariate has 6 trivariates adjacent to it, mark the trivariate as TRIV_NO_BNDRY mask is saved.

See also: TrivTVAdjSrfCmpFuncType, CagdSrfsSame2, CagdSrfsSame3, CagdSrfsSame4, TrivTVGenAdjacencyInfo2, TrivTVAdjTestSrfsApxSameMdfy,

13.2.201 TrivTVGenAdjacencyInfo2 (triv_adj.c:236)

```
void TrivTVGenAdjacencyInfo2(TrivTVStruct **TVVec,
                             int NumTVs,
                             TrivTVAdjSrfCmpFuncType SrfCmpFuncPtr,
                             TrivTVAdjSrfCmpFuncType SrfCmpApxFuncPtr,
                             CagdRType SrfGapTol,
                             CagdBType MarkUVWBndry)
```

TVVec: A vector of the trivariates' info.

NumTVs: Size of TVVec.

SrfCmpFuncPtr: A pointer to the function to compare two surfaces. Can be NULL for a default comparison function.

SrfCmpApxFuncPtr: A pointer to the function to approximately compare two surfaces. Can be NULL to ignore.

SrfGapTol: A tolerance value for the gap between two surfaces.

MarkUVWBndry: If TRUE, we mark if six MIN/MAX surfaces of trivariates are the boundary surfaces or not and save them in Bndry attribute. If all of the boundary surfaces are internal (adjacent to other surfaces), mark them as TRIV_NO_BNDRY

Returns: void

Description: Given a vector of trivariates, find all the adjacent trivariates of each trivariate. Each trivariate can have up to six neighbor trivariates, each of which is adjacent to UMin/UMax/VMin/VMax/WMin/WMax boundary surface of the given trivariate. A unique id is given to each trivariate and then the ids of the adjacent trivariate are saved in the 'TrivAdjID[UMin|UMax|VMin|VMax| WMin|WMax]' attribute in the given trivariate as follows: 'TrivAdjID[UMin|UMax|VMin|VMax|WMin|WMax] i, mod', where i denotes the id of the neighboring trivariate in the specified direction and mod represents the modification flag returning from the user-specified surface comparison function. The id of each trivariate is also saved in the attribute 'TrivID'. Simple comparison tests defined in SrfCmpApxFuncPtr are done prior to checking the compatibility between the boundary surfaces. Only the surfaces that approximately match are re-compared with SrfCmpFuncPtr to to see whether they are compatible or not. If two trivariates are only approximately matched and not precisely matched, the id of adjacent trivariate is saved with the mod value = 0x100. When MarkUVWBndry is true, we mark whether UMin|UMax|VMin|VMax|WMin|WMax surface of each trivariate is a boundary surface or not and saved in Bndry attribute of the trivariate. If the trivariate has 6 trivariates adjacent to it, mark the trivariate as TRIV_NO_BNDRY mask is saved.

See also: TrivTVAdjSrfCmpFuncType, CagdSrfsSame2, CagdSrfsSame3, CagdSrfsSame4, , TrivTVGenAdjacencyInfo, TrivTVAdjTestSrfsApxSameMdfy,

13.2.202 TrivTVInterpPts (trinterp.c:77)

interpolation

least square approximation

```
TrivTVStruct *TrivTVInterpPts(const TrivTVStruct *PtGrid,
                              int UOrder,
                              int VOrder,
                              int WOrder,
                              int TVUSize,
                              int TVVSize,
                              int TVWSize)
```

PtGrid: Input data grid as a trivariate.

UOrder: Of the to be created trivariate.

VOrder: Of the to be created trivariate.

WOrder: Of the to be created trivariate.

TVUSize: U size of the to be created trivariate. Must be at least as large as the array PtGrid.

TVVSize: V size of the to be created trivariate. Must be at least as large as the array PtGrid.

TVWSize: W size of the to be created trivariate. Must be at least as large as the array PtList.

Returns: Constructed interpolating/approximating trivariate.

Description: Given a set of points, PtList, computes a Bspline trivariate of order UOrder by VOrder by WOrder that interpolates or least square approximates the given set of points. PtGrid is a trivariate whose point data is employed toward the fitting. PtGrid also prescribes the parametric domain of the result. lists. The size of the control mesh of the resulting Bspline trivariate Trivar defaults to the number of points in PtGrid (if TV?Size = 0). However, either numbers can smaller to yield a least square approximation of the gievn data set.

See also: TrivInterpTrivar, TrivTVInterpolate,

13.2.203 TrivTVInterpScatPts (trinterp.c:302)

interpolation

least square approximation

```
TrivTVStruct *TrivTVInterpScatPts(const CagdCtlPtStruct *PtList,
                                  int USize,
                                  int VSize,
                                  int WSize,
                                  int UOrder,
                                  int VOrder,
                                  int WOrder,
                                  CagdRType *UKV,
                                  CagdRType *VKV,
                                  CagdRType *WKV)
```

PtList: A NULL terminating array of linked list of points.

USize: U size of the to be created trivariate.

VSize: V size of the to be created trivariate.

WSize: W size of the to be created trivariate.

UOrder: Of the to be created trivariate.

VOrder: Of the to be created trivariate.

WOrder: Of the to be created trivariate.

UKV: Expected knot vector in U direction, NULL for uniform open.

VKV: Expected knot vector in V direction, NULL for uniform open.

WKV: Expected knot vector in W direction, NULL for uniform open.

Returns: Constructed interpolating/approximating trivariate.

Description: Given a set of scattered points, PtList, computes a Bspline trivariate of order UOrder by VOrder by WOrder that interpolates or least squares approximates the given set of scattered points. PtList is a NULL terminated lists of CagdPtStruct structs, with each point holding (u, v, w, x [, y[, z]]). That is, E4 points create an E1 scalar trivariate and E6 points create an E3 trivariate,

See also: TrivInterpTrivar, TrivTVInterpPts, TrivTVInterpolate, , BspSrfInterpScatPts,

13.2.204 TrivTVInterpolate (trinterp.c:121)

interpolation

least square approximation

```
TrivTVStruct *TrivTVInterpolate(const TrivTVStruct *PtGrid,
                                int ULength,
                                int VLength,
                                int WLength,
                                int UOrder,
                                int VOrder,
                                int WOrder)
```

PtGrid: Input data grid as a trivariate.

ULength: Requested length of control mesh of trivariate in U direction. If zero, length of PtGrid in U is used.

VLength: Requested length of control mesh of trivariate in V direction. If zero, length of PtGrid in V is used.

WLength: Requested length of control mesh of trivariate in W direction. If zero, length of PtGrid in W is used.

UOrder: Requested order of trivariate in U direction. If zero, order of PtGrid in U is used.

VOrder: Requested order of trivariate in V direction. If zero, order of PtGrid in V is used.

WOrder: Requested order of trivariate in W direction. If zero, order of PtGrid in W is used.

Returns: Constructed interpolating/approximating trivariate.

Description: Given a set of points on a box grid, PtGrid, the expected lengths U/V/WLength and orders U/V/WOrder of the Bspline trivariate, computes the Bspline trivariate's coefficients that interpolates or least square approximates the given set of points, PtGrid.

See also: TrivInterpTrivar, TrivTVInterpPts, TrivTVInterpScatPts,

13.2.205 TrivTVInterpolateSrfs (trivstrv.c:299)

trivar constructors

interpolation

```
TrivTVStruct *TrivTVInterpolateSrfs(const CagdSrfStruct *SrfList,
                                    int OtherOrder,
                                    CagdEndConditionType OtherEC,
                                    CagdParametrizationType OtherParam,
                                    IrtRType *OtherParamVals)
```

SrfList: List of surfaces to interpolate a trivariate through.

OtherOrder: Other, third, depth order of trivariate.

OtherEC: End condition in the other, third, trivar direction.

OtherParam: Currently only Chord length and uniform are supported.

OtherParamVals: If not NULL, updated with other direction set parameters of the surfaces in the new trivar.

Returns: Constructed trivariate from surfaces.

Description: Constructs a trivariate using a set of surfaces. Surfaces are made to be compatible and then the trivariate is fitted to interpolate them. If OtherOrder is less than the number of surfaces, number of surfaces is used. A knot vector is formed with OtherEC end conditions for the other direction.
See also: TrivTVFromSrfs,

13.2.206 TrivTVInterpolateSrfsChordLenParams (trivstrv.c:225)

surface constructors

```
CagdRType *TrivTVInterpolateSrfsChordLenParams(const CagdSrfStruct *SrfList)
```

SrfList: List of surfaces to consturct a trivariate volume with.

Returns: Vectors of parameters normalized to [0, 1] of parameters, of size of number of surfaces, allocated dynamically.

Description: Computes parameters to interpolate the given surfaces at, as a trivar. Estimate a middle point from each surface and set parameters based on chord length from each middle point to the next.
See also: TrivTVFromSrfs, CagdSrfInterpolateCrvs, CagdSrfInterpolateCrvs, CagdSrfInterpolateCrvsChordLenParams,

13.2.207 TrivTVInvert (symb_tv.c:139)

division

symbolic computation

reciprocal value

```
TrivTVStruct *TrivTVInvert(const TrivTVStruct *TV)
```

TV: A scalar trivar to compute a reciprocal value for.

Returns: A rational scalar trivar that is equal to the reciprocal value of TV.

Description: Given a scalar trivar, returns a scalar trivar representing the reciprocal values, by making it rational (if was not one) and flipping the numerator and the denominator.
See also: MvarCnvrtMVToTV, TrivTVDotProd, TrivTVVecDotProd, TrivTVMultScalar, , TrivTVMult, TrivTVCrossProd,

13.2.208 TrivTVIsMeshC0DiscontAt (triv_gen.c:1440)

```
CagdBType TrivTVIsMeshC0DiscontAt(const TrivTVStruct *TV,
                                  int Dir,
                                  CagdRType t)
```

TV: Multivariate to examine for C0 discontinuity.

Dir: Parametric direction to examine at.

t: Parameter value to examine at.

Returns: TRUE if TV has a C0 discontinuity at parameter t in direction Dir, FALSE otherwise.

Description: Examines the mesh at given parametric location for a C0 discontinuity.
See also: TrivTVMeshC0Continuous, TrivTVKnotHasC0Discont,

13.2.209 TrivTVIsMeshC1DiscontAt (triv_gen.c:1660)

```
CagdBType TrivTVIsMeshC1DiscontAt(const TrivTVStruct *TV,
                                  int Dir,
                                  CagdRType t)
```

TV: Multivariate to examine for C1 discontinuity.

Dir: Parametric direction to examine at.

t: Parameter value to examine at.

Returns: TRUE if TV has a C1 discontinuity at parameter t in direction Dir, FALSE otherwise.

Description: Examines the mesh at given parametric location for a C1 discontinuity.
See also: TrivTVMeshC1Continuous, TrivTVKnotHasC1Discont,

13.2.210 TrivTVKnotHasC0Discont (triv_gen.c:1290)

knot vectors
continuity
discontinuity

```
CagdBType TrivTVKnotHasC0Discont(const TrivTVStruct *TV,
                                 TrivTVDirType *Dir,
                                 CagdRType *t)
```

TV: To examine its potential discontinuity across Dir.

Dir: Direction to examine the discontinuity across. If Dir is set to TRIV_NO_DIR, all directions are searched for C0 discontinuities, and Dir will be updated with the C0 discontinuous direction, if any. If Dir is set to one of TRIV_CONST_U/V/W_DIR, only that directions are searched for the C0 discontinuities.

t: Where to put the parameter value (knot) that can be C0 discontinuous.

Returns: TRUE if found a C0 discontinuity, FALSE otherwise.

Description: Scans the given knot vector of the given trivariate for a potential C0 discontinuity. Looks for multiplicities in the knot sequence and then examine the mesh if indeed the mesh is discontinuous at that location. Assumes knot vectors has open end condition.
See also: TrivTVKnotHasC1Discont, BspKnotC1Discont, TrivTVMeshC1Continuous,

13.2.211 TrivTVKnotHasC1Discont (triv_gen.c:1499)

knot vectors
continuity
discontinuity

```
CagdBType TrivTVKnotHasC1Discont(const TrivTVStruct *TV,
                                 TrivTVDirType *Dir,
                                 CagdRType *t)
```

TV: To examine its potential discontinuity across Dir.

Dir: Direction to examine the discontinuity across.

t: Where to put the parameter value (knot) that can be C1 discontinuous.

Returns: TRUE if found a C1 discontinuity, FALSE otherwise.

Description: Scans the given knot vector of the given trivariate for a potential C1 discontinuity. Looks for multiplicities in the knot sequence and then examine the mesh if indeed the mesh is discontinuous at that location. Assumes knot vectors has open end condition.
See also: TrivTVKnotHasC0Discont, BspKnotC1Discont, TrivTVMeshC1Continuous,

13.2.212 TrivTVListBBox (triv_aux.c:431)

bbox
bounding box

```
CagdBBoxStruct *TrivTVListBBox(const TrivTVStruct *TVs, CagdBBoxStruct *BBox)
```

TVs: To compute a bounding box for.

BBox: Where bounding information is to be saved.

Returns: Bbox.

Description: Computes a bounding box for a list of trivariate freeform function.

13.2.213 TrivTVListMatTransform (triv_gen.c:679)

scaling
rotation
translation
transformations

```
TrivTVStruct *TrivTVListMatTransform(const TrivTVStruct *TVs, CagdMType Mat)
```

TVs: To be transformed.

Mat: Defining the transformation.

Returns: Transformed TVs.

Description: Transforms the given list of TVs as specified by homogeneous matrix Mat.
See also: TrivTVMatTransform, TrivTVMatTransform2, CagdSrfListMatTransform,

13.2.214 TrivTVMatTransform (triv_gen.c:632)

multi-variates

```
TrivTVStruct *TrivTVMatTransform(const TrivTVStruct *TV, CagdMType Mat)
```

TV: Multi-variate to transform.

Mat: Homogeneous transformation to apply to TV.

Returns: Transformed TV.

Description: Transforms the given TV as specified by homogeneous matrix Mat.
See also: TrivTVMatTransform2, TrivTVListMatTransform, CagdSrfListMatTransform,

13.2.215 TrivTVMatTransform2 (triv_gen.c:709)

trivariates

```
void TrivTVMatTransform2(TrivTVStruct *TV, CagdMType Mat)
```

TV: Trivariate to transform.

Mat: Homogeneous transformation to apply to TV.

Returns: void

Description: Transforms, in place, the given TV as specified by homogeneous matrix Mat.
See also: TrivTVMatTransform, TrivTVListMatTransform, CagdSrfListMatTransform,

13.2.216 TrivTVMergeScalar (symb_tv.c:626)

merge
symbolic computation

```
TrivTVStruct *TrivTVMergeScalar(const TrivTVStruct *TVW,
                                const TrivTVStruct *TVX,
                                const TrivTVStruct *TVY,
                                const TrivTVStruct *TVZ)
```

TVW: The weight component of new constructed trivariate, if have any.

TVX: The X component of new constructed trivariate.

TVY: The Y component of new constructed trivariate, if have any.

TVZ: The Z component of new constructed trivariate, if have any.

Returns: A new trivariate constructed from given scalar trivariates.

Description: Given a set of scalar trivariates, treat them as coordinates into a new trivariate. Assumes at least TVX is not NULL in which a scalar trivariate is returned. Assumes TVX/Y/Z/W are either E1 or P1 in which the weights are assumed to be identical and can be ignored if TVW exists or copied otherwise.
See also: TrivTVSplitScalar, TrivTVMergeScalar,

13.2.217 TrivTVMergeScalarN (symb_tv.c:518)

merge

symbolic computation

```
TrivTVStruct *TrivTVMergeScalarN(TrivTVStruct * const *TVVec, int NumTVs)
```

TVVec: A vector of scalar trivariates, TVVec[0] holds weights if any.

NumTVs: Number of trivariates in CrvVec.

Returns: A new trivariate constructed from given scalar trivariates.

Description: Given a vector of scalar trivariates, treat them as coordinates into a new vector trivariate. Assumes at least TVVec[1] is not NULL in which case a scalar trivariate is returned. Assumes TVVec[i] are either E1 or P1 in which case the weights are assumed to be identical and can be simply copied if exist.
See also: TrivTVMergeScalar, TrivTVSplitScalarN,

13.2.218 TrivTVMeshC0Continuous (triv_gen.c:1350)

```
CagdBType TrivTVMeshC0Continuous(const TrivTVStruct *TV,
                                 TrivTVDirType Dir,
                                 int Idx)
```

TV: To examine its potential discontinuity across Dir.

Dir: Direction to examine the discontinuity across.

Idx: Index where to examine the discontinuity.

Returns: TRUE if continuous there, FALSE otherwise.

Description: Examine the mesh of the given surface across direction Dir in index of mesh Index for a real discontinuity in the mesh. This index will typically be for a knot multiplicity potential discont.
See also: TrivTVKnotHasC0Discont, TrivTVIsMeshC0DiscontAt,

13.2.219 TrivTVMeshC1Continuous (triv_gen.c:1559)

```
CagdBType TrivTVMeshC1Continuous(const TrivTVStruct *TV,
                                 TrivTVDirType Dir,
                                 int Idx)
```

TV: To examine its potential discontinuity across Dir.

Dir: Direction to examine the discontinuity across.

Idx: Index where to examine the discontinuity.

Returns: TRUE if continuous there, FALSE otherwise.

Description: Examine the mesh of the given surface across direction Dir in index of mesh Index for a real discontinuity in the mesh. This index will typically be for a knot multiplicity potential discont.
See also: TrivTVKnotHasC1Discont, TrivTVIsMeshC1DiscontAt,

13.2.220 TrivTVMult (symb_tv.c:103)

product

symbolic computation

```
TrivTVStruct *TrivTVMult(const TrivTVStruct *TV1, const TrivTVStruct *TV2)
```

TV1, TV2: Two trivar to multiply coordinate-wise.

Returns: The product of TV1 * TV2 coordinate-wise.

Description: Given two trivars - multiply them coordinate-wise. The two trivars are promoted to same point type before the multiplication can take place.
See also: TrivTVDotProd, TrivTVVecDotProd, TrivTVScalarScale, TrivTVMultScalar, , TrivTVInvert,

13.2.221 TrivTVMultEval (triveval.c:1364)

```
CagdRType *TrivTVMultEval(CagdRType *UKnotVector,
                          CagdRType *VKnotVector,
                          CagdRType *WKnotVector,
                          int ULength,
                          int VLength,
                          int WLength,
                          int UOrder,
                          int VOrder,
                          int WOrder,
                          CagdPType *Mesh,
                          CagdPType *Params,
                          int NumOfParams,
                          int *RetSize,
                          CagdBspBasisFuncMultEvalType EvalType)
```

UKnotVector: U Knot sequence defining the spline space in U.

VKnotVector: V Knot sequence defining the spline space in V.

WKnotVector: W Knot sequence defining the spline space in W.

ULength: Length of Mesh in the U direction.

VLength: Length of Mesh in the V direction.

WLength: Length of Mesh in the W direction.

UOrder: Of the spline space in the U Direction.

VOrder: Of the spline space in the V Direction.

WOrder: Of the spline space in the W Direction.

Mesh: ULength * VLength * WLength control points, in R^3.

Params: At which to evaluate and compute the volume functions.

NumOfParams: Size of Params vector.

RetSize: Number of values returned per evaluation. 3 for position 9 for 1st derivative, etc.

EvalType: Type of evaluation requested: value (position), 1st derivative.

Returns: A vector of size NumOfParams * RetSize, holding the NumOfParams evaluation results, each of size RetSize. For position evaluation, RetSize = 3 and XYZ are returned. For 1st derivatives, RetSize = 9 and the Jacobian is returned, with dX/du, dX/dV, dX/dw first.

Description: Computes multiple evaluations of the given trivariate space, as prescribed by U/V/WKnotVectors and U/V/WOrders, U/V/WLengths, and Mesh, at the requested NumOfParams parameter values, Params.
See also:

13.2.222 TrivTVMultScalar (symb_tv.c:174)

product

symbolic computation

```
TrivTVStruct *TrivTVMultScalar(const TrivTVStruct *TV1,
                               const TrivTVStruct *TV2)
```

TV1, TV2: Two trivars to multiply.

Returns: A trivar representing the product of TV1 and TV2.

Description: Given two trivar - a vector curve TV1 and a scalar curve TV2, multiply all TV1's coordinates by the scalar curve TV2. Returned trivar is a trivar representing the product of the two given trivars.
See also: MvarMVMultScalar, TrivTVDotProd, TrivTVVecDotProd, TrivTVMult, , TrivTVCrossProd,

13.2.223 TrivTVNew (triv_gen.c:47)

trivariates

allocation

```
TrivTVStruct *TrivTVNew(TrivGeomType GType,
                        CagdPointType PType,
                        int ULength,
                        int VLength,
                        int WLength)
```

GType: Type of geometry the curve should be - B-spline, Bezier etc.

PType: Type of control points (E2, P3, etc.).

ULength: Number of control points in the U direction.

VLength: Number of control points in the V direction.

WLength: Number of control points in the W direction.

Returns: An uninitialized freeform trivariate.

Description: Allocates the memory required for a new trivariate.
See also: TrivBzrTVNew, TrivBspTVNew, TrivPwrTVNew,

13.2.224 TrivTVOfRev (trivtrev.c:47)

trivariate of revolution

trivariate constructors

```
TrivTVStruct *TrivTVOfRev(const CagdSrfStruct *Srf)
```

Srf: To create trivariate of revolution around Z with.

Returns: Trivariate of revolution.

Description: Constructs a trivariate of revolution around the Z axis of given surface. Resulting trivariate will be a B-spline trivariate, while input may be either a B-spline or a Bezier surface.
See also: TrivTVOfRev2, TrivTVOfRevAxis, TrivTVOfRevPolynomialApprox,

13.2.225 TrivTVOfRev2 (trivtrev.c:239)

surface of revolution

surface constructors

```
TrivTVStruct *TrivTVOfRev2(const CagdSrfStruct *Srf,
                           CagdBType PolyApprox,
                           CagdRType StartAngle,
                           CagdRType EndAngle)
```

Srf: To create trivariate of revolution around Z with.

PolyApprox: TRUE for a polynomial approximation, FALSE for a precise rational construction.

StartAngle: Starting Angle to consider rotating Srf from, in degrees.

EndAngle: Terminating Angle to consider rotating Srf from, in degrees.

Returns: Trivariate of revolution.

Description: Constructs a trivariate of revolution around the Z axis of the given profile surface from StartAngle to EndAngle. Resulting trivariate will be a B-spline surface, while input may be either a B-spline or a Bezier surface.
See also: TrivTVOfRev, TrivTVOfRevAxis, TrivTVOfRevPolynomialApprox,

13.2.226 TrivTVOfRevAxis (trivtrev.c:176)

trivariate of revolution

trivariate constructors

```
TrivTVStruct *TrivTVOfRevAxis(const CagdSrfStruct *Srf,
                              const TrivV4DType AxisPoint,
                              const TrivV4DType AxisVector,
                              CagdBType PolyApprox)
```

Srf: To create trivariate of revolution around Axis.

AxisPoint: Of axis of rotation of Srf.

AxisVector: Of axis of rotation of Srf.

PolyApprox: TRUE to construct a polynomial approximation volume of revolution, FALSE to create precise rational volume.

Returns: Trivariate of revolution.

Description: Constructs a trivariate of revolution around vector Axis of given profile surface. Resulting trivariate will be a B-spline trivariate, while input may be either a B-spline or a Bezier surface.
See also: TrivTVOfRev, TrivTVOfRev2, TrivTVOfRevPolynomialApprox,

13.2.227 TrivTVOfRevPolynomialApprox (trivtrev.c:319)

trivariate of revolution

trivariate constructors

```
TrivTVStruct *TrivTVOfRevPolynomialApprox(const CagdSrfStruct *Srf)
```

Srf: To approximate a trivariate of revolution around Z with. Srf is assumed planar in a plane holding the Z axis.

Returns: Trivariate of revolution approximation.

Description: Constructs a trivariate of revolution around the Z axis of given profile surface. Resulting trivariate will be a B-spline trivariate, while input may be either a B-spline or a Bezier surface. Resulting trivariate will be a polynomial B-spline trivariate, approximating a trivariate of revolution using a polynomial circle approx. (See Faux & Pratt "Computational Geometry for Design and Manufacturing").
See also: TrivTVOfRev, TrivTVOfRev2, TrivTVOfRevAxis,

13.2.228 TrivTVOpenEnd (triv_gen.c:1094)

conversion

```
TrivTVStruct *TrivTVOpenEnd(const TrivTVStruct *TV)
```

TV: B-spline trivariate to convert to open end conditions.

Returns: A B-spline trivariate with open end conditions, representing the same geometry as TV.

Description: Converts an arbitrary B-spline trivariate to a B-spline trivariate with open end conditions.
See also: TrivCnvrtPeriodic2FloatTV, TrivCnvrtFloat2OpenTV,

13.2.229 TrivTVPointInclusion (triv_aux.c:881)

point inclusion

```
CagdBType TrivTVPointInclusion(TrivTVStruct *TV, const IrtPtType Pt)
```

TV: To compute point inclusion for.

Pt: Point to test if inside TV or not.

Returns: TRUE if inside, FALSE otherwise.

Description: Point inclusion test in a trivariate. TrivTVPointInclusionPrep must be called before this function is invoked for a valid result. Optimized for many queries of point inclusions over this TV.
See also: TrivTVPointInclusionPrep, TrivTVPointInclusionFree,

13.2.230 TrivTVPointInclusionFree (triv_aux.c:914)

point inclusion

```
void TrivTVPointInclusionFree(TrivTVStruct *TV)
```

TV: To free the auxiliary data used for TV point inclusion test.

Returns: void

Description: Freeing step for testing for point inclusions in a trivariate. Should be called after all TrivTV-PointInclusion calls are done.
See also: TrivTVPointInclusion, TrivTVPointInclusionPrep,

13.2.231 TrivTVPointInclusionPrep (triv_aux.c:834)

point inclusion

```
void TrivTVPointInclusionPrep(TrivTVStruct *TV, int n)
```

TV: To make the necessary steps for point inclusion test.

n: Sampling rate. The large this number the better the accuracy (near the boundaries) at the additional computational cost. Roughly the number of samples of the TV in each parametric direction and can vary between 10 to 100.

Returns: void

Description: Preparation step for testing for point inclusions in a trivariate. Must be called before TrivTV-PointInclusion is called. Optimized for many queries of point inclusions over this TV.
See also: TrivTVPointInclusion, TrivTVPointInclusionFree,

13.2.232 TrivTVRefineAtParams (triv_ref.c:41)

trivariates

```
TrivTVStruct *TrivTVRefineAtParams(const TrivTVStruct *TV,
                                   TrivTVDirType Dir,
                                   CagdBType Replace,
                                   CagdRType *t,
                                   int n)
```

TV: Trivariate to refine according to t in direction Dir.

Dir: Direction of refinement. Either U or V or W.

Replace: If TRUE t is a knot vector exactly in the length of the knot vector in direction Dir in TV and t simply replaces that knot vector. If FALSE, the knot vector in direction Dir in TV is refined by adding all the knots in t.

t: Knot vector to refine/replace the knot vector of TV in direction Dir.

n: Length of vector t.

Returns: The refined trivariate. Always a Bspline trivariate.

Description: Given a trivariate, refines it at the given n knots as defined by the vector t. If Replace is TRUE, the values replace the current knot vector. Returns pointer to refined TV (Note a Bezier trivariate will be converted into a Bspline trivariate).

13.2.233 TrivTVRegionFromTV (triv_aux.c:317)

trivariates

```
TrivTVStruct *TrivTVRegionFromTV(const TrivTVStruct *TV,
                                 CagdRType t1,
                                 CagdRType t2,
                                 TrivTVDirType Dir)
```

TV: To extract a sub-region from.

t1, t2: Domain to extract from TV, in parametric direction Dir.

Dir: Direction to extract the sub-region. Either U or V or W.

Returns: A sub-region of TV from t1 to t2 in direction Dir.

Description: Given a tri-variate, returns a sub-region of it.

13.2.234 TrivTVReverse2Dirs (triv_aux.c:595)

```
TrivTVStruct *TrivTVReverse2Dirs(const TrivTVStruct *TV,
                                 TrivTVDirType Dir1,
                                 TrivTVDirType Dir2)
```

TV: To construct a reverse TV for.

Dir1, Dir2: The two directions in TV to swap.

Returns: Reversed/swap trivariate.

Description: Reverse/swap the designated two dirs.
See also: TrivTVReverseDir,

13.2.235 TrivTVReverseDir (triv_aux.c:461)

```
TrivTVStruct *TrivTVReverseDir(const TrivTVStruct *TV, TrivTVDirType Dir)
```

TV: To construct a reverse TV for.

Dir: The direction in TV to reverse.

Returns: Reversed trivariate.

Description: Reverse the designated direction
See also: TrivTVReverse2Dirs,

13.2.236 TrivTVRtnlMult (symb_tv.c:324)

product

symbolic computation

```
TrivTVStruct *TrivTVRtnlMult(const TrivTVStruct *TV1X,
                             const TrivTVStruct *TV1W,
                             const TrivTVStruct *TV2X,
                             const TrivTVStruct *TV2W,
                             CagdBType OperationAdd)
```

TV1X: Numerator of first trivar.

TV1W: Denominator of first trivar. Can be NULL.

TV2X: Numerator of second trivar.

TV2W: Denominator of second trivar. Can be NULL.

OperationAdd: TRUE for addition, FALSE for subtraction.

Returns: The result of TV1X TV2W +/- TV2X TV1W.

Description: Given two trivars - multiply them using the quotient product rule:

`X = X1 W2 +/- X2 W1`

All provided trivars are assumed to be non rational scalar trivars. Returned is a non rational scalar trivar (CAGD_PT_E1_TYPE).
See also: MvarMVRtnlMult. TrivTVDotProd, TrivTVVecDotProd, TrivTVMultScalar , , TrivTVInvert,

13.2.237 TrivTVSetDomain (triv_aux.c:104)

```
TrivTVStruct *TrivTVSetDomain(TrivTVStruct *TV,
                              CagdRType UMin,
                              CagdRType UMax,
                              CagdRType VMin,
                              CagdRType VMax,
                              CagdRType WMin,
                              CagdRType WMax)
```

TV: To set its parametric domain.

UMin: Minimal domain's new U boundary.

UMax: Maximal domain's new U boundary.

VMin: Minimal domain's new V boundary.

VMax: Maximal domain's new V boundary.

WMin: Minimal domain's new W boundary.

WMax: Maximal domain's new W boundary.

Returns: Updated trivariate, in place.

Description: Affinely set the parametric domain of a trivar, in place.
See also: TrivTVDomain, TrivTVSetDomain2,

13.2.238 TrivTVSetDomain2 (triv_aux.c:176)

```
TrivTVStruct *TrivTVSetDomain2(TrivTVStruct *TV,
                               CagdRType Min,
                               CagdRType Max,
                               TrivTVDirType Dir)
```

TV: To set its parametric domain.

Min: Minimal domain's new boundary, in Dir.

Max: Maximal domain's new boundary, in Dir.

Dir: Direction of trivariate to change domain: U, V, or W.

Returns: Updated trivariate, in place.

Description: Affinely set the parametric domain of the trivar in Dir, in place.
See also: TrivTVDomain, TrivTVSetDomain,

13.2.239 TrivTVSplitScalar (symb_tv.c:439)

split

symbolic computation

```
void TrivTVSplitScalar(const TrivTVStruct *TV,
                       TrivTVStruct **TVW,
                       TrivTVStruct **TVX,
                       TrivTVStruct **TVY,
                       TrivTVStruct **TVZ)
```

TV: Trivariate to split.

TVW: The weight component of TV, if have any.

TVX: The X component of TV.

TVY: The Y component of TV, if have any.

TVZ: The Z component of TV, if have any.

Returns: void

Description: Given a trivariate splits it to its scalar component trivariates. Ignores all dimensions beyond the third, Z, dimension.
See also: TrivTVMergeScalar, TrivTVSplitScalar, TrivTVSplitScalarN,

13.2.240 TrivTVSplitScalarNToData (symb_tv.c:364)

split

symbolic computation

```
TrivTVStruct **TrivTVSplitScalarNToData(const TrivTVStruct *TV,
                                        TrivTVStruct **TVs)
```

TV: Trivariate to split.

TVs: A vector of scalar trivariates - components of TV.

Returns: A vector of scalar trivariates - components of TV.

Description: Given a trivariate, splits it to its scalar component trivariates.
See also: TrivTVSplitScalar, TrivTVMergeScalarN,

13.2.241 TrivTVSub (symb_tv.c:68)

subtraction

symbolic computation

```
TrivTVStruct *TrivTVSub(const TrivTVStruct *TV1, const TrivTVStruct *TV2)
```

TV1, TV2: Two trivar to subtract coordinate-wise.

Returns: The difference of TV1 - TV2 coordinate-wise.

Description: Given two trivars - subtract them coordinate-wise. The two trivars are promoted to same point type before the multiplication can take place. Furthermore, order and continuity are matched as well.
See also: MvarMVSub, TrivTVAdd, TrivTVMult,

13.2.242 TrivTVSubdivAtAllC0Discont (triv_sub.c:379)

```
TrivTVStruct *TrivTVSubdivAtAllC0Discont(const TrivTVStruct *TV)
```

TV: To subdivide at all C^0 discontinuity locations.

Returns: Trivariates that result from the subdivision.

Description: Subdivides the given trivariate at all C^0 potential discontinuity locations.
See also: TrivTVsSubdivAtParams, BspKnotAllC1Discont, , TrivTVsSubdivAtAllC0Discont, BspTVsSubdivAtAllDetectedLocations, , TrivTVsSubdivAtAllC1Discont,

13.2.243 TrivTVSubdivAtAllC1Discont (triv_sub.c:437)

```
TrivTVStruct *TrivTVSubdivAtAllC1Discont(const TrivTVStruct *TV)
```

TV: To subdivide at all C^1 discontinuity locations.

Returns: Trivariates that result from the subdivision.

Description: Subdivides the given trivariate at all C^1 potential discontinuity locations.
See also: TrivTVsSubdivAtParams, BspKnotAllC1Discont, , TrivTVsSubdivAtAllC0Discont, BspTVsSubdivAtAllDetectedLocations, , TrivTVsSubdivAtAllC1Discont,

13.2.244 TrivTVSubdivAtParam (triv_sub.c:29)

trivariates

```
TrivTVStruct *TrivTVSubdivAtParam(const TrivTVStruct *TV,
                                  CagdRType t,
                                  TrivTVDirType Dir)
```

TV: Trivariate to subdivide.

t: Parameter to subdivide at.

Dir: Direction of subdivision.

Returns: A list of two trivariates, result of the subdivision.

Description: Given a tri-variate, subdivides it at parameter value t in direction Dir.

13.2.245 TrivTVTransform (triv_gen.c:597)

trivariates

```
void TrivTVTransform(TrivTVStruct *TV,
                     const CagdRType *Translate,
                     CagdRType Scale)
```

TV: Trivariate to transform.

Translate: Translation factor. Can be NULL for non.

Scale: Scaling factor.

Returns: void

Description: Linearly transforms, in place, given TV as specified by Translate (first) and Scale (Second).

13.2.246 TrivTVVecDotProd (symb_tv.c:246)

product

dot product

symbolic computation

```
TrivTVStruct *TrivTVVecDotProd(const TrivTVStruct *TV, const CagdVType Vec)
```

TV: Trivar to multiply and compute a dot product for.

Vec: Vector to project TV onto.

Returns: A scalar trivar representing the dot product of TV . Vec.

Description: Given a trivar and a vector - computes their dot product. Returned trivar is a scalar trivar representing the dot product.
See also: MvarMVVecDotProd, TrivTVDotProd, TrivTVMult, TrivTVMultScalar, , TrivTVInvert, TrivTVCrossProd, TrivTVVecCrossProd,

13.2.247 TrivTVVolume (triv_aux.c:757)

bbox

bounding box

```
CagdRType TrivTVVolume(const TrivTVStruct *TV, CagdBType VolType)
```

TV: To compute its volume.

VolType: TRUE to integrate the surfaces with respect to the XY plane, FALSE to integrate the surfaces with respect to the origin,

Returns: The computed volume (can be negative if TV reversed).

Description: Computes the volume enclosed in the given trivariate.
See also: TrivSrfArea,

13.2.248 TrivTVsSame (triv_gen.c:1220)

```
CagdBType TrivTVsSame(const TrivTVStruct *Tv1,
                      const TrivTVStruct *Tv2,
                      CagdRType Eps)
```

Tv1, Tv2: The two trivariates to compare.

Eps: Tolerance of equality.

Returns: TRUE if trivariates are the same, FALSE otherwise.

Description: Compare the two trivariates for similarity.
See also: CagdSrfsSame, CagdCrvsSame, MvarMVsSame, VMdlVModelsSame.,

13.2.249 TrivTVsSubdivAtAllC0Discont (triv_sub.c:355)

```
TrivTVStruct *TrivTVsSubdivAtAllC0Discont(const TrivTVStruct *TVs)
```

TVs: To subdivide at all C^0 discontinuity locations.

Returns: Trivariates that result from the subdivision.

Description: Subdivides the given list of trivariates at all C^0 potential discontinuity locations.
See also: TrivTVSubdivAtParams, BspKnotAllC1Discont, , TrivTVSubdivAtAllC1Discont.,

13.2.250 TrivTVsSubdivAtAllC1Discont (triv_sub.c:413)

```
TrivTVStruct *TrivTVsSubdivAtAllC1Discont(const TrivTVStruct *TVs)
```

TVs: To subdivide at all C^1 discontinuity locations.

Returns: Trivariates that result from the subdivision.

Description: Subdivides the given list of trivariates at all C^1 potential discontinuity locations.
See also: TrivTVsSubdivAtParams, BspKnotAllC1Discont, , TrivTVsSubdivAtAllC0Discont, BspTVsSubdivAtAllDetectedLocations, , TrivTVSubdivAtAllC1Discont,

13.2.251 TrivTVsSubdivAtAllDetectedLocations (triv_sub.c:297)

```
TrivTVStruct *TrivTVsSubdivAtAllDetectedLocations(const TrivTVStruct *TVs,
                                                  TrivTVTestingFuncType
                                                      TVTestFunc)
```

TVs: Trivariates to subdivide at all detected locations by SrfTestFunc.

TVTestFunc: Trivariate testing function, like TrivTVKnotHasC0Discont.

Returns: Set of subdivided trivariates, or NULL if nothing was detected.

Description: Subdivides the given list of trivariates TVs at all locations TVTesTFunc detects. Examples for TVTesTFunc can be TrivTVKnotHasC0Discont or TrivTVKnotHasC1Discont.
See also: TrivTVKnotHasC0Discont, TrivTVKnotHasC1Discont, TrivTVSubdivAtParam, BspSrfsSubdivAtAllDetectedLocations, BspCrvsSubdivAtAllDetectedLocations,

13.2.252 TrivTriangleCopy (triv_gen.c:489)

```
TrivTriangleStruct *TrivTriangleCopy(const TrivTriangleStruct *Triangle)
```

Triangle: Triangle to duplicate.

Returns: Duplicated triangle.

Description: Allocates and duplicates all slots of a triangle structure.

13.2.253 TrivTriangleCopyList (triv_gen.c:517)

```
TrivTriangleStruct *TrivTriangleCopyList(const TrivTriangleStruct
                                                                *TriangleList)
```

TriangleList: List of triangle to duplicate.

Returns: Duplicated list of triangle.

Description: Duplicates a list of triangle structures.

13.2.254 TrivTriangleFree (triv_gen.c:546)

```
void TrivTriangleFree(TrivTriangleStruct *Triangle)
```

Triangle: Triangle to free.

Returns: void

Description: Deallocates and frees all slots of a triangle structure.

13.2.255 TrivTriangleFreeList (triv_gen.c:568)

```
void TrivTriangleFreeList(TrivTriangleStruct *TriangleList)
```

TriangleList: Triangle list to free.

Returns: void

Description: Deallocates and frees a list of triangle structures.

13.2.256 TrivTriangleNew (triv_gen.c:465)

allocation

```
TrivTriangleStruct *TrivTriangleNew(void)
```

Returns: An uninitialized triangle.

Description: Allocates the memory required for a new triangle.

13.2.257 TrivTrilinearSrf (trivruld.c:142)

```
TrivTVStruct *TrivTrilinearSrf(const CagdPtStruct *Pt000,
                               const CagdPtStruct *Pt001,
                               const CagdPtStruct *Pt010,
                               const CagdPtStruct *Pt011,
                               const CagdPtStruct *Pt100,
                               const CagdPtStruct *Pt101,
                               const CagdPtStruct *Pt110,
                               const CagdPtStruct *Pt111,
                               CagdPointType PType)
```

Pt000, Pt001, Pt010, Pt011, Pt100, Pt101, Pt110, Pt111: The eight corners of the trilinear.

PType: The type of the 8 given points.

Returns: Constructed trilinear.

Description: Constructs a trilinear volume between the given eight corner points.
See also: CagdBilinearSrf,

13.2.258 TrivTwoTVsMorphing (trivmrph.c:36)

morphing

```
TrivTVStruct *TrivTwoTVsMorphing(const TrivTVStruct *TV1,
                                 const TrivTVStruct *TV2,
                                 CagdRType Blend)
```

TV1, TV2: The two trivariates to blend.

Blend: A parameter between zero and one

Returns: TV2 * Blend + TV1 * (1 - Blend).

Description: Given two compatible trivariates (See function TrivMakeTVsCompatible), computes a convex blend between them according to Blend which must be between zero and one. Returned is the new blended trivariate.
See also: SymbTwoCrvsMorphing, SymbTwoCrvsMorphingCornerCut, , SymbTwoCrvsMorphingMultiRes, SymbTwoSrfsMorphing, TrivMakeTVsCompatible,

13.2.259 TrivUpdateBndrySrfinTV (triv_aux.c:997)

```
int TrivUpdateBndrySrfinTV(TrivTVStruct *TV,
                           const CagdSrfStruct *Srf,
                           TrivTVBndryType TVBndry)
```

TV: Trivariate to update one of its boundaries.

Srf: Surface information to update with the selected boundary of TV. face (and not UV reversed etc.).

TVBndry: The boundary in Srf to update with Crv.

Returns: TRUE, if successful, FALSE otherwise.

Description: Update one boundary of trivariate TV using the given surface Srf.
See also: CagdUpdateBndryCrvInSrf,

13.2.260 TrivVectCross3Vecs (geomat4d.c:180)

cross product

```
void TrivVectCross3Vecs(const TrivV4DType A,
                        const TrivV4DType B,
                        const TrivV4DType C,
                        TrivV4DType Res)
```

A, B, C: The three vectors to compute their cross product.

Res: Where the output goes into.

Returns: void

Description: Computes a vector in R^4 that is perpendicular to the given three vectors.

```
with(linalg);
readlib(C);



d := det( matrix( [ [I, J, K, L],
                  [A[0], A[1], A[2], A[3]],
                  [B[0], B[1], B[2], B[3]],
                  [C[0], C[1], C[2], C[3]] ] ) );
coeff( d, I );
coeff( d, J );
coeff( d, K );
coeff( d, L );
```

13.2.261 TrivZTwistExtrudeSrf (trivextr.c:142)

trivariate constructors

```
TrivTVStruct *TrivZTwistExtrudeSrf(const CagdSrfStruct *Srf,
                                   CagdBType Rational,
                                   CagdRType ZPitch)
```

Srf: To twist and extrude in the +Z direction.

Rational: TRUE to construct a rational (and precise) twist, FALSE to approximate using polynomials.

ZPitch: The +Z amount for full 360 degrees. If zero, the result will be a planar (degenerated) surface. A negative value will reverse the twist.

Returns: A twisted extrusion trivariate.

Description: Constructs a full circular twisted/rotated extrusion volume in the +Z direction for the given profile surface. Input surface can be either a Bspline or a Bezier surface.
See also: TrivExtrudeSrf,

Chapter 14

Triangular Library, trng_lib

14.1 General Information

This library provides a subset of functions to manipulate freeform triangular Bezier and Bspline patches. This library heavily depends on the cagd library. Functions are provided to create, copy, and destruct triangular patches, to extract isoparametric curves, to evaluate, refine and subdivide, to read and write triangular patches, to differentiate, and approximate using polygonal representations.

A triangular patch has one prescription of Length and one prescription of Order, the total Order and the length of an edge of the triangle. The control mesh mesh has Length * (Length + 1) / 2 control points,

```
typedef struct TrngTriangSrfStruct {
    struct TrngTriangSrfStruct *Pnext;
    struct IPAttributeStruct *Attr;
    TrngGeomType GType;
    CagdPointType PType;
    int Length;     /* Mesh size (length of edge of triangular mesh. */
    int Order;       /* Order of triangular surface (Bspline only). */
    CagdRType *Points[CAGD_MAX_PT_SIZE];     /* Pointer on each axis vector. */
    CagdRType *KnotVector;
} TrngTriangSrfStruct;
```

The interface of the library is defined in *include/trng_lib.h*.

This library has its own error handler, which by default prints an error message and exit the program called **TrngFatalError**.

All globals in this library have a prefix of **Trng**.

14.2 Library Functions

14.2.1 IritTrngDescribeError (trng_err.c:45)

error handling

```
const char *IritTrngDescribeError(IritTrngFatalErrorType ErrorNum)
```

ErrorNum: Type of the error that was raised.

Returns: A string describing the error type.

Description: Returns a string describing a the given error. Errors can be raised by any member of this trng library as well as other users. Raised error will cause an invocation of IritTrngFatalError function which decides how to handle this error. IritTrngFatalError can for example, invoke this routine with the error type, print the appropriate message and quit the program.

14.2.2 IritTrngFatalError (trng_ftl.c:56)

error handling

```
void IritTrngFatalError(IritTrngFatalErrorType ErrID)
```

ErrID: Error type that was raised.

Returns: void

Description: Trap Trng_lib errors right here. Provides a default error handler for the trng library. Gets an error description using IritTrngDescribeError, prints it and exit the program using exit.

14.2.3 IritTrngSetFatalErrorFunc (trng_ftl.c:28)

error handling

```
TrngSetErrorFuncType IritTrngSetFatalErrorFunc(TrngSetErrorFuncType ErrorFunc)
```

ErrorFunc: New error function to use.

Returns: Old error function reference.

Description: Sets the error function to be used by Trng_lib.

14.2.4 TrngBspTriSrfDerive (trng_der.c:165)

triangular surfaces

```
TrngTriangSrfStruct *TrngBspTriSrfDerive(const TrngTriangSrfStruct *TriSrf,
                                         TrngTriSrfDirType Dir)
```

TriSrf: Triangular Surface to differentiate.

Dir: Direction of differentiation. Either U or V or W.

Returns: Differentiated triangular surface in direction Dir. A Bspline triangular surface.

Description: Given a Bspline triangular surface, computes its partial derivative triangular surface in direction Dir.

14.2.5 TrngBspTriSrfHasOpenEC (trng_aux.c:196)

open end conditions

```
CagdBType TrngBspTriSrfHasOpenEC(const TrngTriangSrfStruct *TriSrf)
```

TriSrf: To check for open end conditions.

Returns: TRUE, if given Bspline triangular surface has open end conditions, FALSE otherwise.

Description: Returns TRUE iff the given triangular Bspline surface has open end coditions.

14.2.6 TrngBspTriSrfNew (trng_gen.c:75)

triangular surfaces

allocation

```
TrngTriangSrfStruct *TrngBspTriSrfNew(int Length,
                                      int Order,
                                      CagdPointType PType)
```

Length: Number of control points along the edge of the triangle.

Order: Order of triangular surface in all U,V,W directions.

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform triangular surface Bspline.

Description: Allocates the memory required for a new Bspline triangular surface.

14.2.7 TrngBspTriSrfOpenEnd (trng_aux.c:216)

open end conditions

```
TrngTriangSrfStruct *TrngBspTriSrfOpenEnd(const TrngTriangSrfStruct *TriSrf)
```

TriSrf: To check for open end conditions.

Returns: A triangular Bspline surface with open end cond.

Description: Returns TRUE iff the given triangular Bspline surface has open end coditions.

14.2.8 TrngBzrTriSrfDerive (trng_der.c:67)

triangular surfaces
derivatives

```
TrngTriangSrfStruct *TrngBzrTriSrfDerive(const TrngTriangSrfStruct *TriSrf,
                                         TrngTriSrfDirType Dir)
```

TriSrf: Triangular Surface to differentiate.

Dir: Direction of differentiation. Either U or V or W.

Returns: Differentiated triangular surface in direction Dir. A Bezier triangular surface.

Description: Given a Bezier triangular surface, computes its principal derivative in direction Dir.
See also: TrngBzrTriSrfDerive,

14.2.9 TrngBzrTriSrfDirecDerive (trng_der.c:115)

triangular surfaces
derivatives

```
TrngTriangSrfStruct *TrngBzrTriSrfDirecDerive(const TrngTriangSrfStruct *TriSrf,
                                              CagdVType DirecDeriv)
```

TriSrf: Triangular Surface to differentiate.

DirecDeriv: Derivative direction vector (coefficients must sum to zero!).

Returns: Differentiated triangular surface in direction Dir. A Bezier triangular surface.

Description: Given a Bezier triangular surface, computes its directional derivative in parametric direction DirectionalDeriv.
See also: TrngBzrTriSrfDerive,

14.2.10 TrngBzrTriSrfNew (trng_gen.c:108)

triangular surfaces
allocation

```
TrngTriangSrfStruct *TrngBzrTriSrfNew(int Length, CagdPointType PType)
```

Length: Number of control points along the edge of the triangle.

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform triangular surface Bezier.

Description: Allocates the memory required for a new Bezier triangular surface.

14.2.11 TrngCnvrtBzr2BspTriSrf (trng_gen.c:452)

conversion
triangular surface

```
TrngTriangSrfStruct *TrngCnvrtBzr2BspTriSrf(const TrngTriangSrfStruct *TriSrf)
```

TriSrf: A Bezier triangular surface to convert to a Bspline TriSrf.

Returns: A Bspline triangular surface representing the same geometry as the given Bezier TriSrf.

Description: Converts a Bezier triangular surface into a Bspline triangular surface by adding open end uniform knot vector to it.

14.2.12 TrngCnvrtGregory2BzrTriSrf (trng_grg.c:31)

conversion
triangular surface

```
TrngTriangSrfStruct *TrngCnvrtGregory2BzrTriSrf(TrngTriangSrfStruct *TriSrf)
```

TriSrf: A Gregory triangular surface to convert to a Bezier TriSrf.

Returns: A Bezier triangular surface representing the same geometry as the given Gregory TriSrf.

Description: Converts a Gregory triangular surface into a rational Bezier triangular surface.

14.2.13 TrngCoerceTriSrfTo (trngcoer.c:55)

coercion

```
TrngTriangSrfStruct *TrngCoerceTriSrfTo(const TrngTriangSrfStruct *CTriSrf,
                                        CagdPointType PType)
```

CTriSrf: To coerce to a new point type PType.

PType: New point type for TriSrf.

Returns: A new trngariate with PType as its point type.

Description: Coerces a triangular surface to point type PType.

14.2.14 TrngCoerceTriSrfsTo (trngcoer.c:27)

coercion

```
TrngTriangSrfStruct *TrngCoerceTriSrfsTo(const TrngTriangSrfStruct *TriSrf,
                                         CagdPointType PType)
```

TriSrf: To coerce to a new point type PType.

PType: New point type for TriSrf.

Returns: New triangular surfaces with PType as their point type.

Description: Coerces a list of triangular surfaces to point type PType.

14.2.15 TrngCrvFromTriSrf (trng_iso.c:306)

isoparametric curves

curve from surface

```
CagdCrvStruct *TrngCrvFromTriSrf(const TrngTriangSrfStruct *TriSrf,
                                 CagdRType t,
                                 TrngTriSrfDirType Dir)
```

TriSrf: To extract an isoparametric curve from.

t: Parameter value of extracted isoparametric curve.

Dir: Direction of extracted isocurve. Either U or V or W.

Returns: An isoparametric curve of TriSrf. This curve inherit the order and continuity of TriSrf in direction Dir.

Description: Extracts an isoparametric curve from the triangular surface TriSrf in direction Dir at the parameter value of t.

See also: BzrSrfCrvFromSrf, BspSrfCrvFromSrf, CagdCrvFromMesh, BzrSrfCrvFromMesh, , BspSrfCrvFromMesh, CagdCrvFromSrf, TrngTriBzrSrf2Curves,

14.2.16 TrngDbg (trng_dbg.c:29)

debugging

```
void TrngDbg(void *Obj)
```

Obj: A triangular surface - to be printed to stderr.

Returns: void

Description: Prints triangular surface to stderr. Should be linked to programs for debugging purposes, so triangular surfaces may be inspected from a debugger.

See also: TrngDbg1,

14.2.17 TrngDbg1 (trng_dbg.c:73)

debugging

```
void TrngDbg1(void *Obj)
```

Obj: A triangular surface - to be printed to stderr.

Returns: void

Description: Prints triangular surface to stderr. Should be linked to programs for debugging purposes, so triangular surfaces may be inspected from a debugger.

See also: TrngDbg,

14.2.18 TrngGregory2Bezier4 (trng_grg.c:71)

conversion

triangular surface

```
void TrngGregory2Bezier4(CagdRType **Qt, CagdRType **Pt)
```

Qt: The resulting Bezier control points

Pt: The Gregory control points

Returns: void

Description: Converts a Gregory triangular surface into a Bezier triangular surface

14.2.19 TrngGregory2Bezier5 (trng_grg.c:234)

conversion

triangular surface

```
void TrngGregory2Bezier5(CagdRType **Qt, CagdRType **Pt)
```

Qt: The resulting Bezier control points

Pt: The Gregory control points

Returns: void

Description: Converts a Gregory triangular surface into a Bezier triangular surface

14.2.20 TrngGregory2Bezier6 (trng_grg.c:422)

conversion

triangular surface

```
void TrngGregory2Bezier6(CagdRType **Qt, CagdRType **Pt)
```

Qt: The resulting Bezier control points

Pt: The Gregory control points

Returns: void

Description: Converts a Gregory triangular surface into a Bezier triangular surface

14.2.21 TrngGrgTriSrfNew (trng_gen.c:135)

triangular surfaces

allocation

```
TrngTriangSrfStruct *TrngGrgTriSrfNew(int Length, CagdPointType PType)
```

Length: Number of control points along the edge of the triangle.

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform triangular surface Gregory.

Description: Allocates the memory required for a new Gregory triangular surface.

14.2.22 TrngParamInDomain (trng_aux.c:82)

triangular surfaces

```
CagdBType TrngParamInDomain(TrngTriangSrfStruct *TriSrf,
                            CagdRType t,
                            TrngTriSrfDirType Dir)
```

TriSrf: To make sure t is in its Dir domain.

t: Parameter value to verify.

Dir: Direction. Either U or V or W.

Returns: TRUE if in domain, FALSE otherwise.

Description: Given a triangular surface and a domain - validate it.

14.2.23 TrngParamsInDomain (trng_aux.c:118)

triangular surfaces

```
CagdBType TrngParamsInDomain(const TrngTriangSrfStruct *TriSrf,
                             CagdRType u,
                             CagdRType v,
                             CagdRType w)
```

TriSrf: To make sure (u, v, w) is in its domain.

u, v, w: To verify if it is in TriSrf's parametric domain.

Returns: TRUE if in domain, FALSE otherwise.

Description: Given a triangular surface and a domain - validate it.

14.2.24 TrngSrfSubdivAtParam (trng_sub.c:33)

subdivision

```
TrngTriangSrfStruct *TrngSrfSubdivAtParam(TrngTriangSrfStruct *TrngSrf,
                                          CagdRType t,
                                          CagdSrfDirType Dir)
```

TrngSrf: To subdivide at the prescibed parameter value t.

t: The parameter to subdivide the curve Crv at.

Dir: Direction of subdivision. Either U or V.

Returns: The subdivided surfaces. Usually two, but can have only one, if other is totally trimmed away.

Description: Given a triangulare surface - subdivides it into two three sub-surfaces at given parametric values u, v, w. Returns pointer to a list of two trngmed surfaces, at most. It can very well may happen that the subdivided surface is completely trimmed out and hence nothing is returned for it.

14.2.25 TrngTriBzrSrf2Curves (trng_iso.c:175)

curves

isoparametric curves

```
CagdCrvStruct *TrngTriBzrSrf2Curves(const TrngTriangSrfStruct *TriSrf,
                                    int NumOfIsocurves[3],
                                    IrtRType Val)
```

TriSrf: To extract isoparametric curves from.

NumOfIsocurves: In each (U or V or W) direction.

Val: If only one isocurve to extract - do so at value Val.

Returns: List of extracted isoparametric curves. These curves inherit the order and continuity of the original Srf. NULL is returned in case of an error.

Description: Routine to extract from a triangular surface NumOfIsoline isocurve list in each param. direction. Iso parametric curves are sampled equally spaced in parametric space. If, however, oout of the three NumOfIsocurves values, two are zero and one NumOfIsocurves values equals to one, extarct one isocurve at that direction, at value Val. NULL is returned in case of an error, otherwise list of CagdCrvStruct.
Consider isoparametric curve of Bezier triangular surface at fixed u = u0:

$$\sum_{i+j+k=n} \frac{n!}{i!\,j!\,(n-i-j)!}\, u0^{i}\, v^{j}\, (1-u0-v)^{n-i-j}\, b_{ijk} =$$

$$\sum_{i=0}^{n} \frac{n!}{i!\,(n-i)!}\, u0^{i} \sum_{j+k=n-i} \frac{(n-i)!}{j!\,(n-i-j)!}\, v^{j}\, (1-u0-v)^{n-i-j}\, b_{ijk} =$$

$$\sum_{i=0}^{n} \frac{n!}{i!(n-i)!} u0^{i} (1-u0)^{n-i} \sum_{j=0}^{n-i} \frac{(n-i)!}{j!\,(n-i-j)!} \left(\frac{v}{1-u0}\right)^{j} \left(1 - \frac{v}{1-u0}\right)^{n-i-j} b_{ijk}$$

Hence, the isoparametric curve of u = u0 is a weighted sum of a sequence of Bezier curves of degree 1 to n-1, each defined over a row of the triangular mesh, over the domain of v = [0, 1-u0].

See also: TrngTriSrf2Curves, TrngCrvFromTriSrf,

14.2.26 TrngTriSrf2CtrlMesh (trngmesh.c:25)

bbox

bounding box

```
CagdPolylineStruct *TrngTriSrf2CtrlMesh(const TrngTriangSrfStruct *TriSrf)
```

TriSrf: To compute a polyline representation for its control mesh.

Returns: A polyline representing TriSrf's control mesh.

Description: Computes a polyline representation to the control mesh of the triangular surface in E3.

14.2.27 TrngTriSrf2Curves (trng_iso.c:354)

curves

isoparametric curves

```
CagdCrvStruct *TrngTriSrf2Curves(const TrngTriangSrfStruct *TriSrf,
                                 int NumOfIsocurves[3])
```

TriSrf: To extract isoparametric curves from.

NumOfIsocurves: In each (U or V or W) direction.

Returns: List of extracted isoparametric curves. These curves inherit the order and continuity of the original Srf. NULL is returned in case of an error.

Description: Routine to extract from a triangular surface NumOfIsoline isocurve list in each param. direction. Iso parametric curves are sampled equally spaced in parametric space. NULL is returned in case of an error, otherwise list of CagdCrvStruct.

See also: TrngTriBzrSrf2Curves, TrngCrvFromTriSrf,

14.2.28 TrngTriSrf2Polygons (trng2ply.c:37)

polygonization

surface approximation

```
IPPolygonStruct *TrngTriSrf2Polygons(const TrngTriangSrfStruct *TriSrf,
                                     const CagdSrf2PlsInfoStrct *TessInfo)
```

TriSrf: To approximate into triangles.

TessInfo: Parameters to control the tessellation into polygons.

Returns: A list of polygons with optional normal and/or UV parametric information. NULL is returned in case of an error.

Description: Routine to convert a single triangular surface to a set of triangles approximating it. FineNess is a fineness control on result and the larger it is more triangles may result. A value of 10 is a good starting value. NULL is returned in case of an error, otherwise list of IPPolygonStruct.

14.2.29 TrngTriSrf2Polylines (trng_iso.c:44)

isoparametric curves

```
CagdPolylineStruct *TrngTriSrf2Polylines(const TrngTriangSrfStruct *TriSrf,
                                         int NumOfIsocurves[3],
                                         CagdRType TolSamples,
                                         SymbCrvApproxMethodType Method)
```

TriSrf: To extract isoparametric curves from.

NumOfIsocurves: In each (U or V or W) direction.

TolSamples: Tolerance of approximation error (Method = 2) or Number of samples to compute on polyline (Method = 0, 1).

Method: 0 - TolSamples are set uniformly in parametric space, 1 - TolSamples are set optimally, considering the isocurve's curvature. 2 - TolSamples sets the maximum error allowed between the piecewise linear approximation and original curve.

Returns: List of polylines representing a piecewise linear approximation of the extracted isoparamteric curves or NULL is case of an error.

Description: Routine to convert a single triangular surface to NumOfIsolines polylines in each parametric direction with SamplesPerCurve in each isoparametric curve in E3. Polyline are always E3 of CagdPolylineStruct type. Iso parametric curves are sampled equally spaced in parametric space. NULL is returned in case of an error, otherwise list of CagdPolylineStruct. Attempt is made to extract isolines along C1 discontinuities first.

14.2.30 TrngTriSrfBBox (trng_aux.c:144)

bbox

bounding box

```
CagdBBoxStruct *TrngTriSrfBBox(const TrngTriangSrfStruct *TriSrf,
                               CagdBBoxStruct *BBox)
```

TriSrf: To compute a bounding box for.

BBox: Where bounding information is to be saved.

Returns: BBox.

Description: Computes a bounding box for a triangular surfaces.

14.2.31 TrngTriSrfCopy (trng_gen.c:160)

triangular surfaces

```
TrngTriangSrfStruct *TrngTriSrfCopy(const TrngTriangSrfStruct *TriSrf)
```

TriSrf: triangular surface to duplicate

Returns: Duplicated triangular surface.

Description: Allocates and duplicates all slots of a triangular surface structure.

14.2.32 TrngTriSrfCopyList (trng_gen.c:208)

triangular surfaces

```
TrngTriangSrfStruct *TrngTriSrfCopyList(const TrngTriangSrfStruct *TriSrfList)
```

TriSrfList: List of triangular surfaces to duplicate.

Returns: Duplicated list of triangular surfaces.

Description: Duplicates a list of triangular surface structures.

14.2.33 TrngTriSrfDerive (trng_der.c:31)

triangular surfaces.

```
TrngTriangSrfStruct *TrngTriSrfDerive(const TrngTriangSrfStruct *TriSrf,
                                      TrngTriSrfDirType Dir)
```

TriSrf: Triangular Surface to differentiate.

Dir: Direction of differentiation. Either U or V or W.

Returns: Differentiated triangular surface in direction Dir.

Description: Given a triangular surface, computes its partial derivative triangular surface in direction Dir.

14.2.34 TrngTriSrfDomain (trng_aux.c:34)

triangular surfaces

```
void TrngTriSrfDomain(const TrngTriangSrfStruct *TriSrf,
                      CagdRType *UMin,
                      CagdRType *UMax,
                      CagdRType *VMin,
                      CagdRType *VMax,
                      CagdRType *WMin,
                      CagdRType *WMax)
```

TriSrf: Triangular surface function to consider.

UMin, UMax: U Domain of TriSrf will be placed herein.

VMin, VMax: V Domain of TriSrf will be placed herein.

WMin, WMax: W Domain of TriSrf will be placed herein.

Returns: void

Description: Given a triangular surface, returns its parametric domain.

14.2.35 TrngTriSrfEval2ToData (trngeval.c:161)

evaluation

triangular surfaces

```
CagdRType *TrngTriSrfEval2ToData(const TrngTriangSrfStruct *TriSrf,
                                 CagdRType u,
                                 CagdRType v,
                                 CagdRType *Pt)
```

TriSrf: To evaluate at given (u, v) parametric location.

u, v: Parametric location to evaluate TriSrf at.

Pt: A vector holding all the coefficients of all components of the triangualr surface's point type. If for example point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Returns: A vector holding all the coefficients of all components of the triangualr surface's point type. If for example point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Description: Evaluates the given triangular surface at a given point. Same as function TrngTriSrfEval with w computed using 'w = 1 - u - v' for Bezier triangular surfaces.

14.2.36 TrngTriSrfEvalToData (trngeval.c:91)

evaluation

triangular surfaces

```
CagdRType *TrngTriSrfEvalToData(const TrngTriangSrfStruct *TriSrf,
                                CagdRType u,
                                CagdRType v,
                                CagdRType w,
                                CagdRType *Pt)
```

TriSrf: To evaluate at given (u, v, w) parametric location.

u, v, w: Parametric location to evaluate TriSrf at.

Pt: A vector holding all the coefficients of all components of the triangualr surface's point type. If for example point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Returns: A vector holding all the coefficients of all components of the triangualr surface's point type. If for example point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Description: Evaluates the given triangular surface at a given point.

14.2.37 TrngTriSrfFree (trng_gen.c:237)

triangular surfaces

```
void TrngTriSrfFree(TrngTriangSrfStruct *TriSrf)
```

TriSrf: triangular surface to free.

Returns: void

Description: Deallocates and frees all slots of a triangular surface structure.

14.2.38 TrngTriSrfFreeList (trng_gen.c:269)

triangular surfaces

```
void TrngTriSrfFreeList(TrngTriangSrfStruct *TriSrfList)
```

TriSrfList: triangular surface list to free.

Returns: void

Description: Deallocates and frees a list of triangular surface structures.

14.2.39 TrngTriSrfListBBox (trng_aux.c:168)

bbox

bounding box

```
CagdBBoxStruct * TrngTriSrfListBBox(const TrngTriangSrfStruct *TriSrfs,
                                    CagdBBoxStruct *BBox)
```

TriSrfs: To compute a bounding box for.

BBox: Where bounding information is to be saved.

Returns: BBox.

Description: Computes a bounding box for a list of triangular surfaces.

14.2.40 TrngTriSrfListMatTransform (trng_gen.c:382)

scaling

rotation

translation

transforms

```
TrngTriangSrfStruct *TrngTriSrfListMatTransform(const TrngTriangSrfStruct
                                                                  *TriSrfs,
                                                CagdMType Mat)
```

TriSrfs: To be transformed.

Mat: Defining the transformation.

Returns: Transformed triangular surface.

Description: Transforms the given list of triangular surfaces as specified by homogeneous matrix Mat.
See also: TrngTriSrfMatTransform, CagdSrfListMatTransform,

14.2.41 TrngTriSrfMatTransform (trng_gen.c:333)

triangular surfaces.

```
TrngTriangSrfStruct *TrngTriSrfMatTransform(const TrngTriangSrfStruct *TriSrf,
                                            CagdMType Mat)
```

TriSrf: Multi-variate to transform.

Mat: Homogeneous transformation to apply to TV.

Returns: Transformed triangular surface.

Description: Transforms the given triangular surface as specified by homogeneous matrix Mat.

14.2.42 TrngTriSrfMatTransform2 (trng_gen.c:413)

triangular surfaces

```
void TrngTriSrfMatTransform2(TrngTriangSrfStruct *TriSrf, CagdMType Mat)
```

TriSrf: Triangular surface to transform.

Mat: Homogeneous transformation to apply to TriSrf.

Returns: void

Description: Transforms, in place, the given triangular surface as specified by homogeneous matrix Mat.
See also: TrngTriSrfMatTransform, TrngTriSrfListMatTransform,

14.2.43 TrngTriSrfNew (trng_gen.c:31)

triangular surfaces

allocation

```
TrngTriangSrfStruct *TrngTriSrfNew(TrngGeomType GType,
                                   CagdPointType PType,
                                   int Length)
```

GType: Type of geometry the curve should be - Bspline, Bezier etc.

PType: Type of control points (E2, P3, etc.).

Length: Number of control points along the edge of the triangle.

Returns: An uninitialized freeform triangular surface.

Description: Allocates the memory required for a new triangular surface.

14.2.44 TrngTriSrfNrmlToData (trngeval.c:197)

evaluation

triangular surfaces

```
CagdVecStruct *TrngTriSrfNrmlToData(const TrngTriangSrfStruct *TriSrf,
                                    CagdRType u,
                                    CagdRType v,
                                    CagdVecStruct *Normal)
```

TriSrf: To evaluate at given (u, v, w) parametric location.

u, v: Parametric location to evaluate normal of TriSrf at.

Normal: A pointer to a vector holding the unit normal information.

Returns: A pointer to a vector holding the unit normal information.

Description: Evaluates the normal of the given triangular surface at a given point.
See also: CagdSrfNormal, BzrSrfNormal, BspSrfNormal, SymbSrfNormalSrf,

14.2.45 TrngTriSrfTransform (trng_gen.c:298)

triangular surfaces

```
void TrngTriSrfTransform(TrngTriangSrfStruct *TriSrf,
                         CagdRType *Translate,
                         CagdRType Scale)
```

TriSrf: Triangular surface to transform.

Translate: Translation factor.

Scale: Scaling factor.

Returns: void

Description: Linearly transforms, in place, given TriSrf as specified by Translate and Scale.

14.2.46 TrngTriSrfsSame (trng_gen.c:489)

```
CagdBType TrngTriSrfsSame(const TrngTriangSrfStruct *Srf1,
                          const TrngTriangSrfStruct *Srf2,
                          CagdRType Eps)
```

Srf1, Srf2: The two surfaces to compare.

Eps: Tolerance of equality.

Returns: TRUE if surfaces are the same, FALSE otehrwise.

Description: Compare the two surfaces for similarity.
See also: CagdSrfsSame,

Chapter 15

User Library, user_lib

15.1 General Information

This library includes user interface related geometrical functions such as ray surface intersection (for mouse click/select operations), etc.

The interface of the library is defined in *include/user_lib.h*.

15.2 Library Functions

15.2.1 IntrSrfHierarchyFreePreprocess (srfpgeom.c:249)

ray surface intersection

```
void IntrSrfHierarchyFreePreprocess(VoidPtr Handle)
```

Handle: As returned by IntrSrfHierarchyPreprocessSrf to release.

Returns: void

Description: Releases the pre processed data structed created by the function IntrSrfHierarchyPreprocessSrf.
See also: IntrSrfHierarchyPreprocessSrf, IntrSrfHierarchyTestRay, , IntrSrfHierarchyTestPt,

15.2.2 IntrSrfHierarchyPreprocessSrf (srfpgeom.c:89)

ray surface intersection

```
VoidPtr IntrSrfHierarchyPreprocessSrf(const CagdSrfStruct *Srf,
                                      IrtRType FineNess)
```

Srf: To preprocess.

FineNess: Control on accuracy, the higher the finer. The surface will be subdivided into approximately FineNess regions in each of the two parametric directions.

Returns: A handle on the preprocessed data, NULL if error.

Description: Preprocess a surface for fast computation of ray-surface intersection. Returns NULL if fails, otherwise a pointer to preprocessed data structure. The preprocessed data is in fact a hierarchy of bounding boxes extracted while the surface is being polygonized.
See also: IntrSrfHierarchyFreePreprocess, IntrSrfHierarchyTestRay, , IntrSrfHierarchyTestPt,

15.2.3 IntrSrfHierarchyTestPt (srfpgeom.c:495)

pt surface min/max distance

```
CagdBType IntrSrfHierarchyTestPt(VoidPtr Handle,
                                 CagdPType Pt,
                                 CagdBType Nearest,
                                 CagdUVType InterUV)
```

Handle: As returned by IntrSrfHierarchyPreprocessSrf.

Pt: To look for nearest/farthest location on the surface.

Nearest: TRUE for nearest, FALSE for farthest.

InterUV: The UV surface coordinates of the nearest/farthest surface location is saved here.

Returns: TRUE if found min/max distance, FALSE otherwise.

Description: Computes the nearest/farthest location on the surface from point Pt.
See also: IntrSrfHierarchyPreprocessSrf, IntrSrfHierarchyFreePreprocess, , IntrSrfHierarchyTestRay,

15.2.4 IntrSrfHierarchyTestRay (srfpgeom.c:306)

ray surface intersection

```
CagdBType IntrSrfHierarchyTestRay(VoidPtr Handle,
                                  CagdPType RayOrigin,
                                  CagdVType RayDir,
                                  CagdUVType InterUV)
```

Handle: As returned by IntrSrfHierarchyPreprocessSrf.

RayOrigin: Starting point of ray.

RayDir: Direction of ray.

InterUV: The UV surface coordinates of the first ray surface intersection location is saved here.

Returns: TRUE if found intersection, FALSE otherwise.

Description: Computes the first intersection of a given ray with the given surface, if any. If TRUE is returned, the InterUV is updated to the interesection.
See also: IntrSrfHierarchyFreePreprocess, IntrSrfHierarchyPreprocessSrf, , IntrSrfHierarchyTestPt,

15.2.5 IritUserDescribeError (user_err.c:76)

error handling

```
const char *IritUserDescribeError(IritUserFatalErrorType ErrorNum)
```

ErrorNum: Type of the error that was raised.

Returns: A string describing the error type.

Description: Returns a string describing a the given error. Errors can be raised by any member of this user library as well as other users. Raised error will cause an invocation of IritUserFatalError function which decides how to handle this error. IritUserFatalError can for example, invoke this routine with the error type, print the appropriate message and quit the program.

15.2.6 IritUserFatalError (user_ftl.c:57)

error handling

```
void IritUserFatalError(IritUserFatalErrorType ErrID)
```

ErrID: Error type that was raised.

Returns: void

Description: Trap User_lib errors right here. Provides a default error handler for the user library. Gets an error description using IritUserDescribeError, prints it and exit the program using exit.

15.2.7 IritUserSetFatalErrorFunc (user_ftl.c:29)

error handling

```
UserSetErrorFuncType IritUserSetFatalErrorFunc(UserSetErrorFuncType ErrorFunc)
```

ErrorFunc: New error function to use.

Returns: Old error function reference.

Description: Sets the error function to be used by User_lib.

15.2.8 IrtImgScaleImage (scalimag.c:49)

```
IrtImgPixelStruct *IrtImgScaleImage(IrtImgPixelStruct *InImage,
                                    int InMaxX,
                                    int InMaxY,
                                    int InAlpha,
                                    int OutMaxX,
                                    int OutMaxY,
                                    int Order)
```

InImage: A vector of RGBRGB... of size (MaxX+1) * (MaxY+1) * 3 or NULL if failed. If however, Alpha is available we have RGBARGBA and InImage is actually IrtImgRGBAPxlStruct.

InMaxX: Maximum X of input image.

InMaxY: Maximum Y of input image.

InAlpha: If TRUE, we have alpha as well and InImage is actually IrtImgRGBAPxlStruct.

OutMaxX: Maximum X of output image.

OutMaxY: Maximum Y of output image.

Order: Of the spline filer. 2 for a bilinear and the higher the Order is the smoother the result will be.

Returns: The scaled image as vector of RGBRGB (or RGBARGBA).

Description: Scale an image by mapping it to a bivariate spline and resampling.
See also: IrtImgReadImage,

15.2.9 MatGenMatScaleCenter (micro5tile.c:2225)

```
void MatGenMatScaleCenter(IrtRType Scl,
                          const IrtPtType Center,
                          IrtHmgnMatType Mat)
```

Scl: The scaling factor.

Center: The location around which to scale.

Mat: The created matrix.

Returns: void

Description: Creates a uniform scaling matrix by factor Scl around location Center.
See also: MatGenMatTrans, MatGenMatUnifromScale,

15.2.10 User2PolyMeshRoundEdge (plyround.c:391)

```
int User2PolyMeshRoundEdge(IPPolygonStruct *Pl1,
                           IPPolygonStruct *Pl2,
                           const IPPolygonStruct *Edge12,
                           IrtRType RoundRadius,
                           IrtRType RoundShape)
```

Pl1, Pl2: The two input meshes sharing edge Edge12 to round along. in place.

Edge12: The common edge(s) to round along. Can be a list of edges to round around all of them.

RoundRadius: The desired radius of the approximated blend.

RoundShape: Bias to affect the rounding size. 1.0 to have no affect, and values larger (smaller) than 1.0 to enlarge (shrink) the rounding size.

Returns: TRUE if successful, FALSE otherwise.

Description: Given two meshes, Pl1 and Pl2, sharing common boundary edge(s), Edge12, updated Pl1 and Pl2 in place and round them along Edge12.
See also: GMPolyMeshSmoothing,

15.2.11 User3DDither2Images (dtr3d2im.c:1463)

```
IPObjectStruct *User3DDither2Images(const char *Image1Name,
                                    const char *Image2Name,
                                    int DitherSize,
                                    int MatchWidth,
                                    int Negate,
                                    int AugmentContrast,
                                    User3DSpreadType SpreadMethod,
                                    IrtRType SphereRad,
                                    IrtRType *AccumPenalty)
```

Image1Name: Name of 1st image to load.

Image2Name: Name of 2nd image to load.

DitherSize: 1, 2, 3 or 4 for (1x1), (2x2), (3x3) or (4x4) dithering.

MatchWidth: Width to allow matching in a row: between pos[i] to pos[i +/- k], k < MatchWidth.

Negate: TRUE to negate the images.

AugmentContrast: Number of iterations to add micro-pixels, to augment the contrast, behind existing pixels. Zero to disable.

SpreadMethod: If allowed (MatchWidth >= RowSize), selects initial random spread to use.

SphereRad: Radius of construct spherical blob, zero to return points.

AccumPenalty: Returns the accumulated error in the dithering-matching, where zero designates no error.

Returns: Micro blobs if SphereRad > 0, center points, if = 0.

Description: Build a 3D models consisting of spherical blobs that looks like the 1st image (gray level) from the XZ plane and like the 2nd image from the YZ plane. The entire constructed geometry is confined to a cube world space of [max(ImageWidth, ImageHeight)]^3.
See also: User3DDither3Images,

15.2.12 User3DDither3Images (dtr3d3im.c:365)

```
IPObjectStruct *User3DDither3Images(const char *Image1Name,
                                    const char *Image2Name,
                                    const char *Image3Name,
                                    int DitherSize,
                                    int MatchWidth,
                                    int Negate,
                                    int AugmentContrast,
                                    User3DSpreadType SpreadMethod,
                                    IrtRType SphereRad,
                                    IrtRType *AccumPenalty)
```

Image1Name: Name of 1st image to load.

Image2Name: Name of 2nd image to load.

Image3Name: Name of 3rd image to load. Optional and can be NULL or a zero length string.

DitherSize: Dithering matrix size to use: 2, 3, or 4.

MatchWidth: Width to allow matching in a row: between pos[i] to pos[i +/- k], k < MatchWidth.

Negate: TRUE to negate the images.

AugmentContrast: Redundancy level for the micro-pixels, to augment the contrast, behind existing pixels. Zero to disable.

SpreadMethod: Selects initial spread to use: Random, Diagonal, etc.

SphereRad: Radius of construct spherical blob, zero to return points.

AccumPenalty: Returns the accumulated error in the dithering-matching, where zero designates no error. In level of achieved covering.

Returns: Center points, in [ImageWidth]^3 space.

Description: Build a 3D models consisting of points/spherical blobs that looks like the 1st image (gray level) from the XZ plane, like the 2nd image from the YZ plane and, optionally, like the 3rd image from the XY plane. The entire constructed geometry is confined to a cube world space of [ImageWidth]^3 (ImageWidth = ImageHeight).
See also: User3DDither3Images2, User3DDither2Images,

15.2.13 User3DDither3Images2 (dtr3d3im.c:226)

```
IPObjectStruct *User3DDither3Images2(const char *Image1Name,
                                     const char *Image2Name,
                                     const char *Image3Name,
                                     int DitherSize,
                                     int MatchWidth,
                                     int Negate,
                                     int AugmentContrast,
                                     User3DSpreadType SpreadMethod,
                                     IrtRType SphereRad,
                                     IrtRType *AccumPenalty)
```

Image1Name: Name of 1st image to load.

Image2Name: Name of 2nd image to load.

Image3Name: Name of 3rd image to load.

DitherSize: 2 or 3 for (2x2x2) or (3x3x3) dithering matrices.

MatchWidth: Width to allow matching in a row: between pos[i] to pos[i +/- k], k < MatchWidth.

Negate: TRUE to negate the images.

AugmentContrast: Number of iterations to add micro-pixels, to augment the contrast, behind existing pixels. Zero to disable.

SpreadMethod: If allowed (MatchWidth >= RowSize), selects initial random spread to use.

SphereRad: Radius of construct spherical blob, zero to return points.

AccumPenalty: Returns the accumulated error in the dithering-matching, where zero designates no error.

Returns: Center points.

Description: Build a 3D models consisting of pixels/spherical blobs that looks like the 1st image (gray level) from the XZ plane, like the 2nd image from the YZ plane and like the 3rd image from the XY plane. The entire constructed geometry is confined to a cube world space of [ImageWidth]^3 (ImageWidth = ImageHeight).
See also: User3DDither3Images,

15.2.14 User3DDitherSetXYTranslations (dtr3d2im.c:822)

```
IPVertexStruct *User3DDitherSetXYTranslations(IPVertexStruct *Vrtcs)
```

Vrtcs: Pixels to shift a tad, in place.

Returns: Translated pixels, in place.

Description: Adds small sub pixels' shifts to the micro pixels in the XY direction.

15.2.15 User3DMicroBlobsCreateRandomMatrix (imgshd3d.c:1022)

```
int **User3DMicroBlobsCreateRandomMatrix(int Size,
                                         User3DSpreadType BlobSpreadMethod)
```

Size: Of random vector to create to spread the blobs.

BlobSpreadMethod: Blob spreading method desired.

Returns: Created matrix as vector of vectors.

Description: Creates a matrix of size (Size x Size) of numbers between 0 and Size-1, randomly distributed so that no row or columns has the same number twice.
See also: User3DMicroBlobsCreateRandomVector,

15.2.16 User3DMicroBlobsCreateRandomVector (imgshd3d.c:935)

```
int *User3DMicroBlobsCreateRandomVector(int Size,
                                        User3DSpreadType BlobSpreadMethod,
                                        IrtBType FirstVec)
```

Size: Of random vector to create to spread the blobs.

BlobSpreadMethod: Blob spreading method desired.

FirstVec: TRUE for first vector, FALSE for second vector.

Returns: Created vector.

Description: Creates a permutation vector of numbers between 0 and Size-1, with a desired distributed, in a vector of size Size.
See also: User3DMicroBlobsCreateRandomMatrix,

15.2.17 User3DMicroBlobsFrom3Images (imgshd3d.c:1176)

```
IPObjectStruct *User3DMicroBlobsFrom3Images(const char *Image1Name,
                                            const char *Image2Name,
                                            const char *Image3Name,
                                            User3DSpreadType BlobSpreadMethod,
                                            IrtRType Intensity,
                                            const IrtVecType MicroBlobSpacing,
                                            const IrtVecType RandomFactors,
                                            int Resolution,
                                            int Negative,
                                            IrtRType CubeSize,
                                            int MergePts)
```

Image1Name: Name of 1st image to load.

Image2Name: Name of 2nd image to load.

Image3Name: Name of 3rd image to load. Optional (Can be NULL).

BlobSpreadMethod: Method of spreading the blobs.

Intensity: A scale affect on the blobs' scale.

MicroBlobSpacing: Spacing to use in the micro blob, in world space coordinates.

RandomFactors: Maximal allowed randomization in XYZ, in world space coordinates.

Resolution: Resolution of created objects (Resolution^2 ellipsoidal blobs are created).

Negative: Default (FALSE) is white blobs over dark background. If TRUE, assume dark blobs over white background.

CubeSize: Size of output.

MergePts: TRUE to merge all points to one list, FALSE each of the Resolution^2 blobs will hold its own point list.

Returns: Resolution^2 pointlists of micro blobs of Resolution^2 spherical blobs.

Description: Creates micro blobs for Resolution^2 ellipsoidal blobs that looks like the 1st image (gray level) from the XZ plane and like the 2nd image from the YZ plane. The entire constructed goemetry is confined to a world cube space of [0, CubeSize]^3.
See also: User3DMicroBlobsTiling,

15.2.18 User3DMicroBlobsTiling (imgshd3d.c:1432)

```
IPPolygonStruct *User3DMicroBlobsTiling(IrtRType XZIntensity,
                                        IrtRType YZIntensity,
                                        IrtRType XYIntensity,
                                        const IrtVecType MicroBlobSpacing,
                                        const IrtVecType RandomFactors)
```

XZIntensity: Intensity (0 to 1) of blob when viewed from XZ dir. Can be invalidated and ignored if negative.

YZIntensity: Intensity (0 to 1) of blob when viewed from YZ dir. Can be invalidated and ignored if negative.

XYIntensity: Intensity (0 to 1) of blob when viewed from XY dir. Can be invalidated and ignored if negative.

MicroBlobSpacing: XYZ spacing between micro blobs.

RandomFactors: Maximal randomization factors to use on micro blobs.

Returns: A pointlist of centers of the tiling micro blobs.

Description: Tile a given blob in the shape of an ellipsoid bounded by [-1, 1]^3, by micro blobs with XYZ spacing as prescribed by ??Intensity/MicroBlobSpacing.
See also: User3DMicroBlobsFrom2Images, User3DMicroBlobsTiling2,

15.2.19 User3DMicroBlobsTiling2 (imgshd3d.c:1559)

```
IPPolygonStruct *User3DMicroBlobsTiling2(IrtRType XZIntensity,
                                         IrtRType YZIntensity,
                                         IrtRType XYIntensity,
                                         const IrtVecType MicroBlobSpacing,
                                         const IrtVecType RandomFactors)
```

XZIntensity: Intensity (0 to 1) of blob when viewed from XZ dir. Can be invalidated and ignored if negative.

YZIntensity: Intensity (0 to 1) of blob when viewed from YZ dir. Can be invalidated and ignored if negative.

XYIntensity: Intensity (0 to 1) of blob when viewed from XY dir. Can be invalidated and ignored if negative.

MicroBlobSpacing: XYZ spacing between micro blobs.

RandomFactors: Maximal randomization factors to use on micro blobs.

Returns: A pointlist of centers of the tiling micro blobs.

Description: Tile a given blob in the shape of a randomized cubed in [-1, 1]^3, by micro blobs with XYZ spacing as prescribed by ??Intensity/MicroBlobSpacing.
See also: User3DMicroBlobsFrom2Images, User3DMicroBlobsTiling,

15.2.20 UserAMFiber3AxisFreeFragments (am3axis_frag_split.c:85)

```
void UserAMFiber3AxisFreeFragments(UserAMFiber3AxisFragStruct *Fragments)
```

Fragments: Struct to free.

Returns: void

Description: Frees a fragments struct.
See also:

15.2.21 UserAMFiber3AxisFreeTValList (am3axis_frag_split.c:323)

```
void UserAMFiber3AxisFreeTValList(UserAMFiber3AxisTValListStruct *List)
```

List: The list to free.

Returns: void.

Description: Free a list of t values.
See also:

15.2.22 UserAMFiber3AxisGetBBoxMaxTVals (am3axis_frag_split.c:287)

```
UserAMFiber3AxisTValListStruct *UserAMFiber3AxisGetBBoxMaxTVals(
                                                const CagdCrvStruct *Crv,
                                                IrtRType Size)
```

Crv: the curve.

Size: the maximal dimension size for a bounding box.

Returns: A list of the requested t values.

Description: Get a List of t (paramteric) values that split a curve to small sub-curves (smaller than size).
See also:

15.2.23 UserAMFiber3AxisGetCrvsFromTValArray (am3axis_frag_split.c:443)

```
void UserAMFiber3AxisGetCrvsFromTValArray(const CagdCrvStruct *Crv,
                                          CagdCrvStruct **CrvArray,
                                          IrtRType *TVals,
                                          int First,
                                          int Last)
```

Crv: The original curve.

CrvArray: The array of sub curves.

TVals: The array of t values.

First: First location of arrays to handle (rest of the input ignored).

Last: Last location of arrays to handle (rest of the input ignored).

Returns: void

Description: Get an array of subcurves, for a given array of t values to split at.
See also:

15.2.24 UserAMFiber3AxisGetFragmentCrvs (am3axis_frag_split.c:488)

```
CagdCrvStruct *UserAMFiber3AxisGetFragmentCrvs(
                                  const UserAMFiber3AxisFragStruct *Fragments)
```

Fragments: The fragment struct.

Returns: List of subcurves.

Description: Get an list of subcurves, for a given fragment struct.
See also:

15.2.25 UserAMFiber3AxisGetKnotsTVals (am3axis_frag_split.c:184)

```
UserAMFiber3AxisTValListStruct *UserAMFiber3AxisGetKnotsTVals(
                                                    const CagdCrvStruct *Crv)
```

Crv: The curve.

Returns: A list of the requested t values.

Description: Get a List of t (paramteric) values at the curve knots.
See also:

15.2.26 UserAMFiber3AxisGetMonotoneTVals (am3axis_frag_split.c:109)

```
UserAMFiber3AxisTValListStruct *UserAMFiber3AxisGetMonotoneTVals(
                                                    const CagdCrvStruct *Crv)
```

Crv: The curve.

Returns: A list of the requested t values.

Description: Get a List of t (paramteric) values that split a curve to monotone sub-curves.
See also:

15.2.27 UserAMFiber3AxisGetTValFragments (am3axis_frag_split.c:40)

```
UserAMFiber3AxisFragStruct *UserAMFiber3AxisGetTValFragments(
                                    const CagdCrvStruct *Crv,
                                    const UserAMFiber3AxisTValListStruct *List)
```

Crv: The curve being split.

List: A list of t values to split at.

Returns: The struct containing the fragment values.

Description: Returns an array specifying curve fragments.
See also:

15.2.28 UserAMFiber3AxisMergeTValLists (am3axis_frag_split.c:353)

```
UserAMFiber3AxisTValListStruct *UserAMFiber3AxisMergeTValLists(
                                        UserAMFiber3AxisTValListStruct *List1,
                                        UserAMFiber3AxisTValListStruct *List2)
```

List1: List 1.

List2: List 2.

Returns: An ordered and merged list.

Description: Merge two given list of t values (resutls in an ordered list).
See also:

15.2.29 UserAMFiber3AxisOrderCrvs (am3axis_order_crvs.c:946)

```
CagdCrvStruct *UserAMFiber3AxisOrderCrvs(
                                    const UserAMFiber3AxisCrvOrderStruct *Crvs,
                                    int Num,
                                    IrtRType Radius,
                                    IrtRType XYRadius,
                                    IrtRType Angle,
                                    IrtRType ZOffs,
                                    IrtRType Accuracy)
```

Crvs: The curves (struct array).

Num: The number of curves.

Radius: The printing radius - diameter of the printed extruded material.

XYRadius: Bottom width of half the extruder bounding frustum.

Angle: Angle of the extruder bounding frustum.

ZOffs: Z offset of the extruder bounding frustum.

Accuracy: Subdivision accuracy of the computation (fragment size).

Returns: An ordered set of curves (probably subdivided).

Description: Order (for printing) and subdivide a list of curves.
See also:

15.2.30 UserAMFiber3AxisSaveCrvsAsSweeps (am3axis_order_crvs.c:1129)

```
void UserAMFiber3AxisSaveCrvsAsSweeps(const CagdCrvStruct *Crvs,
                                      const char *FileName)
```

Crvs: The curves to convert and save.

FileName: File to save the curves in.

Returns: void

Description: Save a list of curves as tube (sweep) models.
See also:

15.2.31 UserAMFiber3AxisSubCrvs (am3axis_sbtrct_crvs.c:235)

```
CagdCrvStruct *UserAMFiber3AxisSubCrvs(const CagdCrvStruct *SubCrvs,
                                       const CagdCrvStruct *FromCrvs,
                                       IrtRType Dist,
                                       IrtRType Accuracy,
                                       IrtBType Invert)
```

SubCrvs: The curves to subtract.

FromCrvs: The curves to subtract from.

Dist: The minimal distance of the subtraction, between fiber and ambient fibers.

Accuracy: The subdivision size (smaller = more accurate) - maximal bbox dimension of a divided segment.

Invert: Return only the subtracted part. Mostly for debugging.

Returns: The result curves.

Description: Subtract one group of curves from another.
See also:

15.2.32 UserBeltCreate (belts.c:61)

```
IPObjectStruct *UserBeltCreate(IPVertexStruct *Circs,
                               IrtRType BeltThickness,
                               IrtRType BoundingArcs,
                               int ReturnCrvs,
                               int *Intersects,
                               const char **Error)
```

Circs: A sequence of circles (pulleys), each as (x, y, r). If r is positive is it a CW pulley, otherwise CCW.

BeltThickness: The thickness of the constructed belt.

BoundingArcs: If non zero, bounding arcs are computed for each linear segment in belt, for each of the two sides of the belt.

ReturnCrvs: TRUE to simply return two closed curves with the left and right sides of the belt. FALSE to return a list with the individual arcs and lines and their attributes.

Intersects: TRUE if left and right sides intersects.

Error: If not set to NULL, holds an error description.

Returns: Two lists of lines/arcs representing the two sides the belt around the given circles (pulleys). If BoundingArcs is positive, two additional lists of arcs, each bounding a line segment in the belt, are build. The bounding arc is expanding the belt's domain with at most distance BoundingArc, from the linear segment.

Description: Builds a belt-curve formed out of a pair of sequences of lines/arcs around the given set of circles (pulleys), in order.

15.2.33 UserCABreakLiNCrvsAtAngularDev (crv_arng.c:705)

```
int UserCABreakLiNCrvsAtAngularDev(UserCrvArngmntStruct *CA,
                                   IrtRType AngularDeviation)
```

CA: The curves arrangement to process.

AngularDeviation: Maximal angular deviation to allow, in degrees.

Returns: TRUE if successful, FALSE otherwise.

Description: Break given linear curves at any C1 discont. that is showing an angular * deviation larger than AngularDeviation. *
See also: UserCrvArngmntCreate,

15.2.34 UserCAMergeCrvsAtAngularDev (crv_arng.c:552)

```
int UserCAMergeCrvsAtAngularDev(UserCrvArngmntStruct *CA,
                                IrtRType AngularDeviation,
                                IrtRType PtPtEps)
```

CA: The curves arrangement to process.

AngularDeviation: Maximal angular deviation to allow, in degrees.

PtPtEps: Epsilon when two points are considered the same.

Returns: TRUE if successful, FALSE otherwise.

Description: Merge given curves that share an end point if their angular deviation is smaller than AngularDeviation.

See also: UserCrvArngmntCreate,

15.2.35 UserClipSrfAtPlane (srf_cntr.c:1187)

```
TrimSrfStruct *UserClipSrfAtPlane(const CagdSrfStruct *Srf,
                                  const IrtPlnType Pln)
```

Srf: To clip against the plane.

Pln: The plane with which to clip Srf.

Returns: A list of trimmed surfaces resulting from the clipping operation. Can be empty.

Description: Clip the given surface, Srf, against a plane. Return a list of trimmed surfaces representing the parts of Srf above the plane.

15.2.36 UserCntrEvalToE3 (srf_cntr.c:449)

```
IPPolygonStruct *UserCntrEvalToE3(const CagdSrfStruct *Srf,
                                  IPPolygonStruct *Cntrs,
                                  UserCntrIsValidCntrPtFuncType ValidCntrPtFunc)
```

Srf: Surface to evaluate contours on.

Cntrs: Contours to evaluate, in place. The input contours should hold UV coordinates in the YZ coefficients of the points.

ValidCntrPtFunc: Each point along the contours is validated through this function if !NULL. In valid points break the contour and are purged away.

Returns: Mapped and validated contour in 3-space.

Description: Evaluate contours Cntrs, in place, into Euclidean space. Input UV contours are mapped through Srf to yield 3-space points. In the process all contour points are being validated through ValidCntrPtFunc, if not NULL.

See also: UserCntrSrfWithPlane,

15.2.37 UserCntrSrfWithPlane (srf_cntr.c:249)

contouring

```
IPPolygonStruct *UserCntrSrfWithPlane(const CagdSrfStruct *Srf,
                                      const IrtPlnType Plane,
                                      IrtRType SubdivTol,
                                      int UseSSI,
                                      int Euclidean)
```

Srf: To approximate its intersection with the given plane.

Plane: To intersect with the given surface.

SubdivTol: Control of subdivision tolerance if UseSSI (see MvarSrfSrfInter function). Control of polygonal approximation of surface (See IritSurface2Polygons function) if !UseSSI.

UseSSI: TRUE to use the multivariate SSI abilities, FALSE to compute a polygonal approximation intersection.

Euclidean: if FALSE and UseSSI is TRUE, returns the contours in the UV parameter space of Srf. Otherwise, contours are returned in Euclidean space.

Returns: A list of polylines approximating the contour.

Description: Computes the intersection of a freeform surface and a plane. If Srf is a scalar surface then it is promoted first into a 3D Euclidean surface with UV as YZ coordinates (and X has scalar field).
See also: UserCntrEvalToE3, MdlInterSrfByPlane,

15.2.38 UserCnvrtCagdPolyline2IritPolyline (usrcnvrt.c:146)

```
IPPolygonStruct *UserCnvrtCagdPolyline2IritPolyline(const CagdPolylineStruct
                                                                    *Poly)
```

Poly: Input polyline to convert into am Irit polyline.

Returns: Converted polyline.

Description: Converts a polyline into an Irit polyline.
See also: CagdCnvrtLinBspCrv2Polyline, CagdCnvrtPolyline2LinBspCrv, , CagdCnvrtPolyline2PtList, UserCnvrtCagdPolylines2IritPolylines,

15.2.39 UserCnvrtCagdPolylines2IritPolylines (usrcnvrt.c:182)

```
IPPolygonStruct *UserCnvrtCagdPolylines2IritPolylines(const CagdPolylineStruct
                                                                   *Polys)
```

Polys: Input polylines to convert into Irit polylines.

Returns: Converted polylines.

Description: Converts a list of polylines into an Irit polylines.
See also: CagdCnvrtLinBspCrv2Polyline, CagdCnvrtPolyline2LinBspCrv, , CagdCnvrtPolyline2PtList, UserCnvrtCagdPolyline2IritPolyline,

15.2.40 UserCnvrtIritPolyline2CagdPolyline (usrcnvrt.c:214)

```
CagdPolylineStruct *UserCnvrtIritPolyline2CagdPolyline(const IPPolygonStruct
                                                                    *Plln)
```

Plln: Input IRIT polyline to convert into a cagd polyline.

Returns: Converted polyline.

Description: Converts an IRIT polyline to a cagd polyline.
See also: CagdCnvrtLinBspCrv2Polyline, CagdCnvrtPolyline2LinBspCrv, , CagdCnvrtPtList2Polyline, UserCnvrtLinBspCrvs2IritPolyline,

15.2.41 UserCnvrtLinBspCrv2IritPolyline (usrcnvrt.c:250)

```
IPPolygonStruct *UserCnvrtLinBspCrv2IritPolyline(const CagdCrvStruct *Crv,
                                                 int FilterIdentical)
```

Crv: Input linear curve to convert into an Irit polyline.

FilterIdentical: TRUE to filter almost identical adjacent vertices.

Returns: Converted polyline.

Description: Converts a cagd linear curve into an Irit polyline.
See also: CagdCnvrtLinBspCrv2Polyline, CagdCnvrtPolyline2LinBspCrv, , CagdCnvrtPolyline2PtList, UserCnvrtCagdPolyline2IritPolyline, , UserCnvrtLinBspCrvs2IritPolylines, UserCnvrtIritPolyline2CagdPolyline,

15.2.42 UserCnvrtLinBspCrvs2IritPolylines (usrcnvrt.c:282)

```
IPPolygonStruct *UserCnvrtLinBspCrvs2IritPolylines(const CagdCrvStruct *Crvs,
                                                   int FilterIdentical)
```

Crvs: Input linear curves to convert into Irit polylines.

FilterIdentical: TRUE to filter almost identical adjacent vertices.

Returns: Converted polylines.

Description: Converts a list of cagd linear curves into Irit polylines.
See also: CagdCnvrtLinBspCrv2Polyline, CagdCnvrtPolyline2LinBspCrv, , CagdCnvrtPolyline2PtList, UserCnvrtCagdPolyline2IritPolyline, , UserCnvrtLinBspCrv2IritPolyline,

15.2.43 UserCnvrtObjApproxLowOrderBzr (usrcnvrt.c:314)

```
void UserCnvrtObjApproxLowOrderBzr(IPObjectStruct *Obj, int ApproxLowOrder)
```

Obj: To lower order approximate (curves, surfaces, trivariates).

ApproxLowOrder: Order to use - either 3 or 4 for quadratic or cubic.

Returns: void

Description: Lower order approximate the given freeform (hierarchy of) object(s), in place. For B-splines. the fitting is performed for each patch with no interior knots. This approximating ensures the interpolation of (end condition) end points.

15.2.44 UserConservativeClipSrfByPlane (srf_cntr.c:1292)

```
CagdSrfStruct *UserConservativeClipSrfByPlane(const CagdSrfStruct *Srf,
                                              const IrtPlnType Pln)
```

Srf: To clip against the plane.

Pln: The plane with which to clip Srf.

Returns: A tensor-product surface which results from the "conservative" clipping operation.

Description: Given a surface (Srf) and a plane (Pln), clip as much from Srf below Pln as possible, while keeping the result a tensor-product surface. That is, clip Srf against Pln (resulting in trimmed geometry), and return the minimal tensor-product surface containing the clipped geometry.
See also: UserClipSrfAtPlane,

15.2.45 UserCrvAngleMap (visible.c:614)

```
IPObjectStruct *UserCrvAngleMap(const CagdCrvStruct *Crv,
                                CagdRType SubdivTol,
                                CagdRType Angle)
```

Crv: Planar closed curve to compute its para/ortho/angular map.

SubdivTol: Of computation. If negative the function whose zero set is the orthogonal map, is returned.

Angle: 0 for parallel maps, 90 for orthogonal maps, or general for general angles.

Returns: Polyline(s) of (r, t) points such that the normals at C(r) and C(t) are orthogonal.

Description: Computes the parallel/orthogonality/angular map of the given planar closed curve. The orthogonality map is the set of pairs of points of Crv that have an orthogonal normal and is a 2D map of size [D x D] where D is the domain of Crv. Similarly the parallel map is the set of pairs of points of Crv that have the same normal direction, and the angular map identifies pairs of points with a prescribed fixed angle between their normals.
See also: UserCrvOMDiagExtreme, UserCrvViewMap, UserCrvVisibleRegions,

15.2.46 UserCrvArngmnt (crv_arng.c:264)

```
UserCrvArngmntStruct *UserCrvArngmnt(UserCAOpType Operation,
                                     const UserCrvArngmntStruct *CA,
                                     const void *Params[])
```

Operation: Create, BreakCrv, Report, Free, etc. (See UserCAOpType).

CA: To operate on its copy or NULL if a new CA.

Params: An array of params depending on Operation.

Returns: Constructed or updated CA.

Description: Main entry points of CA. gets the desired operation to perform, the CA to operate on (or NULL if new one) and parameters that depends on the specific desired operation:

```
USER_CA_OPER_CREATE - Creates a new CA
    Params[0] = List of curves/polylines/trimmed surfaces to extract the
                curves/linear curves/trimming curves for the arrangement.
    Params[1] = Tolerance for considering end points equal.
    Params[2] = Planarity tolerance to consider arrangement planar.
    Params[3] = TRUE to project all curves to be on computed plane.
    Params[4] = Mask for input type to consider:
                0x01 to handle polylines.
                0x02 to handle curves.
                0x04 to handle trimming curves in trimmed surfaces.
USER_CA_OPER_COPY - Creates a new CA
    None.
USER_CA_OPER_FILTER_DUP - Creates a new CA
    Params[0] = Epsilon to consider the curves the same.
    Params[1] = TRUE to update end points to be the same.
USER_CA_OPER_FILTER_TAN - Creates a new CA
    Params[0] = Epsilon angle in degrees to consider two curves with
                the same tangent.
USER_CA_OPER_SPLIT_CRV - Creates a new CA
    Params[0] = Mask for splitting type to consider:
                USER_CA_SPLIT_INFLECTION_PTS to split at inflection pts.
                USER_CA_SPLIT_MAX_CRVTR_PTS to split at max curvatures.
                USER_CA_SPLIT_C1DISCONT_PTS to split at parametric C1
                    discontinuities,  according to the knot sequence.
                USER_CA_SPLIT_REAL_C1DISCONT_PTS to split at actual C1
                    disconts, examined in Euclidean space.
    Params[1] = Tolerance of splitting computation.
USER_CA_OPER_BREAK_LIN - Creates a new CA
    Params[0] = Angular deviation (in degrees) to split linear curves at.
USER_CA_OPER_BREAK_INTER - Creates a new CA
    Params[0] = Intersection computation tolerance.
USER_CA_OPER_BREAK_NEAR_PTS - Creates a new CA
    Params[0] = Number of points to split curves at.
    Params[1] = A list object of pts to examine and split if near them.
    Params[2] = Tolerance to consider a point near/on a curve.
USER_CA_OPER_UNION_CRV - Creates a new CA
    Params[0] = Angular deviation (in degrees) to merge C1 discont
                curves at.
USER_CA_OPER_LSTSQR_CRV - Creates a new CA
    Params[0] = Fitting Parameter to fit smooth quadratic C1 curves to
                linear curves.  Higher order curves are not affected.
                    If Params[0] positive, the fitted curve size is set
                to InputCrvSize * FitC1Crv / 100 (i.e. Params[0] serves
                as percentage of input size.
                    If Params[0] negative, the Fitted curve size is
                simply set to ABS(Params[0]).
USER_CA_OPER_EVAL_CA - Operates on input CA in place
    Params[0] = TRUE to ignore hanging curves that join other curves at
                only one of their end points.
USER_CA_OPER_CLASSIFY - Operates on input CA in place.
    No parameters.
USER_CA_OPER_REPORT - Operates on input CA in place
```

```
    Params[0] = A mask of desired report:
                0x01 to dump info on crvs. 0x02 to also dump the crvs.
                0x04 to report end pts in arrangement if evaluated.
                0x08 to report regions in arrangement if evaluated.
USER_CA_OPER_OUTPUT - Operates on input CA in place
    Params[0] = Style of expected output:
                1 for individual crv segs in each region (loop etc.),
                2 for merged curves so every region is one curve,
                3 for topology as an ordered list of curve segments and
                  each region is a list of indices into the first list.
                  A negative -i index means index i but a reversed crv.
                101, 102, 103: same as 1,2,3 but pt is evaluated at 1/13
                  of curve parametric domain to identify orientation.
    Params[1] = Tolerance of topology reconstruction (in case 3 only).
    Params[2] = Zoffset in Z for the i'th region, by amount i*ZOffset.
USER_CA_OPER_FREE - Operates on input CA in place
    None.
```

See also:

15.2.47 UserCrvArngmntClassifyConnectedRegions (crv_arng.c:1992)

```
int UserCrvArngmntClassifyConnectedRegions(UserCrvArngmntStruct *CA)
```

CA: Curves' arrangement to process.

Returns: TRUE if successful, FALSE otherwise.

Description: Create regions (loops, etc.) from the given arrangement and classify them.
See also: UserCrvArngmntCreate,

15.2.48 UserCrvArngmntCopy (crv_arng.c:1412)

```
UserCrvArngmntStruct *UserCrvArngmntCopy(const UserCrvArngmntStruct *CA)
```

CA: The curve arrangement structure to copy.

Returns: NULL if error or new copy of CA.

Description: Copy all the data structure used in the curve arrangement.
See also: UserCrvArngmntCreate,

15.2.49 UserCrvArngmntCreate (crv_arng.c:430)

```
UserCrvArngmntStruct *UserCrvArngmntCreate(const IPObjectStruct *PCrvs,
                                           CagdRType EndPtEndPtTol,
                                           CagdRType PlanarityTol,
                                           int ProjectOnPlane,
                                           int InputMaskType)
```

PCrvs: Input to derive its planar arrangement. Can be curves, polylines, or trimming curves in trimmed surfaces.

EndPtEndPtTol: Tolerance to consider two (end) points the same.

PlanarityTol: Tolerance to accept all curves as planar.

ProjectOnPlane: TRUE to project off-plane curves onto the plane.

InputMaskType: Bit mask controlling the type of entities to process: 0x01 - process polylines in the input. 0x02 - process curves in the input. 0x04 - process trimming curves in trimmed surfaces in the input.

Returns: A curves' arrangement structure, or NULL if err.

Description: Constructs a new curves' arrangement data structure.
See also:

15.2.50 UserCrvArngmntFilterDups (crv_arng.c:803)

```
int UserCrvArngmntFilterDups(UserCrvArngmntStruct *CA,
                             CagdBType UpdateEndPts,
                             CagdRType EndPtEndPtTol,
                             CagdRType Eps)
```

CA: The curves arrangement to process.

UpdateEndPts: TRUE to update end points at purged segments to the precise position of the retained segment.

EndPtEndPtTol: Tolerance to consider two (end) points the same.

Eps: To consider curves same and Pt-Crv distances to consider same.

Returns: TRUE if successful, FALSE otherwise.

Description: Filters out curves that are duplicated, leaving only one instance. Also seeks partial overlaps and filter such cases as well.
See also: UserCrvArngmntCreate,

15.2.51 UserCrvArngmntFilterTans (crv_arng.c:945)

```
int UserCrvArngmntFilterTans(UserCrvArngmntStruct *CA, CagdRType FilterTans)
```

CA: The curve arrangement structure to modify tangencies.

FilterTans: Tolerance in degrees of two angles to be the same (tangent).

Returns: TRUE if successful, FALSE otherwise.

Description: Scans all shared end points and modify end curves so no two curves will be tangent at the end points.
See also: UserCrvArngmntCreate,

15.2.52 UserCrvArngmntFree (crv_arng.c:1447)

```
int UserCrvArngmntFree(UserCrvArngmntStruct *CA)
```

CA: The curve arrangement structure to free.

Returns: TRUE if successful, FALSE otherwise.

Description: Delete all the data structure used in the curve arrangement.
See also: UserCrvArngmntCreate,

15.2.53 UserCrvArngmntGetCurves (crv_arng.c:3252)

```
CagdCrvStruct *UserCrvArngmntGetCurves(UserCrvArngmntStruct *CA, int XYCurves)
```

CA: Curves' arrangement to process.

XYCurves: TRUE to return curves in XY plane, FASLE to recover the original orientation.

Returns: Fetched curves.

Description: Fetches the curves of the arrangement.
See also:

15.2.54 UserCrvArngmntIsContained (crv_arng.c:3109)

```
int UserCrvArngmntIsContained(const UserCrvArngmntStruct *CA,
                              const CagdCrvStruct *InnerShape,
                              const CagdCrvStruct *OuterLoop)
```

CA: Curves' arrangement to process.

InnerShape: A shape to test if contained in OuterLoop.

OuterLoop: A closed loop region to test if contains InnerShape.

Returns: TRUE if OuterLoop contains InnerShape, FALSE otherwise.

Description: Test if OuterLoop contains the given InnerShape. Both inputs should be merged regions in CA.
See also: SymbCrvPointInclusion,

15.2.55 UserCrvArngmntIsContained2 (crv_arng.c:3143)

```
int UserCrvArngmntIsContained2(const UserCrvArngmntStruct *CA,
                               const CagdPType Pt,
                               const CagdCrvStruct *Loop)
```

CA: Curves' arrangement to process.

Pt: A Point to test for inclusion in Loop.

Loop: A closed loop region to test if contains Pt.

Returns: TRUE if Loop contains Pt, FALSE otherwise.

Description: Test if Loop contains the given Point Pt. Input should be of merged region in CA.
See also: SymbCrvPointInclusion,

15.2.56 UserCrvArngmntLinearCrvsFitC1 (crv_arng.c:1335)

```
int UserCrvArngmntLinearCrvsFitC1(UserCrvArngmntStruct *CA, int FitSize)
```

CA: The curves arrangement to process.

FitSize: If FitC1Crv positive, the Fitted curve size is set to InputCrvSize * FitC1Crv / 100 (i.e. FitSize serves as percetange of input size). If FitSize negative, the Fitted curve size is simply set to ABS(FitC1Crv).

Returns: TRUE if successful, FALSE otherwise.

Description: Approximate linear curves using quadratic C1 curves, in place. Other curves or linear curves with size smaller that (negative) FitSize are transferred to the output as is.
See also: UserCrvArngmntCreate,

15.2.57 UserCrvArngmntOutput (crv_arng.c:3748)

```
int UserCrvArngmntOutput(const UserCrvArngmntStruct *CA,
                         int OutputStyle,
                         CagdRType Tolerance,
                         CagdRType ZOffset)
```

CA: Curves' arrangement to output. Output slot of CA will be updated with result.

OutputStyle: 1 for individual curve segments in each region (loop etc.), 2 for merged curves so every region is one curve, 3 for topology as an ordered list of curve segments and each region is a list of indices into the first list. A negative -i index means index i but a reversed curve. 101, 102, 103 - same as 1,2,3 but a pt is evaluated at 1/13 of curve parametric domain to identify orientation.

Tolerance: Tolerance of intersection locations' computation.

ZOffset: if positive, offset the n'th region in Z by n*ZOffset.

Returns: TRUE if successful, FALSE otherwise.

Description: Emit the result of the curve's arrangement computations. Output is placed on the Output slot of CA.
See also:

15.2.58 UserCrvArngmntPrepEval (crv_arng.c:1783)

```
int UserCrvArngmntPrepEval(UserCrvArngmntStruct *CA)
```

CA: Curves' arrangement to process.

Returns: TRUE if successful, FALSE otherwise.

Description: Prepare the CA for full evaluation - build the Pts and Crv vectors.
See also: UserCrvArngmntCreate,

15.2.59 UserCrvArngmntProcessEndPts (crv_arng.c:1896)

```
int UserCrvArngmntProcessEndPts(UserCrvArngmntStruct *CA)
```

CA: Curves' arrangement to process.

Returns: TRUE if successful, FALSE otherwise.

Description: Detect, up to EndPtTol, end points of curves that can be considered the same and merge these points.
See also: UserCrvArngmntCreate,

15.2.60 UserCrvArngmntProcessIntersections (crv_arng.c:1608)

```
int UserCrvArngmntProcessIntersections(UserCrvArngmntStruct *CA,
                                       CagdRType Tolerance)
```

CA: The curves arrangement to process.

Tolerance: Tolerance of intersection locations' computation.

Returns: TRUE if intersections were detected, FALSE if none.

Description: Computes all intersections in the given list of curves in XY plane and split intersecting curves at the intersection locations.
See also: UserCrvArngmntCreate,

15.2.61 UserCrvArngmntProcessSpecialPts (crv_arng.c:1671)

```
int UserCrvArngmntProcessSpecialPts(UserCrvArngmntStruct *CA,
                                    CagdRType Tolerance,
                                    UserCASplitType CrvSplit)
```

CA: The curves arrangement to process.

Tolerance: Tolerance of intersection locations' computation.

CrvSplit: A mask setting the type pf points to split at.

Returns: TRUE if successful, FALSE otherwise.

Description: Splits all input curves at special locations such as C1 discontinuities and inflections points, in place.
See also: UserCrvArngmntCreate,

15.2.62 UserCrvArngmntRegion2Curves (crv_arng.c:3171)

```
static IPObjectStruct *UserCrvArngmntRegion2Curves(const UserCrvArngmntStruct
                                                                    *CA,
                                                   UserCARefCrvStruct
                                                                *CARefCrv,
                                                   int Merge)
```

CA: Curves' arrangement to process.

CARefCrv: The regions as a list of references to curves.

Merge: If TRUE, merge all curves in a region into one.

Returns: A (list of) curve(s), or NULL if error.

Description: Convert one internal region as lists of curves, into (a merged) Bspline curve(s).
See also: UserCrvArngmntRegion2Curves,

15.2.63 UserCrvArngmntRegions2Curves (crv_arng.c:3294)

```
int UserCrvArngmntRegions2Curves(const UserCrvArngmntStruct *CA,
                                 int Merge,
                                 int XYCurves,
                                 IrtRType ZOffset)
```

CA: Curves' arrangement to process.

Merge: If TRUE, merge all curves in a region into one.

XYCurves: TRUE to return regions in XY plane, FASLE to recover the original orientation.

ZOffset: if positive, offset the n'th region in Z by n*ZOffset.

Returns: TRUE if successful, FALSE otherwise.

Description: Convert internal regions as lists of curves, into merged B-spline curves. Output is placed on the Output slot of CA.
See also:

15.2.64 UserCrvArngmntRegionsTopology (crv_arng.c:3408)

```
int UserCrvArngmntRegionsTopology(const UserCrvArngmntStruct *CA,
                                  int XYCurves,
                                  IrtRType ZOffset)
```

CA: Curves' arrangement to process.

XYCurves: TRUE to return regions in XY plane, FASLE to recover the original orientation.

ZOffset: if positive, offset the n'th region in Z by n*ZOffset.

Returns: TRUE if successful, FALSE otherwise.

Description: Convert internal regions into full topology as a list of two lists:

1. First list is the list of all curves in the arrangements' output.
2. Second list is a list of entities each of which holds a list of indices into the first list. Negative indices indicates the curve should be reversed. Index of first curve is 1. Output is placed on the Output slot of CA.

See also:

15.2.65 UserCrvArngmntReport (crv_arng.c:3585)

```
void UserCrvArngmntReport(const UserCrvArngmntStruct *CA,
                          int DumpCurves,
                          int DumpPts,
                          int DumpRegions,
                          int DumpXYData)
```

CA: To print to stdout.

DumpCurves: 1 to dump info crvs in the CA. 2 to also dump the crvs.

DumpPts: TRUE to dump the points in the CA.

DumpRegions: TRUE to dump the regions in the CA.

DumpXYData: TRUE to dump XY data, FALSE for 3-space data (crvs and pts).

Returns: void

Description: Dumps to stdout, the content of the curve arrangement.
See also:

15.2.66 UserCrvArngmntSplitAtPts (crv_arng.c:1173)

```
int UserCrvArngmntSplitAtPts(UserCrvArngmntStruct *CA,
                             const IPObjectStruct *PtsObj,
                             CagdRType Eps)
```

CA: The curves arrangement to process.

PtsObj: List of points to examine of on curves and if so split curves there. if NULL uses all end points in all given curves.

Eps: to consider curves same and Pt-Crv distances to consider same.

Returns: TRUE if successful, FALSE otherwise.

Description: Filters out curves that are duplicated, leaving only one instance. Also seeks partial overlaps and filter such cases as well.
See also: UserCrvArngmntCreate,

15.2.67 UserCrvCrvtrByOneCtlPt (crvtranl.c:52)

curvature

```
IPObjectStruct *UserCrvCrvtrByOneCtlPt(const CagdCrvStruct *Crv,
                                       int CtlPtIdx,
                                       CagdRType Min,
                                       CagdRType Max,
                                       CagdRType SubdivTol,
                                       CagdRType NumerTol,
                                       int Operation)
```

Crv: To compute its curvature behaviour (convex vs. concave) as a function of the curve parameter and the Euclidean coordiate of the CtlPtIdx's control point.

CtlPtIdx: Index of control point to make a parameter for the curvature.

Min, Max: Domain each coordinate of CtlPtIdx point should vary.

SubdivTol, NumerTol: Tolerance for the silhouette solving, if any.

Operation: 1. Returned is a multivariate of dimension "1 + Dim(Crv)", where Dim(Crv) is the dimension of curve (E2, E3, etc.). 2. Extract the zero set of 1. using marching cubes. 3. Computes the t's silhouette of the 1. by simultaneously solving for 1 and its derivative with respect to t. 4. Same as 3 but evaluate the result into Euclidean space.

Returns: Either the multivariate (1 above) or its t's silhouette (2 above).

Description: Given a parametric curve, Crv, and a control point index CtlPtIdx, compute the curvature sign field of the curve as function of the Euclidean locations of control point index CtlPtIdx.
See also: MvarCrvCrvtrByOneCtlPt, SymbCrv2DCurvatureSign, MvarCrvMakeCtlPtParam,

15.2.68 UserCrvNetwork2Tile (crv_net_tiles.c:649)

```
IPObjectStruct *UserCrvNetwork2Tile(const IPObjectStruct *CrvList,
                                    CagdRType DfltOffset,
                                    CagdRType Tolerance,
                                    CagdRType ExtrudeAmount,
                                    UserCrvNetworkOutType OutType)
```

CrvList: A List of curves in which each curve may or may not have offset information field (a scalar curve) stored as attributes.

DfltOffset: Constant offset amount to be used for curves which do not have offset information field stored as attributes.

Tolerance: Of computation.

ExtrudeAmount: If OutType extruded trivariates, sets extrusion amount.

OutType: 0 for Univariate. 1 for bivariate. 2 for extruded trivariates.

Returns: The constructed tile object.

Description: Given a list of curves with or without offset information as attributes processes it into univariate outline curve tile or bivariate tile or extruded trivariate tile based on the Type asked for.

15.2.69 UserCrvOMDiagExtreme (visible.c:915)

```
IPObjectStruct *UserCrvOMDiagExtreme(const CagdCrvStruct *Crv,
                                     const IPObjectStruct *OM,
                                     int DiagExtRes)
```

Crv: Curve for which we computed the orthogonality map and now seek the diagonal extremes.

OM: The computed othogonality map of Crv.

DiagExtRes: The resolution of the Z-buffer to use to extract the diagonal extreme.

Returns: A polyline object with two polylines, upper diagonal extreme followed by lower diagonal extreme.

Description: Computes the diagonal extremes of the Orthogonality map of curve Crv. Uses a [DiagExtRes x 1] Z-buffer to extract th diagnoal extremes.
See also: UserCrvAngleMap,

15.2.70 UserCrvViewMap (visible.c:751)

```
IPObjectStruct *UserCrvViewMap(const CagdCrvStruct *Crv,
                               const CagdCrvStruct *ViewCrv,
                               CagdRType SubTol,
                               CagdRType NumTol,
                               CagdBType TrimInvisible)
```

Crv: Planar closed curve to compute its view map.

ViewCrv: Planar viewing direction curve in vector space.

SubTol: Used only if TrimInvisible TRUE to control the subdivision Tolerance of the solver. If negative, the solution set is returned as a cloud of points.

NumTol: Of computation. If negative the function M whose zero set is the view map, is returned.

TrimInvisible: If TRUE, trim the regions that are invisible from V.

Returns: Polyline(s) of (Theta, t) or (Theta, t, r) points such that normals at C(t) are orthogonal to ViewCrv(Theta), and C(t) exactly in front of C(r) if (Theta, t, r).

Description: Computes the view map of the given planar closed curve, C(t), with respect to view directions that are prescribed by vector curve ViewCrv(Theta). The view map is defined as the zero of M = < N(t), ViewCrv(Theta) >, where N(t) is the normal field of C(t). Seeking the visible portions, the problem could be partially addressed using the following constraints:

```
< C(t) - C(r), ViewCrv'(Theta) > = 0,
```

finding all points C(t) behind or in front C(r), for some r, such that

```
< C(t) - C(r), ViewCrv(Theta) > > 0,
```

for all matched r values.
See also: UserCrvOMDiagExtreme, UserCrvAngleMap, UserCrvVisibleRegions,

15.2.71 UserCrvVisibleRegions (visible.c:440)

```
CagdCrvStruct *UserCrvVisibleRegions(const CagdCrvStruct *CCrv,
                                     const CagdRType *View,
                                     CagdRType Tolerance)
```

CCrv: Planar curve to compute its visible regions from View.

View: As (x, y, w) where w == 0 for parallel view direction and w = 1 for a point view.

Tolerance: Of computation.

Returns: List of curve segments of Crv that are visible from View.

Description: Computes the regions of planar curve Crv that are visible from a view point (View[2] == 1) or direction (View[2] == 0) View (x, y, w). Return is a list of visible curve segments.
See also: UserCrvOMDiagExtreme, UserCrvAngleMap, UserCrvViewMap,

15.2.72 UserCrvs2DBooleans (crv_bool.c:103)

```
struct IPObjectStruct *UserCrvs2DBooleans(const CagdCrvStruct *Crvs1,
                                          const CagdCrvStruct *Crvs2,
                                          BoolOperType BoolOper,
                                          int MergeLoops,
                                          int *ResultState,
                                          const CagdRType *Tols,
                                          CagdRType ZOffset)
```

Crvs1, Crvs2: Two inputs of (one or more) loops to perform 2D Booleans.

BoolOper: The Boolean operation requested, OR, AND, etc.

MergeLoops: TRUE to merge loops into a single curve.

ResultState: Of operation - 0 if everything is computed fine. -1 if the curves are not intersecting and not contained in one another. 1 if the curves are not intersecting but 1 is inside 2. 2 if the curves are not intersecting but 2 is inside 1.

Tols: Optional vector of four elements to overwrite the default tolerances (EndPtEndPtTol, PlanarityTol, ProjectOnPlane, InputMaskType). See UserCrvArngmntCreate for more. Can be NULL to employ defaults.

ZOffset: Amount to ZOffset the n curve segment by n*ZOffset.

Returns: The result as list of loops, each loop holding the curve segments of the loop, and each curve tagged using "CrvSourceID" if from first (0x01) curve(s), second (0x02) input curve(s) or both (0x03).

Description: Compute the 2D (XY) Booleans of two given sets of closed curves. Crvs1 and Crvs2 should both hold one (or more) closed loops of curves.
See also: UserCrvArngmntCreate,

15.2.73 UserDDMPolysOverPolys (ddm_text.c:464)

```
IPObjectStruct *UserDDMPolysOverPolys(const IPObjectStruct *PlSrf,
                                      const IPObjectStruct *Texture,
                                      IrtRType UDup,
                                      IrtRType VDup,
                                      int LclUV)
```

PlSrf: Polygonal mesh surface to derive a DDM mapping for. This mesh must have a parametrization as UV coordinates as well as normals at the vertices of the polygons.

Texture: Geometry defining the single tile of the DDM texture. The geometry is assumed to span [0..1] in both x and y.

UDup, VDup: The U and V duplication factors.

LclUV: TRUE to keep local UV coordinates for each tile, FALSE to employ the UV coordinates of the surface Srf.

Returns: A set of objects that derive the DDM surface above Srf.

Description: Maps the given DDM texture defined in Texture over mesh surface PlSrf, duplicating it (UDup x VDup) times over the parametric domain of mesh.
See also: UserDDMPolysOverSrf,

15.2.74 UserDDMPolysOverSrf (ddm_text.c:137)

```
IPObjectStruct *UserDDMPolysOverSrf(const CagdSrfStruct *Srf,
                                    const IPObjectStruct *Texture,
                                    IrtRType UDup,
                                    IrtRType VDup,
                                    int LclUV,
                                    int Random)
```

Srf: Surface to derive a DDM mapping for.

Texture: Geometry defining the single tile of the DDM texture. The geometry is assumed to span [0..1] in both x and y.

UDup, VDup: The U and V duplication factors.

LclUV: TRUE to keep local UV coordinates for each tile, FALSE to employ the UV coordinates of the surface Srf.

Random: If TRUE, the placement is made at random over the surface.

Returns: A set of objects that derive the DDM surface above Srf.

Description: Maps the given DDM texture defined in Texture over the surface Srf, duplicating it (UDup x VDup) times over the parametric domain of Srf.
See also: UserDDMPolysOverPolys,

15.2.75 UserDDMPolysOverTrimmedSrf (ddm_text.c:92)

```
IPObjectStruct *UserDDMPolysOverTrimmedSrf(const TrimSrfStruct *TSrf,
                                           const IPObjectStruct *Texture,
                                           IrtRType UDup,
                                           IrtRType VDup,
                                           int LclUV,
                                           int Random)
```

TSrf: Trimmed surface to derive a DDM mapping for.

Texture: Geometry defining the single tile of the DDM texture. The geometry is assumed to span [0..1] in both x and y.

UDup, VDup: The U and V duplication factors.

LclUV: TRUE to keep local UV coordinates for each tile, FALSE to employ the UV coordinates of the surface TSrf.

Random: If TRUE, the placement is made at random over the trimmed srf.

Returns: A set of objects that derive the DDM surface above TSrf.

Description: Maps the given DDM texture defined in Texture over the trimmed surface TSrf, duplicating it (UDup x VDup) times over the parametric domain in non trimmed regions only.
See also: UserDDMPolysOverSrf, UserDDMPolysOverPolys,

15.2.76 UserDexelColorTriangles (dexels.c:2220)

```
IPObjectStruct *UserDexelColorTriangles(IPPolygonStruct *PolyList)
```

PolyList: The list of triangles to color.

Returns: List of two objects, each having triangles of a color.

Description: Colors the triangles with two colors based on cut and uncut portions. The two types of triangles are returned as a list of two objects.

15.2.77 UserDexelDxClearGrid (dexels.c:458)

```
void UserDexelDxClearGrid(UserDexelDxGridStruct *DxGrid)
```

DxGrid: The grid to clear, in place.

Returns: void

Description: Clears the given grid, in place.
See also: UserDexelDxGridFree,

15.2.78 UserDexelDxGridCopy (dexels.c:489)

```
void UserDexelDxGridCopy(UserDexelDxGridStruct *Dest,
                         const UserDexelDxGridStruct *Src)
```

Dest: The destination grid.

Src: The source.

Returns: void

Description: Creates a copy of the source grid into the destination, in place. Assumes a single dexel interval at each location of the source grid.
See also: UserDexelGridNew,

15.2.79 UserDexelDxGridFree (dexels.c:425)

```
void UserDexelDxGridFree(UserDexelDxGridStruct *DxGrid)
```

DxGrid: The grid to free.

Returns: void

Description: Frees the input grid.
See also: UserDexelGridNew, UserDexelDxClearGrid,

15.2.80 UserDexelDxGridSubtract (dexels.c:534)

```
void UserDexelDxGridSubtract(UserDexelDxGridStruct *GridA,
                             const UserDexelDxGridStruct *GridB)
```

GridA: The grid to subtract from.

GridB: The grid to subtract.

Returns: void

Description: Performs a boolean subtraction of GridB from GridA, in place.
See also: UserDexelDxGridUnion,

15.2.81 UserDexelDxGridUnion (dexels.c:633)

```
void UserDexelDxGridUnion(UserDexelDxGridStruct *GridA,
                          const UserDexelDxGridStruct *GridB)
```

GridA: The grid to unite and place answer in.

GridB: The grid to unite.

Returns: void

Description: Performs a boolean union of GridB with GridA, in place.
See also: UserDexelDxGridSubtract,

15.2.82 UserDexelGetDexelGridEnvelope0D (dexels.c:750)

```
UserDexelDxGridStruct *UserDexelGetDexelGridEnvelope0D(
                                                CagdPtStruct *EnvlPts,
                                                CagdPtStruct *EnvlNrmls,
                                                UserDexelDxGridStruct *Stock)
```

EnvlPts: The set of points.

EnvlNrmls: The set of normals.

Stock: The dexel grid to update.

Returns: Intersection as a dexel grid.

Description: Updates the given dexel grid with the given points and corresponding normals, in place.
See also: UserDexelGetDexelGridEnvelope1D,

15.2.83 UserDexelGetDexelGridEnvelope1D (dexels.c:805)

```
UserDexelDxGridStruct *UserDexelGetDexelGridEnvelope1D(
                                          CagdPolylineStruct *PlaneEnvelope,
                                          CagdPolylineStruct *EnvlNrmls,
                                          UserDexelDxGridStruct *Stock)
```

PlaneEnvelope: The given polyline.

EnvlNrmls: The corresponding normals to PlaneEnvelope.

Stock: The dexel grid to intersect PlaneEnvelope with.

Returns: Intersection as a dexel grid.

Description: Computes the intersection of the given polyline with the given dexel grid, in place.
See also: UserDexelGetDexelGridEnvelope0D,

15.2.84 UserDexelGridNew (dexels.c:377)

```
UserDexelDxGridStruct *UserDexelGridNew(int GridType,
                                        CagdRType Ori0,
                                        CagdRType End0,
                                        CagdRType Ori1,
                                        CagdRType End1,
                                        int NumDx0,
                                        int NumDx1)
```

GridType: 0,1,2 for dexels along x,y,z axes.

Ori0: Origin of grid in first direction.

End0: End of grid in first direction.

Ori1: Origin of grid in second direction.

End1: End of grid in second direction.

NumDx0: Number of dexels in first direction.

NumDx1: Number of dexels in second direction.

Returns: Newly allocated grid.

Description: Allocates an empty dexel grid with given specifications.
See also: UserDexelDxGridFree,

15.2.85 UserDexelInitStock (dexels.c:966)

```
void UserDexelInitStock(UserDexelDxGridStruct *DxGrid,
                        CagdRType Top,
                        CagdRType Btm)
```

DxGrid: The grid to initialize, in place.

Top: The top height value.

Btm: The bottom height value.

Returns: void

Description: Initializes the given dexel grid as a rectangular block with given top and bottom values, in place.
See also: UserDexelInitStockSrf, UserDexelInitStockSrf2,

15.2.86 UserDexelInitStockSrf (dexels.c:1158)

```
void UserDexelInitStockSrf(UserDexelDxGridStruct *DxGrid,
                           const CagdSrfStruct *Srf)
```

DxGrid: The grid to initialize, in place.

Srf: The closed surface.

Returns: void

Description: Initializes the given dexel grid in the shape of the given closed surface.
See also: UserDexelInitStockSrf2, UserDexelInitStock,

15.2.87 UserDexelInitStockSrf2 (dexels.c:1080)

```
void UserDexelInitStockSrf2(UserDexelDxGridStruct *DxGrid,
                            CagdSrfStruct *SrfList,
                            CagdRType BtmLevel)
```

DxGrid: The grid to initialize, in place.

SrfList: The list of surfaces which are not closed.

BtmLevel: The bottom height value.

Returns: void

Description: Initializes the given dexel grid in the shape of the given surfaces which are not closed and a bottom height value.
See also: UserDexelInitStockSrf, UserDexelInitStock,

15.2.88 UserDexelReadDexelGridFromFile (dexels.c:1311)

```
UserDexelDxGridStruct *UserDexelReadDexelGridFromFile(char *FileName)
```

FileName: The file name on the disk to read from.

Returns: The grid read from the file.

Description: Reads a dexel grid from a file on the disk.
See also: UserDexelWriteDexelGridToFile,

15.2.89 UserDexelTriangulateDxGrid (dexels.c:2264)

```
IPPolygonStruct *UserDexelTriangulateDxGrid(UserDexelDxGridStruct *DxGrid)
```

DxGrid: The dexel grid.

Returns: List of triangles.

Description: Fits triangles over the dexel grid.
See also:

15.2.90 UserDexelWriteDexelGridToFile (dexels.c:1245)

```
void UserDexelWriteDexelGridToFile(char *FileName,
                                   UserDexelDxGridStruct *DxGrid)
```

FileName: The file name on the disk.

DxGrid: The grid to write.

Returns: void

Description: Writes the given dexel grid to a file on the disk.
See also: UserDexelReadDexelGridFromFile,

15.2.91 UserDitherImagesByCurves (dtr_crvs.c:324)

```
IPObjectStruct *UserDitherImagesByCurves(
                            const UserDitherImagesByCurvesDBStruct *DtrDBInfo,
                            UserDitherImagesByCurvesInfoStruct *DtrInfo)
```

DtrDBInfo: Data base of dithering curves to employ.

DtrInfo: Dither specs, including images, and state (noise, FS, etc.).

Returns: Dithering images, or NULL if error.

Description: Dithers a set of images as given in DtrInfo, using a database of dithering curves found in DtrDBInfo.
See also: UserDitherImagesByCurvesGetImages, UserDitherImagesByCurvesFreeDB, , UserDitherImagesByCurvesGetDB, UserDitherImagesByCurvesSweepTubes,

15.2.92 UserDitherImagesByCurvesFreeDB (dtr_crvs.c:179)

```
void UserDitherImagesByCurvesFreeDB(UserDitherImagesByCurvesDBStruct
                                                                  *DtrDBInfo)
```

DtrDBInfo: To free.

Returns: void

Description: Free a data base of dither curves.
See also: UserDitherImagesByCurves, UserDitherImagesByCurvesGetImages, , UserDitherImagesByCurvesGetDB, ,

15.2.93 UserDitherImagesByCurvesGetDB (dtr_crvs.c:51)

```
int UserDitherImagesByCurvesGetDB(const char *DBFileName,
                                  int RequestedDim,
                                  UserDitherImagesByCurvesDBStruct **DtrDBInfo)
```

DBFileName: Name of DB file.

RequestedDim: Expected dim. (Number of simultaneously dither images).

DtrDBInfo: Where to save the read DB.

Returns: TRUE if successful, FALSe otherwise.

Description: Read a data base for dithering multiple images using freeform curves.
See also: UserDitherImagesByCurves, UserDitherImagesByCurvesGetImages, , UserDitherImagesByCurvesFreeDB,

15.2.94 UserDitherImagesByCurvesGetImages (dtr_crvs.c:216)

```
int UserDitherImagesByCurvesGetImages(
                                  UserDitherImagesByCurvesInfoStruct *DtrInfo,
                                  const char **Images)
```

DtrInfo: Where images are to be saved.

Images: File names of images to read. Number of expected images should be in Dim in DtrInfo.

Returns: TRUE if successful, FALSe otherwise.

Description: Read images from files to simultaneously dither them.
See also: UserDitherImagesByCurves, UserDitherImagesByCurvesFreeDB, , UserDitherImagesByCurvesGetDB, ,

15.2.95 UserDitherImagesByCurvesSweepTubes (dtr_crvs.c:262)

```
IPObjectStruct *UserDitherImagesByCurvesSweepTubes(
                                   UserDitherImagesByCurvesInfoStruct *DtrInfo,
                                   const IPObjectStruct *PCrvs)
```

DtrInfo: Dither specs, including images, and state (noise, FS, etc.).

PCrvs: Dither curves to sweep tubes through.

Returns: Swept dithering curves as thin sweep surfaces.

Description: Sweeps tubes through the given dither curves of prescribed diameter.
See also: UserDitherImagesByCurvesGetImages, UserDitherImagesByCurvesFreeDB, , UserDitherImagesByCurvesGetDB, ,

15.2.96 UserDivideMdlAtAllKnots (srf_cntr.c:1520)

```
MdlModelStruct *UserDivideMdlAtAllKnots(MdlModelStruct *Model,
                                        IrtPlnType Pln,
                                        int Axis,
                                        const CagdRType *KV,
                                        int MinKV,
                                        int MaxKV,
                                        CagdRType *Param)
```

Model: To intersect and subdivide.

Pln: Along which to divide Mdl, or NULL if Mdl need to be divided at Axis.

Axis: Along which to divideMdl. Ignored if Pln is not NULL.

KV: The knots to divide along.

MinKV, MaxKV: The KV range to consider.

Param: If detected an isoparametric values, set Param to it.

Returns: Subdivided model.

Description: Intersect and subdivide the given model, Mdl, at all locations Mdl cross a t = KV[i], MinKV <= i < MaxKV, knot plane. The plane is defined either as (Pln[0], Pln[1], Pln[2], t), or, if the parameter Pln is NULL, as Axis = t.
See also: UserDivideMdlsAtAllMVInteriorKnot, UserDivideOneMdlAtAllMVInteriorKnot,

15.2.97 UserDivideMdlsAtAllMVInteriorKnot (srf_cntr.c:1650)

```
MdlModelStruct *UserDivideMdlsAtAllMVInteriorKnot(const MdlModelStruct *Models,
                                                  const MvarMVStruct *MV)
```

Models: To subdivide.

MV: The multivariate.

Returns: List of models, subdivided at all interior knots of MV, as new models.

Description: Subdivide the give models, Mdls, embedded inside the domain of multivariate MV, at all locations Mdls crosses a knot plane in the domain of MV.
See also: UserDivideMdlsAtAllMVInteriorKnot, , UserDivideOneMdlAtAllMVInteriorKnot,

15.2.98 UserDivideOneMdlAtAllMVInteriorKnot (srf_cntr.c:1596)

```
MdlModelStruct *UserDivideOneMdlAtAllMVInteriorKnot(const MdlModelStruct *Model,
                                                    const MvarMVStruct *MV)
```

Model: To subdivide.

MV: The multivariate.

Returns: List of models, subdivided at all interior knots of MV as new models.

Description: Subdivide the given model, Mdl, embedded inside the domain of multivariate MV, at all locations Mdl crosses a knot plane in the domain of MV.
See also: UserDivideMdlsAtAllMVInteriorKnot, UserDivideMdlAtAllKnots,

15.2.99 UserDivideOneSrfAtAllMVInteriorKnot (srf_cntr.c:1109)

```
TrimSrfStruct *UserDivideOneSrfAtAllMVInteriorKnot(CagdSrfStruct *Srf,
                                                   const MvarMVStruct *MV)
```

Srf: To subdivide.

MV: The multivariate.

Returns: List of trimmed surfaces, subdivided at all interior knots of MV as trimmed surfaces.

Description: Subdivide the give surface, Srf, embedded inside the domain of multivariate MV, at all locations Srf crosses a knot plane in the domain of MV.
See also: UserDivideSrfAtInterCrvs, UserInterSrfAtAllKnots, , UserDivideSrfsAtAllTVInteriorKnot,

15.2.100 UserDivideOneSrfAtAllTVInteriorKnot (srf_cntr.c:967)

```
TrimSrfStruct *UserDivideOneSrfAtAllTVInteriorKnot(CagdSrfStruct *Srf,
                                                   const TrivTVStruct *TV)
```

Srf: To subdivide.

TV: The trivariate.

Returns: List of trimmed surfaces at all interior knots of TV as trimmed surfaces.

Description: Subdivide the give surface, Srf, embedded inside the domain of trivariate TV, at all locations Srf crosses a knot plane in the domain of TV.
See also: UserDivideSrfAtInterCrvs, UserInterSrfAtAllKnots, , UserDivideSrfsAtAllTVInteriorKnot,

15.2.101 UserDivideOneTSrfAtAllMVInteriorKnot (srf_cntr.c:1420)

```
TrimSrfStruct *UserDivideOneTSrfAtAllMVInteriorKnot(const TrimSrfStruct *TSrf,
                                                    const MvarMVStruct *MV)
```

TSrf: To subdivide.

MV: The multivariate.

Returns: List of trimmed surfaces, subdivided at all interior knots of MV.

Description: Subdivide the given trimmed surface, TSrf, embedded inside the domain of multivariate MV, at all locations TSrf crosses a knot plane in the domain of MV.
See also: UserDivideTSrfsAtAllMVInteriorKnot, UserDivideTSrfAtAllKnots,

15.2.102 UserDivideOneTVAtAllMVInteriorKnot (tv_cntr.c:122)

```
MvarComposedTrivStruct *UserDivideOneTVAtAllMVInteriorKnot(
                                                    const TrivTVStruct *TV,
                                                    const MvarMVStruct *MV)
```

TV: To subdivide.

MV: The multivariate.

Returns: List of MvarComposedTrivStruct which result from the intersection of TV with all interior knots of MV. Each element in the can be either a tensor-product trivariate or a V-model.

Description: Subdivide the give trivariate, TV, embedded inside the domain of multivariate MV, at all locations TV crosses a knot plane in the domain of MV.
See also: UserDivideOneTVAtAllMVInteriorKnot, UserDivideTVAtAllKnots,

15.2.103 UserDivideOneVMdlAtAllMVInteriorKnot (srf_cntr.c:1776)

```
VMdlVModelStruct *UserDivideOneVMdlAtAllMVInteriorKnot(
                                            const VMdlVModelStruct *VModel,
                                            const MvarMVStruct *MV)
```

VModel: To subdivide.

MV: The multivariate.

Returns: List of vmodels, subdivided at all interior knots of MV as new vmodels.

Description: Subdivide the given vmodel, VMdl, embedded inside the domain of multivariate MV, at all locations VMdl crosses a knot plane in the domain of MV.
See also: UserDivideVMdlsAtAllMVInteriorKnot, UserDivideVMdlAtAllKnots,

15.2.104 UserDivideSrfAtInterCrvs (srf_cntr.c:819)

```
TrimSrfStruct *UserDivideSrfAtInterCrvs(const CagdSrfStruct *Srf,
                                        const CagdCrvStruct *ICrvs)
```

Srf: To divide following regions defined by the intersection curves.

ICrvs: Either closed loops or curves that start/end on the boundary. Can also be NULL in which case no intersections...

Returns: List of trimmed surfaces of the different regions.

Description: Divides a give surface into regions as specified by all intersection curves, ICrvs, in the domain of Srf. ICrvs either start and end on the boundary of the domain or form closed loops.
See also: UserInterSrfAtAllKnots, UserDivideSrfAtAllTVInteriorKnot, , TrimSrfsFromContours,

15.2.105 UserDivideSrfsAtAllMVInteriorKnot (srf_cntr.c:1156)

```
TrimSrfStruct *UserDivideSrfsAtAllMVInteriorKnot(CagdSrfStruct *Srfs,
                                                 const MvarMVStruct *MV)
```

Srfs: To subdivide.

MV: The multivariate.

Returns: List of trimmed surfaces at all interior knots of TV as trimmed surfaces.

Description: Subdivide the give surfaces, Srfs, embedded inside the domain of multivariate MV, at all locations Srfs crosses a knot plane in the domain of MV.
See also: UserDivideSrfAtInterCrvs, UserInterSrfAtAllKnots , , UserDivideOneSrfAtAllTVInteriorKnot,

15.2.106 UserDivideSrfsAtAllTVInteriorKnot (srf_cntr.c:1030)

```
TrimSrfStruct *UserDivideSrfsAtAllTVInteriorKnot(CagdSrfStruct *Srfs,
                                                 const TrivTVStruct *TV)
```

Srfs: To subdivide.

TV: The trivariate.

Returns: List of trimmed surfaces at all interior knots of TV as trimmed surfaces.

Description: Subdivide the give surfaces, Srfs, embedded inside the domain of trivariate TV, at all locations Srfs crosses a knot plane in the domain of TV.
See also: UserDivideSrfAtInterCrvs, UserInterSrfAtAllKnots , , UserDivideOneSrfAtAllTVInteriorKnot,

15.2.107 UserDivideTSrfAtAllKnots (srf_cntr.c:1341)

```
TrimSrfStruct *UserDivideTSrfAtAllKnots(TrimSrfStruct *TSrf,
                                        IrtPlnType Pln,
                                        int Axis,
                                        const CagdRType *KV,
                                        int MinKV,
                                        int MaxKV,
                                        CagdRType *Param)
```

TSrf: To intersect and subdivide.

Pln: Along which to divide TSrf, or NULL if TSrf need to be divided at Axis.

Axis: Along which to divideTSrf. Ignored if Pln is not NULL.

KV: The knots to divide along.

MinKV, MaxKV: The KV range to consider.

Param: If detected an isoparametric values, set Param to it.

Returns: Subdivided trimmed surface.

Description: Intersect and subdivide the given trimmed surface, TSrf, at all locations TSrf cross a t = KV[i], MinKV <= i < MaxKV, knot plane. The plane is defined either as (Pln[0], Pln[1], Pln[2], t), or, if the parameter Pln is NULL, as Axis = t.
See also: UserDivideTSrfsAtAllMVInteriorKnot, UserDivideOneTSrfAtAllMVInteriorKnot,

15.2.108 UserDivideTSrfsAtAllMVInteriorKnot (srf_cntr.c:1474)

```
TrimSrfStruct *UserDivideTSrfsAtAllMVInteriorKnot(const TrimSrfStruct *TSrfs,
                                                  const MvarMVStruct *MV)
```

TSrfs: To subdivide.

MV: The multivariate.

Returns: List of trimmed surfaces, subdivided at all interior knots of MV.

Description: Subdivide the give trimmed surfaces, TSrfs, embedded inside the domain of multivariate MV, at all locations TSrfs crosses a knot plane in the domain of MV.
See also: UserDivideTSrfsAtAllMVInteriorKnot, , UserDivideOneTSrfAtAllMVInteriorKnot,

15.2.109 UserDivideTVAtAllKnots (tv_cntr.c:184)

```
MvarComposedTrivStruct *UserDivideTVAtAllKnots(MvarComposedTrivStruct *CTV,
                                               int Axis,
                                               const CagdRType *KV,
                                               int MinKV,
                                               int MaxKV,
                                               CagdRType *Param)
```

CTV: To intersect and subdivide.

Axis: Along which to divideVMdl. Ignored if Pln is not NULL.

KV: The knots to divide along.

MinKV, MaxKV: The KV range to consider.

Param: If detected an isoparametric values, set Param to it.

Returns: Subdivided trivariate, as a mix of trivariates and VModels.

Description: Intersect and subdivide the given trivariate, TV, at all locations TV crosses a t = KV[i], MinKV <= i < MaxKV, knot plane. The plane is defined either as (Pln[0], Pln[1], Pln[2], t), or, if the parameter Pln is NULL, as Axis = t.
See also: UserDivideVMdlsAtAllMVInteriorKnot, UserDivideOneVMdlAtAllMVInteriorKnot,

15.2.110 UserDivideTVsAtAllMVInteriorKnot (tv_cntr.c:83)

```
MvarComposedTrivStruct *UserDivideTVsAtAllMVInteriorKnot(
                                                    const TrivTVStruct *TVs,
                                                    const MvarMVStruct *MV)
```

TVs: To subdivide.

MV: The multivariate.

Returns: List of MvarComposedTrivStruct which result from the intersection of TVs with all interior knots of MV. Each element in the can be either a tensor-product trivariate or a V-model.

Description: Subdivide the give trivariates, TVs, embedded inside the domain of multivariate MV, at all locations TVs crosses a knot plane in the domain of MV.
See also: UserDivideOneTVAtAllMVInteriorKnot,

15.2.111 UserDivideVMdlAtAllKnots (srf_cntr.c:1695)

```
VMdlVModelStruct *UserDivideVMdlAtAllKnots(VMdlVModelStruct *VModel,
                                           IrtPlnType Pln,
                                           int Axis,
                                           const CagdRType *KV,
                                           int MinKV,
                                           int MaxKV,
                                           CagdRType *Param)
```

VModel: To intersect and subdivide.

Pln: Along which to divide VMdl, or NULL if VMdl need to be divided at Axis.

Axis: Along which to divideVMdl. Ignored if Pln is not NULL.

KV: The knots to divide along.

MinKV, MaxKV: The KV range to consider.

Param: If detected an isoparametric values, set Param to it.

Returns: Subdivided vmodel.

Description: Intersect and subdivide the given vmodel, VMdl, at all locations VMdl cross a t = KV[i], MinKV <= i < MaxKV, knot plane. The plane is defined either as (Pln[0], Pln[1], Pln[2], t), or, if the parameter Pln is NULL, as Axis = t.
See also: UserDivideVMdlsAtAllMVInteriorKnot, UserDivideOneVMdlAtAllMVInteriorKnot,

15.2.112 UserDivideVMdlsAtAllMVInteriorKnot (srf_cntr.c:1831)

```
VMdlVModelStruct *UserDivideVMdlsAtAllMVInteriorKnot(
                                            const VMdlVModelStruct *VModels,
                                            const MvarMVStruct *MV)
```

VModels: To subdivide.

MV: The multivariate.

Returns: List of vmodels, subdivided at all interior knots of MV, as new vmodels.

Description: Subdivide the give vmodels, VMdls, embedded inside the domain of multivariate MV, at all locations VMdls crosses a knot plane in the domain of MV.
See also: UserDivideVMdlsAtAllMVInteriorKnot, , UserDivideOneVMdlAtAllMVInteriorKnot,

15.2.113 UserFEBuildC1Mat (fntelem1.c:738)

```
UserFECElementStruct *UserFEBuildC1Mat(CagdCrvStruct *Crv1,
                                       CagdSrfStruct *Srf1,
                                       CagdCrvStruct *Crv2,
                                       CagdSrfStruct *Srf2,
                                       int IntegRes)
```

Crv1: A boundary curve of Srf1 to determine its interesction interval with Crv2. Has M control points.

Srf1: The first surface to examine for collision with Srf2.

Crv2: A boundary curve of Srf2 to determine its interesction interval with Crv1.

Srf2: The second surface to examine for collision with Srf1.

IntegRes: Resolution of integration - number of sample per interval.

Returns: Vector of size (M x M) elements of C1.

Description: Evaluates the elements of the C1 intersection matrix as the integral of the basis function's products along the intersecting boundary.
See also: UserFEBuildKMat, UserFEBuildC2Mat, UserFEEvalRHSC,

15.2.114 UserFEBuildC1Mat2 (fntelem1.c:851)

```
UserFECElementStruct *UserFEBuildC1Mat2(CagdPType *Crv1Pts,
                                        int Crv1Length,
                                        int Crv1Order,
                                        CagdPType *Srf1Pts,
                                        int Srf1ULength,
                                        int Srf1VLength,
                                        int Srf1UOrder,
                                        int Srf1VOrder,
                                        CagdPType *Crv2Pts,
                                        int Crv2Length,
                                        int Crv2Order,
                                        CagdPType *Srf2Pts,
                                        int Srf2ULength,
                                        int Srf2VLength,
                                        int Srf2UOrder,
                                        int Srf2VOrder,
                                        CagdEndConditionType EndCond,
                                        int IntegRes)
```

Crv1Pts: Crv1's control points (M).

Crv1Length: Dimensions of Crv1Pts vector.

Crv1Order: Order of Crv1.

Srf1Pts: Srf1's control points.

Srf1ULength, Srf1VLength: Dimensions of Srf1Pts vector.

Srf1UOrder, Srf1VOrder: Orders of Srf1.

Crv2Pts: Curve2's control points.

Crv2Length: Dimensions of Crv2Pts vector.

Crv2Order: Order of Curve2.

Srf2Pts: Srf2's control points.

Srf2ULength, Srf2VLength: Dimensions of Srf2Pts vector.

Srf2UOrder, Srf2VOrder: Orders of Srf2.

EndCond: Float or open, in reconstructed Crv1/2, Srf1/2.

IntegRes: Resolution of integration - number of sample per interval.

Returns: Vector of size (M x M) elements of C1.

Description: Evaluates the elements of the C1 intersection matrix as the integral of the basis function's products along the intersecting boundary.
See also: UserFEBuildKMat, UserFEBuildC2Mat, UserFEBuildC1Mat, UserFEEvalRHSC,

15.2.115 UserFEBuildC2Mat (fntelem1.c:1045)

```
UserFECElementStruct *UserFEBuildC2Mat(CagdCrvStruct *Crv1,
                                       CagdSrfStruct *Srf1,
                                       CagdCrvStruct *Crv2,
                                       CagdSrfStruct *Srf2,
                                       int IntegRes)
```

Crv1: A boundary curve of Srf1 to determine its interesction interval with Crv2. Has N control points.

Srf1: The first surface to examine for collision with Srf2.

Crv2: A boundary curve of Srf2 to determine its interesction interval with Crv1. Has M control points.

Srf2: The second surface to examine for collision with Srf1.

IntegRes: Resolution of integration - number of sample per interval.

Returns: Vector of size (N x M) elements of C2.

Description: Evaluates the elements of the C1 intersection matrix as the integral of the basis function's products along the intersecting boundary.
See also: UserFEBuildKMat, UserFEBuildC1Mat, UserFEEvalRHSC,

15.2.116 UserFEBuildC2Mat2 (fntelem1.c:1159)

```
UserFECElementStruct *UserFEBuildC2Mat2(CagdPType *Crv1Pts,
                                        int Crv1Length,
                                        int Crv1Order,
                                        CagdPType *Srf1Pts,
                                        int Srf1ULength,
                                        int Srf1VLength,
                                        int Srf1UOrder,
                                        int Srf1VOrder,
                                        CagdPType *Crv2Pts,
                                        int Crv2Length,
                                        int Crv2Order,
                                        CagdPType *Srf2Pts,
                                        int Srf2ULength,
                                        int Srf2VLength,
                                        int Srf2UOrder,
                                        int Srf2VOrder,
                                        CagdEndConditionType EndCond,
                                        int IntegRes)
```

Crv1Pts: Crv1's control points.

Crv1Length: Dimensions of Crv1Pts vector (M).

Crv1Order: Order of Crv1.

Srf1Pts: Srf1's control points.

Srf1ULength, Srf1VLength: Dimensions of Srf1Pts vector.

Srf1UOrder, Srf1VOrder: Orders of Srf1.

Crv2Pts: Curve2's control points.

Crv2Length: Dimensions of Crv2Pts vector (N).

Crv2Order: Order of Curve2.

Srf2Pts: Srf2's control points.

Srf2ULength, Srf2VLength: Dimensions of Srf2Pts vector.

Srf2UOrder, Srf2VOrder: Orders of Srf2.

EndCond: Float or open, in reconstructed Crv1/2, Srf1/2.

IntegRes: Resolution of integration - number of sample per interval.

Returns: Vector of size (N x M) elements of C2.

Description: Evaluates the elements of the C2 intersection matrix as the integral of the basis function's products along the intersecting boundary.

See also: UserFEBuildKMat, UserFEBuildC2Mat, UserFEBuildC1Mat, UserFEEvalRHSC,

15.2.117 UserFEBuildKMat (fntelem1.c:316)

```
UserFEKElementStruct *UserFEBuildKMat(CagdSrfStruct *Srf,
                                      int IntegRes,
                                      IrtRType E,
                                      IrtRType Nu,
                                      int *Size)
```

Srf: To compute its K stiffness matrix.

IntegRes: Resolution of integration - number of sample per interval. *

E: Young Module value.

Nu: Poisson coefficient, between 0.0 and 0.5.

Size: Number of degree-of-freedom (basis functions) Srf contains.

Returns: Vector of size n^2 with the (n x n) elmnts of K.

Description: Evaluates the elements of the stiffness matrix K as the integral of the basis function's derivatives of surface Srf. Let B be a linear vector of all n polynomial patches' basis functions of Srf. Let S be the matrix

```
          E      [1          Nu          0]
S = --------- [Nu          1           0] ,
    (1- Nu^2) [0           0  (1-Nu) / 2]
```

let L be the differentiation operator of

```
L = [d/du      0          d/dv]
    [0         d/dv       d/dx] ,
```

and let M be the application of L to all basis functions in B. M is a matrix of size (2n x 3). Finally compute and return the integrals of K = M S M^T , K is a matrix of size (2n x 2n). Because the integration is to be conducted in spatial space instead of parametric domain, we must normalize by the determinant of the Jacobian hence the integration is conducted numerically.

See also: UserFEBuildKMat2, UserFEBuildC1Mat, UserFEBuildC2Mat, UserFEEvalRHSC,

15.2.118 UserFEBuildKMat2 (fntelem1.c:476)

```
UserFEKElementStruct *UserFEBuildKMat2(CagdPType *Points,
                                       int ULength,
                                       int VLength,
                                       int UOrder,
                                       int VOrder,
                                       CagdEndConditionType EndCond,
                                       int IntegRes,
                                       IrtRType E,
                                       IrtRType Nu,
                                       int *Size)
```

Points: Current surface control points.

ULength, VLength: Dimensions of Points vectors.

UOrder, VOrder: Surface Order in U and V.

EndCond: End conditions - open, float, etc.

IntegRes: Resolution of integration - number of sample per interval. *

E: Young Module value.

Nu: Poisson coefficient, between 0.0 and 0.5.

Size: Number of degree-of-freedom (basis functions) Srf contains.

Returns: Vector of size n^2 with the (n x n) elmnts of K.

Description: Evaluates the elements of the stiffness matrix K as the integral of the basis function's derivatives of surface Srf. Let B be a linear vector of all n polynomial patches' basis functions of Srf. Let S be the matrix

```
          E      [1          Nu          0]
S = --------- [Nu          1           0] ,
    (1- Nu^2) [0          0  (1-Nu) / 2]
```

let L be the differentiation operator of

```
L = [d/du      0         d/dv]
    [0         d/dv      d/dx] ,
```

and let M be the application of L to all basis functions in B. M is a matrix of size (2n x 3). Finally compute and return the integrals of K = M S M^T , K is a matrix of size (2n x 2n). Because the integration is to be conducted in spatial space instead of parametric domain, we must normalize by the determinant of the Jacobian hence the integration is conducted numerically.

See also: UserFEBuildKMat,

15.2.119 UserFEEvalRHSC (fntelem1.c:1239)

```
IrtRType UserFEEvalRHSC(UserFECElementStruct *C,
                        CagdCrvStruct *Crv1,
                        CagdCrvStruct *Crv2)
```

C: The current row of C1 matrix of M elements.

Crv1: A boundary curve of Srf1 to determine its interesction interval with Crv2. Has M control points.

Crv2: A boundary curve of Srf2 to determine its interesction interval with Crv1. Has N control points.

Returns: Current value of constraint. Should be zero if satisified.

Description: Evaluates the right hand side (RHS) of the i'th C constraint - of the i'th basis function.
See also: UserFEBuildKMat, UserFEBuildC2Mat, UserFEBuildC1Mat,

15.2.120 UserFEGetInterInterval (fntelem1.c:567)

```
UserFEInterIntervalStruct *UserFEGetInterInterval(CagdCrvStruct *Crv1,
                                                  CagdSrfStruct *Srf1,
                                                  CagdCrvStruct *Crv2,
                                                  CagdSrfStruct *Srf2)
```

Crv1: A boundary curve of Srf1 to determine its intersection interval with Crv2.

Srf1: The first surface to examine for collision with Srf2.

Crv2: A boundary curve of Srf2 to determine its intersection interval with Crv1.

Srf2: The second surface to examine for collision with Srf1.

Returns: A list of elements, each holding one continuous interval of intersection, specifying the proper intervals in both Crv1 and Crv2.

Description: Derives the interval(s) where Crv1 and Crv2 intersect, two boundary curves of Srf1 and Srf2.
See also:

15.2.121 UserFEPointInsideSrf (fntelem1.c:525)

```
CagdBType UserFEPointInsideSrf(CagdSrfStruct *Srf, CagdPType Pt)
```

Srf: Surface to test the inclusion of Pt in, in R^2.

Pt: 2D point to test if inside Srf.

Returns: TRUE if inside, FALSE otherwise.

Description: Returns TRUE if given point Pt is inside surface Srf, FALSE otherwise.
See also:

15.2.122 UserFontBspCrv2Poly (font3d.c:361)

```
IPPolygonStruct *UserFontBspCrv2Poly(CagdCrvStruct *BspCrv,
                                     IrtRType Tolerance)
```

BspCrv: Pointer to the Curve.

Tolerance: The tolerance used calculating the polygon.

Returns: Pointer to the new Polygon.

Description: Converting a closed B-spline to a Polygon.

15.2.123 UserFontBspList2Plgns (font3d.c:472)

```
IPObjectStruct *UserFontBspList2Plgns(IPObjectStruct *BspListObj,
                                      IrtRType Tol,
                                      const char *Name)
```

BspListObj: A list object of Bspline curves.

Tol: Tolerance of approximation.

Name: Base name to use for constructed objects. Can be NULL.

Returns: The created polygons.

Description: Converts a list object of Bspline merged closed curves (outline of the font) to polygons that fill the text.
See also: UserFontBspList2TrimSrfs,

15.2.124 UserFontBspList2Solids (font3d.c:795)

```
IPObjectStruct *UserFontBspList2Solids(IPObjectStruct *BspListObj,
                                       UserFont3DEdgeType ExtStyle,
                                       IrtRType ExtOffset,
                                       IrtRType ExtHeight,
                                       IrtRType Tol,
                                       CagdBType GenTrimSrfs,
                                       const char *Name)
```

BspListObj: A list object of Bspline curves.

ExtStyle: Type of 3D solid geometry: chamfered corners, rounded, etc.

ExtOffset: Size of the chamfer/rounding.

ExtHeight: Height of the 3D solid constructed text.

Tol: Tolerance of approximation.

GenTrimSrfs: TRUE to generate trimmed surfaces. FALSE for polygons.

Name: Base name to use for constructed objects. Can be NULL.

Returns: The created polygons.

Description: Converts a list object of Bspline merged closed curves (outline of the font) to polygons that fill the text.

15.2.125 UserFontBspList2SweptTubes (font3d.c:1064)

```
IPObjectStruct *UserFontBspList2SweptTubes(const IPObjectStruct *BspListObj,
                                           UserFont3DEdgeType CornerStyle,
                                           const CagdCrvStruct *CrossSection,
                                           IrtRType Tol,
                                           const char *Name)
```

BspListObj: A list object of B-spline curves.

CornerStyle: Type of C^1 discontinuous corners: chamfered, rounded, etc.

CrossSection: Cross section curve to sweep.

Tol: Tolerance of approximation.

Name: Base name to use for constructed objects. Can be NULL.

Returns: The sweep surfaces.

Description: Converts a list object of B-spline merged closed curves (outline of the font) to circular sweep surfaces around them.

15.2.126 UserFontBspList2TrimSrfs (font3d.c:550)

```
IPObjectStruct *UserFontBspList2TrimSrfs(IPObjectStruct *BspListObj,
                                         IrtRType Tol,
                                         const char *Name)
```

BspListObj: A list object of Bspline curves.

Tol: Tolerance of approximation.

Name: Base name to use for constructed objects. Can be NULL.

Returns: The created trimmed surfaces.

Description: Converts a list object of Bspline merged closed curves (outline of the font) to trimmed surfaces that fill the text.
See also: UserFontBspList2Plgns,

15.2.127 UserFontBzrList2BspList (font3d.c:401)

```
CagdCrvStruct *UserFontBzrList2BspList(IPObjectStruct *BzrListObj,
                                       IrtBType *HaveHoles)
```

BzrListObj: A list object of Bezier curves.

HaveHoles: Will be set to TRUE if holes are detected (examining the orientation of the loops).

Returns: A list of Bspline loop curves.

Description: Converts a list object of Bezier curves into a list of Bspline curves by chaining adjacent Bezier curve segments into closed Bsplines loops.

15.2.128 UserFontConvertFontToBezier (wfnt2bzr.c:150)

```
IPObjectStruct *UserFontConvertFontToBezier(const UserFontText Text,
                                            const UserFontName FontName,
                                            UserFontStyleType FontStyle,
                                            IrtRType SpaceWidth,
                                            int MergeToBsp,
                                            const char *RootObjName)
```

Text: Text string.

FontName: Font name.

FontStyle: Font style.

SpaceWidth: Space width. Space, in points, added to individual chars.

MergeToBsp: TRUE to merge Bezier curves into larger B-spline curves, FALSE to leave the original (font) Bezier curves.

RootObjName: Name of root object.

Returns: Extracted text object or NULL in case of failure.

Description: Extracts Bezier curves from windows' fonts.

15.2.129 UserFontConvertTextToGeom (font3d.c:103)

```
int UserFontConvertTextToGeom(const UserFontText Text,
                              const UserFontName FontName,
                              UserFontStyleType FontStyle,
                              IrtRType FontSize,
                              IrtRType TextSpace,
                              UserFont3DEdgeType Text3DEdgeType,
                              const IrtRType Text3DSetup[2],
                              IrtRType Tolerance,
                              UserFontGeomOutputType OutputType,
                              IrtBType CompactOutput,
                              const char *PlacedTextBaseName,
                              IPObjectStruct **PlacedTextGeom,
                              char **ErrorStr)
```

Text: The string text to convert to geometry.

FontName: The font to use for conversion.

FontStyle: The font style (italic, bold, etc.)

FontSize: Created text scaling relative factor.

TextSpace: In-word relative spacing.

Text3DEdgeType: For 3D text geometry created edges/corners (i.e. rounded or chamfered.).

Text3DSetup: For 3D text, a vector of (Chamfer offset/Sweep radius size, 3D extruded vertical distance).

Tolerance: For 2D filled polygons and 3D solid text geometry.

OutputType: Selects the type of geometry to create: 0. Outline Bezier curves as in the font data. 1. Outline B-spline curves (merging Bezier curves in 1). 2. Solid 2D polygons, for the outline geometry (polygons). 3. Both Solid 2D and B-spline outline (2+3 above). 4. Full 3D text (polygons). 5. Solid 2D polygons, for the outline geometry ((trimmed) surfaces). 6. Full 3D text ((trimmed) surfaces).

CompactOutput: If TRUE, merged trimming surfaces or polygons into single objects.

PlacedTextBaseName: Base name to use in placed text. Can be NULL.

PlacedTextGeom: Output created geometry will be returned here.

ErrorStr: Will be set with the error message if was one.

Returns: TRUE if successful, FALSE if conversion failed.

Description: Converts Text to geometry. Note this function will behave differently on different platforms (i.e. Windows vs. Unix.

See also: UserFontConvertFontToBezier, UserFontFTStringOutline2BezierCurves, UserFontBspCrv2Poly, UserFontBspList2Solids,

15.2.130 UserFontFTStringOutline2BezierCurves (ffnt2bzr.c:284)

```
IPObjectStruct *UserFontFTStringOutline2BezierCurves(
                                                const UserFontText Text,
                                                const UserFontName FontName,
                                                IrtRType Spacing,
                                                int MergeToBsp,
                                                const char *RootObjName,
                                                const char **ErrStr)
```

Text: String to convert to Bezier outline.

FontName: Font to read the outline from.

Spacing: To apply between the characters.

MergeToBsp: TRUE to merge the Bezier curves into larger B-spline curves, FALSE to leave the original (font) Bezier curves.

RootObjName: Name of root object.

ErrStr: A string describing the error, if any.

Returns: List of curves in IRIT format representing the string.

Description: Given a string and a font name - convert to a set of curves in IRIT format.
See also: Freetype font library, http://www.freetype.org.,

15.2.131 UserFontLayoutOverShape (fontlout.c:149)

```
int UserFontLayoutOverShape(const UserFontText Text,
                            const UserFontName FontName,
                            UserFontStyleType FontStyle,
                            IrtRType FontSize,
                            const IrtRType FontSpace[3],
                            IrtRType Tolerance,
                            UserFont3DEdgeType Text3DEdge,
                            const IrtRType Text3DSetup[2],
                            UserFontAlignmentType Alignment,
                            const IPPolygonStruct *BoundingPoly,
                            UserFontGeomOutputType OutputType,
                            IPObjectStruct **PlacedTextGeom,
                            char **ErrorStr)
```

Text: The text to layout. A string.

FontName: The font name to use. I.e. "Times new Roman".

FontStyle: Font style to use. I.e. italics.

FontSize: The size of the constructed text.

FontSpace: (WordWidth, SpaceWidth, LineHeight) spacing, specified in text font's point units.

Tolerance: For 2D filled polygons and 3D solid text geometry.

Text3DEdge: For 3D text geometry controls edges (i.e. chamferred.).

Text3DSetup: For 3D text, a vector of (Chamfer offset size, 3D extruded vertical distance).

Alignment: Text alignment, left, centered, etc.

BoundingPoly: A closed region to place the text inside.

OutputType: Selects the type of geometry to create: 1. Outline Bezier curves as in the font data. 2. Outline B-spline curves (merging Bezier curves in 1). 3. Filling 2D polygons, for the outline geometry. 4. Full 3D text.

PlacedTextGeom: The result synthesized placed (geometry of the) text. Actually only the base line of the text is guaranteed to be in. NULL if error.

ErrorStr: Will be updated with an error description if was one, or with NULL if all went well.

Returns: TRUE if all text fitted in the BoundingPoly, FALSE otherwise. FALSE might also be returned if we failed to generate the text, in which case ErrorStr will hold some error description.

Description: Layout the given text over the given bounding region. Constructed Text will be controlled by FontName, FontStyle, FontSize, and FontSpace and will be aligned to follow TextAlignment. All construction is conducted over the XY plane, in 2D.
See also: UserFontLayoutOverShape2,

15.2.132 UserFontLayoutOverShape2 (fontlout.c:78)

```
int UserFontLayoutOverShape2(const UserFontText Text,
                             const UserFontName FontName,
                             UserFontStyleType FontStyle,
                             IrtRType FontSize,
                             const IrtRType FontSpace[3],
                             IrtRType Tolerance,
                             UserFont3DEdgeType Text3DEdge,
                             const IrtRType Text3DSetup[2],
                             UserFontAlignmentType Alignment,
                             const CagdCrvStruct *BoundingCrv,
                             UserFontGeomOutputType OutputType,
                             IPObjectStruct **PlacedTextGeom,
                             char **ErrorStr)
```

Text: The text to layout. A string.

FontName: The font name to use. I.e. "Times new Roman".

FontStyle: Font style to use. I.e. italics.

FontSize: The size of the constructed text.

FontSpace: (WordWidth, SpaceWidth, LineHeight) spacing, specified in text font's point units.

Tolerance: For 2D filled polygons and 3D solid text geometry.

Text3DEdge: For 3D text geometry controls edges (i.e. chamferred.).

Text3DSetup: For 3D text, a vector of (Chamfer offset size, 3D extruded vertical distance).

Alignment: Text alignment, left, centered, etc.

BoundingCrv: A closed curve to place the text inside.

OutputType: Output should be original Bezier curves, merged B-spline curves, filled planar polys, 3D polys, etc.

PlacedTextGeom: The placed (geometry of the) text in. Actually only the base line of the text is guaranteed to be in. NULL if error.

ErrorStr: Will be updated with an error description if was one, or with NULL if all went well.

Returns: TRUE if all text fitted in the BoundingPoly, FALSE otherwise. *

Description: Layout the given text over the given bounding region. Constructed Text will be controlled by FontName, FontStyle, FontSize, and FontSpace and will be aligned to follow TextAlignment. All construction is conducted over the XY plane, in 2D.
See also: UserFontLayoutOverShape,

15.2.133 UserFontLayoutOverShapeFree (fontlout.c:195)

```
void UserFontLayoutOverShapeFree(struct UserFontWordLayoutStruct *Words)
```

Words: Linked list of words to free.

Returns: void

Description: Free the allocated words and their geometry.
See also: UserFontLayoutOverShape, UserFontLayoutOverShape2, , UserFontLayoutOverShapeGenWords, UserFontLayoutOverShapePlaceWords.,

15.2.134 UserFontLayoutOverShapeGenWords (fontlout.c:270)

```
UserFontWordLayoutStruct *UserFontLayoutOverShapeGenWords(
                                    const UserFontText Text,
                                    const UserFontName FontName,
                                    UserFontStyleType FontStyle,
                                    IrtRType FontSize,
                                    const IrtRType FontSpace[3],
                                    IrtRType Tolerance,
```

```
                                        UserFont3DEdgeType Text3DEdge,
                                        const IrtRType Text3DSetup[2],
                                        UserFontAlignmentType Alignment,
                                        const IPPolygonStruct *BoundingPoly,
                                        UserFontGeomOutputType OutputType,
                                        IrtBType CompactOutput,
                                        const char *OutputBaseName,
                                        UserFontDimInfoStruct *FontDims,
                                        char **ErrorStr)
```

Text: The text to layout. A string.

FontName: The font name to use. I.e. "Times new Roman".

FontStyle: Font style to use. I.e. italics.

FontSize: The size of the constructed text.

FontSpace: (WordWidth, SpaceWidth, LineHeight) spacing, specified in text font's point units.

Tolerance: For 2D filled polygons and 3D solid text geometry.

Text3DEdge: For 3D text geometry controls edges (i.e. chamferred.).

Text3DSetup: For 3D text, a vector of (Chamfer offset size, 3D extruded vertical distance).

Alignment: Text alignment, left, centered, etc.

BoundingPoly: A closed region to place the text inside.

OutputType: Selects the type of geometry to create: 0. Outline Bezier curves as in the font data. 1. Outline B-spline curves (merging Bezier curves in 1). 2. Solid 2D polygons, for the outline geometry (polygons). 3. Both Solid 2D and B-spline outline (2+3 above). 4. Full 3D text (polygons). 5. Solid 2D polygons, for the outline geometry ((trimmed) surfaces). 6. Full 3D text ((trimmed) surfaces).

CompactOutput: If TRUE, merged trimming surfaces or polygons into single objects.

OutputBaseName: Base name to use in output geometry.

FontDims: Structure to be updated with font dimensions if all went well.

ErrorStr: Will be updated with an error description if was one, or with NULL if all went well.

Returns: Generated words.

Description: Constructed words as a preparation to layout the text over the given bounding region. All construction is conducted over the XY plane, in 2D.
See also: UserFontLayoutOverShape, UserFontLayoutOverShape2, , UserFontLayoutOverShapeFree, UserFontLayoutOverShapePlaceWords.,

15.2.135 UserFontLayoutOverShapePlaceWords (fontlout.c:408)

```
int UserFontLayoutOverShapePlaceWords(UserFontWordLayoutStruct *Words,
                                      IrtRType FontSize,
                                      const IrtRType FontSpace[3],
                                      UserFontAlignmentType Alignment,
                                      const UserFontDimInfoStruct *FontDims,
                                      const IPPolygonStruct *BoundingPoly,
                                      IPObjectStruct **PlacedTextGeom)
```

Words: The ordered list of words to place into bounding shape

FontSize: The size of the constructed text.

FontSpace: (WordWidth, SpaceWidth, LineHeight) spacing, specified in text font's point units.

Alignment: Align each line of the placed words to the left, centered, etc.

FontDims: Dimensions information on this font.

BoundingPoly: The region to place the text in (actually only base line will be in).

PlacedTextGeom: Where to place the created geometry. Organized as an object per character, grouped in list objects as words, all grouped in this list object. word object will have line number, "TextLineCnt", and word count in line, "TextWordCnt", attributes.

Returns: TRUE if all text fitted in the BoundingPoly, FALSE otherwise.

Description: Translate the given (geometry of) words so they fit into the given bounding shape BoundingPoly.
See also: UserFontLayoutOverShape, UserFontLayoutOverShape2, , UserFontLayoutOverShapeFree, UserFontLayoutOverShapeGenWords.,

15.2.136 UserGr2DSwpArrangeTeeth (gear2d_sweep.c:1630)

```
IPObjectStruct *UserGr2DSwpArrangeTeeth(const CagdCrvStruct *Centrode,
                                        const CagdCrvStruct *Tooth,
                                        CagdRType ToothLen,
                                        int NTeeth)
```

Centrode: Curve along which to arrange teeth.

Tooth: Copies of which to place.

ToothLen: Arc-length distance between consecutive teeth.

NTeeth: Number of copies of tooth to place.

Returns: List of curves representing the teeth arrangement.

Description: Arrange prescribed number of copies of given tooth along the given curve after applying the required transformation to the tooth.

15.2.137 UserGr2DSwpComputeCentrode (gear2d_sweep.c:1750)

```
CagdCrvStruct *UserGr2DSwpComputeCentrode(const MvarMVStruct *Rot[2],
                                          const MvarMVStruct *Trans,
                                          const CagdSrfStruct *Plane,
                                          CagdRType SolverStepSize)
```

Rot: Rotational part of rigid motion.

Trans: Translational part of rigid motion.

Plane: The bounded plane in which to compute the centrode.

SolverStepSize: Step size to use in solver.

Returns: The centrode as a curve.

Description: Compute the centrode of the gear as the zero set of the velocity of the given relative motion.

15.2.138 UserGr2DSwpConjugateShape (gear2d_sweep.c:1847)

```
IPObjectStruct *UserGr2DSwpConjugateShape(const IPObjectStruct *CrvSegList,
                                          const MvarMVStruct *Rot[2],
                                          const MvarMVStruct *Trans,
                                          CagdRType SolverStepSize)
```

CrvSegList: Input planar shape represented by a closed loop of of curve segments. Curve segments may meet with G1- discontinuity at end-points.

Rot: Rotational part of rigid motion.

Trans: Translational part of rigid motion.

SolverStepSize: Step size to use in solver.

Returns: The conjugate shape as a closed loop of curve segments.

Description: Compute the conjugate of a given planar shape along given one parameter family of rigid motions. This is done by computing the swept volume of of the input shape, subtracting it from the plane and retaining only the bounded portion of the plane.

15.2.139 UserGr2DSwpGenInverseMotion (gear2d_sweep_motion.c:164)

```
void UserGr2DSwpGenInverseMotion(const MvarMVStruct *Rot[2],
                                 const MvarMVStruct *Trans,
                                 MvarMVStruct *InvRot[2],
                                 MvarMVStruct **InvTrans)
```

Rot: Rotational part of rigid motion.

Trans: Translational part of rigid motion.

InvRot: Rotational part of inverse rigid motion.

InvTrans: Translational part of inverse rigid motion.

Returns: void

Description: Compuates the inverse of the given one parameter family of rigid motions.

15.2.140 UserGr2DSwpGenMotionOblong (gear2d_sweep_motion.c:357)

```
void UserGr2DSwpGenMotionOblong(CagdRType ToothLen,
                                int NumTeethSrc,
                                int NumTeethCirc,
                                int NumTeethLin,
                                CagdBType InverseMotion,
                                MvarMVStruct *Rot[2],
                                MvarMVStruct **Trans)
```

ToothLen: Arc-length of the base of one tooth.

NumTeethSrc: Number of teeth in the circular gear.

NumTeethCirc: Number of teeth in the circular part of the oblong gear.

NumTeethLin: Number of teeth in the linear part of the oblong gear.

InverseMotion: TRUE if inverse of the motion is required.

Rot: To return rotational part of rigid motion in.

Trans: To return translational part of rigid motion in.

Returns: void

Description: Generate a one parameter family of rigid motions involving rolling of a circular gear along an oblong shaped curve without slipping.

15.2.141 UserGr2DSwpGenNonUniformMotion (gear2d_sweep_motion.c:574)

```
void UserGr2DSwpGenNonUniformMotion(MvarMVStruct *Rot[2],
                                    MvarMVStruct **Trans,
                                    CagdRType GearDist,
                                    const CagdCrvStruct *RelVeloFn)
```

Rot: Rotational part of rigid motion.

Trans: Translational part of rigid motion.

GearDist: Radius of circle along which translation occurs.

RelVeloFn: Scalar curve giving the non-uniform circular displacement profile. It must map interval [0,1] to [0,2\pi].

Returns: void

Description: Generate a one parameter family of rigid motions involving rotation by angle 2\pi and translation along a circular path of given radius at non-uniform angular velocity, prescribed by function supplied as an argument.

15.2.142 UserGr2DSwpGenUniformMotion (gear2d_sweep_motion.c:240)

```
void UserGr2DSwpGenUniformMotion(MvarMVStruct *Rot[2],
                                 MvarMVStruct **Trans,
                                 CagdRType GearDist)
```

Rot: Rotational part of rigid motion.

Trans: Translational part of rigid motion.

GearDist: Radius of circle along which translation occurs.

Returns: void

Description: Generate a one parameter family of rigid motions involving rotation by angle 2\pi and translation along a circular path of given radius at uniform angular velocity.

15.2.143 UserGr2DSwpMain (gear2d_sweep.c:2017)

```
void UserGr2DSwpMain(UserGr2DSwpParamStruct *Params)
```

Params: Parameters for call. FunctionSelect member of this structure dictates the type of functionality to be invoked. Rest of the members are specific to the functionality as described below along with return value. USER_GEAR2D_SWEEP_CONJUGATE - Input gear, Rotation, Translation, SolverStepSize. Returns the conjugate gear. USER_GEAR2D_SWEEP_ARRANGE_TEETH - Overall gear shape, tooth shape, Arclength of base of tooth, number of teeth. Returns the gear with such a teeth arrangement. USER_GEAR2D_SWEEP_CENTRODE - Rotation, Translation, Plane of computation, SolverStepSize. Returns the centrode. USER_GEAR2D_SWEEP_UNIFORM_MOTION - Distance between centers of the two gears. Returns rotation and translation. USER_GEAR2D_SWEEP_NON_UNIFORM_MOTION - Distance between center of the two gears, scalar curve specifying non-uniform circular displacement profile. It must map interval [0,1] to [0,2\pi]. Returns rotation and translation. USER_GEAR2D_SWEEP_OBLONG_MOTION - Arclength of tooth-base, number of teeth in source gear, number of teeth in circular part of target gear, number of teeth in linear part of target gear, flag to specify if inverse motion is desired. Returns rotation and translation. USER_GEAR2D_SWEEP_INVERSE_MOTION - Rotation, translation. Returns rotation and translation for inverse motion.

Returns: void

Description: Main function for invoking required functionalities for the 2D gear design.

15.2.144 UserGr2DSwpMatVecMult (gear2d_sweep_motion.c:188)

```
MvarMVStruct *UserGr2DSwpMatVecMult(const MvarMVStruct *Mat[2],
                                    const MvarMVStruct *Vec)
```

Mat: Matrix represented as multivariates.

Vec: Vector represented as multivariate.

Returns: The result of the multiplication.

Description: Multiply a matrix and a vector, both given as multivariates.

15.2.145 UserGr2DSwpReadCrvFromFile (gear2d_sweep.c:240)

```
IPObjectStruct *UserGr2DSwpReadCrvFromFile(char *Filename)
```

Filename: Curve to read.

Returns: The planar shape read.

Description: Reads a planar shape from input file, in the form of a closed loop of curve segments.

15.2.146 UserHCEditCopy (hrmt_crv.c:421)

```
VoidPtr UserHCEditCopy(VoidPtr HC)
```

HC: A handle on the edited Hermite curve to be duplicated.

Returns: Duplicated HC.

Description: Duplicates the data structures for editing planar piecewise cubic Hermite curves.
See also: UserHCEditInit,

15.2.147 UserHCEditCreateAppendCtlpt (hrmt_crv.c:469)

```
int UserHCEditCreateAppendCtlpt(VoidPtr HC,
                                CagdRType x,
                                CagdRType y,
                                int MouseMode)
```

HC: A handle on edited Hermite curve to append control point to.

x, y: The coordinate of the control point.

MouseMode: 0 for mouse down, 1 for mouse move, 2 for mouse up (done).

Returns: TRUE if successful, FALSE otherwise.

Description: Append control point at the end of the current curve.
See also: UserHCEditInit,

15.2.148 UserHCEditCreateDone (hrmt_crv.c:567)

```
int UserHCEditCreateDone(VoidPtr HC, CagdRType LastX, CagdRType LastY)
```

HC: A handle on the edited Hermite curve to insert control point to.

LastX, LastY: Last control point of curve, during creation stage.

Returns: TRUE if successful, FALSE otherwise.

Description: Initializes the interface for editing planar piecewie cubic Hermite curves.
See also: UserHCEditInit,

15.2.149 UserHCEditDelete (hrmt_crv.c:382)

```
void UserHCEditDelete(VoidPtr HC)
```

HC: A handle on the edited Hermite curve to be deleted.

Returns: void

Description: Delete the data structures for editing planar piecewise cubic Hermite curves.
See also: UserHCEditInit,

15.2.150 UserHCEditDeleteCtlpt (hrmt_crv.c:676)

```
int UserHCEditDeleteCtlpt(VoidPtr HC, CagdRType x, CagdRType y)
```

HC: A handle on edited Hermite curve to delete control point from.

x, y: The coordinate of the point.

Returns: TRUE if successful, FALSE otherwise.

Description: Delete the control point at the given location on the current curve.
See also: UserHCEditInit,

15.2.151 UserHCEditDrawCtlpts (hrmt_crv.c:1612)

```
int UserHCEditDrawCtlpts(VoidPtr HC, int DrawTans)
```

HC: A handle on edited Hermite curve to update its Index segment.

DrawTans: Do we want to draw tangents as well?

Returns: TRUE if successful, FALSE otherwise.

Description: Draw the control points of the given HC, using the defined drawing function.
See also: UserHCEditInit,

15.2.152 UserHCEditEvalDefTans (hrmt_crv.c:1845)

```
int UserHCEditEvalDefTans(VoidPtr HC, int Index)
```

HC: A handle on edited Hermite curve to update its tangents.

Index: Of the curve whose tangent is to be updated (with Index + 1).

Returns: TRUE if successful, FALSE otherwise.

Description: Routine to update the tangents of the given HC curve to a reasonable magnitude and direction. Useful when tangent handles are not provided to the end user.
See also: UserHCEditInit,

15.2.153 UserHCEditFromCurve (hrmt_crv.c:132)

```
VoidPtr UserHCEditFromCurve(const CagdCrvStruct *Crv, CagdRType Tol)
```

Crv: Regular curve to convert to an Hermite curve.

Tol: Tolerance of approximation if the input curve is higher degree than cubic.

Returns: A Handle on the created HC to be edited.

Description: Create a Hermite curve from the given regular curve.
See also: SymbApproxCrvAsBzrCubics, UserHCEditInit,

15.2.154 UserHCEditGetCrvRepresentation (hrmt_crv.c:1459)

```
CagdCrvStruct *UserHCEditGetCrvRepresentation(VoidPtr HC, int ArcLen)
```

HC: A handle on edited Hermite curve to convert to Bspline.

ArcLen: TRUE to approximately parametrize the geometry to follow arc length parametrization, FALSE to apply uniform parametrization.

Returns: A Bspline curve representing the given HC.

Description: Constructs one (Bspline) curve representation for the given HC.
See also: UserHCEditInit,

15.2.155 UserHCEditGetCtlPtCont (hrmt_crv.c:283)

```
CagdBType UserHCEditGetCtlPtCont(VoidPtr HC, int Index)
```

HC: A handle on edited Hermite curve to update its Index segment.

Index: Of control point to get its C^1 continuity state.

Returns: TRUE for C^1 continuity, FALSE for C^0.

Description: Gets the continuity of the Index control point in the given curve.
See also: UserHCEditInit,

15.2.156 UserHCEditGetCtlPtTan (hrmt_crv.c:1565)

```
int UserHCEditGetCtlPtTan(VoidPtr HC, int Index, CagdPType Pos, CagdPType Tan)
```

HC: A handle on edited Hermite curve to feicth one of its HC control points and tangent.

Index: Of control point and tangent to fetch. If Index is negative the backward tangent is fecthed.

Pos: Position to fetch.

Tan: Tangent to fetch.

Returns: TRUE if successful, FALSE otherwise.

Description: Returns the position and tangent of the Index control point of the HC curve. First point is index zero. Tan is always pointing along the forward moving direction of the curve.
See also: UserHCEditInit,

15.2.157 UserHCEditGetNumCtlPt (hrmt_crv.c:1534)

```
int UserHCEditGetNumCtlPt(VoidPtr HC)
```

HC: A handle on edited Hermite curve to get its size.

Returns: Number of HC control points.

Description: Returns number of HC control points this HC curve has. These points are end points of Hermite intervals. For example, a curve with three Hermite intervals will have 4 HC control points.
See also: UserHCEditInit,

15.2.158 UserHCEditInit (hrmt_crv.c:88)

```
VoidPtr UserHCEditInit(CagdRType StartX, CagdRType StartY, CagdBType Periodic)
```

StartX, StartY: Starting control point of curve.

Periodic: TRUE if curve is to be closed.

Returns: A Handle on the created HC to be edited.

Description: Initializes the interface for editing planar piecewise cubic Hermite curves.
See also: UserHCEditFromCurve, UserHCEditIsPeriodic, UserHCEditSetPeriodic, , UserHCEditSetCtlPtCont, UserHCEditSetDrawCtlptFunc, UserHCEditDelete, , UserHCEditTranslate, UserHCEditCreateAppendCtlpt, UserHCEditCreateDone, , UserHCEditInsertCtlpt, UserHCEditDeleteCtlpt, UserHCEditMoveCtl, , UserHCEditMoveCtlPt, UserHCEditMoveCtlTan, UserHCEditIsNearCrv, , UserHCEditIsNearCtlPt, UserHCEditIsNearCtlTan, , UserHCEditGetCrvRepresentation, UserHCEditDrawCtlpts, , UserHCEditTransform, UserHCEditRelativeTranslate, UserHCEditEvalDefTans,

15.2.159 UserHCEditInsertCtlpt (hrmt_crv.c:599)

```
int UserHCEditInsertCtlpt(VoidPtr HC, CagdRType x, CagdRType y, CagdRType t)
```

HC: A handle on the edited Hermite curve to insert control point to.

x, y: The coordinate of the point.

t: The parameter location to insert a new control point.

Returns: TRUE if successful, FALSE otherwise.

Description: Insert new control point at the given parameter value of the curve.
See also: UserHCEditInit,

15.2.160 UserHCEditIsNearCrv (hrmt_crv.c:1188)

```
int UserHCEditIsNearCrv(VoidPtr HC,
                        CagdRType x,
                        CagdRType y,
                        CagdRType *t,
                        CagdRType Eps,
                        int NormalizeZeroOne)
```

HC: A handle on edited hermite curve to see if we are close.

x, y: The coordinate of the given location.

t: The closest parameter location detected near the mouse.

Eps: Distance to consider it near.

NormalizeZeroOne: If TRUE detected parameter is normalized into zero to one for beginning to end. Otherwise, if FALSE, the returned parameter is 'x.y' where x is the Hermite segment number and y is the relative location within the segment.

Returns: TRUE if given location is indeed near curve, FALSE otherwise.

Description: Routine to examine if given location is near HC to within Eps.
See also: UserHCEditInit,

15.2.161 UserHCEditIsNearCtlPt (hrmt_crv.c:1295)

```
int UserHCEditIsNearCtlPt(VoidPtr HC,
                          CagdRType *x,
                          CagdRType *y,
                          int *Index,
                          int *UniqueID,
                          CagdRType Eps)
```

HC: A handle on edited Hermite curve to see if we are close.

x, y: The coordinate of the given location. This coordinated will be updated with the precise control point location, if indeed near.

Index: Of closest control point, if found any below distance Eps, in the curve, first points is indexed 0.

UniqueID: The unique integer ID that is assigned to this control point.

Eps: Distance to consider it near.

Returns: TRUE if given location is indeed near a control point, FALSE otherwise.

Description: Routine to examine if given location is near a control point of HC to within Eps.
See also: UserHCEditInit,

15.2.162 UserHCEditIsNearCtlTan (hrmt_crv.c:1381)

```
int UserHCEditIsNearCtlTan(VoidPtr HC,
                           CagdRType *x,
                           CagdRType *y,
                           int *Index,
                           int *UniqueID,
                           CagdBType *Forward,
                           CagdRType Eps)
```

HC: A handle on edited Hermite curve to see if we are close.

x, y: The coordinate of the given location. This coordinated will be updated with the precise control tangent location, if indeed near.

Index: Of closest control tangent, if found any below distance Eps.

UniqueID: The unique integer ID that is assigned to this control point.

Forward: TRUE for forward tangent, FALSE for backward tangent.

Eps: Distance to consider it near.

Returns: TRUE if given location is indeed near a control tangent, FALSE otherwise.

Description: Routine to examine if given location is near a control tangent of HC, to within Eps.
See also: UserHCEditInit,

15.2.163 UserHCEditIsPeriodic (hrmt_crv.c:195)

```
int UserHCEditIsPeriodic(VoidPtr HC)
```

HC: A handle on the Hermite curve to be examined.

Returns: TRUE if given HC curve is periodic, FALSE otherwise.

Description: Gets the periodic conditions of the given HC curve.
See also: UserHCEditInit,

15.2.164 UserHCEditMatTrans (hrmt_crv.c:1693)

```
int UserHCEditMatTrans(VoidPtr HC, IrtHmgnMatType Mat)
```

HC: A handle on edited Hermite curve to transform, in place.

Mat: Transformation matrxi to apply to HC.

Returns: TRUE if successful, FALSE otherwise.

Description: Applies given transformion, in place, to given curve HC as specified by Mat. Only the XY coordinates are processed. Curve HC is first translated by Dir and then scaled by Scale.
See also: CagdCrvTransform, UserHCEditInit, UserHCEditTransform,

15.2.165 UserHCEditMoveCtl (hrmt_crv.c:876)

```
int UserHCEditMoveCtl(VoidPtr HC,
                      CagdRType OldX,
                      CagdRType OldY,
                      CagdRType NewX,
                      CagdRType NewY,
                      int MouseMode,
                      CagdRType *MinDist)
```

HC: A handle on edited Hermite curve to move control pt/tan in.

OldX, OldY: The original coordinate of the control pt/tan.

NewX, NewY: The original coordinate of the control pt/tan.

MouseMode: 0 for mouse down, 1 for mouse move, 2 for mouse up (done), 3 for reset/clear state.

MinDist: If not NULL and MouseMode is 0, returns the minimal distance found.

Returns: TRUE if successful, FALSE otherwise.

Description: Move a control point or tangent (the closest detected) of the current curve.
See also: UserHCEditInit,

15.2.166 UserHCEditMoveCtlPt (hrmt_crv.c:953)

```
int UserHCEditMoveCtlPt(VoidPtr HC,
                        CagdRType OldX,
                        CagdRType OldY,
                        CagdRType NewX,
                        CagdRType NewY,
                        int MouseMode)
```

HC: A handle on edited Hermite curve to move control point in.

OldX, OldY: The original coordinate of the point.

NewX, NewY: The new coordinate of the point.

MouseMode: 0 for mouse down, 1 for mouse move, 2 for mouse up (done).

Returns: TRUE if successful, FALSE otherwise.

Description: Move a control point of the current curve.
See also: UserHCEditInit,

15.2.167 UserHCEditMoveCtlTan (hrmt_crv.c:1031)

```
int UserHCEditMoveCtlTan(VoidPtr HC,
                         CagdRType OldX,
                         CagdRType OldY,
                         CagdRType NewX,
                         CagdRType NewY,
                         int MouseMode)
```

HC: A handle on edited Hermite curve to move control tangent in.

OldX, OldY: The original coordinate of the tangent.

NewX, NewY: The new coordinate of the tangent.

MouseMode: 0 for mouse down, 1 for mouse move, 2 for mouse up (done).

Returns: TRUE if successful, FALSE otherwise.

Description: Move a control tangent of the current curve.
See also: UserHCEditInit,

15.2.168 UserHCEditRelativeTranslate (hrmt_crv.c:1788)

```
int UserHCEditRelativeTranslate(VoidPtr HC, CagdRType *Dir)
```

HC: A handle on edited Hermite curve to relatively transform, in place.

Dir: Relative translation amount, applied in full just to last control point.

Returns: TRUE if successful, FALSE otherwise.

Description: Applies a relative translation, in place, to given curve HC as specified by Dir. The full translation amount is applied to the last point while the first is kept stationary and all interior control points are moved their relative portion.
See also: UserHCEditInit,

15.2.169 UserHCEditSetCtlPtCont (hrmt_crv.c:318)

```
void UserHCEditSetCtlPtCont(VoidPtr HC, int Index, CagdBType Cont)
```

HC: A handle on edited Hermite curve to update its Index segment.

Index: Of control point to change its C^1 continuity.

Cont: TRUE to make C^1 continuous, FALSE for only C^0.

Returns: void

Description: Sets the continuity of the Index control point in the given curve.
See also: UserHCEditInit,

15.2.170 UserHCEditSetDrawCtlptFunc (hrmt_crv.c:356)

```
void UserHCEditSetDrawCtlptFunc(VoidPtr HC,
                                UserHCEditDrawCtlPtFuncType CtlPtDrawFunc,
                                void *FuncData)
```

HC: A handle on edited Hermite curve to update its Index segment.

CtlPtDrawFunc: Function to use to draw control points/tangent.

FuncData: Optional data to pass to the call back function.

Returns: void

Description: Sets the function to invoke when the control points/tangents are to be drawn.
See also: UserHCEditInit,

15.2.171 UserHCEditSetPeriodic (hrmt_crv.c:220)

```
void UserHCEditSetPeriodic(VoidPtr HC, CagdBType Periodic)
```

HC: A handle on Hermite curve to be set periodic (or not).

Periodic: TRUE if curve is to be closed (periodic).

Returns: void

Description: Sets the periodic conditions of the given HC curve.
See also: UserHCEditInit,

15.2.172 UserHCEditTransform (hrmt_crv.c:1738)

```
int UserHCEditTransform(VoidPtr HC, CagdRType *Dir, CagdRType Scl)
```

HC: A handle on edited Hermite curve to transform, in place.

Dir: Translation amount, in the XY plane.

Scl: Scaling amount, in the XY plane.

Returns: TRUE if successful, FALSE otherwise.

Description: Applies an affine transform, in place, to given curve HC as specified by Dir and Scale. Curve HC is first translated by Dir and then scaled by Scale.
See also: CagdCrvTransform, UserHCEditInit, UserHCEditMatTrans,

15.2.173 UserHCEditUpdateCtl (hrmt_crv.c:766)

```
int UserHCEditUpdateCtl(VoidPtr HC,
                        int CtlIndex,
                        CagdBType IsPosition,
                        CagdRType NewX,
                        CagdRType NewY)
```

HC: A handle on edited Hermite curve to move control pt/tan in.

CtlIndex: The index of the control pt/tan to update

IsPosition: TRUE if we aim at updating the position of control pt/tan. If FALSE, positive CtlIndex denotes forward tangent whereas negative CtlIndex denotes a backward tangent.

NewX, NewY: New position or tangent vector to update with.

Returns: TRUE if successful, FALSE otherwise.

Description: Uodate a control point or tangent of the current curve.
See also: UserHCEditInit,

15.2.174 UserHoberConstAngleMdl (hoberman.c:510)

```
IPObjectStruct *UserHoberConstAngleMdl(const CagdCrvStruct *Crv,
                                       CagdRType Offset,
                                       int NumOfScissors,
                                       int RefineCycles,
                                       const IrtVecType PinHoleDiams,
                                       CagdRType ScissorThickness,
                                       CagdRType ScissorRelWidth,
                                       CagdRType ScissorRoundRad,
                                       int AddColorCodes,
                                       CagdRType Tol)
```

Crv: Outline 2D curve to build the expanding model around.

Offset: Offset to apply to Crv to create the strip where the scissor elements will reside.

NumOfScissors: How many scissor-like elements in the shape.

RefineCycles: Zero to ignore, positive to execute that many refinement cycles.

PinHoleDiams: A vector of three values, in order: 1. Desired diameters of pins, 2. Desired diameters of Pin expansions (a bit bigger than diameters of pins. 3. Desired diameters of holes.

ScissorThickness: The Z dimension of each part of the scissors.

ScissorRelWidth: Relative (1.0 for neutral) width of scissors.

ScissorRoundRad: Rounding radius to apply to corners. Zero to disable.

AddColorCodes: TRUE to add color coded IDs to each end of the scissor.

Tol: Tolerances of computations.

Returns: Constructed Hoberman geometry.

Description: Creates a 2D Hoberman-like expanding structure of scissor-like elements. This constructions keeps the angle of teh scissors constant, while varying the distance to the origin of the angle of the scissors.
See also: UserHoberConstRadMdl,

15.2.175 UserHoberConstRadMdl (hoberman.c:117)

```
IPObjectStruct *UserHoberConstRadMdl(const CagdCrvStruct *Crv,
                                     CagdRType Offset,
                                     int NumOfScissors,
                                     int RefineCycles,
                                     const IrtVecType PinHoleDiams,
                                     CagdRType ScissorThickness,
                                     CagdRType ScissorRelWidth,
                                     int AddColorCodes,
                                     CagdRType Tol)
```

Crv: Outline 2D curve to build the expanding model around.

Offset: Offset to apply to Crv to create the strip where the scissor elements will reside.

NumOfScissors: How many scissor-like elements in the shape.

RefineCycles: Zero to ignore, positive to execute that many refinement cycles.

PinHoleDiams: A vector of three values, in order: 1. Desired diameters of pins, 2. Desired diameters of Pin expansions (a bit bigger than diameters of pins. 3. Desired diameters of holes.

ScissorThickness: The Z dimension of each part of the scissors.

ScissorRelWidth: Relative (1.0 for neutral) width of scissors.

AddColorCodes: TRUE to add color coded IDs to each end of the scissor.

Tol: Tolerances of computations.

Returns: Constructed Hoberman geometry.

Description: Creates a 2D Hoberman-like expanding structure of scissor-like elements. This constructions keeps the distance to the origin constant (constant radius) while varying the angles of the scissors.
See also: UserHoberConstAngleMdl,

15.2.176 UserInterSrfAtAllKnots (srf_cntr.c:645)

```
CagdSrfDirType UserInterSrfAtAllKnots(CagdSrfStruct *Srf,
                                      IrtPlnType Pln,
                                      int Axis,
                                      const CagdRType *KV,
                                      int MinKV,
                                      int MaxKV,
                                      CagdRType *Param)
```

Srf: To intersect.

Pln: Along which to divide TSrfs, or NULL if TSrfs need to be divided at Axis.

Axis: Along which to divide TSrfs. Ignored if Pln is not NULL.

KV: The knots to divide along.

MinKV, MaxKV: The KV range to consider.

Param: If detected an isoaprameteric values, set Param to it.

Returns: CAGD_NO_DIR if all intersections are general, or CAGD_CONST_U/V_DIR if found an isoparameteric intersection in the returned value and set Param to it.

Description: Intersect the given surface, Srf, at all locations Srf cross a t = KV[i], MinKV <= i < MaxKV, knot plane. The plane is defined either as (Pln[0], Pln[1], Pln[2], t), or, if the parameter Pln is NULL, as Axis = t. All resulting intersection curves are accumulated in the surfaces as "intercrvs" attributes, in place.
See also: UserDivideSrfAtInterCrvs, UserDivideSrfAtAllTVInteriorKnot,

15.2.177 UserInterSrfByAlignedHyperPlane (srf_cntr.c:532)

```
CagdCrvStruct *UserInterSrfByAlignedHyperPlane(const CagdSrfStruct *Srf,
                                               int Axis,
                                               CagdRType t)
```

Srf: To intersect.

Axis: The axis which is normal to the hyperplane.

t: The position of the hyperplane on the axis.

Returns: The intersection curve(s) of the surface with a plane.

Description: Intersect the given surface, Srf, with an axis-aligned hyperplane. The hyperplane is defined by an axis number and a value.
See also: UserDivideSrfAtInterCrvs, UserDivideSrfAtAllTVInteriorKnot2,

15.2.178 UserInterSrfByAlignedHyperPlane2 (srf_cntr.c:565)

```
MvarPolylineStruct *UserInterSrfByAlignedHyperPlane2(const CagdSrfStruct *Srf,
                                                     int Axis,
                                                     CagdRType t)
```

Srf: To intersect.

Axis: The axis which is normal to the hyperplane.

t: The position of the hyperplane on the axis.

Returns: A piecewise-linear approximation of the intersection of the surface with the plane.

Description: Intersect the given surface, Srf, with an axis-aligned hyperplane. The hyperplane is defined by an axis number and a value.
See also: UserDivideSrfAtInterCrvs, UserDivideSrfAtAllTVInteriorKnot2,

15.2.179 UserKnmtcsEvalAtParams (kinematc.c:1637)

```
void UserKnmtcsEvalAtParams(UserKnmtcsGenInfoStruct *GI,
                            int PolyIdx,
                            int PtIdx)
```

GI: Handle on general information of kinematics module.

PolyIdx: Polyline index.

PtIdx: Point index.

Returns: void

Description: Evaluates kinematic mechanism at given solution. Solution is stored as a linked list of polylines, each position is uniquely determined by indices of the polyline and the point inside this polyline

15.2.180 UserKnmtcsEvalCrvTraces (kinematc.c:1567)

```
CagdCrvStruct *UserKnmtcsEvalCrvTraces(UserKnmtcsGenInfoStruct *GI)
```

GI: Handle on general information of kinematics module.

Returns: List of curves of all traces of all points in the mechanisms.

Description: Evaluates the motions curves of all the points in the kinematic mechanism at the solution.

15.2.181 UserKnmtcsFreeSol (kinematc.c:1197)

```
void UserKnmtcsFreeSol(UserKnmtcsGenInfoStruct *GI)
```

GI: Handle on general information of kinematics module.

Returns: void

Description: Deallocates and frees the kinematic solution stored as a list of multi-variate polyline structures.

15.2.182 UserKnmtcsNumOfSolPts (kinematc.c:1538)

```
int UserKnmtcsNumOfSolPts(UserKnmtcsGenInfoStruct *GI, int PolyIdx)
```

GI: Handle on general information of kinematics module.

PolyIdx: Polyline index.

Returns: Number of points in the polyline.

Description: For a solution polyline at "PolyIdx"'s position, the number of solution points is returned.

15.2.183 UserKnmtcsSolveDone (kinematc.c:1215)

```
void UserKnmtcsSolveDone(UserKnmtcsGenInfoStruct *GI)
```

GI: Handle on general information of kinematics module.

Returns: void

Description: Reset all information given to the solver to initial state.

15.2.184 UserKnmtcsSolveMotion (kinematc.c:1249)

```
int UserKnmtcsSolveMotion(UserKnmtcsGenInfoStruct **GI,
                          const UserKnmtcsStruct *System,
                          CagdRType NumTol,
                          CagdRType SubTol,
                          CagdRType Step,
                          int *SolDim,
                          CagdBType FilterSols)
```

GI: Handle on newly (dynamically) general information of kinematics module.

System: Kinematic system.

NumTol: Numerical tolerance.

SubTol: Subdivision tolerance.

Step: "Animation" step.

SolDim: Dimension of the solution, even if invalid/error.

FilterSols: True to filter and return only numerically converging solutions.

Returns: Number of solution curves, if any. -1 if error.

Description: For given planar kinematic mechanism, the system of constrains is created and solved. The list of solution points (in the parametric space) is returned.

15.2.185 UserLineAcccessSrfLineAccessFromObject (ln_access.c:800)

```
UserLineAccessSrfLineAccessResStruct *UserLineAcccessSrfLineAccessFromObject(
                                                    const IPObjectStruct *Obj)
```

Obj: The object from which to reconstruct the result.

Returns: The reconstructed results structure (or NULL if failed).

Description: Reconstruct a UserLineAccessSrfLineAccessResStruct (a results structure from its Irit Object representation.
See also: UserLineAcccessSrfLineAccessToObject,

15.2.186 UserLineAccessSrfLineAccessResFree (ln_access.c:442)

```
void UserLineAccessSrfLineAccessResFree(
                           UserLineAccessSrfLineAccessResStruct *SrfLineAccess)
```

SrfLineAccess: To free.

Returns: void

Description: Frees a UserLineAccessSrfLineAccessResStruct and all its internal data structures.
See also: UserLineAccessSrfLineAccessResNew,

15.2.187 UserLineAccessSrfLineAccessibility (ln_access.c:2049)

Line-accessibility analysis

periodic surfaces

```
UserLineAccessSrfLineAccessResStruct *UserLineAccessSrfLineAccessibility(
                                 CagdSrfStruct *InSrfs,
                                 UserLineAccessParamsStruct *Params,
                                 IPObjectStruct **ExtraOutObjs)
```

InSrfs: The input surface to analyze for line-accessibility.

Params: The parameters struct for the line-accessibility analysis.

ExtraOutObjs: Output parameter. Any extra visualization data will be written into this parameter. Ignored if NULL.

Returns: A structure containing line-accessibility information for all the patches.

Description: Analyze a surface for accessibility for infinite tangent line-cutting. The function produces a UserLineAccessSrfLineAccessResStruct, which contains the line-accessibility information for each of the small patches into which the surface is partitioned.
See also: UserLineAccessSrfPreProcess,

15.2.188 UserLineAccessSrfPreProcess (ln_access.c:1868)

Line-accessibility analysis

periodic surfaces

```
CagdSrfStruct *UserLineAccessSrfPreProcess(CagdSrfStruct **Srf)
```

Srf: The surface to preprocess. Can be freed and replaced.

Returns: The preprocessed surface.

Description: Preprocess a surface for line-accessibility analysis. This function ensures that the surface is B-spline, and checks if it periodic in one of its directions, and if it collapses into a point at one of the ends of its domains.
See also: UserLineAccessSrfLineAccessibility,

15.2.189 UserLineCutPathToCutDirs (ln_access_cut.c:178)

```
IPVertexStruct *UserLineCutPathToCutDirs(
                const CagdSrfStruct *Srf,
                const UserLineAccessSrfLineAccessResStruct *SrfAccessibility,
                const CagdCrvStruct *CuttingCrv,
                CagdRType MinAngleFromPathTan,
                int NumSamples)
```

Srf: The surface for which the cutting directions are to be computed.

SrfAccessibility: The line accessibility results structure.

CuttingCrv: The cut-path curve in the parametric space of Srf.

MinAngleFromPathTan: The minimum allowed angle of a cutting tangent direction from the curve movement direction Used for preventing cutting tangent directions from being too close to the movement direction of the cut-path.

NumSamples: The number of points to sample on the cut-path.

Returns: A lits of points, each containing the Euclidean coordinates, and the direction of the cutting tangent.

Description: Compute the tangent-line cutting directions for a given cut-path curve, given a surface and a line accessibility results structure.
See also:

15.2.190 UserLineCutRobotPathToRuledSrf (ln_access_cut.c:1005)

```
CagdSrfStruct *UserLineCutRobotPathToRuledSrf(const IPVertexStruct *CutPath,
                                              CagdRType HalfWidth)
```

CutPath: Cut path to approximate as a ruled surface.

HalfWidth: One side (half) of the Width of built ruled surface.

Returns: Created ruled surfaces.

Description: Converts the cut path into a ruled surface.
See also:

15.2.191 UserMJAnchorFree (micro1join.c:481)

```
void UserMJAnchorFree(UserMJAnchorGeomStruct *Anchor)
```

Anchor: A data structure to be freed.

Returns: void

Description: Free the data structure to store an anchor used in tile joining.
See also:

15.2.192 UserMJAnchorFreeList (micro1join.c:503)

```
void UserMJAnchorFreeList(UserMJAnchorGeomStruct *Anchors)
```

Anchors: Anchors to be freed.

Returns: void

Description: Free a list of anchors.
See also:

15.2.193 UserMJComputeJointSrfDir (micro1join.c:1467)

```
CagdBType UserMJComputeJointSrfDir(const TrivTVStruct *TV,
                                   int JointSrfDir,
                                   CagdVecStruct *SrfDir)
```

TV: the trivariate to add a joint to.

JointSrfDir: Direction of tile joint surface to be computed. 0 to 5.

SrfDir: The computed direction vector.

Returns: TRUE if successful, FALSE otherwise.

Description: Computes the direction of the tile joint surface. The direction of the tile joint surface is defined as the partial derivative of the trivariate that the joint surface belongs to. For example is the tile joint surface is the UMin boundary of the tile trivariate, then the direction of this tile joint surface becomes dTdu(UMin, (VMin + VMax) * 0.5, (WMin + WMax) * 0.5). The boundary surface direction and the joint surface's reference TV should be determined and saved before calling this function.
See also:

15.2.194 UserMJComputeSrfProjDist (micro3join.c:416)

```
CagdRType UserMJComputeSrfProjDist(const CagdPType Pt,
                                   const IPObjectStruct *MacroGeom,
                                   UserMJPtProjStruct *ProjResult,
                                   void *AuxData)
```

Pt: A point to compute the projection distance.

MacroGeom: A surface to project the point onto.

ProjResult: The projection result including the projection distance and the projected point.

AuxData: Not used.

Returns: The squared distance between the input and the projection points.

Description: computes the minimum distance between the input point and the input surface (MacroGeom). The input point is projected onto the surface and the projection result is stored in ProjResult.
See also:

15.2.195 UserMJComputeTrimSrfProjDist (micro3join.c:463)

```
CagdRType UserMJComputeTrimSrfProjDist(const CagdPType Pt,
                                       const IPObjectStruct *MacroGeom,
                                       UserMJPtProjStruct *ProjResult,
                                       void *AuxData)
```

Pt: A point to compute the projection distance.

MacroGeom: A trimmed surface to project the point onto.

ProjResult: The projection result including the projection distance and the projected point.

AuxData: Not used.

Returns: The squared distance between the input and the projection points.

Description: computes the minimum distance between the input point and the input trimmed surface (MacroGeom). The input point is projected onto the trimmed surface and if the projected point is outside the trimmed surface, then project the outside point to the nearest trimming curve. The projection result is stored in ProjResult.
See also:

15.2.196 UserMJConnectMToNTilesInJISrfs (micro3tile.c:3729)

```
IPObjectStruct *UserMJConnectMToNTilesInJISrfs(
                                    UserMJJoinInterSrfRefStruct *JISrfRefList,
                                    CagdRType NrmScale,
                                    CagdRType NrmBlendingRatio,
                                    CagdRType SaddleRatio,
                                    CagdBType CheckJacobian)
```

JISrfRefList: A list of the pointers that point to the trimming surfaces that contains the anchoring faces in N-Set and R-Set.

NrmScale: The scale parameter that controls the length of the start and end directions of the sweeping axes of the tubular tiles. The length of the directions is equal to the distance between the start and end points multiplied by NrmScale.

NrmBlendingRatio: The scale parameter that controls the direction of the base trivariate in a bifurcation tile. If 0, the direction of the base trivariate is the derivariate of the trivariate adjacent to the inlet anchoring face. If 1, the direction is the surface normal of the inlet face. Otherwise, the direction of the base trivariate is blended from these two.

SaddleRatio: The scale parameter to control the size of the saddle in the base trivariate. Determines the size of top surface of the base trivariate.

CheckJacobian: If TRUE, the trivariates composed of bridging tiles are constructed to be positive Jacobian as much as possible.

Returns: A list of joint tiles connecting M interior surfaces and N touching surfaces. Consists of 1-1 and 1-2 tiles.

Description: Creates the bridging tiles that connect the anchoring faces of the inside tiles from one microstructure set to the anchoring faces of the boundary tiles from the other microstructure set. Two sets of anchoring faces are combined respectively from the sets stored in each trimming surface of the input trimming surface list. We denote the anchoring face set of the smaller size "R"(rare)-surface-set (R-Set) and the face set of the larger size "N"(numerous)-surface-set (N-Set). Then we find the matchings between R-Set and N-Set. Matchings can be either 1-to-1 or 1-to-2 and constructed via a two-step process: we first find the maximal number of 1-to-1 matchings that minimize the weighted sum of the matchings. The weight of each match is computed from the distance between the matched N and R surfaces. Only the surfaces visible from each other (this visibility is computed based on the anchoring directions) have an edge with a valid weight. The first matching gives us at most R matched pairs depending on the position of the surfaces and leaves us at least (N - R) unmatched surfaces from N-Set. In the second matching, we attempt to find the matches between the remaining (N - R) surfaces from N-set and R surfaces from R-Set. We also minimize the sum of weights in the second matching as well. Finally, we gather the first and second matching results and determine the types of each match. If R_1 surface is matched to N_1 surface in the first matching and N_2 surface in the second matching, we create a bifurcation tile that has R_1 surface as an inlet and N_1 and N_2 surface as outlet surfaces. If R_1 surface is solely matched to N_1, then we create a bridging tile between R_1 and N_1 by sweeping R_1 surface to The directions of the anchoring surfaces are computed by blending the partial derivatives of the trivariate and the surface normal directions using the scale parameter NrmScale.

15.2.197 UserMJConnectToJISrfs (micro3tile.c:3066)

```
IPObjectStruct *UserMJConnectToJISrfs(
                                    UserMJJoinInterSrfRefStruct *JISrfRefList,
                                    CagdRType NrmScale,
                                    CagdRType NrmBlendingRatio,
                                    CagdBType CheckJacobian)
```

JISrfRefList: A list of the pointers that refer to the trimming boundary surfaces storing the info of the tile anchoring faces to be bridged.

NrmScale: The scale parameter to adjust the size of start and end directions of the sweeping axis curve. Controls the shape of the swept volume.

NrmBlendingRatio: The scale parameter that controls the direction of the starting directon of the sweeping axis. If 1, the direction is the surface normal of the inlet face. Otherwise, the direction of the base trivariate is blended from these two.

CheckJacobian: If TRUE, the trivariates composed of bridging tiles are constructed to be positive Jacobian as much as possible.

Returns: A list of the bridging tiles. The list contains all bridging tiles connecting the interior tiles in the same primitive microstructure set, even though the tiles join to the different trimming surface.

Description: Computes the bridging tiles that bridge the anchoring faces of to-be-soon -bridged inside tiles of one microstructure set to their associated trimming boundary surfaces. Each anchoring face is stored in InteriorSrfs of the nearest trimming boundary surface, but we construct the bridging tiles altogether for the input trimming boundary surfaces. A bridging tile is created by sweeping the anchoring face to the trimming surface and the sweeping axis curve to connect the center point of the anchoring face, one intermediate point in the anchoring direction and its projection point on the trimming surface. The start direction of the sweeping axis (the anchoring direction) is derived by blending the partial derivative of the trivariate adjacent to the anchoring face and the surface normal of the anchoring surface, and scaled by NrmScale. After the axis curve of every swept tile is calculated, we detect the collisions between the bridging tiles and resolve the collisions if exist. Only the anchoring faces that are free from collision are used to create the bridging tiles.

15.2.198 UserMJEvalSrfPt (micro3join.c:527)

```
void UserMJEvalSrfPt(const IPObjectStruct *Geom,
                     int GeomId,
                     CagdUVType InUV,
                     CagdPType OutPt,
                     void *AuxData)
```

Geom: A geometry to evalulate the point. If Geom is a trimmed surface, then use the underlying untrimmed surface to evalulate the point. If Geom is a list type, then use the GeomId-th geometry from the list.

GeomId: The index of geometry (only used when Geom is a list).

InUV: (u,v) parameters of the surface.

OutPt: A position of the point.

AuxData: Not used.

Returns: void

Description: evaluates the position of a point on the surface for the given surface parameters.
See also:

15.2.199 UserMJJoinInterSrfFree (micro1join.c:423)

```
void UserMJJoinInterSrfFree(UserMJJoinInterSrfStruct *JISrf)
```

JISrf: A data structure to be freed.

Returns: void

Description: Free the data structure to store a surface that trimming a volume in micro-join.
See also:

15.2.200 UserMJJoinInterSrfListFree (micro1join.c:453)

```
void UserMJJoinInterSrfListFree(UserMJJoinInterSrfStruct *JISrfList)
```

JISrfList: A list of the surfaces trimming volumes and used in micro join.

Returns: void

Description: Free a list of the surfaces that interface trimming volumes in micro joining.
See also:

15.2.201 UserMJJoinOneSidedPtAnchors (micro3tile.c:4715)

```
IPObjectStruct *UserMJJoinOneSidedPtAnchors(
                                        UserMJAnchorGeomStruct *TileAnchors,
                                        int NumAnchors,
                                        CagdRType NrmScale,
                                        UserMJCBStruct *MJCBs)
```

TileAnchors: A list of the pointers that refer to the point anchors of curve tiles.

NumAnchors: The number of tile anchors in TileAnchors.

NrmScale: The scale parameter to adjust the size of start and end directions of the sweeping axis curve. Controls the shape of the swept volume.

MJCBs: A structure to save callback functions and auxiliary info to handle the macro boundary object.

Returns: A list of the bridging tiles. The list contains all bridging tiles connecting the anchor point to the B-reps of the macro object.

Description: Computes the bridging tiles that bridge the anchoring points of to-be- soon-bridged inside curve tiles of one microstructure set to their associated trimming B-reps. A list of precomputed anchoring points is given as an input and a bridging tile is created for each anchoring point by projecting the anchoring point to the nearest B-rep of the macro obejct. For each anchor point, a quadratic curve starting from the anchor point and ending in the trimming B-rep is constructed and saved as a bridging tile.

See also:

15.2.202 UserMJJoinOneSidedSrfAnchors (micro3tile.c:4827)

```
IPObjectStruct *UserMJJoinOneSidedSrfAnchors(
                                        UserMJAnchorGeomStruct *TileAnchors,
                                        int NumAnchors,
                                        CagdRType NrmScale,
                                        CagdRType LstSrfScale,
                                        CagdBType CheckJacobian,
                                        UserMJCBStruct *MJCBs)
```

TileAnchors: A list of the pointers that refer to the anchoring faces.

NumAnchors: The number of the anchors to construct the sweeping tile.

NrmScale: The scale parameter to adjust the size of start and end directions of the sweeping axis curve. Controls the shape of the swept volume.

LstSrfScale: The scale parameter to scale the outlet surface with to the inlet surface.

CheckJacobian: If TRUE, the trivariates composed of bridging tiles are constructed to be positive Jacobian as much as possible.

MJCBs: A structure to save callback functions and auxiliary info to handle the macro boundary object.

Returns: A list of the bridging tiles. The list contains all bridging tiles connecting the interior tiles in the same primitive microstructure set, even though the tiles join to the different macro B-reps.

Description: Computes the bridging tiles that bridge the anchoring faces of to-be-soon -bridged inside tiles of one microstructure set to their associated macro B-reps. A list of anchoring faces is stored in TileAnchors, with the auxiliary information such as anchoring direction, the closest macro B-rep, and the adjacent tile information if exist. Then we construct the bridging tile by sweeping the anchor surfaces to their nearest macro B-reps. The sweeping axis curve connects the center point of the anchoring face, one intermediate point in the anchoring direction and its projection point on the trimming surface. The start direction of the sweeping axis (the anchoring direction) is stored in each anchor struct, but scaled by NrmScale when constructing the sweeping axis curve. After the axis curve of every swept tile is calculated, we detect the collisions between the bridging tiles and resolve the collisions if exist. Only the anchoring faces that are free from collision are used to create the bridging tiles.

See also:

15.2.203 UserMJMarkInteriorJointTiles (micro1join.c:979)

```
void UserMJMarkInteriorJointTiles(IPObjectStruct *TileList,
                                  const UserMJTileStrctStruct *MJStrct)
```

TileList: A list of micro tiles with membership info stored in the R-value of RGB color attribute. The information of to-be-soon-bridging will be stored in the G-value in the end of this function call. function call.

MJStrct: Micro tile structure info.

Returns: void

Description: Detect to-be-soon-bridged inside micro tiles. These to-be-soon-bridged tiles are the tiles that are totally inside the trimmed model and adjacent to the tiles intersecting the non-isoparametric trimming surfaces of the model. When bridging micro-tiles, the to-be-soon-bridged inside tiles are directly connected to the trimming surface when the trimming surface is the outmost boundary surface of the VModel or connected to the boundary tiles in the other microstructure set. This function assumes that the micro tiles are constructed by the regular tiling - tiles are placed in a 3D lattice pattern. In this tiling method, we detect the to-be-soon-bridged inside tiles are detected by collecting all the inside tiles among six neighbor tiles of the on-the- boundary tiles.
See also:

15.2.204 UserMJMergeAndSaveTileTVs (micro3tile.c:4249)

```
IPObjectStruct *UserMJMergeAndSaveTileTVs(UserMJTileStrctStruct **MTileList,
                                          int NumVMdlMacroTVs,
                                          const IPObjectStruct *BridgeTiles,
                                          CagdBType CheckJacobian)
```

MTileList: A list of the primitive microtile sets. Each tile set contains the regular tiles constructed from function composition and the base trivariate which represents the macro shape in which the micro tiles are populated. If all of the tiles are used, then the RGB attribute of the list is set to 1. If the tiles are partially used, the membership information of each tile is encoded in the RGB value of the individual tile.

NumVMdlMacroTVs: Number of the regular tile sets.

BridgeTiles: A list of the briding tile sets. Each briding tile sets contains the briding tiles connecting the tiles in one primitive tile set to their nearest trimming surfaces or the briding tiles joining two different primitive tile sets.

CheckJacobian: If TRUE, the color of each trivariate is determined based on its Jacobian positivity test result. Otherwise the color is determined based on the tile membership test result.

Returns: TRUE if success. FALSE otherwise.

Description: Merge the regular tiles and the bridging tiles into a single list and save the auxiliary information in each trivariate. Each trivariate stores the following information in its attributes:
TrivID - A unique integer ID is assigned to each trivariate. Tile ID and PartID is encoded in TrivID as follows. TrivID = TileID * 10 + PartID For a regular tile, PartID corresponds to the trivariate ID inside the unit tile, and PartID is always 0 for a bridging tile. Now, PartID only supports up to ten-trivariates` unit tile. VMdlTVId - An non-negative integer ID to represent the macro TV to which this trivariate belongs. -1 in case of the bridging tiles. TrivAdjID[UMin |UMax|VMin|VMax|WMin|WMax] - IDs of the adjacent trivariates are stored with the face directions they are interfacing with the modification flags to make them compatible. For instance, if a trivariate 0's UMin iso-surface interfaces with a trivariate 4's WMax iso-surface, TV0 has the attribute of [TrivAdjIDUMin 4 flag0], while TV4 has the attribute of [TrivAdjWMax 0 flag1]. Flag0 and Flag1 are the bitmask of the modification flags from CagdSrfSame3. TrivJcbBnd - The ratio of the minimum margin in the domain of the trivariate, which guarantees the positive Jacobian. It can be one of the values in ClipRatios. If it fails to guarantee positive Jacobian, -1 is saved. Note some trivariates have, by definition, zero Jacobian on the boundaries (like the rounded ones), and that this test is conservative as we compute J symbolically and examine the bounding box of the coefficients of J. TrivCollide - When the global collision between two branch bridging TVs are not resolved in the final microstructure, we mark the colliding pair to their counter parts. rgb - Color is assigned to each trivariate based on TrivJcbBnd.

15.2.205 UserMJMicroFieldTiles (micro3join.c:2048)

```
IPObjectStruct *UserMJMicroFieldTiles(const IPObjectStruct *Tiles,
                                      const IPObjectStruct *TileField,
                                      const IPObjectStruct *MacroGeomObj,
                                      CagdRType AxisNrmRatio,
                                      CagdRType AxisNrmBlendingRatio,
                                      CagdRType LastSrfScale)
```

Tiles: A list of microtiles constructed by regular tiling in the trivariate deformation map TileField.

TileField: A deformation map bounding the macro object and used in microtile construction.

MacroGeomObj: A B-rep macro object to fill the microtiles in.

AxisNrmRatio: The scale parameter that controls the length of the start and end directions of the sweeping axes of the tubular tiles. The length of the directions is equal to the distance between the start and end points multiplied by NrmScale.

AxisNrmBlendingRatio: Only used in trivariate tiles. The scale parameter that controls the direction sweeping axes of the tubular tiles. If 0, the starting direction of the sweeping axis curve derives from the derivative of the trivariate adjacent to the inlet anchoring face. If 1, the direction is the surface normal of the inlet face. Otherwise, the direction is blended from these two.

LastSrfScale: The scale parameter to scale the outlet surface with respect to the inlet surface.

Returns: A list of micro tiles in the B-rep macro object.

Description: Construct the microtiles in the B-rep macro model. The B-rep macro model can be either spline surfaces, trimmed surfaces, a model, a V-model, or a triangular mesh (polygonal mesh). The microtiles must have a homogenous type, either a trivariate, surface, or curve type. As an input the B-rep macro object, a trivariate deformation map (field) bounding the macro object and a list of microtiles constructed in the deformation map should be given. Then the microtiles are classified based on their membership with respect to the B-rep macro object: either as fully inside the tile, intersecting with the macro B-rep, or fully outside the B-rep. For the tiles fully inside the macro boundary object, then anchors which connect the inside tiles to the macro B-rep are identified: For trivariate and surface tiles, the boundary surfaces of tiles become the anchor; for curve tiles, end points of the curve can be anchors. After the anchors are identified, new tiles are constructed and inserted to connect the existing tiles to the B-rep macro object conformally.

See also:

15.2.206 UserMJMicroJoinSynthesize (micro1join.c:2223)

```
IPObjectStruct *UserMJMicroJoinSynthesize(const VMdlVModelStruct *VMdlOrig,
                                          CagdRType NrmScale,
                                          CagdRType NrmBlendingRatio,
                                          CagdRType SaddleRatio,
                                          CagdRType BndTileMargin,
                                          CagdBType CheckJacobian)
```

VMdlOrig: The trimmed VModel representing the macro shape. Each trivariate in the VModel stores the tiling information and the priority value in the attribute "MVMSInfo", if exists.

NrmScale: The scale parameter that controls the length of the start and end directions of the sweeping axes of the tubular tiles. The length of the directions is equal to the distance between the start and end points multiplied by NrmScale.

NrmBlendingRatio: The scale parameter that controls the direction of the base trivariate in a bifurcation tile. If 0, the direction of the base trivariate is the derivative of the trivariate adjacent to the inlet anchoring face. If 1, the direction is the surface normal of the inlet face. Otherwise, the direction of the base trivariate is blended from these two.

SaddleRatio: The scale parameter to control the size of the saddle in the base trivariate. Determines the size of top surface of the base trivariate.

BndTileMargin: If BndTileMargin > 0, the tiles not only intersecting with the trimming surface but also having the minimum distance of one trivariate smaller than the size of the trivariate multiplied by BndTileMargin are classified as intersecting tiles and purged away.

CheckJacobian: If TRUE, the trivariates composed of bridging tiles are constructed to be positive Jacobian as much as possible.

Returns: A list of micro tiles in the trimmed VModel.

Description: Synthesize the microtile in the trimmed VModel. A trimmed VModel has a list of V-cells, each of which holds the list of trivariates that contribute to the construction of the cell. Microstructures are generated according to the priority of these trivariates: from the trivariate of highest priority, we first gather V-cells that hold the given trivariate and have not been tiled yet. The trimming surfaces of these gathered V-cells are collected, which are the outmost boundary surfaces that do not originate from the current trivariate. These trimming surfaces are used in testing the membership of each tile composed in the trivariate: The tiles generated by function composition are classified as either inside (the V-cells), outside, or on-the-boundary, which transversally intersect with the trimming surfaces. All the other tiles except the inside tiles are purged and, among the inside tiles, to-be-bridged tiles are identified. We then insert the new bridging tiles from some anchoring faces of the to-be-bridged tiles to the nearest trimming surfaces (in case that the trimming surface is the outmost boundary surface of the entire VModel), or the anchoring faces of other tiles composed in other trivariate (in case that the trimming surface is an interface between two V-Cells tiled by the different trivariates. The form of the new bridging tile is either a tubular tile that bridges two anchoring faces or one anchoring face to the trimming surface, or a bifurcation tile that bridges one inlet face from one tile set to two outlet anchoring faces in the other tile set.

15.2.207 UserMJSrfNormal (micro3join.c:572)

```
void UserMJSrfNormal(const IPObjectStruct *Geom,
                     int GeomId,
                     CagdUVType InUV,
                     CagdPType OutNrm,
                     void *AuxData)
```

Geom: A geometry to evalulate the surface normal. If Geom is a trimmed surface, then use the underlying untrimmed surface to evalulate the normal. If Geom is a list type, then use the GeomId-th geometry from the list.

GeomId: The index of geometry (only used when Geom is a list).

InUV: (u,v) parameters of the surface.

OutNrm: A unit normal computed.

AuxData: Not used.

Returns: void

Description: evaluates the unit surface normal for the given surface parameters.
See also:

15.2.208 UserMJSrfTangent (micro3join.c:621)

```
void UserMJSrfTangent(const IPObjectStruct *Geom,
                      int GeomId,
                      CagdUVType InUV,
                      CagdSrfDirType Dir,
                      CagdVType OutTan,
                      void *AuxData)
```

Geom: A geometry to evalulate the surface tangent. If Geom is a trimmed surface, then use the underlying untrimmed surface to evalulate the tangent. If Geom is a list type, then use the GeomId-th geometry from the list.

GeomId: The index of geometry (only used when Geom is a list).

InUV: (u,v) parameters of the surface.

Dir: The surface direction to compute the tangent vector.

OutTan: A unit tangent computed.

AuxData: Not used.

Returns: void

Description: evaluates the unit tangent vector for the given direction on the surface.
See also:

15.2.209 UserMJTileJointSrfFree (micro1join.c:392)

```
void UserMJTileJointSrfFree(UserMJTileJointSrfStruct *TileJSrf)
```

TileJSrf: A list of tile bound surfaces in micro joining.

Returns: void

Description: Free a list of data structure to store the information of tile joint surfaces used in joining micro tiles.
See also:

15.2.210 UserMJTileStrctFree (micro1join.c:338)

```
void UserMJTileStrctFree(UserMJTileStrctStruct *MJT)
```

MJT: The data structure to be freed.

Returns: void

Description: Free the data structure to store micro tiling results.
See also:

15.2.211 UserMJTileStrctListFree (micro1join.c:364)

```
void UserMJTileStrctListFree(UserMJTileStrctStruct *MJTList)
```

MJTList: A list of data structures to be freed.

Returns: void

Description: Free a list of data structure to store micro tiling results at once.
See also:

15.2.212 UserMJTileStrctNew (micro1join.c:208)

```
UserMJTileStrctStruct *UserMJTileStrctNew(const TrivTVStruct *BaseMap,
                                          const IPObjectStruct *MicroTiles)
```

BaseMap: Simple trivariate map to define a macro shape.

MicroTiles: A list of micro tile geometries.

Returns: A data structure to store the micro-tiling information.

Description: Allocate a memory to store the information needed in joining micro tiles.
See also: UserMJTileStrctNew2,

15.2.213 UserMJTileStrctNew2 (micro1join.c:271)

```
UserMJTileStrctStruct *UserMJTileStrctNew2(const TrivTVStruct *BaseMap,
                                           const IPObjectStruct *MicroTiles)
```

BaseMap: Simple trivariate map to define the field where the tiles are defined.

MicroTiles: A list of micro tile geometries.

Returns: A data structure to store the micro-tiling information.

Description: Allocate a memory to store the information of tiles. Unlike UserMJTileStrctNew2, the order of each tile is not reduced. Tiles must have the same geometric type (e.g., point, surface, trivariate).
See also: UserMJTileStrctNew,

15.2.214 UserMJVMdlAdjustVModelWithClipping (micro1join.c:1620)

```
VMdlVModelStruct *UserMJVMdlAdjustVModelWithClipping(
                                                const VMdlVModelStruct *VMdl)
```

VMdl: The original trimmed VModel.

Returns: The new trimmed VModel with the adjustment in the region of the trimming surfaces.

Description: Recompute the clipping region of the trimmed surfaces in the VModel. The original VModel has a margin in the domain when clipping because of the surface-surface-intersection computation. However, this margin generates the overlapping between the adjacent trimming surfaces and this sometimes leads to unwanted results in the identification of the opposite surfaces in the VModel. In this function, each trimming surface in the VModel is clipped again with the tighter bound (with no margin in the domain). Then the relationship of the opposite trimming surfaces is also recomputed using the new clipped regions.
See also:

15.2.215 UserMJVMdlBooleanCallBack (micro1join.c:2073)

```
void UserMJVMdlBooleanCallBack(const VMdlParamsStruct *Params,
                               const VMdlVModelStruct *VMdl1,
                               const VMdlVModelStruct *VMdl2,
                               VMdlVModelStruct *ResVMdl,
                               VMdlBoolOpType BoolOp,
                               void *AuxData)
```

Params: The tiling information retrieved from the attribute of IPObjectStruct.

VMdl1: The first operand of the boolean operator.

VMdl2: The second operand of the boolean operator.

ResVMdl: The output VModel computed from the given boolean operation.

BoolOp: The type of boolean operator. Could be one of the following VMDL_BOOL_OP_SUBTRACTION, VMDL_BOOL_OP_UNION or VMDL_BOOL_OP_INTERSECTION.

AuxData: The hash table that maps the trivariates in the input VModels to the trivariates in the output VModel.

Returns: void

Description: Callback function called in the VModel boolean operation. Copy the micro tiling information of the trivariates of the input VModels to the trivariates of output VModel. Depending on the type of boolean operator, the function behaves differently. In case of subtraction, information of the first operation only is copied. In case of union and intersection, the informtion of both operands are copied. In the special case when one of operands is NULL, we search the tiling information stored in the attribute of IPObjectStruct.

15.2.216 UserMake3DStatueFrom2Images (imgshd3d.c:129)

```
IPObjectStruct *UserMake3DStatueFrom2Images(const char *Image1Name,
                                            const char *Image2Name,
                                            int DoTexture,
                                            const IPObjectStruct *Blob,
                                            User3DSpreadType BlobSpreadMethod,
                                            UserImgShd3dBlobColorType
                                                          BlobColorMethod,
                                            int Resolution,
                                            int Negative,
                                            IrtRType Intensity,
                                            IrtRType MinIntensity,
                                            int MergePolys)
```

Image1Name: Name of 1st image to load.

Image2Name: Name of 2nd image to load.

DoTexture: TRUE to add 'uvvals' attributes, FALSE to add actual color to the vertices of the objels.

Blob: If specified used as the blob. If NULL, a 2-cross is used. Blob must be of size one in each axis, probably centered around the origin. Must be a list of 3 objects for blob coloring methods other than "No color".

BlobSpreadMethod: Method of spreading the blobs.

BlobColorMethod: Method of coloring each blob.

Resolution: Resolution of created objects (n^2 objects are created).

Negative: Default (FALSE) is white blobs over dark background. If TRUE, assume dark blobs over white background.

Intensity: The gray level affect on the blobs' scale.

MinIntensity: Minimum intensity allowed. Zero will collapse the blobl completely in one direction which will render it impossible to actually manufacture it.

MergePolys: TRUE to merge all objects' polygons into one object.

Returns: A list (poly if MergePolys) object of Resolution^2 spherical blobs.

Description: Creates Resolution^2 blobs that looks like the 1st image (gray level) from the XZ plane and like the 2nd image from the YZ plane.

15.2.217 UserMake3DStatueFrom3Images (imgshd3d.c:331)

```
IPObjectStruct *UserMake3DStatueFrom3Images(const char *Image1Name,
                                            const char *Image2Name,
                                            const char *Image3Name,
                                            int DoTexture,
                                            const IPObjectStruct *Blob,
                                            User3DSpreadType BlobSpreadMethod,
                                            UserImgShd3dBlobColorType
                                                          BlobColorMethod,
                                            int Resolution,
                                            int Negative,
                                            IrtRType Intensity,
                                            IrtRType MinIntensity,
                                            int MergePolys)
```

Image1Name: Name of 1st image to load.

Image2Name: Name of 2nd image to load.

Image3Name: Name of 3rd image to load.

DoTexture: TRUE to add 'uvvals' attributes, FALSE to add actual color to the vertices of the objels.

Blob: If specified used as the blob. If NULL, a 3-cross is used. Blob must be of size one in each axis, probably centered around the origin. Should hold 3 polygons for "No color" color method and should hold a list of 3 objects for the other methods.

BlobSpreadMethod: Method of spreading the blobs.

BlobColorMethod: Method of coloring each blob.

Resolution: Resolution of created objects (n^2 objects are created).

Negative: Default (FALSE) is white blobs over dark background. If TRUE, assume dark blobs over white background.

Intensity: The gray level affect on the blobs' scale.

MinIntensity: Minimum intensity allowed. Zero will collapse the blobl completely in one direction which will render it impossible to actually manufacture it.

MergePolys: TRUE to merge all objects' polygons into one object.

Returns: A list (poly if MergePolys) object of Resolution^2 spherical blobs.

Description: Creates Resolution^2 3D cross blobs that looks like the 1st image (gray level) from the XZ plane, like the 2nd image from the YZ plane, and like the 3rd image from the XY plane. A 3D cross is used as a blob.

15.2.218 UserMarchOnPolygons (srf_mrch.c:270)

```
IPPolygonStruct *UserMarchOnPolygons(const IPObjectStruct *PObj,
                                     UserSrfMarchType MarchType,
                                     const IPPolygonStruct *PlHead,
                                     IPVertexStruct *VHead,
                                     CagdRType Length)
```

PObj: This polygonal object we march on.

MarchType: Type of march - isoparametric, lines of curvature, etc.

PlHead: Polygon from the polygonal mesh with start at.

VHead: Starting vertex of the march. Must be on Pl.

Length: Length of March.

Returns: A piecewise linear approximation of the march.

Description: Marches and create a polyline on the given polygonal mesh, of given Length. Mesh is assumed to have adjacency information, and normals and curvature information at the vertices, if MarchType is SILHOUTETE or CURVATURE, respectively. Further, the mesh is assumed to consist of triangles only, and is regularized.
See also: BoolGenAdjacencies, UserMarchOnSurface,

15.2.219 UserMarchOnSurface (srf_mrch.c:73)

```
IPPolygonStruct *UserMarchOnSurface(UserSrfMarchType MarchType,
                                    const CagdUVType UVOrig,
                                    const CagdVType DirOrig,
                                    const CagdSrfStruct *Srf,
                                    const CagdSrfStruct *NSrf,
                                    const CagdSrfStruct *DuSrf,
                                    const CagdSrfStruct *DvSrf,
                                    CagdRType Length,
                                    CagdRType FineNess,
                                    CagdBType ClosedInU,
                                    CagdBType ClosedInV)
```

MarchType: Type of march - isoparametric, lines of curvature, etc. for USER_SRF_MARCH_PRIN_CRVTR type, it is assumed that SymbEvalSrfCurvPrep was invoked on Srf before this function is called for the proper preparations.

UVOrig: Origin on surface where the march starts.

DirOrig: Direction to march on surface (projected to tangent plane). If, however, MarchType == SRF_MARCH_ISO_PARAM Dir contains the direction in the parametric, UV, space.

Srf: Surface to march on.

NSrf: Normal field of surface to march on (optional).

DuSrf: Partial with respect to u (optional).

DvSrf: Partial with respect to v (optional).

Length: Length of March.

FineNess: Number of estimated steps along the approximated march.

ClosedInU: TRUE if surface is closed in U direction, FALSE otherwise.

ClosedInV: TRUE if surface is closed in V direction, FALSE otherwise.

Returns: A piecewise linear approximation of the march.

Description: Marches and create a polyline on the surface of given Length. NSrf, DuSrf, and DvSrf are computed locally if not provided which would reduce the efficiency of a sequence of surface marching procedures, on the same surface.
See also: SymbEvalSrfCurvPrep, UserMarchOnPolygons,

15.2.220 UserMicro3DCrossTile (micro1tile.c:1906)

```
IPObjectStruct *UserMicro3DCrossTile(
                                    const UserMicroTileBndryPrmStruct *UMinPrms,
                                    const UserMicroTileBndryPrmStruct *UMaxPrms,
                                    const UserMicroTileBndryPrmStruct *VMinPrms,
                                    const UserMicroTileBndryPrmStruct *VMaxPrms,
                                    const UserMicroTileBndryPrmStruct *WMinPrms,
                                    const UserMicroTileBndryPrmStruct *WMaxPrms,
                                    int NegativeSpace,
                                    char ** const Error)
```

UMinPrms, UMaxPrms, VMinPrms, VMaxPrms, WMinPrms, WMaxPrms: The boundary parameters of the six faces of the tile.

NegativeSpace: TRUE to also build the negative space of the tile. Instead of a trivariate object, a list object of two trivariates is returns, for the positive and negative spaces.

Error: An optional string to return the errors, inf any. Can be NULL to print errors to stderr.

Returns: Constructed tile or NULL if error.

Description: Constructs a 3D cross tile in the unit cube based on the given boundary parameters. Resulting tile will consist of trivariates.
See also: UserMicroBifurcate1to2Tile,

15.2.221 UserMicro4SidedTextureRagTile (micro4tile.c:334)

```
IPObjectStruct *UserMicro4SidedTextureRagTile(CagdRType WallThickness,
                                              int DiagEdge,
                                              int RetSrfObj)
```

WallThickness: The thickness of the vertical walls. <0.5, and 0.05 is a good starting value.

DiagEdge: TRUE for diagonal front edge. FALSE for horizontal edge.

RetSrfObj: TRUE to return a surface object, FALSE a list object of single-surface object.

Returns: Constructed tile or NULL if error.

Description: Build tile geometry with 5 different textured geometry from 5 different directions, as a hollowed cube. Texture is placed on the 5 interior surfaces of the hollowed cube 9with bottom).
See also:

15.2.222 UserMicroAuxClipFlexEnds (micro6tile.c:710)

```
TrivTVStruct *UserMicroAuxClipFlexEnds(TrivTVStruct **TVs,
                                       CagdRType FlexClipRatio,
                                       char ** const Error)
```

TVs: All the (mostly rigid) TVs to clip beginning/end, in place.

FlexClipRatio: To clip in W that is assumed [0, 1]. Can be between zero and 1/2, to clip at ClipRatio and 1.0-ClipRatio.

Error: An optional string to return the errors, inf any. Can be NULL to print errors to stderr.

Returns: The clipped beginnings/ends that are expected to be flexible. NULL if errors or no clipping was applied.

Description: Clips the beginning/end domains of these rigid TVs, as the are expected to be flexible and return these flexible parts. Clipping is done in place.
See also: UserMicroAuxFrameTile, UserMicroAuxFrameTile,

15.2.223 UserMicroAuxFrameTile (micro6tile.c:119)

```
IPObjectStruct *UserMicroAuxFrameTile(
                      const UserMicroAuxeticTileBndryPrmStruct *FaceParams,
                      int CircularBars,
                      int MakeJointsFlex,
                      char ** const Error)
```

FaceParams: Properties of the respective tile face.

CircularBars: TRUE for circular bars, FALSE for square.

MakeJointsFlex: TRUE to make the joints from flexible material, FALSE to set them as rigid.

Error: An optional string to return the errors, inf any. Can be NULL to print errors to stderr.

Returns: Constructed tile or NULL if error.

Description: Constructs a 3D tile with Auxetic behavior. Properties of the top and bottom faces are derives from the side faces.
See also: UserMicroBifurcate1to2Tile, UserMicroZSpringTile, UserMicro3DCrossTile, , UserMicroDiagTile1,

15.2.224 UserMicroAuxeticTile (micro6tile.c:176)

```
IPObjectStruct *UserMicroAuxeticTile(
                                UserMicroAuxeticTileBndryPrmStruct *XMinParams,
                                UserMicroAuxeticTileBndryPrmStruct *XMaxParams,
                                UserMicroAuxeticTileBndryPrmStruct *YMinParams,
                                UserMicroAuxeticTileBndryPrmStruct *YMaxParams,
                                UserMicroAuxeticTileBndryPrmStruct *ZMinParams,
                                UserMicroAuxeticTileBndryPrmStruct *ZMaxParams,
                                int CircularBars,
                                CagdRType FlexClipRatio,
                                CagdBType MakeJointsFlex,
                                CagdRType NegPossionXYShift,
                                char ** const Error)
```

XMinParams, XMaxParams: Properties of the respective tile X faces.

YMinParams, YMaxParams: Properties of the respective tile Y faces.

ZMinParams, ZMaxParams: Properties of the respective tile Z faces.

CircularBars: TRUE for circular bars, FALSE for square.

FlexClipRatio: Portion of rigid bars that are clipped to be flexible. Should be be in the (0.0, 0.5) range.

MakeJointsFlex: TRUE to make the joints from flexible material, FALSE to set them as rigid.

NegPossionXYShift: XY Shift amount of joints with negative Poisson's ratio to prevent collisions with neighboring tiles. Zero to disable.

Error: An optional string to return the errors, inf any. Can be NULL to print errors to stderr.

Returns: Constructed tile or NULL if error.

Description: Constructs a 3D tile with Auxetic behavior. Properties of the top and bottom faces are derives from the side faces.
See also: UserMicroBifurcate1to2Tile, UserMicroZSpringTile, UserMicro3DCrossTile, , UserMicroDiagTile1,

15.2.225 UserMicroBendingTile (micro4tile.c:734)

```
IPObjectStruct *UserMicroBendingTile(const CagdRType FlexWidthHeight[2],
                                     CagdRType RigidLength,
                                     CagdBType BendXY,
                                     CagdBType Merged,
                                     char ** const Error)
```

FlexWidthHeight: The width and height of the central bending material. Both in the (0, 1] range.

RigidLength: The length of two left/right rigid regions. In the (0, 1] range.

BendXY: TRUE, to build a tile that will bend in X and Y, FALSE, for a tile that only bends along X.

Merged: TRUE to merge all tile into one part. FALSE to return the solid part and flexible parts in two lists.

Error: An optional string to return the errors, inf any. Can be NULL to print errors to stderr.

Returns: Object of trivariates representing the bending tile.

Description: Generates a bending tile, as multiple trivariates.
See also:

15.2.226 UserMicroBiStableCollapse2D (micro5tile.c:2524)

```
IPObjectStruct *UserMicroBiStableCollapse2D(CagdRType FrameThickness,
                                            CagdRType JointsSize,
                                            CagdRType SideJointLevel,
                                            CagdRType FlexClipRatio,
                                            const CagdRType *HelicalArms,
                                            char ** const Error)
```

FrameThickness: The thickness of the whole frame.

JointsSize: Size of left/right/top/bottom joints on boundaries.

SideJointLevel: Level of elevation of the left/right joints.

FlexClipRatio: To clip along the rods that are partially flexible. Between (0, 0.5), to clip at ClipRatio and 1.0-ClipRatio.

HelicalArms: Optional parameter to create helical arms. If exists must hold the parameters (NumLoops, MnrRad, MjrRad) in the range of [0, 10] loops (0 to disable), (0.0, 0.5) for MnrRad and (0.0, 5.0) for MjrRad.

Error: To be updated with a string error if had one, or NULL to send errors to stderr.

Returns: A list of two lists of trivariates, first of rigid materials and the second of flexible (rods).

Description: Constructs a 2D tile (with some thickness) with a bistable state.
See also: UserMicroBifurcate1to2Tile, UserMicroZSpringTile, UserMicro3DCrossTile, , UserMicroDiagTile1, UserMicroBiStableTile2D, UserMicroBiStableTile3D, , UserMicroBiStableCollapse2DXY,

15.2.227 UserMicroBiStableCollapse2DXY (micro5tile.c:2765)

```
IPObjectStruct *UserMicroBiStableCollapse2DXY(CagdRType FrameThickness,
                                              CagdRType JointsSize,
                                              CagdRType FlexClipRatio,
                                              char ** const Error)
```

FrameThickness: The thickness of the whole frame.

JointsSize: Size of left/right/top/bottom joints on boundaries.

FlexClipRatio: To clip along the rods that are partially flexible. Between (0, 0.5), to clip at ClipRatio and 1.0-ClipRatio.

Error: To be updated with a string error if had one, or NULL to send errors to stderr.

Returns: A list of two lists of trivariates, first of rigid materials and the second of flexible (rods).

Description: Constructs a 2D tile (with some thickness) with a bistable state, in both X and Y directions.
See also: UserMicroBifurcate1to2Tile, UserMicroZSpringTile, UserMicro3DCrossTile, , UserMicroDiagTile1, UserMicroBiStableTile2D, UserMicroBiStableTile3D, , UserMicroBiStableCollapse2D,

15.2.228 UserMicroBiStableTile2D (micro5tile.c:1597)

```
IPObjectStruct *UserMicroBiStableTile2D(CagdRType FrameSize,
                                        CagdRType FrameHGap,
                                        CagdRType WeightRodsThickness,
                                        CagdRType WeightRodsSpring,
                                        CagdRType WeightRodsAngle,
                                        CagdRType FlexClipRatio,
                                        const CagdRType CenterWeightSizes[3],
                                        char ** const Error)
```

FrameSize: The thickness of the outer frame.

FrameHGap: Optional gap to add for the left/right frame.

WeightRodsThickness: The thickness of the rods of the middle weight.

WeightRodsSpring: Zero for a straight joining rod, positive for a springy joint (larger value, larger spring).

WeightRodsAngle: The angle to place the rods that holds the weight, in degrees.

FlexClipRatio: To clip in W that is assumed [0, 1]. Can be between zero and 1/2, to clip at ClipRatio and 1.0-ClipRatio.

CenterWeightSizes: The XYZ sizes to use for the central weight.

Error: To be updated with a string error if had one, or NULL to send errors to stderr.

Returns: A list of two lists of trivariates, first of rigid materials and the second of flexible (rods).

Description: Constructs a 2D tile (with some thickness) with a bistable state.
See also: UserMicroBifurcate1to2Tile, UserMicroZSpringTile, UserMicro3DCrossTile, , UserMicroDiagTile1, UserMicroBiStableTile3D, UserMicroBiStableCollapse2D,

15.2.229 UserMicroBiStableTile3D (micro5tile.c:1987)

```
IPObjectStruct *UserMicroBiStableTile3D(CagdRType FrameSize,
                                        CagdRType WeightRodsThickness,
                                        CagdRType WeightRodsSpring,
                                        CagdRType WeightRodsAngle,
                                        CagdRType FlexClipRatio,
                                        const CagdRType CenterWeightSizes[3],
                                        char ** const Error)
```

FrameSize: The thickness of the outer frame.

WeightRodsThickness: The thickness of the rods of the middle weight.

WeightRodsSpring: Zero for a straight joining rod, positive for a springy joint (larger value, larger spring).

WeightRodsAngle: The angle to place the rods that holds the weight, in degrees.

FlexClipRatio: To clip in W that is assumed [0, 1]. Can be between zero and 1/2, to clip at ClipRatio and 1.0-ClipRatio.

CenterWeightSizes: The XYZ sizes to use for the central weight.

Error: To be updated with a string error if had one, or NULL to send errors to stderr.

Returns: A list of two lists of trivariates, first of rigid materials and the second of flexible (rods).

Description: Constructs a 3D tile with a bistable state.
See also: UserMicroBifurcate1to2Tile, UserMicroZSpringTile, UserMicro3DCrossTile, , UserMicroDiagTile1, UserMicroBiStableTile2D, UserMicroBiStableCollapse2D,

15.2.230 UserMicroBifurcate1to2Tile (micro2tile.c:462)

```
IPObjectStruct *UserMicroBifurcate1to2Tile(
                                const UserMicroTileBndryPrmStruct *WMinPrms,
                                const UserMicroTileBndryPrmStruct *WMax1Prms,
                                const UserMicroTileBndryPrmStruct *WMax2Prms,
                                CagdRType SeparationGap,
                                CagdRType SaddleSize,
                                CagdBType Trivars,
                                char ** const Error)
```

WMinPrms: Main, base bottom, boundary parameters.

WMax1Prms, WMax2Prms: Split, top, two boundary parameters.

SeparationGap: Between the two centers of the WMax1/2 boundaries.

SaddleSize: Approximately desired saddle gap. > 0.

Trivars: TRUE to generate trivariates, FALSE for surfaces.

Error: An optional string to return the errors, inf any. Can be NULL to print errors to stderr.

Returns: Constructed tile or NULL if error.

Description: Constructs a 1 to 2 bifurcation tile in the unit cube based on the given boundary parameters. Resulting tile will consist of trivariates.

See also: UserMicro3DCrossTile,

15.2.231 UserMicroBuildHelicalCrv (micro5tile.c:2331)

```
CagdCrvStruct *UserMicroBuildHelicalCrv(const CagdRType *HelicalParams,
                                        int ConicHelix)
```

HelicalParams: Helical parameters as (NumLoops, Rad).

ConicHelix: TRUE for a conical helix, FALSE for a regular helix.

Returns: Created helix.

Description: Constructs a helical curve of designated parameters (NumLoops, Rad) from (0.5, 0.5, 0.0) to (0.5, 0.5, 1.0).

See also: UserMicroBiStblColps2DBuildHelix,

15.2.232 UserMicroCloseTrimCurves (micro3strct.c:496)

```
int UserMicroCloseTrimCurves(UserMicroTilePreprocessStruct *UserMicroData,
                             const CagdPType Min,
                             const CagdPType Max)
```

UserMicroData: Preprocessing data, contains open trim curves.

Min: Min bounds of the UV domain.

Max: Max bounds of the UV domain.

Returns: TRUE if successful, FALSE in case of error.

Description: Closes all the open trim-curves on the boundaries of the tile.

15.2.233 UserMicroComputeAffineTrans (micro0strct.c:50)

```
void UserMicroComputeAffineTrans(const TrivTVStruct *TV,
                                 CagdPType Par0,
                                 CagdPType Par1,
                                 CagdPType Par2,
                                 IrtHmgnMatType MatXInv,
                                 CagdRType VertScale,
                                 IrtHmgnMatType Trans)
```

TV: Trivariate to obtain the four image points from.

Par0: Parameter for TV to obtain a point from.

Par1: Parameter for TV to obtain a point from.

Par2: Parameter for TV to obtain a point from.

MatXInv: Inverse of matrix formed from four source points in R^3.

VertScale: Scaling factor for the affine transform in vertical direction.

Trans: The computed affine transform. This is a return argument.

Returns: void

Description: Computes an affine transform from R^3 to R^3 using four points in R^3 and their images under the transform.

15.2.234 UserMicroDiagTile1 (micro5tile.c:161)

```
IPObjectStruct *UserMicroDiagTile1(CagdRType CntrSize,
                                   const CagdRType CrnrSizes[10],
                                   CagdRType CrnrVertScl,
                                   CagdRType SmoothFactor,
                                   const CagdRType Skin[2],
                                   char ** const Error)
```

CntrSize: The size of the center join of the tile.

CrnrSizes: The sizes of the edges at the eight corners. From XYZMin, to XMax, ..., XYZMax. Zero to disable. Then, two final non zero values will signal a need to also add central vertical up and vertical down rods. For the first eight corners, if negative, the joint will be from the side and not from above.

CrnrVertScl: Vertical Scale of corner Boxes, zero to completely eliminate the corner boxes. m

SmoothFactor: Smoothing behavior of tile. Zero to make C^0 continuous.

Skin: Pair of numeric values to control the option of skin in Y/ZMin and/or Y/ZMax. If either is positive a frame is created, and if either is negative, a full skin is created. Zero to disable. The skin will be for ZMin/ZMax if all CrnrSizes are positive and will be for YMin/YMax if all CrnrSizes are negative.

Error: To be updated with a string error if had one, or NULL to send errors to stderr.

Returns: Constructed tile or NULL if error.

Description: Constructs a 3D tile with diagonal edges.
See also: UserMicroBifurcate1to2Tile, UserMicroZSpringTile, UserMicro3DCrossTile, , UserMicroBiStableTile2D,

15.2.235 UserMicroEvalAlphaCoeffs (micro0strct.c:262)

```
void UserMicroEvalAlphaCoeffs(const IPObjectStruct *TileGeom,
                              UserMicroTilePreprocessStruct *UserMicroData)
```

TileGeom: Geometry of the tile to preprocess.

UserMicroData: Preprocessing data.

Returns: void

Description: Computes the refinement matrices for trivariate tiles with split.

15.2.236 UserMicroFunctionalEvaluateEucl (micro2strct.c:4710)

```
CagdBType UserMicroFunctionalEvaluateEucl(UserMicro2TilingStructPtr Tiling,
                                          const CagdRType *EuclideanPnt,
                                          CagdRType *ResValue)
```

Tiling: The tiling.

EuclideanPnt: The given point in Euclidean space. each dimension in a single tile.

ResValue: Output parameter. The value of the tiling function at the given point.

Returns: TRUE if successful, FALSE otherwise.

Description: Computes the value of the tiling function for a given point in the Euclidean space in a given multi-variate.

15.2.237 UserMicroFunctionalEvaluateUV (micro2strct.c:4749)

```
CagdBType UserMicroFunctionalEvaluateUV(UserMicro2TilingStructPtr Tiling,
                                        const CagdRType *UVPnt,
                                        CagdRType *ResValue)
```

Tiling: The tiling.

UVPnt: The given point in parametric space, normalized to [0, 1]^n. Each dimension in a single tile.

ResValue: Output parameter. The value of the tiling function at the given point.

Returns: TRUE if successful, FALSE otherwise.

Description: Computes the value of the tiling function for a given point in the parametric space in a given multi-variate.

15.2.238 UserMicroFunctionalFreeTiling (micro2strct.c:4946)

```
void UserMicroFunctionalFreeTiling(UserMicro2TilingStructPtr Tiling)
```

Tiling: The tiling to deallocate.

Returns: void

Description: Deallocates a tiling structure.

15.2.239 UserMicroFunctionalRandomTiling (micro2strct.c:2075)

```
UserMicro2TilingStructPtr UserMicroFunctionalRandomTiling(
                                            const MvarMVStruct *DeformMV,
                                            const int *NumCells,
                                            const int *Orders,
                                            const int *NumCPInTile,
                                            CagdRType MinCPValue,
                                            CagdRType MaxCPValue,
                                            CagdRType Capping,
                                            int CappingBits,
                                            CagdBType IsC1,
                                            CagdBType UseConnectivityGraph)
```

DeformMV: The deformation function.

NumCells: Array of size Dim, holds the number of tiles in each dimension.

Orders: Array of size Dim, holds the order of each B-spline multivariate in each tile.

NumCPInTile: Array of size Dim, holds the number of control points in each dimension in a single tile.

MinCPValue: Minimal possible value of the control points.

MaxCPValue: Maximal possible value of the control points.

Capping: Size of the tiling border.

CappingBits: Bitwise attributes of which sides to cap.

IsC1: If True the result is C^1 continuous, otherwise C^0.

UseConnectivityGraph: If TRUE, a connectivity graph algorithm is used to ensure inter- and intra-cell connectivity of the tiles.

Returns: New random tiling structure.

Description: Creates a new random tiling structure.

15.2.240 UserMicroFunctionalTiling (micro2strct.c:4619)

```
UserMicro2TilingStructPtr UserMicroFunctionalTiling(
                                const MvarMVStruct *DeformMV,
                                const int *NumCells,
                                const int *Orders,
                                const int *NumCPInTile,
                                CagdRType MinCPValue,
                                CagdRType MaxCPValue,
                                CagdRType Capping,
                                int CappingBits,
                                CagdBType IsC1,
                                CagdBType UseConnectivityGraph,
                                UserMicroFunctionalTileCBFuncType CPValueFunc)
```

DeformMV: The deformation function.

NumCells: Array of size Dim, holds the number of tiles in each dimension.

Orders: Array of size Dim, holds the order of each B-spline multivariate in each tile.

NumCPInTile: Array of size Dim, holds the number of control points in each dimension in a single tile.

MinCPValue: Minimal possible value of the control points.

MaxCPValue: Maximal possible value of the control points.

Capping: Size of the tiling border.

CappingBits: Bitwise attributes of which sides to cap.

IsC1: If True the result is C^1 continuous, otherwise C^0.

UseConnectivityGraph: Ensure connectivity between tiles, and only one component globally.

CPValueFunc: Control point values function.

Returns: New tiling structure.

Description: Creates a new tiling structure. The tiling behavior is controlled by a given control points values function. '

15.2.241 UserMicroFunctionalTilingIsoSurface (micro2strct.c:4809)

```
IPObjectStruct *UserMicroFunctionalTilingIsoSurface(
                                          UserMicro2TilingStructPtr Tiling,
                                          int SamplingFactor)
```

Tiling: The tiling.

SamplingFactor: The fineness of the returned polygons. Higher values will result in a smoother result, but with larger size. parametric space with the tiling.

Returns: List of polygons of the zero set of the tiling.

Description: Computes the zero set of the tiling implicit function, applying the marching cubes method for each multivariate in each tile. If a deformation multivariate is provided, then the zero set of the tiling is composed into it. If not provided (NULL) the tiling zero set is returned as list of polygons.

15.2.242 UserMicroFunctionalTilingVolume (micro2strct.c:5004)

```
CagdRType UserMicroFunctionalTilingVolume(UserMicro2TilingStructPtr Tiling,
                                          CagdRType CubeSize,
                                          CagdBType PositiveVol)
```

Tiling: The tiling.

CubeSize: The size of the edge of the grid's cube.

PositiveVol: If TRUE, the volume of the positive values domain of the tiling is computed, otherwise the volume of the negative values domain is computed.

Returns: The volume.

Description: Computes the volume enclosed inside/outside the zero set of the tiling. The tiling domain is partitioned in each into a multidimensional grid, composed of cubes. The volume is computed as the volume of the cubes whose centers are inside/outside the zero set of the tiling.

15.2.243 UserMicroGenAMSupport (micro_support.c:2277)

```
IPObjectStruct *UserMicroGenAMSupport(
                               const IPObjectStruct *PlObj,
                               const UserMicroGenAMSupportParamStruct *Params)
```

PlObj: A polygonal object to build support for.

Params: The support parameters.

Returns: The built support as a microstructure.

Description: Build support as a microstructure, to a given polygonal model.
See also:

15.2.244 UserMicroGenShellCapForThisTile (micro3strct.c:157)

```
IPObjectStruct *UserMicroGenShellCapForThisTile(
                                IPObjectStruct *TileGeom,
                                const int *TotalRepeats,
                                const int *TileIdxs,
                                int BndryBits,
                                int ShellCapBits,
                                UserMicroTilePreprocessStruct *UserMicroData)
```

TileGeom: Tile to generate shell for.

TotalRepeats: Repetitions along the different axes.

TileIdxs: The current repetition identifier along u, v, w axes.

BndryBits: Specifies where in the original domain, this Bezier patch lies.

ShellCapBits: Specifies which side to apply shelling/capping to.

UserMicroData: Preprocessing data.

Returns: Shell/cap for this tile.

Description: Generate shelling/capping for current tile.

15.2.245 UserMicroGetBndryCrvs (micro3strct.c:250)

```
void UserMicroGetBndryCrvs(const IPObjectStruct *TileGeom,
                           UserMicroTilePreprocessStruct *UserMicroData,
                           const CagdPType MinCoordVals,
                           const CagdPType MaxCoordVals)
```

TileGeom: Geometry of the tile to extract trim curves from.

UserMicroData: Preprocessing data.

MinCoordVals: Min coordinate values for bounding box of tile.

MaxCoordVals: Max coordinate values for bounding box of tile.

Returns: void

Description: Extract the trim curves of given tile.

15.2.246 UserMicroIsInternalBoundaryCP (micro2strct.c:821)

```
int UserMicroIsInternalBoundaryCP(UserMicro2TilingStructPtr T,
                                  const int *CPIndex)
```

T: A tiling.

CPIndex: The control point index.

Returns: 0 - if the given control point lies on the outer boundary of whole tiling or if it is not shared between tiles. 1 - The control point is shared along a face on the x axis. 2 - The control point is shared along a face on the y axis. 3 - The control point is shared along a face on the z axis. 4 - The control point is shared along an edge on the x axis. 5 - The control point is shared along an edge on the y axis. 6 - The control point is shared along an edge on the z axis. 7 - The control point is shared on a corner.

Description: Checks if a given control point lies on the boundary between tiles, and returns an appropriate integer number based on the boundary type.

15.2.247 UserMicroMakeTwistShearTile (micro5tile.c:2914)

```
IPObjectStruct *UserMicroMakeTwistShearTile(CagdRType BaseHeight,
                                            CagdRType RodWidth,
                                            CagdRType RodSpace,
                                            CagdRType XTwistShift,
                                            CagdRType MidRodScl,
                                            CagdRType ElasticFrac,
                                            char ** const Error)
```

BaseHeight: Height (thickness) of bottom/top bases.

RodWidth: Thickness of diagonal rods.

RodSpace: Spacing between parallel diagonal rods.

XTwistShift: Shift in X that controls the diagonal amount in the rods.

MidRodScl: Shrinkage in diagonal rod's thickness. In Eps to 1 range.

ElasticFrac: The faction of the elastic ends in the rods. Between 0.0 and 0.5.

Error: To be updated with a string error if had one, or NULL to send errors to stderr.

Returns: A list of two lists, the first holds the flexible parts in the tile and the second the rigid parts.

Description: Builds a simple diagonal tile, for twisting/shear effects.
See also: UserMicroBifurcate1to2Tile, UserMicroZSpringTile, UserMicro3DCrossTile, , UserMicroDiagTile1, UserMicroBiStableTile2D, UserMicroBiStableTile3D,

15.2.248 UserMicroParseTileFromObj (micro_strct.c:317)

```
UserMicroTileStruct *UserMicroParseTileFromObj(IPObjectStruct *IPObject)
```

IPObject: Input object to parse.

Returns: Parsed tile.

Description: Parses the input object into a tile. A tile can be any IRIT object including lists of objects but if the input is a list object with the first object in the list a number, like: (n, Obj1,..., Objn) N which is used as:

1. If n = 3, Obj1 is used as the 1st tile along w, Obj2 is interior in w, and Obj3 as last tile in w.
2. If n is general, it should match the number of requested element in w, globally and Obji is used for the i'th w elements.

15.2.249 UserMicroPreprocessFractalTile (micro1strct.c:200)

```
void UserMicroPreprocessFractalTile(
                                const IPObjectStruct *TileGeom,
                                UserMicroTilePreprocessStruct *UserMicroData)
```

TileGeom: Geometry of the tile to preprocess.

UserMicroData: Preprocessing data.

Returns: void

Description: Preprocess trivariate tile for fractal-like tiling.

15.2.250 UserMicroPreprocessG0DiscontTile (micro0strct.c:96)

```
void UserMicroPreprocessG0DiscontTile(
                                 const IPObjectStruct *TileGeom,
                                 UserMicroTilePreprocessStruct *UserMicroData)
```

TileGeom: Geometry of the tile to preprocess.

UserMicroData: Preprocessing data.

Returns: void

Description: Preprocess trivariate tile with potential G0-discontinuities.

15.2.251 UserMicroRandomBifurcationTiling (micro2strct.c:4513)

```
UserMicro2TilingStructPtr UserMicroRandomBifurcationTiling(
                                            const MvarMVStruct *DeformMV,
                                            const int *Orders,
                                            const int *NumCPInTile,
                                            CagdRType SubdivTol,
                                            CagdRType RandomFactor,
                                            int CapBits,
                                            CagdRType CappingValue)
```

DeformMV: The deformation multivariate.

Orders: Array of size DeformMV -> Dim, holds the order of each B-spline multivariate in each tile.

NumCPInTile: Array of size DeformMV -> Dim, holds the number of control points in each dimension in a single tile.

SubdivTol: Subdivision threshold.

RandomFactor: Randomization factor [0,1]. Defines a blending between a canonical predefined tile and a random tile.

CapBits: Capping options.

CappingValue: Capping value. The value for the control points on the domain boundary.

Returns: A tiling structure.

Description: Builds a implicit random bifurcation tiling (without composition).

15.2.252 UserMicroReadTileFromFile (micro_strct.c:379)

```
UserMicroTileStruct *UserMicroReadTileFromFile(const char *FileName,
                                               int Messages,
                                               int MoreMessages)
```

FileName: To read the file from.

Messages: Do we want error messages?

MoreMessages: Do we want informative messages?

Returns: Tile read from the file.

Description: Reads a tile from a file.

15.2.253 UserMicroRegImplctAssignValueToUVW (micro5strct.c:1061)

```
int UserMicroRegImplctAssignValueToUVW(
                                const UserMicroRegImplctParamStruct *MRIParam,
                                const CagdPType GlbUVW,
                                const CagdPType TileUVW,
                                int NumImgCoords,
                                CagdRType *ValueBuf)
```

MRIParam: A data structure for regular implicit microstructure.

GlbUVW: UVW coordinates of a point to assign color, in a macro deformation map.

TileUVW: UVW coordinates of a point to assign color, in a tile object space (should be in [0,1]^3).

NumImgCoords: Number of coordinates in color (or other properties)

ValueBuf: Assigned color. RGB is stored in ValueBuf[1..3].

Returns: TRUE if color is assigned properly. FALSE, otherwise.

Description: Compute colors (or materials in higher dimensions) in regular implicit parameters. Color is assigned as follows:

1. If the optional color trivariate is provided, color is assigned using tile coordinates and this trivariate no matter which representation is used to define tile implicit function.
2. If tile is represented by a trivariate, then if tile trivariate is E1 the macro deformation map with global uvw coordinates are used in color assignment. If tile trivariate is E4, tile coordinates are used.
3. If tile has other representations, then macro deformation map is used to assign colors if ColorTV is null.

15.2.254 UserMicroRegImplctGetFacePrmTile (micro5strct.c:552)

```
TrivTVStruct *UserMicroRegImplctGetFacePrmTile(const TrivTVStruct *TV,
                                               int CoordId,
                                               const CagdBBoxStruct *UVWDmn)
```

TV: The macro deformation map that parametrizes face parameter of an implicit cross trivariate tile.

CoordId: The coordinate in TV that parametrizes face parameter.

UVWDmn: The domain of a tile.

Returns: The constructed implicit tile, as a tri-quadratic trivariate of mesh size (5 x 5 x 5).

Description: Construct an implicit cross trivariate given the domain in the macro deformation map TV. Face parameter is parametrized in the CoordId-th dimension of TV. 6 face parameters are evaluated at the center of 6 faces of the tile domain.
See also:

15.2.255 UserMicroRegImplctMarchCube (micro5strct.c:1647)

```
IPObjectStruct *UserMicroRegImplctMarchCube(const IPObjectStruct *PMicroStrct,
                                            IrtRType SamplingFactor,
                                            int SkipFactor,
                                            IrtRType IsoVal)
```

PMicroStrct: A list object containing a regular implicit microstructure specification.

SamplingFactor: Sampling rate control. If a tile is of trivariate type, the trivariate is sampled ULength * VLength * WLength if SamplingFactor is 1. Otherwise, the samplings are (ULength * SamplingFactor) * (VLength * SamplingFactor) * (WLength * SamplingFactor). If other types are used to represent a tile function, the samplings are SamplingFactor * SamplingFactor * SamplingFactor.

SkipFactor: Typically 1. For 2, only every second sample is considered, and for i, only every i'th sample is considered.

IsoVal: At which to extract the iso-surface of a tile. Typically set to 0.

Returns: A list object containing composed polygonal tiles approximated using the marching cubes algorithm.

Description: Computes a polygonal iso surface approximation at iso value IsoVal to given implicit tile function, and composes the polygonal model to a macro deformation map to create the regular implicit structure in a polygonal model.
See also: MarchCubeVolume,

15.2.256 UserMicroRegImplctParamFree (micro5strct.c:991)

```
void UserMicroRegImplctParamFree(UserMicroRegImplctParamStruct *MRIParam)
```

MRIParam: A data structure for regular implicit microstructures.

Returns: void

Description: Free regular implicit microstructure parameters.
See also: UserMicroRegImplctParamNew,

15.2.257 UserMicroRegImplctParamNew (micro5strct.c:863)

```
UserMicroRegImplctParamStruct *UserMicroRegImplctParamNew(
                                            const IPObjectStruct *PMicroStrct)
```

PMicroStrct: A list object containing the regular implicit microstructure parameters.

Returns: A data structure for regular implicit tiling.

Description: Constructs an data structure storing parameters for regular implicit microstructures from a list object. The list object contains (TVMap, ImplicitFuncType, (optionally) ColorTV, TilingSteps) where TVMap defines the macro deformation map, ImplicitFuncType determines the representation of each tile, ColorTV defines colors within a tile, and TilingSteps save a list of vectors of tile repetitions in tile dimensions. ImplicitFuncType can one of the following type: a trivariate object, an integer number, or a string containing expression. If ImplicitFuncType is a numeric object and value is either -1/-2, then parametric trivariate is constructed and used. The tile parameter is refered from the macro deformation map, which must have XYZS or XYZRGBS dimensions, where 'S' parameterize the tile parameter. If ImplicitFuncType is -1, each tile trivariate is calculated on the fly; tile trivariates are precomputed when ImplicitFuncType is -2. When ImplicitFuncType is a nonnegative numeric value, one of the predefined TPMS' is used.
See also: UserMicroRegImplctParamFree,

15.2.258 UserMicroRegularBifurcationTiling (micro2strct.c:4550)

```
IPObjectStruct *UserMicroRegularBifurcationTiling(
                                          const UserMicroParamStruct *Param)
```

Param: The micro structure synthesis parameters (deformation function, tile, tolerances, etc.

Returns: A tiling structure.

Description: Builds a regular bifurcation tiling with composition into a deformation map.

15.2.259 UserMicroSetUniqueGeomIDs (micro_strct.c:1238)

```
void UserMicroSetUniqueGeomIDs(IPObjectStruct *DefObj, int *UniqueGeomID)
```

DefObj: Object (a single tile typically) to update all its leaves with a unique ID.

UniqueGeomID: Reference to the unique geometry ID counter.

Returns: void

Description: Updates all leaves in the given deformed object with unique IDs.

15.2.260 UserMicroSlicerCreateAll (micro_slicer.c:835)

```
TrivTVStruct *UserMicroSlicerCreateAll(const UserMicroSlicerInfoStruct *Slicer)
```

Slicer: The slicer.

Returns: The trivariates created.

Description: recursively create the all microstructure trivariates.
See also:

15.2.261 UserMicroSlicerFree (micro_slicer.c:779)

```
void UserMicroSlicerFree(UserMicroSlicerInfoStruct *Slicer)
```

Slicer: The slicer.

Returns: void

Description: Free a slicer.
See also:

15.2.262 UserMicroSlicerGetBoundarySrfsFromTV (micro_slicer.c:996)

```
CagdSrfStruct *UserMicroSlicerGetBoundarySrfsFromTV(const TrivTVStruct *Trivar)
```

Trivar: The trivariate.

Returns: The trivariate bounding surfaces.

Description: Get the 6 bounding surfaces of a trivariate.
See also:

15.2.263 UserMicroSlicerGetOutline (micro_slicer.c:900)

```
IPPolygonStruct *UserMicroSlicerGetOutline(UserMicroSlicerInfoStruct *Slicer,
                                           CagdRType z)
```

Slicer: The slicer.

z: The slice z level.

Returns: The outline.

Description: Get the outline of the current slice (at the given z).
See also:

15.2.264 UserMicroSlicerInit (micro_slicer.c:701)

```
UserMicroSlicerInfoStruct *UserMicroSlicerInit(
                                        const UserMicroParamStruct *MSParam,
                                        int Levels)
```

MSParam: Default microstructure parameters to use.

Levels: The number of levels in the microstructure.

Returns: The slicer handle.

Description: Initialize a micro struct slicer.
See also:

15.2.265 UserMicroSlicerSetLevel (micro_slicer.c:755)

```
void UserMicroSlicerSetLevel(UserMicroSlicerInfoStruct *Slicer,
                    const UserMicroParamStruct *MSParam,
                    int Level)
```

Slicer: The slicer.

MSParam: The microstructure parameters to use.

Level: The level to update.

Returns: void

Description: set the micro structure parameter at a particular level (0 based).
See also:

15.2.266 UserMicroSlicerSetTolerances (micro_slicer.c:810)

```
void UserMicroSlicerSetTolerances(UserMicroSlicerInfoStruct *Slicer,
                                  CagdRType NumericTol,
                                  CagdRType SubdivTol,
                                  CagdRType TraceTol,
                                  CagdRType SlopeTol)
```

Slicer: The slicer.

NumericTol: The numeric tolerance of the slicer.

SubdivTol: The subdivision tolerance of the slicer.

TraceTol: The tracing tolerance of the slicer.

SlopeTol: The minimal polyline angle (radians) between line segments.

Returns: void

Description: Set the slicer tolerances.
See also:

15.2.267 UserMicroStitchBndryCrvs (micro3strct.c:432)

```
void UserMicroStitchBndryCrvs(UserMicroTilePreprocessStruct *UserMicroData)
```

UserMicroData: Preprocessing data.

Returns: void

Description: Stitches the boundary curves of the tile.

15.2.268 UserMicroStructComposition (micro_strct.c:2668)

```
IPObjectStruct *UserMicroStructComposition(UserMicroParamStruct *Param)
```

Param: Input parameters (See struct definition).

Returns: The deformed tiling.

Description: The unified interface for creating micro structures which may be regular, random or functional.
See also: UserMicroStructComposition2,

15.2.269 UserMicroStructComposition2 (micro_strct.c:2736)

```
IPObjectStruct *UserMicroStructComposition2(UserMicroParamStruct *Param,
                                            const IPObjectStruct *DeformMVs)
```

Param: Input parameters (See struct definition).

DeformMVs: A hierarchy of deformation maps in which case, the micro structures are constructed for them all.

Returns: The deformed tilings over all deformation MVs.

Description: The unified interface for creating micro structures which may be regular, random or functional.
See also: UserMicroStructComposition,

15.2.270 UserMicroStructParamFree (micro_strct.c:3033)

```
void UserMicroStructParamFree(UserMicroParamStruct *MSParam, int FreeSelf)
```

MSParam: Structure to free. It is freed as well, if FreeSelf.

FreeSelf: TRUE to also free the memory associated with Param.

Returns: void

Description: Frees the given structure of MSParam.
See also: UserMicroParamStruct,

15.2.271 UserMicroTileCopy (micro_strct.c:241)

```
UserMicroTileStruct *UserMicroTileCopy(const UserMicroTileStruct *Tile)
```

Tile: The tile to copy.

Returns: Newly created copy.

Description: Creates a copy of a tile.

15.2.272 UserMicroTileCopyList (micro_strct.c:273)

```
UserMicroTileStruct *UserMicroTileCopyList(const UserMicroTileStruct *Tile)
```

Tile: The list of tiles to copy.

Returns: Newly created copy.

Description: Creates a copy of a list of tiles.

15.2.273 UserMicroTileFindNeighbors (micro4strct.c:310)

```
IPObjectStruct **UserMicroTileFindNeighbors(const IPObjectStruct *Tile,
                                            const IPObjectStruct *TileList,
                                            const MvarMVStruct *DeformMV,
                                            const CagdBType *DeformMVPeriodic,
                                            IPObjectStruct **Neighbors)
```

Tile: The query tile to search neighbors.

TileList: A list of tiles to search neighbors of Tile.

DeformMV: Deformation map of macro shape.

DeformMVPeriodic: Periodicity of the deformation map. A vector in the dimension of DeformMV.

Neighbors: A pointer to a vector of six neighbor tile In otehr wordes, the Neighboring tiles adjacent to Tile, in TIleList, in (order of) UMin, UMax, VMin, VMax, WMin, WMax of Tile.

Returns: A pointer to a vector of (up to) six neighbor tiles. tiles adjacent to Tile in (order in) TileList in dir UMin, UMax, VMin, VMax, WMin, WMax of Tiles. Note some of the six neighbors might be NULL if on the boundary. Returns input Neighbors vector, or NULL if error.

Description: Find six neighbor tiles of the given tile, Tile, in a 3D deformation (a trivariate) deformMV.

15.2.274 UserMicroTileFree (micro_strct.c:182)

```
void UserMicroTileFree(UserMicroTileStruct *Tile)
```

Tile: The tile to free.

Returns: void

Description: Frees a tile.

15.2.275 UserMicroTileFreeList (micro_strct.c:211)

```
void UserMicroTileFreeList(UserMicroTileStruct *Tile)
```

Tile: The tile list to free.

Returns: void

Description: Frees a list of tiles.

15.2.276 UserMicroTileNew (micro_strct.c:155)

```
UserMicroTileStruct *UserMicroTileNew(IPObjectStruct *Geom)
```

Geom: Geometry for the tile, used in place. Assumed in [0,1]^3.

Returns: The allocated tile.

Description: Allocates memory for a tile.

15.2.277 UserMicroTileTransform (micro_strct.c:492)

```
UserMicroTileStruct *UserMicroTileTransform(const UserMicroTileStruct *Tile,
                                            IrtHmgnMatType Mat)
```

Tile: The tile to transform.

Mat: The transformation matrix.

Returns: Transformed tile.

Description: Transforms a tile according to the transformation matrix.

15.2.278 UserMicroZSpringTile (micro4tile.c:234)

```
IPObjectStruct *UserMicroZSpringTile(int SpringOrder,
                                     const CagdRType *SpringParams,
                                     const CagdRType *BotTopCrossWidth,
                                     CagdRType BotTopCrossHeight)
```

SpringOrder: The order of the spring can be 2 or 3 for linear or round spring edges, respectively.

SpringParams: A triplet of (SpringWidth, SpringTrans, SpringScale). The SpringWidth is the thickness of the vertical springs. SpringTrans and SpringScale control the extend and scale of the middle zone of the springs.

BotTopCrossWidth: If positive, width of the four cross bottom/top cross arms in (UMin, UMax, VMin, VMax). Zero or negative to disable.

BotTopCrossHeight: If positive, height of the bottom/top cross to add. Zero or negative to disable.

Returns: Constructed tile or NULL if error.

Description: Constructs a 3D spring in Z tile in the unit cube based on the given parameters. Resulting tile will consist of trivariates.
See also: UserMicroBifurcate1to2Tile, UserMicro3DCrossTile,

15.2.279 UserMinDistLineBBox (userpick.c:38)

```
IrtRType UserMinDistLineBBox(const IrtPtType LinePos,
                             const IrtVecType LineDir,
                             const IrtBboxType BBox)
```

LinePos: Point on the 3-space line.

LineDir: Direction of the 3-space line.

BBox: Bounding box, parallel to main planes.

Returns: Minimal distance between the line and the bbox.

Description: Computes the closest distance between a 3-space line and a bounding box.
See also: UserMinDistLinePolylineList, UserMinDistLinePolygonList,

15.2.280 UserMinDistLinePolygonList (userpick.c:209)

```
IrtRType UserMinDistLinePolygonList(const IrtPtType LinePos,
                                    const IrtVecType LineDir,
                                    IPPolygonStruct *Pls,
                                    IPPolygonStruct **MinPl,
                                    IrtPtType MinPt,
                                    IrtRType *HitDepth,
                                    IrtRType *IndexFrac)
```

LinePos: Point on the 3-space line.

LineDir: Direction of the 3-space line.

Pls: List of polygons.

MinPl: Will be set to the closest polyline.

MinPt: Will be set to the closest point.

HitDepth: If a direct hit (zero distance is returned), the depth of the hit is returned here, as the value of parameter t in LinePos + t * LineDir.

IndexFrac: Will be set to the index of the closest point, starting from zero. Index of 3.5 means the closest is in the mid point from point 3 to point 4. This value should be ignored for a direct hit (return value equal zero). Valid for one polygon in input.

Returns: Minimal distance from the 3-space line to the polygons. In case of a direct hit, zero is returned.

Description: Compute the closest distance between a line in three space and a list of polygons. The polygons are assumed to be convex.
See also: UserMinDistLineBBox, UserMinDistLinePolylineList,

15.2.281 UserMinDistLinePolylineList (userpick.c:279)

```
IrtRType UserMinDistLinePolylineList(const IrtPtType LinePos,
                                     const IrtVecType LineDir,
                                     IPPolygonStruct *Pls,
                                     int PolyClosed,
                                     IPPolygonStruct **MinPl,
                                     IrtPtType MinPt,
                                     IrtRType *HitDepth,
                                     IrtRType *IndexFrac)
```

LinePos: Point on the 3-space line.

LineDir: Direction of the 3-space line.

Pls: List of polylines.

PolyClosed: TRUE if polyline is a closed polygon, FALSE otherwise.

MinPl: Will be set to the closest polyline.

MinPt: Will be set to the closest point.

HitDepth: If a hit, the depth of the hit is returned here, as the value of parameter t in LinePos + t * LineDir.

IndexFrac: Will be set to the index of the closest point, starting from zero. Index of 3.5 means the closest is in the mid point from point 3 to point 4. Valid for one polyline in input.

Returns: Minimal distance from the 3-space line to the polylines.

Description: Compute the closest distance between a line in three space and a list of polyline.
See also: UserMinDistLineBBox, UserMinDistLinePolygonList, , UserMinDistPointPolylineList,

15.2.282 UserMinDistPointPolylineList (userpick.c:400)

```
IrtRType UserMinDistPointPolylineList(const IrtPtType Pt,
                                      IPPolygonStruct *Pls,
                                      IPPolygonStruct **MinPl,
                                      IPVertexStruct **MinV,
                                      int *Index)
```

Pt: Point in 3-space.

Pls: List of polylines.

MinPl: Will be set to the closest poly.

MinV: Will be set to the closest vertex.

Index: Will be set to the index of the closest vertex, starting from zero. Can be NULL to ignore.

Returns: Minimal distance from the 3-space point to the polys.

Description: Compute the closest distance between a point in three space and list of polys. Only vertices in the polylines are considered.
See also: UserMinDistLineBBox, UserMinDistLinePolygonList, , GMDistPolyPt, GMDistPolyPt2,

15.2.283 UserMoldReliefAngle2Srf (visible.c:1040)

```
TrimSrfStruct *UserMoldReliefAngle2Srf(const CagdSrfStruct *Srf,
                                       const CagdVType VDir,
                                       CagdRType Theta,
                                       int MoreThanTheta,
                                       CagdRType SubdivTol)
```

Srf: Surface to examine its normals' angular deviation.

VDir: View direction vector.

Theta: Angular deviation to examine, in degrees, in range (0, 90).

MoreThanTheta: TRUE to seek regions of more than theta degrees normal deviation from VDir, FALSE for less than.

SubdivTol: Subdivision tolerance of freeforms silhouette approx.

Returns: List of trimmed surfaces' regions of regions of Srf that present angular deviations of less (more) than Theta to VDir.

Description: Compute and trim regions in Srf that have normals with angular deviation of more (less) than Theta Degrees from the prescribed direction VDir. Computation is based on extractions of Isoclines of Srf from VDir.
See also: SymbSrfIsocline, UserMoldRuledRelief2Srf,

15.2.284 UserMoldRuledRelief2Srf (visible.c:1104)

```
CagdSrfStruct *UserMoldRuledRelief2Srf(const CagdSrfStruct *Srf,
                                       const CagdVType VDir,
                                       CagdRType Theta,
                                       CagdRType SubdivTol)
```

Srf: Surface to examine its normals' angular deviation.

VDir: View direction vector.

Theta: Angular deviation to examine, in degrees, in range (0, 90).

SubdivTol: Subdivision tolerance of freeforms silhouette approx.

Returns: List of ruled surfaces that tangently extends from the computed isoclines at the prescribed VDir/Theta.

Description: Compute and add ruled surfaces at all trimmed relief angles' boundaries. Computation is based on extractions of Isoclines of Srf from VDir.
See also: SymbSrfIsocline, UserMoldReliefAngle2Srf,

15.2.285 UserNCContourToolPath (nc_tpath.c:175)

```
IPObjectStruct *UserNCContourToolPath(const IPObjectStruct *PObj,
                                      IrtRType Offset,
                                      IrtRType ZBaseLevel,
                                      IrtRType TPathSpace,
                                      UserNCGCodeUnitType Units)
```

PObj: Object to process and create 3-axis machining tool path for.

Offset: Tool radius to offset the geometry in PObj with.

ZBaseLevel: Bottom level to machine. Created tool path will never be below this level.

TPathSpace: Space between adjacent paths. Also used as a tolerance bound.

Units: Millimeters or inches.

Returns: List of constant X polylines, in zigzag motion, that covers PObj from above (offseted by Offset).

Description: Computes tool path to 3-axis machine from +Z direction the given geometry (Polygonal meshes/surfaces). Assumes a ball end tool. Stages:

1. Tessellate the model into polygons.
2. Verify the existence of (or create approximation thereof) of normal at the vertices of the polygons, & offset all polygons by desired offset.
3. For each Y-section, contour the model at the desired Y-constant level and compute the upper envelop of all contoured line segments as this Y-constant level as one toolpath.
4. Emit the toolpath flipping every second Y-constant path to create a zigzag motion, and move the tool path down by offset amount so it will be referencing the ball-end tool's bottom.

See also: UserNCPocketToolPath,

15.2.286 UserNCPocketToolPath (nc_tpath.c:997)

```
IPObjectStruct *UserNCPocketToolPath(const IPObjectStruct *PObj,
                                     IrtRType ToolRadius,
                                     IrtRType RoughOffset,
                                     IrtRType TPathSpace,
                                     IrtRType TPathJoin,
                                     UserNCGCodeUnitType Units,
                                     int TrimSelfInters)
```

PObj: Object to process and create pocket machining tool path for.

ToolRadius: Tool radius to offset the geometry in PObj with.

RoughOffset: Offset amount to use in the roughing stage.

TPathSpace: Space between adjacent parallel cut rows. If zero returns just the offset geometry.

TPathJoin: maximal distance between end points of tpath to join. <

Units: Milimeters or inches.

TrimSelfInters: TRUE to try and trim self intersections.

Returns: List of constant X polylines, in zigzag motion, that covers PObj from above (offseted by Offset).

Description: Computes tool path to 2D pocket machining from +Z direction the given geometry (curves/polylines). Geometry is assumed to be closed, possibly with closed islands. Stages:

1. Offset the shape in for roughing, by RoughOffset (+ToolRadius) amount.
2. Compute the contour lines (intersection of parallel lines) with the offseted-in geometry.
3. Merge all contour lines into one large toolpath. Toolpath is sorted so adjacent paths are close within tool diameter, as much as possible to minimize retractions.
4. Add a final, finish, path, at ToolRadius offset from the shape.

See also: UserNCContourToolPath,

15.2.287 UserPackTileCreateRectangularTile (tilepack.c:283)

```
IPObjectStruct *UserPackTileCreateRectangularTile(
                                                const IPObjectStruct *TileObj,
                                                CagdBType Normalize)
```

TileObj: A pointer to a tile represented as an IPObjectStruct. Tile can optionally have the tiling translation vectors, as "vec1", "vec2", and "vec3", defaulting to X, Y, Z if none. Tile can be also a list object of tiles in which case each element in list must have tile position attribute as "TileBndryType" - interior tile or one of the Min/Max U/V/W boundaries (See UserPackTileBndryType). An interior tile will be used instead of missing other tile(s).

Normalize: If TRUE, normalizes extracted rectangle to unit square/cube, otherwise preserves its aspect ratio and size.

Returns: A pointer to the basic rectangular tile.

Description: Creates a rectangular repeatable macro-tile object to be used in order to tile the domain from an already existing tile represented as an IPObjectStruct.
See also: UserPackTilesInDomain, UserPackTileFreeTileObject,

15.2.288 UserPackTileCreateTileObject (tilepack.c:125)

```
struct UserTilePackInfoStruct *UserPackTileCreateTileObject(
                                                const IPObjectStruct *TileObj,
                                                const int *StepsMin,
                                                const int *StepsMax,
                                                int Dim)
```

TileObj: A pointer to a tile represented as an IPObjectStruct. Tile can optionally have the tiling translation vectors, as "vec1", "vec2", and "vec3", defaulting to X, Y, Z if none. Tile can be also a list object of tiles in which case each element in list must have tile position attribute as "TileBndryType" - interior tile or one of the Min/Max U/V/W boundaries (See UserPackTileBndryType). An interior tile will be used instead of missing other tile(s).

StepsMin, StepsMax: Optional explicit number of steps, in vec1/2/3. Can be NULL, in which case, covering steps for the given domain, will be estimated.

Dim: Dimension of tile - 2 or 3.

Returns: The object representing a tile for the tiling procedure - UserPackTilesInDomain, or NULL if error.

Description: Creates a tile object to be used in order to tile the domain from an already existing tile represented as an IPObjectStruct. This object have to be freed using the UserPackTileFreeTileObject function.
See also: UserPackTilesInDomain, UserPackTileFreeTileObject,

15.2.289 UserPackTileFreeTileObject (tilepack.c:483)

```
void UserPackTileFreeTileObject(struct UserTilePackInfoStruct *Tile)
```

Tile: A pointer to a tile represented as an UserTilePackInfoStruct.

Returns: void

Description: Frees a tile object created using UserPackTileCreateTileObject.
See also: UserPackTileCreateTileObject, UserPackTilesInDomain,

15.2.290 UserPackTilesFilterRect (tilepack.c:1041)

```
IPObjectStruct *UserPackTilesFilterRect(const IPObjectStruct *Tiles,
                                        const GMBBBboxStruct *Domain)
```

Tiles: A list of tiles that together cover beyond the Domain.

Domain: The rectangular domain that is a periodic tiles for this tiling type.

Returns: A rectangular tiles as a subset of the input, filtered.

Description: Filters out all tiles that are outside the given bbox domain. Tiles that cross the domain are split in the middle and the halves are filtered again. The end result is a rectangle domain that periodically tiles 2-space.
See also: UserPackTilesInDomain,

15.2.291 UserPackTilesInDomain (tilepack.c:534)

```
IPObjectStruct *UserPackTilesInDomain(UserTilePackInfoStruct *PackTileInfo,
                                      const GMBBBboxStruct *TilingDomain,
                                      int TileInclusion,
                                      CagdRType SkewFactor,
                                      CagdBType MergeNeighbors)
```

PackTileInfo: A pointer to the tile to use for periodically tiling the domain. The tile object also defines the directions to to translate, during the tiling, via "veci" attributes. Will be updated with the number of steps in veci.

TilingDomain: A pointer to the bounding box of the to be tiled domain.

TileInclusion: 0 to exclude all tiles intersecting the domain boundary or outside. 1 to include tiles that are partially outside domain. 2 to assume tile is [0, 1]^k and include all [0,1]^k that are in the domain. 3 do not clipping whatsoever. 4 include intersecting tiles if centroid is inside (for periodicity).

SkewFactor: Optional skewing factor, relevant only if tile has skew direction attribute.

MergeNeighbors: If TRUE, mark the neighbors tiles with the same ID, and the same RGB. Relevant only for rectangular tiles. At least two steps in Y are needed. M *

Returns: A pointer to all tiles packed in the tiled domain.

Description: Tiles a domain periodically.
See also: UserPackTileCreateTileObject, UserPackTileFreeTileObject, , UserPackTilesFilterRect,

15.2.292 UserPackTilesSetIDAttrib (tilepack.c:812)

```
int UserPackTilesSetIDAttrib(const IPObjectStruct *Tiles, int *Steps)
```

Tiles: A list of tiles packed in the tiled domain.

Steps: Number of steps, in vec1/2/3 which were needed for the given domain.

Returns: Maximum ID used in this assignment.

Description: Set unique IDs attribute values to the children leaves of the given object Tile, assign the neighbors tiles with the same ID.
See also: UserPackTilesSetIDAttrib,

15.2.293 UserPackTilesSetRGBAttrib (tilepack.c:957)

```
void UserPackTilesSetRGBAttrib(const IPObjectStruct *Tiles,
                               const IrtBType **RGB)
```

Tiles: A list of tiles packed in the tiled domain.

RGB: Random RGB array.

Returns: void

Description: Set random RGB attribute values to the children leaves of the given object Tile, assign the neighbors tiles with the same RGB.
See also: UserPackTilesSetRGBAttrib,

15.2.294 UserPatchAccessFree (patch_access.c:754)

```
void UserPatchAccessFree(UserPatchAccessInfoStruct *Patches)
```

Patches: The Patches accessibility structure.

Returns: void

Description: Frees the structure used for patch accessibility calculations.
See also:

15.2.295 UserPatchAccessGetNumOfPatches (patch_access.c:902)

```
int UserPatchAccessGetNumOfPatches(const UserPatchAccessInfoStruct *Patches)
```

Patches: The Patch access structure.

Returns: Number of patches.

Description: Get the number of patches.
See also:

15.2.296 UserPatchAccessGetNumOfSrfs (patch_access.c:857)

```
int UserPatchAccessGetNumOfSrfs(const UserPatchAccessInfoStruct *Patches)
```

Patches: The Patch access structure.

Returns: Number of surfaces.

Description: Get the number of surfaces.
See also:

15.2.297 UserPatchAccessGetPatchData (patch_access.c:950)

```
void UserPatchAccessGetPatchData(const UserPatchAccessInfoStruct *Patches,
                                 int PatchId,
                                 UserPatchAccessPatchDataStruct *Data)
```

Patches: The Patch access structure.

PatchId: The ID of the patch.

Data: The patch data returned.

Returns: void

Description: Get a the accessibility information data for a patch, given its ID.
See also:

15.2.298 UserPatchAccessGetPatchVisible (patch_access.c:925)

```
int UserPatchAccessGetPatchVisible(const UserPatchAccessInfoStruct *Patches,
                                   int PatchId)
```

Patches: The Patch access structure.

PatchId: The id of the patch.

Returns: Non-zero for visible patches.

Description: Find out if a patch is visible from the current direction.
See also:

15.2.299 UserPatchAccessGetSrf (patch_access.c:881)

```
const CagdSrfStruct *UserPatchAccessGetSrf(const UserPatchAccessInfoStruct
                                                                  *Patches,
                                           int SrfId)
```

Patches: The Patch access structure.

SrfId: The surface id.

Returns: The surface.

Description: Get the surface with the given id.
See also:

15.2.300 UserPatchAccessPrep (patch_access.c:670)

```
UserPatchAccessInfoStruct *UserPatchAccessPrep(
                              CagdSrfStruct **const Srfs,
                              const UserPatchAccessSrfParamsStruct *SrfParams,
                              int SrfNum)
```

Srfs: The Surfaces.

SrfParams: The size and cone paramters used to adaptively subdivide the surfaces to patches.

SrfNum: The number of surfaces.

Returns: The patch accessibility struct.

Description: Prepares the structure used for patch accessibility calculations.
See also:

15.2.301 UserPatchAccessSetDir (patch_access.c:818)

```
void UserPatchAccessSetDir(UserPatchAccessInfoStruct *Patches,
                           const IrtRType *Dir,
                           IrtRType AccessAngle,
                           IrtRType ExtraRadius)
```

Patches: The Patch access structure.

Dir: The 'up' direction (the view direction is the opposite).

AccessAngle: An extra slack Angle (maintaining visibility with small, up to Angle deviation around Dir).

ExtraRadius: Additional translational slack, treating the Dir vector as a cylinder of ExtraRadius radius.

Returns: void

Description: Sets the current direction information for the patch access structure. This function also performs the accessibility test from this direction.
See also:

15.2.302 UserPatchAccessSetPatchTest (patch_access.c:1011)

```
void UserPatchAccessSetPatchTest(const UserPatchAccessInfoStruct *Patches,
                                 int PatchId,
                                 int SrfId,
                                 int Test)
```

Patches: The Patch access structure.

PatchId: The ID of the patch (USER_PATCH_ACCESS_ALL for all).

SrfId: The surface ID of the patches (USER_PATCH_ACCESS_ALL for all).

Test: The test value to set.

Returns: void

Description: Set whether or not a patch should be tested for accessibility. Unset patches will always be defined as inaccessible following the next accessibility test, using PatchAccessSetDir.
See also:

15.2.303 UserPkPackCircles (circpack.c:378)

```
CagdCrvStruct *UserPkPackCircles(CagdCrvStruct *Bndry,
                                 CagdRType Radius,
                                 int NumIter,
                                 CagdRType NumericTol,
                                 CagdRType SubdivTol)
```

Bndry: The curve specifying the boundary of the container.

Radius: The radius of the circles.

NumIter: The number of iterations to run.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol < SubdivTol.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

Returns: List of circles in B-spline form.

Description: Computes a dense packing of circles of given radius inside a container specified by its boundary.

15.2.304 UserPolyline2LinBsplineCrv (usrcnvrt.c:83)

linear curves

conversion

```
CagdCrvStruct *UserPolyline2LinBsplineCrv(const IPPolygonStruct *Poly,
                                          CagdBType FilterDups)
```

Poly: To convert to a linear B-spline curve.

FilterDups: If TRUE, filters out duplicates points, in polygon, in place.

Returns: A linear Bspline curve representing Poly.

Description: Returns a linear B-spline curve constructed from given polyline.
See also: CagdCnvrtPolyline2LinBspCrv, UserPolylines2LinBsplineCrvs,

15.2.305 UserPolylines2LinBsplineCrvs (usrcnvrt.c:47)

linear curves

conversion

```
CagdCrvStruct *UserPolylines2LinBsplineCrvs(const IPPolygonStruct *Polys,
                                            CagdBType FilterDups)
```

Polys: To convert to linear bspline curves.

FilterDups: If TRUE, filters out duplicates points in polygon, in place.

Returns: Linear Bspline curves representing Poly.

Description: Returns a list of linear Bspline curves constructed from given polylines.
See also: CagdCnvrtPolyline2LinBspCrv, UserPolyline2LinBsplineCrv,

15.2.306 UserPuz3DComposeTileOverModel (puzzle3d.c:2105)

```
IPObjectStruct *UserPuz3DComposeTileOverModel(
                                        const IPObjectStruct *InputTile,
                                        const GMBBBboxStruct *TileBBox,
                                        const MdlModelStruct *Mdl,
                                        const CagdVType PolyApproxInfo,
                                        const CagdRType *BoolTols,
                                        CagdRType MergeStitchedTrimmedTiles)
```

InputTile: Tile to use. Can be a list of tiles for specific boundary cases. See UserPackTileCreateTileObject for more. May be overwritten for individual trimmed surface, via a "TileBndryType" attribute on that trimmed surface.

TileBBox: The Bounding box of the tile.

Mdl: The model to tile as a 3D puzzle. Surfaces in Mdl can have "Steps" string attributes that sets the desired min/max steps of tiling the srf, as (MinU, MinV, MinW, MaxU, MaxV, MaxW), all integers.

PolyApproxInfo: Information on how to handle the tiles, if the output is not supposed to be the outline curve - a list of number values as, Thickness - How thick to make the puzzle pieces. zero to leave as outline curve, positive to set the thickness, -1 to convert to polygons but leave zero thickness, -2 to convert to trimmed surface (also zero thickness). PlTolerance - If the tile is approximated into polygons use this tolerance in the approximation. ExtensionScale - Control over the surface extension scale.

BoolTols: Optional Boolean tolerances. See UserCrvs2DBooleans. Can be NULL to use the defaults. A vector of (ProjectOnPlane, EndPtEndPtTol, PlanarityTol).

MergeStitchedTrimmedTiles: Positive to merge stitched trimmed tiles into one poly object(sets the relative stitched edge to size of tiles). Otherwise, no merge is performed.

Returns: A list of 3D puzzle tiles, assmbled into the model Mdl.

Description: Computes a tiling, following Tiler, of all surfaces in Mdl.
See also: UserPuz3DComposeTileOverSrf,

15.2.307 UserPuz3DComposeTileOverSrf (puzzle3d.c:2318)

```
IPObjectStruct *UserPuz3DComposeTileOverSrf(const IPObjectStruct *InputTile,
                                            const GMBBBboxStruct *TileBBox,
                                            CagdSrfStruct **Srf,
                                            CagdBType MapTo3D,
                                            const CagdVType PolyApproxInfo,
                                            IrtRType *TileXYZScale)
```

InputTile: Tile to use. Can be a list of tiles for specific boundary cases. See UserPackTileCreateTileObject for more.

TileBBox: The Bounding box of the tile.

Srf: The surface to tile as a 3D puzzle. joints in the tiles that are out of the param. domain. Srf can have "Steps" string attributes that sets the desired min/max steps of tiling the srf, as (MinU, MinV, MinW, MaxU, MaxV, MaxW), all integers. or optionally additional 3 float parameters (StepU, StepV, StepW), 9 parameters in all. If attribute "Steps" is "AllTiles", then InputTile is assumed to hold the full set of tiles for the entire domain but normalized to domain [0, 1]^2.

MapTo3D: TRUE to compose the tiles into 3D, FALSE to leave in the 2D parametric domain of Srf.

PolyApproxInfo: Information on how to handle the tiles, if the output is not supposed to be the outline curve - a list of number values as, Thickness - How thick to make the puzzle pieces. zero to leave as outline curve, positive to set the thickness, -1 to convert to polygons but leave zero thickness, -2 to convert to trimmed surface (also zero thickness). PlTolerance - If the tile is approximated into polygons use this tolerance in the approximation. ExtensionScale - Control over the surface extension scale.

TileXYZScale: If not NULL, will be set with the XYZ scales applied to the tile.

Returns: A list of 3D puzzle tiles, assmbled into the model Mdl.

Description: Computes a tiling, following Tiler, of all surfaces in Mdl.
See also: UserPuz3DComposeTileOverModel,

15.2.308 UserPuz3DTest (puzzle3d.c:2441)

```
IPObjectStruct *UserPuz3DTest(IPObjectStruct *PObj,
                              const int *StepsMin,
                              const int *StepsMax,
                              CagdVType PolyApproxInfo)
```

PObj: The surface or model to tile its surfaces with puzzle tiles. Surfaces in Mdl can have "Steps" string attributes that sets the desired grid size of puzzle tiles.

StepsMin, StepsMax: The min/max tiling steps desired.

PolyApproxInfo: Information on how to handle the tiles, if the output is not supposed to be the outline curve - a list of number values as, Thickness - How thick to make the puzzle pieces. zero to leave as outline curve, positive to set the thickness, -1 to convert to polygons but leave zero thickness, -2 to convert to trimmed surface (also zero thickness). PlTolerance - If the tile is approximated into polygons use this tolerance in the approximation. ExtensionScale - Control over the surface extension scale.

Returns: The created puzzle.

Description: Debugging routine for this puzzles-in-3D module.
See also:

15.2.309 UserRegisterPointSetSrf (register.c:610)

```
IrtRType UserRegisterPointSetSrf(int n,
                                 IrtPtType *PtsSet,
                                 const CagdSrfStruct *Srf,
                                 IrtRType AlphaConverge,
                                 IrtRType Tolerance,
                                 UserRegisterTestConverganceFuncType
                                                    RegisterTestConvergance,
                                 IrtHmgnMatType RegMat)
```

n: Number of points in PtsSet1.

PtsSet: The vector of points of first set. Might be changed in place.

Srf: A surface the points set is on upto rigid motion.

AlphaConverge: Convergance factor between almost zero (slow and stable) and one (fast but unstable).

Tolerance: Tolerance termination condition, in L infinity sense.

RegisterTestConvergance: A call back function to check for the convergance of this iterative process.

RegMat: Computed transformation.

Returns: Maximal error as the maximal distance between two corresponding points, or IRIT_INFNTY if failed to converge.

Description: Given a set of points, that is a rigid transformation of a given surface find a transformation that takes the set of points to the surface. A local greedy approach is taken here that guarantee local minimum and hence it is assumed the two sets are pretty close and similar. Corrections by marching on the surface are conducted on the fly.
See also: UserRegisterTestConvergance,

15.2.310 UserRegisterTestConvergance (register.c:259)

```
int UserRegisterTestConvergance(IrtRType Dist, int i)
```

Dist: The current distance between the two points sets.

i: The index of the current iteration, starting with zero.

Returns: TRUE if should quit (accuracy is good enough), FALSE otherwise.

Description: A default test convergance function for UserRegisterTwoPointSets.
See also: UserRegisterTwoPointSets,

15.2.311 UserRegisterTestSrfConvergance (register.c:568)

```
int UserRegisterTestSrfConvergance(IrtRType Dist, int i)
```

Dist: The current distance between the two points sets.

i: The index of the current iteration, starting with zero.

Returns: TRUE if should quit (accuracy is good enough), FALSE otherwise.

Description: A test convergance function for UserRegisterTwoPointSets for surface - point-set registration.
See also: UserRegisterTwoPointSets, UserRegisterTestConvergance,

15.2.312 UserRegisterTwoPointSets (register.c:306)

```
IrtRType UserRegisterTwoPointSets(int n1,
                                  IrtPtType *PtsSet1,
                                  int n2,
                                  IrtPtType *PtsSet2,
                                  IrtRType AlphaConverge,
                                  IrtRType Tolerance,
                                  UserRegisterTestConverganceFuncType
                                      RegisterTestConvergance,
                                  IrtHmgnMatType RegMat)
```

n1: Number of points in PtsSet1.

PtsSet1: The vector of points of first set. Might be changed in place.

n2: Number of points in PtsSet2.

PtsSet2: The vector of points in second set. Might be changed in place.

AlphaConverge: Convergance factor between almost zero (slow and stable) and one (fast but unstable).

Tolerance: Tolerance termination condition, in L infinity sense.

RegisterTestConvergance: A call back function to check for the convergance of this iterative process.

RegMat: Computed transformation.

Returns: Error of match between the two corresponding point sets.

Description: Given two sets of points, one is a rigid transformation of the other, find a transformation that takes the first set of points to the second. A local greedy approach is taken here that guarantee local minimum and hence it is assumed the two sets are pretty close and similar. No assumption is made as to the order of the points in the set. However, and while the two sets need not have the same number of points, the optimal solution is achieved at zero distance when each point in one set is at a corresponding point location of the second set.

See also: UserRegisterTestConvergance,

15.2.313 UserRocketFuelDesign3D (rckt_fuel.c:240)

```
IPObjectStruct *UserRocketFuelDesign3D(
                   const TrivTVStruct *FuelTV,
                   int NumLayers,
                   const int NumElements[2],
                   int SliceThrough,
                   int BndrySrfs,
                   const CagdRType *ApplyRGB,
                   UserRocketFuelThrustProfileFuncType ThrustProfileFunc,
                   UserRocketFuelEvalLclFuelThrustFuncType EvalLclFuelThrustFunc,
                   UserRocketFuelAccelerantRatioFuncType AccelerantRatioFunc,
                   UserRocketFuelDeformElementFuncType DeformElementFunc,
                   UserRocketFuelElementCBFuncType ElementCBFunc,
                   void *CBData)
```

FuelTV: A volume to design the fuel for. The burning in assumed starting from Wmin to Wmax.

NumLayers: How many layers to divide into as time progresses.

NumElements: How many element to place in each layer as (Nx, Ny).

SliceThrough: TRUE to remove a slice out of the fuel to see interior.

BndrySrfs: TRUE to returns the boundary surfaces of the fuel, FALSE returns the full set of trivariates.

ApplyRGB: Add RGB coloring following the heterogeneous materials, If not zero. If positive, use this value as a maximum acceleration/retardant color.

ThrustProfileFunc: Function to determine the needed/required thrust, at this time, t in [0, 1]. If NULL, a constant 1 thrust is used.

EvalLclFuelThrustFunc: Function to return what is the expected thrust by the given small surface patch (a micro-element) If NULL, relative area ratio is used.

AccelerantRatioFunc: Function to determine the amount of accelerant/ retardant to add at this location, given the thrust boost that is expected in this location. If NULL, the identity map is used.

DeformElementFunc: Option function to deform the tile, based on its needed boost. NULL to disable.

ElementCBFunc: Optional function to be invoked on each element.

CBData: Optional data to pass to the call back functions.

Returns: Synthesized heterogeneous geometry. Either as a list of trivariates or a list of boundary surfaces.

Description: This functions build a 3D fuel of heterogeneous materials, from a given global trivariate shape with front from WMin to WMax. Assumes the following set of query functions:

1. Thrust(w) - The desired Thrust at time w.
2. BasicFuelThrust(LclFrontArea, GlblFrontArea) - The provided thrust by element LclFrontArea, knowing the entire front GlblFrontArea. 3 AccelerantRatio(Boost) - Given the desired boost over the the basic burning rate of BasicFuelThrust, as a number >1 or <1 if retardants are required, returns the amount of accelerant (Boost > 1) or returdants (Boost < 1) to add, as a relative blend with the regular fuel.

And also given:

1. A 2D cross section to revolve into rocket fuel basic stock, CrossSecion.
2. Number of layers and number of elements (in x and y) in each layer to use in the approximation.

```
Algorithm:
1. For i = 0, step 1, to n layers do
   // Global current front required thrust;
   b. ExpectedThrust[i] := Thrust(t0 + i * dt);
   c. MinLayerDepth[i] := minimum depth of this i'th layer;
   // Compute the expected basic (no accelerant/retardant)
   // thrust from this layer.
   d. FrontBasicThrust := 0;
2. For i = 0, step 1, to n layers do
   a. For each micro-element (tile) in this i'th layer do
      i.   Layer1Boost := local i'th element front required boost;
      i.   Layer2Boost := local i'th element back required boost;
      ii.  DepthElmnt := depth of local i'th element;
      iii. Derive the boost required at the eight corners of the
           micro-element based on the DepthElmnt / MinLayerDepth[i] ratio
           and Layer1/2Boost at the front/back of the micro-element.
      iv.  Add AccelerantRatio (C function UserRocketAddAccelerant) based
           on information in 3.a.iii.
```

See also:

15.2.314 UserRuledSrfFit (rldmatch.c:481)

```
CagdSrfStruct *UserRuledSrfFit(const CagdSrfStruct *Srf,
                               CagdSrfDirType RulingDir,
                               CagdRType ExtndDmn,
                               int Samples,
                               CagdRType *Error,
                               CagdRType *MaxError)
```

Srf: To fit a ruled surface through.

RulingDir: Either the U or the V direction. This is used only to sample Srf and construct the possibly ruling lines.

ExtndDmn: Amount to extended the selected sampled boundary curves. Zero will not extend and match the ruling from the original boundary to its maximum. Not supported.

Samples: Number of samples to compute the dynamic programming with. Typically in the hundreds. Must be greater than two.

Error: The computed error, following the distance between the rulings and the original surface, Srf. In L2 sense.

MaxError: maximum error detected in L-infinity sense.

Returns: The fitted ruled surface, or NULL if error.

Description: Fit a ruled surface to the given general surface Srf. The best fit is found using a dynamic programming search over all possibly rulings, while each ruling line's distance is measured against the surface.

See also:

15.2.315 UserSCvrCoverSrf (dmn_ccvr.c:216)

cover surface

```
CagdCrvStruct *UserSCvrCoverSrf(const CagdCrvStruct *DomainBndry,
                                CagdCrvStruct *CovrCrv,
                                CagdRType CoverEps,
                                CagdRType NumericTol,
                                CagdRType SubdivTol,
                                int TopK,
                                CagdRType TopEps,
                                CagdRType IntrpBlndRatio)
```

DomainBndry: The closed curve which specifies the boundary of the convex input domain.

CovrCrv: Initial covering curve.

CoverEps: The tolerance value upto which the domain is to be covered.

NumericTol: Numeric tolerance of the numeric stage. The numeric stage is employed only if NumericTol $<$ SubdivTol.

SubdivTol: Tolerance of the subdivision process. Tolerance is measured in the parametric space of the multivariates.

TopK: Specifies how many points to add to the covering curve in every iteration of the covering algorithm.

TopEps: Specifies the distance threshold. In every iteration of the covering algorithm, all points which are at a distance farther than TopEps are added to the covering curve. This is used only if TopK is < 0.

IntrpBlndRatio: The blend ratio for approximation and interpolation. A value of 1.0 will imply interpolation and 0.0 will imply imply approximation. A negative value implies approximation with ability to introduce more than one point between a pair of consecutive control points of the current covering curve while refining it.

Returns: The covering curve.

Description: Given a convex domain in E2, this function returns a random curve which covers the domain upto the specified tolerance.
See also:

15.2.316 UserSnapInterCrvs2Bndry (srf_cntr.c:759)

```
void UserSnapInterCrvs2Bndry(CagdCrvStruct *Crvs,
                             CagdRType UMin,
                             CagdRType UMax,
                             CagdRType VMin,
                             CagdRType VMax)
```

Crvs: To snap to boundary, in place.

UMin, UMax, VMin, VMax: The boundary

Returns: void

Description: Snaps the end points of the given curves to lay on the boundary if they are not closed.
See also:

15.2.317 UserSpkPolyObjBorderDelete (sphere_pack.c:3115)

```
void UserSpkPolyObjBorderDelete(UserSpkBorderStructPtr Border)
```

Border: Border object.

Returns: void

Description: Deletes a Polygonal Object Border object.

15.2.318 UserSpkPolyObjBorderNew (sphere_pack.c:3012)

```
UserSpkBorderStructPtr UserSpkPolyObjBorderNew(const IPObjectStruct *PObj,
                                               IrtRType GridCellSize,
                                               IrtRType MaxSphereRadius)
```

PObj: Polygonal Object to use (Can be safely deleted after this function returns).

GridCellSize: Linear size (width/height/depth) of a grid cell.

MaxSphereRadius: Maximum sphere radius this border will be used with.

Returns: Newly created Polygonal Object Border object.

Description: Creates a new Polygonal Object Border object.

15.2.319 UserSpkSolvingProcessDelete (sphere_pack.c:2580)

```
void UserSpkSolvingProcessDelete(UserSpkSolvingProcessStructPtr Process)
```

Process: Solving Process object.

Returns: void

Description: Deletes a Solving Process object.

15.2.320 UserSpkSolvingProcessFork (sphere_pack.c:2528)

```
UserSpkSolvingProcessStructPtr UserSpkSolvingProcessFork(
                                        UserSpkSolvingProcessStructPtr Parent)
```

Parent: Parent process to fork.

Returns: Handle for the newly created Solving Process object.

Description: Forks a Solving Process object into a new one. The two processes can be used independently in different threads.

15.2.321 UserSpkSolvingProcessNew (sphere_pack.c:2486)

```
UserSpkSolvingProcessStructPtr UserSpkSolvingProcessNew(
                                const UserSpkSolvingSettingsStruct *Settings,
                                UserSpkBorderStructPtr Border,
                                UserSpkSolvingStatusStruct *Status)
```

Settings: Solving settings.

Border: Border to fill with spheres.

Status: (OUT) Status of the solving process after initialization.

Returns: Handle for the newly created Solving Process object.

Description: Creates a new Solving Process object.

15.2.322 UserSpkSolvingProcessRunGravityAttempt (sphere_pack.c:2642)

```
void UserSpkSolvingProcessRunGravityAttempt(
                                        UserSpkSolvingProcessStructPtr Process,
                                        UserSpkSolvingStatusStruct *Status)
```

Process: Solving Process object.

Status: (OUT) Status of the solving process after the attempt.

Returns: void

Description: Runs a Gravity algorithm attempt.

15.2.323 UserSpkSolvingProcessRunRepulsionsAttempt (sphere_pack.c:2604)

```
void UserSpkSolvingProcessRunRepulsionsAttempt(
                                    UserSpkSolvingProcessStructPtr Process,
                                    UserSpkSolvingStatusStruct *Status)
```

Process: Solving Process object.

Status: (OUT) Status of the solving process after the attempt.

Returns: void

Description: Runs a Random Repulsion algorithm attempt.

15.2.324 UserSpkSolvingSettingsGetDefault (sphere_pack.c:2445)

```
UserSpkSolvingSettingsStruct UserSpkSolvingSettingsGetDefault(
                                                    IrtRType SphereRadius)
```

SphereRadius: Sphere radius.

Returns: Default solving settings.

Description: Generates default solving settings.

15.2.325 UserSpkSurfObjBorderDelete (sphere_pack.c:3196)

```
void UserSpkSurfObjBorderDelete(UserSpkBorderStructPtr Border)
```

Border: Border object.

Returns: void

Description: Deletes a Surface Object Border object.

15.2.326 UserSpkSurfObjBorderNew (sphere_pack.c:3142)

```
UserSpkBorderStructPtr UserSpkSurfObjBorderNew(const IPObjectStruct *PObj,
                                               IrtRType SubdivTol,
                                               IrtRType NumerTol)
```

PObj: Polygonal Object to use.

SubdivTol: Subdivision tolerance.

NumerTol: Numerical tolerance.

Returns: Newly created Surface Object Border object.

Description: Creates a new Surface Object Border object.

15.2.327 UserSrfFixedCurvatureLines (srfcrvtr.c:209)

curvature

```
IPObjectStruct *UserSrfFixedCurvatureLines(const CagdSrfStruct *Srf,
                                           CagdRType k1,
                                           CagdRType Step,
                                           CagdRType SubdivTol,
                                           CagdRType NumericTol,
                                           int Euclidean)
```

Srf: To compute lines of one principle curvature equal to k1.

k1: The prescribed principle curvature.

Step: Step size for curve tracing.

SubdivTol: The subdivision tolerance to use.

NumericTol: The numerical tolerance to use.

Euclidean: TRUE to return curvature lines curves in Euclidean space, FALSE in parameter space.

Returns: A list of piecewise linear curves in Srf where one principle curvature is equal to k1.

Description: Computes all locations (typically curves) on input surface Srf that have one principle curvature value equal to a fixed value k1.

```
  Because   k1 k2 = K    and   ((k1 + k2)/2)^2 = H^2,
we have     ((k1 + K / k1)/2)^2 - H^2 = 0.
```

Both K and H^2 are rational so let K = KN / KD (numerator / denominator) and similarly let H^2 = HN / HD. Then, we need to solve for the zeros of (k1 / 2 + KN / (KD * k1 * 2))^2 - HN / HD = 0, or the zeros of (K1 * KD * k1 + KN)^2 * HD - HN * (KD * k1 * 2)^2 = 0.

See also: SymbSrfGaussCurvature, SymbSrfMeanCurvatureSqr,

15.2.328 UserSrfKernel (ff_krnl.c:51)

```
IPObjectStruct *UserSrfKernel(const CagdSrfStruct *Srf,
                              CagdRType SubdivTol,
                              int SkipRate)
```

Srf: To compute its kernel.

SubdivTol: Accuracy of subdivision approximation. 0.01 is a good start.

SkipRate: Step size over the parabolic points, 1 to process them all.

Returns: A polyhedra approximating the kernel, or NULL if empty set.

Description: Computes the kernel of a freeform closed C^1 continuous surface. The parabolic curves are computed and surface tangent planes swept along these parabolic curves clip the volume, resulting with the kernel.

See also: SymbSrfGaussCurvature, UserCntrSrfWithPlane,

15.2.329 UserSrfParabolicLines (ff_krnl.c:254)

```
IPObjectStruct *UserSrfParabolicLines(const CagdSrfStruct *Srf,
                                      CagdRType Step,
                                      CagdRType SubdivTol,
                                      CagdRType NumericTol,
                                      int Euclidean,
                                      int DecompSrfs)
```

Srf: A surface to derive its parabolic lines.

Step: Step size for curve tracing.

SubdivTol: The subdivision tolerance to use.

NumericTol: The numerical tolerance to use.

Euclidean: TRUE to evaluate the parabolic lines into Euclidean space. Ignored if DecompSrfs is TRUE.

DecompSrfs: TRUE to decompose the given Srf into convex/concave and saddle regions.

Returns: Parabolic lines of Srf as a list of polylines approximation, or NULL if none.

Description: Computes the parabolic lines of the given surface, if any.

See also: SymbSrfGaussCurvature, UserSrfparabolicSheets,

15.2.330 UserSrfParabolicSheets (ff_krnl.c:426)

```
IPObjectStruct *UserSrfParabolicSheets(const CagdSrfStruct *Srf,
                                       CagdRType Step,
                                       CagdRType SubdivTol,
                                       CagdRType NumericTol,
                                       CagdRType SheetExtent)
```

Srf: A surface to derive its parabolic lines.

Step: Step size for curve tracing.

SubdivTol: The subdivision tolerance to use.

NumericTol: The numerical tolerance to use.

SheetExtent: Amount to extent the sheet from the parabolic lines.

Returns: Parabolic sheets of Srf as a list of polygonal approximation, or NULL if none.

Description: Computes the parabolic sheets around parabolic lines of given surface, if any, as piecewise linear approximation. These are developable surfaces.
See also: SymbSrfGaussCurvature, UserSrfparabolicLines,

15.2.331 UserSrfSrfInter (srf_ssi.c:75)

```
int UserSrfSrfInter(const CagdSrfStruct *CSrf1,
                    const CagdSrfStruct *CSrf2,
                    int Euclidean,
                    CagdRType Eps,
                    int AlignSrfs,
                    CagdCrvStruct **Crvs1,
                    CagdCrvStruct **Crvs2)
```

CSrf1, CSrf2: Two surfaces to compute their intersection points.

Euclidean: TRUE for curves in Euclidean space, FALSE for pairs of curves in parametric space.

Eps: Accuracy of computation. Currently measured in the parametric domain of the surfaces.

AlignSrfs: If non zero surface are aligned by rotation so that bounding boxes are axes parallel to increase chances of no overlap. If AlignSrfs > 1, stretching is also applied to the surfaces to try and make them orthogonal.

Crvs1: Intersection curves of the first surface. If Euclidean is TRUE the 3-space curves are returned. If Euclidean is FALSE, curves are returned in UV space.

Crvs2: Intersection curves of the second surface.

Returns: TRUE if found intersection, FALSE otherwise.

Description: Computes the intersection curves, if any, of the two given surfaces.
See also: CagdCrvCrvInter,

15.2.332 UserSrfTopoAspectGraph (visible.c:346)

```
IPPolygonStruct *UserSrfTopoAspectGraph(CagdSrfStruct *PSrf,
                                        CagdRType SubdivTol)
```

PSrf: To compute the aspect graph's topology partitioning for.

SubdivTol: Accuracy of parabolic piecewise linear approximation.

Returns: Polylines on the unit sphere, depicting the aspect graph's partitioning lines.

Description: Computes the views on the unit sphere at which the number of silhouette curves is changing. This is a subset of the events typically considered in aspect graphs and, for example, events such as silhouette cusps are not considered. The viewing directions where the number of silhouette curves/loops can be modified are views in the tangent plane of the surface at parabolic points of the surfaces, viewed from the zero principal curvatures' direction. Hence, we simply derive the parabolic lines of the surface and then remap them to the direction of the zero principal curvature there.
See also: SymbSrfGaussCurvature, UserCntrSrfWithPlane, MvarSrfSilhInflections,

15.2.333 UserSrfUmbilicalPts (srfcrvtr.c:48)

curvature

```
MvarPtStruct *UserSrfUmbilicalPts(const CagdSrfStruct *Srf,
                                  CagdRType SubTol,
                                  CagdRType NumTol)
```

Srf: Surface to compute its umbilical points, if any.

SubTol: Subdivision tolerance of computation.

NumTol: Numerical tolerance of computation.

Returns: A list of UV parameters of umbilical points, or NULL if none.

Description: Computes the umbilical points of a given surface. The umbilicals are computed as the zeros of the function C = H^2 - K. C is never negative and hence we actually solve for dC/du = dC/dv = 0 and test the values of C there. Hence (We only consider numerator of C which is sufficient for zeros), C = H^2 - K = (2FM - EN - GL)^2 - 4(LN - M^2)(EG - F^2).

See also: SymbSrfFff, SymbSrfSff, SymbSrfMeanCurvatureSqr, SymbSrfGaussCurvature, , MvarMVsZeros,

15.2.334 UserSrfVisSurfaceVisualize (vol_vis.c:1343)

```
IPObjectStruct *UserSrfVisSurfaceVisualize(
                                const IPObjectStruct *GeomSrf,
                                const IPObjectStruct *PropSrf,
                                UserVolVisGenPolygonRepresentationFuncType
                                                    GenPolygonRepresentFunc,
                                void *GenPolygonRepresentData,
                                UserVolVisGetRGBFromRealFuncType
                                                        GetRGBFromRealFunc,
                                void *GetRGBFromRealData,
                                int PropCoord,
                                IrtRType *MinVal,
                                IrtRType *MaxVal,
                                const char *MapFuncStr,
                                IrtRType MaxEdgeLen,
                                char *Error)
```

GeomSrf: A (list of) geometry of surface(s) (or polys with UV coords.).

PropSrf: The property surface(s) to map it into color over GeomSrf.

GenPolygonRepresentFunc: A function to tesselate the geometry, or NULL to use a default tesselation code.

GenPolygonRepresentData: Optional reference to data to transfer to the call back function GenPolygonRepresentFunc. NULL, to ignore.

GetRGBFromRealFunc: Function to convert a scalar value in [0, 1] to RGB in [0, 255]^3.

GetRGBFromRealData: Optional reference to data to transfer to the call back function GetRGBFromRealFunc. NULL, to ignore.

PropCoord: Axis index of property, 1 for X, 2 for y, etc.

MinVal: Minimum property value found.

MaxVal: Maximum property value found.

MapFuncStr: Optional mapping function f to apply, as f(PropVal).

MaxEdgeLen: To control maximal sizes of polygonal approximation.

Error: Will be updated with the string error if this function failed. Must be a string of 256 bytes or more.

Returns: A polygonal object approximating PolySrf, colored to follow the property, NULL if error.

Description: Routine to map some property onto a (tessellated) surface.

15.2.335 UserSrfVisibConeDecomp (visible.c:57)

```
IPObjectStruct *UserSrfVisibConeDecomp(const CagdSrfStruct *Srf,
                                       CagdRType Resolution,
                                       CagdRType ConeAngle)
```

Srf: To decompose into visibility cones.

Resolution: Of polygonal approximation of surface Srf.

ConeAngle: Of coverage over the unit sphere. ConeAngle prescribes the opening angle of the cone. In Radians.

Returns: A list of trimmed surface regions of surface Srf that are visible from a selected direction (available as "VIewDir" attribute on them) ad that the union of the list covers the entire surface Srf.

Description: Computes a decomposition of surface Srf into regions that are visible with normals deviating in a cone of size as set via ConeAngle.
See also: UserViewingConeSrfDomains, UserVisibilityClassify,

15.2.336 UserSweepSectionDone (toolswep.c:2152)

```
int UserSweepSectionDone(UserSwpGenInfoStruct *GI)
```

GI: General information of sweep sec. module, to free.

Returns: TRUE if successful, FALSE otherwise.

Description: Terminates the sweep section module.
See also:

15.2.337 UserSweepSectionInit (toolswep.c:375)

```
UserSwpGenInfoStruct *UserSweepSectionInit(const IrtRType ToolOrigin[3])
```

ToolOrigin: The point labelled as the origin of the tool. The Position part of motion specifies the new position of the ToolCentroid.

Returns: Allocated dynamically general handle of the sweep sec. module.

Description: Initializes the Sweep section computation.
See also: UserSweepSectionDone,

15.2.338 UserSwpSecCnstrctToolCone (toolswep.c:623)

```
void UserSwpSecCnstrctToolCone(UserSwpGenInfoStruct *GI,
                               IrtRType Center[3],
                               IrtRType MajorRadius,
                               IrtRType MinorRadius,
                               IrtRType Height)
```

GI: Allocated dynamically general handle of the sweep sec. module.

Center: The center of the cone.

MajorRadius: The major radius of the cone.

MinorRadius: The minor radius of the cone.

Height: The height of the cone.

Returns: void

Description: Adds a conical shape to the tool.
See also: UserSwpSecCnstrctToolGnrl, UserSwpSecCnstrctToolSph, , UserSwpSecCnstrctToolCyl,

15.2.339 UserSwpSecCnstrctToolCyl (toolswep.c:494)

```
void UserSwpSecCnstrctToolCyl(UserSwpGenInfoStruct *GI,
                              IrtRType Center[3],
                              IrtRType Radius,
                              IrtRType Height)
```

GI: Allocated dynamically general handle of the sweep sec. module.

Center: The center of the cylinder.

Radius: The radius of the cylinder.

Height: The height of the cylinder.

Returns: void

Description: Adds a cylindrical shape to the tool.
See also: UserSwpSecCnstrctToolGnrl, UserSwpSecCnstrctToolSph, , UserSwpSecCnstrctToolCone,

15.2.340 UserSwpSecCnstrctToolGnrl (toolswep.c:697)

```
void UserSwpSecCnstrctToolGnrl(UserSwpGenInfoStruct *GI,
                               const CagdCrvStruct *Profile)
```

GI: Allocated dynamically general handle of the sweep sec. module.

Profile: The profile to be revolved, given in XZ-plane.

Returns: void

Description: Adds a general surface of revolution to the tool.
See also: UserSwpSecCnstrctToolCyl, UserSwpSecCnstrctToolSph, , UserSwpSecCnstrctToolCone,

15.2.341 UserSwpSecCnstrctToolSph (toolswep.c:421)

```
void UserSwpSecCnstrctToolSph(UserSwpGenInfoStruct *GI,
                              IrtRType Center[3],
                              IrtRType Radius)
```

GI: Allocated dynamically general handle of the sweep sec. module.

Center: The center of the sphere.

Radius: The radius of the sphere.

Returns: void

Description: Adds a spherical shape to the tool.
See also: UserSwpSecCnstrctToolGnrl, UserSwpSecCnstrctToolCyl, , UserSwpSecCnstrctToolCone,

15.2.342 UserSwpSecElimRedundantToolShapes (toolswep.c:798)

```
void UserSwpSecElimRedundantToolShapes(UserSwpGenInfoStruct *GI)
```

GI: Allocated dynamically general handle of the sweep sec. module.

Returns: void

Description: Eliminates redundant shapes from the definition of the tool.
See also: UserSwpSecCnstrctToolCyl, UserSwpSecCnstrctToolSph, , UserSwpSecCnstrctToolCone, UserSwpSecCnstrctToolGnrl,

15.2.343 UserSwpSecGetLineEnvelope (toolswep.c:1571)

```
CagdBType UserSwpSecGetLineEnvelope(UserSwpGenInfoStruct *GI,
                                    int PlnNrmlDir1,
                                    IrtRType PlnValue1,
                                    int PlnNrmlDir2,
                                    IrtRType PlnValue2,
                                    CagdPtStruct **EnvlPts,
                                    CagdPtStruct **Nrmls)
```

GI: Allocated dynamically general handle of the sweep sec. module.

PlnNrmlDir1: The direction of normal to the first plane, 0 for X, 1 for Y and 2 for Z.

PlnValue1: The intercept value of the plane along PlnNrmlDir1.

PlnNrmlDir2: The direction of normal to the second plane, 0 for X, 1 for Y and 2 for Z.

PlnValue2: The intercept value of the plane along PlnNrmlDir2.

EnvlPts: Intersection as a list of points.

Nrmls: Normals to envelope as a list of points.

Returns: TRUE is successful FALSE otherwise.

Description: Computes the intersection of the envelope of the tool with the given line, given as intersection of two hyperplanes. Assumes that the last motion applied to the tool was a cut.
See also: UserSwpSecGetPlaneEnvelope, UserSwpSecGetSrfEnvelope,

15.2.344 UserSwpSecGetPlaneEnvelope (toolswep.c:1287)

```
CagdBType UserSwpSecGetPlaneEnvelope(UserSwpGenInfoStruct *GI,
                                     int PlnNrmlDir,
                                     IrtRType PlnValue,
                                     CagdPolylineStruct **PlnEnvl,
                                     CagdPolylineStruct **Nrmls)
```

GI: Allocated dynamically general handle of the sweep sec. module.

PlnNrmlDir: The direction of normal to the plane, 0 for X, 1 for Y and 2 for Z.

PlnValue: The intercept value of the plane along PlnNrmlDir.

PlnEnvl: Intersection curve as a list of polylines.

Nrmls: Normals to envelope as a list of polylines.

Returns: TRUE is successful FALSE otherwise.

Description: Computes the intersection of the envelope of the tool with the given hyperplane. Assumes that the last motion applied to the tool was a cut.
See also: UserSwpSecGetFlatPlaneEnvelope, UserSwpSecGetSrfEnvelope,

15.2.345 UserSwpSecGetSrfEnvelope (toolswep.c:1707)

```
IPObjectStruct *UserSwpSecGetSrfEnvelope(UserSwpGenInfoStruct *GI)
```

GI: Allocated dynamically general handle of the sweep sec. module.

Returns: Envelope surface as a list of triangles.

Description: Computes the envelope surface of the tool as a 2-manifold by involving 2D constraint solver.
See also: UserSwpSecGetPlaneEnvelope, UserSwpSecGetLineEnvelope,

15.2.346 UserSwpSecMachiningSimulation (toolswep.c:1937)

```
IPPolygonStruct *UserSwpSecMachiningSimulation(
                                            const CagdCrvStruct *ToolProfile,
                                            const CagdPType ToolOrigin,
                                            const IPObjectStruct *MotionData,
                                            int DexelGridType,
                                            const CagdPType GridOrigin,
                                            const CagdPType GridEnd,
                                            int NumDexel0,
                                            int NumDexel1,
                                            const CagdSrfStruct *StockSrf,
                                            CagdRType RectStockTopLevel,
                                            CagdRType RectStockBtmLevel,
                                            const char *OutputSavePath)
```

ToolProfile: Rotate about z-axis to obtain the tool.

ToolOrigin: Coordinates of the tool center.

MotionData: List of List of 6 reals, viz. positions and orientations of the tool, or filename.

DexelGridType: 0,1 or 2 for dexels along x,y or z axis.

GridOrigin: Two coordinates specifying the start of grid.

GridEnd: Two coordinates specifying the end of grid.

NumDexel0: Number of dexels along first grid direction.

NumDexel1: Number of dexels along second grid direction.

StockSrf: Closed surface specifying the stock.

RectStockTopLevel: Top height value for rectangular stock, used if StockSrf is NULL.

RectStockBtmLevel: Bottom height value for rectangular stock, used if StockSrf is NULL.

OutputSavePath: Prefix of path where to save intermediate stocks. Can be empty string.

Returns: Triangulated machined stock.

Description: Performs 5-axis CNC machining simulation on given stock, using the given tool and motion. Stock is specified by a set of closed surfaces, or a rectangular stock of given dimensions. The dexel grid representing the stock is specified too. The tool is specified by its profile which is rotated about the z-axis to obtain the tool. The SLERP motion is given in a text file as a list of position, orientation, one on each line. The final stock after machining is returned as a list of triangles.
See also: UserSweepSectionInit,

15.2.347 UserSwpSecRenderTool (toolswep.c:2107)

```
IPObjectStruct *UserSwpSecRenderTool(UserSwpGenInfoStruct *GI)
```

GI: Allocated dynamically general handle of the sweep sec. module.

Returns: The tool's geometry.

Description: Returns all the surfaces of the tool, as an object.
See also:

15.2.348 UserSwpSecToolCut (toolswep.c:1003)

```
int UserSwpSecToolCut(UserSwpGenInfoStruct *GI,
                      IrtRType Position[3],
                      IrtRType Orientation[3])
```

GI: Allocated dynamically general handle of the sweep sec. module.

Position: New position to move tool to.

Orientation: New orientation of the tool.

Returns: TRUE if successful, FALSE otherwise.

Description: Sweeps the tool to new position and orientation, thereby cutting material from the block. This motion leads to the envelope.
See also: UserSwpSecToolMove,

15.2.349 UserSwpSecToolMove (toolswep.c:915)

```
int UserSwpSecToolMove(UserSwpGenInfoStruct *GI,
                       IrtRType Position[3],
                       IrtRType Orientation[3])
```

GI: Allocated dynamically general handle of the sweep sec. module.

Position: New position to move tool to.

Orientation: New orientation of the tool.

Returns: TRUE if successful, FALSE otherwise.

Description: Moves the tool to new position and orientation without doing any envelope computation.
See also: UserSwpSecToolCut,

15.2.350 UserTVZeroJacobian (tv0jacob.c:43)

```
IPObjectStruct *UserTVZeroJacobian(const TrivTVStruct *TV,
                                   CagdBType Euclidean,
                                   int SkipRate,
                                   const CagdRType Fineness[3])
```

TV: Trivariate to derives its zero jacobian.

Euclidean: TRUE to evaluate into Euclidean space, FALSE to return the polygonal approximation in the parametric space of TV.

SkipRate: Of data in the volume. 1 skips nothing, 2 every second, etc.

Fineness: Of trivariate global refinment level, 0 for no ref.

Returns: The approximation of the zeros of the Jacobian, in all three axes.

Description: Computes a piecewise linear polygonal approximation to the zeros of the Jacobian of the given trivariate function.
See also: UserTrivarZeros,

15.2.351 UserText2OutlineCurves2D (font2d.c:94)

```
IPObjectStruct *UserText2OutlineCurves2D(const char *Str,
                                         IrtRType Space,
                                         IrtRType ScaleFactor,
                                         IrtRType *Height)
```

Str: The text to convert to geometry.

Space: Between individual characters, in X axis.

ScaleFactor: Scale factor to control the size of geometry.

Height: Output parameter to hold the geometry height.

Returns: Geometry representing the given text.

Description: Generate Geometry represents a given text in Str, with Spacing space between character and scaling factor of Scale.
See also: UserText2OutlineCurves2DInit,

15.2.352 UserText2OutlineCurves2DInit (font2d.c:51)

```
IPObjectStruct *UserText2OutlineCurves2DInit(const char *FName)
```

FName: Name of IRIT font file to load.

Returns: The Object list containing the characters, or NULL if none.

Description: Loads the IRIT font file that is specified by the FName file name. An IRIT font file contains the outline geometries of the characters as a list ordered according to the ASCII table starting from 32 (space). Irit font file could be generated by irit script command of the form All = list(Text2Geom(" ", "Times New Roman", 0, 0, 0, list(0.01, 0.3), 0.005, 3), Text2Geom("!", "Times New Roman", 0, 0, 0, list(0.01, 0.3), 0.005, 3), Text2Geom("\"", "Times New Roman", 0, 0, 0, list(0.01, 0.3), 0.005, 3), ...
See also: UserText2OutlineCurves2D,

15.2.353 UserTile2DSemi12312 (tiles2d_sem_reg.c:1051)

```
IPObjectStruct *UserTile2DSemi12312(UserTileSemiRegGeomType TileType,
                                    const CagdRType TileSize)
```

TileType: Type of the tile needed, univariate, bivariate or trivariate.

TileSize: The length of side of the polygons.

Returns: A pointer to the basic tile.

Description: Creates a tile object to be used in order to tile a 2D domain with polygon vertex 12_3_12.
See also: UserTile2DSemi6363, UserTile2DSemi848, UserTile2DSemi4346, UserTile2DSemi33336, UserTile2DSemi1246, UserTile2DSemi43433, , ,

15.2.354 UserTile2DSemi12312D (tiles2d_sem_reg_dual.c:1368)

```
IPObjectStruct *UserTile2DSemi12312D(UserTileSemiRegDualGeomType TileType,
                                     const CagdRType TileSize,
                                     const CagdRType Indent,
                                     const CagdRType CrvAmt)
```

TileType: Type of the tile needed, Univariate_Open, univariate closed, bivariate, trivariate.

TileSize: The length of side of the polygons.

Indent: A Fraction (0 - 1) of the length of a primal edge that is in used the dual reconstruction.

CrvAmt: Curvature sharpness of dual reconstructed curves (0 - 1).

Returns: A pointer to the basic tile.

Description: Creates a tile object to be used in order to tile a 2D domain with polygon vertex 12312Dual.
See also: UserTile2DSemi6363D, UserTile2DSemi44333D, UserTile2DSemi4346D, , UserTile2DSemi33336D, UserTile2DSemi1246D, UserTile2DSemi848D, , ,

15.2.355 UserTile2DSemi12312DSetRectNeighborsAttrib (tiles2d_sem_reg_dual.c:1460)

```
void UserTile2DSemi12312DSetRectNeighborsAttrib(IPObjectStruct *RectTile)
```

RectTile: The tile to update.

Returns: void

Description: Updates the elements in a rectangular tile 12312Dual with attribute referencing their previous/edge neighbor in either/both X, Y directions (adjacent in list object of rectangular tiles, share neighbors because original items tiles are split, in order to create a rectangle domain).
See also: UserTile2DSemi123123D, UserTile2DSemi4346DSetRectNeighborsAttrib, , UserTile2DSemi848DSetRectNeighborsA
, UserTile2DSemi6363DSetRectNeighborsAttrib, , UserTile2DSemi44333DSetRectNeighborsAttrib, , UserTile2DSemi43433DSetRe
, UserTile2DSemi1246DSetRectNeighborsAttrib, ,

15.2.356 UserTile2DSemi1246 (tiles2d_sem_reg.c:1182)

```
IPObjectStruct *UserTile2DSemi1246(UserTileSemiRegGeomType TileType,
                                   const CagdRType TileSize)
```

TileType: Type of the tile needed, univariate, bivariate or trivariate.

TileSize: The length of side of the polygons.

Returns: A pointer to the basic tile.

Description: Creates a tile object to be used in order to tile a 2D domain with polygon vertex 12_4_6.
See also: UserTile2DSemi6363, UserTile2DSemi848, UserTile2DSemi4346, UserTile2DSemi33336, UserTile2DSemi12312, UserTile2DSemi43433, , ,

15.2.357 UserTile2DSemi1246D (tiles2d_sem_reg_dual.c:1879)

```
IPObjectStruct *UserTile2DSemi1246D(UserTileSemiRegDualGeomType TileType,
                                    const CagdRType TileSize,
                                    const CagdRType Indent,
                                    const CagdRType CrvAmt)
```

TileType: Type of the tile needed, Univariate_Open, univariate closed, bivariate, trivariate.

TileSize: The length of side of the polygons.

Indent: A Fraction (0 - 1) of the length of a primal edge that is in used the dual reconstruction.

CrvAmt: Curvature sharpness of dual reconstructed curves (0 - 1).

Returns: A pointer to the basic tile.

Description: Creates a tile object to be used in order to tile a 2D domain with polygon vertex 1246Dual.
See also: UserTile2DSemi6363D, UserTile2DSemi44333D, UserTile2DSemi4346D, , UserTile2DSemi33336D, UserTile2DSemi848D, UserTile2DSemi12312D, , ,

15.2.358 UserTile2DSemi1246DSetRectNeighborsAttrib (tiles2d_sem_reg_dual.c:1988)

```
void UserTile2DSemi1246DSetRectNeighborsAttrib(IPObjectStruct *RectTile)
```

RectTile: The tile to update.

Returns: void

Description: Updates the elements in a rectangular tile 6363Dual with attribute referencing their previous/edge neighbor in either/both X, Y directions (adjacent in list object of rectangular tiles, share neighbors because original items tiles are split, in order to create a rectangle domain).
See also: UserTile2DSemi1246D, UserTile2DSemi4346DSetRectNeighborsAttrib, , UserTile2DSemi6363DSetRectNeighborsAtt
, UserTile2DSemi848DSetRectNeighborsAttrib, , UserTile2DSemi12312DSetRectNeighborsAttrib, , UserTile2DSemi44333DSetRect
, ,

15.2.359 UserTile2DSemi33336 (tiles2d_sem_reg.c:917)

```
IPObjectStruct *UserTile2DSemi33336(UserTileSemiRegGeomType TileType,
                                    const CagdRType TileSize)
```

TileType: Type of the tile needed, univariate, bivariate or trivariate.

TileSize: The length of side of the polygons.

Returns: A pointer to the basic tile.

Description: Creates a tile object to be used in order to tile a 2D domain with polygon vertex 33336.
See also: UserTile2DSemi6363, UserTile2DSemi848, UserTile2DSemi4346, UserTile2DSemi44333, UserTile2DSemi1246, UserTile2DSemi12312, , ,

15.2.360 UserTile2DSemi33336D (tiles2d_sem_reg_dual.c:2060)

```
IPObjectStruct *UserTile2DSemi33336D(UserTileSemiRegDualGeomType TileType,
                                     const CagdRType TileSize,
                                     const CagdRType Indent,
                                     const CagdRType CrvAmt)
```

TileType: Type of the tile needed, Univariate_Open, univariate closed, bivariate, trivariate.

TileSize: The length of side of the polygons.

Indent: A Fraction (0 - 1) of the length of a primal edge that is in used the dual reconstruction.

CrvAmt: Curvature sharpness of dual reconstructed curves (0 - 1).

Returns: A pointer to the basic tile.

Description: Creates a tile object to be used in order to tile a 2D domain with polygon vertex 33336Dual.
See also: UserTile2DSemi6363D, UserTile2DSemi44333D, UserTile2DSemi4346D, , UserTile2DSemi848D, UserTile2DSemi1246D, UserTile2DSemi12312D, , ,

15.2.361 UserTile2DSemi43433 (tiles2d_sem_reg.c:224)

```
IPObjectStruct *UserTile2DSemi43433(UserTileSemiRegGeomType TileType,
                                    const CagdRType TileSize)
```

TileType: Type of the tile needed, univariate, bivariate or trivariate.

TileSize: The length of side of the polygons.

Returns: A pointer to the basic tile.

Description: Creates a tile object to be used in order to tile a 2D domain with polygon vertex 43433.
See also: UserTile2DSemi44333, UserTile2DSemi848, UserTile2DSemi4346, UserTile2DSemi33336, UserTile2DSemi1246, UserTile2DSemi12312, , ,

15.2.362 UserTile2DSemi43433D (tiles2d_sem_reg_dual.c:2171)

```
IPObjectStruct *UserTile2DSemi43433D(UserTileSemiRegDualGeomType TileType,
                                     const CagdRType TileSize,
                                     const CagdRType Indent,
                                     const CagdRType CrvAmt)
```

TileType: Type of the tile needed, Univariate_Open, univariate closed, bivariate, trivariate.

TileSize: The length of side of the polygons.

Indent: A Fraction (0 - 1) of the length of a primal edge that is in used the dual reconstruction.

CrvAmt: Curvature sharpness of dual reconstructed curves (0 - 1).

Returns: A pointer to the basic tile.

Description: Creates a tile object to be used in order to tile a 2D domain with polygon vertex 43433Dual.
See also: UserTile2DSemi6363D, UserTile2DSemi44333D, UserTile2DSemi4346D, , UserTile2DSemi33336D, UserTile2DSemi1246D, UserTile2DSemi12312D, , ,

15.2.363 UserTile2DSemi43433DSetRectNeighborsAttrib (tiles2d_sem_reg_dual.c:2292)

```
void UserTile2DSemi43433DSetRectNeighborsAttrib(IPObjectStruct *RectTile)
```

RectTile: The tile to update.

Returns: void

Description: Updates the elements in a rectangular tile 6363Dual with attribute referencing their previous/edge neighbor in either/both X, Y directions (adjacent in list object of rectangular tiles, share neighbors because original items tiles are split, in order to create a rectangle domain).
See also: UserTile2DSemi43433D, UserTile2DSemi4346DSetRectNeighborsAttrib, , UserTile2DSemi6363DSetRectNeighborsA
, UserTile2DSemi848DSetRectNeighborsAttrib, , UserTile2DSemi12312DSetRectNeighborsAttrib, , UserTile2DSemi44333DSetRect
, ,

15.2.364 UserTile2DSemi4346 (tiles2d_sem_reg.c:759)

```
IPObjectStruct *UserTile2DSemi4346(UserTileSemiRegGeomType TileType,
                                   const CagdRType TileSize)
```

TileType: Type of the tile needed, univariate, bivariate or trivariate.

TileSize: The length of side of the polygons.

Returns: A pointer to the basic tile.

Description: Creates a tile object to be used in order to tile a 2D domain with polygon vertex 4346.
See also: UserTile2DSemi6363, UserTile2DSemi848, UserTile2DSemi44333, , UserTile2DSemi33336, UserTile2DSemi1246, UserTile2DSemi12312, , ,

15.2.365 UserTile2DSemi4346D (tiles2d_sem_reg_dual.c:1687)

```
IPObjectStruct *UserTile2DSemi4346D(UserTileSemiRegDualGeomType TileType,
                                    const CagdRType TileSize,
                                    const CagdRType Indent,
                                    const CagdRType CrvAmt)
```

TileType: Type of the tile needed, Univariate_Open, univariate closed, bivariate, trivariate.

TileSize: The length of side of the polygons.

Indent: A Fraction (0 - 1) of the length of a primal edge that is in used the dual reconstruction.

CrvAmt: Curvature sharpness of dual reconstructed curves (0 - 1).

Returns: A pointer to the basic tile.

Description: Creates a tile object to be used in order to tile a 2D domain with polygon vertex 4346Dual.
See also: UserTile2DSemi6363D, UserTile2DSemi44333D, UserTile2DSemi848D, UserTile2DSemi33336D, UserTile2DSemi1246D, UserTile2DSemi12312D, , ,

15.2.366 UserTile2DSemi4346DSetRectNeighborsAttrib (tiles2d_sem_reg_dual.c:1798)

```
void UserTile2DSemi4346DSetRectNeighborsAttrib(IPObjectStruct *RectTile)
```

RectTile: The tile to update.

Returns: void

Description: Updates the elements in a rectangular tile 4346D with attribute referencing their previous/edge neighbor in either/both X, Y directions (adjacentin list object of rectangular tiles, share neighbors because original items tiles are split, in order to create a rectangle domain).
See also: UserTile2DSemi4346D, UserTile2DSemi848DSetRectNeighborsAttrib, , UserTile2DSemi6363DSetRectNeighborsAttr
, UserTile2DSemi12312DSetRectNeighborsAttrib, , UserTile2DSemi44333DSetRectNeighborsAttrib, , UserTile2DSemi43433DSetR
, ,

15.2.367 UserTile2DSemi44333 (tiles2d_sem_reg.c:394)

```
IPObjectStruct *UserTile2DSemi44333(UserTileSemiRegGeomType TileType,
                                    const CagdRType TileSize)
```

TileType: Type of the tile needed, univariate, bivariate or trivariate.

TileSize: The length of side of the polygons.

Returns: A pointer to the basic tile.

Description: Creates a tile object to be used in order to tile a 2D domain with polygon vertex 44333.
See also: UserTile2DSemi6363, UserTile2DSemi848, UserTile2DSemi4346, UserTile2DSemi33336, UserTile2DSemi1246, UserTile2DSemi12312, , ,

15.2.368 UserTile2DSemi44333D (tiles2d_sem_reg_dual.c:1525)

```
IPObjectStruct *UserTile2DSemi44333D(UserTileSemiRegDualGeomType TileType,
                                     const CagdRType TileSize,
                                     const CagdRType Indent,
                                     const CagdRType CrvAmt)
```

TileType: Type of the tile needed, Univariate_Open, univariate closed, bivariate, trivariate.

TileSize: The length of side of the polygons.

Indent: A Fraction (0 - 1) of the length of a primal edge that is in used the dual reconstruction.

CrvAmt: Curvature sharpness of dual reconstructed curves (0 - 1).

Returns: A pointer to the basic tile.

Description: Creates a tile object to be used in order to tile a 2D domain with polygon vertex 44333Dual.
See also: UserTile2DSemi6363D, UserTile2DSemi848D, UserTile2DSemi4346D, , UserTile2DSemi33336D, UserTile2DSemi1246D, UserTile2DSemi12312D, , ,

15.2.369 UserTile2DSemi44333DSetRectNeighborsAttrib (tiles2d_sem_reg_dual.c:1614)

```
void UserTile2DSemi44333DSetRectNeighborsAttrib(IPObjectStruct *RectTile)
```

RectTile: The tile to update.

Returns: void

Description: Updates the elements in a rectangular tile 6363Dual with attribute referencing their previous/edge neighbor in either/both X, Y directions (adjacent in list object of rectangular tiles, share neighbors because original items tiles are split, in order to create a rectangle domain).
See also: UserTile2DSemi44333D, UserTile2DSemi4346DSetRectNeighborsAttrib, , UserTile2DSemi848DSetRectNeighborsAtt
, UserTile2DSemi12312DSetRectNeighborsAttrib, , UserTile2DSemi6363DSetRectNeighborsAttrib, , UserTile2DSemi43433DSetRe
, ,

15.2.370 UserTile2DSemi6363 (tiles2d_sem_reg.c:515)

```
IPObjectStruct *UserTile2DSemi6363(UserTileSemiRegGeomType TileType,
                                   const CagdRType TileSize)
```

TileType: Type of the tile needed, univariate, bivariate or trivariate.

TileSize: The length of side of the polygons.

Returns: A pointer to the basic tile.

Description: Creates a tile object to be used in order to tile a 2D domain with polygon vertex 6363.
See also: UserTile2DSemi44333, UserTile2DSemi848, UserTile2DSemi4346, UserTile2DSemi33336, UserTile2DSemi1246, UserTile2DSemi12312, , ,

15.2.371 UserTile2DSemi6363D (tiles2d_sem_reg_dual.c:2363)

```
IPObjectStruct *UserTile2DSemi6363D(UserTileSemiRegDualGeomType TileType,
                                    const CagdRType TileSize,
                                    const CagdRType Indent,
                                    const CagdRType CrvAmt)
```

TileType: Type of the tile needed, Univariate_Open, univariate closed, bivariate, trivariate.

TileSize: The length of side of the polygons.

Indent: A Fraction (0 - 1) of the length of a primal edge that is in used the dual reconstruction.

CrvAmt: Curvature sharpness of dual reconstructed curves (0 - 1).

Returns: A pointer to the basic tile.

Description: Creates a tile object to be used in order to tile a 2D domain with polygon vertex 6363Dual.
See also: UserTile2DSemi848D, UserTile2DSemi44333D, UserTile2DSemi4346D, , UserTile2DSemi33336D, UserTile2DSemi1246D, UserTile2DSemi12312D, , ,

15.2.372 UserTile2DSemi6363DSetRectNeighborsAttrib (tiles2d_sem_reg_dual.c:2457)

```
void UserTile2DSemi6363DSetRectNeighborsAttrib(IPObjectStruct *RectTile)
```

RectTile: The tile to update.

Returns: void

Description: Updates the elements in a rectangular tile 6363Dual with attribute referencing their previous/edge neighbor in either/both X, Y directions (adjacent in list object of rectangular tiles, share neighbors because original items tiles are split, in order to create a rectangle domain).
See also: UserTile2DSemi6363D, UserTile2DSemi4346DSetRectNeighborsAttrib, , UserTile2DSemi848DSetRectNeighborsAttr
, UserTile2DSemi12312DSetRectNeighborsAttrib, , UserTile2DSemi44333DSetRectNeighborsAttrib, , UserTile2DSemi43433DSetR
, ,

15.2.373 UserTile2DSemi848 (tiles2d_sem_reg.c:635)

```
IPObjectStruct *UserTile2DSemi848(UserTileSemiRegGeomType TileType,
                                  const CagdRType TileSize)
```

TileType: Type of the tile needed, univariate, bivariate or trivariate.

TileSize: The length of side of the polygons.

Returns: A pointer to the basic tile.

Description: Creates a tile object to be used in order to tile a 2D domain with polygon vertex 848.
See also: UserTile2DSemi6363, UserTile2DSemi44333, UserTile2DSemi4346, UserTile2DSemi33336, UserTile2DSemi1246, UserTile2DSemi12312, , ,

15.2.374 UserTile2DSemi848D (tiles2d_sem_reg_dual.c:1215)

```
IPObjectStruct *UserTile2DSemi848D(UserTileSemiRegDualGeomType TileType,
                                   const CagdRType TileSize,
                                   const CagdRType Indent,
                                   const CagdRType CrvAmt)
```

TileType: Type of the tile needed, Univariate closed, univariate open, bivariate, trivariate.

TileSize: The length of side of the polygons.

Indent: A Fraction (0 - 1) of the length of a primal edge that is in used the dual reconstruction.

CrvAmt: Curvature sharpness of dual reconstructed curves (0 - 1).

Returns: A pointer to the basic tile.

Description: Creates a tile object to be used in order to tile a 2D domain with polygon vertex 848Dual.
See also: UserTile2DSemi6363D, UserTile2DSemi44333D, UserTile2DSemi4346D, UserTile2DSemi33336D, UserTile2DSemi1246D, UserTile2DSemi12312D, , ,

15.2.375 UserTile2DSemi848DSetRectNeighborsAttrib (tiles2d_sem_reg_dual.c:1301)

```
void UserTile2DSemi848DSetRectNeighborsAttrib(IPObjectStruct *RectTile)
```

RectTile: The tile to update.

Returns: void

Description: Updates the elements in a rectangular tile 848D with attribute referencing their previous/edge neighbor in either/both X, Y directions (adjacent in list object of rectangular tiles, share neighbors because original items tiles are split, in order to create a rectangle domain).
See also: UserTile2DSemi848D, UserTile2DSemi4346DSetRectNeighborsAttrib, , UserTile2DSemi6363DSetRectNeighborsAttr
, UserTile2DSemi12312DSetRectNeighborsAttrib, , UserTile2DSemi44333DSetRectNeighborsAttrib, , UserTile2DSemi43433DSetR
, ,

15.2.376 UserTileGetSteps (tilepack.c:1231)

```
void UserTileGetSteps(const struct UserTilePackInfoStruct *TilePackInfo,
                      int *StepsMin,
                      int *StepsMax,
                      CagdRType *RelStepsSize)
```

TilePackInfo: A pointer to a tile Info.

StepsMin, StepsMax: To be updated with steps as encoded in TilePackInfo. Must be of size USER_PACK_TILE_MAX_SPA and can be NULL to ignore.

RelStepsSize: To be updated with the relative scale of the current step sizes as encoded in TilePackInfo. Must be of size USER_PACK_TILE_MAX_SPACE_DIM and can be NULL to ignore.

Returns: void

Description: Fetches the range of steps for tilings, in TilePackInfo.
See also: UserTileSetSteps,

15.2.377 UserTileSetSteps (tilepack.c:1273)

```
void UserTileSetSteps(struct UserTilePackInfoStruct *TilePackInfo,
                      const int *StepsMin,
                      const int *StepsMax,
                      const CagdRType *RelStepsSize)
```

TilePackInfo: A pointer to the tile Info.

StepsMin, StepsMax: To set in TilePackInfo with these steps range. Must be of size USER_PACK_TILE_MAX_SPACE_DIM and can be NULL to ignore.

RelStepsSize: Value to specify, to modify (relatively) the step sizes. Must be of size USER_PACK_TILE_MAX_SPACE_DIM and can be NULL to ignore.

Returns: void

Description: Sets the range of steps for tilings, in PackTileInfo.
See also: UserTileGetSteps,

15.2.378 UserTopoAddCell (unstrct_grid.c:1774)

```
CagdBType UserTopoAddCell(UserTopoUnstrctGeomStruct *Ud,
                          const int *PtIdVec,
                          int PtIdVecLen,
                          IPObjectStruct *Cell,
                          int *CellID)
```

Ud: The grid, modified in place.

PtIdVec: The vector IDs' of points in Ud.

PtIdVecLen: The number of points in PtIdVec.

Cell: The Cell to initialize control-points of.

CellID: ID of the cell. Return argument.

Returns: TRUE if successful, FALSE otherwise.

Description: Adds an Cell to the grid. The Cell is initialized with ctrl-pts specified by the vector of identifiers given in array PtIdVec. Each identifier refers to a point in Ud. Also copies PtIdVec to the attribute list of Cell.

15.2.379 UserTopoAddNewCell (unstrct_grid.c:1273)

```
CagdBType UserTopoAddNewCell(UserTopoUnstrctGeomStruct *Ud,
                             const IPObjectStruct *Cell,
                             int UpdateGeom,
                             int *CellID)
```

Ud: Grid to add new cell to, in place.

Cell: New cell to be added to the grid, along with its points.

UpdateGeom: TRUE to also update the entire geometry on this new cell (adjacency information, etc.).

CellID: ID of the cell. Return argument.

Returns: TRUE if successful, FALSE otherwise.

Description: Append the points of the given cell to set of points in Ud and adds the cell to the Ud, in place.

15.2.380 UserTopoAddNewCell2 (unstrct_grid.c:1195)

```
CagdBType UserTopoAddNewCell2(UserTopoUnstrctGeomStruct *Ud,
                              const IPObjectStruct *Cell,
                              int UpdateGeom,
                              int *CellID)
```

Ud: Grid to add new cell to, in place.

Cell: New cell to be added to the grid, along with its points.

UpdateGeom: TRUE to also update the entire geometry on this new cell (adjacency information, etc.).

CellID: ID of the cell. Return argument.

Returns: TRUE if successful, FALSE otherwise.

Description: Append the points of the given cell to set of points in Ud and adds the cell to the Ud, in place. More efficient than UserTopoAddNewCell for large data and many new cells.

15.2.381 UserTopoAddObjectToField (unstrct_grid.c:1805)

```
CagdBType UserTopoAddObjectToField(UserTopoUnstrctGeomStruct *Ud,
                                   IPObjectStruct *IPObj)
```

Ud: The grid, modified in place.

IPObj: The object to append to the field.

Returns: TRUE if successful, FALSE otherwise.

Description: Adds an IRIT object to the field of the grid.

15.2.382 UserTopoAddPoints (unstrct_grid.c:1148)

```
UserTopoUnstrctGeomStruct *UserTopoAddPoints(
                                    const UserTopoUnstrctGeomStruct *Ud,
                                    const UserTopoUnstrctGeomPtStruct *Pts,
                                    int NumPt,
                                    int **RealIDMap)
```

Ud: Grid to add new set of points to.

Pts: The vector of new points.

NumPt: The number of points in Pts.

RealIDMap: Vector of actual IDs assigned to the points.

Returns: Grid with added points.

Description: Append the given points to set of points in Ud, adjusting Id's of new points if necessary.

15.2.383 UserTopoAllEntitiesWithPoint (unstrct_grid.c:4392)

```
int UserTopoAllEntitiesWithPoint(const UserTopoUnstrctGeomStruct *Ud,
                                 int PtId,
                                 int **EntIds)
```

Ud: The grid.

PtId: ID of the point.

EntIds: Vector if IDs' of entities containing the point. Return argument. Caller must free the memory.

Returns: Number of entities.

Description: Returns a vector of IDs' of entities in grid having given point. Caller is owner.

15.2.384 UserTopoAppendUnstrctGeoms (unstrct_grid.c:2475)

```
UserTopoUnstrctGeomStruct *UserTopoAppendUnstrctGeoms(
                                        const UserTopoUnstrctGeomStruct *UdA,
                                        const UserTopoUnstrctGeomStruct *UdB,
                                        CagdRType Eps,
                                        int **RealIDMap)
```

UdA: First grid to merge.

UdB: Second grid to merge.

Eps: Tolerance for computing adjacency relations.

RealIDMap: Vector of actual IDs assigned to the points.

Returns: Merged grid.

Description: Merges two grids, identifying points within specified tolerance. Optionally merges identified points.

15.2.385 UserTopoApplyFilterToGrid (unstrct_grid.c:2910)

```
UserTopoUnstrctGeomStruct *UserTopoApplyFilterToGrid(
                                          const UserTopoUnstrctGeomStruct *Ud,
                                          CagdBType PurgeUnusedPts)
```

Ud: The grid to filter.

PurgeUnusedPts: Whether to purge unused points.

Returns: The filtered grid.

Description: Filters a grid, wherein the callback function UserTopoFilterGridCBFunc applying filter on the points of the grid. Optionally purge unused points.

15.2.386 UserTopoAssignSequentialCellIDs (unstrct_grid.c:2593)

```
UserTopoUnstrctGeomStruct *UserTopoAssignSequentialCellIDs(
                                        const UserTopoUnstrctGeomStruct *Ud)
```

Ud: To assign sequential IDs to cells of.

Returns: New grid.

Description: Assigns sequential IDs to cells of Ud, beginning from 1.

15.2.387 UserTopoAssignSequentialPointIDs (unstrct_grid.c:2542)

```
UserTopoUnstrctGeomStruct *UserTopoAssignSequentialPointIDs(
                                        const UserTopoUnstrctGeomStruct *Ud)
```

Ud: To assign sequential IDs to points of.

Returns: New grid.

Description: Assigns sequential IDs to points of Ud, beginning from 1.

15.2.388 UserTopoCellClosestToPoint (unstrct_grid.c:6206)

```
int UserTopoCellClosestToPoint(const UserTopoUnstrctGeomStruct *Ud,
                               const CagdPType Pt,
                               UserTopoUnstrctGridClosestEntityType
                                                          ClosestEntity,
                               int *FaceIdx,
                               int *EdgeIdx,
                               int *CrnrIdx,
                               int *UVWMax)
```

Ud: The grid whose cells to consider.

Pt: Point to measure distance from.

ClosestEntity: Search closest faces, closest edges (for surfaces and trivariates) or closest corners.

FaceIdx: For a trivariates, returns the index of the face - 1 to 6 for one of (UMin, UMax, VMin, VMax, WMin, WMax).

EdgeIdx: For a surface or a trivariates, returns the index of the edge - 1 to 4 for one of (UMin, UMax, VMin, VMax).

CrnrIdx: Corner index in lexicographic order from (UMin, {VMin, AMax}) to (UMax {, VMax, WMax}). 1 to 8.

UVWMax: A vector of 3 coordinates to set if Pt is near the min (FALSE) Max of the domain.

Returns: ID of the cell of Ud which is closest to Pt.

Description: Return the ID of the cell closest to the given point.

15.2.389 UserTopoCellsAdjacentToCell (unstrct_grid.c:4468)

```
int UserTopoCellsAdjacentToCell(const UserTopoUnstrctGeomStruct *Ud,
                                int CellID,
                                int **EntIDs)
```

Ud: The grid.

CellID: ID of the Cell to return adjacency-list of.

EntIDs: Vector if IDs' of entities adjacent to CellID. Return argument. Caller must free the memory.

Returns: Number of entities, 0 if CellID not found in Ud.

Description: Returns a vector of IDs' of cells in grid which are adjacent to the given Cell. Caller is owner.

15.2.390 UserTopoCrvBndryFilter (unstrct_grid.c:3351)

```
UserTopoUnstrctGeomStruct *UserTopoCrvBndryFilter(
                                        const UserTopoUnstrctGeomStruct *Ud)
```

Ud: The grid to compute boundary of the curves of.

Returns: Grid with end-pts of curves of Ud.

Description: Computes the grid wherein the points are the end-points of the curve of the input grid.

15.2.391 UserTopoGetCellAttrThreshold (unstrct_grid.c:4568)

```
int UserTopoGetCellAttrThreshold(const UserTopoUnstrctGeomStruct *Ud,
                                 char *AttrName,
                                 int AttrMinVal,
                                 int AttrMaxVal,
                                 int **EntIDs)
```

Ud: The grid.

AttrName: Name of the attribute.

AttrMinVal: Minimum value of attribute.

AttrMaxVal: Maximum value of attribute.

EntIDs: Vector if IDs' of entities. Return argument. Caller must free the memory.

Returns: Number of entities.

Description: Returns a vector of IDs' of entities in grid having the value of an attribute between prescribed values. Caller is owner of vector of IDs.

15.2.392 UserTopoGetCellIntAttr (unstrct_grid.c:4144)

```
int UserTopoGetCellIntAttr(const UserTopoUnstrctGeomStruct *Ud,
                           int CellID,
                           char *AttrName)
```

Ud: The grid.

CellID: ID of the Cell.

AttrName: Name of attribute to return.

Returns: Value of the attribute.

Description: Returns the integer attribute of the Cell.

15.2.393 UserTopoGetCellIntAttrVec (unstrct_grid.c:4183)

```
int UserTopoGetCellIntAttrVec(UserTopoUnstrctGeomStruct *Ud,
                              int *CellIDVec,
                              int NumCellID,
                              char *AttrName,
                              int **AttrValueVec)
```

Ud: The grid.

CellIDVec: Vector of IDs of cells to return the attribute of.

NumCellID: Length of CellIDVec.

AttrName: Name of the attribute.

AttrValueVec: Vector of attribute values. Return argument. Caller is owner.

Returns: TRUE is success, FALSE otherwise.

Description: Returns vector of attribute values for the cell-ID vector.

15.2.394 UserTopoGetCellRealAttr (unstrct_grid.c:4212)

```
CagdRType UserTopoGetCellRealAttr(const UserTopoUnstrctGeomStruct *Ud,
                                  int CellID,
                                  char *AttrName)
```

Ud: The grid.

CellID: ID of the Cell.

AttrName: Name of attribute to return.

Returns: Value of the attribute.

Description: Returns the attribute of the Cell.

15.2.395 UserTopoGetCellRealAttrVec (unstrct_grid.c:4251)

```
int UserTopoGetCellRealAttrVec(UserTopoUnstrctGeomStruct *Ud,
                               int *CellIDVec,
                               int NumCellID,
                               char *AttrName,
                               CagdRType **AttrValueVec)
```

Ud: The grid.

CellIDVec: Vector of IDs of cells to return the attribute of.

NumCellID: Length of CellIDVec.

AttrName: Name of the attribute.

AttrValueVec: Vector of attribute values. Return argument. Caller is owner.

Returns: TRUE is success, FALSE otherwise.

Description: Returns vector of attribute values for the cell-ID vector.

15.2.396 UserTopoGetCellStrAttr (unstrct_grid.c:4280)

```
const char *UserTopoGetCellStrAttr(const UserTopoUnstrctGeomStruct *Ud,
                                   int CellID,
                                   char *AttrName)
```

Ud: The grid.

CellID: ID of the Cell.

AttrName: Name of attribute to return.

Returns: Value of the attribute.

Description: Returns the attribute of the Cell.

15.2.397 UserTopoGetCellStrAttrVec (unstrct_grid.c:4319)

```
int UserTopoGetCellStrAttrVec(UserTopoUnstrctGeomStruct *Ud,
                              int *CellIDVec,
                              int NumCellID,
                              char *AttrName,
                              const char ***AttrValueVec)
```

Ud: The grid.

CellIDVec: Vector of IDs of cells to return the attribute of.

NumCellID: Length of CellIDVec.

AttrName: Name of the attribute.

AttrValueVec: Vector of attribute values. Return argument. Caller is owner of vector of strings but not the strings.

Returns: TRUE is success, FALSE otherwise.

Description: Returns vector of attribute values for the cell-ID vector.

15.2.398 UserTopoGetField (unstrct_grid.c:1830)

```
IPObjectStruct *UserTopoGetField(UserTopoUnstrctGeomStruct *Ud)
```

Ud: The grid, to return field of.

Returns: TRUE if successful, FALSE otherwise.

Description: Returns the field of the grid.

15.2.399 UserTopoGetPointAttrThreshold (unstrct_grid.c:4518)

```
int UserTopoGetPointAttrThreshold(const UserTopoUnstrctGeomStruct *Ud,
                                  char *AttrName,
                                  int AttrMinVal,
                                  int AttrMaxVal,
                                  int **PtIDs)
```

Ud: The grid.

AttrName: Name of the attribute.

AttrMinVal: Minimum value of attribute.

AttrMaxVal: Maximum value of attribute.

PtIDs: Vector if IDs' of points. Return argument. Caller must free the memory.

Returns: Number of points.

Description: Returns a vector of IDs' of points in grid having the value of an attribute between prescribed values. Caller is owner of vector of IDs.

15.2.400 UserTopoGetPointIntAttr (unstrct_grid.c:3951)

```
int UserTopoGetPointIntAttr(const UserTopoUnstrctGeomStruct *Ud,
                            int PtId,
                            char *AttrName)
```

Ud: The grid.

PtId: ID of the point to return the attribute of.

AttrName: Name of the attribute.

Returns: Value of the attribute.

Description: Returns the integer attribute of the point.

15.2.401 UserTopoGetPointIntAttrVec (unstrct_grid.c:3987)

```
int UserTopoGetPointIntAttrVec(UserTopoUnstrctGeomStruct *Ud,
                               int *PtIdVec,
                               int NumPtId,
                               char *AttrName,
                               int **AttrValueVec)
```

Ud: The grid.

PtIdVec: Vector of IDs of points to return the attribute of.

NumPtId: Length of PtIdVec.

AttrName: Name of the attribute.

AttrValueVec: Vector of attribute values. Return argument. Caller is owner.

Returns: TRUE is success, FALSE otherwise.

Description: Returns vector of attribute values for the point-ID vector.

15.2.402 UserTopoGetPointRealAttr (unstrct_grid.c:4015)

```
CagdRType UserTopoGetPointRealAttr(const UserTopoUnstrctGeomStruct *Ud,
                                   int PtId,
                                   char *AttrName)
```

Ud: The grid.

PtId: ID of the point to return the attribute of.

AttrName: Name of the attribute.

Returns: Value of the attribute.

Description: Returns the real attribute of the point.

15.2.403 UserTopoGetPointRealAttrVec (unstrct_grid.c:4051)

```
int UserTopoGetPointRealAttrVec(UserTopoUnstrctGeomStruct *Ud,
                                int *PtIdVec,
                                int NumPtId,
                                char *AttrName,
                                CagdRType **AttrValueVec)
```

Ud: The grid.

PtIdVec: Vector of IDs of points to return the attribute of.

NumPtId: Length of PtIdVec.

AttrName: Name of the attribute.

AttrValueVec: Vector of attribute values. Return argument. Caller is owner.

Returns: TRUE is success, FALSE otherwise.

Description: Returns vector of attribute values for the point-ID vector.

15.2.404 UserTopoGetPointStrAttr (unstrct_grid.c:4080)

```
const char *UserTopoGetPointStrAttr(const UserTopoUnstrctGeomStruct *Ud,
                                    int PtId,
                                    char *AttrName)
```

Ud: The grid.

PtId: ID of the point to return the attribute of.

AttrName: Name of the attribute.

Returns: Value of the attribute.

Description: Returns the string attribute of the point.

15.2.405 UserTopoGetPointStrAttrVec (unstrct_grid.c:4116)

```
int UserTopoGetPointStrAttrVec(UserTopoUnstrctGeomStruct *Ud,
                               int *PtIdVec,
                               int NumPtId,
                               char *AttrName,
                               const char ***AttrValueVec)
```

Ud: The grid.

PtIdVec: Vector of IDs of points to return the attribute of.

NumPtId: Length of PtIdVec.

AttrName: Name of the attribute.

AttrValueVec: Vector of attribute values. Return argument. Caller is the owner of the vector of strings, but not the stings.

Returns: TRUE is success, FALSE otherwise.

Description: Returns vector of attribute values for the point-ID vector.

15.2.406 UserTopoIdToObject (unstrct_grid.c:3864)

```
IPObjectStruct *UserTopoIdToObject(const UserTopoUnstrctGeomStruct *Ud,
                                   int Id)
```

Ud: The grid.

Id: ID of the Cell to return.

Returns: The Cell in grid if found, NULL otherwise.

Description: Returns the object in the grid with given ID.

15.2.407 UserTopoMergePoints (unstrct_grid.c:863)

```
UserTopoUnstrctGeomStruct *UserTopoMergePoints(
                                        const UserTopoUnstrctGeomStruct *Ud,
                                        CagdRType Eps,
                                        CagdBType IdentifyNoMerge,
                                        const CagdBType *MergePtIndices,
                                        int **MergedIDMap,
                                        int *MergeIDMapLen)
```

Ud: Input grid to merge points of.

Eps: Tolerance up to which to merge points.

IdentifyNoMerge: If TRUE, only mark the identical points without actually merging them.

MergePtIndices: Vector of flags indicating if a point is to be considered for merging. The ith point is considered for merging only if ith value in the vector MergePtIndices is TRUE. NULL will consider all points.

MergedIDMap: Vector of pairs of IDs of points which are merged. Length of this vector is twice the number of points which were merged, i.e., 2 * (#pts in Ud - #pts in returned grid).

MergeIDMapLen: Length of vector MergedIDMap.

Returns: Grid with merged points.

Description: Merges points of input grid, which are identical upto the supplied tolerance. ith point of the grid is considered for merging only if ith bit in the input vector is TRUE.

15.2.408 UserTopoModifyPoint (unstrct_grid.c:1853)

```
int UserTopoModifyPoint(UserTopoUnstrctGeomStruct *Ud,
                        int PtId,
                        const UserTopoUnstrctGeomPtStruct *Pt)
```

Ud: The grid, modified in place.

PtId: ID of the point to modify.

Pt: New coordinates of the point.

Returns: TRUE if point found in the grid, FALSE otherwise.

Description: Modifies a point in the grid, in place. Updates all entities which contain this point.

15.2.409 UserTopoNumOfEntOfType (unstrct_grid.c:4436)

```
int UserTopoNumOfEntOfType(const UserTopoUnstrctGeomStruct *Ud,
                           IPObjStructType Type)
```

Ud: The grid.

Type: Type of Cell.

Returns: Number of entities.

Description: The number of entities in the grid of given type.

15.2.410 UserTopoObjectToId (unstrct_grid.c:3832)

```
int UserTopoObjectToId(const UserTopoUnstrctGeomStruct *Ud,
                       const IPObjectStruct *Cell)
```

Ud: The grid.

Cell: Cell whose ID to return.

Returns: ID of the Cell if found in the grid, -1 if not found.

Description: Returns the ID of the object in the grid.

15.2.411 UserTopoPolylineSelector (unstrct_grid.c:6087)

```
int UserTopoPolylineSelector(const UserTopoUnstrctGeomStruct *Ud,
                             const IPPolygonStruct *Poly,
                             const CagdPType Dir,
                             int **CellIDVec)
```

Ud: The grid to select cells of.

Poly: The polygon to use for selection.

Dir: Direction to use for frustum.

CellIDVec: Vector of IDs of cells selected. User is owner.

Returns: Number of cells selected.

Description: Returns IDs of all cells which intersect with the frustum of the polyline in the given direction .

15.2.412 UserTopoPtsOfCell (unstrct_grid.c:4349)

```
int UserTopoPtsOfCell(const UserTopoUnstrctGeomStruct *Ud,
                      int EntId,
                      int **PtIds)
```

Ud: The grid.

EntId: ID of the Cell.

PtIds: Vector if IDs' of points in Cell. Caller must free. Return argument.

Returns: Number of points in Cell, -1 if Cell not found in grid.

Description: Returns a vector of IDs' of points in Cell. Caller is owner.

15.2.413 UserTopoPtsOfCellsWithAttrib (unstrct_grid.c:959)

```
void UserTopoPtsOfCellsWithAttrib(const UserTopoUnstrctGeomStruct *Ud,
                                  UserTopoUnstrctGridAttrType AttrType,
                                  int NumAttr,
                                  const int *NumAttrVals,
                                  const char **AttrNames,
                                  const void *AttrVals,
                                  CagdBType **PtIndxVec)
```

Ud: Input grid to extract points of.

AttrType: Type of attribute.

NumAttr: Number of attributes.

NumAttrVals: Vector containing number of values for each attribute.

AttrNames: Vector of names of attributes.

AttrVals: Array of attribute-values of type specified by AttrType.

PtIndxVec: Vector of length equal to number of points in Ud. The ith element is TRUE if the ith point of Ud is extracted. Return argument. Can be NULL, in which case it is allocated.

Returns: void

Description: Extracts the points of input grid which belong to cells having any of the given list of attributes.

15.2.414 UserTopoPurgeUnusedPts (unstrct_grid.c:2823)

```
UserTopoUnstrctGeomStruct *UserTopoPurgeUnusedPts(
                                        const UserTopoUnstrctGeomStruct *Ud)
```

Ud: Grid to purge points in.

Returns: Grid with unused points purged.

Description: Discards points not used by any cell. Does NOT recompute adjacency relations. User must recompute them.

15.2.415 UserTopoReadGridFromFile (unstrct_grid.c:6542)

```
UserTopoUnstrctGeomStruct *UserTopoReadGridFromFile(const char *FileName)
```

FileName: Name of the file to read from.

Returns: Grid read from file, NULL if error.

Description: Read a grid from a file.

15.2.416 UserTopoSetCellIntAttr (unstrct_grid.c:2176)

```
int UserTopoSetCellIntAttr(UserTopoUnstrctGeomStruct *Ud,
                           IPObjectStruct *Cell,
                           const char *AttrName,
                           int AttrValue)
```

Ud: The grid, modified in place.

Cell: Cell to set attribute of, in place.

AttrName: Name of the attribute.

AttrValue: Value of the attribute.

Returns: Return 1 if successful, 0 otherwise.

Description: Sets an attribute for the Cell.

15.2.417 UserTopoSetCellIntAttrVec (unstrct_grid.c:2215)

```
int UserTopoSetCellIntAttrVec(UserTopoUnstrctGeomStruct *Ud,
                              const int *CellIDVec,
                              int NumCellID,
                              const char *AttrName,
                              const int *AttrValueVec,
                              int NumVals)
```

Ud: The grid, modified in place.

CellIDVec: Vector of IDs of cells to set attribute of, in place.

NumCellID: Length of CellIDVec.

AttrName: Name of the attribute.

AttrValueVec: Vector of values of the attribute.

NumVals: Length of AttrValueVec.

Returns: Return 1 if successful, 0 otheriwse.

Description: Sets attributes for the vector of Cell IDs.

15.2.418 UserTopoSetCellRealAttr (unstrct_grid.c:2275)

```
int UserTopoSetCellRealAttr(UserTopoUnstrctGeomStruct *Ud,
                            IPObjectStruct *Cell,
                            const char *AttrName,
                            CagdRType AttrValue)
```

Ud: The grid, modified in place.

Cell: Cell to set attribute of, in place.

AttrName: Name of the attribute.

AttrValue: Value of the attribute.

Returns: Return 1 if successful, 0 otherwise.

Description: Sets an attribute for the Cell.

15.2.419 UserTopoSetCellRealAttrVec (unstrct_grid.c:2314)

```
int UserTopoSetCellRealAttrVec(UserTopoUnstrctGeomStruct *Ud,
                               const int *CellIDVec,
                               int NumCellID,
                               const char *AttrName,
                               const CagdRType *AttrValueVec,
                               int NumVals)
```

Ud: The grid, modified in place.

CellIDVec: Vector of IDs of cells to set attribute of, in place.

NumCellID: Length of CellIDVec.

AttrName: Name of the attribute.

AttrValueVec: Vector of values of the attribute.

NumVals: Length of AttrValueVec.

Returns: Return 1 if successful, 0 otherwise.

Description: Sets attributes for the vector of Cell IDs.

15.2.420 UserTopoSetCellStrAttr (unstrct_grid.c:2374)

```
int UserTopoSetCellStrAttr(UserTopoUnstrctGeomStruct *Ud,
                           IPObjectStruct *Cell,
                           const char *AttrName,
                           const char *AttrValue)
```

Ud: The grid, modified in place.

Cell: Cell to set attribute of, in place.

AttrName: Name of the attribute.

AttrValue: Value of the attribute.

Returns: Return 1 if successful, 0 otherwise.

Description: Sets an attribute for the Cell.

15.2.421 UserTopoSetCellStrAttrVec (unstrct_grid.c:2413)

```
int UserTopoSetCellStrAttrVec(UserTopoUnstrctGeomStruct *Ud,
                              const int *CellIDVec,
                              int NumCellID,
                              const char *AttrName,
                              const char **AttrValueVec,
                              int NumVals)
```

Ud: The grid, modified in place.

CellIDVec: Vector of IDs of cells to set attribute of, in place.

NumCellID: Length of CellIDVec.

AttrName: Name of the attribute.

AttrValueVec: Vector of values of the attribute.

NumVals: Length of AttrValueVec.

Returns: Return 1 if successful, 0 otherwise.

Description: Sets attributes for the vector of Cell IDs.

15.2.422 UserTopoSetFilterGridCallBackFunc (unstrct_grid.c:2879)

```
UserTopoFilterGridCBFuncType UserTopoSetFilterGridCallBackFunc(
                                        UserTopoFilterGridCBFuncType NewFunc)
```

NewFunc: New function to set.

Returns: The previous function.

Description: Sets a callback function for filtering points in a UG.

15.2.423 UserTopoSetPointIntAttr (unstrct_grid.c:1896)

```
int UserTopoSetPointIntAttr(UserTopoUnstrctGeomStruct *Ud,
                            int PtId,
                            const char *AttrName,
                            int AttrValue)
```

Ud: The grid, modified in place.

PtId: ID of the point to set attribute of.

AttrName: Name of the attribute.

AttrValue: Value of the attribute.

Returns: Return 1 if successful, 0 otherwise.

Description: Sets an attribute for the point.

15.2.424 UserTopoSetPointIntAttrVec (unstrct_grid.c:1933)

```
int UserTopoSetPointIntAttrVec(UserTopoUnstrctGeomStruct *Ud,
                               const int *PtIdVec,
                               int NumPtId,
                               const char *AttrName,
                               const int *AttrValueVec,
                               int NumVals)
```

Ud: The grid, modified in place.

PtIdVec: Vector of IDs of points to set attribute of.

NumPtId: Length of PtIdVec.

AttrName: Name of the attribute.

AttrValueVec: Vector of values of the attribute.

NumVals: Number of values.

Returns: Return 1 if successful, 0 otherwise.

Description: Sets attributes for vector of points.

15.2.425 UserTopoSetPointRealAttr (unstrct_grid.c:1990)

```
int UserTopoSetPointRealAttr(UserTopoUnstrctGeomStruct *Ud,
                             int PtId,
                             const char *AttrName,
                             CagdRType AttrValue)
```

Ud: The grid, modified in place.

PtId: ID of the point to set attribute of.

AttrName: Name of the attribute.

AttrValue: Value of the attribute.

Returns: Return 1 if successful, 0 otherwise.

Description: Sets an attribute for the point.

15.2.426 UserTopoSetPointRealAttrVec (unstrct_grid.c:2027)

```
int UserTopoSetPointRealAttrVec(UserTopoUnstrctGeomStruct *Ud,
                                const int *PtIdVec,
                                int NumPtId,
                                const char *AttrName,
                                const CagdRType *AttrValueVec,
                                int NumVals)
```

Ud: The grid, modified in place.

PtIdVec: Vector of IDs of points to set attribute of.

NumPtId: Length of PtIdVec.

AttrName: Name of the attribute.

AttrValueVec: Vector of values of the attribute.

NumVals: Length of AttrValueVec.

Returns: Return 1 if successful, 0 otherwise.

Description: Sets attributes for vector of points.

15.2.427 UserTopoSetPointStrAttr (unstrct_grid.c:2083)

```
int UserTopoSetPointStrAttr(UserTopoUnstrctGeomStruct *Ud,
                            int PtId,
                            const char *AttrName,
                            const char *AttrValue)
```

Ud: The grid, modified in place.

PtId: ID of the point to set attribute of.

AttrName: Name of the attribute.

AttrValue: Value of the attribute.

Returns: Return 1 if successful, 0 otherwise.

Description: Sets an attribute for the point.

15.2.428 UserTopoSetPointStrAttrVec (unstrct_grid.c:2120)

```
int UserTopoSetPointStrAttrVec(UserTopoUnstrctGeomStruct *Ud,
                               const int *PtIdVec,
                               int NumPtId,
                               const char *AttrName,
                               const char **AttrValueVec,
                               int NumVals)
```

Ud: The grid, modified in place.

PtIdVec: Vector of IDs of points to set attribute of.

NumPtId: Length of PtIdVec.

AttrName: Name of the attribute.

AttrValueVec: Vector of values of the attribute.

NumVals: Length of AttrValueVec.

Returns: Return 1 if successful, 0 otherwise.

Description: Sets attributes for vector of points.

15.2.429 UserTopoSetPoints (unstrct_grid.c:1725)

```
CagdBType UserTopoSetPoints(UserTopoUnstrctGeomStruct *Ud,
                            UserTopoUnstrctGeomPtStruct *Pts,
                            int NumPt)
```

Ud: Grid to set points of.

Pts: The vector of points.

NumPt: The number of points.

Returns: TRUE if successful, FALSE if duplicate IDs found.

Description: Sets the points of the grid, using the points vector inplace.

15.2.430 UserTopoSrfBndryFilter (unstrct_grid.c:3593)

```
UserTopoUnstrctGeomStruct *UserTopoSrfBndryFilter(
                                        const UserTopoUnstrctGeomStruct *Ud)
```

Ud: The grid to compute boundary of the surfaces of.

Returns: Grid with boundary curves of surfaces of Ud.

Description: Computes the grid wherein the points belong to the boundary curves of the surfaces of the input grid.

15.2.431 UserTopoTrivBndryFilter (unstrct_grid.c:3674)

```
UserTopoUnstrctGeomStruct *UserTopoTrivBndryFilter(
                                        const UserTopoUnstrctGeomStruct *Ud)
```

Ud: The grid to compute boundary of the trivars of.

Returns: Grid with boundary srfs of surfaces of Ud.

Description: Computes the grid wherein the points belong to the boundary srfs of the trivars of the input grid.

15.2.432 UserTopoUDData (unstrct_grid.c:3903)

```
void UserTopoUDData(const UserTopoUnstrctGeomStruct *Ud,
                    UserTopoUnstrctGeomReturnStruct *Data)
```

Ud: The grid.

Data: To return the data in.

Returns: void

Description: Returns the data of the grid with given ID.

15.2.433 UserTopoUnstrctGeomFree (unstrct_grid.c:1662)

```
void UserTopoUnstrctGeomFree(UserTopoUnstrctGeomStruct *Ud)
```

Ud: Grid to free.

Returns: void

Description: Frees a UserTopoUnstrctGeomStruct grid.
See also: UserTopoUnstrctGeomNew,

15.2.434 UserTopoUnstrctGeomMain (unstrct_grid.c:7302)

```
UserTopoUnstrctGeomReturnStruct *UserTopoUnstrctGeomMain(
                                    UserTopoUnstructGridOpType OperationID,
                                    UserTopoUnstrctGeomParamStruct *Params)
```

OperationID: ID to select the required functionality.

Params: Parameters of this specific operation (& return val) - USER_UG_CREATE - None. Returns the ID of the newly created UG. USER_UG_FREE - ID of the UG to free. Returns Success-Flag, i.e., TRUE if successful, FALSE otherwise. USER_UG_SET_POINTS - ID of UG, List of Pts, List of Pt-IDs. Returns Success- Flag, vector of actual IDs assigned to pts in grid. USER_UG_ADD_POINTS - ID of UG, List of Pts, List of Pt-IDs. Returns Success- Flag, ID of new UG, number of points in grid, vector of actual IDs assigned to pts in grid. USER_UG_EXTRACT_POINTS - ID of UG, attribute-type, attribute-names attribute-values. Returns binary vector with 1s' for selected points. USER_UG_MERGE_POINTS - ID of UG, flag to identify points without merging, binary vector with 1s' for points to consider for merge. Returns Success- Flag, ID of new UG, number of points in new UG, vector indicating which points were identified. USER_UG_MODIFY_POINT - ID of UG, ID of Point to modify, new coordinates of point. Returns Success- Flag. USER_UG_ADD_CELL - ID of UG, Cell, List of point IDs. Returns Success- Flag, ID of Cell. USER_UG_APPEND_UG - ID of UG1, ID of UG2. Returns Success- Flag, ID of new UG, number of points in grid, vector of IDs assigned to points of second grid. USER_UG_UPDATE - ID of UG, Tolerance for merging identical entities. Returns Success- Flag. USER_UG_PURGE_UNUSED - ID of UG. Returns Success- Flag, ID of new UG, number of points in new UG. USER_UG_ID_TO_CELL - ID of UG, ID of Cell. Returns Success- Flag, Cell. USER_UG_CELL_TO_ID - ID of UG, Cell. Returns Success- Flag, ID of Cell. USER_UG_SET_FILTER_CB_FUNC - Not implemented. USER_UG_FILTER_GRID - Not implemented. USER_UG_DATA - ID of UG USER_UG_CRV_BNDRY_FILTER - ID of UG to filter. Returns Success-flag, ID of new UG, number of pts in grid. USER_UG_SRF_BNDRY_FILTER - ID of UG to filter. Returns Success-flag, ID of new UG (external bndry), ID of new UG (internal bndry), ID of new UG (internal bndry, different function spaces), number of pts in grid. USER_UG_TRIV_BNDRY_FILTER - ID of UG to filter. Returns Success-flag, ID of new UG (external bndry), ID of new UG (internal bndry), ID of new UG (internal bndry, different function spaces), number of pts in grid. USER_UG_SET_POINT_ATTR - ID of UG, attribute-type, attribute name, list of point-ids, list of attribute values. Returns Success-flag. USER_UG_GET_POINT_ATTR - ID of UG, attribute-type, attribute name, list of point-ids. Returns Success-flag, list of attribute values. USER_UG_SET_CELL_ATTR - ID of UG, attribute-type, attribute name, list of cell-ids, list of attribute values. Returns Success-flag. USER_UG_GET_CELL_ATTR - ID of UG, attribute-type, attribute name, list of cell-ids. Returns Success-flag, list of attribute values. USER_UG_GET_ADJACENCY_LIST - ID of UG, ID of cell. Returns Success-flag, list of cell-ids adjacent to given cell. USER_UG_GET_PTID_CELLID_LIST - ID of UG. Returns Success-flag, list of point-IDs, list of cell-IDs of UG. USER_UG_GET_CELL_PTID_LIST - ID of UG, ID of cell. Returns a list (CellID, CellType, list of IDs of points in cell). USER_UG_GET_UG_POINT_LIST - ID of UG. Returns a list of all points in cell, including their IDs. USER_UG_SEQ_POINT_IDS - ID of UG. Returns Success-flag, ID of new UG, number of points in new UG. USER_UG_SEQ_CELL_IDS - ID of UG. Returns Success-flag, ID of new UG, number of points in new UG. USER_UG_ADD_OBJECT_TO_FIELD - ID of UG, Object to add. Returns Success-flag. USER_UG_GET_FIELD - ID of UG. Returns Success-flag, field as an object-list. USER_UG_ADD_NEW_CELL - ID of UG, cell to add. Returns Success-flag, number of points in UG, Cell Id. USER_UG_GET_BEZIER_PATCHES - ID of UG. Returns Success-flag, ID of new UG, number of points in new UG. Converts all B-spline patches to Bezier. USER_UG_REFINE_CELLS - ID of UG, Cell-Id to refine, direction to refine, parameter values to refine at, number of values. Returns Success-flag, ID of new UG, number of points in new UG. USER_UG_PT_CELL_SELECTOR - ID of UG, point, Closest Entity (closest face for 1, closest edge for 2, and closest corner for 2). Optionally, can have a fourth parameter with an attribute to place on the selected entity, as "list(attrName, AttrVal)". Returns Success-flag, ID of selected cell. USER_UG_POLY_CELL_SELECTOR - ID of UG, polyline, direction. Returns Success-flag, IDs of cells selected, number of cell IDs. USER_UG_REFINE_CELLS_AT_PT - ID of UG, RefRatio, RefLen, Pt, Div1, Div2, Div3. Div1/2/3 refer to divisions to refine at for closest entity to Pt. Div1 must be positive. If Div2 < 0, closest edge is considered and refined Div1 times. If Div2 > 0 and Div3 < 0, closest face is considered and refined Div1 x Div2 times. If Div2 > 0 and Div3 > 0, closest volume (trivariate) is considered and refined Div1 x Div2 x Div3 times. RefRatio sets the size ratios between the first division and last (set to one for similar sizes). RefLen defines the desired default edge length after refinement. Returns Success-flag, ID of new UG, number of points in new UG. USER_UG_REFINE_UNREF_CELLS - ID of UG, RefSize. Returns a globally refined grid of all unrefined so far cells in all dirs. Cells are refined in Dir so the refinement will yield sizes (in Euclidean space) smaller than RefSize. If RefSize is zero, 5% of the bbox of the input geometry is set to RefSize. If RefSize is negative, -Refsize of the bbox of the input geometry will be used as RefSize. USER_UG_WRITE_GRID_TO_FILE - ID of UG, file-name (NULL for stdout). Returns Success-flag. USER_UG_READ_GRID_FROM_FILE - File-name to read from. Returns Success-flag, ID of new UG, number of points in new UG. USER_UG_GET_LINEAR_PATCHES - ID of UG. Returns Success-flag, ID of new UG, number of points in new UG. Converts all patches to linear Bezier patches. An approximation of the input UG patches.

Returns: Return parameters.

Description: Main function for invoking required functionalities for the unstructured grid.

15.2.435 UserTopoUnstrctGeomNew (unstrct_grid.c:1602)

```
UserTopoUnstrctGeomStruct *UserTopoUnstrctGeomNew(void)
```

Returns: Newly allocated structure.

Description: Allocates a new UserTopoUnstrctGeomStruct.
See also: UserTopoUnstrctGeomFree, UserTopoUnstrctGeomNew2,

15.2.436 UserTopoUnstrctGeomPtCopyData (unstrct_grid.c:1638)

```
void UserTopoUnstrctGeomPtCopyData(UserTopoUnstrctGeomPtStruct *Dest,
                                   const UserTopoUnstrctGeomPtStruct *Src)
```

Dest: Destination point.

Src: Source point.

Returns: void

Description: Copies the data from source point to destination point, without allocating any new memory.

15.2.437 UserTopoUnstrctGeomUpdate (unstrct_grid.c:3812)

```
void UserTopoUnstrctGeomUpdate(UserTopoUnstrctGeomStruct **Ud,
                               CagdRType Eps)
```

Ud: The grid.

Eps: Tolerance value.

Returns: void

Description: Computes the adjacency relations between all the entities in the grid, upto specified tolerance. Purges unused points if requested, in place.

15.2.438 UserTopoWriteGridToFile (unstrct_grid.c:6405)

```
CagdBType UserTopoWriteGridToFile(const UserTopoUnstrctGeomStruct *Ud,
                                  const char *FileName)
```

Ud: The grid to write.

FileName: Name of the file to write to. NULL or zero length string for stdout.

Returns: TRUE if written successfully, FALSE otherwise.

Description: Write the input grid to the file on disk.

15.2.439 UserTrivarZeros (tv0jacob.c:95)

```
IPObjectStruct *UserTrivarZeros(const TrivTVStruct *TV,
                                const TrivTVStruct *TVEuclidean,
                                int SkipRate,
                                const CagdRType Fineness[3])
```

TV: Trivariate function to approximate its zero set.

TVEuclidean: If provided, use this trivariate to evaluate into Euclidean space.

SkipRate: Of data in the volume. 1 skips nothing, 2 every second, etc.

Fineness: Of trivariate global refinment level, 0 for no ref., in all three axes.

Returns: The approximation of the zeros of the trivariate.

Description: Approximate the zero set of a trivariate function.
See also: UserTVZeroJacobian,

15.2.440 UserTrussCleanBeamICrvs (truss_base.c:2565)

```
void UserTrussCleanBeamICrvs(UserTrussClipAgainstBeamsInfoStruct *ClipInfo)
```

ClipInfo: The clipping information structure to be filled.

Returns: void

Description: Prepare the clipping information for clipping the beams of the truss lattice. This function computes the intersections using the SSI algorithm, which is slow, but supports beams of arbitrary shape.
See also: UserTrussPrepBeamICrvsFast, UserTrussPrepBeamICrvsMatched, , UserTrussCleanBeamICrvs,

15.2.441 UserTrussClipComp (truss_base.c:863)

```
TrimSrfStruct *UserTrussClipComp(const CagdSrfStruct *CompSrf,
                                 const UserTrussClippingInfoStruct *ClipInfo,
                                 CagdCrvStruct *TopLevelCrv,
                                 CagdBType MarkSegs,
                                 CagdBType BndrySegsOnly,
                                 const UserTrussTolerancesStruct *Tol)
```

CompSrf: The surface to clip.

ClipInfo: The information needed to clip the component.

TopLevelCrv: The top-level trimming loop for the component. If NULL, a trimming loop containing the entire domain will be added.

MarkSegs: A flag indicating whether to add subdivision markers (as attributes) for the trimming curve. Used for creating matching trimming curves for model construction.

BndrySegsOnly: A flag indicating whether the trimming curve subdivision markers are required only on the boundary of the domain of the surface. Ignored if MarkSegs is FALSE.

Tol: The tolerances structure of the truss lattice.

Returns: A trimmed surface representation of the clipped component.

Description: Clip a component of the truss lattice (can be beam, sphere, the side plane between the beams, or a base-cap of a beam), according to the logic prescribed by ClipInfo.
See also: UserTrussClippingInfoStruct, UserTrussPrepBeamICrvs, , UserTrussPrepBeamICrvsFast, UserTrussPrepBeamICrvsMatched,

15.2.442 UserTrussCloseOneCrvLoopOppEdges (truss_base.c:1961)

```
CagdCrvStruct *UserTrussCloseOneCrvLoopOppEdges(const CagdCrvStruct *Crv,
                                                const CagdRType SrfDMin[2],
                                                const CagdRType SrfDMax[2],
                                                CagdBType AddOffet,
                                                CagdRType Offset)
```

Crv: The curve to close.

SrfDMin, SrfDMax: The boundaries of the parametric domain of the surface.

AddOffet: If TRUE, the intersection loop is extended by Offset outside the domain of the surface.

Offset: The offset by which to extend the loops (if AddOffset is TRUE).

Returns: The two intersection loops, if formed successfully. Otherwise, NULL.

Description: Attempts to from a pair of close loops out of a curve in the parametric space of a surface, by connecting its start and end points to opposing edges of the boundary of the surface. If the curve does not start and end at opposing edges, the function returns NULL.
See also: UserTrussICrvsCloseLoops, UserTrussCloseOneCrvLoopEdge, , UserTrussCloseOneCrvLoopCorner,

15.2.443 UserTrussComputeSideSrfs (truss_triv.c:331)

```
CagdSrfStruct *UserTrussComputeSideSrfs(const CagdSrfStruct *BeamSrf,
                                        const CagdCrvStruct *ICrvs,
                                        const IrtPlnType Pln1,
                                        const IrtPlnType Pln2)
```

BeamSrf: The outer surface of a half-beam.

ICrvs: The inersection contour of the current half-beam with another half-beam.

Pln1: The base plane of the current half-beam.

Pln2: The base plane of the second half-beam.

Returns: The side surface(s) between the current half-beam and the second.

Description: Constructs the side surface of a half-beam, given its base plane, and the plane of another half-beam that intersects it. The side surface may be split due to the intersection occurring at the start of the parametric domain of the half-beam.

15.2.444 UserTrussConstructLatticeMain (truss_gen.c:905)

truss lattice

sphere packing

```
IPObjectStruct *UserTrussConstructLatticeMain(
                              const IPObjectStruct *InObj,
                              UserTrussSpherePackParamsStruct *SpherePackParams,
                              UserTrussLatticeParamsStruct *TrussParams,
                              const UserTrussTolerancesStruct *Tol)
```

InObj: A surface or polygonal model to fill with a truss lattice via sphere packing, or a list of points from which to construct the truss lattice.

SpherePackParams: The parameters for the sphere packing. Ignored if InObj is a list of points.

TrussParams: The parameters for constructing the truss lattice. connected.

Tol: The tolerances structure of the truss lattice.

Returns: The resulting truss lattice.

Description: Constructs a truss lattice either from a set of points, or via sphere packing. Additionally, a shell can be provided, which will be used for guaranteeing that all the beams are inside the shell, or it can be connected to the lattice by beams.

15.2.445 UserTrussCreateShellSrfConnections (truss_base.c:3322)

```
CagdSrfStruct *UserTrussCreateShellSrfConnections(
                                    UserTrussNodeDefStruct *NodeDefs,
                                    const CagdPType *ShellPts,
                                    const CagdRType *ShellDists,
                                    int NumPts,
                                    CagdRType DistToConnect,
                                    UserTrussNodeDefCallbacksStruct *Callbacks,
                                    const CagdSrfStruct *ShellSrf,
                                    TrimSrfStruct **TrimmedShellSrf,
                                    const UserTrussTolerancesStruct *Tol)
```

NodeDefs: The lattice node definitions. Extra beams will be added to the nodes that are close to the shell surface.

ShellPts: The closest point on the shell surface for each lattice node.

ShellDists: The minimum distance of each lattice node to the shell surface.

NumPts: The number of lattice nodes.

DistToConnect: The maximum distance at which a lattice node will be connected to the shell surface.

Callbacks: The struct that contains the callbacks for setting the properties of the lattice beams.

ShellSrf: The shell surface.

TrimmedShellSrf: Output parameter: the shell surface, trimmed to connect to the fillets.

Tol: The tolerances structure of the truss lattice.

Returns: The fillets that connect the shell surface to the beams of the truss lattice, as a list of surfaces.

Description: Connects a truss lattice to a shell surface. This requires several actions:

1. For each lattice node that is close to the shell, the beam that will connect it to the shell is added.
2. The fillets that connect the shell surface to the beams are constructed.
3. The shell surface is trimmed, to created holes where the fillets are constructed. This function modifies the array of UserTrussNodeDefStructs to include the new beams, returns the trimmed shell surface via an output parameter, and returns the fillets are a list of surfaces.

See also: UserTrussPrepareBeamInfoWithShell,

15.2.446 UserTrussDefaultTol (truss_base.c:3891)

```
void UserTrussDefaultTol(UserTrussTolerancesStruct *Tol)
```

Tol: The tolerances structure in which the values will be set.

Returns: void

Description: Sets the default tolerances in a UserTrussTolerancesStruct. These tolerances are useful for most cases.

15.2.447 UserTrussFilterShortCrvs (truss_base.c:2802)

```
void UserTrussFilterShortCrvs(CagdCrvStruct **Crvs,
                              CagdRType Threshold,
                              CagdBType SegOnly,
                              CagdBType WarnOnly)
```

Crvs: The list of curves to filter.

Threshold: The threshold for filtering out short curves.

SegOnly: If TRUE, linear curves are skipped.

WarnOnly: If TRUE, short curves are not actually deleted, and instead a warning is printed. Used for debugging.

Returns: void

Description: Filters out curves shorter than some threshold, out of a given list of curves. The length is approximated by the length of the control polyline. Computed in place.

15.2.448 UserTrussICrvsCloseLoops (truss_base.c:1508)

```
void UserTrussICrvsCloseLoops(const CagdSrfStruct *OrigSrf,
                              CagdCrvStruct **ICrvs,
                              const CagdRType SrfDMin[2],
                              const CagdRType SrfDMax[2],
                              const IrtPlnType Pln,
                              const CagdPType CylinderBasePt,
                              const CagdVType CylinderVec,
                              CagdRType CylinderR,
                              CagdBType UseCylinder,
                              CagdRType Tol,
                              CagdBType AddOffset,
                              CagdRType Offset)
```

OrigSrf: The surface which contains the intersection curves.

ICrvs: The list of intersetion curves to be closed into loops. Used in-place.

SrfDMin, SrfDMax: The boundaries of the parametric domain of the surface.

Pln: The plane below which the intersection loops should be. Used only if UseCylinder is FALSE.

CylinderBasePt: The center of the base of the cylinder outside which the intersection loops should be. Used only if UseCylinder is TRUE.

CylinderVec: The direction vector of the cylinder. Used only if UseCylinder is TRUE.

CylinderR: The radius of the cylinder. Used only if UseCylinder is TRUE.

UseCylinder: If TRUE, the cylinder information will be used for closing the loops. Othewise, the plane is used.

Tol: The tolerances for testing on which side of the plane the loop is.

AddOffset: If TRUE, the intersection loop is extended by Offset outside the domain of the surface. Used for improving the robustness of the boolean operations on the loops.

Offset: The offset by which to extend the loops (if AddOffset is TRUE).

Returns: void

Description: Given a surface and a list of intersection curves, closes the curves which start and end at the boundary, into loops. The closing of the loops is performed as to guarantee that the trimming loop (as embedded on the surface in E3 space) are ouside some given cylinder or below plane. Curves which are already closed are ignored.
See also: UserTrussCloseOneCrvLoopEdge, UserTrussCloseOneCrvLoopCorner, , UserTrussCloseOneCrvLoopOppEdges,

15.2.449 UserTrussLatticeWithQualityInfo (truss_gen.c:321)

truss lattice

sphere packing

```
IPObjectStruct *UserTrussLatticeWithQualityInfo(
                                    const CagdPType *Pts,
                                    int NumPts,
                                    CagdRType DistToConnect,
                                    UserTrussNodeDefCallbacksStruct *Callbacks,
                                    const CagdRType *QuantizationVector,
                                    const UserTrussTolerancesStruct *Tol)
```

Pts: The center points of the lattice nodes.

NumPts: The number of lattice nodes.

DistToConnect: The maximum distance at which points will be connected.

Callbacks: The struct that contains the callbacks for setting the properties of the lattice beams.

QuantizationVector: The quantization vector for estimating the lattice quality. Must contain values between 0 and 1, and end with 1. The values indicate the range of beam lengths, from the shortest.

Tol: The tolerances structure of the truss lattice.

Returns: The resulting truss lattice.

Description: Constructs a trimmed-surface truss lattice from a given set of lattice node ponits. The lattice (assumed to be the result of sphere packing) is marked according to its quality, with shorter beams being better than long beams (this indicates that the lattice is closer to a "perfect" FCC or HCP lattice). The quality information is quantized to finite number of levels, according to a given quantization vector. The resulting object is a list, with the first object containing all the lattice nodes (spheres), and the rest of the objects contain the beams the beams that correspond to each quantization level.
See also: UserTrussTrimmedLatticeWithShell, UserTrussLatticeWithQualityInfo, , UserTrussTrivLattice,

15.2.450 UserTrussMdlPostProcess (truss_mdl.c:825)

truss lattice

model

3D printing

```
TrimSrfStruct *UserTrussMdlPostProcess(TrimSrfStruct *TSrfs,
                                       const UserTrussTolerancesStruct *Tol)
```

TSrfs: The list of trimmed surfaces that form the node and half-beams.

Tol: The tolerances structure of the truss lattice.

Returns: The node and half-beams, as matching trimmed surfaces.

Description: Processes a truss lattice node and half-beams system into a geometry that can be converted into a model. Assuming the original trimmed surfaces were constructed using UserTrussPrepBeamICrvsMatched, the resulting trimmed surfaces are guaranteed to have matching trimming curve segments at all the contacts between the sphere and the beams, and between the beams that intersect with each other.
See also: UserTrussPrepBeamICrvsMatched,

15.2.451 UserTrussNodeDefCleanup (truss_base.c:3864)

```
void UserTrussNodeDefCleanup(UserTrussNodeDefStruct *NodeDef)
```

NodeDef: The lattice node definition struct.

Returns: void

Description: Frees the internal structures of a UserTrussNodeDefStruct. Does not free the UserTrussNodeDefStruct itself, as it is meant to be used in an array.

15.2.452 UserTrussNodeSrfs (truss_base.c:627)

```
void UserTrussNodeSrfs(CagdRType NodeRadius,
                       const CagdRType *BeamRadii,
                       const CagdRType *BeamFilletRadii,
                       const CagdRType *BeamFilletHeights,
                       const CagdVType *BeamDirections,
                       const CagdRType *BeamHeights,
                       int NumBeams,
                       CagdBType MatchTCrvs,
                       CagdBType ForTriv,
                       CagdRType ExtendLength,
                       CagdSrfStruct **Sphere,
                       CagdSrfStruct **Beams,
                       CagdSrfStruct **BeamTopCaps,
                       CagdSrfStruct **BeamBaseCaps,
                       CagdSrfStruct **FilletCylinders,
                       CagdPType *BeamCenterPts,
                       IrtPlnType *AllPlns,
                       CagdBType *BeamsUniform)
```

NodeRadius: The radius of the sphere at the center of the lattice node.

BeamRadii: The radius of each of the half-beams of the node.

BeamFilletRadii: The radius of each of the fillets of the node.

BeamFilletHeights: The height of each of the fillets of the node.

BeamDirections: The directions of the half-beams leaving the node.

BeamHeights: The heights of the half-beams of the node.

NumBeams: The number of beams leaving the node.

MatchTCrvs: A flag indicating whether the trimming curves of the should be matched (used for converting the truss lattice into a model).

ForTriv: A flag indicating whether to construct the auxiliary geometry for trivariate lattice nodes.

ExtendLength: How much to extend the beam.

Sphere: Out parameter: the sphere at the center of the node.

Beams: Out parameter: The half-beams of the lattice node.

BeamTopCaps: Out parameter: the top caps of the beams (trivariate only).

BeamBaseCaps: Out parameter: the base caps of the beams (trivariate only).

FilletCylinders: Out parameter: cylinders that intersect the sphere at its contact curves with the fillets (only if MatchTCrvs is TRUE).

BeamCenterPts: Out parameter: kernel points at the centers of the beams (trivariate only).

AllPlns: Out parameter: the planes containing the bases of fillets. Used for trimming the beams and the sphere.

BeamsUniform: Out parameter: assigned TRUE if all the beams are of uniform shape (same radius, fillet radius and fillet height).

Returns: void

Description: Constructs all the surfaces of a truss lattice node and its half-beams, without trimming them. Also constructs the auxiliary geometry needed for trivariate lattice nodes, if required.

See also: UserTrussTrimmedNode, LatticeConstructNodeTriv,

15.2.453 UserTrussPrepBeamICrvsFast (truss_base.c:2268)

```
void UserTrussPrepBeamICrvsFast(UserTrussClipAgainstBeamsInfoStruct *ClipInfo,
                                CagdBType DoComputeSideSrfs,
                                CagdBType DoAssignIds,
                                const UserTrussTolerancesStruct *Tol)
```

ClipInfo: The clipping information structure to be filled.

DoComputeSideSrfs: A flag indicating whether to compute the side surfaces for trivariate truss construction.

DoAssignIds: A flag indicating whether to attach attributrs to intersection loops, indicating the index of the beam which caused the intersection.

Tol: The tolerances structure of the truss lattice.

Returns: void

Description: Prepare the clipping information for clipping the beams of the truss lattice. This function assumes that all the beams have a uniform shape (radius, filletting height and filleting radius), and computes the intersections as surface-plane intersections, which is faster than SSI.
See also: UserTrussPrepBeamICrvs, UserTrussPrepBeamICrvsMatched, , UserTrussCleanBeamICrvs,

15.2.454 UserTrussPrepBeamICrvsMatched (truss_mdl.c:82)

```
void UserTrussPrepBeamICrvsMatched(
                           UserTrussClipAgainstBeamsInfoStruct *SphereClipInfo,
                           UserTrussClipAgainstBeamsInfoStruct *BeamsClipInfo,
                           CagdSrfStruct *SphereSrf,
                           const IrtPlnType *SphereClipPlns,
                           const CagdSrfStruct **FilletCylinders,
                           CagdCrvStruct **BeamTopLevelCrvs,
                           const UserTrussTolerancesStruct *Tol)
```

SphereClipInfo: The clipping information structure of the sphere, to be filled.

BeamsClipInfo: The clipping information structure of the beams, to be filled.

SphereSrf: The sphere at the center of the lattice node.

SphereClipPlns: The planes that contain the contact curves of the sphere with the beams. These are used for closing the intersection contours into loops.

FilletCylinders: For each beam, a cylinder with the same orientation as the beam, and the radius of the fillet. These are used for computing the intersections with the sphere.

BeamTopLevelCrvs: Out parameter: the top-level trimming loops of the beams.

Tol: The tolerances structure of the truss lattice.

Returns: void

Description: Prepare the clipping information for clipping the beams of the truss lattice. This function computes the intersections using the SSI algorithm, both for sphere-beam and beam-beam intersections, and guarantees that the parametrizations of the resulting intersection curves match each other. This is needed for converting the truss lattice into models (e.g. for 3D printing).
See also: UserTrussPrepBeamICrvs, UserTrussPrepBeamICrvsFast, , UserTrussCleanBeamICrvs,

15.2.455 UserTrussPrepareBeamInfo (truss_base.c:298)

```
UserTrussNodeDefStruct *UserTrussPrepareBeamInfo(
                                   const CagdPType *Pts,
                                   int NumPts,
                                   CagdRType DistToConnect,
                                   UserTrussNodeDefCallbacksStruct *Callbacks,
                                   CagdBType Normalize)
```

Pts: The center points of the lattice nodes.

NumPts: The number of lattice nodes.

DistToConnect: The maximum distance at which points will be connected.

Callbacks: The struct that contains the callbacks for setting the properties of the lattice beams.

Normalize: If TRUE, the beam direction vectors will be normalized.

Returns: The definitions of the nodes to create (array of length NumPts).

Description: Fills in the information on all half-beams leaving each node in the truss lattice. This includes the beam direction vectors, beam radii, beam half-heights, and filletting information. Also, sets the adjacency matrix for the lattice nodes. The beams are constructed between each two nodes with a distance of at most DistToConnect from each other, and guaranteed to be non-intersecting. The parameters of the beams (radius, filletting radius, filletting height) are set by a callback function, which needs to be provided.
See also: UserTrussPrepareBeamInfoWithShell,

15.2.456 UserTrussPrepareBeamInfoWithShell (truss_base.c:3015)

```
UserTrussNodeDefStruct *UserTrussPrepareBeamInfoWithShell(
                                    const CagdPType *Pts,
                                    int NumPts,
                                    CagdRType DistToConnect,
                                    UserTrussNodeDefCallbacksStruct *Callbacks,
                                    CagdBType Normalize,
                                    const IPObjectStruct *ShellObj,
                                    const UserTrussTolerancesStruct *Tol)
```

Pts: The center points of the lattice nodes.

NumPts: The number of lattice nodes.

DistToConnect: The maximum distance at which points will be connected.

Callbacks: The struct that contains the callbacks for setting the properties of the lattice beams.

Normalize: If TRUE, the beam direction vectors will be normalized.

ShellObj: The shell of the model. Beams that intersect with the shell will be deleted. Can be either a surface a polygonal model.

Tol: The tolerances structure of the truss lattice.

Returns: The definitions of the nodes to create (array of length NumPts).

Description: Fills in the information on all half-beams leaving each node in the truss lattice. This includes the beam direction vectors, beam radii, beam half-heights, and filletting information. Also, sets the adjacency matrix for the lattice nodes. The beams are constructed between each two nodes with a distance of at most DistToConnect from each other, and guaranteed not to intersect with each other, or with the shell of the model (if provided). The parameters of the beams (radius, filletting radius, filletting height) are set by a callback function, which needs to be provided.
See also: UserTrussPrepareBeamInfo,

15.2.457 UserTrussSetConstBeamCallbacks (truss_base.c:4205)

```
void UserTrussSetConstBeamCallbacks(
                                UserTrussConstBeamWidthStruct *BeamData,
                                UserTrussNodeDefCallbacksStruct *Callbacks)
```

BeamData: The beam parameters struct (containing the beam radius, filletting radius and filletting height, etc.).

Callbacks: The callbacks struct, where the functions will be assigned.

Returns: void

Description: Sets the callbacks functions for constructing a truss lattice with constant beam width.
See also: UserTrussSetGradedBeamCallbacks,

15.2.458 UserTrussSetCustomBeamCallbacks (truss_base.c:4286)

```
void UserTrussSetCustomBeamCallbacks(
                                UserTrussBeamInfoPrepFuncType PrepCallback,
                                UserTrussBeamInfoFuncType SetDataCallback,
                                UserTrussBeamInfoCleanFuncType CleanupCallback,
                                void *ExtraData,
                                UserTrussNodeDefCallbacksStruct *Callbacks)
```

PrepCallback: The callback function which will be called before the shape data of the lattice nodes is set.

SetDataCallback: The callback function which will be called for setting the shape data of the sphere and each of the beams.

CleanupCallback: The callback function which will be called after the shape data of the lattice nodes is set.

ExtraData: The extra data that is passed to PrepCallback.

Callbacks: The callbacks struct, where the functions will be assigned.

Returns: void

Description: Sets the callbacks for constructing a truss lattice with custom functions for the sphere radius, beam radius, and filleting radius and height.
See also: UserTrussSetConstBeamCallbacks, UserTrussSetGradedBeamCallbacks,

15.2.459 UserTrussSetGradedBeamCallbacks (truss_base.c:4242)

```
void UserTrussSetGradedBeamCallbacks(
                                UserTrussGradedBeamWidthStruct *GradedBeamData,
                                UserTrussNodeDefCallbacksStruct *Callbacks)
```

GradedBeamData: The beam parameters struct (containing the vector along which the beams are to be graded, as well as the start and values for the beam radius, filletting radius, and filletting height).

Callbacks: The callbacks struct, where the functions will be assigned.

Returns: void

Description: Sets the callbacks functions for constructing a truss lattice with graded beam width.
See also: UserTrussSetConstBeamCallbacks,

15.2.460 UserTrussSplitCrvTo2CoordsCrvs (truss_base.c:2616)

```
CagdCrvStruct *UserTrussSplitCrvTo2CoordsCrvs(const CagdCrvStruct *Crv,
                                              int Crv1Coords,
                                              CagdBType NewIsRational)
```

Crv: The curve to split.

Crv1Coords: The number of coordinates to copy to the first curve.

NewIsRational: If TRUE, the newly curves will be created as rational, the weight of the original curve (if exist), or uniform (if not).

Returns: The two split curves.

Description: Splits the coordinates of a curve into two curves, the first containing the first Crv1Coords coordinates, and the second contains the rest of the coordinates.

15.2.461 UserTrussTrimmedLattice (truss_gen.c:90)

truss lattice

sphere packing

```
IPObjectStruct *UserTrussTrimmedLattice(
                                    const CagdPType *Pts,
                                    int NumPts,
                                    CagdRType DistToConnect,
                                    UserTrussNodeDefCallbacksStruct *Callbacks,
                                    CagdBType PrepForMdl,
                                    CagdBType MultiObj,
                                    const UserTrussTolerancesStruct *Tol)
```

Pts: The center points of the lattice nodes.

NumPts: The number of lattice nodes.

DistToConnect: The maximum distance at which points will be connected.

Callbacks: The struct that contains the callbacks for setting the properties of the lattice beams.

PrepForMdl: If TRUE, the truss lattice will be constructed with matching trimming loops, to allow conversion into a closed model.

MultiObj: If TRUE, each node and half-beams system will be contained in a separate object.

Tol: The tolerances structure of the truss lattice.

Returns: The resulting truss lattice.

Description: Constructs a trimmed-surface truss lattice from a given set of lattice node ponits. The lattice can be returned as a single object conatining all the trimmed surfaces in a single object, or as a list object, with each node and half-beams system in a separate object.

See also: UserTrussTrimmedLatticeWithShell, UserTrussLatticeWithQualityInfo, , UserTrussTrivLattice,

15.2.462 UserTrussTrimmedLatticeWithShell (truss_gen.c:202)

truss lattice

sphere packing

```
IPObjectStruct *UserTrussTrimmedLatticeWithShell(
                                    const CagdPType *Pts,
                                    const CagdPType *ShellPts,
                                    CagdRType *ShellDists,
                                    int NumPts,
                                    CagdRType DistToConnect,
                                    UserTrussNodeDefCallbacksStruct *Callbacks,
                                    CagdBType PruneOnly,
                                    const IPObjectStruct *ShellObj,
                                    CagdBType PrepForMdl,
                                    const UserTrussTolerancesStruct *Tol)
```

Pts: The center points of the lattice nodes.

ShellPts: The closest point on the shell surface for each lattice node.

ShellDists: The minimum distance of each lattice node to the

NumPts: The number of lattice nodes.

DistToConnect: The maximum distance at which points will be connected.

Callbacks: The struct that contains the callbacks for setting the properties of the lattice beams.

PruneOnly: If TRUE, then the shell will only be used for removing beams that go outside the shell. If FALSE, and the shell is a B-spline surface, then shell will be connected to the truss lattice.

ShellObj: The shell object. Can be either a B-spline or polygonal model.

PrepForMdl: If TRUE, the truss lattice will be constructed with matching trimming loops, to allow conversion into a closed model.

Tol: The tolerances structure of the truss lattice.

Returns: The resulting truss lattice.

Description: Constructs a trimmed-surface truss lattice from a given set of lattice node ponits, and an outer shell. The lattice is constructed such as all the beams are inside the shell. Also, if the shell is a B-spline surface, it can be connected to the lattice by beams. The resulting lattice can be returned as a list object with the shell and its connections, and each lattice node and half-beams system in a a separate object.

See also: UserTrussTrimmedLattice, UserTrussLatticeWithQualityInfo, , UserTrussTrivLattice,

15.2.463 UserTrussTrimmedNode (truss_base.c:360)

```
TrimSrfStruct *UserTrussTrimmedNode(CagdRType NodeRadius,
                                    const CagdRType *BeamRadii,
                                    const CagdRType *BeamFilletRadii,
                                    const CagdRType *BeamFilletHeights,
                                    const CagdVType *BeamDirections,
                                    const CagdRType *BeamHeights,
                                    int NumBeams,
                                    CagdBType MatchTCrvs,
                                    CagdRType ExtendLength,
                                    const UserTrussTolerancesStruct *Tol)
```

NodeRadius: The radius of the sphere at the center of the lattice node.

BeamRadii: The radius of each of the half-beams of the node.

BeamFilletRadii: The radius of each of the fillets of the node.

BeamFilletHeights: The height of each of the fillets of the node.

BeamDirections: The directions of the half-beams leaving the node.

BeamHeights: The heights of the half-beams of the node.

NumBeams: The number of beams leaving the node.

MatchTCrvs: A flag indicating whether the trimming curves of the should be matched (used for converting the truss lattice into a model).

ExtendLength: How much to extend the beam.

Tol: The tolerances structure of the truss lattice.

Returns: A node and half-beams system, represented as a list of trimmed surfaces.

Description: Constructs a single truss lattice node with its half-beams.
See also: LatticeConstructNodeTriv,

15.2.464 UserTrussTrivLattice (truss_triv.c:852)

truss lattice

sphere packing

```
IPObjectStruct *UserTrussTrivLattice(
                                    const CagdPType *Pts,
                                    int NumPts,
                                    CagdRType DistToConnect,
                                    UserTrussNodeDefCallbacksStruct *Callbacks,
                                    const UserTrussTolerancesStruct *Tol)
```

Pts: The center points of the lattice nodes.

NumPts: The number of lattice nodes.

DistToConnect: The maximum distance at which points will be connected.

Callbacks: The struct that contains the callbacks for setting the properties of the lattice beams.

Tol: The tolerances structure of the truss lattice.

Returns: The resulting truss lattice.

Description: Constructs a trivariate truss lattice from a given set of lattice node ponits. The lattice is returned as a list object, with each node and half-beams system in a separate object.
See also: UserTrussTrimmedLattice, UserTrussTrimmedLatticeWithShell, , UserTrussLatticeWithQualityInfo,

15.2.465 UserTrussTrivLatticeWithShell (truss_triv.c:928)

truss lattice

sphere packing

```
IPObjectStruct *UserTrussTrivLatticeWithShell(
                                    const CagdPType *Pts,
                                    const CagdPType *ShellPts,
                                    CagdRType *ShellDists,
                                    int NumPts,
                                    CagdRType DistToConnect,
                                    UserTrussNodeDefCallbacksStruct *Callbacks,
                                    const IPObjectStruct *ShellObj,
                                    const UserTrussTolerancesStruct *Tol)
```

Pts: The center points of the lattice nodes.

ShellPts: The closest point on the shell surface for each lattice node.

ShellDists: The minimum distance of each lattice node to the

NumPts: The number of lattice nodes.

DistToConnect: The maximum distance at which points will be connected.

Callbacks: The struct that contains the callbacks for setting the properties of the lattice beams.

ShellObj: The shell object. Can be either a B-spline or polygonal model.

Tol: The tolerances structure of the truss lattice.

Returns: The resulting truss lattice.

Description: Constructs a trivariate truss lattice from a given set of lattice node ponits, and an outer shell. The lattice is constructed such as all the beams are inside the shell. The lattice is returned as a list object, with each node and half-beams system in a separate object.
See also: UserTrussTrimmedLatticeWithShell, UserTrussTrivLattice,

15.2.466 UserTwoObjMaxZRelMotion (zcollide.c:44)

```
IrtRType UserTwoObjMaxZRelMotion(IPObjectStruct *PObj1,
                                 IPObjectStruct *PObj2,
                                 IrtRType FineNess,
                                 int NumIters)
```

PObj1: First static object to place PObj2 on.

PObj2: Second dynamic object that is to be moved down (-Z direction) until it is tangent to PObj1.

FineNess: Of polygonal approximation of PObj1.

NumIters: In the bisectioning of collision's test. 10 is a good start.

Returns: Maximal Z motion possible, or IRIT_INFNTY if no collision or error.

Description: Computes the maximal Z motion to move PObj2 down (-Z direction) so that it does not intersect PObj1. Second object is converted to its Bbox.

15.2.467 UserViewingConeSrfDomains (visible.c:197)

```
IPObjectStruct *UserViewingConeSrfDomains(const CagdSrfStruct *Srf,
                                          const CagdSrfStruct *NSrf,
                                          const IPPolygonStruct *ConeDirs,
                                          CagdRType SubdivTol,
                                          CagdRType ConeAngle,
                                          CagdRType Euclidean)
```

Srf: To compute its visibility.

NSrf: The normal surface computed symbollically for Srf.

ConeDirs: Direction of cone's axes.

SubdivTol: Of Cone - normal surface intersection, for subdivision approximation accuracy.

ConeAngle: The opening angle of the cone, in Radians.

Euclidean: Contours are in Euclidean (TRUE) or parametric (FALSE) space of Srf.

Returns: Set of piecewise linear contours, approximating the intersections of the normal viewing cones and the original surface.

Description: Computes the domain in given surface Srf that is inside the given cones of directions ConeDirs and opening angle ConeAngle.
See also: UserVisibilityClassify, UserSrfVisibConeDecomp,

15.2.468 UserVisibilityClassify (visible.c:127)

```
TrimSrfStruct *UserVisibilityClassify(const IPObjectStruct *SclrSrf,
                                      TrimSrfStruct *TrimmedSrfs)
```

SclrSrf: The scalar surface computed symbollically in UserViewingConeSrfDomains, forming this decomosition.

TrimmedSrfs: To verify they are within the viewing code of ViewingDir. Trimmed surfaces inside this list that are outside the viewing cone are eliminated, in place.

Returns: The trimmed regions inside the viewing cone, as a subset of TrimmedSrfs.

Description: Given a decomposition of surface Srf into a set of TrimmedSrfs into regions that are inside the normal viewing cone as prescribed by SclrSrf, eliminate all regions (trimmed surfaces) that are outside the viewing cone in the given list, TrimmedSrfs.
See also: UserViewingConeSrfDomains, UserSrfVisibConeDecomp,

15.2.469 UserVolVisDisplaceGeometry (vol_vis.c:1229)

```
IPObjectStruct *UserVolVisDisplaceGeometry(const IPObjectStruct *GeomObj,
                                           const IPObjectStruct *PropObj,
                                           IrtRType PropScale,
                                           char *Error)
```

GeomObj: Geometry to use.

PropObj: Property function to use.

PropScale: Property scale to use.

Error: Will be updated with the string error if this function failed. Must be a string of 256 bytes or more.

Returns: Displaced geometry.

Description: Computes a displaced geometry based on a property that is assumed to be E3.
See also:

15.2.470 UserVolVisMapPropertyOnGeometry (vol_vis.c:241)

```
IPObjectStruct *UserVolVisMapPropertyOnGeometry(
                                const IPObjectStruct *GeomObj,
                                const IPObjectStruct *PropObj,
                                UserVolVisGenPolygonRepresentationFuncType
                                                      GenPolygonRepresentFunc,
                                void *GenPolygonRepresentData,
                                UserVolVisGetRGBFromRealFuncType
                                                           GetRGBFromRealFunc,
                                void *GetRGBFromRealData,
                                int PropCoord,
                                IrtRType MaxEdgeLen,
                                IrtRType MinProp,
                                IrtRType MaxProp,
                                char *Error)
```

GeomObj: Geometry to use.

PropObj: Property function to use.

GenPolygonRepresentFunc: A call back function to tesselate the geometry, or NULL to use a default tesselation code.

GenPolygonRepresentData: Optional reference to data to transfer to the call back function GenPolygonRepresentFunc. NULL, to ignore.

GetRGBFromRealFunc: Call back function to convert a scalar value in [0, 1] to RGB in [0, 255]^3.

GetRGBFromRealData: Optional reference to data to transfer to the call back function GetRGBFromRealFunc. NULL, to ignore.

PropCoord: Axis index of property, 1 for X, 2 for y, etc.

MaxEdgeLen: To control maximal sizes of polygonal approximation.

MinProp, MaxProp: Minimal and maximal values the property can assume.

Error: Will be updated with the string error if this function failed. Must be a string of 256 bytes or more.

Returns: Polygonal geometry with RGB attribute reflectin the property values.

Description: Computes a polygonal approximation of the geometry and map the property values as RGB on the (polygonal) geometry.
See also:

15.2.471 UserVolVisMapPropertyOnIsoSrfGeometry (vol_vis.c:473)

```
IPObjectStruct *UserVolVisMapPropertyOnIsoSrfGeometry(
                                const IPObjectStruct *GeomObj,
                                const IPObjectStruct *PropObj,
                                UserVolVisGenPolygonRepresentationFuncType
                                                      GenPolygonRepresentFunc,
                                void *GenPolygonRepresentData,
                                UserVolVisGetRGBFromRealFuncType
                                                          GetRGBFromRealFunc,
                                void *GetRGBFromRealData,
                                TrivTVDirType VisDir,
                                IrtRType ParamVal,
                                int PropCoord,
                                IrtRType MaxEdgeLen,
                                IrtRType MinProp,
                                IrtRType MaxProp,
                                char *Error)
```

GeomObj: Geometry to use.

PropObj: Property function to use.

GenPolygonRepresentFunc: A function to tesselate the geometry, or NULL to use a default tesselation code.

GenPolygonRepresentData: Optional reference to data to transfer to the call back function GenPolygonRepresentFunc. NULL, to ignore.

GetRGBFromRealFunc: Function to convert a scalar value in [0, 1] to RGB in [0, 255]^3.

GetRGBFromRealData: Optional reference to data to transfer to the call back function GetRGBFromRealFunc. NULL, to ignore.

VisDir: Direction to extract the UVW isosurface from: 0, 1, or 2.

ParamVal: Parameter value from which to extract the isosurface.

PropCoord: Axis index of property, 1 for X, 2 for y, etc.

MaxEdgeLen: To control maximal sizes of polygonal approximation.

MinProp, MaxProp: Minimal and maximal values the property can assume. m

Error: Will be updated with the string error if this function failed. Must be a string of 256 bytes or more.

Returns: Polygonal geometry with RGB attribute reflecting the property values.

Description: Computes a polygonal approximation of an isosurface of the geometry and map the property values as RGB on the (polygonal) geometry.
See also:

15.2.472 UserVolVisMapPropertyOnMarchingCubeGeometry (vol_vis.c:1061)

```
IPObjectStruct *UserVolVisMapPropertyOnMarchingCubeGeometry(
                                      const IPObjectStruct *GeomObj,
                                      const IPObjectStruct *PropObj,
                                      UserVolVisGetRGBFromRealFuncType
                                                          GetRGBFromRealFunc,
                                      void *GetRGBFromRealData,
                                      IrtRType IsoVal,
                                      int PropCoord,
```

```
                                        IrtRType MaxEdgeLen,
                                        IrtRType MinProp,
                                        IrtRType MaxProp,
                                        char *Error)
```

GeomObj: Geometry to use.

PropObj: Property function to use.

GetRGBFromRealFunc: Function to convert a scalar value in [0, 1] to RGB in [0, 255]^3.

GetRGBFromRealData: Optional reference to data to transfer to the call back function GetRGBFromRealFunc. NULL, to ignore.

IsoVal: Parameter value from which to extract the isosurface.

PropCoord: Axis index of property, 1 for X, 2 for y, etc.

MaxEdgeLen: To control maximal sizes of polygonal approximation.

MinProp, MaxProp: Minimal and maximal values the property can assume.

Error: Will be updated with the string error if this function failed. Must be a string of 256 bytes or more.

Returns: Polygonal geometry with RGB attribute reflecting the property values.

Description: Computes a polygonal approximation of an (marching cubed) iso-surface of the property and map to the geometry with proper uniform RGB on the (polygonal) geometry.
See also:

15.2.473 UserVolVisMapPropertyOnPlanarSliceGeometry (vol_vis.c:907)

```
IPObjectStruct *UserVolVisMapPropertyOnPlanarSliceGeometry(
                                        const IPObjectStruct *GeomObj,
                                        const IPObjectStruct *PropObj,
                                        const IrtPlnType PlaneEqn,
                                        UserVolVisGetRGBFromRealFuncType
                                                        GetRGBFromRealFunc,
                                        void *GetRGBFromRealData,
                                        int PropCoord,
                                        IrtRType MaxEdgeLen,
                                        IrtRType MinProp,
                                        IrtRType MaxProp,
                                        char *Error)
```

GeomObj: Geometry to use.

PropObj: Property function to use.

PlaneEqn: Plane equation to slice with - ABCD in Ax + By + Cz + D = 0.

GetRGBFromRealFunc: Function to convert a scalar value in [0, 1] to RGB in [0, 255]^3.

GetRGBFromRealData: Optional reference to data to transfer to the call back function GetRGBFromRealFunc. NULL, to ignore.

PropCoord: Axis index of property, 1 for X, 2 for y, etc.

MaxEdgeLen: To control maximal sizes of polygonal approximation.

MinProp, MaxProp: Minimal and maximal values the property can assume.

Error: Will be updated with the string error if this function failed. Must be a string of 256 bytes or more.

Returns: Polygonal geometry with RGB attribute reflecting the property values.

Description: Computes a polygonal approximation of a planar slice of the geometry and map the property values as RGB on the (polygonal) geometry.
See also:

15.2.474 UserVolVisMapPropertyOnPolygons (vol_vis.c:636)

```
void UserVolVisMapPropertyOnPolygons(IPObjectStruct *PlObj,
                                     int UVReversed,
                                     const TrivTVStruct *PropTV,
                                     UserVolVisGetRGBFromRealFuncType
                                                          GetRGBFromRealFunc,
                                     void *GetRGBFromRealData,
                                     int PropCoord,
                                     IrtRType MinProp,
                                     IrtRType MaxProp,
                                     char *Error)
```

PlObj: Polygons to update their vertices with RGB values.

UVReversed: State how the UV parameters of original srf, Pls are from, were flipped/reversed in the match as the following bits, 1 - U was reversed, 2 - V was reversed, 4 - if U and V were flipped. Finally, if Reversed = -1, no UVW values are know and they will be inverted directly (Through propTV) from the E3 Pos.

PropTV: Property Userariates.

GetRGBFromRealFunc: Function to convert a scalar value in [0, 1] to RGB in [0, 255]^3.

GetRGBFromRealData: Optional reference to data to transfer to the call back function GetRGBFromRealFunc. NULL, to ignore.

PropCoord: Axis index in PropTV of property to evaluate, 1 for X, 2 for y, etc.

MinProp, MaxProp: Minimal and maximal values the property can assume.

Error: Will be updated with the string error if this function failed. Must be a string of 256 bytes or more.

Returns: void

Description: Evaluate the RGB of every vertex based on the property trivariate.
See also:

15.2.475 UserVolVisVerifyCorrespondence (vol_vis.c:83)

```
int UserVolVisVerifyCorrespondence(const IPObjectStruct *GeomObj,
                                   const IPObjectStruct *PropObj,
                                   char *Error)
```

GeomObj: The geometry.

PropObj: The property.

Error: Will be updated with the string error if this function failed. Must be a string of 256 bytes or more.

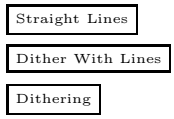
Returns: TRUE if valid, FALSE otherwise.

Description: Verify the validity of the object geometry/property pairs. Can be:

1. a parallel hierarchy of trivariates/vmodels, sharing domains.
2. a model object embedded in a trivariate property.

See also:

15.2.476 UserWDDitherCombiBW (wire_dither.c:534)




```
IPObjectStruct *UserWDDitherCombiBW(IritImgPrcssImgStruct *Image0,
                                    IritImgPrcssImgStruct *Image1,
                                    int NumberOfLines,
                                    int NumberOfPins,
                                    float LineIntensity,
                                    IritImgPrcssImgStruct **Projection0,
                                    IritImgPrcssImgStruct **Projection1,
                                    float Fairness0,
                                    float Fairness1,
                                    float FeatureImportance)
```

Image0: First image to dither.

Image1: Second image to dither.

NumberOfLines: Number of lines to dither with.

NumberOfPins: Number of pins on each edge of the cube.

LineIntensity: The intensity of the drawn lines (relevant in projections).

Projection0: Path for saving projection of the wires on the XY plane.

Projection1: Path for saving projection of the wires on the ZY plane.

Fairness0: Fairness score multiplier for image0.

Fairness1: Fairness score multiplier for image1.

FeatureImportance: Score multiplier for pixels on the edges.

Returns: Wires object.

Description: Wrapper to the dithering with lines function that draws the images in black lines only, Same as calling the original function with (0, 0, 0, 1) as the only available color.

15.2.477 UserWDDitherCombiRGB (wire_dither.c:587)

Straight Lines

Dither With Lines

Dithering

```
IPObjectStruct *UserWDDitherCombiRGB(IritImgPrcssImgStruct *Image0,
                                     IritImgPrcssImgStruct *Image1,
                                     int NumberOfLines,
                                     int NumberOfPins,
                                     float LineIntensity,
                                     IritImgPrcssImgStruct **Projection0,
                                     IritImgPrcssImgStruct **Projection1,
                                     float Fairness0,
                                     float Fairness1,
                                     float FeatureImportance)
```

Image0: First image to dither.

Image1: Second image to dither.

NumberOfLines: Number of lines to dither with.

NumberOfPins: Number of pins on each edge of the cube.

LineIntensity: The intensity of the drawn lines (relevant in projections).

Projection0: Variable to hold the projected image on the XY plane.

Projection1: Variable to hold the projected image on the YZ plane.

Fairness0: Fairness score multiplier for image0.

Fairness1: Fairness score multiplier for image1.

FeatureImportance: Score multiplier for the automatic edge detection

Returns: Wires object.

Description: Wrapper to the dithering with lines function that draws the images in Red, Green and Blue lines only, Same as calling the original function, with (1, 0, 0, 1), (0, 1, 0, 1), (0, 0, 1, 1) Colors only.

15.2.478 UserWDDitherStochasticBW (wire_dither.c:647)

Straight Lines

Dither With Lines

Dithering

```
IPObjectStruct *UserWDDitherStochasticBW(IritImgPrcssImgStruct *Image0,
                                         IritImgPrcssImgStruct *Image1,
                                         int NumberOfLines,
                                         int NumberOfPins,
                                         int NumberOfRandomTrials,
                                         float LineIntensity,
                                         IritImgPrcssImgStruct **Projection0,
                                         IritImgPrcssImgStruct **Projection1,
                                         float Fairness0,
                                         float Fairness1,
                                         int BoundWireLength,
                                         float FeatureImportance)
```

Image0: First image to dither.

Image1: Second image to dither.

NumberOfLines: Number of lines to dither with.

NumberOfPins: Number of pins on each edge of the cube.

NumberOfRandomTrials: Number of random trials to perform, per wire.

LineIntensity: The intensity of the drawn lines (relevant in projections).

Projection0: Path for saving projection of the wires on the XY plane.

Projection1: Path for saving projection of the wires on the ZY plane.

Fairness0: Fairness score multiplier for image0.

Fairness1: Fairness score multiplier for image1.

BoundWireLength: In the stochastic mode whether to use lines that do'nt necessarily lay on the cube surface.

FeatureImportance: Score multiplier for the automatic edge detection.

Returns: Wires object.

Description: Wrapper to the dithering with lines function of one/two given images.

15.2.479 UserWDDitherStochasticRGB (wire_dither.c:704)

Straight Lines

Dither With Lines

Dithering

```
IPObjectStruct *UserWDDitherStochasticRGB(IritImgPrcssImgStruct *Image0,
                                          IritImgPrcssImgStruct *Image1,
                                          int NumberOfLines,
                                          int NumberOfPins,
                                          int NumberOfRandomTrials,
                                          float LineIntensity,
                                          IritImgPrcssImgStruct **Projection0,
                                          IritImgPrcssImgStruct **Projection1,
                                          float Fairness0,
                                          float Fairness1,
                                          int BoundWireLength,
                                          float FeatureImportance)
```

Image0: First image to dither.

Image1: Second image to dither.

NumberOfLines: Number of lines to dither with.

NumberOfPins: Number of pins on each edge of the cube.

NumberOfRandomTrials: Number of random trials to perform, per wire.

LineIntensity: The intensity of the drawn lines (relevant in projections).

Projection0: Path for saving projection of the wires on the XY plane.

Projection1: Path for saving projection of the wires on the ZY plane.

Fairness0: Fairness score multiplier for image0.

Fairness1: Fairness score multiplier for image1.

BoundWireLength: In the stochastic mode whether to use lines that do'nt necessarily lay on the cube surface.

FeatureImportance: Score multiplier for the automatic edge detection.

Returns: Wires object.

Description: Wrapper to the dithering with lines function of one/two given images.

15.2.480 UserWarpTextOnSurface (textwarp.c:65)

```
IPObjectStruct *UserWarpTextOnSurface(CagdSrfStruct *Srf,
                                      const char *Txt,
                                      IrtRType HSpace,
                                      IrtRType VBase,
                                      IrtRType VTop,
                                      IrtRType Ligatures)
```

Srf: Surface to warp the text Txt along. The text is laid along the u axis with the v axis being the height.

Txt: Text to warp inside Srf.

HSpace: Horizontal space between characters.

VBase, VTop: Minimal and maximal fraction of height for regular characters. Measured on the letter 'A'.

Ligatures: If non zero, amount of ligatures' contraction between adjacent characters.

Returns: List object of warped text, one object per character, each character is a list of Bezier curves. NULL if error.

Description: Warps the given text, using the current loaded font, in Bezier form in surface Srf using composition. The characters of Txt are laid one after the other until the entire surface is filled horizontally, or characters in Txt are exhausted. The characters are scaled to fit VBase to VTop of the height (v) direction of the parametric domain of Srf, for the letter 'A'.

See also: GMLoadTextFont, GMMakeTextGeometry, SymbComposeSrfCrv,

15.2.481 main (dtr_crvs.c:582)

```
void main(int argc, char **argv)
```

argc, argv: Command line.

Returns: void

Description: Main module of DBC2 - Read command line and do what is needed...

15.2.482 main (fntelem1.c:1434)

```
void main(void)
```

Returns: void

Description: Test routines for the FE code.

Chapter 16

Volumetric Model Library, vmdl_lib

16.1 General Information

This library provides the necessary tools to represent and process volumetric models (VModels). VModels are sets of trimmed trivariates forming a closed 3-manifold shell. VModels are typically the result of Boolean operations over freeform (trimmed) trivariate NURBs geometry but can also be constructed directly or via other schemes.

The interface of the library is defined in *include/vmdl_lib.h*.

16.2 Library Functions

16.2.1 VMdlAddTrimSrfToVMdl (vmdl_bool.c:3562)

```
void VMdlAddTrimSrfToVMdl(VMdlVModelStruct *VMdl,
                          const TrimSrfStruct *TSrf,
                          CagdRType Tol)
```

VMdl: Input V-model to append trim-surface to, in place.

TSrf: Input trim-surface to append. Assumed to also hold Euclidean trimming curves.

Tol: The tolerance for matching points and curves.

Returns: void

Description: Appends the given trim-surface to the given V-model, inplace. The input trim-surface is assumed to be correctly oriented. The Euclidean trim- curves of the input trim-surfaces are assumed to be present. The input V-model is assumed to have a single volume element.

16.2.2 VMdlAllocInterTrimCurveSeg (vmdl_gen.c:126)

```
VMdlInterTrimCurveSegStruct *VMdlAllocInterTrimCurveSeg()
```

Returns: New empty V-curve.

Description: Allocated new V-curve.

16.2.3 VMdlAllocInterTrimCurveSegLoopInSrf (vmdl_gen.c:226)

```
VMdlInterTrimCurveSegLoopInSrfStruct *VMdlAllocInterTrimCurveSegLoopInSrf()
```

Returns: New empty half-V-surface loop trimming curve.

Description: Allocated new trimming curve in a half-V-surface trimming loop.

16.2.4 VMdlAllocInterTrimCurveSegRef (vmdl_gen.c:168)

```
VMdlInterTrimCurveSegRefStruct *VMdlAllocInterTrimCurveSegRef()
```

Returns: New empty V-curve reference.

Description: Allocated new reference to a V-curve.

16.2.5 VMdlAllocInterTrimPointRef (vmdl_gen.c:245)

```
VMdlInterTrimPointRefStruct *VMdlAllocInterTrimPointRef()
```

Returns: New empty V-point reference.

Description: Allocated new reference to a V-point.

16.2.6 VMdlAllocInterTrimSrf (vmdl_gen.c:102)

```
VMdlInterTrimSrfStruct *VMdlAllocInterTrimSrf()
```

Returns: New empty half-V-surface.

Description: Allocated new half-V-surface.

16.2.7 VMdlAllocInterTrimSrfRef (vmdl_gen.c:206)

```
VMdlInterTrimSrfRefStruct *VMdlAllocInterTrimSrfRef()
```

Returns: New empty half-V-surface reference.

Description: Allocated new reference to a half-V-surface.

16.2.8 VMdlAllocInterTrivTVRef (vmdl_gen.c:187)

```
VMdlInterTrivTVRefStruct *VMdlAllocInterTrivTVRef()
```

Returns: New empty trivariate reference.

Description: Allocated new reference to a trivariate.

16.2.9 VMdlAllocTrimInterPoint (vmdl_gen.c:81)

```
VMdlInterTrimPointStruct *VMdlAllocTrimInterPoint()
```

Returns: New empty V-point.

Description: Allocated new V-point.

16.2.10 VMdlAllocTrimVolumeElem (vmdl_gen.c:58)

```
VMdlVolumeElementStruct *VMdlAllocTrimVolumeElem()
```

Returns: New empty V-cell.

Description: Allocated new V-cell.

16.2.11 VMdlAllocTrimVolumeElemRef (vmdl_gen.c:149)

```
VMdlVolumeElementRefStruct *VMdlAllocTrimVolumeElemRef()
```

Returns: New empty V-cell reference.

Description: Allocated new reference to a V-cell.

16.2.12 VMdlAllocVModel (vmdl_gen.c:33)

```
VMdlVModelStruct *VMdlAllocVModel()
```

Returns: New empty V-model.

Description: Allocated new empty V-model.

16.2.13 VMdlApplyVSrfTrimmedLoopsOnSrf (vmdl_aux.c:707)

```
TrimSrfStruct *VMdlApplyVSrfTrimmedLoopsOnSrf(
                                        const VMdlInterTrimSrfStruct *VSrf,
                                        const CagdSrfStruct *Srf)
```

VSrf: The V-surface to convert.

Srf: Surface to apply to th trimming curves of VSrf on.

Returns: Trimmed surface structure converted from VSrf.

Description: Creates trimmed surface from a given surface (Srf) using the trimming curves of a given V-surface (VSrf). If Srf is the surface of VSrf then this function will convert from a V-surface to a trimmed surface.

See also: VMdlCnvrtVSrf2TrimmedSrf,

16.2.14 VMdlApplyVSrfTrimmedLoopsOnSrfEps (vmdl_aux.c:671)

```
TrimSrfStruct *VMdlApplyVSrfTrimmedLoopsOnSrfEps(
                                        const VMdlInterTrimSrfStruct *VSrf,
                                        const CagdSrfStruct *Srf,
                                        CagdRType Tol)
```

VSrf: The V-surface to convert.

Srf: Surface to apply to the trimming curves of VSrf on.

Tol: Tolerance for the conversion.

Returns: Trimmed surface structure converted from VSrf.

Description: Creates trimmed surface from a given surface (Srf) using the trimming curves of a given V-surface (VSrf). If Srf is the surface of VSrf then this function will convert from a V-surface to a trimmed surface.

See also: VMdlCnvrtVSrf2TrimmedSrf,

16.2.15 VMdlBlendBoolTrivariates (vmdl_blnd_field.c:452)

```
void VMdlBlendBoolTrivariates(VMdlBlendPrepStruct *BlendCoordPrep,
                              const IrtRType Pt[3],
                              CagdBType UseRayCasting)
```

BlendCoordPrep: The VMdlBlendPrepStruct containing the topology of the V-model.

Pt: Point of which we need to compute the barycentric coordinates.

UseRayCasting: TRUE to use ray casting in the inclusion test.

Returns: void

Description: Given the VMdlBlendPrepStruct of a V-model and a point, fills the VMdlBlendPointStruct in the VMdlBlendPrepStruct with the data relative to point Pt. Ray-tracing can be used to check the inclusion of the point in the V-model.

See also: VMdlBlendPrepStruct, VMdlBlendPointStruct, VMdlBlendBoolTrivariatesVElem,

16.2.16 VMdlBlendBoolTrivariatesVElem (vmdl_blnd_field.c:1729)

```
void VMdlBlendBoolTrivariatesVElem(VMdlBlendPrepStruct *BlendCoordPrep,
                                   IrtRType Pt[3],
                                   VMdlBlendVolElemStruct *VElem)
```

BlendCoordPrep: The VMdlBlendPrepStruct containing the topology of the V-model.

Pt: Point of which we need to compute the barycentric coordinates.

VElem: Pointer to the V-cell holding Pt.

Returns: void

Description: Given the VMdlBlendPrepStruct of a V-model, a point, and the V-cell holding the point, fills VMdlBlendPointStruct in the VMdlBlendPrepStruct with the data relative to point Pt. The method assumes the V-cell to be holding the point.
See also: VMdlBlendPrepStruct, VMdlBlendPointStruct, VMdlBlendBoolTrivariates,

16.2.17 VMdlBlendComputeDistCoordinates (vmdl_blnd_field.c:353)

```
void VMdlBlendComputeDistCoordinates(VMdlBlendPrepStruct *BlendCoordPrep,
                                     const CagdRType Pt[3])
```

BlendCoordPrep: VMdlBlendPrepStruct of the V-model.

Pt: Point of which we need to compute the barycentric coordinates.

Returns: void

Description: Given a VMdlBlendPrepStruct and a point, computes its coordinates with respect to the boundary surfaces of the union intersection and stores them in the corresponding structure.
See also: VMdlBlendPrepStruct, ENDWEIGHTS:, distance-based,

16.2.18 VMdlBlendCoordPrep (vmdl_blnd_field.c:139)

```
VMdlBlendPrepStruct *VMdlBlendCoordPrep(const VMdlParamsStruct *Params,
                                        VMdlVModelStruct *VMdl)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VMdl: The V-model for which the topology is needed.

Returns: VMdlBlendPrepStruct for the computation of the blend.

Description: Given a V-model, creates an auxiliary structure for the computation of the blending.
See also: VMdlBlendCoordPrep, VMdlBlendPrepStruct,

16.2.19 VMdlBlendCoordPrepFree (vmdl_blnd_field.c:1207)

```
void VMdlBlendCoordPrepFree(VMdlBlendPrepStruct *Item)
```

Item: pointer to the structure that needs to be freed.

Returns: void

Description: Given a VMdlBlendPrepStruct, it frees its memory.
See also: VMdlBlendPrepStruct,

16.2.20 VMdlBlendDflt1DPropertyFunction (vmdl_blnd_field.c:2825)

barycentric weights
blending
Boolean operations

```
CagdPointType VMdlBlendDflt1DPropertyFunction(VMdlBlendPrepStruct
                                                    *BlendCoordPrep,
                                              const int ObjIndex,
                                              void *Output)
```

BlendCoordPrep: A structure containing useful quantities regarding the V-model.

ObjIndex: Index of the relevant V-primitive.

Output: Pointer to the output.

Returns: Dimension of the property function.

Description: Given a VMdlBlendPrepStruct, an ObjIndex and a pointer to an object returns the scalar value stored in the corresponding trivariate.
See also: distance-based coordinates, attributes, VMdlBlendPrepStruct, , VMdlBlendCoordinatesFuncType,

16.2.21 VMdlBlendDflt3DPropertyFunction (vmdl_blnd_field.c:2693)

barycentric weights
blending
Boolean operations

```
CagdPointType VMdlBlendDflt3DPropertyFunction(VMdlBlendPrepStruct
                                                    *BlendCoordPrep,
                                              const int ObjIndex,
                                              void *Output)
```

BlendCoordPrep: A structure containing useful quantities regarding the V-model.

ObjIndex: Index of the relevant V-primitive.

Output: Pointer to the output.

Returns: Dimension of the property function.

Description: Given a VMdlBlendPrepStruct, an ObjIndex and a pointer to an object returns the (three dimensional) value stored in the corresponding trivariate.
See also: distance-based coordinates, attributes, VMdlBlendPrepStruct, , VMdlBlendCoordinatesFuncType,

16.2.22 VMdlBlendDfltRGBPropertyFunction (vmdl_blnd_field.c:2562)

barycentric weights
blending
Boolean operations

```
CagdPointType VMdlBlendDfltRGBPropertyFunction(VMdlBlendPrepStruct
                                                    *BlendCoordPrep,
                                               const int ObjIndex,
                                               void *Output)
```

BlendCoordPrep: A structure containing useful quantities regarding the V-model.

ObjIndex: Index of the relevant V-primitive.

Output: Pointer to the output.

Returns: Dimension of the property function.

Description: Given a VMdlBlendPrepStruct, an ObjIndex and a pointer to an object returns the RGB of the corresponding V-primitive.
See also: distance-based coordinates, attributes, VMdlBlendPrepStruct, , VMdlBlendCoordinatesFuncType,

16.2.23 VMdlBlendFilletComputeCoordinates (vmdl_blnd_field.c:2281)

```
void VMdlBlendFilletComputeCoordinates(VMdlBlendPrepStruct *BlendCoordPrep,
                                       const CagdRType *Pt)
```

BlendCoordPrep: VMdlBlendPrepStruct of the V-model, contains information of the fillet.

Pt: Point of which we need to compute the barycentric coordinates.

Returns: void

Description: Given a VMdlBlendPrepStruct and a point, computes its Shepard-like coordinates with respect to the boundary surfaces of the fillet and stores them in the corresponding structure.
See also: VMdlBlendPrepStruct, Shepard, ENDWEIGHTS:, Shepard,

16.2.24 VMdlBlendFilletCoordPrep (vmdl_blnd_field.c:2889)

```
VMdlBlendPrepStruct *VMdlBlendFilletCoordPrep(const VMdlParamsStruct *Params,
                                              VMdlVModelStruct *VMdl,
                                              TrivTVStruct *FilletTV,
                                              CagdSrfStruct *BndrSrf1,
                                              CagdSrfStruct *BndrSrf2)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VMdl: The V-model for which the topology is needed.

FilletTV: The fillet trivariate.

BndrSrf1, BndrSrf2: The (internal) boundary surfaces of the fillet.

Returns: VMdlBlendPrepStruct for the computation of the blend.

Description: Given a V-model, a trivariate and two surfaces, it creates a VMdlBlendPrepStruct for the computation of the blending including a fillet.
See also: VMdlBlendCoordPrep, VMdlBlendPrepStruct, VMdlBlendFilletStruct,

16.2.25 VMdlBlendFilletDflt1DPropertyFunction (vmdl_blnd_field.c:2755)

barycentric weights

blending

Boolean operations

```
CagdPointType VMdlBlendFilletDflt1DPropertyFunction(VMdlBlendPrepStruct
                                                    *BlendCoordPrep,
                                                    const int ObjIndex,
                                                    void *Output)
```

BlendCoordPrep: A structure containing useful quantities regarding the V-model.

ObjIndex: Index of the relevant V-primitive.

Output: Pointer to the output.

Returns: Dimension of the property function.

Description: Given a VMdlBlendPrepStruct, an ObjIndex and a pointer to an object returns the scalar value stored in the boundary surfaces of a fillet.
See also: Shepard coordinates, attributes, VMdlBlendPrepStruct, , VMdlBlendCoordinatesFuncType,

16.2.26 VMdlBlendFilletDflt3DPropertyFunction (vmdl_blnd_field.c:2622)

barycentric weights

blending

Boolean operations

```
CagdPointType VMdlBlendFilletDflt3DPropertyFunction(VMdlBlendPrepStruct
                                                    *BlendCoordPrep,
                                                    const int ObjIndex,
                                                    void *Output)
```

BlendCoordPrep: A structure containing useful quantities regarding the V-model.

ObjIndex: Index of the relevant V-primitive.

Output: Pointer to the output.

Returns: Dimension of the property function.

Description: Given a VMdlBlendPrepStruct, an ObjIndex and a pointer to an object returns the (three dimensional) value stored in the boundary surfaces of a fillet.
See also: Shepard coordinates, attributes, VMdlBlendPrepStruct, , VMdlBlendCoordinatesFuncType,

16.2.27 VMdlBlendFilletDfltRGBPropertyFunction (vmdl_blnd_field.c:2497)

barycentric weights
blending
Boolean operations

```
CagdPointType VMdlBlendFilletDfltRGBPropertyFunction(VMdlBlendPrepStruct
                                                     *BlendCoordPrep,
                                                     const int ObjIndex,
                                                     void *Output)
```

BlendCoordPrep: A structure containing useful quantities regarding the V-model.

ObjIndex: Index of the relevant V-primitive.

Output: Pointer to the output.

Returns: Dimension of the property function.

Description: Given a VMdlBlendPrepStruct, an ObjIndex and a pointer to an object returns the RGB of the corresponding V-primitive, taken from the boundary surfaces of a fillet.
See also: Shepard coordinates, attributes, VMdlBlendPrepStruct, , VMdlBlendCoordinatesFuncType,

16.2.28 VMdlBlendFilletFree (vmdl_blnd_field.c:1235)

```
void VMdlBlendFilletFree(VMdlBlendFilletStruct *Item)
```

Item: pointer to the structure that needs to be freed.

Returns: void

Description: Given a VMdlBlendFilletStruct, it frees its memory.
See also: VMdlBlendFilletStruct,

16.2.29 VMdlBlendFilletPropertiesPerPoint (vmdl_blnd_field.c:2377)

barycentric weights
blending
Boolean operations

```
void *VMdlBlendFilletPropertiesPerPoint(
                                    VMdlBlendPrepStruct *BlendCoordPrep,
                                    const IrtRType Pt[3],
                                    void *OutputPtr,
                                    VMdlBlendCoordinatesFuncType Coordinates,
                                    VMdlBlendPropertyFuncType PropertyFunction)
```

BlendCoordPrep: A structure containing useful quantities regarding the V-model.

Pt: Euclidean coordinates of the point for which we need to compute the blend.

OutputPtr: Pointer to the output.

Coordinates: Call-back function for the chosen coordinates.

PropertyFunction: Specified property function for each V-primitive.

Returns: Value of the property in the point.

Description: Given a VMdlBlendPrepStruct, a point and two a call-back functions, it blends a 3 dimensional vector of attributes coordinate-wise. Standard function for fillet blend, if nothing is specified.
See also: Shepard coordinates, attributes, VMdlBlendPrepStruct, , VMdlBlendCoordinatesFuncType,

16.2.30 VMdlBlendFilletPropertiesSlice (vmdl_blnd_field.c:2162)

```
VMdlSlicerOutputImageStruct *VMdlBlendFilletPropertiesSlice(
                                    const VMdlParamsStruct *VMdlParams,
                                    VMdlBlendPrepStruct *BlendCoordPrep,
                                    const double z,
                                    int Resolution[2],
                                    VMdlBlendCoordinatesFuncType Coordinates,
                                    VMdlBlendPropertyFuncType PropertyFunction,
                                    VMdlBlendRuleFuncType BlendingRule,
                                    unsigned int SizePixelProperty)
```

VMdlParams: General params of the VMOdel library.

BlendCoordPrep: A structure containing the blending properties.

z: Z coordinate of the slice of the fillet.

Resolution: Size of the output image.

Coordinates: Call-back function for the chosen coordinates.

PropertyFunction: User specified call-back function to define the property to be blended.

BlendingRule: User specified call-back function to specify how to blend the specified property.

SizePixelProperty: Size of pixel property.

Returns: Output array containing the property values for every point in the fillet.

Description: Given a TrivTVStruct containing trivariates arising from filleting of two surfaces, extrapolates the value of the properties in the filleting area.
See also: VMdlBlendComputeDistCoordinates,

16.2.31 VMdlBlendInIntersection (vmdl_blnd_field.c:100)

```
int VMdlBlendInIntersection(VMdlBlendPrepStruct *BlendCoordPrep,
                            int *VPrimIndex)
```

BlendCoordPrep: Prep struct of the VMdl.

VPrimIndex: If the point belongs to only one VMdl, its index is saved here.

Returns: 1 if the point belongs to an intersection; 0 if it belongs to only one trivariate; -1 if it is outside the V-model.

Description: Given a VMdlBlendPrepStruct and the pointer to an integer, returns an integer that corresponds to how many trivariates the point belongs.
See also: VMdlBlendPrepStruct,

16.2.32 VMdlBlendPointFree (vmdl_blnd_field.c:1271)

```
void VMdlBlendPointFree(VMdlBlendPointStruct *Item)
```

Item: pointer to the structure that needs to be freed.

Returns: void

Description: Given a VMdlBlendPointStruct, it frees its memory.
See also: VMdlBlendPointStruct, ,

16.2.33 VMdlBlendPointListFree (vmdl_blnd_field.c:1301)

```
void VMdlBlendPointListFree(VMdlBlendPointStruct *Item)
```

Item: pointer to the structure list that needs to be freed.

Returns: void

Description: Given a list of VMdlBlendPointStruct, it frees its memory.
See also: VMdlBlendPointStruct, VMdlBlendPointFree,

16.2.34 VMdlBlendPointStructInit (vmdl_blnd_field.c:211)

```
VMdlBlendPointStruct *VMdlBlendPointStructInit(const int SizeTopology,
                                               const int NumTVs)
```

SizeTopology: Size of the topology structure of BlendCoordPrep.

NumTVs: Number of TVs in BlendCoordPrep.

Returns: The point structure.

Description: Given the number of V-primitives in the V-model and the number of trivariates, initializes a point structure related to a BlendCoordPrep.
See also: VMdlBlendCoordPrep, VMdlBlendPrepStruct,

16.2.35 VMdlBlendPropertiesPerPoint (vmdl_blnd_field.c:1401)

barycentric weights

blending

Boolean operations

```
void *VMdlBlendPropertiesPerPoint(VMdlBlendPrepStruct *BlendCoordPrep,
                                  const IrtRType Pt[4],
                                  void *OutputPtr,
                                  VMdlBlendCoordinatesFuncType Coordinates,
                                  VMdlBlendPropertyFuncType PropertyFunction)
```

BlendCoordPrep: A structure containing useful quantities regarding the V-model.

Pt: Euclidean coordinates of the point for which we need to compute the blend.

OutputPtr: Pointer to the output.

Coordinates: Call-back function for the chosen coordinates.

PropertyFunction: Specified property function for each V-primitive.

Returns: Value of the property in the point.

Description: Given a VMdlBlendPrepStruct, a point and two call-back functions, it blends a 3 dimensional vector of attributes coordinate-wise. Standard function, if nothing is specified.
See also: attributes, VMdlBlendPrepStruct, VMdlBlendPropertyFuncType, VMdlBlendCoordinatesFuncType,

16.2.36 VMdlBlendPropertiesSlice (vmdl_blnd_field.c:966)

```
VMdlSlicerOutputImageStruct *VMdlBlendPropertiesSlice(
                                  VMdlBlendPrepStruct *BlendCoordPrep,
                                  const double z,
                                  int Resolution[2],
                                  VMdlBlendCoordinatesFuncType Coordinates,
                                  VMdlBlendPropertyFuncType PropertyFunction,
                                  VMdlBlendRuleFuncType BlendingRule,
                                  unsigned int SizePixelProperty)
```

BlendCoordPrep: The prep structure for the blend.

z: Z coordinate of the slice of the V-model.

Resolution: Resolution of the image, will be modified by VMdlSlicer, to a near-by actual saved size.

Coordinates: Call-back function for the chosen blending functions.

PropertyFunction: User specified call-back function to define the property to be blended.

BlendingRule: User specified call-back functions to define how to blend the property.

SizePixelProperty: Size of the property in each pixel.

Returns: Output array containing the property values for every point.

Description: Given a VMdlBlendPrepStruct, a 'z' value, an image resolution, and the pointer to two functions for the computation of the coordinates and the user specified one, blends the specified attributes according to the chosen coordinates.
See also: VMdlBlendPrepStruct, VMdlBlendCoordinatesFuncType, , VMdlBlendAttributesPerPointFuncType, VMdlSlicerInitVModel,

16.2.37 VMdlBlendPropertiesSliceOneBatch (vmdl_blnd_field.c:1032)

```
VMdlSlicerOutputImageStruct *VMdlBlendPropertiesSliceOneBatch(
                                  VMdlBlendPrepStruct *BlendCoordPrep,
                                  int BatchSz,
                                  const double *ZVals,
                                  const int Resolution[2],
                                  VMdlBlendCoordinatesFuncType Coordinates,
                                  VMdlBlendPropertyFuncType PropertyFunction,
                                  VMdlBlendRuleFuncType BlendingRule,
                                  unsigned int SizePixelProperty)
```

BlendCoordPrep: The prep structure for the blend.

BatchSz: The size of the batch of slices to compute.

ZVals: An array of size BatchSz of Z coordinate of the slices of the V-model.

Resolution: Resolution of the image, will be modified by VMdlSlicer, to a near-by actual saved size.

Coordinates: Call-back function for the chosen blending functions.

PropertyFunction: User specified call-back function to define the property to be blended.

BlendingRule: User specified call-back functions to define how to blend the property.

SizePixelProperty: Size of the property in each pixel.

Returns: Output array containing the images of the slices for the required Z values.

Description: Given a VMdlBlendPrepStruct, an array of 'z' values, an image resolution, and the pointer to two functions for the computation of the coordinates and the user specified one, blends the specified attributes according to the chosen coordinates for a batch of slices. The slicing is performed in parallel, if enabled.
See also: VMdlBlendPropertiesSlice, VMdlBlendPrepStruct, , VMdlBlendCoordinatesFuncType, VMdlBlendAttributesPerPointFuncType, , VMdlSlicerInitVModel,

16.2.38 VMdlBlendVCellPropertiesPerPoint (vmdl_blnd_field.c:2063)

```
VMdlSlicerPropertyStruct *VMdlBlendVCellPropertiesPerPoint(
                                    VMdlBlendPrepStruct *BlendCoordPrep,
                                    IrtRType Pt[3],
                                    VMdlSlicerPropertyStruct *OutputProperty)
```

BlendCoordPrep: A structure containing useful quantities regarding the V-model.

Pt: Euclidean coordinates of the point for which we need to compute the blend.

OutputProperty: Pointer to be filled with the output.

Returns: Pointer to the output.

Description: Given a V-model and a point blends the attributes of the V-model accordingly, using V-cells.

16.2.39 VMdlBlendVCellPropertiesSlice (vmdl_blnd_field.c:1997)

```
VMdlSlicerOutputImageStruct *VMdlBlendVCellPropertiesSlice(VMdlBlendPrepStruct
                                                           *BlendCoordPrep,
                                                    const double z,
                                                    const int *Resolution)
```

BlendCoordPrep: Prep structure of the V-model.

z: Slice.

Resolution: Vector containing the size of the image.

Returns: Image structure.

Description: Given a VMdl, a z value and a resolution outcomes the result of the blending with coordinates based on V-cells.
See also: VMdlBlendPrepStruct, distance-based,

16.2.40 VMdlBlendVElemComputeDistCoordinates (vmdl_blnd_field.c:1881)

```
void VMdlBlendVElemComputeDistCoordinates(VMdlBlendPrepStruct *BlendCoordPrep,
                                          const CagdRType Pt[3])
```

BlendCoordPrep: Prep structure of the V-model.

Pt: Point of which we need to compute the barycentric coordinates.

Returns: void

Description: Given a VMdl and a point, computes its barycentric coordinates with respect to the boundary surfaces of the union intersection using V-cells.
See also: VMdlBlendPrepStruct, distance-based coordinates, VMdlVolumeElementStruct,

16.2.41 VMdlBoolDefaultParams (vmdl_bool.c:3871)

```
VMdlBoolParamsStruct VMdlBoolDefaultParams()
```

Returns: The default parameters for boolean operations.

Description: Creates a params struct with default values for boolean operations.

16.2.42 VMdlBuildFinalVModel (vmdl_cnst.c:88)

```
VMdlVModelStruct *VMdlBuildFinalVModel(TrivTVStruct *TV,
                                       TrimSrfStruct *TrimSrfs,
                                       CagdRType Tol,
                                       void *Context)
```

TV: TV to convert into VModel, in place.

TrimSrfs: The trimming surfaces, in place.

Tol: Tolerance used for stitching etc.

Context: Any extra data needed for building the VModel.

Returns: Constructed VModel, or NULL if error.

Description: Build a VModel from the given TV and trimming surface.

16.2.43 VMdlBuildMdlFromSurfaces (vmdl_bool.c:1236)

```
MdlModelStruct *VMdlBuildMdlFromSurfaces(const VMdlParamsStruct *Params,
                                         const MdlModelStruct *BSrfs)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

BSrfs: Surfaces models to merge.

Returns: New Model object.

Description: Builds a new Model object by merging a list of surfaces represented each as a model object.

16.2.44 VMdlBuildModelFromTVBndries (vmslicer.c:291)

```
MdlModelStruct *VMdlBuildModelFromTVBndries(const TrivTVStruct *Trivar)
```

Trivar: The trivariate to build a (6 faces) MODEL from its boundaries.

Returns: The created model.

Description: Builds a closed MODEL from the surfaces of a trivariate.
See also: MdlAddSrf2Mdl, MdlCnvrtSrf2Mdl,

16.2.45 VMdlCalcMdlEuclCrvs (vmdl_bool.c:2611)

```
void VMdlCalcMdlEuclCrvs(const VMdlParamsStruct *Params,
                         MdlModelStruct *BMdl)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

BMdl: The Model object.

Returns: void

Description: Computes the Euclidean curves of a given Model object.

16.2.46 VMdlCheckVElementConnectivity (vmdl_dbg.c:247)

```
CagdBType VMdlCheckVElementConnectivity(const VMdlVolumeElementStruct
                                                                    *VMdlElement)
```

VMdlElement: The V-cell.

Returns: Success iff the connectivity of the V-cell is valid.

Description: Tests connectivity validity of a V-cell (V-Element).

16.2.47 VMdlClipVModelByPlane (vmdl_bool.c:3728)

```
VMdlVModelStruct *VMdlClipVModelByPlane(const VMdlParamsStruct *Params,
                                        const VMdlVModelStruct *VMdl,
                                        const IrtPlnType Pln,
                                        VMdlBoolOpType BoolOp)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VMdl: VModel to clip against plane Pln.

Pln: Clipping plane.

BoolOp: One of INTERSECTION, SUBTRACTION.

Returns: Portion of VMdl, as a new vmodel, in the positive side of plane Pln. NULL if nothing.

Description: Clips the given vmodel, VMdl, to the portion that is in the positive side of plane Pln.
See also: MdlClipSrfByPlane, MdlClipTrimmedSrfByPlane, MdlClipModelByPlane,

16.2.48 VMdlCnvrtSrf2VMdl (vmdl_aux.c:137)

```
VMdlVModelStruct *VMdlCnvrtSrf2VMdl(const CagdSrfStruct *Srf)
```

Srf: list of boundary surfaces.

Returns: New V-model.

Description: Converts a list of surfaces forming a closed shape into a V-model, having the surface as its boundary.

16.2.49 VMdlCnvrtTrimmedSrf2VMdl (vmdl_aux.c:157)

```
VMdlVModelStruct *VMdlCnvrtTrimmedSrf2VMdl(const TrimSrfStruct *TSrf)
```

TSrf: list of boundary trimmed surfaces.

Returns: New V-model.

Description: Converts a list of trimmed surfaces forming a closed shape into a V-model having the surface as its boundary.

16.2.50 VMdlCnvrtTrivar2VMdl (vmdl_bool.c:812)

```
VMdlVModelStruct *VMdlCnvrtTrivar2VMdl(const VMdlParamsStruct *Params,
                                       const TrivTVStruct *TV,
                                       int PrimID)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

TV: The trivariate to convert to VModel.

PrimID: ID to use for this "VPrimitiveID" attribute, or 0 to assign a whole new ID.

Returns: The constructed V-model.

Description: Builds a V-model from a given trivariate. Having one V-cell, and six boundary V-surfaces, which are the boundary surfaces of the trivariate.

16.2.51 VMdlCnvrtTrivarList2VMdl (vmdl_bool.c:743)

```
VMdlVModelStruct *VMdlCnvrtTrivarList2VMdl(const VMdlParamsStruct *Params,
                                           TrivTVStruct *TVList,
                                           int PrimID)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

TVList: List of trivariates.

PrimID: ID to use for this "VPrimitiveID" attribute, or 0 to assign a whole new ID.

Returns: Constructed V-model.

Description: Creates a V-model from several trivariates. The function assumes that trivariates don't overlap. A V-cell is constructed for each trivariate that are glued together then.

16.2.52 VMdlCnvrtVMdl2Mdl (vmdl_aux.c:313)

```
MdlModelStruct *VMdlCnvrtVMdl2Mdl(const VMdlParamsStruct *Params,
                                  const VMdlVModelStruct *VMdl)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VMdl: The V-model.

Returns: List of boundry Models of the V-cells.

Description: Converts all the V-cells of a given V-model into a list of boundary Models.
See also: VMdlCnvrtVMdls2Mdls,

16.2.53 VMdlCnvrtVMdl2TrimmedSrfs (vmdl_aux.c:231)

```
TrimSrfStruct *VMdlCnvrtVMdl2TrimmedSrfs(const VMdlParamsStruct *Params,
                                         const VMdlVModelStruct *VMdl)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VMdl: The V-model.

Returns: List of trimmed surfaces representing all the half V-surfaces.

Description: Extracts the boundary half-surfaces of all the V-cells in a given V-model into a list of trimmed surfaces.
See also: VMdlCnvrtVMdls2TrimmedSrfs,

16.2.54 VMdlCnvrtVMdls2Mdls (vmdl_aux.c:274)

```
MdlModelStruct *VMdlCnvrtVMdls2Mdls(const VMdlParamsStruct *Params,
                                    const VMdlVModelStruct *VMdls)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VMdls: The linked list of V-models.

Returns: List of boundry Models of the V-cells.

Description: Converts all the V-cells of a given list of V-models into a list of boundary Models.
See also: VMdlCnvrtVMdl2Mdl,

16.2.55 VMdlCnvrtVMdls2TrimmedSrfs (vmdl_aux.c:184)

```
TrimSrfStruct *VMdlCnvrtVMdls2TrimmedSrfs(const VMdlParamsStruct *Params,
                                          const VMdlVModelStruct *VMdls)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VMdls: The linked list of V-models.

Returns: List of trimmed surfaces representing all the half V-surfaces.

Description: Extracts the boundary half-surfaces of all the V-cells in a given list of V-models into a list of trimmed surfaces.
See also: VMdlCnvrtVMdl2TrimmedSrfs,

16.2.56 VMdlCnvrtVSrf2TrimmedSrf (vmdl_aux.c:643)

```
TrimSrfStruct *VMdlCnvrtVSrf2TrimmedSrf(const VMdlInterTrimSrfStruct *VSrf)
```

VSrf: The V-surface to convert.

Returns: Trimmed surface structure converted from VSrf.

Description: Converts a V-surface to trimmed surface.
See also: VMdlApplyVSrfTrimmedLoopsOnSrf,

16.2.57 VMdlCreateFillet (vmdl_fillet.c:792)


```
void VMdlCreateFillet(const VMdlParamsStruct *Params,
                      const CagdSrfStruct *Srf1List,
                      const CagdSrfStruct *Srf2List,
                      CagdRType RailDist,
                      int R1Orient,
                      int R2Orient,
                      CagdRType TanScale,
                      int ApproxCrvsCtlPts,
                      TrivFilletingMethodType FilletingMethod,
                      CagdBType PreciseFillet,
                      TrivTVStruct **FilletTVs,
                      MdlModelStruct **FilletMdls,
                      CagdSrfStruct **SubSrfs1,
                      CagdSrfStruct **SubSrfs2)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

Srf1List, Srf2List: Two list of surfaces to construct a fillet in between each pair of them. *

RailDist: The distance of the rail curves from the intersection curve of Srf1 and Srf2

R1Orient, R2Orient: +/-1 for left/right orientation of the corresponding rail curve. 0 to choose the option that results with a longer rail curve.

TanScale: The desired magnitude of the fillet's tangents that connect it with Srf1 and Srf2.

ApproxCrvsCtlPts: The number of control points used to approximate the curves used for fillet construction.

FilletingMethod: The required filleting method

PreciseFillet: If true, the primary surfaces of the fillet are computed precisely, using surface-surface composition. Otherwise, the primary surafces are approximated, using least square fitting.

FilletTVs: If not NULL, the list of fillet trivariates will be stored here.

FilletMdls: If not NULL, a list of models representing the fillets' boundary surfaces will be stored here

SubSrfs1: If not NULL, a list of the fillets' shared boundaries with TV1 will be stored here.

SubSrfs2: If not NULL, a list of the fillets' shared boundaries with TV1 will be stored here.

Returns: void.

Description: Constructs a (list of) fillet trivariate(s) between the two given boundary surfaces, and creates a (list of) model(s) representing the trivariate(s) boundary surfaces. The fillet meets with the surfaces with G1 continuity, and its boundary curves are the intersection curve of Srf1 and Srf2, and two rail curves that are computed as an approximate Euclidean offest of the intersection curve on each of the surfaces.
See also: TrivTVFillet,

16.2.58 VMdlCreateFillet2 (vmdl_fillet.c:862)

VModels
blending
filleting

```
VMdlVModelStruct *VMdlCreateFillet2(const VMdlParamsStruct *Params,
                                    const CagdSrfStruct *Srf1List,
                                    const CagdSrfStruct *Srf2List,
                                    CagdRType RailDist,
                                    int R1Orient,
                                    int R2Orient,
                                    CagdRType TanScale,
                                    int ApproxCrvsCtlPts,
                                    TrivFilletingMethodType FilletingMethod,
                                    CagdBType PreciseFillet)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

Srf1List, Srf2List: Two list of surfaces to construct a fillet in between each pair of them. *

RailDist: The distance of the rail curves from the intersection curve of Srf1 and Srf2

R1Orient, R2Orient: +/-1 for left/right orientation of the corresponding rail curve. 0 to choose the option that results with a longer rail curve.

TanScale: The desired magnitude of the fillet's tangents that connect it with Srf1 and Srf2.

ApproxCrvsCtlPts: The number of control points used to approximate the curves used for fillet construction.

FilletingMethod: The required filleting method

PreciseFillet: If true, the primary surfaces of the fillet are computed precisely, using surface-surface composition. Otherwise, the primary surafces are approximated, using least square fitting.

Returns: A VModel representing the volumetric fillet.

Description: Constructs a VModel representing a volumetric fillet between the two given boundary surfaces.
See also: VMdlCreateFillet, TrivTVFillet,

16.2.59 VMdlDbg (vmdl_dbg.c:100)

debugging

```
void VMdlDbg(void *Obj)
```

Obj: A model object - to be printed to stderr.

Returns: void

Description: Prints vmodel objects to stderr. Should be linked to programs for debugging purposes, so model objects may be inspected from a debugger.
See also: VMdlDbg1,

16.2.60 VMdlDbg1 (vmdl_dbg.c:129)

debugging

```
void VMdlDbg1(void *Obj)
```

Obj: A model object - to be printed to stderr.

Returns: void

Description: Prints vmodel objects to stderr. Should be linked to programs for debugging purposes, so model objects may be inspected from a debugger.
See also: VMdlDbg,

16.2.61 VMdlDebugVerify (vmdl_dbg.c:307)

```
CagdBType VMdlDebugVerify(const VMdlVModelStruct *VMdl)
```

VMdl: The VModel to verify.

Returns: TRUE if the the data structure is valid, FALSE otherwise.

Description: Verifies the validity of the pointers of a VModel.

16.2.62 VMdlDebugVerifyPrint (vmdl_dbg.c:358)

```
void VMdlDebugVerifyPrint(const VMdlVModelStruct *VMdl)
```

VMdl: The VModel to verify.

Returns: void

Description: Verifies the validity of the pointers of a VModel. Prints an error message if any issues are found.
See also: VMdlDebugVerify,

16.2.63 VMdlDefaultParams (vmdl_aux.c:1202)

```
void VMdlDefaultParams(VMdlParamsStruct *Params)
```

Params: A pointer to a parameters struct. The values in theis struct will be filled with default values.

Returns: void

Description: Sets default values for VModel operation parameters.
See also:

16.2.64 VMdlEncDflt1DVectorField (vmdl_enc_field.c:1187)

```
size_t VMdlEncDflt1DVectorField(VMdlBlendPrepStruct *BlendCoordPrep,
                                VMdlBlendPointStruct *Pt,
                                VMdlSlicerPropertyStruct *PropertyOut)
```

BlendCoordPrep: A structure containing useful quantities regarding the V-model.

Pt: Euclidean coordinates of the point for which we need to compute the blend.

PropertyOut: Property to be filled with the output.

Returns: Dimension of the property function.

Description: Given a VMdlBlendPrepStruct and a point structure, returns the vector field (1D - gray scale) associated to a V-model and its dimensional size.
See also: VMdlBlendPrepStruct, VMdlBlendPointStruct, , VMdlBlendDfltPropertiesPerPoint,

16.2.65 VMdlEncDflt3DVectorField (vmdl_enc_field.c:1139)

```
size_t VMdlEncDflt3DVectorField(VMdlBlendPrepStruct *BlendCoordPrep,
                                VMdlBlendPointStruct *Pt,
                                VMdlSlicerPropertyStruct *PropertyOut)
```

BlendCoordPrep: A structure containing useful quantities regarding the V-model.

Pt: Euclidean coordinates of the point for which we need to compute the blend.

PropertyOut: Property to be filled with the output.

Returns: Dimension of the property function.

Description: Given a VMdlBlendPrepStruct and a point structure, returns the vector field (3D) associated to a V-model and its dimensional size.
See also: VMdlBlendPrepStruct, VMdlBlendPointStruct, , VMdlBlendDfltPropertiesPerPoint,

16.2.66 VMdlEncDfltRGBVectorField (vmdl_enc_field.c:1092)

```
size_t VMdlEncDfltRGBVectorField(VMdlBlendPrepStruct *BlendCoordPrep,
                                 VMdlBlendPointStruct *Pt,
                                 VMdlSlicerPropertyStruct *PropertyOut)
```

BlendCoordPrep: A structure containing useful quantities regarding the V-model.

Pt: Euclidean coordinates of the point for which we need to compute the blend.

PropertyOut: Property to be filled with the output.

Returns: Dimension of the property function.

Description: Given a VMdlBlendPrepStruct and a point structure, returns the vector field (RGB colors) associated to a V-model and its dimensional size.
See also: VMdlBlendPrepStruct, VMdlBlendPointStruct, , VMdlBlendDfltPropertiesPerPoint,

16.2.67 VMdlEncPropertyFunctionError (vmdl_enc_field.c:1902)

```
IrtRType VMdlEncPropertyFunctionError(
                              const VMdlParamsStruct *Params,
                              VMdlBlendPrepStruct *BlendCoordPrep,
                              const int Resolution[3],
                              VMdlBlendPropertyFuncType PropertyFunction,
                              VMdlBlendPropertyFuncType ReconPropertyFunction,
                              VMdlBlendCoordinatesFuncType Coordinates)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

BlendCoordPrep: Prep handle for blending.

Resolution: Resolution of the 3D grid over which computing the RMSE.

PropertyFunction: Original property function.

ReconPropertyFunction: Reconstructed property functions.

Coordinates: Coordinates function for blending.

Returns: weighted average of the RMSE of the reconstruction.

Description: Routine to compute the RMSE of all the property functions in a V-model.

16.2.68 VMdlEncRetrieveProperties (vmdl_enc_field.c:771)

Least square approximation

```
TrivTVStruct *VMdlEncRetrieveProperties(
                                    VMdlBlendPrepStruct *BlendCoordPrep,
                                    const int NumSamplePoints,
                                    CagdBType CreateNewTriv,
                                    const int AddedRefinedKnots,
                                    VMdlEncVectorFieldFuncType F,
                                    VMdlBlendCoordinatesFuncType Coordinates)
```

BlendCoordPrep: Prep structure of the V-model.

NumSamplePoints: Of the discretization of the parametric spaces of TVs in BlendCoordPrep.

CreateNewTriv: If TRUE outputs a trivariate struct, otherwise saves the output in the original TVs.

AddedRefinedKnots: If positive, knows to added to the property.

F: Vector field over the V-model.

Coordinates: Optional call back function to compute distance.

Returns: TVs interpolating the original properties.

Description: Outputs the result of the least square for retrieving the original properties, before the blend. The output is given as a TrivTVStruct or encoded into the geometry, depending on user specifications.
See also: TrivTVStruct, VMdlBlendPropertyFunctionFuncType, VMdlBlendPrepStruct, , ConstructPointStruct, VMdlBlendAssembleLstSqrInverseMatrix,

16.2.69 VMdlEncVModelVectorFieldError (vmdl_enc_field.c:1972)

```
IrtRType VMdlEncVModelVectorFieldError(
                                VMdlBlendPrepStruct *BlendCoordPrep,
                                const int Resolution[3],
                                VMdlEncVectorFieldFuncType VectorField,
                                VMdlBlendPropertyFuncType ReconPropertyFunction,
                                VMdlBlendCoordinatesFuncType Coordinates,
                                VMdlBlendRuleFuncType BlendingRule)
```

BlendCoordPrep: Prep handle for blending.

Resolution: Resolution of the 3D grid over which computing the RMSE.

VectorField: Original vector field.

ReconPropertyFunction: Reconstructed property functions.

Coordinates: Coordinates function for blending.

BlendingRule: User specified call-back functions to define how to blend the properties.

Returns: weighted average of the RMSE of the reconstruction.

Description: Routine to compute the RMSE of reconstructed vector field over a V-model.

16.2.70 VMdlEvalEuclidCrvsForTrimCrvs (vmdl_cnst.c:40)

```
void VMdlEvalEuclidCrvsForTrimCrvs(TrimSrfStruct *TrSrf)
```

TrSrf: Trim-surface to populate trim-curves of, in place.

Returns: void

Description: Evaluates and populates the Euclidean trim-curves of the input trim- surface from its UV curves, in place.

16.2.71 VMdlExtractTrimCrvLoop (vmdl_bool.c:1339)

```
CagdCrvStruct **VMdlExtractTrimCrvLoop(const VMdlInterTrimSrfStruct
                                                              *VTrimSrfStruct,
                                       int *NumLoops)
```

VTrimSrfStruct: V-surface.

NumLoops: Output parameter, which holds the number of trimming curves.

Returns: Array of trimming curves.

Description: Returns an array (copied) of trimming curves of a given V-surface.

16.2.72 VMdlExtractVElements (vmdl_aux.c:411)

```
VMdlVModelStruct *VMdlExtractVElements(const VMdlVModelStruct *VMdl)
```

VMdl: The V-model to divide into its individual V-elements, each in one V-Model.

Returns: List of V-models, one V-model for each V-element.

Description: Extracts the V-elements of a given V-model into a list of V-models. each V-element is wrapped by a new V-model. (The V-elements are copied).

16.2.73 VMdlExtrudeTrimSrf (vmdl_cnst.c:283)

```
VMdlVModelStruct *VMdlExtrudeTrimSrf(const TrimSrfStruct *Section,
                                     CagdVecStruct *Dir)
```

Section: Input trim surface to extrude.

Dir: Direction to extrude along.

Returns: The extruded V-model.

Description: Computes an extruded V-model from input trim surface, in the given direction.

16.2.74 VMdlExtrudeTrimSrfExtra (vmdl_cnst.c:310)

```
VMdlVModelStruct *VMdlExtrudeTrimSrfExtra(const VMdlParamsStruct *Params,
                                          const TrimSrfStruct *Section,
                                          CagdVecStruct *Dir,
                                          void *ExtraData)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

Section: Input trim surface to extrude.

Dir: Direction to extrude along.

ExtraData: Extra data to pass to the stitching function.

Returns: The extruded V-model.

Description: Computes an extruded V-model from input trim surface, in the given direction. This function allows passing extra data to the stitching function.

16.2.75 VMdlFilletDefaultParams (vmdl_fillet.c:1004)

sweep

trivariate constructors

```
VMdlFilletParamsStruct VMdlFilletDefaultParams()
```

Returns: The default parameters for for filleting.

Description: Creates a params struct with default values for filleting.
See also:

16.2.76 VMdlFreeInterTrimCurveSeg (vmdl_gen.c:414)

```
void VMdlFreeInterTrimCurveSeg(VMdlInterTrimCurveSegStruct *VTrimCrv)
```

VTrimCrv: V-curve to deallocate.

Returns: void

Description: Frees a V-curve.

16.2.77 VMdlFreeInterTrimCurveSegLoopInSrf (vmdl_gen.c:508)

```
void VMdlFreeInterTrimCurveSegLoopInSrf(
                                VMdlInterTrimCurveSegLoopInSrfStruct *CrvSegLoop)
```

CrvSegLoop: A trimming loop of a half-V-surface to free.

Returns: void

Description: Frees a trimming loop of a half-V-surface.

16.2.78 VMdlFreeInterTrimCurveSegRef (vmdl_gen.c:453)

```
void VMdlFreeInterTrimCurveSegRef(VMdlInterTrimCurveSegRefStruct *CrvSegRef)
```

CrvSegRef: Reference to V-curve to deallocate.

Returns: void

Description: Frees a reference to a V-curve.

16.2.79 VMdlFreeInterTrimPnt (vmdl_gen.c:368)

```
void VMdlFreeInterTrimPnt(VMdlInterTrimPointStruct *VTrimPnt)
```

VTrimPnt: To deallocate.

Returns: void

Description: Frees a V-point.

16.2.80 VMdlFreeInterTrimPointRef (vmdl_gen.c:538)

```
void VMdlFreeInterTrimPointRef(VMdlInterTrimPointRefStruct *PntRef)
```

PntRef: Reference to a V-point to deallocate.

Returns: void

Description: Frees a reference to a V-point.

16.2.81 VMdlFreeInterTrimSrf (vmdl_gen.c:386)

```
void VMdlFreeInterTrimSrf(VMdlInterTrimSrfStruct *VTrimSrf)
```

VTrimSrf: Half-V-surface to deallocate.

Returns: void

Description: Frees a half-V-surface.

16.2.82 VMdlFreeInterTrimSrfRef (vmdl_gen.c:489)

```
void VMdlFreeInterTrimSrfRef(VMdlInterTrimSrfRefStruct *SrfRef)
```

SrfRef: Reference to a half-V-surface to deallocate.

Returns: void

Description: Frees a reference to a half-V-surface.

16.2.83 VMdlFreeInterTrivTVRef (vmdl_gen.c:471)

```
void VMdlFreeInterTrivTVRef(VMdlInterTrivTVRefStruct *TrivTVRef)
```

TrivTVRef: Reference to a trivariate to deallocate.

Returns: void

Description: Frees a reference to a trivariate.

16.2.84 VMdlFreeOneModel (vmdl_gen.c:264)

```
static void VMdlFreeOneModel(VMdlVModelStruct *Mdl)
```

Mdl: V-model to deallocate.

Returns: void

Description: Frees a V-model.

16.2.85 VMdlFreeTrimVolElem (vmdl_gen.c:345)

```
void VMdlFreeTrimVolElem(VMdlVolumeElementStruct *VCell)
```

VCell: V-cell to deallocate.

Returns: void

Description: Frees a V-cell.

16.2.86 VMdlFreeTrimVolumeElemRef (vmdl_gen.c:435)

```
void VMdlFreeTrimVolumeElemRef(VMdlVolumeElementRefStruct *TrimVolElemRef)
```

TrimVolElemRef: Reference to V-cell to deallocate.

Returns: void

Description: Frees a reference to a V-cell.

16.2.87 VMdlFromBoundaryModel (vmdl_bool.c:2541)

```
VMdlVModelStruct *VMdlFromBoundaryModel(const VMdlParamsStruct *Params,
                                        const MdlModelStruct *BMdl,
                                        const TrivTVStruct *TV)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

BMdl: Boundary model to serve as a set of trimming surfaces of the VModel.

TV: The trivariate volume of the VModel.

Returns: Constructed VModel.

Description: A constructore f or a VModel from a trivariate and a (boundary) model.
See also: VMdlVolElementFromBoundaryModel,

16.2.88 VMdlGetBndryVElement (vmdl_bool.c:1468)

```
MdlModelStruct *VMdlGetBndryVElement(const VMdlParamsStruct *Params,
                                     VMdlVolumeElementStruct *VCell)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VCell: The V-cell.

Returns: Boundary Model of the V-cell.

Description: Return a boundary Model of a given V-cell. The Model object is created if it was not created before.

16.2.89 VMdlGetBoundaryCurves (vmdl_bool.c:1588)

```
CagdCrvStruct *VMdlGetBoundaryCurves(const VMdlVModelStruct *Mdl)
```

Mdl: The V-model.

Returns: List of curves.

Description: Returns a list of all the V-curves geometry copies in a given V-model.

16.2.90 VMdlGetBoundarySurfaces2 (vmdl_bool.c:1539)

```
IPObjectStruct *VMdlGetBoundarySurfaces2(const VMdlParamsStruct *Params,
                                         const VMdlVModelStruct *Mdl)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

Mdl: The V-model.

Returns: Object of trimmed surfaces list.

Description: Returns the boundary surfaces of each V-cell in a given V-model as a trimmed surfaces object struct.

16.2.91 VMdlGetBoundaryVModel (vmdl_bool.c:1155)

```
MdlModelStruct *VMdlGetBoundaryVModel(const VMdlParamsStruct *Params,
                                      const VMdlVModelStruct *Mdl)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

Mdl: A V-model to convert to a model (B-rep) only.

Returns: List of Model objects of the boundaries of the V-model's V-cells.

Description: Converts each V-cell of a given V-model into a Model object.

16.2.92 VMdlGetOuterBoundarySurfacesVModel (vmdl_bool.c:1388)

```
MdlModelStruct *VMdlGetOuterBoundarySurfacesVModel(
                                                const VMdlParamsStruct *Params,
                                                const VMdlVModelStruct *VMdl)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VMdl: The V-model.

Returns: Boundary Model.

Description: Constructs the outer boundary composed of the boundary half-V-surfaces of a given V-model as a Model object.

16.2.93 VMdlGetStitchCallback (vmdl_aux.c:1135)

```
VMdlTSrfs2StitchedTSrfsFuncType VMdlGetStitchCallback(void)
```

Returns: The stitching function reference.

Description: Returns the callback function for stitching trimmed surfaces into the boundary of a V-element.

16.2.94 VMdlGetTVBoundarySrf (vmdl_aux.c:907)

```
const TrivTVStruct *VMdlGetTVBoundarySrf(const VMdlVModelStruct *VMdl,
                                         const CagdSrfStruct *Srf,
                                         TrivTVBndryType *TVBndry,
                                         int *Reversed)
```

VMdl: VModel to search the trivariate in.

Srf: The surface to seek its trivariate, for which Srf is a boundary.

TVBndry: Boundary specification of Srf in the trivariate, if found.

Reversed: Optional parameter to state how the UV parameters of Srf were flipped/reversed in the match as the following bits, 1 - U was reversed, 2 - V was reversed, 4 - if U and V were flipped.

Returns: The found trivariate or NULL if none.

Description: Finds, if any, the trivariate that Srf is a boundary surface of, in the given VModel. This function does the search geometrically and does not take into account the topological information in VMdl.
See also: VMdlGetTVBoundarySrfEps,

16.2.95 VMdlGetTVBoundarySrfEps (vmdl_aux.c:789)

```
const TrivTVStruct *VMdlGetTVBoundarySrfEps(const VMdlVModelStruct *VMdl,
                                            const CagdSrfStruct *Srf,
                                            TrivTVBndryType *TVBndry,
                                            int *Reversed,
                                            CagdRType Tol)
```

VMdl: VModel to search the trivariate in.

Srf: The surface to seek its trivariate, for which Srf is a boundary.

TVBndry: Boundary specification of Srf in the trivariate, if found.

Reversed: Optional parameter to state how the UV parameters of Srf were flipped/reversed in the match as the following bits, 1 - U was reversed, 2 - V was reversed, 4 - if U and V were flipped.

Tol: The tolerance.

Returns: The found trivariate or NULL if none.

Description: Finds, if any, the trivariate that Srf is a boundary surface of, in the given VModel. This function does the search geometrically and does not take into account the topological information in VMdl.

16.2.96 VMdlGlueVModels2 (vmdl_bool.c:1859)

```
void VMdlGlueVModels2(VMdlVModelStruct **ResVMdl,
                      VMdlVModelStruct **OtherVMdls,
                      int NumVMdls,
                      CagdRType SrfDiffEps,
                      CagdBType CalculateConnectivity)
```

ResVMdl: The result glued V-model.

OtherVMdls: Array of V-models to Glue.

NumVMdls: Number of V-models in OtherVMdls.

SrfDiffEps: Surface comparison tolerance.

CalculateConnectivity: If True, topological connectivity is computed.

Returns: void

Description: This function glues list of V-models. The result is saved in ResVMdl input/output parameter.

16.2.97 VMdlGlueVModelsAppend (vmdl_bool.c:1916)

```
CagdBType VMdlGlueVModelsAppend(VMdlVModelStruct **Mdl1,
                                const VMdlVModelStruct *Mdl2,
                                CagdRType SrfDiffEps,
                                CagdBType CalculateConnectivity,
                                MiscPHashMap TVHMap)
```

Mdl1: First V-model.

Mdl2: Second V-model.

SrfDiffEps: Surface comparison tolerance.

CalculateConnectivity: If True, topological connectivity is computed.

TVHMap: The Hashing map.

Returns: TRUE in success, and FALSE otherwise.

Description: This function glues two V-models. The result is saved in the first one.

16.2.98 VMdlInterTrimCurveSegCopy (vmdl_gen.c:557)

```
VMdlInterTrimCurveSegStruct *VMdlInterTrimCurveSegCopy(
                                        VMdlInterTrimCurveSegStruct *CurveSeg)
```

CurveSeg: A V-curve to copy.

Returns: New copy of the given V-curve.

Description: Copies a V-curve.

16.2.99 VMdlInterTrimCurveSegLoopInSrfCopy (vmdl_gen.c:889)

```
VMdlInterTrimCurveSegLoopInSrfStruct *VMdlInterTrimCurveSegLoopInSrfCopy(
                                VMdlInterTrimCurveSegLoopInSrfStruct *CrvInLoop)
```

CrvInLoop: A curve of a loop in a half-V-surface to copy.

Returns: New copy of the given curve in loop.

Description: Copies a trimming curve in a trimming loop in a half-V-surface.

16.2.100 VMdlInterTrimPointCopy (vmdl_gen.c:995)

```
VMdlInterTrimPointStruct *VMdlInterTrimPointCopy(VMdlInterTrimPointStruct
                                                                    *VMdlPnt)
```

VMdlPnt: A V-point to copy.

Returns: New copy of the given V-point.

Description: Copies a V-point.

16.2.101 VMdlInterTrimSrfSCopy (vmdl_gen.c:923)

```
VMdlInterTrimSrfStruct *VMdlInterTrimSrfSCopy(VMdlInterTrimSrfStruct *VMdlSrf)
```

VMdlSrf: A half-V-surface to copy.

Returns: New copy of the given half-V-surface.

Description: Copies a half-V-surface.

16.2.102 VMdlIsNonTrimmedVCell (vmdl_aux.c:579)

```
TrivTVStruct *VMdlIsNonTrimmedVCell(const VMdlParamsStruct *Params,
                                    const VMdlVolumeElementStruct *VElem)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VElem: The V-element.

Returns: If the V-element is in a non-trimmed box a sub-domain of one of its trivariates, the trivariate is returned, and NULL otherwise.

Description: Checks if the given V-element (V-cell) is non-trimmed in the parametric domain of one of its associated (containing) trivariates.

16.2.103 VMdlIsNonTrimmedVCellOfTV (vmdl_aux.c:508)

```
CagdBType VMdlIsNonTrimmedVCellOfTV(const VMdlParamsStruct *Params,
                                    const VMdlVolumeElementStruct *VElem,
                                    const TrivTVStruct *TV)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VElem: The V-element.

TV: The trivariate.

Returns: TRUE iff the V-element is in a non-trimmed box a sub-domain of the trivariate, and NULL otherwise.

Description: Checks if the given V-element (V-cell) is non-trimmed in the parametric domain of a given trivariate.

16.2.104 VMdlIsNonTrimmedVModel (vmdl_aux.c:615)

```
TrivTVStruct *VMdlIsNonTrimmedVModel(const VMdlParamsStruct *Params,
                                     const VMdlVModelStruct *VMdl)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VMdl: The V-model.

Returns: If the V-model is in a non-trimmed box a sub-domain of one of its trivariates, the trivariate is returned, and NULL otherwise.

Description: Checks if the given V-model is non-trimmed in the parametric domain of one of its associated (containing) trivariates. The V-model should contain only one V-element.

16.2.105 VMdlIsTVBoundaryVSrf (vmdl_aux.c:733)

```
CagdBType VMdlIsTVBoundaryVSrf(const VMdlInterTrimSrfStruct *VSrf,
                               const TrivTVStruct *TV,
                               TrivTVDirType *IsoDir,
                               CagdRType *IsoVal)
```

VSrf: The V-surface to check.

TV: The trivariate.

IsoDir, IsoVal: Will hold the iso parametric direction and value, if VSrf lies on iso-parametric surface of TV.

Returns: TRUE iff VSrf lies on iso-parametric surface of TV.

Description: Checks if the surface of a given V-surface is an iso parametric surface of a given trivariate, by examining special attributes saved in the both trivariate and V-surface.

16.2.106 VMdlOfRevAxisTrimSrf (vmdl_cnst.c:575)

trivariate of revolution

trivariate constructors

```
VMdlVModelStruct *VMdlOfRevAxisTrimSrf(const TrimSrfStruct *Section,
                                       const TrivV4DType AxisPoint,
                                       const TrivV4DType AxisVector,
                                       CagdRType StartAngle,
                                       CagdRType EndAngle,
                                       CagdBType Rational)
```

Section: Input trim surface to revolve.

AxisPoint: Of axis of rotation of Srf.

AxisVector: Of axis of rotation of Srf.

StartAngle: Starting angle of revolution, in degrees.

EndAngle: Ending angle of revolution, in degrees.

Rational: TRUE for precise rational computation, FALSE for a polynomial approximation.

Returns: The revolved V-model.

Description: Computes an revolved V-model from the input trim surface, around the specified axis.
See also: VMdlOfRevTrimSrf,

16.2.107 VMdlOfRevAxisTrimSrfExtra (vmdl_cnst.c:615)

trivariate of revolution

trivariate constructors

```
VMdlVModelStruct *VMdlOfRevAxisTrimSrfExtra(const VMdlParamsStruct *Params,
                                            const TrimSrfStruct *Section,
                                            const TrivV4DType AxisPoint,
                                            const TrivV4DType AxisVector,
                                            CagdRType StartAngle,
                                            CagdRType EndAngle,
                                            CagdBType Rational,
                                            void *ExtraData)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

Section: Input trim surface to revolve.

AxisPoint: Of axis of rotation of Srf.

AxisVector: Of axis of rotation of Srf.

StartAngle: Starting angle of revolution, in degrees.

EndAngle: Ending angle of revolution, in degrees.

Rational: TRUE for precise rational computation, FALSE for a polynomial approximation.

ExtraData: Extra data to pass to the stitching function.

Returns: The revolved V-model.

Description: Computes an revolved V-model from the input trim surface, around the specified axis.
See also: VMdlOfRevTrimSrf, VMdlOfRevAxisTrimSrf,

16.2.108 VMdlOfRevTrimSrf (vmdl_cnst.c:405)

```
VMdlVModelStruct *VMdlOfRevTrimSrf(const TrimSrfStruct *Section,
                                   CagdRType StartAngle,
                                   CagdRType EndAngle,
                                   CagdBType Rational)
```

Section: Input trim surface to revolve.

StartAngle: Starting angle of revolution, in degrees.

EndAngle: Ending angle of revolution, in degrees.

Rational: TRUE for precise rational computation, FALSE for a polynomial approximation.

Returns: The revolved V-model.

Description: Computes a revolved V-model, around the z-axis of the trim surface.
See also: VMdlOfRevAxisTrimSrf,

16.2.109 VMdlOfRevTrimSrfExtra (vmdl_cnst.c:439)

```
VMdlVModelStruct *VMdlOfRevTrimSrfExtra(const VMdlParamsStruct *Params,
                                        const TrimSrfStruct *Section,
                                        CagdRType StartAngle,
                                        CagdRType EndAngle,
                                        CagdBType Rational,
                                        void *ExtraData)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

Section: Input trim surface to revolve.

StartAngle: Starting angle of revolution, in degrees.

EndAngle: Ending angle of revolution, in degrees.

Rational: TRUE for precise rational computation, FALSE for a polynomial approximation.

ExtraData: Extra data to pass to the stitching function.

Returns: The revolved V-model.

Description: Computes a revolved V-model, around the z-axis of the trim surface.
See also: VMdlOfRevTrimSrf, VMdlOfRevAxisTrimSrf,

16.2.110 VMdlPrimBoxVMdl (vmdl_prim.c:76)

```
VMdlVModelStruct *VMdlPrimBoxVMdl(CagdRType MinX,
                                  CagdRType MinY,
                                  CagdRType MinZ,
                                  CagdRType MaxX,
                                  CagdRType MaxY,
                                  CagdRType MaxZ)
```

MinX, MinY, MinZ: Minimal coordinates of the box.

MaxX, MaxY, MaxZ: Maximal coordinates of the box.

Returns: Constructed V-model.

Description: Constructs V-model of a box
See also: VMdlPrimBoxVMdl2,

16.2.111 VMdlPrimBoxVMdl2 (vmdl_prim.c:36)

```
VMdlVModelStruct *VMdlPrimBoxVMdl2(const VMdlParamsStruct *Params,
                                   CagdRType MinX,
                                   CagdRType MinY,
                                   CagdRType MinZ,
                                   CagdRType MaxX,
                                   CagdRType MaxY,
                                   CagdRType MaxZ)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

MinX, MinY, MinZ: Minimal coordinates of the box.

MaxX, MaxY, MaxZ: Maximal coordinates of the box.

Returns: Constructed V-model.

Description: Constructs V-model of a box
See also: VMdlPrimBoxVMdl,

16.2.112 VMdlPrimCone2VMdl (vmdl_prim.c:417)

```
VMdlVModelStruct *VMdlPrimCone2VMdl(const CagdVType Center,
                                    CagdRType MajorRadius,
                                    CagdRType MinorRadius,
                                    CagdRType Height,
                                    CagdBType Rational,
                                    CagdRType InternalCubeEdge)
```

Center: Center of the cone.

MajorRadius: MajorRadius of the bottom base.

MinorRadius: MajorRadius of the bottom base.

Height: Height of the cone.

Rational: Indicates a rational or polynomial spline of the result.

InternalCubeEdge: Internal cube edge length. If zero, a single trivariate is constructed that is singular only on the boundary.

Returns: Constructed V-model.

Description: Constructs V-model of a generalized cone, composed of five extruded V-cells, an inner cube, and four outer V-cells all glued together into one V-model.
See also: VMdlPrimCone2VMdl2,

16.2.113 VMdlPrimCone2VMdl2 (vmdl_prim.c:365)

```
VMdlVModelStruct *VMdlPrimCone2VMdl2(const VMdlParamsStruct *Params,
                                     const CagdVType Center,
                                     CagdRType MajorRadius,
                                     CagdRType MinorRadius,
                                     CagdRType Height,
                                     CagdBType Rational,
                                     CagdRType InternalCubeEdge)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

Center: Center of the cone.

MajorRadius: MajorRadius of the bottom base.

MinorRadius: MajorRadius of the bottom base.

Height: Height of the cone.

Rational: Indicates a rational or polynomial spline of the result.

InternalCubeEdge: Internal cube edge length. If zero, a single trivariate is constructed that is singular only on the boundary.

Returns: Constructed V-model.

Description: Constructs V-model of a generalized cone, composed of five extruded V-cells, an inner cube, and four outer V-cells all glued together into one V-model.
See also: VMdlPrimCone2VMdl,

16.2.114 VMdlPrimConeVMdl (vmdl_prim.c:326)

```
VMdlVModelStruct *VMdlPrimConeVMdl(const CagdVType Center,
                                   CagdRType Radius,
                                   CagdRType Height,
                                   CagdBType Rational,
                                   CagdRType InternalCubeEdge)
```

Center: Center of the cone.

Radius: Radius of the base.

Height: Height of the cone.

Rational: Indicates a rational or polynomial spline of the result.

InternalCubeEdge: Internal cube edge length. If zero, a single trivariate is constructed that is singular only on the boundary.

Returns: Constructed V-model.

Description: Constructs V-model of a cone, composed of five extruded V-cells, an inner cube, and four outer V-cells all glued together into one V-model.
See also: VMdlPrimConeVMdl2,

16.2.115 VMdlPrimConeVMdl2 (vmdl_prim.c:278)

```
VMdlVModelStruct *VMdlPrimConeVMdl2(const VMdlParamsStruct *Params,
                                    const CagdVType Center,
                                    CagdRType Radius,
                                    CagdRType Height,
                                    CagdBType Rational,
                                    CagdRType InternalCubeEdge)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

Center: Center of the cone.

Radius: Radius of the base.

Height: Height of the cone.

Rational: Indicates a rational or polynomial spline of the result.

InternalCubeEdge: Internal cube edge length. If zero, a single trivariate is constructed that is singular only on the boundary.

Returns: Constructed V-model.

Description: Constructs V-model of a cone, composed of five extruded V-cells, an inner cube, and four outer V-cells all glued together into one V-model.
See also: VMdlPrimConeVMdl,

16.2.116 VMdlPrimCubeSphereVMdl (vmdl_prim.c:156)

```
VMdlVModelStruct *VMdlPrimCubeSphereVMdl(const CagdVType Center,
                                         CagdRType Radius,
                                         CagdBType Rational,
                                         CagdRType InternalCubeEdge)
```

Center: Center of the sphere.

Radius: Radius of the sphere.

Rational: Indicates a rational or polynomial spline of the result.

InternalCubeEdge: Internal cube edge length. If zero, a single trivariate is constructed that is singular only on the boundary.

Returns: Constructed V-model.

Description: Constructs V-model of a sphere, composed of seven V-cells, a cube in inside and other six spherical non-singular V-cells.
See also: VMdlPrimCubeSphereVMdl2,

16.2.117 VMdlPrimCubeSphereVMdl2 (vmdl_prim.c:111)

```
VMdlVModelStruct *VMdlPrimCubeSphereVMdl2(const VMdlParamsStruct *Params,
                                          const CagdVType Center,
                                          CagdRType Radius,
                                          CagdBType Rational,
                                          CagdRType InternalCubeEdge)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

Center: Center of the sphere.

Radius: Radius of the sphere.

Rational: Indicates a rational or polynomial spline of the result.

InternalCubeEdge: Internal cube edge length. If zero, a single trivariate is constructed that is singular only on the boundary.

Returns: Constructed V-model.

Description: Constructs V-model of a sphere, composed of seven V-cells, a cube in inside and other six spherical non-singular V-cells.
See also: VMdlPrimCubeSphereVMdl,

16.2.118 VMdlPrimCylinderVMdl (vmdl_prim.c:501)

```
VMdlVModelStruct *VMdlPrimCylinderVMdl(const CagdVType Center,
                                       CagdRType Radius,
                                       CagdRType Height,
                                       CagdBType Rational,
                                       CagdRType InternalCubeEdge)
```

Center: Center of the cylinder.

Radius: Radius of the base.

Height: Height of the cylinder.

Rational: Indicates a rational or polynomial spline of the result.

InternalCubeEdge: Internal cube edge length. If zero, a single trivariate is constructed that is singular only on the boundary.

Returns: Constructed V-model.

Description: Constructs V-model of a cylinder, composed of five extruded V-cells, an inner cube, and four outer V-cells all glued together into one V-model.
See also: VMdlPrimCylinderVMdl2,

16.2.119 VMdlPrimCylinderVMdl2 (vmdl_prim.c:453)

```
VMdlVModelStruct *VMdlPrimCylinderVMdl2(const VMdlParamsStruct *Params,
                                        const CagdVType Center,
                                        CagdRType Radius,
                                        CagdRType Height,
                                        CagdBType Rational,
                                        CagdRType InternalCubeEdge)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

Center: Center of the cylinder.

Radius: Radius of the base.

Height: Height of the cylinder.

Rational: Indicates a rational or polynomial spline of the result.

InternalCubeEdge: Internal cube edge length. If zero, a single trivariate is constructed that is singular only on the boundary.

Returns: Constructed V-model.

Description: Constructs V-model of a cylinder, composed of five extruded V-cells, an inner cube, and four outer V-cells all glued together into one V-model.
See also: VMdlPrimCylinderVMdl,

16.2.120 VMdlPrimTorusVMdl (vmdl_prim.c:242)

```
VMdlVModelStruct *VMdlPrimTorusVMdl(const CagdVType Center,
                                    CagdRType MajorRadius,
                                    CagdRType MinorRadius,
                                    CagdBType Rational,
                                    CagdRType InternalCubeEdge)
```

Center: Center of the torus.

MajorRadius: Major radius.

MinorRadius: Minor radius.

Rational: Indicates a rational or polynomial spline of the result.

InternalCubeEdge: Internal cube edge length. If zero, a single trivariate is constructed that is singular only on the boundary.

Returns: Constructed V-model.

Description: Constructs V-model of a torus, composed of five V-cells of revolution, an inner cube, and four outer V-cells all glued together into one V-model. *
See also: VMdlPrimTorusVMdl2,

16.2.121 VMdlPrimTorusVMdl2 (vmdl_prim.c:193)

```
VMdlVModelStruct *VMdlPrimTorusVMdl2(const VMdlParamsStruct *Params,
                                     const CagdVType Center,
                                     CagdRType MajorRadius,
                                     CagdRType MinorRadius,
                                     CagdBType Rational,
                                     CagdRType InternalCubeEdge)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

Center: Center of the torus.

MajorRadius: Major radius.

MinorRadius: Minor radius.

Rational: Indicates a rational or polynomial spline of the result.

InternalCubeEdge: Internal cube edge length. If zero, a single trivariate is constructed that is singular only on the boundary.

Returns: Constructed V-model.

Description: Constructs V-model of a torus, composed of five V-cells of revolution, an inner cube, and four outer V-cells all glued together into one V-model. *
See also: VMdlPrimTorusVMdl,

16.2.122 VMdlRuledTrimSrf (vmdl_cnst.c:260)

```
VMdlVModelStruct *VMdlRuledTrimSrf(const TrimSrfStruct *TSrf1,
                                   const CagdSrfStruct *Srf2,
                                   int OtherOrder,
                                   int OtherLen)
```

TSrf1: Input trim surface.

Srf2: Input surface.

OtherOrder: Usually two, but one can specify higher orders in the ruled direction. OtherOrder must never be larger than OrderLen.

OtherLen: Usually two control points in the ruled direction which necessitates a linear interpolation.

Returns: The ruled V-model.

Description: Computes a ruled V-model from an input trim surface and a surface.

16.2.123 VMdlRuledTrimSrfExtra (vmdl_cnst.c:151)

```
VMdlVModelStruct *VMdlRuledTrimSrfExtra(const VMdlParamsStruct *Params,
                                        const TrimSrfStruct *TSrf1,
                                        const CagdSrfStruct *Srf2,
                                        int OtherOrder,
                                        int OtherLen,
                                        void *ExtraData)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

TSrf1: Input trim surface.

Srf2: Input surface.

OtherOrder: Usually two, but one can specify higher orders in the ruled direction. OtherOrder must never be larger than OrderLen.

OtherLen: Usually two control points in the ruled direction which necessitates a linear interpolation.

ExtraData: Extra data to pass to the stitching function.

Returns: The ruled V-model.

Description: Computes a ruled V-model from an input trim surface and a surface.

16.2.124 VMdlSetBoolOpCBFunc (vmdl_bool.c:3845)

```
VMdlVModelBoolOpCBFuncType VMdlSetBoolOpCBFunc(
                                        VMdlVModelBoolOpCBFuncType CBFunc)
```

CBFunc: The new (pointer to) function to use or NULl to disable.

Returns: The previous (pointer to) function.

Description: Set global variable that controls a call back function to be invoked at the end of each VModel Boolean operation with the two input VModel and the resulting VModel.

16.2.125 VMdlSetSplitPeriodicTV (vmdl_bool.c:3816)

```
CagdBType VMdlSetSplitPeriodicTV(CagdBType Split)
```

Split: TRUE iff Split periodic TV when converted to V-model.

Returns: The current value.

Description: Set global variable that controls splitting periodic trivariates into several non-periodic ones when converted to V-models.

16.2.126 VMdlSetStitchCallback (vmdl_aux.c:1111)

```
VMdlTSrfs2StitchedTSrfsFuncType VMdlSetStitchCallback(
                                        VMdlTSrfs2StitchedTSrfsFuncType Func)
```

Func: New error stitching callback function.

Returns: Old stitching function reference.

Description: Sets the callback function for stitching trimmed surfaces into the boundary of a V-element.

16.2.127 VMdlSlicerAddGeom (vmslicer.c:1235)

```
int VMdlSlicerAddGeom(const VMdlParamsStruct *Params,
                      VMdlSlicerInfoStruct *Info,
                      const IPObjectStruct *Geom,
                      int Priority)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

Info: Initialized already slicing information.

Geom: additional geometry to initialize for the slicing.

Priority: The slicing priority of the geometry. The higher the value wins - like closer to a viewer in a regular Z buffer.

Returns: TRUE if successful, FALSE otherwise.

Description: Adds an (optional) container to the model to be sliced. This container is assumed to hold the initialized model and serves as an aquarium holding the model. Indeed, the container will typically be transparent. This function can only be invoked after the geometry was initialized and before slicing has been commenced.
See also: VMdlSlicerInitTrivMdl, VMdlSlicerAddGeom, VMdlSlicerInitVModel, , VMdlSlicerInitVModelVElement, VMdlSlicerInitVElement,

16.2.128 VMdlSlicerAssignValueToDataXY (vmslicer.c:1765)

```
void VMdlSlicerAssignValueToDataXY(VMdlSlicerInfoStruct *Info,
                                   int x,
                                   int y,
                                   CagdRType *ValueBuf)
```

Info: The slicer info.

x, y: The coordinates in the images.

ValueBuf: The buffer into which the value will be assigned. Should be able to contain at least four CagdBType.

Returns: void

Description: Given a VMdlSlicerInfoStruct, assign the value of the X,Y coordinates in the slice image.
See also: VMdlSlicerSliceAtZLevel,

16.2.129 VMdlSlicerAssignValueToImgXY (vmslicer.c:1892)

```
void VMdlSlicerAssignValueToImgXY(VMdlSlicerInfoStruct *Info,
                                  int x,
                                  int y,
                                  VMdlSlicerOutputImageStruct *Img)
```

Info: The slicer info.

x, y: The coordinates in the images.

Img: The image struct, representing the slice.

Returns: void

Description: Given a VMdlSlicerInfoStruct, assign the value of the X,Y coordinates in the slice image.
See also: VMdlSlicerSliceAtZLevel,

16.2.130 VMdlSlicerDefaultParams (vmslicer.c:3409)

```
VMdlSlicerParamsStruct VMdlSlicerDefaultParams()
```

Returns: The default parameters for the slicer.

Description: Creates a params struct with default values for the slicer.
See also:

16.2.131 VMdlSlicerFree (vmslicer.c:1432)

```
void VMdlSlicerFree(VMdlSlicerInfoStruct *Info)
```

Info: The slicing info to free.

Returns: void.

Description: Free the given slicer info.
See also: VMdlSlicerInitTrivar, VMdlSlicerInitVElement, VMdlSlicerInitTrivMdl, , VMdlSlicerSliceAtZLevel, VMdlSlicerGetCurrSliceXY, , VMdlSlicerGeSliceSize, VMdlSlicerFree,

16.2.132 VMdlSlicerGetCurrSliceXY (vmslicer.c:3328)

```
int VMdlSlicerGetCurrSliceXY(VMdlSlicerInfoStruct *Info,
                             int x,
                             int y,
                             CagdRType *Params,
                             CagdRType *Pos,
                             const TrivTVStruct **TV)
```

Info: The slicer info.

x: X coordinate of slice pixel to get.

y: Y coordinate of slice pixel to get.

Params: The UVW parameter values for the pixel, in TV, if any.

Pos: The position (of the center) of the pixel.

TV: The trivariate that covered this position, if any. If TV is NULL, this parameter is ignored and not updated.

Returns: If in heterogeneous slicing mode (VMDL_SLICE_HETEROGENEOUS) returns TRUE if the pixel is not empty, FALSE otherwise. Otherwise, returns the TV index under this pixel if covered, or -IRIT_MAX_INT if not covered.

Description: Get the information for a single pixel in the slice.
See also: VMdlSlicerInitTrivar, VMdlSlicerInitVElement, VMdlSlicerInitTrivMdl, , VMdlSlicerSliceAtZLevel, VMdlSlicerFree, , VMdlSlicerGeSliceSize,

16.2.133 VMdlSlicerGetSliceSize (vmslicer.c:3388)

```
void VMdlSlicerGetSliceSize(VMdlSlicerInfoStruct *Info, int *Size)
```

Info: The slicer info.

Size: The size of the slices.

Returns: void

Description: Returns the size (in pixels) of all slices.
See also: VMdlSlicerInitTrivar, VMdlSlicerInitVElement, VMdlSlicerInitTrivMdl, , VMdlSlicerSliceAtZLevel, VMdlSlicerGetCurrSliceXY, , VMdlSlicerFree,

16.2.134 VMdlSlicerImageGetPixel (vmslicer.c:1980)

```
void VMdlSlicerImageGetPixel(const VMdlSlicerOutputImageStruct *Img,
                             int x,
                             int y,
                             void *PixelData)
```

Img: The image struct.

x, y: The coordinates in the images.

PixelData: The data to assign to the pixel.

Returns: void

Description: Get the value of the pixel at the x, y coordinates from a given image, and assign it to a given buffer. The buffer must be able to contain the pixel struct of the image.
See also: VMdlSlicerImageSetPixel, VMdlSlicerImageGetPixelPtr, , VMdlSlicerImageGetPixelConstPtr,

16.2.135 VMdlSlicerImageGetPixelConstPtr (vmslicer.c:2046)

```
const void *VMdlSlicerImageGetPixelConstPtr(
                                        const VMdlSlicerOutputImageStruct *Img,
                                        int x,
                                        int y)
```

Img: The image struct.

x, y: The coordinates in the images.

Returns: A const pointer to the pixel at the x, y coordinates of the image.

Description: Get a pointer to the pixel at the x, y coordinates from a given image. The returned pointer is const, so it cannot be used to assign data to the image.
See also: VMdlSlicerImageGetPixelPtr, , VMdlSlicerImageGetPixel, VMdlSlicerImageSetPixel,

16.2.136 VMdlSlicerImageGetPixelPtr (vmslicer.c:2011)

```
void *VMdlSlicerImageGetPixelPtr(VMdlSlicerOutputImageStruct *Img,
                                 int x,
                                 int y)
```

Img: The image struct.

x, y: The coordinates in the images.

Returns: A pointer to the pixel at the x, y coordinates of the image.

Description: Get a pointer to the pixel at the x, y coordinates from a given image. The returned pointer is not const, so it can be used to assign data to the image.
See also: VMdlSlicerImageGetPixelConstPtr, , VMdlSlicerImageGetPixel, VMdlSlicerImageSetPixel,

16.2.137 VMdlSlicerImageSetPixel (vmslicer.c:1947)

```
void VMdlSlicerImageSetPixel(VMdlSlicerOutputImageStruct *Img,
                             int x,
                             int y,
                             void *PixelData)
```

Img: The image struct.

x, y: The coordinates in the images.

PixelData: The data to assign to the pixel.

Returns: void

Description: Assign a value to a pixel at the x, y coordinates in a given image. The value must be the same data structure as the pixel type of the image.
See also: VMdlSlicerImageGetPixel, VMdlSlicerImageGetPixelPtr, , VMdlSlicerImageGetPixelConstPtr,

16.2.138 VMdlSlicerImplicitSliceAtZLevel (vmslicer.c:3889)

```
VMdlSlicerOutputImageStruct *VMdlSlicerImplicitSliceAtZLevel(
                                                VMdlSlicerInfoStruct *Info,
                                                CagdRType z)
```

Info: The slicer info.

z: Z coordinate of slice.

Returns: The image that results from slicing.

Description: Compute the current slice of microstructures with implicit tiles at the given Z height. A point is considered inside the implicit if its value is negative.
See also: VMdlSlicerImplicitTileInitInfo, VVMdlSlicerImplicitTileInfoFree,

16.2.139 VMdlSlicerImplicitTileInfoFree (vmslicer.c:3593)

```
void VMdlSlicerImplicitTileInfoFree(VMdlSlicerInfoStruct *SlicerInfo)
```

SlicerInfo: The implicit tiling info to free.

Returns: void

Description: Free the given struct which stores the implicit tiling info.
See also: MdlSlicerImplicitTileInitInfo, VMdlSlicerImplicitSliceAtZLevel,

16.2.140 VMdlSlicerImplicitTileInitInfo (vmslicer.c:3559)

```
int VMdlSlicerImplicitTileInitInfo(
                                   struct VMdlSlicerInfoStruct *SlicerInfo,
                                   const struct IPObjectStruct *ImplctMicroModel)
```

SlicerInfo: The data structure to store slicing information.

ImplctMicroModel: Object to extract parameters about regular implicit tile slicing. ImplctMicroModel is a list object containing (TVDeformationMap, ImplicitTileFunction, (optional) ColorTrivariate, TilingSteps). TilingSteps, XMin, YMin, XMax, YMax, ZRes, XRes, YRes),

Returns: 1 if succeeded to parse options, 0 otherwise.

Description: Prepare the auxiliary data to slice implicit tiles.
See also: VVMdlSlicerImplicitTileInfoFree, VMdlSlicerImplicitSliceAtZLevel,

16.2.141 VMdlSlicerInitModel (vmslicer.c:1123)

```
VMdlSlicerInfoStruct *VMdlSlicerInitModel(
                                          const MdlModelStruct *BMdl,
                                          const VMdlSlicerParamsStruct *Params)
```

BMdl: Model for the boundary of the volume.

Params: The parameters for slicing operations.

Returns: The handle to the slicer information.

Description: Init the parametric slicer info that can then be used to get the slices, using only a model for boundaries (no heterogeneity here).
See also: VMdlSlicerInitTrivar, VMdlSlicerInitVElement, VMdlSlicerInitVModel, , VMdlSlicerInitVModelVElement, VMdlSlicerFree, VMdlSlicerAddGeom, , VMdlSlicerSliceAtZLevel, VMdlSlicerGetCurrSliceXY, , VMdlSlicerInitTrivMdl, VMdlSlicerGeSliceSize, VMdlSlicerInitTrivPoly,

16.2.142 VMdlSlicerInitTrivMdl (vmslicer.c:991)

```
VMdlSlicerInfoStruct *VMdlSlicerInitTrivMdl(
                                          const TrivTVStruct *Trivars,
                                          const IPObjectStruct *BMdl,
                                          const VMdlSlicerParamsStruct *Params)
```

Trivars: The trivar(s).

BMdl: Model for the boundary of the volume.

Params: The parameters for slicing operations.

Returns: The handle to the slicer information.

Description: Init the parametric slicer info that can then be used to get the slices, used as trivariate(s) for the interior, and a model for boundaries.
See also: VMdlSlicerInitTrivar, VMdlSlicerInitVElement, VMdlSlicerInitVModel, , VMdlSlicerInitVModelVElement, VMdlSlicerFree, VMdlSlicerAddGeom, , VMdlSlicerSliceAtZLevel, VMdlSlicerGetCurrSliceXY, , VMdlSlicerInitTrivPoly, VMdlSlicerGeSliceSize,

16.2.143 VMdlSlicerInitTrivPoly (vmslicer.c:1058)

```
VMdlSlicerInfoStruct *VMdlSlicerInitTrivPoly(
                                        const TrivTVStruct *Trivars,
                                        const IPObjectStruct *BPls,
                                        const VMdlSlicerParamsStruct *Params)
```

Trivars: The trivar(s).

BPls: Polygonal model of the boundary of the volume.

Params: The parameters for slicing operations.

Returns: The handle to the slicer information.

Description: Init the parametric slicer info that can then be used to get the slices, used as trivariate(s) for the interior & a polygonal model for boundaries.
See also: VMdlSlicerInitTrivar, VMdlSlicerInitVElement, VMdlSlicerInitVModel, , VMdlSlicerInitVModelVElement, VMdlSlicerFree, VMdlSlicerAddGeom, , VMdlSlicerSliceAtZLevel, VMdlSlicerGetCurrSliceXY, , VMdlSlicerInitTrivPoly, VMdlSlicerGeSliceSize,

16.2.144 VMdlSlicerInitTrivar (vmslicer.c:927)

```
VMdlSlicerInfoStruct *VMdlSlicerInitTrivar(
                                        const TrivTVStruct *Trivars,
                                        const VMdlSlicerParamsStruct *Params)
```

Trivars: The trivars. One or more in a linked list.

Params: The parameters for slicing operations.

Returns: The handle to the slicer information.

Description: Init the parametric slicer info that can then be used to get the slices, using trivariate(s), with the boundary surfaces used for boundaries.
See also: VMdlSlicerInitVElement, VMdlSlicerInitTrivMdl, VMdlSlicerInitVModel, , VMdlSlicerInitVModelVElement, VMdlSlicerFree, VMdlSlicerAddGeom, , VMdlSlicerSliceAtZLevel, VMdlSlicerGetCurrSliceXY, , VMdlSlicerGeSliceSize,

16.2.145 VMdlSlicerInitVElement (vmslicer.c:693)

```
VMdlSlicerInfoStruct *VMdlSlicerInitVElement(
                                       const VMdlParamsStruct *Params,
                                       VMdlVolumeElementStruct *VolumeElement)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VolumeElement: The volume element.

Returns: The handle to the slicer information.

Description: Init the parametric slicer info that can then be used to get the slices, using a VCell.
See also: VMdlSlicerInitTrivar, VMdlSlicerInitTrivMdl, VMdlSlicerInitVModel, , VMdlSlicerInitVModelVElement, VMdlSlicerFree, VMdlSlicerAddGeom, , VMdlSlicerSliceAtZLevel, VMdlSlicerGetCurrSliceXY, , VMdlSlicerGeSliceSize,

16.2.146 VMdlSlicerInitVModel (vmslicer.c:845)

```
VMdlSlicerInfoStruct *VMdlSlicerInitVModel(
                                        const VMdlParamsStruct *Params,
                                        const VMdlVModelStruct *VModel)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VModel: The volume model.

Returns: The handle to the slicer information.

Description: Init the parametric slicer info that can then be used to get the slices, using all VCells in a VModel, simultaneously.
See also: VMdlSlicerInitTrivar, VMdlSlicerInitTrivMdl, VMdlSlicerInitVElement, , VMdlSlicerInitVModelVElement, VMdlSlicerFree, VMdlSlicerAddGeom, , VMdlSlicerSliceAtZLevel, VMdlSlicerGetCurrSliceXY, , VMdlSlicerGeSliceSize,

16.2.147 VMdlSlicerInitVModelVElement (vmslicer.c:741)

```
VMdlSlicerInfoStruct *VMdlSlicerInitVModelVElement(
                                        const VMdlParamsStruct *Params,
                                        VMdlSlicerInfoStruct *RetIn,
                                        const VMdlVModelStruct *VModel,
                                        VMdlVolumeElementStruct *VolumeElement)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

RetIn: If provided, updates this info, instead of creating one.

VModel: The volume model.

VolumeElement: The volume element. NULL for first global initialization.

Returns: The handle to the slicer information.

Description: Init the parametric slicer info that can then be used to get the slices, using all VCells in a VModel, one VCell at a time.
See also: VMdlSlicerInitTrivar, VMdlSlicerInitTrivMdl, VMdlSlicerInitVElement, , VMdlSlicerInitVModel, MVMdlSlicerFree, VMdlSlicerAddGeom, , VMdlSlicerSliceAtZLevel, VMdlSlicerGetCurrSliceXY, , VMdlSlicerGeSliceSize, ,

16.2.148 VMdlSlicerOutputImageFree (vmslicer.c:3530)

```
void VMdlSlicerOutputImageFree(VMdlSlicerOutputImageStruct *Item)
```

Item: Pointer to the structure that needs to be freed.

Returns: void

Description: Given a VMdlSlicerOutputImageStruct, it frees its memory.
See also: VMdlSlicerOutputImageStruct,

16.2.149 VMdlSlicerOutputImageFreeArr (vmdl_blnd_field.c:2931)

```
void VMdlSlicerOutputImageFreeArr(VMdlSlicerOutputImageStruct *Items,
                                  const int NumItems)
```

Items: Pointer to the array that needs to be freed.

NumItems: Number of items in the array.

Returns: void

Description: Given an array of VMdlSlicerOutputImageStruct, frees the memory of its contained images, as well as the array itself.

16.2.150 VMdlSlicerSaveImage (vmslicer.c:3444)

```
void VMdlSlicerSaveImage(VMdlSlicerOutputImageStruct *Image,
                         const char *Output)
```

Image: Matrix storing the RGB values at each point.

Output: File name for saving the image.

Returns: void

Description: Given a VMdlSlicerOutputImageStruct, a file name for the output and the resolution of the image, it prints the image at the address.
See also: VMdlSlicerOutputImageStruct,

16.2.151 VMdlSlicerSliceAtZBatch (vmslicer.c:2092)

```
VMdlSlicerOutputImageStruct *VMdlSlicerSliceAtZBatch(
                                                const VMdlParamsStruct *Params,
                                                VMdlSliceOpType SliceOper,
                                                VMdlSlicerInfoStruct *Info,
                                                CagdRType *ZVals,
                                                int BatchSz)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

SliceOper: 0 for full (heterogeneous) slicing as an image. This mode is not supported by this function. 1 for outline of slice as curves (Image name is ignored). 2 for outline of slice as an image. 3 for a cover set as linear curves. 4 for all covered pixels in the slice as an image.

Info: The slicer info.

ZVals: Z coordinate of slices.

BatchSz: The number of Z-values in the batch.

Returns: The images that result from the slicing (array of size BatchSz).

Description: Compute the slices at the given batch of Z heights. This function will use parallel computation, if enabled.

See also: VMdlSlicerInitTrivar, VMdlSlicerInitVElement, VMdlSlicerInitTrivMdl, , VMdlSlicerFree, VMdlSlicerGetCurrSliceXY, VMdlSlicerGeSliceSize, , VMdlSlicerSliceAtZLevel, VMdlSlicerSliceAtZBatchToFiles,

16.2.152 VMdlSlicerSliceAtZBatch2 (vmslicer.c:2188)

```
VMdlSlicerOutputImageStruct *VMdlSlicerSliceAtZBatch2(
                                                const VMdlParamsStruct *Params,
                                                VMdlSliceOpType SliceOper,
                                                VMdlSlicerInfoStruct *Info,
                                                CagdRType ZMin,
                                                CagdRType ZMax,
                                                CagdRType ZStep)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

SliceOper: 0 for full (heterogeneous) slicing as an image. This mode is not supported by this function. 1 for outline of slice as curves (Image name is ignored). 2 for outline of slice as an image. 3 for a cover set as linear curves. 4 for all covered pixels in the slice as an image.

Info: The slicer info.

ZMin, ZMax, ZStep: The Z coordinate to iterate along, of slices.

Returns: The images that result from the slicing.This array is allocated dynamically and terminates with a zero size image: Imgs[Last].Resolution[0] = Imgs[Last].Resolution[1] = 0.

Description: Compute the slices at the given batch of Z heights. This function will use parallel computation, if enabled.

See also: VMdlSlicerInitTrivar, VMdlSlicerInitVElement, VMdlSlicerInitTrivMdl, , VMdlSlicerFree, VMdlSlicerGetCurrSliceXY, VMdlSlicerGeSliceSize, , VMdlSlicerSliceAtZLevel, VMdlSlicerSliceAtZBatch, , VMdlSlicerSliceAtZBatchToFiles,

16.2.153 VMdlSlicerSliceAtZBatchToFiles (vmslicer.c:2272)

```
void VMdlSlicerSliceAtZBatchToFiles(const VMdlParamsStruct *Params,
                                    VMdlSliceOpType SliceOper,
                                    VMdlSlicerInfoStruct *Info,
                                    CagdRType *ZVals,
                                    int BatchSz,
                                    const char *FileNameBase)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

SliceOper: 0 for full (heterogeneous) slicing as an image. This mode is not supported by this function. 1 for outline of slice as curves (Image name is ignored). 2 for outline of slice as an image. 3 for a cover set as linear curves. 4 for all covered pixels in the slice as an image. 5 for slicing implicit microstructures.

Info: The slicer info.

ZVals: Z coordinate of slices.

BatchSz: The number of Z-values in the batch.

FileNameBase: The base for the image file names.

Returns: void

Description: Compute the slices at the given batch of Z heights, and save the images to files. The file names follow a pattern based on FileNameBase, with a number inserted before the extension. This function will use parallel computation for slicing, if enabled, but not for saving the files.
See also: VMdlSlicerInitTrivar, VMdlSlicerInitVElement, VMdlSlicerInitTrivMdl, , VMdlSlicerFree, VMdlSlicerGetCurrSliceXY, VMdlSlicerGeSliceSize, , VMdlSlicerSliceAtZLevel, VMdlSlicerSliceAtZBatch, , VMdlSlicerSliceAtZBatchToFiles2,

16.2.154 VMdlSlicerSliceAtZBatchToFiles2 (vmslicer.c:2391)

```
void VMdlSlicerSliceAtZBatchToFiles2(const VMdlParamsStruct *Params,
                                     VMdlSliceOpType SliceOper,
                                     VMdlSlicerInfoStruct *Info,
                                     CagdRType ZMin,
                                     CagdRType ZMax,
                                     CagdRType ZStep,
                                     const char *FileNameBase)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

SliceOper: 0 for full (heterogeneous) slicing as an image. This mode is not supported by this function. 1 for outline of slice as curves (Image name is ignored). 2 for outline of slice as an image. 3 for a cover set as linear curves. 4 for all covered pixels in the slice as an image. 5 for slicing implicit microstructures.

Info: The slicer info.

ZMin, ZMax, ZStep: The Z coordinate to iterate along, of slices.

FileNameBase: The base for the image file names.

Returns: void

Description: Compute the slices at the given batch of Z heights, and save the images to files. The file names follow a pattern based on FileNameBase, with a number inserted before the extension. This function will use parallel computation for slicing, if enabled, but not for saving the files.
See also: VMdlSlicerInitTrivar, VMdlSlicerInitVElement, VMdlSlicerInitTrivMdl, , VMdlSlicerFree, VMdlSlicerGetCurrSliceXY, VMdlSlicerGeSliceSize, , VMdlSlicerSliceAtZLevel, VMdlSlicerSliceAtZBatch, , VMdlSlicerSliceAtZBatchToFiles,

16.2.155 VMdlSlicerSliceAtZLevel (vmslicer.c:1492)

```
void VMdlSlicerSliceAtZLevel(const VMdlParamsStruct *Params,
                             VMdlSlicerInfoStruct *Info,
                             CagdRType z,
                             int BatchSize)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

Info: The slicer info.

z: Z coordinate of slice.

BatchSize: The size of the batch, in the case of single-slice parallel slicing. If BatchSize <= 0, the slice is computed in a single thread. If multiple slices are being computed in parallel, then BatchSize should be 0 (single-slice parallel slicing should not be used).

Returns: void

Description: Compute the current slice at the given Z height.
See also: VMdlSlicerInitTrivar, VMdlSlicerInitVElement, VMdlSlicerInitTrivMdl, , VMdlSlicerFree, VMdlSlicerGetCurrSliceXY, VMdlSlicerGeSliceSize, , VMdlSlicerSliceAtZLevel2,

16.2.156 VMdlSlicerSliceAtZLevelCoverage (vmslicer.c:2482)

```
IPObjectStruct *VMdlSlicerSliceAtZLevelCoverage(const VMdlParamsStruct *Params,
                                                VMdlSliceOpType SliceOper,
                                                VMdlSlicerInfoStruct *Info,
                                                CagdRType z)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

SliceOper: 0 for full (heterogeneous) slicing as an image. This mode is not supported by this function. 1 for outline of slice as curves (Image name is ignored). 2 for outline of slice as an image. 3 for a cover set as linear curves. 4 for all covered pixels in the slice as an image.

Info: The slicer info.

z: Z coordinate of slice.

Returns: Sliced geometry or NULL if an image is computed.

Description: Compute the current outline or covering pixels of the slice at the given Z height.
See also: VMdlSlicerInitTrivar, VMdlSlicerInitVElement, VMdlSlicerInitTrivMdl, , VMdlSlicerFree, VMdlSlicerGetCurrSliceXY, VMdlSlicerGeSliceSize, , VMdlSlicerSliceAtZLevel,

16.2.157 VMdlSplitVModelInDir (vmdl_aux.c:360)

```
void VMdlSplitVModelInDir(const VMdlParamsStruct *Params,
                          VMdlVModelStruct *VM,
                          CagdRType Dx,
                          CagdRType Dy,
                          CagdRType Dz)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VM: V-model.

Dx, Dy, Dz: Explosion direction (Dx,Dy,Dz).

Returns: void

Description: Explodes a given V-model in a given direction relative to its center, in place.

16.2.158 VMdlStitchMdlModel (vmdl_bool.c:376)

```
int VMdlStitchMdlModel(const VMdlParamsStruct *Params, MdlModelStruct *Mdl)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

Mdl: Model to seek trimming curves to stitch together, in place.

Returns: Number of trimming curves stitched together, in place.

Description: Scans the given model for trimming curves that could be stitched together. A pair of trimming curves can be stitched together if they have the same Euclidean representation and they now have no neighbors. Two trimming curves are considered with "same Euclidean representation" if their end points are the same upto given tolerance (should probably do somewhat better here).
See also: MdlStitchModel,

16.2.159 VMdlSubdivDefaultParams (vmdl_subdv.c:3927)

```
VMdlSubdivParamsStruct VMdlSubdivDefaultParams()
```

Returns: The default parameters for subdivision operations.

Description: Creates a params struct with default values for subdivision operations.
See also:

16.2.160 VMdlSubdivideVElemToBezierVElements (vmdl_subdv.c:2112)

```
VMdlVModelStruct *VMdlSubdivideVElemToBezierVElements(
                                        const VMdlParamsStruct *Params,
                                        const VMdlVolumeElementStruct *VElem,
                                        const TrivTVStruct *InpTV)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VElem: Input VElement.

InpTV: One of the trivariates enclosing VElem (If NULL, the first TV will be used).

Returns: List of VModels, each contains one VElement with trimmed Bezier trivariate, and associated with the appropriate Bezier portion of TV, as well as the a copy of the other original trivariates associated with the given V-element (VElem).

Description: Subdivide a given VElement into (trimmed) Bezier VModels by dividing at all internal knots of a given trivariate TV. The result if a list of VModels each containing one V-element.

16.2.161 VMdlSubdivideVElement (vmdl_subdv.c:1661)

```
VMdlVModelStruct *VMdlSubdivideVElement(const VMdlParamsStruct *Params,
                                        VMdlVolumeElementStruct *VElem,
                                        const TrivTVStruct *TV,
                                        TrivTVDirType Dir,
                                        IrtRType t,
                                        IrtRType *OtherParamAttribVals,
                                        CagdBType HandleKnotLineIntersections)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VElem: Input VElement.

TV: One of the trivariates enclosing VElem (If NULL, the first TV will be used.

Dir: Direction of the subdivision.

t: Parametric value in the domain of TV to subdivide at.

OtherParamAttribVals: If not null, passed as UV attributes to the subdivision iso-surface.

HandleKnotLineIntersections: If the subdivision can be at knot lines, A different Srf-Srf intersection is used.

Returns: List of two VModels VElements after subdivision.

Description: Subdivide a given VElement at a parametric value of one of its TVs.

16.2.162 VMdlSubdivideVMdlToBezierVElements (vmdl_subdv.c:2227)

```
VMdlVModelStruct *VMdlSubdivideVMdlToBezierVElements(
                                        const VMdlParamsStruct *Params,
                                        const VMdlVModelStruct *VMdl)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VMdl: Input V-model.

Returns: List of VModels, each contains one VElement with trimmed Bezier trivariate, and associated with the appropriate Bezier portion of TV, as well as the a copy of the other original trivariates associated with the given V-element (VElem).

Description: Subdivide a given V-model, assuming it contains only one V-element, into (trimmed) Bezier VModels by dividing at all internal knots of a given trivariate TV. The result is a list of VModels each containing one V-element.

16.2.163 VMdlSubdivideVModel (vmdl_subdv.c:2012)

```
VMdlVModelStruct *VMdlSubdivideVModel(const VMdlParamsStruct *Params,
                                      VMdlVModelStruct *VMdl,
                                      TrivTVDirType Dir,
                                      IrtRType t)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VMdl: V-model to subdivide.

Dir: Parametric direction of subdivision (of the TV in the V-element).

t: Parametric value in the domain of TV to subdivide at.

Returns: List of VModels VElements after subdivision.

Description: Subdivide a given V-model, assuming having only one V-element, at a parametric value of its first trivariate.
See also:

16.2.164 VMdlSweepDefaultParams (vmdl_cnst.c:847)

```
VMdlSweepParamsStruct VMdlSweepDefaultParams()
```

Returns: The default parameters for boolean operations.

Description: Creates a params struct with default values for boolean operations.
See also:

16.2.165 VMdlSweepTrimSrf (vmdl_cnst.c:689)

sweep

trivariate constructors

```
VMdlVModelStruct *VMdlSweepTrimSrf(const TrimSrfStruct *TSrf,
                                   const CagdCrvStruct *Axis,
                                   const CagdCrvStruct *ScalingCrv,
                                   CagdRType Scale,
                                   const VoidPtr Frame,
                                   int FrameOption)
```

TSrf: Input trim surface.

Axis: Of the constructed sweep trivariate.

ScalingCrv: Optional scale or profile curve. NULL if none. The Y axis of this curve is using as scaling as the function of the parametrization. X axis can be used to define a 2D parametric curve so this curve can be displayed but X is ignored by this function.

Scale: If no Scaling Crv, Scale is used to apply a fixed scale on the CrossSection curve.

Frame: An optional vector or a curve to specify the binormal orientation. Otherwise, Frame must be NULL.

FrameOption: If 2 Frame is a curve, if 0 or 1 a vector (if Frame is not NULL). If 0, the vector is used just to init the first frame.

Returns: Constructed sweep trimmed trivariate.

Description: Computes a trimmed sweep V-model from an input trim surface and sweep information.
See also: CagdSweepSrf, CagdSweepAxisRefine, TrivSweepTV,

16.2.166 VMdlSweepTrimSrfExtra (vmdl_cnst.c:735)

sweep

trivariate constructors

```
VMdlVModelStruct *VMdlSweepTrimSrfExtra(const VMdlParamsStruct *Params,
                                        const TrimSrfStruct *TSrf,
                                        const CagdCrvStruct *Axis,
                                        const CagdCrvStruct *ScalingCrv,
                                        CagdRType Scale,
                                        const VoidPtr Frame,
                                        int FrameOption,
                                        void* ExtraData)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

TSrf: Input trim surface.

Axis: Of the constructed sweep trivariate.

ScalingCrv: Optional scale or profile curve. NULL if none. The Y axis of this curve is using as scaling as the function of the parametrization. X axis can be used to define a 2D parametric curve so this curve can be displayed but X is ignored by this function.

Scale: If no Scaling Crv, Scale is used to apply a fixed scale on the CrossSection curve.

Frame: An optional vector or a curve to specify the binormal orientation. Otherwise, Frame must be NULL.

FrameOption: If 2 Frame is a curve, if 0 or 1 a vector (if Frame is not NULL). If 0, the vector is used just to init the first frame.

ExtraData: Extra data to pass to the stitching function.

Returns: Constructed sweep trimmed trivariate.

Description: Computes a trimmed sweep V-model from an input trim surface and sweep information.
See also: CagdSweepSrf, CagdSweepAxisRefine, TrivSweepTV, VMdlSweepTrimSrf,

16.2.167 VMdlTestMdlProps (vmdl_dbg.c:194)

```
void VMdlTestMdlProps(MdlModelStruct *Mdl)
```

Mdl: Model object.

Returns: void

Description: Tests Model object topological properties.

16.2.168 VMdlUntrimVModel (vmdl_subdv.c:3186)

```
TrivTVStruct *VMdlUntrimVModel(const VMdlParamsStruct *Params,
                               const VMdlVModelStruct *VMdl,
                               const TrivTVStruct *TV,
                               const TrivTVStruct *OriginalTV,
                               CagdBType InParamSpace,
                               int InvSrfApproxOrder,
                               CagdRType InvApproxErr)
```

Params: The general parameters of the module.

VMdl: V-model subdivision cache.

TV: The Bezier trivariate containing the V-model.

OriginalTV: The containing trivariate. Tthis will be identical to TV at first call in case that VElem is originally from a Bezier trivariate. The original trivariate could be a B-spline in in case VElem is result of a subdivision to bezier elements.

InParamSpace: If the untrimming will be performed in the parametric space (of OriginalTV).

InvSrfApproxOrder: The surfaces approximation order when inverting to the parametric space or OriginalTV.

InvApproxErr: The desired approximation error in the inversion process.

Returns: List of trivariates tiling the V-model.

Description: Untrims a given V-model into a list of tensor product trivariates. VMdl is assumed to be contained in a Bezier trivariate. And assumed to have only one V-element.

16.2.169 VMdlVModelBBox (vmdl_bool.c:3337)

```
CagdBBoxStruct *VMdlVModelBBox(const VMdlParamsStruct *Params,
                               const VMdlVModelStruct *VMdl,
                               CagdBBoxStruct *VMdlBBox,
                               int OnlyGeom)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VMdl: The V-model.

VMdlBBox: Output parameter holding the result BBox.

OnlyGeom: TRUE for only geometry bbox, FALSE if higher dimensions to be considered as well.

Returns: VMdlBBox.

Description: Computes the bounding box of a V-model.

16.2.170 VMdlVModelBoolOp (vmdl_bool.c:2800)

```
VMdlVModelStruct *VMdlVModelBoolOp(const VMdlParamsStruct *Params,
                                   const VMdlVModelStruct *VMdlA,
                                   const VMdlVModelStruct *VMdlB,
                                   VMdlBoolOpType OpType)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VMdlA: First V-model.

VMdlB: Second V-model.

OpType: Boolean operation type.

Returns: Boolean operation V-model result.

Description: Performs Boolean operation between two V-models.

16.2.171 VMdlVModelCopy (vmdl_gen.c:595)

```
VMdlVModelStruct *VMdlVModelCopy(const VMdlParamsStruct *Params,
                                 const VMdlVModelStruct *VMdl)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VMdl: A V-model to copy.

Returns: New copy of the given V-model.

Description: Copies a V-model.

16.2.172 VMdlVModelCopyList (vmdl_aux.c:102)

```
VMdlVModelStruct *VMdlVModelCopyList(const VMdlParamsStruct *Params,
                                     const VMdlVModelStruct *VMdls)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VMdls: List of V-models to copy.

Returns: List of copied V-models, or NULL in case of failure.

Description: Copies a list of V-models.

16.2.173 VMdlVModelFree (vmdl_gen.c:322)

```
void VMdlVModelFree(VMdlVModelStruct *Mdl)
```

Mdl: V-model to deallocate.

Returns: void

Description: Frees a V-model list.

16.2.174 VMdlVModelFreeList (vmdl_aux.c:81)

```
void VMdlVModelFreeList(VMdlVModelStruct *VMdls)
```

VMdls: List of V-models.

Returns: void

Description: Frees a list of V-models.

16.2.175 VMdlVModelFromVElement (vmdl_gen.c:1027)

```
VMdlVModelStruct *VMdlVModelFromVElement(const VMdlVolumeElementStruct *VElem,
                                         CagdBType UseVElemInPlace)
```

VElem: The V-element in the V-model.

UseVElemInPlace: If True, the VElem will be used in place, otherwise a new (deep) copy of VElem will be created.

Returns: New V-model with VElem (or a copy of VElem) as its V-element.

Description: Creates a new V-model having one (given) V-Element.

16.2.176 VMdlVModelIntersect (vmdl_bool.c:2752)

```
VMdlVModelStruct *VMdlVModelIntersect(const VMdlParamsStruct *Params,
                                      const VMdlVModelStruct *VMdlA,
                                      const VMdlVModelStruct *VMdlB)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VMdlA: First V-model.

VMdlB: Second V-model.

Returns: Boolean operation V-model result.

Description: Computes the intersection between two v-models.

16.2.177 VMdlVModelListBBox (vmdl_aux.c:51)

```
CagdBBoxStruct *VMdlVModelListBBox(const VMdlParamsStruct *Params,
                                   const VMdlVModelStruct *Mdls,
                                   CagdBBoxStruct *CagdBbox,
                                   int OnlyGeom)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

Mdls: List of V-models.

CagdBbox: Output parameter that will hold the bounding box.

OnlyGeom: TRUE for only geometry bbox, FALSE if higher dimensions to be considered as well.

Returns: CagdBbox.

Description: Computes axis aligned bounding box of list of V-models.

16.2.178 VMdlVModelNegate (vmdl_bool.c:2649)

```
VMdlVModelStruct *VMdlVModelNegate(const VMdlParamsStruct *Params,
                                   const VMdlVModelStruct *VMdl)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VMdl: The V-model.

Returns: Negated V-model.

Description: Negates the orientation of a given V-model.

16.2.179 VMdlVModelReplaceTV (vmdl_aux.c:446)

```
CagdBType VMdlVModelReplaceTV(VMdlVModelStruct *VMdl,
                              TrivTVStruct *OldTV,
                              const TrivTVStruct *NewTV)
```

VMdl: The V-model to manipulate.

OldTV: The trivariate to replace. If NULL, the first trivariate of VMdl will be used.

NewTV: The new trivariate.

Returns: TRUE iff the given trivariate is used in the given V-model, FALSE if error.

Description: Replaces a given trivariate that is part of the V-model structure with a new one (can be used for refinement for example). Note: the new trivariate is copied, and the old one is deallocated.

16.2.180 VMdlVModelSubtract (vmdl_bool.c:2729)

```
VMdlVModelStruct *VMdlVModelSubtract(const VMdlParamsStruct *Params,
                                     const VMdlVModelStruct *VMdlA,
                                     const VMdlVModelStruct *VMdlB)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VMdlA: First V-model.

VMdlB: Second V-model.

Returns: Boolean operation V-model result (VMdlA - VMdlB).

Description: Subtraction operation between two V-models.

16.2.181 VMdlVModelSymDiff (vmdl_bool.c:2703)

```
VMdlVModelStruct *VMdlVModelSymDiff(const VMdlParamsStruct *Params,
                                    const VMdlVModelStruct *VMdlA,
                                    const VMdlVModelStruct *VMdlB)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VMdlA: First V-model.

VMdlB: Second V-model.

Returns: Boolean operation V-model result.

Description: Symmetric difference operation between two V-models.

16.2.182 VMdlVModelTransform (vmdl_bool.c:3433)

```
void VMdlVModelTransform(const VMdlParamsStruct *Params,
                         VMdlVModelStruct *VMdl,
                         IrtHmgnMatType Mat)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VMdl: The V-model.

Mat: Transformation matrix.

Returns: void

Description: Transforms a V-model by a transformation matrix.

16.2.183 VMdlVModelUnion (vmdl_bool.c:2775)

```
VMdlVModelStruct *VMdlVModelUnion(const VMdlParamsStruct *Params,
                                  const VMdlVModelStruct *VMdlA,
                                  const VMdlVModelStruct *VMdlB)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VMdlA: First V-model.

VMdlB: Second V-model.

Returns: Boolean operation V-model result.

Description: Computes the intersection between two v-models.

16.2.184 VMdlVModelsSame (vmdl_aux.c:1057)

```
CagdBType VMdlVModelsSame(const VMdlParamsStruct *Params,
                          const VMdlVModelStruct *VMdl1,
                          const VMdlVModelStruct *VMdl2)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

VMdl1, VMdl2: The two VModel to compare.

Returns: TRUE if VModels are the same, FALSE otherwise.

Description: Compare the two VMOdel for similarity.
See also: CagdSrfsSame, CagdCrvsSame, MvarMVsSame, TrivTVSame.,

16.2.185 VMdlVolElemenTransform (vmdl_bool.c:3404)

```
void VMdlVolElemenTransform(const VMdlParamsStruct *Params,
                            VMdlVolumeElementStruct *Element,
                            CagdRType *Trans, CagdRType Scale)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

Element: The V-cell.

Trans: Translation vector.

Scale: Uniform scale amount.

Returns: void

Description: Transforms a V-cell by a translation and scale.

16.2.186 VMdlVolElementBBox (vmdl_bool.c:3277)

```
CagdBBoxStruct *VMdlVolElementBBox(const VMdlParamsStruct *Params,
                                   const VMdlVolumeElementStruct *Element,
                                   CagdBBoxStruct *VEBBox,
                                   int OnlyGeom)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

Element: The V-cell.

VEBBox: Output parameter holding the result BBox.

OnlyGeom: TRUE for only geometry bbox, FALSE if higher dimensions to be considered as well.

Returns: VEBBox.

Description: Computes the bounding box of a V-cell.

16.2.187 VMdlVolElementFromBoundaryModel (vmdl_bool.c:2332)

```
VMdlVModelStruct *VMdlVolElementFromBoundaryModel(
                                const VMdlParamsStruct *Params,
                                const MdlModelStruct *InBMdl,
                                VMdlInterTrivTVRefStruct *ElementTVsRefList)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

InBMdl: Model object of the V-cell boundary surfaces.

ElementTVsRefList: Reference list of trivariates.

Returns: The generated V-model.

Description: Constructs a V-model containing one V-cell having a given boundary and list of associated trivariates.

16.2.188 VMdlVolElementMatTransform (vmdl_bool.c:3374)

```
void VMdlVolElementMatTransform(const VMdlParamsStruct *Params,
                                VMdlVolumeElementStruct *Element,
                                IrtHmgnMatType TransMat)
```

Params: The common parameters to VModel operations. If NULL, default values are used.

Element: The V-cell.

TransMat: Transformation matrix.

Returns: void

Description: Transforms a V-cell by a transformation matrix.

16.2.189 VMdlVolumeElementCopy (vmdl_gen.c:628)

```
VMdlVolumeElementStruct *VMdlVolumeElementCopy(const VMdlVolumeElementStruct
                                                        *VolumeElement)
```

VolumeElement: A V-cell to copy.

Returns: New copy of the given V-cell.

Description: Copies a V-cell.

16.2.190 VMdlVolumeElementDeepCopy (vmdl_gen.c:720)

```
VMdlVolumeElementStruct *VMdlVolumeElementDeepCopy(
                                        const VMdlVolumeElementStruct *VElem)
```

VElem: A V-cell to copy.

Returns: New copy of the given V-cell.

Description: Copies a V-cell (Deep copy - copies all related trivariates, V-surfaces, V-curves, V-points).

16.2.191 VMdlVxlAllocVoxelVModel (vmvoxels.c:1639)

```
VMdlVoxelVModelStruct *VMdlVxlAllocVoxelVModel(const int *Sizes,
                                               TrivImagePixelType VoxelType)
```

Sizes: Sizes (in pixels) of Voxel model to create, in X, Y, Z.

VoxelType: Pixel type of in the images - byte, RGBA of float, etc.

Returns: New allocated voxel model.

Description: Allocates the memory required for a new voxel model.
See also: VMdlVxlMdl2VoxelVModel, VMdlVxlMdl2VoxelVModel2, VMdlVxlCrv2VoxelVModel, , VMdlVxlCopyVoxelVModel, VMdlVxlPrintVoxelVModel,

16.2.192 VMdlVxlAreaVxlVMdl (vmvoxels.c:1131)

```
CagdRType VMdlVxlAreaVxlVMdl(const VMdlVoxelVModelStruct *VxlVMdl,
                             const CagdRType IsoLevel)
```

VxlVMdl: A voxel vmodel to compute surface area at IsoLevel.

IsoLevel: The iso level at which to estimate the surface area.

Returns: The surface area.

Description: Given a Voxel vmodel compute its surface area, at the given isolevel. Computes marching cubes over the geometry and then compute the area.
See also: VMdlVxlMdl2VoxelVModel, VMdlVxlMdl2VoxelVModel2, VMdlVxlCrv2VoxelVModel, , VMdlVxlFreeVoxelVModel, VMdlVxlCopyVoxelVModel, , VMdlVxlGetRealVoxelVolume, VMdlVxlPrintVoxelVModel,

16.2.193 VMdlVxlCalculateAverage (vmvoxels.c:512)

```
void VMdlVxlCalculateAverage(const VMdlVoxelVModelStruct *VxlVMdl,
                             int Dim,
                             int i,
                             int j,
                             int k,
                             TrivImagePixelTypeUnion *Avg)
```

VxlVMdl: A voxel vmodel to calculate the average voxels value. Dim x Dim x Dim surrounding.

Dim: Matrix size to use (Dim x Dim x Dim).

i, j, k: Indices of voxel in the volume.

Avg: Output parameter, Average value of the Dim x Dim x Dim matrix.

Returns: void

Description: Given a voxel in a voxel model, compute the average value of the surrounding voxels.
See also: VMdlVxlMdl2VoxelVModel, VMdlVxlMdl2VoxelVModel2, VMdlVxlCrv2VoxelVModel, , VMdlVxlFreeVoxelVModel, VMdlVxlCopyVoxelVModel, VMdlVxlVoxelOffsetBW, , VMdlVxlVoxelOffsetGray, VMdlVxlInPixelOnBndry, VMdlVxlPrintVoxelVModel,

16.2.194 VMdlVxlCopyVoxelVModel (vmvoxels.c:1571)

```
VMdlVoxelVModelStruct *VMdlVxlCopyVoxelVModel(const VMdlVoxelVModelStruct
                                                                    *VxlVMdl)
```

VxlVMdl: Voxel vmodel to copy.

Returns: Copied Voxel model.

Description: Copy a voxel vmodel.
See also: VMdlVxlMdl2VoxelVModel, VMdlVxlMdl2VoxelVModel2, VMdlVxlCrv2VoxelVModel, , VMdlVxlFreeVoxelVModel, VMdlVxlGetRealVoxelVolume, , VMdlVxlPrintVoxelVModel,

16.2.195 VMdlVxlCrv2VoxelVModel (vmvoxels.c:801)

```
VMdlVoxelVModelStruct *VMdlVxlCrv2VoxelVModel(const IPObjectStruct *Crv,
                                              CagdRType Tol,
                                              const int *Sizes,
                                              TrivImagePixelType PixelType)
```

Crv: The curve(s) to voxelize. Assumed between [0, Dims[i]], in all axes - i in 0, 1, 2.

Tol: Tolerance to approximate curve(s) in Crv to polyline(s).

Sizes: Sizes (in pixels) of Voxel model to create, in X, Y, Z.

PixelType: The type of image to create and return as the slice. Can be TRIV_IMAGE_UNDEFINED_TYPE to use the default type.

Returns: Built Voxel vmodel, or NULL if error.

Description: Builds a Voxel model from given curve(s) and desired resolution.
See also: VMdlVxlFreeVoxelVModel, VMdlVxlCopyVoxelVModel, , VMdlVxlGetRealVoxelVolume, VMdlVxlMdl2VoxelVMod
, VMdlVxlEdge2VoxelVModel, VMdlVxlPrintVoxelVModel, ,

16.2.196 VMdlVxlEdge2VoxelVModel (vmvoxels.c:698)

```
void VMdlVxlEdge2VoxelVModel(VMdlVoxelVModelStruct *VxlVMdl,
                             const CagdRType *Pt1,
                             const CagdRType *Pt2)
```

VxlVMdl: The Voxels VMDL struct to scan convert the edges into.

Pt1, Pt2: The two end points of the edge, in VxlVMdl dimensions.

Returns: void

Description: Scan converts the edge (Pt1, Pt2) into VxlVMdl struct.
See also: VMdlVxlFreeVoxelVModel, VMdlVxlCopyVoxelVModel, , VMdlVxlGetRealVoxelVolume, VMdlVxlMdl2VoxelVMod
, VMdlVxlPrintVoxelVModel,

16.2.197 VMdlVxlFindMaxNeighbor (vmvoxels.c:623)

```
void VMdlVxlFindMaxNeighbor(const VMdlVoxelVModelStruct *VxlVMdl,
                            int i,
                            int j,
                            int k,
                            TrivImagePixelTypeUnion *Max)
```

VxlVMdl: A voxel vmodel to find the maximum value within the neighbors of the given voxel.

i, j, k: Indices of voxel in the volume.

Max: Output parameter, Maximal voxel value.

Returns: void

Description: Given a voxel in a voxel model, find the maximum value within (6-connectivity) neighboring voxels.
See also: VMdlVxlMdl2VoxelVModel, VMdlVxlMdl2VoxelVModel2, VMdlVxlCrv2VoxelVModel, , VMdlVxlFreeVoxelVModel, VMdlVxlCopyVoxelVModel, VMdlVxlVoxelOffsetBW, , VMdlVxlVoxelOffsetGray, VMdlVxlInPixelOnBndry, VMdlVxlPrintVoxelVModel, , VMdlVxlFindMinNeighbor, VMdlVxlVoxelOffsetGray,

16.2.198 VMdlVxlFindMinNeighbor (vmvoxels.c:575)

```
void VMdlVxlFindMinNeighbor(const VMdlVoxelVModelStruct *VxlVMdl,
                            int i,
                            int j,
                            int k,
                            TrivImagePixelTypeUnion *Min)
```

VxlVMdl: A voxel vmodel to find the minimum value within the neighbors of the given voxel.

i, j, k: Indices of voxel in the volume.

Min: Output parameter, Minimal voxel value.

Returns: void

Description: Given a voxel in a voxel model, find the minimum value within (6-connectivity) neighboring voxels.
See also: VMdlVxlMdl2VoxelVModel, VMdlVxlMdl2VoxelVModel2, VMdlVxlCrv2VoxelVModel, , VMdlVxlFreeVoxelVModel, VMdlVxlCopyVoxelVModel, VMdlVxlVoxelOffsetBW, , VMdlVxlVoxelOffsetGray, VMdlVxlInPixelOnBndry, VMdlVxlPrintVoxelVModel, , VMdlVxlFindMaxNeighbor, VMdlVxlVoxelOffsetGray,

16.2.199 VMdlVxlFreeVoxelVModel (vmvoxels.c:1608)

```
void VMdlVxlFreeVoxelVModel(VMdlVoxelVModelStruct *VxlVMdl)
```

VxlVMdl: Voxel vmodel to free.

Returns: void

Description: Frees a voxel model.
See also: VMdlVxlMdl2VoxelVModel, VMdlVxlMdl2VoxelVModel2, VMdlVxlCrv2VoxelVModel, , VMdlVxlCopyVoxelVModel, VMdlVxlPrintVoxelVModel,

16.2.200 VMdlVxlGetPixelInPimage (vmvoxels.c:229)

```
int VMdlVxlGetPixelInPimage(const VMdlVoxelVModelStruct *VxlVMdl,
                            int i,
                            int j,
                            int k,
                            TrivImagePixelTypeUnion *Value)
```

VxlVMdl: A voxel vmodel to get the given voxel value.

i, j, k: Indices of voxel in the volume.

Value: Output parameter, will be assigned the pixel value.

Returns: TRUE if successful, FALSE otherwise.

Description: Get a given pixel value.
See also: VMdlVxlMdl2VoxelVModel, VMdlVxlMdl2VoxelVModel2, VMdlVxlCrv2VoxelVModel, , VMdlVxlFreeVoxelVModel, VMdlVxlCopyVoxelVModel, VMdlVxlVoxelOffsetBW, , VMdlVxlVoxelOffsetGray, VMdlVxlOutPixelOnBndry, VMdlVxlPrintVoxelVModel,

16.2.201 VMdlVxlGetPixelInPimage2 (vmvoxels.c:174)

```
int VMdlVxlGetPixelInPimage2(const VMdlVoxelVModelStruct *VxlVMdl,
                             int ImgIdx,
                             int VxlIdx,
                             TrivImagePixelTypeUnion *Value)
```

VxlVMdl: A voxel vmodel to get the given voxel value.

ImgIdx: Image index of voxel.

VxlIdx: Index of voxel in the image.

Value: Output parameter, will be assigned the pixel value.

Returns: TRUE if successful, FALSE otherwise.

Description: Get a given pixel value.
See also: VMdlVxlMdl2VoxelVModel, VMdlVxlMdl2VoxelVModel2, VMdlVxlCrv2VoxelVModel, , VMdlVxlFreeVoxelVModel, VMdlVxlCopyVoxelVModel, VMdlVxlVoxelOffsetBW, , VMdlVxlVoxelOffsetGray, VMdlVxlOutPixelOnBndry, VMdlVxlPrintVoxelVModel,

16.2.202 VMdlVxlGetRealVoxelVolume (vmvoxels.c:662)

```
CagdRType VMdlVxlGetRealVoxelVolume(const VMdlVoxelVModelStruct *VxlVMdl)
```

VxlVMdl: A voxel vmodel to compute the volume of a single voxel.

Returns: The real volume of a single voxel.

Description: Given a Voxel vmodel, compute the volume of a single voxel.
See also: VMdlVxlMdl2VoxelVModel, VMdlVxlMdl2VoxelVModel2, VMdlVxlCrv2VoxelVModel, , VMdlVxlFreeVoxelVModel, VMdlVxlCopyVoxelVModel, , VMdlVxlPrintVoxelVModel,

16.2.203 VMdlVxlHeuristicRadon (vmvoxels.c:1768)

```
VMdlVoxelVModelStruct *VMdlVxlHeuristicRadon(VMdlVxlRadonProjStruct
                                                    *ProjectionList,
                                             int ProjCount)
```

ProjectionList: Array of Projection elements, in each element there are a few variables. First is a pointer of unsigned char array, you should read the gray scale image and assign it the pointer. Second should contain image's width/length. Third is Theta field with the angle of the projection in degrees.

ProjCount: The number of elements in ProjectionList array, i.e. the number of projection we used.

Returns: Built Voxel vmodel of the projected images, or NULL if error.

Description: In this function, we reconstruct a voxels' cube model that (approximately) project into the given images relatively to their angles. We use the inverse radon transform to build the 3D cube, by solving a linear system of constraints, using a heuristic method. Written by Haitham Fadila and Noam Jacobi.

16.2.204 VMdlVxlInPixelOnBndry (vmvoxels.c:389)

```
int VMdlVxlInPixelOnBndry(const VMdlVoxelVModelStruct *VxlVMdl,
                          int i,
                          int j,
                          int k)
```

VxlVMdl: A voxel vmodel to test if the given voxel is on the boundary, with zero values denoting outside, non-zero inside.

i, j, k: Indices of voxel in the volume.

Returns: TRUE if this inside pixel has neighbors that are outside model.

Description: Check if this given pixel, that is assumed inside the binary voxel vmodel, is on the boundary, by seeking outside (6-connectivity) neighboring pixels.
See also: VMdlVxlMdl2VoxelVModel, VMdlVxlMdl2VoxelVModel2, VMdlVxlCrv2VoxelVModel, , VMdlVxlFreeVoxelVModel, VMdlVxlCopyVoxelVModel, VMdlVxlVoxelOffsetBW, , VMdlVxlVoxelOffsetGray, VMdlVxlOutPixelOnBndry, VMdlVxlPrintVoxelVModel,

16.2.205 VMdlVxlMdl2VoxelVModel (vmvoxels.c:867)

```
VMdlVoxelVModelStruct *VMdlVxlMdl2VoxelVModel(const IPObjectStruct *Mdl,
                                              VMdlSliceOpType SliceOper,
                                              const CagdRType *Dims,
                                              TrivImagePixelType PixelType,
                                              int AntiAliasing)
```

Mdl: The model to slice.

SliceOper: 0 for full (heterogeneous) slicing as an image. This mode is not supported by this function. 1 for outline of slice as curves (Image name is ignored). 2 for outline of slice as an image. 3 for all covered pixels in the slice as an image.

Dims: Resolution the slices as (XRes, YRes, ZRes).

PixelType: The type of image to create and return as the slice. Can be TRIV_IMAGE_UNDEFINED_TYPE to use the default type.

AntiAliasing: Resolution Factor. value greater than 1 means this is a higher-resolution voxel model built for antialiasing algorithm.

Returns: Built Voxel vmodel, or NULL if error.

Description: Builds a Voxel model from a given model (or trivar or vmodel) and desired resolution.
See also: VMdlVxlFreeVoxelVModel, VMdlVxlCopyVoxelVModel, VMdlVxlMdl2VoxelVModel2, , VMdlVxlGetRealVoxelVolume, VMdlVxlCrv2VoxelVModel, , VMdlVxlPrintVoxelVModel,

16.2.206 VMdlVxlMdl2VoxelVModel2 (vmvoxels.c:1008)

```
VMdlVoxelVModelStruct *VMdlVxlMdl2VoxelVModel2(const IPObjectStruct *Mdl,
                                               VMdlSliceOpType SliceOper,
                                               const CagdRType *Dims,
                                               TrivImagePixelType PixelType,
                                               int AntiAliasing)
```

Mdl: The model to slice.

SliceOper: 0 for full (heterogeneous) slicing as an image. This mode is not supported by this function. 1 for outline of slice as curves (Image name is ignored). 2 for outline of slice as an image. 3 for all covered pixels in the slice as an image.

Dims: Resolution the slices as (XRes, YRes, ZRes).

PixelType: The type of image to create and return as the slice. Can be TRIV_IMAGE_UNDEFINED_TYPE to use the default type.

AntiAliasing: Factor to enlarge the resolution and apply averaging to allow antialiasing. Can be 2/3/4 for x2/x3/x4 images sizes.

Returns: Built grayscale voxel vmodel, or NULL if error.

Description: Builds a grayscale voxel model from a given binary model and desired resolution.
See also: VMdlVxlFreeVoxelVModel, VMdlVxlCopyVoxelVModel, VMdlVxlMdl2VoxelVModel, , VMdlVxlGetRealVoxelVolume, VMdlVxlCrv2VoxelVModel, , VMdlVxlPrintVoxelVModel, VMdlVxlVoxelOffsetGray,

16.2.207 VMdlVxlMrchCubeVxlVMdl (vmvoxels.c:1082)

```
IPObjectStruct *VMdlVxlMrchCubeVxlVMdl(const VMdlVoxelVModelStruct *VxlVMdl,
                                       const CagdRType IsoLevel)
```

VxlVMdl: A voxel vmodel to compute marching cubes for.

IsoLevel: The iso level of the marching cubes to use.

Returns: Resulting marching cube model (a polygonal model).

Description: Given a Voxel vmodel, compute marching cubes over it.
See also: VMdlVxlMdl2VoxelVModel, VMdlVxlMdl2VoxelVModel2, VMdlVxlCrv2VoxelVModel, , VMdlVxlFreeVoxelVModel, VMdlVxlCopyVoxelVModel, , VMdlVxlGetRealVoxelVolume, VMdlVxlPrintVoxelVModel,

16.2.208 VMdlVxlOutPixelOnBndry (vmvoxels.c:449)

```
int VMdlVxlOutPixelOnBndry(const VMdlVoxelVModelStruct *VxlVMdl,
                           int i,
                           int j,
                           int k)
```

VxlVMdl: A voxel vmodel to test if the given voxel is on the boundary, with zero values denoting outside, non-zero inside.

i, j, k: Indices of voxel in the volume.

Returns: TRUE if this outside pixel has neighbors that are intside model.

Description: Check if this given pixel, that is assumed outside the binary voxel vmodel, is on the boundary, by seeking intside (6-connectivity) neighboring pixels.
See also: VMdlVxlMdl2VoxelVModel, VMdlVxlMdl2VoxelVModel2, VMdlVxlCrv2VoxelVModel, , VMdlVxlFreeVoxelVModel, VMdlVxlCopyVoxelVModel, VMdlVxlVoxelOffsetBW, , VMdlVxlVoxelOffsetGray, VMdlVxlInPixelOnBndry, VMdlVxlPrintVoxelVModel,

16.2.209 VMdlVxlPrintVoxelVModel (vmvoxels.c:1694)

```
void VMdlVxlPrintVoxelVModel(VMdlVoxelVModelStruct *VxlVMdl,
                             const char *VxlFormat,
                             FILE *f,
                             int Row)
```

VxlVMdl: Voxel vmodel to print to file.

VxlFormat: The printf format to use, for printing a pixel.

f: The file to print to.

Row: Image row index to print (one line) or -1 to print all the image (the full volume).

Returns: void

Description: Prints a voxel model.
See also: VMdlVxlMdl2VoxelVModel, VMdlVxlMdl2VoxelVModel2, VMdlVxlCrv2VoxelVModel, , VMdlVxlCopyVoxelVModel, VMdlVxlFreeVoxelVModel,

16.2.210 VMdlVxlSetPixelInPimage (vmvoxels.c:282)

```
int VMdlVxlSetPixelInPimage(const VMdlVoxelVModelStruct *VxlVMdl,
                            int i,
                            int j,
                            int k,
                            TrivImagePixelTypeUnion *Value)
```

VxlVMdl: A voxel vmodel to set the given voxel value.

i, j, k: Indices of voxel in the volume.

Value: Pointer to allocated space to update from.

Returns: TRUE if successful, FALSE otherwise.

Description: Set the given pixel value.
See also: VMdlVxlMdl2VoxelVModel, VMdlVxlMdl2VoxelVModel2, VMdlVxlCrv2VoxelVModel, , VMdlVxlFreeVoxelVModel, VMdlVxlCopyVoxelVModel, VMdlVxlVoxelOffsetBW, , VMdlVxlVoxelOffsetGray, VMdlVxlOutPixelOnBndry, VMdlVxlPrintVoxelVModel,

16.2.211 VMdlVxlSetPixelInPimage2 (vmvoxels.c:334)

```
int VMdlVxlSetPixelInPimage2(const VMdlVoxelVModelStruct *VxlVMdl,
                             int ImgIdx,
                             int VxlIdx,
                             TrivImagePixelTypeUnion *Value)
```

VxlVMdl: A voxel vmodel to set the given voxel value.

ImgIdx: Image index of voxel.

VxlIdx: Index of voxel in the image.

Value: Pointer to allocated space to update from.

Returns: TRUE if successful, FALSE otherwise.

Description: Set the given pixel value.
See also: VMdlVxlMdl2VoxelVModel, VMdlVxlMdl2VoxelVModel2, VMdlVxlCrv2VoxelVModel, , VMdlVxlFreeVoxelVModel, VMdlVxlCopyVoxelVModel, VMdlVxlVoxelOffsetBW, , VMdlVxlVoxelOffsetGray, VMdlVxlOutPixelOnBndry, VMdlVxlPrintVoxelVModel,

16.2.212 VMdlVxlVolumeVxlVMdl (vmvoxels.c:1161)

```
CagdRType VMdlVxlVolumeVxlVMdl(const VMdlVoxelVModelStruct *VxlVMdl)
```

VxlVMdl: A voxel vmodel to compute its volume.

Returns: The volume.

Description: Given a Voxel vmodel compute its volume, by counting voxel...
See also: VMdlVxlMdl2VoxelVModel, VMdlVxlMdl2VoxelVModel2, VMdlVxlCrv2VoxelVModel, , VMdlVxlFreeVoxelVModel, VMdlVxlCopyVoxelVModel, , VMdlVxlGetRealVoxelVolume, VMdlVxlPrintVoxelVModel,

16.2.213 VMdlVxlVoxelOffsetBW (vmvoxels.c:1211)

```
VMdlVoxelVModelStruct *VMdlVxlVoxelOffsetBW(const VMdlVoxelVModelStruct
                                                                    *VxlVMdl,
                                            int NumPixels)
```

VxlVMdl: A voxel vmodel to compute the offset in voxels.

NumPixels: If positive, how many layers to remove in the offset. If negative, how many layers to add in the offset.

Returns: The given voxel vmodel, after the offset, or NULL if error.

Description: Given a black and white voxel vmodel, compute voxels offset.
See also: VMdlVxlMdl2VoxelVModel, VMdlVxlCrv2VoxelVModel, VMdlVxlFreeVoxelVModel, , VMdlVxlCopyVoxelVModel, VMdlVxlGetRealVoxelVolume, , VMdlVxlVoxelOffsetGray, VMdlVxlInPixelOnBndry, VMdlVxlOutPixelOnBndry, , VMdlVxlPrintVoxelVModel,

16.2.214 VMdlVxlVoxelOffsetGray (vmvoxels.c:1413)

```
VMdlVoxelVModelStruct *VMdlVxlVoxelOffsetGray(
                                        const VMdlVoxelVModelStruct *VxlVMdl,
                                        int NumPixels)
```

VxlVMdl: A voxel vmodel to compute the offset in voxels.

NumPixels: If positive, how many layers to remove in the offset. If negative, how many layers to add in the offset.

Returns: The given voxel vmodel, after the offset, or NULL if error.

Description: Given a grayscale voxel vmodel, compute voxels offset.
See also: VMdlVxlMdl2VoxelVModel, VMdlVxlMdl2VoxelVModel2, VMdlVxlCrv2VoxelVModel, , VMdlVxlFreeVoxelVModel, VMdlVxlCopyVoxelVModel, VMdlVxlGetRealVoxelVolume, , VMdlVxlInPixelOnBndry, VMdlVxlOutPixelOnBndry, VMdlVxlPrintVoxelVModel, VMdlVxlFindMinNeighbor, VMdlVxlFindMaxNeighbor,

16.2.215 VMdlWriteObjectToFile (vmdl_dbg.c:160)

```
CagdBType VMdlWriteObjectToFile(IPObjectStruct *Obj,
                                const char *FileName,
                                CagdBType ShowOnGuIrit)
```

Obj: Irit object.

FileName: Save file name.

ShowOnGuIrit: To open the object in GuIrit.

Returns: TRUE in succeed, FALSE otherwise.

Description: Writes an object to file and possibly opens it in GuIrit viewer.

Chapter 17

Extra Library, xtra_lib

17.1 General Information

This library is not an official part of IRIT and contains public domain code that is used by routines in IRIT.
The interface of the library is defined in *include/extra_fn.h*.

17.2 Library Functions

17.2.1 BzrCrvInterp (bzrintrp.c:211)

interpolation

```
void BzrCrvInterp(IrtRType *Result, IrtRType *Input, int Size)
```

Result: Where the interpolated control points will be placed.

Input: Points to interpolate at node parameter values.

Size: Of control polygon.

Returns: void

Description: Blends the input points using the Interp array and puts the result blended point in Result array. Input and Result array are of size Size.

17.2.2 JacobiMatrixDiag4x4 (diag_mat.c:110)

```
void JacobiMatrixDiag4x4(IrtRType M[4][4],
                         IrtRType U[4][4],
                         IrtRType D[4][4],
                         IrtRType V[4][4])
```

M: Matrix to diagonalize.

U: The left orthogonal (=rotation) matrix.

D: The diagonal matrix.

V: The eight orthogonal matrix.

Returns: void

Description: performs a diagonalization of a 4x4 matrix, as U D V, D diagonal and U and V orthogonal matrices U = V^-1 = V^T.
See also: JacobiMatrixDiagNxN,

17.2.3 JacobiMatrixDiagNxN (diag_mat.c:51)

```
void JacobiMatrixDiagNxN(IrtRType *M[],
                         IrtRType *U[],
                         IrtRType *D[],
                         IrtRType *V[],
                         int n)
```

M: Matrix to diagonalize.

U: The left orthogonal (=rotation) matrix.

D: The diagonal matrix.

V: The eight orthogonal matrix.

n: Dimension of the matrices as n times n.

Returns: void

Description: Performs a diagonalization of a NxN matrix, as U D V, D diagonal and U and V orthogonal matrices U = V^-1 = V^T.
See also: JacobiMatrixDiag4x4,

17.2.4 SvdDecomp (nure_svd.c:342)

```
void SvdDecomp(IrtRType **a, int m, int n, IrtRType *w, IrtRType **v)
```

a: Input matrix also serving to store U orthogonal output, of dimension m x n, m >= n.

m, n: The dimensions of a.

w: The diagonal singular values, as a vector of length m.

v: The V orthogonal output, dimensions n x n.

Returns: void

Description: SVD decomposition of matrix A as U W V, U of size m x m, V of size n x n and W is diagonal with the signular values, stored as a vector. Due to the Fortran style of this code all indices are from 1 to k, so an array from 1 to k actually have k + 1 elements in this C translation.
See also: SvdMatrixNxN,

17.2.5 SvdLeastSqr (nure_svd.c:798)

singular value decomposition

linear systems

```
IrtRType SvdLeastSqr(XtraSvdLSGenInfoStruct *SLSGI, IrtRType *x, IrtRType *b)
```

SLSGI: The SVD least sqaure general info reference.

x: The vector of sought solution of size Nx.

b: The vector of coefficients of size NData.

Returns: The reciprocal of the condition number, if A != NULL, zero otherwise.

Description: Least square solves A x = b. The vector X is of size Nx, vector b is of size NData and matrix A (SVD decomposed in SLSGI) is of size Nx by NData. Uses singular value decomposition.
See also: SvdMatrix4x4, SvdLeastSqrInit, SvdLeastSqrFree,

17.2.6 SvdLeastSqrFree (nure_svd.c:845)

singular value decomposition

linear systems

```
void SvdLeastSqrFree(XtraSvdLSGenInfoStruct *SLSGI)
```

SLSGI: The SVD least sqaure general info reference.

Returns: void

Description: Least square solves A x = b. The vector X is of size Nx, vector b is of size NData and matrix A (SVD decomposed in SLSGI) is of size Nx by NData. Uses singular value decomposition.
See also: SvdMatrix4x4, SvdLeastSqr, SvdLeastSqrInit,

17.2.7 SvdLeastSqrInit (nure_svd.c:658)

singular value decomposition

linear systems

```
XtraSvdLSGenInfoStruct *SvdLeastSqrInit(IrtRType *A,
                                        int NData,
                                        int Nx,
                                        IrtRType *RecipCondNum)
```

A: The matrix of size Nx by NData.

NData, Nx: Dimensions of input.

RecipCondNum: The Recirocating value of the condition number.

Returns: The SVD least sqaure general info reference.

Description: Init the least squares solver of A x = b. Matrix A is of size Nx by NData. Uses singular value decomposition.
See also: SvdMatrix4x4. SvdLeastSqr, SvdLeastSqrFree,

17.2.8 SvdMatrix4x4 (nure_svd.c:45)

singular value decomposition

linear systems

matrices

```
int SvdMatrix4x4(IrtHmgnMatType M,
                 IrtRType U[3][3],
                 IrtVecType S,
                 IrtRType V[3][3])
```

M: Source matrix to decompose.

U: Left orthonormal matrix.

S: Singular components.

V: Right orthonormal matrix.

Returns: FALSE if failed, TRUE otherwise.

Description: Perform singular value decomposition on the 3x3 main minor of the affine transformation matrix.
See also: SvdLeastSqr, SvdMatrixNxN,

17.2.9 SvdMatrixNxN (nure_svd.c:284)

singular value decomposition

linear systems

matrices

```
void SvdMatrixNxN(IrtRType *M, IrtRType *U, IrtRType *S, IrtRType *V, int n)
```

M: Source matrix to decompose.

U: Left orthonormal matrix, size n x n.

S: A vector holding the singular components.

V: Right orthonormal matrix, size n.

n: size of square matrix, size n x n.

Returns: void

Description: Perform singular value decomposition on an NxN matrix M as U S V.
See also: SvdLeastSqr, SvdDecomp, SvdMatrix3x3,

Chapter 18

Programming Examples

This chapter describes several simple examples of C programs that exploits the libraries of IRIT. All external function are defined in the *include* subdirectory of IRIT and one can 'grep' there for the exact include file that contains a certain function name. All C programs in all C files should include 'irit_sm.h' as their first include file of IRIT, before any other include file of IRIT. Header files are set so C++ code can compile and link to it without any special treatment.

18.1 Setting up the Compilation Environment

In order to compile programs that uses the libraries of IRIT, a makefile has to be constructed. Assuming IRIT is installed in */usr/local/irit*, here is a simple makefile that can be used (for a unix environment):

```
IRIT_DIR = /usr/local/irit

include $(IRIT_DIR)/makeflag.unx

OBJS = program.o

program: $(OBJS)
$(CC) $(CFLAGS) -o program $(OBJS) $(LIBS) -lm $(MORELIBS)
```

The simplicity of this makefile is drawn from the complexity of makeflag.unx. The file makeflag.unx sets the CC, CFLAGS, LIBS, and MORELIBS for the machined using among other things. Furthermore, makeflag.unx also sets the default compilation rules from C sources to object files. The file makeflag.unx had to be modified once, when IRIT was installed on this system. If the used system is not a unix environment, then the file makefile.unx will have to be replaced with the proper makeflag file. In an OS2 environment, using the emx gcc compiler, the makefile is even simpler since the linking rule is also defined in makeflag.os2:

```
IRIT_DIR = \usr\local\irit

include $(IRIT_DIR)\makeflag.os2

OBJS = program.o

program.exe: $(OBJS)
```

Finally, here is a proper makefile for Windows NT:

```
IRIT_DIR = \usr\local\irit

include $(IRIT_DIR)\makeflag.wnt

OBJS = program.obj

program.exe: $(OBJS)
        $(IRITCONLINK) -out:program.exe $(OBJS) $(LIBS) $(W32CONLIBS)
```

18.2 Simple C Programs using IRIT

Now that we have an idea how to compile C code using IRIT, here are several examples to read, manipulate and write IRIT data files. You will be able to find all these examples in the doc/cexample directory.

18.2.1 Boolean Operations over Polyhedra (boolean.c)

This example demonstates the use of the Boolean operations package (bool_lib), creates few polygonal primitives and apply Boolean operations between them. The resulting model is dumped to stdout.

```
/*****************************************************************************
* Simple example to demonstrate the use of Booleans.                         *
******************************************************************************
* (C) Gershon Elber, Technion, Israel Institute of Technology                *
******************************************************************************
* Written by:  Gershon Elber                                Ver 1.0, June 1995   *
*****************************************************************************/

#include "inc_irit/irit_sm.h"
#include "inc_irit/allocate.h"
#include "inc_irit/iritprsr.h"
#include "inc_irit/geom_lib.h"
#include "inc_irit/bool_lib.h"

void main(int argc, char **argv)
{
    IPObjectStruct *PBox, *PCylin, *PCone, *PSphere, *PTmp1, *PTmp2;
    IrtVecType Pt, Dir;

    /* Make sure all polygonal models have circular vertex lists. */
    IPSetPolyListCirc(TRUE);

    /* And set the resolution desired. */
    PrimSetResolution(50);

    /* Build the primitives. */
    Pt[0] = Pt[1] = -1;
    Pt[2] = 0;
    PBox = PrimGenBOXObject(Pt, 2, 2, 1);
    Pt[0] = Pt[1] = 0;
    Pt[2] = 0.5;
    Dir[0] = Dir[1] = 0;
    Dir[2] = 3;
    PCone = PrimGenCONEObject(Pt, Dir, 0.7, 0);
    Dir[2] = 2.5;
    PSphere = PrimGenSPHEREObject(Dir, 0.6);
    Pt[0] = Pt[1] = 0;
    Pt[2] = -1;
    Dir[0] = Dir[1] = 0;
    Dir[2] = 5;
    PCylin = PrimGenCYLINObject(Pt, Dir, 0.2, 3);

    /* Time for some Booleans. */
    PTmp1 = BooleanOR(PBox, PCone);
    PTmp2 = BooleanOR(PTmp1, PSphere);
    IPFreeObject(PTmp1);
    PTmp1 = BooleanSUB(PTmp2, PCylin);
    IPFreeObject(PTmp2);

    IPFreeObject(PBox);
    IPFreeObject(PCone);
    IPFreeObject(PSphere);
    IPFreeObject(PCylin);

    IPStdoutObject(PTmp1, FALSE);
    IPFreeObject(PTmp1);


    exit(0);
}
```

18.2.2 Distance Maps to Planar Curves (dist_map.c)

This example demonstates the use of symbolic distance maps computations to freeform curves, in the plane, and the sampling of the map into an image that is dumped out.

```
/*****************************************************************************
* Computes a 2D grid of distances approximating the distance field to a crv. *
*****************************************************************************
* (C) Gershon Elber, Technion, Israel Institute of Technology               *
*****************************************************************************
* Written by:  Gershon Elber                                Ver 1.0, June 2003   *
*****************************************************************************/

#include "inc_irit/irit_sm.h"
#include "inc_irit/allocate.h"
#include "inc_irit/iritprsr.h"
#include "inc_irit/user_lib.h"
#include "inc_irit/geom_lib.h"
#include "inc_irit/cagd_lib.h"
#include "inc_irit/symb_lib.h"
#include "inc_irit/misc_lib.h"

#define MAP_DIST_2_COLOR(RelDist, Pxl)  { \
        (Pxl) -> r = ((unsigned char) ((RelDist) * 255)); \
        (Pxl) -> g = ((unsigned char) (4 * (RelDist) * (1.0 - (RelDist)) * 255)); \
        (Pxl) -> b = ((unsigned char) ((1.0 - (RelDist)) * 255)); }

static char *CtrlStr =
#ifdef IRIT_DOUBLE
    "dist_map x%-XMin|XMax!F!F y%-YMin|YMax!F!F t%-Tolerance!F h%- I!-ImageName|XSize|YSize!s!d!d IritFile!*s";
#else
    "dist_map x%-XMin|XMax!f!f y%-YMin|YMax!f!f t%-Tolerance!f h%- I!-ImageName|XSize|YSize!s!d!d IritFile!*s";
#endif /* IRIT_DOUBLE */

/*****************************************************************************
* DESCRIPTION:                                                               M
* Main module - Reads command line and do what is needed...                     M
* Example: "dist_map.exe -I cubic1.ppm 100 100 cubic1.itd"                    M
*                                                                            *
* PARAMETERS:                                                                M
*   argc, argv:  Command line.                                               M
*                                                                            *
* RETURN VALUE:                                                              M
*   void                                                                     M
*                                                                            *
* KEYWORDS:                                                                  M
*   main                                                                     M
*****************************************************************************/
void main(int argc, char **argv)
{
    int NumFiles, Error,
        TolFlag = FALSE,
        XDomainFlag = FALSE,
        YDomainFlag = FALSE,
        HelpFlag = FALSE,
        ImageFlag = FALSE,
        ImageXSize = 100,
        ImageYSize = 100;
    IrtRType
        XDomain[2] = { -1.0, 1.0 },
        YDomain[2] = { -1.0, 1.0 },
        Tolerance = 0.01;
    char **FileNames,
        *ImageName = NULL;
    IPObjectStruct *PObjs;
    MiscWriteGenInfoStructPtr GI;

    if ((Error = GAGetArgs(argc, argv, CtrlStr,
                           &XDomainFlag, &XDomain[0], &XDomain[1],
                           &YDomainFlag, &YDomain[0], &YDomain[1],
```

```
                        &TolFlag, &Tolerance,
                        &HelpFlag, &ImageFlag, &ImageName,
                        &ImageXSize, &ImageYSize,
                        &NumFiles, &FileNames)) != 0) {
    GAPrintErrMsg(Error);
    GAPrintHowTo(CtrlStr);
    exit(1);
}

if (ImageName == NULL || NumFiles != 1) {
    fprintf(stderr, "No image or irit data files to process.\n");
    exit(2);
}

if (HelpFlag) {
    GAPrintHowTo(CtrlStr);
    exit(0);
}

fprintf(stderr, "Processing domain [%f : %f] x [%f : %f]\n\tto image \"%s\" of size [%d x %d]\n",
        XDomain[0], XDomain[1],
        YDomain[0], YDomain[1],
        ImageName, ImageXSize, ImageYSize);

/* Get the data from all the input files. */
IPSetFlattenObjects(TRUE);
if ((PObjs = IPGetDataFiles(FileNames, NumFiles, TRUE, TRUE)) == NULL) {
    fprintf(stderr, "Failed to load file \"%s\"\n", FileNames[0]);
    exit(1);
}
fprintf(stderr, "Successfully loaded crv geometry \"%s\"\n",
        FileNames[0]);

/* Ray trace the glass object. */
if (IP_IS_CRV_OBJ(PObjs)) {
    int i, j;
    IrtRType MaxDist, **Image;
    IrtImgPixelStruct
        *ImageLine = (IrtImgPixelStruct *)
            IritMalloc(sizeof(IrtImgPixelStruct) * ImageXSize);

    Image = (IrtRType **) IritMalloc(sizeof(IrtRType *) * ImageYSize);
    for (j = 0; j < ImageYSize; j++)
        Image[j] = (IrtRType *) IritMalloc(sizeof(IrtRType) * ImageXSize);

    MaxDist = SymbDistBuildMapToCrv(PObjs -> U.Crvs, Tolerance,
                                    XDomain, YDomain,
                                    Image, ImageXSize, ImageYSize);

    /* Dump the map as an image. */
    GI = IrtImgWriteOpenFile(argv, ImageName, IRIT_IMAGE_PPM3_TYPE,
                             FALSE, ImageXSize, ImageYSize);

    for (j = 0; j < ImageYSize; j++) {
        for (i = 0; i < ImageXSize; i++)
            MAP_DIST_2_COLOR(Image[i][j] / MaxDist, &ImageLine[i]);

        IrtImgWritePutLine(GI, NULL, ImageLine);
    }

    IrtImgWriteCloseFile(GI);

    for (j = 0; j < ImageYSize; j++)
        IritFree((VoidPtr) Image[j]);
    IritFree((VoidPtr) Image);
    IritFree((VoidPtr) ImageLine);


}
```

```
    else {
        fprintf(stderr, "... but expected a curve, aborting!\n");
    }

    exit(0);
}
```

18.2.3 Importance Edges Evaluations in Polygonal Meshes (imprtnc.c)

This example demonstates the tesselation of arbitrary input model(s) into polygons and the probabalistic estimation of importance edges in the polygonal geometry. Dumped out is the polygonal geometry with vertices having UV coordinates if available, normals if available, and importance.

```
/*****************************************************************************
*   Computes the importance of vertices and dump as vertex list + polygons   *
*****************************************************************************
* (C) Gershon Elber, Technion, Israel Institute of Technology               *
*****************************************************************************
* Written by:  Gershon Elber                                 Ver 1.0, June 1995   *
*****************************************************************************/

#include "inc_irit/irit_sm.h"
#include "inc_irit/allocate.h"
#include "inc_irit/iritprsr.h"
#include "inc_irit/geom_lib.h"
#include "inc_irit/misc_lib.h"
#include "inc_irit/ip_cnvrt.h"

IRIT_STATIC_DATA char
    *CtrlStr = "Imprtnc c%- h%- DFiles!*s";
IRIT_STATIC_DATA int GlblVrtxImportanceCount,
    GlblCrvtrInfoFlag = FALSE;
IRIT_STATIC_DATA IrtRType GlblVrtxImportanceVal;
IRIT_STATIC_DATA IPVertexStruct *GlblVrtxImportance;

static void DumpOneTraversedObject(IPObjectStruct *PObj,
                                   IrtHmgnMatType Mat,
                                   void *Data);
static IPObjectStruct *SetCurvatureEstimates(IPObjectStruct *PObj);
static void DumpOneObjData(IPObjectStruct *PObj);
static void ProcessVertexImportance(IPVertexStruct *V1,
                                    IPVertexStruct *V2,
                                    IPPolygonStruct *Pl1,
                                    IPPolygonStruct *Pl2);
static void GenPolyImportance(IPObjectStruct *PObj);

void main(int argc, char **argv)
{
    int NumFiles, Error,
        HelpFlag = FALSE;
    char **FileNames;
    IPObjectStruct *PObjects;
    IrtHmgnMatType CrntViewMat;

    if ((Error = GAGetArgs(argc, argv, CtrlStr, &GlblCrvtrInfoFlag,
                           &HelpFlag, &NumFiles, &FileNames)) != 0) {
        GAPrintErrMsg(Error);
        GAPrintHowTo(CtrlStr);
        exit(1);
    }

    if (HelpFlag) {
        fprintf(stderr, "This is Importance testing...\n");
        GAPrintHowTo(CtrlStr);
        exit(0);
    }

    /* Get the data files: */
    IPSetFlattenObjects(FALSE);
```

```
    if ((PObjects = IPGetDataFiles(FileNames, NumFiles, TRUE, FALSE)) == NULL)
        exit(1);
    PObjects = IPResolveInstances(PObjects);

    if (IPWasPrspMat)
        MatMultTwo4by4(CrntViewMat, IPViewMat, IPPrspMat);
    else
        IRIT_GEN_COPY(CrntViewMat, IPViewMat, sizeof(IrtHmgnMatType));

    /* Here some useful parameters to play with in tesselating freeforms: */
    IPFFCState.FineNess = 15; /* Resolution of tesselation, larger is finer. */
    IPFFCState.ComputeUV = TRUE;       /* Wants UV coordinates for textures. */
    IPFFCState.FourPerFlat = TRUE;/* 4 polygons per ~flat patch, 2 otherwise.*/
    IPFFCState.LinearOnePolyFlag = TRUE;   /* Linear srf generates one poly. */

    IPTraverseObjListHierarchy(PObjects, CrntViewMat, DumpOneTraversedObject);

    IPFreeObjectList(PObjects);
}

/*****************************************************************************
* DESCRIPTION:                                                               *
*   Update curvature property attribute to vertices of the given poly model. *
*                                                                            *
* PARAMETERS:                                                                *
*   PObj:   Poly model to udpate curvatrue info into.                            *
*                                                                            *
* RETURN VALUE:                                                              *
*   IPObjectStruct *:   A poly model, similar to PObj, but with curvature    *
*                        information attached as attributes.                  *
*****************************************************************************/
static IPObjectStruct *SetCurvatureEstimates(IPObjectStruct *PObj)
{
    int OldCirc = IPSetPolyListCirc(TRUE);
    IPPolygonStruct *Pl;
    IPObjectStruct *PTmp1, *PTmp2;

    /* Convert to a regular polygonal model with triangles only. */
    for (Pl = PObj -> U.Pl; Pl != NULL; Pl = Pl -> Pnext) {
        if (IPVrtxListLen(Pl -> PVertex) != 3)
            break;
    }
    PTmp2 = IPCopyObject(NULL, PObj, FALSE);
    GMVrtxListToCircOrLin(PTmp2 -> U.Pl, TRUE);
    if (Pl != NULL) {
        PTmp1 = GMConvertPolysToTriangles(PTmp2);
        IPFreeObject(PTmp2);
        PTmp2 = GMRegularizePolyModel(PTmp1, TRUE, 0.0);
        IPFreeObject(PTmp1);
    }

    GMVrtxListToCircOrLin(PTmp2 -> U.Pl, FALSE);
    IPSetPolyListCirc(OldCirc);

    GMPlCrvtrSetCurvatureAttr(PTmp2 -> U.Pl, 1, TRUE);

    return PTmp2;
}

/*****************************************************************************
* DESCRIPTION:                                                               *
*   Call back function of IPTraverseObjListHierarchy. Called on every non    *
* list object found in hierarchy.                                            *
*                                                                            *
* PARAMETERS:                                                                *
*   PObj:       Non list object to handle.                                   *
*   Mat:        Transformation matrix to apply to this object.               *
*   Data:       Optional call back data -ignored.                             *
*                                                                            *
```

```
* RETURN VALUE:                                                            *
*   void                                                                   *
*****************************************************************************/
static void DumpOneTraversedObject(IPObjectStruct *PObj,
                                   IrtHmgnMatType Mat,
                                   void *Data)
{
    IPObjectStruct *PObjs;

    if (IP_IS_FFGEOM_OBJ(PObj))
        PObjs = IPConvertFreeForm(PObj, &IPFFCState);
    else
        PObjs = PObj;

    for (PObj = PObjs; PObj != NULL; PObj = PObj -> Pnext) {
        if (IP_IS_POLY_OBJ(PObj)) {
            if (GlblCrvtrInfoFlag) {
                IPObjectStruct
                    *PTri = SetCurvatureEstimates(PObj);

                DumpOneObjData(PTri);
                IPFreeObject(PTri);
            }
            else
                DumpOneObjData(PObj);
        }
    }
}

/*****************************************************************************
* DESCRIPTION:                                                              *
*   Convert the polygonal mesh into a vertex list with importance and the   *
* polygons as indices into this list.                                       *
*                                                                           *
* PARAMETERS:                                                               *
*   PObj:   A polygonal mesh to dump.                                       *
*                                                                           *
* RETURN VALUE:                                                             *
*   void                                                                    *
*****************************************************************************/
static void DumpOneObjData(IPObjectStruct *PObj)
{
    IPPolyVrtxArrayStruct
        *PVIdx = IPCnvrtIritPolyToPolyVrtxArray(PObj, TRUE, 0);
    int        i,
        **Polygons = PVIdx -> Polygons;
    IPVertexStruct
        **Vertices = PVIdx -> Vertices;

    /* Compute importance for the vertices as "Imprt" attributes.  Note the */
    /* PVIdx data structure points on the original vertices so we are fine. */
    GenPolyImportance(PObj);

    /* Dump the vertices: */
    fprintf(stderr, "OBJECT \"%s\" - VERTICES:\n", PObj -> ObjName);
    for (i = 0; Vertices[i] != NULL; i++) {
        IrtRType R;
        float *uv;

        printf("%3d:  %6.3f %6.3f %6.3f", i,
               Vertices[i] -> Coord[0],
               Vertices[i] -> Coord[1],
               Vertices[i] -> Coord[2]);
        if (IP_HAS_NORMAL_VRTX(Vertices[i]))
            printf(" [%6.3f %6.3f %6.3f]",
                   Vertices[i] -> Normal[0],
                   Vertices[i] -> Normal[1],
                   Vertices[i] -> Normal[2]);
        if ((uv = AttrIDGetUVAttrib(Vertices[i] -> Attr, IRIT_ATTR_CREATE_ID(uvvals))) != NULL)
```

```
            printf(" {%6.3f %6.3f}", uv[0], uv[1]);

        R = AttrIDGetRealAttrib(Vertices[i] -> Attr, IRIT_ATTR_CREATE_ID(Imprt));
        if (!IP_ATTR_IS_BAD_REAL(R))
            printf(" (%6.3f)", R);

        if (GlblCrvtrInfoFlag) {
            printf("\n\t<K1=%6.3f D1 = %s>\n\t<K2=%6.3f D2 = %s>",
                   AttrIDGetRealAttrib(Vertices[i] -> Attr, IRIT_ATTR_CREATE_ID(K1Curv)),
                   AttrIDGetStrAttrib(Vertices[i] -> Attr, IRIT_ATTR_CREATE_ID(D1)),
                   AttrIDGetRealAttrib(Vertices[i] -> Attr, IRIT_ATTR_CREATE_ID(K2Curv)),
                   AttrIDGetStrAttrib(Vertices[i] -> Attr, IRIT_ATTR_CREATE_ID(D2)));
        }

        printf("\n");
    }

    fprintf(stderr, "\nOBJECT \"%s\" - Vertices in polygons:\n",
            PObj -> ObjName);
    for (i = 0; PVIdx -> PPolys[i] != NULL; i++) {
        IPPolyPtrStruct
            *PPoly = PVIdx -> PPolys[i];

        printf("%3d: ", i);
        for ( ; PPoly != NULL; PPoly = PPoly -> Pnext) {
            printf("%3d ", AttrIDGetIntAttrib(PPoly -> Poly -> Attr, IRIT_ATTR_CREATE_ID(_PIdx)));
        }
        printf("\n");
    }

    /* Dump the polygons: */
    fprintf(stderr, "\nOBJECT \"%s\" - Polygons from vertices:\n",
            PObj -> ObjName);
    for (i = 0; Polygons[i] != NULL; i++) {
        int *Pl = Polygons[i];

        printf("%3d: ", i);
        while (*Pl >= 0)
            fprintf(stderr, " %5d", *Pl++);

        fprintf(stderr, "    -1\n");
    }

    IPPolyVrtxArrayFree(PVIdx);
}

/*****************************************************************************
* DESCRIPTION:                                                               *
*   Call back function for GMPolyAdjacncyVertex to process every edge of a   *
* given vertex.  The edge is provided as (V, V -> Pnext)                     *
*                                                                            *
* PARAMETERS:                                                                *
*   V1, V2:    Two vertices defining this edge.  Note the vertices are NOT   *
*              necessarily chained together into a list.                      *
*   Pl1, Pl2:  The two polygons that share this edge.  The edge (V1, V2) is  *
*              in both Pl1 and Pl2, with not necessarily the exact pointers  *
*              IPVertexStruct of V1 and V2.                                      *
*                                                                            *
* RETURN VALUE:                                                              *
*   void                                                                     *
*****************************************************************************/
static void ProcessVertexImportance(IPVertexStruct *V1,
                                    IPVertexStruct *V2,
                                    IPPolygonStruct *Pl1,
                                    IPPolygonStruct *Pl2)
{
    if (!IRIT_PT_APX_EQ_EPS(V1 -> Coord, GlblVrtxImportance -> Coord,
                            IRIT_EPS) &&
        !IRIT_PT_APX_EQ_EPS(V2 -> Coord, GlblVrtxImportance -> Coord,
```

```
                            IRIT_EPS))
        fprintf(stderr, "Edge does not match the given vertex - adj error!\n");

    if (Pl1 != NULL && Pl2 != NULL) {
        GlblVrtxImportanceCount++;
        GlblVrtxImportanceVal += acos(IRIT_DOT_PROD(Pl1 -> Plane,
                                                    Pl2 -> Plane));
    }
}

/*****************************************************************************
* DESCRIPTION:                                                               *
*   Computes importance for each vertex in the given polygonal mesh.         *
*                                                                            *
* PARAMETERS:                                                                *
*   PObj:         Polygonal model to process for importance.                     *
*                                                                            *
* RETURN VALUE:                                                              *
*   void                                                                     *
*****************************************************************************/
static void GenPolyImportance(IPObjectStruct *PObj)
{
    VoidPtr
        PAdj = GMPolyAdjacncyGen(PObj, IRIT_EPS);
    IPPolygonStruct *Pl;

    for (Pl = PObj -> U.Pl; Pl != NULL; Pl = Pl -> Pnext) {
        int PlImportanceCount = 0;
        IrtRType
            PlImportanceVal = 0.0;
        IPVertexStruct *V;

        for (V = Pl -> PVertex; V != NULL; V = V -> Pnext) {
            GlblVrtxImportanceVal = 0.0;
            GlblVrtxImportanceCount = 0;
            GlblVrtxImportance = V;

            GMPolyAdjacncyVertex(V, PAdj, ProcessVertexImportance);

            if (GlblVrtxImportanceCount > 0) {
                GlblVrtxImportanceVal /= (GlblVrtxImportanceCount * M_PI);

                AttrIDSetRealAttrib(&V -> Attr, IRIT_ATTR_CREATE_ID(Imprt), GlblVrtxImportanceVal);
            }
            else
                fprintf(stderr, "Failed to compute Importance for vertex\n");
        }
    }

    GMPolyAdjacncyFree(PAdj);
}
```

18.2.4 Least sqaures curve fitting and process communication (lst_sqrs.c)

This example creates a Bspline curve that least squares approximates a given set of data points. In the default values, a quadratic Bspline curve of 10 control points least sqaures approximates a set of 100 three dimensional points. Also shown is a polyline of the original (100) points. Arbitrary number of curves will be displayed every, one for every ¡return¿ keystroke until Q¡return¿ is typed.

```
/*****************************************************************************
* Least squares fit a curve to random points.                                *
* We also see how to fork out a display device and communicate with it.      *
*****************************************************************************
* (C) Gershon Elber, Technion, Israel Institute of Technology                *
*****************************************************************************
* Written by:  Gershon Elber                                Ver 1.0, June 1995   *
*****************************************************************************/

#include "inc_irit/irit_sm.h"
```

```
#include "inc_irit/iritprsr.h"
#include "inc_irit/allocate.h"
#include "inc_irit/attribut.h"
#include "inc_irit/cagd_lib.h"
#include "inc_irit/geom_lib.h"
#include "inc_irit/grap_lib.h"
#include "inc_irit/misc_lib.h"

static char *CtrlStr =
    "Lst_Sqrs n%-#Pts!d d%-Degree!d f%-DOF!d p%-PrgmName!s h%-";

void main(int argc, char **argv)
{
    int i, Error, PrgmIO,
        NumOfPoints = 100,
        NumOfPtsFlag = FALSE,
        Degree = 3,
        DegreeFlag = FALSE,
        NumOfDOF = 10,
        NumOfDOFFlag = FALSE,
        PrgmFlag = FALSE,
        HelpFlag = FALSE;
    char *Err,
        *Program = getenv("IRIT_DISPLAY");

#ifdef __WINNT__
    if (Program == NULL)
        Program = "wntgdrvs -s-";
#endif /* __WINNT__ */
#ifdef __UNIX__
    if (Program == NULL)
        Program = "x11drvs -s-";
#endif /* __UNIX__ */

    if ((Error = GAGetArgs(argc, argv, CtrlStr,
                           &NumOfPtsFlag, &NumOfPoints,
                           &DegreeFlag, &Degree,
                           &NumOfDOFFlag, &NumOfDOF,
                           &PrgmFlag, &Program,
                           &HelpFlag)) != 0) {
        GAPrintErrMsg(Error);
        GAPrintHowTo(CtrlStr);
        exit(1);
    }

    if (HelpFlag) {
        GAPrintHowTo(CtrlStr);
        exit(0);
    }

    IPSocSrvrInit();            /* Initialize the listen socket for clients. */

    if ((PrgmIO = IPSocExecAndConnect(Program,
                                      getenv("IRIT_BIN_IPC") != NULL)) >= 0) {
        char Line[IRIT_LINE_LEN];
        IPObjectStruct
            *PClrObj = IPGenStrObject("command_", "clear", NULL);

        do {
            CagdPtStruct
                *PtList = NULL;
            IPPolygonStruct
                *PPoly = IPAllocPolygon(0, NULL, NULL);
            CagdCrvStruct *Crv;
            IPObjectStruct *PCrvObj, *PPolyObj;

            for (i = 0; i < NumOfPoints; i++) {
                int j;
                IPVertexStruct *V;
```

```
            CagdPtStruct
                *Pt = CagdPtNew();

            if (i == 0) {
                for (j = 0; j < 3; j++)
                    Pt -> Pt[j] = IritRandom(-1.0, 1.0);
            }
            else {
                for (j = 0; j < 3; j++)
                    Pt -> Pt[j] = PtList -> Pt[j] + IritRandom(-0.1, 0.1);
            }

            V = IPAllocVertex(0, NULL, PPoly -> PVertex);
            for (j = 0; j < 3; j++)
                V -> Coord[j] = Pt -> Pt[j];
            PPoly -> PVertex = V;

            IRIT_LIST_PUSH(Pt, PtList);
        }

        Crv = BspCrvInterpPts(PtList, Degree + 1,
                              NumOfDOF, CAGD_UNIFORM_PARAM, FALSE);
        CagdPtFreeList(PtList);

        CagdCrvWriteToFile3(Crv, stdout, 0, "This is from LstSqrs", &Err);

        /* Generate objects out of the geometry and set proper attrs. */
        PCrvObj = IPGenCRVObject(Crv);
        AttrSetObjectColor(PCrvObj, IG_IRIT_CYAN);

        PPolyObj = IPGenPOLYObject(PPoly);
        IP_SET_POLYLINE_OBJ(PPolyObj);
        AttrSetObjectColor(PPolyObj, IG_IRIT_YELLOW);

        /* Clear old data and display our curve and data. */
        IPSocWriteOneObject(PrgmIO, PClrObj);
        IPSocWriteOneObject(PrgmIO, PCrvObj);
        IPSocWriteOneObject(PrgmIO, PPolyObj);

        IPFreeObject(PCrvObj);
        IPFreeObject(PPolyObj);

        gets(Line);
    }
    while (Line[0] != 'q' && Line[0] != 'Q');

    IPSocDisConnectAndKill(TRUE, PrgmIO);
}

exit(0);
}
```

18.2.5 Multivariate Solver (msolve.c)

This example demonstates the use of the multivariate solver from the mvar_lib library. Input is a set of n polynomial equations in m variables and dumped out are the solutions to these equations.

```
/*****************************************************************************
 * Solves a set of polynomial equations.                                     *
*****************************************************************************
* (C) Gershon Elber, Technion, Israel Institute of Technology               *
*****************************************************************************
* Written by:  Gershon Elber                              Ver 1.0, June 2003   *
*****************************************************************************/

/* Use exmaple:  'msolve -d 1 -c "0,1,1,0,  0,1,-1,0"' */

#include "inc_irit/irit_sm.h"
#include "inc_irit/iritprsr.h"
```

```
#include "inc_irit/allocate.h"
#include "inc_irit/attribut.h"
#include "inc_irit/cagd_lib.h"
#include "inc_irit/geom_lib.h"
#include "inc_irit/mvar_lib.h"

#define SUBDIV_TOL         1e-3
#define NUMERIC_TOL         1e-8
#define MIN_DOMAIN        -2
#define MAX_DOMAIN         2

static char *CtrlStr =
    "MSlove n%-NumEqns!d v%-NumVars!d d%-MaxDeg!d c%-Coeffs!s h%-";

void main(int argc, char **argv)
{
  int i, j, Error, *Lengths,
        NumOfEqnsFlag = FALSE,
        NumOfEqns = 2,
        NumOfVarsFlag = FALSE,
        NumOfVars = 2,
        MaxDegreeFlag = FALSE,
        MaxDegree = 2,
        CoefFlag = FALSE,
        HelpFlag = FALSE;
    char *Coef,
        /*        (x-1)^2 + y^2 = 2,      (x+1)^2 + y^2 = 2. */
        *Coefs = "-1,-2,1,0,0,0,1,0,0,  -1,2,1,0,0,0,1,0,0";
    MvarMVStruct **MVs, *MVTmp;
    MvarPtStruct *Pts, *Pt;
    MvarConstraintType *Constraints;
    MvarZeroPrblmSpecStruct ZeroProblemSpec;

    Coefs = IritStrdup(Coefs);               /* Do not process static data. */

    if ((Error = GAGetArgs(argc, argv, CtrlStr,
                           &NumOfEqnsFlag, &NumOfEqns,
                           &NumOfVarsFlag, &NumOfVars,
                           &MaxDegreeFlag, &MaxDegree,
                           &CoefFlag, &Coefs,
                           &HelpFlag)) != 0) {
        GAPrintErrMsg(Error);
        GAPrintHowTo(CtrlStr);
        exit(1);
    }

    if (HelpFlag) {
        GAPrintHowTo(CtrlStr);
        exit(0);
    }

    printf("Processing %d equations of max degree %d, and %d variables,\nCoefs=\"%s\"\n",
           NumOfEqns, MaxDegree, NumOfVars, Coefs);

    /* Create the polynomial constraints as multivariates. */
    MVs = (MvarMVStruct **) IritMalloc(sizeof(MvarMVStruct *) * NumOfEqns);
    Lengths = (int *) IritMalloc(sizeof(int) * NumOfVars);
    for (i = 0; i < NumOfVars; i++)
        Lengths[i] = MaxDegree + 1;
    Constraints = (MvarConstraintType *) IritMalloc(sizeof(MvarConstraintType)
                                                    * NumOfEqns);
    for (i = 0; i < NumOfEqns; i++)
        Constraints[i] = MVAR_CNSTRNT_ZERO;
    Coef = strtok(Coefs, ",");

    for (i = 0; i < NumOfEqns; i++) {
        IrtRType *p;

        MVs[i] = MvarMVNew(NumOfVars, MVAR_POWER_TYPE,
```

```
                        CAGD_PT_E1_TYPE, Lengths);
        p = MVs[i] -> Points[1];

        for (j = 0; j < pow((MaxDegree + 1), NumOfVars); j++, p++) {
            if (sscanf(Coef, "%lf", p) != 1) {
                fprintf(stderr, "Error in parsing the coefficients string\n");
                exit(1);
            }
            Coef = strtok(NULL, ",");
        }

        MvarDbg(MVs[i]);

        MVTmp = MvarCnvrtPwr2BzrMV(MVs[i]);
        MvarMVFree(MVs[i]);
        MVs[i] = MVTmp;

        MvarDbg(MVs[i]);

        /* Set the domain of the multivariate. */
        for (j = 0; j < NumOfVars; j++) {
            MVTmp = MvarMVRegionFromMV(MVs[i], MIN_DOMAIN, MAX_DOMAIN, j);
            MvarMVFree(MVs[i]);
            MVs[i] = MVTmp;
        }

        MVTmp = MvarCnvrtBzr2BspMV(MVs[i]);
        MvarMVFree(MVs[i]);
        MVs[i] = MVTmp;

        for (j = 0; j < NumOfVars; j++) {
            BspKnotAffineTransOrder2(MVs[i] -> KnotVectors[j],
                                     MVs[i] -> Orders[j],
                                     MVs[i] -> Orders[j] + MVs[i] -> Lengths[j],
                                     MIN_DOMAIN, MAX_DOMAIN);
        }

        MvarDbg(MVs[i]);
    }

    MVAR_MVS_ZERO_INIT_PROBLEM_SPEC(ZeroProblemSpec, MVs, Constraints,
                                    NumOfEqns, SUBDIV_TOL, NUMERIC_TOL, 0.0);
    Pts = MvarMVsZerosOD(&ZeroProblemSpec);
    for (Pt = Pts; Pt != NULL; Pt = Pt -> Pnext) {
        printf("(");
        for (i = 1; i <= NumOfVars; i++)
            printf("%s%.13lg", i > 1 ? ", " : "", Pt -> Pt[i-1]);
        printf(")\n");
    }

    MvarPtFreeList(Pts);
    for (i = 0; i < NumOfEqns; i++)
        MvarMVFree(MVs[i]);
    IritFree((VoidPtr) MVs);
    IritFree((VoidPtr) Constraints);
    IritFree((VoidPtr) Lengths);

    exit(0);
}
```

18.2.6 Compute Area of a Polygonal Model (polyarea.c)

Here is a simple program to compute the total area of all polygons in the given data file. The program expects one argument on the command line which is the name of the file to read, and it prints out one line with the total computed area.

```
/*****************************************************************************
* Compute the area of a polyhedra object.                                    *
*****************************************************************************
```

```
* (C) Gershon Elber, Technion, Israel Institute of Technology                *
*****************************************************************************
* Written by:  Gershon Elber                                 Ver 1.0, June 1995   *
*****************************************************************************/

#include "inc_irit/irit_sm.h"
#include "inc_irit/iritprsr.h"
#include "inc_irit/allocate.h"
#include "inc_irit/geom_lib.h"

void main(int argc, char **argv)
{
    int Handler;

    if (argc == 2) {
        if ((Handler = IPOpenDataFile(argv[1], TRUE, TRUE)) >= 0) {
            IPObjectStruct
                *PObj = IPGetObjects(Handler);

            /* Done with file - close it. */
            IPCloseStream(Handler, TRUE);

            /* Process the geometry - compute the accumulated area. */
            if (IP_IS_POLY_OBJ(PObj) && IP_IS_POLYGON_OBJ(PObj))
                fprintf(stderr, "Area of polyhedra is %lf\n",
                        GMPolyObjectArea(PObj, TRUE));
            else
                fprintf(stderr, "Read object is not a polyhedra.\n");

            IPFreeObject(PObj);
        }
        else {
            fprintf(stderr, "Failed to open file \"%s\"\n", argv[1]);
            exit(1);
        }
    }
    else {
        fprintf(stderr, "Usage: PolyArea geom.itd\n");
        exit(2);
    }

    exit(0);
}
```

18.2.7 Converts a Freeform Surface into Polygons (polygons.c)

This true filter reads a single surface from stdin and dumps out a polygonal approximation of it to stdout. They are several parameters that controls the way a surface is approximated into polygons and in this simple filter they are being held fixed in a set of integer variables.

```
/*****************************************************************************
* Read a surface and dump out a tesselated version of it.                    *
*****************************************************************************
* (C) Gershon Elber, Technion, Israel Institute of Technology                *
*****************************************************************************
* Written by:  Gershon Elber                                 Ver 1.0, June 1995   *
*****************************************************************************/

#include "inc_irit/irit_sm.h"
#include "inc_irit/cagd_lib.h"
#include "inc_irit/iritprsr.h"
#include "inc_irit/ip_cnvrt.h"
#include "inc_irit/allocate.h"

void main(int argc, char **argv)
{
    int Handler,
        FourPerFlat = TRUE, /* Settable parameters of IritSurface2Polygons. */
        FineNess = 20,
```

```
        ComputeUV = FALSE,
        ComputeNrml = FALSE,
        Optimal = FALSE;

    if ((Handler = IPOpenDataFile("-", TRUE, TRUE)) >= 0) {
        IPObjectStruct
            *PObj = IPGetObjects(Handler);

        /* Done with file - close it. */
        IPCloseStream(Handler, TRUE);

        /* Process the surface into polygons. */
        if (IP_IS_SRF_OBJ(PObj)) {
            IPPolygonStruct
                *PPoly = IPSurface2Polygons(PObj -> U.Srfs, FourPerFlat,
                                            FineNess, ComputeUV,
                                            ComputeNrml, Optimal);
            IPObjectStruct
                *PObjPoly = IPGenPOLYObject(PPoly);

            IPStdoutObject(PObjPoly, FALSE);

            IPFreeObject(PObjPoly);
        }
        else
            fprintf(stderr, "Read object is not a surface.\n");

        IPFreeObject(PObj);
    }
    else {
        fprintf(stderr, "Failed to read from stdin\n");
        exit(1);
    }

    exit(0);
}
```

18.2.8 Linear Transformations' Filter (transfrm.c)

This little more complex program transfroms all the geometry in the read data which can be any number of files according to the specified transformations on the command line. The command line is parsed via **GAGetArgs** and its associated functions. The transformation matrix is then computed with the aid of the matrix package and applied to the read geometry at once.

```
/*****************************************************************************
* Transforms input stream following command line specs.  This examples also  *
* show how to read from input stream call back (in-memory).                   *
*****************************************************************************
* (C) Gershon Elber, Technion, Israel Institute of Technology                *
*****************************************************************************
* Written by:  Gershon Elber                                Ver 1.0, June 1995   *
*****************************************************************************/

#include "inc_irit/irit_sm.h"
#include "inc_irit/allocate.h"
#include "inc_irit/iritprsr.h"
#include "inc_irit/geom_lib.h"
#include "inc_irit/misc_lib.h"

static char *CtrlStr =
#ifdef IRIT_DOUBLE
    "Transfrm x%-Degs!F y%-Degs!F z%-Degs!F t%-X|Y|Z!F!F!F s%-Scale!F h%- DFiles!*s";
#else
    "Transfrm x%-Degs!f y%-Degs!f z%-Degs!f t%-X|Y|Z!f!f!f s%-Scale!f h%- DFiles!*s";
#endif /* IRIT_DOUBLE */

static int ReadStreamCallBackIO(int c);

void main(int argc, char **argv)
```

```
{
    int NumFiles, Error,
        RotXFlag = FALSE,
        RotYFlag = FALSE,
        RotZFlag = FALSE,
        TransFlag = FALSE,
        ScaleFlag = FALSE,
        HelpFlag = FALSE;
    char **FileNames;
    IrtRType RotXDegrees, RotYDegrees, RotZDegrees, TransX, TransY, TransZ,
        Scale;
    IrtHmgnMatType Mat1, TransMat;
    IPObjectStruct *PObjs, *PObjsTrans, *PObj;

    if ((Error = GAGetArgs(argc, argv, CtrlStr,
                           &RotXFlag, &RotXDegrees,
                           &RotYFlag, &RotYDegrees,
                           &RotZFlag, &RotZDegrees,
                           &TransFlag, &TransX, &TransY, &TransZ,
                           &ScaleFlag, &Scale,
                           &HelpFlag,
                           &NumFiles, &FileNames)) != 0) {
        GAPrintErrMsg(Error);
        GAPrintHowTo(CtrlStr);
        exit(1);
    }

    if (HelpFlag) {
        fprintf(stderr, "This is Transform...\n");
        GAPrintHowTo(CtrlStr);
        exit(0);
    }

    /* Construct the transformation matrix: */
    MatGenUnitMat(TransMat);
    if (RotXFlag) {
        MatGenMatRotX1(IRIT_DEG2RAD(RotXDegrees), Mat1);
        MatMultTwo4by4(TransMat, TransMat, Mat1);
    }
    if (RotYFlag) {
        MatGenMatRotY1(IRIT_DEG2RAD(RotYDegrees), Mat1);
        MatMultTwo4by4(TransMat, TransMat, Mat1);
    }
    if (RotZFlag) {
        MatGenMatRotZ1(IRIT_DEG2RAD(RotZDegrees), Mat1);
        MatMultTwo4by4(TransMat, TransMat, Mat1);
    }
    if (TransFlag) {
        MatGenMatTrans(TransX, TransY, TransZ, Mat1);
        MatMultTwo4by4(TransMat, TransMat, Mat1);
    }
    if (ScaleFlag) {
        MatGenMatUnifScale(Scale, Mat1);
        MatMultTwo4by4(TransMat, TransMat, Mat1);
    }

    /* Get all the data from all the input files. */
    if (NumFiles == 0) {
        /* Test reading data from memory... */
        int Handler = IPOpenStreamFromCallBackIO(ReadStreamCallBackIO,
                                                 NULL, IP_ITD_FILE,
                                                 TRUE, FALSE);

        PObjs = IPGetObjects(Handler);

        IPCloseStream(Handler, TRUE);
    }
    else {
        PObjs = IPGetDataFiles(FileNames, NumFiles, TRUE, TRUE);
```

```
    }

    /* Apply the transformation to all geometry in input file(s) and dump */
    /* the transformed geometry to stdout.                                */

    PObjsTrans = GMTransformObjectList(PObjs, TransMat);
    for (PObj = PObjsTrans; PObj != NULL; PObj = PObj -> Pnext)
        IPStdoutObject(PObj, FALSE);

    IPFreeObjectList(PObjs);
    IPFreeObjectList(PObjsTrans);

    exit(0);
}

/*****************************************************************************
* DESCRIPTION:                                                               *
*   A call back function to read IRIT data in directly from memory.  This    *
* function will return the next char read for memory.                        *
*                                                                            *
* PARAMETERS:                                                                *
*   c:   Ignored.                                                            *
*                                                                            *
* RETURN VALUE:                                                              *
*   char:     Next character from memory stream.                             *
*****************************************************************************/
static int ReadStreamCallBackIO(int c)
{
    IRIT_STATIC_DATA int
        LineNum = 0,
        CharNum = 0;
    IRIT_STATIC_DATA char
        *Data[] = {
            "[OBJECT [COLOR 11] [RGB \"55, 255, 255\"] [WIDTH 0.05] SADDLE",
            "    [SURFACE BEZIER 3 3 E3",
            "        [0 0 0]",
            "        [0.05 0.2 0.1]",
            "        [0.1 0.05 0.2]",
                "",
            "        [0.1 -0.2 0]",
            "        [0.15 0.05 0.1]",
            "        [0.2 -0.1 0.2]",
                "",
            "        [0.2 0 0]",
            "        [0.25 0.2 0.1]",
            "        [0.3 0.05 0.2]",
            "    ]",
            "]",
            "[OBJECT [COLOR 2] [DWidth 4] FINAL",
            "    [POLYGON [PLANE 1 0 0 0.5] 4",
            "        [[NORMAL 1 0 1] -0.5 2 1]",
            "        [[NORMAL 1 1 0] [INTERNAL] -0.5 1 1]",
            "        [[NORMAL 1 0 2] -0.5 1 -1]",
            "        [[NORMAL 1 0 0] -0.5 2 -1]",
            "    ]",
            "]",
            NULL
        };

    /* End of line or end of memory stream!? */
    while (Data[LineNum] == NULL || Data[LineNum][CharNum] == 0) {
        if (Data[LineNum] == NULL)                          /* Last line. */
            return EOF;
        else {
            LineNum++;
            CharNum = 0;
#            ifdef DEBUG_ECHO_INPUT
               putchar('\n');
               putchar('\r');
```

```
#             endif /* DEBUG_ECHO_INPUT */
            return ' ';
        }
    }

#   ifdef DEBUG_ECHO_INPUT
        putchar(Data[LineNum][CharNum]);
#   endif /* DEBUG_ECHO_INPUT */

    return (char) Data[LineNum][CharNum++];
}
```

Here is the result of running 'transfrm -h':

```
This is Transform...
Usage: Transfrm [-x Degs] [-y Degs] [-z Degs] [-t X Y Z] [-s Scale] [-h] DFiles
```

When you are considering the usefulness of this tool remember that the transformations are applied to the geometry in an internal order which is different from the command line order. That is,

```
transfrm -x 30 -y 30 geometry.dat > tgeometry.dat
```

will compute the exact same transfrom as,

```
transfrm -y 30 -x 30 geometry.dat > tgeometry.dat
```

This, while rotation is not a commutative operation. Nonetheless, you may split the operations. That is:

```
transfrm -y 30 geometry.dat | transfrm -x 30 - > tgeometry.dat
```

or

```
transfrm -x 30 geometry.dat | transfrm -y 30 - > tgeometry.dat
```

will do exactly what one expects.

Index