3D Printable Word Cloud

Project by: Dror Dahan

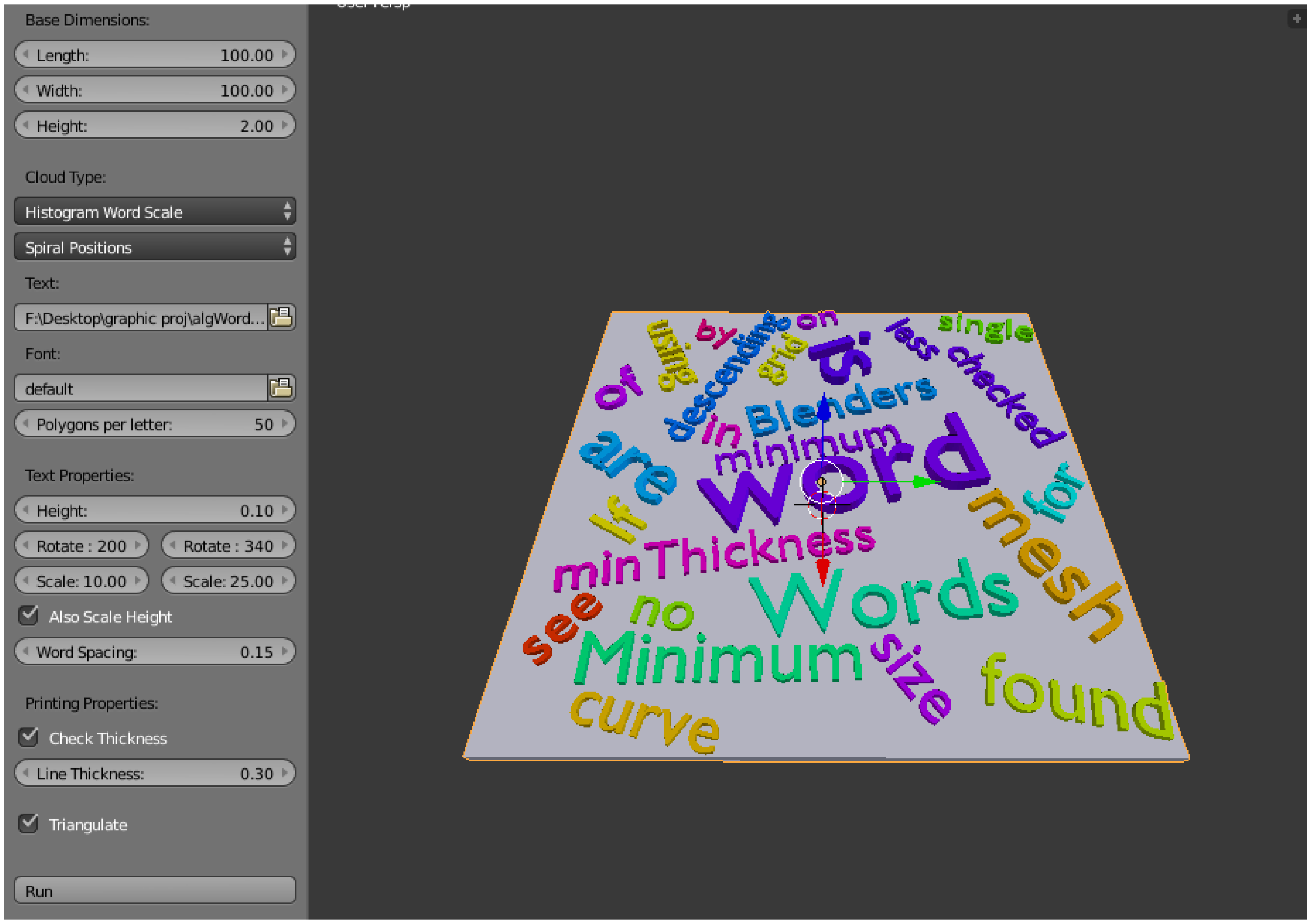
Instructed by: Prof. Mirela Ben Chen

About

The project goal is to implement a word cloud generator in 3D while taking in parameters such as "line thickness" so that the final result is suitable for 3D printing.

It is implemented as a Blender plugin and written in Python.
It has a comfortable user interface for setting a large number of parameters and running the algorithm.

The final result can then be exported for 3D printing in various formats:
.dae .abc .3ds .fbx .ply .obj .x3d .stl



Base Dimensions:
Length: 100.00
Width: 100.00
Height: 2.00

Cloud Type:
Histogram Word Scale
Spiral Positions

Text:
F:\Desktop\graphic proj\algWord...

Font:
default

Polygons per letter: 50

Text Properties:
Height: 0.10
Rotate : 200 | Rotate : 340
Scale: 10.00 | Scale: 25.00
☑ Also Scale Height
Word Spacing: 0.15

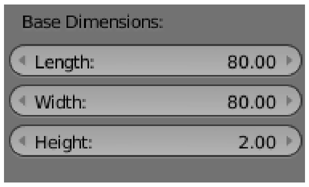
Printing Properties:
☑ Check Thickness
Line Thickness: 0.30

☑ Triangulate

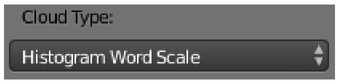
Run

<u>Parameters and Considerations</u>

(for installation instructions see install.pdf)

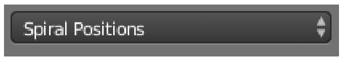

Base Dimensions:
Length: 80.00
Width: 80.00
Height: 2.00

Control dimensions of base box.

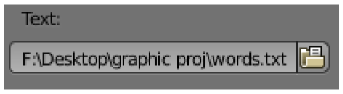

Cloud Type:
Histogram Word Scale

<u>Histogram Word Scale</u> – Word scale is relative to the number of times each word appears in the text.

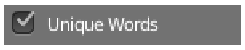
<u>Random Word Scale</u> – Word scale is random.


Spiral Positions

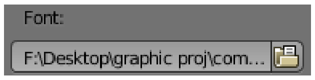
<u>Spiral Positions / Grid Positions</u> – Select array of positions that will be used to position words on base box.
(See below for visualization)


Text:
F:\Desktop\graphic proj\words.txt

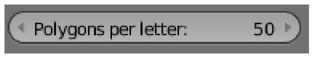
Input text file.


☑ Unique Words

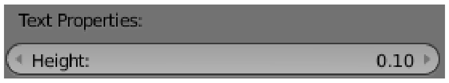
When using “Random Word Scale” it is possible to only read unique words or also have duplicates.


Font:
F:\Desktop\graphic proj\com…

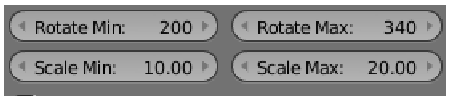
Select font file.


Polygons per letter: 50

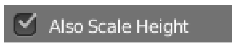
Average polygon budget per letter.
Should be higher for fonts with high detail.


Text Properties:
Height: 0.10

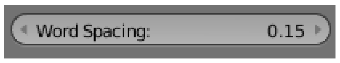
Extrusion height for words.


Rotate Min: 200 Rotate Max: 340
Scale Min: 10.00 Scale Max: 20.00

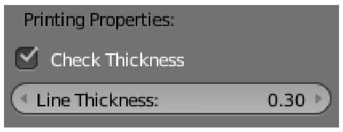
Range of rotation and scaling for words
(Rotation is always random in the range).


☑ Also Scale Height

Select whether to apply scaling in z axis or keep uniform word height.


Word Spacing: 0.15

Control spacing between words.

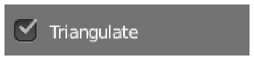

Printing Properties:

✓ Check Thickness

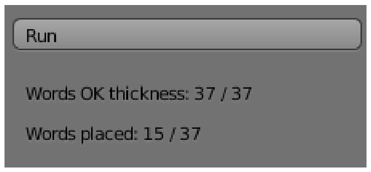
Line Thickness: 0.30

"Check thickness” will calculate min thickness for each word, if the word is too thin then it will not be used.

In the case of random word scale, words may be scaled beyond their random value in order to match required thickness.


✓ Triangulate

Triangulate mesh of final result.



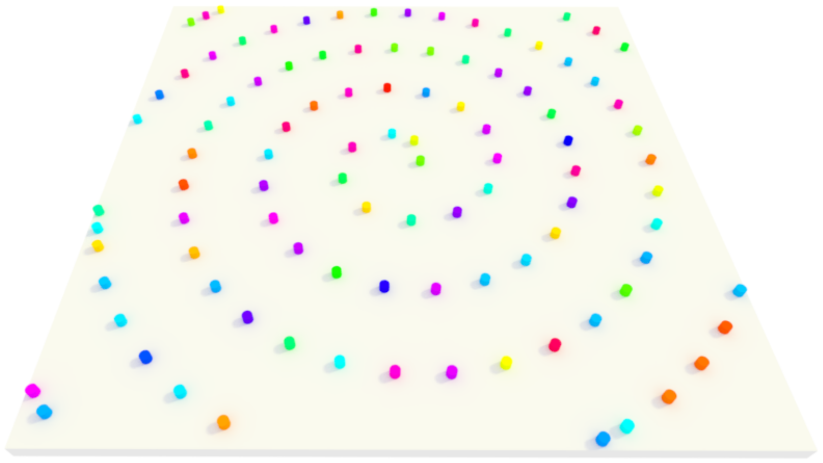
Words OK thickness: 37 / 37

Words placed: 15 / 37

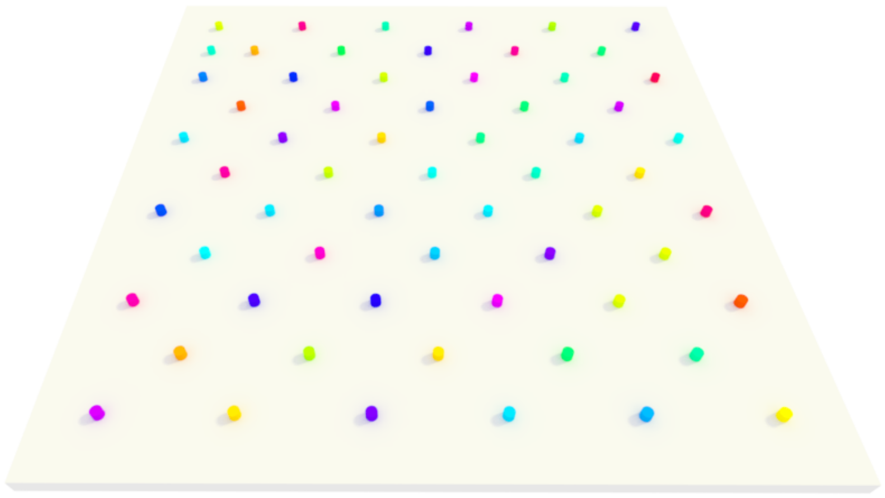
After each run some stats are displayed:

Number of words that passed thickness check.
Number of words successfully placed.

Spiral positions:



Grid Positions:



Algorithm Outline

- Add the base box mesh.

- Create array of positions and rotations (x, y, r)
  Positions depend on the chosen type: "Spiral Positions" or "Grid Positions".
  Rotations are random in range: $(rotateMin,\ rotateMax)$

- Read words from file
Words may be unique or not, depending on the chosen options.
For histogram scaling, count the number of appearances of each word.

- Add word meshes
  Create flat word text object.
  Set text font.
  Set material color with a random hue.
  Convert the word from curve object to mesh.
  Apply decimate.
  Extrude the word.

- Minimum thickness is calculated for each word (see minThickness algorithm below)

- Minimum word scale is calculated for each word.
  $wordMinScale\ =\ lineThickness\ /\ wordMinThickness$

- Words are scaled in range $(scaleMin,\ scaleMax)$
  The user can select between two options for scaling:

  1) Histogram Scaling: word scale is relative to the number of appearances in text.

  2) Random Scaling: word scale is random in range $(scaleMin,\ scaleMax)$
  but not less than $min(wordMinScale, scaleMax)$

  If chosen scale is less than $wordMinScale$ then the word is deleted.

- Words are sorted by size, descending order.

- Words are positioned.

For positioning use a displaced copy of words in order to enforce word spacing.

```
  Loop words:
      Loop positions:
          Transform word to position
          Check for intersection with previously positioned words (using BVHTree class)
          Check if the word is partially outside of base boundary.

          If one of the checks is true, then continue to next position
          Otherwise, a good position was found, continue to next word
      End loop

      No position was found, delete current word.
  End loop
```

- Words are joined together to a single mesh.

- Apply union with the base mesh.

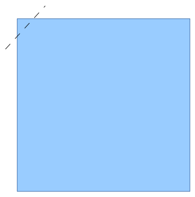
- Triangulate final result.

MinThickness Algorithm

The goal is to measure the minimum thickness of an object.
Minimum thickness value is used to decide if a word can be 3D printed or not.

First we notice that any <180° angle has minimum thickness approaching 0 and will not be printed perfectly because corners get rounded.

For example:



Obviously we don’t want all objects to return with minThickness close to 0.

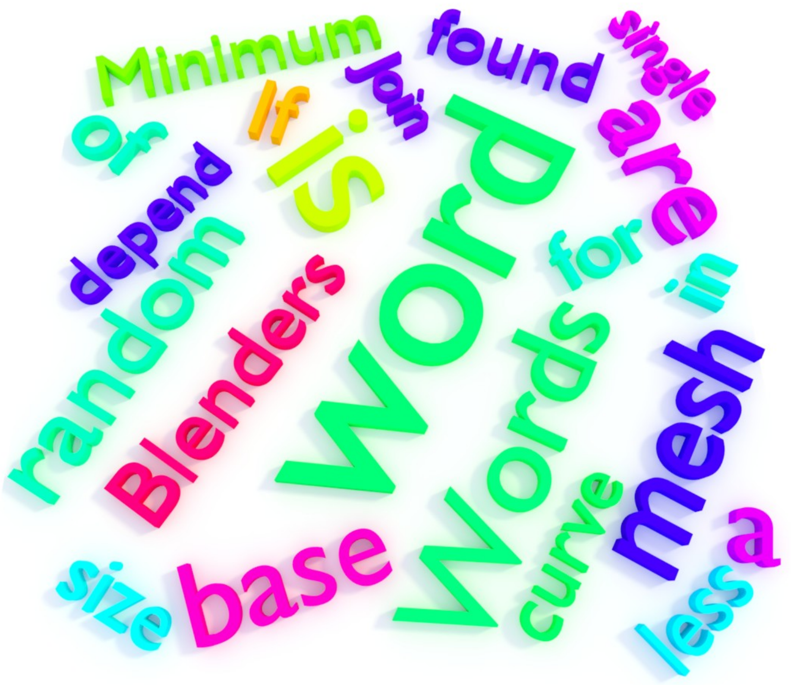
Our simple solution is to ignore thickness that is measured between adjacent faces.

Algorithm:

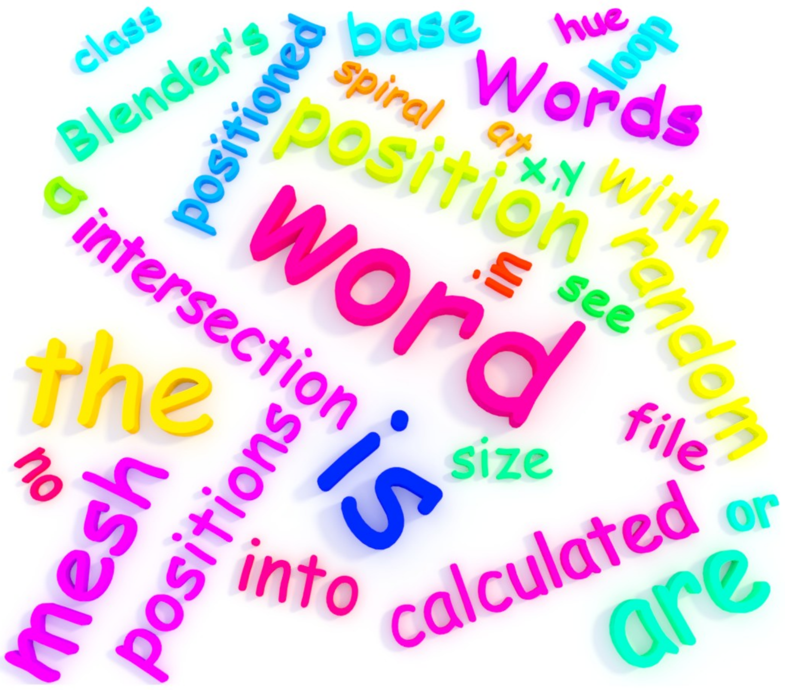
```
initialize $minThickness = \infty$
loop faces:
   set $direction = -faceNormal$
   set $direction.z = 0$
   loop vertices of face:
      cast a ray from current vertex position in direction of $direction$

      if the ray hit a non adjacent face:
         set $thickness = dist(vertex, hitLocation)$
         set $minThickness = min(minThickness, thickness)$
```

Results

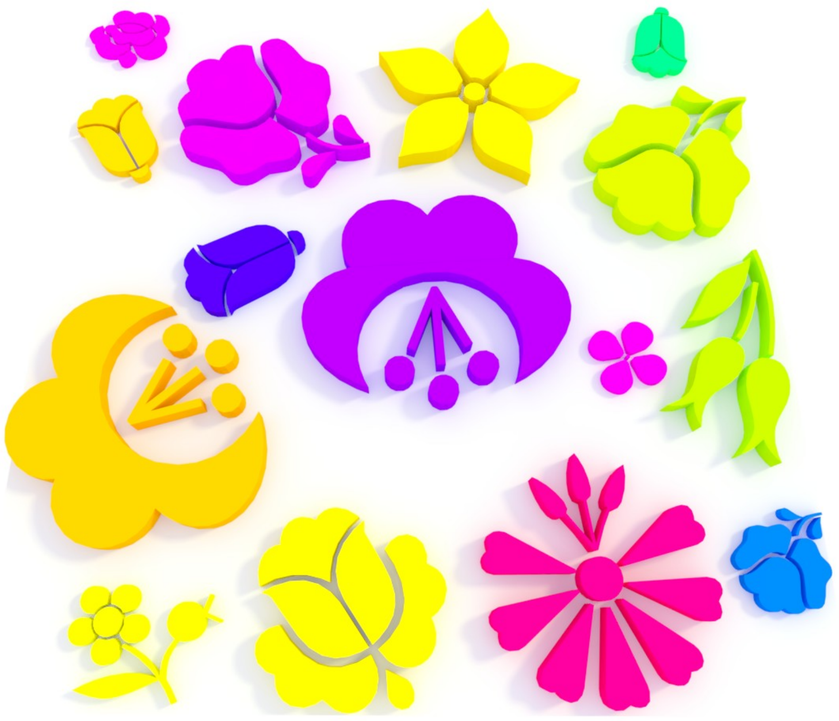
1) Using default font, histogram scaling, uniform word height.



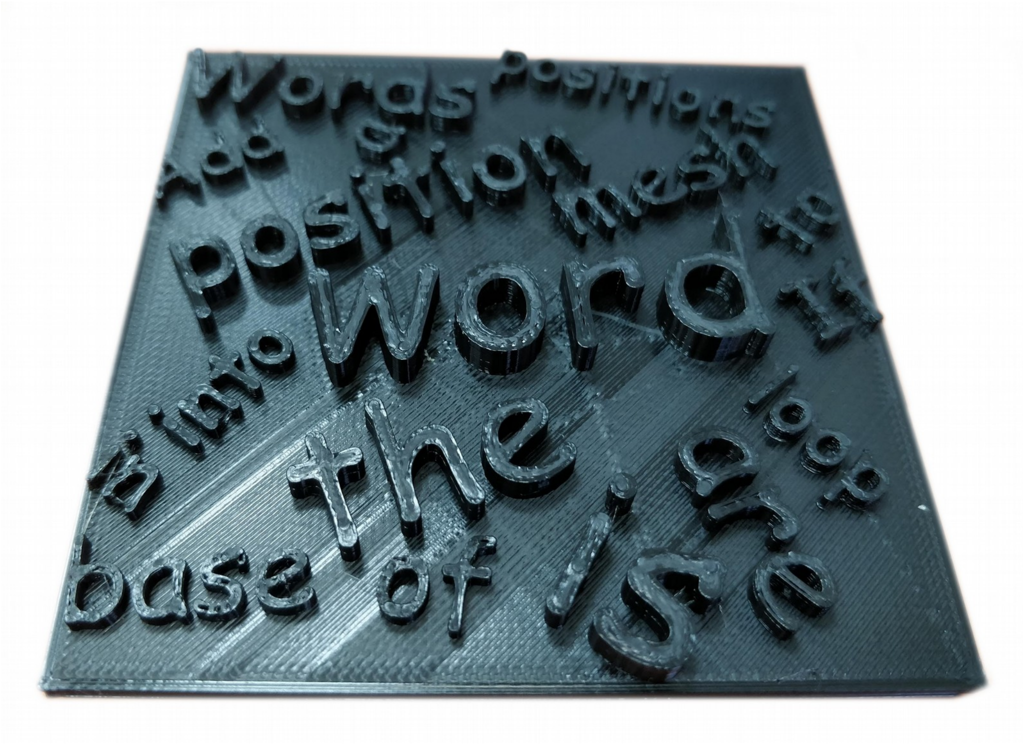
2) Using comic font, histogram scaling, also scale word height.
The comic font is more round and allows for smaller words with the same line thickness.



3) Using flowers font, random scaling mode.
Input text has one letter per word.



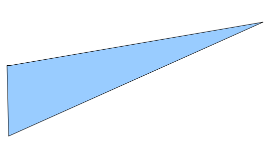
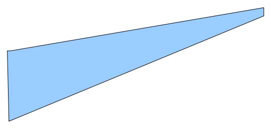
4) 3D printed result



Limitations and future work

minThickness function is not geometric

In the minThickness algorithm we ignore thickness between adjacent faces as a simple solution to allow corner rounding. This makes the result depend on meshing and not purely on geometry. For example:

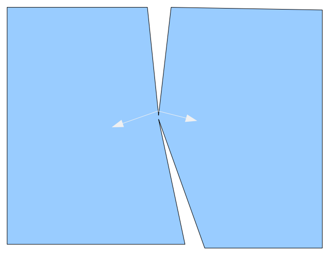



The left mesh returns $minThickness = \infty$, while the right mesh returns some small value

minThickness function does not find the exact minimum

We ray cast in $-normal$ direction for each face that a vertex belongs to.
In practical cases this method gives a good thickness estimate.

Yet it's possible to think of an edge case that would not work well
For example:



In order to increase accuracy for such edge cases, we would need to ray cast in many more directions per vertex.

Performance

For every word all positions are tested, in order, until a position is found where there is no intersection with previous words.
This may cause a slow run when trying to place a large number of words.
The same method is used in a 2D word cloud algorithm, but it tests for overlapping pixels between words which is much cheaper.

Testing intersections in 2D or a more intelligent method for choosing positions could help to improve run time.