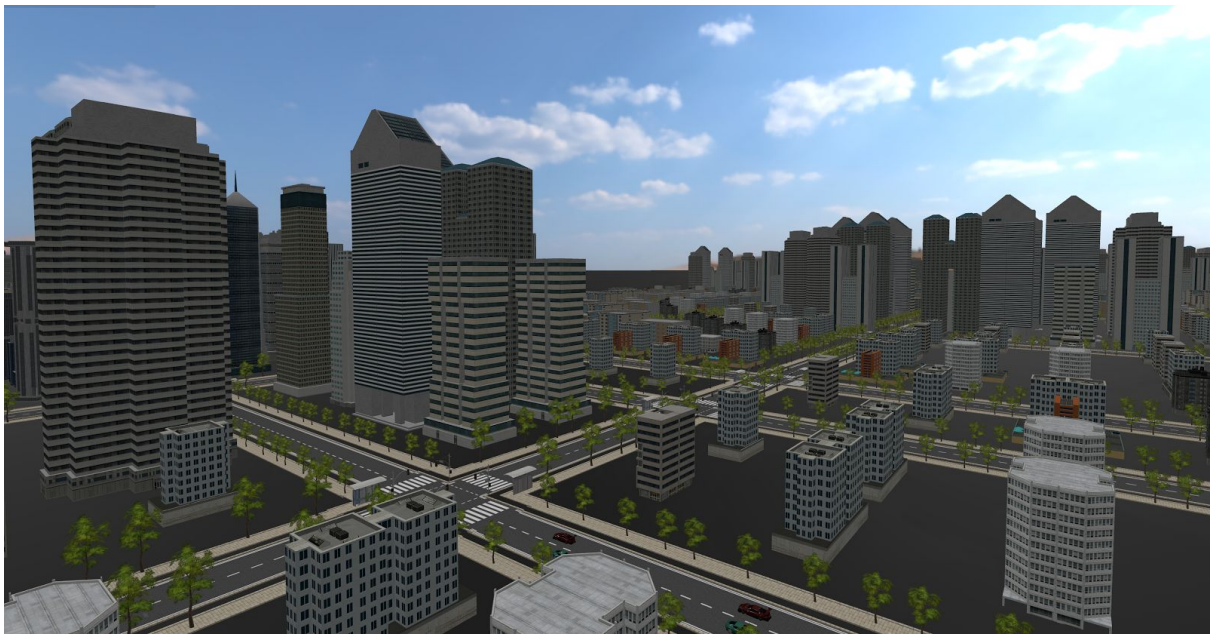


Perlin City - Procedural 3D City Generation Project

Alex Nicola, Nati Goel
Supervisors - Yaron Honen and Boaz Sternfeld

GIP & CGGC Labs, Technion – Israel Institute of Technology

Watch Trailer
VR DEMO

Visit Project Site

Abstract

Based on a known method of creating a single random city block using perlin noise, we have created a method of procedurally generating an infinite city without the intervention of human designer.
Perlin city is deterministic, realistic and beautiful.
To achieve best performance possible, we used object pooling, separation of building objects upon multiple frames using coroutines, detecting when to create and destroy and addition of detailed objects that are displayed and hidden based on distance.

Project Idea

We wanted to create an infinite city that is procedurally generated on real time with no drops in FPS. The city should feel alive with cars, people and trees and the player could use the HTC VIVE to get inside our virtual reality city.

The city is built in a deterministic manner, this allows the player to revisit the same locations on the map, making the player familiar with the place although it was randomly generated.

You can use HTC Vive to navigate the city in a virtual realty.

We developed the project using Unity3d - 5 (Version 5.6.2)

Introduction

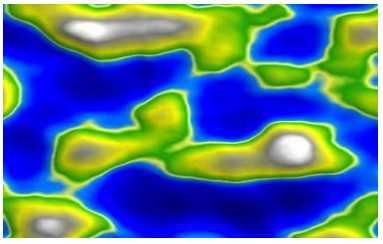
Procedural Generation

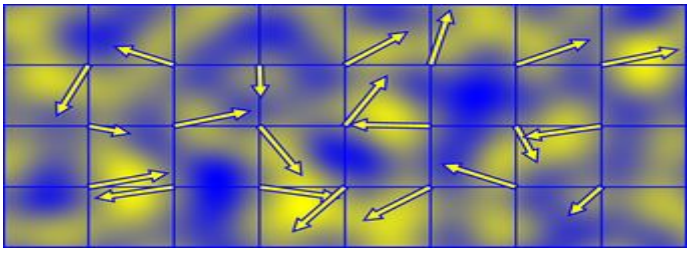
Programmatic generation of game content (rather than manual design), which enables generating unlimited amount of content, in a very short time.
In many video games, it is used to create infinite worlds, or generating different types of content using a few random parameters, for unpredictable gameplay.

Perlin Noise

A technique used to produce natural appearing textures on computer generated surfaces for motion picture visual effects.
The basic perlin noise function receives X and Y inputs, and returns a value between 0.00 and 1.00.
public **float** **perlin**(**float** x, **float** y)

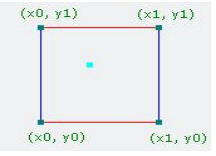
Perlin is deterministic and provides smooth transition between results of close inputs (as opposed to random noise function).
Every color represent a value on the grid.



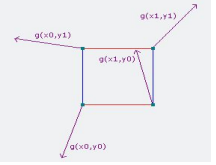


How does it work?

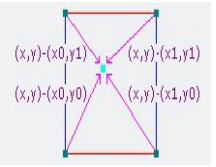
The input coordinates are translated into a point within a unit square, from a pre-allocated grid.



Each of the 4 edges of the square is assigned with a pseudo-random gradient vector.



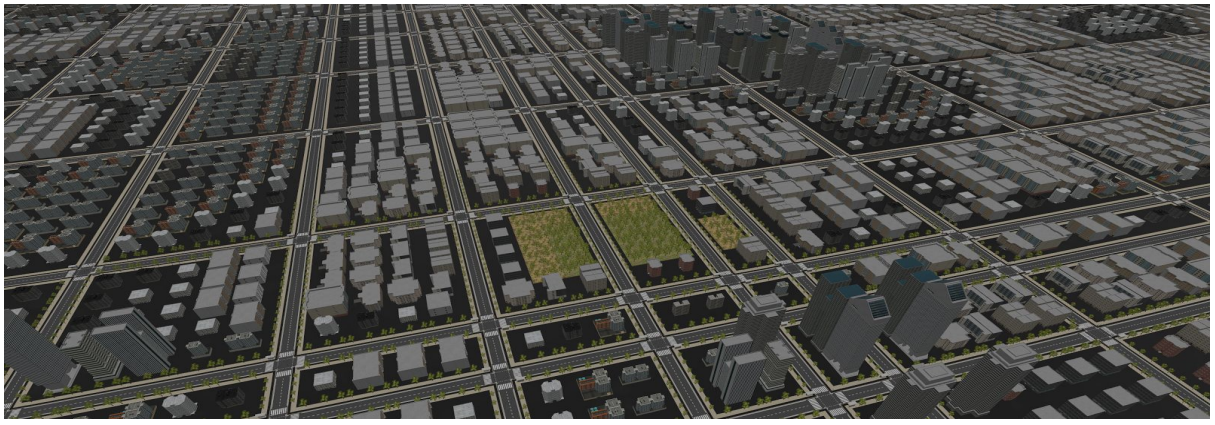
Distance vectors are calculated between the edges and the input point. Afterwards, 4 dot products are calculated between each distance vector and it's concordant gradient vector.



The return value is an interpolation between all 4 dot products.

How do we use the Perlin noise?
We have over 40 models of buildings arranged by height and type. Every value on the perlin map is mapped to a specific building and placed on the real world map. Aftewards, a grid of vertical and horizontal roads are placed on the real world.

The smooth transitions between the values help make the city appear realistic. Tall buildings appear together in a cluster and there is a smooth transition on the height and type of the buildings that are being placed.



For more details about perlin noise, please see:
https://en.wikipedia.org/wiki/Perlin_noise
http://flafla2.github.io/2014/08/09/perlinnoise.html
https://www.youtube.com/watch?v=MJ3bvCkHJtE
https://www.youtube.com/watch?v=Or19ilef4wE

Challenges

- Determinism - when the player returns to a location, the same building block should be created.
- How many to build at the same time without having FPS drops
- When to build?
- Object pooling - should we create and destroy buildings or recycle them?

Determinism

When we sample values from the perlin map to decide how to build a building block, we take the block's coordinates into consideration.
This way when the player returns to the same spot, the same area from the perlin map is sampled and the same buildings and roads are created.

We are also considering a seed parameter, which is randomly determined at the beginning of the run, in order to get a different city map each time.

How many to build at the same time?

We use a system of Blocks. A Block is a square grid with buildings, parks and roads that has a predefined number of objects in it. Instead of building all the objects at the same frame(causing this frame to take 2-3 seconds to be rendered) we split the objects on many frames using coroutines. This way we achieve a seamless building flow without stopping the rendering of the frames.

When to build?

When player reaches a threshold distance from the borders of a block, we build a block in front of him. When player is getting too far from a block (usually behind him), the block will be destroyed.

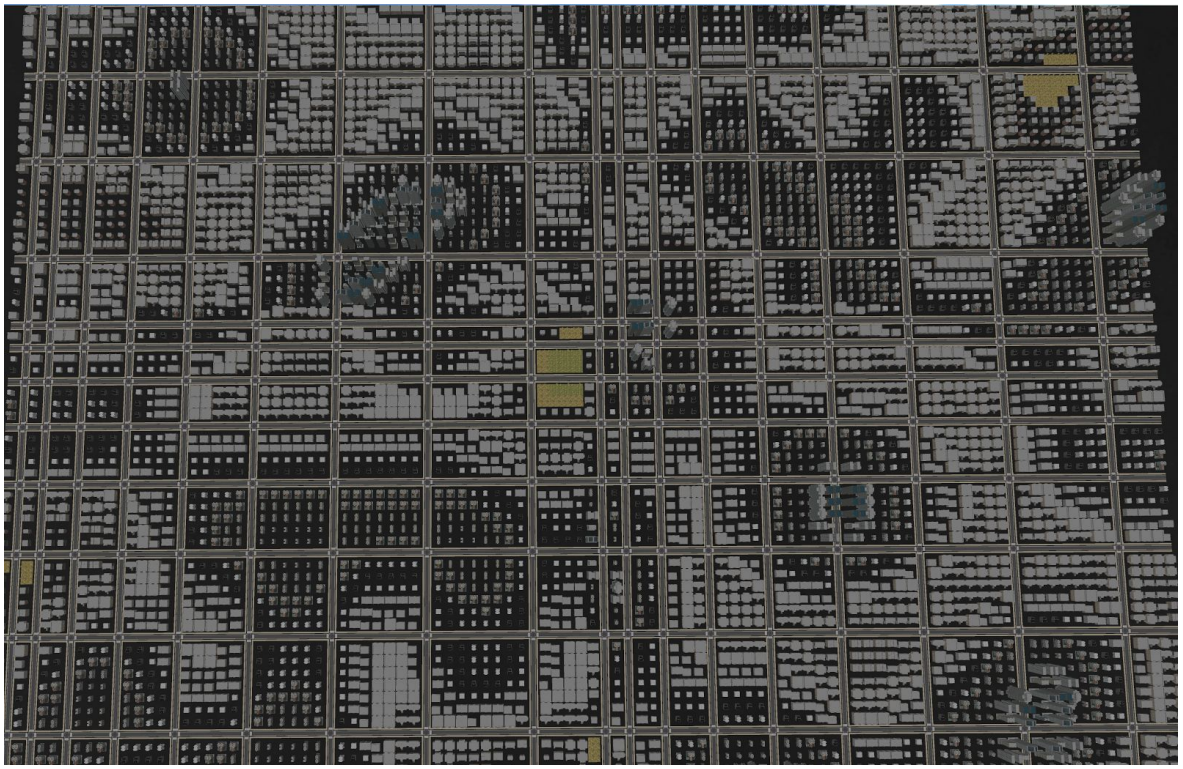
Object pooling

The object pool pattern is a software creational design pattern that uses a set of initialized objects kept ready to use – a "pool" – rather than allocating and destroying them on demand. A client of the pool will request an object from the pool and perform operations on it. When the client finish with the object, it will be returned to the pool instead of being destroyed. This saves costly allocating and releasing of memory actions.

Results of our algorithm, a comperation of real city vs ours:

Real city - BUENOS AIRES

Perlin City - Our generated city has the same patterns



Realism

We added walking people and cars that only appear when you are in a certain distance.

Trees also appear with a higher threshld of distance.

Every crossroad has traffic lights and bus stops.

Trees, cars and people only appear when you are close enough to them.

City background sound is heard only when getting closer to the ground.

Summary

Our goal was to implement procedural generation of a deterministic and realistic city. After some research, we found out about perlin noise - a very effective technique, used in many procedural content generation algorithms.

During our work, we faced mainly performance related difficulties, which we solved using Object Pooling, building/destroying limited amount of objects on each frame and changing object's level of detail based on the player's position.

We tried to parameterize our algorithm as much as possible, in order to examine the effect of each parameter, and to achieve best performance by tuning the different parameters.

Using Unity editor, it is possible to configure: building block size, block radius triggering the building of new blocks, division factor applied on perlin noise, object pool size, limit number of objects to build/destroy on each frame and the threshold of blocks destroying distance from player.
We have also added a possibility to disable the use of object pooling, for comparison.
At the beginning of each run, the object pool is allocated, this is a very heavy action which cause low fps rate for a few seconds.

The techniques we used was proven to be effective, and we are very happy with the results.

For demonstration, we've implemented 3 possible ways to tour our city - Freelock camera, a jet aircraft and Virtual reality flying using HTC Vive.

What's next?

Possible improvements and extentions to our project:

- Add interaction between the user and other characters in the scene.
- Implement a more complex road system.
- Add terrains separating between cities.
- Build Video games taking place in Perlin city (e.g car racing game).
- Driving/Flight simulator implemented in unity can be easily integrated in our project.

Sources

Procedural generation
https://www.youtube.com/watch?v=TgbuWfGeG2o

Object Pooling and corutines
https://unity3d.com/learn/tutorials/topics/scripting/object-pooling
https://unity3d.com/learn/tutorials/topics/2d-game-creation/recycling-obstacles-object-pooling
http://answers.unity3d.com/questions/795494/create-gameobjects-in-threads.html
https://www.youtube.com/watch?v=bM3CXzj5xM4

Content generation using perline noise
https://www.youtube.com/watch?v=xkuniXI3SEE
http://www.redblobgames.com/maps/terrain-from-noise/