Kulprawee Prayoonsuk

CSCI 5565 - Introduction to Computer Graphics

05/11/20

What's all that noise?

Introduction

Terrain rendering is a crucial part of many virtual mediums such as virtual art and video game. In order to create a more interesting and unique experience for users many video games have elected to use procedural terrain generation tools in order to algorithmically produce it's setting. This innovation not only help reduce the time-consuming process of terrain design and modeling, but also remove the limit of terrain size and details while allowing user to experience unique environment each time. This process is generally done by using an algorithm to generate a matrix of value and then using those value to render an environment. The value is generally the height value of each square of the terrain. Variety of algorithms for terrain value exists and are use for different purposes, in this project we will focus on the Perlin noise and the Simplex noise algorithm on a basic natural terrain consist of hill and valley.

Benefits of procedurally generated terrain in video game

Create setting with larger scale with ease

If the setting is created manually each element of the scene is fixed and texture, model and lighting must be custom-made to that setting. By procedurally creating the terrain you can have an infinite world that re not constraint to a fixed design. With that however could result in an increase in computing power as preloaded map and texture cost far less to compute.

Lower to cost of production

By not having to custom made each environment we can significantly cut down on time take to produce each environment along with other artistic assets such as texture as those can also be made to be slightly differ with similar methods. Although this also help cut the time and cost of quality control, it also may take out the finer details and creative touch that human can apply.

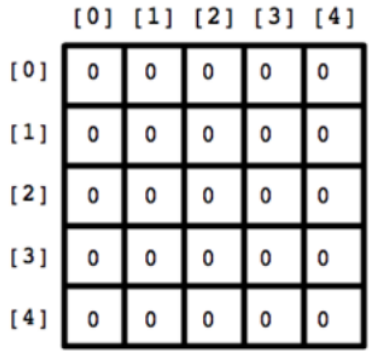
Increase variety and replyability

Most game contain linear story which after experiencing it once the user will have already known where everything will be which does not give them much reason to try to experience it again. By not having a fixed environment this allow user to experience something new each time they interact with the environment.

Backgrounds

Heightmap

Heightmap is an array, generally 2D array, use to represent the coordinate of the environment. Each index of the array then contains a value which represent the height of that coordinate.



Perlin noise

Perlin noise was developed in 1983 by Ken Perlin out of the desire to create a more realistic noise for visualization. This is because human is able to tell if noise look “too random” and thus will not appear natural. Perlin noise allow use to generate a pseudo-random appearance while still able to have control over some variable. Currently Perlin noise is widely use in generating synthetic texture for visual element such as water, cloud, smoke, surfaces etc.

If one were to plot the point that were generated using a generic random methods included in most library, you would get a graph that is similar to the one right (Figure 1.6) while Perlin noise would achieve a smoother curve similar to the one on the left (Figure 1.5). This is because in the regular random, there are no coloration between any of the points thus resulting in this jerky motion. However, with the Perlin noise there exist a relationship between the current points and its previous point, however it is still “random” because we have no idea in which direction will the curve continue which give it unpredictable trend. This allow for a more "control chaos" Allow us to create randomness that is smooth and have a more organic quality to it, which is often the property of thing in the natural world.

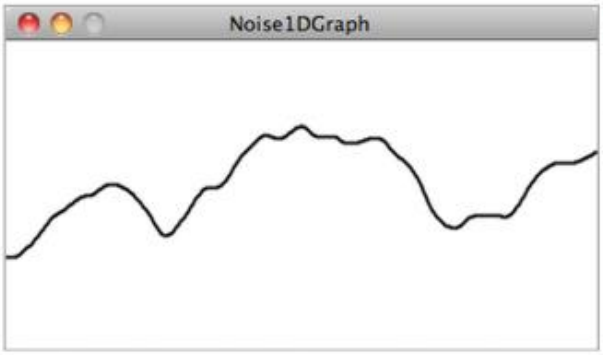


Figure I.5: Noise

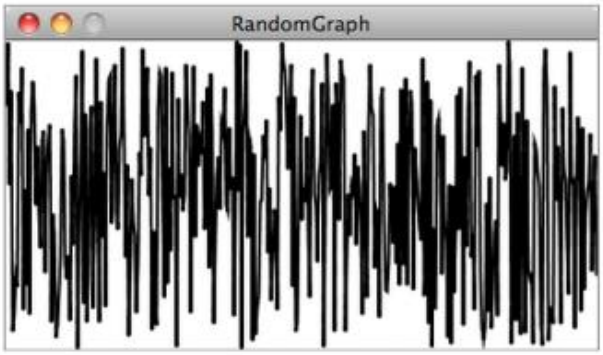


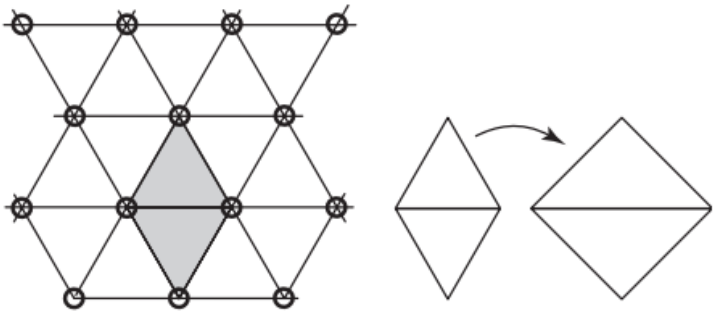
Figure I.6: Random

Once you have a pseudo-random gradient at regularly spaced point, a point is then associated with integer coordinate. For a given point between two integer point the value is then interpolate (to estimate within two known values in a sequence of values) between the two point.

Included in the project folder are visual representation of a ball moving on a plane where the position is generated using regular random vs Perlin noise. This example required Processing3 to run.

Simplex noise

Simplex noise is an improvement to the Perlin noise. The main different being that Simplex utilize the Simplex grid in their calculation. The idea of simplex grid is for a given dimension, choose the most compact shape that can be repeat to fit into the original space. For example, you can fit multiple triangles together in such a way that it produces a square.



Simplex grid

The reason that simplex shape is use is because they contain fewer corners than a hypercube (an example is a square in 2D or a cube in 3D) making it more convenient to interpolate values base on its corners. Once the values are interpolated a straight summation of multiplication of the extrapolation of the gradient ramp and a radially symmetric attenuation function from each corner is apply.

Included in the project folder are animated visual representation of Perlin noise vs Simplex Noise This example required Processing3 to run.

Implementation

Heightmap generation

For the height map generation I populated a 2D vector with a value randomly generated using regular random, Perlin noise and Simplex noise and use those store value to set the color of the corresponding coordinate's pixel to the value of the grey scale via flColor3f function. The grid is then displayed to give us a heightmap. First, we loop through each coordinate position then go through series of interpolation using the Hermite blending function: 3t^2 - 2t^3 then loop through the depth to apply the value of frequency. As for the Simplex noise algorithm I apply the algorithm from open source code and then rescale the value to fit in our range of float 0 to 1 by re scaling the value by percentage. A hash map is recommended for both noises according to the paper by Stefan Gustavsons instead of generating hash table is likely due to memory and efficiency concern.

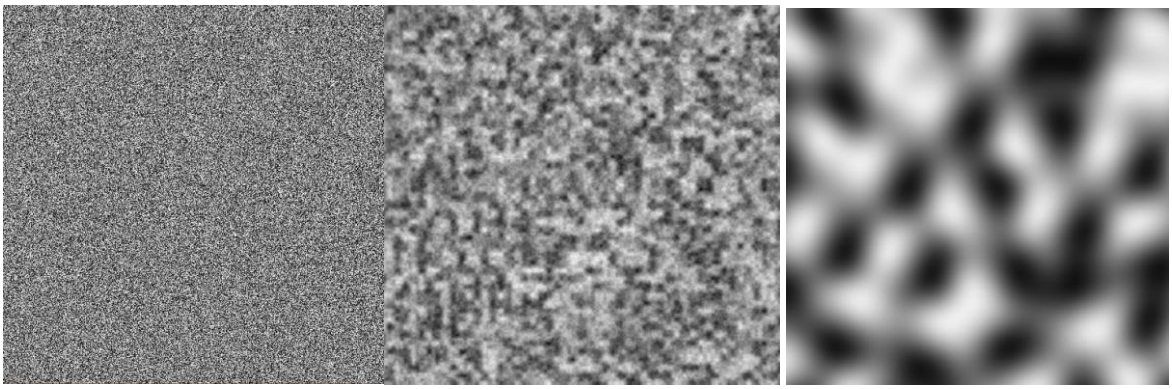
The value of the parameter can be change in order to get varying heightmap such as value of scaling factor of the x,y coordinate value. The scaling is necessary because both Perlin and Simplex noise algorithm require coordinate that are closer together and using a whole integer leap will be too large for the algorithm to take into account it's neighboring values. For Perlin noise value of frequency of noise function being apply and depth, which is the number of samples to generate can be adjust in order to get 'smoother' look.

Terrain Generation

The terrain Generation is implemented in Java Processing 3 due to lack of time to implemented in OpenGl, base on "3D Terrain Generation with Perlin noise in Processing" which I modify to include a regular random and Simplex noise option using Open Simplex Java source code , which is a Java This is done by creating a grid of triangular mesh on an angle map. Then each point on the mesh is adjusted base on value store in a 2D array generated by the 3 methods similar to how it is done for the height map. The Terrain is "moving" by re drawing the value of the terrain every single loop.

Result and Discussion

Noise Height Map Comparison



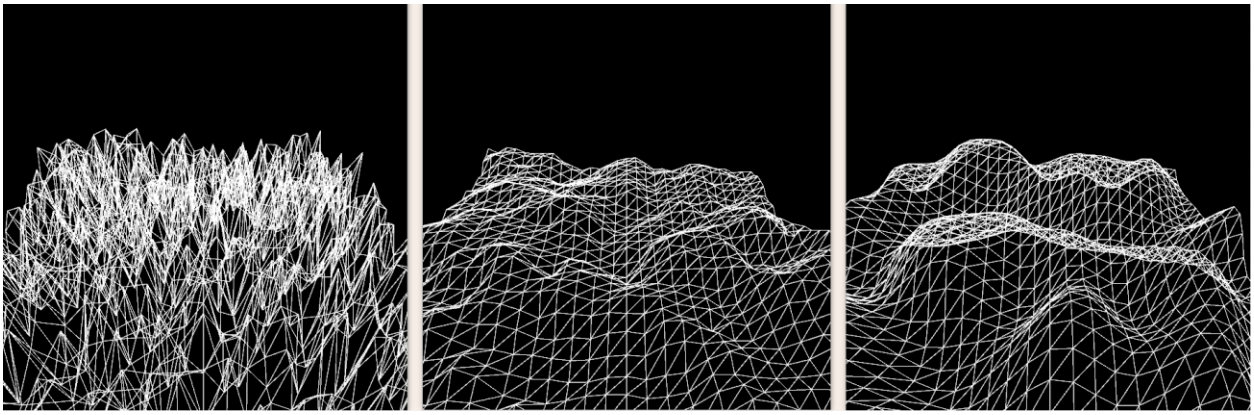
From Left to Right: Regular Random, Perlin Noise, Simplex Noise

The above is the resulting heightmap. Each of the three have a very clearly different characteristic. The first regular random look much like television static and look the grainiest as there is no coloration to be found. The second Perlin heightmap look smooth than the first map and it can be observe that there starting to be a clearly clustering of regions. Finally, in Simplex version we have a very smooth texture that look much more organic, this is because the noise is made by taking the whole area into consideration instead of just neighboring region.

Looking at both algorithm for Perlin noise the complexity is O(n^3) as it loops through each coordinate value and then one more loop to apply the frequency. The algorithm took an average of 1590.896 ms to run over an area of 1000x1000 pixels.

The complexity of Simplex algorithm is O(n^2). Although it involves more calculation steps, each step involved is in linear time which allow this method to be much faster. The algorithm took an average of 195.679ms to run over an area of 1000x1000 pixels, which is about 8 time faster. Which will be a huge time saver when you consider life application with millions and millions more pixels.

Terrain Generation Comparison



From Left to Right: Regular Random, Perlin Noise, Simplex Noise

For better visual please view the animated version in the result PowerPoint.

Beyond the benefit of significant reduction in computation. One main property of Simplex noise is the reduction in directional artifact. This is better observe in the animated version, but if you take a look at the terrain generated via Perlin noise in the animated you will observe that some part of terrain are still moving independently of the whole environment thus creating this clearly separated region or the direction artifact. In comparison, the Simplex terrain are moving as a whole and any changes in one region affect the whole environment seamlessly creating an even smoother look and feel. In term of usage however, if you want to create a more solid texture, I may still choose to use Perlin noise over simplex. For example, a mountain, I feel will look more realistic if it uses some of the rougher edges that the Perlin's algorithm leave behind. If I were to animate a water texture, then I may choose to use Simplex as the smoothness of it may be more suitable for that application. The parameter of Simplex can be adjusted however so it is possible to produce a texture that are less smooth. Overall one thing is clear, Simplex noise's efficiency is far superior to that of Perlin noise.

Sources

Simplex noise demystified - Stefan Gustavsons:

http://weber.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf

Understanding Perlin Noise:

https://flafla2.github.io/2014/08/09/perlinnoise.html

OpenSimplex Noise Source Code (Java):

https://github.com/processing/processing/blob/master/core/src/processing/core/PApplet.java#L5070

Simplex Noise Source Code (C++):

https://github.com/SRombauts/SimplexNoise/blob/master/src/SimplexNoise.cpp

3D Terrain Generation with Perlin noise in Processing:

https://thecodingtrain.com/CodingChallenges/011-perlinnoiseterrain.html