Terrain generation using noise

Timur Yakupov

I. Introduction

Terrain generation is a common problem for a game development. Usually it takes a lot of time and resources so making a terrain generator is a good solution.

At this report the terrain generation which is based on noise is represented.

Two different times of noise were chosen:

- Value noise
- Perlin noise

There are also added some variables that can be used to make generation more controllable, like lacunarity, persistence, etc.

II. Necessary materials

Decision to use a game engine was made due to the lack of experience in game development. Unity3D was chosen to work with for its simplicity and its documentation which is easy to understand. Another purpose for using Unity3D is that there are a lot of guides available in the net.

III. Methods

The generation of terrain consists of 2 main steps. First step is to implement the noise itself and apply it to a texture. The second step is to use this noise to make a 3D terrain. But before discussing the whole method of generation, let's take a look how value noise and Perlin noise differ.

A. Value Noise vs. Perlin Noise

Value noise is a type of noise commonly used as a procedural texture primitive in computer graphics. It is conceptually different from, and often confused with gradient noise, examples of which are Perlin noise and Simplex noise. This method consists of the creation of a lattice of points which are assigned random values. The noise function then returns the interpolated number based on the values of the surrounding lattice points.

While Value noise is smooth, it has a blocky appearance. The patterns look random but are clearly constrained to a grid, which is undesirable when trying to create a more chaotic or natural-looking surface. If we orient those gradients in different directions and then interpolate between them, then the grid becomes a lot less obvious. As Ken Perlin was the first to use this technique, it is known as Perlin noise.

B. Surface Generation

First of all, it is necessary to mention, that there is no focus on describing how Perlin or Value noise work, it was implemented based on descriptions taken from the Internet and it does not contains any innovations.

The overall algorithm goes as follows:

- Generate a texture using perlin or value noise
- Apply it to the surface
- Perturb the shape of the surface, it can be simply done by displacing the vertices along the Z axis using noise samples
- Use a coloring that somewhat resembles terrain strata, you end up with a result that looks a lot like a landscape
- Apply the transformation of generated terrain based on a global variables

There are several global variables that are used to modify generated terrain.

The noise results in quite strong displacement of the surface. But often there is a desire for a more subtle displacement, so **strength** is a variable to control the strength of the noise.

The factor by which the frequency of noise changes is often known as the **lacunarity**, while the amplitude factor is often named **persistence** or gain. These variables affect the noise generation and the generated terrain respectively.

IV. Implementation

The implementation itself consists of 4 classes:

- **Noise** class for producing noise itself. Class contains methods for both Perlin and value noises.
- **TextureCreator** class for creating a texture and applying noise on it.
- **SurfaceCreator** class for applying a texture to a surface. It also contain a method for randomizing variables to generate a new terrain.
- **Camera** class for controlling camera.

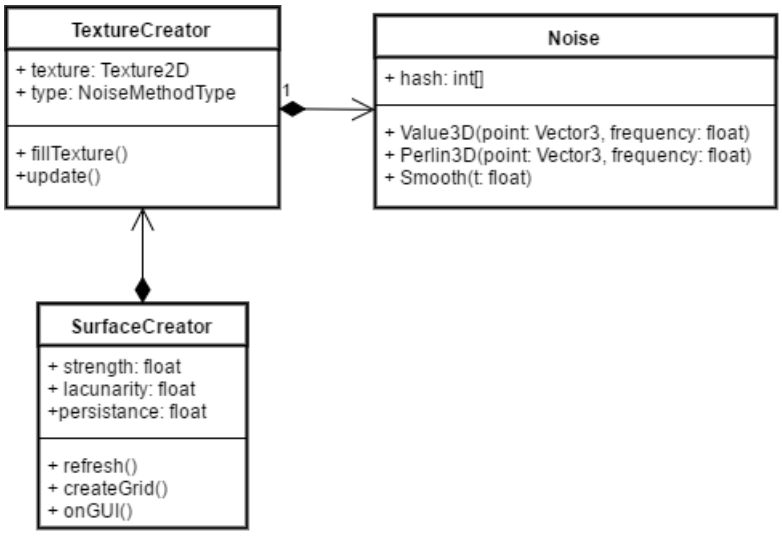
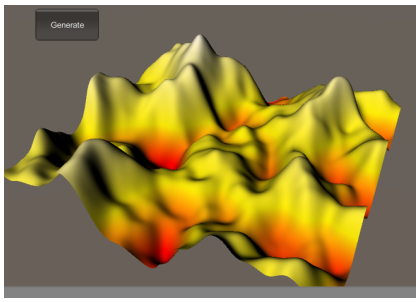
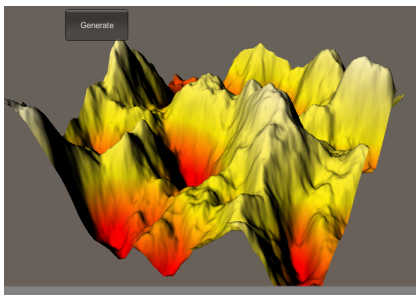


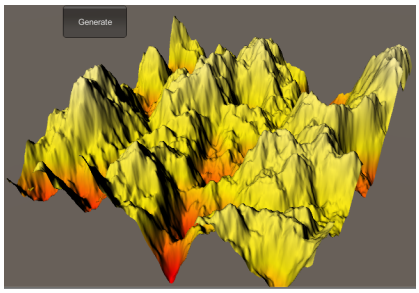
Fig. 1. UML diagram of implementarion

V. Results

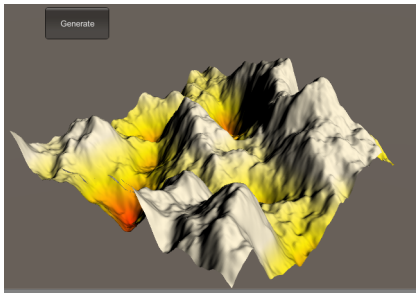
Here are the examples of generating the terrain using Perlin Noise:

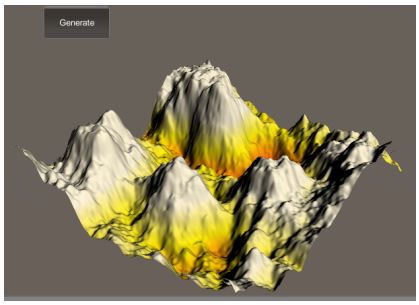


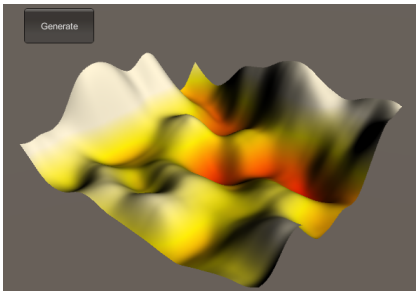




Here are the examples of generating the terrain using Value Noise:







These screen shots are made from Unity3D while generating with different parameters.

As can be seen on examples of generated terrain with using both Perlin and Value noise, both of them can be successfully be used for this purpose.

However generating terrain with the same parameters using Value and Perlin noise differs, there is no visually noticeable difference in quality of generated terrain. So, it can be said that both of them are equally suitable for terrain generation with no clear advantage of one method over another. At least, no advantages can be seen in this implementation of terrain generation.

There is no way to definitely say that one generated terrain is good, while other is not. It depends on the needs of developer and/or designer. This implementation is supposed to be used as offline tool for creating a terrain, but the parameters can be adjusted to use it for on-line generation for terrain in some game or other project.

VI. Taxonomy

This implementation can be used for terrain generation, but, as it is said above, it should be used for **offline** generation to let a developer decide what is suitable and what is not.

Terrain is one of the most important things in games, it can be both **necessary** if it is used for level generation or **unnecessary** content if it is used only for background generation.

Randomness is used for noise generation, so it's **stochastic**.

There are a lot of parameters used, like vector for noise generation and global variables to adjust the terrain, i.e. **Parameter vectors**.

In this implementation there is no testing, so algorithms are **Constructive**.

VII. Conclusion

This project shows that both Value and Perlin noise can successfully be used for the generation of terrain. It gave a lot of experience in using these algorithms and also gave a better understanding of how noise functions work.