Colorado School of Mines

CSCI 444

Advanced Computer Graphics

2D Procedural Heightmap Generation with Randomized Biomes

Author: **Robinson Merillat** *Supervisor:* **Dr. Jeffery Paone**

May 9, 2018

1 abstract

The purpose of this project is to explore procedural terrain generation, creating a realistic and believable terrain while attempting to conserve system memory and process time to prevent visible frame drop. I used a two pass technique storing computing a 2D noise texture in the first pass and using it in displacement mapping in the vertex shader of the second pass (heightmapping). Using several noise algorithms in continuum, I was able to reduce the number of octaves needed and still create a believable and interesting environment. A lot could be done to improve this project however, I am very happy with how it turned out and what I leaned about noise algorithms in the process.

Contents

1 abstract **1**

2 Introduction **3**

2.1 background . . . 3

2.2 Intro . . . 3

3 Previous Works **4**

3.1 Designer Worlds: Procedural Generation of Infinite Terrain from Real-World Elevation Data . . . 5

3.2 Realtime Procedural Terrain Generation . . . 5

3.3 GDC talk on Building worlds with Noise Gerneration in No Man's Sky . . . 5

4 Problem Statement **6**

4.1 Project Scope . . . 6

5 Problem Solution **6**

6 Results **8**

7 Conclusion **9**

8 Future Work **9**

9 Sources **10**

2 Introduction

2.1 background

Procedural Generation: "A method of creating data algorithmically as opposed to manually". [wikipedia]

Procedural generation has been around almost as long as computing has. It has been used to generate sample data, create music, and in computer graphics is often used to generate textures and object meshes. Modern video games often take advantage of this method to create randomized environments and events in order to maintain player interest. This method has several advantages over stored data including smaller file sizes, a greater amount of content, and a less predictable system.

Procedural Terrain Generation is a subsection of procedural generation that focuses on manipulating a mesh to create terrain that simulates a real or fantasy like world. It involves most visual aspects a biome including flora, and fauna.

Procedural generation works extremely well for terrain because, as we have found out, nature itself follows mathematical equations and isn't truly completely random. Trees bunch together and grow where water flows, erosion is a function of time. Plants grow in fractal like patterns. Each of these examples can are mathematical equations that can be manipulated to create a repeatable pattern.

2.2 Intro

Procedural Terrain Generation (PTG) is the culmination of everything that fascinates me; the way that Biology and Physics work together and using algorithms to study the relationships between them. Not only is this subject a personal interest of mine, but it is also heavily used in many industries, most heavily of which is the gaming industry. Many games with repetitive gameplay require procedural terrain to keep things interesting for the player. Each time you load into a map on Diablo you aren't quite sure where that one cave is going to be, or if it'll even show up at all. This kind of unknown exploration is what keeps players playing. Aside from games; PTG is also used in space programs to model the surface of nearby celestial bodies, allowing for simulations to be run for new technologies.

The difficulty of PTG comes in two forms. The first of which is integrating several algorithms together to where the output of one becomes the input to another and so on for as many algorithms as you'd like to use. Secondarily is determining just how the data should be manipulated to generate a realistic, desirable, result. Just how much should ridges be sharpened and peaks be weathered? What weight should each of the algorithms have? And in which order should they be used to get a result that looks and feels right?

The primary goal that I had for this project is to begin to understand how the various algorithms manipulated the terrain and how I can efficiently utilize them to create terrains that I would want to explore as a player within a game.

3 Previous Works

As previously mentioned, PTG is not a new techniques, not is it one that has been entirely flushed out. Below are several works that have inspired and instructed me on my path of discovering PTG.

3.1 Designer Worlds: Procedural Generation of Infinite Terrain from Real-World Elevation Data

This research paper delves into sending real world geographical elevation data into a variation of the perlin noise algorithm called value noise. The geographical data used is from the United States Geological Survey of Utah. The value noise algorithm differs from the noise algorithm in that it chooses a randow height to interpolate between as opposed to the gradients. Additionally value noise isn't contrained to be zero at grid points. It has a small speedup over perlin noise, however this is only visible at higher dimensions. [Parberry]

3.2 Realtime Procedural Terrain Generation

This paper provides an overview of erosion techniques used in games for terrain synthesis. It also goes into detail on how basic perlin noise is inferrior to voronoi diragrams. The 2D texture is then perturbed. Afterwards, seperate erosion techniques can be used such as thermal erosion, Hydraulic erosion, or an algorithm that the writers propose that combines the advantages of the two previously mentioned algorithms. [Olsen]

3.3 GDC talk on Building worlds with Noise Gerneration in No Man's Sky

This video clip from the Game Developer's Conference in March of 2017 discusses the techniques used by the developers of No Man's Sky to procedurally generate an entire univers in no more than 300MB of code, and 200GB of pre-developed assets. They create their own generation technique using a new noise method dubbed Uber Noise by the speaker Sean Murray. It combines several noise methods including Analytical derivative and domain warping.

[Murray]

4 Problem Statement

Many of the traditional programs and methods of terrain generation take into account large amounts of system Memory as most dedicated graphics cards (GPU) have their own memory. However, for an Integrated GPU, like that on a Macintosh laptop, the onboard GPU steals from the computers' RAM rather than its own. This oftentimes prevents running heavy graphical software such as high performance games on these laptops. The purpose of this project is to explore procedural terrain generation, creating a realistic and believable terrain while attempting to conserve system memory and process time to prevent visible frame drop.

4.1 Project Scope

Throughout the development of this program, it was quickly obvious that many of the stretch goals that I would have liked to add were not within the scope of this project due to the various time constraints. Many of the goals and stretch goals of this project involved understanding the algorithms that can create believable biomes. Each of which is on its' own a Master's thesis worth of understanding. As such I decided to forego some of these investigations until I had more time to sit down and understand the material.

5 Problem Solution

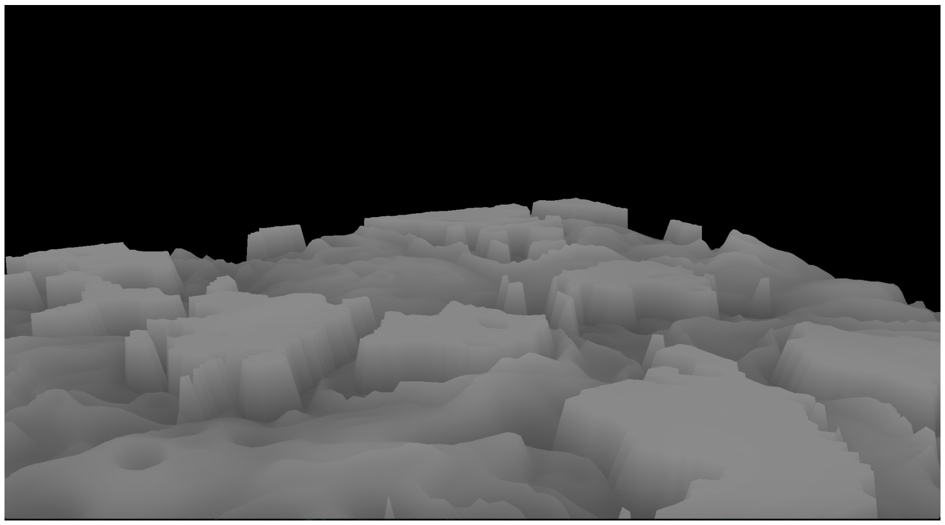
The solution that I came up with require a lot of trial and error testing, along with hours of searching for better, more efficient noise algorithms. Using openGL 3.3+ in c++ I created a two pass method that generated the noise texture on the first pass through the rendering pipeline, and then generated the terrain on the second pass. I used a Frame Buffer Object (FBO) to store the results of the first pass' fragment shader to a 2D texture. This texture was then used in the vertex shader of the second pass to perform displacement mapping. This technique uses the RGB values of the texture to distort a mesh. In this case, I was using a form of displacement mapping called Hightmapping. This technique sets the y(vertical) value of each vertex equal to a scalar * the color at that vertex on the noise texture. As the texture is black and white, a range of heights can be computed, 0 being pure black and 1 being pure white. The scalar value then perturbs the height to accentuate or diminish the features.

Several noise algorithms are defined in the fragment shader of the first pass such as: Classic Perlin Noise, Voronoi (Cell) Noise, Fractal Brownian Motion, Ridged Noise, Billowed Noise, and Erosion Noise. Using these algorithms, I constructed several "biome" functions that utilized a different combinateion for these algorithms in order to simulate the related biome. I implimented an Island, Mountain, Canyon/Desert, and Rugged terrain function. A randomized value seeded on the current system clock is sent into the shader as a uniform and that determines which biome is generated. Additionally, a few other pre-computed random values are sent in as scalars to the various noise algorithms. This gives a small amount of randomization, however, it is often times not a very visible differnece from one generation to another.

6 Results

Testing for this project was performed on an Apple Macbook Pro(Retina, 13-inch, Late 2012). It has a 2.5GHz Intel Core i5 processor, 8Gb of 1600MHz DDR3 RAM, and an Intel HD Graphics 4000 1536Mb onboard GPU. Using this device let me test on a system that a typical user may use, at least as far as the processor and GPU. This means that if the program runs well on this device, chances are it'll run great on a newer or better spec'd device.

I had to run several iterations and tests of simply modifying scalars and changing the order of different noise algorithms get desired effects for each biome, however, in the end, I believe that I have created some relatively believable terrains that run smoothly on the tested computer.



Using a combination of several algorithms allowed me to keep the number of octaves used in the noise algorithms low. This is helped by the scale and perspective of the view. If I were on the surface of the terrain, I may want the octaves to be higher to generate more close up detail, however, from far away, fewer octaves are necessary, especially if you added surface normals and decent lighting.

7 Conclusion

This project was a lot of fun, but very complex for me. Although I love them, I have trouble understanding algorithms and how they can be manipulated. As such I likely spent more time than was necessary of fiddling with the various algorithms and trying to get them to work.

If I were to do this project again, I would most certainly start by loading in a low poly model and trying to get a Tessellation shader working to create a level of detail as this can save a lot of processing power. I would have also implimented better lighting and textures to make the terrain look more appealing than a bunch of black and white textures.

8 Future Work

I plan to continue with this project in an attempt of increasing both the complexity and the performance. Some of the methods in which I may increase the performance may be to convert the mesh to be drawn as a Triangle Strip, conserving valuable vertices. Predetermining which fragments will be in view and discarding them, and using Tessellation Shaders to implement a Level of Detail hierarchy to render low poly far away and high poly close up to the camera. Complexity wise, I would like to get more randomized and versatile noise algorithms. I would also like to use more accurate erosion algorithms and investigate procedurally populating each terrain with low poly flora and fauna. Lastly, I would like to implement using 3D noise with voxels to create more randomizes and interesting terrain.

I would like to turn this from just a 2D terrain generation simulation in 3D into a 3D ecosystem simulation in 4D (time).

9 Sources

1. Procedural Generation, Wikipedia, [Link].

2. Ian Parberry, Designer Worlds: Procedural Generation of Infinite Terrain from Real-World Elevation Data, Journal of Computer Graphics Techniques (JCGT), vol. 3, no. 1, 74-85, 2014

3. Olsen, J. (2018). Realtime Synthesis of Eroded Fractal Terrain for Use in Computer Games. [ebook] Available at: https://pdfs.semanticscholar.org/5961/c577478f21707dad53905362e0ec [Accessed 23 Mar. 2018].

4. Smelik, R., Tutenel, T., Bidarra, R. and Benes, B. (2014). A Survey on Procedural Modelling for Virtual Worlds. Computer Graphics Forum, 33(6), pp.31-50.

5. Sean Murray GDC-Talk (2017) — Building Worlds with Noise Generation — No Man's Sky. (2017). [video] Game Developer's Conference Link

6. Vivo, Patricio Gonzolez, The Book of Shaders, GLSL-noise Implimentations [Github]