DH2323 Project Specification

Marcus Samuelsson

May 16, 2024

Contents

1 Introduction

Pseudo-random generation has become a cornerstone in modern game development, enabling the creation of diverse and expansive virtual environments. This project focuses on the development of a pseudo-random landmass generator, leveraging the mathematical concepts of Perlin noise and Fractal Brownian Motion (fBm) to create intricate and varied terrain.

The report provides a comprehensive overview of the landmass generation process, starting from the generation of pseudo-random values, moving through the conversion of these values into a three-dimensional mesh, and a quick look at creating a texture to bring the terrain to life.

Each section of the report delves into the theoretical underpinnings of the respective stage of the process, elucidating the mathematical and computational principles at play. Furthermore, the report presents the practical implementation of these theories in Unity, a popular game development platform, using C# as the primary programming language.

The project encapsulates several key aspects of game development, including procedural generation and 3D graphics. The result is a robust landmass generator capable of producing a wide array of terrains, each with its unique topography and aesthetic.

2 Implementation and Results

This section delves into the various technologies harnessed in the execution of this project, elucidating their roles and how they interplay to facilitate the successful implementation of the pseudo-random landmass generator. It provides a detailed exploration of each technology's application, underscoring the significance of their integration in achieving the desired outcomes. Furthermore, it also shows the results from each step of the process.

2.1 Perlin noise

Perlin noise is a pseudo-random sequence of floating-point numbers generated across a two-dimensional plane. It is a gradient noise, lending itself well to natural phenomena like the complex patterns of a terrain. Unity conveniently incorporates a Perlin noise function within its Mathf library, which generates and returns a floating-point value in the range of 0.0 to 1.0[1].

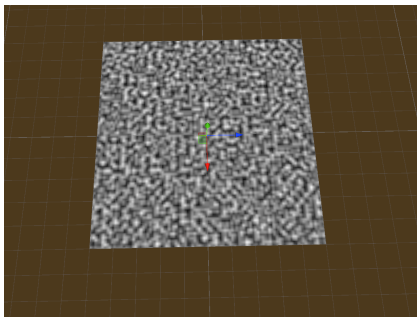

Figure 1: Perlin noise generated using unity's built-in function rendered as a texture on a plane.

The pseudo-random pattern generated by Perlin noise has a wide array of applications. Its strength lies in its ability to produce patterns that transition smoothly, thereby preventing abrupt changes[2]. This makes Perlin noise an excellent choice for generating organic and natural-looking patterns.

2.2 Fractal Brownian Motion

Perlin noise works great for generating a pseudo-random gradient pattern. However, as seen in Figure 1 the noise does not resemble the shape of a typical landscape. This is where Fractal Brownian Motion(fBm) comes in to help create a more fitting gradient. Using these three values octaves, lacunarity, and persistence it is possible to reshape the generated gradient noise.

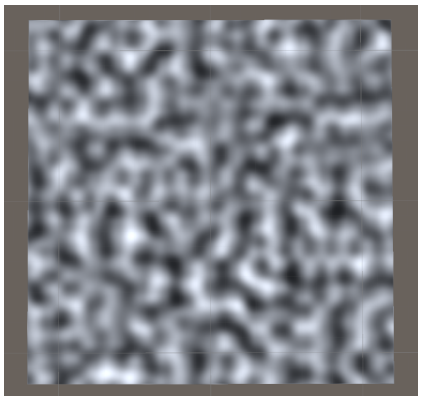

Figure 2: Perlin noise generated using unity's built-in function rendered as a texture on a plane in combination with fBm.

2.2.1 Frequency

Frequency in the context of noise generation refers to how often the values in the noise changes, which could also be explained as how close together the features of the noise are. This means that higher frequencies give smaller more frequent features, while a smaller frequency returns larger, less frequent features.

2.2.2 Amplitude

Amplitude in the context of noise generation refers to the range or strength of the values in each layer of noise(Octaves). Where a higher value corresponds to a larger range of values in the different layers, and a smaller value brings less range.

2.2.3 Octaves

Octaves, a term borrowed from music, represent different iterations of noise. In Fractal Brownian Motion (fBm), the number of octaves equates to the number of noise layers, each with a distinct frequency and amplitude. However, unlike in music, the frequency doesn't necessarily double with each octave in fBms[3][4]. The number of octaves is the number of noise layers, where each layer has a set variation in their frequency and amplitude.

2.2.4 Lacunarity

Lacunarity, derived from the Latin word for gap, is used to create gaps between frequency variations caused by octaves[5]. In fBm, lacunarity controls the frequency of each successive noise layer[3]. A lacunarity of 1 results in all octaves having the same frequency, while a value of 2 doubles the frequency with each octave. Higher lacunarity values yield more detailed but chaotic noise, while lower

values produce smoother, less detailed noise.

2.2.5 Persistence

Persistence controls the change in amplitude between each successive octave. It is set to a value between 0 and 1. Where a value closer to 1 will result in a rougher noise resulting in a more mountainous landmass, while a smaller value will lead to a larger change in amplitude and a smoother landmass.

2.3 Coloring and Texture

The generated noise results in a two-dimensional array of floating-point numbers, each ranging between 0 and 1. These values serve as the blueprint for shaping the landmass, with each value corresponding to the height at a specific coordinate (x, z) in the world space. This noise not only shapes the terrain but also influences its visual representation. By mapping these noise values to colors, we can generate a texture that visually communicates the height variations of the terrain. This color-coded texture is then applied to the terrain, providing a visual gradient that intuitively represents the topography of the landmass.

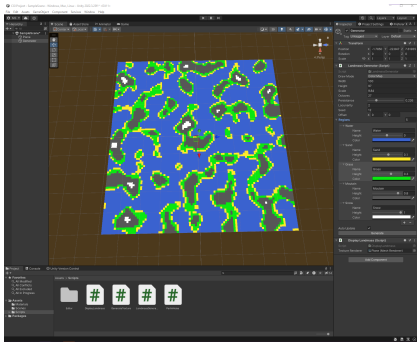

Figure 3: The generated perlin noise combined with fBm with a texture of colors for different heights.

Figure 3 presents a plane textured with a combination of Perlin noise and Fractal Brownian Motion (fBm). In Unity, texture creation can be achieved through two distinct methods: either by individually modifying each pixel or by generating a one-dimensional array containing all colors, which is then assigned to a texture of the corresponding size.

Figures 1 and 2 were both generated by creating a gradient from black to white, with each pixel's value corresponding to the equivalent value in the generated noise map. This noise map is a two-dimensional array of normalized floating-point numbers.

To apply color, each value in the noise map is iteratively examined and assigned a color. This color is determined by a range set in the Unity inspector, allowing for a smooth transition from one color to the next, thereby creating a visually appealing and informative representation of the terrain's topography.

2.4 Mesh generation

In Unity, meshes form the cornerstone of 3D modeling, constructed using a network of polygons, specifically triangles. The creation of a mesh is predicated on the compilation of data across four distinct arrays:

- Vertex Array: This array holds the vertices, the fundamental points in 3D space that define the shape of the mesh[6].
- Normal Array: This array contains the normals, which are vectors perpendicular to the mesh surface, crucial for lighting calculations.
- Triangle Array: This array defines the triangles, the basic building blocks of the mesh, by referencing groups of three vertices in the vertex array[6].
- Texture Coordinates (UVs) Array: This array stores the UV coordinates, which map the 2D texture onto the 3D mesh.

These arrays collectively encapsulate the geometric and visual characteristics of a mesh, enabling the creation of complex 3D models in Unity.

2.4.1 Normal Array and Mesh Rendering

While creating mesh data in Unity, the inclusion of a normal array is optional. By default, the normal of each mesh, which determines the side of the mesh that is rendered when viewed in the engine[7], is dictated by the sequence in which vertices are added to a triangle.

The orientation of the normal, and consequently the rendering direction of the mesh, is contingent on the order of vertex addition for the triangle. For instance, if the vertices lie flat on the y-axis, a triangle constructed with vertices in a clockwise order would render upwards. Conversely, if the vertices were added in a counter-clockwise order, the mesh would render downwards.

This orientation is computed using the **RecalculateNormals()** function on the generated mesh, once all the information has been added to the mesh data. This function recalculates the normals of the mesh from the triangles and vertices, ensuring accurate rendering.

2.4.2 Texture coordinates(UVs)

UVs are two-dimensional texture coordinates that correspond to the vertices of a mesh's geometry. They play a crucial role in enabling a 2D texture to be accurately applied onto a 3D object. Essentially, UVs establish a link between each vertex point of the mesh and a specific pixel on the texture. This mapping ensures that the texture aligns correctly with the mesh, facilitating a cohesive visual representation[8].

2.4.3 Perlin Noise and Mesh Generation

The process of generating a mesh from Perlin noise is simplified with the establishment of a class that encapsulates the mesh data. The dimensions of the arrays within this class are de-

termined by the height and width of the terrain, which are set in the inspector of the **LandmassGenerator** file.

The number of vertices per line for both width and height is calculated by subtracting one from the width and height of the terrain respectively. This is because the vertices are shared between adjacent squares of the mesh grid.

The MeshData constructor then initializes three arrays:

- The vertices array, which stores the 3D coordinates of each vertex. Its size is the product of the number of vertices per line of width and height.
- The uvs array, which stores the 2D texture coordinates for each vertex. Its size is the same as the vertices array.
- The triangles array, which stores the indices of the vertices that form each triangle. Its size is six times the product of one less than the number of vertices per line of width and height, as each square in the grid is made up of two triangles, and each triangle is defined by three vertices.

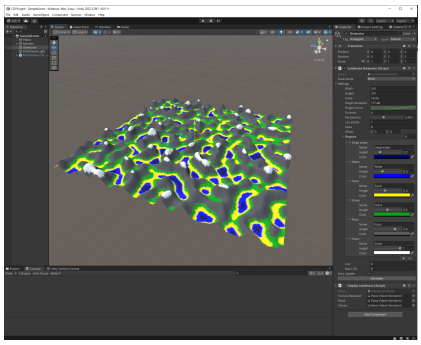

Figure 4: The landmass with all the features shown so far applied.

However, one must be mindful of a constraint when generating the mesh. Unity imposes a limit on the maximum number of vertices that a single mesh object can contain, capped at 65535 vertices[9]. Consequently, the product of the terrain's height and width must not exceed this maximum limit. This necessitates a restriction on the maximum size of the terrain, ensuring that the mesh generation process adheres to Unity's vertex limit.

2.5 Level of detail

Level of Detail (LOD) is a technique used to enhance performance by rendering simpler meshes when they are further from the camera. In the context of our landmass generator, which works with relatively simple shapes, it's straightforward to incorporate an LOD system during mesh construction.

This is achieved by skipping some vertices and enlarging the triangles. However, care must be taken to ensure the mesh doesn't deform due to insufficient vertices to connect the mesh's start and end properly. This is particularly

important when dealing with variable width and height.

To prevent deformation, both width and height should be divisible by the same number. The maximum LOD is calculated by halving the height and width until further division is not possible. This means that some landmass shapes can be simplified more than others.

In this scenario, the largest possible LOD is 6, based on the maximum vertex size. This means the maximum number of vertices that can be skipped is 12 (6 * 2), but this is only feasible for a limited range of sizes.

3 Future works

There are many different ways to expand upon the landmass generator. In this section some potential addons will be described.

3.1 Endless Landmass Generation (Chunks)

In the context of this project, the Level of Detail (LOD) implementation primarily serves aesthetic purposes, given that the landmass, even at its largest size, does not significantly impact performance. However, this feature could be leveraged for more complex scenarios, such as the generation of an endless landmass. This could be achieved through a technique known as chunking. Most of these features are not in this project as they would require a substantial amount of time to do.

Chunking refers to the process of dividing large datasets into smaller, more manageable segments, known as chunks[10]. In the context of this project, each mesh, limited by Unity's maximum vertex count, would represent a chunk. As the camera navigates through the game, additional chunks would be loaded based on the camera's position, all derived from the same endlessly generated noise map.

Each segment of the noise map would correspond to a separate mesh, generated as the camera approaches. This approach would allow for the effective utilization of the LOD system implemented in Section 2.5. More chunks could be displayed simultaneously if those further away are generated with fewer polygons, thereby optimizing performance while maintaining visual fidelity.

3.2 Landmass Population

While the generated landmass closely resembles a realistic terrain, it could be further enhanced by populating it with natural objects such as trees, rocks, bushes, and grass. This would require either modeling these objects or sourcing them online, which could be time-consuming. However, the implementation process should be relatively straightforward.

Similar to the color application process, an interface could be added to the landmass inspector in Unity. This interface would allow for the addition of various objects. Each object could be assigned a height range on the landmass where it could potentially be instantiated.

During the terrain generation process, a random number of objects, deter-

mined by a specified spawn rate, could be instantiated at suitable locations. These locations would be determined based on the height data derived from the noise map. This approach would add a layer of realism to the landmass, making it more visually appealing and immersive.

3.3 Shaders

Shaders represent a significant opportunity for enhancing the visual appeal and interactivity of the terrain. They can introduce a wide range of visual improvements to the landmass and significantly expand the customization capabilities of the implementation.

With shaders, it becomes possible to create realistic water effects, replacing the current simplistic blue patches. They also enable smoother transitions between colors and allow for the application of textures instead of just flat colors. These are just some small examples of improvements that would enhance the look a lot already, but shaders could to a lot more than this to improve the project's aesthetics.

However, implementing shaders would require learning a new programming language, which could be a time-consuming endeavor. Given the complexity and the time investment required, this aspect could be the most challenging to expand upon in this project. Nevertheless, the potential improvements in visual quality and realism make it a worthwhile consideration.

4 References

[1] Unity Technologies. *Mathf.PerlinNoise*. URL: https://docs.unity3d.com/ScriptReference/Mathf.PerlinNoise.html.

[2] URL: https://www.khanacademy.org/computing/computer-programming/programming-natural-simulations/programming-noise/a/perlin-noise.

[3] patricio gonzalez vivo and jen lowe. *The book of shaders*. URL: https://thebookofshaders.com/.

[4] Inigo Quilez. URL: https://iquilezles.org/articles/fbm/.

[5] Charles Sturt University Audrey Karperien. URL: https://imagej.net/ij/plugins/fraclac/FLHelp/Lacunarity.htm.

[6] Ahmed Schrute. *Understanding mesh anatomy in Unity*. 2021. URL: https://medium.com/shader-coding-in-unity-from-a-to-z/understanding-mesh-anatomy-in-unity-2507cb1ae011.

[7] Unity Technologies. *Mesh Data*. URL: https://docs.unity3d.com/Manual/AnatomyofaMesh.html#normal.

[8] 2022. URL: https://www.pluralsight.com/blog/film-games/understanding-uvs-love-them-or-hate-them-theyre-essential-to-know.

[9] Unity Technologies. *Mesh.indexFormat*. URL: https://docs.unity3d.com/ScriptReference/Mesh-indexFormat.html.

[10] Webopedia Staff. *What is a chunk (Data Chunk)?* 2022. URL:

https://www.webopedia.com/
definitions/chunk/.