

Analysing System Efficiency for Procedural Biome Generation

Alexandre Kouda (21009967)

Project Supervisor Gary Ushaw

Abstract

This dissertation looks at unity and its use to make procedurally generated biomes and cities efficiently while being able to maintain a high level of detail for these structures. It also looks at multiple opinions in the game industry on the various usages of tools and strategies in order to provide justification for my project.

The motivation for this project stems from an avid interest in open world games such as Minecraft and Terraria that each use different procedural generation methods to provide a unique experience for their users. This dissertation gives back to the community by testing the effectiveness of these methods and showing possible counter option for future games.

The resulting application from this dissertation allows for the generation of multiple biomes and cities with the ability to endlessly traverse the planes of the game. It allows for randomised seed to provide a unique experience every time.

Declaration

“I declare that this dissertation represents my own work except where otherwise stated.”

Acknowledgments

I would like to firstly thank my project supervisor Gary Ushaw for letting me take part in this project. He has provided guidance when needed and always made time to help. I would like to also thank my family and friends who not only helped with testing of my project but were there when I needed them. To them I dedicate this dissertation.

Table of Contents

Acknowledgments	4
1 Introduction	8
1.1 Purpose	8
1.2 Project Aim and Objectives.....	8
1.3 Dissertation Outline	9
2 Research	11
2.1 Beginning of Procedural Content Generation in games	11
2.2 Natural Terrain/Object Generation	11
2.3 Procedural content generation methods in popular games	14
2.4 Optimization of procedurally generated content.....	16
2.5 Different uses for procedural Generation.	17
2.6 Limitations of procedural generation.....	18
2.7 Research Summary.....	19
3 System Design	20
3.1 Requirements	20
3.2 High-level design	20
3.3 Biome Selection UI Design	21
3.4 Mesh Generation Design	22
3.4.1 Noise Scale.....	22
3.4.2 Octaves	22
3.4.3 Persistence	22
3.4.4 Lacunarity	23
3.4.5 Seed	23
3.5 Texture Generation Design.....	23
3.5.1 Texture.....	23
3.5.2 Tint/Colour	24
3.5.3 Tint Strength.....	24
3.5.4 Start Height	24
3.5.5 Blend Strength.....	24
3.6 Terrain Generation Design(Come back).....	24
3.6.1 Mesh Height Curve	24
3.6.2 Mesh Height Multiplier	25
3.7 Map Generation Design	25
3.8 Map Generation Settings Design	25
3.8.1 Render Distance.....	26

3.8.2 Level of Detail.....	26
3.8.3 Chunk Size	26
4 Implementation.....	27
4.1 Implementation of the Mesh Generator.....	27
4.1.1 NoiseData.....	27
4.1.2 Terrain Data	30
4.1.3 Mesh Generation	31
4.2 Implementation of the Texture Generator	33
4.3 Implementation of the Map Generator	36
4.3.1 Editor Level of Detail.....	36
4.3.2 Data as Assets	39
4.4 Implementation of the Map Generator Settings	40
4.5 Implementation of Biome Selection UI.....	41
4.6 Implementation of Extra Technologies	43
4.6.1 Fall off map	43
4.6.2 Flat Shading	44
5 Testing	46
5.1 Testing Strategies	46
5.1.1 Environmental testing	46
5.1.2 System Usage testing.....	46
5.1.3 Ease of use.....	47
5.1.4 Load Testing	47
5.1.5 Full build test	47
5.2 Testing Results.....	47
5.2.1 Environmental testing results.....	47
5.2.2 System Usage testing results	47
5.2.3 Ease of use testing results	52
5.2.4 Load testing results	53
5.2.5 Full build test	54
5.3 Testing Summary	55
6 Evaluation.....	56
6.1 Research	56
6.2 Design	57
6.2.1 Requirements Breakdown.....	57
6.2.2 Entity Relation Diagram.....	57
6.2.3 Design of the application	58

6.3 Implementation	59
6.4 Testing.....	59
6.4.1 Testing Strategies.....	60
6.4.2 Testing Results	60
6.5 Overall evaluation	61
6.5.1 Research.....	61
6.5.2 Design.....	61
6.5.3 Implementation.....	61
6.5.4 Testing	61
7 Conclusion	62
7.1 Aims & Objectives	62
7.2 Skills Learnt.....	64
7.3 Future Developments for the Application	64
7.3.1 Allowing users to modify some aspects of generation.	64
7.3.2 Adding detail and animation to textures.....	64
7.3.3 Adding the generation of multiple biomes simultaneously.	64
7.3.4 Walking on the biomes.....	65
7.4 Final Thoughts	65
References	66
Appendices	67
Full Island Biome.....	67
Full Maze Mountain Biome	68
Full Mountain Biome.....	70
Full Desert Biome	72
Full Grassland Biome	73
Full Christmas Biome	74

1 Introduction

1.1 Purpose

Procedural generation is a form of synthetic media (AI generated content) and a core method in the gaming industry. It creates data algorithmically through the combination of human generated content and algorithms with computer generated randomness. These algorithms are very practical in the video games industry as "Procedural generation is a widely used approach to create a variety of virtual content. It can be used at different levels, from individual objects to large-scale open worlds and game scenarios." ^[1]. Thus, it has become a priority for computer scientist to understand how to implement this algorithm efficiently in a system without compromising the quality of the game.

I gained an interest in procedural generation as early as 9 years old when I first played Minecraft. I was fascinated by how I was able to traverse an "endless" world without my old laptop completely failing to run, while also being able to create unlimited amounts of these world which each in turn had their own completely unique endless world. This was the main factor as to why I decided to investigate terrain and city generation as Minecraft not only provided different biomes but cities that went along side with these biomes.

In my project, I intend to explore the different technologies used to procedurally generate cities and biomes to provide a balance of both detail and processing power and analyse the effects each technology poses on the system, I will do this by making a game that procedurally generates biomes and cities as the player walks around the world. I will test different architecture and use visual representation to discuss and evaluate my findings. Further I will also look at popular examples of similar games such as Minecraft and terraria to help further my research and understanding.

1.2 Project Aim and Objectives

The aim of the project is to:

"To explore how different technologies that effect procedural generation of biomes and Analyse efficient uses of these technologies to create a balance between level of detail and amount of processing."

The project aim is split into 6 objectives:

- **Explore and identify the current methods used in procedural generation and how each can be used.**
- **Identify the potential limitations of the technologies used in procedural generation.**
- **Compare how different technologies effect the balance between level of detail and processing.**

- **Implement and develop a usable procedurally generated game which showcases different biomes.**
- **Analyse and summarize existing published examples of games that use procedural generation technologies.**
- **Implement and evaluate at least 2 methods and how they affect procedural generation.**

1.3 Dissertation Outline

Introduction

- Purpose
- Project Aims and Objectives
- Dissertation Outline

An introduction to the dissertation explaining what the project will be tackling and how this fits into the projects aims and objectives

Research

- Beginning of Procedural Content Generation in games
- Natural Terrain/Object Generation in games
- Procedural content generation methods in popular games
- Optimisation of procedurally generated content
- Different methods for procedural content generation

All relevant research I have completed such as academic papers and possible interviews carried out with experts in industry.

System Design

- Requirements
- High-level design
- Biome Selection UI
- Overall Map generation

An intricate account of how I designed my project

Implementation

- Overall Biome Generation
- Extra technologies added

An intricate account of how I completed my project

Testing

- Testing Strategies
- Testing Results

All analysis and summary of project testing and results

Evaluation

- Summary of all sections
- Pros and improvements to be made

An evaluation of the work that was done and my opinions on how it went

Conclusion

- Opinion on how the project went in terms of my aims and objectives
- Skills learnt
- Future Developments
- Final Thoughts

An evaluation of the work that was done and my opinions on how it went

2 Research

2.1 Beginning of Procedural Content Generation in games

This sub section will be used to explore where procedural content generation originates from and its impacts on today's games.

The first computer game was developed in 1962 at MIT by Stephen Ruseell and was called Spacewar which ran on a PDP-1 computer. The game consisted of two players each controlling a space ship and circling round a blank dark battlefield trying to shoot each other. Games stayed rather basic for until the "Commonly cited "first" uses of PCG" "from the early 1980s: Rogue or Elite"[2].

These games used Procedural generation which is a method of creating data algorithmically instead of manually, this is usually done by combining human-generated content, Computer-generated randomness and algorithms in order to usually create computer graphics such as textures or 3D models. In the instance of Rogue as it was an early instance of PCG(procedurally generated content) where system limitations were more prevalent it showcased procedurally generated levels where a player navigated a maze type of environment at the time referred to as a dungeon.

"The game that named an entire subgenre and defined nearly all of its tropes, Rogue is undeniably an influential game."^[3] This subgenre of games called rogue-like that allowed for unique game experiences and allowed to make much bigger games with the current technological limitations of the time; This is because in early game history there were issues of memory limitations which was one of the main driving factors for procedurally generated games.

Not only was this game a staple for PCG but because of this initial step for PCG many of the most popular games in the world such as No Man's sky, Minecraft and Terraria were able to be created which previously were not feasible due to system limitations and as a result wouldn't be able to be accessed by all.

2.2 Natural Terrain/Object Generation

This sub section is used to look at terrain generation and the different ways it can be used.

One of the possible ways of procedurally generating natural terrain is to split the generation of possible naturally occurring objects such as:

- Plants
- Rivers(Water)
- Terrain

A common practise used to visualise plants is the L-System due to naturally reoccurring patterns in plants known as fractals. A Survey of Procedural Content Generation of Natural Objects in Games by Tianhan Gao and Jiahui Zhu states:

“As early as 1969, biologist Aristid Lindenmayer [12] used Lindenmayer (L-system) system to simulate the growth process of complex organisms such as algae and fungi, and later extended it to simulate higher plant species and complex branching systems.

System [13] is a context-free syntax. Each rule generated is only applicable to one symbol in geometry, and other symbols are not affected by the rules, starting from the initial structure. By replacing some parts of the syntax notation to form an object, and iteratively applying some rules for the string symbols to create a branch structure, each recursive iteration will increase the growth level of the string, and finally the string can represent the branch structure of the growing tree.”^[4]

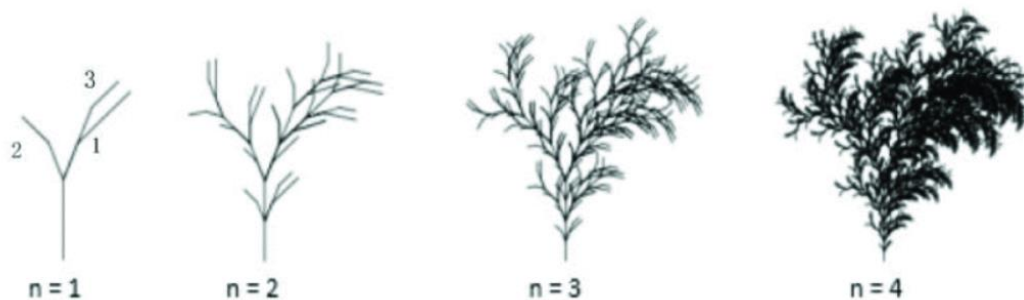


Figure 1^[4]: Fractal tree generated by L-system

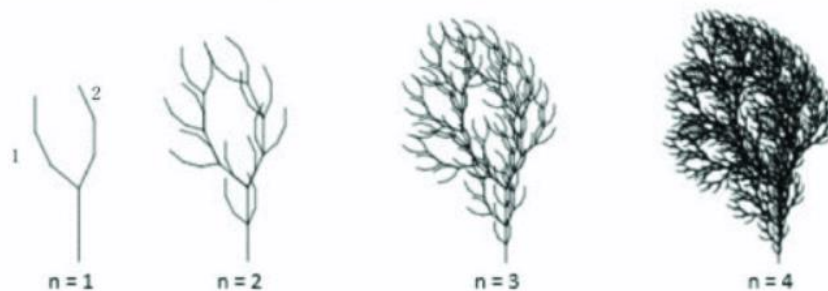


Figure 2^[4]: Fractal tree generated by L-system

“Some of the plants in nature have inevitable self-similarities, and Mandelbort [8] is called fractal. Fractal [9] is a roughly divided geometry, which is divided into several parts of the original geometry, each of which is only different to size of the original whole. Fractal is defined as a geometric branch of describing the geometric patterns contained in nature. Fractal has global determinism and local randomness. Fractal structure [10] has higher stability and fault tolerance than Euclidean geometry with stronger certainty, this is why fractals are so common to nature, from trunks and branches to complex leaf vein structures”^[4]

This section of the paper was relevant because it explains why techniques such as the L-System work in order to help generate naturally occurring objects and allowed me to have a more in depth understanding as to why and how we are able to generate content that is able to look so similar to nature.

Futher into the paper they lightly touched on possible implementations that have been put forward for the procedural generation of rivers:

“In order to increase the authenticity of river landscape generated, A. Peytavie and T. Dupont [14] and others put forward a novel program framework to create River Landscape: taking bare-earth as input, deducing river network trajectories affected by water flow, carving riverbeds in terrain, and then automatically generating corresponding blend-flow tree for the water surface. The width, depth, and shape of the riverbed is derived from topography and river type. The water surface is defined by a time-varying continuous function encoded as a blend-flow tree, in which leaves are parameterized procedural flow primitives. The resulting framework can produce various of river forms, ranging from delaying winding rivers to the torrent of surging currents. These models also include surface effects, such as foam and leaves flowing down the river.”^[4]

Although I will not be going as in-depth into water as adding things such as flow and particles. It is important to understand how to efficiently implement water surfaces as many of the biomes I am going to implement have a form of water in them.

Lastly, they also describe the use of a technique in order to create procedurally generated terrain through Perlin noise:

“The generation of height map is usually based on fractal noise generator. The 2D Perlin noise map created by fractal Brownian Motion [20] is a series of fluctuation data onto smoothness and predictability (see Fig. 7). It generates noise by sampling and interpolating points in a random vector grid, and scales and accumulates several noises with increasing frequency into an elevation map, which is suitable for creating a landscape with mountains and valleys.

Perlin noise algorithm has excellent performance, because each grid point can be calculated independently of the values of neighbouring points, it is very suitable for parallel processing. However, the terrain generated by noise algorithm is generally uniform without the change of surface details. Therefore, if you want to add detail features of the smooth terrain, you can further to modify the terrain using algorithms based on physical phenomena, such as erosion.”^[4]

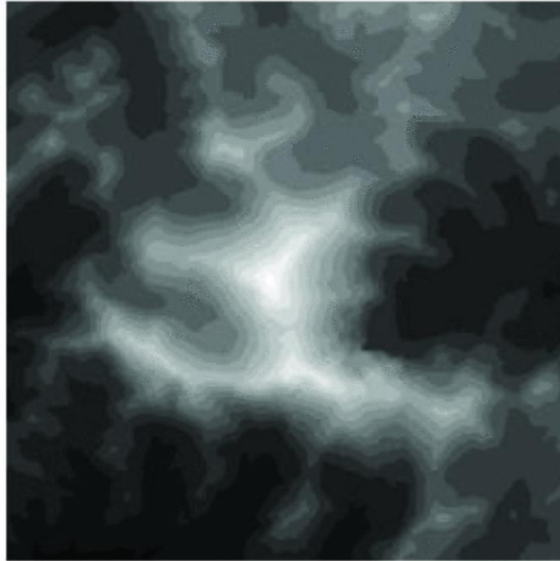


Figure 3: Height map of mountains obtained in greyscale

This was particularly important part of the paper as using Perlin noise allows you not only to get realistic mapping for the generation of terrain such as mountains; But it also allows for randomisation of this generation, meaning you are able to provide an efficient way to give the user a unique experience while upkeeping a high level of detail. This is supported by the papers closing statements *“Procedural content generation is becoming more and more comfortable in creating the natural objects and some complex aspects of the game.”*^[4]

2.3 Procedural content generation methods in popular games

This sub section looks at how procedural generation is used from the point of view of widely popular games.

Examples of popular games that use procedural generation are:

- Minecraft
- No mans sky

Minecraft is one of the most popular games in the world with over one billion downloads since its launch. Its been a game that has stood the test of time and it has done so by providing the user with a unique and fun experience regardless of how many worlds created.

This is because when creating a world a seed is generated which is a random unique value. This value dictates what feels like an “endless” procedural generation of landscapes and structures in the world and by giving users a unique seed for every world it allows users to gain new experiences with every iteration of the game they play.

Using this knowledge and in order to get a further understanding of the specifics of how biomes are procedurally generated in Minecraft I looked at The world of Minecraft Generation article by Alan Zucconi which stated:

“Generating the biome map is the most critical phase, as it serves as the blueprint the rest of the world is built from... It all starts with a noise map—an image generated randomly—which features only two colours, representing land and ocean in a 1 to 10 proportion. The process is not dissimilar to rolling a D10 for each pixel: if you get a 1, it becomes land, otherwise it becomes an ocean.”^[5]

Importantly I noted the reoccurring theme of games using noise maps to help provide randomness in generation as efficiently as possible.

No Man’s Sky is another popular procedurally generated game with over 10 million copies sold. This Sci-Fi space adventure game uses procedural generation to create the feel of “endless” space and planets to explore with each planet having unique features.

To understand how this worked more I looked at the article Perlin Noise: The Evolving Algorithm Behind the Diverse Universes of No Man’s Sky written by Pratyaksh which stated:

“To expand the horizons of “No Man’s Sky” and breathe life into its worlds, the developers at Hello Games ventured to enrich the original Perlin noise algorithm with added complexity. Here are some essential modifications:

- 1. Layered Perlin Noise: Adding multiple layers of Perlin noise creates a multi-dimensional effect on planetary landscapes. By blending several noise functions with varying scales and amplitudes, the algorithm gives rise to intricate features, like rugged mountain ranges contrasting with serene valleys and expansive plains. This layering effect heightens the realism and depth of the generated terrains.*
- 2. Octaves and Persistence: In their pursuit of realism, the developers introduced the concept of octaves to Perlin noise. Octaves represent the number of noise layers applied to a specific point on a planet. By carefully adjusting the persistence parameter, which determines each octave’s amplitude, the team balanced coherence and randomness, leading to distinct biomes and ecosystems.*
- 3. Fractal Nature: Inspired by the beauty of natural landscapes, the developers infused fractal properties into Perlin noise. Leveraging self-similarity and recursion, the algorithm enhanced the ability to create realistic environmental patterns, such as coastlines, river systems, and other natural formations, bolstering the authenticity of the game’s universe.”^[6]*

Although I may not be creating a universe these three techniques are extremely useful as I can adapt them to my own project for example by changing the Octaves and Persistence of the Perlin noise I will be able to create distinct biomes.

2.4 Optimization of procedurally generated content

This sub section will be used to see what techniques are available to optimise game performance by dynamically changing the level of detail mesh terrain based on CPU Usage.

The two methods discussed in the paper are:

- Moving Average
- Level of detail

“The Method Moving Average adds a limit to the divisor value where the value is not directly obtained from the total amount of data in units time. With the limitation of the value of the divisor makes the smoothed data can be controlled. Limitation can be determined according to needs where the greater the value of the delimiter”

This method is important as it's used for refinement. It allows for the contiguous checking of adjacent values so that you can take an average value from the nearest point in order to provide a trend estimate. As a result, it removes potential data defects and smoothens out trend components. You can decide how much you smoothen because *“Each average consists of $2k+1$, the greater the k value the more average and smoother the MA estimate.”*

This method is definitely useful as it directly helps me get towards my aim of providing high level of detail with as much efficiency as possible.

“LOD algorithm can be interpreted as reducing the number of points, for example, a map has a length of 9 points and then simplified to 6 points so that the number of triangles that will be formed will decrease and make a surface so less detailed this aims to reduce CPU load and make a tile more efficient.”

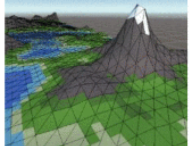
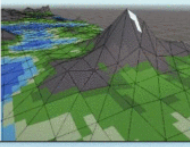
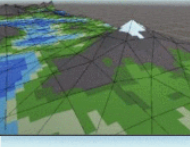



Visualization Map	Level
	Level 1 Number of points = 241 Number of lines = 345600
	Level 2 Number of points = 121 Number of lines = 172800
	Level 3 Number of points = 81 Number of lines = 115200
	Level 4 Number of points = 61points Number of= 86400
	Level 5 Number of points = 49 Number of lines = 69120
	Level 6 Number of points = 41 Number of lines = 57600

Figure 4 Example levels of details

This image helps simplify how level of detail works and overall level of detail algorithms are used to simplify the number of points so that fewer polygons are formed meaning a reduced amount of rendering. Many games use level of detail to provide a better experience in the form of graphical settings. Implementing LOD algorithms can mean that the project can be accessible to all systems.

The paper also touches on further points such as:

- Noise Generation
- Use of Perlin Noise
- Meshes
- CPU Usage

2.5 Different uses for procedural Generation.

This sub section will be used to explore ways in which procedural generation can be used in game.

Looking at the Procedural Generation: An Overview article these were some of the many possible uses^[8]:

- Creating Levels
- Terrain/Landscapes
- Animation
- Dialogue/Storylines
- Object Instantiation
- Loot System

After looking at this list and expanding Procedural Generation in level creation is seen in 2D games such as rogue-like games which were discussed in section 2.1.

For terrain and landscapes many methods can be employed. As seen in sections 2.2 and 2.3 the use of Perlin Noise is very important in creating realistic randomised Terrain. It can then be mapped to a mesh in order to make 3D structures which allow for a unique and expansive world.

Procedural generation can be used in animation in order to generate large amounts of a general animation in order to save the user time. For example when generating a concert procedural generation can be used to make simple animation for the mass of a large crowd which would otherwise be time consuming.

“Often to create a unique playthrough, games will procedurally generate the dialogue. Moreover, the decisions the player makes in the game may affect the storyline and allow the storyline to generate its own content based on how the character responds to a certain event.”

Many games use PCG for object instantiation. Major examples of this is games such as Subnautica using it for generation of particles in the water as the user traverses the depths of the ocean. It can also be used to generate “enemies, animals, trees” and much more.

“Often quest-based games will use an algorithm to create a loot system. This algorithm may generate loot based on the player’s current level. If the player has a high rank, the generated loot will likely be more rare as opposed to that of a low ranked player.”

Overall procedural generation is so beneficial to the gaming industry, as seen by this small list of possible implementations it can be used to make almost anything.

2.6 Limitations of procedural generation

This sub section will be used to understand the possible limitations of procedural generation in order to help give me an understanding of what it is able to do.

According to the limitations can be broken down into:

- Quality Control
- Hardware tax
- Generate unplayable maps

“Computers may be faster at crunching numbers than us humans, but there's one thing that we're vastly superior at, and that's creativity. No matter how amazing the procedural algorithm is, you lose the human touch. The little changes and subtleties that a seasoned designer can bring to a project are sacrificed.

It also means that you can't guarantee the same gameplay quality to all players. Some players may generate a really great map that facilitates gameplay, while others may generate a map that actively prohibits it.”

When speaking about Quality Control the most relevant part of this would be the inability to provide the same good experience for all users. I will have to assure that there is the highest chance of being able to have a good experience despite what can be potentially produced.

“These algorithms can be intense and require a lot of computing power. If you develop a game that makes heavy use of procedural generation, you need to ensure that a regular consumer PC or console is able to meet its demands.”

When speaking about Hardware Issues this is directly relevant as I am looking for a way to provide a high level of detail while having efficient processing. I will look further into how to manage this in the development section.

“When generating a 3D terrain map, you may accidentally generate a terrain that is too high for the player to climb or blocks off an area that needs to be accessible. The same goes for a 2D map. Later in this book we'll be generating dungeon rooms randomly. So, for example, we need to ensure that each room has a valid entrance and exit.”

Lastly in order to prevent generation of unplayable maps I will make sure that the parameters for the generation of the terrain allow for the highest chance of the biome being produced to be usable.

2.7 Research Summary

Overall, my findings provided me with a very good understanding of procedural content generation and how I might be able to implement it into my application.

A short summary of points of research I have directly considered and will use in my project is provided below:

- The use of Perlin Noise to create a map that the mesh can use
- Differing levels of detail through the use of simplified polygons
- Creating levels using procedural generation
- Meshes and how to implement them
- The importance of managing procedural generation due to high processing cost

3 System Design

This section will be used to describe in detail the design solution developed to meet the aim of the project. This will be done through the modularisation of the overall project and breaking it into requirements; Evaluating those requirements and making a design that will successfully meet all of them.

3.1 Requirements

- The project should allow for the procedural generation of a biomes.
- Should allow the user to traverse the terrain to see it procedurally generate.
- The project should allow the user settings for controlling the level of detail provided to provide system efficiency.
- The project should have multiple methods employed for the generation of realistic terrain.
- It should provide a friendly graphical user interface.

3.2 High-level design

To begin my design, I started by identifying the major components and their impacts on the overall design. A key aspect in order to reach my aim was for the user to be able to control multiple possible options in the game to maximise the efficiency and level of detail. I will do this by splitting up the algorithms that generate the meshes, map, Player controller, textures and UI. These then should be put together to provide the overall project in which the user will be able to traverse a biome of their choice that will procedurally generate.

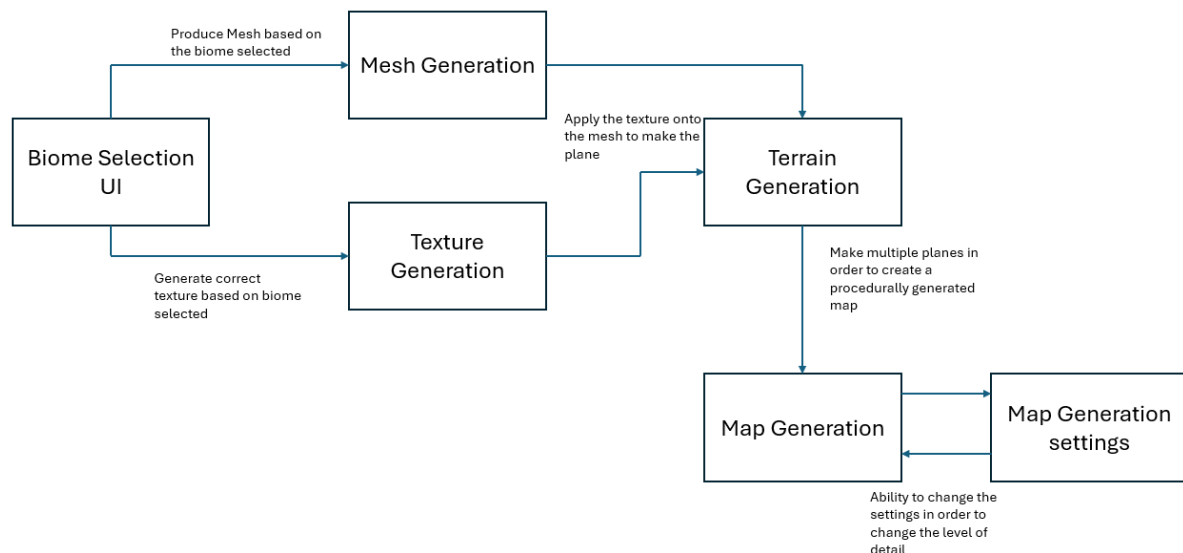


Figure 6 High level system design

A small description of the high-level components are below

Biome Selection UI: This is where the user will begin and will provide a selection of biomes they are able to explore. Depending on which biome is selected this will influence how the other components generate their values.

Mesh Generation: This component is used to represent where the mesh will be generated. All properties for the mesh such as height will be decided here.

Texture Generation: This component will be used in order to generate all the colours and textures that each specific mesh will have. These values will be based on the height of the mesh.

Plane Generation: This component is the combination of the mesh and the textures in order to create a coloured realistic looking plane

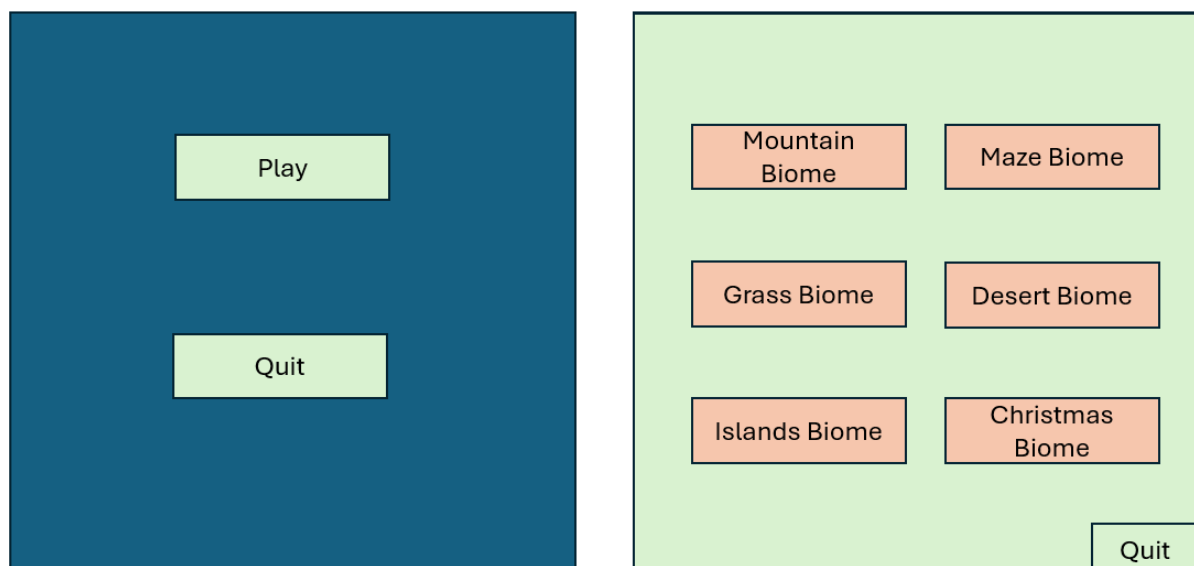
Map Generation: This component will be used to create the full biome. It will place planes in a way that they are all connected and generate as the user moves

Map Generation Settings: These will be the settings used in order to change the possible levels of detail In order to maximise efficiency of the system

3.3 Biome Selection UI Design

This will be the first page the user sees, it will be a friendly, aesthetic, and easy to use interface that allows the user to explore and access different biomes I have created, this will be done using a series of buttons and images. Upon selecting the user will be taken to the scene that corresponds to the user's choice, they will be able to instantly traverse this biome once it has loaded.

Below is a plan for approximately what this UI could look like:



3.4 Mesh Generation Design

The mesh generator is responsible for making a randomised mesh that can be manipulated based on multiple parameters.

This will be done through the generation of a noise map such as the example in Figure 3 and then using that to raise it into a 3D mesh by allowing it to displace the vertices in the mesh based on the values of the noise map. After extensive research the parameters for the generation of the mesh can be broken down into:

- Noise Scale
- Octaves
- Persistence
- Lacunarity
- Seed

3.4.1 Noise Scale

Perlin noise is the algorithm that will be used to create the noise map. This is because Perlin noise is often used as it most accurately mimics naturally occurring hierarchical structures and as a result is perfect for biome generation.

The Noise scale parameter will be used to dictate how far away the sample points be from each other. The larger the noise scale the closer the sample points so the height values will change less frequently and vice versa.

This is perfect as I can create diverse terrains with a plethora of mountains or a flat landscape with long grass all dependant on how I adjust the noise scale. This can also be related back to the research carried out in section 2.3 that encourages the user of Perlin noise.

3.4.2 Octaves

Octaves are a method of adding depth to a noise space. This is done by adjusting the number of scales of noise to add. The more scales added the finer the detail; However, if you overuse the octaves parameter the noise can introduce high spatial frequencies which can lead to distortion and errors.

A potential way of implementing this is by adding a slider so that I cannot overuse the octaves' function.

This will be useful in creating varied forms of the same landscape, for example a rocky mountain face or just a simple plane mountain face.

3.4.3 Persistence

Persistence is used as a multiplier in order to adjust how fast the amplitude of each octave diminishes. Increasing the persistence function means that the range of values that the Perlin noise can take increases making it rougher.

This paired with octaves will allow me further customisability of the biomes I am creating and allow for an expansive world.

3.4.4 Lacunarity

Lacunarity determines the change in frequency between the octaves. This means that as you decrease the lacunarity the coarser the noise and as a result you get larger and more visible structures. As you increase you get more uniform noise.

This can be used in order to create a variety in the amount of a specific terrain. For example, in this singular mesh you can vary the number of mountains that appear, if you want a few large mounts or many small rock faces.

3.4.5 Seed

Seeds are a random generated number used to not only provide reproducibility of results. As setting a set seed number returns the same pseudo random numbers in the noise.

This will also mean that although a player can select the same biome, they will never traverse the same exact one allowing for the difference in biomes being their distinguishability from each other and not recognition due to having identical patterns.

3.5 Texture Generation Design

The texture generator is responsible for applying textures to the polygons in the mesh to make it look realistic and not grey scaled. This will be done through the manipulation of the colour of the mesh based on the height between 0 being the floor and 1 being the maximum point at which the polygons lie.

To maximise my ability to create unique biomes layers should be able to be made and each layer should be customisable with these parameters:

- Texture
- Tint/Colour
- Tint Strength
- Start Height
- Blend Strength

3.5.1 Texture

This parameter will allow for me to add a Texture in the form of a PNG to the layer. This PNG should be duplicated and stretched such that they look as realistic as possible.

This should allow for improvements in the aesthetic of the generation as custom art which can be made or outsourced can be added to the generation and procedurally generated.

3.5.2 Tint/Colour

This parameter will allow for me to pick a hue for the layer which can be either used in conjunction with the Texture or as a replacement for it.

3.5.3 Tint Strength

After having selected a colour, this will be how strong it shows up. This will not only allow you to pick different shades of colours but also allow me to pick a colour that I can adjust in ways to compliment my texture thus improving the level of detail.

3.5.4 Start Height

Start height will dictate the beginning of the layer I have chosen. The layer will then end either at the start of the next layer or if there are no layers above it at 1. For example, water can be from heights(h) $0 \leq h < 0.20$ and then grass can be all heights $0.20 \leq h \leq 1$. This will allow me to create elaborate meshes that I can then use in order to create highly detailed biomes.

3.5.5 Blend Strength

This design of parameters was good so far until I realised most layers would look unrealistic as they would dramatically cut between things such as grass and water and dirt. So, this blend strength parameter will blend between 2 adjacent layers, the larger the value of this blend strength parameter the more it will blend.

This will provide a more natural look to all textures and tints while providing an increased level of detail.

3.6 Terrain Generation Design

With the Mesh plotted against the noise map and the textures for the colours added the Terrain Generation will be used to add the height to the mesh which will also enable the layers of the texture to operate.

For now, the Terrain generator can be simply broken down into a few parameters:

- Mesh height curve
- Mesh height Multiplier

3.6.1 Mesh Height Curve

The mesh height curve is very important as it will dictate where start height for the layers should be in proportion to the map and how fast the terrain transforms based on the height. It allows for the edges to be rounded off and transition correctly between layers, so I can control whether it is a sharp or smooth transitions between the layers.

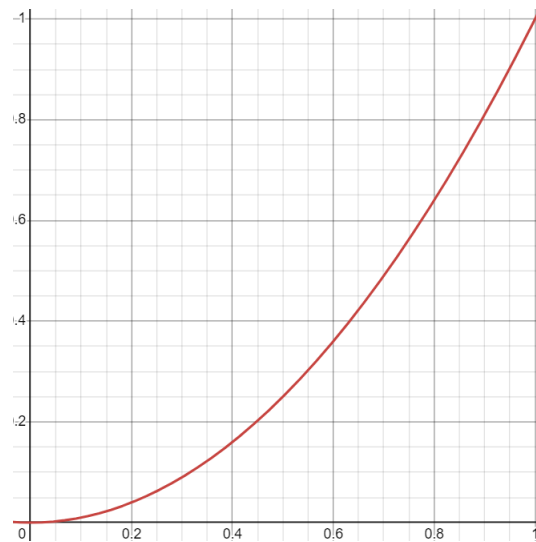


Figure 7 Example Graph for Mesh Height Curve

3.6.2 Mesh Height Multiplier

This parameter will be used to multiply each polygon based on the grey scale they are over. Since each point in the noise map is on a grey scale between 0 – 1, the polygons above those points will be multiplied by the mesh height multiplier parameter to form the heights and shapes of the mesh. This will result in a 3D shape which can finally be used to create the biome.

3.7 Map Generation Design

Map generation is one of the most important parts of the project as it puts all the techniques together. This will place meshes all around in accordance with the viewers current position, they should be able to do a 360 and see the biome in every direction. The user will also be able to walk in any direction and it will generate more terrain as they traverse the biome, in order to keep efficiency, it will hide planes when the user is too far and show them again when the user is close enough.

Another major thing the map generator will provide is a seamless connection between meshes. This is because if meshes are just randomly generated and thrown together then there will be holes in the map so the generation should occur with the respect to the meshes around it.

3.8 Map Generation Settings Design

Map Generation Settings will be used in order to manage the parameters that would affect the processing of the biome. These can be split into these different parameters:

- Render Distance
- Level of Detail
- Chunk size

3.8.1 Render Distance

This parameter will be used to dictate how far the generation will occur away from the users' point of view. It will allow for the feeling of "endless" terrain without having to process the entirety of the biome and allowing for smooth gameplay.

3.8.2 Level of Detail

Similar to popular games that allow you to control the level of graphical detail, this parameter will allow me to control the level of detail for the meshes that are produced. This will allow me to measure the trade-off for system processing for level of detail, which in most cases as you increase the level of detail you increase the amount of processing needed.

having to process the entirety of the biome and allowing for smooth gameplay.

3.8.3 Chunk Size

Chunk size is used to dictate the overall size of a singular mesh. Having this as a parameter would mean that I can test what size would be best for each biome in terms of processing efficiency as it wouldn't affect the level detail because all meshes are joined together so the user wouldn't be able to tell.

4 Implementation

This section will be used to explain how I was able to implement the creation of 6 biomes, Mountain, Maze Mountain, Christmas, Desert, Grass and Islands. each able to be procedurally generated and using a variety of procedural generation techniques for each individual Biome which will all be explored in this section.

4.1 Implementation of the Mesh Generator

In the end when creating the mesh generator is a combination of section 3.4 and 3.5. Mesh generation controlled by the class *MeshGen* which uses a combination of the NoiseData script TerrainData script in order to create a singular mesh that I can edit and manipulate before turning it into a biome.

This is done by getting the Data from the Noise map created and then mapping polygons appropriately by height based on the grey scale of the noise. This is all explained in detail below.

4.1.1 NoiseData

This script oversees the creation of a noise map using the Perlin Noise function. This noise map can be viewed through a plane and is adjustable by many different parameters. These parameters consist of choosing between local and global normal, which are touched on in more detail in section 4.6, Noise Scale, Octaves, Persistence, Lacunarity, Seed and offset. These parameters were all explained in the System design section but with these parameters now added I was able to make Noise Maps that would in future be used to create 3D meshes of terrains such as mountains. Having such a large potential for customisability meant I could test out a lot of different NoiseMaps and how they would look for each biome.

Examples of a few of these NoiseMaps the biome they represent with settings described can be seen in these example figures each represented with a plane in Unity3DEditor:

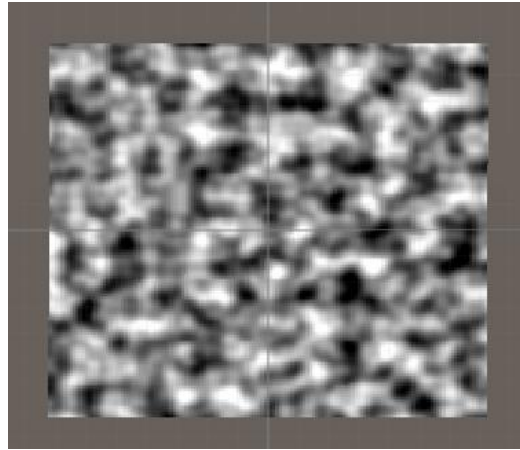


Figure 8 NoiseMap for Mountains Biome (Normalise Mode: Global, Noise Scale: 12.45, Octaves: 2, Persistence: 0.54, Lacunarity: 2, Seed 0)

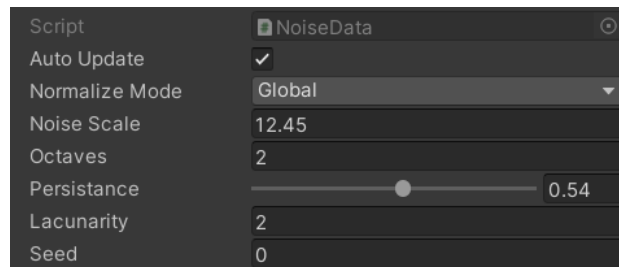


Figure 9 NoiseMap for Mountains Biome parameters in editor

As explained in section 3.6 “Seeds are a random generated number used to not only provide reproducibility of results. As setting a set seed number returns the same pseudo random numbers in the noise.

This will also mean that although a player can select the same biome, they will never traverse the same exact one allowing for the difference in biomes being there distinguishability form each other and not recognition due to having identical patterns.”

Meaning that although I have used the same values because the seed is different you can see a noticeable difference between Figure 10 and 8

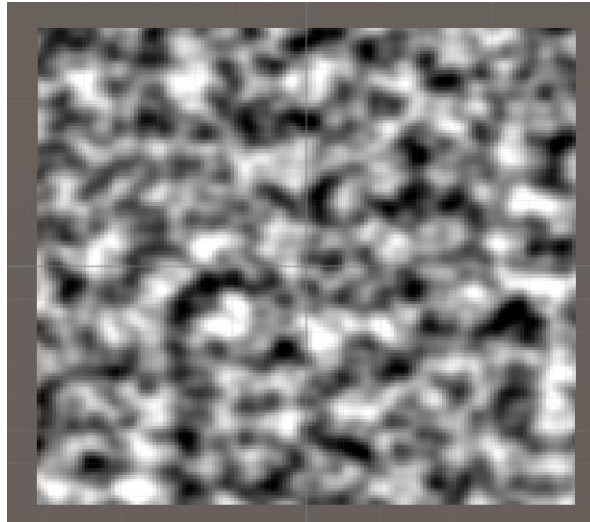


Figure 10 Same NoiseMap for Mountains Biome (seed: 24)

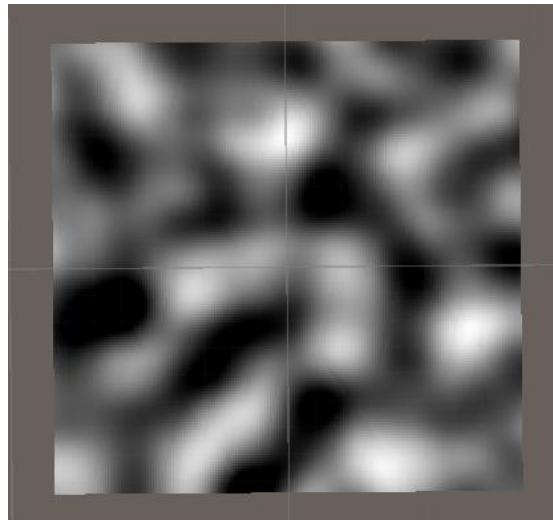


Figure 11 NoiseMap for Desert Biome (Normalise Mode: Global, Noise Scale: 29.68, Octaves: 2, Persistence: 1, Lacunarity: 1, Seed 0)

This next example is a noise map in conjunction with a falloff map in order to create secluded islands. This will be further discussed in detail in the next section.

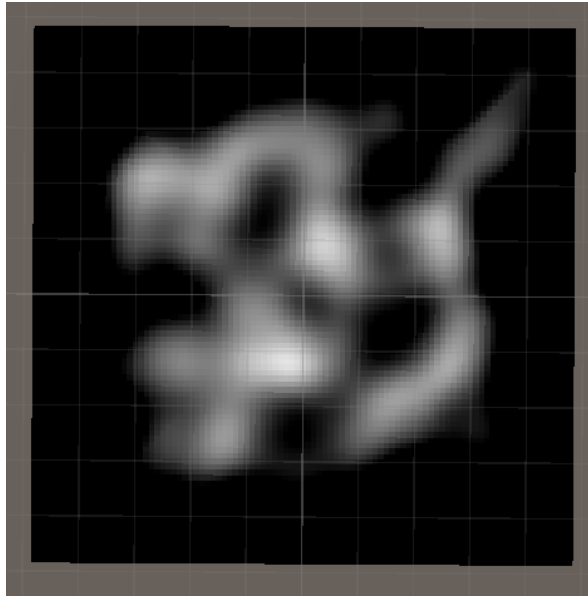


Figure 12 NoiseMap for Desert Biome (Normalise Mode: Global, Noise Scale: 16.44, Octaves: 2, Persistence: 0.88, Lacunarity: 1.15, Seed 4)

4.1.2 Terrain Data

This script oversees the parameters used to influence the mesh based on the Noise Data. The parameters that can be adjusted consist of Uniform Scale, Use Flat Shading, Use Falloff, Mesh Height Multiplier and Mesh Height Curve. Uniform Scale allows you to change the size of the mesh as a whole.

Mesh Height Multiplier allows for the multiplication of the height based on the grayscale value of the point the mesh is at, all shades between white and black are placed from 0-1 (grey scale) and dependant on the number acquired for example black being 1 then for figure 11 you would multiply that point by 10 in order to acquire its correct height.

The mesh height multiplier levels out the terrain in the instance of the mountain terrain I wanted a smooth transition between the mountains and water which will be explored more in Section 4.2 when adjusting textures.

It also allows for the use of two methods I employed in order to increase the diversity of the meshes (Each will be explored more thoroughly in Section 4.6), Flat shading and Fall off maps.

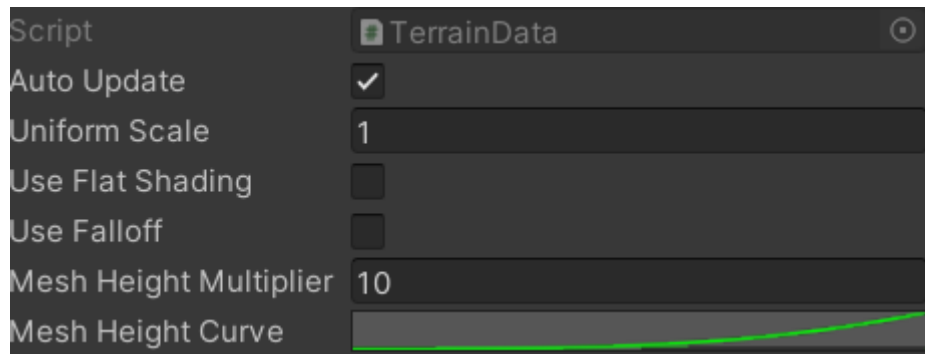


Figure 13 Terrain Data Parameters in unity editor

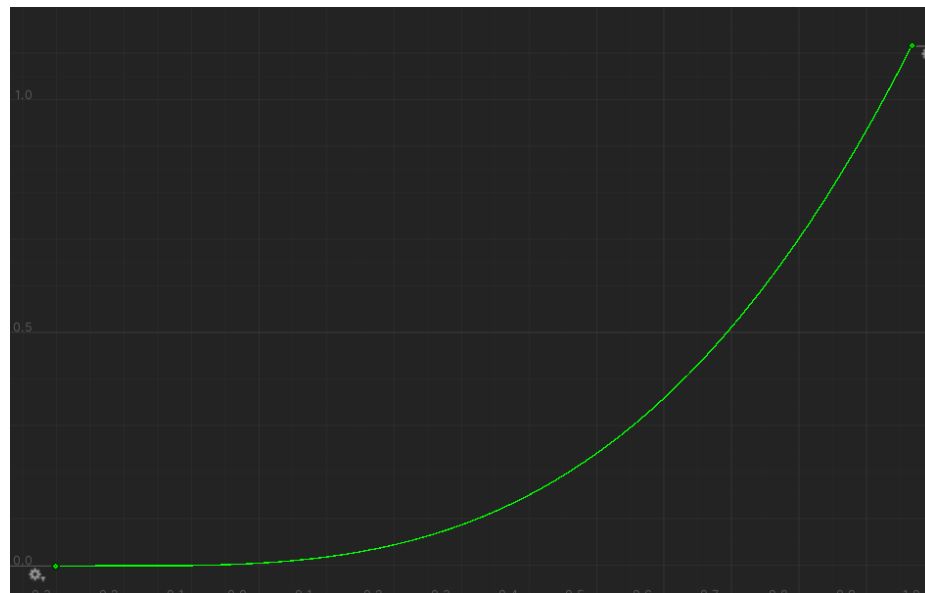


Figure 14 Mesh height multiplier expansion in unity editor

4.1.3 Mesh Generation

The *MeshGen* Script is responsible for putting the data provided by the terrain and the noise map and applying it onto a mesh. It does this by storing an array of points across the noise map; the vertices are also taken into consideration so that we can store each angular point of the polygon. This portion of code represents the creation of the polygons and vertex that are then used by the mesh generator to create a full mesh.

```

1 reference
public void AddVertex(Vector3 vertexPosition, Vector2 uv, int vertexIndex)
{
    if (vertexIndex < 0)
    {
        borderVertices[-vertexIndex - 1] = vertexPosition;
    }
    else
    {
        vertices[vertexIndex] = vertexPosition;
        uvs[vertexIndex] = uv;
    }
}

2 references
public void AddTriangle(int a, int b, int c)
{
    if (a < 0 || b < 0 || c < 0)
    {
        borderTriangles[borderTriangleIndex] = a;
        borderTriangles[borderTriangleIndex + 1] = b;
        borderTriangles[borderTriangleIndex + 2] = c;
        borderTriangleIndex += 3;
    }
    else
    {
        triangles[triangleIndex] = a;
        triangles[triangleIndex + 1] = b;
        triangles[triangleIndex + 2] = c;
        triangleIndex += 3;
    }
}

```

Figure 15 Creating Vertex and Polygon in c# MeshGen script

Furthermore an important factor that I had to come back to implement was the *borderVertices* and *borderTriangles* Array. This information needed to be in separate arrays to the main arrays because the *borderTriangles* and *borderVertices* were all the last points and angles at the edge of the mesh. This would then later be used in order to connect multiple meshes. As discussed earlier creating lots of random meshes wouldn't work for a full biome as there would be holes everywhere so creating a stored array of information that would later be used to link meshes was necessary.

After going over the stores of these polygons and vertices it then multiplies each point by the appropriate multiples based on the grey scale and mesh Height multiplier. By combining the two scripts these calculations are applied to a mesh and visualised. The results are shown in the figure below:

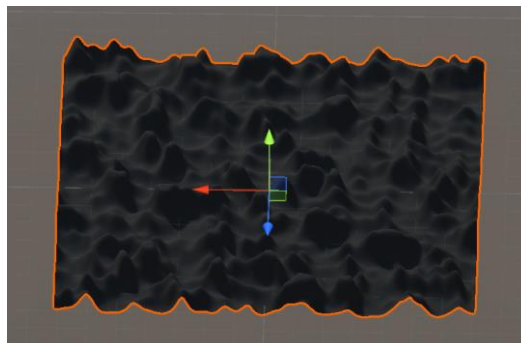


Figure 16 Mesh For Mountain Biome

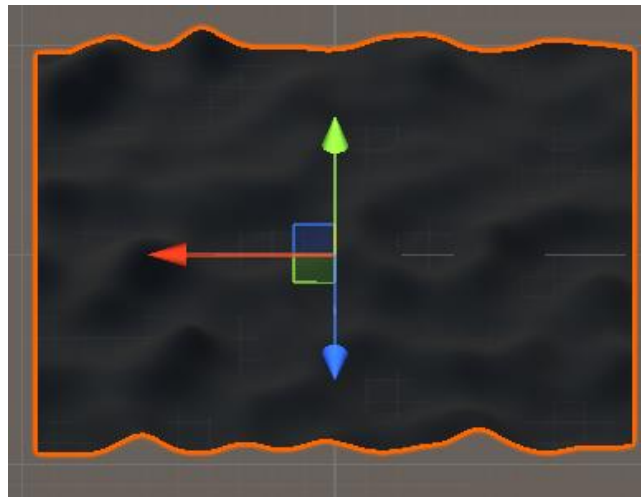


Figure 17 Mesh For Grassland Biome



Figure 18 Mesh For Maze Mountains Biome

4.2 Implementation of the Texture Generator

Meshes on their own look quite bland and although you are able to see the potential shapes but without any colours or textures you its feels bland and looks quite odd. The texture generator allows you to create an array of layers which each later having adjustable parameters that will affect the mesh. These parameters consist of Texture, Tints, Tint Strength, Start Height, Blend Strength and Texture Scale. All these settings are added to the Mesh based on the start height selected; There is no limit to the layers, so this allowed me to create a large variety of biomes due to the customisability of not only the texture but also normal colours.

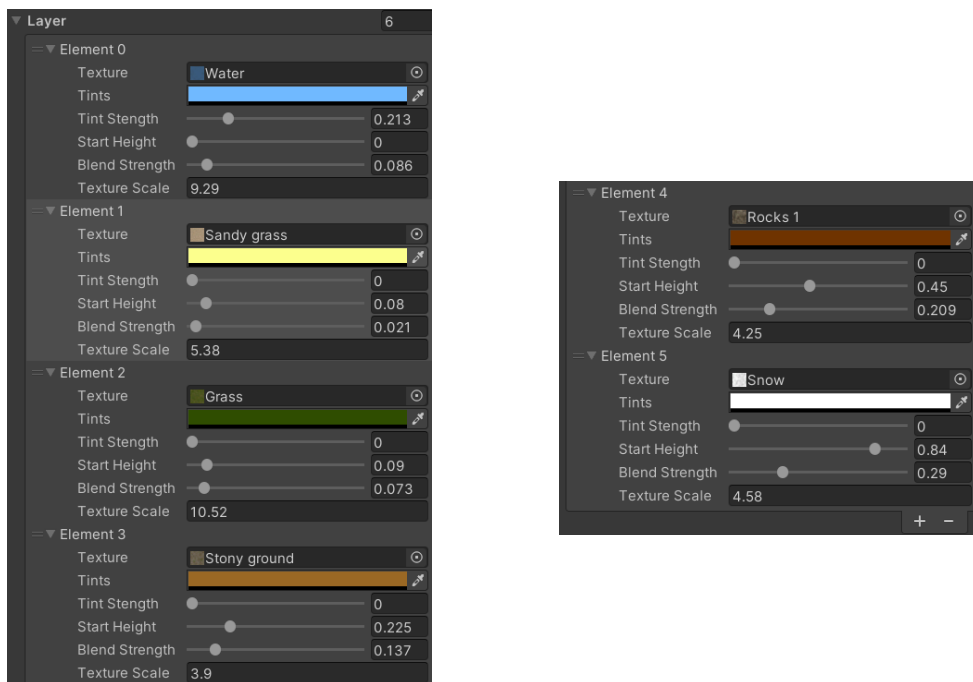


Figure 19 Applying and changing layers in the UnityEditor

```
[System.Serializable]
1 reference
public class Layer
{
    public Texture2D texture;
    public Color tints;
    [Range(0, 1)]
    public float tintStength;
    [Range(0, 1)]
    public float startHeight;
    [Range(0, 1)]
    public float blendStrength;
    public float textureScale;
}
```

Figure 20 Layer Class

```
2 references
public void ApplyToMaterial(Material material)
{
    material.SetInt("layerCount", layer.Length);
    material.SetColorArray("baseColours", layer.Select(x => x.tints).ToArray());
    material.SetFloatArray("baseStartHeights", layer.Select(x => x.startHeight).ToArray());
    material.SetFloatArray("baseBlends", layer.Select(x => x.blendStrength).ToArray());
    material.SetFloatArray("baseColourStrength", layer.Select(x => x.tintStength).ToArray());
    material.SetFloatArray("baseTextureScales", layer.Select(x => x.textureScale).ToArray());
    Texture2DArray texturesArray = GenerateTextureArray(layer.Select(x => x.texture).ToArray());
    material.SetTexture("baseTextures", texturesArray);

    UpdateMeshHeights(material, savedMinHeight, savedMaxHeight);
}
```

Figure 21 Applying this layer to a Material

This code is particularly important because when trying to understand the implementation for applying the parameters from the colours and textures was applying them to a material then using that material to render the mesh. Figure 21 shows the parameters that would store and taken from the layer being applied to the material. The material is then used to when rendering the mesh.

Below are examples of meshes produced when adding the material texture on top of the mesh through the use of a mesh renderer:

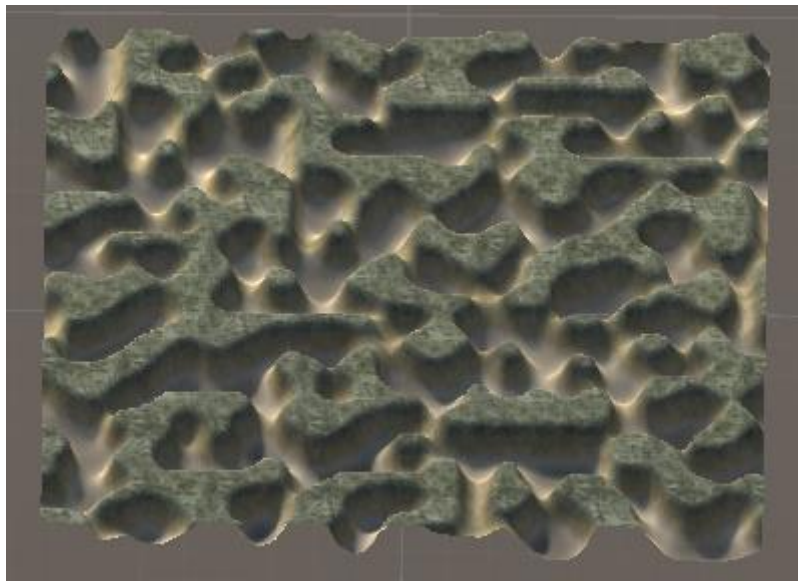


Figure 22 Maze Mountain Mesh with Texture.

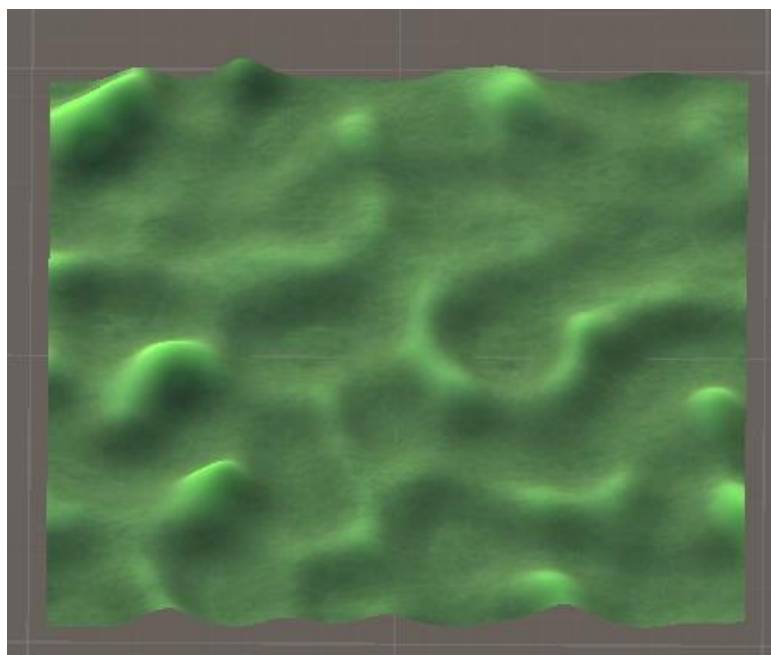


Figure 23 Grassland Mesh with Texture.

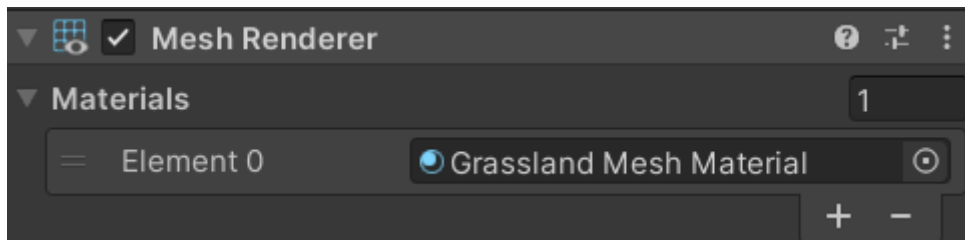


Figure 24 Grassland Mesh Material Showcase.

4.3 Implementation of the Map Generator

Map generation is controlled by the *Map Generator* script which controls the generation of the biomes as a whole. The possible parameters that can be changed are the DrawMode in editor, is what type of method I was wanting to visualise in the editor at that moment which could have been one of three things: A Mesh, A Noise map or a Fall Off map. It then took the data for the noise map, Texture and Terrain as an asset which will be explained in the next figure. The level of detail that the mesh is viewed in in the editor which will be further explained. Chunk size which was how large each individual mesh could be while generating the biome and flat shading chunk size which is how big the chunks could be when flat shading.

4.3.1 Editor Level of Detail

A huge part main aim of my dissertation is to “create a balance between level of detail and amount of processing.”. Similar to the research done in section 2.4 I wanted to implement options for the level of detail where each mode just simplified the mesh more and more. This is done by taking simplifying the lengths of the polygons by a factor of the level of detail, specifically the level of detail chosen using the table below.

LOD	Divisibility Factor
1	1
2	2
3	4
4	6
5	8

Table 1 Level of detail vs the divisibility factor

The equation for this is $(LOD - 1) * 2$ but if the level of detail is 0 then default it to 1. This now meant I have a potential for 5 different levels of detail and leads on to the importance of the link between the chunk size and LOD.

Since the level of detail modes are dividing the lengths of the polygons by the divisibility factor it would mean that the chunk size would have to be numbers divisible by all 5 of the LOD modes.

Unity only allows up to 255 size limit per mesh meaning that the chunk size could only be numbers that were divisible by all those LOD and between 1 – 255. In the end the array of numbers for the Chunk Size I used was 48, 72, 96, 120, 144, 168, 192, 216, 240 Which sat within an array called *supportedChunkSizes*.

Now with suitable Chunk sizes and different levels of detail I was able to see how this level of detail would look on singular meshes as seen below. This table also relates directly back to the research done on level of detail in section 2.4.

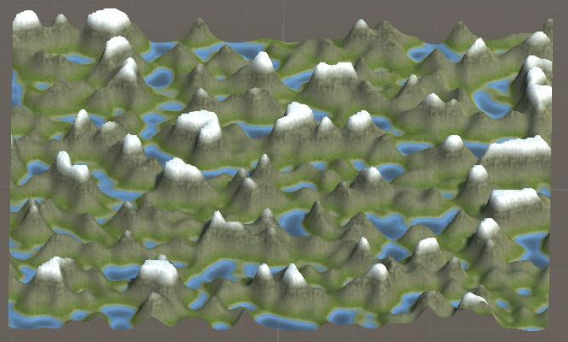
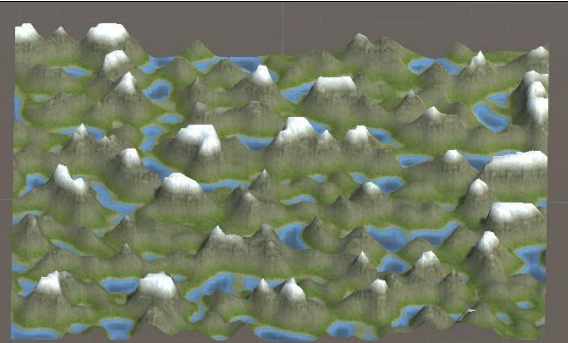
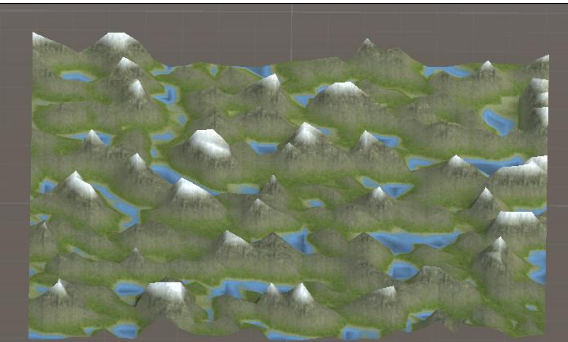
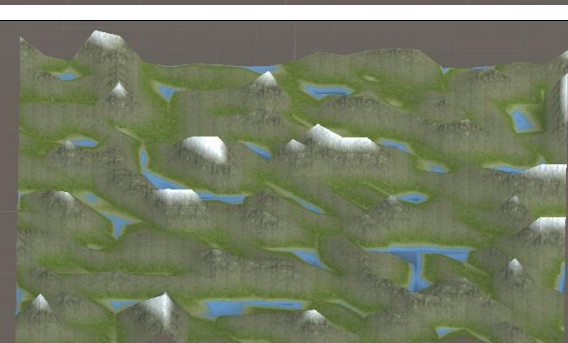
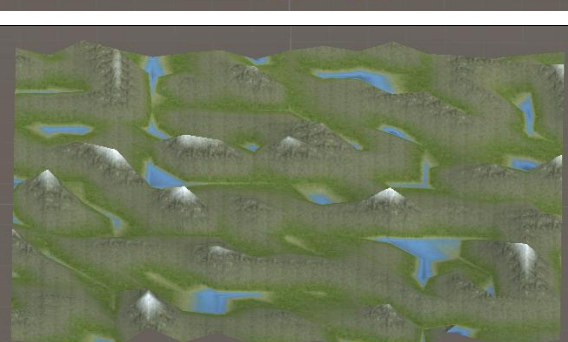
LOD	Mountain Mesh
1	 A 3D visualization of a mountain mesh at LOD 1. The terrain is composed of many small, sharp peaks and valleys, rendered in a greenish-grey color with blue highlights in the valleys. The mesh is dense and detailed.
2	 A 3D visualization of a mountain mesh at LOD 2. The terrain is composed of many small, sharp peaks and valleys, rendered in a greenish-grey color with blue highlights in the valleys. The mesh is dense and detailed.
3	 A 3D visualization of a mountain mesh at LOD 3. The terrain is composed of many small, sharp peaks and valleys, rendered in a greenish-grey color with blue highlights in the valleys. The mesh is dense and detailed.
4	 A 3D visualization of a mountain mesh at LOD 4. The terrain is composed of many small, sharp peaks and valleys, rendered in a greenish-grey color with blue highlights in the valleys. The mesh is dense and detailed.
5	 A 3D visualization of a mountain mesh at LOD 5. The terrain is composed of many small, sharp peaks and valleys, rendered in a greenish-grey color with blue highlights in the valleys. The mesh is dense and detailed.

Table 2 Level of detail vs how it affects Mountain Mesh

As seen as you go down the different level of detail modes you can see a noticeable difference. This knowledge was then used to implement the render distance which is explained in section.

4.3.2 Data as Assets

As said previously the different potential assets that can be made are for the noise map, texture and terrain as these are what are passed to the map generator. It was important for me to split these up as it allowed for the easier generation of different biomes. I could stores the values I created for each biome and or mesh and then when it was time to load the correct world I could just use the assigned assets and it would know how to create the correct biome.

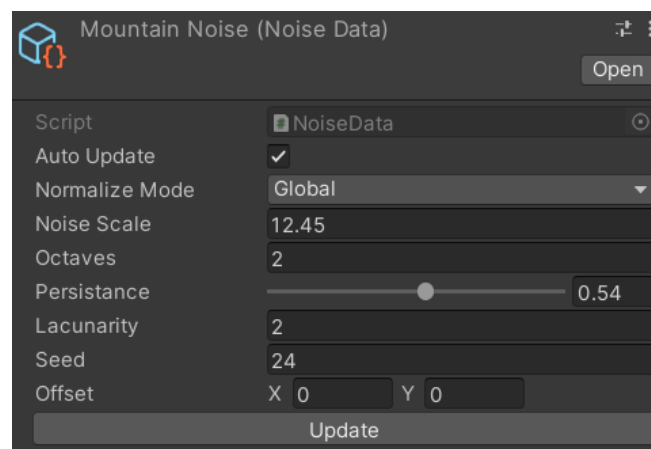


Figure 25 Mountain Noise Data Object as an Asset

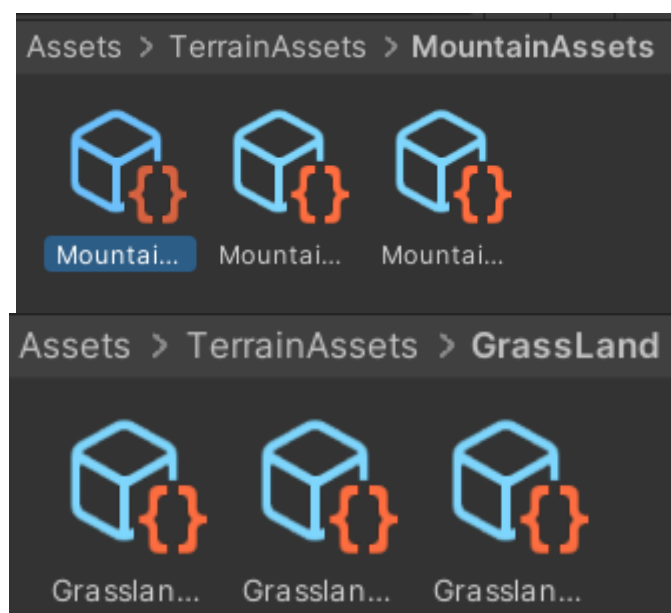


Figure 26 Example of storing Mountain Biome Assets separately to other Biomes.

4.4 Implementation of the Map Generator Settings

The map generation settings are accompanied by an *Endless Terrain* script that is in charge of creating the planes with the correct render distance which means placing planes in a certain distance around the viewer with distance thresholds from the viewer for the level of detail for these planes, this was done by having an array that would store the amount of different distance thresholds needed. Then each layer in the array allowed you to pick its Lod from 0 to 4 (zero included) with the distance it would surround the viewer by.

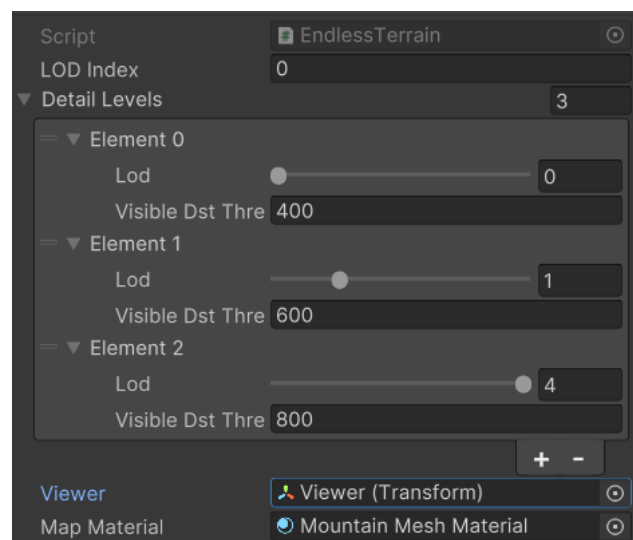


Figure 27 Mountain Noise Data Object as an Asset

Another important implementation that was discussed was the material for the mesh. The endless terrain takes the material to create meshes, meshes are then created around the viewer based on other meshes so that they join up without having gaps.

If the viewer moves from a spot, then the mesh adapts and creates more meshes on the outside but it doesn't delete the old chunks it just hides them. This allows you to move along an "endless" terrain without significant processing issues while also seeing that it's the same world being explored.

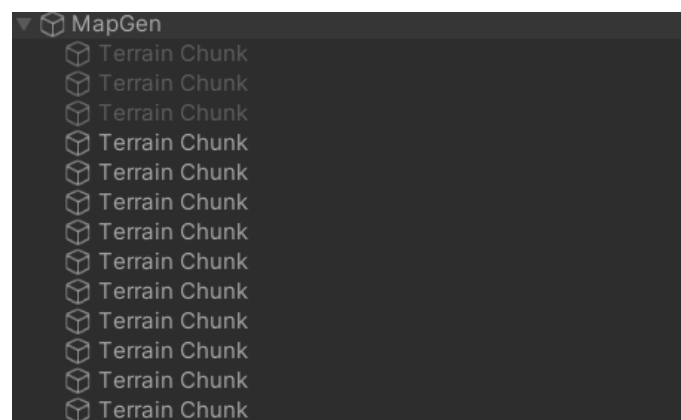


Figure 28 Terrain chunks being hidden and shown

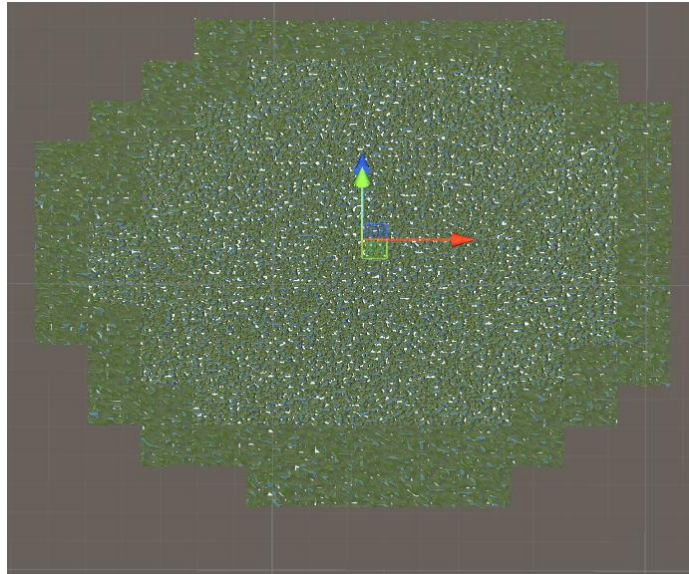


Figure 29 Mountain Biome around the viewer zoomed out

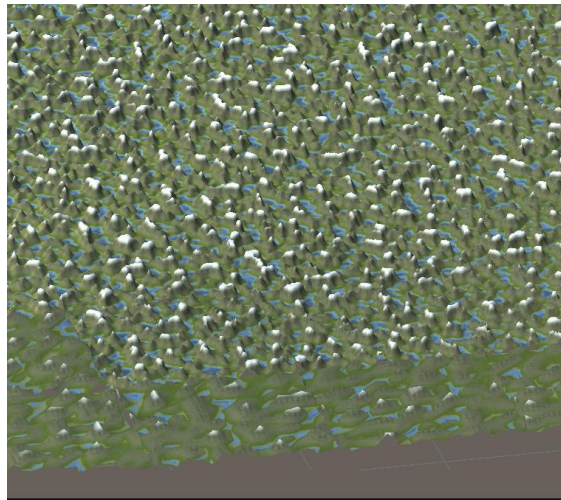


Figure 30 Mountain Biome around the viewer close up

4.5 Implementation of Biome Selection UI

The biome selection UI was easy to implement. It consists of a background image with buttons that take you to the relevant scene based on what you have pressed. This was done to add ease of use for the game and a nice aesthetic. Buttons are also highlighted on hover and press to help the user understand that they are selectable.



Figure 31 Main Menu



Figure 32 Second Menu Screen After pressing Play



Figure 33 Button Highlight after hover

4.6 Implementation of Extra Technologies

This section will be used to explain extra technologies that were added that were not in the original design.

4.6.1 Fall off map

A fall off map in the context of procedural generation is a map going from 0 to 1 just like the noise maps just all the higher values are in the centre and the outer values are all 0. This is then used in conjunction with the noise map so that only the values for the noise map near the centre will appear and then they will fade as they get towards the edges. This inspired me to create a new biome with this technology which would just be plenty of isolated small islands as nothing would generate around the centre.

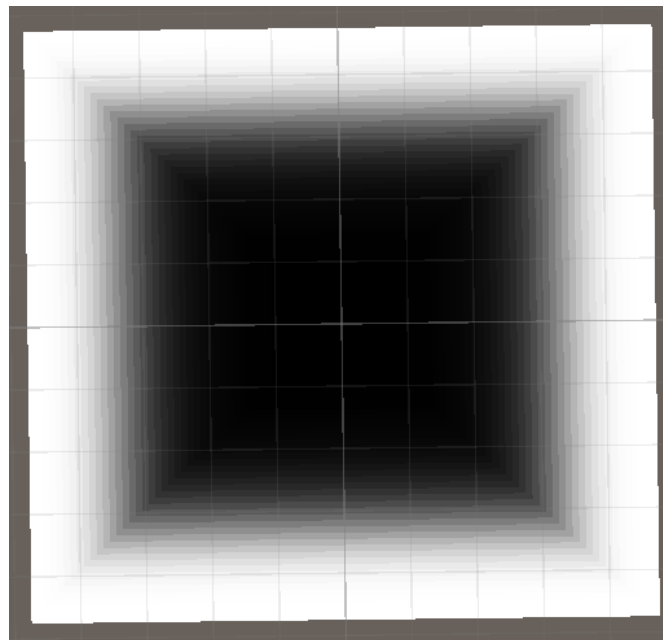


Figure 33 Fall off map by itself

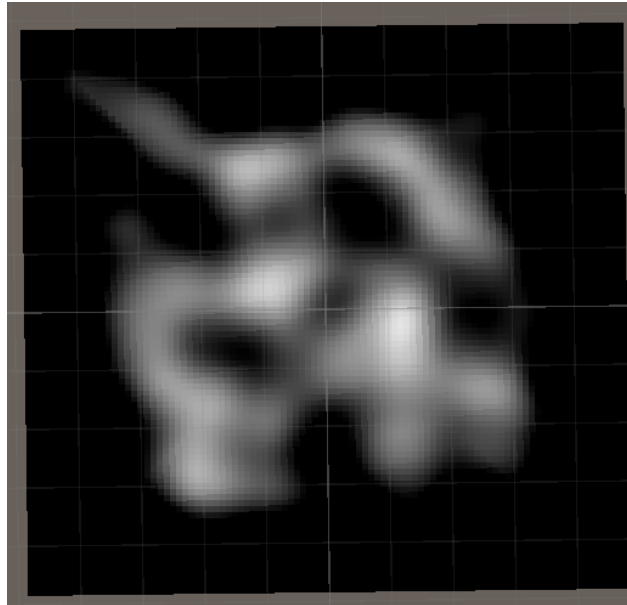


Figure 34 Fall off map with noise map

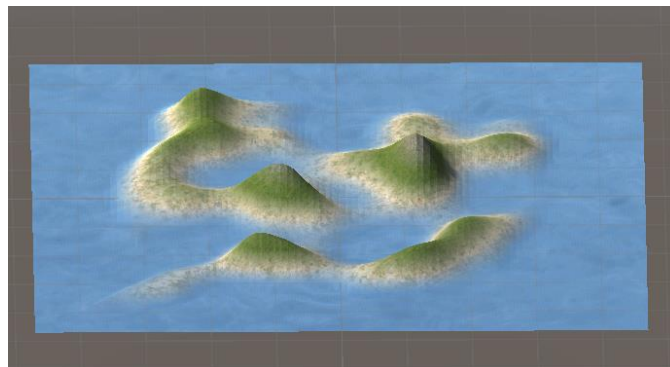


Figure 35 Resulting islands Mesh with Fall off map

4.6.2 Flat Shading

Flat shading is a shading model which calculates the illumination of the environment at a single point for each polygon; This means that that the colour is the same for the entire polygon. This will remove smoothness and may look worse but can improve system efficiency and will be very valuable to explore.

This was also added to the islands, the idea was to create a biome that showcased the new skills with high a high level of detail and the best processing as it would be the most simplified of all the biomes, I removed the texture to make it clearer as seen below.

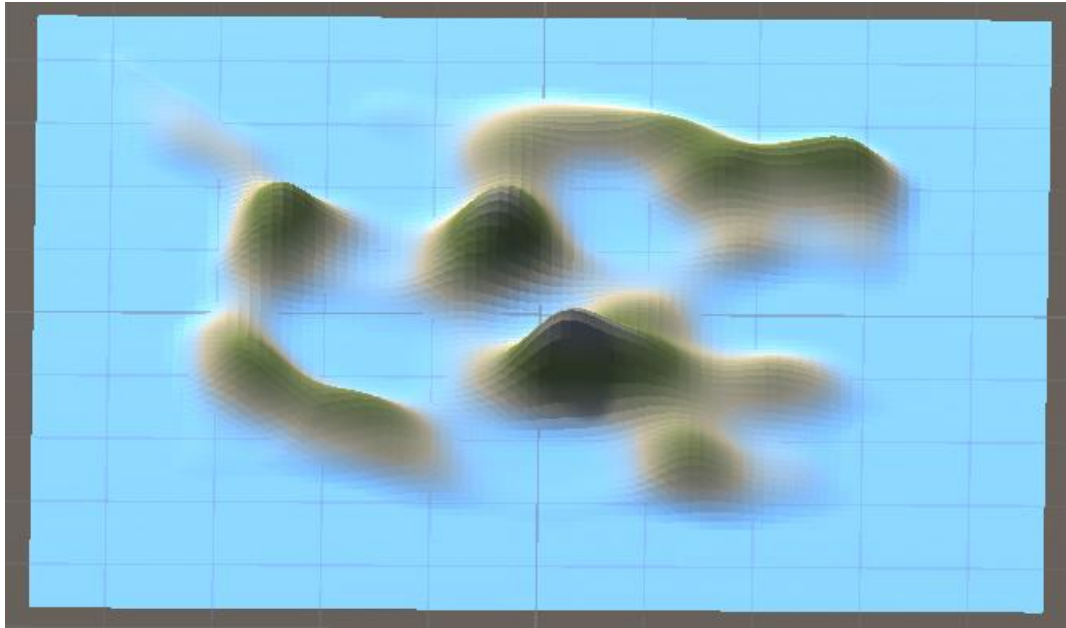


Figure 36 Resulting islands Mesh with Flat Shading

5 Testing

This section will be used to discuss appropriate testing strategies for the project and how these strategies were then executed.

5.1 Testing Strategies

To ensure that appropriate testing of my project is done black box testing was carried out:

- System Usage
- Environment testing
- Black box testing
- Load Testing

It should be noted that to keep to the no ethical impacts of my project all user testing was conducted on close friends and family members and as a result won't have an affect or change to my current ethical considerations.

5.1.1 Environmental testing

The game being able to run on multiple platforms was an important feature of this application. Allowing for it to run consistently over multiple operating systems would increase the accessibility of the application. To test this, it was installed and run on the systems shown in table 3.

System Name	MacBook Pro	Windows	Linux
Operating System	Snow Leopard	Windows 11	Fedora 8
Processor	2.4 GHz Intel Core 2 duo	AMD Ryzen 5	Intel
Memory	2gb	8gb	512mb
Graphics	Nvidia Geforce 8600M GT	RTX 3050	Nvidia Geforce 6400
Hard Disk	160gb	256gb	30gb

Table 3 Systems used for testing the application

5.1.2 System Usage testing

Testing was completed using a Windows computer with specs shown in the table below. The in-built Unity system application profiler was used to check the memory and the process Usage while the application was running. Other than Unity there were no other applications running on the system while the tests were being carried out. The laptop was plugged in. The mountain scene was used for all testing with no added changes from the figures shown in section 3. Each test will be carried out multiple times with different levels of detail on order to see the effects on the system.

All the biomes were made using different techniques. The grasslands and Desert biomes were made in a way that the only differences are the difference in the size of the generation, so a comparison has to be made whether there is a noticeable difference

aesthetically and on the system. Lastly the islands biome was made using two techniques that would decrease the strain on the system, so comparisons need to be made to see if these techniques worked successfully.

5.1.2.1 Memory Usage

Biome generation requires a lot of memory as it is a very intensive operation. It is important to measure how much was needed to better apply my application for consumer use.

5.1.2.2 Processor Usage

Biome generation requires a vast number of calculations to be done in order for the biome to be generated. These calculations can be seen in the profiler and exactly how much it is affecting the system.

5.1.3 Ease of use

Analysing the ease of use of the application was done by asking the users to perform specific actions on the application. The time taken for each task was then averaged and the most relevant feedback was taken.

5.1.4 Load Testing

A menu is provided before having access to each biome. It is important to test that from the start menu you can access all the biomes and each load correctly. Furthermore, each world allows the user to explore the biome and see it generate; the results of which will be explained.

5.1.5 Full build test

All testing so far was completed in the unity editor. It is important to see how it would run as a standalone application and if the experience is the same as the editor.

5.2 Testing Results

5.2.1 Environmental testing results

The application worked on all 3 main operating systems with no problems.

5.2.2 System Usage testing results

Using the unity profiler I able to see the CPU and memory usage of the different biomes. The main comparisons I wanted to make were between similar biomes that used different techniques. The first biomes I wanted to check were the grassland and deserts, this is because apart from colour their only other difference was the render distance and I wanted to see the effect on the memory of this change. Unity profiler allowed you to see in detail the effect on the memory and exactly what was using it.

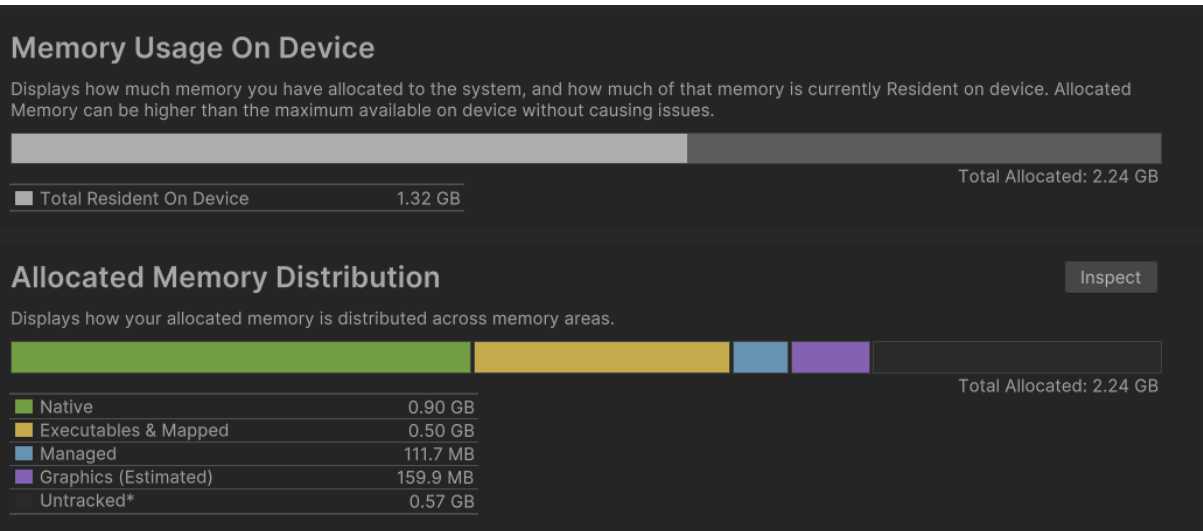
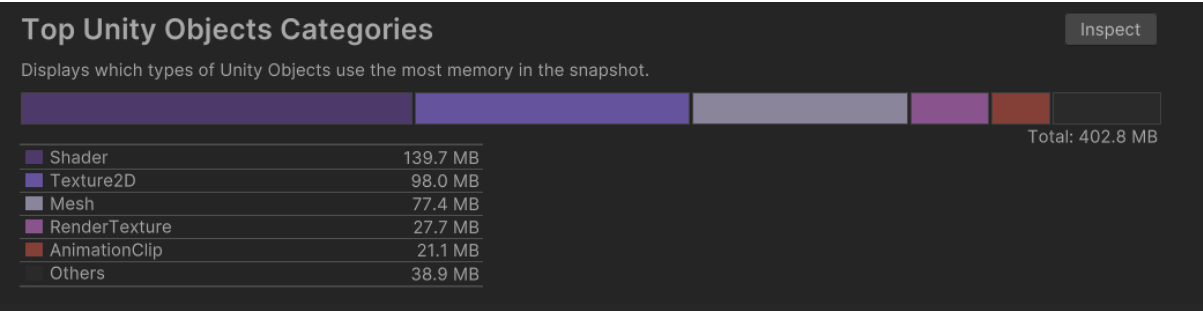
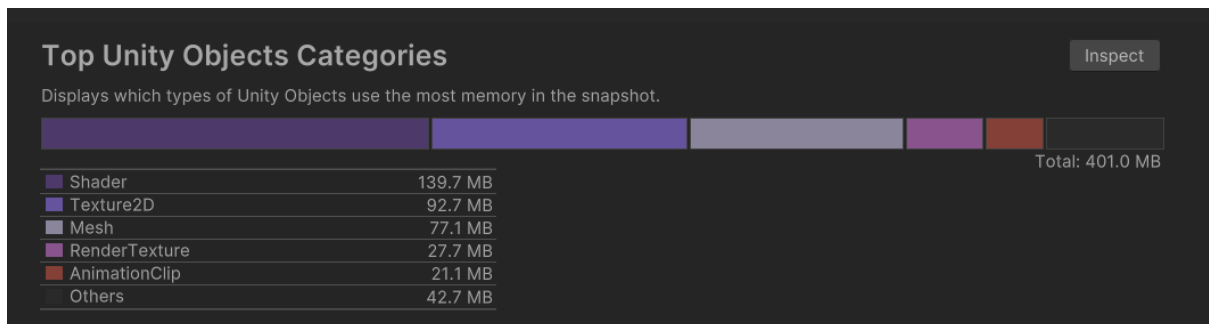


Figure 37 Results of Memory for desert biome



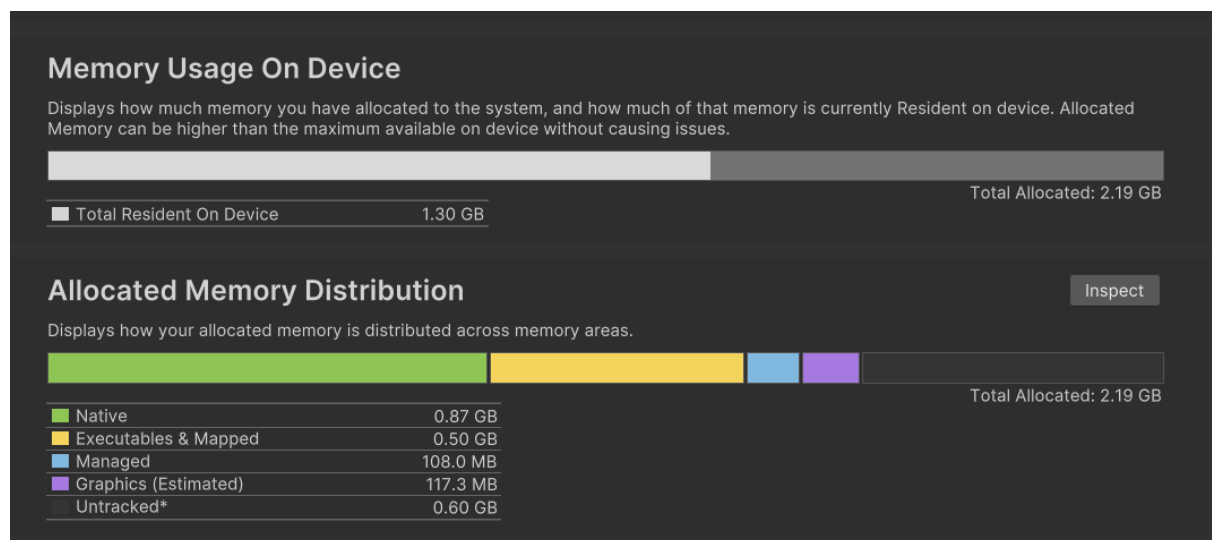
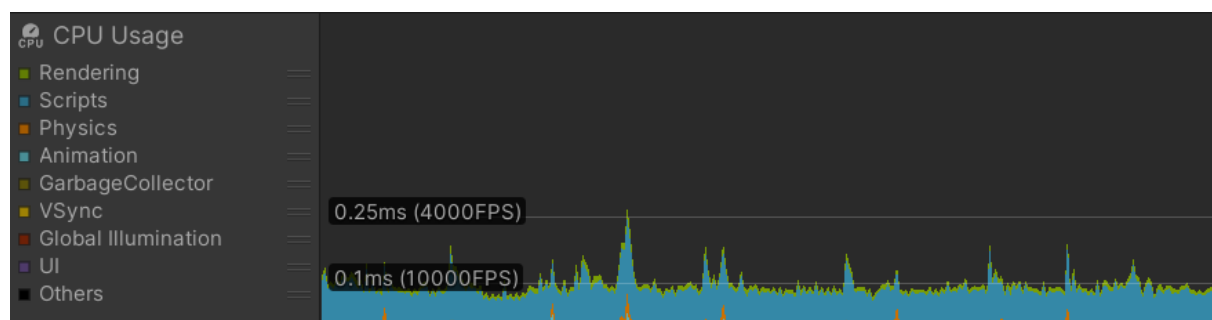


Figure 38 Results of Memory for Grassland biome

Comparing the two the results seem to suggest that there was not much difference in the amount of memory each was using. Despite the fact that the grasslands had a reduced render distance which can lead me to assume that over time even with the reduced render distance the strain on the computer is almost identical. This would be contrary to the research done in section 2 and will be gone over in the evaluation.

Mountains



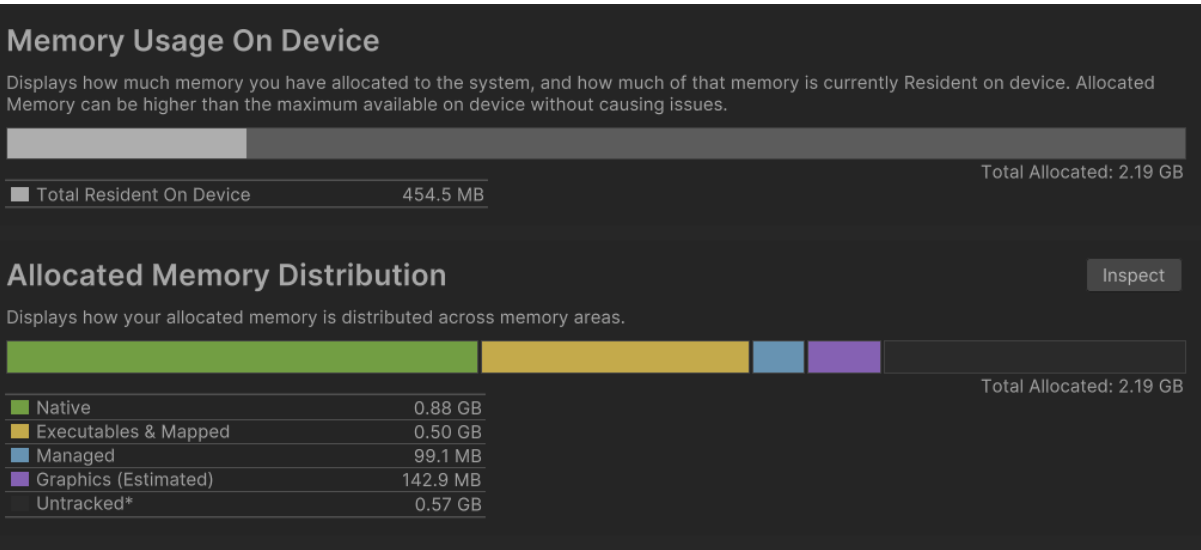


Figure 39 Results of CPU and Memory for Mountains biome

Maze Mountains

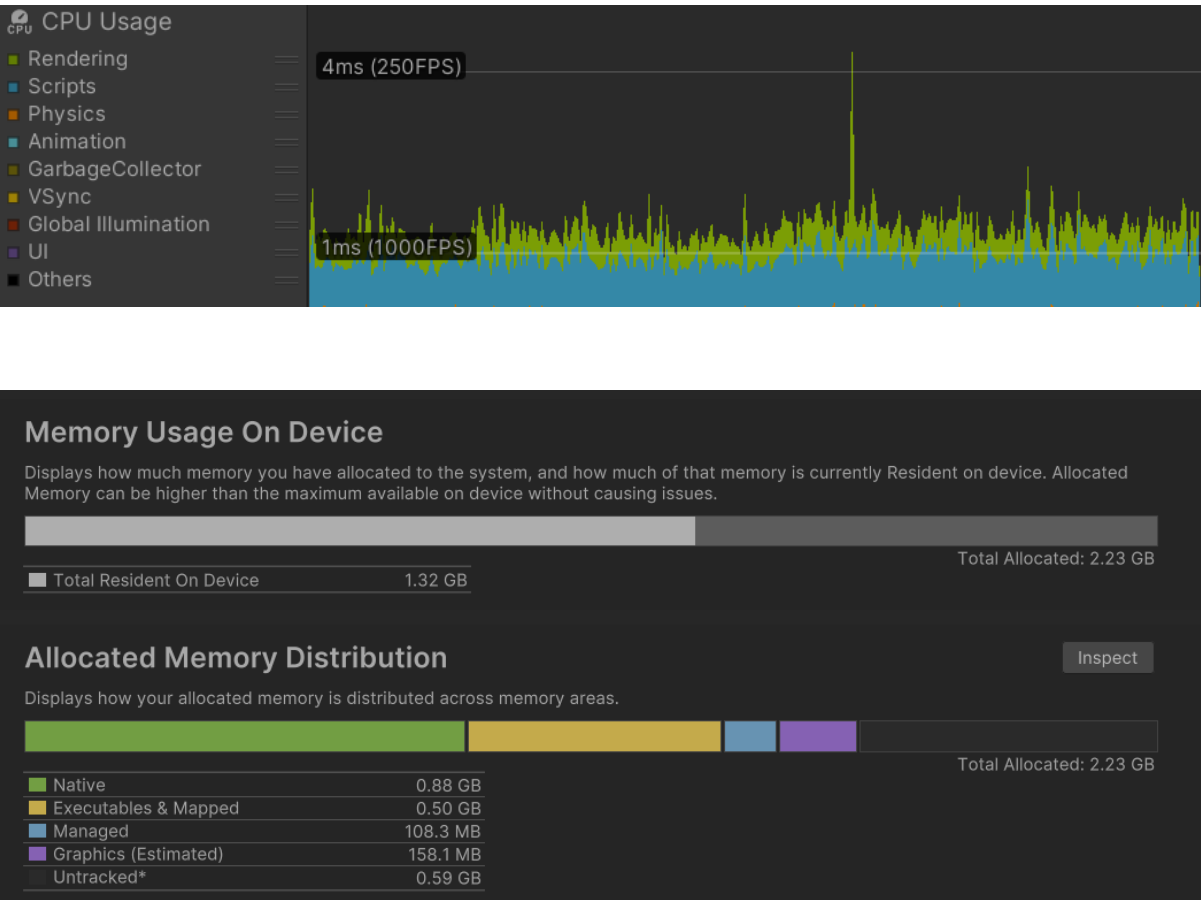


Figure 40 Results of CPU and Memory for Maze Mountain biome

When comparing the next tow similar biomes which were the normal mountain biomes and the maze mountains. A large discrepancy can be seen in the spikes in CPU and the amount of memory usage the maze mountains used in comparison to the regular mountain biome. The spikes seen in the CPU are from rednering new chunks, as seen in the figures above for rendering there was a 4ms spike in comparison to a small 0.25 ms spike in the regular mountain biome. Furthermore playing the Maze mountain biome used aorund 1.32gb of total memory whereas the regular mountnain biome used around 452mb of total memory.

I believe this discrepancy was caused by how much larger he maze mountains are in comparison to regular mountains all caused by a decrease in the octaves when generating the biomes.

Islands

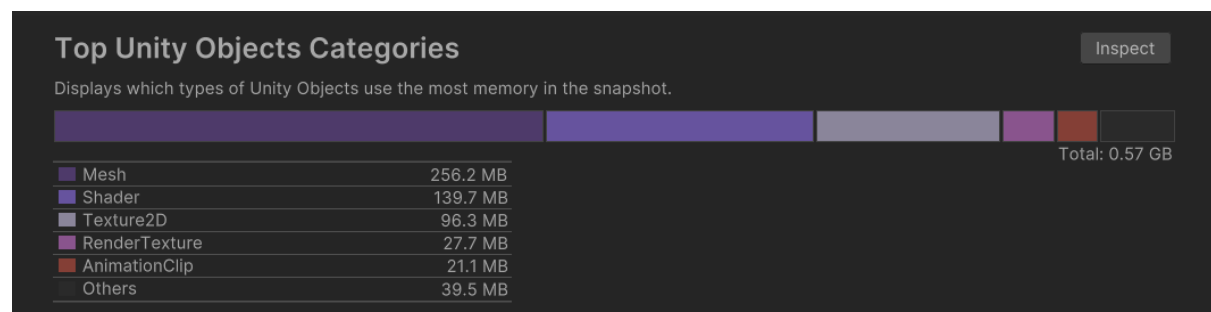


Figure 41 Results of Memory for Island biome

I also wanted to run a quick test on the memory usage of the islands. Comparing this result to that of the grassland and desert it shows that the objects for the island biome used almost 170mb more than the other two. This result was very confusing as I used flat shading a process meant to reduce the load on the device; This was again contrasting to the research I had done in section 2 and will be further analysed in the evaluation.

5.2.3 Ease of use testing results

Action Performed	Expected User Behaviour	Average Time Taken	Overview of User Comments
Press Play	Click the big play button in the centre of the screen to start the game.	<5 seconds	Was very easy as the play button is easily visible
Play Mountain Biome	Go from the start menu and successfully select the mountain Biome in order to play it.	<30 seconds	After pressing play the different biomes were easily distinguishable so it was quite easy to use
Play Desert Biome	Go from the start menu and successfully select the Desert Biome in order to play it.	<30 seconds	After pressing play the different biomes were easily distinguishable so it was quite easy to use
Quit Game after selecting a biome	Press the quit button while playing	<150 seconds	This was quite difficult as I didn't know escape had to be pressed before being able to press quit
Quit game in Menu 2	After pressing play press quit	<10 seconds	The quit button was immediately apparent.

Table 4: Ease of use testing

5.2.4 Load testing results

Biome	Actual Output	Decision
Mountains	From the biome selection menu, the button to select the mountain Biome works correctly. All textures and assets loaded correctly and it runs smoothly. The user is then taken to the mountain biome and is able to look around and explore the biome.	Pass
Maze Mountains	From the biome selection menu, the Maze Mountains biome button works correctly. Loaded slightly slower than the average biome but all textures loaded correctly and it ran smoothly. The user is able to look around and explore the biome	Pass
Grasslands	From the biome selection menu the Grasslands biome button worked correctly. Loaded very fast and all textures loaded correctly including added fog. The user is able to traverse and look around the biome	Pass
Islands	Selection of the islands biome was successful and the button worked correctly. The islands biome failed to load although it had been successfully loading in the unity editor and as a result did not pass the test. Fixes will have to be made accordingly	Failed
Desert	The desert biome is able to be selected from the biome selection menu. The biome loaded at moderate speed and all textures loaded correctly including the desert fog The user is able to look around and traverse the biome freely	Pass
Christmas	The Christmas biome is able to be selected from the biome selection menu. The biome loaded with moderate speed and all textures loaded correctly. The user was able to look around but was unable to move and explore the biome and as a result failed the test Fixes will be made accordingly.	Failed

Table 4: Load testing

Furthermore all fully loaded biomes can be seen in the appendix after having implemented these fixes.

5.2.5 Full build test

This test consisted of me using the unity editor to build my application into an executable .exe file in order to see if it would run as a game. This test revealed a major flaw in the application which was that textures from outside sources were not being rendered when it was turned into an executable. As a result, as seen in the figures below the biomes came out a bit off in comparison to the final result pictures seen below. Due to time constraints which will be further analysed in the evaluation I was unable to fix this but I am glad I this test brought it to light.

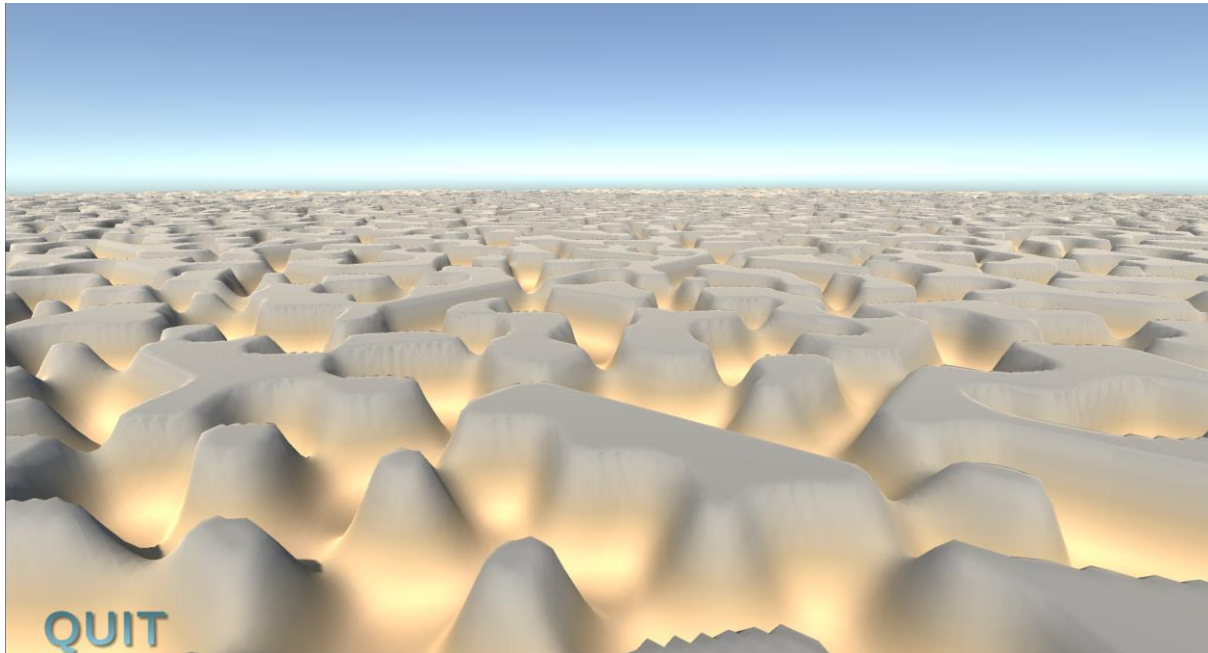


Figure 42 Results of colours only maze mountains biome

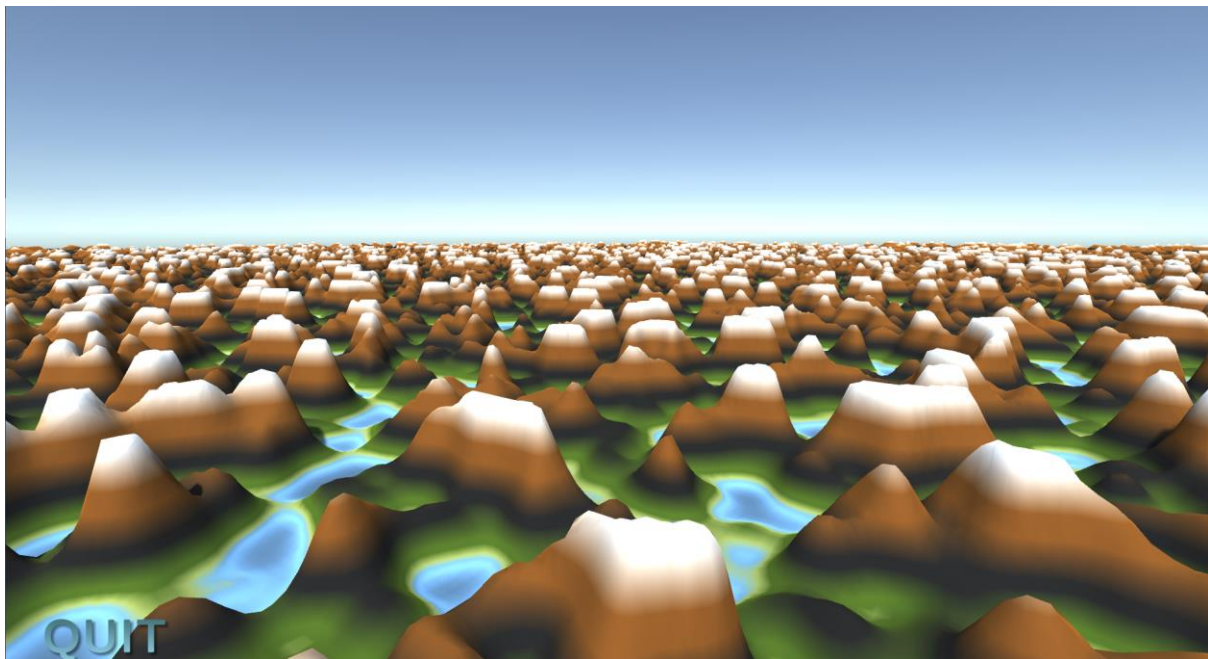


Figure 43 Results of colours only normal mountains biome

5.3 Testing Summary

Overall I believe that the test went well; This section show me the importance of testing as without having been through these test I wouldn't of realised how many potential issues could occur on a seemingly completed application. It also gave me good insight into how technologies interact with the system and how users in general interact with a new game.

6 Evaluation

After having developed the application its very important to evaluate the process that went into making it. As a result I will be reviewing each section of this dissertation with the purpose of find whether:

- The results for a given section was enough to satisfy the intended outcome
- The limitations of each section and how they can be improved
- Which objectives a given section fulfilled and was that specific objectives were fully satisfied.
- The pros and improvements of each approach.

After this section the dissertation will then be evaluated as a whole in order to see whether the aims and objectives as a whole were met including the feasibility of continuing the development and implementation of the application.

6.1 Research

A large variety of research was undertaken for this dissertation. The intended outcome of the research was to get a general overview of the topic of procedural generation while learning key techniques and methods employed by experienced professionals, but also achieving the objectives that this section would be able to fulfil. For the multitude of research areas I presented different research techniques were employed including:

1. Background reading:
This was a straightforward technique utilized to give general background knowledge on each of the research areas. Due to the large multitude of research done into procedural generation finding academic papers to read that were directly related to this dissertation wasn't very difficult.
2. Goal Oriented reading:
This method of research was applied in order to keep the research as relevant as possible. Although the titles of many papers can be interesting a lot of them deviate into things that are no longer relevant to the goal at hand; Although sometimes its okay to get a broad perspective on the topic as a whole. If all research was carried out without the goals in mind the research would end up being insignificant and as a result useless
3. Practical Investigation:
Many techniques were investigated throughout the course of the project. In order to get a best understanding of the techniques sometimes it was relevant to run tests for the techniques on a lower scale in order to confirm that they were in fact relevant to the project and would help the overall aim/objectives.

The objectives of this section were to:

- Explore and identify the current methods used in procedural generation and how each can be used.

- Identify the potential limitations of the technologies used in procedural generation.
- Analyse and summarize existing published examples of games that use procedural generation technologies.

The information acquired through my research allowed me to fully complete all three of these objectives. This is because the goal oriented research approach allowed me to focus on the objectives of the dissertation.

6.2 Design

The purpose of the design section was to provide myself with a clear structure on how my application could be made so I wasn't going at the problem blind. A lot of thought went into the design section, using the knowledge I had acquired from the research I split my design into a modularisation of the aims and objectives. I tried to cover as many points as possible.

6.2.1 Requirements Breakdown

I started by breaking down the requirements that my project would have to meet in order to satisfy my aims and objectives. These requirements would allow me to understand the specifications for my application which boiled down to:

- The project should allow for the procedural generation of a biomes.
- Should allow the user to traverse the terrain in order to see it procedurally generate.
- The project should allow the user settings for controlling the level of detail provided to provide system efficiency.
- The project should have multiple methods employed for the generation of realistic terrain.
- It should provide a friendly graphical user interface.

These aims were very clear and covered all the bases allowing me to then build an overview using a high level entity relationship diagram.

6.2.2 Entity Relation Diagram

An entity relationship diagram was used as a visualization of how I would tackle creating the different sections of my code. It showed the different relationships between methods such as texture generation and mesh generation and how I would have to make them interact. It also provided much needed ease as before this visualization procedural generation seemed like an impossibly large undertaking but with this came an understanding that the problem could be tackled and built up with hard work over time.

6.2.3 Design of the application

After completing a visualisation of the application I now had to break down the entities into more detailed and constructive sections that would allow me to have a direct pointer as to how to implement my application. This consisted of:

- **Biome Selection UI Design**
This section was as simple as making a simple aesthetic design for the UI that the user is greeted with before accessing the application. This was important in order to provide ease of use for the application and improve the overall immersive experience of the game
- **Mesh Generation Design**
This section provided a high level overview of the parameters that would need to be added in order to successfully generate a mesh. It built on the research that had been undertaken on noise and how it could be used to generate a realistic 3D mesh.
- **Texture Generation Design**
This section was used in order to find understand the parameters needed in applying colours and textures to the mesh. As important as the shape of the biome is, without colours or textures it would lose its realism so it was important to understand how to incorporate this in conjunction with the mesh generation design
- **Terrain Generation Design**
The design of terrain generator meant that the parameters would be in charge of making the now calculated mesh 3D and applying the textures and colours to that mesh in order to create a singular mesh chunk. I could then visualise this chunk in the editor before turning it into an overall biome.
- **Map Generation Design**
This section would be in charge of the endless generation of the now create mesh chunks. It was important to understand that I couldn't just create random chunks as their would be many gaps in my work so the map generator would have to be responsible for creating a link between the meshes.
- **Map generation additional settings design**
These settings would work in conjunction with the map generator. It would allow you to change many factors that influenced the level of details of the meshes. This was particularly important as it allowed me to tie my application back to the original aims and objectives.

Although this stage did not directly satisfy any of the aims or objectives set out at the start of this dissertation without the high level overview of how I was going to implement

my application. I would not have taken such a structured and technical approach to this dissertation.

6.3 Implementation

The purpose of the implementation section was to showcase how the application was created. This was done through a culmination of background reading and lots of testing. During development the overall picture built by the design section was always kept in mind. Changes were made from the overall design section and new features were added as follows:

- **Combination of the Terrain and map generator**
When implementing I realised I could streamline and improve upon the initial design I had suggested. This came in the form of allowing the map generator to control not only the creation of singular meshes but also the creation of the full biome. This saved a lot of time as I realised I would have been doubling my workload and doing similar processes for no reason
- **Addition of Assets and materials**
Through more research done while implementing I learnt that through the use of materials and assets I could save data from the scripts and form of the mesh. I could then use then to store the values and materials of individual biomes in order to generate them as I pleased.
- **Addition of new technologies**
This section I introduced Fall off maps and Flat shading which had not been discussed prior. Since research is ongoing even during implementation when I had found out about these methods, I decided to add them as they would help provide a more in depth view of methods and techniques and directly contributed to my aims and objectives.

This stage was important as it directly fulfilled the objectives of the dissertation, the ones completed were:

- **Implement and develop a usable procedurally generated game which showcases different biomes.**
- **Implement and evaluate at least 2 methods and how they affect procedural generation.**

By the end of the implementation section both these objectives were completed to a satisfactory level.

6.4 Testing

Lastly came testing, this was split into two sections. Testing strategies, where I thoroughly planned out all the test I wanted to carry out and then testing results where I explored the results of the tests and what would be done because of them.

6.4.1 Testing Strategies

This section was important as I needed to make sure that testing was structured and fair. It allowed me to plan an overview of how I was going to test my application and what technologies and or methods I could use in order to test this. This stage of testing went quite smoothly as planning didn't take too long because many of the ideas and technologies were readily available.

6.4.2 Testing Results

The results for testing went much differently than expected. I was initially very confident that my application was full proof and had zero to no errors but with the testing split up into the section shown below many fixable issues were discovered and resolved.

- **Environmental testing results**
This test was needed in order to make sure that my application was accessible to all potential users and all test passed.
- **System Usage testing results**
System usage test were important as they provided me with results on how the processing was being affected by my application. This directly ties to the aim of creating a level of balance between level of detail and processing. Overall, these results were unsatisfactory, this is because the unity profiler wasn't providing the detail I would've wanted when looking at CPU usage and due to time constraints I wasn't able to develop it more. It did provide good and relevant data on the memory usage, but this was contrary to all the base research I had completed initially so that left me feeling unsatisfactory with the way I approached this particular test.
- **Ease of use testing results**
This was mostly a way to get some much needed user feedback on my application. After working on something for so long it's easy to get tunnel vision and the feedback provided allowed me to take a step back and make minor improvements that I would've otherwise missed.
- **Load testing**
This was mostly general diagnostics test to check whether all the functions were working from start to finish. Again, it uncovered a few errors I had made which were promptly fixed.
- **Full build testing**
The full build test built on the loading testing in which I wanted to again test the full functionality of the application but as a standalone application outside of the editor. This uncovered a huge problem in which textures that I had used from a directory outside of unity wouldn't be added and loaded onto the executable application. This meant I had to substitute texture for general colours losing the

feel of realism that the textures got. Overall I am thankful for this test as without it the project would've looked completely underwhelming despite the hard work that went into it.

6.5 Overall evaluation

Reflecting back on the entire dissertation there were many merits that came with each section but also some points of improvement. They are as follows:

6.5.1 Research

- Pros
 - Background research provided a lot of valuable insight into the overall topic of procedural generation.
 - It allowed me to directly complete objectives set in this dissertation.
 - I learnt about technologies I would've otherwise never come across without this research.
- Improvements
 - Summarise research more
 - Increase the depth of research and go for quality of quantity of the research

6.5.2 Design

- Pros
 - Gave me a in depth idea of how the structure of my application would be made through modularisation of the overall problem.
 - Easy to understand requirements that tied back to the overall aims of the dissertation.
- Improvements
 - Constantly going over the design even after initial completion would allow more refined overall result as with design improvements comes easier implementation

6.5.3 Implementation

- Pros
 - Having a good base knowledge from research and design allowed for the smooth and thorough implementation of my application
 - Due to the way the application was implemented new technologies were able to be added as the project went on
- Improvements
 - Should have used version control and prototyping instead of only working on the completed version of the project.

6.5.4 Testing

- Pros
 - Due to concise planning and testing many errors were spotted and resolved before the end of the project.
- Improvements
 - Find better technologies in order to have more accurate tests.

7 Conclusion

After having designed and implemented my application and evaluated the process that went behind it. This section will be used to review the original aims and objectives and whether they were met. This will include a reflecting on my own abilities in achieving these objectives, alongside this will be a review into the skills and knowledge I acquired from this project. The work complete in this dissertation will also provide me with a reference to decide future developments for the project and where expansions or fixes can be implemented. Using my self-evaluation in conjunction with the research into future developments I can see the most beneficial technical knowledge for myself to develop.

7.1 Aims & Objectives

The original “To explore how different technologies that effect procedural generation of biomes and Analyse efficient uses of these technologies to create a balance between level of detail and amount of processing.” To achieve this aim I firstly undertook a large variety of academic and general research to broaden my understanding of the topic of procedural generation as a whole. The extent to which the research helped me can be seen by its constant references throughout the project and just how many ideas tied back to the original research.

To accomplish this, aim an application was made in Unity 3D that allowed for the procedural generation of a multitude of biomes. This implementation included a variety of techniques and methods that allowed me to create such a high-quality application.

However, it was demonstrated that the drawbacks of procedural generation came with the high costs to the system. There was a direct correlation with how detailed the meshes and biomes produced were with how smoothly the application ran; however, due to the methods learnt during the research section of the dissertation techniques such as different levels of detail, render distance and flat shading were implemented to help find a balance between processing and detail.

Overall, the biomes produced were all to very high quality and standard, but it is debateable whether I found the perfect balance between level of detail and amount of processing as they ran with differing efficiency. Therefore, I can consider my aim partially met as I was able to use a variety of techniques in or to efficiently create biomes using procedural generation but was unable to find an exact balance between level of detail and processing that satisfied me.

Objectives	Outcome	Explanation	Further Development
Explore and identify the current methods used in procedural generation and how each can be used.	Partially Achieved	The background research that was done in a way that the methods found were very well researched and documented in this dissertation. More methods were found after the initial research was already complete and implemented into the application. If they were important enough to implement, I should've also identified them at the start	In order to rectify this further research into Flat shading and Fall off maps needs to be carried out
Identify the potential limitations of the technologies used in procedural generation.	Achieved	During the research section potential limitations of procedural generation were explored and how they could affect my application. This was understood and taken into account throughout the development and implementation of the entire project	Look into ways to now prevent and overcome these limitations in an efficient way.
Compare how different technologies effect the balance between level of detail and processing.	Achieved	During the implementation section biomes were purposely created using different technologies so that when arriving at the testing section comparisons could easily be made and drawn about each technique. These were also further looked at in the evaluation section	Implement only the techniques that provide the best balance between level of detail and processing
Implement and develop a usable procedurally generated game which showcases different biomes.	Achieved	The application this dissertation is centered around allows you to view and explore 6 different biomes and is accessible on a variety of OS systems.	Improve the application through the release of updates
Analyse and summarize existing published examples of games that use procedural generation technologies.	Achieved	Extensive research was carried out on popular games such as Minecraft, Terraria and No Man's Sky and how each uses procedural generation techniques in order to make a fun and unique game. These techniques were then taken into consideration for my own project.	Try implement techniques seen in these games to expand the world provided by my application for example space exploration.
Implement and evaluate at least 2 methods and how they affect procedural generation.	Achieved	A variety of techniques were employed in the making of this application. The total resulting in much more than 2 and were all thoroughly analysed in the implementation section of this dissertation.	Maximize the potential of the currently used methods as many were employed but not used to their full extent.

7.2 Skills Learnt

After having researched into developing the application, most of the technologies and methods discussed were very complicated and new. Given this I had quite a limited background knowledge of how to tackle this problem; I had only programmed in unity 3D once before for a games programming module. This consisted of me making a first person shooter but other than some broad slight knowledge on c# none of the skills I had learnt in that module were applicable to this project.

The skills I learnt can be broken down into this list:

- Improved understanding of c# especially code structure
- In depth understand of the maths that goes behind procedural generation
- Enhanced understanding of the features that Unity 3D provides and how they can be used
- Improved problem solving as I can now understand methods to improve an already successful application
- How to create a functional and quality game from scratch

7.3 Future Developments for the Application

Working on the application for so long and building a higher understanding of procedural generation has allow me to understand that a range of possible new improvements and features can be incorporated to this application that have not been yet applied. Some of these are stated in the section below.

7.3.1 Allowing users to modify some aspects of generation.

A major improvement would be the modification of settings discussed in the implementation such as render distance. Depending on the render distance the user has to generate more meshes and as a result slowing their system down. Being able to adjust this live while in the game would be a huge improvement to the application and allow users to quickly and efficiently access the game in a way that suits their system.

7.3.2 Adding detail and animation to textures.

After some added research it is possible to create simple animations for textures such as water by just moving the offset of the texture. Another potential improvement can be adding things such as reflections to certain texture such as mirrors and water. These two potential improvements can increase the level of detail provided and give a more human touch to a game mostly generated by the computer.

7.3.3 Adding the generation of multiple biomes simultaneously.

Currently in the game you select a singular biome that you want to “endlessly” generate. For the overall improvement of the application, I believe taking biome generation like

Minecraft will allow for a better experience for the user. This is because all the content is available at once but it also increases the potential uniqueness of the world.

Things that would have to be considered are % rates at which certain biomes spawn. If a new biome is spawned the chance of it spawning again should increase to not just have random singular meshes of random biomes spawned everywhere. Lastly how to transition the meshes between different meshes.

With my current implementation I believe it's possible as all meshes are stored in a material so the only new thing that would have to be calculated are the spawn rates of the biomes.

7.3.4 Walking on the biomes

Currently in the application you are just able to fly around in order to view how the biomes generate. Being able to walk around the biome would make the game feel more realistic, it would be as simple as adding a collision matrix to the meshes and a capsule collider to the viewer. The main issue with this improvement is that it would be very costly to the system as constantly generating collision matrices will slow any system down. I would have to find a way to efficiently add this without affecting the current processing of the application.

7.4 Final Thoughts

Overall looking back I believe that the dissertation went very well. There is a large change in how things go compared to how is expected and I will use that as a learning experience for future projects.

Looking at the topics discussed in this dissertation, gaming is becoming more and more popular and techniques such as procedural generation are integral in providing expansive and unique worlds in an industry where the amount of content is so important.

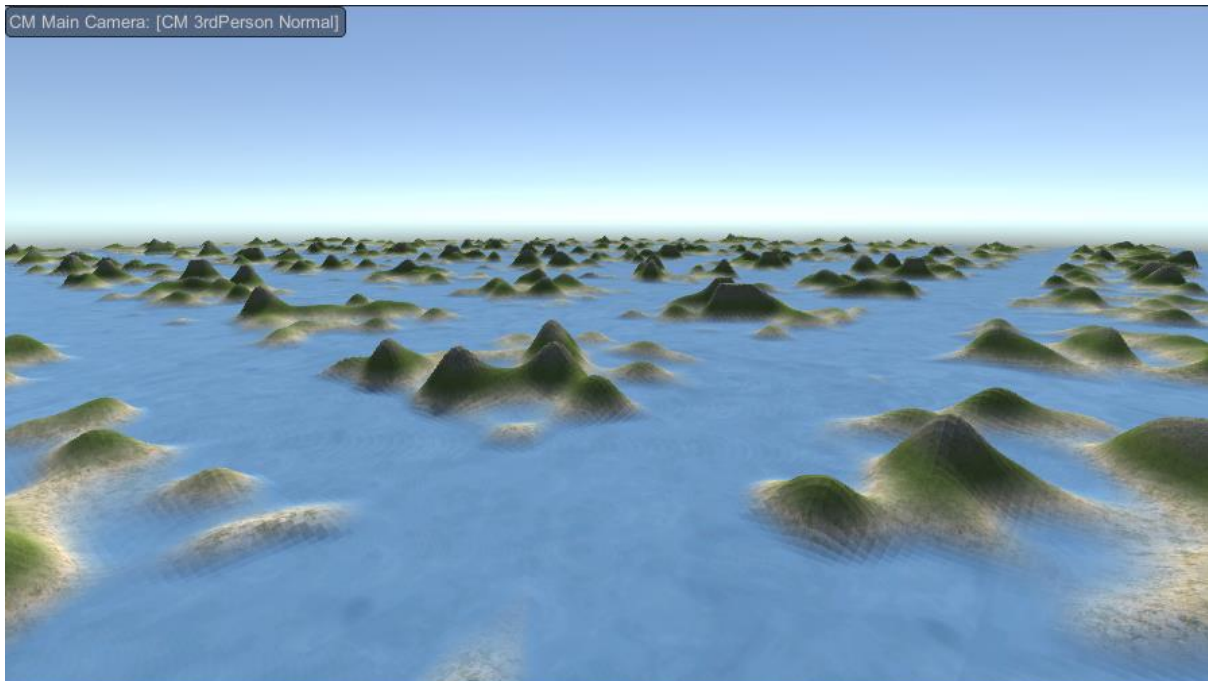
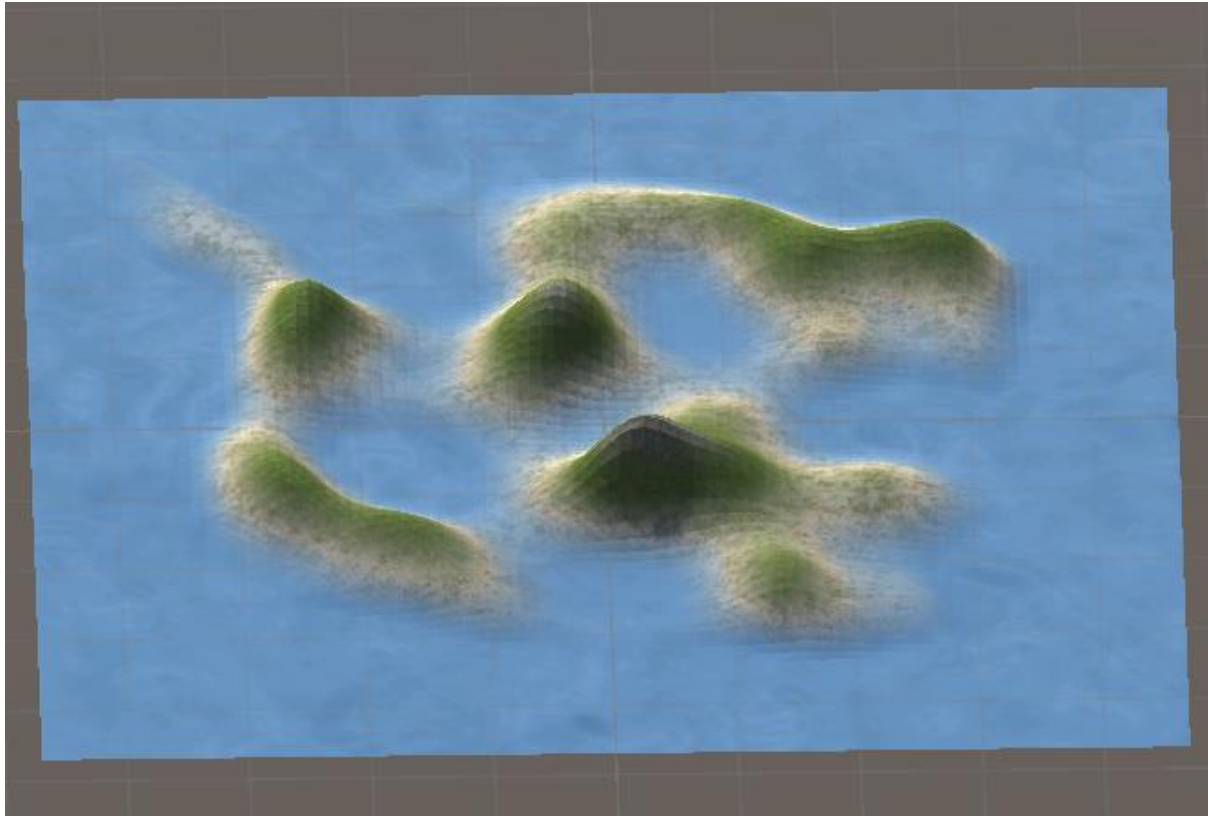
Going back to the original aim as new techniques and methods are discovered I believe that there will be a point where high levels of detail can be achieved with low processing limitations and at that point procedural generation will make leaps and bounds in the industry as the possibilities for “endlessly” generating anything will not be limited by the processing power of a system. Thus, it is important to familiarise oneself with the technologies and methods discussed in this dissertation.

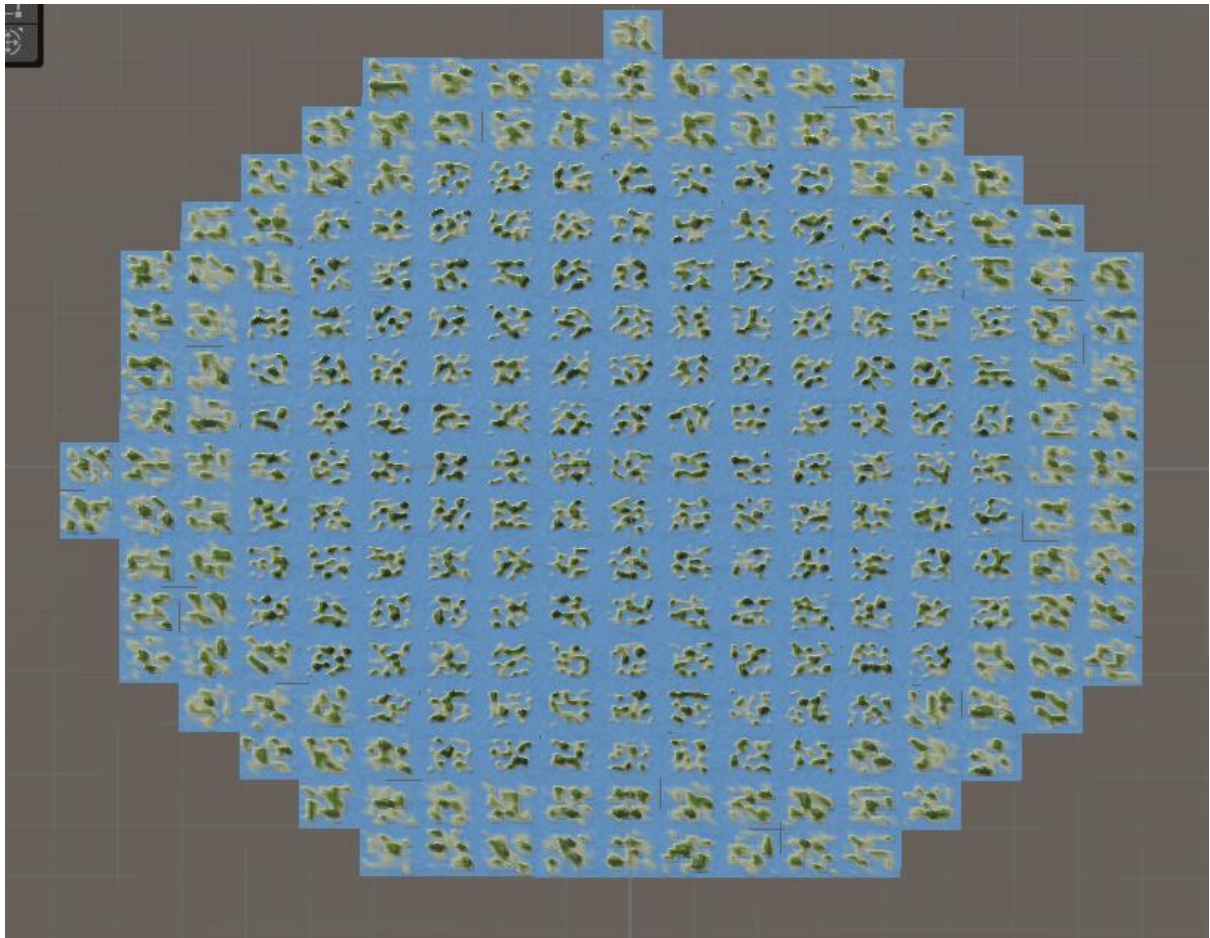
References

- [1] Mark Hendriks, Sebastiaan Meijer, Joeri Van Der Velden, Alexandru Iosup (2013) Procedural Content Generation for Games: A Survey
- [2] Smith, Gillian. "An Analog History of Procedural Content Generation." *International Conference on Foundations of Digital Games* (2015) - <https://www.semanticscholar.org/paper/An-Analog-History-of-Procedural-Content-Generation-Smith/2400a5a51c8ff438f9e080f6c21735853916344e#cited-papers>
- [3] RPGreats Rogue review (2020) <https://www.rpgreats.com/2020/11/rogue.html>
- [4] A Survey of Procedural Content Generation of Natural Objects in Games. Tianhan Gao, Jiahui Zhu (2022) <https://ieeexplore.ieee.org/document/9722677/authors#authors>
- [5] The world of Minecraft Generation article(2022) Alan Zucconi <https://www.alanzucconi.com/2022/06/05/minecraft-world-generation/>
- [6] Perlin Noise: The Evolving Algorithm Behind the Diverse Universes of No Man's Sky (2023) Pratyaksh <https://medium.com/@pratyaksh.notebook/perlin-noise-the-evolving-algorithm-behind-the-diverse-universes-of-no-mans-sky-f2cc8ddacd52>
- [7] A. E. Minarno, A. R. Agisna, W. A. Kusuma, W. Suharso and H. Wibowo, "Optimizing Game Performance with Dynamic Level of Detail Mesh Terrain Based on CPU Usage,"(2020) <https://ieeexplore.ieee.org/document/9081835>
- [8] Procedural Generation: An overview Article (2021) Kenny <https://kentpawson123.medium.com/procedural-generation-an-overview-1b054a0f8d41>
- [9] Procedural Content Generation for C++ Game Development (2016) by Dale Green
- [10] Jeff Kaffitz For noise generation <http://campi3d.com/External/MariExtensionPack/userGuide5R8/Understandingsomebasicnoiseterms.html#Octaves>
- [11] Libnoise For noise generation <https://libnoise.sourceforge.net/glossary/#:~:text=also%3A%20Lacunarity%2C%20Persistence-,Persistence,produces%20%22rougher%22%20Perlin%20noise.>
- [12] Apple Developer <https://developer.apple.com/documentation/gameplaykit/gkcoherentnoisesource/lacunarity>
- [13] Venism https://www.vensim.com/documentation/noise_seed.html

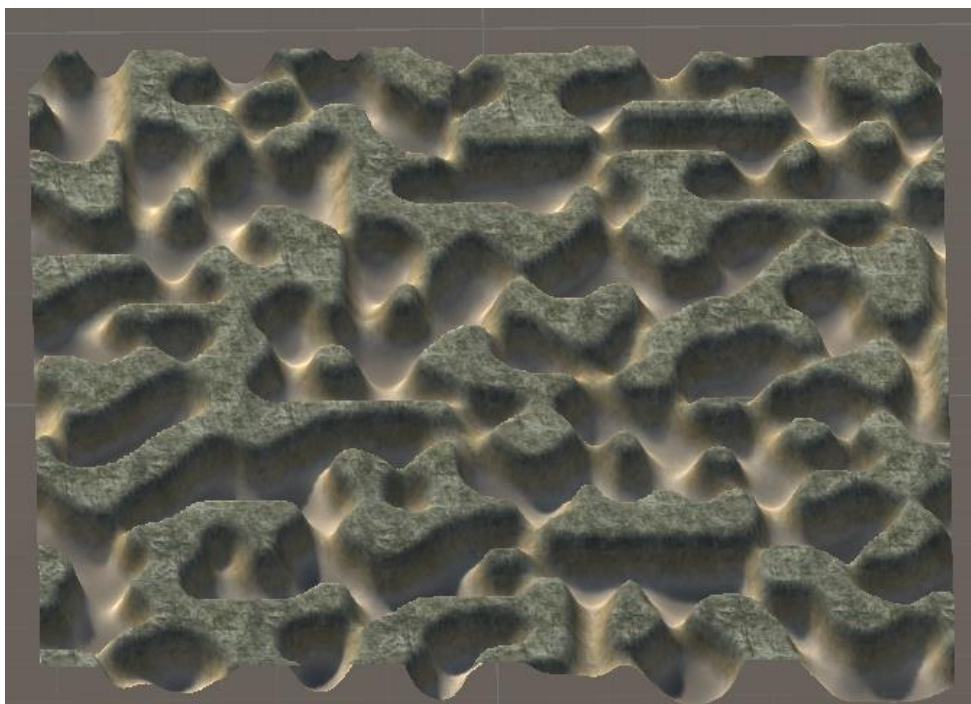
Appendices

Full Island Biome

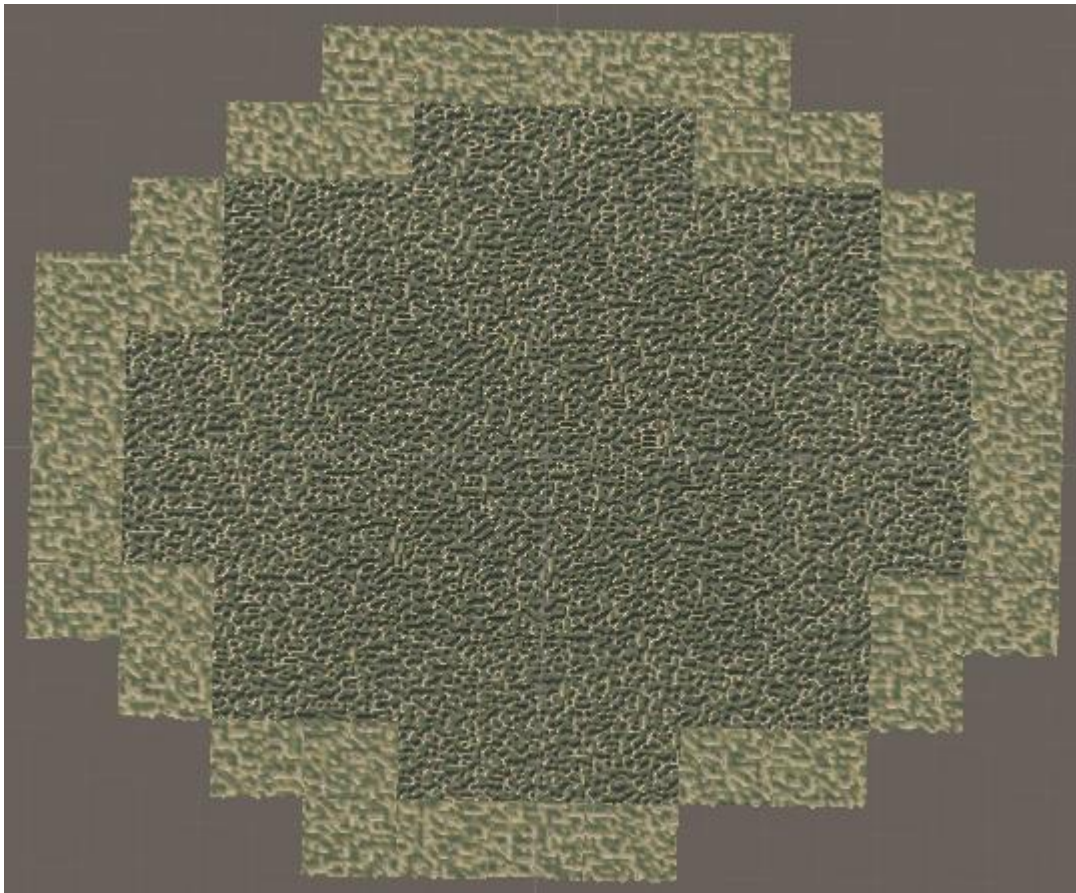




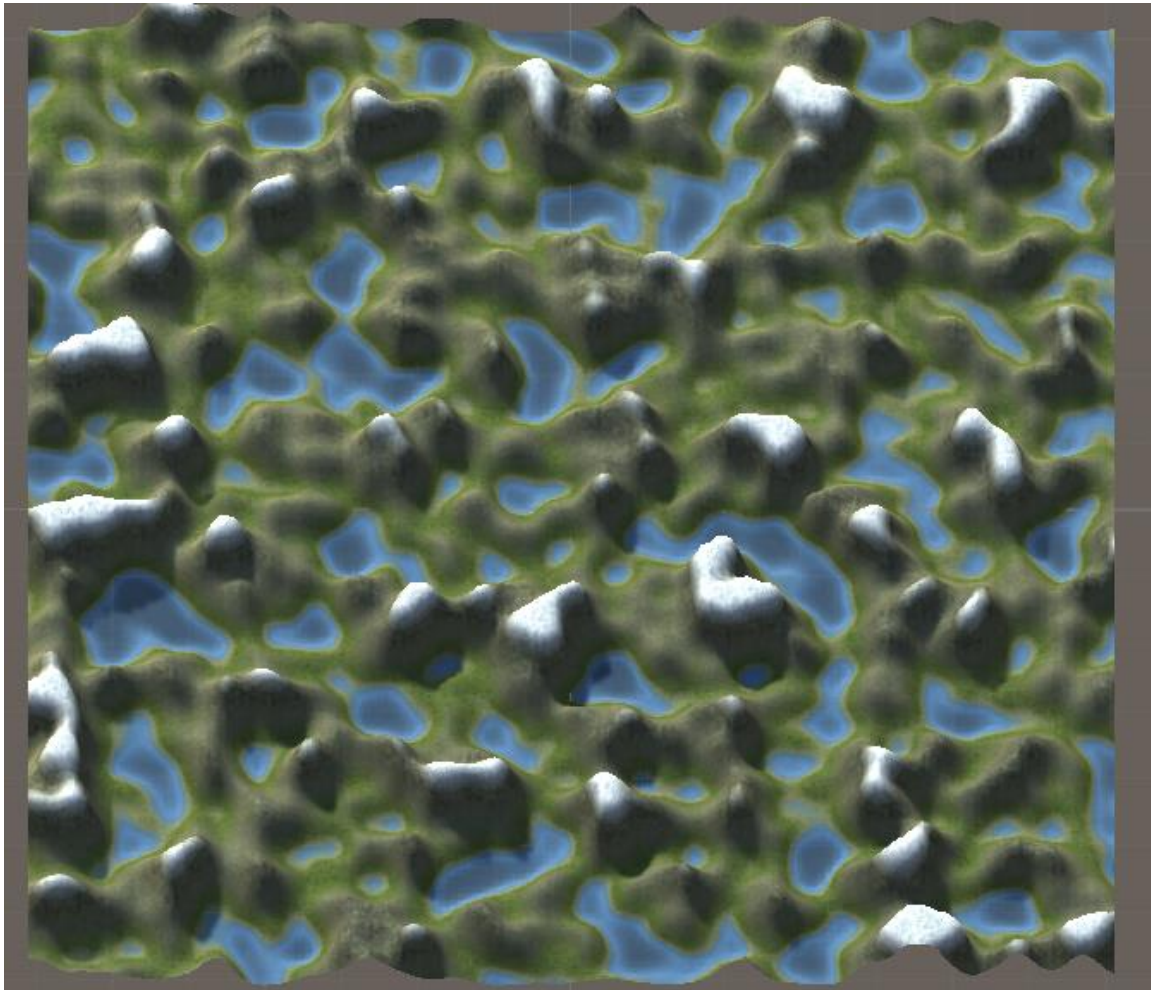
Full Maze Mountain Biome



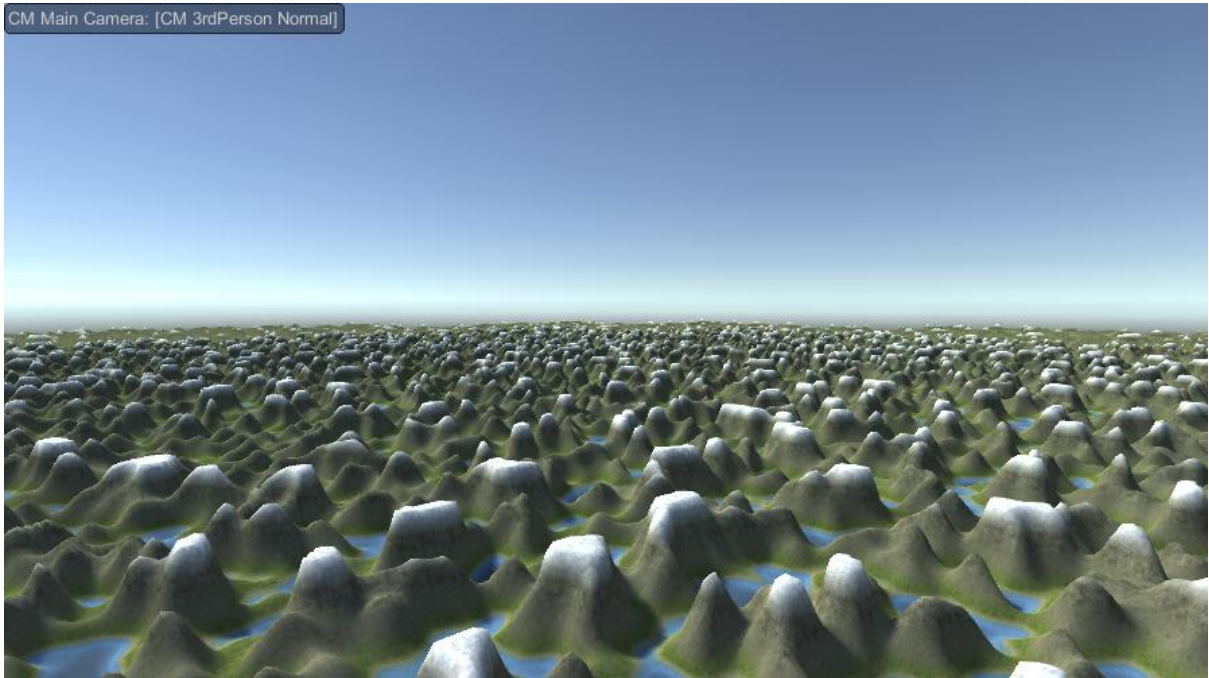
CM Main Camera: [CM 3rdPerson Normal]

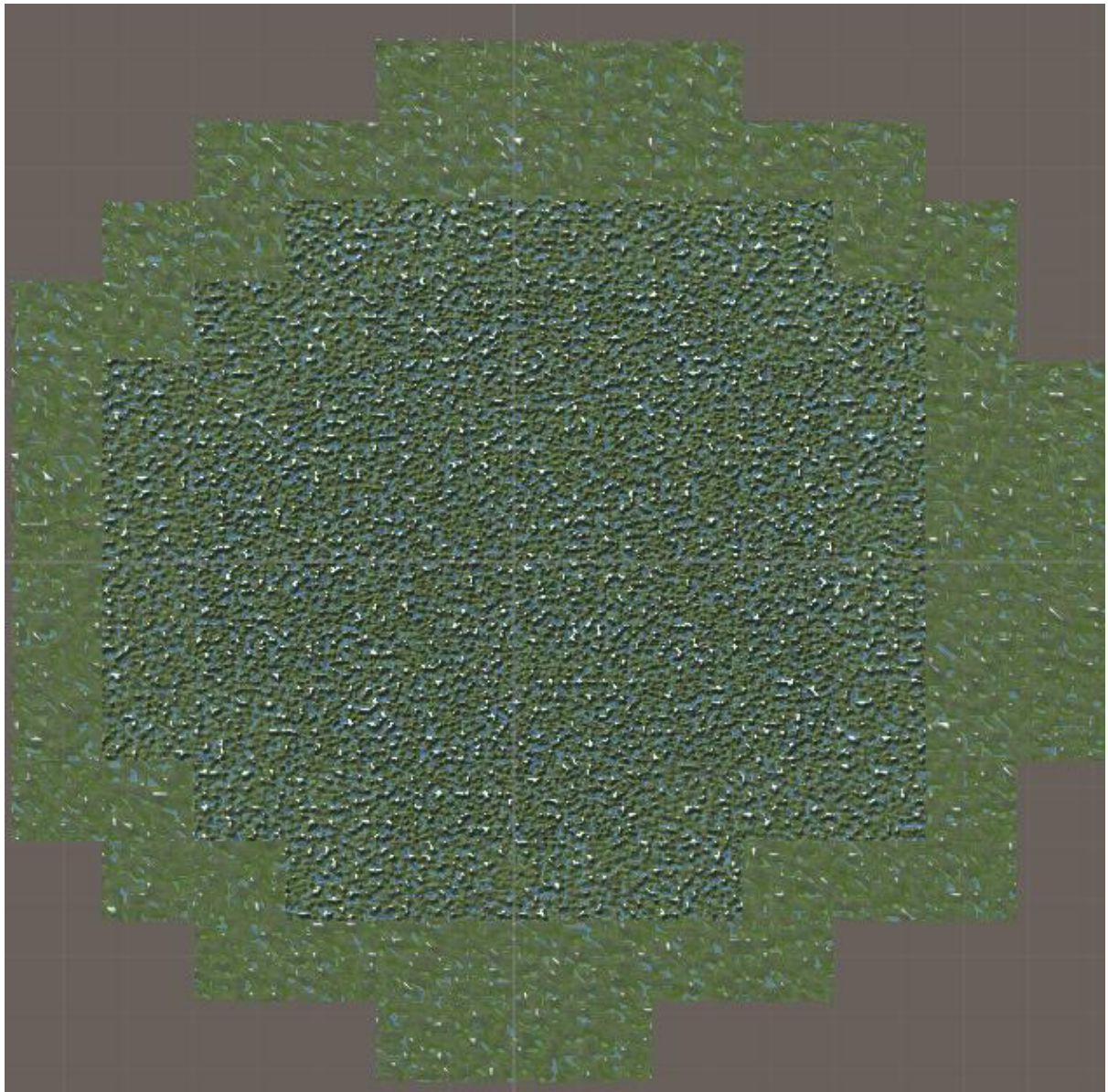


Full Mountain Biome



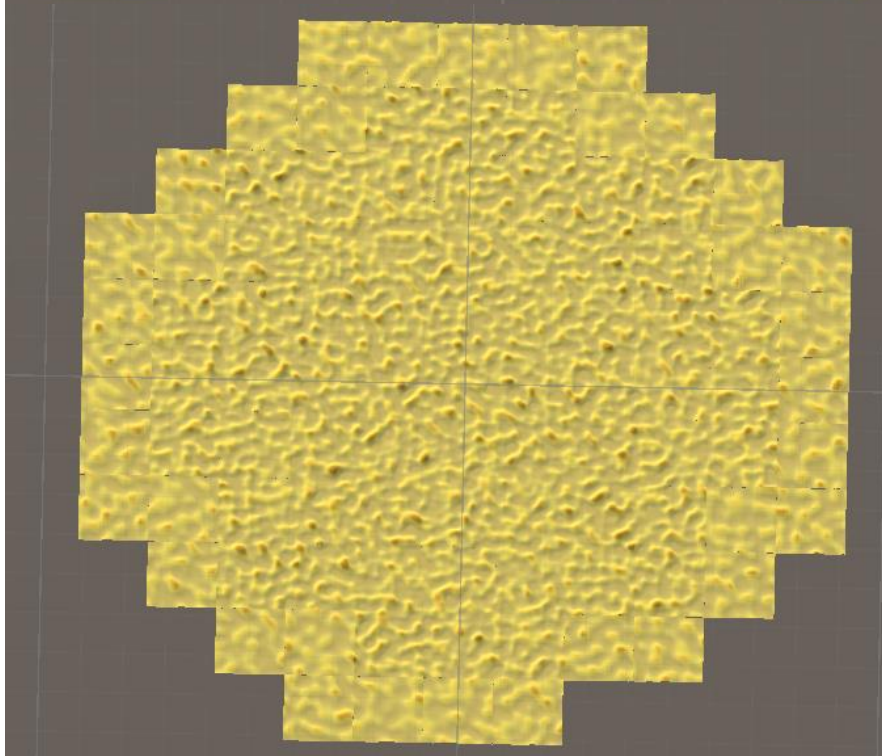
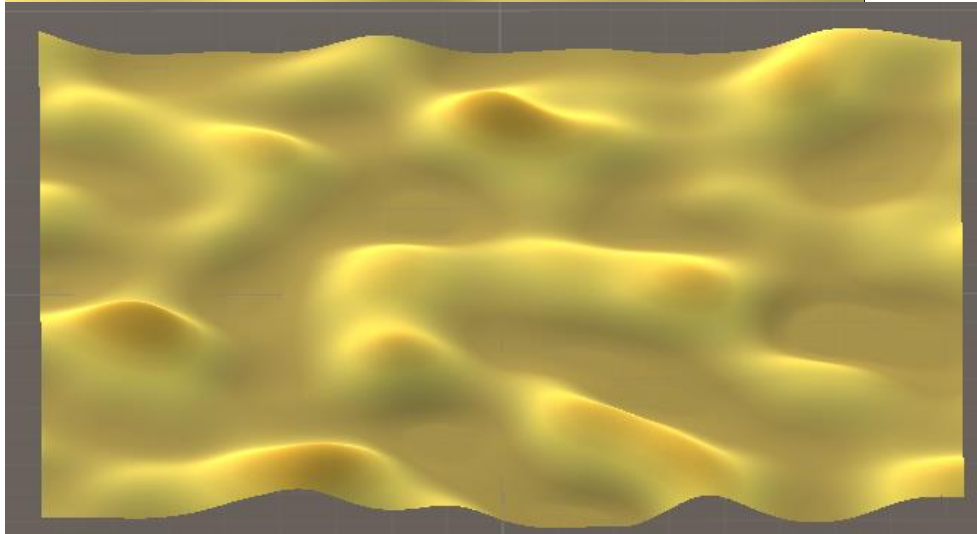
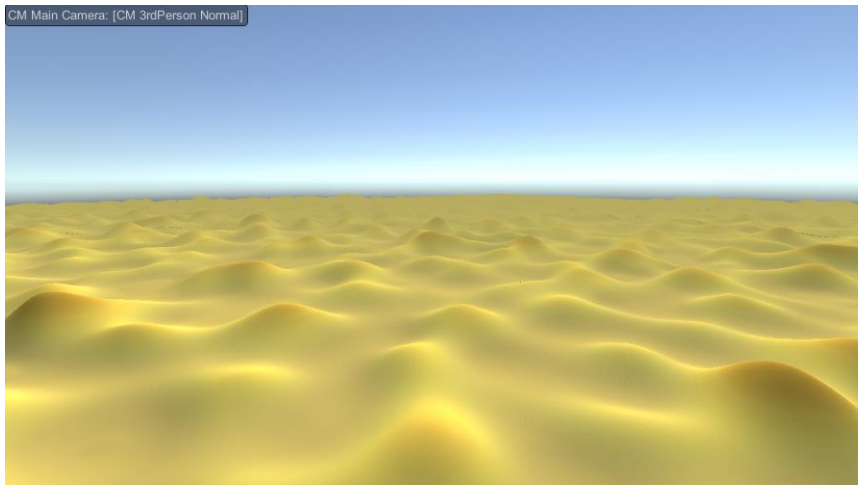
CM Main Camera: [CM 3rdPerson Normal]



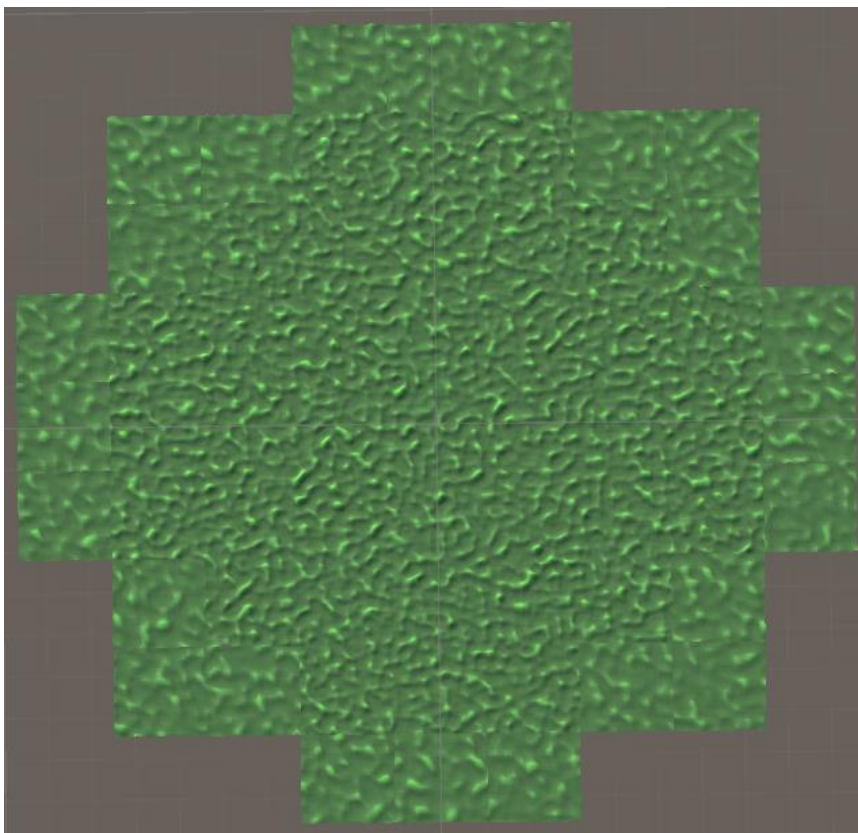
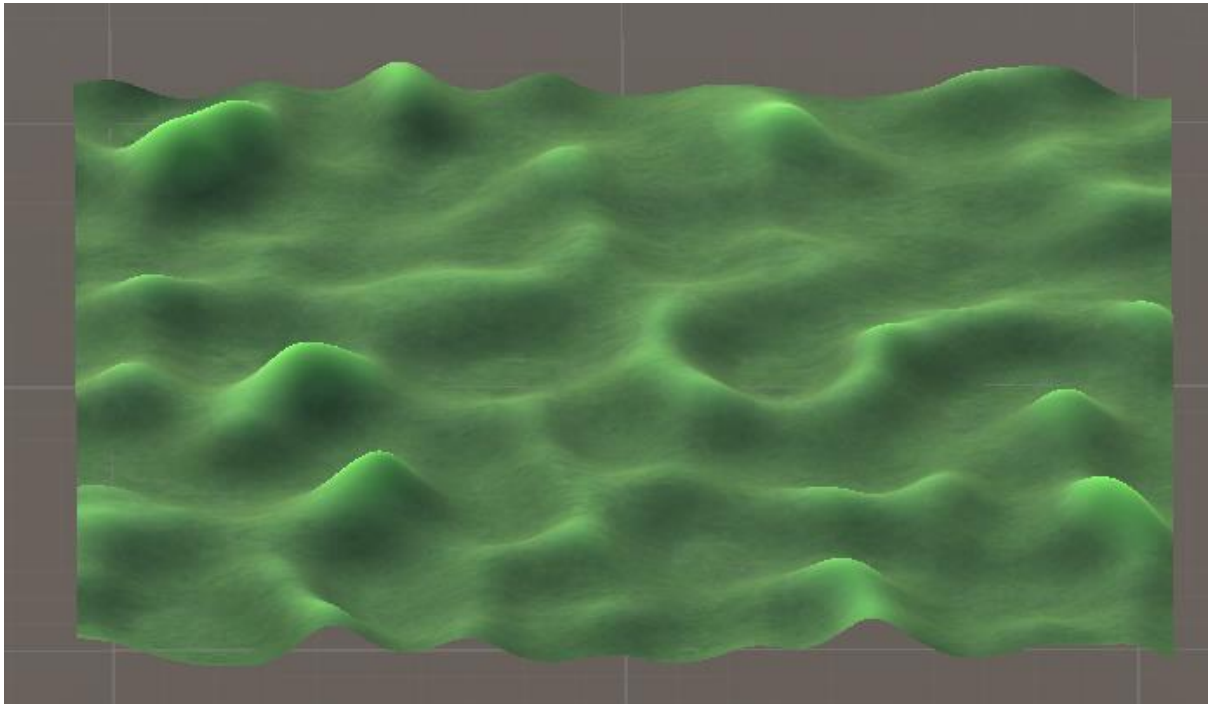


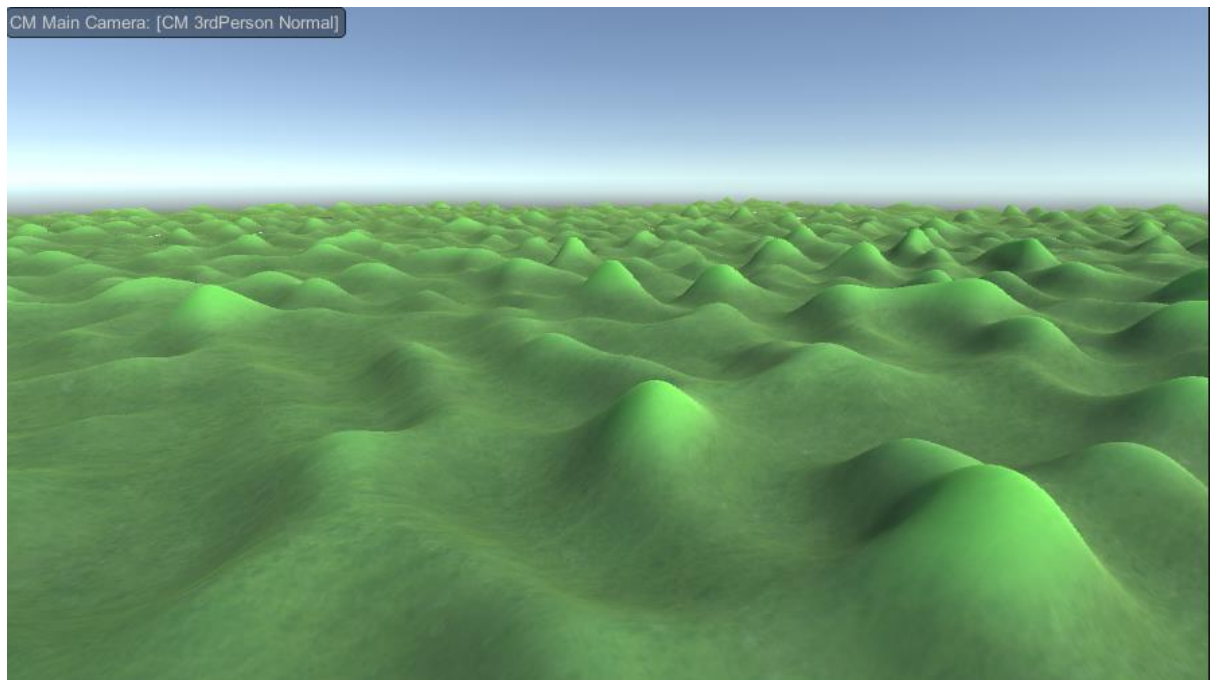
Full Desert Biome

CM Main Camera: [CM 3rdPerson Normal]

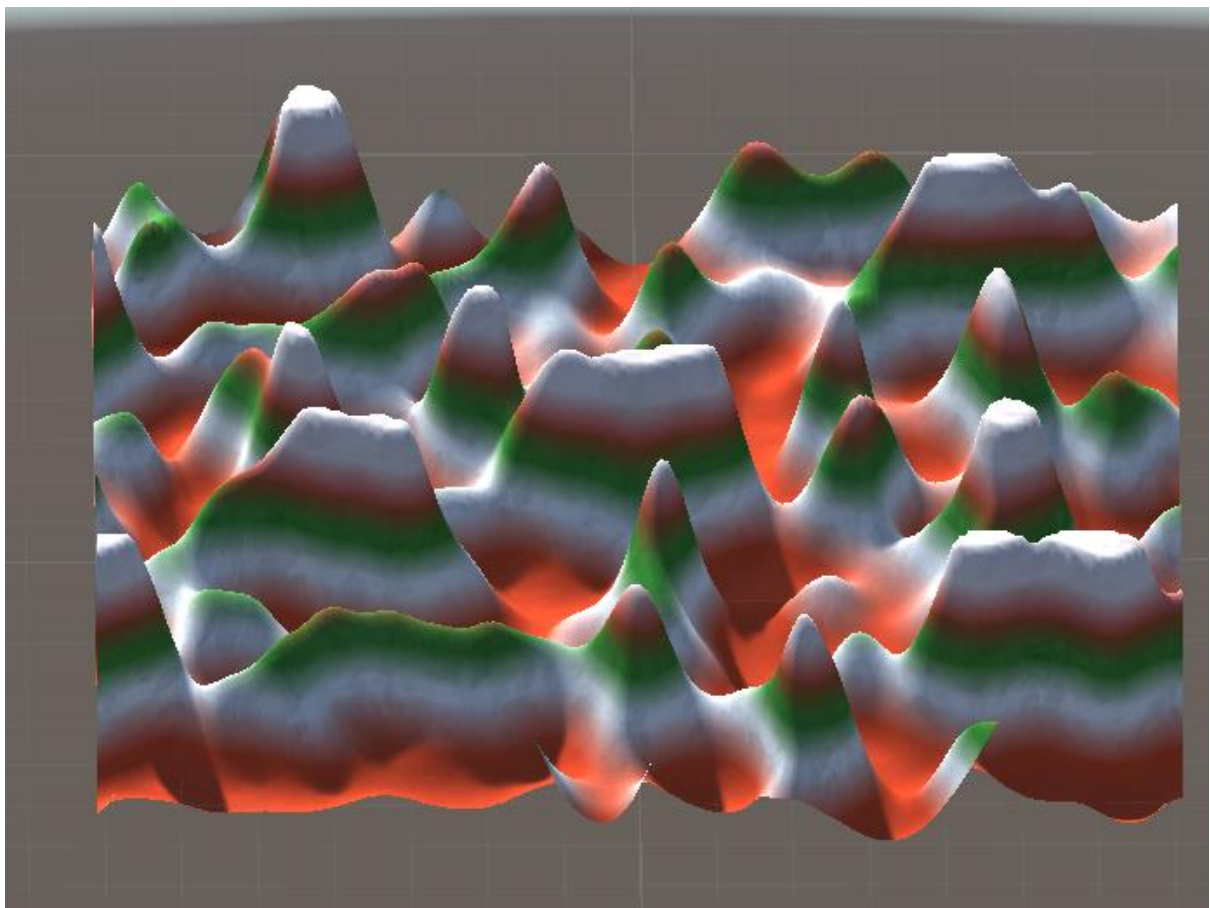


Full Grassland Biome





Full Christmas Biome



CM Main Camera: [CM 3rdPerson Normal]

