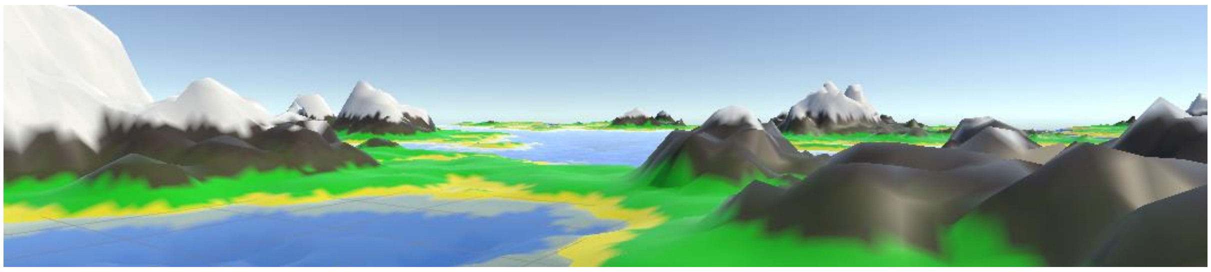
MultiGen TG Brief

Brief

The MultiGen Terrain Generator is a complex modular world generation system that can be used to generate complex terrain, endlessly and randomly. MultiGen TG will be completely customizable within the Unity editor. This allows for easier development of more interesting terrain possibilities. Throughout this brief I will be using the acronym *TG* to refer to *Terrain Generator*.



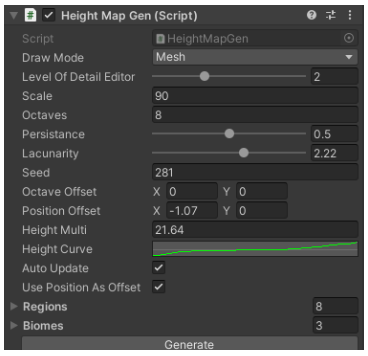
Goals

The MultiGen TG aims to provide a quick and easy tool for developers to create procedural worlds for their games. The tools provided in MultiGen TG should make world creation a two-step process, allowing for quick implementation of other features. MultiGen TG should also allow for fine refinement of the terrain parameters to create many different types and styles of terrain. The user should be able to easily add and configure biomes. This should be able to be done almost entirely through the editor.

Third-Party Libraries

The MultiGen TG will use the Unity built-in libraries and the C# Random library. The Unity Math library will be used to generate the Perlin noise maps. The Unity Engine library will be used to interface with the Unity editor.

The C# Random library will be used to generate random numbers within a range. This will be used to modify the noise maps that are used to generate the height map.



Height Map Gen (Script)

Script	HeightMapGen
Draw Mode	Mesh
Level Of Detail Editor	2
Scale	90
Octaves	8
Persistance	0.5
Lacunarity	2.22
Seed	281
Octave Offset	X 0 Y 0
Position Offset	X -1.07 Y 0
Height Multi	21.64
Height Curve	
Auto Update	✓
Use Position As Offset	✓
Regions	8
Biomes	3

Figure 1: Early UI for MultiGen

Integration

The MultiGen TG should be easily integrated into a new project. The process should be as simple as adding a new GameObject and assigning the MultiGen Terrain Generator script component to this GameObject.

Maths and Algorithms

- **Noise**

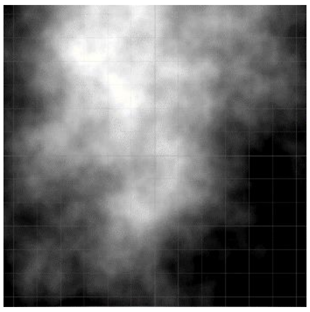


Figure 2: An example of the height map generated by the Perlin Noise. White areas represent higher elevation.

I will be using the Perlin Noise function provided by Unity's Math library. This will be used to generate the noise maps that form the height map. The noise shape will primarily be controlled by 3 variables: Octaves, Persistence, and Lacunarity.

- **Octaves:** The octaves is the number of layers of Perlin Noise that will be applied to create the height map. Each layer (octave) will contribute to the height map less and will have smaller and finer details than the last octave.
- **Persistence:** The persistence is the amount by which each octave decreases its contribution. Persistence is applied as exponential decay, meaning that with a persistence of 0.5 each octave will contribute half as much to the final height as the last octave.
- **Lacunarity:** The lacunarity is how much of an increase in detail each octave will have. Lacunarity is applied as exponential growth, meaning that with a lacunarity of 2 each octave will have twice as much detail as the previous octave.

- **LODs & Interpolation:**

The height map that is generated by the Perlin Noise does a good job of creating terrain that looks realistic from a distance. However, if we were to add a player straight onto this terrain the effect will likely be lost due to things like polygon count limitations and unrealistic scales. To solve this problem, I will be implementing a system of LODs that allow for higher amounts of resolution close to the player but lower resolution as the chunk gets further from the player.

- **LODs:** The main idea behind LODs is skipping every **i** vertex, **i** being the number of vertices to skip. The problem with simply skipping a number of vertices is that depending on the number of vertices and the value of **i**, it is possible that an out of bounds exception will occur. The solution to this problem is to use LOD levels that have values for **i** that are a factor of the edge length of each chunk, defaulted at 255 which is the maximum length that supports Unity's cap of 65,535 vertices per mesh.
- **Interpolation:** Interpolation can be very useful for smoothing out terrain as the player get closer. The main use of interpolation in this project will be to create higher resolution textures for the terrain.

Modularity

The MultiGen TG will feature a modular biome system. This system will have two systems for biome generation that a developer could use. The first system works based on an area's humidity, heat, and elevation to allow for a more realistic environment. The second system uses a Perlin Noise map to select biomes randomly. The user should also be able to quickly add new biomes by adding a biome asset that they can then modify to specify what temperature and humidity and height it spawns at or what value between 0 and 1 that it spawns at if using the single map Perlin value.

BIBLIOGRAPHY

Oliveira, R. (2023) *Complete guide to procedural level generation in unity – part 1*, *GameDev Academy*. Available at: https://gamedevacademy.org/complete-guide-to-procedural-level-generation-in-unity-part-1/ (Accessed: 10 May 2023).

Minecraft terrain generation in a Nutshell (2022) *YouTube*. Available at: https://www.youtube.com/watch?v=CSa5O6knuwI (Accessed: 10 May 2023).

Lague, S. (2016) *Procedural landmass generation (E06: LOD)*, *YouTube*. Available at: https://www.youtube.com/watch?v=417kJGPKwDg&list=PLFt_AvWsXl0eBW2EiBtl_sxmDtSgZBxB3&index=6 (Accessed: 10 May 2023).

Lague, S. (2016b) *Procedural landmass generation (E07: Endless terrain)*, *YouTube*. Available at: https://www.youtube.com/watch?v=xlSkYjiE-Ck&list=PLFt_AvWsXl0eBW2EiBtl_sxmDtSgZBxB3&index=8 (Accessed: 10 May 2023).