My complex system is a procedural terrain generation tool for quickly generating infinite terrain. It uses perlin noise to generate extensive, complex terrain. My original goal was to allow a developer to modularly implement different biomes that are selected based on a number of factors such as heat, humidity and elevation. These different biomes would also allow for different types of vegetation to grow depending on their desired environments. In the final product I managed to implement the vegetation system and have it rely on the maps elevation to spawn different types of plants. The biomes ended up being too much of an overscope and I did not manage to implement my desired system for the modular biomes.

The main reason for my failure in implementing the fully desired system was due to the complexity of the modules and how I wanted them to function. I had envisioned a system of scriptable objects that allow you to add a biome, and set the desired; height, humidity and heat regions that you wanted the biome to be located at. After looking into realistic systems for generating heat and humidity regions I found implementing these systems to be more difficult than I had originally planned.

In order to reach the goal of a modular terrain system I created a system for modular vegetation generation and a more in depth height map system that allowed you to generate larger continent regions. The height map system allowed for smoothly interpolating between different height values to create smoother slopes into mountain ranges and better distinction between ocean areas and the continents. The vegetation system allowed the user to create a list of plant prefabs and assign specific height ranges that they can spawn in. It also allowed for the user to assign distribution within this height range using an animation curve. The user could add as many different types of vegetation as they needed and they could assign the distribution of the plants.

A graph of a number of data

Description automatically generated with medium confidenceMy system required many optimizations to run smoothly and efficiently within a larger game. The largest and most important optimisation that I made was the inclusion of multi-threaded (MT) generation. This required me to develop my own simplified version of the random number generator that was multi-threading friendly. Multi-threading brought my FPS on the computer I was using on campus from bellow 20fps, to above 80. On my personal computer I was able to achieve frame rates of up to 150.

Another major optimisation was to split the terrain into terrain chunks, which allowed for higher resolution meshes. This also meant that I was able to limit the amount of the terrain that is being rendered and updated. I also had a similar system for the vegetation. The vegetation would spawn in when the player is within a certain range and then the plant’s game object will destroy it when it is out of this range. The vegetation has its own distance value.

The third and final optimisation that I implemented was LOD meshing for the terrain. LOD stands for Level Of Detail and is often used in games to improve the performance without major decrease in image quality. LOD works by swapping out meshes and textures with lower resolution version as objects get further from the camera. This results in less time being wasted calculating and rendering details and depth that would be lost visually due to the distance of object from the camera. My LOD system decreased the resolution of the meshes as they got further away.