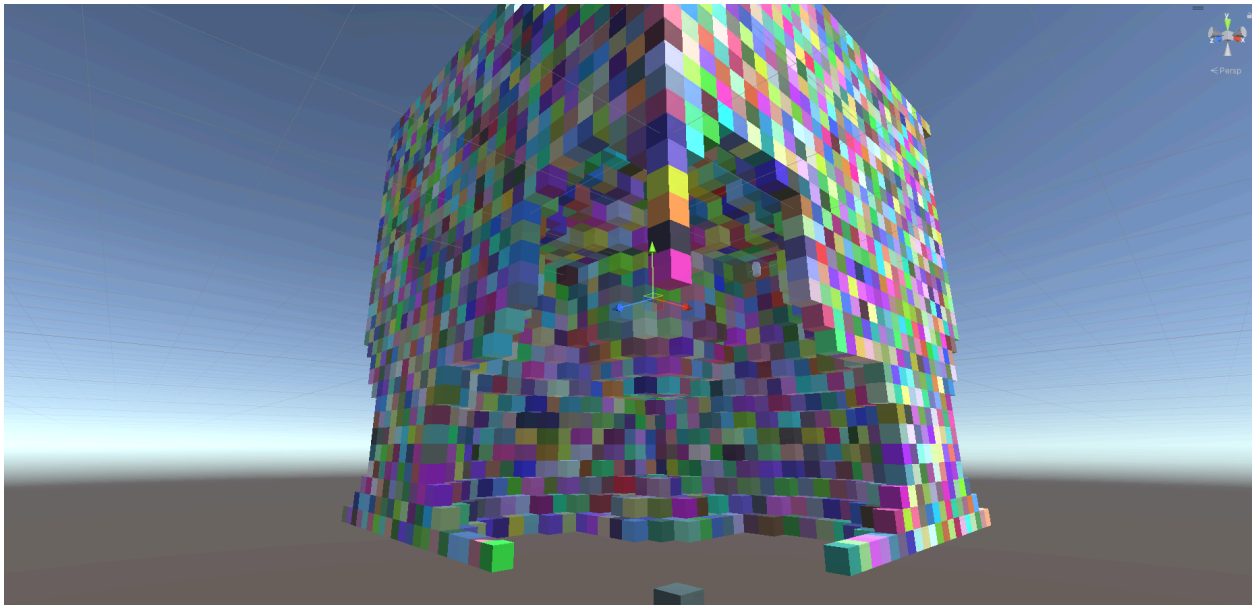
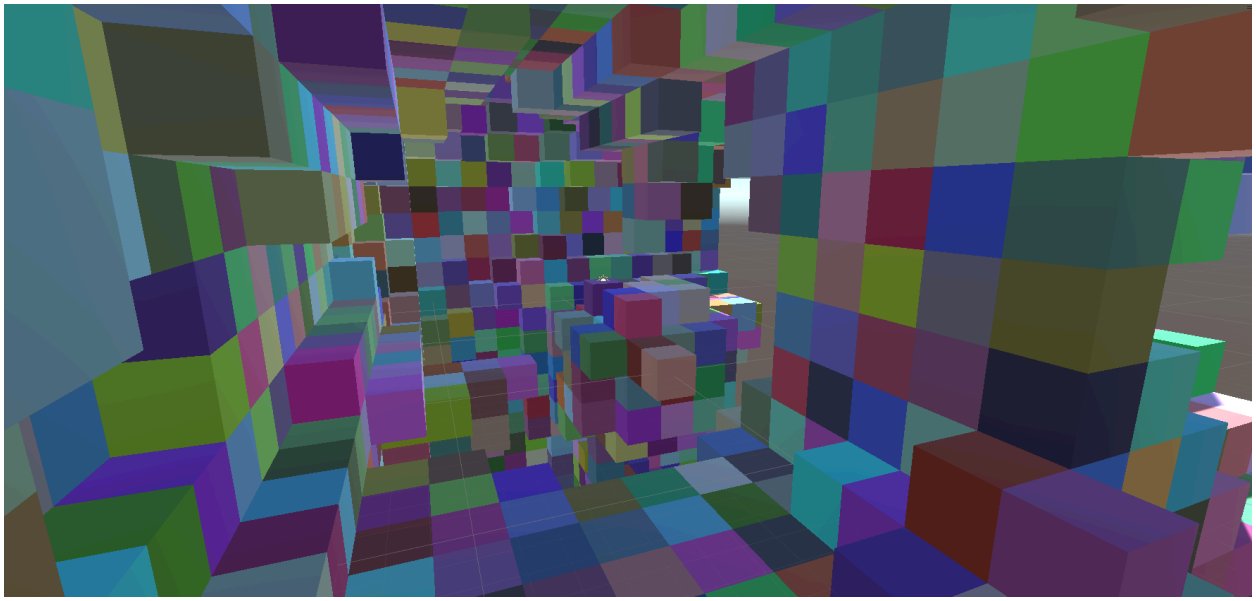
This post is for game developers and people who would like to create the world generation.

LINK TO GITHUB: https://github.com/LiamHarrison25/GPR-340-Final-Cave

Brief overall project

For my Project, I worked with Liam Harrison to create a procedural world generation using 3D Perlin noise. We also created chunking with the help of spatial quantisation.





Platform suitability and Inspiration:

Our project would be suitable for any platform, as we have used techniques that could be easily transferred over to things like marching cubes.

Our biggest inspiration for this project was Minecraft.

Techniques:

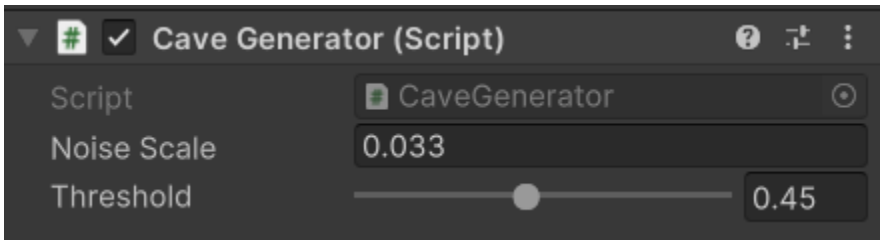
Spacial Quantisation:
Spatial quantisation was a core element of our project. We used this in several ways. Firstly, spacial quantisation helped us to quantise the grid, this meant we had a list that contained several chunks, and these chunks held individual cells. By doing this, we can store information for example, if the cell should be enabled or disabled. We used dictionaries that had a key that had been quantised. If we pick any position in space, we convert it into an int, this means any of the values between the interval where this chunk starts and ends are quantised into a vector3 Int. We did this again and created another key for the dictionary inside the chunk which stored the cells.

To achieve this, we used Mr Tolstenko's quantisation techniques. We learned from this blog to understand how to use Quantisation and how to apply it to our project.
https://courses.tolstenko.net/courses/artificialintelligence/readings/spatial-quantization/

Perlin noise world generation:
We wanted to capture the essence of randomness but still purposeful. We loved the idea of caves sprawling organically, similar to Minecraft. The main idea of **procedurally generating** is a world generating from a noise type, we chose Perlin noise because there was already a built-in functionality within Unity. The noise was difficult to work with as we had to tweak it to be perfect with the size of our cells in the world. In unity, we added a noise scale to change how much of the Perlin noise we wanted per cell. The further zoomed in on the Perlin noise, the less noise is presented, and the bigger the caves are. The way we used Perlin noise was to get 3 position variables (x,y,z), multiply it by a noise scale to use Perlin noise by testing each value with another and then spit out a value between 0 and 1. This then was tested against a threshold to determine if the cell should exist or not.

We used this source to get a better grasp on 3D Perlin noise:
https://www.youtube.com/watch?v=TZFv493D7jo

Unity hierarchy to change how Perlin noise is used:

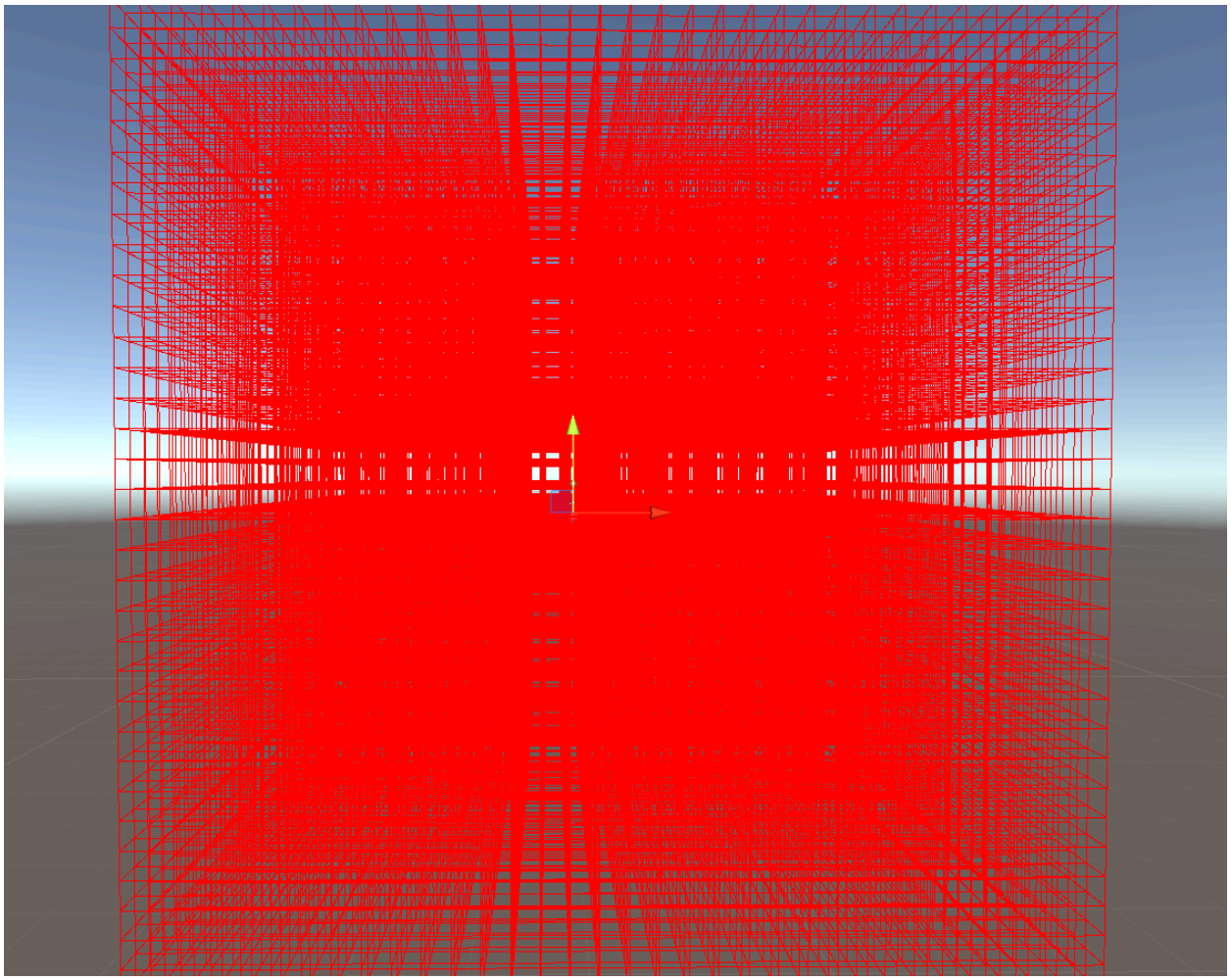


Chunking and cells:

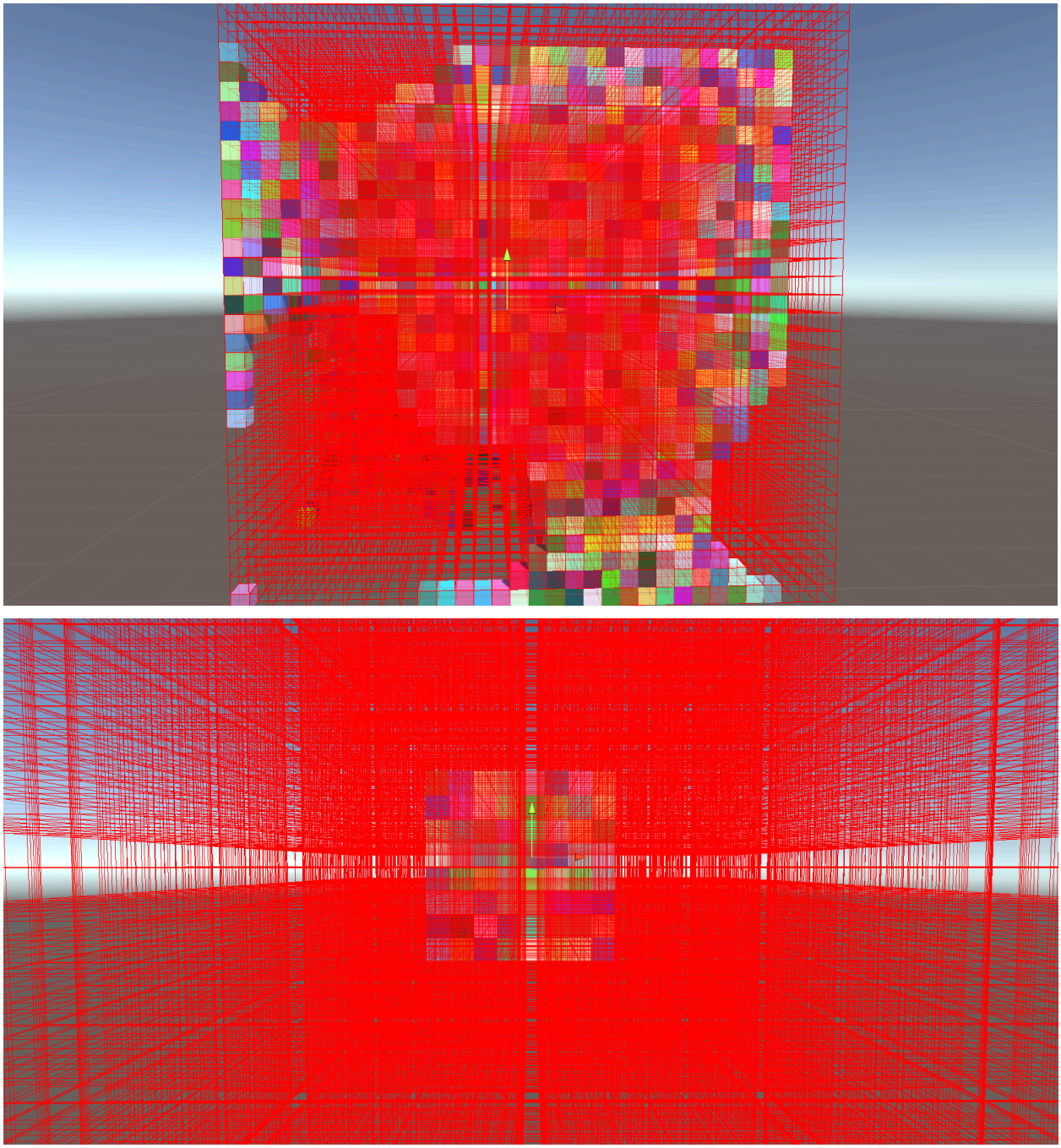
We ran into the problem of unity being very laggy when spawning a lot of objects. This was a huge issue and was hard stopped by the technology we are using. So, to solve this problem, we decided to use the **technique of chunking**. Chunking means splitting up the world into sections and splitting those chunks into even more sections called cells. This meant that we could control the amount of “chunks” viewable to the player. This technique helped us to control the amount of lag so the scene was playable. All we have to do is set the render distance before we press play and then we can have almost an infinite amount of chunks that can be loaded and unloaded. We then paired this up with tracking the player's position, so the render distance uses the player’s position and only updates if the player moves chunks. This helps with unnecessary chunk reloading.

These videos helped give insights into the above information by letting us understand chunking better:
https://www.youtube.com/watch?v=CSa5O6knuwI&t=327s

https://www.youtube.com/watch?v=YyVAaJqYAfE&t=1469s
Grid:



Grid Gizmo vs the render distance



Issues and future improvements:

One issue with our project is that it didn't calculate the cell positions correctly within the Perlin noise, it's most likely due to the keying in the values; it's something with the local position instead of the world position.

The other issue is we wanted to reduce the amount of lag in the scene. We wanted to include something called Combined instances in unity, this means instead of individually spawning

every cell, we combined the meshes so the vertex count was reduced significantly. Unfortunately, We didn't have enough time to finish the project and it ended up just sitting in the test branch.

A personal improvement with our code is to re-do. With the knowledge I have now, and looking at how many interactions we had to do to even make the grid script and chunking work out, I would love to re-create this project later. Making it cleaner, more efficient and more optimised. We are so close to making this work. We just needed more time to complete it.

Additional Information:
I want to point out we used Microsoft live-share, which allowed us to work on the same file and code simultaneously. This allowed us to bounce ideas off each other It was the whole reason this project was fun.

We also used this to have a target to aim for:
https://github.com/Auburn/FastNoise2Bindings