Optimisation

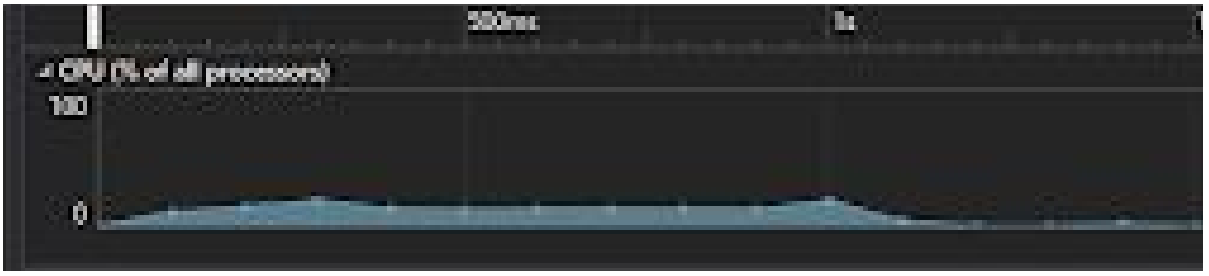
From the first batch of optimisation data the worst issue was in under my control was the generate terrain function which is only called once to start the program and feed in the mesh. I believe this is because I am making more calls to the get layered noise both in the function and in the get Colour sub function to get the position/colour of each vertex than necessary.

```
glm::vec3 Terrain::getColour(double nx, double ny)
{
    glm::vec3 col(getLayeredNoise(nx, ny) + 0.5, (getLayeredNoise(nx, ny) + 0.5), (getLayeredNoise(nx, ny) + 0.5));
    col = col + glm::vec3(0, 1 - (getLayeredNoise(nx, ny) + 0.75), 0);
    if (getLayeredNoise(nx, ny) <= -0.1)
    {
        col = col + glm::vec3(0,0, 1);
    }
    return col;
}
```

So I decided to first optimise get colour because it was the easiest one to do by only calling the get Layered noise function once.

```
glm::vec3 Terrain::getColour(double nx, double ny)
{
    double layeredNoise = getLayeredNoise(nx, ny);
    glm::vec3 col(layeredNoise + 0.5, (layeredNoise + 0.5), (layeredNoise + 0.5));
    col = col + glm::vec3(0, 1 - (layeredNoise + 0.75), 0);
    if (layeredNoise <= -0.1)
    {
        col = col + glm::vec3(0,0, 1);
    }
    return col;
}
```

This lead to a load time of 1.2 seconds on my system to shorten down to 0.9.



I then realised that we were already generating the layered noise for each point before we even entered the get colour function, so why not just pass that into the get colour function instead of the noise location.

```
void Terrain::generateTerrain(int maxX, int maxY, float noiseAmplification, float heightAmplification)
{
    double previousNx = 0, previousNy = 0;
    for (double x = 1; x < maxX; x++)
    {
        double nx = x / maxX * noiseAmplification;
        for (double y = 1; y < maxY; y++)
        {
            double ny = y / maxY * noiseAmplification;

            //vertex positions
            glm::vec3 p1(x - 1, getLayeredNoise(previousNx, previousNy) * heightAmplification, y - 1);
            glm::vec3 p2(x, getLayeredNoise(nx, previousNy) * heightAmplification, y - 1);
            glm::vec3 p3(x, getLayeredNoise(nx, ny) * heightAmplification, y);
            glm::vec3 p4(x - 1, getLayeredNoise(previousNx, ny) * heightAmplification, y);

            Vertex v1(p1, getColour(previousNx, previousNy));
            Vertex v2(p2, getColour(nx, previousNy));
            Vertex v3(p3, getColour(nx, ny));
            Vertex v4(p4, getColour(previousNx, ny));

            mesh->addSquare(v4, v3, v2, v1);

            previousNy = ny;

        }

        previousNx = nx;
    }
    xMax = maxX;
    yMax = maxY;
    noiseAmp = noiseAmplification;
    heightAmp = heightAmplification;
}
```

```
void Terrain::generateTerrain(int maxX, int maxY, float noiseAmplification, float heightAmplification)
{
    double previousNx = 0, previousNy = 0;
    for (double x = 1; x < maxX; x++)
    {
        double nx = x / maxX * noiseAmplification;
        for (double y = 1; y < maxY; y++)
        {
            double ny = y / maxY * noiseAmplification;

            //vertex layered noise
            double ln1 = getLayeredNoise(previousNx, previousNy);
            double ln2 = getLayeredNoise(nx, previousNy);
            double ln3 = getLayeredNoise(nx, ny);
            double ln4 = getLayeredNoise(previousNx, ny);

            //vertex positions
            glm::vec3 p1(x - 1, ln1 * heightAmplification, y - 1);
            glm::vec3 p2(x, ln2 * heightAmplification, y - 1);
            glm::vec3 p3(x, ln3 * heightAmplification, y);
            glm::vec3 p4(x - 1, ln4 * heightAmplification, y);

            Vertex v1(p1, getColour(ln1));
            Vertex v2(p2, getColour(ln2));
            Vertex v3(p3, getColour(ln3));
            Vertex v4(p4, getColour(ln4));

            mesh->addSquare(v4, v3, v2, v1);

            previousNy = ny;

        }

        previousNx = nx;
    }
    xMax = maxX;
    yMax = maxY;
    noiseAmp = noiseAmplification;
    heightAmp = heightAmplification;
}
```

This further pushed the load time down by a tenth of a second.

I then defined the function passing the address of the object rather than duplication the double and solidified the previous result with the bumps representing setting up the opengl buffers fitting before the one second mark.

By reducing the calls to the get noise function I managed to cut a third of of the initial load time of the game and fit the opengl initialization under a second from one and a half seconds.