Generation of Islands and Simulating Climate Change

Using OpenGL

Steven Kirby

BSC Computer Science (Game Engineering)

Submitted: 03/05/2019

Wordcount: 14779

Pages: 100

Supervisor: Dr. Richard Davison

Abstract

This dissertation graphically showcases the effect that climate change could have on the United Kingdom (UK) due to rising sea levels; explores techniques used to render graphics in real time using OpenGL and measures the performance of differing tessellation values, model sizes and primitive sizes.

Declaration

I declare that all writing and graphics in this dissertation are of my own work except where stated or otherwise implied.

Thanks to

Thank you to Dr. Richard Davison for the support and guidance that was provided throughout the production of this simulation and dissertation.

I would also like to thank Richard for allowing me to use the codebase he has authored to teach OpenGL in the graphics and games modules at Newcastle University as a base for my dissertation

Contents

Chapter 1: Introduction

1.1 Motivation and Rationale

The primary motivation for this dissertation is to bring awareness to the potential ramifications of climate change accelerated by human activity in the last century that has offset the balance of the natural carbon cycle, effectively overloading the ecosystem and causing sea levels to rise.

This impact could be demonstrated using a real-time graphics simulation. This is a personal motivation of mine; to learn graphics programming, explore graphics programming techniques and impact on performance, using OpenGL and OpenGL Shader Language (GLSL).

Climate change causes damage to not only areas of human habitation, but also to the habitat of vulnerable species such as the polar bear. Polar bears live on the sea ice, which is melting due to the worlds rising temperatures, causing them to travel further or even lose access to their main food source, seals, and in some cases, starve because of this. (National Geographic, Leahy, 2018)[1]

This dissertation however, is based around the effect of climate change on an island (the UK to be specific) generated using OpenGL, and showcases these issues through a graphical simulation of a rising sea level caused by melting glaciers.

Existing simulations

NASA has produced an interactive 2D map showing the effect of sea levels rising up to 6-metres, obtained from using still images by the Centre for Remote Sensing of Ice Sheets (CReSIS) which shows what area of land would be covered by water.

This is shown in Figure 1 (right).

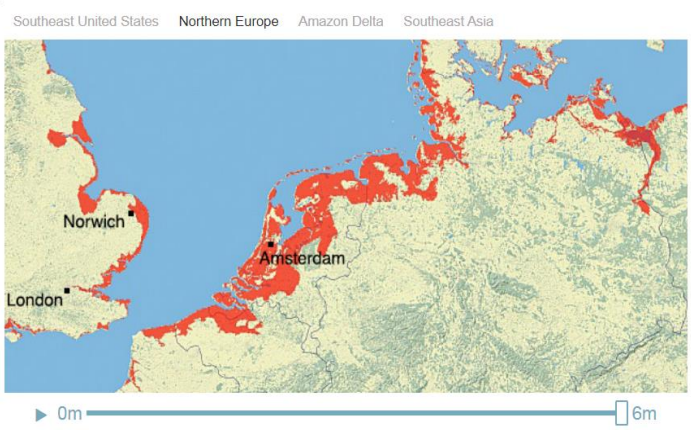


Figure 1: Inundation Of Water (Increase of 6 Metres)

The original still pictures that NASA used *Appendix 1 – Sea Level Rising* are shown in the appendices at the end of this dissertation.

NASA has also produced a 3D simulation of sea level rise, this is discussed in more detail later on in Chapter 3: Research.

1.2 Aim and Objectives

The aim and objectives helped guide me through this dissertation and ensured that I was on track for completion within the timescale and gave me goals to work towards.

1.2.1 Aim

The aim of the dissertation is as follows:

"To simulate the effects of climate change on a 3D generated model of an island using OpenGL"

This changed from an original aim of generating an archipelago, (collection of islands) using Perlin noise early on in the dissertation.

1.2.2 Objectives

The objectives of the dissertation are as follows:

1. Estimate and present (via charts) predicted CO_2 and Sea Levels in the future.
2. Research how Perlin noise and heightmaps could be used in rendering a 3D island and its surrounding sea.
3. Render a realistic implementation of a 3D island using OpenGL and GLSL shaders.
4. Simulate rising sea levels, rendering a separate sea entity using OpenGL and GLSL shaders.
5. Measure land mass loss depending on sea levels during the simulation.
6. Measure and compare the speed of initial generation and frame rates of the simulation in progress at different detail levels.

The first objective became irrelevant to the simulation as I decided to focus on the sea level rather than a time based simulation dependant on CO_2 levels, there were a lot more factors affecting global warming than just CO_2 levels, such as Ozone, Nitrous Oxide and Water Vapour.

The fifth objective was unfeasible by the time I had attempted it. In part due to doing majority of calculations on the shaders and the difficulty of returning data from these shaders.

Both of these objectives are discussed in more detail later in the dissertation.

1.3 Dissertation Structure

Planning

A small chapter on the planning of the dissertation that guided me through the project to ensure I was on track; this will give an insight into the original design and expected implementation of the simulation.

Research

Any research and relevant information used to achieve the aim and objectives of the dissertation are presented here, as well as any important data, resources or learning materials that I have used.

Implementation

A detailed step-by-step process of how I implemented the simulation.

This chapter explains why I decided to implement it that way, any issues I may have come across and how I went about fixing these issues.

Testing and Experimenting

Although testing was done during development using a code and fix process, this chapter discusses how I tested the simulation after development had stopped.

This chapter also includes how I went about approaching experiments on the simulation in order to establish tangible data to meet my objectives.

Results

The results of the simulation experiments, simple and advanced experiments are presented graphically and the results observed via these graphs are discussed in detail.

Evaluation

The evaluation discusses the finished simulation in relation to the completion of the objectives and aim of the dissertation, as well as the software engineering process.

Conclusion

The final chapter of the dissertation reflects on; the project as a whole, what I have learnt from the project and what improvements or additions could be made to the simulation in the future.

Chapter 2: Planning

2.1 Initial Plan

This is the original plan that I set out for this dissertation and expected it to change as I continued programming the simulation.

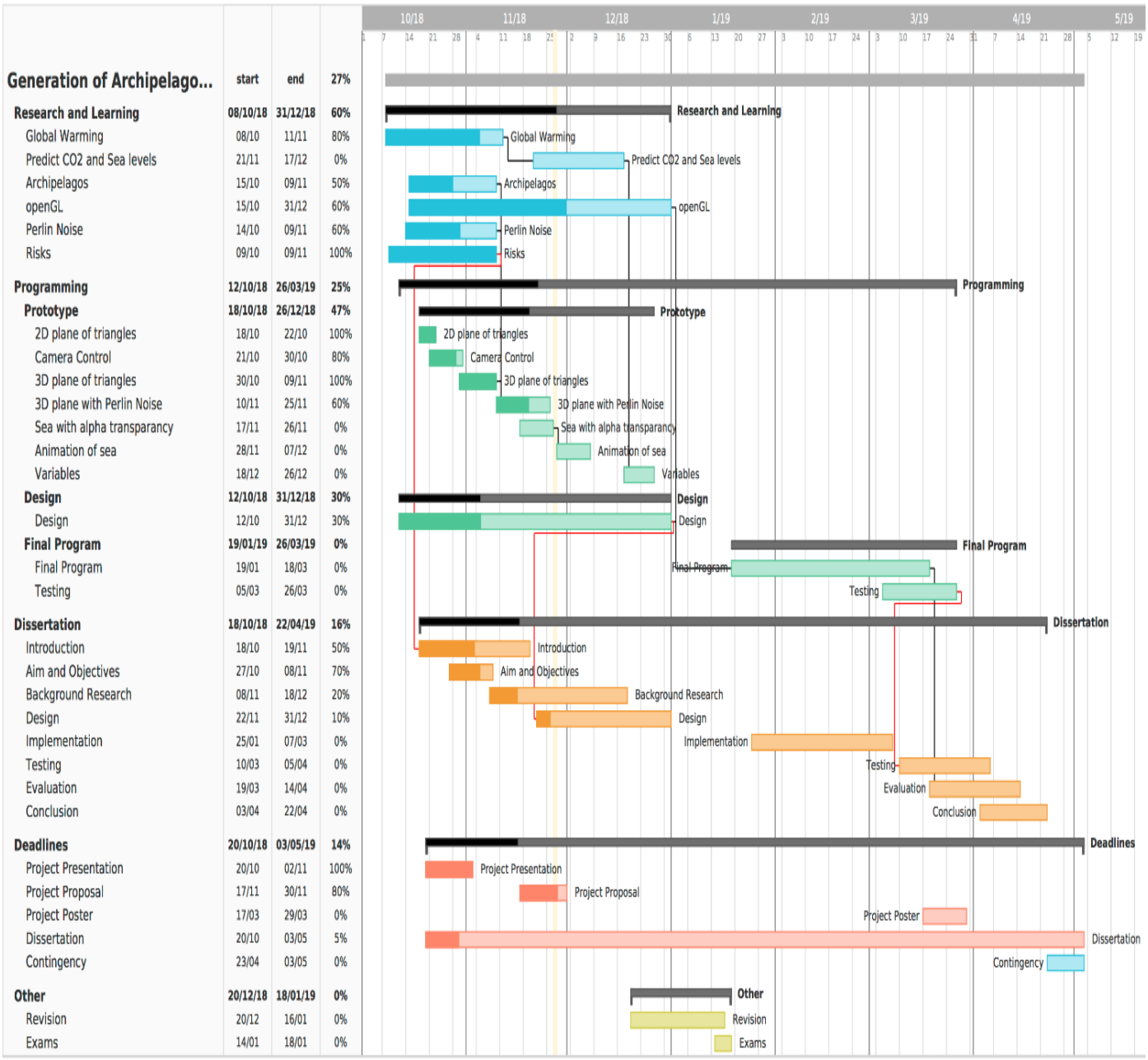


Figure 2: Initial Gantt Chart

I created a Gantt chart to help plan the project and dissertation, using an online tool called TeamGantt; this chart is shown in Figure 2 (above).

As can be seen in the Gantt chart and mentioned previously, I was originally going to generate an archipelago using Perlin noise, this was changed relatively early on into development and was altered, to suit a real life simulation of climate change instead, as Perlin noise is random, this would not have worked for a real life simulation.

This required using a heightmap of the island instead; the updated Gantt chart in the next section shows these changes.

2.2 Updated Plan

I have updated the completion percentages as I progressed through the dissertation, adding or removing tasks where necessary.

The updated Gantt chart, as at January 25th 2019, is shown in Figure 3 (below),

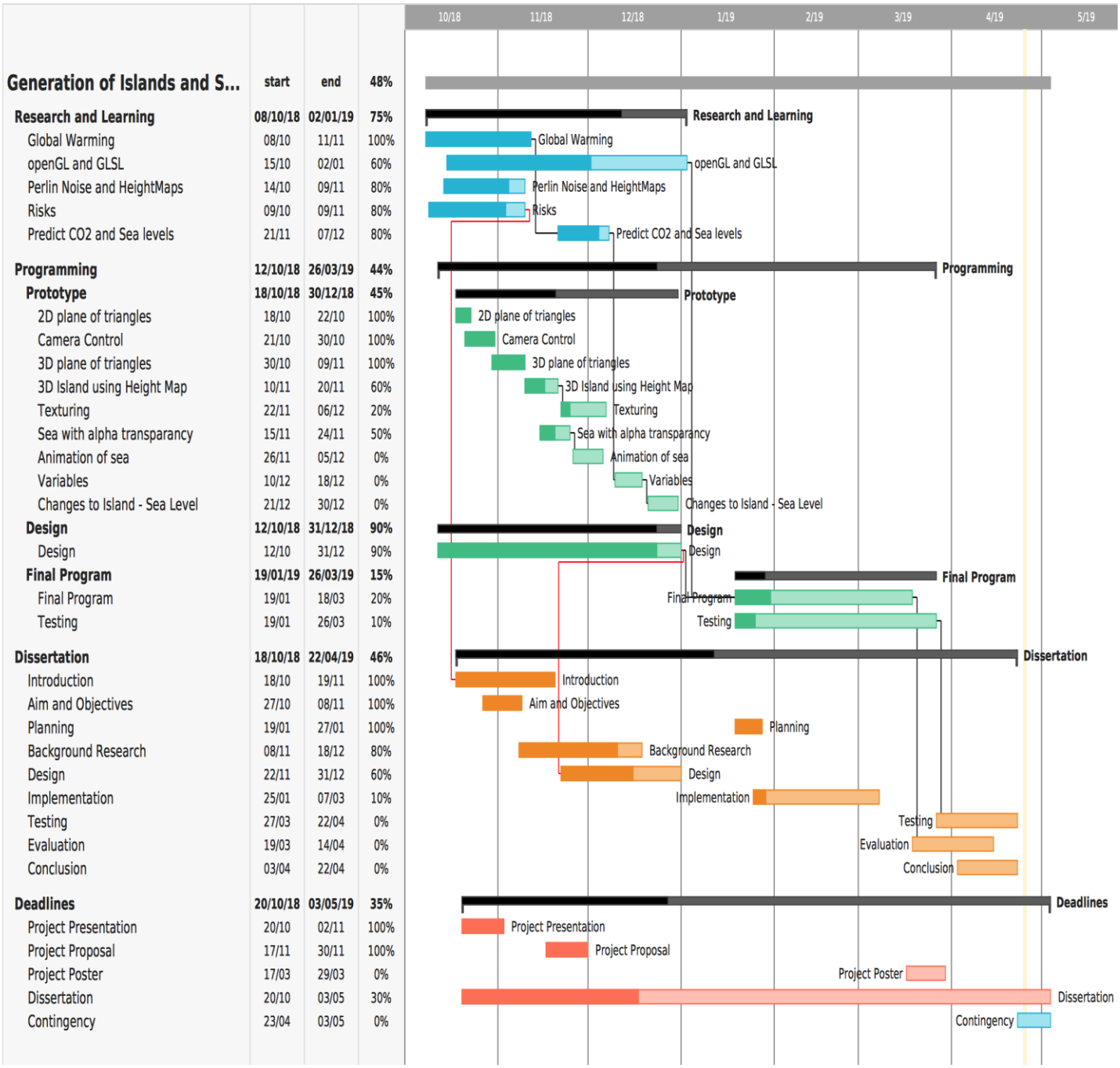


Figure 3: Updated Gantt Chart

By having an updated plan I could make sure that I was on track for finishing within the timescale, the plan also included some contingency time to complete the dissertation by and would have time If anything went wrong.

This time could also be used to proof read the dissertation further or fix any final issues with the simulation before the demonstration.

Many of the tasks in the plan changed due to researching deeper into a graphics technique or a change in the design.

I endeavoured not to change the design, however, if I did, it was usually in favour of making the final simulation look more realistic, more accurate or to improve performance.

The plan did not change from this point onwards except to update progress hence I have not included a more recent version of the plan.

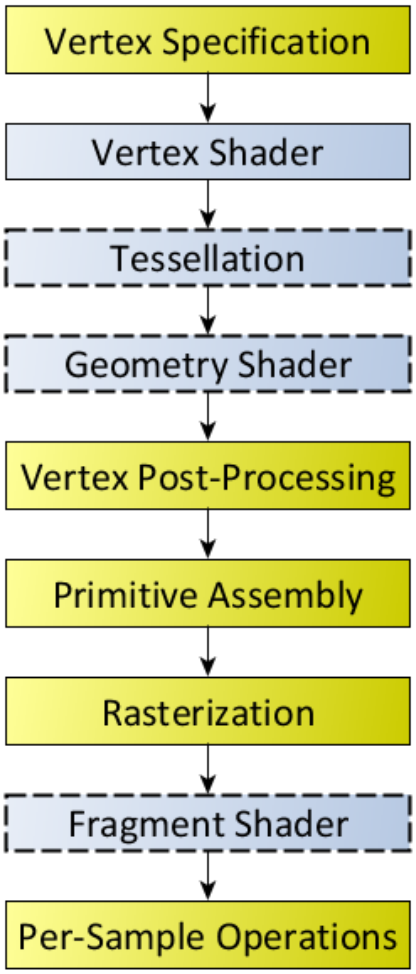
Chapter 3: Research

To achieve the best results with the simulation I chose a few topics to research, in both the fields of graphics programming and climate change.

The background material I read increased my understanding of the graphics pipeline and programming techniques that I could use for the simulation.

These research areas helped to enhance the simulation and improve its detail, I was also able to improve the accuracy of the simulation with a greater understanding of climate change and its effect on an island.

3.1 OpenGL and GLSL



OpenGL or Open Graphics Library is a library used for producing graphics to the screen and allows the programmer to interact with the hardware available to them such as the Graphics Processing Unit (GPU) on the graphics card, monitor and I/O devices.

GLSL is OpenGL's shader language; the syntax is very similar to that of the C programming language.

Shaders are the programmable part of the rendering pipeline shown in Figure 4 (left). (Khronos.org, 2019)[2]

They are often used to create graphics for animated movies and games by the film industry and games industry respectively to create stunning graphics.

The rendering pipeline is a set of steps, which follow each other to display graphics to the screen; some steps are optional if their functionality is not needed (represented by a broken border).

These shaders are what will allow me to exploit the graphics card's processing power; graphics cards are designed to work on large blocks of data in parallel and suited for many of the algorithms that graphics require to work efficiently.

Figure 4: OpenGL Rendering Pipeline

3.1.1 Languages and Toolkits

As OpenGL is an Application-Programming Interface (API) rather than a specific programming language, it can be used for practically any language with many libraries available.

Setting up an OpenGL capable window in itself can be quite complicated. Many libraries or toolkits have been developed to ease this process; this includes several that are used exclusively for creating just a window within which OpenGL can display images on.

Graphics Library Framework (GLFW) is one of the more recent libraries available. https://www.opengl.org/resources/libraries/windowtoolkits/

There were a couple of issues getting OpenGL to run on my laptop and desktop computer as they are both quite old, but after adjusting some of the settings and forcing the laptop and desktop to use their dedicated graphics cards when executing the program, I was able to use OpenGL 4.6, whereas the CPUs built in graphics processor in these machines could not as they were limited to OpenGL 3.1 with no updates available.

3.1.2 Alternatives

OpenGL is a powerful library and lets you control almost everything, as you are working very closely with the hardware.

Another option could have been to use a middleware engine such as Unity for this project; this would have made producing the simulation much easier and quicker and would have produced a more accurate simulation in the end. However, I wanted to learn the process that went into the graphics pipeline and using Unity would have abstracted away from some of that process.

OpenGL is equivalent to DirectX by Microsoft, however where OpenGL has a big advantage, is the ability for it to be used on multiple operating systems unlike DirectX, which is Windows only.

One of the newer API to be released for graphics rendering is Vulkan, although no doubt very powerful, I did not consider using it due to its scarcity of tutorials as with anything that's relatively new, but I would like to learn it in the future should the games industry pick it up.

The Graphics for Games module in the first semester provided a codebase which set up a lot of the underlying framework for OpenGL, such as producing a window for us to render our graphics on and has a lot of functions such as binding textures to shaders more easily and more succinctly.

The codebase uses C++ which is a common language used in the games industry and one I was keen to use, I had also been using C++ in another module during the first semester, Programming for Games.

As I had learnt a lot using this codebase during the module it seemed the best option to use this for my dissertation rather than trying to roll my own version.

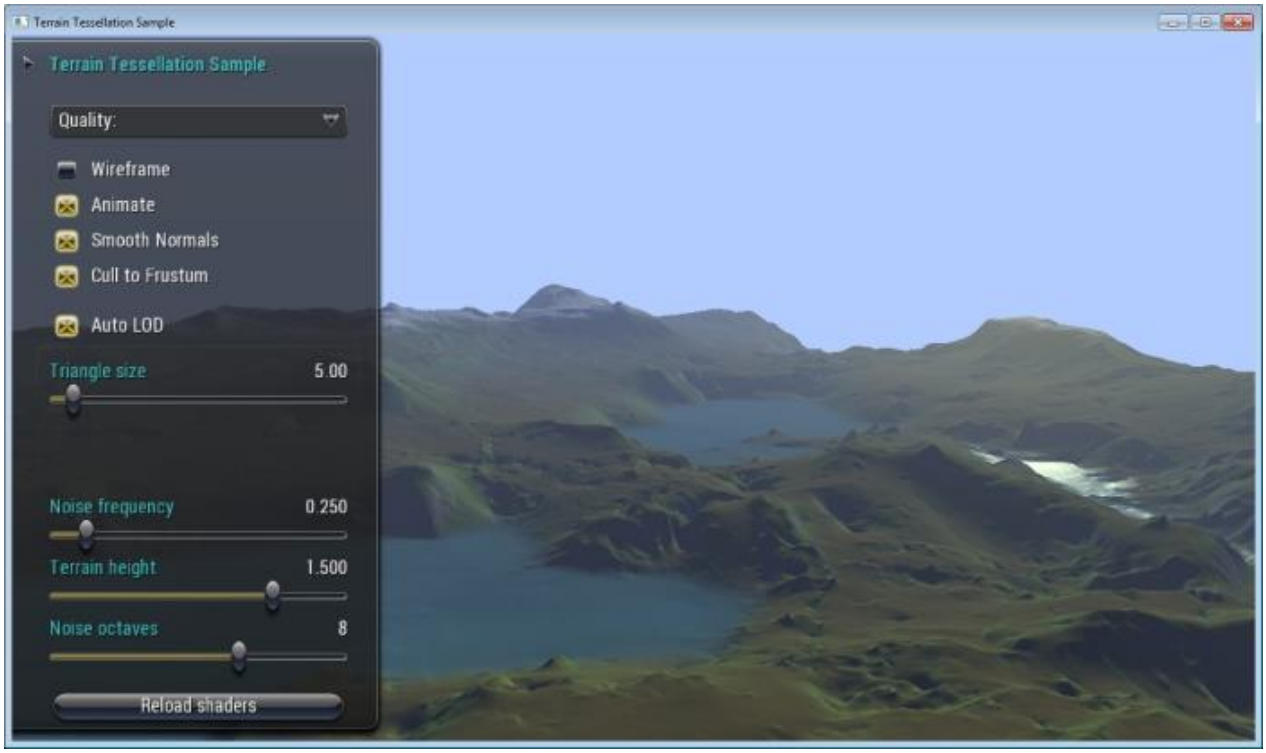


Figure 5: Example of Terrain Tessellation in OpenGL

Figure 5 (above) shows the capability of OpenGL in creating detailed terrains with tessellation.

3.2 Noise Generation and Heightmaps

I researched noise generation algorithms and heightmaps to determine what would be the best option for creating a realistic island.

There are several approaches to creating assets in graphics programming.

One approach could be to create every asset at the start and calling them when needed.

Another approach could be to create the assets on the fly, procedurally generating them, as needed, using algorithms to determine the requirements of the new asset. This could involve needing to add more terrain onto the edge of a plane as the camera moves towards it without creating holes in the plane; one approach to this is to use Perlin Noise.

3.2.1 Perlin Noise

Noise generation algorithms introduce randomness whilst not being just 'noise', such as the noise seen when tuning in an old television.

People are very good at detecting when something looks random, and noise on the television is as random looking as it gets, despite being caused by radio waves and cosmic radiation and therefore not random at all.

These algorithms work on smoothing out the random noise into a shape or structure that can be discerned as a pattern despite still being random, ripples in water is a good example of what a noise generation algorithm can do.

If this is applied to terrain in video games, where the user can see the result visually, the result doesn't look pleasing or represent a plausible terrain.

Depending on what the application of the noise is used for, will determine the amount of manipulation the noise needs for it to look correct.

In the case of generating an island to accurately simulate climate change, the noise would need to resemble an island that would be like existing islands on Earth. This would require a lot of smoothing and multiple layers of noise to create the uneven terrains found on Earth.

There are also other uses for noise, including increasing the detail of a simple texture, such as the grain of wood on a tree trunk by turning uniform circles into more natural age rings on a tree stump or log. (Vandevenne, 2004)[3]

Further uses include but are not limited to, mimicking handwritten text or more producing a procedural fire effect. (Flafla2.github.io, 2014)[4]

3.2.2 Heightmaps

Heightmaps are usually grey-scale images and their use can range from texture details to an entire island, archipelago or even the entire world. Heightmaps do not necessarily have to be real islands or textures and could be created by an artist's imagination rather than a satellite image from space or photograph.

Lighter coloured pixels on a heightmap normally represent an area with higher elevation, as the RGB (red, green, blue) values are higher, e.g. white would be 255,255,255 (1.0, 1.0, 1.0 in OpenGL) and black would be 0, 0, 0 (0.0, 0.0, 0.0 in OpenGL).

So, it therefore makes the most sense that the lower areas are represented with darker pixels, and areas of a single gradient are flat areas but could be either high or low.

As the picture is in greyscale, the RGB values are identical to one another, meaning only one of the colour channels (red channel for example) would be needed to tell a height.

However, this means we can only represent 256 different heights using a standard 8-bit RGB image, this can be dramatically increased by using a 24-bit RGB image using all 3 colour channels, bringing the heights we can represent to a much more acceptable 16,777,216. (En.wikipedia.org/wiki/Heightmap, 2019)[5]

If we wanted to, we could also use the alpha channel of RGBA (Red, Green, Blue, Alpha), bringing this number to almost 4.3 billion heights that could be represented.

Depending on the required accuracy of the map we could choose the appropriate option.

For an example, consider the peak of Everest (highest point on earth) is approximately 8,850 metres at its summit and the Challenger Deep (lowest point on earth) is approximately 10,916 metres deep giving us a difference in height of 19,766 metres that might need to be represented.

We could represent this per centimetre height, and would need 1,976,600 heights, so would require the 24bit RGB image, and could represent an approximate maximum accuracy of $1/8^{th}$ of a centimetre or 1.25mm using this image to its full capacity.

Taking it an extreme step further, using the alpha channel we could represent every height in the world to an approximate accuracy of $1/217^{th}$ of a millimetre or 4.6 micrometres (0.0046mm).

Of course, this information doesn't have to be just heights, it could hold other data, such as the texture to use on that point on the height map and just about any other possibility that could be held using any unused colour channels, which is fine for a single pixel. However, the problem then becomes the large amount of data the whole map would require.

Considering the earths circumference as approximately 40,000 kilometres, a 40,000,000 x 40,000,000 pixel image (1 pixel/metre) using 16bits of 2 Colour channels (big enough to represent heights of the world accurate to 1 metre) would result in a 2.84PB (petabyte) file, which is massive in itself to store and would take an unfeasibly long time to do anything computational upon, in a real-time graphics simulation.

There are also other uses for images that are very similar to heightmaps such as displacement maps and normal maps.

Displacement maps can be used for texturing much like Perlin noise, except the use of displacement maps does not produce a random looking result. Because of this it can be used for creating an effect such as footprints in snow or fingerprints, these displacement maps can be used for tessellation to create detailed terrain as shown in Figure 6 (below). (Widmark, 2012)[6]

Figure 6: Displacement Map Example (Detailed Terrain)

Normal maps are used for calculating correct lighting in graphics, they can be calculated and produced using the normal of a primitive relative to a light source, or simply put, whether the triangle is facing the light or away from it.

These can be calculated in real time, however, can use a lot of unnecessary processing power and an image is much more efficient once produced and loaded. If the model were not static the image would need to be recalculated each time it changed.

3.3 Climate Change

Climate change is a change in behaviour of the weather and the effects of this, therefore climate change encompasses glaciers melting due to global warming raising the earth's surface temperature.

Global warming causes climate change but is only one aspect of it.

Glaciers melting adds to the sea level by adding water volume that was once stored on land in the form of ice, however, rising temperatures also cause the sea to expand; much like how metal reacts to heat, by expanding. This in turn causes sea levels to rise.

3.3.1 Global Warming and CO_2 Levels

Rising temperatures across the world is known as global warming and has a correlation with CO_2 levels; CO_2 is a greenhouse gas but is only part of the greenhouse effect causing rising temperatures.

Figure 7 (below) shows the greenhouse effect in simple terms. This is a good thing for earth in moderation, without it the planet would be very cold, but too much and it becomes too hot.

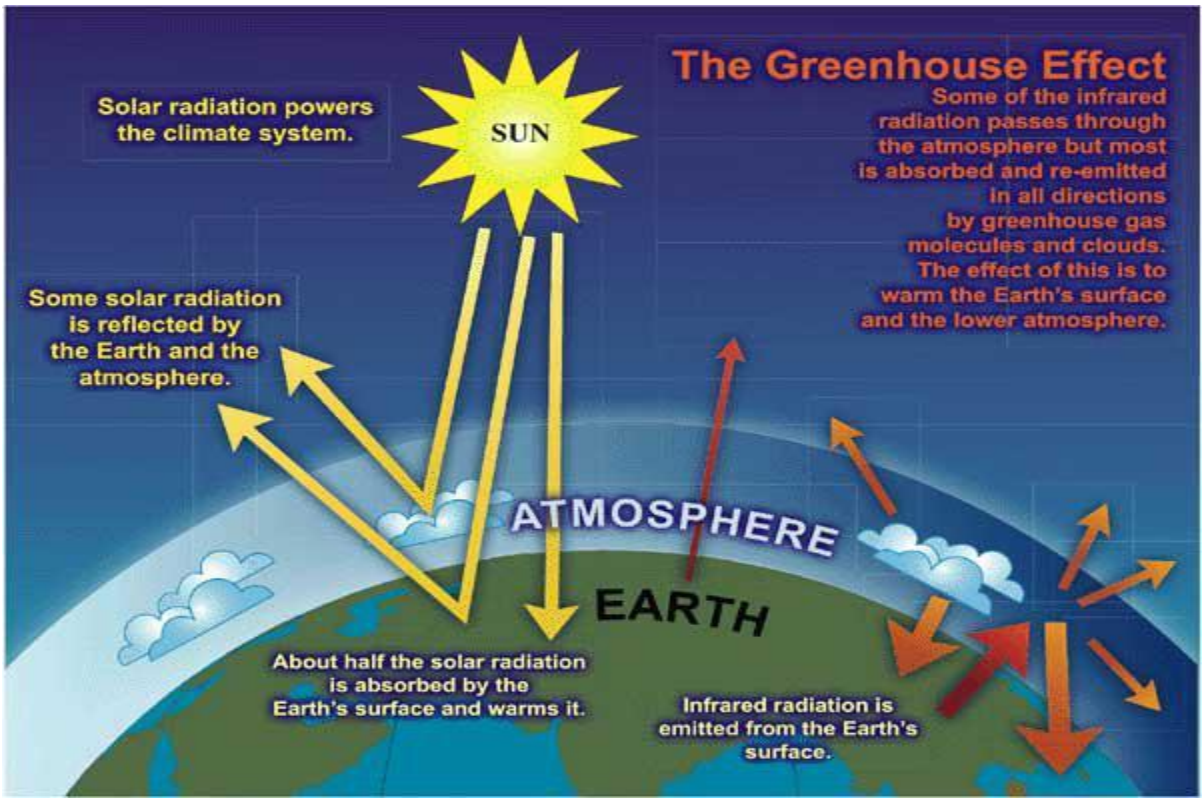


Figure 7: The Greenhouse Effect

Carbon Cycle

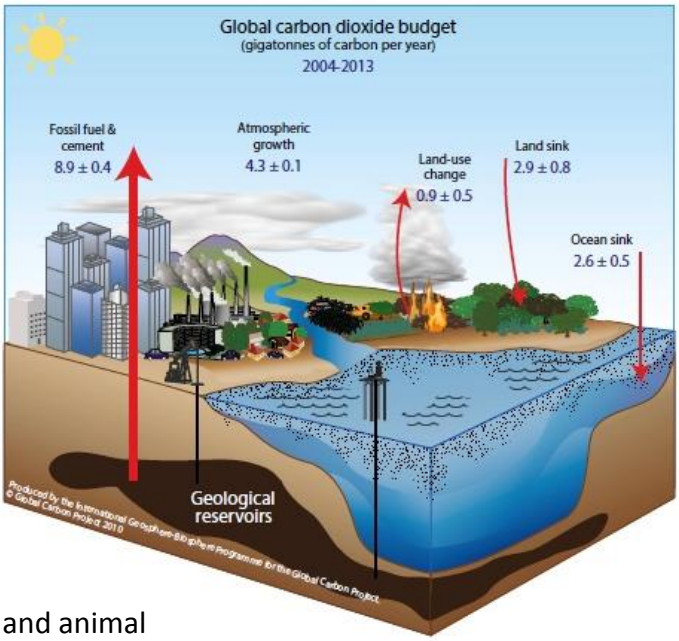


Figure 8: Carbon Cycle

The carbon cycle is the exchange of CO_2 between the atmosphere and earth.

There has been a natural exchange of CO_2 for millions of years, most likely since life began on earth, with the only contributing natural catalyst of climate change being solar fluctuations and volcanic eruptions.

Plant life and oceans absorb CO_2 from the atmosphere keeping natural CO_2 emissions from causing climate change too quickly.

Natural CO_2 emissions are caused by human and animal respiration and decomposition (What's Your Impact, 2018)[7], volcanoes erupting and underground magma releasing CO_2 through porous rocks and hot springs also add to the atmosphere.

The graph in Figure 9 (below) shows a steep increase in CO_2 emissions due to the burning of fossil fuels by humans since the industrial revolution in the early 1800s, and has increased heavily in the last 60 or so years.

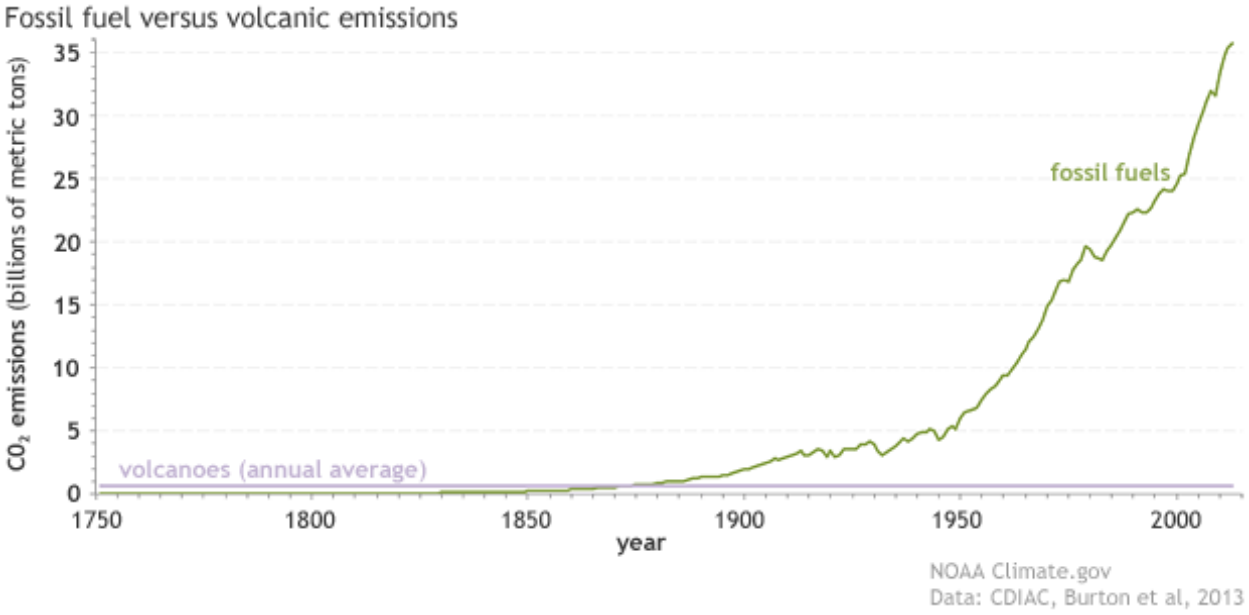


Figure 9: Fossil Fuel Emissions

3.3.2 Sea Level and its Consequence

It is documented that a sea level rise of up to 80 metres could occur should all the ice on earth melt entirely and make its way into the sea. (Poore, Tracey and Williams, 2019)[8]

It is predicted that in 2100 the sea level will have risen by 2 metres; this may not seem like a lot but is enough to affect a lot of low laying coastal areas. (Cumming, 2019)[9]

Figure 10: Flooding image

Sea levels rising will cause areas that humans inhabit to flood, reducing the area in which we can live safely and ultimately pushing us away from the coastline to higher elevations where the sea level has not yet reached.

Areas of farmland and even fresh water drinking reserves could be affected causing a shortage of fresh local food and potable water. Studies have shown that sea water can intrude up to 50% more underground than above ground, affecting welling and other methods of obtaining water for drinking (Scientific American, 2019)[10], this could mean we would need to import even more food, adding to CO_2 levels further. Below is an extract from the reference, humans settle close to water, it is a resource, but also a danger.

> *"40 percent of world population lives less than 40 miles (60 kilometers)*
>
> *from the shoreline"*

The image in Figure 11 (below) is the Nile Delta and is under threat from rising sea levels.

As can be seen on the map, much of the area is below sea level or only a few metres above it, the area provides Egypt with lots of resources and food. (Shenker, 2009)[11]

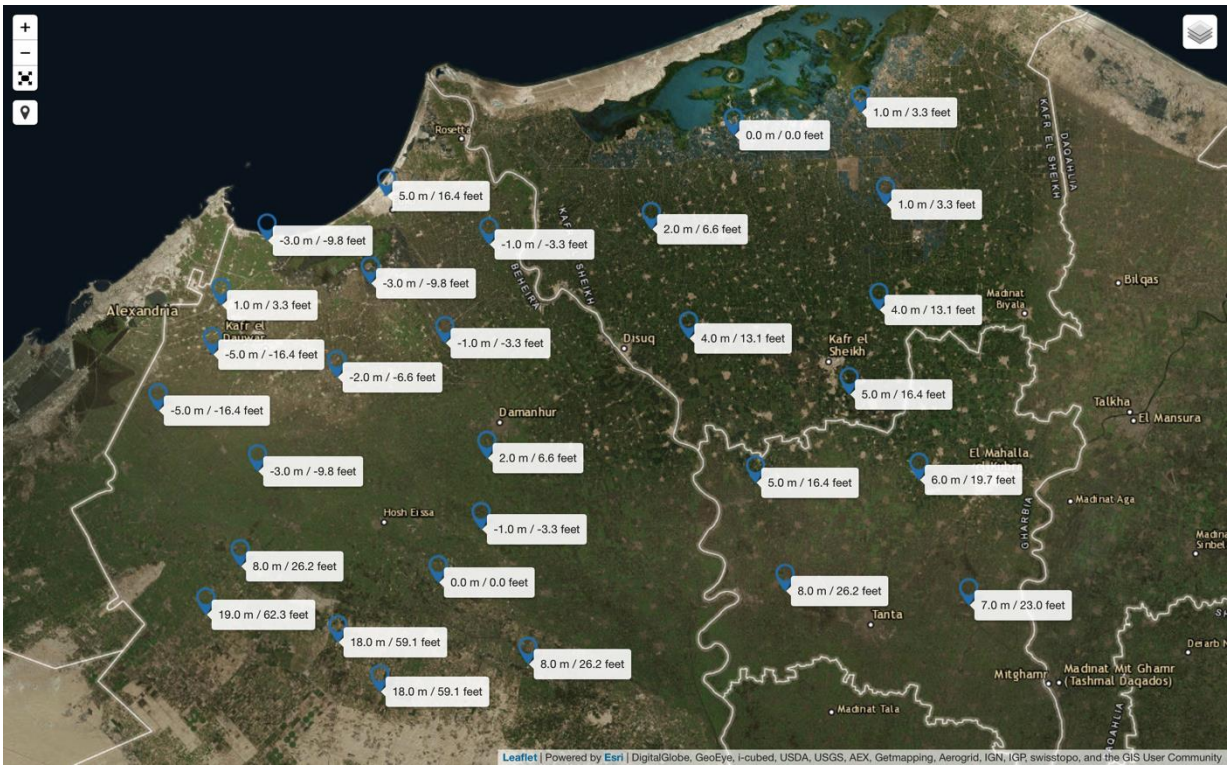


Figure 11: Elevation Map (Nile Delta)

A lot of people live there, and the area is also steeped in ancient Egyptian history which is under threat of being flooded should sea levels rise.

Figure 11 (above) was created using a tool by clicking on points on the map to show the approximate elevation above sea level of that area. (Freemaptools.com, 2019)[12]

The images and graphs in Figure 12 (below) on the following page were obtained from a study that explores the impact of sea levels rising upon developing country and goes very in depth. (Dasgupta, Meisner and More, 2007)[13]

I have chosen to only show a selection of figures from the study that related to the Nile delta in Egypt, specifically the effect on its population and agriculture.

Figure 2b
Middle East and North Africa region: Exposed population (5m SLR)

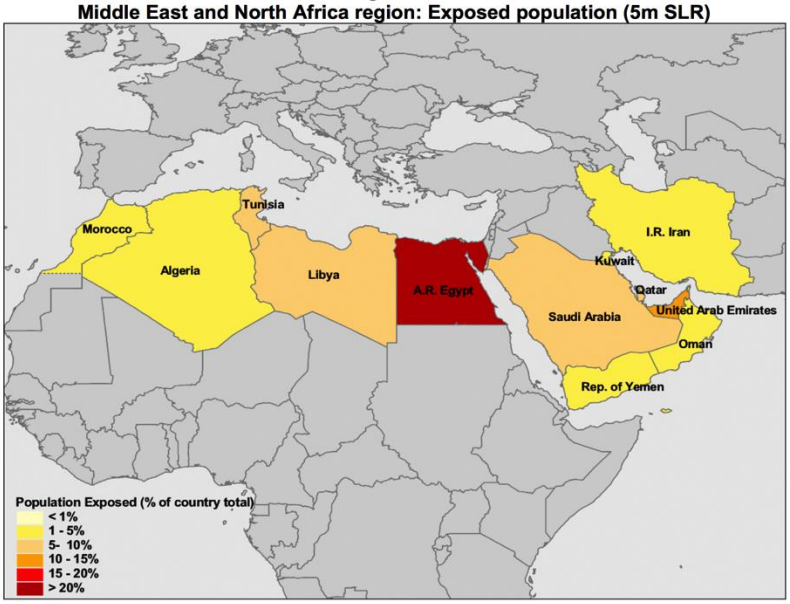


Figure 2c
Middle East and North Africa region: Population impacted

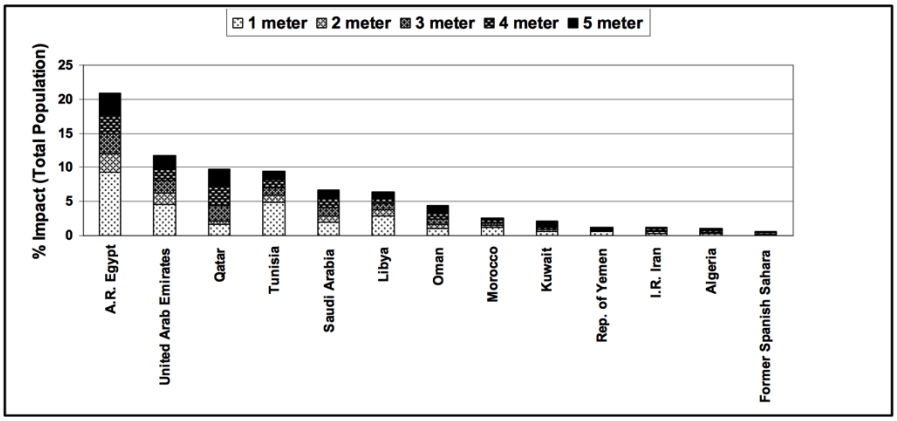


Figure 2f
Middle East and North Africa: Agricultural extent impacted

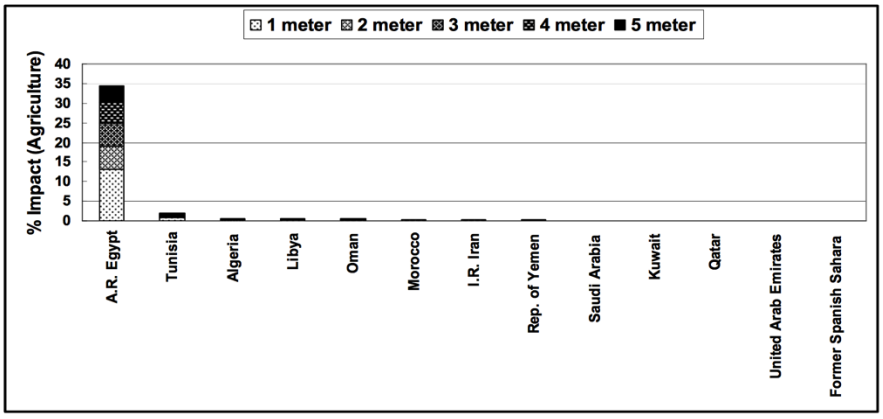


Figure 12: Egypt Impact of Sea Level Rising

Preventing Climate Change

> *"We are the first generation to know we are destroying our planet and the last one that can do anything about it"*
>
> Tanya Steele (Chief Executive, WWF)

Preventing climate change altogether is not possible, but reducing its impact and stopping its acceleration is, the information below is related to people's awareness and how we can prevent a climate change catastrophe.

Reducing greenhouse gas emissions is one way, but in order to counteract what has already been done, we need to take action to reverse it, the main elimination of carbon dioxide is by plants and trees, we need to plant more and stop deforestation.

Although it helps it cannot be the only solution to climate change, technology and science is improving every year and can be used to help, bringing either awareness to the situation by up to date studies or reducing CO_2 emissions with cleaner more efficient energy sources.

Climate change is a trending topic at the moment, with documentaries and a relatively hot Spring season; in 2009 the topic trended heavily on Google before dying off as shown in Figure 13 (below). The colours in the graph (red, orange and blue) relate to Global Warming, Climate Change and Sea Level respectively as search topics.

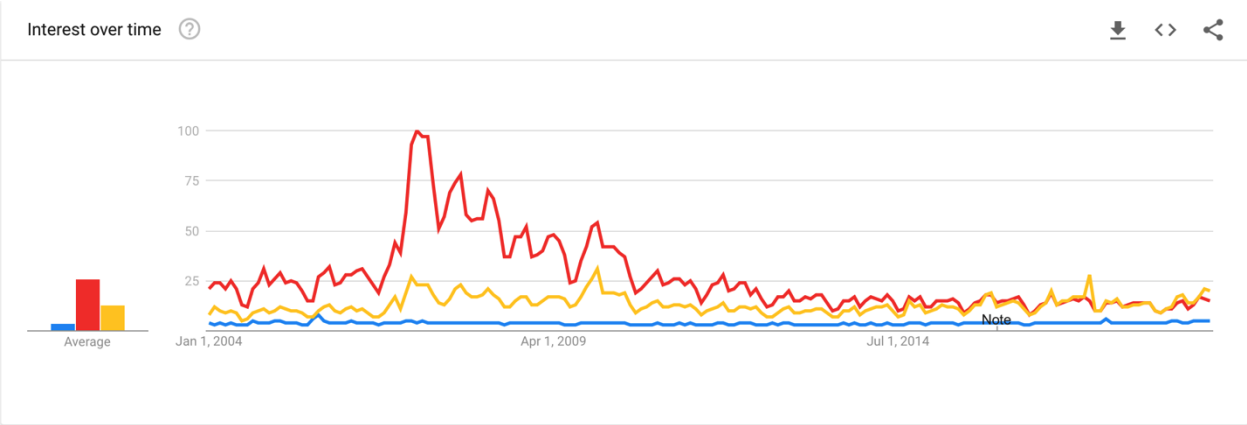


Figure 13: Interest Graph (Climate change)

Inland countries and cities that are not at risk of the sea level rises affecting them, at least not in the immediate term are simply not searching for these topics as much as coastal countries.

This can be seen from search trends by city, using Google topic search tool, Figure 14 (below).

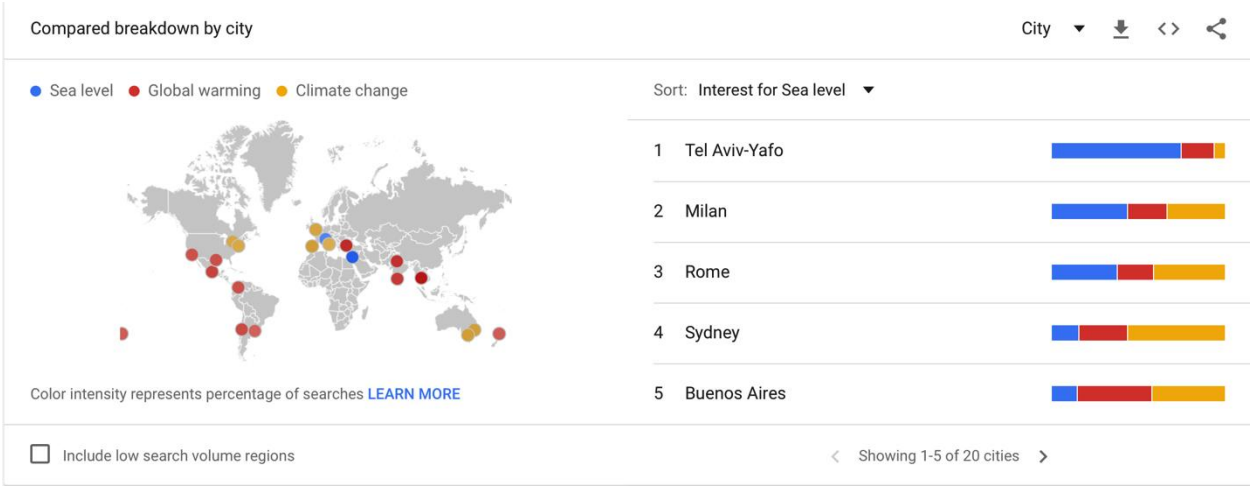


Figure 14: City Search Trends

Although a big proportion of the population lives near to the coast, possibly biasing this data. There are numerous cities that have large populations that do not reside by the coast and are still not searching about climate change and sea levels.

Drastic measures may need to be taken in the future such as constructing costly sea defense barriers; this, however, only delays the issue and does not solve global warming or the risk of extinction to species such as the polar bear.

At this point in time, we may need to actually intervene using techniques such as air and sea CO_2 capture, whereby the CO_2 is extracted and stored elsewhere so it is not released, this however requires a lot of energy, so would need to use renewable energy sources such as wind, solar and hydro to result in a net loss of CO_2 being removed from the atmosphere. (Mulligan, Ellison and Levin, 2018)[14]

3.4 Existing Work

As well as the graphic in Figure 1, there are several other works that I have come across relating to sea levels and climate change.

NASA has a dedicated subdomain to climate change, (Climate.nasa.gov, 2019)[15] and have models of earth with measurements such as CO_2 levels and Sea Level Variation taken using satellites as shown in Figure 15 (below).

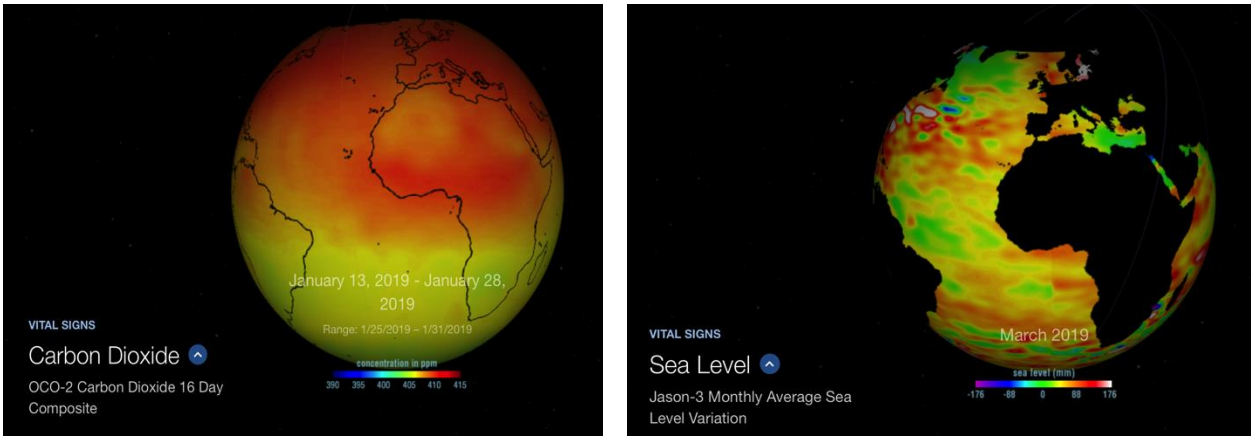


Figure 15: Satellite Measurements

One of the more interesting works they have done is a 3D simulation of Antarctica melting and showing the sea level rise approximation for different cities of the world.

Figure 16 (below) is a screenshot of the simulation running; it predicts a 2m rise for London over the next 500 years from Antarctica partially melting alone. (Vesl.jpl.nasa.gov, 2019)[16]

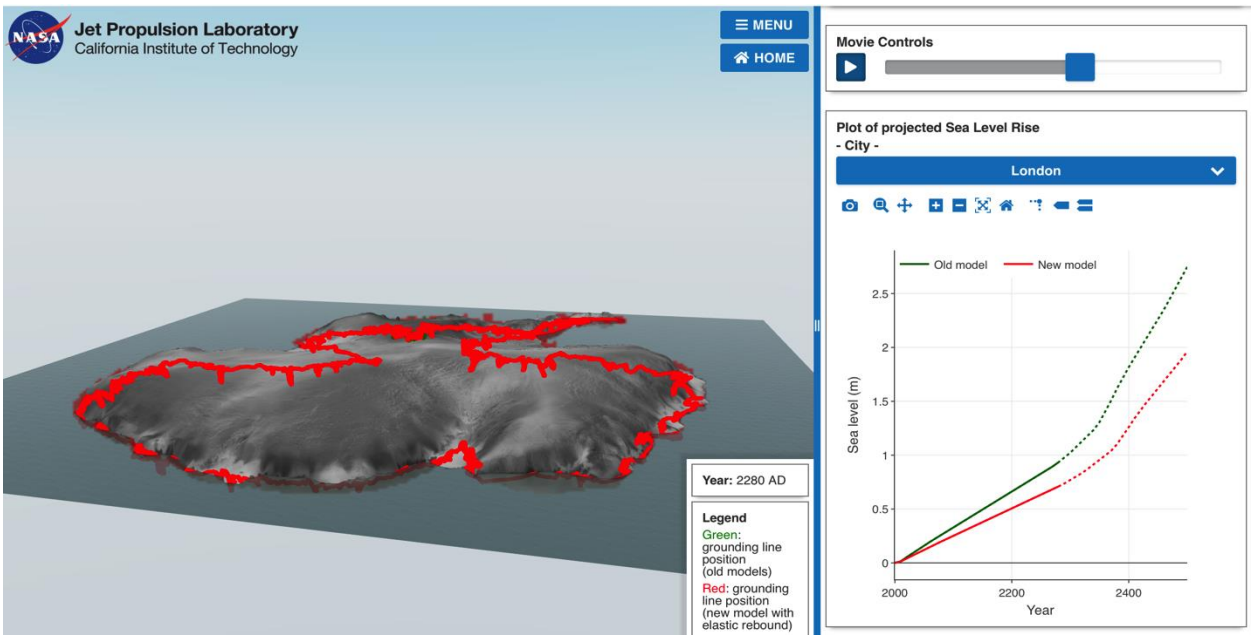


Figure 16: Antarctica Simulation

The simulation in Figure 17 (below) is very close to what I attempted to simulate, even though I didn't find this until I was much of the way through programming my simulation.

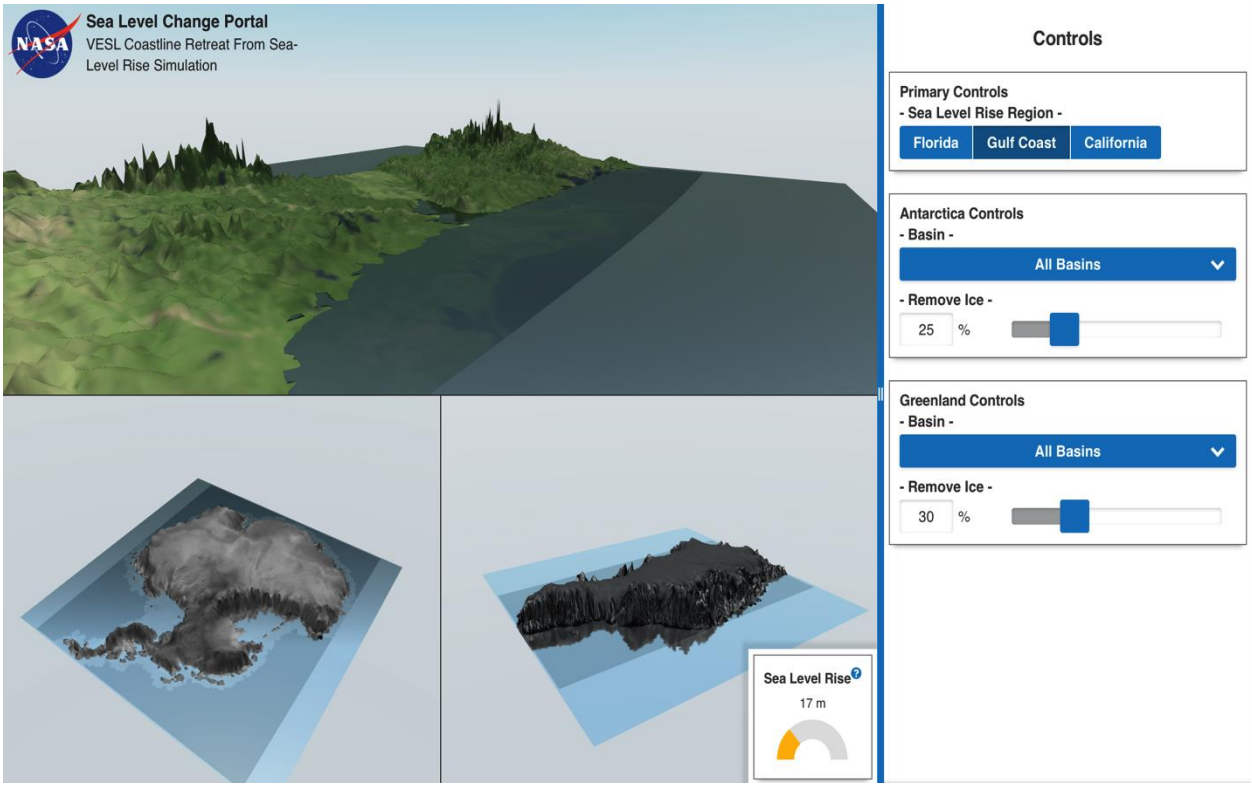


Figure 17: Sea Level Simulation

<u>Positives</u>

- Correlation and interactive experience for user, with Ice melting and the sea level rising in top model, as well as showing the melting of each ice basin in the lower 2 models.
- Top model looks good graphically from a distance and sea transparency is good albeit a little bit dark.

<u>Negatives</u>

- Limited regions that can be simulated, it would be better if able to upload a heightmap.
- No detailed texturing used, even when zooming in.
- Some of the heights are a little too steep and unnatural looking.

Overall the simulation (Sealevel.nasa.gov, 2019)[17] is very good and appears to be accurate, but could be improved aesthetically, although the above points are my opinions only.

Chapter 4: Implementation

I have separated this chapter into several sections in order to explain how I went about implementing what I had researched about OpenGL, GLSL shaders and climate change.

4.1 Design

The design stage of the simulation was relatively simple and laid foundations for my approach to implementing the ideas I had, using the researched material and techniques that I had learnt.

I decided on an approach that uses a separate sea entity, this could be manipulated independently to the island and would be able to use the OpenGL depth test, rather than attempting to use a shader to simulate the water rising and blending with the island.

A simple representation of the architecture is shown in Figure 18 (below). As I only edited a few classes inside the codebase, of which there are numerous, I have not detailed each class within.

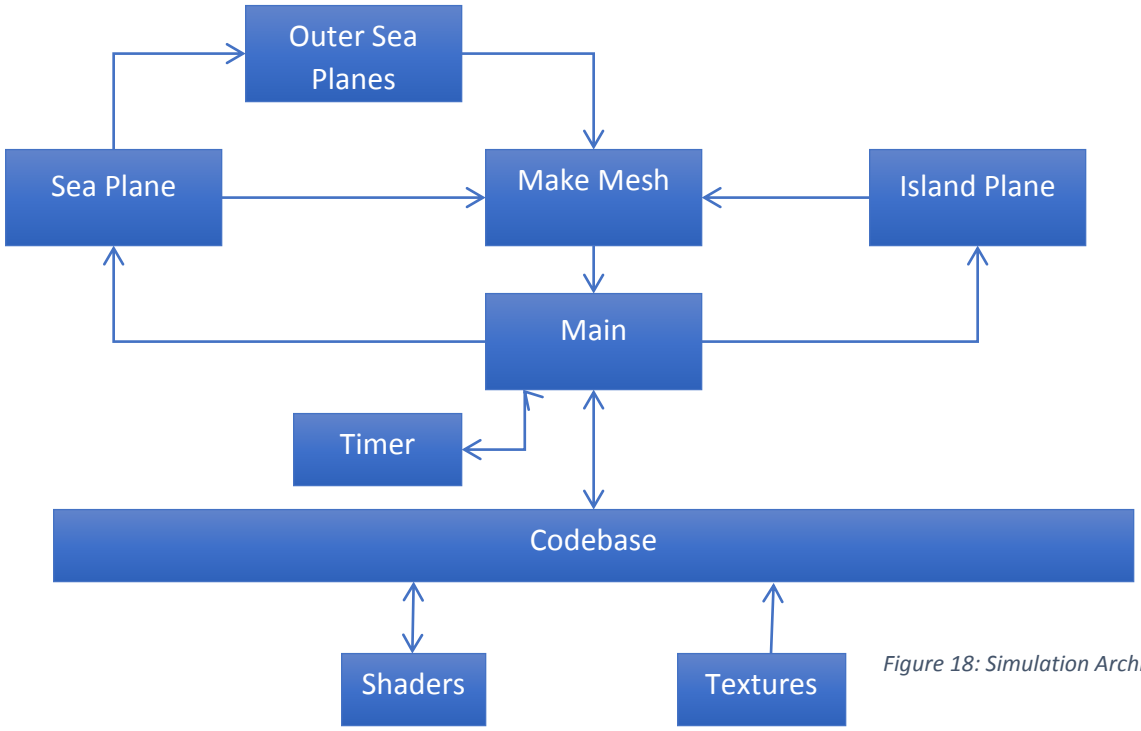


Figure 18: Simulation Architecture (Simple)

The shaders and textures I use are explained in further detail in Resources (4.2).

Sea Plane

The sea is a simple plane of triangles that uses tessellation and a texture.

I decided to add a technique known as "Watertight Tessellation" (Bunnell, 2005)[18] as I wanted to create more detail and smaller patches for tessellation of the sea around the island itself.

Surrounding this central plane would be patches to be tessellated to a lesser level, which would save on the graphics processing power.

The sea would have a water texture only and originally be tessellated to include waves, however when interacting with the island the waves created issues with clipping in the middle of the island and weren't necessary, so were removed.

Island Plane

The design originally planned on using Perlin noise in order to create a realistic archipelago or set of islands, however, a short way into programming the simulation, I decided to focus on a real-life scenario, more specifically, I would use a heightmap of a real island.

In this case, I chose to use the UK, but the code could be applied to any heightmap providing other resources accompanying this map were available, in the case they were not available, several lines of code would need to be removed or commented out.

The island has textures of grass, rock and snow to represent the island and its mountainous areas.

To simulate erosion on the island, sand will replace the textures where the erosion took place.

4.2 Resources

The following resources are what I used in order to simulate the effects of climate change on the UK. The textures are used to enhance the look of the simulation, and the shaders that have been used to render the results onto the screen.

4.2.1 Island Maps

Height Map

A detailed high-resolution 8bit greyscale map of the UK; the light areas represent the higher elevations and the dark areas represent the lower elevations.

Using this I am able, with shaders, to create a basic 3D map of the UK with accurate elevation.

This image was obtained from imgur.com with no obvious accreditation to an author and is shown in Figure 19 (right).

Figure 19: Height Map

Major Rivers Map

The rivers map is a stripped-down edit of several maps *Appendix 2 – River Maps (pre-edit)* which have been combined and touched up to create the result shown in Figure 20 (right).

Having rivers on the simulation will show where the main rivers are flowing in the UK and show how sea levels rising will interact with them.

Unfortunately, I was unable to find an appropriate map that had the rivers of Ireland, as I had to ensure they lined up with the other maps in terms of scale and skew, the Scotland, England and Wales rivers needed editing to ensure they lined up as accurately as possible.

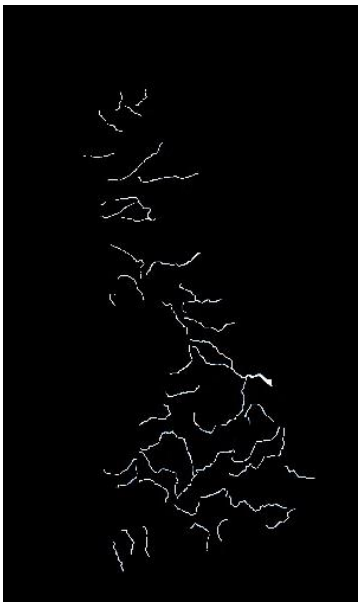


Figure 20: River Map Edit

Distance from Coast Map

A very simple map I created using Photoshop by editing the original UK heightmap shown in Figure 19 (above), to ensure a perfect match. This map enables me to see the coastal areas in GLSL by utilizing the red channel in RGB.

The lower red channel value on the coastal areas represent a lesser resistance to the erosion and sea level aspect of the simulation when the sea level rises.

The most likely areas to be affected by rising sea levels would be the coasts. Water flows to find the lowest point but can't get to the lowest points if there are higher elevations surrounding that area hence the need for this map.

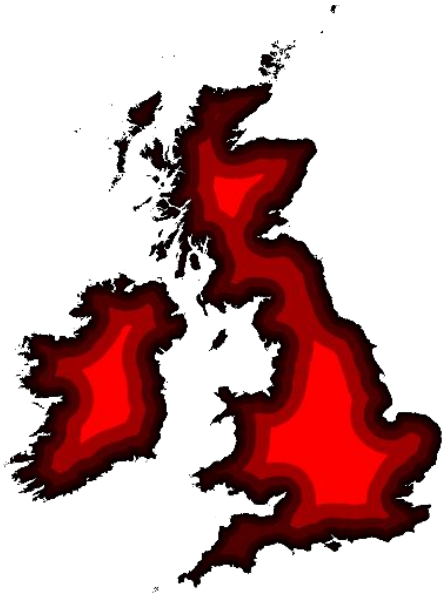


Figure 21: Distance From Coast Map

Ground Strength Map

The ground strength map shows the least permeable ground areas and as such will be the last parts to erode. This will be more relevant for a longer time span, however, will still have a small effect on near future erosion, and therefore I have included it.

There is a very dark green area known as "The Wash" which runs from Skegness on the north side, to Hunstanton on the south. This area is mostly made up of sedimentary material deposits and has caused once coastal towns such as King's Lynn to be further inland than they used to be. (En.wikipedia.org, 2019)[19]

I was unable to find a map that included the ground strength of Ireland.

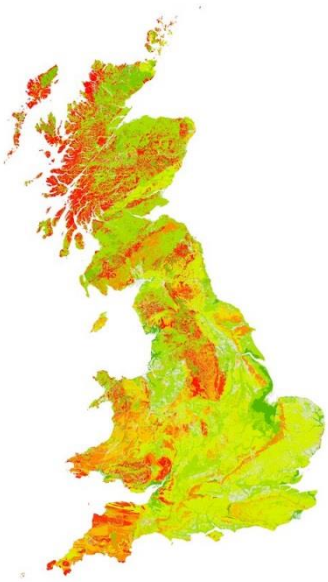


Figure 22: Ground Strength Map

Population Map

The population map shows the hot spots of human habitation in the UK. As can be expected, this shows the areas of major cities and the distinct lack of habitation in the highest elevation areas (magenta).

The colours used in Figure 23 (right) would not work well in my simulation, so I have edited this image *Appendix 3 –Population Map (Greyscale)* using photoshop, setting each colour to a degree of lightness so the differences in population are correct in the now greyscale image, the most populous areas are red, followed by yellow, and light blue, the least populous areas being dark blue and magenta, these are now light to dark respectively.

The reason I added this is to show what the level of population is in the areas that are under threat of sea levels rising.

This can be toggled on and off during the simulation.

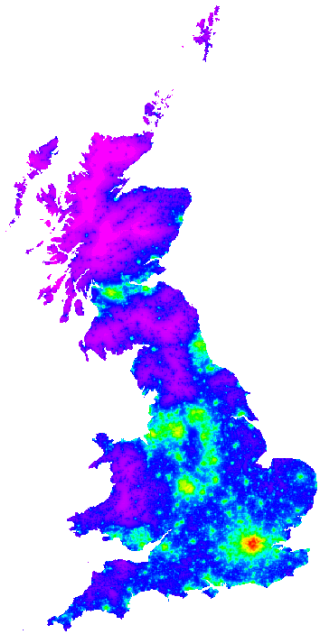


Figure 23: Population Map

4.2.2 Textures

The textures that I used for the simulation are few in numbers; however, they provide a relatively pleasing look to the simulation.

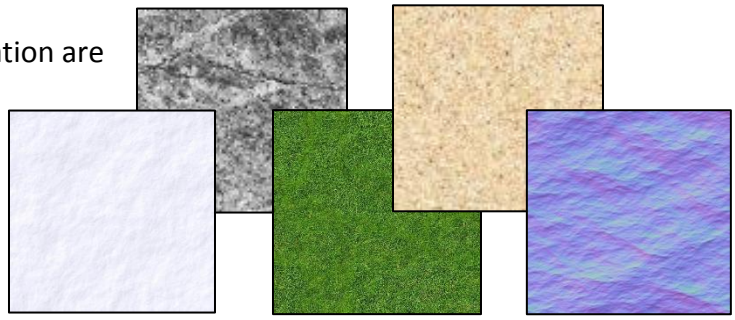


Figure 24: Textures used in simulation (Snow, Rock, Grass, Sand, Water)

The snow texture is based on elevation using the heightmap. As mentioned previously, this is set in the fragment shader and everything below this elevation is grass. The rock texture uses the ground hardness map to determine where it should be used.

The rock and grass texture are blended together to provide a smooth transition between the heights rather than a harsh split of one texture then the next.

The sand texture replaces other textures that are inundated by water as the sea level rises during the simulation; this is usually the grass texture that is being replaced due to the erosion aspects mentioned previously.

The water texture is semi-transparent and has an alpha channel. As the sea level rises, the texture blends with the textures below it, due to it being a separate entity.

4.2.3 Shaders

The shaders used in the programming of the simulation included multiples of the following types of shaders:

- Vertex Shaders
- Fragment Shaders
- Tessellation Control Shaders
- Tessellation Evaluation Shaders

Vertex Shaders

The vertex shader can be used for a few applications, but not in this case, the vertex shader is the only mandatory shader, even if it just passes data on without doing anything.

Fragment Shaders

The fragment shader is used to calculate the colour of each fragment on the model, there may be multiple fragments per pixel but would not be multiple pixels per fragment.

The reason for this is due to being able to have two models behind one another and have transparency, which would need to calculate a colour based on this blending of colours.

Tessellation Control Shaders

The tessellation control shader is used to tell the GPU the number of triangles to split the patch into as well as how to do so that best fits the application of tessellation. And can be used to match the correct number of triangles and size to a neighboring plane, tessellation is optional and subsequently so are the shaders.

Tessellation Evaluation Shaders

The tessellation evaluation shader performs the splitting of the patch and is a mandatory part of tessellation if wishing to use tessellation, whereas the tessellation control shader is optional, as the vertex shader values would be used if the tessellation control shader were not present.

4.3 Programming

Quad Plane

To shape how the simulation would look I started by rendering a plane with 2 triangles (known as a quad) and rotated it 90 degrees so that it would lay flat. This would be the base for the island and a second similar plane the base for the sea.

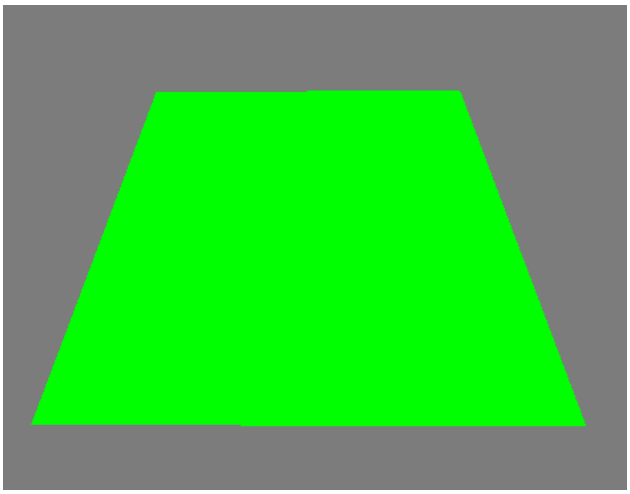


Figure 25: Filled Quad

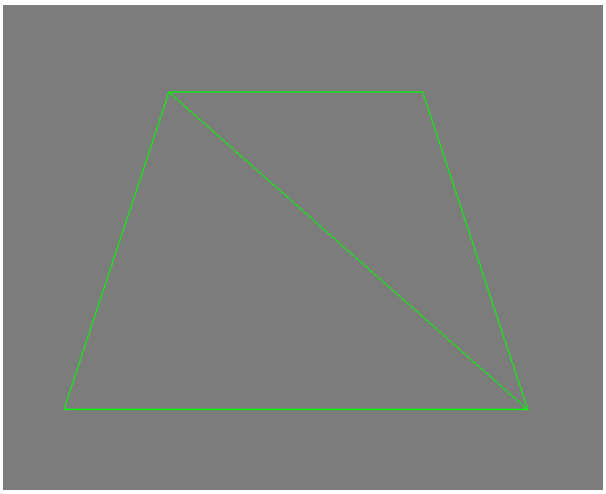


Figure 26: Wireframe Quad

They are coloured appropriately in green and blue by passing an RGBA value to the renderer.

Adding the second sea plane too close to the island plane caused Z-Fighting.

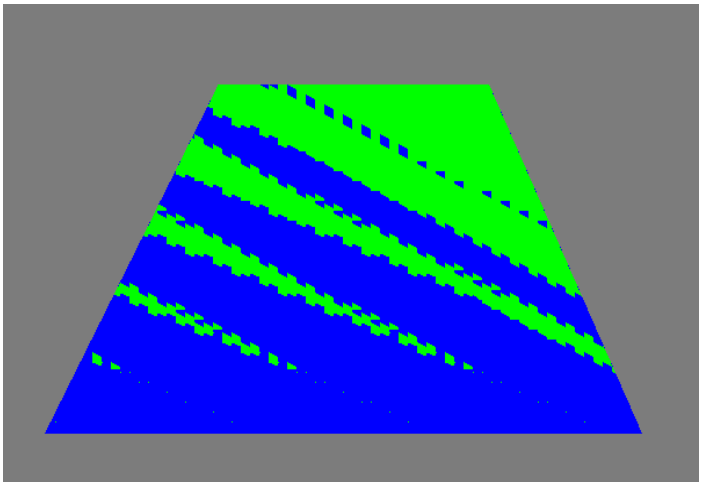


Figure 27: Z-Fighting Planes

Z-Fighting is where the GPU doesn't know which colour should be displayed due to identical or very similar Z-Values. This was rectified by increasing the distance between the planes, but is not a good solution, as the sea will clip with the island during simulation.

I also halved the size of the island plane at this point to surround the island with sea on all sides.

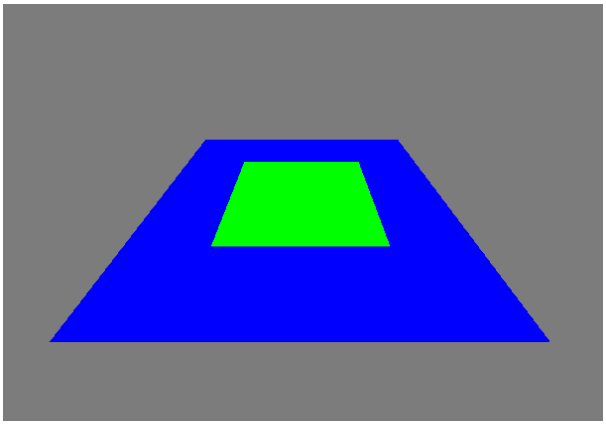


Figure 28: Half size, No Z-Fighting (Filled)

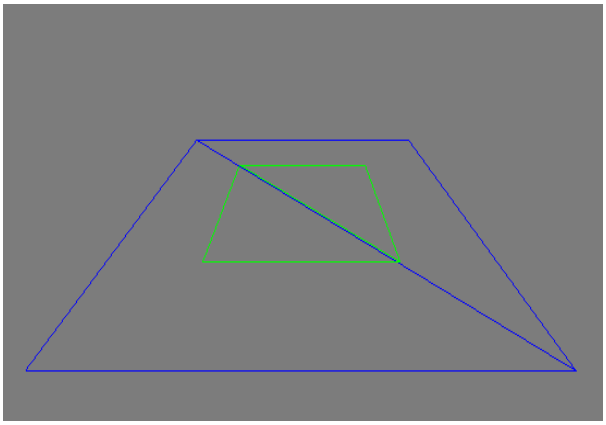


Figure 29: Half size, No Z-Fighting (Wireframe)

Camera and Controls

The camera was an important aspect of the simulation to complete early on in order to debug any issues I was having, by being able to inspect the island from different angles and ensure that any changes being made looked good from both a distance and up close.

I already had the wireframe toggle added at this point, alongside alpha blending toggles and depth toggles. These two latter toggles were removed however as they are set in the code.

The keyboard controls I used were as follows:

- WASD to move the X-axis and Y-axis
- Q and Z to zoom in and out using the Z-axis
- F to toggle the Wireframe
- P and L to increase and decrease island tessellation (added later)
- O and K to increase and decrease sea tessellation (added later)

The mouse controls I used were as follows:

- Left click and drag
  - (Left/Right) to change the yaw
  - (Up/Down) to change the pitch
- Scroll wheel to zoom in and out using z-axis

Technically the camera doesn't actually move in OpenGL, there is no such thing as a camera object as such, but what it does do is move the scene the opposite way of what we would regard the camera moving, however this isn't entirely accurate, say we had a multiplayer game, we couldn't all be moving the world around at the same time, that would cause chaos, what is really happening is the view of the world is changing, world coordinates stay the same, and a character may be placed on that world in different locations as they move. (Overvoorde, 2012)[20]

To program the controls, I put them inside a while loop that runs until the simulation closes, this would check if any of the keys or mouse buttons had been pressed or held down depending on the appropriate need.

If the framerate was extremely low, sometimes a key press may be missed as it was being checked during the loop.

Heightmap

Adding the heightmap to the plane using the code shown in *Appendix 4 –Two Triangle Quad Planes Code,* gives us a black background and tints the greyscale texture a green colour due to the original plane being green. This still allows us to see the shades representing the heights.

To remove the black background, I added code to the fragment shader to discard any fragments that were black or close to black.

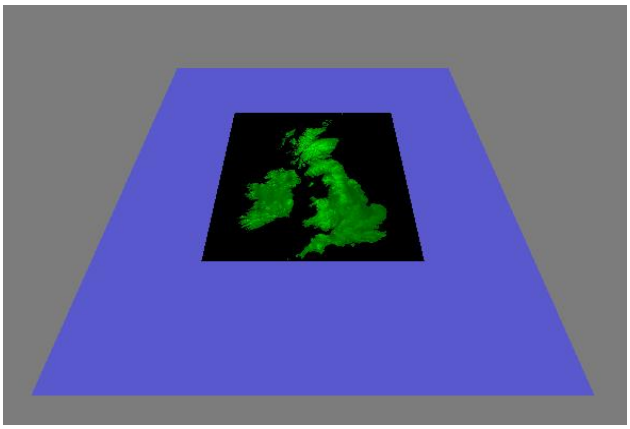


Figure 30: Island Texture Added

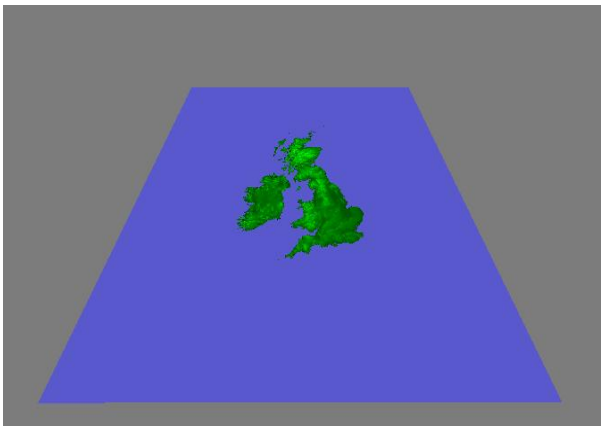


Figure 31: Black Background Removed

From a distance this looks ok and could even be mistaken for 3D, however on closer inspection, it is just a flat image on the flat plane.

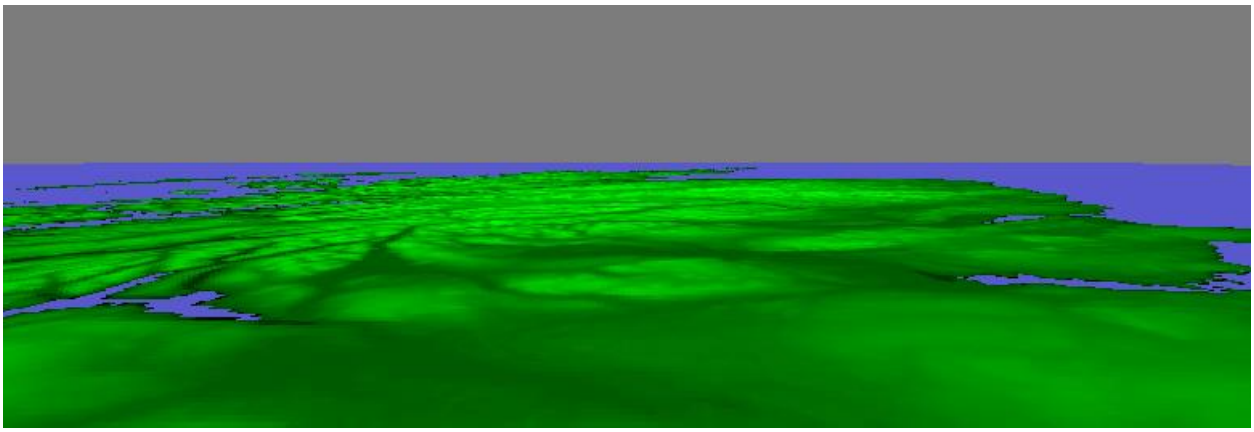


Figure 32: Flat Surface

Tessellation

To use tessellation and control its level, I needed both a Tessellation Control Shader (TCS) and a Tessellation Evaluation Shader (TES). The code used in these shaders is adapted from the shaders used in the graphics module.

I needed to change the primitive type to a patch rather than the triangles I was using previously a patch does not necessarily have to be any specific shape although I chose to use 4 vertices for the patch to cover the quad I had used for the plane.

Tessellating this singular patch to the maximum value of 64, for both inner and outer levels in the TCS gave me the below result, this does not look all that different to the original pre-tessellation result, but you can see slight variation in height.

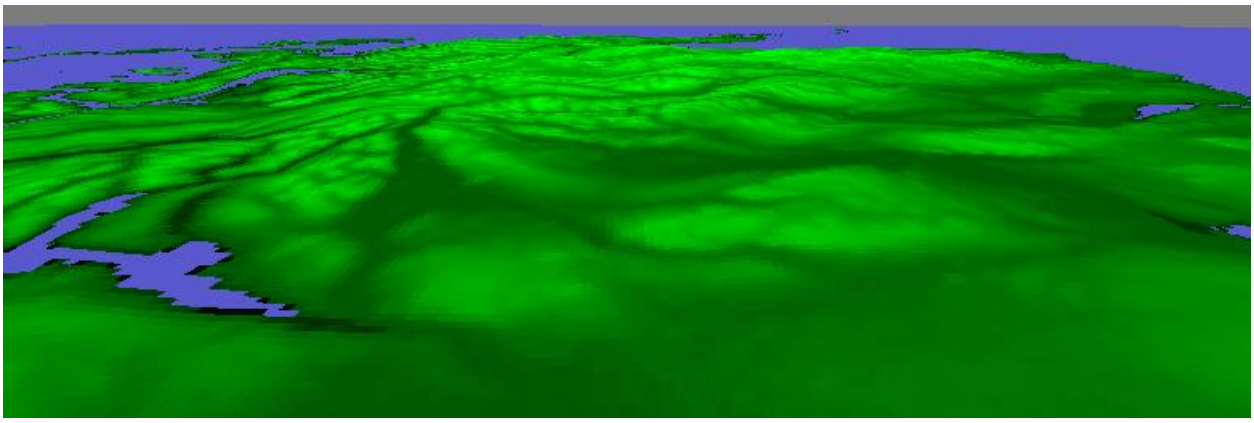


Figure 33: Maximum Tessellation of a Quad (Filled)

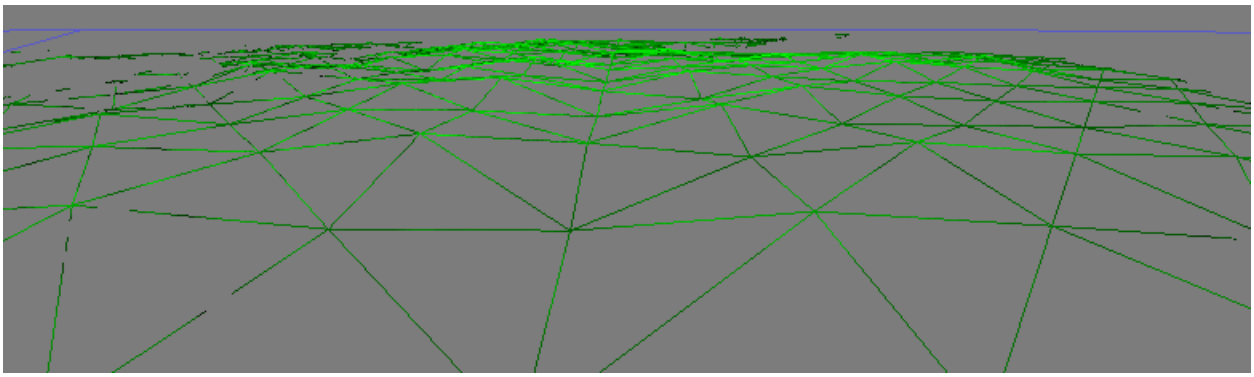


Figure 34: Maximum Tessellation of a Quad (Wireframe)

As the simulation needed greater height detail than what the maximum tessellation can provide, I needed to increase the amount of patches that the texture covered, this would allow me to increase the amount of triangles inside the model and therefore the amount of detail and steps in heights that could be used.

Grid of Quads

I recreated the plane with a grid of quads, using the sea first; I achieved this by using a nested-for loop (a loop within a loop) with rows and columns, there was an issue using this approach when reaching the end of the columns, it was stretching the triangle from the end on the right to begin a new row on the left.

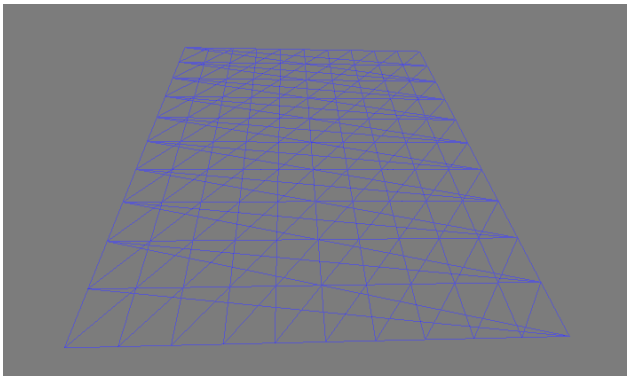


Figure 35: Grid Of Quads (Incorrect)

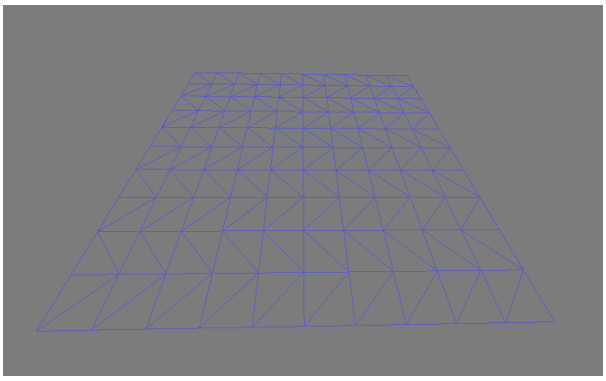


Figure 36: Grid Of Quads (Correct)

To fix this I had to reverse the loop every other row. *Appendix 5 – Grid of Quads Code*

Figure 36 (above - right) shows the resulting fix after changing the code, this would mean textures wouldn't be distorted and stretched across the plane and was important to correct.

Figure 37 (below) shows the order in which the vertices are being placed using this code with a triangle strip.

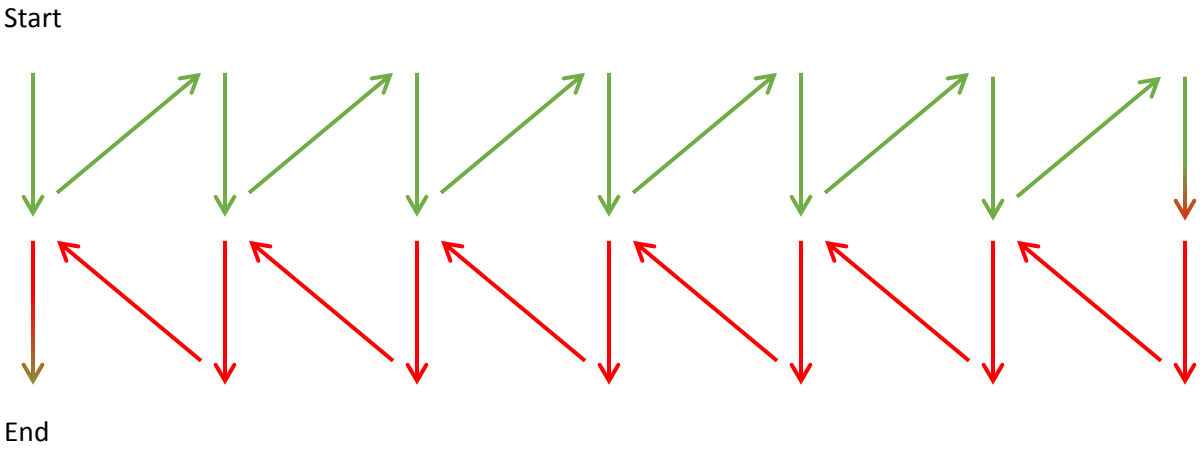


Figure 37: Triangle Strip Vertices Order

I made the size of each quad a variable so that the detail could be changed easily which is good practice instead of having "magic numbers" about, being able to change it easily would allow me to see the differences between the CPU and the GPU quickly in experimenting later.

Calculating the (U, V) coordinates

The texture coordinates needed to line up for several reasons, the main reason being if they did not line up with the neighboring patch than the island would be a mess of different parts of the texture.

The second reason being that the tessellation of the patches in the tessellation evaluation stage uses the heightmap to set the z-axis of the vertices that have been created, causing tearing between mismatched z values of the edges on a tessellated patch.

In order to correctly calculate the coordinates for each patch, I had to divide the (U, V) coordinates between the number of patches; this was a simple calculation (1.0/number of patches) as the (U, V) coordinates range from 0.0 to 1.0.

The heightmap could not repeat hence why the calculation never allows a patch to exceed 1.0, however, the textures that I wanted to use on the island would need to repeat, and allowing the (U, V) coordinate to exceed 1.0 allows the repeating of the textures. (Learnopengl.com, 2019)

When exceeding 1.0, depending on the OpenGL setting, this could either:

> Loop back to 0.0 in the case of GL_REPEAT, effectively ignoring the integer part of the (U, V) coordinate and is the default for OpenGL.
>
> Decrease from 1.0 down to 0.0 with GL_MIRRORED_REPEAT, in which the coordinates then increase back up to 1.0 and so on.
>
> GL_CLAMP_TO_EDGE uses the minimum or maximum values of the coordinates until the edge of the model once the texture would normally repeat. This results in a stretched appearance.
>
> GL_CLAMP_TO_BORDER has a user specified colour surrounding the texture up to the edge of the model.

Due to these options not allowing the repetition of the texture, they aren't appropriate to use for the island plane textures (grass, rock, snow, sand) to achieve the result I was looking for so I used a GL_REPEAT, with GL_MIRRORED_REPEAT the repeat was more obvious and looked slightly odd.

Figures 38 and 39 (below) show the difference between these options and the more obvious artifacts of the mirrored option, the same areas for each picture are circled in red for easy comparison.

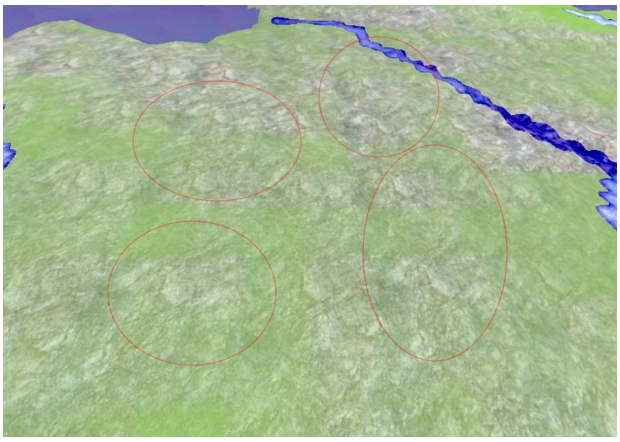


Figure 38: GL_REPEAT

Figure 39: GL_MIRRORED_REPEAT

Adding textures

To add the textures, I needed to bind the textures to their own position in the shader so that they could be used within the fragment shader.

The fragment shader is called only once per fragment so the order of checks and calculations is important in order to decide what colour should be used for that fragment.

This value can be added to or taken away from until the end of the shader where it is finally sent to be rasterized.

Below is the result of using a texture dependent on heights using the heightmap, where the red value from the heightmap was used as the world height variable at the start of the shader.

Figure 40: Sharp Texture Changes (Zoomed in)

Figure 41: one to one Rock and Grass Texture

Using a different texture dependent on the heightmap value causes this very sharp edge to the texture where they change as can be seen in Figure 40 (above left).

This did not look good and needed changing; also, due to the texture coordinates being mapped one to one on the simulation, the rock grass texture has a vast amount of different detailing on it and is stretched, this should not be visible from this distance, in order to look correct it should be less visible and more uniform in colour for the entire area.

To fix this, I multiplied the texture IN coordinate by a value, depending how small the texture needs to be, the higher this value was *Appendix 6 – Fragment Shader Code (Lines 46-50)*, this produced a better result but was still not quite the result I was going for.

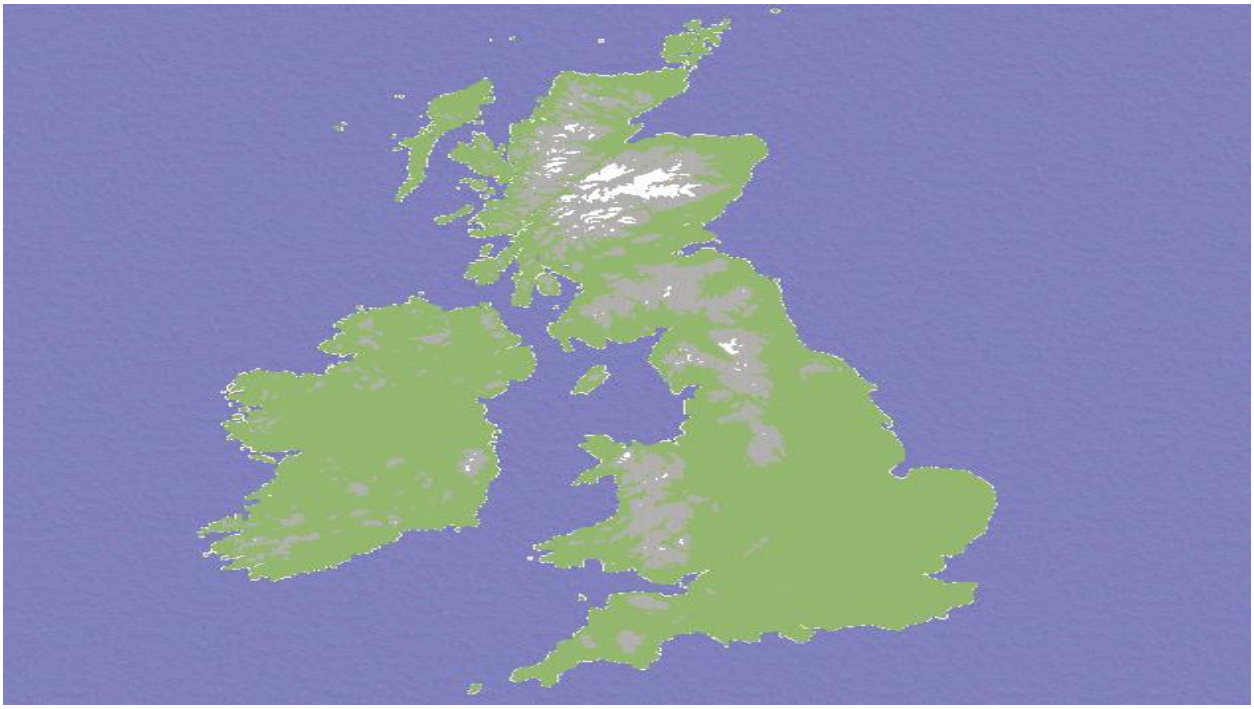


Figure 42: Multiplied Texture IN.Position Values (Result)

Figure 42 (above) is an improvement but the textures were still blocky, and the rock was obviously a repeated texture as can be seen in Figure 43 (below).

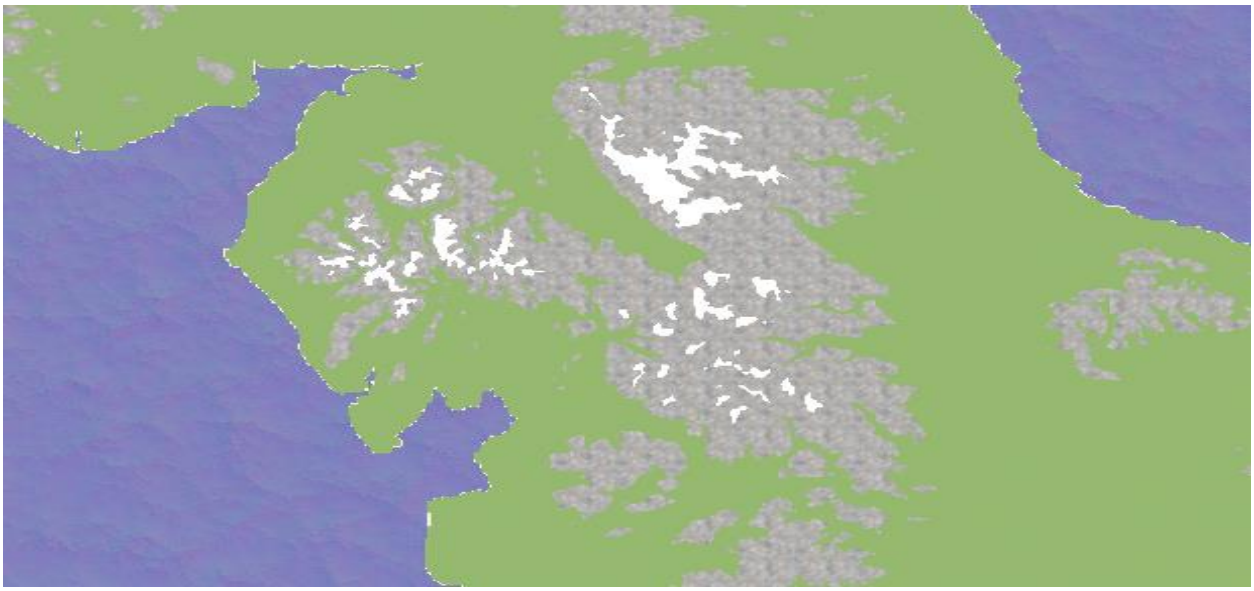


Figure 43: Repeated Textures Obvious

To fix this, and get the look that I was going for, I would need to blend the textures with one another. There was also a bug with the grass in this image, this is fixed, and the result shown in the next section in Figure 44 (below left).

Blending Textures

Blending the textures was a process achieved by mixing the colours together in the fragment shader.

I also at this point decided that the ground hardness map should dictate where the rock textures were being blended into the grass texture, this mostly lined up with the hilly areas of the height map so worked well.

As can be seen in Figure 44 (below left), the textures don't look right, they should not be this defined at this distance.

Figure 44: Grainy Textures

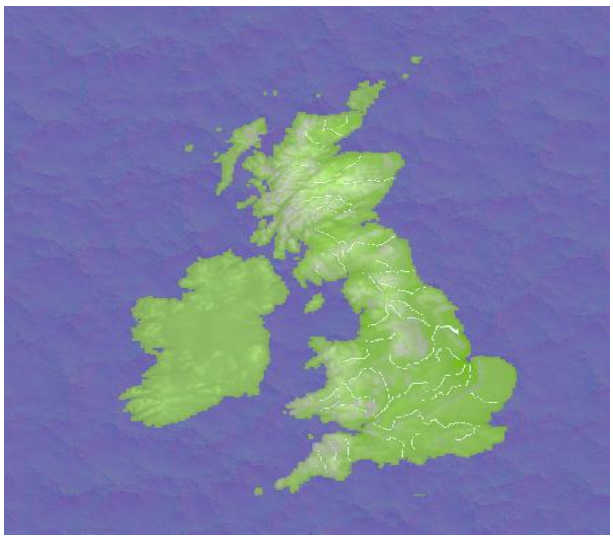


Figure 45: Mipmapping Added

This was fixed by using a technique called mipmapping, the premise behind mipmapping is a series of progressively smaller images, each less detailed than the previous. These are used dependent on how far from the camera the model is, creating a smoother look to the texture.

Adding mipmapping to both the grass texture and the rock textures gave the smoother result I was looking for in Figure 45 (above right).

Sea level and Sea texture

The sea level needed to rise and drop, the user can change this in real time by increasing and decreasing the level as the simulation is running, using the up-arrow key and down arrow key.

As the sea level is increasing, the separate sea entity rises above the lower elevations of the island model. As depth testing is enabled, this would correctly show on top of the island model and is the desired effect I was after.

A separate vertex shader for the sea was used as this set the z-axis value for the sea entity, this took in a uniform float, meaning it is accessible by every shader.

The z-axis value is the negative of sea level as in OpenGL the z-axis is lower the further away from the origin it is, and as the sea would effectively be rising past the origin towards the camera it needed to be negative whilst keeping the sea level variable positive to convert this value into metres.

It was important that textures used for the sea, could tile well, especially for the sea, being a flat plane with no mountains and lack of blending any tiling was obvious as is shown in Figure 46 (below), this was due to darker edges of the texture, so the texture was changed to a tile-able texture.

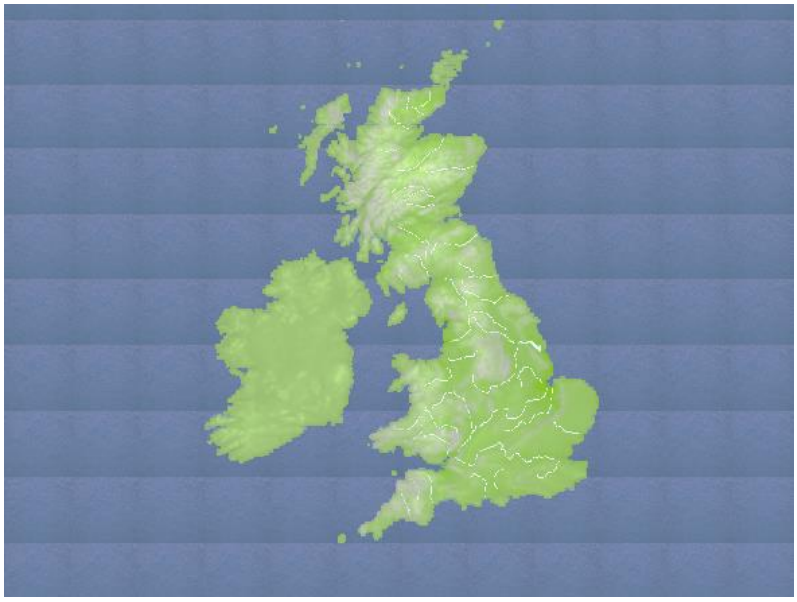


Figure 46: Non Tiling Sea Texture

The watertight tessellation mentioned earlier connects the middle sea section with the outer sea sections using two tessellation shaders, one for the outer sections and one for the inner, *Appendix 8 – Sea Tessellation Shader Code*, the outer version simply just divides the tessellation

inner levels by 2 but is otherwise unchanged, the result of this is shown in Figures 47,48,49 and 50 (below).

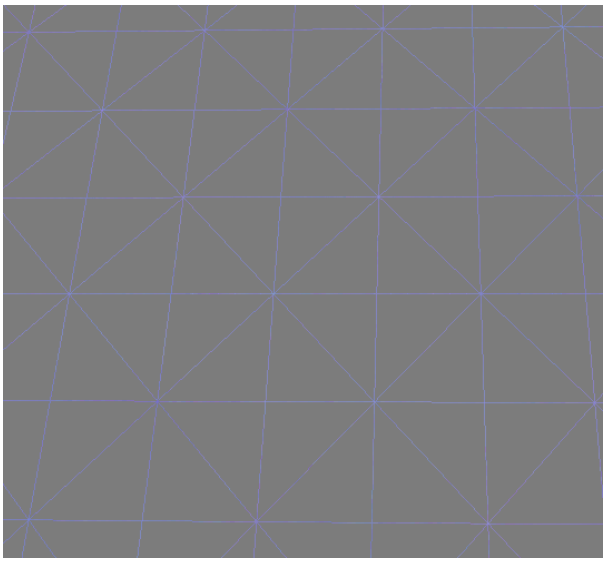


Figure 47: Watertight Tessellation (Level 1)

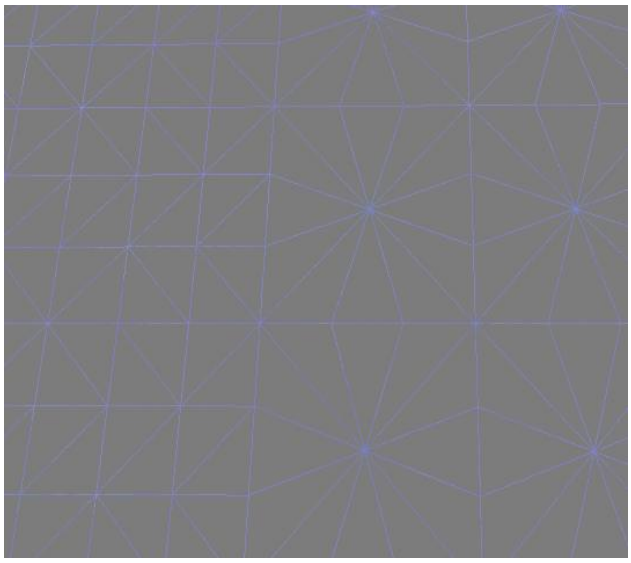


Figure 48: Watertight Tessellation (Level 2)

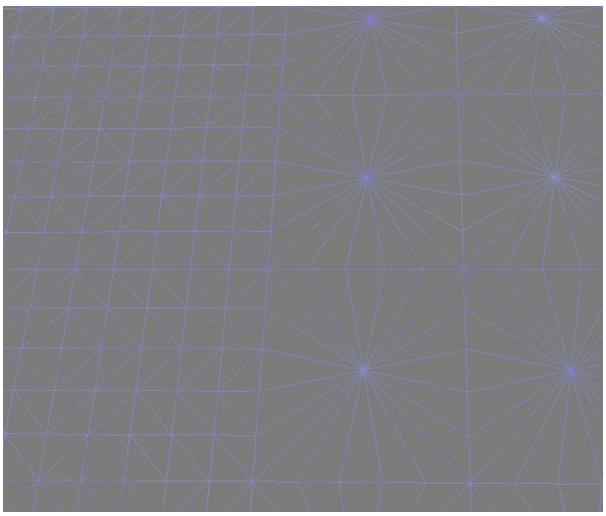


Figure 49: Watertight Tessellation (Level 4)

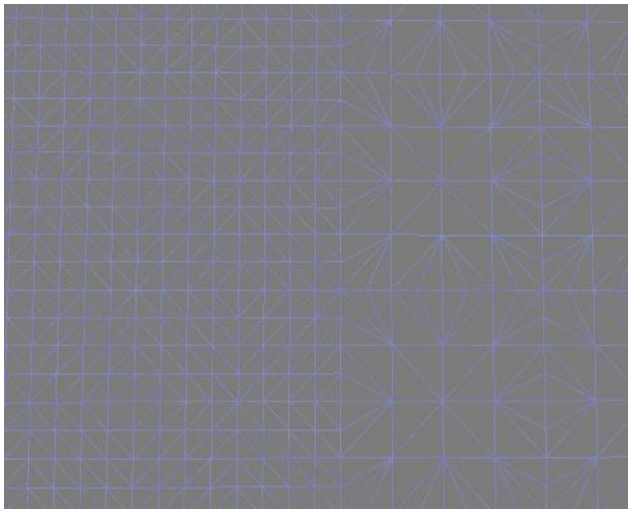


Figure 50: Watertight Tessellation (Level 8)

We can see that as the level of tessellation increases in the sea plane, triangles inside of each patch on the outer sea section changes.

Inside the patch triangles change shape and size in order to match up with the triangle edges, this is needed for both the adjacent outer sea section patches and also for the inner sea section patches as tessellation increases.

Erosion

To try and simulate basic erosion I used the distance from coast map I had created alongside the ground hardness and heightmaps.

To use these, I had them have a percentage that added up together to 100% and depending whether the sea level was above this added value would decide if the texture would turn into the sand texture.

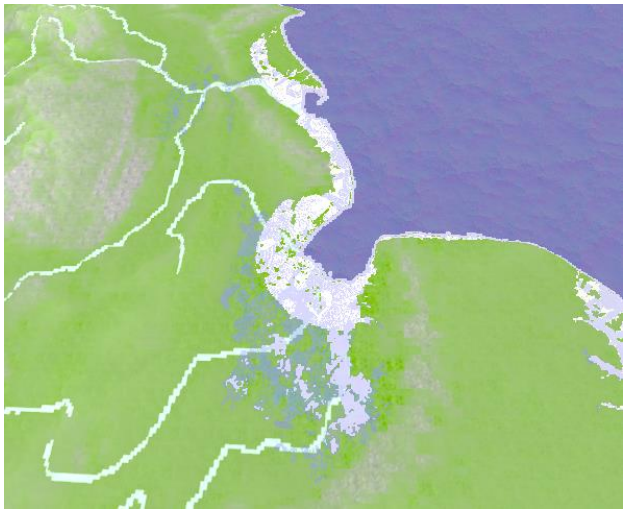


Figure 51: 1% Distance to Coast Effect

Figure 52: 5% Distance to Coast Effect

Figures 51 and 52 (above) use a 1% effect and 5% effect respectively for the distance from coast map, the change is subtle but you can see the 5% image is a little bit smoother and filled in than the 1% image, with a 0% effect the sand would be under all of the water that can be seen on these images.

Another aspect of the erosion is the hardness map, which can be seen here (circled in red), it does not erode the rock area but is inundated by water, this is done in the fragment shader and can be seen in *Appendix 7 – Updated Fragment Shader Code.*

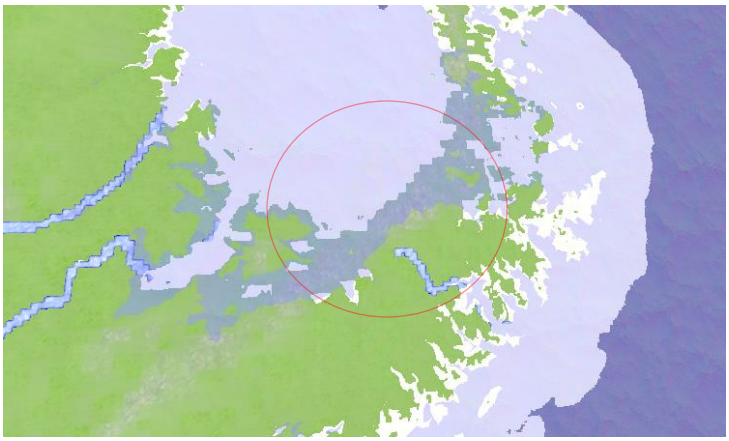


Figure 53: 15% Hardness Effect

Land Loss

Attempting to calculate the land loss and return this value to the CPU with the approach that I took using the shaders to do the calculations was not an easy process and as such I did not accomplish this objective.

The rendering pipeline is generally a one-way process, information goes in, the rendered image comes out and is displayed on screen, information isn't expected to be passed back to the CPU in this manner.

Population overlay

Adding the population map shows the areas of high habitation, it is blended into other textures.

I needed to reduce the green and blue colour of each fragment as the population was not very clear and blended into the island too much.

I decided to reduce them in line with how strong the population was, and the result worked out quite well.

The overlay can be toggled and doesn't seem to affect performance of the simulation whether it is on, or off.

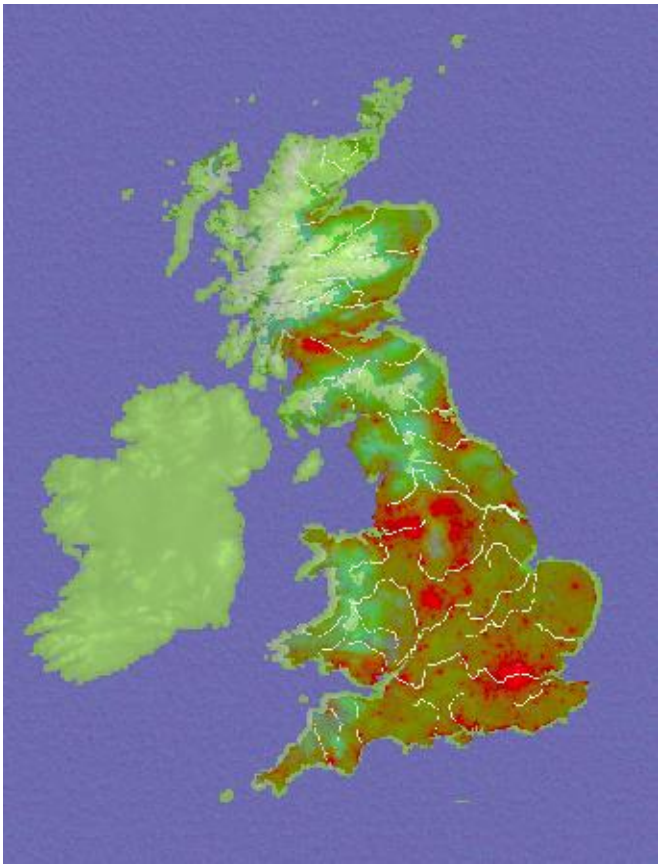


Figure 54: Population Map Overlay

Render Times and FPS

The render time is recorded using an adapted timer (Ramónster, 2019), this timer starts before any setup code runs and ends just before the render while loop, this captures the time it takes from starting the program to rendering an image in the window.

Another timer then starts inside the loop to capture each frame render time, this render time is in milliseconds (ms) and is converted to frames per second (FPS) by simply doing 1000/render time (ms).

These values are rendered on screen for the user to see, this is displayed every second using a variable called delta time, each time the loop runs, the new render time is added to the delta time.

When delta time exceeds 1 second the current render time is saved and displayed to the user, without this, the times would be unreadable, as it would be changing too fast.

What the user is actually seeing is just a snapshot when delta time exceeds the 1 second time, after this is saved and displayed, delta time is reduced by 1 second exactly, not reset to 0.0, as leaving the left-over time will giving a more accurate timing.

Recording the data

For the simple experiments, the frame render time was recorded, simply outputting the times to a text file using an output file stream and overwriting the file each time.

In order to compare the data for the more advanced experiments, I needed to adapt the output file to resemble a CSV file to use within Excel.

All relevant variables are now recorded for analysis later and can be recorded without having to restart the program; this is explained more in the Chapter 6: Results.

Final Tweaks

One of the final tweaks that I added to improve the look of the simulation, was to add mipmapping to the heightmap itself, I also considered bilinear filtering.

Both techniques reduced the steep edges of the mountainous areas and made the coastal edges a little bit smoother. This can be seen in Figures 55, 56 and 57 (below).

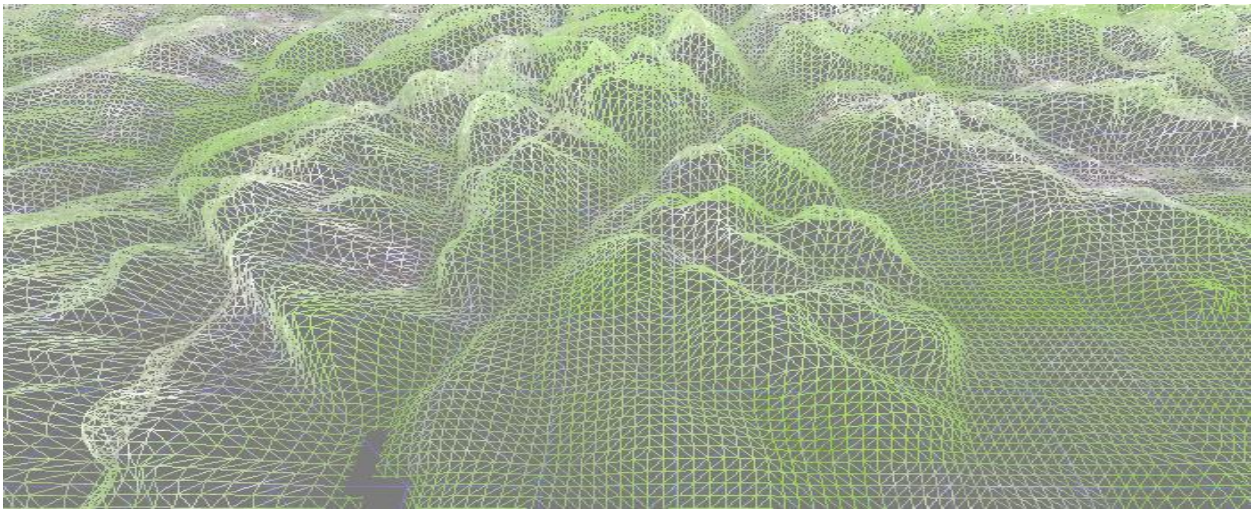


Figure 55: Mipmapping Enabled

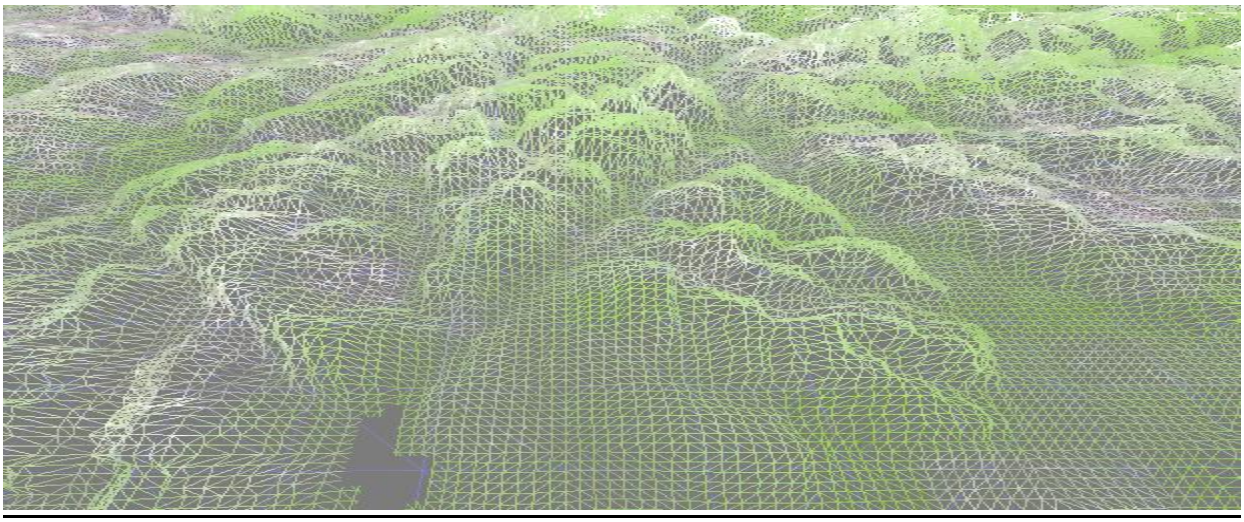


Figure 56: No Mipmapping or Bilinear Filtering.

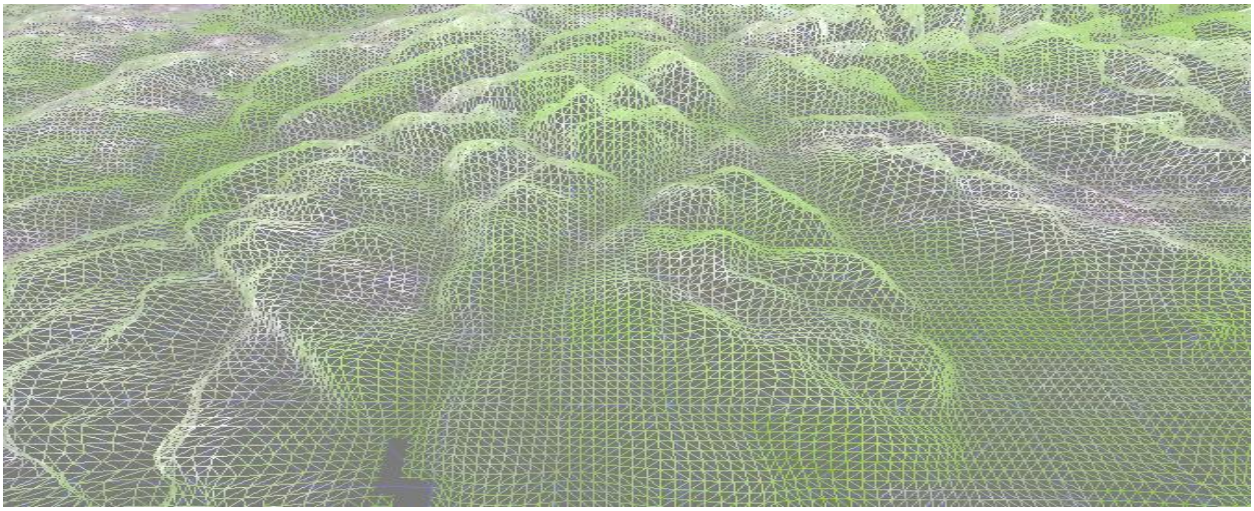


Figure 57: Bilinear Filtering Enabled

Chapter 5: Testing and Experimenting

5.1 Testing the Simulation

I needed to test the program so I could be sure the data I was comparing was as accurate as possible. By fully testing each aspect of the simulation, I could demo the program reliably without needing constant supervision, in case the program crashed for example.

Test Plan

I created a Test Plan *Appendix 9 – Test Plan* to keep track of what I had tested, and to ensure that everything that needed to be covered, was, and worked correctly.

The test plan is very simple, but covered the major aspects of the simulation, I was usually testing whilst I was coding, as graphics programming can be very unforgiving.

Code and Fix

Code and Fix is normally a frowned upon approach of programming which has little to no planning or design involved, this was not entirely true for me as I had planned the architecture of the simulation and knew what I wanted the result to look like.

This however didn't stop me coming across a design flaw when it came down to measuring the land loss, as mentioned previously.

Being new to graphics programming, I found this was the easiest way for me to learn mistakes that I was making, as I was making them.

I feel that I have developed a reliable simulation that is unlikely to crash randomly or produce any unexpected graphics issues during demonstration or general use.

5.2 Experimenting

To achieve the data that I wanted to compare and visualize, I used a variety of experiments.

As mentioned in Chapter 4: Implementation, I recorded the time in milliseconds that it takes to render each frame (ms/frame), written to a file every second. I then transferred this data into an Excel spreadsheet, removing any obvious outliers, before finally calculating an average, to ensure that the average was as accurate as possible.

To achieve a stable frame rate and keep the results consistent, I used a full-size desktop with adequate specifications and most importantly could keep the temperatures stable to prevent CPU or GPU throttling that is commonly experienced on laptops, as they cannot dissipate the heat as efficiently as a desktop.

The important specifications for the system used in testing were as follows:

CPU: Intel i5 – 2500K (Sandy Bridge) – Not Overclocked (OpenGL 3.1)

GPU: NVidia GTX 560-Ti (448 core) – Not Overclocked (OpenGL 4.6) – V-Sync disabled

Ram: Corsair Vengeance 16GB DDR3 1600MHz

Monitor: 1080p 28inch Acer 60hz

Operating System: Windows 7 Ultimate

Storage: 120GB SATA SSD

I also monitored CPU temperatures and GPU temperatures using Open Hardware Monitor (Möller, 2019)[21] to see the workload of the GPU, ensure neither the CPU and GPU were getting too hot and ensure it wasn't affecting the performance; thus the accuracy of the results.

The data this monitoring can provide could be used in itself to check performance of the simulation but as there were other programs running in the background, using Excel, using Notepad++ to record the times as I was experimenting, would not be accurate and as such only used it to ensure no throttling was occurring.

Disabling V-Sync (Vertical-Sync) on the graphics card was very important for testing as V-Sync limits the frame rate to 60FPS, and can go no higher, this resulted in a minimum frame render time of 16ms, which was not appropriate for my simulation which as initial tests show, can run at approximately 0.4ms frame render time. This is a big difference from 16ms and would not allow me to see any changes at these low times if it was limited.

5.2.1 Planning the Experiments

Simple Experiments

I used tables of variables that could change the detail and performance of the simulation; these tables *Appendix 10 – Experiment Tables* provided a structure to the experiments, covering the simple bases and allowed me to keep track of the results easily by numbering the tests.

Advanced Experiments

I then decided it would be best after doing the simple experiments to change the information that was being recorded in a comma separated CSV style file to encompass more data, the extra data on top of the time taken to render the frame now included:

- X, Y, Z - axis positions
- Pitch and Yaw
- Sea level
- Tessellation levels
- Wireframe mode (On/Off)
- Grid Size
- Triangle Size

This data was recorded per second like before, would allow me to use the simulation as normal, changing variables as the simulation ran where possible or changing the setup values such as grid size or triangle size, which could not be changed during the running of the program and would require a restart.

Using Excel, I created graphs to see if there were any correlations in the data, using this big data approach sped up the experimenting process and would show any correlation between the time taken to render the frame and any of the variables that had changed.

5.2.2 Displaying the Results

In order to easily read the information gained from the results of these experiments I have displayed most of the results as line graphs, radars or other visual formats, these results are discussed in Results and Evaluation (Chapter 6).

Displaying the data in a visual format enables me to see if there are any correlations between the variables and the time taken to render the frame.

If there was an apparent result but some scenarios were not tested and were missing data in the graph, I could then manipulate the use of the simulation, appending to the file to check if this was indeed the case by testing the theory out further on only the missing data.

An example of this could be:

Apparent results:

Having a lower Z-axis (zoomed out) causes time taken to render frame to increase more than tessellation did.

Missing Data:

High tessellation values at high Z-axis values (zoomed in).

Course of action:

Run simulation and increase tessellation values whilst zoomed in to the simulation.

Actual results:

The Graph now shows that it was in fact the tessellation values affecting the time taken to render the frame more than the Z-axis values.

During testing I noticed that a function in the codebase that allowed me to display text on screen, was causing a big performance decrease and affecting the render time.

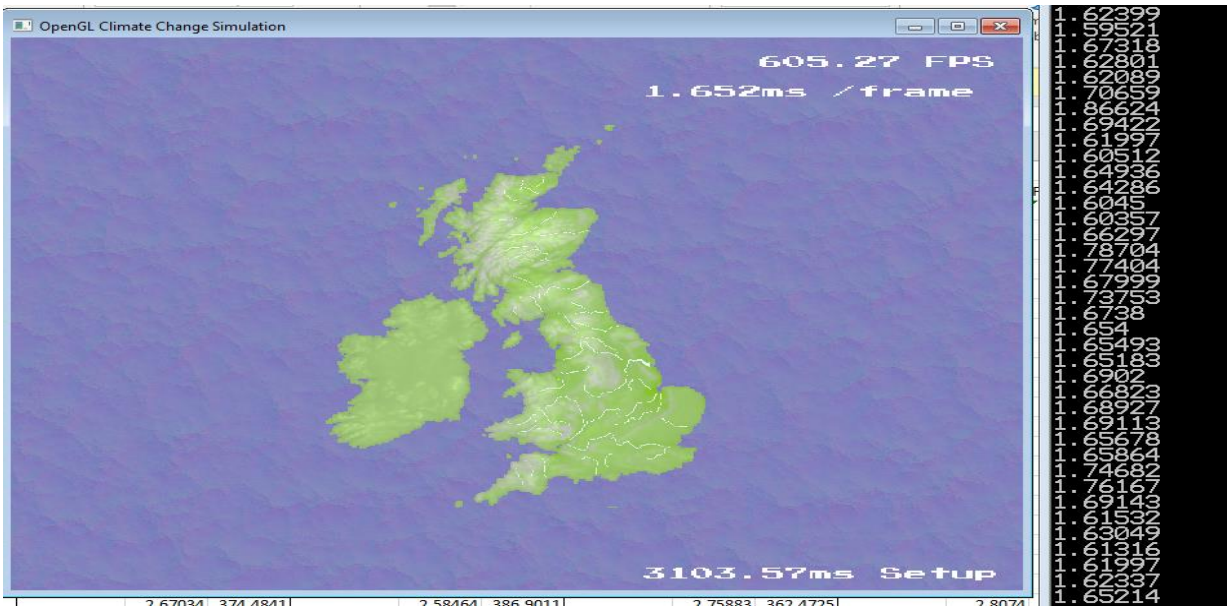


Figure 58: Rendering with Text

The render time was not a uniform increase, which would have been acceptable for use while doing my experiments. Instead rather, it created a minimum render time depending on the amount of text that was being used, at approximately 0.4ms per line of text.

Once exceeding this minimum render time, depending on the amount of lines used, did not increase the render time, causing results of the experiments to be very inaccurate.

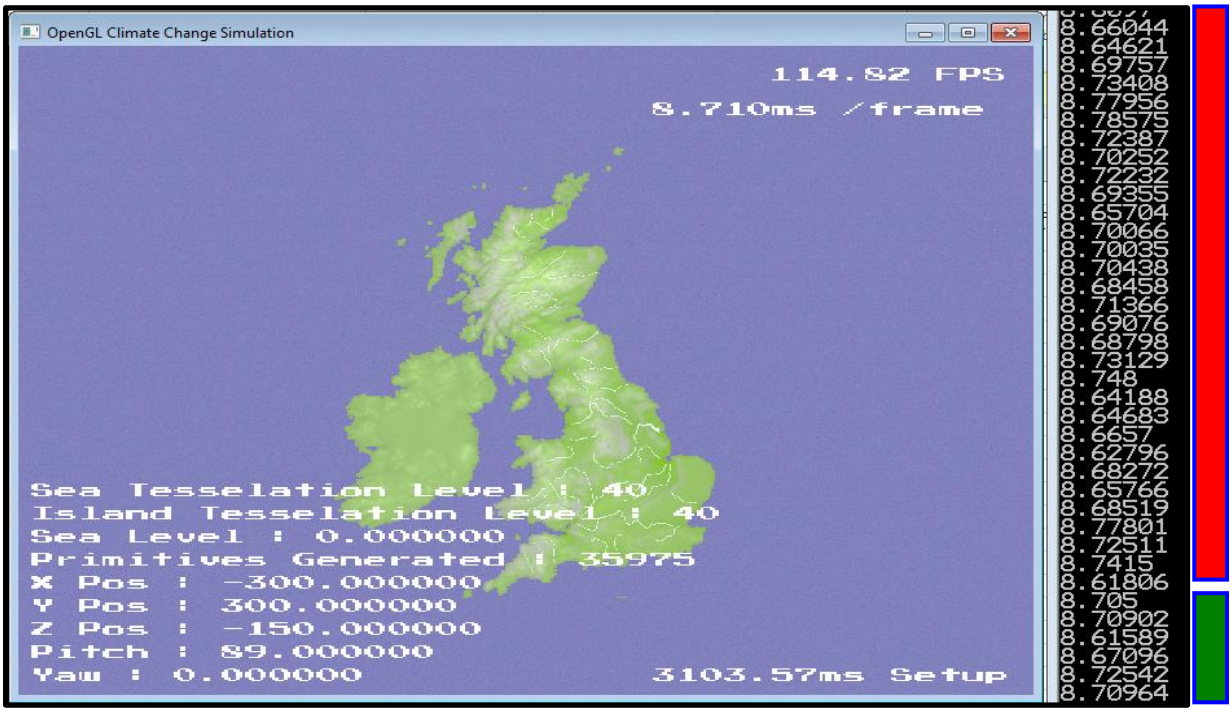


Figure 59: Render Time Not Affected

The picture above does not show it very clearly, but the debug text information causes the render time to be approximately 8.7ms, the green section of the render times is with the extra text showing. The red area is when this extra text is not showing and as can be seen the time doesn't increase.

I decided to comment out all text being displayed to the screen whilst experimenting, however, the text will be displayed during the demonstration, as the FPS is useful when not analyzing the data.

After this was removed the minimum time taken to render a frame reduced from 1.6ms to 0.36ms as shown in Figure 60 (below).

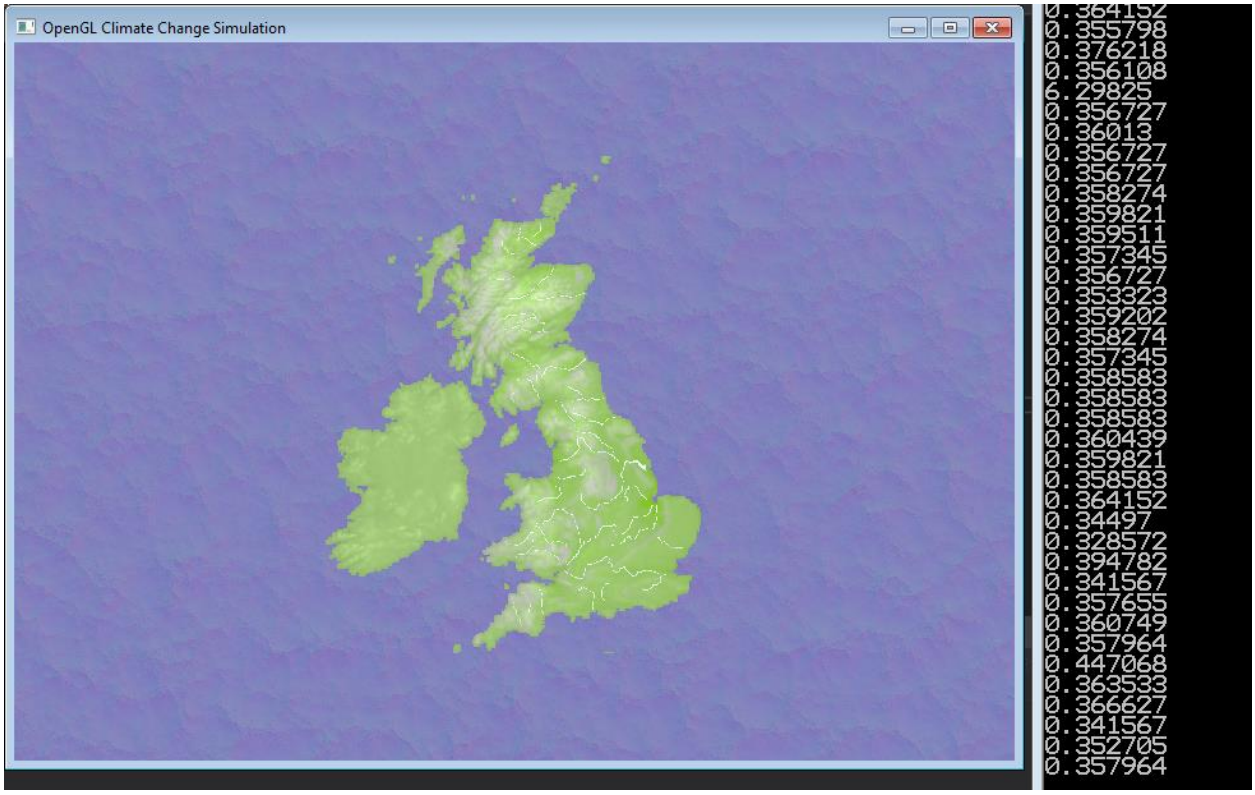


Figure 60: Rendering without Text

Chapter 6: Results

This chapter shows, in detail, a comparison of the variables in the simulation, how changing them affects the performance of the simulation and shows the results visually in graphical formats most appropriate to the experiments.

6.1 Simple Experiments

Below is the idle graph showing temperatures and memory use before these experiments, this is the baseline in which the computer has the programs needed open ready to use, these programs were limited to Excel, Notepad++, Visual Studio 17, Open Hardware Monitoring and Snipping tool. Minimized where appropriate with exception to Visual Studio 17 as this was what I was using to time the experiments and ensure each had at least 1 minute of run time after the setup time.

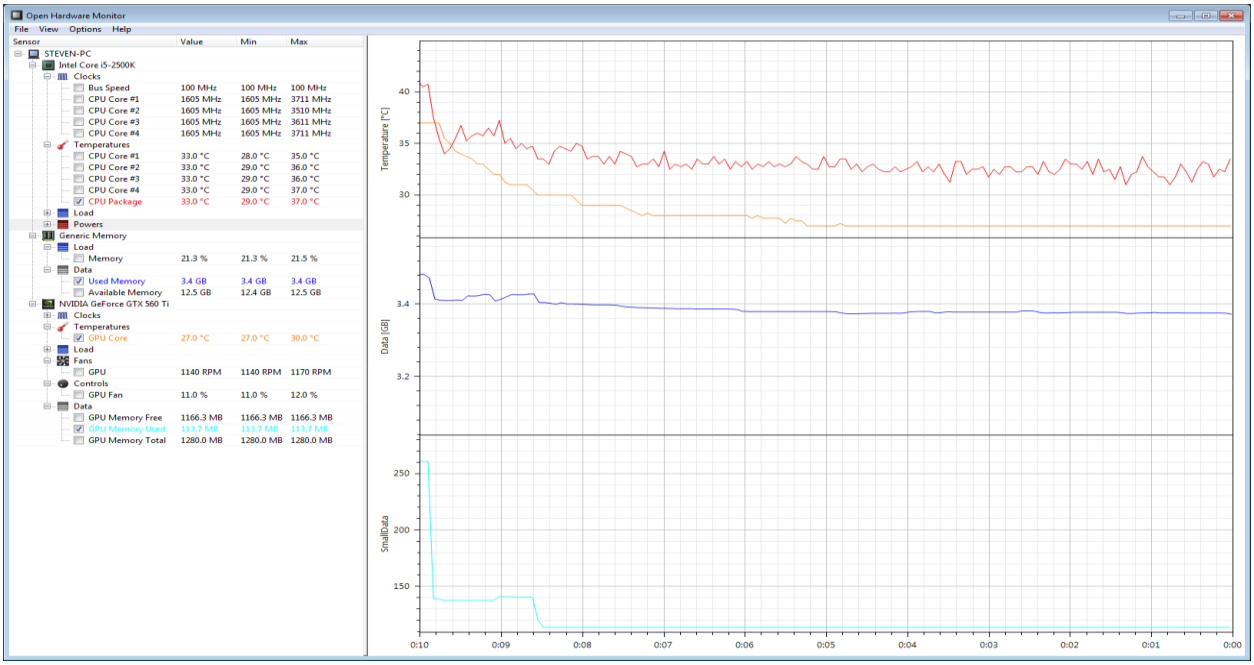


Figure 61: Idle Temperature and Memory Use Graph

The graph moves from right to left and in this case is a 10-minute window, I kept track of CPU and GPU temperatures, Memory and GPU memory, (from top to bottom respectively).

Default Variables used in the simple experiments below can be found in *Appendix 10 – Experiment Tables.*

6.1.1 Tessellation

The graphs below show the result of increasing the tessellation of both the sea and island planes, the other variable's default values are used and remain the same throughout the experiment.

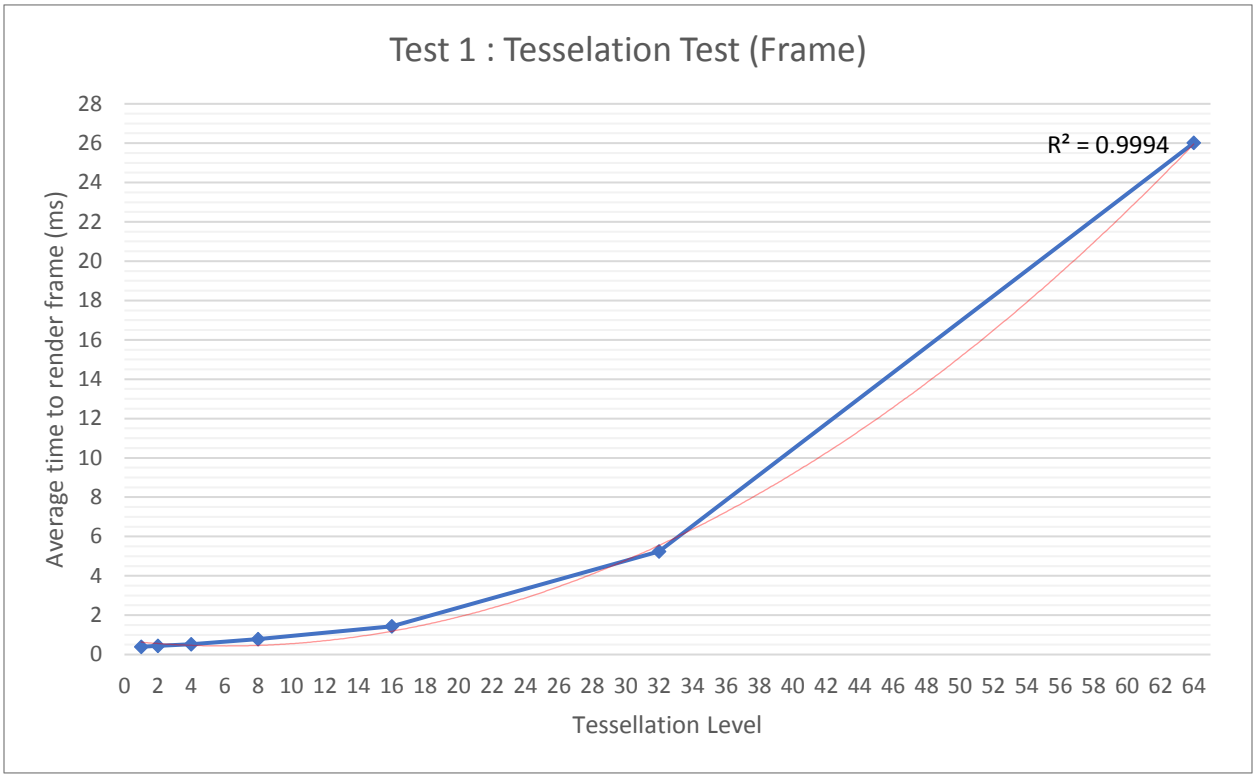


Figure 62: Tessellation Experiment Graph (Frame Time)

This graph and other tests with graphs similar were produced using the data recorded per second and recorded in an excel spreadsheet shown in *Appendix 11 – Experiments Results Example (non-Graphical)*, this is only an example for a few data points of the graph as the spreadsheet is much larger.

As can be seen on the example results, the list of times is ordered by lowest to highest time, using an excel function to work out the 1st quartile and 3rd quartile, then using these results, the interquartile range (3rd – 1st).

From this I have an upper and lower bound, with which, the results furthest away from either the upper or lower bound are worked out and then manually removed, the spreadsheet then works out which to remove next, based on the new interquartile range.

This is not recalculated in some statistical applications, however in this instance, there is reason to remove them, as the results should be a very small standard deviation as no other activity should affect the frame rate.

The setup graph shows the time it took to setup each experiment; this time is mostly influenced by grid size and triangle size, and therefore did not change, as these variables remained default.

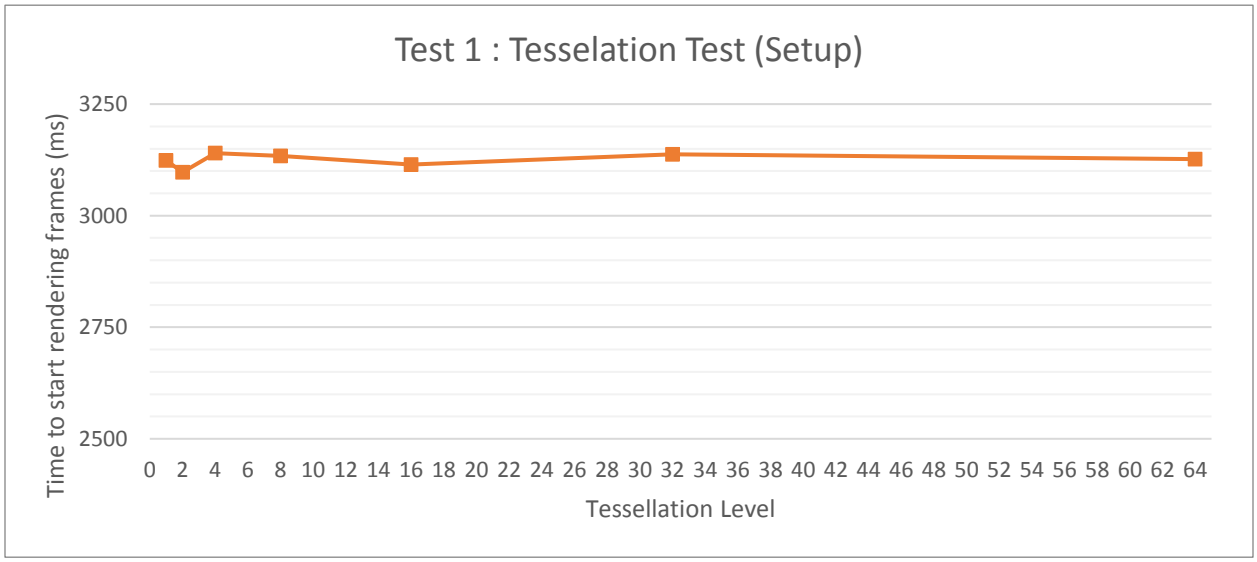


Figure 63: Tessellation Experiment Graph (Setup)

The results of tessellation were as expected, an increase of time taken to render the frame as tessellation values increased. What was not expected was the extreme difference between the tessellation levels when exceeding 16.

This seems to follow a $O(N^2)$ "big o" curve as shown by the trendline (red) in the graph, big o is the largest order in an equation, and as such is very approximate, but shows us what we could expect to see if we were able to use 128 tessellation levels in the future (current maximum 64).

R^2 is how well the trendline fits the graph, where 1.0 is the desired result of R^2, and as can be seen in the graph, it is very close to this.

Figure 64 (below) shows the change in temperatures and memory used whilst running the experiments, on most occasions I did not wait long enough for the GPU to fully cool back to its original temperatures, but at these temperatures no throttling would occur and the experiment was not how well the graphics card performed at staying cool.

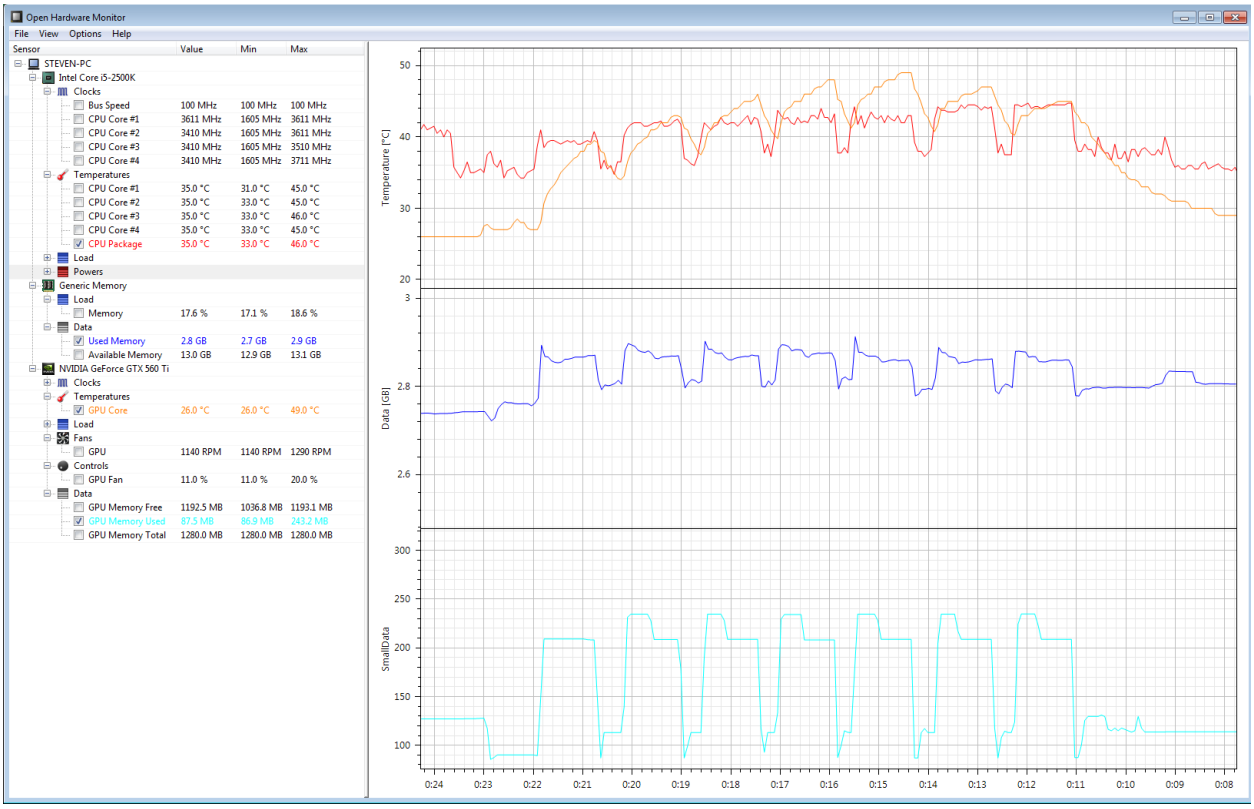


Figure 64: Temperature and Memory Use (Tessellation)

As can be seen, Memory requirements did not change depending on the tessellation, with exception to GPU memory; where a tessellation value of 1 did not require that extra starting bit of memory that all the other tessellation values required before dropping back to almost the same, temperatures were gradually increasing but I feel this was more due to not letting it fully cool to idle temperatures, before the next experiment.

6.1.2 Triangle Size

The graphs below show the result of changing the triangle size in the grid.

The way the grid works is by having a set size, e.g. 200.0, this isn't 200 triangles or even 200 quads, the amount of triangles is dependent on the size of the triangles, so for this example, if the triangle size was 5, the grid would have 40 quads wide, and 40 quads high, which would be 1600 quads and totals 3200 triangles.

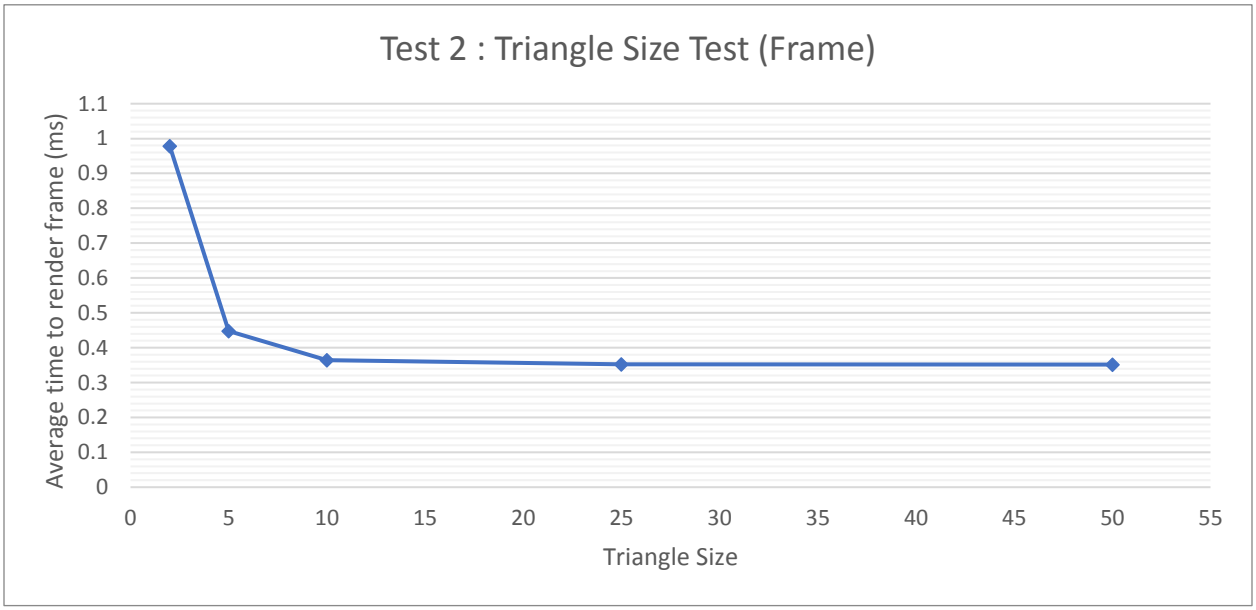


Figure 65: Triangle Size Experiment Graph (Frame)

There is a big difference between a triangle size of 2 and 5, and will need further testing, I will also need to test for much smaller triangles (in the decimals).

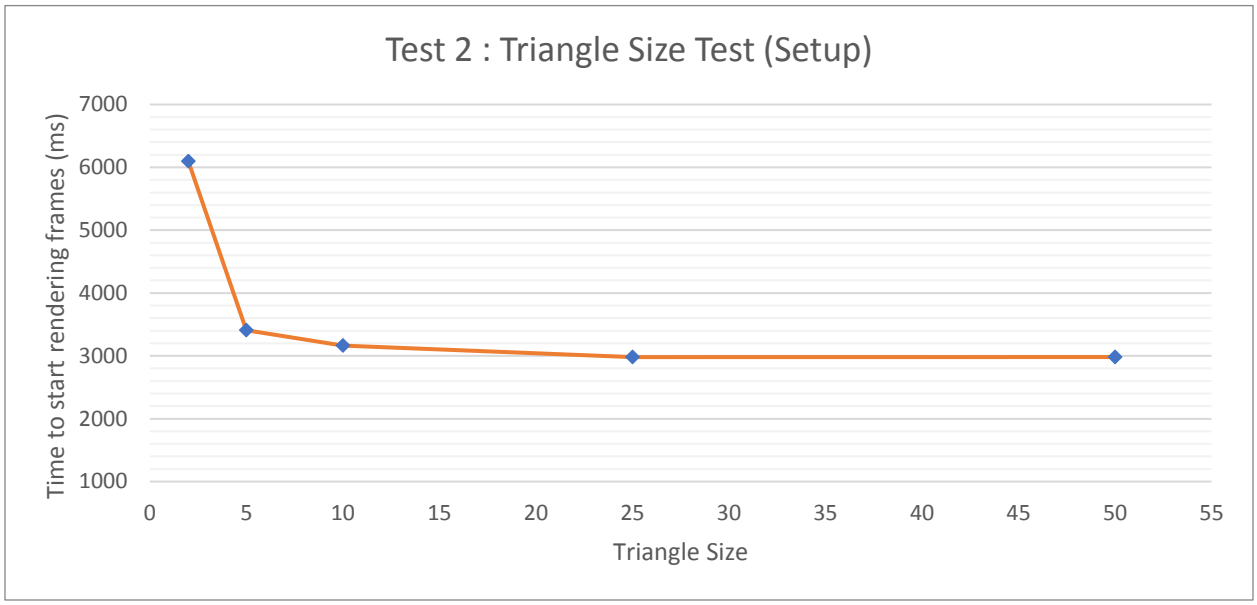


Figure 66: Triangle Size Experiment Graph (Setup)

The frame rate and setup graphs look very similar which I was not expecting, my expectation was that the frame rate would be less affected by changing the triangle size compared to the setup time that it took.

But thinking about it more clearly, the smaller the triangle is, and thus the more triangles there are for the grid size, meaning that the GPU has a greater number of vertices that it must deal with.

The graphs also show that exceeding a triangle size of 5 does not increase performance much at all, at least not with a grid size of 200.0, the default for this test.

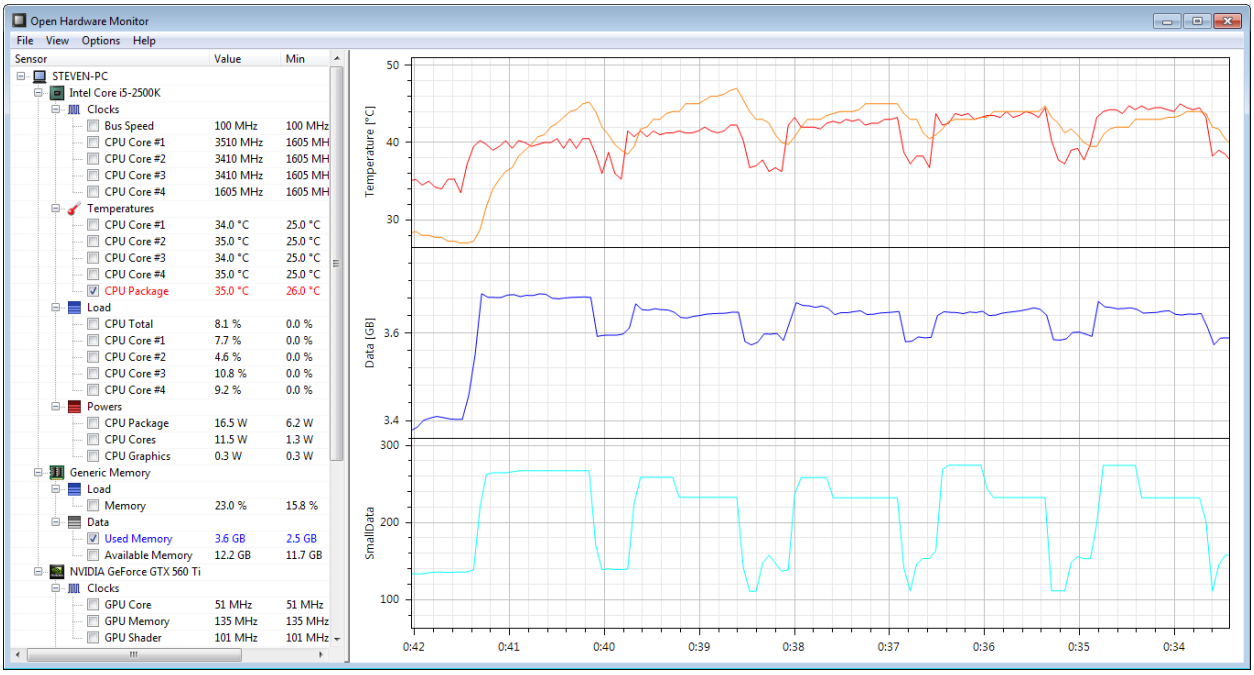


Figure 67: Temperature and Memory Use (Triangle Size)

As the triangle size increased the workload for the GPU dropped and so the temperature for the GPU dropped as can be seen in the graph, the CPU temperature gradually increased but as with the previous experiment I believe this was due to not letting the CPU cool to idle temperatures before I started the next experiment.

The GPU memory was higher and remained higher in the case of the smallest triangle size tested of 2, this is surprising as I thought the GPU memory may drop as it does with the other experiment, this could be an anomaly as I am not sure why this would occur, the required memory was slightly higher when the triangle size was smaller, as expected, as there was more data that needed to be held to be sent to the GPU.

6.1.3 Grid Size

As mentioned above in the triangle test, the grid size is a set size, this test will change that set size to test its effect on performance.

The graphs below show the result of changing this grid size with a default triangle size of 10.

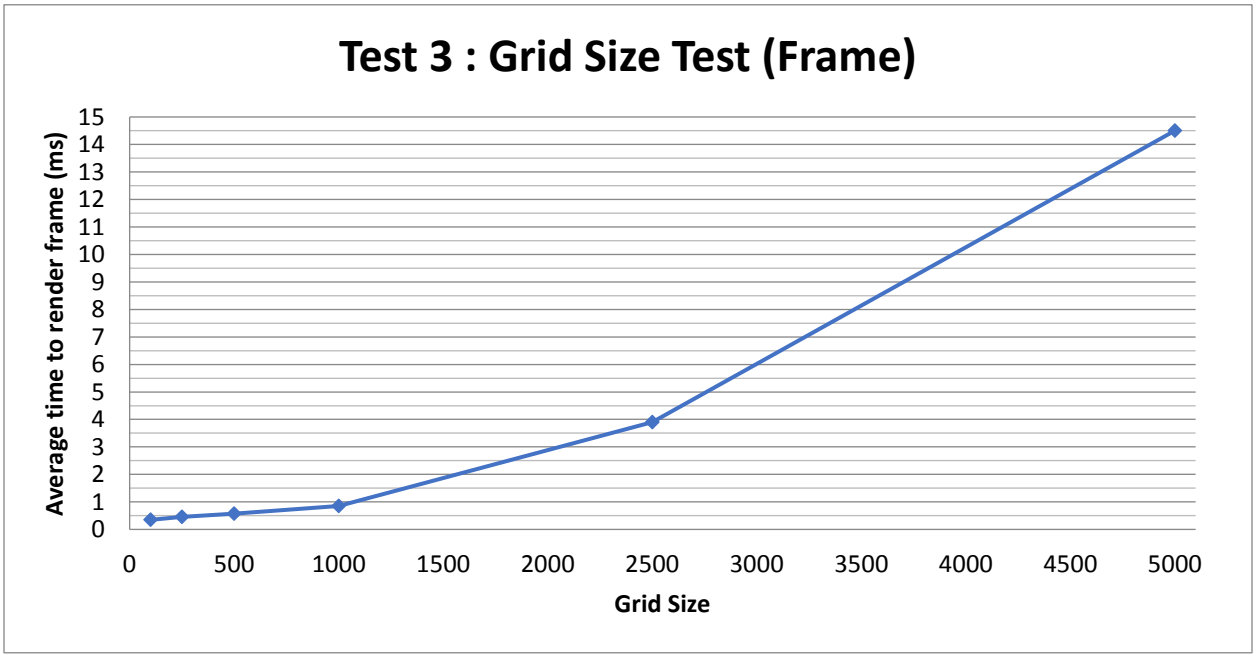


Figure 68: Grid Size Experiment Graph (Frame)

Grid size affected the render time a lot, despite the culling (removing) of triangles that are not in the visible perspective. The graph is similar to the reducing the size of the triangles graph.

A grid size above 1000 starts to affect the performance greatly as it does with the setup time in Figure 69 (below).

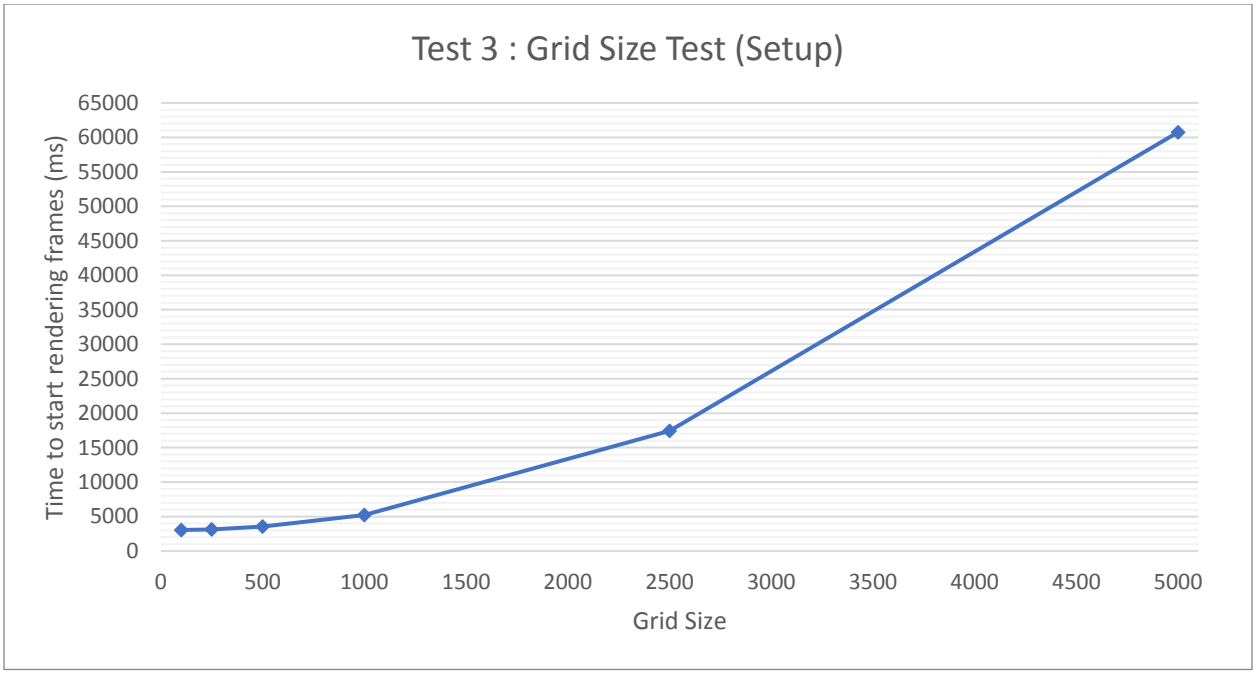


Figure 69: Grid Size Experiment Graph (Setup)

Once the grid size reaches 5000, there is a 60 second wait, which to load a complicated game doesn't sound unreasonable, but in the case of this relatively simple simulation, is a very long time.

Again I was not really expecting this result of the graphs looking very similar and was expecting that the Setup time would be more affected than the frame rate as the same amount of triangles are actually on screen whereas all the triangles or as many as possible would need to be loaded into the GPU's RAM.

A grid size of 5000 with a triangle size of 10 would result in, a 500x500 grid of quads, or 500,000 triangles. Comparatively in terms of number of triangles, that were of size 2 would be the same using a grid size of 1000.

The graphs below show comparative values where the number of triangles remains the same.

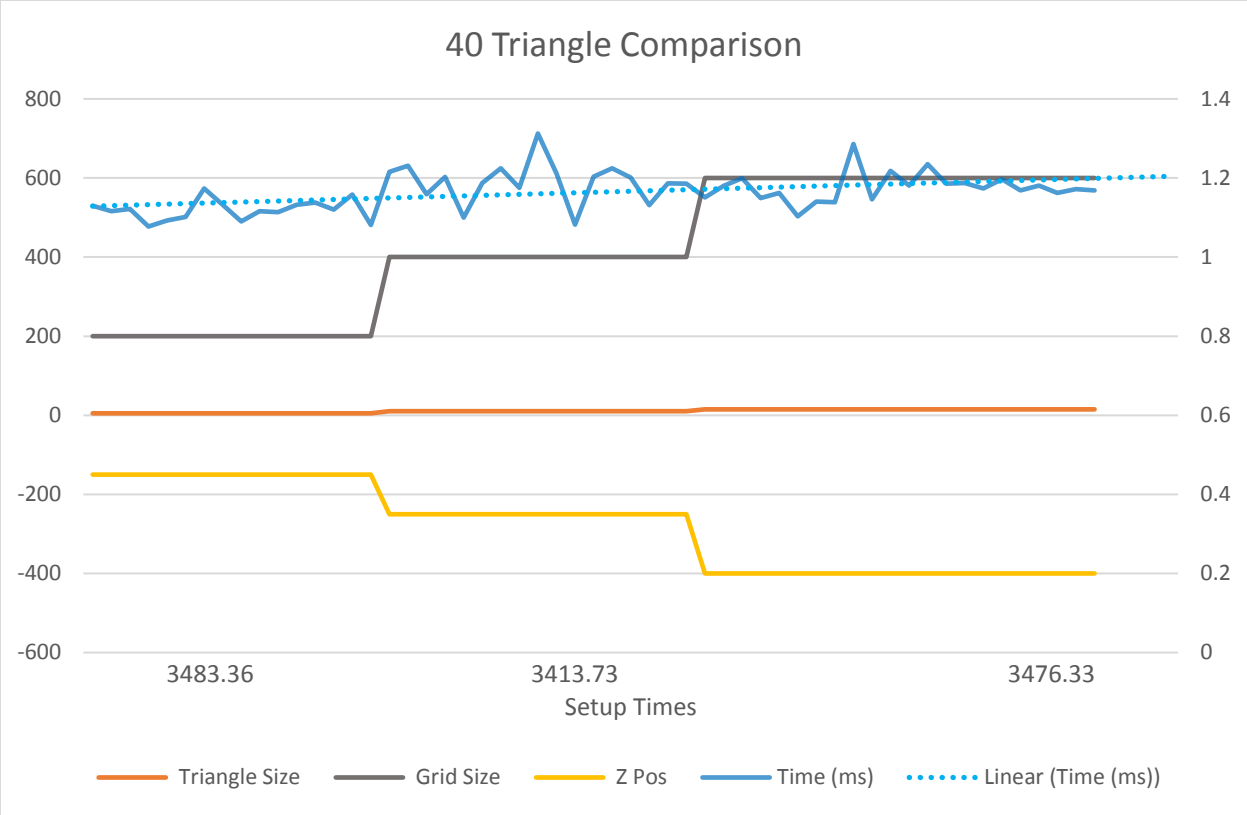


Figure 70: 40 Triangle Comparison Graph

The Z position was adjusted to compensate for the size of the model so that the entire island was visible.

This graph shows a slight increase when using a larger triangle and grid size, this could be because there are more pixels to account for when rasterizing (filling in the triangle) despite the total amount of pixels on the screen being the same.

As the number of samples used was very little (less than 100) we cannot draw a conclusive result from this graph, and it could just be a coincidence that the graph tends upwards.

6.2 Advanced Experiments

The following experiment uses data from normal use of the simulation.

Comparing Tessellation

The graph shown in Figure 71 (below) is ordered (ascending) by time taken to render the frame, using only non-wireframe results, it is a stacked graph (Sea Tessellation (Grey) is added on top of the Island Tessellation (Orange)).

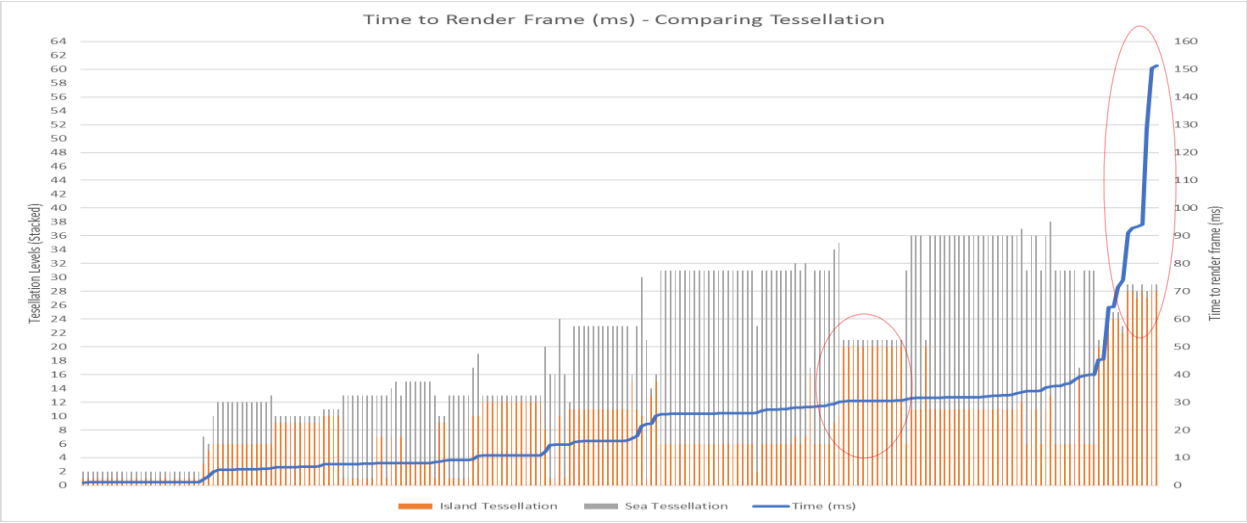


Figure 71: Tessellation Comparison Graph (Non-Wireframe)

Breaking the graph down we can see the following points:

- Lowest time (ms) taken to render the scene is when both Tessellation levels are the lowest they can be; at a level of 1.
- Tessellation levels of the Sea Plane don't affect the time taken to render the frame as much as the Island Plane does despite having nine times the area of the Island Plane, possibly due to the island mod value which increases the triangles for the island only.
- There is an anomaly in the data where Island Tessellation is at 20 and Sea Tessellation is at 1, Render time is approximately 30ms.

Likewise, in this graph also sorted (ascending) by time taken to render a frame, comparing only the wireframe results, has a spike just prior to the spike in which the sea tessellation is much higher than the island tessellation.

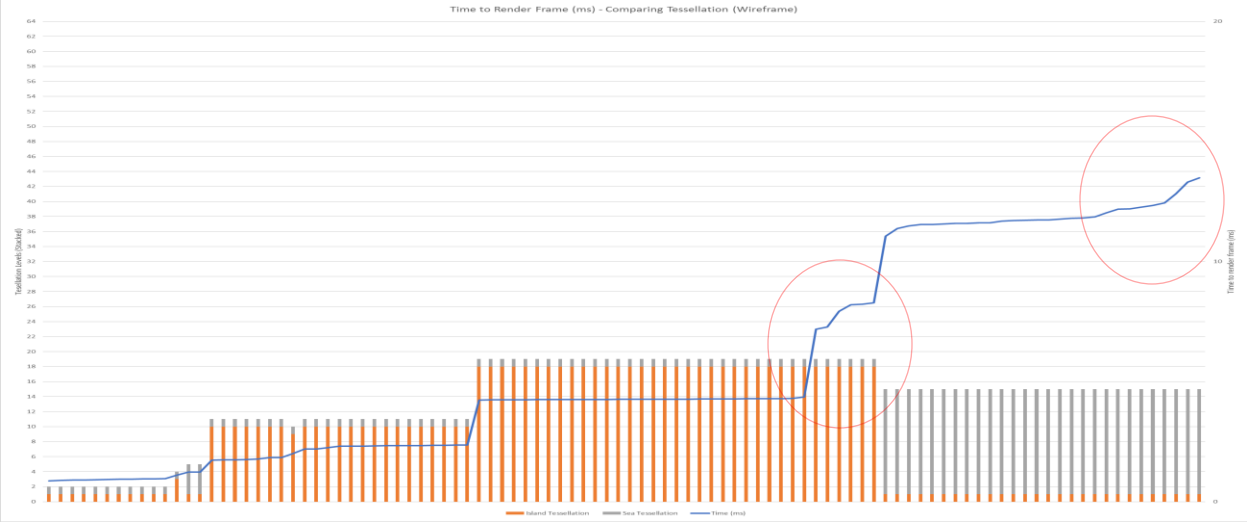


Figure 72: Tessellation Comparison (Wireframe)

My take on this graph is that the wireframe is a lot of detail to render, due to lines that must be rasterized on screen, this is most in effect when tessellation is high on the larger Sea Plane.

All these results suggest another factor is contributing to the spikes in time where there is no obvious cause as in the case of Figure 72 (above) where the spikes are circled in red, I will need to factor in more variables in to the graph in order to check what is the cause of these spikes, I believe it may be due to zoom level, discussed later in this section.

Below is a set of images showing some different tessellation values.

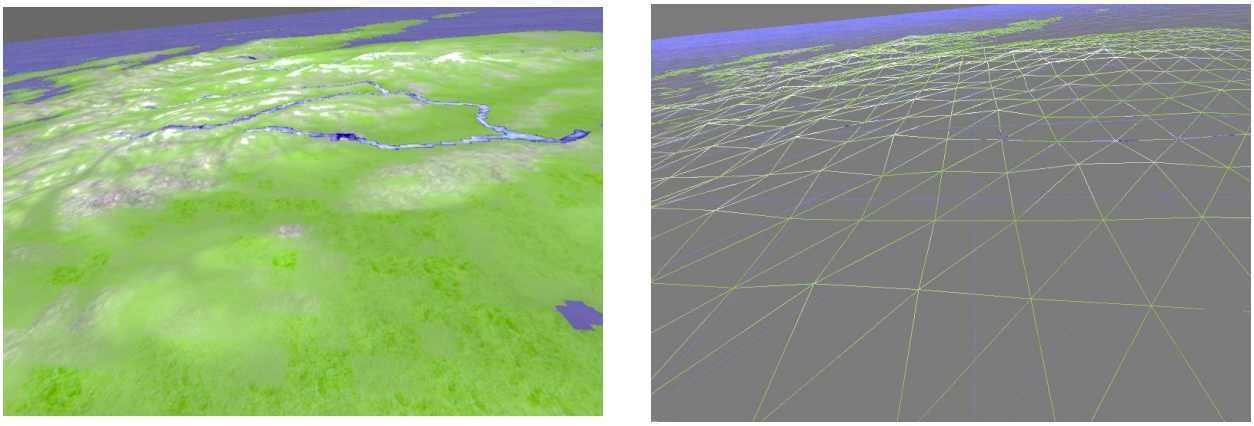


Figure 73: Tessellation Level 1 (Filled / Wireframe)

At a tessellation level of 1 there is slight height variation, this is due to the tessellation evaluation shader being called and changing the vertex heights despite adding no tessellated triangles to the patches, the only variable changing in these images is the tessellation level.

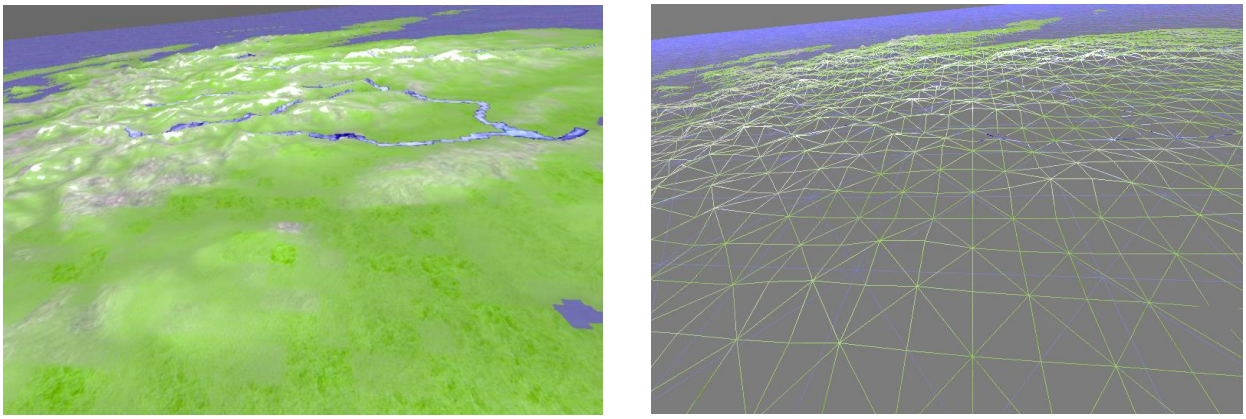


Figure 74: Tessellation Level 2 (Filled/Wireframe)

Already a bit more height variation being added for very little performance cost.

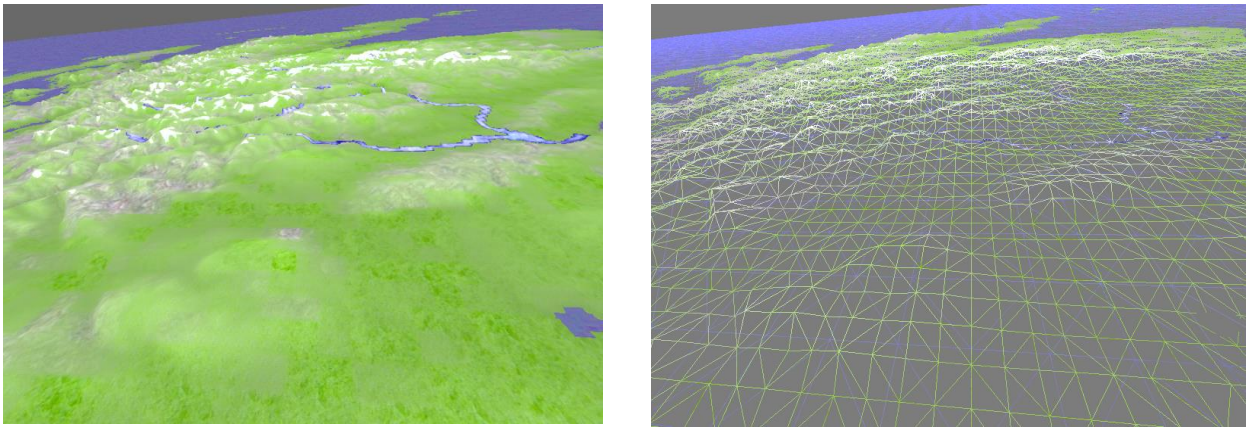


Figure 75: Tessellation Level 4 (Filled/Wireframe)

A lot more height variation is added and a few more triangles.

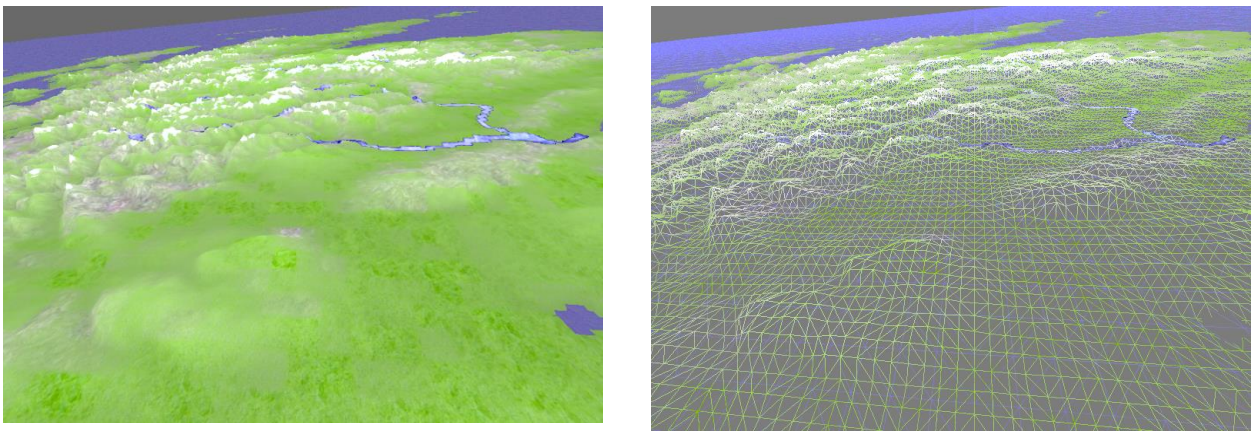


Figure 76: Tessellation Level 8 (Filled/Wireframe)

Much more detailed ridges and a lot more triangles.

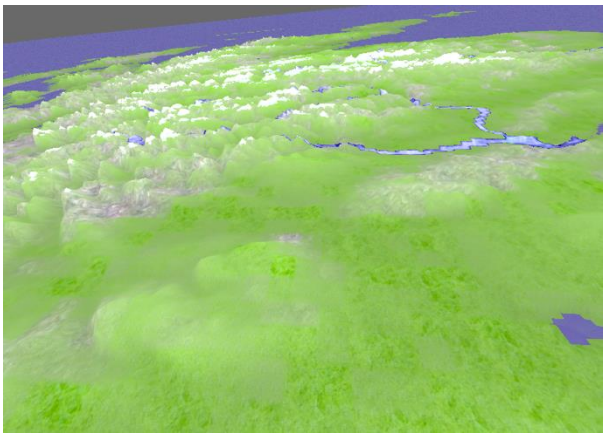
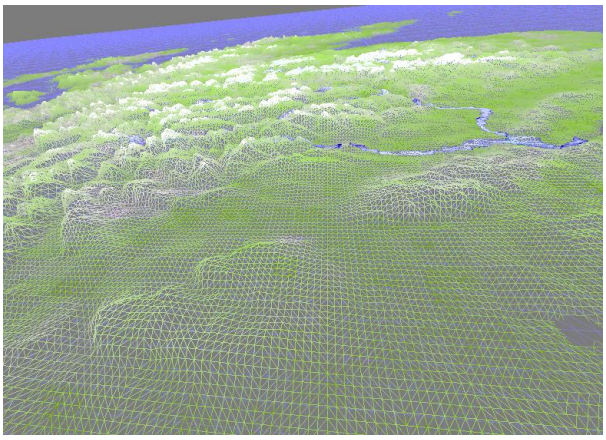



Figure 77: Tessellation Level 16 (Filled/Wireframe)

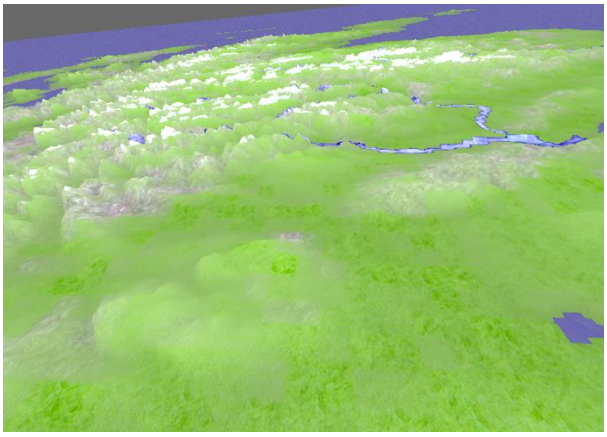
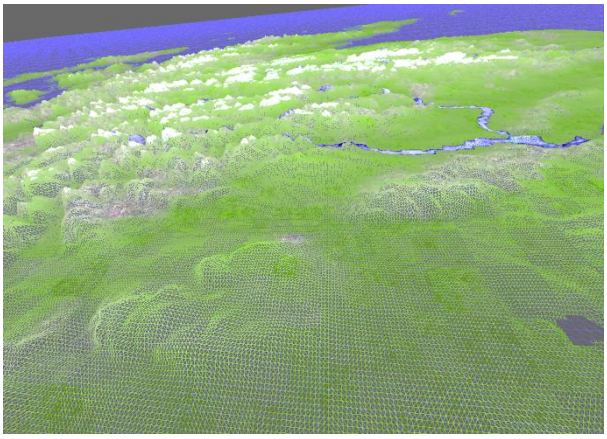



Figure 78: Tessellation Level 32 (Filled/Wireframe)

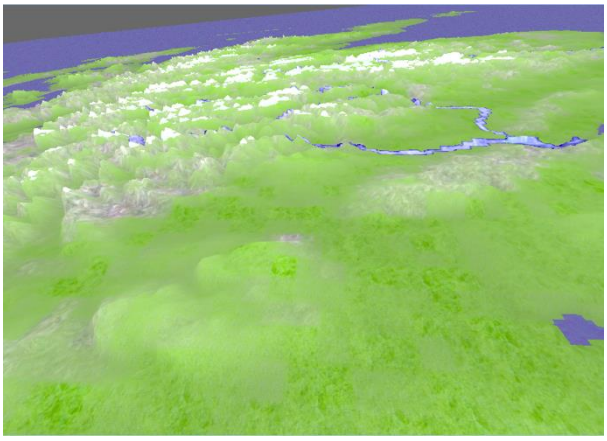
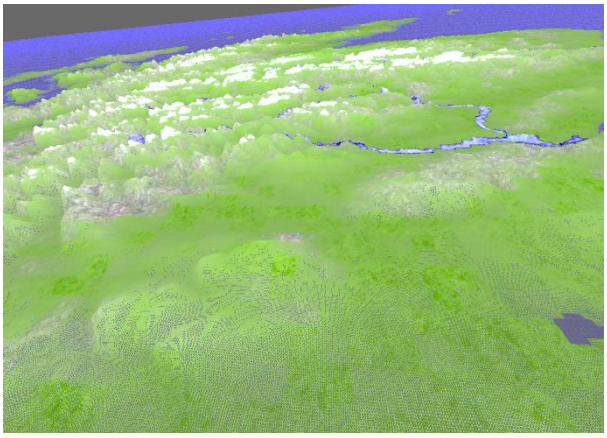



Figure 79: Tessellation Level 64 (Filled/Wireframe)

After a Level of 16 the detail doesn't gain any significant increase on a model of this size with no displacement mapping or detailed mountain heightmap to utilize the extra triangles.

The performance starts to plummet at levels 32 and 64.

The following experiments are controlled use of the simulation; however, the data is still random.

Multiple Textures

This experiment uses the advanced experiments data file to record data but is very simple with a low number of samples taken.

The reason for this experiment was to see the difference in performance when loading multiple textures in the program to use inside the shaders.

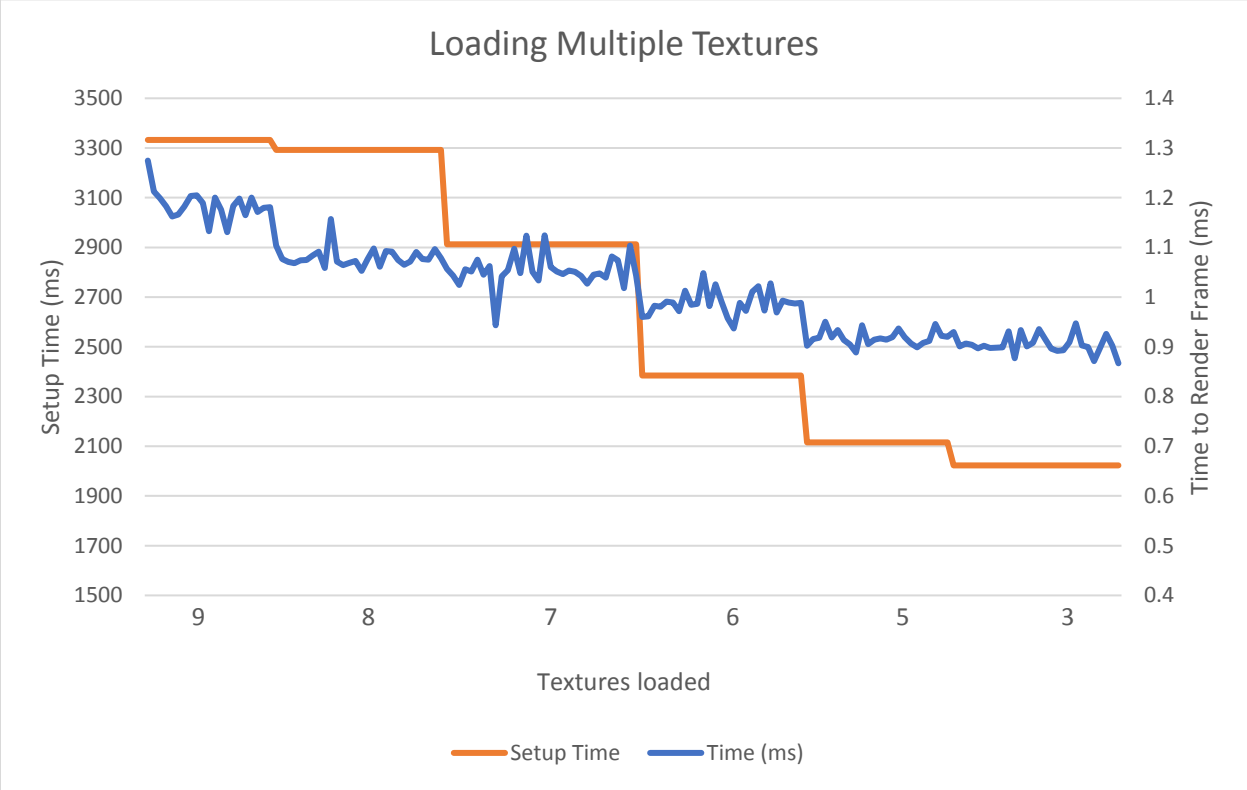


Figure 80: Loading Multiple Textures Comparison Graph

The graph shows that the less textures that are being used in the simulation, the quicker the setup time, my first thought may be that it is the size of the textures that were being loaded causing the differences in setup time.

I investigated the sizes of these files, and found that this was indeed the case, as is shown below in the examples.

For example:

The population texture file being removed accounts for the drop from 9 to 8 textures.

Size of File: 195,099 bytes

Effect: Dropped very slightly in setup time (approx. 40ms), heavily in render time (approx. 0.13ms).

The distance texture file being removed accounts for the drop from 8 to 7 textures.

Size of File: 16,784,613 bytes

Effect: Dropped in both frame render time (approx. 0.03ms) and setup time (approx. 200ms).

The hardness texture file accounts for the drop from 7 to 6 textures and is the same size as the previous, but also is the driving factor behind blending the rock texture into the grass, so may have affected frame render time with that.

Size of File: 16,784,613 bytes

Effect: Dropped in both frame render time (approx. 0.08ms) and setup time (approx. 300ms).

This again isn't conclusive with the setup times but makes sense that the bigger files take a long time to load in, this doesn't account for the 100ms difference between the same sized files.

I would need to repeat this experiment multiple times to get an average setup time which would be more accurate.

Zooming In

There are three methods of zooming into the model; this is either with the Z-axis or rotating the model closer to the camera with pitch and yaw.

Figure 81 (below) shows the performance results of zooming in with the Z-axis, it is ordered from the longest distance zoom of -400 up to 0 where the model is located.

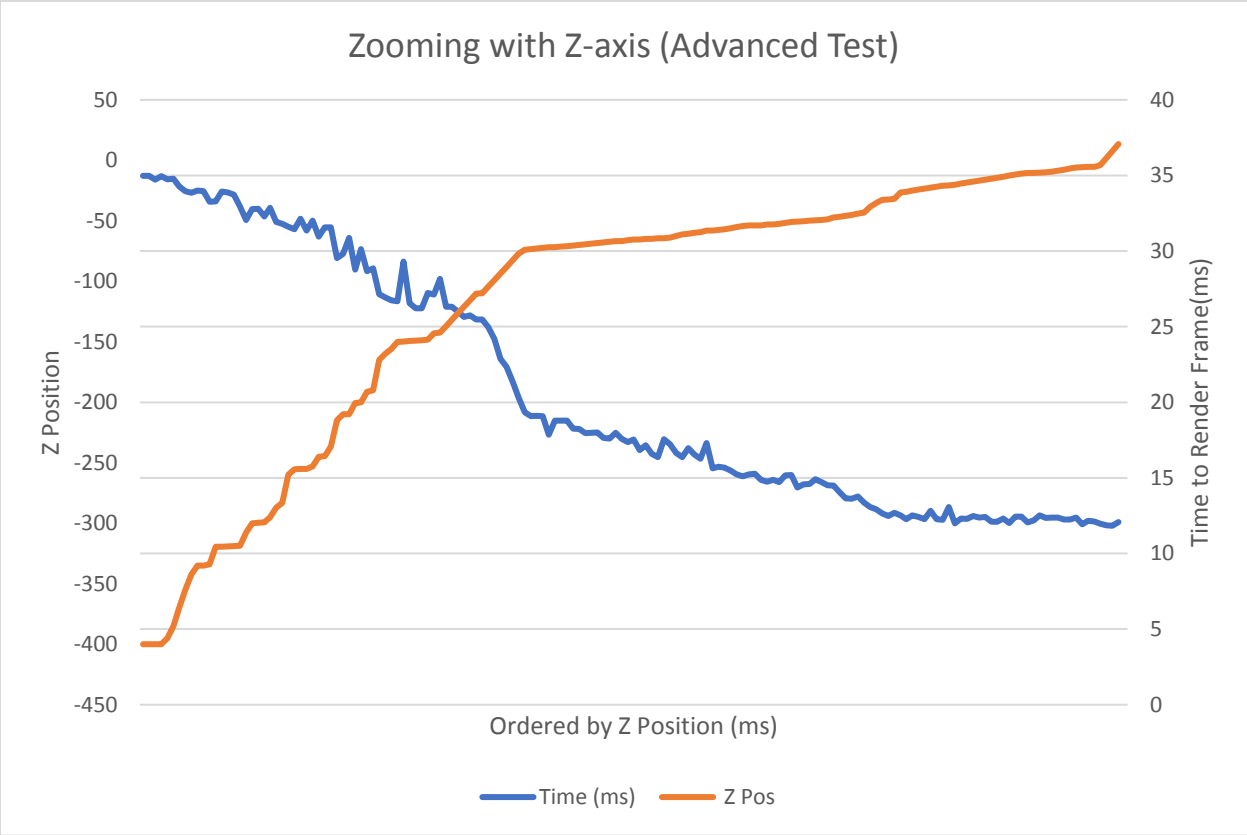


Figure 81: Zooming (Z-Axis) Graph

The render time is quite high due to the other values; the values during the test are as follows and do not change throughout:

Island tessellation: 16
Sea tessellation: 8
Island Modification: 5
Grid Size: 200
Triangle Size: 5
X, Y: -300,300
Pitch, Yaw: 89, 0
Wireframe: Off
Sea Level: 0.0

The graph showed us that the further zoomed out from the model that we were, the more time that it took to render each frame, with exception to a few anomalies this is quite conclusive and I believe the reason is due to culling of triangles that are not on screen, so do not get processed by the GPU until they come back into view.

Another way of zooming into the model is using pitch and yaw, Figure 82 (below) shows this, however, I have used FPS instead of time to render frame in order to make the graph more visually aesthetic.

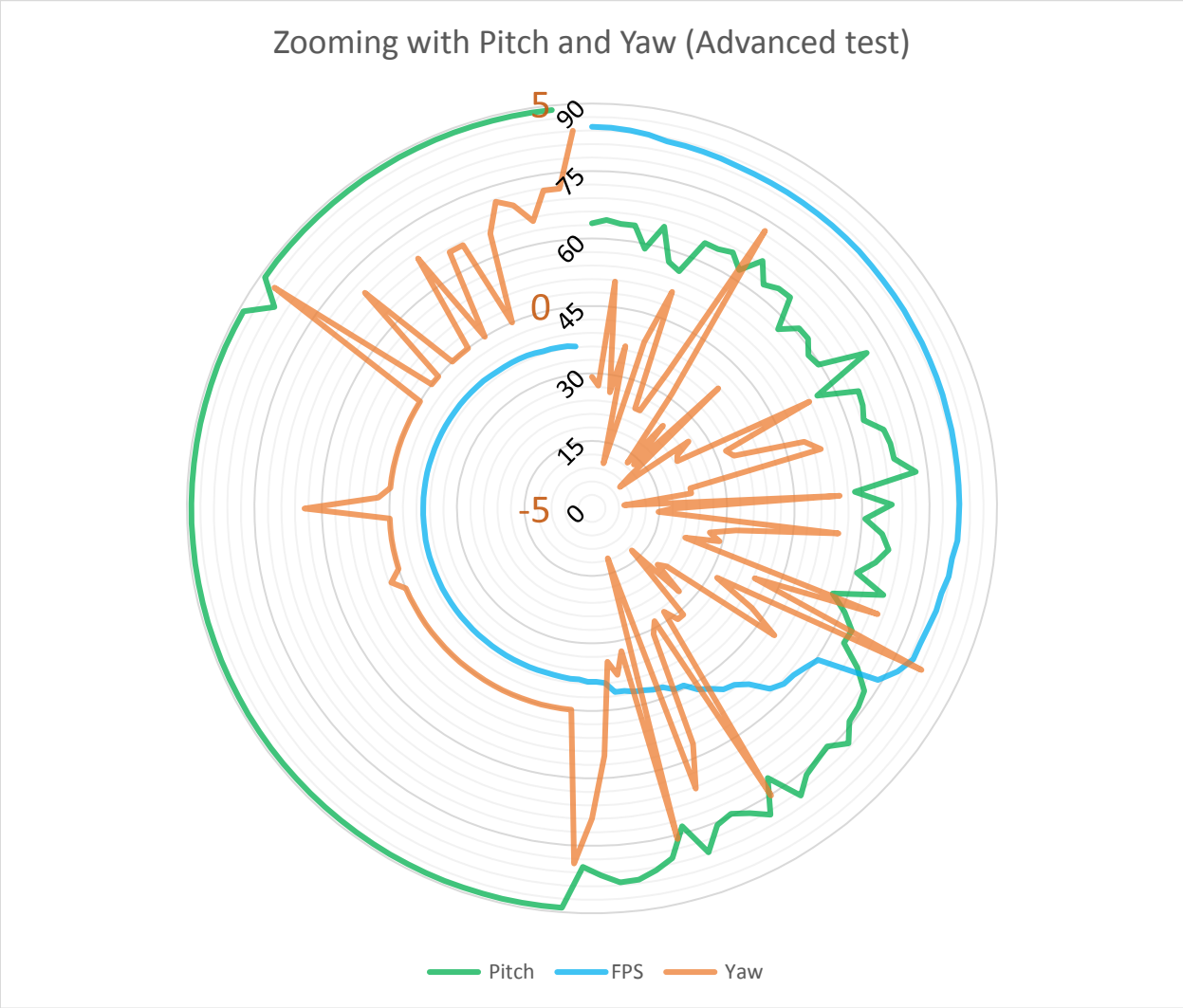


Figure 82: Zooming (Pitch and Yaw) Graph

The fixed values are the same as before except for the Z-axis which is now fixed at -150 and the yaw and pitch will be the variables changed, Pitch is restricted between 0 and 89 and Yaw is restricted between -5 and 5 and uses the red axis.

The graph is ordered using the highest FPS down to the lowest in the clockwise direction.

Some observations of the graph:

- Yaw occasionally didn't affect the frame time as can be shown in the last quarter of the graph.
- Generally, the lower the pitch the higher the FPS is.

X and Y Axis

This experiment is used to see whether what is on the screen makes any difference to the render time.

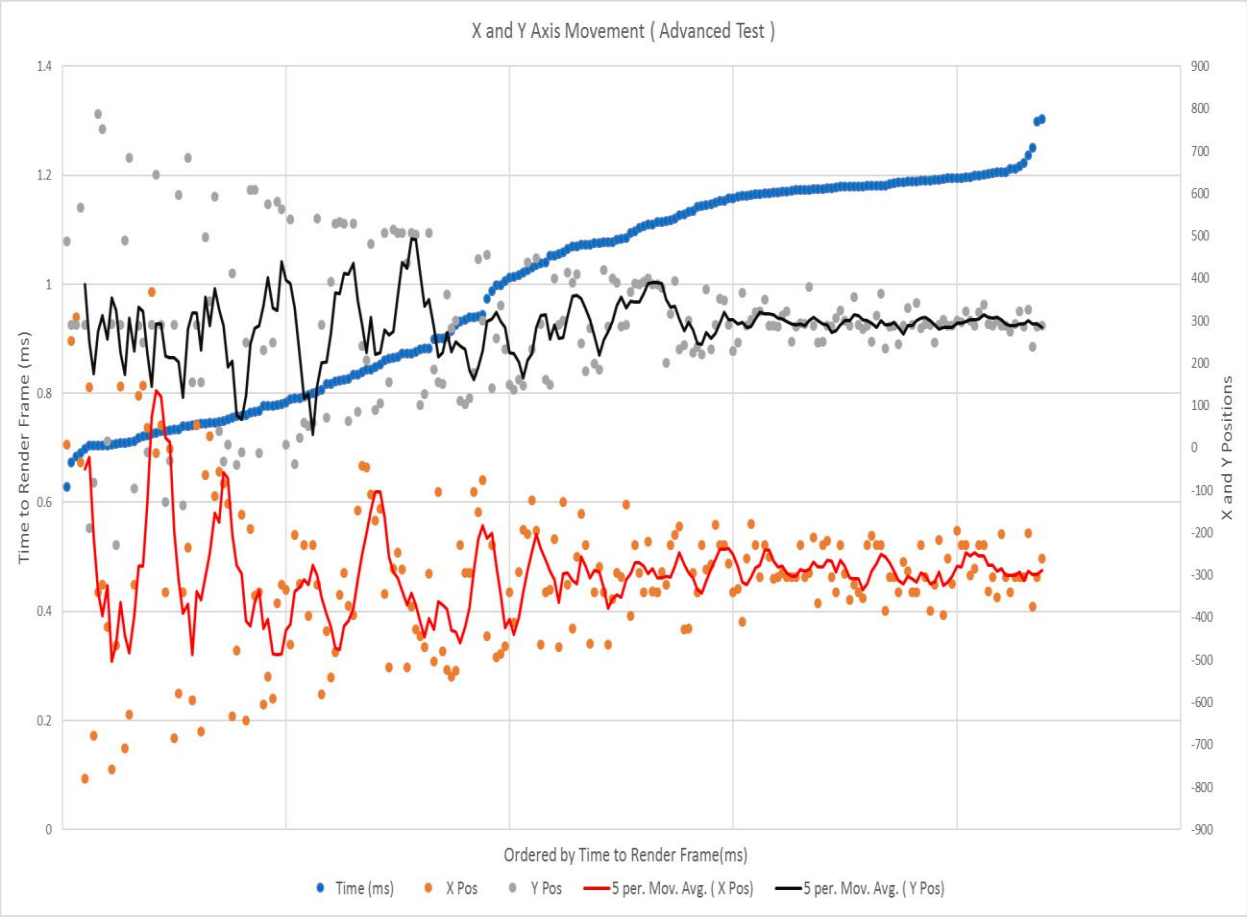


Figure 83: X and Y Position Graph

The graph is ordered by time to render frame (ms).

Some observations of the graph:

- The moving average, using every 5 data points starts off erratic and shows us that is where the data is most spread.
- These averages flatten out at an X, Y value of (300, 300) approximately, which is exactly where the model is located in this experiment.

So as the graph is ordered in time to render, we can tell that it takes longer to render when more of the island is in view, this makes sense, as there are 2 planes of triangles when the island is in full view.

Chapter 7: Evaluation

7.1 Meeting the Objectives

I managed to accomplish the aim of dissertation and most of the objectives, even though I feel the accuracy and graphics of the simulation could be improved.

Research

I did not meet objective 1, whilst researching this topic and CO_2 levels, I discovered many more factors affected climate change and the time it takes for sea levels to rise, so changed the simulation to a sea level approach rather than time based.

I researched different techniques for rendering a 3D island and its surrounding sea as well as other topics, meeting objective 2.

Graphical Simulation

The simulation showcases the sea levels rising and the potential impact that this has on a generated 3D model of the UK with a separate sea entity, meeting the aim of the dissertation and meeting objectives 3 and 4.

Tangible Data and Results Analysis

Data can be measured and evaluated using the simulation and exporting the data file into excel to be analysed.

To recap on what has been measured.

- Different levels of tessellation and its effect on frame render time and setup time.
- Triangle sizes and Grid sizes (they became somewhat linked in the way I have implemented the grid) effect on frame render time and setup time.
- Tessellation differences of 2 planes of triangles on frame render time.
- Multiple textures effect on frame render time and setup time.
- Zooming into the model with z-axis, pitch and yaw and its effect on frame render time.
- Moving x and y axis and its effect on render time.

I was not able to measure land loss, meaning I did not accomplish objective 5, however, with more time and more experience, I could achieve objective 5.

I managed to achieve and exceed objective 6, measuring more than just different detail levels of the simulation; I measured a lot of different aspects of the simulation.

7.2 Software Engineering Process

As mentioned previously, I approached this project with a code and fix mindset, this is not the best approach and doesn't normally factor much planning or design before starting to code which led to me not being able to complete an objective relating to the land loss. I do however feel this was slightly due to inexperience in graphics programming.

7.2.1 Planning and Design

Planning

The Gantt chart was very useful in planning the project, following the Gantt chart to implement each part prevented me from getting confused about what I had done and what I had to do next.

By not researching some of the techniques I would need to use ahead of time, i.e. returning data from the GPU, caused me issues that could have been resolved with this knowledge in hindsight.

If I were to do the planning stage again, I would factor in the learning curve and although bound to change, ensure I knew which techniques I would be using, possibly creating several plans rather than a singular one, so that the plan would be made after all research was done.

Design

I had a pretty clear idea in my head of what I wanted the base structure of the simulation to look like, so not a lot of thought went into the design stage, this turned out to be a bad idea as I was adding more ideas or removing ideas than I would have liked.

If I were to do the design stage again, I would take a more structured approach than just including my ideas into the Gantt chart, such as researching which techniques could be used for each part of the design, then evaluating the choices I had, before trying to implement them.

7.2.2 Implementation and Testing

Implementation

The progression shown in Chapter 4: Implementation was a good approach; breaking down the complicated job into smaller jobs helped me see my progress on the simulation and kept me from getting overwhelmed.

There were a lot of hurdles when it came to develop the simulation, but I persisted with most of the issues and have learnt a lot in the process.

Testing

As I had used a code and fix approach there were not a lot of issues whilst testing the end product that needed to be rectified, as any problems that arose were fixed as soon as I had coded them, it can be quite tricky to debug the GPU as you cannot send information back, or use breakpoints.

Chapter 8: Conclusion

8.1 Overall Outcome

I am pleased with the overall look of the simulation and found it very interesting analyzing the results obtained from the performance side of the simulation.

The results that I obtained gave me a lot of insight into the costliest operations and how to have a good balance of detail in graphics, without impacting the performance too much.

The climate change and sea level modeling side of the simulation did not go as well as I would have liked, with a lack of knowledge and experience or an inability to come up with an feasible solution, due to lack of planning on how to implement certain aspects of the simulation, such as measuring land loss.

The outcome of the simulation shows the impact of sea levels rising on the UK and I believe it to be somewhat accurate, however feel that the erosion aspect of the simulation could be improved as well as some of the tessellation-based graphics, especially when inspecting the island up close.

I believe the results of the different tessellation, grid size, triangle size and the more advanced testing were done well, they showed me the most costly operations of the simulation, I can use this knowledge to improve upon the performance and therefore increase the detail that can be added to the simulation without exceeding an acceptable frame render time.

I was lacking in some research topics, especially relating to past work on real time graphics and techniques, these would have been helpful in programming the simulation, had I researched this topic earlier.

8.2 What I Learned

Learning from the mistakes I made is as valuable as learning from the things that went well; even so I have split them into positives and negatives for clarification.

This does not include my opinions on the simulation as this is mentioned earlier in 7.3

The negatives include what I could have done better on my choice of dissertation subject and the way I approached it.

Positives

- Graphics Pipeline

Understanding the graphics pipeline is key to any graphics demo, game or simulation. I have learnt a great deal about how each shader works towards displaying an image on the screen.

I can apply the knowledge that I have gained during this project to other projects I undertake, with less of a steep learning curve.

- Climate Change

I researched climate change and factors surrounding it that will affect us in the future.

It has put in to perspective what needs to be done to protect the world for future generations and will hopefully do the same for others reading this dissertation.

- Determination

When things were not going my way, I persevered and eventually completed each task I was attempting to accomplish, this doesn't mean everything I did was successful, far from it.

This applies to both managing to succeed in getting the result I was looking for or researching and deciding it was unfeasible or out of scope for this dissertation to implement that feature.

- Planning and Time Management

Having a final deadline for submission enabled me to set goals and meet them to my best ability, having a presentable demo and cohesive dissertation by the submission date required time management and planning, some aspects of this could have been improved however.

Negatives

- Document every stage of development

I was more focused on making the simulation better in one way or another, despite following my plan, I did not document all the issues I had, and did not take screenshots at the time of development of each stage.

This led on to when writing Chapter 4: Implementation of this dissertation, finding it difficult to remember each step I went through to achieve the outcome to be able to explain it easily.

This meant I had to reprogram some of the earlier stages. Using what I had learnt, meant this was quicker than the first time around of programming, however, was a use of time that could have been better spent, either improving the simulation, adding more features to it or explaining the write up of the dissertation more thoroughly.

- Unclear Goals

I didn't feel as though the dissertation had a specific goal, I wanted to explore lots of graphics techniques and performance, and on top of that, try to accurately simulate climate change.

And although I learnt a lot, I don't feel like I have achieved a specific goal even whilst meeting most of my objectives and meeting my Aim to a certain degree.

If I were to do this again, I would concentrate on one aspect of the graphics such as level of detail based on distance and how to best implement that whilst maintaining good performance.

To summarize, I tried to tackle more things than I could handle and ended up with just an OK simulation in most aspects, instead of a visually stunning simulation or an accurate simulation.

- Not my forte

Leading on from unclear aims, at times I felt as though I was doing a geography dissertation, climate change is a vast subject, with its causes, effects and mitigation tactics.

This was an interesting subject; it was one that I didn't know much about, which was partly the reason behind it being a motivation for this dissertation.

It distracted me however, from putting all my efforts into researching graphics and being able to produce a better graphical demo and dissertation with more technical detail on the performance and results of the graphics programming.

8.3 Future Work

If I had more time to work on this project I would like to add to and improve upon a few aspects of the simulation:

Climate change

- Weather

  I would like to add a weather system, which changes relating to the average global temperature, this would require a lot of research into how climate change could affect the weather, e.g. if this could cause snow storms in the summer months or heat waves in the winter months for example, it could also shape the landscape such as turning the green fields of the UK into a desert island.

- How green

  Using a time variable combined with a CO_2 variable instead of a sea level variable could change the outcome, depending on how "green" we were.

- Rivers

  Rivers would be affected by climate change and sea levels rising and would like to simulate this by either making the rivers wider, faster or even could be slower depending on the geological formation around the river.

Graphics

- Erosion

  More accurately simulate erosion of the island, using displacement mapping and/or Perlin noise to texture the land differently as it erodes.

- Textures

  The textures although blended well do not look as realistic as hoped and I would like to add a degree of randomness to them by adding Perlin noise to the patch that the texture was on, tessellate the patch with this noise to add roughness to rock textures for example when the camera is quite close to the plane.

  Another technique could be to use masking, which is another heightmap generated from the previous heightmap. (Andersson, 2014)[22]

Very similar to what I attempted with using the hardness map to decide where the rock texture would be, however this is created procedurally depending on slope calculated using the heightmap.

- Rivers

  Using a separate texture and adding it onto the island for the entire river system of the UK did not produce the outcome I was looking for; it often did not match up with the grooves of the mountainous regions where the water should be flowing.

  This may have been better as a separate entity.

Performance

- Distance based Tessellation

  Changing the tessellation of patches based on the distance from the camera in the scene would improve performance.

  I did have the whole scene change the tessellation based on the Z-axis value, however, this was not the effect I was looking for despite improving performance, as the user is able to use pitch to zoom into the island without changing the tessellation I decided to make the user be able to control the tessellation value instead of this.

References

1. Leahy, S. (2018). *Polar Bears Really Are Starving Because of Global Warming, Study Shows*. [online] News.nationalgeographic.com. Available at: https://news.nationalgeographic.com/2018/02/polar-bears-starve-melting-sea-ice-global-warming-study-beaufort-sea-environment/ [Accessed 1 Apr. 2019].

2. Khronos.org. (2019). *Rendering Pipeline Overview - OpenGL Wiki*. [online] Available at: https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview [Accessed 7 Apr. 2019].

3. Vandevenne, L. (2004). *Texture Generation using Random Noise*. [online] Lodev.org. Available at: https://lodev.org/cgtutor/randomnoise.html [Accessed 3 Apr. 2019].

4. Flafla2.github.io. (2014). *Understanding Perlin Noise*. [online] Available at: https://flafla2.github.io/2014/08/09/perlinnoise.html [Accessed 3 Apr. 2019].

5. En.wikipedia.org. (2019). *Heightmap*. [online] Available at: https://en.wikipedia.org/wiki/Heightmap [Accessed 7 Apr. 2019]

6. Widmark, M. (2012). *Terrain in Battlefield 3: A modern, complete and scalable system*. [online] Media.contentapi.ea.com. Available at: https://media.contentapi.ea.com/content/dam/eacom/frostbite/files/gdc12-terrain-in-battlefield3.pdf [Accessed 1 May 2019].

7. whatsyourimpact.org. (2018). *Main sources of carbon dioxide emissions*. [online] Available at: https://whatsyourimpact.org/greenhouse-gases/carbon-dioxide-emissions [Accessed 3 Apr. 2019].

8. Poore, R., Tracey, C. and Williams Jr., R. (2019). *Fact Sheet fs002-00: Sea Level and Climate*. [online] Pubs.usgs.gov. Available at: https://pubs.usgs.gov/fs/fs2-00/ [Accessed 5 Apr. 2019].

9. Cumming, V. (2019). *This is how far seas could rise thanks to climate change*. [online] Bbc.co.uk. Available at: http://www.bbc.co.uk/earth/story/20160408-this-is-how-far-seas-could-rise-thanks-to-climate-change [Accessed 6 Apr. 2019].

10. Scientific American. (2019). *Could the Oceans Rise Enough to Reverse the Flow of Rivers?*. [online] Available at: https://www.scientificamerican.com/article/could-the-oceans-rise-enough/ [Accessed 8 Apr. 2019].

11. Shenker, J. (2009). *Nile Delta: 'We are going underwater. The sea will conquer our lands'*. [online] the Guardian. Available at: https://www.theguardian.com/environment/2009/aug/21/climate-change-nile-flooding-farming [Accessed 1 May 2019].

12. Freemaptools.com. (2019). *Elevation Finder*. [online] Available at: https://www.freemaptools.com/elevation-finder.htm [Accessed 10 Apr. 2019].

13. Dasgupta, S., Meisner, C. and More (2007). *The Impact of Sea Level Rise on Developing Countries: A Comparative Analysis*. [online] Documents.worldbank.org. Available at: http://documents.worldbank.org/curated/en/156401468136816684/pdf/wps4136.pdf [Accessed 1 May 2019].

14. Mulligan, J., Ellison, G. and Levin, K. (2018). *6 Ways to Remove Carbon Pollution from the Sky | World Resources Institute*. [online] Wri.org. Available at: https://www.wri.org/blog/2018/09/6-ways-remove-carbon-pollution-sky [Accessed 25 Apr. 2019].

15. Climate.nasa.gov. (2019). *Climate Change: Earth Now*. [online] Available at: https://climate.nasa.gov/earth-now [Accessed 2 Apr. 2019].

16. Vesl.jpl.nasa.gov. (2019). *Slowdown in Antarctic Mass Loss from Solid Earth and Sea-Level Feedbacks Simulation | Sea Level | VESL | JPL | NASA*. [online] Available at: https://vesl.jpl.nasa.gov/sea-level/slr-uplift/ [Accessed 6 Apr. 2019].

17. Sealevel.nasa.gov. (2019). *Coastline Retreat From Sea-Level Rise Simulation | Sea Level | VESL | JPL | NASA*. [online] Available at: https://sealevel.nasa.gov/vesl/web/sea-level/slr-eustatic/ [Accessed 8 Apr. 2019].

18. Bunnell, M. (2005). *GPU Gems 2*. [online] NVIDIA. Available at: https://developer.nvidia.com/gpugems/GPUGems2/gpugems2_chapter07.html [Accessed 9 Apr. 2019].

19. En.wikipedia.org. (2019). *The Wash*. [online] Available at: https://en.wikipedia.org/wiki/The_Wash [Accessed 14 Apr. 2019].

20. Overvoorde, A. (2012). *OpenGL - Transformations*. [online] Open.gl. Available at: http://open.gl/transformations#TransformationsinOpenGL [Accessed 16 Apr. 2019].

21. Möller, M. (2019). *Open Hardware Monitor - Core temp, fan speed and voltages in a free software gadget*. [online] Openhardwaremonitor.org. Available at: https://openhardwaremonitor.org [Accessed 19 Apr. 2019].

22. Andersson, J. (2014). *Terrain Rendering in Frostbite*. [online] Dice.se. Available at: http://www.dice.se/wp-content/uploads/2014/12/Chapter5-Andersson-Terrain_Rendering_in_Frostbite.pdf [Accessed 1 May 2019].

Background Reading

1. En.wikipedia.org. (2019). *Shader*. [online] Available at: https://en.wikipedia.org/wiki/Shader [Accessed 3 Apr. 2019].

2. Learnopengl.com. (2019). *LearnOpenGL - Textures*. [online] Available at: https://learnopengl.com/Getting-started/Textures [Accessed 18 Apr. 2019].

3. Khronos.org. (2019). *OpenGL Wiki*. [online] Available at: https://www.khronos.org/opengl/wiki/ [Accessed 6 Apr. 2019].

4. Callery, S. (2019). *The Causes of Climate Change*. [online] Climate Change: Vital Signs of the Planet. Available at: https://climate.nasa.gov/causes/ [Accessed 27 Jan. 2019].

5. Scott, M. and Lindsey, R. (2016). *Which emits more carbon dioxide: volcanoes or human activities? | NOAA Climate.gov*. [online] Climate.gov. Available at: https://www.climate.gov/news-features/climate-qa/which-emits-more-carbon-dioxide-volcanoes-or-human-activities [Accessed 3 Apr. 2019].

6. Glick, D. (2019). *The Big Thaw*. [online] National Geographic. Available at: https://www.nationalgeographic.com/environment/global-warming/big-thaw/ [Accessed 6 Apr. 2019].

7. En.wikipedia.org. (2019). *Global warming*. [online] Available at: https://en.wikipedia.org/wiki/Global_warming#Greenhouse_gases [Accessed 5 Apr. 2019].

8. Ramónster (2019). *How to use QueryPerformanceCounter?*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/1739259/how-to-use-queryperformancecounter [Accessed 21 Apr. 2019].

Images Used

Figure 1: Inundation of Water

https://climate.nasa.gov/interactives/climate-time-machine

Figures 2,3: Gannt Charts

https://www.teamgantt.com

Figure 4: OpenGL Rendering Pipeline

https://www.khronos.org/opengl/wiki/File:RenderingPipeline.png

Figure 5: Example of Terrain Tessellation in OpenGL

https://docs.nvidia.com/gameworks/content/gameworkslibrary/graphicssamples/opengl_samples/images/terraintessellation-screenshot_thumb_700_0.jpg

Figure 6: Displacement Map Example (Detailed Terrain)

https://media.contentapi.ea.com/content/dam/eacom/frostbite/files/gdc12-terrain-in-battlefield3.pdf

Figure 7: The Greenhouse Effect

https://www.niwa.co.nz/sites/niwa.co.nz/files/styles/large/public/sites/default/files/images/imported/0007/73447/Greenhouse_effect2_0.jpg?itok=Wd-IRLRk

Figure 8: Carbon Cycle

https://www.researchgate.net/profile/Francesco_Tubiello/publication/323696018/figure/fig6/AS:560281676980225@1510592969615/Schematic-representation-of-the-overall-perturbation-of-the-global-carbon-cycle-caused-2.png

Figure 9: Fossil Fuel Emissions

https://www.climate.gov/sites/default/files/volcano-v-fossilfuels-1750-2013-620.png

Figure 10: Flooding image

https://www.northnorfolknews.co.uk/polopoly_fs/1.3086059!/image/3405277030.jpg_gen/derivatives/landscape_490/3405277030.jpg

Figure 11: Elevation Map (Nile Delta)

https://www.freemaptools.com/elevation-finder.htm

Figure 12: Egypt Impact of Sea Level Rising

http://documents.worldbank.org/curated/en/156401468136816684/pdf/wps4136.pdf

Figures 13,14: Search Topic Trends

https://trends.google.com/trends/explore?date=all&q=%2Fm%2F0gx2d,%2Fm%2F0d063v,%2Fm%2F0cs9q

Figure 15: Satellite Measurements

https://climate.nasa.gov/earth-now

Figure 16: Antarctica Simulation

https://vesl.jpl.nasa.gov/sea-level/slr-uplift/

Figure 17: Sea Level Simulation

https://sealevel.nasa.gov/vesl/web/sea-level/slr-eustatic/

Figure 19: Height Map

https://i.imgur.com/V0txo.jpg

Figure 22: Ground Strength Map

https://www.bgs.ac.uk/products/groundConditions/images/strength.jpg

Figure 23: Population Map

http://www.geog.leeds.ac.uk/papers/98-8/pop.gif

Figure 24: Textures

Sand:
https://previews.123rf.com/images/wutichaistudio/wutichaistudio1706/wutichaistudio170600346/80373036-golden-sand-texture-background-h-i-g-h-r-e-s-o-l-u-t-i-o-n.jpg

Grass:

https://i.pinimg.com/564x/ef/23/8b/ef238b5a703395ae88807b1e5eb4c5ad.jpg

Rock:

http://wdc3d.com/wp-content/uploads/2010/04/stone_1_2048x2048.jpg

Snow:

http://cdn.hasshe.com/img/s/_0xatQ5zSNt_oeEo2d4yEAHaHa.jpg

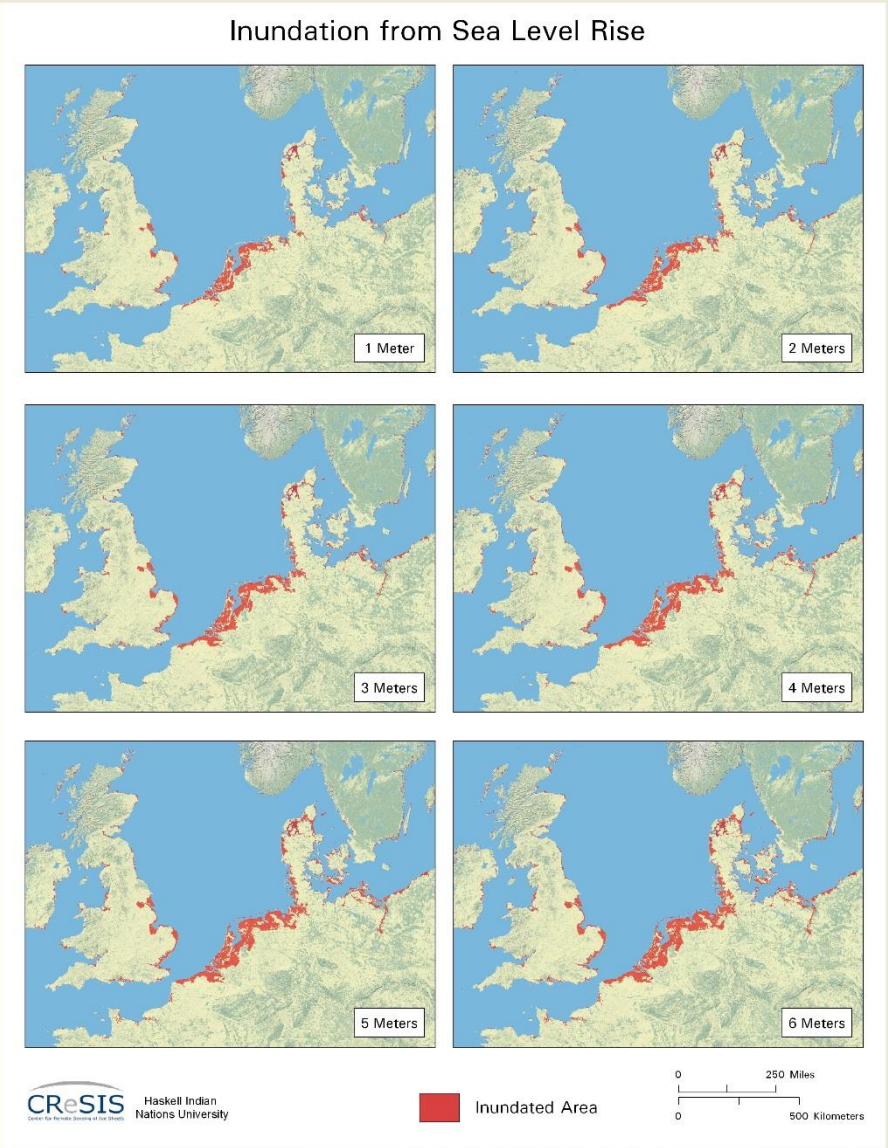
Water:

https://i.stack.imgur.com/nRHPu.png

Appendix

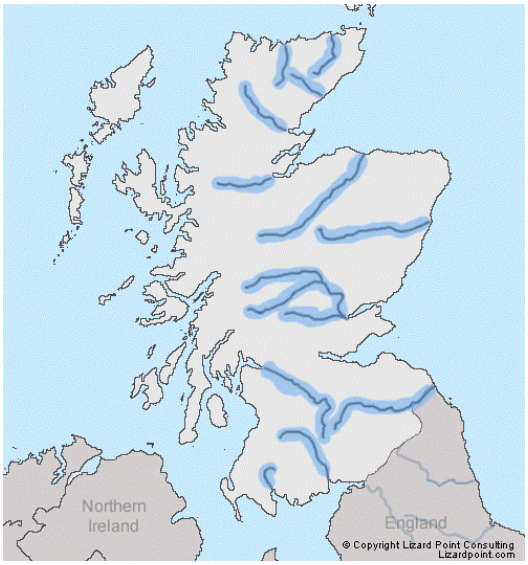
Appendix 1 – Sea Level Rising

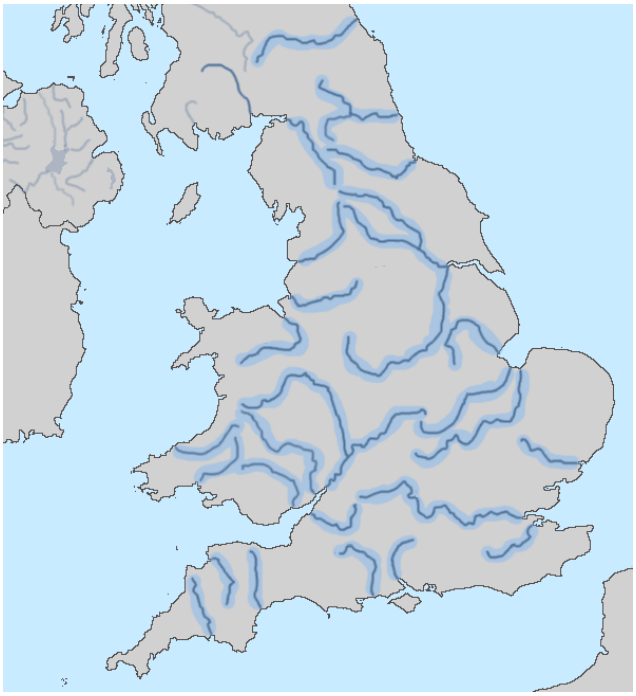
Obtained (3rd November 2018, https://www.cresis.ku.edu/sites/default/files/Research/Maps/NorthernEurope/JPG/northern_europe_1to6.jpg)



Appendix 2 – River Maps (pre-edit)

Obtained (26th February 2019, https://lizardpoint.com/geography/images/maps/england-wales-rivers-highlighted.gif , https://lizardpoint.com/geography/images/maps/scotland-rivers-highlighted.gif)





Appendix 3 – Population Map (Greyscale)

Appendix 4 –Two Triangle Quad Planes Code

```
void SeaPlane(Renderer& renderer) {

    OGLMesh* seaMesh = new OGLMesh();

    //Verticies Coordinates (x,y,z)
    std::vector<Vector3> verts = {
        Vector3(seaSize, seaSize, seaLevel),  //Bottom Right
        Vector3(seaSize, -seaSize, seaLevel), //Top Right
        Vector3(-seaSize, seaSize, seaLevel), //Bottom Left
        Vector3(-seaSize, -seaSize, seaLevel) //Top Left
    };

    //Verticies Colours (r,g,b,a)
    std::vector<Vector4> cols = {
        Vector4(0.0f,0.0f,1.0f,0.5f), //Blue Half Alpha
        Vector4(0.0f,0.0f,1.0f,0.5f), //Blue Half Alpha
        Vector4(0.0f,0.0f,1.0f,0.5f), //Blue Half Alpha
        Vector4(0.0f,0.0f,1.0f,0.5f)  //Blue Half Alpha
    };

    //Set verticies and colours of mesh. specify geometry primitive as triangle strip
    seaMesh->SetVertexPositions(verts);
    seaMesh->SetVertexColours(cols);
    seaMesh->SetPrimitiveType(GeometryPrimitive::TriangleStrip);
    seaMesh->UploadToGPU();

    //Rotate plane
    Matrix4 modelMat = Matrix4::Rotation(270.0f, Vector3(1.0f, 0.0f, 0.0f));

    renderer.AddRenderObject(new RenderObject(seaMesh, modelMat));
}

void IslandPlane(Renderer& renderer) {

    OGLMesh* islandMesh = new OGLMesh();

    //Verticies Coordinates (x,y,z)
    std::vector<Vector3> verts = {
        Vector3(seaSize/2, seaSize/2, seaLevel + 1.0f),  //Bottom Right
        Vector3(seaSize/2, -seaSize/2, seaLevel + 1.0f), //Top Right
        Vector3(-seaSize/2, seaSize/2, seaLevel + 1.0f), //Bottom Left
        Vector3(-seaSize/2, -seaSize/2, seaLevel + 1.0f) //Top Left
    };

    //Verticies Colours (r,g,b,a)
    std::vector<Vector4> cols = {
        Vector4(0.0f,1.0f,0.0f,1.0f), //Green
        Vector4(0.0f,1.0f,0.0f,1.0f), //Green
        Vector4(0.0f,1.0f,0.0f,1.0f), //Green
        Vector4(0.0f,1.0f,0.0f,1.0f)  //Green
    };

    //Texture Coordinates (u,v)
    std::vector<Vector2> tex = {
        Vector2(1.0f, 0.0f), //Bottom Right
        Vector2(1.0f, 1.0f), //Top Right
        Vector2(0.0f, 0.0f), //Bottom Left
        Vector2(0.0f, 1.0f)  //Top Left
    };

    //Rotate plane
    Matrix4 modelMat = Matrix4::Rotation(270.0f, Vector3(1.0f, 0.0f, 0.0f));

    //Load Heightmap
    TextureBase* heightTex = OGLTexture::RGBATextureFromFilename("UK.png");

    //Set verticies, colours and texture coords of mesh. specify geometry primitive as triangle strip
    islandMesh->SetVertexPositions(verts);
    islandMesh->SetVertexColours(cols);
    islandMesh->SetVertexTextureCoords(tex);
    islandMesh->SetPrimitiveType(GeometryPrimitive::TriangleStrip);
    islandMesh->UploadToGPU();

    //Declare Render Object so can set texture on it before sending to renderer
    RenderObject* Object = new RenderObject(islandMesh, modelMat);
    Object->SetBaseTexture(heightTex);

    renderer.AddRenderObject(Object);
}
```

Appendix 5 – Grid of Quads Code

```
void SeaPlane(Renderer& renderer) {

    OGLMesh* seaMesh = new OGLMesh();

    vector<Vector3> planeVector;
    vector<Vector4> colsVector;
    int TRI_SIZE = 20;
    const int TRI_AMOUNT = 10;

    for (int y = 0; y < TRI_AMOUNT; y++) {

        if (y % 2 == 0) {
            for (int x = 0; x <= TRI_AMOUNT; x++) {

                planeVector.emplace_back(Vector3(x * TRI_SIZE, (y + 1) * TRI_SIZE, seaLevel));
                planeVector.emplace_back(Vector3(x * TRI_SIZE, y * TRI_SIZE, seaLevel));

                colsVector.emplace_back(Vector4(0.0f, 0.0f, 1.0f, 0.5f)); //Blue Half Alpha
                colsVector.emplace_back(Vector4(0.0f, 0.0f, 1.0f, 0.5f)); //Blue Half Alpha

            }
        } else {
            for (int x = TRI_AMOUNT; x >= 0 ; x--) {

                planeVector.emplace_back(Vector3(x * TRI_SIZE, (y + 1) * TRI_SIZE, seaLevel));
                planeVector.emplace_back(Vector3(x * TRI_SIZE, y * TRI_SIZE, seaLevel));


                colsVector.emplace_back(Vector4(0.0f, 0.0f, 1.0f, 0.5f)); //Blue Half Alpha
                colsVector.emplace_back(Vector4(0.0f, 0.0f, 1.0f, 0.5f)); //Blue Half Alpha

            }
        }
    };
    //Set verticies and colours of mesh. specify geometry primitive as triangle strip
    seaMesh->SetVertexPositions(planeVector);
    seaMesh->SetVertexColours(colsVector);
    seaMesh->SetPrimitiveType(GeometryPrimitive::TriangleStrip);
    seaMesh->UploadToGPU();

    //Rotate plane
    Matrix4 modelMat = Matrix4::Rotation(270.0f, Vector3(1.0f, 0.0f, 0.0f));

    renderer.AddRenderObject(new RenderObject(seaMesh, modelMat));
}
```

Appendix 6 – Fragment Shader Code

```
uniform sampler2D populationTex;
uniform sampler2D snowTex;

in vec4 gl_FragCoord;

in Vertex {
    vec2 texCoord;
} IN;

out vec4 fragColor;

uniform float seaLevel;
uniform int showPopulation;

float worldheight;
float distanceFromCoast;
float hardness;
float population;

float detail;

vec4 rocktexture;
vec4 grasstexture;
vec4 rivertexture;
vec4 sandtexture;
vec4 snowtexture;
vec4 alphamap;

vec4 blend;

void main(void){

    worldheight = texture(mainTex,IN.texCoord).x;
    distanceFromCoast = texture(distanceTex, IN.texCoord).x;
    hardness = (texture(hardnessTex, IN.texCoord).y * 1.5) - (texture(hardnessTex,IN.texCoord).x * 1.5);
    population = texture(populationTex,IN.texCoord).x;

    rocktexture = texture(rockTex, IN.texCoord * 100);
    grasstexture = texture(grassTex, IN.texCoord * 1000);
    rivertexture = texture(riverTex, IN.texCoord);
    sandtexture = texture(sandTex, IN.texCoord * 40);
    snowtexture = texture(snowTex, IN.texCoord * 1000);

    if (worldheight < 0.0002){
        discard;
    } else {

        fragColor = texture(mainTex,IN.texCoord);
        fragColor.x = 0.0f;
        fragColor.z = 0.0f;

        if ( worldheight < 0.005){
            fragColor = sandtexture;
        }

        if ( worldheight > 0.005 && worldheight < 0.3 ){
            fragColor = grasstexture;
        }

        if ( worldheight > 0.3 && worldheight < 0.6 ){
            fragColor = rocktexture;
        }

        if (worldheight > 0.6){
            fragColor = snowtexture;
        }
    }
```

Appendix 7 – Updated Fragment Shader Code

```
void main(void){

    worldheight = texture(mainTex,IN.texCoord).x;
    distanceFromCoast = texture(distanceTex, IN.texCoord).x;
    hardness = (texture(hardnessTex, IN.texCoord).y * 1.5) - (texture(hardnessTex,IN.texCoord).x * 1.5);
    population = texture(populationTex,IN.texCoord).x;

    rocktexture = texture(rockTex, IN.texCoord * 100);
    grasstexture = texture(grassTex, IN.texCoord * 1000);
    rivertexture = texture(riverTex, IN.texCoord);
    sandtexture = texture(sandTex, IN.texCoord * 40);
    snowtexture = texture(snowTex, IN.texCoord * 1000);

    blend = (-hardness * (rocktexture)) + ((1.0 + hardness) * grasstexture);

    if (worldheight < 0.0002){
        discard;
    } else {

        if(rivertexture.z > 0.1){


            blend = ( -hardness * rocktexture) + ((worldheight + 1.0 + hardness) * rocktexture) + ((worldheight + 1.0 + hardness) * rivertexture);


            fragColor = blend;
            fragColor.y -= 0.5;
            fragColor.x -= 0.5;

        } else {


            blend = ( -hardness * rocktexture) + ((worldheight + 1.0 + hardness) * grasstexture);


            if(worldheight > 0.75){

                blend = mix(blend, snowtexture, worldheight);

            }

            fragColor = blend;

            if (showPopulation > 0 && population > 0.1){

                fragColor.y -= population;
                fragColor.z -= population;
                fragColor.x =  population;
            }

        }
    }

    if ((worldheight * 0.95) + (distanceFromCoast * 0.05) - (hardness * 0.15)  < seaLevel ){

            blend = ( -hardness * rocktexture) + ((worldheight + 1.0 + hardness) * sandtexture) + (grasstexture/2);
            fragColor = blend;

    }
```

Appendix 8 – Sea Tessellation Shader Code

```
#version 400 core
layout(vertices =4) out;

in Vertex
{
    vec2 texCoord;
} IN[];

out Vertex
{
    vec2 texCoord;
} OUT[];


//uniform float distanceFromCamera;
uniform int seaTess;

void main(void)
{

    //float dfc = distanceFromCamera/30;
    int tessMod = seaTess;

    gl_TessLevelInner[0] = min(64, max(tessMod, 1));
    gl_TessLevelInner[1] = min(64, max(tessMod, 1));
    gl_TessLevelOuter[0] = min(64, max(tessMod, 1));
    gl_TessLevelOuter[1] = min(64, max(tessMod, 1));
    gl_TessLevelOuter[2] = min(64, max(tessMod, 1));
    gl_TessLevelOuter[3] = min(64, max(tessMod, 1));

    OUT[gl_InvocationID].texCoord = IN[gl_InvocationID].texCoord * tessMod;

    gl_out[gl_InvocationID].gl_Position = gl_in[gl_InvocationID].gl_Position;

}
```

Appendix 9 – Test Plan

Test No	Test Name/ Description	Test Purpose	Expected Result	Actual Result	Solution - if Applicable
1	Running program	Ensure the program starts	The program should run.	Program Runs.	N/A
2	Tessellation variable (sea)	Correct changing of tessellation for the sea.	Pressing O/K should increase/decrease the tessellation by 1.	Tessellation variable (sea) increased/decreased however was too fast.	Change the button down to button pressed.
3	Tessellation variable (island)	Correct changing of tessellation for the island.	Pressing P/L should increase/decrease the tessellation by 1.	Tessellation variable (island) increased/decreased however was too fast.	Change the button down to button pressed.
4	X-Axis and Y-Axis Camera Control	Check left, right up down controls.	W – Up A – Left S – Down D – Right	WASD works as expected. Speed changes depending on FPS.	N/A (Could change Speed depending on FPS)
5	Z-Axis Camera Control	Check Zoom in/out Controls work as expected.	Q – Zoom in Slow Z – Zoom out Slow Scroll(Up/Down) faster zooming.	Q Z and Scrolling works as expected. Speed changes depending on FPS.	N/A (Could change Speed depending on FPS)
6	Pitch and Yaw Camera Control	Check Pitch and Yaw controls work as expected.	Holding Left mouse and dragging should change pitch and yaw.	Pitch and yaw changed as expected, however model could be lost if too far.	Limit the values that the pitch and yaw can be between.
7	Sea Level variable	Check sea level rises and drops.	Pressing UP/Down should change cause sea level to rise/drop.	Works as expected, although issue with tessellated water texture noticed.	Water texture contributes to z-axis value during tessellation, remove.
8	Accuracy of FPS and Time/Frame	Check consistent FPS, and its accuracy.	Using a thread sleep of 1000ms should result in 1000ms timer.	The FPS is relatively stable and the timer results were accurate to the sleep time.	N/A
9	Wireframe	Ensure wireframe mode worked.	Pressing F should toggle between wireframe and filled mode.	Wireframe toggled between on and off. FPS decreased when on.	N/A
10	Population Overlay	Ensure Population mode worked.	Pressing F1 should toggle the population overlay on and off.	Population overlay toggled on and off, colour not visible enough and blends with textures.	While adding the to the red value of the fragment, the green and blue value needs to be reduced.

Appendix 10 – Experiment Tables

Tessellation Test with default values.						
Test Number	Grid Size	Triangle Size	Island Triangle Mod	Sea Level	Sea Tessellation	Island Tessellation
1.0	200	10	2	0	1	1
1.1	200	10	2	0	2	2
1.2	200	10	2	0	4	4
1.3	200	10	2	0	8	8
1.4	200	10	2	0	16	16
1.5	200	10	2	0	32	32
1.6	200	10	2	0	64	64

Triangle Size Test with Tessellation values of 1 (Island mod cant produce value less than 1)						
Test Number	Grid Size	Triangle Size	Island Triangle Mod	Sea Level	Sea Tessellation	Island Tessellation
2.0	200	2	2	0	1	1
2.1	200	5	2	0	1	1
2.2	200	10	2	0	1	1
2.3	200	25	2	0	1	1
2.4	200	50	2	0	1	1

Grid Size Test with Tessellation values of 1						
Test Number	Grid Size	Triangle Size	Island Triangle Mod	Sea Level	Sea Tessellation	Island Tessellation
3.0	100	10	2	0	1	1
3.1	250	10	2	0	1	1
3.2	500	10	2	0	1	1
3.3	1000	10	2	0	1	1
3.4	5000	10	2	0	1	1

Appendix 11 – Experiment Results Example (non-graphical)

	A	B	C	D	E	F	G	H	I
1	**Test 1**	Test 1.0	**Setup**	3124.14		Test 1.1	**Setup**	3097.36	
2		**Average ms**	*ms/Frame*	*FPS*		**Average ms**	*ms/Frame*	*FPS*	
3		0.394976226	0.368483	2713.83	TRUE	0.435953333			
4		**Average FPS**	0.368793	2711.55	TRUE	**Average FPS**			
5		2531.80	0.368793	2711.55	TRUE	2293.82			
6			0.371887	2688.99	TRUE				
7			0.374671	2669.01	TRUE				
8		**Standard Deviation**	0.374981	2666.80	TRUE	**Standard Deviation**			
9		0.014416808	0.379003	2638.50	TRUE	0.005638566	0.421389	2373.10	TRUE
10		**Lower Quartile**	0.38024	2629.92	TRUE	**Lower Quartile**	0.422936	2364.42	TRUE
11		0.3868145	0.380859	2625.64	TRUE	0.4319855	0.425411	2350.67	TRUE
12		**Upper Quartile**	0.381787	2619.26	TRUE	**Upper Quartile**	0.425721	2348.96	TRUE
13		0.40607425	0.382406	2615.02	TRUE	0.44049375	0.426649	2343.85	TRUE
14		**Interquartile range**	0.3855	2594.03	TRUE	**Interquartile range**	0.427577	2338.76	TRUE
15		0.01925975	0.386428	2587.80	TRUE	0.00850825	0.428196	2335.38	TRUE
16		**Upper**	0.386428	2587.80	TRUE	**Upper**	0.429124	2330.33	TRUE
17		0.434963875	0.386428	2587.80	TRUE	0.453256125	0.429124	2330.33	TRUE
18		**Lower**	0.386737	2585.74	TRUE	**Lower**	0.43098	2320.29	TRUE
19		0.357924875	0.387047	2583.67	TRUE	0.419223125	0.43098	2320.29	TRUE
20			0.387047	2583.67	TRUE		0.43129	2318.63	TRUE
21		Outliers Removed: 4	0.387356	2581.60	TRUE	Outliers Removed: 12	0.431908	2315.31	TRUE
22			0.387356	2581.60	TRUE		0.431908	2315.31	TRUE
23			0.387666	2579.54	TRUE		0.432218	2313.65	TRUE
24			0.387666	2579.54	TRUE		0.433455	2307.04	TRUE
25			0.387666	2579.54	TRUE		0.434383	2302.12	TRUE
26			0.388284	2575.43	TRUE		0.434383	2302.12	TRUE
27			0.388284	2575.43	TRUE		0.434383	2302.12	TRUE
28			0.388594	2573.38	TRUE		0.435002	2298.84	TRUE
29			0.388594	2573.38	TRUE		0.435312	2297.20	TRUE
30			0.388903	2571.34	TRUE		0.435621	2295.57	TRUE
31			0.389213	2569.29	TRUE		0.43593	2293.95	TRUE
32			0.389213	2569.29	TRUE		0.43624	2292.32	TRUE
33			0.389522	2567.25	TRUE		0.436549	2290.69	TRUE
34			0.389522	2567.25	TRUE		0.436859	2289.07	TRUE
35			0.389522	2567.25	TRUE		0.436859	2289.07	TRUE