322COM  
Advanced Graphics Programming  
Terrain Generation  
(Re-sit)  
  
Task  
2.  
Written Report

Student ID

9032499

Coventry University

**Author’s Note:**

GitHub repository

< <https://github.coventry.ac.uk/lattb/9032499_322COM_Task2_Resit> >

Video

< <https://www.youtube.com/watch?v=bmqeZnhqfaw> >

**Table of Contents**

Table of Contents………………………………………………………………………………...1.

Table of Figures………………………………………………………………………………….1.

Introduction…...………………………………………………………………………...………..3.

Procedural Generation (Textures/Animation) …………………………………………………...4.

Camera…………………………………………………………………………………………..11.

Optimization & Performance……………………………………………………………………12.

Application Final Output………………………………………………………………………..14.

Reflection…………………………………………………………………………………….….15.

List of References……………………………………………………………………………….16.

**Table of Figures**

Figure 1 Procedural Plane………………………………………………………………………..4.

Figure 2 Modified Noise Algorithm Terrain……………………………………………………..4.

Figure 3 Initialization find four Corners…………………………………………………………5.

Figure 4 Diamond Step Iteration one…………………………………………………………….5.

Figure 5 Square Step one…………………………………………………………………...……6.

Figure 6 Diamond Step Interaction two………………………………………………………….6.

Figure 7 Textured Terrain (Grass)………………………………………………………………..7.

Figure 8 Multi-Textured Terrain…………………………………………………………………7.

Figure 9 Multi-Textured Terrain Modifiers……………………………………………………...8.

Figure 10 Water Animation Modifiers…………………………………………………………..9.

Figure 11 Animated Water Basics………………………………………………………………10.

Figure 12 Camera Initialization…………………………………………………………………11.

Figure 13 Performance Profiling………………………………………………………………..12.

Figure 14 Multiple Image Initialization…………………………………………………………13.

Figure 15 Final Render………………………………………………………………………….14.

**Introduction**

The following report explores the concepts and demonstration of procedural terrain generation, focusing on the challenges encountered during the development and areas implemented. The principles are aimed to produce a randomized terrain with elements such as sandy beaches, snowy mountains, blending along each section and animated water depending on the height of the terrain, which is flexible to each instance of the generation.

**Procedural Generation (Textures/Animation)**

The engine consists of two types of procedurally generated elements such as textured terrain and animated water. First and foremost, the engine generates a flat terrain *figure 1*, from which it is applied with a diamond-square noise algorithm *A1essandro. (2021, August 30.)*, modifying the values of each vertex on a triangle that produces a randomly generated field based on its values *figure 2*.

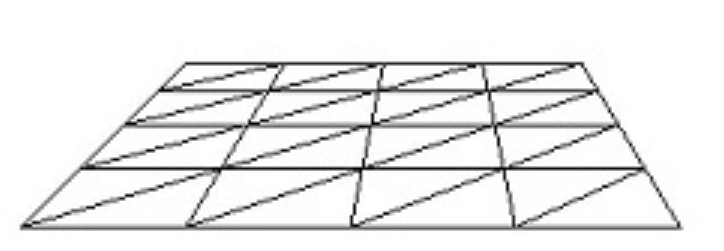


Figure Procedural Plane

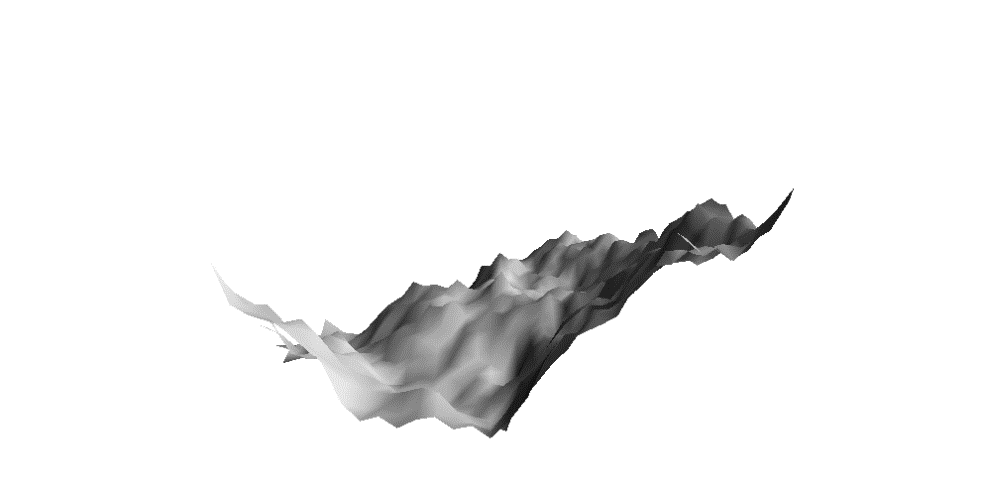


Figure Modified Noise Algorithm Terrain

The diamond square algorithm process first initializes the corners of the terrain represented in red *figure 3* and finds the center from the four corners *figure 4*, which then locates new corner vertices and divides them into further smaller squares *figure 5* applying an offset and an average value of the four corners to the height of the center. The steps are repeated in *figure 6* until the effect desired is achieved, the higher the values more iteration and the better results it produces.

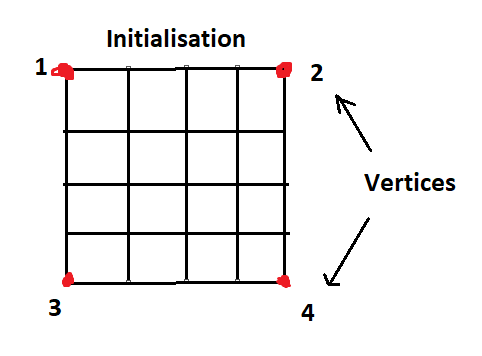


Figure Initialization find four Corners

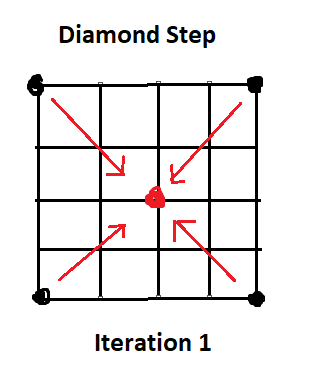


Figure Diamond Step Iteration one

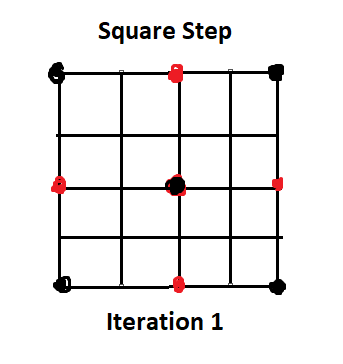


Figure Square Step one

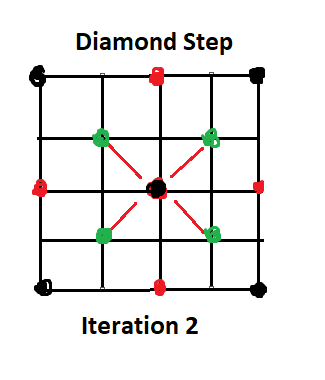


Figure Diamond Step Interaction two

To create a textured-based map on the terrain *figure 7*, we need to retrieve values for each vertex point, which determines the position of the vertex point to apply the appropriate textures, such as sandy beaches, grassy lands and mountain ranges with snow.

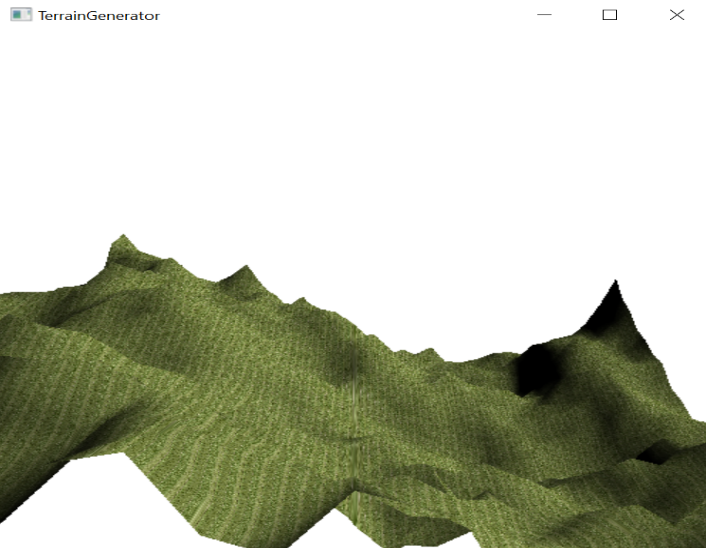


Figure Textured Terrain (Grass)

To apply these textures, we first initialize values using the BMP image loading library, which allows us to read and store the information of each image in memory. These values are then shared with vertex and fragment shaders, which retrieve and apply appropriate texture values based on the height of each vertex point within the terrain. By the use of this method, it is possible to create a dynamic and textured landscape that includes a blending factor between each area of the map such seen in *figure 8*, *LearnOpenGL. (n.d.).*

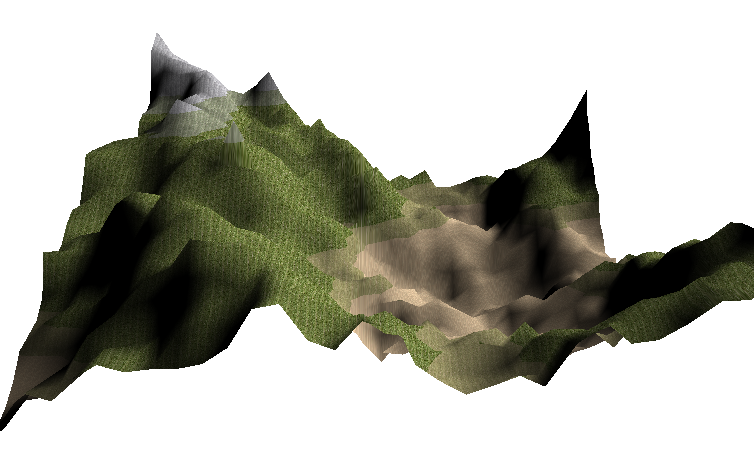


Figure Multi-Textured Terrain

The factor is achieved by defining checks on the height, by establishing the basis of a “Down-range” & “Up-range”, which define the areas where a texture must begin, and end as seen in *figure 9*.

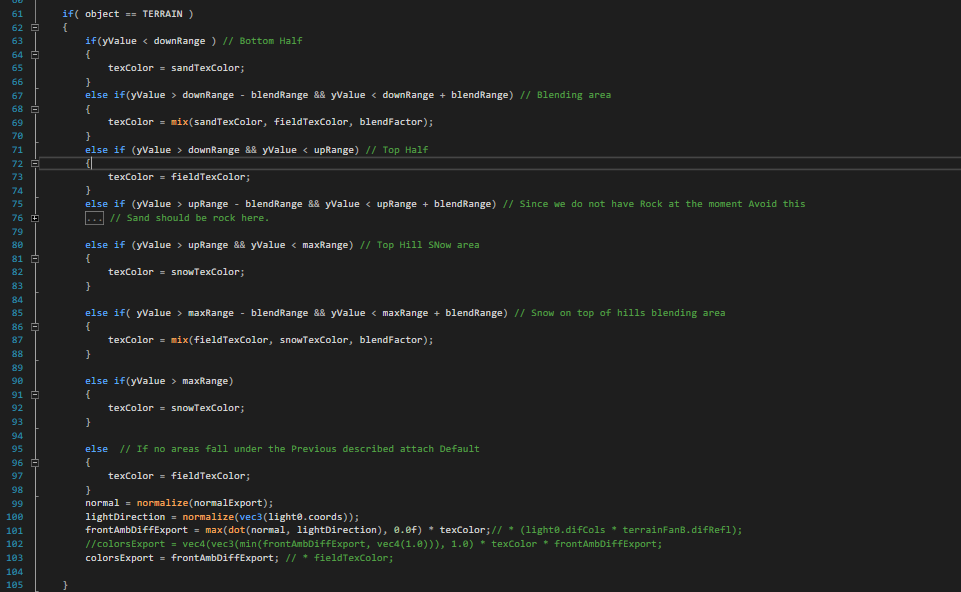


Figure Multi-Textured Terrain Modifiers

The implementation was simple, though areas of initializing textures remained difficult initially they only produced a pure black area, which was a combination of poor storing of the image values and applying them in shaders. Due to the scale and size, it became significantly harder to de-bug the lines of code as they kept increasing in value and were used across a multitude of areas where the problem could be taking place, *LightHouse3D. (n.d).*

Similarly, the process for generating procedural water starts by creating a simple plane as shown in *figure 1*. However, the approach differs from the terrain by instead of applying a noise calculation, instead, we modify the values on a basis of rows and their movement to adjust the height data for each section creating an animated effect. The animation is achieved using a vertex shader in *figure 10*, modifying the y-axis and z-axis by applying the waveTime modifier. Which controls the speed of the waves generated in the water, producing more of a more realistic and dynamic terrain to explore.

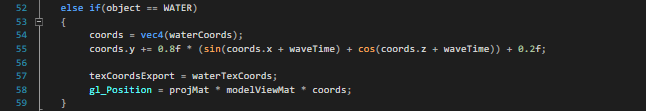


Figure Water Animation Modifiers

To further improve the water *figure 11*, we could use the same approach made to apply textures on the terrain, by identifying the height of water and determining the maximum and minimum range factors. Would allow the use of bubbles or differences in colour at the top and bottom areas of the waves, producing a significant enhancement effect and giving characteristics to its procedural design.

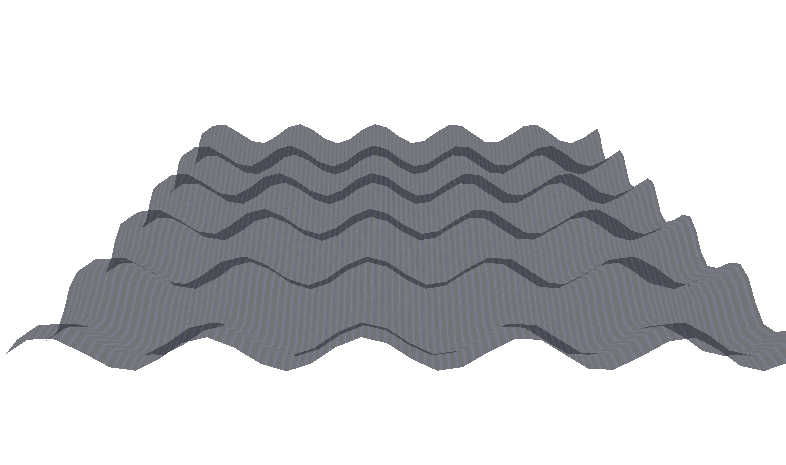


Figure Animated Water Basics

**Camera**

Implementing a 3D camera for terrain exploration was difficult due to lack of understanding in regard to the provided examples regarding the use of perspective, projection and model matrices functionality *figure 12*. Changing values often disabled part of the textures or removed them.

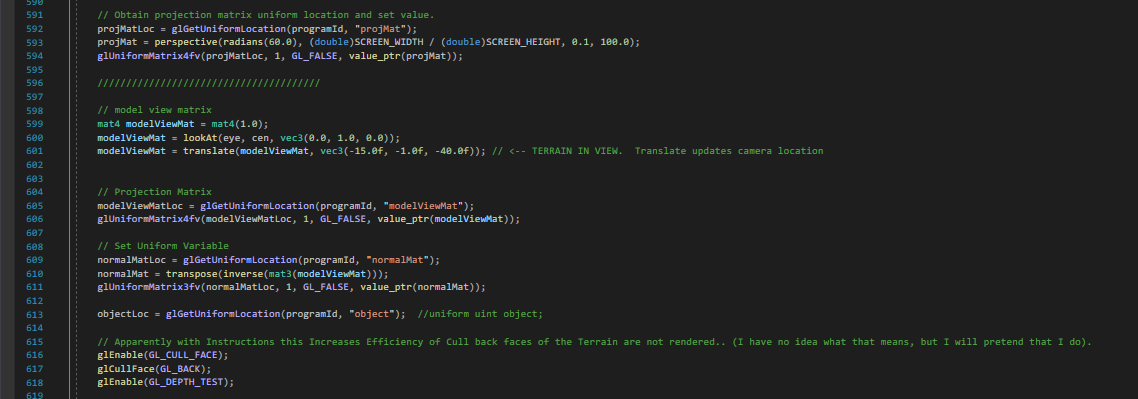


Figure Camera Initialization

Despite the difficulties in the implementation, it was successfully added through trial and error until the desired outcome was achieved to represent the use of the keyboard movement around the scene *Dey, S. (2019).*

**Optimization & Performance**

The run-time of the application is quite significant using up to 75 (MB) stored for the process having a significant impact on the CPU usage as a factor in *figure 13*.

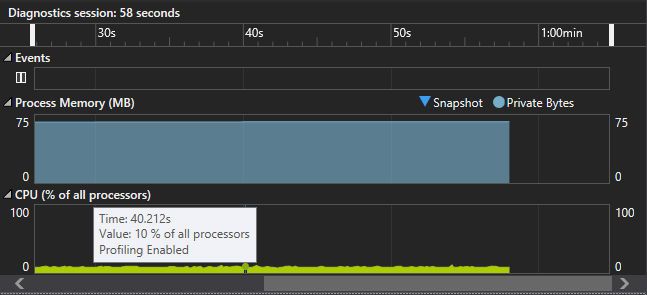


Figure Performance Profiling

The use of classes would significantly divide the code into smaller chunks, which in return would simplify de-bugging and navigability as profiling shows that multiple factors are contributing to the application speed. Improvement can be made in *Figure 14* separating the functions to represent each one, adding values automatically avoiding repetitive commands.

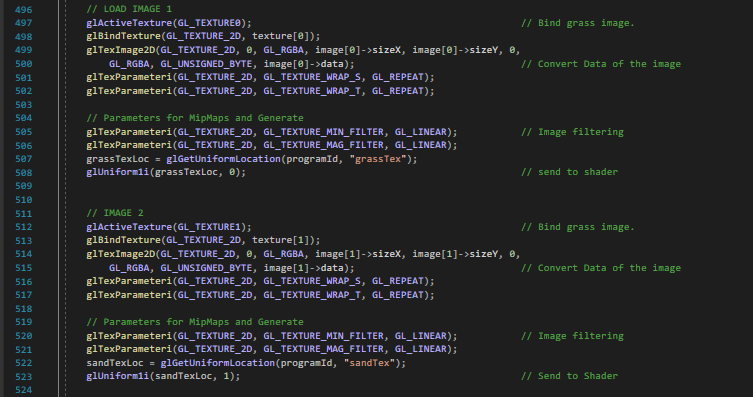


Figure Multiple Image Initialization

**Application Final Output**

Combining the GLM library, BMP library, use of multi-textures, procedurally generated terrain and animated water produces the following *figure 15*.

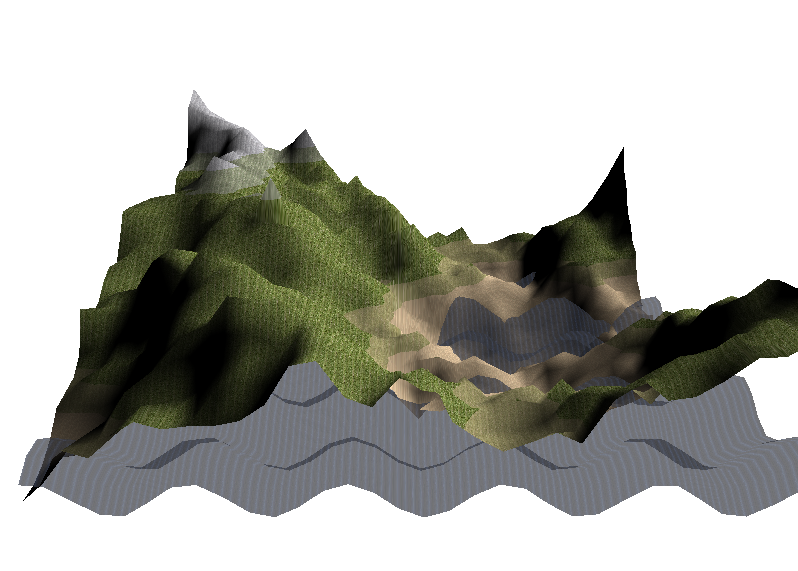


Figure Final Render

**Reflection**

During the development, the difficult part was understanding the integration of using shaders, which served various issues in multiple areas. Affected areas: implementing a background skybox and new textures, which the examples required a specific way to set up a camera or extra functionality. In case there was a difference in a value the entire functionality stopped working, which was hard to comprehend, though changing values and applying them in areas ultimately solved these issues. In the future, I would instead consider the using external material as the task lacked necessary guidance and was difficult to learn. If I take anything away from the experience, it was a useful preparation for industry due to ambiguous and misleading requirements.

|**List of References**

1. *A1essandro. (2021, August 30.). Diamond-And-Square.* <https://github.com/A1essandro/Diamond-And-Square>
2. *Dey, S. (2019). Using GLM for Mathematics. In Learn OpenGL: Create great graphics with OpenGL, a powerful cross-platform graphics library (pp. 11-12).* [*https://subscription.packtpub.com/book/game-development/9781789342253/1/ch01lvl1sec12/using-glm-for-mathematics*](https://subscription.packtpub.com/book/game-development/9781789342253/1/ch01lvl1sec12/using-glm-for-mathematics)
3. *LearnOpenGL. (n.d.). Blending. LearnOpenGL.com.* <http://learnopengl.com/Advanced-OpenGL/Blending>
4. *LightHouse3D. (n.d). Terrain rendering: Algorithms and source code* [www.lighthouse3d.com/opengl/terrain/index.php?mpd2](http://www.lighthouse3d.com/opengl/terrain/index.php?mpd2)