Terrain Generation and Optimisation
An Exploration of Complexity

Marshall Sharp

Falmouth University Games Academy

Abstract

On this poster, we take a look at OpenGl, how to generate a model utilising it, and how to apply this knowledge to a Unity project to create and optimise a scalable size mesh and water/wave system. We also take a look at external optimisation tools with useage of NVIDIA Nsight. Overall, the project runs at 144fps when running the least or default indicies and vertcies, to the lowest of 55 FPS when running the maximum amount.

Introduction

Procedual Terrain Generation (PTG) has existed within the games industry since the turn of the 21st Century[3], and the creation of the graphics card and, subsequently, Graphics programming, has opened up unlimited possibilities for career pathways such as Engine Development, avenues of technical art, and similar Avenues utilising graphics programming - The most common languages for graphics programming being OpenGL and Vulkan, both come with benifits and drawbacks. This Poster will take a look at OpenGL, understanding it's basics and a look into how you can use a method to generate terrain in Unity, an Engine that is capable of using OpenGL to run it's in-engine visuals.

Main Objectives

1. Cover the basics and logic behind OpenGL Model and Mesh generation.
2. Explore and research how to generate a mesh using graphics programming.
3. Create a tool for visualising terrain through a mesh.
4. Optimise the tool for lower end machines.

Materials and Methods

0.1 Terrain generation: The basics

Terrain generation works, in short, by utilising the same system as Mesh generation, which takes an array of vertex's and indicies, passing them through the VAO, VBO and EBO buffers in conjunction with the Depth buffer and rendering them on the screen, along with any textures for the mesh. If you want to load an entire model, you will need to go through each mesh in the object file (FBX, OBJ or STL are common file formats for this), rendering them and applying textures Accordingly thanks to its UV. This is the basis on which almost every 3D game relies on. Figure1 is a backpack that is using a techunique involving ASSIMP (asset importer), GLFW and glad plugins/add-ons, doing what was explained here.

Figure 1: An OpenGL render, utilising code from [1]

For Terrain generation, The program will calculate each vertex and/or indicies, and draw each triangle between three of them, and repeat it flipped, where it would then make a full square. It will then loop through this for it's length and width. It will also calculate the height of each of the indicies, and place them accordingly, using a shader graph to aid in a texture gradient that is applied to the mesh. Without any textures, this is what is displayed in figure 2, which is a prototype of this generation. The basis for this is using a method from these content creators, which have been compiled into a playlist [1].

It will also detect if it should act more like sea/ocean water, where it will begin moving and displacing the triangles using a similar method it uses to how it finds the land.

Optimization tools

For optimization, what is used is NVIDIA nsight[2], which is an application for debugging C++ projects. One issue this has is you cannot debug the unity project, due to it using C#. However, what can be done is debugging the build. In order to accomplish this, the developer must create a empty C++ project, and build the game into that project. They then must select Nsight to run the games .exe file. This will cause the unity build to open with the Nsight debugger enabled.

To optimise the project, it will be done in engine utilising the Profiler tool. The tool tracks Rendering, CPU usage, Memory/RAM usage, along with FPS. it function is similar to the diagnostic tools built into Visual Studio, where it pauses/breaks the program to get CPU and memory performance. For most OpenGL projects, you will beusing the one seen in figure 2.

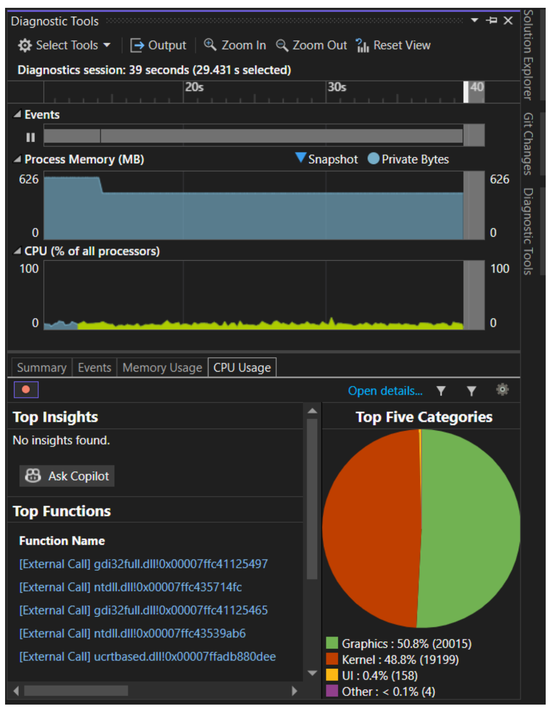

Figure 2: The Diagnostic tools within Visual studio. This is similar to the one Unity uses

Results

The tool, by default, runs at 145FPS using a 4050 laptop GPU and as seen in figure 3. The build used for the GPU optimisation goes up to 450 and reached a low of 55FPS. The tool also renders in very low poly's, and so for a cleaner terrain you would need to subdivide the generated mesh. Without this, it can lead to stretched faces and, if it has collision, impassible or hard to navigate terrain. The solution to this situation was to increment the verticies whenever it obtained the data to generate the mesh.

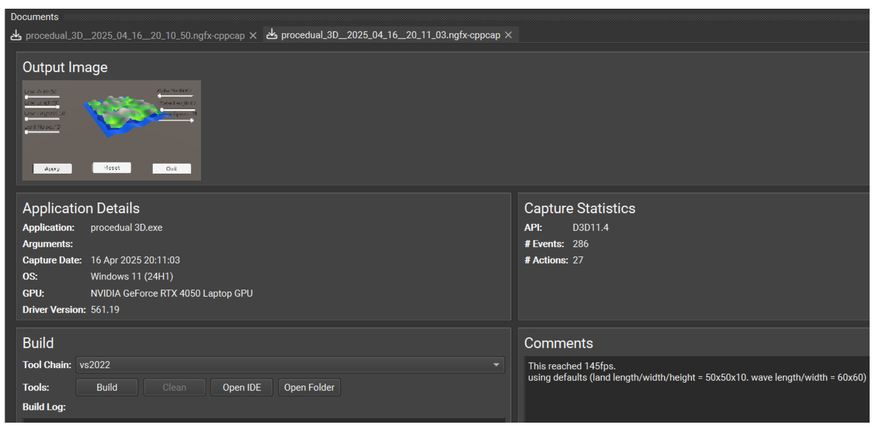

Figure 3: NSight result with the FPS noted. Using default values.

Conclusions

On this poster, background on mesh generation is given and how it can be applied to terrain, in addition, a method of optimsation on the GPU are researched and used. Further research in this field will be done with culling algorithms in a Generated City using the GL libary for Unity.

References

[1] Joey de Vries. Learnopengl.com. https://learnopengl.com/, 2015.

[2] Nvidia. Nvidia nsight. https://developer.nvidia.com/nsight-graphics, feb 12 2018.

[3] Jon Peddie. The eras of gpu development. https://blog.siggraph.org/2025/04/evolution-of-gpus.html/, 2025.

[4] Daniel Shiffman. *The Nature of Code*. No Starch Press® Inc., March 2024.

Acknowledgements

[1] https://youtube.com/playlist?list=PLNfnoxGFwMBv0gdAmP6JhydVtd30DKilU&si=KShD_Idr4KxJmqDR