# Advanced Shader Programming – Coursework Report IMAT3907

A colorful planet in space

Description automatically generated

## Introduction

This report provides a deep analysis of the design and implementation methodologies I have used to develop a terrain generation system, as per the specification of the marking scheme.

This project was driven by three core objectives, to develop a modular shader pipeline capable of rendering advanced lighting and visual effects, the use of appropriate Level of Detail (LOD) handled through shader code and the generation of an efficient and visually attractive procedurally generated terrain.

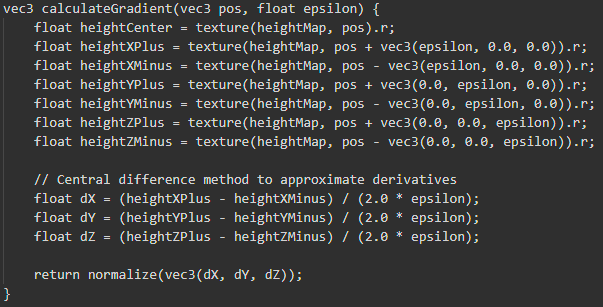
Thus, my system utilizes a robust shader pipeline, capable of handling distinct terrain features such as: The Planet, Water, Atmosphere, Moon & Clouds. To set my work apart from the norm I decided to use dedicated compute shaders to generate a cubic, subdivided mesh for the planet's terrain and another to develop a Perlin noise-based cube map for height mapping.

The shader sequence, composed of Vertex, Tessellation Control, Tessellation Evaluation, Geometry, and Fragment shaders, dynamically transforms the vertices. Notably, the terrain is sculpted through two pivotal processes: heightmap application and a mathematical model that morphs a cubic structure into a spherical one. This dual modification results in a planetary terrain that is both computationally efficient and visually striking.

[Further details explain the utility and justification of each shader stage & application are listed in their relevant sections of this report document.]

## Lighting

*Per Vertex Normals:* In preparation for my lighting calculations using TBN Matrixes In my geometry shader, I chose to use per-vertex normals which are calculated using the Central Difference Method. This technique approximates surface gradients by evaluating the rate of change across three dimensions, ensuring that the normals are of unit length for accurate light calculations. The Central Difference Method is more accurate than the first order forward or backward differences as it utilizes points on both sides of each vertex to compute gradients. Historically, the significance of this method was recognized by Lagrange fifty years after its introduction by Brook Taylor in 1715, who lauded finite difference methods as the cornerstone of differential calculus [1].



*Skybox:* As the goal of creating an entire universe and rendering it with lighting in real time is a very difficult computational task, I opted to use a Skybox to encapsulate my scene with a seamless universe like backdrop. Skyboxing is a method of encapsulating a scene with a textured cube that stores its data in a cube map, consisting of 6 seamless texture that represent the different faces of a cube. By converting the viewport’s projection matrix to a mat 3 and back to a mat 4 then projecting the position coordinates to pos.xyww the cube will be projected surrounding the camera and thus does not need a model matrix as it will always be centred around the viewpoint. To avoid future fragments being ignored due to the skybox’s proximal depth coordinates, depth mapping must be disabled for the skybox’s draw call. In 2004 Marino [2] released a book in which he explored “3D Game-based Filmmaking”, the process described in this book seems to be one of the earliest uses of a skybox / skydome. He describes his methodology in replicating expansive backdrops for films using a green screen with game engine technology rendering seamless textures into the film, instead of the costly process of hiring artists to paint and detail physical backdrops for filming, or filming in exotic locations.

*Billboarding:* The cost of rendering objects like clouds and tree’s is roughly the same as the cost of rendering my singular planet within the scene, to keep computation time down rather than spend computation time on rendering a high quality 3d model of a cloud, I opted for an advanced billboarding technique that uses multiple textures of a cloud at different elevation levels and azimuth angle, computes a local coordinate system based on the direction of the cloud from the centre of the planet, and then compares that with the elevation and azimuth angles of the cloud to the view Position. This is then used to select the appropriate texture from the texture array. The method of billboarding itself is relatively simple in contrast to this method, the billboards are simply a quad generated at the singular position given to the shader, scaled by a scale uniform that is passed from the C++ code base and the texture is oriented by taking the cross product of the Up direction and the view direction, then the up direction is recomputed based on the cross product of the Right and view Direction.

*Physically Based Rendering & Blinn Phong:*  PBR is a mathematical model proposed by Greenberg, D. P. [3] in a publication from Cornell in which he detailed the use of a mathematical model to simulate the interactions of light with surfaces that mimic real world properties using complex formulae for light reflection and absorption. This maths is based around material properties of Albedo, Roughness, metalness and AO. In contrast, the Blinn-Phong model, a simpler and less computationally demanding method, approximates specular highlights using exponentiation of dot products, which can lead to less accurate rendering under diverse lighting environments. However, after profiling the computation time, the computation of PBR with a high-fidelity model at high tessellation levels took roughly 330% as much time as the Blinn Phong and near 780% as long as a basic diffuse only Phong Lighting Model.

The Computation Times / Ram Usage (6 Different Texture Sets 4k resolution):

|  |  |  |  |
| --- | --- | --- | --- |
|  | Diffuse Only | Blinn Phong | PBR |
| Computation Time (ms) | 0.070 ms | 0.232 | 0.546 |
| RAM Usage for Textures | None required | 2.31 GB | 7.62 GB |

Without further explanation, these results are very easy to interpret, the increase in computation time and Ram usage dependent on lighting model shows a massive increase in both using PBR. Yet, PBR's main advantage is its accuracy and photorealism which are particularly beneficial in applications like film and realistic video games. However, while public opinion has not been surveyed here, my opinion as the coder is that I prefer the simplistic diffuse only version of my rendering over both Blinn Phone and PBR and while the implementation of each of these is usable in my scene, I have elected to make Diffuse only the default state of the scene. [This can be changed with the ImGui Interface on runtime]

*Image Based Lighting: -- Used in Water Shader*

## Level Of Detail (LOD)

*Subdivided Mesh In Compute Shader:*

*Dynamic Tesselation Level:*

*Distance Dependent Heuristic:*

*Future Improvements ?*

# Terrain

*Biome Distributor & Terrain Generator:*

*Biome Renderer:*

*Procedural Terrain Method:*

*Water Effect Comparison:*

*Compute Based Terrain Optimisations:*

# Innovation Beyond Course Spec

# Critical Reflection

# Bibliography

[1] Brook Taylor, 1715, Methodus Incrementorum Directa & Inversa: <https://books.google.co.uk/books?id=r-Gq9YyZYXYC&printsec=frontcover&redir_esc=y#v=onepage&q&f=false>

[2] 2004, Paul Marino, 3D Game-Based Filmmaking: The Art of Machinima

[3] Greenberg, D. P. (n.d.). A framework for realistic image synthesis. *Communications of the ACM, 42*(8), 44–53. Retrieved 4 22, 2024, from http://www.graphics.cornell.edu/pubs/1997/GTS+97.pdf