IMAT 3907 Advanced Shader Programming

Coursework Report

By Radoslaw Zajdel

P17215040

Contents

[Abstract 2](#_Toc38809637)

[Introduction 2](#_Toc38809638)

[Terrain 2](#_Toc38809639)

[Grid 2](#_Toc38809640)

[Heights 2](#_Toc38809641)

[Normals 2](#_Toc38809642)

[Colour/Texture 2](#_Toc38809643)

[Lighting 2](#_Toc38809644)

[Fog 2](#_Toc38809645)

[Level Of Detail 2](#_Toc38809646)

[TessControlShader 2](#_Toc38809647)

[TessEvaluationShader 2](#_Toc38809648)

[Shadows 2](#_Toc38809649)

[Light set-up 2](#_Toc38809650)

[Framebuffer 2](#_Toc38809651)

[Fragment Shader 3](#_Toc38809652)

[Depth Buffer 3](#_Toc38809653)

[Conclusion 3](#_Toc38809654)

# Abstract

This report is to demonstrate my understanding of what was being taught and explained within the module content, the assessment task, the code it consists of and how it all works together. It will particularly cover techniques which were required to put the code up together and achieve final results.

# Introduction

To understand how the program works, it is worth to go back to basics and remind what a shader program is. The definition says that this is a program which runs in graphic pipeline and tells the computer how to render each pixel. It is also worth to mention that by utilising the power of the GPU shader applications running on this unit are much more efficient and fluent than those using only CPU, which is usually much less powerful. The name of a shader came from the ultimate goal of this type of program, which is to control lights and shadings. [1] This is what was finally achieved in the coursework. Within this assessments there are many techniques implemented which will all be covered later on.

# Instruction

To demonstrate all functionalities of the program there are many keys which turn different effects on and off:

**Q** – turn shadows on/off

**F** – turn fog on/off

**E** – turn the grid on/off

**B** – turn the blinn-phong shading off/on

**P** – print camera position

**WASD** – move around the scene

**Left Shift (hold while moving)** – slowly increase the movement speed

**Left Ctrl (hold while moving)** – slowly decrease the movement speed

**Mouse Scroll In –** zoom in (e.g. to see tessellation)

**Mouse Scroll Out –** zoom out

# Terrain

At first stages of the development of this application, the terrain was generated with the use of height map. It gave very nice effect and allowed to use different source files but did not have much potential for improvement and colours looked flat and did not vary nicely. Within the further development stages, the procedural terrain generating technique was implemented. It is the one used by the program now and it utilizes perlin noise function and makes all calculations on GPU to maintain potential.

## Grid

The initial grid is created with the use of terrain constructor included within the Terrain class. It specifies number of grids in width, height and a grid size. Increasing first two values lead to GPU lags. Therefore values given in the constructor in the final iteration of the program have numbers as they were at the beginning and are well balanced providing nice visual effect and not sweating the GPU at the same time.

## Heights

As mentioned before, all heights are generated procedurally. All calculations are done on GPU for better performance and are in the tessEvaluationShader.tes file. The whole process utilizes various functions to return the sum of noise functions. The great advantage of perlin noise is that it can easily be used for infinite terrain generation and gives a nice variety between generated terrains.

## Normals

Normals maths are also done in tessEvaluationShader.tes. As the program makes use of perlin noise, apart from usual normalize function, those calculations need cycleOctaves function to work. Octaves are essential for correct effect as they take multiple noise functions with different parameters and adds them together.

## Colour/Texture

There were two options to add some life to the project. One was colouring the terrain dependant on height and the other was to use textures. This project utilizes the first option and multiple colours including: dark blue, blue, yellow, green, brown, gray and light blue for fog. Each colour is assigned to a different height. Colours next to each other mix with each other to give a sensation of fluent colour switch. The trickiest part of implementing the height dependant colours was to tune each smoothstep value to receive nice, fluent colour transitions.

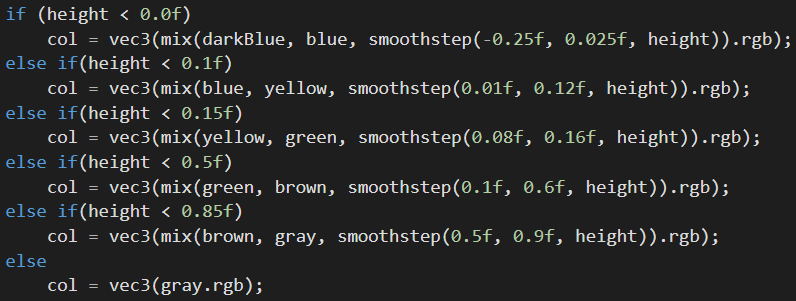


Fig. . Height dependant colours

## Lighting

There are two lightning options available on the scene. The default one is the usual phong reflection. After pressing B, it is possible to switch to much nicer are more realistic blinn-phong. Lightning calculations are performed in plainFrag.fs file. Switching between different lightning profiles is possible thanks to performing essential calculations separately and passing the lightning state by a uniform straight to the fragment shader file.

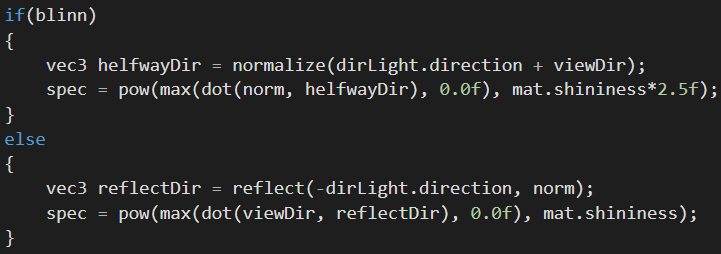


Fig. . Lightning calculations

## Fog

Creating and implementing fog was one of the easiest and most satisfying tasks to complete. This mechanic is very simple to understand and all it does is fading objects into the colour of the background usually called sky. This effect is strictly dependant on the distance of the terrain from the camera. The further the terrain is, the stronger the fade is. As for every other mechanic, this one also requires essential calculations to be performed correctly. In the code, they are placed in tessEvaluationShader.tes file and the visibility value is later passed to fragment shader.

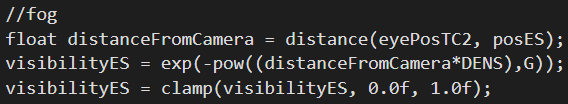


Fig. . Fog calculations

# Level Of Detail

Level Of Detail (LOD) is often called tessellation. This is incredibly powerful technique which helps with utilizing performance of GPU in best possible way. Once implemented and tuned correctly, it makes sure that the number of vertices of the object decreases as they are further away from the camera. If adjusted correctly it will be impossible to tell that this is working but the number of displayed elements will be hugely decreased, therefore saving a lot of GPU memory. Within this there are three stages, from which two are programmable: Tessellation Control Shader, Primitive Generator and Tessellation Evaluation Shader (first and third are programmable).

## TessControlShader

As the first programmable part of tessellation process, Tessellation Control Shader file uses GetTessLevel function to return the varying tessellation level parameter. This is later used to calculate the tessellation level based on the distances between camera and the vertex.

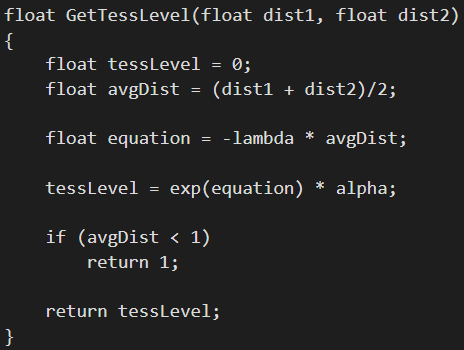


Fig. . GetTessLevel function

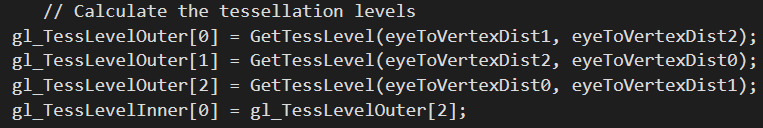


Fig. . Calculation tessellation levels

## TessEvaluationShader

This file is where all the magic happens as all transforms are performed here.

# Shadows

## Light set-up

## Framebuffer

## Fragment Shader

## Depth Buffer

# Conclusion

# Appendences

1. Omar Shehata, 15 Apr 2015, A Beginner’s Guide to Coding Graphics Shaders, gamedevelopment.tutsplus.com, https://gamedevelopment.tutsplus.com/tutorials/a-beginners-guide-to-coding-graphics-shaders--cms-23313