I confirm that the code contained in this file (other than that provided or authorised) is all my own work and has not been submitted elsewhere in fulfilment of this or any other award.

Signature – Ruari McGhee

Coursework

Graphics Programming – m3i622944

Ruari McGhee – S1432402

Table of Contents

[Main Game 2](#_Toc39888459)

[Class Description 2](#_Toc39888460)

[Class Behaviour 2](#_Toc39888461)

[Setting Up Game 2](#_Toc39888462)

[Running Game 2](#_Toc39888463)

# Background

For this project’s third shader the aim is to experiment with implementing raymarching into the scene.

Raymarching works differently to how the other objects in the scene are being handled. Instead of taking the object’s

## Compute Shaders

Unlike the other shaders used in the labs and the other coursework shaders, Compute shaders are not part of the regular shader pipeline

Due to this there is a lot of freedom in what they can do

This means that instead of each pixel being calculated one at a time the GPU can handle the calculations in parallel.

## Compute Shader Setup

### Texture

Instead of rendering a model to the screen the compute shader instead calculates if a primitive

### Mesh

# RayMarching Shader

## Raycasting

– Note – Originally the FOV given to the camera is 70 degrees but for OpenGL to correctly initialise the perspective projection for the camera this must be converted into radians.

## Distance Functions

## Lighting

# Rendering

Since the output is saved as a texture this can be rendered to the screen on a simple plane in front of the camera.

The plane used has its vertexes resemble the coordinate system that OpenGL uses for its window, the top right corner is at (1, 1) and the bottom left corner is at (-1, -1).

Because of this no transformation needs to take place in a vertex shader and the fragment shader only needs to