# Computer Graphics Coursework – Self Assessment Document

**Name:** *Shane Abrihan* **ID number:** 22480889

Complete the self-assessment grid below by writing a short explanation of how you have satisfied the requirement and how it has implemented in your code.

|  |  |  |
| --- | --- | --- |
| **Learning outcome** | **Mark** | **Weighted mark** |
| 1. Use appropriate mathematical tools (40%) |  | 0 |
| 2. Develop a 3D graphics application (30%) |  | 0 |
| 3. Write shader code (30%) |  | 0 |
|  | Total | 0 |

Your mark for each Learning Outcome (LO) is the highest mark achieved based on the criteria specified in the self-assessment grid. Note that you will need to have satisfied all criteria at the lower mark bands to be awarded marks in the higher mark bands, e.g., to get a mark in the 70 - 80 band for a learning outcome you will have needed to have satisfied all criteria in the 40 – 50 and 50 – 60 mark bands.

## Learning Outcomes:

**LO1** Select and use appropriate mathematical tools for constructing and manipulating geometry in 3D space.

**LO2** Develop an interactive 3D graphics application using an industry-standard API.

**LO3** Write shader code for the programmable pipeline on modern graphics hardware using an industry standard shader language.

## Self-assessment Grid

|  |  |  |
| --- | --- | --- |
| **Mark** | **Criterion** | **Comments (state how and where you have achieved the criterion)** |
| 42, 45, 48 | LO1: Basic use of vector and matrix objects | I have used glm::vec3 to define the positions of the cube objects that I use. Matrix operations have been handled with the use of glm::mat4, which is used for model, view and projection transformations. |
| LO2: Application compiles and runs without alterations to the source code of CMake file. | Paste a screenshot of your application below |
| LO3: Implementation of shaders to apply appropriate textures to objects. | To apply textures to my cube object, I have used the fragment and vertex shader. The main texture that I have used, which is the Minecraft dirt block, is loaded by using stb\_image and mapped to the cube by using UV coordinates. |
| 52, 55, 58 | LO1: Basic use of translation, rotation and scaling transformations. | I have used the translation and scaling matrices in the render loop to define the cubes position and its size. This is evident in the loop that loops through the object vector. I have opted not to use rotation transformations as it wouldn’t have made sense for a Minecraft themed island to be rotated. |
| LO1: Implementation of glm library functions for calculating view and projection matrices. | I have used glm library functions to calculate the view and projection matrices within the camera.calculateMatrices() function. I have used glm:lookAt() to calculate the view matrix based on the position and direction of the camera. I have used glm::perspective() to calculate the projection matrix based on the aspect ratio and the FOV. (field of view). |
| LO2: 3D virtual world has been created using instances of a single object type. | I have created my 3D world using multiple instances of a cube object. This is done by: setting the positions of each cube, adding the cubes to the object vector, and finally in the render loop, it loops through the object vector and draws each cube. |
| LO3: Use of shaders to apply dynamic lighting from point light sources |  |
| 62, 65, 68 | LO1: Implementation of students own functions for calculating view and projection matrices. |  |
| LO2: 3D world created using multiple object types. |  |
| LO2: Users can navigate the virtual world using keyboard and mouse inputs. | These inputs have been implemented with the use of the camera class. The window detects inputs by using the glfwGetKey and glfwGetCursorPos functions. |
| LO3: Use of shaders to apply dynamic lighting from different types of light sources. |  |
| 72 75, 78 | LO1: Implementation of students own functions to replace glm functions (e.g., glm::length(), glm::dot(), glm::cross() etc.). |  |
| LO1: Implementation of quaternions to calculate rotation matrix. |  |
| LO2: Interactive dynamic aspects of the virtual word and controllable by the user (e.g., position of objects, location and function of light sources etc.). |  |
| LO3: Appropriate implementation of normal and specular maps. |  |
| 85, 90, 100 | LO1: Use of quaternions to calculate view matrix. |  |
| LO1: Use of SLERP to smooth out changes in camera direction. |  |
| LO2: Implementation of a third person camera with the ability to switch between first and third period view. |  |
| LO2: The position of the camera or character obeys the constraints of the physical space (e.g., can’t pass through objects, can’t hover in midair etc.). |  |
| LO3: Use of shaders to apply parameter driven effects within the scene, e.g., light properties controlled using camera/character position. |  |