# Computer Graphics Coursework – Self Assessment Document

**Name:** *Mason Kelly-Scholes* **ID number:** *23679754*

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 | A glm::mat4 translation matrix is created in the main function to position the textured quad. This matrix is passed as a uniform (transformation) to the vertex shader, demonstrating the use of matrix transformations in rendering. (Line 212-218) |
| LO2: Application compiles and runs without alterations to the source code of CMake file. |  |
| LO3: Implementation of shaders to apply appropriate textures to objects. | A texture (crate.jpg) is loaded and applied to a quad using vertex and fragment shaders. UV coordinates are passed to the shaders and used to sample the texture with sampler2D in the fragment shader. The final color is based on the texture, confirming correct shader-based texture mapping. (Lines 139-148 169-199) |
| 52, 55, 58 | LO1: Basic use of translation, rotation and scaling transformations. | A glm::mat4 translation matrix, rotation matrix, and scaling matrix are created in the main function to transform the textured quad. These matrices are combined into a single transformation matrix and passed as a uniform to the vertex shader, demonstrating the use of basic translation, rotation, and scaling transformations in OpenGL. (Lines 216-236) |
| LO1: Implementation of glm library functions for calculating view and projection matrices. | The program uses the Camera class to calculate both the view matrix (via glm::lookAt) and the projection matrix (via glm::perspective). These matrices are used to construct the MVP matrix passed to the vertex shader, demonstrating the use of glm library functions for camera and perspective setup. |
| LO2: 3D virtual world has been created using instances of a single object type. | A 3D virtual world is created by instantiating multiple objects of a single type (Object struct representing cubes) with different positions, rotations, scales, and angles. These instances are stored in a vector and rendered individually in the scene, demonstrating the use of multiple instances of one object type to build the virtual world. |
| LO3: Use of shaders to apply dynamic lighting from point light sources | The program implements dynamic lighting by passing point light properties—such as position and color—to the fragment shader as uniforms. The shader calculates ambient, diffuse, and specular reflections using the Phong lighting model, with lighting computations performed per fragment to create realistic lighting effects that respond to the light source position in view space. |
| 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. |  |
| 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. |  |