# Computer Graphics Coursework – Self Assessment Document

**Name: Zulfaqaar Wadiwala** **ID number: 22528471**

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%) |  | 23.2 |
| 2. Develop a 3D graphics application (30%) |  | 20.4 |
| 3. Write shader code (30%) |  | 20.4 |
|  | Total | 64 |

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 used basic vector and matrix operations throughout the code to build the 3D space, mainly using GLM vectors and matrices for positioning, transforming, and rotating objects. For example, I used glm::vec3 for positions models (e.g. coursework.cpp line 173-189). |
| LO2: Application compiles and runs without alterations to the source code of CMake file. | The application runs without requiring any changes to the provided GitHub repos code/assets and CMake files. Only CMake needs to be used to build the project. |
| LO3: Implementation of shaders to apply appropriate textures to objects. | I used shaders to apply textures onto the models in the scene. For example, in coursework.cpp (lines 214-217), I assigned diffuse, normal, and specular textures to the walls, which has also been done for the teapot, crates and floors. These textures are handled by fragmentShader.glsl which blends lighting and texture. |
| 52, 55, 58 | LO1: Basic use of translation, rotation and scaling transformations. | I applied translation, rotation and scaling transformations when positioning and orienting objects in the scene. This can be seen in coursework.cpp (lines 155-162) where I used the position, rotation, scale and angle transformations. |
| LO1: Implementation of glm library functions for calculating view and projection matrices. | Camera::calculateMatrices() (camera.cpp lines 9–19). The view matrix uses glm::lookAt based on the camera’s position and direction, and the projection matrix uses glm::perspective to simulate the camera’s perspective view. |
| LO2: 3D virtual world has been created using instances of a single object type. | I have used the teapot object and positioned them using different Object instances. This can be seen in coursework.cpp (lines 135-162). The model is applied at line 102. |
| LO3: Use of shaders to apply dynamic lighting from point light sources | I used shaders to apply dynamic lighting effects using point lights. In coursework.cpp (lines 116–133), I created two point light sources with attenuation values. These were passed into the shader, and the lighting effect is handled dynamically in fragmentShader.glsl (lines 39-40, 79-109). |
| 62, 65, 68 | LO1: Implementation of students own functions for calculating view and projection matrices. | N/A |
| LO2: 3D world created using multiple object types. | As well as the teapots, the scene also features other objects such as crates and different types of wall planes. These are all created using different models (coursework.cpp lines 164-297) . |
| LO2: Users can navigate the virtual world using keyboard and mouse inputs. | The user can explore the world by moving with W, A, S, D keys and rotating the camera using the mouse. This functionality is handled in coursework.cpp from lines 370-387 for keyboard and 389-402 for mouse input. The matrices and calculations are handled by camera.hpp and camera.cpp. |
| LO3: Use of shaders to apply dynamic lighting from different types of light sources. | In addition to the point lights, the fragment shader (fragmentShader.glsl lines 42-43, 111-149) handles dynamic lighting for spotlights. This light is created in coursework.cpp (lines 126-130) with direction and attenuation values and is processed using the lightSources.toShader() function during each frame. |
| 72 75, 78 | LO1: Implementation of students own functions to replace glm functions (e.g., glm::length(), glm::dot(), glm::cross() etc.). | N/A |
| LO1: Implementation of quaternions to calculate rotation matrix. | N/A |
| 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.). | N/A |
| LO3: Appropriate implementation of normal and specular maps. |  |
| 85, 90, 100 | LO1: Use of quaternions to calculate view matrix. | N/A |
| LO1: Use of SLERP to smooth out changes in camera direction. | N/A |
| LO2: Implementation of a third person camera with the ability to switch between first and third period view. | N/A |
| 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.). | N/A |
| LO3: Use of shaders to apply parameter driven effects within the scene, e.g., light properties controlled using camera/character position. | N/A |