# Computer Graphics Coursework – Self Assessment Document

**Name:** *Jack* **ID number:** *23635926*

**Please walk towards the teapot in the centre of the scene, press the 1,2,3,4,5 keys for fun effects. Spacebar to jump and E to shoot once you have your weapon.**

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 | Use of glm::vec3 and glm::mat4 objects through the project. E.g camera position and camera view matrix. |
| LO2: Application compiles and runs without alterations to the source code of CMake file. |  |
| LO3: Implementation of shaders to apply appropriate textures to objects. | Vertex shader and fragment shader are in the project files, data is sent to them in the model and light classes. |
| 52, 55, 58 | LO1: Basic use of translation, rotation and scaling transformations. | In the object loop, all three translations are used to calculate the matrices for each object and combined into the model matrix. |
| LO1: Implementation of glm library functions for calculating view and projection matrices. | The camera class contains the “calculateMatrices” method which calculates the view and perspective matrices. |
| LO2: 3D virtual world has been created using instances of a single object type. | The teapot 3D model is drawn in the scene. |
| LO3: Use of shaders to apply dynamic lighting from point light sources | There is a point light source present in the scene rendered by sending data to the shaders in the light class. |
| 62, 65, 68 | LO1: Implementation of students own functions for calculating view and projection matrices. | The camera class contains the methods “calculateView” and “calculatePerspective” which are the replacements for glm::lookAt and glm::perspective to calculate the view and perspective matrices and are used in the “calculateMatrices” method. |
| LO2: 3D world created using multiple object types. | There is both the teapot object and the wall objects present in the scene. |
| LO2: Users can navigate the virtual world using keyboard and mouse inputs. | The W,A,S,D keys can be used to move the camera and the mouse can be used to look around the scene. |
| LO3: Use of shaders to apply dynamic lighting from different types of light sources. | The lightSources object contains both the spotlight and a directional light. The directional light source can be enabled by pressing the Q button |
| 72 75, 78 | LO1: Implementation of students own functions to replace glm functions (e.g., glm::length(), glm::dot(), glm::cross() etc.). | The maths class contain the methods to replace ant glm functions used including length, dot and cross along with others. They are called in place of the glm functions whenever they are needed |
| LO1: Implementation of quaternions to calculate rotation matrix. | The quaternion class within the maths files contains the “matrix” method which calculates the rotation matrix, this is used in the Maths class’s “rotate” method. |
| 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.). | The player can walk towards the teapot in the centre, picking it up when they collide with it. The teapot is then held at the player’s side and the spotlight is modified to become a flashlight. |
| LO3: Appropriate implementation of normal and specular maps. | Each of the models has a normal and specular map applied to them which is interpreted by the model class and shown on the model in the scene. |
| 85, 90, 100 | LO1: Use of quaternions to calculate view matrix. | In the camera class’s “quaternionCamera” function, the camera’s view matrix is calculated by using the quaternion orientation’s matrix function |
| LO1: Use of SLERP to smooth out changes in camera direction. | The maths class contains the SLERP function which is used to modify the orientation matrix before it is sue to calculate the view matrix |
| LO2: Implementation of a third person camera with the ability to switch between first and third period view. | Pressing the enter key will swap between first third person which is one by translating the view matrix. |
| 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.). | The camera (if in first person) and the player model (if in third person) are able to collide with objects (the teapot in the centre) and the walls. |
| LO3: Use of shaders to apply parameter driven effects within the scene, e.g., light properties controlled using camera/character position. | Pressing the 1,2,3,4,5 keys changes the spotlight colour. |