# Computer Graphics Coursework – Self Assessment Document

**Name:** *Joshua Jackson* **ID number:** *22512619*

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 | The use of vectors can be seen in my cube position array, in the coursework.cpp file, where they are used to give the positions of where I want the crates to be in the 3D world space. |
| 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. | The implementation of shaders to apply textures to objects can be seen in the coursework.cpp file when I load textures on to the cube, the wall and the floor models. |
| 52, 55, 58 | LO1: Basic use of translation, rotation and scaling transformations. | Use of translation rotation and scaling transformations can be seen in coursework.cpp file when I add objects to the objects vector, such as the multiple cubes, the walls and the floor and ceiling. |
| LO1: Implementation of glm library functions for calculating view and projection matrices. | Use of glm library functions can be seen in the camera.cpp file where the lookAt() and perspective() functions are used to calulate the view and projection matrices for the camera. |
| LO2: 3D virtual world has been created using instances of a single object type. | The cube object type is used multiple times in the coursework.cpp file to create a 3D virtual world. |
| LO3: Use of shaders to apply dynamic lighting from point light sources | Point light sources can be seen used in the coursework.cpp file to apply dynamic lighting to the scene. |
| 62, 65, 68 | LO1: Implementation of students own functions for calculating view and projection matrices. | My own lookAt and perspective functions were made in the maths.cpp file and used in the camera.cpp file for the calculateMatrices function. |
| LO2: 3D world created using multiple object types. | Multiple object types such as the cube, and the plane can be seen used in the coursework.cpp file, even used in multiple instances. |
| LO2: Users can navigate the virtual world using keyboard and mouse inputs. | The player can use keyboard inputs and control the camera using the mouse using the code at the very bottom of the coursework.cpp file |
| LO3: Use of shaders to apply dynamic lighting from different types of light sources. | I used both pointLight sources and spotLight sources in my code to apply multiple types of light sources to the world space. |
| 72 75, 78 | LO1: Implementation of students own functions to replace glm functions (e.g., glm::length(), glm::dot(), glm::cross() etc.). | I have created my own dot and cross functions in the maths.cpp file which are used by my own lookAt function |
| LO1: Implementation of quaternions to calculate rotation matrix. | I have used quaternions to calculate my rotation matrix in the maths.cpp file under the Quaternion::matrix() function |
| 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.). | As seen at the bottom of coursework.cpp, when the user presses the 1 key, all objects on screen with the name cube will rotate. |
| LO3: Appropriate implementation of normal and specular maps. | Normal and specular maps are used multiple times for many different objects such as the cube, wall and floor objects in the coursework.cpp file |
| 85, 90, 100 | LO1: Use of quaternions to calculate view matrix. | The use of quaternions to calculate the view matrix can be seen in the camera.cpp file when orientation.matrix() is called, matrix being a quaternion rotate function in the maths.cpp file. |
| LO1: Use of SLERP to smooth out changes in camera direction. | SLERP is used in the camera.cpp file to smooth changes in camera direction under the quaternionCamera() function. |
| 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. |  |