# Computer Graphics Coursework – Self Assessment Document

**Name:** *Jake Henderson* **ID number:** *23626171*

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 implemented the basic vectors and matrices such as glm::vec3 and glm::mat4, for example for when define the positions of the room at lines 303 to 375 in coursework.cpp and when the model matrix is calculated at line 430 in coursework.cpp. |
| LO2: Application compiles and runs without alterations to the source code of CMake file. | Screenshot of the application running is located below the table. |
| LO3: Implementation of shaders to apply appropriate textures to objects. | Multiple different textures have been applied to multiple objects as seen at lines 128 to 212 in coursework.cpp. |
| 52, 55, 58 | LO1: Basic use of translation, rotation and scaling transformations. | Different objects have been rotated, positioned and scaled in different ways to fit the build, this can be seen at lines 343 to 384 in coursework.cpp. |
| LO1: Implementation of glm library functions for calculating view and projection matrices. | The glm functions for the calculating view and projection matrices have been replaced by my own math functions, an example can be seen here for when a test was done at line 388 in coursework.cpp. |
| LO2: 3D virtual world has been created using instances of a single object type. | See Below. |
| LO3: Use of shaders to apply dynamic lighting from point light sources | I have added a point lights which appears once all the objects have collected, once touched, from which it switches between the colours red and blue. This can be seen at line 522 in coursework.cpp and any calculations for the shaders were performed in lightVertexShader.glsl and lightFragmentShader.glsl. |
| 62, 65, 68 | LO1: Implementation of students own functions for calculating view and projection matrices. | I implemented my own LookAt() and Perspective() functions at lines 61 to 113 in Maths.cpp and lines 32 to 33 in Maths.hpp. |
| LO2: 3D world created using multiple object types. | Multiples objects such as cube.obj, plane.obj and sphere.obj were used in the 3D world. This can be seen at lines 111 to 124 in coursework.cpp and these assets can be found in the asset folder. |
| LO2: Users can navigate the virtual world using keyboard and mouse inputs. | The procedure keyboardInput() and mouseInput() are called which allows the user to freely move and rotate the camera without changing the y positions unless jumping, this is to allow the user to feel like they are walking around the level and not just flying. These can be found at lines 572 to 637 in coursework.cpp. |
| LO3: Use of shaders to apply dynamic lighting from different types of light sources. | I use the point light as well as the spotlights used to show the objects as well as another to spotlight which works like a flashlight. These can be found at lines 263 to 287 in coursework.cpp. |
| 72 75, 78 | LO1: Implementation of students own functions to replace glm functions (e.g., glm::length(), glm::dot(), glm::cross() etc.). | I have implemented my own functions for glm:: length(), glm::dot(), glm::cross() and glm::normalize() to replace the current functions. These can be found at lines 115 to 140 in Maths.cpp and at lines 35 to 41 in Maths.hpp. |
| LO1: Implementation of quaternions to calculate rotation matrix. | The implementation of the quaternion class can be found in Maths.cpp and Maths.hpp in which it calculates the 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.). | The user can interact with the spinning cubes by touching them, in which the objects will disappear and the spotlight above will go red. I also have a flashlight which the user can turn on and off by pressing ‘f’. These can be found at lines 459 to 480 for the spotlights in coursework.cpp and at lines 114 to 117 in Lights.cpp. The flashlight can be found at lines 590 to 604 in coursework.cpp and at lines 52 to 59 in Lights.cpp. |
| LO3: Appropriate implementation of normal and specular maps. | I have added normal and specular maps to objects in the build, this can be found at lines 188 to 212 in coursework.cpp. |
| 85, 90, 100 | LO1: Use of quaternions to calculate view matrix. | Quaternions are used to calculate the correct orientation by use SLERP which is then used to calculate the view matrix. This can be found at lines 38 to 41 in camera.cpp. |
| LO1: Use of SLERP to smooth out changes in camera direction. | SLERP has been added to work out the orientation, the SLERP function can found at lines 33 to 59 in Maths.cpp. |
| LO2: Implementation of a third person camera with the ability to switch between first and third period view. | Not implemented. |
| 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.). | Not implemented. |
| LO3: Use of shaders to apply parameter driven effects within the scene, e.g., light properties controlled using camera/character position. | When a player gets near the spinning objects, the spotlight turns yellow and once the players far enough away it goes back to yellow. This can be found at lines 480 to 498 in coursework.cpp and at lines 130 to 138 in Lights.cpp. |

A screenshot of a video game

AI-generated content may be incorrect.Here is the screenshot for “LO2: Application compiles and runs without alterations to the source code of CMake file”: