# Computer Graphics Coursework – Self Assessment Document

**Name:** *Archie Murray* **ID number:** *23684612*

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%) | 90 | 36 |
| 2. Develop a 3D graphics application (30%) | 85 | 25.5 |
| 3. Write shader code (30%) | 100 | 30 |
|  | Total | 91.5 |

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 MVP model in rendering |
| LO2: Application compiles and runs without alterations to the source code of CMake file. | CMake was altered to include freetype2 library for loading ttf files into texture data (usage is equivalent to stb\_image where it is only loading font data and is then manually loaded to the GPU through my own code) |
| LO3: Implementation of shaders to apply appropriate textures to objects. | Lighting shaders Implemented along with custom text shader |
| 52, 55, 58 | LO1: Basic use of translation, rotation and scaling transformations. | Implementation of Object class modelMat() function (common/object.cpp:4) |
| LO1: Implementation of glm library functions for calculating view and projection matrices. | common/camera.cpp:23 (own implementation) |
| LO2: 3D virtual world has been created using instances of a single object type. | Multiple instances but see source/coursework.cpp:249 |
| LO3: Use of shaders to apply dynamic lighting from point light sources | source/coursework.cpp:196 + source/fragmentShader.glsl + source/vertexShader.glsl |
| 62, 65, 68 | LO1: Implementation of students own functions for calculating view and projection matrices. | Maths::perpsective + ortho + lookAt (common/maths.cpp) |
| LO2: 3D world created using multiple object types. | source/coursework.cpp:249  Different models for crate, floating tux, player (teapot), walls, collision debug render (Press T to toggle) |
| LO2: Users can navigate the virtual world using keyboard and mouse inputs. | source/coursework.cpp:340 Mouse Look  source/coursework.cpp:346 Movement  Input is gathered into mouseDelta and movementInput vectors and then processed at above mentioned positions as camera system means it is somewhat dependent on game state |
| LO3: Use of shaders to apply dynamic lighting from different types of light sources. | source/coursework.cpp:194 All 3 types of lights implemented – different colours to help differentiate |
| 72 75, 78 | LO1: Implementation of students own functions to replace glm functions (e.g., glm::length(), glm::dot(), glm::cross() etc.). | common/maths.cpp Implemented cross, magnitude + sqrMagnitude, dot, normalise – in some cases when used in collision code 2d variants have been made too |
| LO1: Implementation of quaternions to calculate rotation matrix. | common/maths.cpp:39  Quaternions have a matrix() member that returns a glm::mat4 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.). | Player can use T to toggle between rendering collider debug, Q + E toggle between CCTV-style cameras in the corners of space, player can also use Up + Down arrow keys to increase the speed of a moving object when nearby |
| LO3: Appropriate implementation of normal and specular maps. | Normal and specular highlighting data is calculated in Model class during calculateNormals() and draw() – used in shader when calculating lighting |
| 85, 90, 100 | LO1: Use of quaternions to calculate view matrix. | common/camera.cpp:23  common/maths.cpp:39 Adapted to use deltaTime to ensure consistent behaviour during frame drops |
| LO1: Use of SLERP to smooth out changes in camera direction. | common/camera.cpp:25  common/maths.cpp:69 adapted to use deltaTime to ensure consistent behaviour during frame drops |
| LO2: Implementation of a third person camera with the ability to switch between first and third period view. | source/coursework.cpp:574 (camera keys)  source/coursework.cpp:365 (first person) Q + E cycle forwards and backwards between 4 CCTV-style cameras in the corners and an FPS camera. Player model is only rendered using FPS camera rotation + position when a CCTV camera is active |
| 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.). | Basic 2D collision is implemented using the dot product of the velocity normal and the direction of collision from player to closest point on collider when above a minimum value player movement is restricted |
| LO3: Use of shaders to apply parameter driven effects within the scene, e.g., light properties controlled using camera/character position. | Model tint is implemented to have classic HSL rainbow on the floating tux, similarly each camera has a custom tint value (initially blue, green, red, yellow). Player can control diffuse parameters on moving teapot using UIOP to increment and HJKL to decrement |