# Computer Graphics Coursework – Self Assessment Document

**Name:** Qamar Iqbal Gutierrez **ID number:** 23668160

|  |  |  |
| --- | --- | --- |
| **Learning outcome** | **Mark** | **Weighted mark** |
| 1. Use appropriate mathematical tools (40%) | 100 | 40 |
| 2. Develop a 3D graphics application (30%) | 40 | 12 |
| 3. Write shader code (30%) | 20 | 6 |
|  | Total | 58 |

## 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 it for the Camera.hpp Constructor and member variable. Also to create the camera object.  In the Camera for the Quaternion Camera  In the Main file for object structure Definition.  This is declared in the maths.cpp and .hpp |
| LO2: Application compiles and runs without alterations to the source code of CMake file. |  |
| LO3: Implementation of shaders to apply appropriate textures to objects. | Shader Loading and Texture loading tin the main file, texture.hpp and model.cpp.  Setting the texture Uniform in Shaders then binding during rendering. |
| 52, 55, 58 | LO1: Basic use of translation, rotation and scaling transformations. | Applied transformation to cubes in the render loop. Transformation functions in maths.cpp maths.hpp |
| LO1: Implementation of glm library functions for calculating view and projection matrices. | Applied in camera to calculation Matrices of view and projection and within the Camera variables I had multiple member variables such as eye, target front, yaw, pitch and so on which transforms view space coordinates.  In the Main file calling the Calculation Matrices to have functionality in complied software.  Quaternion Camera which relies on glm library as well as member variables. |
| LO2: 3D virtual world has been created using instances of a single object type. | Creation of cubes in Main file then using transformations to rotate and shift the cubes around to create a 3d space. |
| LO3: Use of shaders to apply dynamic lighting from point light sources |  |
| 62, 65, 68 | LO1: Implementation of students own functions for calculating view and projection matrices. | Quaternion camera and calculating matrices in the camera.hpp and .cpp files to have smooth mouse viewing. |
| LO2: 3D world created using multiple object types. |  |
| LO2: Users can navigate the virtual world using keyboard and mouse inputs. | In main file keyboard and mouse inputs are defined, viewing with mouse uses quaternions and moving with keyboard using WASD to navigate through the 3d world. |
| LO3: Use of shaders to apply dynamic lighting from different types of light sources. |  |
| 72 75, 78 | LO1: Implementation of students own functions to replace glm functions (e.g., glm::length(), glm::dot(), glm::cross() etc.). | In the Camera.hpp and Camera.cpp I change the calculateCameraVectors function to use quaternion yaw and pitch to calculate viewing with mouse which uses dot product, cross product and other mathematical principles. |
| LO1: Implementation of quaternions to calculate rotation matrix. | Quaternion class in maths.hpp and Quaternion functions in maths.cpp. gives a 4x4 matrix to calculate view and rotation based on the function’s mathematical formulas.  In Camera.cpp I have used a Quaternion camera function, calculating camera orientation then using SLERP then calculating view 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.). |  |
| LO3: Appropriate implementation of normal and specular maps. |  |
| 85, 90, 100 | LO1: Use of quaternions to calculate view matrix. | Used Quaternions to calculate 4x4 view matrix to calculate camera orientation using pitch, yaw and SLERP |
| LO1: Use of SLERP to smooth out changes in camera direction. | Used the line Maths: SLERP(… 0.2f) to use mathematical principle linear interpolation to smooth out mouse movements and remove the jitteriness seen without the SLERP method. This is in Camera files method. |
| 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. |  |