# Computer Graphics Coursework – Self Assessment Document

**Name:** *Emmett Coyle* **ID number:** *22491518*

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 | Vectors and matrices play a big role in handling object positions and transformations. The glm::vec3 is used to input the positions of light points and the camera.  In maths.cpp, I used functions like translate to apply a translation matrix using glm::vec3 to move objects around the scene, as well as other transformations like scale which applies non-uniform scaling based on the vector components and rotate which is made by using an axis vector and an angle . |
| LO2: Application compiles and runs without alterations to the source code of CMake file. |  |
| LO3: Implementation of shaders to apply appropriate textures to objects. | The vertexShader is used by OpenGL to deal with the vertex coordinates and calculate coordinates of the fragment shader.  The fragmentShader is where the colour of the texrure is retrieved.  In the coursework.cpp class, the shaders are called from the model,cpp class, which uses the vertexShader & fragmentShader to add the shaders to the objects. |
| 52, 55, 58 | LO1: Basic use of translation, rotation and scaling transformations. | In maths.cpp, I implemented functions for translations, rotations and scaling using the glm library. These functions allow me to work out the right transformations of any object in coursework.cpp. An example of how these functions are used can be found between lines 200-208. |
| LO1: Implementation of glm library functions for calculating view and projection matrices. | To calculate the view and projection matrices, I used the lookAt and perspective function from the glm library. In camera.cpp on line 11, I use lookAt to create the view matrix. It takes in the cameras position (eye), the point the camera is looking at (target) and a worldUp vector which points straight up.  On line 14, I used the perspective function to set up the projection matrix. It takes four inputs: fov, which is the field of view between the top and bottom planes, aspect, which is the windows width to height ratio and near & fat values which define the distances from the origin to the near and far clipping planes. |
| LO2: 3D virtual world has been created using instances of a single object type. | In coursework.cpp between lines 191-198 I created several box objects. First I called the constructor for the box model which loads the vertex and texture coordinates from an obj file. Then I added each box to the objects vector and set up their positions, rotations, scales and angles using an array. I finally drew the box in the render loop on line 259. |
| LO3: Use of shaders to apply dynamic lighting from point light sources | In coursework.cpp between lines 117-141 I added several different light sources, each with their own position, colour and attenuation values. To make adding point lights easier, I created a function in light.cpp that made the process easier to carry out than in courswork.cpp. Between lines 111-114 in coursework.cpp, I set up lighting properties to allow light to bounce and reflect off objects in the scene: ka controls ambient reflection that spreads evenly across surfaces, kd handles diffuse reflection for rough surfaces, and ks along with ns manage specular reflection, which depends on the lights source position. |
| 62, 65, 68 | LO1: Implementation of students own functions for calculating view and projection matrices. |  |
| LO2: 3D world created using multiple object types. |  |
| LO2: Users can navigate the virtual world using keyboard and mouse inputs. |  |
| 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.). |  |
| LO1: Implementation of quaternions to calculate 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.). |  |
| LO3: Appropriate implementation of normal and specular maps. |  |
| 85, 90, 100 | LO1: Use of quaternions to calculate view matrix. |  |
| LO1: Use of SLERP to smooth out changes in camera direction. |  |
| 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. |  |