# Computer Graphics Coursework – Self Assessment Document

**Name:** Haroon Jaleel **ID number:** 23728516

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 application extensively uses glm::vec3 for representing positions (e.g., camera, lights, objects) and colors, and glm::mat4 for all transformations (model, view, projection). This is foundational to the entire application, primarily visible in coursework.cpp |
| 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 fragment shader (fragmentShader.glsl) uses uniform sampler2D diffuseMap to sample a texture. In coursework.cpp, addTexture() is called to load PNG files, and the Model::draw() method binds these textures before rendering. |
| 52, 55, 58 | LO1: Basic use of translation, rotation and scaling transformations. | In coursework.cpp, glm::translate, glm::scale, and glm::rotate are used to position, size, and animate the two planets. The model matrix is updated each frame in the main render loop to create the rotation effect. |
| LO1: Implementation of glm library functions for calculating view and projection matrices. | The projection matrix is calculated using glm::perspective in the main render loop of coursework.cpp. The view matrix is calculated using a custom mymaths::lookAt function, demonstrating understanding beyond just using GLM. |
| LO2: 3D virtual world has been created using instances of a single object type. | The scene is created by loading a single model file (moon.obj) and creating two separate instances of it. Each instance is then transformed (scaled and positioned) independently and assigned different textures to represent two distinct planets. |
| LO3: Use of shaders to apply dynamic lighting from point light sources | The fragment shader (fragmentShader.glsl) implements a Blinn-Phong lighting model. It calculates lighting from two point light sources defined in an array (pointLightPositions, pointLightColors). Their contribution is calculated per-pixel and includes distance-based attenuation. |
| 62, 65, 68 | LO1: Implementation of students own functions for calculating view and projection matrices. | A custom lookAt function has been implemented in the mymaths namespace (maths.hpp / maths.cpp). This function calculates the view matrix from first principles using vector cross products and normalization. It is called by Camera::GetViewMatrix() in camera.cpp, completely replacing glm::lookAt. |
| LO2: 3D world created using multiple object types. | yes |
| LO2: Users can navigate the virtual world using keyboard and mouse inputs. | The application features first-person camera controls. Keyboard input (WASD, Space, Shift) is handled in the keyboardInput function to move the camera. Mouse input is captured in the mouseCallback function to control the camera's pitch and yaw, allowing full 3D navigation. |
| LO3: Use of shaders to apply dynamic lighting from different types of light sources. | The shader (fragmentShader.glsl) calculates lighting from two distinct types of sources: a global directional light (globalLightPos) that provides overall scene illumination, and two point lights (pointLightPositions) that add localized color and have distance-based attenuation. |
| 72 75, 78 | LO1: Implementation of students own functions to replace glm functions (e.g., glm::length(), glm::dot(), glm::cross() etc.). | The mymaths namespace (maths.hpp / maths.cpp) contains custom implementations of length, normalize, dot, and cross functions. These are used within the custom mymaths::lookAt function and the Camera::updateCameraVectors method. |
| LO1: Implementation of quaternions to calculate rotation matrix. | Na |
| 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 virtual world is dynamic, with both planets rotating automatically over time (controlled by glfwGetTime() in coursework.cpp). While the user doesn't directly control the objects or lights, they can fully control their viewpoint of this dynamic scene via the camera. |
| LO3: Appropriate implementation of normal and specular maps. | The fragment shader (fragmentShader.glsl) uses a sampler2D specularMap. The color sampled from this map is used to modulate the intensity and color of the specular reflection, allowing different parts of a surface (e.g., craters vs. rock) to have different shininess. This is implemented for both planets. |
| 85, 90, 100 | LO1: Use of quaternions to calculate view matrix. | Na |
| 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. |  |