# Computer Graphics Coursework – Self Assessment Document

**Name:** Maaria Faiz Rasul **ID number:** *22499140*

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 | Utilized glm::vec3 and glm::mat4 for positioning, scaling, and orienting celestial objects like Earth, Venus, and the Sun in 3D space. |
| LO2: Application compiles and runs without alterations to the source code of CMake file. | The project compiles and executes successfully with the provided CMakeLists.txt file, with no manual modifications required. |
| LO3: Implementation of shaders to apply appropriate textures to objects. | Basic vertex and fragment shaders were developed to render textures (e.g., Earth and Venus surface maps) loaded via stb\_image and accurately mapped to spherical meshes. |
| 52, 55, 58 | LO1: Basic use of translation, rotation and scaling transformations. | Applied glm::translate, glm::rotate, and glm::scale to animate orbital motion and simulate planetary rotation and revolution around the Sun. |
| LO1: Implementation of glm library functions for calculating view and projection matrices. | Used glm::lookAt() for camera positioning and glm::perspective() to set up a perspective projection, giving the scene realistic depth. |
| LO2: 3D virtual world has been created using instances of a single object type. | Created a starfield by placing multiple identical star objects throughout the background to simulate a cosmic environment. |
| LO3: Use of shaders to apply dynamic lighting from point light sources | A point light representing the Sun was implemented in the fragment shader. Lighting calculations include attenuation and color, affecting how planets are illuminated. |
| 62, 65, 68 | LO1: Implementation of students own functions for calculating view and projection matrices. | Developed custom functions to compute view and projection matrices manually, instead of using built-in glm functions like lookAt and perspective. |
| LO2: 3D world created using multiple object types. | The scene features different celestial bodies (e.g., Earth, Venus, the Sun, and stars), each modeled and textured uniquely. |
| LO2: Users can navigate the virtual world using keyboard and mouse inputs. | Implemented first-person navigation using WASD controls and mouse movement to allow exploration of the solar system from various viewpoints. |
| LO3: Use of shaders to apply dynamic lighting from different types of light sources. | Implemented directional and point lights with Phong lighting model using light.type uniform in shader for flexibility. |
| 72 75, 78 | LO1: Implementation of students own functions to replace glm functions (e.g., glm::length(), glm::dot(), glm::cross() etc.). | Included both a central point light (Sun) and adjustable ambient lighting. Shader logic differentiates light types via uniforms, using the Phong reflection model. |
| LO1: Implementation of quaternions to calculate rotation matrix. | Created and used custom implementations of vector math functions (length, dot product, cross product) for light direction and orbital mechanics. |
| 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.). | Used quaternions to compute rotation matrices for smooth and continuous rotation of planets, avoiding gimbal lock. |
| 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. | Light intensity, specular strength, and color dynamically change based on player distance using uniforms updated each frame. |