# Computer Graphics Coursework – Self Assessment Document

**Name:** *Olesya Panyushkina* **ID number:** *23639594*

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 | I have used vector and matrix objects throughout the whole project, for example vectors in scene object and lights position coordinates and orientation matrices in quaternion camera |
| LO2: Application compiles and runs without alterations to the source code of CMake file. |  |
| LO3: Implementation of shaders to apply appropriate textures to objects. | I have implemented shaders in fragmentShader and vertexShader, and have multiple objects in the scene with different textures applied, for example the “cheese”, “rat”(player) and “floor” objects |
| 52, 55, 58 | LO1: Basic use of translation, rotation and scaling transformations. | When instantiating objects in the scene, prior to pushing back each object, I have applied and changed the translation(position), rotation and scaling transformations, for example lines 195-198 in coursework.cpp |
| LO1: Implementation of glm library functions for calculating view and projection matrices. | I have implemented the glm library functions for calculating view and projection matrices, for example in my Camera object |
| LO2: 3D virtual world has been created using instances of a single object type. | My 3D world was created using multiple object types, for example Light for light sources and Objects in the scenes |
| LO3: Use of shaders to apply dynamic lighting from point light sources | I use different types of light sources, for example point and spot light (lines 134-177), as well as different colors and light directions. |
| 62, 65, 68 | LO1: Implementation of students own functions for calculating view and projection matrices. | I have implemented my own functions for view and projection such as LookAtPersonal() in lines (503-528) and projectionPersonal() in lines (532-543) |
| LO2: 3D world created using multiple object types. | My 3D world was created using multiple object types, for example Light for light sources and Objects in the scenes |
| LO2: Users can navigate the virtual world using keyboard and mouse inputs. | My users can navigate the world space using WASD and rotate the camera using the mouse, as well as switch between first and third person camera by pressing SPACE. |
| LO3: Use of shaders to apply dynamic lighting from different types of light sources. | I used different types of light sources, for example point and spot light (lines 134-177), as well as different colors and light directions. |
| 72 75, 78 | LO1: Implementation of students own functions to replace glm functions (e.g., glm::length(), glm::dot(), glm::cross() etc.). | I have implemented my own functions own functions to replace glm functions(lines 476-500), such as a function to normalize a vector (make it a unit vector), function to check if an object is in front of the player, or a function to check if camera rays intersect with an object (if object is in view) |
| LO1: Implementation of quaternions to calculate rotation matrix. | In my maths.cpp file in lines (23-31) I have implemented a rotation matrix that uses quaternions for its calculations |
| 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 objects in my virtual world are dynamic, for example the moving/floating up cheese objects once approached moving light sources if the cheese is approached by the player |
| LO3: Appropriate implementation of normal and specular maps. | My fragment shader uses all diffuse, normal and specular maps in the fragmentShader.glsl |
| 85, 90, 100 | LO1: Use of quaternions to calculate view matrix. | My camera object class has quaternionCamera method, that calculates a view matrix and utilizes said view matrix for the vectors of the camera (right, up and front) |
| LO1: Use of SLERP to smooth out changes in camera direction. | My camera object class has quaternionCamera method, that uses SLERP for it’s orientation (line 26-43 in camera.cpp) |
| LO2: Implementation of a third person camera with the ability to switch between first and third period view. | I have implemented a third person camera, and the player can switch between first and third person by pressing SPACE (lines 380-421) |
| 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.). | The position of the first and third person camera cant pass though the floor and doesn’t let the player fly up in the sky. The player and the position of the camera also cannot pass through objects like the “cheese” which is evidenced in lines (528-551) |
| LO3: Use of shaders to apply parameter driven effects within the scene, e.g., light properties controlled using camera/character position. | The lights in my scene change based on the player’s position, as the player approaches one of the cheese objects |