# Computer Graphics Coursework – Self Assessment Document

**Name:** Sabeel Khan **ID number:** *23640569*

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 | In my code I’ve used GLSLs vec3 and vec2 types in vertexShader.glsl to store the vertex positions and texture coordinates. They are passed into the shader using vertex attributes in the render loop in coursework.cpp. The vec3 vertex position is then converted into a vec4 which allows for matrix multiplication (used for the MVP)  For transformations I have used glm::mat4 matrix objects in coursework.cpp. These include translation, rotation and scaling matrices (within the render loop) which is then multiplied to form a final matrix MVP |
| LO2: Application compiles and runs without alterations to the source code of CMake file. |  |
| LO3: Implementation of shaders to apply appropriate textures to objects. | In my code, textures are applied to the 3D objects using the fragment and vertexshaders. In coursework.pp, the texture cat.jpg is loaded and bounded using glBindTexture. The UV coordinates are then passed to the shaders with a attribute index (location) of 1. In vertexShader.glsl, the UV coordinates are then received and passed onto the fragmentShader.glsl, where the texture is sampled using the UV coordinates and then rendered onto the objects. |
| 52, 55, 58 | LO1: Basic use of translation, rotation and scaling transformations. | In my code, I implemented translation, rotation and scaling using mat4 matrices through the custom functions that are defined in maths.cpp. The functions are Maths::translate, Maths::rotate and Maths::Scale. I then used these functions in coursework.cpp, in the render loop.  I then also applied combined the individual transformation matrices within the render loop to a single modelMatrix (translate \* rotate \* scale) which is a key part of the MVP matrix |
| LO1: Implementation of glm library functions for calculating view and projection matrices. | Please see 62, 65, 68 🡪 LO1: Implementation of students own functions for calculating view and projection matrices. |
| LO2: 3D virtual world has been created using instances of a single object type. | In my code I have created a 3D world using multiple instances of a single object type (in coursework.cpp).First the code defines a resuable Object struct which stores information of the position, rotation etc. The code then defines an array of vec3 positions for each of the cube instances, and then uses a loop to instantiate the cubes with different positions (already set in the array positions[]), and modifying the orientations by modifying the values in Object before pushing it back to the vector.  In the render loop, the code then loops through the objects vector, applying the different individual transformations to each of the cubes, calculating and implementing the MVP and then glDrawElements is called at the end. |
| 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. | In my code, I implemented my own custom functions for calculating the view and projection matrices. These functions are defined in maths.cpp, as Maths::customLookAt and Maths::customPerspective. These functions are based on the information given to me from the lectures and labsheets  The customLookAt function constructs the view matrix by first translating a default mat4 matrix by -eye and then calculates the front, right and up vectors based on the eye (cameras position), the target and the world up vector. These vectors are then used to construct the rotation matrix which is combined with the translation matrix which shifts the worldspace so that the camera is at 0,0,0  The customPerspective function calculates a perspective projection matrix given an fov, aspect ratio, near and far planes. The function then calculates the top and right values based on the given parameters. The projection matrix is then constructed based on the formula given in the lectures for each element in the matrix  Both functions are then used in coursework.cpp within the render loop to form the MVP matrix |
| 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. |  |