# Computer Graphics Coursework – Self Assessment Document

**Name:** *Joseph Hill* **ID number:** 23630877

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 | Inside my loadObj method, I make use of the std::vector<glm::vec3> and vec2 functions. I load .obj files that I declare and using this, I loop through each vertex of the triangle wire mesh and get the indices of its attributes, store them and then in my main, I call model.draw and bind my textures and draw the indices of all the vertexes of the triangles. During my render loop I also loop through all my objects in my scene and calculate the model matrix, to get the translation, scale, rotation of each model. This I then  send to the vertex shader before drawing the models. |
| 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 main I compile shaders with my shaderId and my lightShaderID. I load in my glsl vertex shader and fragment shader. My vertex shader outputs my texture co-ordinates and calculates the TBN matrix to transform view space into tangent space. I then activate my shader in main and use my model.addtexture method to apply textures from files to my objects |
| 52, 55, 58 | LO1: Basic use of translation, rotation and scaling transformations. | My basic translation I use in the initiation of objects, I declare their position, rotation and scale before I draw them. I also use glm::translate, glm::rotate and I combine them into the model matrix, which I send to the shader. I also dynamically rotate my teapot object using quaternion slerp within a proximity. |
| LO1: Implementation of glm library functions for calculating view and projection matrices. | My code uses the glm lookAt function in my Camera::CalculateMatrices (which I no longer use) I swapped it to the camera::quaternionCamera. These matrices are updates and used in my render loop. For my projection I use glm::perspective to simulate a realistic camera lense and allow myself an adjustable FOV |
| LO2: 3D virtual world has been created using instances of a single object type. | I load each model (cube, wall, sphere, teapot) using my model class and reuse it by creating multiple object Structs that store their own position rotation and scale. I store all these objects in an objects array and loop through it and draw each one in my render loop. |
| LO3: Use of shaders to apply dynamic lighting from point light sources | I define my point lights just after the end of the window creation. In my fragment shader I apply lighting calculations of the ambient, diffuse and specular lighting. I have a light class that is able to send multiple lights to my fragment shader and interact dynamically with my 3D scene. |
| 62, 65, 68 | LO1: Implementation of students own functions for calculating view and projection matrices. | In my camera.cpp I have a custom LookAt and custom Perspective function inside my calculate matrices method. It manually calculates the view matrix by building a coordinate system using the cameras position, target and up vector |
| LO2: 3D world created using multiple object types. | I load each model (cube, wall, sphere, teapot) using my model class and reuse it by creating multiple object Structs that store their own position rotation and scale. I store all these objects in an objects array and loop through it and draw each one in my render loop. |
| LO2: Users can navigate the virtual world using keyboard and mouse inputs. | Using my keyboard Input in my coursework.cpp I use w,a,s,d to move the camera front right left and back. I also use the camera yaw and pitch and glfwGetCursorpos to move the camera up down left and right. |
| LO3: Use of shaders to apply dynamic lighting from different types of light sources. | I have both spotlights and point lights in my scene. The point lights emit light in all directions from its position whereas the spotlight simulates a cone of light using an angle cutoff (cosPhi) the fragment shader checks the angle between the light direction and the fragment direction. I then place all my lightsources into an object class light, and send them all to the shader to calculate the source properties in my camera view. |
| 72 75, 78 | LO1: Implementation of students own functions to replace glm functions (e.g., glm::length(), glm::dot(), glm::cross() etc.). | Inside my maths class I have functions for my own dot cross and length functions, throughout most of the code I use my own versions of the dot and cross and length. The best example is in the camera cpp where inside the customLookAt method I use Maths::Dot() (which is my custom Dot function) and above I also use Maths::Cross() they work by using the dot and cross formula to calculate the product. |
| LO1: Implementation of quaternions to calculate rotation matrix. | Inside my maths cpp and hpp I have a quaternion class which I use to calculate the matrix. It has the w, x, y, z components which converts a quaternion into a mat4 rotation matrix. This replaces my Euler based rotation. |
| 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.). | Inside my render loop I go through each Lightsource and flash them between blue and red, once my camera is within the proximity of the spotlight radius. Within my teapot drawing code, I also use my quaternion class to dynamically spin the teapot around the y axis. |
| LO3: Appropriate implementation of normal and specular maps. | In my shader pipeline I sample the normalMap texture in the fragment shader and transform it into tangent space using the TBN matrix, which I calculate in the Vertex Shader. I distinguish between what map im adding in the AddTexture method, where I pass in either addTexture(“texture.png”, “normal”) or addTexture(“texture.png”, “specular”) |
| 85, 90, 100 | LO1: Use of quaternions to calculate view matrix. | In my camera class I use my QuaternionCamera method which builds quaternions from the cameras pitch and yaw. I use my quaternion::matrix() method to turn it into a rotation matrix and apply it to calculate the front and right vectors of the camera. I use these vectors to construct the view matrix, and inside my render loop I call the camera.quaternionCamera instead of the Euler-angle one. |
| LO1: Use of SLERP to smooth out changes in camera direction. | Again inside my camera class I apply Slerp to the orientation of the camera within the quaternionCamera() method. I then use this on the view matrix calculation. |
| 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.). | Inside my keyboard input method, I loop through each object within the objects vector and using each of their positions I check the distance from myself to the object, if my distance is < 1.0f of the object, I push the camera object backwards to simulate collisions. I also clamp the movement so that I cannot pass through walls using glm::clamp(new pos , wall pos) |
| LO3: Use of shaders to apply parameter driven effects within the scene, e.g., light properties controlled using camera/character position. |  |