# Computer Graphics Coursework – Self Assessment Document

**Name:** *Nasir Brooks* **ID number:** *22527043*

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 code uses **glm** for defining vectors and matrices. For example, the glm::vec3 type is used for light positions, and glm::mat4 is used for transformations like translation and rotation.  glm::vec3 lightPosition = glm::vec3(2.0f, 2.0f, 2.0f); // Vector  glm::mat4 translate = Maths::translate(objects[i].position); // Matrix  glm::mat4 rotate = Maths::rotate(objects[i].angle, objects[i].rotation); |
| 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 code successfully loads shaders and applies textures to objects like the wheel by calling the addTexture method, which maps texture files to materials used in the rendering process.  shaderID = LoadShaders("vertexShader.glsl", "fragmentShader.glsl");  lightShaderID = LoadShaders("lightVertexShader.glsl", "lightFragmentShader.glsl");  glUseProgram(shaderID);  wheel.addTexture("../assets/wheel\_metallic", "diffuse"); |
| 52, 55, 58 | LO1: Basic use of translation, rotation and scaling transformations. | The code demonstrates basic transformations using translation, rotation, and scaling matrices, allowing for the manipulation of object properties within the scene.  glm::mat4 translate = Maths::translate(objects[i].position); // Translation  glm::mat4 scale = Maths::scale(objects[i].scale); // Scaling  glm::mat4 rotate = Maths::rotate(objects[i].angle, objects[i].rotation); // Rotation |
| LO1: Implementation of glm library functions for calculating view and projection matrices. | The **camera** object calls the calculateMatrices() method to compute and update both the view and projection matrices, which are essential for camera movement and rendering in the 3D space.  camera.calculateMatrices(); // Calculates view and projection matrices. |
| LO2: 3D virtual world has been created using instances of a single object type. | The code uses a loop to instantiate the object (likely the **wheel** model) multiple times with different positions, creating instances of a single object type for the 3D world.  for (unsigned int i = 0; i < 10; i++) {  object.position = positions[i];  objects.push\_back(object);  } |
| LO3: Use of shaders to apply dynamic lighting from point light sources | The code adds point light sources dynamically with specified properties like position and color and passes the light information to the shader program for correct lighting calculations.  LightSource.addPointLight(glm::vec3(-12.0f, 33.0f, 12.0f), blue, 0.2f, 0.05f, 0.01f);  glUniform3fv(glGetUniformLocation(shaderID, "lightPosition"), 1, &viewSpaceLightPosition[0]); |
| 62, 65, 68 | LO1: Implementation of students own functions for calculating view and projection matrices. | camera.calculateMatrices(); |
| LO2: 3D world created using multiple object types. | Model wheel("../assets/wheel.obj");  Model plane("../assets/plane.obj"); |
| LO2: Users can navigate the virtual world using keyboard and mouse inputs. | if (glfwGetKey(window, GLFW\_KEY\_W) == GLFW\_PRESS)  camera.eye += 5.0f \* deltaTime \* camera.front; // W key to move forward  camera.yaw += 0.005f \* float(xPos - 1024 / 2); // Mouse input for yaw  camera.pitch += 0.005f \* float(768 / 2 - yPos); // Mouse input for pitch |
| LO3: Use of shaders to apply dynamic lighting from different types of light sources. | LightSource.addPointLight(glm::vec3(5.0f, 10.0f, -5.0f), blue, 0.2f, 0.05f, 0.01f);  glUniform3fv(glGetUniformLocation(shaderID, "lightPosition"), 1, &viewSpaceLightPosition[0]); |
| 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. |  |