# Computer Graphics Coursework – Self Assessment Document

**Name:** *Sam Sharples* **ID number:** *22466273*

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 an array of vector objects through using glm::vec3 position as in line 130 “glm::vec3 basketballPositions[] = {  glm::vec3(-3.0f, 1.0f, -2.0f),  glm::vec3(-2.0f, 1.0f, -5.0f),  glm::vec3(-3.0f, 1.0f, -6.0f),  glm::vec3(-3.0f, 1.0f, -4.0f),  glm::vec3(-2.0f, 1.0f, -7.0f),  glm::vec3(-3.0f, 1.0f, -6.5f),  }; And matrices used in lines 266 to 269  glm::mat4 translate = Maths::translate(objects[i].position);  glm::mat4 scale = Maths::scale(objects[i].scale);  glm::mat4 rotate = Maths::rotate(objects[i].angle, objects[i].rotation);  glm::mat4 model = translate \* rotate \* scale; |
| 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 line 154 to line 158  basketball.addTexture("../assets/DiffuseLeather.JPG", "diffuse");  basketball.addTexture("../assets/NormalLeather.PNG", "normal");  basketball.addTexture("../assets/SpecularLeather.PNG", "specular");  football.addTexture("../assets/DiffuseFootball.JPG", "diffuse");  football.addTexture("../assets/NormalFootball.JPG", "normal");  Textures have been applied to objects and use of shaders. |
| 52, 55, 58 | LO1: Basic use of translation, rotation and scaling transformations. | Basic use of scaling and rotation in lines 165 to 167 -  object.position = basketballPositions[i];  object.rotation = glm::vec3(1.0f, 1.0f, 1.0f);  object.scale = glm::vec3(0.75f, 0.75f, 0.75f);  Basic translation in line 266  glm::mat4 translate = Maths::translate(objects[i].position); |
| LO1: Implementation of glm library functions for calculating view and projection matrices. | Using my own function to calculate the view and projection matrices.  In lines 247 and 248  camera.target = camera.eye + camera.front;  camera.calculateMatrices(); |
| LO2: 3D virtual world has been created using instances of a single object type. | Created multiple instances of multiple object type as seen in lines 93 and 96 to load the models of football.obj and basketball.obj.  Model basketball("../assets/basketball.obj");  Model football("../assets/Soccer\_Ball.obj");  I positioned each object to create a 3D world using arrays of vectors as in lines 132 to 141 in case of the basketballs  glm::vec3 basketballPositions[] = {  glm::vec3(-3.0f, 1.0f, -2.0f),  glm::vec3(-2.0f, 1.5f, -5.0f),  glm::vec3(-3.0f, 1.2f, -6.0f),  glm::vec3(-3.0f, 1.0f, -4.0f),  glm::vec3(-2.0f, 1.5f, -7.0f),  glm::vec3(-3.0f, 1.5f, -6.5f),  }; |
| LO3: Use of shaders to apply dynamic lighting from point light sources | I create a point light from lines 125 to 129  lightSources.addPointLight(  glm::vec3(0.0f, 0.0f, 1.0f),  glm::vec3(1.0f, 0.9f, 0.7f),  1.0f, 0.09f, 0.032f  );  Then pass light to the shaders in line 262  lightSources.toShader(shaderID, camera.view); |
| 62, 65, 68 | LO1: Implementation of students own functions for calculating view and projection matrices. | Using my own function to calculate the view and projection matrices.  In lines 247 and 248  camera.target = camera.eye + camera.front;  camera.calculateMatrices(); |
| LO2: 3D world created using multiple object types. | I used obj of basketballs and football to create a 3D world and positioned them accordingly.  I positioned each object to create a 3D world using arrays of vectors as in lines 132 to 141 in case of the basketballs  glm::vec3 basketballPositions[] = {  glm::vec3(-3.0f, 1.0f, -2.0f),  glm::vec3(-2.0f, 1.5f, -5.0f),  glm::vec3(-3.0f, 1.2f, -6.0f),  glm::vec3(-3.0f, 1.0f, -4.0f),  glm::vec3(-2.0f, 1.5f, -7.0f),  glm::vec3(-3.0f, 1.5f, -6.5f),  }  And in lines 144 to 153 to position the footballs using a vector array  glm::vec3 footballPositions[] = {  glm::vec3(-3.0f, 0.7f, -2.0f),  glm::vec3(-2.0f, 0.7f, -5.0f),  glm::vec3(-3.0f, 0.7f, -6.0f),  glm::vec3(-3.0f, 0.7f, -4.0f),  glm::vec3(-2.0f, 0.7f, -7.0f),  glm::vec3(-3.0f, 0.7f, -6.5f),  }; |
| LO2: Users can navigate the virtual world using keyboard and mouse inputs. | I have added keyboard input controls and mouse input controls.  From lines 320 to 351.  void keyboardInput(GLFWwindow\* window)  {  if (glfwGetKey(window, GLFW\_KEY\_ESCAPE) == GLFW\_PRESS)  glfwSetWindowShouldClose(window, true);  if (glfwGetKey(window, GLFW\_KEY\_W) == GLFW\_PRESS)  camera.eye += 5.0f \* deltaTime \* camera.front;  if (glfwGetKey(window, GLFW\_KEY\_S) == GLFW\_PRESS)  camera.eye -= 5.0f \* deltaTime \* camera.front;  if (glfwGetKey(window, GLFW\_KEY\_A) == GLFW\_PRESS)  camera.eye -= 5.0f \* deltaTime \* camera.right;  if (glfwGetKey(window, GLFW\_KEY\_D) == GLFW\_PRESS)  camera.eye += 5.0f \* deltaTime \* camera.right;  }  void mouseInput(GLFWwindow\* window)  {  double xPos, yPos;  glfwGetCursorPos(window, &xPos, &yPos);  glfwSetCursorPos(window, 1024 / 2, 768 / 2);  camera.yaw += 0.005f \* float(xPos - 1024 / 2);  camera.pitch += 0.005f \* float(768 / 2 - yPos);  camera.calculateCameraVectors();  } |
| LO3: Use of shaders to apply dynamic lighting from different types of light sources. | I used a dynamic from a spotlight using a calculation including cos sin and time to make a moving spotlight across time in lines from lines 252 to 261.  glm::vec3 newSpotPos = glm::vec3(4.0f \* cos(time), 5.0f, 6.0f \* sin(time));  glm::vec3 newSpotDir = glm::vec3(0.0f, -1.0f, 0.0f);  if (!lightSources.lightSources.empty() && lightSources.lightSources[0].type == 2) {  lightSources.lightSources[0].position = newSpotPos;  lightSources.lightSources[0].direction = glm::normalize(newSpotDir);  lightSources.lightSources[0].colour = glm::vec3(0.0f, 0.0f, 1.0f);  } |
| 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. |  |