Computer Graphics Coursework – Self Assessment Document

Name: Sajid Uddin Your Name ID number: 23662232

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%) | w | 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 used glm::mat4 and glm::vec3 from the GLM library to create model, view and projection matrices. I added a translation to move the camera back using glm::translate and used glm::perspective for the projection. The model matrix was used to rotate and translate two cube instances in the scene. |
| 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 made a simple vertex and fragment shader. The vertex shader uses model, view, and projection matrices to place the cube in the right position. The fragment shader colours the cube bright green. I loaded them using LoadShaders() and they show the cubes correctly on screen. The shaders don’t use textures or lighting yet, just a solid colour. |
| 52, 55, 58 | LO1: Basic use of translation, rotation and scaling transformations. | I used glm::translate to move one cube to the right and another one to the left. I used glm::rotate to spin one cube over time. I also used glm::scale to shrink a cube to half size. This shows I understand how to apply translation, rotation, and scaling in a 3D scene. |
| LO1: Implementation of glm library functions for calculating view and projection matrices. | I used glm::translate to move the camera back so the cubes are visible, and glm::perspective to make the scene look 3D. These are from the glm library and help show depth in the scene. |
| LO2: 3D virtual world has been created using instances of a single object type. | I used glm::translate to move the camera back so the cubes are visible in the scene, and glm::perspective to create a 3D perspective view. Both functions are from the glm library and help show depth, making the scene look realistic. |
| LO3: Use of shaders to apply dynamic lighting from point light sources | I added lighting to my cubes using shaders. I passed the light direction, light colour, and object colour into the shader as uniforms. In the vertex shader, I calculated the surface normal and position. The fragment shader uses these to apply ambient and diffuse lighting, so the cube faces light up depending on their angle. This makes the scene look more realistic. |
| 62, 65, 68 | LO1: Implementation of students own functions for calculating view and projection matrices. |  |
| LO2: 3D world created using multiple object types. | A 3D world was constructed using multiple cube objects, each manipulated through scaling, translation, and rotation, demonstrating different object transformations. |
| LO2: Users can navigate the virtual world using keyboard and mouse inputs. | Users can move through the virtual world using keyboard input (WASD) and look around using mouse movement, supporting full 360° navigation. |
| LO3: Use of shaders to apply dynamic lighting from different types of light sources. | I added directional lighting to my scene using fragment and vertex shaders. It includes both diffuse and specular lighting, and I passed the camera position into the shader to calculate the specular highlights properly. I used normals for each vertex to make the lighting look more realistic across the cube surfaces. It’s a basic setup, but it helped me understand how light interacts with 3D objects in OpenGL. I haven’t added more complex light types like point or spotlights, but the current lighting works well for my scene. |
| 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. | implemented third person camera toggle system where it will switch point of view with the press of C, it is only limited to fixed angle, though you cannot move around you can move the camera at 360 degree angles. |
| 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. |  |