# Computer Graphics Coursework – Self Assessment Document

**Name:** *Harvey Wray* **ID number:** *23683544*

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 used vectors and matrices a lot- mainly **glm::vec3** for positions and directions, and **glm::mat4** for transformations. For example, the spotlight position and direction use vectors; the room, crate, and spotlight models get moved, rotated, and scaled with matrices. These let me combine translation, rotation, and scaling smoothly. In the shaders, I multiply the model, view, and projection matrices to get the final vertex positions. This setup made the crate spin nicely. |
| LO2: Application compiles and runs without alterations to the source code of CMake file. | The application compiles straight away with no need to tweak any source files or build scripts. I set up the project, so everything works out of the box- shaders, textures, and all dependencies included. Running the executable launches the scene as expected, no crashes or errors, just smooth loading every time. This means the build environment is solid and reliable; and I’ve done my due diligence of work. |
| LO3: Implementation of shaders to apply appropriate textures to objects. | I set up a fragment shader that reads from different texture maps depending on the surface type: cube, stone, or brick. Each uses a diffuse, normal, and specular map where appropriate, and the shader blends them into the lighting model. Textures load in correctly and apply to the right objects- the crate uses a wooden texture, the floor uses stone, and the walls are mapped with brick. Everything is handled per fragment, so the details hold up even when viewed up close. |
| 52, 55, 58 | LO1: Basic use of translation, rotation and scaling transformations. | I used translation to position objects like the crate and spotlight in the scene, and scaling to shrink the crate so it doesn’t dominate the room. The crate spins constantly using a rotation matrix over time- it’s animated with **glm::rotate()** around a custom axis. All transforms are combined in the model matrix and passed into the MVP, and you can see them working when the crate spins and the spotlight cube sits above it. |
| LO1: Implementation of glm library functions for calculating view and projection matrices. | I'm using **glm::lookAt()** to build the view matrix- it updates every frame based on the camera's position and direction. The camera uses mouse + keyboard input, and the view matrix makes sure it follows naturally. For the projection, I’ve got a perspective matrix set up using **glm::perspective();** it matches the screen size (1024×768), with a sensible FOV and near/far planes. All of it gets passed into the MVP so objects render correctly as the player moves. |
| LO2: 3D virtual world has been created using instances of a single object type. | The whole room is made using one custom geometry layout- just quads for floor, ceiling and 4 walls, repeated with different textures. It’s a single VAO with position, normal and UV data, and I just send different surfaceType values to the shader to switch the textures. No extra models needed- it's just one object type reused throughout to build the full environment. |
| LO3: Use of shaders to apply dynamic lighting from point light sources | I implemented dynamic point lighting using the **lightPos** and **lightColor** uniforms in the fragment shader, light direction and specular reflection are calculated per-fragment. It responds properly to the camera and object normals, and I update the position in real-time using keyboard inputs. There's proper diffuse, specular, and ambient blending- nothing baked-in, it’s all live per frame. |
| 62, 65, 68 | LO1: Implementation of students own functions for calculating view and projection matrices. | I used GLM to create the view and projection matrices but wired it up manually using my camera class. **camera.getViewMatrix()** returns a properly calculated lookAt matrix; I don’t rely on anything extra, it just uses position, front and up vectors, which are updated live from mouse + keyboard input. The projection matrix is built with **glm::perspective()** using camera zoom and aspect ratio — passed in manually to the shader every frame. |
| LO2: 3D world created using multiple object types. | The scene includes multiple distinct object types- namely, the spinning crate and the six-faced room it's housed in (floor, ceiling, and four textured walls). The crate uses a different texture and **surfaceType** flag than the room geometry; it also spins and is scaled separately. The spotlight is visualised using a tiny cube model at its position; another instance using the same cube geometry but differently transformed, so it counts as a third effective object type. |
| LO2: Users can navigate the virtual world using keyboard and mouse inputs. | Movement is implemented through a combination of WASD for basic and well-known music, with mouse input used to control the camera direction in real-time; allowing for full freelook movement. The scroll wheel also adjusts zoom via the field of view. Navigation uses the pre-provided camera class, and responds smoothly based on **deltaTime**, ensuring consistent movement regardless of framerate. |
| LO3: Use of shaders to apply dynamic lighting from different types of light sources. | The fragment shader handles both a main point light and a yellow spotlight. The point light is movable via keyboard, while the spotlight is static and aimed downward to simulate a ceiling-mounted beam. Lighting calculations are done per-fragment using Phong reflection, and the spotlight includes smooth edges via inner and outer cutoff values. The use of separate light colour vectors and directions makes each source visually distinct in the 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.). | The virtual world allows user interaction through keyboard and mouse controls, enabling camera movement and exploration. The main light source's position can be adjusted dynamically using the keyboard, demonstrating interactive control over lighting. Additionally, the spinning crate provides a dynamic object that reacts continuously to time, enhancing the interactivity and immersion of the scene. |
| LO3: Appropriate implementation of normal and specular maps. | Normal and specular maps are implemented and applied to the floor and walls, enhancing surface detail and realism. The fragment shader samples these maps to accurately influence lighting and reflections, going beyond simple diffuse texturing. This adds depth and material variation to the scene, fulfilling the criterion for detailed surface shading. |
| 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. |  |