# Computer Graphics Coursework – Self Assessment Document

**Name:** *Your Name* **ID number:** *12345678*

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 | **Matrixes and Vectors Are Used For deifine Crucial Math Structs in Coursework.cpp**  Mat4 ProjectionMatrix = PerspectiveFov(45.0f, 16.0f / 9.0f, 0.1f, 10000.0f);  glm::mat4 ModelMatrix = glm::scale(glm::mat4(1.0f), glm::vec3(500, 30, 500));  glm::mat4 view = glm::mat4(1.0f);  coursework.cpp line 131  struct LightSource  {  glm::vec3 position;  glm::vec3 ambientColor;  glm::vec3 diffuseColor;  glm::vec3 specularColor;  float focalStrength;  float specularIntensity;  };  Line 15 |
| LO2: Application compiles and runs without alterations to the source code of CMake file. |  |
| LO3: Implementation of shaders to apply appropriate textures to objects. | Textures Are Applied In the Fragmentshader .glsl As shown in the Image Above Line 90 |
| 52, 55, 58 | LO1: Basic use of translation, rotation and scaling transformations. | **Transformations Are Used To Manipulate 3D models That Are Rendered for proper placement**  ModelMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(1.0f, 20.0f, 1.0f)) \* glm::scale(glm::mat4(1.0f), glm::vec3(70.5f));  glUniformMatrix4fv(modelLoc, 1, false, &ModelMatrix[0][0]);  StoneAltar.draw(Program, true);  Coursework.cpp line 347 |
| LO1: Implementation of glm library functions for calculating view and projection matrices. | **In Maths File Alternative Functions For Perspective And Lookat Are Implemented They Are Used To Replace the Glm counterparts in coursework.cpp**  Mat4 PerspectiveFov(float fovYDeg, float aspect, float zNear, float zFar)  {  float fovYRad = toRadians(fovYDeg);  float tanHalfFovy = std::tan(fovYRad / 2.0f);  Mat4 result;  result.cols[0] = Vec4(1.0f / (aspect \* tanHalfFovy), 0.0f, 0.0f, 0.0f);  result.cols[1] = Vec4(0.0f, 1.0f / tanHalfFovy, 0.0f, 0.0f);  result.cols[2] = Vec4(0.0f, 0.0f, -(zFar + zNear) / (zFar - zNear), -1.0f);  result.cols[3] = Vec4(0.0f, 0.0f, -(2.0f \* zFar \* zNear) / (zFar - zNear), 0.0f);  return result;  }  Mat4 LookAt(const Vec3& eye, const Vec3& center, const Vec3& up) {  Vec3 f = (center - eye).normalize();  Vec3 s = f.cross(up).normalize();  Vec3 u = s.cross(f);  Mat4 result;  result.cols[0] = Vec4(s.x, u.x, -f.x, 0.0f);  result.cols[1] = Vec4(s.y, u.y, -f.y, 0.0f);  result.cols[2] = Vec4(s.z, u.z, -f.z, 0.0f);  result.cols[3] = Vec4(-s.dot(eye), -u.dot(eye), f.dot(eye), 1.0f);  return result;  }  Maths.cpp Line 96 |
| LO2: 3D virtual world has been created using instances of a single object type. |  |
| LO3: Use of shaders to apply dynamic lighting from point light sources | **Shaders Use Multiple Point Lights that Can Be Toggled On Or Off To Show their Constant Effect on the Scene**  if(bUseLighting == true)  {  // properties  phongResult += calcDirectionalLight(vec3(0, 1, 0.7), lightNormal, fragmentPosition, viewDirection);  for(int i = 0; i < TOTAL\_LIGHTS; i++)  {  if(!toggleLight1 && i == 0 ) {  continue;  }  if(!toggleLight2 && i == 1 ) {  continue;  }  else{  phongResult += CalcLightSource(lightSources[i], lightNormal, fragmentPosition, viewDirection);  }  }  FragmentsShader Line 67 |
| 62, 65, 68 | LO1: Implementation of students own functions for calculating view and projection matrices. | **In Maths File Alternative Functions For Perspective And Lookat Are Implemented They Are Used To Replace the Glm counterparts in coursework.cpp**  Mat4 PerspectiveFov(float fovYDeg, float aspect, float zNear, float zFar)  {  float fovYRad = toRadians(fovYDeg);  float tanHalfFovy = std::tan(fovYRad / 2.0f);  Mat4 result;  result.cols[0] = Vec4(1.0f / (aspect \* tanHalfFovy), 0.0f, 0.0f, 0.0f);  result.cols[1] = Vec4(0.0f, 1.0f / tanHalfFovy, 0.0f, 0.0f);  result.cols[2] = Vec4(0.0f, 0.0f, -(zFar + zNear) / (zFar - zNear), -1.0f);  result.cols[3] = Vec4(0.0f, 0.0f, -(2.0f \* zFar \* zNear) / (zFar - zNear), 0.0f);  return result;  }  Mat4 LookAt(const Vec3& eye, const Vec3& center, const Vec3& up) {  Vec3 f = (center - eye).normalize();  Vec3 s = f.cross(up).normalize();  Vec3 u = s.cross(f);  Mat4 result;  result.cols[0] = Vec4(s.x, u.x, -f.x, 0.0f);  result.cols[1] = Vec4(s.y, u.y, -f.y, 0.0f);  result.cols[2] = Vec4(s.z, u.z, -f.z, 0.0f);  result.cols[3] = Vec4(-s.dot(eye), -u.dot(eye), f.dot(eye), 1.0f);  return result;  }  Maths.cpp Line 96 |
| LO2: 3D world created using multiple object types. | Scene Containing Multiple Objects Loaded in Coursework.cpp |
| LO2: Users can navigate the virtual world using keyboard and mouse inputs. | **Keyboard And Mouse Imputs Allow Users to Navigate and Interact with The Scene World its implemented in coursework.cpp**  void keyboardInput(GLFWwindow \*window)  {  if (glfwGetKey(window, GLFW\_KEY\_ESCAPE) == GLFW\_PRESS)  glfwSetWindowShouldClose(window, true);  float speed = 5.125f;  if (glfwGetKey(window, GLFW\_KEY\_ESCAPE) == GLFW\_PRESS)  {  glfwSetWindowShouldClose(window, true);  }  if (glfwGetKey(window, GLFW\_KEY\_W) == GLFW\_PRESS)  camera.ProcessKeyboard(FORWARD, deltaTime \* speed);  if (glfwGetKey(window, GLFW\_KEY\_S) == GLFW\_PRESS)  camera.ProcessKeyboard(BACKWARD, deltaTime \* speed);  if (glfwGetKey(window, GLFW\_KEY\_A) == GLFW\_PRESS)  camera.ProcessKeyboard(LEFT, deltaTime \* speed);  if (glfwGetKey(window, GLFW\_KEY\_D) == GLFW\_PRESS)  camera.ProcessKeyboard(RIGHT, deltaTime \* speed);  //glfwGetCursorPos(window, &KeyboardCursor\_x, &keyboard\_cursor\_y);  if (glfwGetKey(window, GLFW\_KEY\_LEFT) == GLFW\_PRESS) {  // KeyboardCursor\_x -= 0.01f;  camera.ProcessMouseMovement(-KeyboardCursor\_x, 0.0f);  }  if (glfwGetKey(window, GLFW\_KEY\_RIGHT) == GLFW\_PRESS) {  //KeyboardCursor\_x += 0.01f;  camera.ProcessMouseMovement(KeyboardCursor\_x, 0.0f);  }  if (glfwGetKey(window, GLFW\_KEY\_UP) == GLFW\_PRESS) {  //keyboard\_cursor\_y += 0.01f;  camera.ProcessMouseMovement(0.0f, keyboard\_cursor\_y);  }  if (glfwGetKey(window, GLFW\_KEY\_DOWN) == GLFW\_PRESS) {  //keyboard\_cursor\_y -= 0.01f;  camera.ProcessMouseMovement(0.0f, -keyboard\_cursor\_y);  }  if (glfwGetKey(window, GLFW\_KEY\_1) == GLFW\_PRESS)  {  ToggleLight1 = !ToggleLight1;  }  if (glfwGetKey(window, GLFW\_KEY\_2) == GLFW\_PRESS)  {  ToggleLight2 = !ToggleLight2;  }  }  Coursework.cpp Line 385 |
| LO3: Use of shaders to apply dynamic lighting from different types of light sources. | **Two**  **different Light Sources Are Used it includes Point Lights and Directional Lights**  vec3 calcDirectionalLight(vec3 Dir, vec3 lightNormal, vec3 vertexPosition, vec3 viewDirection)  {  float intensity = 0.50f;  vec3 ambient;  vec3 diffuse;  vec3 specular;  // Calculate light direction and distance  vec3 lightDirection = normalize(Dir);  // Ambient  ambient = vec3(0.5f) \* Ka;  // Diffuse  float impact = max(dot(lightNormal, lightDirection), 0.0);  diffuse = impact \* vec3(0.5f) \* intensity;  // Specular  vec3 reflectDir = reflect(-lightDirection, lightNormal);    return (ambient + diffuse);  }  // calculates the color when using a directional light.  vec3 CalcLightSource(LightSource light, vec3 lightNormal, vec3 vertexPosition, vec3 viewDirection)  {  float intensity = 50.0f;  vec3 ambient;  vec3 diffuse;  vec3 specular;  // Calculate light direction and distance  vec3 lightDirection = normalize(light.position - vertexPosition);  float distance = length(light.position - vertexPosition);  // === Hardcoded attenuation factors ===  float constant = 1.0;  float linear = 0.00014;  float quadratic = 0.00003;  float attenuation = 1.0 / (constant + (linear \* distance) + ( quadratic \* distance \* distance) );  // Ambient  ambient = light.ambientColor \* Ka;  // Diffuse  float impact = max(dot(lightNormal, lightDirection), 0.0);  diffuse = impact \* light.diffuseColor \* intensity;  // Specular  vec3 reflectDir = reflect(-lightDirection, lightNormal);  float specularComponent = pow(max(dot(viewDirection, reflectDir), 0.0), 32.0f);  if(bUseNormAndSpec)  {  specular = specularComponent \* (light.specularIntensity \* Ns) \* light.specularColor \* intensity \* texture(specMap, fragmentTextureCoordinate).rgb;  }  else{  specular = specularComponent \* (light.specularIntensity \* Ns) \* light.specularColor \* intensity;  }  // Apply attenuation  diffuse \*= attenuation;  specular \*= attenuation;  return (ambient + diffuse + specular);  }  fragmentShader.glsl line 115 |
| 72 75, 78 | LO1: Implementation of students own functions to replace glm functions (e.g., glm::length(), glm::dot(), glm::cross() etc.). | **Replacement Functions For Glm Utility functions Are Added in Maths.hpp**  Vec3 Vec3::cross(const Vec3& rhs) const {  return {  y \* rhs.z - z \* rhs.y,  z \* rhs.x - x \* rhs.z,  x \* rhs.y - y \* rhs.x  };  }  float Vec3::dot(const Vec3& rhs) const  {  return x \* rhs.x + y \* rhs.y + z \* rhs.z;  }  Vec3 Vec3::normalize() const  {  float length = std::sqrt(x \* x + y \* y + z \* z);  return { x / length, y / length, z / length };  }  float toRadians(float degrees)  {  return degrees \* (PI / 180.0f);  }  Mat4 Multiply(const Mat4& a, const Mat4& b)  {  Mat4 result;  for (int col = 0; col < 4; ++col) {  for (int row = 0; row < 4; ++row) {  result.cols[col].x = a.cols[0].x \* b.cols[col].x + a.cols[1].x \* b.cols[col].y + a.cols[2].x \* b.cols[col].z + a.cols[3].x \* b.cols[col].w;  result.cols[col].y = a.cols[0].y \* b.cols[col].x + a.cols[1].y \* b.cols[col].y + a.cols[2].y \* b.cols[col].z + a.cols[3].y \* b.cols[col].w;  result.cols[col].z = a.cols[0].z \* b.cols[col].x + a.cols[1].z \* b.cols[col].y + a.cols[2].z \* b.cols[col].z + a.cols[3].z \* b.cols[col].w;  result.cols[col].w = a.cols[0].w \* b.cols[col].x + a.cols[1].w \* b.cols[col].y + a.cols[2].w \* b.cols[col].z + a.cols[3].w \* b.cols[col].w;  }  }  return result;  }  Mat4 Identity() {  return Mat4{  Vec4{ 1.0f, 0.0f, 0.0f, 0.0f },  Vec4{ 0.0f, 1.0f, 0.0f, 0.0f },  Vec4{ 0.0f, 0.0f, 1.0f, 0.0f },  Vec4{ 0.0f, 0.0f, 0.0f, 1.0f }  };  }  Mat4 Translate(const Mat4& m, const Vec4& v)  {  Mat4 result = m;  result.cols[3].x += m.cols[0].x \* v.x + m.cols[1].x \* v.y + m.cols[2].x \* v.z;  result.cols[3].y += m.cols[0].y \* v.x + m.cols[1].y \* v.y + m.cols[2].y \* v.z;  result.cols[3].z += m.cols[0].z \* v.x + m.cols[1].z \* v.y + m.cols[2].z \* v.z;  result.cols[3].w += m.cols[0].w \* v.x + m.cols[1].w \* v.y + m.cols[2].w \* v.z;  return result;  }  Maths.cpp Line 3 |
| LO1: Implementation of quaternions to calculate rotation matrix. | **A Definition Of Quaternions Is Added In Maths.h This Is used in Rotations Of Camera Mode 2 acessed By toggling with Key 3**  struct Quat  {  float x, y, z, w;  Quat() : x(0), y(0), z(0), w(1) {}  Quat(float x, float y, float z, float w) : x(x), y(y), z(z), w(w) {}  Quat normalize() const  {  float len = std::sqrt(x \* x + y \* y + z \* z + w \* w);  return { x / len, y / len, z / len, w / len };  }  };  Maths.h Line 68 |
| 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.). | **Users Can Toggle Lighting With Key Board Presses 1 And 2 But Also Mouse Button press LEFT and Right for Controlled Toggles**  glfwSetMouseButtonCallback(window, (GLFWmousebuttonfun)[](GLFWwindow\* window, int button, int action, int mod)  {  if (button == GLFW\_MOUSE\_BUTTON\_1)  {  if (action == GLFW\_PRESS)  {  ToggleLight1 = !ToggleLight1;  }  }  if (button == GLFW\_MOUSE\_BUTTON\_2)  {  if (action == GLFW\_PRESS)  {  ToggleLight2 = !ToggleLight2;  }  }  });  glfwSetKeyCallback(window, (GLFWkeyfun)[](GLFWwindow\* window, int button, int scancode, int action, int mod)  {  if (button == GLFW\_KEY\_3)  {  if (action == GLFW\_PRESS)  {  CameraType++;  if (CameraType >= 3)  {  CameraType = 0;  }  }  }  });  Coursework.cpp Line 85 |
| LO3: Appropriate implementation of normal and specular maps. | Normal And Specular Maps Are used to Give the Wooden Deck A More Realistic Look  vec3 phongResult = vec3(0.0f);  vec3 lightNormal = normalize(fragmentVertexNormal);  vec3 viewDirection = normalize(viewPosition - fragmentPosition);  if(bUseNormAndSpec)  {  // obtain normals from normals map in range [0,1]  lightNormal = texture(normalMap, fragmentTextureCoordinate).rgb;    // transform normals vector to range [-1,1]  lightNormal = normalize(lightNormal \* 2.0 - 1.0);    lightNormal = (TBN \* lightNormal);//transformation to tangent space  }  **Fragmentshader.glsl line 51**  if(bUseNormAndSpec)  {  specular = specularComponent \* (light.specularIntensity \* Ns) \* light.specularColor \* intensity \* texture(specMap, fragmentTextureCoordinate).rgb;  }  else{  specular = specularComponent \* (light.specularIntensity \* Ns) \* light.specularColor \* intensity;  }  **Fragmentshader.glsl line 172** |
| 85, 90, 100 | LO1: Use of quaternions to calculate view matrix. | Quaternions Are Used In Camera Class to Calculate the View Matrix Which is Limited In Movement Since Its Not Designed to Use Lookat Like a first-person Camera which is Also implemented  Mat4 GetViewMatrixQuat()  {  Quat Xrot(0.0f, 1.0f, 0.0f, Yaw);  Quat Yrot(1.0f, 0.0f, 0.0f, Pitch);  Mat4 rotation = QuaternionToMatrix((Xrot)); // inverse of the camera's rotation  Mat4 rotation2 = QuaternionToMatrix((Yrot)); // inverse of the camera's rotation    Mat4 translation = Translate(Identity(), Vec4(-Position.x, -Position.y, -Position.z, 1.0f)); // inverse of the camera's position  Mat4 view = Multiply( Multiply( rotation, rotation2 ) , translation);  return view;  }  **Camera.hpp Line 80** |
| 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. | Uniforms Used In Manipulating Visual Effects During Runtime  uniform bool bUseNormAndSpec=false;  uniform bool bUseTexture=true;  uniform bool bUseLighting=true;  uniform bool toggleLight1=true;  uniform bool toggleLight2=true;  uniform vec4 objectColor = vec4(1.0f);    **fragmentshader.glsl line 34** |