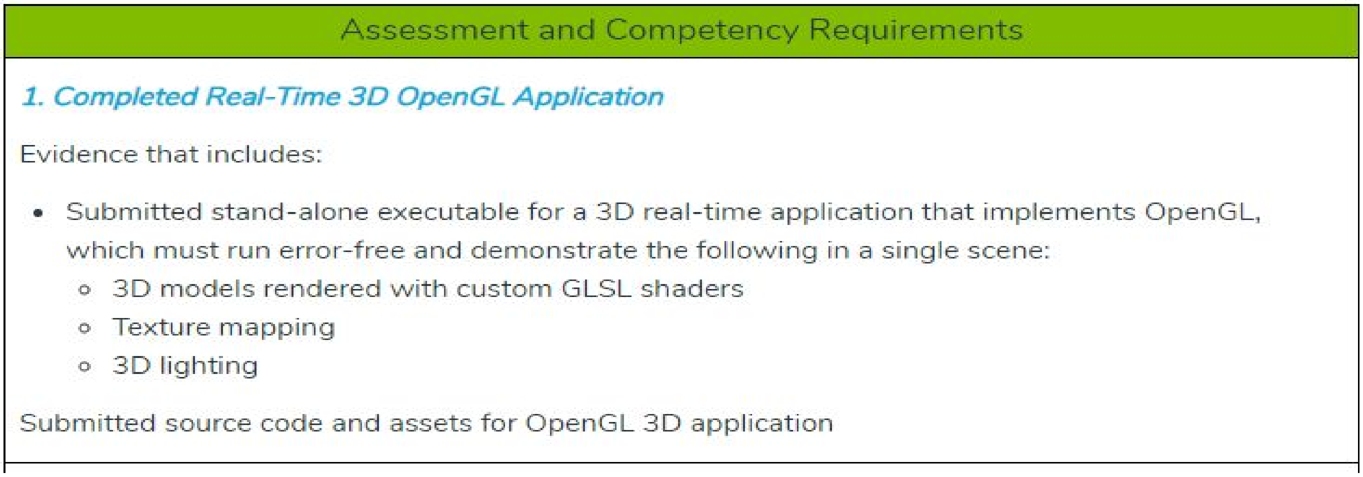
Name: Ralenski Doucet
Contents: Computer Graphics Competency Documentation



Assessment and Competency Requirements

1. Completed Real-Time 3D OpenGL Application

Evidence that includes:

- Submitted stand-alone executable for a 3D real-time application that implements OpenGL, which must run error-free and demonstrate the following in a single scene:
  - 3D models rendered with custom GLSL shaders
  - Texture mapping
  - 3D lighting

Submitted source code and assets for OpenGL 3D application

- **How I explain what texture mapping is:**

Texture mapping is a graphic design process in which a(2-D) surface, called a texture map, is "wrapped around" a (3-D)object.Thus, the 3-D object acquires a surface texture similar to that of the 2-D surface. Texture mapping is just like the applying of wallpaper, paint,or veneer to a real object.

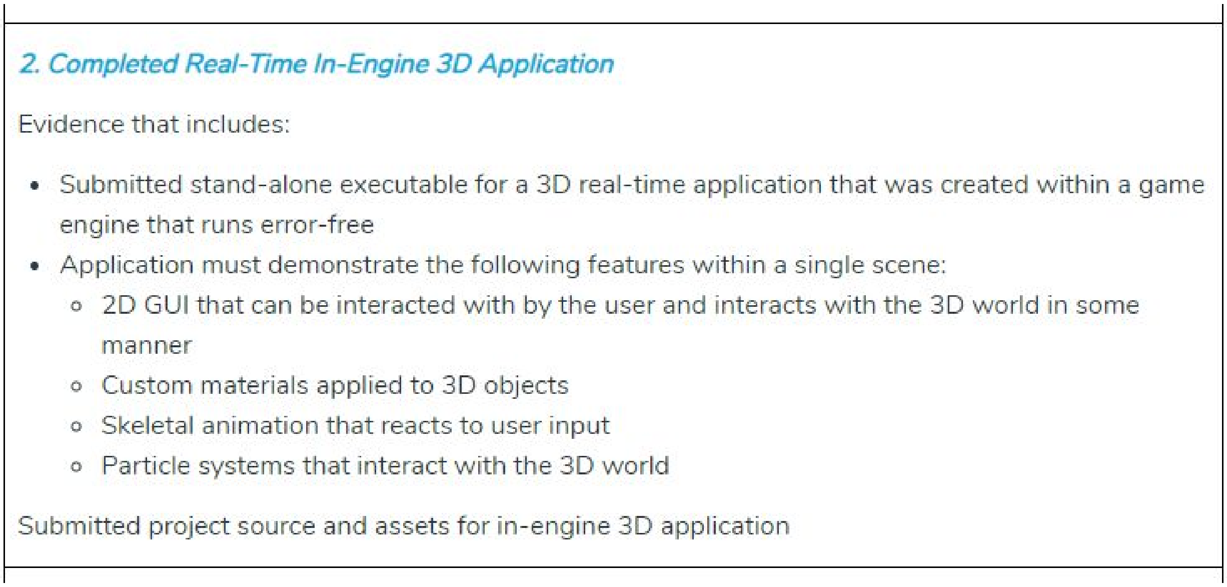
- **How to code texture mapping:**

1st: You need to take in a vector 2 for texture coordinates.
2nd: You need a uniform sampler that is 2-D.
***Uniform variables are used to communicate with your vertex or fragment shader from outside". In your shader you use the uniform qualifier to declare the variable:**
uniform variables are read-only and have the same value among all processed vertices. You can only change them within your C++ program.*
3rd: Than you need to output a vector4 that is going to be the pixels color.

- **How I Explain what 3d Lighting:**

There are 4 types of Lights used in games. They include: Ambient, Point, Directional, and Spot lights.
The normal for a light is the direction of the surface a light is shining on is facing .
The direction of the normal is used to calculate the direction of the light.A normal is usually a normalised unit vector. To Be able to code Lighting you need to understand the formulas.
The elements used in the formula are as follows:
-K refers to the surface material property colours (ambient, diffuse, specular)
– **I** refers to the light properties (ambient, diffuse, specular)
– **N** is the surface normal vector
– **Lm** is the light direction, the Incident ray
– **Rm** is the light’s Reflected ray
– **V** is a view direction that represents a ray from the surface to the camera
– **A** is a specular power used to control the sharpness of specular reflection

Name: Ralenski Doucet
Contents: Computer Graphics Competency Documentation



2. Completed Real-Time In-Engine 3D Application

Evidence that includes:

- Submitted stand-alone executable for a 3D real-time application that was created within a game engine that runs error-free
- Application must demonstrate the following features within a single scene:
  - 2D GUI that can be interacted with by the user and interacts with the 3D world in some manner
  - Custom materials applied to 3D objects
  - Skeletal animation that reacts to user input
  - Particle systems that interact with the 3D world

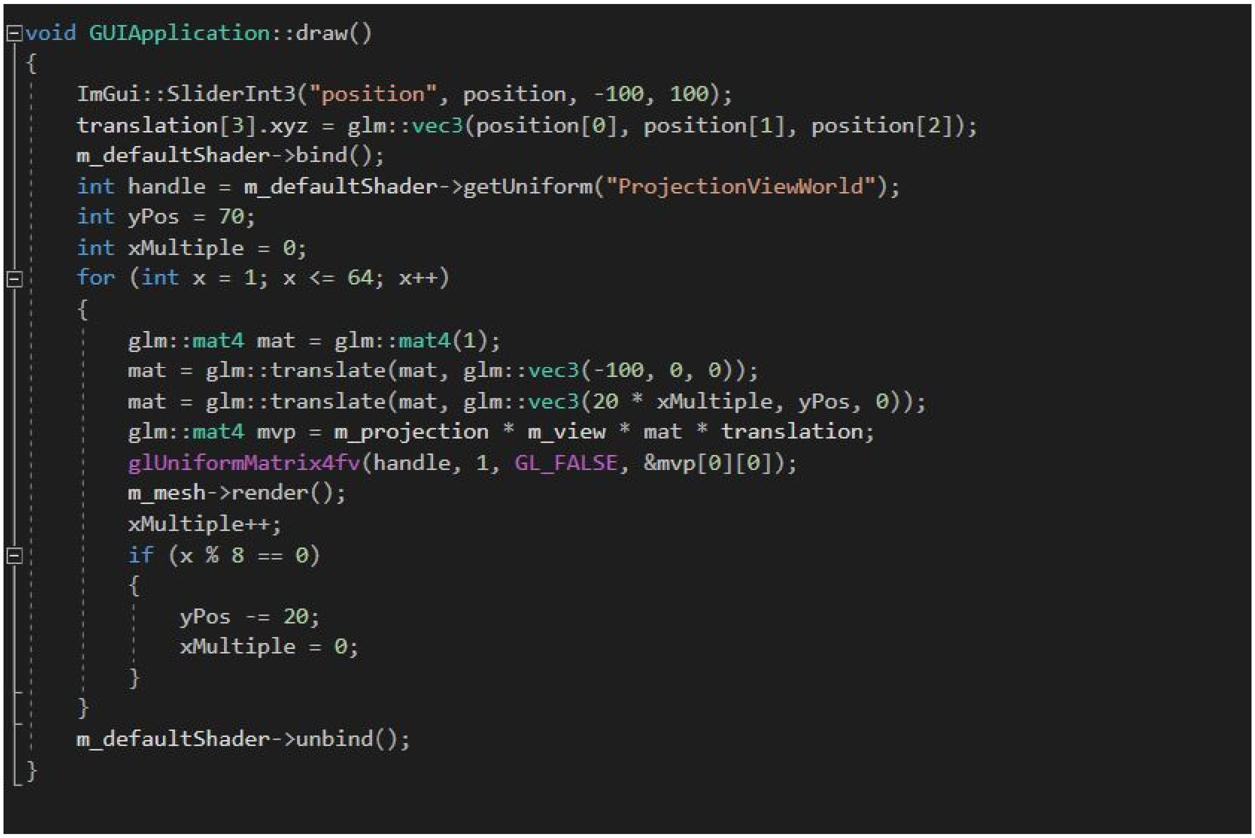
Submitted project source and assets for in-engine 3D application

- A Pictures of my code that shows that I should be able to make a 2-D UI System

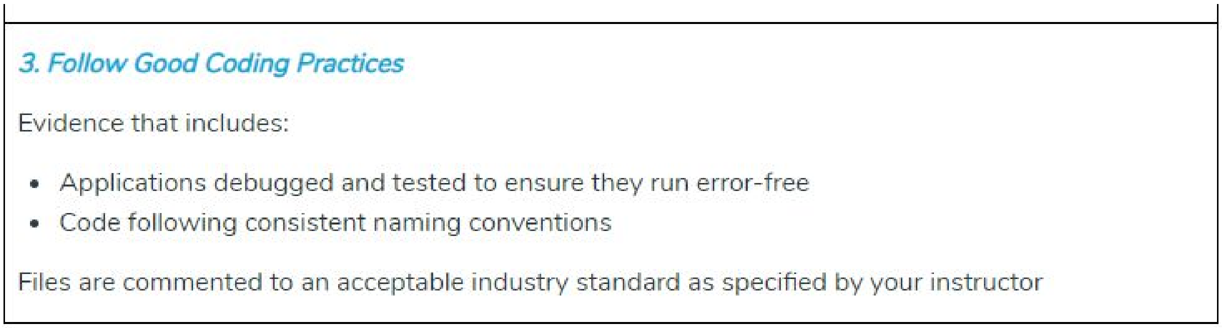
```
int position[3] = { 0,0,0 };
void GUIApplication::startup()
{
    m_mesh = new MeshRenderer();
    m_defaultShader = new Shader();
    m_transform = new Transform();
    std::vector<unsigned int> indices = { 0 ,1,2,2,3,0 };
    std::vector<MeshRenderer::Vertex> vertexs;
    m_transform->SetModel(glm::mat4(1));
    m_mesh->initialize(indices, vertexs);
    m_defaultShader->load("vertex.vert", Shader::SHADER_TYPE::VERTEX);
    m_defaultShader->load("fragment.frag", Shader::SHADER_TYPE::FRAGMENT);
    m_defaultShader->attach();
}

void GUIApplication::shutdown()
{
}

void GUIApplication::update(float dt)
{
    m_transform->SetModel(glm::mat4(1));
    glm::vec3 eye = glm::vec3(0, -10, 200);
    m_view = glm::lookAt(eye, glm::vec3(0), glm::vec3(0, 1, 0));
    m_projection = glm::perspective(glm::quarter_pi<float>(), 800 / (float)600, 0.1f, 1000.f);
}
```



```
void GUIApplication::draw()
{
    ImGui::SliderInt3("position", position, -100, 100);
    translation[3].xyz = glm::vec3(position[0], position[1], position[2]);
    m_defaultShader->bind();
    int handle = m_defaultShader->getUniform("ProjectionViewWorld");
    int yPos = 70;
    int xMultiple = 0;
    for (int x = 1; x <= 64; x++)
    {
        glm::mat4 mat = glm::mat4(1);
        mat = glm::translate(mat, glm::vec3(-100, 0, 0));
        mat = glm::translate(mat, glm::vec3(20 * xMultiple, yPos, 0));
        glm::mat4 mvp = m_projection * m_view * mat * translation;
        glUniformMatrix4fv(handle, 1, GL_FALSE, &mvp[0][0]);
        m_mesh->render();
        xMultiple++;
        if (x % 8 == 0)
        {
            yPos -= 20;
            xMultiple = 0;
        }
    }
    m_defaultShader->unbind();
}
```



3. Follow Good Coding Practices

Evidence that includes:

- Applications debugged and tested to ensure they run error-free
- Code following consistent naming conventions

Files are commented to an acceptable industry standard as specified by your instructor