CITS3003 Project Report

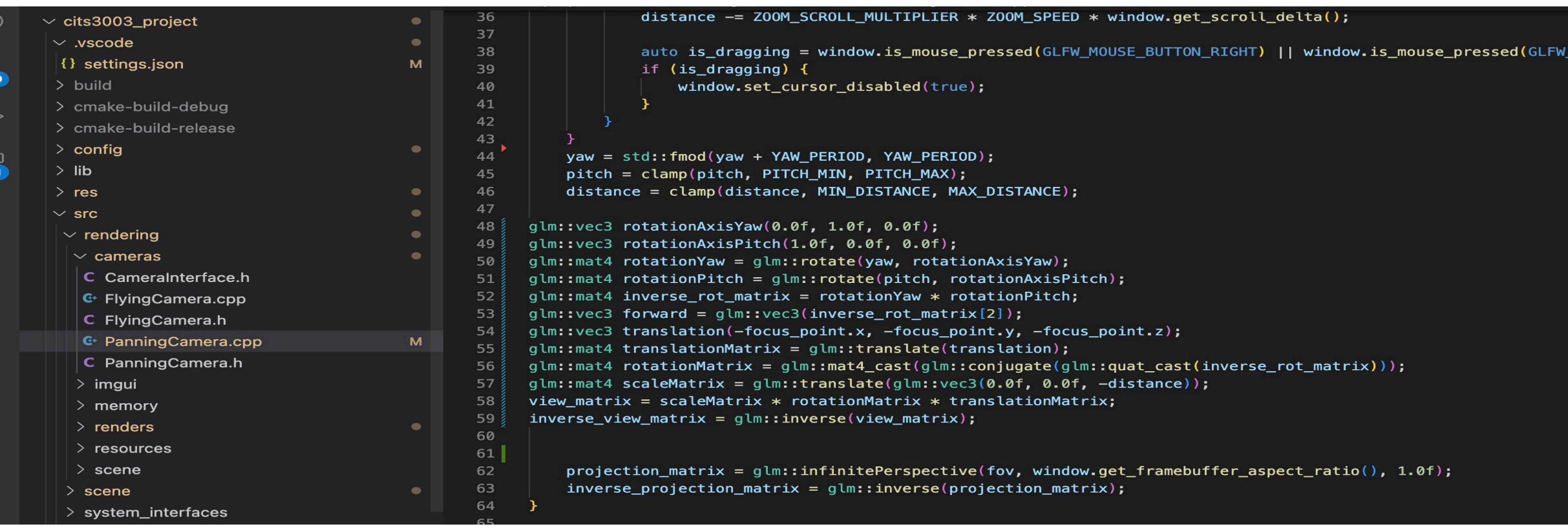
Group: 17
Zihang Fu(23349119)
Rongjin Chen(22771494)

We've done a–i, and for i we've added item textures reflection which is not implemented

Task A

Through the combination of the rotation matrix and the translation matrix, the camera rotation, translation and the setting of the viewing point are realized, so as to calculate the view matrix and the inverse view matrix. These matrices can be used to make camera observations and coordinate transformations in rendering.

Change in line 48–58.



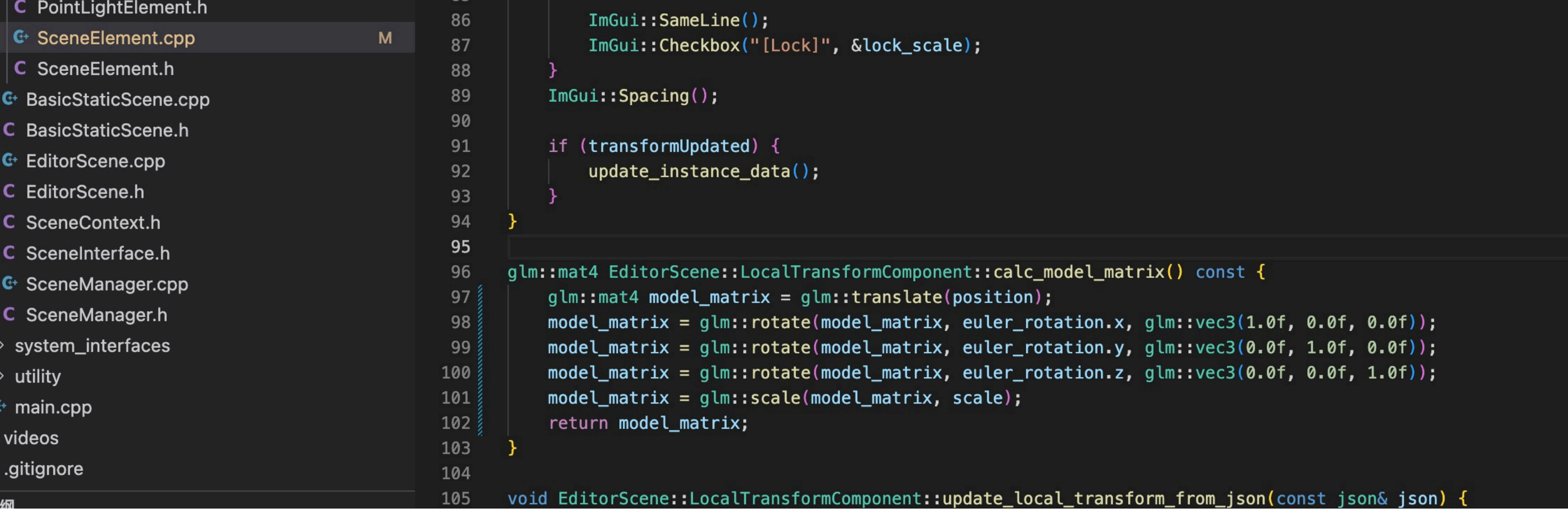
1. Create a rotation matrix that rotates **yaw angles** around the **Y-axis**. Create a rotation matrix that rotates the **pitch Angle** around the **X-axis**. The rotation matrices of yaw and pitch are multiplied to obtain the rotation matrix of the camera, **inverse_rot_matrix**.
2. The forward direction of the camera is extracted from the third column of **inverse_rot_matrix** (forward), A reverse translation vector translation is created by inverting the coordinates of focus_point, and then a **translationMatrix** is created to translate the object from the viewpoint position to the origin.
3. **inverse_rot_matrix** is converted to a quaternion, then its conjugate quaternion is calculated, and finally the conjugate quaternion is converted back to the rotation matrix. The purpose of this step is to obtain the **rotationMatrix** of the camera. (A quaternion contains both real and imaginary parts. The conjugate operation negates the imaginary part of the quaternion while leaving the real part unchanged. For quaternions representing rotations, the conjugate operation can be understood as

reversing the direction of the rotation axis. A quaternion representing the opposite rotation can be obtained by computing the conjugate quaternion. This conjugate quaternion is converted back to rotation matrix form and used to represent rotation operations)

4. Create a **scaleMatrix**, that scales the object along the negative z-axis in the viewing space. Matrix multiplication is performed in order to obtain the final **view_matrix**, which represents the camera viewing pose and position in the scene.

Task B

The idea is to apply translation, rotation, and scale transformations using functions from the glm library, and then apply these transformations to the model matrix in turn. Finally, the function returns the computed model matrix. The model matrix is a 4x4 matrix used to transform the local coordinate system of the object into the world coordinate system.



Change in line 97–102.

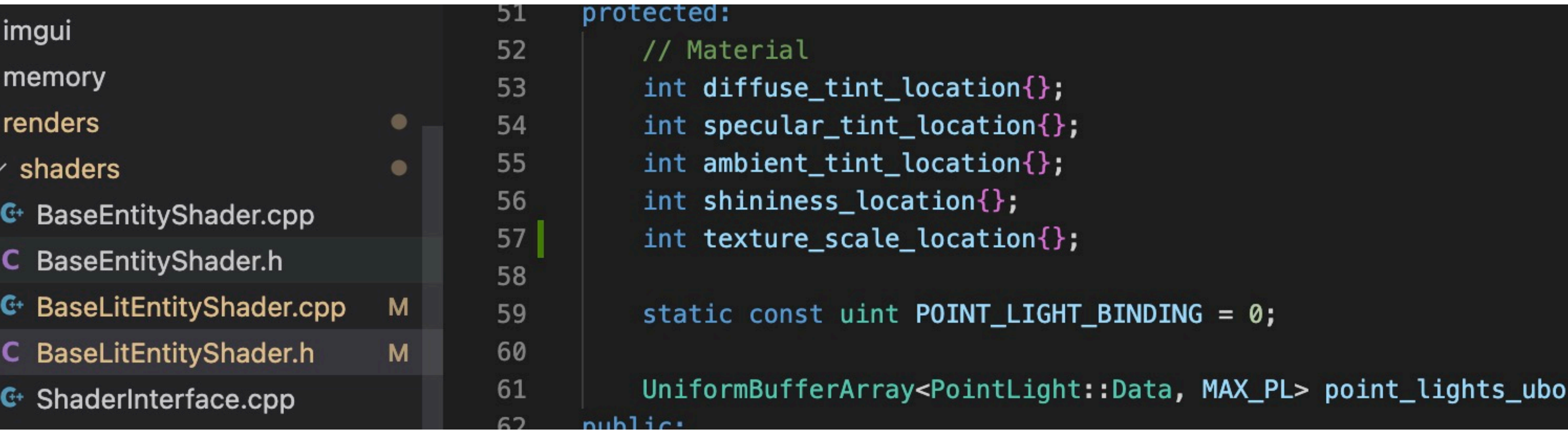
1. Translation: The glm::translate() function is used to translate the model from the origin to a given position.
2. Rotation: Perform a rotation transformation around the x, y and z-axis, calculated using the glm::rotate() function and the rotation vector represented by the **Euler Angle**.
3. Scale: The glm::scale() function is used to change the size of the model given the scale vector.

Task C

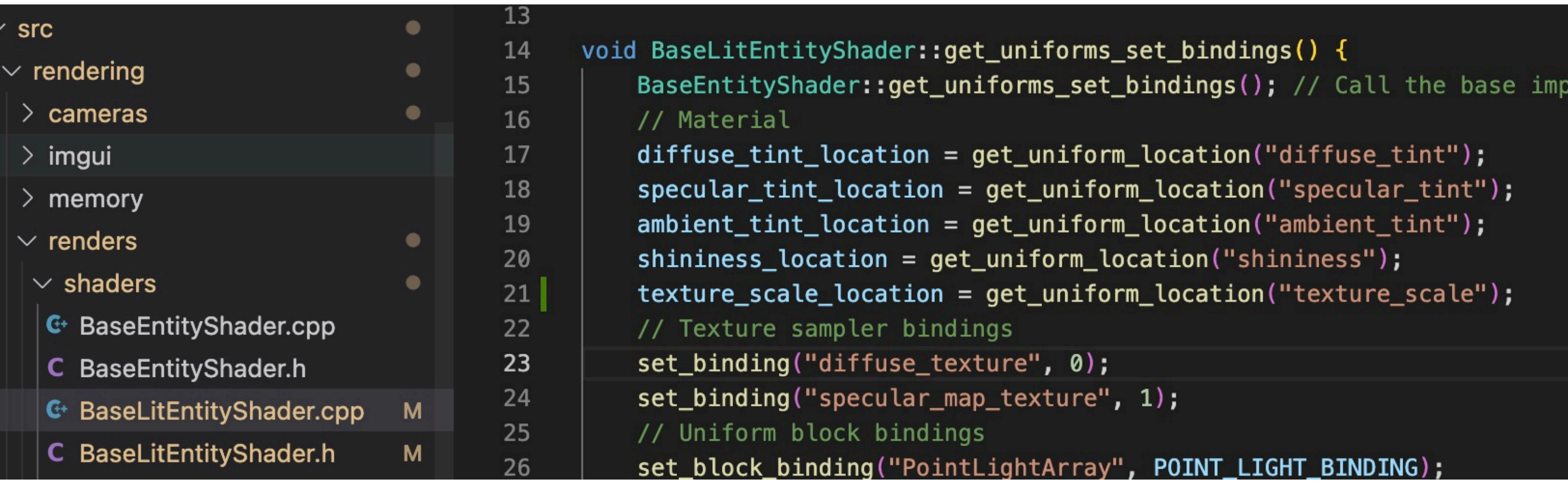
Look at how the other material properties (e.g., shininess) are implemented. Complete the creation of texture_scale by learning.

1. Add the variable **texture_scale**. A floating point number representing the scaling factor of the texture.(add in line 27)

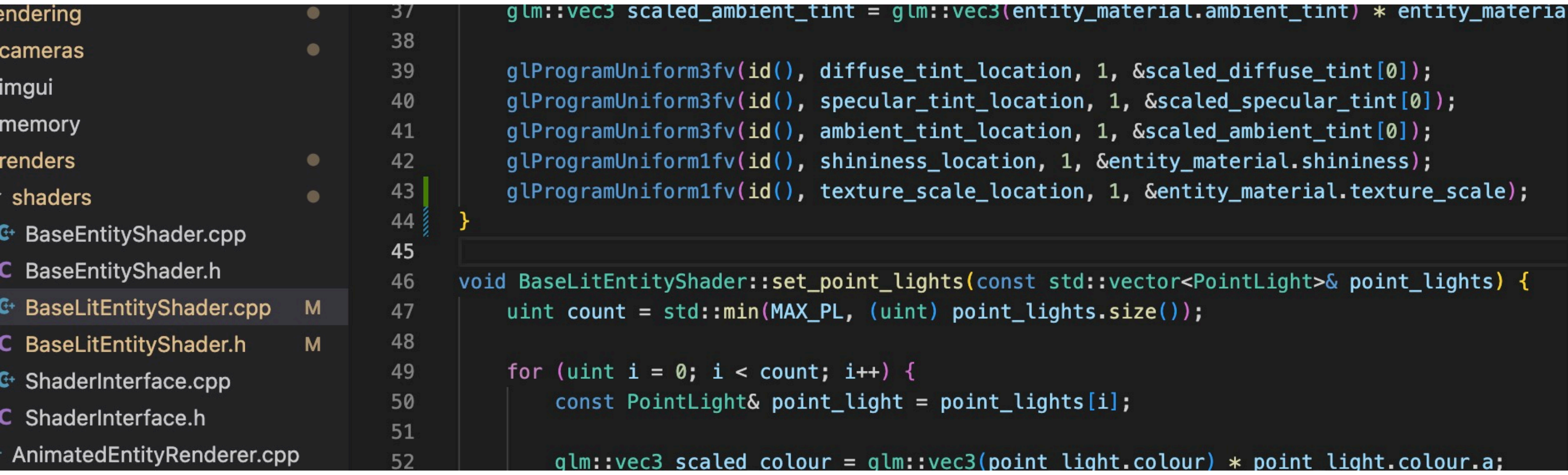
2. Create **texture_scale_location**, Used to store the location of the associated uniform variable in the corresponding shader program.(add in line 57)



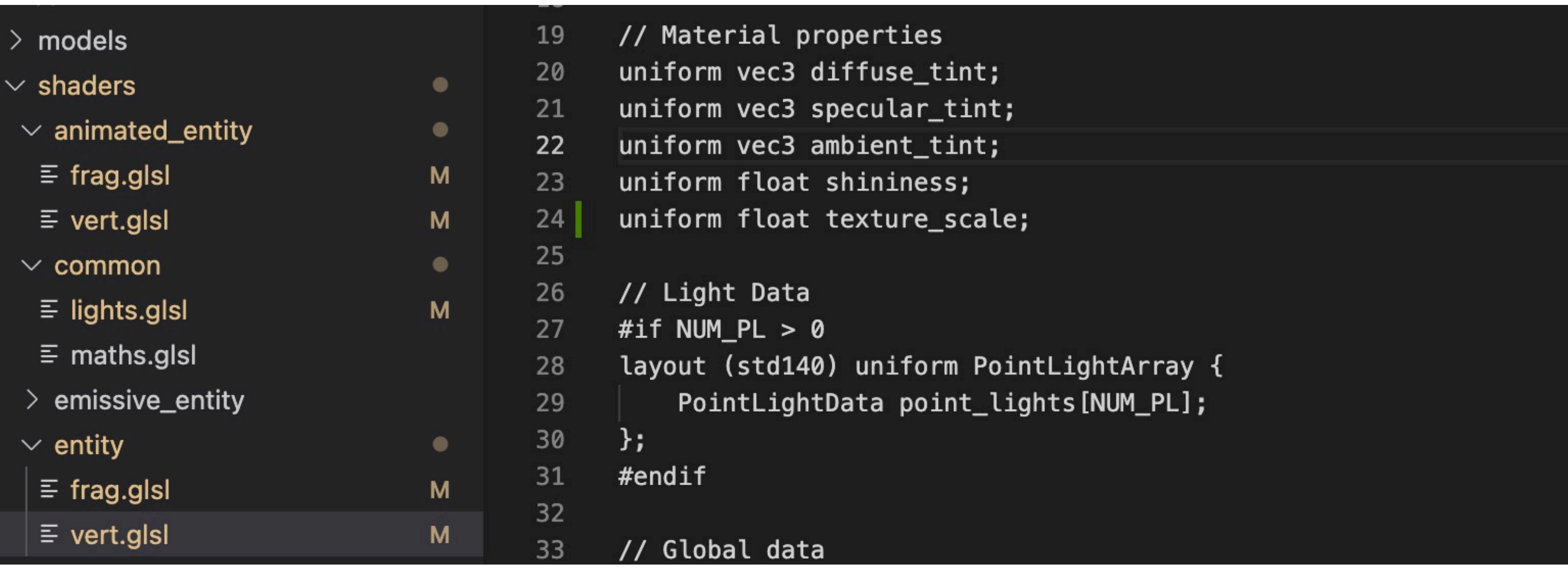
3. Gets the location of the corresponding uniform variable. In order to properly pass data to the shader during rendering. (add in line 21)



4. The call passes the **texture_scale** value of the material to the texture_scale uniform variable in the shader. **&entity_material.texture_scale** represents the memory address of the texture_scale value. (add in line 43)



5. Uniform float texture_scale. (add in line 24)



6. By multiplying the texture coordinates with the scaling factor, the size and density of the texture on the surface of the object can be controlled so

that the area covered by the texture on the surface of the object is increased. (change in line 41)

```
36
37  void main() {
38      // Transform vertices
39      vertex_out.ws_position = (model_matrix * vec4(vertex_position, 1.0f)).xyz;
40      vertex_out.ws_normal = normalize(normal_matrix * normal);
41      vertex_out.texture_coordinate = texture_coordinate * texture_scale;
42
43      gl_Position = projection_view_matrix * vec4(vertex_out.ws_position, 1.0f);
44      }
45
```

7. Add UI control. (add in line 124)

```
120 id EditorScene::LitMaterialComponent::add_material_imgui_edit_section(MasterRenderScene& /*r
121   // Set this to true if the user has changed any of the material values, otherwise the chan
122   bool material_changed = false;
123   ImGui::Text("Material");
124   material_changed = ImGui::DragFloat("Texture Scale", &material.texture_scale, 0.01f);
125   material_changed |= ImGui::ColorEdit3("Diffuse Tint", &material.diffuse_tint[0]);
126   ImGui::Spacing();
127   material_changed |= ImGui::DragFloat("Diffuse Factor", &material.diffuse_tint[3], 0.01f, 0
128   material_changed |= ImGui::ColorEdit3("Specular Tint", &material.specular_tint[0]);
129   ImGui::Spacing();
```

8. Add a **texture_scale** member to the child object **material** of **json**, store the value of the variable in the corresponding field, to complete the **save** and **load** functions. (add in line 150 and line 159)

```
145     auto m = json["material"];
146     material.diffuse_tint = m["diffuse_tint"];
147     material.specular_tint = m["specular_tint"];
148     material.ambient_tint = m["ambient_tint"];
149     material.shininess = m["shininess"];
150     material.texture_scale = m["texture_scale"]; //json 文件存储进m变量
151 }
152
153 json EditorScene::LitMaterialComponent::material_into_json() const {
154     return {"material", {
155         {"diffuse_tint", material.diffuse_tint},
156         {"specular_tint", material.specular_tint},
157         {"ambient_tint", material.ambient_tint},
158         {"shininess", material.shininess},
159         {"texture_scale", material.texture_scale},//增加json 文件格式
160     }};
161 }
162
```

Task D

Add UI controls for adjusting the ambient, diffuse, emissive, and specular tints, as well as the shine amount, at this point in the code.

Add in line 125–134

1. creates a colour picker control for adjusting the diffuse tint of the material.(125) creates a draggable float control for adjusting the diffuse factor of the material.(127) creates a color picker control for adjusting the specular tint of the material.(128) creates a draggable float control for adjusting the specular factor of the material. (130)creates a color picker control for

adjusting the ambient tint of the material.(131) creates a draggable float control for adjusting the ambient factor of the material.(133) creates a draggable float control for adjusting the shininess of the material.(134)

2. For example, line 125 **&material.diffuse_tint[0]** is used to pass a pointer to the diffuse tint colour array to the ImGui control so that it can modify the values of the colour directly. In line 127, &material.diffuse_tint[3] is taking the address of the fourth element ([3]) of the diffuse_tint array in the material object, it represents the diffuse factor value. 0.01f is the speed or increment by which the value can be dragged. 0.0f is the minimum value allowed for the drag control. FLT_MAX is the maximum value allowed for the drag control. FLT_MAX is a constant that represents the maximum finite floating-point value supported by the system.

Task E

Before using float near in the projection matrix, the near value was fixed at 1.0f, but in the update function, we need to change the near variable that

we defined earlier. The synchronization variable enables the sliding block to be controlled.
Both camera files(flyingcamera and panning camera) need to be changed, since the cameras all are projectional camera.

```
ImGui::SliderFloat("Near Plane", &near, 0.001f, 1.0f, "%.3f", ImGuiSliderFlags_Logarithmic);
```

```
projection_matrix = glm::infinitePerspective(fov, window.get_framebuffer_aspect_ratio(), near);// near
inverse_projection_matrix = glm::inverse(projection_matrix);
```

Task F

The light source attenuation is calculated according to the light source attenuation formula in light.glsl, as shown below at no.47 line. We could get attenuation variable. The dynamic light rendering feedback is then obtained by multiplying the ambient light, reflection variable, and diffuse variable by the visual distance.

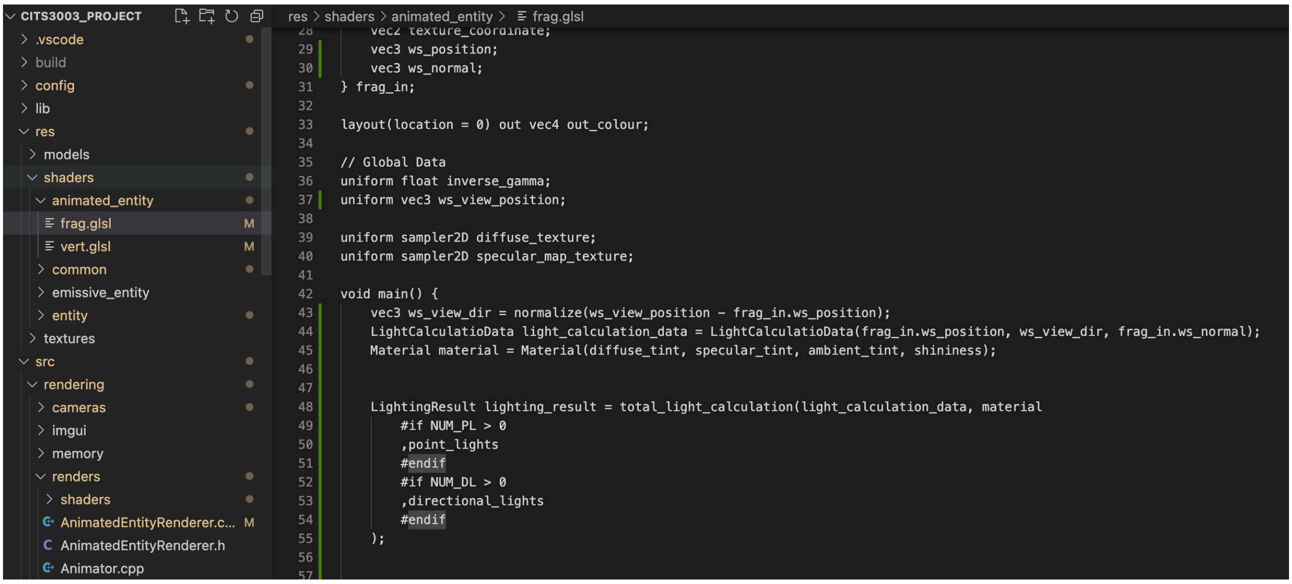
```
39 // Point Lights, distance of point light
40 void point_light_calculation(PointLightData point_light, LightCalculatioData calculation_data, float shininess, inout vec3 total_
41     vec3 ws_light_offset = point_light.position - calculation_data.ws_frag_position;
42 
43     //Add this to calculate the distance between the light source and the fragment
44     float distance = length(ws_light_offset);
45 
46     // Calculate the attenuation based on the distance
47     float attenuation = 1.0 / (distance * distance);
48 
49     // Ambient * attenuation 乘以视距
50     vec3 ambient_component = ambient_factor * point_light.colour * attenuation;
51 
52     // Diffuse
53     vec3 ws_light_dir = normalize(ws_light_offset);
54     float diffuse_factor = max(dot(ws_light_dir, calculation_data.ws_normal), 0.0f);
55     vec3 diffuse_component = diffuse_factor * point_light.colour* attenuation;
56 
57     // Specular
58     vec3 ws_halfway_dir = normalize(ws_light_dir + calculation_data.ws_view_dir);
59     float specular_factor = pow(max(dot(calculation_data.ws_normal, ws_halfway_dir), 0.0f), shininess);
60     vec3 specular_component = specular_factor * point_light.colour * attenuation;
```

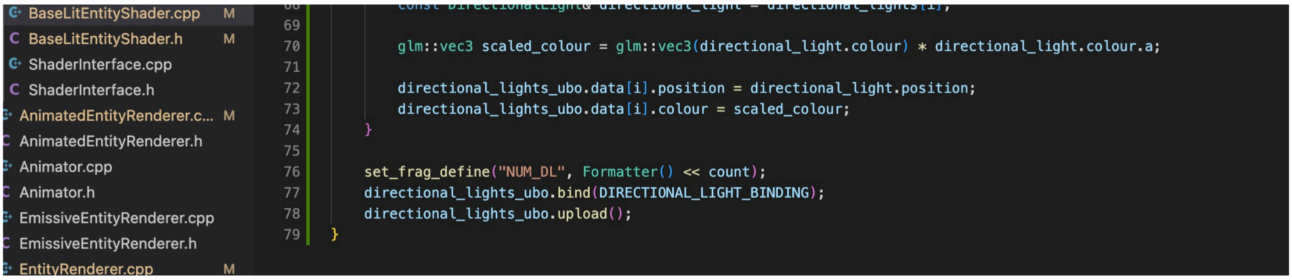
Task G

To change the light rendering from vertex shader to fragment shader, we need to change the light rendering pipeline from vertex shader to fragment shader, and finally change the vertex rendering function call to fragment.

First, copy the global variables of the uniform class into the fragment shader, because that's where these variables will be used in the rendering pipeline. The vertex output variable is then passed to the fragment shader, where the vertex shader functions that perform the ray rendering are placed.

```
in VertexOut {
    vec2 texture_coordinate;
    vec3 ws_position;
    vec3 ws_normal;
} frag_in;

layout(location = 0) out vec4 out_colour;

// Global Data
uniform float inverse_gamma;

uniform sampler2D diffuse_texture;
uniform sampler2D specular_map_texture;

void main() {
    vec3 ws_view_dir = normalize(ws_view_position - frag_in.ws_position);
    LightCalculatioData light_calculation_data = LightCalculatioData(frag_in.ws_position, ws_view_dir, frag_in.ws_normal);
    Material material = Material(diffuse_tint, specular_tint, ambient_tint, shininess);

     LightingResult lighting_result = total_light_calculation(light_calculation_data, material
        #if NUM_PL > 0
        ,point_lights
        #endif
        #if NUM_DL > 0
        ,directional_lights
        #endif
    );
```





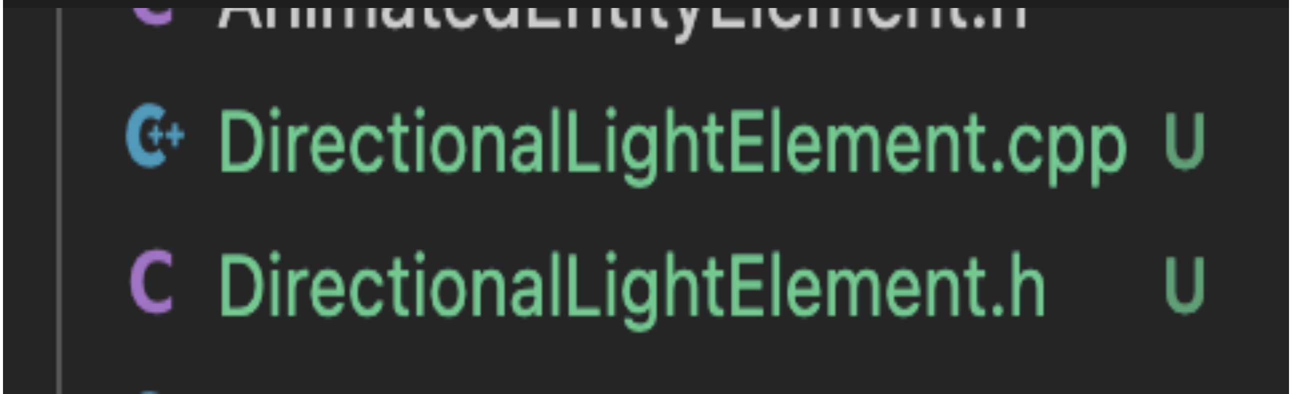
Task H

Because we want to add a new directional light source, so in light.glsl, the calculation of the point light source should be calculated again for the directional light source, but because the directional light source does not need to calculate the vector position of the light source offset, meaning that the directional light source is a fixed vector of light, so its offset should be normalised.

Because we want to operate through the ui, we need to re-execute the operation logic of the point light in our newly created cpp and head files, and finally add their execution logic to editorscene.cpp. At the same time, we need to enable the shader to enable the directional light in baselitentityshader.cpp.

```
/// All the                                              e registered here to be
light_gener
    {PointLightElement::ELEMENT_TYPE_NAME, [](const SceneContext& scene_cont
    {DirectionalLightElement::ELEMENT_TYPE_NAME, [](const SceneContext& scen
};

/// All the element generators, new element types must be registered here to
json_generators = {
    {EntityElement::ELEMENT_TYPE_NAME,           [](const SceneContext& scene_
    {AnimatedEntityElement::ELEMENT_TYPE_NAME,   [](const SceneContext& scene_
    {EmissiveEntityElement::ELEMENT_TYPE_NAME,   [](const SceneContext& scene_
    {PointLightElement::ELEMENT_TYPE_NAME,       [](const SceneContext& scene_
    {GroupElement::ELEMENT_TYPE_NAME,            [](const SceneContext&, Eleme
    {DirectionalLightElement::ELEMENT_TYPE_NAME, [](const SceneContext& scen
};
```

没有找到结果，请点击“更多释义”详细查询




```
//dl
void directional_light_calculation(DirectionalLightData directional_light, LightCalculatioData calculation_data, fl
    vec3 ws_light_offset = normalize(directional_light.direction);

    //Add this to calculate the distance between the light source and the fragment
    float distance = length(ws_light_offset);

    // Calculate the attenuation based on the distance
    float attenuation = 1.0 / (distance * distance);

    // Ambient * attenuation 乘以视距
    vec3 ambient_component = ambient_factor * directional_light.colour * attenuation;

    // Diffuse
    vec3 ws_light_dir = ws_light_offset;
    float diffuse_factor = max(dot(ws_light_dir, calculation_data.ws_normal), 0.0f);
    vec3 diffuse_component = diffuse_factor * directional_light.colour* attenuation;

    // Specular
    vec3 ws_halfway_dir = normalize(ws_light_dir + calculation_data.ws_view_dir);
    float specular_factor = pow(max(dot(calculation_data.ws_normal, ws_halfway_dir), 0.0f), shininess);
    vec3 specular_component = specular_factor * directional_light.colour * attenuation;

    total_diffuse += diffuse_component;
    total_specular += specular_component;
    total_ambient += ambient_component;
}
```

Task I

Add a button to reset the Material values and Add a button to reset the Transformation values.

```cpp
// Add a button to reset the values
if (ImGui::Button("Reset Material")) {
    // Reset the material values to their initial state
    material.texture_scale = 1.0f;
    material.diffuse_tint = { 1.0f, 1.0f, 1.0f, 1.0f };
    material.specular_tint = { 1.0f, 1.0f, 1.0f, 1.0f };
    material.ambient_tint = { 1.0f, 1.0f, 1.0f, 1.0f };
    material.shininess = 128;
    material_changed = true; // Set the material as changed
}
```

```cpp
// Add a button to reset the values
if (ImGui::Button("Reset Transformation")) {
    position = glm::vec3(0.0f, -0.01f, 0.0f);
    euler_rotation = glm::vec3(0.0f);
    scale = glm::vec3(10.0f,1.0f,10.0f);
    transformUpdated = true;
}
```

In emissiveentityshader, modify two shaders to implement the reflected light function.

```
void main() {
    vec3 texture_colour = texture(emissive_texture, frag_in.texture_co
    vec3 emissive_colour = emissive_tint * texture_colour;

    // 计算视线方向
    vec3 view_direction = normalize(-frag_in.ws_position);

    // 计算光照方向
    vec3 direction = normalize(light_direction); // 光源方向

    // 计算法线
    vec3 normal = normalize(frag_in.ws_normal);

    // 计算反射方向
    vec3 reflection_direction = reflect(direction, normal);

    // 计算反光颜色
    float specular = pow(max(dot(reflection_direction, view_direction)

    // 最终颜色 = 发光颜色 + 反光颜色
    vec3 final_colour = emissive_colour + specular;

    out_colour = vec4(final_colour, 1.0);
    out_colour.rgb = pow(out_colour.rgb, vec3(inverse_gamma));
}
```

```
    vec3 ws_normal;
    vec2 texture_coordinate;
} vertex_out;

// Per instance data
uniform mat4 model_matrix;

// Global data
uniform mat4 projection_view_matrix;
uniform vec3 diffuse_tint;
uniform vec3 specular_tint;
uniform vec3 ambient_tint;
uniform float shininess;

void main() {
    vertex_out.ws_position = (model_matrix * vec4(vertex_position, 1.0f)).xyz;
    vertex_out.ws_normal = mat3(transpose(inverse(model_matrix))) * vertex_normal;
    vertex_out.texture_coordinate = texture_coordinate;

    gl_Position = projection_view_matrix * vec4(vertex_out.ws_position, 1.0f);
}
```