1. 实现阴影的方法：阴影映射

阴影映射：

以光的位置为视角进行渲染

要点：

（1）第一次渲染：将待渲染物体坐标转换到以光的位置为视角的坐标系下，计算其深度值，将深度值通过深度缓冲存储在深度纹理中，由于深度测试，纹理最后输出的值是最近深度。

```
// configure depth map FBO
// -----------------------
const unsigned int SHADOW_WIDTH = 1024, SHADOW_HEIGHT = 1024;
unsigned int depthMapFBO;
glGenFramebuffers(1, &depthMapFBO);
// create depth texture
unsigned int depthMap;
glGenTextures(1, &depthMap);
glBindTexture(GL_TEXTURE_2D, depthMap);
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, SHADOW_WIDTH, SHADOW_HEIGHT, 0, GL_DEPTH_COMPONENT, GL_FLOAT, NULL)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
// attach depth texture as FBO's depth buffer
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, depthMap, 0);
glDrawBuffer(GL_NONE);
glReadBuffer(GL_NONE);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

上图代码创建一个深度纹理并绑定到帧缓冲中。

```
glViewport(0, 0, SHADOW_WIDTH, SHADOW_HEIGHT);
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
glClear(GL_DEPTH_BUFFER_BIT);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, woodTexture);
renderScene(simpleDepthShader);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

上图是第一次渲染，将基本的深度信息写入深度纹理

（2）第二次渲染：将待渲染物体在以光的位置为视角的坐标系下深度值与深度纹理中深度值做对比，如果小于最近深度值，则在光照下，否则在光照外，则要设置成阴影的样子。该阶段渲染进行正常的纹理映射。

```
glViewport(0, 0, SCR_WIDTH, SCR_HEIGHT);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
shader.use();
glm::mat4 projection = glm::perspective(glm::radians(camera.Zoom), (float)SCR_WIDTH / (float)SCR_HEIGHT, 0.1f, 100.0f);
glm::mat4 view = camera.GetViewMatrix();
shader.setMat4("projection", projection);
shader.setMat4("view", view);
// set light uniforms
shader.setVec3("viewPos", camera.Position);
shader.setVec3("lightPos", lightPos);
shader.setMat4("lightSpaceMatrix", lightSpaceMatrix);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, woodTexture);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, depthMap);
renderScene(shader);

// render Depth map to quad for visual debugging
```

上图是第二次渲染，传入投影矩阵等信息进行光照和深度计算，正常绑定纹理映射进行

渲染。

（3）第二次渲染的片段着色器：若在阴影内，则只保留散射值，若在阴影外，则正常计算光照

```
// calculate shadow
float shadow = ShadowCalculation(fs_in.FragPosLightSpace);
vec3 lighting = (ambient + (1.0 - shadow) * (diffuse + specular)) * color;
```

上图通过 ShadowCalculation 计算深度结果，如果在阴影内就为 1，计算光照时只保留散射，否则正常计算。

```
float ShadowCalculation(vec4 fragPosLightSpace)
{
    // perform perspective divide
    vec3 projCoords = fragPosLightSpace.xyz / fragPosLightSpace.w;
    // transform to [0,1] range
    projCoords = projCoords * 0.5 + 0.5;
    // get closest depth value from light's perspective (using [0,1] range fragPosLight as coords)
    float closestDepth = texture(shadowMap, projCoords.xy).r;
    // get depth of current fragment from light's perspective
    float currentDepth = projCoords.z;
    // check whether current frag pos is in shadow
    float shadow = currentDepth > closestDepth  ? 1.0 : 0.0;

    return shadow;
}
```

从深度纹理获取最近深度值用来和当前深度值作比较输出 0/1

渲染结果：

Imgui：移动光源位置观察阴影移动,见 gifs