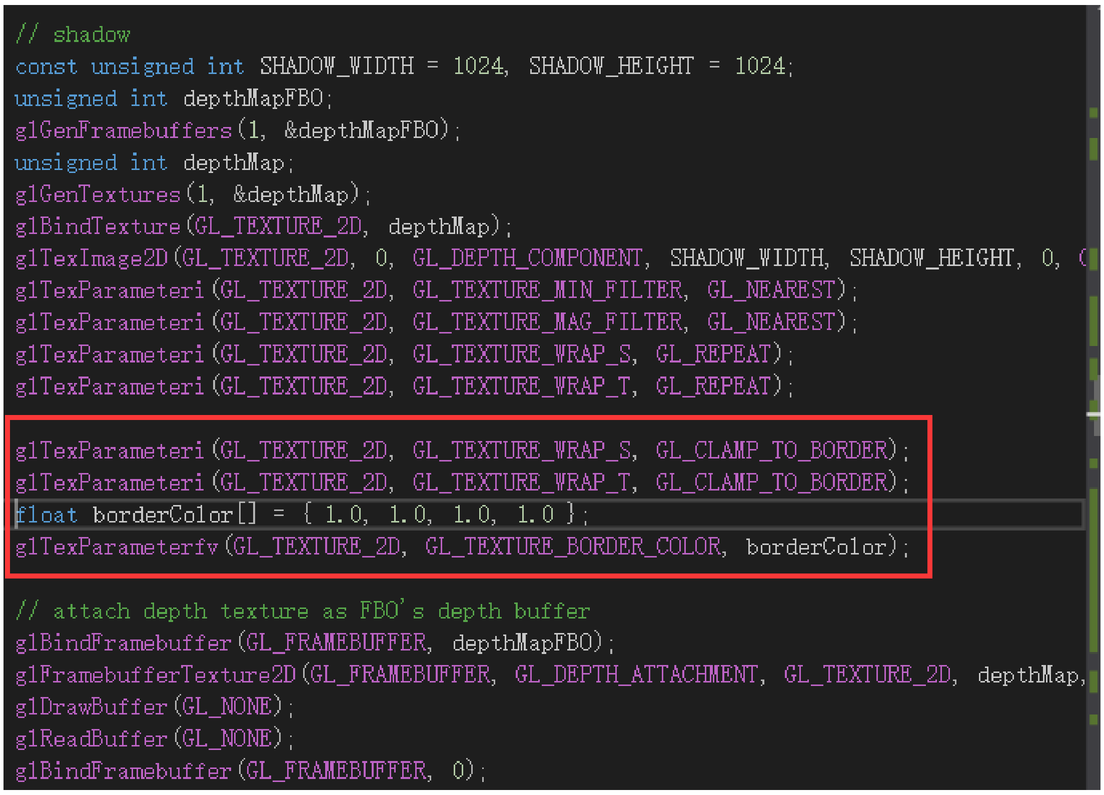
Basic:

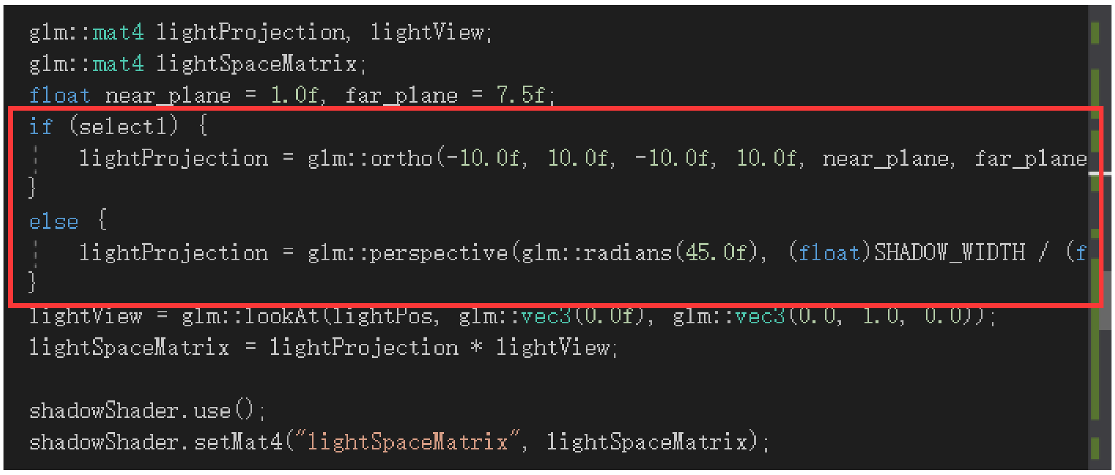
结合代码解释算法：

首先按照教程，创建一个 2D 纹理贴图，图中红框部分代码为 border，用以防止重复渲染：



```
// shadow
const unsigned int SHADOW_WIDTH = 1024, SHADOW_HEIGHT = 1024;
unsigned int depthMapFBO;
glGenFramebuffers(1, &depthMapFBO);
unsigned int depthMap;
glGenTextures(1, &depthMap);
glBindTexture(GL_TEXTURE_2D, depthMap);
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, SHADOW_WIDTH, SHADOW_HEIGHT, 0,
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);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);
float borderColor[] = { 1.0, 1.0, 1.0, 1.0 };
glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, borderColor);

// attach depth texture as FBO's depth buffer
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, depthMap,
glDrawBuffer(GL_NONE);
glReadBuffer(GL_NONE);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

建立一个变换矩阵 lightSpaceMatrix，用以将视角空间转移到光源的视角空间，图中红框代码可以看到实现了 GUI 选择两种不同的投影方式：



```
glm::mat4 lightProjection, lightView;
glm::mat4 lightSpaceMatrix;
float near_plane = 1.0f, far_plane = 7.5f;
if (select1) {
    lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, near_plane, far_plane
}
else {
    lightProjection = glm::perspective(glm::radians(45.0f), (float)SHADOW_WIDTH / (f
}
lightView = glm::lookAt(lightPos, glm::vec3(0.0f), glm::vec3(0.0, 1.0, 0.0));
lightSpaceMatrix = lightProjection * lightView;

shadowShader.use();
shadowShader.setMat4("lightSpaceMatrix", lightSpaceMatrix);
```

修改渲染贴图，存储深度信息，然后将视图改回原来的大小：

```
glViewport(0, 0, SHADOW_WIDTH, SHADOW_HEIGHT);
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
glClear(GL_DEPTH_BUFFER_BIT);
glActiveTexture(GL_TEXTURE);
renderScene(shadowShader);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
glCullFace(GL_BACK);
```

改回：

```
int display_w, display_h;
glfwMakeContextCurrent(window);
glfwGetFramebufferSize(window, &display_w, &display_h);
glViewport(0, 0, SCR_WIDTH, SCR_HEIGHT);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

而阴影的渲染过程在 Shader 中实现：

对上次作业的 phongShader 进行修改，顶点着色器中加入光源视角的位置坐标：

```
uniform mat4 lightSpaceMatrix;

out VS_OUT {
    vec3 FragPos;
    vec3 Normal;
    vec4 FragPosLightSpace;
} vs_out;

void main()
{
    vs_out.FragPos = vec3(model * vec4(aPos, 1.0));
    vs_out.FragPosLightSpace = lightSpaceMatrix * vec4(vs_out.FragPos, 1.0);
    vs_out.Normal = mat3(transpose(inverse(model))) * aNormal;
    gl_Position = projection * view * model * vec4(vs_out.FragPos, 1.0);
}
```

在片段着色器中判断顶点位置是否在阴影里：

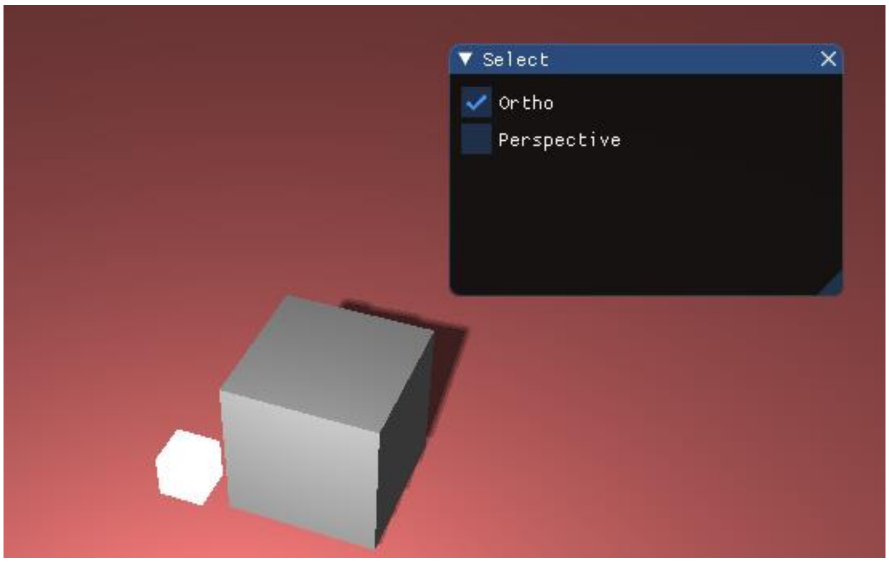
```
//perform perspective divide
vec3 projCoords = FragPosLightSpace.xyz / FragPosLightSpace.w;
//transform to [0.1] range
projCoords = projCoords * 0.5 + 0.5;
float closestDepth = texture(shadowMap, projCoords.xy).r;
float currentDepth = projCoords.z;
```

最后经过计算求得 shadow 分量之后，要加到 result 中：

```
vec3 result = (ambient + (1.0 - shadow) * (diffuse + specular)) * color;
FragColor = vec4(result, 1.0);
```

Bonus:

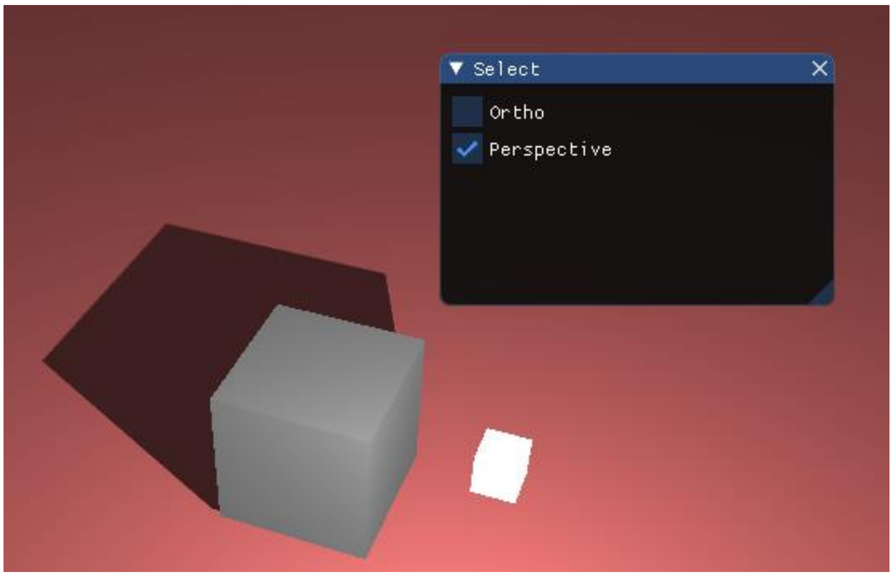
使用 GUI 可修改投影方式，代码在上面已展示，下面是结果：



▼ Select

✓ Ortho

Perspective



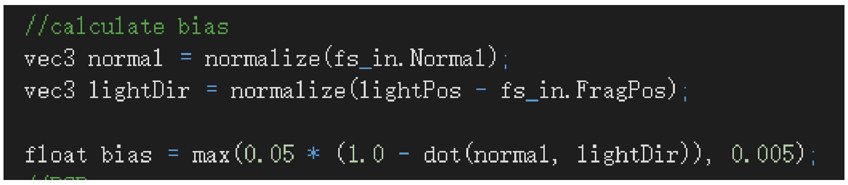
▼ Select

Ortho

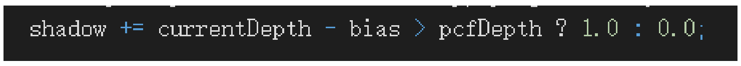
✓ Perspective

优化：

1. 使用 Shadow Bias 解决失真问题，代码在片段着色器中实现：

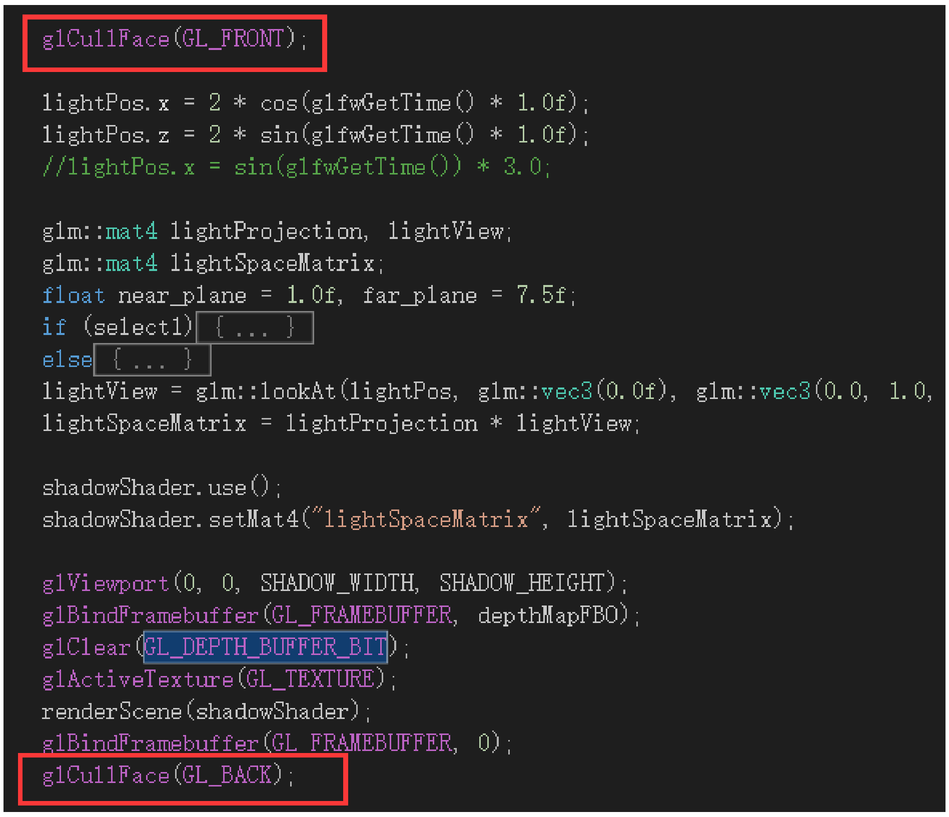


```
//calculate bias
vec3 normal = normalize(fs_in.Normal);
vec3 lightDir = normalize(lightPos - fs_in.FragPos);

float bias = max(0.05 * (1.0 - dot(normal, lightDir)), 0.005);
```



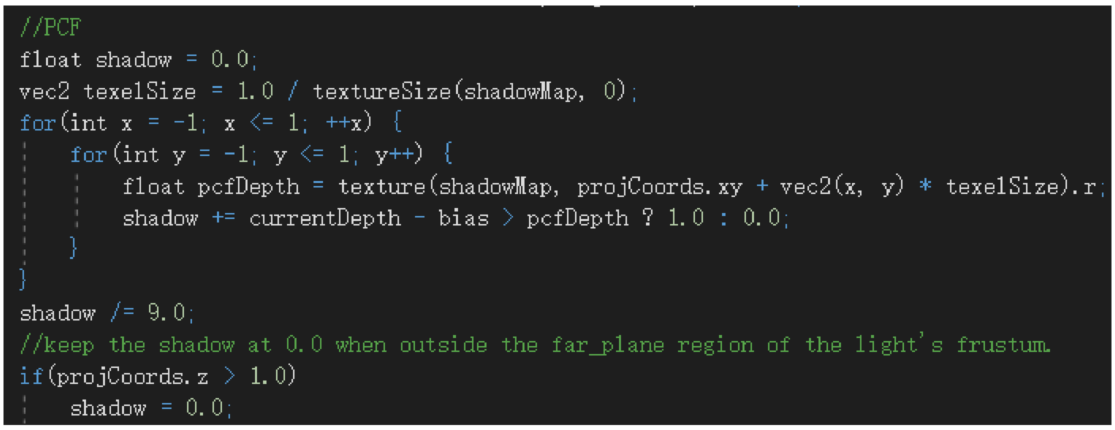
```
shadow += currentDepth - bias > pcfDepth ? 1.0 : 0.0;
```

2. 进行正面剔除，修复 peter panning：



```
glCullFace(GL_FRONT);

lightPos.x = 2 * cos(glfwGetTime() * 1.0f);
lightPos.z = 2 * sin(glfwGetTime() * 1.0f);
//lightPos.x = sin(glfwGetTime()) * 3.0;

glm::mat4 lightProjection, lightView;
glm::mat4 lightSpaceMatrix;
float near_plane = 1.0f, far_plane = 7.5f;
if (select1) { ... }
else { ... }
lightView = glm::lookAt(lightPos, glm::vec3(0.0f), glm::vec3(0.0, 1.0,
lightSpaceMatrix = lightProjection * lightView;

shadowShader.use();
shadowShader.setMat4("lightSpaceMatrix", lightSpaceMatrix);

glViewport(0, 0, SHADOW_WIDTH, SHADOW_HEIGHT);
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
glClear(GL_DEPTH_BUFFER_BIT);
glActiveTexture(GL_TEXTURE);
renderScene(shadowShader);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
glCullFace(GL_BACK);
```

3. 使用 PCF，缓解阴影边缘的锯齿化：



```
//PCF
float shadow = 0.0;
vec2 texelSize = 1.0 / textureSize(shadowMap, 0);
for(int x = -1; x <= 1; ++x) {
    for(int y = -1; y <= 1; y++) {
        float pcfDepth = texture(shadowMap, projCoords.xy + vec2(x, y) * texelSize).r;
        shadow += currentDepth - bias > pcfDepth ? 1.0 : 0.0;
    }
}
shadow /= 9.0;
//keep the shadow at 0.0 when outside the far_plane region of the light's frustum.
if(projCoords.z > 1.0)
    shadow = 0.0;
```