Homework 7 - Shadowing Mapping

Basic：

1、实现方向光源的 Shadowing Mapping：

- 要求场景中至少有一个 object 和一块平面（用于显示 shadow）
- 光源的投影方式任选其一即可
- 在报告里结合代码，解释 Shading Mapping 算法

2、修改 GUI

Bonus：

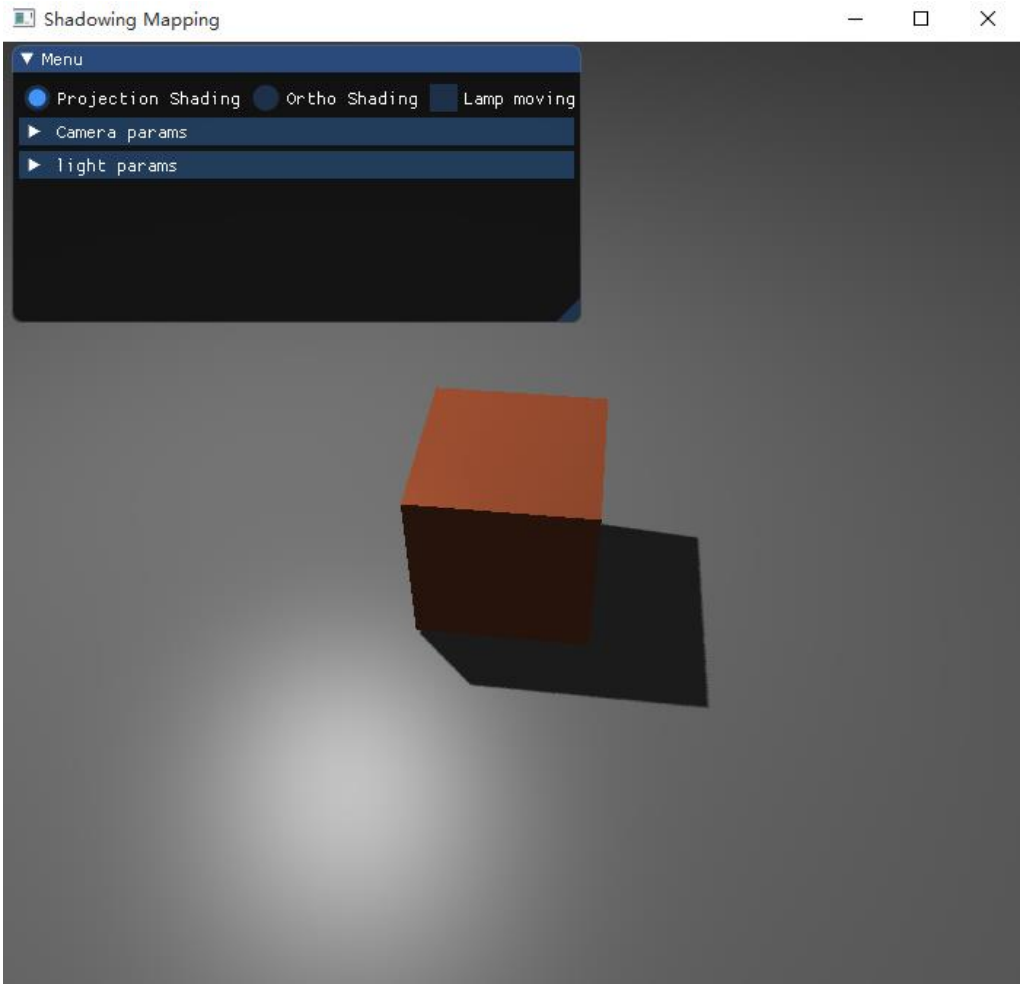
1、实现光源在正交/透视两种投影下的 Shadowing Mapping

2、优化 Shadowing Mapping

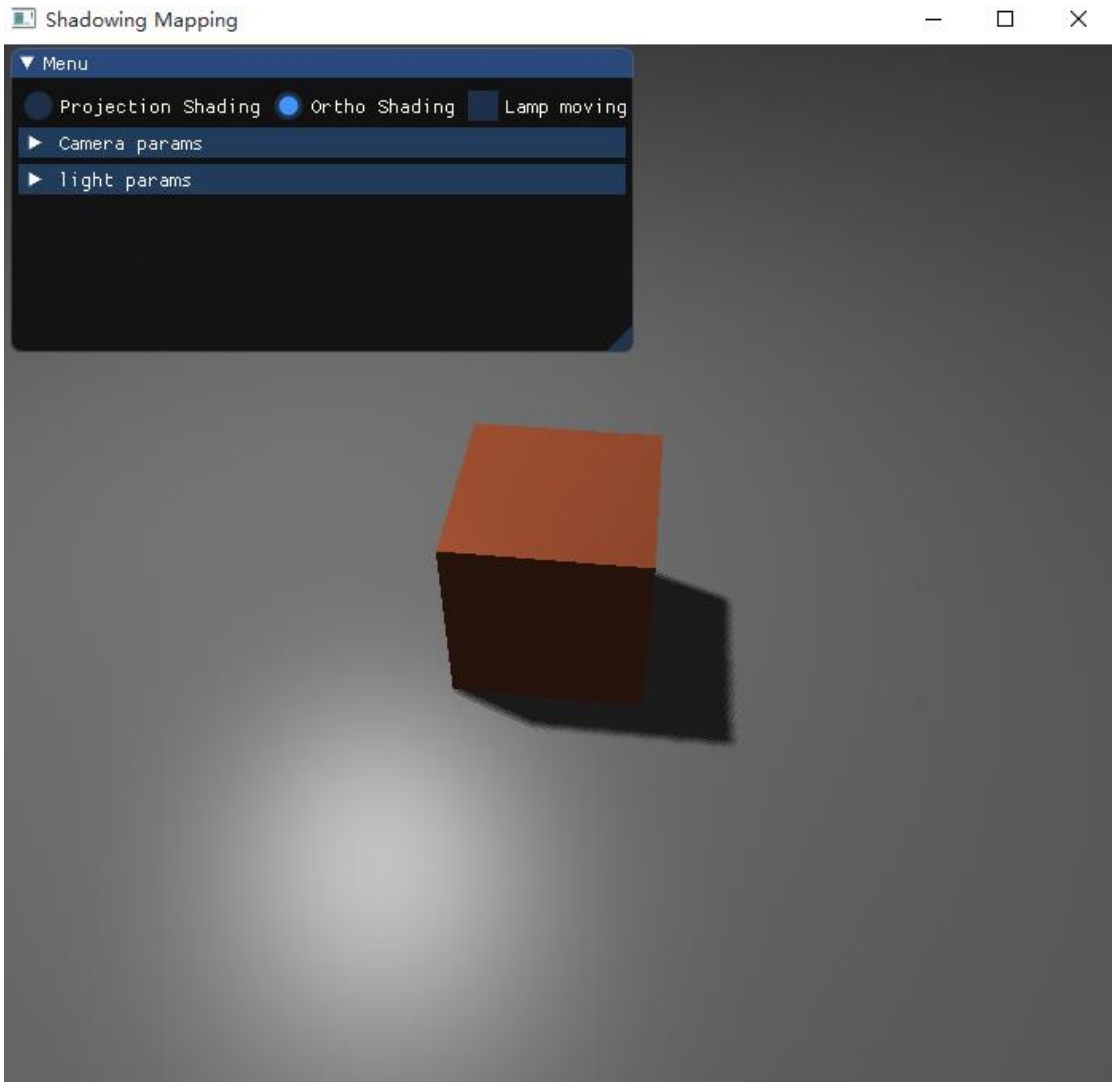
实现结果：

说明：

由于 Bonus 是在 Basic 上的优化，故最终结果为 Basic+Bonus 实现的共同结果，不再展示 Basic 单独的实现结果。具体展示效果见**演示视频.mp4**



透视投影下的阴影渲染



正交投影下的阴影渲染

Basic 部分实现

阴影渲染**两大基本步骤**：

1、 以光源视角渲染场景，得到深度图（DepthMap），并存储为 texture；

2、 以 camera 视角渲染场景，使用 Shadowing Mapping 算法（比较当前深度值与在 DepthMap Texture 的深度值），决定某个点是否在阴影下。

具体实现过程如下：

1、 创建一个 2D 纹理贴图，设置好属性和参数。需要注意，在设置纹理的 GL_REPEAT 参数后，需要补充一个 borderColor 来防止纹理图片重复渲染。

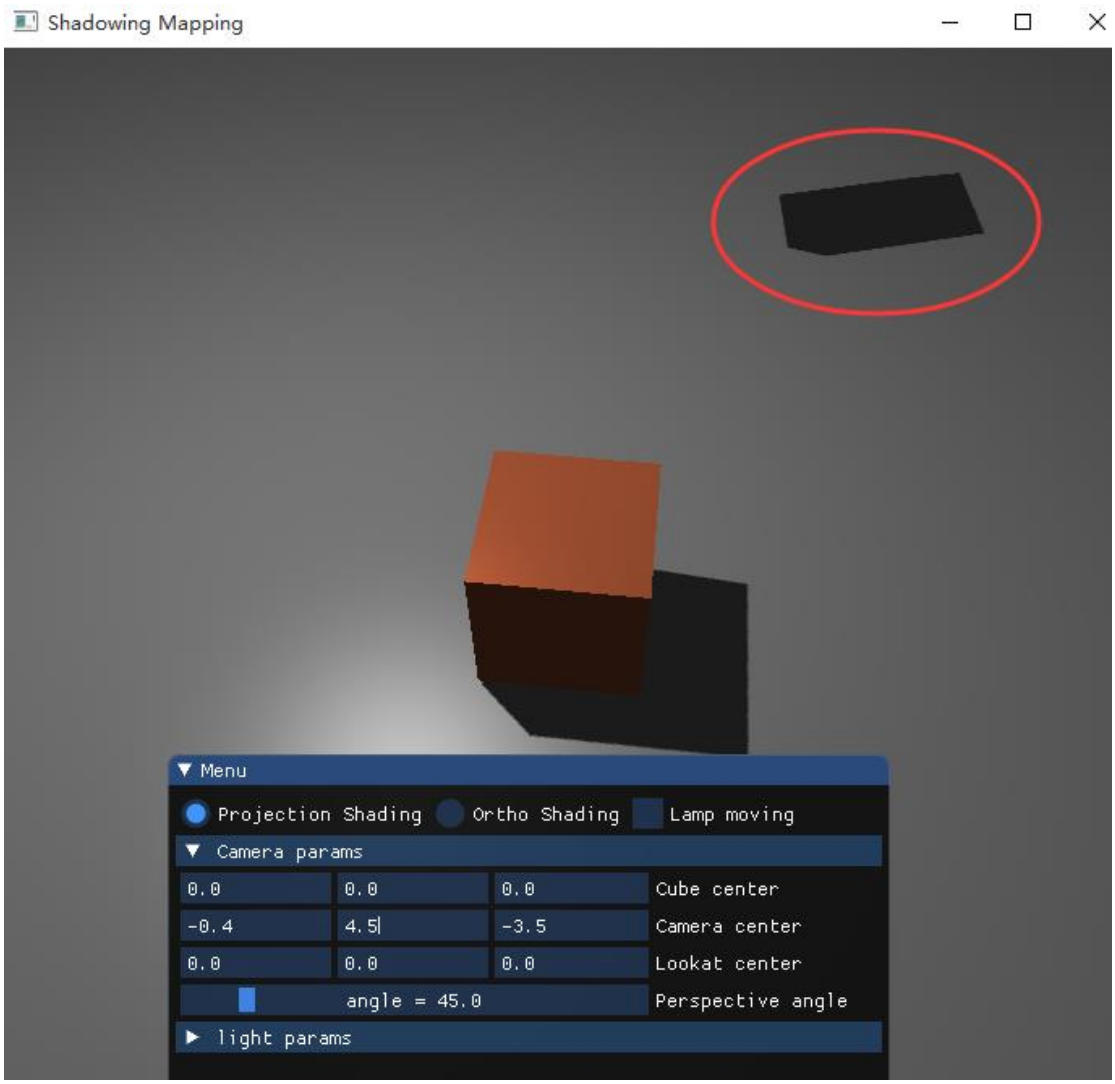
```
//Configure depth map FBO
const GLuint SHADOW_WIDTH = 1024, SHADOW_HEIGHT = 1024;
GLuint depthMapFBO;
glGenFramebuffers(1, &depthMapFBO);
// create depth texture
GLuint depthMap;
glGenTextures(1, &depthMap);
glBindTexture(GL_TEXTURE_2D, depthMap);

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_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

//防止纹理贴图在远处重复渲染
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);
GLfloat borderColor[] = { 1.0, 1.0, 1.0, 1.0 };
glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, borderColor);

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);
```

否则，远处会重复出现一些阴影。



2、 通过一个变换矩阵 lightSpaceMatrix，将视角空间转移到以光源为视角的空间，并可以通过选择正交投影来模拟平行光源，选择透视投影来模拟点光源。

```
//render depth scene to texture (from light's perspective)
//get light projection/view matrix
glCullFace(GL_FRONT);
glm::mat4 lightProjection, lightView;
glm::mat4 lightSpaceMatrix;
GLfloat near_plane = 1.0f, far_plane = 7.5f;
if (mode == 1) {
    //Orthographic
    lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, near_plane, far_plane);
}
else {
    //projection
    lightProjection = glm::perspective(45.0f, (float)SHADOW_WIDTH / (float)SHADOW_HEIGHT, near_plane, far_plane);
}
```

3、 修改视图大小，进行渲染深度纹理贴图，该阶段则可存储好深度信息，而无需着色器操作。然后，将视图改回原屏幕大小，使用深度贴图辅助原视图的渲染。

```
lightView = glm::lookAt(lightPos, glm::vec3(0.0f), glm::vec3(0.0, 1.0, 0.0));
lightSpaceMatrix = lightProjection * lightView;
//render scene from light's point of view
shadow_shader.use();
shadow_shader.setMat4("lightSpaceMatrix", glm::value_ptr(lightSpaceMatrix));

glViewport(0, 0, SHADOW_WIDTH, SHADOW_HEIGHT);
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
glClear(GL_DEPTH_BUFFER_BIT);
glActiveTexture(GL_TEXTURE);
RenderScene(shadow_shader);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
glCullFace(GL_BACK);

// ----------------------------------------
// reset viewport
glViewport(0, 0, WIN_WIDTH, WIN_HEIGHT);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

阴影渲染的判断和操作细节在着色器中完成，**具体过程如下：**

1、 在顶点着色器中，增加并计算顶点位置在光源视角观察的空间中的位置坐标 FragPosLightSpace.

```
vs_out.FragPos = vec3(model * vec4(aPos, 1.0));
vs_out.FragPosLightSpace = lightSpaceMatrix * vec4(vs_out.FragPos, 1.0);
```

2、 在片段着色器中，通过从顶点着色器传过来的 FragPosLightSpace，以及外部程序传进来的 ShadowMap，算出当前像素位置离镜头最近的深度和当前深度，然后判断该点是否在阴影中。

```
//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;
```

3、 计算出出一个表示阴影强度的分量 shadow。

4、 在原始的计算光照的公式中加上阴影。

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

Bonus 部分实现：

1. 实现光源在正交/透视两种投影下的 Shadowing Mapping

只需修改光源观察空间的投影变换矩阵即可。

```
if (mode == 1) {
    //Orthographic
    lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, near_plane, far_plane);
}
else {
    //projection
    lightProjection = glm::perspective(45.0f, (float)SHADOW_WIDTH / (float)SHADOW_HEIGHT, near_plane, far_plane);
}
lightView = glm::lookAt(lightPos, glm::vec3(0.0f), glm::vec3(0.0, 1.0, 0.0));
lightSpaceMatrix = lightProjection * lightView;
```

2. 优化 Shadowing Mapping

改进 1：使用阴影偏移（shadow bias）解决阴影失真(Shadow Acne)问题。

```
float bias = max(0.05 * (1.0 - dot(normal, lightDir)), 0.005);
shadow += currentDepth - bias > pcfDepth ? 1.0 : 0.0;
```

改进 2：进行正面剔除，修复 peter 游移。

```
//get light projection/view matrix
glCullFace(GL_FRONT);
glm::mat4 lightProjection, lightView;
glm::mat4 lightSpaceMatrix;
GLfloat near_plane = 1.0f, far_plane = 7.5f;
if (mode == 1) {
    //Orthographic
    lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, near_plane, far_plane);
}
else {
    //projection
    lightProjection = glm::perspective(45.0f, (float)SHADOW_WIDTH / (float)SHADOW_HEIGHT, near_plane, far_plane);
}
lightView = glm::lookAt(lightPos, glm::vec3(0.0f), glm::vec3(0.0, 1.0, 0.0));
lightSpaceMatrix = lightProjection * lightView;
//render scene from light's point of view
shadow_shader.use();
shadow_shader.setMat4("lightSpaceMatrix", glm::value_ptr(lightSpaceMatrix));

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

改进 3：使用 PCF。PCF 在片段着色器中，对于没一点的 shadow，对其周围的临近点同时进行采样作平均，以达到平滑的效果。这样做可以缓解阴影边缘明显的锯齿化。

```
//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;

return shadow;
```