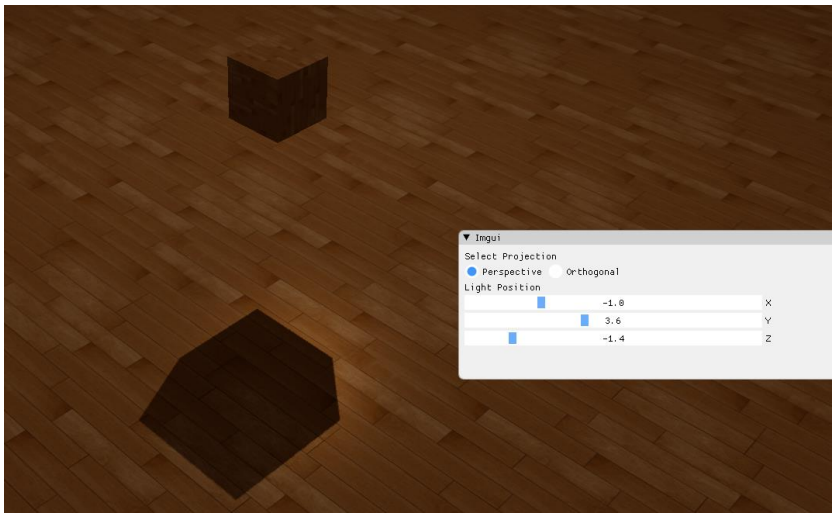
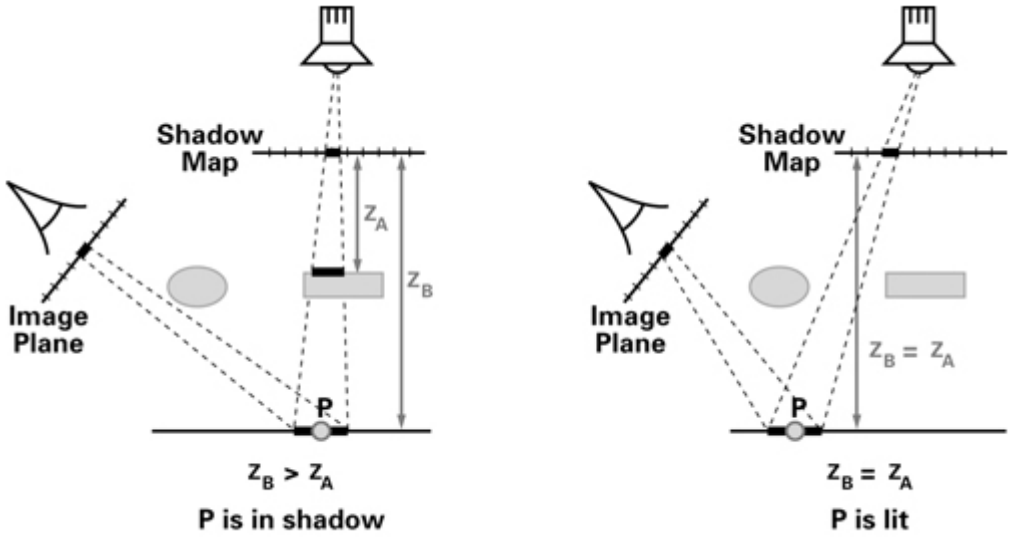
1. 实现方向光源的 Shadowing Mapping:
   要求场景中至少有一个 object 和一块平面(用于显示 shadow)
   光源的投影方式任选其一即可
   在报告里结合代码，解释 Shadowing Mapping 算法。



解释算法：



首先将摄像机设置到光源所在的位置，然后对与景进行第二次绘制.此次绘制将每个可以看见的片元到光源的距离记录到一幅纹理图中的对应像素中。接着将摄像机恢复到实际摄像机所处的位置，绘制场景。此次绘制时将前面步骤产生的纹理采用投影贴图的方式应用到场景中。绘制每个片元时，根据投影贴图纹理采样的结果换算出光源与此片元连线中距光源最近的片元距离(ZA)，再计算出此片元距光源的实际距离(ZB)。若 ZB>ZA，则需要绘制的片元处于阴影中，采用阴影的颜色着色，否则此片元不在阴影中，进行既定的光照着色。

1、创建帧缓冲对象：

```
GLuint depthMapFBO;
glGenFramebuffers(1, &depthMapFBO);
```

2、创建 2D 纹理

```
const GLuint SHADOW_WIDTH = 1024, SHADOW_HEIGHT = 1024;
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_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

3、把生成的深度纹理作为帧缓冲的深度缓冲：

```
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);
```

4、生成深度贴图：

```
glViewport(0, 0, SHADOW_WIDTH, SHADOW_HEIGHT);
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
glClear(GL_DEPTH_BUFFER_BIT);
ConfigureShaderAndMatrices();
RenderScene();
glBindFramebuffer(GL_FRAMEBUFFER, 0);
glViewport(0, 0, SCR_WIDTH, SCR_HEIGHT);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
ConfigureShaderAndMatrices();
glBindTexture(GL_TEXTURE_2D, depthMap);
RenderScene();
```

5、进行光源空间变换

```
glm::mat4 lightProjection, lightView;
glm::mat4 lightSpaceMatrix;
GLfloat near_plane = 1.0f, far_plane = 7.5f;
// Orthographic
lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, near_plane, far_plane);
// Projection
lightProjection = glm::perspective(124.0f, (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;
```

6、渲染至深度贴图

利用 shadow_mapping_depth 的着色器

```
layout (location = 0) in vec3 aPos;
uniform mat4 model;
uniform mat4 lightSpaceMatrix;
void main(){gl_Position = lightSpaceMatrix * model * vec4(aPos, 1.0);}
```

然后进行渲染深度缓冲

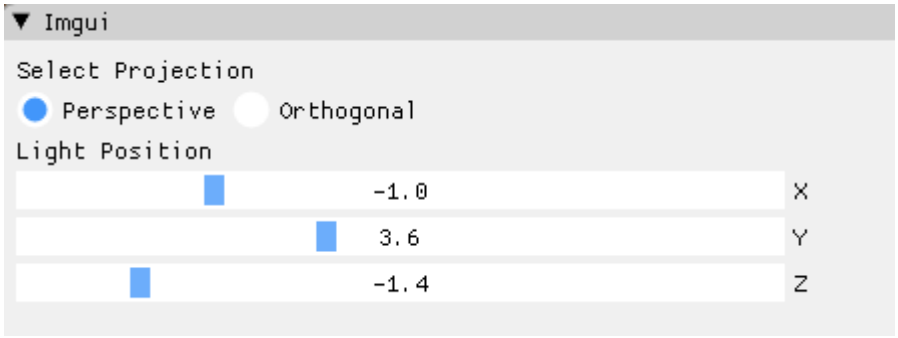
```
simpleDepthShader.Use();
glUniformMatrix4fv(lightSpaceMatrixLocation, 1, GL_FALSE, glm::value_ptr(lightSpaceMatrix));
glViewport(0, 0, SHADOW_WIDTH, SHADOW_HEIGHT);
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
glClear(GL_DEPTH_BUFFER_BIT);
RenderScene(simpleDepthShader);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

7、最后进行阴影渲染

代码见 cube.fs

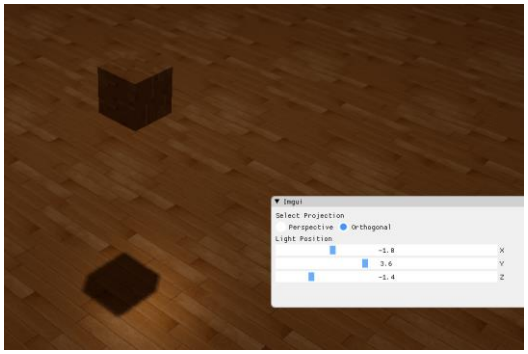
在 cube 的着色器中 ShadowCalculation()函数及 main()函数中进行阴影计算，然后进行渲染。

2. 修改 GUI

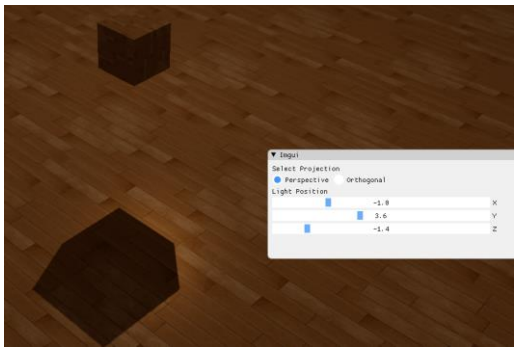


Bonus:1. 实现光源在正交/透视两种投影下的 Shadowing Mapping

正交



透视



Bonus:2. 优化 Shadowing Mapping

1、解决阴影失真和悬浮问题：

```
Float bia = max(0.05*(1.0-dot(normal, lightDir)), 0.005);
glCullFace(GL_FRONT);
RenderSceneToDepthMap();
glCullFace(GL_BACK);
```

2、解决采样过多：

```
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);
```

3、利用 pcf 来抗锯齿

```
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;
```