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。

2. 在报告里结合代码，解释 Shadowing Mapping 算法
3. 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 (可结合 References 链接，或其他方法。优化方式越多越好，在报告里说明，有加分)

阴影介绍

阴影是光线被阻挡的结果；当一个光源的光线由于其他物体的阻挡不能够达到一个物体的表面的时候，那么这个物体就在阴影中了

阴影映射(Shadow Mapping)背后的思路非常简单：我们以光的位置为视角进行渲染，我们能看到的东西都将被点亮，看不见的一定是在阴影之中了。

为了实现阴影映射，我们需要测试光线方向上物体的点的远近，

我们通过 z-buffer 进行这一过程。在深度缓冲中的一个值是摄像机视角下，对应于一个片元的一个 0 到 1 之间的深度值。z-buffer 中深度值显示从光源的透视图下见到第一个片元。

渲染场景 function : renderSence()

渲染 cube：

和之前一样，将顶点数组与着色器绑定。这里根据 VAO 的 ID 判读那是否生成了顶点数组对象，如果没有则根据顶点数组生成对应的 VAO 和 VBO，并且解析指针；如果已经初始化完毕，则直接绑定 cubeVAO，并根据 36 个顶点绘制 cube 表面的 12 个三角形。

渲染 plane：

平面同样先生成 planeVAO 和 planeVBO 并绑定缓存，解析指针，渲染时绑定，并绘制表示平面的两个三角形。

场景渲染根据上面初始化的 plane 和 cube 通过改变 model 变换矩阵生成不同位置的 plane 和 cube.

深度贴图

深度贴图是从光的透视图里渲染的深度纹理，用它计算阴影。因为我们需要将场景的渲染结果储存到一个纹理中，我们将再次需要帧缓冲。

创建帧缓冲并进行 2D 纹理载入

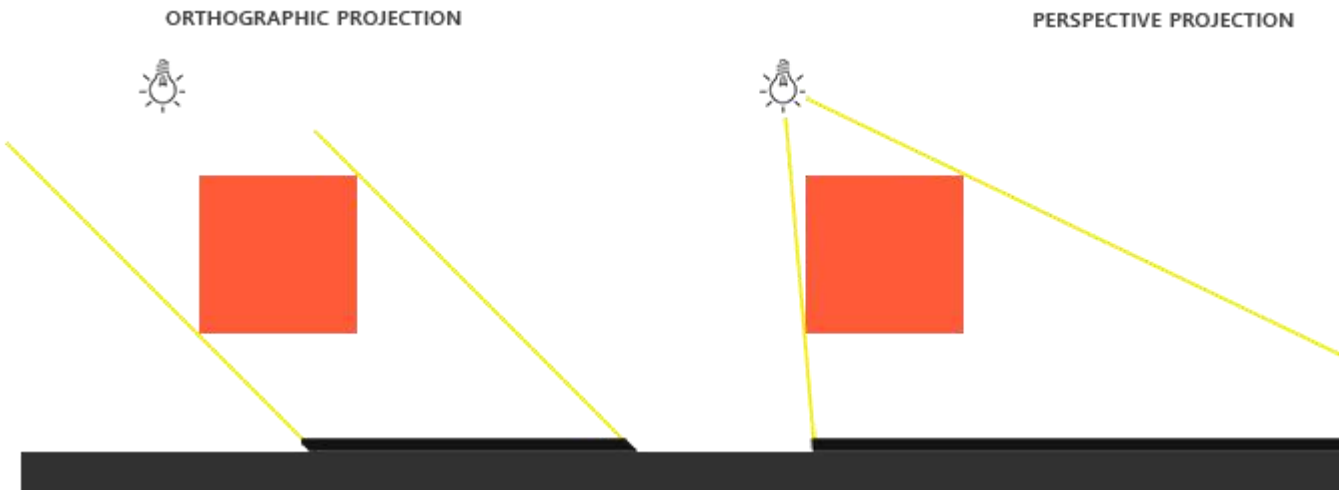
```
GLuint depthMapFBO;
glGenFramebuffers(1, &depthMapFBO);

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);
```

根据在光的透视图下的场景判断深度

因为我们只需要使用深度信息，而不需要颜色，所以这两项为 none

glDrawBuffer(GL_NONE);　　glReadBuffer(GL_NONE);

渲染深度贴图



首先，选择透视投影还是正交投影。然后将物体*lightProjection

变换得到其在光透视坐标下的相对坐标

```
lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, near_plane, far_plane);
```

将渲染好的深度贴图贴到物体的表面上

当我们以光的透视图进行场景渲染的时候，我们会用一个比较简单的着色器，这个着色器除了把顶点变换到光空间以外，不会做得更多了。这个简单的着色器叫做 simpleDepthShader，就是使用下面的这个着色器：

```
#version 330 core

layout (location = 0) in vec3 position;



uniform mat4 lightSpaceMatrix;

uniform mat4 model;

void main(){

    gl_Position = lightSpaceMatrix * model * vec4(position, 1.0f);

}
```

这个顶点着色器将一个单独模型的一个顶点，使用 lightSpaceMatrix 变换到光空间中。由于我们没有颜色缓冲，最后的片元不需要任何处理，所以我们可以简单地使用一个空像素着色器：这个空像素着色器什么也不干，运行完后，深度缓冲会被更新。

```
#version 330 core

void main(){

    // gl_FragDepth = gl_FragCoord.z;

}
```

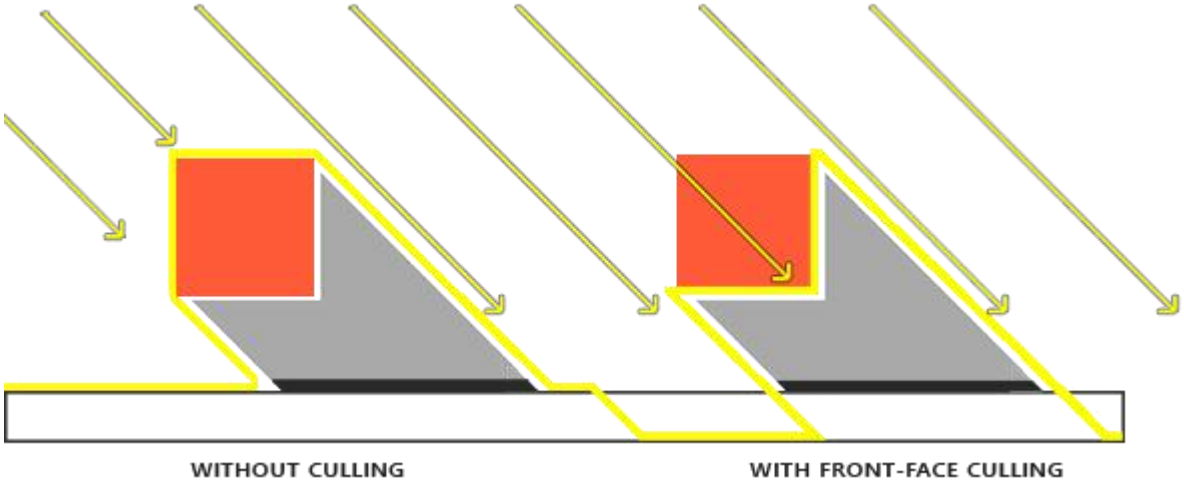
shadowing mapping 算法

shadow_shader.fs 中，首先要在顶点着色器中进行变换，将顶点着色器转换为世界坐标系的结果和顶点在光空间中的坐标位置。

根据上次介绍的光照模型，并添加 shadow 判断变量表示该像素是在阴影内还是阴影外，如果是阴影部分，就只渲染环境光；如果不在阴影部分，就加上漫反射和镜面反射的渲染。

```
float ShadowCalculation(vec4 fragPosLightSpace)
{
    // 执行透视除法
    vec3 projCoords = fragPosLightSpace.xyz / fragPosLightSpace.w;
    // 变换到[0,1]的范围
    projCoords = projCoords * 0.5 + 0.5;
    // 取得最近点的深度(使用[0,1]范围下的 fragPosLight 当坐标)
    float closestDepth = texture(shadowMap, projCoords.xy).r;
    // 取得当前片元在光源视角下的深度
    float currentDepth = projCoords.z;
    // 检查当前片元是否在阴影中
    float shadow = currentDepth > closestDepth  ? 1.0 : 0.0;

    return shadow;
}
```

避免悬浮



使用阴影偏移可能会导致悬浮问题，我们需要剔除正面来解决，因为只需要深度贴图的深度值，即使是背面深度出现错误，因为无法看到，所以不会被发现。插入以下代码 便可以解决悬浮问题

glCullFace(GL_FRONT);

glCullFace(GL_BACK);