Homework 7 – shadowing mapping

注：本次作业的完整演示请见/doc/pic/*.gif

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Shadowing Mapping算法

2. **修改GUI**
   只保留了之前的rotating

   光源投影方式依然选择Phong shading

   首先建立深度贴图，需要用它计算阴影，因为我们需要将场景的渲染结果储存到一个纹理中，我们将需要一个帧缓冲对象并创建2D纹理供深度缓冲使用。

```
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, 1024, 1024, 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);
GLfloat borderColor[] = { 1.0, 1.0, 1.0, 1.0 };
glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, borderColor);
glBindFramebuffer(GL_FRAMEBUFFER, depthMap);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D,
depthMap, 0);
glDrawBuffer(GL_NONE);
glReadBuffer(GL_NONE);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

   后面的渲染分为两步：

   首先以光源为视角渲染整个场景，计算每个片段的深度，此处考虑正交投影下，即只考虑无限远处光源，光线简化为平行光，不考虑透视，物体大小不随camera远近变化。此处使用正交投影，并计算世界坐标变换到光源坐标的矩阵。

```
glm::mat4 lightSpaceMatrix = glm::mat4(1.0f);
glm::mat4 lightView = glm::mat4(1.0f);
glm::mat4 lightProjection = glm::mat4(1.0f);
float near_plane = 1.0f, far_plane = 7.5f;
lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, near_plane, far_plane);
lightView = glm::lookAt(lightPos, glm::vec3(0.0f), glm::vec3(0.0, 1.0, 0.0));
lightSpaceMatrix = lightProjection * lightView;
depth_shader.use();
depth_shader.setMat4("lightSpaceMatrix", lightSpaceMatrix);
```

然后以camera视角渲染场景，比较当前深度值（到光源距离）与深度贴图中储存的深度值判断某个点是否在阴影下，如果当前点比深度缓冲中同一点离光源更远，说明场景中有物体比当前采样点离光源更近，当前点处于阴影中。

使用的shader如下：
对于深度着色器，只需要将深度值渲染到深度贴图中，做一个空间变换即可。片段着色器由于会在渲染时自动被计算深度缓冲，不需要进行任何操作。

```
#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);
}
```

在第二次渲染中使用修改后的Phong Shading，顶点着色器也需要进行一个空间变换。

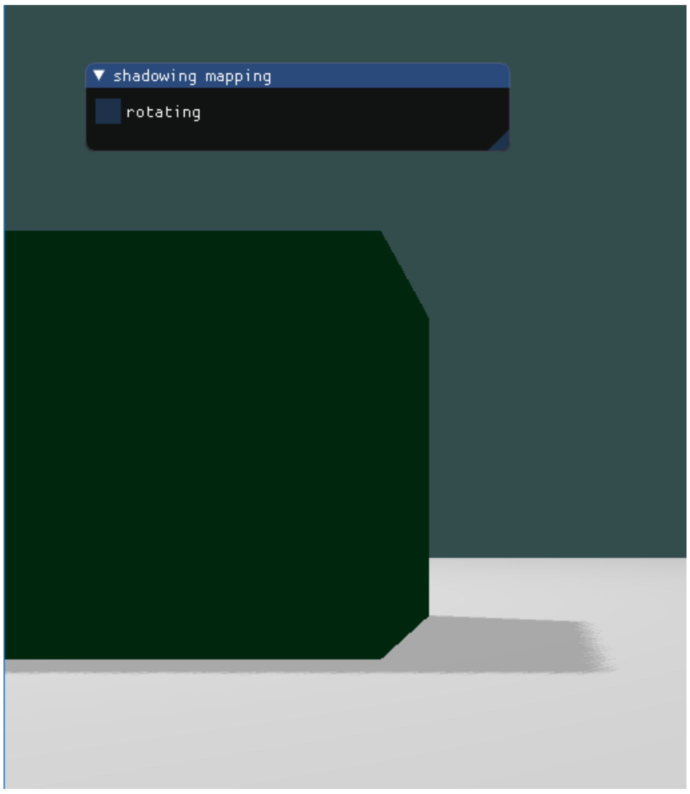
```
#version 330 core
layout(location = 0) in vec3 aPos;
layout(location = 1) in vec3 aNormal;

out vec3 Normal;
out vec3 FragPos;
out vec4 FragPosLightSpace;

uniform mat4 lightSpaceMatrix;
uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

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

片段着色器中需要计算阴影的值。即当前点与深度最近点比较，判断阴影值为0或1

```
#version 330 core
out vec4 FragColor;

in vec3 Normal;
in vec3 FragPos;
in vec4 FragPosLightSpace;

uniform sampler2D shadowMap;

uniform vec3 lightPos;
uniform vec3 lightColor;
uniform vec3 objectColor;
uniform vec3 viewPos;


float ShadowCalculation(vec4 fragPosLightSpace) {
    vec3 projCoords = fragPosLightSpace.xyz / fragPosLightSpace.w;
    projCoords = projCoords * 0.5 + 0.5;
    float closestDepth = texture(shadowMap, projCoords.xy).r;
    float currentDepth = projCoords.z;
     vec3 lightDir = normalize(lightPos - FragPos);
     float bias = max(0.05 * (1.0 - dot(normalize(Normal), lightDir)), 0.02);
    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;
    if(projCoords.z > 1.0)
        shadow = 0.0;
    return shadow;
}
```

```
void main() {
     vec3 ambient = 1.0 * lightColor;
     vec3 norm = normalize(Normal);
     vec3 lightDir = normalize(lightPos - FragPos);
     float diff = max(dot(norm, lightDir), 0.0);
     vec3 diffuse = diff * lightColor;
     vec3 viewDir = normalize(viewPos - FragPos);
    vec3 reflectDir = reflect(-lightDir, norm);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);
    vec3 specular = 0.5 * spec * lightColor;
     float shadow = ShadowCalculation(FragPosLightSpace);
    vec3 result = (ambient + (1.0 - shadow) * (diffuse + specular)) * objectColor;
     FragColor = vec4(result, 1.0);
}
```

效果如下图，GUI 的完整演示请见/doc/pic/basic.gif



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。)