Homework 7

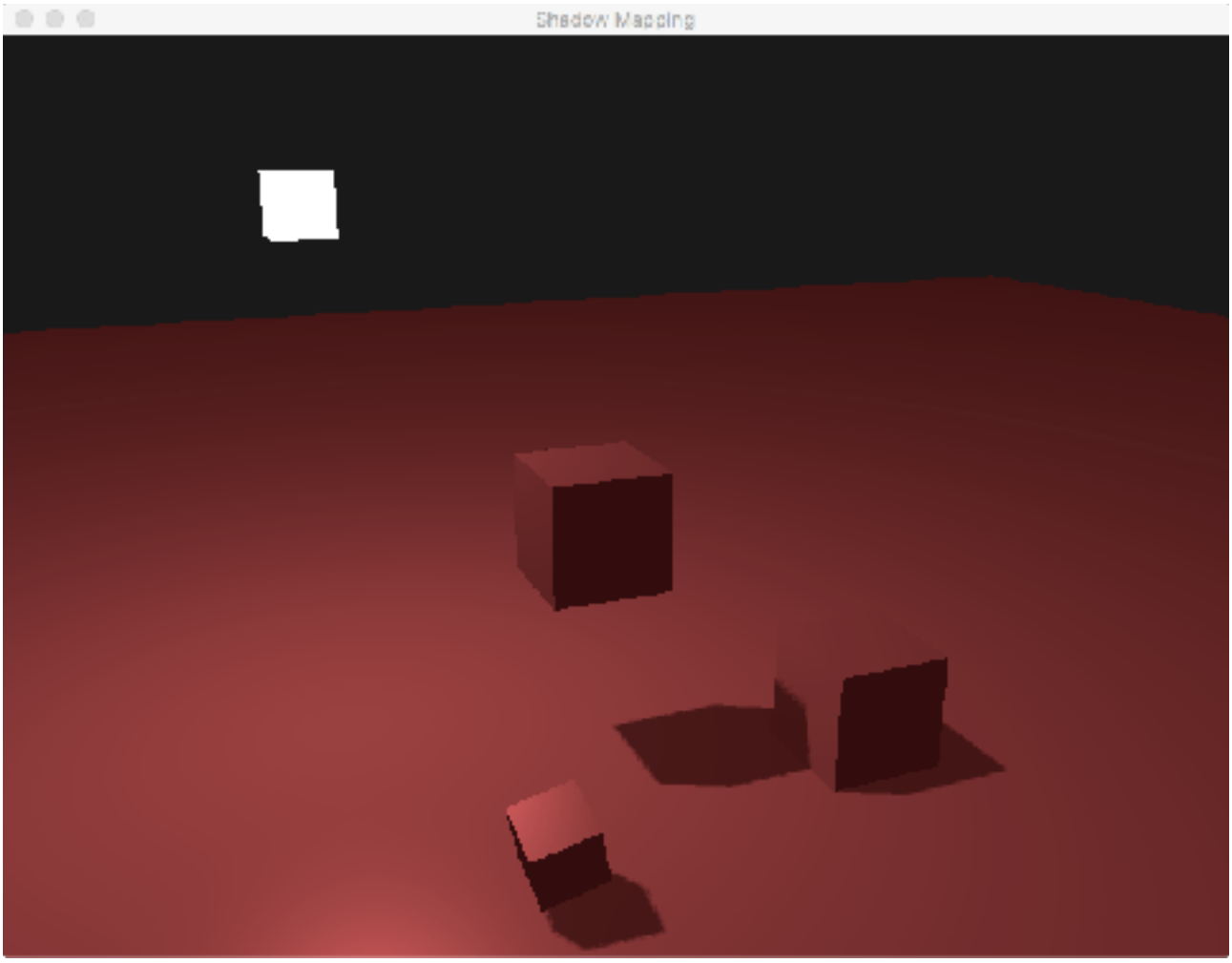
姓名：张家侨　　学号：15331390

Basic：

---

1. 实现Shadowing Mapping：

实现效果：



实现原理：

对于阴影效果的实现，一个关键要点就是如何知道一个点是否在阴影中，换个角度来想，就是一个点在光源看来能否被看到。如果看到，那就不在阴影中，如果看不到，那就在阴影中。这就是阴影映射的基本思想。那么，如何判断一个点是否被光源看到呢？这就可以通过深度贴图的方法来实现。之前我们使用过深度缓冲，判断点是否离摄像机比较近，然后渲染在前。通过深度贴图纹理，我们将光源视角下点的深度信息保存下来，然后在渲染光照效果的时候，通过判断当前点的深度是否是离光源最近的，是就不在阴影中，不是就在阴影中，相应减弱其光照效果即可。这就是阴影渲染的阴影映射方法的原理。

```
// 创建深度贴图，用于获取光源的视角深度
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_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);
```

所以首先我们需要生成一个深度贴图，用于获取深度信息。
然后，在光源空间下，通过一个空的着色器，只有点的位置信息，渲染获得深度信息。

```
// 从光源视角渲染场景，获取深度图
simpleDepthShader.use();
simpleDepthShader.setMat4("lightSpaceMatrix", lightSpaceMatrix);
glViewport(0, 0, SHADOW_WIDTH, SHADOW_HEIGHT);
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
    glClear(GL_DEPTH_BUFFER_BIT);
    RenderScene(simpleDepthShader);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

然后通过改良的光照渲染着色器，判断点的深度，渲染出阴影。

```
// 从光源视角渲染场景，获取深度图
simpleDepthShader.use();
simpleDepthShader.setMat4("lightSpaceMatrix", lightSpaceMatrix);
glViewport(0, 0, SHADOW_WIDTH, SHADOW_HEIGHT);
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
    glClear(GL_DEPTH_BUFFER_BIT);
    RenderScene(simpleDepthShader);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

这里的光照模型和教程的一样，采用的是Blinn-Phong光照模型，下面介绍着色器程序中如何实现渲染阴影。光照的部分和原模型是一样的，就是最后的效果呈现上要考虑是否是阴影。我们设阴影程度分级为0-1，0表示没有阴影，1表示完全在阴影中，那么通过对镜面与散射效果的阴影效果加权，就能实现阴影效果。公式如下：

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

下一步，就是确定阴影的权重了，通过判断点在光空间下的深度与之前生成的深度贴图的深度，我们就能实现阴影判断。

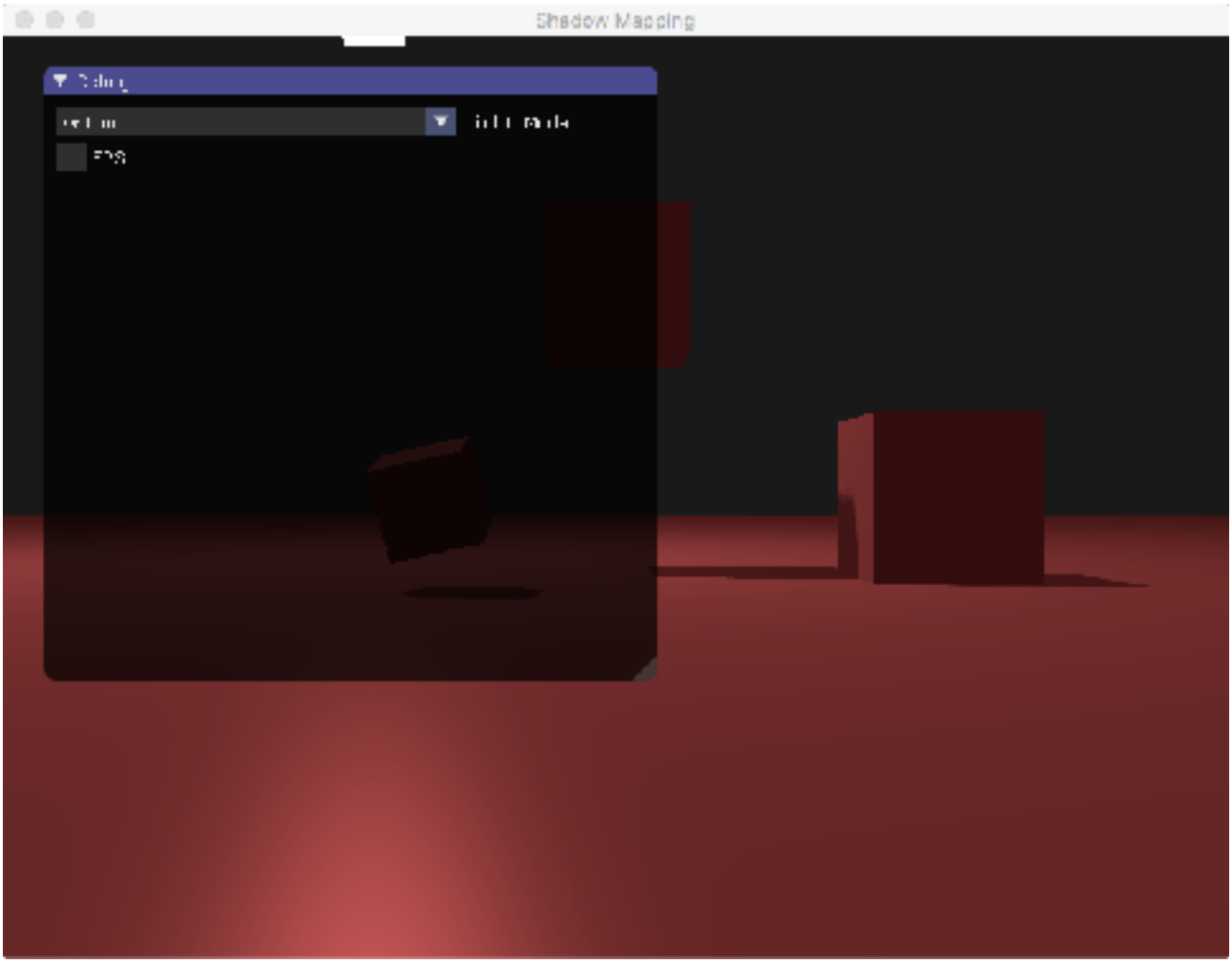
```glsl
float ShadowCalculation(vec4 fragPosLightSpace)
{
    // 透视除法
    vec3 projCoords = fragPosLightSpace.xyz / fragPosLightSpace.w;
    // 变换到0-1
    projCoords = projCoords * 0.5 + 0.5;
    // 获得当前位置离光源最近的深度
    float closestDepth = texture(depthMap, projCoords.xy).r;
    // 获得目前的深度
    float currentDepth = projCoords.z;
    // 计算偏离
    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);
    // 原始判断，不是1就是0
    // float shadow = currentDepth - bias > closestDepth  ? 1.0 : 0.0;
```

上面传入的点坐标是定点着色器计算得到的点相对于光空间的坐标。

---

2. 修改GUI：

实现效果：

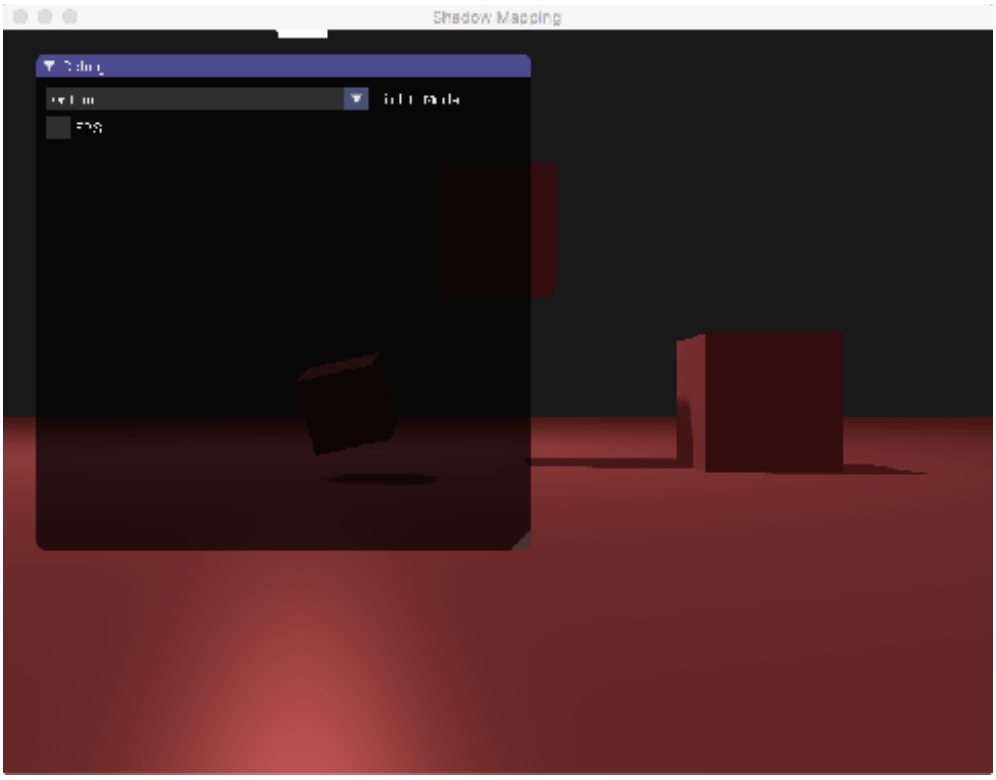


Bonus：

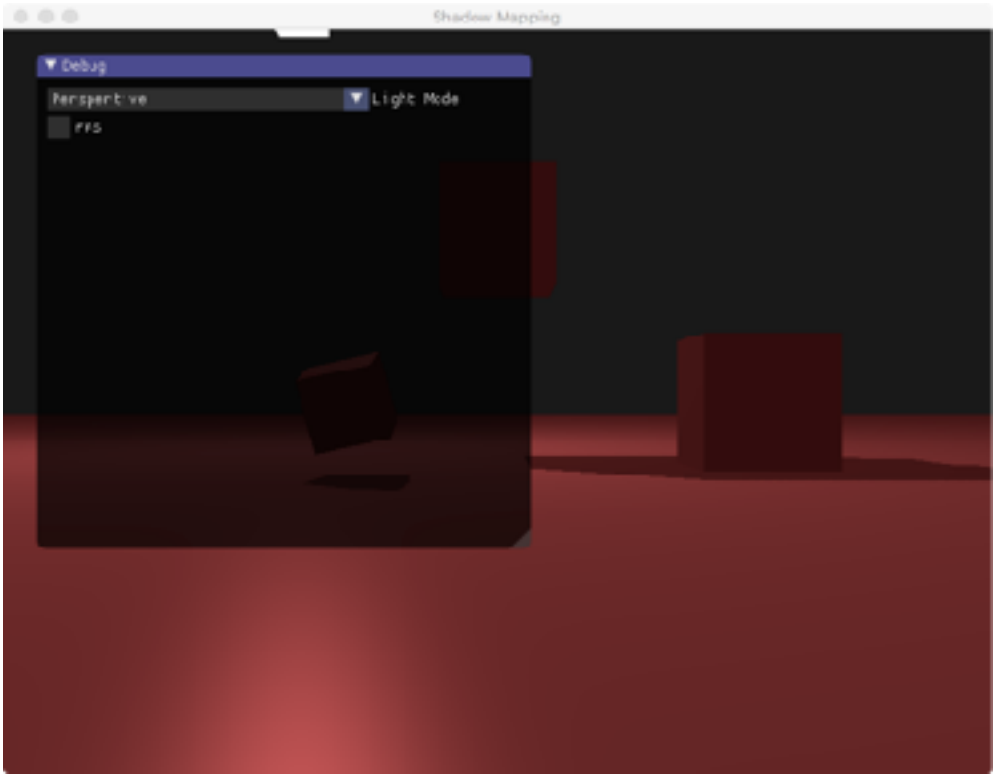
---

1. 实现光源在正交和透视两种投影下的Shadowing Mapping

实现效果：
正交：



透视：



2. 优化Shadowing Mapping

这里我的优化方案是根据教程里的PCF方法进行优化。这是一个类似于图像处理中的平滑操作，通过用当前点与周围的点的深度值做平均，就能消除原来的1到0的跳跃变换，阴影过渡变得稍微平滑一点。

```
// 原始判断，不是1就是0
// float shadow = currentDepth - bias > closestDepth  ? 1.0 : 0.0;
// PCF优化
float shadow = 0.0;
vec2 texelSize = 1.0 / textureSize(depthMap, 0);
for(int x = -1; x <= 1; ++x)
{
    for(int y = -1; y <= 1; ++y)
    {
        float pcfDepth = texture(depthMap, projCoords.xy + vec2(x, y) * texelSize).r;
        shadow += currentDepth - bias > pcfDepth  ? 1.0 : 0.0;
    }
}
shadow /= 9.0;
```