

# 计算机图形学 Homework 7 - Shadowing Mapping

陈铭涛  
16340024

May 10, 2019

## 1 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

Shadow Mapping 的主要步骤为：

- (a) 从光源出发进行渲染，无法渲染到的位置即视为阴影中，在这一部中通过光源的位置构建 view 矩阵，与投影矩阵结合构成 lightSpaceMatrix 矩阵。代码如下：

---

```
auto lightProjection = mode == Ortho ? ortho.mat() :  
    perspective.mat();  
auto lightView = glm::lookAt(source, center, glm::vec3(0, 1.0, 0));  
auto lightSpaceMat = lightProjection * lightView;
```

---

其中 source 为光源位置，center 为场景中央。在顶点着色器中与 model 矩阵结合将模型顶点变换到光空间的深度：

---

```
uniform mat4 lightSpaceMatrix;  
uniform mat4 model;  
void main()  
{  
    gl_Position = lightSpaceMatrix * model * vec4(position, 1.0f);  
}
```

---

Shadow Mapping 的主要内容位于 ShadowMapping 类中，其中使用了一个帧缓冲对象并将一个深度纹理作为深度缓冲：

---

```
FBO.bind();
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT,
    GL_TEXTURE_2D, depthMap, 0);
glDrawBuffer(GL_NONE);
glReadBuffer(GL_NONE);
FBO.unbind();
```

---

每次渲染深度贴图的流程为：切换视图参数，从光源渲染场景，还原视图状态：

```
GLint prevViewPort[4];
glGetIntegerv(GL_VIEWPORT, prevViewPort);

getDepthMapShader().use();
getDepthMapShader().setMat4("lightSpaceMatrix", lightSpaceMat);

// Render depth map
glViewport(0, 0, width, height);
FBO.bind();
glClear(GL_DEPTH_BUFFER_BIT);
renderFunc();
FBO.unbind();

glViewport(prevViewPort[0], prevViewPort[1], prevViewPort[2],
    prevViewPort[3]);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

---

(b) 使用生成的深度贴图渲染阴影，在顶点着色器中计算顶点的光空间位置：

```
FragPos = vec3(model * vec4(pos, 1.0));
FragPosLightSpace = lightSpaceMatrix * model * vec4(FragPos, 1.0);
```

---

在片段着色器中通过光空间位置与深度贴图计算阴影值，若一点在当前光源视角的深度高于深度贴图下的深度，则该点在阴影下：

```
float calculateShadow(vec4 fragPosLightSpace) {
    // Needed for perspective
    vec3 projCoords = fragPosLightSpace.xyz / fragPosLightSpace.w;
    projCoords = projCoords * 0.5 + 0.5;
    // Depth from shadowMap
    float closeDepth = texture(shadowMap, projCoords.xy).r;

    // Depth of current fragment from light's perspective
    float currentDepth = projCoords.z;
    vec3 normal = normalize(normal);
    vec3 lightDir = normalize(lightPos - FragPos);
```

```

float thetaCos = dot(normal, lightDir);

float shadow = currentDepth > closeDepth ? 1.0 : 0.0;

// 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;
}

```

---

由输出的阴影值修改漫反射与镜面反射的效果以显示出阴影效果：

```

float shadow = calculateShadow(FragPosLightSpace);
vec3 result = (ambient + (1 - shadow) * (diffuse + specular));
if (useTexture) {
    result *= texture(objectTex, TexCoord).rgb;
} else {
    result *= objectColor;
}
FragColor = vec4(result, 1.0);

```

---

场景中包含一个平面，一个表示光源的 Cube 和两个用于显示阴影的 Cube，橙色的 Cube 放置于平面表面，绿色的 Cube 悬浮于平面并旋转。

获得的效果如下：

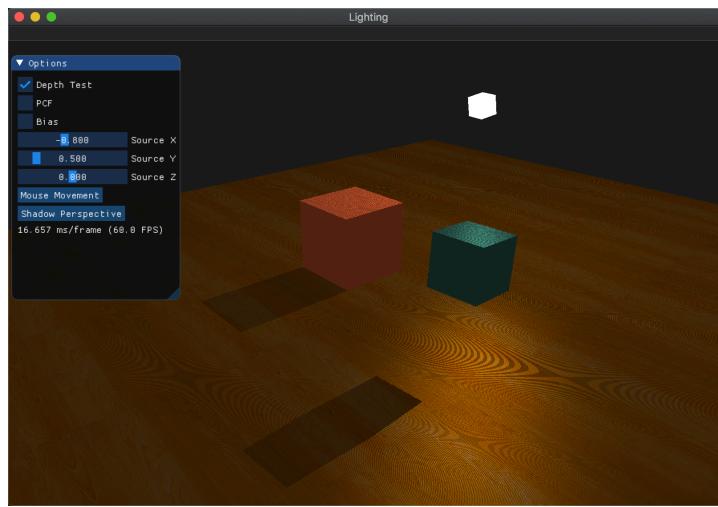


Figure 1: 阴影效果，白色 Cube 为光源，使用的投影为正交投影

## 2. 修改 GUI

修改 GUI 后可提供的功能有：

- (a) 开启或关闭深度测试
- (b) 启动或关闭 PCF 消除锯齿
- (c) 启动或关闭 bias 以消除阴影失真
- (d) 调整光源位置
- (e) 进入鼠标模式以通过鼠标调整摄像机方向
- (f) 光源改为使用透视投影

用户也可通过键盘 wasd 键输入控制摄像机位置。

GUI 的效果如下：

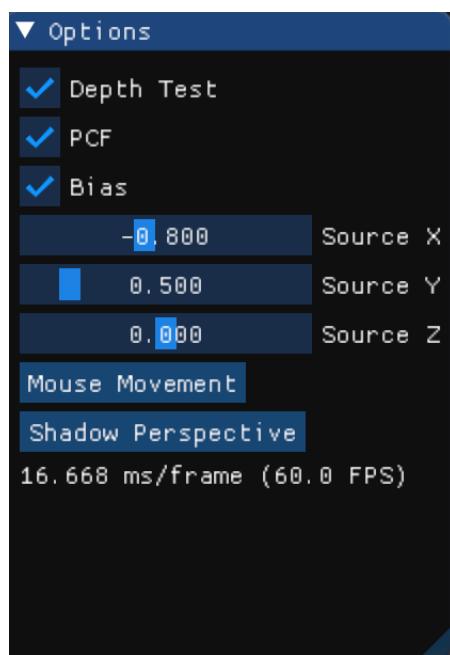


Figure 2: GUI

## 2 Bonus

### 1. 优化 Shadowing Mapping

- (a) 阴影失真

上图可见实现的阴影效果会导致场景各处出现明显的线条状，即阴影失真，其出现原因如下图所示：

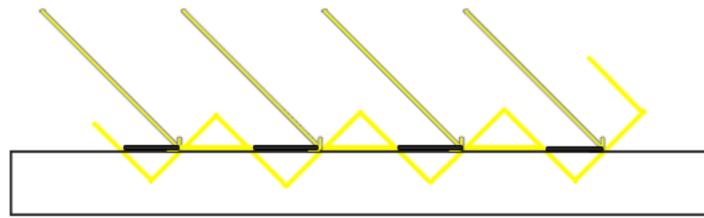


Figure 3: 阴影失真

当光以一个角度射向表面时，多个片元会在一个斜坡的深度中采样，使得一部分像素的深度在表面之上，一部分在表面之下，从而导致条纹阴影的出现。

一种解决方法是在表面深度上添加一个 Bias，如下图所示：

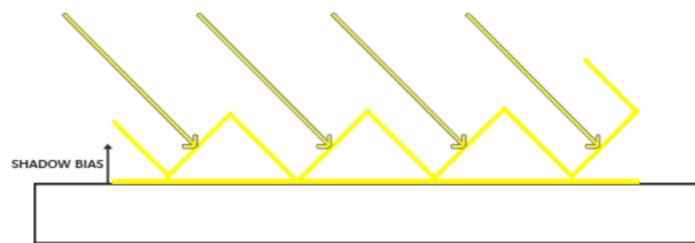


Figure 4: 阴影偏移

其在片段着色器中的实现如下：

---

```
float thetaCos = dot(normal, lightDir);
float bias = 0;
if (useBias) {
    bias = max(0.05 * (1.0 - thetaCos), 0.005);
}
shadow = currentDepth - bias > closeDepth ? 1.0 : 0.0;
```

---

应用后效果如下，可见条纹状现象已去除：

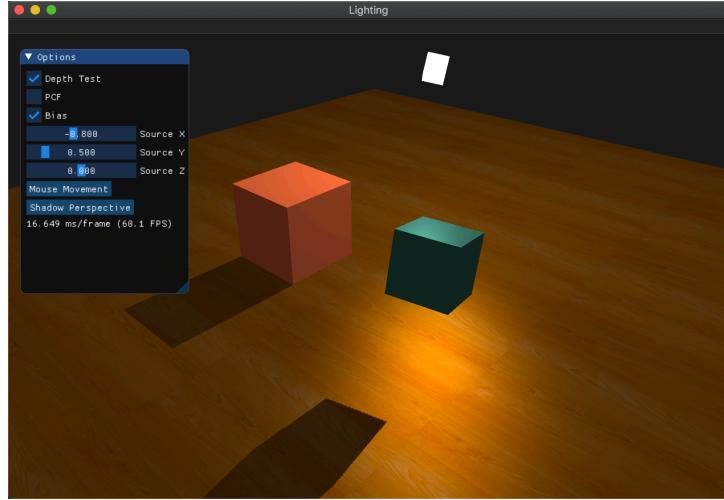


Figure 5: 应用 Bias 后的阴影效果

(b) PCF 将摄像机移近阴影处，观察到阴影存在着许多锯齿：

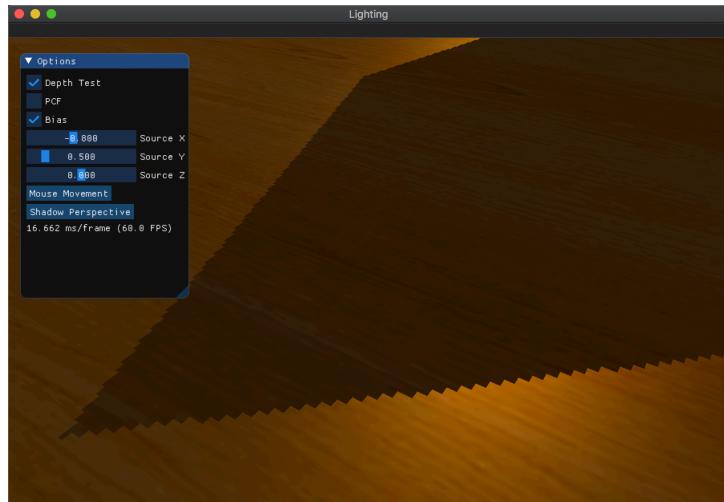


Figure 6: 阴影锯齿

随着深度贴图解析度上升，锯齿的现象可以减少，在此之外的一种减少锯齿现象的方式是 PCF，其基本思路是对每一个点都对其周围点采样并取平均值为该点的阴影值。在片段着色器中的实现为：

---

```

if (pcf) {
    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 /= 9.0f;
}
```

---

应用后效果如下，可见较之未开启时，阴影锯齿更为柔和：

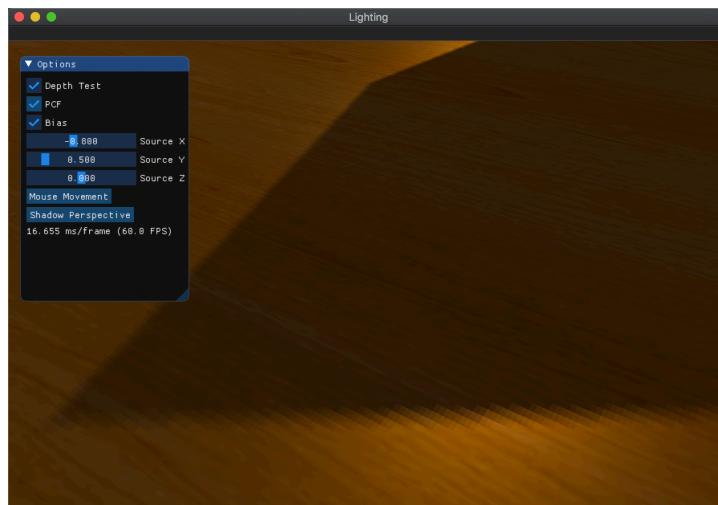


Figure 7: 开启 PCF 后效果