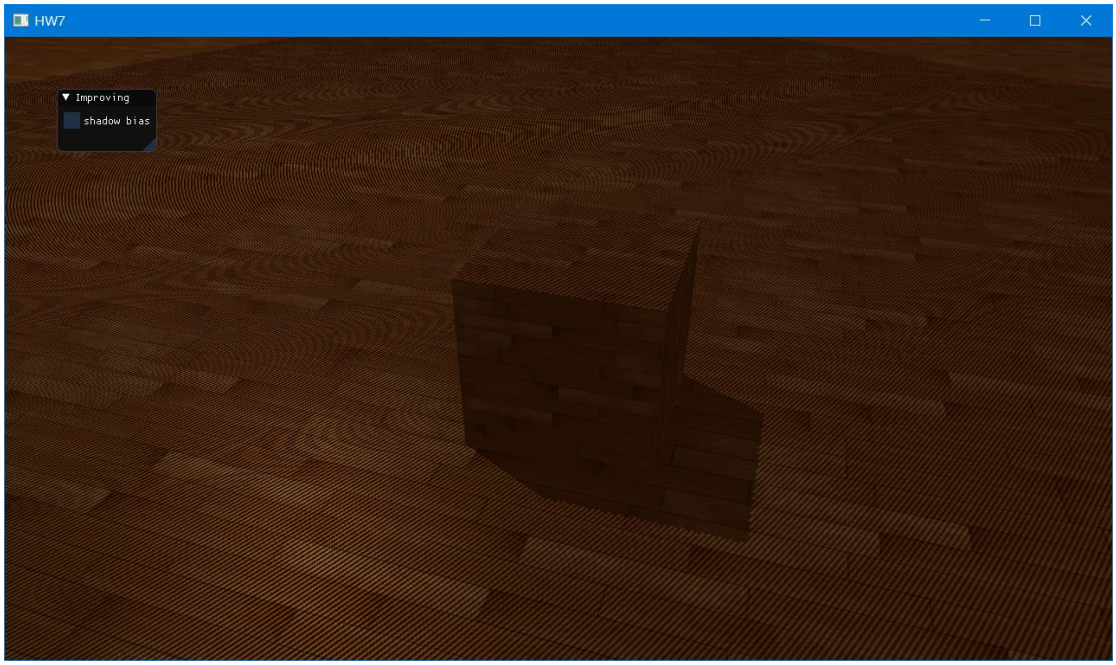
Report

Basic：



Shadow Mapping 的主要思路就是点亮光线经过的片段中距离光源最近的片段。但由于对光线穿过的所有片段进行遍历的计算量很大，因此需要借助深度缓冲的方法。

类似深度测试，我们需要变换世界空间使物体的坐标变为以光源为视角，并通过类似比较 z 分量获得深度的方法。由于此处光线是平行光，类似我们的正射投影：

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

并设置摄影机的属性（位置，目标和上向量）：

```
lightView = glm::lookAt(lightPos, glm::vec3(0.0f), glm::vec3(0.0, 1.0, 0.0));
```

于是得到所谓的光空间变换矩阵：

```
lightSpaceMatrix = lightProjection * lightView;
```

利用光空间，我们可以得到关于光源的深度缓冲。我们如果希望计算片元是否在阴影中，我们可以利用这个深度缓冲计算其是否在阴影中。

我们在着色器将顶点变换到光空间：

```
gl_Position = lightSpaceMatrix * model * vec4(aPos, 1.0);
```

然后我们创建一个帧缓冲对象，创建一个 2D 纹理，提供给帧缓冲的深度缓冲使用，再设置把生成的深度纹理作为帧缓冲的深度缓冲，就可以生成我们的深度贴图了。

在我们渲染阴影的时候，我们在顶点着色器中得到顶点在光空间的坐标：

```
vs_out.FragPosLightSpace = lightSpaceMatrix * vec4(vs_out.FragPos, 1.0);
```

然后在片段着色器中对比需要渲染的片段与在光空间下相同 xy 坐标的深度贴图的深度：

```glsl
// 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;
```

得到该片段是否在阴影中并渲染。

Bonus：

```glsl
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);

float shadow = currentDepth - bias > closestDepth  ? 1.0 : 0.0;
```

通过协调坡度得到的偏移 bias，用 currentDepth 减去 bias 得到更小的深度值，使当 currentDepth 实际上小于 closestDepth 但却因为 closestDepth 由于解析度不足，导致由于不同片段具有相同深度而造成 currentDepth 错误地大于 closestDepth 的错误结果得到修正，以消去不应该存在的阴影。