可视计算与交互概论 Lab3 报告

王蕙钰 2100018733

Task1: Phong Illumination

1. 顶点着色器和片段着色器的关系是什么样的？顶点着色器中的输出变量是如何传递到片段着色器当中的？

顶点着色器负责对每个顶点进行操作，包括对位置的变换、法线的变换、计算光照等。输出的主要是每个顶点的位置，这个位置用于将顶点映射到屏幕上，以及一些附加的数据（例如颜色、法线、纹理坐标等），这些数据会传递给后续的片段着色器使用。片段着色器接收从顶点着色器传递而来的数据，并对每个生成的片段进行处理，输出该片段的最终颜色以及其他可能的附加信息，如深度值。顶点着色器中的输出数据需要传递到片段着色器中使用，这个过程通常是通过插值完成的。每个顶点着色器的输出数据会在三角形面上进行插值，片段着色器能够访问这些插值后的数据。

顶点着色器声明一个输出，在片段着色器声明一个输入，当类型和名字都一样时 OpenGL 会将二者联系在一起，在实现链接对象时完成变量的传输。在本段代码中，in 实现了片段着色器接受顶点着色器的输出。

2. 代码中的 if (diffuseFactor.a < .2) discard; 这行语句，作用是什么？为什么不能用 if (diffuseFactor.a == 0.) discard; 代替？

这行语句的意思是如果漫反射小于一定阈值则不执行片段着色操作，片段也不会写入帧缓冲区。不能代替是因为 float 型不能用“=”来判断，筛选的片段的 diffuseFactor.a 应该是一个区间，等于 0 要求过于严格导致大部分本应该不绘制的地方不能通过判断。

实现思路：

根据定义分别计算漫反射和镜面反射，公式分别为：

diffusion = Il * kd * max(0, cos theta)

specular = Il * ks * (cos phi) ^ ns

Blinn-Phone 模型引入了半程向量 h，公式为：

$$L = k_d(I_a + I_d \max(0, \mathbf{n} \cdot \mathbf{l})) + k_s I_d \max(0, \mathbf{n} \cdot \mathbf{h})^p.$$

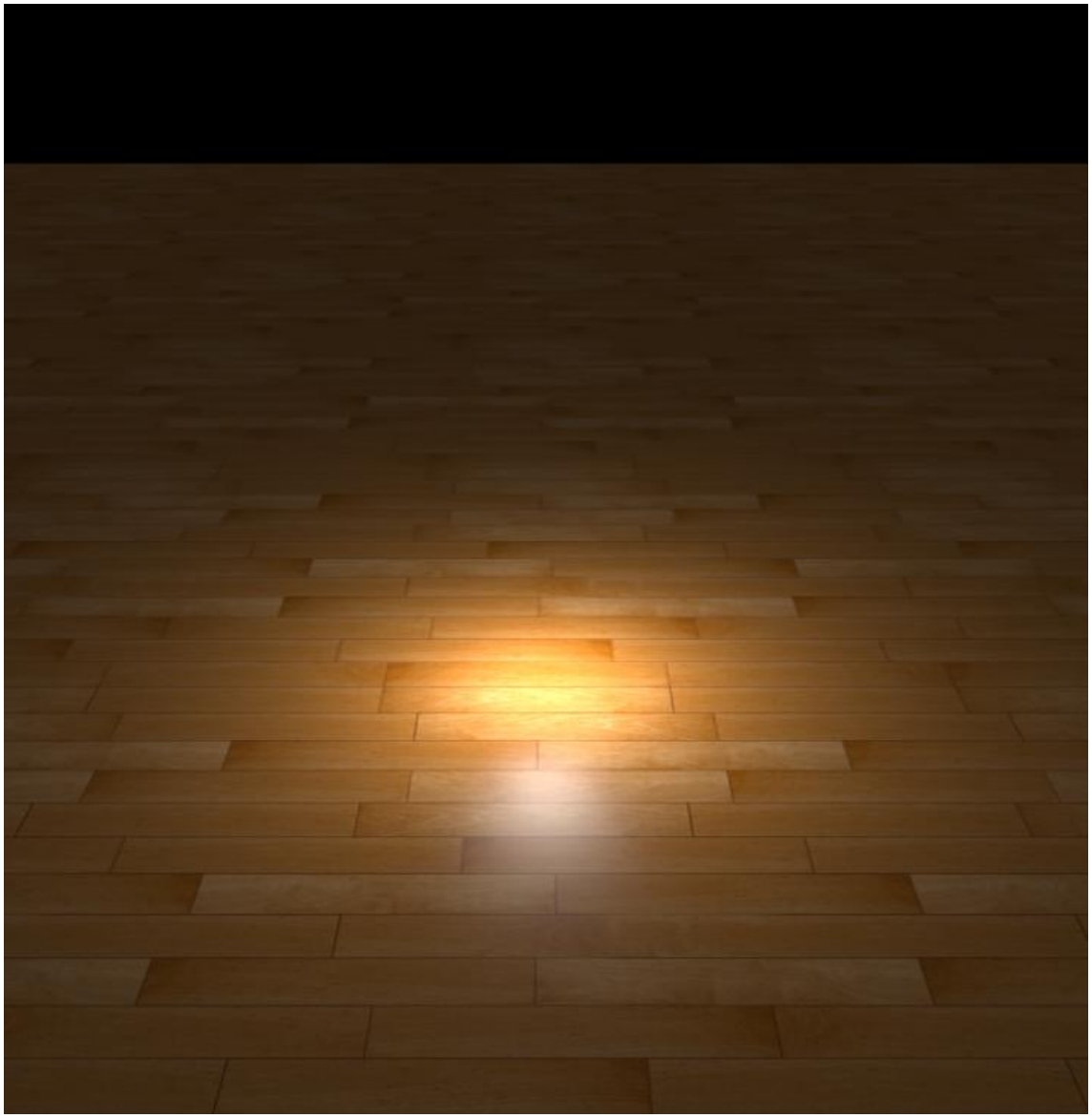
依据公式计算即可。

凹凸映射实现思路：通过添加凹凸贴图改变表面法向从而改变光线。先构建切线空间方向，再通过高度图估算表面梯度，再计算扰动后的法线。参考文档中的示例（https://gamedev.stackexchange.com/questions/174642/how-to-write-a-shader-that-only-uses-a-bump-map-without-a-normal-map）：

```
// Bump mapping
// from paper: Bump Mapping Unparametrized Surfaces on the GPU
vec3 vn = normalize( vnormal );
vec3 posDX = dFdx ( vworldpos.xyz );  // choose dFdx (#version 420) or dFdxFine
vec3 posDY = dFdy ( vworldpos.xyz );
vec3 r1 = cross ( posDY, vn );
vec3 r2 = cross ( vn , posDX );
float det = dot (posDX , r1);
float Hll = texture( bumptex, tc ).x;    //-- height from bump map texture, tc=t
float Hlr = texture( bumptex, tc + dFdx(vtexcoord.xy) ).x;
float Hul = texture( bumptex, tc + dFdy(vtexcoord.xy) ).x;
// float dBs = ddx_fine ( height );     //-- optional explicit height
// float dBt = ddy_fine ( height );

// gradient of surface texture. dBs=Hlr-Hll, dBt=Hul-Hll
vec3 surf_grad = sign(det) * ( (Hlr - Hll) * r1 + (Hul - Hll)* r2 );
float bump_amt = 0.7;       // bump_amt = adjustable bump amount
vec3 vbumpnorm = vn*(1.0-bump_amt) + bump_amt * normalize ( abs(det)*vn - surf_g
```

效果：



Task 2: Environment Mapping

实现思路：参考 OpenGL 的教程

skybox：完成坐标系的转换，并欺骗深度测试

所以，我们将会最后渲染天空盒，以获得轻微的性能提升。这样子的话，深度缓冲就会填充满所有物体的深度值了，我们只需要在提前深度测试通过的地方渲染天空盒的片段就可以了，很大程度上减少了片段着色器的调用。问题是，天空盒很可能会渲染在所有其他对象之上，因为它只是一个1x1x1的立方体（译注：意味着距离摄像机的距离也只有1），会通过大部分的深度测试。不用深度测试来进行渲染不是解决方案，因为天空盒将会复写场景中的其它物体。我们需要欺骗深度缓冲，让它认为天空盒有着最大的深度值1.0，只要它前面有一个物体，深度测试就会失败。

在坐标系统小节中我们说过，**透视除法**是在顶点着色器运行之后执行的，将`gl_Position`的`xyz`坐标除以w分量。我们又从深度测试小节中知道，相除结果的z分量等于顶点的深度值。使用这些信息，我们可以将输出位置的z分量等于它的w分量，让z分量永远等于1.0，这样子的话，当透视除法执行之后，z分量会变为`w / w = 1.0`。

```
void main()
{
    TexCoords = aPos;
    vec4 pos = projection * view * vec4(aPos, 1.0);
    gl_Position = pos.xyww;
}
```

最终的**标准化设备坐标**将永远会有一个等于1.0的z值：最大的深度值。结果就是天空盒只会在没有可见物体的地方渲染了（只有这样才能通过深度测试，其它所有的东西都在天空盒前面）。

envmap：计算反射向量 R，使用 R 来从天空盒中取样

我们根据观察方向向量`I`和物体的法向量`N`，来计算反射向量`R`。我们可以使用GLSL内建的`reflect`函数来计算这个反射向量。最终的`R`向量将会作为索引/采样立方体贴图的方向向量，返回环境的颜色值。最终的结果是物体看起来反射了天空盒。

因为我们已经在场景中配置好天空盒了，创建反射效果并不会很难。我们将会改变箱子的片段着色器，让箱子有反射性：

```
#version 330 core
out vec4 FragColor;

in vec3 Normal;
in vec3 Position;

uniform vec3 cameraPos;
uniform samplerCube skybox;

void main()
{
    vec3 I = normalize(Position - cameraPos);
    vec3 R = reflect(I, normalize(Normal));
    FragColor = vec4(texture(skybox, R).rgb, 1.0);
}
```

我们先计算了观察/摄像机方向向量`I`，并使用它来计算反射向量`R`，之后我们将使用`R`来从天空盒立方体贴图中采样。注

效果：

Task 3: Non-Photorealistic Rendering

1. 参考 Labs/3-Rendering/CaseNonPhoto.cpp 中的 OnRender 函数，代码是如何分别渲染模型的反面和正面的？

先用 glCullFace 判断正反面，禁用多边形正面或者背面上的光照、阴影和颜色计算及操作，消除不必要的渲染计算。

1. 反面：

```
glCullFace(GL_FRONT); //只绘制反面，即剔除正面
glEnable(GL_CULL_FACE); //开启面剔除功能，提高效率
for (auto const & model : _sceneObject.OpaqueModels) {
    auto const & material = _sceneObject.Materials[model.MaterialIndex];
    model.Mesh.Draw({ _backLineProgram.Use() }); //反面使用 _backLineProgram 渲染
}
```

2. 正面：

```
glCullFace(GL_BACK);//只绘制正面，即剔除反面
glEnable(GL_DEPTH_TEST); //开启深度测试
for (auto const & model : _sceneObject.OpaqueModels) {
    auto const & material = _sceneObject.Materials[model.MaterialIndex];
    model.Mesh.Draw({ material.Albedo.Use(), material.MetaSpec.Use(), _program.Use() });
}//正面使用 material.Albedo.Use(), material.MetaSpec.Use(), _program 渲染
```

2. npr-line.vert 中为什么不简单将每个顶点在世界坐标中沿着法向移动一些距离来实现轮廓线的渲染？这样会导致什么问题？

npr-line.vert 中实现轮廓线的函数：

```
void main() {
    vec4 clipPos = u_Projection * u_View * vec4(a_Position, 1.);
    vec3 clipNorm = mat3(u_Projection) * mat3(u_View) * a_Normal;
    vec2 offset = normalize(clipNorm.xy) / vec2(u_ScreenWidth, u_ScreenHeight) * u_LineWidth
* clipPos.w * 2;
    clipPos.xy += offset;
    gl_Position = clipPos;
}
```

中：

将顶点位置和法向转移到 Clip Space 后，1. 偏移量除以屏幕的宽度和高度，用于将屏幕空间的像素值标准化到裁剪空间坐标（-1 到 1 范围），防止偏移量在不同分辨率或宽高比下失真；2. 乘以顶点的 clipPos.w 值，确保了偏移量在不同深度（Z 轴方向）下表现出正确的透视缩放效果。

如果按照题干方法，则不能实现上述效果，产生问题如：由于透视，轮廓线的宽度在屏幕上不一致，远处的线看起来更细，近处的线更粗；由于世界坐标系法向与轮廓线偏移方向不一样一致导致轮廓线可能变形。

实现思路：

1. 根据着色公式：

$$I = ((1 + dot(l, n)) / 2) * k\_cool + (1 - ((1 + dot(l, n)) / 2)) * k\_warm$$

2. 为了实现分段效果，先将 dotvalue 离散化再使用着色公式。

效果：

Task 4: Shadow Mapping

1. 想要得到正确的深度，有向光源和点光源应该分别使用什么样的投影矩阵计算深度贴图?
有向光源使用正交投影，点光源使用透视投影。

2. 为什么 phong-shadow.vert 和 phong-shadow.frag 中没有计算像素深度，但是能够得到正确的深度值?
因为使用的物体坐标是相对于光源的坐标，在有向光的情况下深度即为坐标的 z 值，即 float curDepth = pos.z。

实现思路：使用阴影贴图计算光照时求 closestDepth 只需要从贴图中读取数据即可。
效果：

Task5:

1. 光线追踪和光栅化的渲染结果有何异同？如何理解这种结果？

光栅化渲染是将场景中的三角形投影到屏幕上的像素中，然后计算每个像素的颜色。只能实现部分光照效果，间接光照（经过多次反射折射）、软阴影等无法很好表达。由于计算速度快适合实时渲染。光线追踪模拟光线从摄像机发射，追踪光线与场景中物体的交点以计算颜色。由于是模拟真实的物理世界，可以表达更多的效果的效果，如间接光照、全局效果。缺点是计算量大，渲染速度慢。

实现思路：

总体思路：从屏幕上每一点发射一条光 ray，如果它不与任何物体相交则返回环境值；如果有相交则遍历每一个光源，考察 shadowray（连接交点和光源），判断该点是否能被该光照到。根据该点的透明度计算反射和折射，添加该点的贡献并从该点发射 secondary ray 重复操作。

1. 求光线与三角的交点参考以下论文

Denoting $E_1 = V_1 - V_0$, $E_2 = V_2 - V_0$ and $T = O - V_0$, the solution to equation (4) is obtained by using Cramer's rule:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{|-D,\ E_1,\ E_2|} \begin{bmatrix} |T,\ E_1,\ E_2| \\ |-D,\ T,\ E_2| \\ |-D,\ E_1,\ T| \end{bmatrix} \tag{5}$$

¿From linear algebra, we know that $|A,\ B,\ C| = -(A \times C) \cdot B = -(C \times B) \cdot A$. Equation (5) could therefore be rewritten as

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{(D \times E_2) \cdot E_1} \begin{bmatrix} (T \times E_1) \cdot E_2 \\ (D \times E_2) \cdot T \\ (T \times E_1) \cdot D \end{bmatrix} = \frac{1}{P \cdot E_1} \begin{bmatrix} Q \cdot E_2 \\ P \cdot T \\ Q \cdot D \end{bmatrix}, \tag{6}$$

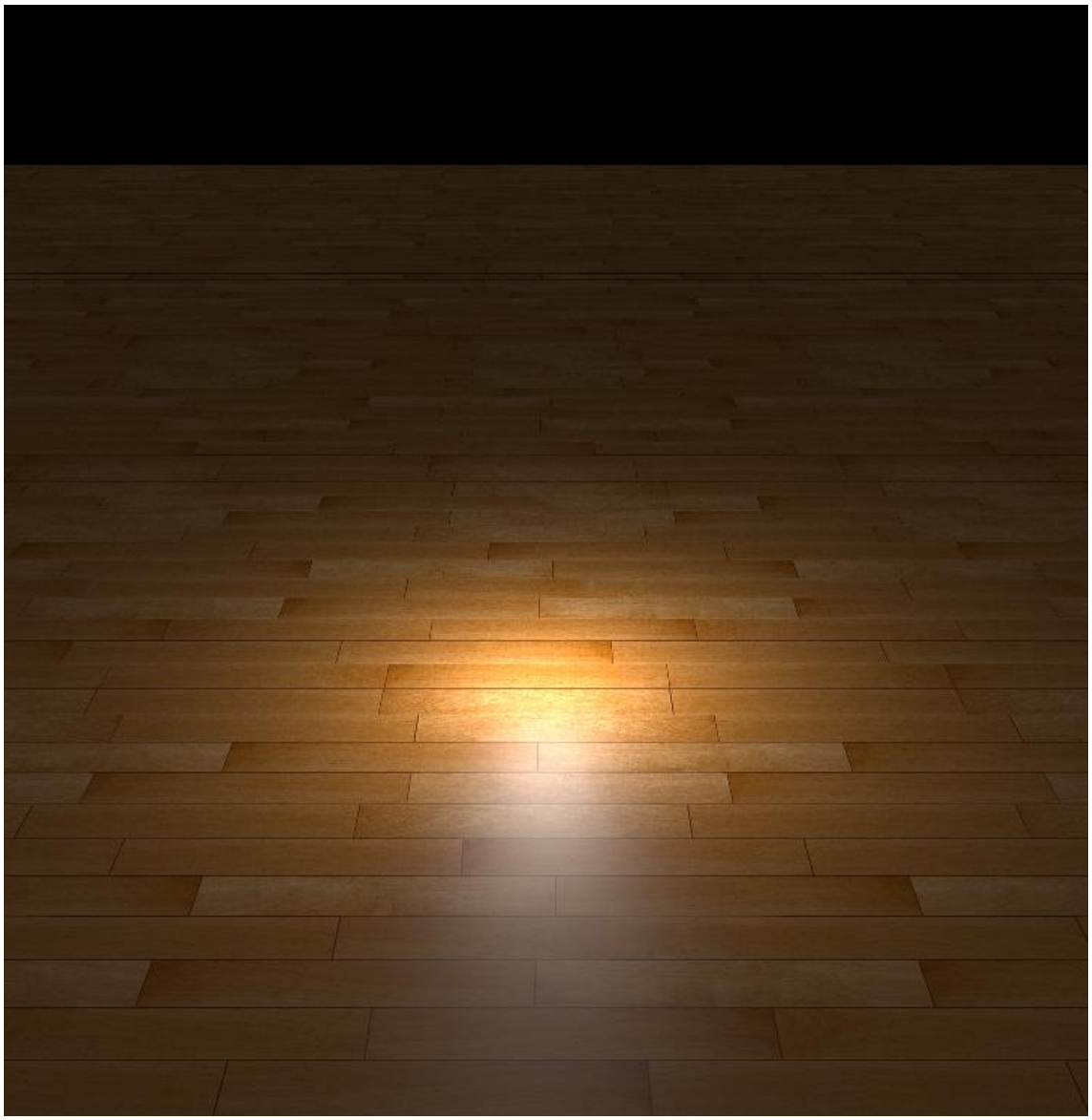
where $P = (D \times E_2)$ and $Q = T \times E_1$. In our implementation we reuse these factors to speed up the computations.

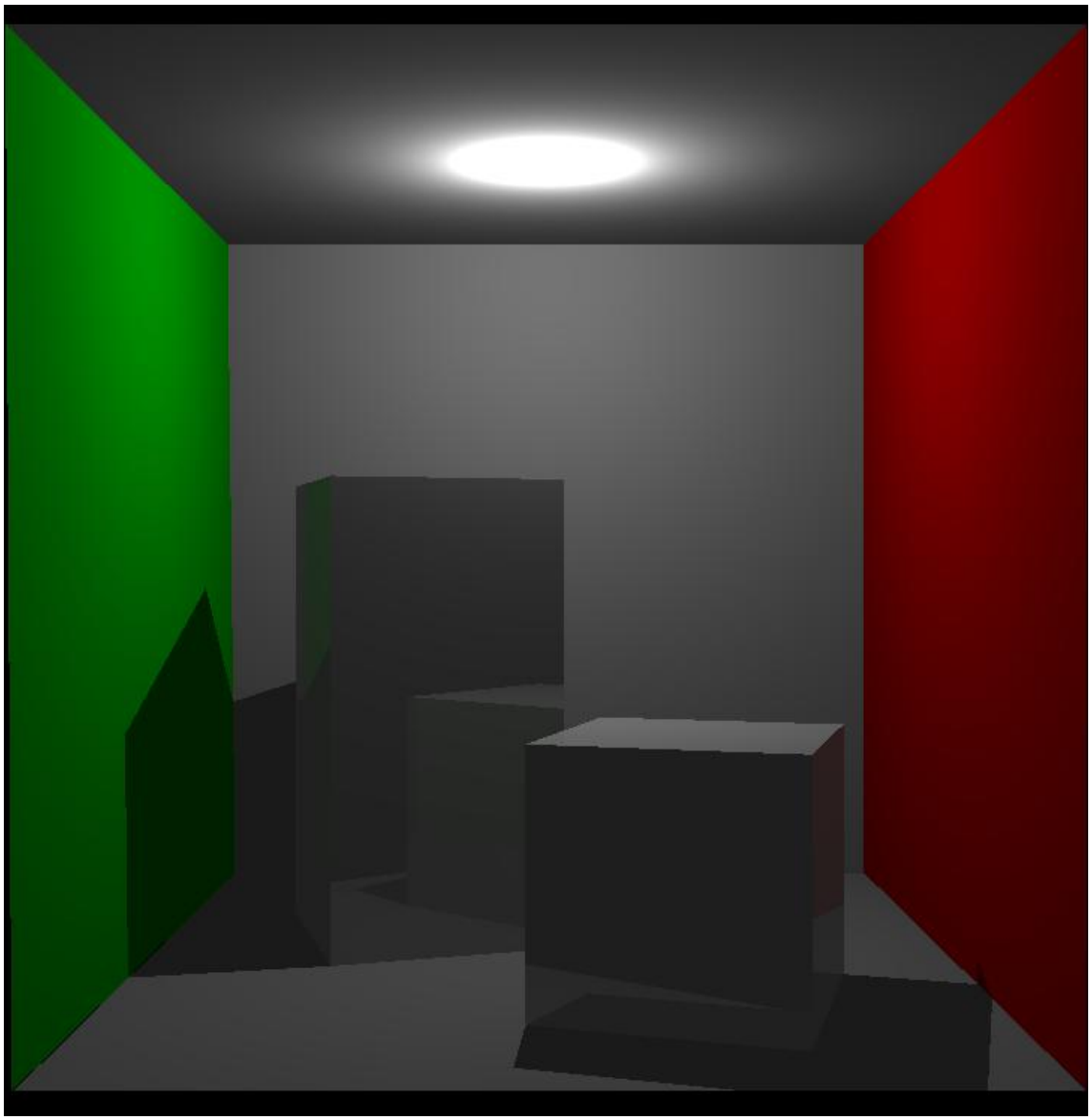
参考文献：https://dl.acm.org/doi/pdf/10.1145/1198555.1198746

2. 根据已经求出的 ks，kd 等信息，依据 Blinn-phone 模型着色

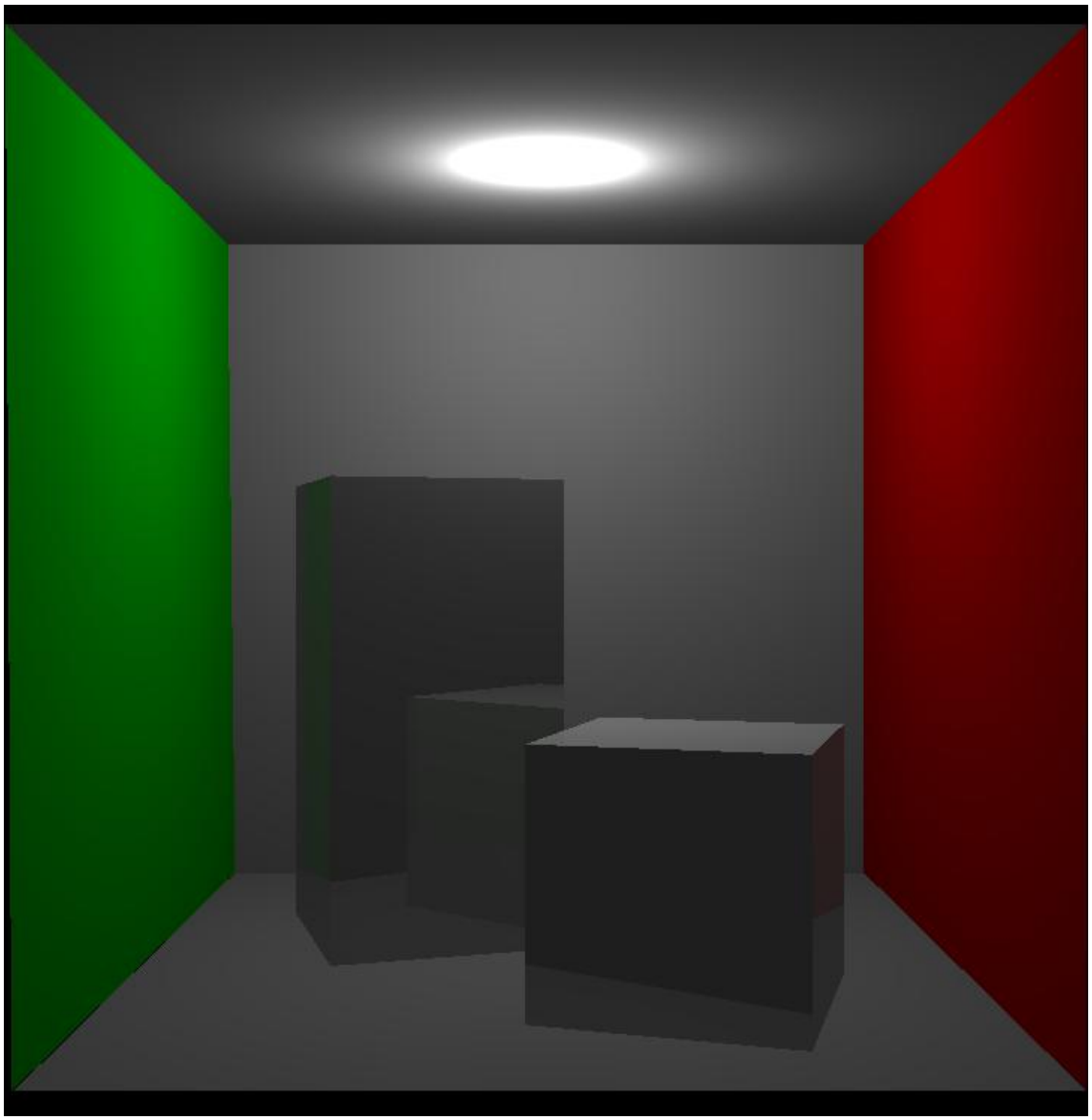
3. 如果 shadowray 在抵达光源前与物体相交，且物体是不透明的，则认为该点不能被该光源照到。

效果：





（有阴影，cornell box）



（无阴影）