

# Lab3 报告

叶茂 2200017852

## 1 Phong Illumination

### 1.1

首先将输入的各个向量归一化，方便后续通过点积求方向。使用到的公式为  $L = k_d(I_a + I_d \max(0, \mathbf{n} \cdot \mathbf{l})) + k_s I_d \max(0, \mathbf{n} \cdot \mathbf{h})^p$ ，实际代码中忽略了  $I_a$ ；若不使用Blinn-Phong模型，则通过`reflect(-1, n)`计算反射向量，再计算反射向量与法向量的点积；若使用 Blinn-Phong模型，则计算半程向量  $\mathbf{h}$  与法向量的点积即可。

### 1.2 Bonus

根据论文中提供的代码框架，首先求出顶点在屏幕上的  $x$  和  $y$  方向上的梯度 ( $dFdx$  和  $dFdy$ )，根据这两个梯度和法向量构建出切向量。随后同理计算顶点纹理的梯度，利用 `texture` 函数求出相应三个点的高度值，利用高度差和切向量计算法向量的偏移值。最后根据系数将原法向量和新法向量插值得到最终的法向量。

### 1.3 Questions

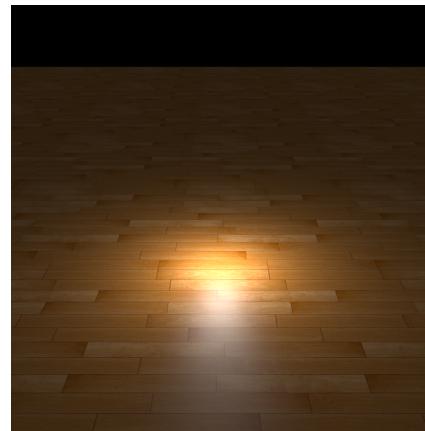
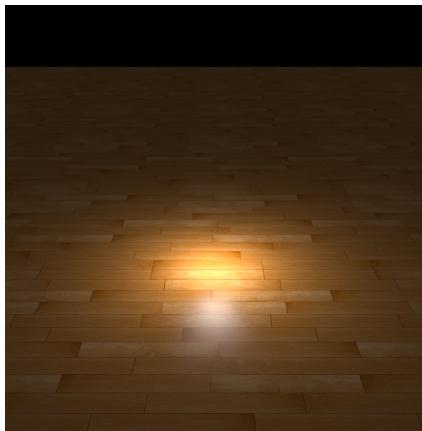
#### 1.3.1 Question 1

顶点着色器的输出一般作为片段着色器的输入。顶点着色器的输出变量会通过插值传递给片段着色器。

#### 1.3.2 Question 2

`if (diffuseFactor.a < .2) discard;` 这句语句中的  $a$  变量是颜色的透明度，当透明度小于 0.2 时，则不渲染这个片段，即处理透明化。若用`if(diffuseFactor.a == 0.) discard;`代替，首先不能保证透明的物体其颜色透明度分量一定为 0，条件过于严苛。其次，将小于 0.2 的全部当做透

明处理，具有一定的平滑效果，能同时处理透明和半透明的材质，而不会出现很突兀的变化。





## 2 Environment Mapping

首先将\_View矩阵转化为 $3 \times 3$ 矩阵再转回 $4 \times 4$ 矩阵，这样做可以消除矩阵中的位移成分，使得天空盒不随着视角变化而产生距离的变化。随后将顶点位置与映射矩阵相乘得到对应的天空盒纹理坐标，将得到的向量z分量设为和w分量一样的值，确保天空盒永远在其他物体之后。对于环境贴图，求出视线的出射向量，使用texture函数获取出射向量对应的环境贴图颜色即可。



### 3 Non-Photorealistic Rendering

计算出冷暖颜色的系数  $k_{cool} = \frac{1+l\cdot n}{2}$  以及  $k_{wram} = \frac{1-l\cdot n}{2}$ 。根据两个系数差值冷暖颜色即可实现非真实感渲染。为了得到颜色分界线的效果，当冷色系数较大时返回冷色，当暖色系数较大时返回暖色，否则返回冷暖色的平均值，这样就实现了颜色分界线，具体效果随相关阈值变化。

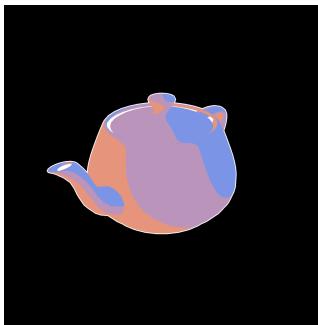
#### 3.1 Questions

##### 3.1.1 Question 1

在OnRender函数中, 使用`glCullFace(GL_FRONT)`; 和`|glCullFace(GL_BACK)`; 以及`glEnable`分别剔除模型正面和背面进行渲染，达到绘制轮廓线的效果。

##### 3.1.2 Question 2

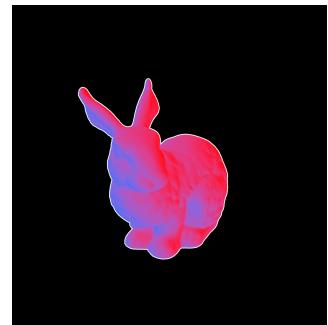
直接移动顶点坐标在不同视角下轮廓线的效果可能不好，线的宽度不好控制，且偏移的效果和法向的质量有关。



(a) 颜色分界



(b) 颜色分界



(c) 普通效果

## 4 Shadow Mapping

本题需要补充的代码较少，根据光的方向，使用 `texture` 函数获取深度贴图对应位置的深度值即可。在点光源中，由于获取的深度在  $[0, 1]$ ，故需要乘以 `u_FarPlane` 将其还原再与当前深度进行比较。

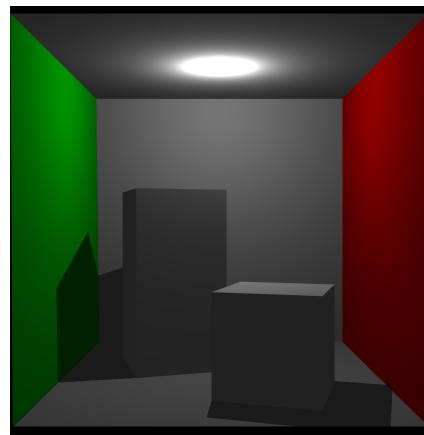
### 4.1 Questions

#### 4.1.1 Question 1

对于有向光源，使用变换矩阵将坐标映射到光空间之后，使用的是正交投影矩阵。对于有向光源，使用的是透视投影矩阵。

#### 4.1.2 Question 2

这两个着色器实现的是有向光源的阴影，因此将顶点坐标映射到光空间并消除齐次坐标的影响后， $z$  分量变换到  $[0, 1]$  就可以作为深度使用。



## 5 Whitted-Style Ray Tracing

光线求交部分，我按照讲义中的方法实现。

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{\mathbf{P} \cdot \mathbf{E}_1} \begin{bmatrix} \mathbf{Q} \cdot \mathbf{E}_2 \\ \mathbf{P} \cdot \mathbf{T} \\ \mathbf{Q} \cdot \mathbf{D} \end{bmatrix}$$

判断

$t, u, v, 1-u-v$  是否大于等于 0 即可。光线追踪部分，首先计算环境光。对于每条 shadow ray，若与光源之间无障碍物则之间计算该顶点的Blinn-Phong着色；若与光源之间有障碍，则需要判断所有障碍是否都可视为透明 ( $\text{alpha} < 0.2$ )，若障碍都为透明，则着色，否则不着色。在有障碍时，起初我从顶点发射光线，使用 while 循环遍历所有碰撞点并得到其 alpha 值，若遇到  $\text{alpha} >= 0.2$  则说明该顶点在 shadow 中，不着色。但 `intersector.IntersectRay(Ray(pos, dir))` 返回的是光线与最近面片交点的相关信息，并没有处理光源，所以即使顶点与光源之间没有障碍，仍会返回与面片相交的顶点，而不是不相交，导致顶点最终总是在 shadow 中。因此，我改而从光源出发，按照光线方向遍历所有障碍物交点，若交点材质可视为透明则穿透，否则循环终止。计算每个障碍物交点与顶点的距离，若小于一定的阈值且路径上障碍物均“透明”，则说明光源与顶点之间“无障碍”，进行着色。

### 5.1 Questions

#### 5.1.1 Question 1

光线追踪的渲染结果更有真实感，因为它模拟了真实场景中的光线反射、折射和全局光照等，也能够更好的渲染阴影。相比之下，光栅化仅仅是将面片映射到屏幕，缺少真实感。

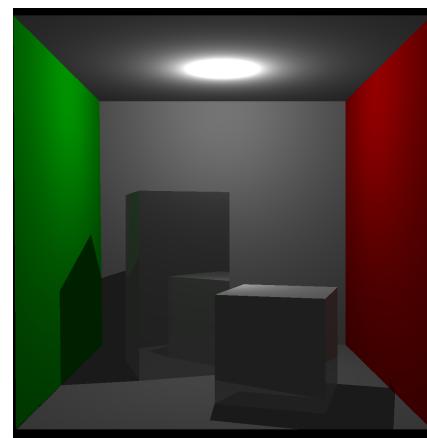
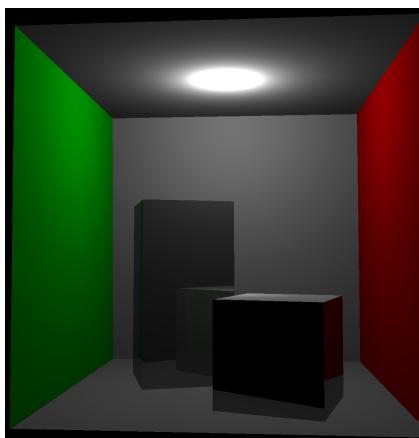




图 9: Breakfast Room,max depth=5, 渲染时间: $\approx$ 6h+RTX3060+ 无加速结构