可视计算与交互概论 lab3 实验报告

邱荻 2000012852

Task 1: Phong Illumination (3')

在完成这个任务的同时，请在报告中回答下列问题：

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？

   顶点着色器应该接收的是一种特殊形式的输入，它从顶点数据中直接接收输入。片段着色器，它需要一个 vec4 颜色输出变量，因为片段着色器需要生成一个最终输出的颜色。在发送方着色器中声明一个输出，在接收方着色器中声明一个类似的输入。当类型和名字都一样的时候，OpenGL 就会把两个变量链接到一起，它们之间就能发送数据了（这是在链接程序对象时完成的）。

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？

   如果漫反射太小了就退出片段着色器，不执行后面的片段着色操作。片段也不会写入帧缓冲区。因为 float 型不能用“=”来判断，如果用等于 0 代替，那么有很多比较暗的地方就没法实现不绘制。

思路：

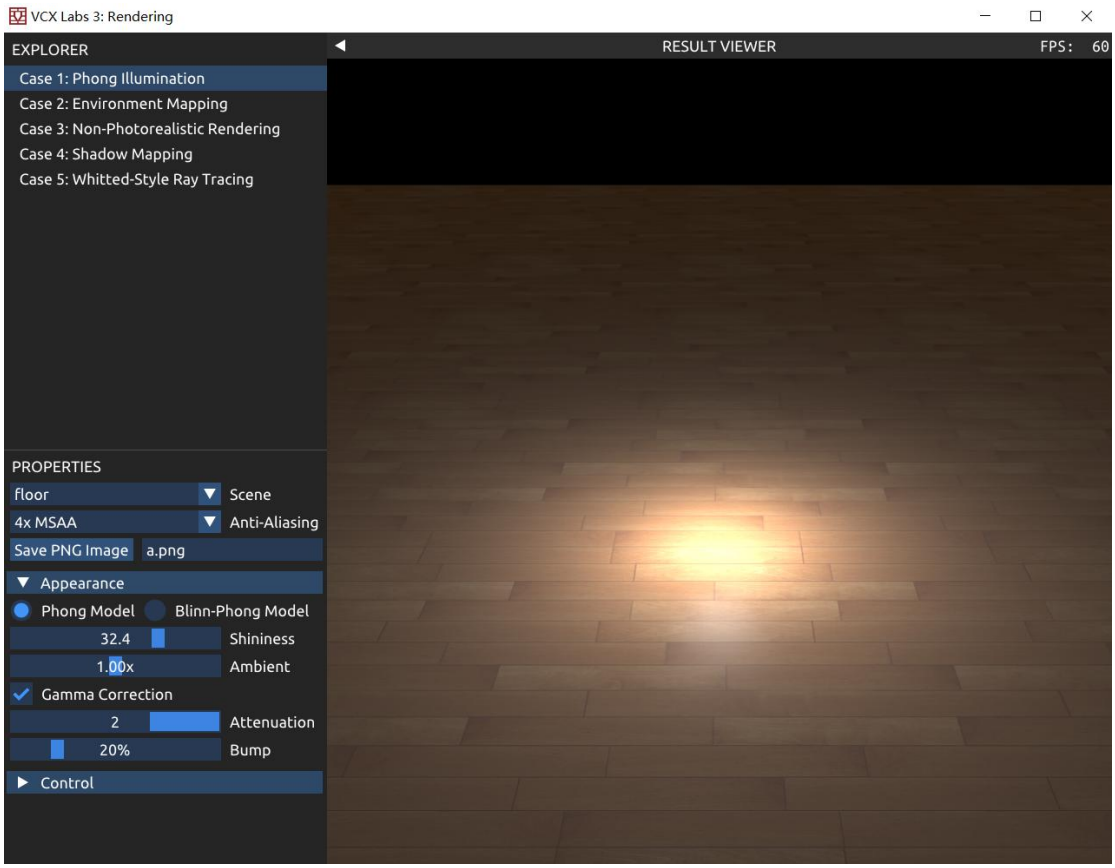
Phong 模型：

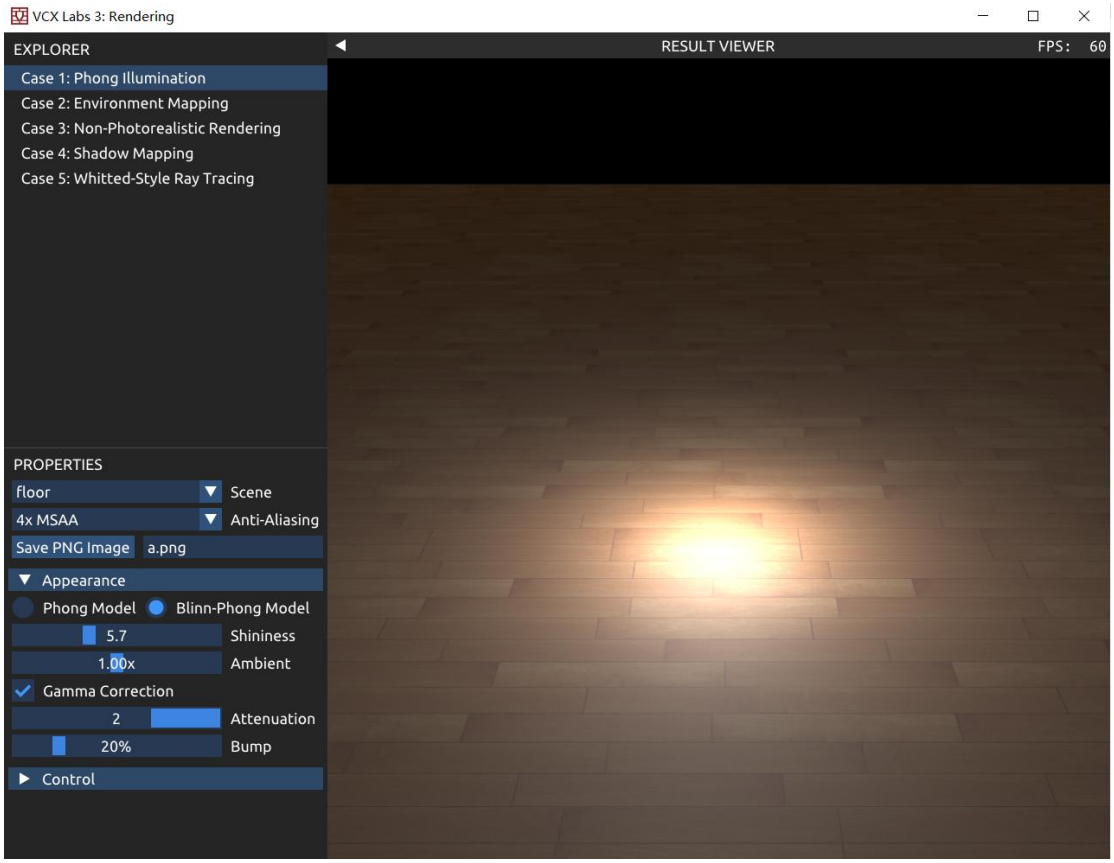
$$
\begin{aligned}
I &= I_a + I_d + I_s \\
&= k_a L_a + k_d \max(0, \vec{n} \cdot \vec{l}) L_d + k_s (\vec{r} \cdot \vec{v})^s L_s
\end{aligned}
$$

Blinn-Phong 模型：

$$
\begin{aligned}
I &= I_a + I_d + I_s \\
&= k_a L_a + k_d \max(0, \vec{n} \cdot \vec{l}) L_d + k_s (\vec{n} \cdot \vec{h})^s L_s
\end{aligned}
$$

运行结果：





利用 `u_HeightMap` 实现凹凸映射 **(bump mapping),**思路是直接参考所给代码

Here is a complete solution for GLSL:

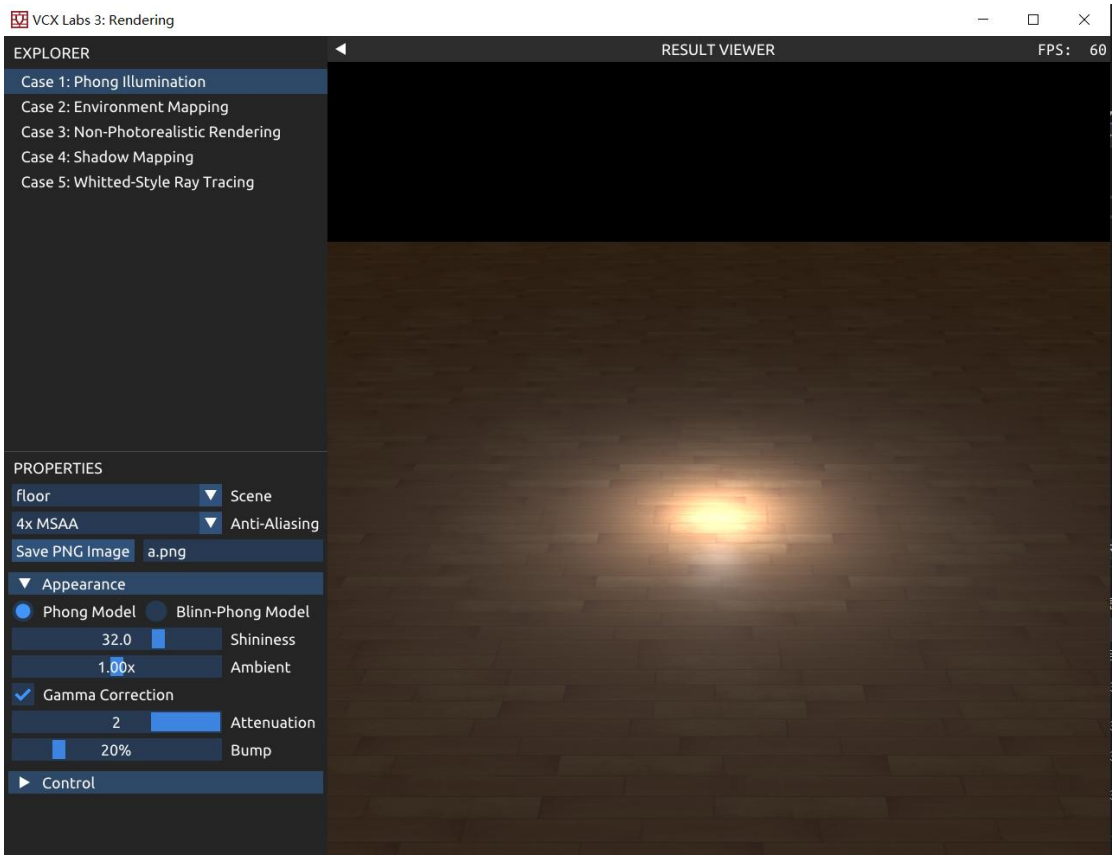
```
// Bump mapping
// from paper: Bump Mapping Unparametrized Surfaces on the GPU
vec3 vn = normalize( vnormal );
vec3 posDX = dFdx ( vworldpos.xyz );  // choose dFdx (#version 420) or dFdxFine (#ver
vec3 posDY = dFdy ( vworldpos.xyz );
vec3 r1 = cross ( posDY, vn );
vec3 r2 = cross ( vn , posDX );
float det = dot (posDX , r1);
float Hll = texture( bumptex, tc ).x;    //-- height from bump map texture, tc=textur
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_grad )
```

和

```
    vec2 dSTdx = dFdx( vUv );
    vec2 dSTdy = dFdy( vUv );
    float Hll = bumpScale * texture2D( bumpMap, vUv ).x;
    float dBx = bumpScale * texture2D( bumpMap, vUv + dSTdx ).x - Hll;
    float dBy = bumpScale * texture2D( bumpMap, vUv + dSTdy ).x - Hll;
    return vec2( dBx, dBy );
```

运行结果：



Task 2: Environment Mapping

思路：
参考这里

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;

out vec3 Normal;
out vec3 Position;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main()
{
    Normal = mat3(transpose(inverse(model))) * aNormal;
    Position = vec3(model * vec4(aPos, 1.0));
    gl_Position = projection * view * model * vec4(aPos, 1.0);
}
```

和这里

我们根据观察方向向量$\bar{I}$和物体的法向量$\bar{N}$，来计算反射向量$\bar{R}$。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$\bar{R}$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);
}
```

最终结果：

Task 3: Non-Photorealistic Rendering

在完成这个任务的同时，请在报告中回答下面的问题：

1. 代码中是如何分别渲染模型的反面和正面的？（答案在 `Labs/3-Rendering/CaseNonPhoto.cpp` 中的 `OnRender()` 函数中）

用 glCullFace 来指定正反面，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。禁用正面时用 backLineProgram 渲染，禁用反面时用

{ material.Albedo.Use(), material.MetaSpec.Use(), _program.Use() }参数渲染。

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？

   这样会导致整个图形都是白色的，就像是一个茶壶镀了一层白色的绣，而不是实现白色的轮廓。

思路：
着色公式直接代入：

I = ((1 + dot(l, n)) / 2) * k_cool + (1 - ((1 + dot(l, n)) / 2)) * k_warm

最终结果：

Task 4: Shadow Mapping

在阅读和补全代码的过程中，请在报告中回答下面的问题：

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：正交投影矩阵

   点光源：透视投影矩阵

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？

   因为在前面的步骤中已经得到深度贴图了。

思路：参考这里

完整的shadowCalculation函数是这样的：

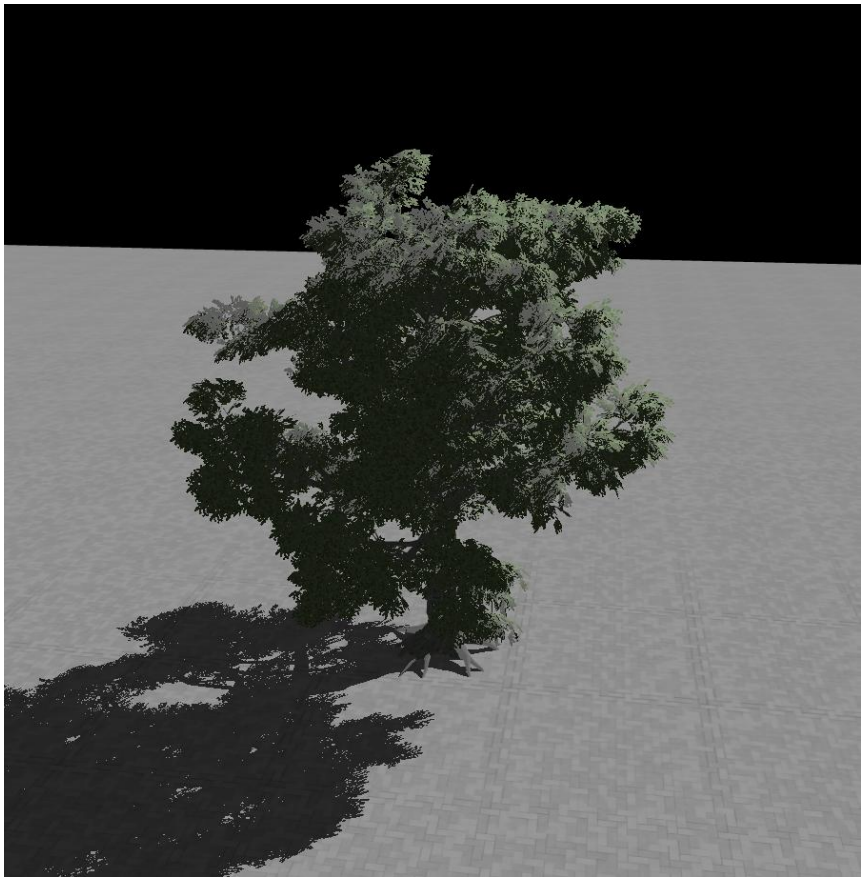
```
float ShadowCalculation(vec4 fragPosLightSpace)
{
    // 执行透视除法
    vec3 projCoords = fragPosLightSpace.xyz / fragPosLightSpace.w;
    // 变换到[0,1]的范围
    projCoords = projCoords * 0.5 + 0.5;
    // 取得最近点的深度(使用[0,1]范围下的fragPosLight当坐标)
    float closestDepth = texture(shadowMap, projCoords.xy).r;
    // 取得当前片段在光源视角下的深度
    float currentDepth = projCoords.z;
    // 检查当前片段是否在阴影中
    float shadow = currentDepth > closestDepth  ? 1.0 : 0.0;

    return shadow;
}
```

和这里

```
float ShadowCalculation(vec3 fragPos)
{
    vec3 fragToLight = fragPos - lightPos;
    float closestDepth = texture(depthMap, fragToLight).r;
}
```

最终结果：



Task 5: Whitted-Style Ray Tracing

在阅读和补全代码的过程中，请在报告中回答下面的问题：

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？

光栅化不能很好的解决一些问题，比如说一些全局的效果，软阴影，Glossy Reflection(类似于毛玻璃这样的反射)，间接光照(Indirect Illumination 光线在抵达摄像机/人眼之前反射了多次的情况)。以上效果用光栅化的办法效果不怎么好。光栅化很快，但效果不怎么好。光线追踪效果好，但是慢。因为光线追踪是模拟真实的光线和物体的碰撞，所以效果更好。

思路如下：

`IntersectTriangle：`

A faster approach, giving barycentric coordinate directly

Derivation in the discussion section!

$$\vec{O} + t\vec{D} = (1 - b_1 - b_2)\vec{P}_0 + b_1\vec{P}_1 + b_2\vec{P}_2$$

$$\begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{\vec{S}_1 \cdot \vec{E}_1} \begin{bmatrix} \vec{S}_2 \cdot \vec{E}_2 \\ \vec{S}_1 \cdot \vec{S} \\ \vec{S}_2 \cdot \vec{D} \end{bmatrix}$$

Cost = (1 div, 27 mul, 17 add)

Where:

$$\vec{E}_1 = \vec{P}_1 - \vec{P}_0$$
$$\vec{E}_2 = \vec{P}_2 - \vec{P}_0$$
$$\vec{S} = \vec{O} - \vec{P}_0$$
$$\vec{S}_1 = \vec{D} \times \vec{E}_2$$
$$\vec{S}_2 = \vec{S} \times \vec{E}_1$$

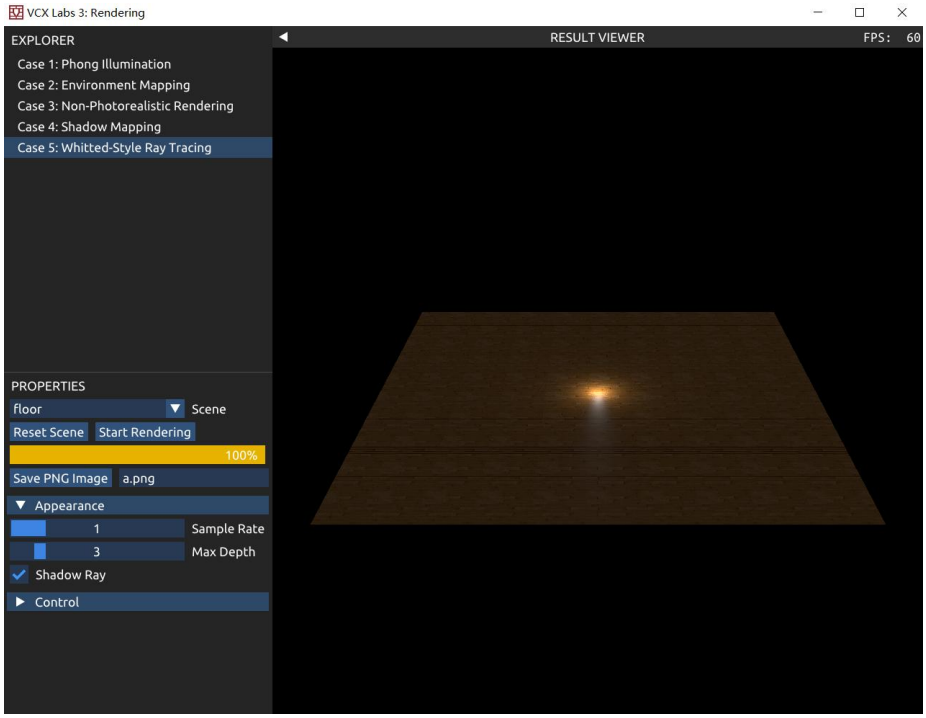
Recall: How to determine if the "intersection" is inside the triangle?

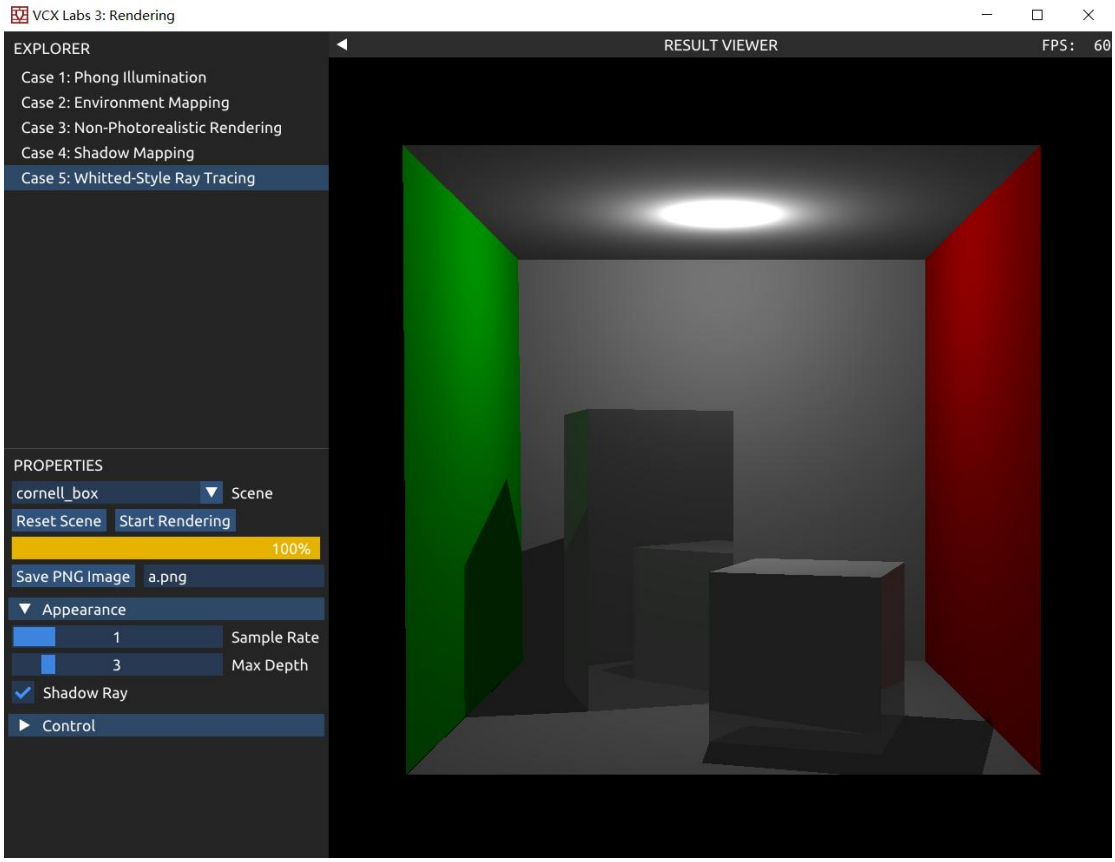
Hint:
(1-b1-b2), b1, b2 are barycentric coordinates!

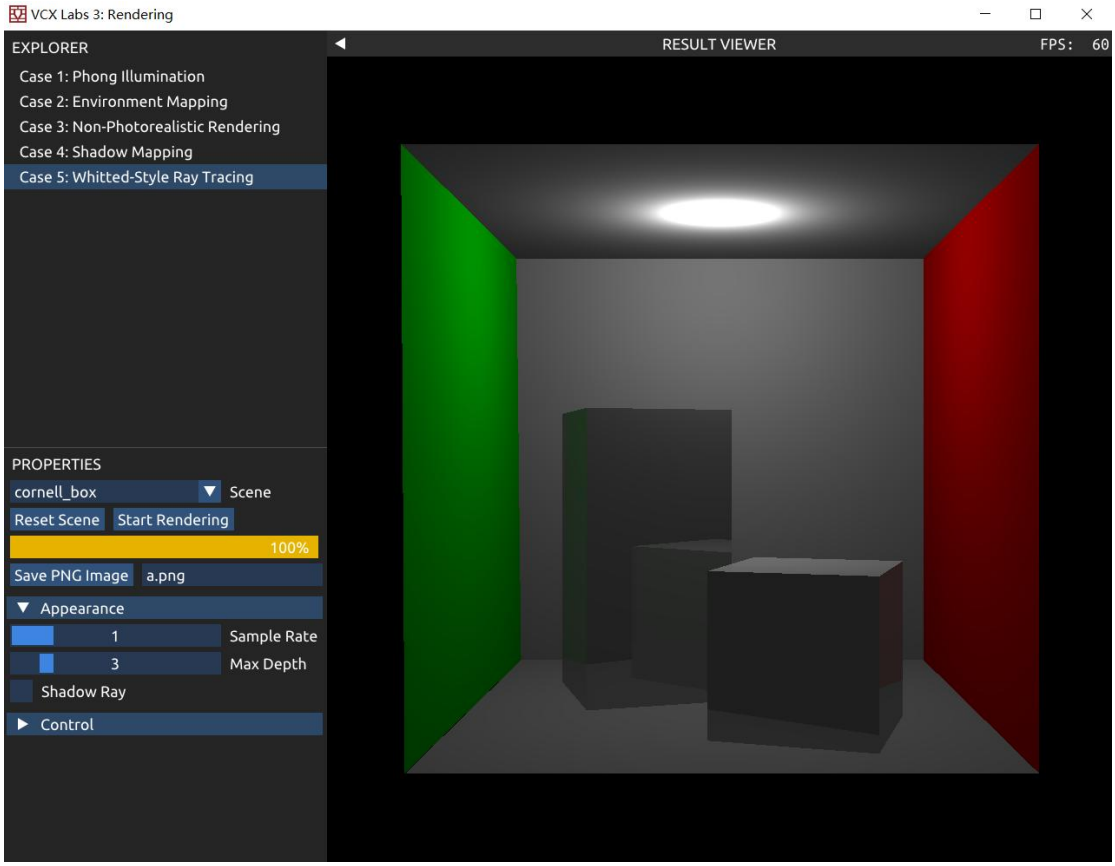
`RayTrace：`

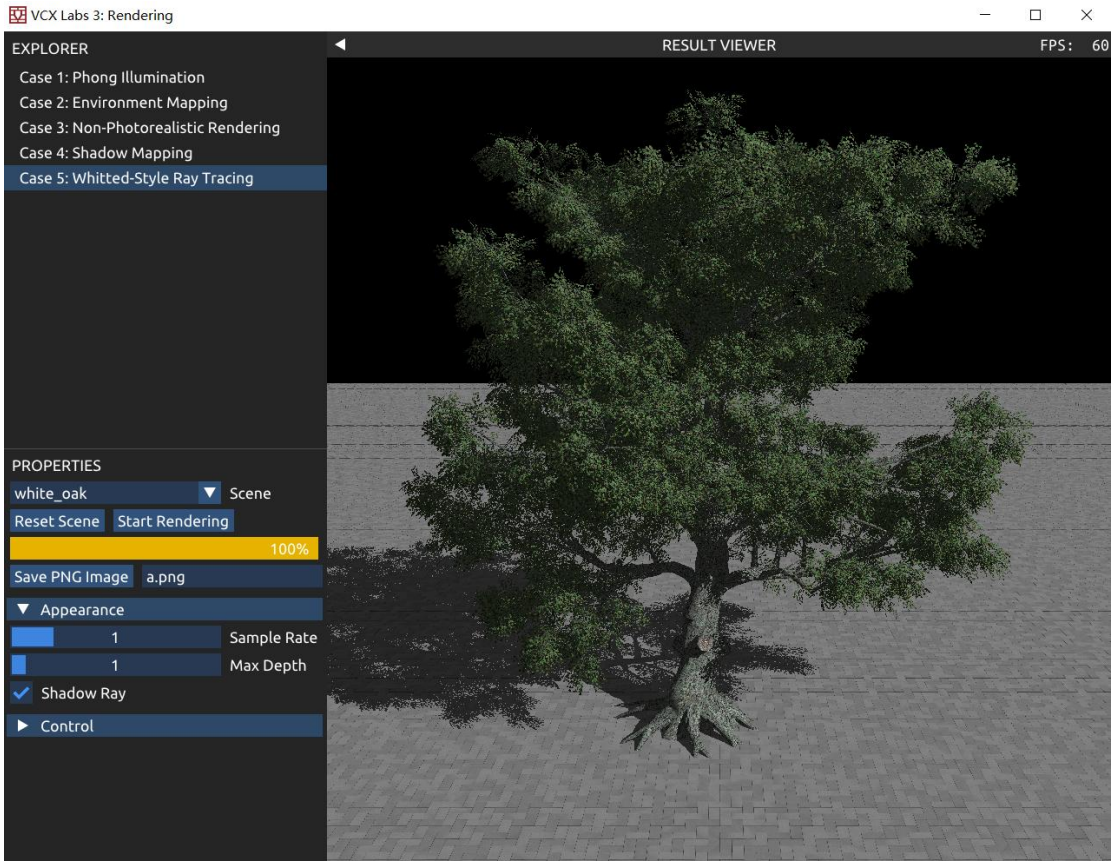
通过每个像素发射一条光线，得到通过这个光线打到场景中的某个物体上来确定这个像素能看到什么，再从这个打到物体上的点发射一条向着光源的光线，根据能不能到达光源来判断该点是否是阴影。Whitted 方法只对反射、折射的材质进行递归处理，当碰到其他材质（例如 diffuse）便停下。

结果如下：




VCX Labs 3: Rendering
EXPLORER
Case 1: Phong Illumination
Case 2: Environment Mapping
Case 3: Non-Photorealistic Rendering
Case 4: Shadow Mapping
Case 5: Whitted-Style Ray Tracing
PROPERTIES
cornell_box
Scene
Reset Scene
Start Rendering
100%
Save PNG Image
a.png
Appearance
1
Sample Rate
3
Max Depth
Shadow Ray
Control
RESULT VIEWER
FPS: 60


VCX Labs 3: Rendering
EXPLORER
Case 1: Phong Illumination
Case 2: Environment Mapping
Case 3: Non-Photorealistic Rendering
Case 4: Shadow Mapping
Case 5: Whitted-Style Ray Tracing
PROPERTIES
cornell_box
Scene
Reset Scene
Start Rendering
100%
Save PNG Image
a.png
Appearance
1
Sample Rate
3
Max Depth
Shadow Ray
Control
RESULT VIEWER
FPS: 60


VCX Labs 3: Rendering
EXPLORER
Case 1: Phong Illumination
Case 2: Environment Mapping
Case 3: Non-Photorealistic Rendering
Case 4: Shadow Mapping
Case 5: Whitted-Style Ray Tracing
PROPERTIES
white_oak
Scene
Reset Scene
Start Rendering
100%
Save PNG Image
a.png
Appearance
1
Sample Rate
1
Max Depth
Shadow Ray
Control
RESULT VIEWER
FPS: 60