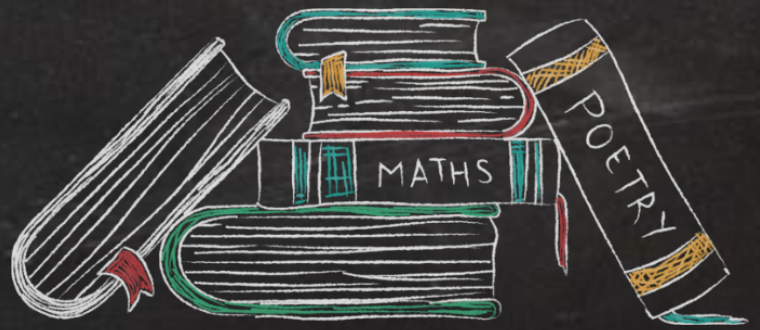
今日内容

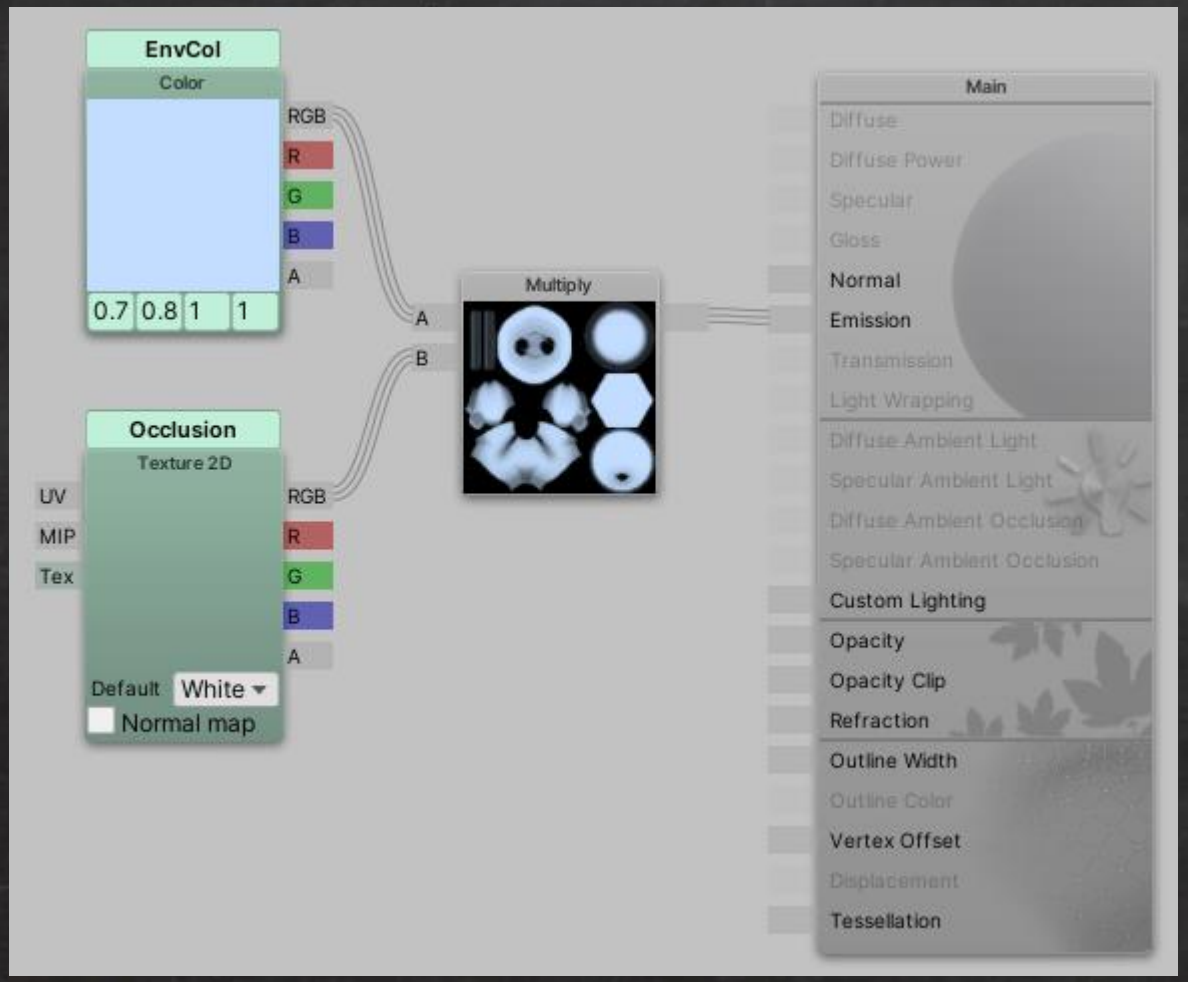
- 幻术·筑基·一重境
- 幻术·筑基·三重境
- 符文·筑基·三重境
- 幻术·筑基·影
- 符文·筑基·影
- 幻术·化神·OldSchool+
- 课业修行

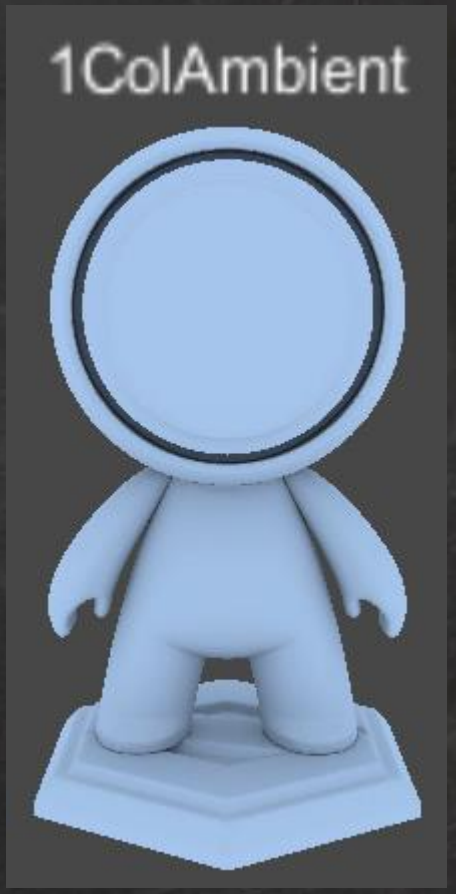
$\frac{24 \times 5}{33}$

1 幻术·筑基·一重境



01 单色环境光·1ColAmbient

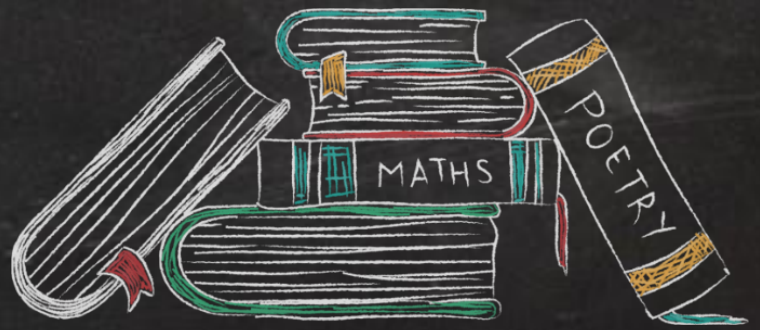




（演示）

$\frac{24 \times 5}{33}$

2 幻术·筑基·三重境



01 三色环境光·3ColAmbient

（演示）

EnvUpCol
Color
RGB R G B A
1 1 1 1

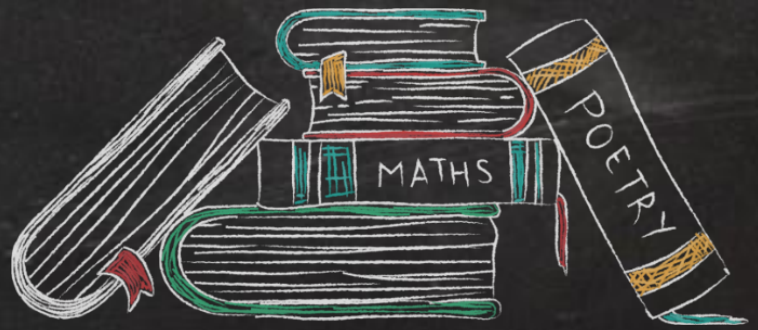
EnvSideCol
Color
RGB R G B A
0.4 0.4 0.4 1

EnvDownCol
Color
RGB R G B A
0 0 0 1

// 向上部分遮罩
// 侧面部分遮罩
// 向下部分遮罩
// 环境遮挡
// 混合三重环境色

Normal Dir.
N
Perturbed
In
Comp. Mask
G
Value 0
Value 1
Value -1
Max
Subtract
Subtract
Multiply
Max
Multiply
Multiply
Multiply
Add
Occlusion
Texture 2D
UV MIP Tex
RGB R G B A
Default White
Normal map
Add
Multiply
Normal
Emission
Transmission
Light Wrapping
Diffuse Ambient Light
Specular Ambient Light
Diffuse Ambient Occlusion
Specular Ambient Occlusion
Custom Lighting
Opacity
Opacity Clip
Refraction
Outline Width
Outline Color
Vertex Offset
Displacement

3 符文·筑基·三重境



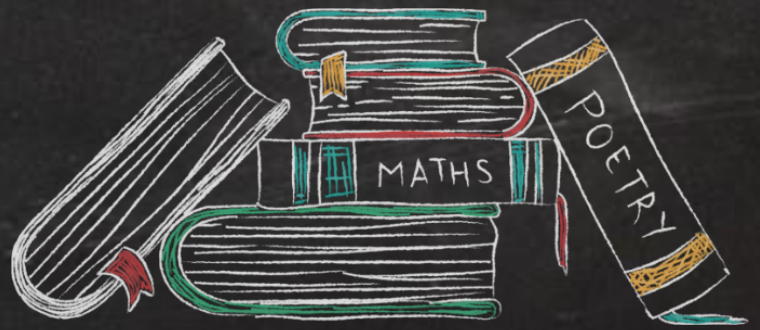
01 三色环境光代码实现·3ColAmbient

1. 以手写Lambert （第3课）作为模板，复制粘贴大法；
2. 修改Shader路径名；
3. 定义面板参数：
   - 贴图参数的定义方法：_XXX（“面板标签”，2d） = “white” {}
   - “white” {} 代表缺省纹理为纯白贴图，其他还有“black” {} “gray” {} …
4. 对应的声明输入参数；
5. 输入结构追加uv0；
6. 输出结构追加uv；
7. 顶点Shader中对uv做赋值；
8. 准备向量 nDir；
9. 通过nDir计算朝上，朝下，侧面各部位遮罩；
10. 通过部位遮罩混合最终环境光颜色；
11. 采样Occlusion图，获得环境遮挡信息；采样贴图方法：tex2D(_Texture, uv)；
12. 用环境遮挡信息对环境光做遮挡；
13. 输出最终颜色。

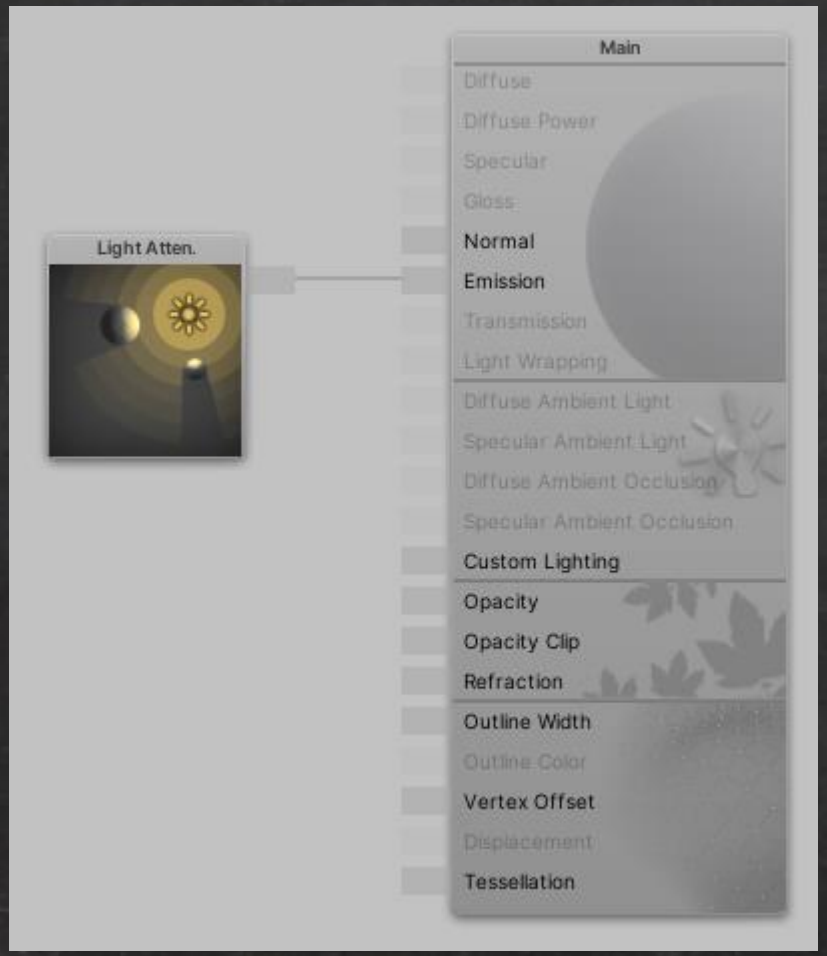
```
Shader "AP1/L07/3ColAmbient" {
    Properties {
        _Occlusion  ("环境遮挡图", 2d)      = "white" {}
        _EnvUpCol   ("朝上环境色", color)   = (1.0, 1.0, 1.0, 1.0)
        _EnvSideCol ("侧面环境色", color)   = (0.5, 0.5, 0.5, 1.0)
        _EnvDownCol ("朝下环境色", color)   = (0.0, 0.0, 0.0, 1.0)
    }
    SubShader {
        Tags {
            "RenderType"="Opaque"
        }
        Pass {
            Name "FORWARD"
            Tags {
                "LightMode"="ForwardBase"
            }
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag
            #include "UnityCG.cginc"
            #pragma multi_compile_fwdbase_fullshadows
            #pragma target 3.0
            // 输入参数
            uniform float3 _EnvUpCol;
            uniform float3 _EnvSideCol;
            uniform float3 _EnvDownCol;
            uniform sampler2D _Occlusion;
            // 输入结构
            struct VertexInput {
                float4 vertex : POSITION;   // 将模型顶点信息输入进来
                float4 normal : NORMAL;     // 将模型法线信息输入进来
                float2 uv0 : TEXCOORD0;     // 将模型UV信息输入进来 0通道 共4通道
            };
            // 输出结构
            struct VertexOutput {
                float4 pos : SV_POSITION;   // 由模型顶点信息换算而来的顶点屏幕位置
                float3 nDirWS : TEXCOORD0;  // 由模型法线信息换算来的世界空间法线信息
                float2 uv : TEXCOORD1;      // 追加UV信息用语像素Shader采样贴图
            };
            // 输入结构>>>顶点Shader>>>输出结构
            VertexOutput vert (VertexInput v) {
                VertexOutput o = (VertexOutput)0;                   // 新建一个输出结构
                o.pos = UnityObjectToClipPos( v.vertex );           // 变换顶点信息 并将其塞给输出结构
                o.nDirWS = UnityObjectToWorldNormal(v.normal);      // 变换法线信息 并将其塞给输出结构
                o.uv = v.uv0;                                       // 图森破
                return o;                                           // 将输出结构 输出
            }
            // 输出结构>>>像素
            float4 frag(VertexOutput i) : COLOR {
                // 准备向量
                float3 nDir = i.nDirWS;                             // 获取nDir
                // 计算各部位遮罩
                float upMask = max(0.0, nDir.g);                    // 获取朝上部分遮罩
                float downMask = max(0.0, -nDir.g);                 // 获取朝下部分遮罩
                float sideMask = 1.0 - upMask - downMask;           // 获取侧面部分遮罩
                // 混合环境色
                float3 envCol = _EnvUpCol * upMask + _EnvSideCol * sideMask + _EnvDownCol * downMask;
                // 采样Occlusion贴图
                float occlusion = tex2D(_Occlusion, i.uv);
                // 计算环境光照
                float3 envLighting = envCol * occlusion;
                // 返回最终颜色
                return float4(envLighting, 1.0);
            }
            ENDCG
        }
    }
    FallBack "Diffuse"
}
```

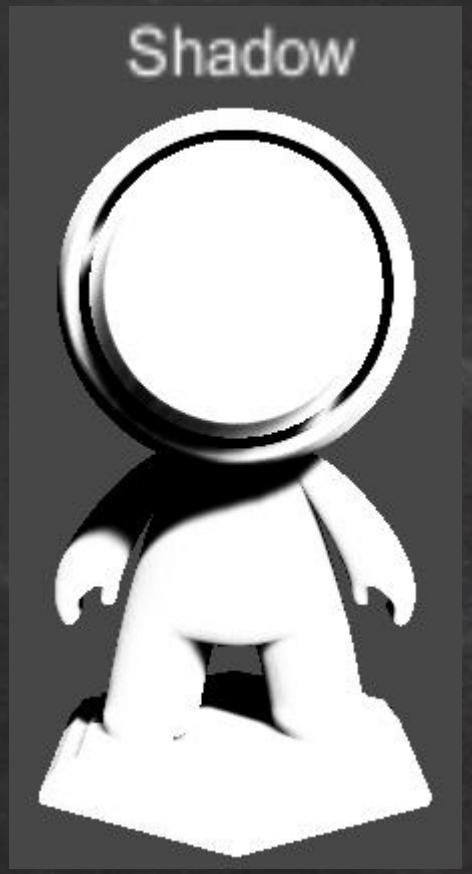
$\frac{24\times5}{33}$

4 幻术·筑基·影



01 Unity自带投影实现

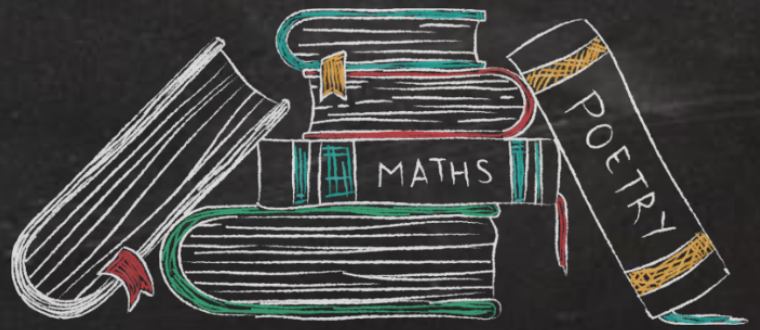




(演示)

$\frac{24 \times 5}{33}$

5 符文·筑基·影



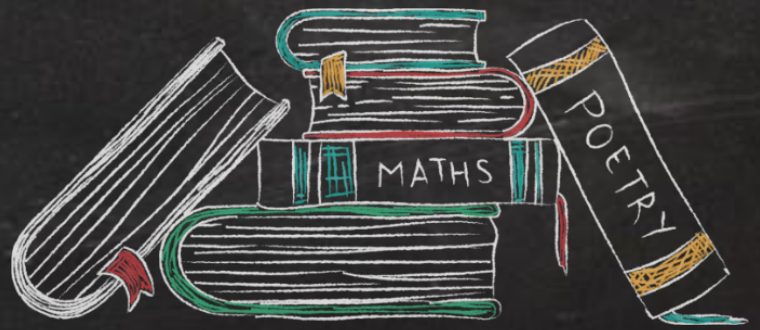
01 Unity内置投影代码调用方法

1. 以手写FlatCol（第3课）作为模板，复制粘贴大法；
2. 修改Shader路径名；
3. 无需面板参数；
4. 无需声明输入参数；
5. 无需修改输入结构；
6. 输出结构追加：LIGHTING_COORDS(0, 1)：
   - 此为Unity封装好的输出结构内容，照写就行，暂时不看细节；
   - 括号中的参入，如(0, 1)；0，1分别代表占用了TEXCOORD1和TEXCOORD2；
7. 顶点Shader中必须调用Unity封装好的方法：TRANSFER_VERTEX_TO_FRAGMENT(o);
8. 像素Shader中获取投影信息同样通过Unity提供的方法：LIGHT_ATTENUATION(i)；
9. 将投影作为结果输出。

```
Shader "AP1/L07/Shadow" {
    Properties {
    }
    SubShader {
        Tags {
            "RenderType"="Opaque"
        }
        Pass {
            Name "FORWARD"
            Tags {
                "LightMode"="ForwardBase"
            }
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag
            #include "UnityCG.cginc"
            #include "AutoLight.cginc"        // 使用Unity投影必须包含这两个库文件
            #include "Lighting.cginc"         // 同上
            #pragma multi_compile_fwdbase_fullshadows
            #pragma target 3.0
            // 输入结构
            struct VertexInput {
                float4 vertex : POSITION;    // 将模型的顶点信息输入进来
            };
            // 输出结构
            struct VertexOutput {
                float4 pos : SV_POSITION;    // 由模型顶点信息换算而来的顶点屏幕位置
                LIGHTING_COORDS(0,1)         // 投影用坐标信息 Unity已封装 不用管细节
            };
            // 输入结构>>>顶点Shader>>>输出结构
            VertexOutput vert (VertexInput v) {
                VertexOutput o = (VertexOutput)0;              // 新建一个输出结构
                o.pos = UnityObjectToClipPos( v.vertex );      // 变换顶点信息 并将其塞给输出结构
                TRANSFER_VERTEX_TO_FRAGMENT(o)                 // Unity封装 不用管细节
                return o;                                      // 将输出结构 输出
            }
            // 输出结构>>>像素
            float4 frag(VertexOutput i) : COLOR {
                float shadow = LIGHT_ATTENUATION(i);           // 同样Unity封装好的函数 可取出投影
                return float4(shadow, shadow, shadow, 1.0);
            }
            ENDCG
        }
    }
    FallBack "Diffuse"
}
```

$\frac{24 \times 5}{33}$

6 幻术·化神·OldSchool+



01 简化理解光照构成

对光的遮挡就是投影

光照

光源

漫反射

镜面反射

遮挡

环境

漫反射

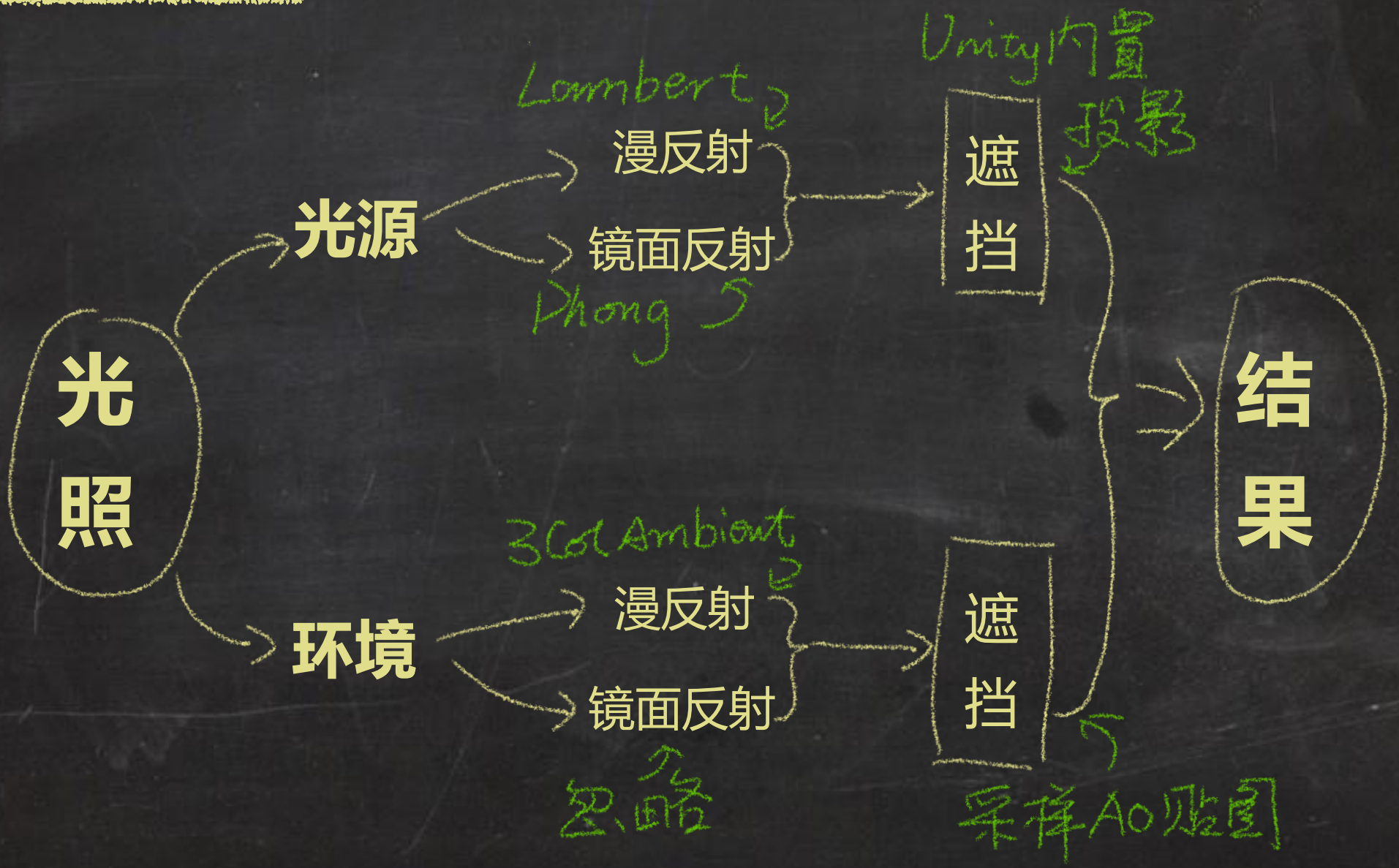
镜面反射

遮挡

结果

对环境的遮挡就是AO

02 OldSchoolPlus构成



03 一锅炖不下

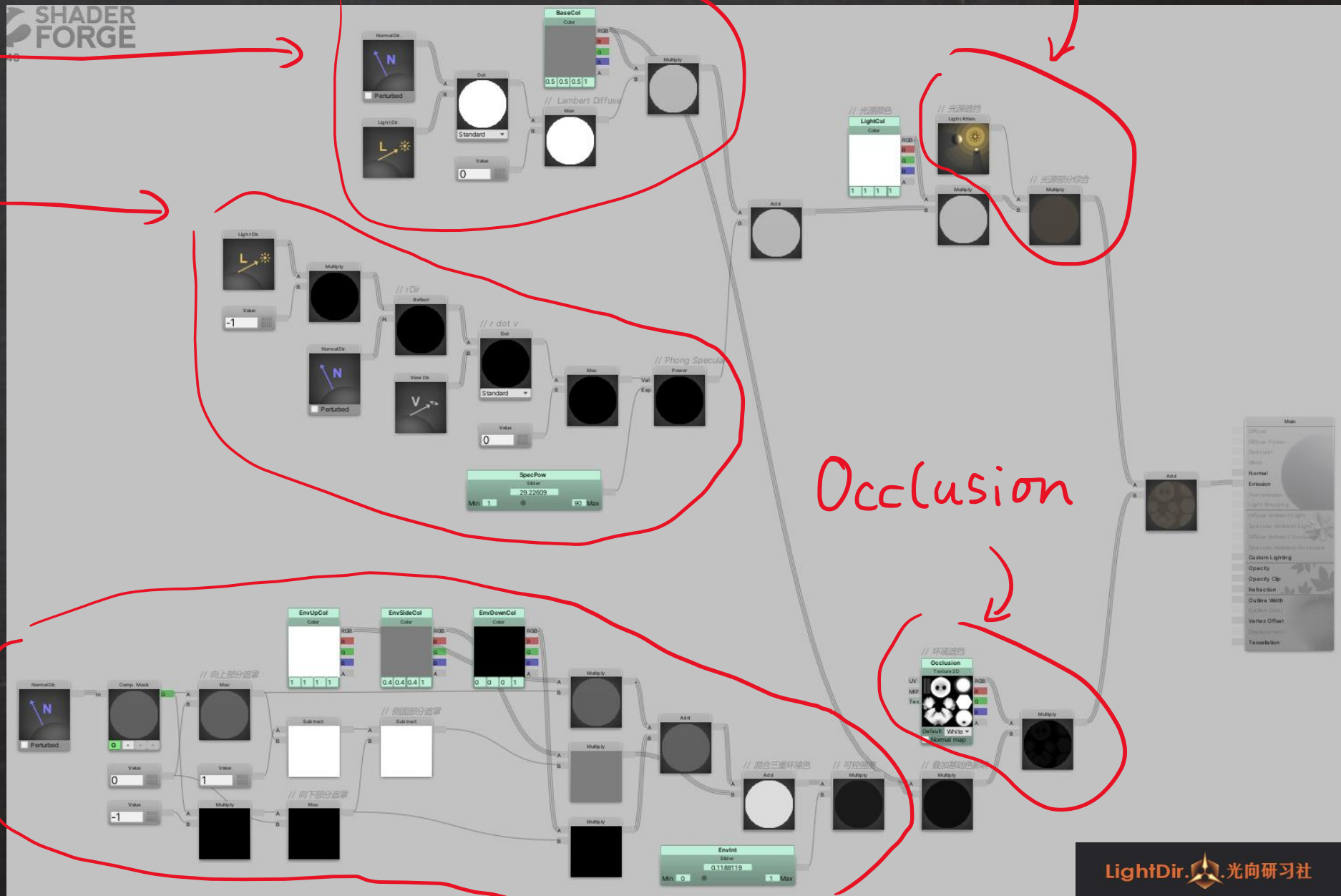
Lambert

Phong

3Col Ambient

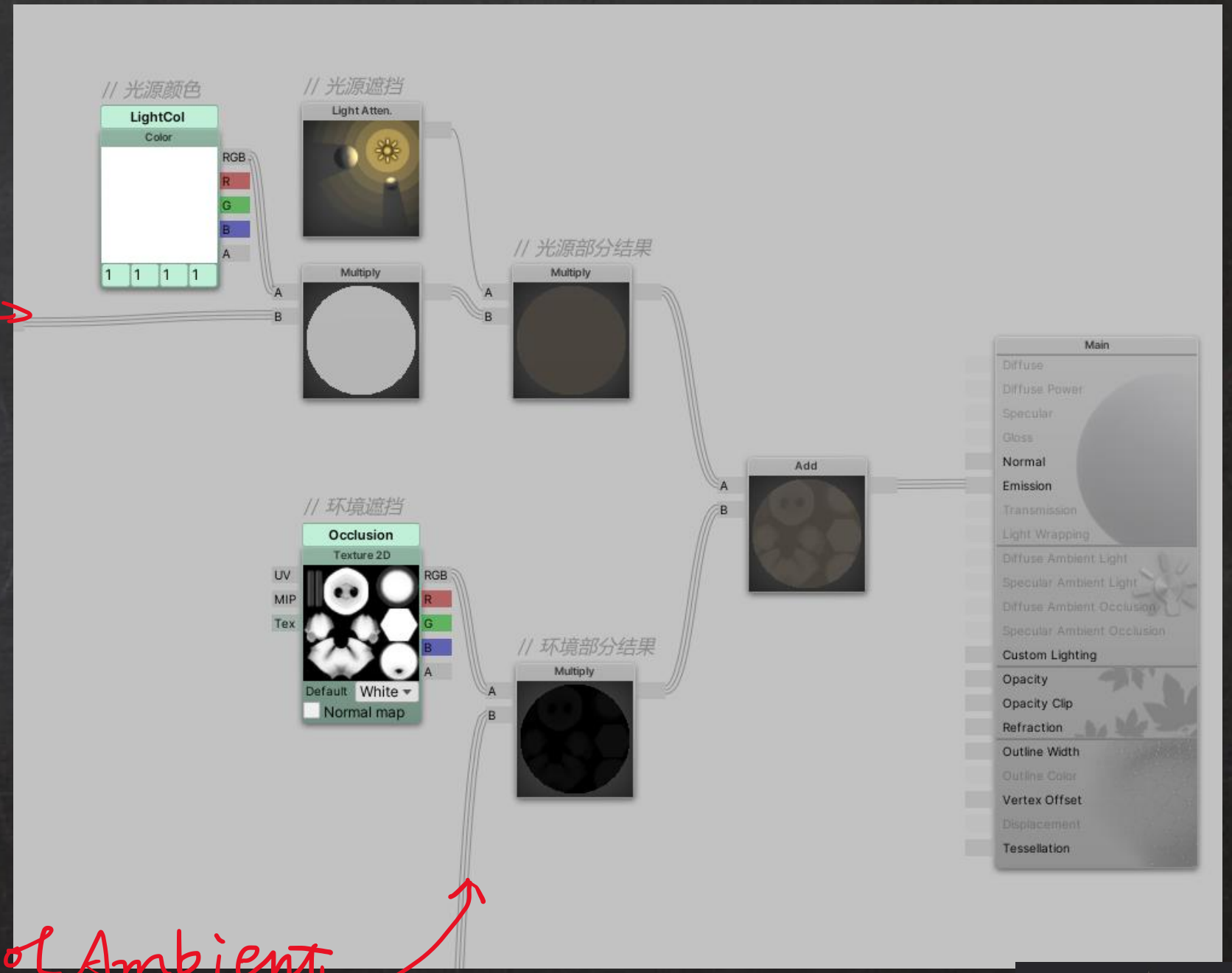
Shadow

Occlusion



04 混合细节

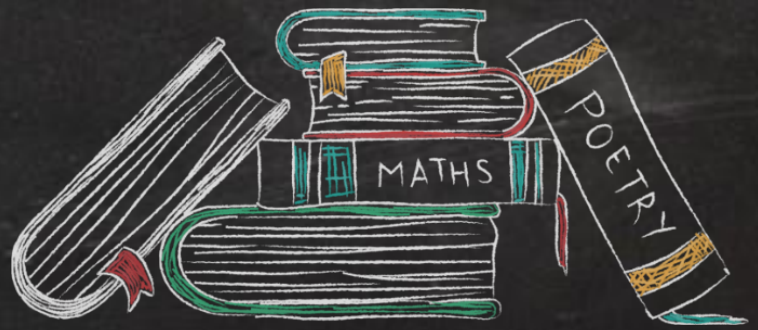
Lambert + Phong



3 Col Ambient

$\frac{24 \times 5}{33}$

7 课业修行



01 可以搞点事情了

- 连连看作业：
  - 3ColAmbient，OldSchoolPlus；
- 符文作业：
  - 3ColAmbient，OldSchoolPlus；
- 创意作业：
  - 建议必做：找一个模型，角色场景随意，尝试应用OldSchoolPlus；
  - 选做：以OldSchoolPlus为基础，观察并改进为你自己光照模型；