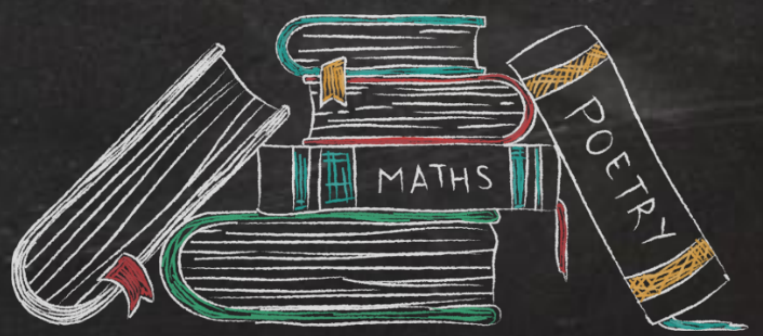

大家好
欢迎加入LightDir（光向）研习社
欢迎大家一同探索开源共享的知识分享模式

今日内容

- 瞳术·漫返&镜返
- 幻术·筑基·镜返
- 符文·漫返&镜返
- 作业

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

1 瞳术·漫返&镜返



01 回顾下初中物理

漫反射-Diffuse:

- 漫无目的，四面八方，均匀的反射；
- 追求漫反射的列子：电影荧幕；

镜面反射-Specluar:

- 有目的，沿反射方向，不均匀的反射；
- 追求漫反射的列子：化妆镜，车漆；

02 黑话·各种向量

常用向量：（全要记）

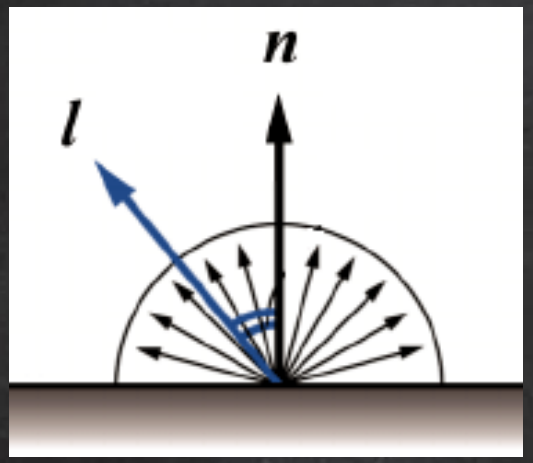
- nDir：法线方向，点乘操作时简称n；
- lDir：光照方向，点乘操作时简称l；
- vDir：观察方向，点乘操作时简称v；
- rDir：光反射方向，点乘操作时简称r；
- hDir：半角方向(Halfway)，lDir和vDir的中间角方向，点乘操作时简称h；

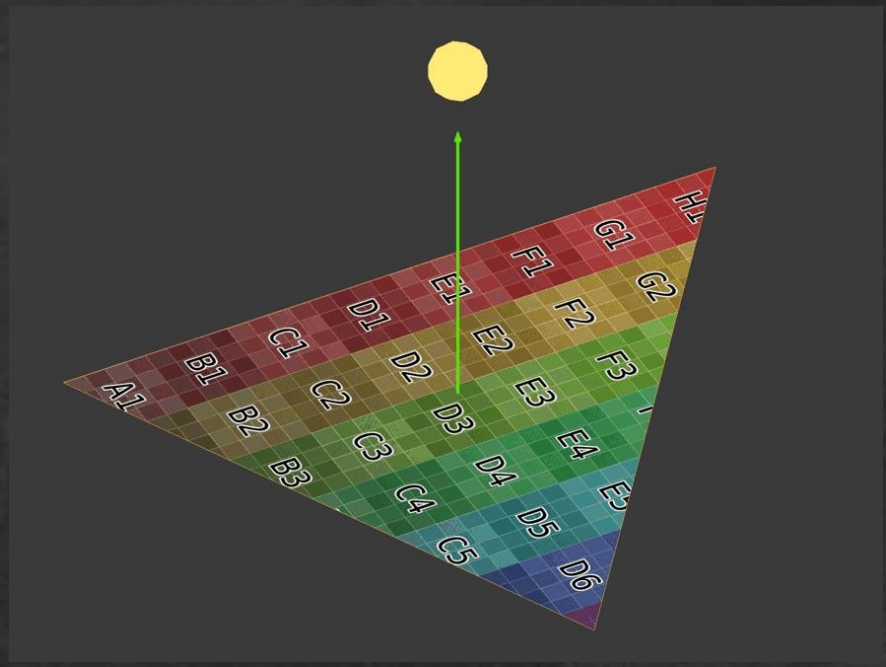
所在空间：（暂时只记WS，其余看热闹）

- OS：ObjectSpace 物体空间，本地空间；
- WS：WorldSpace 世界空间；
- VS： ViewSpace 观察空间；
- CS：HomogenousClipSpace 齐次剪裁空间；
- TS：TangentSpace 切线空间；
- TXS：TextureSpace 纹理空间；

例：nDirWS：世界空间下的法线方向；

03 漫反射



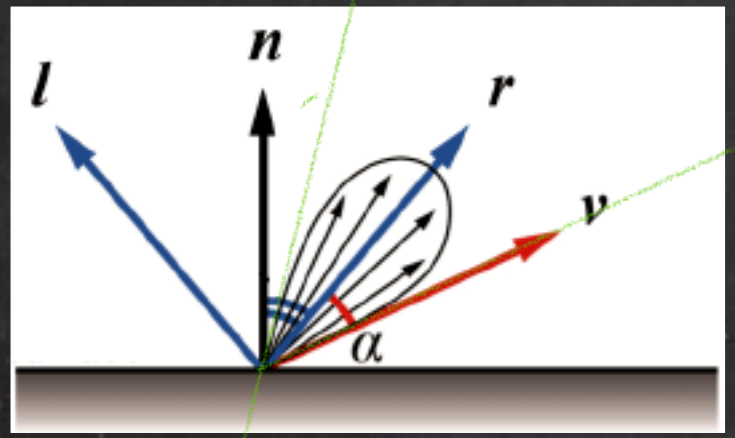


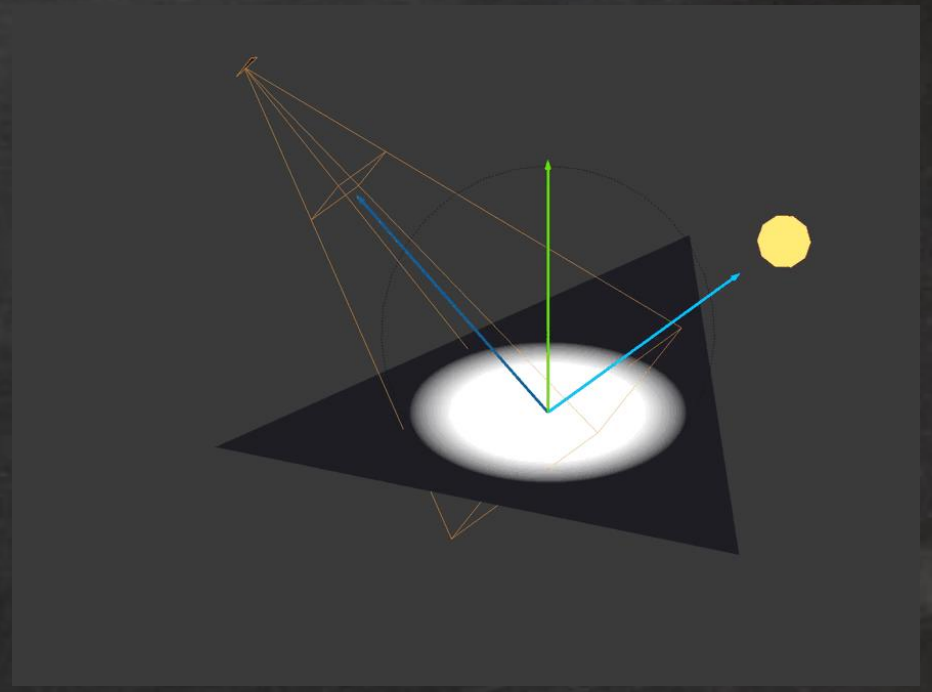
漫反射-Diffuse:

- 因其向四面八方均匀散射，所以反射亮度和观察者看的方向无关；
- 实现方式：Lambert（n dot l），显然vDir不参与计算；

04 镜面反射

可观察 Specular 的视角范围



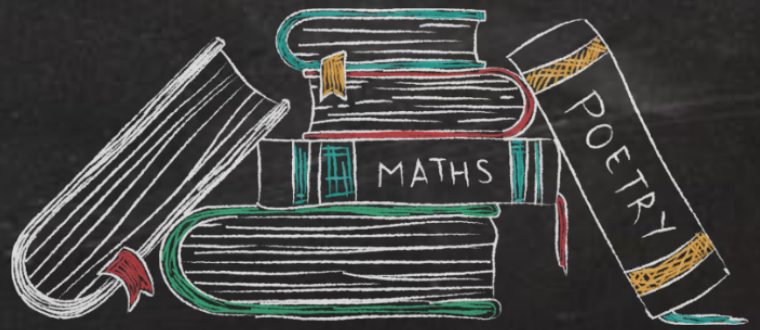


镜面反射-Specular:

- 因其反射有明显方向性，所以观察者的视角决定了反射光线的有无，明暗；
- 实现方式：
  - Phong（r dot v），即光反射方向和视角方向越重合，反射越强；
  - Blinn-Phong（n dot h），即法线方向和半角方向越重合，反射越强；

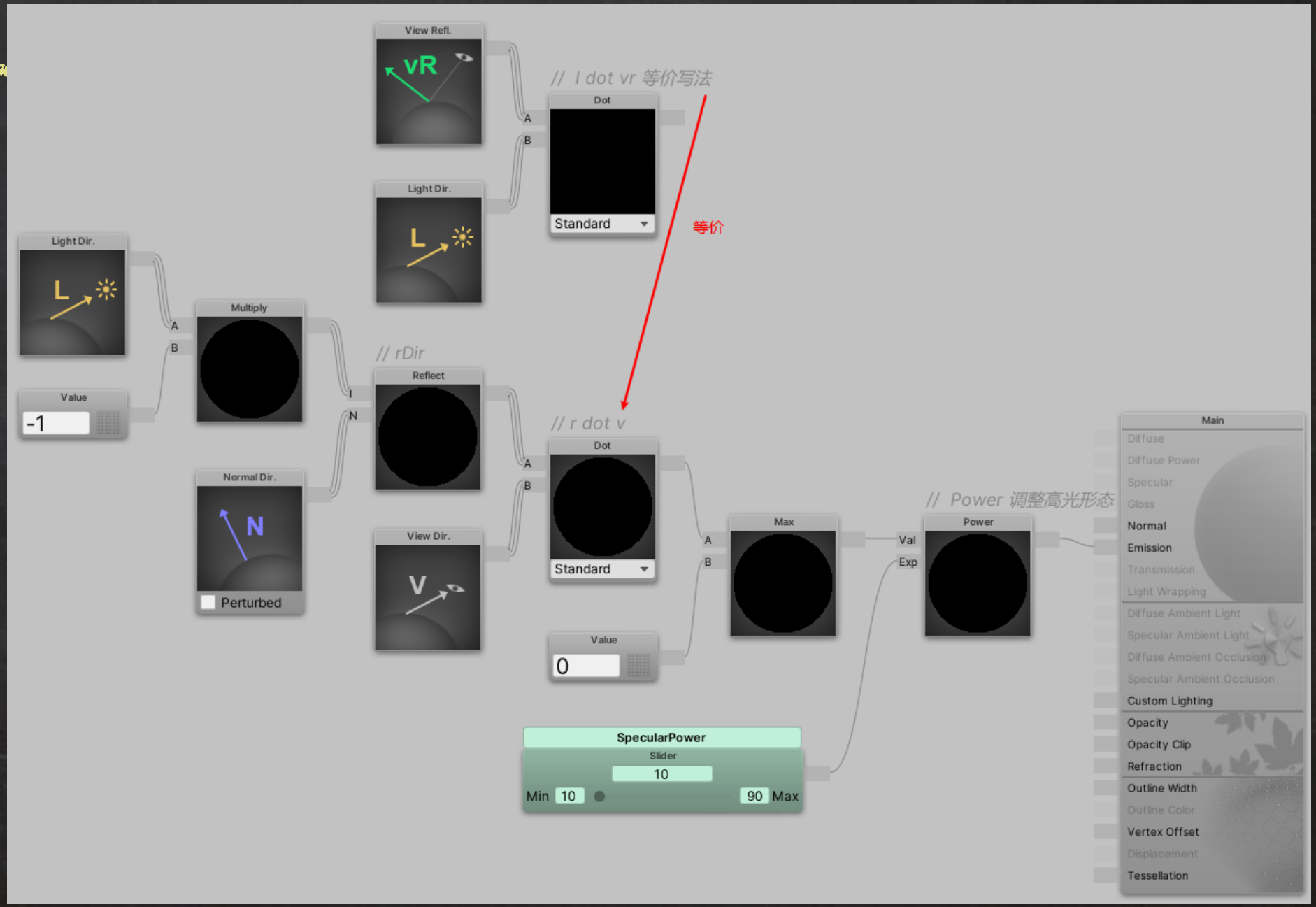
$\frac{24 \times 5}{33}$

2 幻术·镜返



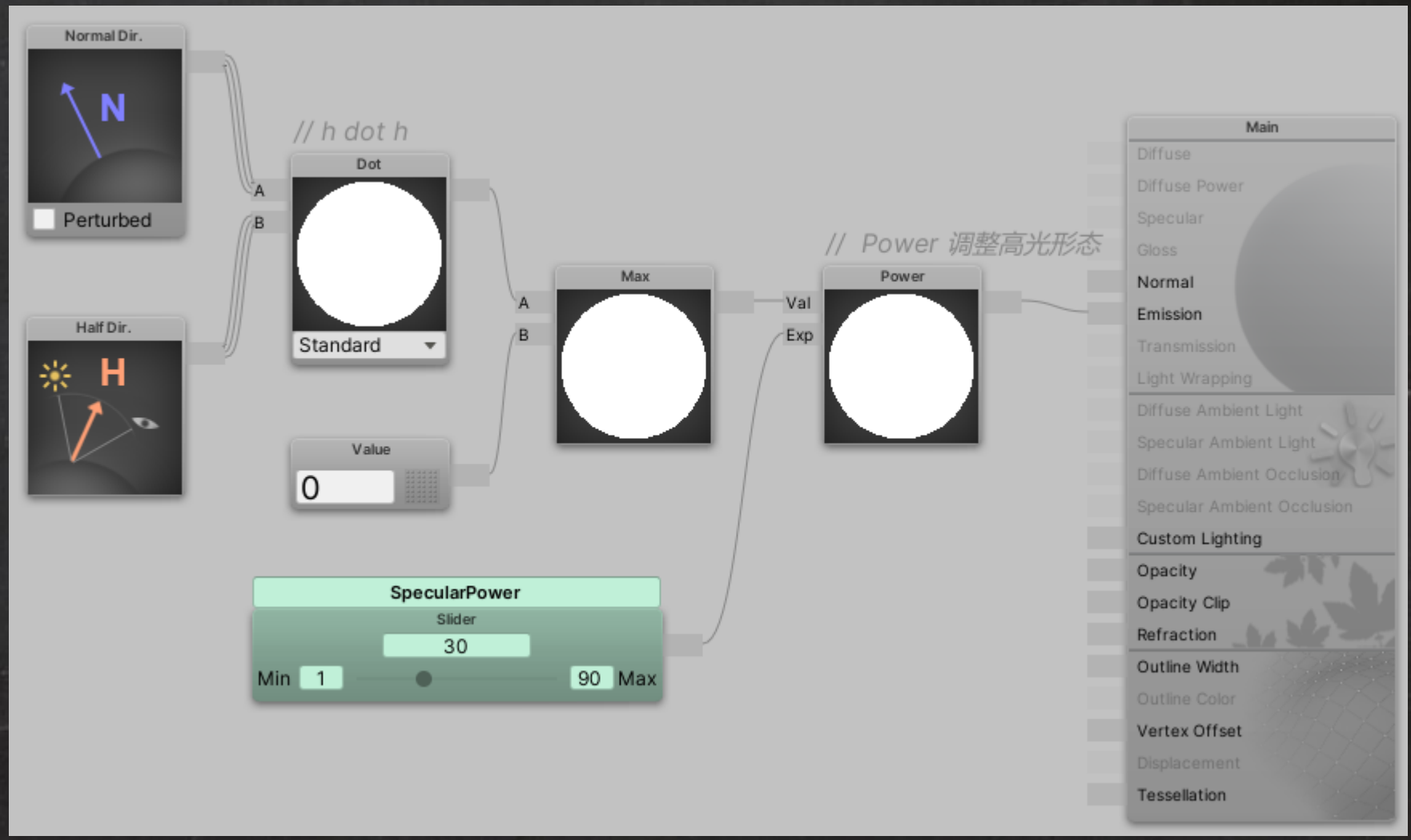
01 Phong

- Phong（r dot v）
  - rDir=Reflect(-lDir, nDir);
- Power是啥?
  - 一般叫高光次幂;
  - PS里正片叠底叠一叠;



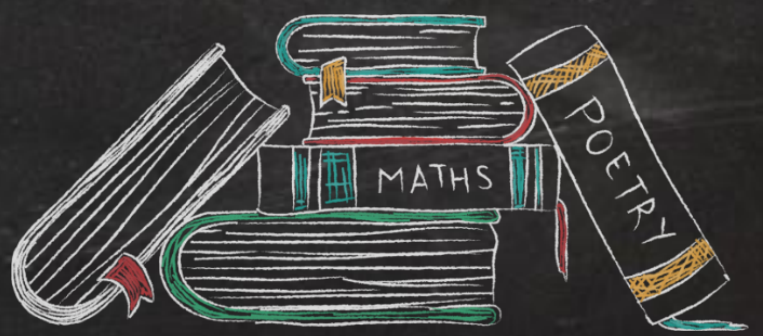
02 Blinn-Phong

- B-Phong（n dot h）
  - hDir，SF给爷备好了诶;



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

3 符文·漫返&镜返



01 祭出你的模板

- 我们准备写一个Lambert漫反射+Blinn-Phong镜面反射这样一个光照模型相对完善的Shader；这是一种上古套路，所以吾称之为OldSchool；
- 回顾第3课，建议大家用FlatCol或者Lambert作为自己的Shader模板，避免从0开始；
- 拿到模板第一件事，改路径！改路径！改路径！Shader重名会发生奇怪的事情；
- 路径规范命名：AP1/L05/OldSchool，意思是它是AP1这个工程中，L05第5课这个分支里，一个叫OldSchool的一个Shader；

```
Shader "AP1/L05/OldSchool" {
    Properties {
    }
    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
            };
            // 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
            };
            // 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
                return o;                                           // 将输出结构 输出
            }
            // 输出结构>>>像素
            float4 frag(VertexOutput i) : COLOR {
                float3 nDir = i.nDirWS;                             // 获取nDir
                float3 lDir = _WorldSpaceLightPos0.xyz;             // 获取lDir
                float nDotl = dot(i.nDirWS, lDir);                  // nDir点积lDir
                float lambert = max(0.0, nDotl);                    // 截断负值
                return float4(lambert, lambert, lambert, 1.0);      // 输出最终颜色
            }
            ENDCG
        }
    }
    FallBack "Diffuse"
}
```

02 添加材质参数

- 需要暴露一个参数：高光次幂；和一个物体基本颜色的参数；
- 暴露参数需要在两个地方整活：
  - Properties{} 段里定义材质面板；
  - VertexInput前声明参数（我习惯在这，也可以使用前再声明）；

```
Properties {
    _MainCol ("颜色", color) = (1.0, 1.0, 1.0, 1.0)
    _SpecularPow ("高光次幂", range(1, 90)) = 30
}
```

- Properties段定义参数的格式：
  - 格式：_名称（“面板标签”，类型(参数，可无)）= 默认值
  - 如第一行：
    - 变量名：_MainCol（起手下划线，首字大写）
    - 面板标签：颜色
    - 类型：color 颜色类型 无参数
    - 默认值：（1，1，1，1）白色

```
#pragma target 3.0
// 输入参数
// 修饰字（满足小朋友太多的问号,保全发量的大家看热闹,其实不写好像也闹太套）
    // uniform  共享于vert,frag
    // attibute 仅用于vert
    // varying  用于vert,frag传数据
uniform float3 _MainCol;     // RGB够了 float3
uniform float _SpecularPow;  // 标量 float
// 输入结构
```

- VertexInput前的参数声明：
  - 要和Properties段定义的参数一一对应；
  - 适当的选取数据类型；

03 修改输入输出结构和顶点Shader

- 像素Shader需要vDir，计算vDir需要世界空间顶点位置posWS；
  - vDir = 归一化（摄像机位置 – posWS）
  - posWS的变换方法可从SF结点拆解出来

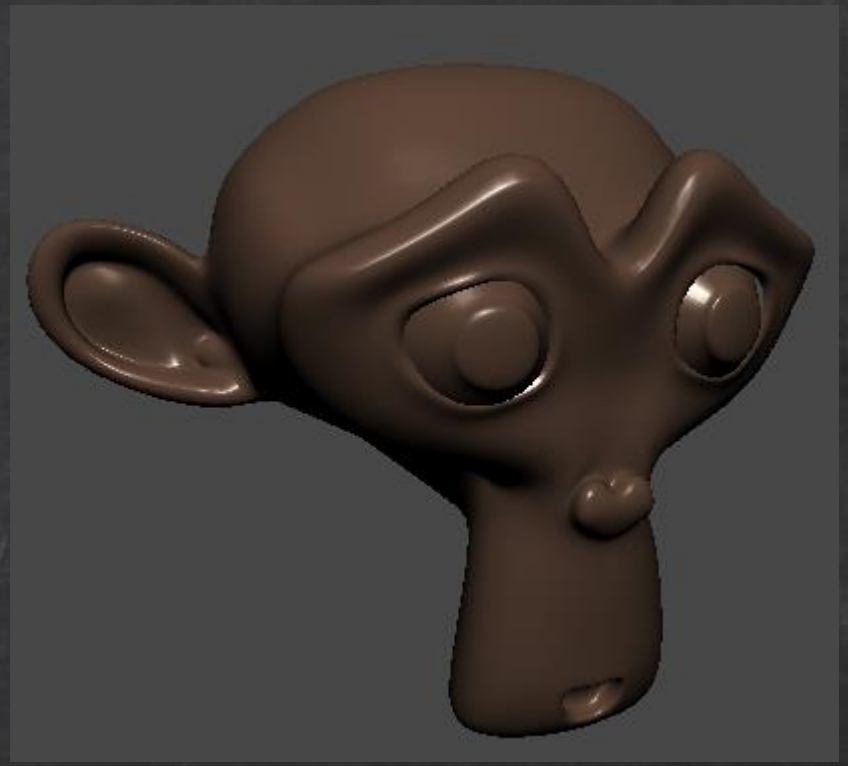
```
// 输入结构
struct VertexInput {
    float4 vertex : POSITION;    // 顶点信息 Get✔
    float4 normal : NORMAL;      // 法线信息 Get✔
};
// 输出结构
struct VertexOutput {
    float4 posCS : SV_POSITION;      // 裁剪空间（暂理解为屏幕空间吧）顶点位置
    float4 posWS : TEXCOORD0;        // 世界空间顶点位置
    float3 nDirWS : TEXCOORD1;       // 世界空间法线方向
};
// 输入结构>>>顶点Shader>>>输出结构
VertexOutput vert (VertexInput v) {
    VertexOutput o = (VertexOutput)0;                          // 新建输出结构
        o.posCS = UnityObjectToClipPos( v.vertex );            // 变换顶点位置 OS>CS
        o.posWS = mul(unity_ObjectToWorld, v.vertex);          // 变换顶点位置 OS>WS
        o.nDirWS = UnityObjectToWorldNormal(v.normal);         // 变换法线方向 OS>WS
    return o;                                                  // 返回输出结构
}
```

04 像素Shader

- 像素Shader稍长一点，但是其实一点不难；普及下我的代码风格，**四段法**去理解：
  1. 准备所有要用到的向量，不会计算去SF拆；
  2. 准备所有要用到的中间数据，这里是两个点积结果；
  3. 编写光照模型；
  4. 后处理（本例无），返回结果；

```
// 输出结构>>>像素
float4 frag(VertexOutput i) : COLOR {
    // 准备向量
    float3 nDir = i.nDirWS;
    float3 lDir = _WorldSpaceLightPos0.xyz;
    float3 vDir = normalize(_WorldSpaceCameraPos.xyz - i.posWS.xyz);
    float3 hDir = normalize(vDir + lDir);
    // 准备点积结果
    float ndotl = dot(nDir, lDir);
    float ndoth = dot(nDir, hDir);
    // 光照模型
    float lambert = max(0.0, ndotl);
    float blinnPhong = pow(max(0.0, ndoth), _SpecularPow);
    float3 finalRGB = _MainCol * lambert + blinnPhong;
    // 返回结果
    return float4(finalRGB, 1.0);
}
```

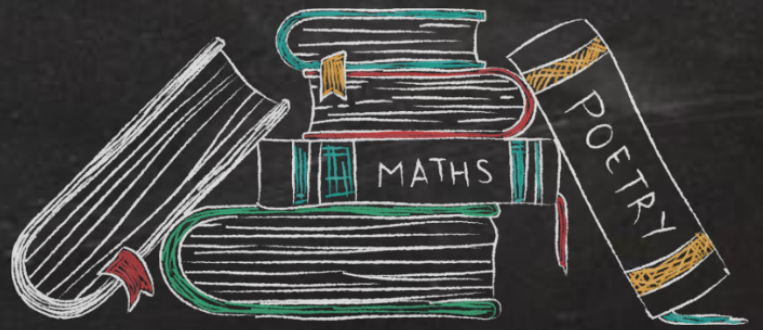
05 结果



纵享丝滑

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

4 作业



01 作业

- 连连看作业：
  - 本节所有连连看例子：Phong，Blinn-Phong，OldSchool；
  - OldSchool·改（Lambert+Phong）
- 符文作业：
  - OldSchool
  - 尝试OldSchool·改
- 创意：
  - 不限，尽量根据已学知识发挥创意。


Thanks