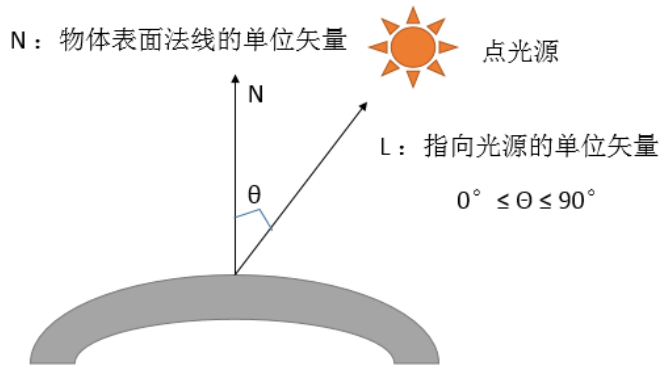
# Unity Shader_6---------标准光照模型

## 标准光照模型

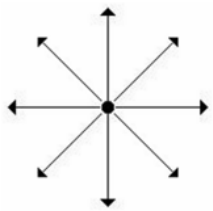
着色：根据材质属性（漫反射、镜面反射等）、光照信息（光源方向、辐射度等），使用一个等式去计算沿某个观察方向的出射度的过程，这个过程可称之为光照模型（Lighting Model）

**· 漫反射**

**Lambert**




漫反射图示



对那些被物体表面散射到各个方向的辐射度进行建模，
视角位置不重要，反射完全随机，
一般可认为在任何的反射方向上的反射光的分布是一样的，
主要受 入射光的角度 影响
漫反射的光强仅与入射光的方向和反射点表面的法线单位向量的夹角余弦值成正比

Lambert兰伯特光照模型

I(diffuse) = I(Ambient)+I(light) Kd(0<Kd<1：对漫反射光的反射属性) (L · N)

光照函数实现方法：

| c(diffuse) | = (c(light)· m(diffuse)) | max(0,L · N) |
| --- | --- | --- |
| 当前物体表面颜色值 | 光源颜色 材质漫反射颜色 | |

unity内置Lighting.cginc中给出的Lambert光照模型 部分参考代码：

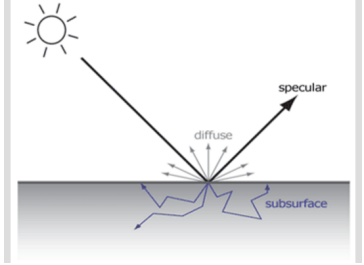
```
fixed4 _LightColor0;

inline fixed4 LightingLambert (SurfaceOutput s, fixed3 lightDir, fixed atten)
{
    fixed diff = max (0, dot (s.Normal, lightDir));

    fixed4 c;
    c.rgb = s.Albedo * _LightColor0.rgb * (diff * atten * 2);
    c.a = s.Alpha;
    return c;
}
```

**扩展：半兰伯特光照模型（视觉加强技术）：为Subsuface Scattering(表面散射)**

Direct surface scattering (left), plus **subsurface scattering** (middle), create the final image on the right.



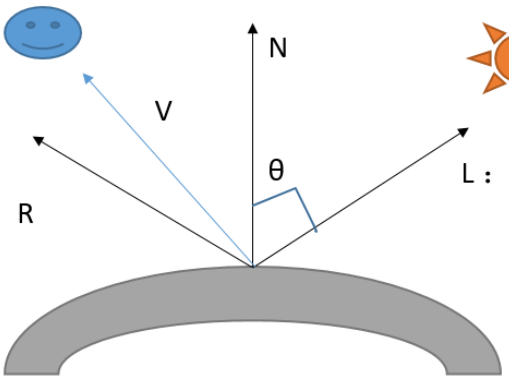

半Lambert兰伯特光照模型（纯 视觉加强技术，
直观：亮度增强）
I(diffuse) = I(Ambient)+I(light) Kd(0<Kd<1：对漫反射光的反射属性) (L · N)

光照函数实现方法：
c(diffuse) = (c(light)· m(diffuse)) (α(N·L)+β) //移除了max(0,L · N) 一般α、 β均取0.5
当前物体表面颜色值 光源颜色 材质漫反射颜色 α倍的缩放加上β偏移=> 模型背光面
从全0即仅售全局光影响 到 背光面也有明暗变化

**· 镜面反射**

**Phong光照模型**

N： 物体表面法线的单位矢量




点光源

L： 指向光源的单位矢量

0° ≤ Θ ≤ 90°
注意：入射角 = 反射角
所以：R = 2（nor(N) · L）N - L

Phong光照模型：
I(spec) = I(Amibent) + I(light) Kd(0<Kd<1：对漫反射光的反射属性) (L · N)

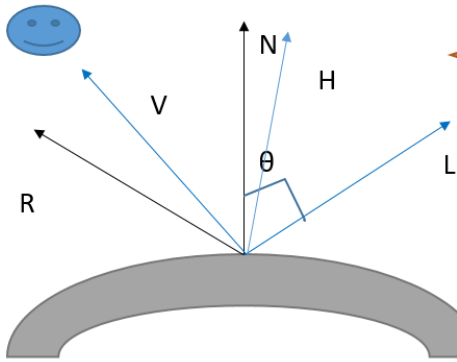
+ I(light) Ks(表面高光系数) ( V · R)$^{n}$高光指数 或者加上 I(light)·Ks cos (ns · α)
ns为镜面反射指数，越大越接近镜面

c(specular) = (c(light) m(specular)) max (0, V · R)$^{m(gloss)}$

$m(gloss)$ 材质的光泽度，反光度
m(specular) 材质的高光反射颜色
c(light) 光源的颜色和强度
Max()函数防止点积为负数，导致物体被背后的光源照亮正面

**扩展：Blinn-Phong光照模型**

N： 物体表面法线的单位矢量




点光源

L： 指向光源的单位矢量

0° ≤ Θ ≤ 90°

注意：引入H，H为V L的角平分线并归一

所以：H = V + L / |V + L|

Blinn-Phong光照模型：

I(spec) = I(Amibent) + I(light) Kd(0<Kd<1：对漫反射光的反射属性) (L · N)

+ I(light) Ks(表面高光系数) $(N \cdot H)^{n高光指数}$ ns为镜面反射指数，越大越接近镜面

c(specular) = (c(light) m(specular)) max (0, N · H)$^{m(gloss)}$ 使用N H的点积，计算速度更快

$m(gloss)$ 材质的光泽度，反光度

m(specular) 材质的高光反射颜色

c(light) 光源的颜色和强度

Max()函数防止点积为负数，导致物体被背后的光源照亮正面

具体实现见 CGInclude中的Lighting.cginc

## 表面着色器光照模型及自定义光照模型

### 半兰伯特：

```
1. Shader "Half_Lambert_Lighting"
2. {
3. //---------------------------------【属性】---------------------------------------
4. Properties
5. {
6.    _MainTex ("【主纹理】Texture", 2D) = "white" {}
7. }
8.
9. //---------------------------------【子着色器】---------------------------------
10. SubShader
11. {
12.    //-----------子着色器标签----------
13.    Tags { "RenderType" = "Opaque" }
14.    //------------------开始CG着色器编程语言段-----------------
15.    CGPROGRAM
16.
17.    //【1】光照模式声明：使用自制的半兰伯特光照模式
18.    #pragma surface surf QianMoHalfLambert
19.
20.    //【2】实现自定义的半兰伯特光照模式
21.    half4 LightingQianMoHalfLambert (SurfaceOutput s, half3 lightDir, half atten)
22.    {
23.       half NdotL =max(0, dot (s.Normal, lightDir));
24.
25.       //在兰伯特光照的基础上加上这句，增加光强
26.       float hLambert = NdotL * 0.5 + 0.5;
27.       half4 color;
```

```
        //修改这句中的相关参数
        color.rgb = s.Albedo * _LightColor0.rgb * (hLambert * atten * 2);
        color.a = s.Alpha;
        return color;
    }

    //【3】输入结构
    struct Input
    {
        float2 uv_MainTex;
    };

    //变量声明
    sampler2D _MainTex;

    //【4】表面着色函数的编写
    void surf (Input IN, inout SurfaceOutput o)
    {
        //从主纹理获取rgb颜色值
        o.Albedo = tex2D (_MainTex, IN.uv_MainTex).rgb;
    }

    //--------------------结束CG着色器编程语言段------------------
    ENDCG
}

Fallback "Diffuse"
}
```

**顶点&片元着色器实现光照模型**

**顶点着色器：逐顶点光照（计算量较少）**

**片元着色器：逐像素光照**

**单色镜面反射：（Phong光照），逐像素的光照**

```
Shader "浅墨Shader编程/Volume13/4.Specular"
{
    //-----------------------------------【属性值】-----------------------------------
    Properties
    {
        //主颜色
        _Color("Main Color", Color) = (1, 1, 1, 1)
        //镜面反射颜色
        _SpecColor("Specular Color", Color) = (1, 1, 1, 1)
        //镜面反射光泽度
        _SpecShininess("Specular Shininess", Range(1.0, 100.0)) = 10.0
    }

    //-----------------------------------【唯一的子着色器】-----------------------------------
    SubShader
    {
        //渲染类型设置：不透明
        Tags{ "RenderType" = "Opaque" }


        //-------------------------------唯一的通道-----------------------------
        Pass
```

```
    {
        //光照模型ForwardBase
        Tags{ "LightMode" = "ForwardBase" }
        //===========开启CG着色器语言编写模块===========
        CGPROGRAM

        //编译指令:告知编译器顶点和片段着色函数的名称
        #pragma vertex vert
        #pragma fragment frag

        //顶点着色器输入结构
        struct appdata
        {
            float4 vertex : POSITION;//顶点位置
            float3 normal : NORMAL;//法线向量坐标
        };

        //顶点着色器输出结构
        struct v2f
        {
            float4 pos : SV_POSITION;//像素位置
            float3 normal : NORMAL;//法线向量坐标
            float4 posWorld : TEXCOORD0;//在世界空间中的坐标位置
        };


        //变量的声明
        float4 _LightColor0;
        float4 _Color;
        float4 _SpecColor;
        float _SpecShininess;

        //-------------------------------【顶点着色函数】----------------------------
        // 输入：顶点输入结构体
        // 输出：顶点输出结构体
        //-------------------------------------------------------------------------------
        //顶点着色函数
        v2f vert(appdata IN)
        {
            //【1】声明一个输出结构对象
            v2f OUT;

            //【2】填充此输出结构
            //输出的顶点位置为模型视图投影矩阵乘以顶点位置，也就是将三维空间中的坐标投影到了二维窗口
            OUT.pos = mul(UNITY_MATRIX_MVP, IN.vertex);
            //获得顶点在世界空间中的位置坐标
            OUT.posWorld = mul(_Object2World, IN.vertex);
            //获取顶点在世界空间中的法线向量坐标
            OUT.normal = mul(float4(IN.normal, 0.0), _World2Object).xyz;

            //【3】返回此输出结构对象
            return OUT;
        }

        //-------------------------------【片段着色函数】----------------------------
        // 输入：顶点输出结构体
        // 输出：float4型的像素颜色值
        //-------------------------------------------------------------------------------
        fixed4 frag(v2f IN) : COLOR
        {
            //【1】先准备好需要的参数
            //获取法线的方向
```

```
85.             float3 normalDirection = normalize(IN.normal);
86.             //获取入射光线的方向
87.             float3 lightDirection = normalize(_WorldSpaceLightPos0.xyz);
88.             //获取视角方向
89.             float3 viewDirection = normalize(_WorldSpaceCameraPos - IN.posWorld.xyz);
90.
91.             //【2】计算出漫反射颜色值  Diffuse=LightColor * MainColor * max(0,dot(N,L))
92.             float3 diffuse = _LightColor0.rgb * _Color.rgb * max(0.0, dot(normalDirection, lightDirection));
93.
94.             //【3】计算镜面反射颜色值
95.             float3 specular;
96.             //若是法线方向和入射光方向大于180度，镜面反射值为0
97.             if (dot(normalDirection, lightDirection) < 0.0)
98.             {
99.                 specular = float3(0.0, 0.0, 0.0);
100.             }
101.             //否则，根据公式进行计算 Specular =LightColor * SpecColor *pow(max(0,dot(R,V)),Shiness),R=reflect(-L,N)  //
光源方向取反
102.             else
103.             {
104.                 float3 reflectDirection = reflect(-    lightDirection, normalDirection);
105.
                specular = _LightColor0.rgb * _SpecColor.rgb * pow(max(0.0, dot(reflectDirection, viewDirection)), _SpecShi
niness);
106.             }
107.
108.             //【4】合并漫反射、镜面反射、环境光的颜色值
109.
        float4 diffuseSpecularAmbient = float4(diffuse, 1.0) + float4(specular, 1.0) + UNITY_LIGHTMODEL_AMBIENT;
110.
111.             //【5】将漫反射-镜面反射-环境光的颜色值返回
112.             return diffuseSpecularAmbient;
113.         }
114.
115.         //============结束CG着色器语言编写模块============
116.         ENDCG
117.     }
118.   }
119. }
```