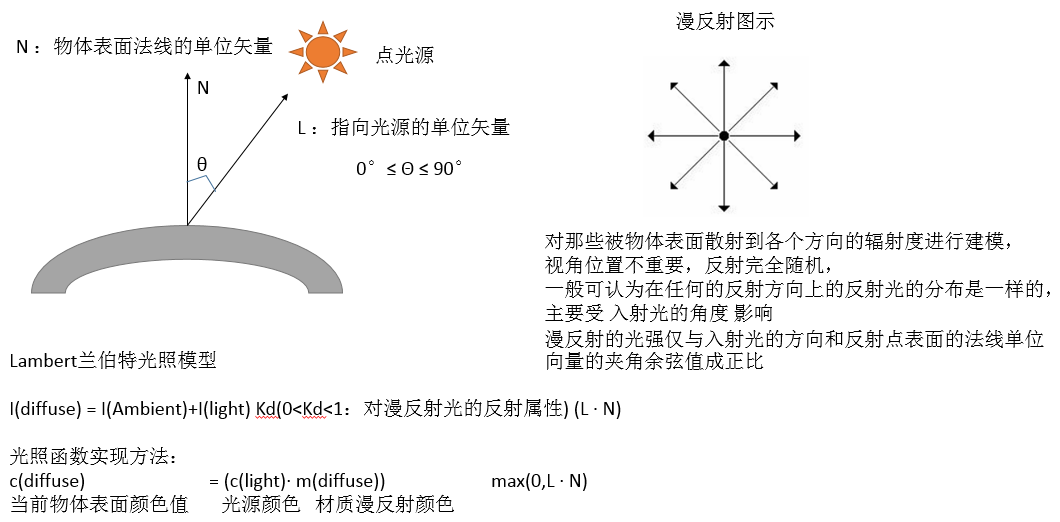
**标准光照模型**

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**



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;

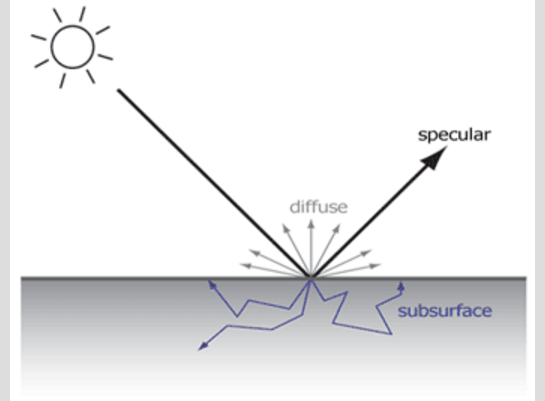
    c.rgb = s.Albedo \* \_LightColor0.rgb \* (diff \* atten \* 2);

    c.a = s.Alpha;

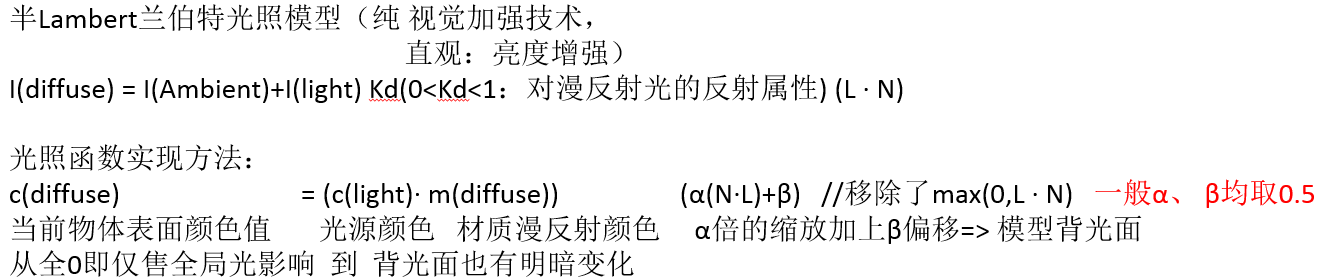
    **return** c;

}

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

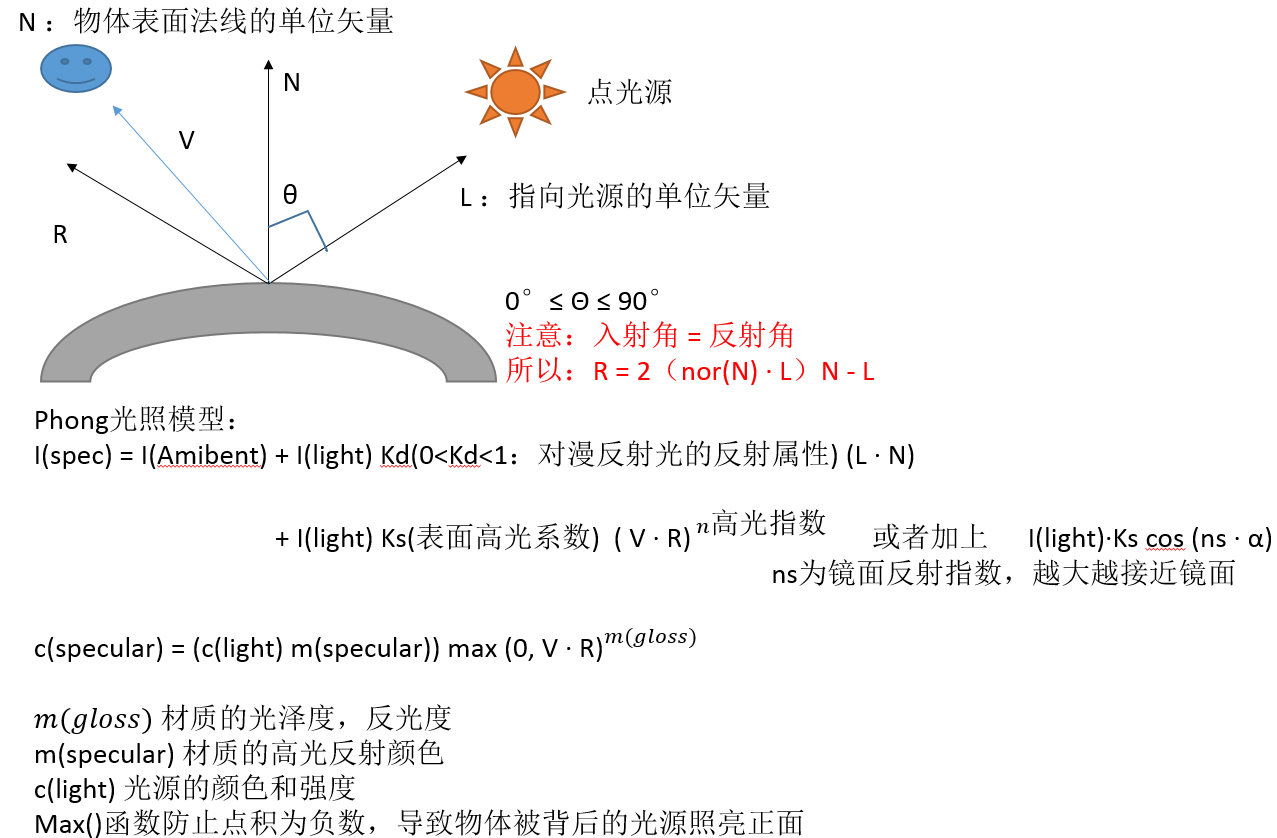




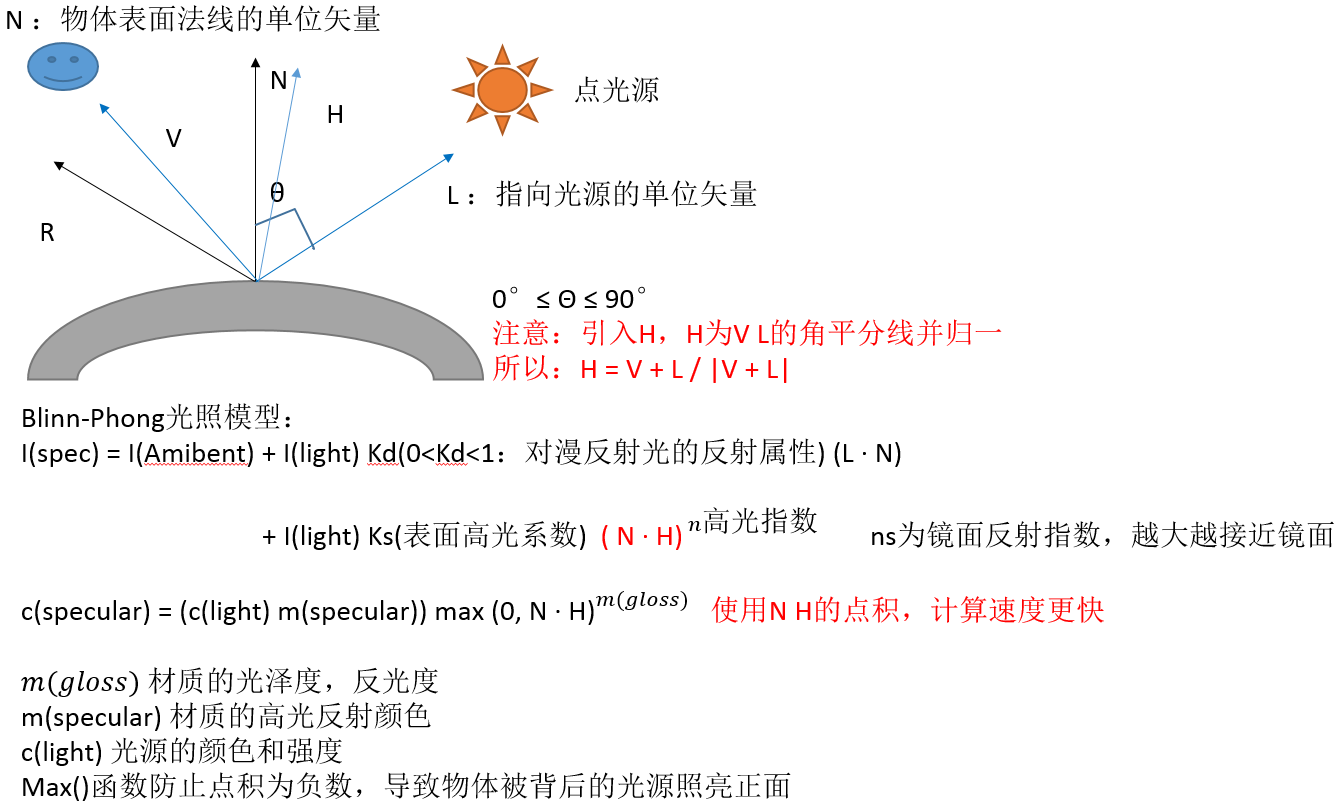


**· 镜面反射**

**Phong光照模型**



**扩展：Blinn-Phong光照模型**



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：**

Shader "Half\_Lambert\_Lighting"

{

//--------------------------------【属性】----------------------------------------

Properties

{

    \_MainTex ("【主纹理】Texture", 2D) = "white" {}

}

//--------------------------------【子着色器】----------------------------------

SubShader

{

    //-----------子着色器标签----------

    Tags { "RenderType" = "Opaque" }

    //-------------------开始CG着色器编程语言段-----------------

    CGPROGRAM

    //【1】光照模式声明：使用自制的半兰伯特光照模式

    #pragma surface surf QianMoHalfLambert

    //【2】实现自定义的半兰伯特光照模式

    half4 LightingQianMoHalfLambert (SurfaceOutput s, half3 lightDir, half atten)

    {

        half NdotL =max(0, dot (s.Normal, lightDir));

        //在兰伯特光照的基础上加上这句，增加光强

        **float** hLambert = NdotL \* 0.5 + 0.5;

        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

                float3 normalDirection = normalize(IN.normal);

                //获取入射光线的方向

                float3 lightDirection = normalize(\_WorldSpaceLightPos0.xyz);

                //获取视角方向

                float3 viewDirection = normalize(\_WorldSpaceCameraPos - IN.posWorld.xyz);

                //【2】计算出漫反射颜色值  Diffuse=LightColor \* MainColor \* max(0,dot(N,L))

                float3 diffuse = \_LightColor0.rgb \* \_Color.rgb \* max(0.0, dot(normalDirection, lightDirection));

                //【3】计算镜面反射颜色值

                float3 specular;

                //若是法线方向和入射光方向大于180度，镜面反射值为0

                **if** (dot(normalDirection, lightDirection) < 0.0)

                {

                    specular = float3(0.0, 0.0, 0.0);

                }

                //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**

                **else**

                {

                    float3 reflectDirection = reflect(-    lightDirection, normalDirection);

                        specular = \_LightColor0.rgb \* \_SpecColor.rgb \* pow(max(0.0, dot(reflectDirection, viewDirection)), \_SpecShininess);

                }

                //【4】合并漫反射、镜面反射、环境光的颜色值

                float4 diffuseSpecularAmbient = float4(diffuse, 1.0) + float4(specular, 1.0) + UNITY\_LIGHTMODEL\_AMBIENT;

                //【5】将漫反射-镜面反射-环境光的颜色值返回

                **return** diffuseSpecularAmbient;

            }

            //===========结束CG着色器语言编写模块===========

            ENDCG

        }

    }

}