Ue4 BRDF公式解析

more never

PBR 着色主要分两部分：漫反射和 高光计算

代码都在BRDF.ush 文件中

第一部分 漫反射

Lambert模型BRDF:

公式原型

$$f(l,v) = \frac{C_{diff}}{\pi}$$

```
float3 Diffuse_Lambert( float3 DiffuseColor )
{
        return DiffuseColor * (1 / PI);
}
```

Burley 2012, "Physically-Based Shading at Disney"

基于物理的渲染Shading at Disney

公式原型

$$f_d = \frac{baseColor}{\pi}(1 + (FD90 - 1)(1 - cos\theta_l)^5)(1 + \mathrm{FD90} - 1)(1 - cos\theta_v)^5)$$

$$\text{其中 FD90} = 0.5 + 2 * \text{roughness} * cos^2\,\theta_d$$

$$\mathrm{NoL} = cos\theta_l\,, NoV = cos\theta_v \text{ 可以认为就是参数}$$

```
float3 Diffuse_Burley( float3 DiffuseColor, float Roughness, float NoV, float NoL, flo
{
        float FD90 = 0.5 + 2 * VoH * VoH * Roughness;
        float FdV = 1 + (FD90 - 1) * Pow5( 1 - NoV );
        float FdL = 1 + (FD90 - 1) * Pow5( 1 - NoL );
        return DiffuseColor * ( (1 / PI) * FdV * FdL );
}
```

- **Gotanda 2012, "Beyond a Simple Physically Based Blinn-Phong Model in Real-Time"**

ppt下载

公式原型如下

$$L_r(\theta_r,\theta_i,\phi_r-\phi_i;\sigma)=\frac{\rho}{\pi}E_0\cos\theta_i(A+B\,\mathrm{Max}(0,\cos(\phi_r-\phi_i))\sin\alpha\tan\beta)$$

经过变化产生

$$L_r=\frac{\rho}{\pi}E_0(N\cdot L)\left((1.0-0.5\frac{\sigma^2}{\sigma^2+0.33})+(0.45\frac{\sigma^2}{\sigma^2+0.09})\mathrm{Max}(0,\frac{E-N(N\cdot E)}{\|E-N(N\cdot E)\|}\frac{L-N(N\cdot L)}{\|L-N(N\cdot L)\|})\frac{\sqrt{(1-\mathrm{Max}(N\cdot L,N\cdot E)^2)(1-\mathrm{Min}(N\cdot L,N\cdot E)^2)}}{\mathrm{Max}(N\cdot L,N\cdot E)}\right)$$

再次变化获得

$$L_r=\frac{\rho}{\pi}E_0\left[(N\cdot L)(1.0-0.5\frac{\sigma^2}{\sigma^2+0.33})+\left((0.45\frac{\sigma^2}{\sigma^2+0.09})\mathrm{Max}(0,E\cdot L-(N\cdot E)(N\cdot L))\mathrm{Min}(1,\frac{N\cdot L}{N\cdot E})\right)\right]$$

假设 NoV = N * E, NoL = N * L

VoL = E * L

```
float3 Diffuse_OrenNayar( float3 DiffuseColor, float Roughness, float NoV, float NoL,
{
        float a = Roughness * Roughness;
        float s = a;// / ( 1.29 + 0.5 * a );
        float s2 = s * s;
        float VoL = 2 * VoH * VoH - 1;          // double angle identity
        float Cosri = VoL - NoV * NoL;
        float C1 = 1 - 0.5 * s2 / (s2 + 0.33);
        float C2 = 0.45 * s2 / (s2 + 0.09) * Cosri * ( Cosri >= 0 ? rcp( max( NoL, NoV
        return DiffuseColor / PI * ( C1 + C2 ) * ( 1 + Roughness * 0.5 );
}
```

第二部分 高光反射

基于微表面理论的**Cook-Torrance模型**计算

Cook-Torrance模型BRDF:

$$f(l,v)=\frac{D(h)F(v,h)G(l,v)}{4(n*l)(n*v)}=\frac{D(h)F(v,h)G(l,v)}{4Cos\theta_i Cos\theta_O}$$

其实就是3个函数 D(h) 法线分布函数（NDF）

F(v,h) 菲尼尔方程 （Fresnel）

G(l,v) 几何衰减因子 (Geometrial Attenuation Factor)

1，法线分布函数(NDF)

UE4 采用**GGX / Trowbridge-Reitz模型**

$$D(h)=\frac{a^2}{\pi((n*h)^2(a^2-1)+1)^2}$$

```
float D_GGX( float Roughness, float NoH )
{
	float a = Roughness * Roughness;
	float a2 = a * a;
	float d = ( NoH * a2 - NoH ) * NoH + 1; // 2 mad
	return a2 / ( PI*d*d );                                    // 4 mul, 1 rcp
}
```

- **Blinn 1977, "Models of light reflection for computer synthesized pictures"**

$$D(h) = (dot(N * H))^{e}$$

在这基础之上再乘以 $\frac{e+2}{2\pi}$ 可以满足能量守恒，使入射到出射能量不损失。如果按照 UE4 的规则，取 $a = roughness^2$ 那么 $e = \frac{2}{a^2} - 2$

$$D(h) = \frac{e+2}{2\pi} * ((N * H))^{e}$$

`PhongShadingPow( NoH, n)` 就是 Noh n

```
float D_Blinn( float Roughness, float NoH )
{
	float a = Roughness * Roughness;
	float a2 = a * a;
	float n = 2 / a2 - 2;
	return (n+2) / (2*PI) * PhongShadingPow( NoH, n );             // 1 mad, 1 ex
}
```

- **Anisotropic GGX, Burley 2012, "Physically-Based Shading at Disney"**

基于物理的渲染Shading at Disney

$$D_{GTR}(\theta_h) = \frac{(\gamma-1)(\alpha^2-1)}{\pi(1-(\alpha^2)^{1-\gamma})}\frac{1}{(1+(\alpha^2-1)\cos^2\theta_h)^{\gamma}}$$

$$\phi_h = 2\pi\xi_1$$

$$\cos\theta_h = \sqrt{\frac{1-[(\alpha^2)^{1-\gamma}(1-\xi_2)+\xi_2]^{\frac{1}{1-\gamma}}}{1-\alpha^2}}$$

转化为

$$D_{GTRaniso} = \frac{1}{\pi}\frac{1}{a_x a_y}\frac{1}{((h-x)^2/a_x^2+(h*y)^2/a_y^2+(h*n)^2)^2}$$

```
        float ax = RoughnessX * RoughnessX;
        float ay = RoughnessY * RoughnessY;
        float XoH = dot( X, H );
        float YoH = dot( Y, H );
        float d = XoH*XoH / (ax*ax) + YoH*YoH / (ay*ay) + NoH*NoH;
        return 1 / ( PI * ax*ay * d*d );
}
```

2, 菲涅尔方程（Fresnel）

UE4采用了**Schlick**的近似值计算方法， 方程如下

$$F(v,h) = F_o + (1 - F_o)(1 - v*h)^5$$

公式转化为

$$F(v,h) = F_o + (1 - F_o)\,2^{(-5.55473(v*h)-6.98316)\,(v*h)}$$

取 50* specualrColor.g 作为近似 值

```
float3 F_Schlick( float3 SpecularColor, float VoH )
{
        float Fc = Pow5( 1 - VoH );                                     // 1 sub, 3 mu
        //return Fc + (1 - Fc) * SpecularColor;         // 1 add, 3 mad

        // Anything less than 2% is physically impossible and is instead considered to
        return saturate( 50.0 * SpecularColor.g ) * Fc + (1 - Fc) * SpecularColor;
}
```

标准实现

```
float3 F_Fresnel( float3 SpecularColor, float VoH )
{
        float3 SpecularColorSqrt = sqrt( clamp( float3(0, 0, 0), float3(0.99, 0.99, 0.
        float3 n = ( 1 + SpecularColorSqrt ) / ( 1 - SpecularColorSqrt );
        float3 g = sqrt( n*n + VoH*VoH - 1 );
        return 0.5 * Square( (g - VoH) / (g + VoH) ) * ( 1 + Square( ((g+VoH)*VoH - 1)
}
```

3, 几何衰减因子（Geometrical Attenuation Factor）

UE4采用**Schlick模型**的计算该项，方程如下

$$\alpha = \left(\frac{roughness + 1}{2}\right)^2, k = \alpha/2$$

$$G(v) = \frac{n * v}{(n * v)(1 - k) + k}$$

$$G(l,v) = G(l)\,G(v) \quad \text{对参数 l,v 套用 2 次公式}$$

```
        float k = Square( Roughness ) * 0.5;
        float Vis_SchlickV = NoV * (1 - k) + k;
        float Vis_SchlickL = NoL * (1 - k) + k;
        return 0.25 / ( Vis_SchlickV * Vis_SchlickL );
}
```

- **Kelemen 2001, "A microfacet based coupled specular-matte brdf model with importance sampling"**

pdf下载

```
float Vis_Kelemen( float VoH )
{
        // constant to prevent NaN
        return rcp( 4 * VoH * VoH + 1e-5);
}
```

参考文章

UE4中的基于物理的着色（一）

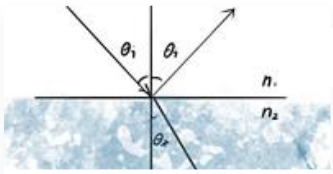
DragonSama：UE4中的基于物理的着色（二）

基于物理着色（二）- Microfacet材质和多层材质

编辑于 2018-04-23 10:59

虚幻 4（游戏引擎）

知乎 首发于 **UE4 学习笔记**



UE4基于物理的着色(二) 菲涅尔反射

前狗

UE4反射(F

引言 这篇文
中的成员函数
void Proces
Function, vo
函数，Proce
用到Thunk函

YaoJ1...