知乎 首发于 PBR

学习PBR路程（四、Specular）


商贾豪
学海无涯苦作舟

关注他

4 人赞同了该文章

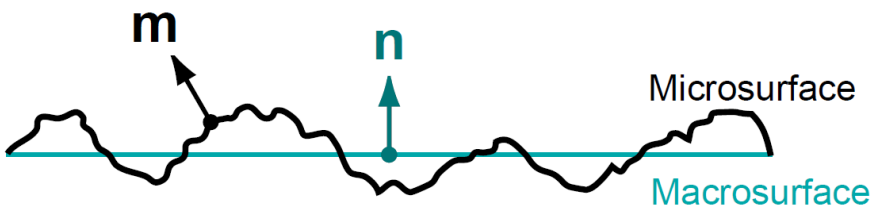
目前业界广泛采用的Microfacet Cook-Torrance BRDF形式

$$f(l,v) = \frac{D(h)F(v,h)G(l,v,h)}{4(n\cdot l)(n\cdot v)}$$

D项

微平面模型的一个重要特性是微平面法线**m**的统计分布（statistical distribution）。 此分布由曲面的法线分布函数（Normal Distribution Function，NDF）定义

法线分布函数（Normal Distribution Function，NDF）在一些文献中也用Specular D进行表示。



1、Blinn-Phong分布

$$D_p(\mathbf{m}) = \frac{\alpha_p + 2}{2\pi}(\mathbf{n}\cdot\mathbf{m})^{\alpha_p}$$

幂αp是Blinn-Phong NDF的“粗糙度参数”;高值表示光滑表面，低值表示粗糙表面。对于非常光滑的曲面，值可以任意高（一个完美的镜面αp=∞），并且通过将αp设置为0可以实现最大随机曲面（均匀NDF）

αp参数不便于艺术家操纵或直接绘制，因为它带来的视觉变化非常不均匀。出于这个原因，经常让美术师们操纵“界面值”，即通过非线性函数从中导出αp。例如：**αp=ms**，其中s是0到1之间的艺术家操纵值，**m**是给定的电影或游戏中αp的上限。这种映射被多款游戏使用，包括《使命召唤：黑色行动（Call of Duty: Black Ops）》，其中**m**被设置为值8192

UE4中，则采用映射 $\alpha_p = 2\alpha^{-2} - 2$, 那么得到的Blinn-Phong的形式为

$$D_p(\mathbf{m}) = \frac{1}{\pi\alpha^2}(\mathbf{n}\cdot\mathbf{m})^{(\frac{2}{\alpha^2}-2)}$$

```
float D_Blinn( float Roughness, float NdotH )
{
        float a2 = Roughness * Roughness;
    float n = 2 / a2 - 2;
    return (n+2) / (2*3.14159) * pow( NdotH, n );
}
```

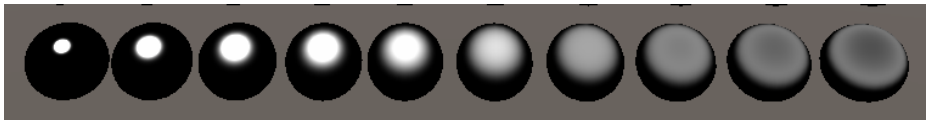
知乎 首发于 PBR

2 Beckmann分布

$$D_b(\mathbf{m}) = \frac{1}{\pi\alpha^2(\mathbf{n}\cdot\mathbf{m})^4} exp^{\left(\frac{(\mathbf{n}\cdot\mathbf{m})^2-1}{a^2(\mathbf{n}\cdot\mathbf{m})^2}\right)}$$

Beckmann分布在某些方面与Phong分布非常相似。 两种法线分布的参数可以使用关系式 $\alpha_p = 2\alpha_b^{-2} - 2$ 进行等效

```
float D_Beckmann( float Roughness, float NdotH )
                        {
                                float a2 = Roughness * Roughness;
                            float NoH2 = NdotH * NdotH;
                            return exp( (NoH2 - 1) / (a2 * NoH2) ) / ( 3.14159 * a2 *
                        }
```

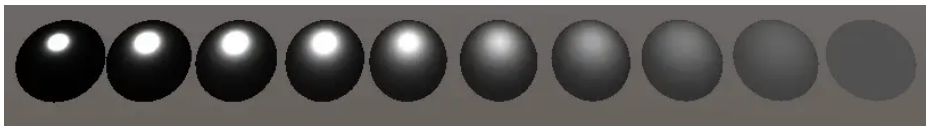


3 GGX（Trowbridge-Reitz）分布

$$D_{GGX}(\mathbf{m}) = \frac{\alpha^2}{\pi((\mathbf{n}\cdot\mathbf{m})^2(\alpha^2-1)+1)^2}$$

GGX拥有最长的尾部

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

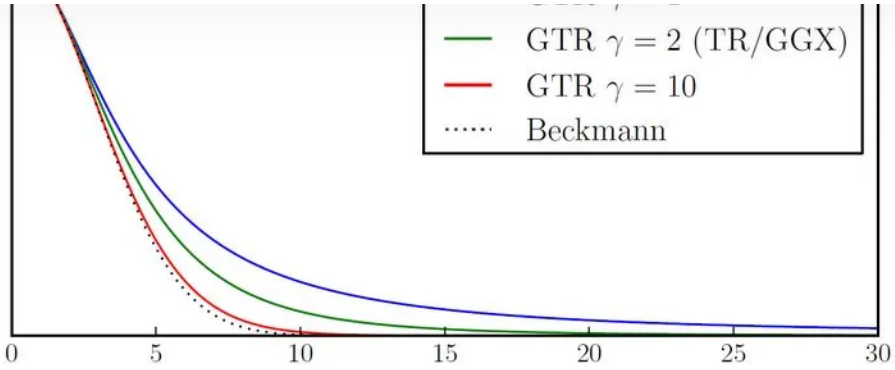


4 Generalized-Trowbridge-Reitz（GTR）分布

$$D_{GTR}(\mathbf{m}) = \frac{c}{(1+(\mathbf{n}\cdot\mathbf{m})^2(\alpha^2-1))^\gamma}$$

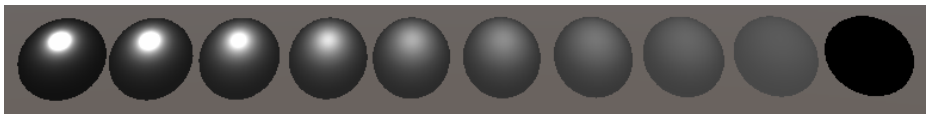
- 其中，γ参数用于控制尾部形状。 当γ= 2时，GTR等同于GGX。 随着γ的值减小，分布的尾部变得更长。而随着γ值的增加，分布的尾部变得更短。上式中：
- γ=1时，GTR即Berry分布
- γ=2时，GTR即GGX（Trowbridge-Reitz）分布

知乎 首发于 PBR

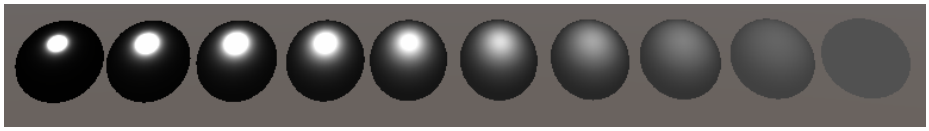



```
float D_GTR1(float alpha, float dotNH)
                        {
                            float a2 = alpha * alpha;
                            float cos2th = dotNH * dotNH;
                            float den = (1.0 + (a2 - 1.0) * cos2th);

                            return (a2 - 1.0) / (3.1415 * log(a2) * den);
                        }
 float D_GTR2(float Roughness, float NdotH,float y)
                        {
                            float a2 = Roughness * Roughness;
                            float cos2th = NdotH * NdotH;
                            float den = (1.0 + (a2 - 1.0) * cos2th);

                                return a2 / (3.1415 * pow(den,y));
                        }
```

y = 1:



y= 2:



D项各向异性

迪士尼原理着色模型中，各向同性粗糙度参数*r*与第二标量参数*k*aniso组合

$$k_{\text{aspect}} = \sqrt{1 - 0.9\,k_{\text{aniso}}},$$
$$\alpha_x = \frac{r^2}{k_{\text{aspect}}},$$
$$\alpha_y = r^2\,k_{\text{aspect}}.$$

Sony Imageworks 则使用了一种不同的参数化，允许任意程度的各向异性，我下面的ax和ay使用的是这种方式

知乎 首发于 PBR

$$\alpha_y = r^2\,(1 - k_{\text{aniso}}).$$

```
float ax = _Roughness * _Roughness * (1 + _Kaniso);
float ay = _Roughness * _Roughness * (1 - _Kaniso);
```

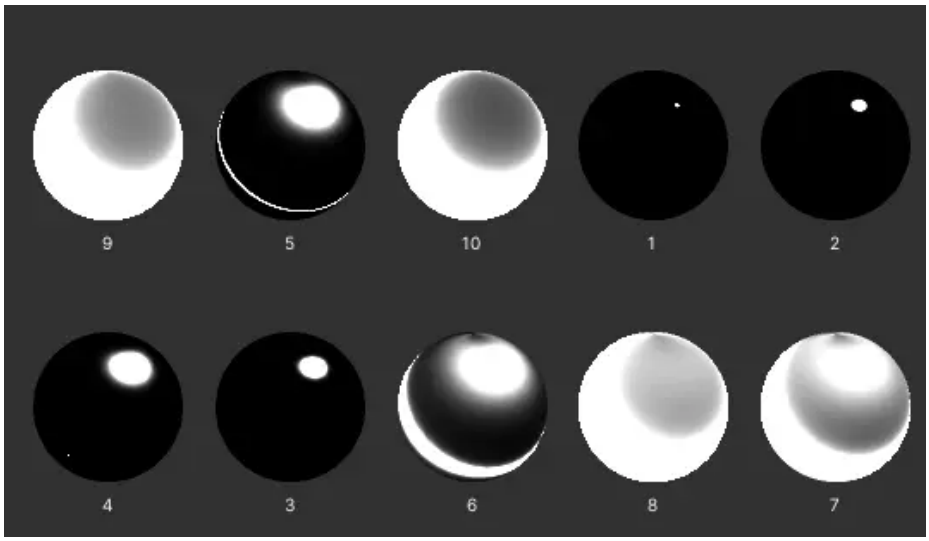
1 Anisotropic Beckmann Distribution

$$D_{Baniso}(\mathbf{m}) = \frac{1}{\pi\alpha_x\alpha_y(\mathbf{n}\cdot\mathbf{m})^4} exp(-\frac{\frac{(\mathbf{t}\cdot\mathbf{m})^2}{\alpha_x^2} + \frac{(\mathbf{b}\cdot\mathbf{m})^2}{\alpha_y^2}}{(\mathbf{n}\cdot\mathbf{m})^2})$$

m为微表面法线（可以理解为half半矢量），**n**为宏观表面法线，**t**为切线方向，**b**为副法线方向。

- 其中，参数**αx**和**αy**分别表示沿切线（tangent）方向**t**和副法线（binormal）方向**b**的粗糙度。若**αx = αy**，则上式缩减回各向同性形式。
- 需要注意的是，一般的shader写法，会将切线方向**t**写作X，副法线（binormal）**b**方向写作Y。

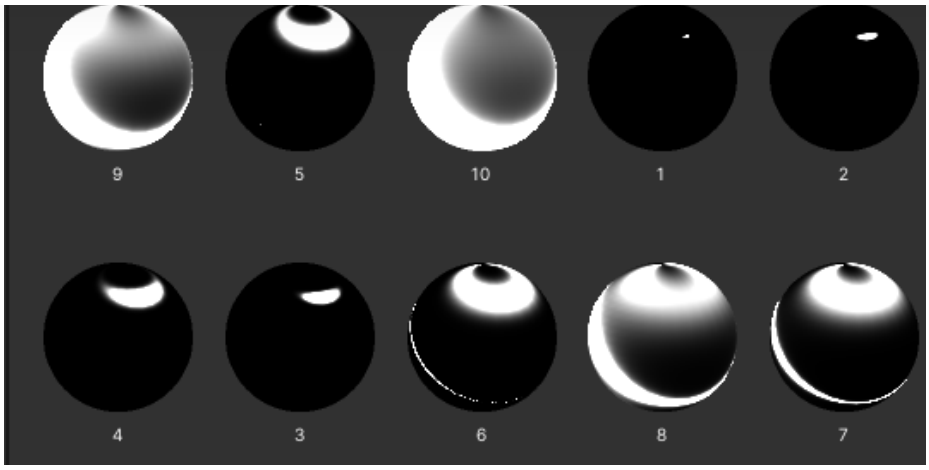
```
UE4
float D_Beckmann_aniso( float ax, float ay, float NoH, float3 H, float3 X, float3 Y )
{
    float XoH = dot( X, H );
    float YoH = dot( Y, H );
    float d = - (XoH*XoH / (ax*ax) + YoH*YoH / (ay*ay)) / NoH*NoH;
    return exp(d) / ( PI * ax*ay * NoH * NoH * NoH * NoH );
}

//世界空间
float D = D_Beckmann_aniso(ax,ay,NdotH,halfDir,i.tangent,i.binormal);
```



Kaniso = 0.1

Kaniso = 0.1

知乎 首发于 PBR



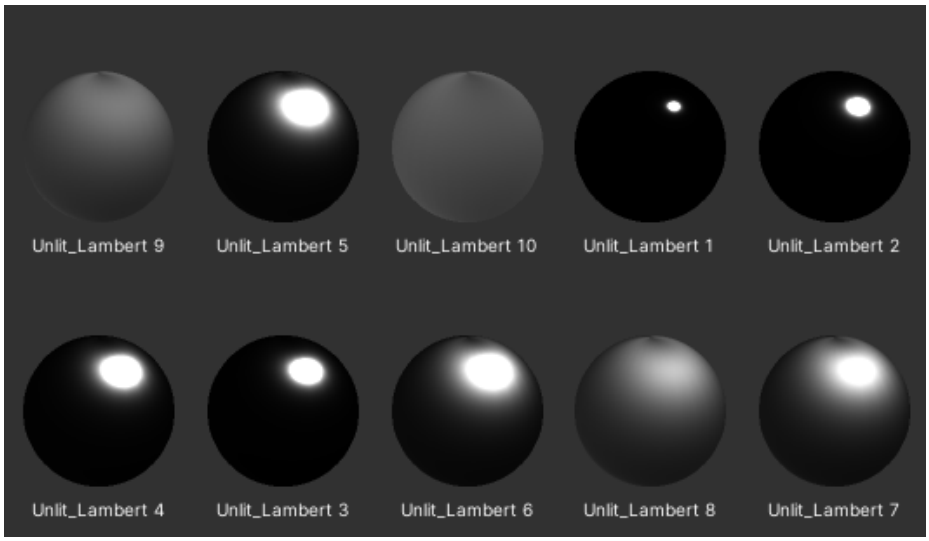
Kaniso = 0.5

粗糙度变大的时候会有奇怪的现象不知道是什么原因... 1 - 10 粗糙度[0.1 - 0.9]

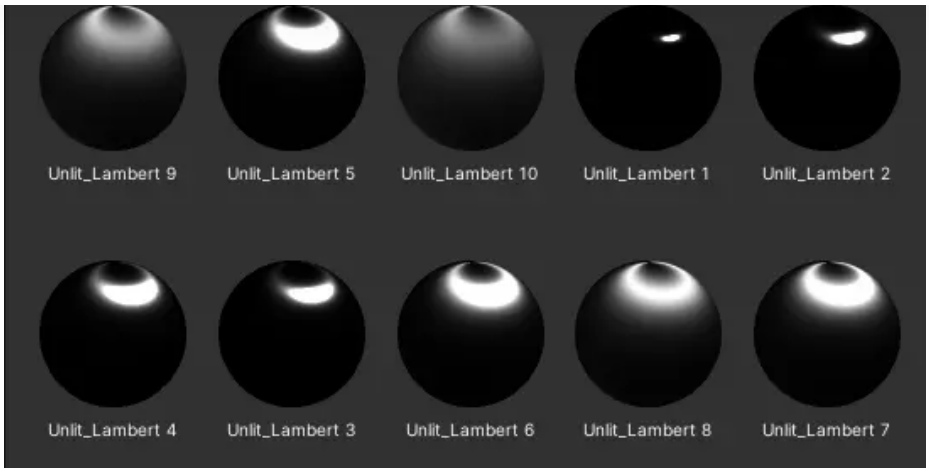
2 Anisotropic GGX Distribution

$$D_{GGXaniso}(\mathbf{m}) = \frac{1}{\pi\alpha_x\alpha_y}\frac{1}{\left(\frac{(\mathbf{t}\cdot\mathbf{m})^2}{\alpha_x^2}+\frac{(\mathbf{b}\cdot\mathbf{m})^2}{\alpha_y^2}+(\mathbf{n}\cdot\mathbf{m})^2\right)^2}$$

```
UE4
float D_GGXaniso( float ax, float ay, float NoH, float3 H, float3 X, float3 Y )
{
    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 );
}
float D = D_GGXaniso(ax,ay,NdotH,halfDir,i.tangent,i.binormal);
```



Kaniso = 0.1



Kaniso = 0.5

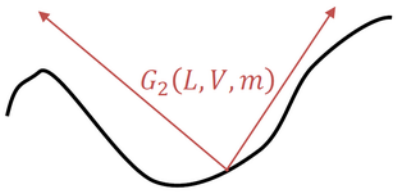
F项

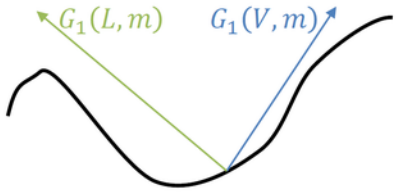
在这里我就不再实现了，知乎用户，这里我已经写过了，一般使用Schlick

G项

几何函数（Geometry Function）是保证Microfacet BRDF理论上能量守恒，逻辑上自洽的重要一环。其描述了微平面自阴影的属性，表示具有半矢量法线的微平面（microfacet）中，同时被入射方向和反射方向可见（没有被遮挡的）的比例，即未被遮挡的m = h微表面的百分比。

几何函数的两种主要形式：G1和G2。G1为微平面在单个方向（光照方向L或观察方向V）上可见比例。G2为微平面在光照方向L和观察方向V两个方向上可见比例-





Cook-Torrance GSF(1982)

Cook-Torrance GSF解决了三种几何衰减情况。第一种情况是光在没有干涉的情况下被反射，而第二种情况是一些反射光在反射后被阻挡，第三种情况是一些光在到达下一个microfacet之前被阻挡

Cook-Torrance [11]:

$$G_{Cook-Torranck}(\mathbf{l},\mathbf{v},\mathbf{h}) = \min\left(1, \frac{2(\mathbf{n}\cdot\mathbf{h})(\mathbf{n}\cdot\mathbf{v})}{\mathbf{v}\cdot\mathbf{h}}, \frac{2(\mathbf{n}\cdot\mathbf{h})(\mathbf{n}\cdot\mathbf{l})}{\mathbf{v}\cdot\mathbf{h}}\right)$$

```
float CookTorrenceGeometricShadowingFunction (float NdotL, float NdotV,
                        float VdotH, float NdotH){
                                float Gs = min(1.0, min(2*NdotH*NdotV / VdotH,
                        2*NdotH*NdotL / VdotH));
                                return  (Gs);
                        }
```

Neumann GSF（1999）

Neumann-Neumann GSF是另一个适用于各向异性正态分布的GSF的例子。它根据更大的视图方向或光方向产生更明显的几何阴影。

知乎 首发于 PBR

$$G_{Neumann}(\mathbf{l},\mathbf{v},\mathbf{h}) = \frac{(\mathbf{n}\cdot\mathbf{l})(\mathbf{n}\cdot\mathbf{v})}{\max(\mathbf{n}\cdot\mathbf{l},\mathbf{n}\cdot\mathbf{v})}$$

```
float NeumannGeometricShadowingFunction (float NdotL, float NdotV){
                                float Gs = (NdotL*NdotV)/max(NdotL, NdotV);
                                return  (Gs);
                        }
```

Kelemen GSF（2001）

几何阴影的比例不是恒定的，而是随着视角的变化而变化

$$G_{Kelemen}(\mathbf{l},\mathbf{v},\mathbf{h}) = \frac{(\mathbf{n}\cdot\mathbf{l})(\mathbf{n}\cdot\mathbf{v})}{(\mathbf{v}\cdot\mathbf{h})^2}$$

```
float KelemenGeometricShadowingFunction (float NdotL, float NdotV,
                        float LdotV, float VdotH){
                                float Gs = (NdotL*NdotV)/(VdotH * VdotH);
                                return   (Gs);
                        }
```

Implicit（2013）

通过将法线与光线的点积乘以法线与视点的点积来计算自阴影

$$G_{Implicit}(\mathbf{l},\mathbf{v},\mathbf{h}) = (\mathbf{n}\cdot\mathbf{l})(\mathbf{n}\cdot\mathbf{v})$$

```
float ImplicitGeometricShadowingFunction (float NdotL, float NdotV){
        float Gs =  (NdotL*NdotV);
        return Gs;
}
```

分离遮蔽阴影类型

Smith GGX导出的G项

UE4

$$\begin{aligned}
a &= roughness^2 \\
\Lambda(\mathbf{v}) &= (\mathbf{n}\cdot\mathbf{l})((\mathbf{n}\cdot\mathbf{v})(1-a)+a) \\
\Lambda(\mathbf{l}) &= (\mathbf{n}\cdot\mathbf{v})((\mathbf{n}\cdot\mathbf{l})(1-a)+a) \\
G_2(\mathbf{l},\mathbf{v},\mathbf{h}) &= \frac{0.5}{\Lambda(\mathbf{v})+\Lambda(\mathbf{l})}
\end{aligned}$$

```
// Appoximation of joint Smith term for GGX
// [Heitz 2014, "Understanding the Masking-Shadowing Function in Microfacet-Based BRDF
float Vis_SmithJointApprox( float a2, float NoV, float NoL )
```

知乎 首发于 PBR

```
    float Vis_SmithV = NoL * ( NoV * ( 1 - a ) + a );
    float Vis_SmithL = NoV * ( NoL * ( 1 - a ) + a );
    return 0.5 * rcp( Vis_SmithV + Vis_SmithL );
}
```

Google Filament

$$a = roughness$$
$$\Lambda(\mathbf{l}) = (\mathbf{n}\cdot\mathbf{v})\sqrt{(-(\mathbf{n}\cdot\mathbf{l})a^2 + (\mathbf{n}\cdot\mathbf{l}))(\mathbf{n}\cdot\mathbf{l}) + a^2}$$
$$\Lambda(\mathbf{v}) = (\mathbf{n}\cdot\mathbf{l})\sqrt{(-(\mathbf{n}\cdot\mathbf{v})a^2 + (\mathbf{n}\cdot\mathbf{v}))(\mathbf{n}\cdot\mathbf{v}) + a^2}$$
$$G_2(\mathbf{l},\mathbf{v},\mathbf{h}) = \frac{0.5}{\Lambda(\mathbf{v}) + \Lambda(\mathbf{l})}$$

```
float V_SmithGGXCorrelated(float NoV, float NoL, float a)
{
    float a2 = a * a;
    float GGXL = NoV * sqrt((-NoL * a2 + NoL) * NoL + a2);
    float GGXV = NoL * sqrt((-NoV * a2 + NoV) * NoV + a2);
    return 0.5 / (GGXV + GGXL);
}
```

Smith-Beckman GSF

最初是为了与Beckman NDF一起使用而创建的，Walter等人推测它是一个适合与Phong NDF一起使用的GSF

```
float BeckmanGeometricShadowingFunction (float NdotL, float NdotV, float roughness){
                            float roughnessSqr = roughness*roughness;
                            float NdotLSqr = NdotL*NdotL;
                            float NdotVSqr = NdotV*NdotV;


                            float calulationL = (NdotL)/(roughnessSqr * sqrt(1- NdotLS
                            float calulationV = (NdotV)/(roughnessSqr * sqrt(1- NdotVS


                            float SmithL = calulationL < 1.6 ? (((3.535 * calulationL)
                         + (2.181 * calulationL * calulationL))/(1 + (2.276 * calulati
                        (2.577 * calulationL * calulationL))) : 1.0;
                            float SmithV = calulationV < 1.6 ? (((3.535 * calulationV)
                        + (2.181 * calulationV * calulationV))/(1 + (2.276 * calulatio
                         (2.577 * calulationV * calulationV))) : 1.0;


                                float Gs =  (SmithL * SmithV);
                                return Gs;
                        }
```

Schlick-Beckman GSF

这是贝克曼函数的Schlick近似。它的工作原理是将粗糙度乘以2/PI的平方根，而不是计算，我们只是预先计算为0.797884

知乎 首发于 PBR

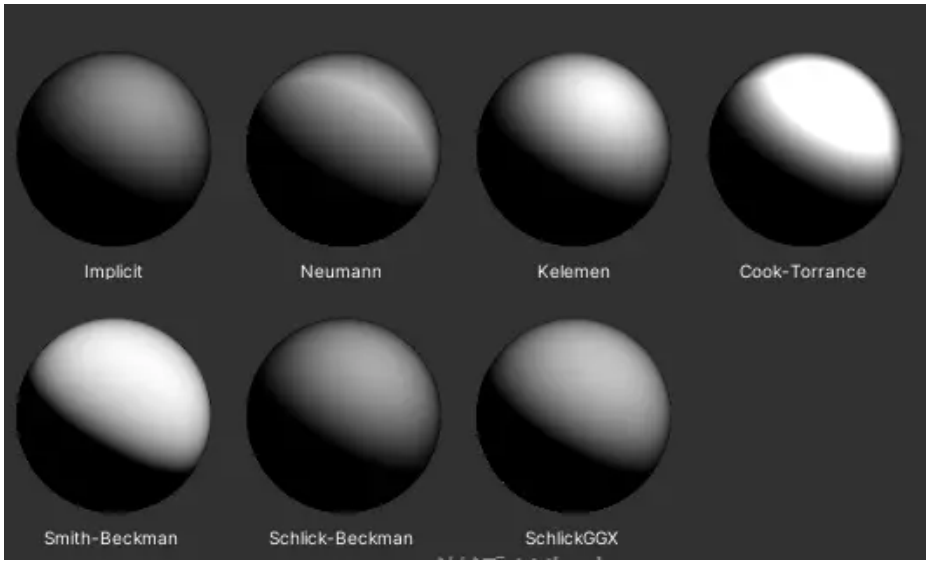
$$G_{Schlick}(\mathbf{v}) = \frac{\mathbf{n}\cdot\mathbf{v}}{(\mathbf{n}\cdot\mathbf{v})(1-k)+k}$$

```
float SchlickBeckmanGeometricShadowingFunction (float NdotL, float NdotV,
                        float roughness){
                           float roughnessSqr = roughness*roughness;
                           float k = roughnessSqr * 0.797884560802865;


                           float SmithL = (NdotL)/ (NdotL * (1- k) + k);
                           float SmithV = (NdotV)/ (NdotV * (1- k) + k);


                               float Gs =  (SmithL * SmithV);
                               return Gs;
                       }
```

Schlick-GGX GSF

GGX的Schlick近似简单地将粗糙度值除以2

```
float SchlickGGXGeometricShadowingFunction (float NdotL, float NdotV, float roughness)
    float k = roughness / 2;


    float SmithL = (NdotL)/ (NdotL * (1- k) + k);
    float SmithV = (NdotV)/ (NdotV * (1- k) + k);


        float Gs =  (SmithL * SmithV);
        return Gs;
}
```



_Roughness = 1

Mark:

Unity BRDF公式解析 - BaoqingWu - 博客园

Physically Based Rendering Algorithms In Unity - Jordan Stevens

编辑于 2022-01-12 16:25

知乎 首发于 PBR

已赞同 5 ▼ 添加评论 分享 喜欢 收藏 申请转载 ···

发布一条带图评论吧

还没有评论，发表第一个评论吧

文章被以下专栏收录

PBR

推荐阅读

学习PBR路程（三、Diffuse）

1.Lambert: 漫反射光的光强仅与入射光的方向和反射点处表面法向夹角的余弦成正比兰伯特定律的表面从所有的观察方向看来都是一样明亮的(ps:由约翰·海因里希·兰伯特在1760年提出的) float l...

商贾豪

由浅入深学

此篇由博主最
客园中，现
对MarkDow
击博客园链
本文动机 PB
Based Rend

向往