UE4中的真实渲染

[译]Real Shading in Unreal Engine 4（UE4中的真实渲染)(1) - 知乎 (zhihu.com)

[译]Real Shading in Unreal Engine 4（UE4中的真实渲染)(1) - 知乎 (zhihu.com)

一、介绍-Introduction

大约在一年前（2012年），我们决定投入一些时间去提升我们的着色模型并且包含一个更加基于物理的材质工作流。它部分驱动自渲染更真实的图像的需求，但是我们也对我们通过更基于物理的方法进行材质创建、使用分层材质可能达到的目标感兴趣。艺术家感觉这可能是对工作流和质量的一个巨大的提升，并且我已经在另一个工作室第一时间看到了这些成效，在那里我们已经过渡到离线混合的材质层。我们的其中一位技术美术在Epic在着色器中做分层的实验，实现了结果很有希望，这成为了一个额外的需求。

为了支持这个方向，我们知道材质的分层需要简单，并且高效。迪士尼的演讲来的时间十分完美[2]，涉及到了他们的基于物理的着色和使用Wreck-It Ralph的材质模型。Brent Burley 展示了一个非常小的材质参数集合可以足够精致的表现离线特性的电影渲染。他童谣展示了一个相当实用的着色模型可以紧密的适用于大多数采样的材质。他们的工作成为了我们的一个灵感和基础，并且像他们的“规则”相似，我们也决定去定义一个我们自己的系统的目标。

实时性能-Real-Time Performance

- 首要的是，它需要在每次许多可见灯光的条件下高效地使用。

简化复杂度-Reduced Complexity

- 参数尽可能的少。大批的参数不仅会导致很难来做一个决定，需要反复试验和试错，或者相互关联的属性对于一个预期效果需要许多的值来改变。
- 我们需要能够使用基于图像的光照（IBL）和交互式地可分析光源，所以参数必须在多有的光照类型中表现一致。

直观的界面-Intuitive Interface

- 我们更倾向易于理解的值，而不是像折射率（index of refraction）这样的物理参数。

感知线性-Perceptually Linear

- 我们希望通过蒙版支持分层，但是我们只能承受逐像素一次着色的负担。这意味着混合参数着色必须尽可能的匹配着色结果的混合。

简单掌握-Easy to Master

- 我们想要避免需要电介质和导体的技术理解，同时最小化创建基本的貌似物理的材质所需要的努力。

健壮-Robust

- 错误的创建物理上不可信的材质很困难。
- 所有参数的结合的应该尽可能的健壮和可信。

具有表现力-Expressive

- 延迟渲染限制了我们可以使用的着色模型的数量，所以我们的基本着色模型需要足够描述覆盖现实世界中99%的材质。
- 为了能够混合所有可分层的材质，所以，他们之间需要共享相同的参数集。

灵活-Flexible

- 其他的项目或者被授权开发的项目，可能不是以照片级真实渲染为目标，所以也需要足够灵活来允许非真实感渲染。

二、着色模型-Shading Model

2.1、漫反射双向反射分布函数-Diffuse BRDF

我们评估了Burley的漫反射模型，但是只观察到与Lambertain模型相比轻微的差别（等式1），所以我们不能判断额外的开销的合理性和必要性。除此之外，任何更复杂的漫反射模型很难高效的使用基于图像或球面谐调的光照。因此，我们不在评估其他选择上投入精力。

$$f(\mathbf{l},\mathbf{v}) = \frac{\mathbf{c_{diff}}}{\pi} \tag{1}$$

这里Cdiff是材质的漫反射率（diffuse albedo）。

UE4.21的当前实现，请参考BRFD.usf文件。

C++

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

2.2、微表面镜面反射双向反射分布函数-Microfacet Specular BRDF

常规的Cook-Torrance[5,6]微表面镜面反射着色模型是：

$$f(\mathbf{l},\mathbf{v}) = \frac{D(\mathbf{h})\,F(\mathbf{v},\mathbf{h})\,G(\mathbf{l},\mathbf{v},\mathbf{h})}{4\,(\mathbf{n}\cdot\mathbf{l})\,(\mathbf{n}\cdot\mathbf{v})} \tag{2}$$

请查阅[9]来获取更多扩展信息。

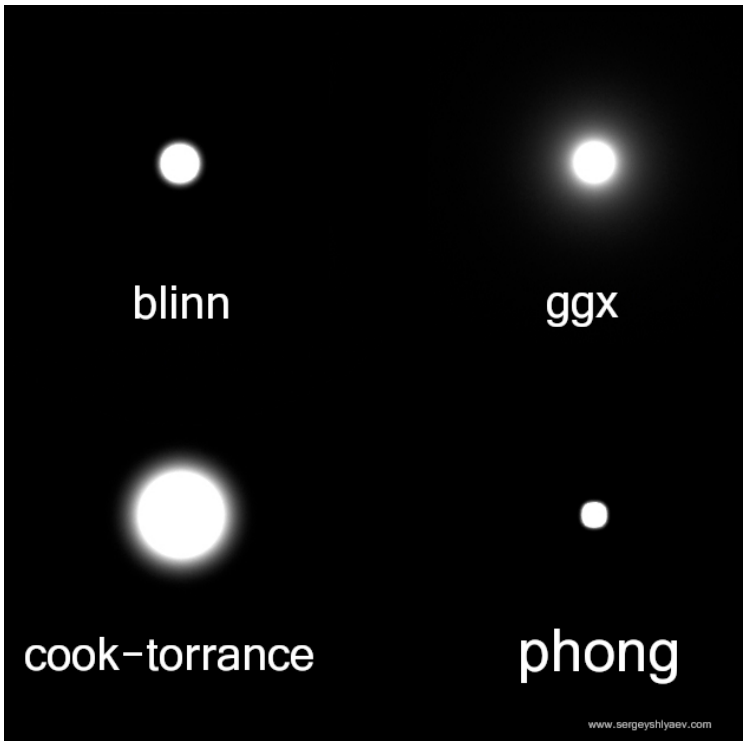
我们从迪士尼使用的模型开始，在与其他更高效的实现方式对比的情况下，评估了公式中每一项的重要性。这个工作远比听起来要困难的多；已经公布的公式中的每项输入不一定使用了相同的参数，但这对于正确的比较是至关重要的。

2.2.1、镜面反射D-Specular D

对于法线分布函数（NDF），我们发现迪士尼选择的GGX/Trowbridge-Reitz的成本是值得的。相比于Blinn-Phong的公式，额外的消耗相当的小，并且提供了更好的“长尾效果”，这个长尾效果的表现吸引了我们的艺术家。我们也采用了迪士尼重新定义的参数α=Roughness^2。

$$D(\mathbf{h}) = \frac{\alpha^2}{\pi\left((\mathbf{n}\cdot\mathbf{h})^2\left(\alpha^2-1\right)+1\right)^2} \tag{3}$$

> 译者注：**法线分布函数D(Normal Distribution Function)**：或者说镜面分布，从统计学上近似的表示了与向量h取向一致的微平面的比率。
>
> 举例来说，假设给定一个向量v，如果我们的微平面中有35%与向量v取向一致，则正态分布函数或者说NDF将会返回0.35。
>
> 我们来增加一个针对NDF在长尾效果方面的展示和对比，如下图所示。



注：主流NDF函数高光长尾效果的对比，GGX拥有最好的长尾表现。

> UE4.21中的代码实现：*从代码中看，跟D(h)的实现是完全一致的。*

C++

```
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
}
```

> 在UE4中，有另外一个关于各项异性的GGX实现，主要的实现思路是在X轴、Y轴两个方向上分别进行计算roughness。**Anisotropic GGX, Burley 2012, "Physically-Based Shading at Disney"**

C++

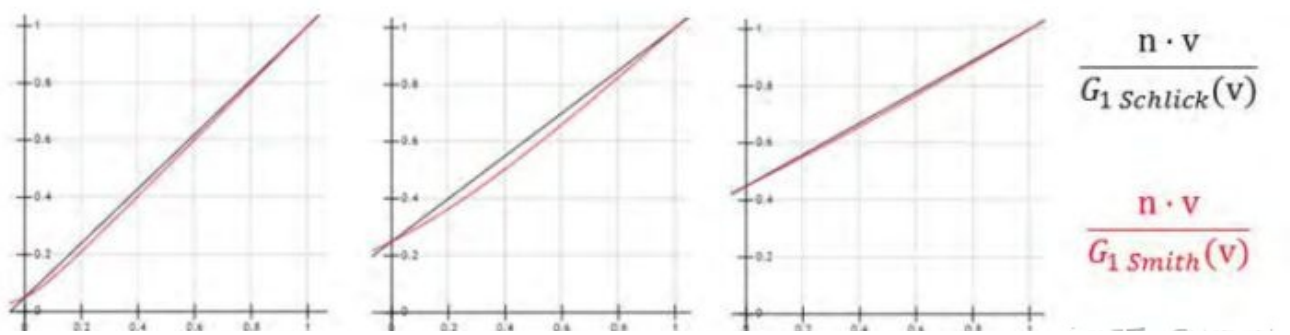
```
float D_GGXaniso( float RoughnessX, float RoughnessY, float NoH, float3 H, float3 X, float3 Y )
{
        float ax = RoughnessX * RoughnessX;
        float ay = RoughnessY * RoughnessY;
        float XoH = dot( X, H );
        float YoH = dot( Y, H );
        float d = XoH*XoH / (ax*ax) + YoH*YoH / (ay*ay) + NoH*NoH;
        return 1 / ( PI * ax*ay * d*d );
}
```

2.2.2、镜面反射G-Specular G

我们评估了镜面几何衰减项的多种选择。 到最后，我们选择使用Schlick模型[19]，但k =α/ 2，以便更好地拟合Smith模型GGX [21]。 通过这种修改，Schlick模型与α= 1的Smith完全匹配，并且相当[0，1]范围内的近似逼近（如图2所示）。 我们还选择使用迪士尼的修改通过在平方之前使用

$$\frac{Roughness + 1}{2}$$

重新映射粗糙度以减少“热度hotness”。 请务必注意该调整仅用于光源计算；如果应用于基于图像的照明（IBL），则在掠射角度部分的颜色太暗。



$\alpha = 0.1$ $\alpha = 0.5$ $\alpha = 0.9$

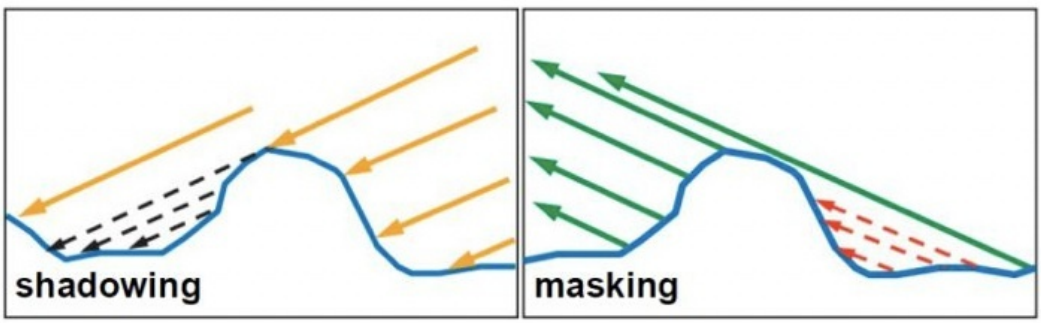
图2：使用k=α/2的Schlick非常接近Smith匹配

$$k = \frac{(Roughness + 1)^2}{8}$$

$$G_1(\mathbf{v}) = \frac{\mathbf{n} \cdot \mathbf{v}}{(\mathbf{n} \cdot \mathbf{v})(1 - k) + k}$$

$$G(\mathbf{l}, \mathbf{v}, \mathbf{h}) = G_1(\mathbf{l})\, G_1(\mathbf{v}) \tag{4}$$

> **注：**在基于物理的渲染中，几何函数（Geometry Function）是一个0到1之间的标量，描述了微平面自阴影的属性，表示了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（Geometry Function）即是对能顺利完成对光线的入射和出射交互的微平面概率进行建模的函数。



- **阴影（Shadowing）表示微平面对入射光的遮挡，一般为对光源方向L而言。**
- **遮蔽（masking）表示微平面对出射光的遮挡，一般为对观察方向V而言。**

注：几何函数G(x)的光学涵义

注：白色表示没有微平面的阴影，黑色表示微平面彻底被遮蔽

在UE4老的实现代码中，UE4将G(l,v)项和

$$4(n \circ l)(n \circ v)$$

合并为Vis(l, v)，Schlick模型的实现代码如下：

C++

```
float Vis_Schlick( float Roughness, float NoV, float NoL )
{
        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 );
}
```

但是，在UE4.21的代码中，实际的实现代码又发生了改变。如下所示。似乎是又朝着Smith的实现去进行了调整。

C++

```
// Tuned to match behavior of Vis_Smith
float Vis_Schlick( float a2, float NoV, float NoL )
{
        float k = sqrt(a2) * 0.5;
        float Vis_SchlickV = NoV * (1 - k) + k;
        float Vis_SchlickL = NoL * (1 - k) + k;
        return 0.25 / ( Vis_SchlickV * Vis_SchlickL );
}
```

2.2.3、镜面反射F-Specular F

对于菲涅尔，我们做出使用Schlick近似的经典选择 [19]，但是有一点修改，我们使用了球面高斯Spherical Gaussian近似[10]来代替power计算。这稍微提高了计算效率，并且差异微不可察，公式为：

$$F(\mathbf{v},\mathbf{h}) = F_0 + (1 - F_0)\, 2^{(-5.55473(\mathbf{v}\cdot\mathbf{h}) - 6.98316)(\mathbf{v}\cdot\mathbf{h})} \qquad (5)$$

$$F_{Schlick}(v,n) = F_0 + (1 - F_0)(1 - dot(v,n))^5$$

Schlick近似的经典公式

> 采用球面高斯的近似来代替Power的选择，主要是考虑在GPU上运算的效率来考虑。

Let's focus on the second part

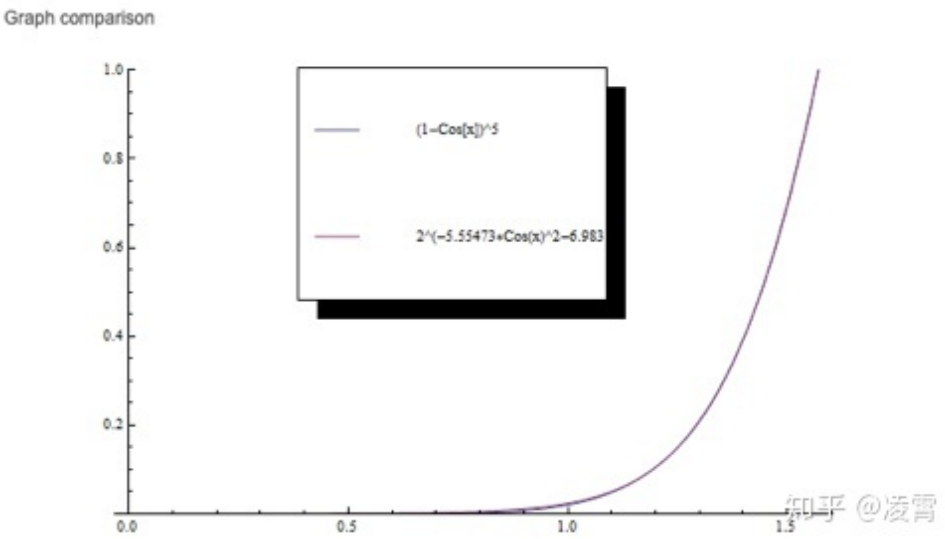
```
pow(1 - dotEH, 5)
```

Which is a SUB, LOG2, MUL, EXP2
We will apply the SG approximation to this expression with the added constant from the first section. In this case, we have a know the SpecularPower value in order to not add any additional instructions. We can also choose a constant x as the best fit solution for SpecularPower 5 which is x = 0.788 (see Mathematica file at end of the post for details of this result).

```
pow(1 - dotEH, 5) = exp2((5 + 0.788)/Log(2) * ((1-dotEH) - 1)) = exp2(-8.35 * dotEH)
```


Which is MUL, EXP2. We save 50% instructions.

注：GPU指令减少了50%



注：可以看到，这个拟合几乎是完美的。

> 但是，从UE4.21的代码来看，似乎并没有采用这个球面高斯趋近，而是采用了一种更高效的方式。按照实现应该是Fc + (1-Fc) * SpecularColor，但是，在第一个Fc前面增加了一个 50 * SpecularColor.g，但是，Fc还是采用了Pow5的处理。（哪位朋友了解原因，可以下面评论中帮忙解释一下。）

C++

```
float3 F_Schlick( float3 SpecularColor, float VoH )
{
        float Fc = Pow5( 1 - VoH );                                     // 1 sub, 3 mul
        // Anything less than 2% is physically impossible and is instead considered to be shadowing
        return saturate( 50.0 * SpecularColor.g ) * Fc + (1 - Fc) * SpecularColor;
}
```

> 译者注：下面的部分介绍的就是IBL了，但是，从BRDF的整体性角度来说，还缺失了一个环境光的漫反射部分，为了让读者有一个更完整的理解，我这里非常简单地来介绍一下这部分。
>
> 漫反射环境光照部分一般采用传统IBL中辉度环境映射（Irradiance Environment Mapping）技术，并不是基于物理的特有方案。

$$L_o(p,\omega_o) = k_d\frac{c}{\pi}\int_{\Omega} L_i(p,\omega_i)n\cdot\omega_i d\omega_i$$

环境光漫反射部分的计算公式

> 这个东西是没办法直接使用的，所以我们使用蒙特卡洛积分的期望法来进行积分。具体推导过程如下所示：

$$\begin{aligned}
\frac{1}{\pi}\int_{\Omega} L_i\left(p,\omega_i\right) n\cdot\omega_i \mathrm{d}\omega_i &= \frac{1}{\pi}\int_0^{2\pi}\int_0^{\frac{\pi}{2}} L_i\left(p,\omega_i\right) n\cos\theta\sin\theta \mathrm{d}\theta\mathrm{d}\phi \\
&\approx \frac{1}{\pi}\frac{1}{N_1 N_2}\sum_i^{N_1}\sum_j^{N_2}\frac{L_i\left(p,\phi_j,\theta_i\right)\cos(\theta_i)\sin(\theta_i)}{p(\theta_i,\phi_j)} \\
&= \frac{1}{\pi}\frac{1}{N_1 N_2}\sum_i^{N_1}\sum_j^{N_2}\frac{L_i\left(p,\phi_j,\theta_i\right)\cos(\theta_i)\sin(\theta_i)}{\frac{1}{2\pi*0.5\pi}} \\
&= \pi\frac{1}{N_1 N_2}\sum_i^{N_1}\sum_j^{N_2} L_i\left(p,\phi_j,\theta_i\right)\cos(\theta_i)\sin(\theta_i)
\end{aligned}$$

> 在写代码实现的时候，用蒙特卡洛积分求系数的过程大概就是一系列的相乘与求和，伪代码：

C++

```
void SH_Coefficients()
{
    double weight =4.0 * PI;
    //生成n条光线进行采样
    for(int i=0; i<n_samples; ++i)
    {
        //生成带抖动的无偏采样方向(θ,ϕ)
        for(int n=0; n<n_coeff; ++n)
        {
        //对于某一个light probe，它的每个球谐展开系数c_i就要累加起所有的【某方向上的irradiance * 这个方向上SH函数值】
        result[n] += light(θ,ϕ)* samples[i].SH_basis_coeff[n];
        }
    }
    // 把蒙特卡洛积分的常数项乘上去（恒定的采样权重，总采样数）
    double factor = weight / n_samples;
    for(i=0; i<n_coeff; ++i)
    {
        result[i] = result[i] * factor;
    }
}
```

> 上面的result[i]就是某个点上光照分布的球面函数经过”编码”之后得到的球谐系数。我们可以用离线预计算的系数，在runtime通过效率比较高的方式近似重构出原来的光照。

辉度环境映射（Irradiance Environment Map）

通过上面的伪代码，将左边的环境贴图，计算成辉度环境映射，从而用于运行时刻的运算

2.2.4、基于图像的光照-Image-Based Lighting

为了在基于图像的光照使用这个着色模型，需要解决辐射率积分，这通常使用重要性采样来完成。（注：关于辐射度这部分内容相当丰富和复杂，这里就不解释了。读者可以去查找辐射度的相关文

章。）下面的等式描述了这个数值积分：

$$\int_H L_i(\mathbf{l}) f(\mathbf{l}, \mathbf{v}) \cos\theta_{\mathbf{l}} d\mathbf{l} \approx \frac{1}{N} \sum_{k=1}^{N} \frac{L_i(\mathbf{l}_k) f(\mathbf{l}_k, \mathbf{v}) \cos\theta_{\mathbf{l}_k}}{p(\mathbf{l}_k, \mathbf{v})} \tag{6}$$

> **重要性采样**（Importance Sample）即通过现有的一些已知条件（分布函数），想办法集中于被积函数分布可能性较高的区域(重要的区域)进行采样，进而可高效地计算准确的估算结果的的一种策略。

下面的HLSL代码展示了如何在我们的着色模型中实现：

> （以下与原文略不一致，使用了4.21的代码替换）

C++

```
float4 ImportanceSampleGGX( float2 E, float Roughness )
{
    float m = Roughness * Roughness;
    float m2 = m * m;

    float Phi = 2 * PI * E.x;
    float CosTheta = sqrt( (1 - E.y) / ( 1 + (m2 - 1) * E.y ) );
    float SinTheta = sqrt( 1 - CosTheta * CosTheta );

    float3 H;
    H.x = SinTheta * cos( Phi );
    H.y = SinTheta * sin( Phi );
    H.z = CosTheta;

    float d = ( CosTheta * m2 - CosTheta ) * CosTheta + 1;
    float D = m2 / ( PI*d*d );
    float PDF = D * CosTheta;

    return float4( H, PDF );
}
```

C++

```
float4 ImportanceSampleGGX( float2 E, float Roughness )
{
    float m = Roughness * Roughness;
    float m2 = m * m;

    float Phi = 2 * PI * E.x;
    float CosTheta = sqrt( (1 - E.y) / ( 1 + (m2 - 1) * E.y ) );
    float SinTheta = sqrt( 1 - CosTheta * CosTheta );

    float3 H;
    H.x = SinTheta * cos( Phi );
    H.y = SinTheta * sin( Phi );
    H.z = CosTheta;

    float d = ( CosTheta * m2 - CosTheta ) * CosTheta + 1;
    float D = m2 / ( PI*d*d );
    float PDF = D * CosTheta;

    return float4( H, PDF );
}
```

即使有重要性采样，许多的样本仍然需要被采集。样本数量可以通过mip maps显著的减少，但是数量仍然需要大于16以满足足够的数量（注：原文1024次采样，4.20为32次采样）。因为我们为了获得局部的反射信息，需要在许多张环境贴图中进行逐像素的混合，我们实践中只能支持针对每张贴图进行一次样本采样。（注：原文在这里写得并不清楚，其实，想表达的意思是说：所以呢，咱们还得想其他办法。）

2.2.4.1、分解求和近似-Split Sum Approximation

为了实现这个，我们通过将上面的公式6分解成为两个求和部分来近似求解。每一个分离出来的求和公式可以被进行提前的预计算。对于一个常数

$$L_i(l)$$

这个近似是准确的，并且对于常规的环境来说，此公式也相当的精确。

$$\frac{1}{N}\sum_{k=1}^{N}\frac{L_i(\mathbf{l}_k)f(\mathbf{l}_k,\mathbf{v})\cos\theta_{\mathbf{l}_k}}{p(\mathbf{l}_k,\mathbf{v})} \approx \left(\frac{1}{N}\sum_{k=1}^{N}L_i(\mathbf{l}_k)\right)\left(\frac{1}{N}\sum_{k=1}^{N}\frac{f(\mathbf{l}_k,\mathbf{v})\cos\theta_{\mathbf{l}_k}}{p(\mathbf{l}_k,\mathbf{v})}\right) \tag{7}$$

2.2.4.2、预积分环境贴图-Pre-Filtered Environment Map

我们对于不同的粗糙度值，预计算第一个求和项并且将结果保存在CubeMap的mip-map层级中。这是在游戏工业使用的典型方式[1,9]。一个较小的区别是，我们使用了重要性采样和我们的着色模型

中的GGX分布对环境贴图做了卷积计算。

因为这是微表面模型，分布的形状改变依赖于到表面上的观察角度，所以**我们假设这个角度是0**，例如，n=v=r。

各向同性的假设是近似的第二个来源，并且它不幸地意味着我们不会在掠射角获得漫长的反射。

比起分离的和近似，实际上这是我们IBL解决方案中更大的错误来源。正如下面代码展示的，我们已经发现使用

$$cos\theta_{lk}$$

可以实现更佳的结果。

下面的代码，选自UE4.21版本。

C++

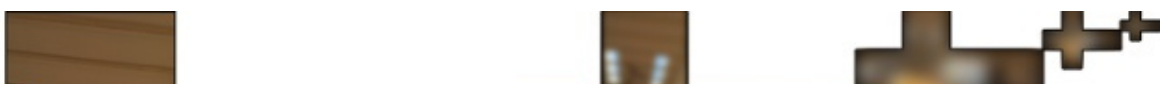
```
float3 PrefilterEnvMap( uint2 Random, float Roughness, float3 R )
{
        float3 FilteredColor = 0;
        float Weight = 0;

        const uint NumSamples = 64;
        for( uint i = 0; i < NumSamples; i++ )
        {
                float2 E = Hammersley( i, NumSamples, Random );
                float3 H = TangentToWorld( ImportanceSampleGGX( E, Pow4(Roughness) ).xyz, R );
                float3 L = 2 * dot( R, H ) * H - R;

                float NoL = saturate( dot( R, L ) );
                if( NoL > 0 )
                {
                        FilteredColor += AmbientCubemap.SampleLevel( AmbientCubemapSampler, L, 0 ).rgb * NoL;
                        Weight += NoL;
                }
        }

        return FilteredColor / max( Weight, 0.001 );
}
```

> 译者注：从代码上看，两个代码主要有两个变化，一个是在原来的代码中进行了1024次采样，而最新的代码中，只进行了64次采样；第二个变化，针对最后进行的除法进行了一个保护，使用max(Weight, 0.001)替代了原来的Weight。



预积分环境贴图

> 上图中的预积分环境贴图，是通过PrefilterEvnMap来进行离线计算并生成的。因为它是离线渲染的，所以，使用1024次采样，还是64次采样，基本上没什么太大的差距。至于为什么UE的最新代码，降低了大量的采样，我就不得而知了。
>
> 在这个输入的参数中，有一个Roughness的参数，这个就是为什么会使用5级的mipmap的原因。毕竟针对不同的roughness，会从不同的mipmap来进行采样，这样也就获得了不同的辐照度信息。
>
> 说白了，这还是一种在不能进行光线追踪情况下进行的一个模拟实现。将生成的Cubemap视为辐照度信息，将不同等级的MipMap作为不同Roughness的辐照贴图。这种方案既考虑到了实时渲染的效率问题，也在一定程度上保证了这个渲染的真实性。

2.2.4.3、环境双向反射分布函数-Environment BRDF

第二个求和项包含了其他所有的部分。这与用一个纯白色的环境对镜面反射双向反射分布函数进行积分操作是一样的，例如，

$$L_i(I_k) = 1$$

。通过Schlick的菲涅尔代替：F(v,h)=F0+(1-F0)(1-v·h)^5，我们发现F0可以因式分解到积分外。

$$\int_H f(\mathbf{l},\mathbf{v})\cos\theta_{\mathbf{l}}\mathrm{d}\mathbf{l} = F_0\int_H \frac{f(\mathbf{l},\mathbf{v})}{F(\mathbf{v},\mathbf{h})}\left(1-(1-\mathbf{v}\cdot\mathbf{h})^5\right)\cos\theta_{\mathbf{l}}\mathrm{d}\mathbf{l} + \int_H \frac{f(\mathbf{l},\mathbf{v})}{F(\mathbf{v},\mathbf{h})}(1-\mathbf{v}\cdot\mathbf{h})^5\cos\theta_{\mathbf{l}}\mathrm{d}\mathbf{l} \quad (8)$$

这余下了两个输入（粗糙度Roughness和cosθv），和两个输出（F0的缩放和偏移），所有的值都在[0,1]的范围内。我们预计算这个函数的结果，并且保存到一个2D的查找表中（因为精确度是非常重要的，所以，在这里使用R16G16的格式）。

> 上式留下了两个输入（Roughness 和 cos θv）和两个输出（缩放和向F0的偏差（a scale and bias to F0）），即把上述方程看成是F0 * Scale + Offset的形式。 我们预先计算此函数的结果并将其存储在2D查找纹理（LUT，look-up texture）中。

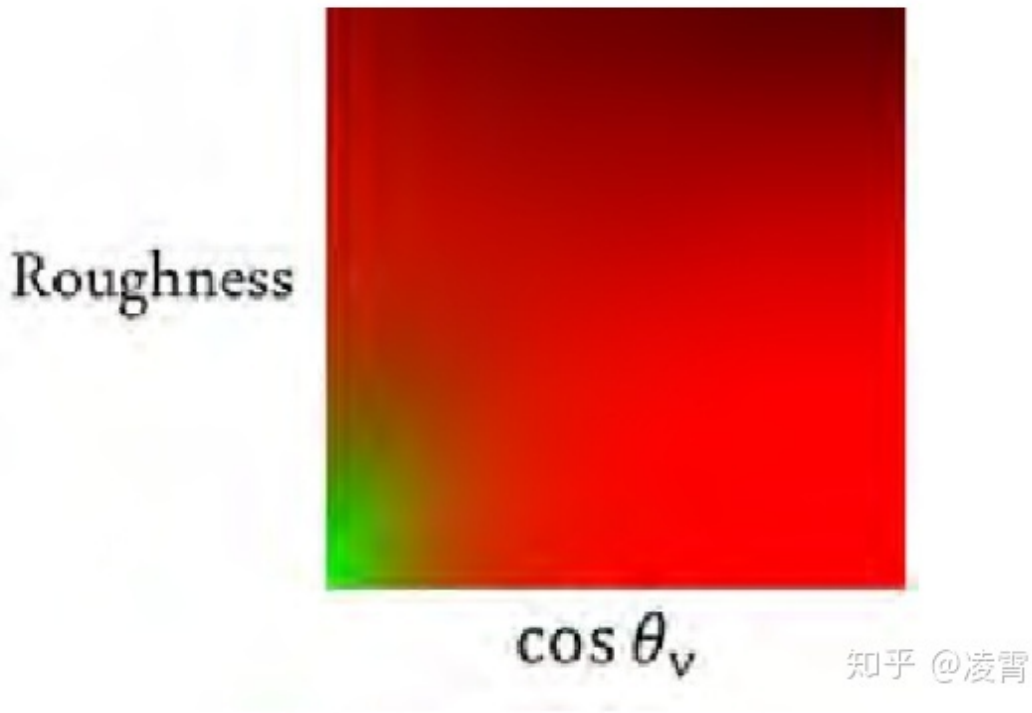


图3：2D纹理查找表

> 这张红绿色的贴图，输入roughness、cosθ，输出环境BRDF镜面反射的强度。这个是关于roughness、cosθ与环境BRDF镜面反射强度的固有映射关系。这个事可以进行离线预计算。
>
> 具体的取出方式为：

$$\frac{1}{N}\sum_{k=1}^{N}\frac{f(l_k,v)cos\theta_{l_k}}{p(l_k,v)} = \mathrm{LUT.r} * F_0 + \mathrm{LUT.g}$$

> 即UE4是通过把Fresnel公式的F0提出来，组成F0 * Scale +Offset的方式，再将Scale和Offset的索引存到一张2D LUT上。靠roughness和 NdotV进行查找。下面是从UE4.21代码中找到的制作这张LUT贴图的函数。请注意在代码中，针对A，B的两个变量的赋值。

C++

```
float3 IntegrateBRDF( uint2 Random, float Roughness, float NoV )
{
        float3 V;
        V.x = sqrt( 1.0f - NoV * NoV );        // sin
        V.y = 0;
        V.z = NoV;                                                  // cos

        float A = 0;
        float B = 0;
        float C = 0;

        const uint NumSamples = 64;
```

```
        for( uint i = 0; i < NumSamples; i++ )
        {
                float2 E = Hammersley( i, NumSamples, Random );

                {
                        float3 H = ImportanceSampleGGX( E, Pow4(Roughness)
).xyz;
                        float3 L = 2 * dot( V, H ) * H - V;

                        float NoL = saturate( L.z );
                        float NoH = saturate( H.z );
                        float VoH = saturate( dot( V, H ) );

                        if( NoL > 0 )
                        {
                                float a = Square( Roughness );
                                float a2 = a*a;
                                float Vis = Vis_SmithJointApprox( a2, NoV, NoL
);
                                float Vis_SmithV = NoL * sqrt( NoV * (NoV -
NoV * a2) + a2 );
                                float Vis_SmithL = NoV * sqrt( NoL * (NoL -
NoL * a2) + a2 );
                                //float Vis = 0.5 * rcp( Vis_SmithV +
Vis_SmithL );

                                // Incident light = NoL
                                // pdf = D * NoH / (4 * VoH)
                                // NoL * Vis / pdf
                                float NoL_Vis_PDF = NoL * Vis * (4 * VoH /
NoH);

                                float Fc = pow( 1 - VoH, 5 );
                                A += (1 - Fc) * NoL_Vis_PDF;
                                B += Fc * NoL_Vis_PDF;
                        }
                }

                {
                        float3 L = CosineSampleHemisphere( E ).xyz;
                        float3 H = normalize(V + L);

                        float NoL = saturate( L.z );
                        float NoH = saturate( H.z );
                        float VoH = saturate( dot( V, H ) );

                        float FD90 = ( 0.5 + 2 * VoH * VoH ) * Roughness;
```

```
                        float FdV = 1 + (FD90 - 1) * pow( 1 - NoV, 5 );
                        float FdL = 1 + (FD90 - 1) * pow( 1 - NoL, 5 );
                        C += FdV * FdL * ( 1 - 0.3333 * Roughness );
                }
        }

        return float3( A, B, C ) / NumSamples;
}
```

在完成这项工作之后，我们发现目前的和同时进行的研究，几乎都是和我们一致的结果。Whilst Gotanda使用了3D查找表[8]，Drobot优化它到2D的查找表[7]，就和我们所做的那样。另外，作为这个课题的一员——Lazarov又向前迈进了一步[11]，展示了相似积分的一对解析近似（这个方案中使用了不同的D和G函数）。

最后，为了近似重要性采样的引用，我们将这两个预计算的求和进行相乘的操作。

C++

```
float3 ApproximateSpecularIBL( uint2 Random, float3 SpecularColor, float Roughness, float3 N, float3 V )
{
        // Function replaced with prefiltered environment map sample
        float3 R = 2 * dot( V, N ) * N - V;
        float3 PrefilteredColor = PrefilterEnvMap( Random, Roughness, R );
        //float3 PrefilteredColor = FilterEnvMap( Random, Roughness, N, V );

        // Function replaced with 2D texture sample
        float NoV = saturate( dot( N, V ) );
        float2 AB = IntegrateBRDF( Random, Roughness, NoV ).xy;

        return PrefilteredColor * ( SpecularColor * AB.x + AB.y );
}
```

> 注：这个函数，只是说明了整个算法而已，在真正去进行计算的时候，是不会采用这个方法的，而是，选择直接进行针对预积分贴图采样的算法。如下面的代码所示。

C++

```
void MainPS(in noperspective float4 UVAndScreenPos : TEXCOORD0, float4 SvPosition : SV_POSITION, out float4 OutColor : SV_Target0)
{
        ......
        {
                float Mip = ComputeCubemapMipFromRoughness( GBuffer.Roughness, AmbientCubemapMipAdjust.w );
                float3 SampleColor = TextureCubeSampleLevel( AmbientCubemap, AmbientCubemapSampler, R, Mip ).rgb;
                SpecularContribution += SampleColor * EnvBRDF( GBuffer.SpecularColor, GBuffer.Roughness, NoV );
                //SpecularContribution += ApproximateSpecularIBL( Random, GBuffer.SpecularColor, GBuffer.Roughness, GBuffer.WorldNormal, -ScreenVector );
        }
}

half3 EnvBRDF( half3 SpecularColor, half Roughness, half NoV )
{
        // Importance sampled preintegrated G * F
        float2 AB = Texture2DSampleLevel( PreIntegratedGF, PreIntegratedGFSampler, float2( NoV, Roughness ), 0 ).rg;

        // Anything less than 2% is physically impossible and is instead considered to be shadowing
        float3 GF = SpecularColor * AB.x + saturate( 50.0 * SpecularColor.g ) * AB.y;
        return GF;
}
```

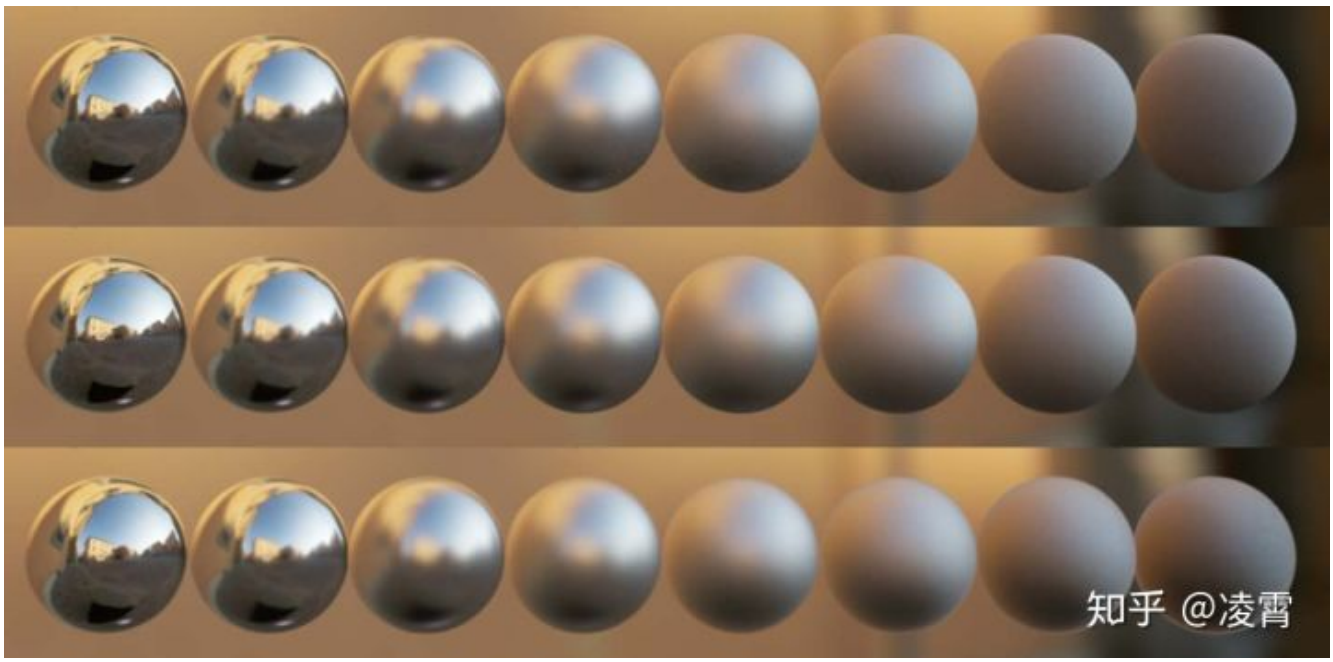


图4：最上方为参考（重要性采样方案），分解求和近似（Split sum approxmation）位于中间，包含n=v假设的Complete Approximation在最底部。径向对称假设引入了误差最大，但是组合近似依然和参考十分相似。

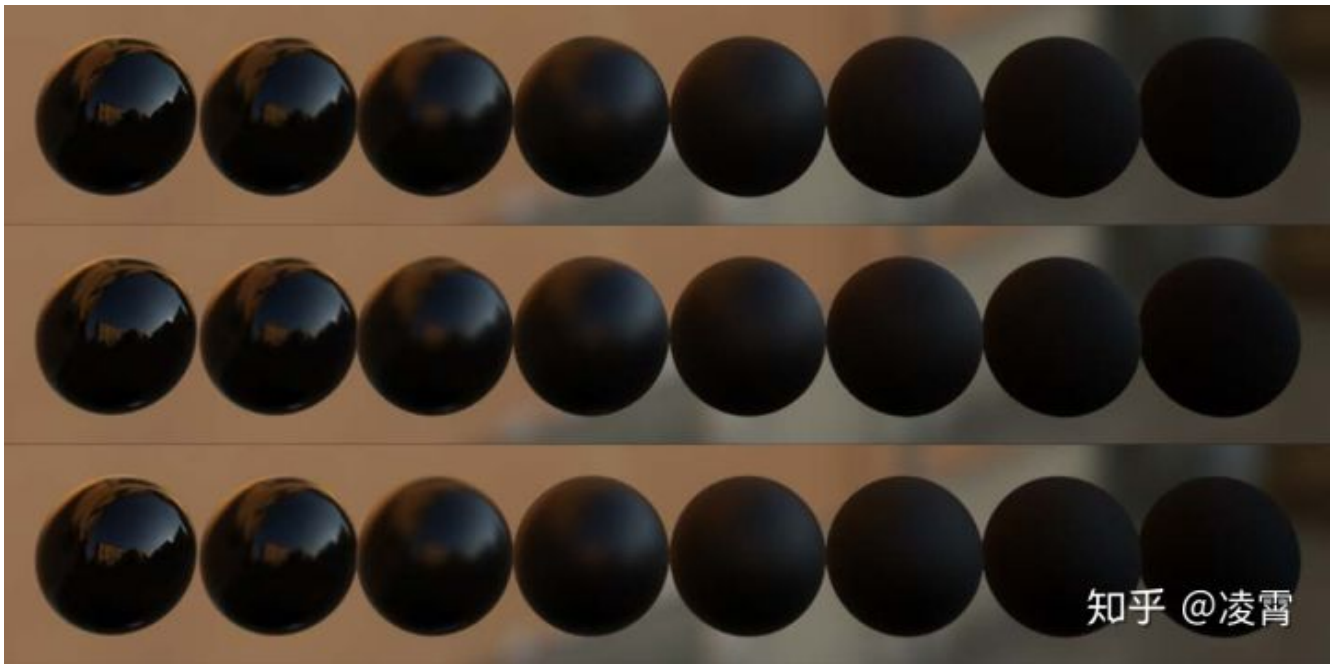


图5：电介质的对比图（对比项与图4相同）

2.3、材质模型-Material Mode

我们的材质模式是迪士尼的材质模型的简化版本，着眼于实时渲染的效率。限制参数的数量对于优化G-Buffer的空间十分重要，减少贴图的空间占用和存取，并且最小化在像素着色器中混合材质图形的成本。

下面是我们的基础材质模型：

- 基本颜色BaseColor 单一颜色。更容易理解的概念

- 金属度Metallic 不需要理解电介质和导体的反射率，更少的错误空间
- 粗糙度Roughness 它的概念非常清晰，相反的是光泽度gloss通常需要解释
- Cavity 用于表示小规模的遮蔽（这个似乎在最终的版本上，并没有出现）

基本颜色，金属度，粗糙度与迪士尼的模型相同，但是Cavity参数并没有在Disney的模型中被提及。所以我们来重点介绍一下Cavity。Cavity用于指定比运行时阴影系统能够处理的几何体更小的阴影，通常是由于几何体只存在于法线图中（**译者注**：这句话翻译的比较绕。其实应该是说，由于这些几何体的渲染很多都是依赖法线贴图来表现一些细节，但是，Cavity要求的会更精细。）。例如地板板之间的裂缝或衣服的缝隙。

最值得注意的遗漏是Specular参数。实际上，我们在完成Infiltrator演示之前一直在继续使用这个参数，但最终我们并不喜欢它。首先，我们觉得 "specular "是一个很糟糕的参数名称，它造成了很多混乱，而且在一定程度上不利于从艺术家控制镜面强度到控制粗糙度的过渡。艺术家和图形程序员都普遍忘记了它的范围，以为默认值是1，而实际默认值是Burley的0.5（对应4%的反射率）。Specular被有效使用的案例几乎都是为了小范围的阴影处理。我们发现可变折射率（IOR）对于非金属来说并不重要，所以我们最近用更容易理解的Cavity参数取代了Specular。非金属的F0现在是一个常数0.04。（**译者注**：引擎中依然使用镜面反射参数，镜面反射参数输入范围[0,1]，默认为0.5，F0的值为镜面反射参数*0.08。后面关于为什么没有取消，也有介绍。）

以下是迪斯尼模型中的参数，我们选择不在我们的基础Material Model中采用，而是作为特殊情况处理。（**译者注**：下面简单介绍了一下不采用的原因，有一些是开销，有一个干脆是不知道Disney的哥们是咋做的。呵呵。。。外国人也挺实在。）

- 次表面Subsurface 阴影贴图的采样不同
- 各向异性Anisotropy 需要更多的IBL采样
- 清漆ClearCoat 需要双倍的IBL采样
- 辉光Sheen 在Burley的笔记中没有很好的定义

除了Subsurface之外，我们没有在生产中使用过这些特殊情况下的模型，而Subsurface在我们的*Elemental*演示中被用于制作更好效果的冰。此外，我们还有一个专门针对皮肤的着色模型。在未来，我们正在考虑采用一种混合Deferred/Forwarding着色方法，以更好地支持更多的特殊着色模型。目前，我们采用的是纯粹的延迟着色方法，不同的着色模型是通过存储在G-Buffer中的着色模型ID的动态分支来处理的。（**译者注：**从UE现在的代码中看，它支持布料-Cloth、头发-Hair、眼睛-Eye、双面树叶-Two sided Foliage、预积分皮肤效果-Pre-Integrated Skin、次表面分布图-SubSurface Profile效果等。当然，从材质模型和提供各种不同的效果实现，这还不是一个概念，但是，利用UE选择的材质模型，至少能比较好地支撑很多效果的实现。）

2.3.1 Experiences 体验

有一种情况，我现在已经经历了好几次了。我会告诉刚开始过渡到使用不同的*Roughness*的艺术家，"像以前使用SpecularColor一样使用*Roughness*”，不久后我就会听到兴奋地惊讶地说道："这下好用了！" 但后面有一个有趣的评论是："*Roughness*的感觉是反的。" 事实证明，艺术家们希望看到他们所创作的贴图是按照下面的规律来展现的-更亮的纹素等于更亮的镜面高光。如果图像存储

Roughness，那么明亮相当于更高的*Roughness*，而这将导致不那么强烈的Specular效果。（**译者注**：后面这半段是在解释Artist为什么觉得是反的。）

我还收到过无数次的问题是：”Metallic是二元的吗？（**译者注**：表示非0即1）”对于这个问题，我最初会解释混合材料或分层材料的微妙之处。后来我了解到，最好是直接说 "是的！" 原因是一开始的艺术家们不愿意把参数设置成绝对值；我发现金属的Metallic值为0.8的情况很常见。接下来讨论的Material Layers，应该是描述99%的情况下Metallic不是0就是1的方式。

在过渡过程中，我们也遇到了一些问题，这些材质将不能再被复制。其中最重要的一组问题来自于目前Epic公司正在制作的游戏《Fortnite-堡垒之夜》。《Fortnite》采用了非真实感渲染的艺术方向，并特意使用了互补色的漫反射和镜面反射，这在物理上是不可行的，也是我们的新Material模型中故意不能重现的。经过长时间的讨论后，我们决定继续支持旧的DiffuseColor/SpecularColor作为引擎切换，以保持Fortnite的质量，因为当时Fortnite已经进入开发阶段。但是，我们不认为新的模型会像迪士尼在《Wreck-It Ralph》中使用的那样，排除了非真实感的渲染模型，我们打算在未来的项目中继续支持使用非真实感的渲染模型。

2.3.2 材质分层-Material Layers

与我们之前的方法相比，将来自于共享库的Material Layers进行混合提供了许多好处，之前的方法是单独指定材质的参数，而在新的方法中，这些参数的值来自于为每个特定模型制作的纹理。

- 在多个Assets中复用以前的工作；

- 减少了单一Asset的复杂性；

- 统一和集中了游戏外观所使用的材质，使美术和技术方向更容易。

为了完全接受这种新的工作流程，我们需要重新思考我们的工具。虚幻引擎在UE3的早期就有一个基于节点图的材质编辑器。这个节点图指定了输入（纹理、常量）、操作和输出，并将其编译成着色器代码。

尽管材质分层是这项工作的主要目标，但出乎意料的是，在工具端几乎不需要添加什么东西来支持材质层的创建和混合。UE4的材质编辑器中的节点图的部分已经可以被分组到函数中，并从多个材质中使用。这个功能对于实现材质图层来说是很自然的。将材质层保留在我们基于节点的编辑器中，而不是作为一个固定的函数系统放在上面，这使得图层可以以可编程的方式进行映射和组合。

为了简化工作流程，我们添加了一个新的数据类型，即材质属性，它保存了所有的材质输出数据。这个新的类型和我们的其他类型一样，可以作为单引脚传入和传出材料函数，也可以沿线传递，直接输出。有了这些变化，材质图层可以像以前的纹理一样，拖入、组合、操作和输出材质图层，就像以前的纹理一样。事实上，由于采用了图层，大多数的材质图往往会更简单，因为采用图层是定制给特定材质的主要东西，所以图层是如何映射和混合的。这比以前的参数特定操作要简单得多。

由于有一个小的线性材质参数集，实际上完全在着色器中混合图层是非常实用的。我们觉得这比纯粹的离线合成系统的质量有了很大的提高。由于能够在不同频率下映射数据，纹理数据的表观分辨率可以非常高：每个顶点或低频纹理数据可以是唯一的，图层混合蒙版、法线贴图和Cavity贴图在每个网格中指定，材料层在网格表面上分层。更高级的情况下，可能会使用更多的频率。虽然由于着色器的成本问题，我们可以使用的图层数量实际上是有限的，但是，我们的艺术家还没有发现很多被限制的

情况。（**译者注**：这里使用了很多频率，在原文中是Frequency，翻译是没问题，但是，我不是太理解这个具体指的什么含义。哪位大神有更好的理解，可以评论分享下。）

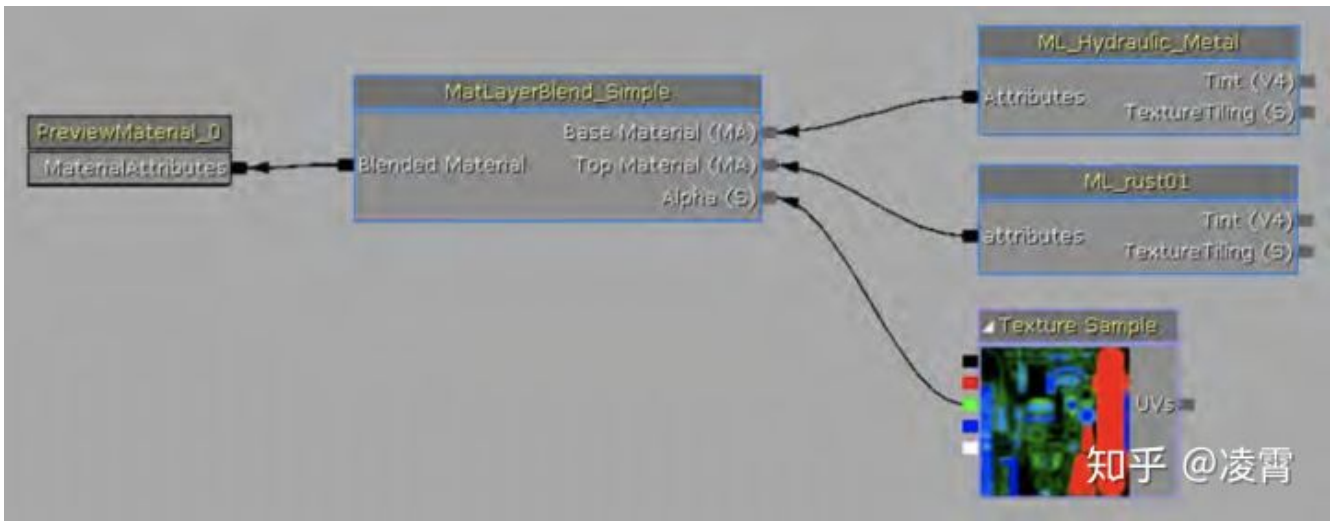


图6：在UE4材质编辑器中的简单材质分层

一个值得关注的地方是，案例中的艺术家通过将一个网格分割成多个部分来绕过了in-shader分层的限制，导致了更多的Draw Call。虽然我们期望在UE4中，由于CPU端代码优化，我们会有更少的Draw Call次数，但这似乎是未来可能会成为问题的根源。我们还没有研究的一个领域是使用动态分支来降低图层100%覆盖的区域的着色器成本。

到目前为止，我们在Material Layers方面的体验是非常正面的。我们已经看到，既得到了效率的提高，也看到了质量的大幅提升。我们希望通过改进Material Layer库的UI界面，使艺术家们更容易找到和预览图层。除了目前的运行时系统外，我们还打算研究一个离线合成/烘烤系统，以支持更多的图层并提供更好的可扩展性。

图7：许多材质层进行互换，并与铁锈来进行混合

图8：利用多频率的细节处理产生的Material Layer的结果

> 译者注：在UE4中，提供了两个比较容易混淆的概念，分别是”Layered Material-分层材质“和”Material Layers-材质图层“。
>
> 分层材质：可以将分层材质想像成“材质内的材质”。它们提供了一种方法来创建单个具有一系列子材质（即 *层*）的材质，这些子材质可以使用按像素的操作（例如蒙版）放在对象表面上。它们适合于处理独特表面类型之间的复杂混合。在上面的火箭图中，最右边的火箭使用了单独的材质层，包括铬合金、铝和铜，并且按像素在各材质之间进行混合。此效果可通过分层材质轻松实现。
>
> 材质图层：材质图层使您能够使用新的材质图层和材质图层混合资源来 **将您的材质组合在堆栈中**。这使您能够无需手动构建节点部分即可构建正确的材质图表。该功能类似于Material函数，但支持创建子实例。

分层材质的示例

2.4、光照模型-Lighting Model

正如着色一样，我们希望光照模型也能更偏向物理来提高我们的展现效果。我们投入研究的两个领域是光照衰减和非精确发射源（non-punctual sources of emission）——通常被称之为区域光源（Area Lights）。

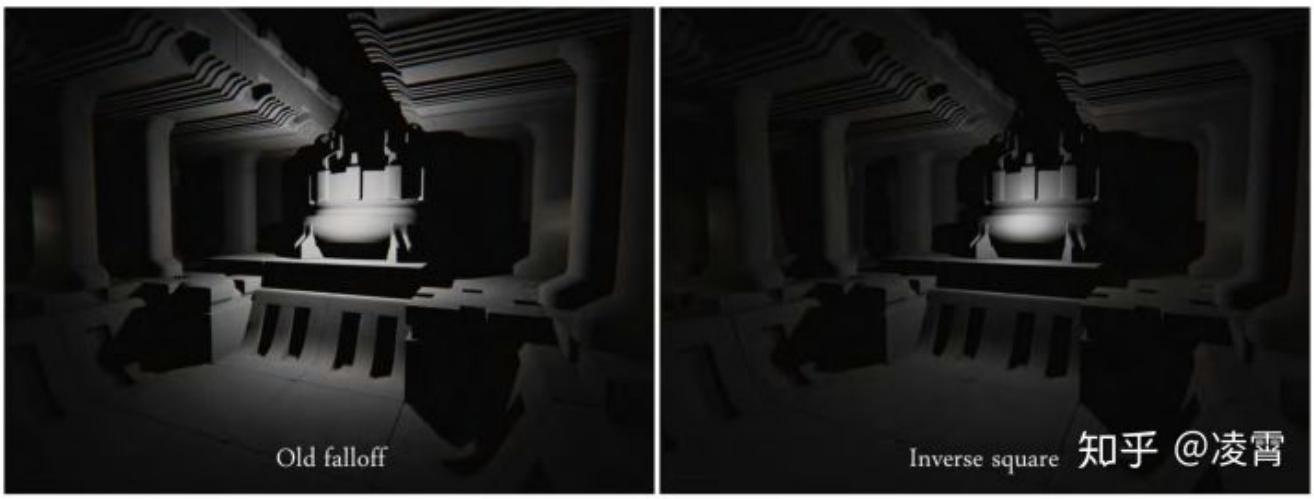
2.4.1、光照衰减

提升光照衰减的质量是相当直接的：我们采纳了物理精确的反平方衰减并切换到光度学的光通量的亮度单位。在改变这个衰减模型后，我们需要解决一个小问题：这种衰减函数没有办法直接处理无穷远处光的能量才能衰减到0的问题。出于效率的考量——无论是实时计算还是离线计算——我们依然需要人工限制灯光的影响。有许多的方式可以实现[4]，但是我们选择对多数的灯光影响保持相对不受影响的方式去修改反平方函数，同时提供一个逐渐过渡到0的方案。一个比较好的性质是通过修改灯光的半径并不会改变它的有效亮度。当光照被艺术性的锁定时，这个属性是非常重要的，但是因为性能的原因灯光范围依然需要调整。

$$\mathbf{falloff} = \frac{\text{saturate}(1 - (distance/lightRadius)^4)^2}{distance^2 + 1} \tag{9}$$

分母中的1是为了防止函数在靠近光源的距离上产生接近无穷大的值，它可以作为某些不需要表现物理准确性的情况下，让艺术家可以进行调整的参数。

这种简单的改变所带来的质量差异，特别是在有很多局部光源的场景中是很重要的，它很可能是性能和效果折中的最后手段。（**译者注**：原文这句话非常难理解，我使用意译的方式。意思是说，在有很多Local Light的时候，可以通过这个分母的调整，来调整光所影响的范围，从而提升运算效率。）



2.4.2、区域光

区域光源不只是产生更逼真的图像。在使用基于物理材料时，它们也相当重要。我们发现，如果没有它们，艺术家们往往在直觉上会避免画出非常低的粗糙度值，因为这导致了无限小的镜面高光，看起来很不自然。从本质上说，他们试图重现来自准点光源的区域照明效果。

不幸的是，这种反应导致了阴影和照明之间的耦合，打破了基于物理渲染的核心原则之一：当材质在不同的光照环境中使用时，不需要对材质进行修改，而这些材质的光照环境与创建时的环境不同。

区域灯是一个活跃的研究领域。在离线渲染中，常见的解决方案是从光源表面的许多点进行光照-使用均匀采样或重要性采样[12][20]。这对于实时渲染来说完全不切实际。在讨论可能的解决方案之前，我们的要求是这样的。

- 一致的材质外观：用于漫反射BRDF和镜面反射BRDF评估的能量不能是显著不同。

- 当实心角接近零时，接近点光模型：我们不想让我们的着色模型失去任何一个方面来实现这个目标。

- 足够快，可以在任何地方使用：否则，我们无法解决上述的 "偏差roughness "问题。

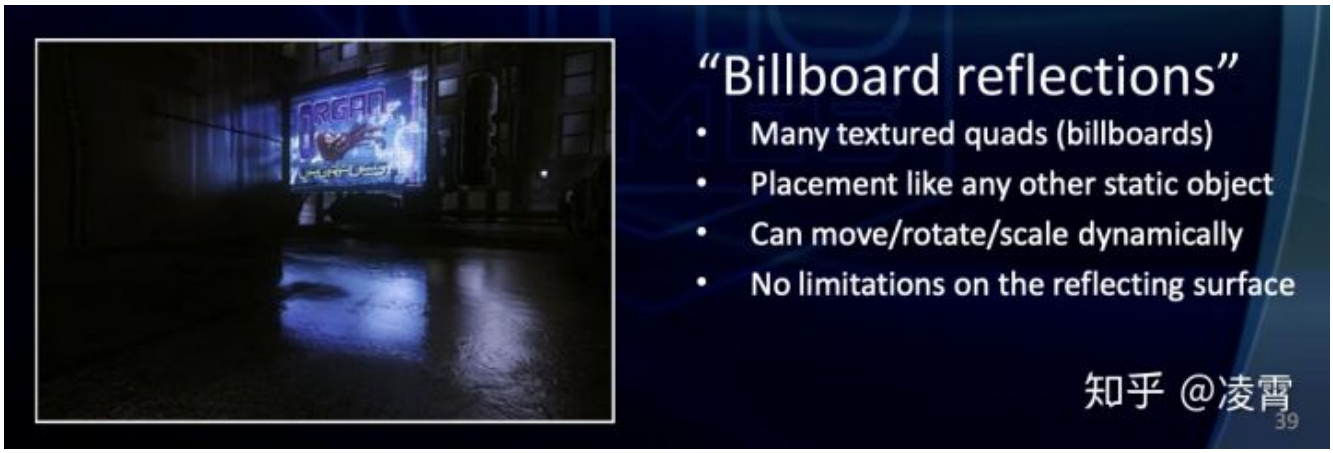
2.4.2.1 Billboard反射

Billboard反射[13]是一种可用于离散光源的IBL形式。将存储发射光的二维图像映射到三维空间中的一个矩形图像上。类似于环境贴图预滤波，对图像进行不同大小的镜面分布锥体进行预滤波。从这个图像中计算镜面阴影可以被认为是一种圆锥追踪的形式，其中一个圆锥近似于镜面NDF。在圆锥中心的射线与Billboard的平面相交。然后将图像空间中的相交点作为纹理坐标，并以相交处的圆锥半径作为纹理坐标，推导出一个适当的预滤波mipmap级别。但是，可悲的是，虽然图像可以直接表达非常复杂的区域光源，但Billboard反射由于以下原因未能满足我们的第二个要求。

- 图像是在一个平面上进行预过滤的，所以在图像空间中可以表示的实体角是有限的。

- 当光线不与平面相交时，就无法获得Lighting的数据。

- 光向量l是未知的，或者假设为反射向量。



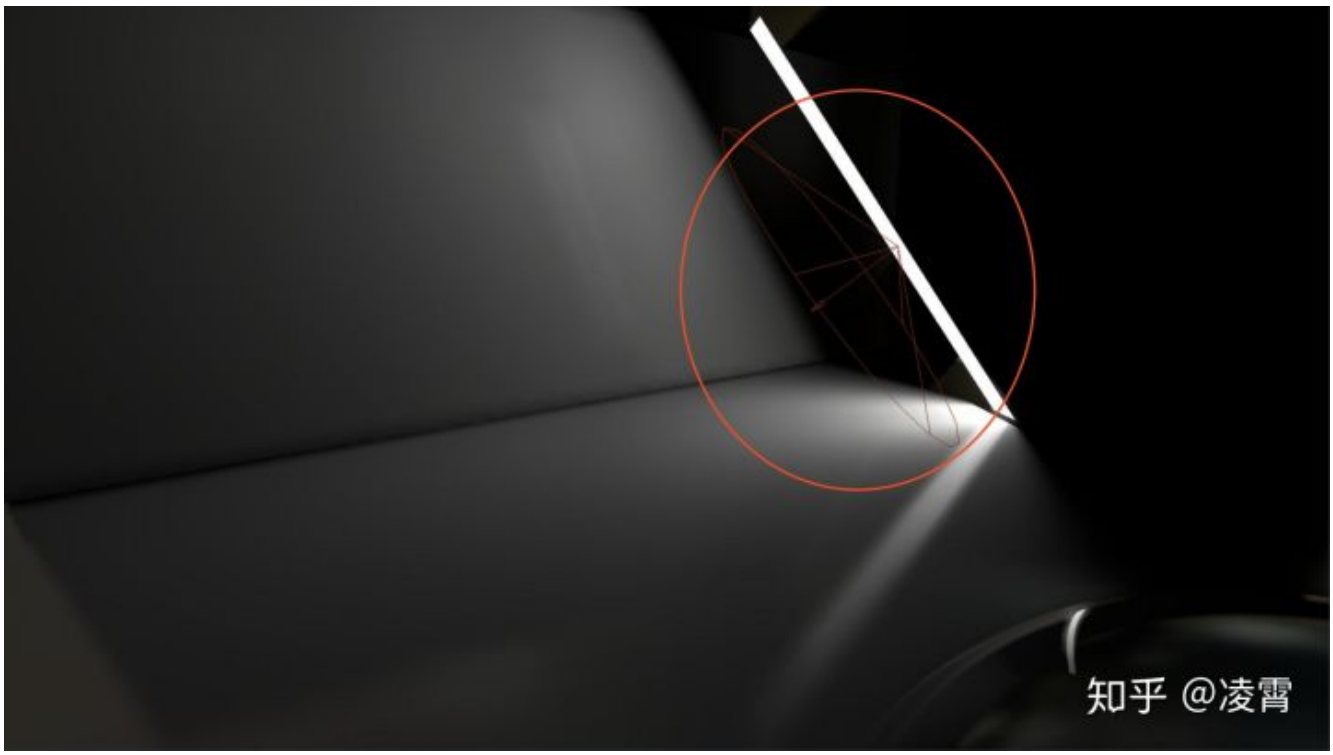
[13]这个PPT中关于Billboard Reflection的介绍

> 译者注：这个Billboard反射是UE3中提供的一个feature，从[13]中可以看出来。作者在这里拿出来将，只是说，对于区域光这个领域的最新研究和实践进展，以及是否作为一个比较重要的参考实现。

2.4.2.2、椎体相交-Cone Intersection

锥体追踪不需要预先过滤，可以通过分析来完成。我们实验过的一个版本是用Oat的圆锥与球体相交方程[15]来追踪圆锥，但成本太高，不实用。最近由Drobot[7]提出的另一种方法是将圆锥与面向阴影点的圆盘相交。然后在相交区域上进行多项式近似NDF的多项式积分。

随着Drobot最近的进步，这似乎是一个有趣的研究领域，但就目前的形式来看，它并不能满足我们的要求。由于使用圆锥体，镜面分布必须是径向对称的。这排除了拉伸高光，这是微表面Specular模型的一个非常重要的特征。此外，就像广告牌反射一样，也没有办法定义出一个光的方向向量，而这个恰恰是Shading模型所必须的。



[7]中的一个案例，可以看出是使用Cone来模拟一个矩形光源

> 译者注：[7]是Guerrilla Games公司进行的《Lighting Killzone : Shadow Fall》演讲。这个主要是在Guerrilla Games公司对基于物理的Lighting和Area Lighting的报告内容。在这个PPT中，针对Area Light，他们都是采用Cone Interaction的方式来处理的。显然，从本文的角度上看，并不认为使用Cone来模拟Area Light是一个适应广泛的方式。虽然，从效果来看，也不是不能接受。下面给一张他们公司产品《KillZone》的截图。
>
> Guerrilla Games是一家隶属于Sony Interactive Entertainment LLC的荷兰第一方游戏开发商，曾为索尼电脑娱乐开发杀戮地带以及为Eidos Interactive开发越南1967。

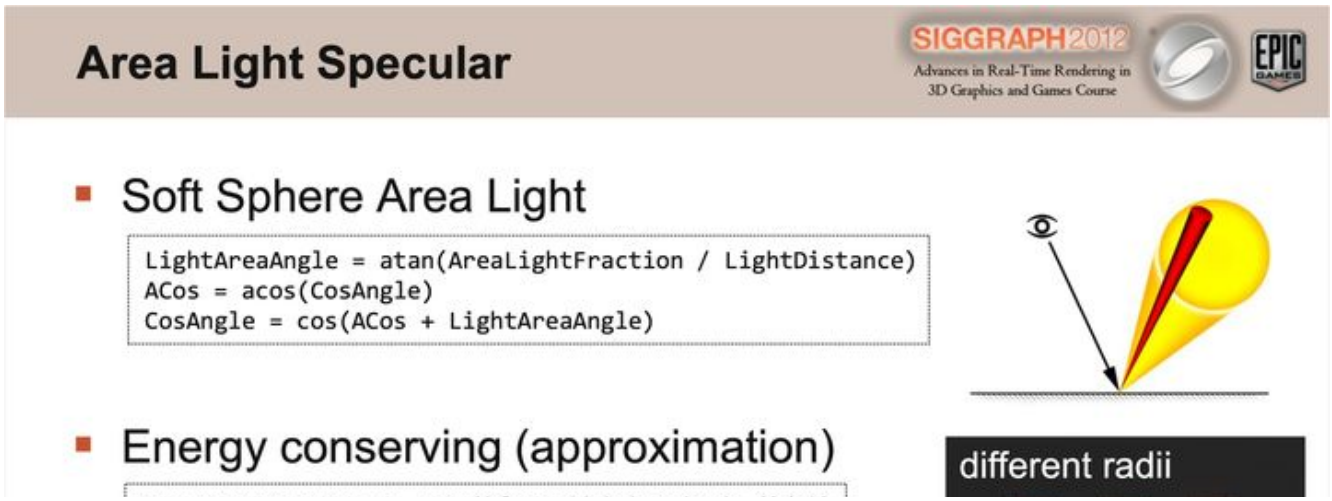
《KillZone-杀戮地带》的截图

2.4.2.3、镜面反射D修改-Specular D Modification

我们去年提出的一种方法[14]是根据光源的实心角来修改镜面分布。这背后的理论是认为光源的分布与相应的圆锥角D(h)相同。将一个分布与另一个分布进行卷积运算，可以通过将两个圆锥角相加得到一个新的圆锥。为此，将方程3中的α换算成有效锥角，加上光源的角度，再换算回来。现在用这个α′来代替α。我们用下面的近似值来实现这个目的。

$$\alpha' = \text{saturate}\left(\alpha + \frac{sourceRadius}{3 * distance}\right) \tag{10}$$

虽然效率很高，但遗憾的是，这种技术并不能满足我们的第一个要求，因为当用大面积的灯光照射时，非常有光泽的材料会显得很粗糙。这听起来可能很明显，但当镜面NDF是紧凑的，例如Blinn-Phong，这种技术的效果要好得多，从而更好地匹配光源的分布。对于我们所选择的Shading模型（基于GGX），这并不可行。



```
SpecularLighting /=  pow(ACos + LightAreaAngle, 2) * 10
```

> **译者注**：上面两页PPT是文中提及到的[14]中的内容。从第二张PPT中，可以看到四张对比图中间的奖杯表现的确实不是特别好。

图10：左侧为参考，右侧为Specular D修改下的效果。由于光源是球形的面积光，在gloosy比较高的材质以及在掠射角度下显示的比较粗糙，如抛光后的铜头所示。

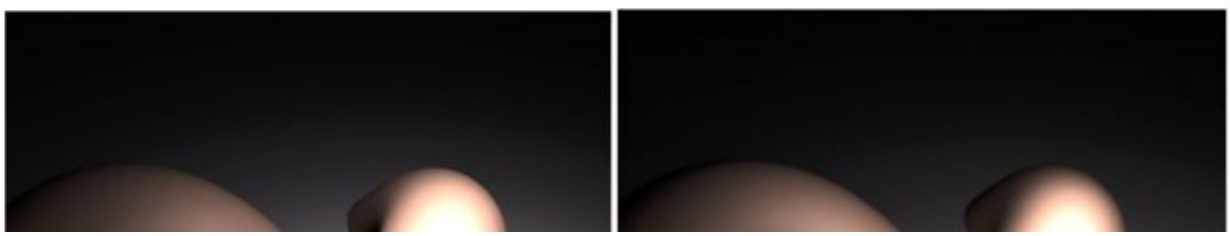
2.4.2.4、代表点-Representative Point

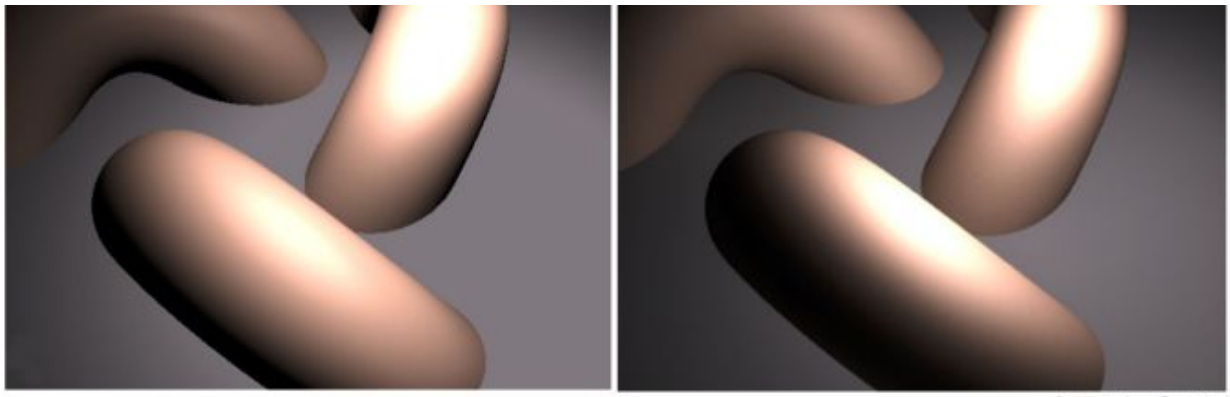
如果对于一个特定的着色点（Shading Point），我们可以把来自区域光的所有光都看作是来自光源表面上的一个代表点，那么我们的着色模型就可以直接使用了。一个合理的选择是贡献最大的点。对于Phong分布来说，这是光源上与反射光线夹角最小的点。

这种技术之前已经发表过[16][22]，但能量守恒问题一直没有得到解决。通过移动发射光的原点，我们有效地增加了光的实体角，但没有补偿额外的能量。纠正它比除以实心角略微复杂一些，因为能量差异取决于镜面分布。例如，对于粗糙的材质，改变入射光的方向会导致能量的变化很小，但对于有光泽的材质，能量的变化可能是巨大的。

> **译者注**：[22]的摘要：在大多数的渲染系统中，区域光源被分解成许多点光源，这需要繁重的计算才能得到逼真的结果。在本文中，我们提出了一种新颖的方法：**用点光源近似区域光源，对着色模型的每个分量进行点光源的近似**。点光源的位置和强度取决于场景点的位置和方向。因此，场景的渲染是利用即时的点光源进行着色。我们工作的关键贡献在于，我们能够实时地用任意形状和任意强度分布的区域光源进行遮光，并且遮光模型可以是通用的。而且计算成本与区域光源的复杂度无关。实验结果表明，我们的方法所产生的结果与地面真相相当。我们的方法可以扩展到其他类型的局部光源，如弧形光源和体积光源等。

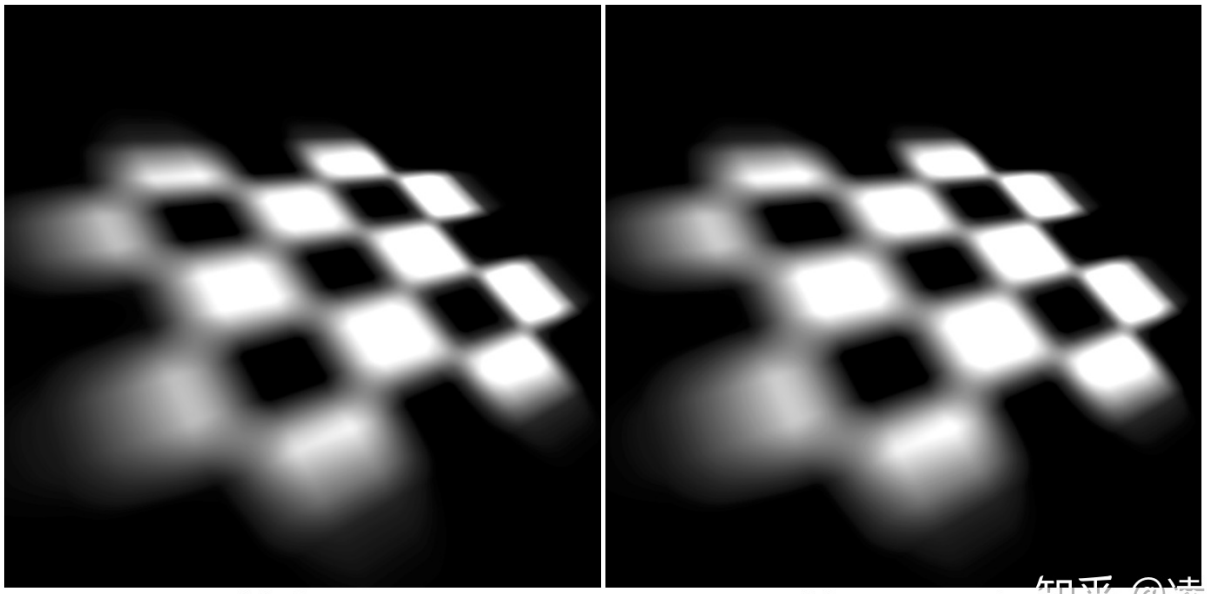
[PDF] One-Shot Approximate Local Shading | Semantic Scholarwww.semanticscholar.org/paper/One-Shot-Approximate-Local-Shading-Wang-Lin/5ff1eed023f9f9c77d787a00d10a5d443f1525a0





(a) Ours (b) *Mental Ray*

图：跟Mental Ray进行对比。这里的光源是一个Disk形状的。



(a) Ours (b) Ray Tracing

图：跟光追模型进行对比。这里的光源是一个checker board模式。

2.4.2.5、球形光-Sphere Lights

如果球在地平线之上，球形光的辐照度等于点光[18]。尽管反直觉，如果我们接受误差，这意味着当球落在地平线之下的时候，我们仅仅需要处理镜面反射光照。我们通过寻找距离射线最小的点近似发现与反射射线之间最小的角度的点。对于一个球这是简单明了的：

$$\begin{aligned} \mathbf{centerToRay} &= \mathbf{L} - (\mathbf{L}\cdot\mathbf{r})\,\mathbf{r} \\ \mathbf{closestPoint} &= \mathbf{L} + \mathbf{centerToRay} * \text{saturate}\left(\frac{sourceRadius}{|\mathbf{centerToRay}|}\right) \\ \mathbf{l} &= \|\mathbf{closestPoint}\| \end{aligned} \tag{11}$$

这里，L是从着色点到光源中心的向量，并且sourceRadius是灯光球体的半径，r是反射向量。在这个案例中，射线与球相交，计算点将会是射线到球体中心的最近点。一旦标准化，它就是相同的。

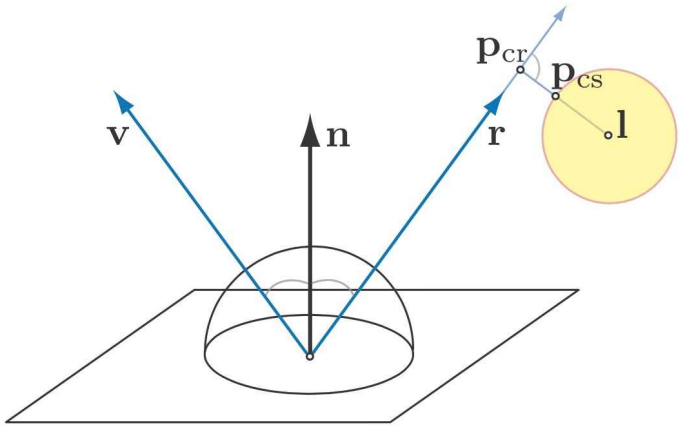


Figure 10.10. Karis representative point approximation for spheres. First, the point on the reflection ray closest to the sphere center $\mathbf{l}$ is computed: $\mathbf{p}_{cr} = (\mathbf{l} \cdot \mathbf{r})\mathbf{r} - \mathbf{l}$. The point on the sphere surface closest to $\mathbf{p}_{cr}$ is then $\mathbf{p}_{cs} = \mathbf{l} + \mathbf{p}_{cr} \cdot \min(1, \frac{\text{radius}}{||\mathbf{p}_{cr}||})$.

图：《Real-Time Rendering 4》中的一张图

> 从这张图中，

$$P_{cs}$$

是最后选中的closet point，然后，光线的方向就是从

$$P_{cs}$$

到P。

通过移动发射光的原点到球的表面，我们通过球的对角高效的拓宽了镜面反射分布。尽管它不是一个微表面的分布，这可以用标准化的Phong分布来解释。

$$I_{point} = \frac{p+2}{2\pi} \cos^p \phi_r \tag{12}$$

$$I_{sphere} = \begin{cases} \frac{p+2}{2\pi} & \text{if } \phi_r < \phi_s \\ \frac{p+2}{2\pi} \cos^p(\phi_r - \phi_s) & \text{if } \phi_r \geq \phi_s \end{cases} \tag{13}$$

这里

$$\phi_r$$

是r和L之间的夹角，并且

$$\phi_s$$

是球对角的半角，

$$l_{point}$$

是标准化的，意味着积分的结果在半球上为1。

$$l_{sphere}$$

显然不再标准化并取决于p的指数，积分可以大的多。

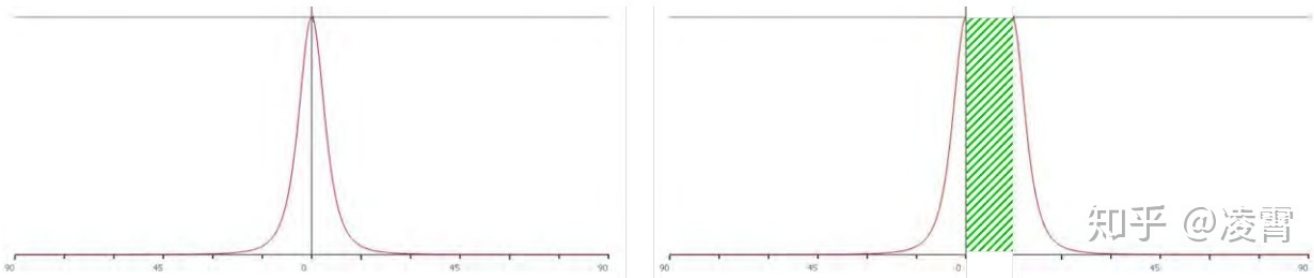


图11：用等式13解释的拓宽效应的可视化

为了近似模拟能量上的增长，我们拓宽了光的实心角分布，应用之前描述的镜面反射D修改。我们使用标准化因子实现更广泛的分布并替换初始的标准化因子。对于GGX，标准化因子是1/πα2。为了对代表点操作推导一个近似的标准化，我们将新的拓宽的标准化因子除以初始的标准化因子：

$$\text{SphereNormalization} = \left(\frac{\alpha}{\alpha'}\right)^2 \tag{14}$$

代表点方法的结果满足我们所有的需求。通过正确的处理能量守恒，无论光源大小如何，材质表现一致。光滑材质依然提供了锐利边缘的镜面反射高光，并且因为它只是修改了BRDF的输入，我们的着色模型不会受任何影响。最后，它足够高效，允许我们的艺术家在任何地方使用它。

2.4.2.6、灯管-Tube Lights

球形灯对于表现**灯泡**有帮助，灯管（胶囊体）对于表现现实世界中相当广泛的荧光灯有帮助。开始，我们用一个有长度但是半径为0，也被称为线性灯光来解决灯管。只要线段在水平线之上，线段的辐照度可以被解析地积分[16,17]。

$$\int_{\mathbf{L}_0}^{\mathbf{L}_1} \frac{\mathbf{n}\cdot\mathbf{L}}{|\mathbf{L}|^3} d\mathbf{L} = \frac{\frac{\mathbf{n}\cdot\mathbf{L}_0}{|\mathbf{L}_0|} + \frac{\mathbf{n}\cdot\mathbf{L}_1}{|\mathbf{L}_1|}}{|\mathbf{L}_0||\mathbf{L}_1| + (\mathbf{L}_0\cdot\mathbf{L}_1)} \tag{15}$$

这里

$$L_0$$

和

$$L_1$$

是从着色点到线段终点的向量。

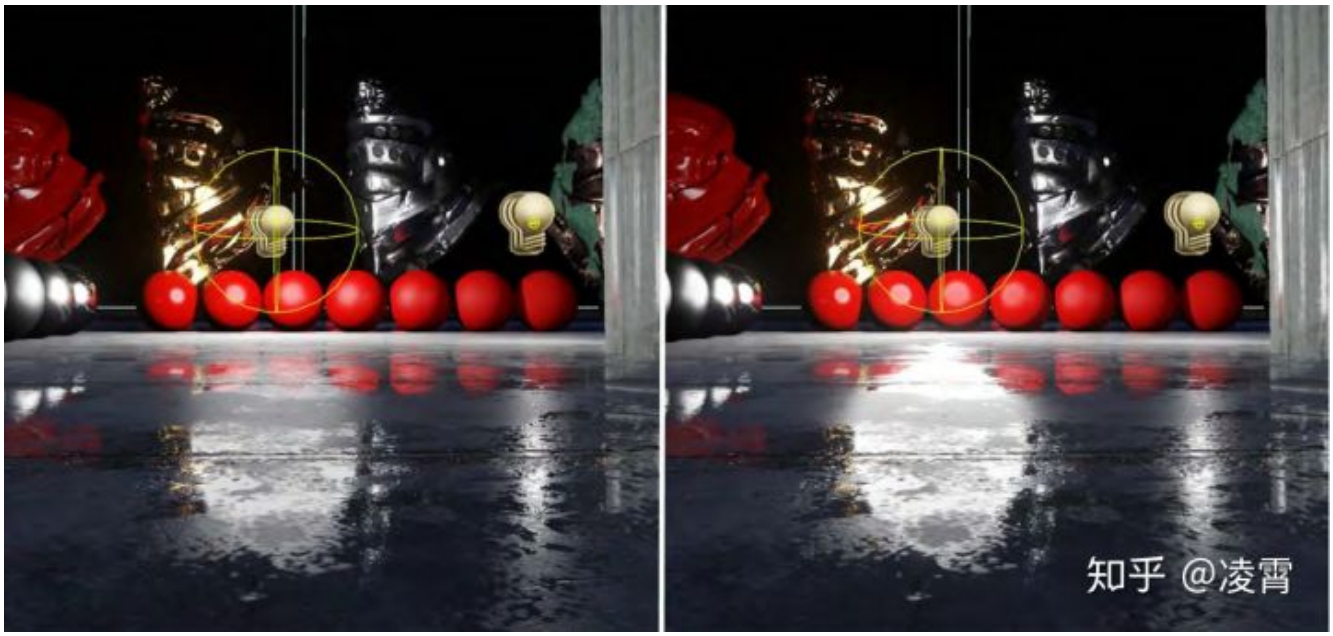


图12：左边为参考，右边是代表点方法。尽管能量守恒不完美，我们的近似值与参考值匹配得令人信服。

我们修改了这个方程式来防止辐照度为负值，除以0，并且当长度为0时匹配我们的点光衰减。

$$\text{irradiance} = \frac{2 * \text{saturate}\left(\frac{\mathbf{n}\cdot\mathbf{L}_0}{2|\mathbf{L}_0|} + \frac{\mathbf{n}\cdot\mathbf{L}_1}{2|\mathbf{L}_1|}\right)}{|\mathbf{L}_0||\mathbf{L}_1| + (\mathbf{L}_0 \cdot \mathbf{L}_1) + 2} \tag{16}$$

对于线性光镜面反射我们需要在下面的方程组里解出t：

$$\begin{aligned} \mathbf{L}_d &= \mathbf{L}_1 - \mathbf{L}_0 \\ \mathbf{l} &= \|\mathbf{L}_0 + \text{saturate}(t)\mathbf{L}_d\| \end{aligned} \tag{17}$$

Picott[16]发现与r最小角的t的值：

$$t = \frac{(\mathbf{L}_0 \cdot \mathbf{L}_d)(\mathbf{r} \cdot \mathbf{L}_0) - (\mathbf{L}_0 \cdot \mathbf{L}_0)(\mathbf{r} \cdot \mathbf{L}_d)}{(\mathbf{L}_0 \cdot \mathbf{L}_d)(\mathbf{r} \cdot \mathbf{L}_d) - (\mathbf{L}_d \cdot \mathbf{L}_d)(\mathbf{r} \cdot \mathbf{L}_0)} \tag{18}$$

和球形光的案例相似，我们近似了最小角并且求解，而不采用最短距离的求解方法：

$$t = \frac{(\mathbf{r} \cdot \mathbf{L}_0)(\mathbf{r} \cdot \mathbf{L}_d) - (\mathbf{L}_0 \cdot \mathbf{L}_d)}{} \tag{19}$$

$$|\mathbf{L}_d|^2 - (\mathbf{r} \cdot \mathbf{L}_d)^2$$

这会造成边界没有正常处理的情况，这样就不能总是找到最近点，但是这样计算起来成本略低，并且看起来提供了方程式18一样合理的结果。

注意，这是因为方程式18和19都把r视作一条线段而不是射线是重要的，解决方案都未能正确处理指向远离线段的射线。甚至对于完美的平面，这会造成从一个结束点到另一个结束点的生硬变化。我们通过在计算点和每一个结束点之间选择来解决这个问题，但是这样做成本较高。此时我们只好接受了这个瑕疵。

为了使能量守恒，我们应用了用于球形光的相同概念。镜面反射分布通过灯光的对角已经被拓宽，但是这次只是一维的，所以我们使用GGX的各向异性版本[2]。各向异性GGX的标准化因子是1/παxαy，在各向同性案例中，这里αx=αy=α，这给了我们：

$$\text{LineNormalization} = \frac{\alpha}{\alpha'} \tag{20}$$

因为我们只改变了灯光的原点并应用了一个能量守恒项，这些操作可以被加速。用线段和球来做，近似形状的卷积并且很好的模拟了灯管的表现。灯管的结果展示在图13。

图13：使用能量守恒的代表点方法的灯管

3、结论

我们在着色，材质和灯光领域转向基于物理的实现，已经被证实是非常成功的。在我们最近的《渗透者》demo的图形部分贡献极大，并且我们计划在未来所有的项目中使用这些实现。事实上，这些改变的可行部分已经集成进《堡垒之夜》，在这项工作开始之前便已经进展顺利的项目。我们打算在这些领域以实现更高的灵活性的目的继续提升，并且提升目标是各种场景和所有级别的硬件可以享受基于物理方法的好处的可扩展性。

4、鸣谢

我要感谢Epic Games，特别是渲染团队中的每一个人在这项工作和艺术家，提供了方向、 反馈，并最终做出美丽的东西与它的艺术家们的手。我要特别感谢Sébastien Lagarde，他发现了我的重要性取样数学中的一个错误，当修复后导致我们的环境BRDF解决方案的发展。永远不要低估了在全球范围内有才华的授权者来查看你写的代码是多么有价值。最后，我要感谢Stephen Hill和Stephen McAuley，感谢他们的反馈。

图9：反平方衰减的方案带来了更自然的渲染效果