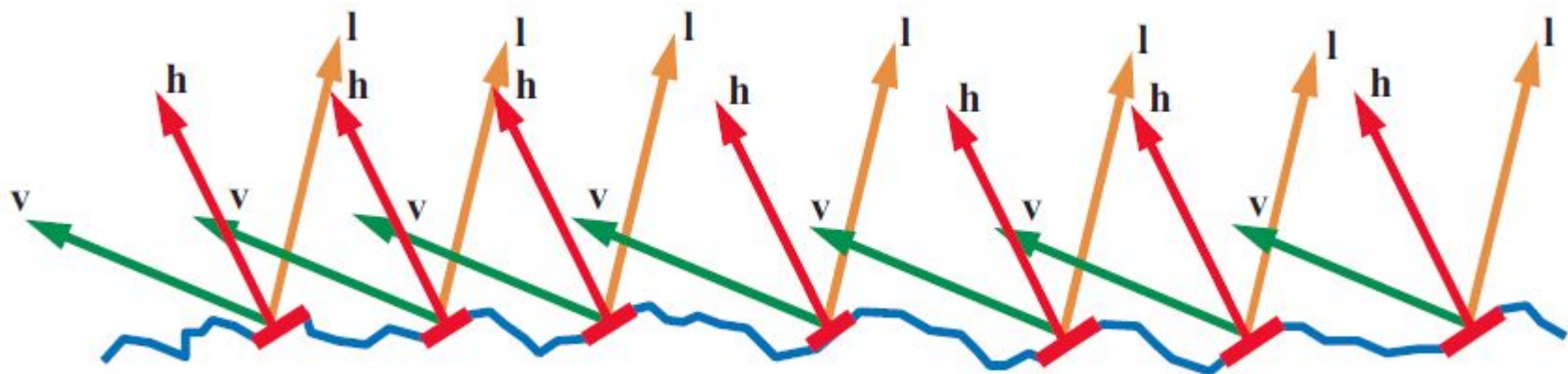


虽然是概述，但内容并还是有些多，写上一篇 PBR 概念概述后，也在考虑怎么继续下去，最后还是觉得先多写一些东西再慢慢总结，所以还是尽量把这些年 PBR 相关的 Paper 精粹沉淀下来吧。

因为 UE4 开源的缘故，所以一开始还从它入手。相关的 ppt 和 notebook 可以从下面的链接下载，同期的黑色行动 2（black op2）的 PBR 使用也是很有参考价值的，加上本文里也有 OP2 的 IBL 近似方法的介绍，如果没看过那也很值得下载的。  
<http://blog.selfshadow.com/publications/s2013-shading-course/>

UE4 的 paper 里的 PBR 介绍包括三部分：Shading Model ，Lighting Model ，Material Model，这篇就先从 Shading Model，也就是使用的 BRDF 开始吧，但要满足一个游戏的所有渲染效果，靠一个通用的 BRDF 也是无法达到的，所以也只能算是个概述吧，随着使用和学习的应用，也会继续补完 Shading Model 的介绍的。

首先，PBR 最大的特点还是引入了微平面概念



着色平面不再是一个完美的反射平面，而是想象成更多微小的反射平面组成。所以也就有了粗糙度的概念

Diffuse BRDF  
可以选择简单的 Lambert 或支持 Microfacet 的 Oren-Nayar  
UE4 默认使用的是 Lambert

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

Specular BRDF  
支持微平面概念的 Cook-Torrance Microfacet BRDF ，在直接照明和间接照明的 Shading Model 里使用

$$f_r(\mathbf{l}, \mathbf{e}) = \frac{D(\mathbf{l}, \mathbf{e})F(\mathbf{l}, \mathbf{e})G(\mathbf{l}, \mathbf{e})}{4(\mathbf{n} \cdot \mathbf{l})(\mathbf{n} \cdot \mathbf{e})}$$

由 Fresnel 项，NDF(Normal Distribution Function)项 Geometry Factor 项来组成，以获得更加物理的效果  
F D G 项会根据设备性能选择最适合的公式  
暂定 F 是 Schlick'的近似

$$F(\theta) = f_0 + (1 - f_0) \cdot 2^{-9.60232 \cos^8 \theta - 8.58092 \cos \theta}$$

F0 是垂直入射时的反射率(法线方向的 Specular Reflectance)，一般也就是存在 Specular Color map 里的数值了。

NDF(Normal Distribution Function) GGX

$$\alpha = \text{Roughness}^2$$

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

$$\frac{1}{\pi\alpha^2}$$

这个能量守恒因子(normalization factor)，用来保证出射光 < 入射光的，具体的求导会放在另外一篇里一同解说。  
Geometry Factor GGX

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

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

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

直接光照还是比较简单的，将公式和需要的参数直接套入现有的渲染管线就可以了。Forward Rendering 还好，多加几个参数也没有影像，如果是 Deferred Rendering 的话，就需要把 F0(Specular Color)放入到 Gbuffer 了。但这样对于以前 CE 那种 Gbuffer 还是有影响的。

孤岛危机 2 Deferred Lighting Slim G-Buffer

§A8B8G8R8 World Space BF Normals 24bpp + Glossiness 8bpp RT1

§Readback D24S8 Depth + Stencil bits for tagging indoor surfaces 8pp RT0

孤岛危机 3 Hybrid Deferred Rendering Thin G-Buffer 2.0 只传入 Specular Color 的灰度值

Channels				Format
Depth			AmbID, Decals	D24S8
N.x	N.y	Gloss, Zsign	Translucency	A8B8G8R8
Albedo Y	Albedo Cb,Cr	Specular Y	Per-Project	A8B8G8R8

罗马之子 Deferred Shading ， PBR 需要完整的 Specular Color

RT0	RGB: Normals XYZ		A: Translucency Luminance/Prebaked AO Term
RT1	RGB: Diffuse Albedo		A: Subsurface Scattering Profile
RT2	R: Roughness	GBA: Specular YCbCr/Transmittance CbCr	

而 UE4 是提供了 forward rendering 和 Deferred Shading 两种方案，因为 Computrer Shader 可以在 TBDR（Tile Based Deferred Rendering）上的应用，所以 Deferred lighting 这种方法基本上到 PBR 上已经基本没有什么优势了。基本上一个比较完整的 GBuffer+TBDR 管线算是现在的主流设计方式了。这些到了后面 PBR 渲染管线设计时再具体描述吧。

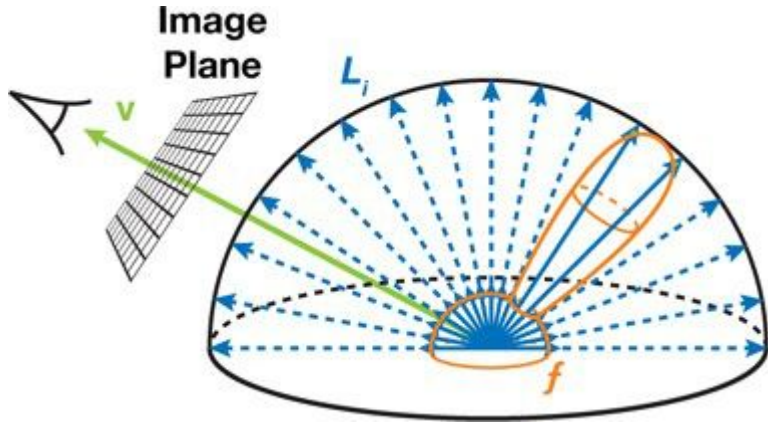
Image Based Lighting

使用预处理的环境光贴图来做光源的间接照明方案。

原始公式 IBL 公式，u 是入射光方向，v 是视点方向，Li 是每一个入射光，也就是 Environment Map 的信息，f 是我们前面提到的 BRDF 着色模型

$$L_{\theta}(\mathbf{v}) = \int_H L_i(\mathbf{u}) f(\mathbf{u}, \mathbf{v}) \cos \theta_{\mathbf{u}} d\mathbf{u}.$$

重要度采样（Importance Sampling）



原始公式是要对周围光照做一个均匀的随机 Sampling（Hammersley 随机采样），但像光滑材质上，大量的光会聚集在 Specular 方向上(镜面反射方向)，均匀采样无法获得准确的结果。在无法改变采用分布的情况下，使用 PDF(probability density function 概率采样函数)是一个近似解决的方法，把 PDF(p) 在公式里作为分母使用，PDF 是 0~1 的一个浮点数，在接近 Specular 方向，这种采样数需要较高的地方，PDF 值会变得较低，提高了最后采样的数值(间接来说就是提升了次数)，相反，在采样数较低的地方，PDF 值会更高，间接减少采样次数。也就有了下面这个公式的近似。

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



```
float3 ImportanceSampleGGX( float2 Xi, float Roughness , float3 N )
```

```
{
    float a = Roughness * Roughness;
    float Phi = 2 * PI * Xi.x;
    float CosTheta = sqrt( (1 - Xi.y) / ( 1 + (a*a - 1) * Xi.y ) );
    float SinTheta = sqrt( 1 - CosTheta * CosTheta );
    float3 H;
    H.x = SinTheta * cos( Phi );
    H.y = SinTheta * sin( Phi );
    H.z = CosTheta;
    float3 UpVector = abs(N.z) < 0.999 ? float3(0,0,1) : float3(1,0,0);
    float3 TangentX = normalize( cross( UpVector , N ) );
    float3 TangentY = cross( N, TangentX );
    // Tangent to world space
    return TangentX * H.x + TangentY * H.y + N * H.z;
}
```



```
float3 SpecularIBL( float3 SpecularColor , float Roughness , float3 N, float3 V )
```

```
{
    float3 SpecularLighting = 0;
    const uint NumSamples = 1024;
    for( uint i = 0; i < NumSamples; i++ )
    {
        float2 Xi = Hammersley( i, NumSamples );
        float3 H = ImportanceSampleGGX( Xi, Roughness , N );
        float3 L = 2 * dot( V, H ) * H - V;
        float NoV = saturate( dot( N, V ) );
        float NoL = saturate( dot( N, L ) );
        float NoH = saturate( dot( N, H ) );
        float VoH = saturate( dot( V, H ) );
        if( NoL > 0 )
        {
            float3 SampleColor = EnvMap.SampleLevel( EnvMapSampler , L, 0 ).rgb;
            float G = G_Smith( Roughness , NoV, NoL );
            float Fc = pow( 1 - VoH, 5 );
            float3 F = (1 - Fc) * SpecularColor + Fc;
            // Incident light = SampleColor * NoL
            // Microfacet specular = D*G*F / (4*NoL*NoV)
            // pdf = D * NoH / (4 * VoH)
            SpecularLighting += SampleColor * F * G * VoH / (NoH * NoV);
        }
    }
}
```



```
    return SpecularLighting / NumSamples;
}
```



上面是计算 Specular 间接光的 shader 伪代码，1024 次对实时的 GPU 来说还是很难的，需要对公式做拆分

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

把上面的公式拆分成两部分，而第 1 个部分和环境光贴图相关的，可以一起进行预计算，也是下面要说到的 Pre-Filtered Environment Map

#### Pre-Filtered Environment Map

而 UE4 在拆分时还是做了一些额外的改动，那就是第 1 个部分里的除了采样环境光外，为了更多预计算，把第 2 部分里基于 GGX 的 PDF 也放到了预处理里，PDF 公式里需要的 V(视口向量)和 N(法线)，所以这里只能就只能假设  $\mathbf{n} = \mathbf{v} = \mathbf{r}$  了。



```
float3 PrefilterEnvMap( float Roughness , float3 R )
{
    float3 N = R;
    float3 V = R;
    float3 PrefilteredColor = 0;
    const uint NumSamples = 1024;
    for( uint i = 0; i < NumSamples; i++ )
    {
        float2 Xi = Hammersley( i, NumSamples );
        float3 H = ImportanceSampleGGX( Xi, Roughness , N );
        float3 L = 2 * dot( V, H ) * H - V;
        float NoL = saturate( dot( N, L ) );
        if( NoL > 0 )
        {
            PrefilteredColor += EnvMap.SampleLevel( EnvMapSampler , L, 0 ).rgb * NoL;
            TotalWeight += NoL;
        }
    }
    return PrefilteredColor / TotalWeight;
}
```



PrefilterEnvMap 生成部分的 shader 代码。

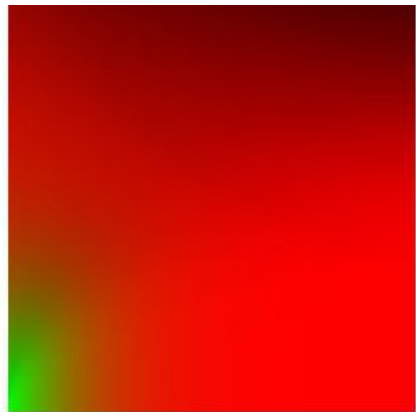
而后面的部分，我们可以通过 Schlick 近似的 Fresnel 公式来进行拆分。

$$F_{schlick} = (1.0 - F_0)(1 - \cos \theta)^5 + F_0$$

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

这个时候，我们可以把方程看成是  $F_0 * \text{Scale} + \text{Offset}$  的形式了， $F_0$  也就是 Specualr Color 可以从材质获取，也就是说，我们把 Scale 和 Offest 预计算出来。并通过 roughness 和  $\mathbf{N} \cdot \mathbf{V}$ ，也就是  $\cos \theta$  作为 LUT 的查找项

Roughness



$\cos \theta_v$

这样就可以把公式重新组合起来：



```
float2 IntegrateBRDF( 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;
    const uint NumSamples = 1024;
    for( uint i = 0; i < NumSamples; i++ )
    {
        float2 Xi = Hammersley( i, NumSamples );
        float3 H = ImportanceSampleGGX( Xi, Roughness , N );
        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 G = G_Smith( Roughness , NoV, NoL );
            float G_Vis = G * VoH / (NoH * NoV);
            float Fc = pow( 1 - VoH, 5 );
            A += (1 - Fc) * G_Vis;
            B += Fc * G_Vis;
        }
    }
    return float2( A, B ) / NumSamples;
}
```



最后把第一部分 pre-filterer 的 cubemap 和第 2 部分计算的部分相乘，就都出 IBL 的最终结果了



```
float3 ApproximateSpecularIBL( float3 SpecularColor , float Roughness , float3 N, float3 V )
{
    float NoV = saturate( dot( N, V ) );
    float3 R = 2 * dot( V, N ) * N - V;
    float3 PrefilteredColor = PrefilterEnvMap( Roughness , R );
    float2 EnvBRDF = IntegrateBRDF( Roughness , NoV );
    return PrefilteredColor * ( SpecularColor * EnvBRDF.x + EnvBRDF.y );
}
```



这里需要注意一点：EPIC 在 ppt 里提供的 shader 代码，并不是实际运行的代码，也就是说 PrefilterEnvMap 和 IntegrateBRDF 这两个函数还是 ALU 方式的实现，而实际上是应该用 LUT 的方式来替换的。也就是下面的 shader 代码



```
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;
}
```



PreIntegratedGF 就是我们前面提到的那张红绿的 LUT 图，这里最后算得的结果，才是 UE4 最终选择的近似方案，也是

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

里后面的部分，而前面部分则保存在 AmbientCubemap 里，对 AmbientCubemap 采样



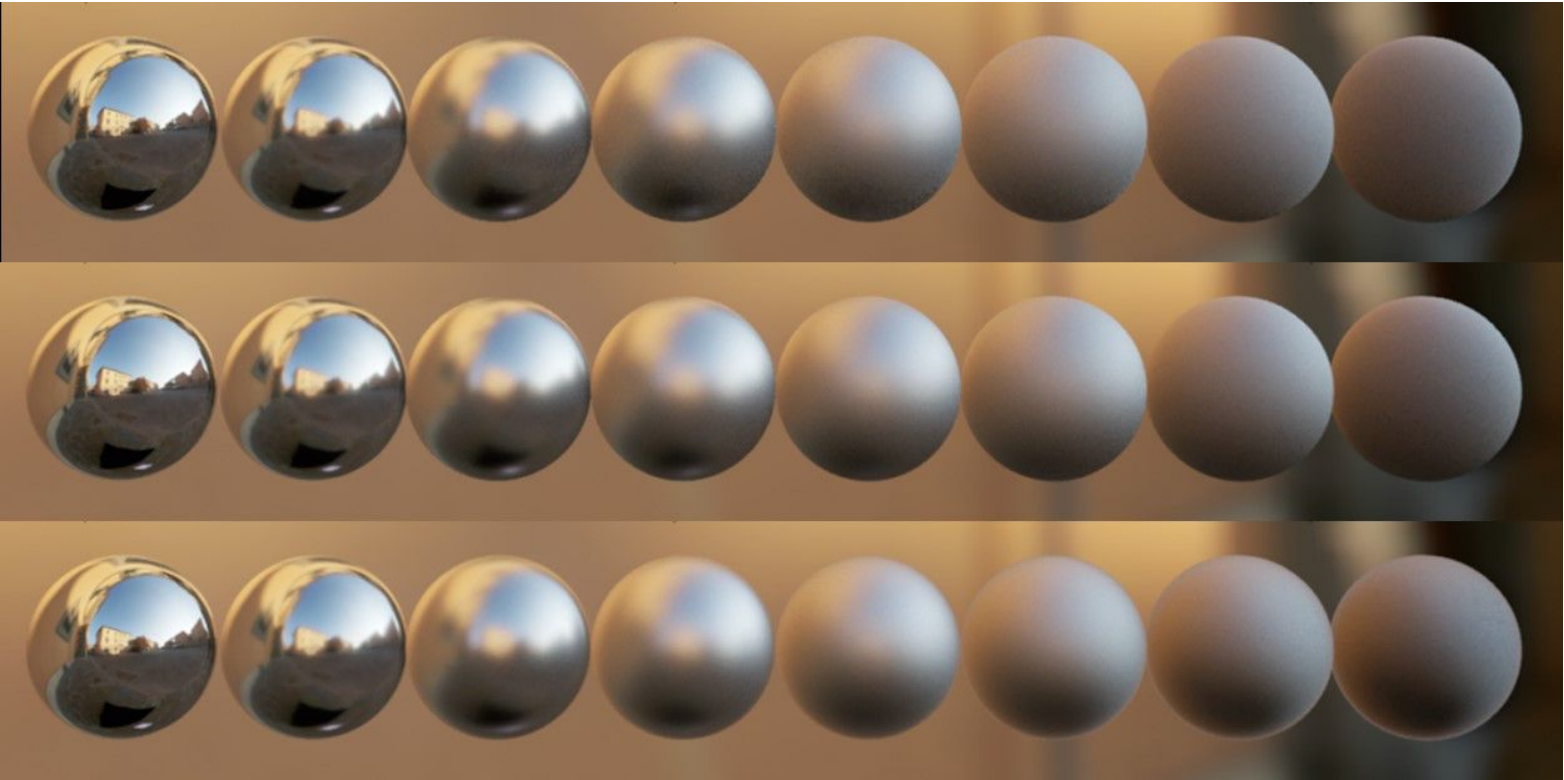
```
floatMip=ComputeCubemapMipFromRoughness(GBuffer.Roughness,AmbientCubemapMipAdjust.w );
float3SampleColor=TextureCubeSampleLevel(AmbientCubemap,AmbientCubemapSampler, R,Mip).rgb;

SpecularContribution+=SampleColor*EnvBRDF(GBuffer.SpecularColor,GBuffer.Roughness,NoV);
```

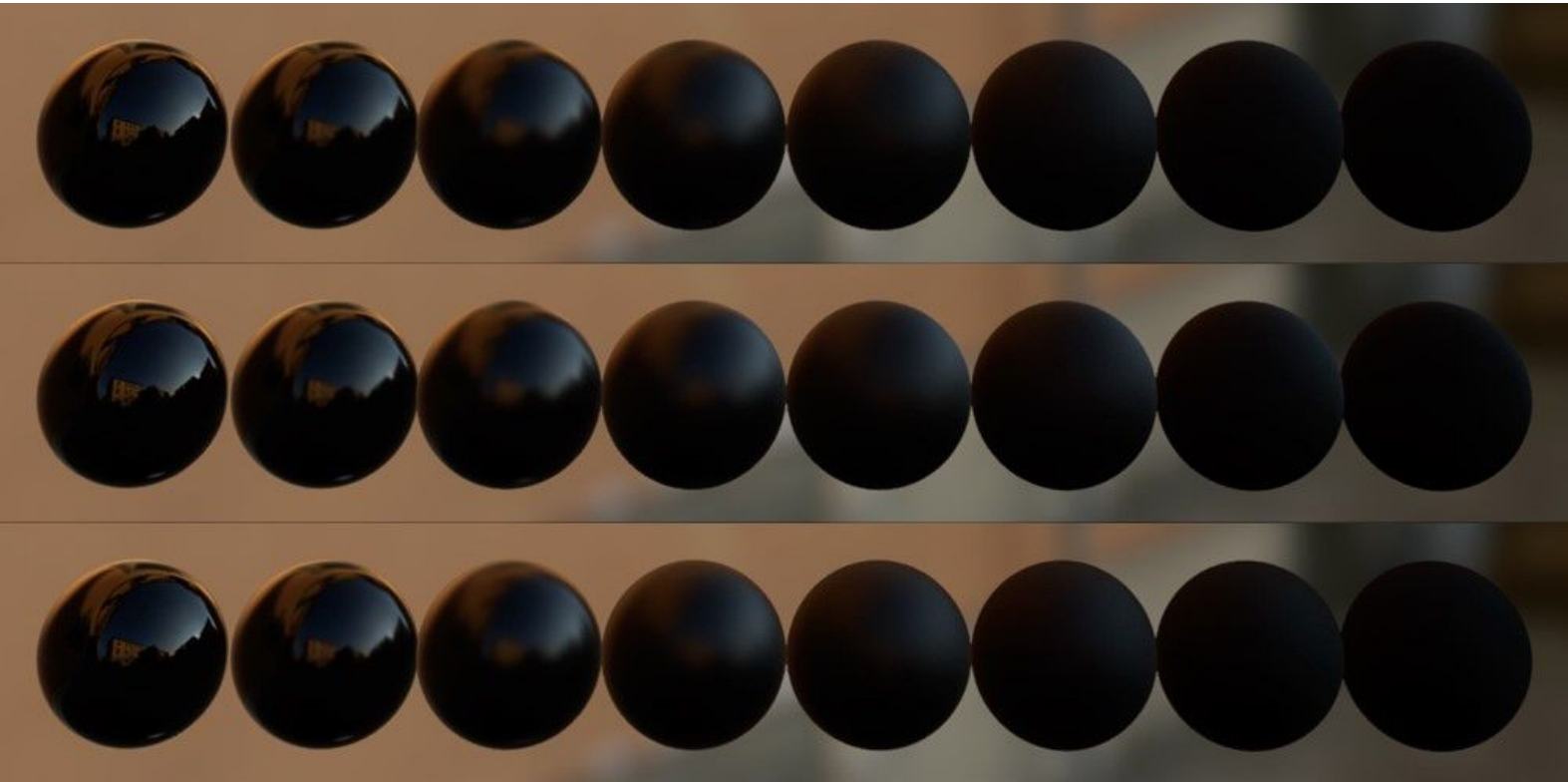


再把结果相乘，就得到了最终的 Specular 的颜色。

不同精度条件下的渲染效果



上一排是完全在 shader 里计算的，中间是按正规拆分后近似的结果(PDF 在 shader 里计算)，下面则是完全近似的方法(PDF 放在了 Pre-Filtered EnvMap 里，假设 r = n = v)



同样的近似方法，在非金属(绝缘体)上的效果比较

PS:最后这里还是想注释一下，也就是 PrefilterEnvMap 具体的生成算法，以及如何根据不同的粗糙度生成 mip，ue4 的 ppt 里并没有涉及，这个我想会在以后的文章里具体介绍吧。



另外，在 GLES2.0 的移动设备上，因为 texutre sample 最高只有 8 张，UE4 为了节省 LUT，还提供另外一种更加近似的方式，应该是参考了黑色行动 2（后面简称 ops2 吧）里的方法，在他们的 paper 里把这个叫做“ground truth”

这里还是想介绍一下这种方法，这里我们可以看到 ops2 里也做了和 ue4 类似的拆分。

ground truth 的近似

$$\left(4\int Env(l)D_{env}(h)\cos(\omega)d\omega\right)\left(\int BRDF_{env}(l,v,h)\cos(\omega)d\omega\right)$$

前面部分是 Pre-filtered Environment map，后面则是 Environment BRDF，不过他这并没有 PDF，而是直接把 D 项（Normal Distribution Function）放到前面去预计算了。

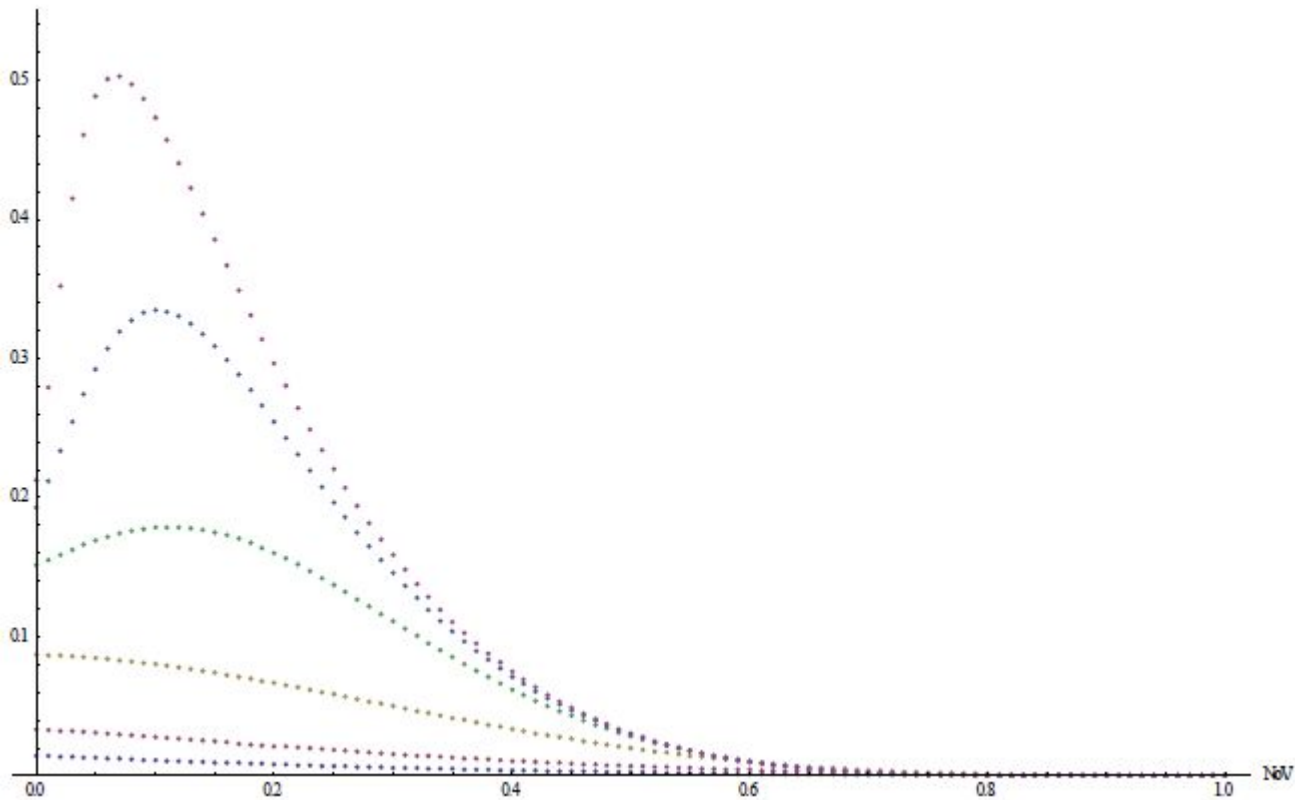
然后再说后面部分的拆分，前面提到，UE4 是通过把 Fresnel 公式的 F0 提出来，做成 F0 \* Scale + Offset 的方式，再 Scale 和 Offset 索引的存到了一张 2D LUT 上。靠 roughness 和 NoV(N dot V)来查找而 ops2 的方法，也一样是从 Schlick 的 Fresnel 公式入手来拆分(这里 rf0 和 F0 是一样的)。

$$F(l,h)=rf_0+(1-rf_0)(1-h\cdot l)^5$$

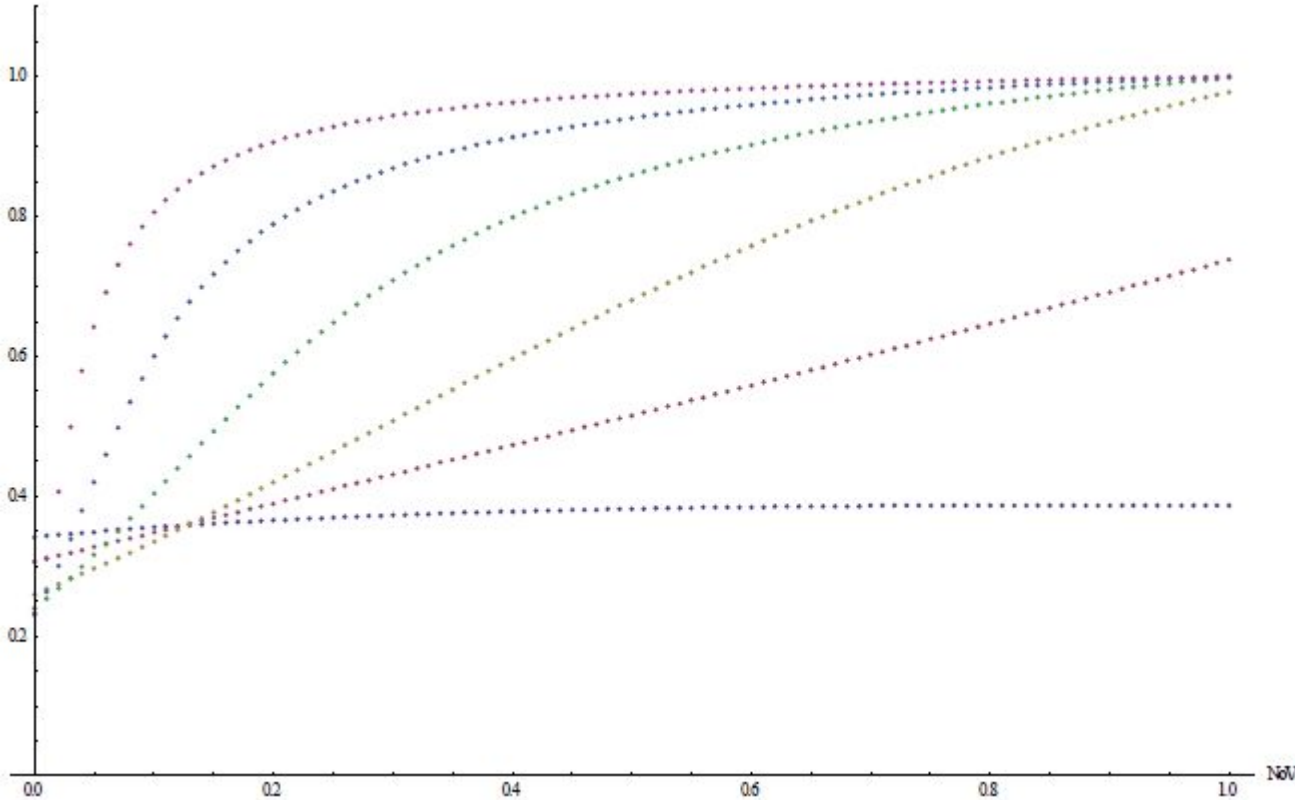
这里是从加法这里做拆分

$$\int BRDF_{env}\cos(\omega)d\omega=rf_0\int D_{env}V\cos(\omega)d\omega+(1-rf_0)\int D_{env}V(1-h\cdot l)^5\cos(\omega)d\omega$$

大概就变成了 rf0 \* a1+ (1-rf0) \* a0 的形式，这个公式很容易理解为一个关于 rf0 的一个线性插值公式。所以只要能计算出 a1 (rf0 = 1)和 a0(rf0 = 0)，就可以通过线性公式求出任意 rf0 情况下的结果了。接下来就是想办法来近似出 a0 和 a1 的曲线函数了



ground truth rf0 = 0 的曲线



ground truth  $rf_0 = 1$  的 曲线

然后他们整出两个近似的曲线函数出来



```
float a0( float g, float NoV )
{
    float t1 = 11.4 * pow( g, 3 ) + 0.1;
    float t2 = NoV + ( 0.1 - 0.09 * g );
    return ( 1 - exp( -t1 * t2 ) ) * 1.32 * exp2( -10.3 * NoV );
}

float a1( float g, float NoV )
{
    float t1 = max( 1.336 - 0.486 * g, 1 );
    float t2 = 0.06 + 3.25 * g + 12.8 * pow( g, 3 );
    float t3 = NoV + min( 0.125 - 0.1 * g, 0.1 );
    return min( t1 - exp2( -t2 * t3 ), 1 );
}
```

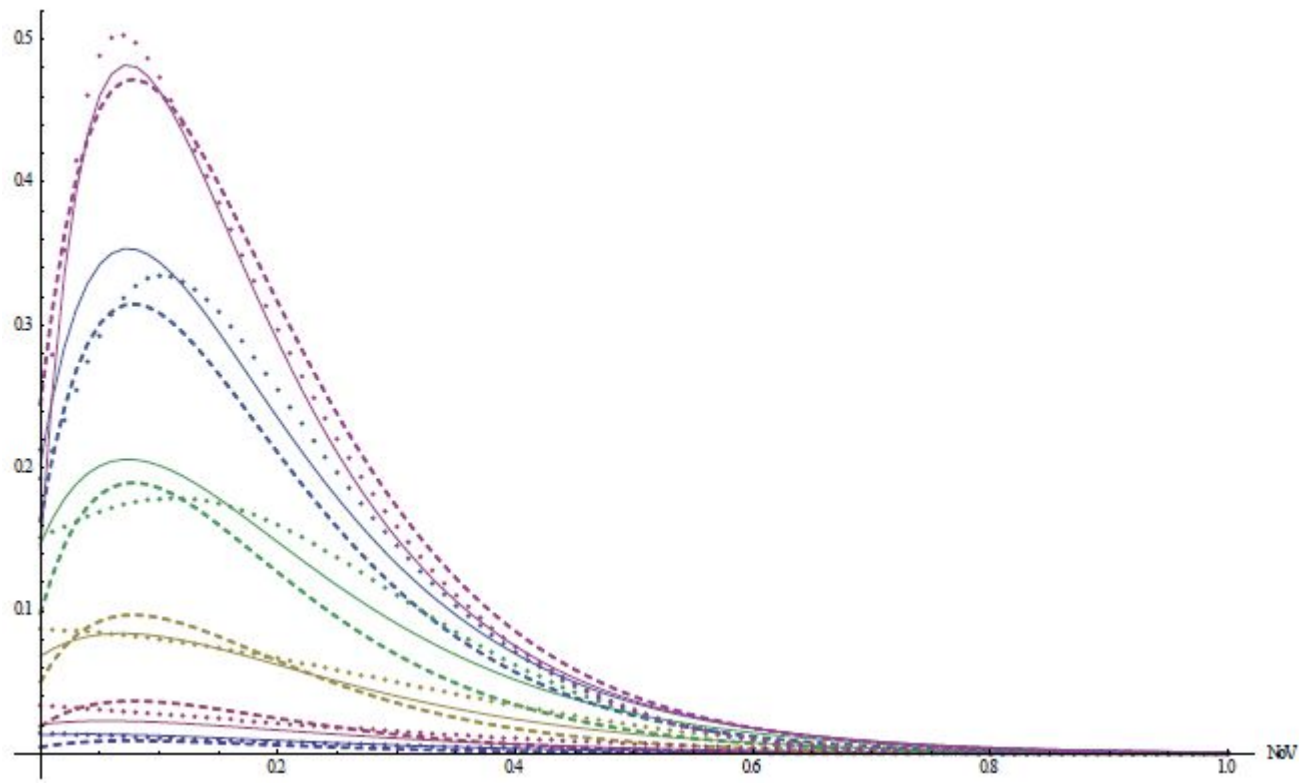


并进一步的做优化



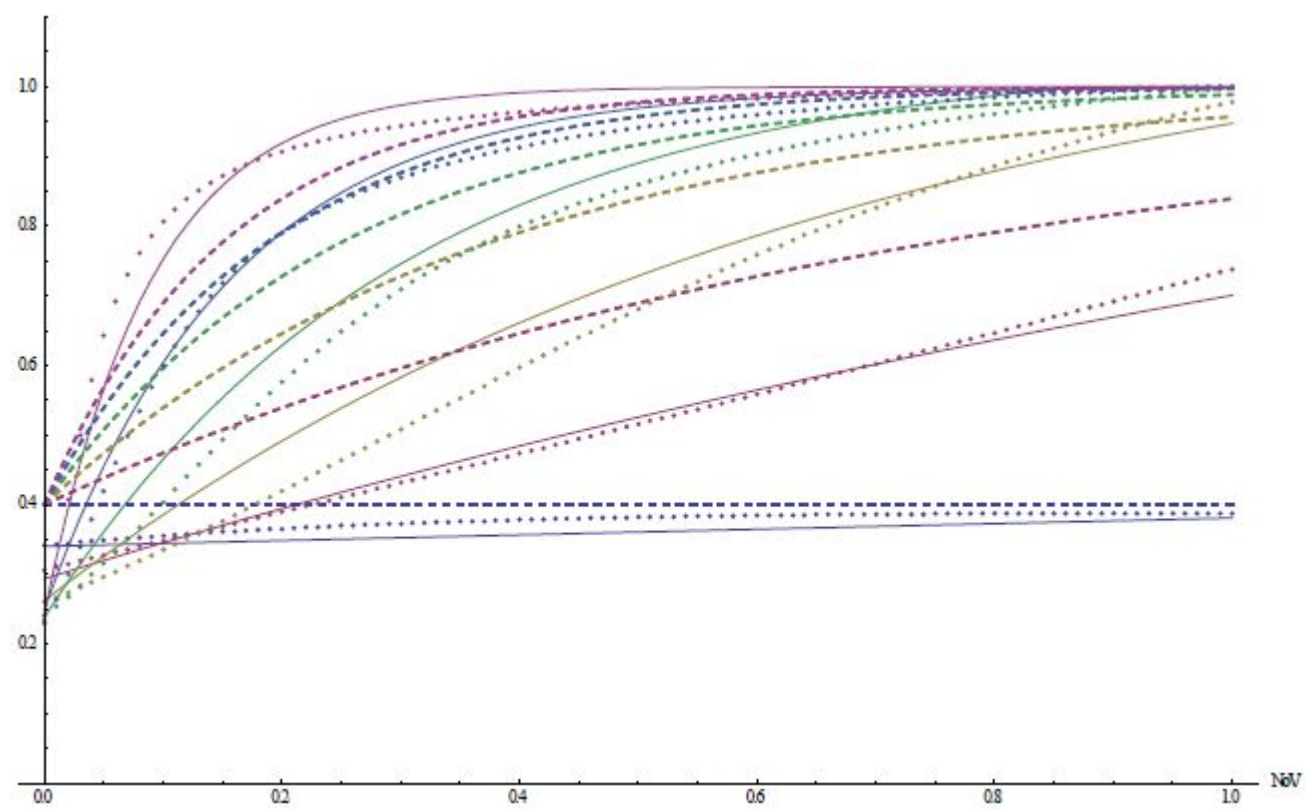
```
float a0f( float g, float NoV )
{
    float t1 = 0.095 + g * ( 0.6 + 4.19 * g );
    float t2 = NoV + 0.025;
    return t1 * t2 * exp2( 1 - 14 * NoV );
}

float a1f( float g, float NoV )
{
    float t1 = 9.5 * g * NoV;
    return 0.4 + 0.6 * ( 1 - exp2( -t1 ) );
}
```

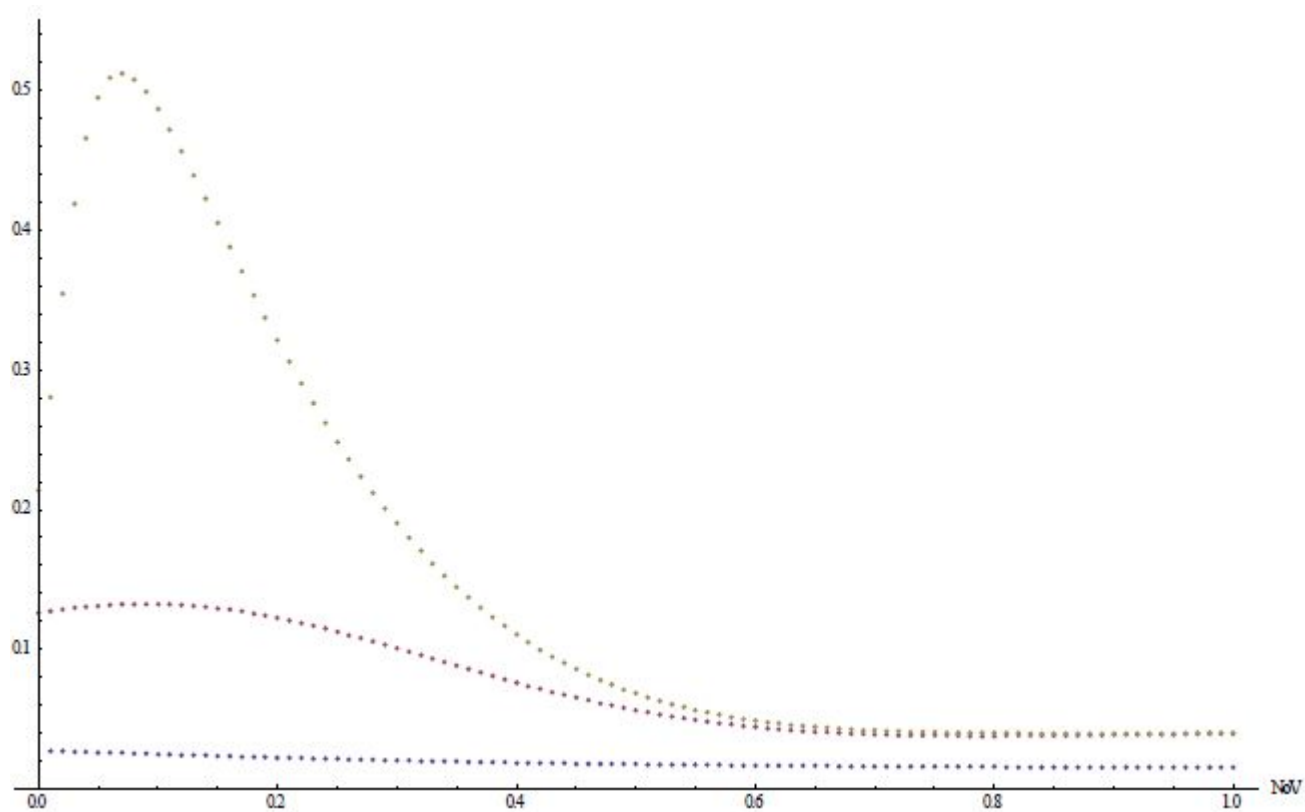


$rf_0$ (ground truth)是点线， $a_0$  是实线， $a_{0f}$  是线段



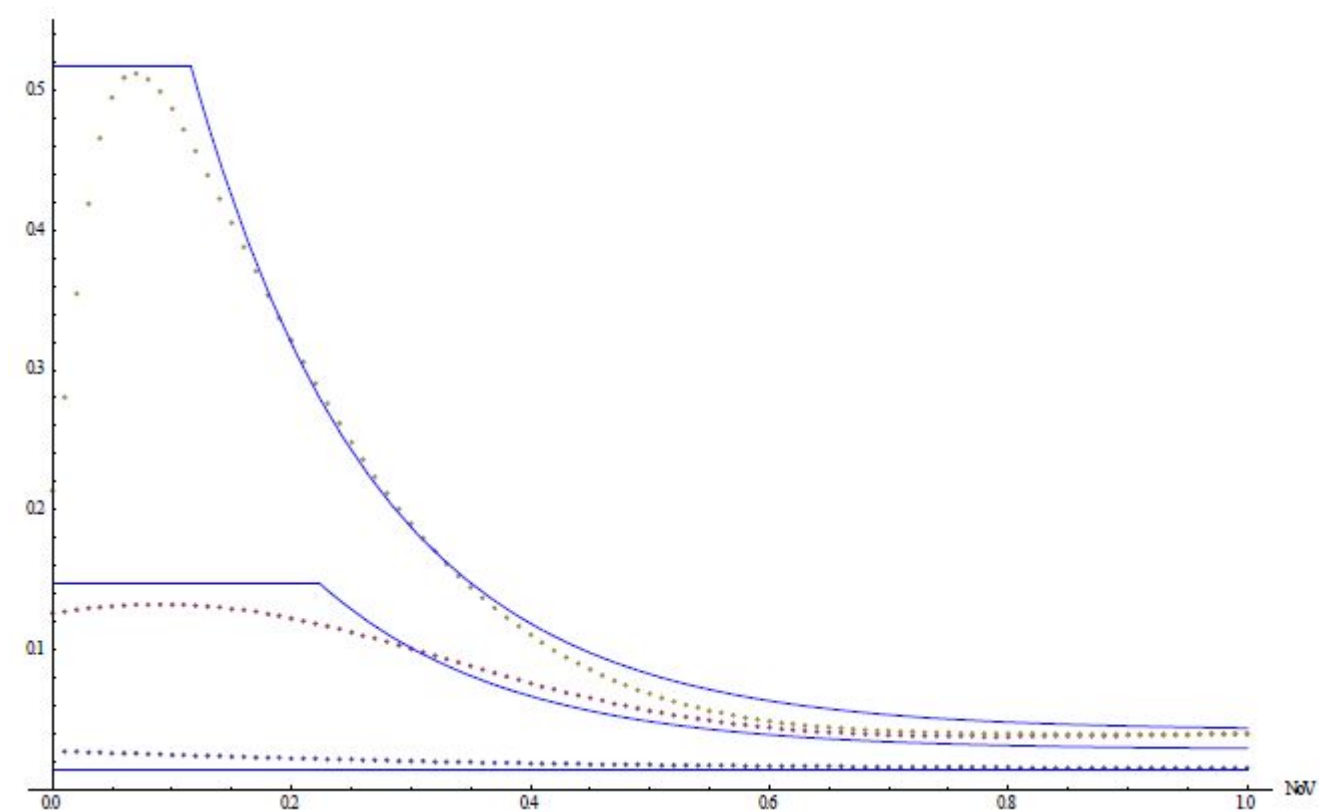


rf1(ground truth)是点线，a1 是实线，a1f 是线段  
 但这个曲线被美术人员反映环境光反射效果过亮，特别是 dielectric（电介质/绝缘体/非金属）和 gloss 低的情况。所以就把 rf0 = 0.04 这个对非金属比较通用的值，作为求 a0 的参数



ground truth rf0 = 0.04 的 曲线 不在是 a0 曲线，而叫 a004 曲线了--  
 那么，拟合出来的更廉价的 a004 曲线的公式

```
float a004( float g, float NoV )
{
    float t = min( 0.475 * g, exp2( -9.28 * NoV ) );
    return ( t + 0.0275 ) * g + 0.015;
}
```



ground truth rf0 =0.04 是点线 a004 是实线

另外，因为在游戏里，金属的使用情况并不多，也就是说  $a1(rf0 = 1)$  在实际计算插值时，贡献的参数并不是那么占主要的，所以，可以做 **a1f** 做进一步粗糙近似成 **a1vf**

```
float a1vf( float g )
{
    return 0.25 * g + 0.75;
}
```

再用 a004 和 a1vf 算出新的 a0r

```
float a0r( float g, float NoV )
{
    return ( a004( g, NoV ) - a1vf( g ) * 0.04 ) / 0.96;
}
```

至此，a0 和 a1 的最终近似版本也完成了，前面我们提到实际计算就是关于 rf0 的插值运算  
这里我们把 rf0 提出来

$rf0 * a1 + (1-rf0) * a0 = rf0 (a1 - a0) + a0$ ，那么最后的 Environment BRDF 近似公式



```
float3 EnvironmentBRDF( float g, float NoV, float3 rf0 )
{
    float4 t = float4( 1/0.96, 0.475, (0.0275 - 0.25 * 0.04)/0.96, 0.25 );
    t *= float4( g, g, g, g );
    t += float4( 0, 0, (0.015 - 0.75 * 0.04)/0.96, 0.75 );
    float a0 = t.x * min( t.y, exp2( -9.28 * NoV ) ) + t.z;
    float a1 = t.w;
    return saturate( a0 + rf0 * ( a1 - a0 ) );
}
```



OP2 的近似方法就先讲到这里了，PPT 的公式推导还是太简单，建议还是看 notebook 的吧，如果有问题可以留言给我讨论

### UE4 的 mobile PBR

接下来继续说 UE4，他的近似公式也是照抄 op2 的，  
材质为金属时的近似公式



```
half3 EnvBRDFApprox( half3 SpecularColor, half Roughness, half NoV )
{

```

```
const half4 c0 = { -1, -0.0275, -0.572, 0.022 };
const half4 c1 = { 1, 0.0425, 1.04, -0.04 };
half4 r = Roughness * c0 + c1;
half a004 = min( r.x * r.x, exp2( -9.28 * NoV ) ) * r.x + r.y;
half2 AB = half2( -1.04, 1.04 ) * a004 + r.zw;
return SpecularColor * AB.x + AB.y;
}
```



材质为非金属时的近似公式



```
half EnvBRDFAproxNonmetal( half Roughness, half NoV )
{
    // Same as EnvBRDFAprox( 0.04, Roughness, NoV )
    const half2 c0 = { -1, -0.0275 };
    const half2 c1 = { 1, 0.0425 };
    half2 r = Roughness * c0 + c1;
    return min( r.x * r.x, exp2( -9.28 * NoV ) ) * r.x + r.y;
}
```



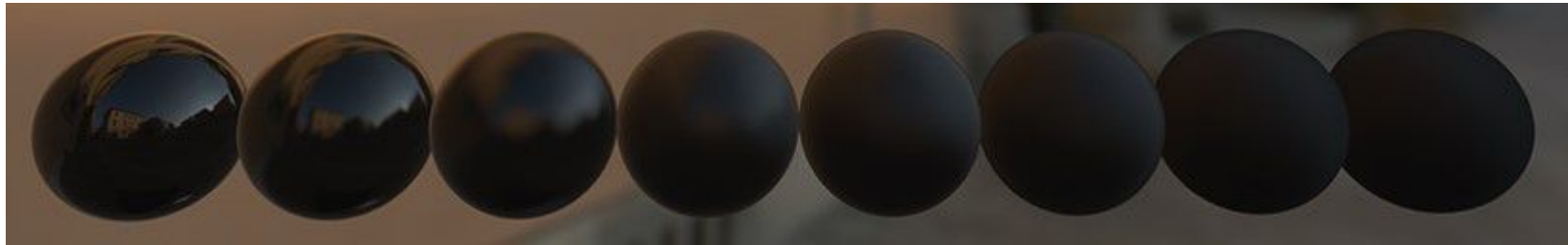
在非金属的情况下，Specular 没有颜色而只是一个亮度，这里就假设为 0.04 了

然后还可以进一步优化，就是在 roughness = 1 的时候，不在运行上面的拟合函数，而是直接给出一个拟合结果就可以了

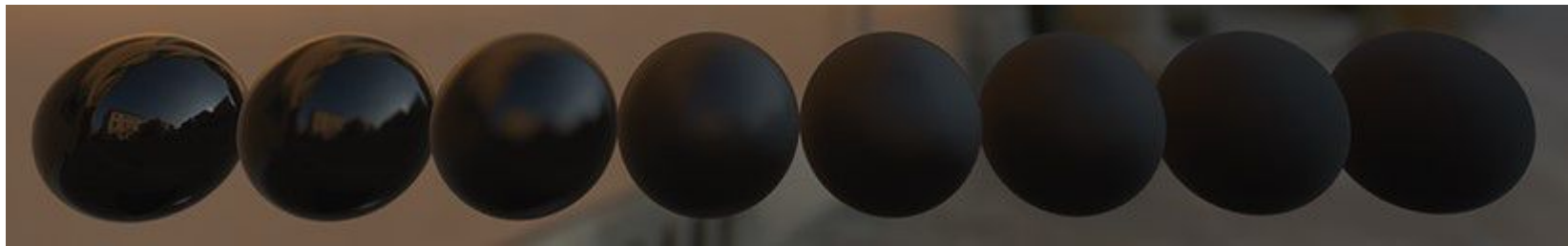
DiffuseColor+=SpecularColor\*0.45;

SpecularColor=0;

下面是和使用黑色行动 2 里的拟合方式的对比效果



使用 LUT 的



移动平台，用 ALU 拟合替代 LUT 的

最后还有一个 Directional Light 的近似，不过感觉还是光照模式说完再写好一些。

就暂时到此为止了，如果有错误还请留言或直接联系我，这里先感谢了