PBR Guide

--- 《茜色》项目组客户端工程师 文立

PBR是指基于物理的渲染(Physically Based Rendering)，它是一些基于现实世界的物理原理所构成的渲染技术的集合。PBR使用一种更加符合物理规律的方式来模拟光线，因此这种渲染方式与经典的Phong或者Blinn-Phong光照算法相比更加真实。

1. 概述

（以下主要译自THE PBR GUIDE BY ALLEGORITHMIC）。

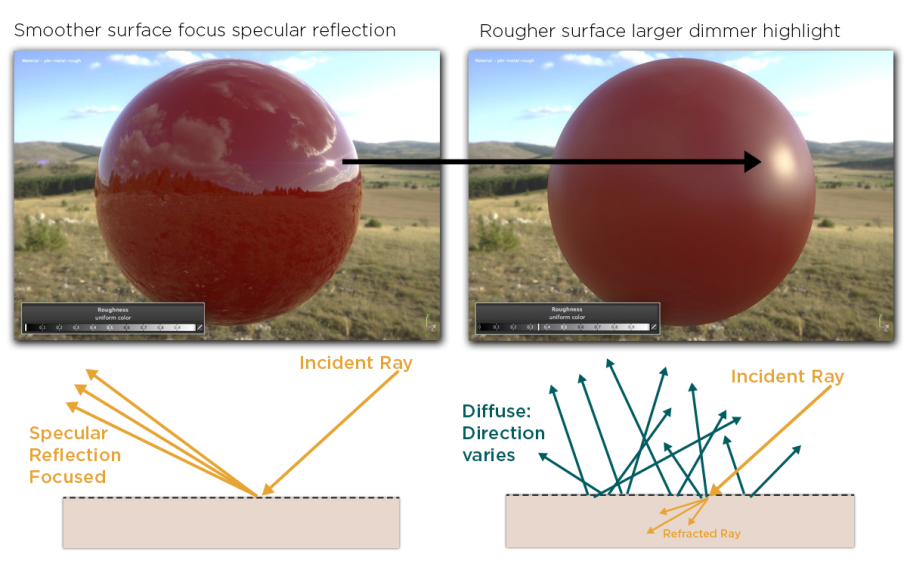
**光线**

光在相同介质中沿直线传播，当照射到物体表面时，光会发生反射和折射。当光线在各向异性的介质（即半透明物质）中传播时，光线可能会被吸收或者被散射。当光线在介质中被吸收时，光能量以其它形式能量（通常是热量）散失。光的颜色也会根据介质的吸收特性而改变，但光线传播方向不变。当光线在介质中被散射时，光线传播方向会随机变化，其变化的幅度取决于介质的性质。散射会随机改变光的传播方向但不会影响光照强度。光线在介质中传播得越远，吸收和散射带来的能量损失就越多，因此，物体的厚度在光线的吸收和散射中扮演着重要的角色。

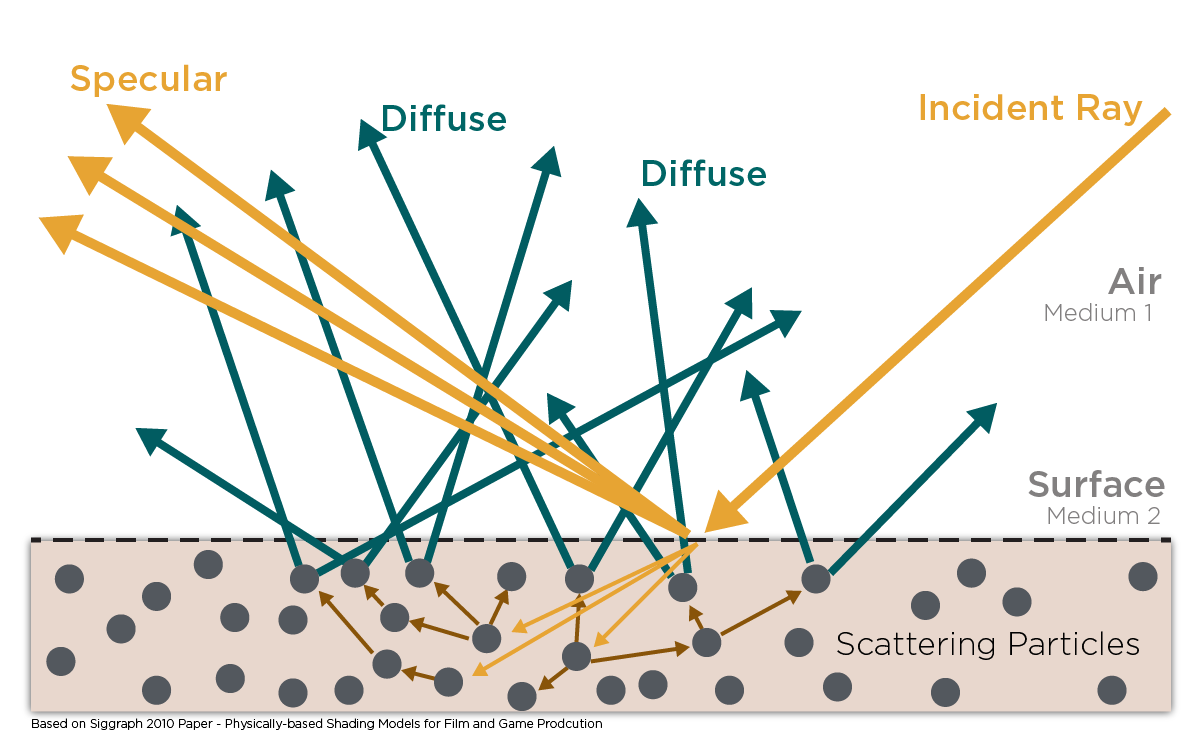
**漫反射与高光反射(Diffuse and Specular Reflection)**

高光反射是指在物体表面直接反射的光照。光线在表面被反射遵循反射定律。但是，大多数表面是不规则的，反射方向会根据表面粗糙度(roughness)随机变化。尽管光照方向变化，光照强度仍然不变。

越是粗糙的表面，高光反射范围越大，亮度越暗，而光滑表面高光反射更加集中，更亮。当然，两种表面反射的光照总量是相同的。



漫反射是经过折射的光照。光线从一种介质进入另一种介质，在物体内部发生多次散射，最后再次折射离开物体，在几乎同一入射点回到原始介质。漫反射材质往往对光线更加具有吸收性质。当折射光线在材质内部传播足够远的话，光线可能被完全吸收。这也意味着漫反射出去的光线并没有在物体中传播多远，出射点距离入射点十分接近。传统的着色模型中**Lambertian**模型并没有考虑表面粗糙度，其它漫反射模型，比如**Oren-Nayar**模型考虑了表面粗糙度的影响。

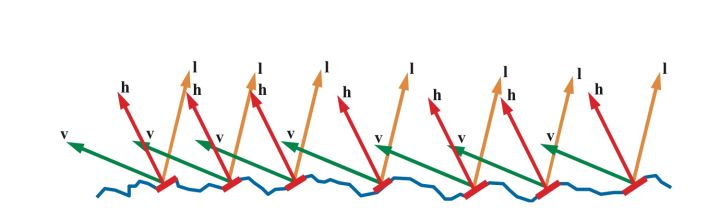


**微表面理论(Microfacet Theory)**

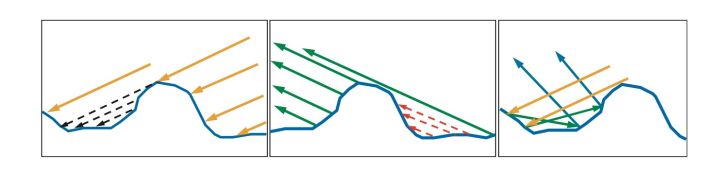
理论上说，漫反射和高光反射都依赖于光线照射点处的表面不规则性。然而实际中，表面粗糙程度对漫反射影响要小很多，主要是因为散射发生在物体内部，光线出射方向往往与入射方向和表面粗糙度无关。

在实际的PBR工作流中，这种表面不规则性可以使用一张粗糙度或者高光贴图表示。基于物理的双向反射分布函数(Bidirectional Reflectance Distribution Function, BRDF)以微表面理论为基础，其假设物体表面有无数微小的随机朝向的理想反射镜面。

表面法线正好朝向光线方向和观察方向的半角方向的微表面会反射可见光。

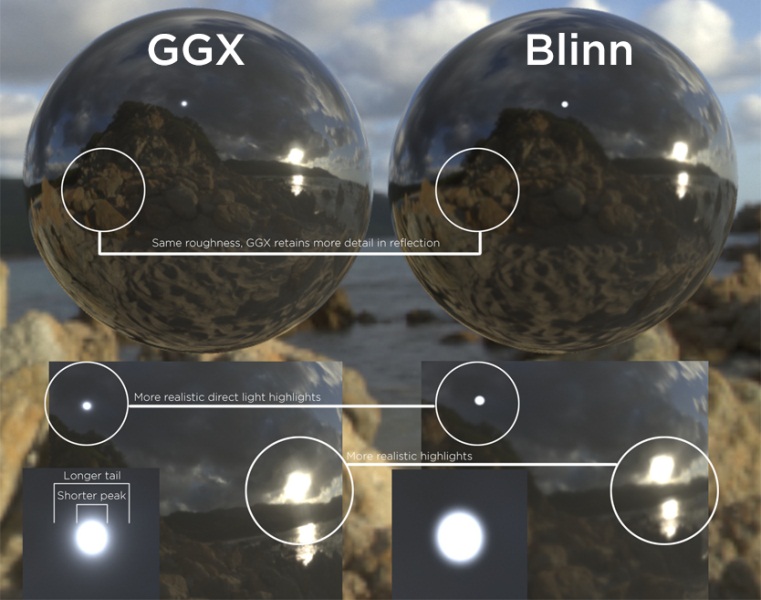


不过，即使微表面法线正好朝向半角方向，由于微表面彼此之间的遮挡和掩盖，微表面也不会反射可见光。



**BRDF**

双向反射分布函数描述了表面的反射性质。在计算机图像学中，有许多不同的BRDF模型，有些并非基于物理的。迪士尼在Siggraph 2012上提出了迪士尼准则的反射模型，该模型采用GGX微表面分布。GGX分布提供了一种较好的高光分布：在高光峰值处具有一个很小的尖峰，在衰竭时具有一个拖尾，十分接近真实效果。



**能量守恒**

在基于物理的渲染方案中，能量守恒扮演着重要的角色。表面出射的光照总量（包括反射和散射）不会超过接收的总能量。也就是说，从物体表面反射的光照强度不会高于入射光照强度。

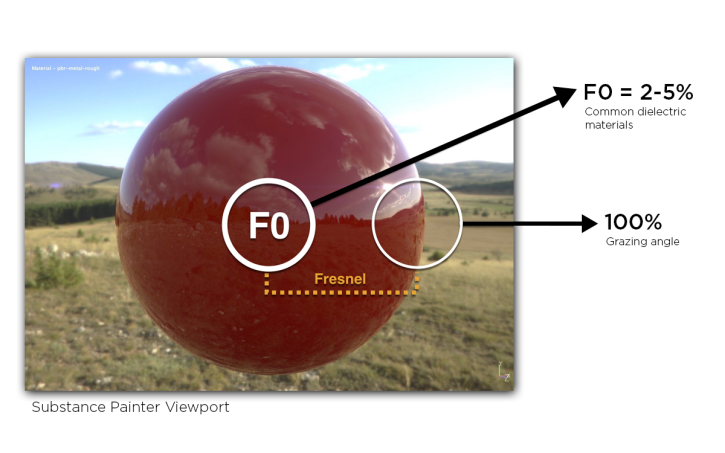
**菲涅尔效应(Fresnel Effect)**

菲涅尔反射效应在基于物理的着色模型BRDF中十分重要。菲涅尔效应是指表面反射的光照总量取决于观察角度。比如一个池塘，当你径直朝下看下池水表面，你会直接看到池底；当你以近平行于池水的角度观察池塘，你会发现池水高光反射更加强烈，几乎看不见池底。当以近90度进行观察时，所有表面几乎都是100%反射。

对于粗糙表面来说，反射会逐渐增加但不会接近100%反射。主要是因为这里的角度是每个微表面的法线和光照之间的角度，而不是宏观表面法线和光照之间角度。因此我们在宏观表面看到的实际是微表面的菲涅尔效应的平均结果。

**F0(Fresnel Reflectance at 0 degree)**

当光线垂直照射物体表面（0度入射角）时，有一部分光线被直接反弹回去称为高光。根据表面折射率(Index of Refraction, IOR)，可以获得反射的总量，称为F0(Fresnel Zero)。

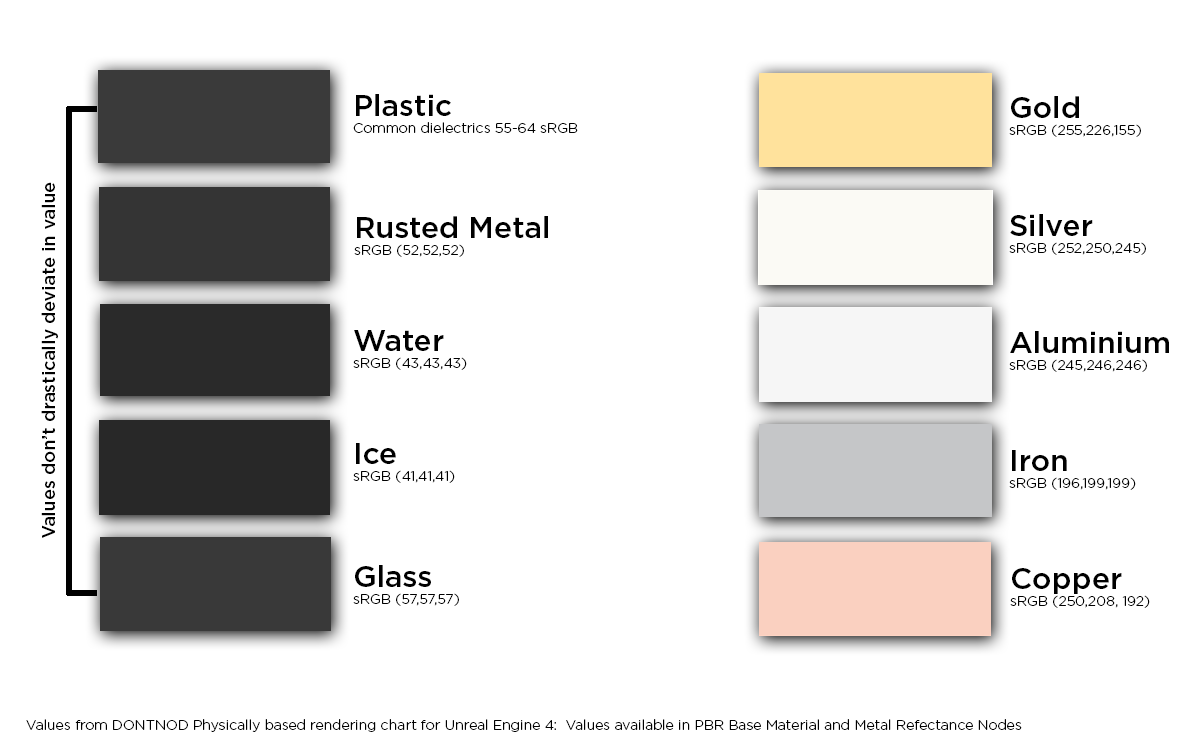


大多数绝缘体的F0范围在0.02-0.05之间，导体的F0范围在0.5-1.0之间。物体表面的反射特性与折射率之间的关系如下：



F0反射特性十分重要。非金属的F0值是一个灰度值，金属的F0值是一个RGB值。实际应用中，大部分光滑绝缘体的表面F0值在2%-5%之间，而在平视角度反射达到100%。

绝缘体的菲涅尔反射变化很小。不同的表面粗糙度对实际的反射效果影响甚微。下图展示了从非金属到金属的F0值的范围变化。



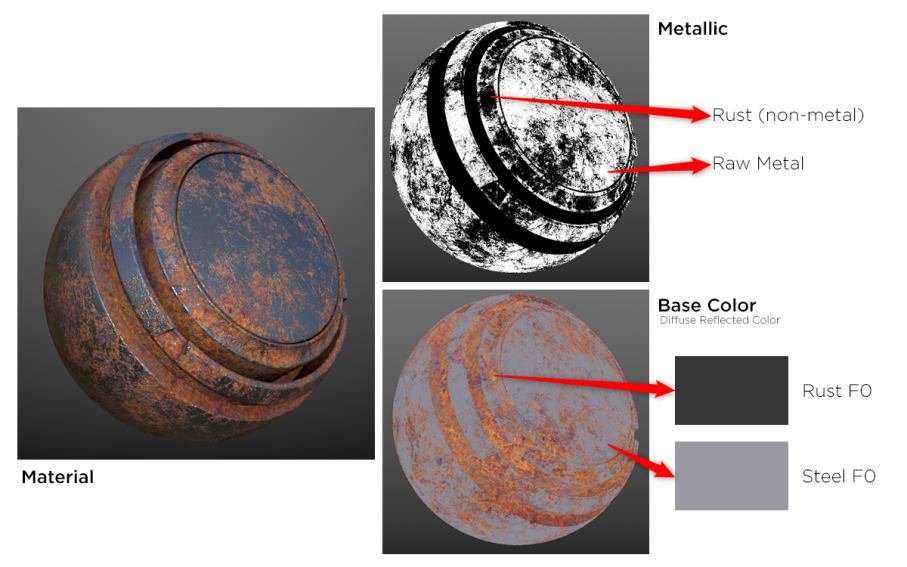
**导体和绝缘体**

在创作PBR材质时，用金属和非金属的方式去分类物体是非常有用的。

金属是热和电的良导体。金属内部的电场强度总是0，当光线照射金属时，大部分光线会被反弹而剩余的光线会被完全吸收。对于抛光金属反射率往往能达到70%到100%。

有些金属对光线波长有着选择性吸收，例如金会吸收高波长的光线比如蓝光，因此金看上去呈现黄色。由于折射的光线都被吸收，因此金属表现颜色全部来自发射的光线，所以金属没有漫反射颜色贴图。

金属可以被腐蚀，腐蚀会改变金属的发射性质，腐蚀区域通常被当做绝缘物质看待。



非金属是不良的电导体。折射光会被散射或者吸收，它会反射很少的光照，并且会有一张Albedo贴图来表示其本身的颜色。通常绝缘体的F0值在2%-5%之间，除了宝石之外大部分绝缘体的F0不会超过4%。

2. 基本理论

**光照模型**

为了模拟非光学平面的散射，有许多光照模型，下面介绍几种：

1）经验模型

经验模型并不是基于物理准则，而是更具经验公式，通过调整参数来模拟光照。

1975年Phong提出Phong反射模型：



其中，a表示环境光(Ambient)，d表示漫反射(Diffuse)，s表示高光(specular)，k表示反射系数，I表示光的颜色或者亮度，α模拟表面粗糙程度，值越小越粗糙，值越大越光滑。

1977年Blinn对Phong模型进行修改，提出了经典的Blinn-Phong反射模型：



其中，半角向量H是光照方向L和观察方向V的中间向量：。

Blinn-Phong模型运算简单，适合早期硬件实现，但是毕竟只是一个经验模型，表现力有限。

2）基于物理的分析模型

1981年Cook-Torrance提出了基于物理的反射模型，被广泛采用。

**反射方程**

在表面半球上，某点的入射光照来自于整个半球范围内光照，需要考虑各个方向的光照贡献：



其中，表示某个无限小的立体角在某个点上的辐射率，而立体角可以视作是入射方向向量。

发射方程计算了点p在方向上被反射出来的辐射率的总和。

**BRDF**

BRDF接受入射方向，出射方向，平面法线n以及一个用来表示微平面粗糙度的参数a作为函数的输入参数。BRDF可以近似求出每束光线对一个给定材质属性的平面上最终反射出来的光线的贡献程度。

Cook-Torrance BRDF兼有漫反射和高光反射2个部分：



其中，是入射光线中被折射部分的能量所占的比例，而是被反射部分的比例。BRDF的左侧表示的是漫反射部分，使用表示，称为Lambertian漫反射，用如下公式表示：



除了Lambertian漫反射之外，常见的漫反射模型还有Oren-Nayar漫反射，Disney漫反射等。后面2种漫反射都考虑了表面粗糙度的影响，而Lambertian漫反射没有考虑。不过Lambertian漫反射已足够满足实际需求，并且计算十分简单。

BRDF高光反射是研究的重点。它的形式如下：



Cook-Torrance BRDF的高光反射部分包含3个函数，分母部分包含一个标准化因子。3个函数分别为D-法线分布函数(Normal Distribution Function), F-菲涅尔反射函数(Fresnel Function)，G-几何函数(Geometry Function)：

法线分布函数：估算在表面粗糙度影响下，微表面法向向量与半角向量一致的微表面数量。

几何函数：描述了微表面彼此之间的遮挡掩盖效应。当一个平面比较粗糙是，平面表面上的微平面可能挡住其它微平面从而减少有效反射光线。

菲涅尔反射函数：描述在不同观察角度下反射光线所占的比例。

法线分布函数D和几何函数G的乘积一起决定了有效反射光照。

**菲涅尔反射函数F**

菲涅尔反射函数可以用Fresnel-Schlick近似法求得近似解：



其中，F0是平面的基础反射率。

计算机图形学中广泛采用了Schlick菲涅尔函数。

导体材质表面的基础反射率一般在0.5和1.0之间变化，此外，对于导体或者金属表面而言基础反射率一般是带有色彩的，因此F0要用RGB表示。

正是由于金属和非金属表面所独有的特性引出了金属工作流(Metallic Workflow)的概念，我们需要额外使用一个金属度(Metalness)来编写表面材质。

通过预先计算绝缘体与导体的F0值，可以对两种类型的表面使用相同的Fresnel-Schlick近似。可以这样实现：

*float3 F0 = float3(0.04, 0.04, 0.04);*

*F0 = lerp(F0, albedo, metalness);*

**法线分布函数D**

法线分布函数表示表面法线的统计分布，其满足归一性：



这个性质十分重要，后面还会用到。

法线分布函数是一个标量，它决定了高光的大小，亮度和形状。随着表面粗糙度降低，与表面法线朝向一致的微表面逐渐增大，法线分布函数会变得很大。

在图形领域有几种常见的法线分布函数：

* Blinn
* Beckman
* GGX

Blinn法线分布函数



Beckman法线分布函数



GGX法线分布函数



实际应用中，GGX在高光峰值处具有一个很小的尖峰，在衰竭时具有一个拖尾，十分接近真实效果，而被广泛使用。



实际chrome数据，GGX，Beckmann模拟

**几何函数G**

几何函数从统计学上近似微平面间相互遮蔽的比例，这种遮蔽会减少有效反射光照。与法线分布函数类似，几何函数采用一个表面的粗糙度作为参数，表面越粗糙微平面间的遮蔽效果越明显。

在图形领域有几种常见的几何函数：

Implicit几何函数



Neumann几何函数



Cook-Torrance几何函数



Kelemen几何函数



Schlick-Beckmann几何函数





UE4中使用的几何函数是GGX与Schlick-Beckmann近似的结合体，因此又称Schlick-GGX



其中，k是a基于几何函数是针对直接光照还是针对IBL光照的重映射(Remapping)。

为了有效地估算几何部分，需要将观察方向（几何遮蔽(Geometry Obstruction)）和光照方向（几何阴影(Geometry Shadowing)）都考虑进去。可以使用史密斯法把两者纳入其中：



已经介绍了Cook-Torrance BRDF中所有元素，最终的反射方程如下：



**直接光照**

对于场景中的每个光源（平行光，点光源或聚光灯），BRDF中的积分可以直接去掉，因为我们已经知道所有贡献光源的位置，每个光源仅有一个方向上的光线会影响物体表面的辐射率，对物体表面上的一个点的着色不需要求解积分。只需要遍历所有有贡献的光源，计算它们总的辐照度即可。

这里计算相对简单，易于实现。

需要注意的是，因为计算是基于物理的，所有的输入都应是线性的，所有的计算都应在线性空间，否则会得到不正常的光照，如果是在Gamma空间需要进行Gamma校正。同时，因为现实世界的光照范围变化很大，最后计算得到的颜色在色谱上也会有比较大变化空间，使用默认的LDR(Low Dynamic Range)会产生截断，所以需要进行色调映射(Tonemapping)，将HDR(High Dynamic Range)值映射到LDR值。

（以下主要译自LearnOpgnGL.com PBR部分）

**IBL**

IBL，基于图像的光照(Imaged Based Lighting)，是一组照明物体的技术，与直接光照不同，其将周围环境视作一个巨大的光源。这一般是通过使用立方体贴图(Cubemap) 计算渲染方程，每个贴图元素作为一个光源。通过这种方式，可以获得周围环境的全局光照，提升真实感。基于图像的光照以周围环境为输入可以更准确地反映环境光照。

将反射方程分为Diffuse和Specular两个部分，分别计算。

**Diffuse Irradiance**

采用Lambertian漫反射，漫反射积分表示为



可以看到，积分只与有关。我们可以预先计算一个立方体贴图存储每个采样方向的漫反射积分。这个立方体贴图称为Irradiance map。

为了求解间接光照漫反射积分，从表面半球内的所有方向对环境光照进行采样，然而我们只能尽可能多地离散采样，采样点均匀分布在半球表面，然后对所有采样结果进行平均。

漫反射积分进一步表示为



伪代码如下

*float3 irradiance = float3(0, 0, 0);*

*float3 up = float3(0, 1, 0);*

*float3 right = cross(up, normal);*

*up = cross(normal, right);*

*float samleDelta = 0.025;*

*float nSamples = 0;*

*for(float phi = 0; phi < 2\*pi; pi += sampleDelta)*

*{*

*for(float theta = 0; theta < 0.5\*pi; theta += sampleDela)*

*{*

*// spherical to cartesian (in tangent space)*

*float3 tangentSample = float3(sin(theta) \* cos(phi), sin(theta) \* sin(phi), cos(theta));*

*// tangent space to world space*

*float3 sampleVec = tangentSample.x \* right + tangentSample.y \* up + tangentSample.z \* normal;*

*irradiance += texCUBE(environmentMap, sampleVec).rgb \* cos(theta) \* sin(theta);*

*++nSamples;*

*}*

*irradiance = pi \* irradiance \* (1 / nSamples);*

*}*

irradiance map代表了表面半球上所有间接光照的漫反射积分。求出irradiance map之后，可以对其采样直接获得环境漫反射积分。

*float3 irradiance = texCUBE(irradianceMap, N).rgb;*

*float3 diffuse = irradiance \* albedo;*

*float3 ambient = kD \* diffuse;*

需要注意，因为环境光来自表面半球内所有方向，无法找到一个半角向量求解Fresnel反射函数。因此，我们利用表面法线和观察方向之间的夹角来计算Fresnel函数，同时考虑表面的粗糙度，将Fresnel反射函数修改为

*float3 FresnelScklickRoughness (float3 F0, flaot cosTheta, float roughness)*

*{*

*return F0 + (max(float3(1.0 - roughness), F0) - F0) \* pow(1.0 - cosThea, 5.0);*

*}*

**Specular IBL**

高光反射积分为



可以看到Cook-Torrance高光反射积分决定于入射光照方向和观察方向。Epic Games提出了一种解决方案，可以实时对高光部分进行预卷积，进行分割近似。

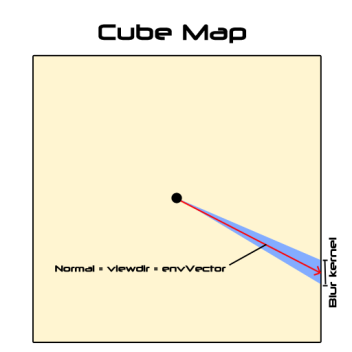
分割近似将反射方程分成2个独立的部分，分别对其进行卷积，然后将2个结果融合进高光反射。分割近似方程为



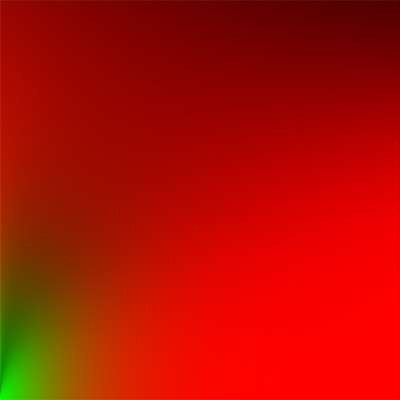
第一部分称为pre-filtered environment map，它与irradiance map类似，将预计算的环境光照积分存为一个立方体贴图，不过这次需要考虑表面粗糙度。物体表面越粗糙，高光反射越模糊。我们在pre-filtered map的mipmap中有序存储各级粗糙的结果。比如，一个5级mipmap中存储5个不同粗糙度的pre-filtered map如图



我们根据Cook-Torrance BRDF中的法线分布函数(NDF)生成采样向量。但是并不知道表面法线信息和观察方向，因此假定观察方向，法线都等于输出采样方向。



第二部分为反射积分中的BRDF部分。我们可以给定输入表面粗糙度roughness和光照与表面法线之间的夹角预计算BRDF响应。Epic Games将预计算的各个粗糙度下各个法线方向和光照方向组合的BRDF响应存储到一种2D查找表(Lookup Texture, LUT)里，称为BRDF积分贴图(BRDF Integration Map)。这张2D查找表输出一个相对于表面菲涅尔响应的缩放系数（红色）和偏移值（绿色），最终得到反射方程的第二部分。



通过BRDF积分贴图和预滤波环境贴图，我们可以算出高光反射积分：

*float lod = GetMipLevelFromRoughness(roughness);*

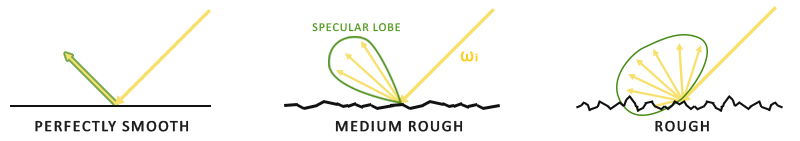
*float3 prefilteredColor = texCUBELod(prefilteredEnvMap, refVec, lod);*

*float2 envBRDF = tex2D(BRDFIntegrationMap, float2(NdotV, roughness)).xy;*

*float3 indirectSpecular = prefilteredColor \* (F \* envBRDF.x + envBRDF.y);*

**Pre-filtering environment map**

之前计算irradiance map的时候，对环境贴图的采样采用的是半球上均匀分布的采样向量，但是对于高光反射来说，由于表面粗糙度的影响，反射光线在一个反射瓣内。



我们可以将反射瓣看作给定入射光照下，围绕半角向量随机朝向的反射光线。因此主要对反射瓣内的入射光线进行采样可以提高采样效率，这个过程称为重要性采样(Importance Sampling)。

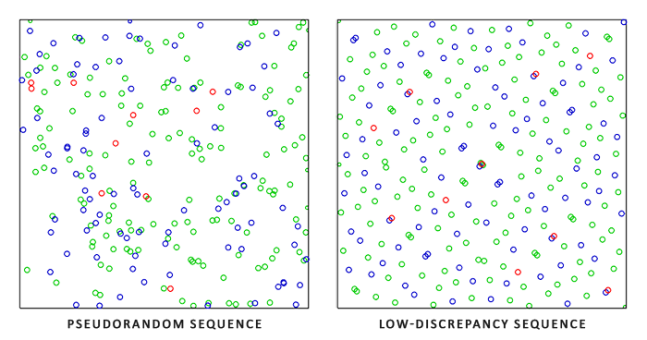
**蒙特卡洛积分和重要性采样**

蒙特卡洛积分是一种统计模拟方法，通过N个样本来估计积分结果。N个样本是根据概率分布函数在积分域上随机选择，并求其平均值。



我们根据采样的点概率密度函数进行相应加权，计算积分。当较小时，意味着使用均匀采样时需要更多的采样点；当较大时，意味着使用均匀采样需要更少的采样点。为了弥补这一性质，我们将每个采样点乘以的倒数（《GPU Gems3 GPU-Based Importance Sampling》）。

我们采用低差异序列产生随机采样点，它产生的随机点更加均匀。



光照的高光反射，反射向量限制在反射瓣内，反射瓣的大小取决于表面粗糙度。重要性采样集中在限制区域进行采样，效率更高。

我们采用反演法(The Inversion Method)计算重要性采样。反演法是将一个或多个均匀分布的随机变量，映射为期望分布的随机变量过程。可以通过下列步骤有概率密度函数获取随机值。

* 计算累积分布函数:
* 计算累积分布函数的反函数:
* 在[0, 1]区间生成均匀分布的随机变量
* 将代入，求出满足概率密度对应的随机数：

我们使用的BRDF的GGX法线分布函数D为



因为法线分布函数满足归一性：



对于重要性采样，一般选择概率密度函数为。需要注意，这个概率密度函数是相对于半角向量的，对于入射光照向量的概率密度函数是



在球坐标系下表示为



通过反演法求得采样角度为





其中，，为[0, 1]上的均匀分布。

**低差异序列**

我们采用Hammersley序列作为低差异采样序列。N个采样点的Hammersley序列可以这样生成：

*float RadicalInverse\_Vdc (int bits)*

*{*

*bits = (bits << 16u) | (bits >> 16u);*

*bits = ((bits & 0x55555555u) << 1u) | ((bits & 0xAAAAAAAAu) >> 1u);*

*bits = ((bits & 0x33333333u) << 2u) | ((bits & 0xCCCCCCCCu) >> 2u);*

*bits = ((bits & 0x0F0F0F0Fu) << 4u) | ((bits & 0xF0F0F0F0u) >> 4u);*

*bits = ((bits & 0x00FF00FFu) << 8u) | ((bits & 0xFF00FF00u) >> 8u);*

*return float(bits) \* 2.3283064365386963e-10; // / 0x100000000*

*}*

*float2 Hammersley (int i, int N)*

*{*

*return float2(float(i)/float(N), RadicalInverse\_Vdc(i));*

*}*

**GGX重要性采样**

根据之前的计算，GGX重要性采样可以实现为

*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);*

*// 生成H向量*

*float3 H;*

*H.x = sinTheta \* cos(phi);*

*H.x = sinTheta \* sin(phi);*

*H.x = cosTheta;*

*float3 upVector = abs(N.z) < 0.999 ? float3(0, 0, 1) : float3(1, 0, 0);*

*float3 tangent = normalize(cross(upVector, N));*

*float3 bitangent = cross(N, tangent);*

*// tangent space to world space*

*return tangent \* H.x + bitangent \* H.y + N \* H.z;*

*}*

生成Pre-Filtered Environment Map就很容易了

*float3 PrefilteredEnvMap (float roughness, float3 R)*

*{*

*// 假定R(环境贴图输出方向，wo),N,V相同*

*float3 N = R;*

*float3 V = R;*

*float3 prefilteredColor = 0;*

*const uint NSamples = 1024;*

*float totalWeight = 0;*

*for(int i = 0; i < NSamples; ++i)*

*{*

*float2 Xi = Hammersley(i, NSamples);*

*float3 H = ImportanceSampleGGX(Xi, roughness, N);*

*float3 L = 2 \* dot(V, H) \* H - V; // 光线入射方向*

*float NdotL = saturate(dot(N, L));*

*if (NdotL > 0)*

*{*

*// 会有近似误差,乘以NdotL效果更好(Epic Games, <<Real Shading in Unreal Engine 4>>)*

*prefilteredColor += texCUBE(envMap, L).rgb \* NdotL;*

*totalWeight += NdotL;*

*}*

*}*

*return prefilteredColor / totalWeight;*

*}*

**Pre-computing the BRDF**

然后我们开始计算分割近似的第二部分：BRDF部分，其方程为



通过一系列展开和变形可以得到



上述结果可以理解为对F0的缩放和偏移，注意已经包含F项，相除会直接消去。

采用之前计算环境贴图的方法，根据表面法线和观察方向之间的夹角和表面粗糙度roughness与计算BRDF积分，并将结果存到一张纹理上，称为BRDF积分贴图(BRDF Integration Map)。

生成BRDF积分贴图的代码如下

*float2 IntegrateBRDF (float NdotV, float roughness)*

*{*

*float3 V;*

*V.x = sqrt(1 - NdotV \* NdotV);*

*V.y = 0;*

*V.z = NdotV;*

*float A = 0;*

*float B = 0;*

*float3 N = float3(0, 0, 1);*

*const uint NSamples = 1024;*

*for (int i = 0; i < NSamples; ++i)*

*{*

*float2 Xi = Hammersley(i, NSamples);*

*float3 H = ImportanceSampleGGX(Xi, N, roughness);*

*float3 L = normalize(2 \* dot(V, H) \* H - V);*

*float NdotL = saturate(L.z);*

*float NdotH = saturate(H.z);*

*float VdotH = saturate(dot(V, H));*

*if (NdotL > 0)*

*{*

*float G = GeometrySmith(N, V, L, roughness);*

*// microfacet specular = D \*G \* F / (4 \* NdotL \* NdotV)*

*// pdf = D \* NdotH / (4 \* VdotH)*

*// ==> microfaced specular \* NdotL / F / pdf = G \* VdotH / (NdotV \* NdotH)*

*float G\_Vis = (G \* VdotH) / (NdotH \* NdotV);*

*float Fc = pow(1 - VdotH, 5);*

*A += (1 - Fc) \* G\_Vis;*

*B += Fc \* G\_Vis;*

*}*

*}*

*return float2(A, B) / NSamples;*

*}*

这样，我们就有了预滤波环境贴图(Pre-filtered Environment Map)和BRDF二维查找表(BRDF 2D LUT)，可以根据分割求和近似重建间接高光积分。

*float3 ApproximateSpecularIBL (float3 specularColor, float roughness, float3 N, float3 V)*

*{*

*float NdotV = saturate(dot(N, V));*

*float3 R = 2 \* dot(V, N) \* N - V;*

*float3 prefilteredColor = PrefilterEnvMap(roughness, R);*

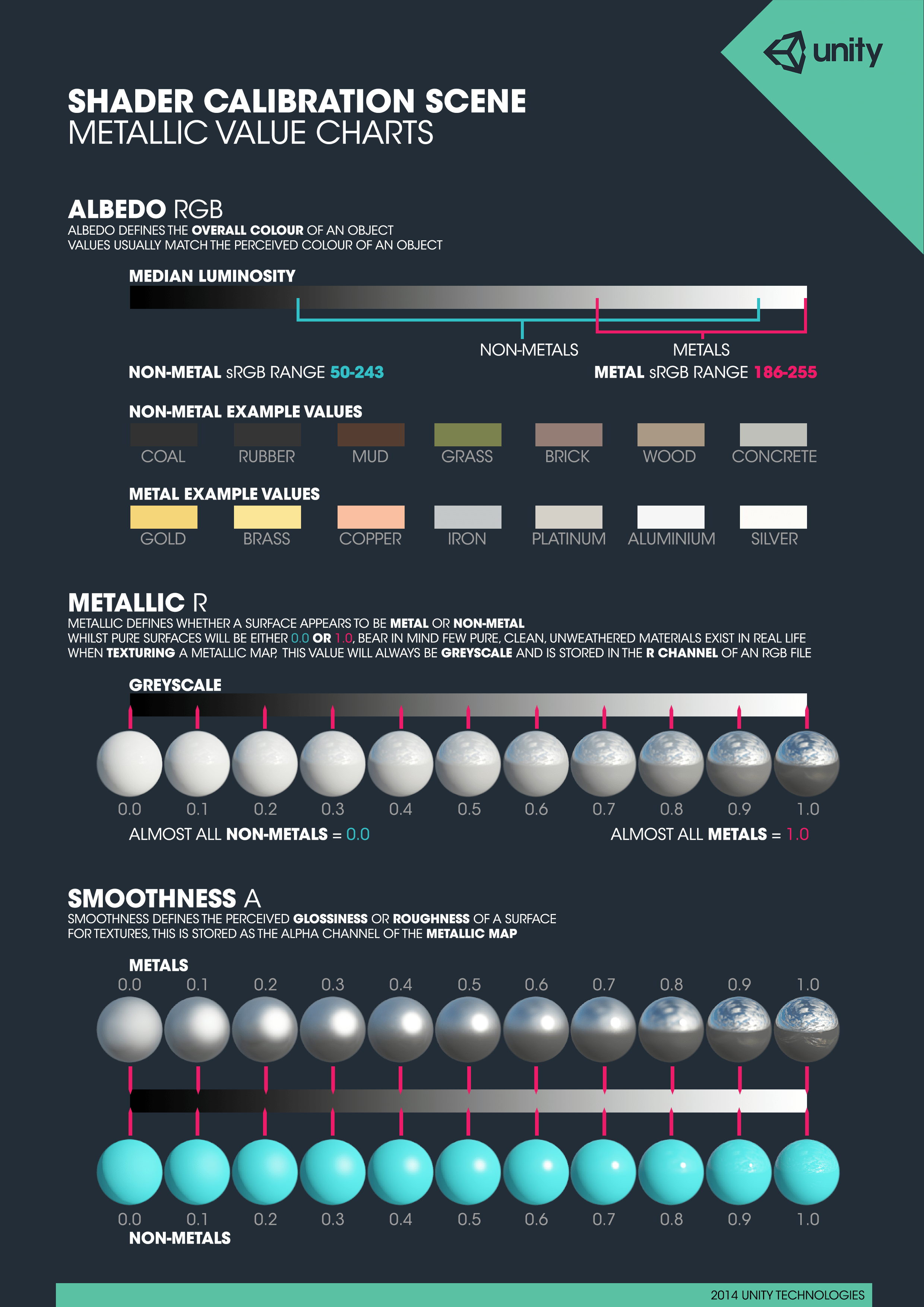
*float2 envBRDF = IntegrateBRDF(NdotV, roughness);*

*return prefilteredColor \* (specularColor \* envBRDF.x + envBRDF.y);*

*}*

3. Unity标准材质输入参数

Standard.shader是Unity内置标准着色器，使用金属工作流(Metallic workdflow)。其有多个纹理插槽，其中最重要的是Albedo, Metallic, Normal贴图。



**Albedo**

Albedo代表物体表面的颜色，贴图上不包含任何光照阴影。

**Normal**

法线贴图代表每个片段的法线朝向，调整表面光照效果，可以极大地提升表现效果，它一般定义在切线空间。

**Metallic**

Metallic代表每个片段的金属性，它是一个灰度值。在Unity中，metallic贴图一般是2张贴图的组合，R通道通常表示metallic，A通道通常表示smoothness。材质通过metallic值确定是金属还是非金属，金属表面反射更强，色彩更强，非金属表面反射偏弱，色彩偏暗。

smoothness表示表面的粗糙程度。粗糙表面反射更发散更模糊，光滑表面发射更集中更清晰。程序中一般使用roughness计算，roughness=1-smoothness。

4. Unity标准着色器

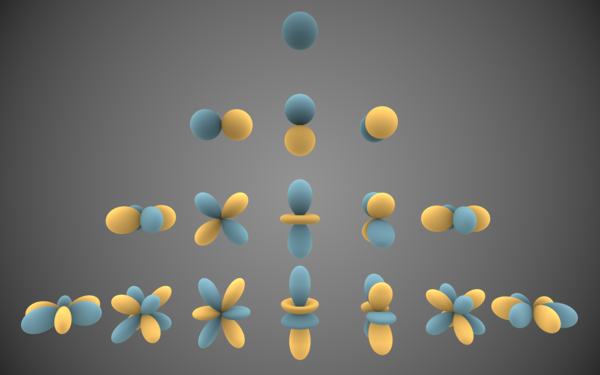
**Unity GI(Global Illumination)**

简要介绍一下Unity中的全局光照，有助于更好地理解标准着色器。这一部分不是特别清晰，后面还会继续深入理解。

全局光照能够捕捉间接光照，增加环境光源对材质的影响。在Unity中，有2种不同的技术来计算全局光照，烘焙全局光照(Baked GI)和预计算实时全局光照(Precomputed Realtime GI)。烘焙GI和预计算实时GI都只针对静态模型。

当烘焙一张光照贴图(Lightmap)时，场景内静态对象会基于光的影响生成一张贴图，并迭在场景对象之上建立照明效果。烘焙光照产生的贴图无法再运行时改变，因此被定义为静态的。

使用光照探头(Light Probe)可以使非静态模型得到周围静态模型的辐射光。其核心是利用球谐函数对球面亮度信号编码和重建



在Unity中，烘焙GI 的LightProbe采用了3阶球谐函数（9个参数），实时GI中的LightProbe采用了2阶球谐函数（4个参数）

**Standard.shader**

Unity标准着色器Standard.shader包含7个CG头文件，分别是

UnityStandardBRDF.cginc—主要存放着色器BRDF相关函数

UnityStandardConfig.cginc—主要存放着色器配置相关代码

UnityStandardCore.cginc—主要存放标准着色器主要代码（如顶点、片段着色器等相关函数）

UnityStandardInput.cginc—主要存放着色器输入结构的函数

UnityStandardMeta.cginc—主要存放着色器meta通道相关函数

UnityStandardShadow.cginc—主要着色器阴影贴图采样相关函数

UnityShaddardUtils.cginc—主要存放着色器共用的一些工具函数

此外，Unity还为标准着色器自定义了编辑器UI界面，存于StandardShaderGUI.cs文件中。

Standard.shader中定义了标准着色器输入相关的属性，里面有5个pass，分别是ForwardBase, ForwardAdd, ShadowCaster, Deffered和Meta通道。ForwardBase通道处理主平行光源，我们主要分析这个通道。

**vertForwardBase**

vertForwardBase顶点函数的输入结构为VertexInput，主要包括顶点，顶点法线，3套uv和顶点切线，其中某些变量是通过关键字进行包含的。输出结构为VertexOutputForwardBase，主要包括顶点，uv，观察向量，世界坐标，以及世界空间法线或者切线空间法线。需要注意的是，ambientOrLightmapUV是光照贴图uv坐标或者是球谐光照，属于全局光照的部分。

这样顶点函数也就十分简单了，主要包括顶点，uv，世界坐标，法线。ambientOrLightmapUV的赋值是VertexGIForward，里面主要是光照贴图uv变换或者球谐光照计算。

*half4 ambientOrLightmapUV = 0;*

*// Static lightmaps*

*#ifdef LIGHTMAP\_ON*

*ambientOrLightmapUV.xy = v.uv1.xy \* unity\_LightmapST.xy + unity\_LightmapST.zw;*

*ambientOrLightmapUV.zw = 0;*

*// Sample light probe for Dynamic objects only (no static or dynamic lightmaps)*

*#elif UNITY\_SHOULD\_SAMPLE\_SH*

*#ifdef VERTEXLIGHT\_ON// Approximated illumination from non-important point*

*lightsambientOrLightmapUV.rgb = Shade4PointLights (unity\_4LightPosX0, unity\_4LightPosY0,unity\_4LightPosZ0,unity\_LightColor[0].rgb, unity\_LightColor[1].rgb,unity\_LightColor[2].rgb,unity\_LightColor[3].rgb, unity\_4LightAtten0, posWorld, normalWorld);*

*#endif*

*ambientOrLightmapUV.rgb = ShadeSHPerVertex (normalWorld, ambientOrLightmapUV.rgb);*

*#endif*

*…*

顶点函数里面还有视差贴图(\_PARALLAXMAP)的相关处理，这些可以暂且跳过。

**fragForwardBase**

fragForwardBase片段函数主要通过fragForwardBaseInternal实现。

里面首先就是FRAGMENT\_SETUP(s)，进行alpha测试，以及基本片段数据包括法线向量，观察向量，世界坐标的计算等。特别的是UNITY\_SETUP\_BRDF\_INPUT，设置BRDF计算的输入数据，根据设置的不同，分别有SpecularSetup，RoughnessSetup和MetallicSetup。我们主要使用MetallicSetup，里面根据金属工作流计算了metallic，smoothness，oneMinusReflectivity，diffColor，specColor等。

*half oneMinusReflectivity;*

*half3 specColor;*

*half3 diffColor = DiffuseAndSpecularFromMetallic (Albedo(i\_tex), metallic, /\*out\*/ specColor, /\*out\*/ oneMinusReflectivity);*

然后是计算主光源和光照衰减。

*UnityLight mainLight = MainLight ();*

*UNITY\_LIGHT\_ATTENUATION(atten, i, s.posWorld);*

接着计算全局光照GI

UnityGI gi = FragmentGI(s. occlusion, i.ambientOrLightmapUV, atten, mainLight);

UnityGI包括直接光照和间接光照，直接光照就是mainLight，主要是间接光照计算。间接光照根据设置的不同，选择使用光照贴图lightmap或者球谐光照，这些都是间接漫反射光照。

*#if defined(LIGHTMAP\_ON) || defined(DYNAMICLIGHTMAP\_ON)*

*d.ambient = 0;*

*d.lightmapUV = i\_ambientOrLightmapUV;*

*#else*

*d.ambient = i\_ambientOrLightmapUV.rgb;*

*d.lightmapUV = 0;*

*#endif*

间接镜面反射是通过LightProbe实现的

*#if defined(UNITY\_SPECCUBE\_BLENDING)||*

*defined(UNITY\_SPECCUBE\_BOX\_PROJECTION)*

*d.boxMin[0] = unity\_SpecCube0\_BoxMin; // .w holds lerp value for blending*

*#endif*

如果定义了Box Projection，还要调整采样方向，需要SpecCube位置和边界

*#ifdef UNITY\_SPECCUBE\_BOX\_PROJECTION*

*d.boxMax[0] = unity\_SpecCube0\_BoxMax;*

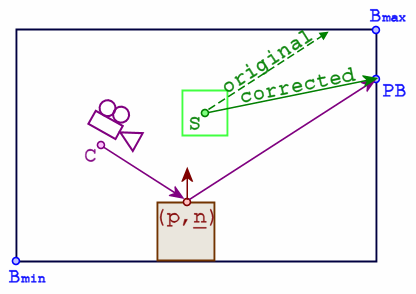
*d.probePosition[0] = unity\_SpecCube0\_ProbePosition;*

*d.boxMax[1] = unity\_SpecCube1\_BoxMax;*

*d.boxMin[1] = unity\_SpecCube1\_BoxMin;*

*d.probePosition[1] = unity\_SpecCube1\_ProbePosition;*

*#endif*



最后通过UnityGlobalIllumination计算全局光照。

正如前面所说，间接漫反射光照选择使用光照贴图或者球谐光照。

*#if UNITY\_SHOULD\_SAMPLE\_SH // 使用球谐光照*

*o\_gi.indirect.diffuse = ShadeSHPerPixel(normalWorld, data.ambient, data.worldPos);*

*#endif*

*#if defined(LIGHTMAP\_ON) // 使用光照贴图*

*// Baked lightmaps*

*half4 bakedColorTex = UNITY\_SAMPLE\_TEX2D(unity\_Lightmap, data.lightmapUV.xy);*

*half3 bakedColor = DecodeLightmap(bakedColorTex);*

*…*

*#endif*

间接镜面反射光照通过LightProbe计算

#ifdef UNITY\_SPECCUBE\_BOX\_PROJECTION

*// we will tweak reflUVW in glossIn directly (as we pass it to Unity\_GlossyEnvironment twice for probe0 and probe1), so keep original to pass into BoxProjectedCubemapDirection*

*half3 originalReflUVW = glossIn.reflUVW;*

*glossIn.reflUVW = BoxProjectedCubemapDirection (originalReflUVW, data.worldPos, data.probePosition[0], data.boxMin[0], data.boxMax[0]);*

*#endif*

*#ifdef \_GLOSSYREFLECTIONS\_OFF*

*specular = unity\_IndirectSpecColor.rgb;*

*#else*

*half3 env0=Unity\_GlossyEnvironment(*

*UNITY\_PASS\_TEXCUBE(unity\_SpecCube0),data.probeHDR[0], glossIn);*

*…*

*#endif*

最后就是计算UNITY\_BRDF\_PBS了。根据BRDF计算最终片段输出颜色。

5. UnityStandardBRDF

UNITY\_BRDF\_PBS是在UnityPBSLighting.cginc里面定义的。Unity根据相关配置的不同，定义了3个不同的BRDF，分别是BRDF1\_Unity\_PBS，BRDF2\_Unity\_PBS，BRDF3\_Unity\_PBS。3个BRDF计算逐渐简单，性能逐渐提高，效果当然对应下降。

**BRDF1\_Unity\_PBS**

这是Unity的标准BRDF，diffuse部分采用了Disney Diffuse计算，specular部分采用了GGX法线分布函数，Smith方法的Visiblity计算和Schlick菲涅尔近似。

DisneyDiffuse考虑了表面粗糙度的影响



其中，。

V，D，F的计算都采用了标准的近似。

**BRDF2\_Unity\_PBS**

这套BRDF使用了漫反射部分使用了Lambertian近似，高光反射部分使用了修改的GGX分布函数和修改的Visibility计算，Fresnel反射采用1/LdotH近似。

G/V项使用近似计算(Jonh Hable, “Optimizing GGX Shaders with dot(L, H)”)



在0°角观察时，V值为1，在90°角观察时，V值为。

VF项进一步近似为



最终得到的Specular BRDF为



*half a = roughness;*

*float a2 = a\*a;*

*float d = nh \* nh \* (a2 - 1.f) + 1.00001f;*

*float specularTerm = a2 / (max(0.1f, lh\*lh) \* (roughness + 0.5f) \* (d \* d) \* 4);*

**BRDF3\_Unity\_PBS**

使用一张LUT unity\_NHxRoughness存储高光反射，进一步减少计算，提升性能。在实际实现中，使用代替节省指令，并用调整实际效果。

*half2 rlPow4AndFresnelTerm = Pow4 (float2(dot(reflDir, light.dir), 1-nv));*

*half3 BRDF3\_Direct(half3 diffColor, half3 specColor, half rlPow4, half smoothness){*

*half LUT\_RANGE = 16.0; // must match range in NHxRoughness() function in GeneratedTextures.cpp*

*// Lookup texture to save instructions*

*half specular = tex2D(unity\_NHxRoughness, half2(rlPow4, SmoothnessToPerceptualRoughness(smoothness))).r \* LUT\_RANGE;*

*#if defined(\_SPECULARHIGHLIGHTS\_OFF)*

*specular = 0.0;*

*#endif*

*return diffColor + specular \* specColor;*

*}*

这是最简单的标准着色器实现，适用于低端机器。

6. 参考资料

<https://academy.allegorithmic.com/courses/the-pbr-guide-part-1>

<https://blog.selfshadow.com/publications/s2012-shading-course/burley/s2012_pbs_disney_brdf_notes_v3.pdf>

<https://blog.selfshadow.com/publications/s2013-shading-course/>

<https://graphicrants.blogspot.com/2013/08/specular-brdf-reference.html>

<https://learnopengl.com/PBR>

<https://agraphicsguy.wordpress.com/2015/11/01/sampling-microfacet-brdf/>

<https://wiki.jmonkeyengine.org/jme3/advanced/pbr_part3.html#ibl-specular>

<https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch20.html>

<https://blog.teamtreehouse.com/beginners-guide-physically-based-rendering-unity>

<https://www.trentreed.net/blog/physically-based-shading-and-image-based-lighting/>

<https://www.cnblogs.com/zhouxin/p/5168632.html>

<https://www.cnblogs.com/murongxiaopifu/p/8997720.html>

<https://blog.csdn.net/BugRunner/article/details/6994379>

<http://www.cnblogs.com/bananana/p/8867537.html>

<https://chetanjags.wordpress.com/2015/08/26/image-based-lighting/>

<https://zhuanlan.zhihu.com/p/20091064>

<https://zhuanlan.zhihu.com/p/34741174>

<https://zhuanlan.zhihu.com/p/53086060>

<https://blog.csdn.net/poem_qianmo/article/details/49556461>

<http://filmicworlds.com/blog/optimizing-ggx-shaders-with-dotlh/>

<https://community.arm.com/developer/tools-software/graphics/b/blog/posts/moving-mobile-graphics#siggraph2015>