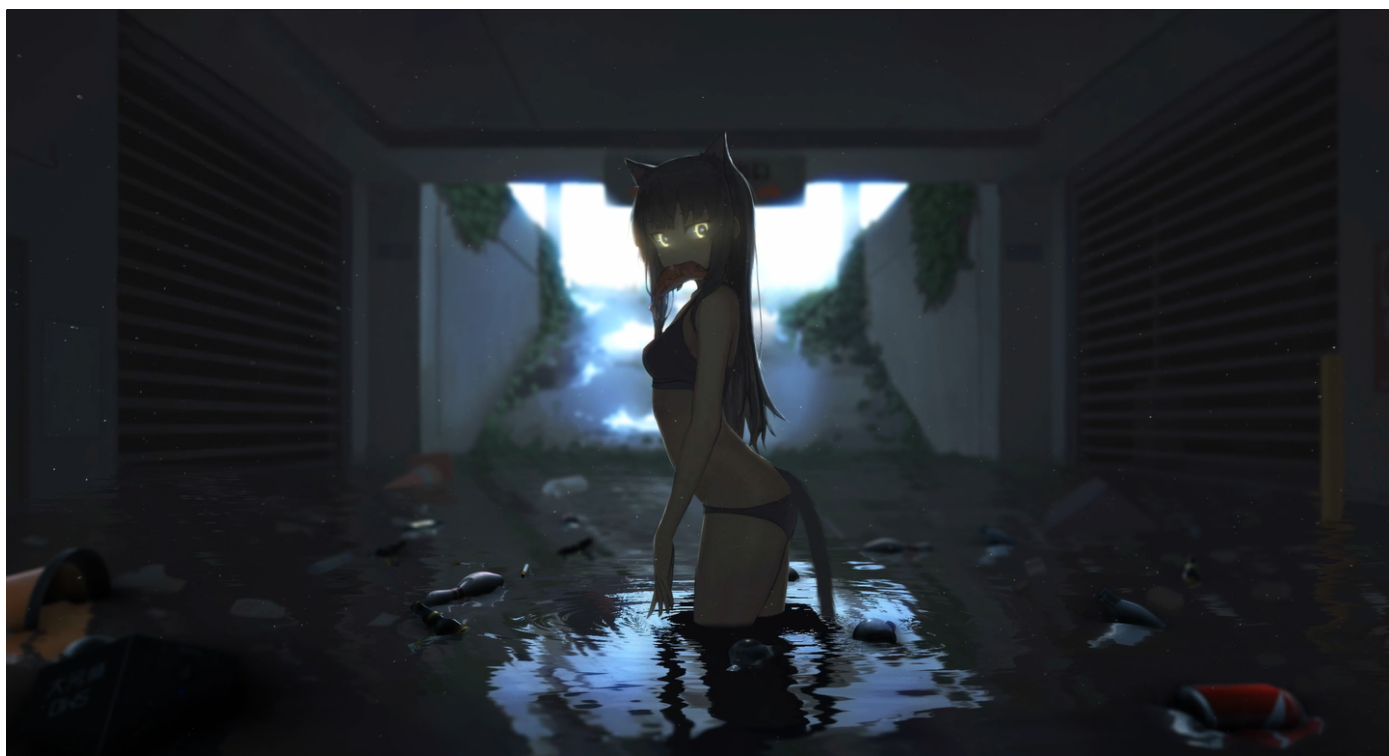


## Games202 高质量实时渲染笔记lecture 05 Real-Time Environment Mapping



本文是闫令琪教授所教授的Games-202:Real-Time High Quality Rendering学习笔记的第五讲 Real-Time Environment Mapping, 本人属于新手上路暂无驾照, 有错误欢迎各位大佬指正.\

[GAMES202-高质量实时渲染\\_哔哩哔哩\\_\(° - °\)つ口 干杯~-bilibili](#)

上节课我们介绍了PCF,PCSS和VSSM,虽然VSSM已经逐渐没有人使用了,但是他其中的算法思想是十分值得我们去思考的.

今日目录:

# Today

- Finishing up on shadows
  - Distance field soft shadows
- Shading from environment lighting
  - The split sum approximation
- Shadow from environment lighting

知乎 @WhyS0fAr

## I)Finishing up on shadows

---

### A-> Signed Distance Function

首先我们来复习一下SDF,在games101里我们知道,Distance Function就是空间中任何一点,到某个物体的最小距离.

# From GAMES101: Distance Functions

Distance functions:

At any point, giving the **minimum distance** (could be **signed distance**) to the closest location on an object

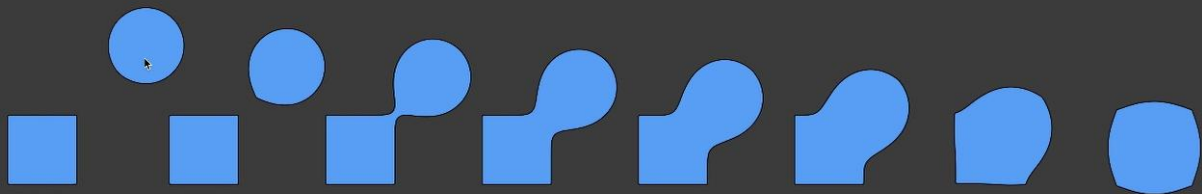


知乎 @WhyS0fAr

Signed的意思就是加入规定负数代表在物体内部,正数则表示在物体外部.这样不止定义了距离,还定义了方向.

## From GAMES101: Distance Functions

- Can blend any two distance functions  $d_1$ ,  $d_2$



知乎 @WhyS0fAr

SDF大多用在与几何之类的方向上,比如我有两个物体的SDF,我们将两个SDF进行Blend操作,从而得到一个新的SDF,也就得到了一个新的几何物体,因为当距离0为实,即可认为是物体的边界, SDF可以很准确地反应物体的边界。几何转换SDF的时候,可以很好地把几何进行过渡。

## **B -> SDF的应用:**

### **Usage 1 - >Ray Marching**

假设我们已经知道场景的SDF,现在有一根光线,我们试图让光线和SDF所表示的隐含表面进行求交,也就是我们要用sphere tarcing(ray marching)进行求交.

Ray marching是一个很有趣且聪明的思想:

任意一点的SDF我们是已知的,因此在 $P_0$ 点时,我们以它的SDF( $P_0$ )为半径做一个圆(此处假设在2D内,如果在3D内则是一个球),在这个圆内无论是哪个方向前进,只要不超过半径距离,都不会碰见物体,是安全的.

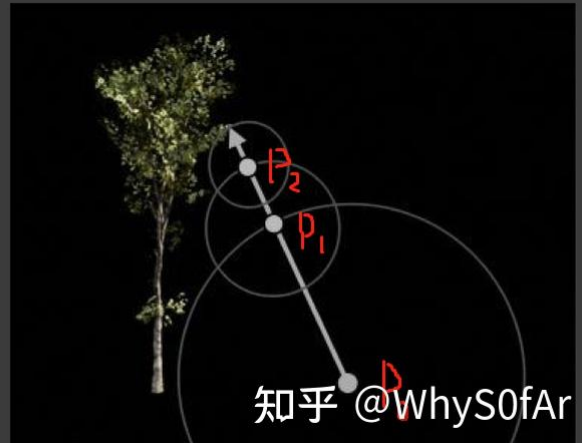
因此我们可以利用这个特性不断的朝一个方向前进,以 $P_0$ 开始朝一方向走SDF( $P_0$ )到达  $P_1$ 点,若仍与物体表面相距甚远,则以 $P_1$ 点为新起点继续走SDF( $P_1$ )到达  $P_2$ 点,假设此处  $P_2$ 点的SDF足够小,也就是代表离物体表面足够接近了,则进行求交操作.

如果在超一方向trace非常远的距离但仍然什么都没trace到,此时就可以舍弃这条光线,也就是停止了.

# The Usages of Distance Fields

- Usage 1

- Ray marching (sphere tracing) to perform ray-SDF intersection
- Very smart idea behind this:
- The value of SDF == a “safe” distance around
- Therefore, each time at  $p$ , just travel  $SDF(p)$  distance



*Ray marching*

## Usage 2 - > 生成软阴影

将安全距离的概念进行延伸，在任意一点通过sdf可以获得一个safe angle.

# The Usages of Distance Fields

- Usage 2
  - Use SDF to determine the (approx.) percentage of occlusion
  - the value of SDF -> a "safe" angle seen from the eye
- Observation
  - Smaller "safe" angle <-> less visibility



我们取点P为shading point往一方向打出一根光线,光线上的  
一点a,有一个SDF值 $SDF(a)$ ,也就是在a点以 $SDF(a)$ 为半径所做的球  
或圆内是安全的,不会碰到物体.

把shading point和面光源 相连,所得到的安全角度越小,被  
遮蔽的可能越高,就可以认为

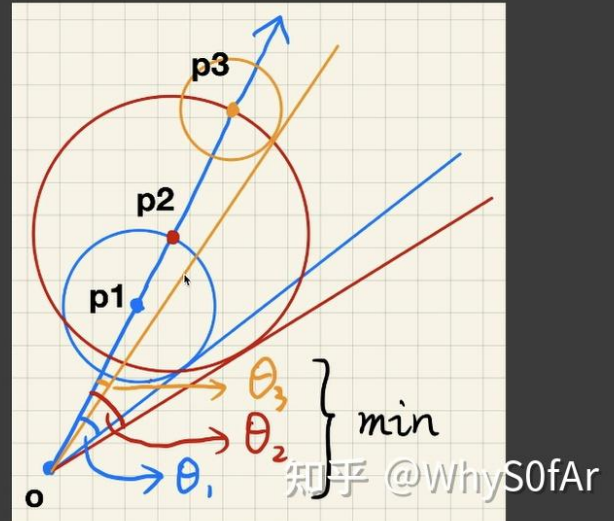
**safe angle**越小, 阴影越黑,越趋近于硬阴影;

**safe angle**够大就视为不被遮挡没有阴影,也就越趋近于软阴  
影。

因此我们需要知道的应是如何从ray marching中求出safe  
angle:

# Distance Field Soft Shadows

- During ray matching
  - Calculate the “safe” angle from the eye at every step
  - Keep the minimum
  - How to compute the angle?



我们以o为起点,沿一个方向推进,仍然是ray matching的步骤,在p1点以SDF(p1)进行推进,其余点也是一样,此处主要是为了求safe angle,我们在起点o沿每个点的sdf为半径所形成的圆做切线,从而求出各个点的safe angle,我们最后再取其中最小的角度作为总的safe angle.

也就是我们在trace的每次过程中都求出一个safe angle,到最后进行相交操作时取最小的safe angle来用.

那么我们该怎么去计算这个角度?


从图我们可以知道,以p1点为例,从o点到p1的距离为斜边,sdf(p1)是直角边,因此我们用arcsin就可以求出safe angle了.

但是!!!!!!!



arcsin的计算量其实是十分大的,因此在shader中我们不用反三角函数.

- How to compute the angle?



$$\arcsin \frac{\text{SDF}(p)}{p - o} \quad \min \left\{ \frac{k \cdot \text{SDF}(p)}{p - o}, 1.0 \right\}$$

知乎 @WhyS0fAr

只要sdf长度除以光线走过的距离乘一个k值,再限定到1以内,就能得到遮挡值或者说是visibility,而k的大小是控制阴影的软硬程度.



# Distance Field Soft Shadows

- How to compute the angle?



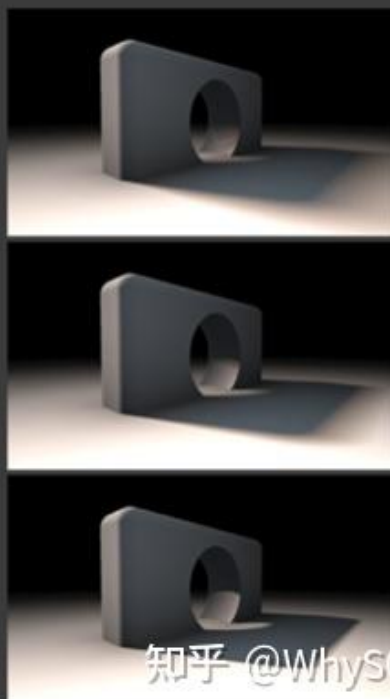
$$\arcsin \frac{\text{SDF}(p)}{p - o} \quad \min \left\{ \frac{k \cdot \text{SDF}(p)}{p - o}, 1.0 \right\}$$

- Larger  $k \leftrightarrow$  earlier cutoff of penumbra  $\leftrightarrow$  harder

$k = 2$

$k = 8$

$k = 32$



我们从图中右半部分可以看出来,当k值越大时候,就越接近硬阴影的效果,也就是它限制了可能半影的区域:

k越小,半影区域越大,越接近软阴影效果.

K越大,半影区域越小,越接近硬阴影效果.

这个思想并不是去考虑一个  $\frac{\text{安全角度}}{\text{面光源覆盖的总角度}}$  从而求出一个比值,再用这个比值作为visibility.

而是求出safe angle,安全角度越大,阴影越软;安全角度越小,阴影越硬.

**Conclusion:**

SDF是一个快速的高质量的软阴影生成方法(比shadow map快是忽略了SDF生成的时间),但是在存储上的消耗非常大,而且生成SDF的时间也要很久, SDF是预计算, 在有动态的物体的情况就得重新计算SDF。

## **Shading from environment lighting**

在环境光照下做一点的shading时是不用去考虑shadow的,首先我们复习一下什么是环境光照:

在games101中我们知道,环境贴图就是在场景中任意一点往四周看去可看到的光照,将其记录在一张图上这就是环境光照,或者也可以叫做IBL(image-based lighting).这里我们认为看到的光照来自于无限远处,这也就是为什么用环境光照去渲染物体时会产生一种漂浮在空中的感觉,因为光照来自于无限远处.

通常我们用spherical map和cube map来存储环境光照.

如果已知环境光照,此时放置一个物体在场景中间,在不考虑遮挡时我们该如何去得到任何一物体上任何一shading part的shading值呢?

首先要先来看rendering equation:

# Shading from Environment Lighting

- Informally named **Image-Based Lighting (IBL)**
- How to use it to shade a point (**without shadows**)?
  - Solving the rendering equation

$$L_o(p, \omega_o) = \int_{\Omega^+} L_i(p, \omega_i) f_r(p, \omega_i, \omega_o) \cos \theta_i V(p, \omega_i) d\omega_i$$

环境光照 (IBL) 指向  $L_i(p, \omega_i)$

渲染方程定义只考虑来自正半球方向的光照 指向  $\int_{\Omega^+}$

BRDF 指向  $f_r(p, \omega_i, \omega_o)$

$V(p, \omega_i)$  被红色叉号划掉

知乎 @WhyS0fAr

由于我们在这里不考虑遮挡,所以舍去visibility项.通用的解法是使用蒙特卡洛积分去解,但是蒙特卡洛需要大量的样本才能让得出的结果足够接近,如果我们对每个shading point都做一遍蒙特卡洛,那样的话将会花费很多时间在采样上,太慢了.

(ps:一般来说shader中如果出来Sampling的话是不会用于rtr中的,但是由于这几年技术的提升,使得sampling在rtr中也有了可行性.)

# Shading from Environment Lighting

- General solution — Monte Carlo integration
  - Numerical
  - Large amount of samples required
- Problem — can be slow
  - In general, sampling is not preferred in shaders\*
  - **Can we avoid sampling?**



知乎 @WhyS0fAr

既然采样这么慢,那么如果我们不进行采样呢?

## 基本思路

由于我们不考虑visibility项,那么rendering equation就只是brdf项和lighting项相乘再积分。

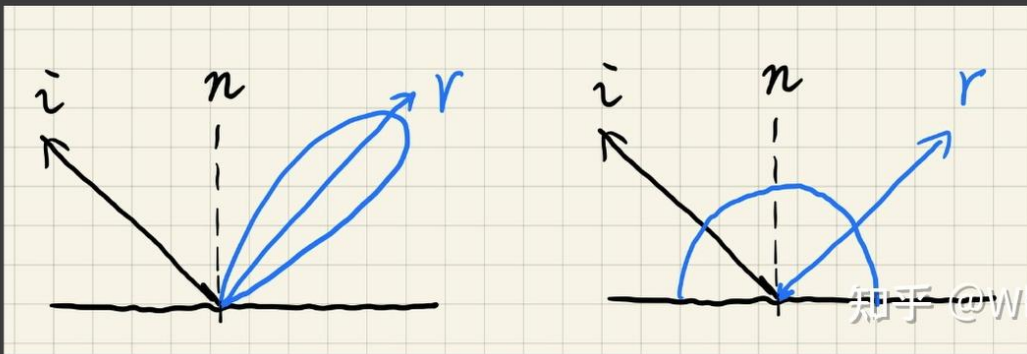
## brdf又分为两种情况:

1. brdf为glossy时,覆盖在球面上的范围很小,也就是small support(积分域).
2. brdf为diffuse时,它会覆盖整个半球的区域,但是是smooth的,也就是值的变化不大,就算加上cos也是相对平滑的.

# Shading from Environment Lighting

- Observation

- If the BRDF is glossy — small support!
- If the BRDF is diffuse — smooth!
- Does the observation remind you of something?



此时应该想到了我们之前讲的近似公式:

## The Classic Approximation

- Recall: the approximation

- Note the slight edit on  $\Omega_G$  here

$$\int_{\Omega} f(x)g(x) dx \approx \frac{\int_{\Omega_G} f(x) dx}{\int_{\Omega_G} dx} \cdot \int_{\Omega} g(x) dx$$

- Conditions for acceptable accuracy?

知乎 @WhyS0fAr

由于brdf项满足accuracy condition,即small support或smooth,从而我们将rendering equation的lighting项拆出让他变为：

- BRDF satisfies the accuracy condition in any case
  - We can safely take the lighting term out!

$$L_o(p, \omega_o) \approx \frac{\int_{\Omega_{f_r}} L_i(p, \omega_i) d\omega_i}{\int_{\Omega_{f_r}} d\omega_i} \cdot \int_{\Omega^+} f_r(p, \omega_i, \omega_o) \cos \theta_i d\omega_i$$

知乎 @WhySofAr

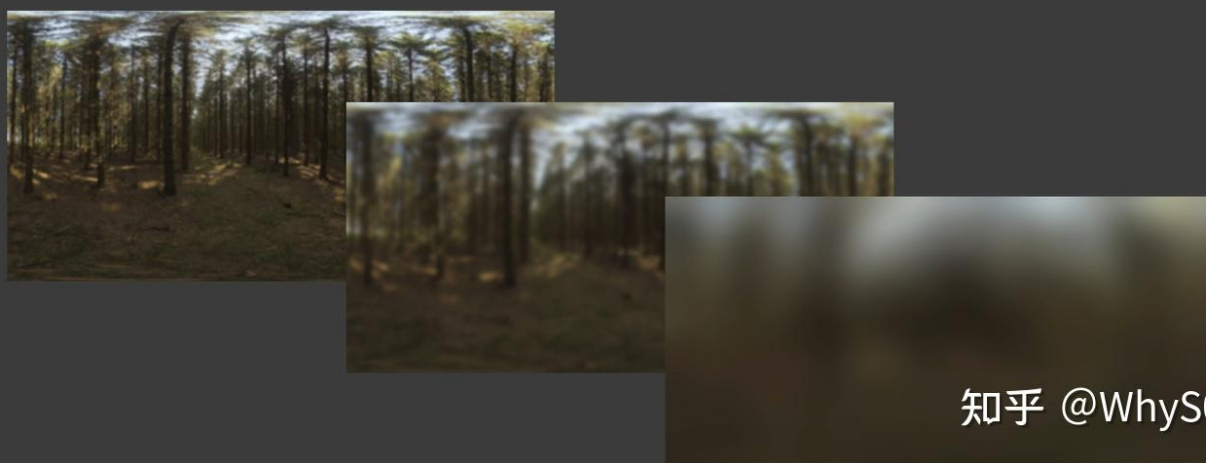
## The spilt sum: 1st stage

把light项拆分出来,然后将brdf范围内的lighting积分起来并进行normalize,其实就是将IBL这张图给模糊了.

模糊就是在任何一点上取周围一片范围求出范围内的平均值并将平均值写回这个点上,主滤波的核取多大取决于BRDF占多大,BRDF的区域越大,最后取得的图也就越模糊.

# The Split Sum: 1st Stage

- **Prefiltering** of the environment lighting
  - Pre-generating a set of differently filtered environment lighting
  - Filter size in-between can be approximated via trilinear interp.



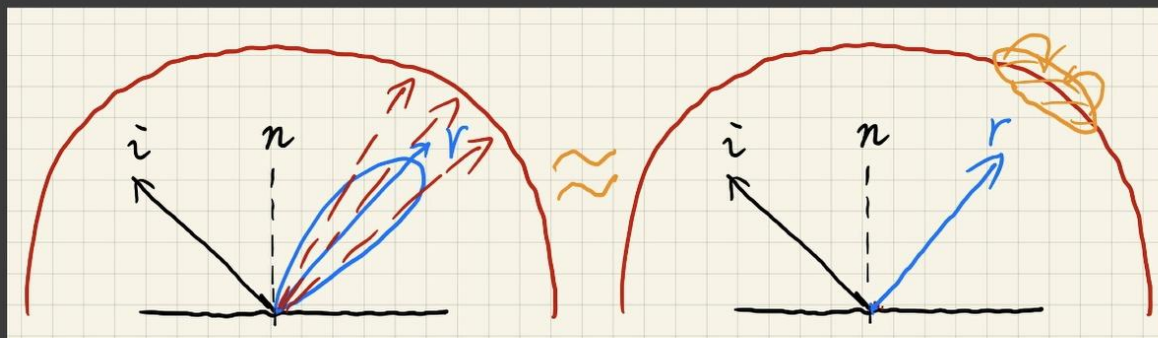
而这些模糊过的图是我们在进行rendering之前生成的,也就是pre-filtering,Prefiltering就是提前把滤波环境光生成,提前将不同卷积核的滤波核的环境光生成一系列模糊过的图(和mipmap相似),当我们需要时进行查询即可,其他尺寸的图则可以经过这些已生成的通过三线性插值得到.

之前提过,拆分就是为了做一个Pre-filtering,那么做pre-filtering是为了干什么?



# The Split Sum: 1st Stage

- Then query the pre-filtered environment lighting at the  **$r$  (mirror reflected) direction!**



知乎 @WhyS0fAr

左图为brdf求shading point值时,我们要以一定立体角的范围内进行采样再加权平均从而求出shading pointd的值.

右图为我们从镜面反射方向望去在pre-filtering的图上进行查询,由于图上任何一点都是周围范围内的加权平均值,因此在镜面反射方向上进行一次查询就等价于左图的操作,并且不需要采样,速度更快.

## The spilt sum: 2nd stage

到此我们解决了拆分后的前半部分积分采样的问题,那么接下来我们处理BRDF项采样的问题:

# The Split Sum: 2nd Stage

- The second term is still an integral
  - How to avoid sampling this term?

$$L_o(p, \omega_o) \approx \frac{\int_{\Omega_{f_r}} L_i(p, \omega_i) d\omega_i}{\int_{\Omega_{f_r}} d\omega_i} \cdot \int_{\Omega^+} f_r(p, \omega_i, \omega_o) \cos \theta_i d\omega_i$$

知乎 @WhyS0fAr

接下来讲的方法并不是最优方法,现如今已经有更简单方便的方法了,但是本课我们主要是为了学习方法背后的思想.

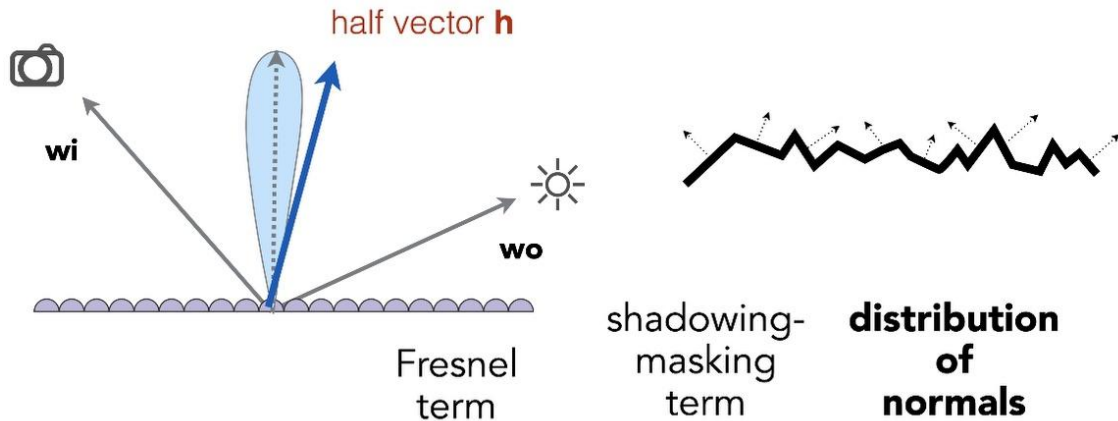
- Idea
  - Precompute its value for all possible combinations of variables roughness, color (Fresnel term), etc.

知乎 @WhyS0fAr

我们仍然可以用预计算来解决后半部分积分采样的问题,但是预计算的话我们需要将参数的所有可能性均考虑进去,但是比较多,包括roughness、color等。考虑所有参数的话我们需要打印出一张五维或者更高的表格,这样会拥有爆炸的存储量,因此我们需要想办法降低维度,也就是减少参数量从而实现预计算.

## Recall: Microfacet BRDF

- What kind of microfacets reflect  $w_i$  to  $w_o$ ?  
(hint: microfacets are mirrors)



$$f(\mathbf{i}, \mathbf{o}) = \frac{\mathbf{F}(\mathbf{i}, \mathbf{h}) \mathbf{G}(\mathbf{i}, \mathbf{o}, \mathbf{h}) \mathbf{D}(\mathbf{h})}{4(\mathbf{n}, \mathbf{i})(\mathbf{n}, \mathbf{o})}$$

知乎 @WhySOfAr

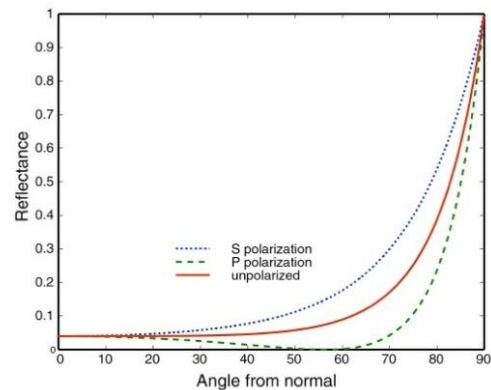
在microfacet brdf中中，考虑的是菲涅尔项、阴影项以及法线项，由于此时暂时不考虑阴影，此处需要关注的是Fresnel term和distribution of normals。

# The Fresnel Term and the NDF

Fresnel term: the Schlick's approximation

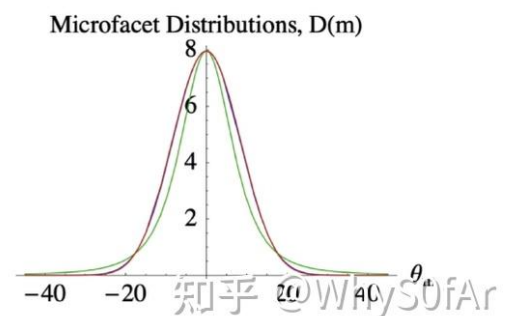
$$R(\theta) = R_0 + (1 - R_0)(1 - \cos \theta)^5$$

$$R_0 = \left( \frac{n_1 - n_2}{n_1 + n_2} \right)^2$$



The NDF term: e.g. Beckmann distribution

$$D(h) = \frac{e^{-\frac{\tan^2 \theta_h}{\alpha^2}}}{\pi \alpha^2 \cos^4 \theta_h}$$



Fresnel term可以近似成一个基础反射率 $R_0$ 和入射角度的指数函数

法线发布函数 (NDF) 是一个一维的分布，其中有两个变量，一个变量定义是diffuse还是gloosy，另一个是half vector和法线中间的夹角，可以近似成入射角度相关的数，这样就变成了3维的预计算。

(PS:在这里我们认为反射角,入射角,half vector可以用一个角  $\theta$  代替).

至此我们有了三个变量:基础反射率 $r_0$ ,roughness  $\alpha$ 和角度  $\theta$ , 三维的预计算仍然是一个存储量爆炸的结果,因此我们还要想办法减少参数量.

所以我们通过将Schlick近似带入后半部分的积分中：

- Taking the Schlick's approximation into the 2nd term

- The "base color" is extracted!

$$\int_{\Omega^+} f_r(p, \omega_i, \omega_o) \cos \theta_i d\omega_i \approx R_0 \int_{\Omega^+} \frac{f_r}{F} (1 - (1 - \cos \theta_i)^5) \cos \theta_i d\omega_i + \int_{\Omega^+} \frac{f_r}{F} (1 - \cos \theta_i)^5 \cos \theta_i d\omega_i$$

基础反射 $R_0$ 被拆出积分式，需要预计算的两个量就只有 roughness  $\alpha$  和角度  $\theta$ ，可以将预计算结果绘制成一张纹理，在使用时进行查询即可。

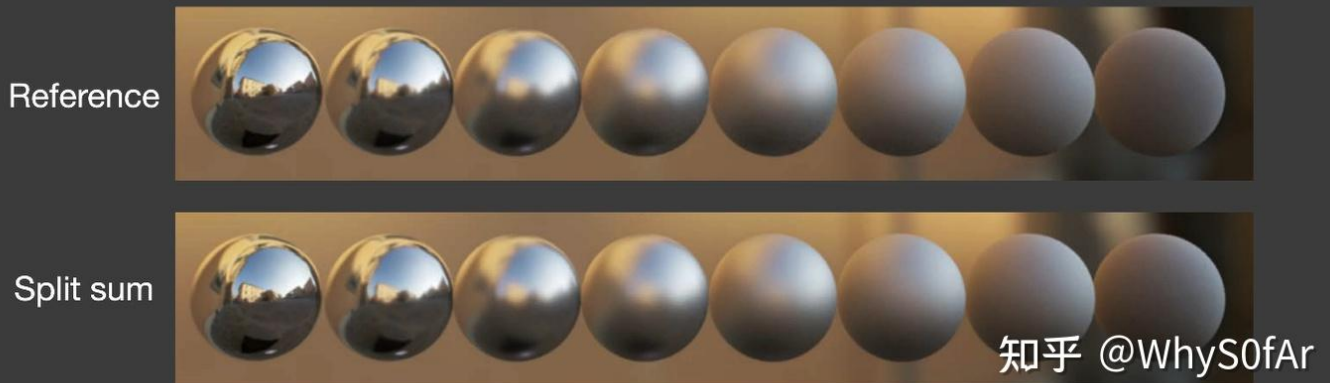
- Each integral produces one value for each (roughness, incident angle) pair
  - Therefore, each integral results in a 2D table (texture)



我们可以看到,最后产生的结果是十分满意的:

# The Split Sum Approximation

- Finally, completely avoided sampling
- Very fast and almost identical results



问题:

1.  $\text{fresnel}$  需要预计算吗

$\text{fresnel}$  项被拆开了，避免了对变量的依赖性

2 这张预计算是固定的吗？

是固定的。

3 Microfacet 在  $\text{ggx}$  中会多参数吗

不会

4. 深度学习在实时渲染中有什么应用吗

深度学习在实时渲染中并不成功，太慢了。