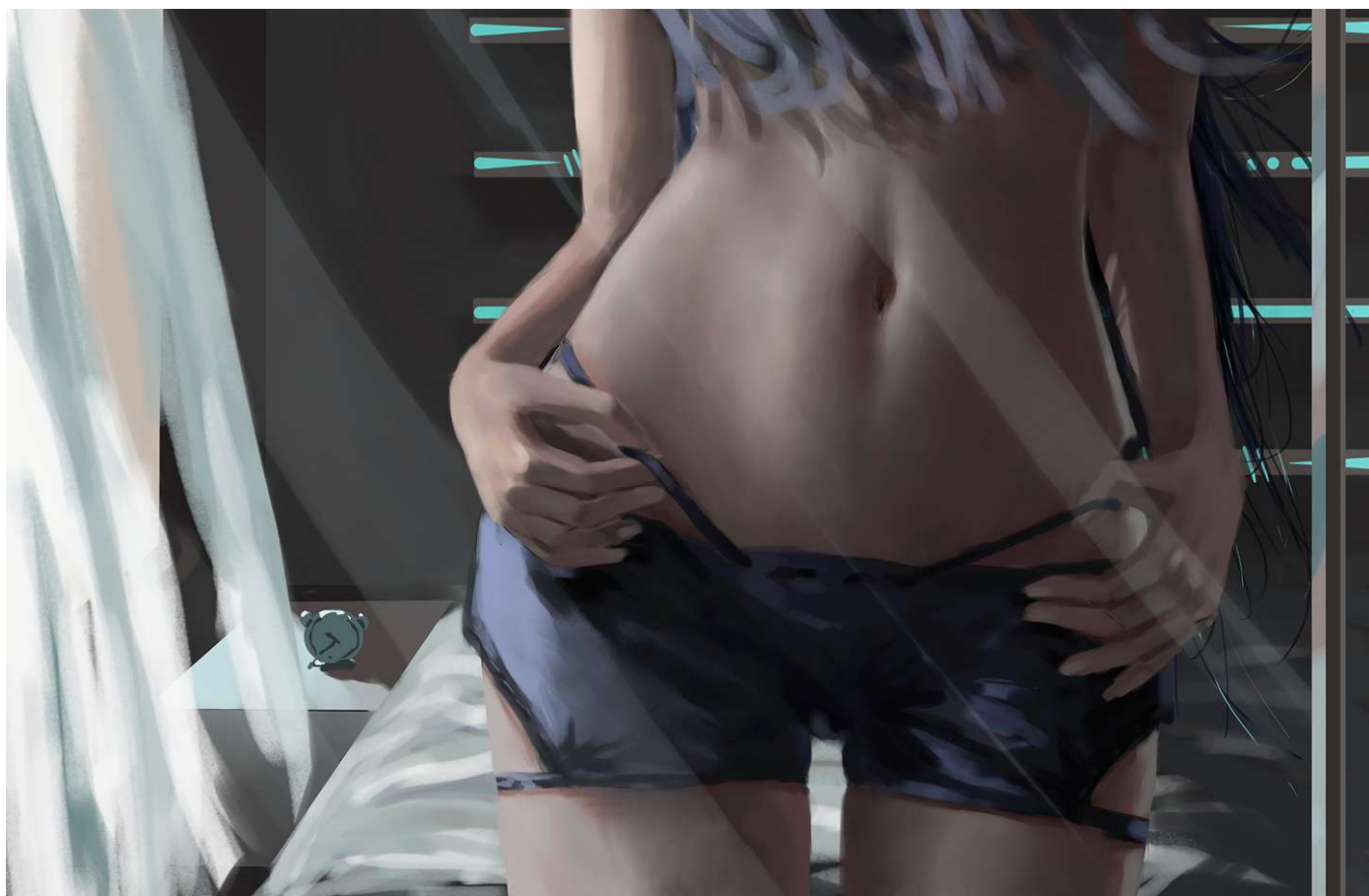


GAMES202 高质量实时渲染笔记Lecture07: Real-Time Global Illumination (In 3D)



本文是闫令琪教授所教授的Games-202:Real-Time High Quality Rendering学习笔记本的第七讲 Real-Time Global Illumination (In 3D), 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bilibili](#)

在上节课我们讲了如何在已知环境光照和物体的brdf是Diffuse的情况下通过prt计算出任意一个shading point在考虑shadow后的shading result.

注意:此处的shading point指的是几何形体上的vertex.

PRT预计算了两部分,一部分是lighting项,一部分是除了lighting项以外的部分,称其为light transports项.

1.brdf是diffuse相当于是一个常数,因此可以提到外面.

2.由于环境光照是一个球面函数,因此可以用sh基函数的线性组合表示,把其代入,由于 l_i 相对于积分是一个常数,可以提出.

Diffuse Case

$$L(\mathbf{o}) = \rho \int_{\Omega} \underline{L(\mathbf{i})} V(\mathbf{i}) \max(0, \mathbf{n} \cdot \mathbf{i}) d\mathbf{i}$$

$L(\mathbf{i}) \approx \sum l_i B_i(\mathbf{i})$

lighting coefficient basis function

$$L(\mathbf{o}) \approx \rho \sum l_i \int_{\Omega} B_i(\mathbf{i}) V(\mathbf{i}) \max(0, \mathbf{n} \cdot \mathbf{i}) d\mathbf{i}$$

知乎 @WhyS0fAr

3.剩下的light transport项也可以看作一个球面函数,现如今积分里剩基函数和球面函数,相当于函数投影到基函数求系数,因此变为了一个系数.

Diffuse Case

$$L(\mathbf{o}) = \rho \int_{\Omega} L(\mathbf{i}) V(\mathbf{i}) \max(0, \mathbf{n} \cdot \mathbf{i}) d\mathbf{i}$$

$L(\mathbf{i}) \approx \sum l_i B_i(\mathbf{i})$

lighting coefficient basis function

$$L(\mathbf{o}) \approx \rho \sum l_i \int_{\Omega} B_i(\mathbf{i}) V(\mathbf{i}) \max(0, \mathbf{n} \cdot \mathbf{i}) d\mathbf{i}$$

Precompute

$$L(\mathbf{o}) \approx \rho \sum l_i T_i$$

◎ Reduce rendering computation to dot product 知乎@WhySofAr

最终我们在任意一个shading point点求得的shading result,在实际计算时只是lighting项投影到sh基函数上求得的li系数组成的vector和Light transport项投影到SH基函数上求得的Ti系数组成的vector做了个点乘而已。

在实际运算时,我们对每个顶点预计算出它的shading result,三角形内部任一点则通过插值得到它的shading result,这种叫做 **per vertex shading**.

或者我们在三角形内部先插值好lighting的vector和light transport的vector,然后再去计算pixel上的shading result,这种叫做 **per pixel shading**.

今天首先我们从另外一个角度重新来看, 怎么对Light transport做预计算:

1. 理解一：上节课是我们先把Light写成基函数的表示方法，把Light的系数拆分出去，然后剩下的部分看做把Light transport投影到基函数上得到系数,最后做一个点乘得到 Shading result.
2. 理解二：直接把渲染方程中的lighting和light transport，都用sh基函数表示

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

The diagram illustrates the decomposition of lighting and light transport into coefficients and basis functions. On the left, the lighting function is represented as $L(\omega_i) \approx \sum_p c_p B_p(\omega_i)$. A red arrow points from the coefficient c_p to the text 'lighting coefficient', and another red arrow points from the basis function $B_p(\omega_i)$ to the text 'basis function'. On the right, the light transport function is represented as $T(\omega_i) \approx \sum_q c_q B_q(\omega_i)$. A red arrow points from the coefficient c_q to the text 'light transport coefficient', and another red arrow points from the basis function $B_q(\omega_i)$ to the text 'basis function'. A watermark '知乎 @WhySofAr' is visible in the bottom right corner of the diagram.

然后把两个都展开成求和，然后把求和符号拆出去，然后就变成了一个双重求和的结果,每个求和要乘三样东西：

- 1.对应的两个系数
- 2.积分值(积分与实际场景无关,是两个基函数的product integral)

这样就会发现这样推导的结果与上一节课的结果不太一样

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

$$= \sum_p \sum_q c_p c_q \int_{\Omega^+} B_p(\omega_i) B_q(\omega_i) d\omega_i$$

$$L(\omega_i) \approx \sum_p c_p B_p(\omega_i)$$

$$T(\omega_i) \approx \sum_q c_q B_q(\omega_i)$$

知乎 @WhyS0fAr

如果基函数的个数为n的话,做一个向量点乘的复杂度应该是O(n),为什么在这里是双重求和变成了O(n²)了呢?

这是因为SH具有正交性,也就是当p=q时候, $B_p(\omega_i) B_q(\omega_i)$ 才不为0, 也就是这个二维矩阵上只有对角线上有值,因此只需要计算对角线上的值就行了,所以算法复杂度仍然是O(N)。

对于glossy的物体做预计算


Diffuse和glossy的区别在于,diffuse的brdf是一个常数,而glossy的brdf是一个4维的brdf(2维的输入方向,2维的输出方向).

如果仍然按照上面的办法投影到sh上会出现一些问题,因为light transport包含visibility和brdf,brdf又是一个4维的函数(关于i和o的函数),给一个O就会有一个不同的brdf,给定一个任意的观察方向O,light transport都会投影出一组完全不同的VECTOR,且vector中的每一个元素都是一个o的函数.

或者直观一点来说,glossy物体有一个很重要的性质,它是和视点有关的.diffuse的物体不管视角如何旋转改变,你看到的Shading point的结果是不会改变的,因为整个Diffuse shading和视角是无关的.

但是glossy不是这样的,glossy是和视角有关的,不同的视角得到的shading result也是不一样的,因此O不一样,L(O)也不一样.所以即使light transport即使投影到了i方向上的基函数,所得到的仍然是一个关于O的函数而不是系数.

Glossy Case

$$L(\mathbf{o}) = \int_{\Omega} L(\mathbf{i}) V(\mathbf{i}) \rho(\mathbf{i}, \mathbf{o}) \max(0, \mathbf{n} \cdot \mathbf{i}) d\mathbf{i}$$

$$L(\mathbf{o}) \approx \sum l_i T_i(\mathbf{o})$$

知乎 @WhySofAr

我们将4D的函数投影在2D上之后,虽然得到的是一个关于O的函数,但是现在这个函数也只是关于O了,因此我们在O的方向上将其投影到SH基函数上.

$$T_i(\mathbf{o}) \approx \sum t_{ij} B_j(\mathbf{o})$$

transport matrix basis function

知乎 @WhyS0fAr

Glossy Case

$$L(\mathbf{o}) = \int_{\Omega} L(\mathbf{i}) V(\mathbf{i}) \rho(\mathbf{i}, \mathbf{o}) \max(0, \mathbf{n} \cdot \mathbf{i}) d\mathbf{i}$$

$$L(\mathbf{o}) \approx \sum l_i T_i(\mathbf{o})$$

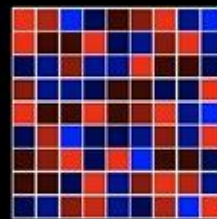
$$L(\mathbf{o}) \approx \sum \left(\sum l_i t_{ij} \right) B_j(\mathbf{o})$$

$$T_i(\mathbf{o}) \approx \sum t_{ij} B_j(\mathbf{o})$$

transport matrix basis function

reflected radiance coefficient \approx light coefficient

 \approx  *



transport matrix

知乎 @WhyS0fAr

因此,light transport上就不再认为得到的是向量了,而是一个矩阵,也就是对于任意一个O都会得到一串VECTOR,最后把所有不同O得到的VECTOR摆在一起,自然而然就形成了一个矩阵.

或者这样理解,我们最后得到的是不同方向上的radiance,自然而然是一个向量,我们将lighting投影到SH上得到的是一个向量,只有向量 * 矩阵 得到的结果才是向量,因此这里只能是矩阵.

可想而知,这样的话将会产生巨大的存储.

Time Complexity

- ◎ #SH Basis : 9/**16**/25
 - ◎ Diffuse Rendering
 - At each point: dot-product of size 16
 - ◎ Glossy Rendering
 - At each point: vector(16) * matrix (16*16)
- 知乎 @WhyS0fAr

正常情况下人们会用多少阶的基函数呢？

基函数个数：9个（三阶） 16（四阶） 25个（五阶）

我们以四阶为例：

Diffuse 物体：每个点需要两个长度为16的向量点乘；(diffuse 情况下一般三阶就足够了)

Glossy 物体：每个点需要16阶向量与16*16矩阵乘。(一般需要高阶一点)

Glossy Rendering Results



No Shadows/Inter



Shadows



Shadows+Inter

- Glossy object, 50K mesh
- Runs at 3.6 fps on 2.2Ghz P4, ATI Radeon 9500 @WhySofAr

这里看出来PRT Glossy比Diffuse 效率要差很多,而当Glossy非常高频的时候,也就是接近镜面反射的情况的时候, PRT就没有那么好用, 我们虽然可以采用更高阶的SH来描述高频信息,但是使用SH甚至远不如直接采样方便。

图二中脚下关于阴影的遮挡充分考虑了visibility (也就是考虑了阴影) 效果就非常好;

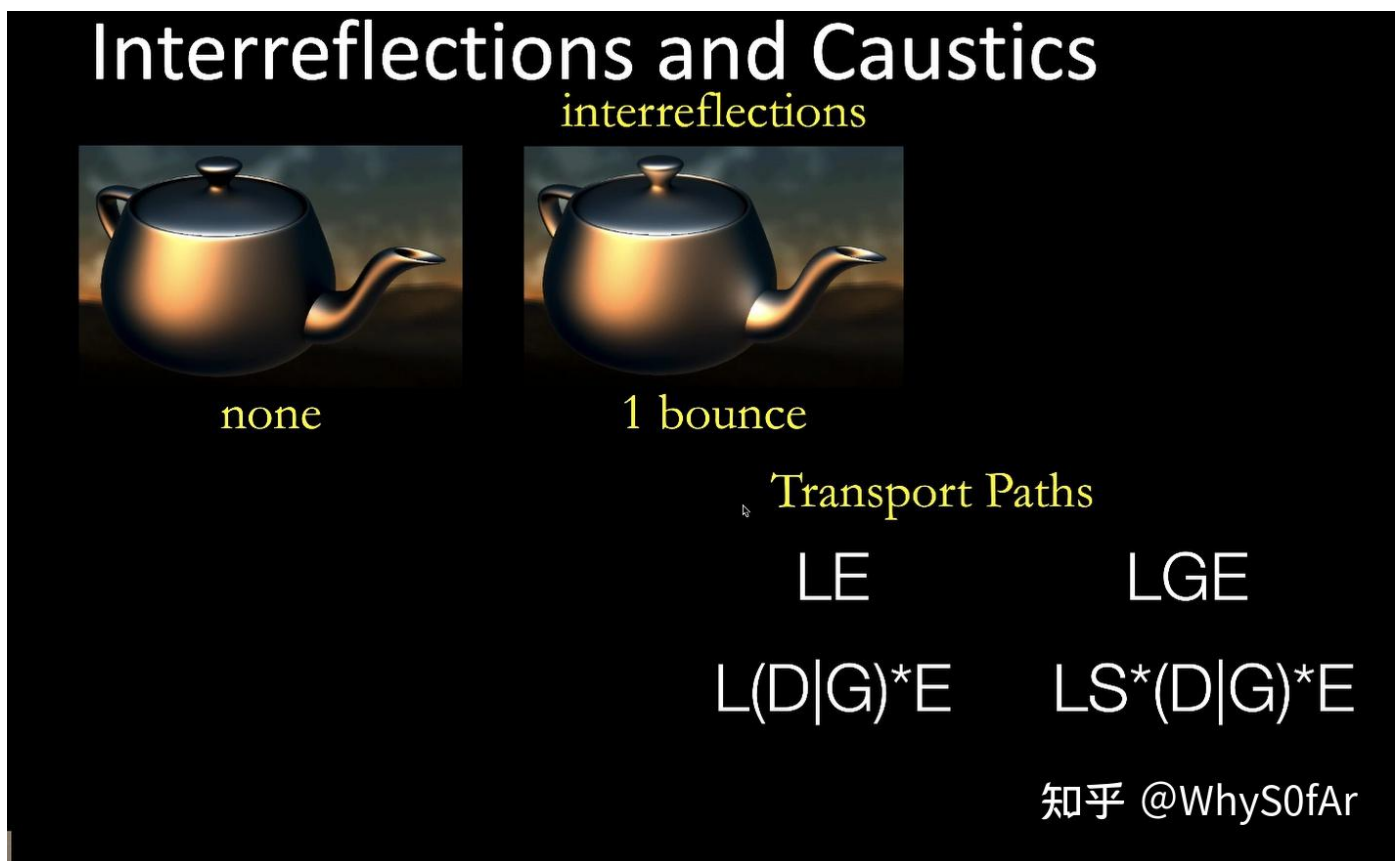
图三考虑了多次光线Bounce的结果。

那么怎样考虑把多次bounce当作Light transport的一部分呢?

我们可以用一系列的表达式来描述不同光线传播的路径都是一种什么类。

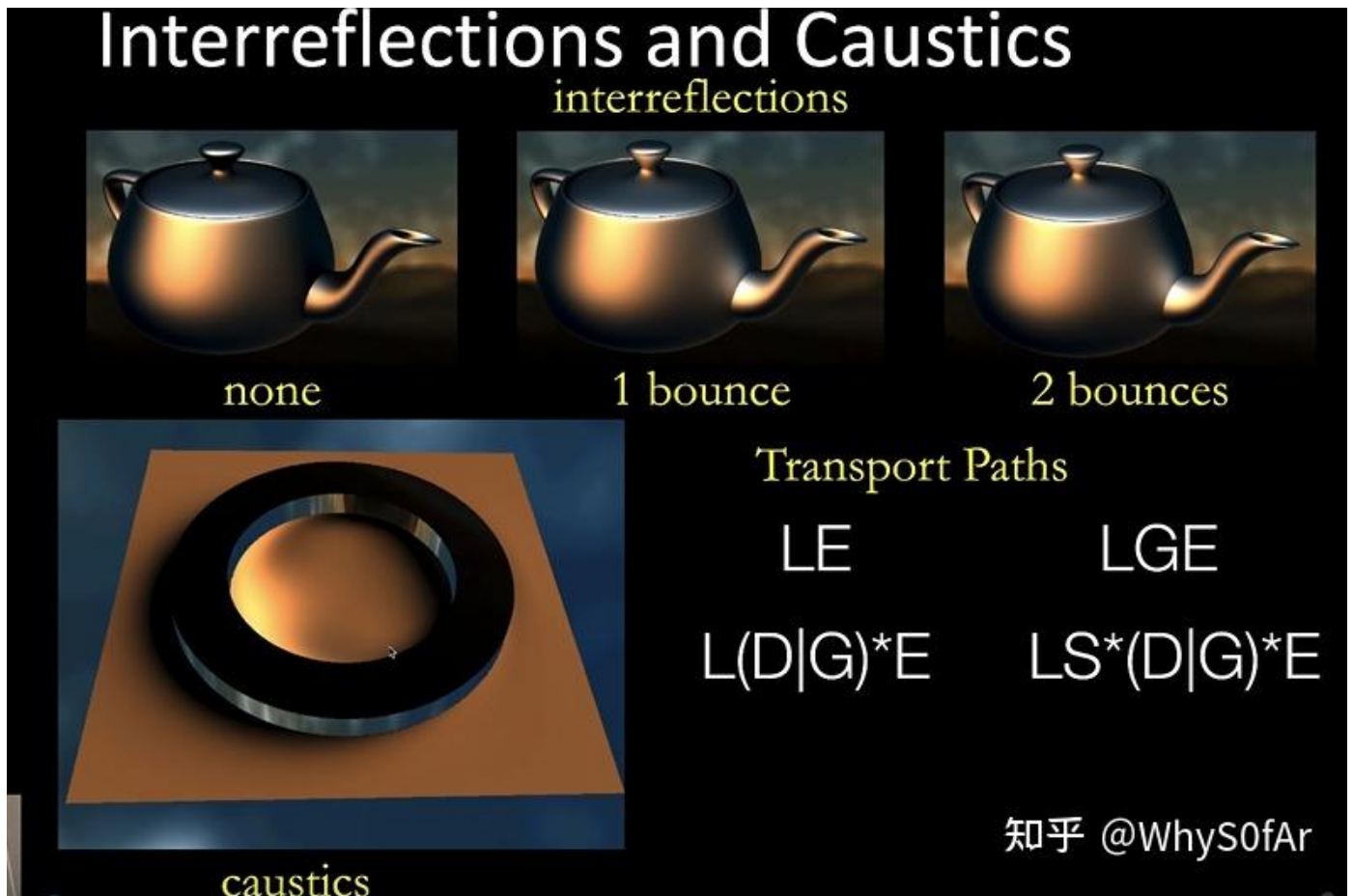
区分材质区分为三种：

1. Diffuse
2. Specular 镜面反射
3. Glossy 介于两者之间



1. LE: Light直接到眼镜；
2. LGE: light打到Glossy物体然后到眼镜。
3. LGGE: 多bounce一次,就是light先打到壶嘴,在bounce到壶身,最后到eye。(L->glossy->glossy->eye)

4. $L(D|G)E$: Light从光源出发,打到一个物体,可能是diffuse也可能是glossy,*表示bounce次数,最后到达EYE.
5. $LS(D|G)*E$: 打到Specular面上,然后聚焦到Diffuse物体上,最后被眼睛看到。也就是caustics.



从上面可以看出所有路径开始都是L最后都是E, 因此我们在运用PRT时候, 拆分为light和light transport之后,不管中间bounce几次,我们只需要预计算算出Light transport就行, 不论多么复杂的bounce,我们只需要计算出light transport就能得出最后的shading result。

所以说，只要采用了PRT的思路，把light和light transport,不管light transport有多复杂,bounce了多少次,只要进行了预计算,渲染时实际跑的时候是很简单的,因为实际跑的时间是与transport的复杂度无关的。

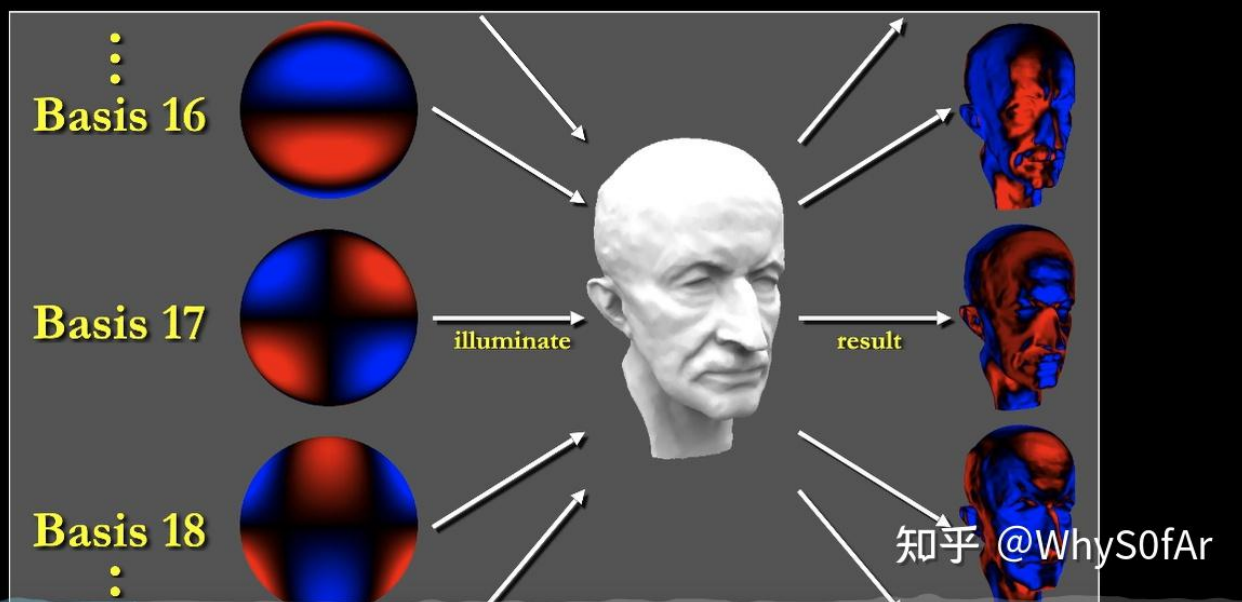
overall,这一页只是为了告诉我们,可以把任意复杂的light transport给预计算出来,只是light transport越复杂在预计算时花费的时间多而实际跑时候是很快的。

那么怎么算呢?

Recall: Precomp. of light transport

light transport
$$T_i \approx \int_{\Omega} B_i(\mathbf{i}) \underline{V(\mathbf{i}) \max(0, \mathbf{n} \cdot \mathbf{i})} d\mathbf{i}$$

◎ Just regular computation with some weird lighting



理解方式1：把light transport和sh基函数做了一个**Product Integral**

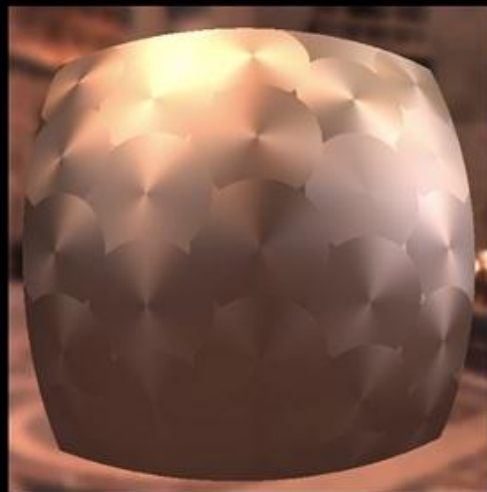
理解方式2：把light transport的预计算看作是一个在一些奇怪lighting下做的渲染过程。

如果我们把基函数看为lighting项,那么这就是rendering equation,我们把light transport投影到basis上,相当于用basis这个Lighting照亮物体,每个basis得到一个渲染图,最后我们进行重建从而得出最后的shaing值.

下图是不同BRDF的渲染结果:

- 各项异性的BRDF
- 普通的BRDF
- 不同位置BRDF不同的物体（BRDF维度增加）。

Arbitrary BRDF Results



Anisotropic BRDFs

Other BRDFs

知乎 @WhySofAr
Spatially Varying

Sloan在02年提出的这个方法（即PRT），使用球谐函数估计光照和光线传输，将光照变成光照系数，将光线传输变成系数或者矩阵的形式，通过预计算和存储光线传输将渲染问题变为每个vertex/shading point：点乘（diffuse表面）、向量矩阵乘法（glossy表面）。

Summary of [Sloan 02]

- ⊙ Approximate Lighting and light transport using basis functions (SH)
 - Lighting -> lighting coefficients
 - light transport -> coefficients / matrices
- ⊙ Precompute and store light transport
- ⊙ Rendering reduced to:
 - Diffuse: dot product
 - Glossy: vector matrix multiplication

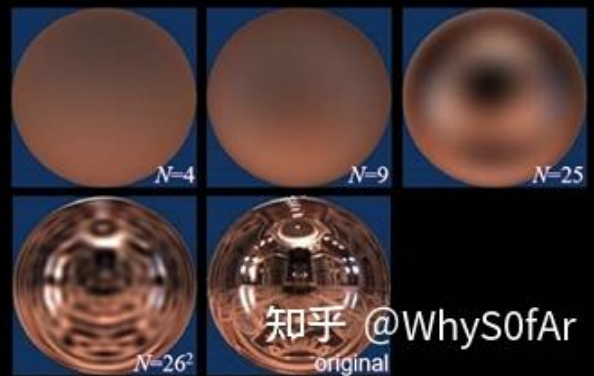
知乎 @WhyS0fAr

但该方法也有其缺点：

- 由于球谐函数的性质，该方法比较适合使用于低频的情况（可用于高频但不合适,如图即使使用了26*26阶的sh仍然得不到比较好的效果）
- 当改变场景或者材质时需要重新预计算light transport，此外预计算的数据比较大。

Limitations [Sloan 02]

- ⊙ Low-frequency
 - Due to the nature of SH
- ⊙ Dynamic lighting, but static scene/material
 - Changing scene/material invalidates precomputed light transport
- ⊙ Big precomputation data



更多的基函数

此外，基函数除了可以使用球谐函数外，还有很多选择，比如 Wavelet、Zonal Harmonics、Spherical Gaussian、Piecewise Constant等。

More basis functions

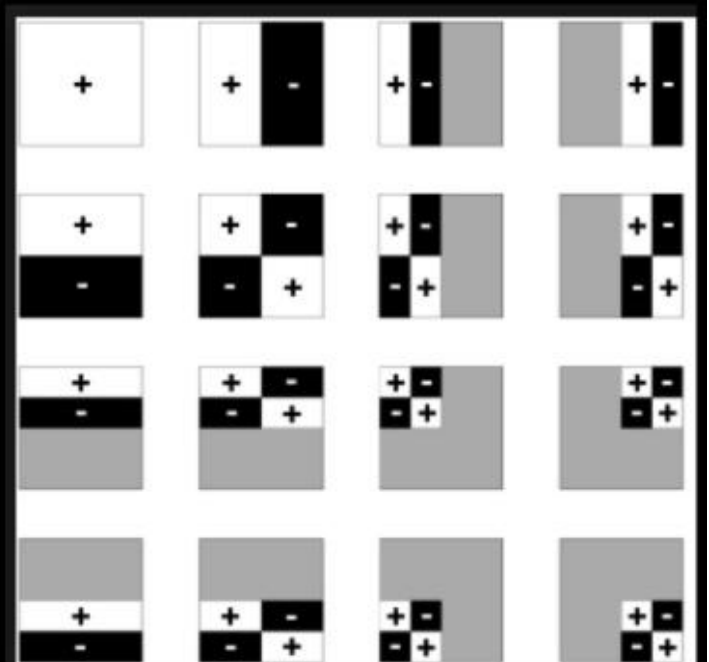
- ◎ Spherical Harmonics (SH)
- ◎ Wavelet
- ◎ Zonal Harmonics
- ◎ Spherical Gaussian (SG)
- ◎ Piecewise Constant

知乎 @WhyS0fAr

这里以Haar小波为例，小波变换的过程就是投影过程，相比于球谐函数对低频内容友好（球谐函数使用少量的基去表示），小波变换可以全频率表示，但是只有很少的系数是非零的。

Wavelet [Ng 03]

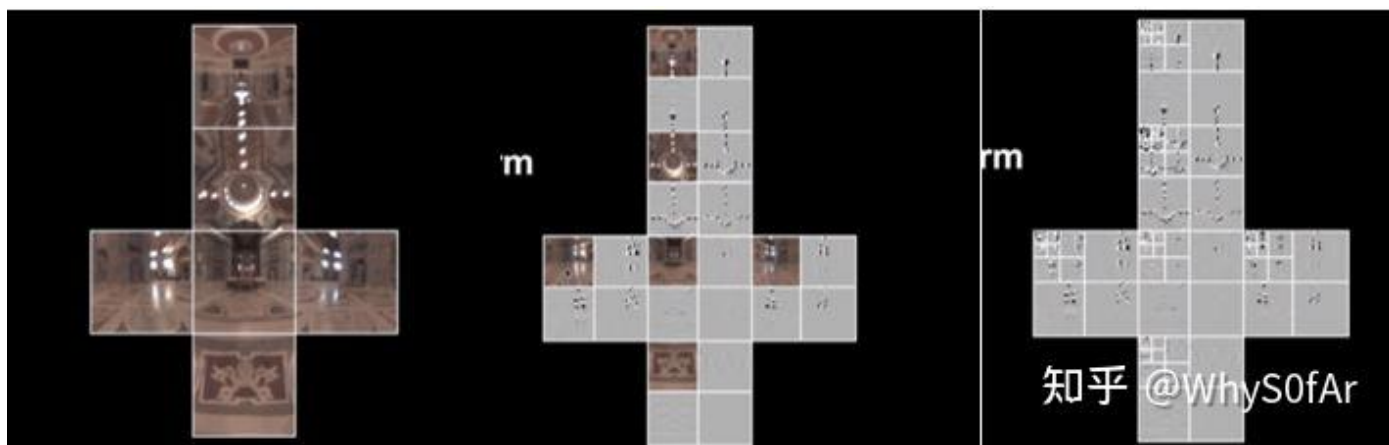
- ◎ 2D Haar wavelet
- ◎ Projection:
 - Wavelet Transformation
 - Retain a small number of non-zero coefficients
- ◎ A non-linear approximation
- ◎ All-frequency representation



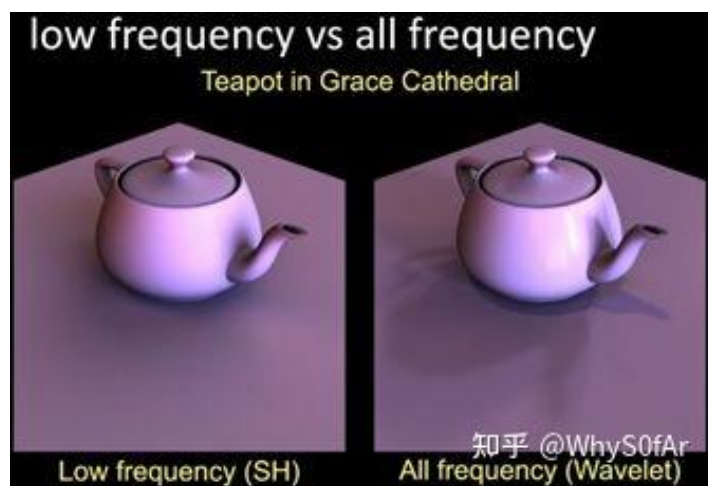
知乎 @WhyS0fAr

由于小波是平面上的函数，为了防止变换后在球面上出现缝隙，所以采用了Cubemap来作为环境光而不是spherical map。

从图中可以看到,小波变化是把每张图的高频信息留在这张图的左下,右上和右下三部分,而把剩余的低频信息放在左上角,左上角的信息可以继续进行小波变换,我们会发现高频的东西很少,对于绝大部分来说是0,不断地进行小波变换可以得到一个很不错的既保留了低频又保留了高频的压缩.



但是小波也有自己的缺陷：不支持旋转（使用球谐函数进行表示时，由于球谐函数具有**simple rotation**的性质，所以支持光源的旋转）。



Real-Time Global Illumination (in 3D)

全局光照是增强真实感的重要部分,因此也是十分复杂的一部分.

从下图中我们会发现,没有任何一处是全黑的看不见东西的,比如书本的下方,蜡烛内部等.也就是光线bounce了很多次才到达我们的眼里,在GAMES101中我们在布林冯模型中,假设 ambient term,认为来自四面八方的间接光照都是相同的,且最后的shading和normal是无关的,这样得到的结果是十分不真实的.

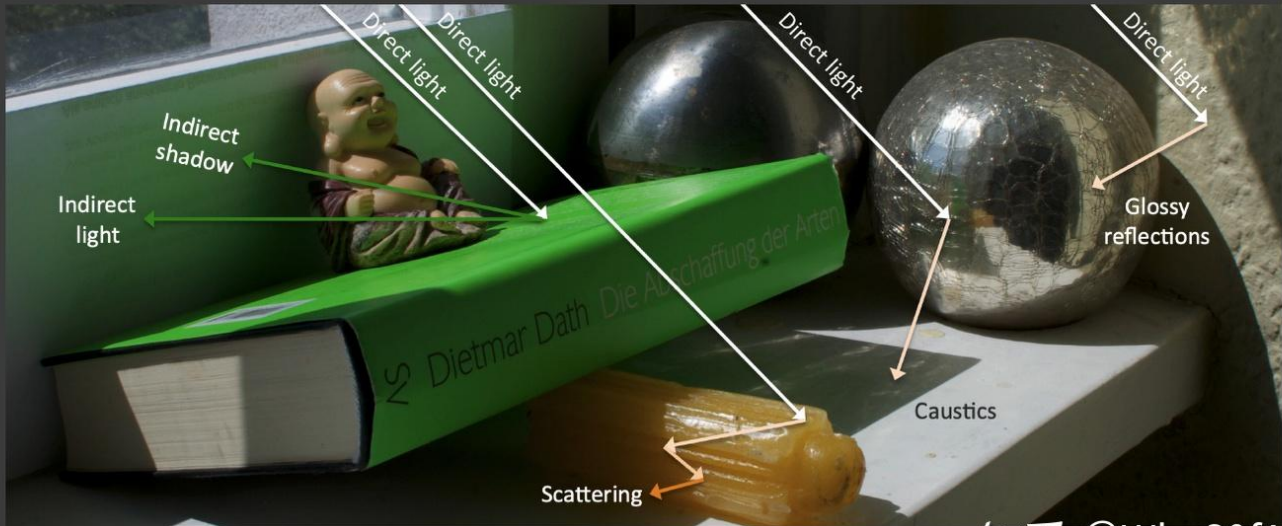
下图中的全局光照就不能这么假设了,大家可以看到书本下方各位置的亮度不同,还可以看到一些caustics等从金属球反射出的间接光,他们之间是不一样的,如果还做Ambient term那种假设从而只提升了一点亮度是无法得到图中的结果的.

Overall,全局光照很复杂.

Reflective Shadow Map(RSM)

Introduction

- Global Illumination (GI) is important but **complex**



[Ritschel et al., The State of the Art in Interactive Global Illumination]

知乎 @WhyS0fAr

在实时渲染中,全局光照是比直接光照多一次**bounce**的间接光照.

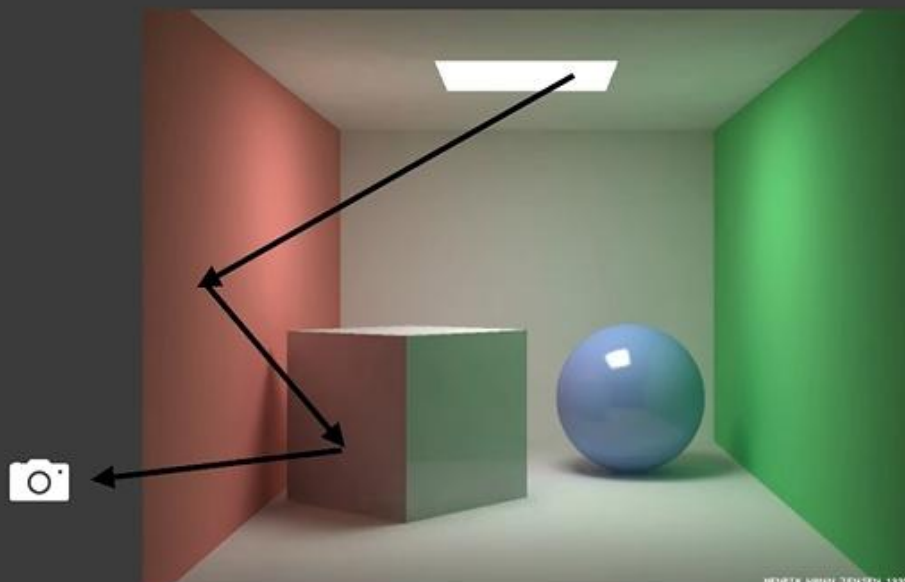
如图中,光线弹射两次,先打到红色的墙壁上,在达到box面上,最后被camera/eye看到,这就是在rtr中要解决的所谓的"全局光照".

我们希望他:

- ①简单->实现起来不麻烦
- ②快速->因为全局光照非常难算

Introduction

- In RTR, people seek simple and fast solutions to **one bounce indirect illumination**



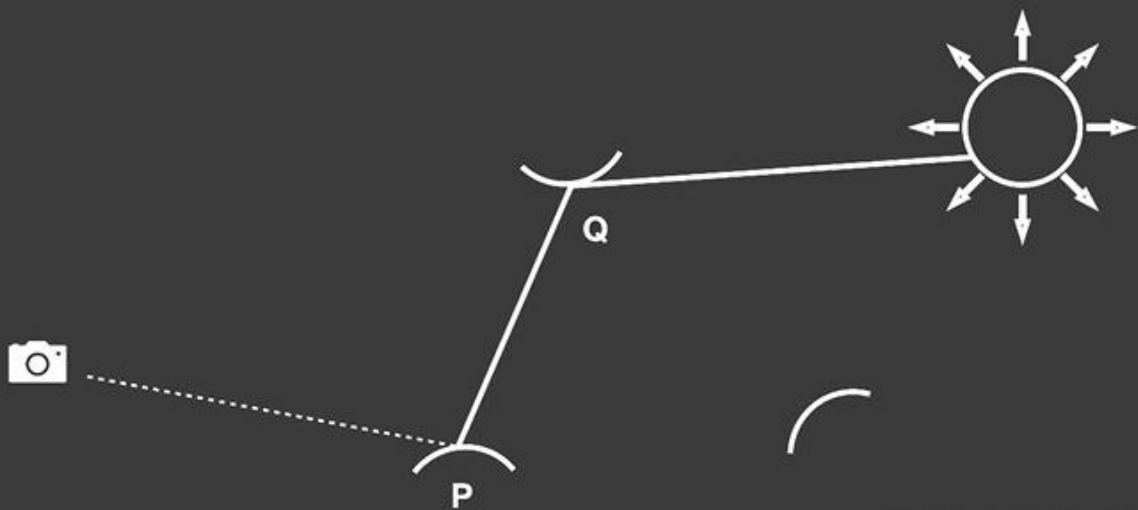
[Image courtesy of Prof. Henrik Wann Jensen]

知乎 @WhyS0fAr

我们通过这幅图来理解什么是一次**bounce**的间接光照:

Understanding

- From GAMES101 (Lecture 16):
Any directly lit surface will act as a light source again



知乎 @WhyS0fAr

[Image courtesy of Prof. Henrik Wann Jensen]

在games101 lecture16中我们讲path tracing时说,在做tracing时,从camera出发打出一条光线到点P,P点又往场景中不同的方向发射光线,如果打到光源则表示接受的是**直接光照**。

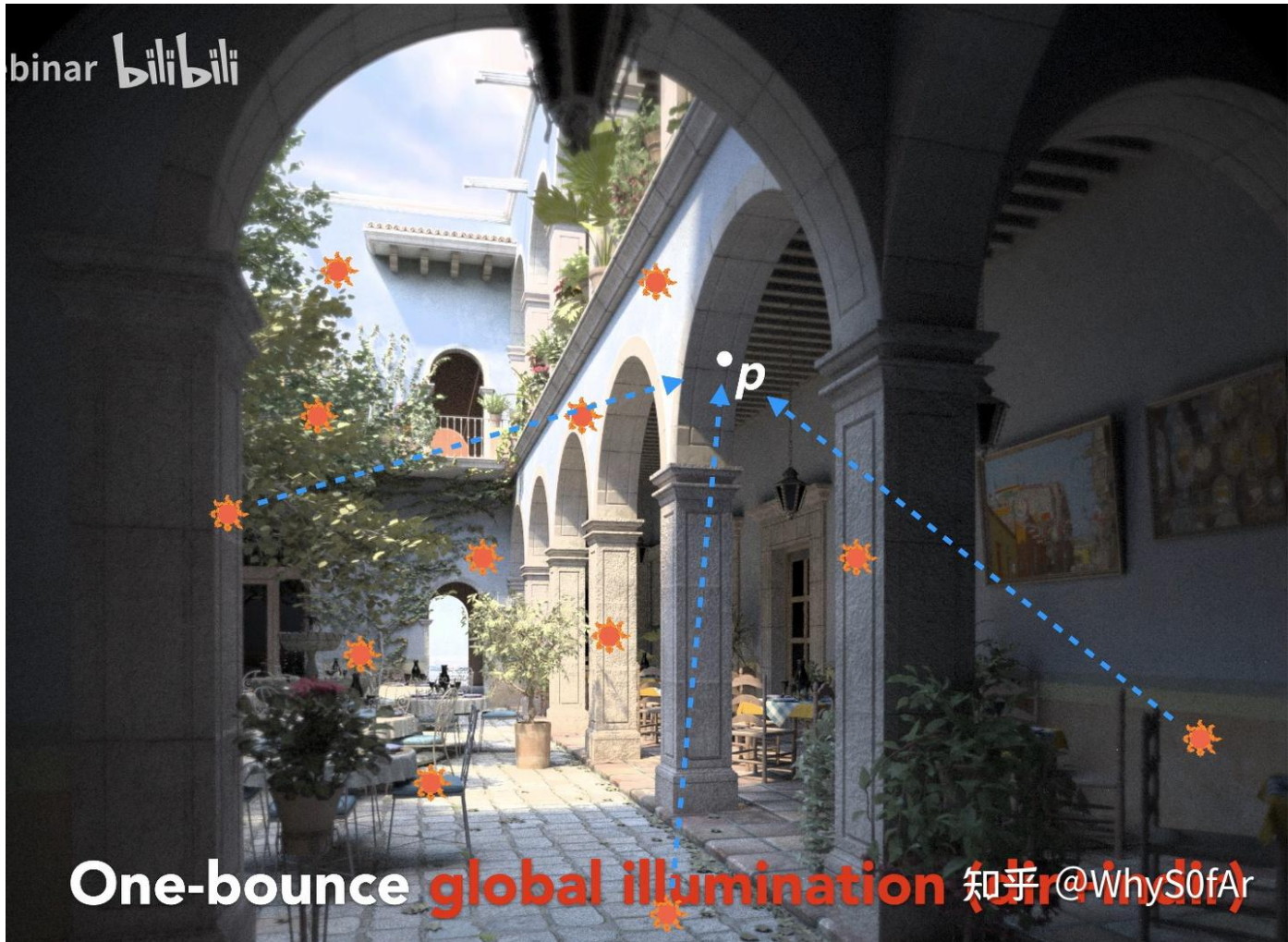
如果没打到光源而是打到了点Q,那么我们认为P点接收到的光照是从Q点反射到P点的Radiance,也就是Q点接收到的直接光照所反射出的光照打到P点上。

点Q接受的是直接光照, 我们则认为接收到直接光照的表面可以被当作次级光源(Secondary Light Source)用自身反射的光照来照亮其他物体。



直接光照

我们可以看到P点在柱子后面,不可能接收到直接光照,我们稀疏的标示出次级光源.



现在的P点就是所有标示次级光源反射出的光照来照亮P点所得到的结果。

Key Observations

- What are needed to illuminate any point p with indirect illumination?
- Q1: Which surface patches are directly lit
 - Hint: what technique tells you this?
- Q2: What is the contribution from each surface patch to p
 - Then sum up all the surface patches' contributions
 - Hint: each surface patch is like an area light

知乎 @WhyS0fAr

那么为了计算点P的shading需要知道什么？

1： 哪些表面会被直接照到？

2： 如何计算每个surface patch对着色点p的贡献？

首先我们来解决问题1----->哪些表面会被直接照到？

这里可以使用shadow map， shadow map上每一个像素可以看成是一个小surface patch。

Reflective Shadow Maps (RSM)

- Q1: Which surface patches are directly lit
 - Perfectly solved with a classic shadow map
 - Each pixel on the shadow map is a small surface patch
- The exact outgoing radiance for each pixel is known
 - But only for the direction to the camera

知乎 @WhyS0fAr

- Assumption
 - Any reflector is diffuse
 - Therefore, outgoing radiance is uniform toward all directions

知乎 @WhyS0fAr

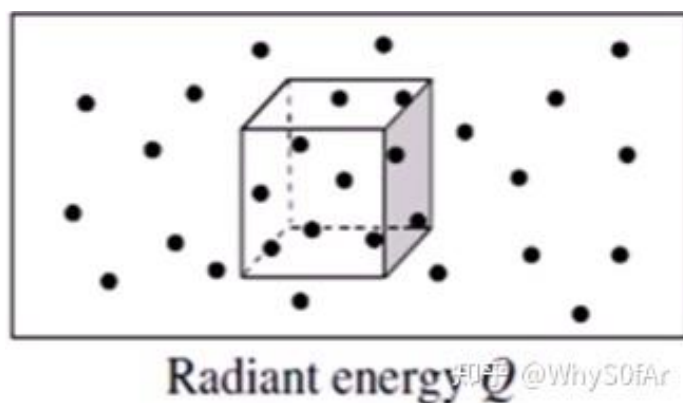
假设:所有的反射物(次级光源)都是diffuse的:

次级光源如果想照亮点P,观察方向是从P点去观察次级光源的,也就是对于不同的P点来说出射方向是unknown的,因此是无法计算P点的shading的,为了不依赖于观察方向,在RSM中我们假设,所有次级光源(reflector)都是diffuse(这里是假设reflector,没有要求receiver也是diffuse),故outgoing radiance在所有方向上都是uniform的,这样不管从camera看过去还是从点p看过去所得到的结果是一样的。

接下来我们来解决问题2----->如何计算每个surface patch对着色点p的贡献?

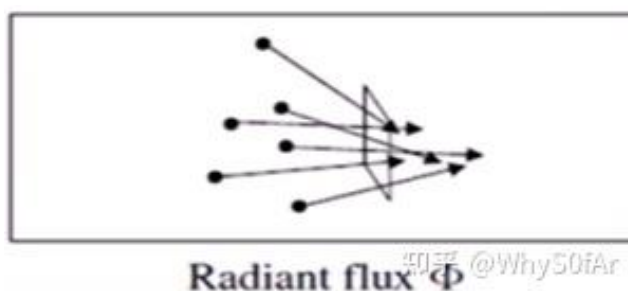
考虑所有surface patch的贡献, 进行求和; 并且每个surface patch可以看成是一个area light。

在此我们需要复习一下GAMES101中的一些辐射度量学的知识:



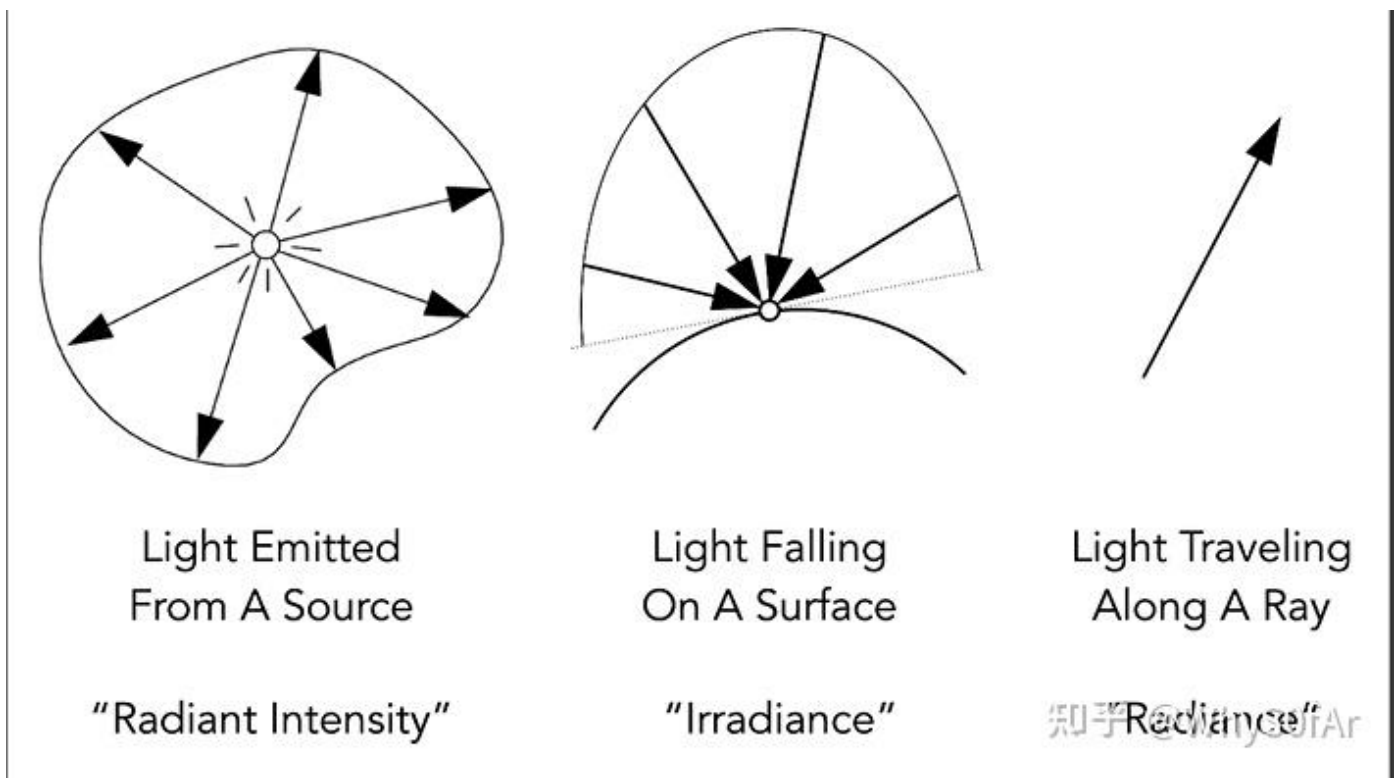
Radiant Engery

- **Radiant Engery:**光能-->一个区域内光子能量的总和,用 Q 来表示,单位是焦耳(J).



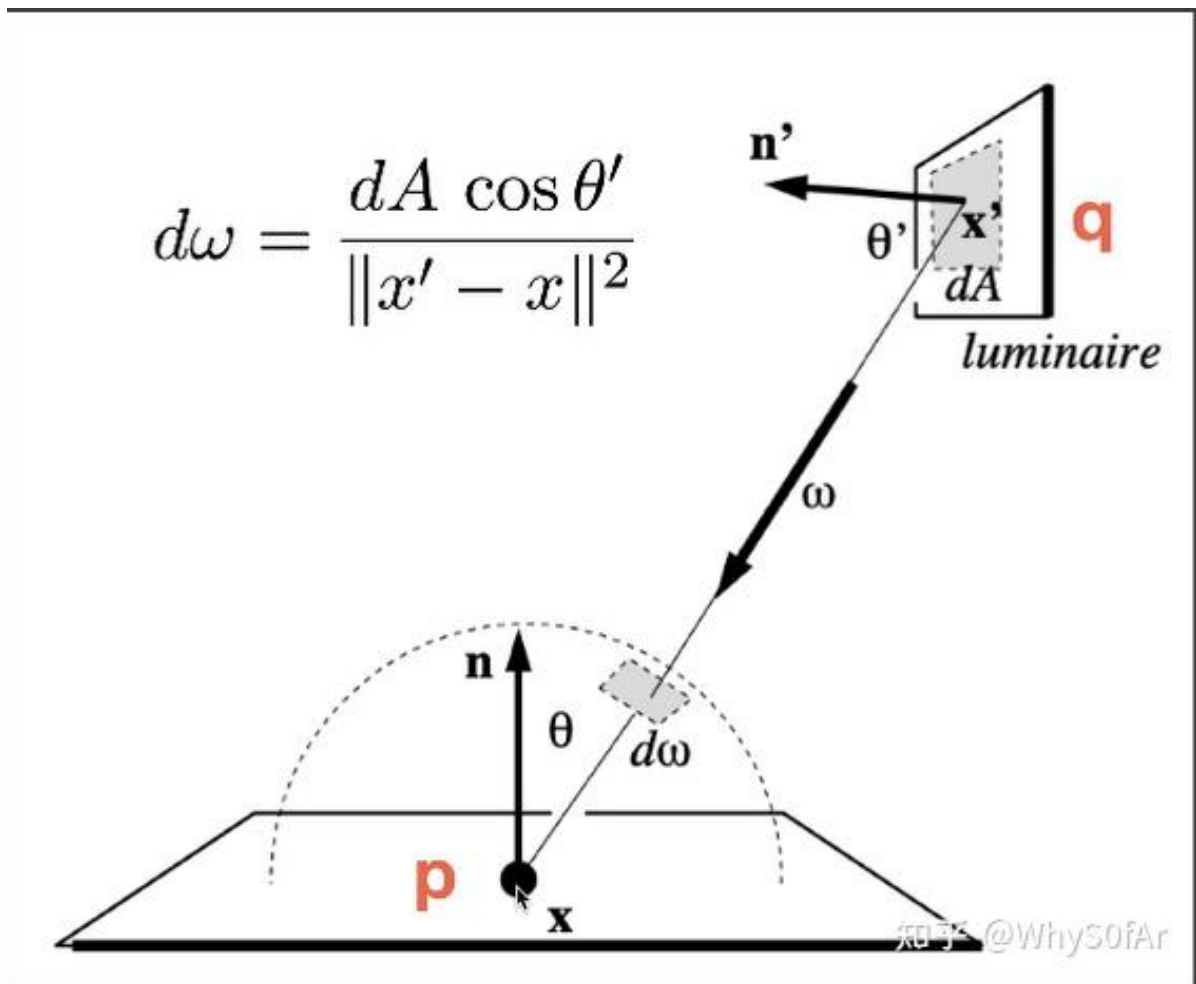
Radiant Flux

- **Radiant Flux:**光通量-->在单位时间内穿过单位截面积的光能.



- **Radiant Intensity:** 一个单位立体角上对应的能量。(单位立体角上的光通量)
- **Irradiance:** 在一个单位面积下对应的能量。(单位面积上的光通量)
- **Radiance:** 一个单位立体角下单位面积上的能量。(个人理解是,单位立体角上通过单位投影面积的光通量)

回到问题本身,我们之前说过每一个小的patch都可能对照亮p点做出贡献,因此我们可以先计算出一个patch做出的贡献,之后用求和的形式将所有patch的贡献加在一起.



我们可以看到 q 是一个patch去照亮点 P

q 其实就是RSM中一个Texel所对应的patch,在games101中我们说过,原本计算 q 对 p 点的贡献,我们应该是对整个立体角进行采样,但是这样的话很浪费很多的sample,为何不直接在light处采样然后去计算 p 点的shading值呢.

$$\begin{aligned}
 L_o(p, \omega_o) &= \int_{\Omega_{\text{patch}}} L_i(p, \omega_i) V(p, \omega_i) f_r(p, \omega_i, \omega_o) \cos \theta_i d\omega_i \\
 &= \int_{A_{\text{patch}}} L_i(q \rightarrow p) V(p, \omega_i) f_r(p, q \rightarrow p, \omega_o) \frac{\cos \theta_p \cos \theta_q}{\|q - p\|^2} dA
 \end{aligned}$$

也就是把立体角的积分变成了对light区域面积的积分,如果当区域足够小的时候dA甚至不用积分,直接相乘后相加就行,现在我们要解的是patch在接受直接光照后反射出的radiance是多少,也就是从q点到p点的radiance,那么如何解呢?

- For a diffuse reflective patch

$$- f_r = \rho / \pi$$

$$- L_i = f_r \cdot \frac{\Phi}{dA} \quad (\Phi \text{ is the incident flux or energy})$$

知乎 @WhyS0fAr

对于每个次级光源点来说,由于我们假设它的brdf是diffuse的,因此次级光源的fr积分后是个常数,此时我们把Li代入到式子中会发现dA刚好会被抵消.之后式子会少去dA,多了个 ϕ_p ,然后 ϕ_p 和 $\frac{\cos\theta_p \cos\theta_q}{||q-p||^2}$ 又组成了下列公式中的Ep与剩余部分结合求出了一个次级光源p对着色点所得到的shading结果,再将积分域中所有的结果加在一起,就是着色点最后被间接光照照亮所得到的shading值.

$$E_p(x, n) = \Phi_p \frac{\max\{0, \langle n_p | x - x_p \rangle\} \max\{0, \langle n | x_p - x \rangle\}}{||x - x_p||^4 \rightarrow 2}. \quad (1)$$

公式求的是次级光源的光线贡献在着色点上的Irradiance,Ep表示次级光源对着色点贡献的入射irradiance.

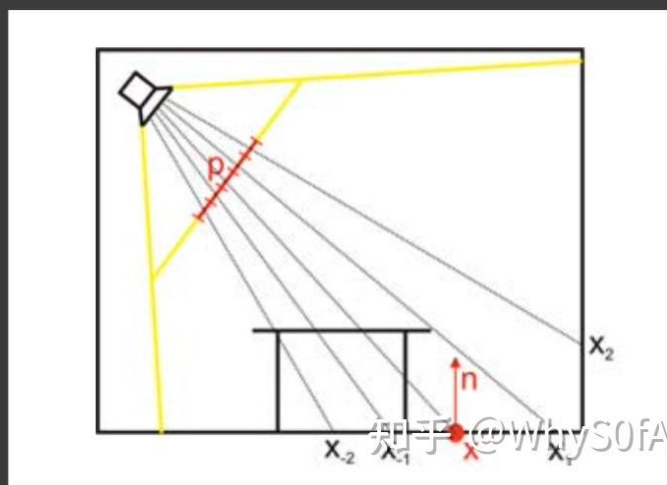
在此仍然存在一些问题:

次级光源能否看到着色点?

- 渲染方程中的 V 是指次级光源到着色点是否可见，如果想解决这个问题，需要为每个次级光源算一次shadow map，这个运算不好算也是很难算的,这个时候我们要学习乌鸦的精神:"难办,那就别办了."
- 值得注意的是原文公式中的分母是4次方，闫老师认为是作者考虑了次级光源发射的光线有衰减，这里应该是2次形式。(他们后来说闫老师吃键盘了,我毕竟还没看到后面所以我先留着.)更新:闫老师确实吃键盘了,但是老师没有错,paper也没有错,因为在分子中有两个 $Xp-X$ ，当把这两个放到分母中后，结果与我们推导的结果一致，同为2次方。

Reflective Shadow Maps (RSM)

- Not all pixels in the RSM can contribute
 - Visibility (still, difficult to deal with)
 - Orientation
 - Distance



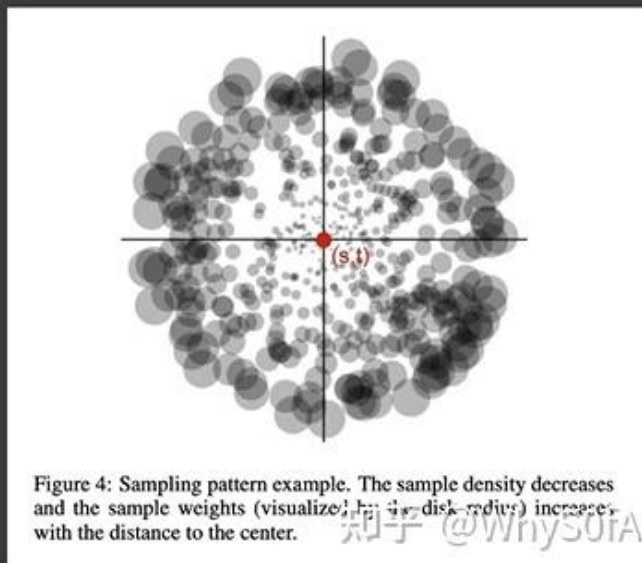
由于可见性、方向性以及距离的不同，对于某一个着色点，认为shadow map上所有的pixel不可能都有贡献：

- -Visibility（仍然非常难算）；
- -方向：比如X-1点在SM中记录的是桌子的表面，而且这个表面的点法线方向是朝上的，因此根本不可能照亮X点；
- -距离：因为远处的次级光源贡献很少，通常只要找距离足够近的次级光源就行了。

因此为了加速这一过程,我们认为在shadow map中着色点 x 的位置和间接光源 x_p 的距离可以近似为它们在世界空间中的距离。所以我们认为，对着色点 x 影响大的间接光源在shadow map中一定也是接近的。

于是我们决定先获取着色点 x 在shadow map中的投影位置 (s, t) ，在该位置附近采样间接光源，多选取一点离着色点近的VPL，并且为了弥补越往外采样数越少可能会带来的问题，引入了权重，越近了权重越小，越远的权重越大。那么对于一个shading point差不多找400个次级光源来计算是比较合适的。如下图所示。

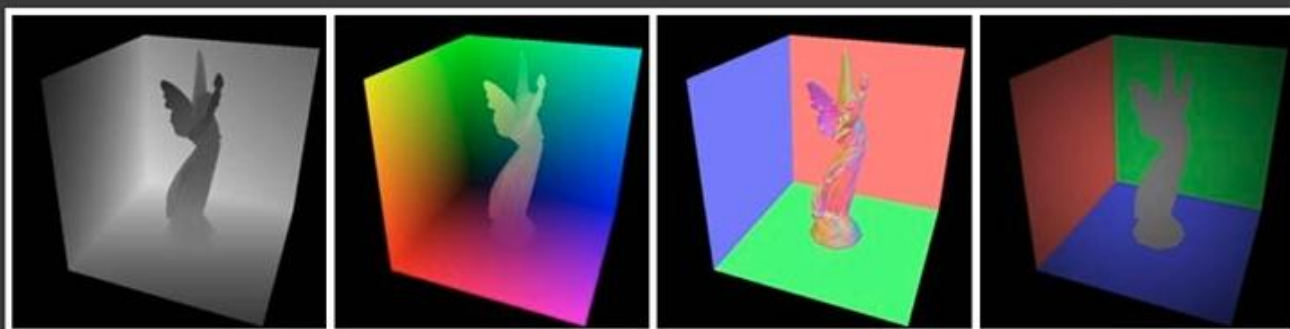
- Acceleration
 - In theory, all pixels in the shadow map can contribute to p
 - Can we decrease the number?
 - Hint: Steps 1 and 3 in PCSS
- Sampling to the rescue



数据存储:

RSM在每一个像素 p 中都需要存储深度值 d_p ，世界坐标 x_p ，法线 n_p ，反射光功率 ϕ_p ，如下图的可视化效果，四个map对应像素 p 的四个参数。

- What is needed to record in an RSM?
 - Depth, world coordinate, normal, flux, etc.



知乎 @WhyS0fAr

RSM效果通常应用于游戏中手电筒的次级光照，如图：

Reflective Shadow Maps (RSM)

- Often used for flashlights in video games
 - Gears of War 4, Uncharted 4, The Last of US, etc.



在此可以更加理解我上文写的对于式子的理解

屋顶某个点亮了的区域就是手电筒直接照亮区域对屋顶那个点的贡献,屋顶点积分的时候积分域就是手电筒直接照亮的位置.

优点:

- 易于实现

缺点:

- 性能随着直接光源数的增加而降低(因为需要计算更多的 shadow map)

- 对于间接光照，没有做可见性检查
- 有许多假设：反射物需要是diffuse等
- 需要在质量和采样率上做一个平衡

全局光照这里,其实是最折磨我的,因为数学底子不好所以在理解paper中的式子时很难让我迷糊过来,还好在看了很多大佬的笔记和咨询了大佬 [@沙滩Beachc](#)之后才让我恍然大悟,计算机图形学真的是很难又很有趣.....我还有很长的路要走啊.

引用:

[1] Reflective Shadow Maps - Carsten Dachsbacher, Marc Stamminger

[2] GAMES202-高质量实时渲染-闫令琪

[3] GAMES202 高质量实时渲染笔记Lecture07: Real-Time Global Illumination (In 3D) -沙滩beachc

[4] 【论文复现】 Reflective Shadow Maps

[Monica的小甜甜: 【论文复现】 Reflective Shadow Maps](#)