

## 目录

Q1-- KD 树、BSP 树、BVH 树.....	3
1.中间节点和叶子节点存储的对象: .....	3
2.中间节点存储信息的数据结构: .....	3
3.对于点云模型和网格模型, 选择哪种树形加速结构存储合适.....	3
Q2-- Whitted-Style Ray Tracing.....	3
1. 解释 Whitted-Style Ray Tracing .....	3
2. Whitted-Style Ray Tracing 当中, 如果初级光线数目 $n$ , 那么第一次弹射光线数目总和 小于多少?第三次弹射光线数目小于多少?.....	3
Q3-- 光线和三角面片求交.....	4
1. 采用 Möller Trumbore Algorithm 算法, 判断一条光线和三角面片求交的公式是什么? .....	4
2. 公式求解出的三个参数需要满足什么条件, 才能判断光线在三角形内部有交点? ...	4
Q4-- 光线和 AABB 包围求交.....	4
Q5-- 立体角 .....	4
1.右图区域 A 对应的立体角怎么计算? A 为该区域面积。.....	4
2.半球面立体角多大? .....	4
Q6-- 辐射度量学 .....	5
Q6-- 渲染方程.....	6
Q7-- Monte Carlo Path Tracing.....	7
Q8-- 微表面 BRDF 模型 .....	8
1. 解释 F.....	8
2. 解释 D, G .....	8
Beckmann 模型 .....	8
GGX 模型 .....	8
The Kulla-Conty Approximation .....	10
Q9-- BRDF 的性质.....	11
Q10-- 查询纹理图上一块区域均值.....	11
方法一—MipMap:.....	11
方法二-- Summed Area Tables (SAT).....	11
Q10-- Shadow Volume.....	12
1. 概论 .....	12
2. z-pass.....	12
3. z-fail.....	12
Q11-- Shadow Mapping .....	13
1. 是一种 2-pass 的算法.....	13
2. 自遮挡问题.....	13
3. 走样问题.....	13
Q12—PCF&PCSS.....	15
1. PCF 思路 .....	15
2. 由 PCF 到 PCSS.....	15
Q13-- Variance soft shadow mapping(VSSM).....	16

1. 解决 PCSS 的第三步 Percentage Closer Filtering 的速度慢 .....	16
2. 解决 PCSS 的第一步需要将范围内所有像素的深度走一遍从而求平均遮挡深度 Average Blocker Depth 的速度慢.....	16
Q14-- Distance Field Soft Shadows (DFSS) .....	17
Q14-- Shading from Environment Lighting.....	18
1. 环境光照的基本概念和 split-sum .....	18
2. 球谐函数 (Spherical Harmonics, SH) .....	18
3. Precomputed Radiance Transfer (PRT)方法 .....	19
Q15-- Real-Time Global Illumination (in 3D).....	20
Reflective Shadow Maps (RSM).....	20
Light Propagation Volumes (LPV).....	21
Voxel Global Illumination (VXGI).....	22
Q16-- Real-Time Global Illumination (screen space).....	23
Screen Space Ambient Occlusion (SSAO).....	23
Screen Space Directional Occlusion (SSDO) .....	24
Screen Space Reflection (SSR) .....	25
Q17-- RTRT.....	26
1. SPP (Sample per pixel)。 .....	26
2. 降噪方案: 时间上的 (Temporal) 滤波.....	26
(1) 物体在两帧之间的对应关系 .....	26
(2) Temporal filtering .....	26
3. 联合双边滤波 (Cross/Joint bilateral filtering) .....	27
(1) 大滤波核 (Large Filters) .....	27
4. Spatiotemporal Variance-Guided Filtering (SVGF) .....	28
参考资料.....	29

## Q1-- KD 树、BSP 树、BVH 树

### 三维空间， KD 树、BSP 树、BVH 树

- 1) 介绍中间节点和叶子节点分别存储的对象；
- 2) 写出中间节点存储信息的数据结构；
- 3) 对于点云模型和网格模型，分别用哪种树形加速结构存储合适？为什么？

解答:

#### 1.中间节点和叶子节点存储的对象:

- KD 树中，中间节点存储一个分割超平面，叶子节点存储实际的对象。
- BSP 树中，中间节点存储分割平面，叶子节点存储物体列表。
- BVH 树中，中间节点存储包围盒，叶子节点存储物体列表和包围盒。
- 每种树的中间节点都还需要存储指向子节点的指针。

#### 2.中间节点存储信息的数据结构:

- KD 树中，中间节点存储分割超平面，一般是一条与坐标轴平行的线或面。用一个整数指定划分维度，另一个浮点数指定划分点的位置。
- BSP 树中，中间节点存储一个平面，由法向量和原点到平面的距离表示。可以使用向量和标量来存储这些信息。
- BVH 树中，中间节点存储一个包围盒，表示其子节点的空间范围。包围盒可以由其最小和最大顶点坐标表示，也可以使用中心点、半径等其他方式表示。

#### 3.对于点云模型和网格模型，选择哪种树形加速结构存储合适

- 对于点云模型，KD 树和 BVH 树都是有效的选择。如果点云较密集且分布均匀，则 KD 树的效率更高，因为它可以更好地划分空间。如果点云较稀疏，则 BVH 树可能更适合，因为它可以更好地处理不规则形状和间隙。
- 对于网格模型，BVH 树通常是更好的选择，因为它可以有效地处理不规则形状和间隙，并且具有较好的层次结构。BSP 树也可以用于网格模型，但相对于 BVH 树，它需要更多的计算时间和内存空间。

## Q2-- Whitted-Style Ray Tracing

解答:

### 1. 解释 Whitted-Style Ray Tracing

光线和物体求交之后，只考虑完美的反射和折射，所以在交点处不需要做光线的重要性采样，每个交点处，可能会有两条次级光线，一条反射，一条折射（如果非透明物体，则只有反射）。光线达到漫反射表面，传播结束。

### 2. Whitted-Style Ray Tracing 当中，如果初级光线数目 $n$ ，那么第一次弹射光线数目总和小于多少？第三次弹射光线数目小于多少？

(1)  $< 2n$

(2)  $< (2^3)n$

### Q3-- 光线和三角面片求交

光线和三角面片求交: 光线用起始点  $O$  和方向  $d$  表示为  $r(t) = o + td$ . 三角面片三个顶点位置记为  $P_0, P_1, P_2$ .

解答:

1. 采用 Möller Trumbore Algorithm 算法, 判断一条光线和三角面片求交的公式是什么?

$$\vec{O} + t\vec{D} = (1 - b_1 - b_2)\vec{P}_0 + b_1\vec{P}_1 + b_2\vec{P}_2$$

, 依据该公式求解出  $t, b_1, b_2$

2. 公式求解出的三个参数需要满足什么条件, 才能判断光线在三角形内部有交点?

$$t > 0, b_1 > 0, b_2 > 0, (1 - b_1 - b_2) > 0$$

### Q4-- 光线和 AABB 包围求交

光线用起始点  $O$  和方向  $d$  表示为  $r(t) = o + td$ . AABB 包围盒用 6 个面坐标表示--  $x=x_0, x=x_1 (x_0 < x_1)$   $y=y_0, y=y_1 (y_0 < y_1)$   $z=z_0, z=z_1 (z_0 < z_1)$ . 如何判断光线是否和 AABB 包围盒相交? 写出计算公式, 以及所求解出的参数的取值范围。

解答:

$$t_{\min_x} = (x_0 - O_x)/d_x, \quad t_{\max_x} = (x_1 - O_x)/d_x; \quad t_{\min_y} = (y_0 - O_y)/d_y, \quad t_{\max_y} = (y_1 - O_y)/d_y;$$
$$t_{\min_z} = (z_0 - O_z)/d_z, \quad t_{\max_z} = (z_1 - O_z)/d_z.$$

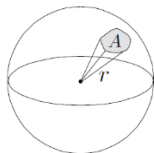
$$t_{\text{enter}} = \max\{t_{\min_x}, t_{\min_y}, t_{\min_z}\}, t_{\text{exit}} = \min\{t_{\max_x}, t_{\max_y}, t_{\max_z}\}$$

当满足  $t_{\text{enter}} < t_{\text{exit}} \ \&\& \ t_{\text{exit}} \geq 0$  时, 光线和 AABB 包围盒有交点

### Q5-- 立体角

解答:

1. 右图区域  $A$  对应的立体角怎么计算?  $A$  为该区域面积。



$$\text{Solid Angle } \Omega = \frac{A}{r^2}$$

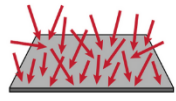
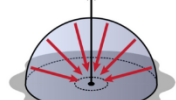
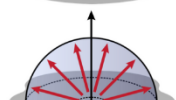

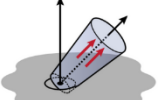
2. 半球面立体角多大?

(1) 球面 **Sphere has  $4\pi$  steradians**

(2) 半球面  $\pi/2$

## Q6-- 辐射度量学

解答:

• flux:	$\Phi(A)$	$\left[ \frac{J}{s} = W \right]$	
• irradiance:	$E(\mathbf{x}) = \frac{d\Phi(A)}{dA(\mathbf{x})}$	$\left[ \frac{W}{m^2} \right]$	
• radiosity:	$B(\mathbf{x}) = \frac{d\Phi(A)}{dA(\mathbf{x})}$	$\left[ \frac{W}{m^2} \right]$	
• intensity:	$I(\vec{\omega}) = \frac{d\Phi}{d\vec{\omega}}$	$\left[ \frac{W}{sr} \right]$	
• radiance:	$L(\mathbf{x}, \vec{\omega}) = \frac{d^2\Phi(A)}{\cos\theta dA(\mathbf{x})d\vec{\omega}}$	$\left[ \frac{W}{m^2 sr} \right]$	

(1) Radiant flux (power): the energy emitted, reflected, transmitted or received, **per unit time**.

(单位时间能量 → 功率)

(2) Radiant(luminous) **Intensity**: the power per unit solid angle (立体角) emitted by a point light source.

(3) Irradiance: power per unit area (perpendicular/ projected)

**(4) Radiance: power per unit solid angle, per projected unit area.**

Irradiance 和 Radiance 之间的区别就在于是否有方向性

- Irradiance: total power received by area dA 四面八方的光线积分起来
- Radiance: power received by area dA from "direction" d $\omega$

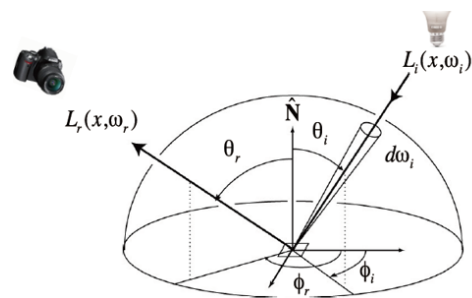
## Q6-- 渲染方程

解答:

The Rendering Equation

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{\Omega^+} L_i(p, \omega_i) f_r(p, \omega_i, \omega_o) (n \cdot \omega_i) d\omega_i$$

Reflected Light (Output Image)	Emission	Incident Light (from light source)	BRDF	Cosine of Incident angle
-----------------------------------	----------	--	------	-----------------------------



$p$ : 着色点 (反射点)  
 $n$ : 反射点的法向  
 $\omega_o$ : 出射方向  $\omega_i$ : 入射方向  
 $L_o$  表示在位置  $p$  处沿着  $\omega_o$  出射方向的辐射亮度 (radiance)  
 $L_e$  表示着色点在出射方向上的自发光  
 $L_i$  表示入射辐射亮度,  
 $f_r$  表示双向反射分布函数BRDF.

## Q7-- Monte Carlo Path Tracing

写出 Monte Carlo Path Tracing 伪代码，并简要加注说明。

解答:

```
1. ray_generation (camPos, pixel) // camPos--相机位置, pixel--被渲染的像素
2.     Uniformly choose N sample positions within the pixel // 在像素内均匀采样 N 个点
3.     pixel_radiance = 0.0 // 像素的 radiance 初始值为 0
4.     For each sample in the pixel
5.         shoot a ray r (camPos, cam_to_sample) // 计算相机位置到采样点的射线方向
6.         If ray r hit the scene at p
7.             pixel_radiance += 1 / N * shade(p, sample_to_cam) // 如果有交点, 计算交点处的 radiance, 并累加到像素的 radiance 中
8.     Return pixel_radiance
```

```
1. shade(p, wo) // p--交点, wo--出射方向
2.     // 来自于光源的贡献
3.     Uniformly sample the light at x' ( pdf_light = 1 / A ) // 均匀采样光源上的一个点 x'; 光源采样概率 = 1 / light_area
4.
5.     shoot a ray from p to x' // 光源上采样点到交点
6.     If the ray is not blocked in the middle // 如果从交点到采样点的射线不被遮挡
7.         L_dir = L_i * f_r * cos  $\theta$  * cos  $\theta'$  / |x' - p|^2 / pdf_light // 计算来自光源的贡献
8.
9.     // 来自于其他反射面的贡献
10.    L_indir = 0.0 // 初始值为 0
11.    Test Russian Roulette with probability P_RR // 对俄罗斯轮盘的解释: 手动指定一个概率 P_RR, if( $\xi > P_{RR}$ ) return 0.0
12.    Uniformly sample the hemisphere toward wi (pdf_hemi = 1 / 2pi) // 均匀采样半球
13.    Trace a ray r(p, wi) // 创建从交点出发的光线
14.    If ray r hit a non-emitting object at q
15.        L_indir = shade(q, -wi) * f_r * cos  $\theta$  / pdf_hemi / P_RR // 递归计算来自其他物体的贡献
16.
17.    Return L_dir + L_indir // 总
```

## Q8-- 微表面 BRDF 模型

### 微表面 BRDF 模型

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

解答:

#### 1. 解释 F

(1)F: Fresnel term

①Reflectance depends on incident angle (and polarization of light) [翻译: 反射率取决于入射角 (和光的偏振)]

②

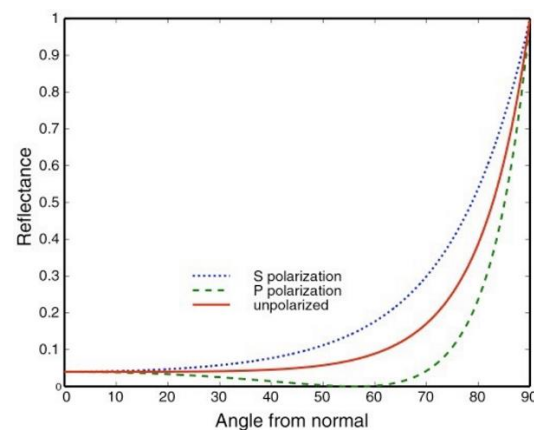


图 1: 绝缘体

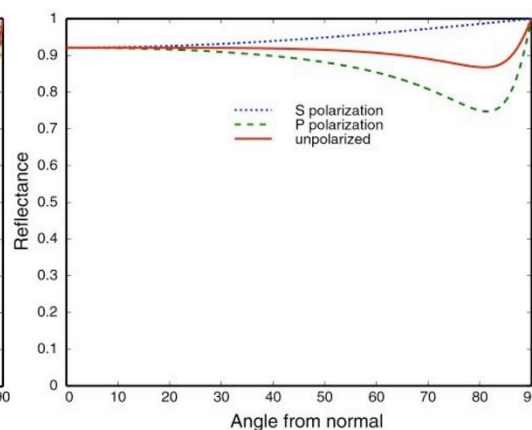


图 2: 导体

③近似: Schlick's approximation

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

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

#### 2. 解释 D, G

(1)D(h): 基于 h 的法线 Distribution, h 是 half vector

①当朝向比较集中的时候会得到 Glossy 的结果,如果朝向特别集中指向时认为是 specular 的.当分布杂乱无章时候, 因此认为表面也非常复杂,得到的结果也就类似 diffuse 的.

②我们把为表面法线分布的函数定义为 Normal Distribution Function (NDF)

#### Beckmann 模型

②-1) Beckmann 模型

The NDF term: e.g. Beckmann distribution

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

$\alpha$ 描述的是法线粗糙程度, 粗糙程度这个值越小, 表面就越光滑

$\theta_h$ : 微表面半程向量法线与宏观表面法线 (0, 0, 1) 方向 n 的夹角

#### GGX 模型

②-2) GGX 模型 (TR 模型)



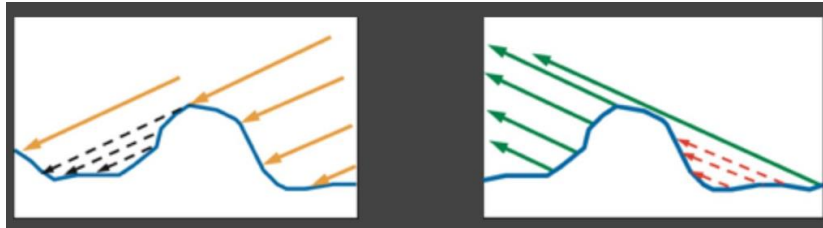
$$NDF_{GGX}(n, h, \alpha) = \frac{\alpha^2}{\pi((n \cdot h)^2(\alpha^2 - 1) + 1)^2}$$

Beckmann 的高光会逐渐消失,而 GGX 的高光会减少而不会消失,这就意味着高光的周围我们看到一种光晕的现象.

GGX 除了高光部分,其余部分会像 Diffuse 的感觉.

(2)G: shadowing-masking term, 微表面互相遮挡的损耗

①解决的问题就是微表面之间的自遮挡问题,尤其是在角度接近 grazing angle 时.



左边这种从 light 出发发生的微表面遮挡现象叫做 Shadowing; 右边这种从 eye 出发发生的微表面遮挡现象被称为 Masking.

②对于

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

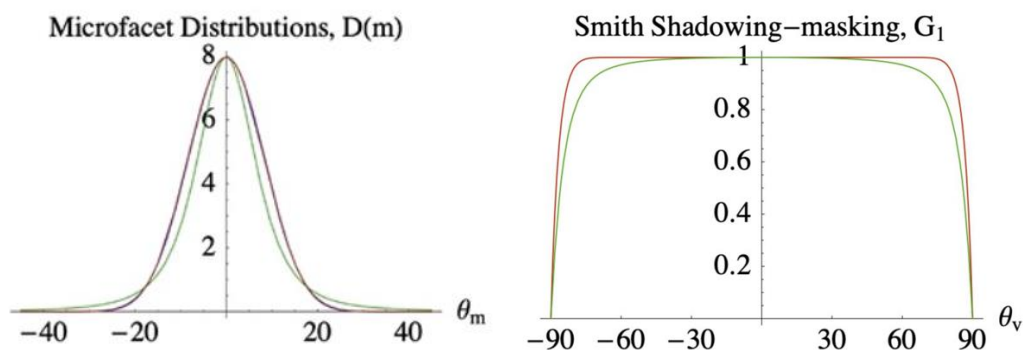
分母中存在的法线与入射方向  $(n, i)$  的点乘, 法线与出射方向  $(n, o)$  的点乘, 当在 grazing angle 时入射方向、出射方向与法线角度接近  $90^\circ$ , 因此点乘结果会非常小接近 0, 分子除以一个接近于 0 的值会导致结果变的巨大, 就会导致我们看到的这张图整个外圈是白的。

③The Smith Shadowing-Masking

在 The Smith Shadowing-Masking 中,我们把 shadowing 和 masking 分开考虑

$$G(i, o, m) \approx G_1(i, m)G_1(o, m)$$

对比



在两种不同 NDF 下预测出的 G 项有一些细微差别但其实相差不大. 垂直入射的时候 Shadowing-Masking 不起作用, 在 grazing angle 时,G 项变得非常小接近于 0,从而我们解决了分子除以分母导致函数结果过大,整体发白的问题.

④问题: 由于没有考虑光线在表面上的多次弹射, 只考虑了微表面遮挡的情况, 当粗糙度越来越大的时候, 能量是不守恒的,因此才导致了粗糙度增大引起了能量损失这一现象.

把丢失的能量补回去: The Kulla-Conty Approximation-- 补上一个 brdf, 其积分起来为 1-E

### The Kulla-Conty Approximation

#### ④-1) 无颜色

- Adding everything up, we have the color term

- Which will be directly multiplied on the **uncolored additional BRDF**

$$\frac{F_{avg} E_{avg}}{1 - F_{avg} (1 - E_{avg})}$$

#### ④-1)-1

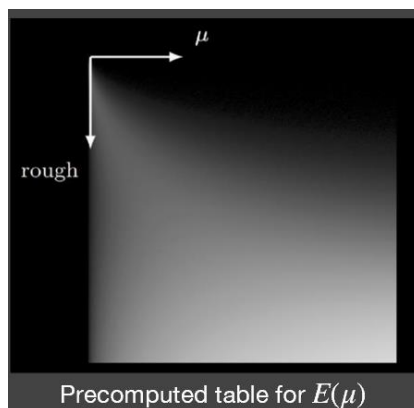
平均菲涅尔项的计算就是计算在所有角度下, 菲涅尔项的平均值

$$F_{avg} = \frac{\int_0^1 F(\mu) \mu d\mu}{\int_0^1 \mu d\mu} = 2 \int_0^1 F(\mu) \mu d\mu$$

#### ④-1)-2

- But  $E_{avg}(\mu_o) = 2 \int_0^1 E(\mu_i) \mu_i d\mu_i$  is still unknown (as analytic)

通过打表直接查询



#### ④-2) 有颜色

#### ④-2)-1

有颜色就意味着, 物体发生了能量吸收的情况, 也就是有额外能量损失, 也就是单次反射的积分结果不是 1. 因此我们可以先考虑没有颜色的情况, 然后再去考虑他的颜色是什么。

#### ④-2)-2

做法: 颜色项\*不带有颜色的 BRDF 上

$$\frac{F_{avg} E_{(u)}}{1 - F_{avg} (1 - E_{(u)})}$$

## Q9-- BRDF 的性质

解答:

- 1) BRDF 非负
- 2) 线性性质: 一个 BRDF 函数可以分解为几个函数相加
- 3) 可逆性: 交换入射和出射方向, 得到的 BRDF 值相同
- 4) 能量守恒: 表示反射多少能量
- 5) 各向同性 BRDF: BRDF 值和入射光线&出射光线的相对方位角相关, 即三维 BRDF
- 6) 各项异性 BRDF: BRDF 值和入射光线&出射光线的绝对方位角相关, 四维 BRDF

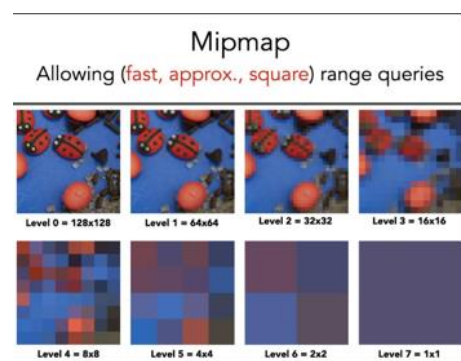
## Q10-- 查询纹理图上一块区域均值

查询纹理图上一块区域均值方法

解答:

方法一—MipMap:

不准, 还只能在正方形里进行. 由于他要做插值, 因此即便是方形有时也会不准确.



方法二-- Summed Area Tables (SAT)

SAT 是百分百准确的一个数据结构. SAT 的出现是为了解决范围查询(在区域内快速得到平均值), 并且, 范围内求平均值是等价于范围内求和的, 毕竟总和除以个数=平均值.

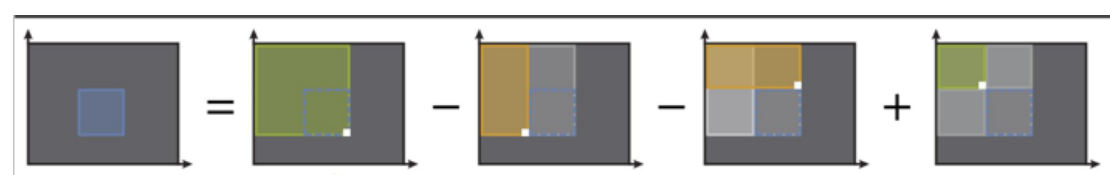
1. 在 1 维情况下其实就是一维数组. SAT 这种数据结构就是做了预处理, SAT 上任意的一个元素就等于原来数组从最左边的元素加到这个元素的和, 如图



2. 在 2 维 (二维数组) 情况下, 有一个任意矩形 (横平竖直), 蓝色矩形内的总和为两个绿色矩形内的总和减去两个橙色矩形内的总和。同样可以采用 SAT 的方法, 做一个表, 做出从左上角到这个元素的和。因此只需要查表 4 次就可以得出精准的区域求和。

在 2 维情况下, 可以通过建立 m 行中每行的 SAT, 再建立 n 列中每列的 SAT, 最终可以获得一个 2 维的 SAT.

由于 gpu 的并行度很高, 行与行或列与列之间的 sat 可并行, 因此具有  $m \times n$  的时间复杂度。



## Q10-- Shadow Volume

解答:

### 1. 概论

Shadow Volume Method (影子体积法) 是一种计算实时阴影的技术, 它的基本思想是在三维空间中创建一些几何体, 这些几何体和光源一起构成的几何体是有空间关系的, 可以用来计算阴影。

Shadow Volume Method 有两种主要的实现方式: Depth Pass Method (深度通道法) 和 Depth Fail Method (深度失败法)。

Depth Pass Method (深度通道法) 是通过渲染深度缓存来实现的。该方法中, 我们首先绘制一个由光源发出的“光锥”(cone), 然后绘制与场景物体相交的阴影体积 (shadow volume), 最后再重新渲染场景物体, 只有在场景物体在阴影体积内的片段才被绘制, 其他片段则不进行渲染, 这样就能得到阴影效果。

Depth Fail Method (深度失败法) 则是通过渲染表面阴影来实现的。该方法中, 我们首先绘制一个由光源发出的“光锥”, 然后绘制与场景物体相交的阴影体积, 并在阴影体积内使用 Stencil buffer 标记。最后, 我们通过渲染场景物体来填充阴影部分, 但是只有在深度测试失败的像素点 (即在阴影体积内) 才被填充, 从而得到阴影效果。

Shadow Volume Method 的实现步骤包括:

1. 创建阴影体积: 根据光源位置、场景中的物体以及场景边界, 创建与物体相交的阴影体积。
2. 填充阴影: 通过渲染深度缓存或 Stencil buffer 来填充阴影部分。
3. 绘制阴影: 根据阴影的填充部分, 在场景中绘制相应的阴影。

Shadow Volume Count (影子体积数量) 指的是需要绘制的阴影体积数量, 即每一个场景物体需要对应一个阴影体积。在复杂场景中, 阴影体积数量可能非常大, 这会影响渲染效率。为了提高效率, 可以使用一些优化技术, 比如 Shadow Culling (影子剔除) 和 Level of Detail (细节级别)。

### 2. z-pass

用一句简单的话来概括, z-pass 的算法就是从视点向物体引一条视线, 当这条射线进入 shadowvolume 的时候, stencil 值加一, 而当这条射线离开 shadowvolume 的时候, stencil 值减一。如果 stencil 值为零, 则表示实现进入和离开 shadowvolume 的次数相等, 自然就表示物体不在 shadowvolume 内了。

根据每个像素的 stencil 值判断其是否处于阴影当中 (如果 stencil 的值大于零, 则这个像素在 shadowvolume 内, 否则在 shadowvolume 的外面), 然后据此绘制阴影效果。

### 3. z-fail

视点在 shadowvolume 外面的情况, z-pass 算法工作的很好, 不过一旦视点进入到了 shadowvolume 里面, z-pass 算法就会立即失效。

(1)Pass1: enable z-write/z-test, 渲染整个场景, 得到 depth map (这一步和 z-pass 的完全一样)

(2)Pass2: disable z-write, enable z-test/stencil-write。渲染 shadow volume, 对于它的 back face, 如果 z-test 的结果是 fail, stencil 值加一, 如果 z-test 的结果是 pass, stencil 值不变。对于 front face, 如果 z-test 的结果是 fail, stencil 值减一, 如果结果是 pass, stencil 值不变。(名词解释见下)

在实现阴影的算法中, 阴影体 (shadow volume) 是指光线被遮挡的物体的空间几何体。在实现阴影的算法中, 通常是通过阴影体来确定哪些像素处于阴影中。阴影体由多个面组成, 每个面都有正面和背面, 正面是指面的法向量指向外部, 背面是指法向量指向内部。在渲染阴影体时, 我们需要考虑到光线从哪个面进入阴影体, 从哪个面离开阴影体。

在阴影体的背面和正面的判断中, 通常是通过“面的法向量与光线方向的点积”来进行的。如果点积结果为正数, 则说明光线从面的正面进入阴影体, 此时这个面就是阴影体的背面; 如果点积结果为负数, 则说明光线从面的背面进入阴影体, 此时这个面就是阴影体的正面。

而 z-test 是深度测试的一种形式, 它是在渲染时用来判断一个像素是否可见的。具体地, z-test 会将每个像素的深度值与深度缓冲区中已有的深度值进行比较, 如果当前像素的深度值比深度缓冲区中的深度值更小, 则该像素是可见的, 否则该像素就被遮

挡或者被视为不可见。在实现阴影算法中, z-test 通常用来判断光线是否穿过了阴影体。如果穿过了, 就说明该像素处于阴影中, 否则该像素不处于阴影中。

## Q11-- Shadow Mapping

解答:

### 1. 是一种 2-pass 的算法

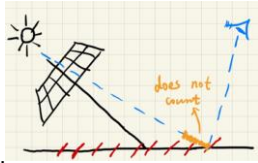
(1)从 light 处出发看向场景生成一张记录每个像素中最近物体深度的一张图

(2)从 camera(眼睛)出发,再渲染一遍场景.

这次要看渲染中的每个像素中都要来判断是否能被 light 照到,如果被照到则不在阴影中,反之,则存在于阴影中-- 做法是对渲染的每个像素, 对比 1-pass 得到的 shadow map 中这一点的最浅深度和 2-pass 中从点连到 light 处的深度

### 2. 自遮挡问题

(1)原因: 从 Light 处看向场景时, 认为在各个像素覆盖区域内都是一个常数的深度, 场景被离散化. 在 2-pass,也就是从 camera 处看向场景时,后面的红色小片在连接 light 时,会被误认为被前面的红色小片遮挡住,从而产生了错误的阴影,这个现象在 light 与平面趋于平行时

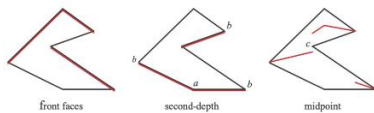


候最严重.

(2)解决方案: 在黄色区域内的遮挡关系给舍弃掉,技巧是,当 Light 处垂直于场景时,我们可以让这个遮挡区域尽可能小一点,当 light 趋近于平行场景时,我们让这个区域尽可能大一点,我们可以引入一个 bias 的概念(黄色区域),来降低自遮挡情况,bias 是根据角度调整的,并非常数. 具体方式就是当一个点深度大于记录深度的值超过一个阈值时, 我们才认为这个点在阴影内。

(3)bias 也引出了另一个问题: 当 Bias 调的过大时,我们会丢失原本应该存在的阴影.

解决方案-- Second-depth shadow mapping: 舍弃 biasd 的概念,在渲染时不仅存最小的深度,还要存第二小的深度,然后用最小深度和第二小深度中间的深度来作比较.



(实际中并没有人去使用这个技术)

### 3. 走样问题

(1)原因: shadow map 本身就存在分辨率, 当分辨率不够大自然会看到锯齿,因为 shadow map 上每一个像素都可以看为小片,那么投出来的阴影自然会存在锯齿.

(2)解决方案

#### ①Shadow casters

Shadow casters 是一种基于 GPU 渲染的技术,它使用多个深度纹理来捕捉场景中每个物体的深度信息.每个深度纹理只包含与当前物体相关的深度信息,这使得 Shadow Mapping 过程更加准确。

在 Shadow casters 中,每个物体都会被渲染到其自己的深度纹理中,而不是与整个场景一起渲染到一个深度纹理中。

在渲染每个物体时,Shadow casters 技术还使用偏移技术来消除多边形偏移问题.这种技术可以在物体表面的深度值和阴影表面之间添加一个小的偏移量,以解决阴影映射中的走样问题。

#### ②Resolution Enhancement

Resolution Enhancement 技术通过增加深度纹理的分辨率来解决这个问题.具体来说,它使用一个额外的深度纹理,将场景从光源的角度再次渲染,但这次渲染的分辨率比原始渲染高。

在使用 Resolution Enhancement 技术时，阴影映射的过程分为两个阶段。在第一阶段中，场景从光源的角度渲染到一个低分辨率的深度纹理中。然后，这个深度纹理被用来计算阴影，这会产生阴影失真和噪点。在第二阶段中，使用一个额外的深度纹理，将场景从光源的角度再次渲染，但这次渲染的分辨率比第一次高。然后，这个高分辨率的深度纹理被用来计算阴影，以提高阴影的质量。

### ③ Cascaded shadow maps (CSW)

CSM 将场景分为多个级别，每个级别有不同的分辨率和覆盖范围。在实现过程中，可以使用一些算法或启发式方法来计算这些级别的参数。例如，可以根据观察视点的距离和方向计算每个级别的大小和位置，以确保在不同的距离和方向下都能获得高质量的阴影。

④ 计算每个点的 shading，然后去乘这个点的 visibility 得到的就是最后的渲染结果

④-1) 利用约等式  $\int_{\Omega} f(x)g(x) dx \approx \frac{\int_{\Omega} f(x) dx}{\int_{\Omega} dx} \cdot \int_{\Omega} g(x) dx$  来简化 rendering equation

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

把 visibility 看作是 f(x), 提取出来并作归一化处理

处理

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

有

④-2)

- 满足什么条件下可以分离出来？
  - 点光源
  - 面积光
  - diffuse



## Q12—PCF&PCSS

### PCF(percentage closer filtering) Shadow Mapping & PCSS

解答:

#### 1. PCF 思路

之前判断点是否在阴影中时,把 shading point 连向 light 然后跟 Shadow map 对应的这一点深度比较,判断点是否在阴影内,之前是做一次比较

PCF 的区别是,对于这个 shading point, 仍要判断其是否在阴影内,但是把它投影到 light 之后不再只找其对应的单个像素,而是找其周围一圈的像素,把周围像素深度比较的结果加起来平均一下, 就得到一个 0-1 之间的数, 就得到了一个模糊的结果。

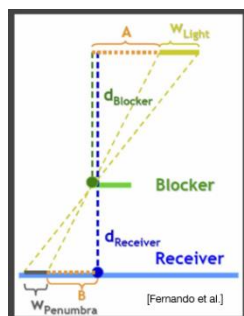
#### 2. 由 PCF 到 PCSS

已知 filter size 越大, 阴影本身越软

又已知阴影接受物与阴影投射物的距离越小,阴影越锐利.(eg. 钢笔笔尖离纸越近的地方产生的阴影越锐利)

因此,要生成软阴影的话,需要决定一个软阴影的半影区,也就是 filter size 的大小.

(1)分析



可以发现,若 blocker 越靠近 receiver,  $w_{Penumbra}$  就会越小.

有此数学关系

$$w_{Penumbra} = (d_{Receiver} - d_{Blocker}) \cdot w_{Light} / d_{Blocker}$$

那么可以用 filter size  $\propto$  blocker distance

(2)确定 blocker 距离光源的位置

用平均遮挡距离: 不能直接使用 shadow map 中对应单个点的深度来代表 blocker 距离,一方面因为如果该点的深度与周围点的深度差距较大(遮挡物的表面陡峭或者对应点正好有一个孔洞),将会产生一个错误的效果,另一方面是因为 blocker 上的每个点距离光源的距离是不同的,深度也是不一样的. 故选择使用平均遮挡距离来代替,所以平常我们指的 blocker depth 其实是 Average blocker depth.

(3)步骤

- Step 1: Blocker search  
(getting the average blocker depth in a certain region)
- Step 2: Penumbra estimation  
(use the average blocker depth to determine filter size)
- Step 3: Percentage Closer Filtering

### Q13-- Variance soft shadow mapping(VSSM)

#### 1. 解决 PCSS 的第三步 Percentage Closer Filtering 的速度慢

(1)第三步 PCF 之中,要在 shadow map 上对其周围的一圈像素的各个最小深度与 Shading point 比较,从而判断是否遮挡,也就是要求出范围内有百分之多少的像素比它浅.

正态分布就只需要方差和平均值就能得出,更加的方便快捷,这也就是 VSSM 的核心思想,通过正态分布来知道大约占百分之几

①快速求均值: mipmap 或 Summed Area Tables (SAT) [见 Q10]

②结合期望与方差之间的关系公式  $\text{Var}(X) = E(X^2) - E^2(X)$  来得到方差

②-1) $E^2(x)$ : shadow map 中存储的 depth 就是公式中的 x

②-2) $E(x^2)$ : 额外生成一张 shadow map 存  $\text{depth}^2$ , 叫做 square-depth map

(2)得到期望和方差之后,根据切比雪夫不等式近似得到一个 depth 大于 shading point 点深度的面积.,也就是求出了未遮挡 Shading point 的概率,从而可以求出一个在 1-0 之间的 visibility.

①PDF: 概率密度函数 (probability density function), CDF: 累积分布函数 (cumulative distribution function)

对于一个通用的高斯的 PDF, 对于这类 PDF, 可以直接把 CDF 结果, 输出为一个表, 叫误差函数 Error Fuction, 误差函数有数值解

②切比雪夫不等式

$$P(x > t) \leq \frac{\sigma^2}{\sigma^2 + (t - \mu)^2} \quad \begin{array}{l} \mu : \text{mean} \\ \sigma^2 : \text{variance} \end{array}$$

将这个不等式近似为相等, 不用计算 CDF 了, 而是通过 1 - 求出的  $x > t$  的面积值得到 CDF 切比雪夫不等式有一个苛刻的条件:  $t$  大于均值时才能用。

#### 2. 解决 PCSS 的第一步需要将范围内所有像素的深度走一遍从而求平均遮挡深度 Average Blocker Depth 的速度慢

①蓝色是深度小于 shading point 的遮挡区域,其平均深度为  $Z_{occ}$  红色是深度大于 shading point 的非遮挡区域.其平均深度为  $Z_{unocc}$ .并且我们认为区域内的像素总数为 N,非遮挡的像



4	4	8	8
4	4	8	8
4	4	8	8
4	4	8	8
6	6	6	8
6	6	6	8
6	6	6	8
6	6	6	8

素为  $N_1$  个,遮挡的像素为  $N_2$  个.

②有公式:  $\frac{N_1}{N} Z_{unocc} + \frac{N_2}{N} Z_{occ} = Z_{Avg}$ , 即, 非遮挡像素占的比例 \* 非遮挡物的平均深度 + 遮挡像素占的比例 \* 遮挡物的平均深度 = 总区域内的平均深度.

总区域内的平均深度用 mipmap 或者 SAT 去求,然后用 shadow map 和 square-depth map

求方差,最后根据切比雪夫不等式近似求出  $N_1 / N = P(x > t)$ ,  $N_2 / N = 1 - P(x > t)$

③大胆的假设:认为非遮挡物的平均深度 = shading point 的深度

#### 3. 缺点和解决方案

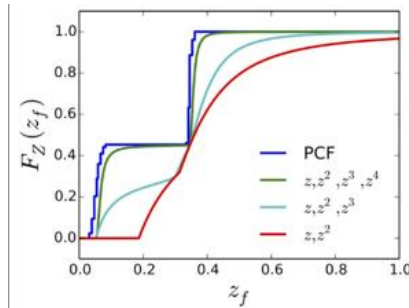
(1)VSSM 的缺点: 不是正态分布强行按正态分布算就会出现漏光和过暗的结果。在阴影的承接面不是平面的情况下也会出现阴影断掉的现象。

(2)解决方案: Moment Shadow Mapping (MSM)



①引入更高阶的 moments 来得到更加准确的深度分布情况.想要描述的更准确,就要使用更高阶的 moment(矩)

矩的定义有很多,最简单的矩就是记录一个数的次方,VSSM 就等于用了前两阶的矩  
如果保留前 M 阶的矩,就能描述一个阶跃函数,阶数等  $2/M$ ,就等于某种展开。越多的阶数就和原本的分布越拟合。一般来说 4 阶就够用。



## Q14-- Distance Field Soft Shadows (DFSS)

### 1.概述

Distance Field Soft Shadows (DFSS) 是一种实时渲染技术,用于在场景中生成柔和、逼真的阴影。下面是实现 DFSS 的分步介绍:

第一步:介绍 SDF (Signed Distance Function)

SDF 是一种表示几何形状的函数,它可以计算给定点与最近几何形状之间的距离。SDF 函数的输出值可以是正数、负数或零,分别表示点在形状外部、内部或在形状表面上。通过使用 SDF 函数,我们可以用一个简单的公式来描述复杂的几何形状,而不需要使用大量的三角形或其他复杂的几何图形。

第二步:介绍 SDF 用于 Ray Marching

在实时渲染中,Ray Marching 是一种常用的技术,用于从相机位置开始,沿着视线(光线)寻找场景中最近的表面。使用 SDF 函数,我们可以通过 Ray Marching 来寻找离光线最近的表面,而无需将场景表示为大量的三角形或其他几何形状。

Ray Marching 的基本思想是从相机位置开始,沿着视线(光线)向前移动,同时根据 SDF 函数计算出每个点与最近几何形状之间的距离。这个距离值可以用来确定下一步的前进距离,从而避免与场景中的几何形状相交。当我们找到距离相机最近的表面时,我们可以计算出与表面的交点,从而可以进一步对场景进行渲染。

第三步:介绍 SDF 用于生成软阴影

为了生成柔和、逼真的阴影,我们需要考虑光线和场景中的对象之间的相对位置关系。一个简单的方法是将“安全距离”概念应用到 SDF 函数中。安全距离是指在两个对象之间需要保持的最小距离,以避免对象之间的相互遮挡。使用 SDF 函数,我们可以计算出在任意一点的安全角度。如果我们知道光线的方向和安全角度,我们可以计算出在该点处的阴影强度。在 Ray Marching 过程中,我们可以使用这个安全角度来计算出柔和的、逼真的阴影。具体来说,我们可以使用逐步迭代的方式来计算出在光线路径上的每个点的安全角度,从而生成柔和、逼真的阴影。

2. 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。

3. 把 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

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

visibility, k 越小,半影区域越大,越接近软阴影效果。

## Q14-- Shading from Environment Lighting

### 1. 环境光照的基本概念和 split-sum

环境光照在实时渲染中扮演着重要的角色，它可以为场景中的物体提供全局的光照信息，增强了场景的真实感。下面将介绍实时渲染中环境光照的基本概念。

第一部分：环境光照的定义

环境光照是指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，也可以叫做 IBL (image-based lighting)。通常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。在实时渲染中，我们通常用 spherical map 和 cube map 来存储环境光照。

第二部分：使用蒙特卡洛积分求解

在 rendering equation 中，我们可以用蒙特卡洛积分去解环境光照。但是由于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。

第三部分：使用近似公式拆分 rendering equation 的 lighting 项

由于 brdf 项满足 accuracy condition，即 small support 或 smooth，因此我们可以将 rendering equation 的 lighting 项用近似公式拆出。这样可以大大降低采样的复杂度。

第四部分(split-sum1)：拆分 lighting 项并预先滤波

我们可以将 lighting 项拆分出来，然后将 brdf 范围内的 lighting 积分起来并进行 normalize，从而将 IBL 这张图给模糊了。

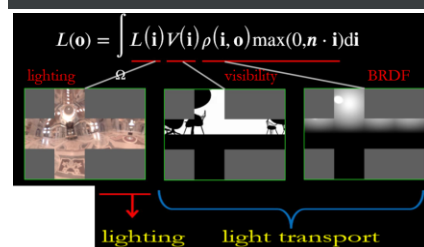
Prefiltering 就是提前把滤波环境光生成，提前将不同卷积核的滤波核的环境光生成一系列模糊过的图（和 mipmap 相似），当我们需要时进行查询即可，其他尺寸的图则可以经过这些已生成的通过三线性插值得到。

第五部分(split-sum2)：预计算 BRDF 项并降低维度

对于 microfacet BRDF，我们需要预先计算的是 Fresnel 项和法线分布函数（NDF）。Fresnel 项可以近似为一个基础反射率  $R_0$  和入射角的指数函数，这个函数可以通过 Schlick 近似来计算。而 NDF 是一个一维的分布函数，它描述了半角向量和法线之间的夹角分布，可以通过各种数学模型来计算。

在实际计算中，我们将 BRDF 拆分为两部分，一部分是已经预计算的环境光照，另一部分是 BRDF 的积分，即 Fresnel 项和 NDF 项。为了减少维度，我们可以通过将 Schlick 近似带入积分公式，将基础反射率  $R_0$  拆分出来，只需要预计算粗糙度和入射角度  $\theta$  两个变量即可。

最终，我们将预计算的结果绘制成一张纹理，然后在渲染时进行查询即可。这样可以大大加快实时渲染的速度，同时保持较高的渲染质量。



### 2. 球谐函数 (Spherical Harmonics, SH)

实时渲染中的环境光照中的 PRT 方法的 SH 部分主要包括以下几部分：

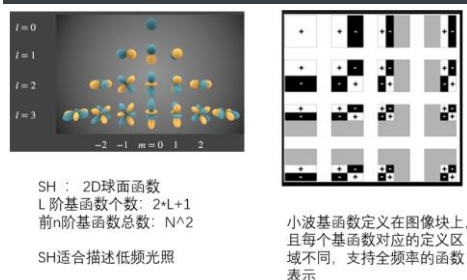
第一部分：阴影问题。在已知如何不采样去计算不考虑 shadow 时的 shading 值的情况下，如何在有了环境光照情况下得到物体被环境光照射下生成的阴影呢？目前的技术难以实现，主要是由于“many light”问题和 sampling 问题。解决这个问题需要考虑可见性项，这个问题尚未得到有效的解决。

第二部分：数学工具。PRT 方法引入了一种数学工具——球谐函数 (Spherical Harmonics, SH)，它是一系列的基函数，可以把一个函数表示为这些基函数的线性组合。由于它是定义在球面上的，因此可以理解为是对方向的函数。与一维的傅里叶一样，SH 也存在不同频率的函数，但不同频率的函数个数也不同，频率越高所含有的基函数越多。

第三部分：基函数。由于已知环境光和一个 diffuse 的物体，在不考虑 Shadow 的情况下，如何计算 shading 值？引入球谐函数，可以把环境光表示为一系列的球谐函数的线性组合。通常，描述环境漫反射部分用 3 阶 SH 就足够了。

第四部分：球谐函数的操作和性质。球谐函数定义了一系列的操作和性质，如投影操作和重建操作。通过投影操作，可以确定基函数前面的系数；而通过重建操作，可以用系数和基函数恢复原来的函数。球谐函数还具有正交性质和旋转性质。

第五部分：3 阶 SH 足够的原因。对于任意的 product integral（两个函数先乘积再积分），我们将其认为是做了一个卷积操作，可以理解为 spatial 域上的两个信号进行一个卷积，等于在频域上让两个信号相乘。如果两个信号中有一个信号是低频的，那么频域上相乘后得到的结果也是低频的，最终相乘在积分的结果也是低频的。低频意味着变换更加地 smooth 或者有着 slow 的变化。因此，无论光照项有多么复杂，其本应该用多高频的基函数去表示，但我们希望得到的是其与 BRDF 之积的积分，所以可以使用比较低频的基函数。通常，3 阶 SH 已经足够描述环境光照



### 3. Precomputed Radiance Transfer (PRT)方法

Precomputed Radiance Transfer (PRT)方法是实时渲染中用于计算环境光照的一种方法，其主要优点是能够高精度地计算出全局光照部分产生的阴影。PRT 方法的基本思想是将 Rendering Equation 分为两部分，即光照部分和光传输部分。在预计算阶段，PRT 方法先计算出光照部分的基函数表示，然后再计算光传输部分，最终将两部分的结果组合起来，得到最终的渲染结果。

PRT 方法的具体实现是利用基函数的原理，将 Rendering Equation 中的三个部分（光照、可见性和 BRDF）都表示成球面函数的形式，然后利用基函数对其中的一些部分进行预计算，从而减少渲染的计算量。在实时渲染中，我们通常把 Rendering Equation 分成两部分来计算，即光照部分和光传输部分，其中光照部分可以用基函数来表示，在预计算阶段计算出来；而光传输部分通常是不变的，可以认为是场景中每个物体自己的性质，也可以用基函数来表示，并在预计算阶段计算出来。这样，在渲染时只需要对每个像素进行一次点乘运算即可得到最终的渲染结果。

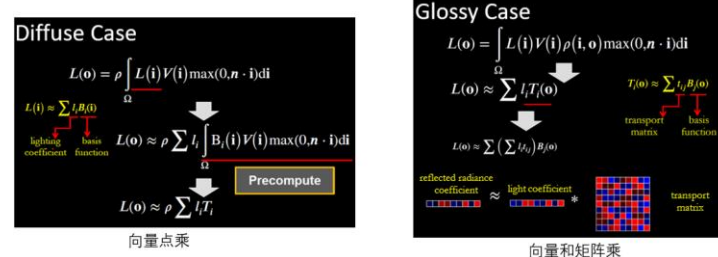
对于 diffuse 物体，由于光照部分可以用基函数来表示，因此我们可以将其代入 Rendering Equation 中，并对其中的某些部分进行简化，从而得到预计算的公式。具体来说，我们可以将 BRDF 的部分提取出来，并把 Bi 限制在基函数上，然后再将其投影到光传输部分的基函数上，这样就可以得到一个仅需一次点乘运算的预计算公式。

对于 glossy 物体，由于其 BRDF 是一个 4 维的函数（包含两个输入方向和两个输出方向），因此需要将其投影到一个 2D 的平面上，然后再将其投影到光传输部分的基函数上，最终得到一个矩阵表示。由于 glossy 物体的光照是和视点有关的，因此我们需要在视点方向上对其投影到基函数上，这样就可以得到最终的预计算结果。

PRT 方法的缺点是，由于光传输部分已经做了预计算，因此场景不能动，只能对静态物体进行计算。但是对于光源的旋转等变换，我们可以通过将其投影到基函数上来解决，由于基函数具有旋转不变性，因此可以得到旋转后的基函数。

另一种基函数——Haar 小波。与球谐函数相比，小波变换可以全频率表示，但只有少数系数是非零的。为了避免在球面上出现缝隙，通常使用 Cubemap 作为环境光源。小波变换将高频信息存储在图像的一些小区域中，而将低频信息存储在图像的大区域中。这种方法可以在保留低频信息的同时，压缩高频信息，从而减少存储空间。但是，小波变换有一个缺陷，即不支持旋转，因此需要重新计算基函数以适应旋转。

### PRT



## Q15-- Real-Time Global Illumination (in 3D)

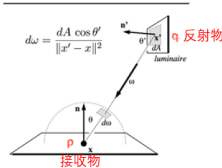
在实时渲染中,全局光照是比直接光照多一次 bounce 的间接光照.

Reflective Shadow Maps (RSM)

1. 直接光照作为次级光源, 进一步照亮场景中其他物体



2. 找距离  $p$  近的次级光源  $q$ , 照亮  $p$ , 采用 2D 空间采样方式获取  $q$ . 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.



次级光源 (反射物): diffuse.

被次级光源照亮的点可以是 glossy.

次级光源片元很小, 光源积分, 可以通过一次采样实现

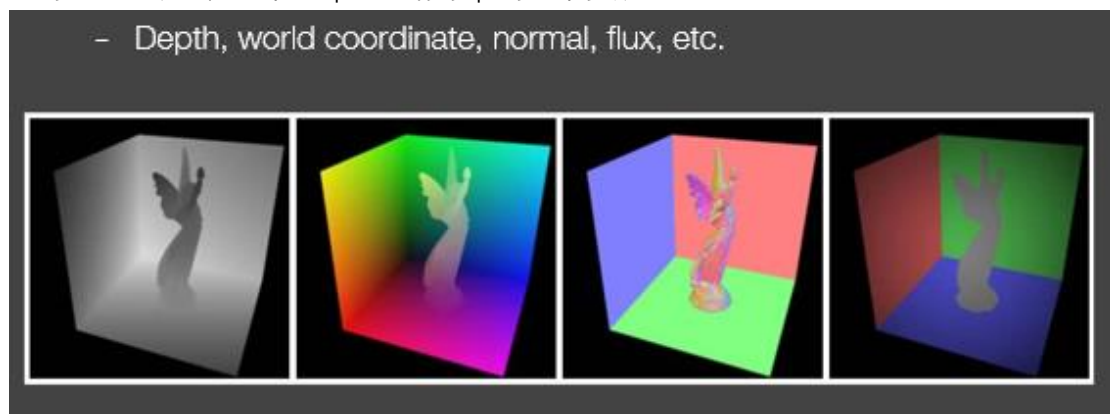
$$L_o(p, \omega_o) = \int_{\Omega_{\text{patch}}} L_i(p, \omega_i) V(p, \omega_i) f_r(p, \omega_i, \omega_o) \cos \theta_i d\omega_i$$

$$= \int_{A_{\text{patch}}} L_i(q \rightarrow p) V(p, \omega_i) f_r(p, q \rightarrow p, \omega_o) \frac{\cos \theta_p \cos \theta_q}{\|q - p\|^2} dA$$

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

$f_r = \rho / \pi$  For s  
 $L_i = f_r \frac{\Phi}{dA}$  ( $\Phi$  is the incident flux or energy)

3. 数据存储: RSM 在每一个像素中都需要存储深度值, 世界坐标, 法线, 反射光功率, 如下图的可视化效果, 四个 map 对应像素  $p$  的四个参数.



4.

- 好处:
  - 容易实现—shadow map 流程: 第一个 pass 生成 RSM 次级光源 (4 个 map, depth, normal, coordinate, flux ---for q); 第二个流程计算 RSM 次级光源对  $p$  的贡献,在投影 2D 空间上采样找到 shadow map 上距离着色点 (接收物)  $p$  比较近的次级光源  $q$ , 进而  $q$  照亮  $p$ ;

- 存储小：屏幕空间存储相对于光源深度  $depth$ （用于计算阴影）、对应点的世界坐标(方面后面做  $p-q$  距离计算)、反射物（次级光源）的法线（用于计算  $\cos$ ）、存  $flux$ （和光源相关，和法向没有关系）；
- 速度快；
- 问题：
  - 有多少个光源，就需要多少 RSM；
  - 假设反射物是  $diffuse$  的（次级光源不能是  $glossy$ ）；
  - 不计算反射物到接收物之间的可见性，导致不真实；
  - 计算距离接收物近的次级光源时，2Dshadow map 上距离假设能够反映三维空间实际距离，导致误差；
  - 距离接收物近的次级光源采样多效果好；采样少效果差；

### Light Propagation Volumes (LPV)

1. 原则：radiance 在空间传播的过程中在某条直线上是不变的量

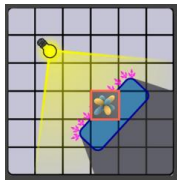
2. 步骤 1：生成次级光源

针对每个直接光源生成阴影图，每个像素对应一个片元。具体操作过程中，可以采样一些能量高的片元作为初始的虚拟次级光源。



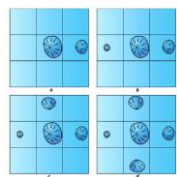
步骤 2：次级光源注入空间 grid

次级光源注入空间格子；用 SH 表示格子内不同方向上次级光源出射的 radiance 的分布，常用前两阶 SH（4 个基函数）。



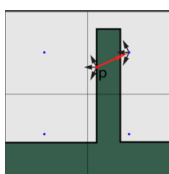
步骤 3：radiance 传播

每个格子内部 radiance 向周围 6 个格子传播，传播的时候不考虑格子之间的 visibility；待传播稳定后，每个格子内部 radiance 继续用 SH 表示。



步骤 4：渲染

对于每个一着色点，找到其所在的格子，使用格子内所有方向 radiance (SH) 照亮这个着色点（不考虑格子内部的遮挡）



3.

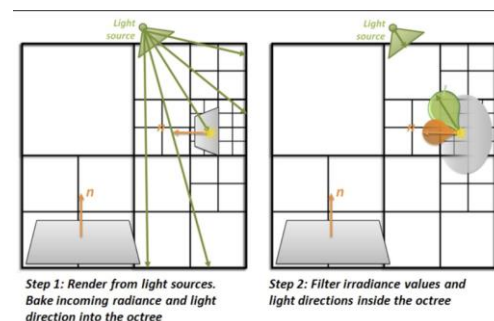
- 优点：
  - 对任何一个 shading point  $p$ ，可以快速检索得到任意个方向照射到  $p$  的 radiance 信息，也就是间接光照的信息，全局光照效果质量好，速度快。
- 问题：
  - 次级光源假设 diffuse，对于 glossy 的接收物适用，每个格子存储的相当于 light transport，不能表示高频材质的次级光源。
  - 没有考虑格子内存储的 light transport (SH) 对于着色点的可见性，导致 Light leaking (漏光)。

### Voxel Global Illumination (VXGI)

1. 预处理阶段：from light

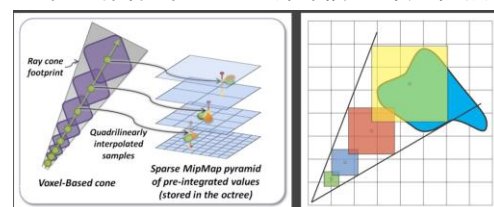
(1) 步骤一：从光源绘制场景，得到次级光源，并且把次级光源的入射 radiance，以及光源方向，存储到八叉树

(2) 步骤二：八叉树每个格子中，filter 所有次级光源 irradiance 和光源方向



2. 渲染阶段：from camera

步骤三：对于每个着色点，如果是 glossy 的表面，沿着反射方向 trace 一个 cone，查找八叉树中和 cone 相交的体素。随着 cone 半径的增加，查找八叉树上一层的体素。进而用相交体素中存储的次级光线的信息计算在该 cone 方向的出射信息，进而照亮着色点。



3.

- 优点：
  - 支持反射物是 glossy 的全局光照计算，全局光照质量高。
- 缺点：
  - 需要八叉树体素存储反射物入射光分布，存储代价高；
  - 每个着色点需要做 cone tracing 的操作，计算代价相对于 LPV 更高。



## Q16-- Real-Time Global Illumination (screen space)

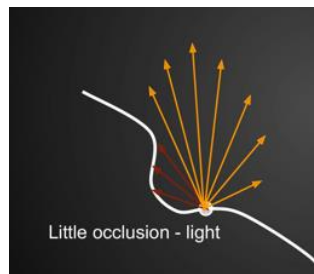
### Screen Space Ambient Occlusion (SSAO)

1. 假设间接光照是常数；任意位置任意方向间接光照一样

但是考虑了不同方向的 visibility 问题

假设间接光照是从很远处到达着色点，在着色点所在半径范围内，考虑哪些被遮挡

2. 在环境光一样的情况下考虑遮挡关系。



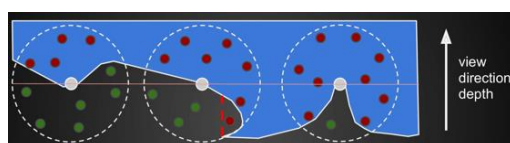
图中红色部分表示被遮挡,黄色部分表示未被遮挡,我们可以明显的得出一个结论,左边的 Shading point 要比右边的亮一点.

3. 把一个 product integral 中的一项拆除去并除以一个空积分进行归一化,由于这里要考虑的是各个方向的 visibility,因此把 visibility 项拆出去.

$$L_o^{\text{indir}}(\mathbf{p}, \omega_o) \approx \frac{\int_{\Omega^+} V(\mathbf{p}, \omega_i) \cos \theta_i d\omega_i}{\int_{\Omega^+} \cos \theta_i d\omega_i} \cdot \int_{\Omega^+} L_i^{\text{indir}}(\mathbf{p}, \omega_i) f_r(\mathbf{p}, \omega_i, \omega_o) \cos \theta_i d\omega_i$$
$$\square \triangleq k_A = \frac{\int_{\Omega^+} V(\mathbf{p}, \omega_i) \cos \theta_i d\omega_i}{\pi} \quad \square = L_i^{\text{indir}}(\mathbf{p}) \cdot \frac{\rho}{\pi} \cdot \pi = L_i^{\text{indir}}(\mathbf{p}) \cdot \rho$$

4. 假设 1: 以 shading point 为圆心的三维空间球内随机撒一些点, 判断这些点能否被 shading point 看到;

假设 2: 深度图---可以作为场景几何的一个简单近似, 距离 camera 最近的深度----可以作为物体内外的判断;



5. Pass1 : camera 第一轮渲染得到 depth buffer;

Pass2: 对于每个着色点, 在以其为圆心的球内部的采样, 并将采样点投影到 camera, 查询 pass1 中 depth buffer 存储的深度, 如果当前采样点深度更大, 那么在物体内部, 被挡住了, 计算遮挡采样点相对于所有采样点的比率, 用于该着色点环境光照的遮挡系数。

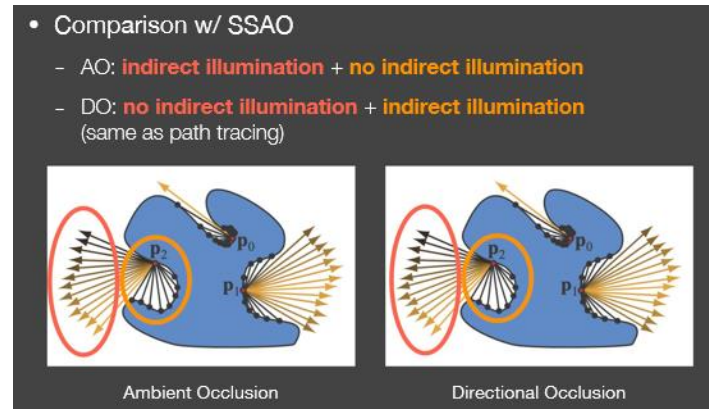
问题: 屏幕空间接触物体, 会产生错误的接触阴影。

## Screen Space Directional Occlusion (SSDO)

1.SSAO 和 SSDO 的区别——假设相反：

--SSAO：左图红色圈接受间接光照；橙色框不接受间接光照；

--SSDO：右图红色圈里面光线没有打到场景里面物体，不会产生间接光照；橙色框打到了其他物体，才会产生间接光照。



SSAO—考虑有限的范围内是否存在遮挡；所以假设环境光从比较远的地方过来；

SSDO—考虑某一个范围内是否打到物体，如果打到物体则产生间接光，所以考虑的是小的范围内间接光照的影响，假设间接光照来自比较近的地方。

2. Pass1 : camera 第一轮渲染得到 depth buffer, color buffer, normal, position 等；

Pass2: 对于每个着色点  $p$ ，在以其为圆心的法向所在的半球内部采样，并将采样点投影到 camera，依据 Pass1 中 depth 信息判断采样点是否被遮挡；

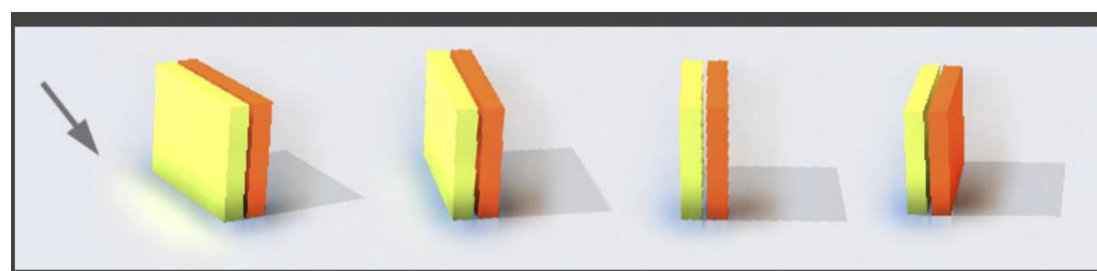
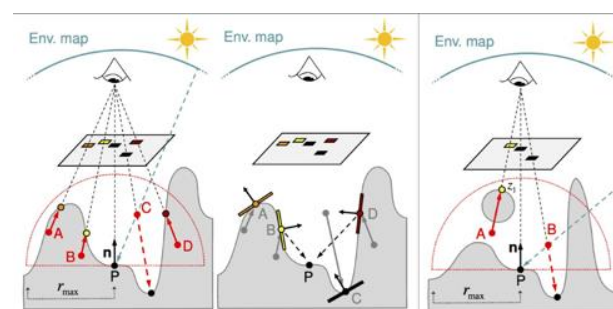
如果采样点没有被遮挡，不需要计算间接光照；

如果采样点被遮挡，那么依据遮挡点 color buffer 中存储的信息，为  $P$  提供间接光照；对所有被遮挡采样点对  $p$  的间接光照累加。

3. 优点：可以实现色溢等间接光照效果；

缺点：pass1 中只记录屏幕空间的信息，只记录了距离视点的最近的点，导致缺少部分间接光照。

SSDO 只能解决小范围间接光照。

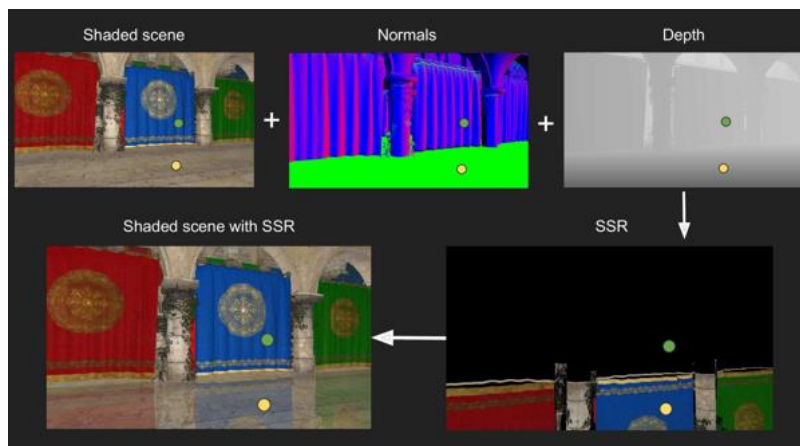




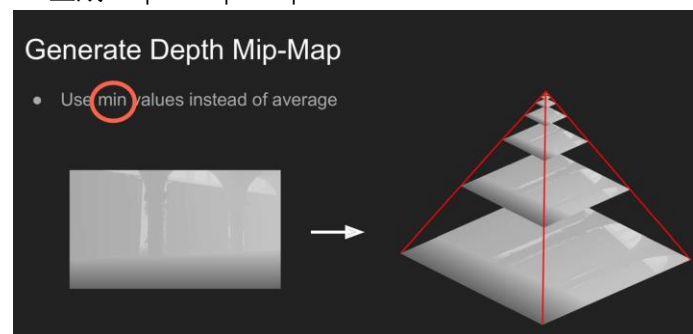
## Screen Space Reflection (SSR)

1.

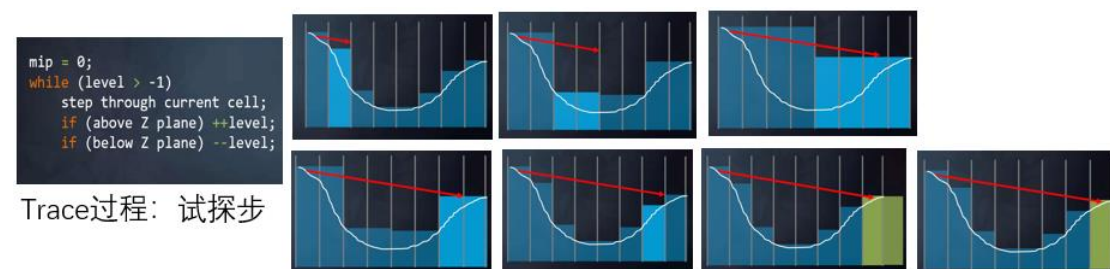
- 两次绘制
- Pass1: render from camera, 生成 color, normal, depth buffer
- Pass2: 对每个 glossy/镜面的着色点:
  - 1) 沿着镜面反射方向 trace 光线 (如果 BRDF 是 glossy, 那么反射方向附近采样几个方向进行 trace), 和 camera 场景壳求交
  - 2) 找到交点后, 求交点对 shading point 的贡献



2. 生成 Depth Mip-map



3.



4.

- 优点
  - 存储小、速度快
  - 可以实现镜面和光泽表面反射效果; 反射物可以是具有复杂几何形状的表面;
- 缺点
  - 只能反射平面空间的信息, 产生“截断”、反射信息缺失等。

## Q17-- RTRT

### 1. SPP (Sample per pixel)。

(1)一个光路样本，就是至少四条光线：从相机出发的光线\*1（这部分用光栅化会更快），打在着色点直接反射到光源的光线\*1，打在着色点间接反射到光源的光线（需要再反射一次，所以至少 2 条）

(2) SPP 的噪声极大.因此 RTRT 最核心的内容是降噪

### 2. 降噪方案: 时间上的 (Temporal) 滤波

#### (1) 物体在两帧之间的对应关系

①考虑滤波当前帧，并且认为前一帧是已经滤波好的，假设场景运动连续，复用前一帧的信息，同时使用 **motion vector** 来寻找物体在两帧之间的对应关系，即当前帧的某个像素点上一帧在哪个像素点，那么就可以把上一帧对应像素点的颜色拿过来用。

②在时间上滤波之前，可以先在空间上做一个滤波：保留低频信息，去掉高频噪声。（问题：怎么解决低频噪声？高频没有信息嘛？）

空间滤波的过程是，输入为原图和滤波核（filter kernel）K，输出滤波后的图。

②-1)高斯滤波器（Gaussian filter）。对目标像素（i,j）周围的像素(k,l)做高斯卷积，卷积核的权重可以定义如下：卷积时，把周围每个像素的色值乘上权重再相加，最后把结果除以总权重就是(i,j)滤波后的色值。

$$w(i, j, k, l) = \exp\left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2}\right)$$

②-2) 双边滤波（Bilateral filtering）。高斯滤波固然保留低频信息，但是会把边缘给模糊掉。人们希望保留仍然明显的边界。

那么边界是什么？剧烈变化的色值。

改进：考虑卷积目标像素(i,j)周围的某个像素(k,l)，如果两者的色值相差过大，则像素(k,l)对卷积的贡献变小。

实际上，是对卷积核做更多的控制：

$$w(i, j, k, l) = \exp\left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2} - \frac{\|I(i,j) - I(k,l)\|^2}{2\sigma_r^2}\right)$$

第二项会基于色值的差距来调整权重，色值差距越大，权重越小。

#### (2) Temporal filtering

##### ①

①-1)几何缓冲区（G-Buffer）。在渲染过程中可以低代价地得到一些屏幕空间的信息：像素深度，法线，世界坐标

上面提到了 motion vector，它描述当前帧一个像素点对应到上一帧在哪个像素点，用上一帧的结果来指导当前帧，达到降噪的效果。

①-2)Back Projection。对于在当前帧 i 的像素 x，它对应的着色点对应着上一帧 i-1 的哪个像素？

首先，要知道这个像素的世界坐标。如果 G-buffer 里面存了世界坐标，直接拿过来用就行了。如果没有存，就需要经过下面一系列变换来反推了：对于像素坐标 x，这是一个四维齐次坐标向量，包括屏幕上的横纵坐标以及深度，以及齐次坐标提供的一个维度，其实也就是经过 MVP 变换和透视投影 E 之后得到的坐标。那么如果要从 x 反求出该点本来的三维世界坐标，就需要对上述的 MVP 和 E 作逆变换：

1 SPP path tracing =

- 1 rasterization (primary) +
- 1 ray (primary visibility) +
- 1 ray (secondary bounce) +
- 1 ray (secondary vis.)

①-3) 上一帧像素  $x'$  的值就可以贡献到本帧的像素  $x$  的值上。用  $\sim$  表示没有经过滤波的色值,  $-$  表示经过空间上 (Spatial) 滤波的色值, 那么有:

$$\bar{C}^{(i)} = \text{Filter}[\hat{C}^{(i)}]$$

但是这样的效果并不尽人意。因此考虑时间上的降噪:

$$\bar{C}^{(i)} = \alpha \bar{C}^{(i)} + (1 - \alpha) C^{(i-1)}, \alpha = 0.1 - 0.2$$

主要依赖于上一帧的

信息。但是如果场景变化剧烈, 就会出现问题。几乎可以理解为某种角度的 TAA (Temporal Anti-Aliasing, TAA)。

②问题: 鬼影、拖尾

clamping—拖尾因为用了上一帧的信息。让上一帧  $C(i-1)$  拉近到  $C(i)$  的颜色, 再做插值。上一帧 clamp 到当前帧均值方差范围内。

检测方法: 用物体 ID 作为物体 index, 如果当前帧物体通过 MV 映射到上一帧不同物体上, 那么调整  $a$  (依据差别的大小, 提升  $a$  的数值), 会使得结果更加 noisy, 或者更模糊。

### 3. 联合双边滤波 (Cross/Joint bilateral filtering)

#### (1) 大滤波核 (Large Filters)

图中有不少可以免费得到的信息: G-Buffer (法线, 深度, 坐标, 物体 id 等等)。G-Buffer 是无噪声的!

这个过程中, 使用的指标不必归一化 (滤波会进行归一化), 也不是一定要用高斯函数。

考虑使用深度、法线、色值。

联合双边滤波实际上就是比双边滤波考虑更多的指标。

大滤波核 (Large Filters), 例如  $64 \times 64$ 。计算代价很高。解决方法包括:

①拆分实现 (Separate Passes)。用水平和竖直顺次的两次卷积来代替  $N \times N$  卷积核:

$$N^2 \rightarrow N + N$$

为什么能把 2D 的 axis-aligned 高斯卷积核拆成两个 1D 的高斯核?

$$G_{2D}(x, y) = G_{1D}(x) \cdot G_{1D}(y)$$

$$\iint F(x_0, y_0) G_{2D}(x_0 - x, y_0 - y) dx dy$$

$$= \int (\int F(x_0, y_0) G_{1D}(x_0 - x) dx) G_{1D}(y_0 - y) dy$$

但是当  $G$  不是高斯函数的时候, 这种拆分就是不准确的。

不过强行拆分的效果也问题不大。

②逐步增大的卷积核大小 (Progressively Growing Sizes)。

使用 a-trous wavelet。多趟算法。

实际上是空洞卷积 (dilated convolution)。

例如每一趟都做  $5 \times 5$  的卷积, 那么从小的卷积核开始, 逐趟把 dilution 调大, 应用更大的空洞卷积。

为什么要逐趟增加卷积核大小? 为什么可以用空洞卷积?

从本质上思考。卷积核越大, 意味着保留更低频的特征, 也就是说逐渐采用更加低通的滤波核。所以先采用保留相对更高频的低频信息的小滤波核, 可以防止之后进行更低通滤波时的

频谱混叠。为什么会出现频谱混叠？因为空洞卷积中存在一个卷积核的采样过程。

#### 4. Spatiotemporal Variance-Guided Filtering (SVGF)

优先去除 noise，即使以 overblur 作为代价。

问题：光源移动的时候，阴影会有残影。

深度指标采用非高斯的权值函数：

$$w_z = \exp\left(-\frac{|z(p)-z(q)|}{\sigma_z|\nabla z(p)\cdot(p-q)|+\epsilon}\right)$$

非常值的分母是为了解决拍摄时在梯度变化剧烈面上两点的贡献（虽然深度差异大，但应该权值大一些）

法线指标的权值函数：

$$w_n = \max(0, n(p) \cdot n(q))^{\sigma_n}$$

如果使用了法线贴图，不要用法线贴图应用之后的法线。

颜色指标（luminance）的权值函数：

$$w_l = \exp\left(-\frac{|l_i(p)-l_i(q)|}{\sigma_l\sqrt{g_{3\times3}(\text{Var}(l_i(p)))}+\epsilon}\right)$$

其中，Var 是通过 spatial-temporal 算出来的色值方差，而在此之后再进行一次 spatial 的 filter。

### 参考资料

- [1] Immortal. (2021). GAMES101 笔记 . Notion. <https://www.notion.so/GAMES101-b0e27c856cde429b8672671a54c34817>
- [2] Carlh3. (2021, October 26). Games202 高质量实时渲染课堂笔记 . Zhihu. <https://zhuanlan.zhihu.com/p/363333150>
- [3] 王浩. (2014, June 24). 阴影锥(Shadow Volume)原理与展望---真实的游戏效果的实现. Zhihu. <https://www.cnblogs.com/dragon2012/p/3806026.html>
- [4] 雾归流 . (2023, January 25). 二十四、GAMES202\_实时光线追踪 . Zhihu. <https://zhuanlan.zhihu.com/p/592768658>

注: 黑底白字代表由 chatgpt 生成