

Real-Time Ray Tracing

Mengzhu Wang

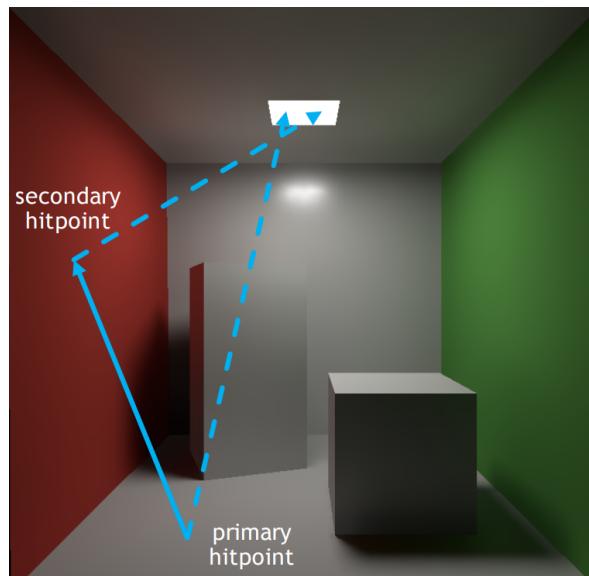
2024 年 5 月 19 日

1 Basic idea

RTX 是一种能够实现实时光线追踪的硬件，它每秒能追踪 10 Giga 的光线，还要除以分辨率和帧数，并且在 1s 内还要进行降噪等后处理。1 sample per pixel 就是每个像素采样一个样本，从而得到光追结果。1 SPP 从 camera 出发射出一根光线，与场景的交点为 primary hitpoint。primary hitpoint 和光源连线判断是否被遮挡得到 shadow ray。另外从 primary hitpoint 打出的一根光线与物体交点为 secondary hitpoint，该点与光源连线判断是否遮挡得到 secondary shadow ray。综上所述，1 SPP path tracing 有四条光线，分别是

- 1 rasterization (primary)
- 1 ray (primary visibility)
- 1 ray (secondary bounce)
- 1 ray (secondary visibility)

第一步是光栅化而非 trace primary ray，因为光栅化相当于在每个 pixel 都穿过一根 ray，而且光栅化速度更快，所以第一步就完成了所有的 primary ray，每个 shading point 剩下 trace 三条光线。



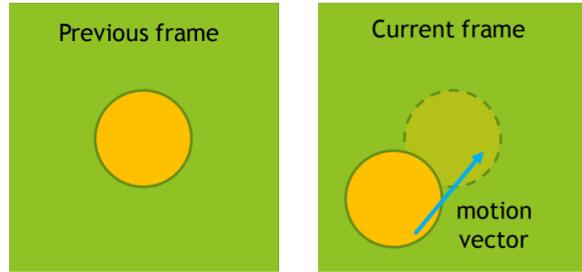
蒙特卡洛路径追踪在 1 SPP 情况下的结果噪声很大，所以对于 RTRT 最关键的技术就是降噪，要求质量好且速度快。

2 Temporal Accumulation / Filtering

工业界采用的是基于 temporal 的降噪，主要思想是假设前一帧被降噪且能复用，通过 motion vector 找到当前帧像素对应到上一帧所在位置，相当于增加了 SPP。

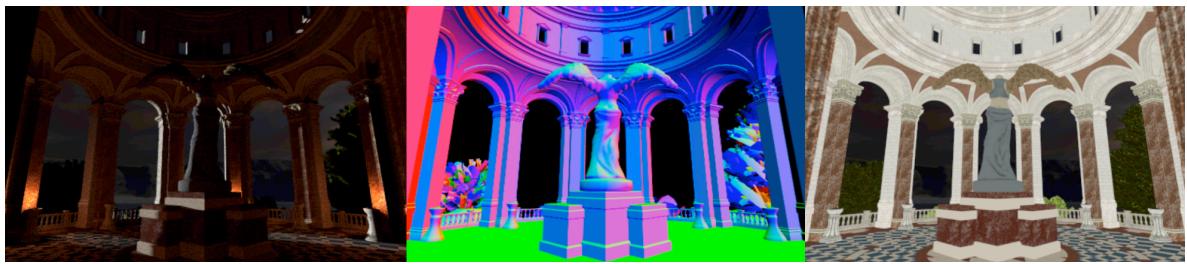
2.1 Motion vector

Motion vector 描述了屏幕空间中的物体在帧与帧之间运动的相对位置，即上一帧像素对应的片元，与下一帧该片元对应的像素位置的相对移动。



2.2 G-Buffer

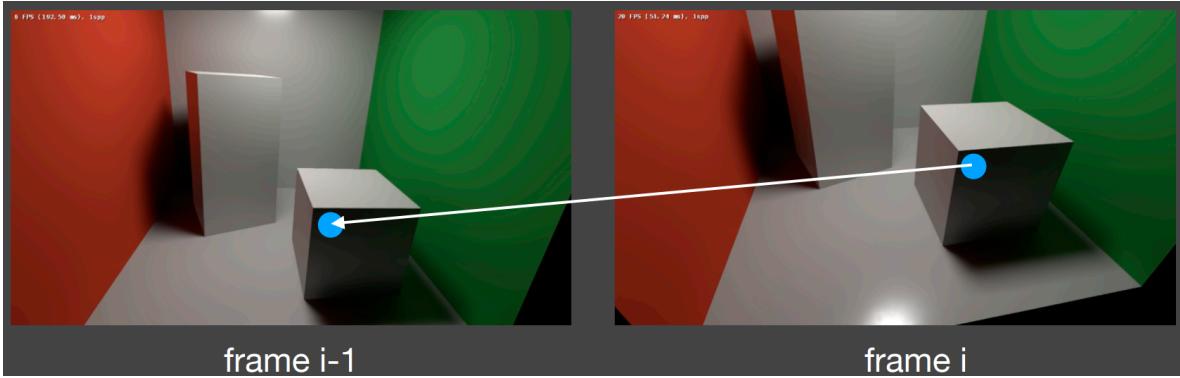
G-Buffer 全称 Geometry buffer，它在渲染过程中获得屏幕空间上的信息，如深度图、法线、世界坐标等。



2.3 Back Projection

Back projection 用于找到第 i 帧的像素在第 $i-1$ 帧对应的像素。首先计算当前像素 x 的世界坐标 s 。如果 G-buffer 中有世界坐标，只需要直接采样。如果没有，通过 VP 变换和视口变换的逆变换得到，即 $s = M^{-1}V^{-1}P^{-1}E^{-1}x$ 。当前帧世界坐标乘以帧移动的逆矩阵就得到了上一帧中这个点

的世界坐标，即 $s' = T^{-1}s$ 。最后再将上一帧的世界坐标转换为屏幕坐标，得到 $x' = E'P'V'M's'$ 。



2.4 Temporal Accum./Denoising

首先对当前帧做空间域滤波，即

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

通过 back projection 可以得到前一帧的采样值 $C^{(i-1)}$ ，将它和当前帧的采样值进行混合，得到时间域滤波的结果

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

α 一般取 0.1-0.2，也就是说上一帧的贡献率通常为 80%-90%。

2.5 Temporal Failure

时间域的信息并不总是能够复用的，比如以下情况：

- 切换场景，即相邻两帧渲染完全不同的内容
- 在狭长通道如走廊内，镜头不断后移，画面出现通道两旁的新内容（屏幕空间问题）
- 背景中原来被遮挡的地方突然出现（disocclusion）

如果这些情况下仍然采用时间域信息复用，参考上一帧，会出现拖影（lagging）现象。要解决以上问题，有 clamping 和 detection 两种思路：

- clamping：将前一帧的结果拉近当前帧，再做混合，可以削弱两帧差异过大的情况下，前一帧对当前帧的影响。
- detection：通过比较当前帧当前像素的 ID 与上一帧像素的 ID 是否一致来判断是否使用前一帧。如果不一致，那么就可以对 α 进行调，增大空间域滤波的比重，但这种方法会在相应的区域重新引入噪声。

时间复用在 shading 时也存在问题。

- 若物体和相机不动，只移动光源，那么 motion vector 始终为 0，导致一直复用上一帧的信息，从而导致 detached/lagging shadows。

- glossy 反射时，平面不动，motion vector 为 0，平面始终复用前一帧。那么移动影子时平面上的反射效果也会有拖尾。

3 Spatial Filtering

噪声在图像中通常表现为高频信号，降噪实际上是设计低通滤波器来去除高频噪声。滤波的计算实际上就是对当前像素的周围像素做加权平均，然后除以权重和用于归一化。高斯滤波是空间滤波常用的一种方法，它能够去除噪声，但也会损失有用的高频信息，导致边界模糊。

3.1 Bilateral Filtering

基于物体边界处颜色变化剧烈的观察，引入双边滤波。如果像素 i 和像素 j 的颜色差异很大，认为是在边界，考虑减少像素 j 给 i 的贡献，其滤波核计算如下：

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

该方法的一个问题是颜色变化剧烈的像素实际上无法区分是边界还是噪声。

3.2 Joint Bilateral Filtering

高斯滤波提出了一个滤波标准，即两像素之间的距离；双边滤波提出了两个滤波标准，即像素位置距离和颜色距离。联合双边滤波就是利用更多的特性来指导滤波，尤其适用于对路径追踪的结果降噪。

联合双边滤波采用 G-buffer 引导滤波。因为在渲染中可以通过光栅化免费获取很多 G-buffer 的信息，如深度法线、颜色等。光栅化过程与多次 bounce 无关，因此 G-buffer 也是没有噪声的。

联合双边滤波的滤波核不局限于高斯，任何随着距离增大减小的函数都可以，如指数绝对值、余弦函数等。

3.3 Implementing Large Filter

每个像素滤波需要遍历 $N \times N$ 的邻域，这对于大滤波核来说开销巨大。有两种主要方法可以解决大滤波核的实现问题。

3.3.1 Sol. 1: Separate Passes

二维高斯滤波可以拆分为一维水平滤波和一维垂直滤波，滤波采样的数量就从 N^2 降为 $2N$ 次。从数学上分析，二维高斯函数的定义就是水平垂直高斯滤波相乘：

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

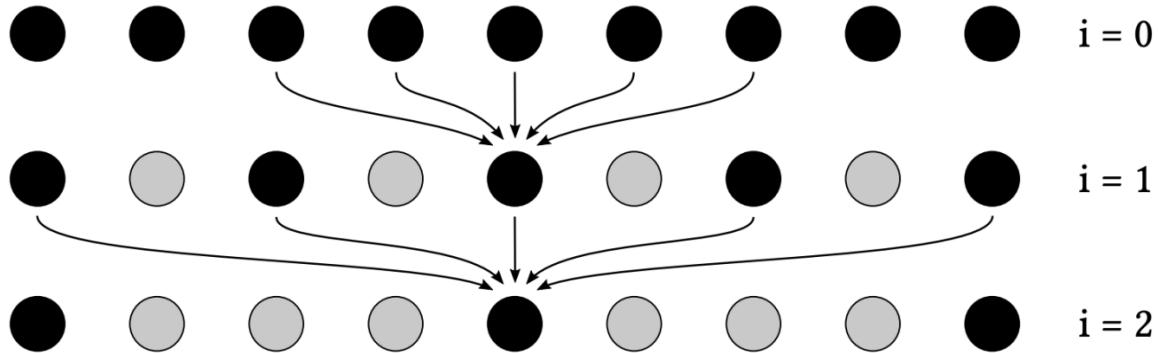
上式成立是因为

$$\int \int F(x_0, y_0) G_{2D}(x_0 - x, y_0 - y) dx dy = \int (\int F(x_0, y_0) G_{1D}(x_0 - x) dx) G_{1D}(y_0 - y) dy$$

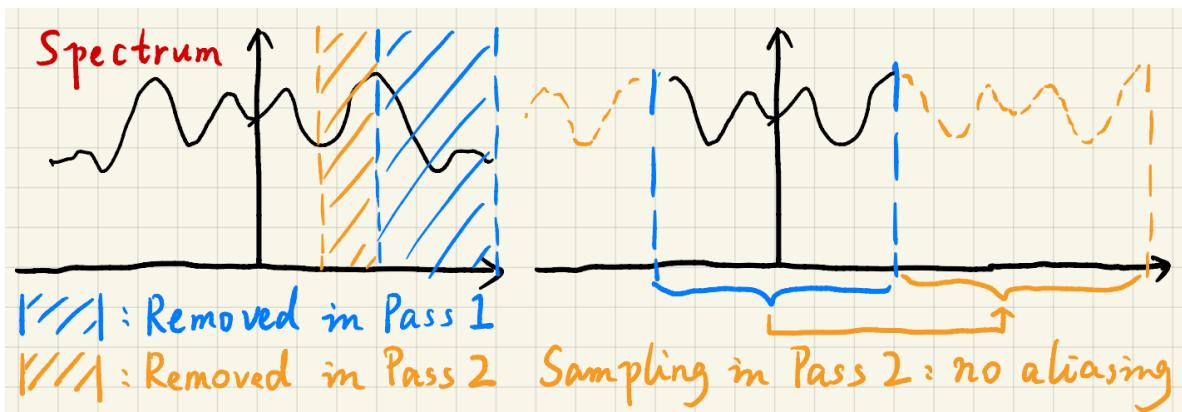
该做法一般只适用于高斯滤波，其他双边滤波器没有可拆分的良好性质。

3.3.2 Sol. 2: Progressively Growing Sizes

另一种方法是固定滤波核大小，用逐渐增长的滤波范围滤波多次。以 a-trous wavelet 为例，每趟都是 5×5 的滤波核，第 i 个 pass 滤波间隔为 2^i (i 从 0 开始)，采样跨度过为 $5 + 4(2^i - 1)$ 。要用 64×64 的滤波核，只需要进行 5 个 pass，查询次数从 64×64 减小到 $5 \times 5 \times 5$ 。



更大的滤波核相当于除去低频信息，为了更多地保留低频信息中相对高频的信息，选择逐渐增大滤波核。这里的原理是采样就是搬移频谱。第一个 pass 去掉左图高频的蓝色部分，第二个 pass 相当于在 9×9 的滤波核中采样 5×5 的滤波，该采样间隔对应右图的蓝色部分，也就是第一个 pass 去掉高频后的两倍，不会发生混叠。



3.4 Outlier Removal

渲染时会出现一些特别亮的点，滤波后其亮度贡献到周围的点使得周围一片区域变亮。

检测 outlier，可以取每个像素周围 7×7 的区域，计算均值和方差，在范围 $[\mu - k\sigma, \mu + k\sigma]$ 之外的值就是 outlier。然后将越界值 clamp 到范围内。

在时间域滤波中也可以用 clamp 缓解拖影问题。为了防止前一帧和当前帧的像素差异过大，对前一帧的采样点做 clamp 处理，即 $\text{clamp}(C^{i-1}, \mu - k\sigma, \mu + k\sigma)$ 。不过 temporal clamping 只是一个 noise 和 lagging 的 tradeoff 的方法。

4 Specific Filtering Approaches for RTRT

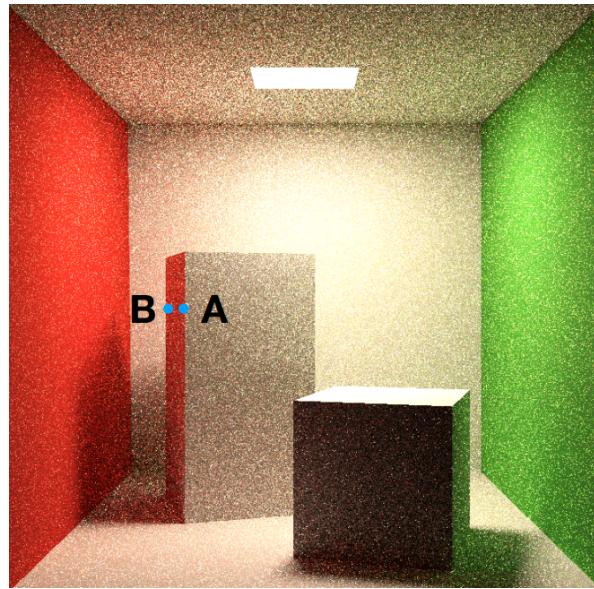
4.1 Spatiotemporal Variance-Guided Filtering (SVGF)

SVGF 考虑使用 3 种因素引导滤波

4.1.1 Depth

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

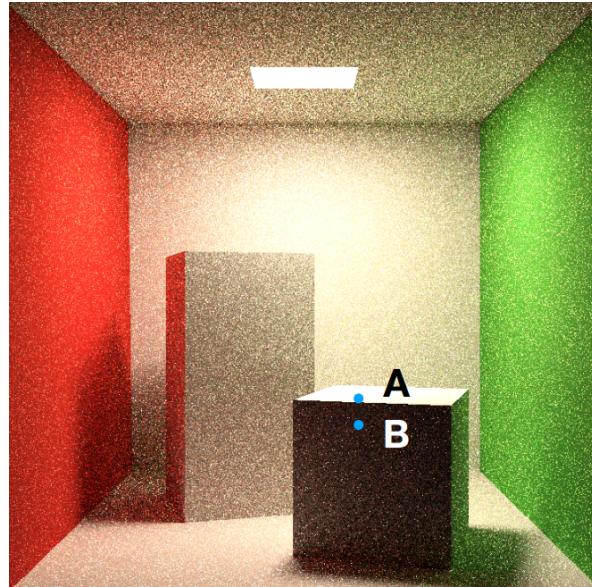
A 和 B 在同一平面上且颜色相似，所以它们应该互相贡献，但是它们的深度差异很大，直接用深度判断贡献较小，所以考虑 A 和 B 沿平面方向上的深度变化。最终在公式中同时考虑深度和平面梯度的关系。



4.1.2 Normal

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

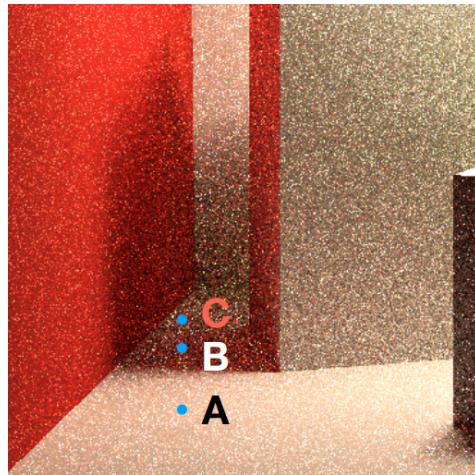
用两个点法线向量求一个点积，由于求出来的值有可能是负值，因此使用 \max 把负的值给 clamp 到 0。而 σ_n 是为了更突出法线变化，放大法线变化效果。如果使用了法线贴图，不要用法线贴图应用之后的法线。



4.1.3 Luminance

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

任意两点间考虑颜色差异，如果颜色差异过大，则认为两点位置靠近边界，此时 A 和 B 的贡献不应该过大。但是由于噪声的存在会出现一些干扰，也就是 B 点虽然在阴影里，但可能刚好选择的点是一个噪声，导致 B 点比较亮，同时 A 不在阴影里也很亮，那么 A 和 B 亮度差异小，就会互相贡献，导致错误。这一点可以用方差解决。



首先将 RGB 转为灰度。计算周围 7×7 范围内的方差，利用 motion vector 计算上一帧对应像素

的方差（相当于时域滤波），最后在周围 3*3 区域做空间滤波。

4.2 Recurrent AutoEncoder (RAE)

RAE 采用循环神经网络，基于 G-buffer 和 1 SPP 的渲染图像进行去噪。它能利用 temporal 的信息，但最后结果有 overblur 的问题。

