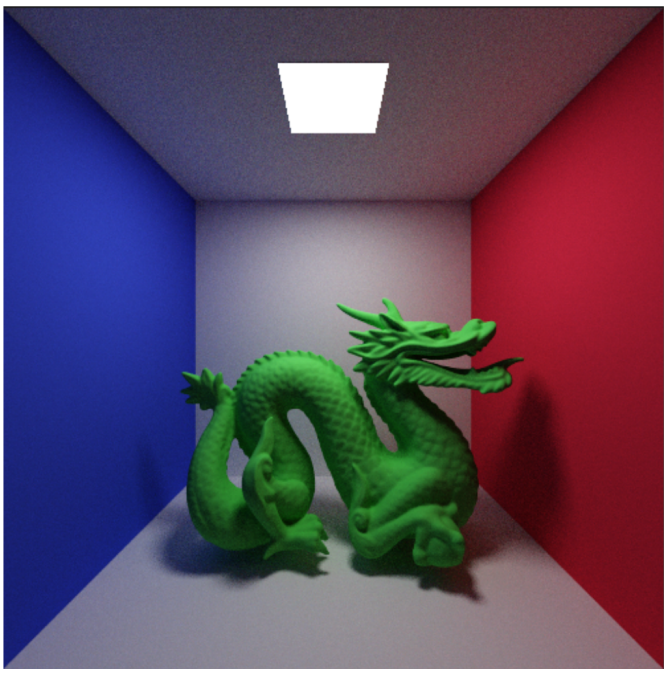
PBRT 系列 11-代码实战-路径追踪

Dezeming Family

2021 年 3 月 15 日

因为本书是电子书，所以会不断进行更新和再版（更新频率会很高）。如果您从其他地方得到了这本书，可以从官方网站：https://dezeming.top/ 下载新的版本（免费下载）。

本书目标：学习光传输方程，路径积分形式。俄罗斯轮盘与路径追踪算法。学习直接采样光和多重重要性采样。实现直接采样光积分器和路径追踪积分器。渲染出下面的效果：



本文于 2022 年 7 月 12 日进行再版，提供了源码。注意图形 GUI 界面和本文中展示的有点区别，但并不影响学习。源码见网址 [https://github.com/feimos32/PBRT3-DezemingFamily]。

前言

本来系列 11 我要介绍微表面材质的，在我写了 2 章以后我突然认识到一个大问题，Whitted 光线追踪在渲染半透明物体时非常有局限性。没法进行多次跟踪折射，也没法实现全反射，渲染效果非常有限。本来规划第 15 本书再写路径追踪的，现在看来还是要先实现路径追踪再去搞材质。

路径追踪在现工业界和光子映射作为主流的渲染算法，虽然更多的渲染技术，即时辐射度、VCM 等，但路径追踪的稳定性和简洁性使其几乎是离线渲染领域的霸主。实时光线追踪技术和滤波去噪技术使得路径追踪技术一直蓬勃发展。

尽管路径追踪涉及一些理论知识，但幸运的是，有了前面的基础，本书并不会很难。

本文的售价是 5 元（电子版），但是并不直接收取费用。如果您免费得到了这本书的电子版，在学习和实现时觉得有用，可以往我们的支付宝账户（17853140351，可备注：PBRT）进行支持，您的赞助将是我们 Dezeming Family 继续创作各种图形学、机器学习、以及数学原理小册子的动力！

20211220：增加了一些关于路径追踪源码的解释和示意图。

目录

一　光传输方程

光传输方程（LTE）是描述场景中光辐射平衡分布的方程（光能守恒，光分布正确）。它给出了表面上某一点的总反射 radiance，包括表面 emission、BSDF 和到达该点的入射光分布。

计算 LTE 难点在于一个点的入射 radiance 受场景中所有对象的几何和散射特性的影响。照射在红色物体上的强光可能会导致场景中附近物体上出现淡红色，或者玻璃可能会将光线聚焦到桌面上形成焦散。导致这种复杂性的渲染算法通常称为全局照明算法，以区别于仅在着色计算中使用有关局部曲面特性信息的局部照明算法。

1 1　基本推导

光传输方程取决于我们在选择使用辐射度量学来描述光时已经做出的基本假设，即波动光学效应不重要，场景中的光分布处于平衡状态。

LET 的关键是能量平衡，既然我们假设照明是一个线性过程，那么系统的能量输入量和能量输出量之间的差值也必须等于发射能量和吸收能量之间的差值。

$$\Phi_o - \Phi_i = \Phi_e - \Phi_a \tag{一.1}$$

其中 Φ_o 表示能量离开物体，Φ_i 表示能量进入物体，Φ_e 表示能量从物体发出，Φ_a 表示能量被物体吸收。

为了加强表面的能量平衡，发出的的 radiance 必须等于自身发射的 radiance（发光物体）加上散射的入射 radiance 根据 BSDF 计算的反射分布。

$$L(p,\omega_o) = L_e(p,\omega_o) + \int_\Omega f(p,\omega_o,\omega_i)L(t(p,\omega_i),-\omega_i)|cos\theta_i|d\omega_i \tag{一.2}$$

其中，$L_e(p,\omega_o)$ 自身表示发出的光，后面的积分项表示散射到 ω_o 方向的光。$L(t(p,\omega_i),-\omega_i)$ 表示从某点沿着 ω_i 方向射到点 p 的 radiance，需要注意 $t(p,\omega_i)$ 就表示光是从该点射到表面上的。

1 2　LTE 的解析解

虽然解析解对通用渲染没有什么帮助，但它可以在实现中帮助调试积分器。如果一个积分器没有计算出一个与解析解相匹配的解，那么显然积分器中有一个 bug。

例如，考虑一个球体的内部，其中球体表面上的所有点都具有朗伯 BRDF，并且在所有方向上都发射恒定的辐射量 L_e。在球面上的任意一点（球内表面）我们有：

$$L(p,\omega_o) = L_e(p,\omega_o) + \int_\Omega L(t(p,\omega_i),-\omega_i)|cos\theta_i|d\omega_i \tag{一.3}$$

所有方向的 L 都是一定的，球体内部任何点的外照射辐射分布（即所有方向的 radiance 都相同）必须与任何其他点相同，所以积分部分得到 $c\pi L$：

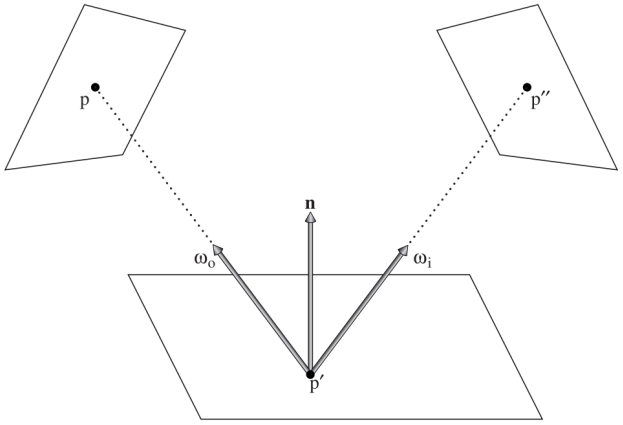
$$L = L_e + c\pi L \tag{一.4}$$

我们还可以这么考虑，当前 p 点的 L 值等于自身发出的 radiance，加上被反射一次以后照到 p 的 radiance，加上反射两次以后照到 p 的 radiance。我们使用反射率：

$$L = L_e + \rho_{hh}(L_e + \rho_{hh}(L_e + ... = \frac{L_e}{1-\rho_{hh}} \tag{一.5}$$

1 3 LTE 的面积分形式

光传输还可以表示为面积分的形式。我们假设光在下面三个点之间反射：



几何项为：

$$G(p \leftrightarrow p') = V(p \leftrightarrow p') \frac{|cos\theta||cos\theta'|}{\|p - p'\|^2} \tag{一.6}$$

其中 $V(p \leftrightarrow p')$ 表示两点间是否被遮挡，如果被遮挡了就为 0，否则为 1。至于为什么有两个 cos 项，其中一个是渲染方程里的与投影面有关的 cos，另一个是来自于对形状采样，如果大家记得如何计算直接采样光源，那么这个 cos 与距离的平方项大家应该不会很陌生。

$$\frac{|cos\theta'|}{\|p - p'\|^2} \tag{一.7}$$

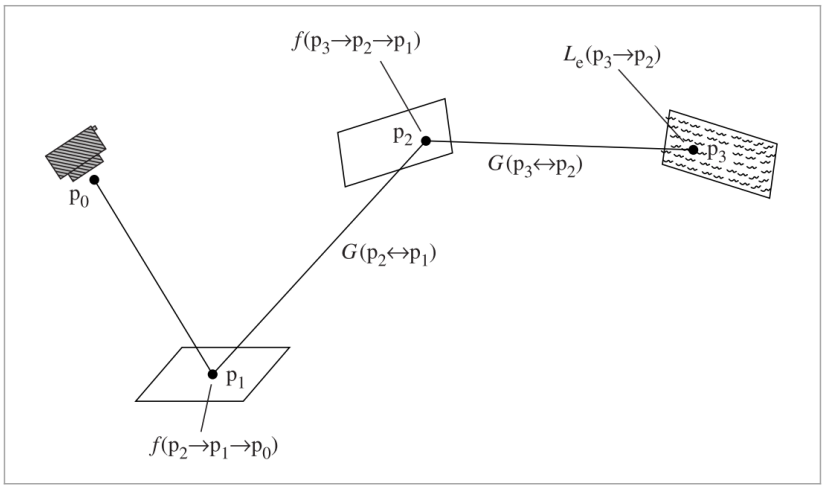
因此光传输方程变为了面积分形式：

$$L(p' \to p) = L_e(p' \to p) + \int_A f(p'' \to p' \to p) L(p'' \to p') G(p'' \leftrightarrow p') dA(p'') \tag{一.8}$$

这里的 A 表示场景中所有的表面，当然，对于某点来说，场景中所有的表面一般不是都可见的，所以可能相当多的表面与当前点的几何项是 0。

1 4 路径上的积分

利用面积积分形式，我们可以导出一种更灵活的 LTE 形式，称为光传输的路径积分公式，它将辐射表示为高维路径空间中路径上的积分。使用路径空间的一个主要动机是，它提供了一个作为路径上显式积分的测量值表达式，而不是由能量平衡方程产生的繁杂的递归来定义。



路径积分我们可以写为:

$$L(p_1 \to p_0) = L_e(p_1 \to p_0) + \int_A L_e(p_2 \to p_1) f(p_2 \to p_1 \to p_0) G(p_2 \leftrightarrow p_1) dA(p_2) \tag{一.9}$$

$$+ \int_A \int_A L_e(p_3 \to p_2) f(p_3 \to p_2 \to p_1) G(p_3 \leftrightarrow p_2) \tag{一.10}$$

$$\times f(p_2 \to p_1 \to p_0) G(p_2 \leftrightarrow p_1) dA(p_3) dA(p_2) + ... \tag{一.11}$$

这样看着还不是很明显，我们令：

$$T(\overline{p_n}) = \prod_{i=1}^{n-1} f(p_{i+1} \to p_i \to p_{i-1}) G(p_{i+1} \leftrightarrow p_i) \tag{一.12}$$

我们设:

$$P(\overline{p_n}) = \overbrace{\int_A \int_A \cdots \int_A}^{n-1} L_e(p_n \to p_{n-1}) \tag{一.13}$$

$$\times (\prod_{i=1}^{n-1} f(p_{i+1} \to p_i \to p_{i-1}) G(p_{i+1} \leftrightarrow p_i)) dA(p_2) \cdots dA(p_n) \tag{一.14}$$

$$= \overbrace{\int_A \int_A \cdots \int_A}^{n-1} L_e(p_n \to p_{n-1}) T(\overline{p_n}) dA(p_2) \cdots dA(p_n) \tag{一.15}$$

这样我们就可以重新写路径积分公式了：

$$L(p_1 \to p_0) = \sum_{n=1}^{\infty} P(\overline{p_n}) \tag{一.16}$$

完美镜面物体和点光源会出现 Delta 分布的场景，虽然处理这种情况会给渲染积分器带来一些额外的复杂性，但我们也仍然经常使用它，因为降低了要计算的积分的维数，将部分积分项变成了一个简单的和。

例如在仅有点光源直接光照时，$P(\overline{p_n})$ 中有单个点光源：

$$P(\overline{p_n}) = \int_A L_e(p_2 \to p_1) f(p_2 \to p_1 \to p_0) G(p_2 \leftrightarrow p_1) dA(p_2) \tag{一.17}$$

$$= \frac{\delta(P_{light} - p_2) L_e(P_{light} \to p_1)}{p(P_{light})} f(p_2 \to p_1 \to p_0) G(p_2 \leftrightarrow p_1) \tag{一.18}$$

1 5　被积函数的划分

科研者已经开发出许多渲染算法，这些算法在某些条件下特别擅长解决 LTE 问题，但在其他条件下却不能很好地（或根本不能）工作。例如，Whitted 积分器仅处理镜面反射的多次反弹，而忽略来自漫反射和光泽 BSDF 的多次散射。而如果您做过光子映射就会知道，光子在漫反射表面存储效果会更好，而光滑表面还是得依靠光线追踪。

因为我们希望能够导出正确的光传输算法，该算法能够考虑所有可能的散射模式，而不忽略任何贡献，也不重复计算其他贡献，因此必须仔细考虑特定解决方案方法所考虑的 LTE 部分。解决这个问题的一个好方法是以各种方式划分 LTE。

$$L(p_1 \to p_0) = P(\overline{p_1}) + P(\overline{p_2}) + \sum_{i=3}^{\infty} P(\overline{p_i}) \tag{一.19}$$

上式右边第一项表示表面自身发出的光。第二项使用直接光照来解决。第三项的计算是路径追踪的重点。

将 $P(\overline{p_n})$ 项划分为小光源 $L_{e,s}$ 照明和大光源 $L_{e,l}$ 照明：

$$P(\overline{p_n}) = \overbrace{\int_A \int_A \cdots \int_A}^{n-1} (L_{e,s}(p_n \to p_{n-1}) + L_{e,l}(p_n \to p_{n-1}))T(\overline{p_n})dA(p_2)\cdots dA(p_n) \tag{一.20}$$

$$= \overbrace{\int_A \int_A \cdots \int_A}^{n-1} L_{e,s}(p_n \to p_{n-1})T(\overline{p_n})dA(p_2)\cdots dA(p_n) \tag{一.21}$$

$$+ \overbrace{\int_A \int_A \cdots \int_A}^{n-1} L_{e,l}(p_n \to p_{n-1})T(\overline{p_n})dA(p_2)\cdots dA(p_n) \tag{一.22}$$

这两个积分可以独立计算，可能使用完全不同的算法或不同数量的样本，选择的方式可以很好地处理不同的条件。只要小光源积分忽略来自大光源的任何发射，大光源积分忽略来自小光源的发射，并且所有光源都被分类为“大”或“小”，最后就可以计算出正确的结果。

同理，BSDF 也可以进行划分，比如划分为完美镜面 BSDF（Delta 分布）和其他 BSDF。

二　俄罗斯轮盘

本章我们讲述俄罗斯轮盘赌的原理。

2 1　使用俄罗斯轮盘赌的原因

估计器 F 的效率定义为：

$$\epsilon[F] = \frac{1}{V[F]T[F]} \tag{二.1}$$

其中 $V[F]$ 表示的是方差，$T[F]$ 表示的是运行时间。

俄罗斯轮盘赌可以提高渲染的效率。例如对于直接采样光时，光源 radiance 设为 L_d，使用蒙特卡洛估计渲染方程的公式为：

$$\frac{1}{2}\sum_{i=1}^{2}\frac{f_r(p,\omega_o,\omega_i)L_d(p,\omega_i)|cos\theta_i|}{p(\omega_i)} \tag{二.2}$$

求和的每一项的大部分计算费用来自于从 p 点追踪阴影光线，以查看从 p 点所看到的光源是否被遮挡。但是对于 BRDF 项非常小的时候，对最终的估计贡献并不会很大。我们可以使用俄罗斯轮盘赌来解决这个问题。

2 2　俄罗斯轮盘赌的方法

选择终止概率 q，当某次要渲染的时候，当当前随机数属于 q 内时，积分值在某些时候不计算，而是使用某个常数值替代；当当前随机数属于 $1-q$ 的那部分时，积分值开始计算，但最终结果需要乘以权重 $1/(1-q)$，即：

$$F' = \begin{cases} \frac{F-qc}{1-q} & \zeta > q \\ c & otherwise \end{cases} \tag{二.3}$$

使用俄罗斯轮盘赌的期望值与正常求积分值是一样的：

$$E[F'] = (1-q)(\frac{E[F]-qc}{1-q}) + qc = E[F] \tag{二.4}$$

终止概率 q 可以用多种方式选择，例如它可以基于所选特定样本计算出的被积函数值，随着被积函数值变小而增加。

俄罗斯轮盘赌从不会减少方差。事实上，除非 $c = F$，否则它总是会增加方差。然而，如果根据概率来选择，从而跳过可能对最终结果作出微小贡献的样本，则确实可以提高效率。

其中一个陷阱是，选择不当的俄罗斯轮盘赌权重可以大大增加方差。想象一下，将俄罗斯轮盘赌应用到所有的相机光线，终止概率为 .99：我们只跟踪 1% 的相机光线，每个光线的权重为 $1/.01 = 100$。从严格的数学意义上讲，得到的图像仍然是“正确”的，尽管从视觉上看结果会很糟糕：大部分是黑色像素，还有一些非常明亮的像素。

2 3　splitting 算法

虽然俄罗斯轮盘赌减少了评估不重要样本所花的精力，但为了提高效率，splitting 算法会增加样本的数量。再次考虑仅由直接照明计算反射的问题。忽略像素滤波，该问题可以被写为像素 a 区域和每个 (x,y) 像素位置的表面上的可见点处的方向 S^2 的球体上的二重积分：

$$\int_A\int_{S^2} L_d(x,y,\omega)\ dxdyd\omega \tag{二.5}$$

其中 L_d 是当前像素 (x,y) 射到的某个表面在方向 ω 上接受到的光照量。如果场景中有许多光源，或者如果有区域光投射软阴影，则可能需要数十或数百个样本来计算具有可接受方差水平的图像。不幸的是，

每个样本都需要在场景中跟踪两条光线：一条是从成像面上的位置 (x, y) 计算第一个可见表面，另一条是沿着 ω 到光源的阴影光线。

这种方法的问题是，如果取 $N = 100$ 个样本来估计这个积分，那么将跟踪 200 条光线：100 条相机光线和 100 条阴影光线。然而，100 个相机光线可能比良好的像素抗锯齿所需的光线多得多，因此对最终结果中的方差减少的贡献相对较小。*splitting* 算法解决了这个问题，通过形式化的方法在集成的某些维度中为其他维度中的每个样本抽取多个样本。

通过 *splitting* 算法，可以用 N 个图像样本和 M 个光样本来编写该积分的估计器，在像素 (x, y) 采样到的每个样本的光贡献量为：

$$\frac{1}{M}\sum_{i=1}^{M}\frac{L_d(x, y, \omega_i)}{p(\omega_i)} \tag{二.6}$$

因此，我们可以只采集 5 个图像样本，但每个图像样本采集 20 个光样本，总共跟踪 105 条光线，而不是 200 条。采集的 100 个区域光样本，来计算高质量的软阴影。

三　路径追踪算法

Kajiya（1986）在第一篇描述光传输方程的论文中介绍了路径追踪技术。路径跟踪生成从相机开始到场景中光源结束的散射事件路径。可以认为它是 Whitted 方法的扩展，Ray 反弹的过程包括 Delta 分布和非 Delta BSDF 以及光源，而不仅仅是在 Delta 项（完美镜面处）进行反弹。

虽然直接从基本光传输方程导出路径跟踪稍微容易一些，但我们将从路径积分形式来处理它，这有助于对路径积分方程的理解，并使向双向路径跟踪的推广更容易被理解。

3 1　算法简介

相机 Ray 与场景的第一个交点 p_1 传到相机像素 p_0 的 radiance 可以表示为：

$$L(p_1 \to p_0) = \sum_{i=1}^{\infty} P(\overline{p_i}) \tag{三.1}$$

这就会带来两个问题：第一，$P(\overline{p_i})$ 有无穷多项，我们应该怎么计算。第二，给定一个 $P(\overline{p_i})$，它的下一个用于蒙特卡洛估计的采样方向应该如何取呢。

对于路径跟踪，我们可以利用这样一个事实：对于物理上有效的场景，具有更多顶点的路径比具有更少顶点的路径散射的光更少（并不完全正确，但大多数情况就是如此），这是 BSDF 中能量守恒的自然结果。

因此，我们将始终估计前几个 $P(\overline{p_i})$ 项，然后开始应用俄罗斯轮盘赌在有限个项之后停止采样，俄罗斯轮盘不会引入偏差。

$$P(\overline{p_1}) + P(\overline{p_2}) + P(\overline{p_3}) + \frac{1}{1-q}\sum_{i=4}^{\infty} P(\overline{p_i}) \tag{三.2}$$

以这种方式使用俄罗斯轮盘赌并不能解决需要计算无穷和的问题，如果我们把这个想法更进一步，设每一项终止概率 q_i：

$$\frac{1}{1-q_1}\left(P(\overline{p_1}) + \frac{1}{1-q_2}\left(P(\overline{p_2}) + \frac{1}{1-q_3}\left(P(\overline{p_3}) + \cdots\right.\right.\right. \tag{三.3}$$

3 2　路径采样

假如我们的积分路径有 i 段，那么就有 $i+1$ 个顶点，其中第 0 个顶点位于成像面的像素处，第 i 个顶点位于光源。

如果我们定义关于场景中 n 个物体的离散概率，然后对第 i 个物体表面上的路径顶点进行采样的概率应为：

$$p_i = \frac{A_j}{\sum_j A_j} \tag{三.4}$$

其中 A_i 表示第 i 个物体的表面积。我们假设所有物体都是均匀采样的，那么最终在某个点采样到的概率密度为：

$$p_i = \frac{A_i}{\sum_j A_j}\frac{1}{A_i} = \frac{1}{\sum_j A_j} \tag{三.5}$$

给定一组顶点以这种方式采样，然后我们可以采样场景中光源上的最后一个顶点，以相同的方式定义其 Pdf。虽然我们可以使用采样路径顶点的相同技术对灯光上的点进行采样，但这将导致高方差，因为对于不在光源照射范围的所有路径，计算值都为零。期望值仍然是积分的正确值，但收敛速度非常慢。更好的方法是只对发光物体的区域进行采样，并相应地更新概率。

对于如何用这种通用方法设置采样概率，我们可以想出各种方法，例如如果我们知道来自少数对象的间接照明对场景中的大部分照明都有贡献，那么我们可以为在这些对象上生成路径顶点分配更高的概率，并适当更新采样权重。然而，这种方式的采样路径有两个相互关联的问题。第一种可能导致高方差，而第二种可能导致不正确的结果。

第一个问题是，如果路径中有一对相互不可见的相邻顶点，那么许多路径将没有贡献。考虑在复杂的建筑模型中应用此面积采样方法：如果路径中的相邻顶点之间有一堵或两堵墙，则这对路径就会没有贡献，并且估计方差会很大。

第二个问题是，如果被积函数中有 Delta 函数（例如点光源或完全镜面反射 BSDF），这种采样技术将永远无法选择路径顶点，即使没有 delta 分布，随着 BSDF 变得越来越光滑，几乎所有的路径都将具有较低的贡献，因为光滑物体的 BSDF 在大多方向具有较小或零的值，估计方差会很大。同理，如果没有明确采样方案，小面积光源也会是方差来源。

3 3 增量构建路径

增量构建路径可以解决上面的两个问题。在每个顶点处，BSDF 采样出新的路径，我们尝试寻找一条总体贡献更大的路径，通过作出一系列选择，找到具有重要局部贡献的方向。虽然可以想象这种方法可能无效的情况（比如反弹了很多次之后都没有打到光源），但它通常是一种很好的策略。

由于该方法根据立体角采样 BSDF 来构造路径，并且由于路径积分 LTE 是场景中表面积上的积分，因此需要进行转换——将表面积相关的积分转换到立体角相关：

$$
\begin{aligned}
L(p' \to p) &= L_e(p' \to p) + \int_A f(p'' \to p' \to p) L(p'' \to p') \frac{|cos\theta'||cos\theta''|}{\|p - p'\|^2} dA(p'') \\
L(p' \to p) &= L_e(p' \to p) + \int_{\mathcal{H}^2} f(\omega' \to p' \to p) L(\omega' \to p') |cos\theta'| d\omega'
\end{aligned}
\tag{三.6}
$$

这里我将 $f(\omega' \to p' \to p)$ 描述为从 ω' 射到点 p'，再散射到 p 的 BSDF。$\mathcal{H}^2$ 表示 p' 点所在表面的法向量的所在半球。

将根据立体角的概率密度 p_ω 转换为根据面积 p_A 的概率密度（PBRT-V3 原书上下式是 $|\cos\theta_i|$，但是因为我们当前点是 p_i，我们打算采样 p_{i+1}，所以相关对表面积分的 cos 项应该是在 p_{i+1} 所在表面上的法向量相关的，也可能是我理解错 PBRT 书的意思了，但目前来看我的写法是没有问题的）：

$$
p_A = p_\omega \frac{|cos\theta_{i+1}|}{\|p_i - p_{i+1}\|^2} \tag{三.7}
$$

其中，p_A 是采样表面的概率密度，p_ω 是采样方向的概率密度，θ_{i+1} 是 p_{i+1} 点处入射光线与表面法线的夹角。

这种修正使得除了 $cos\theta_{i+1}$ 项外，几何项 $G(p_i \leftrightarrow p_{i+1})$ 的所有项从 $P(\overline{p_i})$ 中抵消（其实这就相当于把下面的对面积积分的公式改为了对 BSDF 采样方向，上面所述的东西 [1] 比较严谨，但其实无非就是：对光采样，也对 BSDF 采样，只不过对 BSDF 采样的时候，wi 方向与物体相交后，该物体不能是面光源）。

$$
L(p' \to p) = L_e(p' \to p) + \int_A f(p'' \to p' \to p) L(p'' \to p') \frac{|cos\theta'||cos\theta''|}{\|p - p'\|^2} dA(p'') \tag{三.8}
$$

此外，我们已经知道 p_i 和 p_{i+1} 必须是相互可见的，因为我们是跟踪 Ray 来找到 p_{i+1} 的，所以可见性项通常等于 1。考虑这一点的另一种方法是光线跟踪提供了一种对 G 的可见性分量进行重要采样的操作。

因此，如果我们使用这种采样技术，但我们仍然从光源 $p_A(p_i)$ 表面上的某些分布中采样最后一个顶点 p_i，则路径的蒙特卡罗估计值为（下面这个公式很简单，如果您一时半会不理解是什么意思也没有关系，直接看下一节算法描述就好了）：

$$
\frac{L_e(p_i \to p_{i-1}) f(p_i \to p_{i-1} \to p_{i-2}) G(p_i \leftrightarrow p_{i-1})}{p_A(p_i)} \times \left(\prod_{j=1}^{i-2} \frac{f(p_{j+1} \to p_j \to p_{j-1}) |cos\theta_j|}{p_\omega(p_{j+1} - p_j)} \right) \tag{三.9}
$$

上面的式子是一条最终路径，该路径抛去了每个顶点上的直接光照，只考虑相机 Ray 不断反弹最后到达光源作为终点。

3 4 算法描述

把算法流程写下来，为了大家好理解，我先写成递归跟踪光线的形式，到我们真正实现的时候则会使用循环跟踪光线的形式。

```
Li(p, wo)//从p点沿wo方向发出的radiance
    //采样直接光照
    以概率密度L_pdf=1/A采样光照点，光照值为L_d，p到光源处采样点x方向为wi
    测试可见性
    如果可见：L_direct = L_d*f(wo,wi)*cos(theta1)*cos(theta2)/|x-p|^2/L_pdf
    //采样间接光照
    以概率pR测试俄罗斯轮盘
    根据BSDF选择wi方向，概率密度为S_pdf
    Ray(p,wi)计算与场景相交于x
    如果击中非光源：
        L_indirect = Li(x,-wi)*f(wo,wi)*cos(theta1)/S_pdf/(1-pR)
    return L_direct + L_indirect
```

四　采样与光的分布函数

本章研究 1 维与 2 维分布函数。

4 1　一维分段常函数分布

1 维分段常函数：

$$f(x) = \begin{cases} v_0 & x_0 \le x \le x_1 \\ v_1 & x_1 \le x \le x_2 \\ \cdots \end{cases} \tag{四.1}$$

我们假设分段常函数定义在 [0,1] 范围内，若想扩展到其他范围只需要倍乘即可。累积概率密度函数 CDF 可以很容易就计算得到。

Distribution1D 类的构造函数输入 n 个分段常函数值，之后它会复制这些值到自己的内存区，然后计算 CDF。

SampleContinuous 函数使用随机数来采样，得到在 [0,1] 区间的采样值及其概率密度。该函数首先调用 FindInterval 来寻找 u 在哪个 CDF 区间。

SampleDiscrete 用来采样到离散值，例如，我们只需要对 n 个区间进行采样，而不需要知道具体采样到该区间的哪个值，因此我们就可以使用该函数。举个例子，我们把无限面光源划分为 100 个小块，有的小块更亮，我们就多采样这些小块，有的小块不亮，我们少采样这些小块，该函数就帮我们选择去哪个小块进行采样。

Distribution1D 的具体方法其实很简单，移植也没有什么困难的，因此我们不再赘述，想弄明白每句代码的同学可以自己看源码（最好的办法就是你自己写一遍，顶多一两个小时，之后你会发现你写的和 PBRT 中的实现几乎是一样的过程）。

4 2　二维分段常函数分布

二维与一维情况类似，Distribution2D 非常简单就能移植到自己的系统中。

Distribution2D 的 Pdf 函数是求采样值的 Pdf，我们构建个程序测试一下：

```
1  float aa[16] =
       {1.0,4.5,7.6,2.1,4.1,9.1,1.8,2.6,11.4,16.3,6.4,7.2,1.5,1.9,2.1,5.4};
2  Distribution2D d2d(aa,4,4);
3  Point2f ta(0.423, 0.769); float apdf;
4  Point2f tu = d2d.SampleContinuous(ta,&apdf);
5  float bpdf = d2d.Pdf(tu);
```

测试可知，apdf 的值等于 bpdf 的值。

4 3　光采样分布

我们需要采样光，自然而然需要合理的采样方案。core 文件夹下的 lightdistrib.h 和 lightdistrib.cpp 文件中定义的 LightDistribution 基类描述了场景中光源的分布情况，表示我们采样光源的时候对每个光源应该采样的次数的比重，例如均匀采样的分布，例如根据能量进行采样的分布，还有例如基于空间位置进行采样的分布。我们以前都是进行均匀采样的，现在我们需要了解一下其他的采样方法。

PathIntegrator 路径积分器会在预处理阶段处理场景中的光，根据我们选择的处理光分布的方法（例如按能量或者均匀采样）来将每个光源按照重要性生成 1 维分段概率密度函数。

```
1  UniformLightDistribution：均匀采样，每个光源被采样到的概率相同
2  PowerLightDistribution：按能量划分，总能量更大的光源重要性更大
```

```
SpatialLightDistribution：按空间重要性划分（空间划分为体素）
```

鉴于我们目前的康奈尔盒只有两个光源（两个三角形面光源构成一个正方形），我们就只使用均匀采样就可以了。一次介绍过多的方法只会增加复杂性。（我觉得不同的光分布可能是 PBRT 书 [1] 发布以后才实现的，因此书中并未提到这些内容，但源码中的注释非常详细）。

LightDistribution 基类我们可以直接移植。派生类只移植 UniformLightDistribution 就可以了。我们修改一下创建光采样分布的函数：

```
std::unique_ptr<LightDistribution> CreateLightSampleDistribution(
    const std::string &name, const Scene &scene) {
    return std::unique_ptr<LightDistribution>{
        new UniformLightDistribution(scene)};
}
```

五　直接光照积分器

在实现路径追踪之前，我们需要研究它的基础类：直接光采样。

你可能会说，Whitted 不也是直接光采样吗？其实很相近，但也有很多不同。对于直接光照积分器而言，有很多我们需要考虑的细节，本章的目的是实现直接光照效果。

直接光照采用了多重重要性采样，如何理解多重重要性采样，简单来说，就是我们使用两种采样方法，然后为了保证结果无偏，需要使用组合策略将两种方法得到的值组合在一起。

5 1　直接光照积分器类

DirectLightingIntegrator 类定义于 core 文件夹下的 directlighting.cpp 和 directlighting.h 文件中。该类的 Li 函数与 Whitted 函数基本一致，只是对光采样的过程根据 LightStrategy 来调用不同的函数对光采样。

该类的 Preprocess 会根据 LightStrategy 判断，如果要对所有的光源都采样，则需要进行预处理，我们在《采样器和渲染器》书中提到过，Request2DArray 函数只在环境光遮蔽积分器（ao.h 和 ao.cpp）以及直接光照积分器中会用到。我提一下使用直接光照积分器中采样序列的流程：

```
1 积分器Render函数Preprocess，申请采样序列（分配内存），为每个光源申请
    samplesPerPixel乘以每个光源采样样本数
2 在对每个像素的渲染中：
3     Render函数调用积分器的StartPixel，为序列赋随机数
4     使用随机序列
```

回忆《采样器和渲染器》一书，RoundCount 的作用，表示我们虽然选择了对光采样的样本数，假如采样 9 个，但是有时候对于采样器而言生成 2 的 n 次方个样本更快，因此我们总共生成了 16 个样本（不过对于 halton 采样器来说我们设置采样几个样本就会得到几个样本）。

DirectLightingIntegrator 类的移植非常简单，除了采样光源的几个函数，我们很容易就能把它移植到自己的系统上，注意 MemoryArena 对象根据自己是否移植过它来进行删减。我倾向于使用 new，因此删除该对象。

5 2　采样光源

本节我们研究下面两个函数：

```
1 UniformSampleAllLights
2 UniformSampleOneLight
```

我们首先大体看一下这两个函数做了什么。

UniformSampleAllLights 函数遍历光源数组，对每个光都采样。如果 Request2DArray 数组没有申请到，那么就只对每个光采样一次；否则就对每个光都采样设定的样本量，得到最终结果。

UniformSampleOneLight 函数则是根据光分布来抽样出对哪个光源采样，如果有 Distribution1D 对象，则 lightPdf 就是 SampleDiscrete 求得的概率密度，否则相当于每个光源被采样到的概率相同，lightPdf 就是 1 除以光源数量。

所以这两个函数其实很简单，复杂性主要在于它们都调用了 EstimateDirect 函数。我们先简单定义一下 EstimateDirect 函数，返回 Spectrum(0.f) 即可，然后移植上面的两个函数到我们的渲染器中。

编译不报错后，本节就结束了。

5 3　直接采样光的算法

现在只剩最后一个函数 EstimateDirect 我们的直接对光采样积分器就实现了。

以前学习 MIS 的时候资料确实特别少，唯一能看的就是 [5] 这篇几百页的论文了。若要研究 MIS 的无偏和方差，需要一定的统计学知识，我这里只讲我们是怎么使用多重重要性采样（MIS）来保证无偏渲染的。

我们现在使用蒙特卡洛方法估计直接光照（L_d 表示直接光照）：

$$\frac{1}{N}\sum_{i=1}^{N}\frac{f_r(p,\omega_o,\omega_i)L_d(p,\omega_i)|cos\theta_i|}{p(\omega_i)} \tag{五.1}$$

这里的 $p(\omega_i)$ 表示的就是我们生成 ω_i 方向的概率密度。随机方向可以由对光采样来得到，也可以由对 BSDF 采样来得到。

假如我们使用了两种采样方法，总共采样了 N 次，两种方法各自采样了 n_1 和 n_2 次，即：

$$N = n_1 + n_2 \tag{五.2}$$

这两种采样技术产生的样本概率密度分别为 $p_1(x)$ 和 $p_2(x)$。

这时我们估计的结果就是：

$$\begin{aligned} F &= \left(\sum_{j=1}^{n_1}\right)\frac{\omega_1(x)f(x)}{n_1p_1(x)} + \left(\sum_{j=1}^{n_2}\right)\frac{\omega_2(x)f(x)}{n_2p_2(x)} \\ &= \left(\sum_{j=1}^{n_1}\right)\frac{n_1p_1(x)}{n_1p_1(x)+n_2p_2(x)}\frac{f(x)}{n_1p_1(x)} + \left(\sum_{j=1}^{n_2}\right)\frac{n_2p_2(x)}{n_1p_1(x)+n_2p_2(x)}\frac{f(x)}{n_2p_2(x)} \end{aligned} \tag{五.3}$$

这里面的权重 ω_i 有多种选择方式，比如上面使用了平衡启发式，除了平衡启发式还有幂启发式（“启发”其实没有什么含义，就是表示“基于直觉的、捏造的”意思，大家不要产生什么其他联想）。

对于光滑材质，肯定采样 BSDF 更好，小于小光源，肯定采样灯光更好，但我们一般不知道应该怎么采样，所以就用两种采样方法，再通过上面的 MIS 方法把它们组合起来。当你用了一种采样技术求得采样值后，你还需要将这个采样值计算它在另一种方法下产生的概率，然后再计算 ω_i 权重。

若真的想把 MIS 研究透，建议阅读一下 [5] 论文。

EstimateDirect 函数主要过程很简单，首先先对光源进行多重重要性采样，然后再对 BSDF 进行多重重要性采样。强烈建议大家先移植再阅读源码，因为移植的时候把用于体渲染的参与介质去掉以后，其实代码非常清晰和简单。

对光多重重要性采样中，首先定义了两个 Pdf：lightPdf 和 scatteringPdf。然后对光采样，得到 lightPdf 值和光照值 Li 和光照方向-wi，之后再计算散射 scatteringPdf 值和 BSDF 分布。若 f 不为黑，则启用幂启发式算法来求混合权重。

对物体采样 BSDF 时首先判断要采样的光是否是 Delta 分布的点光源，如果不是就可以继续采样。得到两个 Pdf 来混合权重，得到最终值。

5 4 程序测试

首先在 SamplerIntegrator::Render 函数中加入（以前可能用不到所以没有添加）：

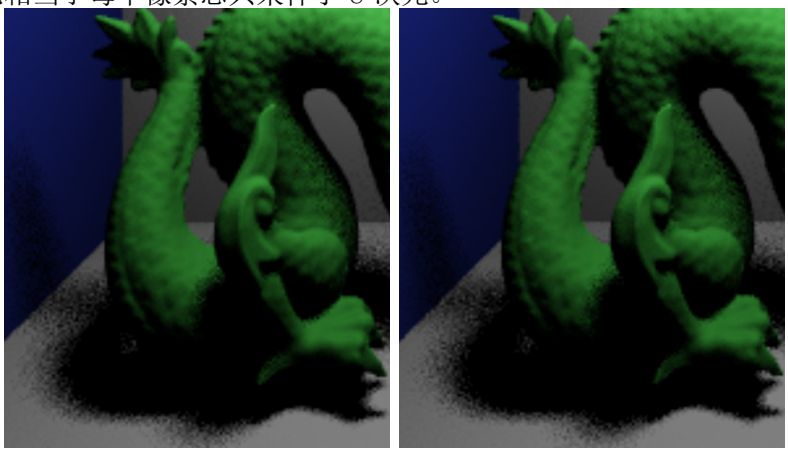
```
1    Preprocess(scene, *sampler);
```

虽然直接光照可以传入光采样分布（调用 UniformSampleOneLight 时），但鉴于 PBRT 中没有使用，我们也就不使用光分布了。后面路径追踪的时候，PathIntegrator 有存储光采样分布的成员变量。

积分器的构造也很简单：

```
1    std::shared_ptr<Integrator> integrator = std::make_shared<
         DirectLightingIntegrator>(LightStrategy::UniformSampleOne, 15, camera
         , sampler, ScreenBound);
```

渲染结果如下：左边是使用 DirectLightingIntegrator 算法，采样 halton 采样器，采样策略是 UniformSampleOne，每个像素采样 8 次，用时 0.686135 秒。右边是使用 WhittedIntegrator 算法，采样 halton 采样器，每个像素采样 4 次，用时 0.4028 秒。直接光采样 8 次相当于每个像素对光采样了 8 次（我们场景中有两个三角形的光源），Whitted 积分器每像素采样 4 次，但是每像素都会对所有灯光采样一遍，因此也相当于每个像素总共采样了 8 次光。



理论上来说，粗糙物体小光源更适合对光采样，光滑物体大光源更适合对 BSDF 采样。而我们的场景中光源较大，物体也很粗糙，因此两种方法（重要性采样和多重重要性采样）得到的结果没有太大差别。

六　路径追踪的实现和本书结语

首先值得一提的是，PBRT 书 [1] 和源码中已经有些地方不一样了，比如对光采样的函数 UniformSampleOneLight，源码中加入了光采样分布。我们本文中并没有描述那些新加入的方法，只使用的均匀采样，所以其实没有什么区别。

路径追踪算法一章的最后一节中，我们说过，路径追踪器最后一个路径一定会采样到光源。您可能会说，如果最后一个路径没有采样到光源那不就不对了吗？其实不是这样。

我们试想这个递归的过程，如果我们假设当前一轮是最后一轮 i，那么当前一轮是上一轮 i-1 对 BSDF 采样得到的新路径。如果 i 这个路径没有采样到光，同时根据轮盘赌，不再继续跟踪，就意味着上一轮 i-1 采样 BSDF 的过程得到了 0，那么如果上一轮直接采样光得到了值，则说明其实上一轮才是最后一轮。因此 Ray 最终一定会采样到光源而后截止。

6 1　路径追踪的基本结构

PathIntegrator 类定义在 path.h 和 path.cpp 文件中。

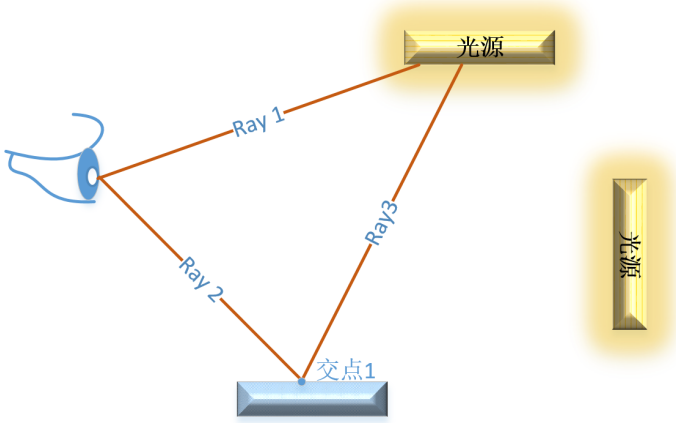
Preprocess 函数在预处理中会构造光采样分布，默认使用面积来均匀采样（所有光源每个点被采样到的概率相同，因此面积较大的面光源更容易被采样到）。

Li 函数的结构表示如下（去掉参与介质与次表面散射的内容）：

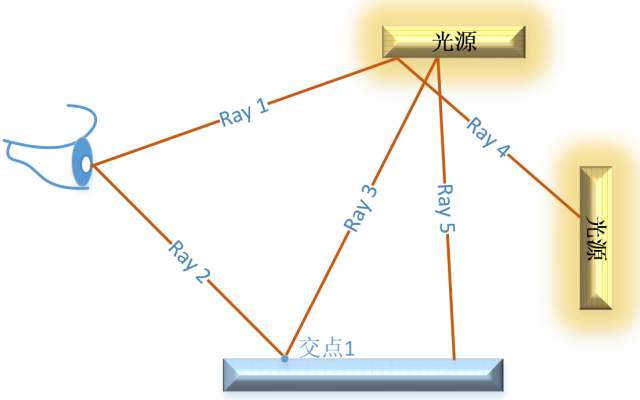
```
1   Spectrum L(0.f), beta(1.f);
2   生成Ray
3   bool specularBounce = false;
4   for(int bounces = 0;;++bounces){
5       //与场景计算交点isect
6       ......
7       //如果是第一次相交或是镜面：如果有交点，L+=交点表面自身发出的光；否
            则，L += 环境光
8       ......
9       //达到最大反弹深度maxDepth或与场景没有交点就终止光线
10      ......
11      //计算散射BSDF
12      ......
13      //采样直接光照值
14      ......
15      //采样BSDF产生新路径并根据俄罗斯轮盘确定是否终止路径
16      ......
17  }
18  return L;
```

跟踪路径中，beta 的作用就是不断跟踪 BSDF 间接照明的权重。反射次数越多，beta 的值就越低。beta.y() 表示 RGB 转 XYZ 的 Y 值，我们在《颜色与光谱》中说过，Y 值一般指亮度，该值用于俄罗斯轮盘，决定是否终止光线。

有了基本步骤的描述，代码就显得浅显易懂，所以不逐行解释了。这里提一下第二步，如果是第一次求交（bounces == 0）或者是镜面（specularBounce），则判断当前有没有交点。没有的话就采样个环境光照明然后就退出了；有交点的话，就计算这个交点发出的光。所以说，如果在路径中与光源相交，除非是第一次相交（或者上一次反弹经过了镜面反射），否则并不会再把采样到的光源的贡献记录进来，这样就保证了能量的守恒（因为在 UniformSampleOneLight 函数中已经对前面的交点计算了一次反弹的照明，如果再把与当前的交点的照明考虑进来，则就会能量过剩）。注意，从别的光源上照射到 diffuse 面光源上再进行反弹的光也是会被正确计算的，因为路径追踪时并不会在你采样到光源的时候就终止：



如上图，当 Ray 1 采样到光源时，就会计算得到：L+= 交点表面自身发出的光。Ray 3 采样到光源以后，并不会相加光源自身发出的光，因为在交点 1 时已经估计了对交点 1 的直接光照，再加上光源自身发出的光后会重复。当 Ray 1 和 Ray 2 采样到交点以后，按照其表面材质属性来进行下一次反弹：



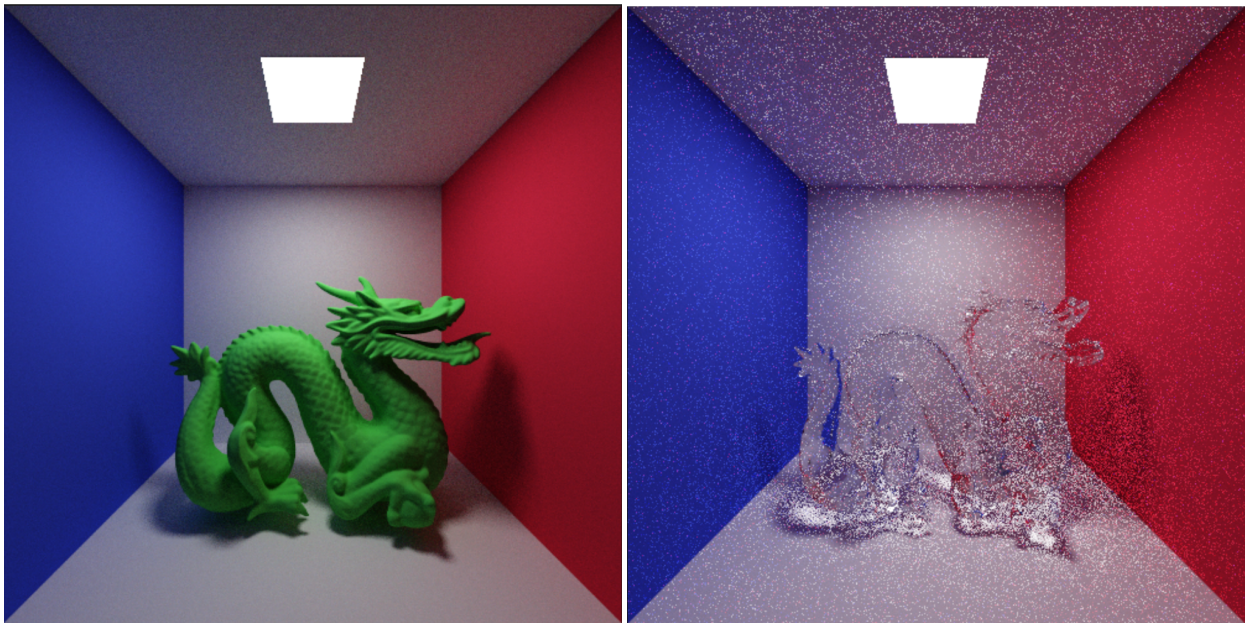
所以说，竖着的光源发出的光可以发射到横着的光源上，然后再照射到人眼里。这也是完全符合物理规律的。

6 2 路径追踪的移植与测试

移植很简单，删除与参与介质和次表面散射相关的内容就可以了（每次删除大片代码的时候我的内心都是很焦灼的，毕竟以后我们还得再补回来，参与介质很重要，等我们要渲染烟雾的时候这些内容都得添加回来）。我们构建并测试，每像素采样 64 个点。一个采用 lambertian 反射模型，另一个采用 FresnelSpecular 散射模型。

```
1    std::shared_ptr<Integrator> integrator = std::make_shared<PathIntegrator
       >(maxDepth, camera, sampler, ScreenBound, 1.f);
```

得到渲染结果：



可以看到，左图中龙的下方出现一定的绿色，靠近红墙的白墙上出现浅红，这就是颜色溢出的效果（color bleeding）。右图噪点非常多，这其中有很多原因。路径追踪中想解决这种情况就得需要海量的采样，或者使用双向方法，例如光子映射。

6 3 总结

简单来说，路径追踪其实就是，计算一次反弹照射的光，然后计算两次反弹照射的光，然后计算三次反弹照射的光，以此类推。我们需要在采样路径的时候，用蒙特卡洛方法来逐步估计。

到现在，本书就结束了，本书总共写了不到两天时间。很多人都把 PBRT[1] 的路径追踪当成最终目标，而我们目前已经把这个目标实现了，顺带着把直接采样光的积分器也实现了。

下一本书我们就开始研究微表面材质，将这些材质实现以后，就可以渲染出很好看的图像了！

参考文献

[1] Pharr M, Jakob W, Humphreys G. Physically based rendering: From theory to implementation[M]. Morgan Kaufmann, 2016.

[2] Shirley P. Ray Tracing in One Weekend[J]. 2016.

[3] Shirley P. Ray Tracing The Next Week[J]. 2016.

[4] Shirley P. Ray Tracing The Rest Of Your Life[J]. 2016.

[5] Veach E . Robust Monte Carlo Methods for Light Transport Simulation[D]. Stanford University. 1998.