Dezeming Family

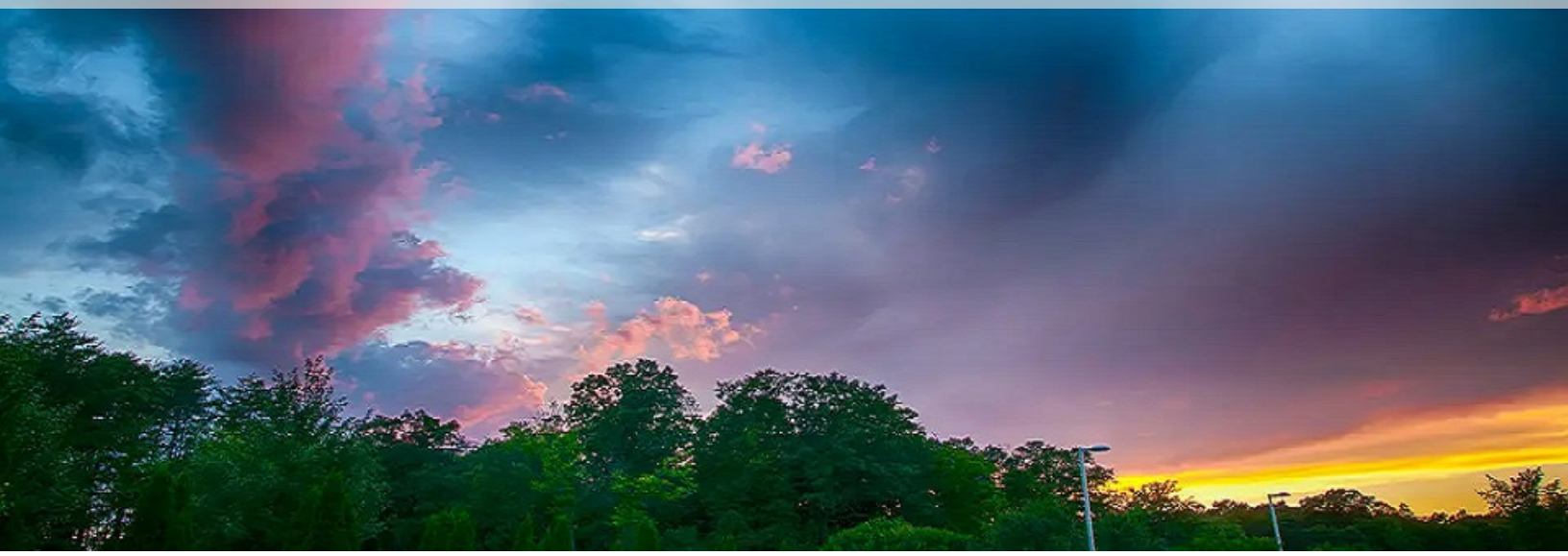
PBRT 系列 24-高级积分器-双向路径追踪

Art. No 0
ISBN 000–00–0000–00–0
Edition 0.0

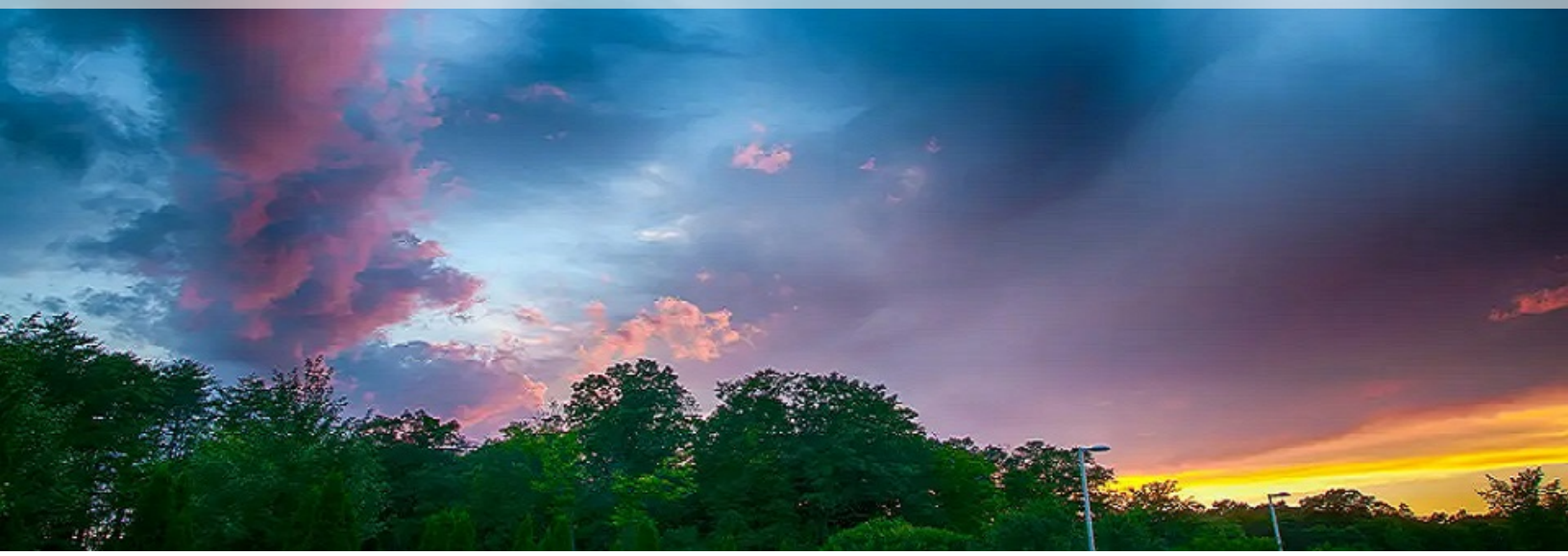
Cover design by Dezeming Family

Published by Dezeming
Printed in China

目录



前言及简介



DezemingFamily 系列文章和电子书**全部都有免费公开的电子版**，可以很方便地进行修改和重新发布。如果您获得了 *DezemingFamily* 的系列电子书，可以从我们的网站 *[https://dezeming.top/]* 找到最新的版本。对文章的内容建议和出现的错误也欢迎在网站留言。

0.1 本文前言

简单的渲染积分器只需要从相机发出光线，而双向方法则同时从相机和光源发出光路径，然后将每次散射的顶点连接起来。

双向光传输的构造基石是很神奇的。一方面，光散射的物理性质通常相对于光传输方向是可逆的，这导致从光源或从相机开始的散射路径的数学表达式非常相似。另一方面，根据路径方向，这两种方法之间存在细微但重要的差异（我们会首先介绍这些差异）。

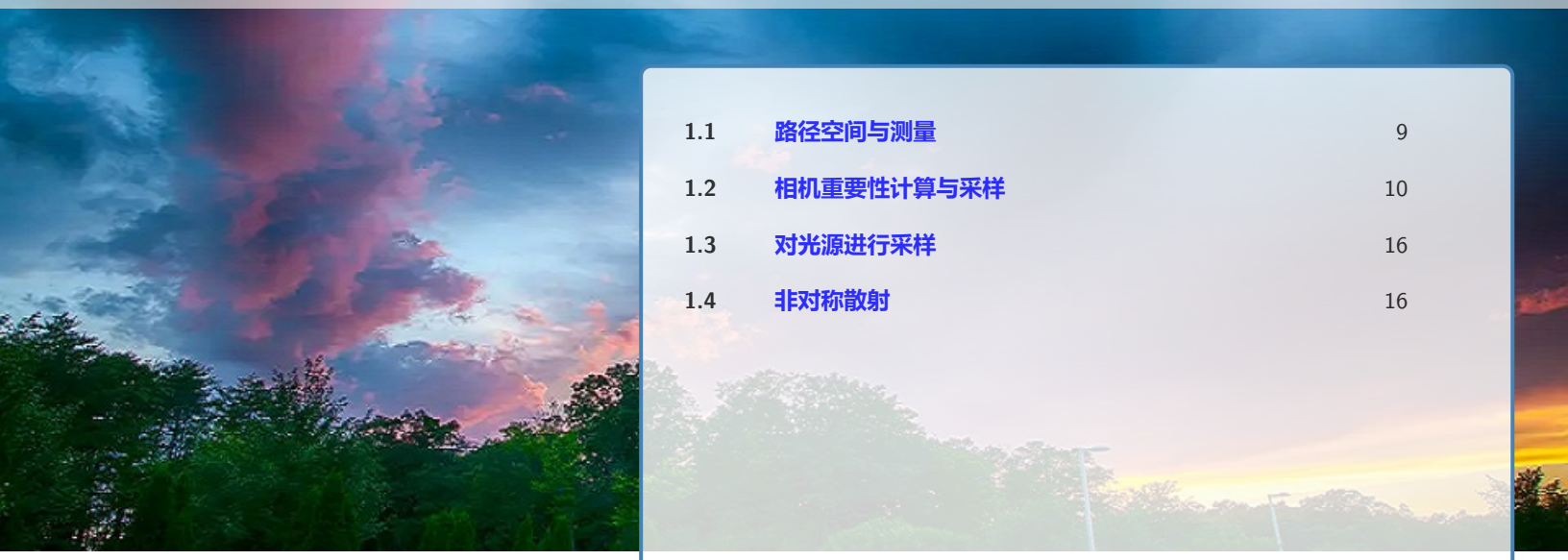
双向路径跟踪这种无偏方法可以比常规路径跟踪更有效，因为它具有双向性质，并且通过将多重要性采样（我们还会介绍一下多重重要性采样）应用于重新加权路径贡献，进一步减少了方差。

双向路径追踪的原理最好还是参考论文 [2]，或者参考 DezemingFamily 的《模拟光传输的鲁棒的蒙特卡洛方法（Eric Veach）全文解读》，我们这里只会描述 PBRT 中的流程。随着写作的深入，越来越觉得“光线追踪”这种说法非常不严谨，ray tracing 本意应该是射线追踪，而在国内由于早期引入的专家们翻译问题，目前大家都用“光线”一词来描述用于追踪路径的射线，本文中，“采样光线”和“射线”的意义是相同的。

双向方法难度很大，无论是理解上还是编程实现上，都需要很多基础知识，为了让读者更好地理解双向过程，我们刻意设计了很多示意图，比如：

希望在丰富的图示和解释下，能够帮助读者尽可能更容易地突破双向方法这个难啃的骨头。我们还在本文最后一章尝试将整个 BDPT 方法串联讲解一遍，加深读者对算法整体的把握。但是我还是不得不在一开始就提醒一下：双向路径追踪是一个很难很复杂的算法，它的实现和理解绝对不是很轻松就能够搞定的，这里面需要大量的反复思考。

1. 路径空间度量方程



本章讲解从相机发射以及从光源发射的路径之间的区别与联系。

1.1 路径空间与测量

光积分形式可以简单写为如下结果：

$$L(p_1 \to p_0) = \sum_{n=1}^{\infty} P(\overline{p}_n) \tag{1.1.1}$$

根据该式，我们不但可以计算出最终渲染得到辐射度值，还可以计算模型上的每个点处的辐射度值。

当计算图像中像素 j 的值时，我们想去沿着像素周边区域发出的光线进行积分（由于像素滤波，一个像素邻域的采样都会对当前像素有影响），积分值根据像素重建滤波器来计算贡献（我们以前一直忽略像素重建滤波器，而是直接取平均，如果要使用像素重建滤波器，那么一个样本对离着越近的像素贡献会越大）。我们暂时忽略景深（以便胶片平面上的每个点对应于相机的单个出射方向），我们可以将像素值写为（注意 p_0 表示相机上的点）：

$$I_j = \int_{A_{film}} \int_{\mathcal{S}^2} W_e^{(j)}(p_{film}, \omega) L_i(p_{film}, \omega) |\cos\theta| d\omega dA(p_{film}) \tag{1.1.2}$$

$$= \int_{A_{film}} \int_{A} W_e^{(j)}(p_0 \to p_1) L(p_1 \to p_0) G(p_0 \leftrightarrow p_1) dA(p_1) dA(p_0) \tag{1.1.3}$$

其中，$W_e^{(j)}(p_0 \to p_1)$ 表示：

$$W_e^{(j)}(p_0 \to p_1) = f_j(p_0) \delta\Big(t(p_0, \omega_{camera}(p_1)) - p_1\Big) \tag{1.1.4}$$

首先是上面的式子中，大家可能会好奇为什么1.1.2需要对角度积分，这跟景深和离焦模糊并没有什么关系，因为每个像素格子邻域都是一个范围，光线从这个范围穿过的角度也是有一个范围（当然，一个像素邻域的光线都比较集中在一个小范围里）。而公式1.1.3则表明了这一点，虽然

积分范围是全部面（p_1 所在的面），但只有恰好符合在 $\delta(t(p_0,\omega_{camera}(p_1))-p_1)$ 处的位置的滤波器值才会大于 0。

上面的式子可以继续写为：

$$\begin{aligned} I_j &= \int_{A_{film}}\int_A W_e^{(j)}(p_0\to p_1)L(p_1\to p_0)G(p_0\leftrightarrow p_1)dA(p_1)dA(p_0) \\ &= \sum_i \int_A\int_A W_e^{(j)}(p_0\to p_1)P(\overline{p}_i)G(p_0\leftrightarrow p_1)dA(p_1)dA(p_0) \end{aligned} \tag{1.1.5}$$

$P(\overline{p}_i)$ 表示反射 i 次到达相机的光路径。

$$I_j = \sum_i \int_A \dots \int_A W_e^{(j)}(p_0\to p_1)T(P(\overline{p}_i))L_e(p_{i+1}\to p_i)G(p_0\leftrightarrow p_1)dA(p_{i+1})\dots dA(p_0) \tag{1.1.6}$$

其中：

$$T(\overline{p}_n) = \prod_{i=1}^{n-1} f(p_{i+1}\to p_i\to p_{i-1})G(p_{i+1}\leftrightarrow p_i) \tag{1.1.7}$$

发射的辐射亮度 L_e（光源发射轮廓的度量）和加权函数 $W_e^{(j)}(p_0\to p_1)$（像素 j 的相机灵敏度轮廓度量）以很神奇的对称方式出现：这两个项都没有特别处理，因此我们可以推断出发射和相机测量的概念在数学上是可互换的。这种对称的含义很重要，它表示我们可以用两种不同的方式来思考渲染过程。第一种解释是，光可以从光源发射，在场景周围反弹，并到达传感器，我们在传感器中描述其对测量的贡献。或者，我们可以将传感器视为发射一个虚量，当它到达光源时产生一个测量值。

这个想法不仅仅是一个理论构想，它可以应用于实践。一个很好的例子是 Sen 等人的双重摄影工作，该工作表明，通过处理使用单独相机拍摄的输入照片，可以从视频投影仪的角度拍摄照片。这可以解释为将投影仪变成摄像机，同时使用原始摄像机作为“光源”照亮场景。

通过以这种方式简单地交换相机和光源的方法，我们可以创建一种称为粒子跟踪的方法，该方法跟踪来自光源的光线，以递归地估计到达表面的入射重要性。这本身不是一种特别有用的渲染技术，但它构成了其他方法（如双向路径跟踪和光子映射）的基本组成部分。

W_e 项描述的值称为场景中 p_0 和 p_1 之间光线的重要性。当测量方程用于计算像素测量值时，重要性通常部分或完全由 δ 函数描述（如上式）。除了图像形成之外，许多其他类型的测量可以通过适当构造的重要性函数来描述，这里描述的形式可以用于表示测量方程所描述的积分路径，这些积分也是必须估计计算的。

1.2 相机重要性计算与采样

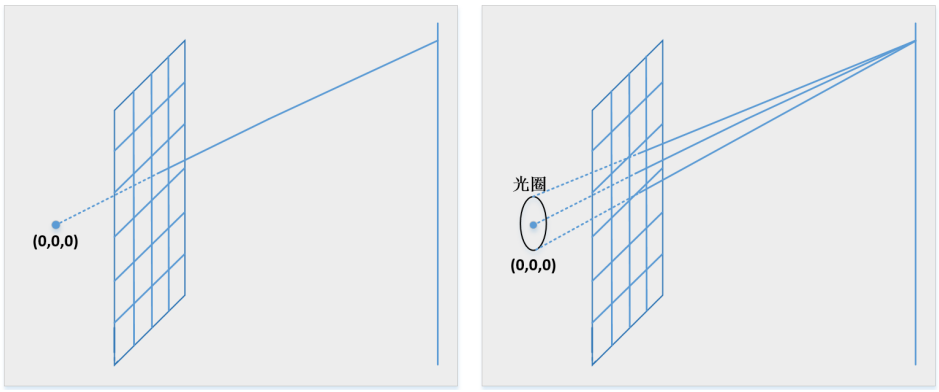
双向光传输算法需要计算场景中任意点的重要性函数值的能力；例如，这对于计算沿从光源开始的路径的点的重要性非常有用。尽管大家还不了解双向方法，但我先提前解释一下，这些内容大家可以以后回过头来看。在双向方法中，有可能光源生成的子路径会需要采样相机镜头上的顶点，此时不像相机开始构建子路径那样，初始权重 beta 为 1（关于 β 的含义，以前在路径追踪中有过定义，在《PBRT 系列 25-随机渐进式光子映射》的第一章中我也重新做了描述；而在双向方法中，β 对于光源路径还包括了光源的发射辐射度项）。本小节和后面两个小节（【相机采样的

概率密度与重要性】与【对相机进行采样】）所介绍的采样和概率计算全都是跟这种情况有关，即相机子路径只有一个顶点，光源子路径有一个以上的顶点数（注意后面会描述，光源子路径和相机子路径不能都只有一个顶点）。其他时候我们是不需要 We() 函数（该函数被 Sample_Wi() 函数调用）的，但是在生成相机路径顶点时，我们需要计算相机射线的方向和位置概率密度，需要调用 Camera::Pdf_We() 函数。

所以说，下面的内容跟双向路径追踪有很大关系，读者现在就接触可能会有些抽象，因此个人建议可以先简单了解一下或者直接跳过，等后面遇到了上述描述的情况以后再回来重新看这几个小节。在倒数第二章节中我会描述一下为什么要采样相机以及一些相对来说比较晦涩的内容。

1.2.1 相机知识回顾

相机与透镜采样（离焦模糊效果）中，我们的透视投影相机相当于设置一个焦距和聚焦平面 (Plane of focus)，我们需要在镜头的孔径 (Aperture) 处采样，得到一条射向与聚焦平面相交点的光线，我们得到的相机初始 ray 的位置是在孔径 (Aperture) 上的，场景中对于在聚焦平面所在距离的点并不会模糊。当镜头孔径大于 0 时，采样镜头的方式如下：



相机先根据聚焦平面的距离计算出沿着当前相机射线的方向前进多少距离才能到聚焦平面，然后获得当前射线的焦点位置作为 pFocus；然后根据光圈大小，以 $(0,0,0)$ 为圆心，采样圆盘，得到新的射线出发点 o，o 到 pFocus 的方向就是射线方向。

相机空间中，相机原点在 $(0,0,0)$，透视投影相机近平面在程序中一般设为了 $1e-2f$，远平面设为 1000（通过 screenToCamera，将屏幕空间中 z 值为 0 的位置映射到相机空间中深度为 $1e-2f$ 的位置；将 z 值为 1 的位置映射到相机空间中深度为 1000 的位置）：

```
1 // CameraToScreen:
2 CameraToScreen = Perspective(fov, 1e-2f, 1000.f);
3 screenToCamera = Inverse(CameraToScreen);
```

由此，RasterToCamera 将光栅空间上的 $(RasterX, RasterY, 0)$ 映射到相机空间的 $(x, y, 1e-2f)$。

1.2.2 计算相机光线响应

Camera::We() 方法输入原点为 p 且方向为 ω 的光线，并计算从相机 p 上的点在方向 ω 上发射的重要性。该函数给定 ray 的方向，我们就还需要一些变换，转换到光栅空间以及相机空间的位置。

```
1 virtual Spectrum We(const Ray &ray, Point2f *pRaster2 = nullptr) const
    ;
```

We() 函数的传入参数中，如果 pRaster2 不是 nullptr 指针，那么 pRaster2 参数就赋予光线在胶片光栅上的位置（分辨率就是胶片分辨率，但是可以表示为更准确的分数形式，不必为整数）。源码中只给透视相机实现了该函数，其他相机是没有实现 We() 函数的。透视投影相机 We() 函数分为下面几个步骤：

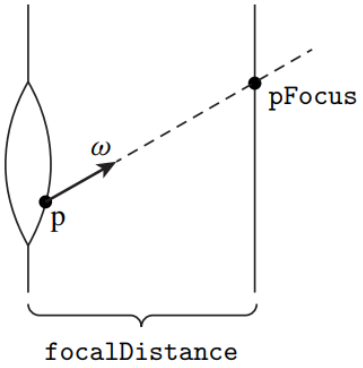
```
1  virtual Spectrum We(const Ray &ray, Point2f *pRaster2 = nullptr) const
       {
2    // (1)插值相机矩阵，检查是否 omega 是朝前的
3    ......
4    // (2)把光线映射到光栅网格上
5    ......
6    // (3)如果需要，返回光栅位置
7    ......
8    // (4)对于超出界限的点返回零重要性
9    ......
10   // (5)计算投影相机的镜头面积
11   ......
12   // (6)返回成像平面点的重要性
13   ......
14 }
```

(1)：在给定某时间下（相机可能是运动中的）的 camera-to-world 变换，该方法通过将摄像机空间观察方向 $(0,0,1)$ 变换到世界空间并计算它们之间角度的余弦来检查方向 ω 是否指向相机所面对的同一半球（如果这些方向角度相距超过 90 度，则相机将不会从其 GenerateRay() 方法返回该方向上的光线，因此可以立即返回重要性值 0）。

(2)(3)(4)：接下来，一个稍微复杂的测试将检查这个光线是否是从胶片区域开始的光线。将 pFocus 点变换到对应的光栅上的点 pRaster2，如果该点在胶片范围之外，则点 p 在相机的观察体之外，那么再次返回零重要性值。给定光栅空间点，很容易检查它是否在图像范围内。

对于具有有限孔径 (aperture) 的相机（涉及离焦模糊效果），我们在镜头上有一个点 p 及其方向（如下图）。我们还不知道这条光线对应于胶片上的哪个点（该点不一定在像素中心），但我们知道当采样某条射线 $\mathbf{r}_1$ 并进行离焦模糊时，所以依据离焦模糊采样到的相机射线都在平面 z=focalDistance 处聚焦。我们想找到这条射线 $\mathbf{r}_1$ 所在的像素，可以计算射线与焦点平面的交点，然后用透视投影矩阵变换该点，就可以得到胶片上的对应点。

对于针孔孔径，不再有所谓的聚焦平面，所以我们计算与任意设置为 $z = 1$ 的平面的交点，以在执行投影之前沿着离开相机的光线获得一个点。



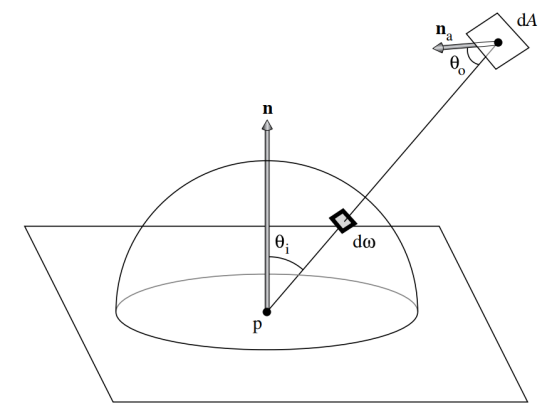
代码如下：

```
Point3f pFocus = ray((lensRadius > 0 ? focalDistance : 1) / cosTheta);
    Point3f pRaster = Inverse(RasterToCamera)(Inverse(c2w)(pFocus));
```

我们本章后面的小节介绍重要性，该步骤由 (5) 和 (6) 完成。但在介绍重要性之前，我们会先介绍镜头上的点均匀采样图像平面点时所对应的立体角相关的概率密度。

1.2.3 相机采样某方向的概率密度

Knowledge 1.1 (立体角积分和面积分) 我们回忆一下关于面积分和立体角积分的知识，后面在概率密度转换时是有用的。



对立体角的积分可以转换为对面积的积分（θ 表示 dA 表面法向量与到 p 点的向量夹角，也就是角 θ_o；r 是从 p 到 dA 的距离）：

$$d\omega = \frac{dA\cos\theta}{r^2} \tag{1.2.1}$$

于是，irradiance 可以写为：

$$E(p,n) = \int_A L\cos\theta_i \frac{\cos\theta_o dA}{r^2} \tag{1.2.2}$$

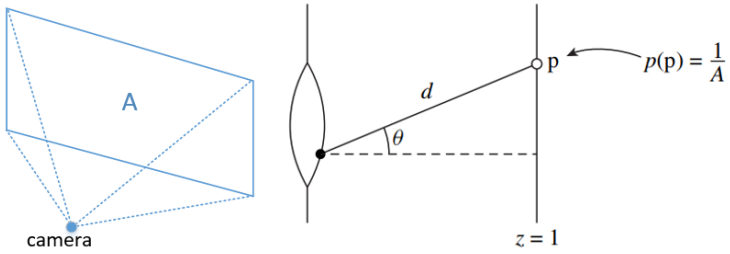
我们假设相机采样不依赖于某像素范围，而是把相机初始射线的方向看做随机生成的（这与双向方法有关），当我们有某个相机镜头上的顶点 $\mathbf{z}_0$ 时，我们想计算出它经过随机采样方向后，

初始射线能够与场景中某个点 $\mathbf{z}_1$ 相交的概率密度。

pbrt 中的透视相机是理想传感器，因为它可以在胶片区域上生成均匀分布的样本，我们现在将使用这个事实来导出相应的定向采样分布。首先，我们将定义一个相机空间图像矩形，所有相机发出的光线都要穿过该矩形在深度为 $z = 1$ 上的一个点。下面的代码段是透视相机构造函数的一部分，它使用 RasterToCamera 变换，并除以 z 坐标来计算矩形的边界，从而得到矩形的面积 A：

```
Point2i res = film->fullResolution;
Point3f pMin = RasterToCamera(Point3f(0, 0, 0));
Point3f pMax = RasterToCamera(Point3f(res.x, res.y, 0));
pMin /= pMin.z;
pMax /= pMax.z;
A = std::abs((pMax.x - pMin.x) * (pMax.y - pMin.y));
```

A 就是在 $z = 1$ 处相机成像面的面积：



我们需要相机均匀地在 A 平面上产生样本，因此，图像平面上点的面积度量概率密度为 $1/A$。现在考虑在镜头上的点的 PDF（假设已知镜头上有一个点，该点位置确定），根据面和立体角的转换，当 ω 在视锥体内部，则 $p(\omega) = \frac{d^2}{A\cos\theta}$，由于平面在 $z = 1$ 处，因此 $d = \frac{1}{\cos\theta}$，所以 $p(\omega) = \frac{1}{A\cos^3\theta}$。注意，镜头平面和图像平面是平行的，所以 θ 就是图像平面法向量与采样方向的夹角。从镜头上的某个点去采样图像平面上的点时，这个采样概率密度的计算和直接采样面光源的概率密度计算是完全一样的。

在 PerspectiveCamera::Pdf_We() 函数中就是这么计算的：

```
void PerspectiveCamera::Pdf_We(const Ray &ray, Float *pdfPos, Float *
    pdfDir) const {
  // 一些类似于We()函数的操作，变换到相机空间
  ......
  // 计算概率密度
  Float lensArea = lensRadius != 0 ? (Pi * lensRadius * lensRadius) :
      1;
  *pdfPos = 1 / lensArea;
  *pdfDir = 1 / (A * cosTheta * cosTheta * cosTheta);
}
```

注意这里镜头面积，如果相机有景深效果，则镜头面积设为 πr^2；否则，就把镜头面积设为 1（这样概率密度就是 1 了）。这里的 pdfPos 计算的是镜头上的采样点的概率密度值。

1.2.4 相机在某方向的光线重要性

下面的积分式中有 W_e 项，该项与相机有关，表示射入到相机的光线的重要性：

$$I_j = \sum_i \int_A \dots \int_A W_e^{(j)}(p_0 \to p_1) T(P(\overline{p}_i)) L_e(p_{i+1} \to p_i) G(p_0 \leftrightarrow p_1) dA(p_{i+1}) \dots dA(p_0) \tag{1.2.3}$$

其中 W_e 表示为：

$$W_e^{(j)}(p_0 \to p_1) = f_j(p_0)\delta\Big(t(p_0, \omega_{camera}(p_1)) - p_1\Big) \tag{1.2.4}$$

为什么射入到相机的光线不是同等重要的呢？我们按照最简单的思路，就是射入到相机一个像素的光（或者说相机采样得到的辐射度）取平均值，就得到了该像素值。所以按理来说，“重要性”定义为 1 不就可以了吗？但要注意的是，在求一个像素内所有光线的贡献时，这是一个积分的过程，保证被积分的 W_e 项积分为 1 就意味着传感器对于所有光线在这个区间内的响应为 1。注意由于采样到的方向不依赖于像素（后面会介绍的 PerspectiveCamera::Sample_Wi() 函数），所以计算的 W_e 值也不会依赖于像素，而是在整个图像平面构成的区间积分。

重要性函数不必遵守任何规范化 (normalization) 约束（就像面光源发出的辐射度那样，这是因为传感器响应不但与输入有关，还与自身属性有关，比如传感电压、信号放大器等）。但是，PBRT 中将 PerspectiveCamera 的重要性函数定义为光线空间上的单位化 PDF，这既便于后续操作，也与 PerspectiveCamera::GenerateRay() 函数返回的权重值 1 一致。

投影相机在路径追踪时的 β 值为 1，意味着一个像素内的估计的入射辐射度值就是传感器响应值，且不与角度有关。在估计时，令 $W_e^{(j)}$ 在区间内积分为 1 也与这里设的 $\beta = 1$ 是为了保持一致的。虽然每条光线入射到像素内的权重都是一样的，但双向方法对相机射线进行采样时，就不能设每条射线的概率密度都相同了，因为不同的立体角下的相机射线采样密度是不同的。

重要性函数在 $W_e(\omega) > 0$ 的区间内变化平滑，定义此变化是为了消除实际针孔相机可能具有的渐晕效果（由于胶片或者传感器边缘区域较中心接受到的光更少所导致图像边缘偏暗的现象），并确保像素以辐射度为单位记录值（这也是 PerspectiveCamera::GenerateRay() 返回权重值为 1 的另一个原因）。

我们创建一个归一化重要性函数 $W_e(p, \omega)$（注意此时的 p 点在镜头上），该函数定义在 $A_{lens} \times \mathcal{S}^2$（ray 空间）中，该函数必须满足光线空间归一化标准（θ 是光线方向与 z 轴正方向的夹角）：

$$\int_{A_{lens}} \int_{\mathcal{S}^2} W_e(p, \omega) |\cos\theta| d\omega dA(p) = 1 \tag{1.2.5}$$

我们可以看出（因为 $p(\omega)$ 与 $A(p)$ 无关，所以很容易总结出来下式；注意 r 是镜头的半径）：

$$W_e(p, \omega) = \frac{p(\omega)}{\pi r^2 \cos\theta} = \begin{cases} \frac{1}{A\pi r^2 \cos^4\theta} & \omega \ within \ frustum \\ 0 & otherwise \end{cases} \tag{1.2.6}$$

之所以要用 $p(\omega)$ 除以 $\pi r^2 \cos\theta$，这里面的 $\cos\theta$ 项是因为抵消掉了积分式中的 $\cos\theta$ 项。由此，We 在整个相机成像面和镜头所构成的区间中的积分结果为 1，这就是相当于规范化的 PDF。

1.2.5 对相机进行采样

假如我们在场景中有一个顶点，我们希望从相机镜头上采样一个点，并计算采样到这个点的概率密度（这与双向方法有关）。当我们构建的路径中相机子路径只有一个顶点时，就会从相机镜头上重新采样一个新的顶点，然后再计算路径贡献。Sample_Wi() 函数对相机镜头上的一个点进行采样，并计算重要性到达场景中给定参考位置的入射方向，注意该方法返回的 pdf 值是相对于场景中参考点的立体角定义的。

PerspectiveCamera::Sample_Wi() 实现均匀采样镜头上的一个点，计算入射到参考点处方向的重要性（调用 PerspectiveCamera::We() 函数）。在代码中，lensIntr 是 Interaction 对象，其交点位置是 $(pLens.x, pLens.y)$，交点法向量是相机初始朝向 $(0,0,1)$ 变换到世界空间的值，lensArea 是镜头面积。概率密度计算就是：

$$pdf = \frac{\|lensIntr.p - ref.p\|^2}{|(lensIntr.n \cdot *wi)|lensArea} \tag{1.2.7}$$

这和对光源采样的 PDF 计算是完全一样的。

1.3 对光源进行采样

对于双向光传输算法，还需要添加一种光采样方法 Sample_Le()，该方法从离开光的光线分布中采样光线，以 *ray 返回光线，以 *nLight 返回光源上的点处的表面法线（实际上，类似于 Camera::GenerateRay()。u1 和 u2 都是 Point2 类型的随机数，两个随机数可用于采样光线的原点，两个随机数可用于采样其方向。

并非所有光源采样的实现都需要所有这些值。例如，离开点光源的所有光线的原点都相同。该方法返回两个 PDF 值：光线原点相对于光源表面积的概率密度及其方向相对于立体角的概率密度。光线采样的联合概率是这两种概率的乘积。例如对于面光源，采样到光源点的概率与面光源表面积大小有关，而采样到某个方向的概率就是半球内根据 cos 值采样的均匀概率。因为书 [1] 的 16.1.2 章节讲解的非常全面，而且具体到每种光源类型也非常简单，所以这里不再展开介绍。后面会提一下无限面光源的采样。

1.4 非对称散射

非对称散射的问题要想理解清楚，一定需要先理解双向路径追踪方法以及折射材质的原理，但这里我们会先简要先提一下什么是非对称散射，后面更详细的内容可以参考《模拟光传输的鲁棒的蒙特卡洛方法（Eric Veach）全文解读》中的《双向路径追踪架构总结》部分关于非对称散射的描述。

材料和几何体的某些方面可能导致光传输模拟中的非对称行为，其中入射辐射度和重要性在一个点上以不同方式散射。如果不考虑这些差异，则在渲染同一输入场景时，基于辐射度和重要性传输的渲染算法将产生不同且不一致的结果。结合辐射度和重要性传输的双向技术尤其受到影响，因为其设计基本上基于对称原则。非对称情况可以参考 [2] 论文中的描述。

回忆路径吞吐量项 $T(\overline{p}_i)$，定义为（顶点都是按照路径顺序来的）：

$$T(\overline{p}_n) = \prod_{i=1}^{n-1} f(p_{i+1} \to p_i \to p_{i-1}) G(p_{i+1} \leftrightarrow p_i) \tag{1.4.1}$$

从光源出发找携带重要性的路径来追踪光线来估计光处的入射重要性，意味着顶点将会由上述顺序的逆序来产生。因此，除非采取特殊预防措施，否则 BSDFs 的输出方向参数与入射方向将（错误地）反转。因此，我们在顶点 p_i 定义了伴随 BSDF f (adjoint BSDF)，其唯一作用是使用交换的参数估计原始 BSDF（就是把输出方向变为了输入方向）：

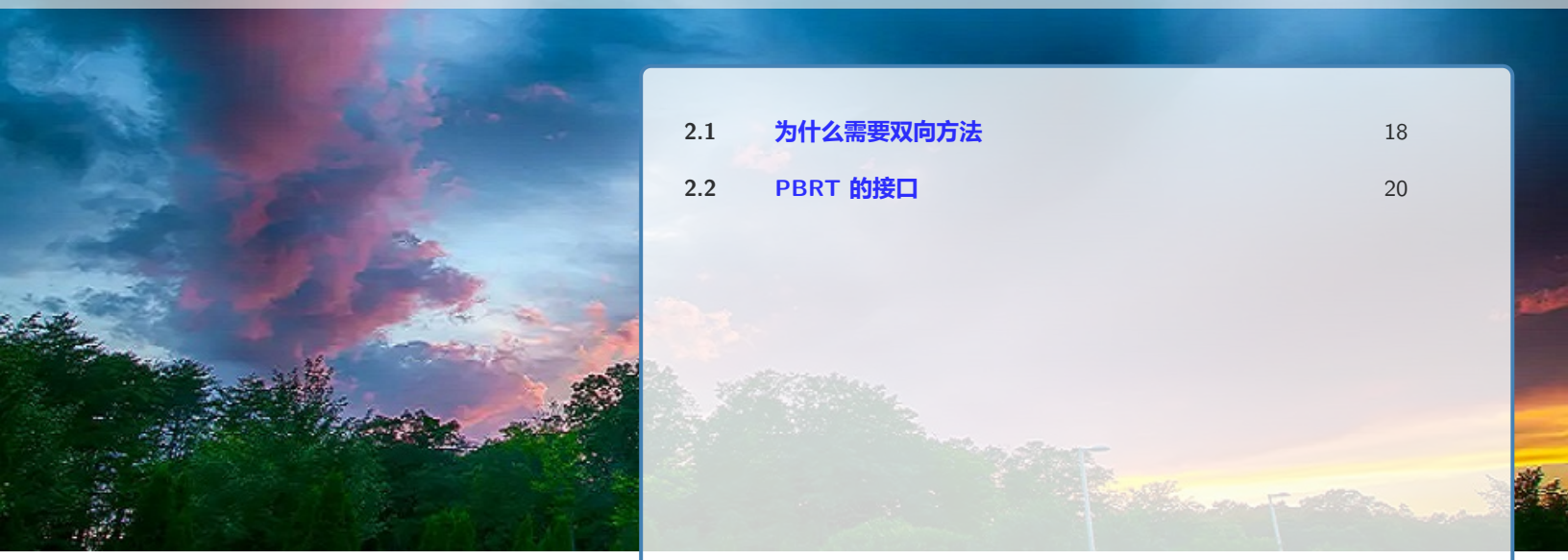
$$f^*(p, \omega_o, \omega_i) = f(p, \omega_i, \omega_o) \tag{1.4.2}$$

然后，所有的基于重要性传输（携带重要性的路径，构建的是光路径）的采样步骤将会使用 BSDF 的伴随形式而不是以前的版本。PBRT 中的大多数 BSDF 都是对称的，所以 f 和 f^* 之间没有什么区别。但是，与着色法线和折射到具有不同折射率的介质中的光相关的某些情况需要额外注意。

传输模式 TransportMode 是一个枚举对象，用于告诉非对称 BSDF 在重要性传输还是辐射度传输两种模式下自由切换。这里的传输模式个人感觉和论文 [1] 中表示的不太一样。这里的 TransportMode::Radiance 模式更像是在“估计辐射度”，而不是要“传输辐射度”。我们按照 PBRT 中的描述（后面每次遇到传输模式我都会强调一下），即相机路径是辐射度传输（BSDF 值也是光入射方向到出射方向的计算），它传递重要性；光源路径是重要性传输（BSDF 值也是光入射方向到出射方向的计算）。

代码中会根据 TransportMode 来区分相机子路径和光源子路径，进而用于对折射和着色法向量引起的非对称现象进行控制。折射是直接在构建路径时处理的；对于着色法向量，CorrectShadingNormal() 函数表示使用正确的着色法向量的矫正因子。

2. 双向路径追踪初步介绍

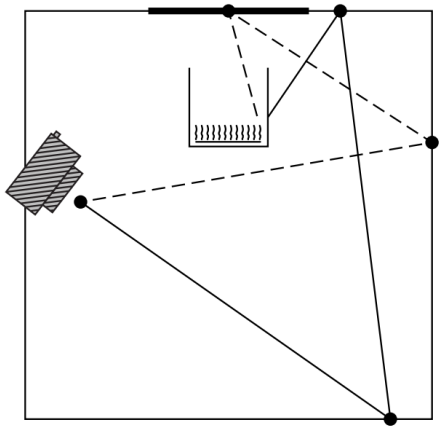


本章讲解双向路径追踪算法的基本内容和基础知识。

2.1 为什么需要双向方法

在 Veach 的论文 [2] 中有双向路径追踪很详细的介绍，我们也在《模拟光传输的鲁棒的蒙特卡洛方法（Eric Veach）全文解读》系列中对该论文进行了解释和叙述。

当光源的位置比较苛刻时，只从相机发射采样光线效率便不会很高。例如，光源照亮天花板上的一个小区域，使得房间的其余部分仅由该区域反射的间接照明照亮：



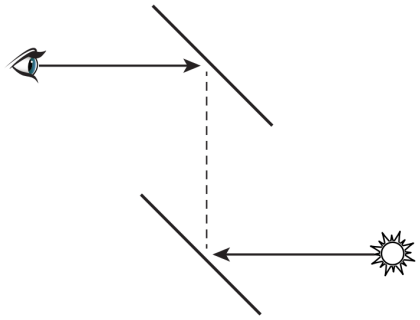
如果我们只跟踪从相机开始的路径，我们很难在天花板上的照明区域中采样路径顶点（阴影光线也都被光源四周的壁遮挡了）。大多数路径将没有贡献，而其中一些恰好击中天花板上的小区域的路径将有很大贡献，所得图像将具有高方差。

通过构造从一端的相机和另一端的灯光开始并在中间连接可见性光线的路径，可以更有效地处理此类困难的场景。由此产生的双向路径跟踪算法（以下称为 BDPT）是标准的路径追踪算法的推广，可以更有效。与随机渐进光子映射（后面再介绍）相比，BDPT 是无偏的，且不会模糊场景照明。

BDPT 首先会构建一个从相机出发的子路径，假设得到 t 个顶点的路径 $p_0, p_1, ..., p_{t-1}$。之后再从光源出发，构建一个 s 个顶点的光源路径 $q_0, q_1, ..., q_{s-1}$。给定两条子路径，一个携带了光的路径就可以通过连接顶点来得到：

$$\overline{p} = q_0, ..., q_{s'-1}, p_{t'-1}, ..., p_0 \tag{2.1.1}$$

这里的 $s' \leq s$ 和 $t' \leq t$。如果在 $q_{s'}$ 和 $p_{t'}$ 之间没有遮挡，那么就可以计算连接顶点处的 BSDF 来计算路径贡献：



更一般地，这些子路径都可以使用路径空间积分理论来组合起来，过程就是：

```
对每个像素：
  像素颜色 = trace(像素位置pos)
trace(pos)
  从eye位置穿过该像素发射Ray,不断反射，得到Ray路径RayPath
  从光源位置发射Light，不断反射，得到光路径LightPath
  combine(RayPath,LightPath)
combine(RayPath,LightPath)
  color = 0
  遍历LightPath的每个点LP
      遍历RayPath的每个点RP
          如果他们之间的可见性==1
              计算LP–RP的权重w
              color += w * 光照量
  return color
```

从表面上看，这种方法与光子映射的两个阶段有些相似；一个关键区别是 BDPT 计算无偏估计，而不计算密度估计。

在如何处理给定光路可以以多种不同方式构建这一方面也存在显著差异。到目前为止，BDPT 算法相比于以前的算法一共有三种改进，以提高其实际性能。前两个类似于路径跟踪的改进，第三个是方差减少技术。

- 子路径可以被重用，也就是说，对于 t 个相机顶点路径和 s 个光源顶点路径，我们可以构造大量独一无二的子路径，这些路径的长度从 2 个顶点到 $s+t$ 个顶点各有不同。

- 第二个优化是，当子路径之一只有单个顶点时，就不是直接连接，而是进行采样（比如当

$s = 1$ 时，相当于直接采样光得到一个新的顶点；当 $t = 1$ 时，会采样相机得到一个新的顶点）。

- 第三种优化对用于生成给定长度的路径的各种策略进行加权（比直接对构建相同长度路径的所有策略得到的结果进行平均更好）。BDPT 连接子路径的方法意味着包含 n 个散射事件的路径可以以 $n+3$ 种不同方式生成。我们可以预期，某些策略对于生成某些类型的路径是一个不错的选择，而对于其他类型的路径则非常糟糕。多重要性抽样可用于将一组连接策略组合成单个估计器，该估计器在对于每个策略最佳的地方来使用每个策略。多重要性抽样 MIS 的应用对于 BDPT 的效率至关重要。

BDPT 的连接策略之一是将光源子路径顶点直接连接到摄影机：这些路径几乎总是创建一个贡献，其光栅坐标与当前渲染的像素不同（与连线和光栅交点位置有关），这跟 SamplerIntegrator 是不同的。因此，BDPTIntegrator 源自更一般的 Integrator 接口。

2.2 PBRT 的接口

BDPTIntegrator 需要一个相机对象来发送相机光线路径。

所有子路径创建和连接步骤都在 BDPTIntegrator::Render() 中的像素上的并行循环中执行，这个结构与 SamplerIntegrator::Render() 函数在结构上是非常相似的。我们忽略这些相似的内容，而是直接来看负责生成和连接一个像素样本的子路径的代码片段：

```
// 使用BDPT来产生单个样本：
Point2f pFilm = (Point2f)pPixel + tileSampler->Get2D();
// 追踪相机的子路径和光的子路径
......
// 计算全部的BDPT连接策略
......
```

我们首先为两个子路径的顶点分配两个数组。除了表面上的顶点 Vertex 外，顶点 Vertex 还可以表示参与介质中散射事件的顶点、光源上的顶点或相机镜头上的顶点。

对于每个子路径，必须多分配一个顶点来在灯光或相机上存储起始顶点（顶点数比最大路径长度多 1）。相机子路径需要再多一个顶点，多出来的顶点可以用来相机路径随机地与光源相交，此策略有不少好处，比如对于渲染仅从镜面反射的面光源非常重要（根据我从源码实现来看，其实意思就是让相机路径多采样一个顶点，免得镜面物体比较多的时候无法追踪到镜面反射后的光源）。（为什么不需要光源子路径多一个顶点，是因为允许光路径随机与相机镜头相交的相应策略在实践中是没什么用的。）

```
Vertex *cameraVertices = arena.Alloc<Vertex>(maxDepth + 2);
Vertex *lightVertices = arena.Alloc<Vertex>(maxDepth + 1);
```

GenerateCameraSubpath() 和 GenerateLightSubpath() 函数可以产生两条子路径，

```
// 使用BDPT来生成单个样本
```

```
Point2f pFilm = (Point2f)pPixel + tileSampler->Get2D();
// 追踪相机子路径
Vertex *cameraVertices = arena.Alloc<Vertex>(maxDepth + 2);
Vertex *lightVertices = arena.Alloc<Vertex>(maxDepth + 1);
int nCamera = GenerateCameraSubpath(scene, *tileSampler, arena,
    maxDepth + 2, *camera, pFilm, cameraVertices);
// 光分布
const Distribution1D *lightDistr = lightDistribution->Lookup(
    cameraVertices[0].p());
// 追踪光线子路径
int nLight = GenerateLightSubpath(scene, *tileSampler, arena, maxDepth
    + 1, cameraVertices[0].time(), *lightDistr, lightToIndex,
    lightVertices);
```

在子路径都生成以后，一个嵌套 for 循环会迭代所有两个子路径的顶点对，并尝试去连接它们。在这些循环中，在这些循环中，s（光源顶点数量）和 t（相机顶点数量）对应于从对应的子路径使用的顶点的数量，为 0 意味着不使用来自相应子路径的散射事件。我们只支持 $s = 0$ 的情况（$t \neq 0$ 也就是说光线必须要先从相机上采样镜头上的一个点），这是由于光线在多次散射中与相机相交的概率实在是小，所以会带来巨大的方差。由于不支持与 $t = 0$ 的相机相交的情况，因此相机子路径上的循环从 $t = 1$ 开始。

路径长度（子路径）为 1 对应于将相机镜头或光源上的点连接到另一个子路径。对于光源端点，这与 light::Sample_Li() 提供的标准的光采样方法相同；我们的实现使用此现有功能。对于相机端点，我们将依赖对称模拟 Camera::Sample_Wi()。由于不能同时使用 Camera::Sample_Wi() 和 Light::Sample_Li()，因此我们跳过 $s = t = 1$ 的情况。

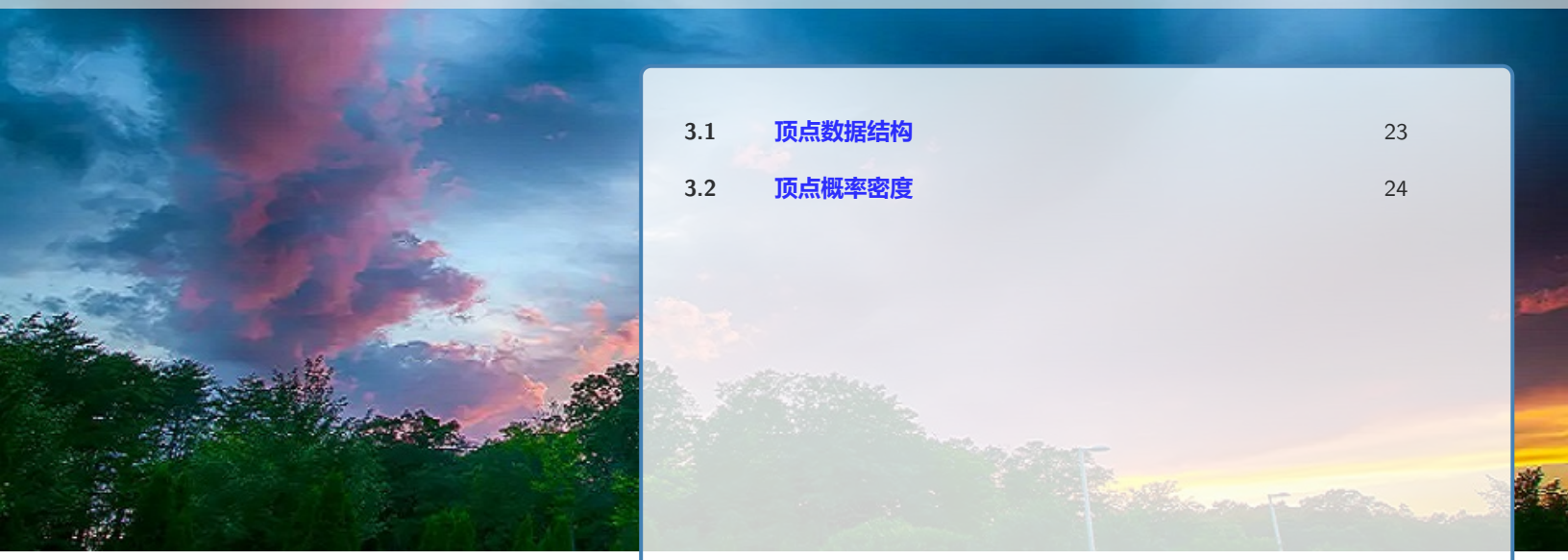
```
for (int t = 1; t <= nCamera; ++t) {
  for (int s = 0; s <= nLight; ++s) {
    int depth = t + s - 2;
    if ((s == 1 && t == 1) || depth < 0 || depth > maxDepth)
      continue;
    // Execute the $(s, t)$ connection strategy and update _L_
    ......
  }
}
```

在上面的代码中，计算连接策略需要用到 ConnectBDPT() 函数。

```
Spectrum Lpath = ConnectBDPT(
  scene, lightVertices, cameraVertices, s, t,
  *lightDistr, lightToIndex, *camera, *tileSampler,
  &pFilmNew, &misWeight);
```

```
5 // 此时Lpath值代表当前采样像素的贡献，MIS加权的子路径贡献直接累加到像
    素值上
6 if (t != 1) L += Lpath;
7 // 此时Lpath值不一定是当前像素，需要根据pFilmNew的位置来锁定目标像素
8 else film->AddSplat(pFilmNew, Lpath);
```

3. 双向路径追踪的顶点类



本章讲解双向路径追踪算法的顶点类 *Vertex* 的功能。

3.1 顶点数据结构

定义 Vertex 类型，它可以表示任何类型的路径顶点（包括参与介质上的散射点），处理顶点的函数封装在顶点类里，实际操作都是让顶点的自身属性去完成，而不是在 BDPT 的实际代码中用大量的 if-else 语句来根据当前顶点类型选择计算方式。

```
enum class VertexType { Camera, Light, Surface, Medium };
```

Vertex 类中有 beta 值，该值包含到当前顶点为止生成的路径中顶点的 BSDF 或相位函数值、穿透率和余弦项的乘积，除以它们各自的采样 PDF（双向方法是支持体渲染的，这里的 β 和体路径追踪器的 β 含义是一样的）。对于光源子路径，还包括发射的辐射度除以发射位置和方向的概率密度。对于相机子路径，辐射度替换为重要性。

Vertex 类中包含了交点类型，但是由于交点有三种：表面交点、光源交点以及终点交点（相机镜头上或者光源上），但因为每个顶点只属于其中一个，所以用 union 来表示：

```
union {
  EndpointInteraction ei;
  MediumInteraction mi;
  SurfaceInteraction si;
};
```

EndpointInteraction 是只由 BDPT 使用的类，它里面有一个指针，指向光源或者相机。该类的成员函数也都是给其基类 Interaction 赋值。

Vertex 类包含了一大堆构造顶点的函数（第三节再详细介绍）：

```
CreateCamera
CreateLight
CreateMedium
CreateSurface
```

以及 GetInteraction 函数用来获得交点对象。

返回顶点位置、时间、几何法向量以及着色法向量值的函数：

```
Point3f &p();
Float time();
Normal3f &ng();
Normal3f &ns();
```

一个 bool 类型的 delta 变量，用来记录当前点的采样是否是 delta 类型分布（也就是当前点是否是完美镜面点）。

IsOnSurface 成员函数用来检测顶点是否是在表面上的点，根据 Vertex::ng() 是否是 0 向量点就能够判断得到。

Vertex::f() 函数只需要处理表面顶点和介质顶点，因为 BDPT 只会在表面顶点和介质顶点调用它，注意它的传入参数只可能是子路径中的下一个顶点。

Vertex::IsConnectible() 函数表示是否当前顶点可用于连接策略。比如当顶点是镜面交点，它就无法与其他子路径连接。同时，假设参与介质顶点和相机顶点永远是可连接的（当然对于正交投影相机不满足该条件，不过还好正交投影相机在 PBRT 程序实现中并不支持 BDPT 相关功能）。

Vertex::IsLight() 用来测试顶点是否可以被解释为一个光源。例如，当表面顶点本身是面光源时，该顶点可以根据 BDPT 的连接策略承担不同的角色：它可以被重新解释为光源并用作路径端点，或者它可以用作普通散射事件以生成更长的路径。

Vertex::IsDeltaLight() 用来判断光源是否是 Delta 类型的光源。

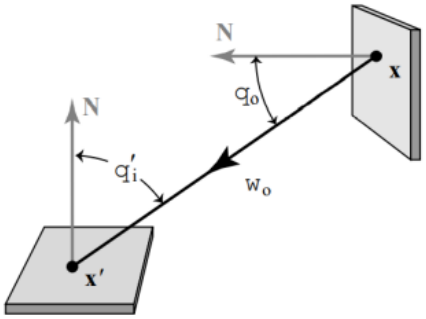
Vertex::IsInfiniteLight() 用来判断光源是否是无限面光源。该顶点可以有两种产生方式：（1）是从无限面光源上采样一个顶点；（2）是相机光线没有与场景相交，则与环境全景图相交，这种情况下 ei.light 就是一个 nullptr 指针。

Vertex::Le() 用来计算从相交的光源向另一个顶点发出的辐射。

3.2 顶点概率密度

路径的概率密度可以表示为其各个顶点的概率密度的乘积。

在 ConvertDensity 函数把立体角表示的概率密度转换为面积表示的概率密度。该函数的参数中，next 表示下个顶点，pdf 表示传入的与立体角相关的概率密度。有人可能好奇，我们对立体角采样又不是对某个面进行采样（类似面光源），那怎么能从立体角相关的概率密度转换到面积相关的概率密度呢？这是因为单位立体角对应于单位球面上面积为 1 的弧面对应的角度范围，而当 next 采样点越远，则单位面积对应的立体角就会越小，这个转换关系与距离的平方成反比（在《光传输的路径积分与符号表示》也有解释）：



注意在 ConvertDensity 函数中，当 next 点是参与介质散射点时，概率密度不需要乘以 cos 项。另外，对于无限环境光而言需要特殊处理，我们后面再解释。

每个顶点都有两个概率密度：pdfFwd 存储前向密度（由路径采样算法生成的当前顶点的每单位面积的概率）；pdfRev 是假设如果光传输方向相反的顶点概率密度，也就是说，相机路径中用辐射度传输代替重要性传输（或者在光源路径中用重要性传输代替辐射度传输），反向权重对于计算 MIS 权重来说是非常重要的。

Vertex::Pdf() 方法返回的是给定顶点的每单位面积的概率密度。给定一个上一个顶点 prev，它计算了离开当前顶点 (*this) 采样下一个顶点 next 的概率密度。prev 参数可能等于 nullptr。对于处理光源则需要用到 PdfLight() 函数，我们后面再说。

```
Float Pdf(const Scene &scene, const Vertex *prev,
  const Vertex &next) const {
    // (1)计算当前顶点到上个顶点prev以及到下一个顶点next的方向
    ......
    // (2)根据顶点类型计算方向密度
    ......
    // (3)通过ConvertDensity转化为每单位面积的密度
    ......
}
```

wn：当前点到 next 的方向；wp：当前点到 prev 的方向。第 (2) 步中注意 unused 变量表示“用不到的量”（如果当前点是相机镜头上的点，那么）：

```
  Float pdf = 0, unused;
  if (type == VertexType::Camera)
    ei.camera->Pdf_We(ei.SpawnRay(wn), &unused, &pdf);
```

计算其他概率密度需要用到 wp 和 wn，这是因为根据 BSDF 采样时，出射方向的采样会与入射方向有关：

```
  else if (type == VertexType::Surface)
    pdf = si.bsdf->Pdf(wp, wn);
  else if (type == VertexType::Medium)
    pdf = mi.phase->p(wp, wn);
```

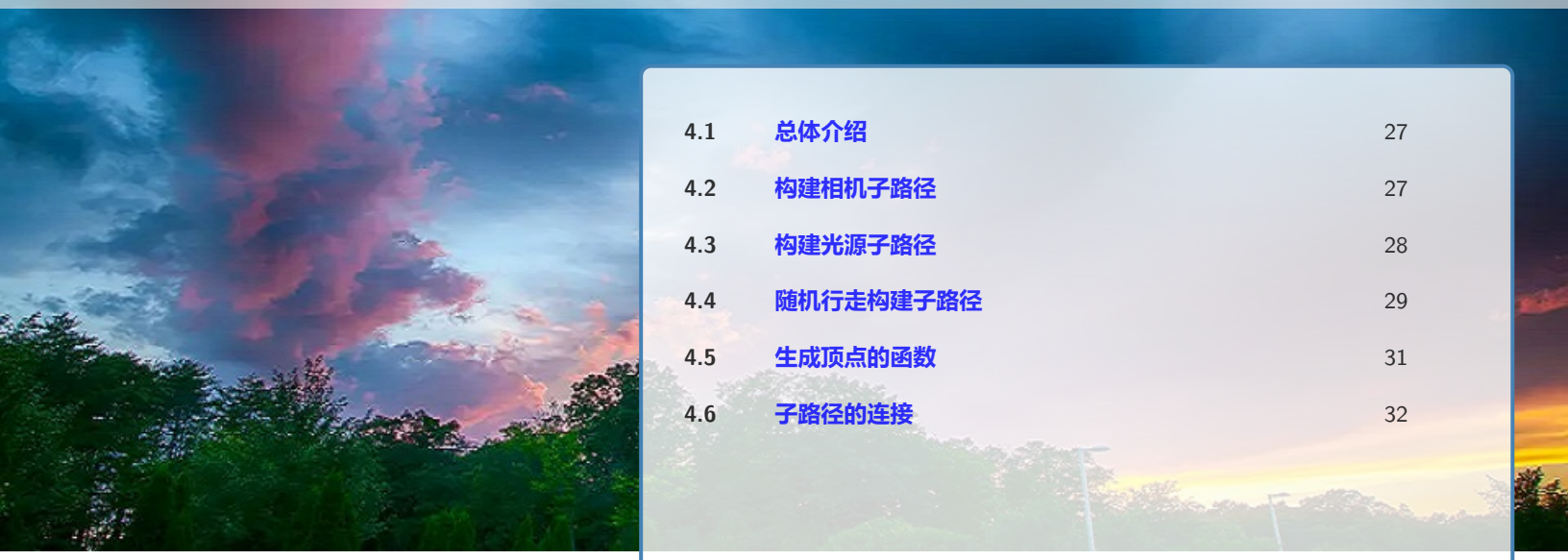
对于光源的采样，有两种方式，一是采样光源点（通过 Light::Sample_Le()）；二是根据光线追踪来与发光表面求交。为了根据 MIS 来计算权重，需要求其 PDF。求光源采样的 PDF 方法为 Vertex::PdfLight 函数（我们忽略无限面光源的情况）：

```
Float PdfLight(const Scene &scene, const Vertex &v) const {
  // 计算到v点的方向
  Vector3f w = v.p() - p();
  Float invDist2 = 1 / w.LengthSquared();
  w *= std::sqrt(invDist2);
  // 计算概率密度，如果当前Vertex类型是光源，说明可以使用端点描述的光源对象，否则就获取表面上的光源（一定是面光源）
  Float pdf;
  const Light *light = type == VertexType::Light ? ei.light : si.primitive->GetAreaLight();
  Float pdfPos, pdfDir;
  light->Pdf_Le(Ray(p(), w, Infinity, time()), ng(), &pdfPos, &pdfDir);
  pdf = pdfDir * invDist2;
  // 如果 v 是在表面上，就乘以cos项，得到与每单位面积有关的概率密度
  if (v.IsOnSurface()) pdf *= AbsDot(v.ng(), w);
  return pdf;
}
```

PBRT 中并没有类似光源 PDF 的相机 PDF，即 Vertex::PdfCamera() 函数，这是因为 PBRT 中的相机并不是一个实体，它不属于场景的一部分，光线无法与相机求交，因此也不需要进行查询概率密度。

Pdf() 和 PdfLight() 方法使用在另一个给定顶点的位置测量的在当前顶点实施的重要性策略的方向概率密度。然而，这不足以完全描述路径端点的行为，其采样路径从 $4D$ 分布生成采样射线（$2D$ 平面采样 $+2D$ 方向采样）。另一个 PdfLightOrigin() 方法通过提供关于光源本身的样本的空间分布的信息来弥补（出于与之前相同的原因，不需要相机端点的专用 PdfCameraOrigin() 方法），在后面路径采样中遇到会再次解释 PdfLightOrigin() 方法。

4. 构建光源子路径和相机子路径



本章讲解双向路径追踪算法中构建光源子路径和相机子路径的详细过程。

4.1 总体介绍

GenerateCameraSubpath() 和 GenerateLightSubpath() 都是先进行一些初始化工作，然后再调用 RandomWalk() 函数来构建子路径。

传入 GenerateCameraSubpath 函数的 maxDepth 是 BDPTIntegrator 类定义的 maxDepth 再加上 2;传入 GenerateLightSubpath 函数的 maxDepth 是 BDPTIntegrator 类定义的 maxDepth 再加上 1。这在前面描述过。

子路径构造完以后，就循环来连接子路径成完整路径：

```
for (int t = 1; t <= nCamera; ++t) {
  for (int s = 0; s <= nLight; ++s) {
      int depth = t + s - 2;
      ......
  }
}
```

我们先分别介绍两个函数的初始化过程，然后再介绍 RandomWalk() 函数。

4.2 构建相机子路径

GenerateCameraSubpath() 可以分为两步：（1）从相机子路径中采样初始射线；（2）从相机子路径中生成第一个顶点，并开启随机行走函数 RandomWalk()。

在第（1）步中，首先通过 GenerateRayDifferential 函数生成射线微分，然后用 ScaleDifferentials 函数来放缩微分量（这都是常规路径追踪里的内容）。

此时，beta=1，并传入 CreateCamera 函数来构建相机顶点：

```
Spectrum beta = camera.GenerateRayDifferential(cameraSample, &ray);
ray.ScaleDifferentials(1 / std::sqrt(sampler.samplesPerPixel));
path[0] = Vertex::CreateCamera(&camera, ray, beta);
camera.Pdf_We(ray, &pdfPos, &pdfDir);
```

我们在 RandomWalk() 函数中只用到了 pdfDir，因为我们需要知道生成下一个顶点的概率（也就是 pdfDir）。这个 pdfDir 是跟投影立体角相关的概率（可以参考 Veach 论文解读之《双向路径追踪》）。

RandomWalk() 函数中的 maxDepth 参数为 maxDepth-1，Vertex *path 参数为 path+1（因为已经生成了 path[0] 顶点，所以其他顶点从 path[1] 开始生成，在 RandomWalk() 函数中生成的最大路径顶点数为 maxDepth-1）。

值得注意的是，相机子路径中，path[0].pdfRev 会在 RandomWalk() 中被赋值，而 path[0].pdfFwd 并没有被赋值，但是它们在后面计算路径 MIS 权重中都没有用到（即计算 r_i 时并不会用到该值，我们后面会再提一下）。

4.3 构建光源子路径

GenerateLightSubpath() 函数中，lightPdf 表示选择从哪个光源进行采样的概率，pdfPos 和 pdfDir 的含义与之前相同，就是在确定了采样哪个光源以后，采样该光源上的某个点的概率以及发光方向的概率。

以漫反射面光源为例，Sample_Le 函数的采样方向的概率密度就是 CosineHemispherePdf 计算的概率密度。

生成第一个顶点定义为：

```
path[0] = Vertex::CreateLight(light.get(), ray, nLight, Le, pdfPos *
    lightPdf);
Spectrum beta = Le * AbsDot(nLight, ray.d) / (lightPdf * pdfPos *
    pdfDir);
int nVertices = RandomWalk(scene, ray, sampler, arena, beta, pdfDir,
    maxDepth - 1, TransportMode::Importance, path + 1);
```

采样到 path[0] 顶点的概率是 lightPdf*pdfPos，也就是选择采样哪个光源的概率乘以在这个光源上采样到这个位置的概率。传入到 RandomWalk() 函数中的概率密度依然是 pdfDir，也就是采样到下一个顶点的概率。我们同样忽略关于无限面光源的内容。

这里的 beta 用相关联的采样权重来进行初始化。beta 的导出有很多种方式，比如光子映射中用“功率”的角度来理解；或者类似 PBRT 中的光子映射，光源发出辐射度经过多次散射一直到人眼中；或者双向方法中，对两个方向的路径用概率来估计，经过公式推导得到；但在各种方式下得到的最终公式都是一致的。

4.4 随机行走构建子路径

RandomWalk() 函数会接收给定的路径端点采样的方向和概率。其构造函数中，beta 表示路径吞吐量权重 (path throughput weight)，与路径追踪和光子映射中的 β 是具有同样含义的；以及一个 pdfFwd 变量作为采样方向的立体角相关的概率密度。

```
int RandomWalk() {
  if (maxDepth == 0) return 0;
  int bounces = 0;
  Float pdfFwd = pdf, pdfRev = 0;
  while (true) {
    <构建路径中的下一个顶点> {
      MediumInteraction mi;
      <追踪光线，如果有参与介质就采样参与介质>
      if(mi.IsValid()){
        <记录介质交点，计算前向概率密度>
        <采样方向，计算前一个顶点的反向概率密度>
      }else{
        <处理和构建表面交点>
      }
    }
    <计算上一个顶点处的反概率密度>{
      prev.pdfRev = vertex.ConvertDensity(pdfRev, prev);
    }
  }
}
```

4.4.1 参与介质

在 < 记录介质交点，计算前向概率密度 > 即调用 CreateMedium() 函数，该函数会创建一个参与介质交点，然后根据之前的顶点和当前新生成的顶点来做一个概率密度转化，转化为面积相关的概率密度：

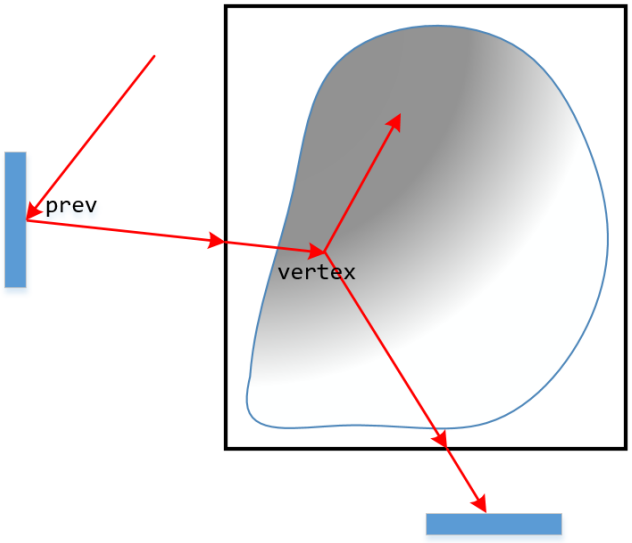
```
v.pdfFwd = prev.ConvertDensity(pdf, v);
```

在 < 采样方向，计算前一个顶点的反向概率密度 > 中，pdfFwd 和 pdfRev 的值是一样计算的。最后等创建完顶点以后再将 pdfRev 转化为立体角相关的概率密度（这对表面顶点也是一样的）：

```
  pdfFwd = pdfRev = mi.phase->Sample_p(-ray.d, &wi, sampler.Get2D());
  prev.pdfRev = vertex.ConvertDensity(pdfRev, prev);
```

假设的随机游走的前一顶点处的采样密度，由于相位函数相对于其参数通常是对称的，因此我们只需重用 pdfFwd 计值。

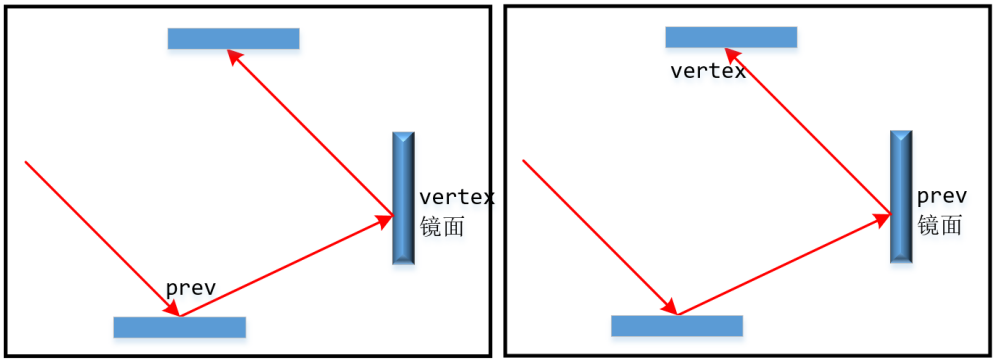
我们细化一下该过程，此时，上一个顶点是表面交点，Ray 与参与介质包围盒相交，不记录顶点，而是继续前进，并在参与介质散射：



此时，前向由 prev 找到 vertex 的概率密度就是在 prev 进行散射的概率密度。从 vertex 反向找到 prev 的概率密度就是在 vertex 上散射的概率密度，该概率密度与从 vertex 经过散射找到下一个顶点的概率密度是相同的。

4.4.2 表面交点

在表面求交的过程也相似，只是在采样新的方向并计算 pdfFwd 和 beta 以后，还需要判断当前散射类型是否是镜面散射。如果是镜面散射，当前顶点记录为 delta 类型顶点，且 pdfRev 和 pdfFwd 都设为 0。我们再用一个图描述一下它的意义：



第i次迭代　　第i+1次迭代

在第一次迭代中，prev 反射的光线与镜面相交，从 prev 找到 vertex 的概率密度是大于 0 的。但是从镜面找到 prev 的概率密度为 0，因为镜面点是 delta 分布的点；同理，在第二次迭代中，prev 就是镜面点，从 prev 通过镜面找到 vertex 的概率密度也是 0。所以在第 i 次迭代中，如果当前点是镜面点，就设 pdfFwd=0，也就是说从镜面点追踪到下一个顶点的概率是 0（该值会在

下次循环迭代中赋值到下一个顶点的 pdfPwd 成员变量）。

4.5 生成顶点的函数

本节总结一下所有生成顶点的函数，但不再做过多解释。

先关注生成光源顶点的函数：

```
// 创建光源初始点时使用
inline Vertex Vertex::CreateLight(const Light *light, const Ray &ray,
    const Normal3f &Nl, const Spectrum &Le, Float pdf) {
  Vertex v(VertexType::Light, EndpointInteraction(light, ray, Nl), Le)
      ;
  v.pdfFwd = pdf;
  return v;
}
// 两种用途：
// 用途1：当相机射线没有与任何内容相交时使用，只会在构建相机路径时使用
// 用途2：s=1时，用于直接采样光源来创建光源交点
inline Vertex Vertex::CreateLight(const EndpointInteraction &ei, const
    Spectrum &beta, Float pdf) {
  Vertex v(VertexType::Light, ei, beta);
  v.pdfFwd = pdf;
  return v;
}
```

在 BDPT 的核心代码中，生成光源顶点有以下几句：

```
// 创建光源初始点时使用
path[0] = Vertex::CreateLight(light.get(), ray, nLight, Le, pdfPos *
    lightPdf);
// 当相机射线没有与任何内容相交时使用，只会在构建相机路径时使用
vertex = Vertex::CreateLight(EndpointInteraction(ray), beta, pdfFwd);
// s=1时，用于直接采样光源来创建光源交点
sampled = Vertex::CreateLight(ei, lightWeight / (pdf * lightPdf), 0);
```

生成相机顶点的函数：

```
inline Vertex Vertex::CreateCamera(const Camera *camera, const Ray &
    ray, const Spectrum &beta) {
  return Vertex(VertexType::Camera, EndpointInteraction(camera, ray),
      beta);
}
```

```
4 inline Vertex Vertex::CreateCamera(const Camera *camera, const
    Interaction &it, const Spectrum &beta) {
5   return Vertex(VertexType::Camera, EndpointInteraction(it, camera),
      beta);
6 }
```

生成表面顶点或者参与介质顶点的函数：

```
1 inline Vertex Vertex::CreateSurface(const SurfaceInteraction &si,
    const Spectrum &beta, Float pdf, const Vertex &prev) {
2   Vertex v(si, beta);
3   v.pdfFwd = prev.ConvertDensity(pdf, v);
4   return v;
5 }
6 inline Vertex Vertex::CreateMedium(const MediumInteraction &mi, const
    Spectrum &beta, Float pdf, const Vertex &prev) {
7   Vertex v(mi, beta);
8   v.pdfFwd = prev.ConvertDensity(pdf, v);
9   return v;
10 }
```

4.6 子路径的连接

ConnectBDPT() 函数由下面几个步骤组成：

```
1   <忽略无限面光源带来的无效连接>
2   <执行连接，把贡献写入L>
3   <计算连接策略的MIS权重>
```

连接形式一共有四种：

```
1 s=0: 相机子路径是一个完整的路径
2 s=1: 在光源采样一个点，连接到相机子路径上
3 t=1: 相机镜头上采样一个点，连接到光源子路径上
4 t>1,s>1: 执行双向方法的连接策略
```

4.6.1 路径 $s = 0$

此时，如果相机路径的最后一个点是光源，那么就返回光照结果：

```
1 const Vertex &pt = cameraVertices[t - 1];
2 if (pt.IsLight()) L = pt.Le(scene, cameraVertices[t - 2]) * pt.beta;
```

pt.Le() 函数会根据顶点位置来计算方向，pt.beta 就是估计光线重要性策略的 beta。

4.6.2 路径 $t = 1$

此时直接从相机镜头上采样一个新的点，而不是使用原先相机路径中镜头上的点。如果光源子路径顶点 q_{s-1} 支持采样连接（非镜面点，且成功在镜头上采样到了点），就进行连接；否则它的贡献就是 0。

代码中先通过 camera.Sample_Wi() 函数来采样 Wi，然后用 Wi 乘以光散射后的辐射度：

```
const Vertex &qs = lightVertices[s - 1];
sampled = Vertex::CreateCamera(&camera, vis.P1(), Wi / pdf);
L = qs.beta * qs.f(sampled, TransportMode::Importance) * sampled.beta;
```

此时因为路径贡献的像素可能就不是之前的像素了，所以在 Render() 函数中为每个像素定义了 pFilmNew。由于该路径不再贡献到当前像素，所以在计算 MIS 权重时也要注意到这一点。

4.6.3 路径 $s = 1$

相当于直接采样光，和采样相机镜头基本上是一致的。这里会根据光分布来采样。

4.6.4 路径 $t > 1, s > 1$

设相机路径 t 个顶点，光源路径 s 个顶点（PBRT 书 1011 页写反了），相机路径 $\overline{p}_t$ 和光源路径 $\overline{q}_s$ 则描述为：

$$\overline{p}_t = p_0, p_1, \ldots, p_{t-1}$$
$$\overline{q}_s = q_0, q_1, \ldots, q_{s-1}$$

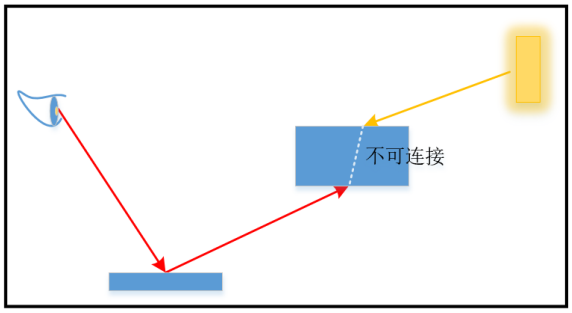
路径贡献可以估计为：

$$\hat{P}(\overline{q}_s\overline{p}_t) = L_e\hat{T}(\overline{q}_s)\Big[\hat{f}(q_{s-2} \to q_{s-1} \to p_{t-1})\hat{G}(q_{s-1} \leftrightarrow p_{t-1}) \\ \hat{f}(q_{s-1} \to p_{t-1} \to p_{t-2})\Big]\hat{T}(\overline{p}_t)W_e \tag{4.6.1}$$

其中 $\hat{T}$ 是吞吐量函数，$L_e\hat{T}(\overline{q}_s)$ 和 $\hat{T}(\overline{p}_t)W_e$ 早已被保存了（即各个路径下的 β 值）。

当有一个顶点定义为不可连接点，则连接失败。不可连接点，比如点光源、方向光、完全镜面材质（无漫反射或高光 BxDF 组件）。

此外，两条路径的权重以及连接时这两个 BSDF 通常是 0，这是因为一个点可能相对于另一个点位置在不可穿透的表面外侧：



之后，将计算的 L 值乘以 MIS 权重。

5. 多重重要性采样

本章讲解双向路径追踪算法中应用多重重要性采样方法组合不同的路径样本的原理和代码。本节难度要比前面的内容都要大，因此我刻意花了较长时间绘制了不少图示，在我曾经学习双向方法时，也是一遍一遍画图来理解每个符号和步骤，现在把这些分析结果呈现给读者们。

5.1 符号定义

即使没有多重重要性采样，双向方法在一些场景中效果仍会比路径追踪好得多，比如间接光占主导地位时（比如一个有灯罩的灯朝向天花板）。但是双向方法也会由于一些过大的路径贡献，导致图像有很多很亮的点，比如相机路径恰好找到了天花板上的一个很亮的地方。MIS 方法可以有效解决，它自动地认识到光子路径上如果能有至少一个散射事件时，采样策略会更准确（即光最好先散射一次，而不是相机路径一下就散射到了光能直接照到的地方）。

MISWeight() 就是计算 MIS 权重的。设 $n=s+t$，则一个路径可以写为：

$$\overline{x}=(x_0,\ldots,x_{n-1})=(q_0,\ldots,q_{s-1},p_{t-1},\ldots,p_0) \tag{5.1.1}$$

对于采样到顶点 x_i 的每单位面积的概率，重要性传输时定义为 $p^{\rightarrow}(x_i)$，辐射度传输时定义为 $p^{\leftarrow}(x_i)$，所以概率密度就可以写为：

$$p_s(\overline{x})=p^{\rightarrow}(x_0)\cdots p^{\rightarrow}(x_{s-1})\cdot p^{\leftarrow}(x_s)\cdots p^{\leftarrow}(x_{n-1}) \tag{5.1.2}$$

s 下标表示光源子路径有 s 个顶点。注意在 PBRT 中的描述，光源传输的是重要性，相机传输的是辐射度（其实本来应该是相机路径估计辐射度，传输重要性；光源路径传输辐射度，估计重要性）。

设 $i+j=s+t$，描述同样长度的路径时，相机和光源子路径可能有不同的长度：

$$p_i(\overline{x})=p^{\rightarrow}(x_0)\cdots p^{\rightarrow}(x_{i-1})\cdot p^{\leftarrow}(x_i)\cdots p^{\leftarrow}(x_{n-1}) \tag{5.1.3}$$

其中 $0\le i\le n$。

5.2 计算平衡启发式

平衡启发式权重：

$$w_s(\bar{x}) = \frac{p_s(\bar{x})}{\sum_i p_i(\bar{x})} \tag{5.2.1}$$

其他启发式其实都类似。

路径概率密度很容易低于或超过单精度甚至双精度可表示值的范围。单个顶点的面积密度与场景维度的平方成反比：将场景均匀缩放到其大小的一半将使顶点概率密度增加四倍。例如当计算具有 10 个顶点的路径的面积积密度时，相同的缩放操作将导致路径密度增加约一百万倍。当处理非常小或非常大的场景时（与单位体积的康奈尔盒相比），$p_i(\bar{x})$ 可以很容易超过有效范围。

其次，简单的 MIS 实现具有 $O(n^4)$ 的时间复杂度，其中 n 是最大路径长度，即使用各个顶点密度相乘计算出 $p_i(\bar{x})$ 会比较耗时。至于为什么是 $O(n^4)$ 的时间复杂度：由于一共最多有 n 个顶点，计算 $p_i(\bar{x})$ 的复杂的就是 n；对于每个路径，计算所有的 p_i 的复杂度也是 $O(n)$；而由于路径的数量级是 $O(n^2)$，所以计算 MIS 的复杂度就是 $O(n^4)$。

把权重写为比值形式再计算就会避免数值问题和效率问题：

$$w_s(\bar{x}) = \frac{1}{\sum_i \frac{p_i(\bar{x})}{p_s(\bar{x})}} = \left(\sum_{i=0}^{s-1} \frac{p_i(\bar{x})}{p_s(\bar{x})} + 1 + \sum_{i=s+1}^{n} \frac{p_i(\bar{x})}{p_s(\bar{x})} \right)^{-1} \tag{5.2.2}$$

定义 $r_i(\bar{x}) = \frac{p_i(\bar{x})}{p_s(\bar{x})}$，可以改写为：

$$r_i(\bar{x}) = \frac{p_i(\bar{x})}{p_s(\bar{x})} = \frac{p_i(\bar{x})}{p_{i+1}(\bar{x})} \frac{p_{i+1}(\bar{x})}{p_s(\bar{x})} = \frac{p_i(\bar{x})}{p_{i+1}(\bar{x})} r_{i+1}(\bar{x}) \quad (i < s)$$
$$r_i(\bar{x}) = \frac{p_i(\bar{x})}{p_s(\bar{x})} = \frac{p_i(\bar{x})}{p_{i-1}(\bar{x})} \frac{p_{i-1}(\bar{x})}{p_s(\bar{x})} = \frac{p_i(\bar{x})}{p_{i-1}(\bar{x})} r_{i-1}(\bar{x}) \quad (i > s)$$

且我们可以知道：

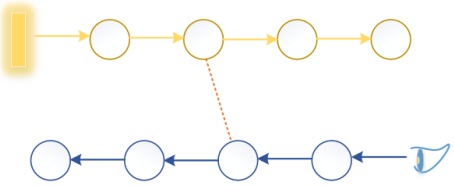
$$\frac{p_i(\bar{x})}{p_{i+1}(\bar{x})} = \frac{p^{\rightarrow}(x_0) \cdots p^{\rightarrow}(x_{i-1}) \cdot p^{\leftarrow}(x_i) \cdot p^{\leftarrow}(x_{i+1}) \cdots p^{\leftarrow}(x_{n-1})}{p^{\rightarrow}(x_0) \cdots p^{\rightarrow}(x_{i-1}) \cdot p^{\rightarrow}(x_i) \cdot p^{\leftarrow}(x_{i+1}) \cdots p^{\leftarrow}(x_{n-1})} = \frac{p^{\leftarrow}(x_i)}{p^{\rightarrow}(x_i)}$$
$$\frac{p_i(\bar{x})}{p_{i-1}(\bar{x})} = \frac{p^{\rightarrow}(x_0) \cdots p^{\rightarrow}(x_{i-2}) \cdot p^{\rightarrow}(x_{i-1}) \cdot p^{\leftarrow}(x_i) \cdots p^{\leftarrow}(x_{n-1})}{p^{\rightarrow}(x_0) \cdots p^{\rightarrow}(x_{i-2}) \cdot p^{\leftarrow}(x_{i-1}) \cdot p^{\leftarrow}(x_i) \cdots p^{\leftarrow}(x_{n-1})} = \frac{p^{\rightarrow}(x_{i-1})}{p^{\leftarrow}(x_{i-1})}$$

由此我们最终得到关系式：

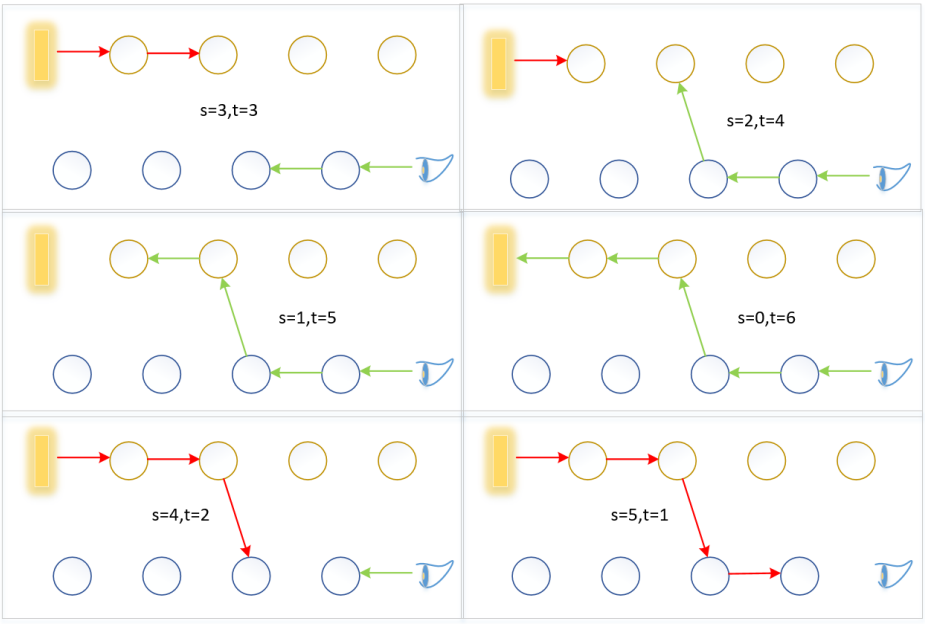
$$r_i(\bar{x}) = \begin{cases} 1, & if\ i = s \\ \frac{p^{\leftarrow}(x_i)}{p^{\rightarrow}(x_i)} r_{i+1}(\bar{x}), & if\ i < s \\ \frac{p^{\rightarrow}(x_{i-1})}{p^{\leftarrow}(x_{i-1})} r_{i-1}(\bar{x}), & if\ i > s \end{cases} \tag{5.2.3}$$

5.3 详细分析

假如我们成功连接了一个路径：



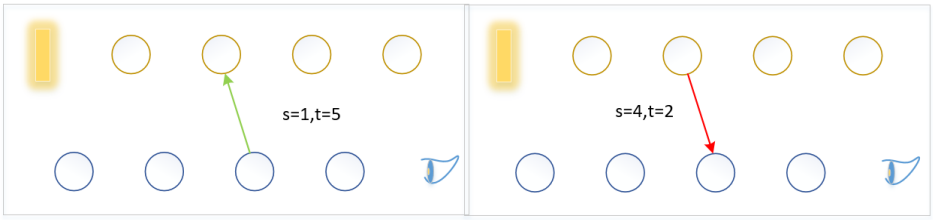
我们需要计算它在整个路径长度为 $n = s + t = 3 + 3 = 6$ 的所有策略中的概率密度。比如有的策略要采样四个光源顶点和两个相机顶点，有的策略要采样四个相机顶点和两个光源顶点。总共有六种策略（红色路径表示光源子路径构建，绿色表示相机子路径构建）：



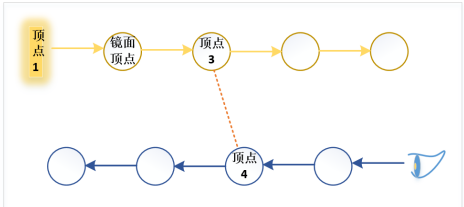
这六种策略在某个场景下可以得到这样的结果：

注意图中有镜面物体，因此当 $s = 5, t = 1$ 时，看到的镜面物体完全是黑的，此时路径连接不成功（此时从相机上采样一个顶点的策略连接到光源路径的概率为 0），但 MIS 方法会将更高的权重分配给 $s = 2$ 以及 $s = 2$ 的路径，使得镜面物体效果更好。而对于漫反射区域，则光源子路径顶点数较高时效果更好，因此在这些区域的样本中 MIS 会将更高的权重分配给 $s = 4$ 和 $s = 5$。

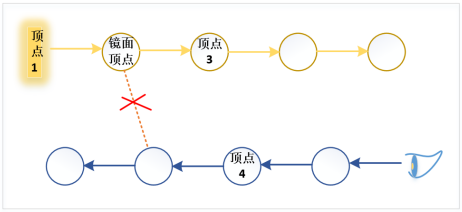
注意下面连线的这些路段中，生成箭头所指的顶点的概率密度是需要重新计算的，因为原来保存的 pdfPwd 和 pdfRev 都不会包含这些信息：



现在再考虑某种特例，镜面物体：



光源路径生成该镜面顶点的概率是大于 0 的（即镜面顶点的 pdfPwd 值大于 0），而从镜面顶点生成到顶点 3 的概率为 0（即顶点 3 的 pdfPwd 值等于 0）；且从顶点 3 生成镜面顶点的概率为 0（即镜面顶点的 pdfRev 值为 0）。由于是镜面顶点，所以下面的连接是不合规的：



也就是说，总路径长度为 $n = 6$ 的时候，$s = 2$ 这种路径是不存在的。在计算 MIS 权重时，这种情况也会被跳过（且不对最终图像有贡献）。

5.4 PBRT 中的代码描述

当总路径只有两个顶点时，只可能是相机子路径生成的。因为同时从相机子路径和光源子路径采样一个顶点的方式并不存在，且相机子路径又必须采样一个及以上的顶点，因此两个顶点都是相机子路径生成的，此路径权重为 1。

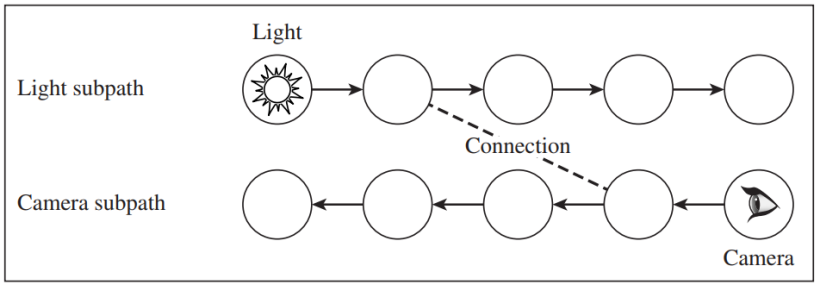
之后其他的情况都会计算到 sumRi 这个变量中（将全部 r_i 都加起来），在程序开始之前会定义一个帮助函数 remap0：

```
auto remap0 = [](Float f) -> Float { return f != 0 ? f : 1; };
```

该函数将 0 值映射为 1，这是为了在路径中处理 Dirac delta 函数的。这种顶点不能被连接策略进行连接，但在计算已有路径顶点概率时，它会出现在分母和分子上然后相互抵消，因此我们直接映射到 1。后面遇到以后会再解释。

计算顶点概率密度需要 Vertex::pdfFwd 和 Vertex::pdfRev，这些值仅仅捕获相机和原始光源子路径的信息，而在当相机子路径或者光源子路径的顶点数目为 1，则会直接采样相机或者直接采样光，则需要重新计算。在最后合成之前，我们会先把概率密度都矫正，然后再用先前的迭代方法。由于矫正步骤比较复杂，我们放在下一节介绍。

计算权重的基本步骤分为两步：



第一步从相机子路径开始，第二步从光源子路径开始：

```
<考虑相机子路径的假设的连接策略>
<考虑光源子路径的假设的连接策略>
```

5.4.1 公式与详细图示

我们再列一下公式：

$$r_i(\overline{x}) = \frac{p_i(\overline{x})}{p_s(\overline{x})}$$

$$w_s(\overline{x}) = \frac{1}{\sum_i \frac{p_i(\overline{x})}{p_s(\overline{x})}} = \left(\sum_{i=0}^{s-1} \frac{p_i(\overline{x})}{p_s(\overline{x})} + 1 + \sum_{i=s+1}^{n} \frac{p_i(\overline{x})}{p_s(\overline{x})} \right)^{-1}$$

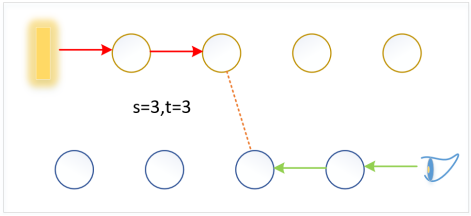
$$r_i(\overline{x}) = \begin{cases} 1, & if\ i = s \\ \frac{p^{\leftarrow}(x_i)}{p^{\rightarrow}(x_i)} r_{i+1}(\overline{x}), & if\ i < s \\ \frac{p^{\rightarrow}(x_{i-1})}{p^{\leftarrow}(x_{i-1})} r_{i-1}(\overline{x}), & if\ i > s \end{cases}$$

上标箭头朝左表示相机路径得到某个点，上标箭头朝右表示光源路径得到的某个点（与上图的朝向是一致的）。

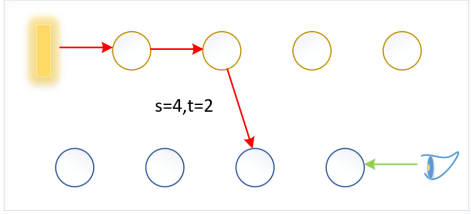
对于长度为 $n = s + t$ 的路径，光源路径部分（$i < s$）和相机路径部分（$i > s$）分别用两个循环来处理：

```
1 <考虑相机子路径的假设的连接策略>
2 Float ri = 1;
3 // t-1种可能，因为相机顶点数不能为0
4 for (int i = t - 1; i > 0; --i) {.....}
5 <考虑光源子路径的假设的连接策略>
6 // s种可能
7 ri = 1;
8 for (int i = s - 1; i >= 0; --i) {.....}
```

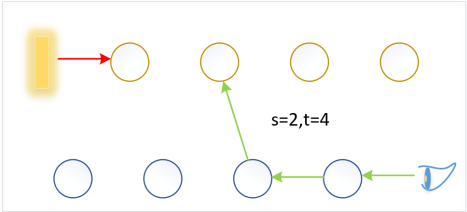
也就是说，当连接点位于 $s = 3, t = 3$ 这种情况时，$r_3(\overline{x}) = 1$：



下图对应于 $r_4(\overline{x}) = r_3(\overline{x}) \frac{p^{\rightarrow}(x_3)}{p^{\leftarrow}(x_3)}$（注意，$p^{\rightarrow}(x_3)$ 表示光源路径生成顶点 x_3 的概率密度，该值会在前面的步骤中矫正，关于矫正我们放在下一节介绍）：



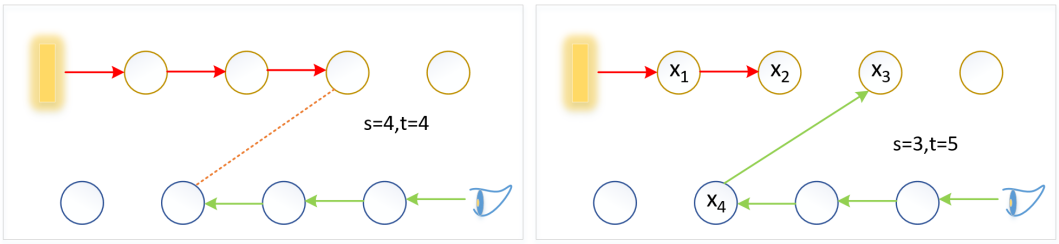
下图对应于 $r_2(\overline{x}) = r_3(\overline{x}) \frac{p^{\leftarrow}(x_2)}{p^{\rightarrow}(x_2)}$（注意，$p^{\leftarrow}(x_2)$ 表示光源路径生成顶点 x_2 的概率密度，同理，该值会在前面的步骤中矫正）：



另外值得注意的是，$r_5(\bar{x}) = r_4(\bar{x})\frac{p^{\rightarrow}(x_4)}{p^{\leftarrow}(x_4)}$，此时相当于 $i > s$ 部分的 MIS 权重已经计算完毕，也就是对于 $n = 6$ 的策略中，我们并不会用到 $p^{\rightarrow}(x_5)$ 和 $p^{\leftarrow}(x_5)$，也就是最后一个顶点（相机子路径的第一个顶点）的 pdfPwd 和 pdfRev。前面提起过，相机子路径中，path[0].pdfRev 会在 RandomWalk() 中被赋值，而 path[0].pdfFwd 并没有被赋值，但是它们在后面计算路径 MIS 权重中都没有用到（即计算 r_i 时并不会用到该值）。

5.4.2 注意镜面点的处理

但是需要注意的是，要想生成某个顶点，不但要保证当前顶点不是 delta 类型的顶点，还需要保证前一个顶点也不能是 delta 类型的顶点，否则由前一个顶点生成当前顶点的概率一定是 0。比如下面的图示：



此时路径长度为 $n = s + t = 4 + 4 = 8$。右图中表示为了计算 MIS 权重规划的路径，$r_4(\bar{x}) = 1$，$r_3(\bar{x}) = \frac{p^{\leftarrow}(x_3)}{p^{\rightarrow}(x_3)} r_4(\bar{x})$。如果 x_4 是镜面点，那么从 x_4 生成 x_3 的概率密度就是 0，即 $p^{\leftarrow}(x_3) = 0$，遇到这种情况时需要排除。

如果 x_3 是镜面点，那么上图中，从 x_4 散射到 x_3 方向的光经过镜面散射后，恰好到 x_2 的概率为 0（也就是说如果 x_3 是镜面点，则 x_2, x_3, x_4 三点能够构成路径的概率为 0）。

以上两种情况，也就是说，对于当前路径为 $\bar{x}$，光源顶点数量为 s，在计算 MIS 权重时的累积 $r_i(\bar{x})$ 时，当 $i < s$ 时，要保证点 x_i 和点 x_{i+1} 不能是镜面点：

```
for (int i = t - 1; i > 0; --i) {
  ri *= remap0(cameraVertices[i].pdfRev) / remap0(cameraVertices[i].
     pdfFwd);
  if (!cameraVertices[i].delta && !cameraVertices[i - 1].delta)
      sumRi += ri;
}
```

光源路径那部分也是同理，不再赘述。注意因为双向方法中，相机默认只实现了透视投影相机，所以相机不是 delta 类型的，而光源却有很多 delta 类型的，比如方向光和点光源，因此光源部分需要判断是否是 delta 类型的光源：

```
for (int i = s - 1; i >= 0; --i) {
```

```
    ri *= remap0(lightVertices[i].pdfRev) / remap0(lightVertices[i].
        pdfFwd);
    bool deltaLightvertex = i > 0 ? lightVertices[i - 1].delta :
        lightVertices[0].IsDeltaLight();
    if (!lightVertices[i].delta && !deltaLightvertex) sumRi += ri;
}
```

计算完以后，返回 (1/1+sumRi) 的值。

5.5 根据当前策略来临时更新顶点属性

在上一节中计算权重之前，需要 < 为当前策略临时修改更新顶点属性 >。为了避免过多的代码量来对更新和后续的清理，我们定义一个帮助类 ScopedAssignment（范围内分配），它临时修改给定变量，并在程序离开作用域以后恢复。它存储了指向内存中任意类型的指针 ScopedAssignment::target 以及原始值 ScopedAssignment::backup：

```
template <typename Type>
class ScopedAssignment {
  public:
    // ScopedAssignment Public Methods
    ......
    ~ScopedAssignment() {
        if (target) *target = backup;
    }
  private:
    Type *target, backup;
};
```

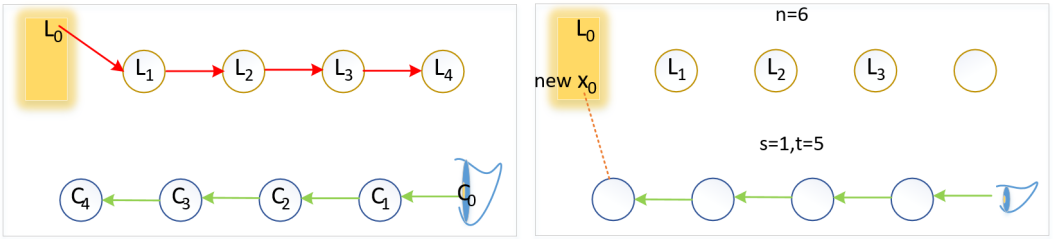
主要的更新操作包含了寻找连接顶点以及它的前面的顶点，并更新顶点概率密度以及其他属性：

- (1)< 查找连接顶点和它们的上一个顶点 >
- (2)< 更新策略中 s=1 或者 t=1 的顶点 >
- (3)< 把连接顶点标记为非退化 (non-degenerate)>
- (4)< 更新顶点 p_{t-1} 的反向概率密度 >
- (5)< 更新顶点 p_{t-2} 的反向概率密度 >
- (6)< 更新顶点 q_{s-1} 和 q_{s-2} 的反向概率密度 >

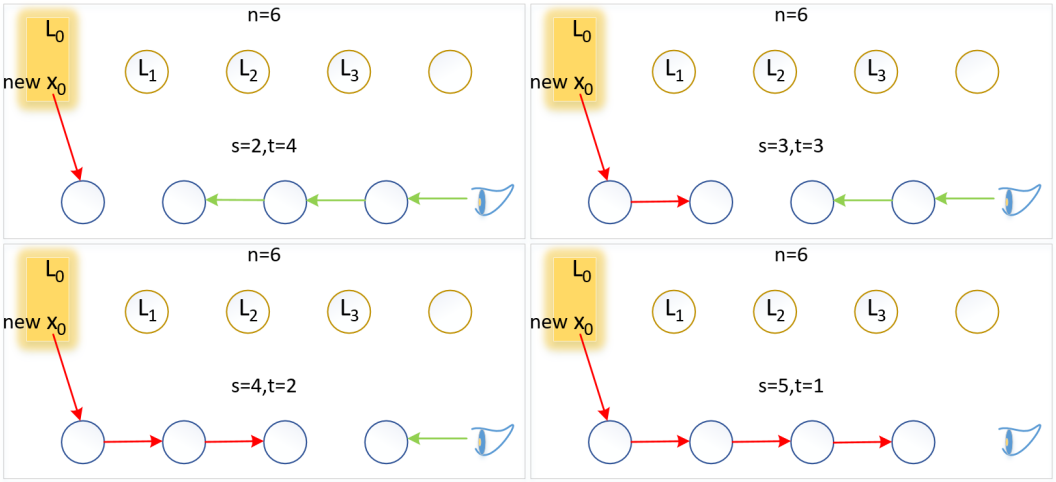
在步骤 (1) 中，相当于找到下面的几个顶点：

```
lightVertices[s - 1] //s>0才存在
cameraVertices[t - 1] //t>0才存在
lightVertices[s - 2] //s>1才存在
cameraVertices[t - 2] //t>1才存在
```

MISWeight() 是被函数 ConnectBDPT() 调用的。对于步骤 (2)，当前的连接路径中，当 $s=1$ 或者 $t=1$ 时，需要对光源或者相机采样新的点，采样到的点保存在变量 sampled（Vertex 类型）中。参考如下示意图：



左图表示构建的初始路径。右图表示当前连接策略是 $s=1, t=5$，此时需要从光源上采样一个新的点。对于新的完整路径，对 $n=6$ 其他策略计算权重：



对于步骤 (3)，考虑有些材质 BSDF 同时包含了镜面散射和非镜面散射组件（比如 UberMaterial）。如果在构建子路径时，采样的 BSDF 是它的镜面组成部分，则它的 delta 标志就设为 true；但是在计算路径连接时，因为它包含有非镜面组件，所以有可能会把路径连接起来，所以这里先统一把 pt 和 qs 指向的连接点的 delta 标志都设为 false（指针 pt 和 qs 分别指向两条子路径的连接点）。即使是完美镜面材质也要这样设置，这是为了方便后面的计算。如果概率密度计算为 0 就可以完全排除完美镜面材质了。

在 ConnectBDPT() 调用 MISWeight() 之前会判断 qs 和 pt 是否存在非镜面组件（通过 IsConnectible() 函数来判断），如果这两个点都包含非镜面组件，才会继续执行 MIS 权重的计算，所以只要开始计算 MIS 权重，则 ps 和 pt 必定都是可以相互连接的点。

对于步骤 (4)，更新顶点 p_{t-1} 的反向概率密度。注意对于 p_{t-1} 点，反向的概率密度是采样 $q_{s-2} \to q_{s-1} \to p_{t-1}$ 得到的。当 $s=0$ 时，p_t 是相机子路径与光源的交点，相应的反向概率密度则是通过 Light::Sample_Le() 函数采样一个光源样本点（只采样点，不采样方向），它的概率密度

通过 Vertex::PdfLightOrigin() 函数来实现。

对于步骤 (5)，更新顶点 p_{t-2} 的反向概率密度。反向的概率密度是采样 $q_{s-1} \to q_{t-1} \to p_{t-1}$ 得到的。注意 $s=0$ 仍然是一个特例，此时概率密度不但包含了点的位置，还包含了方向，通过调用 Vertex::PdfLight() 函数来实现。

对于步骤 (6)，和 (4)(5) 很相似，只是不存在 $s=0$ 这种情况了。计算概率密度通过调用 Vertex::Pdf 来实现，传入参数 prev 表示 cameraVertices[t-2]，如果它存在，说明此时 pt 不是相机上的点，此时计算概率密度会通过 BSDF::Pdf() 函数或者 PhaseFunction::Pdf() 函数来实现；如果它不存在，说明此时 pt 是相机上的点，就需要调用 Camera::Pdf_We() 函数。

5.6 特殊路径的理解

现在我们考虑一下 $t=1$ 时的贡献如何计算到整幅图像上。

PBRT 中，$t=1$ 时的结果会保存在 pFilmNew 中，它可能与当前像素的 pFilm 并不在同一个像素内。

但是要注意的是，每 spp 中，我们每个像素都会采样一个样本，换句话说，每个像素都会采样到 $t=1$ 的这种路径，因此可以说，$t=1$ 这种情况应该需要单独考虑——它确实是双向路径追踪的一部分，但是它贡献的像素却不一定是当前像素所在邻域。在《Veach-模拟光传输的鲁棒的蒙特卡洛方法》中的《双向路径追踪》中我们介绍过，$t=1$ 的情况（PBRT 不支持 $t=0$ 的情况）会单独保存在一张光图 (light map) 中，而其他情况都会保存在相机图 (camera map) 中，最后两张图会合并。

$t=1$ 的路径会计算 MIS 权重。我们知道，每个路径在计算 MIS 权重时，长度为 $n>3$ 的路径都会考虑到 $s=1, t=n-1$ 这种策略，而这种策略的原始方法，也就是真的从相机上采样一个顶点，然后借用光源路径上的两个及以上的顶点数的贡献会随机散布到各个像素内。假如我们对双向方法采样 spp 为 100（构建 100 条相机子路径和 100 条光源子路径，再互相连接），那么我们先把这些所有的 $t=1$ 的 MIS 加权路径贡献保存到光图中，最后将光图的值除以 100，所谓光图中单个像素的估计，然后再将光图的值和相机图的值相加，就是最后双向方法的输出：

```
1  std::unique_ptr<FilmTile> filmTile = camera->film->GetFilmTile(
       tileBounds);
2  for (int t = 1; t <= nCamera; ++t) {
3    for (int s = 0; s <= nLight; ++s) {
4      // 计算双向方法
5      Point2f pFilmNew;
6      ......
7      if (t != 1)
8        L += Lpath;
9      else // 贡献到新的像素中
10       film->AddSplat(pFilmNew, Lpath);
11   }
12 }
```

```
13 filmTile->AddSample(pFilm, L);
```

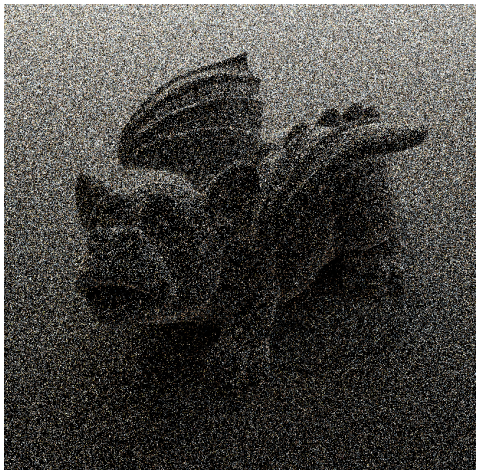
合并以后，最后会根据样本量来缩放像素值：

```
1 film->WriteImage(1.0f / sampler->samplesPerPixel);
```

我们注释掉相机图的值，只看光源图：

```
1 if (t != 1)
2   ;// L += Lpath;
3 else
4   film->AddSplat(pFilmNew, Lpath);
```

经过 512spp 渲染得到下面的结果：



5.7 关于采样相机 We

5.7.1 Veach 论文的表示与 PBRT 中的区别与联系

这里的符号表示请参考《Veach-模拟光传输的鲁棒的蒙特卡洛方法》中的《双向路径追踪》。我们对比 PBRT 中的 β 和 Veach 论文中的 α 之间的关系。为了理解起来更容易，我们只考虑只有一个面光源，场景全都是漫反射材质，没有参与介质。

在构建相机子路径时，PBRT 的做法表示如下。其中，β_i^C 表示 β 在第 i 个顶点的路径吞吐量（类似于路径追踪），第 0 个顶点 $\mathbf{z}_0$ 在相机上。

$$
\begin{aligned}
\beta_1^C &= 1 \\
\beta_2^C &= \frac{f(\mathbf{z}_2 \to \mathbf{z}_1 \to \mathbf{z}_0)|N(\mathbf{z}_1) \cdot \omega_{\mathbf{z}_1 \to \mathbf{z}_2}|}{p_\omega(\omega_{\mathbf{z}_1 \to \mathbf{z}_2})} \\
\beta_3^C &= \frac{f(\mathbf{z}_3 \to \mathbf{z}_2 \to \mathbf{z}_1)|N(\mathbf{z}_2) \cdot \omega_{\mathbf{z}_2 \to \mathbf{z}_3}|}{p_\omega(\omega_{\mathbf{z}_2 \to \mathbf{z}_3})} \times \beta_2^C \\
\beta_4^C &= \frac{f(\mathbf{z}_4 \to \mathbf{z}_3 \to \mathbf{z}_2)|N(\mathbf{z}_3) \cdot \omega_{\mathbf{z}_3 \to \mathbf{z}_4}|}{p_\omega(\omega_{\mathbf{z}_3 \to \mathbf{z}_4})} \times \beta_3^C
\end{aligned}
$$

相机射线从 $\mathbf{z}_0$ 出发，采样到表面顶点 $\mathbf{z}_1$，该点记录 β_1^C，然后采样散射方向并计算 β_2^C；之后进入第二轮循环，采样到表面顶点 $\mathbf{z}_2$，该点记录 β_2^C，然后采样新的散射方向并计算 β_3^C；以此类推。

在构建光源子路径时，PBRT 的做法表示如下。其中，β_i^L 表示 β 在第 i 个顶点的路径吞吐量，第 0 个顶点 $\mathbf{y}_0$ 在光源上。

$$
\begin{aligned}
\beta_1^L &= \frac{Le \cdot |N(\mathbf{y}_0) \cdot \omega_{\mathbf{y}_0 \to \mathbf{y}_1}|}{p(\mathbf{y}_0) p_\omega(\mathbf{y}_0 \to \mathbf{y}_1)} \\
\beta_2^L &= \frac{f(\mathbf{y}_0 \to \mathbf{y}_1 \to \mathbf{y}_2)|N(\mathbf{y}_1) \cdot \omega_{\mathbf{y}_1 \to \mathbf{y}_2}|}{p_\omega(\omega_{\mathbf{y}_1 \to \mathbf{y}_2})} \times \beta_1^L \\
\beta_3^L &= \frac{f(\mathbf{y}_1 \to \mathbf{y}_2 \to \mathbf{y}_3)|N(\mathbf{y}_2) \cdot \omega_{\mathbf{y}_2 \to \mathbf{y}_3}|}{p_\omega(\omega_{\mathbf{y}_2 \to \mathbf{y}_3})} \times \beta_2^L \\
\beta_4^L &= \frac{f(\mathbf{y}_2 \to \mathbf{y}_3 \to \mathbf{y}_4)|N(\mathbf{y}_3) \cdot \omega_{\mathbf{y}_3 \to \mathbf{y}_4}|}{p_\omega(\omega_{\mathbf{y}_3 \to \mathbf{y}_4})} \times \beta_3^L
\end{aligned}
$$

光源射线从 $\mathbf{y}_0$ 出发，采样到表面顶点 $\mathbf{y}_1$，该点记录 β_1^L，然后采样散射方向并计算 β_2^L；之后进入第二轮循环，采样到表面顶点 $\mathbf{y}_2$，该点记录 β_2^L，然后采样新的散射方向并计算 β_3^L；以此类推。

可以看到，PBRT 中的 β 和 Veach 论文中的 α 基本是无差别的，除了在相机发射时，Veach 的方法会采样相机：

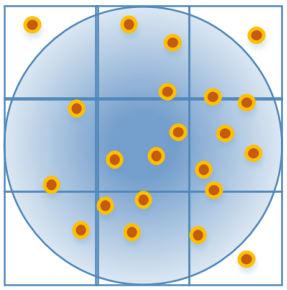
$$
\frac{W_e^{(0)}(\mathbf{z}_0)|N(\mathbf{z}_0) \cdot \omega_{\mathbf{z}_0 \to \mathbf{z}_1}|}{p(\mathbf{z}_0) p_\omega(\omega_{\mathbf{z}_0 \to \mathbf{z}_1})} \tag{5.7.1}
$$

而 PBRT 中采样相机路径时虽然会对每个像素发出的光线再采样光圈用来模拟景深，但并不会对相机顶点直接采样，这要保证每条入射光线的权重都是 1——我们下一个小节会详细介绍这个问题。

5.7.2 重要性与权重

在我写本小节时，我一直觉得关于相机初始权重 $\beta = 1$ 和 W_e 之间的关系实在是“只可意会不可言传”，所以能描述成什么样我也没有完全的把握，只能尽力尝试一下。

假如一个像素可以接收到邻域 3×3 个像素范围的入射辐射度，但随着入射辐射度的权重与所在位置随着像素中心的距离增加而增加。下图中，样本 L_i 所在圆内颜色越深的位置表示对于中间像素的贡献的权重 w_i 越高。最终像素值就是 $\frac{\sum_{i=1}^N w_i L_i}{\sum_{i=1}^N w_i}$。



如果不考虑像素邻域滤波，且设相机传感器响应值就是入射辐射度值，则假设每个像素只采样一个样本 L_1，那么该像素值就是 L_1，这也是为什么 $\beta = 1$（入射辐射度等于传感器响应值）。

但是当我们需要从相机采样一个点时，尽管每条入射光线对于相机响应权重都是一样的，但是采样到每条光线的比例却不一定相同，进而带来一个问题——它的响应值也不能相同。考虑对

相机采样时发生的情况：假设我们只会采样到两个方向 ω_1 和 ω_2，一个方向的光线比另外一个方向更容易采样到，采样 100 次，其中 70 次采样到 ω_1，30 次采样到 ω_2，那么按照取平均的思路，最终贡献到像素中也是 ω_1 方向估计的辐射度占比更高一些。

注意完整的路径空间表示应该是：

$$f_j(\overline{x}) = L_e(\mathbf{x}_0 \to \mathbf{x}_1)G(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1) \cdot \left(\prod_{i=1}^{k-1} f_s(\mathbf{x}_{i-1} \to \mathbf{x}_i \to \mathbf{x}_{i+1})G(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i+1})\right) W_e^{(j)}(\mathbf{x}_{k-1} \to \mathbf{x}_k)$$

也就是说相机顶点 $\mathbf{z}_0$ 与相机路径第二个点 $\mathbf{z}_1$ 之间也是有几何项的。原本的重要性函数应该满足：

$$W_e^{(j)}(p_0 \to p_1) = f_j(p_0)\delta\Big(t(p_0, \omega_{camera}(p_1)) - p_1\Big) \tag{5.7.2}$$

但是需要注意的是我们从某个点（这个点通常是光源路径最后一个点，即 lightVertices[s-1]）采样相机镜头时，并不会局限于某个像素范围，而是在任意空间都有可能，所以这里发射的重要性也不会局限于某个像素范围（主要是因为首先你并不知道你采样镜头时会贡献到哪个像素，其次就是某个样本不但会贡献到当前像素，还会对周边区域有一些影响）：

$$\int_{A_{lens}} \int_{\mathcal{S}^2} W_e(p, \omega)|\cos\theta| d\omega dA(p) = 1 \tag{5.7.3}$$

在第一章中有不少这些解释，可以配合起来一起看。

5.7.3 两个概率密度计算函数的区别

PerspectiveCamera::Pdf_We() 函数会在其他函数调用 Vertex::Pdf() 中被调用，或者 GenerateCameraSubpath() 创建相机路径时被直接调用。MISWeight() 会调用 Vertex::Pdf() 函数，但是只会在 $t = 1$ 时，此时相机路径只有一个顶点，pt 变量就是该顶点：

```
1  Vertex *pt = t > 0 ? &cameraVertices[t - 1] : nullptr;
```

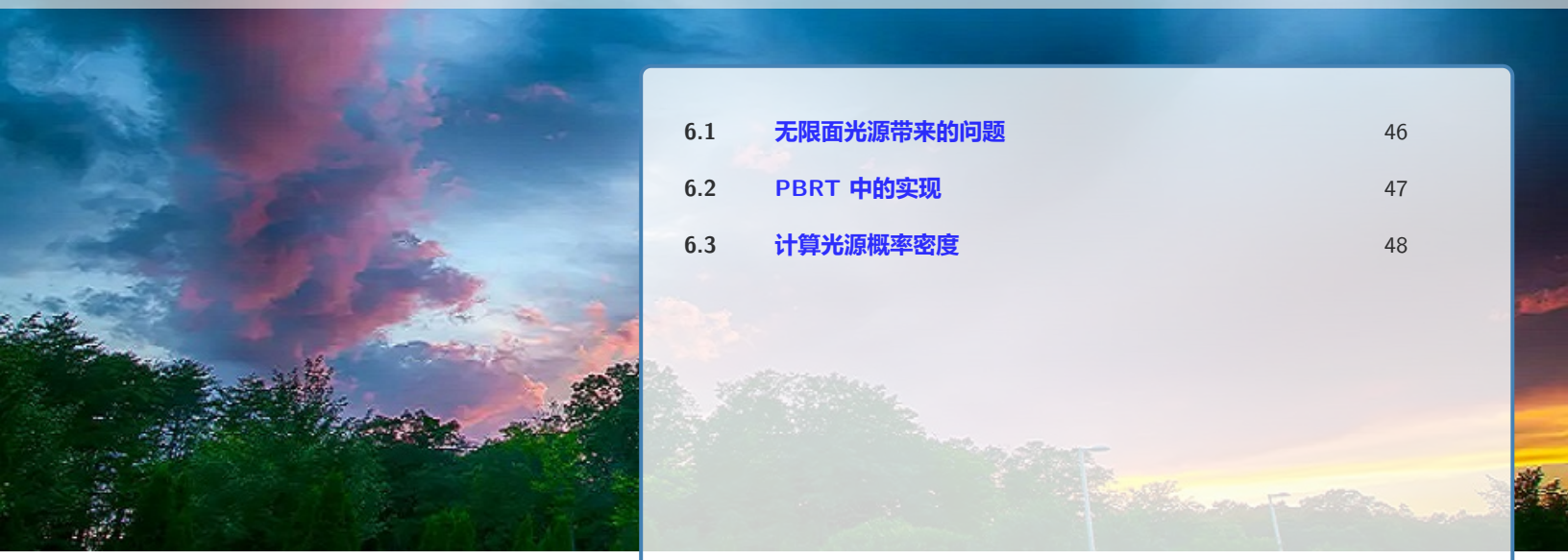
此时需要计算 qs->pdfRev，就会调用：

```
1  if (qs) a6 = {&qs->pdfRev, pt->Pdf(scene, ptMinus, *qs)};
```

又因为 pt 是相机点，所以 pt 调用的 Pdf() 函数就是 Pdf_We()。

PerspectiveCamera::Sample_Wi() 函数中计算的概率密度 pdf 从某个点处对相机采样得到的概率密度，这个概率密度和采样面光源是一样的；PerspectiveCamera::Pdf_We() 函数计算得到的 pdfDir 是已知镜头上相机顶点的前提下，采样来找到下一个顶点的概率（角度相关的概率密度）。这两个概率密度千万不要搞混了。

6. 无限面光源



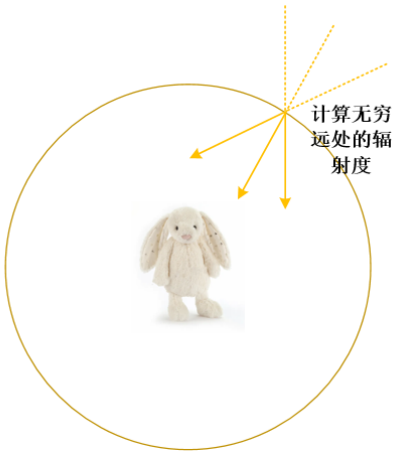
本章讲解双向路径追踪算法中如何集成和应用无限面光源。

6.1 无限面光源带来的问题

无限环境面光源定义在无穷远的距离处，因此难以直接和 BDPT 的方法协调一致。BDPT 需要要求面概率密度，因此需要有限大小的发光表面。

当然我认为使用无限面光源照明时可以不用双向方法，或者将场景中的光源分开渲染，比如无限面光源用路径追踪，其他光源用双向路径追踪方法。但是我们还是可以学习一下 PBRT 中的实现方案。

我们可以用有限的形状来表示无限面积的灯光。例如，无限面光源的辐射发射分布可以由围绕场景的大发射球体来描述，其中球体内部每个点处发射辐射度的方向分布与相同方向的无限区域光发射辐射相同。这种方法需要一个非常复杂的 InfinityAreaLight 实现，除了 BDPT 兼容性之外，没有任何实际好处：



PBRT 将在 BDPT 中将无限面光源作为一种特例，而不是更改 InfiniteAreaLight 的功能。由于无限面光源的照明最自然地集成在立体角上，因此 PBRT 的方法将为顶点抽象层添加对立体角

集成的支持。实际上，场景可能包含多个无限面光源；当估计发射辐射度或确定样本概率时，我们将把它们视为亮度组合在一起的一个光源。

6.2 PBRT 中的实现

首先，我们将在摄影机路径光线离开场景时创建一个特殊的端点顶点。RandomWalk() 在生成摄影机子路径时，只要找不到曲面交点，就会调用从摄影机片段跟踪时捕获逃逸光线。在此处调用的 Vertex::CreateLight() 方法的实现中，记录每单位立体角概率的 pdfFwd 变量直接存储在 Vertex::pdfFwd 中，无需通过 Vertex::ConvertDensity() 进行转换。

相机子路径上存在光源顶点会导致某些无意义的连接策略。例如，我们不可能将相机子路径上的无限面光源顶点连接到光源子路径上另一个顶点。ConnectBDPT() 中的以下检查检测并忽略此类连接尝试：

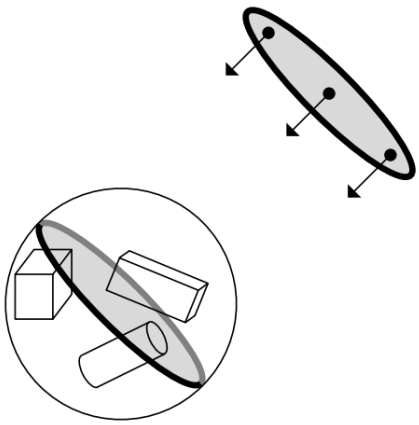
```
1 // 这种情况只可能是由于相机最终采样到无限面光源造成的
2 if (t > 1 && s != 0 && cameraVertices[t - 1].type == VertexType::Light
    )
3   return Spectrum(0.f);
```

然后看 ConvertDensity() 函数，该函数会检测是否是无限面光源，如果是，就直接返回 pdf 而不是转化为面积相关的 pdf。

对于无限面光源的采样函数 InfiniteAreaLight::Sample_Le()，我们需要一些操作来纠正它的概率密度，这里是在 GenerateLightSubpath() 函数中实现的，即如何生成光源子路径的第一个和第二个顶点：

```
1 // 为无限面光源纠正子路径采样密度
2 if (path[0].IsInfiniteLight()){
3   <为无限面光源计算path[0]的空间概率密度>
4   <为无限面光源计算path[1]的空间概率密度>
5 }
```

注意采样无限面光源的方式有些奇特，它会首先采样一个方向，然后根据场景尺寸来模拟一个圆盘，采样一个顶点位置：



Sample_Le() 函数中，计算的 pdfDir（方向相关概率密度）就是根据无限面光源的明暗分布来采样全景图得到的概率；计算的 pdfPos（位置相关概率密度）就是采样圆盘得到的概率。

当 path[0] 已经确定好采样方向了以后（无限面光源的入射方向），path[1] 的位置恰好就能够对应在圆盘上采样的位置。但是如果 path[1] 表面不是正对着光入射方向，则概率密度会多出一个 $cos(\theta)$ 项，其中 θ 就是光入射方向和 path[1] 表面法向量的夹角：

```
if (nVertices > 0) {
  path[1].pdfFwd = pdfPos;
  if (path[1].IsOnSurface())
  path[1].pdfFwd *= AbsDot(ray.d, path[1].ng());
}
```

path[0].pdfFwd 的计算会调用 InfiniteLightDensity() 函数。我们下一节再解释各种概率密度的用途。

6.3 计算光源概率密度

InfiniteLightDensity() 会根据光分布 Distribution1D 来计算采样概率密度。

给定光源入射方向，光源类的 Pdf_Li() 函数会计算概率密度。但光分布下可能有很多光源，每种光的概率密度都不同。假如我们的场景中有 5 个光源，其中三个是面光源（标记为 light[0]，light[1] 和 light[2]），两个是无限面光源（标记为 light[3] 和 light[4]），假设光分布是根据功率来计算的，设这五个光源的功率分别是 [12,14,8,30,31]，那么在计算生成光分布时，lightDistr.func[i] 保存了每个光源的功率值，lightDistr.funcInt 是所有光的总功率值。那么 InfiniteLightDensity() 返回的结果就是：

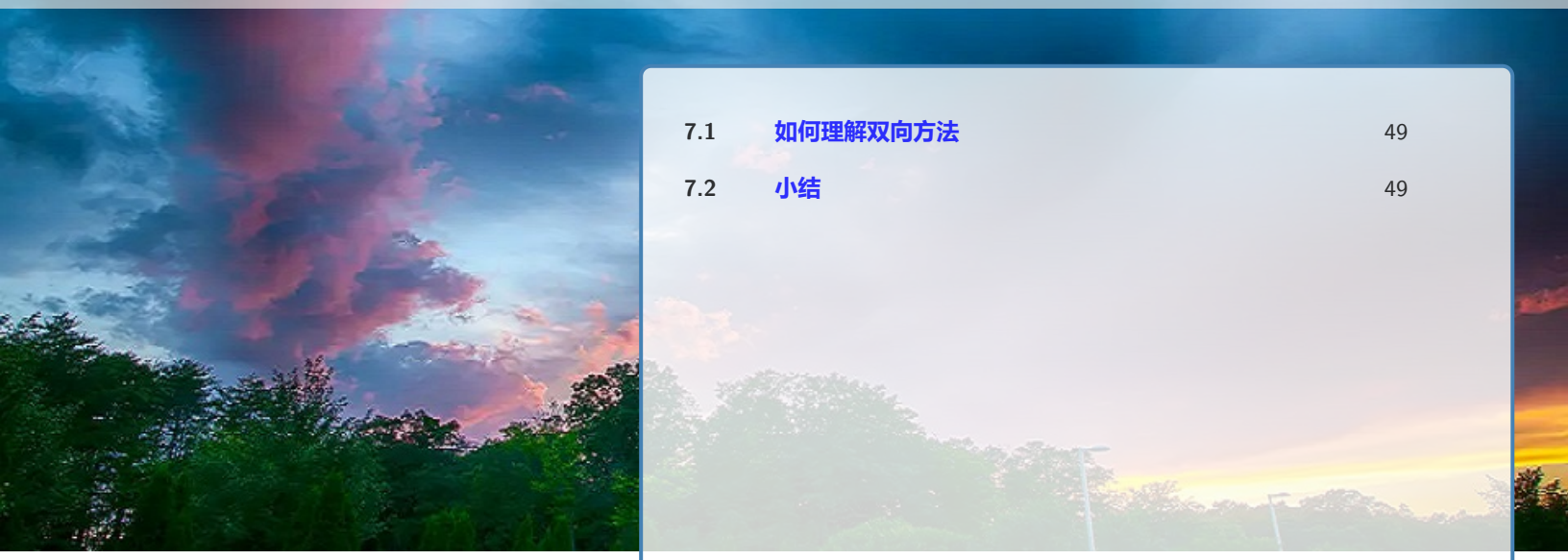
$$\frac{30\times light[4].pdf(\omega)+31\times light[4].pdf(\omega)}{(12+14+8+30+31)\times 5} \tag{6.3.1}$$

也就是“从多个光源中产生当前光线的概率”。至于为什么要把无限面光源的概率给统一，我并不认为这是某种必要的规定，我觉得可能是为了使面光源的照明是“混合”在一起而不是“叠加”在一起。因为我们计算发光 Le 时只采样了一个无限面光源的值，而不是把两个无限面光源都采样相同位置的辐射度然后把值叠加。PdfLightOrigin() 中对于无限面光源也是调用的这个函数。

而对于 PdfLight()，调用该函数时我们已经知道了选定哪个光源，而我们想要知道发出的光线的面积相关的概率密度；对于无限面光源来说，其单位面积相关的概率密度就是在整个球面上进行采样：

```
Point3f worldCenter;
Float worldRadius;
scene.WorldBound().BoundingSphere(&worldCenter, &worldRadius);
pdf = 1 / (Pi * worldRadius * worldRadius);
// v是无限面光源发出的光线相交的顶点
if (v.IsOnSurface()) pdf *= AbsDot(v.ng(), w);
```

7. 双向路径追踪总结



本章对本文做一个简单的总结和概括。

7.1 如何理解双向方法

我们在 PBRT 实现和原理中刻意避开了一些要素，比如重要性流和辐射度流之间的区别和联系。但除了一些细微的地方（尤其是非对称散射问题的理解），并不会对理解双向方法造成太多影响。

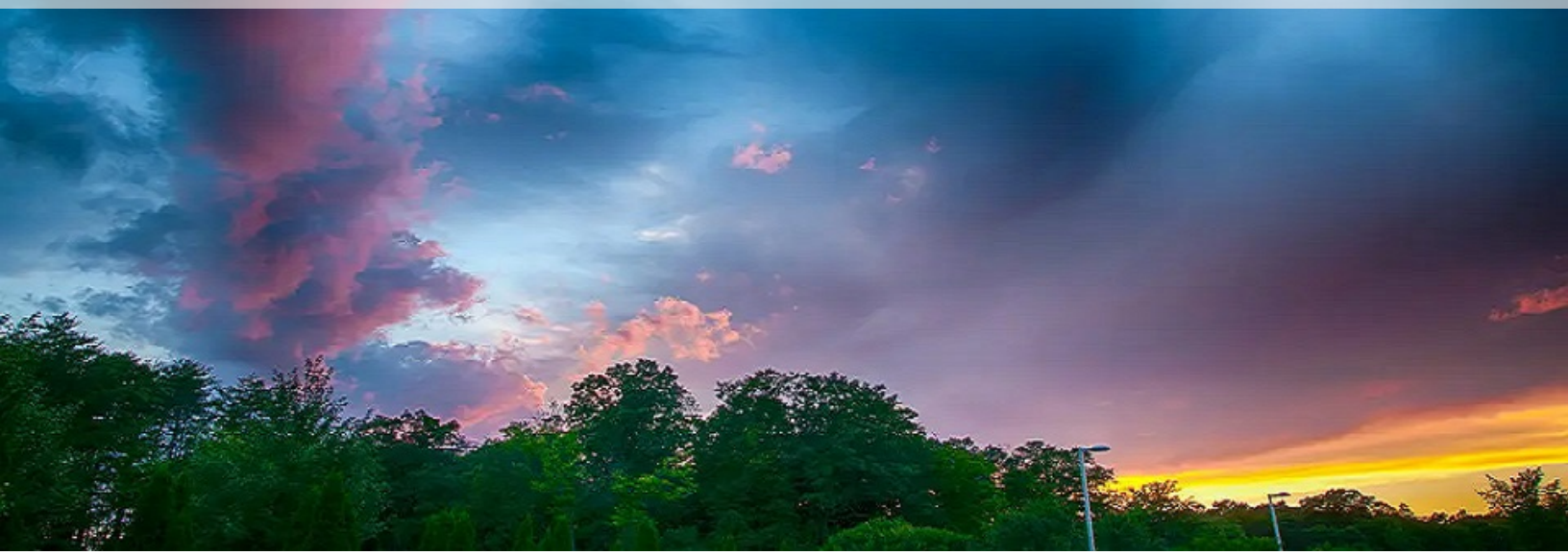
然而，想要真正理解双向方法，还需要长时间参透和领会，尤其是对于什么是辐射度量学，要反复去思考概率密度、路径积分的内容，这些都是能够更好地理解双向方法的基石（本质其实还是数学与一些工程学的应用）。

7.2 小结

本文开始写于 2022 年 8 月 20 日，途中因为很多事情，以及在写作中遇到了一些困难，导致 12 月 6 日才刚刚能完成了第一版。接近五十页的内容，也是 PBRT 系列里最长的电子书之一了，然而，它里面涉及的知识还得包括《模拟光传输的鲁棒的蒙特卡洛方法（Eric Veach）全文解读》中的很多内容。

在我刚接触双向方法的时候，由于资料非常稀少，而且仅有的一些文章和解释也都是在照抄[2]，所以遇到问题根本不知道应该请教谁，只能一点一点抠代码和写程序验证。PBRT 的实现相对来说还算比较贴合原文，所以阅读起来稍微轻松一点。

参考文献



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

[2] Veach E . Robust Monte Carlo Methods for Light Transport Simulation[J]. Ph.d.thesis Stanford University Department of Computer Science, 1998.