图形学大作业报告

计科60 王蕴韵 2016011346

功能一览

- √ 光线追踪算法：PM，PPM，SPPM
- √ 漫反射、折射、反射
- √ 球体、平面、贝塞尔曲面参数求交
- √ 贝塞尔曲面物体支持坐标变换和包围球加速
- √ kd 树加速三个算法
- √ 贴图
- √ 景深
- √ 超采样
- √ OpenMP
- √ 变换视角和对焦点的短视频

文件一览

所在文件夹	文件名	功能简介
pm/sppm/video	vec.hpp	三维向量vec3d的 struct 以及附带操作，光线 ray 的 struct 以及附带操作
pm/sppm/video	unit_test.cpp	用于各个模块的单元测试，pm和 sppm 中两个文件不同
pm/sppm/video	parameters.hpp	用于常量参数设置和场景设置，pm和 sppm 中两个文件不同
pm/sppm/video	objects.h	球体、平面、贝塞尔（仅在 sppm/video中）的类定义
pm/sppm/video	objects.cpp	球体、平面、贝塞尔（仅在 sppm/video中）的类实现
pm/sppm/video	kdtree.hpp	kd 树，用于存光子（PM）/视野点（SPPM），pm和 sppm 中两个文件不同
pm	pm.cpp	PM 算法的实现
sppm	sppm.cpp	SPPM 算法的实现

video	video.cpp	视频的实现，用了 PPM
sppm/video	colorful.ppm	用于墙壁贴图
sppm	bezier.py	网上找的开源代码，用于构建贝塞尔曲线，调试使用
sppm	matrix.hpp	矩阵 struct，含求逆操作，本来准备用于贝塞尔求交迭代，后来改了算法没有使用

总计不重复的代码大概1k2-1k3行吧。

功能位置

因为 sppm 功能最全，所以有重复的全部算在 sppm 里。

功能	所在文件	代码行数
判断光线和物体相交	sppm/sppm.cpp	12:35
反射光线	sppm/sppm.cpp	79:85
折射光线	sppm/sppm.cpp	87:102
弹光线直到一个漫反射面	sppm/sppm.cpp	109:135
算带景深的颜色	sppm/sppm.cpp	137:157
超采样+图片写入	sppm/sppm.cpp	159:172
发射像素(x,y)处的光线并保存在 kd 树	sppm/sppm.cpp	174:198
发射光子迭代	sppm/sppm.cpp	200:289
贴图	sppm/sppm.cpp	230:269
读贴的图	sppm/sppm.cpp	291:305
kd树的实现	sppm/kdtree.hpp	14:133
动态插入点	sppm/kdtree.hpp	137:143
动态更新点	sppm/kdtree.hpp	145:157
球和平面的定义	sppm/objects.h	12:63
贝塞尔曲线的实现	sppm/objects.h	65:145
贝塞尔曲面的定义	sppm/objects.h	147:156
球和平面的求交求法向实现	sppm/objects.cpp	3:28
贝塞尔曲面求交（含包围球优化）	sppm/objects.cpp	38:111

参数和场景设置	sppm/parameters.hpp	6:67
向量的各种操作	sppm/vec.hpp	6:66
OpenMP	sppm/sppm.cpp	37:77, 147:155

贝塞尔解法

贝塞尔部分我参考了翁家翌同学的大作业报告中的实现思路，但是实现细节与他不同。

贝塞尔曲线的存储与使用

由公式

$$\sum_{i=0}^{n} w_i \binom{n}{i} t^i (1-t)^{n-i}$$

可得，贝塞尔曲线可以由控制点写成关于 t 的两个 n 次多项式。

我使用了预处理组合数+二项式展开来计算多项式系数。并将贝塞尔存储为这两个多项式。构建的时间复杂度$O(n^2)$。

贝塞尔曲线的 struct 提供调用x(t), y(t), dx(t), dy(t)来获得 t 点的 x，y 值和一次导。使用多项式存储方便这里的计算，单次求值复杂度$O(n)$。

参数曲面求交

构建绕 y 轴（在视野中的竖直方向）旋转贝塞尔曲线生成的贝塞尔曲面。

设入射光线的入射点为$\vec{o}$，射线方向的单位向量为$\vec{d}$。

首先考虑$\vec{d}$的 y 不为 0 的情况。

设交点在贝塞尔曲线取 t 值的时候取到。那么有：

$$o.y + d.y \cdot r = y(t)$$

其中，r 表示光线传播的距离。

$$r = \frac{y(t) - o.y}{d.y}$$

$$intersection = (o.x + d.x \cdot r, y(t), o.z + d.z \cdot r)$$

$$dist(intersection, y_{axis}) = (o.x + d.x \cdot \frac{y(t) - o.y}{d.y})^2 + (o.z + d.z \cdot \frac{y(t) - o.y}{d.y})^2 = (x(t))^2$$

设

$$a_1 = \frac{d.x}{d.y}$$
$$a_2 = \frac{d.z}{d.y}$$
$$a_3 = o.x - \frac{o.y \cdot d.x}{d.y}$$
$$a_4 = o.z - \frac{o.y \cdot d.z}{d.y}$$

则方程变为

$$(a_1^2 + a_2^2)(y(t))^2 + (2a_1a_3 + 2a_2a_4)y(t) + (a_3^2 + a_4^2) - (x(t))^2 = 0$$

这是一个关于t的多项式，借助一阶导数可以用牛顿迭代解决。

再考虑$\vec{d}$的 y 为 0 的情况，此时入射光线垂直贝塞尔曲面的中心轴。由于$d.y$不能为除数，所以上面的方法不适用。

不过此时只要求解出$y(t) = d.y$，就可以把问题转化到$y = d.y$的平面上，变成了一个二维射线与圆求交的问题。

求解$y(t) = d.y$，仍然用牛顿迭代，可以特判一下和上面的迭代一起进行。

坐标变换和包围球加速

为了使贝塞尔物体更加灵活一点，不是只能竖直摆放，我加入了坐标变换。包括平移变换和围绕坐标轴旋转角度。

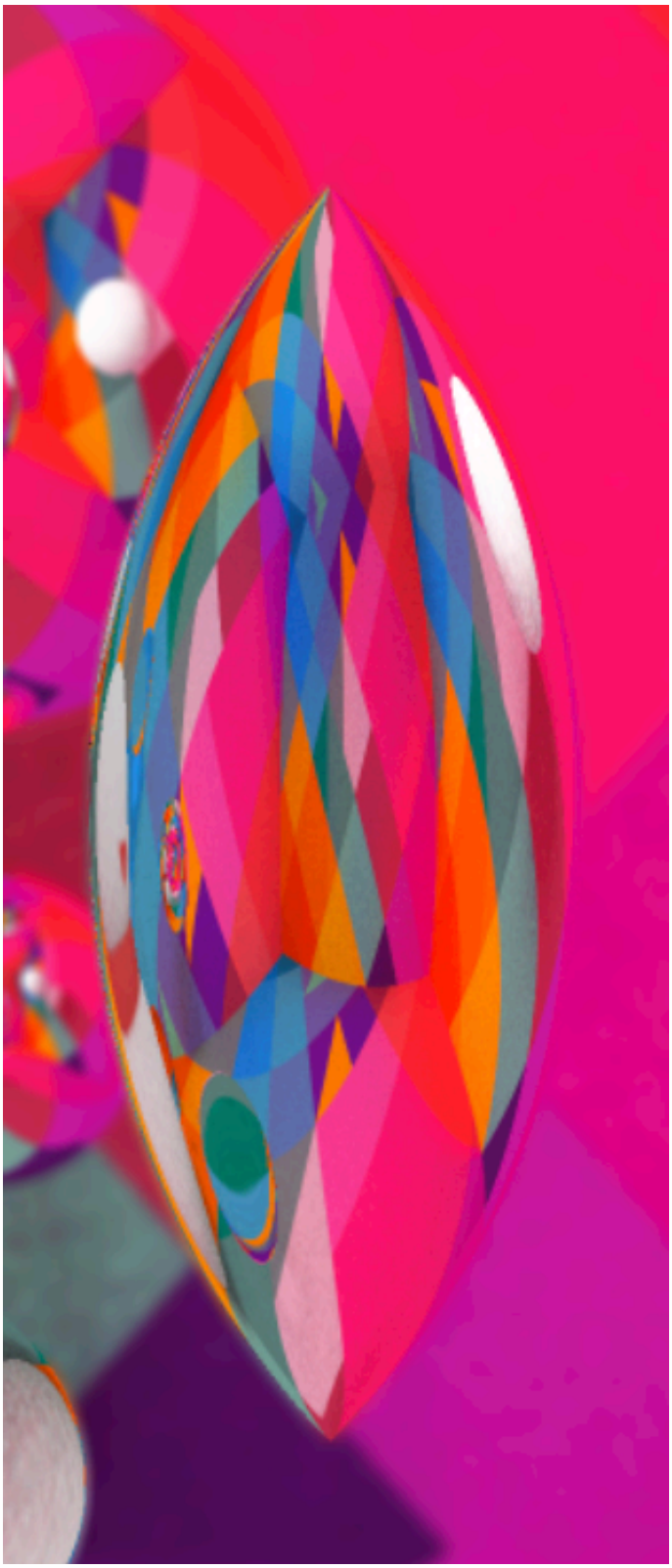
具体实现就是把入射光线先做逆变换到以贝塞尔物体的中心为坐标系，计算完光线反射后再变换回原坐标系。

包围球加速可以减少一些不必要的求交，使用一个球包住整个物体，光线先判断是否和球相交。

实现细节和实例

实现起来有很多细节需要注意，精度需要 long double 来保证，因为计算比较复杂容易丢精度。

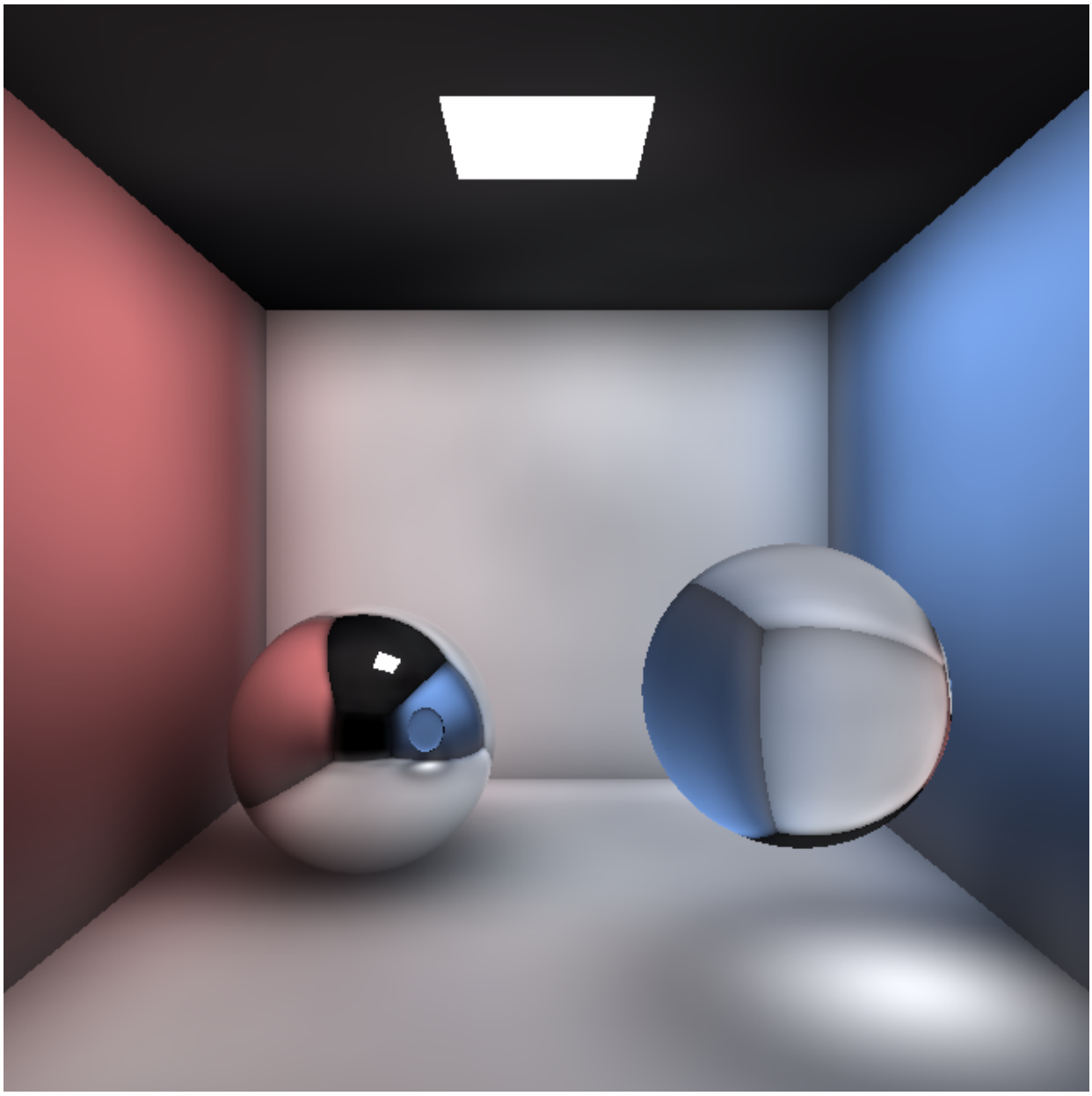
最终的结果对尖角的渲染也不会出问题，因为计算的时候避开了所有趋近于零的除数。如下图。



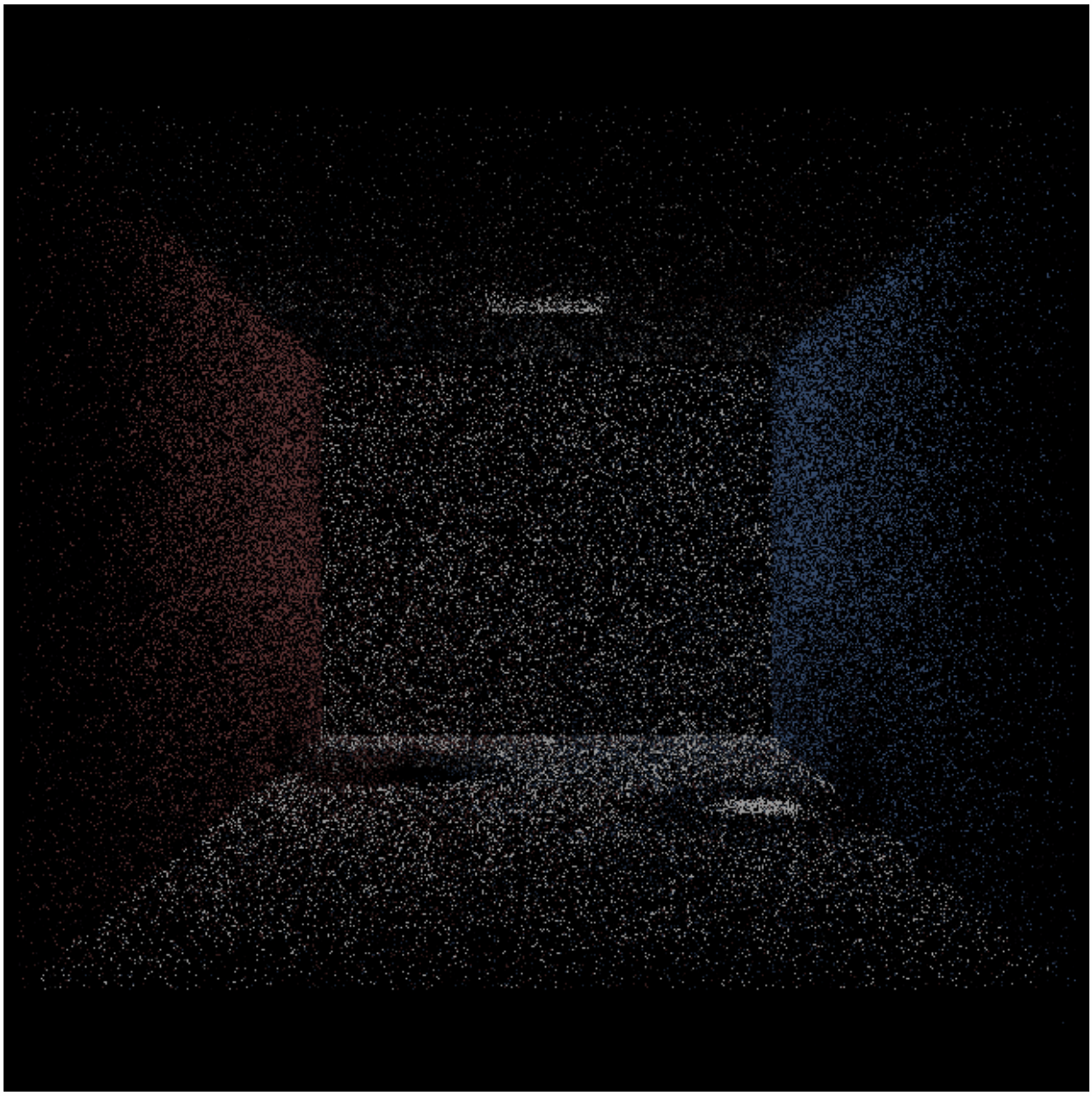
PM、PPM 和 SPPM

我最开始不清楚他们仨的特性和实现难度，就先实现了 PM。PM 和 PPM 架构差距还挺大的，PPM 和 SPPM 就很相似。

PM 是第一遍放光子构建光子图，第二遍再根据视角发出的光线的落点去搜寻某半径内的光子。这样的问题是需要大量的光子才能保证渲染的精度，而存储和查询这些光子会耗费大量的时间和空间，尤其是空间，是损耗不起的。所以在我的努力下也只能获得如下的效果图：



这是把半径取得特别大得到的比较光滑的效果，他的光子图如下：



用了4万个光子，渲染一张5s，但是局限在纯色，半径一小就惨不忍睹。想贴图等扩展的我只好重写一个 PPM。

PPM 是把PM 的思路反过来了，先发视角光线并存下来，只后一直发光子看他离哪些视角光线近就去修改他们的值。好处是解放了空间，只需要记录视角光线，空间可控，光子有时间就可以一直发。

PPM 的缺点在于光线视角恒定，在比较粗糙/模糊的表面很难渲染出平滑的效果。SPPM 在此基础上增加了对光线视角的随机扰动，使得出来的图更平滑。对于半径的收缩，我是从一个初始半径开始全局收缩，每隔5轮乘上0.9。

所以我实现了 PPM 后又把它改成了 SPPM，就变成三个都写了一遍。其实最开始我还仿照 smallpt 写了一个入门 pt 来熟悉光线运作，没有跑图，就不放了。

反射折射漫反射

反射公式：（来源于[Jensen 2001]）

$$\vec{\omega}_s = 2(\vec{\omega} \cdot \vec{n})\vec{n} - \vec{\omega}$$

折射公式：（来源于[Jensen 2001]）

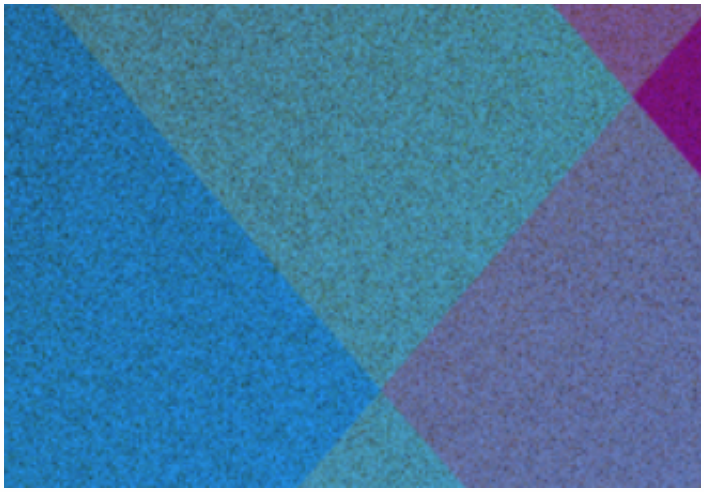
$$\vec{\omega}_r = -\frac{\eta_1}{\eta_2}(\vec{\omega} - (\vec{\omega}\cdot\vec{n})\vec{n}) - \left(1 - \left(\frac{\eta_1}{\eta_2}\right)^2 (1 - (\vec{\omega}\cdot\vec{n})^2)\right)\vec{n}$$

漫反射：

一开始我使用了均匀各个方向散光的理想漫反射模型，出来的图发现塑料感太重了，像这样，完全没有质感：

然后我开始乱尝试调整，最后发现以40%的概率直接镜面反射光出去，剩下的均匀散光可以得到一种粗糙的质感，比较满意。

注意到在渲染的时候半径也要控制合适，太小会出现这样：



感觉像砂纸，不太喜欢。调整后最后的结果图如下：

雪人的质感太难了，水平不够。

kd 树

用来查找半径 r 内的邻居，并对相关的信息修改。使用二进制分组合并实现动态加点和查询，虽然在这里似乎并用不上动态。

景深

凭借相机的使用常识，景深由光圈和对焦距离决定得最明显，我记录了每个像素点到视角的距离，根据对焦距离确定一个清晰圆面，剩下的由距离离这个平面越远越模糊，光圈越大（光圈数值越小）模糊得越厉害，模糊的窗口大小的公式是 (kFocusDist - realDist) / kAparture 。

超采样、OpenMP

超采样没什么好说的，就是四个像素当一个像素用取个平均。

OpenMP 是一个简单的并行，注意控制全局变量要分线程处理再合并。在 mac 上不知为啥还用不了，上服务器才行，吐槽一下 mac 的 g++。

视频

上服务器开多进程跑多张图，租了一个内存挺大 CPU 挺多的服务器跑了5h 跑了60张图，然后倒序再放一遍，用 FFmpeg 合成成视频。做这个是想模拟现实中的对焦过程，看到每个球轮流变清晰又变模糊很好玩，开个循环播放还可以无缝衔接。

参考资料

Smallpt: Global Illumination in 99 lines of C++ by Kevin Beason

翁家翌同学的大作业报告: https://github.com/Trinkle23897/Computational-Graphics-THU-2018/blob/master/hw2/report/report.pdf

Knaus C , Zwicker M . Progressive photon mapping[J]. ACM Transactions on Graphics, 2011, 30(3):1-13.

Jensen H W . Realistic image synthesis using photon mapping[M]. A. K. Peters, Ltd. 2001.

Hachisuka T , Jensen H W . Stochastic progressive photon mapping[J]. Acm Transactions on Graphics, 2009, 28(5):1-8.