渲染大作业实验报告

计 42 方燠 2014011322

2016 年 6 月 27 日

目录

1 主体算法实现 **2**

1.1 Photon Mapping 2

1.2 Progressive Photon Mapping 2

1.3 Probablistic Progressive Photon Mapping 3

2 特效实现 **3**

2.1 纹理贴图 (Texture) 3

2.2 复杂模型求交加速 (BVH) 4

2.3 软阴影 (Soft Shadow) 4

2.4 景深 (Depth of Field) 4

2.5 材质 (Materials) 5

2.6 焦散 (Caustics) 5

2.7 环境光贴图 (Image Based Lighting) 5

2.8 体积光（Volume Light） 5

3 实验结果及说明 **6**

3.1 Water Scene 6

3.2 Angel Scene 7

3.3 Dragon Scene 7

1 主体算法实现

1.1 Photon Mapping

Photon Mapping 是一个 93 年被提出的算法，它可以很好地模拟辉映、焦散线等特效。以下是对于光子映射算法的一个基本介绍：

基于 Photon Mapping 的全局光照算法有两步：第一步，从光源向场景发射光子，并在它们碰到非镜面物体时将它们保存在一个光子图（photon map）中，以建立光子图。第二步，使用统计技术从光子图中提取出场景中所有点的入射通量以及反射辐射能。

光子图与场景表述是完全分离开的，这一特性使得光子映射方法能处理很复杂的场景，包括千万个三角面片，实例化的几何体，复杂的过程式物体。

1.2 Progressive Photon Mapping

Progressive Photon Mapping 是对 Photon Mapping 算法的一个改进。

在基本的 Photon Mapping 中，图像好坏和它的收敛性等等，直接受到光子数目的影响。为了得到清楚的图像，我们可能需要发射成百上千甚至上亿个光子，并要将他们全部存储在光子图中。这对于算法的实现和硬件性能都要求极高。

Progressive Photon Mapping 的全局光照算法也分为两步：第一步，从相机向场景发射视线，当碰到非镜面物体时候保存在一个三维哈希网格（grid hash），我们把这些焦点称作 HPoint；第二部，从光源向场景发射光子，以每个 HPoint 作为碰撞检测球的重心，当光子碰到非镜面物体时，依次查询它周围所有的 HPoint，查看是否出现在碰撞检测球内，如果出现则改变 HPoint 点上的光通量，需要注意的是，碰撞检测球的半径会随着碰撞次数减小。

需要注意的是，由于视线较为有限，对于每个反射面和折射面我们都需要分别递归计算出 HPoint，把他们全部存储在三维哈希网格中。Progressive Photon Mapping 的好处在于光子如何和 HPoint 碰撞，可以直接对光通量造成影响，而不需要把这些光子依次存储下来。

因而，我们可以使用 Progressive Photon Mapping 算法，计算上十亿甚至上百亿光子对整张图片造成的影响，而不需要消耗大量的硬件资源。

1.3 Probablistic Progressive Photon Mapping

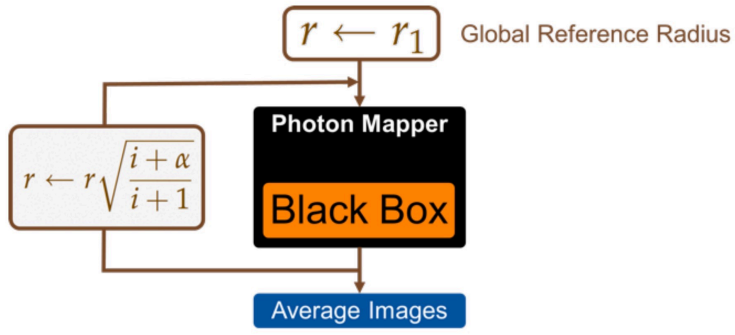
不难发现，Progressive Photon Mapping 算法存有一定的局限性，也就是它的视线是固定的。想要记录下所有的 HPoint，只需要对每个像素发射一轮视线。

考虑如下情形：场景非常复杂，某个局部细节很多，相邻的像素所显示的图像可能相差甚远，那么直接采用 Progressive Photon Mapping 就可能产生局部的高频噪声，这是非常糟糕的。

此外，固定的视线，导致景深等特效无法实现。Probablistic Progressive Photon Mapping 的思想非常简单，发射多轮视线，并且在每一轮后发射若干光子。每一轮都能够生成一张图片，最后对于所有的图片取平均。在该算法中，相当于把 Progressive Photon Mapping 算法当成一个 Black Box 使用。

但是需要改进的一点是，在 Probablistic Progressive Photon Mapping 算法中，每个 HPoint 不在具有独立的碰撞检测球半径，而使用一个全局的碰撞检测球半径进行替代，在每轮发射视线以后，统一缩小这个全局的半径，使得最终图收敛。

算法的大致流程如下图所示：



Can be implemented via scripting (and indeed done with pbrt)!

2 特效实现

2.1 纹理贴图 (Texture)

主要实现对各种图片文件的读写，就可以方便地实现纹理贴图。

我使用了之前用到过的 stb 的一系列库，可以很方便地读写图片，通过

对 unsigned char 数组进行操作即可。

2.2 复杂模型求交加速 (BVH)

复杂模型的实现较为常规，主要需要能够读取.obj 文件和使用数据结构来加速求交。需要注意的是，所用的数据结构不是 kd-tree，而应该使用 bvh 来完成。

kd-tree 操作的基本元素是一个点，依次对每一维度进行划分，并且将一堆点分成均匀的两堆；而 bvh 操作的基本元素是三角面片，对于一堆三角面片，首先需要求出它的长方体包围盒，然后对最长的一维进行划分，将一堆三角面片分成两堆，然后递归进行操作。

由于三角形面片求交加速在整个算法中，可以说是最低层的东西，实现的好坏直接影响了整个算法的效率，在具体的实验过程中，我参考了 NVIDIA 相关求交算法的实现。使得整张图能在一个较短的时间内收敛。

2.3 软阴影 (Soft Shadow)

软阴影的实现较为简单，在向外发射光子的时候，我们改变光子的出射位置即可。比如光子出现的位置均匀分布在一个平面上，就能很好地模拟面光源的效果，从而产生软阴影。

由于是面光源，在阴影的边缘附近，会出现光源被部分遮挡的情况，从而产生较亮的阴影。

2.4 景深 (Depth of Field)

景深实际上是模拟了相机的光圈，模拟光线经过镜头前的一个凸透镜后，光路发生改变的情况。

在具体实现中，我们需要确定光圈的大小和焦距。我们知道凸透镜成像时，同一点发出的光会汇聚在焦平面上，由于光路是可逆的，也就是说最终某个像素所发射的视线只会在焦平面处汇聚，而对于场景中其他距离的部分，则会变得模糊。

因而，我们只需要在整个凸透镜上均匀采样，发射视线并通过凸透镜的成像特性，计算出视线穿过凸透镜后的出射位置和方向即可。

2.5 材质 (Materials)

由于光子映射这一类算法难以实现 brdf，例如金属材质（glossy）可以很方便地在光线追踪中实现，但是用光子映射写起来就比较复杂。

在大作业中我使用了近似的方法。对于物体的表面，按比率混合漫反射，全反射和折射的系数，达到模拟各种不同材质的效果。

2.6 焦散 (Caustics)

焦散是光子映射实现最方便的特效，由于凹凸透明物品，会对穿过其表面的光线产生汇聚和发散作用。

通过简单的二维正弦函数叠加可以产生水波的效果，我写了一个小脚本来生成水波的网格，并用它实现了焦散特效。

2.7 环境光贴图 (Image Based Lighting)

环境光贴图（Image Based Lighting）是真实感渲染中的常用技术，能够极大地提升所渲染场景的真实感。

实现起来比较简单，主要使用 HDR 贴图，可以视作将整个场景包围在一个巨大的球形贴图中，当光子和视线射向无穷远处时，分别进行特殊处理即可。

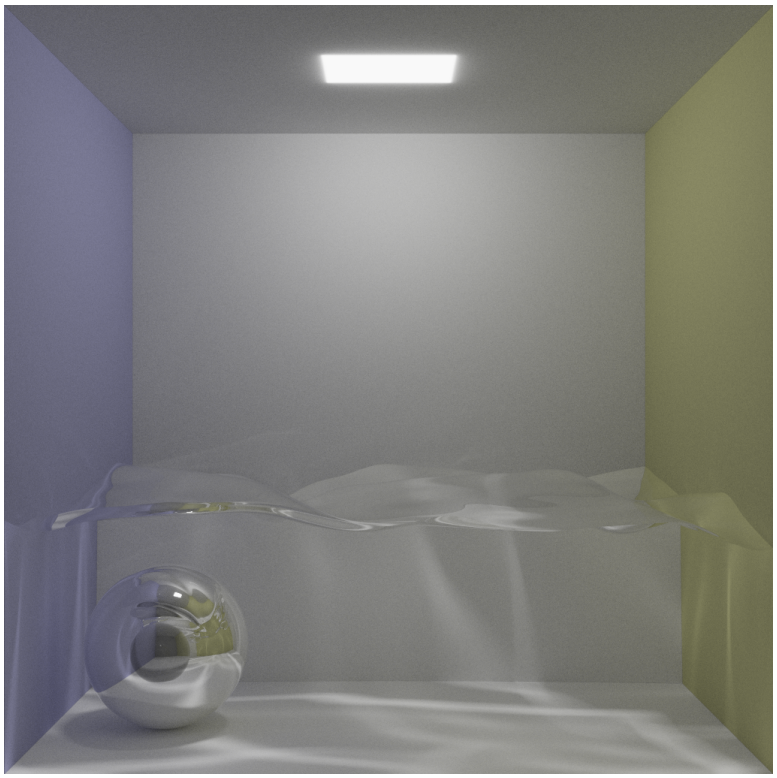
2.8 体积光（Volume Light）

体积光对于提升真实感也有很大的帮助。在光子映射算法中，光路只会在物体表面发生改变，因而无法观察到光子穿过空间而产生的轨迹。

在参考相关论文以后，我们知道光子在空间中停下的概率是以指数级变化的的，因而体积光的渲染，分为如下这么几步：首先在计算视线的时候，通过重要性采样，在空间中留下 HPoint；在发射光子的过程中，根据光子每次运动的距离，按照对数函数计算其停下的概率，当停在空间中时，对周围的 HPoint 产生影响。

3 实验结果及说明

3.1 Water Scene



主要通过光子映射相关算法实现焦散，通过三角函数模拟水波。

3.2 Angel Scene

右边的墙壁是纹理贴图和模拟的玻璃的材质，中间是一个较为复杂的模型，实现了体积光的渲染。

3.3 Dragon Scene

龙是模拟的金银铜三种金属材质，地面是模拟的大理石材质，将反射和大理石纹理按比率混合，使用环境光贴图，整个场景被一个房间包围，此外

还实现了景深的效果。