

基于光线追踪算法的真实感图像渲染器

计 93 王哲凡 2019011200

2021 年 6 月 27 日

目录

| | |
|--------------------------|----------|
| 1 项目结构 | 2 |
| 2 算法选型 | 3 |
| 2.1 路径追踪 | 3 |
| 2.2 光子图构建 | 3 |
| 2.3 图片输出 | 3 |
| 2.4 SPPM 的优势 | 3 |
| 3 光线追踪加速 | 5 |
| 3.1 KD-Tree 加速 | 5 |
| 3.2 OpenMP 加速 | 5 |
| 4 参数曲面求交 | 6 |
| 5 复杂网络与场景 | 7 |
| 6 其他效果 | 8 |
| 6.1 软阴影 | 8 |
| 6.2 抗锯齿 | 8 |
| 6.3 景深 | 8 |
| 6.4 贴图 | 8 |
| 6.5 重心插值 | 9 |
| 6.6 体积光 | 10 |

1 项目结构

主要沿用 PA1 的框架，通过提供 run_all.sh 脚本即可进行模型的渲染。

通过给定参数可以选择渲染的输入文件，如 scene01.txt；也可修改 run_all.sh 中的参数适应不同输入文件。

除此之外的文件：

- deps/：与 PA1 基本相同，进行了一定函数的添加。
- include/：包括主要实现的各种类或者函数，部分实现放在了对应的.cpp 文件中。
- mesh/：各种可能使用到的 3D 物体模型。
- output/：主要的渲染成果图片。
- src/：对应于 include 中的部分复杂功能实现以及主程序 main.cpp。
- testcases/：构建的各种场景输入文件。
- texture/：各种用于贴图等的图片。

2 算法选型

本次实验我主要采用了 SPPM 即随机渐进式光子映射。

光子映射（PM）算法主要分为两个步骤，分别是光子图的构建以及通过传统路径追踪进行辐射率的计算。

而渐进式光子映射（PPM）则改为了先进性路径追踪再通过光子图构建更新，并且将光子图的构建拆分为多轮取平均使得算法近乎必定可以完全收敛。

随机光子映射（SPPM）则是将两个步骤均改为了多轮，使得模型的存储等问题得到了更好的解决。

2.1 路径追踪

基本与路径追踪（PT）算法一致，即从相机向每个像素点发出射线，对于击中的点根据击中物体的材质进行俄罗斯赌徒随机，选择漫反射/镜面反射/折射等，并实时更新光的权重等信息。

在一轮中，对于每个像素点，分别进行一次迭代，直到深度超过限制或者被漫反射面吸收为止。

通过这样的算法我们可以构建出不超过像素点个数个击中点，这些击中点容易通过 KD-Tree 进行维护，进而可以较为方便地进行 k 近邻的查询，更新等。

2.2 光子图构建

这个步骤类似于从光源出发的路径追踪，通过光源出发的射线进行同样的随机模拟，最终也会落在某个漫反射面上。

这样我们可以通过对在其某个范围内的光子进行统一更新权重来实现模拟光线与物体相交的过程，我们可以利用上一步建立的 KD-Tree 对于之前的击中点进行快速的更新。

而为了保证收敛的可靠性，我们还需要将每次击中点的估计半径不断缩减，这也可以通过 KD-Tree 部分完成。

2.3 图片输出

最后我们对于每个像素点会有一个累计的击中点统计数据，通过对于其记录的光通量等信息我们可以计算出最终的辐射率，即可用于当前的图片输出。

2.4 SPPM 的优势

首先相对于 PM，SPPM 通过多轮的方式，将渲染的过程变得更容易可视化以及容易进行迭代更新。

而相对于 PPM，SPPM 是更加符合工业需求的算法，一来其统计像素内的所有可视区域更符合直觉，二来其只须为每个像素点开辟空间存储，而无需考虑像素点所有可能的击中点（单轮只有一个击中点，多轮可重复使用）。

最后，相比于传统的单向路径追踪算法（PT），SPPM 改进了它在多镜面反射、折射的环境下，漫反射面难以很好地追踪反射、折射强光的问题，也就是难以呈现焦散现象。

下图即为一个焦散效果的展示：

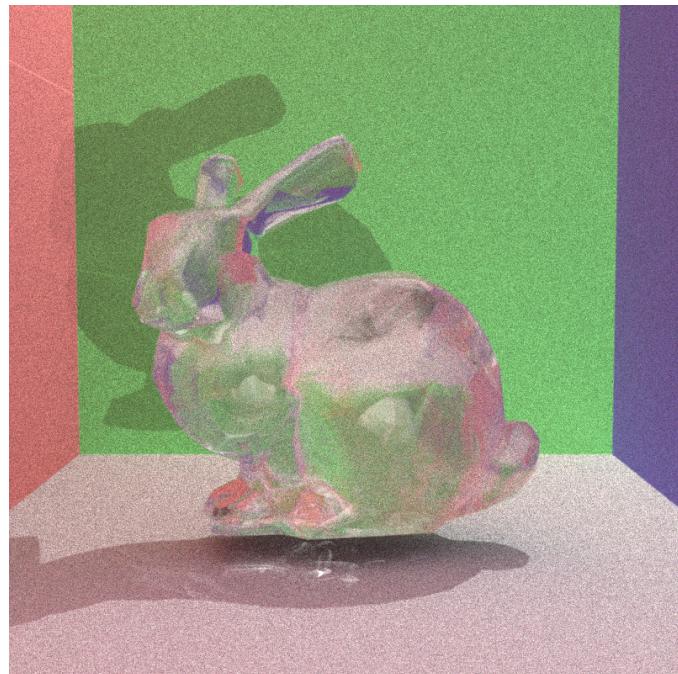


图 1: SPPM 渲染下的焦散效果

3 光线追踪加速

主要通过算法和硬件两个角度进行了加速。

3.1 KD-Tree 加速

实现了对于三角面片组成物体的求交加速，通过将三角面片进行对应坐标的划分，将其构成树状结构。

在需要求交时，首先对于左右子树构成的包围盒进行求交测试，只有当其与包围盒存在交点并且小于当前最近距离时才进行递归，并且优先进行包围盒距离更近的一边子树的递归。

具体实现见 objectkdtree.hpp/cpp。

3.2 OpenMP 加速

由于 SPPM 每轮的两个步骤分别是对于像素点和光源进行的，其每次 PT 都有极强的并行性。

因此我对于光子和像素进行了多线程并行加速，具体采用了 OpenMP 的 API，在多核 CPU 的主机上，得到了较好的性能表现提升。

4 参数曲面求交

主要通过牛顿迭代法实现了参数曲面与射线的解析求交。

在具体的过程中，我们可以通过 μ 和 θ 来表示在参数曲面上的点 $\mathbf{f}(\mu, \theta)$ ，其中 μ 表示参数曲线的自变量， θ 表示转角；我们还可以通过 t 表示光线的交点即 $\mathbf{p} = t \cdot \mathbf{d} + \mathbf{o}$ ，其中 \mathbf{d}, \mathbf{o} 分别表示射线的方向和源点。

我们可以通过求解 $|(\mathbf{p} - \mathbf{f}(\mu, \theta))|^2$ 的最小值来判断求交结果与焦点位置。

特别值得注意的是， μ 的范围应被控制在参数曲线的合理范围内，如 Bezier 曲线的合理范围为 $[0, 1]$ ，而 B 样条曲线则与其 t 数组取值相关。

最终得到的效果可以参考如下：



图 2: 参数曲面求交效果

5 复杂网络与场景

主要利用了往届学长的场景模型，进行了一个客厅场景的绘制，总共包含了 58 万个三角面片。
效果可见下图：



图 3: 复杂场景-客厅渲染效果

6 其他效果

6.1 软阴影

SPPM 算法自带软阴影。

6.2 抗锯齿

由于 SPPM 第一步会对于像素点进行重复的光线追踪，因此通过对其光线添加一个横纵坐标各 ± 1 的随机扰动可以更好地解决像素点过渡不平滑的问题，即抗锯齿。

6.3 景深

通过添加焦距 focus 和光圈半径 aperture 可以使得相机聚焦在焦平面附近的物体，而使得较远的物体变得模糊。

具体实现上，可以通过在相机坐标下，固定像素在焦平面上位置，偏移相机位置，进而获得偏移射线完成。

具体的景深的效果可见下图：

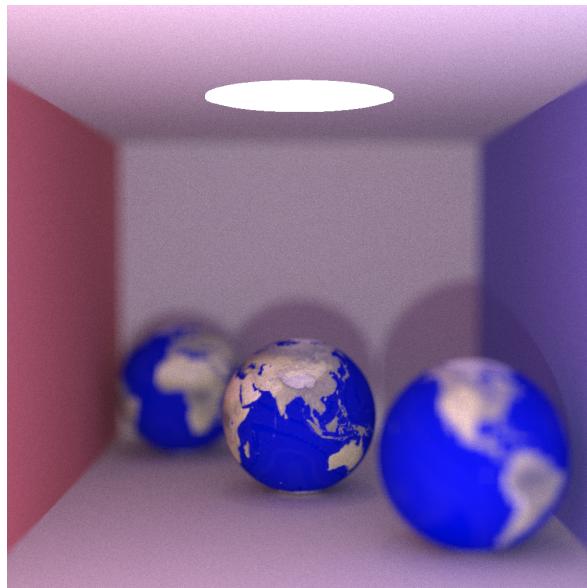


图 4: 景深效果

可以看到，通过设置焦距在中间的球上，使得只有中间的球没有模糊。

具体实现可参见 camera.hpp。

6.4 贴图

通过实现像素点到图片平面的 $\mathbb{R}^3 \rightarrow \mathbb{R}^2$ 映射，即可完成基本的贴图。

对于参数曲面的贴图效果可参见图 2，另一种球面的贴图参考如下：

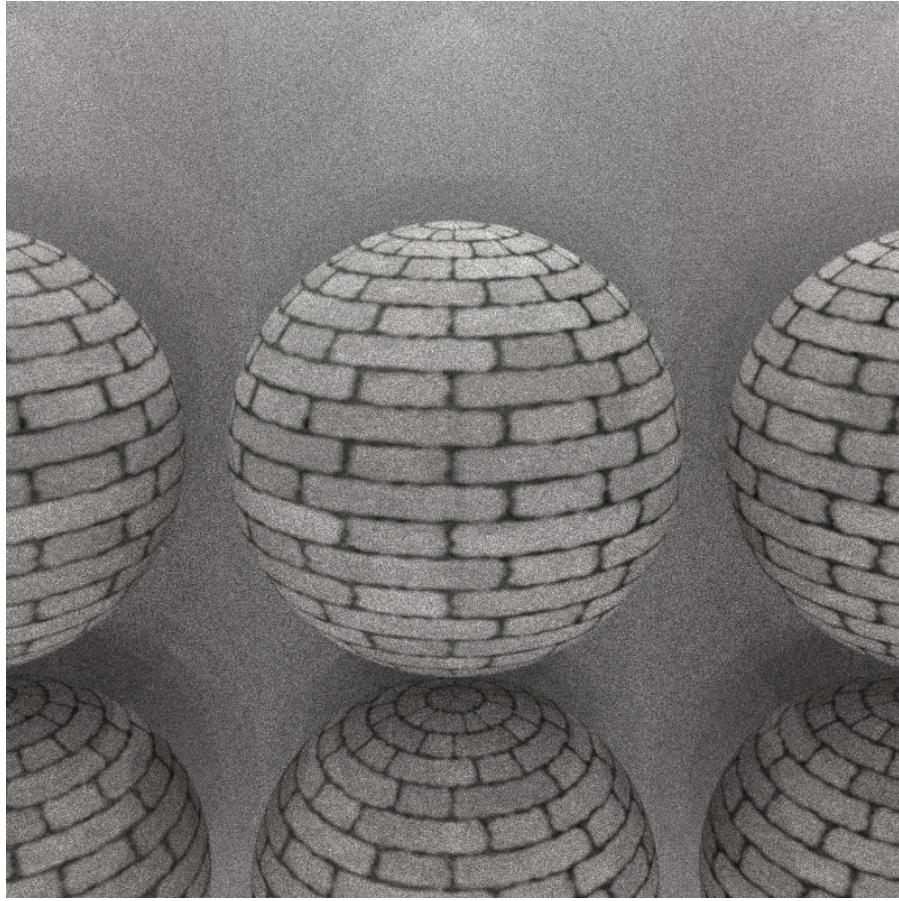


图 5: 球面贴图

尝试实现了凹凸贴图效果，但在 SPPM 算法运行下效果不佳（可参考上图）。
具体实现位于 `texture.hpp/cpp`。

6.5 重心插值

对于部分物体模型，由于提供了在每个三角面片每个点上的法向量值，在进行求交法向量计算时，可以通过重心插值拟合更真实的物体模型信息。

设三角形三个点的坐标分别为 $\mathbf{A}, \mathbf{B}, \mathbf{C}$ ，其内一点坐标为 \mathbf{P} ，则可得到其分割的三个三角形面积分别为：

$$S_1 = \frac{|(\mathbf{B} - \mathbf{P}) \times (\mathbf{C} - \mathbf{P})|}{2}, S_2 = \frac{|(\mathbf{C} - \mathbf{P}) \times (\mathbf{A} - \mathbf{P})|}{2}, S_3 = \frac{|(\mathbf{A} - \mathbf{P}) \times (\mathbf{B} - \mathbf{P})|}{2}$$

若三个点对应的法向量为 $\mathbf{n}_A, \mathbf{n}_B, \mathbf{n}_C$ ，则 \mathbf{P} 点的法向量可以估计为：

$$\mathbf{n}_P = \frac{S_1 \mathbf{n}_A + S_2 \mathbf{n}_B + S_3 \mathbf{n}_C}{S_1 + S_2 + S_3}$$

具体实现可参见 triangle.hpp。

6.6 体积光

通过设置媒介，以及对于经过媒介的光线的强度、方向处理，可以近似模拟实际在大气中传播的光线。

没有体积光的效果如下：

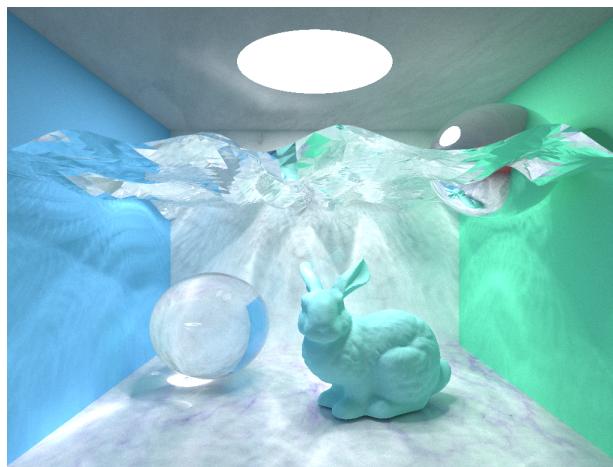


图 6: 无体积光场景

低光强使用体积光效果如下：

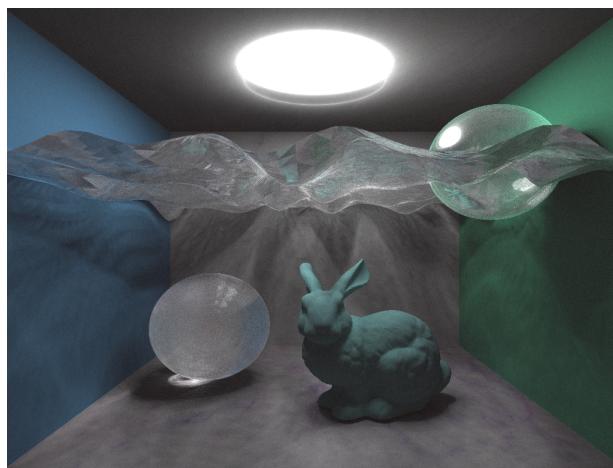


图 7: 使用体积光场景

可以看到反而在弱光源下，光源附近出现了光晕。