

Realistic Image Rendering

计 73 郑林楷 2017011474

2019 年 6 月

目录

1 代码整体框架	2
2 Bézier 曲线造型与曲面建模	2
2.1 Bézier 曲面造型	2
2.2 Bézier 曲面建模	2
3 真实感场景渲染	3
3.1 渲染算法	3
3.1.1 Path Tracing	3
3.1.2 Progressive Photon Mapping	3
3.1.3 Stochastic Progressive Photon Mapping	3
3.2 渲染特效	4
3.3 渲染加速	4
3.3.1 包围盒	4
3.3.2 K-D Tree	5
3.3.3 OpenMP	5

1 代码整体框架

- `bezier.hpp` 实现了 Bézier 曲线类，包括求曲线上的点、一阶导数等功能；
- `face.hpp` 网格面类，包括求交和加载 `obj` 文件等功能；
- `hitpoint.hpp` 碰撞点类，以及实现了存放碰撞点的 K-D Tree；
- `kdtree.hpp` 实现了存放网格面的 K-D Tree；
- `main.cpp` 主程序，包含相机参数和主算法框架，以及初始化、景深效果、超采样等功能；
- `object.hpp` 物件类，定义了平面、球、旋转 Bézier 曲面和网格；
- `polynome.hpp` 多项式类，实现了多项式四则运算、求解和一阶求导的功能；
- `ray.hpp` 光线类，包含起点和方向；
- `render.hpp` 实现了光线追踪的渲染功能；
- `scene.hpp` 定义场景以及物件；
- `texture.hpp` 纹理类，包含纹理文件、颜色和材质；
- `utils.hpp` 包含基本的一些定义；
- `vec3.hpp` 三维向量类，包含向量四则运算、叉积、点乘等运算。

图片读入使用的是 `stb`¹。图片输出有两种方式，一是使用 `ppm` 格式输出，二是使用 OpenCV 输出成 `png` 格式。

2 Bézier 曲线造型与曲面建模

2.1 Bézier 曲面造型

首先我们可以很显然地知道 Bézier 参数曲线 $Q(f, t)$ 本质上就是两个关于 t 的 n 次多项式，关于证明详情可见于翁家翌同学的图形学实验报告²。

我们可以在 $O(n^2)$ 的时间内预处理出多项式的各个系数，这样便可以在 $O(n)$ 的时间内计算 $Q(f, t)$ 的值以及其一阶导数。

2.2 Bézier 曲面建模

代码中实现的是旋转体 Bézier 曲面。

设参数直线 L 上一点 P_k 到旋转轴的距离为 $D(k)$ ，高度为 $H(k)$ ，那么求直线与 Bézier 曲面交点相当于求解：

$$\begin{cases} D(k) = x(t) \\ H(k) = y(t) \end{cases}$$

¹<https://github.com/nothings/stb>

²<https://github.com/Trinkle23897/Computational-Graphics-THU-2018/blob/master/hw2/report/report.pdf>

其中 $x(t)$ 和 $y(t)$ 是二维 Bézier 参数曲线上参数 t 所对应的点的坐标。

我们可以从 $D(k) = x(t)$ 解得 $k = f(t)$ 然后代入 $H(k) = y(t)$ 求解 t ，本质上是在求解一个 $2n$ 次的多项式的零点。代码中采用了牛顿迭代法进行多项式求零点。

3 真实感场景渲染

3.1 渲染算法

3.1.1 Path Tracing

Path Tracing[1] 是光线追踪的改良。Path Tracing 在碰到漫反射面时随机选择一个反射方向继续追踪，直到碰到光源。也正因为如此，Path Tracing 收敛极慢。

效果如图 1 所示。



图 1: Path Tracing 效果图

3.1.2 Progressive Photon Mapping

PPM[2] 分别从相机和光源一起释放出射线，两边的射线在某个漫反射面上相聚，最后进行统计得到渲染图像。

若干视线从相机出发，经过 eye tracing 落在漫反射面上形成视点。同时，光源随机发出若干光子，经过反射、折射和随机漫反射后也会若干次与漫反射面相碰撞，并对周围「送光」，即光子对一定范围内的所有视点有一定的光强贡献。视点所在之处获得的光强以及颜色就决定了视线所对应的像素点的颜色。

代码实现上对视点使用 KD-Tree 进行储存，以便后续「送光」的计算。

3.1.3 Stochastic Progressive Photon Mapping

SPPM[3] 是 PPM 的改良版。与 PPM 不同在于，SPPM 在每一轮光子发射完毕后重新进行一次 eye tracing，并且每次 eye tracing 都带上一个随机的扰动，让每次追踪留下

的视点都不一样。这样虽然会降低渲染效率，但随机扰动使得渲染得更加真实。

效果如图 2 所示。



图 2: Stochastic Progressive Photon Mapping 效果图

3.2 渲染特效

代码中实现了软阴影、超采样抗锯齿、纹理贴图和景深等功能。

软阴影 设置光源为面光源。

超采样抗锯齿 对于每个像素点，细分成 4 个子像素点并分别进行光线追踪，结果取这 4 个子像素点的平均值。

纹理贴图 将物体表面的三维坐标经过某种变换映射到二维平面，即可对应到纹理贴图上的一像素点，该像素点的颜色即为物体表面的颜色。效果如图 2 所示。

景深 光线追踪时，对每条射线保持经过焦点不动，并在射线的端点加上微小扰动，扰动的幅度由光圈大小决定，以此模拟带焦距的相机模型。效果如图 3 所示。

3.3 渲染加速

3.3.1 包围盒

在实现光线与旋转 Bézier 曲面求交时，如果发现光线不会与旋转 Bézier 曲面的包围盒相交，则可知道其与旋转 Bézier 曲面也不会相交。

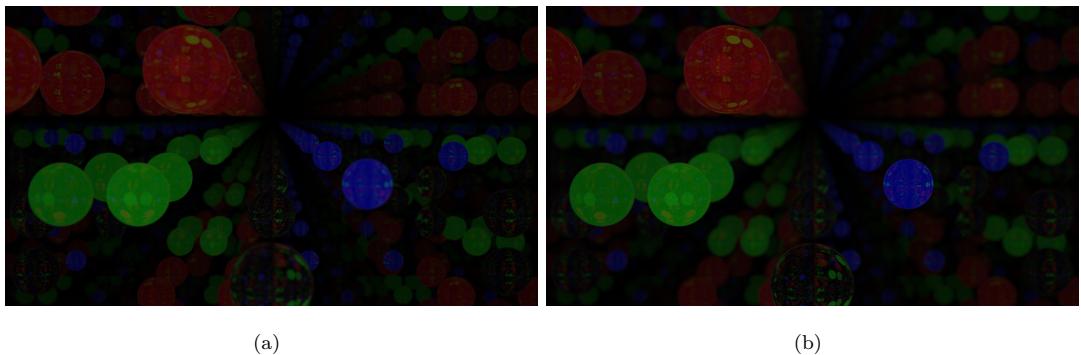


图 3: 景深效果图。

六面镜子围成一立方体，立方体内有三个自发光荧光球和一个玻璃球。

(a) 焦点对准镜后玻璃球的像；(b) 焦点对准镜前玻璃球。

3.3.2 K-D Tree

K-D Tree 主要用于 PPM/SPPM 算法寻找碰撞点以及网格的快速求交，单次查询的均摊复杂度为 $O(\log n)$ ，其中 n 是 K-D Tree 内点数。

3.3.3 OpenMP

使用 OpenMP 进行多线程渲染能大大提升渲染速率。PT 算法可按一行像素为单位分给各个线程进行渲染。PPM/SPPM 算法可按光子为单位分给各个线程进行光子映射。

参考文献

- [1] Kevin Beason. smallpt: Global illumination in 99 lines of c++. <http://www.kevinbeason.com/smallpt/>. Accessed: 2019-06-30.
- [2] Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. Progressive photon mapping. In *ACM Transactions on Graphics (TOG)*, volume 27, page 130. ACM, 2008.
- [3] Toshiya Hachisuka and Henrik Wann Jensen. Stochastic progressive photon mapping. In *ACM Transactions on Graphics (TOG)*, volume 28, page 141. ACM, 2009.