

Realistic Rendering

张立博

2021012487

zhanglb21@mails.tsinghua.edu.cn

2023.6.18

摘要

基于路径追踪和渐进式光子映射算法，实现了一个真实感渲染器。支持的效果包括折射，反射，阴影，参数曲面，求交加速，漫反射，景深，软阴影，抗锯齿，纹理贴图，法向插值以及复杂场景。本文依次介绍了不同功能的实现方法，以及实现过程中遇到的问题和解决方法。最后给出反思和总结。

1 引言

该项目基于清华大学 2023 春计算机图形学基础课程框架完成。笔者在实现过程中查阅了很多论文，课程，仓库，博客等相关资料，在此表示感谢。同时，笔者也将在实现过程中遇到的问题和解决方法结合框架代码较为详细地记录在此，希望能够帮助到后来的同学。

2 功能实现

2.1 算法选型

基础知识

1. 反射定律和折射定律公式
2. 菲涅尔效应：随着入射角角度的变化，光线在物体表面的反射和折射比例会随之变化。在计算折射光线时通过计算菲涅尔方程的 Schlick 近似实现菲涅尔效应可以使渲染结果更加真实
3. 材质类型：三维向量，分别表示漫反射，镜面反射，折射的比例

2.1.1 路径追踪

背景

光线投射：从相机发出光线，与场景中的物体求交，遍历光源利用 Phong 模型计算着色，只考虑了直接光照，没有考虑间接光照。

光线追踪：从相机发出光线，与场景中的物体求交，如果交点为漫反射材质，则停止追踪，计算着色，如果交点为反射或折射材质，则递归追踪反射或折射光线，最终累加直接光照和间接光照的贡献。这样虽然考虑了间接光照，但没有考虑光线的漫反射。

路径追踪：在漫反射表面继续追踪直至击中光源，考虑了光线的漫反射，镜面反射，折射，可以实现全局光照的效果，最后根据渲染方程计算每一个像素点的颜色。

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega} f_r(x, \omega_i, \omega_o) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i$$

光源：“图形学认为，图像颜色的源头它是什么？它是光源。光源这个词有秘密，它真有源吗？如有！”

在光线投射光线追踪的代码框架中，光源是无形的，比如点光源和方向光；但在路径追踪算法中，判定击中无形的光源或对无形的光源进行采样是不现实的，所以需要将光源转化为有形发光的物体，比如球体，盒子等。

在实际实现的过程中，可以为材质加入自发光项 emission，将发光的球体或三角

网格组成的区域作为光源。

实现

1. 对于每个像素点，从相机发出多根光线进行追踪（蒙特卡洛方法，抗锯齿）
2. 对于每根光线，当递归深度大于最大深度，用俄罗斯轮盘赌算法决定是否终止追踪（避免无限递归，防止能量损失）
 - (a) 光线与场景中的物体求交，如果没有交点，返回背景颜色
 - (b) 如果相交，计算下一条光线的方向，递归追踪，最终累加直接光照和间接光照的贡献
3. 取平均值作为该像素点的颜色

效果

下图为采样 5000 次渲染的基础路径追踪效果，测例选自 smallpt，可以看出实现了漫反射，镜面反射，折射，抗锯齿，软阴影等效果。

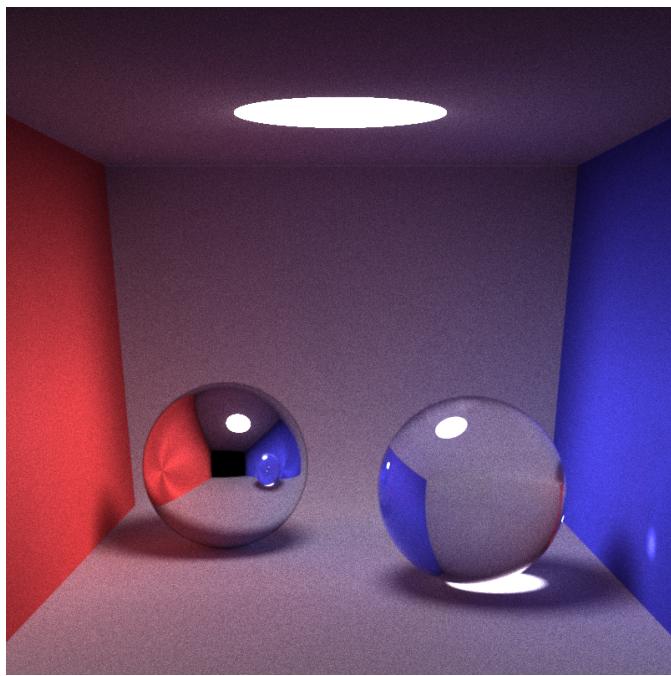


图 1: 路径追踪效果图

2.1.2 光子映射

背景

路径追踪算法难以实现焦散的效果，而光子映射算法可以较为容易地实现焦散的效果。

实现

pm（光子映射）是先从光源发射大量光子形成光子图，然后追踪视线，当视线击中物体时，从光子图中搜索最近的光子，计算光子的贡献，最终累加得到该像素点的颜色。

上述方法的缺点是光子图的大小与光子数目成正比，当光子数目较大时，搜索光子的时间开销较大。

ppm（渐进式光子映射）和 sppm（随机渐进式光子映射）交换了追踪视线和追踪光子的顺序。第一步进行视线追踪，当视线与漫反射物体相交时，记录并存储可见点信息，用 kd-tree 或 hash 表等数据结构维护；第二步进行光子追踪，当光子与漫反射物体相交时，查询可见点，更新可见点的光通量，半径等信息。最终根据可见点的信息计算每个像素点的颜色。

ppm 进行一轮视线追踪，多轮光子追踪； sppm 在此基础上进行多轮视线追踪和光子追踪，考虑了一个区域而不是一个点的光子信息，优势是可以更有效地渲染分布式光线追踪效果。

笔者尝试实现了其中的 ppm，由于时间有限，只进行了基础测例的渲染。（代码位于 sppm.hpp，由于一开始尝试实现的是 sppm，名称未改）
效果

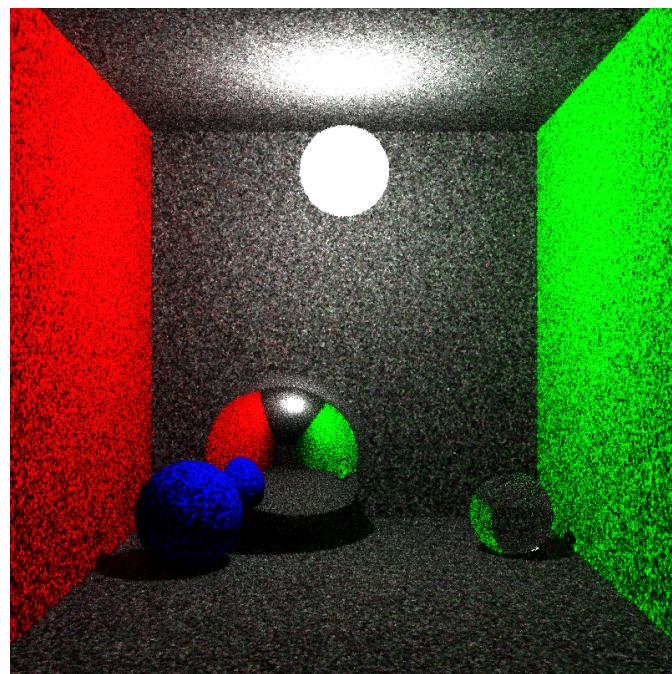


图 2: ppm 效果图

2.2 参数曲面

数值求交

使用牛顿迭代法求光线与参数曲面的交点，通过简化可以得到一个只含有变量 t 的方程 $f(t) = 0$ 。

具体过程如下：

1. 计算迭代初值：光线与参数曲面的包围盒求交，取离交点最近的曲线离散点的 t 作为迭代初值

2. 迭代求解：根据牛顿迭代法，迭代公式为 $t_{n+1} = t_n - \frac{f(t_n)}{f'(t_n)}$
3. 当 $f(t)$ 近似为 0 时，判断交点是否满足要求，若满足则设置交点信息，否则继续迭代直至迭代次数达到最大值

效果图如下：

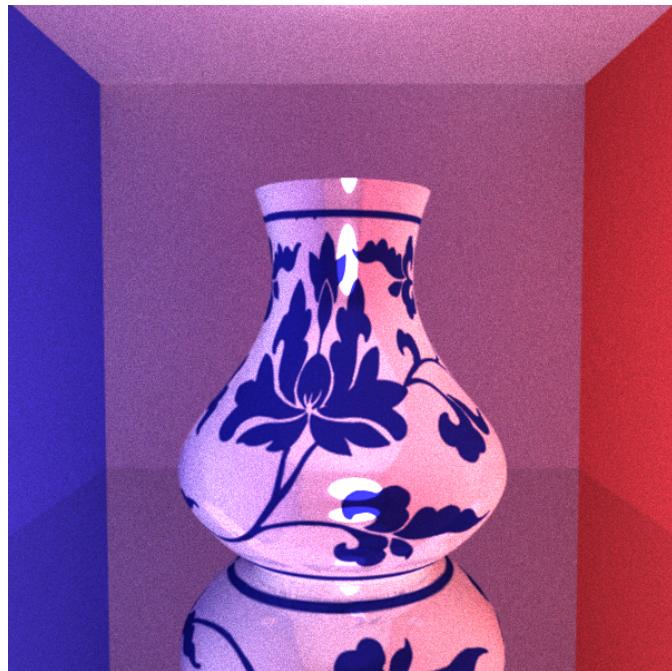


图 3: 参数曲面效果图

2.3 求交加速

实现的求交加速主要包括三个方面：包围盒，BVH 树，多线程。

2.3.1 包围盒

对于每个物体计算其 AABB 包围盒，进行求交时先判断光线与包围盒是否相交，若不相交则不需要进行求交判断，若相交则进行求交判断。

2.3.2 BVH 树

BVH 树是一种二叉树，每个节点对应一个包围盒，叶子节点对应一个物体。具体实现中 BVH 的应用有两层，第一层将场景中所有物体组织为一个 BVH 树，第二层将复杂物体（如三角形网格）组织为一个 BVH 树。求交时，先判断光线与根节点的包围盒是否相交，若不相交则不需要进行求交判断，若相交则递归判断左右子树。

2.3.3 多线程

由于渲染中很多操作是相互独立的，所以可以使用多线程进行加速。具体实现中使用 openmp 库进行多线程加速，对于每个像素点的超采样或不同像素点的光线追踪，使用 openmp 的 parallel for 指令进行加速。笔者电脑为 12 核，加入多线程后渲染时间大约为原来的 $1/12$ 。同时笔者还尝试租用服务器进行渲染，但免费的服务器配置较低，仅为单核，效果远不如本地渲染。如果可以租用到多核服务器，可以进一步加速渲染。

2.4 景深

原始的透视相机生成的光线都是从相机位置发出的，这样每一个像素都与场景中的某个点一一对应，渲染结果会很清晰，但缺少景深效果。

为了实现景深效果，需要在透视相机 PerspectiveCamera 的基础上加入两个参数，分别是焦距和光圈大小。

焦距是相机到焦平面的距离，光圈大小是相机镜头的大小。

在相机生成光线时，对于每个像素：

1. 在以光圈为直径的圆盘上随机采样，将光线的新发射点设为相机初始位置加上采样点
2. 找到相机与像素连线和焦平面的交点
3. 将新发射点与焦平面交点连线作为光线的方向

光圈越大，景深效果越明显。

效果图如下：



图 4：未带景深

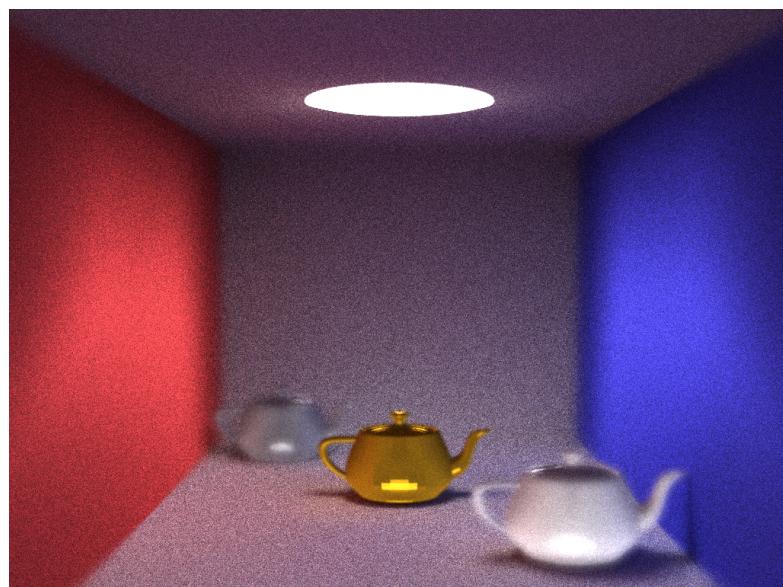


图 5：附带景深

可以清晰地看到，在附带景深的效果图中，位于焦平面的黄铜茶壶清晰可见，而

位于焦平面之前的陶瓷茶壶和位于焦平面之后的金属茶壶模糊不清。
而在未带景深的效果图中，三个茶壶均清晰可见。

2.5 复杂场景

2.5.1 复杂三角形网格

基础的三角形网格只包含顶点信息 $v(x, y, z)$ 和基础面片索引 $f(v_1, v_2, v_3)$

复杂的网格场景还包含了顶点法向信息 $vn(x, y, z)$ 和纹理信息 $vt(u, v)$ ，此时面片索引一般为 $f(v_1/vn_1/vt_1, v_2/vn_2/vt_2, v_3/vn_3/vt_3)$

笔者实现了读取复杂网格的功能，具体实现方法为判断面片索引中是否包含 / 符号，若包含则同时记录法向索引和纹理索引，然后根据索引信息通过重心插值计算交点的法向和纹理坐标。

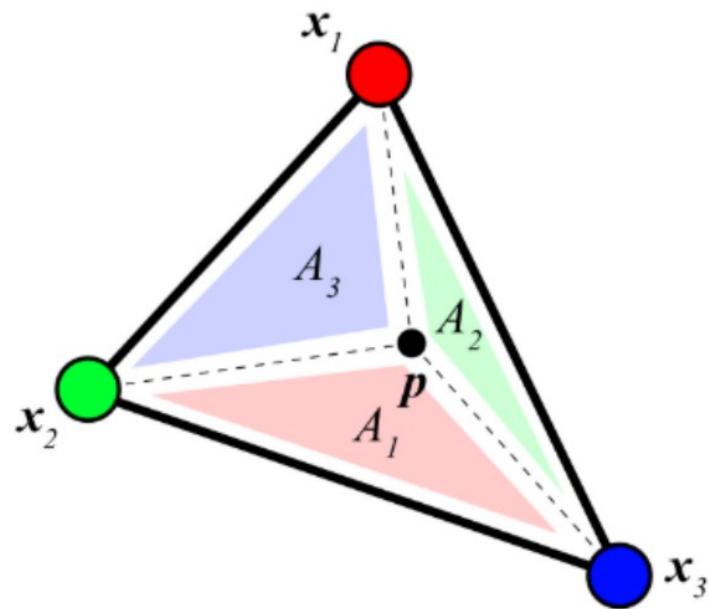


图 6: 重心插值示意图

2.5.2 复杂网格资源获取

1. GitHub: 如common-3d-test-models, 可以找到很多常见免费的.obj 模型
2. Unity: Unity Asset Store 中有很多免费的模型, 可以导出为.obj 格式。建议选择 lowpoly 模型, 减少渲染时间

Armadillo		.zip	.obj	Stanford	dblp
Beast		.zip	.obj	Autodesk	dblp
Beetle		missing*(quad mesh?)	.obj, common alt .obj	Ivan Sutherland	missing
Cheburashka		.zip	.obj	Ilya Baran(?)	dblp
Cow		.zip	.obj	Viewpoint Animation Engineering / Sun Microsystems	dblp
Fandisk		.m (Hoppe Mesh format)	.obj	CAD part Pratt & Whitney/Hughes Hoppe	dblp

图 7: 模型资源示意图

2.5.3 复杂场景搭建工具

1. VsCode 插件: 3D Viewer for VSCode, 可以直接在 VsCode 中预览.obj 模型, 并且可以调整模型的大小, 位移和旋转, 有助于搭建自己的复杂场景
2. smallpt 附带构图: smallpt 附带了一些复杂的测例 (复杂构图网址), 可以进行参考和改进

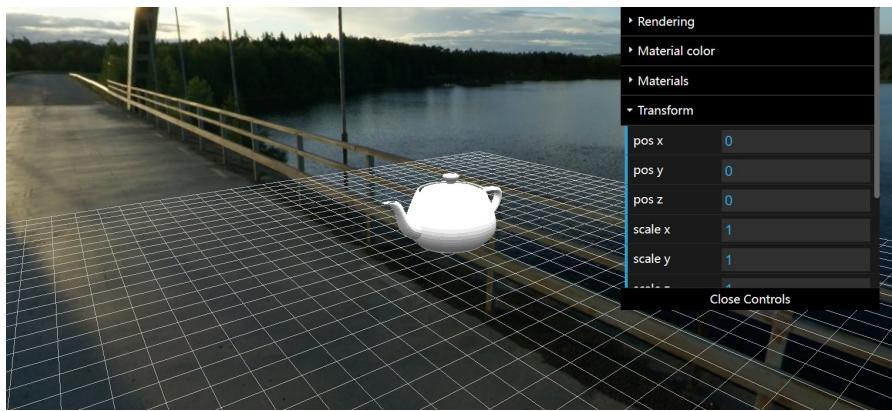


图 8: 插件效果示意图

当然，最直接的获取复杂网格资源和场景构图的方式是参考往年同学的测例，但弊端是可能会因为光源，材质等属性定义不同无法直接使用。

复杂场景效果图如下：



图 9: 夜心钻

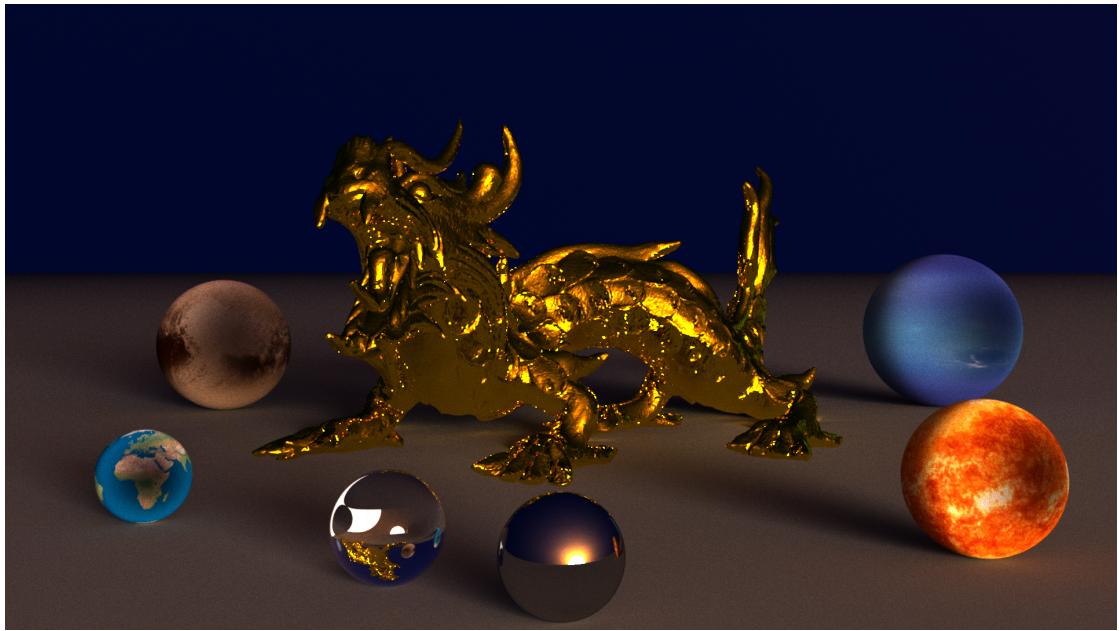


图 10: 天火龙与星界球

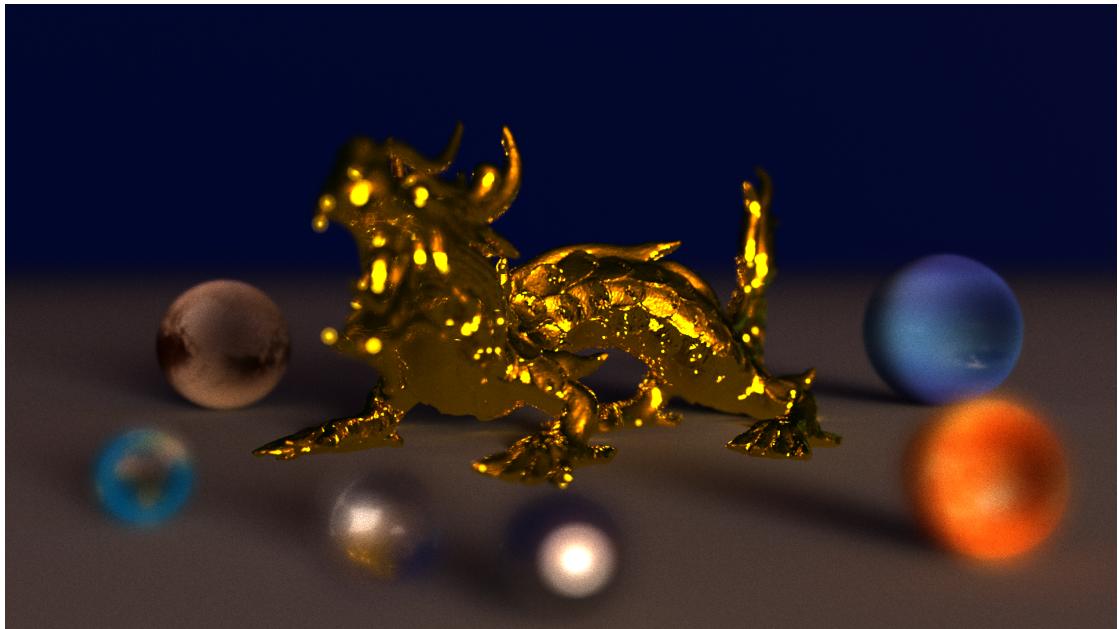


图 11: 天火龙与星界球-带景深

图 10 和图 11 中的巨龙模型由十几万个三角面片构成, 图片分辨率为 1920×1080 。

2.6 纹理贴图

纹理贴图的实现主要是将交点的三维坐标映射到二维坐标 (u,v) ，然后根据 (u,v) 在纹理图中的颜色代替物体本身的颜色。

对于一般的平面，球体和参数曲面，可以根据对应的公式直接计算 (u,v) 。

对于复杂的三角形网格，可以直接指定每个顶点的纹理坐标，然后通过重心插值计算交点的纹理坐标。

2.7 法向插值

当渲染复杂多边形网格时，每个顶点关联一条法线，在相交时通过插值计算交点的法线。这样获得的法线相比未插值结果更加光滑，渲染效果更加真实。

纹理贴图和法向插值效果图如下：

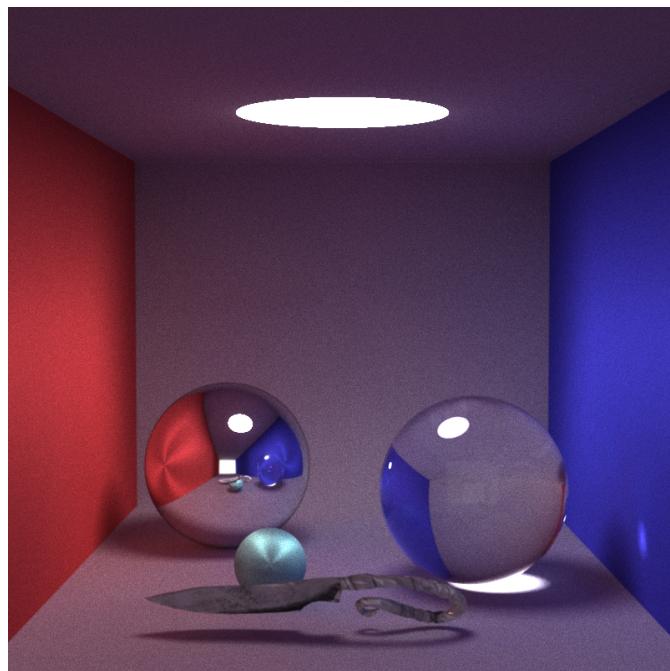


图 12: 纹理贴图和法向插值

场景中的刀模型在计算法向和纹理坐标时使用了 vn 和 vt 信息进行重心插值，可以看到刀的表面光滑且带有纹理。

2.8 抗锯齿

对于每个像素点，从相机发射多条光线进行采样，采样时加入随机扰动，最后取平均值作为该像素点的颜色。

2.9 软阴影

路径追踪和光子映射系列算法自带软阴影效果。

2.10 其他

2.10.1 框架数据类型问题

助教提供的框架中向量元素均采用 float 类型，但在实现过程中发现 float 类型在某些场景下精度不够，导致渲染结果出现了误差。

如下图：

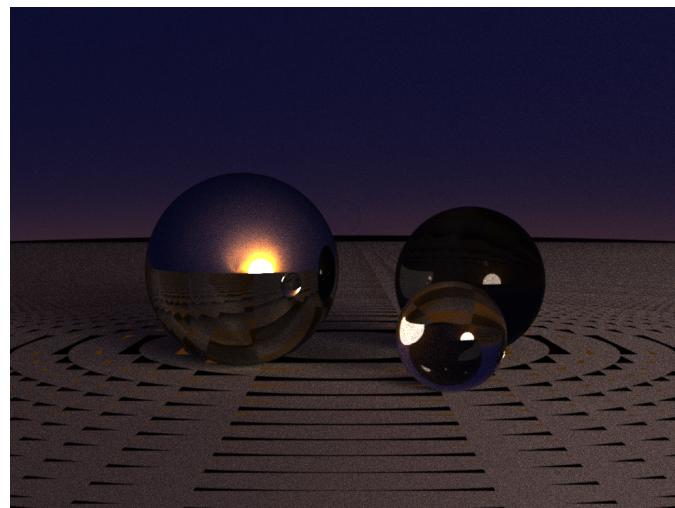


图 13: float 精度问题

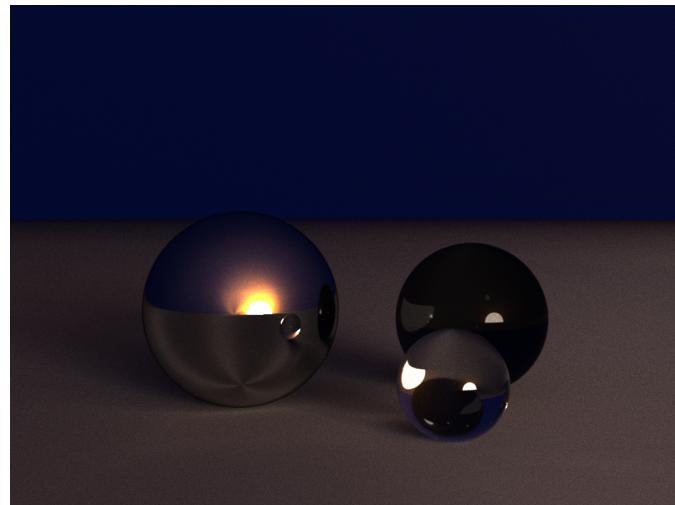


图 14: 修改后的结果

图 13 中的地面向一个巨大的球体组成，但由于 float 精度问题，四舍五入后地面出现了明显的棱角。

修改后图 14 中的地面向一个平面组成，避免了上述问题，但遗憾的是无法体现出水平面和天空交界处的光晕效果。

同时笔者注意到如 smallpt 等项目中向量元素均采用 double 类型，这样的精度会更高，可以避免上述问题。

但由于时间有限，笔者未对框架代码中的这一部分进行修改，仅作为发现的问题

记录在此。

3 反思和建议

3.1 反思

3.1.1 理解原理，由易到难

本次大作业前后做了接近 1 个月，但前期进展十分缓慢，主要原因是在刚开始时我盲目自信地直接选择实现光子映射系列的算法选型，在查阅了诸多论文和之前同学仓库后实现的效果并不理想，于是只能先将其搁置，转而实现路径追踪算法。

在完成大作业的过程中由于我很多原理理解的不是很清楚，于是只能再去查阅课件与 B 站上 Games101 的视频进行理解，期间又花费了很多功夫。

于是告诫自己和建议看到这篇报告的同学，在完成大作业乃至将来的学习中，首先要尽可能理解基本原理，从易到难，不要好高骛远，盲目自信。

3.1.2 阅读经典代码

在完成大作业的过程中，我参考了很多相关项目，但大多数代码结构混乱，注释不清，对自己的帮助十分有限。

最后发现很多图形学渲染相关的经典代码，如 smallpt, smallpaint, raytracing in one week 等结构清晰，注释详细，对于理解原理和实现方法有很大的帮助。

其中强烈推荐samllpaint，其包括了路径追踪，光子映射，体积光等算法的 C++ 实现，每种实现仅为一个.cpp 文件，阅读起来十分流畅，还附带了一个.exe 程序帮助观察不同算法不同参数下的效果。（可惜我发现得较晚...）

3.1.3 弥补数理基础

与其余计算机相关课程不同，我认为计算机图形学基础这门课主要的难点是数学物理的知识，比如牛顿迭代法，菲涅尔方程，蒙特卡洛方法等等。

在完成大作业的过程中，我发现自己的数学物理基础较为薄弱，为代码的实现增添了较多障碍。

3.2 建议

3.2.1 提供更加有效的参考资料

在完成大作业的过程中，相信很多同学和我一样，花费了大量的时间在查阅资料上，但很多资料并不是很有效，甚至有些资料是错误的。

所以希望助教和老师能够提供更加有效的参考资料帮助同学们完成大作业，比如更多的经典论文和代码链接，一些有用的技巧等等。

3.2.2 鼓励同学们交流

在完成大作业的过程中，同学间的交流可以帮助大家更好地解决个人遇到的问题，节省时间完成更多的效果。

由于我是非信息学院的学生，所以在大作业的完成过程中几乎没有和同学交流，这是我认为的一个遗憾。在树洞上看到有的同学对于多线程的讨论我才发现可以使用多线程进行加速，这样的一次简单“交流”就使我后续的渲染时间节省为原来的 1/10。

4 参考文献

笔者在完成大作业的过程中参考了很多相关资料，在此列出部分比较重要的参考文献，向这些作者表示感谢。

1. smallpt
2. samllpaint
3. Stochastic Progressive Photon Mapping, Toshiya Hachisuka, 2009

4. Progressive Photon Mapping, Toshiya Hachisuka, 2008
5. THUComputerGraphics
6. THU-Computer-Graphics-2020