Path Tracer 程序说明文档

1、简介

本文是 Path Tracer 程序的说明文档，主要介绍该程序的特点和使用方法。该程序主要参照《Physically Based Rendering, Third Edition》来实现，采用逆向光线追踪和蒙特卡洛方法对 Rendering Equation 进行计算，使用 OpenMP 库来进行并行绘制加速。

2、程序运行说明：

压缩包中给了源代码和可执行文件，可以选择直接运行或者重新编译。

2.1 从源代码编译

若要重新编译，最好用 macOS 下的 XCode 打开工程文件，在 header search path 里添加 Eigen3 库的位置，如果要并行计算的话需要配置好 OpenMP，然后即可编译。若在 Windows 下需要手动创建工程并添加所有文件和 Eigen 库，可能会有编码格式的问题需要手动调整。

2.2 程序运行

在 macOS 系统中，进入 execute 文件夹，在 shell 中运行以下命令即可直接运行程序：

```
./PathTracer
```

可以使用默认的 config 文件，直接运行就可以得到 cbox 场景的结果。

可以对 config 文件进行配置，第一行 Scene x 表示绘制第 x 个场景，x 只能为 1，2 或者 3，分别对应 cbox、veach 和自己的场景；第二行 Resolution 后面的两个数字为生成的图像的 height 和 weight;第三行 SampleCount 和第四行 MaxDepth 分别为每个像素点的采样数量和光线的最大反射次数。结果存储在 result/img.ppm。

result 文件夹下的其他 ppm 文件是已经渲染好的一些示例图片。

3、源文件说明

文件名	说明
main.cpp	主函数
CameraModel/Camera.h CameraModel/Camera.cpp	相机类，包含相机参数和光线生成的方法
CameraModel/Ray.h CameraModel/Ray.cpp	光线类，包含光线的参数和光线求交的方法
CameraModel/Intersection.h CameraModel/Intersection.cpp	包含光线和物体交点信息的类
Material/Material.h Material/Material.cpp	材质类，包含对材质采样的方法
Material/BSDF.h Material/BSDF.cpp	BSDF 类，一个 BSDF 包含若干个 BRDF 和 BTDF
Material/BxDF.h Material/BxDF.cpp	BxDF 类，包含各种 BRDF 和 BTDF，以及针对它们的采样方法
Scene/Scene.h Scene/Scene.cpp	场景类，包含初始化场景的方法，存储场景中的各种参数和元素
Scene/Shape.h Scene/Shape.cpp	包含各种形状的类，以及针对它们的光线求交方法和采样方法
Scene/Light.h Scene/Light.cpp	包含各种光源的类，以及对光照的计算方法和光源采样的方法
Scene/Example.h Scene/Example.cpp	示例场景的描述，目前可用的是 cbox 和 veach 两个场景
Utils/Sampler.h Utils/Sampler.cpp	包含各种采样器的类，以及它们的实现
Utils/Transform.h Utils/Transform.cpp	矩阵变换的类，主要用于视图和模型变换
Utils/FileParser.h Utils/FileParser.cpp	解析 config 文件的类

Core/OBJ_Loader.h	读取 OBJ 文件的库，是别人的开源代码
Core/typeAlias.h	定义了一些类型的别名
Core/globalConstants.h	定义了一些全局变量
Core/sampling.h	定义了一些常用的采样方法
Core/operation.h	定义了一些数学上的函数

4、程序的功能和结构

4.1 主要工作

- 直接光照采样，采用多重重要性采样（MIS）
- 针对多种 BRDF 采用不同的的重要性采样策略

4.2 程序实现的功能

该程序的算法步骤，括号的内容是程序中对应的类或者变量的名称：

（1） 建立场景(Scene)，需要的信息如下：：

①绘制参数：每个像素点的采样数量(sampleCount)，光线的最大深度(maxDepth)

②相机(Camera)：相机的视点（target），相机位置(origin)，相机的 up 向量(up)

③光源(Light)：光源的数量、位置、类型。目前支持的光源类型是点光源和面光源，面光源需要附着在场景中的一个 Shape 上。

④物体(Shape)：目前支持的类型有球形（Sphere）、三角形（Triangle）和矩形（Rectangle），对于每种形状需要给出不同的参数。

⑤表面材质(Material)：对于场景中的所有物体都需要给出表面材质，其中主要是给出 BSDF，一个 BSDF 由若干个 BRDF 和 BTDF 组成，目前支持的 BxDF 有 Constant Reflection、Lambertian、Specular Reflection、Specular Transmission、Fresnel Specular、Phong Specular、分别对应于均匀漫反射、完全镜面反射、完全透射、菲涅尔材质、冯氏高光。BSDF 的种类有 Phong 和 Fresnel 两种，前者包括 4 种 BRDF(Constant Reflection、Lambertian、

Specular Reflection)和 1 个 BTDF(Specular Transmission)，后者只包含 Fresnel Specular 一种 BRDF。

（2） 生成光线：

根据 Camera 的参数，sampleCount 和相平面上的点来生成光线，使用 Stratified Sampler 生成相平面上的采样点，初始化所有光线的 Radiance 为 (0.0, 0.0, 0.0)，对应 RGB 三个通道，以及当前交点的权重 beta 为 1.0。

（3） 并行地计算一条光线与场景中物体的交点，并进行如下判断：

①若没有交点，则返回该光线的 Radiance，转到（4）。

②若该交点是与光源的交点，进行判断：

若这是光线的第一个交点或者这条光线是由 Specular 表面反射而来，则 Radiance 加上光源在该点的亮度，再处理直接光照和表面反射。

若不是上述情况，则忽略光源的属性，当成普通表面来处理。

③如果达到了光线的最大反射次数(maxDepth)，则转到（4）

③对于交点是普通表面的情况，先使用多重重要性采样（MIS）对直接光照进行采样，再使用单重要性采样对表面 BSDF 的方向ω_i和值 f 进行采样。

④根据③中的结果，设置出射光的起点和方向，令 beta*=f，回到①。

（4） 待该像素点的所有 sample ray 都完成之后，计算所有光线的平均 Radiance 作为该像素的颜色。

（5） 生成图像，对存储的所有像素点的颜色值进行 gamma 校正，之后转换为 3 通道 0~255 的整数值，存储为 ppm 格式的 RGB 图像。

5、绘制结果：

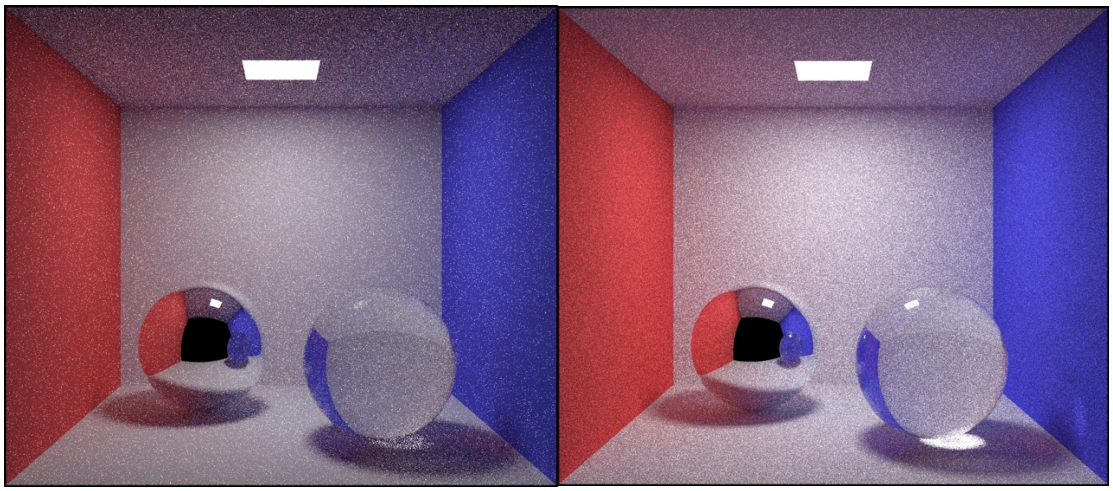
绘制结果的原图在 result 文件夹下以“场景名称-是否采用 MIS-每个像素的 sample 数-光线最大反射次数-以秒为单位的绘制时间.ppm”命名，这里只贴出部分结果。

软硬件环境如下表所示：

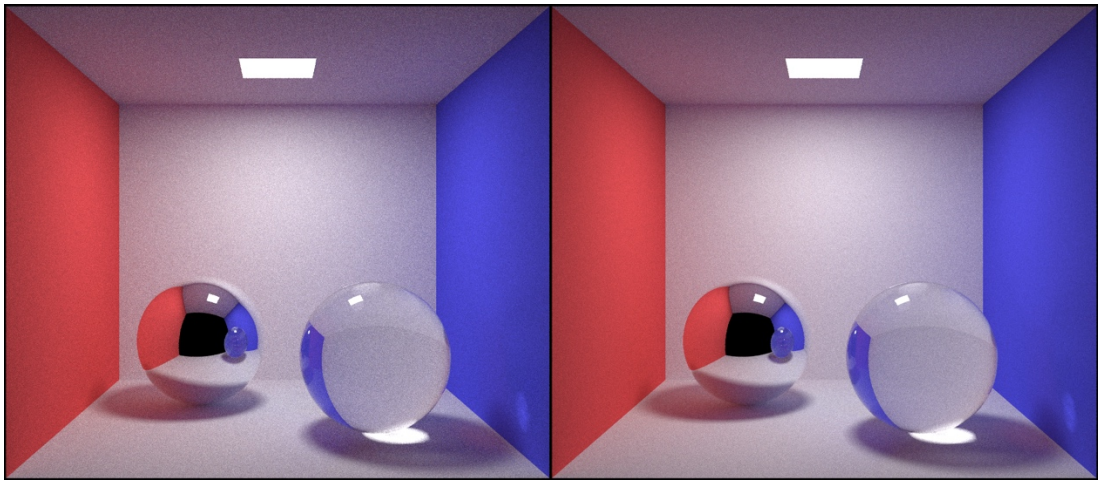
表 1 程序运行的软硬件环境

项目	参数
操作系统	macOS 10.13.3
CPU	Intel Core i7 7700HQ
GPU	AMD Radeon Pro 555
内存	16GB
IDE	XCode 9.2

5.1 cbox 场景：



（a） （b）



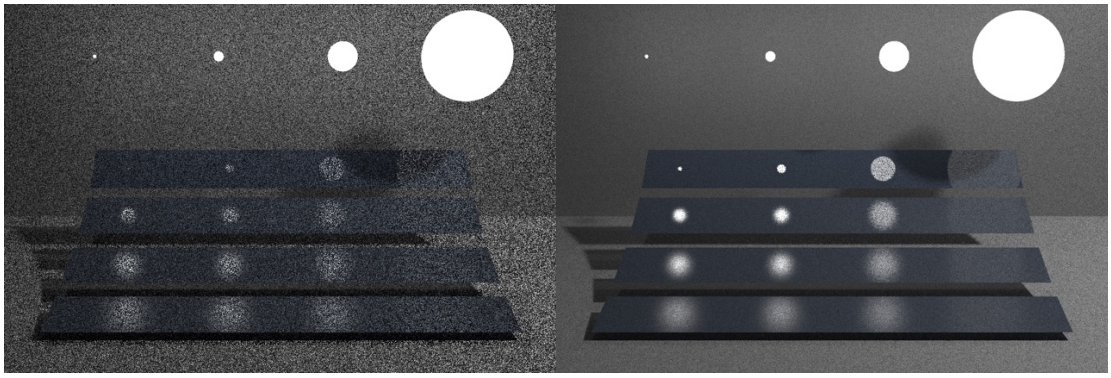
（c） （d）

四幅图的 maxDepth 均为 10，绘制的具体情况如下表所示：

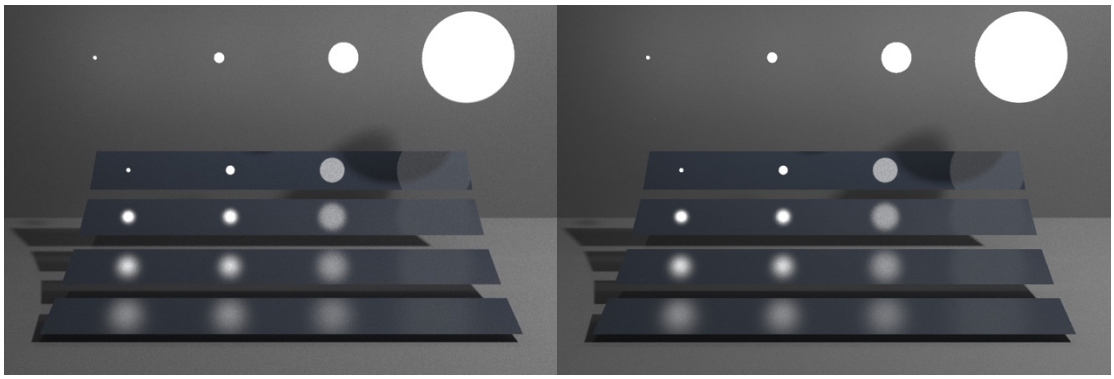
表 2 cbox 场景绘制情况

图像	每个像素点的采样数	分辨率	绘制耗时(秒)
a	4	1000 * 1150	38
b	64	1000 * 1150	227
c	576	1000 * 1150	1674
d	1024	1000 * 1150	2956

5.2 veach 场景



（a）　　（b）



（c）　　（d）

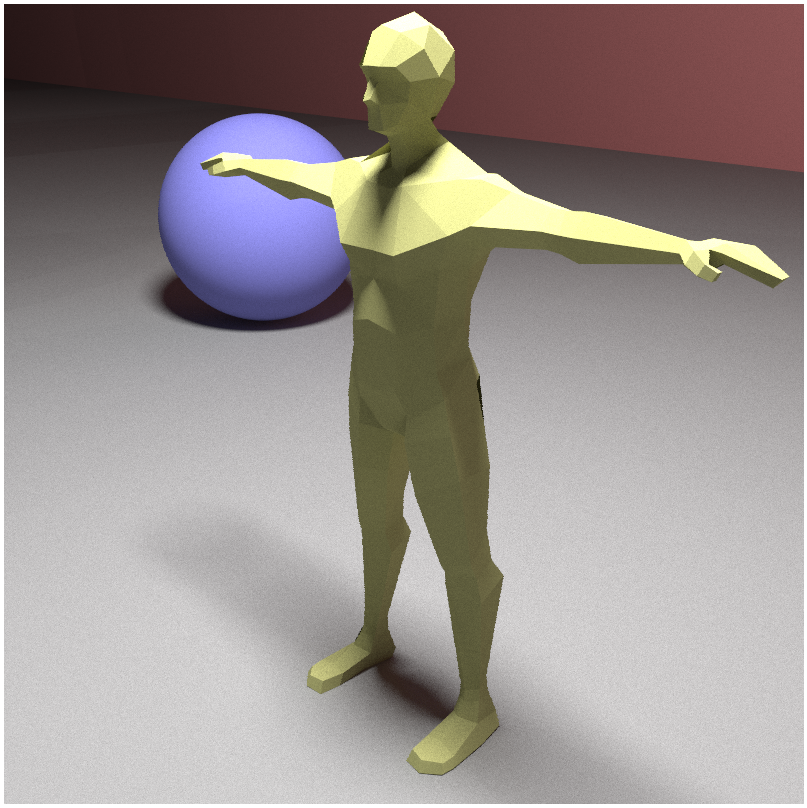
四幅图的 maxDepth 均为 10，绘制的具体情况如下表所示：

表 3 veach 场景绘制情况

图像	每个像素点的采样数	分辨率	绘制耗时(秒)
a	4	768 * 1152	16
b	64	768 * 1152	82
c	576	768 * 1152	636
d	1024	768 * 1152	1122

5.3 其他场景

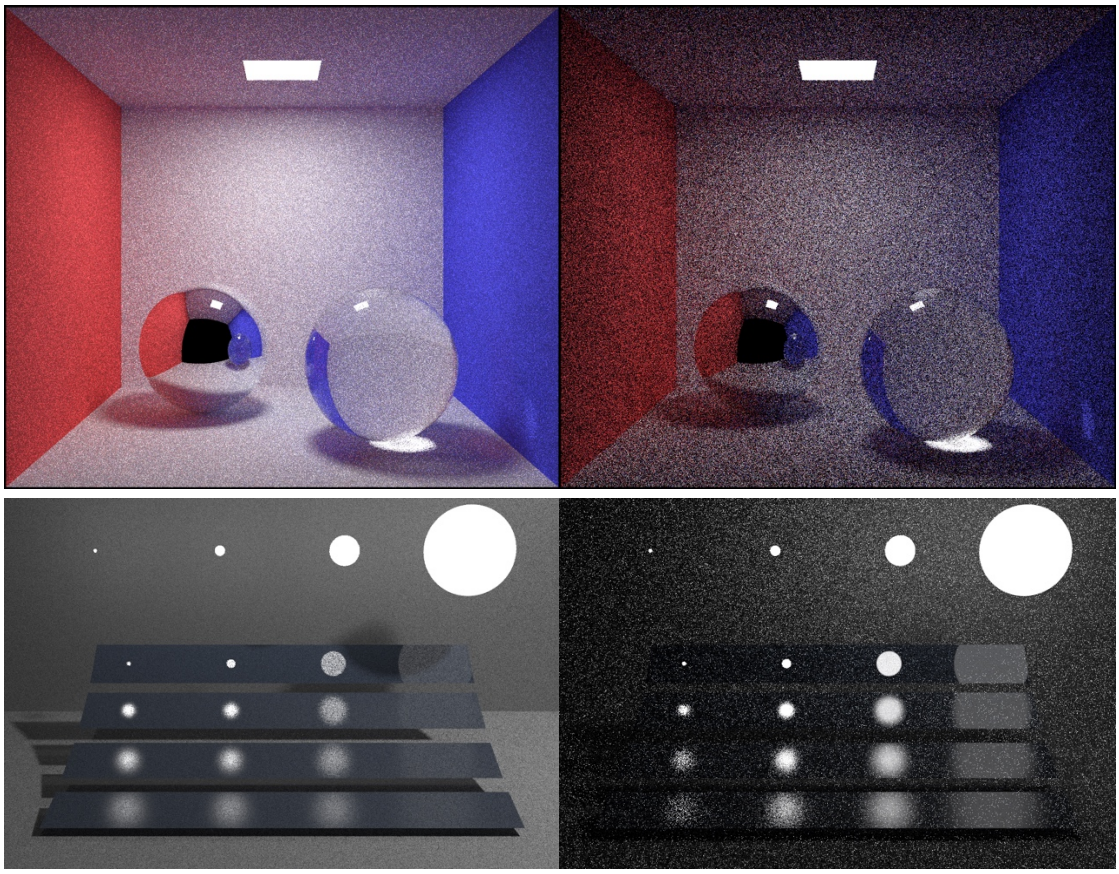
在完成这个光线追踪器之后，自己写了一个场景进行绘制，这个场景从文件中读入 obj 文件，之后进行绘制，由于绘制时间较长，在作业截至之前只绘制出一张图像：



上图的分辨率为 800 * 800，sampleCount 为 1024，耗时 18477 秒。

5.4 与普通的光线追踪算法的对比

普通的光线追踪算法是从光线出发到光源结束，在每个交点处不对直接光照进行采样的算法，比较常见的就是一个名为 smallpt 的开源项目，这样的算法相比于本文实现的算法来说收敛要慢很多，下面进行对比来说明本算法的优越性。



上面两组图中，左图是本程序得到的结果，右边是不采样直接光照得到的结果，所有的绘制参数均相同，sampleCount 均为 64。可以明显看出，在相同的采样数量的情况下，本程序得到的结果要明显好于普通的光线追踪算法，能够在较少的 sample 下得到很好的绘制结果。