基于线采样的 Monte Carlo 渲染实验报告

王明 161220124

1 问题描述

本次作业需要解决三个问题。

第一，如何使用 Monte Carlo 方法来解决渲染方程中的直接光照。

第二，如何对于面光源进行线采样。

第三，如何基于线采样的 Monte Carlo 方法来实现路径追踪。

第四，其他方面可以做哪些创新、优化或者改进。

2 解决思路

2.1Monte Carlo 方法解渲染方程中的直接光照

Monte Carlo 是使用一种采样的方法来解决函数的积分问题，特别是在函数的积分无法直接求解的时候，通常使用 Monte Carlo 方法来求解，这种方法是一种数值方法的求解，最终得到是一个最终的值。

下面是标准的渲染方程中的直接光照的计算（这里简化了表示）：

$$L_{direct}(\dots) = \int_A L(\dots) f_r(\dots) G(\dots) V(\dots) dA$$

其中$L_{direct}(\dots)$是直接光照的能量，$L(\dots)$是光源的能量，$f_r(\dots)$是反射的能量，$G(\dots)$是角度的矫正量，$V(\dots)$是光源的可见性（也就是光源和物体之间是否有别的物体阻挡），dA是光源面积的积分。

Monte Carlo 方法来计算一个积分的方法如下：

积分的形式如右边的公式所示$\int_b^a f(x)dx$，采取一个采样分布$X_i \sim p(x)$，此时得出的计算结果为$F_N = \frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)}$，如果$X_i$采用最常见的均匀分布（也就是$X_i \sim p(x) = C\ (constant)$），那么根据$\int_b^a p(x)dx = 1$，也就是$\int_b^a Cdx = 1$，那么$C = \frac{1}{b-a}$，那么此时计算此积分的结果为$F_N = \frac{b-a}{N}\sum_{i=1}^{N} f(X_i)$。

有了此积分方法后，我们以数值方法解出上面的渲染方程中的直接光照，首先我们将$V(\dots)$看成 1，也就是光源对于物体来讲是完全可见的，中间没有阻挡。此时方程变成了下面的公式：

$$L_{direct}(\dots) = \int_A L(\dots) f_r(\dots) G(\dots) dA$$

此时，传统的方法是使用点采样（point sampling）的方法，也就是在整个光源上随机进行均匀的采样点，然后结合 Monte Carlo 积分方法，就可以得到如下的计算方法：

$$F_N = \frac{A}{N}\sum_{i=1}^{N} L(X_i)f_r(X_i)G(X_i)$$

这样的点采样相对来说比较简单，但是对于面光源来说，点采样是一个二维积分，如果能够有一种采样方法使得其只需要一维的积分，那么就可以大大加快收敛的速度。基于此想法，有研究者就提出了线采样（line sampling）的算法，这种算法的思想是对于面光源每次都采样一条线，那么就可以将二维积分转变成一维积分，但是这样做会有额外的开销，采样有开销，在计算可见性时也会有额外的开销。但是好处是收敛的阶数会变高，然后收敛的速度会变快。

根据线采样的思想，标准的渲染方程中的直接光照的计算应该是以下的形式：

$$L_{direct}(\dots) = \int_a^b \int_0^{l(u)} L(\dots)f_r(\dots)G(\dots)V(\dots)dldu$$

其中dl是线段长度的积分，du是很多条不同的线段的积分，其中$u \in [a,b]$，代表的是u可以采样的空间，是一个维的空间。中间的这一部分可以用$L_{line}(\dots)$来表示，也就是下面所示的形式：

$$L_{line}(\dots) = \int_0^{l(u)} L(\dots)f_r(\dots)G(\dots)V(\dots)dl$$

那么此时的$L_{direct}(\dots)$就可以写成下面的形式：

$$L_{direct}(\dots) = \int_a^b L_{line}(\dots)du$$

这时我们就得到了对于渲染方程中的直接光源的线采样。

其中对于一条采样得到的具体的$L_{line}(\dots)$的计算方式如下：

$$L_{line}(\dots) = \int_0^{l} L(\dots)f_r(\dots)G(\dots)V(\dots)dt$$

假设光源的能量是一个常量，并且完全可见，那么可以得到：

$$L_{line}(\dots) = \int_0^{l} L_{light}f_r(\dots)G(\dots)dt = L_{light}\int_0^{l} f_r(\dots)G(\dots)dt$$

其中$G(\dots)$是角度的矫正量，是可以直接计算的，而$f_r(\dots)$的计算就是和不同的材质模型计算有关，常见的有漫反射模型、Blinn-Phong 模型等等。

至此我们就得到了基于线采样的 Monte Carlo 方法解渲染方程中的直接光照的方法，其中有一个问题其实还没有解决，那就是我们现在是默认光源都是对物体可见的，但是实际的情况下并不都是这样的理想情况，关于$V(\dots)$的计算，我们在下一节具体介绍线采样的方法后进行说明。

2.2 对于面光源进行线采样

在之前的章节中提到点采样的方法，其采样的方法也是非常得简单，也就是在面光源的面积范围内随机采样一个点就可以了。而线采样的话，是需要在面光源上采样一条直线，通常情况下的做法是，在面光源的平面内确定一个固定的方向，该方向的范围是面光源的大小来决定的，然后对这个固定的方向进行采样得到一个采样点，在采样点处做这个固定方向的垂线，得到的于面光源相交的部分就是我们采样得到的线光源。着一开始的固定方向做一条线，穿过面光源，此时就得到了线采样的结果，也就是一条线光源。其采样的过程如下图所示：

可以看到向量l就是我们确定的固定的方向，而从a到b的区间就是我们的确定的采样范围范围，u 就是我们在线段上的一个采样点，然后做垂线，相交的蓝线

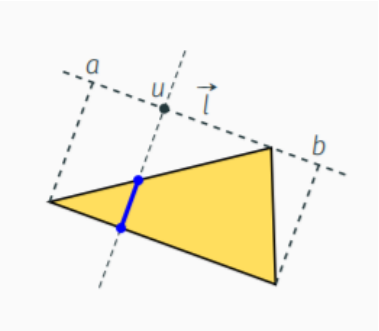

图 1 三角面光源线采样示意图

部分就是我们采样得到的面光源。

以上就是我们具体的在面光源上进行线采样的过程。接下来解决可见性的问题。对于点光源而言，可见与不可见则是在采样点和着色点之间发射一条光线，进行求交测试，如果之间有物体，那么该直接光照的能量直接置 0。而对于线采样来说，结果不是非 0 即 1，我们应该计算一条线光源上有多少的部分是可见的，通常的计算方法是连接线光源的两个端点和着色点形成一个三角形，然后计算这个三角形和其他物体有没有相交的部分，如果有，将这个相交的部分映射到线光源上，最后我们可能会得到断断续续的线段，一段可见，一段不可见，此时直接将可见的长度相加，就得到了可见的比例。这种方法虽然可行，但是其计算量是非常大的，效率不高。我们可以考虑一种估计性的方法来近似结果。基于的考虑是一条光线的求交测试比一个三角形的求交测试要快得多（并且使用了 BVH 加速结构），因此我们可以在线光源上随机采样 N 个点，然后按照点光源的方法做可见性测试，假设其中有 M 个可见，我们就可以认为线光源可见的比例是 M/N。这样是一种利用估计快速检测可见性的方法。

2.3 基于线采样的 Monte Carlo 方法来实现路径追踪

我们现在已经完成了线采样并且推导出了 Monte Carlo 方法来计算渲染方程中的直接光照部分。接下来我们展现具体的基于线采样的路径追踪(Path Tracing)算法:

```
shade(p, wo):
    LinesampleLight(line, pdf_light)
    Compute visible coefficient v from line & p
    sample a point x from line
    L_dir = L_i * fr(wo, ws, N) * dot(ws, N) * dot(ws, NN) / |x - p|^2 / pdf_light * len * v
    L_indir = 0.0
    Test Russian Roulette with probability prob
    wi = sample(wo, N)
    Trace a ray r(p, wi)
    If ray r hit a non-emitting object at q:
        L_indir = shade(q, wi) * eval(wo, wi, N) * dot(wi, N) / pdf(wo, wi, N) / prob
    return L_dir + L_indir
```

其中前一部分 L_dir 是计算直接光照的能量，其中就用到了对面光源的线采样，后一部分 L_indir 是计算间接光照的能量，也就是其它物体反射的光的能量。最终的计算 p 点来自 wo 方向的能量就是 L_dir 和 L_indir 的能量之和。

2.4 其他方面的创新、优化和改进

第一，注意到串行运行的速度较慢，而对于每个像素的渲染是互相独立的，因此在多核 CPU 上可以利用多线程加速。因此我实现了四线程加速，主要利用了 C++标准库中提供的 std::thread 线程库。

第二，一开始对于材质这一项，只实现了漫反射这种最简单的材质模型，由于漫反射在诸如球体这样的曲面上没有好的应用效果，因此我实现了微表面模型，使得可以体现出表面的粗糙度，从而可以模拟一种金属磨砂的效果。

3 结果与分析

3.1 代码说明

本次大作业的实验部分是基于闫令琪老师在 GAMES101 上开设的“现代计算机图形学入门”中的第七次实验（关于实现路径追踪的实验）。本次代码全是采用 C++语言编写。

整个项目代码的结构如下:

images：用来保存图片结果的文件夹

models: 用来保存模型，包含 Cornell Box 和 Teapot 两个模型。

AreaLight 类：定义面光源

Bounds 类：定义边界

BVH 类：定义包围盒

global 类：定义一些全局变量

Intersection 类：定义光线和物体的交点属性

Light 类：定义光的属性

Material 类：定义材质

OBJ_Loader 类：用来解析 obj 文件得到三角形面片

Object 类：所有物体的父类，定义了很多必要的虚函数

Ray 类：定义一条光线的属性

Render 类：渲染类，具体地对于每个像素进行渲染

Scene 类：构建需要渲染的场景，并且实现光线追踪

Sphere 类：定义球体类

Triangle 类：定义三角形面片

Vector 类：定义向量、坐标等在三维空间中的属性、计算

main 函数：程序额入口，加载模型并且启动渲染

CMakeLists.txt：这个是使用 cmake 编译需要的文件

代码执行的方法如下:

运行的平台是 Ubuntu18.04（我自己是用的虚拟机运行），需要安装 cmake 和

make 编译工具。

运行的命令如下，首先需要在命令行进入到 Code 文件夹，然后执行如下命令进行编译运行：

```
$ mkdir build
$ cd ./build
$ cmake ..
$ make
$ ./Raytracing
```

3.2 运行结果

下面所有的渲染均是采用 16spp，四线程加速的结果。

以下分别是点采样和线采样渲染 Cornell Box 的结果：

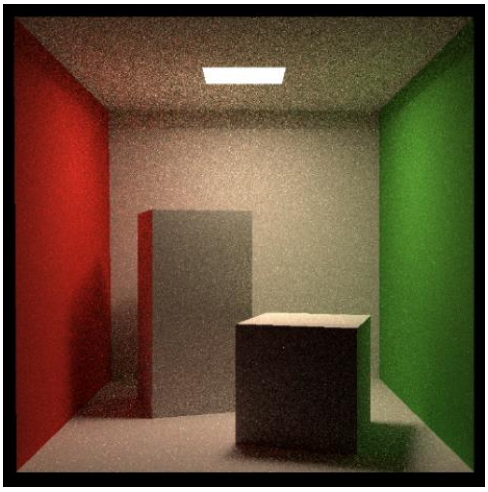

图 2 基于点采样的 Cornell Box

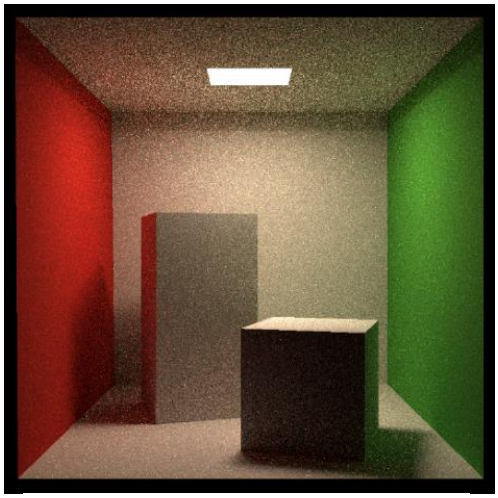

图 3 基于线采样的 Cornell Box

可以看出线采样的结果相对来说更好一些，点采样的结果相对来说偏暗一些，原因是点采样一个点如果被判断是不可见的，那么就会认为整个光源都是不可见的，而线采样会计算光源可见的比例，因此渲染出来的结果更加真实。

以下是采用了微表面模型后不同的粗糙度的小球的线采样的渲染结果：

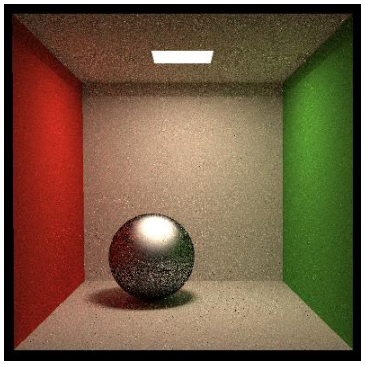

图 4 表面粗糙度 0.4

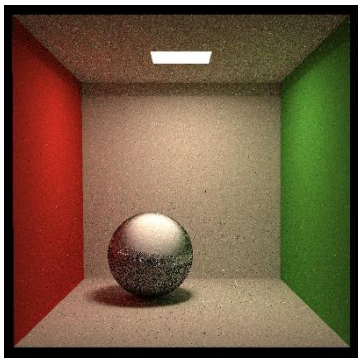

图 5 表面粗糙度 0.6

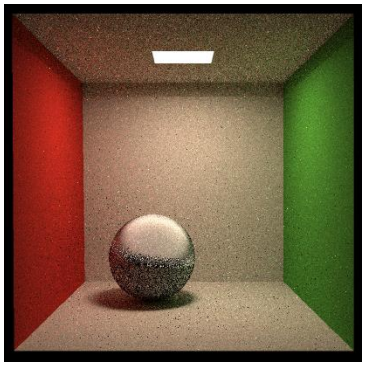

图 6 表面粗糙度 0.8

可以看出使用微表面模型之后会有一种金属的质感，并且随着粗糙度越大，越来越接近漫反射的结果，粗糙度越低，表面越光滑，其表面的高光就越明显。

以下分别是采用漫反射和微表面材质模型后的对于 Teapot 茶壶的渲染结果：

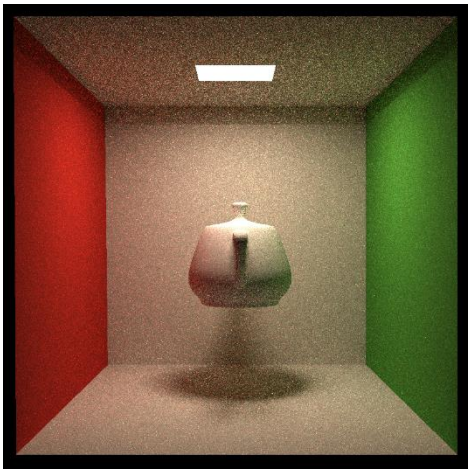

图 7 基于线采样的漫反射材质

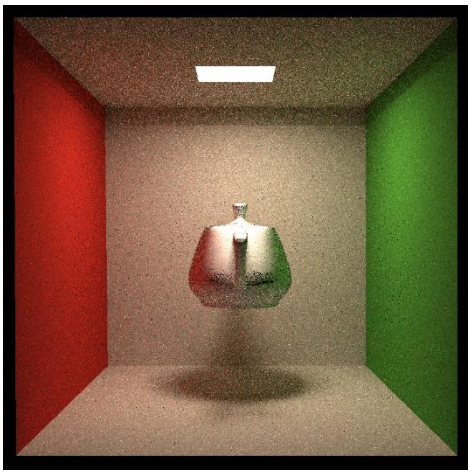

图 8 基于线采样的微表面材质

左图是漫反射材质，看上去是石头的质感，右图是微表面材质，看上去是金属的质感。

3.3 程序性能

这里展示了单线程和多线程（四线程）渲染的用时对比。

方法	模型	用时
单线程	Cornell Box	3941 秒
多线程（四线程）	Cornell Box	1041 秒

多线程（四线程）	小球	2371 秒
多线程（四线程）	茶壶	1399 秒

不可否认的是自己写的基于 CPU 的路径追踪渲染器还是比较慢的，但是使用了多线程技术之后可以看到渲染的速度有很大的提升，大概可以提升四倍。

4 结论

本次实验实现了基于线采样的 Monte Carlo 方法用于求解渲染方程中直接光照的计算。并且在漫反射材质的基础上实现了微表面模型。除此之外，为了实现了多线程加速来提升渲染的速度。

对于面光源的线采样的 Monte Carlo 方法相对于点采样的方法来说，将需要积分的范围从二维降到了一维，从而可以大大提升渲染的收敛速度，也会一定程度上提升渲染的效果。它的缺点就是开销较大，首先采样一条线光源会有额外计算开销，其次在可见性的求解中，也会用到三角面和场景的求交，这一点的开销相比于光线和场景的求交来说开销会大很多。