光线追踪算法

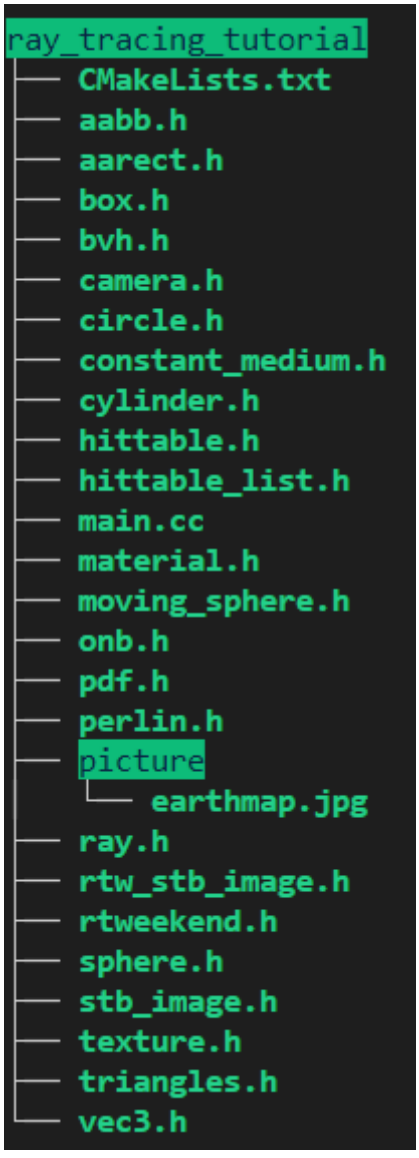
MG21330069 殷天润

实现功能

1. 学习了Ray tracing in one weekend,ray tracing the next week, ray tracing the rest of your life，实现了光线追踪算法，使用BVH中的AABB包围盒进行加速，借助蒙特卡洛以及重要性采样算法提高了采样的质量，让10采样的情况下的图片也能达到一定的质量。
2. 物体上：实现了球体，可以运动的球体，圆柱，三角形，三棱锥，圆盘，矩形；他们都实现了AABB包围盒；其中矩形和圆盘写了PDF概率密度函数以及对应的random函数，能够作为光源并且发挥重要性采样的效果，同时也允许多个光源；圆盘，圆柱，三角形三棱锥都允许任意的法相。
3. 材质上：实现了玻璃，金属，漫反射材质，并且实现了一部分的纹理和静态贴图
4. 摄像机允许定义位置，允许散焦模糊，光线实现了反走样，软阴影，处理了表面坏点。另外我也实现了旋转和平移操作。

项目编译

项目开发环境是ubuntu 20.04以及ubuntu 18.04，需要安装cmake，项目结构是:



编译以及运行方法:

```
cmake .
make
./raytracer_in_one_weekend > test.ppm
```

为了便于测试，代码中的采样为10，速度很快。

另外可以通过 https://github.com/Tyler-ytr/ray_tracing_tutorial.git 获取代码。

问题描述

光线追踪

Ray Tracing光线追踪算法实际上是反向的。原理是：从摄像机开始发射光线，照射到物体上，由于物体的材质发生反射折射到像素点上，如果没有照射到物体上那就会返回背景的颜色，最后像素点对颜色进行平均。实际上对于每一个像素，摄像机都会向这个像素的随机位置"发射"多条光线，这些光线的均值就会作为像素点的颜色。事实上，发射的光线越多，也就是采样越高，那么成像的质量会越好。

上述这些原理进一步诱发了以下问题：

1. 光线照射到物体上会发生什么？不同材质下光线会有怎样的表现？怎么从算法层面进行加速？
2. 像素点导致的锯齿应该怎么解决？
3. 怎么让成像更加真实？

构建光线追踪的过程其实也就是回答上述问题的过程。

本报告及项目仅仅初步解决了这些问题，同时由于时间所限是实现在单核CPU上的，运行语言是C++，开发时通过cmake在ubuntu上编译运行。

解决思路

光线和物体

首先一根光线中的点可以通过$C(t)=\vec{O}+t\vec{V}$进行描述，其中O相当于光线发出的点，V是方向向量。对于光线和物体而言关键的是求出光线和物体的交点以及物体在交点向外（也就是光线这一侧）的法向量。之后材质部分就可以根据这些信息来计算光线的反射或者是折射。

加速的方法是通过BVH树来进行加速，核心思想是用一个很好算交点的盒子包住物体，光线打到盒子上才会进一步进行求交点运算，否则就可以略过这步了。整个数据结构为了查询速度构建成二叉树的形式，并且多个物体的包围盒也可以进一步合并进而进一步加快速度。对于每一种物体而言都需要知道如何依赖自身的信息构建包围盒。我是用的包围盒是AABB包围盒，构建包围盒需要知道物体的三维最小值和三维最大值，进而构建一个竖直的立方体进行包裹。

平面

平面可以通过在平面上的点P_0以及平面的法向量$\vec{n}$进行描述，平面上面的任意一点P实际上都满足：$\vec{P}-\vec{P}_0=\overrightarrow{P_0P}\perp\vec{n}$，也就是$\overrightarrow{P_0P}\cdot\vec{n}=0$；然后和光线的参数方程进行联立（表示光线射到物体上）可以得到$(\vec{O}+t\vec{V}-\vec{P}_0)\cdot\vec{n}=0$，从而可以解得$t=-\frac{(\vec{O}-\vec{P}_0)\cdot\vec{O}-\vec{n}}{\vec{B}\cdot\vec{n}}$；（这里分母为0显然是不合法的）

然后需要判断t是否合法，需要根据平面进行区分的讨论：

- 对于圆面而言，只需要判断焦点到圆心的距离是否小于R就行了。圆面的法向量通过输入来确定（我的输入除了圆心之外还有一个远点，用圆心指向远点的向量来表示法向量）
- 对于三角面，通过这种方法进行求解的话有两种方法进行合法性判断，一个是同向法，另一个是重心法，具体可以参考博客;

另外1997年的论文《Fast, minimum storage ray-triangle intersection》有更加快速的三角形交点的求解方法，我在项目中参考了这篇博客的内容对该方法进行了实现。

法向量可以通过三角形边的叉乘来进行确定。

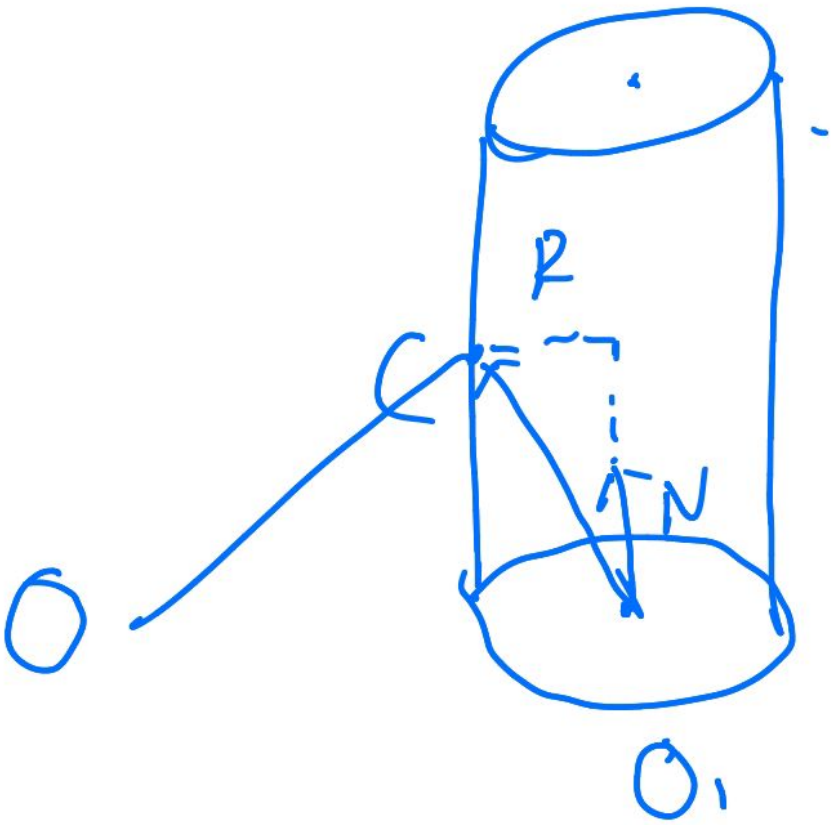
然后是包围盒的部分：

- 对于圆面而言，首先通过它的法向量可以和比如(0,0,1)或者(0,1,0)进行叉乘得到圆面上面的一个向量称为$\vec{A}$，然后$\vec{A}$和法向量进一步叉乘可以得到圆面上面的另一个向量$\vec{B}$，这两个都可以取单位向量。然后通过这两个向量就可以构建一个正方形包裹住圆面，然后通过这四个坐标就可以求得需要的两个极值点了。（如果两个极值点的某一个维度相等，需要加一个类似于0.001的值防止包围盒贴合）
- 对于三角形而言，直接通过三个点的坐标求极值点就行了。

圆柱

求交点的方法：

1. 光线的表达式为:$C(t) = \vec{O} + t\vec{V}$;
2. 假设一个圆柱的底面圆心是顶面圆心是$\vec{O}_1$，顶面圆心是$\vec{O}_2$，半径是R；那么可以得到底面往上的单位向量$\vec{N}$是$\frac{\overrightarrow{O_1O_2}}{||\overrightarrow{O_1O_2}||}$;
3. 已知叉乘$\vec{a} \times \vec{b} = ||\vec{a}||||\vec{b}||sin\theta$
4. 现在考虑C与圆柱侧面相交的情况；



根据上面一条3叉积的定义不难发现：

$$(\overrightarrow{O_1C} \times \vec{N}) = R^2$$

由于$\vec{C}$是关于t的一个表达式，那么可以把上面的公式表示成一元二次方程$at^2+bt+c=0$;其中:

- $a=(\vec{V}\times\vec{N})^2$
- $b=2(\vec{V}\times\vec{N})\cdot(\overrightarrow{O_1O}\times\vec{N})$
- $c=(\overrightarrow{O_1O}\times\vec{N})^2-R^2$

假设方程的根为$t_1,t_2(t_1<t_2)$:

- 如果$0<t_1<t_2$，那么t_1可能为解
- 如果$t_1<0<t_2$，那么光源在圆柱内，t_2可能为解
- 如果$t_1<t_2<0$说明无解;

把上面的解带入$C(t)$，然后判断$(\overrightarrow{O_1C}\cdot\vec{N}_1)>0$ && $(\overrightarrow{O_2C}\cdot\vec{N}_2)>0$来保证交点在圆柱的高度范围内

5. 如果是和圆柱底面或者顶面相交:

   使用上面提到的方法进行求解交点

6. 比较第4步和第5步的t，取最小值为实际交点对应的t

求包围盒的办法:

- 圆柱通过上下两个底面根据平面中的圆面求包围盒的方法可以得到八个顶点，然后用这八个顶点求极值点即可。

球体

求交点的方法:

- 这里通过球心到球面的是距离是R的原理，结合光线方程的方法进行求解，这两个性质可以得到相对于t的一元二次方程:

  $$(\vec{O}+t\vec{V}-\vec{C})^2=R^2$$

  如果一元二次方程无解说明不存在交点，如果存在解 选择t值小的解即可（远处的被遮挡了）;

  法向量就是球心指向交点的向量。

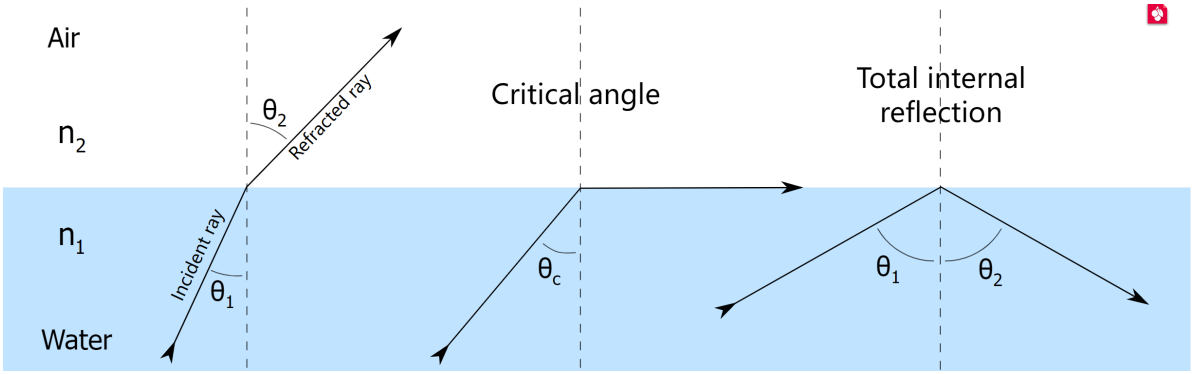
三棱锥

可以用四个顶点构建四个三角形面片，进而可以求交点和对应的法向量；包围盒通过四个顶点就可以求到极值点

光线和材质

上面一部分可以得到交点以及物体在这个交点对应的法向量，物体的材质决定了发出的光线。这里对于物体而言主要有两种情况，光线会进行折射或者反射。

如下图是维基百科的例子:



反射

对于反射而言，假设入射光线是$\vec{v}$，出射光线是$\vec{v}'$。入射光线可以根据平行或者垂直平面分解为$\vec{v}_\perp$以及$\vec{v}_\parallel$;假设平面的法向量是$\vec{n}$，那么可以得到$\vec{v}_\perp = (\vec{v}\cdot\vec{n})\vec{n}$；显然，出射方向这个分量的方向就是$-\vec{v}_\perp$；然后出射光的平行的部分保持不变，所以有:

$$\vec{v}' = \vec{v}'_\parallel + \vec{v}'_\perp = \vec{v}_\parallel - \vec{v}_\perp = \vec{v}_\parallel + \vec{v}_\perp - 2\vec{v}_\perp = \vec{v} - 2(\vec{v}\cdot\vec{n})\vec{n}$$

折射

这部分的理论依据是Snell法则:

$$\eta \cdot \sin\theta = \eta' \cdot \sin\theta'$$

在上图中θ就是θ_1，θ'就是θ_2。如果知道了θ'就知道了折射防线的方向了。

这里使用书上的结论，将出射光R'拆成$R'_\perp, R'_\parallel$，假设法向量是N，并且他们都是单位向量，有:

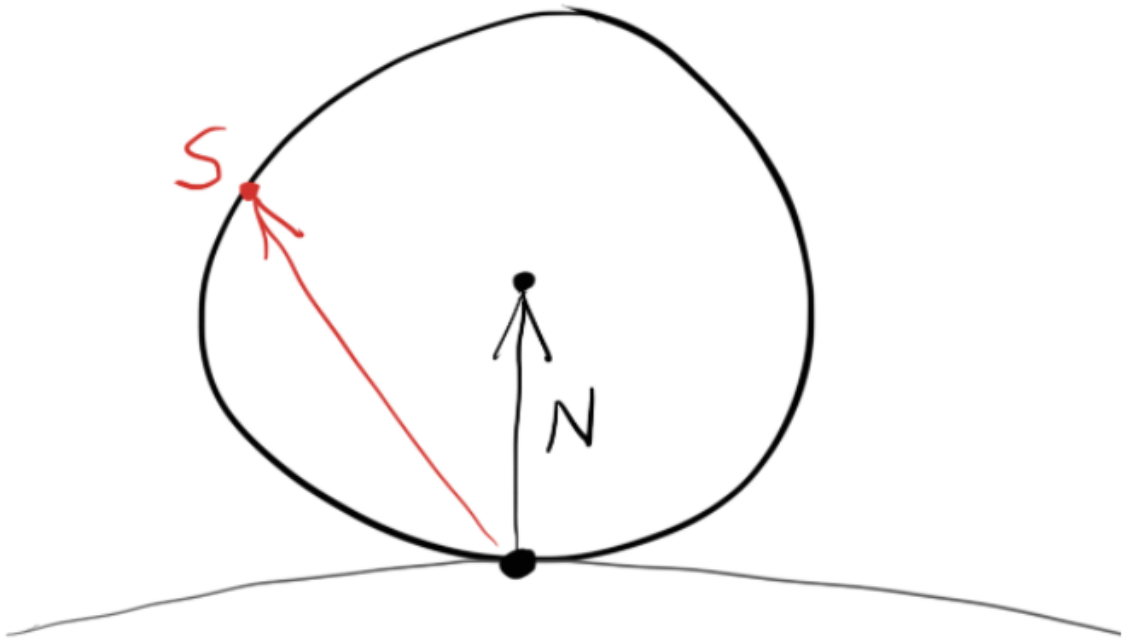
$$R'_\parallel = \frac{\eta}{\eta'}(R + \cos\theta N) = \frac{\eta}{\eta'}(R + (-R\cdot N)N)$$

$$R'_\perp = -\sqrt{1 - |R'_\parallel|^2}N$$

η与介质性质有关，比如玻璃是1.5，空气是1.0。

其他部分

对于特定的材质而言，还有可能需要考虑的是光线吸收了多少(attenuate)以及是否有漫反射等等。如果发生漫反射的话，其中的一种做法是如下图:

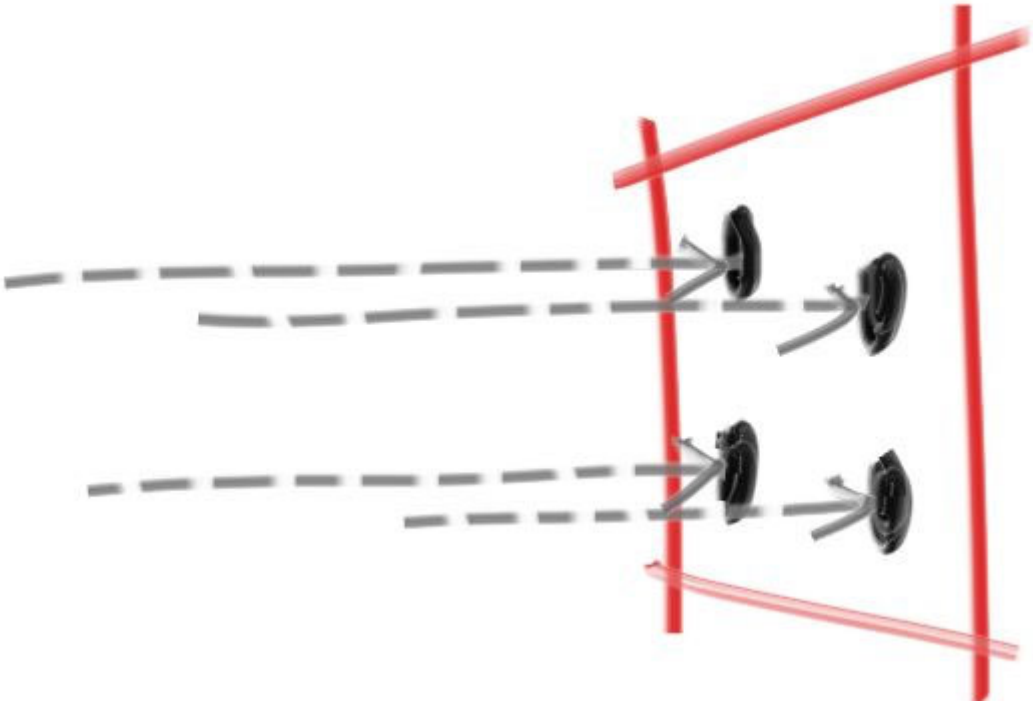


在光线与物体的交点的法向量那一侧首先生成一个单位球，然后在这个球面上生成一个随机的向量作为出射的光线。

另外材质还可能有纹理，比如全球的地图或者柏林噪声等等。这块其实要求着对于物体表面特定的点要转换成二维坐标u,v进而对应纹理上面的坐标（显然纹理作为贴图是二维的）。

反走样

最基础的版本，光线相当于一个像素点射一次，这就会带来锯齿问题，因为像素点不一定只有中心所代表的颜色。从这个像素点的角度而言，为了更加精准就需要多次采样求均值，一个比较好的方法是把像素点进一步分割成多个小块然后在这些小块的随机位置射入光线最后求颜色平均值(jitter)。



蒙特卡洛算法和重要性采样

这部分是rest of life的内容，主要的目的在于让随机的过程（比如光源发光，漫反射）等等更加贴近物理世界，可以让较少的采样就能带来较好的结果。这部分要求实现PDF以及对应的随机分布的生成，我实现了圆面以及矩形用于光源采样，以及漫反射，金属，玻璃材质用来生成更加逼真的表面，时间所限这篇报告不再赘述。

测试案例以及效果演示

我搭建的场景是我进一步修改的康奈尔盒子:

```
    objects.add(make_shared<yz_rect>(0, 555, 0, 555, 555, green));
    objects.add(make_shared<yz_rect>(0, 555, 0, 555, 0, red));
    objects.add(make_shared<circle>
(point3(278,554,279),point3(278,0,279),60,light));
    objects.add(make_shared<circle>
(point3(500,500,500),point3(0,0,0),60,light));
    objects.add(make_shared<circle>
(point3(60,500,500),point3(500,0,0),60,light));
    //objects.add(make_shared<flip_face>(make_shared<xz_rect>(213, 343, 227, 332,
554, light)));
    objects.add(make_shared<xz_rect>(0, 555, 0, 555, 555, white));
    objects.add(make_shared<xz_rect>(0, 555, 0, 555, 0, white));
    objects.add(make_shared<xy_rect>(0, 555, 0, 555, 555, white));
```

不同之处是里面都是圆形光源并且为了体现我实现了各个方向的圆形光源在盒子的两角加入了两个光源.

然后里面的物体是金属盒子（镜面反射），斜在空中的圆柱（体现我实现了任意方向的圆柱）材质是白色的lambertian，一个蓝色的lambertian正三棱锥，然后是三个小球，包括了一个地图贴图球，玻璃球以及动态模糊的lambertian球。

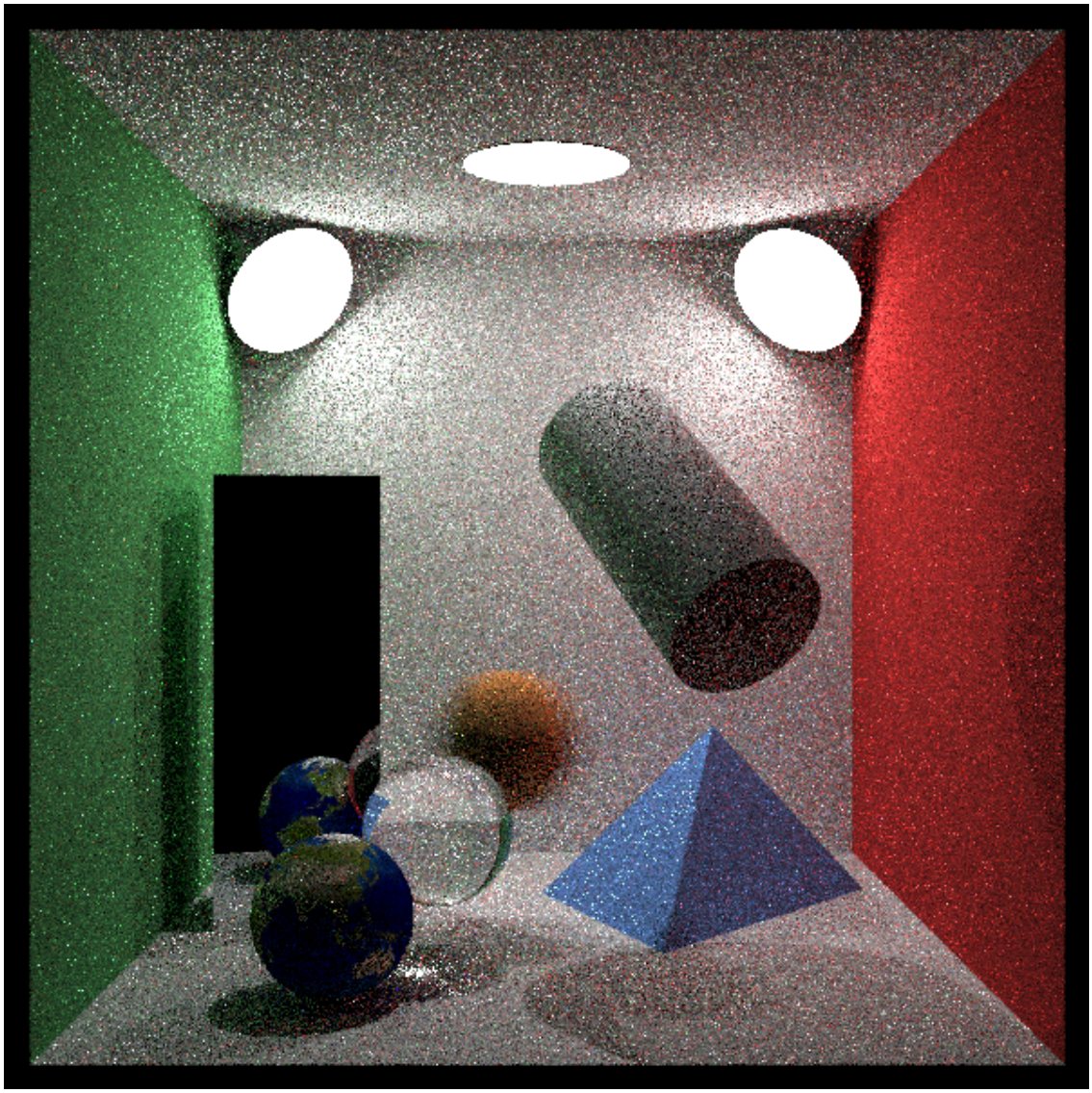
```
    shared_ptr<material> aluminum = make_shared<metal>(color(0.8, 0.85, 0.88),
0.0);
    shared_ptr<hittable> box1 = make_shared<box>(point3(0,0,0),
point3(120,330,120), aluminum);
    box1 = make_shared<rotate_y>(box1, 10);
    box1 = make_shared<translate>(box1, vec3(400,0,295));
    objects.add(box1);

    shared_ptr<hittable> cylinder1 = make_shared<cylinder>(point3(0,0,0),
point3(120,120,120),50,white);
    cylinder1 = make_shared<rotate_y>(cylinder1, -18);
    cylinder1 = make_shared<translate>(cylinder1, vec3(160,230,80));
    objects.add(cylinder1);

    shared_ptr<hittable> pyramid1=make_shared<pyramid>
(point3(0,140,0),point3(-1.44*70,0,-1.44*70),point3(140,0,0),point3(0,0,140),blu
e);
    pyramid1 = make_shared<rotate_y>(pyramid1, 70);
    pyramid1 = make_shared<translate>(pyramid1, vec3(150,0,350));
    objects.add(pyramid1);

    auto glass = make_shared<dielectric>(1.5);
    objects.add(make_shared<sphere>(point3(350,90,190), 50 , glass));

    objects.add(make_shared<sphere>(point3(410,50,120), 50 , earth_surface));
```

```
    auto moving_sphere_material = make_shared<lambertian>(color(0.7, 0.3, 0.1));
    vec3 center1 = point3(290, 140, 260);
    objects.add(make_shared<moving_sphere>(center1, center1+vec3(30,0,0), 0, 1,
50, moving_sphere_material));
```

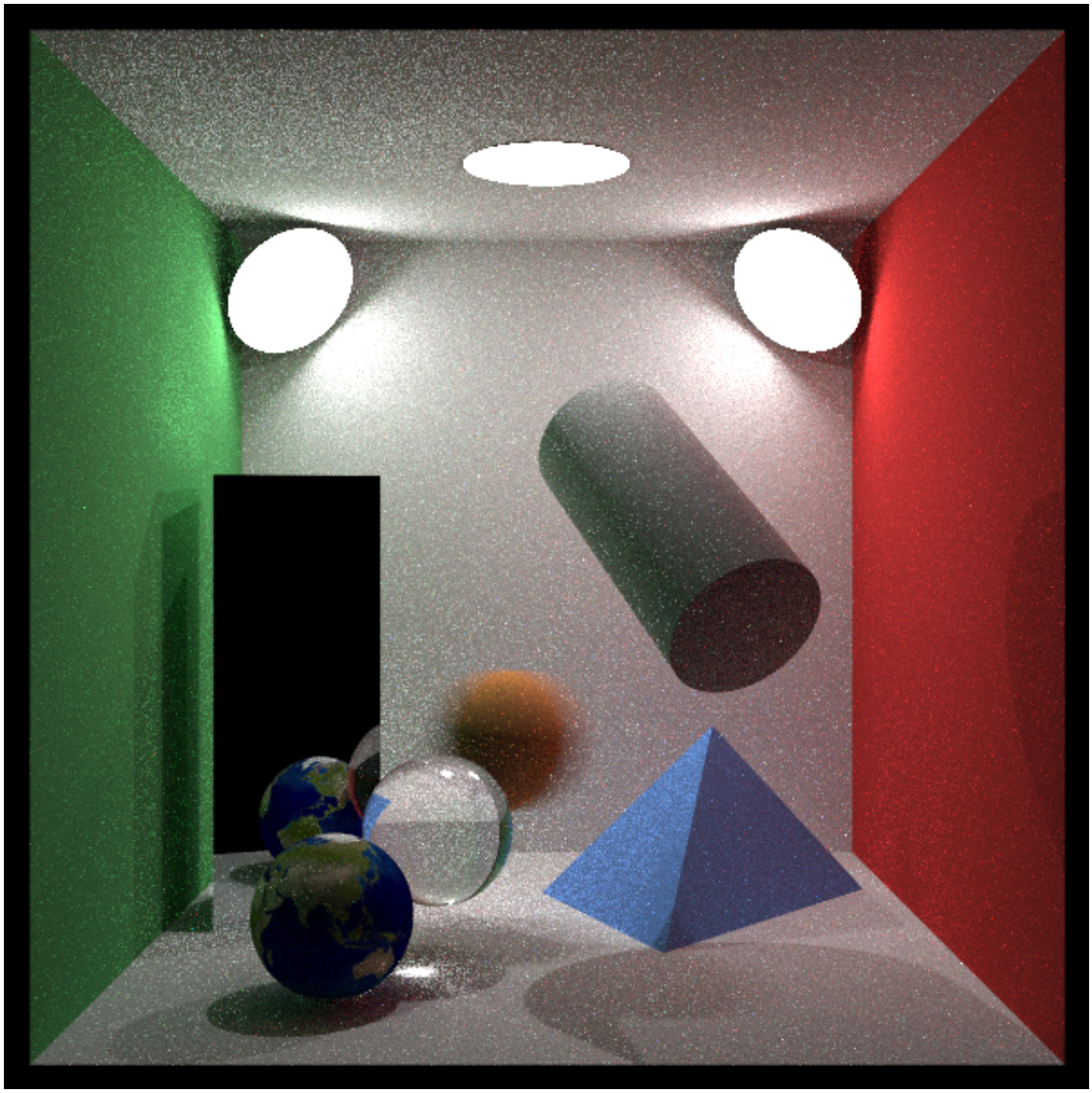
其中光线迭代次数都是50次。

以下是实验结果，包含了10采样(几十秒)，200采样(分钟级别)，1000采样(分钟到小时级别)：

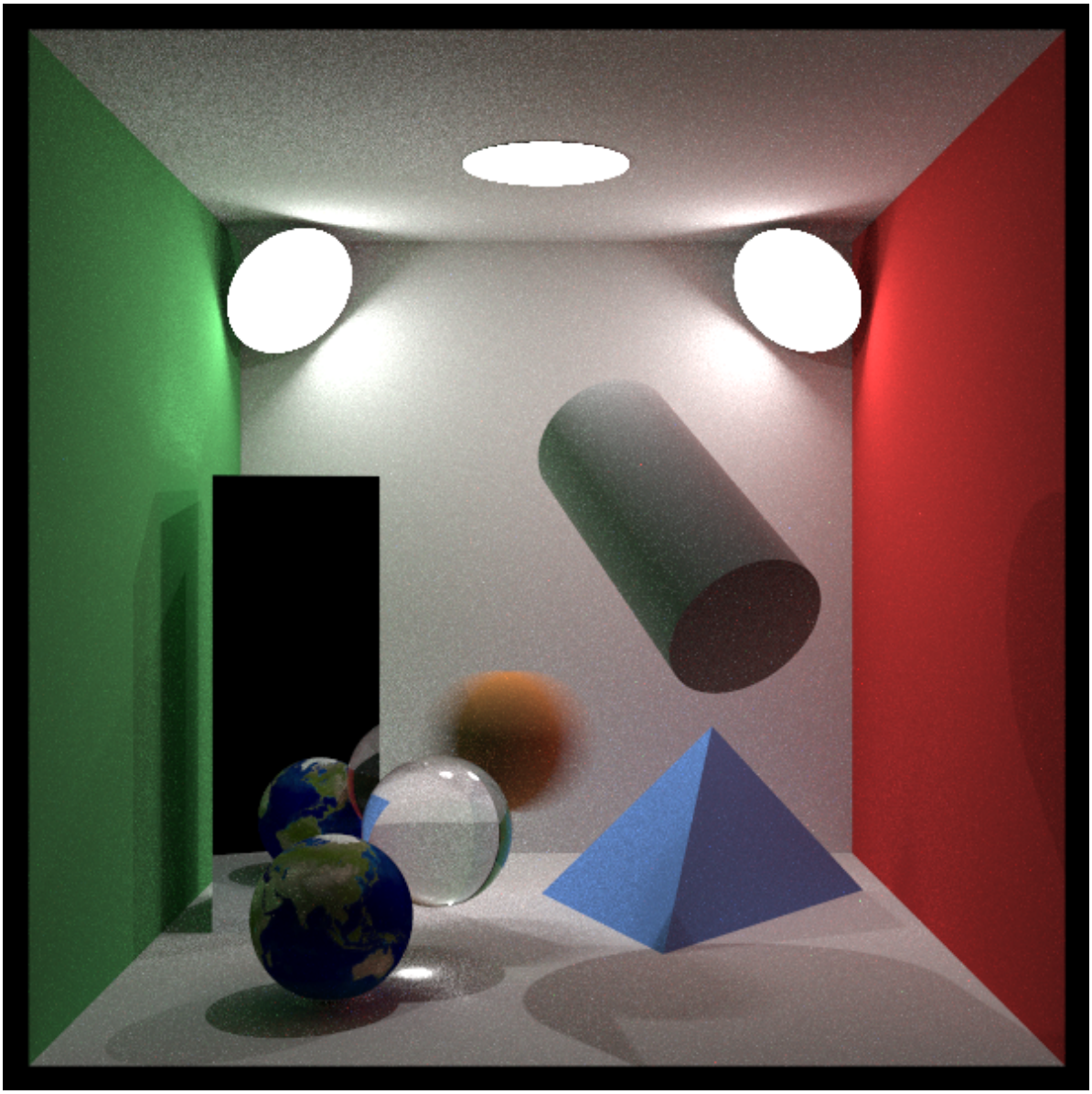
- 10采样:



- 200采样:



- 1000采样:



ps

- 不错的获取三维坐标的网站：https://www.geogebra.org/3d