《计算机图形学实验报告》

渲染方向：全局光照

班级：[redacted]

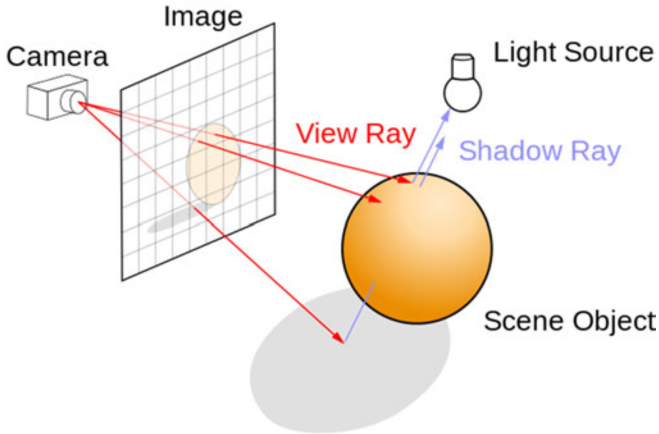
姓名：[redacted]

学号：[redacted]

二〇二二年 十二月

一、路径追踪算法

从视点（相机）向每个像素点发出多条光线，每条光线与物体表面相交时根据表面的材质属性继续采样一个方向，发出另一条光线，如此迭代，直到光线打到光源上（或达到递归次数限制）。同时使用蒙特卡洛积分求解渲染方程，并将同一像素点的多条光线的计算结果取平均值，作为单个像素的颜色值。



二、补全函数逻辑描述

1）Generating Camera Rays

①Camera::generate_ray，先利用相机的垂直视场角度大小 vert_fov 计算出屏幕高度，以及宽高比 aspect_ratio 计算出屏幕的宽度。然后将输入的二维屏幕坐标转化为相机坐标系下的坐标，以相机为射线起点，则该坐标值即为射线方向，由此便得到了相机坐标系下的射线。最后，通过矩阵变换将射线转换到世界坐标系下。

②Rect::sample，利用 RNG::unit()函数生成[0，1]间的随机值，并乘以矩形的长和宽，从而得到了矩形内均匀随机采样的二维点坐标。

③Pathtracer::trace_pixel，对同一个像素进行超采样时，为了使每次采样的射线不同，需要利用 Rect::sample()函数对像素点坐标进行随机偏移。

2）Intersecting Objects

①Sphere::hit，先利用射线 ray 的 point、dir 两个参数以及球的半径 radius 计算出联立后

的二次方程系数。然后再判断方程是否有解，若有解则继续判断交点是否在射线范围之内。当只有一个交点在射线范围内时，该点即为实际交点；当两个交点都在射线范围内时，较小的解即为实际交点。该点的法向即为球心到该点坐标的单位向量。（具体步骤见算法简述）

②Triangle::hit，先判断射线是否与三角形面片平行，若平行则无交点。否则计算出射线与三角形所在平面的交点，并判断交点是否在三角形内以及射线范围内，若在则射线与三角形相交。最后通过三角形重心坐标插值计算出交点法向。（具体步骤见算法简述）

3）Path Tracing

①Pathtracer::trace，该函数通过递归实现对光线的追踪，我们需要做的是删去返回物体表面法向颜色的代码。

②BSDF_Lambertian::scatter，调用 sampler.sample()函数随机对半球上任一方向进行采样，然后调用 evaluate()函数计算衰减率。

③BSDF_Lambertian::evaluate，计算漫反射时的衰减率：albedo * cos(theta)。

④BSDF_Lambertian::pdf，根据 cosine-weighted hemisphere 模型计算概率密度函数。

⑤Pathtracer::sample_indirect_lighting，使用 BSDF::scatter()函数随机采样一个光线方向，并将光线方向从局部坐标系变换到世界坐标系，再以此构造入射光线。注意，为了避免入射光线与场景的第一个交点为光线起点，需令 ray.dist_bounds.x=0.000001。此外，还应让递归深度减 1。然后，调用 Pathtracer::trace()函数计算入射光的辐照率，由于这里计算的是间接光照，因此应该使用函数返回值的第二个分量。最后，使用蒙特卡洛积分计算光线贡献值并乘以衰减率 attenuation。

⑥Pathtracer::sample_direct_lighting，该函数实现与间接光照几乎一样，不同点在于该函数计算的是直接光照，因此应让递归深度为 0，且使用 Pathtracer::trace()函数返回值的第一个分量作为辐照率。注意，由于点光源无法与追踪的光线相交，因此要调用 point_lighting()函数单独计算点光源的贡献并与辐照率相加。

4）Materials

①Pathtracer::sample_indirect_lighting 与 Pathtracer::sample_direct_lighting，由于折射和反射不使用蒙特卡洛积分，因此在子实验三的基础上，需调用 BSDF::is_discrete()函数来判断是否需要蒙特卡洛积分。

②reflect，根据出射光线方向，计算入射光线方向。（具体步骤见算法简述）

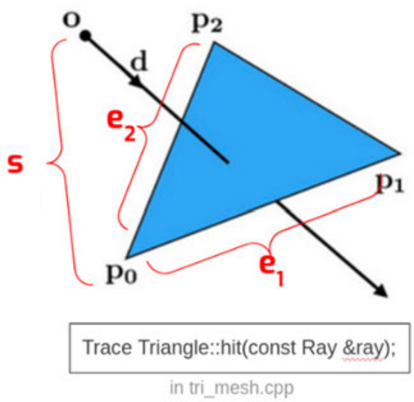
③BSDF_Mirror::scatter，调用 reflect()函数计算光线入射方向，并令衰减率为反射比。

④refract，先根据出射光线与法线的夹角求出两种介质的折射率，并根据斯涅尔公式及三角函数关系计算出射光线、入射光线各自与法线夹角的正弦、余弦值，然后据此判断是否发生全反射，若没有全反射则计算折射时的入射光线方向。（具体步骤见算法简述）

⑤BSDF_Glass::scatter，调用 refract()函数计算光线入射方向及判断是否发生全反射。当发生全反射时，调用 reflect()函数计算光线方向并令衰减率为反射比。若没有发生全反射，则使用 Schilck 近似方法计算菲涅尔系数，并以菲涅尔系数为概率来决定发生反射还是折射。折射时入射光线方向为 refract()函数返回值，同时令衰减率为透射率。

三、算法简述

1）射线与三角形求交点



射线方程为：$r(t) = \vec{o} + t \cdot \vec{d}$（其中$\vec{o}$为射线起点，$\vec{d}$为射线方向）

假设射线与三角形所在平面的交点为 P，$\overrightarrow{P_0P_1} = \overrightarrow{e_1}$，$\overrightarrow{P_0P_2} = \overrightarrow{e_2}$，$\overrightarrow{P_0O} = \vec{s}$，则：

$$\overrightarrow{P_0P} = u \cdot \overrightarrow{e_1} + v \cdot \overrightarrow{e_2}$$

$$\overrightarrow{OP} = t \cdot \vec{d}$$

根据

$$\overrightarrow{OP_0} + \overrightarrow{P_0P} = \overrightarrow{OP}$$

可得方程如下

$$-\vec{s} + u \cdot \overrightarrow{e_1} + v \cdot \overrightarrow{e_2} = t \cdot \vec{d}$$

当$\left(\overrightarrow{e_1} \times \vec{d}\right) \cdot \overrightarrow{e_2} = 0$时，方程无解，射线与三角形所在平面平行；

当$(\overrightarrow{e_1} \times \vec{d}) \cdot \overrightarrow{e_2} \neq 0$时，解得

$$u = -\frac{(\vec{s} \times \overrightarrow{e_2}) \cdot \vec{d}}{(\overrightarrow{e_1} \times \vec{d}) \cdot \overrightarrow{e_2}}$$

$$v = \frac{(\overrightarrow{e_1} \times \vec{d}) \cdot \vec{s}}{(\overrightarrow{e_1} \times \vec{d}) \cdot \overrightarrow{e_2}}$$

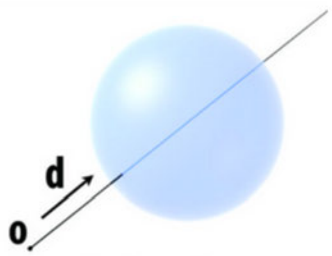
$$t = -\frac{(\vec{s} \times \overrightarrow{e_2}) \cdot \overrightarrow{e_1}}{(\overrightarrow{e_1} \times \vec{d}) \cdot \overrightarrow{e_2}}$$

当$0 \leq u, v \leq 1$且$u + v \leq 1$时，交点在三角形内。此外，交点还应在射线范围内。

交点坐标为：$\vec{P} = \overrightarrow{P_0} + u \cdot \overrightarrow{e_1} + v \cdot \overrightarrow{e_2} = (1 - u - v) \cdot \overrightarrow{P_0} + u \cdot \overrightarrow{P_1} + v \cdot \overrightarrow{P_2}$

交点法向为：$\vec{P}.normal = (1 - u - v) \cdot \overrightarrow{P_0}.normal + u \cdot \overrightarrow{P_1}.normal + v \cdot \overrightarrow{P_2}.normal$

2）射线与球形求交点



Trace Sphere::hit(const Ray &ray);

in shapes.cpp

射线方程为：$r(t) = \vec{o} + t \cdot \vec{d}$（其中$\vec{o}$为射线起点，$\vec{d}$为射线方向）

球的方程为：$\|\vec{x} - \vec{c}\|^2 - r^2 = 0$（其中$\vec{c}$为球心，$r$为半径）

当球心在原点时，有$\vec{c} = (0, 0, 0)$

联立两个方程可得

$$\|\vec{o} + t \cdot \vec{d}\|^2 - r^2 = 0$$

即

$$\|\vec{d}\|^2 \cdot t^2 + 2(\vec{o} \cdot \vec{d}) \cdot t + \|\vec{o}\|^2 - r^2 = 0$$

根据一元二次方程的求根公式，当$\Delta < 0$时方程无解，射线与球无交点；

当$\Delta \geq 0$时，可求出方程的两个根t_1、$t_2 (t_1 \leq t_2)$。

若两个根均在射线范围内，则t_1为实际交点；若仅有一个根在射线范围内，则t_2为实际

交点；若没有根在射线范围内，则射线与球无交点。

交点坐标可通过将根代入射线方程求得，法向即为球心到交点坐标的单位向量。

3）蒙特卡洛积分

在路径追踪算法中，需要对渲染方程进行求解，而方程中的积分项计算复杂，可以使用蒙特卡洛算法进行估算：

$$\int f(x)dx = \frac{1}{n}\sum_{k=1}^{n}\frac{f(X_k)}{pdf(X_k)}$$

式中，f(x)是被积函数，pdf 是概率密度函数，n 是采样个数。

需要注意的是，随着反射次数的增加，光线数量会爆炸增长，计算量将无法负担。因此，每次只采样一个方向，即令 n=1。这样又产生一个新的问题：虽然蒙特卡洛积分是无偏估计，但样本越少其偏差越大。故需要对同一个像素随机选择不同的光线进行采样，并将结果求平均，从而减少图像的噪声。

4）cosine-weighted-sampling

在蒙特卡洛积分中，概率密度函数 pdf 与被积函数 f(x)越相似，积分的收敛速度越快。当$pdf = cf(x)$，即 pdf 完全正比于被积函数时，方差达到 0，此时只需要 1 个 sample 就能得到积分的正确结果。但在实际应用中这往往是难以实现的，因此一个策略是让 pdf 与被积函数中的一部分成正比。

由于

$$f(\omega) = L_i(\omega)cos\theta$$

故可令

$$pdf(\omega) = c \cdot cos\theta$$

由

$$\int_{\Omega} pdf\,(\omega)d\omega = \int_0^{2\pi}\int_0^{\frac{\pi}{2}} c \cdot cos\theta \cdot sin\theta d\theta d\phi = 1$$

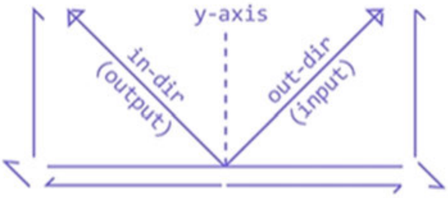
解得

$$c = \frac{1}{\pi}$$

故

$$pdf(\omega)=\frac{cos\theta}{\pi}$$

5）反射向量求解



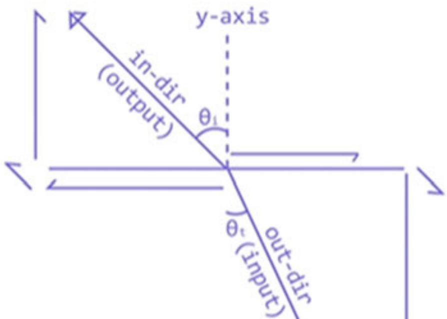
设入射光线为$\vec{I}$，出射光线为$\vec{O}$，法线为$\vec{N}$，则有：

$$\vec{I}+\vec{O}=2\vec{O}\cdot\vec{N}\cdot\vec{N}$$

代入$\vec{N}=(0,1,0)$，$\vec{O}=(x,y,z)$，可得

$$\vec{I}=(-x,y,-z)$$

6）折射向量求解



设入射光线为$\vec{I}$，出射光线为$\vec{O}=(x,y,z)$，法线为$\vec{N}=(0,1,0)$。

由于入射光线、出射光线及法线共面，故可令

$$\vec{I}=\alpha\vec{O}+\beta\vec{N}=(\alpha x,\alpha y+\beta,\alpha z)$$

由于

$$cos\theta_i=\vec{I}\cdot\vec{N}$$

且入射光线$\vec{I}$为单位向量，可得方程组

$$\begin{cases} \alpha y + \beta = cos\theta_i \\ \alpha^2(x^2 + z^2) = sin^2\theta_i \end{cases}$$

注意：入射光线与出射光线分别位于法线两侧，因此$\alpha < 0$。

故

$$\vec{I} = (-\frac{x}{\sqrt{x^2 + z^2}}sin\theta_i, cos\theta_i, -\frac{z}{\sqrt{x^2 + z^2}}sin\theta_i)$$

可由 Snell's Law

$$n_i sin\theta_i = n_t sin\theta_t$$

及三角函数关系

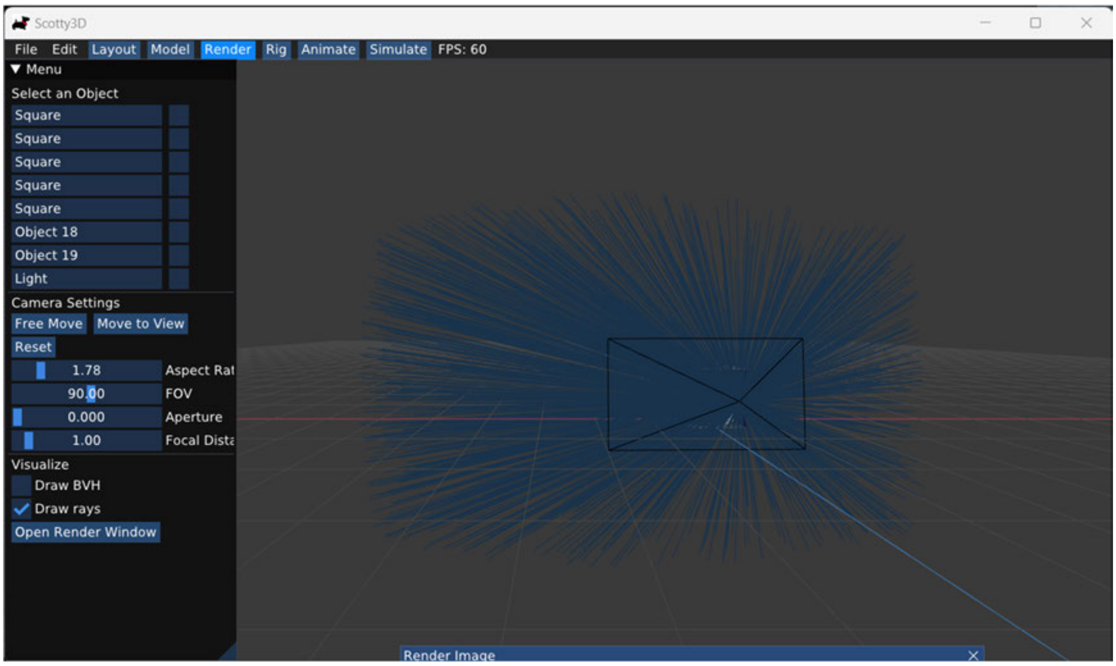
$$cos^2\theta + sin^2\theta = 1$$

求出式中的三角函数值$sin\theta_i$与$cos\theta_i$。

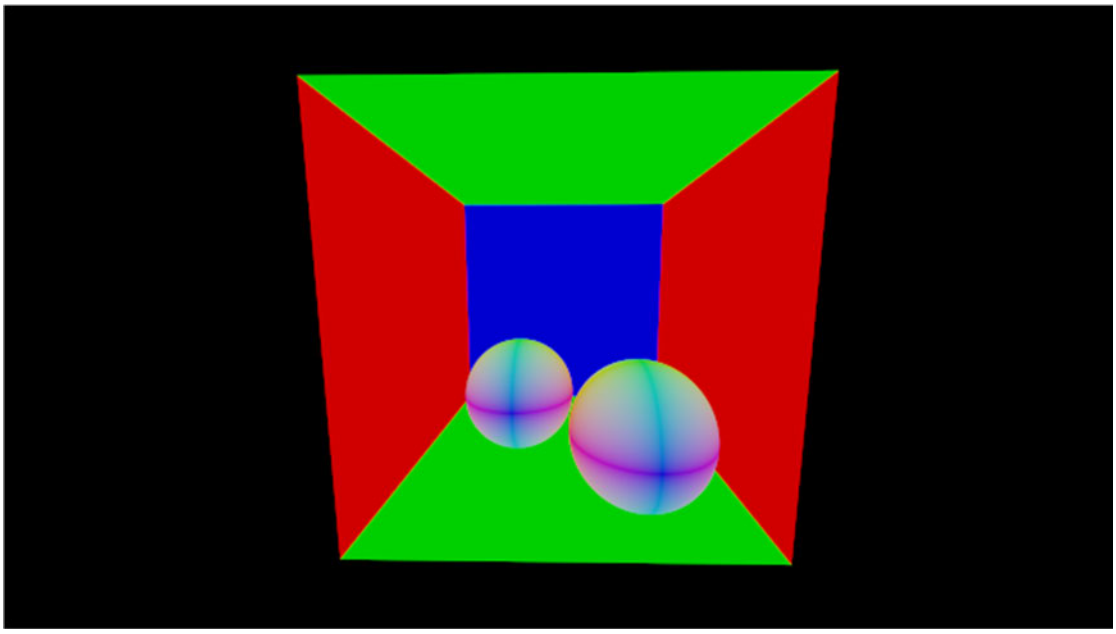
另外，当$\vec{O} \cdot \vec{N} > 0$，即$y > 0$时，入射光线$\vec{I}$的$y$分量应为$-cos\theta_i$。

四、实验结果

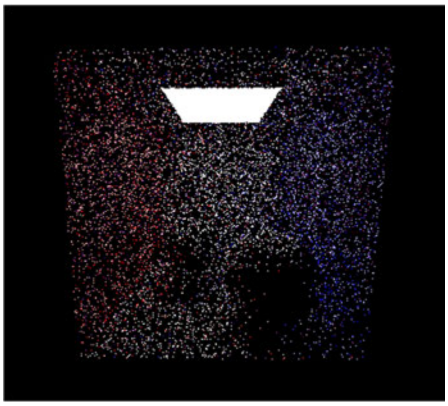
1）Generating Camera Rays



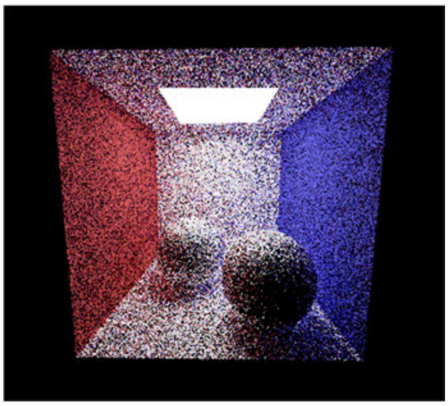
2）Intersecting Objects



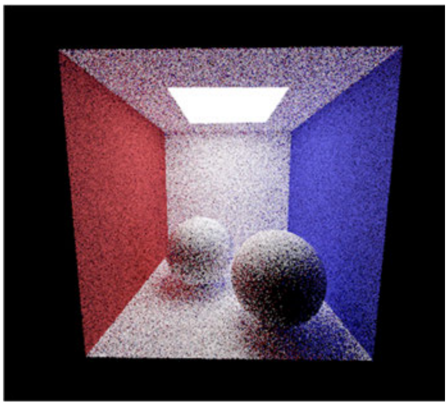
3）Path Tracing



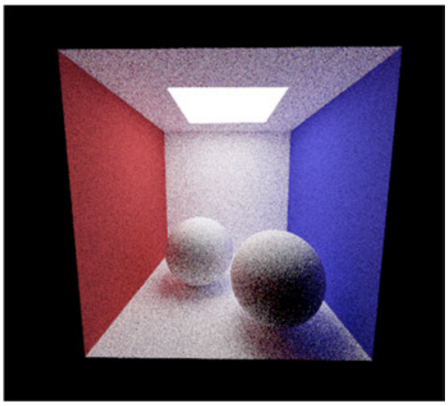
1 spp



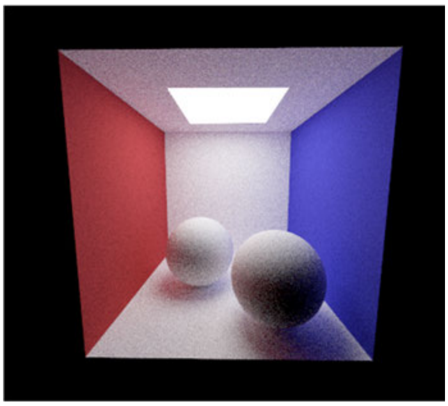
16 spp



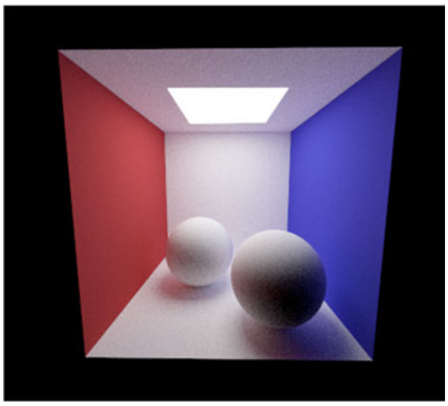
64 spp



256 spp

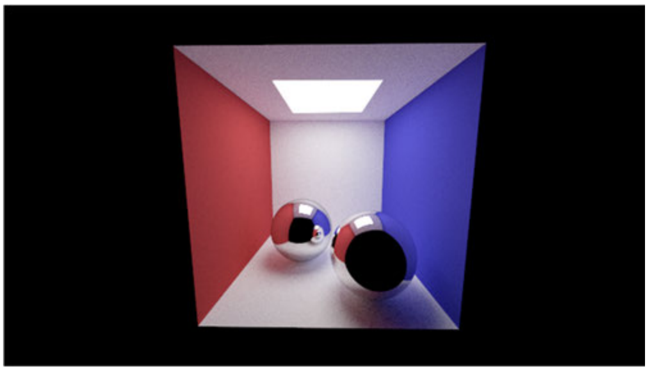


1024 spp

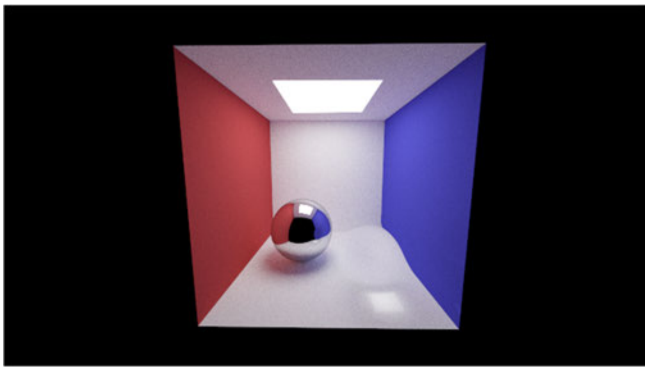


4096 spp

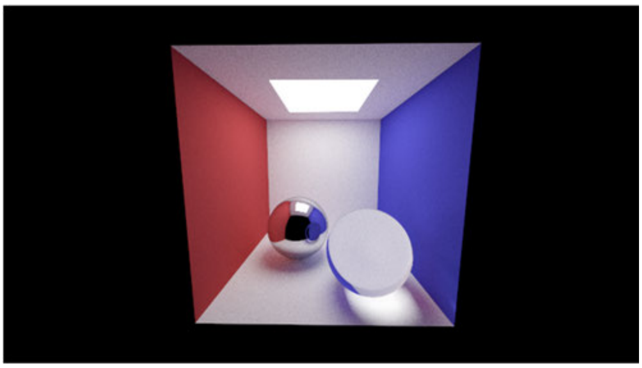
4）Materials



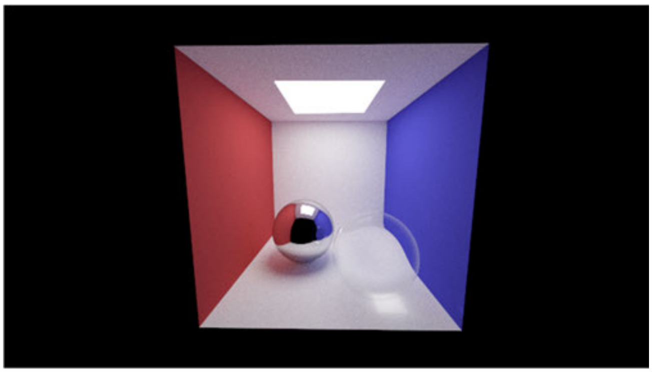
Reflection only



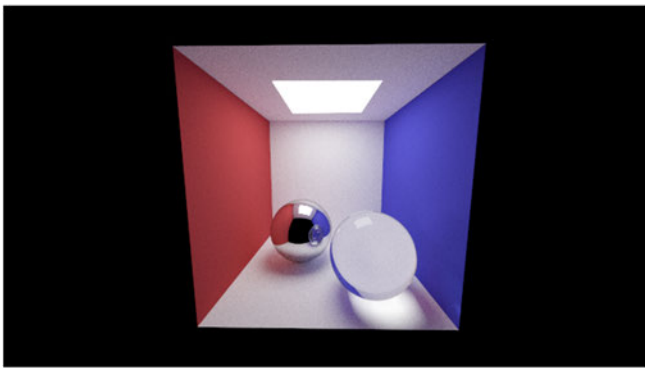
Refraction only，IOR=1



Refraction only，IOR=1.5



Glass，IOR=1



Glass，IOR=1.5

五、参考资料

1）Physically Based Rendering: From Theory to Implementation (pbr-book.org)

2）GAMES101-现代计算机图形学入门-闫令琪（bilibili.com）

3）Disney BSDF 深度解析 - 知乎 (zhihu.com)

4）全局光照(蒙特卡洛路径追踪) - 知乎 (zhihu.com)

5）重要性采样和多重重要性采样在路径追踪中的应用 - 知乎 (zhihu.com)

6）射线与球的相交测试 - 知乎 (zhihu.com)

7）Möller–Trumbore 算法 - 知乎 (zhihu.com)

8）光线的反射（reflect）与折射（refract） - 知乎 (zhihu.com)

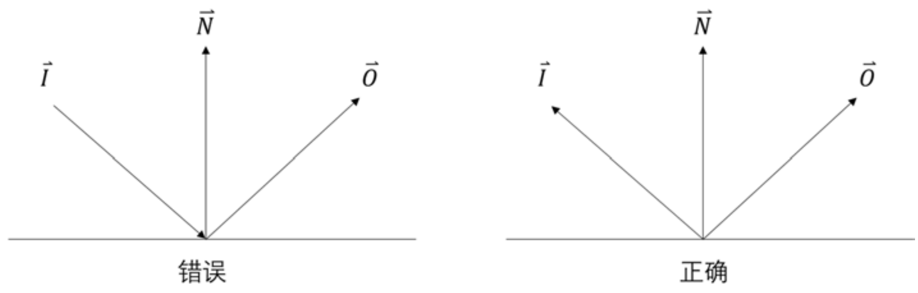
9）光的反射与折射——从 Snell、Fresnel 到 Schlick - 知乎 (zhihu.com)

六、遇到的困难与解决方案

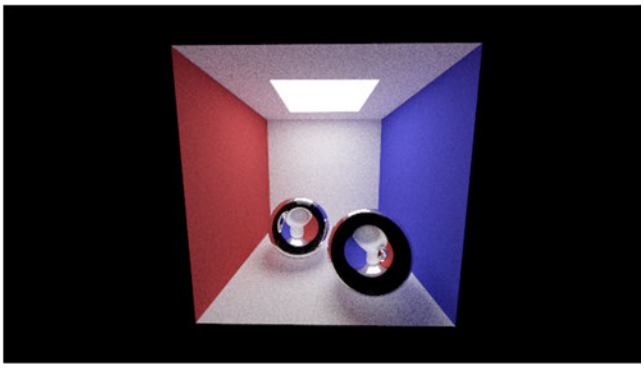
1）超采样：刚开始我在 Pathtracer::trace_pixel()函数中对单个像素进行 Pathtracer::n_samples 次超采样，并将采样的结果求平均后作为函数返回值。但在渲染过程中发现仅采样 32 条光

线就耗时 4 分钟左右，查阅代码后发现 Pathtracer::do_trace()函数对每个像素调用 samples 次 trace_pixel()函数进行采样并求平均值。因此，Pathtracer::trace_pixel()函数的实际工作是对传入的像素点坐标进行随机偏移后再采样，从而确保每次采样的光线不同。所以之前采样 32 条光线实际上相当于 32*32=1024 次采样。

2）局部坐标系下的光线方向：



反射的错误结果如下：



七、心得体会与课程建议

通过本次实验，我不仅学会了如何使用路径追踪算法实现全局光照，更重要的是，编程与调试的过程充分锻炼了我的查阅资料、算法实现转化的能力。总之，此次实验让我受益匪浅。此外，建议老师增加对需补全函数的详细功能描述及 Scotty3D 中部分类的介绍，如 Spectrum、BSDF 等。