

# 作业9 路径追踪 作业报告

李喆昊 PB17050941

皇家超跑	恶龙斯矛革
	

## 目录

1. 路径追踪
  - 原理介绍
    - 基于物理的光线渲染基础知识
    - 光线追踪算法
    - 蒙特卡洛采样方法
    - 路径追踪算法
  - 实现中的关键问题与解决方法
  - 测试结果
2. 环境贴图重要性采样
  - 原理介绍
  - 实现中的关键问题与解决方法
  - 测试结果
3. 总结反思
4. 参考文献

## 一. 路径追踪

### 1. 原理介绍

- 基于物理的光线渲染基础知识 (PBR: Physics Based Rendering)

为了能够使用计算机绘制出接近真实世界光影效果的图片，我们需要考虑物理上，光线是如何与物体相互作用然后进入人眼、造就色彩斑斓的美丽世界的。这里由于篇幅的限制，略去对几何光学、波动光学知识的介绍，仅介绍一些光线渲染技术中的物理概念。

#### ▪ 辐照率

光线的辐照率被定义为光线单位面积上的功率。空间位置 $x$ 上沿着方向 $w$ 的辐照率可以用符号 $L(x, w)$ 表示。

## ■ BRDF函数

现实世界中不同材质的物体对于光的反射模式千变万化。为了描述材质在空间位置 $x$ 对入射光线的反射情况，我们使用BRDF函数  $f_r(x, w_i \leftrightarrow w)$ 。

## ■ 渲染方程

渲染方程是光线渲染的核心，它给出了对光线渲染物理模型的准确描述，满足渲染方程的光线渲染算法可以保证结果的物理正确性。

$$L(x, w) = L_e(x, w) + L_r(x, w) \quad (1)$$

其中  $L_e(x, w)$  代表着物体接收到来自光源的全局光照的辐照率， $L_r(x, w)$  代表着物体接收到的来自别的非自发光物体的反射光（局部光照）的辐照度，其公式如下：

$$L_r(x, w) = \int_{\Omega_\infty} L_i(x, w_i) f_r(x, w_i \leftrightarrow w) \langle n_x, w_i \rangle dw_i \quad (2)$$

某一点的光照是来自全局光照与局部光照的叠加，如下图所示。



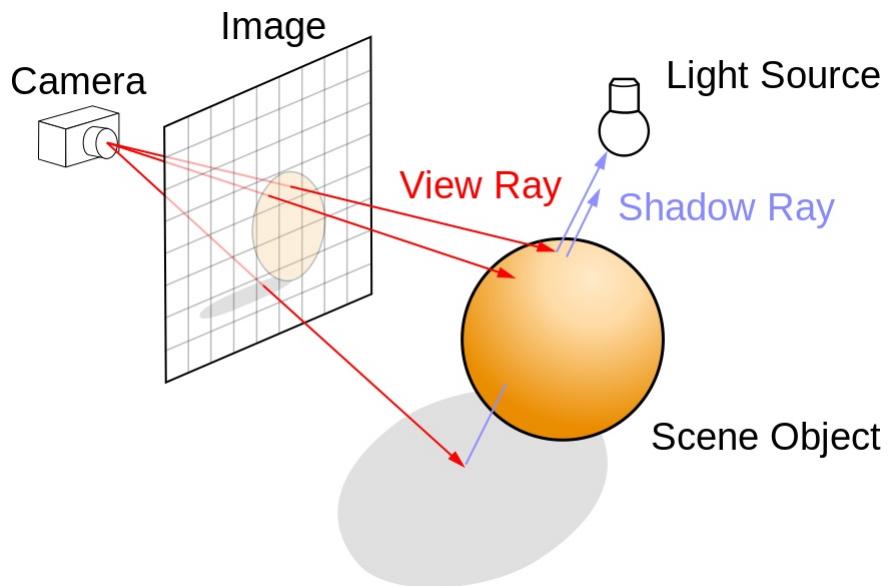
Direct illumination

Global illumination  
(Direct + Indirect)

可以看到在仅有全局光照的情况下，画面较暗，很多没有被光源直接照射到的物体的细节被隐藏在黑暗之中。而在有全局光照+局部光照的情况下，画面整体变亮，并且能够照亮左图并不能看到的物体细节。

## ○ 光线追踪算法

光线追踪算法是光线渲染早期的一种重要的渲染方法。正如其名，其核心思想是让光线从相机出发寻找光源。准确地说，从相机平面上每一个像素点发射出一条光线，该光线在场景中将经过一系列反射/折射，最终到达光源或者背景。最终到达背景的光在该像素点处呈现背景相应位置的颜色（乘以路径上的一系列因子），而到达光源的光在该处呈现光源颜色（同样地，乘以路径上的一系列因子）。



其原始算法 (Whitted-style) 主要过程如下。

```

color trace(point p, vector d, int step)
{
    color local, reflected, transmitted;
    point q;
    normal n;
    if(step > max) return(background_color);

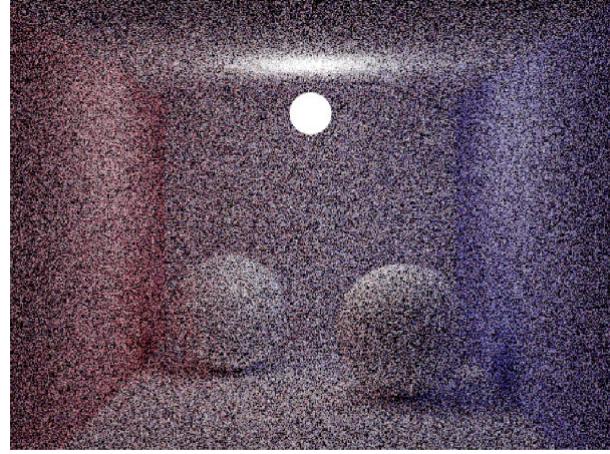
    q = intersect(p,d,status);
    if(status == light_source)
        return(light_source_color);
    if(status == no_intersection)
        return(background_color);

    n = normal(q);
    r = reflect(q,n);
    t = transmit(q,n);
    local = phong(q,n,r);
    reflected = trace(q,r,step+1);
    transmitted = trace(q,t,step+1);
    return(local+reflected+transmitted);
}

```

但光线追踪算法主要存在的缺陷有：

1. 当光源面积较小时，从相机发射出的光线最终到达光源的概率随之变小，导致很多光线因为无法到达光源被浪费，形成大量噪点。如下图所示。



64 samples per pixel

2. 光线追踪算法实际上并不是在求解渲染方程，而是基于简单的光线反射/折射原理，因此结果的物理正确性不能得到保证。

为了解决这些问题，研究者提出了下面将介绍的基于蒙特卡洛采样方法的路径追踪算法。

- 蒙特卡洛采样方法

蒙特卡洛采样方法是利用概率统计知识求解数值积分的一种方法。

问题描述如下：

我们需要求解如下所示的数值积分。

$$I = \int_a^b f(x) dx \quad (3)$$

可以使用计算机通过采样的方法对该积分的值进行估计：我们的目标是构造随机变量 $\langle I \rangle$ ，得到对 $I$ 的一个无偏估计，即 $E[\langle I \rangle] = I$ 。如果我们已经知道了(3)式中的随机变量 $X$ 的概率分布 $p(X)$ ，

则可以令  $\langle I \rangle := \frac{f(X)}{p(X)}$ ，使得

$$E[\langle I \rangle] = E\left[\frac{f(X)}{p(X)}\right] = \int_a^b \frac{f(x)}{p(x)} p(x) dx = I \quad (4)$$

这样，我们就得到了 $I$ 的一个无偏估计。

但问题是：实际中，我们往往不知道 $X$ 的概率分布函数 $p(X)$ ，只能对其进行估计。最常使用的估计有均匀分布等。假设我们共采样 $n$ 个样本 $x_j, j = 1, 2, \dots, n$ 。 $\hat{I}_j := \frac{f(x_j)}{p(x_j)}$ ，则我们就得到了对原积分的一个估计。

$$\bar{I} := \frac{1}{n} \sum_{j=1}^n \hat{I}_j \quad (5)$$

该估计满足：

$$\int f(x) dx = \lim_{n \rightarrow \infty} \bar{I} = \lim_{n \rightarrow \infty} \frac{1}{n} \sum_{j=1}^n \hat{I}_j \quad (6)$$

根据不同采样点的重要性，我们还可以引入参数 $w$ ，满足：

$$\sum_{i=1}^n w_i = 1 \quad (7)$$

则：

$$\langle I \rangle := \sum_{i=1}^n w_i(x_i) \frac{f(x_i)}{p_i(x_i)} \quad (8)$$

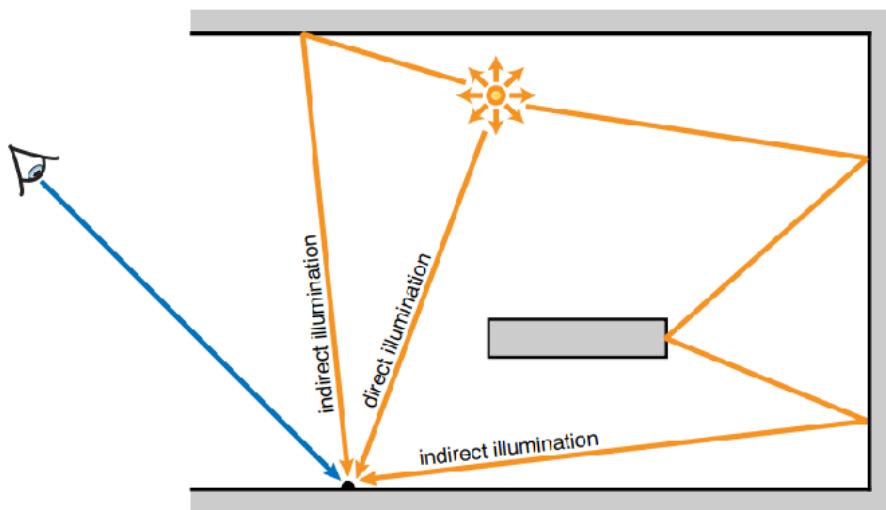
以上就是使用蒙特卡洛采样算法求解数值积分的核心思想。

- 路径追踪算法

路径追踪算法是基于蒙特卡洛采样算法的光线渲染方法，其核心思想与光线追踪算法一致，也是让光线从相机出发寻找光源，但具体做法是：从相机平面的每一个像素点发射出多条光线寻找光源，光线在与场景中物体相交，发生反射时，按照预先设定的概率分布函数从半球面选择一个方向出射（称为BRDF采样），并在最终表达式中除去这个概率因子。其余部分与光线追踪一致。

路径追踪算法的本质是将多条光路对光照效果的贡献按照它们出现的概率结合在了一起，实现了渲染方程。

如下图所示。



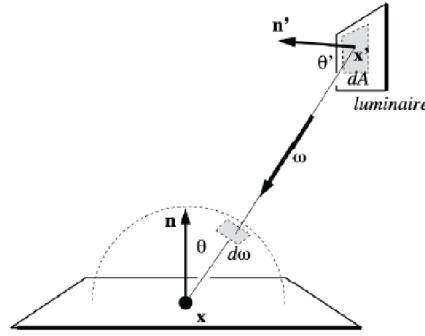
值得一提的是，为了能够让算法中的光线反射的迭代能够终止，我们可以引入Russian Roulette的技巧：通过每次迭代时生成一个随机数来决定是否进行下一次光线的反射，阻止了无限迭代的可能。

其算法过程如下所示。

```
shade(p, wo)
    Test Russian Roulette with p_rr, if fail then return 0.0;
    Randomly choose one direction wi with pdf(wi);
    Trace a ray r(p, wi);
    if ray r hit the light:
        return L_i * f_r * cos / pdf(wi) / p_rr;
    else if ray r hit an obj at position q:
        return shade(q, -wi) * f_r * cos / pdf(wi) / p_rr;

ray_generation(camPos, pixel)
    Uniformly choose N sample positions within pixel;
    pixel_radience = 0.0;
    for each sample in the pixel:
        Shoot a ray r(camPos, cam_to_sample_direction);
        If ray r hit the scene at p:
            pixel_radience += 1/N * shade(p, cam_to_sample_direction);
    return pixel_radience;
```

但值得注意的是，路径追踪算法并没有解决当光源面积很小时，发射出的光线不能够寻找到光源的问题。解决这一问题的一个很自然的思路是：既然从相机出发发射光无法到达光源，那么能否直接从光源出发发射光到达物体，直接计算全局光照？根据这一思路，我们可以使用对光源采样的策略提高路径追踪算法的效率。



此时由几何关系，将在BRDF半球上的积分转换到面光源所在的平面A上：

$$\begin{aligned} L_o(x, w_o) &= \int_{\Omega} L_i(x, w_i) f_r(x, w_i, w_o) \cos\theta dw_i \\ &= \int_A L_i(x, w_i) f_r(x, w_i, w_o) \frac{\cos\theta \cos\theta'}{\|x' - x\|^2} dA \end{aligned} \quad (9)$$

改进后的路径追踪算法中 shade 的主要过程为：

```

shade(p, wo)
    // direct light, sampling the light
    Uniformly sample the light at x with pdf_light = 1/A;
    Check visibility, if invisible L_dir = 0;
    else L_dir = L_i * f_r * cos(theta1) * cos(theta2) / |x - p|^2 /
    pdf_light;

    // indirect light, BRDF sampling
    L_indir = 0.0;
    Test Russian Roulette with p_rr
    Randomly choose one direction wi with pdf(wi);
    Trace a ray r(p, wi);
    if ray r hit an non-emitting obj at position q:
        return shade(q, -wi) * f_r * cos / pdf(wi) / p_rr;

    return L_dir + L_indir;

```

## 2. 实现中的关键问题与解决方法

- 环境光检查可见性的方法

由于环境光设置其光源在无穷远处，所以不能使用助教提供的

`IntersectorVisibility::Instance().Visit` 函数检查可见性，这里我采取的方式是从原交点处发射一束光线，检查其是否与其他物体（面光源、非自发光物体）相交，如果相交，则环境光源被阻挡，否则环境光源可见。代码如下：

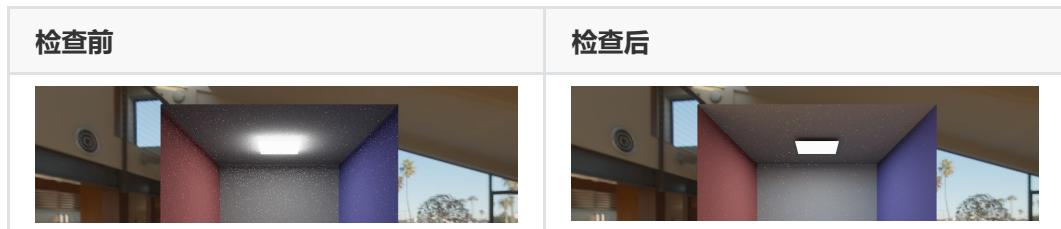
```

auto new_intersection = IntersectorClosest::Instance().visit(&BVH,
    rayf3(intersection.pos, (-
sample_light_rst.n).cast_to<vecf3>()));
if (!new_intersection.IsIntersected())
{
    // visible
}

```

- 当前点是否在面光源背面的判断方法

通过判断cos的正负号可以判断点是否在面光源的背面，结果如下所示。



### 3. 测试结果

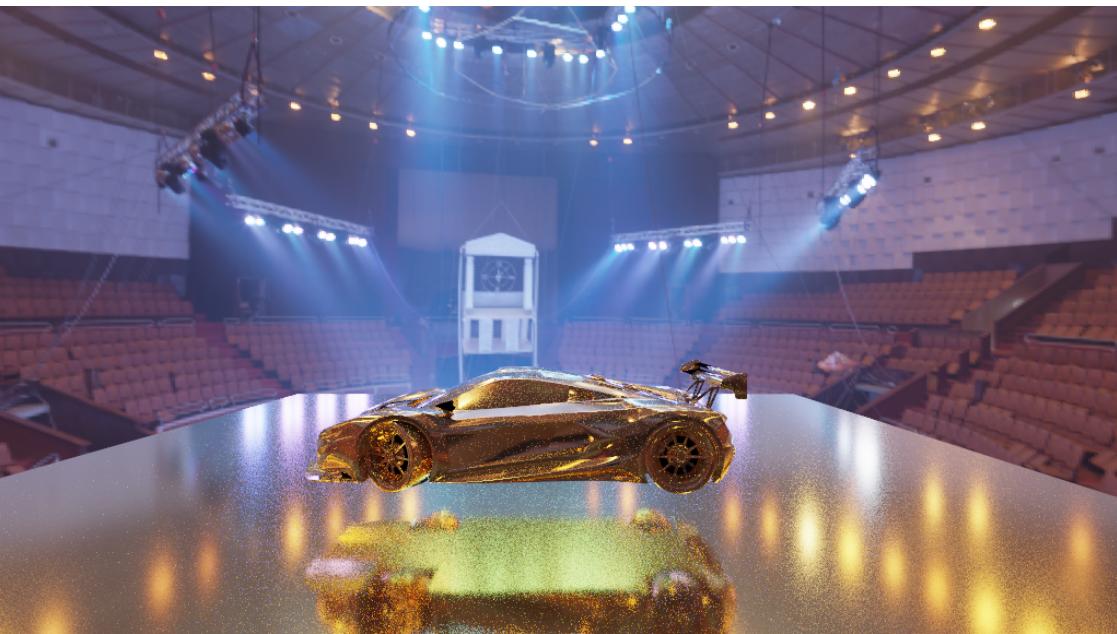
康奈尔盒



星空下的球



皇家超跑 (个人最满意的图)



街头超跑



恶龙斯矛革和它占有的宝物



另一只镜面龙斯矛革



不同材质（铜，金，银）的M4A1（模仿CSGO（反恐精英）中沙漠小城的场景）



## 二. 环境贴图重要性采样

### 1. 原理介绍

环境贴图中，环境光源的分布往往是不均匀的。因此在全局光照中计算环境光的部分，我们可以对环境贴图按照像素的亮度进行重要性采样，如下图所示。



构建抽样表可以使用别名法（[参考资料](#)）使得采样的时间复杂度为 $O(1)$ .

在别名法中我们将构建环境贴图中每个像素点的采样概率表 $P_{img}$ ， $p_{img(i,j)}$ 与 $p(w_i)$ 的转换关系如下：

$$p(\omega_i) = \frac{wh}{2\pi^2 \sin \theta} p_{img}(i, j) \quad (10)$$

其中 $w, h$ 分别为环境贴图的宽度与高度， $\theta$ 为 $w_i$ 与交点法向量的夹角。

### 2. 实现中的关键问题与解决方法

- 使用别名法构建抽样表

使用了 `vector<pair<std::pair<int, int>, pair<int, int>> >`  
`env_light_alias_table_idx` 存放别名法中的像素点的下标，使用 `vector<pair<float, float>> env_light_alias_table_p` 存放别名法中的概率表。

采样时只需要生成两个随机数，为  $O(1)$  时间。

```
auto rand_n = int(rand01<float>() * (env_h * env_w - 1));
auto rand_p = rand01<float>();
std::pair<int, int> sample_idx;
if (rand_p <= std::get<0>(env_light_alias_table_p[rand_n]))
{
    sample_idx = std::get<0>(env_light_alias_table_idx[rand_n]);
}
else sample_idx = std::get<1>(env_light_alias_table_idx[rand_n]);
```

- 环境贴图像素坐标与入射光线向量之间的转换

转换过程：环境贴图像素坐标  $(i, j) \rightarrow$  环境贴图纹理坐标  $(u, v) \rightarrow$  球面角坐标  $(\theta, \phi) \rightarrow$  入射光线向量  $w_i$

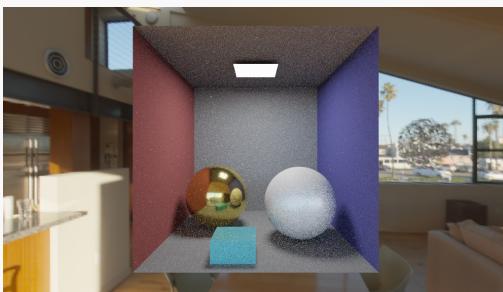
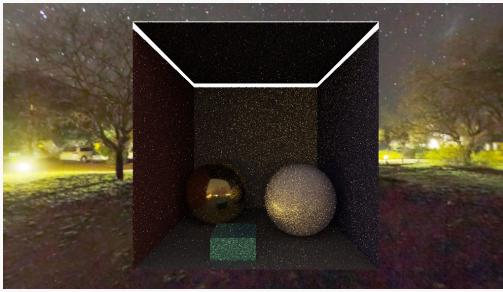
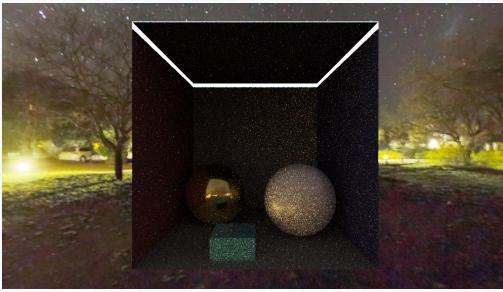
关系为：

- $i \in [0, w]$  ,  $w$  是图像宽度
- $j \in [0, h]$  ,  $h$  是图像高度
- $u, v \in [0, 1]$  , 且  $u = i/w$  ,  $v = j/h$
- $\theta \in [0, \pi]$  ,  $\theta = \pi(1 - v)$
- $\phi \in [0, 2\pi]$  ,  $\phi = 2\pi u$
- $\omega_i = (\sin \theta \sin \phi, \cos \theta, \sin \theta \cos \phi)$

### 3. 测试结果

为验证环境贴图重要性采样的效果，进行了两组对比实验。

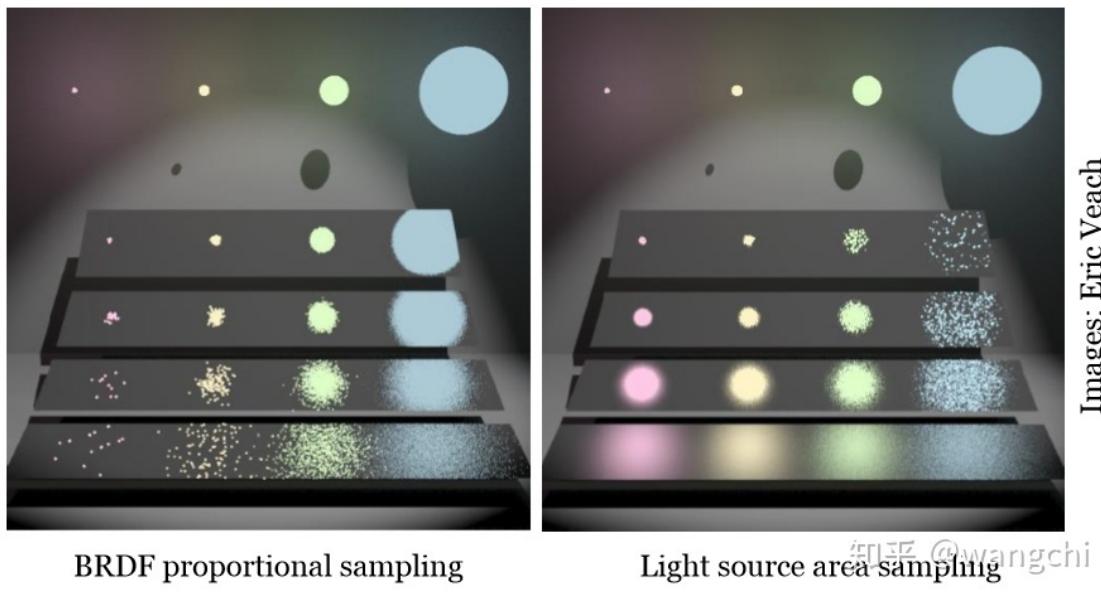
没有进行环境光重要性采样（对照组）	进行了环境光重要性采样（实验组）
Cornell box (40spp)	Cornell box (40spp)

没有进行环境光重要性采样（对照组）	进行了环境光重要性采样（实验组）
	
夜晚环境中，去掉顶光源的Cornell box (40spp)	夜晚环境中，去掉顶光源的Cornell box (40spp)
	

可以看到，相比于对环境光进行均匀采样，进行了环境光重要性采样后的画面某些地方比原来更暗（而由于环境光源分布的不均，这些地方确实应该更暗一些）。

#### 4. 补充：对面光源进行重要性采样 & 多重重要性采样

实际中仅使用BRDF采样或仅使用对光源的重要性采样结果都不能达到完美的效果，如下图所示。



通过多重重要性采样可以解决上面的问题，如下图所示。本次作业由于时间关系没有进行相关的对比实验。

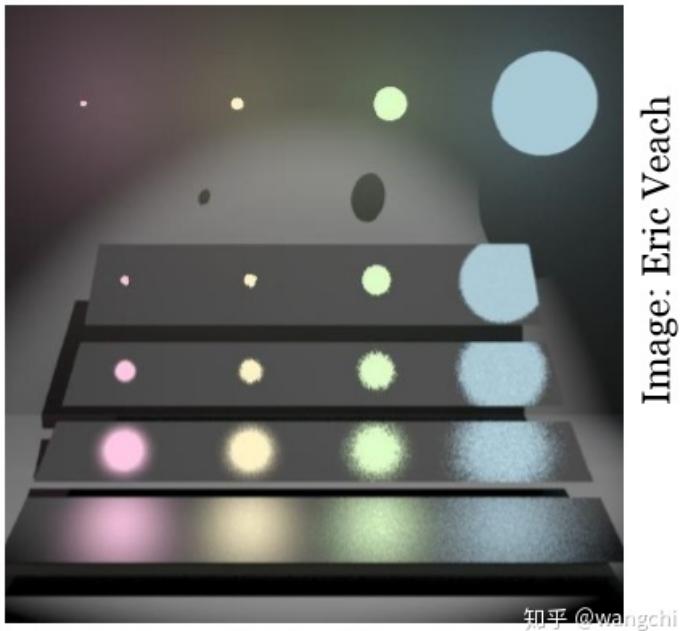


Image: Eric Veach

知乎 @wangchi

### 三. 总结反思

- 总结：通过本次作业，查阅了大量相关文献，理解了光线追踪、路径追踪、蒙特卡洛等方法的思想与原理，增加了对渲染的兴趣。
- 反思：
  - 在做事过程中应当舍弃过度的完美主义，而更应该追求效率与速度，花费大量时间的“完美”结果实际上是一种低效的结果。
  - 一开始对别名法的一些细节不太理解（如环境贴图像素坐标与入射光线向量之间的转换），在课程群内询问无果。在上网搜索后发现助教的知乎文章中对相关方法与变量的含义进行了解释，解决了我很多的疑惑。在遇到问题时询问他人的同时自己也应当充分使用google与知乎等平台，先做充分的理解与调研。

### 四. 参考文献

1. 基于物理着色：BRDF <https://zhuanlan.zhihu.com/p/21376124>
2. 《Real-Time Rendering 3rd》提炼总结 第九章·全局光照:光线追踪、路径追踪与GI技术进化编年史 <https://zhuanlan.zhihu.com/p/29418992>
3. 详解球面环境映射 - Spherical Environment Mapping <https://zhuanlan.zhihu.com/p/84494845>
4. 深入理解 PBR/基于图像照明 (IBL) <https://zhuanlan.zhihu.com/p/66518450>
5. 金属, 塑料, 傻傻分不清楚 <https://zhuanlan.zhihu.com/p/21961722>
6. 一篇光线追踪的入门 <https://zhuanlan.zhihu.com/p/41269520>
7. 时间复杂度O(1)的离散采样算法—— Alias method/别名采样方法 <https://blog.csdn.net/haolexiao/article/details/65157026>
8. 多重重要性采样 (很好的blog) <https://airguanz.github.io/2018/10/15/multiple-importance-sampling.html>
9. 光线追踪器Ray Tracer：进阶篇 (很好的图形学博主) <https://yangwc.com/2019/05/23/RayTracer-Advance/>