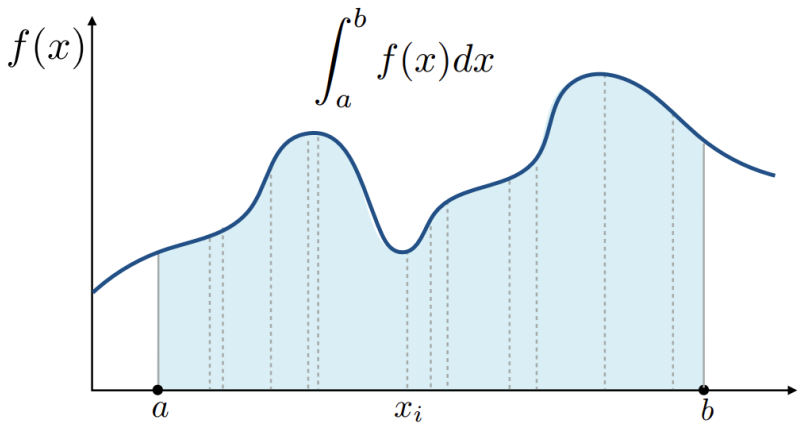
Lecture 16

Monte Carlo Integration 蒙特卡洛积分

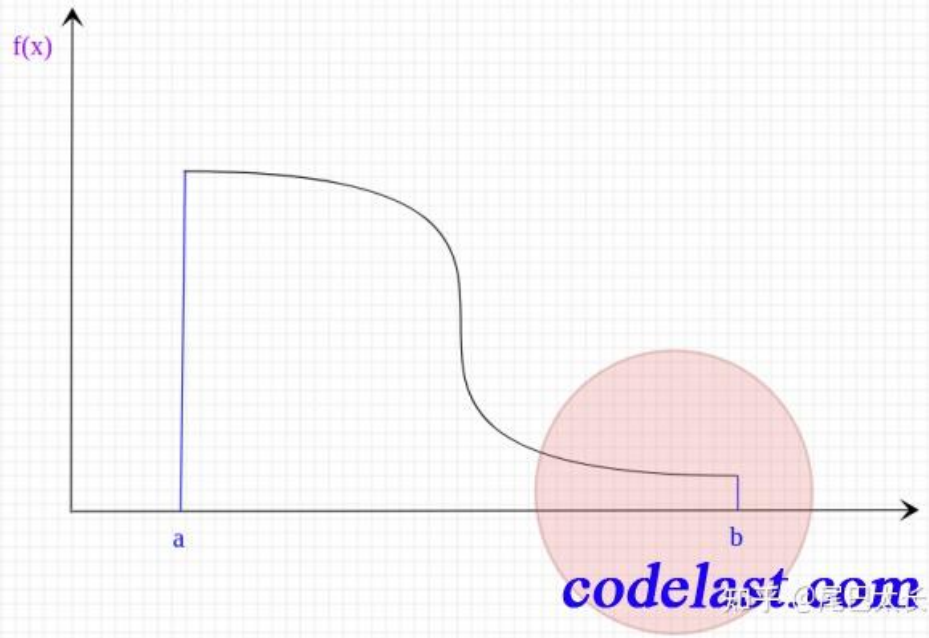
对于一个积分 $\int_a^b f(x)dx$ ，要直接的积分是较难的，为了近似求解其结果，需要采用平均采样的方式来得出结果，最常见的就是微积分中不断划分小区间，近似求得积分结果。



重要性采样

$$\begin{aligned}\int_a^b f(x)dx = \\ &= \frac{1}{N}[(b-a)*f(X_1)+(b-a)*f(X_2)+(b-a)*f(\\ &= \frac{b-a}{N}\sum_{i=1}^{N} f(X_i) \\ &= \frac{1}{N}\sum\frac{f(X_i)}{p(X_i)}\end{aligned}$$

此处 $p(X_i)$ 在均匀采样下，就等于 $\frac{1}{b-a}$ ，而这是最简单的均匀方式，如果对于函数值变化较快的函数，均匀采样的效果并不好，如下图



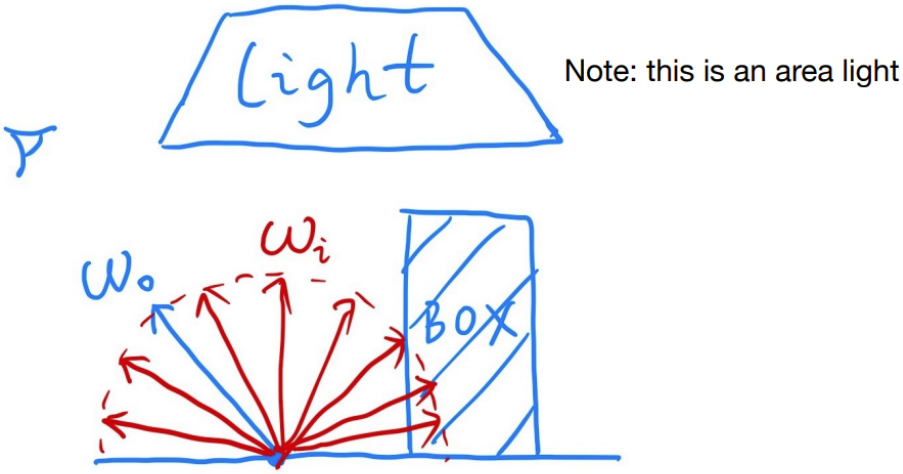
要想在少采样的前提下获得好的近似结果，需要在该函数左侧 $f(x)$ 较大的区域采样多次，而在右侧圆圈内，尽量少采样。 **note: 所以对于 p(x) 的选择会影响采样结果**

Path Tracing 路径追踪

A simple Monte Carlo Solution

- 对于 Lecture 13 > Recursive(Whitted-Style) Ray Tracing 中的光线追踪，设定遇到镜面物体则反射，遇到diffuse物体则停止，在物理上这不正确。
- 对于 Lecture 15 > BRDF 中的**渲染方程**，我们可以借用其形式，并使用 **蒙特卡洛积分** 对其积分形式进行化简，得到一个可求的最终式。

此处我们设定在球面上均匀采样，并且设置从像素点往外出射光线，如下图



$$L_o(p,\omega_o) = \int_{\Omega^+} L_i(p,\omega_i) f_r(p,\omega_i,\omega_o)(n\cdot\omega_i)\,\mathrm{d}\omega_i$$

Monte Carlo integration: $\int_a^b f(x)\,\mathrm{d}x \approx \frac{1}{N}\sum_{k=1}^{N}\frac{f(X_k)}{p(X_k)} \quad X_k \sim p(x)$

What's our "f(x)"? $L_i(p,\omega_i) f_r(p,\omega_i,\omega_o)(n\cdot\omega_i)$

What's our pdf? $p(\omega_i) = 1/2\pi$

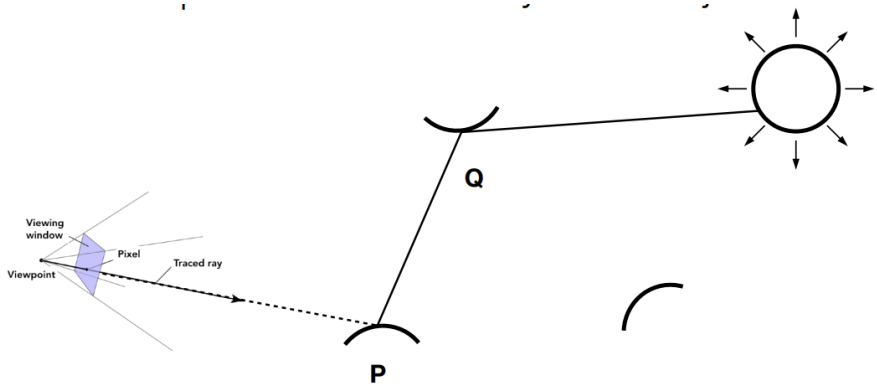
根据蒙特卡洛积分方法的化简，得到表达式

$$L_o(p,\omega_o) \approx \frac{1}{N}\sum_{i=1}^{N}\frac{L_i(p,\omega_i) f_r(p,\omega_i,\omega_o)(n\cdot\omega_i)}{p(\omega_i)}$$

其伪代码如下

```
shade(p, wo)
    Randomly choose N directions wi~pdf
    Lo = 0.0
    For each wi
        Trace a ray r(p, wi)
        If ray r hit the light
            Lo += (1 / N) * L_i * f_r * cosine / pdf(wi)
    Return Lo
```

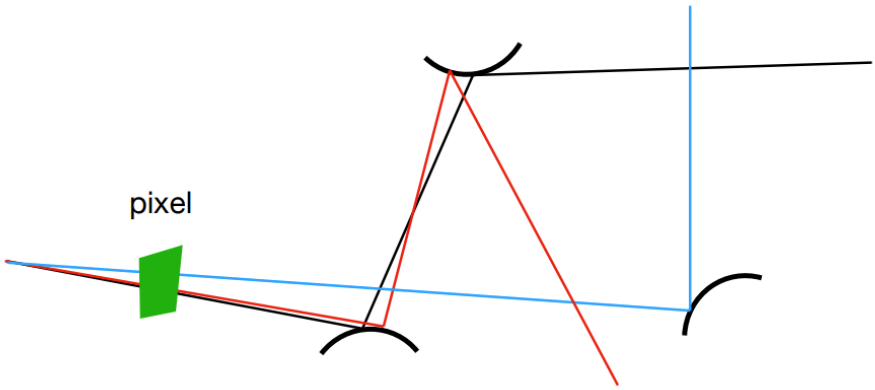
全局光照 Global illumination 对于光线从物体反射而来的情况，我们也将其视作一种 间接的光线，与上小节一样，计算穿过物体反向回来的 radiance 即可



其伪代码如下

```
shade(p, wo)
    Randomly choose N directions wi~pdf
    Lo = 0.0
    For each wi
        Trace a ray r(p, wi)
        If ray r hit the light
            Lo += (1 / N) * L_i * f_r * cosine / pdf(wi)
        Else If ray r hit an object at q
            Lo += (1 / N) * shade(q, -wi) * f_r * cosine
            / pdf(wi)
    Return Lo
```

问题1 但其中光线在多个物体间的弹射会使得总光线呈 **指数级上升**，这对于计算是不可接受的，解决方案只能是一个像素上一处发出一条光线，并对该像素上重复取多条光，平均其结果。



其伪代码如下

```
ray_generation(camPos, pixel)
    Uniformly choose N sample positions within the pixel
    pixel_radiance = 0.0
    For each sample in the pixel
        Shoot a ray r(camPos, cam_to_sample)
        If ray r hit the scene at p
            pixel_radiance += 1 / N * shade(p, sample_to_cam)
    Return pixel_radiance
```

问题2 对于光线不断弹射的情况，我们并没有设置光线结束弹射的条件，这也会导致循环无法结束。 这里设置一个 **俄罗斯轮盘赌（RR）**，设定一个概率 P，选择光线是停止弹射/继续弹射。即有 P 的概率得到结果

为 L_0/P ,有 $(1-P)$ 的概率得到的结果为 0。

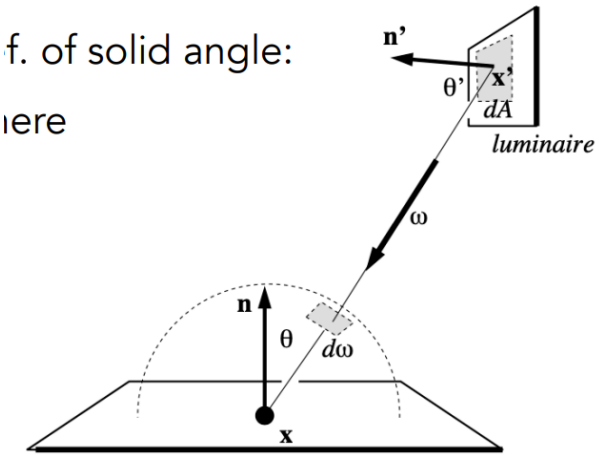
```
shade(p, wo)
    Manually specify a probability P_RR
    Randomly select ksi in a uniform dist. in [0, 1]
    If (ksi > P_RR) return 0.0;

    Randomly choose ONE direction wi~pdf(w)
    Trace a ray r(p, wi)
    If ray r hit the light
        Return L_i * f_r * cosine / pdf(wi) / P_RR
    Else If ray r hit an object at q
        Return shade(q, -wi) * f_r * cosine / pdf(wi) / P_RR
```

问题3 对于从像素着色点不断发出光线的方式，对于一些面积较小的面光源，会导致大量射线的浪费，我们可以将 **对于着色点的立体角进行积分** 变换成 **对于光源表面积进行积分**，此处需要进行 **变量替换**

$$L_0 = \int f_r \, cosd\omega$$

$$d\omega = \frac{dAcos\theta'}{||x' - x||^2}$$



此处可以将渲染公式进一步转换

$$\begin{aligned} L_o(x, \omega_o) &= \int_{\Omega^+} L_i(x, \omega_i) f_r(x, \omega_i, \omega_o) \cos\theta \, \mathrm{d}\omega_i \\ &= \int_A L_i(x, \omega_i) f_r(x, \omega_i, \omega_o) \frac{\cos\theta \cos\theta'}{\|x' - x\|^2} \, \mathrm{d}A \end{aligned}$$

Sampling the Light

对于上述公式，我们可以做一个总结

- 对于光源：我们在光源上做积分，求得其对着色点的影响
- 对于非光源：我们在着色点上发出射线，打到光源则忽略，打到物体则递归计算其影响，并使用 RR 处理光线是否继续弹射的问题。

伪代码如下

```
shade(p, wo)
    # Contribution from the light source.
    Uniformly sample the light at x' (pdf_light = 1 / A)
    L_dir = L_i * f_r * cos θ * cos θ' / |x' - p|^2 / pdf_light

    # Contribution from other reflectors.
    L_indir = 0.0
    Test Russian Roulette with probability P_RR
    Uniformly sample the hemisphere toward wi (pdf_hemi = 1 / 2pi)
    Trace a ray r(p, wi)
    If ray r hit a non-emitting object at q
        L_indir = shade(q, -wi) * f_r * cos θ / pdf_hemi / P_RR

    Return L_dir + L_indir
```

并且对于光源上的积分，需要考虑光源与着色点之间是否连线中无其他物体的阻挡，如下

```
# Contribution from the light source.
L_dir = 0.0
Uniformly sample the light at x' (pdf_light = 1 / A)
Shoot a ray from p to x'
If the ray is not blocked in the middle
    L_dir = …
```

Now path tracing is finally done!