CS667 **Lecture 10: Monte Carlo Solutions to Rendering Equations** 24 Feb 2005

Kelvin Cai Lecturer: Steve Marschner

---

1 Introduction

1. Monte Carlo path tracing basics (for vacuum and surfaces)
2. Monte Carlo path tracing for volumes. To solve the volumetric Rendering Equation

2 Rendering Equation

$$L_e(x,w) = L_e^o(x,w) + \int_{\mathbb{H}^2} f_r(x,w,w')L_i(x,w')d\mu(w')$$

We can treat the Rendering Equation as integration problem, using Monte Carlos.

$$\tilde{L_e}(x,w) = L_e^o(x,w) + \frac{1}{N}\sum_{i=1}^{n}\frac{f(w_i)}{p(w_i)} \qquad \text{or}$$

$$\tilde{L_e}(x,w) = \frac{1}{N}\sum_{i=1}^{n}\left[L_e^o(x,w) + \frac{f_r(x,w,w_i)L_i(x,w_i)}{p(w_i)}\right]$$

However, we don't know L_i. But it is nice that this estimator is linear in L_i, so if we substitute an unbiased estimator for L_i, then the expected value of $\tilde{L_e}(x,w)$ will still be correct:

$$\tilde{L_e}(x,w) = \frac{1}{n}\sum_{i=1}^{n}\left[L_e^o(x,w) + \frac{f_r(x,w,w_i)\tilde{L_i}(x,w_i)}{p_\mu(w_i)}\right]$$

3 Path tracing

One could do this using $p \equiv \frac{1}{\pi}$ (uniform in projected solid angle). Result:

```
func. radianceEstimator(x,w)
        surface, point, normal = traceRay(x,w)
        result = sufrace.emittedRadiance(point,-w)
        w' = chooseDirection(normal)
        result += π * sufrace.brdfValue(point,w,-w) * radianceEstimator(point, w')
        return result
```

```
func. pixelValue(i,j)
        result = 0
        for i=1 to N
            x, w = generateRay(i,j)
            result += radianceEstimator(x,w) / N
        return result
```

This is called *path tracing* because there is no branching. Only thing missing is termination condition.

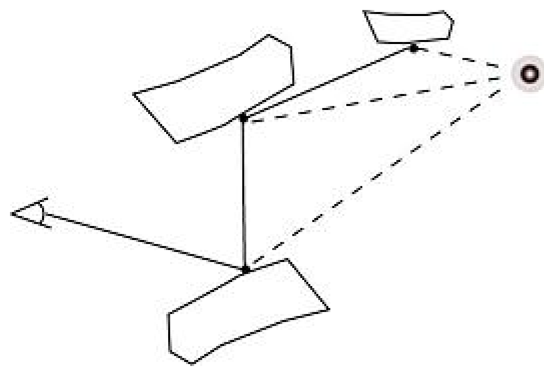

Figure 1: Path-tracing algorithm

4 Russian Roulette

How to stop the recursion ?

1. Wait until we hit something with $f_r = 0$, stop there (eg. light source, background, etc.)
2. Stop after a certain depth, but this is biased!
3. Russian Roulette

Russian Roulette: a graphic noun for a simple idea.

Suppose we have a way to compute an unbiased estimator of randomw variable X. $E\{x_i\}$ is the value we want. To avoid always evaluating x_i, replace X with $\frac{1}{r}RX$ where

$$R = \begin{cases} 0 & \text{probability } (1-r) \\ 1 & \text{probability } r \end{cases}$$

$$E\{R\} = r$$

$$E\{\frac{1}{r}RX\} = \frac{1}{r}\underbrace{E\{R\}E\{X\}}_{independence} = E\{X\}$$

5 Direct Lighting and Indirect Lighting

In this form, the path tracer is not much good for generating images in our lifetime. As we've observed, when a small light source provides a substantial fraction of the illumination, sampling the whole hemisphere won't do well. A standard optimization is to seperate out direct lighting.

Write L_i as L_i^o(due to emission)+L_i^r(due to reflection) then

$$\begin{aligned} L_e(x,w) &= L_e^o(x,w) + \int_{\mathbb{H}^2} f_r(x,w,w')[L_i^o(x,w') + L_i^r(x,w')]d\mu(w') \\ &= L_e^o(x,w) + \int_{\mathbb{H}^2} fr(x,w,w')L_i^o(x,w')d\mu(w') && \longleftarrow \text{direct part} \\ &+ \int_{\mathbb{H}^2} fr(x,w,w')L_i^r(x,w')d\mu(w') && \longleftarrow \text{recursive part} \end{aligned}$$

Usually, we solve the direct lighting part by changing the integral to area form. Result:

```
func. radianceEstimator(x,w) {
    return emittedRadiance(x,w) + indirectRadianceEstimator(x,w)
}

func. reflectedRadianceEstimator(x,w) {
    return directRadianceEstimator(x,w) + indirectRadianceEstimator(x,w)
}

func. directRadianceEstimator(x,w) {
    y = lightSourcs.choosePoint() // uniform wrt. area
    return sourceArea * visible(x,y) * G(x,y) * emittedRadiance(y, dir(y,x))
}

func. indirectRadianceEstimator(x,w) {
    w' = chooseDirection(normal(x)) // uniform wrt. proj. solid angle
    y = traceRay(x,w)'
    return pi * brdf(x,w,w') * reflectedRadianceEstimator(y,-w')
}
```

Further crucial optimization: importance sampling by BRDF when appropriate

6 Path Tracing for Volumetric Media (Homogeneous)

Similarly to surface path tracing. We can start with the integral and recursively expand again.

Assumptions to avoid extra confusion :

- medium enclosed by surface;
- surfaces only outside volume.

$$L(x,w) = \alpha(x,y)L_e(y,w) + \sigma_s \int_y^x \alpha(x',x) \int_{4\pi} p(x',w,w')L(x',w')dw'dx'$$

Monte Carlo approach is to rewrite this as an estimator

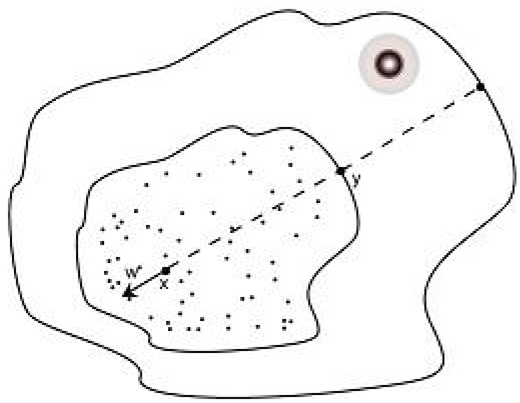

Figure 2: Volumetric Rendering

$$\tilde{L_e}(x,w) = \alpha(x,y)L_e(y,w) + \frac{\sigma_s}{N}\sum_{i=1}^{N}\frac{\alpha(x_i,x)\int_{4\pi} p(w,w')L(x_i,w')dw'}{p(x_i)}$$

For homogenous, we can compute α directly : $\alpha(x,y) = e^{-\sigma_s||x-y||}$.
But the integral at center we can't. So use an estimator.

$$\tilde{L_e}(x,w) = \alpha(x,y)L_e(y,w) + \frac{\sigma_s}{MN}\sum_{i=1}^{N}\frac{\alpha(x_i,x)}{p(x_i)}\sum_{j=1}^{M}\frac{p(w,w_j)\tilde{L}(x_i,w_j)}{p(w_j)} \longleftarrow \text{similary, use estimator for } L$$

This double sum could also be arrived at by thinking of the double integral:

$$L(x,w) = \alpha(x,y)L_e(y,w) + \sigma_s\int_y^x\int_{4\pi}\alpha(x',x)p(x',w,w')L(x',w')dw'dx'$$

$$\tilde{L}(x,w) = \alpha(x,y)L_e(y,w) + \frac{\sigma_s}{N}\sum_{k=1}^{N}\frac{\alpha(x_k,x)p(w,w_k)\tilde{L}(x_k,w_k)}{p(x_k,w_k)}$$

The difference is between stopping at one place and obseving a bunch at directions vs. looking at one direction per stop:

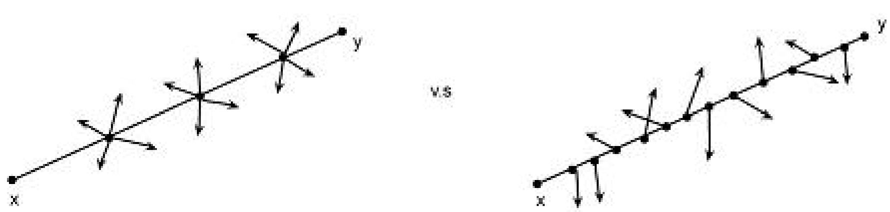

Figure 3: Two different volumetric path-tracing

Combining this all into one estimator as path tracing; and using uniform sampling:

$$\tilde{L}(x,w) = \frac{1}{N}\sum_{k=1}^{N} \alpha(x,y)L_e(y,w) + \sigma_x||x-y|| * 4\pi * \alpha(x_k,x)p(w,w_k)\underbrace{\tilde{L}(x_k,w_k)}_{\text{recursive}}$$

(code on slide)