Monte Carlo Path Tracer: The Nasty Details

COS 526, Fall 2008
Adam Finkelstein
Slides from Funkhouser, Rusinkiewicz

Conceptual Goal

- Estimate radiance
  - From different parts of the scene
  - Towards the camera
  - Recall: this is proportional to 'film plane' irradiance
- To do this, simulate paths of light
  - From light sources
  - To camera
- Actually trace paths from camera to lights

Outline

- Simple path tracer
- Importance sampling
- Sampling techniques
- Russian roulette

Monte Carlo Path Tracer I

- For each pixel, repeat n times:
  - Choose a ray with p=camera, d=(0, 1) within pixel
  - Pixel color += (1/n) * TracePath(p, d)
- TracePath(p, d) returns (r, g, b):
  - Trace ray (p, d) to find nearest intersection p'
  - Select with probability 50%:
    - Emitted:
      - return 2 * (L_{out} + diffuse \cdot L_{amb}
    - Reflected:
      - generate ray in random direction d''
      - return 2 * \gamma (d'' \cdot -d) * (r, g, b) * TracePath(p', d')

Monte Carlo Path Tracer II

- For each pixel, repeat n times:
  - Choose a ray with p=camera, d=(0, 1) within pixel
  - Pixel color += (1/n) * TracePath(p, d)
- TracePath(p, d) returns (r, g, b):
  - Trace ray (p, d) to find nearest intersection p'
  - Select with probability 50%:
    - Emitted:
      - return 2 * (L_{out} + diffuse \cdot L_{amb}
    - Reflected:
      - generate ray in random direction d''
      - return 2 * \gamma (d'' \cdot -d) * (r, g, b) * TracePath(p', d')

Monte Carlo Path Tracer III

- For each pixel, repeat n times:
  - Choose a ray with p=camera, d=(0, 1) within pixel
  - Pixel color += (1/n) * TracePath(p, d)
- TracePath(p, d) returns (r, g, b):
  - Trace ray (p, d) to find nearest intersection p'
  - Select with probability 50%:
    - Emitted:
      - return 2 * (L_{out} + diffuse \cdot L_{amb}
    - Reflected:
      - generate ray in random direction d''
      - return 2 * \gamma (d'' \cdot -d) * (r, g, b) * TracePath(p', d')
Monte Carlo Path Tracer I

- For each pixel, repeat n times:
  - Choose a ray with \( p = \text{camera} \), \( d = (0, 1, 0) \) within pixel
  - Pixel color += \( (1/n) \times \text{TracePath}(p, d) \)
- \text{TracePath}(p, d) \text{ returns } (r, g, b):
  - Trace ray \((p, d)\) to find nearest intersection \( p' \)
  - Select with probability 50%:
    - Emitted:
      - return \( 2 \times \text{min}(L_{\text{emitted}}, L_{\text{pixel}}) \)\( \times \text{Weight} = \text{probability} \)
    - Reflected:
      - generate ray in random direction \( d' \)
      - return \( 2 \times \text{min}(L_{\text{reflected}}, L_{\text{pixel}}) \)\( \times \text{Weight} = \text{probability} \)

Drawbacks

- This algorithm is unbiased, but horribly inefficient:
  - Sample “emitted” 50% of the time, even if emitted=0
  - Reflect rays in random directions, even if mirror
  - If light source is small, rarely hit it
- Goal: improve efficiency without introducing bias

Monte Carlo Path Tracer II

- For each pixel, repeat n times:
  - Choose a ray with \( p = \text{camera} \), \( d = (0, 1, 0) \) within pixel
  - Pixel color += \( (1/n) \times \text{TracePath}(p, d) \)
- \text{TracePath}(p, d) \text{ returns } (r, g, b):
  - Trace ray \((p, d)\) to find nearest intersection \( p' \)
  - Select with probability 50%:
    - Emitted:
      - return \( 2 \times \text{min}(L_{\text{emitted}}, L_{\text{pixel}}) \)\( \times \text{Weight} = \text{probability} \)
    - Reflected:
      - generate ray in random direction \( d' \)
      - return \( 2 \times \text{min}(L_{\text{reflected}}, L_{\text{pixel}}) \)\( \times \text{Weight} = \text{probability} \)

Outline

- Simple path tracer
  - Importance sampling
- Sampling techniques
- Russian roulette

Improving Path Tracer

- Method: importance sampling
- Probability of picking path depends on energy
  - Don’t pick low-energy paths
  - Go out of your way to select high-energy paths
- Can apply at “micro” level (e.g., selecting reflected ray directions)
- Can apply at “macro” level (e.g., selecting reflected/emitted or casting rays to lights)

Importance Sampling

- Can pick paths however we want, but contribution weighted by 1/probability

\[
\int f(x)dx = \frac{1}{N} \sum_{i=1}^{N} y_i
\]

\[
y_i = \frac{f(x_i)}{p(x_i)}
\]
Monte Carlo Path Tracer I

- TracePath(p, d) returns (r,g,b):
  - Trace ray (p, d) to find nearest intersection p'
  - Select with probability 50%:
    - Emitted:
      return 2 * (L_{emitted} * L_{paths} + L_{emission})
    - Reflected:
      generate ray in random direction d'
      return 2 * |(d - d') * (n - d')| * TracePath(p, d)

Monte Carlo Path Tracer II

- TracePath(p, d) returns (r,g,b):
  - Trace ray (p, d) to find nearest intersection p'
    - If |L - (0,0,0)| < 0.0001 then p_{emp} = 0
    - else if f = (0,0,0) then p_{emp} = 1
    - else p_{emp} = 0.9
    - If random() < p_{emp} then
      - Emitted:
        return 4 * (L_{p_{emp}} + L_{paths} + L_{emission})
      - Reflected:
        generate ray in random direction d'
        return 4 * (L_{p_{emp}} * |(d - d') * (n - d')|) * TracePath(p, d)

Another Variation

- Reflected case:
  - Pick a light source
  - Trace a ray towards that light
  - Trace a ray anywhere except for that light
    - Rejection sampling
    - Divide by probabilities

Monte Carlo Path Tracer III

- TracePath(p, d) returns (r,g,b):
  - Trace a ray from point p in direction d
    - Emitted:
      return 1
    - Reflected:
      generate ray in random direction d' towards a light
      L = (1/2) * (L_{p_{emp}} * |(d - d') * (n - d')|) * TracePath(p', d')
      generate ray in random direction d' not towards the light
      L = (1/2) * (L_{p_{emp}} * |(d - d') * (n - d')|) * TracePath(p', d')
      return (1/2) * L

Monte Carlo Path Tracer III

- What are probabilities?
  - p_{emitted} = 1/(solid angle of light) for ray to light source
  - (1 - the above) for non-light ray
  - Extra factor of 2 because shooting 2 rays

Outline

- Simple path tracer
- Importance sampling
  - Sampling techniques
    - Russian roulette
2-D Sampling Techniques

- At several points in this algorithm, need to sample a 2D domain
  - Within a pixel, when generating paths (easy)
  - Within a triangle, when sampling a light source
  - Within the hemisphere of reflected directions
    - Uniform
    - Weighted by cosine
    - Weighted by BRDF

Sampling a Triangular Domain

- To generate a point within a triangle with vertices $V_0$, $V_1$, $V_2$:
  - Generate random $s$ and $t$ on $[0, 1]$
  - If $s + t > 1$, let $s = 1 - s$ and $t = 1 - t$
  - Construct point $V_3 = s(V_1 - V_0) + t(V_2 - V_0)$

Reflected Ray Sampling

- Uniform directional sampling: how to generate random ray on hemisphere?
  - Option #1: rejection sampling
    - Generate random numbers $(x, y, z)$, with $x, y, z$ in $[-1, 1]$
    - If $x^2 + y^2 + z^2 > 1$, reject
    - Normalize $(x, y, z)$
    - If pointing into surface (ray dot $n < 0$), flip

Uniform Directional Sampling

- Option #2: inversion method
  - In polar coords, density must be proportional to $\sin \theta$
  - Integrate, invert to $\cos \theta$
- So, recipe is:
  - Generate $\phi$ in $0..2\pi$
  - Generate $z$ in $0..1$
  - Let $\theta = \cos^{-1} z$
  - $(x, y, z) = (\sin \theta \cos \phi, \sin \theta \sin \phi, \cos \theta)$

BRDF Importance Sampling

- Better than uniform sampling:
  - Importance sampling
  - Because you divide by probability, ideally:
    - probability $\propto f_r \cos \alpha$
- [Lafortune, 1994]:

$$f_r(x, \hat{\omega}_t, \hat{\omega}_o) = k_d \frac{1}{\pi} + k_a \frac{n + 2}{2\pi} \cos^n \alpha$$

BRDF Importance Sampling

- For cosine-weighted Lambertian:
  - Density $= \cos \theta$
  - Integrate, invert to $\cos \phi$ (sqrt)
  - So, recipe is:
    - Generate $\phi$ in $0..2\pi$
    - Generate $z$ in $0..1$
    - Let $\theta = \cos^{-1} (\text{sqrt}(z))$
**BRDF Importance Sampling**

- Phong BRDF: \( f_s \propto \cos^s \alpha \) where \( \alpha \) is angle between outgoing ray and ideal mirror direction
- Constant scale = \( k_s (n+2)/(2\pi) \)
- Ideally we would sample this times \( \cos \theta \)
  - Difficult
  - Easier to sample BRDF itself, then multiply by \( \cos \theta \)
  - That's OK — still better than random sampling

**BRDF Importance Sampling**

- Recipe for sampling specular term:
  - Generate \( x \) in 0.1
  - Let \( \alpha = \cos^{-1} (2x^{1/2}) \)
  - Generate \( \phi \) in 0.2\pi
- This gives direction w.r.t. ideal mirror direction

**BRDF Importance Sampling**

- Recipe for combining terms:
  - \( r = \text{random}() \)
  - \( \text{If } r < k_d \text{ then} \)
    - \( d = \text{sample diffuse direction} \)
    - weight \( = 1/N_d \)
  - \( \text{else if } (r < k_d + k_s) \text{ then} \)
    - \( d = \text{sample specular direction} \)
    - weight \( = 1/N_s \)
  - \( \text{else} \)
    - terminate ray

**Outeine**

- Simple path tracer
- Importance sampling
- Sampling techniques
- Russian roulette

**Russian Roulette**

- Maintain current weight along path (need another parameter to TracePath)
- Terminate ray probabilistically if weight is less than some threshold
- Scale radiance along path by probability

```
if (weight < Threshold) then
  if (random() < weight) then terminate path
  else weight = weight / (1 - P)
```

**Programming Assignment**