Monte Carlo Path Tracer: The Nasty Details

COS 526, Fall 2006

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, $d=(\theta,\phi)$ 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 * (Le_{red}, Le_{green}, Le_{blue})
    - Reflected:
      generate ray in random direction d'
      return 2 * $f_r(d \to d')$ * $(n \cdot d')$ * TracePath(p', d')

Monte Carlo Path Tracer I

- For each pixel, repeat n times:
  - Choose a ray with p=camera, $d=(\theta,\phi)$ 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 * (Le_{red}, Le_{green}, Le_{blue})
    - Reflected:
      generate ray in random direction d'
      return 2 * $f_r(d \to d')$ * $(n \cdot d')$ * TracePath(p', d')

Average over paths

Monte Carlo Path Tracer I

- For each pixel, repeat n times:
  - Choose a ray with p=camera, $d=(\theta,\phi)$ 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 * (Le_{red}, Le_{green}, Le_{blue})
    - Reflected:
      generate ray in random direction d'
      return 2 * $f_r(d \to d')$ * $(n \cdot d')$ * TracePath(p', d')

RGB color

Monte Carlo Path Tracer I

- For each pixel, repeat n times:
  - Choose a ray with p=camera, $d=(\theta,\phi)$ 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 * (Le_{red}, Le_{green}, Le_{blue})
    - Reflected:
      generate ray in random direction d'
      return 2 * $f_r(d \rightarrow d')$ * $(n \cdot d')$ * TracePath(p', d')

Weight = 1/probability

Monte Carlo Path Tracer I

- For each pixel, repeat n times:
  - Choose a ray with p=camera, $d=(\theta,\phi)$ 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 * (Le_{red}, Le_{green}, Le_{blue})
    - Reflected:
      generate ray in random direction d'
      return 2 * $f_r(d \rightarrow d')$ * $(n \cdot d')$ * TracePath(p', d')

Path terminated when emission is evaluated

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

Outline

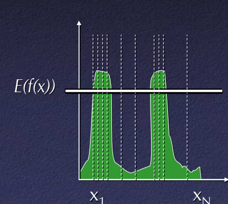
- Simple path tracer
- Importance sampling
- Sampling techniques
- Russian roulette

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



$$\int_{\Omega} f(x)dx = \frac{1}{N}\sum_{i=1}^{N} Y_i$$

$$Y_i = \frac{f(x_i)}{p(x_i)}$$

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 * (Le_{red}, Le_{green}, Le_{blue})
    - Reflected:
      - generate ray in random direction *d′*
      - return 2 * $f_r(d \rightarrow d')$ * $(n \cdot d')$ * 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 Le = (0,0,0) then $p_{emit} = 0$
    else if f_r = (0,0,0) then $p_{emit} = 1$
    else $p_{emit} = .9$
  - If random() < p_{emit} then
    - Emitted:
      - return (1/ p_{emit}) * (Le_{red}, Le_{green}, Le_{blue})
    - Reflected:
      - generate ray in random direction *d′*
      - return (1/ (1–p_{emit})) * $f_r(d \rightarrow d')$ * $(n \cdot d')$ * 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 ray (*p*, *d*) to find nearest intersection *p′*
  - If Le = (0,0,0) then $p_{emit} = 0$
    else if f_r = (0,0,0) then $p_{emit} = 1$
    else $p_{emit} = .9$
  - If random() < p_{emit} then
    - Emitted:
      - return (1/ p_{emit}) * (Le_{red}, Le_{green}, Le_{blue})
    - Reflected:
      - generate ray in random direction *d′* towards a light
      - $L_r = (1/2p_{light})$ * $f_r(d \rightarrow d')$ * $(n \cdot d')$ * TracePath(*p′*, *d′*)
      - generate ray in random direction *d′* not towards the light
      - L_r += $(1/2*(1-p_{light}))$ * $f_r(d \rightarrow d')$ * $(n \cdot d')$ * TracePath(*p′*, *d′*)
      - return (1/ (1–p_{emit})) * L_r

Monte Carlo Path Tracer III

- What are probabilities?
  - p_{light} = 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
- Ø 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 $s \leftarrow 1 - s$ and $t \leftarrow 1 - t$
  - Construct point $v_0 + s(v_1 - v_0) + t(v_2 - v_0)$

Reflected Ray Sampling

- Uniform directional sampling: how to generate random ray on a 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 θ (remember d(solid angle) = sin θ $d\theta\, d\phi$)
  - Integrate, invert → $\cos^{-1}$
- So, recipe is
  - Generate ϕ in 0..2π
  - 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 $\propto f_r * \cos\theta_i$
- [Lafortune, 1994]:

$$f_r(x, \vec{\omega}_i, \vec{\omega}_o) = k_d \frac{1}{\pi} + k_s \frac{n+2}{2\pi} \cos^n \alpha$$

BRDF Importance Sampling

- For cosine-weighted Lambertian:
  - Density = cos θ sin θ
  - Integrate, invert → $\cos^{-1}$(sqrt)
- So, recipe is:
  - Generate ϕ in 0..2π
  - Generate z in 0..1
  - Let $\theta = \cos^{-1}(\text{sqrt}(z))$

BRDF Importance Sampling

- Phong BRDF: $f_r \propto \cos^n \alpha$ where α is angle between outgoing ray and ideal mirror direction
- Constant scale = $k_s(n+2)/(2\pi)$
- Can't sample this times $\cos \theta_i$
  - Can only sample BRDF itself, then multiply by $\cos \theta_i$
  - That's OK – still better than random sampling

BRDF Importance Sampling

- Recipe for sampling specular term:
  - Generate *z* in 0..1
  - Let $\alpha = \cos^{-1} (z^{1/(n+1)})$
  - Generate ϕ_α in $0..2\pi$
- This gives direction w.r.t. ideal mirror direction

BRDF Importance Sampling

- Recipe for combining terms:
  - r = random()
  - If ($r < k_d$) then
    - d′ = sample diffuse direction
    - weight = $1/k_d$
  - else if ($r < k_d + k_s$) then
    - d′ = sample diffuse direction
    - weight = $1/k_s$
  - else
    - terminate ray

Outline

- Simple path tracer
- Importance sampling
- Sampling techniques
- Ø 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 < Thresh) then
    If (random() < P) then terminate path
    else weight = weight / (1 – P)
```