Advanced Computer Graphics (Fall 2009)
CS 294, Rendering Lecture 5: Monte Carlo Path Tracing
Ravi Ramamoorthi

http://inst.eecs.berkeley.edu/~cs294-13/fa09

Acknowledgements and some slides: Szymon Rusinkiewicz and Pat Hanrahan

To Do
- Start working on assignment
- Find partners for this purpose

Motivation
- General solution to rendering and global illumination
- Suitable for a variety of general scenes
- Based on Monte Carlo methods
- Enumerate all paths of light transport

Monte Carlo Path Tracing

Advantages
- Any type of geometry (procedural, curved, ...)
- Any type of BRDF (specular, glossy, diffuse, ...)
- Samples all types of paths (LSDP)
- Accuracy controlled at pixel level
- Low memory consumption
- Unbiased - error appears as noise in final image

Disadvantages (standard Monte Carlo problems)
- Slow convergence (square root of number of samples)
- Noise in final image

Monte Carlo Path Tracing

Big diffuse light source, 20 minutes

1000 paths/pixel
Monte Carlo Path Tracing

Integrate radiance for each pixel by sampling paths randomly.

\[ L(x,\hat{n}) = L(x,\hat{n}) + \int_{\Omega} f(x,\hat{x}',\hat{n}) L(x,\hat{x}') (\hat{n} \cdot \hat{x}') d\hat{x}' \]

Sampling Techniques

Problem: how do we generate random points/directions during path tracing and reduce variance?

- Importance sampling (e.g. by BRDF)
- Stratified sampling

Simplest Monte Carlo Path Tracer

For each pixel, cast \( n \) samples and average over paths:

- Choose a ray with \( p \)-camera, \( d = (0,0,\theta) \) within pixel
- Pixel color \( \rightarrow (1/n) \ast \text{TracePath}(p, d) \)

TracePath\((p, d)\) returns \((r, g, b)\) (and calls itself recursively):

- Trace ray \((p, d)\) to find nearest intersection \( p'\)
- Select with probability (say) 50%:
  - Emitted:
    - return \( 2 \ast (L_{\text{red}}, L_{\text{green}}, L_{\text{blue}}) \) // 2 = 1/(50\%)
  - Reflected:
    - generate ray in random direction \( d'\)
    - return \( 2 \ast \int f(d \rightarrow d') \ast (n \cdot d') \ast \text{TracePath}(p', d') \)

Simple Monte Carlo Path Tracer

- Step 1: Choose a ray \((u,v,\theta,\phi)\) [per pixel]; assign weight \( = 1 \)
- Step 2: Trace ray to find intersection with nearest surface
- Step 3: Randomly choose between emitted and reflected light
  - Step 3a: If emitted, return weight \( \ast L_e \)
  - Step 3b: If reflected, weight \( \ast \) reflectance
    - Generate ray in random direction
    - Go to step 2

Outline

- Motivation and Basic Idea
- Implementation of simple path tracer
- Variance Reduction: Importance sampling
- Other variance reduction methods
- Specific 2D sampling techniques

Simple Monte Carlo Path Tracer

For each pixel, cast \( n \) samples and average over paths:

- Choose a ray with \( p \)-camera, \( d = (0,0,\theta) \) within pixel
- Pixel color \( \rightarrow (1/n) \ast \text{TracePath}(p, d) \)

TracePath\((p, d)\) returns \((r, g, b)\) (and calls itself recursively):

- Trace ray \((p, d)\) to find nearest intersection \( p'\)
- Select with probability (say) 50%:
  - Emitted:
    - return \( 2 \ast (L_{\text{red}}, L_{\text{green}}, L_{\text{blue}}) \) // 2 = 1/(50\%)
  - Reflected:
    - generate ray in random direction \( d'\)
    - return \( 2 \ast \int f(d \rightarrow d') \ast (n \cdot d') \ast \text{TracePath}(p', d') \)
For each pixel, cast n samples and average

- Choose a ray with \( p \)-camera, \( d = (0, \phi) \) within pixel
- Pixel color \( \rightarrow (1/n) \times \text{TracePath}(p, d) \)

\text{TracePath}(p, d) \text{ returns (r, g, b) \{and calls itself recursively\}}:

- Trace ray \((p, d)\) to find nearest intersection \( p' \)
- Select with probability (say) 50%:
  - Emitted: return \( 2 \times (L_{\text{red}}, L_{\text{green}}, L_{\text{blue}}) \) \{Weight = \text{Probability} \}
  - Reflected: generate ray in random direction \( d' \)
    return \( 2 \times f(d \rightarrow d') \times (n \cdot d') \times \text{TracePath}(p', d') \)

Path Tracing

Arnold Renderer (M. Fajardo)

- Works well diffuse surfaces, hemispherical light

Advantages and Drawbacks

- Advantage: general scenes, reflectance, so on
  - By contrast, standard recursive ray tracing only mirrors
- This algorithm is \textit{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
  - Variance reduction using many of the methods discussed for Monte Carlo integration last week
  - Subject of much interest in graphics in 90s till today

Outline

- Motivation and Basic Idea
- Implementation of simple path tracer
- \textit{Variance Reduction: Importance sampling}
- Other variance reduction methods
- Specific 2D sampling techniques
Importance Sampling

- Pick paths based on energy or expected contribution
  - More samples for high-energy paths
  - Don’t pick low-energy paths
- At “macro” level, use to select between reflected vs emitted, or in casting more rays toward light sources
- At “micro” level, importance sample the BRDF to pick ray directions
- Tons of papers in 90s on tricks to reduce variance in Monte Carlo rendering

Importance Sampling

Can pick paths however we want, but contribution weighted by 1/probability
- Already seen this division of 1/prob in weights to emission, reflectance

\[
E(f(x)) = \frac{1}{N} \sum_{i=1}^{N} Y_i
\]

\[
Y_i = \frac{f(x_i)}{p(x_i)}
\]

Simplest Monte Carlo Path Tracer

For each pixel, cast n samples and average
- Choose a ray with \( p=\text{camera}, d=(\theta, \phi) \) within pixel
- Pixel color \( \rightarrow (1/n) \cdot \text{TracePath}(p, d) \)

TracePath\((p, d)\) returns \((r, g, b)\) [and calls itself recursively]:
- Trace ray \((p, d)\) to find nearest intersection \( p' \)
- Select with probability (say) 50%:
  - Emitted
    - return \( 2 \cdot (L_{\text{ered}}, L_{\text{egreen}}, L_{\text{eblue}}) \) // \( 2 = 1/(50\%) \)
  - Reflected
    - generate ray in random direction \( d' \)
    - return \( (1/(1-p_{\text{emit}})) \cdot f(d \rightarrow d') \cdot (n \cdot d) \cdot \text{TracePath}(p', d') \)

Importance sample Emit vs Reflect

TracePath\((p, d)\) returns \((r, g, b)\) [and calls itself recursively]:
- Trace ray \((p, d)\) to find nearest intersection \( p' \)
- If \( Le = (0,0,0) \) then \( p_{\text{emit}}=0 \) else \( p_{\text{emit}}=0.9 \) (say)
- If random() < \( p_{\text{emit}} \) then:
  - Emitted:
    - return \( (1/p_{\text{emit}}) \cdot (L_{\text{ered}}, L_{\text{egreen}}, L_{\text{eblue}}) \)
  - Else Reflected:
    - generate ray in random direction \( d' \)
    - return \( (1/(1-p_{\text{emit}})) \cdot f(d \rightarrow d') \cdot (n \cdot d) \cdot \text{TracePath}(p', d') \)

Importance sample Emit vs Reflect

TracePath\((p, d)\) returns \((r, g, b)\) [and calls itself recursively]:
- Trace ray \((p, d)\) to find nearest intersection \( p' \)
- If \( Le = (0,0,0) \) then \( p_{\text{emit}}=0 \) else \( p_{\text{emit}}=0.9 \) (say)
- If random() < \( p_{\text{emit}} \) then:
  - Emitted:
    - return \( (1/p_{\text{emit}}) \cdot (L_{\text{ered}}, L_{\text{egreen}}, L_{\text{eblue}}) \)
  - Else Reflected:
    - generate ray in random direction \( d' \)
    - return \( (1/(1-p_{\text{emit}})) \cdot f(d \rightarrow d') \cdot (n \cdot d) \cdot \text{TracePath}(p', d') \)

Outline

- Motivation and Basic Idea
- Implementation of simple path tracer
- Variance Reduction: Importance sampling
- Other variance reduction methods
- Specific 2D sampling techniques
More variance reduction

- Discussed “macro” importance sampling
  - Emitted vs reflected
- How about “micro” importance sampling
  - Shoot rays towards light sources in scene
  - Distribute rays according to BRDF

One Variation for Reflected Ray

- Pick a light source
- Trace a ray towards that light
- Trace a ray anywhere except for that light
  - Rejection sampling
- Divide by probabilities
  - \( \frac{1}{\text{solid angle of light}} \) for ray to light source
  - \((1 - \text{the above})\) for non-light ray
  - Extra factor of 2 because shooting 2 rays

Russian Roulette

- Maintain current weight along path
  (need another parameter to TracePath)
- Terminate ray iff \(|\text{weight}| < \text{const.}\)
- Be sure to weight by 1/probability

Monte Carlo Extensions

Unbiased
- Bidirectional path tracing
- Metropolis light transport

Biased, but consistent
- Noise filtering
- Adaptive sampling
- Irradiance caching
Monte Carlo Extensions

Unbiased
- Bidirectional path tracing
- Metropolis light transport

Biased, but consistent
- Noise filtering
- Adaptive sampling
- Irradiance caching

Monte Carlo Extensions

Unbiased
- Bidirectional path tracing
- Metropolis light transport

Biased, but consistent
- Noise filtering
- Adaptive sampling
- Irradiance caching

Monte Carlo Extensions

Unbiased
- Bidirectional path tracing
- Metropolis light transport

Biased, but consistent
- Noise filtering
- Adaptive sampling
- Irradiance caching

Monte Carlo Extensions

Unbiased
- Bidirectional path tracing
- Metropolis light transport

Biased, but consistent
- Noise filtering
- Adaptive sampling
- Irradiance caching

Monte Carlo Path Tracing Image

2000 samples per pixel, 30 computers, 30 hours

Jensen

Filtered

Jensen

Unfiltered

Jensen

Fixed

Ohbuchi

Adaptive

Ohbuchi

Monte Carlo Path Tracing Image

2000 samples per pixel, 30 computers, 30 hours

Jensen
Outline

- Motivation and Basic Idea
- Implementation of simple path tracer
- Variance Reduction: Importance sampling
- Other variance reduction methods
- Specific 2D sampling techniques

2D Sampling: Motivation

- Final step in sending reflected ray: sample 2D domain
- According to projected solid angle
- Or BRDF
- Or area on light source
- Or sampling of a triangle on geometry
- Etc.

Sampling Projected Solid Angle

Generate cosine weighted distribution

Sampling Upper Hemisphere

- 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

Sampling Upper Hemisphere

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

BRDF Importance Sampling

- Better than uniform sampling: importance sampling
- Because you divide by probability, ideally probability $\propto f_r \cdot \cos \theta_f$
BRDF Importance Sampling

- For cosine-weighted Lambertian:
  - Density = \( \cos \theta \sin \theta \)
  - Integrate, invert \( \rightarrow \cos^{-1}(\sqrt{z}) \)
  - So, recipe is:
    - Generate \( \phi \) in \( 0..2\pi \)
    - Generate \( z \) in \( 0..1 \)
    - Let \( \theta = \cos^{-1}(\sqrt{z}) \)

Phong BRDF: \( f_{\alpha} \propto \cos^{n} \alpha \) where \( \alpha \) 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

Recipe for sampling specular term:

- Generate \( z \) in \( 0..1 \)
- Let \( \alpha = \cos^{-1}(z^{1/(n+1)}) \)
- Generate \( \phi \) in \( 0..2\pi \)
- This gives direction w.r.t. ideal mirror direction
- Convert to \((x,y,z)\), then rotate such that \( z \) points along mirror dir.

Summary

- Monte Carlo methods robust and simple (at least until nitty gritty details) for global illumination
- Must handle many variance reduction methods in practice
- Importance sampling, Bidirectional path tracing, Russian roulette etc.
- Rich field with many papers, systems researched over last 10 years