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
- Big diffuse light source, 20 minutes
- 1000 paths/pixel

Advantages
- Any type of geometry (procedural, curved, ...)
- Any type of BRDF (specular, glossy, diffuse, ...)
- Samples all types of paths \( (u, S, D, F) \)
- 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

Integrate radiance for each pixel by sampling paths randomly

\[
L_e(x, w) = L_e(x, w) + \int_0^1 f_l(x, w', w)L_i(x, w')(w' \cdot n)dw
\]
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 = \(L_e\)
  - Step 3b: If reflected, weight = \(\text{reflectance} \times \text{weight} \times \text{reflectance}\)
    Generate ray in random direction
    Go to step 2

Sampling Techniques

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

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

Outline

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

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 += \((1/n) \times \text{TracePath}(p, d)\)

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 \(2 \times (L_{\text{red}}, L_{\text{green}}, L_{\text{blue}}) / 2 = 1/(50\%)\)
  - Reflected:
    - generate ray in random direction \(d'\)
    - return \(2 \times f(d \rightarrow d') \times (n \cdot d') \times \text{TracePath}(p', d')\)

Weight = 1/probability
Remember: unbiased requires having \(f(x) / p(x)\)
Simplest Monte Carlo Path Tracer

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

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 $2 \cdot (L_\text{red}, L_\text{green}, L_\text{blue}) / 2 = 1/(50\%)$
  - Reflected:
    - generate ray in random direction $d'$$
    - return $2 \cdot f(d \Rightarrow d') \cdot (n \cdot d') \cdot \text{TracePath}(p', d')$

Arnold Renderer (M. Fajardo)

- Works well diffuse surfaces, hemispherical light

From CS 283(294) a few years ago

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 now standard in production. I consulted on Pixar’s system for upcoming movies

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 $= \langle f(n) \rangle \cdot \text{TracePath}(p, d)$

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 * ($L_{\text{red}}, L_{\text{green}}, L_{\text{blue}}$) // $2 = 1/50\%$
  - Reflected: generate ray in random direction $d'$
    return $2 \cdot f(d \rightarrow d') \cdot (n \cdot d') \cdot \text{TracePath}(p', d')$

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 $L_e = (0,0,0)$ then $p_{\text{emit}} = 0$ else $p_{\text{emit}} = 0.9$ (say)
- If random() < $p_{\text{emit}}$, then:
  - Emitted: return ($U_{\text{random}} \cdot L_{\text{green}}, L_{\text{green}}, L_{\text{blue}}$)
  - Else Reflected: generate ray in random direction $d'$
    return $(U_{\text{random}}) \cdot f(d \rightarrow d') \cdot (n \cdot d') \cdot \text{TracePath}(p', d')$

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
  - 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

### Russian Roulette
- Maintain current weight along path (need another parameter to TracePath)
- Terminate ray iff |weight| < 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

---

**Path Tracing: Include Direct Lighting**

Step 1. Choose a camera ray $r$ given the $(x,y,u,v,t)$ sample

- weight = 1
- $L = 0$

Step 2. Find ray-surface intersection

Step 3.

- $L += \text{weight} \times L(r)\text{(light sources)}$
- weight *= \text{reflectance}(r)$
- Choose new ray $r’ \sim \text{BRDF pdf}(r)$
- Go to Step 2.
Monte Carlo Extensions

Unbiased
- Bidirectional path tracing
- Metropolis light transport

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

Heinrich

Monte Carlo Extensions

Unbiased
- Bidirectional path tracing
- Metropolis light transport

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

Ohbuchi

Monte Carlo Extensions

Unbiased
- Bidirectional path tracing
- Metropolis light transport

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

Jensen

Monte Carlo Extensions

Unbiased
- Bidirectional path tracing
- Metropolis light transport

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

Jensen

Monte Carlo Path Tracing Image

2000 samples per pixel, 30 computers, 30 hours
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 \( z \) 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 proportional to \( f_r \cdot \cos \theta \)
### BRDF Importance Sampling

- For cosine-weighted Lambertian:
  - Density = \( \cos \theta \sin \theta \)
  - Integrate, invert \( \theta = \cos^{-1}(\sqrt{z}) \)

- So, recipe is:
  - Generate \( \phi \) in \( 0..2\pi \)
  - Generate \( z \) in \( 0..1 \)
  - Let \( \theta = \cos^{-1}(\sqrt{z}) \)

### BRDF Importance Sampling

- Phong BRDF: \( f_r \sim \cos^n \alpha \) 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

### BRDF Importance Sampling

- Recipe for sampling specular term:
  - Generate \( z \) in \( 0..1 \)
  - Let \( \alpha = \cos^{-1}(\sqrt{z}) \)
  - 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