Motivation: Monte Carlo Path Tracing

- Key application area for sampling/reconstruction
- Core method to solve rendering equation
- Widely used production+realtime (with denoising)
- General solution to rendering, global illumination
- Suitable for a variety of general scenes
- Based on Monte Carlo methods
- Enumerate all paths of light transport
- We mostly treat this as a black box, but background is still important

Monte Carlo Path Tracing

Advantages
- Any type of geometry (procedural, curved, ...)
- Any type of BRDF (specular, glossy, diffuse, ...)
- Samples all types of paths (LSD)*E
- 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

\[ L_e(x, \omega) = L_e(x, \omega) + \int_{\Omega} f(x, \omega', \omega) L_e(x, \omega') (\omega' \cdot n) d\omega' \]
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 \(= Le\)
  - Step 3b: If reflected, weight \(= reflectance\)
    - 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

Simple Monte Carlo Path Tracer

For each pixel, cast \(n\) samples and average over paths
- Choose a ray with \(p=\text{camera}, d=(\theta,\phi)\) 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 \(2 \cdot (L_{\text{ered}}, L_{\text{egreen}}, L_{\text{eblue}}) / 2 = 1/(50\%)\)
  - Reflected: generate ray in random direction \(d'\)
    - return \(2 \cdot f(d \rightarrow d') \cdot (n \cdot d') \cdot \text{TracePath}(p', d')\)

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

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 \(2 \cdot f(d \rightarrow d') \cdot (n \cdot d') \cdot \text{TracePath}(p', d')\)
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) * 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 $2^\times \left(\frac{\text{Le}_\text{red} + \text{Le}_\text{green} + \text{Le}_\text{blue}}{2} \right)$ Path terminated when Emitted evaluated
  - Reflected: generate ray in random direction d’
    return $2^\times \left(\frac{||d-d’||}{\text{fre}}\right)^\times \text{TracePath}(p’, d’)

Arnold Renderer (M. Fajardo)

- Works well diffuse surfaces, hemispherical light

From UCB class many years ago

Advantages and Drawbacks

- Advantage: general scenes, reflectance, so on
  - By contrast, standard recursive ray tracing only mirrors
- 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
  - 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
- 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 initial Pixar system for MU (2011).

**Simplest Monte Carlo Path Tracer**

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

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 $2 \frac{(L_{\text{red}}, L_{\text{green}}, L_{\text{blue}})}{2} = \frac{1}{0.5}$
  - Reflected: generate ray in random direction $d'$
    return $2 \frac{f(d \cdot d') \cdot |n \cdot d|}{|n \cdot d|} \cdot \text{TracePath}(p', d')$

**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 $\frac{1}{p_{\text{emit}}} \cdot (L_{\text{red}}, L_{\text{green}}, L_{\text{blue}})$
  - Else Reflected:
    generate ray in random direction $d'$
    return $\frac{1}{1-p_{\text{emit}}} \cdot f(d \cdot d') \cdot |n \cdot d| \cdot \text{TracePath}(p', d')$

**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

\[ \text{weight} = 1; \]
\[ L = 0 \]

Step 2. Find ray-surface intersection

Step 3.

\[ L \leftarrow \text{weight} \times \text{Lr(light sources)} \]
\[ \text{weight} \leftarrow \text{reflectance}(r) \]
Choose new ray \( r' \) ~ 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

Unfiltered

Filtered

Monte Carlo Extensions

Unbiased
- Bidirectional path tracing
- Metropolis light transport

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

Adaptive

Fixed

Monte Carlo Extensions

Unbiased
- Bidirectional path tracing
- Metropolis light transport

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

Jensen

Irradiance Caching Example

Final Image

Sample Locations

Stratified Sampling

Stratified sampling like jittered sampling
Allocate samples per region
\[ N = \sum N_i \]
\[ F_i = \frac{1}{N} \sum N_i F_i \]

New variance
\[ \text{var}(F_i) = \frac{1}{N} \sum N_i \text{var}(F_i) \]

Thus, if the variance in regions is less than the overall variance, there will be a reduction in resulting variance

For example: An edge through a pixel
\[ \text{var}(F_i) = \frac{1}{N} \sum F_i - \frac{(\sum F_i)^2}{N^2} \]

Comparison of simple patterns

Latin Hypercube
Quasi Monte Carlo
Ground Truth
Uniform
Random
Stratified

16 samples for area light, 4 samples per pixel, total 64 samples

If interested, see my recent paper “A Theory of Monte Carlo Visibility Sampling”

Figures courtesy Tianyu Liu
Bidirectional Path Tracing

Path pyramid (k = l + e = total number of bounces)

Comparison

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 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 θ
    (remember d(solid angle) = sin θ dθ dϕ)
  - Integrate, invert → cos⁻¹

  So, recipe is
  - Generate ϕ in 0..2π
  - Generate z in 0..1
  - Let θ = cos⁻¹ z
  - (x,y,z) = (sin θ cos ϕ, sin θ sin ϕ, cos θ)
BRDF Importance Sampling

- Better than uniform sampling: importance sampling
- Because you divide by probability, ideally probability proportional to \( f_r \cdot \cos \theta \)

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

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 \( \cos \theta \)
  - Can only sample BRDF itself, then multiply by \( \cos \theta \)
  - That’s OK – still better than random sampling

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

Mies House: Swimming Pool
### Optional Path Tracing Assignment

- If you have not taken CSE 168 or done path tracer
- Follow CSE 168 on UCSD online, build path tracer
- Includes guide for raytracing if not already done
- For your benefit only, optional do not turn in (since many people wanted it for knowledge)
- You can use it in final project, but don’t need to, and may be better off using off-the-shelf renderer
- If you do use it in final project, document it
- Again, it is optional and not directly graded

### 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 30 years
- For rest of the course, we largely take this as a black box, focusing on sampling and reconstruction