Monte-Carlo Ray Tracing

Image by Henrik
Class evaluation

- [http://sixweb.mit.edu/](http://sixweb.mit.edu/)

-
Global illumination

- So far, we've seen only direct lighting (red here)
- We also want indirect lighting
  - Full integral on hemisphere (multiplied by BRDF)
  - In practice, send tons of random rays
What else can we integrate?

- Pixel: antialiasing
- Light sources: Soft shadows
- Lens: Depth of field
- Time: Motion blur
- BRDF: glossy reflection
- Hemisphere: indirect lighting

\[ \iiint L(x, y, t, u, v) \, dx \, dy \, dt \, du \, dv \]
Domains of integration

- Pixel, lens (Euclidean 2D domain)
- Time (1D)
- Hemisphere
  - Work needed to ensure uniform probability
- Light source
  - Same thing: make sure that the probabilities and the measures are right.
Example: Light source

- Integrate over surface or over angle
- Be careful to get probabilities and integration measure right!
  - More in 6.839

Sampling the source uniformly

Sampling the hemisphere uniformly
A little bit of eye candy for motivation

- Glossy material rendering
- Random reflection rays around mirror direction
  - 1 sample per pixel
A little bit of eye candy for motivation

- Glossy material rendering
- Random reflection rays around mirror direction
  - 256 sample per pixel
Error/noise expressed as variance

- We use random rays
- Run the algorithm again, → get different image
- What is the noise/variance/standard deviation?
Goal today

• Monte Carlo integration
• Better sampling:
  – importance
  – stratification
• First simple integrals
  – e.g. glossy, soft shadows
• Then full global illumination
• Acceleration & other graphics tricks
Questions?

• Image by Henrik
Integration

- compute integral of weird arbitrary function
  - e.g. integral over area light source
- Continuous problem $\Rightarrow$ we need to discretize
Integration

- You know trapezoid
Integration

• Now come Monte Carlo: use random samples and compute average
  – We don’t keep track of spacing between samples
  – But we kind of hope it will be on average $1/n$
Monte-Carlo computation of $\pi$

- Take a square
- Take a random point $(x,y)$ in the square
- Test if it is inside the $\frac{1}{4}$ disc $(x^2+y^2 < 1)$
- The probability is $\frac{\pi}{4}$

Integral of the function that is one inside the circle, zero outside
Monte-Carlo computation of $\pi$

- The probability is $\pi/4$
- Count the inside ratio $n = \#$ inside / total $\#$ trials
- $\pi \approx n \times 4$

- The error depends on the number of trials

```java
long N_SAMPLES=(long)1e9;
long nSuccess=0;
long nTrial=0;
int pow=1;
for (long i=0; i<N_SAMPLES; i++) {
    double x=Math.random();
    double y=Math.random();
    if (x*x+y*y<1) nSuccess++;
    /*
    if (nTrial%pow==0) {
        double myPi=4.0*(double)nSuccess/(double)i;
        System.out.println("n=\\nTrial+" pi="myPi);" pow*=2;
    }
    */
    double myPi=4.0*(double)nSuccess/(double)N_SAMPLES;
    System.out.println(myPi);
}
```
Monte-Carlo computation of $\pi$

- The probability is $\pi/4$
- Count the inside ratio $n = \# \text{inside} / \text{total} \# \text{trials}$
- $\pi \approx n \times 4$
- The error depends on the number of trials

**Demo**

```python
def piMC(n):
    success = 0
    for i in range(n):
        x = random.random()
        y = random.random()
        if x*x + y*y < 1:
            success = success + 1
    return 4.0 * float(success) / float(n)
```
Why not use Simpson integration?

• Yeah, to compute $\pi$, Monte Carlo is not very efficient
• But convergence is independent of dimension
• Better to integrate high-dimensional functions
• For $d$ dimensions, Simpson requires $N^d$ domains
Questions?

• Image from the ARNOLD Renderer by Marcos Fajardo
Example

\[ I = \int_0^1 5x^4 \, dx \]

- We know it should be 1.0

- In practice with uniform samples:
Monte-Carlo Recap

• Expected value is the integrand
  – Accurate “on average”

• Variance decrease in $1/N$
  – Error decreases in $1/\sqrt{n}$

• Good news:
  – Math are mostly over for today
  – OK, it’s bad news if you like math
    (and you should)
Advantages of MC Integration

• Few restrictions on the integrand
  – Doesn’t need to be continuous, smooth, ...
  – Only need to be able to evaluate at a point
• Extends to high-dimensional problems
  – Same convergence
• Conceptually straightforward
• Efficient for solving at just a few points
Disadvantages of MC

• Noisy
• Slow convergence
• Good implementation is hard
  – Debugging code
  – Debugging maths
  – Choosing appropriate techniques
Questions?

• Images by Veach and Guibas

Naïve sampling strategy    Optimal sampling strategy
Smarter sampling

Sample a non-uniform probability
Called importance sampling
how to get probabilities right?
Example: Glossy rendering

- Integrate over hemisphere
- BRDF times cosine times incoming light

$I(\omega_i)$
Sampling a BRDF

$U(\omega_i)$

$P(\omega_i)$

Slide courtesy of Jason Lawrence
Sampling a BRDF

$U(\omega_i)$

$P(\omega_i)$

25 Samples/Pixel

Slide courtesy of Jason Lawrence

MIT EECS 6.837
Sampling a BRDF

\[ U(\omega_i) \]

\[ P(\omega_i) \]

75 Samples/Pixel

Slide courtesy of Jason Lawrence
Questions?

1200 Samples/Pixel

Traditional importance function

Better importance by Lawrence et al.
Stratified sampling

• With uniform sampling, we can get unlucky
  – E.g. all samples in a corner
• To prevent it, subdivide domain $\Omega$ into non-overlapping regions $\Omega_i$
  – Each region is called a stratum
• Take one random samples per $\Omega_i$
Example

- Borrowed from Henrik Wann Jensen

\[ f(x) = e^{\sin(3x^2)} \]

<table>
<thead>
<tr>
<th>N</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.75039</td>
</tr>
<tr>
<td>10</td>
<td>1.9893</td>
</tr>
<tr>
<td>100</td>
<td>1.79139</td>
</tr>
<tr>
<td>1000</td>
<td>1.75146</td>
</tr>
<tr>
<td>10000</td>
<td>1.77313</td>
</tr>
<tr>
<td>100000</td>
<td>1.77862</td>
</tr>
</tbody>
</table>

\[ f(x) = e^{\sin(3x^2)} \]

<table>
<thead>
<tr>
<th>N</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.70457</td>
</tr>
<tr>
<td>10</td>
<td>1.72858</td>
</tr>
<tr>
<td>100</td>
<td>1.77925</td>
</tr>
<tr>
<td>1000</td>
<td>1.77606</td>
</tr>
<tr>
<td>10000</td>
<td>1.77610</td>
</tr>
<tr>
<td>100000</td>
<td>1.77610</td>
</tr>
</tbody>
</table>

Unstratified
\[ O\left(1/\sqrt{N}\right) \]

Stratified
\[ O\left(1/N\right) \]
Stratified sampling - bottomline

• Cheap and effective
  – Mostly for low-dimensional domains

• Typical example: jittering for antialiasing
  – Signal processing perspective: better than uniform because less aliasing (spatial patterns)
  – Monte-Carlo perspective: better than random because lower variance (error for a given pixel)
Questions?

• Image from the ARNOLD Renderer by Marcos Fajardo
Global illumination

- e.g. indirect lighting bouncing off the walls
The Rendering Equation

\[ L(x', \omega') = E(x', \omega') + \int \rho_{x'}(\omega, \omega')L(x, \omega)G(x, x')V(x, x') \, dA \]

- emission
- BRDF
- Incoming light
- Geometric term
- visibility

[Kajiya 1986]
Ray Casting

- Cast a ray from the eye through each pixel
Ray Tracing

- Cast a ray from the eye through each pixel
- Trace secondary rays (light, reflection, refraction)
Monte-Carlo Ray Tracing

- Cast a ray from the eye through each pixel
- Cast random rays from the visible point
  - Accumulate radiance contribution
Monte-Carlo Ray Tracing

- Cast a ray from the eye through each pixel
- Cast random rays from the visible point
- Recurse
Monte-Carlo

- Cast a ray from the eye through each pixel
- Cast random rays from the visible point
- Recurse
Monte-Carlo

- Systematically sample primary light
Results
Monte Carlo Path Tracing

- Trace only one secondary ray per recursion
- But send many primary rays per pixel
- (performs antialiasing as well)
Results

- 10 paths/pixel

Think about it: we compute an infinite-dimensional integral with 10 samples!!!
Results: glossy

- 10 paths/pixel
Results: glossy

- 100 paths/pixel
Importance of sampling the light

Without explicit light sampling

1 path per pixel

With explicit light sampling

4 path per pixel
Why use random numbers?

- Fixed random sequence
- We see the structure in the error
Questions?

- Vintage path tracing by Kajyia
Questions?
Path Tracing is costly

- Needs tons of rays per pixel
Direct illumination
Global Illumination
Indirect illumination: smooth
Irradiance cache

- The indirect illumination is smooth
Irradiance cache

- The indirect illumination is smooth
Irradiance cache

- The indirect illumination is smooth
- Interpolate nearby values
Irradiance cache

- Store the indirect illumination
- Interpolate existing cached values
- But do full calculation for direct lighting
Irradiance caching

- Yellow dots: computation of indirect diffuse contribution
Radiance software by Greg Ward

- The inventor of irradiance caching
- http://radsite.lbl.gov/radiance/
Photon mapping

- Preprocess: cast rays from light sources
- Store photons
Photon mapping

- Preprocess: cast rays from light sources
- Store photons (position + light power + incoming direction)
Photon map

- Efficiently store photons for fast access
- Use hierarchical spatial structure (kd-tree)
Photon mapping - rendering

- Cast primary rays
- For secondary rays
  - reconstruct irradiance using adjacent stored photon
  - Take the $k$ closest photons
- Combine with irradiance caching and a number of other techniques
Photon map results
Photon mapping - caustics

- Special photon map for specular reflection and refraction
• 1000 paths/pixel
• Photon mapping
Photon mapping

- Animation by Henrik Wann Jensen
Questions?

- Image by Henrik
References

• 6.839!

• Eric Veach’s PhD dissertation

• Physically Based Rendering
  by Matt Pharr, Greg Humphreys
References
Questions?