Rendering: Path Tracing Basics

Bernhard Kerbl

Research Division of Computer Graphics
Institute of Visual Computing & Human-Centered Technology
TU Wien, Austria
Today’s Goal

- We combine the things we learned so far to make our first unbiased path tracer for diffuse materials:
  - Light Physics
  - Monte Carlo Integration
  - The Rendering Equation
  - The Path Tracing Algorithm

- Different iterations and considerations for performance

- Introduce a dedicated BSDF/BRDF material data structure
BSDF/BRDF..?

\[ L_e(x, v) = E(x, v) + \int_{\Omega} f_r(x, \omega \rightarrow v) L_i(x, \omega) \cos(\theta_x) \, d\omega \]

- Bidirectional Scattering Distribution Function (BSDF) accounts for the light transport properties of the hit material

- Bidirectional Reflectance Distribution Function (BRDF) considers only the **reflection** of incoming light onto a surface

- Only diffuse materials for now, more in Materials lecture
We usually distinguish three basic material types
- Perfectly diffuse (light is scattered equally in/from all directions)
- Perfectly specular (light is reflected in/from exactly one direction)
- Glossy (mixture of the other two, specular highlights)
Material Types

- We usually distinguish three basic material types
  - Perfectly diffuse (light is scattered equally in/from all directions)
  - Perfectly specular (light is reflected in/from exactly one direction)
  - Glossy (mixture of the other two, specular highlights)
What you will need

- A basic scene with objects, some of which emit light
- An output image to store color information in
- A camera model in the scene for which you can make rays
  - E.g., with this detailed description by Scratchapixel
- A function to trace rays through your scene and find the closest hit
  - E.g., using the famous Möller-Trumbore algorithm
- Additional information for each hit point (normal, material, object)
  - So you can detect if an object was a light source, or
  - Use surface normals for computations
  - ...

Rendering – Path Tracing Basics
Today’s Roadmap

- Rendering Equation Recap
- Direct Lighting
- Path Tracing v1.0
- BSDF Interface
- Russian Roulette
- Path Tracing v0.5
- Sample Distribution

What is indirect illumination?
How do multiple bounces work?
Can we add other effects too?
What is indirect illumination?
How do multiple bounces work?
Can we add other effects too?
Recursive Rendering Equation, Recap

$$L_e(x, v) = E(x, v) + \int_{\Omega} f_r(x, \omega \rightarrow v) L_i(x, \omega) \cos(\theta_x) \, d\omega$$

- Light going in direction $v$
- Light emitted from $x$ in direction $v$
- Material, modelled by the BRDF
- Light from direction $\omega$
- Solid angle
Recursive Rendering Equation, Recap

\[ L_e(x, v) = E(x, v) + \int_{\Omega} f_r(x, \omega \rightarrow v) L_i(x, \omega) \cos(\theta_x) \, d\omega \]

- Light going in direction \( v \)
- Light emitted from \( x \) in direction \( v \)
- Material, modelled by the BRDF

**Evaluate** light from direction \( \omega \) **recursively**

Solid angle
Recursive Rendering Equation, Recap

- To get the next bounce, we just evaluate this function recursively

\[
L(x_1 \rightarrow v) = E(x_1 \rightarrow v) + \int_{\Omega_1} f_r(x_1, \omega_1 \rightarrow v) L(x_1 \leftarrow \omega_1) \cos(\theta_x) \, d\omega_1
\]

\[
L(x_1 \leftarrow \omega_1) = L(x_2 \rightarrow \omega_2) !
\]

\[
L(x_2 \rightarrow \omega_2) = E(x_2 \rightarrow \omega_2) + \int_{\Omega_2} f_r(x_2, \omega \rightarrow \omega_2) L(x_2 \leftarrow \omega) \cos(\theta_x) \, d\omega
\]
Today’s Roadmap

What is indirect illumination? How do multiple bounces work? Can we add other effects too?

Direct Lighting

Path Tracing v1.0

BSDF Interface

Russian Roulette

Path Tracing v0.5

Sample Distribution

Rendering Equation Recap
- We trace rays through a pixel that randomly bounce into the scene.
  - Some will get lucky and hit light sources at first or second hit point!
Let’s use the rendering equation to resolve direct light

\[ L(x \rightarrow v) = E(x \rightarrow v) + \int_{\Omega} f(x, \omega \rightarrow v)L(x \leftarrow \omega) \cos(\theta_\omega) \, d\omega \]
Let’s simplify our notation a bit for our current scenario

\[ E(x \rightarrow v) \quad \ldots \quad E_x \]

\[ f(x, \omega \rightarrow v) \quad \ldots \quad \frac{1}{\pi} \]

\[ L(x \rightarrow v) = E_x + \int_{\Omega} \frac{1}{\pi} L(x \leftarrow \omega) \cos(\theta_{\omega}) \ d\omega \]
Let’s simplify our notation a bit for our current scenario

\[ L(x \rightarrow v) = E_x + \int_{\Omega} \frac{1}{\pi} L(x \leftarrow \omega) \cos(\theta_\omega) d\omega \]
If all we look for is direct lighting, then we stop after the first bounce.

$$L(x \to v) = E_x + \int \frac{1}{\pi} (E_y + \int_{\Omega'} \ldots) \cos(\theta_\omega) \, d\omega$$
If all we look for is direct lighting, then we stop after the first bounce.

\[ L(x \rightarrow v) = E_x + \int_{\Omega} \frac{1}{\pi} E_y \cos(\theta_\omega) \, d\omega \]
Direct Lighting with the Rendering Equation

- Replace indefinite integral with Monte Carlo integral

\[ L(x \rightarrow v) = E_x + \frac{1}{N} \sum_{i=1}^{N} \left( \frac{1}{\pi} E_y \cos(\theta_{\omega_i}) \frac{1}{p(\omega_i)} \right) \]
A quick word on $\omega$ and $p(\omega)$

- Ensure for Monte Carlo integration that $\omega$ and $p(\omega)$ match
  - This can actually be quite tricky!

- You can use **uniform** hemisphere sampling (we will derive it later)
  - For each $\omega$, draw two uniform random numbers $x_1, x_2$ in range $[0, 1)$
  - Calculate $\cos(\theta) = x_1$, $\sin(\theta) = \sqrt{1 - \cos^2(\theta)}$
  - Calculate $\cos(\phi) = \cos(2\pi x_2)$, $\sin(\phi) = \sin(2\pi x_2)$
  - $\omega = Vector3(\cos(\phi) \sin(\theta), \sin(\phi) \sin(\theta), \cos(\theta))$
  - $p(\omega) = \frac{1}{2\pi}$ (yes, always!)
Bring $\omega$ into “world space”

- Resulting $\omega$ is in local coordinate frame: $z$ axis is normal to surface

- To intersect scene, rays will usually need to be in *world space*

- Use coordinate transform between local and world\textsuperscript{[1]}. Nori users:
  - From world space to local $\rightarrow$ \texttt{Intersection::toLocal()}
  - From local to world space $\rightarrow$ \texttt{Intersection::shFrame::toWorld()}

\[22\]
Pseudo Code for Direct Lighting of each Pixel \((px, py)\)

\[
v_{\text{inv}} = \text{camera.gen_ray}(px, py)\\
x = \text{scene.trace}(v_{\text{inv}})\\
pixel\_color = x.\text{emit}\\
f = 0\\
\text{for } (i = 0; i < N; i++)
\]

\[
\omega_i, \text{prob}_i = \text{hemisphere_uniform_world}(x)\\
r = \text{make_ray}(x, \omega_i)\\
y = \text{scene.trace}(r)\\
f += 1/\pi \times y.\text{emit} \times \text{dot}(x.\text{normal}, \omega_i) / \text{prob}_i
\]

\[
pixel\_color += f/N
\]

\[
L(x \rightarrow v) = E_x + \frac{1}{N} \sum_{i=1}^{N} \left( \frac{1}{\pi} E_y \cos(\theta_{\omega_i}) \frac{1}{p(\omega_i)} \right)
\]
Direct Lighting with the Rendering Equation

Success!

\[ L(x \rightarrow v) = E_x + \frac{1}{N} \sum_{i=1}^{N} \left( \frac{1}{\pi} E_y \cos(\theta_{\omega_i}) \frac{1}{p(\omega_i)} \right) \]
We can move the sum out...

```python
for (i = 0; i < N; i++)
    v_inv = camera.gen_ray(px, py)
    x = scene.trace(v_inv)
    f = x.emit
    omega_i, prob_i = hemisphere_uniform_world(x)
    r = make_ray(x, omega_i)
    y = scene.trace(r)
    f = 1/pi * y.emit * dot(x.normal, omega_i) / prob_i
    pixel_color += f
    pixel_color /= N
```

\[
L(x \rightarrow v) = \frac{1}{N} \sum_{i=1}^{N} \left( E_x + \frac{1}{\pi} E_y \cos(\theta_{\omega_i}) \frac{1}{p(\omega_i)} \right)
\]
...and call a function to compute color from the view ray

```python
for (i = 0; i < N; i++)
    v_inv = camera.gen_ray(px, py)
    pixel_color += Li(v_inv)
pixel_color /= N
```

```python
function Li(v_inv)  // can be anything! Direct light, AO...
    ...  // Must return a Monte Carlo f(x)/p(x)
```

Note: In our test framework, Nori, the different integrators will implement different behaviors for \( Li \). The main loop over \( N \) takes care of the **sum** and the **division by \( N \)**, which are part of a Monte Carlo integral:

\[
\frac{1}{N} \sum_{i=0}^{N} \frac{f(x)}{p(x)}
\]

It is our job to write integrator functions that return \( \frac{f(x)}{p(x)} \) so the result will be a proper Monte Carlo integration.
What if I don’t hit anything in the scene?

- Importance rays go on forever
- In reality, sooner or later you would hit some sort of medium
- In practice, your digital scene might just “end”
- In that case, there are no follow-up bounces
- Just return 0, or “black”
Path Tracing

Just keep bouncing

...this one weird trick instantly makes your renderings prettier!
What is indirect illumination?
How do multiple bounces work?
Can we add other effects too?

Russian Roulette
BSDF Interface
Path Tracing v1.0
Direct Lighting
Path Tracing v0.5
Sample Distribution
Updating our Material Term

- Before, we assumed diffuse, white material term $\frac{1}{\pi}$

- If you have material information for your hit points, use that instead

- Produce completely diffuse materials, but now with color

- Assumed material information: albedo (diffuse RGB color) in [0, 1)
  - For each hit point, read out the albedo $\rho$
  - Now we will be using $f_r = \frac{\rho}{\pi}$
Results from following view path over multiple bounces

source: own work
Indirect Illumination

- Difficult in real-time graphics – comes naturally in path tracing!
Let’s go one step further!

\[ E(x \rightarrow v) \quad \ldots \quad E_x \]

\[ f(x, \omega \rightarrow v) \quad \ldots \quad f_r \]

\[ L(x \rightarrow v) = E_x + \int_{\Omega} f_r L(x \leftarrow \omega) \cos(\theta_\omega) \, d\omega \]
Let's go one step further!

\[
L(x \rightarrow v) = E_x + \int_{\Omega} f_r L(x \leftarrow \omega) \cos(\theta_{\omega}) \, d\omega
\]
Let’s look at this new equation in detail...

\[
L(x \rightarrow v) = E_x + \int_{\Omega} f_r \left( E_{x'} + \int_{\Omega'} f_r' L(x' \leftarrow \omega') \cos(\theta_{\omega'}) \, d\omega' \right) \cos(\theta_{\omega}) \, d\omega
\]
Indirect Lighting with the Rendering Equation

A pattern emerges!

\[ L(x \rightarrow v) = E_x + \int_{\Omega} f_r \left( E_{x'} + \int_{\Omega'} f_r' L(x' \leftarrow \omega') \cos(\theta_{\omega'}) \, d\omega' \right) \cos(\theta_{\omega}) \, d\omega \]
A real solution could go on much longer (ideally indefinitely)

\[ L(x \rightarrow v) = E_x + \int_{\Omega} f_r \left( E_{x'} + \int_{\Omega'} f_{r'} \ldots \cos(\theta_{\omega'}) d\omega' \right) \cos(\theta_{\omega}) d\omega \]
With Monte Carlo integration, we might replace all \( \int \) with \( \sum \)

\[
L(x \to v) = E_x + \frac{1}{N} \sum_{i=1}^{N} f_r \left( E_{x'} + \frac{1}{N} \sum_{j=1}^{N} f'_{r} \cdots \cos(\theta_{\omega_j}) \frac{1}{p(\omega_j)} \right) \cos(\theta_{\omega_i}) \frac{1}{p(\omega_i)}
\]
Indirect Lighting with the Rendering Equation

With Monte Carlo integration, we might replace all $\int$ with $\sum$

$$L(x \rightarrow v) = E_x + \frac{1}{N} \sum_{i=1}^{N} f_r \left( E_{x'} + \frac{1}{N} \sum_{j=1}^{N} f_{r'} \cdots \cos(\theta_{\omega_j}) \frac{1}{p(\omega_j')} \right) \cos(\theta_{\omega_i}) \frac{1}{p(\omega_i)}$$
Let’s implement this with recursion!

\[ v_{\text{inv}} = \text{camera}.\text{gen\_ray}(px, py) \]
\[ \text{pixel\_color} = L_i(v_{\text{inv}}) \]

\[
\text{function } L_i(v_{\text{inv}}) \\
\quad x = \text{scene}.\text{trace}(v_{\text{inv}}) \\
\quad f = 0 \\
\quad \text{for } (i = 0; i < N; i++) \\
\quad \quad \text{omega}_i, \text{prob}_i = \text{hemisphere\_uniform\_world}(x) \\
\quad \quad \text{ray} = \text{make\_ray}(x, \text{omega}_i) \\
\quad \quad f += \frac{x.\text{alb}}{\pi} * L_i(\text{ray}) * \text{dot}(x.\text{normal}, \text{omega}_i) / \text{prob}_i \\
\quad \text{return } x.\text{emit} + f/N
\]
Each hemisphere integral computed with N samples (seems fair!)

Question: what to pick for N?
- Let’s start with something low
- Like 4, 8, 16, 32

Let’s see what we get...

Renderer never stops. We need a stopping criterion!
Adding hard stopping criterion

\[
v_{inv} = \text{camera.gen\_ray}(px, py) \\
\text{pixel\_color} = L_i(v_{inv}, 0)
\]

function \(L_i(v_{inv}, D)\)

\[
\text{if } (D >= \text{NUM\_BOUNCES}) \\
\quad \text{return } 0 \\
x = \text{scene.trace}(v_{inv}) \\
f = 0 \\
\text{for } (i = 0; i < N; i++) \\
\quad \text{omega}_i, \text{prob}_i = \text{hemisphere\_uniform\_world}(x) \\
\quad \text{ray} = \text{make\_ray}(x, \text{omega}_i) \\
\quad f += x.\text{alb} / \pi * L_i(\text{ray}, D+1) * \text{dot}(x.\text{normal}, \text{omega}_i) / \text{prob}_i \\
\text{return } x.\text{emit} + f/N
\]
Let’s run this again!

- NUM_BOUNCES = 3

- Samples for each sum and runtime

<table>
<thead>
<tr>
<th>N</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3s</td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>
Let’s run this again!

- NUM_BOUNCES = 3

- Samples for each sum and runtime

<table>
<thead>
<tr>
<th>N</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3s</td>
</tr>
<tr>
<td>8</td>
<td>22s</td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>
Let's run this again!

- **NUM_BOUNCES = 3**

- **Samples for each sum and runtime**

<table>
<thead>
<tr>
<th>N</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3s</td>
</tr>
<tr>
<td>8</td>
<td>22s</td>
</tr>
<tr>
<td>16</td>
<td>3m 33s</td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>
Let’s run this again!

- **NUM_BOUNCES = 3**

- Samples for each sum and runtime

<table>
<thead>
<tr>
<th>N</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3s</td>
</tr>
<tr>
<td>8</td>
<td>22s</td>
</tr>
<tr>
<td>16</td>
<td>3m 33s</td>
</tr>
<tr>
<td>32</td>
<td>43m 20s</td>
</tr>
</tbody>
</table>
Wisdom of the Day

The more samples, the better

...and also slower
Today’s Roadmap

- Rendering Equation Recap
- Direct Lighting
- Path Tracing v1.0
- BSDF Interface
- Russian Roulette
- Sample Distribution
- Path Tracing v0.5

- What is indirect illumination?
- How do multiple bounces work?
- Can we add other effects too?
It’s time to reconsider our approach

- It works, but it’s slow and the quality is not great
- Also, this was only 3 bounces
- Advanced effects can require much more than that!
- Were we too naive, is path tracing doomed?
  - Definitely no!
  - There’s a vast range of tools we can use to raise performance
We can write this one big integral slightly differently

\[
L(x \to v) = E_x + \int_{\Omega} f_r \left( E_{x'} + \int_{\Omega'} f_{r'} \cos(\theta_{\omega'}) d\omega' \right) \cos(\theta_{\omega}) d\omega
\]

\[
L(x \to v) = E_x \\
+ \int_{\Omega} f_r E_{x'} \cos(\theta_{\omega}) d\omega \\
+ \int_{\Omega} f_r \int_{\Omega'} f_{r'} E_{x''} \cos(\theta_{\omega'}) \cos(\theta_{\omega}) d\omega' d\omega \\
+ \int_{\Omega} f_r \int_{\Omega'} f_{r'} \int_{\Omega''} f_{r''} E_{x'''} \cos(\theta_{\omega''}) \cos(\theta_{\omega'}) \cos(\theta_{\omega}) d\omega'' d\omega' d\omega \\
+ \ldots
\]
Reconsidering Sample Distribution

We can write this one big integral slightly differently

\[
L(x \rightarrow v) = E_x + \int_{\Omega} f_r \left( E_{x'} + \int_{\Omega'} f_{r'} \cdots \cos(\theta_{\omega'}) d\omega' \right) \cos(\theta_{\omega}) d\omega \\
L(x \rightarrow v) = \boxed{E_x} + \int_{\Omega} f_r E_{x'} \cos(\theta_{\omega}) d\omega \\
+ \int_{\Omega} f_r \int_{\Omega'} f_{r'} E_{x''} \cos(\theta_{\omega'}) \cos(\theta_{\omega}) d\omega' d\omega \\
+ \int_{\Omega} f_r \int_{\Omega'} f_{r'} \int_{\Omega''} f_{r''} E_{x'''} \cos(\theta_{\omega''}) \cos(\theta_{\omega'}) \cos(\theta_{\omega}) d\omega'' d\omega' d\omega \\
+ \cdots
\]

Can you guess what each of these lines does for our final image?
Reconsidering Sample Distribution

\[ L(x \rightarrow v) = E_x \]

\[ + \int_{\Omega} f_r E_x' \cos(\theta_\omega) \, d\omega \]

\[ + \int_{\Omega} f_r \int_{\Omega'} f_r' E_x'' \cos(\theta_{\omega'}) \cos(\theta_\omega) \, d\omega' \, d\omega \]

\[ + \int_{\Omega} f_r \int_{\Omega'} f_r' \int_{\Omega''} f_r''' E_x''' \cos(\theta_{\omega''}) \cos(\theta_{\omega'}) \cos(\theta_\omega) \, d\omega'' \, d\omega' \, d\omega \]

\[ + \ldots \]
\[ L(x \rightarrow v) = E_x \]
\[ + \int_{\Omega} f_r E_x' \cos(\theta_\omega) \, d\omega \]
\[ + \int_{\Omega} f_r \int_{\Omega'} f_r' E_{x''} \cos(\theta_{\omega'}) \cos(\theta_\omega) \, d\omega' \, d\omega \]
\[ + \int_{\Omega} f_r \int_{\Omega'} f_r' \int_{\Omega''} f_r''' E_{x'''} \cos(\theta_{\omega''}) \cos(\theta_{\omega'}) \cos(\theta_\omega) \, d\omega'' \, d\omega' \, d\omega \]
\[ + \ldots \]
Reconsidering Sample Distribution

\[ L(x \rightarrow v) = E_x + \int_{\Omega} f_r E_{x'} \cos(\theta_{\omega}) \, d\omega \]

\[ + \int_{\Omega} \int_{\Omega'} f_r' E_{x''} \cos(\theta_{\omega'}) \cos(\theta_{\omega}) \, d\omega' \, d\omega \]

\[ + \int_{\Omega} \int_{\Omega'} \int_{\Omega''} f_r'' E_{x'''} \cos(\theta_{\omega''}) \cos(\theta_{\omega'}) \cos(\theta_{\omega}) \, d\omega'' \, d\omega' \, d\omega \]

\[ + \ldots \]
Reconsidering Sample Distribution

\[ L(x \rightarrow v) = \begin{align*} & E_x \\
& + \int_\Omega f_r E_{x'} \cos(\theta_\omega) \, d\omega \\
& + \int_\Omega f_r \int_{\Omega'} f_r' E_{x''} \cos(\theta_{\omega'}) \cos(\theta_\omega) \, d\omega' \, d\omega \\
& + \int_\Omega f_r \int_{\Omega'} f_r' \int_{\Omega''} f_r'' E_{x'''} \cos(\theta_{\omega''}) \cos(\theta_{\omega'}) \cos(\theta_\omega) \, d\omega'' \, d\omega' \, d\omega \\
& + \ldots \end{align*} \]
We used more samples for the contributions from longer light paths

The return-on-investment is not that great!

1 sample  N samples  N^2 samples  N^3 samples
We have seen a different way of writing these results last time.

The path integral form used a single integral for each bounce!

\[
L(x \rightarrow v) = E_x + \int_{\Omega} f_r E_{x'} \cos(\theta_\omega) \, d\omega
\]

\[
+ \int_{\Omega} f_r \int_{\Omega'} f_r' E_{x''} \cos(\theta_{\omega'}) \cos(\theta_\omega) \, d\omega' \, d\omega
\]

\[
+ \int_{\Omega} f_r \int_{\Omega'} f_r' \int_{\Omega''} f_r'' E_{x'''} \cos(\theta_{\omega''}) \cos(\theta_{\omega'}) \cos(\theta_\omega) \, d\omega'' \, d\omega' \, d\omega
\]

+ ...
We have seen a different way of writing these results last time.

The path integral form used a single integral for each bounce!

\[
L(x \to v) = E_x + \int_{\Omega_1} f_r E'_{x'} \cos(\theta_\omega) \, d\mu(x) \\
+ \int_{\Omega_2} f_r f_r' E''_{x''} \cos(\theta_{\omega'}) \cos(\theta_\omega) \, d\mu(x) \\
+ \int_{\Omega_3} f_r f_r f_r' E'''_{x'''} \cos(\theta_{\omega''}) \cos(\theta_{\omega'}) \cos(\theta_\omega) \, d\mu(x) \\
+ \ldots
\]
Let’s replace each integral with Monte Carlo integration again

\[
L(x \rightarrow v) = E_x \\
+ \frac{1}{N} \sum_{i=1}^{N} f_r \ E_x' \cos(\theta_\omega) \frac{1}{p(\omega)} \\
+ \frac{1}{N} \sum_{i=1}^{N} f_r f_x' \cos(\theta_{\omega'}) \cos(\theta_\omega) \frac{1}{p(\omega)p(\omega')} \\
+ \frac{1}{N} \sum_{i=1}^{N} f_r f_x' f_x'' \cos(\theta_{\omega''}) \cos(\theta_{\omega'}) \cos(\theta_\omega) \frac{1}{p(\omega)p(\omega')p(\omega'')} \\
+ \ldots
\]
Our new sample distribution

- We use the same number of samples for all light paths

- No more exponential sample growth!

N samples

N samples

N samples

N samples
Let’s replace each integral with Monte Carlo integration again

\[
L(x \rightarrow v) = E_x \\
+ \frac{1}{N} \sum_{i=1}^{N} f_r E_x' \cos(\theta_\omega) \frac{1}{p(\omega)} \\
+ \frac{1}{N} \sum_{i=1}^{N} f_r f_r' E_x'' \cos(\theta_\omega') \cos(\theta_\omega) \frac{1}{p(\omega)p(\omega')} \\
+ \frac{1}{N} \sum_{i=1}^{N} f_r f_r' f_r'' E_x''' \cos(\theta_\omega'') \cos(\theta_\omega') \cos(\theta_\omega) \frac{1}{p(\omega)p(\omega')p(\omega'')} \\
+ \ldots
\]
Reconsidering Sample Distribution

We can again pull the sum to the front...

\[
L(x \to v) = \frac{1}{N} \sum_{i=1}^{N} (E_x \\
+ f_r E_{x'} \cos(\theta_\omega) \frac{1}{p(\omega)} \\
+ f_r f_r' E_{x''} \cos(\theta_{\omega'}) \cos(\theta_\omega) \frac{1}{p(\omega)p(\omega')} \\
+ f_r f_r' f_r'' E_{x'''} \cos(\theta_{\omega''}) \cos(\theta_{\omega'}) \cos(\theta_\omega) \frac{1}{p(\omega)p(\omega')p(\omega'')} \\
+ \ldots)
\]
…and rewrite to highlight the original recursion

\[ L(x \rightarrow v) = \frac{1}{N} \sum_{i=1}^{N} \left( E_x + f_r \left( E_{x'} + f_r' \left( \ldots \right) \cos \theta \omega' \frac{1}{p(\omega')} \right) \cos(\theta \omega) \frac{1}{p(\omega)} \right) \]

We are back to a single sum for integration with recursion!

This also fits perfectly with our main loop and interface design
Path Tracing with a Single Sum

for (i = 0; i < N; i++)
    v_inv = camera.gen_ray(px, py)
    pixel_color += Li(v_inv, 0)
pixel_color /= N

function Li(v_inv, D)
    if (D >= NUM_BOUNCES)
        return 0
    x = scene.trace(v_inv)
    f = x.emit
    omega, prob = hemisphere_uniform_world(x)
    r = make_ray(x, omega)
    f += x.alb/pi * Li(r, D+1) * dot(x.normal, omega)/prob
    return f

\[
L(x \rightarrow v) = \frac{1}{N} \sum_{i=1}^{N} \left( E_x + f_r (...) \cos(\theta_\omega) \frac{1}{p(\omega)} \right)
\]
Path Tracing with a Single Sum

```python
for (i = 0; i < N; i++)
    v_inv = camera.gen_ray(px, py)
    pixel_color += Li(v_inv, 0)
pixel_color /= N
```

```python
function Li(v_inv, D)
    if (D >= NUM_BOUNCES)
        return 0
    x = scene.trace(v_inv)
    f = x.emit
    omega, prob = hemisphere_uniform_world(x)
    r = make_ray(x, omega)
    f += x.alb/pi * Li(r, D+1) * dot(x.normal, omega)/prob
    return f
```

\[
L(x \rightarrow v) = \frac{1}{N} \sum_{i=1}^{N} \left( E_x + f_r (...) \cos(\theta_\omega) \frac{1}{p(\omega)} \right)
\]
Path Tracing with a Single Sum

for (i = 0; i < N; i++)
    v_inv = camera.gen_ray(px, py)
    pixel_color += Li(v_inv, 0)
pixel_color /= N

function Li(v_inv, D)
    if (D >= NUM_BOUNCES)
        return 0
    x = scene.trace(v_inv)
    f = x.emit
    omega, prob = hemisphere_uniform_world(x)
    r = make_ray(x, omega)
    f += x.alb/pi * Li(r, D+1) * dot(x.normal, omega)/prob
    return f

$L(x \rightarrow v) = \frac{1}{N} \sum_{i=1}^{N} \left( E_x + f_r (...) \cos(\theta_\omega) \frac{1}{p(\omega)} \right)$
Path Tracing with a Single Sum

for (i = 0; i < N; i++)
    v_inv = camera.gen_ray(px, py)
    pixel_color += Li(v_inv, 0)
pixel_color /= N

function Li(v_inv, D)
    if (D >= NUM_BOUNCES)
        return 0
    x = scene.trace(v_inv)
    f = x.emit
    omega, prob = hemisphere_uniform_world(x)
    r = make_ray(x, omega)
    f += x.alb/pi * Li(r, D+1) * dot(x.normal, omega)/prob
    return f

$L(x \rightarrow v) = \frac{1}{N} \sum_{i=1}^{N} \left( E_x + f_r(\ldots) \cos(\theta_\omega) \frac{1}{p(\omega)} \right)$
Path Tracing with a Single Sum

for (i = 0; i < N; i++)
    v_inv = camera.gen_ray(px, py)
    pixel_color += Li(v_inv, 0)
pixel_color /= N

function Li(v_inv, D)
    if (D >= NUM_BOUNCES)
        return 0
    x = scene.trace(v_inv)
    f = x.emit
    omega, prob = hemisphere_uniform_world(x)
    r = make_ray(x, omega)
    f += x.alb/pi * \( \text{Li}(r, D+1) \) * \( \text{dot}(x.\text{normal}, \omega)/\text{prob} \)
    return f

\[
L(x \to v) = \frac{1}{N} \sum_{i=1}^{N} \left( E_x + f_r \cos(\theta_\omega) \frac{1}{p(\omega)} \right)
\]
Path Tracing with a Single Sum

for (i = 0; i < N; i++)
    v_inv = camera.gen_ray(px, py)
    pixel_color += Li(v_inv, 0)
pixel_color /= N

function Li(v_inv, D)
    if (D >= NUM_BOUNCES)
        return 0
    x = scene.trace(v_inv)
    f = x.emit
    omega, prob = hemisphere_uniform_world(x)
    r = make_ray(x, omega)
    f += x.alb/pi * Li(r, D+1) * dot(x.normal, omega)/prob
    return f

\[ L(x \rightarrow v) = \frac{1}{N} \sum_{i=1}^{N} \left( E_x + f_r \left( \ldots \right) \cos(\theta_\omega) \frac{1}{p(\omega)} \right) \]
Path Tracing Implementations – Comparison

3 bounces, 3 nested sums, $N = 16$

3 bounces, 1 sum, $N = 2048$
Wisdom of the Day

Your samples are precious

...put them where they matter!
How many bounces are enough?

- Remember: if we want to be physically correct, then we must consider all possible light paths (i.e., journeys of photons)

- Photons stop bouncing when they have been entirely absorbed

- Problem: no real-world material absorbs 100% of incoming light

- No matter how many bounces, the probability might never go to absolute zero → so we should never stop?
Today’s Roadmap

- Rendering Equation Recap
- Direct Lighting
  - Path Tracing v1.0
  - BSDF Interface
  - Russian Roulette
  - Sample Distribution
  - Path Tracing v0.5

What is indirect illumination? How do multiple bounces work? Can we add other effects too?
How do we handle infinity?

- In many cases, most contribution comes from the first few bounces.

Can we exploit this fact and make long paths possible, but unlikely?
How do we handle infinity?

- In many cases, most contribution comes from the first few bounces.

![Diagram showing different stages of light bounces](image)

Can we exploit this fact and make long paths possible, but unlikely?
**Russian Roulette (RR)**

- Pick $0 < p_{RR} < 1$. Draw uniform random value $x$ in $[0, 1)$ to decide
  - $x < p_{RR}$: keep going for another bounce
  - $x \geq p_{RR}$: end path

- The longer a path goes on, the more likely it is to get terminated

- The probability of a ray surviving the $D^{th}$ bounce is $p_{RR}^D$

- Whenever a path continues with another bounce, compensate for its (un)-likeliness by weighting the returned color from $L$ with $\frac{1}{p_{RR}}$
Both objects emit light

Very aggressive RR

Assume fixed $p_{RR} = \frac{1}{5}$

Most paths ($\frac{4}{5}$) fail to capture the pink sphere
Both objects emit light

Very aggressive RR

Assume fixed $p_{RR} = \frac{1}{5}$

Most paths $\left(\frac{4}{5}\right)$ fail to capture the pink sphere
• Both objects emit light

• Very aggressive RR

• Assume fixed $p_{RR} = \frac{1}{5}$

• When we do hit it, the division by $p$ compensates for times where we missed it
Both objects emit light

Very aggressive RR

Assume fixed $p_{RR} = \frac{1}{5}$

The rarity (and multiplier) of later bounces is growing continuously!
In code, you might be tempted to use $\frac{1}{p_{RR}}$ to compensate for RR.

Don’t! $\frac{1}{p_{RR}}$ is enough for each individual bounce!

If you use the recursive implementation, your effective RR compensation will grow with each bounce, all by itself...

Maybe look at pseudocode and ponder the previous slide for a bit.
Russian Roulette Implementation

for (i = 0; i < N; i++)
    v_inv = camera.gen_ray(px, py)
    pixel_color += Li(v_inv, 0)
    pixel_color /= N

function Li(v_inv, D)
    ...
    f = x.emit
    ...
    rr_prob = Some value between 0 and 1
    if (uniform_random_value() >= rr_prob)  // This path ends here
        return f
    ...
    f += x.alb/pi * Li(r, D+1) * dot(x.normal, omega) / (prob * rr_prob)
    return f
“...but if the possibility for infinitely long paths remains, doesn’t that mean that my renderer may take forever to finish?”

Almost certainly no

In practice, if you choose an adequate $p_{RR}$, you are more likely to get struck by lightning while reading this than that ever happening

“Ok, cool, so the lower I choose $p_{RR}$, the better, right? Can we just take something really small?” Well, not exactly.
High $p_{RR}$ vs low $p_{RR}$, same number of samples

$p_{RR} = 0.7$: 50 seconds

$p_{RR} = 0.1$: 35 seconds
High $p_{RR}$ vs low $p_{RR}$, same number of samples

$p_{RR} = 0.7$: 50 seconds

$p_{RR} = 0.1$: 35 seconds
High $p_{RR}$ vs low $p_{RR}$, different number of samples

$p_{RR} = 0.7$: 50 seconds

$p_{RR} = 0.1$, 4x as many samples: 150 seconds
High $p_{RR}$ vs low $p_{RR}$, different number of samples

$p_{RR} = 0.7$: 50 seconds

$p_{RR} = 0.1$, 4x as many samples: 150 seconds
If $p(x)$ is low, but $f(x)$ is not $\to$ high contribution of rare events!

These “fireflies” tend to stick around!

Choose $p_{RR}$ dynamically: compute it at each bounce according to possible color contribution ("throughput")

$$p_{RR} = \max_{\text{RGB}} \left( \prod_{d=1}^{D-1} \left( \frac{f_r(x_d, \omega_d \rightarrow v_d) \cos \theta_d}{p(\omega_d)p_{RR_d}} \right) \right)$$
for (i = 0; i < N; i++)
    v_inv = camera.gen_ray(px, py)
    pixel_color += Li(v_inv, 0, 1)
    pixel_color /= N

function Li(v_inv, D, throughput)
    ...
    rr_prob = max_coefficient(throughput)  // Throughput is RGB
    ...
    brdf = x.alb/pi
    cosTheta = dot(x.normal, omega)
    throughput *= brdf * cosTheta / (prob * rr_prob)
    ...
    f += brdf * Li(r, D+1, throughput) * cosTheta / (prob * rr_prob)
    return f
Use guaranteed **minimal** path length before Russian Roulette starts

- E.g., no Russian Roulette before the third bounce
- Preserves a minimal path length for indirect illumination
- Guaranteed bounces have $p_{RR} = 1$ always

Some materials absorb barely any incoming light (mirrors!)

- Imagine two mirrors opposite of each other
- Ray may bounce between them forever
- Bad: limit bounces to a strict maximum
- Better: clamp $p_{RR}$ to a value $< 1$, e.g. 0.99
In practice, the distribution of samples is usually more dynamic:

- Especially bounce 1 (direct light/shadows) receives more attention.
You can interpret the $p_{RR}$ as an additional factor to MC integral.

Originally, we divided by $p(\omega)$ to account for two things:

- Scaling to approximate the whole domain (i.e., entire hemisphere)
- Different probabilities of $\omega$ (not a factor with uniform sampling)
Choosing and Compensating

- Probability density of following a particular $\omega$ changes to $p(\omega)p_{RR}$
- Compensate for the fact that sometimes we chose to stop
- Weight rare events higher (e.g., many bounces off dark materials)

- We have to compensate whenever we pick one of multiple options

- This is usually the case when you draw a random variable to decide on choosing one option and not the others
Another example: picking a light source for light source sampling

- For light source sampling, you can sample them all or just pick one
- If you do pick only one, must compensate for making that choice!
- Simplest: pick uniformly, multiply result by number of lights (why?)
Wisdom of the Day

Monte Carlo is all about picking samples and then compensating...

...and if you pick your samples carefully, we call it importance sampling.
Today's Roadmap

What is indirect illumination?
How do multiple bounces work?
Can we add other effects too?

Direct Lighting

Path Tracing v1.0

BSDF Interface

Russian Roulette

Sample Distribution

Path Tracing v0.5

Rendering Equation Recap

Equation Recap

BSDF Interface

Russian Roulette
Materials and the BSDF

- We made a path tracer for diffuse materials, and diffuse only
Capturing Path Behavior in a BSDF Data Structure

- We made a solution that works, but only for diffuse materials

- There are a lot of exciting other options
  - Specular
  - Glossy
  - ...

- Path tracer should allow adding materials without rewriting it all

- Encapsulate material-dependent rendering factors in BSDF class
function Li(v_inv, D, throughput)

... omega, prob = hemisphere_uniform_world(x)

... brdf = x.alb/pi
throughput *= brdf * cosTheta / (prob * rr_prob)

... f += brdf * Li(r, D+1, throughput) * cosTheta / (prob * rr_prob)

...
function Li(v_inv, D, throughput) {
    ... 
    omega, prob = hemisphere_uniform_world(x)
    ... 
    brdf = x.alb/pi
    throughput *= brdf * cosTheta / (prob * rr_prob)
    ... 
    f += brdf * Li(r, D+1, throughput) * cosTheta / (prob * rr_prob)
    ... 
}

Some materials will reflect incoming light entirely in one single direction (mirrors). Sampling the hemisphere in this case is pointless! Also: we might be able to do something smarter than uniform sampling.
function $\text{Li}(v_{\text{inv}}, D, \text{throughput})$

...  
\text{omega, prob} = \text{hemisphere_uniform_world}(x)  
...  
\text{brdf} = \frac{x.\text{alb}}{\pi}  
\text{throughput} *= \text{brdf} \times \frac{\cos\text{Theta}}{(\text{prob} \times \text{rr}_\text{prob})}  
...  
f += \text{brdf} \times \text{Li}(r, D+1, \text{throughput}) \times \frac{\cos\text{Theta}}{(\text{prob} \times \text{rr}_\text{prob})}  
...  

Super simple term that never changes. Obviously, this only makes sense if the amount of reflected light is the same in all directions, independent of $v$ and $\omega$. Only a fully diffuse BSDF gets away with this.
function Li(v_inv, D, throughput)

... 

omega, prob = hemisphere_uniform_world(x) 

... 

brdf = x.alb/pi 

throughput *= brdf * cosTheta / (prob * rr_prob) 

...

f += brdf * Li(r, D + 1, throughput) * cosTheta / (prob * rr_prob) 

...

For some materials, like glass, this cosine term is not needed, cancels out or has to be removed for reasons of energy conservation.
function Li(v_inv, D, throughput)
...

    omega, prob = hemisphere_uniform_world(x)
...
    brdf = x.alb/pi
    throughput *= brdf * \text{cosTheta} / (prob * rr\_prob)
...
    f += brdf * Li(r, D+1, throughput) * \text{cosTheta} / (prob * rr\_prob)
...

Many issues! Example: we could have perfect mirrors that only reflect in a single direction. Hence the probability of other directions is 0. Danger of division by 0!
Implement Basic Diffuse BRDFs with BSDF Interface

- Nori BSDF class has three methods: `eval`, `pdf`, `sample`

- Use auxiliary struct parameter `bRec` to pass $\nu$ ($\vec{w}_i$) and $\omega$ ($\vec{w}_o$)

- `eval(bRec)`: $f(x, \omega \rightarrow \nu)$, material ability to reflect light from $\omega$ to $\nu$

- `pdf(bRec)`: $p(\omega)$, compute probability density of choosing sample $\omega$

- `sample(bRec)`: make & store $\omega$ in `bRec`, compute material multiplier
Implement Basic Diffuse BRDFs with BSDF Interface

- Nori BSDF class has three methods: **eval**, **pdf**, **sample**

- Use auxiliary struct parameter `bRec` to pass $\nu$ (\( \cdot \nu_{\text{in}} \)) and $\omega$ (\( \cdot \omega_{\text{o}} \))

- **eval**(\( bRec \)): return $\frac{\rho}{\pi}$ if $\nu$, $\omega$ lie in hemisphere around $n$ (diffuse)

- **pdf**(\( bRec \)) : return $\frac{1}{2\pi}$ if $\omega$ lies in hemisphere around $n$ (all $\omega$ equal)

- **sample**(\( bRec \)): create uniform $\omega$, return $\left( \cos \theta \cdot \text{eval}() \right) / \text{pdf}()$
Using the New BSDF Class in Path Tracing Code

- Isolates material-specific factor computations in a single function

- Simplifies code and will make extension with other materials easy

```
function Li(v_inv, D, throughput)
...
    omega, brdf_multiplier = sample(x, v_inv)
...
    throughput *= brdf_multiplier / rr_prob
...
    f += brdf_multiplier * Li(r, D+1, throughput) / rr_prob
...
```
Path-Tracing is *Multidimensional*

- We already know some of them:
  - Constructing a new ray after each bounce ($2D$)
  - Evaluating RR continuation probability ($D$)
  - ...

- Other possible choices we have not yet considered:
  - Lens coordinates (for depth-of-field) (2)
  - Time (for motion blur) (1)
  - ...

Rendering – Path Tracing Basics
Depth-of-Field

- Simulate the behavior of camera lenses

- Depending on shape, focal length and aperture, lenses have limited distance range in which objects appear sharp

- If they are closer or farther away, they cause a blurry „circle of confusion“

- Can be used to highlight objects of interest
Can simulate depth-of-field for a thin lens with focal length $f$.

Create camera ray $r$ through pixel as before.

Find focal point $f$ along $r$ at distance $f$.

Pick random location $x, y$ on the lens (2D disk) inside the aperture.

Shoot camera ray from $x, y$ through $f$. 

Far from focal length (blurred)

Close to focal length (sharp)
Motion Blur

- Mostly an artistic effect to simulate a familiar camera phenomenon

- Occurs when medium exposure is longer than rate of motion of objects in the captured scene

- Can help convey the impression of moving objects in still image

- In path tracing, all we need is an additional integrated time variable
Motion Blur in Path Tracing

- Option 1: make camera position a function of time $t$
  - Draw a random $t$, create adapted view ray $r$
  - Follow path through the scene
  - Check which triangles ray intersects

- Option 2: make geometry a function of time $t$
  - Draw and store a random $t$, create view ray $r$
  - Follow path through the scene
  - Check which triangles ray intersects at $t$
Today’s Roadmap

- Rendering Equation Recap
- Direct Lighting
- Path Tracing v1.0
- Path Tracing v0.5
- BSDF Interface
- Sample Distribution
- Russian Roulette

What is indirect illumination?
How do multiple bounces work?
Can we add other effects too?
Everything that is wrong with our path tracer right now

- So far, we only rendered very simple scenes (Cornell box)

- What happens if we run a slightly more challenging scene?
  - Ajax bust, 500k triangles
  - Takes 17 hours (!) to get a boring, noisy image...

- Is path tracing doomed (again)?

- No! We will make better images in seconds!
Room for Improvement

- Economize on samples – squeeze out whatever we can
  - Better sampling strategies (importance sampling)
  - Exploiting light source sampling (next-event estimation)
  - Combining sampling strategies (multiple importance sampling)

- Improving our scene intersection tests
  - Build spatial acceleration structures
  - Optimized traversal strategies

- Support spectacular specular, glossy and transparent materials
References and Further Reading

[1] Creating an Orientation Matrix or Local Coordinate System
https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/geometry/creating-an-orientation-matrix-or-local-coordinate-system

https://medium.com/@elope139/depth-of-field-in-path-tracing-e61180417027
