Which is Real?

Left image is a photograph, right image is rendered by path tracing. This famous ground-truth test of a renderer is the origin of the “cornell box” 3D models—there’s a real box at Cornell.

Of course, modern renderers with good input can simulate non-realistic scenes:
In a way that still has realistic lighting, or fantastical geometry as if it was real...
I can’t tell what is captured by a camera and what is rendered anymore in movies...
In this scene from Deadpool, you probably expect there to be some computer graphics...
But did you expect *the entire scene* to be CGI?
Action movies are increasingly like this...
Iron Man 3
And pretty much every car commercial and car scene in a film is now virtual, thanks to The Mill, one of my favorite VFX studies.

This is terrific stuff. Let’s review the math that you studied previously in other graphics courses that lets us create algorithms for rendering images like these.
## Measurements

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Symb.</th>
<th>SI Unit</th>
<th>1 ≈</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Distance</strong></td>
<td>$x$</td>
<td>meters (m)</td>
<td>Counter height</td>
</tr>
<tr>
<td><strong>Area</strong></td>
<td>$A$</td>
<td>Square meters (m²)</td>
<td>Desk top</td>
</tr>
<tr>
<td><strong>Angle</strong></td>
<td>$\theta$</td>
<td>radians (rad)</td>
<td>57 degrees</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Northernmost latitude in ON</td>
</tr>
<tr>
<td><strong>Solid Angle</strong></td>
<td>$\Gamma$</td>
<td>steradians (sr)</td>
<td>Dodecahedron face</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The Americas</td>
</tr>
<tr>
<td><strong>Power</strong> (energy/time)</td>
<td>$\Phi$</td>
<td>Watts (W)</td>
<td>Cell phone flashlight</td>
</tr>
<tr>
<td><strong>Radiance</strong></td>
<td>$L$</td>
<td>W/(m² sr)</td>
<td>A bright indoor ray</td>
</tr>
<tr>
<td><strong>“Biradiance”</strong></td>
<td>$\beta$</td>
<td>W/m²</td>
<td>Cell flashlight at arm’s length</td>
</tr>
</tbody>
</table>

A tall person has a height of 2m and surface area of 2m².
class Vector3 {
public: float x, y, z;
};
typedef Vector3 Point3;

class Ray {
public:
    Point3 origin;
    Vector3 direction;
};
typedef Color3 Radiance3;
typedef Color3 Power3;

class Light {
public:
    Power3 power;
    Point3 position;
};

class Image {
public:
    Array<Color3> pixel;
    int width, height;
};

class Surface {
public:
    Array<Point3> position;
    Array<int> index;
};

Stop for questions on what we covered last time.
Transformations

\[ R_{(x,y,z),\theta} = I + \sin \theta \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} + (1 - \cos \theta) \begin{bmatrix} -y^2 - z^2 & xy & zx \\ xy & -z^2 - x^2 & yz \\ zx & yz & -x^2 - y^2 \end{bmatrix} \]

\[ S_{x,y,z} = \begin{bmatrix} x & 0 & 0 \\ 0 & y & 0 \\ 0 & 0 & z \end{bmatrix} \]

\[ T_{(x,y,z)} = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \end{bmatrix} \]

Roll: \[ R_{\hat{z},\theta} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \]

Yaw: \[ R_{\hat{y},\theta} = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix} \]

Pitch: \[ R_{\hat{x},\theta} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix} \]
Primary Rays

Screen space:
\(x, y, w, h\)

World space:
\(z_p = -1\)
\(H = -2 \tan(\theta_y / 2) z_p\)
\(W = H \times w / h\)

\(y_p = -(y / h - \frac{1}{2}) \times H\)
\(x_p = (x / w - \frac{1}{2}) \times W\)

(You don’t have to do this in this week’s project)
PHENOMENA
"Color Bleeding"
"Diffuse Interreflection"

"Soft Shadows"
“Ambient Occlusion”
The Rendering Equation

\[ \hat{\omega}_{0,Y} = -\hat{\omega}_{1,X} \]

Emitter

Surface

Camera

Virtual Image Plane

Real Image Plane

Pinhole Aperture

Sensor

Pixel
The Rendering Equation

\[ L_o(X, \hat{\omega}_o) = L_e(X, \hat{\omega}_o) + \int_{S^2} L_i(X, \hat{\omega}_i) f_X(\hat{\omega}_i, \hat{\omega}_o) |\hat{\omega}_i \cdot \hat{n}| d\hat{\omega}_i \]

Outgoing direction

Incoming direction

A point in the scene

\textbf{All incoming directions}
(a sphere)

\[ L_{\text{out}}(X, \hat{\omega}_o) = L_{\text{em}}(X, \hat{\omega}_o) + \int_{S^2} L_{\text{in}}(X, \hat{\omega}_i) f_X(\hat{\omega}_i, \hat{\omega}_o) |\hat{\omega}_i \cdot \hat{n}| d\hat{\omega}_i \]
The Rendering Equation

\[ L_0(X, \hat{\omega}_o) = L_e(X, \hat{\omega}_o) + \int_{S^2} L_i(X, \hat{\omega}_i) f_X(\hat{\omega}_i, \hat{\omega}_o) |\hat{\omega}_i \cdot \hat{n}| d\hat{\omega}_i \]

Outgoing light  Emitted light  Incoming light  Material
Monte Carlo Integration

\[
\int_{A \subseteq M} g(m \in M) \, dm \approx \sum_{j=0}^{N-1} \frac{g(m_j) \in G}{p(m_j) \in M^{-1}} \cdot \frac{1}{N}
\]

The integral we're estimating

Integrand

Probability density function for choosing \( m \)

Number of samples

Note that the units check out...the left side has units from \([G] \times [M]\), and the right is \([G] / [M]^{-1}\)

\( p() \) is a continuous probability DENSITY function. It has to be non-negative everywhere and integrate to 1 over \( M \). It can have individual values of any magnitude, though, including infinity, as long as that is for a very small portion of the domain.
Limiting Case

\[ \int_A g(m) \, dm \approx \sum_{j=0}^{N-1} \frac{g(m_j)}{p(m_j)} \frac{1}{N} \]

Let \( N = 1 \)

\[ \approx \frac{g(m)}{p(m)} \]

The simplest version of the integrator takes a single sample and uses that as the value, weighting it by \( 1/p \)

Let’s take this simple case and substitute the rendering equation’s integral into it...
Let $m = \hat{\omega}_i$

\[
g(m) = L_i(X, \hat{\omega}_i) \ f_X(\hat{\omega}_i, \hat{\omega}_o) \ |\hat{\omega}_i \cdot \hat{n}|.\]

\[
A = S^2
\]

\[
p(m) = \frac{1}{4\pi \ \text{sr}}
\]

\[
\int_{S^2} L_i(X, \hat{\omega}_i) \ f_X(\hat{\omega}_i, \hat{\omega}_o) \ |\hat{\omega}_i \cdot \hat{n}| \ d\hat{\omega}_i \approx \frac{g(m)}{p(m)}
\]

Let the pdf be uniform on the sphere. How do I choose an incoming vector uniformly at random on the sphere? Easy: choose a uniform point in the $[-1, 1]^3$ cube. Throw it away if it is outside of the sphere and try again.

Points in the unit radius ball map equally to the sphere, so if you just normalize the resulting point, it will be a uniformly distributed vector.

The whole thing now boils down into this simple expression, which is just the
integrand applied to a random direction times $4\pi$
Discover $Y$ by ray casting

Choose $\omega_i$ uniformly on $S^2$

$$L_i(X, \omega_i) = L_o(Y, -\omega_i)$$

$$L_o(X, \omega_o) = L_e(X, \omega_i) + L_i(X, \omega_i) f_X(\omega_i, \omega_o) |\omega_i \cdot \hat{n}| \approx \frac{L_i(X, \omega_i) f_X(\omega_i, \omega_o) |\omega_i \cdot \hat{n}| 4\pi \text{sr}}{4\pi}$$

Let’s put it all together for a complete solution

\[
\int_{\mathbf{S}^2} \mathbb{L} \mathbb{m} \mathbb{h} \mathbb{r} \{i\} \{X, \hat{\omega}_i\} \mathbb{f} \mathbb{X} \{\hat{\omega}_i\} \mathbb{c} \mathbb{d} \mathbb{t} \{\hat{n}\} \approx \frac{\mathbb{L} \mathbb{m} \mathbb{h} \mathbb{r} \{i\} \{X, \hat{\omega}_i\} \mathbb{f} \mathbb{X} \{\hat{\omega}_i\} \mathbb{c} \mathbb{d} \mathbb{t} \{\hat{n}\}}{4\pi \text{sr}}
\]

\[
L \mathbb{m} \mathbb{h} \mathbb{r} \{i\} \{X, \hat{\omega}_i\} \mathbb{c} \mathbb{d} \mathbb{t} \{\hat{n}\} = L \mathbb{m} \mathbb{h} \mathbb{r} \{o\} \{Y, -\hat{\omega}_i\}
\]

Let the pdf be uniform on the sphere. How do I choose an incoming vector uniformly at random on the sphere? Easy: choose a uniform point in the $[-1, 1]^3$ cube. Throw it away if it is outside of the sphere and try again. Points in the unit radius ball map equally to the sphere, so if you just normalize the resulting point, it will be a uniformly distributed vector
And you just run a SECOND Monte Carlo integration by running this at each pixel many times and averaging the results. That’s it. No shadow rays, no direct illumination, no biradiance.
This is really the only light transport code that you need to turn any mesh with non-impulse materials...
Like this, into this...
And to kick it off, you just need a main loop that iterates over pixels...
Note that this casts multiple rays at each pixel and averages their results. That’s a Monte Carlo Integrator applied to the pixel values instead of the rendering equation.

There are only two problems with this elegant solution:
Remainding Problems

1. Increase convergence rate
2. Handle impulses in $L_i$ and $f_X$

1. It converges really slowly. You’d need millions of rays per pixel to get glossy surfaces to look good
2. It can’t handle impulses in $L_{\text{in}}$ (Point lights!) or in $f$ (mirrors and refraction!)

We can fix both with an algorithmic optimization called importance sampling...
Choose $p(m) \propto g(m)$

\[
\int_A g(m) \, dm \approx \sum_{j=0}^{N-1} \frac{g(m_j)}{p(m_j)} \frac{1}{N}
\]

- O(N) evaluations of $g$ and $p$
- Small $g(m)$ values are inefficient: integral value per compute time is small
- Ideal: choose $p(m) = g(m) / c$

\[
\approx \sum_{j=0}^{N-1} \frac{c}{N}
\]

“Pure” path tracing doesn’t work for point lights. There’s zero probability that a ray hits a point light, and if it did, the radiance would be infinite. So, we have to put direct illumination, shadow rays, and biradiance back in to make point lights (vs. area lights) work. Sorry. We only use path tracing for indirect light. BUT: The direct illumination code is a lot faster than hoping a random ray will hit the light source, and you already wrote and debugged it anyway.
Monte Carlo integration tells us that it would be optimal to make the denominator probability density function \( p() \) proportional to the integrand in the numerator. But the whole point was that we don’t yet know the value of the numerator. What do we do?

One solution is that assume that light comes into this point \( X \) equally from all directions (it doesn’t matter for correctness if this assumption is wrong, and it is often pretty reasonable). In that case, the hard value to compute: the incoming light, goes away and we are just sampling from a probability distribution that is proportional to the cosine-weighted material.

\[
\approx \frac{L_i(X, \hat{\omega}_1) \ f_X(\hat{\omega}_1, \hat{\omega}_o) \ |\hat{\omega}_1 \cdot \hat{n}|}{p(\hat{\omega}_1)}
\]

How do we choose \( p \) proportional to the integrand we don’t yet know?

“BSDF importance sampling” assumes \( L_i \) is constant and chooses

\[
p(\hat{\omega}_1) \propto f_X(\hat{\omega}_1, \hat{\omega}_o) \ |\hat{\omega}_1 \cdot \hat{n}|
\]
\[ \omega_i, \text{ weight} = \text{scatter}(X, n, \omega_o) \]

Where:

\[
\text{weight} = \frac{f_X(\hat{\omega}_i, \hat{\omega}_o) |\hat{\omega}_i \cdot \hat{n}|}{p(\hat{\omega}_i)}
\]

This also solves the problem of impulses (infinite values with zero range) in \( f() \) [and less importantly, in \( L \)]. We just build a function...or more likely, call into a library routine...that gives us the ratio of \( f^\ast \cos / p() \) after sampling (called a weight), where it will handle the infinities and make them "cancel" by taking a limit as the value approaches the impulse.

This also solves the problem of how we can choose a single ray when we have different values of \( f \) for difference frequencies (colors) of light: the weight is spectral, and varies depending on the ratio of \( f(\lambda) \) to the monochrome \( p(w) \).
Direct Illumination

- Monte Carlo integration of direct illumination
- Compute biradiance $\beta_j$ for a set of patches on lights
  - (assume no shadows)
- Choose light patch $j$ with probability $p = \beta_j / \sum \beta_k$
- Only cast a shadow ray to patch $j$
- Compute direct illumination as $L = \frac{\beta_j |\hat{\omega}_i \cdot \hat{n}|}{p}$

*Tip: implement direct illumination from all lights at first with a loop. This is an optimization.*

This is straight up importance sampling, except that we don’t know the cumulative distribution function for the lights...so we estimate that stochastically as well.

This is called importance resampling and was introduced to graphics by Talbot et al. 2005 [https://dl.acm.org/citation.cfm?id=2383674](https://dl.acm.org/citation.cfm?id=2383674), but the core idea is from stats and goes back to Rubin 1987.
<table>
<thead>
<tr>
<th>Three Separate Monte Carlo Integrators in Basic Path Tracing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. <strong>Pixel Area:</strong></td>
</tr>
<tr>
<td>- large $N$</td>
</tr>
<tr>
<td>- $p = \text{uniform on square}$</td>
</tr>
<tr>
<td>2. <strong>Direct Illumination:</strong></td>
</tr>
<tr>
<td>- $N = 1$</td>
</tr>
<tr>
<td>- $p = \text{relative biradiance}$</td>
</tr>
<tr>
<td>3. <strong>Indirect Illumination:</strong></td>
</tr>
<tr>
<td>- $N = 1$</td>
</tr>
<tr>
<td>- $p \propto f_X(\hat{\omega}_i, \hat{\omega}_o) \mid \hat{\omega}_i \cdot \hat{n}$</td>
</tr>
</tbody>
</table>
Summary

• The Rendering Equation describes steady-state recursive light flow
• Monte Carlo integration: add random samples weighted by the probability of choosing them
• Pure path tracing is amazingly simple: MC integrate the Rendering Equation
• Importance sampling makes path tracing more efficient and elegantly handles impulses [e.g., perfect mirrors]

For full details, see the Graphics Codex 2.17 chapters on Materials, Numerical Calculus, and Path Tracing