Rendering equation and path tracing

http://graphics.cs.cmu.edu/courses/15-468

15-468, 15-668, 15-868
Physics-based Rendering
Spring 2024, Lecture 11
Course announcements

- All quizzes up to TQ4 graded on Canvas!
Overview of today’s lecture

• Rendering equation.

• Path tracing with next-event estimation.
Slide credits

Most of these slides were directly adapted from:

• Wojciech Jarosz (Dartmouth).
Direct vs. Indirect Illumination

Where does $L_i$ "come from"?

$$L_r(x, \omega_r) = \int_{H^2} f_r(x, \omega_i, \omega_r) L_i(x, \omega_i) \cos \theta_i \, d\omega_i$$
Direct vs. Indirect Illumination

Direct illumination

Indirect illumination

Direct + indirect illumination
All-in-One!

Ritschel et al. [2012]
Reflection Equation

Reflected radiance is the weighted integral of incident radiance

\[ L_r(x, \vec{\omega}_r) = \int_{H^2} f_r(x, \vec{\omega}_i, \vec{\omega}_r) L_i(x, \vec{\omega}_i) \cos \theta_i \, d\vec{\omega}_i \]
Energy equilibrium:

\[ L_o(x, \vec{\omega}_o) = L_e(x, \vec{\omega}_o) + L_r(x, \vec{\omega}_o) \]

- outgoing
- emitted
- reflected

Rendering Equation


Energy equilibrium:

\[ L_o(x, \vec{\omega}_o) = L_e(x, \vec{\omega}_o) + \int_{H^2} f_r(x, \vec{\omega}_i, \vec{\omega}_o) L_i(x, \vec{\omega}_o) \cos \theta_i \, d\vec{\omega}_i \]

outgoing  emitted  reflected
Light Transport

In free-space/vacuum, radiance is constant along rays.

We can relate incoming radiance to outgoing radiance:

\[ L_i(x, \omega) = L_0(r(x, \omega), -\omega) \]
Rendering Equation

\[ L(x, \bar{\omega}) = L_e(x, \bar{\omega}) + \int_{H^2} f_r(x, \bar{\omega}', \bar{\omega}) L(r(x, \bar{\omega}'), -\bar{\omega}') \cos \theta' d\bar{\omega}' \]

Only outgoing radiance on both sides

- we drop the “o” subscript
- Fredholm equation of the second kind (recursive)
- Extensive operator-theoretic study (that we will not cover here, but great reading group material)
Rendering Equation

\[ L(x, \omega) = L_e(x, \omega) + \int_{H^2} f_r(x, \omega', \omega) L(r(x, \omega'), -\omega') \cos \theta' \, d\omega' \]
\[ L(x, \omega) = L_e(x, \omega) + \int_{H^2} f_r(x, \omega', \omega') L(r(x, \omega'), -\omega') \cos \theta' d\omega' \]
Rendering Equation

\[ L(x, \bar{\omega}) = L_e(x, \bar{\omega}) + \int_{H^2} f_r(x, \bar{\omega}', \bar{\omega}) L(r(x, \bar{\omega}'), -\bar{\omega}') \cos \theta' \, d\bar{\omega}' \]
Rendering Equation

\[ L(x, \omega) = L_e(x, \omega) + \int_{H^2} f_r(x, \omega', \omega') L(r(x, \omega'), -\omega') \cos \theta' \, d\omega' \]
The rendering equation is given by:

\[ L(x, \bar{\omega}) = L_e(x, \bar{\omega}) + \int_{H^2} f_r(x, \bar{\omega}', \bar{\omega}) L(r(x, \bar{\omega}'), -\bar{\omega}') \cos \theta' d\bar{\omega}' \]
Rendering Equation

\[ L(x, \omega) = L_e(x, \omega) + \int_{H^2} f_r(x, \omega', \tilde{\omega}) L(r(x, \omega'), -\omega') \cos \theta' \, d\omega' \]
\[ L(x, \vec{\omega}) = L_e(x, \vec{\omega}) + \int_{H^2} f_r(x, \vec{\omega}', \vec{\omega}) L(r(x, \vec{\omega}'), -\vec{\omega}') \cos \theta' \, d\vec{\omega}' \]
Rendering Equation

\[ L(\mathbf{x}, \bar{\omega}) = L_e(\mathbf{x}, \bar{\omega}) + \int_{H^2} f_r(\mathbf{x}, \bar{\omega'}, \bar{\omega}) L(r(\mathbf{x}, \bar{\omega'}), -\bar{\omega'}) \cos \theta' d\bar{\omega'} \]

Recursion

Light source

\[ L(r(\mathbf{x}, \bar{\omega'}), -\bar{\omega'}) \]
Rendering Equation

\[ L(x, \omega) = L_e(x, \omega) + \int_{H^2} f_r(x, \omega', \omega) L(r(x, \omega'), -\omega') \cos \theta' \, d\omega' \]
Rendering Equation

\[ L(x, \tilde{\omega}) = L_e(x, \tilde{\omega}) + \int_{H^2} f_r(x, \tilde{\omega}', \tilde{\omega}) L(r(x, \tilde{\omega}'), -\tilde{\omega}') \cos \theta' \, d\tilde{\omega}' \]
Rendering Equation

\[ L(x, \omega) = L_e(x, \omega) + \int_{H^2} f_r(x, \omega', \omega)L(r(x, \omega'), -\omega') \cos \theta' \, d\omega' \]
Rendering Equation

\[ L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos \theta' \, d\vec{\omega}' \]
Rendering Equation

\[
L(x, \bar{\omega}) = L_e(x, \bar{\omega}) + \int_{H^2} f_r(x, \bar{\omega}', \bar{\omega}) L(r(x, \bar{\omega}'), -\bar{\omega}') \cos \theta' \, d\bar{\omega}'
\]
Rendering Equation

\[ L(x, \bar{\omega}) = L_e(x, \bar{\omega}) + \int_{H^2} f_r(x, \bar{\omega}', \bar{\omega}) L(r(x, \bar{\omega}'), -\bar{\omega}') \cos \theta' \, d\bar{\omega}' \]

light source

\[ L(r(x, \bar{\omega}'), -\bar{\omega}') \]

\[ L(x, \bar{\omega}) \]

\[ x \]
Rendering Equation

\[ L(x, \bar{\omega}) = L_e(x, \bar{\omega}) + \int_{H^2} f_r(x, \bar{\omega}', \bar{\omega}) L(r(x, \bar{\omega}'), -\bar{\omega}') \cos \theta' d\bar{\omega}' \]
Rendering Equation

\[ L(x, \bar{\omega}) = L_e(x, \bar{\omega}) + \int_{H^2} f_r(x, \bar{\omega}', \bar{\omega}) L(r(x, \bar{\omega}'), -\bar{\omega}') \cos \theta' \, d\bar{\omega}' \]
Rendering Equation

\[ L(x, \bar{\omega}) = L_e(x, \bar{\omega}) + \int_{H^2} f_r(x, \bar{\omega}', \bar{\omega}) L(r(x, \bar{\omega}'), -\bar{\omega}') \cos \theta' d\bar{\omega}' \]
Path Tracing
Path Tracing

\[ L(x, \omega) = L_e(x, \omega) + \int_{H^2} f_r(\bar{x}, \omega', \omega) L(r(x, \omega'), -\omega') \cos \theta' d\omega' \]

\[ L(x, \omega) \approx L_e(x, \omega) + \frac{f_r(\bar{x}, \omega', \omega) L(r(x, \omega'), -\omega') \cos \theta'}{p(\omega')} \]
Path Tracing Algorithm

\[ L(x, \omega) = L_e(x, \omega) + L_T(x, \omega) \]

Color \(\text{color}(\text{Point } x, \text{ Direction } \omega, \text{ int } \text{moreBounces})\):

\[
\text{if not moreBounces:} \\
\quad \text{return } L_e(x,-\omega)
\]

// sample recursive integral
\(\omega' = \text{sample from BRDF}\)
\[
\text{return } L_e(x,-\omega) + \text{BRDF} \times \text{color}(\text{trace}(x, \omega'), \text{moreBounces}-1) \times \text{dot}(n, \omega') / \text{pdf}(\omega')
\]
Path Tracing with Shadow Rays

1 path/pixel
Path Tracing

\[ L(x, \tilde{\omega}) = L_e(x, \tilde{\omega}) + \int_{H^2} f_r(\tilde{x}, \tilde{\omega}', \tilde{\omega}) L(r(x, \tilde{\omega}'), -\tilde{\omega}') \cos \theta' \, d\tilde{\omega}' \]

\[ L(x, \tilde{\omega}) \approx L_e(x, \tilde{\omega}) + \frac{f_r(\tilde{x}, \tilde{\omega}', \tilde{\omega}) L(r(x, \tilde{\omega}'), -\tilde{\omega}') \cos \theta'}{p(\tilde{\omega}')} \]
Improving quality: the wrong way

\[ L(x, \tilde{\omega}) = L_e(x, \tilde{\omega}) + \int_{H^2} f_r(x, \tilde{\omega}', \tilde{\omega}) L(r(x, \tilde{\omega}'), -\tilde{\omega}') \cos \theta' \, d\tilde{\omega}' \]

\[ L(x, \tilde{\omega}) \approx L_e(x, \tilde{\omega}) + \frac{1}{N} \sum_{k=1}^{N} \frac{f_r(x, \tilde{\omega}_k', \tilde{\omega}) L(r(x, \tilde{\omega}_k'), -\tilde{\omega}_k') \cos \theta_k'}{p(\tilde{\omega}_k')} \]
The problem

Exponential growth!

3-bounce contributes less than 1-bounce transport, but we estimate it with 25× as many samples!
Improving quality

Just shoot more rays/pixel

- avoid exponential growth: make sure not to branch!

Each ray will start a new **path**

We can achieve antialiasing/depth of field/motion blur at the same time “for free”!
Path Tracing with Shadow Rays

1 path/pixel
Path Tracing with Shadow Rays

4 paths/pixel
Path Tracing with Shadow Rays

16 paths/pixel
Path Tracing with Shadow Rays

64 paths/pixel
Path Tracing with Shadow Rays

256 paths/pixel
Path Tracing with Shadow Rays

1024 paths/pixel
When do we stop recursion?

Truncating at some fixed depth introduces bias

Solution: Russian roulette
Russian Roulette

Probabilistically terminate the recursion

New estimator: evaluate original estimator $X$ with probability $P$ (but reweighted), otherwise return zero:

$$X_{rr} = \begin{cases} \frac{X}{P} & \xi < P \\ 0 & \text{otherwise} \end{cases}$$

**Unbiased**: same expected value as original estimator:

$$E[X_{rr}] = P \cdot \left( \frac{E[X]}{P} \right) + (1 - P) \cdot 0 = E[X]$$
Russian Roulette

This will actually increase variance!

- but it will improve efficiency if $P$ is chosen so that samples that are expensive, but are likely to make a small contribution, are skipped

You are already doing this

- probabilistic absorption in BSDF (instead of scattering)
Partitioning the Integrand

Direct illumination: sometimes better estimated by sampling emissive surfaces

Let’s estimate direct illumination separately from indirect illumination, then add the two
- i.e., shoot shadow rays (direct) and gather rays (indirect)
- be careful *not to double-count*!

Also known as *next-event estimation (NEE)*
Path Tracing with NEE

\[ L(x, \tilde{\omega}) = L_e + \int_{A_e} \cdots L_e(x \leftrightarrow x') \cdots dA_e(x') + \int_{H^2 \setminus A_e} \cdots L(x, \tilde{\omega}') \cdots d\tilde{\omega}' \]
Path Tracing Algorithm with NEE

\[ L(x, \omega) = L_e(x, \omega) + L_{\text{dir}}(x, \omega) + L_{\text{ind}}(x, \omega) \]

Color `color(Point x, Direction \omega, int moreBounces)`: 

```plaintext
if not moreBounces:
    return Le;
```

// next-event estimation: compute \( L_{\text{dir}} \) by sampling the light
\( \omega_1 = \text{sample from light} \)
\( L_{\text{dir}} = \text{BRDF} \times \text{color}(\text{trace}(x, \omega_1), 0) \times \text{dot}(n, \omega_1) / \text{pdf}(\omega_1) \)

// compute \( L_{\text{ind}} \) by sampling the BSDF
\( \omega_2 = \text{sample from BSDF} \);
\( L_{\text{ind}} = \text{BSDF} \times \text{color}(\text{trace}(x, \omega_2), \text{moreBounces}-1) \times \text{dot}(n, \omega_2) / \text{pdf}(\omega_2) \)

return \( L_e + L_{\text{dir}} + L_{\text{ind}} \)
Path Tracing Algorithm with NEE

\[ L(x, \tilde{\omega}) = L_e(x, \tilde{\omega}) + L_{\text{dir}}(x, \tilde{\omega}) + L_{\text{ind}}(x, \tilde{\omega}) \]

Color `color(Point x, Direction \omega, int moreBounces, bool includeLe)`: 

\[ L_e = \begin{cases} \text{includeLe} \, ? \, L_e(x, -\omega) & : \text{black} \\ \text{if not moreBounces:} \\ \quad \text{return } L_e \end{cases} \]

// next-event estimation: compute \( L_{\text{dir}} \) by sampling the light
\( \omega_1 = \text{sample from light} \)
\[ L_{\text{dir}} = \text{BRDF} \times \text{color(trace(x, } \omega_1, \text{ 0, true) } \times \text{dot(n, } \omega_1) / \text{pdf(} \omega_1) \]

// compute \( L_{\text{ind}} \) by sampling the BSDF
\( \omega_2 = \text{sample from BSDF} \)
\[ L_{\text{ind}} = \text{BSDF} \times \text{color(trace(x, } \omega_2, \text{ moreBounces-1, false) } \times \text{dot(n, } \omega_2) / \text{pdf(} \omega_2) \]

return \( L_e + L_{\text{dir}} + L_{\text{ind}} \)
Questions?

We should really be using MIS or mixture sampling
Naive Path Tracing

\[ L(x, \omega) = L_e(x, \omega) + L_r(x, \omega) \]

Color color(Point x, Direction ω, int moreBounces):

    if not moreBounces:
        return Le(x,-ω)

    // sample recursive integral
    ω’ = sample from BRDF
    return Le(x,-ω) + BRDF * color(trace(x, ω’), moreBounces-1) * dot(n, ω’) / pdf(ω’)

52
Path Tracing with mixture sampling

\[ L(x, \omega) = L_e(x, \omega) + L_T(x, \omega) \]

Color color(Point x, Direction \( \omega \), int moreBounces):

if not moreBounces:
    return Le(x,-\omega)

// sample recursive integral
\( \omega' = \) sample from mixture PDF
return \( L_e(x,-\omega) + \) BRDF * color(trace(x, \( \omega' \)), moreBounces-1) * dot(n, \( \omega' \)) / pdf(\( \omega' \))
Path Tracing Algorithm with NEE

color trace(Point x, Direction ω, int moreBounces, bool includeLe):

get scene intersection x, and normal n

Le = includeLe ? Le(x, -ω) : black

if not moreBounces:
    return Le

// next-event estimation: compute Ldir by sampling the light
ω1 = sample from light
Ldir = BRDF * trace(x, ω1, θ, true) * dot(n, ω1) / pdf(ω1)

// compute Lin by sampling the BSDF
ω2 = sample from BSDF
Lind = BSDF * trace(x, ω2, moreBounces-1, false) * dot(n, ω2) / pdf(ω2)

return Le + Ldir + Lind
Path Tracing Algorithm with NEE+MIS

color trace(Point x, Direction \( \omega \), int moreBounces, float \( L_{eweight} \)):

get scene intersection \( x \), and normal \( n \)

\[ L_{e} = L_{eweight} \times L_{e}(x,-\omega) \]

if not moreBounces:
    return \( L_{e} \)

// next-event estimation: compute \( L_{dir} \) by sampling the light
\( \omega_{1} = \) sample from light
\[ L_{dir} = BRDF \times trace(x, \omega_{1}, 0, mis-weight_{1}) \times \frac{\text{dot}(n, \omega_{1})}{pdf(\omega_{1})} \]

// compute \( L_{ind} \) by sampling the BSDF
\( \omega_{2} = \) sample from BSDF
\[ L_{ind} = BSDF \times trace(x, \omega_{2}, moreBounces-1, mis-weight_{2}) \times \frac{\text{dot}(n, \omega_{2})}{pdf(\omega_{2})} \]

return \( L_{e} + L_{dir} + L_{ind} \)
Path Tracing on 99 Lines of C++

```cpp
#include <math.h> // Make p+23 -8*epenr smallpt.cpp or smallpt

#include <stdio.h> // Remove =fopen() for g++ version <= 4.2

struct Vec { // position, also color (r,g,b)
  double x, y, z;
  Vec(x, y, z)
};

struct Sphere { // material types, used in radiance()
  double rad,
  // radius
  Vec o, e, c,
  // position, also color (r,g,b)
  Refl_t refl;
  // material types, used in radiance()
  struct {
    double x, y, z;
  } d;
  // diffuse
  struct {
    double x, y, z;
  } s;
  // specular
  struct {
    double x, y, z;
  } t;
  // refractive
  struct {
    double x, y, z;
  } r;
  // in camera
  Vec p,
  // from camera
  double eps=1e-4,
  // max rad
  b=op.
};

int main(int argc, char *argv[]) {
  FILE *f = fopen("image.ppm", "w"); // Image writer
  // Write image to PPM file.
  return 0;
}
```

smallpt by Kevin Beason
Directions for making pictures using numbers
(explained using only the ten hundred words people use most often)

\[ L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega} f_r(x, \omega_i \rightarrow \omega_o) L_i(x, \omega_i) (\omega_i \cdot n) \, d\omega_i \]

- Light that leaves the position on the stuff and reaches the eye
- Light made by the stuff (sometimes because it is very hot)
- Direction towards the eye
- Position on the stuff
- Light can be added
- Said a man who sat under a tree many years ago
- The answer to how much light from an interesting direction that will keep going in the direction towards the eye, after hitting stuff at the position (this is easy for mirrors, not so easy for everything else)
- How much the light becomes less bright because the stuff leans away from the interesting direction
- For lots of interesting directions inside half a ball facing up from the stuff, add up all the answers in between

This idea came from http://xkcd.com/1133/

@levork