Lecture 14:
Global Illumination

Computer Graphics
CMU 15-462/15-662, Fall 2015

Slides credit: a majority of these slides were created by Matt Pharr and Pat Hanrahan
The reflection equation

$\mathbf{L}_o(p, \omega_o) = \mathbf{L}_e(p, \omega_o) + \int_{H^2} f_r(p, \omega_i \rightarrow \omega_o) \mathbf{L}_i(p, \omega_i) \cos \theta_i \, d\omega_i$

Need to know incident radiance.

So far, have only computed incoming radiance from scene light sources.
**Review: solving the reflection equation**

\[ L_r(p, \omega_r) = \int_{H^2} f_r(p, \omega_i \rightarrow \omega_r) L_i(p, \omega_i) \cos \theta_i \, d\omega_i \]

- **Basic Monte Carlo estimate:**
  - Generate directions $\omega_j$ sampled from some distribution $p(\omega)$
  - Compute estimator

\[
\frac{1}{N} \sum_{j=1}^{N} f_r(p, \omega_j \rightarrow \omega_r) \frac{L_i(p, \omega_j) \cos \theta_j}{p(\omega_j)}
\]
Optimizing the direct lighting estimate
“Splitting”

- Consider estimating an integrand of the form:

\[ \int_A \int_B f(a, b) \, da \, db \]

- Standard Monte Carlo estimator: draw \( N \) samples from a distribution \( p \), then compute:

\[ \frac{1}{N} \sum_{i=1}^{N} \frac{f(a_i, b_i)}{p(a_i, b_i)} \]

- Alternative: take multiple samples of \( b \) for each sample of \( a \). New estimator:

\[ \frac{1}{N_a} \frac{1}{N_b} \sum_{i=1}^{N_a} \sum_{j=1}^{N_b} \frac{f(a_i, b_{i,j})}{p(a_i, b_{i,j})} \]
“Splitting”

\[ \frac{1}{N_a} \frac{1}{N_b} \sum_{i=1}^{N_a} \sum_{j=1}^{N_b} \frac{f(a_i, b_{i,j})}{p(a_i, b_{i,j})} \]

- **Motivation:** can improve efficiency if the evaluation of
  
  \[ f(a_i, b_{i,1}), f(a_i, b_{i,2}), \ldots \]
  
  can share some computation between them

- **Common example:** when estimating direct lighting, use multiple samples to make reflection equation estimate for each camera ray.
const int samples_pixel = 4;           // samples take per pixel (camera rays)
const int samples_refl_estimate = 4;   // samples taken per refl estimate

Spectrum pixel_response(int x, int y) {     // integrate over pixel area
    Spectrum result = 0;
    for (int i=0; i<samples_pixel; i++) {
        Ray camera_ray = compute_random_camera_ray(x, y);  // shared across iterations of
        Intersection isect;                                // inner loop

        if (trace(camera_ray, &isect)) {
            Spectrum Lr = 0;
            for (int j=0; j<samples_refl_estimate; j++) {  // integrate over directions
                Vector3D wi;
                float pdf;
                generate_dir_sample(&isect.brdf, &wi, &pdf);    // random direction
                Spectrum f = isect.brdf.f(-camera_ray.d, wi);
                Spectrum Li = trace_ray(Ray(isect.hit_p, wi));  // compute incoming Li
                Lr += f * Li * fabs(dot(wi, isect.hit_n)) / pdf;
            }
            result += Lr / samples_refl_estimate;
        }
    }
    return result / samples_pixel;
}
64 image samples x 1 light sample, 13.2 seconds
1 image sample x 64 light samples, 7.0 seconds
8 image samples x 8 light samples, 7.7 seconds
Hemispherical solid angle sampling, 100 sample rays
(random directions drawn uniformly from hemisphere)
Single area light source

```cpp
class Spectrum {
public:
  virtual float operator[](const int& i) const { return 0; }
};

Spectrum pixel_response(int x, int y) {
  Spectrum result = 0;
  for (int i=0; i<samples_pixel; i++) {
    Ray camera_ray = compute_random_camera_ray(x, y);  // shared across iterations of inner loop
    Intersection isect;
    if (trace(camera_ray, &isect)) {
      Spectrum Lr = 0;
      for (int j=0; j<samples_refl_estimate; j++) {  // integrate over directions
        Vector3D wi;
        float pdf;
        light.generate_dir_sample(&wi, &pdf);    // dir to random point on light
        Spectrum f = isect.brdf.f(-camera_ray.d, wi);
        Spectrum Li = trace_ray(Ray(isect.hit_p, wi));  // compute incoming Li
        Lr += f * Li * fabs(dot(wi, isect.hit_n)) / pdf;
      }
      result += Lr / samples_refl_estimate;
    }
  }
  return result / samples_pixel;
}
```
Review: sampling directions vs. sampling light source area

Light source area sampling, 100 sample rays
How does the pseudocode change if we have multiple light sources?

$$L_r(p, \omega_r) = \int_{H^2} f_r(p, \omega_i \rightarrow \omega_r) L_i(p, \omega_i) \cos \theta_i \ d\omega_i$$
const int samples_pixel = 4; // samples take per pixel (camera rays)
const int samples_refl_estimate = 4; // samples taken per refl estimate

Spectrum pixel_response(int x, int y) { // integrate over pixel area
    Spectrum result = 0;
    for (int i=0; i<samples_pixel; i++) {
        Ray camera_ray = compute_random_camera_ray(x, y); // shared across iterations of
        Intersection isect; // inner loop
        if (trace(camera_ray, &isect)) {
            Spectrum Lr = 0;
            for (int j=0; j<num_lights; j++) {
                for (int k=0; k<samples_refl_estimate; k++) { // integrate over directions
                    Vector3D wi;
                    float pdf;
                    light[j].generate_dir_sample(&wi, &pdf); // dir to random point on light
                    Spectrum f = isect.brdf.f(-camera_ray.d, wi);
                    Spectrum Li = trace_ray(Ray(isect.hit_p, wi)); // compute incoming Li
                    Lr += f * Li * fabs(dot(wi, isect.hit_n)) / pdf;
                }
                result += Lr / samples_refl_estimate;
            }
            return result / samples_pixel;
        }
    }
}
const int samples_pixel = 4; // samples take per pixel (camera rays)
const int samples_refl_estimate = 4; // samples taken per refl estimate

Spectrum pixel_response(int x, int y) { // integrate over pixel area
    Spectrum result = 0;
    for (int i=0; i<samples_pixel; i++) {
        Ray camera_ray = compute_random_camera_ray(x, y); // shared across iterations of
        Ray intersection isect; // inner loop
        if (trace(camera_ray, &isect)) {
            Spectrum Lr = 0;
            for (int j=0; j<samples_refl_estimate; j++) { // integrate over directions
                Vector3D wi;
                float light_pdf, pdf;
                int light_index = randomly_pick_light(&light_pdf);
                light[light_idx].generate_dir_sample(&wi, &pdf); // dir to random point
                Spectrum f = isect.brdf.f(-camera_ray.d, wi);
                Spectrum Li = trace_ray(Ray(isect.hit_p, wi)); // compute incoming Li
                Lr += f * Li * fabs(dot(wi, isect.hit_n)) / (light_pdf * pdf);
            }
            result += num_lights * Lr / samples_refl_estimate;
        }
    }
    return result / samples_pixel;
}
Russian roulette

- Recall definition of Monte Carlo efficiency:

\[
\text{Efficiency} \propto \frac{1}{\text{Variance} \times \text{Cost}}
\]

- Idea: want to avoid spending time evaluating function for samples that make a **small contribution** to the final result

- Consider a low-contribution sample of the form:

\[
L = \frac{f_r(\omega_i \rightarrow \omega_o) \cdot L_i(\omega_i) \cdot V(p, p') \cdot \cos \theta_i}{p(\omega_i)}
\]
Russian roulette

\[ L = \left( \frac{f_r(\omega_i \rightarrow \omega_o) L_i(\omega_i) V(p, p')} {p(\omega_i)} \right) \cos \theta_i \]

\[ V(p, p') \]

- If tentative contribution (in brackets) is small, total contribution to the image will be small regardless of \( V(p, p') \)
- Ignoring low-contribution samples introduces systemic error
  - No longer an unbiased estimator
- Instead, randomly discard low-contribution samples in a way that leaves estimator unbiased
Russian roulette

- New estimator: evaluate original estimator with probability $p_{rr}$, reweight. Otherwise ignore.
- Same expected value as original estimator:

$$p_{rr} E \left[ \frac{X}{p_{rr}} \right] + E[(1 - p_{rr})0] = E[X]$$

- Want to choose $p_{rr}$ to maximize Monte Carlo efficiency...
No Russian roulette: 6.4 seconds
Russian roulette: terminate 50% of all contributions with luminance less than 0.25: 5.1 seconds
Russian roulette: terminate 50% of all contributions with luminance less than 0.5: 4.9 seconds
Russian roulette: terminate 90% of all contributions with luminance less than 0.125: 4.8 seconds
Russian roulette: terminate 90% of all contributions with luminance less than 1: 3.6 seconds
Estimating indirect lighting
The reflection equation

\[ L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{H^2} f_r(p, \omega_i \rightarrow \omega_o) L_i(p, \omega_i) \cos \theta_i \, d\omega_i \]

Need to know incident radiance.
So far, have only computed incoming radiance from scene light sources.
Accounting for indirect illumination

\[ L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{H^2} f_r(p, \omega_i \rightarrow \omega_o) L_i(p, \omega_i) \cos \theta_i \, d\omega_i \]

Incoming light energy from direction \( \omega_i \) may be due to light reflected off another surface in the scene (not an emitter)
Direct illumination
One-bounce global illumination
Two-bounce global illumination
Four-bounce global illumination
Eight-bounce global illumination
Sixteen-bounce global illumination
Direct illumination + reflection + transparency

Image credit: Henrik Wann Jensen
Global illumination solution

Image credit: Henrik Wann Jensen
Energy balance

- **Accountability**
  - \([\text{outgoing}] - [\text{incoming}] = [\text{emitted}] - [\text{absorbed}]\)

- **Macro level:**
  - The total light energy put into the system must equal the energy leaving the system (usually, via heat)
    \[ \Phi_o - \Phi_i = \Phi_e - \Phi_a \]

- **Micro level:**
  - The energy flowing into a small region must equal the energy flowing out:
    \[ E_o(p) - E_i(p) = E_e(p) - E_a(p) \]
Surface balance equation

\[ \text{[outgoing]} = \text{[emitted]} + \text{[reflected]} \]

\[ L_o(p, \omega_o) = L_e(p, \omega_o) + L_r(p, \omega_o) \]

\[ = L_e(p, \omega_o) + \int_{H^2} f_r(p, \omega_i \rightarrow \omega_o) L_i(p, \omega_i) \cos \theta_i \, d\omega_i \]
The reflection equation

\[ L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{H^2} f_r(p, \omega_i \rightarrow \omega_o) L_i(p, \omega_i) \cos \theta_i \, d\omega_i \]

Let’s rewrite using transport function: \( tr(p, \omega) \)
Returns first intersection point on surface in the scene along ray defined by \((p, \omega)\)
The rendering equation: directional form

Write incident radiance in terms of exitant radiance at first visible surface:

\[ L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{H^2} f_r(p, \omega_i \rightarrow \omega_o) L_o(tr(p, \omega_i), -\omega_i) \cos \theta_i \, d\omega_i \]

\[
\begin{align*}
\text{Light scattering} & \quad \uparrow \\
\text{Light transport} & \quad \downarrow
\end{align*}
\]

Radiance invariance along rays:

\[ L_i(p, \omega_i) = L_o(tr(p, \omega_i), -\omega_i) \]

“Radiance arriving at \( p \) from direction \( \omega_i \) is same as radiance leaving \( p' \) towards \( p \).”

CMU 15-462/662, Fall 2015
Path tracing: overview

- Partition the rendering equation into direct and indirect illumination
- Use Monte Carlo to estimate each partition separately
  - One sample for each—no splitting!
  - Assumption: 100s of samples per pixel
- Terminate paths with Russian roulette
Partitioning the rendering equation

\[ L_o(p, \omega_o) = L_e(p, \omega_o) + \]
\[ \int_{H^2} f_r(\omega_i \rightarrow \omega_o) L_{o,d}(tr(p, \omega_i), -\omega_i) \cos \theta_i \, d\omega_i + \]
\[ \int_{H^2} f_r(\omega_i \rightarrow \omega_o) L_{o,i}(tr(p, \omega_i), -\omega_i) \cos \theta_i \, d\omega_i \]

- **Incident “direct” illumination:** \( L_{o,d}(tr(p, \omega_i), -\omega_i) \)
  - Handle with traditional direct lighting techniques
  - But: only a single shadow ray!
- **Incident “indirect” illumination:** \( L_{o,i}(tr(p, \omega_i), -\omega_i) \)
  - Handle with recursive evaluation of the rendering equation
Path tracing: indirect illumination

\[
\int_{H^2} f_r(\omega_i \rightarrow \omega_o) L_{o,i}(tr(p, \omega_i), -\omega_i) \cos \theta_i \, d\omega_i
\]

- Sample incoming direction from some distribution (e.g. proportional to BRDF):
  \[
  \omega_i \sim p(\omega)
  \]
- Recursively call path tracing function to compute incident indirect radiance
- Monte Carlo estimator:
  \[
  \frac{f_r(\omega_i \rightarrow \omega_o) L_{o,i}(tr(p, \omega_i), -\omega_i) \cos \theta_i}{p(\omega_i)}
  \]
The rendering equation: area form

- Can equivalently write rendering equation as an integral over surface area of objects in the scene
  - Apply change of variables \( d\omega = \frac{\cos \theta}{r^2} dA \)
  - Introduce binary visibility function: \( V(p \leftrightarrow p') \)

\[
L_o(p, \omega_o) = L_e(p, \omega_o) + \int_A f_r(p, (p' - p) \rightarrow \omega_o) L_o(p', (p - p')) \cos \theta_i V(p \leftrightarrow p') \cos \theta' \frac{\cos \theta'}{|p - p'|^2} dp'
\]

\[
= L_e(p, \omega_o) + \int_A f_r(p, (p' - p) \rightarrow \omega_o) L_o(p', (p - p')) G(p \leftrightarrow p') dp'
\]

\[
G(p \leftrightarrow p') = V(p \leftrightarrow p') \frac{\cos \theta \cos \theta'}{|p - p'|^2}
\]
The rendering equation: sum over paths

\[ L_o(p_1 \rightarrow p) = L_e(p_1 \rightarrow p) \]

\[ + \int_{A} L_e(p_2 \rightarrow p_1) f(p_2 \rightarrow p_1 \rightarrow p) G(p_1 \leftrightarrow p_2) dA(p_2) \]

\[ + \int_{A} \int_{A} L_e(p_3 \rightarrow p_2) f(p_3 \rightarrow p_2 \rightarrow p_1) G(p_2 \leftrightarrow p_3) f(p_2 \rightarrow p_1 \rightarrow p) G(p_1 \leftrightarrow p_2) dA(p_3) dA(p_2) \]

\[ + \cdots \]

\[ L_o(p_1 \rightarrow p) = \sum_{n=1}^{\infty} P(\overline{p_n}) \]

Path of length 1

Paths of length 2

Energy reaching \( p \) from all paths of length \( n \)

Path of length \( n \) (\( n+1 \) vertices)

Path of length \( n+1 \) vertices
The rendering equation: sum over paths

\[ L_o(\bar{p}_n) = \int_A \int_A \int_A \cdots \int_A L_e(p_n \rightarrow p_{n-1}) \]

\[ \times \left( \prod_{i=1}^{n-1} f(p_{i+1} \rightarrow p_i \rightarrow p_{i-1}) G(p_{i+1} \leftrightarrow p_i) \right) \]

\[ = \int_A \int_A \int_A \cdots \int_A L_e(p_n \rightarrow p_{n-1}) T(\bar{p}_n) dA(p_2) \cdots dA(p_n) \]

Path “throughput” = fraction of light from light source at point \( p_n \) reaching \( p \)
The rendering equation: sum over paths

\[ L_o(p_1 \rightarrow p) = \frac{1}{1 - q_1} (P(\bar{p}_1) + \frac{1}{1 - q_2} (P(\bar{p}_2) + \frac{1}{1 - q_3} (P(\bar{p}_3))... \]

\[ q_i = \text{probability of terminating before paths of length 1} \]
**Path tracing pseudocode**

\[
L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{H^2} f_r(\omega_i \rightarrow \omega_o) L_{o,d}(tr(p, \omega_i), -\omega_i) \cos \theta_i \, d\omega_i + \int_{H^2} f_r(\omega_i \rightarrow \omega_o) L_{o,i}(tr(p, \omega_i), -\omega_i) \cos \theta_i \, d\omega_i
\]

```cpp
Spectrum pathtrace(Ray ray) {
    Intersection isect = scene.intersect(ray);
    Vector2D wo = -ray.d;
    Spectrum Lo = isect.Le(wo); // surface emission in direction wo
    Lo += estimate_direct_lighting(isect, wo); // (see code on earlier slide, but do not
    // include Le in estimate)
    Vector2D wi;
    float pdf;
    generate_direction_sample(isect.brdf, wo, &wi, &pdf); // random direction to sample indirect
    Spectrum f = isect.brdf.f(wo, wi);
    float terminateProbability = 1.f - f.rho(); // termination probability based on
    // reflectance (averaged over spectrum). Lower
    // reflectance = high chance of terminating
    if (RandomFloat() < terminateProbability)
        return Lo;
    return Lo + ((f * pathtrace(Ray(isect.P, wi)) * Dot(wi, isect.N) / (pdf * (1-terminateProbability))));
}
```
Path tracing pseudocode

\[
L_o(p, \omega_o) \approx L_e(p, \omega_o) + \frac{f_r(\omega_i \rightarrow \omega_o) L_{o,d}(tr(p, \omega_i), -\omega_i) \cos \theta_i}{p(\omega_i)} + \frac{f_r(\omega'_i \rightarrow \omega_o) L_{o,i}(tr(p, \omega'_i), -\omega'_i) \cos \theta'_i}{p(\omega'_i)}
\]

Spectrum pathtrace(Ray ray) {
    Intersection isect = scene.intersect(ray);
    Vector2D wo = -ray.d;
    Spectrum Lo = isect.Le(wo); // surface emission in direction wo
    Lo += estimate_direct_lighting(isect, wo); // (see code on earlier slide, but do not
    // include Le in estimate)
    Vector2D wi;
    float pdf;
    generate_direction_sample(isect.brdf, wo, &wi, &pdf); // random direction to sample indirect
    Spectrum f = isect.brdf.f(wo, wi);
    float terminateProbability = 1.f - f.rho(); // termination probability based on
    // reflectance (averaged over spectrum). Lower
    // reflectance = high chance of terminating
    if (RandomFloat() < terminateProbability)
        return Lo;
    return Lo + ((f * pathtrace(Ray(isect.P, wi)) * Dot(wi, isect.N) / (pdf * (1-terminateProbability)))};
One sample per pixel
32 samples per pixel
1024 samples per pixel
Linear operators

- Linear operators act on functions like matrices act on vectors

\[ h(x) = (L \circ f)(x) \]

- They are linear in that:

\[ (L \circ af + bg) = a(L \circ f) + b(L \circ g) \]

- Types of linear operators:

\[ (K \circ f) = \int k(x, x') f(x') \, dx' \]

\[ (D \circ f) = \frac{\partial f}{\partial x} \]
Light transport operators

\[ L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{H^2} f_r(p, \omega_i \rightarrow \omega_o) L_o(tr(p, \omega_i), -\omega_i) \cos \theta_i \, d\omega_i \]

**Full light transport operator:** \( K = R \circ T \)

**Reflection operator:** \( R \circ g = \int_{H^2} f_r(\omega_i \rightarrow \omega_o) g(\omega_i) \cos \theta_i \, d\omega_i \)

**Light propagation operator:** \( T \circ f = f(tr(p, \omega), -\omega) \)

\[ L_i = T \circ L_o \]
Solving the rendering equation

- Rendering equation:

\[ L = L_e + K \circ L \]

\[ (I - K) \circ L = L_e \]

- Solution:

\[ L = (I - K)^{-1} \circ L_e \]

- Neumann series:

\[ (I - K)^{-1} = \frac{1}{I - K} = I + K + K^2 + K^3 + \cdots \]
Successive approximations

\[ (I - K)^{-1} = \frac{1}{I - K} = I + K + K^2 + K^3 + \cdots \]

Energy conservation makes it possible to show that a solution exists

\[ L^1 = L_e \]
\[ L^2 = L_e + K \circ L^1 \]
\[ \vdots \]
\[ L^n = L_e + K \circ L^{n-1} \]
$L_e$
$K \odot K \odot K \odot K \odot K \odot K \odot L_e$
\[
\sum_{i=1}^{2} K^i \circ L_e
\]
\[ \sum_{i=1}^{3} K^i \circ L_e \]
\[ \sum_{i=1}^{4} K^i \circ L_e \]
\[ \sum_{i=1}^{5} K_i \circ L_e \]
\[ \sum_{i=1}^{7} K_i \circ L_e \]
Sum over paths: implications

- Don’t need to only follow paths from the camera
  - Gives grounding for other (more efficient) algorithms for evaluating the rendering equation: ray tracing from lights, bidirectional path tracing, etc...

- Don’t even need to sample path vertices sequentially!
  - See bi-directional path tracing

- Sample path vertices from arbitrary area distributions
Summary: rendering equation

- **Indirect illumination:**
  - Illumination of point is due to both light directly emitted from sources and from light reflected from other surfaces
  - Critical to realistic image synthesis

- **Rendering equation:** describes reflection and transport of light in a scene
  - In practice, solved using Monte Carlo techniques
  - Today: discussed path tracing solution, but much more efficient techniques exist