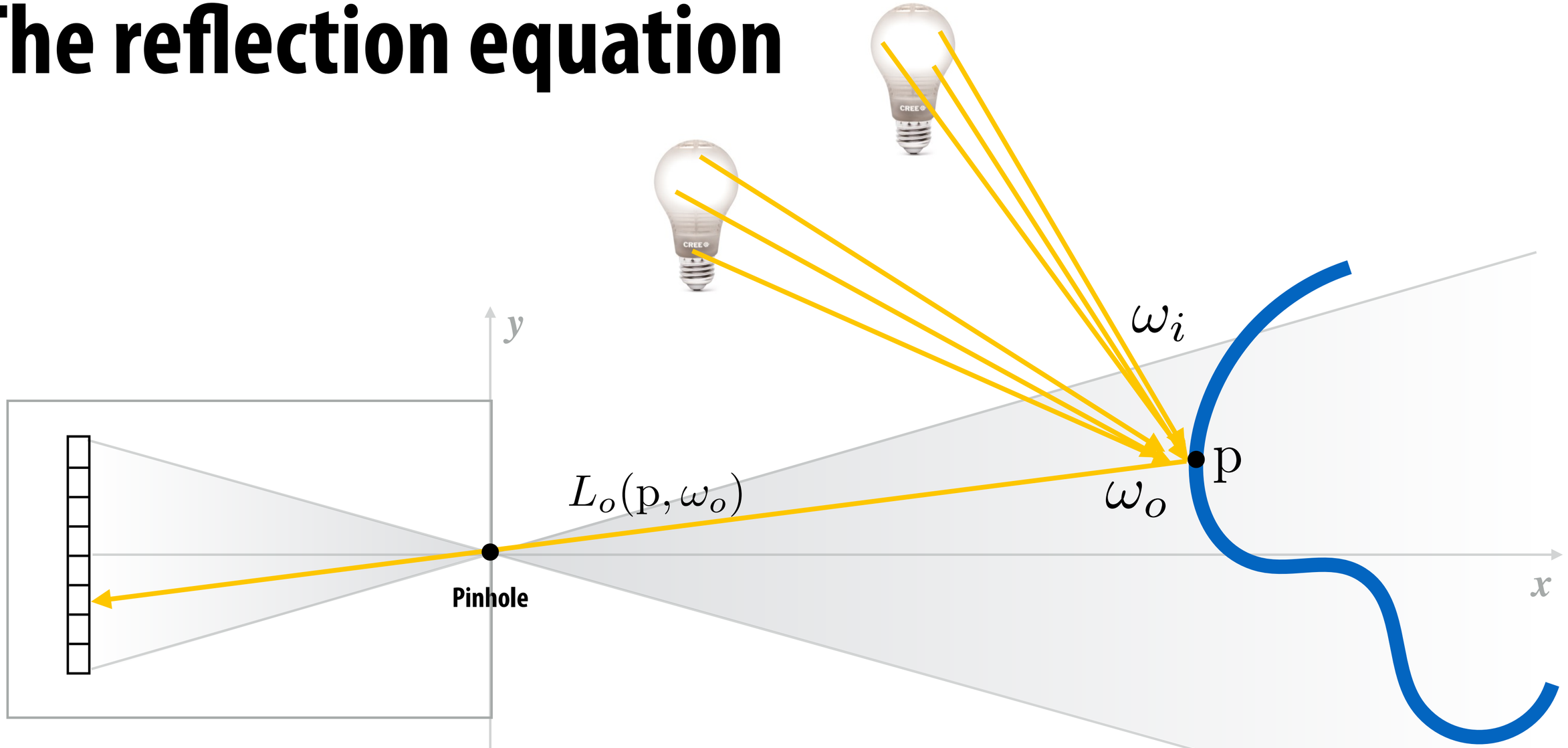
Lecture 14:

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



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

Need to know incident radiance.

So far, have only computed incoming radiance from scene light sources.

Review: solving the reflection equation

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

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

$$\frac{1}{N} \sum_{j=1}^{N} \frac{f_r(\mathrm{p}, \omega_j \rightarrow \omega_r)\, L_i(\mathrm{p}, \omega_j)\, \cos\theta_j}{p(\omega_j)}$$

Optimizing the direct lighting estimate

"Splitting"

- Consider estimating an integrand of the form:

$$\int_A \int_B f(a,b)\,\mathrm{d}a\,\mathrm{d}b$$

- 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.**

Pseudocode: estimating reflected light

```
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;
}
```

Reflectance estimate

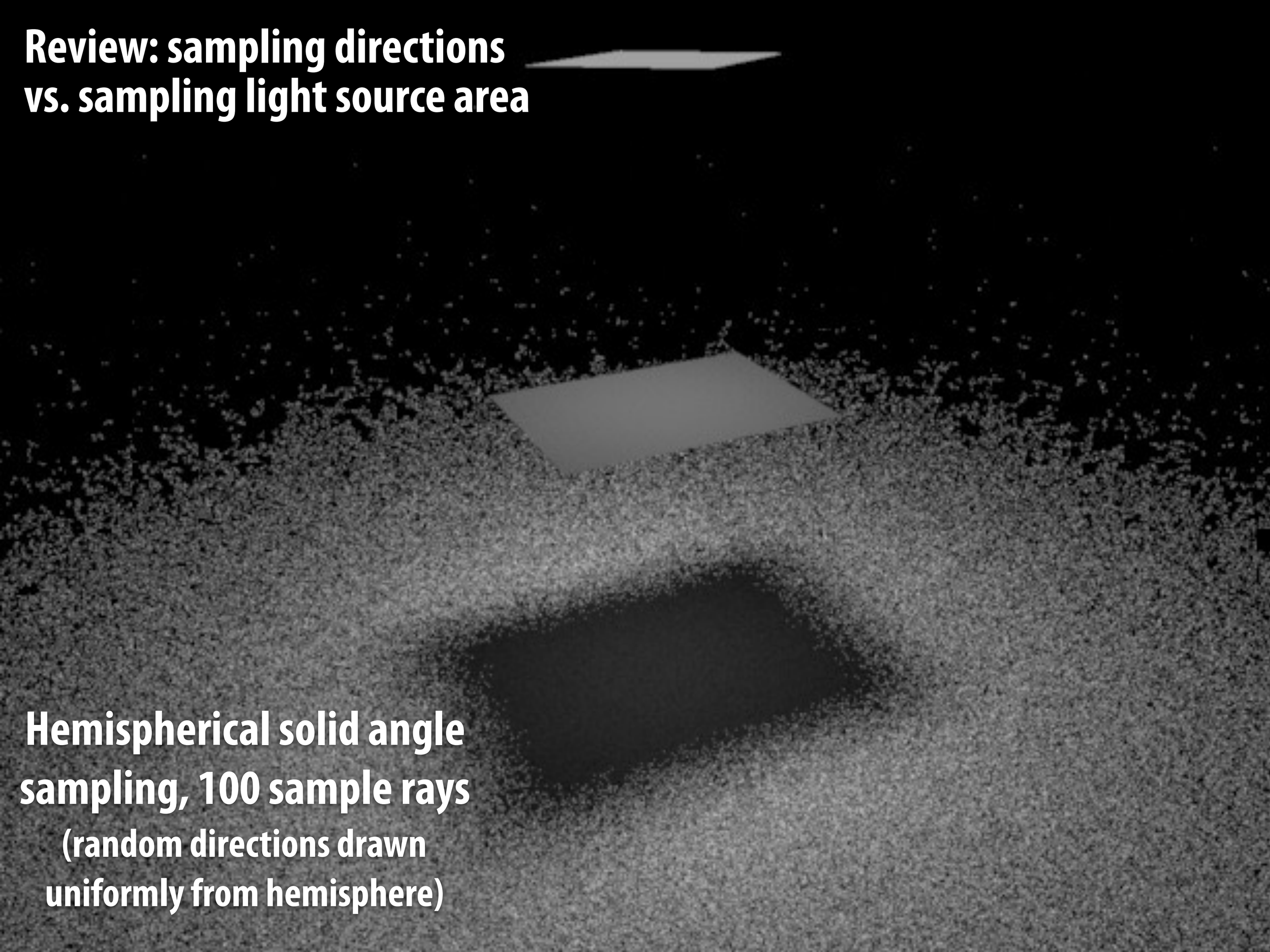
Estimate of light incident on pixel

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

Review: sampling directions vs. sampling light source area



Hemispherical solid angle sampling, 100 sample rays

(random directions drawn uniformly from hemisphere)

Single area light source

```
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;
            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?

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

Multiple area light sources

```
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;
}
```

Multiple sources, fixed number of reflectance estimate samples

```
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 light pdf, pdf;
            int light_index = randomly_pick_light(&light_pdf);
            light[light_idx].generate_dir_sample(&wi, &pdf);   // dir to random point
                                                               // on random 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)) / (light_pdf * pdf);
        }
        result += num_lights * Lr / samples_refl_estimate;
      }
   }
   return result / samples_pixel;
}
```

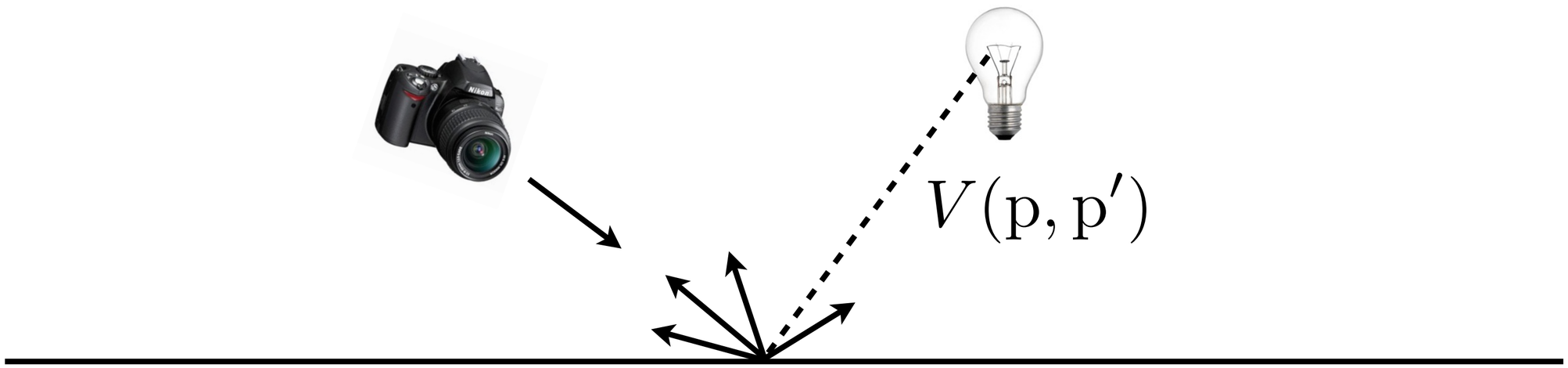
Russian roulette

- **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:**

$$L = \frac{f_r(\omega_i \rightarrow \omega_o)\, L_i(\omega_i)\, V(\mathrm{p}, \mathrm{p}')\, \cos\theta_i}{p(\omega_i)}$$



Russian roulette

$$L = \frac{f_r(\omega_i \rightarrow \omega_o)\, L_i(\omega_i)\, V(\mathrm{p}, \mathrm{p}')\, \cos\theta_i}{p(\omega_i)}$$

$$\downarrow$$

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

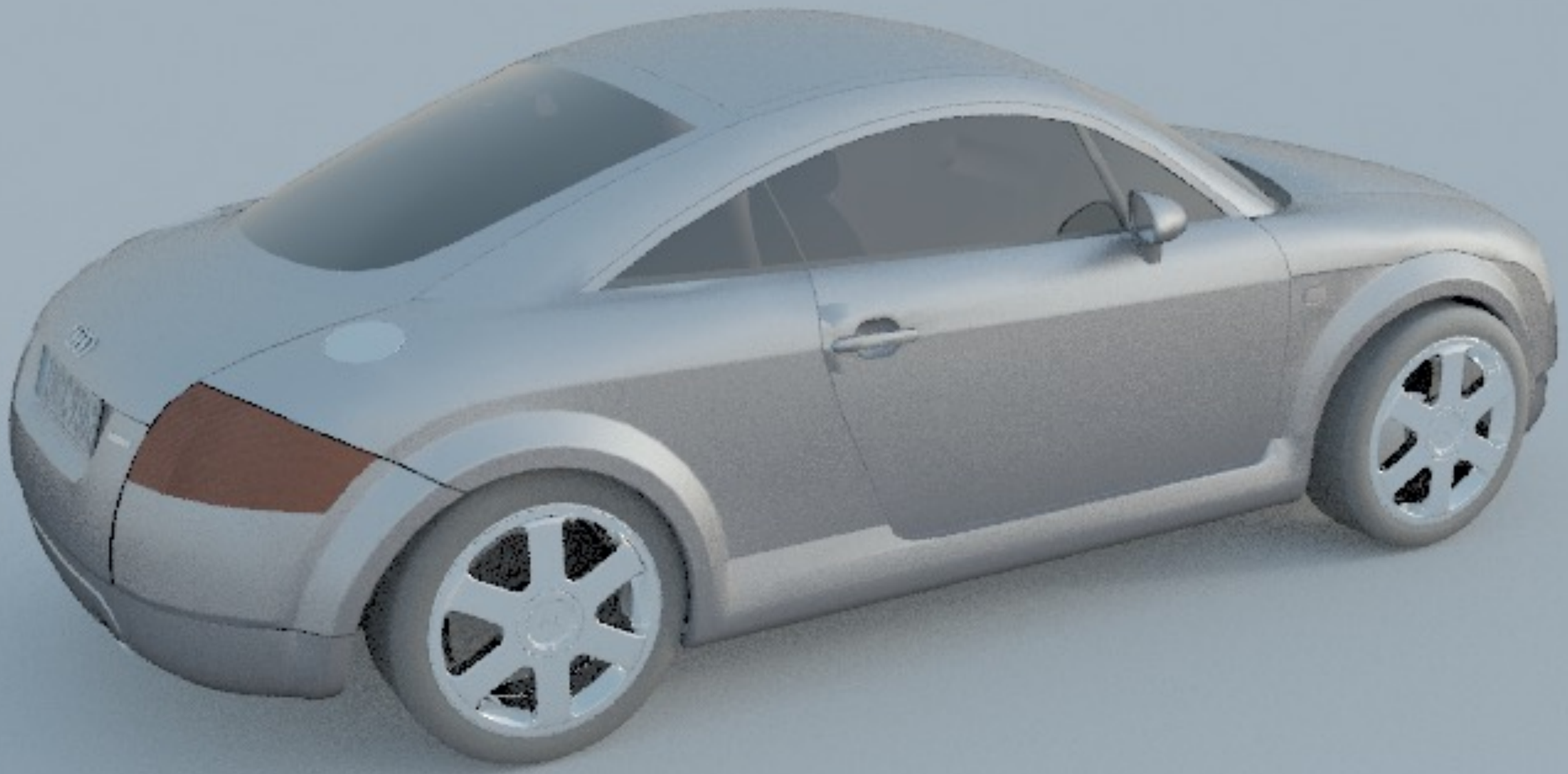
- If tentative contribution (in brackets) is small, total contribution to the image will be small regardless of $V(\mathrm{p}, \mathrm{p}')$
- Ignoring low-contribution samples introduces systemic error
  - No longer an unbiased estimator
- Instead, randomly discard low-contribution samples in a way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

Russian roulette

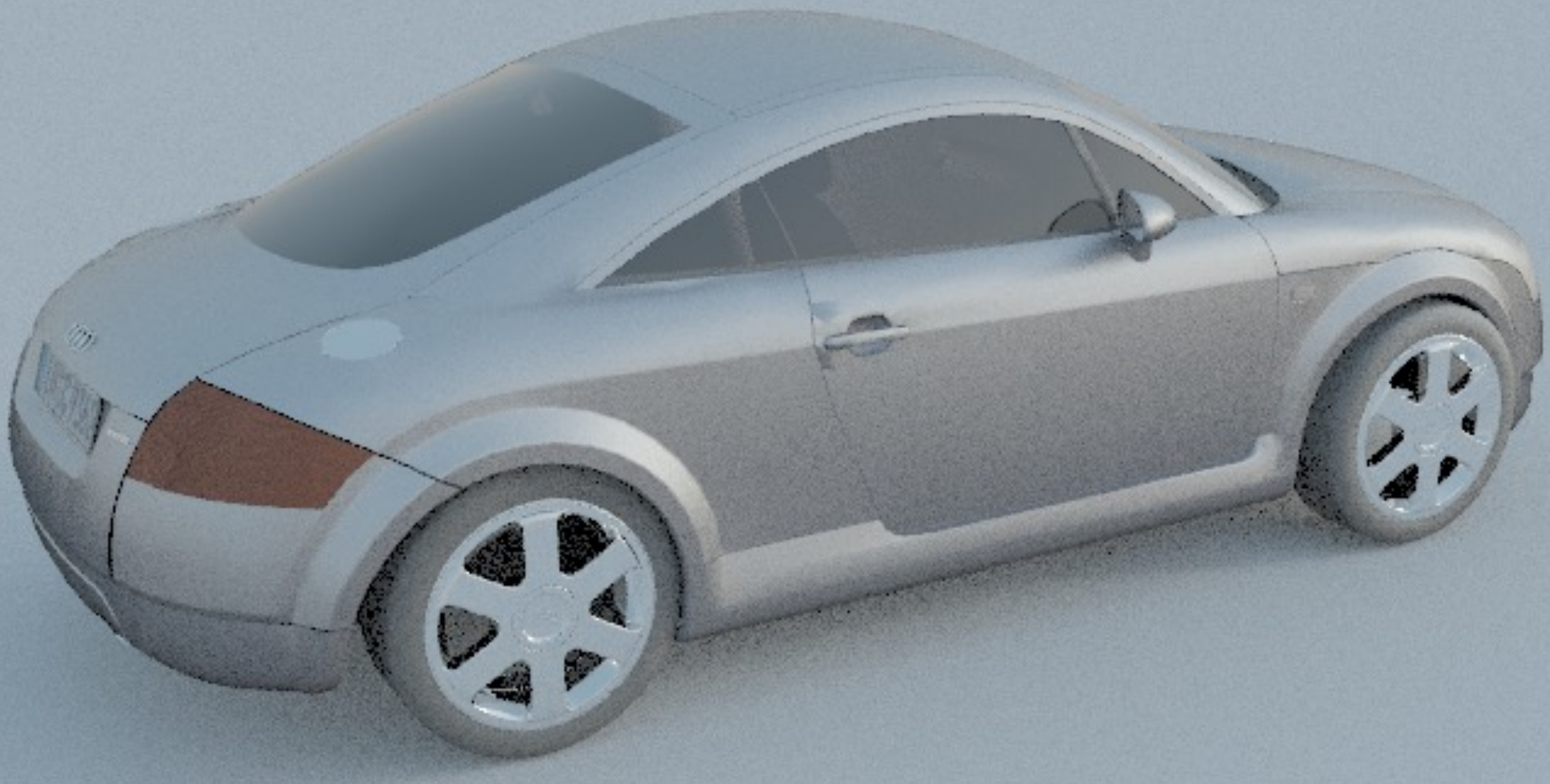
- New estimator: evaluate original estimator with probability p_{rr}, reweight. Otherwise ignore.
- Same expected value as original estimator:

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

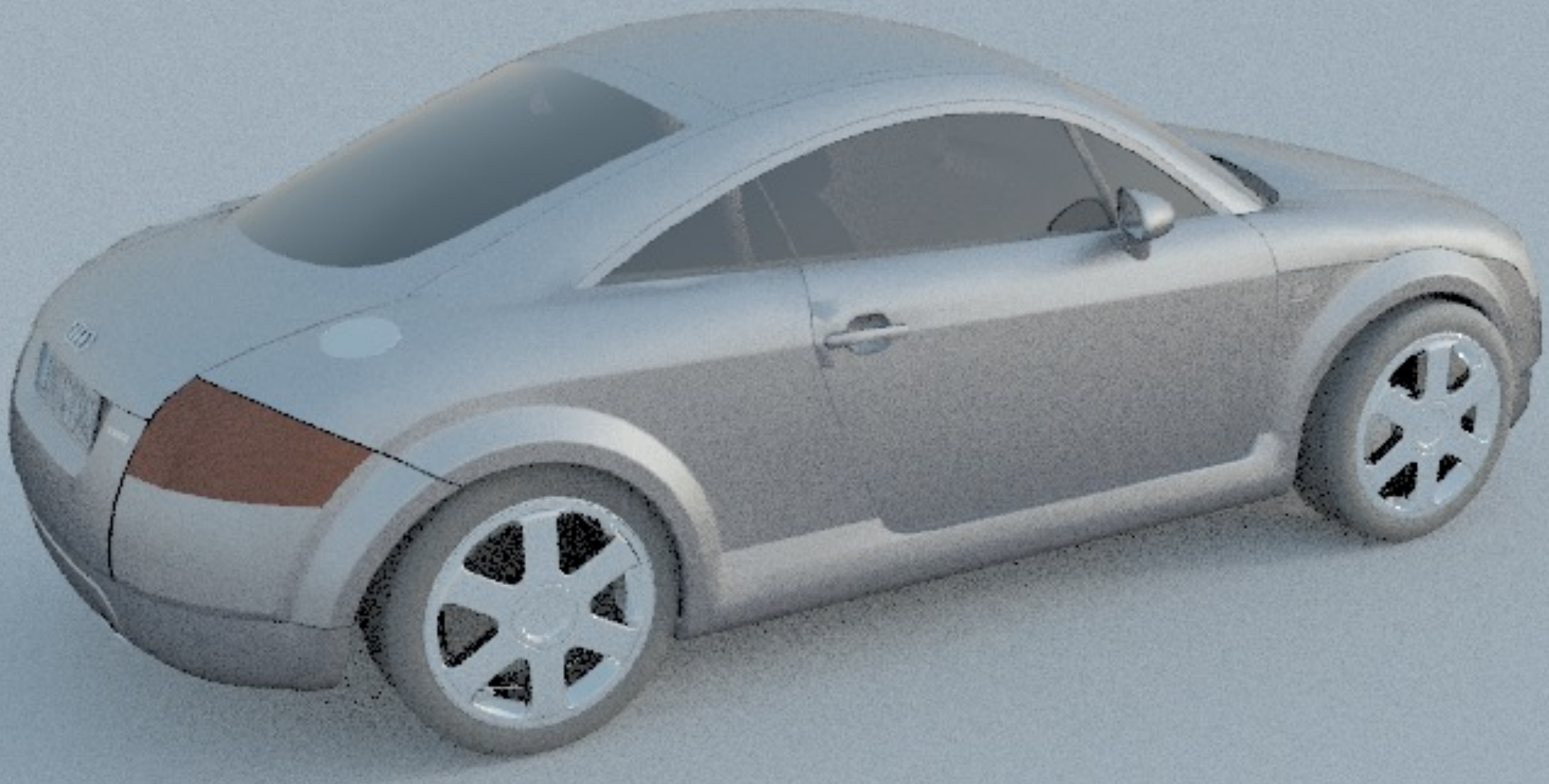
- Want to choose p_{rr} to maximize Monte Carlo efficiency...



No Russian roulette: 6.4 seconds



Russian roulette: terminate 50% of all contributions with luminance less than 0.25: 5.1 seconds



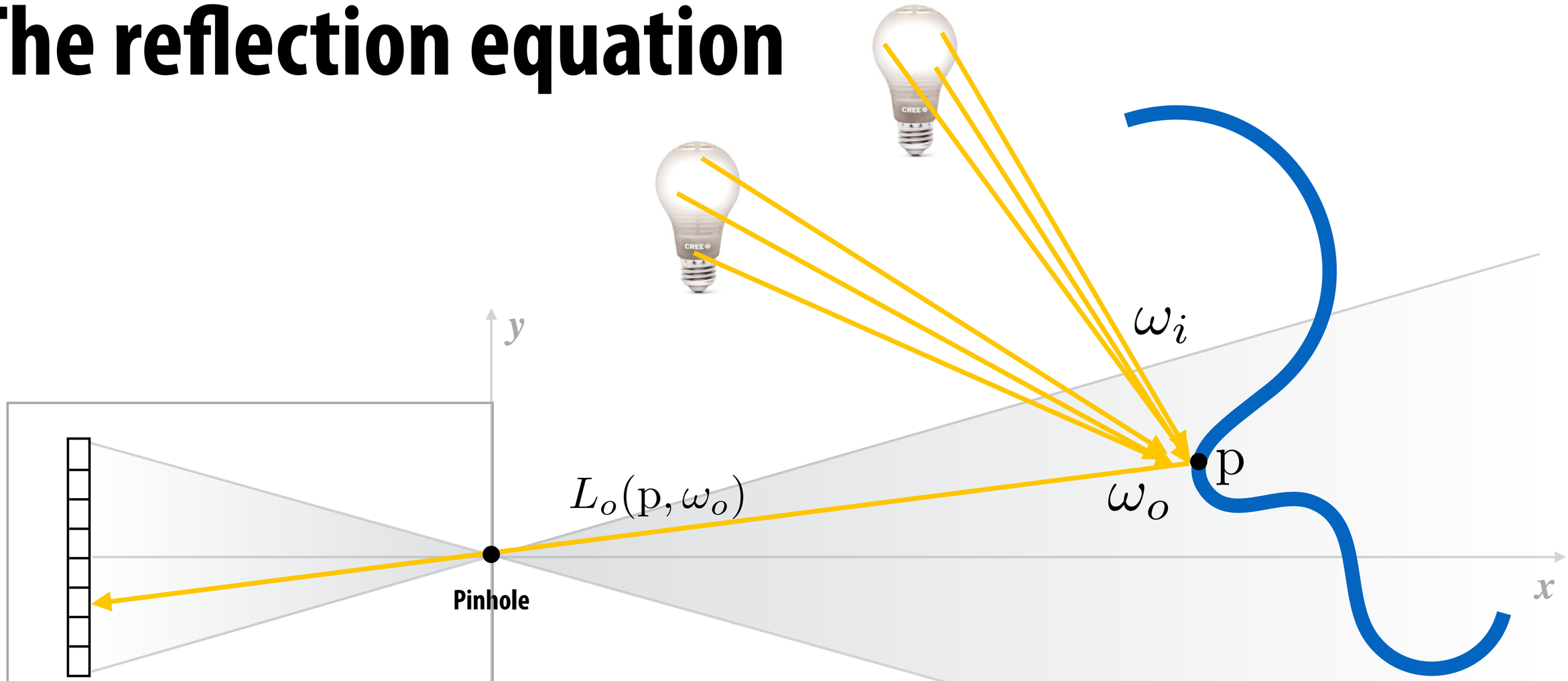
Russian roulette: terminate 50% of all contributions with luminance less than 0.5: 4.9 seconds

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

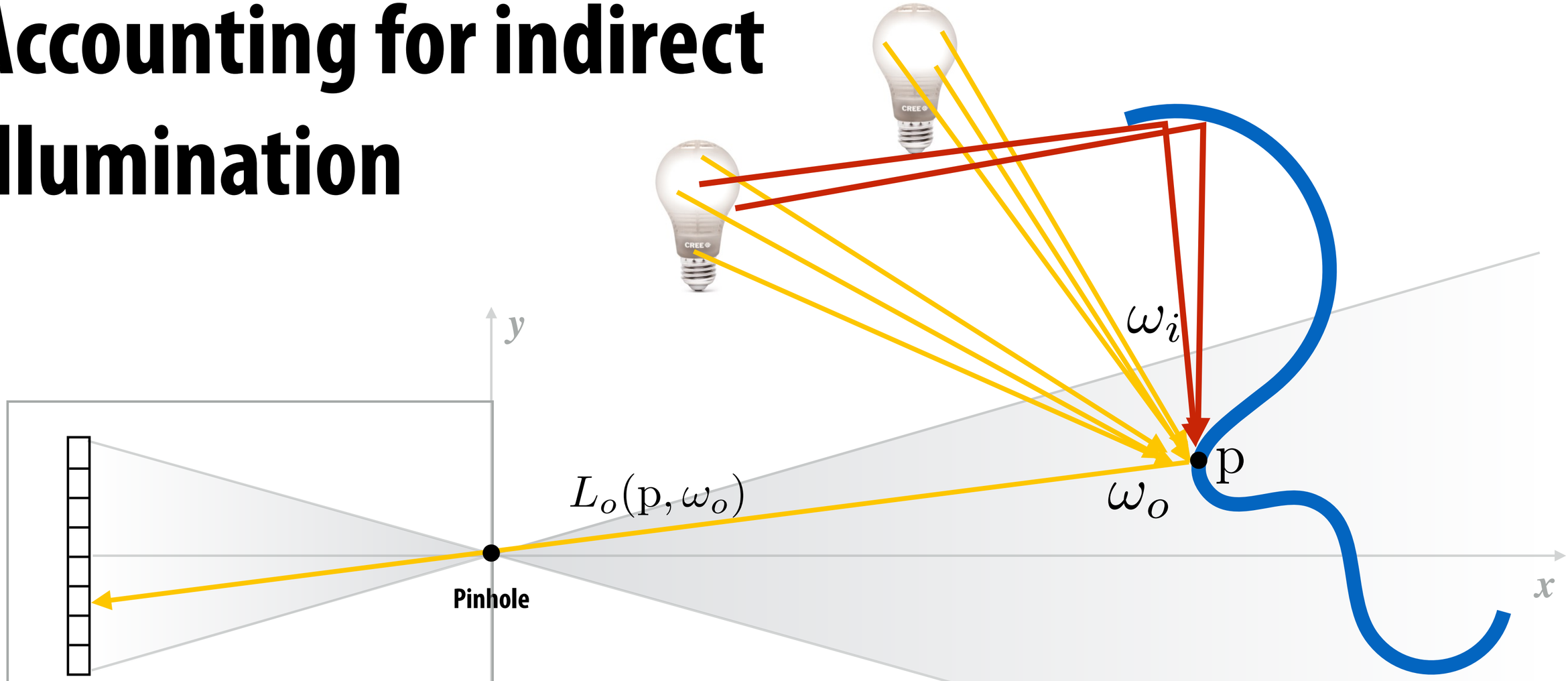


$$L_o(\mathrm{p}, \omega_o) = L_e(\mathrm{p}, \omega_o) + \int_{H^2} f_r(\mathrm{p}, \omega_i \rightarrow \omega_o)\, L_i(\mathrm{p}, \omega_i)\, \cos\theta_i \, \mathrm{d}\omega_i$$

Need to know incident radiance.

So far, have only computed incoming radiance from scene light sources.

Accounting for indirect illumination



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

Incoming light energy from direction ω_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**
  - **[outgoing] - [incoming] = [emitted] - [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:**

$$E_o(\mathrm{p}) - E_i(\mathrm{p}) = E_e(\mathrm{p}) - E_a(\mathrm{p})$$

Surface balance equation

[outgoing] = [emitted] + [reflected]

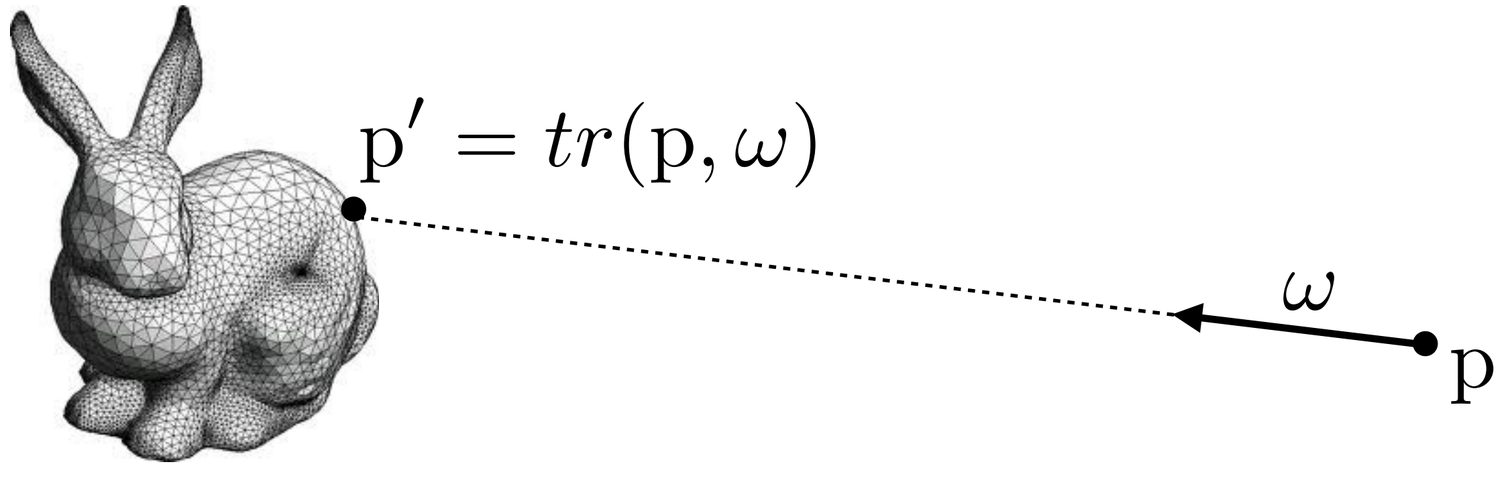
$$
\begin{aligned}
L_o(\mathrm{p}, \omega_o) &= L_e(\mathrm{p}, \omega_o) + L_r(\mathrm{p}, \omega_o) \\
&= L_e(\mathrm{p}, \omega_o) + \int_{H^2} f_r(\mathrm{p}, \omega_i \rightarrow \omega_o)\, L_i(\mathrm{p}, \omega_i)\, \cos\theta_i \, \mathrm{d}\omega_i
\end{aligned}
$$

The reflection equation

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

Let's rewrite using transport function: $tr(\mathrm{p}, \omega)$

Returns first intersection point on surface in the scene along ray defined by (p, ω)



The rendering equation: directional form

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

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

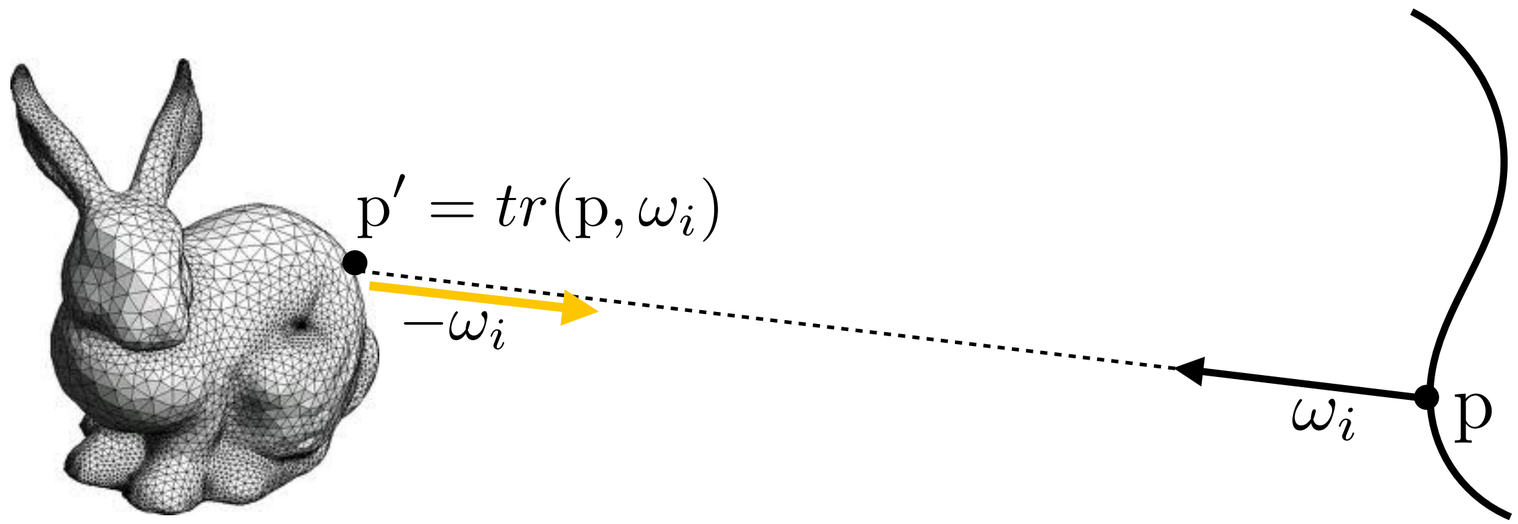
Light scattering

Light transport

Radiance invariance along rays:

$$L_i(\mathrm{p}, \omega_i) = L_o(tr(\mathrm{p}, \omega_i), -\omega_i)$$

"Radiance arriving at p ***from direction*** ω_i ***is same as radiance leaving*** p' ***towards*** p***."***



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

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

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

Path tracing: indirect illumination

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

- **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:**

$$\frac{f_r(\omega_i \rightarrow \omega_o)\, L_{o,i}(tr(\mathrm{p}, \omega_i), -\omega_i)\, \cos\theta_i}{p(\omega_i)}$$

The rendering equation: area form

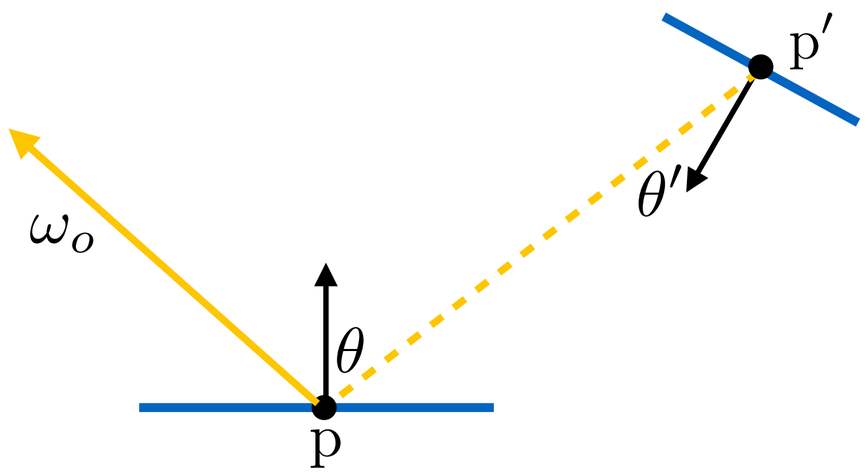
- Can equivalently write rendering equation as an integral over surface area of objects in the scene
  - Apply change of variables $\mathrm{d}\omega = \dfrac{\cos\theta}{r^2}\mathrm{d}A$
  - Introduce binary visibility function: $V(\mathrm{p} \leftrightarrow \mathrm{p}')$

$$L_o(\mathrm{p},\omega_o) = L_e(\mathrm{p},\omega_o) +$$

$$\int_A f_r(\mathrm{p}, \underbrace{(\mathrm{p}'-\mathrm{p})}_{\omega_i} \to \omega_o)\, L_o(\mathrm{p}', \underbrace{(\mathrm{p}-\mathrm{p}')}_{-\omega_i})\, \cos\theta_i\, V(\mathrm{p} \leftrightarrow \mathrm{p}')\, \frac{\cos\theta'}{|\mathrm{p}-\mathrm{p}'|^2}\mathrm{d}\mathrm{p}'$$

$$= L_e(\mathrm{p},\omega_o) + \int_A f_r(\mathrm{p}, (\mathrm{p}'-\mathrm{p}) \to \omega_o)\, L_o(\mathrm{p}', (\mathrm{p}-\mathrm{p}'))\, G(\mathrm{p} \leftrightarrow \mathrm{p}')\, \mathrm{d}\mathrm{p}'$$

$$G(\mathrm{p} \leftrightarrow \mathrm{p}') = V(\mathrm{p} \leftrightarrow \mathrm{p}')\frac{\cos\theta\cos\theta'}{|\mathrm{p}-\mathrm{p}'|^2}$$



The rendering equation: sum over paths

$$
\begin{aligned}
L_o(\mathrm{p}_1 \to \mathrm{p}) = & L_e(\mathrm{p}_1 \to \mathrm{p}) \\
& + \int_A L_e(\mathrm{p}_2 \to \mathrm{p}_1) f(\mathrm{p}_2 \to \mathrm{p}_1 \to \mathrm{p}) G(\mathrm{p}_1 \leftrightarrow \mathrm{p}_2) \mathrm{d}A(\mathrm{p}_2) \\
& + \int_A \int_A L_e(\mathrm{p}_3 \to \mathrm{p}_2) f(\mathrm{p}_3 \to \mathrm{p}_2 \to \mathrm{p}_1) G(\mathrm{p}_2 \leftrightarrow \mathrm{p}_3) f(\mathrm{p}_2 \to \mathrm{p}_1 \to \mathrm{p}) G(\mathrm{p}_1 \leftrightarrow \mathrm{p}_2) \mathrm{d}A(\mathrm{p}_3) \mathrm{d}A(\mathrm{p}_2) \\
& + \cdots
\end{aligned}
$$

Path of length 1

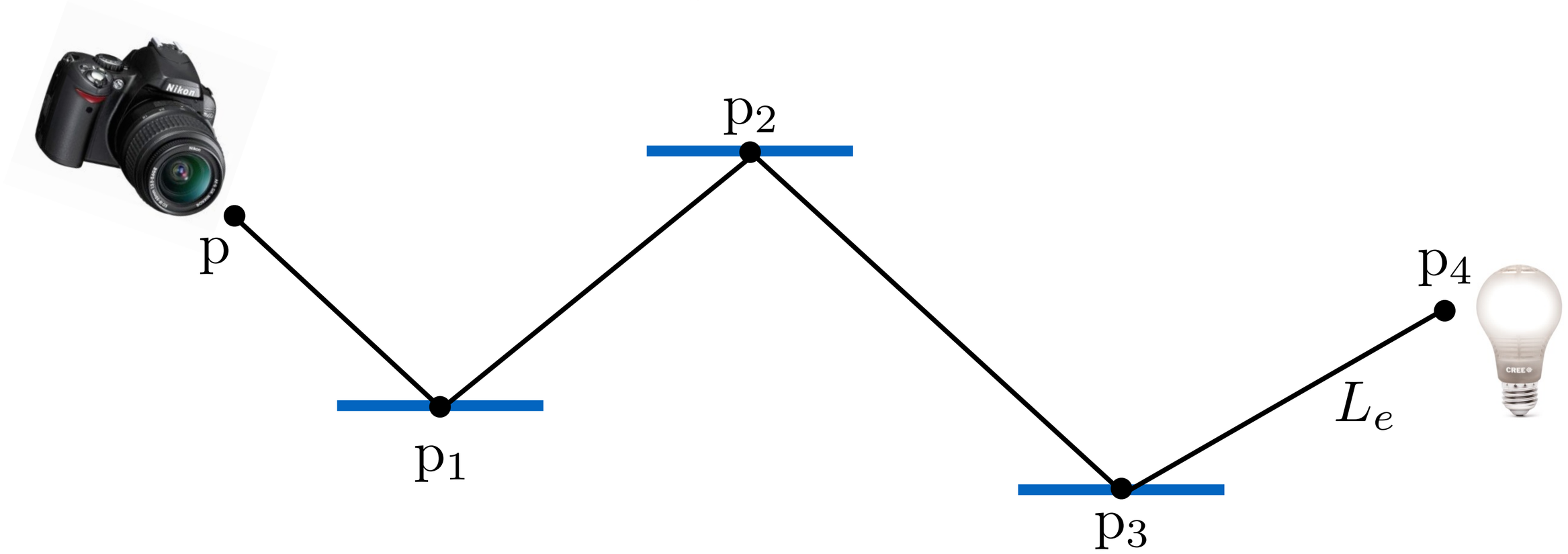
Paths of length 2

Paths of length 3

Energy reaching p from all paths of length n

$$
L_o(\mathrm{p}_1 \to \mathrm{p}) = \sum_{n=1}^{\infty} P(\bar{\mathrm{p}}_n)
$$

Path of length n (n+1 vertices)



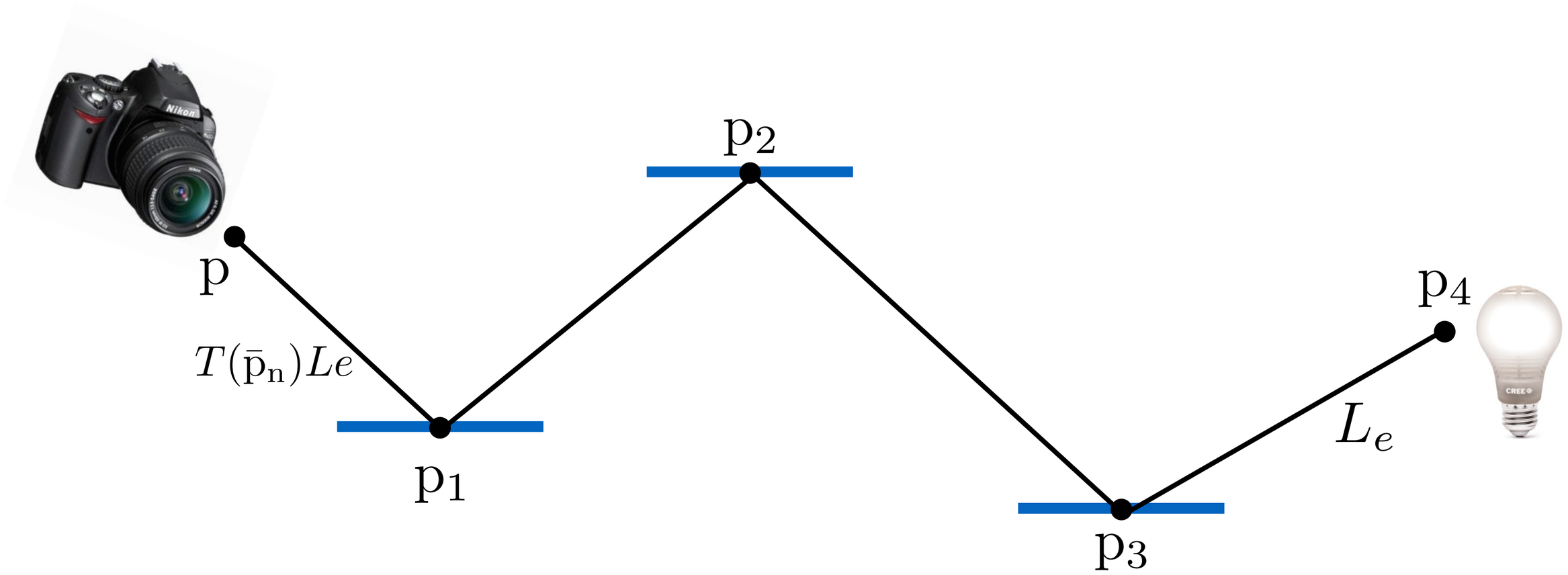
The rendering equation: sum over paths

$$L_o(\bar{\mathrm{p}}_\mathrm{n}) = \int_A \int_A \int_A \cdots \int_A L_e(\mathrm{p_n} \rightarrow \mathrm{p_{n-1}})$$

$$\times \left(\prod_{i=1}^{n-1} f(\mathrm{p_{i+1}} \rightarrow \mathrm{p_i} \rightarrow \mathrm{p_{i-1}}) G(\mathrm{p_{i+1}} \leftrightarrow \mathrm{p_i}) \right) \mathrm{d}A(\mathrm{p_2}) \cdots \mathrm{d}A(\mathrm{p_n})$$

$$= \int_A \int_A \int_A \cdots \int_A L_e(\mathrm{p_n} \rightarrow \mathrm{p_{n-1}}) T(\bar{\mathrm{p}}_\mathrm{n}) \mathrm{d}A(\mathrm{p_2}) \cdots \mathrm{d}A(\mathrm{p_n})$$

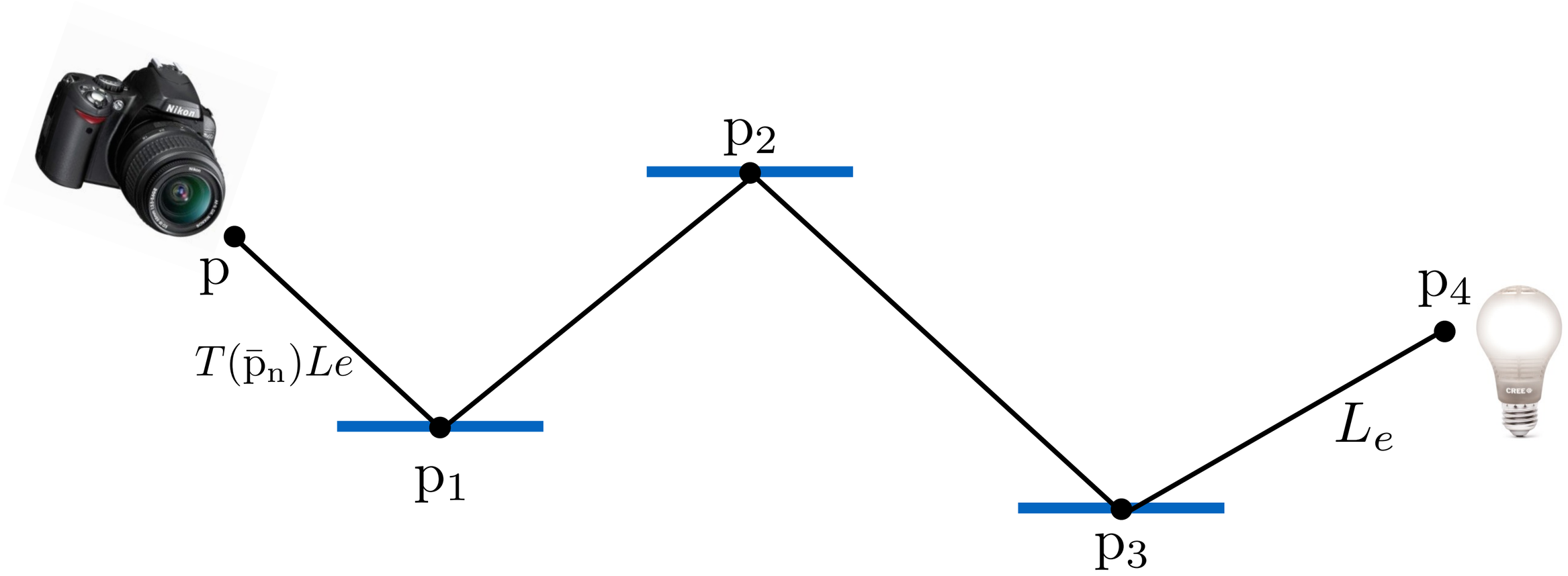
Path "throughput" = fraction of light from light source at point p_n reaching p



The rendering equation: sum over paths

$$L_o(\mathrm{p}_1 \to \mathrm{p}) = \frac{1}{1-q_1}(P(\bar{\mathrm{p}}_1) + \frac{1}{1-q_2}(P(\bar{\mathrm{p}}_2) + \frac{1}{1-q_3}(P(\bar{\mathrm{p}}_3)...$$

q_i = probability of terminating before paths of length 1



Path tracing pseudocode

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

```
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)));
}
```

Path tracing pseudocode

$$L_o(\mathrm{p},\omega_o) \approx L_e(\mathrm{p},\omega_o) + \frac{f_r(\omega_i \rightarrow \omega_o)\, L_{o,d}(tr(\mathrm{p},\omega_i), -\omega_i)\, \cos\theta_i}{p(\omega_i)} +$$

$$\frac{f_r(\omega_i' \rightarrow \omega_o)\, L_{o,i}(tr(\mathrm{p},\omega_i'), -\omega_i')\, \cos\theta_i'}{p(\omega_i')}$$

$p(\omega_i')$ ← *P(choosing w'$_i$ | not_terminating) P(not terminating)*

```
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:**

$$(K \circ f) = \int k(x, x')\, f(x')\, \mathrm{d}x'$$

$$(D \circ f) = \frac{\partial f}{\partial x} x$$

Light transport operators

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

Full light transport operator: $K = R \circ T$

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

Light propagation operator: $T \circ f = f(tr(\mathrm{p}, \omega), -\omega)$

$$L_i = T \circ L_o$$

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

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

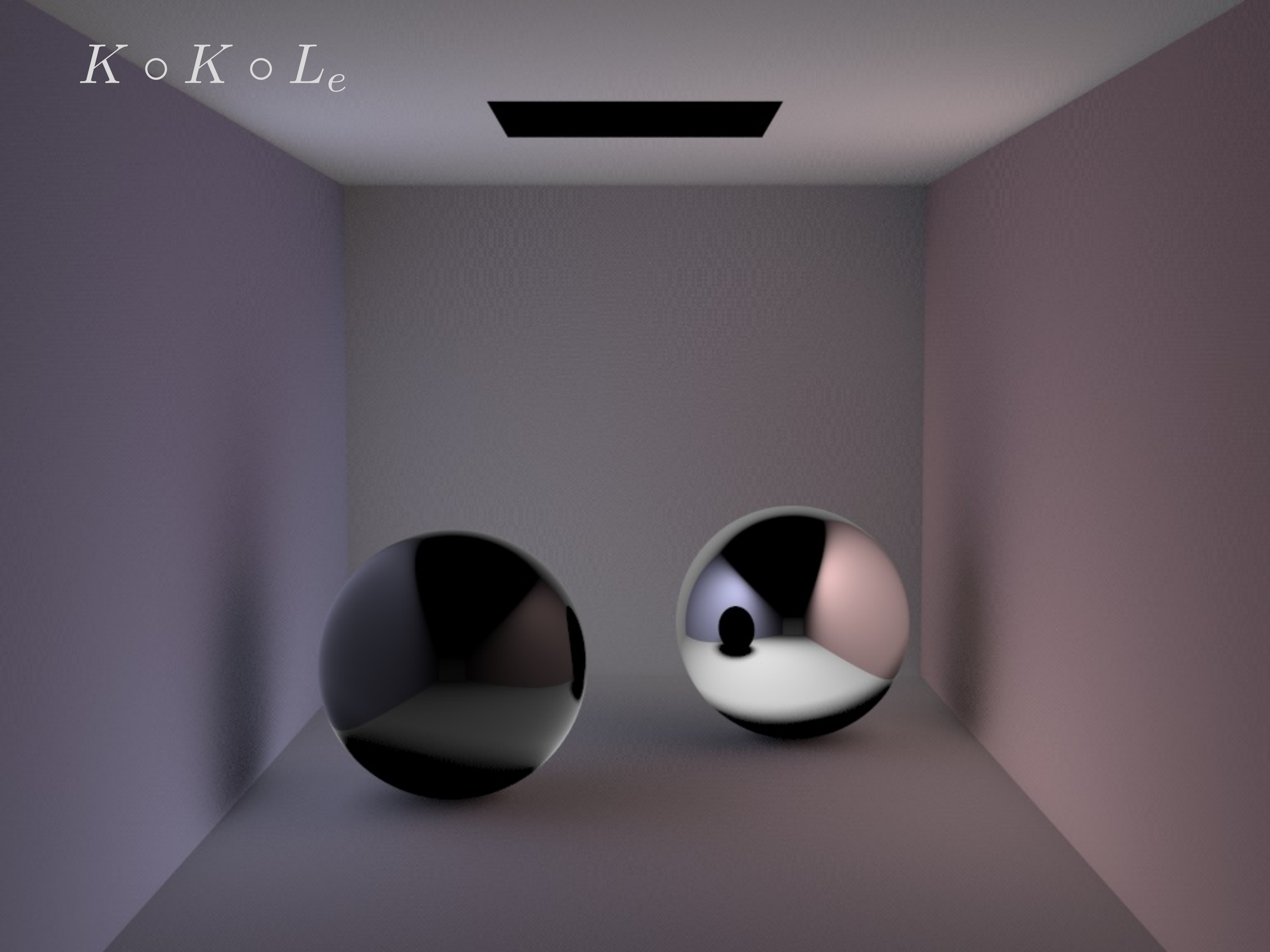
$$\begin{aligned} L^1 &= L_e \\ L^2 &= L_e + K \circ L^1 \\ &\vdots \\ L^n &= L_e + K \circ L^{n-1} \end{aligned}$$

Energy conservation makes it possible to show that a solution exists

L_e

$K \circ L_e$

$K \circ K \circ L_e$

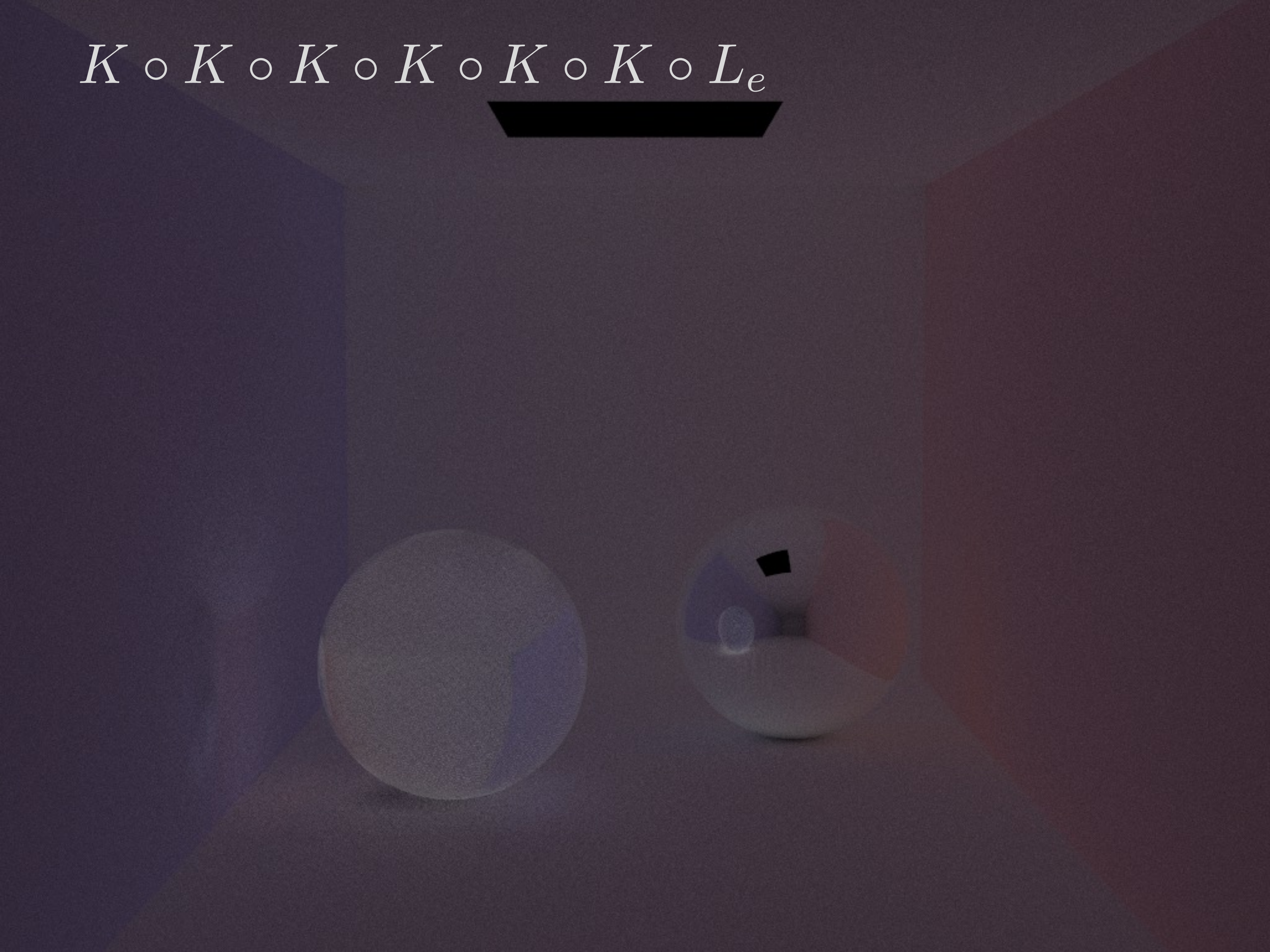


$K \circ K \circ K \circ L_e$

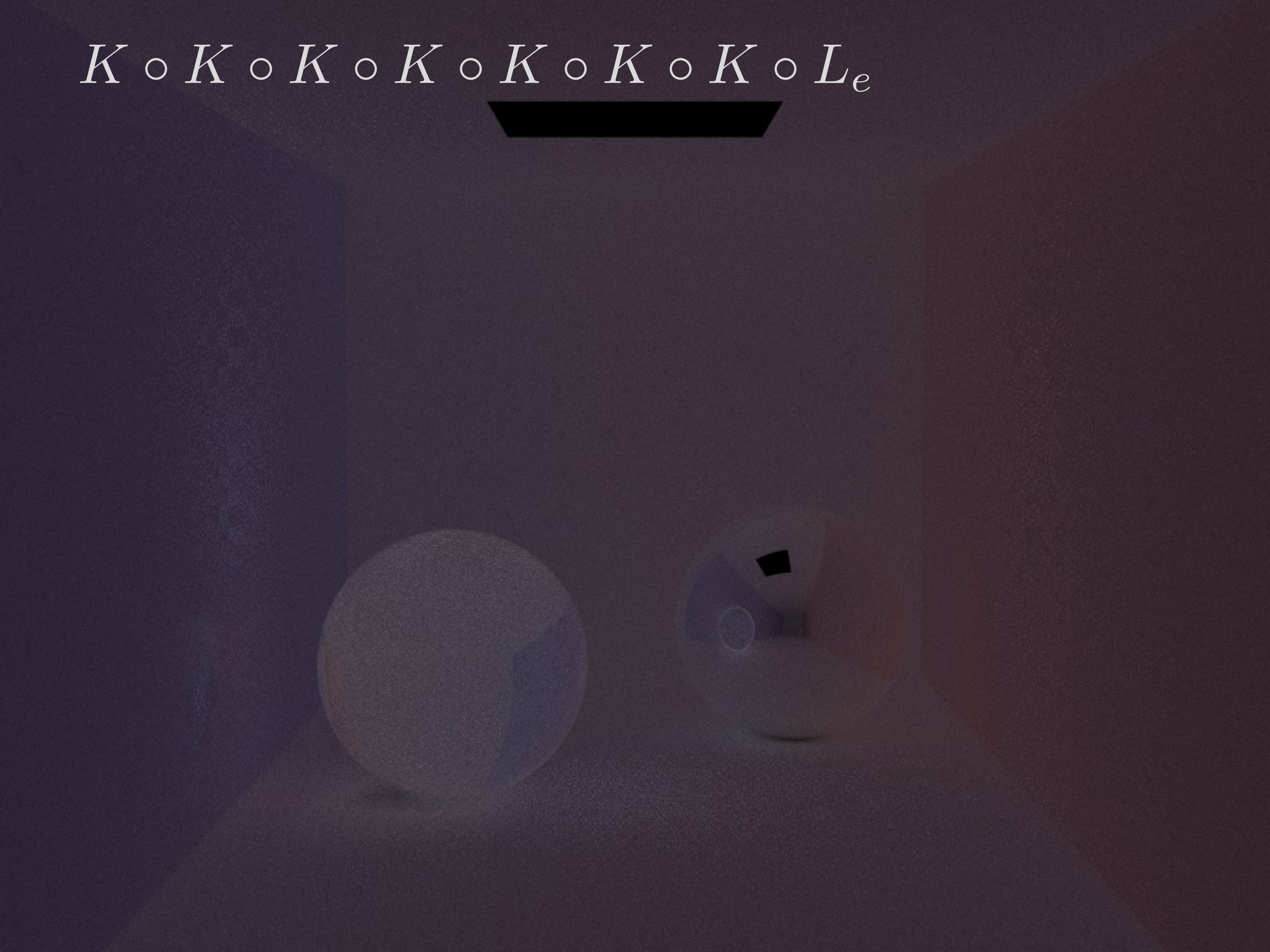
$$K \circ K \circ K \circ K \circ L_e$$

$$K \circ K \circ K \circ K \circ K \circ L_e$$

$$K \circ K \circ K \circ K \circ K \circ K \circ L_e$$



$$K \circ K \circ K \circ K \circ K \circ K \circ K \circ L_e$$



$$K \circ K \circ K \circ K \circ K \circ K \circ K \circ K \circ L_e$$

$$\sum_{i=1}^{1} K^i \circ L_e$$

$$\sum_{i=1}^{2} K^i \circ L_e$$

$$\sum_{i=1}^{3} K^i \circ L_e$$

$$\sum_{i=1}^{4} K^i \circ L_e$$

$$\sum_{i=1}^{5} K^i \circ L_e$$

$$\sum_{i=1}^{7} K^i \circ L_e$$

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