PHYSICALLY-BASED RENDERING

Based on slides by Derek Nowrouzezahrai

Monte Carlo for Global Illumination

MC for the Rendering Equation

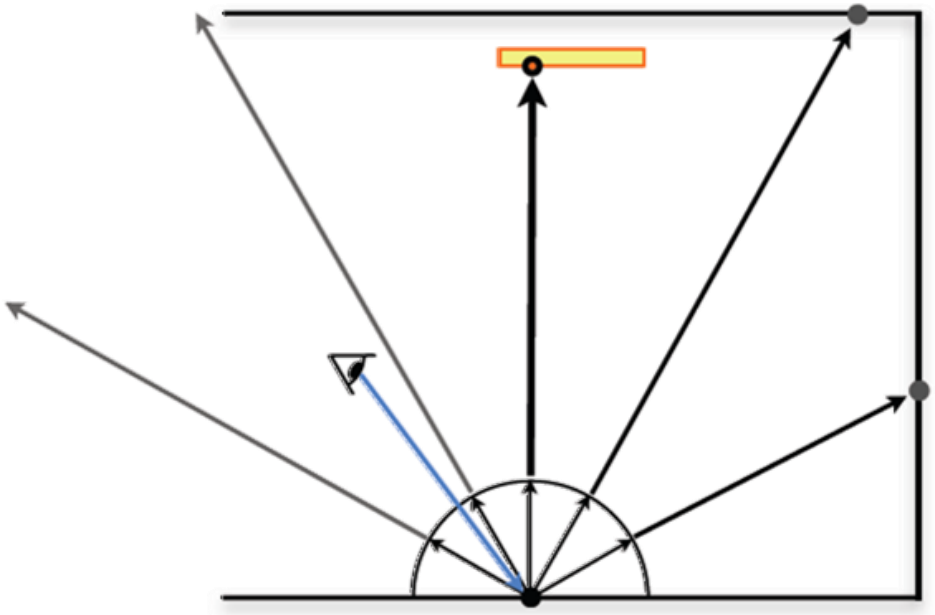
- What happens if we apply MC integration to the rendering equation?

$$L(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega} L(x', \omega_i)\, f_r(x, \omega_i, \omega_o) \cos\theta_i \, d\omega_i$$

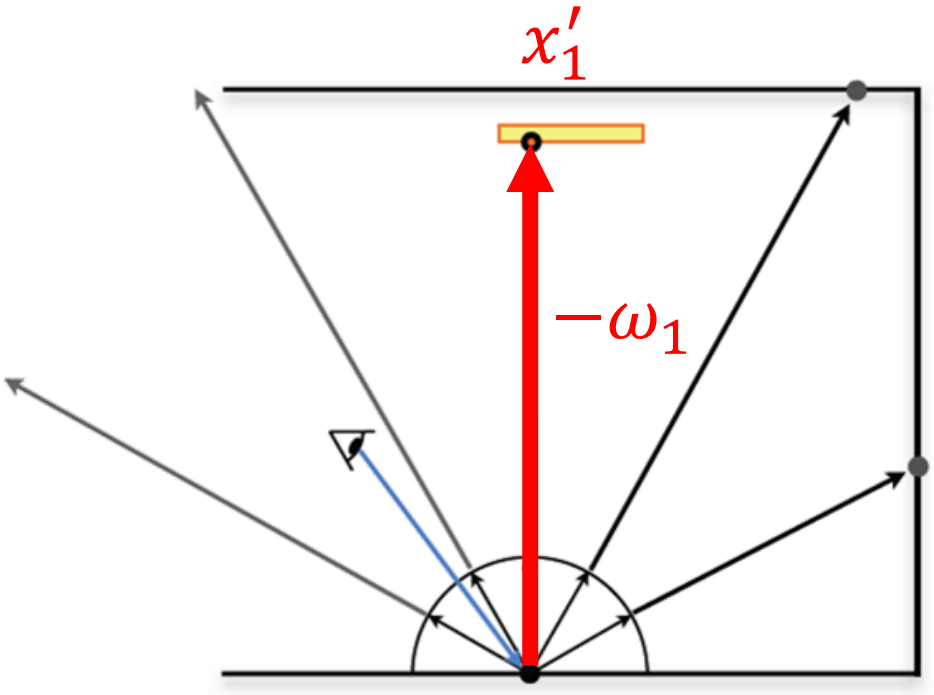
$$L(x, \omega_o) \approx L_e(x, \omega_o) + \frac{1}{N} \sum_i \frac{L(x_i', \omega_i)\, f_r(x, \omega_i, \omega_o) \cos\theta_i}{pdf(\omega_i)}$$

MC for the Rendering Equation

$$L(x,\omega_o) \approx L_e(x,\omega_o) + \frac{1}{N}\sum_i \frac{L(x_i',\omega_i)\, f_r(x,\omega_i,\omega_o)\cos\theta_i}{pdf(\omega_i)}$$



MC for the Rendering Equation



$$f_r(x_1') = 0$$

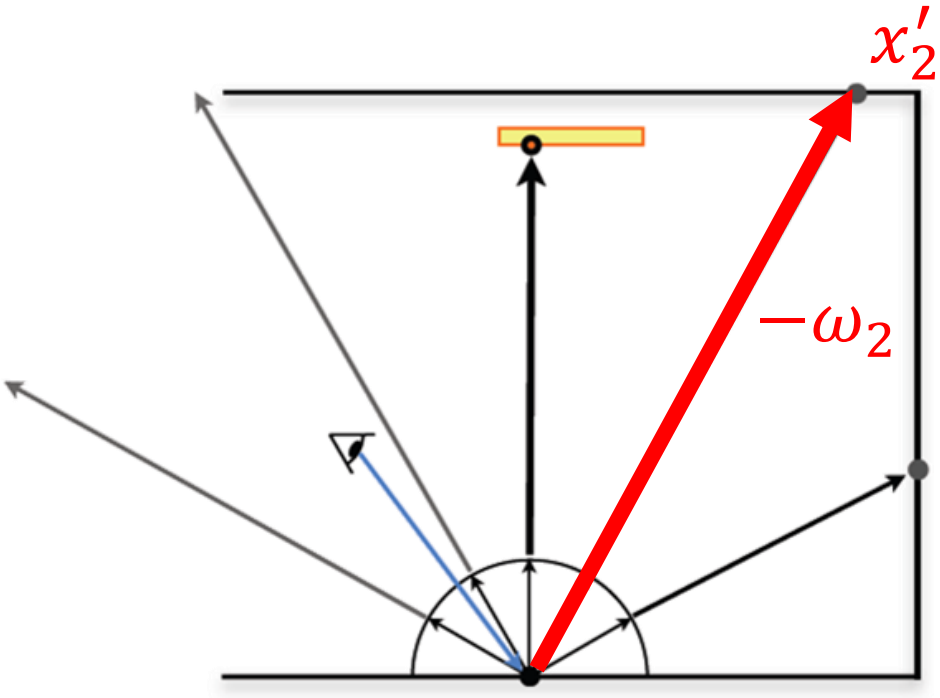
$$L(x_1', \omega_1) = L_e(x_1', \omega_1)$$

$$L(x_1', \omega_1) = L_e(x_1', \omega_1) + \frac{1}{N}\sum_j \frac{L(x_j'', \omega_j)\, f_r(x_1', \omega_j, \omega_1) \cos\theta_j}{pdf(\omega_j)}$$

- What's the value at the sample $L(x_1', \omega_1)$?

$$L(x, \omega_o) = L_e(x, \omega_o) + \frac{1}{N}\sum_i \frac{L(x_i', \omega_i)\, f_r(x, \omega_i, \omega_o) \cos\theta_i}{pdf(\omega_i)}$$

MC for the Rendering Equation



$$f_r(x_2') \neq 0$$

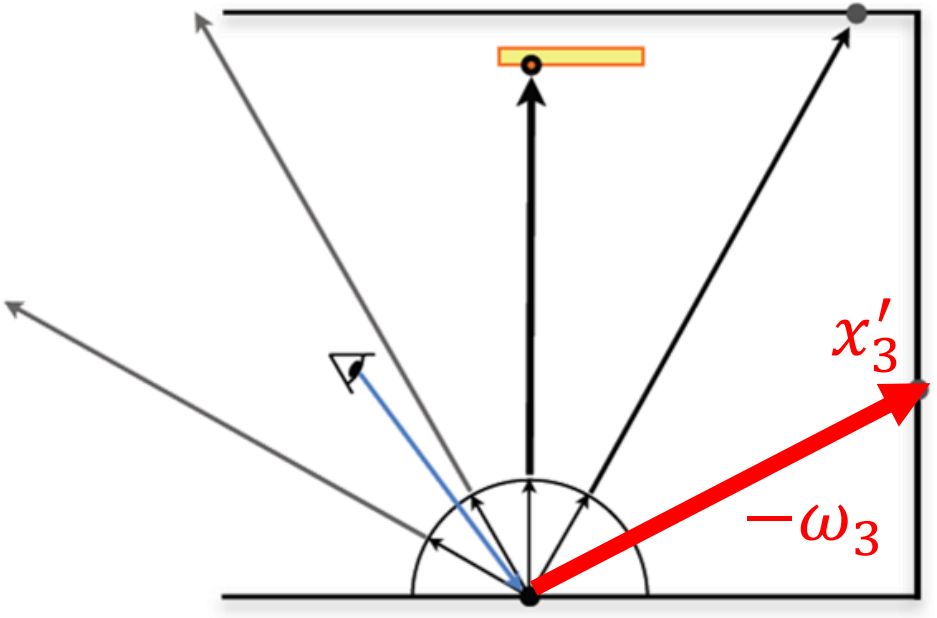
$$L_e(x_2', \omega_2) = 0$$

$$L(x_2', \omega_2) = L_e(x_2', \omega_2) + \frac{1}{N}\sum_k \frac{L(x_k''', \omega_k)\, f_r(x_2', \omega_k, \omega_2) \cos\theta_k}{pdf(\omega_k)}$$

- What's the value at the sample $L(x_2', \omega_2)$?

$$L(x, \omega_o) = L_e(x, \omega_o) + \frac{1}{N}\sum_i \frac{L(x_i', \omega_i)\, f_r(x, \omega_i, \omega_o) \cos\theta_i}{pdf(\omega_i)}$$

MC for the Rendering Equation



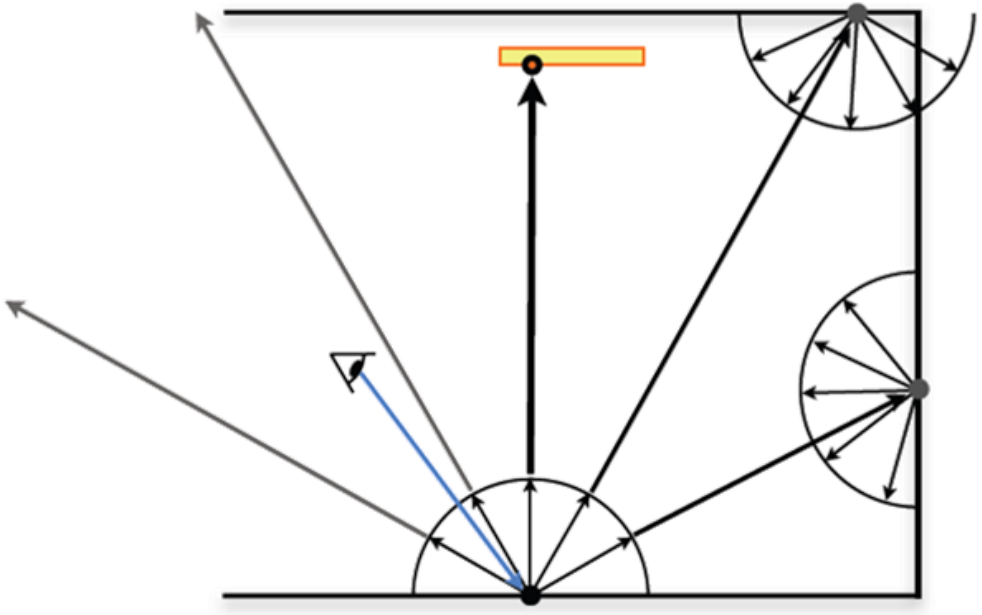
$$f_r(x_2') \neq 0$$
$$L_e(x_2', \omega_2) = 0$$
$$f_r(x_3') \neq 0$$
$$L_e(x_3', \omega_3) = 0$$

$$L(x_3', \omega_3) = L_e(x_3', \omega_3) + \frac{1}{N}\sum_l \frac{L(x_l'''', \omega_3)\, f_r(x_3', \omega_l, \omega_3) \cos\theta_l}{pdf(\omega_l)}$$

- What's the value at the sample $L(x_3', \omega_3)$?

$$L(x, \omega_o) = L_e(x, \omega_o) + \frac{1}{N}\sum_i \frac{L(x_i', \omega_i)\, f_r(x, \omega_i, \omega_o) \cos\theta_i}{pdf(\omega_i)}$$

MC for the Rendering Equation



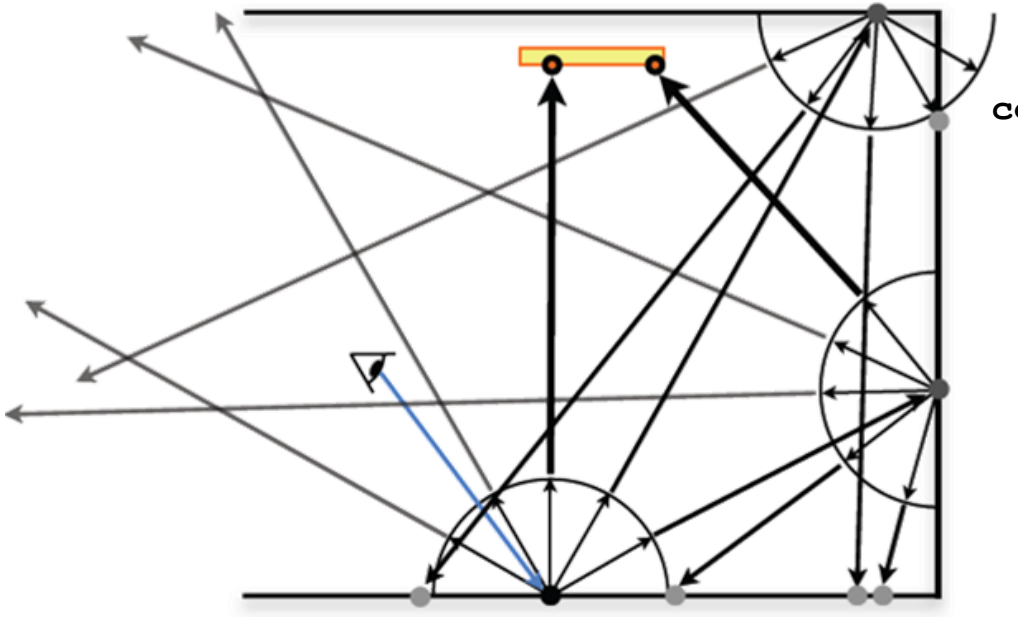
$$L(x_2', \omega_2) = \frac{1}{N}\sum_k \frac{L(x_k''', \omega_k)\, f_r(x_2', \omega_k, \omega_2) \cos\theta_k}{pdf(\omega_k)}$$

$$L(x_3', \omega_3) = \frac{1}{N}\sum_l \frac{L(x_l'''', \omega_3)\, f_r(x_3', \omega_l, \omega_3) \cos\theta_l}{pdf(\omega_l)}$$

- And so on and so forth, proceeding recursively…

$$L(x, \omega_o) = L_e(x, \omega_o) + \frac{1}{N}\sum_i \frac{L(x_i', \omega_i)\, f_r(x, \omega_i, \omega_o) \cos\theta_i}{pdf(\omega_i)}$$

MC for the Rendering Equation



```
color shade( point x )
  if( x.Le != 0 ) return x.Le;
  color L = 0;
  for( for sample i ) {
    omega_i = random direction distri-
              buted according to pdf;
    L += shade( trace(x, omega_i) )
         * brdf * cos / pdf(omega_i);
  }
  return L /= N;
```

- When/how does the recursion stop*?
- Other problems?
  - You spent exponentially **more computation time** for effects that **contribute exponentially less** to the final image!

Exponential Explosion

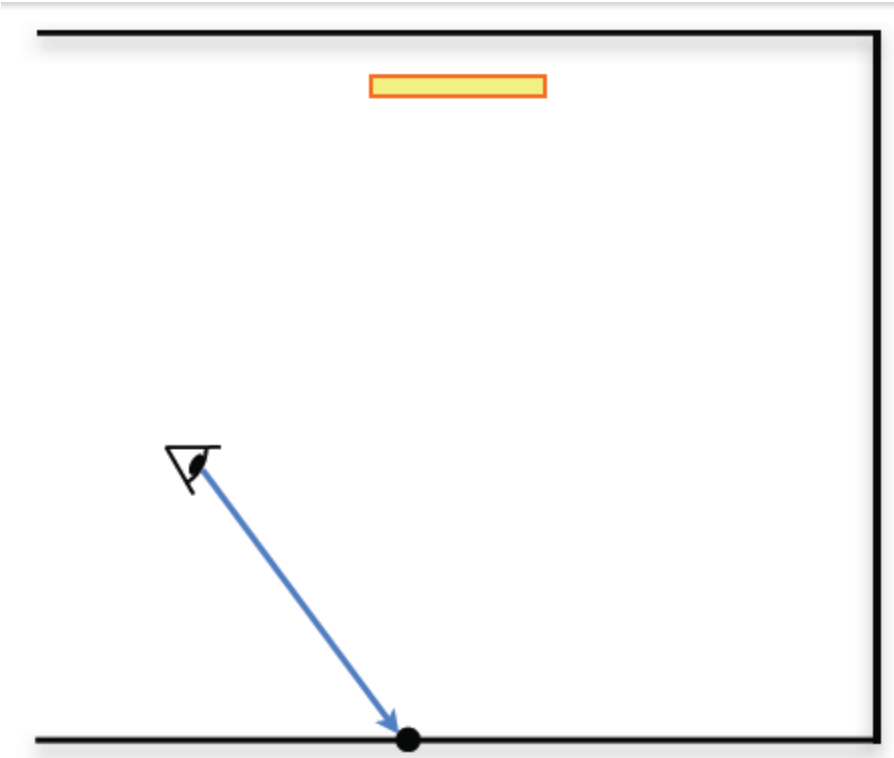
```
color shade( point x )
  if( x.Le != 0 ) return x.Le;
  color L = 0;
  for( each sample i ) {
    omega_i = random direction distributed according to pdf;
    L += shade( trace(x, omega_i) ) * brdf * cos / pdf(omega_i);
  }
  return L /= N;
```

- How can we avoid this exponential compute explosion?
  - Use an MC estimator with a single sample!

$$L(x, \omega_o) \approx L_e(x, \omega_o) + \frac{L(x', \omega)\, f_r(x, \omega, \omega_o) \cos\theta}{pdf(\omega)}$$

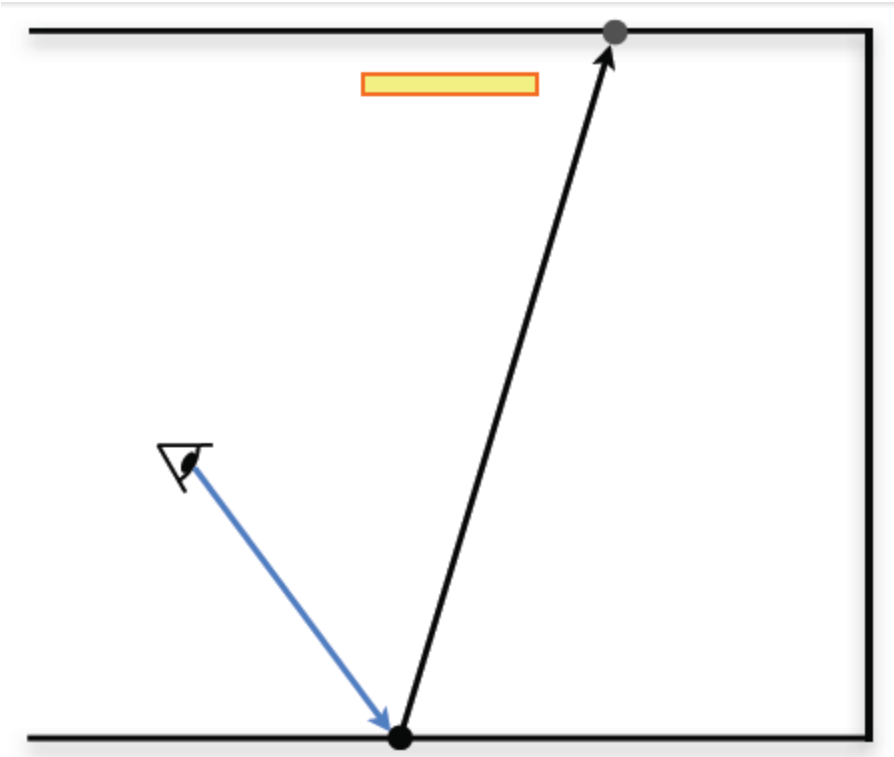
Path-tracing

$$L(x, \omega_o) \approx L_e(x, \omega_o) + \frac{L(x', \omega)\, f_r(x, \omega, \omega_o) \cos\theta}{pdf(\omega)}$$



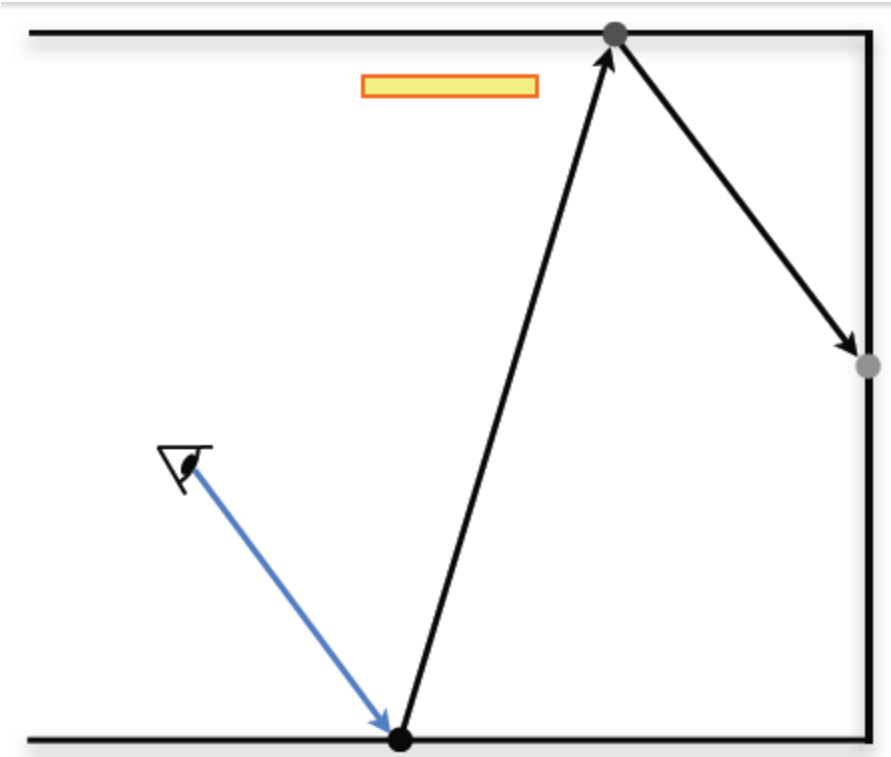
Path-tracing

$$L(x, \omega_o) \approx L_e(x, \omega_o) + \frac{L(x', \omega)\, f_r(x, \omega, \omega_o) \cos\theta}{pdf(\omega)}$$



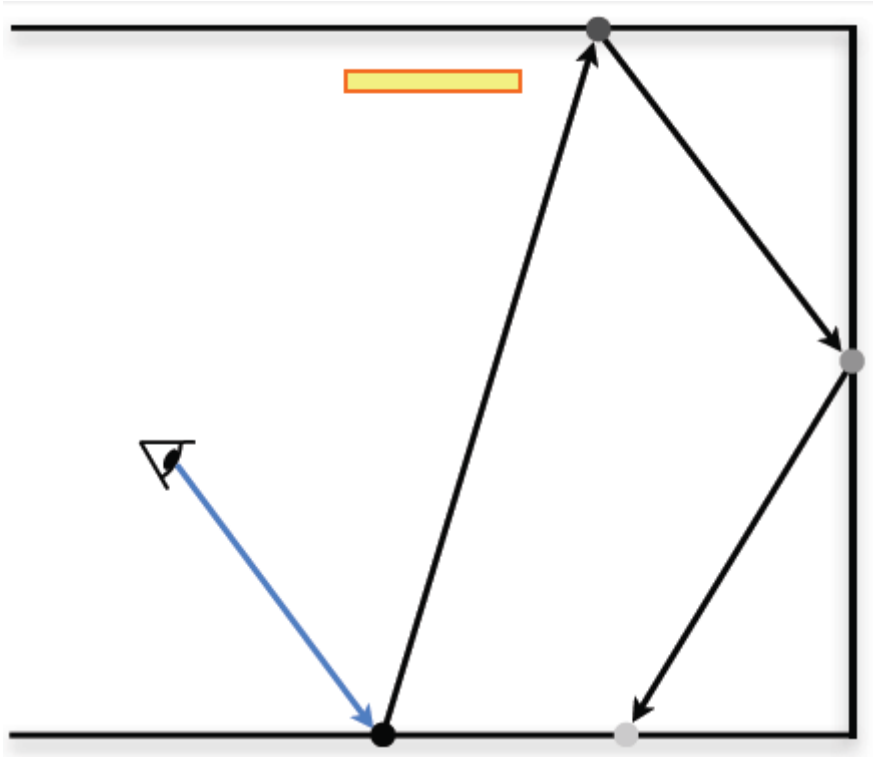
Path-tracing

$$L(x, \omega_o) \approx L_e(x, \omega_o) + \frac{L(x', \omega)\, f_r(x, \omega, \omega_o) \cos\theta}{pdf(\omega)}$$



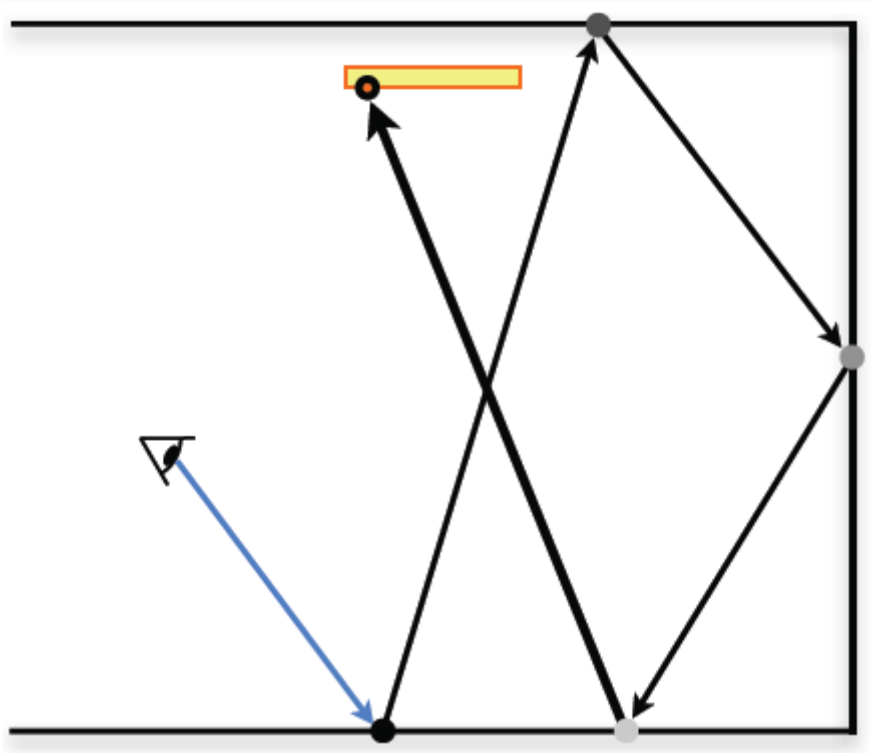
Path-tracing

$$L(x, \omega_o) \approx L_e(x, \omega_o) + \frac{L(x', \omega)\, f_r(x, \omega, \omega_o) \cos\theta}{pdf(\omega)}$$



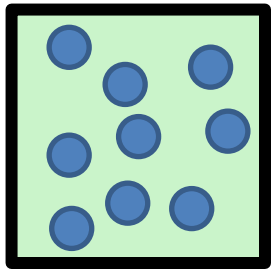
Path-tracing

$$L(x, \omega_o) \approx L_e(x, \omega_o) + \frac{L(x', \omega)\, f_r(x, \omega, \omega_o) \cos\theta}{pdf(\omega)}$$



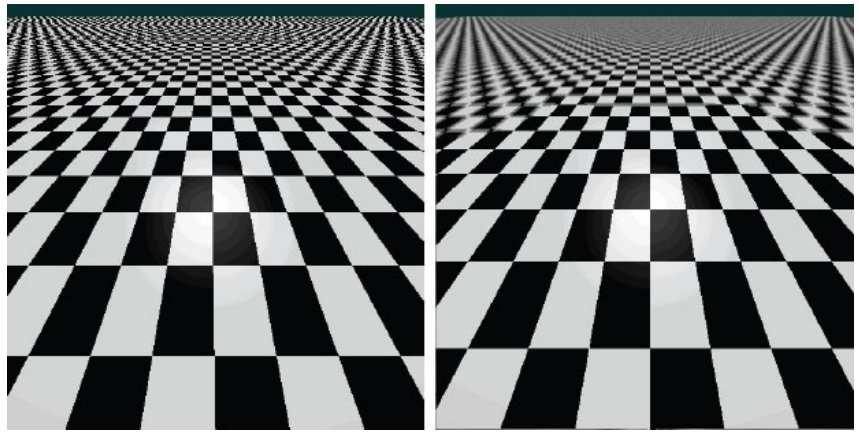
Implicit Path-tracing

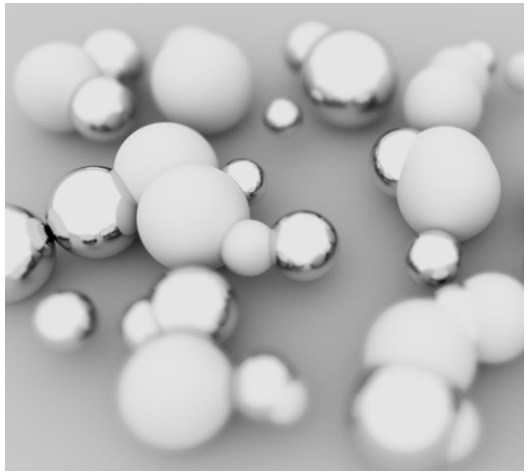
```
color shade( point x )
  if( x.Le != 0 ) return x.Le;
  color L = 0;
  omega_i = random direction distributed according to pdf;
  L += shade( trace(x, omega_i) ) * brdf * cos / pdf(omega_i);
  return L;
```

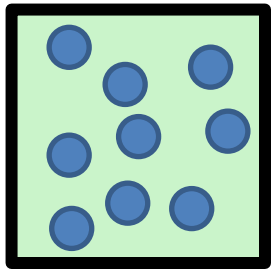


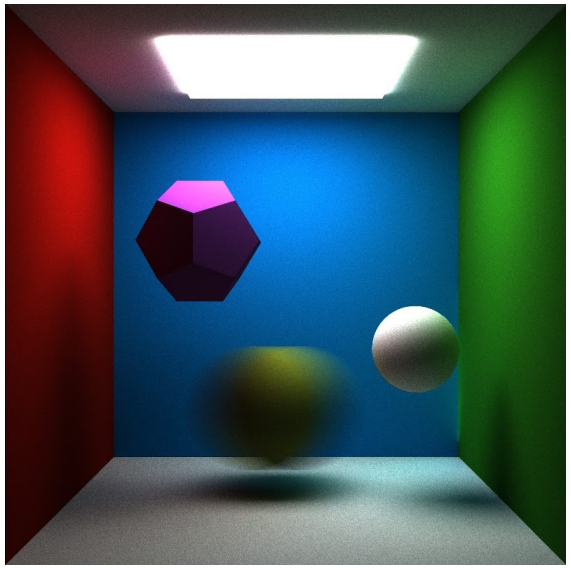
- Then, in order to better approximate the integral, we just need to generate (and average) more *paths*
  - Shoot more rays through each pixel, each one creating a new path that contributes (potentially) to the pixel's final color
- We can get anti-aliasing and other distribution effects "for free" when doing this pixel sampling

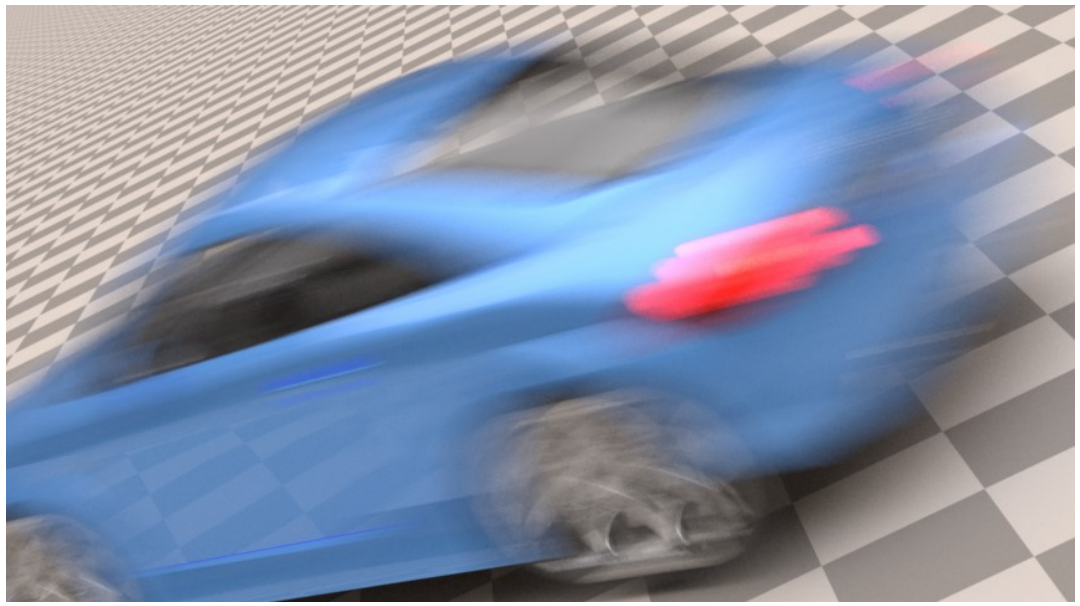
Distribution Effects











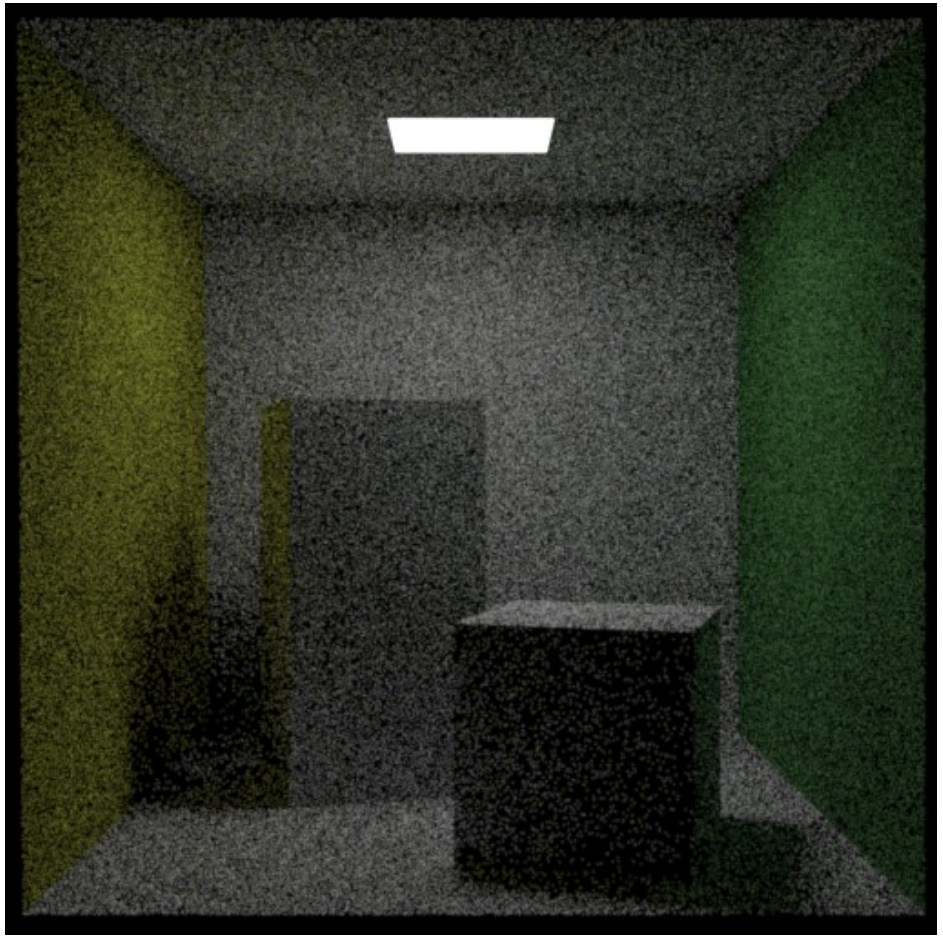
How do we stop?

Implicit Path-tracing: noisy results

$$L(x,\omega_o) \approx \frac{1}{N}\sum_{N}\left[L_e(x,\omega_o) + \frac{L(x',\omega)\, f_r(x,\omega,\omega_o)\cos\theta}{pdf(\omega)}\right]$$

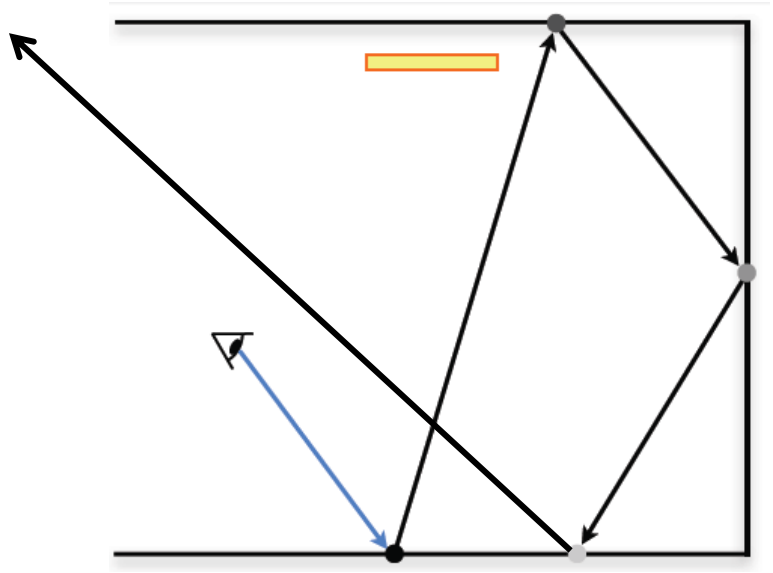
```
color shade( point x )
  if( x.Le != 0 ) return x.Le;
  color L = 0;
  omega_i = random direction
            sampled from pdf;
  L += shade( trace(x, omega_i) )
       * brdf * cos / pdf(omega_i);
  return L;


...
color c = 0;
for( each pixel sample s)
  c += shade(trace(eye, s))
c /= N;
...
```



Implicit Path-tracing: noisy results

- Why is there so much noise?
- The recursion terminates (with a non-zero value) only* if (the last vertex of) the path hits a light!
  - Tons of wasted paths!



Split the Integral + Specialize

- By separately sampling the direct- and indirect-contributions of the rendering equation (with different strategies), we can reduce variance

$$L = L_e(x, \omega_o) + L_d(x, \omega_o) + L_{ind}(x, \omega_o)$$

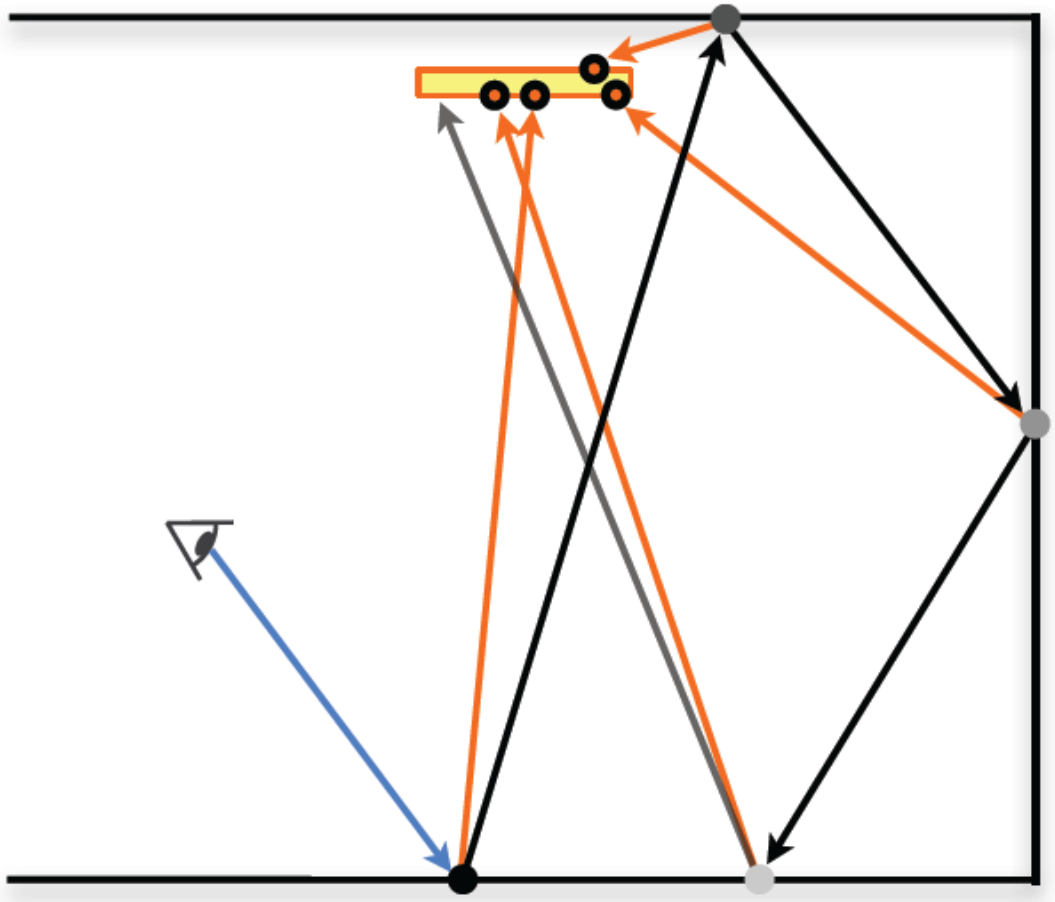
- Here, we'll still use a 1-sample estimator for each term
  - Direct lighting will be *explicitly* sampled
    - Need to be careful not to double count light…

$$\approx L_e + \frac{L_e(x_d, \omega_d) f_r(x, \omega_i, \omega_d) V(x, \omega_d) \cos\theta_d \cos\theta_{x\to d}}{d_{x\to d}^2 \, pdf(x_d)}$$

$$+ \frac{L(x', \omega) \, f_r(x, \omega, \omega_o) \cos\theta}{pdf(\omega)}$$

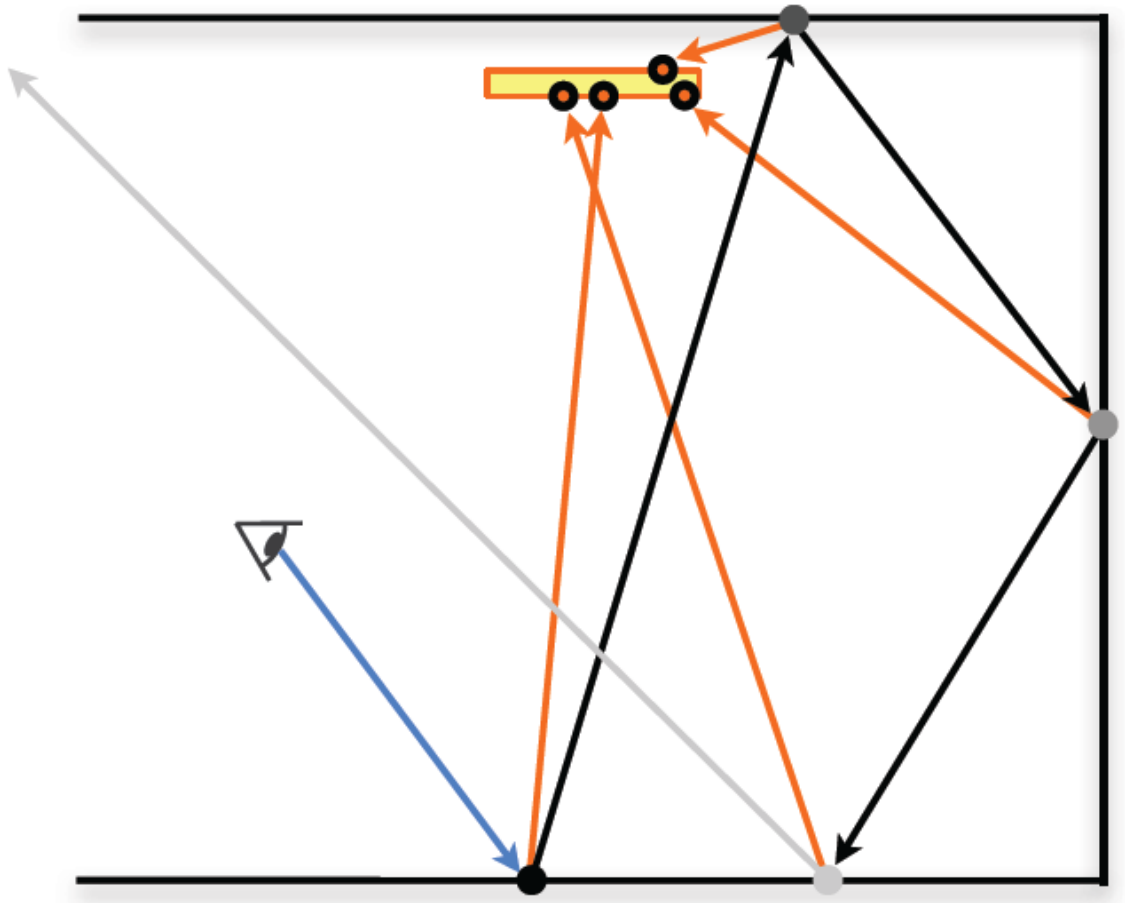
Explicit Path-tracing

(a.k.a. path-tracing with explicit direct lighting)

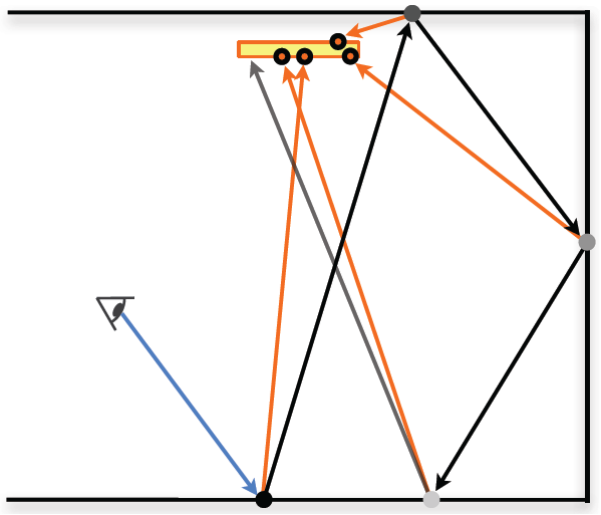


Explicit Path-tracing

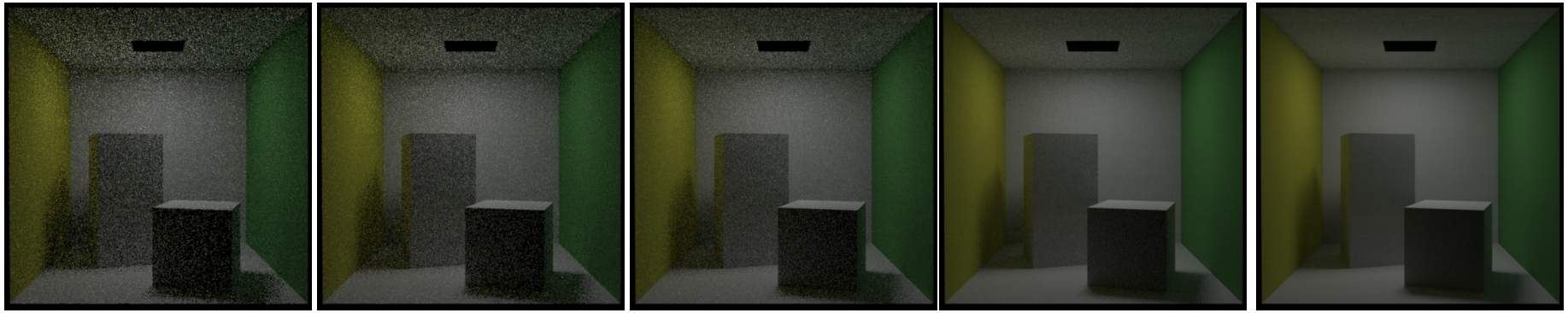
(a.k.a. path-tracing with explicit direct lighting)



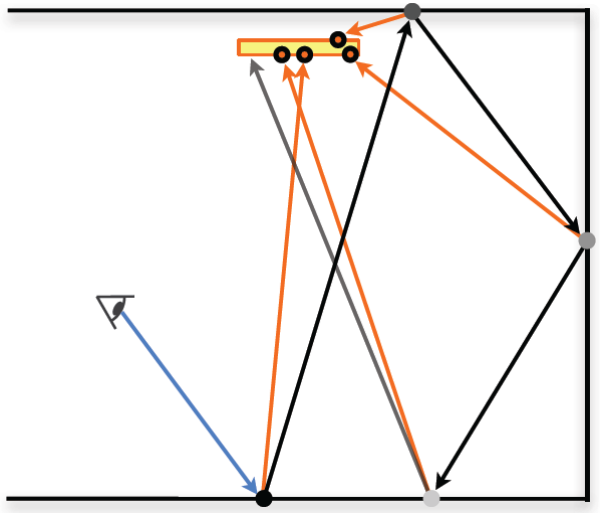
Explicit Path-tracing



```
color shade( point x )
  color L = x.Le;
  point p = random point on the light;
  omega_i = random direction from pdf;
  // L_d
  if( trace(x, p) )
    L += fr * cos * p.Le * cos'/(|x-p|^2 * pdf_x)
  // L_ind
  L += shade(trace(x,omega_i)) * fr' * cos' / pdf;
  return L;
```



Infinite Recursion

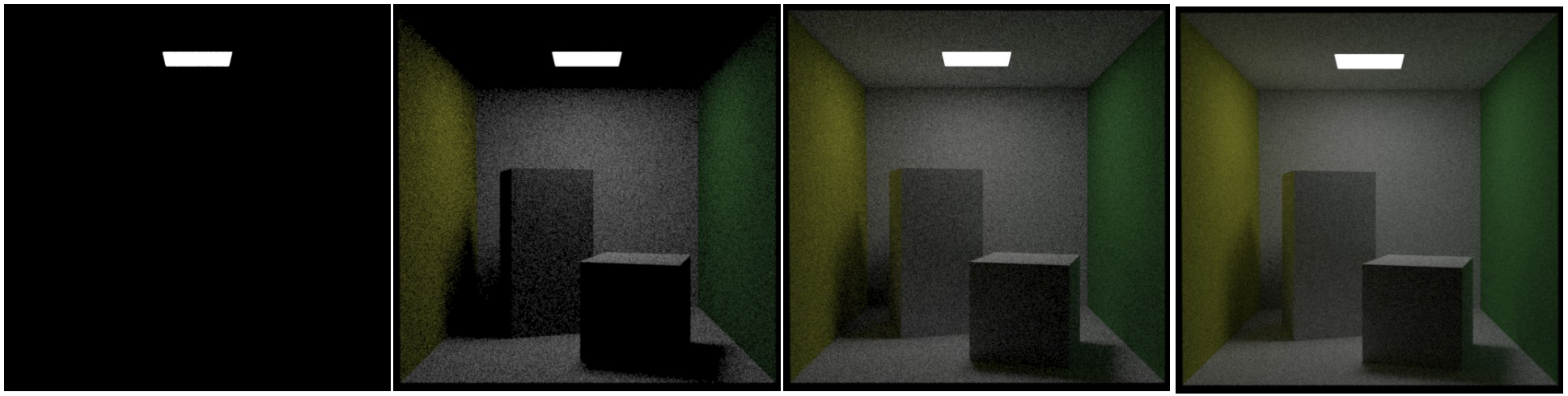


```
color shade( point x )
  color L = x.Le;
  point p = random point on the light;
  omega_i = random direction from pdf;
  // L_d
  if( trace(x, p) )
    L += fr * cos * p.Le * cos'/(|x-p|^2 * pdf_x)
  // L_ind
  L += shade(trace(x,omega_i)) * fr' * cos' / pdf;
  return L;
```

There's still a (potentially) infinite recursion!

How to terminate the recursion?

- When/how do we stop the recursion?
  - Clamp with a maximum path length (recursion depth)?
    - This *biases* our result



p=1 p=2 p=3 p=4

  - Are there unbiased ways to terminate recursion?

Russian Roulette

- Russian roulette is a probabilistic approach to terminate MC processes without changing their expected value / average (i.e., without biasing the result)

1. Pick some threshold / termination-probability χ
2. Generate a random number ε between (0,1]
3. Return $V' = \frac{V}{1-\chi}$ for a sample when $\varepsilon \leq \chi$ or 0 otherwise

$$E[V'] = (1-\chi)\left[\frac{E[V]}{1-\chi}\right] = E[V]$$

Russian Roulette

- To apply RR to path-tracing, how do we choose χ?
  - Note, as with the *pdf*, no matter your choice the estimator will remain "correct"
    - But, as with the *pdf*, a "smarter" choice will improve convergence and reduce variance
  - e.g., for diffuse f_r, a typical choice is $\chi = \rho_x$
    - Interpretation: the higher the reflectance ρ_x, the higher the surface's reflection, and so we would like to promote such reflections with higher probability

Explicit Path Tracing w/ Russian Roulette

- So, our final path tracing algorithm is:

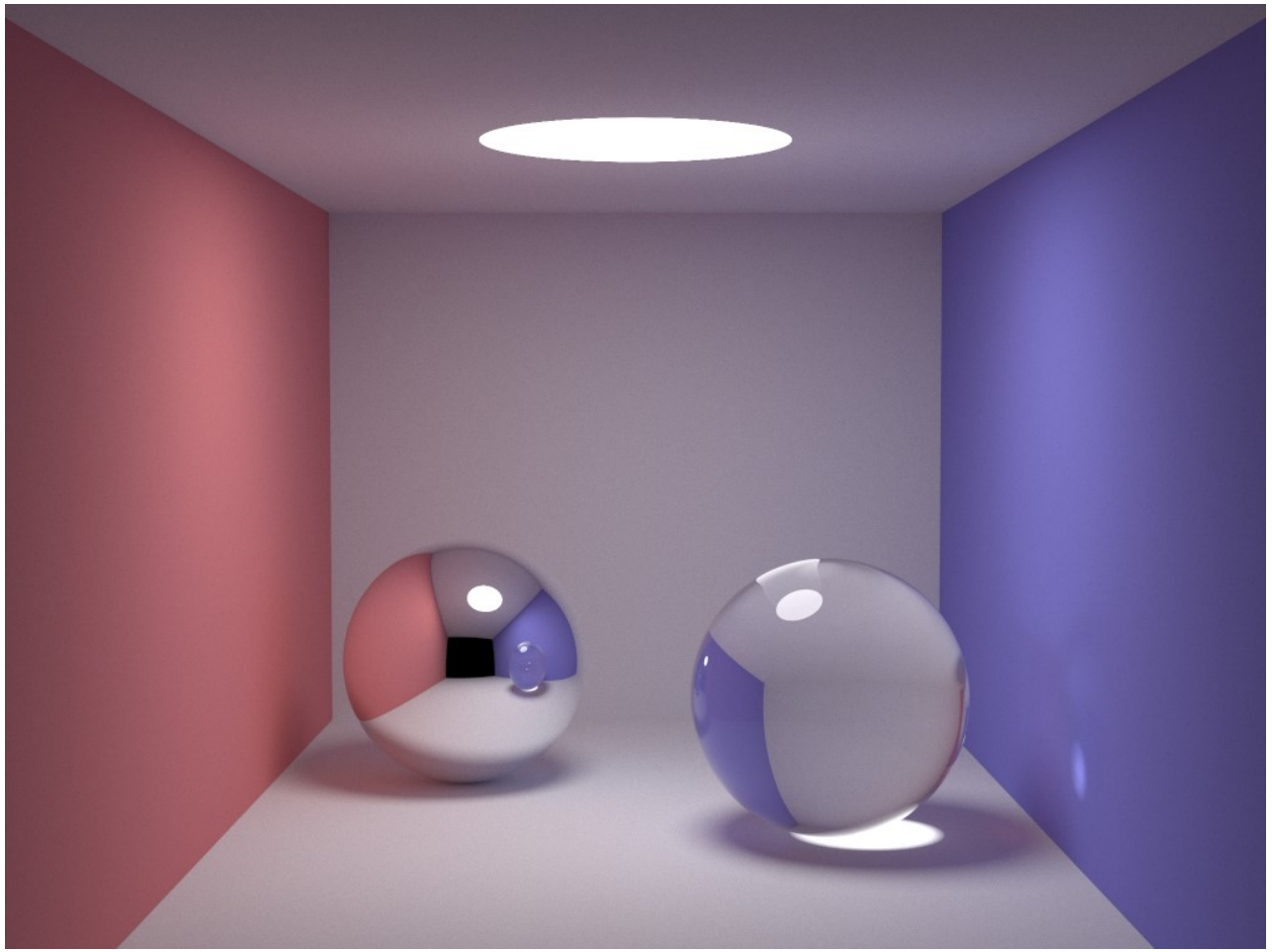
```
color shade( point x )
  // L_d
  point p = random point on the light;
  if( trace(x, p) )
    L += fr * cos * p.Le * cos'/(|x-p|^2 * pdf_x)

  // L_ind
  if( rand() < X )
    omega_i = random direction taken from pdf;
    L += shade(trace(x,omega_i)) * fr' * cos' / (pdf * (1-X));

  return L;
```

- With a basic ray tracer in place, we can code a basic *path tracer* in about 50 (very careful) lines of code!

Explicit Path Tracing w/ Russian Roulette



http://www.kevinbeason.com/smallpt/

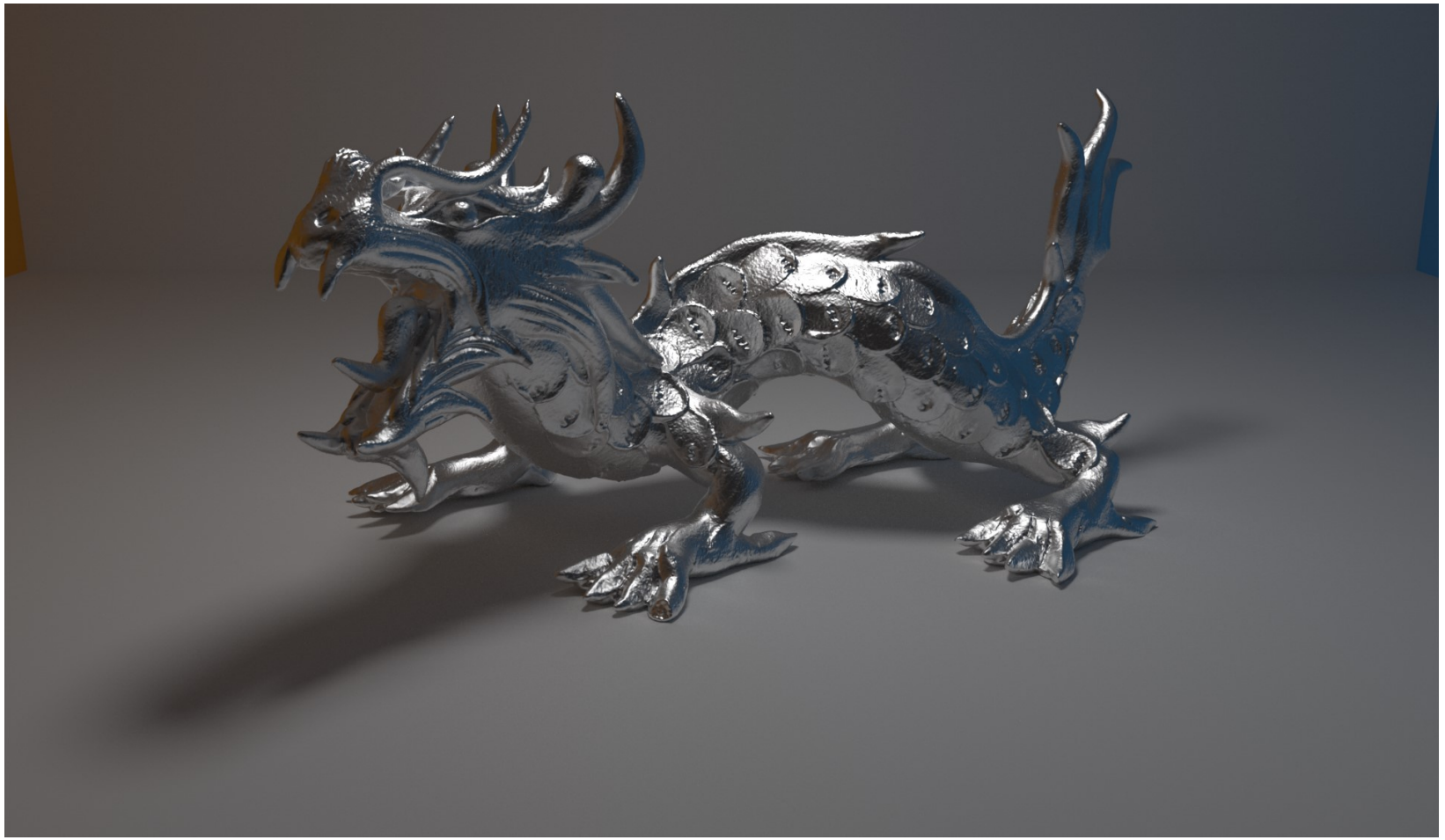
99 lines of C++

Problems with Path Tracing

- Advantages:
  - Easy to implement!
  - Converges to the right answer!

Problems with Path Tracing

– Converges to the right answer



Problems with Path Tracing

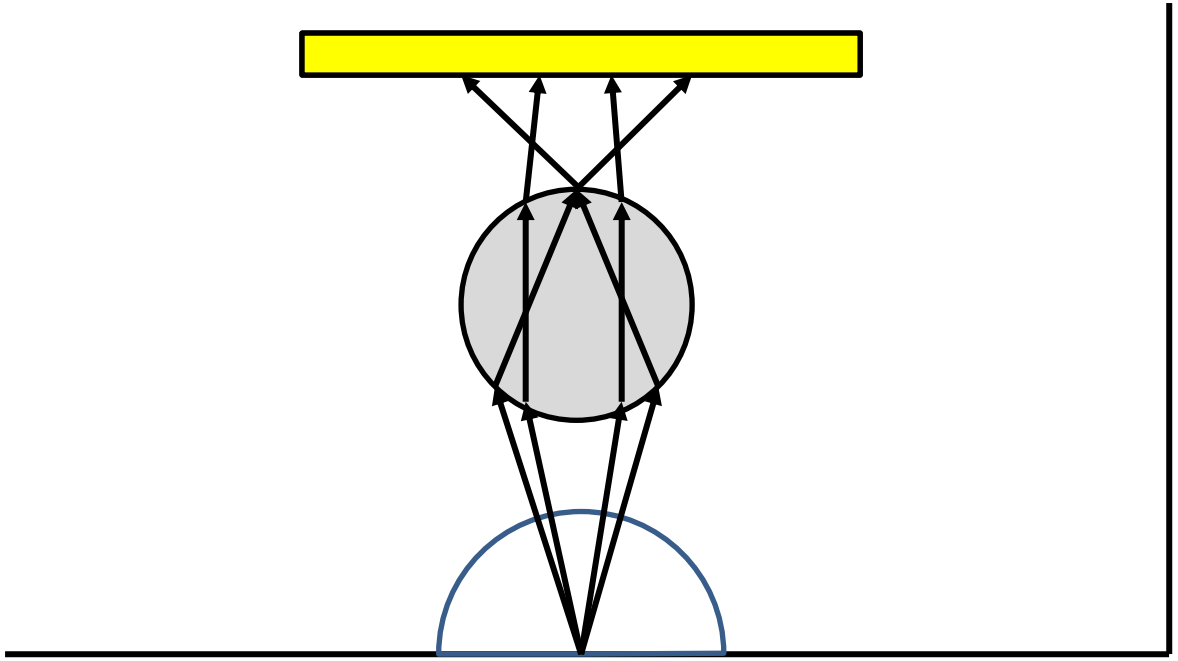
- Disadvantages:
  - S-L-O-W and noisy convergence

Problems with Path Tracing

– S-L-O-W and noisy convergence

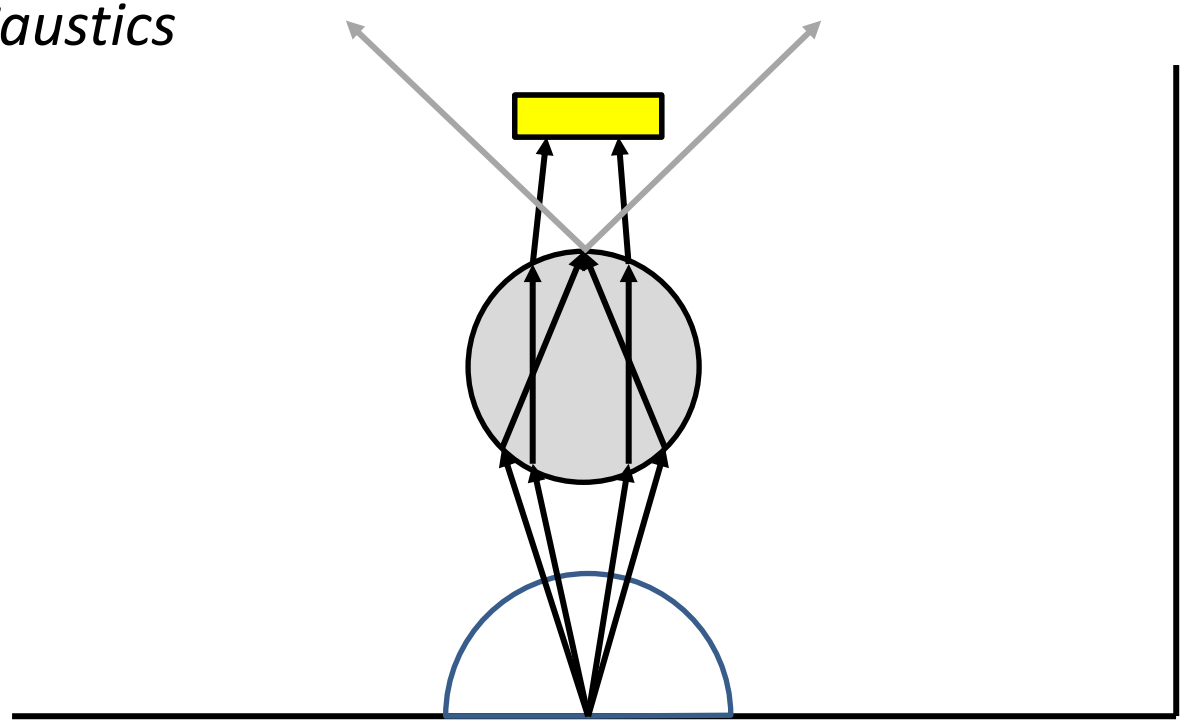
Problems with Path Tracing

- Disadvantages:
  - Difficulty handling complex illumination paths:
    - *Caustics* (L S* D E)



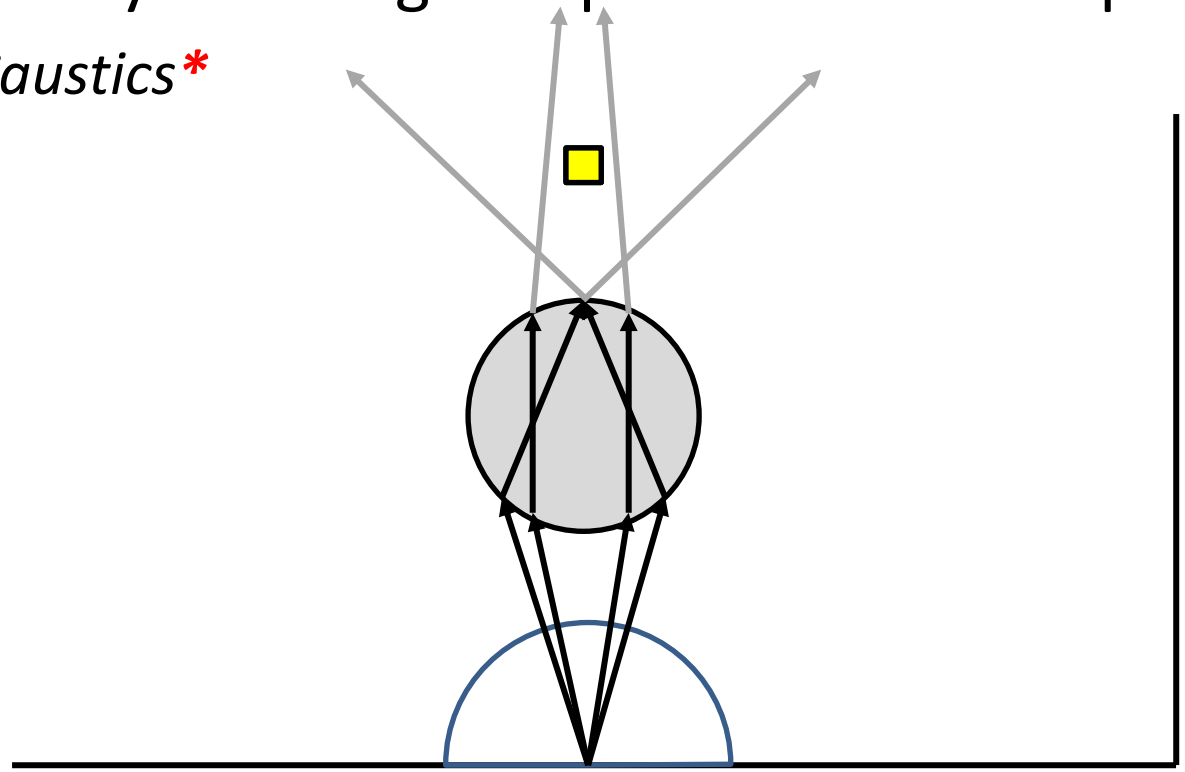
Problems with Path Tracing

- Disadvantages:
  - Difficulty handling complex illumination paths:
    - *Caustics*



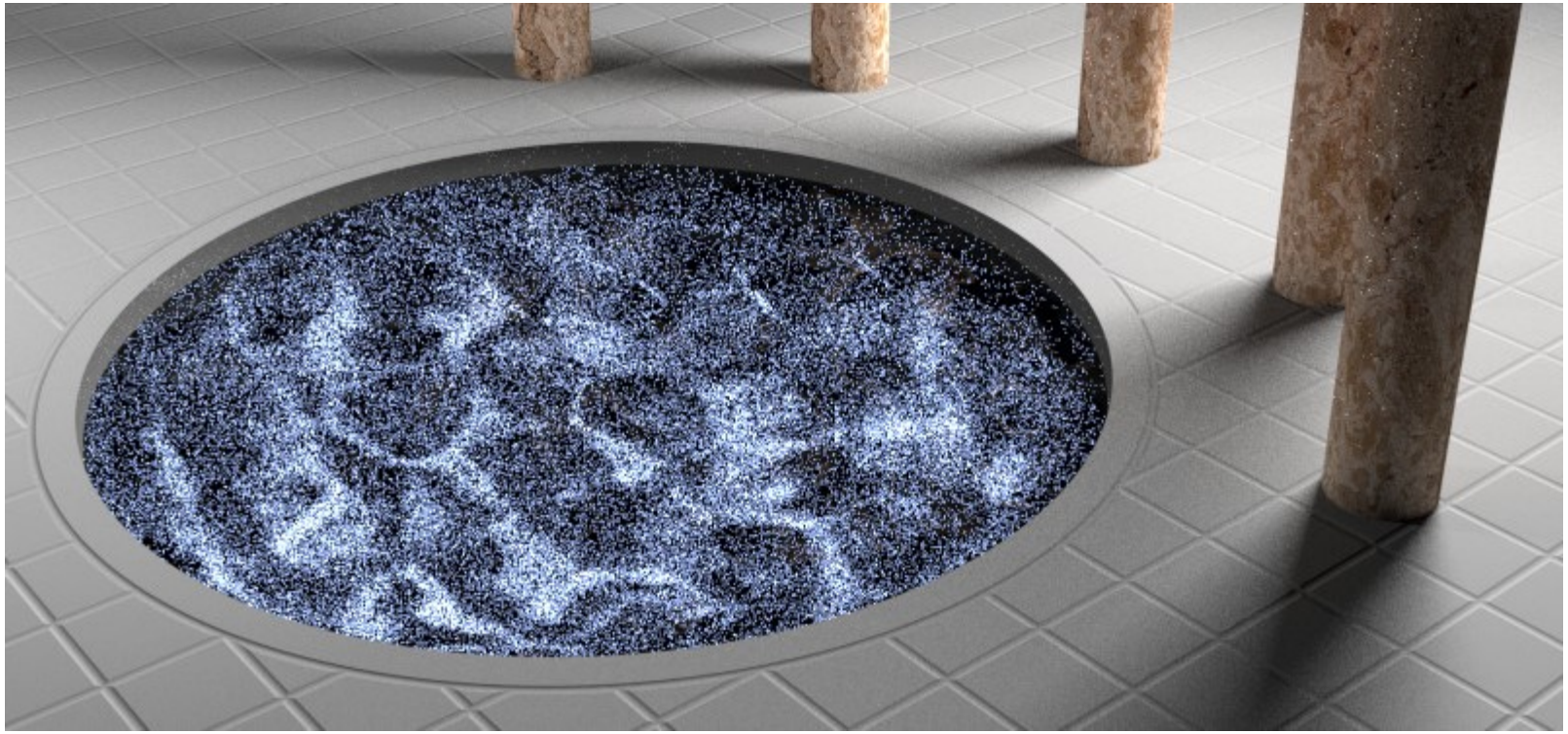
Problems with Path Tracing

- Disadvantages:
  - Difficulty handling complex illumination paths:
    - *Caustics**



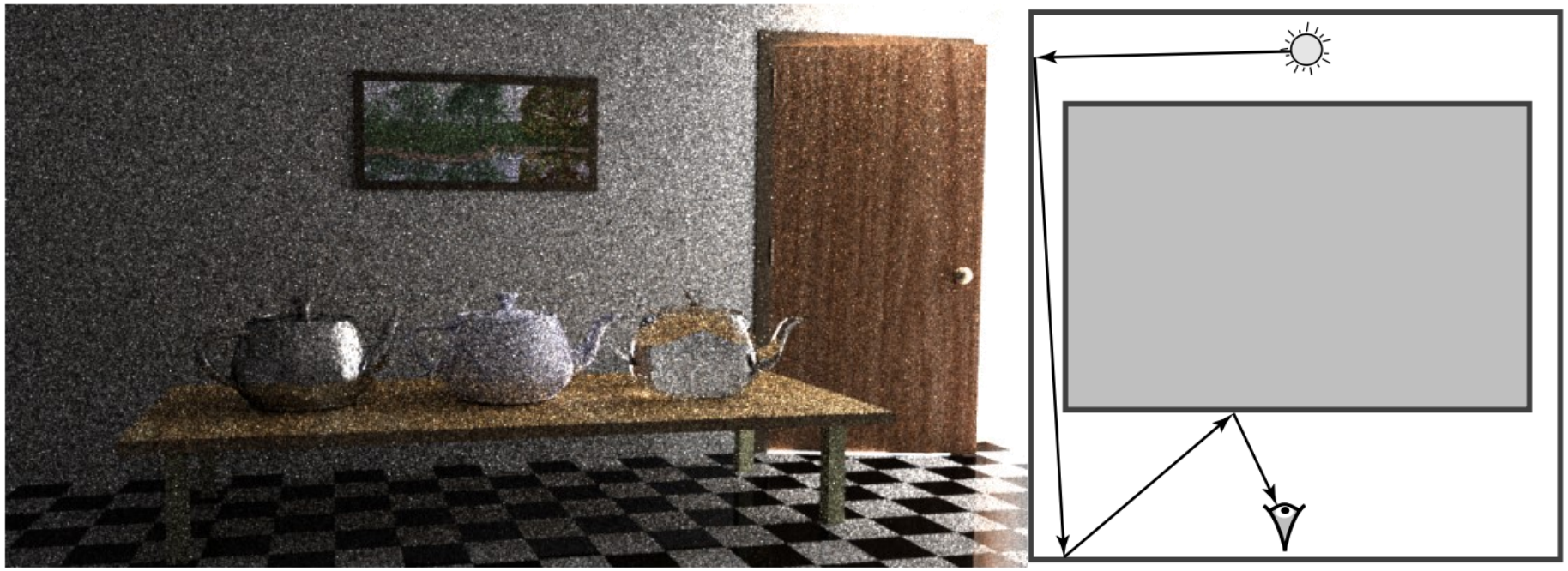
Problems with Path Tracing

- Disadvantages:
  - Difficulty handling complex illumination paths:
    - *Caustics*



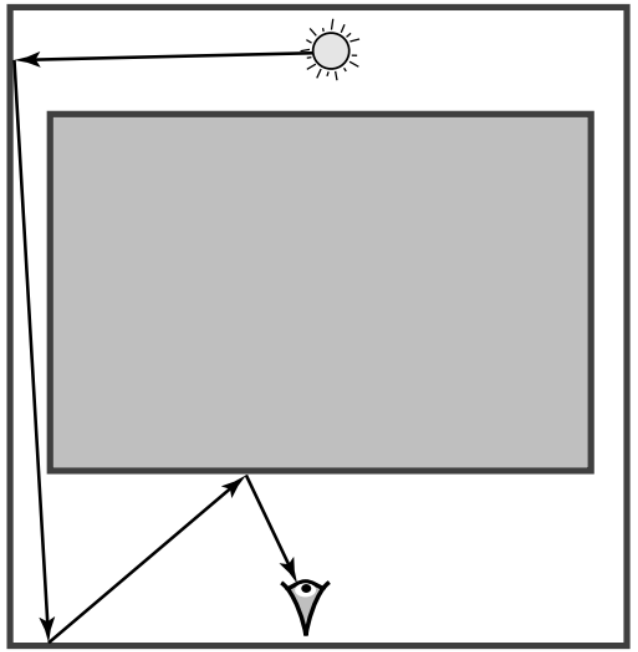
Problems with Path Tracing

- Disadvantages:
  - Difficulty handling complex illumination paths:
    - Lighting contributions primarily due to indirect light



Problems with Path Tracing

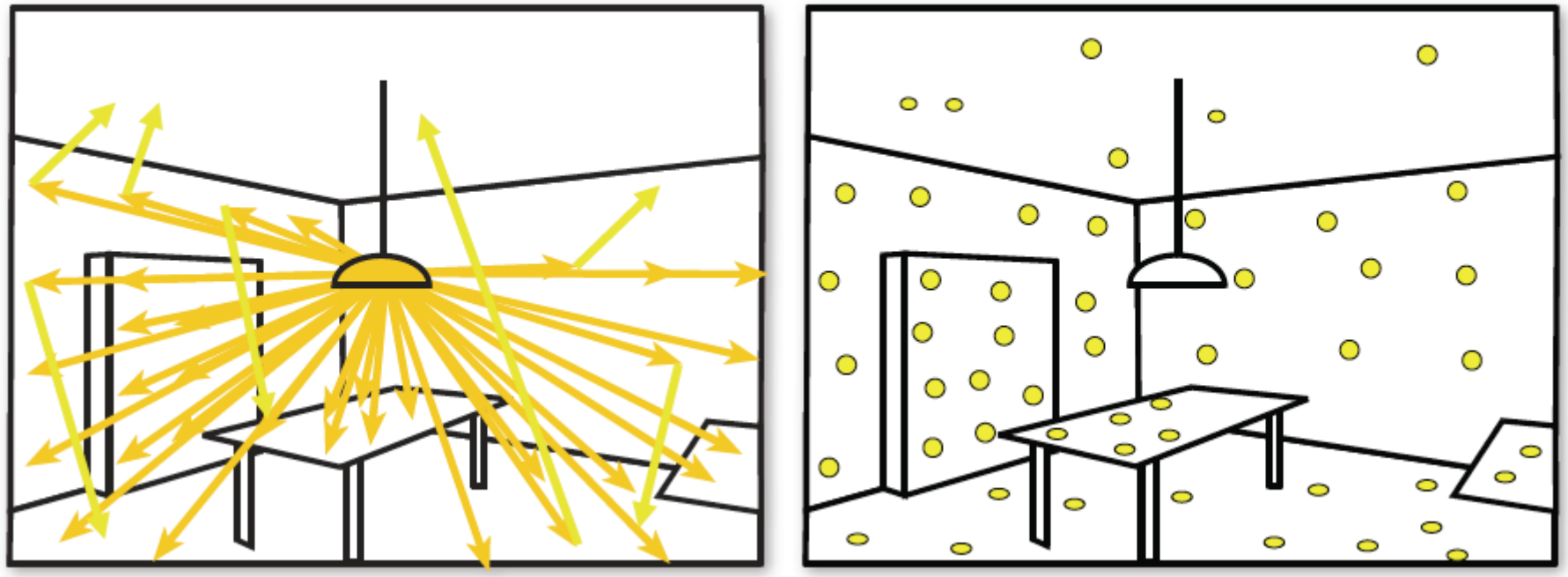
- Disadvantages:
  - Difficulty handling complex illumination paths:
    - Lighting contributions primarily due to indirect light



Photon Mapping

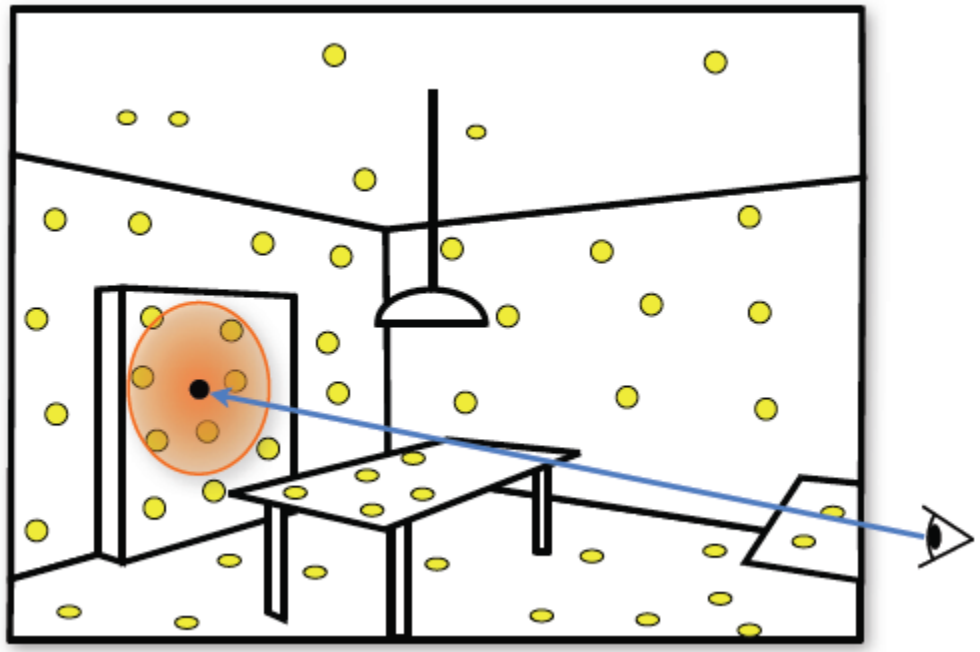
Photon Mapping

- Basic Photon Mapping is a two-step algorithm:
  1. Trace, reflect and deposit light particles (*photons*)



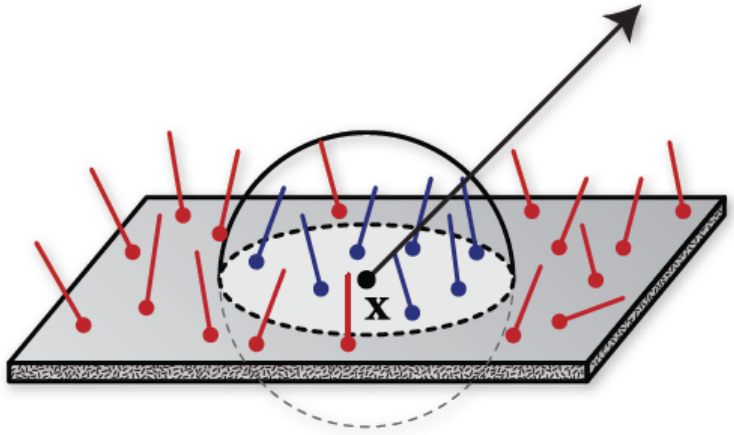
Photon Mapping

- Basic Photon Mapping is a two-step algorithm:
  1. Trace, reflect and deposit light particles (*photons*)
  2. Estimate light particle density to render image



Photon Mapping

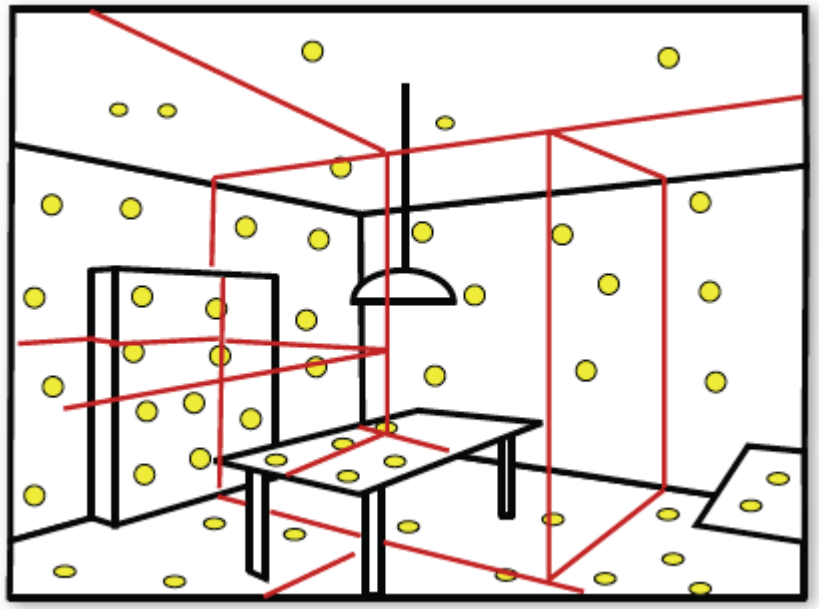
2. Estimate light particle density to render image

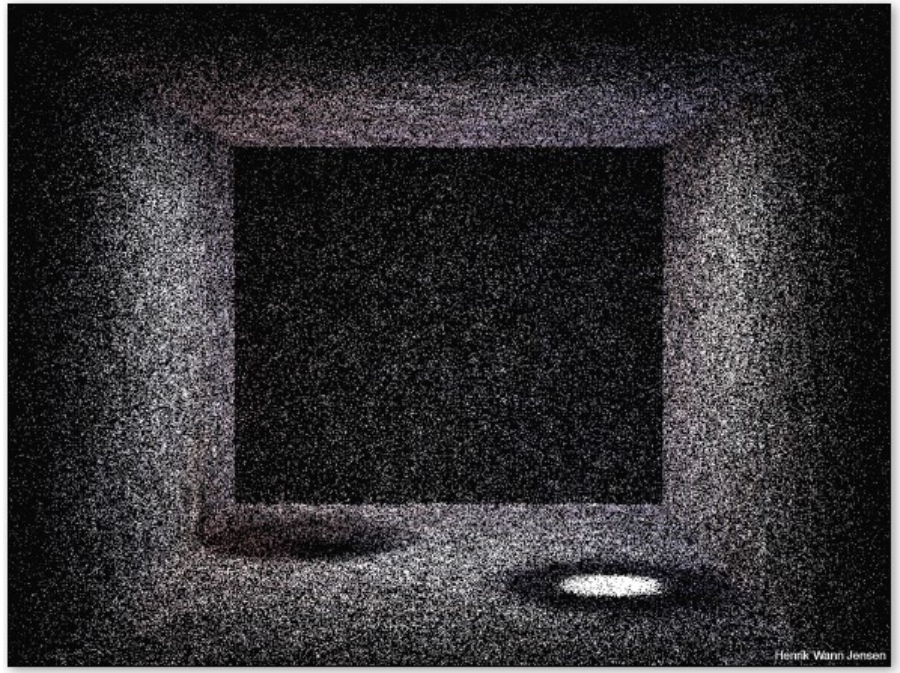


$$L(x, \omega_o) \approx \sum_{p=1}^{N} \frac{\Phi_p(x, \omega_p)}{\Delta A} f_r(x, \omega_p, \omega_o) \cos\theta_p$$

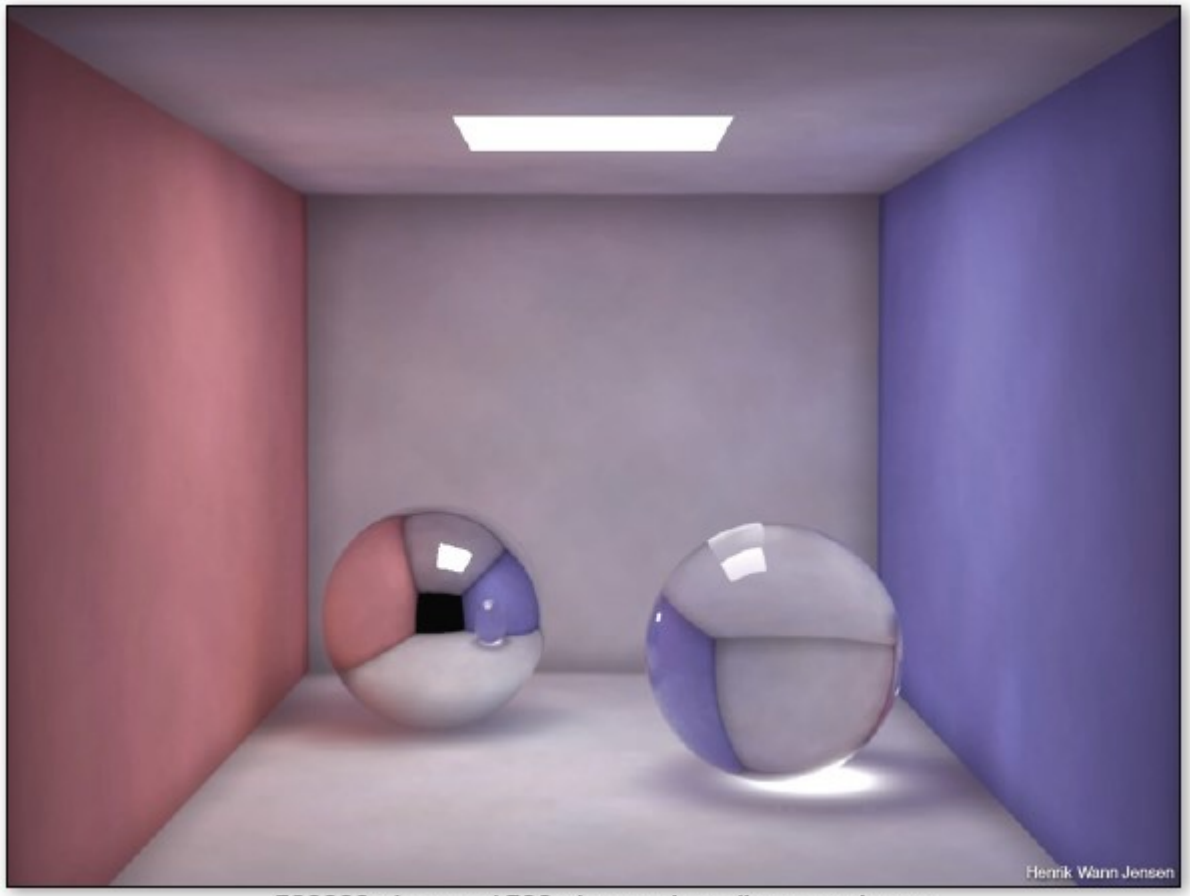
Photon Mapping

- We typically store the photons in a *kd-tree* in order to do quick lookups necessary for efficient density estimation





Photon Mapping



500000 photons / 500 photons in radiance estimate

Review and More Information

- Fundamentals of Computer Graphics
  - Section 24.4 Path Tracing (brief)
- Physically Based Rendering, from theory to implementation
  - http://www.pbr-book.org/3ed-2018/contents.html
  - https://www.pbrt.org/
  - Chapter 13 covers Monte Carlo Integration
  - Chapter 14 includes details on Path tracking (section 14.5)