Rendering Algorithms

Spring 2014

Matthias Zwicker

Universität Bern

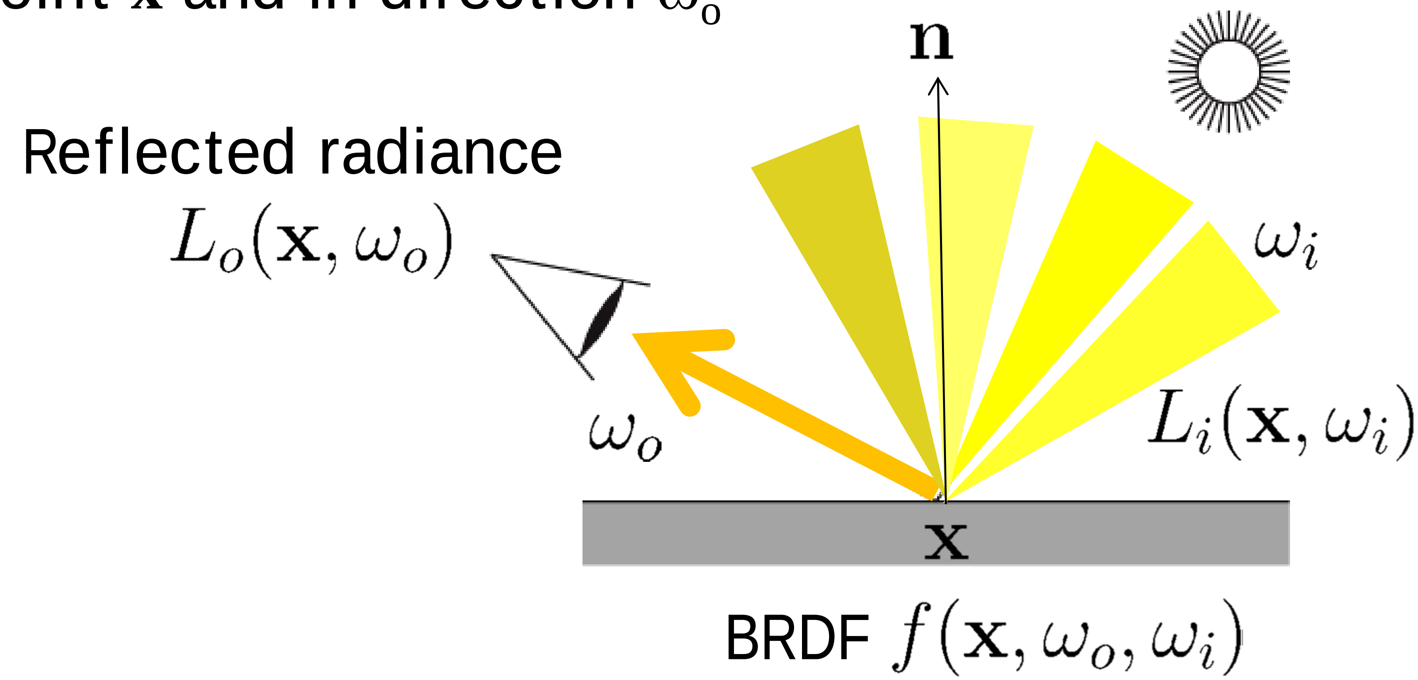
Today

Global illumination

- The Rendering Equation
- Monte Carlo path tracing

So far: reflection equation

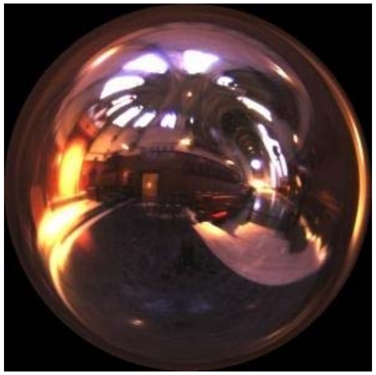
- Given incident light L_i over hemisphere $H^2(\mathbf{n})$ and BRDF f at point $\mathbf{x}$, what is reflected light L_o
- L_o is a radiance distribution: reflected radiance at each point $\mathbf{x}$ and in direction ω_o



$$L_o(\mathbf{x}, \omega_o) = \int_{\mathcal{H}^2(\mathbf{n})} f(\mathbf{x}, \omega_o, \omega_i) L_i(\mathbf{x}, \omega_i) \cos\theta_i d\omega_i$$

So far: reflection equation

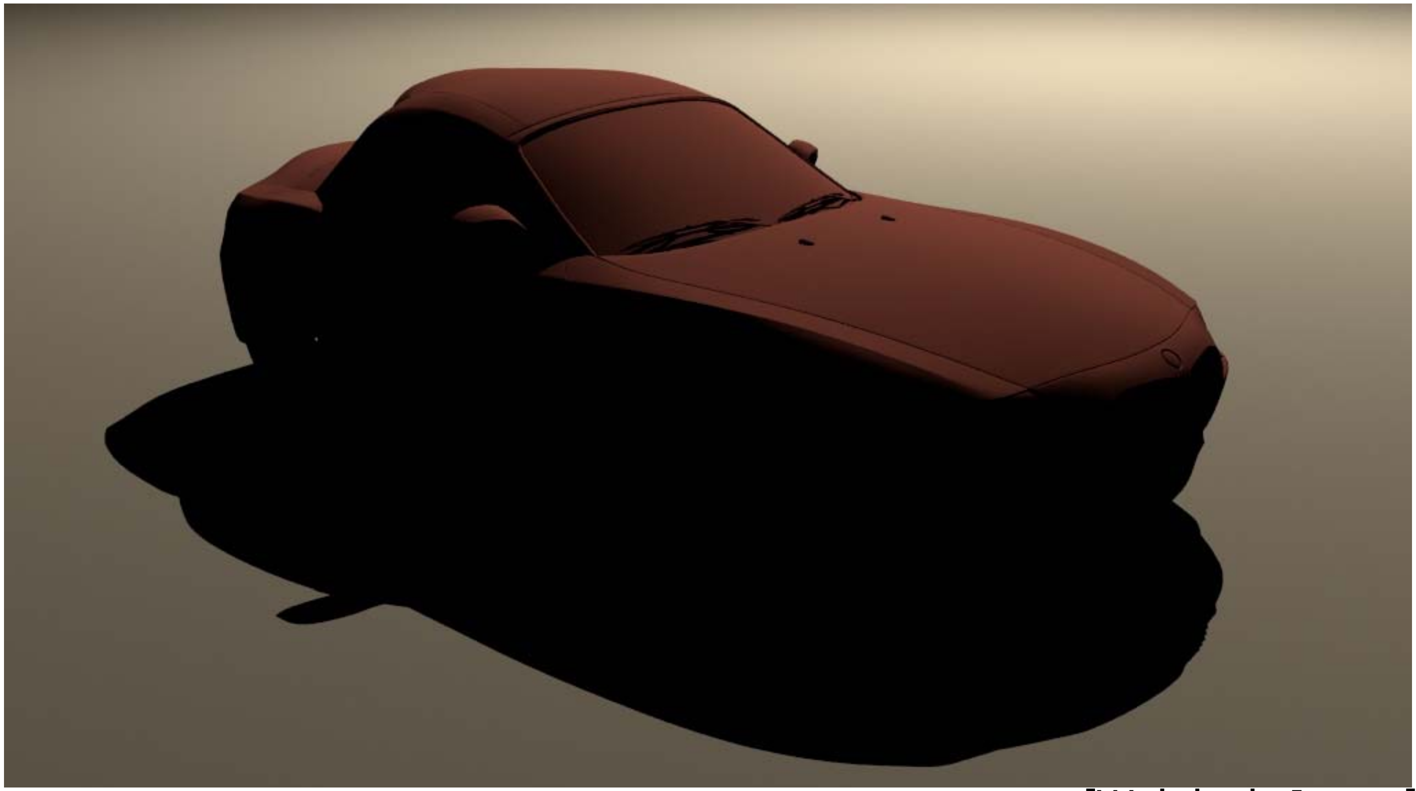
- For example, incident radiance given by environment map or known light sources



- Evaluating the reflection equation using Monte Carlo integration

So far: reflection equation

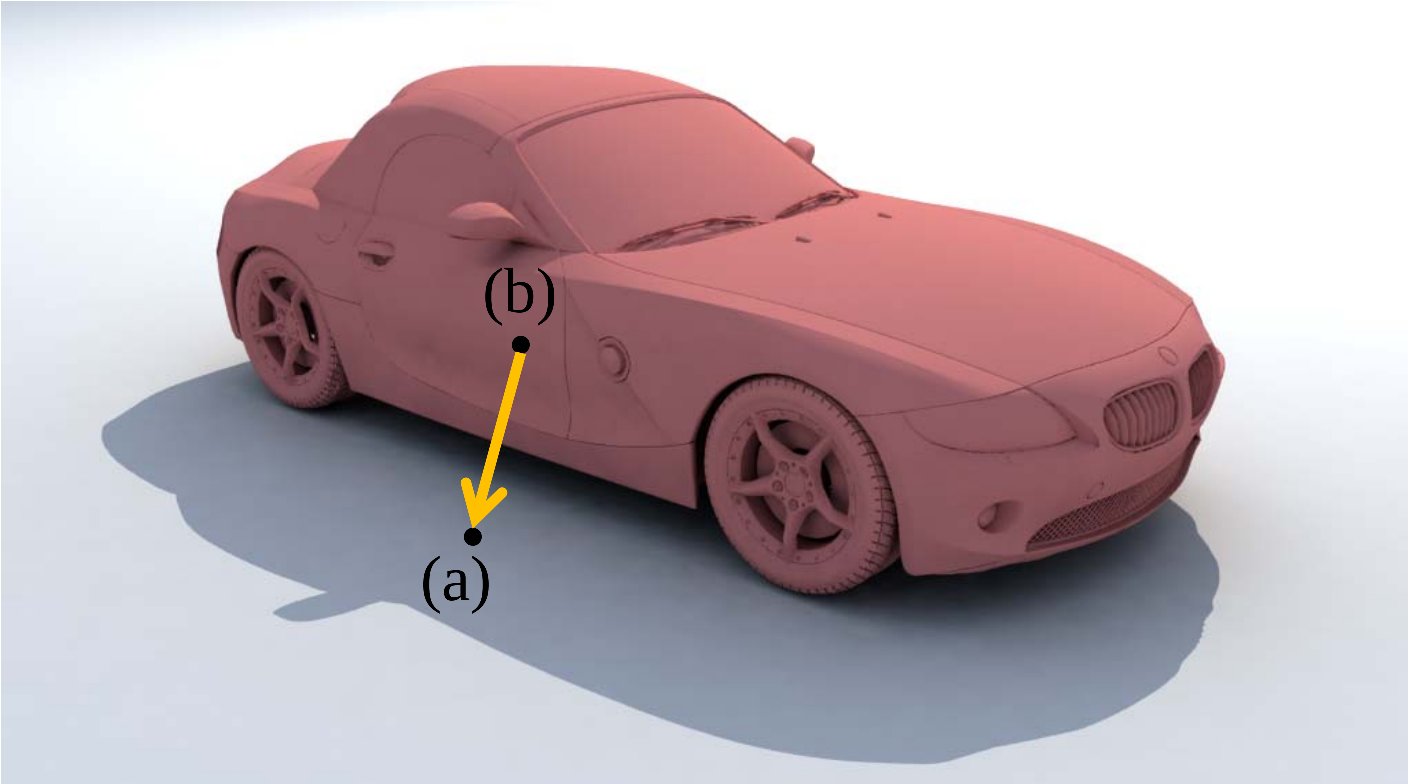
- Direct illumination from area and point lights



[Wojciech Jarosz]

Global illumination

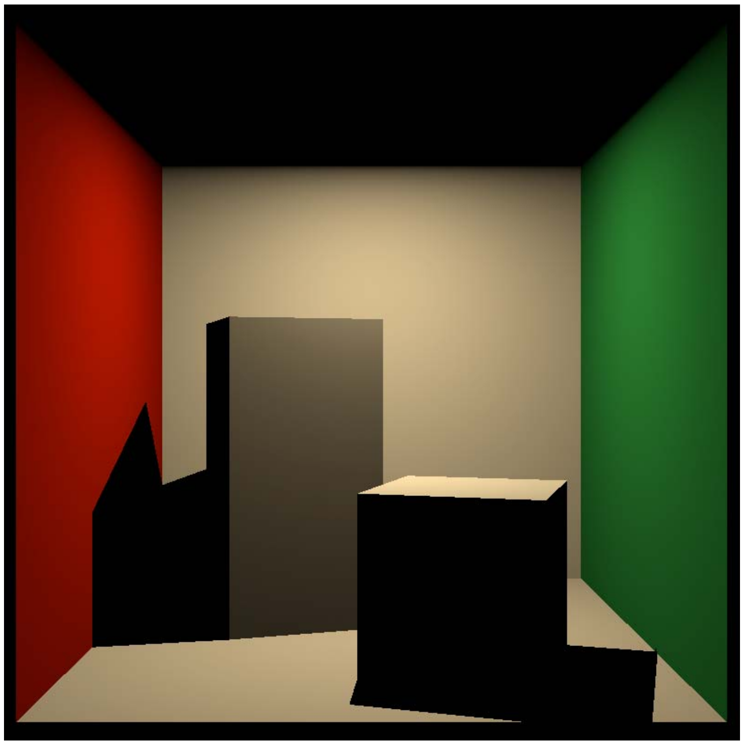
- Indirect illumination, „multiple bounces of light“
- Incident light at one point (a) depends on reflected light at other point (b)
  - Etc. etc. recursively



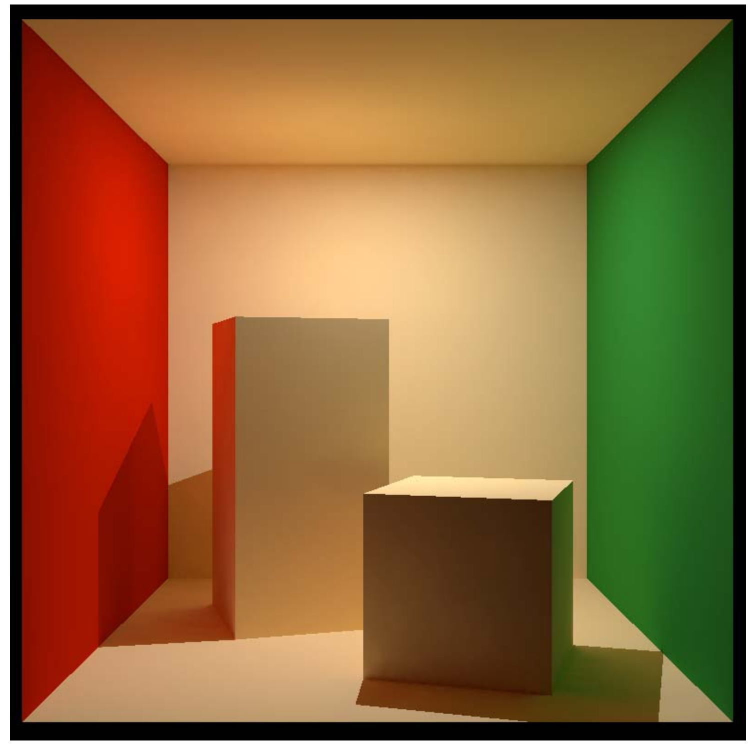
[Wojciech Jarosz]

Global illumination

Direct only



Direct & indirect



Real photograph

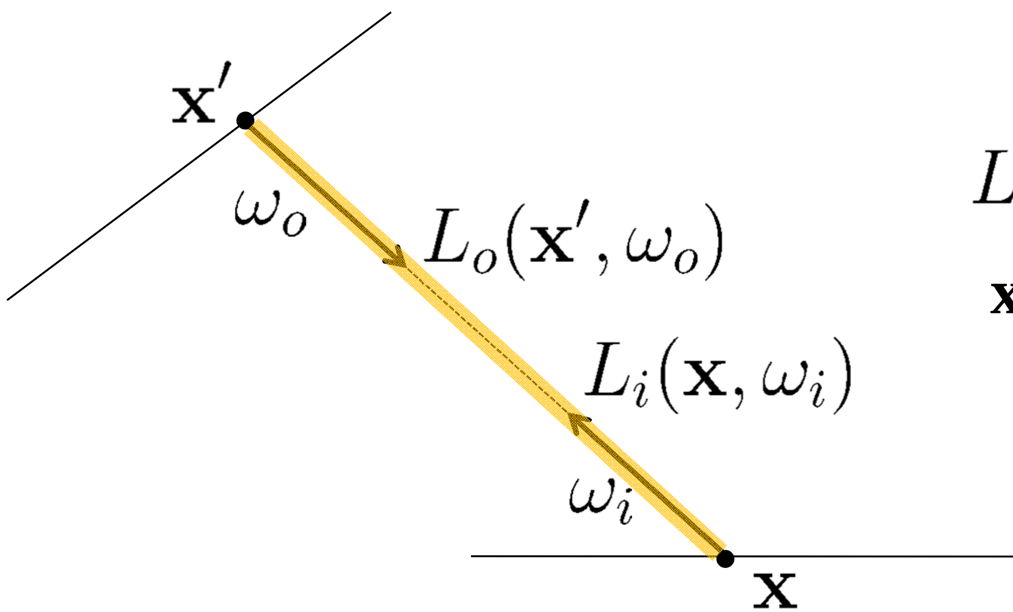
Global illumination

- How to represent idea of „multiple bounces of light“ compactly in an equation?
- Think of both reflected and incident radiance as unknown
- Reflection equation expresses equilibrium between incident and reflected radiance

$$\underbrace{L_o(\mathbf{x}, \omega_o)}_{\text{reflected}} = \int_{\mathcal{H}^2(\mathbf{n})} f(\mathbf{x}, \omega_o, \omega_i) \underbrace{L_i(\mathbf{x}, \omega_i)}_{\text{incident}} \cos\theta_i d\omega_i$$

Trick with notation

- Radiance doesn‘t change along ray
- Get rid of distinction between incident L_i and reflected radiance L_o
- Will denote reflected radiance with L



$$L_i(\mathbf{x}, \omega_i) = L_o(\mathbf{x}', -\omega_i)$$

$\mathbf{x}$‘ is point where ray $\mathbf{x}, \omega_i$ hits surface

Light sources

- Light sources are represented by known function $L_e(\mathbf{x},\omega_o)$
- L_e is emitted radiance at each surface point $\mathbf{x}$ in each direction ω_o
- L_e is zero if point $\mathbf{x}$ is not on a light source

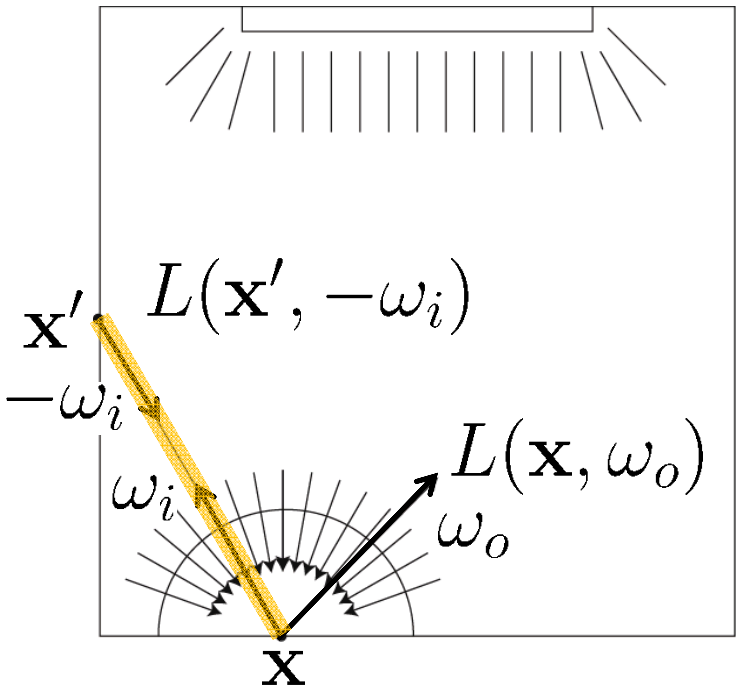
Rendering equation

http://en.wikipedia.org/wiki/Rendering_equation

$$L(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_{\mathcal{H}^2(\mathbf{n})} f(\mathbf{x}, \omega_o, \omega_i) L(\mathbf{x}', -\omega_i) \cos\theta_i d\omega_i$$

Reflected radiance appears as unknown on both sides of equation

Conservation of energy: outgoing light is sum of emitted light and reflected light, which is integral of incident light weighted by BRDF and cosine term



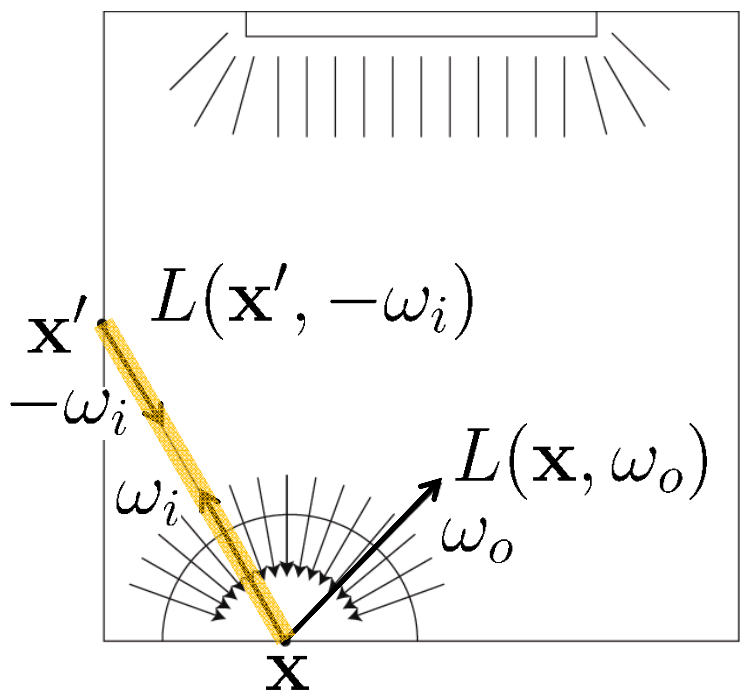
Rendering equation

http://en.wikipedia.org/wiki/Rendering_equation

$$L(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_{\mathcal{H}^2(\mathbf{n})} f(\mathbf{x}, \omega_o, \omega_i) L(\mathbf{x}', -\omega_i) \cos\theta_i d\omega_i$$

Reflected radiance appears as unknown on both sides of equation

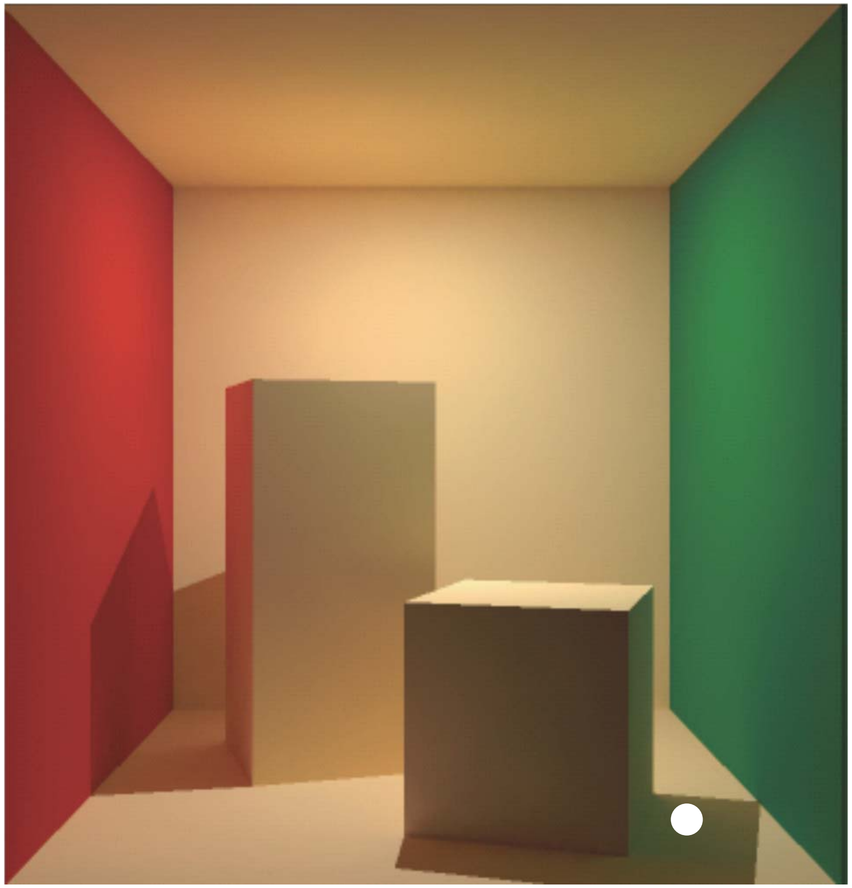
Solution: find radiance $L(x,\omega_o)$ such that reflection equation is satisfied simultaneously at each point x *and* direction ω_o



Rendering equation

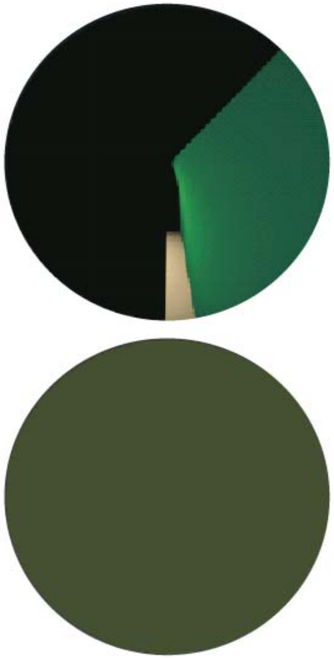
http://en.wikipedia.org/wiki/Rendering_equation

- Reflection equation satisfied at each point



[Wojciech Jarosz]

Incident radiance



Reflected radiance (on diffuse surface)

Note

- Rendering equation due to Jim Kajiya
  http://en.wikipedia.org/wiki/Rendering_equation
- Still the defining model for photo-realistic rendering today
- „Ultimately, all rendering algorithms try to (approximately) solve the rendering equation“
- Remember
  - Rendering equation based on approximate physical model (geometric optics)
  - Cannot model wave effects such as polarization, diffraction

Today

Global illumination

- The Rendering Equation
- Monte Carlo path tracing

Solving the rendering equation

Naive approach

- Recursive ray tracing as for mirror reflection, but shoot many rays in each step
  - Distribute rays over hemisphere at each step
- Problem: exponential explosion of number of rays
  - Exponentially more longer paths than shorter paths
  - But longer paths contribute less to image than shorter ones

Solving the rendering equation

- Similar idea, but smarter
- Intuition: Compute radiance $L(\mathbf{x}, \omega_o)$ as "integral over all light paths that connect light sources and eye"
- Paths have different lengths (number of bounces)
  - Can formulate one integral for all paths of each length
- Approach
  - Sum over all path lengths
    - Integral of radiance transported along all paths of given length (use Monte Carlo integration)

Mathematical formulation

- Rendering equation is Fredholm integral equation of second kind
  http://en.wikipedia.org/wiki/Fredholm_integral_equation
- Solution via series expansion
  - Neumann series
    http://en.wikipedia.org/wiki/Neumann_series
  - Liouville-Neumann series
    http://en.wikipedia.org/wiki/Liouville-Neumann_series

Rendering equation

(*) $L = E + \underbrace{\mathcal{T}(L)}$

$$\mathcal{T}(L)(x,\omega_o) = \int f(x,\omega_i,\omega_o)\, L(x',-\omega_i)\cos\theta_i\, d\omega_i$$

Transport operator, represents "one bounce of light"

Series expansion by recursive substitution

$L_0 = E$ | initial "guess", substitute into (*)

$L_1 = E + \mathcal{T}(L_0) = E + \mathcal{T}(E)$

$L_2 = E + \mathcal{T}(L_1) = \underbrace{E + \mathcal{T}(E + \mathcal{T}(E)) = E + \mathcal{T}(E) + \mathcal{T}^2(E)}_{\text{linearity of transport operator}}$

$\vdots$

$L_i = E + \mathcal{T}(E) + \mathcal{T}^2(E) + \ldots + \mathcal{T}^i(E)$

E: Light sources, path length 0

$\mathcal{T}(E)$: One bounce, path length 1 (direct illumination)

$\mathcal{T}^2(E)$: Two bounces, path length 2 (one bounce indirect illumination)

$$= \underline{\underline{\sum_{k=0}^{i} \mathcal{T}^k(E)}}$$

| if written out explicitly, $\mathcal{T}^k$ is a $2k$ dimensional integral

Convergence

Because of energy conservation (each bounce absorbs some light)

"$\gamma \|\mathcal{T}^k(E)\| = \|\mathcal{T}^{k+1}(E)\|$" with $\boxed{\gamma < 1}$

$\Rightarrow \|\mathcal{T}^\infty(E)\| = 0$

this implies

$$\boxed{L = \sum_{k=0}^{\infty} \mathcal{T}^k(E)}$$

is a converging geometric sum, and a solution to (*)!

Called Liouville-Neumann series

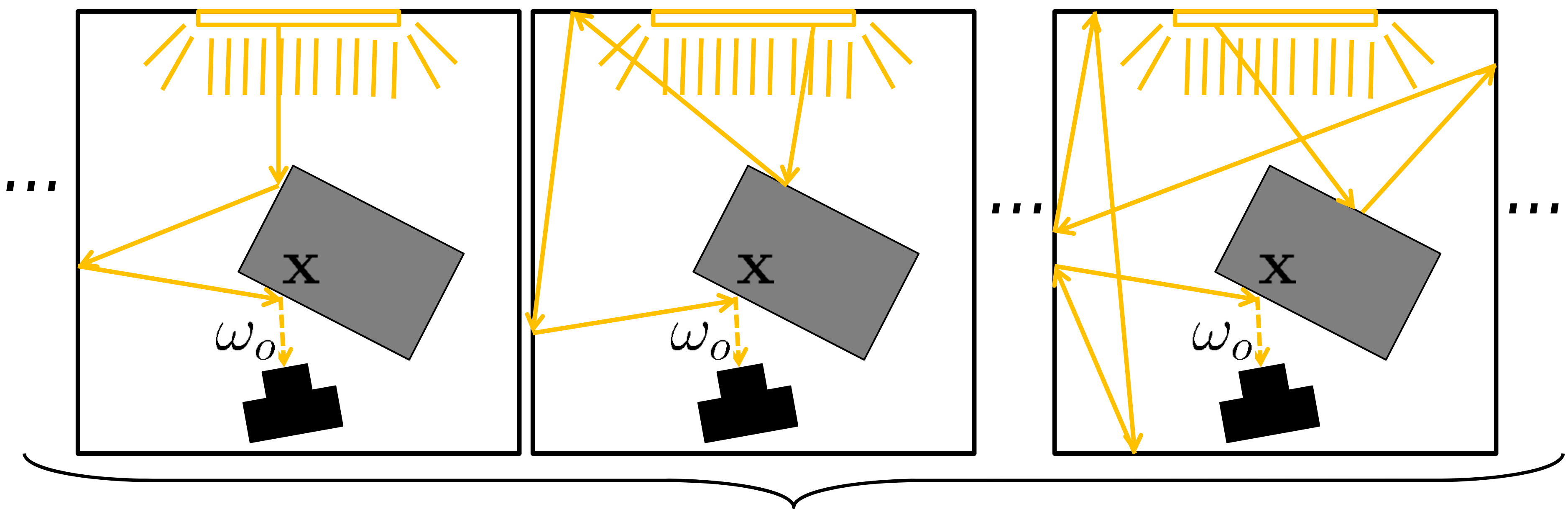
Monte Carlo path tracing

- Approximate integrals using Monte Carlo integration
  - Sample individual paths randomly
  - Estimate is simply sum over all paths
- Remember: each sample/path weighted with its inverse probability!
  - Need to keep track of sample/path probabilities

Path tracing

- Want to compute radiance through each pixel in image
  - Sum of radiance over all light paths connecting light and camera
- Light paths have different lengths

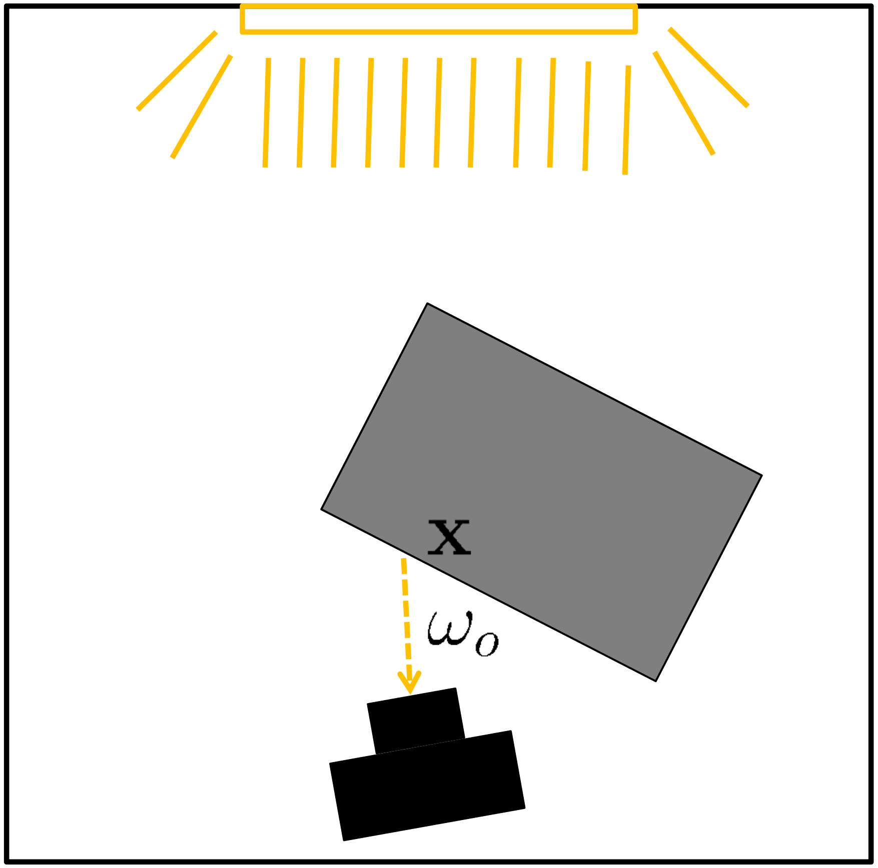
$$\sum \text{Length 3} + \sum \text{Length 4} + \ldots \sum \text{Length 7}$$



$$L(\mathbf{x}, \omega_o)$$

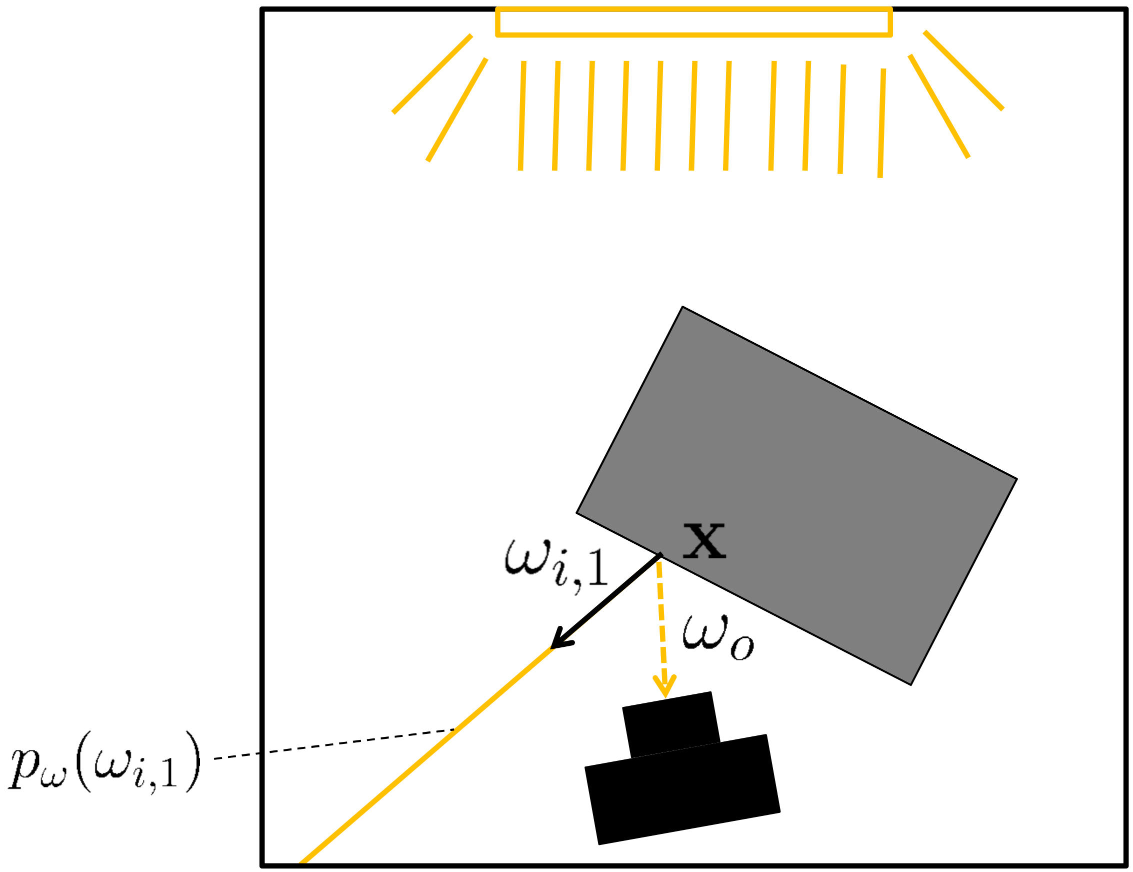
Construct random path

- Path of length k: after k-1 steps, sample light source (i.e., shoot "shadow ray")



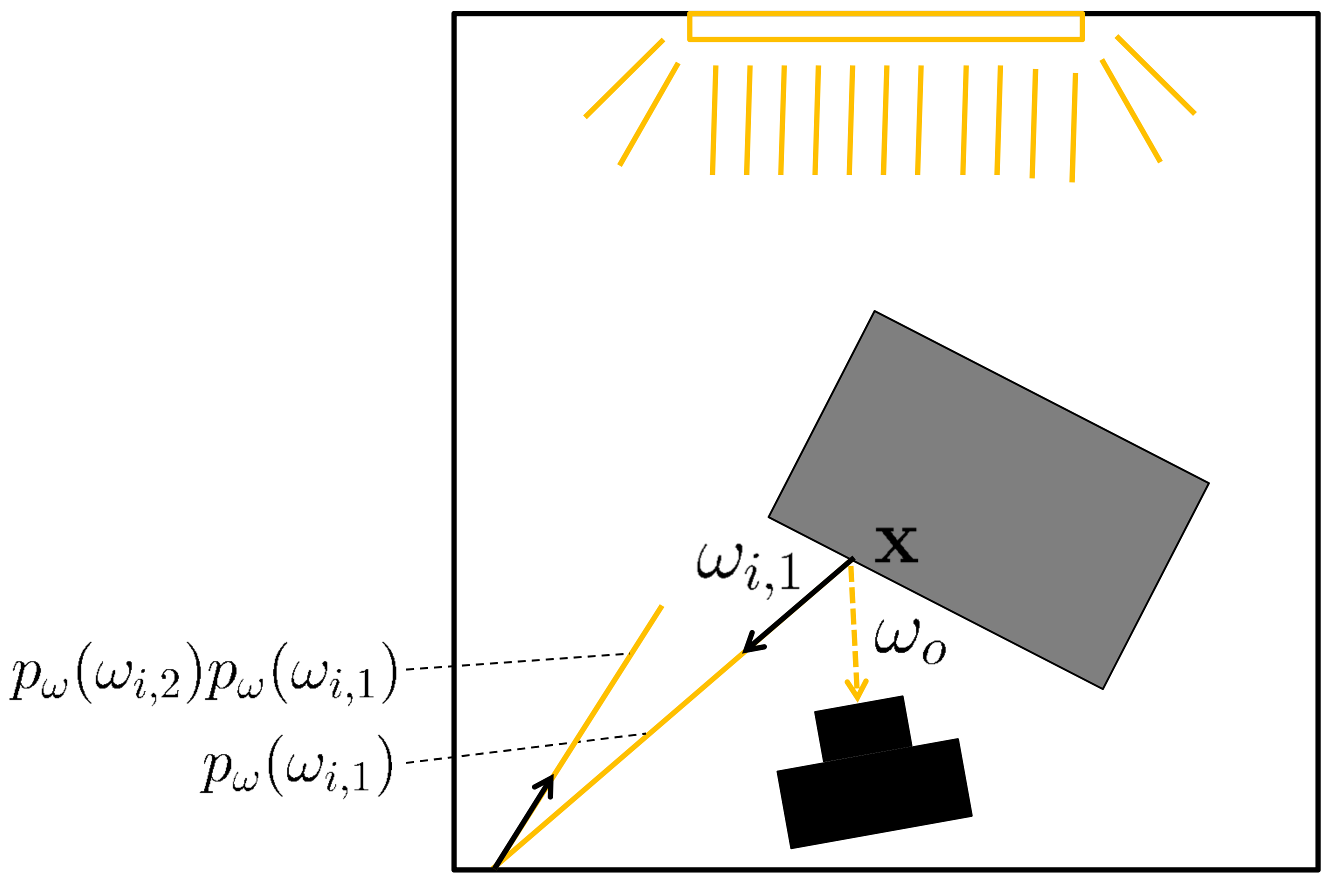
Construct random path

- Path of length k: after k-1 steps, sample light source (i.e., shoot "shadow ray")



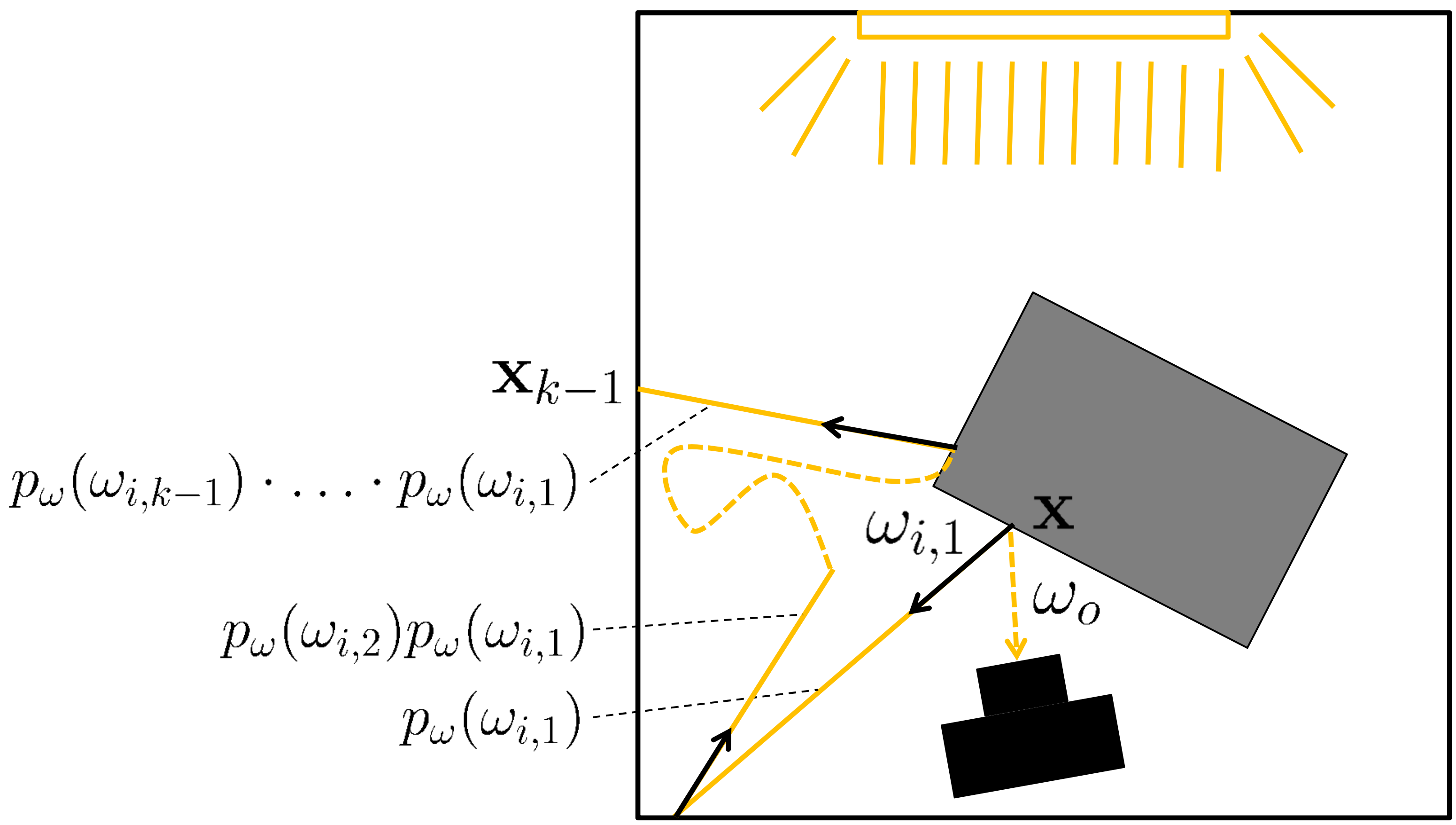
Construct random path

- Path of length k: after k-1 steps, sample light source (i.e., shoot "shadow ray")



Construct random path

- Path of length k: after k-1 steps, sample light source (i.e., shoot “shadow ray”)



Construct random path

- Path of length k: after k-1 steps, sample light source (i.e., shoot "shadow ray")

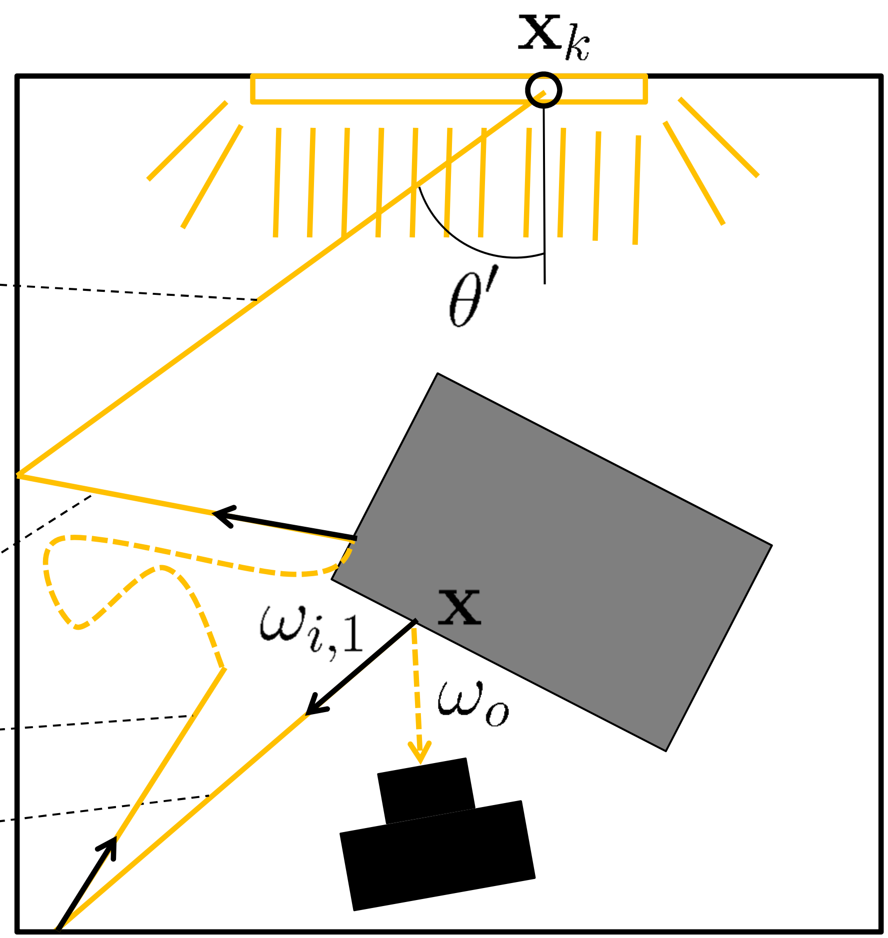
$$p_\omega(\mathbf{x}_k) = \frac{p_A(\mathbf{x}_k)\|\mathbf{x}_k - \mathbf{x}_{k-1}\|^2}{cos\theta'}$$

$$p_\omega(\mathbf{x}_k)p_\omega(\omega_{i,k-1}) \cdot \ldots \cdot p_\omega(\omega_{i,1})$$

$$p_\omega(\omega_{i,k-1}) \cdot \ldots \cdot p_\omega(\omega_{i,1})$$

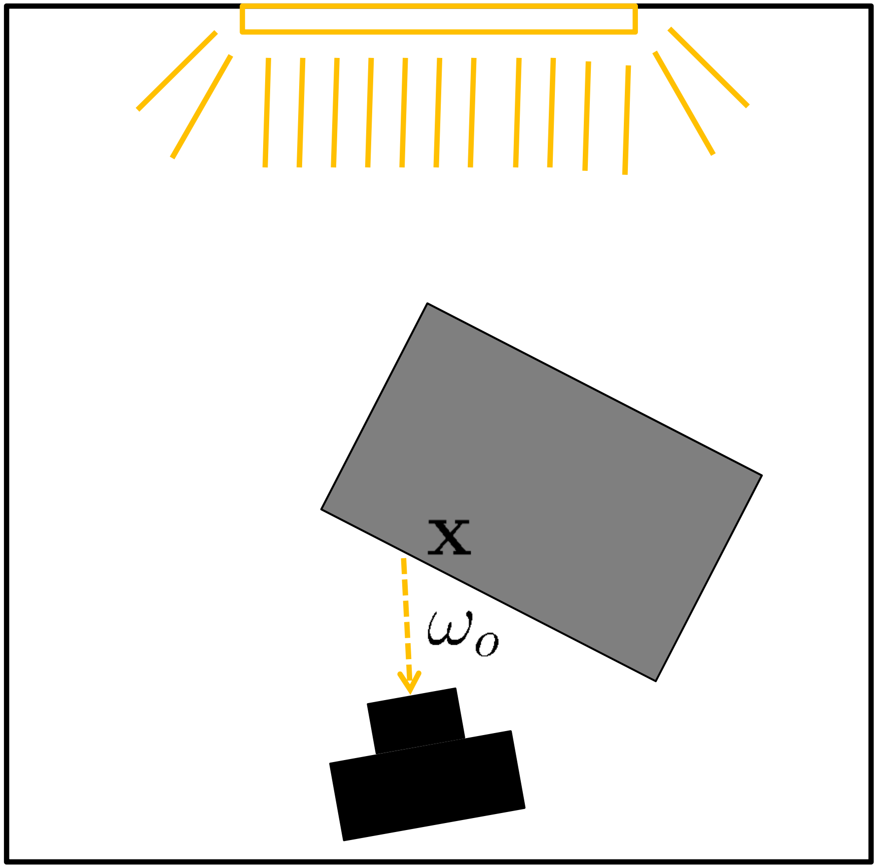
$$p_\omega(\omega_{i,2})p_\omega(\omega_{i,1})$$

$$p_\omega(\omega_{i,1})$$



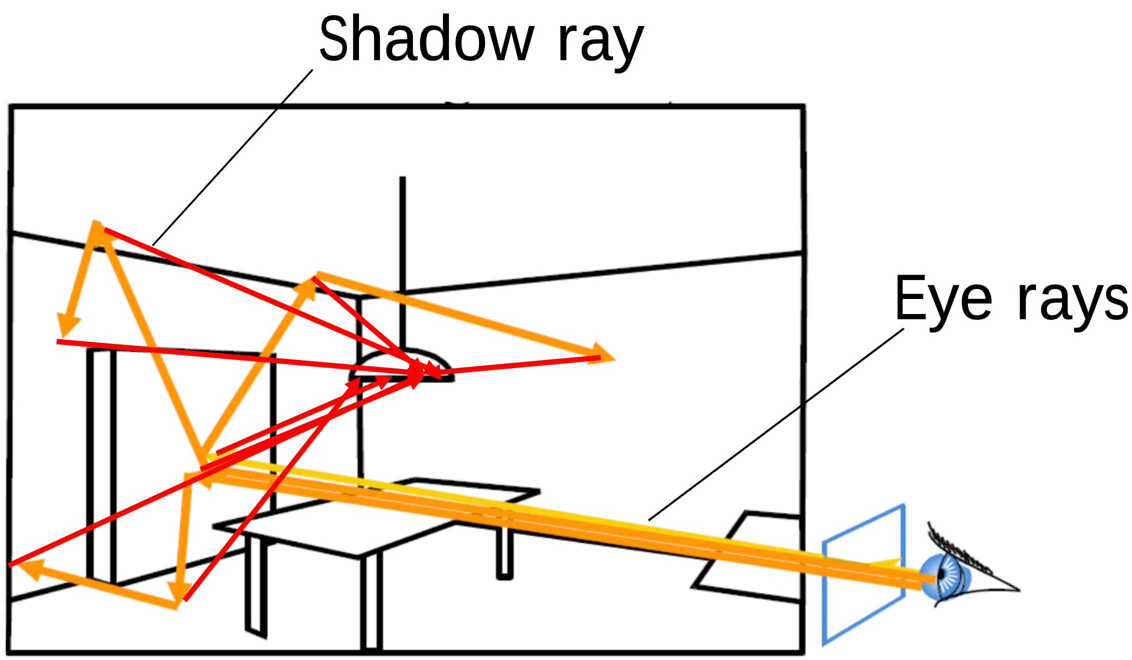
Idea for improvement

- Could reuse each shorter subsequence of a path as a shorter path
- In each step during path construction, sample light once



Path tracing

- Construct paths incrementally starting at the eye
- Shoot shadow rays at each path vertex
- “Each eye ray contributes one path of each length”
- Each eye ray contributes one sample to integral for each path length



[Cutler, Durand]

Russian roulette

- Issues
  - What should be the maximum path length?
  - Longer paths should be less likely, since they carry less radiance
- Introduce probability q to terminate path in each step
  - If termination: stop extending path (probability q)
  - Otherwise: sample next path segment as usual, probability of new path needs to be multiplied by $1\text{-}q$

Pseudocode notes

- Computes pixel color using N primary rays, i.e., paths
- Sequence of termination probabilities (Russian roulette) `q[k]` for k-th vertex along path
- PDF `p_w` measures density over solid angle
  - If point on light `x` is sampled over area, probability `p_w(x)` needs to include conversion to density over solid angle (see before)!
- `alpha` values
  - Store (*product of BRDFs and cosine factors*)/(*probability for path*)
  - Really are vectors with 3 components for RGB, here as scalar for simplicity
  - BRDF `BRDF(w_o,w_i)` with `w_o =` direction of previous path segment, `w_i =` direction of new path segment
  - `cos` is cosine of `w_i` to normal
- `shade(hitRecord,x)` includes multiplication of light (emission) with
  - BRDF at hit point with incident direction towards point `x` on light source
  - Cosine factor of direction towards point on light

Path tracing pseudocode

```
// in main rendering loop
color = 0
for i from 1 to N

  // in integrator
  alpha = 1
  hitRecord = shoot primary ray
  k = 0
  while(true)
    x = sample a point on a light source with pdf p_w(x)
    c = c + alpha*shade(hitRecord,x)/(p_w(x))
    break with probability q[k]
    hitRecord = shoot ray along w_i with pdf p_w(w_i)
    alpha = alpha*BRDF(w_o,w_i)*cos/(p_w(w_i)*(1-q[k]))
    k++

c = c/N
```

Notes: Russian roulette

- Never terminate at very first steps
- Usually $q_1 = q_2 = 0$
- Otherwise, constant probability q_i=0.5 should work fine

Notes: Probability densities

- PDF for sampling directions
  - Uniform $p_\omega = 1/2\pi$
  - Cosine weighted $p_\omega(\omega_i) = (\omega_i \cdot \mathbf{n})/\pi$
  - Advanced: importance sampling the BRDF
- Implementation in `Material.getShadingSample`
- Details on importance sampling in PBRT book by Pharr, Humphreys
  http://www.pbrt.org/

Notes: Probability densities

- PDF for sampling light sources
  - Uniform sampling

$$p_\omega(x) = 1/(\textit{number of lights}) * \\ 1/(\textit{area of selected light}) * \\ \textit{conversion to pdf over directions}$$

- Implementation
  - Select light in integrator
  - Sample location on selected light in `LightGeometry.sample`
- Advanced: multiple importance sampling

Notes: Probability densities

- Sampling area light source

Conversion to solid angle

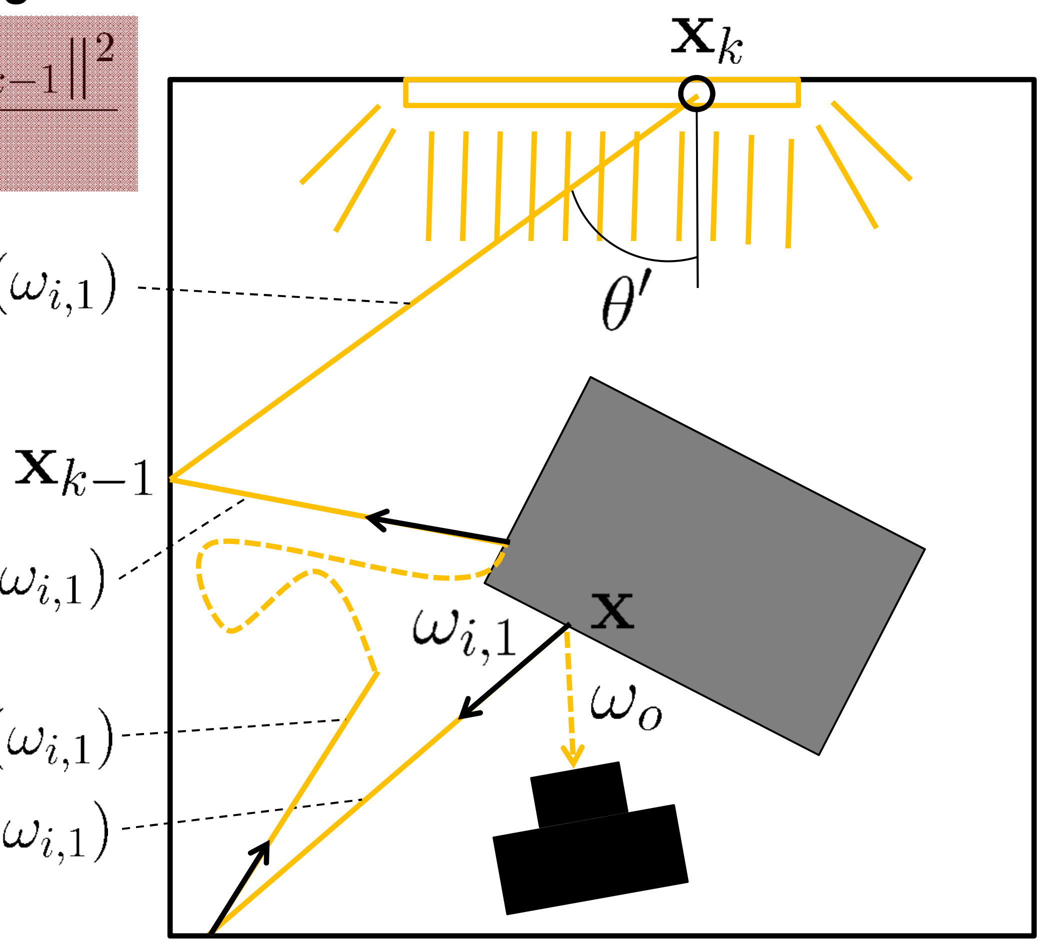
$$p_\omega(\mathbf{x}_k) = \frac{p_A(\mathbf{x}_k)\|\mathbf{x}_k - \mathbf{x}_{k-1}\|^2}{cos\theta'}$$

$$p_\omega(\mathbf{x}_k)p_\omega(\omega_{i,k-1}) \cdot \ldots \cdot p_\omega(\omega_{i,1})$$

$$p_\omega(\omega_{i,k-1}) \cdot \ldots \cdot p_\omega(\omega_{i,1})$$

$$p_\omega(\omega_{i,2})p_\omega(\omega_{i,1})$$

$$p_\omega(\omega_{i,1})$$



Notes: Refractive objects

- Pseudocode assumes reflective, refractive BRDFs are represented correctly
  - Including cosine factor in BRDF!
  - Cancels out with cosine factor in pseudocode

$$f(\mathbf{x}, \omega_o, \omega_i) = F_r(\omega_o) \frac{\delta(\omega_i - R(\omega_o, \mathbf{n}))}{|\cos\theta_i|}$$

- In practice, handle reflective/refractive objects as distinct case
  - Sampling directions: pick mirror reflection (refraction) with probability 1
  - Don‘t explicitly divide/multiply by cosine
  - Implementation: use flag in `Material.ShadingSample`

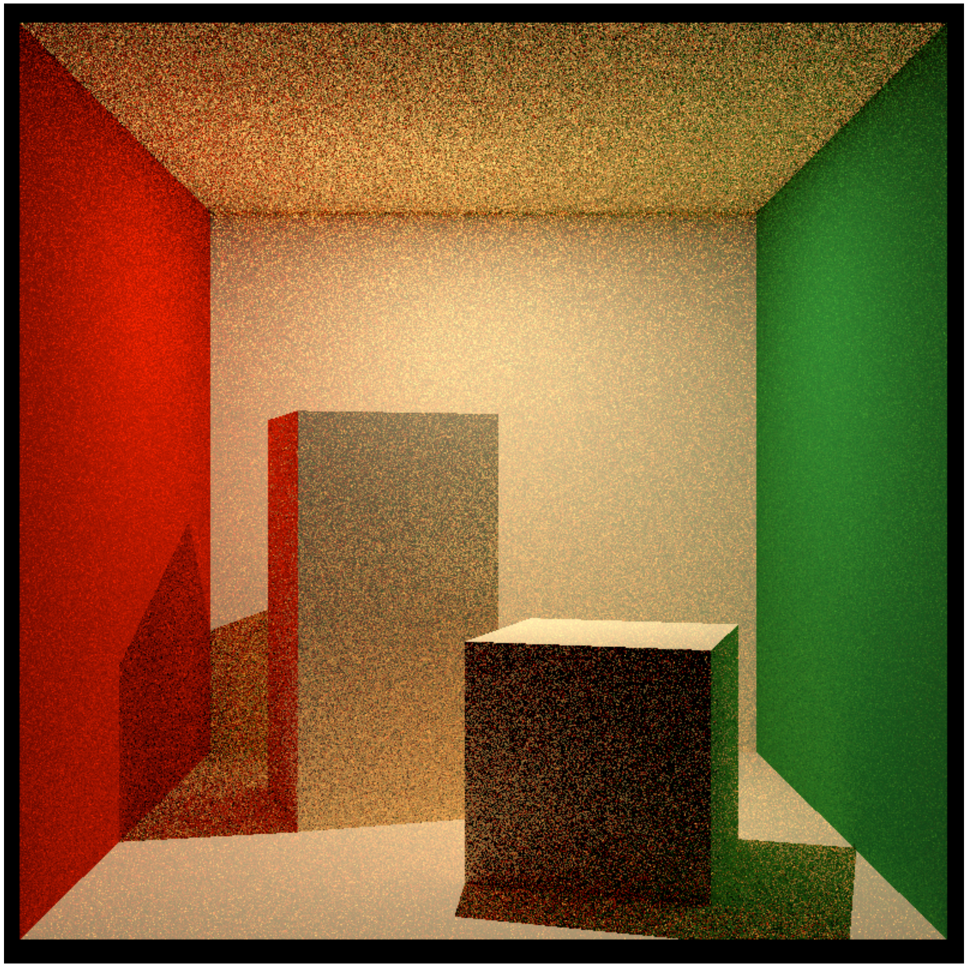
Notes: Refractive objects

- Randomly sample (trace) either reflected or refracted ray with given probability
  - Can pick constant probability
  - Can pick probability based on Fresnel reflection coefficient
- Need to include probability in overall pdf of path!
- Refractive objects tend to produce a lot of noise in path tracing...

Notes: Emitting surfaces

- Emitting surfaces (area lights) should be part of regular scene geometry
- If a ray (accidentally) hits emitting surface, don‘t add emission
  – Emission is taken care of by shadow rays
- Exception: need to add emission if
  – Eye ray hits emitting surface
  – Ray segment was generated from refractive surface (in this case, no shadow ray needs to be generated)

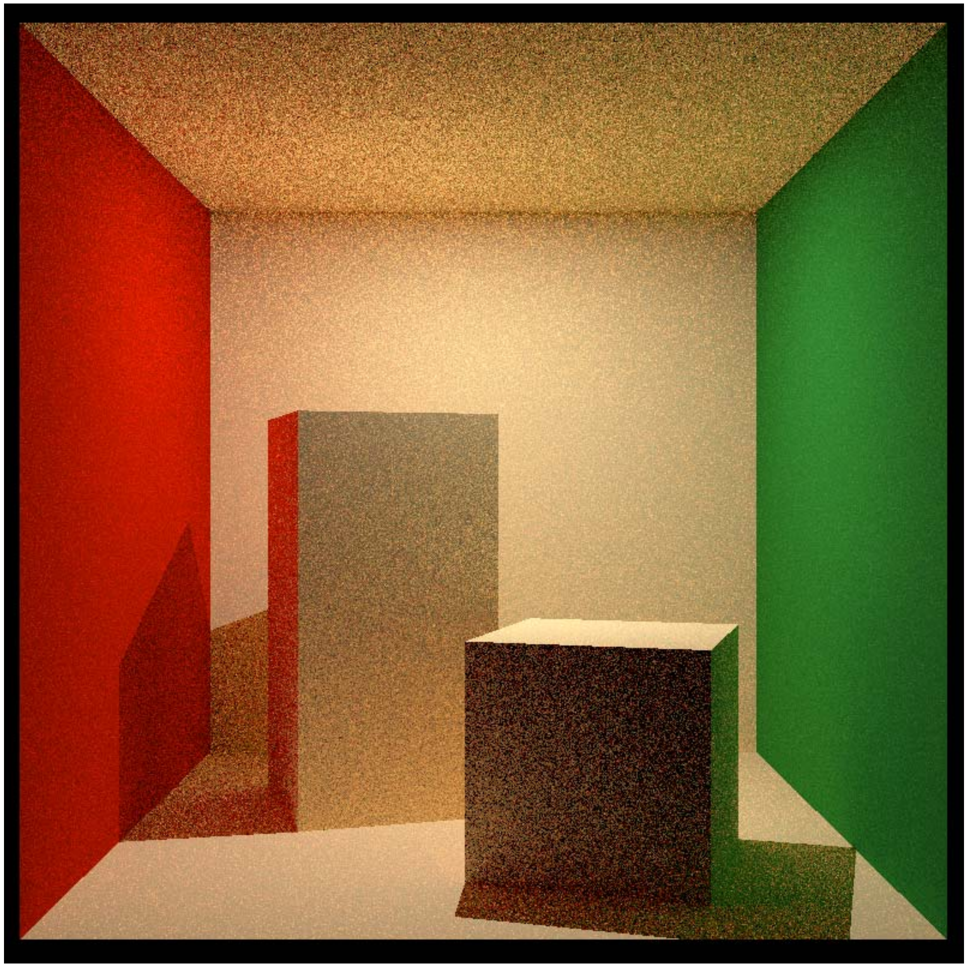
Path tracing example



[Kajiya `86]

4 rays/pixel

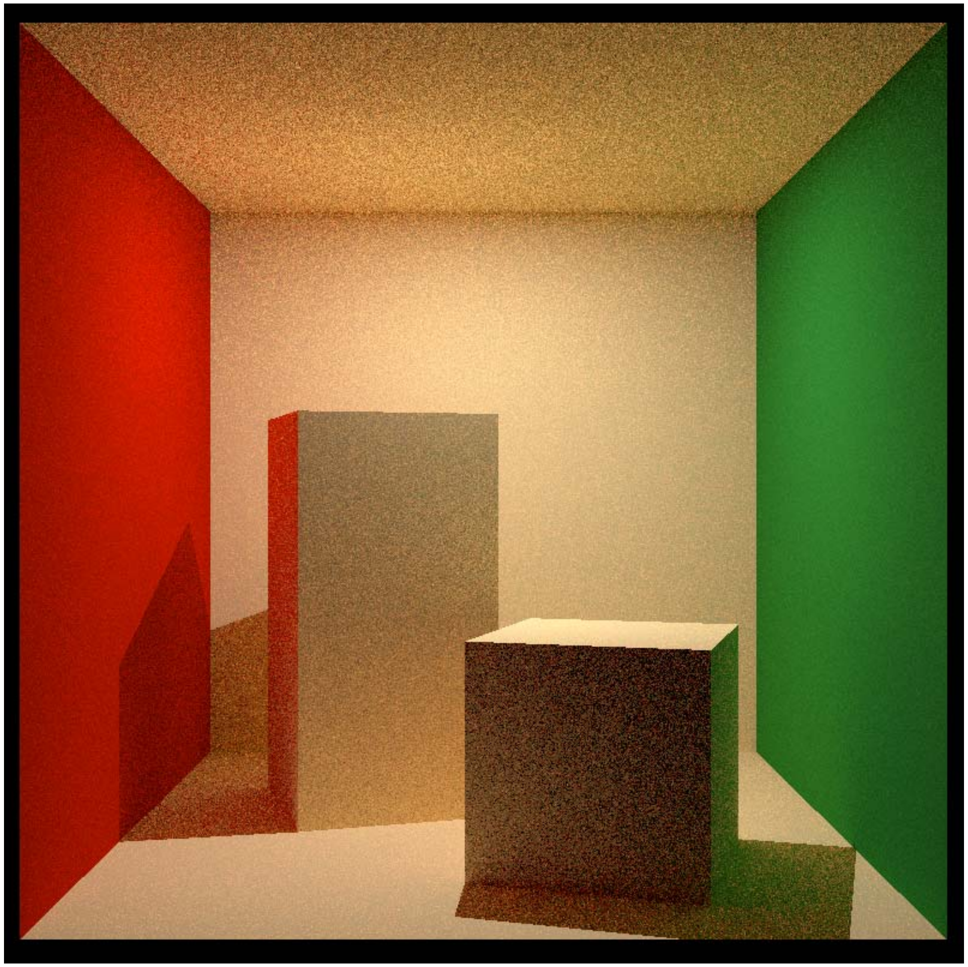
Path tracing example



[Kajiya `86]

8 rays/pixel

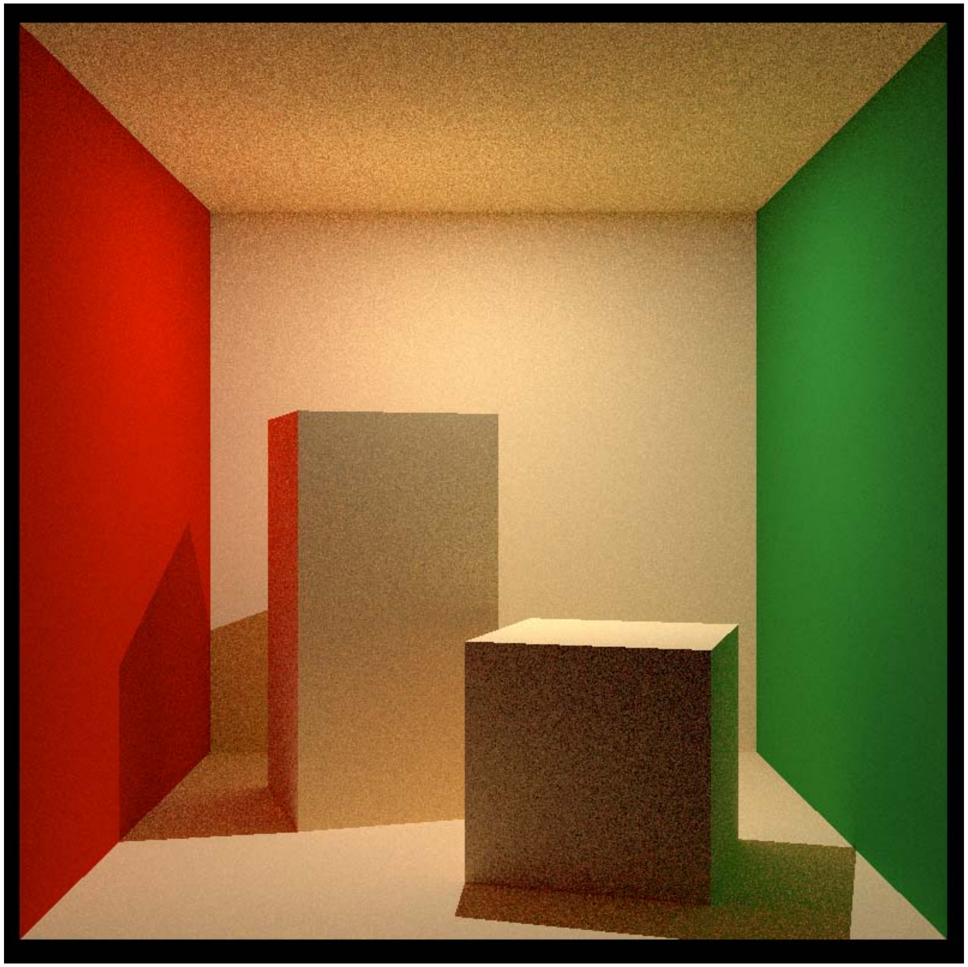
Path tracing example



[Kajiya `86]

16 rays/pixel

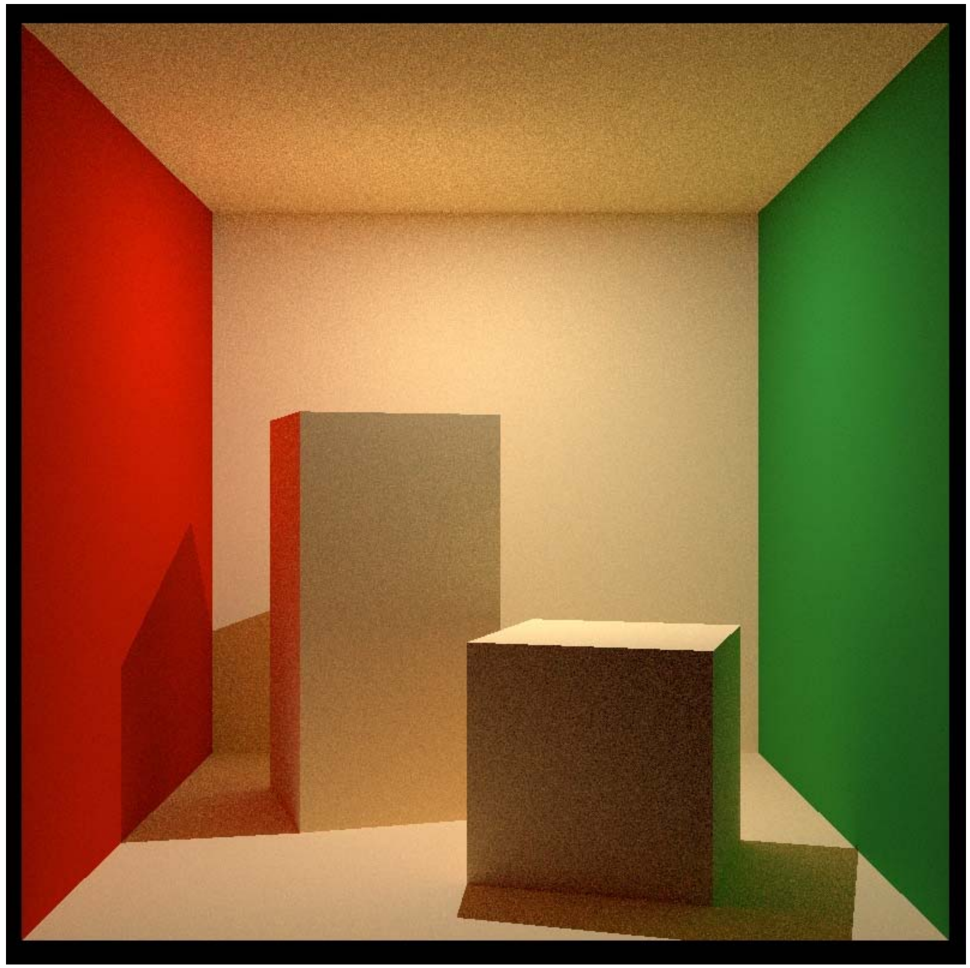
Path tracing example



[Kajiya `86]

32 rays/pixel

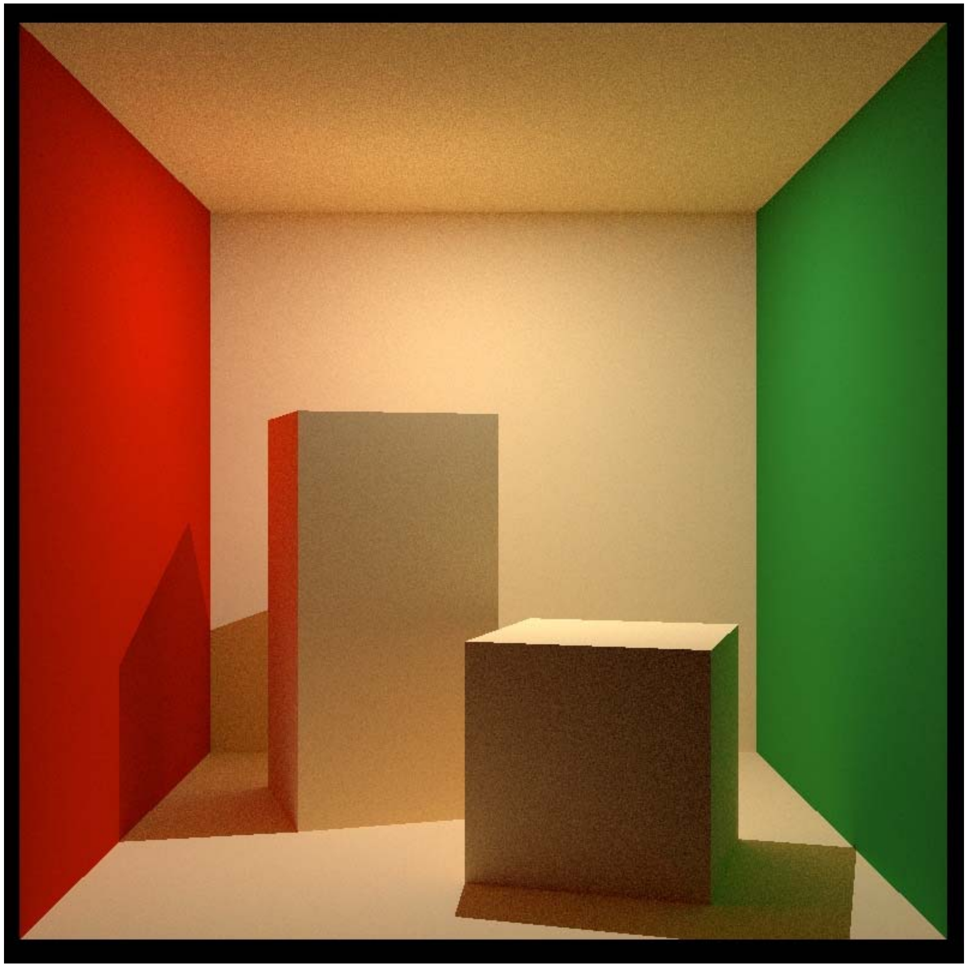
Path tracing example



[Kajiya `86]

64 rays/pixel

Path tracing example



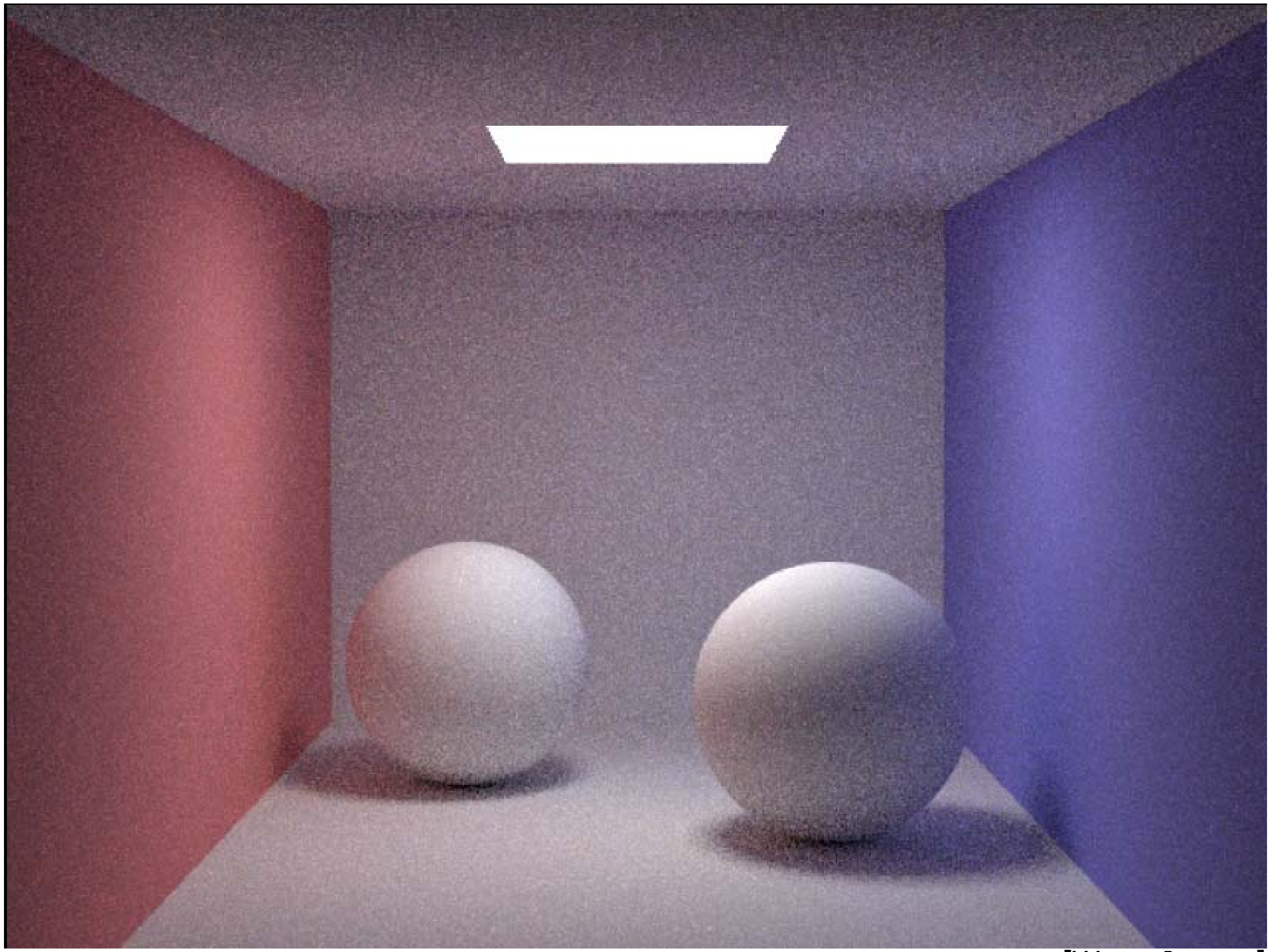
[Kajiya `86]

128 rays/pixel

Path tracing: the good

- **Unbiased**
  Expected value for each pixel is the correct solution of the rendering equation, independent of number of samples
- **Consistent**
  If we shoot infinitely many rays, we will get the correct solution

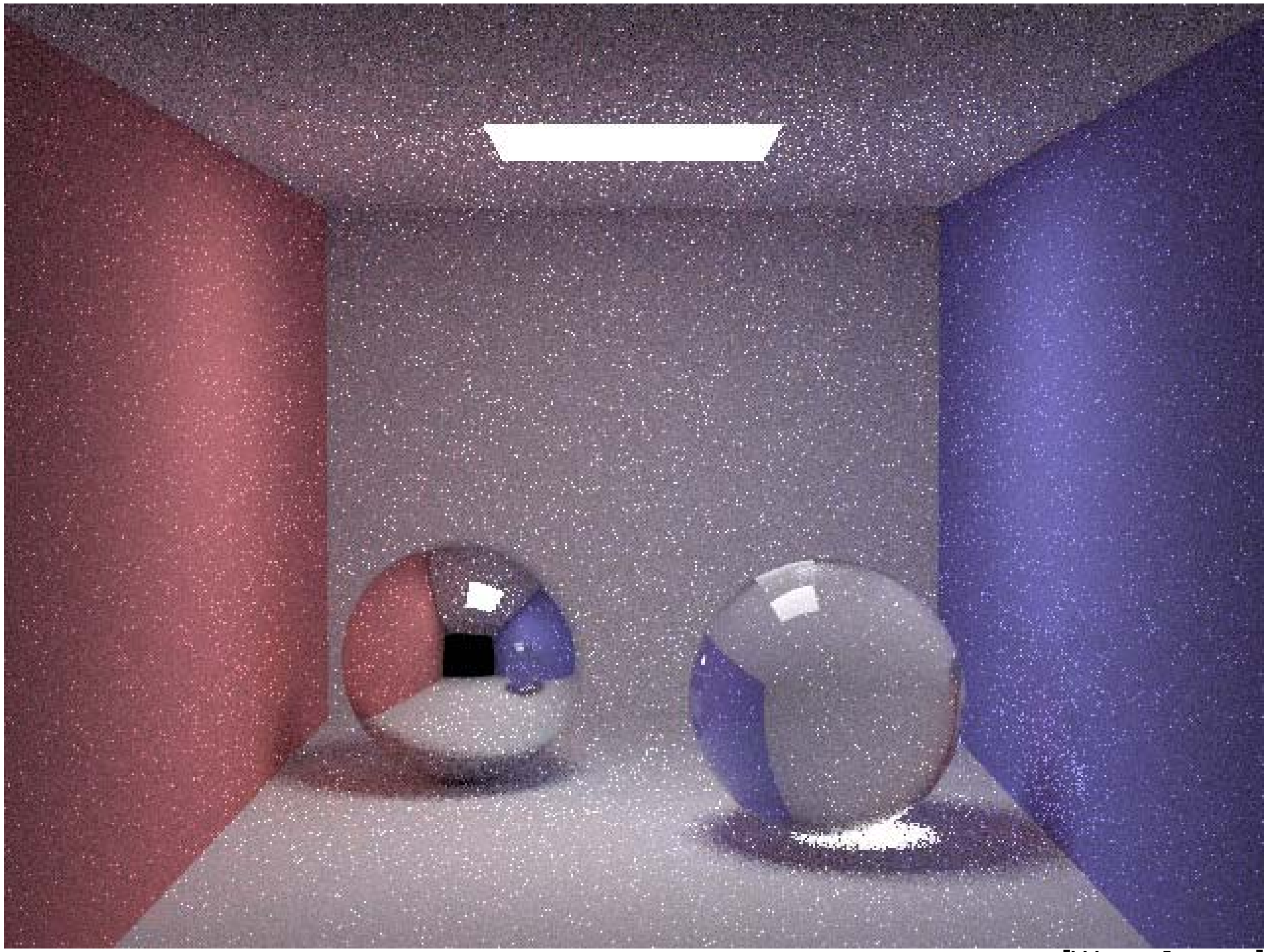
And the bad...



[Wann Jensen]

10 paths/pixel

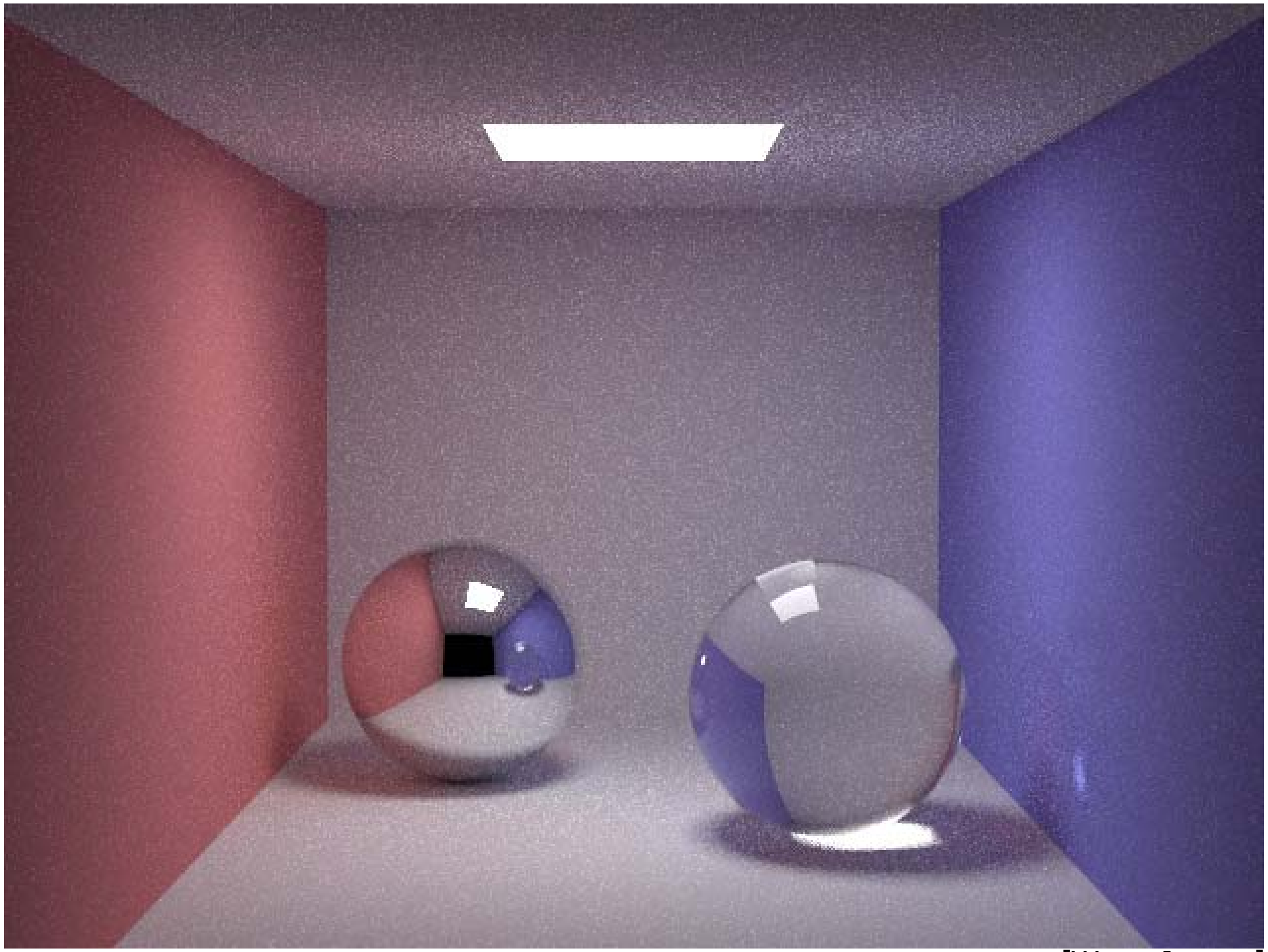
And the bad...



[Wann Jensen]

10 paths/pixel

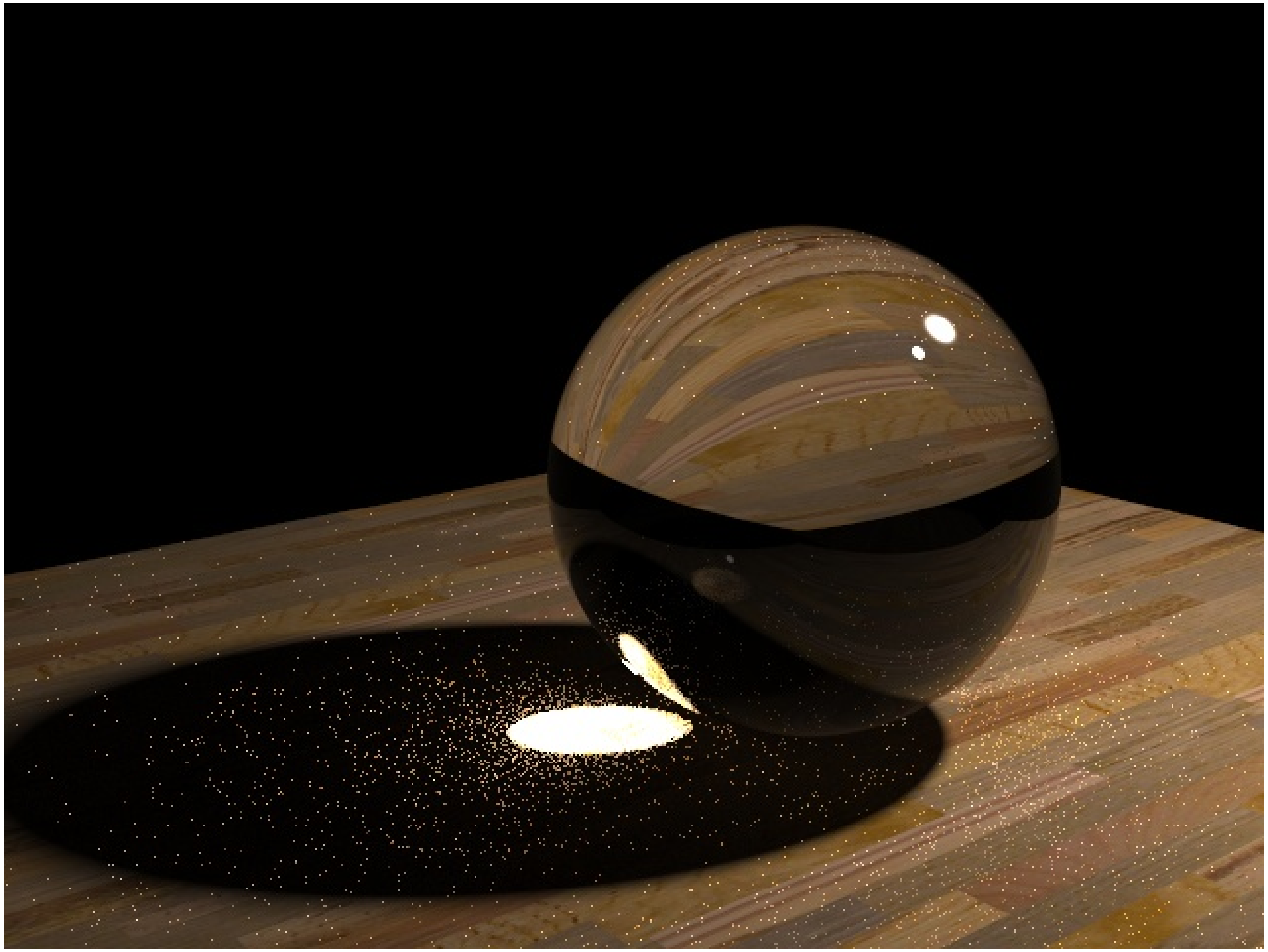
And the bad...



[Wann Jensen]

100 paths/pixel

And the bad...



[Wann Jensen]

1000 paths/pixel

Light transport notation

- Light L
- Diffuse D
- Specular S
- Eye E
- Example

Summary

- Path tracing often used as reference algorithm
  - Generates „ground truth“ image with enough samples (thousands)
- Not very practical because of noise issues
  - In particular for certain types of light paths (caustics)
  - Even with sophisticated sampling (multiple importance sampling, low-discrepancy sequences)

Next time

- Photon mapping