Computer Graphics I

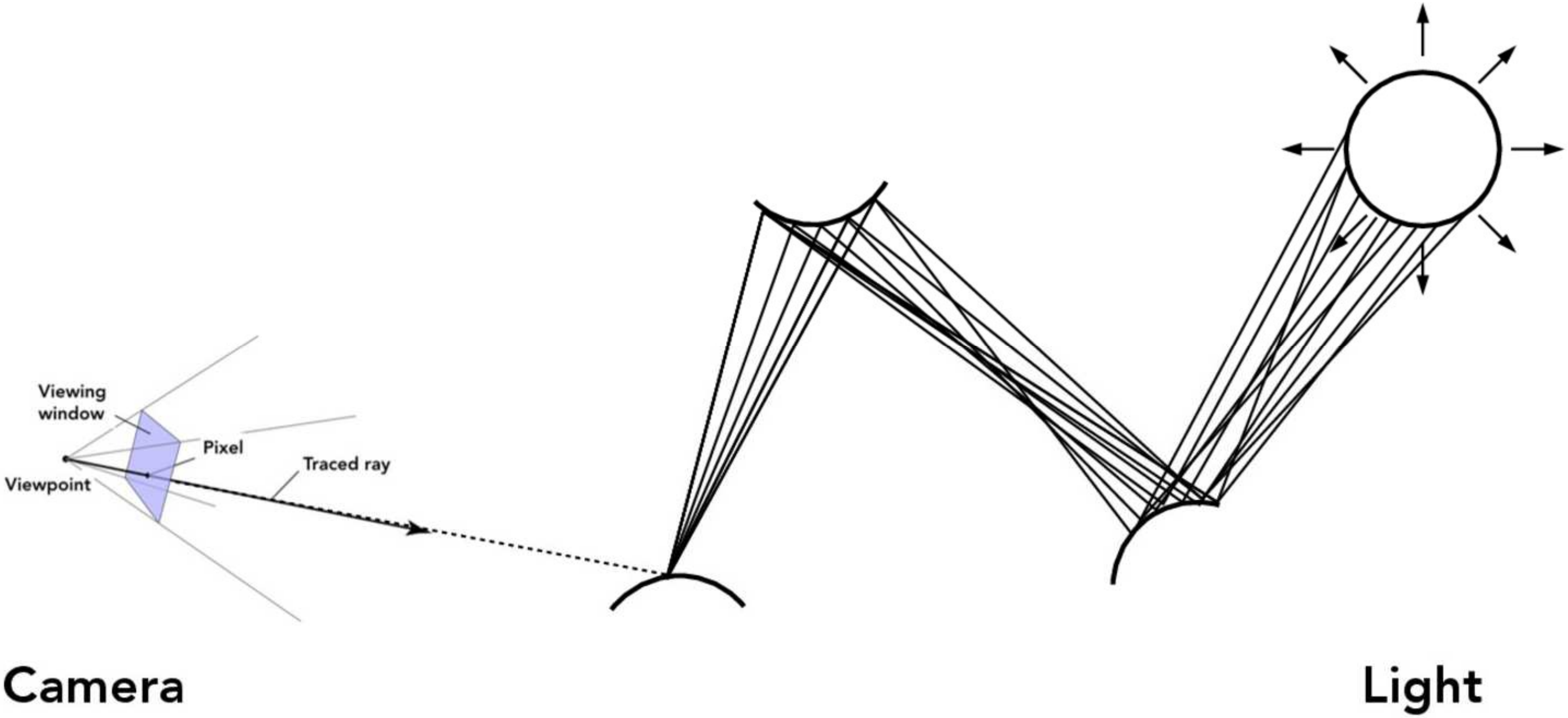
Lecture 14: Global illumination 2

Xiaopei LIU

School of Information Science and Technology
ShanghaiTech University

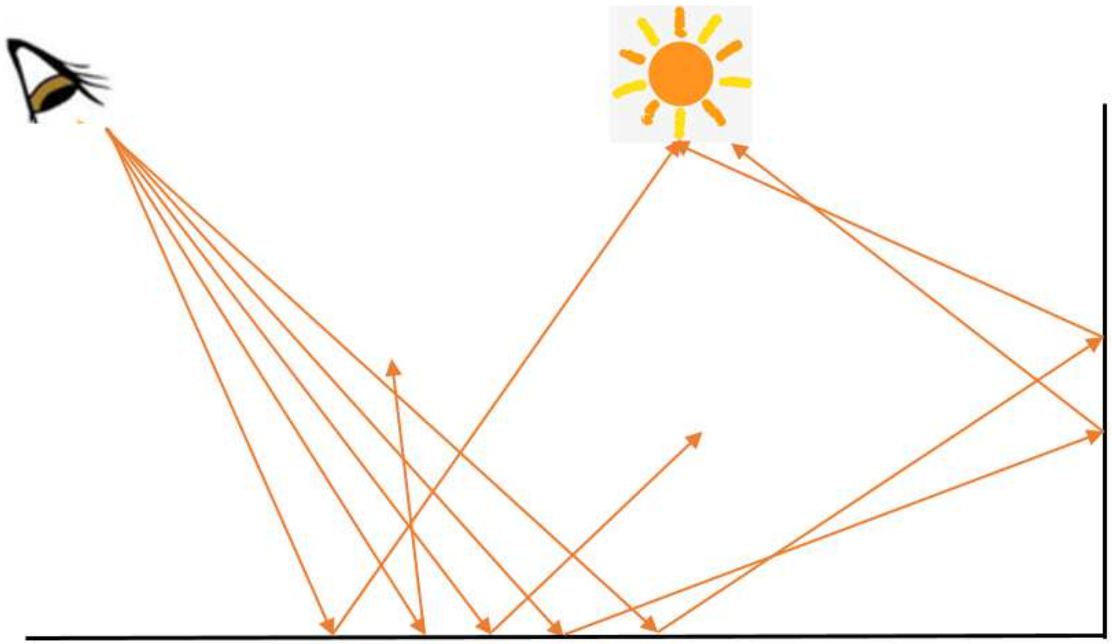
Recall on path tracing

- **Camera rays only**



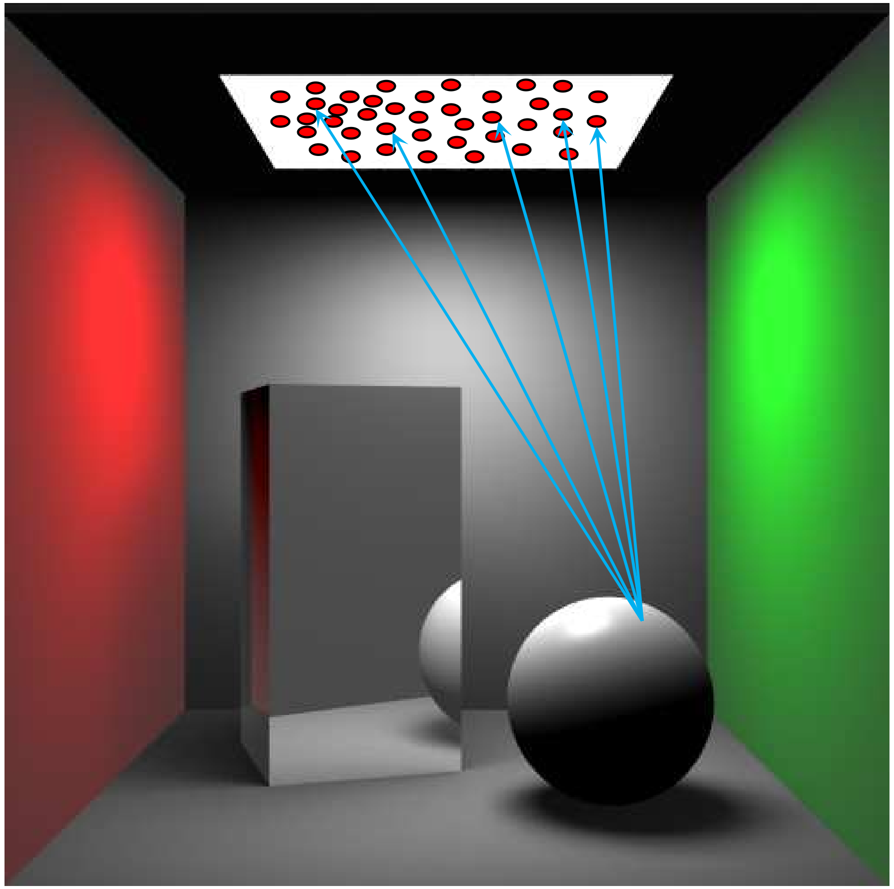
Recall on path tracing

- **Approximation**
  - Instead of shooting multiple rays per intersection, we shoot only one ray
  - Instead of shooting only a few rays per pixel, we should large amount of rays per pixel



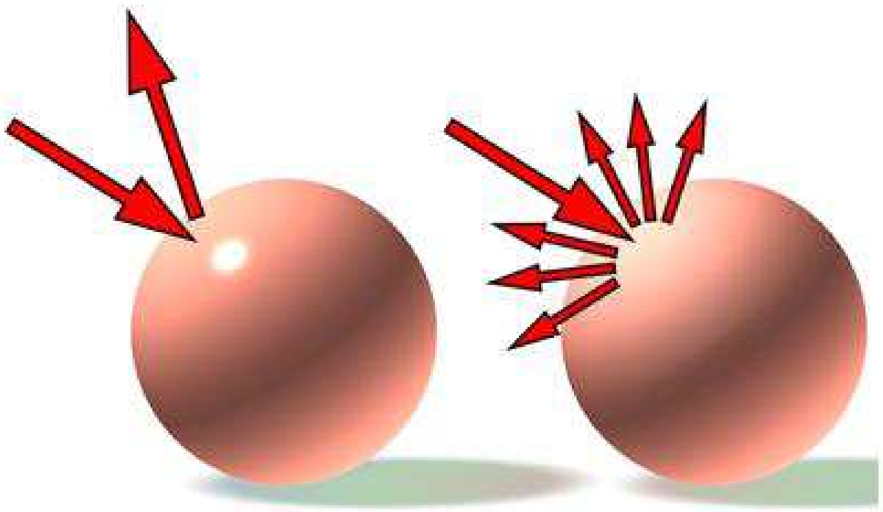
Recall on path tracing

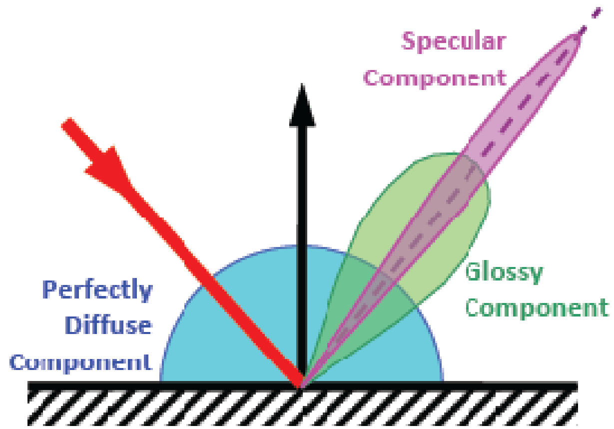
- **Path sampling**
  - Sampling according to light sources (direct lighting)
    - With respect to the local surface orientation



Recall on path tracing

- **Path sampling**
  - Sampling according to BSDF (indirect lighting)
    - Different components composed together





Recall on path tracing

- **Multiple importance sampling**
  - We can estimate distribution for BSDF *f* and light source L_d, but not for the whole
  - Combine the BSDF and light source distribution

$$\frac{1}{n_f}\sum_{i=1}^{n_f}\frac{f(X_i)g(X_i)w_f(X_i)}{p_f(X_i)}+\frac{1}{n_g}\sum_{j=1}^{n_g}\frac{f(Y_j)g(Y_j)w_g(Y_j)}{p_g(Y_j)}$$

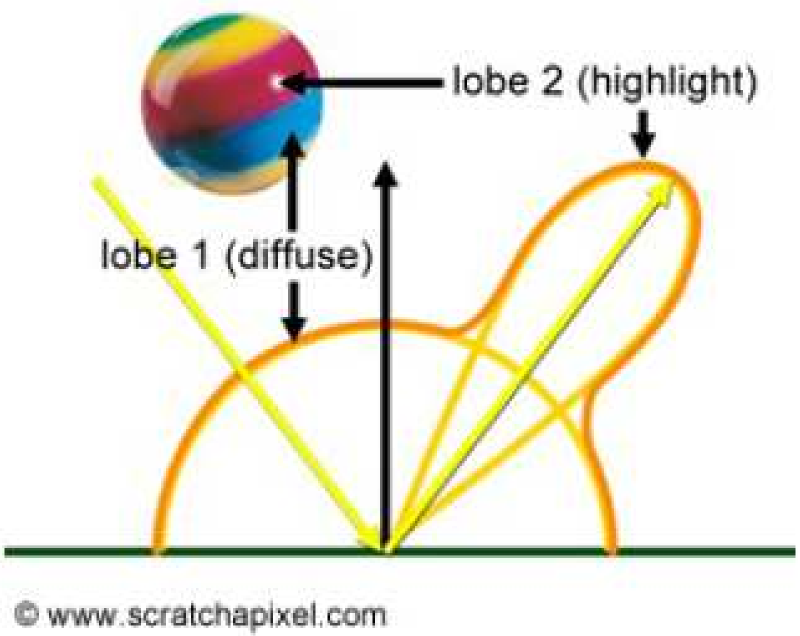
  - Balance heuristic for weights

$$w_s(x)=\frac{n_s p_s(x)}{\sum_i n_i p_i(x)}$$

Recall on path tracing

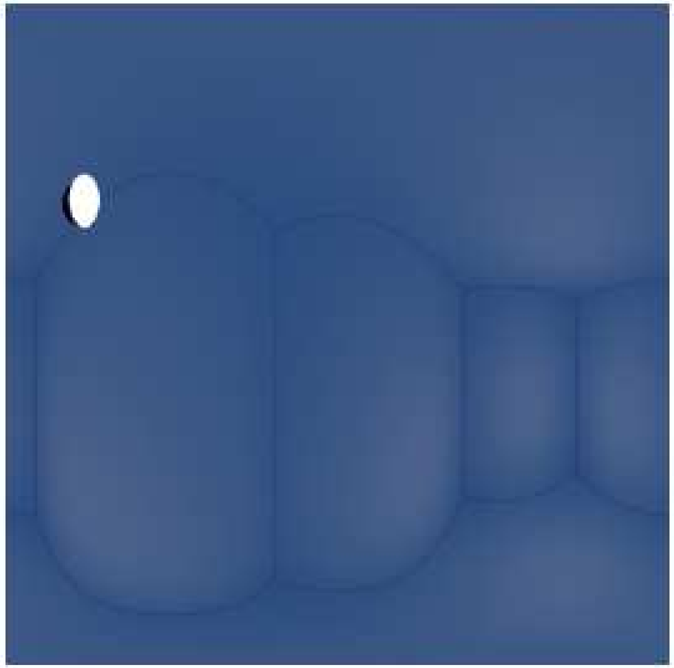
- **Importance sampling in rendering**

materials: sample important "lobes"



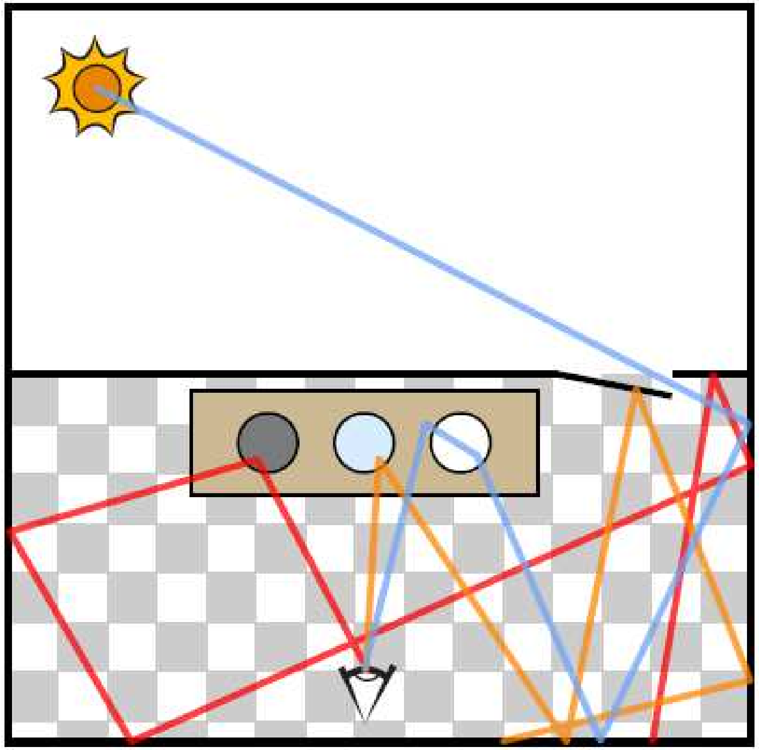
(important special case: perfect mirror!)

illumination: sample bright lights



Recall on path tracing

- **Good paths can be hard to find!**



Idea:
Once we find a good path, perturb it to find nearby "good" paths.

bidirectional path tracing

Metropolis light transport (MLT)

1. Metropolis-Hastings Algorithm

Review on Metropolis sampling

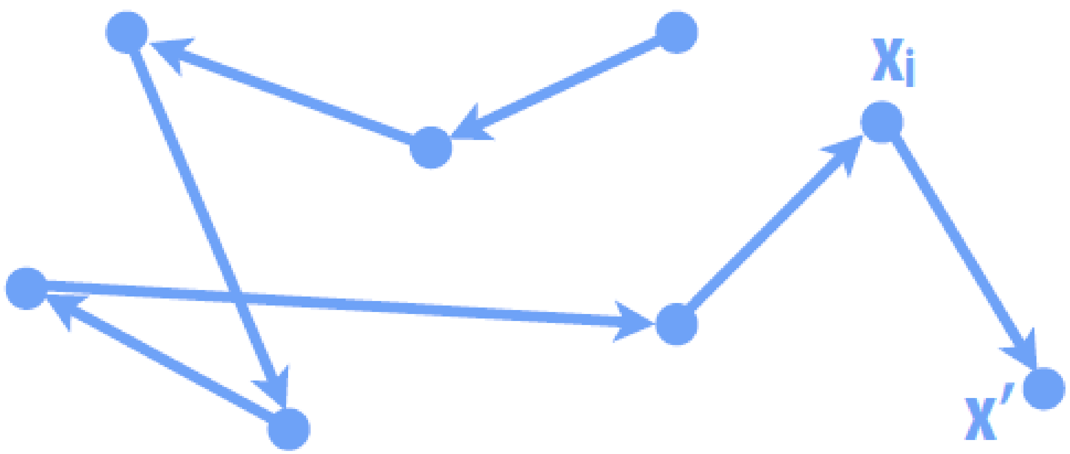
- **Basic algorithm**
  - Generate a set of samples X_i from a function f defined over an arbitrary dimensional space Ω
    - Select the first sample X_0
    - Each sample X_i is generated using a random mutation to X_{i-1} to compute a proposed sample X'
    - In order to compute X', we must compute a tentative transition function T($X \rightarrow X'$) : the transition probability
    - Compute the acceptance probability a($X \rightarrow X'$)

$$a(X \to X') = \min\left(1, \frac{f(X')\,T(X' \to X)}{f(X)\,T(X \to X')}\right) \quad \Rightarrow \quad a(X \to X') = \min\left(1, \frac{f(X')}{f(X)}\right)$$

Symmetric proposal

Metropolis-Hastings algorithm (MH)

- Standard Monte Carlo: sum up independent samples
- MH: take random walk of dependent samples ("mutations")
- Basic idea: prefer to take steps that increase sample value



$\alpha := f(x') / f(x)$ "transition probability"

if random x_i in $[0,1] < \alpha$:

  $x_{i+1} = x'$

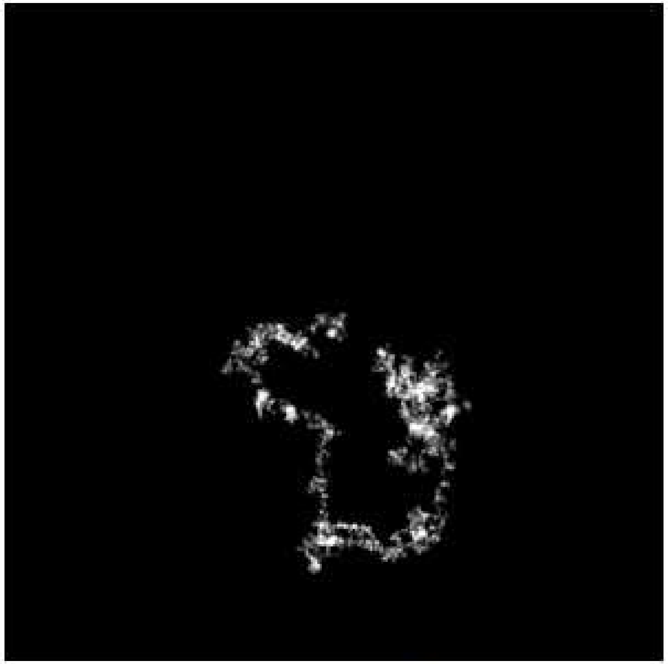
else:

  $x_{i+1} = x_i$

- If careful, sample distribution will be proportional to integrand
  - make sure mutations are "ergodic" (reach whole space)
  - need to take a long walk, so initial point doesn't matter ("mixing")

Metropolis-Hastings: sampling an Image

- Want to take samples proportional to image density f
- Start at random point; take steps in (normal) random direction
- Occasionally jump to random point (ergodicity)
- Transition probability is "relative darkness" $f(x')/f(x_i)$



short walk

long walk

(original image)

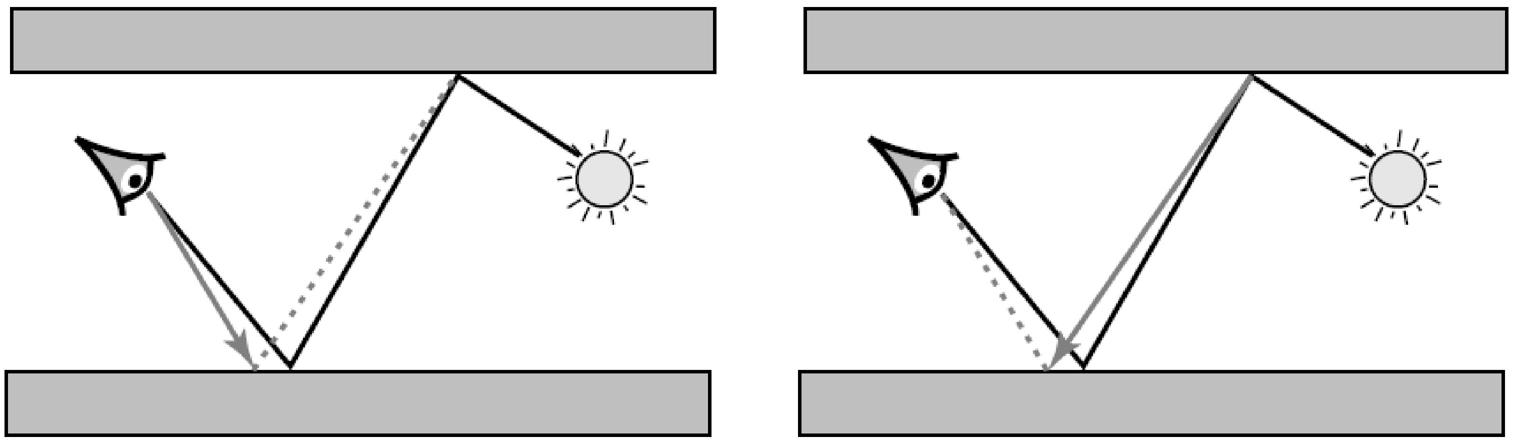
Metropolis Light Transport

- **A variant of bidirectional path tracing**
  - Application of Metropolis-Hastings algorithm to the rendering equation
  - Construct paths from the eye to a light source using bidirectional path tracing
  - Each path is found by mutating the previous path

$\bar{x} \leftarrow \textsc{InitialPath}()$
$image \leftarrow \{$ array of zeros $\}$
for $i \leftarrow 1$ **to** N
  $\bar{y} \leftarrow \textsc{Mutate}(\bar{x})$
  $a \leftarrow \textsc{AcceptProb}(\bar{y}|\bar{x})$
  if $\textsc{Random}() < a$
    then $\bar{x} \leftarrow \bar{y}$
  $\textsc{RecordSample}(image, \bar{x})$
return $image$

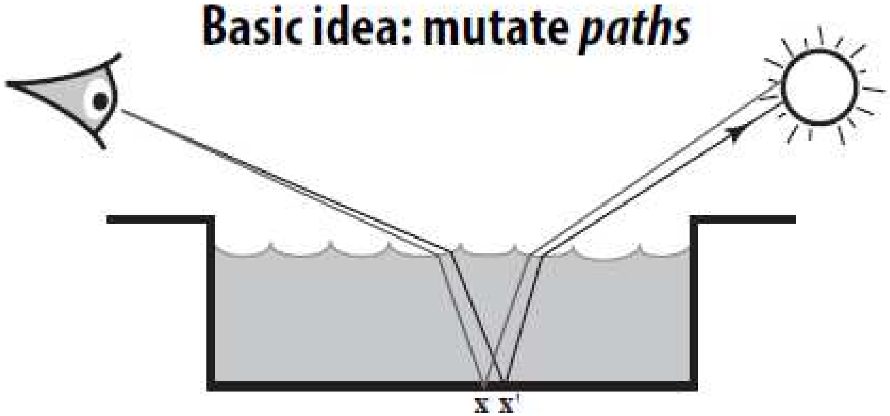
Metropolis Light Transport

- **Path mutation (perturbation)**
  - Local exploration
    - When a path makes a large contribution to image
    - Sample more similar path by small perturbation



Lens perturbation Caustic perturbation

Metropolis light transport



(For details see Veach, “Robust Monte Carlo Methods for Light Transport Simulation”)

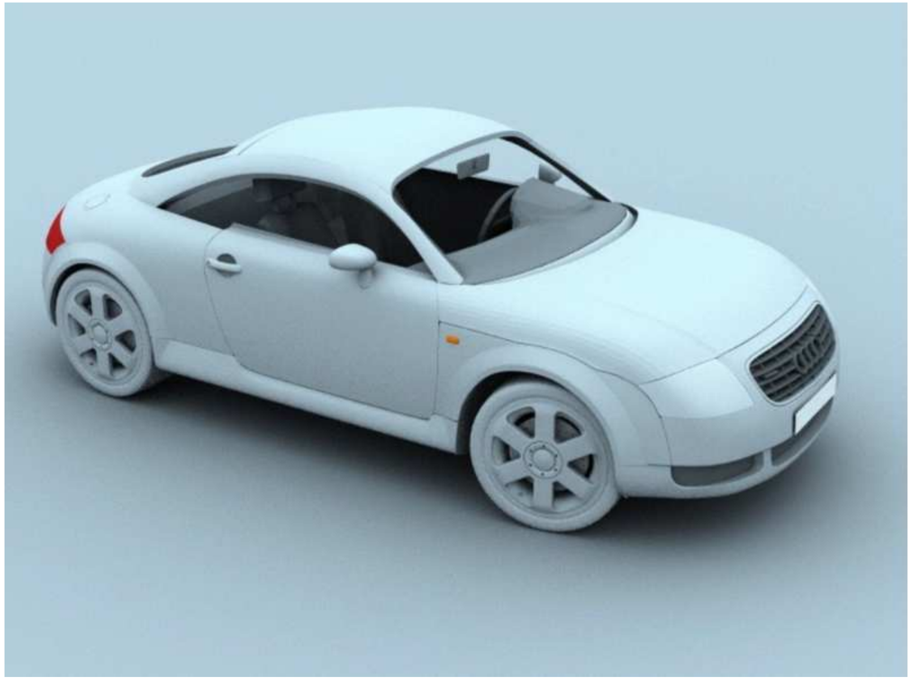
path tracing

Metropolis light transport (same time)

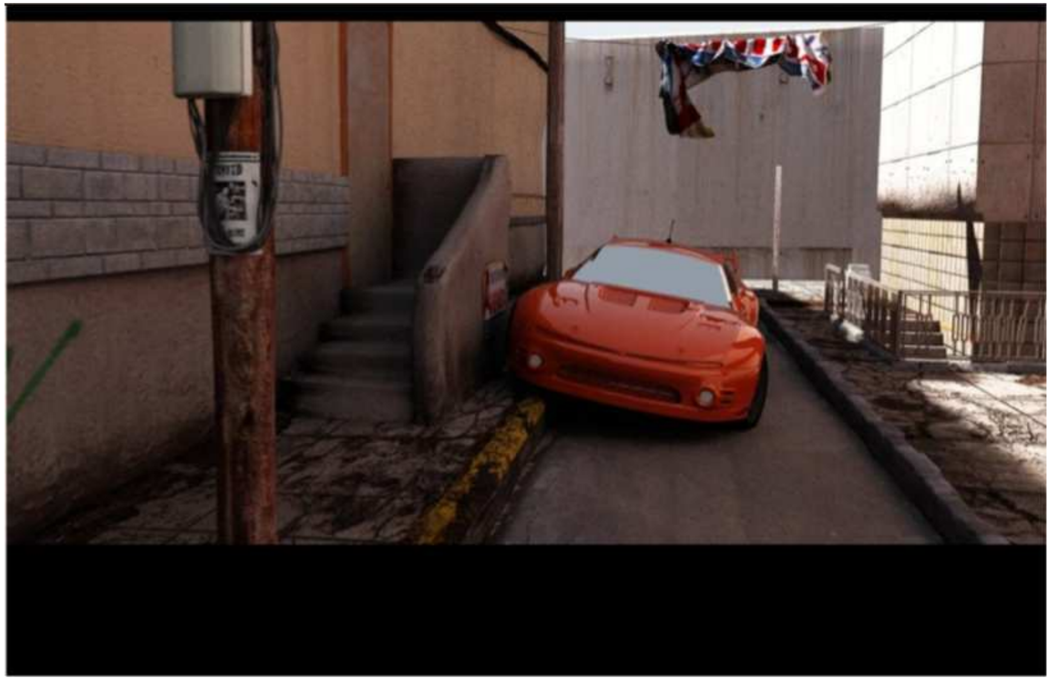
2. Bidirectional Path Tracing

Bidirectional path tracing

- **What is traditional path tracing good for?**
  - Diffuse surfaces, large area of lighting



M. Fajardo, Arnold Path Tracer



1356 x 654, 16 paths/pixel, 2 bounces, 250,000 faces

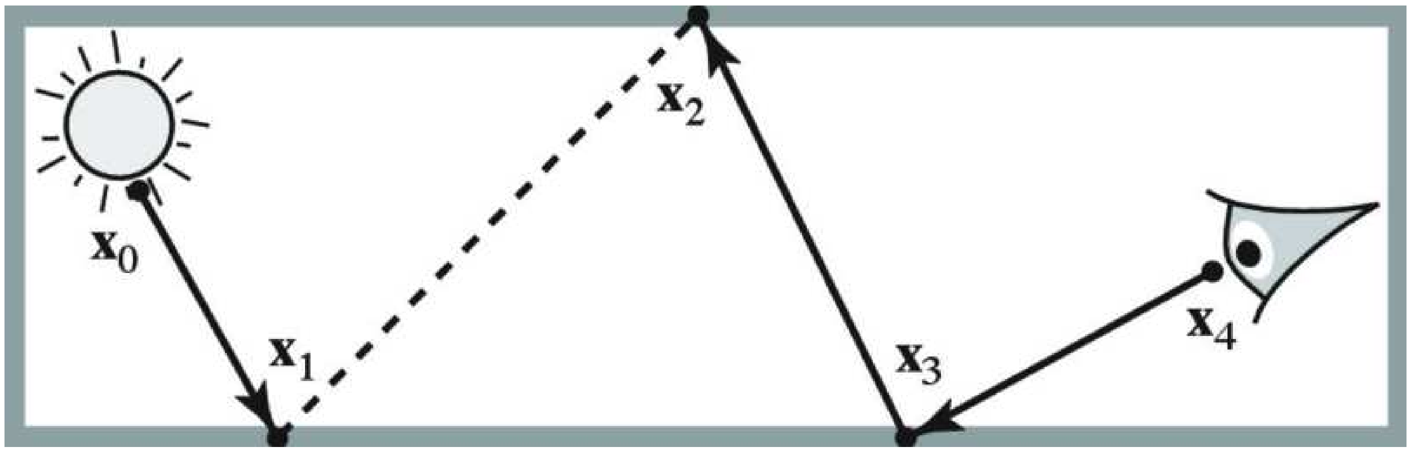
Bidirectional path tracing

- **What makes traditional path tracing difficult?**
  - Concentrated lighting, e.g., caustics

1000 paths/pixel

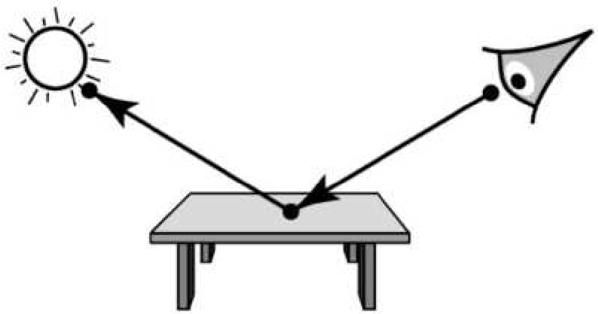
Bidirectional Path Tracing

- **Forward path tracing**
  - No control over path length
  - Hit light source after n bounces, or get terminated by Russian roulette
- **Idea**
  - Connect paths from light source and camera (eye)
  - Construct bidirectional paths



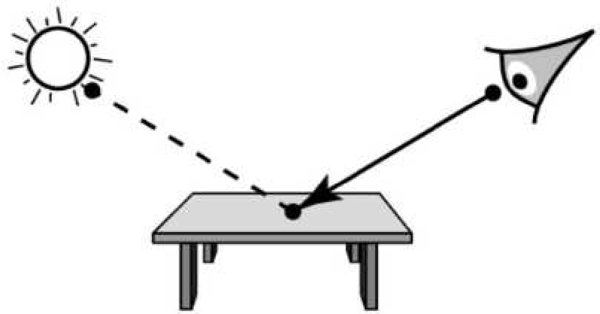
Bidirectional path tracing

- **All one-bounce cases**

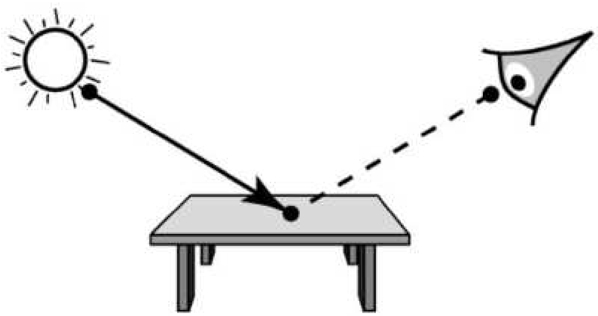


standard (forward) path tracing

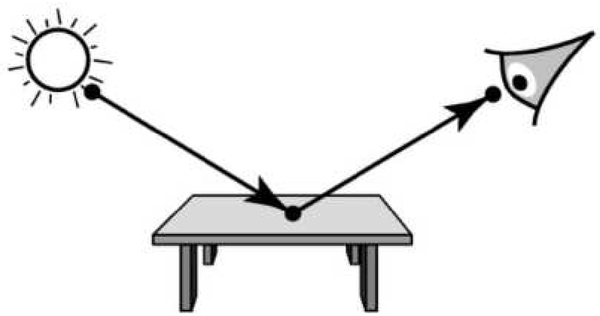
fails for point light sources



direct lighting



visualize particles from light

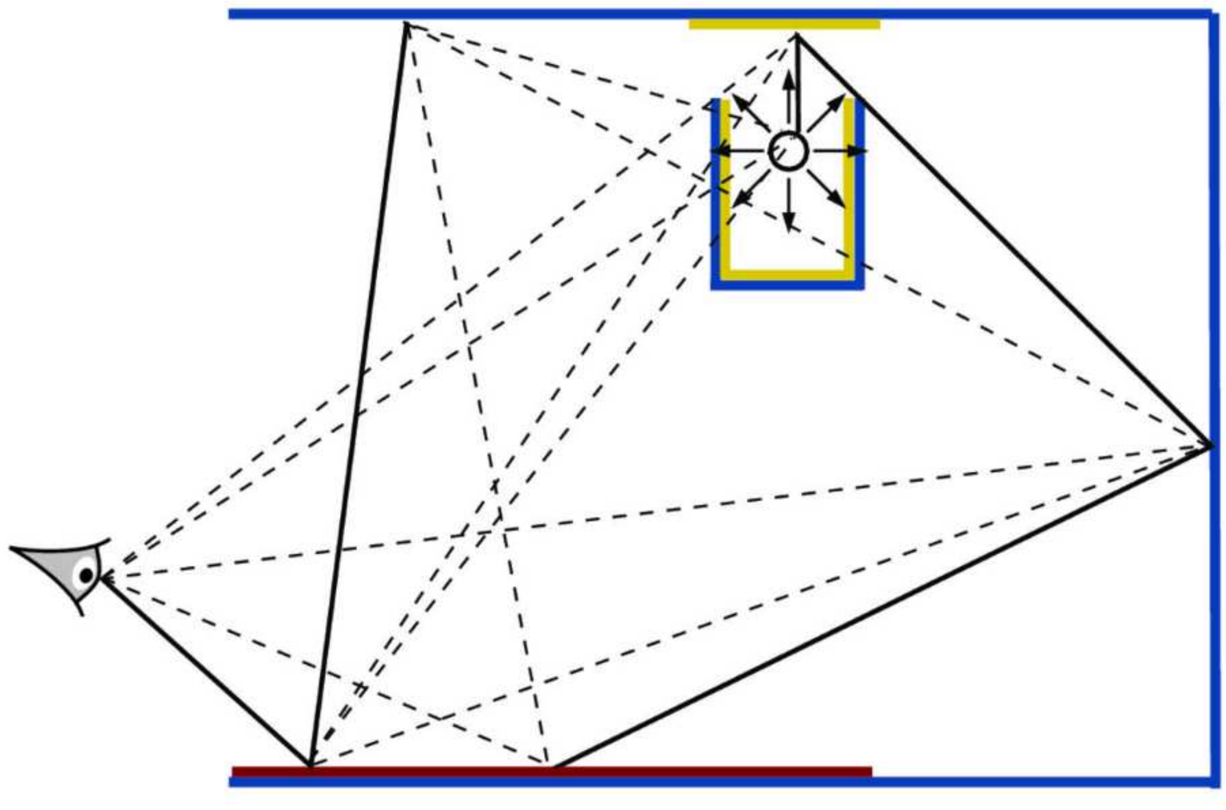


backward path tracing

fails for a pinhole camera

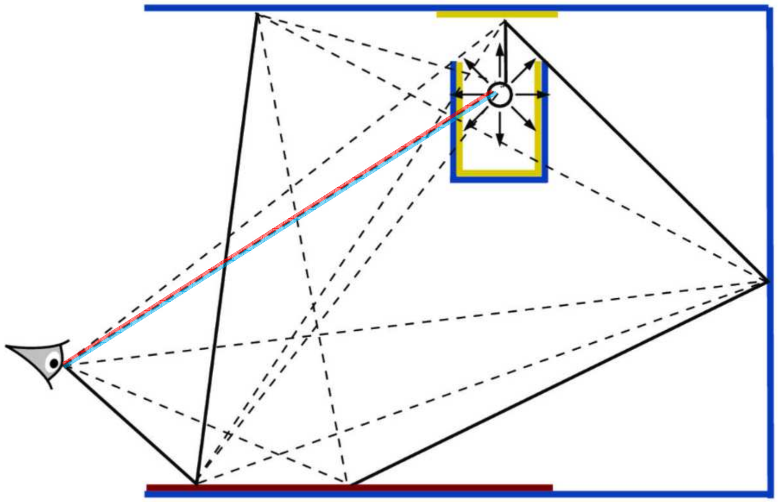
Bidirectional Path Tracing

- **Five-bounce example**
  - A combination of path from light source and camera



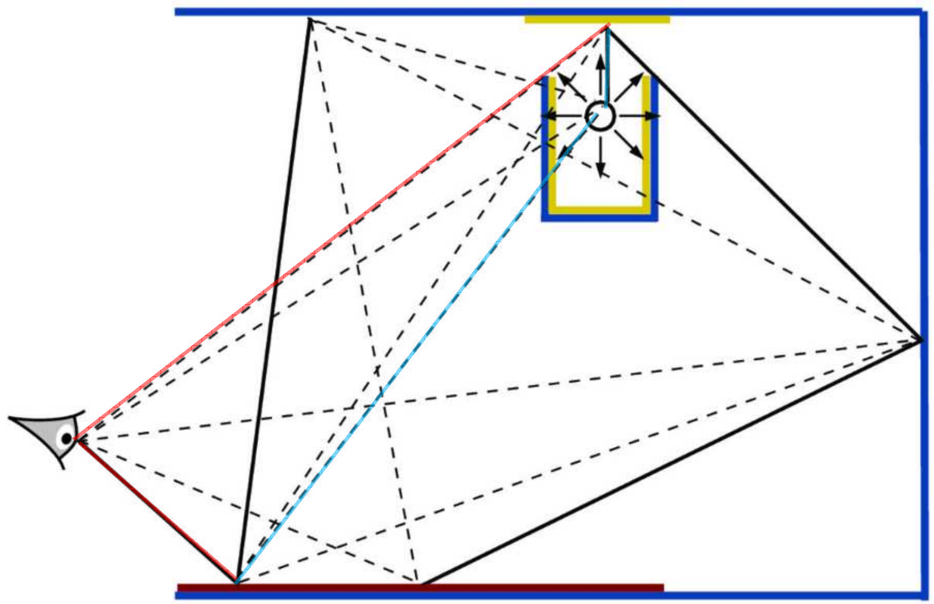
Bidirectional path tracing

- **Fixed-length bounce rendering**
  - Enumerating all possibilities
    - s : number of light source ray path
    - t : number of camera ray path
    - Rendering is based on a fixed length L: s+t=L
  - s+t=1: direct emission



Bidirectional path tracing

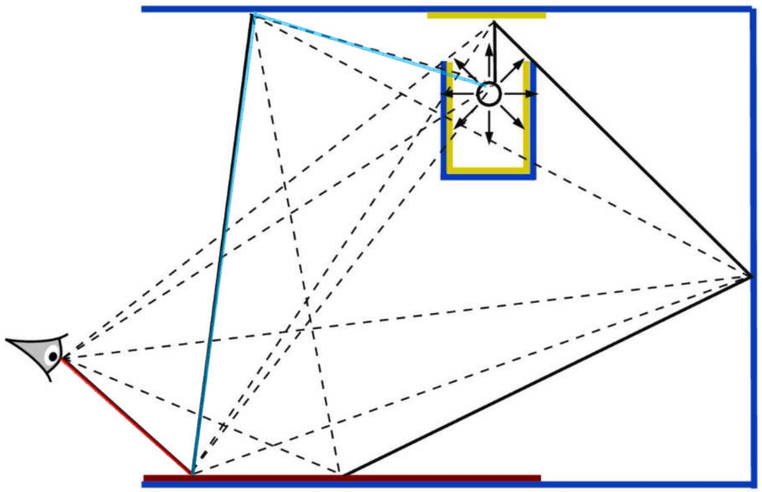
- **Fixed-length bounce rendering**
  - s+t=2: direct illumination



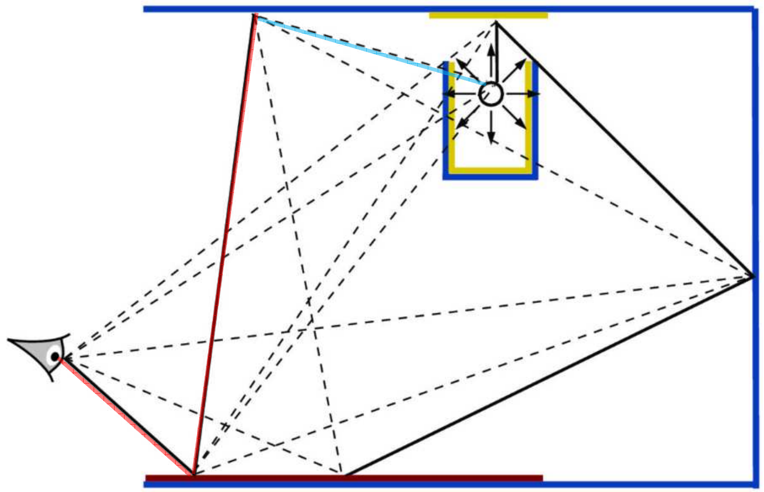
s=1, t=1

Bidirectional path tracing

- **Fixed-length bounce rendering**
  - s+t=3: indirect illumination



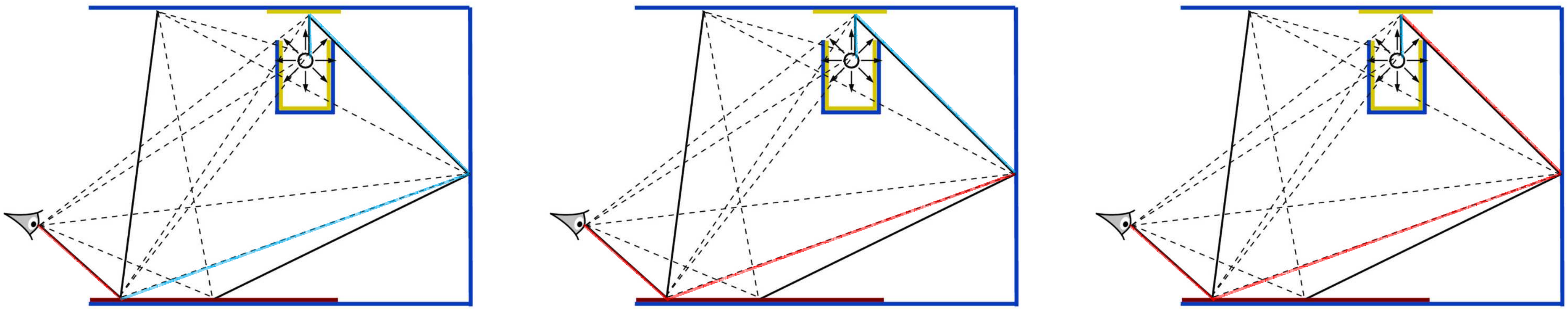
s=2, t=1



s=1, t=2

Bidirectional path tracing

- **Fixed-length bounce rendering**
  - s+t=4: indirect illumination



s=3, t=1 s=2, t=2 s=1, t=3

Bidirectional path tracing

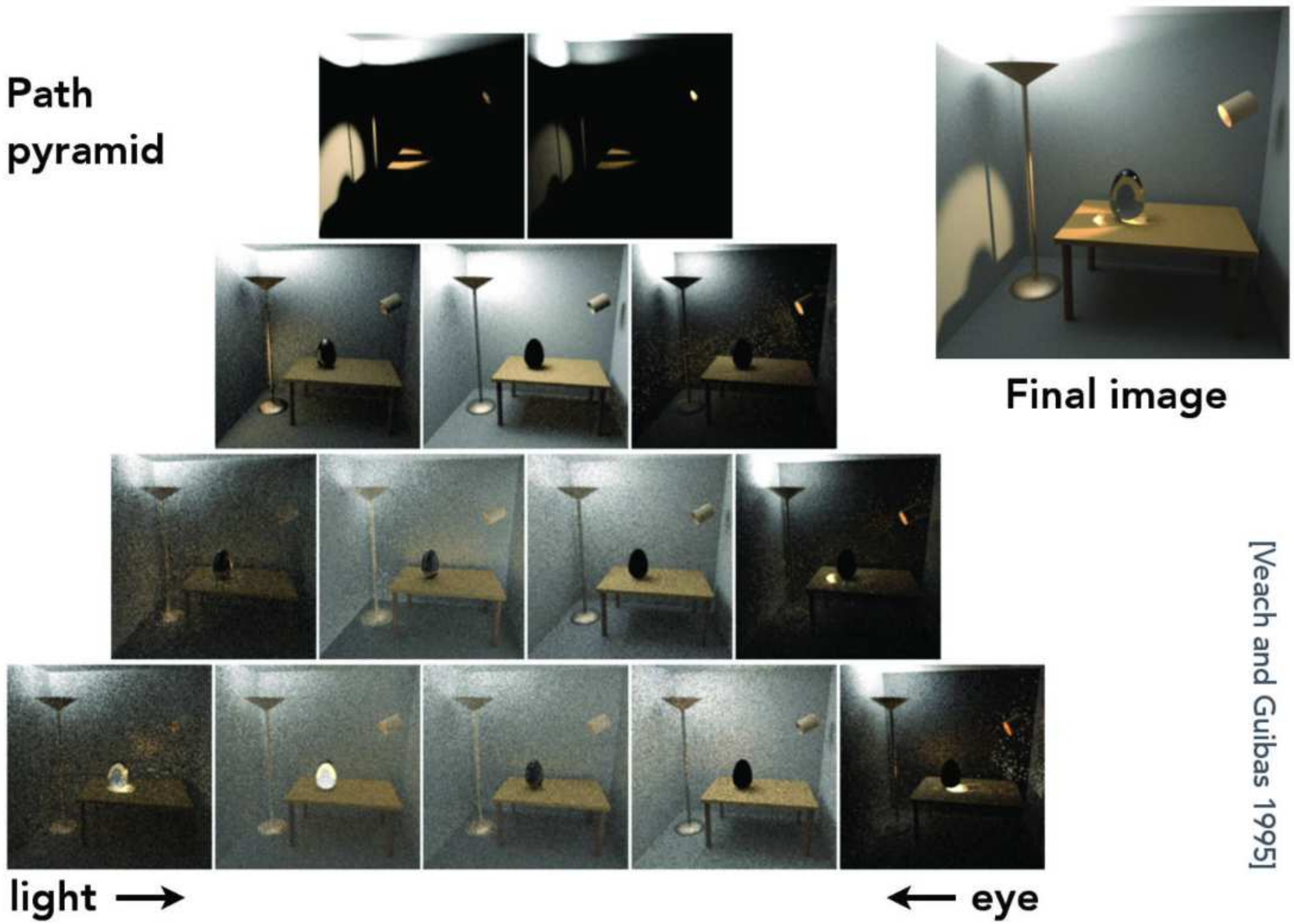
- **Comparison with traditional path tracing**

Bidirectional Path Tracing

Path Tracing

Bidirectional path tracing

- **Contributions of different path lengths**

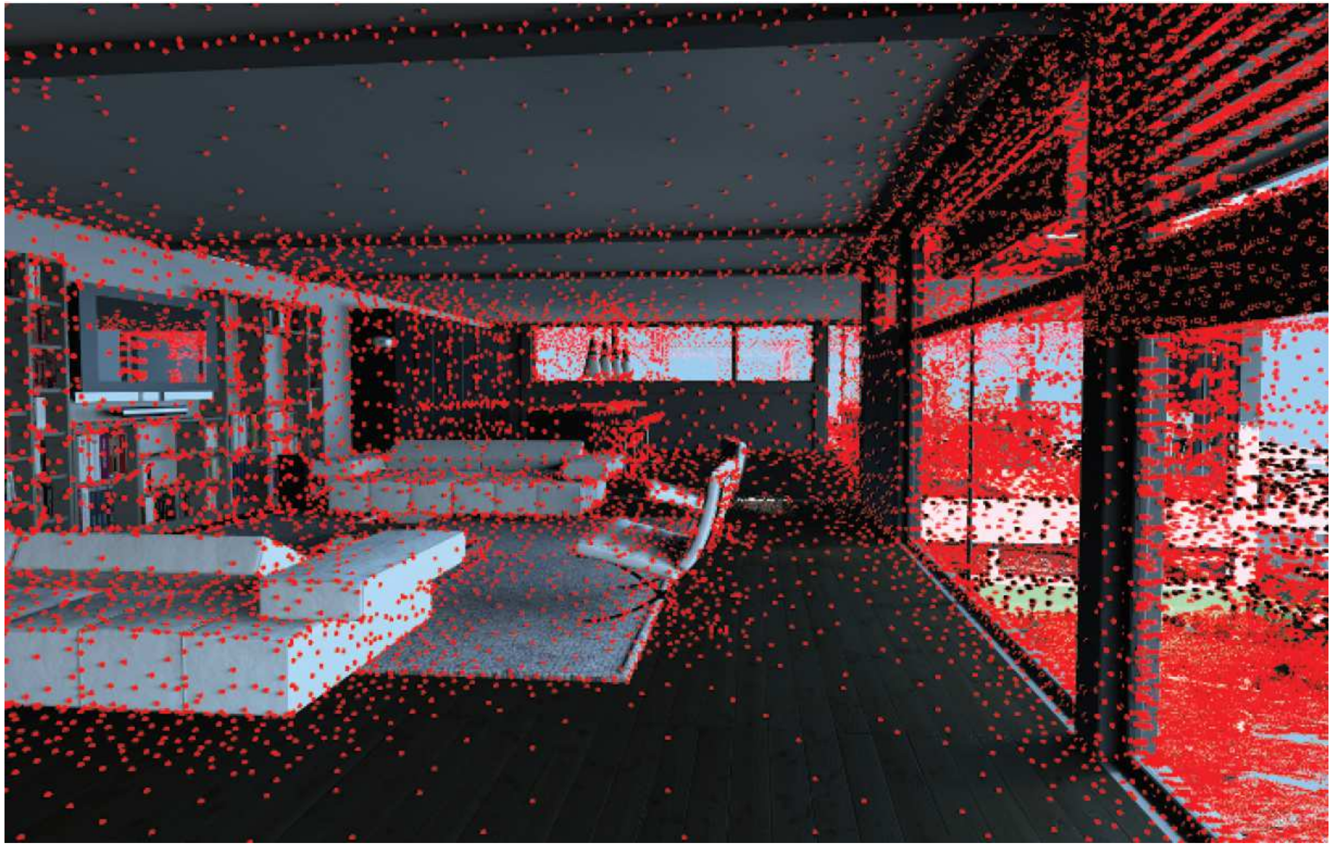


[Veach and Guibas 1995]

3. Photon Mapping

Irradiance cache

- **Store illumination at sparse points**
  - Approximate indirect illumination
  - Lookup cached illumination for radiance estimation



Photon mapping

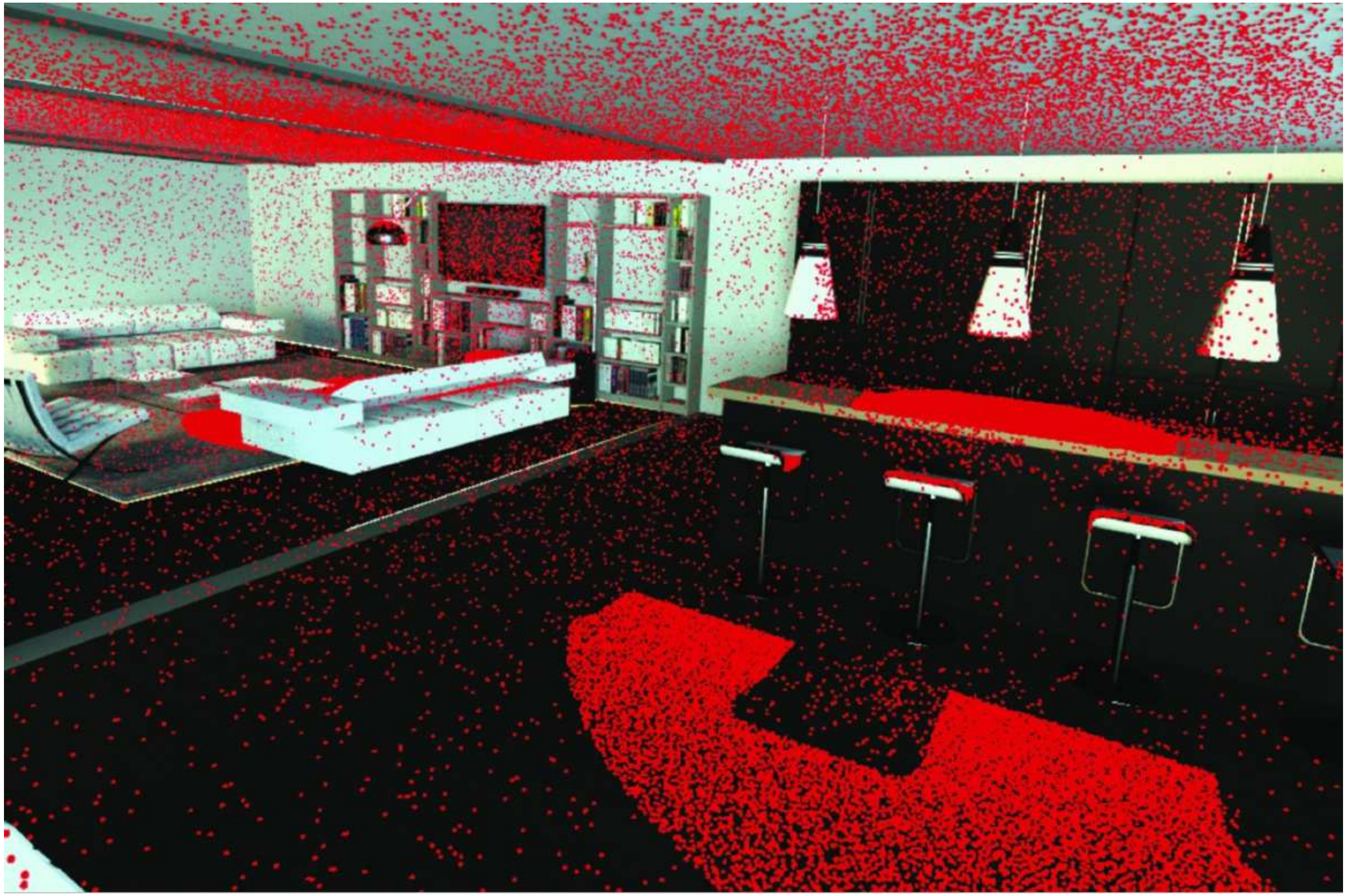
- **A two-pass global illumination algorithm**
  - Developed by Henrik Wann Jensen
  - Solve rendering equations approximately
  - Capable of simulating
    - Caustics, diffuse inter-reflections
    - Participating media
  - Same flexibility as general Monte-Carlo ray tracing
  - A fraction of the computation time

Photon mapping

- **Key idea**
  - Photon: energy carrying packets
  - A two-pass method
    - Similar as bidirectional path tracing
  - First pass
    - Build the photon map structure
    - Emitting photons from the light sources into the scene
    - Store them in a *photon map* when hitting non-specular objects
  - Second pass
    - Extract information about incoming flux and reflected radiance from photon map for camera ray radiance

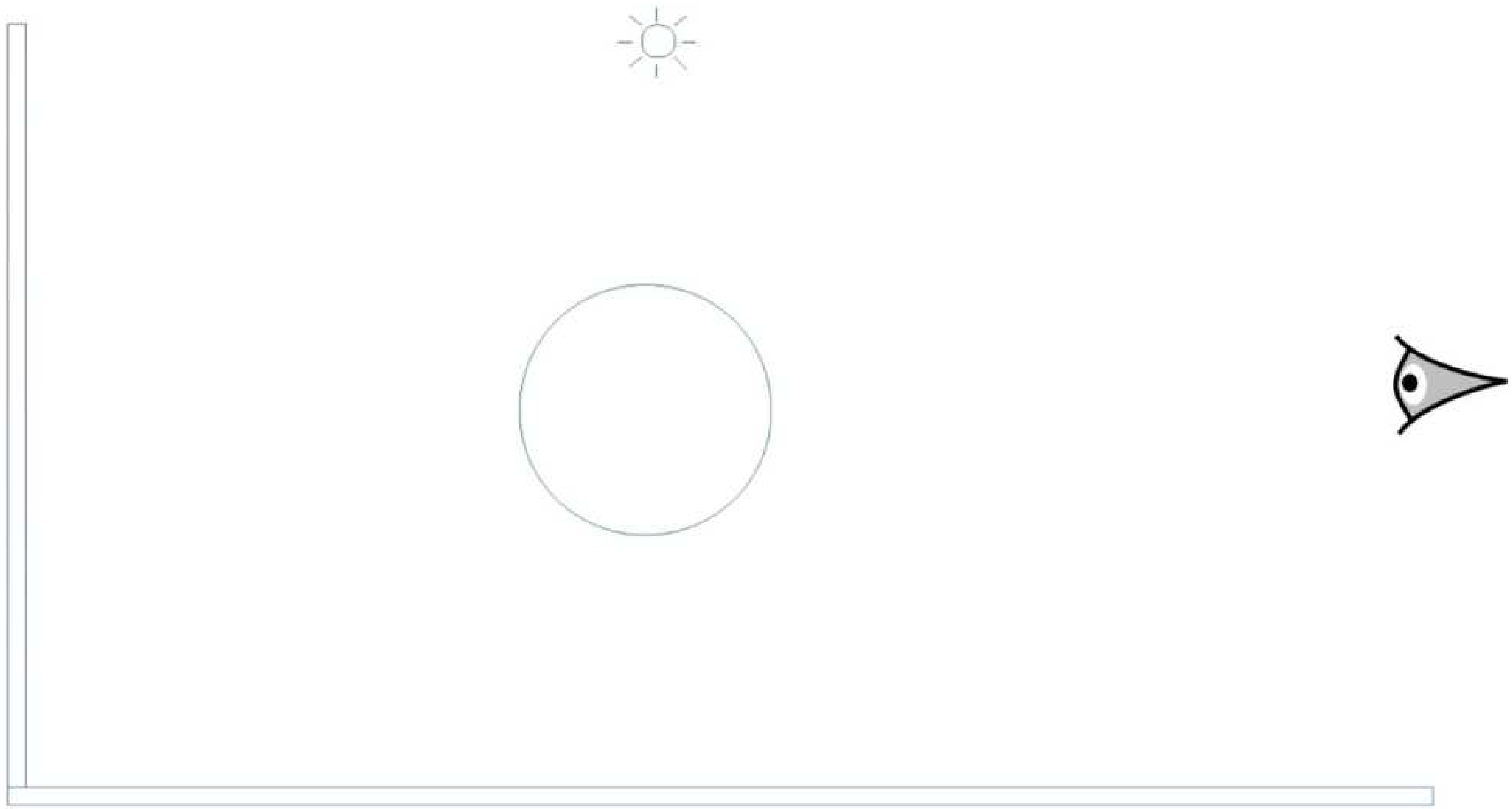
Photon mapping

- **Photon distribution**



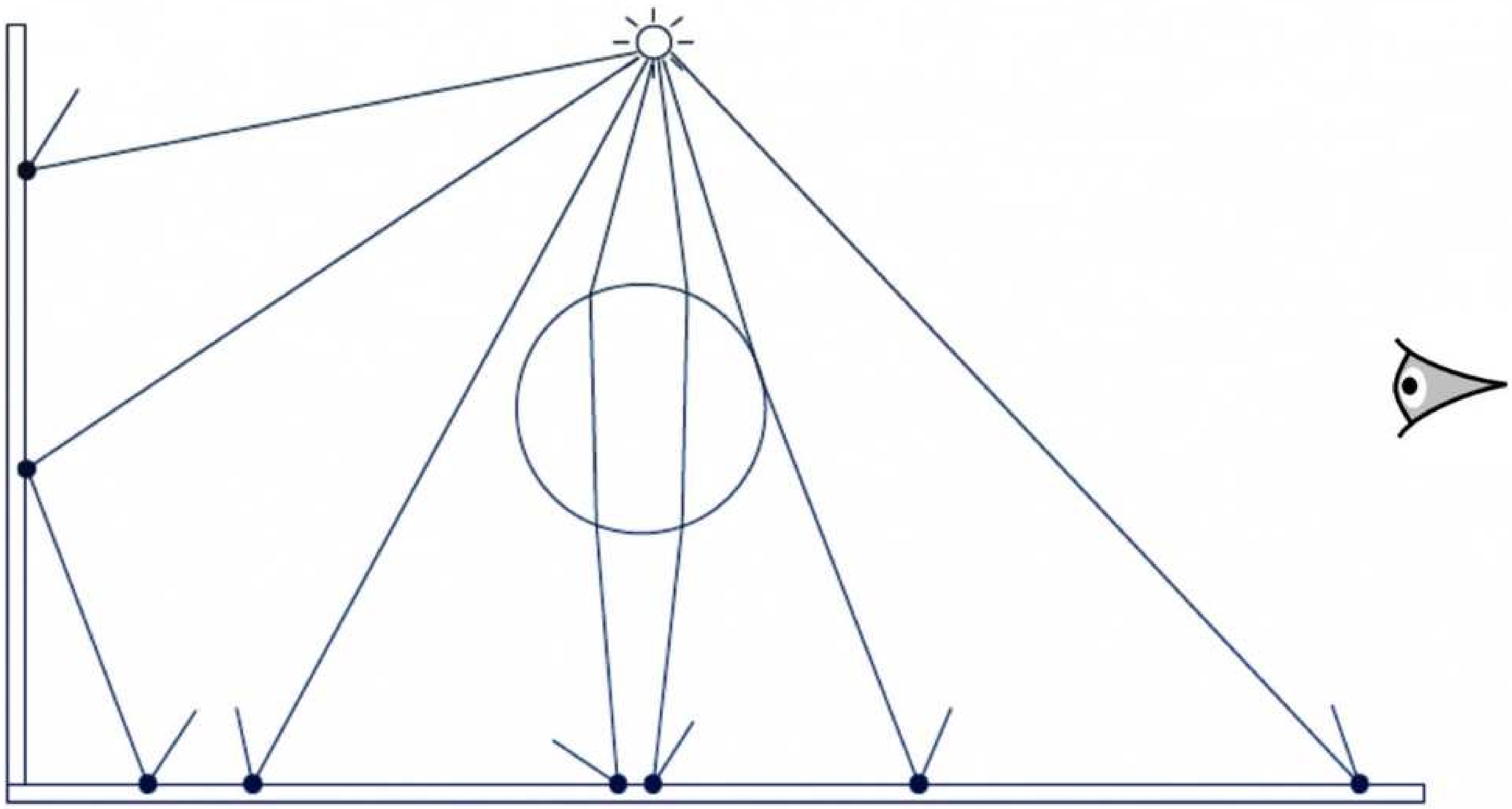
Photon mapping

- **A simple test scene**



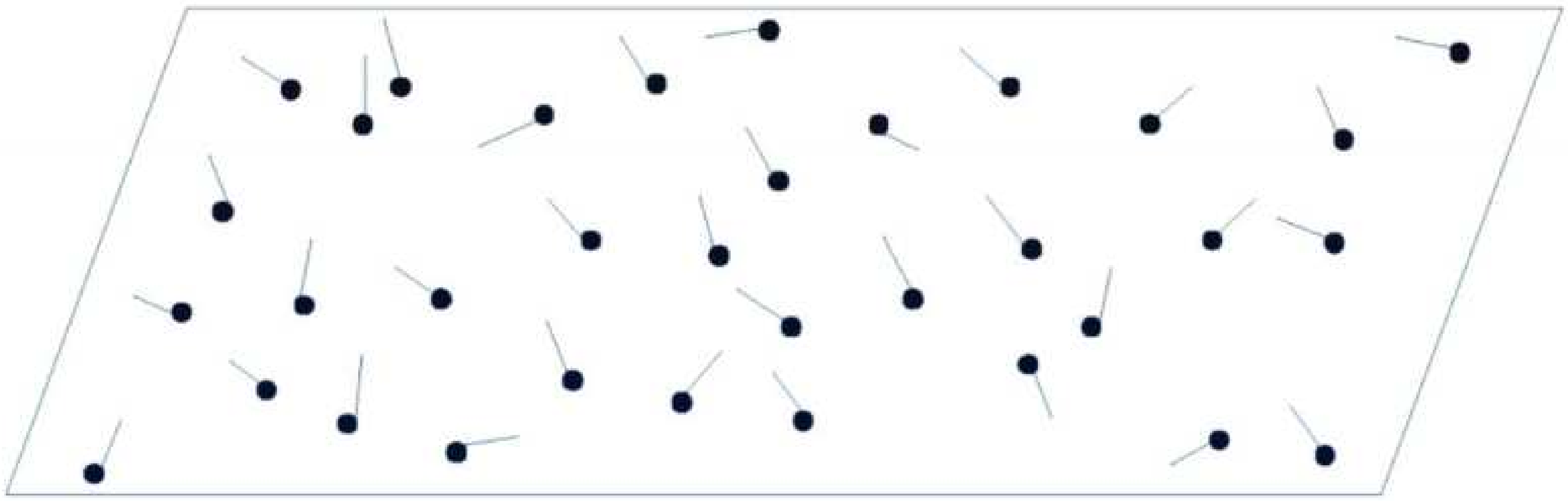
Photon mapping

- **Photon tracing**



Photon mapping

- **Photons on surface**

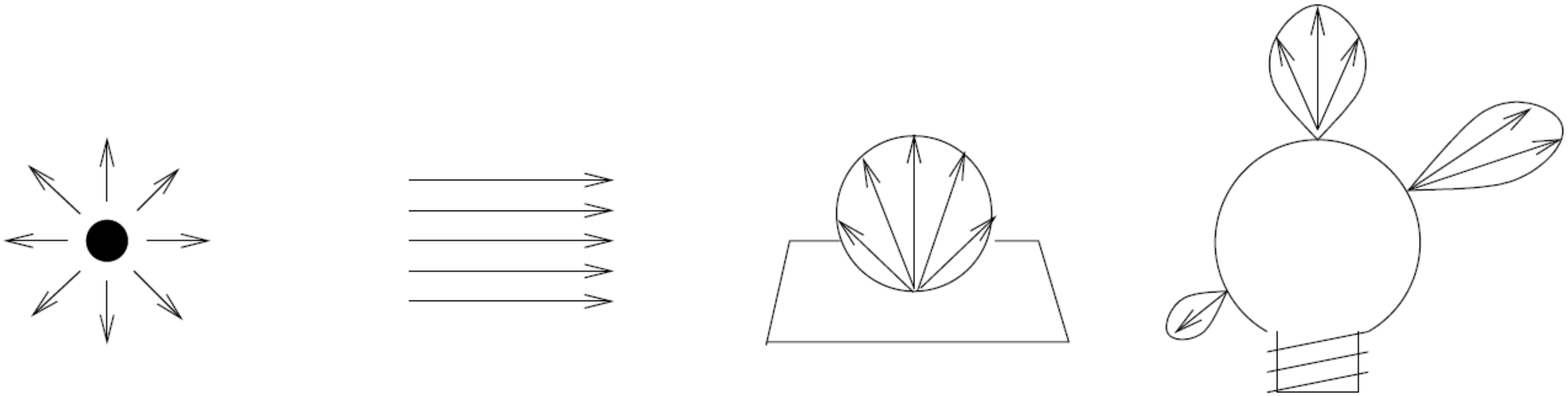


Photon mapping

- **Photo emission**
  - Assumption
    - Each photon is assumed to have the same energy
  - Diffuse point light source
    - Emitted in uniformly distributed random directions
  - Directional light
    - Emitted in the same direction
  - Diffuse square light source
    - Random positions on the square
    - Emission directions form a cosine distribution

Photon mapping

- **Photo emission**
  - In general
    - Arbitrary shape, arbitrary emission probability profile



    - Power of each emitted photon

$$P_{photon} = \frac{P_{light}}{n_e}$$

Photon mapping

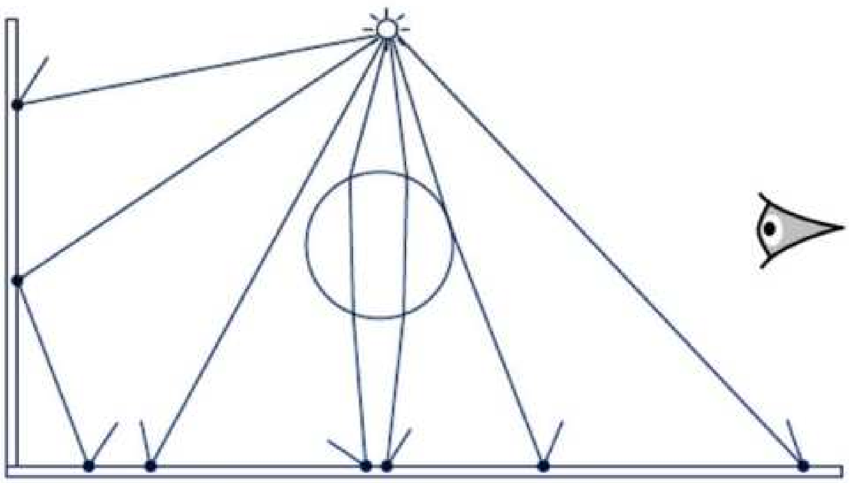
- **Photon tracing**
  - The reverse of ray tracing
    - Photo tracing: propagate flux
    - Ray tracing: gather radiance
  - Photon hitting an object
    - Reflected, transmitted, or absorbed
    - Decide probabilistically based on material parameters
    - Use Russian roulette
  - Decision making (with Russian roulette sampling)

Photon mapping

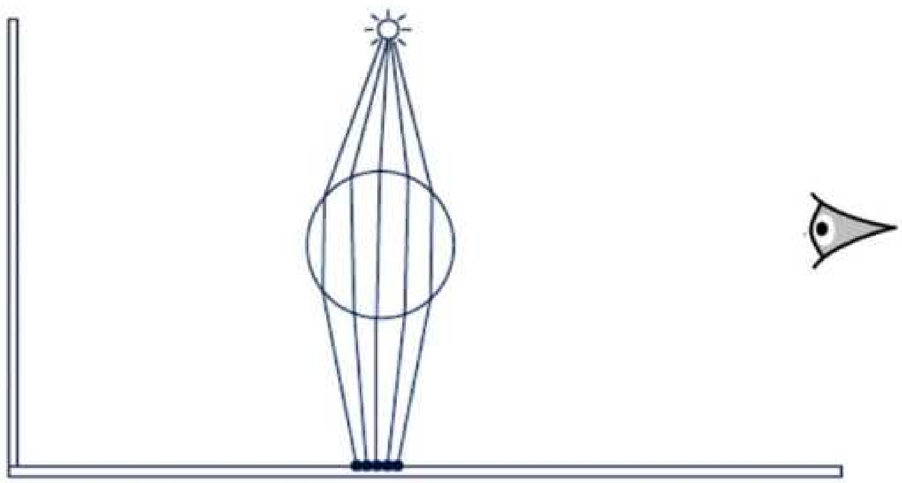
- **Photon storage**
  - Which photon-surface interactions are stored in the photon map?
    - Where they hit diffuse non-specular surfaces
  - Data is stored in a global data structure: photo map
  - What are stored?
    - Position, incoming photon power, incident direction

Photon mapping

- **Two photon maps**
  - Global photon map (low frequency)
  - Caustic photon map (high frequency)



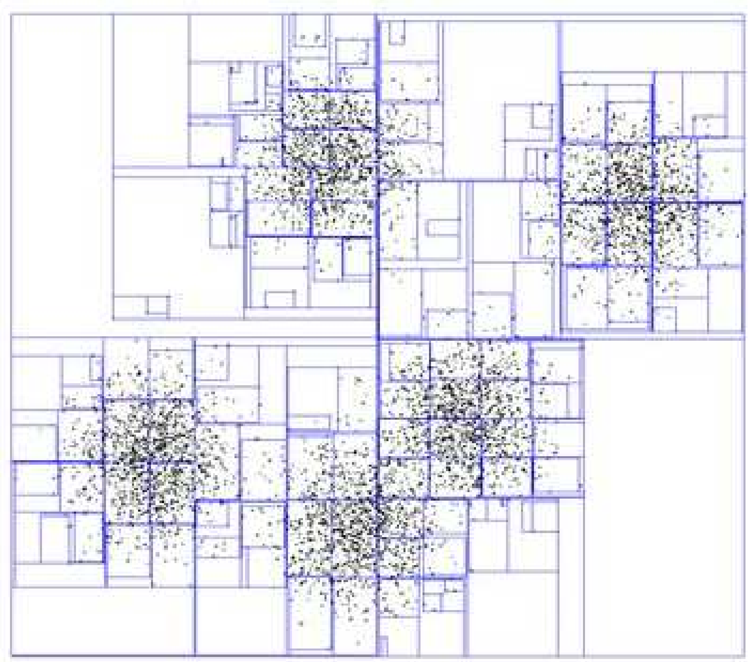
global photon map



caustics photon map

Photon mapping

- **Balanced k-d tree**
  - Divide the samples at the median
  - Efficiently find the neighbors for rendering



Photon mapping

- **Photon map creation**
  - Generate random path starting from the light sources
    - Randomly sample points and outgoing directions on lights
  - Follow paths through the scene
    - Find intersections with non-specular surfaces
    - At each surface intersection
      - Deposit a photon representing a sample of illumination at that point from the incident direction
  - Build a balanced kd tree
    - All photons are stored in a space-partitioning kd tree
    - Nearby photons can easily be located by local search of the tree

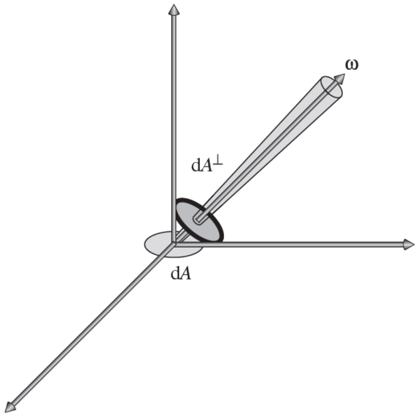
Photon mapping

- **Radiance estimate**
  - Fundamental radiance computation

$$L_r(x,\vec{\omega}) = \int_{\Omega_x} f_r(x,\vec{\omega}',\vec{\omega}) L_i(x,\vec{\omega}') |\vec{n}_x \cdot \vec{\omega}'| \, d\omega_i'$$

  - Rewrite incoming radiance in terms of photons
    - According to radiance definition

$$L_i(x,\vec{\omega}') = \frac{d\,\Phi_i(x,\vec{\omega}')}{\cos\theta_i \, d\omega_i' \, dA_i}$$



    - Rewrite the integral

$$\begin{aligned} L_r(x,\vec{\omega}) &= \int_{\Omega_x} f_r(x,\vec{\omega}',\vec{\omega}) \frac{d\,\Phi_i(x,\vec{\omega}')}{\cos\theta_i \, d\omega_i' \, dA_i} |\vec{n}_x \cdot \vec{\omega}'| \, d\omega_i' \\ &= \int_{\Omega_x} f_r(x,\vec{\omega}',\vec{\omega}) \frac{d\,\Phi_i(x,\vec{\omega}')}{dA_i} \,. \end{aligned}$$

Photon mapping

- **Radiance estimate**
  - Incoming flux is approximated using the photon map
  - Searching the nearest n photons
  - Each photon p has equal power (energy)

$$L_r(x,\vec{\omega}) = \int_{\Omega_x} f_r(x,\vec{\omega}',\vec{\omega}) \frac{d\Phi_i(x,\vec{\omega}')}{dA_i} \approx \sum_{p=1}^{n} f_r(x,\vec{\omega}_p,\vec{\omega}) \frac{\Delta\Phi_p(x,\vec{\omega}_p)}{\Delta A}$$

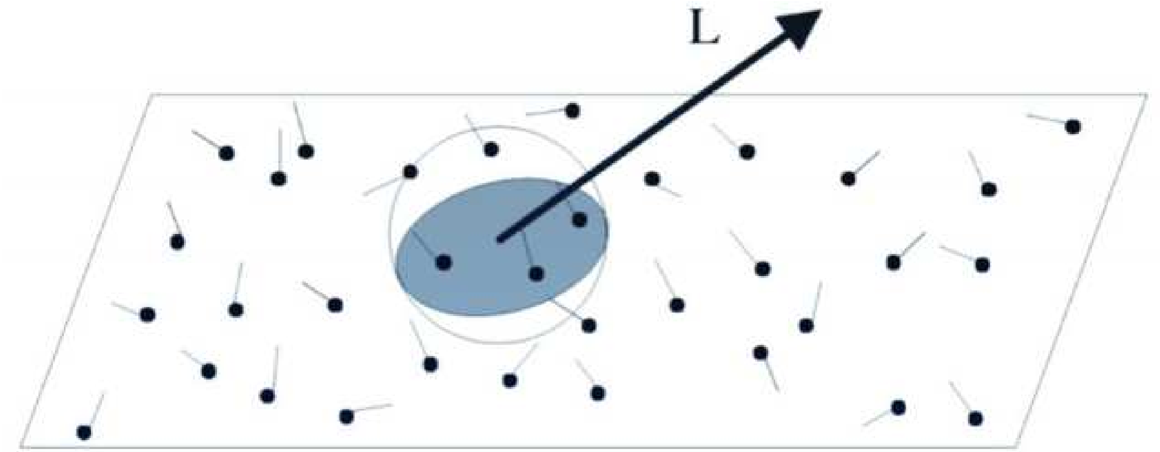
  - Assuming that the surface is locally flat
    - Projecting the sphere onto the tangent surface

$$\Delta A = \pi r^2$$

Photon mapping

- **Radiance estimate**
  - 1. Search nearest n photons around hit point
  - 2. Compute the reflected power by BRDF for each photon
  - 3. Divide the sum by projected area on tangent plane

$$L_r(x, \vec{\omega}) \approx \frac{1}{\pi r^2} \sum_{p=1}^{N} f_r(x, \vec{\omega}_p, \vec{\omega}) \Delta\Phi_p(x, \vec{\omega}_p)$$



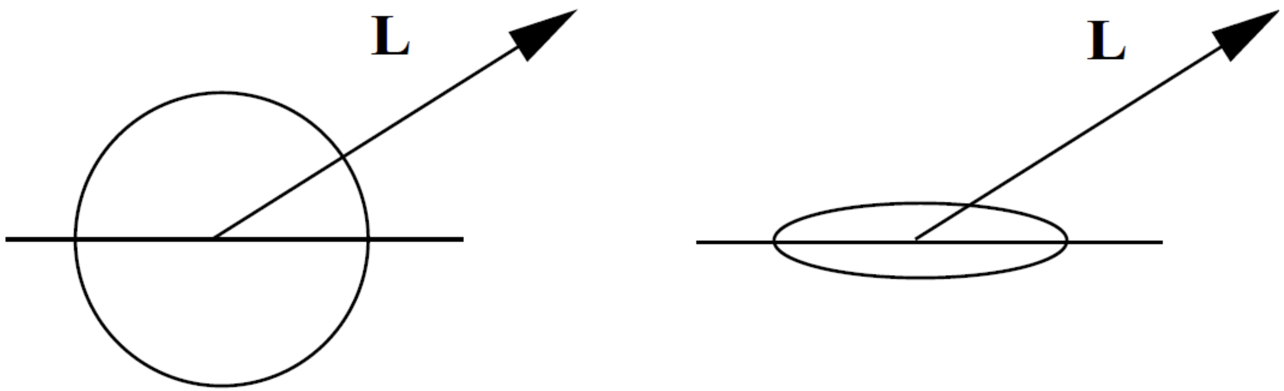
Photon mapping

- **Radiance estimate**
  - Photon mapping is not unbiased
    - Introducing statistical error
    - Reduced noise
  - BUT it is consistent
    - Converge as the number of photons goes to infinity

$$\lim_{N\to\infty} \frac{1}{\pi r^2} \sum_{p=1}^{\lfloor N^\alpha \rfloor} f_r(x, \vec{\omega}_p, \vec{\omega}) \Delta\Phi_p(x, \vec{\omega}_p) = L_r(x, \vec{\omega}) \quad \text{for } \alpha \in]0,1[$$

Photon mapping

- **Locate photons**
  - Symmetric v.s. non-symmetric
  - Non-symmetric for edges (more accurate)
  - Selection regions are geometry dependent



Photon mapping

- **Filtering**
  - To reduce the amount of blur at sharp edges
  - Useful for scenarios like caustics
  - Radiance estimate is filtered
    - Increase the weight of photons close to the hit point

- **Cone filter**
  - A cone-shape weight

$$w_{pc} = 1 - \frac{d_p}{k\,r} \qquad L_r(x,\vec{\omega}) \approx \frac{\sum_{p=1}^{N} f_r(x,\vec{\omega}_p,\vec{\omega})\Delta\Phi_p(x,\vec{\omega}_p)w_{pc}}{(1-\frac{2}{3k})\pi r^2}$$

Photon mapping

- **Gaussian filter**
  - Give good results when filtering caustics

$$w_{pg} = \alpha \left[1 - \frac{1 - e^{-\beta \frac{d_p^2}{2r^2}}}{1 - e^{-\beta}} \right]$$

$$\alpha = 0.918$$

$$\beta = 1.953$$

$$L_r(x, \vec{\omega}) \approx \sum_{p=1}^{N} f_r(x, \vec{\omega}_p, \vec{\omega}) \Delta\Phi_p(x, \vec{\omega}_p) w_{pg}$$

Photon mapping

- **Rendering based on photon map**
  - Outgoing radiance

$$L_o(x,\vec{\omega}) = L_e(x,\vec{\omega}) + L_r(x,\vec{\omega}) \qquad L_r(x,\vec{\omega}) = \int_{\Omega_x} f_r(x,\vec{\omega}',\vec{\omega}) L_i(x,\vec{\omega}') \cos\theta_i \, d\omega_i'$$

  - The BRDF is separated into a sum of two components
    - Specular/glossy + diffuse

$$f_r(x,\vec{\omega}',\vec{\omega}) = f_{r,s}(x,\vec{\omega}',\vec{\omega}) + f_{r,d}(x,\vec{\omega}',\vec{\omega})$$

Photon mapping

- **Rendering based on photon map**
  - Incoming radiance classification
    - Direct illumination $L_{i,l}(x, \vec{\omega}')$
    - Caustics $L_{i,c}(x, \vec{\omega}')$
      - Indirect illumination from the light sources via specular reflection or transmission
    - Diffuse indirect illumination $L_{i,d}(x, \vec{\omega}')$
  - Incoming light

$$L_i(x, \vec{\omega}') = L_{i,l}(x, \vec{\omega}') + L_{i,c}(x, \vec{\omega}') + L_{i,d}(x, \vec{\omega}')$$

Photon mapping

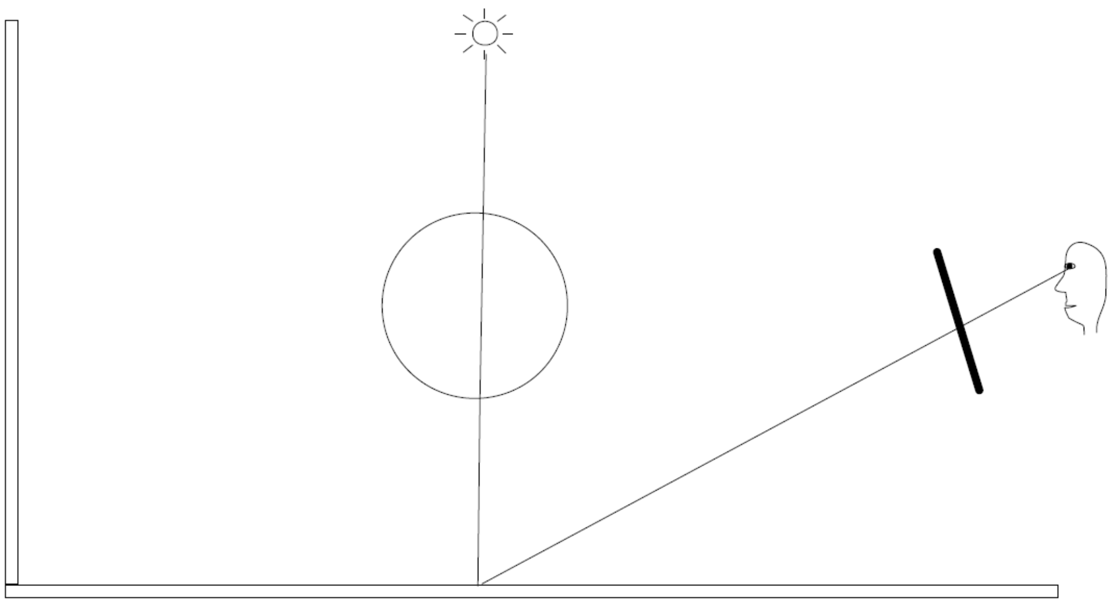
- **Reflected radiance splitting**
  - Reflected radiance splitting

$$
\begin{aligned}
L_r(x,\vec{\omega}) &= \int_{\Omega_x} f_r(x,\vec{\omega}',\vec{\omega}) L_i(x,\vec{\omega}') \cos\theta_i \, d\omega_i' \\
&= \int_{\Omega_x} f_r(x,\vec{\omega}',\vec{\omega}) L_{i,l}(x,\vec{\omega}') \cos\theta_i \, d\omega_i' + \\
&\quad \int_{\Omega_x} f_{r,s}(x,\vec{\omega}',\vec{\omega}) (L_{i,c}(x,\vec{\omega}') + L_{i,d}(x,\vec{\omega}')) \cos\theta_i \, d\omega_i' + \\
&\quad \int_{\Omega_x} f_{r,d}(x,\vec{\omega}',\vec{\omega}) L_{i,c}(x,\vec{\omega}') \cos\theta_i \, d\omega_i' + \\
&\quad \int_{\Omega_x} f_{r,d}(x,\vec{\omega}',\vec{\omega}) L_{i,d}(x,\vec{\omega}') \cos\theta_i \, d\omega_i' \, .
\end{aligned}
$$

Photon mapping

- **Direct illumination**
  - Standard distributed ray tracing
  - Multiple shadow/light rays are shot

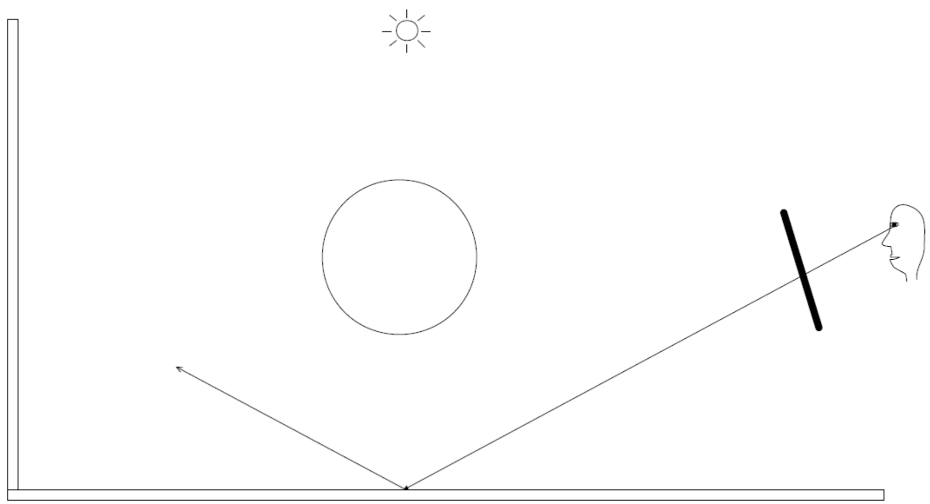
$$\int_{\Omega_x} f_r(x, \vec{\omega}', \vec{\omega}) L_{i,l}(x, \vec{\omega}') \cos\theta_i \, d\omega_i'$$



Photon mapping

- **Specular and glossy reflection**
  - Strongly dominated by $f_{r,s}$ which has a narrow peak around the mirror direction
  - Standard Monte Carlo ray tracing, sampling based on $f_{r,s}$

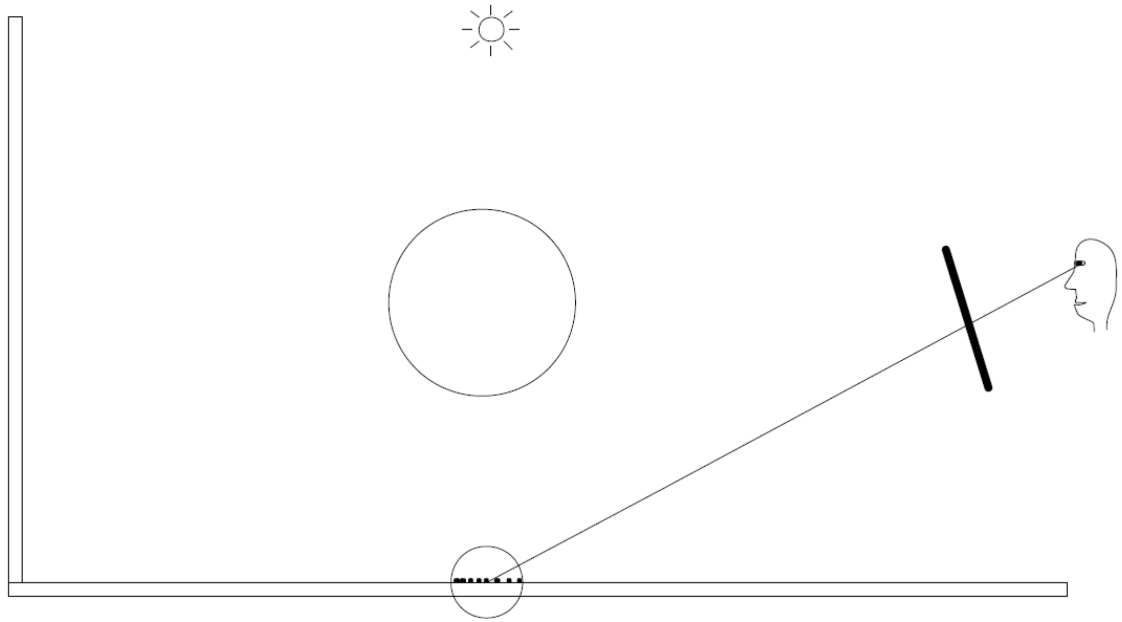
$$\int_{\Omega_x} f_{r,s}(x, \vec{\omega}', \vec{\omega})(L_{i,c}(x, \vec{\omega}') + L_{i,d}(x, \vec{\omega}')) \cos\theta_i \, d\omega_i'$$



Photon mapping

- **Caustics**
  - Solved by using a radiance estimate from the caustics photon map

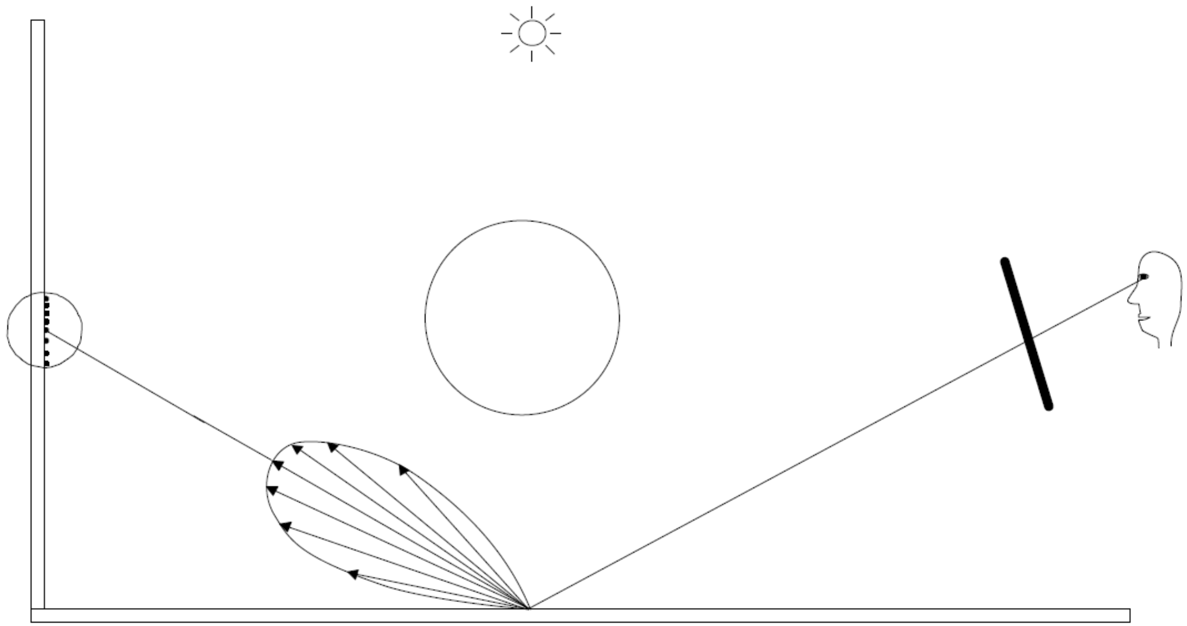
$$\int_{\Omega_x} f_{r,d}(x,\vec{\omega}',\vec{\omega})L_{i,c}(x,\vec{\omega}')\cos\theta_i \, d\omega_i'$$



Photon mapping

- **Multiple diffuse reflections**
  - The approximate evaluation
  - Based on the global photon map

$$\int_{\Omega_x} f_{r,d}(x, \vec{\omega}', \vec{\omega}) L_{i,d}(x, \vec{\omega}') \cos\theta_i \, d\omega_i'$$



Photon mapping

- **Bias and consistency in estimators**

Unbiased: $$E[X] = \int_a^b f(x)\,dx$$

- **Example:** $$\frac{1}{N}\sum_i^N f(x_i)$$

Consistent: $$\lim_{N\to\infty} E[X] = \int_a^b f(x)\,dx$$

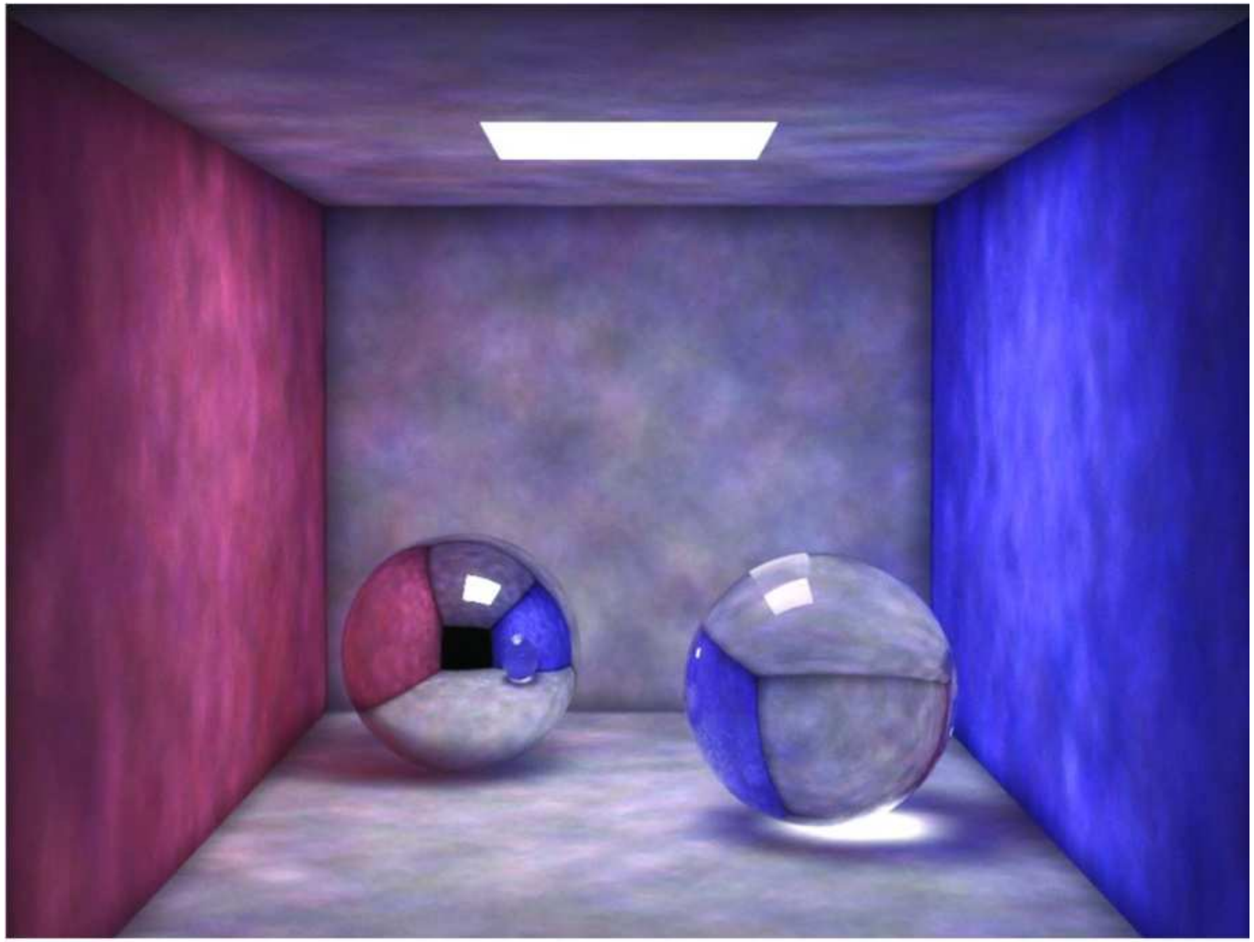
- **Example:** $$\frac{1}{N+1}\sum_i^N f(x_i)$$

Photon mapping

- **Biased v.s. consistent estimators**
  - Graphical interpretation
    - Consistent : the image approaches correct solution as some parameter is increased
    - Unbiased : produces correct result on average
  - Potential value of biased but consistent estimators
    - May have lower variance
    - May look better (less noise)

Photon mapping

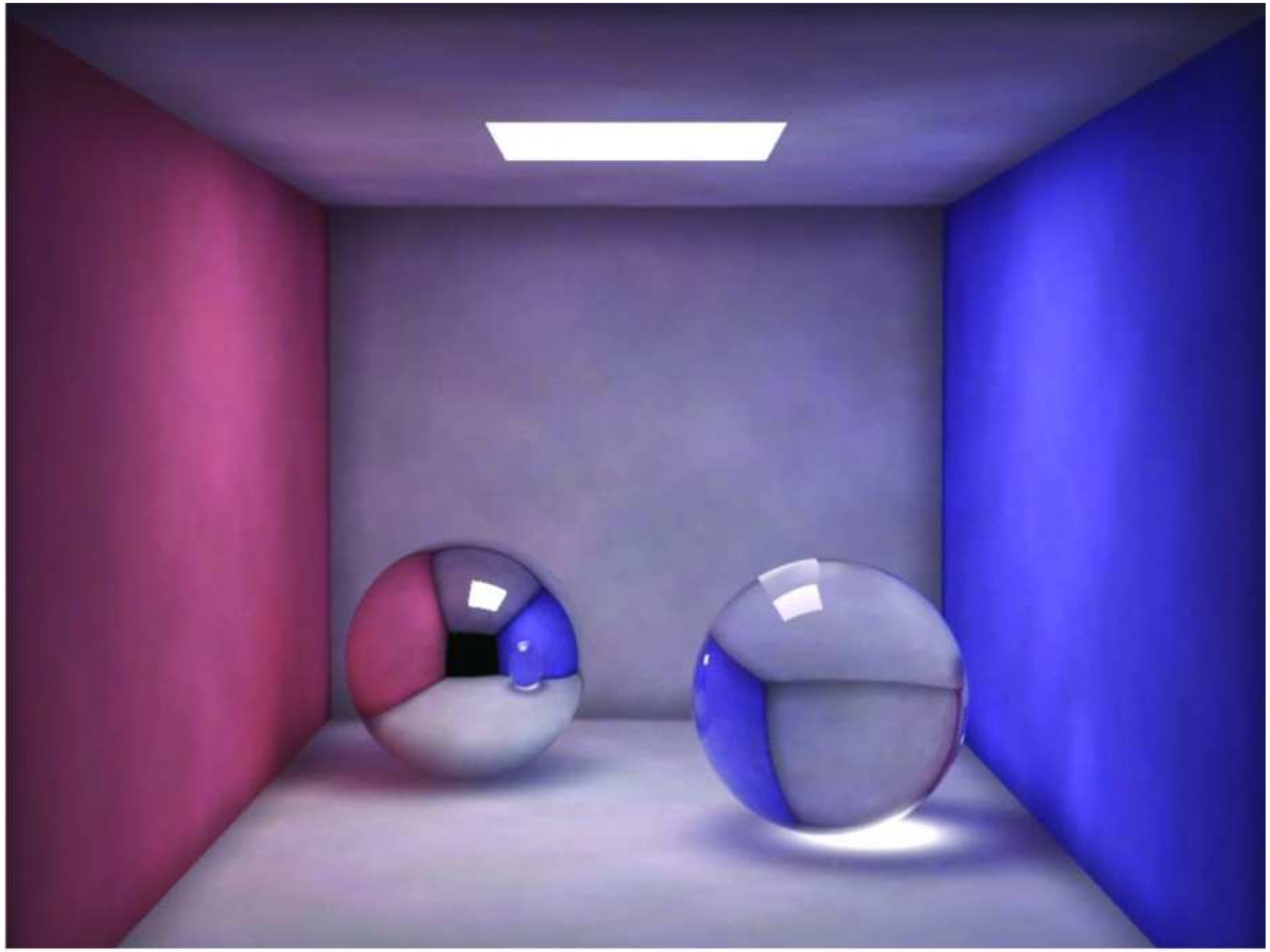
- **Biased but consistent estimators**



100000 photons, 50 photons in radiance estimate

Photon mapping

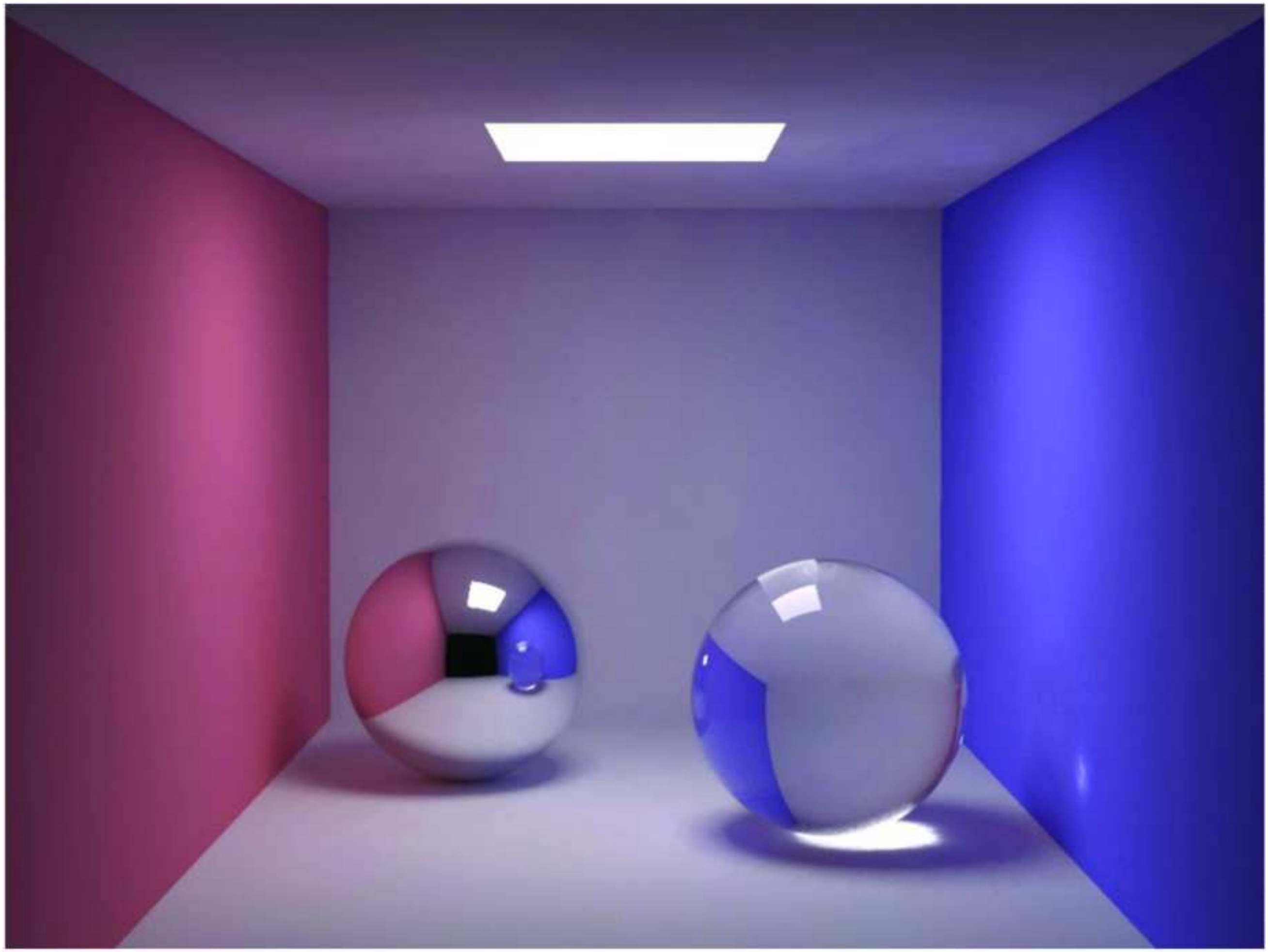
- **Biased but consistent estimators**



500000 photons, 500 photons in radiance estimate

Photon mapping

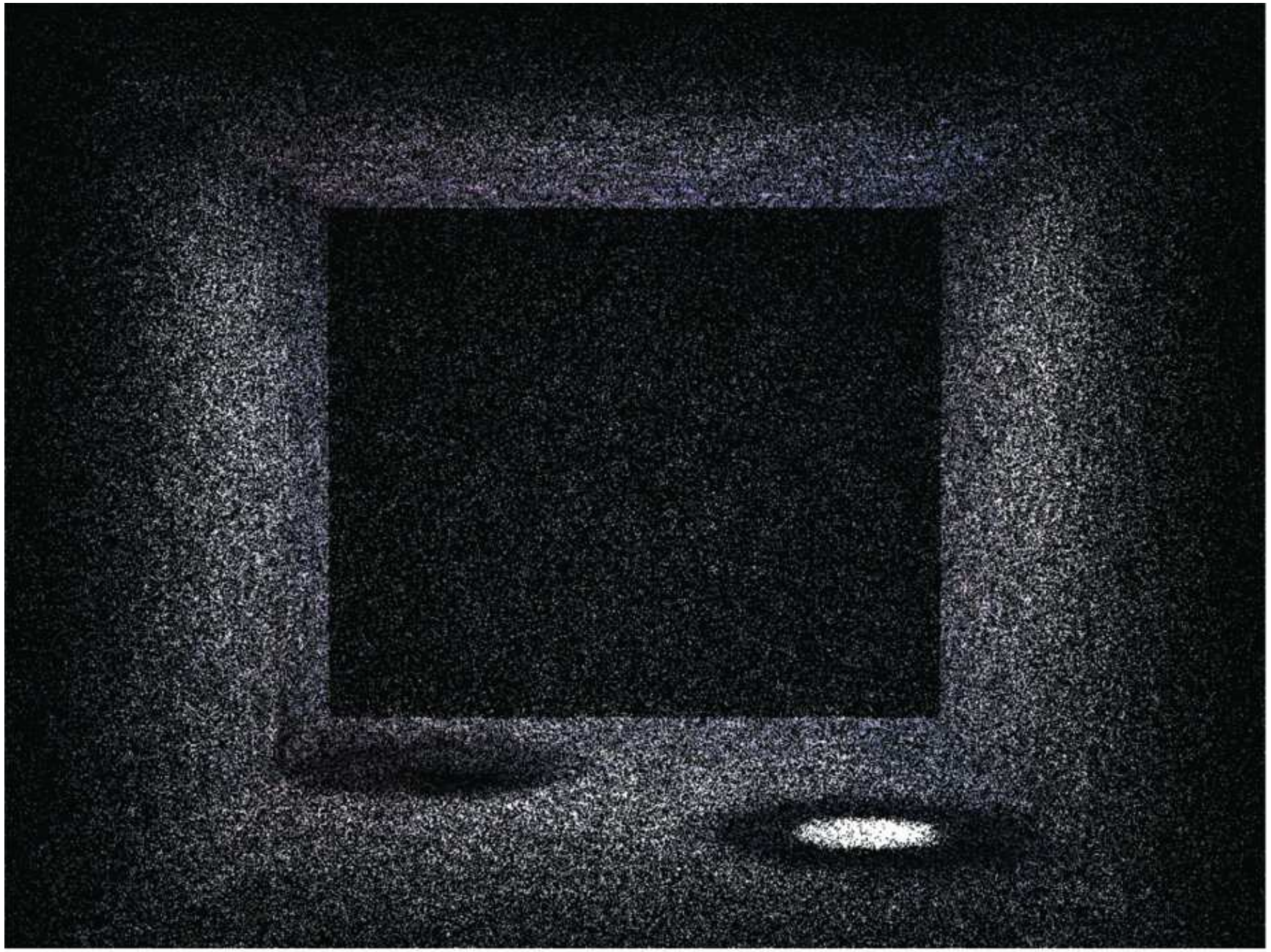
- **Cornell box**



200000 global photons, 50000 caustic photons

Photon mapping

- **Cornell box: photons**



200000 global photons

Photon mapping

- **Caustics from a glass sphere**

Photon mapping: 10000 photons, 50 photons in estimate

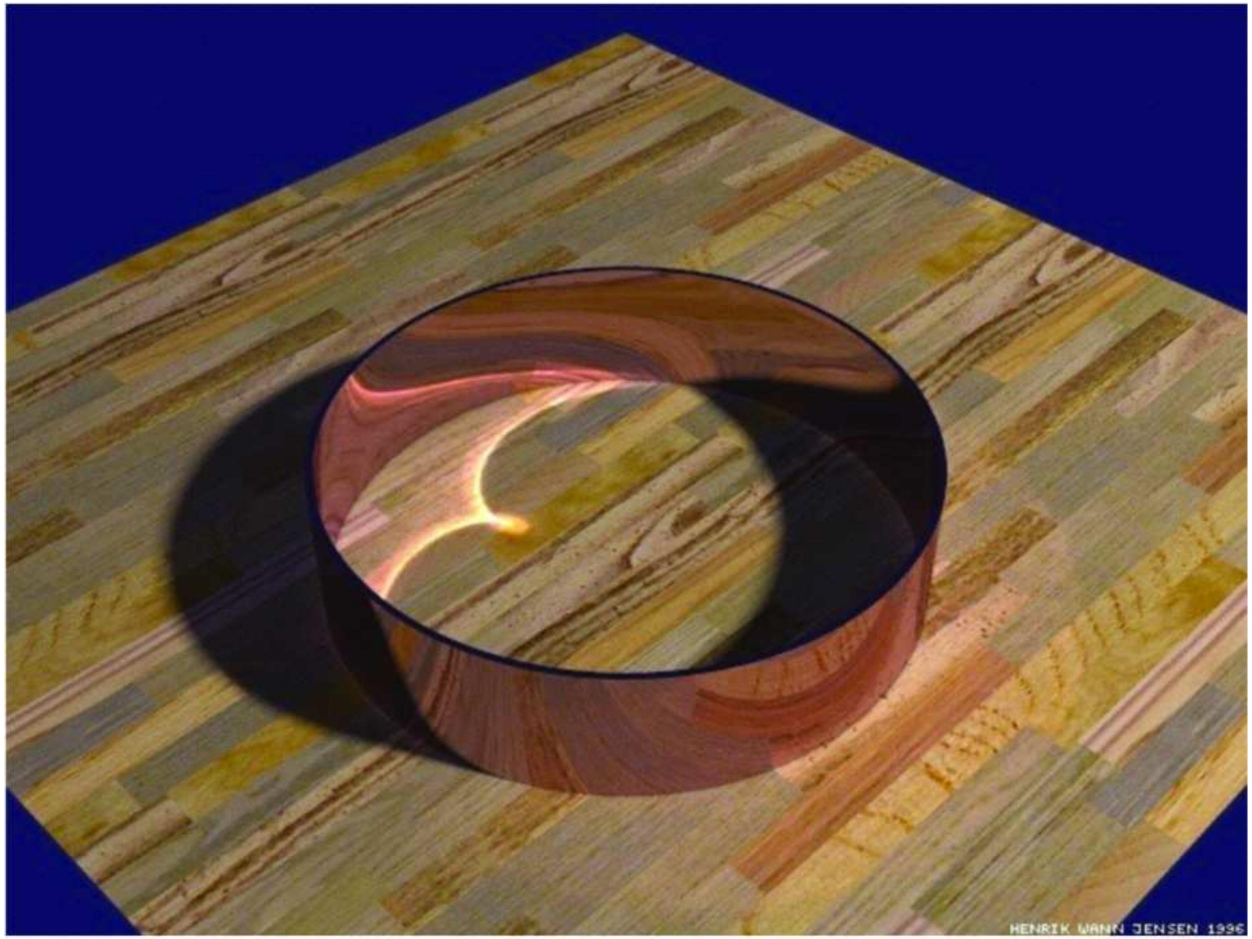
Photon mapping

- **Caustics from a glass sphere**

Path tracing: 1000 paths/pixels

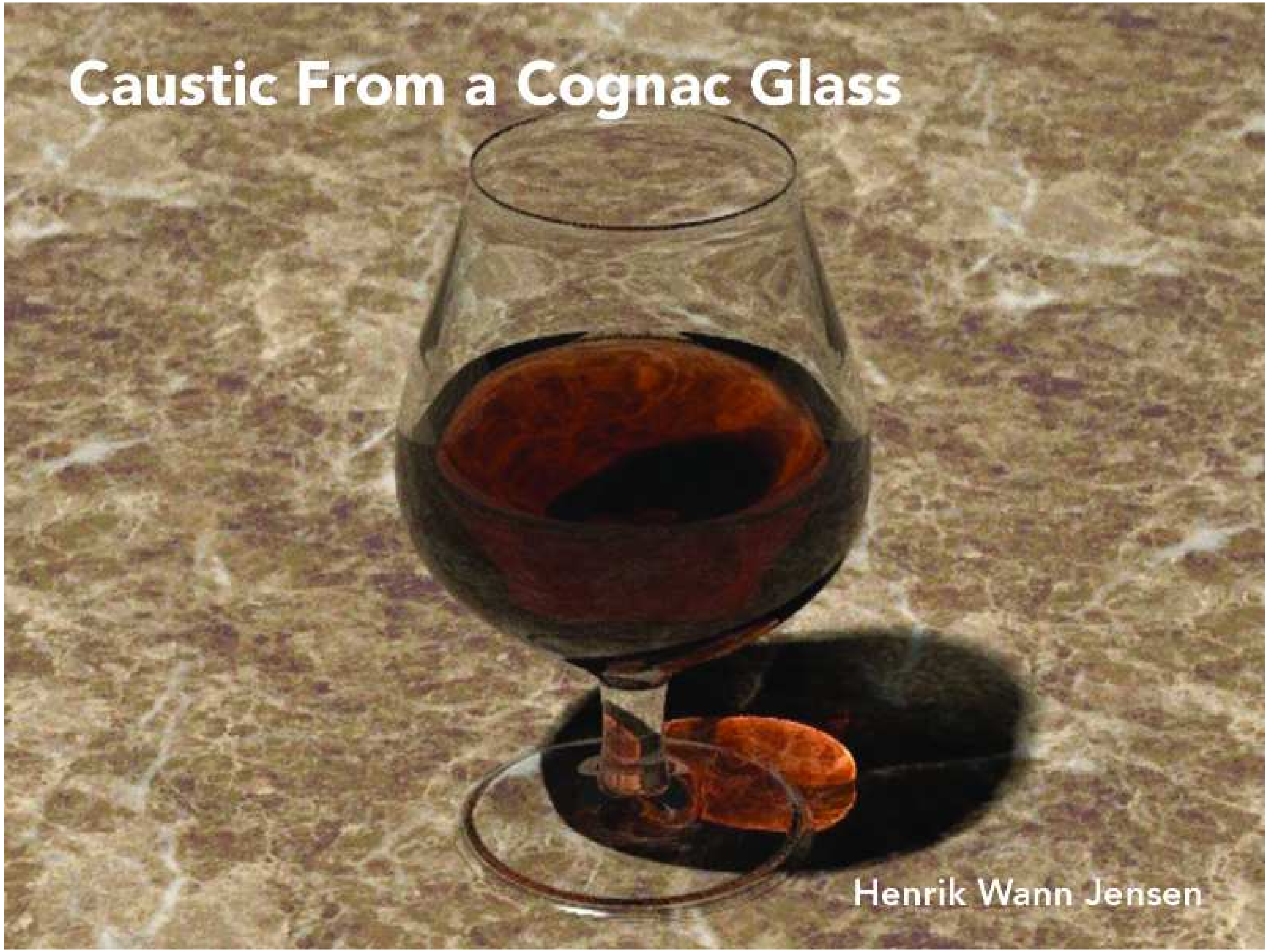
Photon mapping

- **Reflection inside a metal ring**



50000 photons, 50 photons in radiance estimate

Photon mapping



Photon mapping

- **Example code**
  - https://github.com/Mikepicker/PhotonMapping/blob/master/PhotonMapping/main.cpp

```cpp
//--------------Parameters--------------
//--------------------------------------
#define PHOTON_MAPPING                  // define for Photon Mapping, undefine for Ray Tracing
#define WINDOW_WIDTH 512                // Window width
#define WINDOW_HEIGHT 512               // Window height
#define MAX_RAY_DEPTH 5                 // Max recursive depth
#define PHOTONS 500                             // Number of photons
#define CAUSTICS_PHOTONS 20000  // Number of photons for caustic objects
#define LIGHT_POWER 500                 // Lights power
#define ESTIMATE 100                                            // Number of nearest neighbors
#define CAUSTICS_ESTIMATE 500
#define EXPOSURE 11

//------------------------------------
//--------------Vector3--------------
//------------------------------------
template<typename T>
class Vec3
{
public:
	T x, y, z;
	Vec3() : x(T(0)), y(T(0)), z(T(0)) {}
	Vec3(T xx) : x(xx), y(xx), z(xx) {}
	Vec3(T xx, T yy, T zz) : x(xx), y(yy), z(zz) {}
	Vec3& normalize()
	{
		T nor2 = length2();
		if (nor2 > 0) {
			T invNor = 1 / sqrt(nor2);
			x *= invNor, y *= invNor, z *= invNor;
		}
		return *this;
	}

	// clam values under/above min/max
	Vec3& gate(T min, T max)
	{
		if (x < min)
			x = min;
		if (y < min)
			y = min;
		if (z < min)
```

```cpp
//--------------------------------------
//--------------Geom Object--------------
//--------------------------------------
template<typename T>
class GeomObject {

public:
	Vec3<T> surfaceColor, emissionColor;    /// surface color and emission (light)
	T transparency, reflection;             /// surface transparency and reflectivity

	GeomObject(const Vec3<T> &sc, const T &refl = 0, const T &transp = 0, const Vec3<T> &ec = 0)
		: surfaceColor(sc), reflection(refl), transparency(transp), emissionColor(ec)
	{}

	// get object position
	virtual Vec3<T> getPosition() = 0;
	virtual bool intersect(const Vec3<T> &rayOrig, const Vec3<T> &rayDir, Vec3<T>* pHit = NULL, Vec3<T>* nHit = NULL) const = 0;
	virtual Vec3<T> computeBRDF() const = 0;
	virtual Vec3<T> randomPoint() const = 0;
};

//--------------------------------
//--------------Plane--------------
//--------------------------------
template<typename T>
class Plane : public GeomObject<T> {

public:
	Vec3<T> position;               // plane position
	Vec3<T> normal;         // vector normal to the plane

	Plane(const Vec3<T> &p, const Vec3<T> &n, const Vec3<T> &sc,
		const T &refl = 0, const T &transp = 0, const Vec3<T> &ec = 0) :
		position(p), normal(n), GeomObject(sc, refl, transp, ec)
	{}

	// get plane position
	Vec3<T> getPosition() { return position; }
```

Real-time ray tracer

- **NVIDIA OptiX:**
  - GPU-based programmable ray tracer

Credit: NVIDIA (this ray-traced image can be rendered at interactive rates on modern GPUs)

Next lecture: Volume rendering 1