CS 87/187, Spring 2016

Rendering Algorithms

Photon Mapping

Prof. Wojciech Jarosz

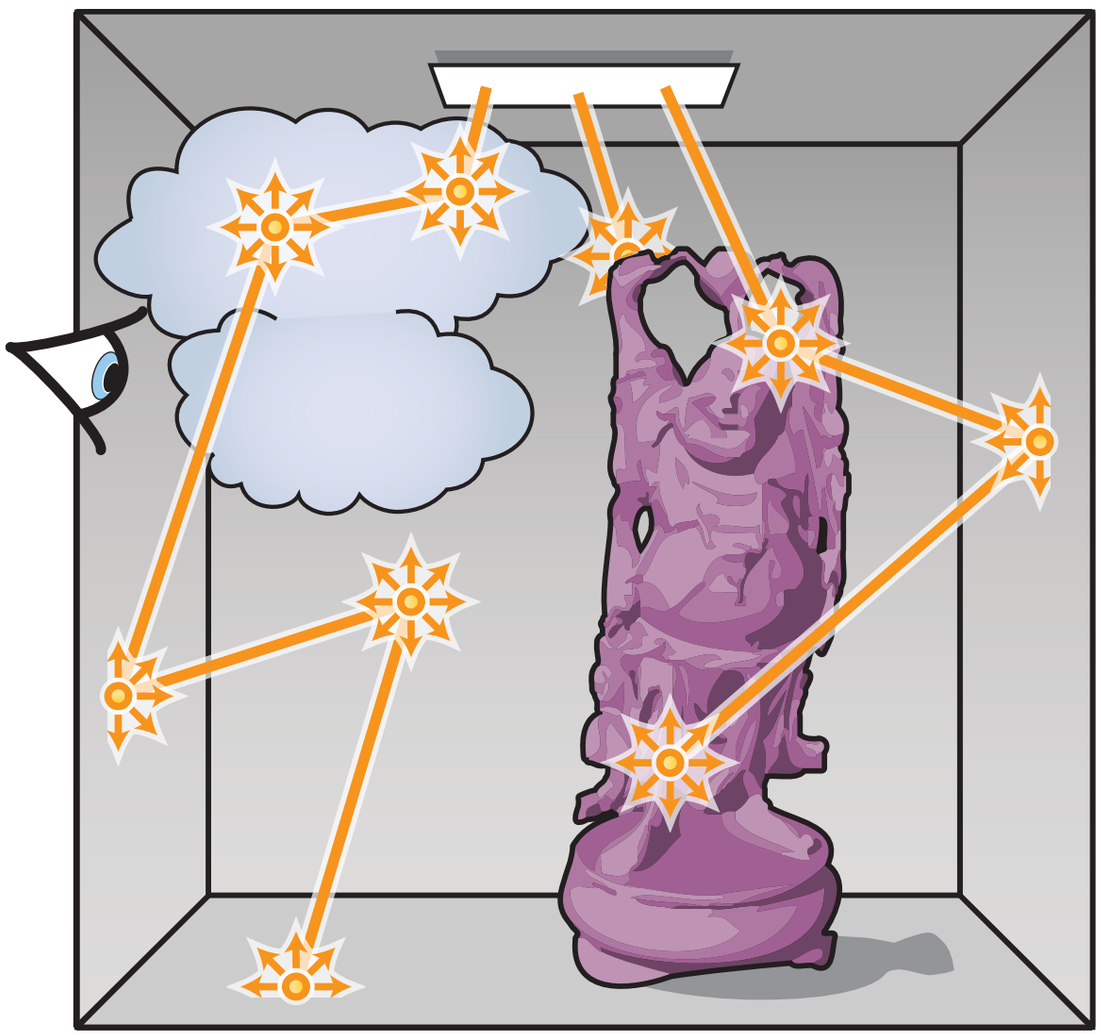
wojciech.k.jarosz@dartmouth.edu

(with some slides by Jan Novák)

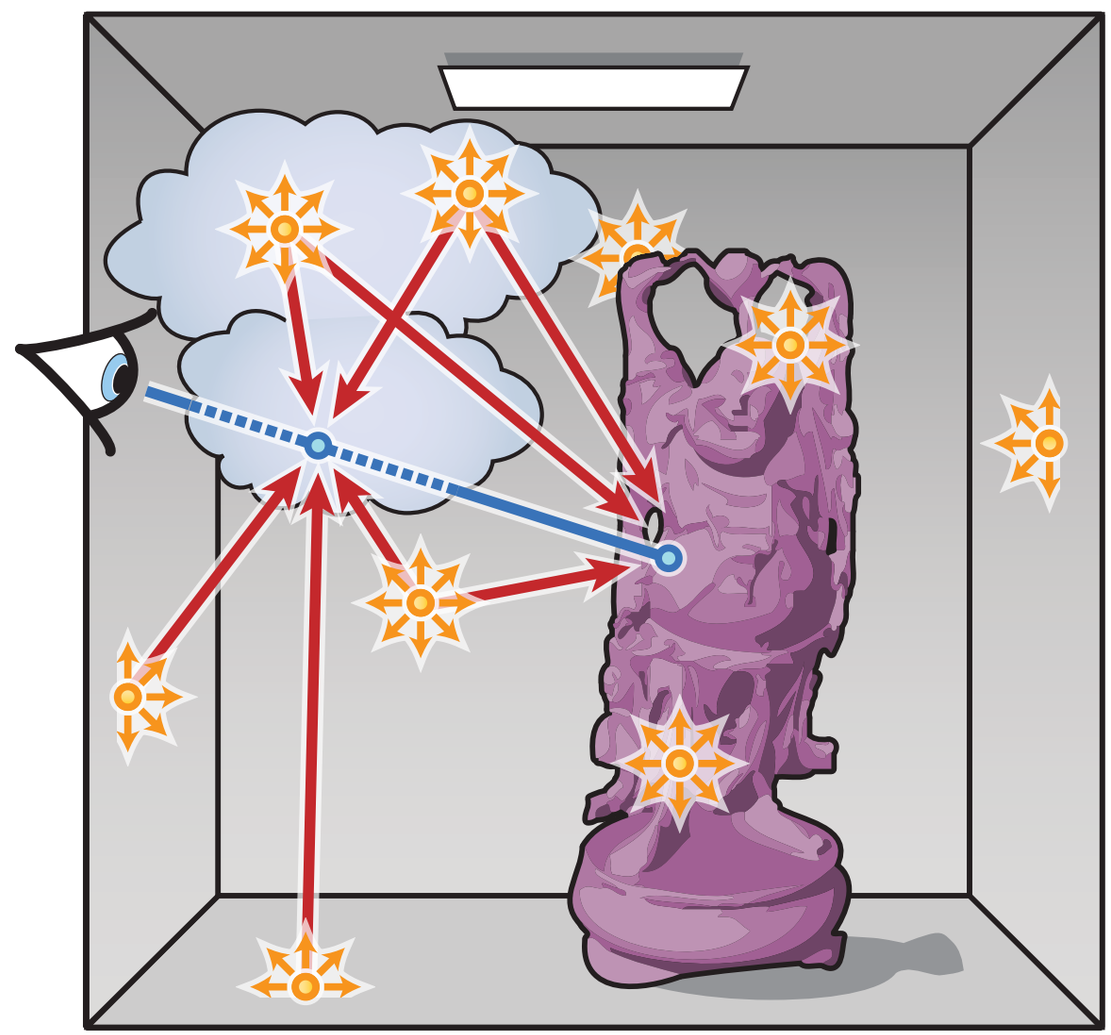
Dartmouth

Last Time: Many-Light Rendering

Generation of VPLs

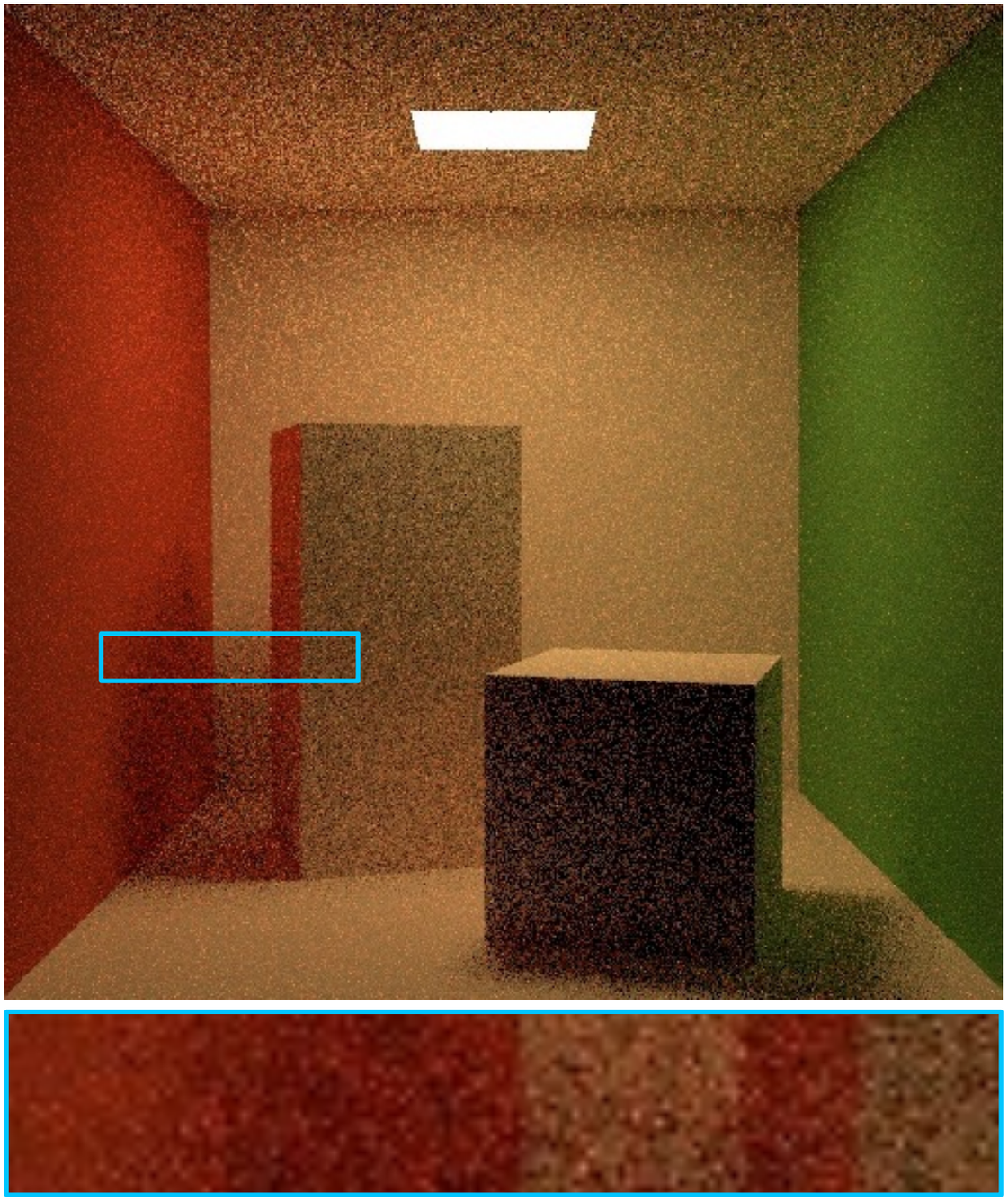


Lighting with VPLs

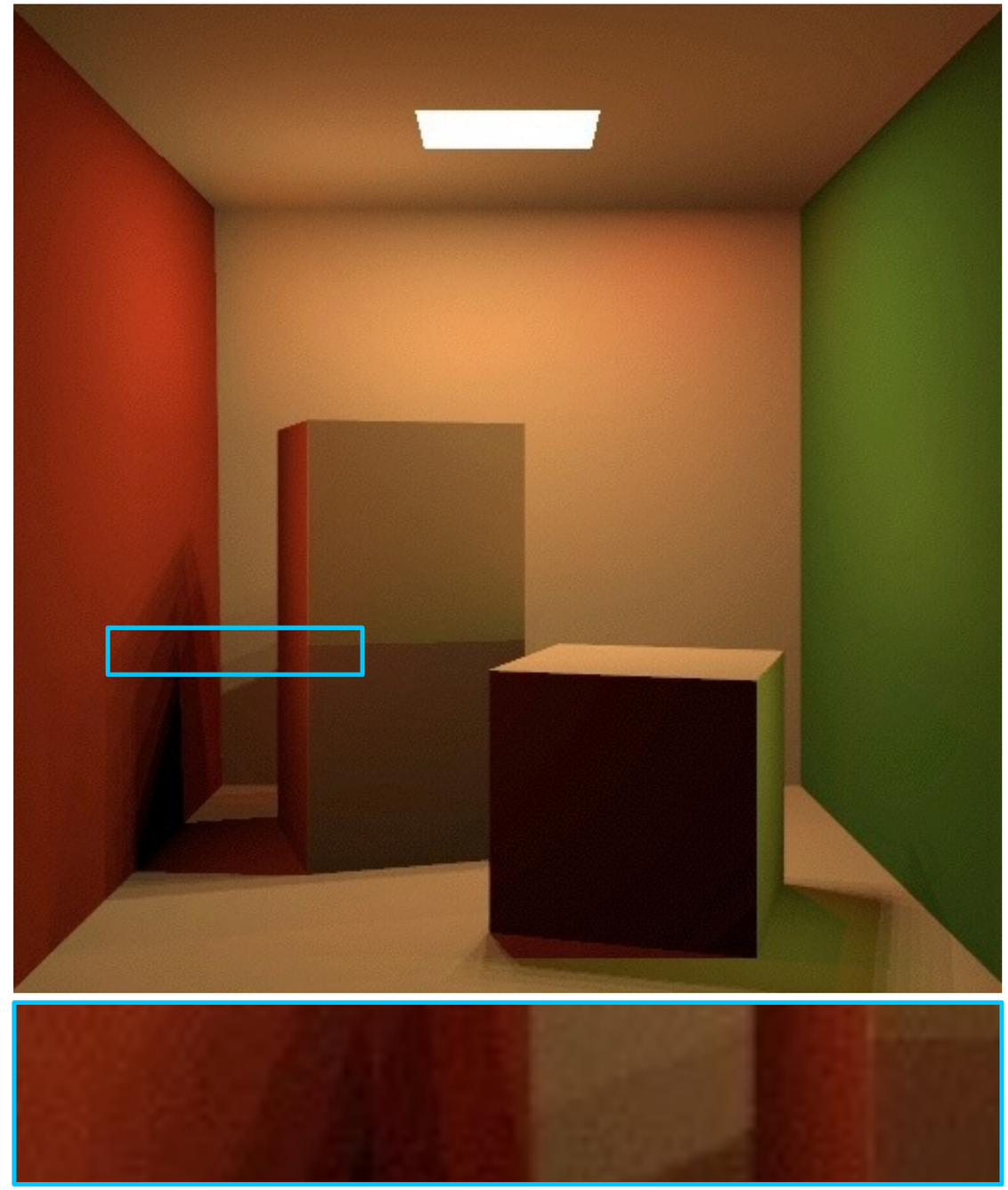


VPL = Virtual Point Light

Path Tracing vs. Many-Lights



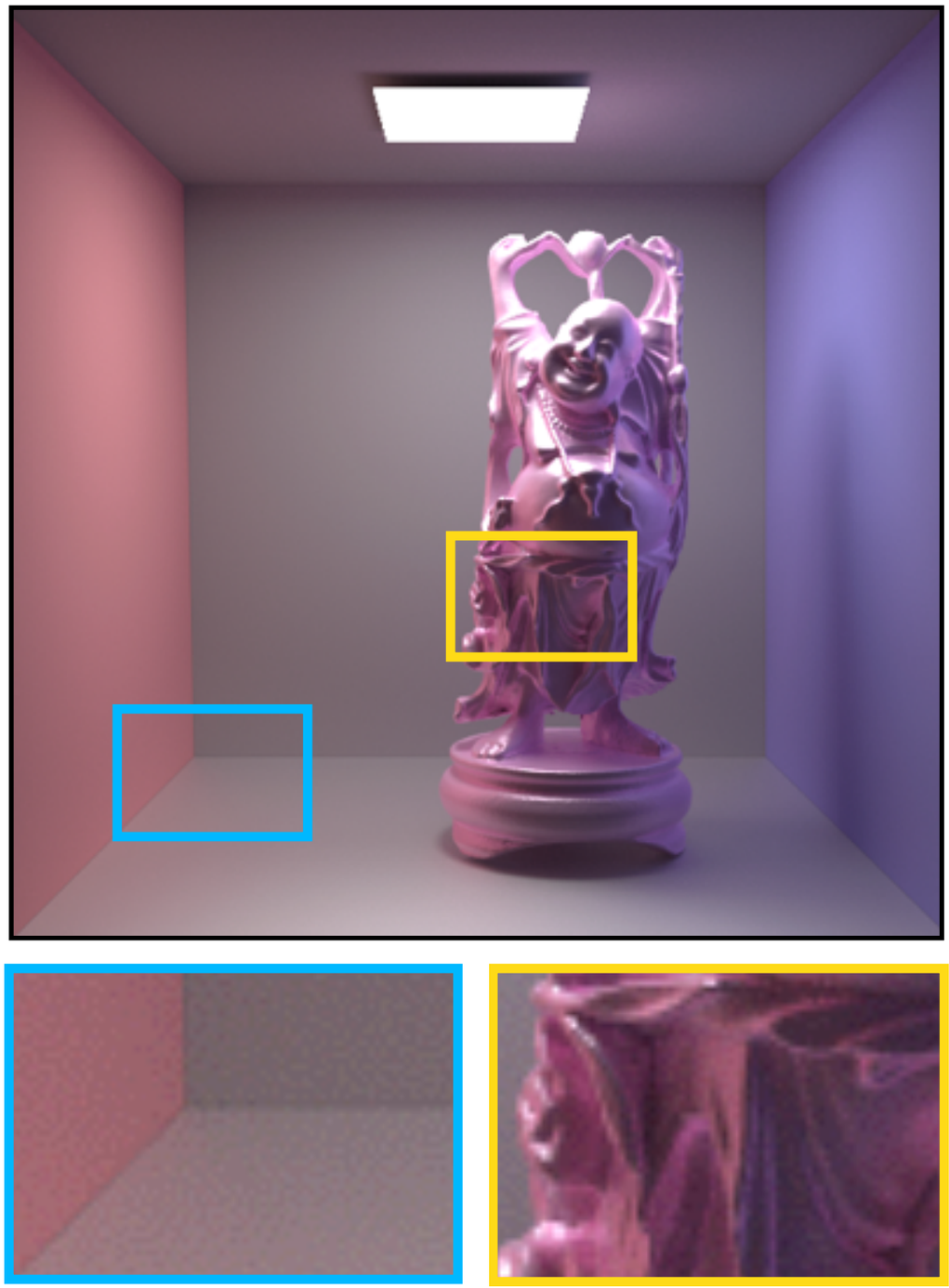
4 paths/pixel
(*random* error shows up as noise)



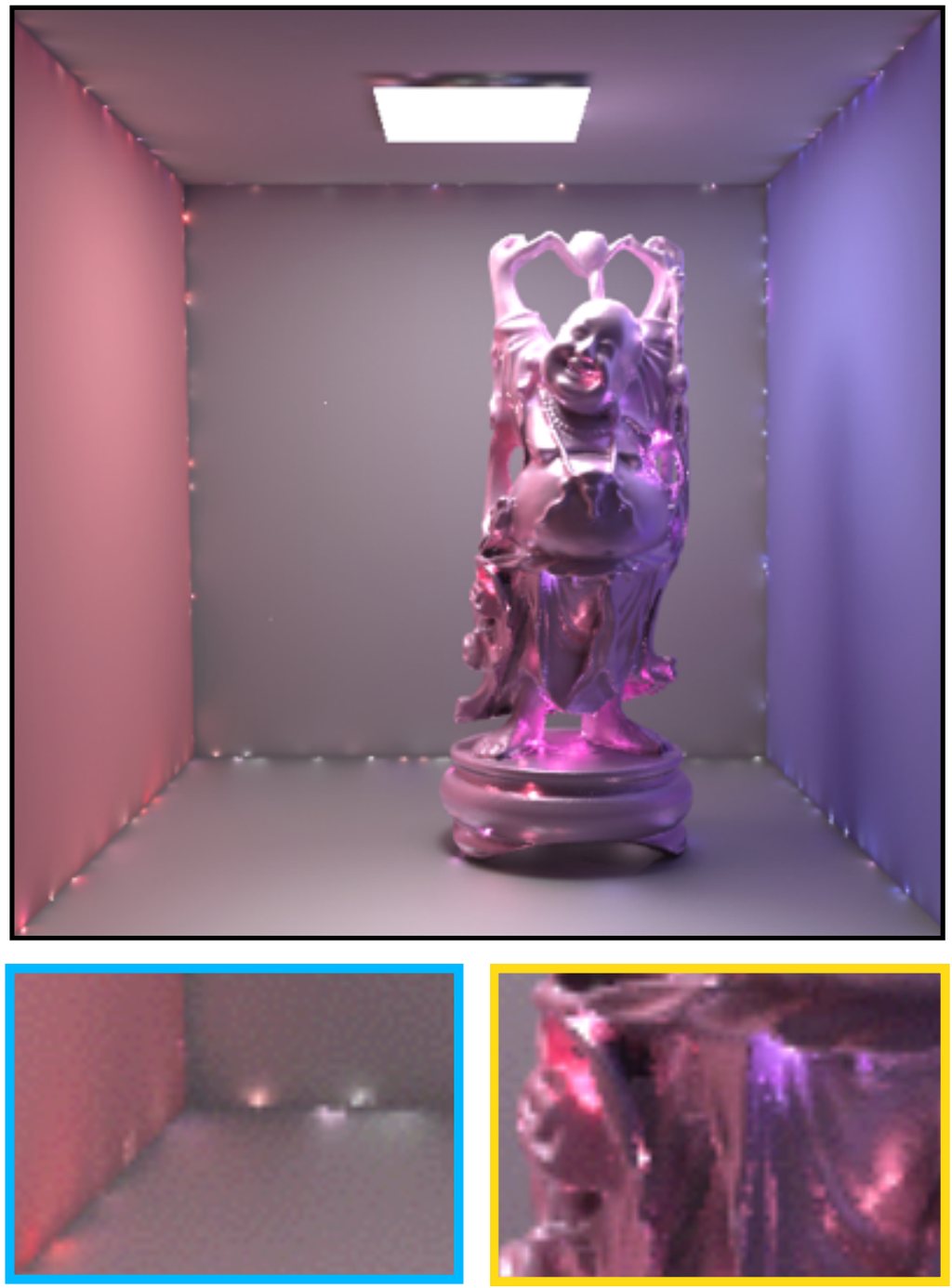
16 VPLs
(*correlated* error shows up as structured artifacts)

Lighting with VPLs

Reference

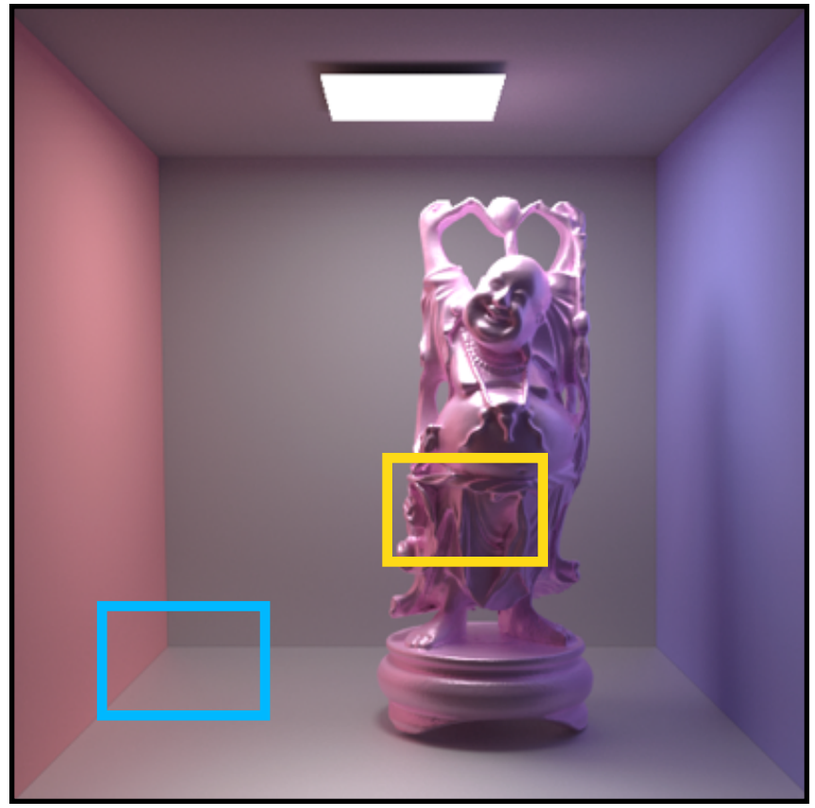


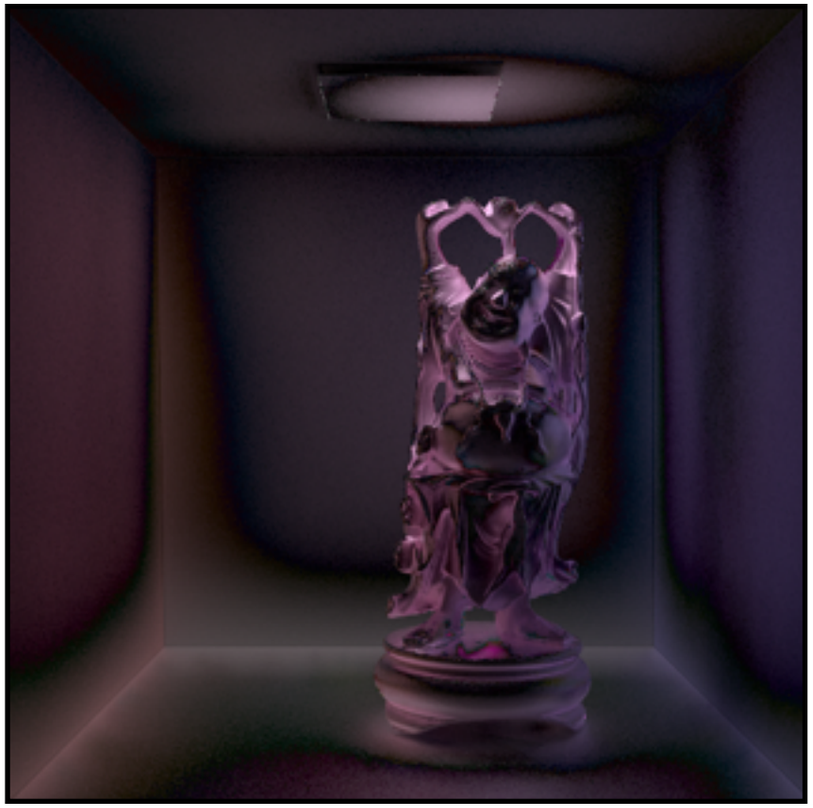
Approximation with VPLs

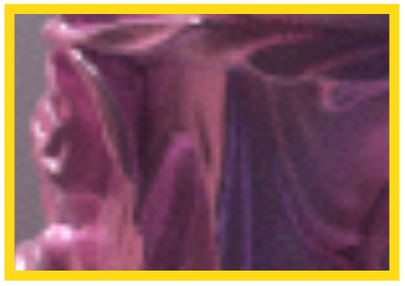


Bounding the G-term

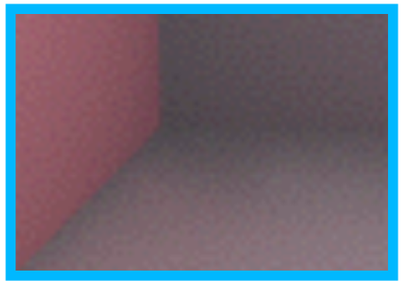
Reference ← Difference → VPLs with bounded G

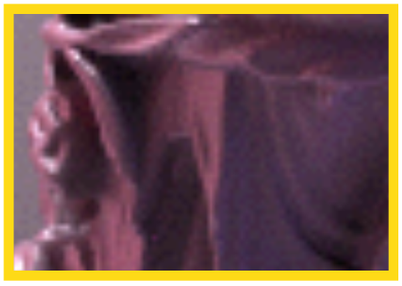




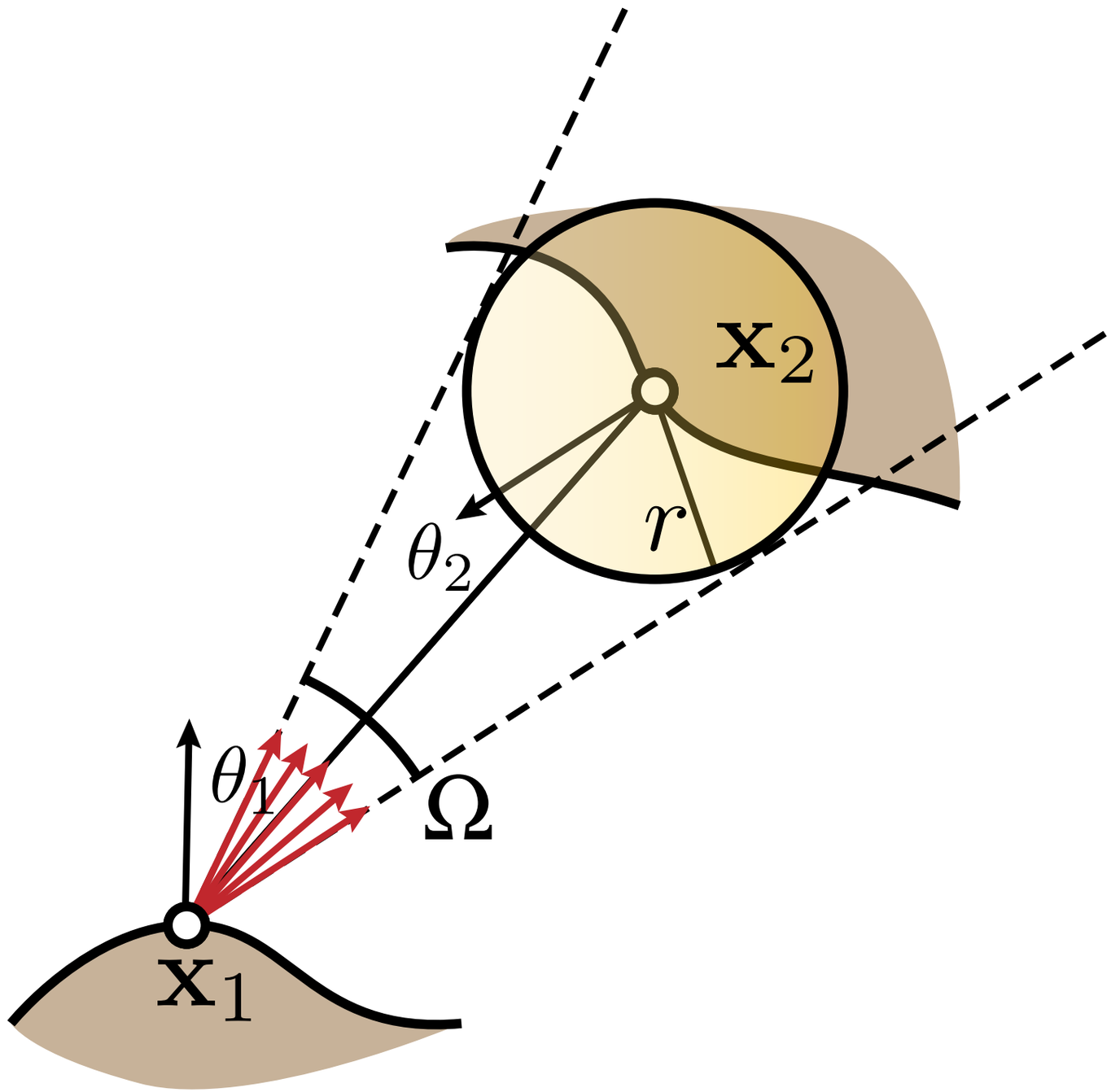


$$G(\mathbf{x},\mathbf{y}) - G_b(\mathbf{x},\mathbf{y})$$



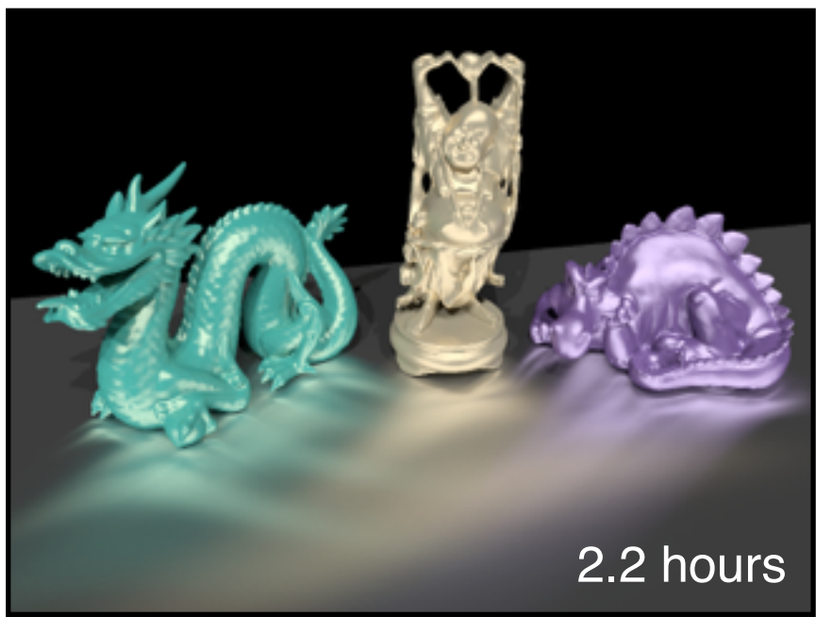


Virtual Spherical Lights

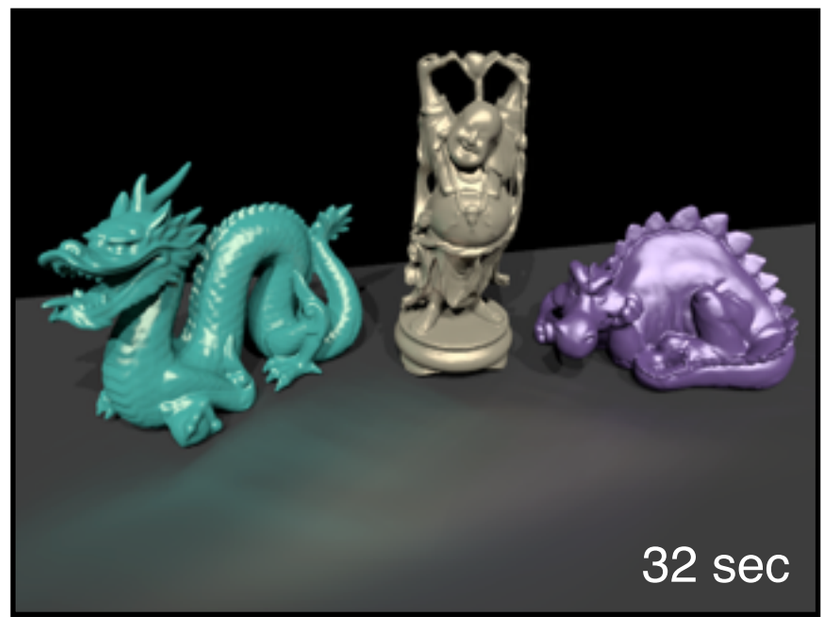


Virtual Spherical Lights

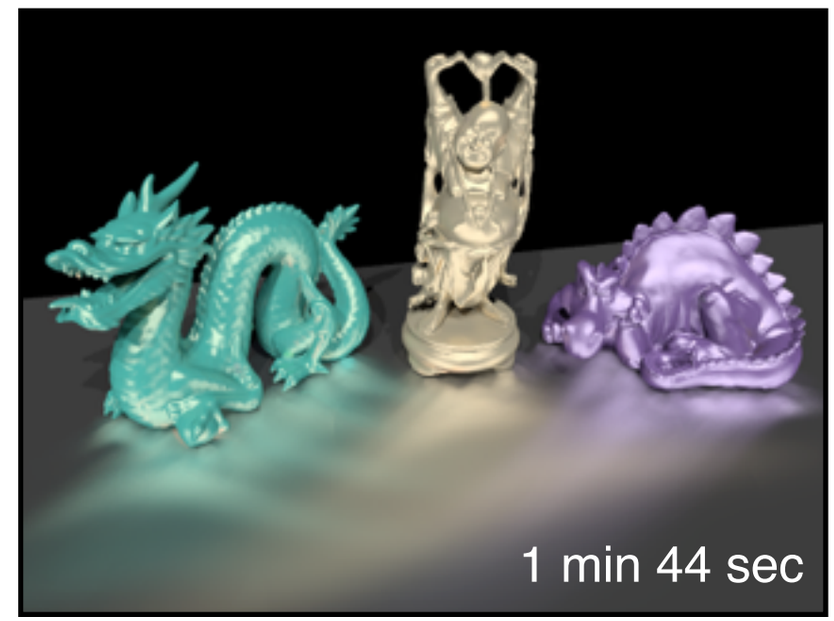
Reference



Bounded



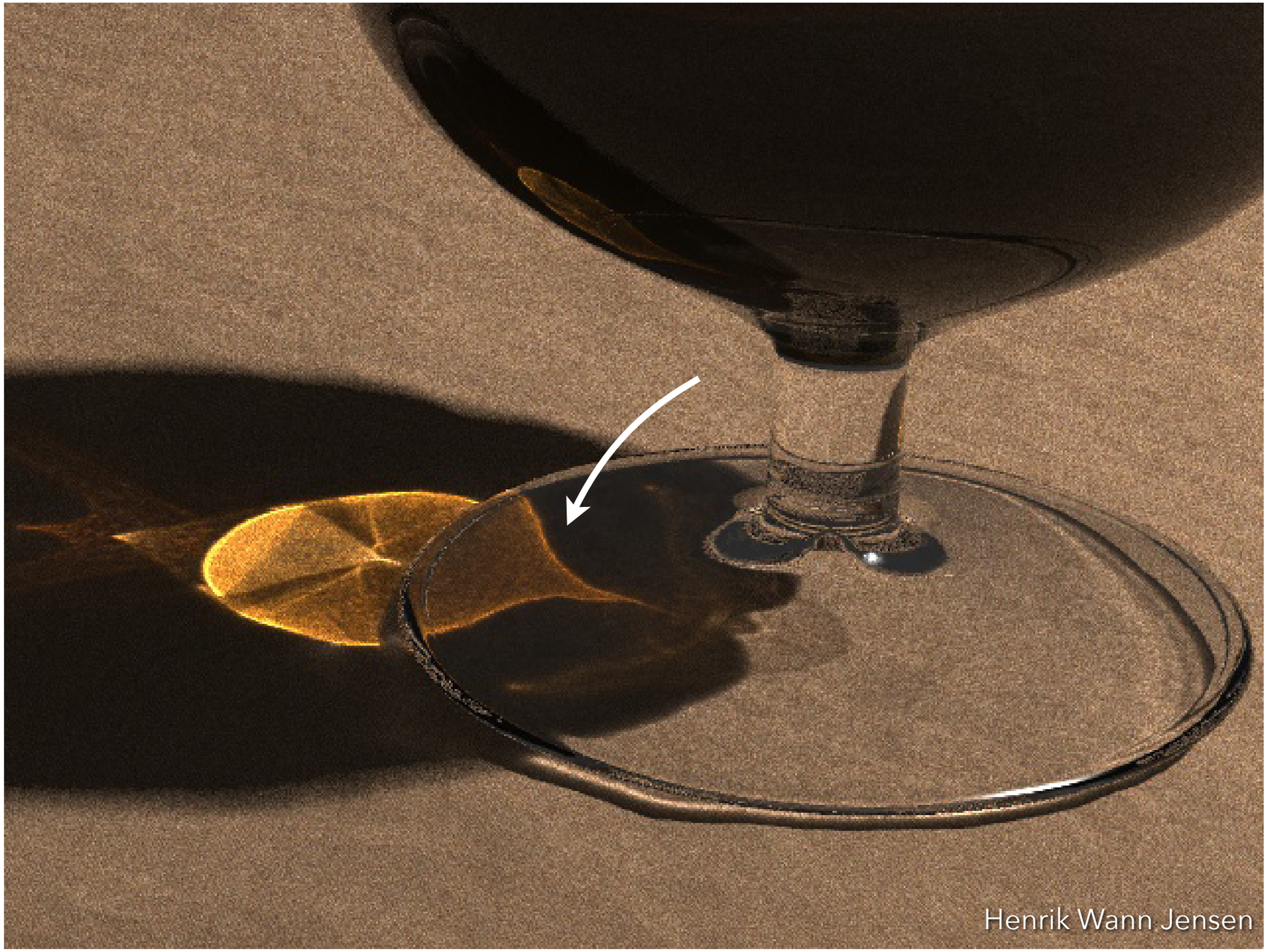
VSLs



Today's Menu

- Difficult light paths
- "Backward" Ray Tracing
- Photon Mapping
- Progressive Photon Mapping
- Photon Relaxation
- Relation to the path integral formulation

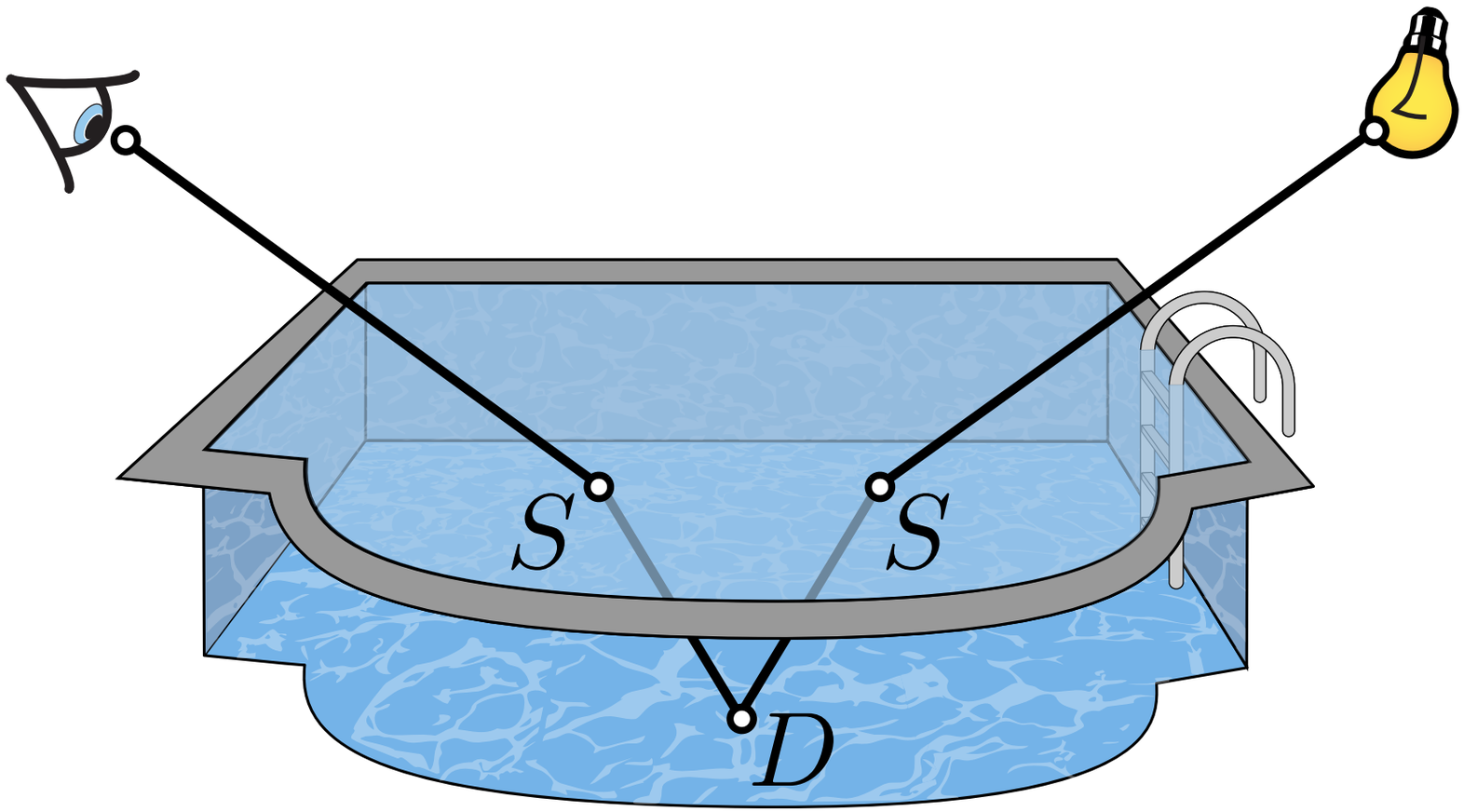
Specular-Diffuse-Specular Paths



Specular-Diffuse-Specular Paths

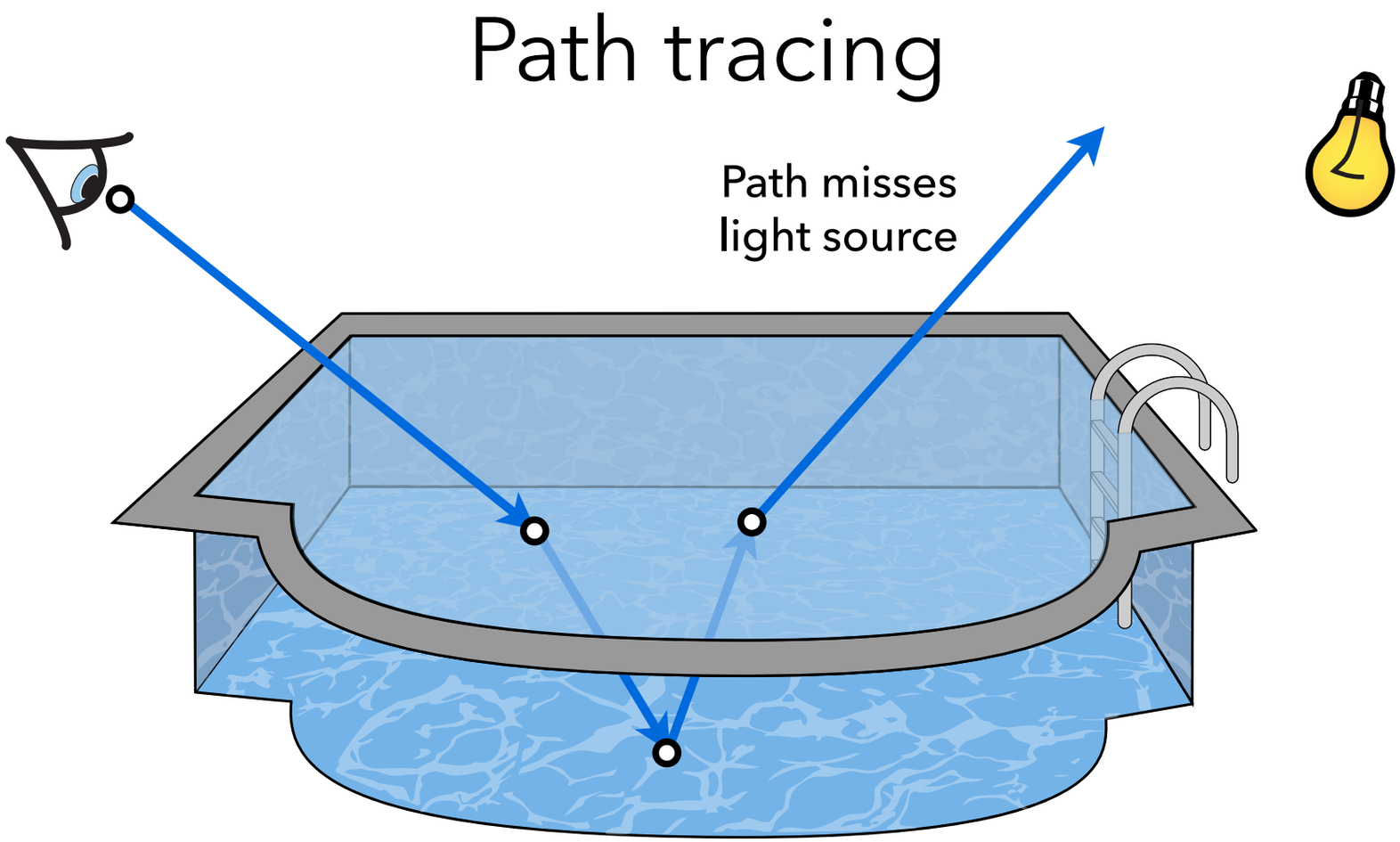
Reference | Bidirectional PT

Specular-Diffuse-Specular Paths



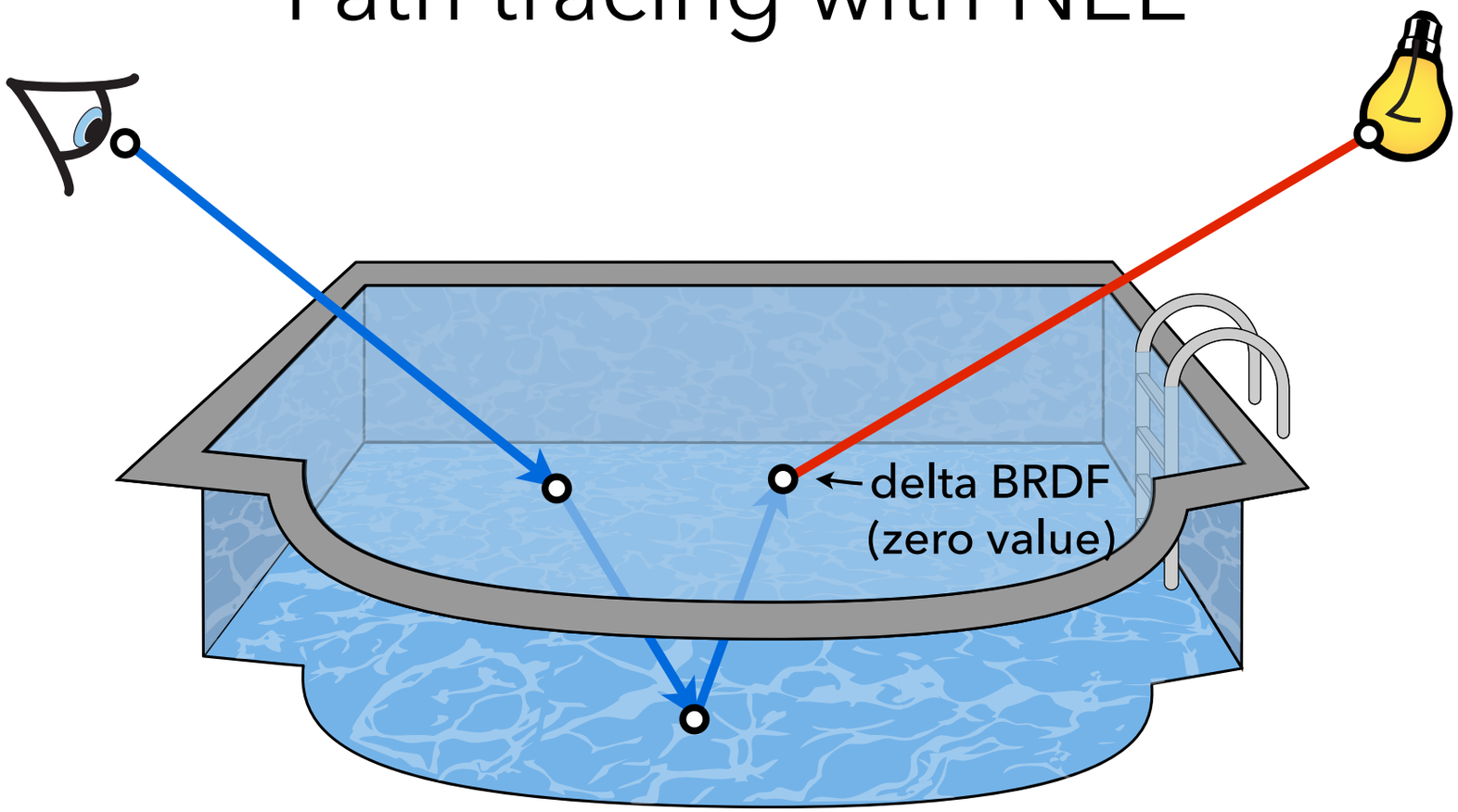
$LSDSE$ paths are difficult for unbiased techniques

Specular-Diffuse-Specular Paths



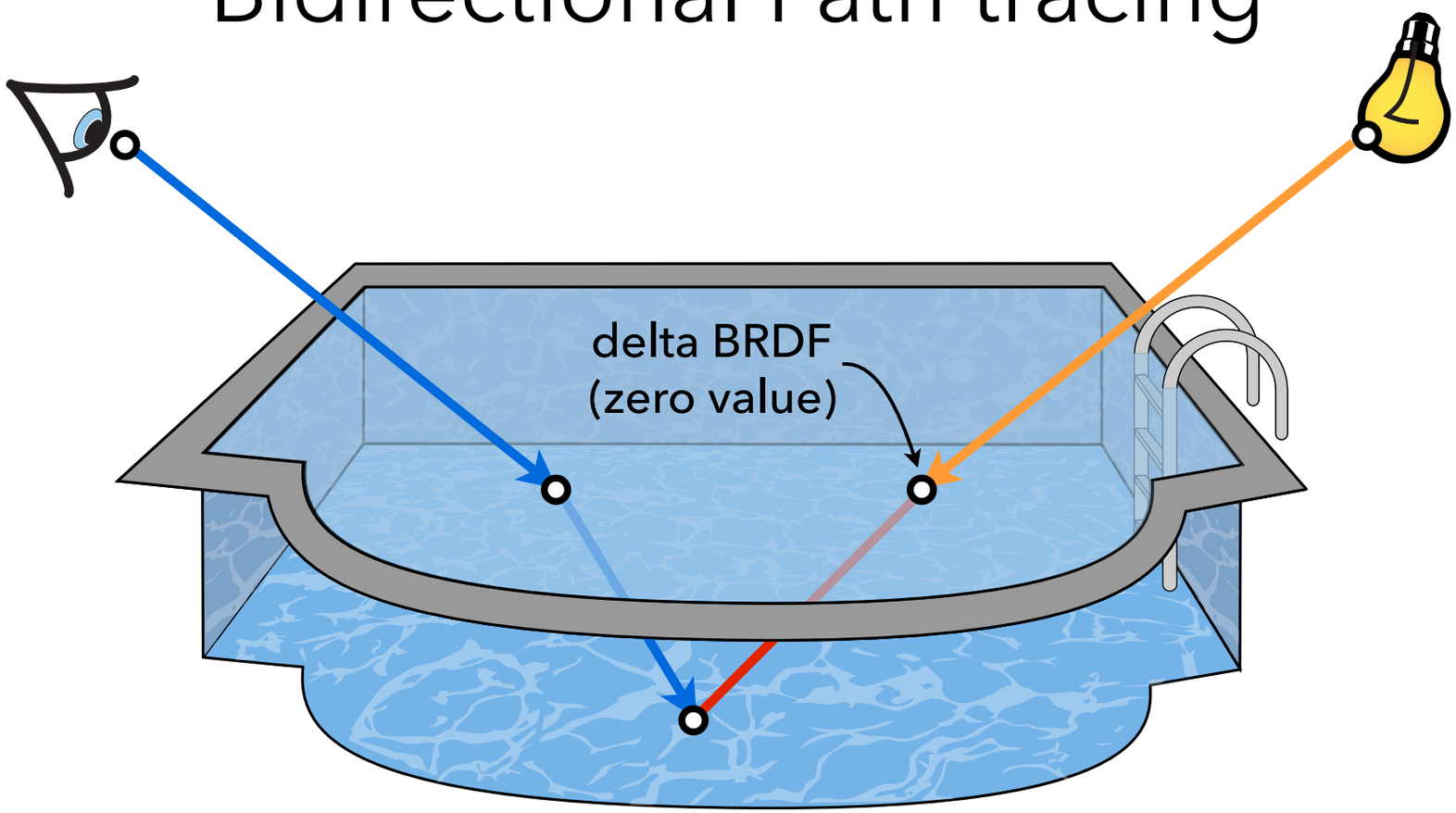
Specular-Diffuse-Specular Paths

Path tracing with NEE



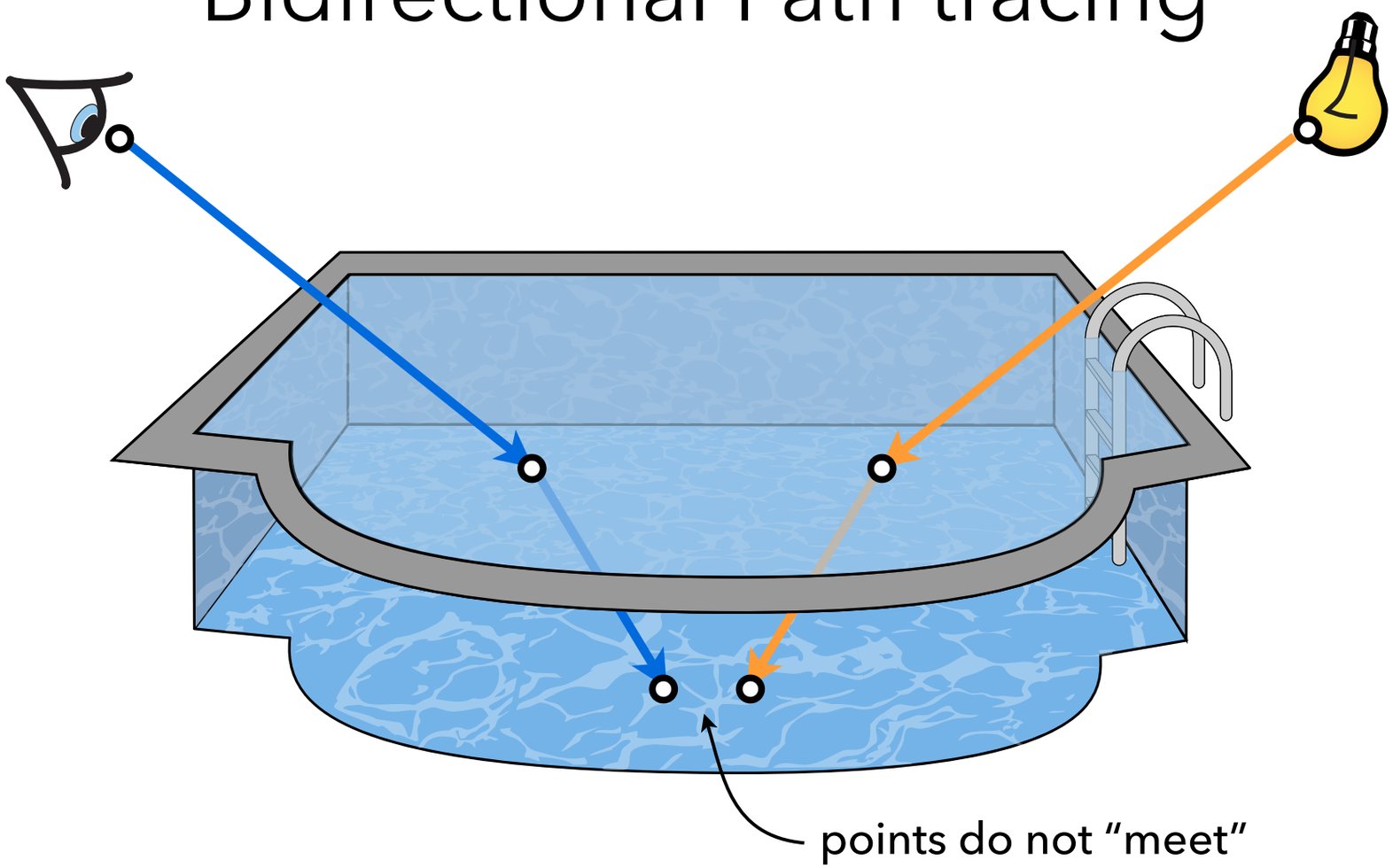
Specular-Diffuse-Specular Paths

Bidirectional Path tracing



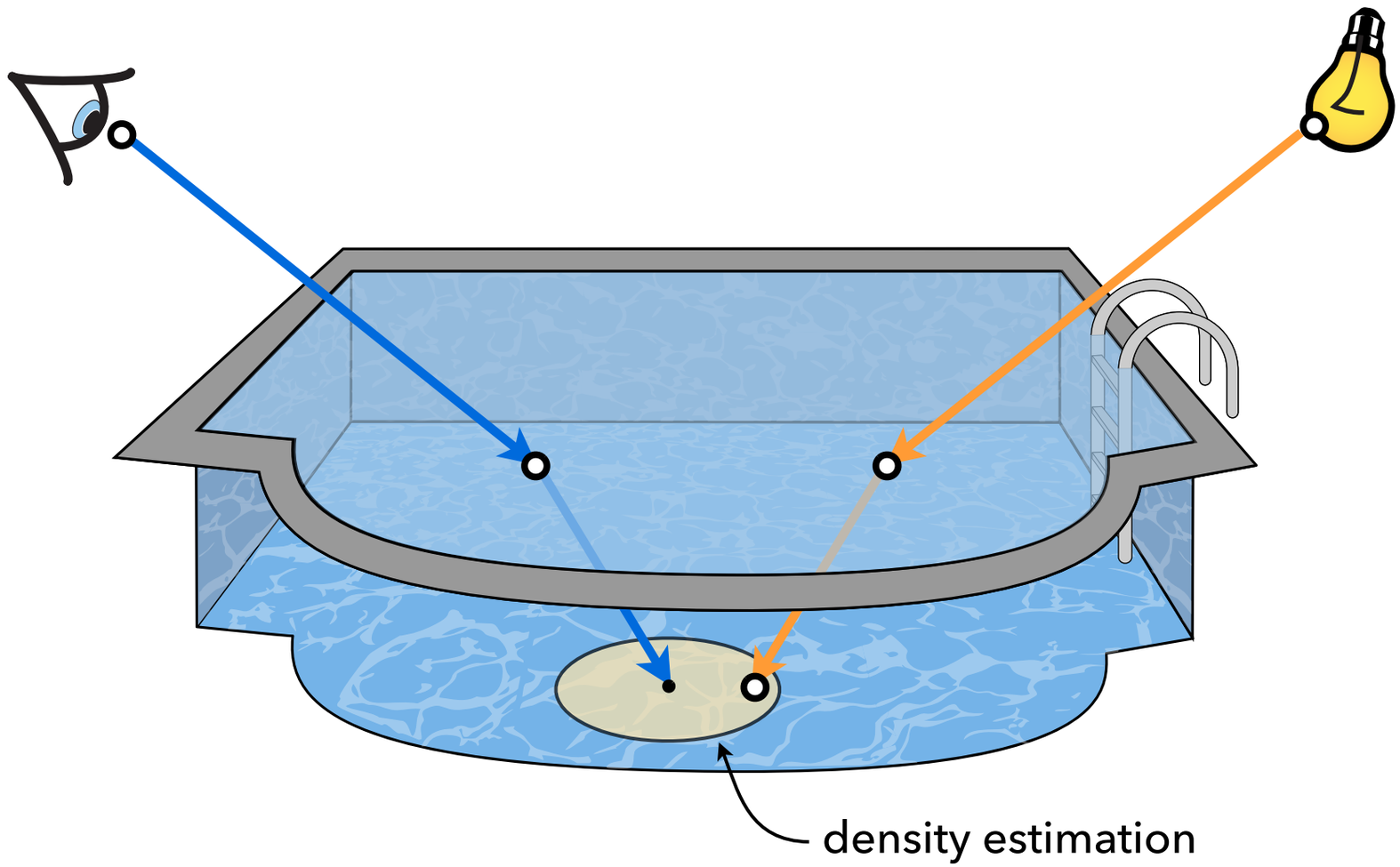
Specular-Diffuse-Specular Paths

Bidirectional Path tracing



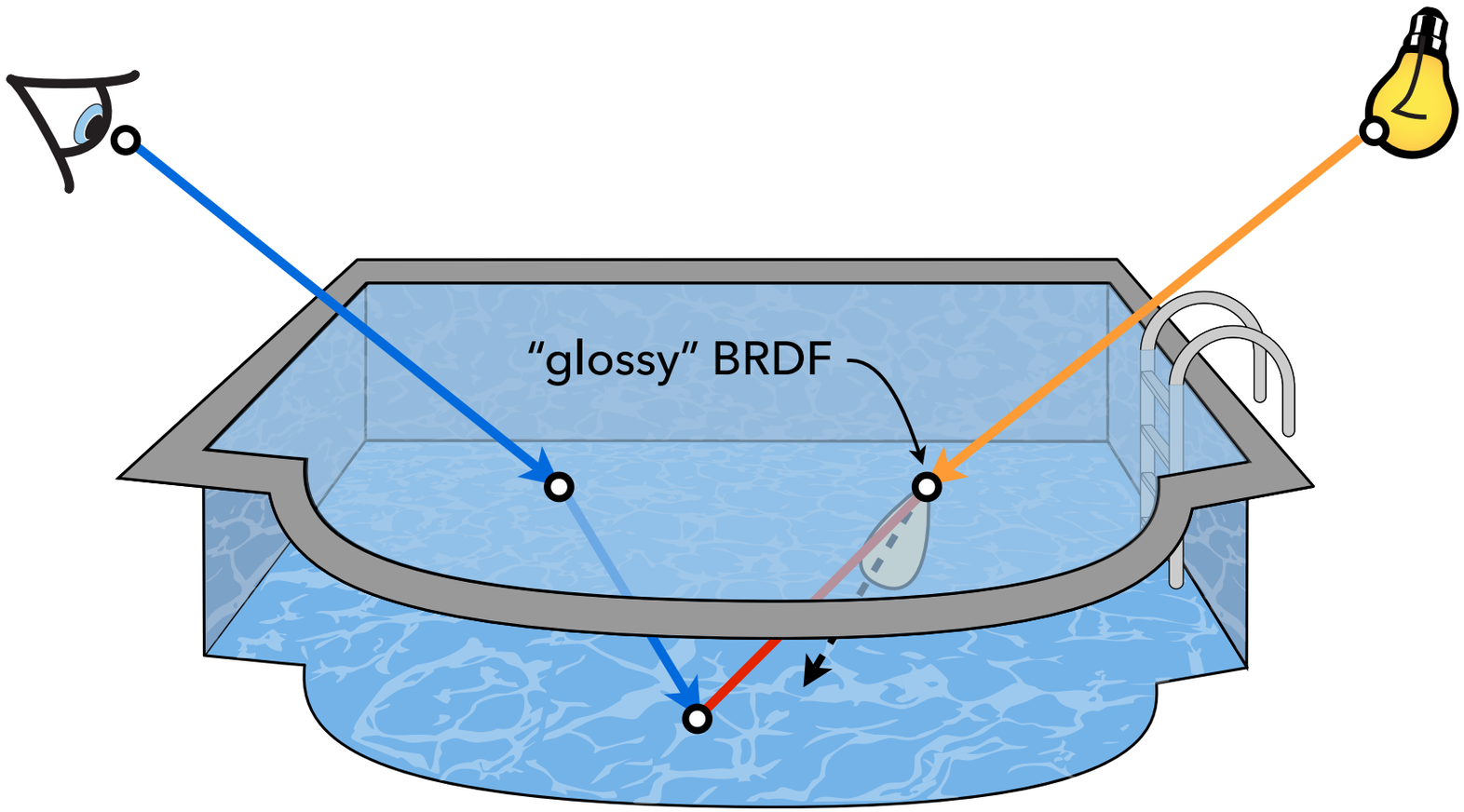
What now?

Specular-Diffuse-Specular Paths



Regularize delta functions (path points, specular BRDFs):
e.g. by employing kernel density estimation (blurring in space)

Specular-Diffuse-Specular Paths



Regularize delta functions (path points, specular BRDFs):
e.g. by regularizing a BRDF (blurring in angle)

“Backward” Ray Tracing

(predecessor of photon mapping)

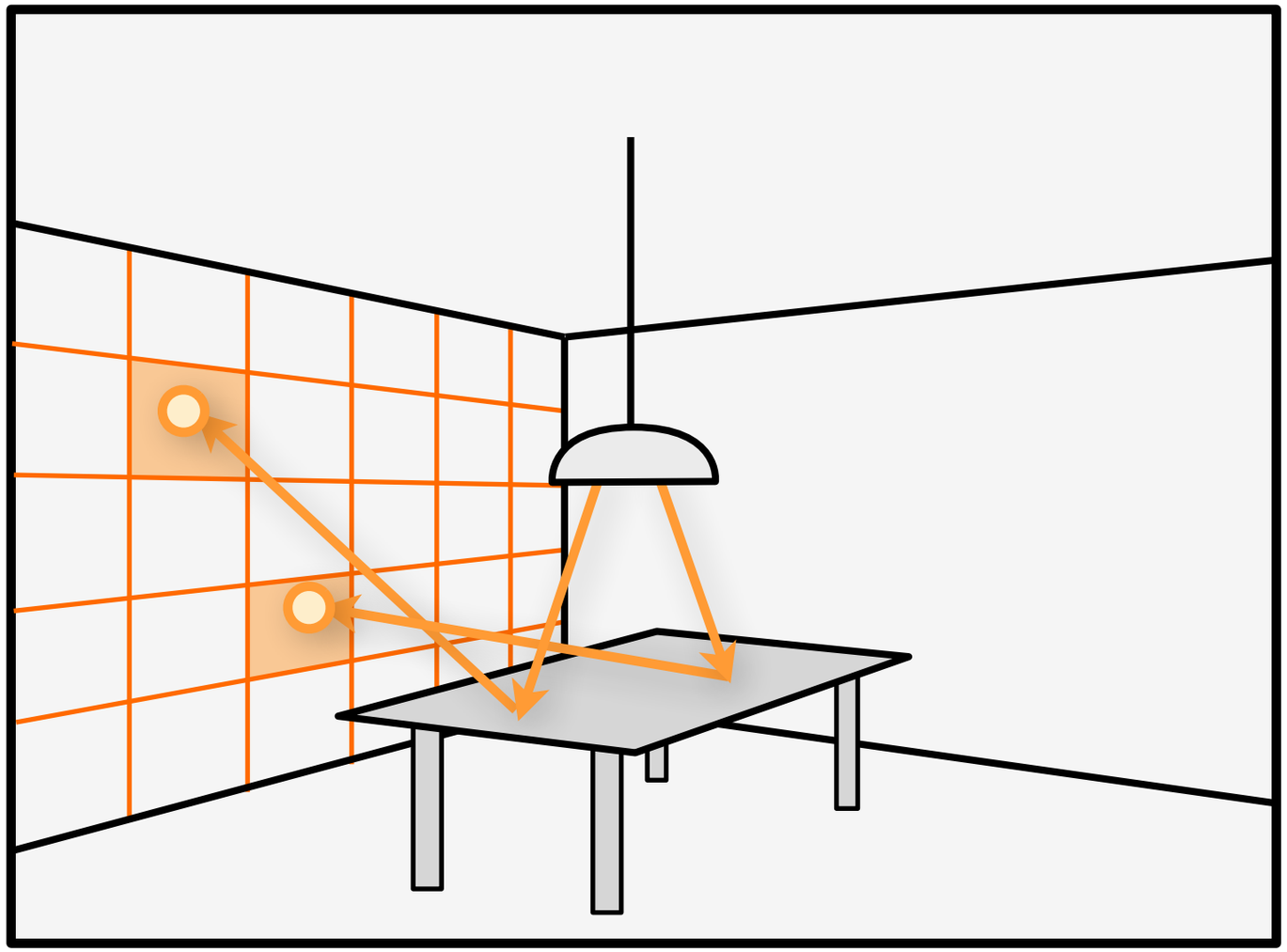
"Backward" Ray Tracing

- James Arvo. In *Developments in Ray Tracing*, SIGGRAPH '**86** Course Notes
- Emit photons from light sources and store them in *illumination maps*
- *Illumination map* = texture for accumulating irradiance

Note on the name of the technique: In retrospect, Arvo regretted using the term "backward" to refer to tracing light paths since many later publications use it in the opposite sense, i.e. tracing eye paths. To avoid confusion, he recommends terms such as *light tracing* and *eye tracing* as they are unambiguous.

"Backward" Ray Tracing

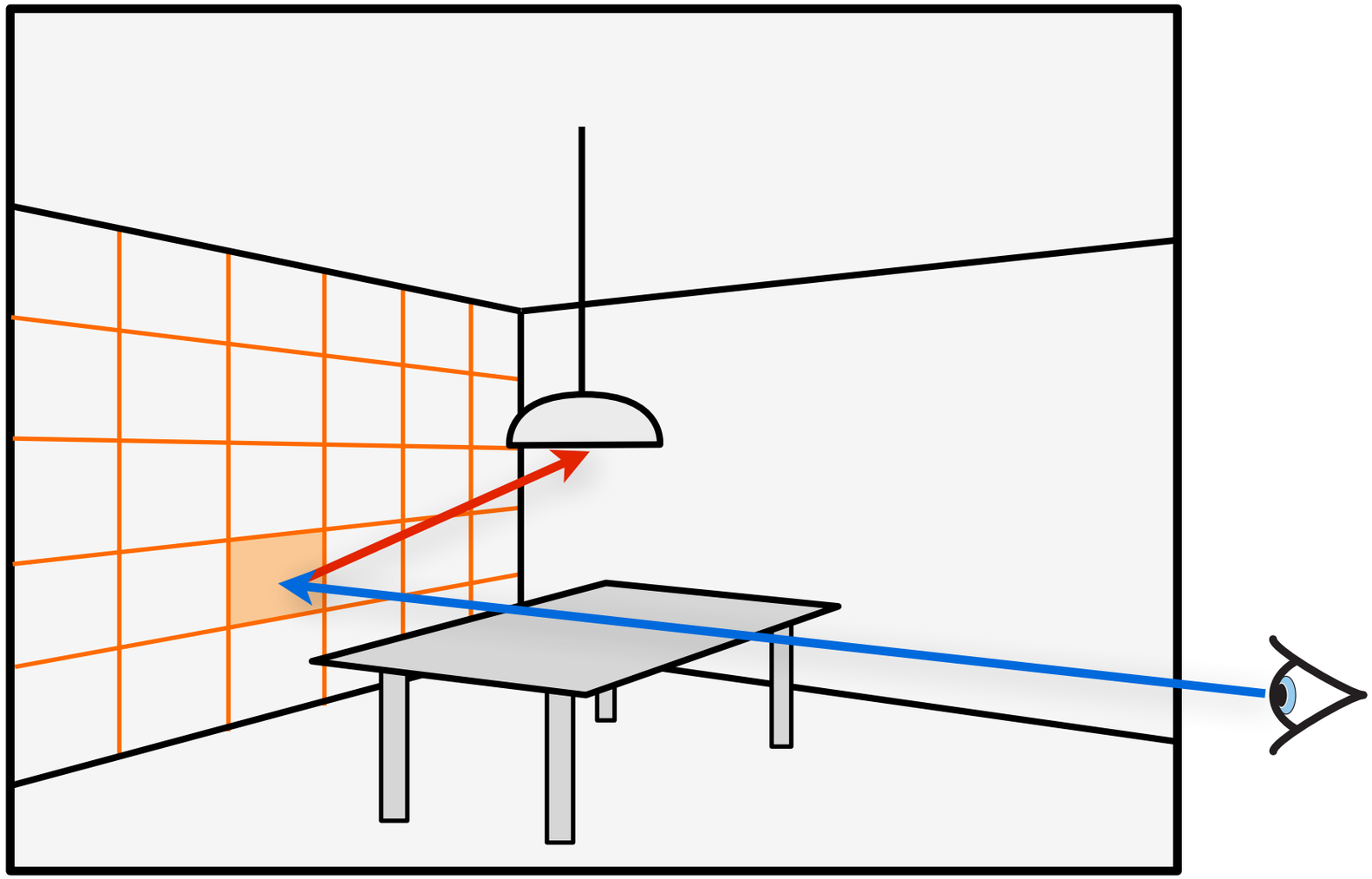
- Preprocess:
  - shoot light from light sources
  - deposit photon energy in illumination maps



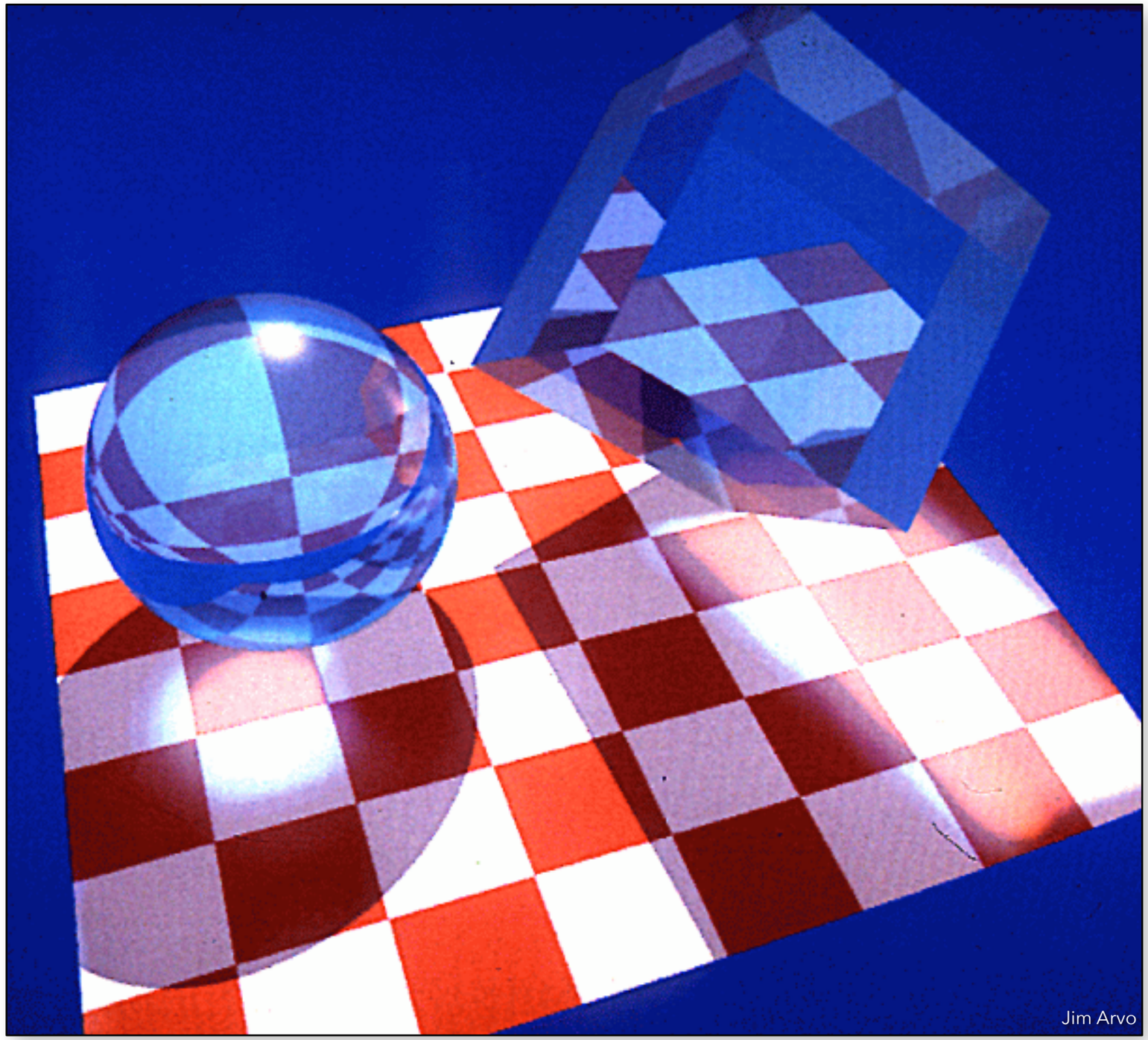
Irradiance: "number of photons hitting a small patch of a wall per second, divided by size of patch"

"Backward" Ray Tracing

- For each shading point
  - compute direct lighting
  - lookup indirect lighting from illumination maps



"Backward" Ray Tracing



"Backward Ray Tracing", by James Arvo. In *Developments in Ray Tracing*, SIGGRAPH `86 Course Notes, Volume 12.

"Backward" Ray Tracing

✓ One of the first techniques to simulate caustics!

✘ Requires parametrizing surfaces or meshing

- Difficult to handle complex or procedural geometry

✘ Hard to choose illumination map resolution

- high resolution with few photons: high-frequency *noise*
- low resolution with many photons: *blurred* illumination

Visual Break

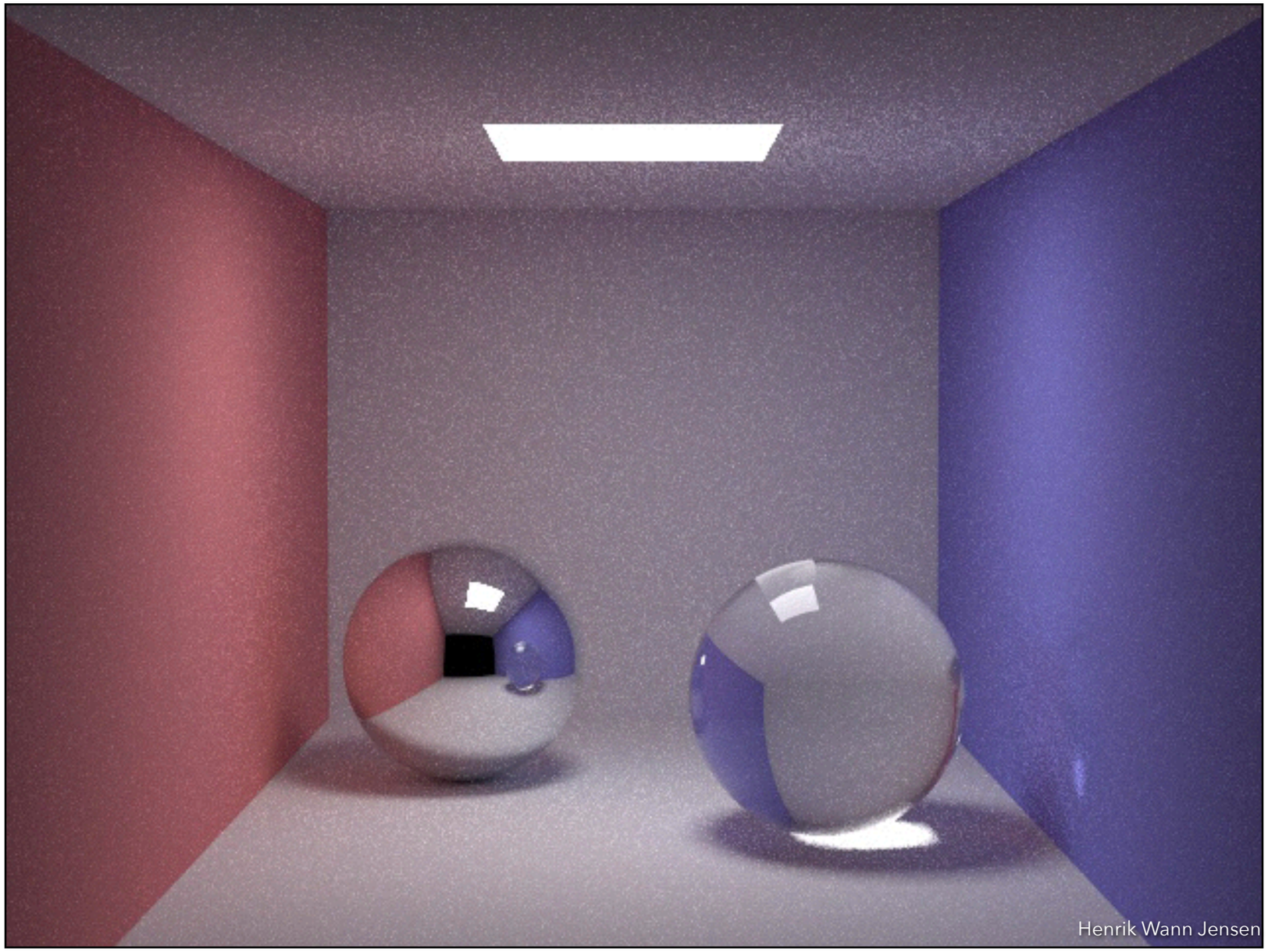
Image courtesy of L. Eno (rendered with VRay)

Visual Break

 Image courtesy of ARC media (rendered with VRay)

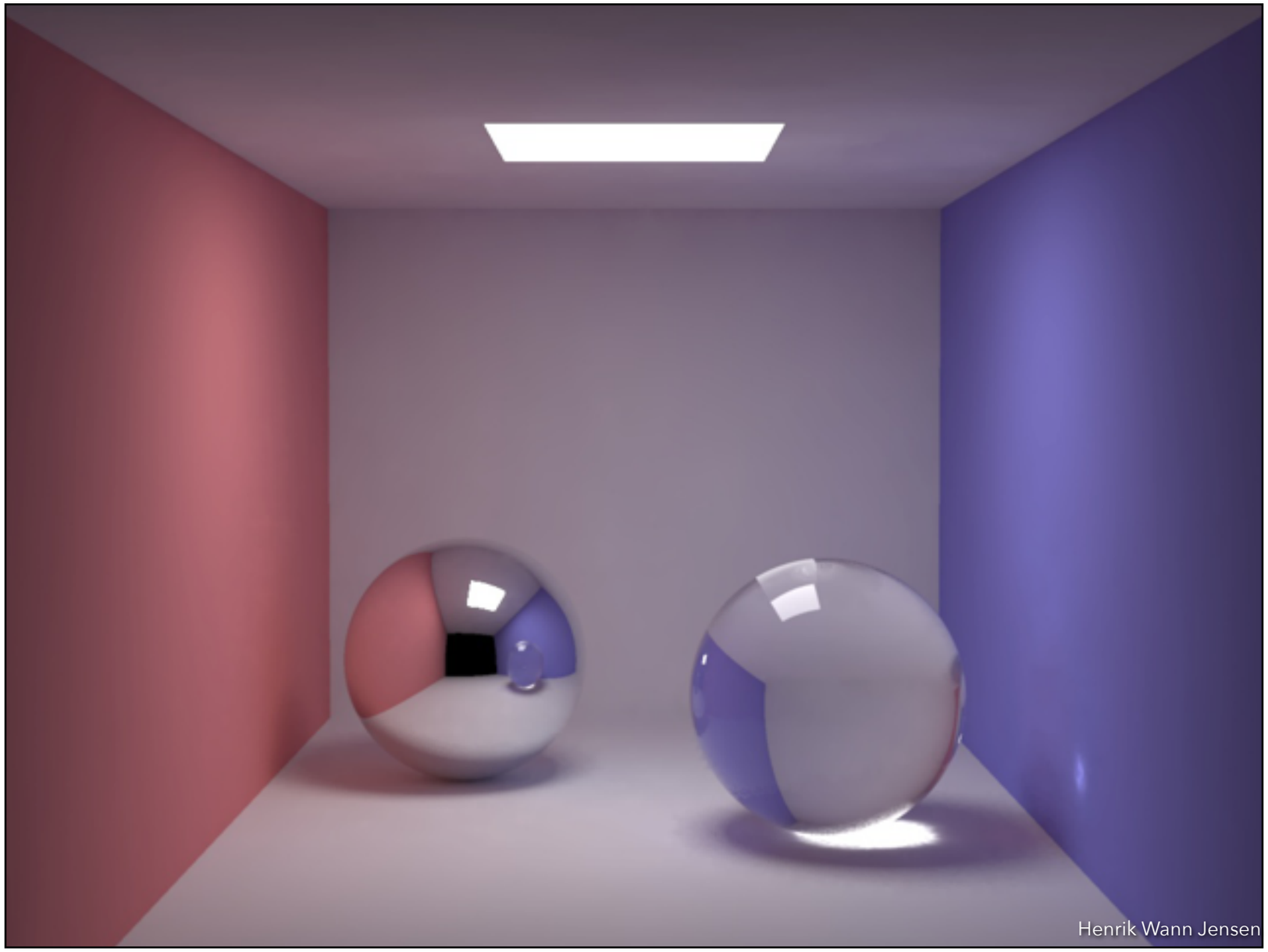
Photon Mapping

Path Tracing



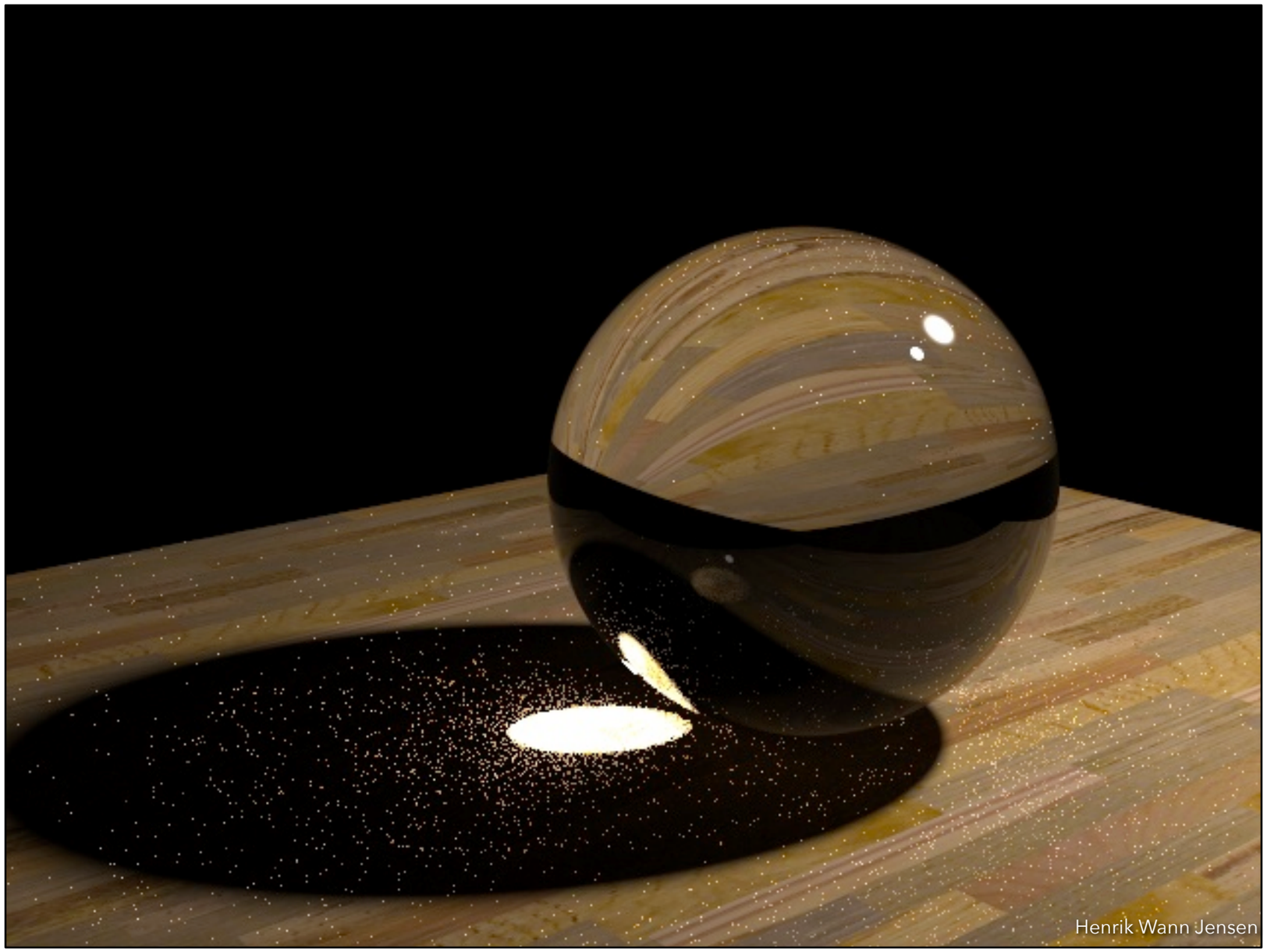
100 paths/pixel (5 minutes)

Photon Mapping



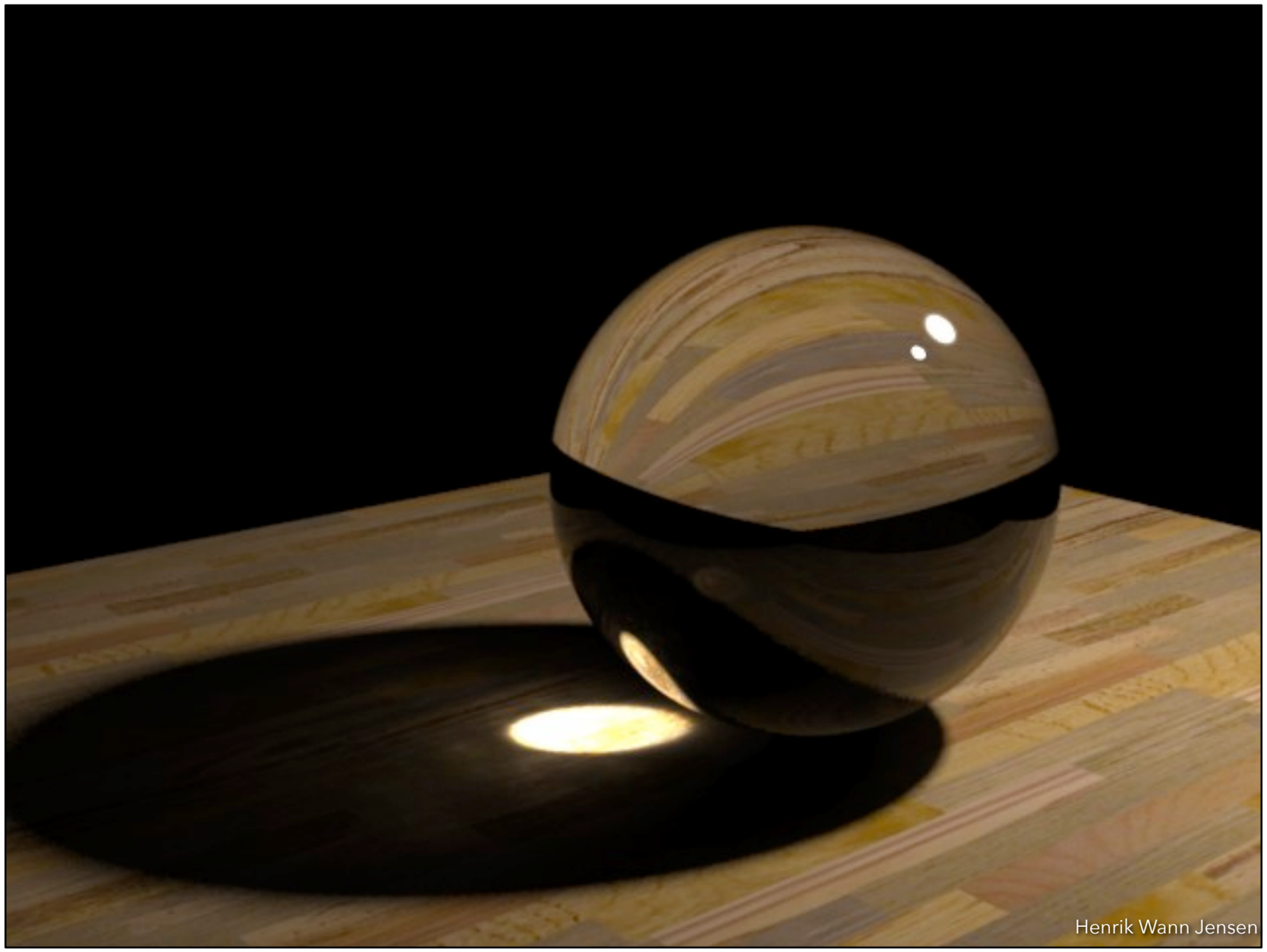
10 rays/pixel (5 seconds)

Path Tracing



1000 paths/pixel

Photon Mapping



Photon Mapping

- A two-pass algorithm:
  - Pass 1: Tracing of photons from light sources, and caching them in a photon map
  - Pass 2: Tracing from the eye and approximating indirect illumination using the photons
- Similar to "backward" ray tracing, but different way of storing photons & computing density

Photon Mapping

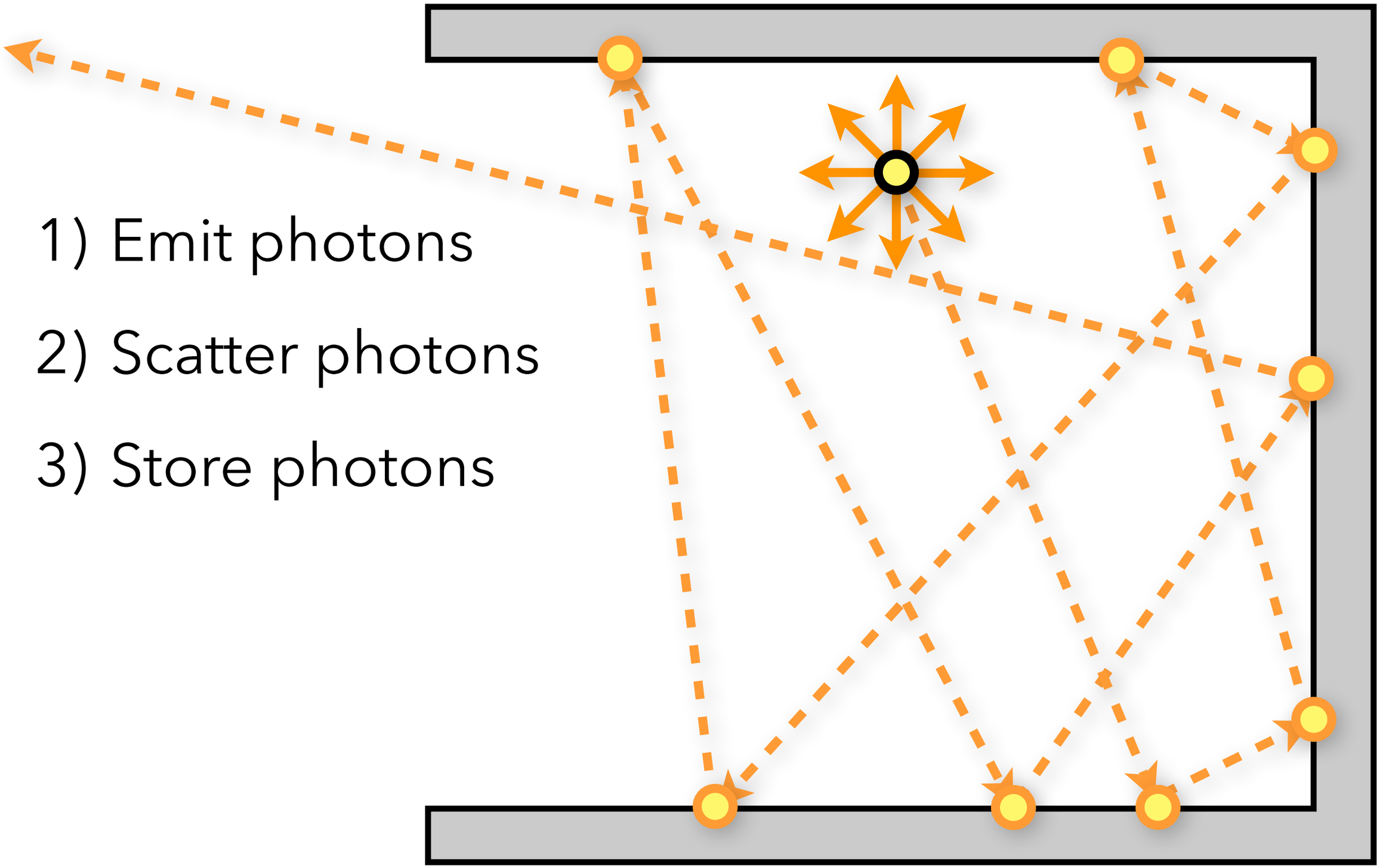
- A two-pass algorithm:
  - Pass 1: Tracing of photons from light sources, and caching them in a photon map
  - Pass 2: Tracing from the eye and approximating indirect illumination using the photons
- Similar to "backward" ray tracing, but different way of storing photons & computing density

Photon Tracing

1) Emit photons

2) Scatter photons

3) Store photons

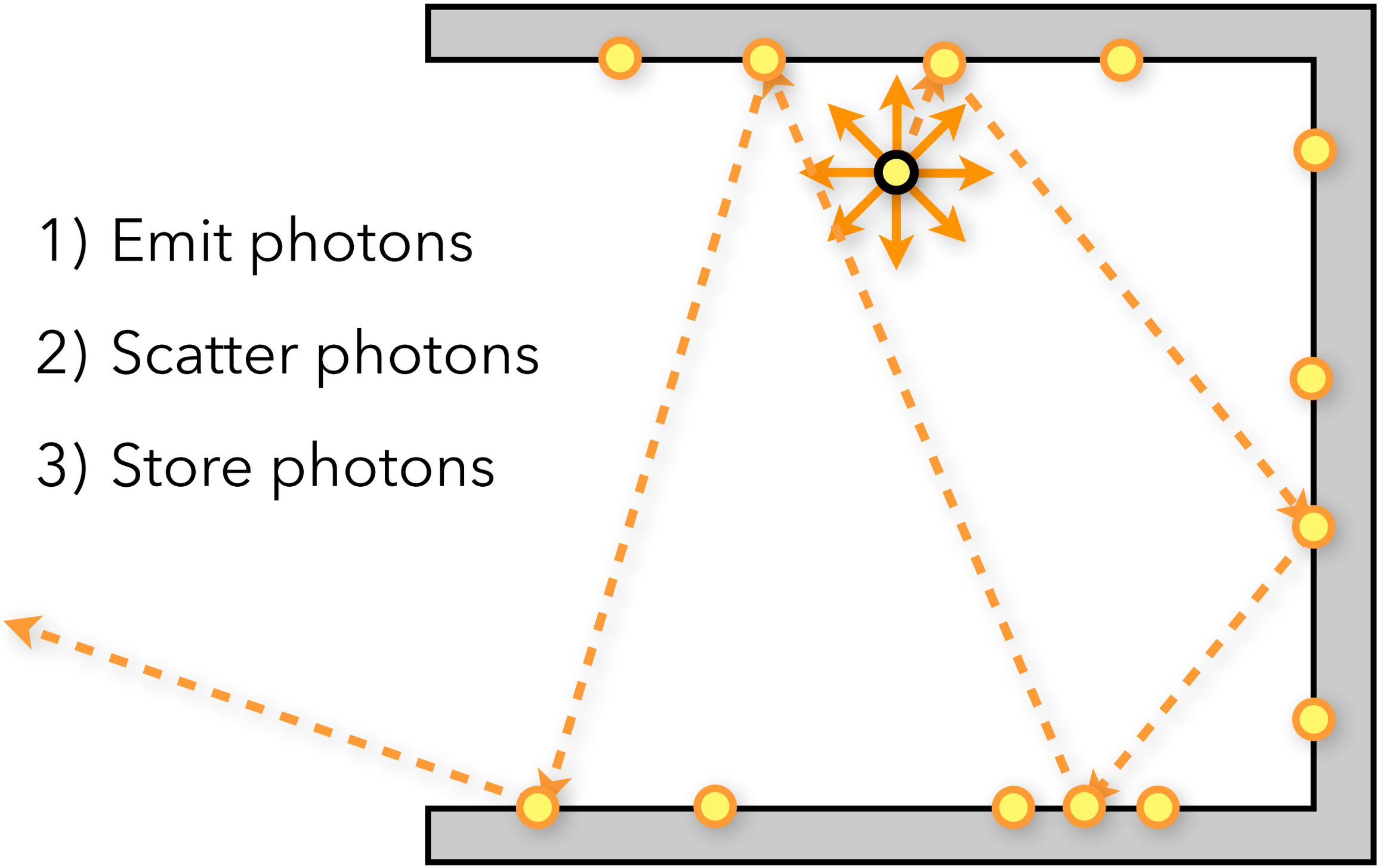


Photon Tracing

1) Emit photons

2) Scatter photons

3) Store photons

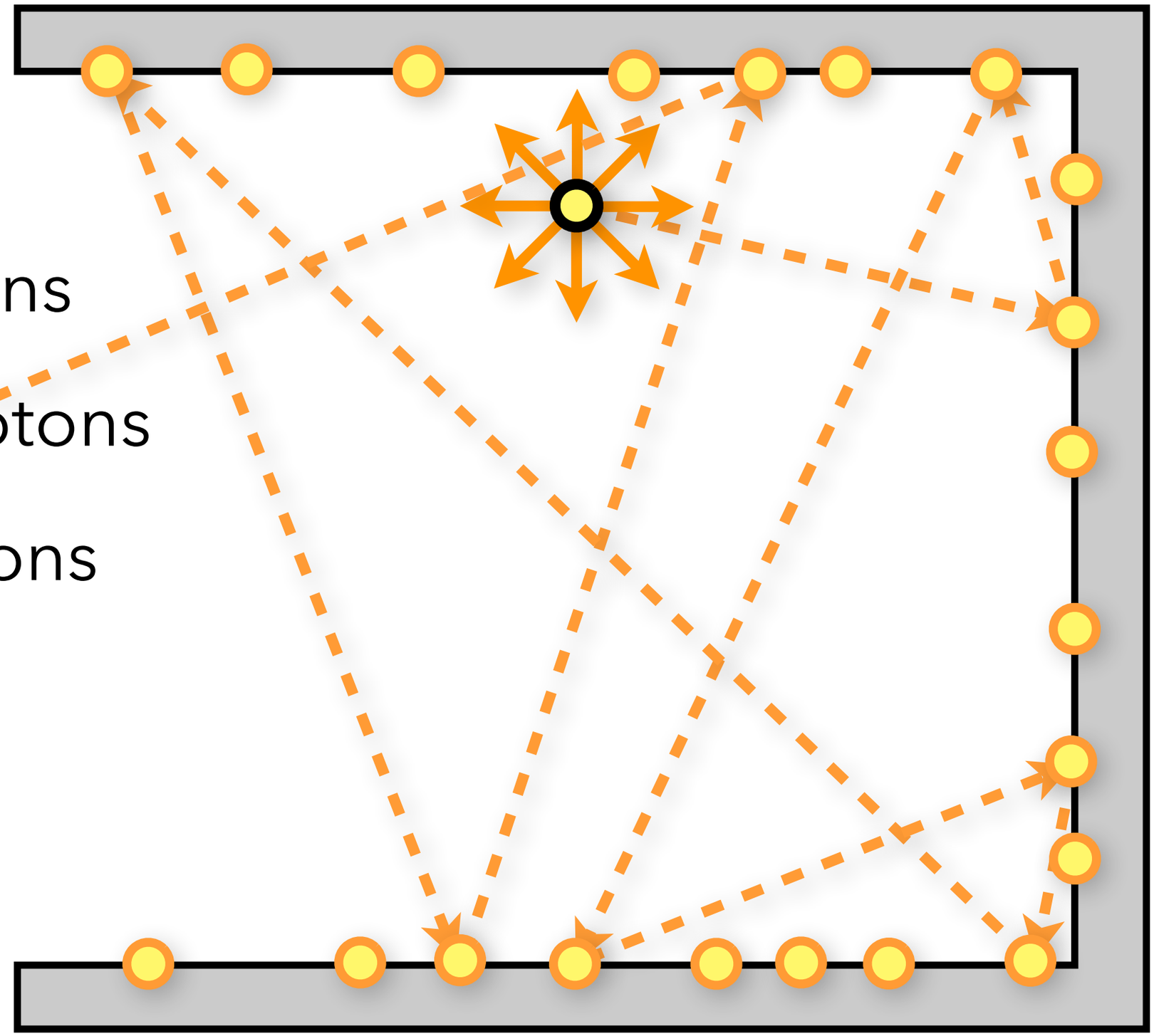


Photon Tracing

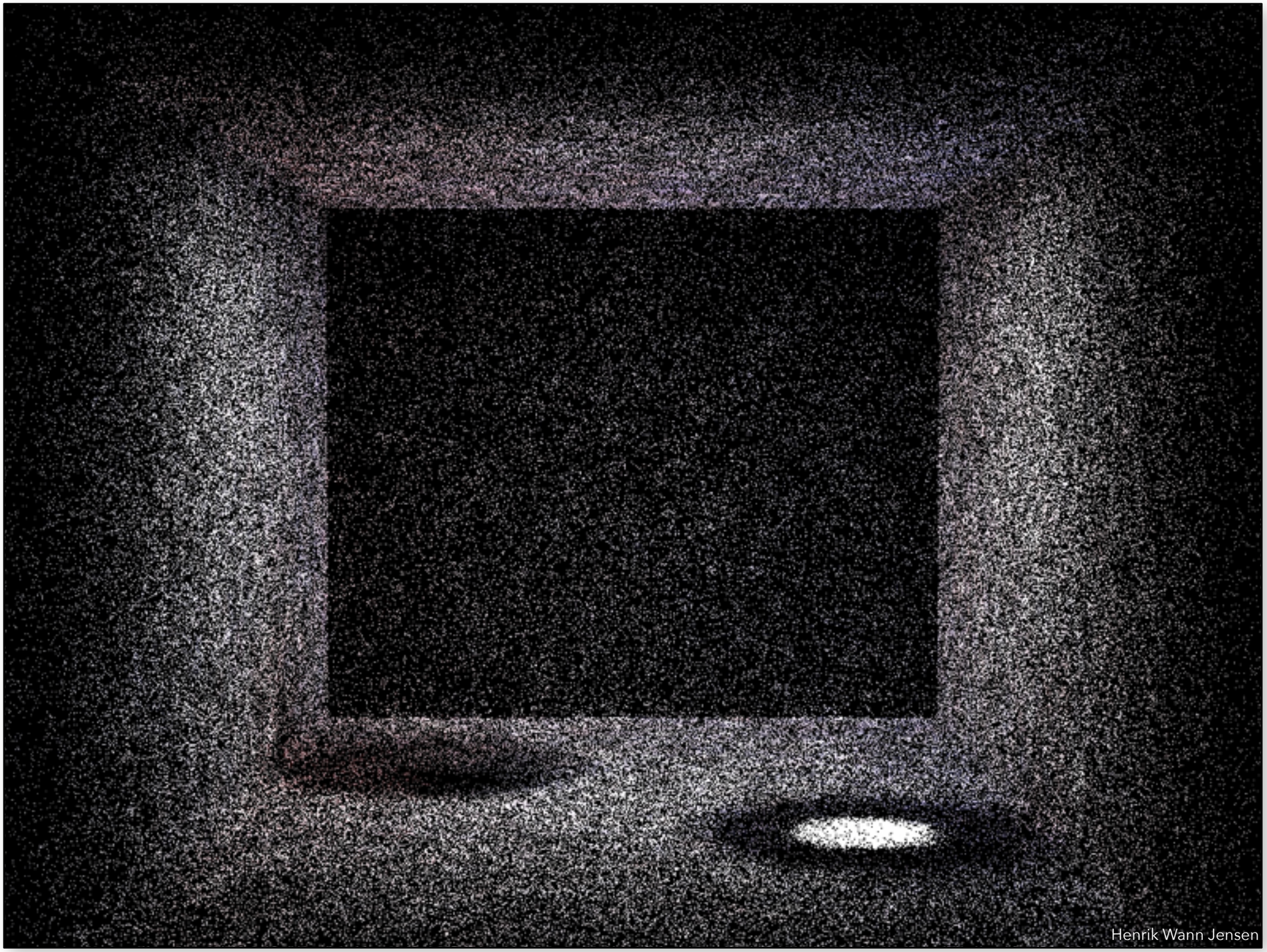
1) Emit photons

2) Scatter photons

3) Store photons



Visualization of the Photon Map



Photon Emission

- Define initial:
  - $\mathbf{x}_p$: position
  - $\vec{\omega}_p$: direction
  - Φ_p : photon power
- Recipe:
  - Sample position on surface area of light with $p(\mathbf{x}_p)$
  - Sample direction $p(\vec{\omega}_p|\mathbf{x}_p)$

$$\Phi_p = \frac{1}{M}\frac{L_e(\mathbf{x}_p, \vec{\omega}_p)}{p(\mathbf{x}_p)p(\vec{\omega}_p|\mathbf{x}_p)}$$

# of emitted photons

Photon Emission

- Interesting derivation:
  - if PDFs are proportional to the emission:

$$p(\mathbf{x}_p) = \frac{\int_{H^2} L_e(\mathbf{x}_p, \vec{\omega}) d\vec{\omega}}{\int_A \int_{H^2} L_e(\mathbf{x}, \vec{\omega}) d\vec{\omega} d\mathbf{x}} \qquad p(\vec{\omega}_p | \mathbf{x}_p) = \frac{L_e(\mathbf{x}_p, \vec{\omega}_p)}{\int_{H^2} L_e(\mathbf{x}_p, \vec{\omega}) d\vec{\omega}}$$

  - then:

$$\Phi_p = \frac{1}{M} \frac{L_e(\mathbf{x}_p, \vec{\omega}_p)}{p(\mathbf{x}_p) p(\vec{\omega}_p | \mathbf{x}_p)}$$

$$= \frac{1}{M} \frac{\cancel{L_e(\mathbf{x}_p, \vec{\omega}_p)}}{\frac{\cancel{\int_{H^2} L_e(\mathbf{x}_p, \vec{\omega}) d\vec{\omega}}}{\int_A \int_{H^2} L_e(\mathbf{x}, \vec{\omega}) d\vec{\omega} d\mathbf{x}} \frac{\cancel{L_e(\mathbf{x}_p, \vec{\omega}_p)}}{\cancel{\int_{H^2} L_e(\mathbf{x}_p, \vec{\omega}) d\vec{\omega}}}} = \frac{\Phi}{M}$$

Total power of the light source

If you *perfectly importance sample* the emitted radiance, just take the *total power* and divide by # of *emitted* photons.

Photon Emission

- Photons carry power (flux) not radiance!
  - a fraction of the light source power
  - in most practical implementations, each photon carries multiple wavelengths (e.g. RGB)

Photon Emission Examples

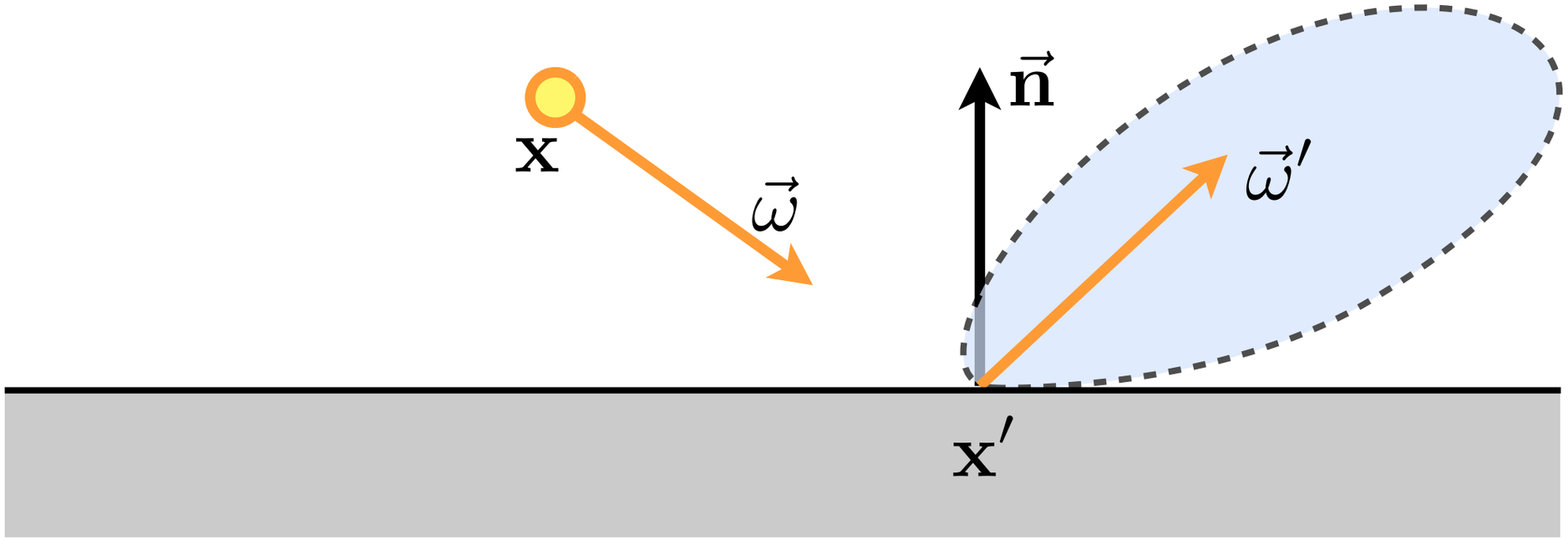
- Isotropic point light:
  - Generate uniform random direction over sphere
- Spotlight:
  - Generate uniform random direction within spherical cap
- Diffuse square light
  - Generate uniform random location on square
  - Generate cosine-weighted direction over hemisphere

Pseudocode

```
void generatePhotonMap()
    repeat:
        (l, Prob_l) = chooseRandomLight()
        (x, ω, Φ) = emitPhotonFromLight(l)
        tracePhoton(x, ω, Φ / Prob_l)
    until we have enough photons;
    divide all photon powers by number of emitted photons

void tracePhoton(x, ω, Φ)
```

Pseudocode




```
void tracePhoton(x, ω, Φ)
    (x’, n) = nearestSurfaceHit(x, ω)
    possiblyStorePhoton(x’, ω, Φ)
    (ω’, pdf) = sampleBSDF(x’, -ω)
    Φ’ = Φ * absDot(n, ω’) * BSDF(x’, -ω, ω’) / pdf
    tracePhoton(x’, ω’, Φ’)
```

Storing Photons

- Store only on diffuse (or moderately glossy) surfaces
  - Specular surfaces need to be handled using path tracing from the camera
- Stored data: [36 bytes]

```
struct Photon
{
    float position[3];
    float power[3];
    float direction[3];
};
```

Storing Photons

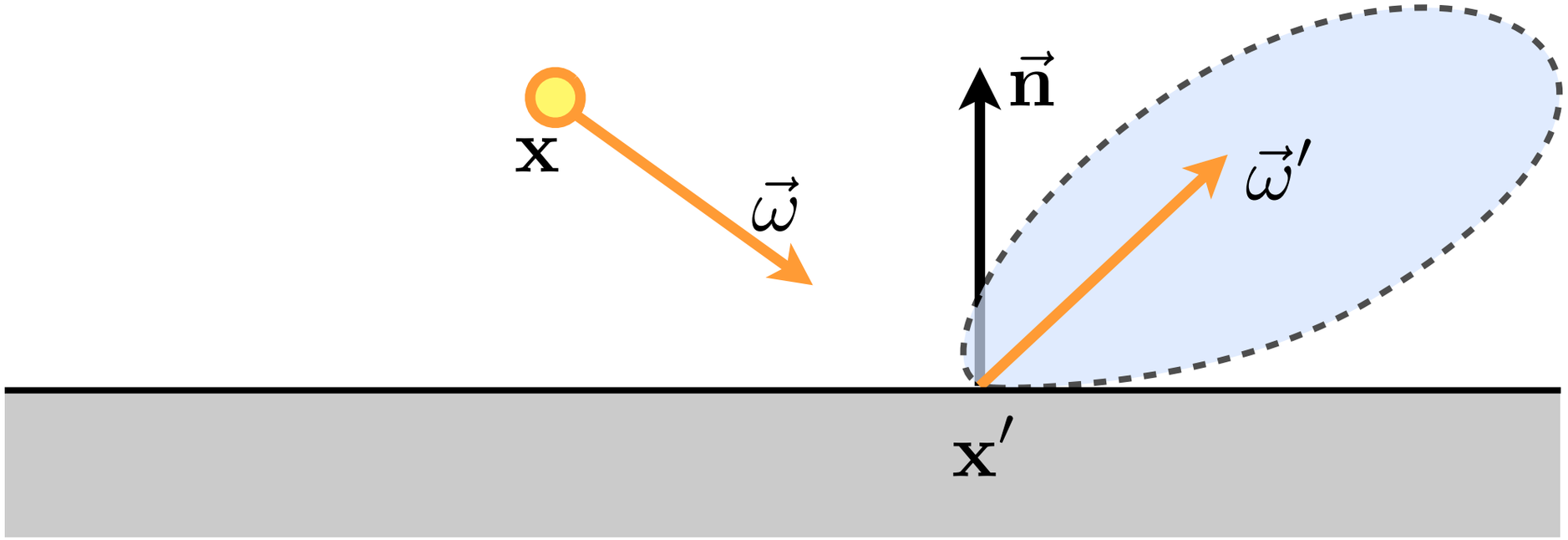
- Store only on diffuse (or moderately glossy) surfaces
  - Specular surfaces need to be handled using path tracing from the camera
- Stored data: [18 bytes, better!]

```
struct Photon
{
    float position[3];
    char power[4];          // Packed RGBE format
    char phi, theta;        // Packed direction
};
```

Scattering of Photons

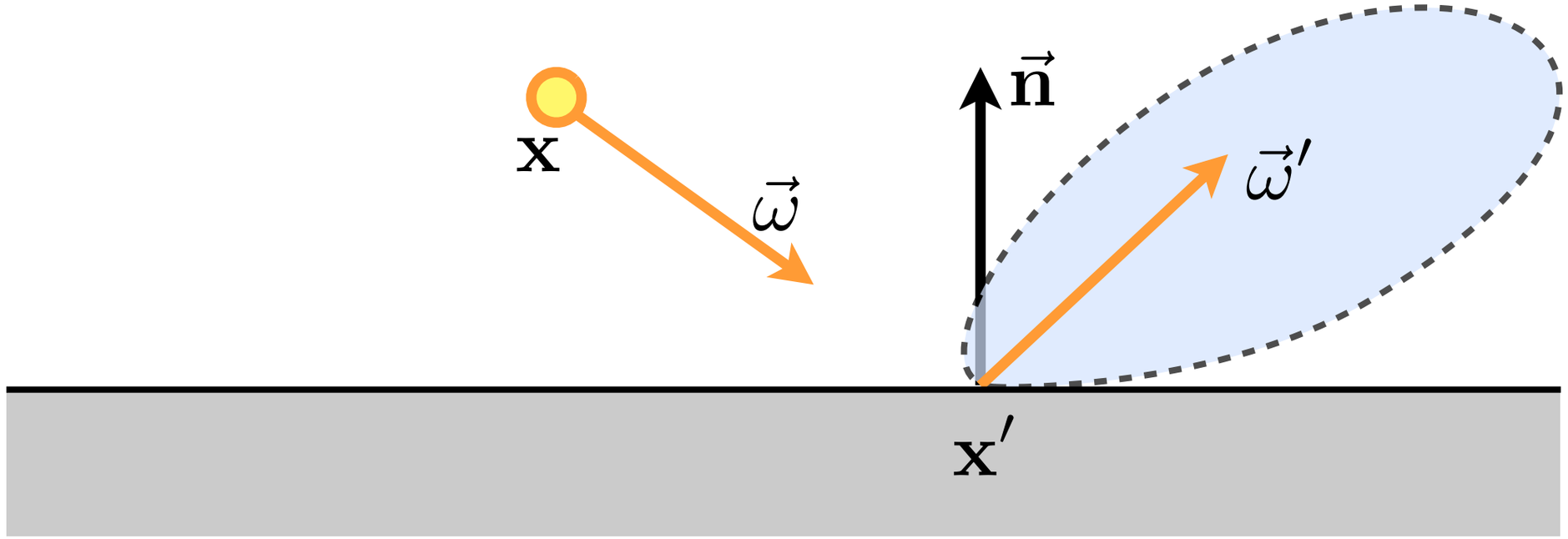
- Photons can be:
  - absorbed or scattered (reflected or refracted)
  - BSDF sampling chooses either reflection or refraction
  - the power of the scattered photon is lowered to account for absorption
- Problem:
  - as photons bounce they carry less and less power
  - ideally all stored photons would have the same power
  - also, when should we terminate the recursion?
- Solution: Russian roulette

Pseudocode




```
void tracePhoton(x, ω, Φ)
    (x’, n) = nearestSurfaceHit(x, ω)
    possiblyStorePhoton(x’, ω, Φ)
    (ω’, pdf) = sampleBSDF(x’, -ω)
    Φ’ = Φ * absDot(n, ω’) * BSDF(x’, -ω, ω’) / pdf
    tracePhoton(x’, ω’, Φ’)
```

Pseudocode




```
void tracePhoton(x, ω, Φ)
    (x’, n) = nearestSurfaceHit(x, ω)
    possiblyStorePhoton(x’, ω, Φ)
    (ω’, pdf) = sampleBSDF(x’, -ω)
    Φ’ = Φ * absDot(n, ω’) * BSDF(x’, -ω, ω’) / pdf
    if survivedRussianRoulette(Φ, Φ’)
        tracePhoton(x’, ω’, Φ’)
```

Photon Path Termination

- Probabilistically terminate the photon walk using Russian roulette

$$E[F'] = p \cdot 0 + (1-p) \cdot \frac{E[F]}{1-p} = E[F]$$

- Use termination probability:

$$p = 1 - \min\left(1, \frac{\Phi'}{\Phi}\right)$$

Photon Path Termination

```
bool survivedRussianRoulette(Φ, Φ’)
    p = 1 - min(1, Φ’/Φ)
    if rand() < p:
        // terminate
        return false
    else:
        // continue with re-weighted power
        Φ’ = Φ’/(1-p)
        return true
```

if Φ’/Φ is smaller than 1, then Φ’= Φ’/(1-p) = Φ

i.e. the scattered photon has the same power!

Russian Roulette Example

- 300 photons with power 1.0 W hit a surface with reflectance 50%
- Instead of reflecting 300 photons with power 0.5 W, RR will make ~150 photons continue with power 1.0 W
- Very important!

Visual Break

 Image courtesy of B. Ybarra (rendered with VRay)

Photon Mapping

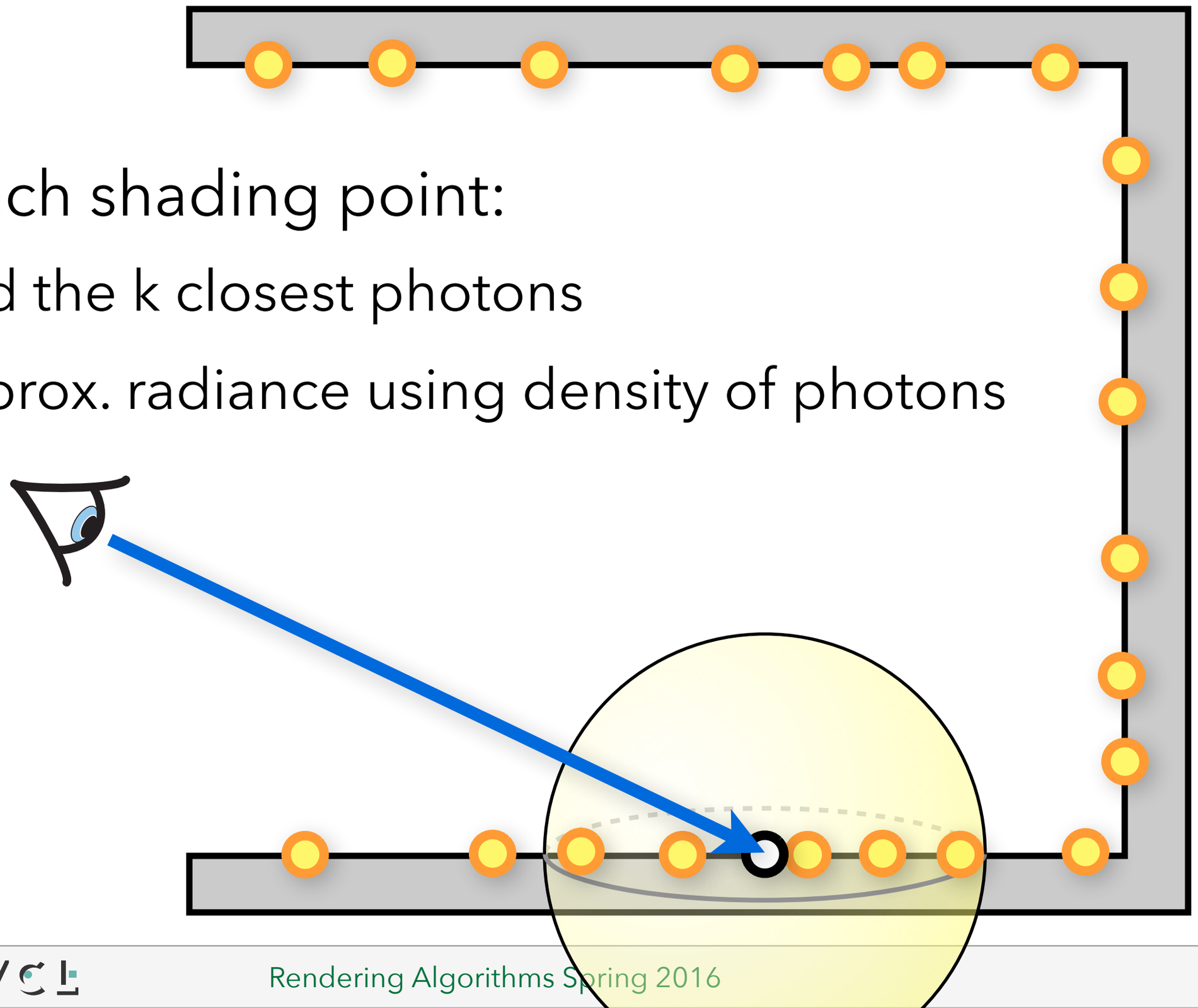
- A two-pass algorithm:
  - Pass 1: Tracing of photons from light sources, and caching them in a photon map
  - Pass 2: Tracing from the eye and approximating indirect illumination using the photons

Photon Mapping

- A two-pass algorithm:
  - Pass 1: Tracing of photons from light sources, and caching them in a photon map
  - Pass 2: Tracing from the eye and approximating indirect illumination using the photons

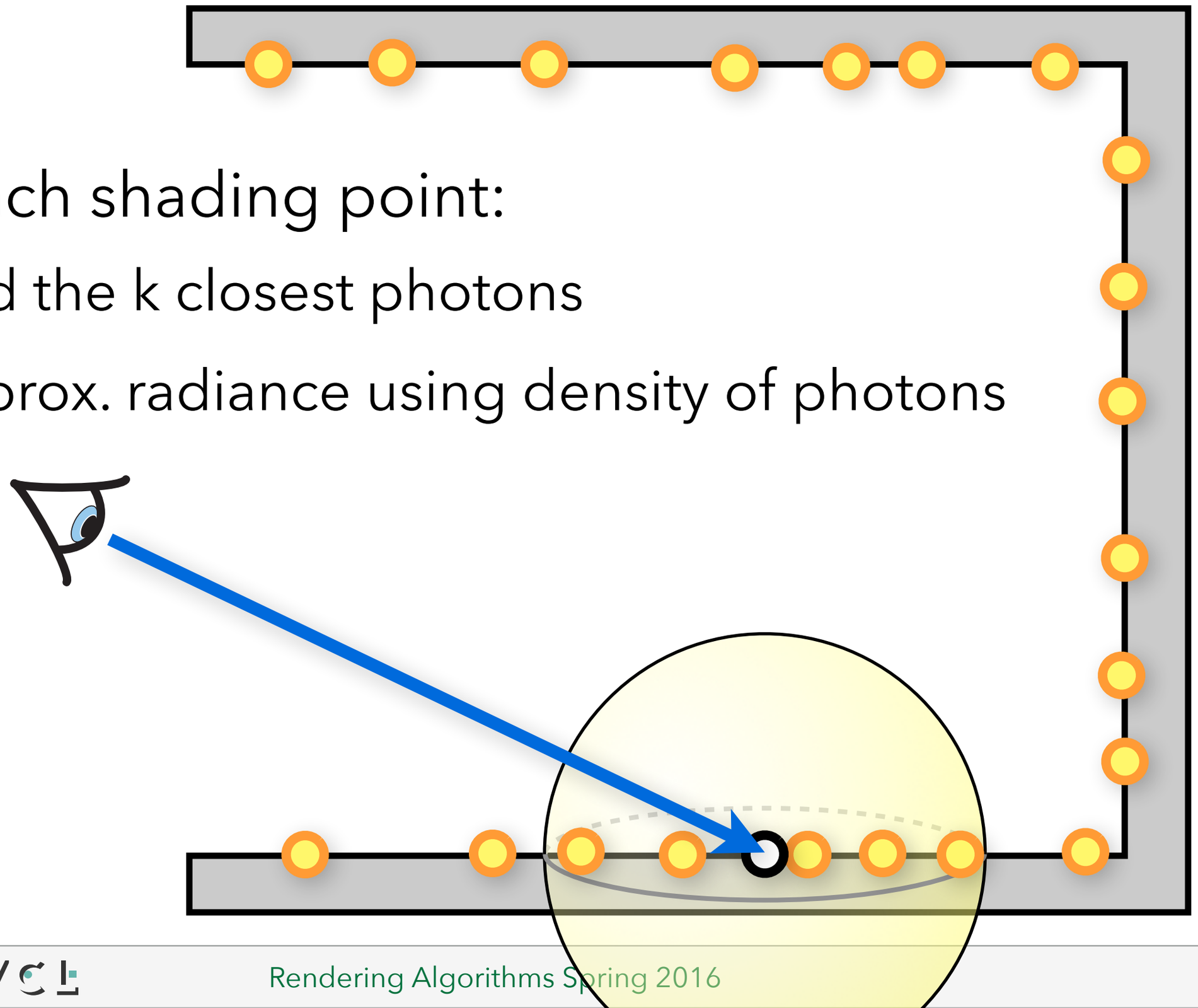
Rendering

- For each shading point:
  - Find the k closest photons
  - Approx. radiance using density of photons



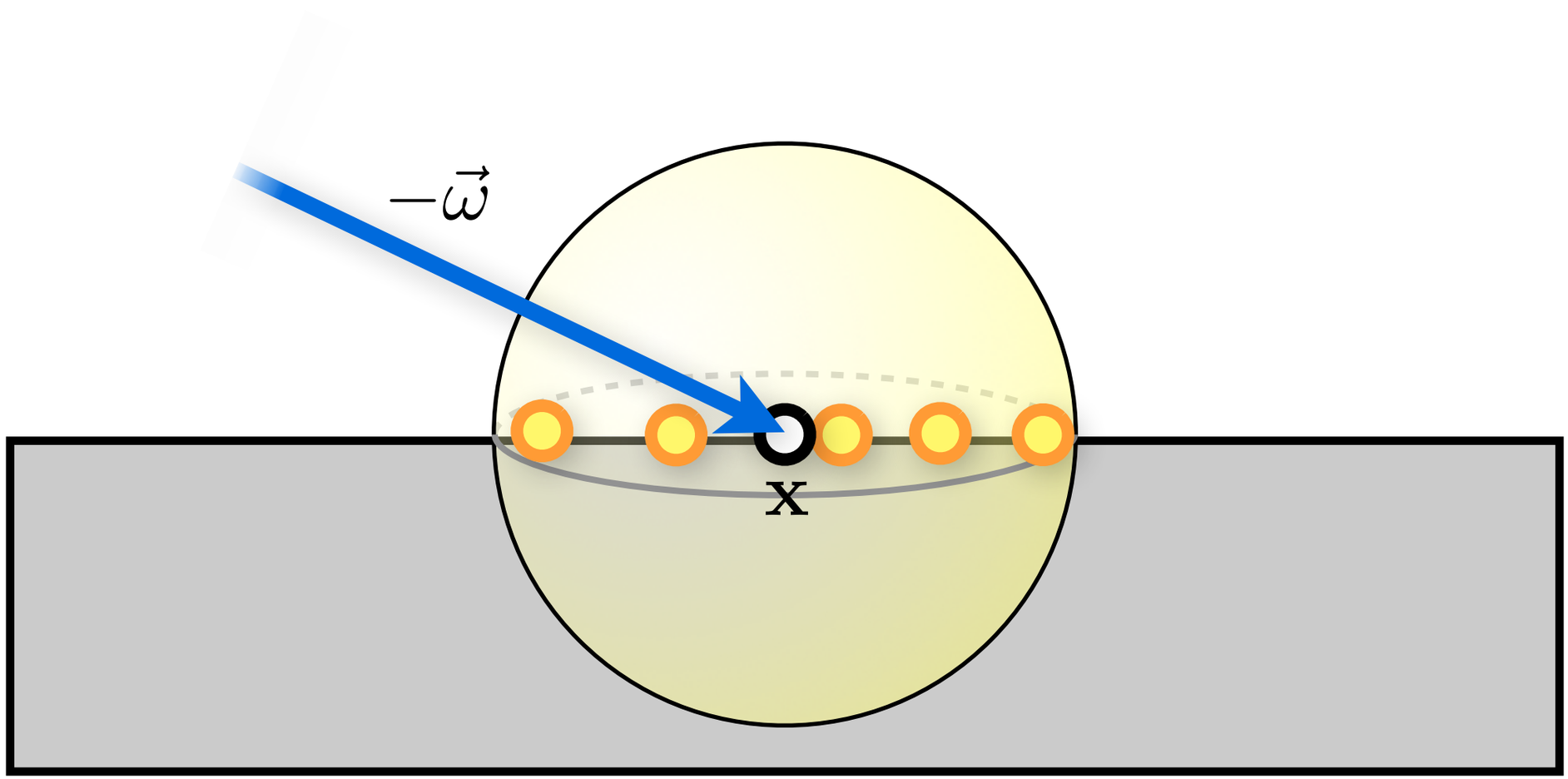
Rendering

- For each shading point:
  - Find the k closest photons
  - Approx. radiance using density of photons



The Radiance Estimate

- Based on kernel density estimation
  - Non-parametric way of estimating the probability density of a random variable (photon density)



The Radiance Estimate

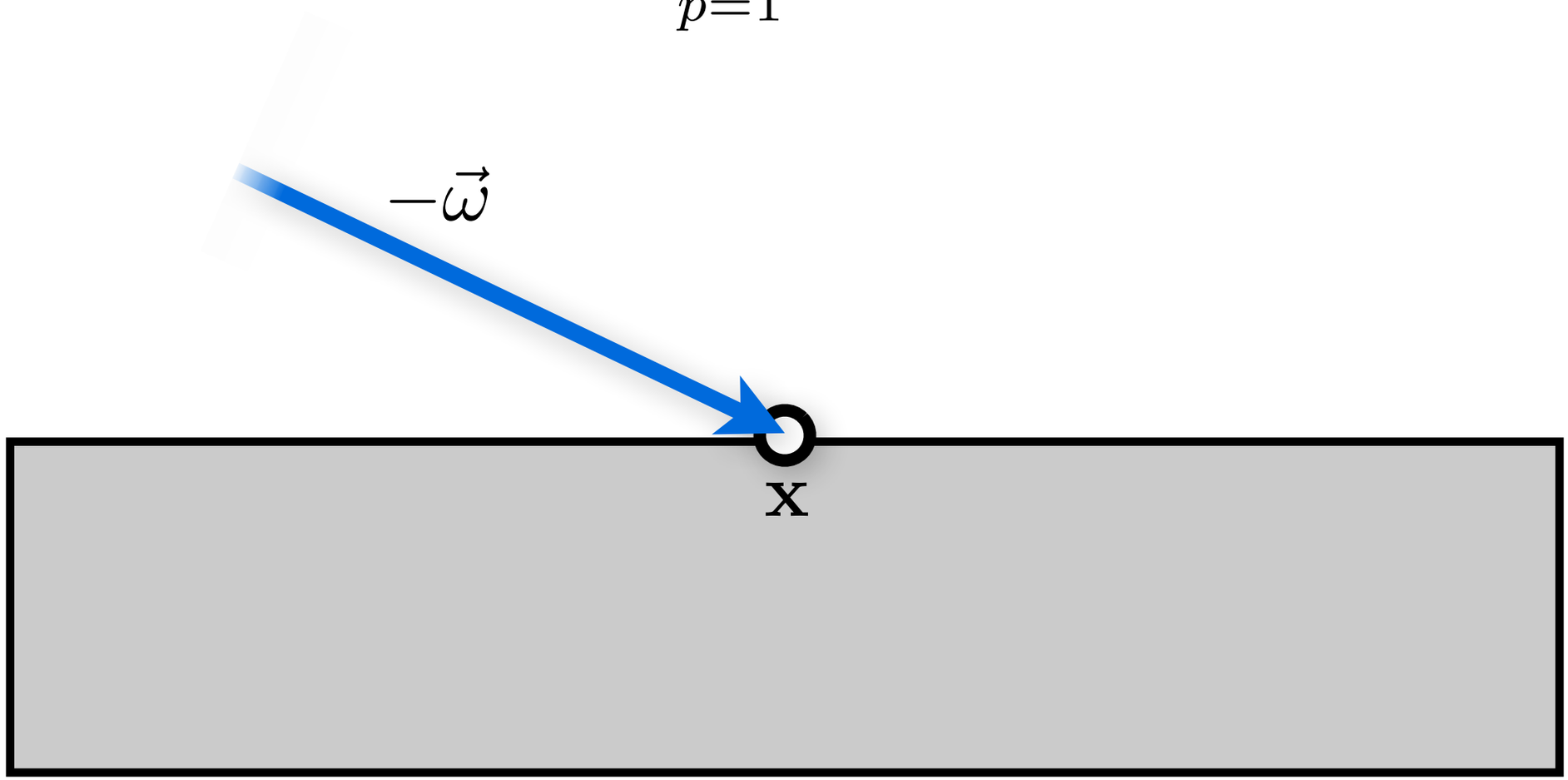
- Based on kernel density estimation
  - Non-parametric way of estimating the probability density of a random variable (photon density)

$$
\begin{aligned}
L_r(\mathbf{x}, \vec{\omega}) &= \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L_i(\mathbf{x}, \vec{\omega}') \cos\theta' d\vec{\omega}' \\
&= \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) \frac{d\Phi^2(\mathbf{x}, \vec{\omega}')}{\cos\theta' d\vec{\omega}' dA} \cos\theta' d\vec{\omega}' \\
&= \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) \frac{d\Phi^2(\mathbf{x}, \vec{\omega}')}{dA} \\
&\approx \sum_{p=1}^{n} f_r(\mathbf{x}, \vec{\omega}_p, \vec{\omega}) \frac{\Delta\Phi_p(\mathbf{x}, \vec{\omega}_p)}{\Delta A}
\end{aligned}
$$

The Radiance Estimate

- Approach 1: first define area, then find photons

$$L_r(\mathbf{x}, \vec{\omega}) \approx \sum_{p=1}^{k} f_r(\mathbf{x}, \vec{\omega}_p, \vec{\omega}) \frac{\Phi_p}{A}$$



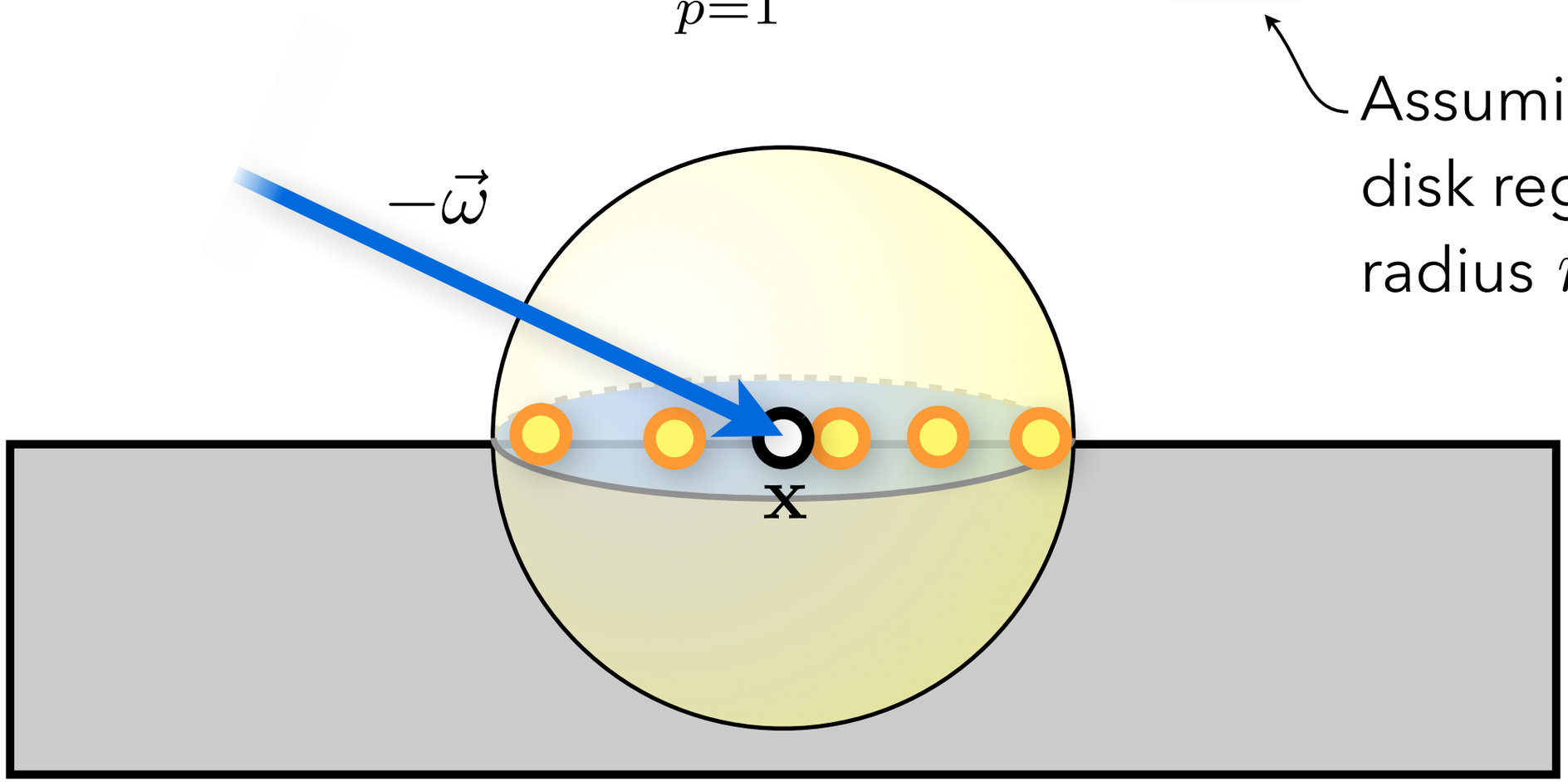
The Radiance Estimate

- Approach 1: first define area, then find photons

# of photons within disk

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

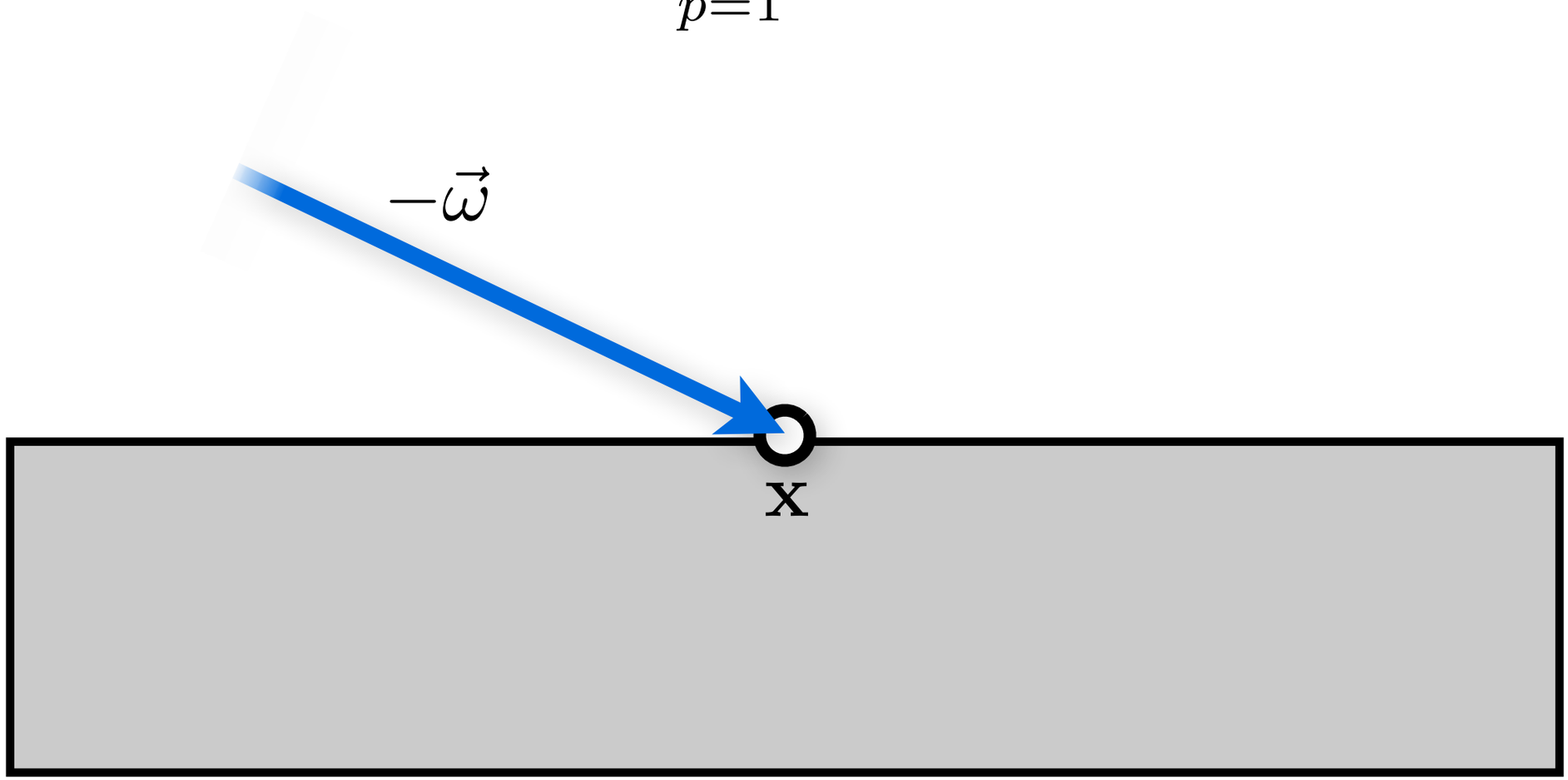
Assuming a disk region of radius r



The Radiance Estimate

- Approach 2: first find k nearest photons, then define area

$$L_r(\mathbf{x}, \vec{\omega}) \approx \sum_{p=1}^{k} f_r(\mathbf{x}, \vec{\omega}_p, \vec{\omega}) \frac{\Phi_p}{A}$$



The Radiance Estimate

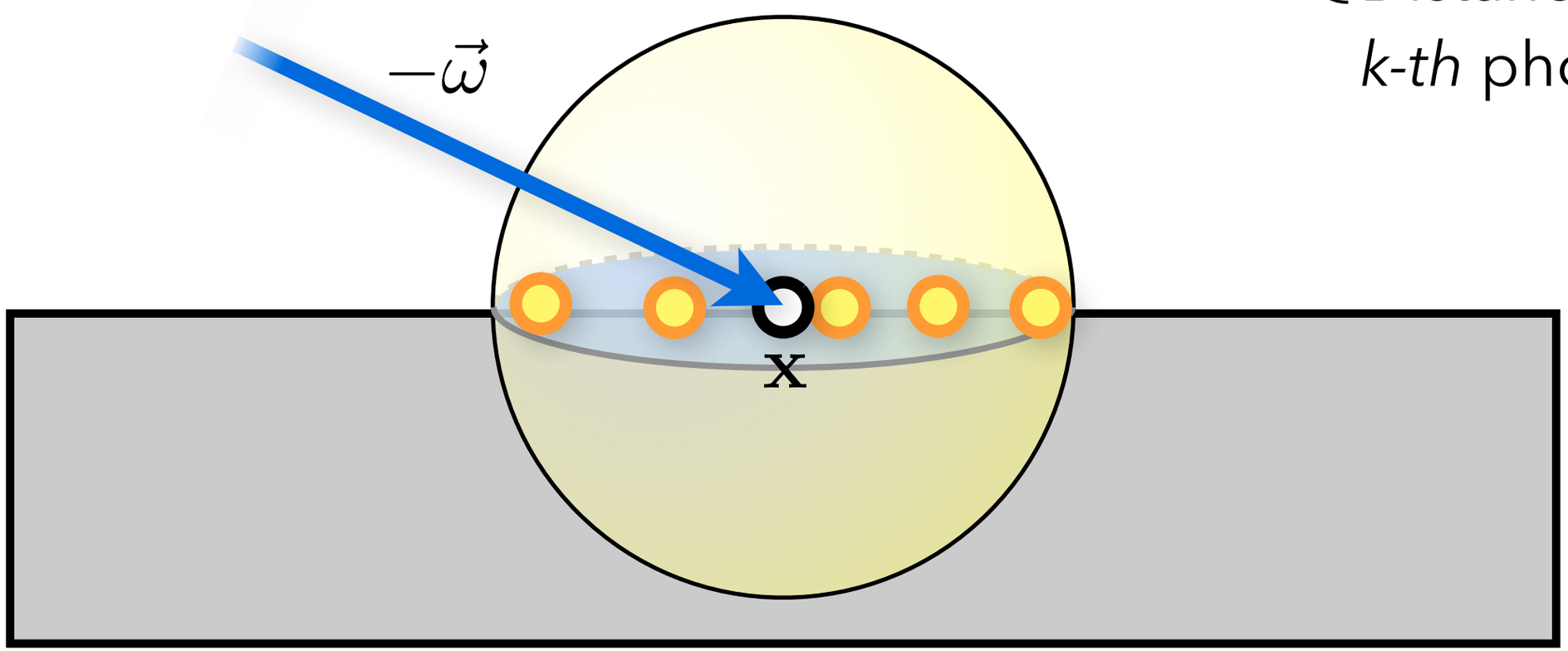
- Approach 2: first find k nearest photons, then define area

Ignore the *k-th* photon

[García et al. 2012]

$$L_r(\mathbf{x}, \vec{\omega}) \approx \sum_{p=1}^{\boxed{k-1}} f_r(\mathbf{x}, \vec{\omega}_p, \vec{\omega}) \frac{\Phi_p}{\boxed{\pi r_k^2}}$$

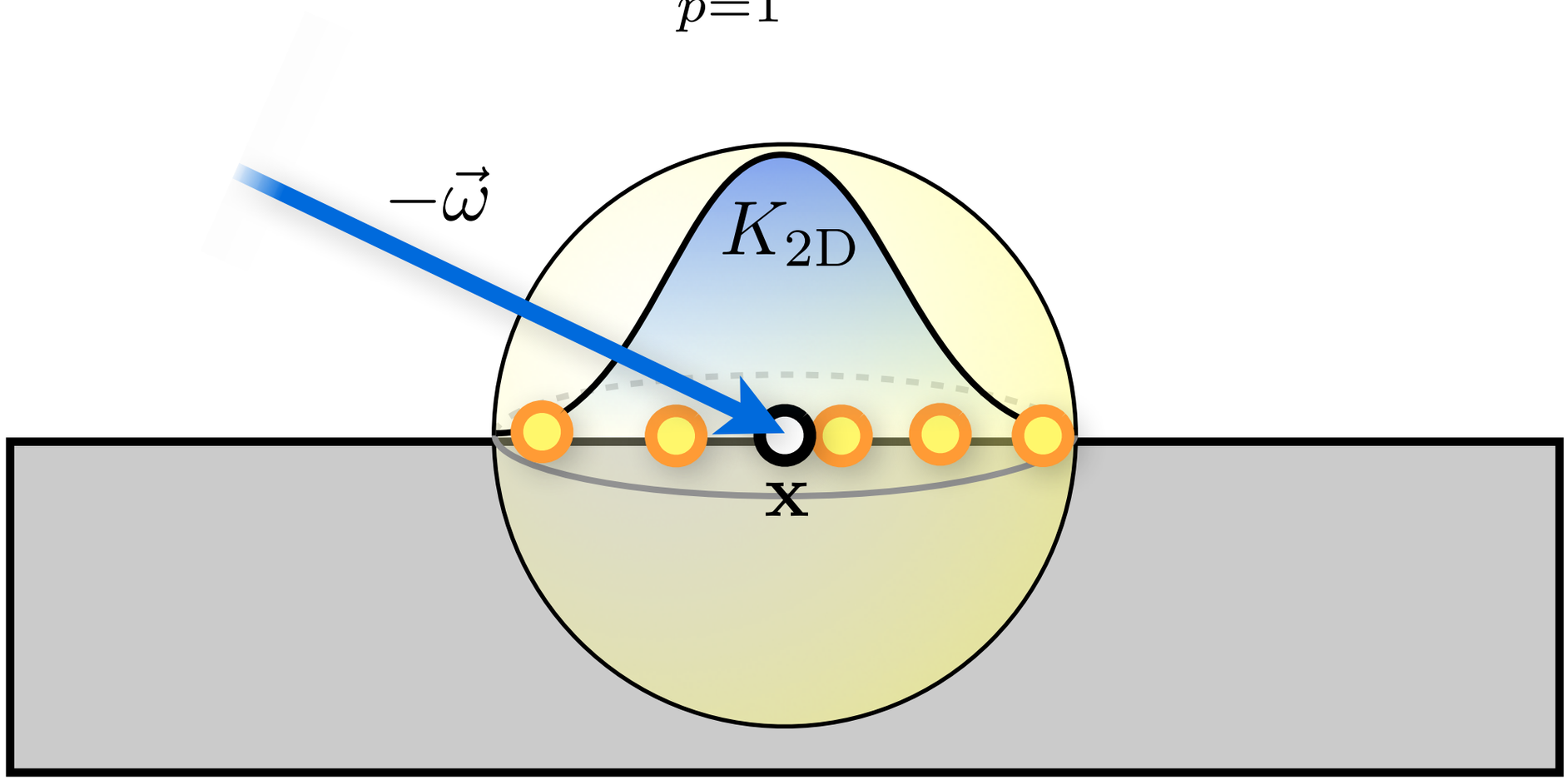
Distance to the *k-th* photon



The Radiance Estimate

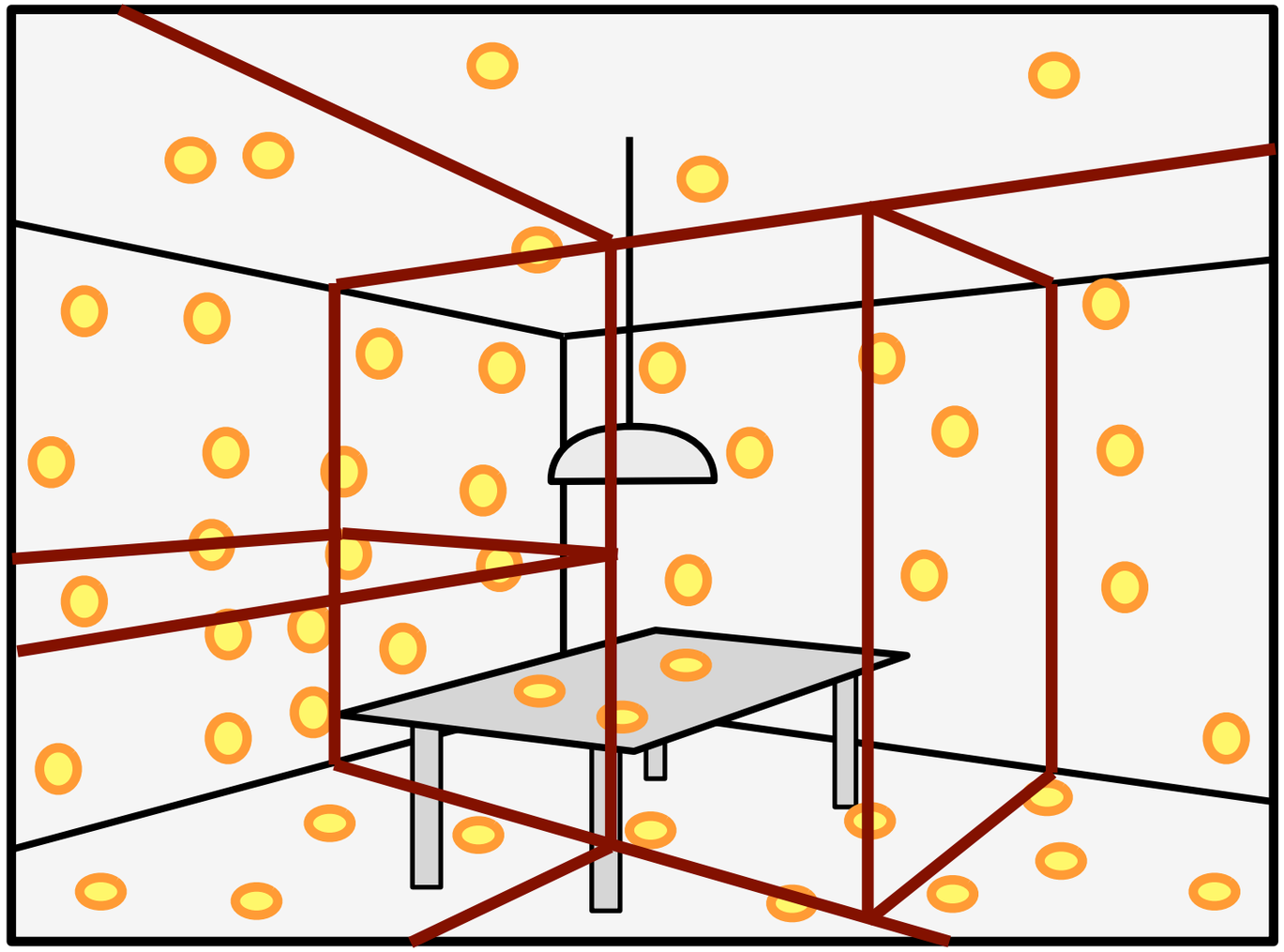
- Using a non-constant kernel:

$$L_r(\mathbf{x}, \vec{\omega}) \approx \sum_{p=1}^{k-1} f_r(\mathbf{x}, \vec{\omega}_p, \vec{\omega}) \Phi_p \boxed{K_{2\mathrm{D}}(r_p, r_k)}$$

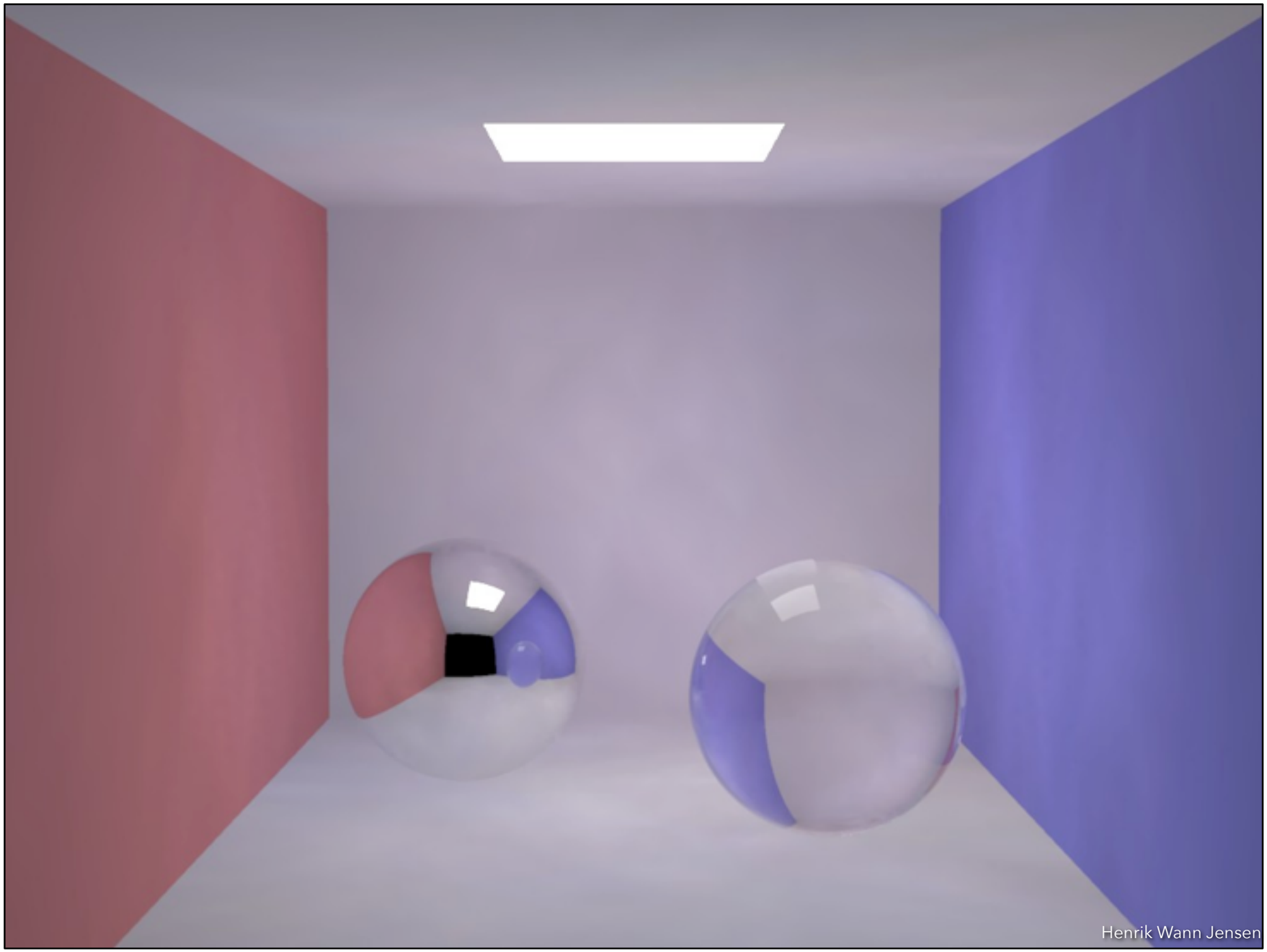


The Photon Map Data Structure

- Requirements:
  - Compact (we want many photons)
  - Fast nearest neighbor search
- KD-tree

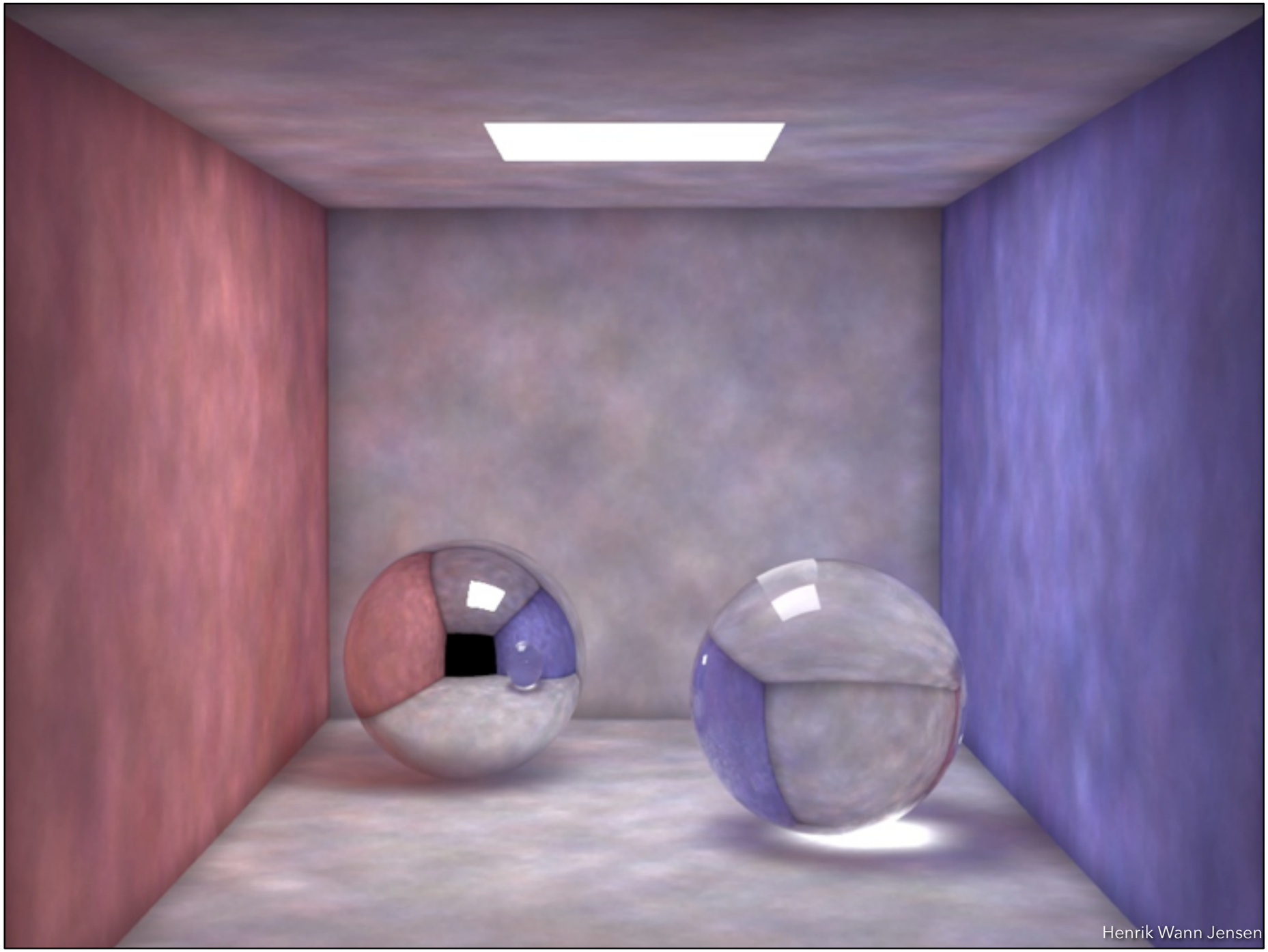


Photon Mapping



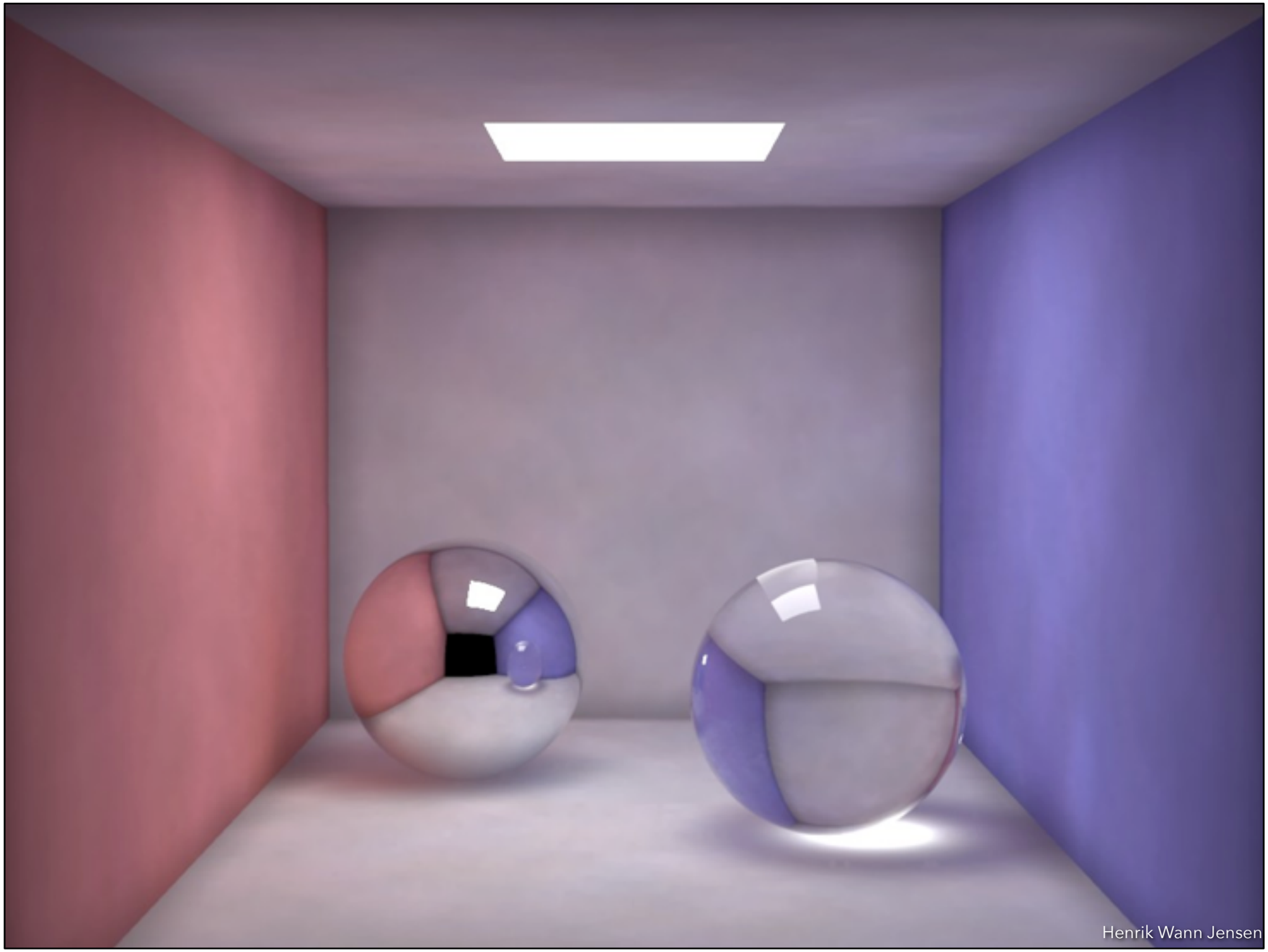
200 photons / 50 photons in radiance estimate

Photon Mapping



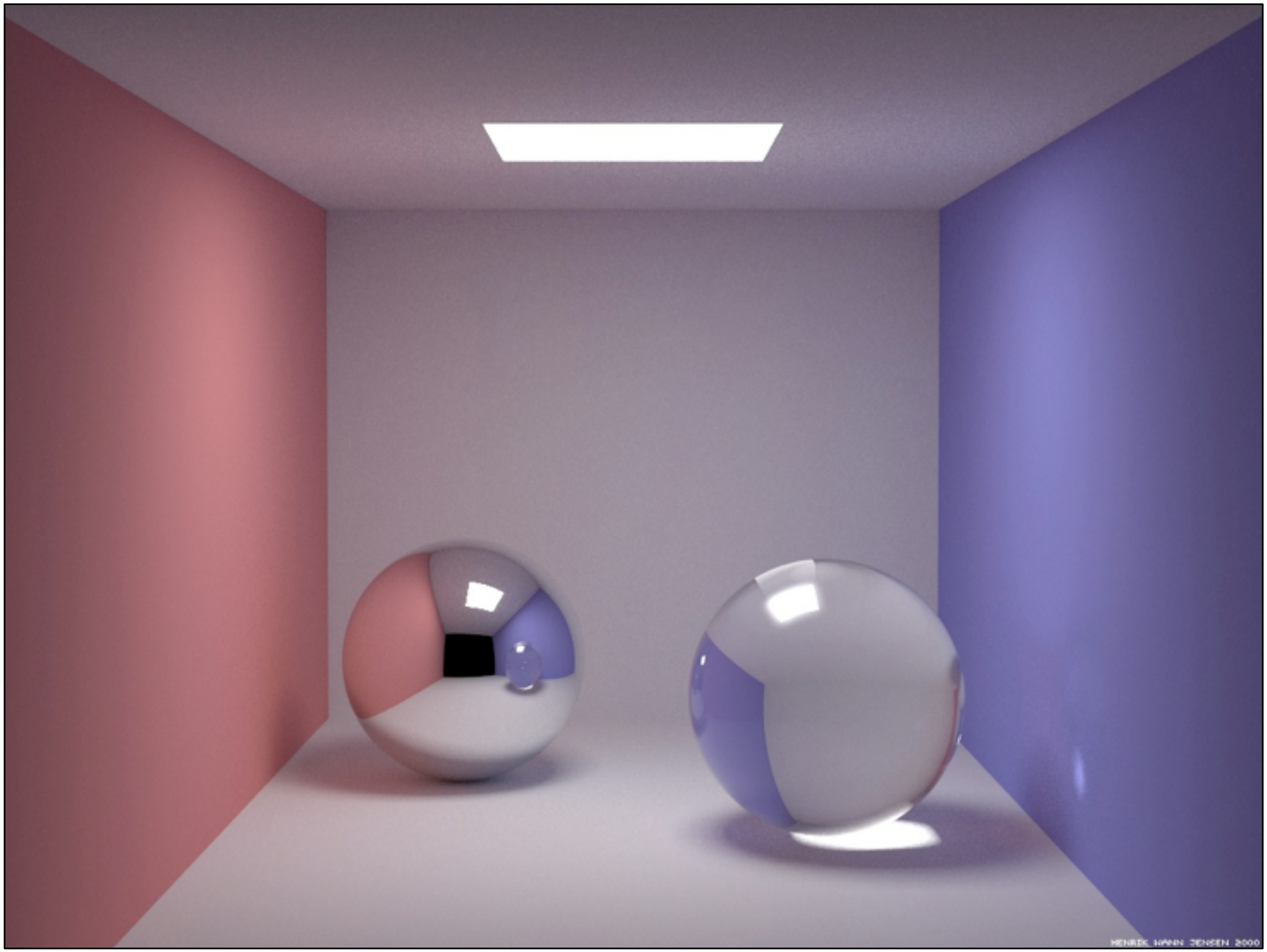
100,000 photons / 50 photons in radiance estimate

Photon Mapping

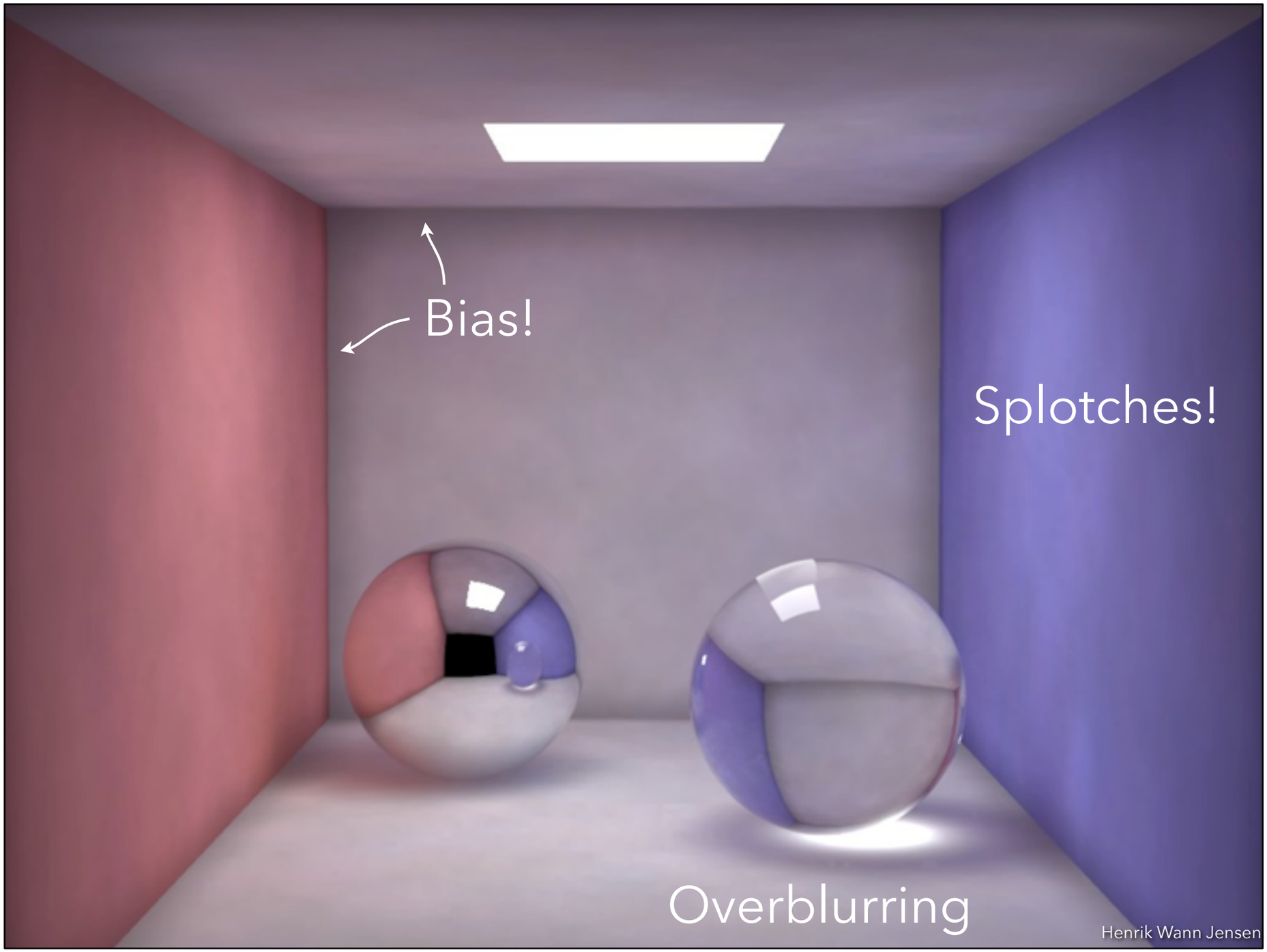


500,000 photons / 500 photons in radiance estimate

Path Tracing



Photon Mapping



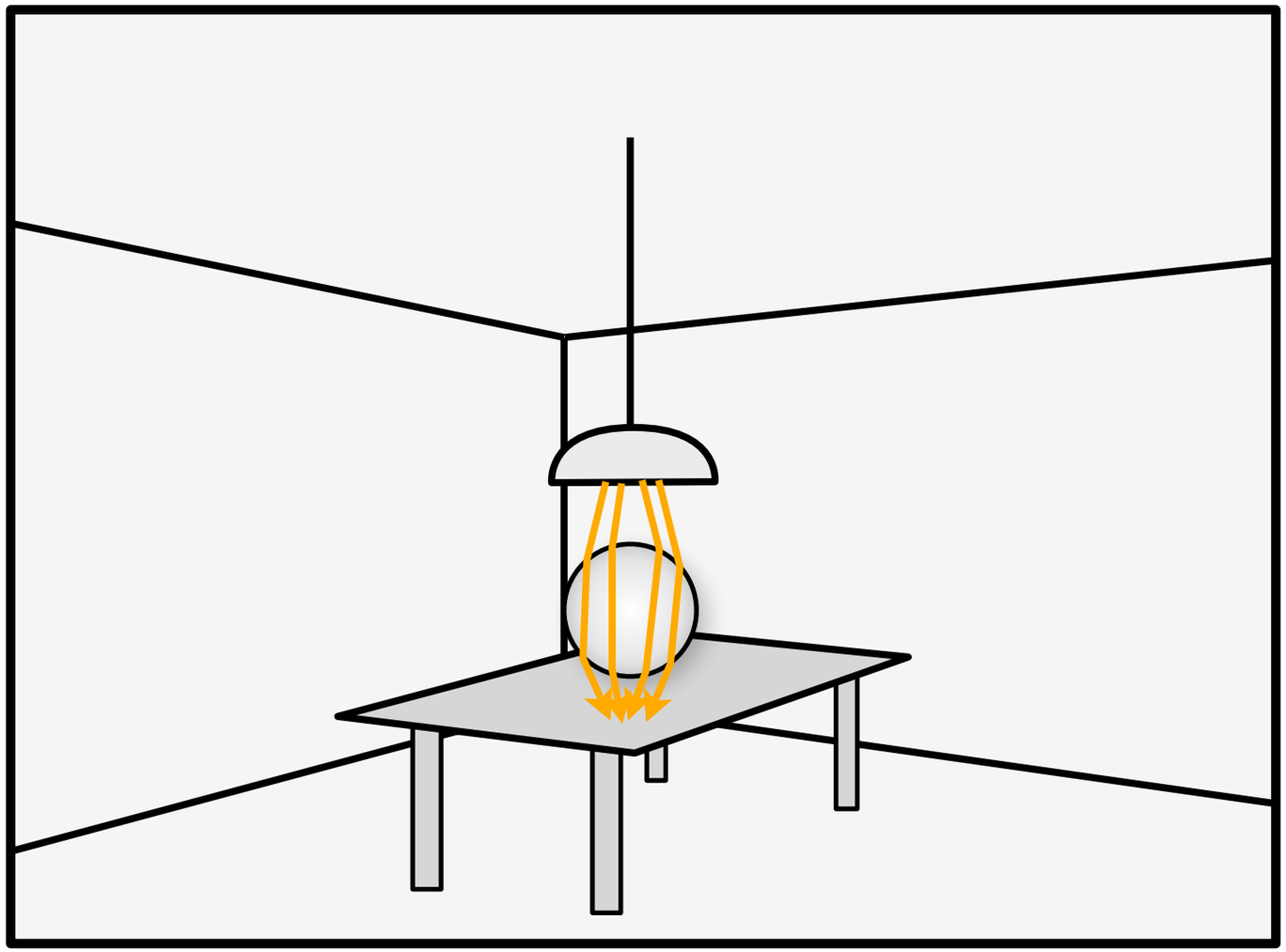
500,000 photons / 500 photons in radiance estimate

Photon Mapping

- Radiance estimate contains error/bias
  - Produces darker/brighter, blotchy, blurry appearance
  - Requires *many* photons for high quality
- Split up lighting computation into components:
  - Direct lighting
  - Caustics (caustic photon map)
  - Diffuse indirect illumination (global photon map)

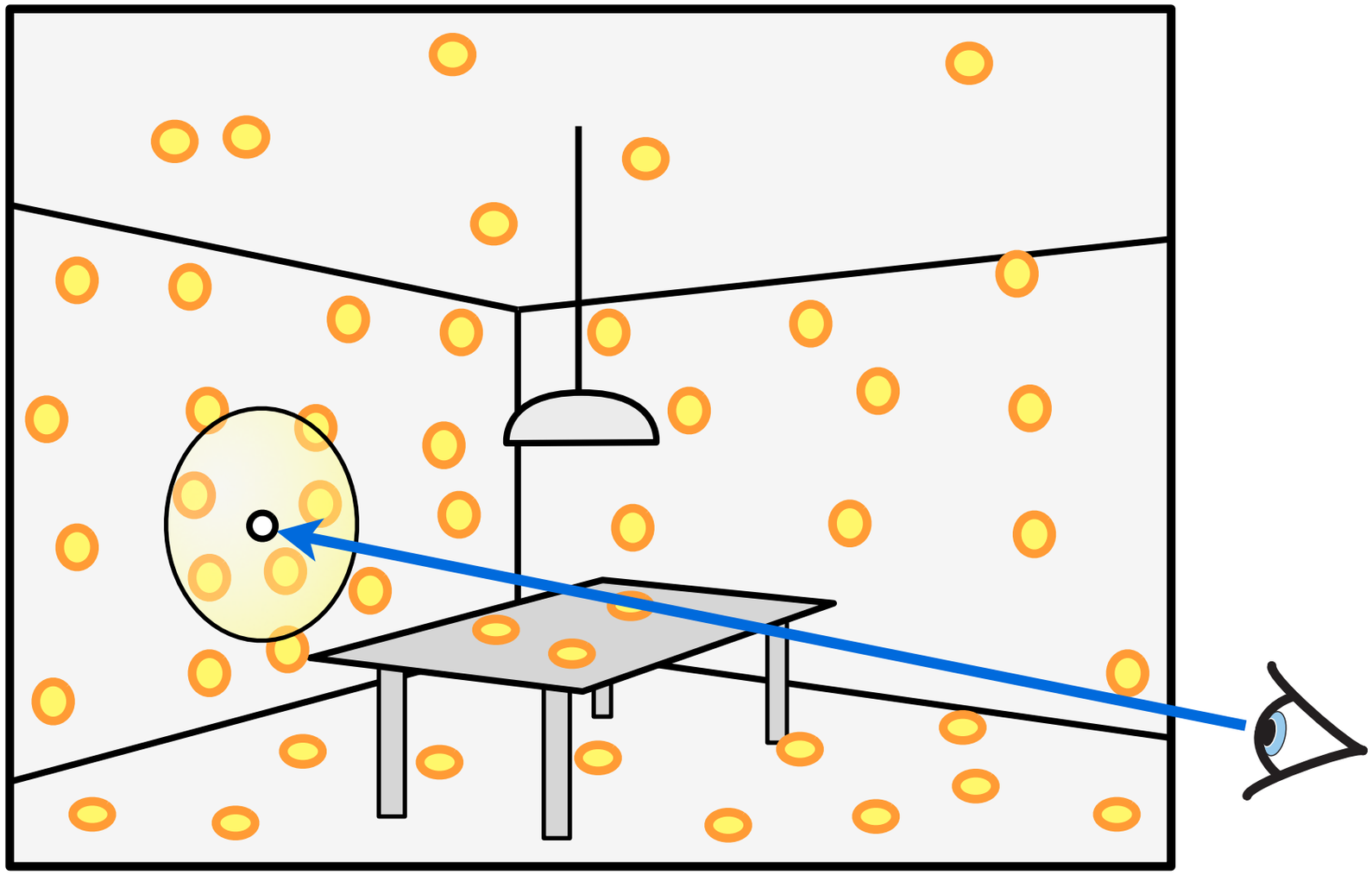
Improving Caustics

- Higher quality photon map for caustics
  - Only stores LS+D paths
  - Many photons shot directly at specular objects



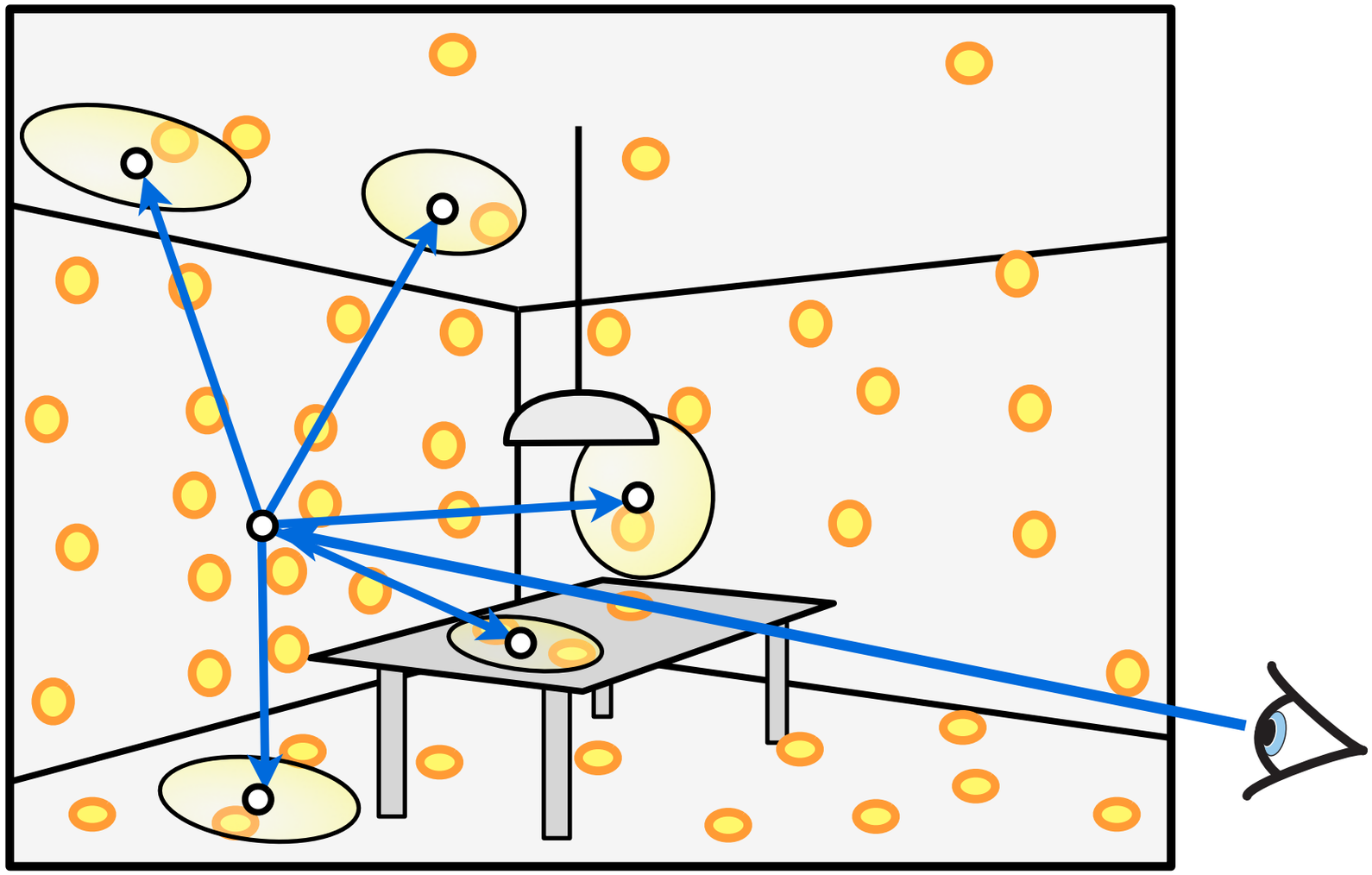
Improving Diffuse Indirect

- Original approach: direct density estimation



Improving Diffuse Indirect

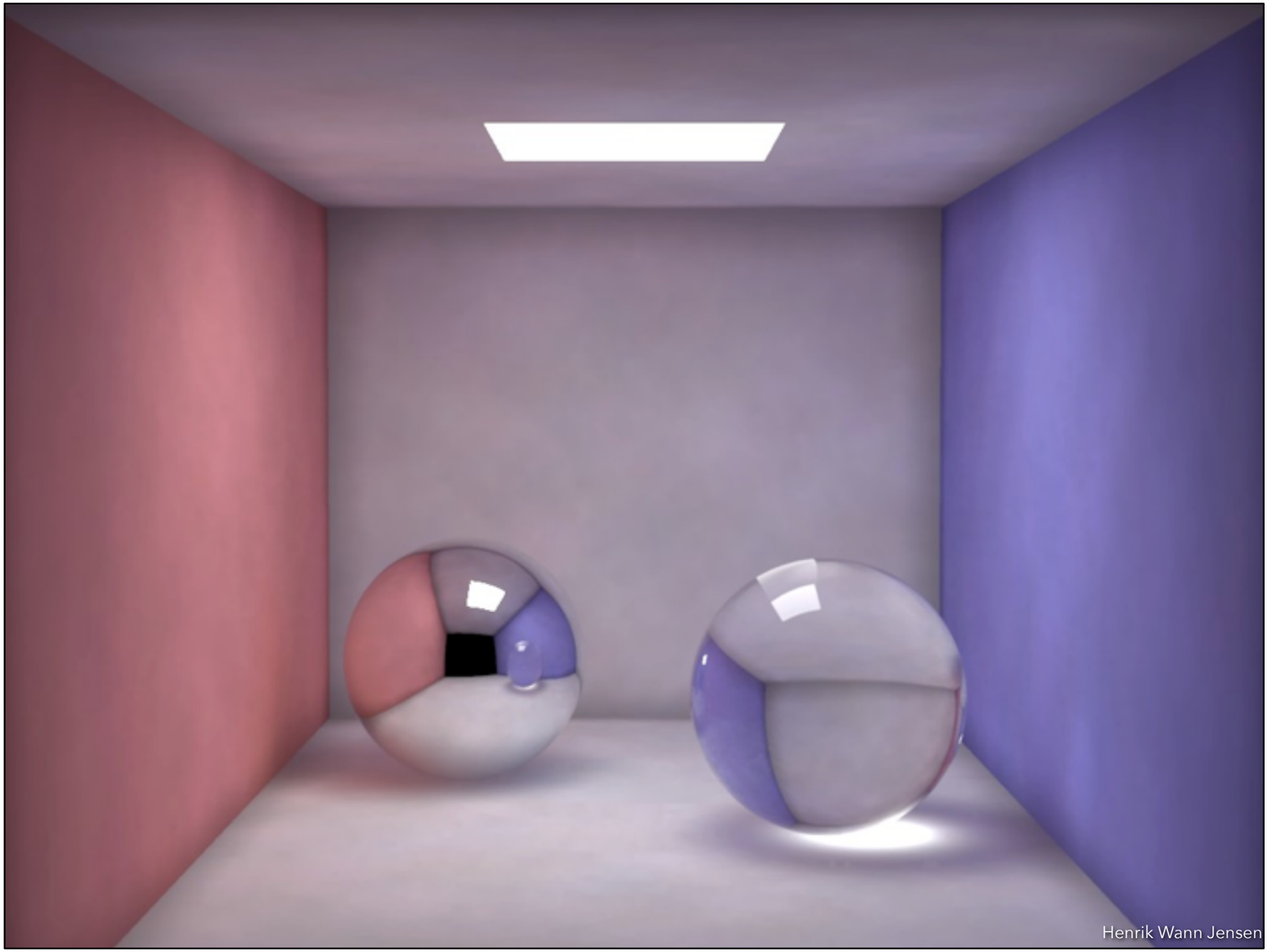
- Improved approach: using *final gather*
  - Use MC ray tracing to push the (blotchy) density estimation 1+ bounces further from the camera



Improved Photon Mapping

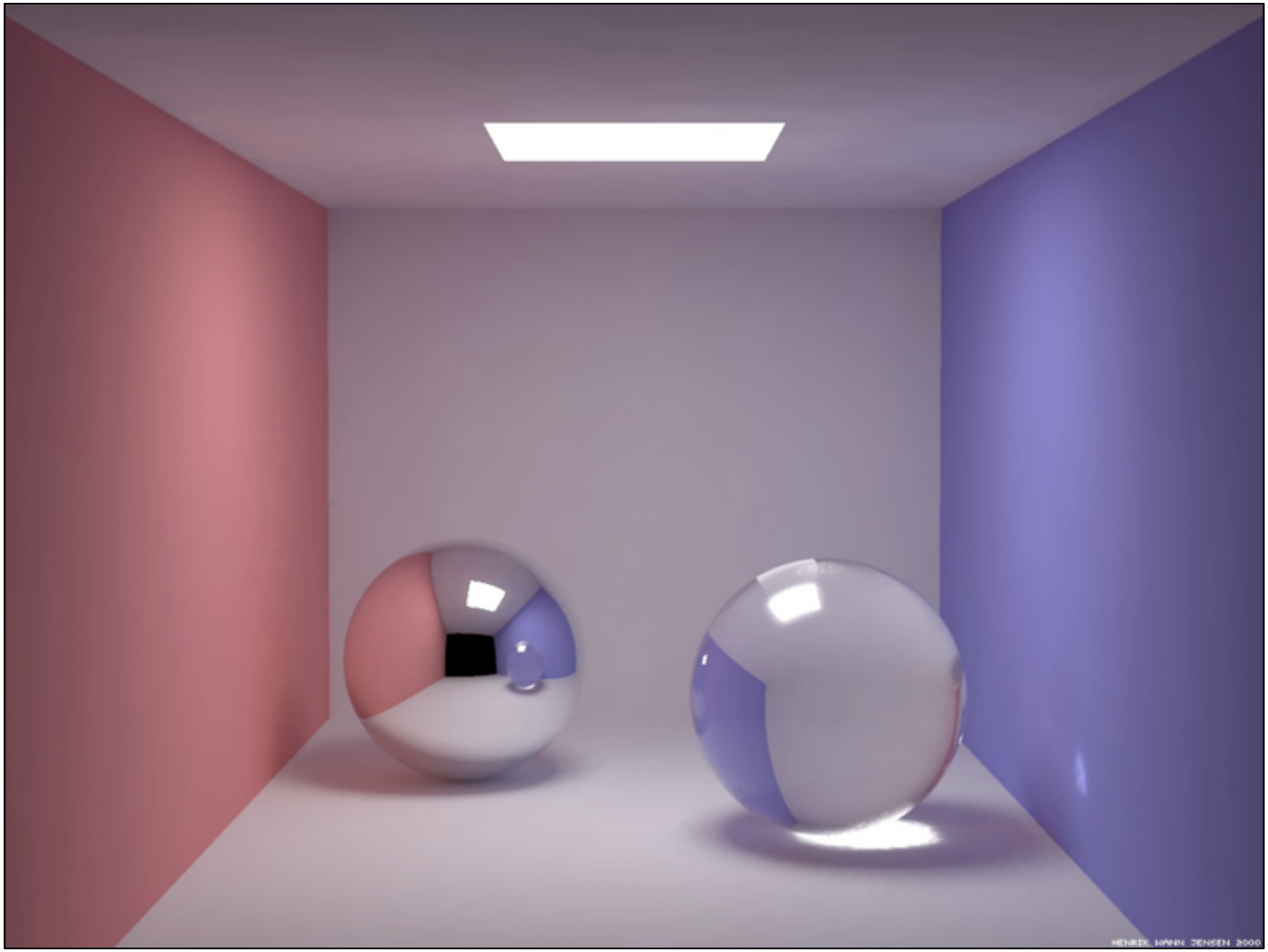
- Camera tracing
  - Trace camera paths until they hit the first non-specular surface point $\mathbf{x}$
- At $\mathbf{x}$ we sum:
  - Emission
  - **Direct illumination:** Estimated using ray tracing
  - **Caustics:** Density estimation at $\mathbf{x}$ using only the *caustic* photon map
  - **Remaining indirect:** Estimated with final gathering, by shooting gather rays from $\mathbf{x}$ and performing density estimation in *global* photon map at first non-specular vertex $\mathbf{y}$

Photon Mapping



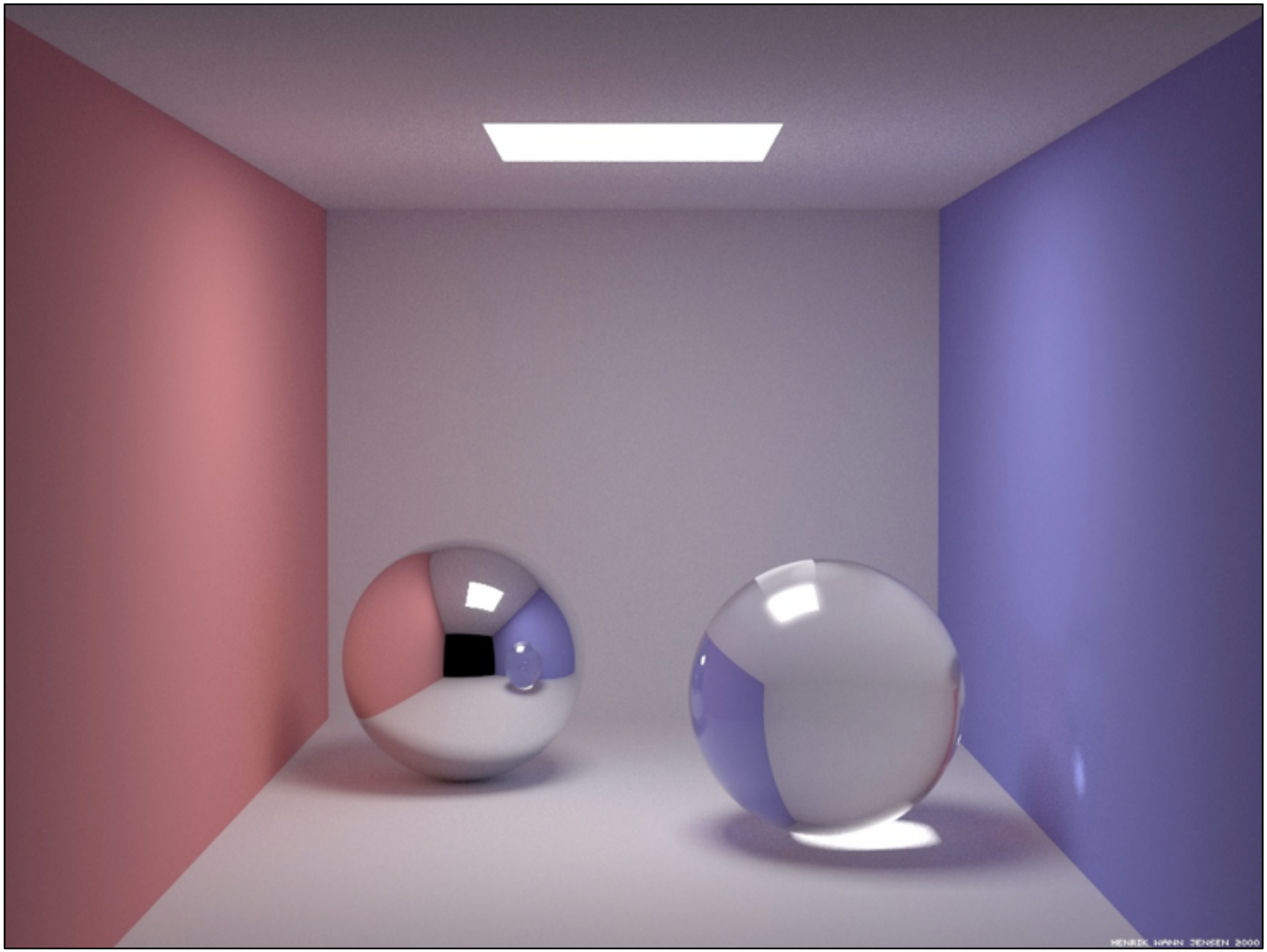
500000 photons / 500 photons in radiance estimate

Photon Mapping (Improved)



final gather + global photon map (200000) + caustic photon map (50000)

Path Tracing



Validation Tests

- Test idea 1:
  - store only direct photons
  - visualize photon map directly
  - compare to standard direct illumination
  - should look identical with many photons
- Test idea 2:
  - create a perfectly transparent sphere (IOR = 1.0)
  - store only caustic photons
  - render direct illumination + caustics
  - shadow should disappear

Photon Mapping - Summary

- Advantages
  - Handles difficult paths more robustly than unbiased algorithms
  - Consistent estimator
  - Reuse of computation (photons)
- Disadvantages
  - Bias shows up in many different forms
  - Requires additional data structure (KD-tree)
  - No progressive rendering
  - Large memory footprint

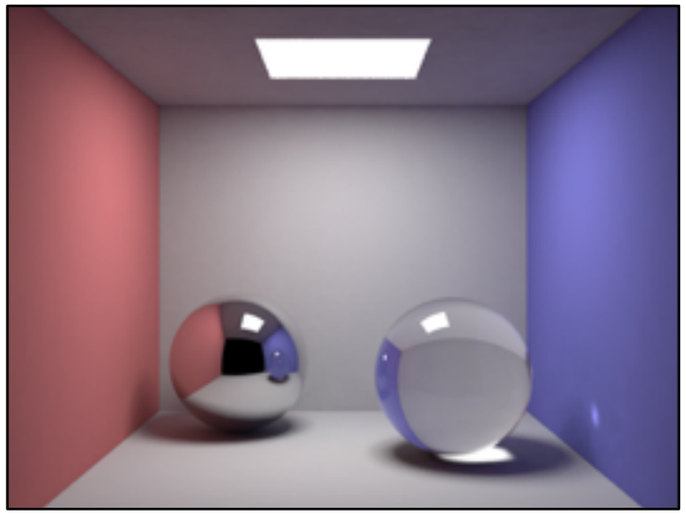
Visual Break

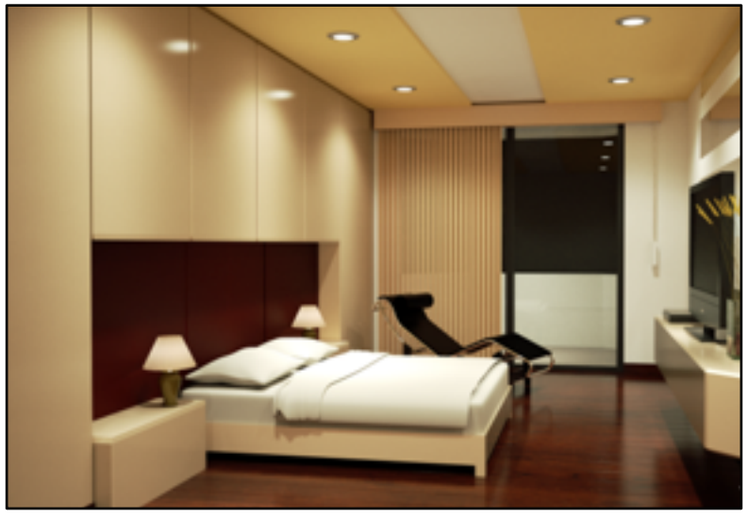
Light Sources in the Real World

- Complex shape
- Covered with transparent materials
- Only a small part emits light

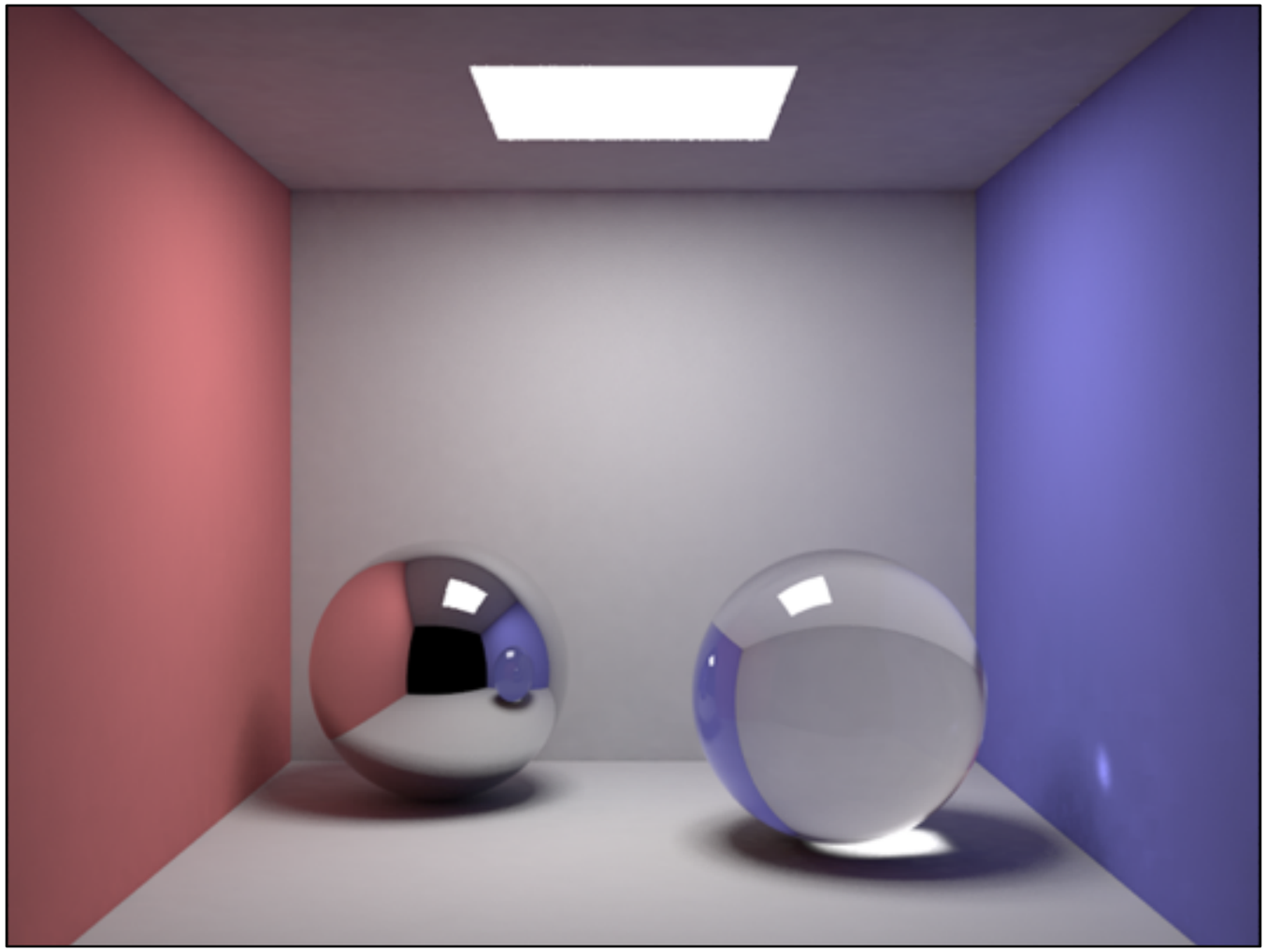
Light Sources in CG

- Simple shape
- Bare light source
- Entire part emits light

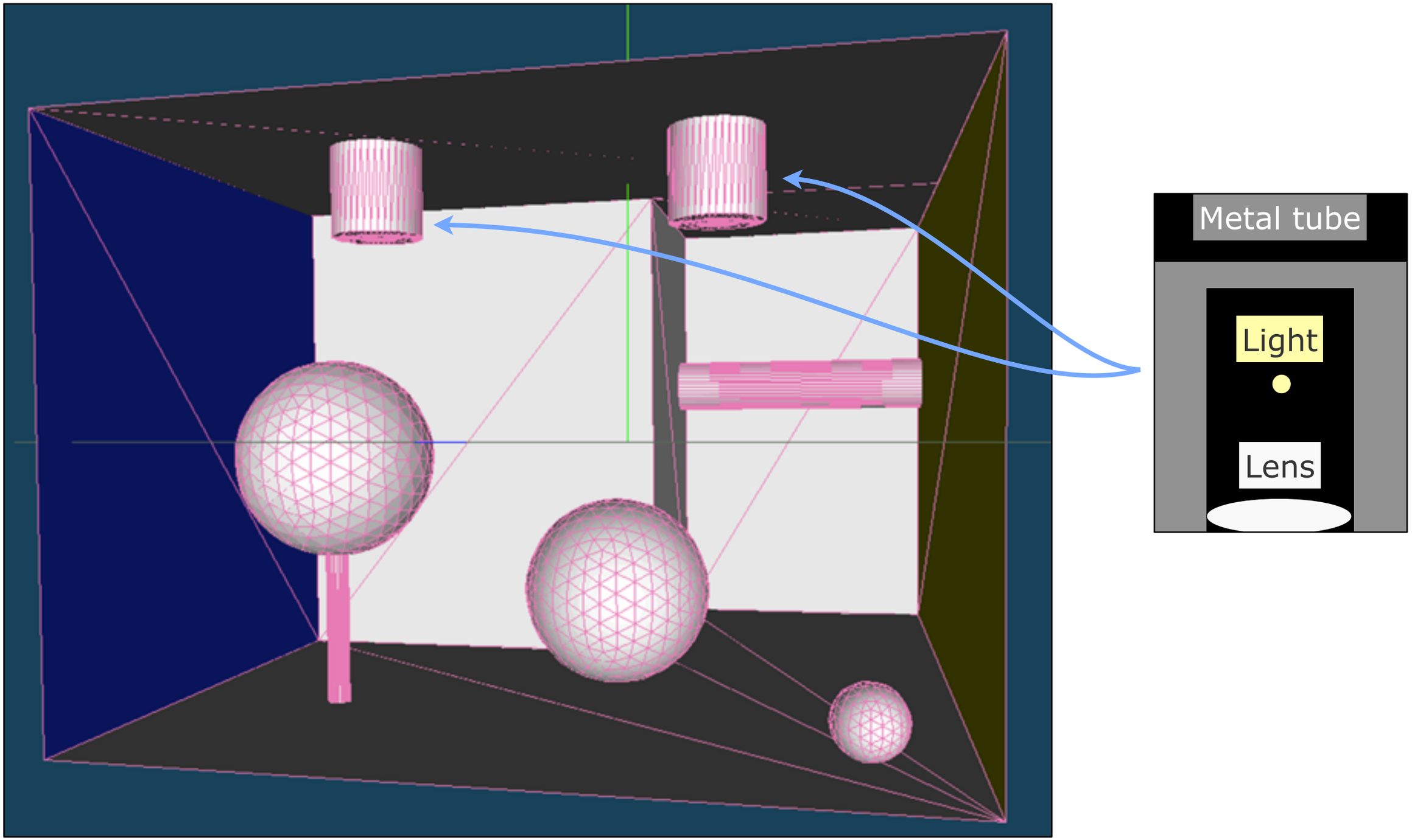




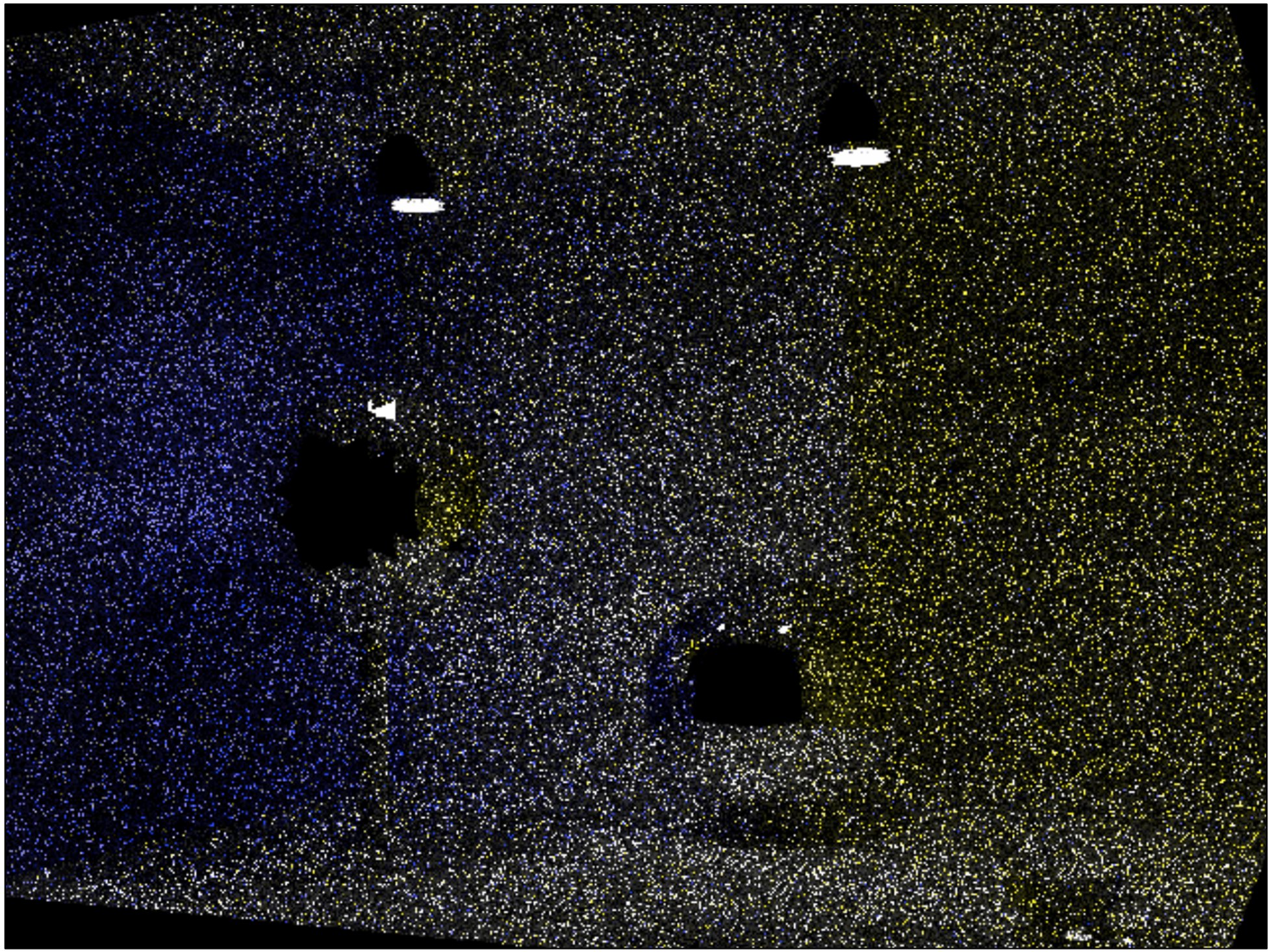
Why?



Scene with "Realistic" Lights

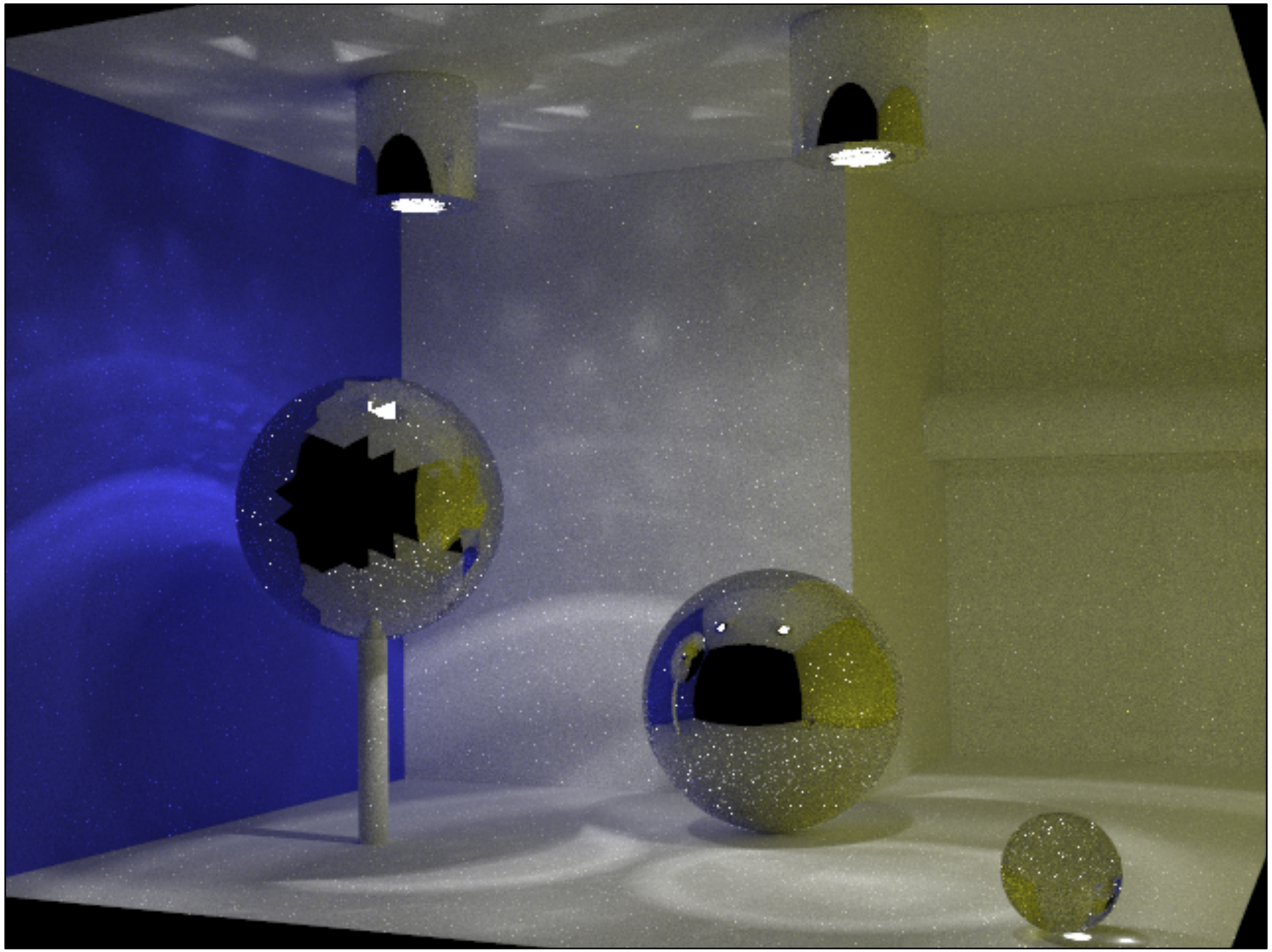


Path Tracing



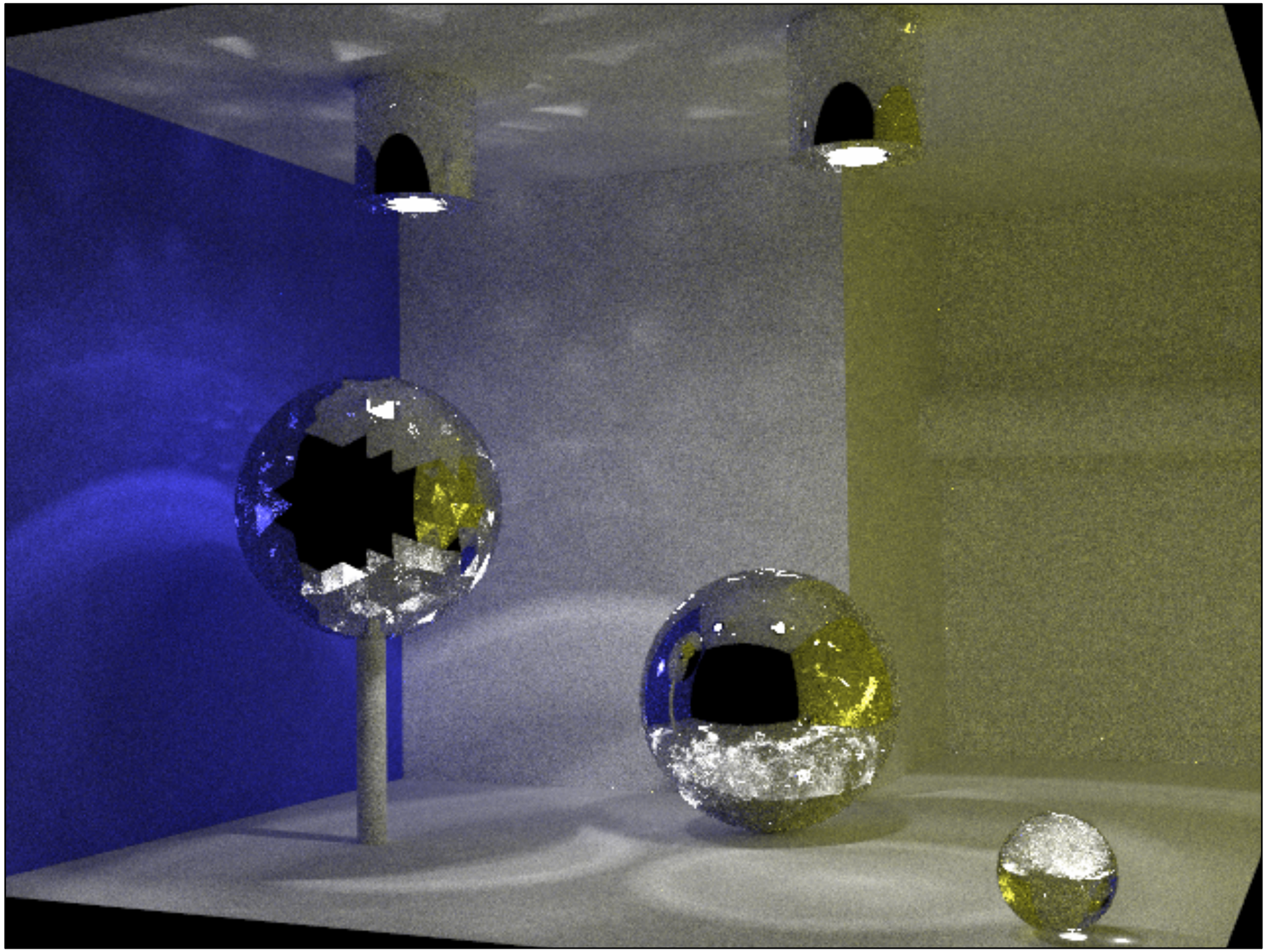
Images courtesy of T. Hachisuka

Bidirectional Path Tracing



Images courtesy of T. Hachisuka

Metropolis Light Transport

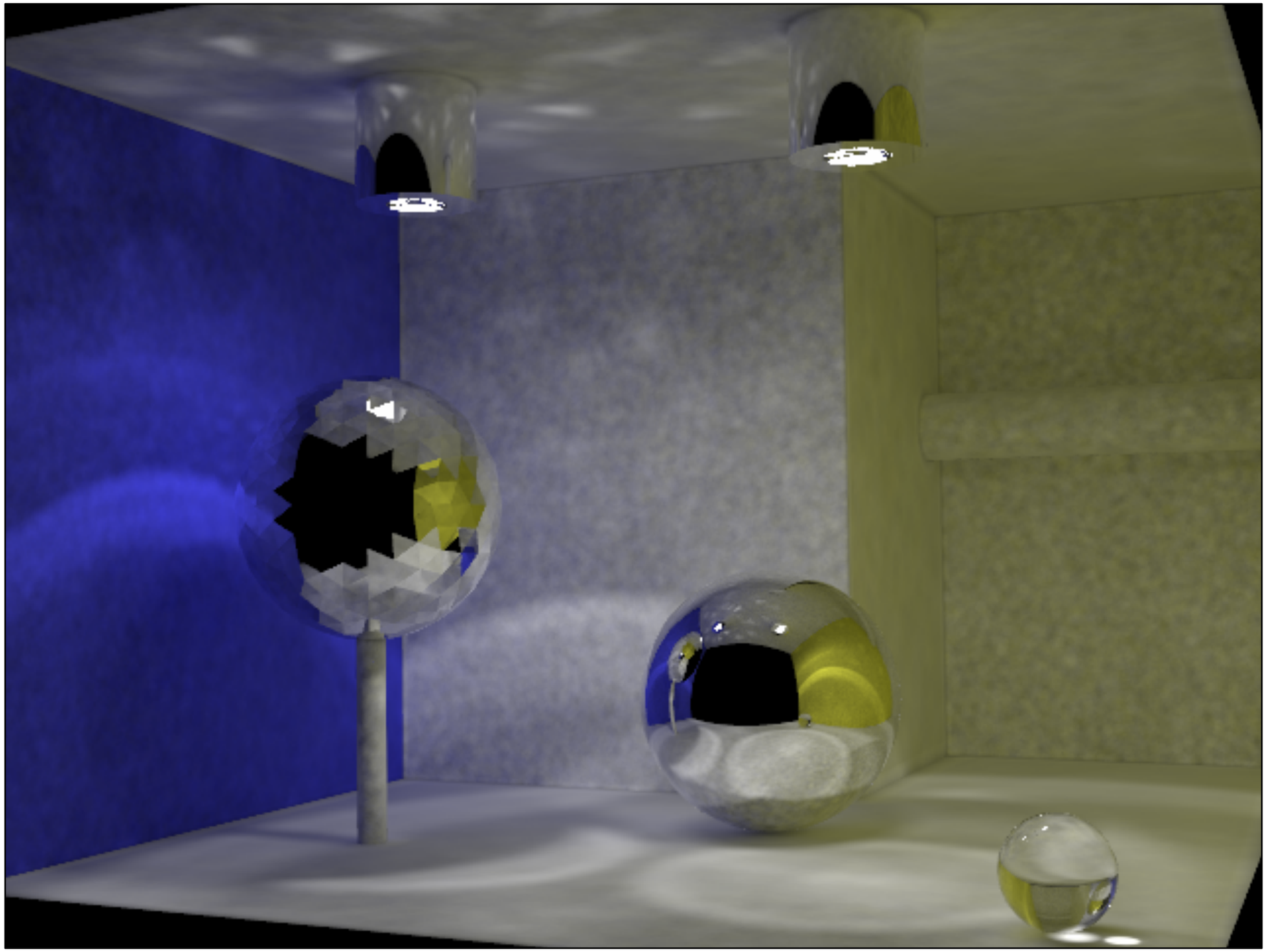


Images courtesy of T. Hachisuka

Robustness of Rendering Methods

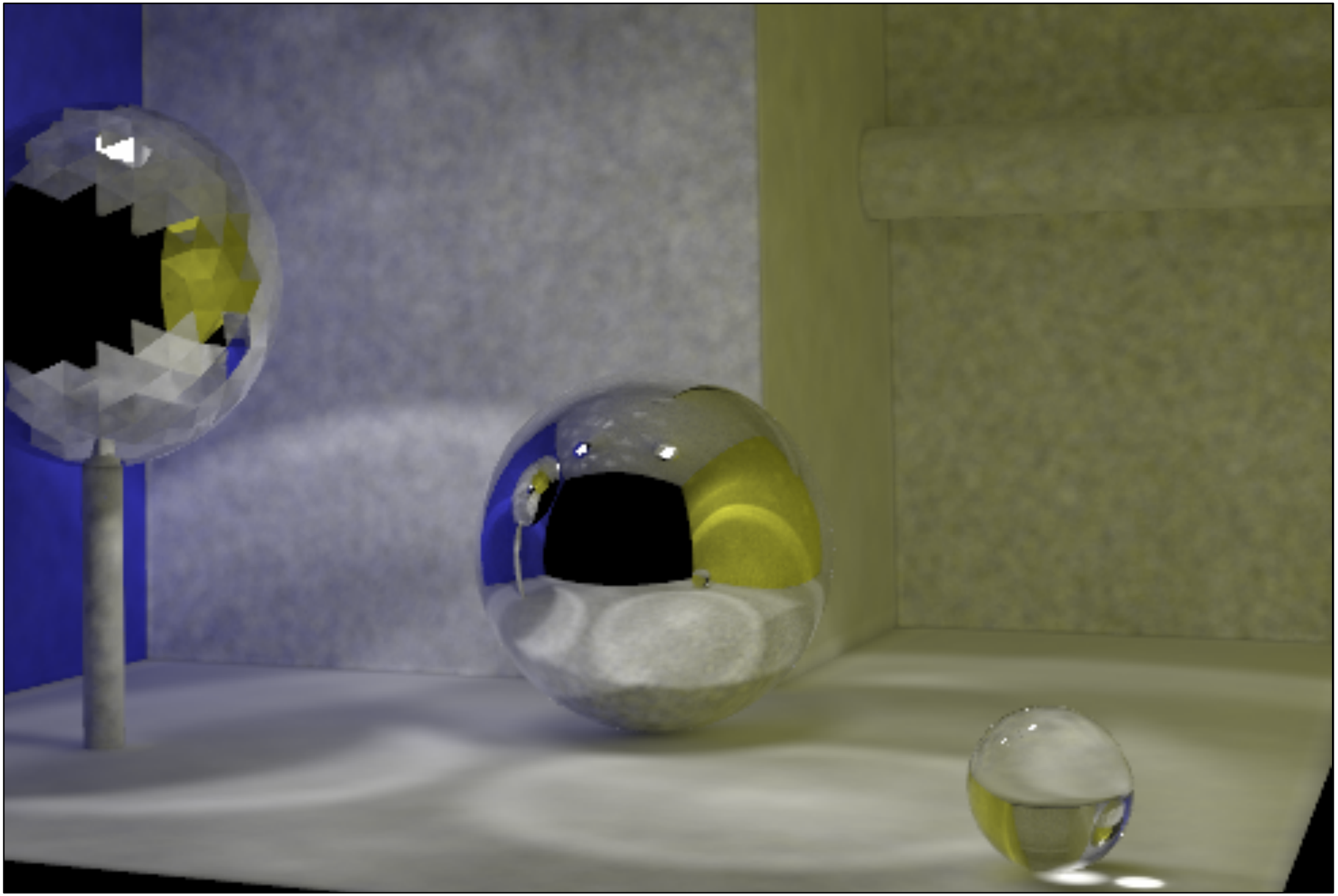
- None of these unbiased methods can handle real light sources well:
  - Path Tracing
  - Bidirectional Path Tracing
  - Metropolis Light Transport
- Photon Mapping?

Photon Mapping



Images courtesy of T. Hachisuka

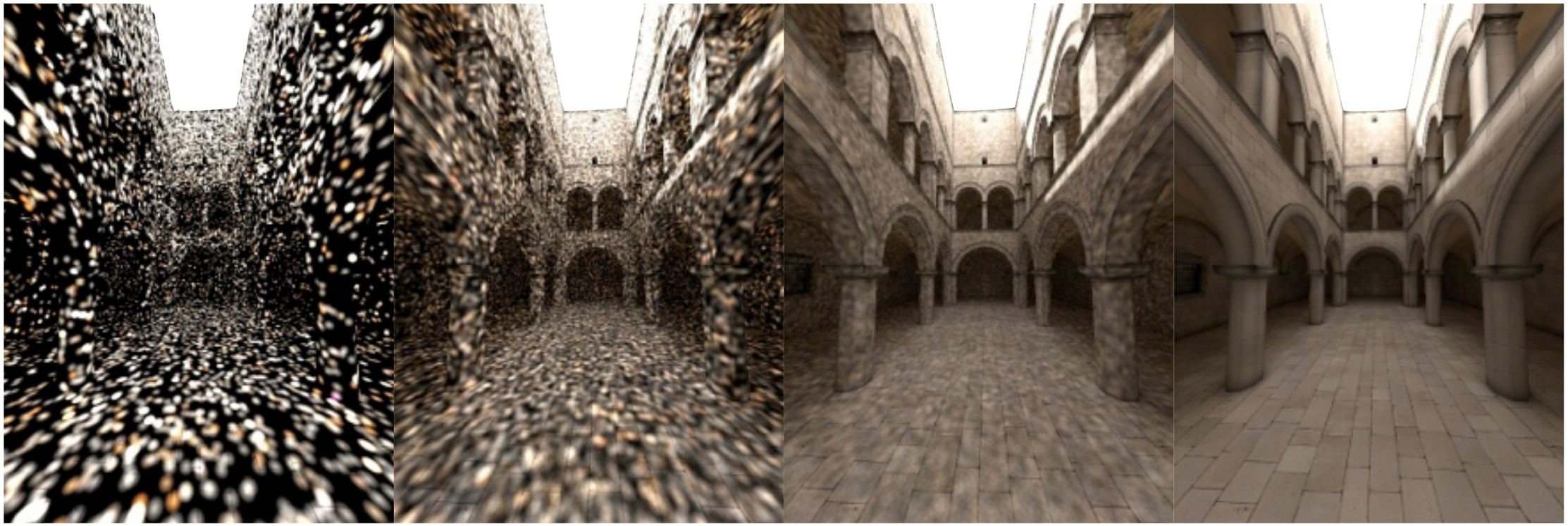
Photon Mapping



Images courtesy of T. Hachisuka

Consistency of Photon Mapping

- Result converges to the correct solution



- Conditions for convergence:
  - Infinitesimally small radius
  - Infinite number of nearby photons
    - **Infinite storage requirement!**

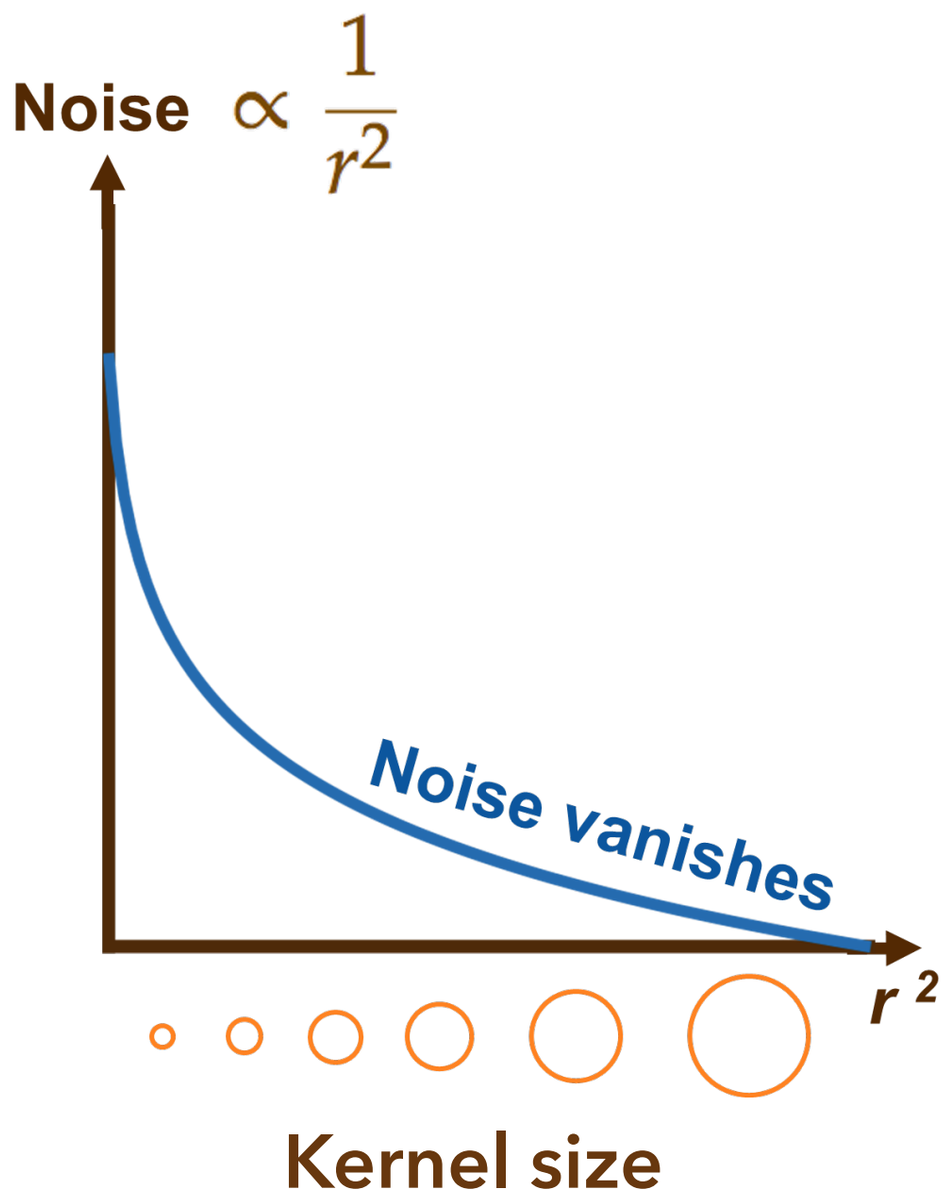
Progressive Photon Mapping

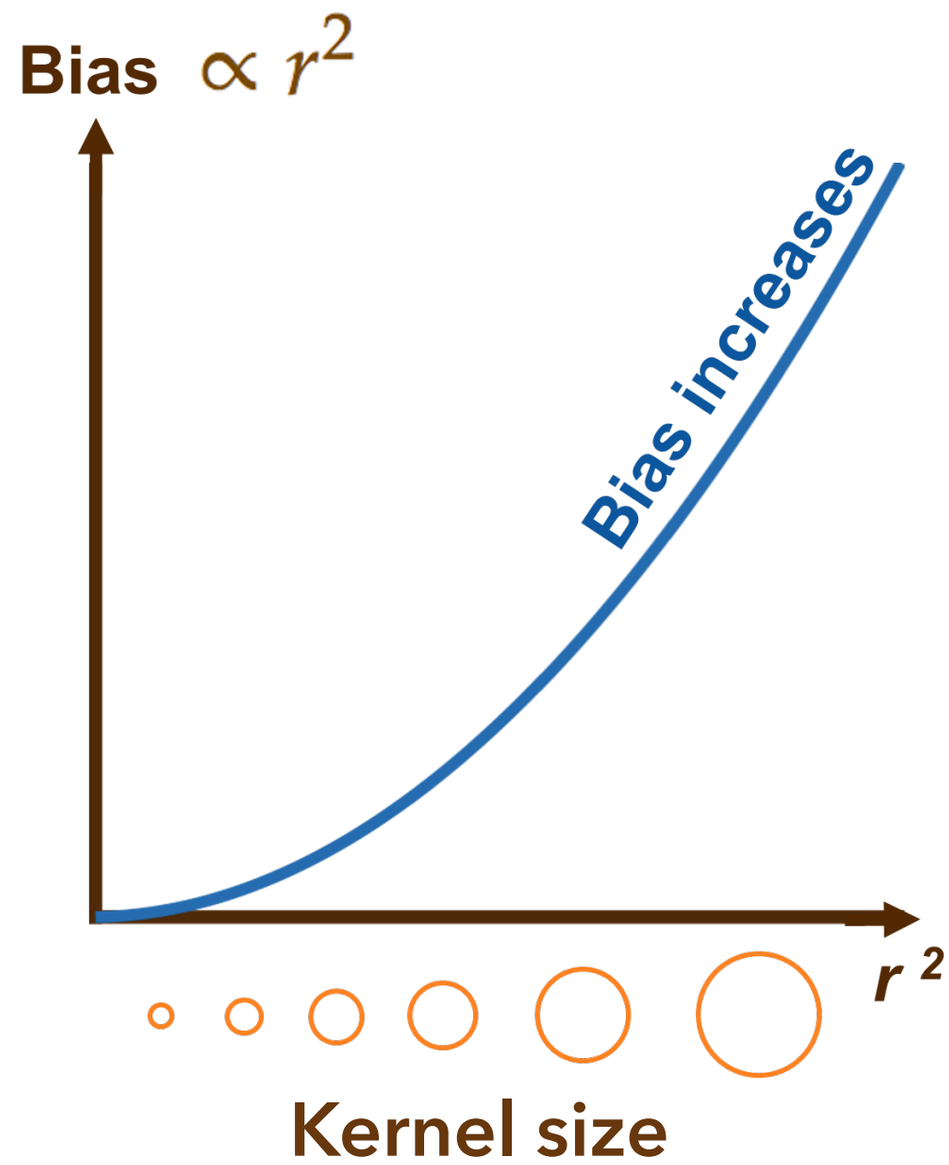
Key Idea

- Progressively shrink the density estimation kernel
- Hachisuka et al. 2008, 2009, ...
  - store/update statistics at each camera ray hitpoint
- Knaus & Zwicker 2011
  - no statistics, just render independent images with smaller and smaller radius, and average

Density Estimation Kernel

- MSE depends on the size of the kernel





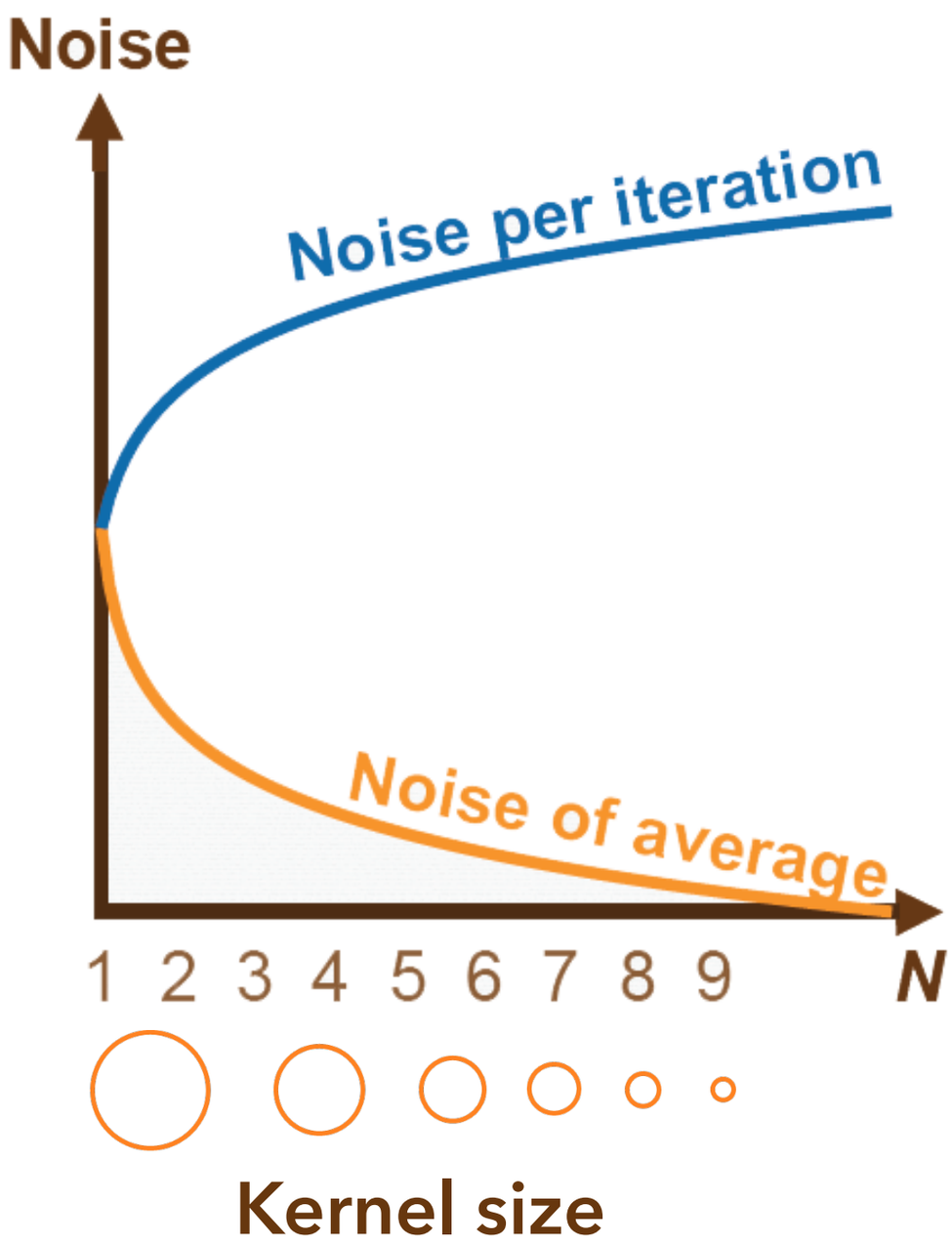
Radius Reduction

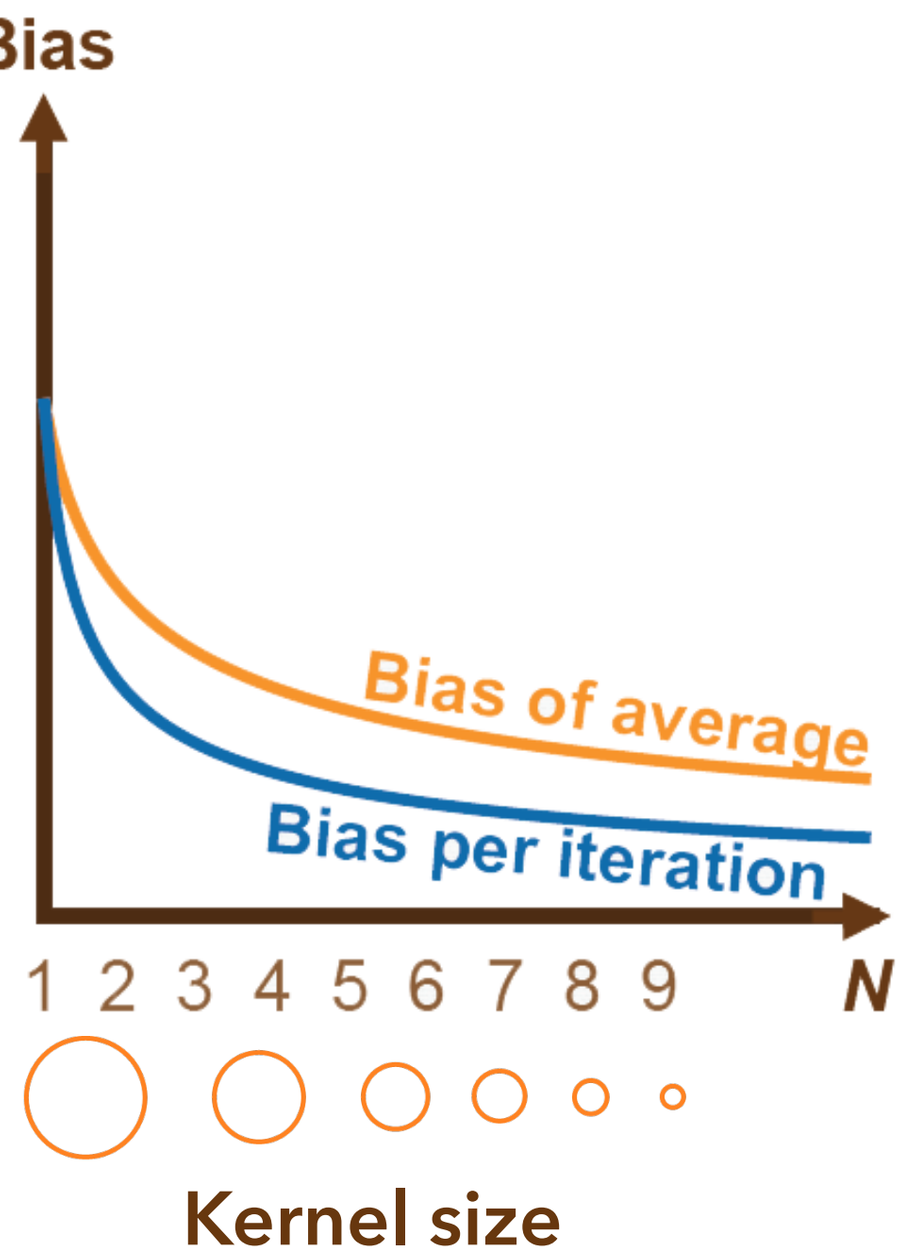
- Given:
  - Iteration i
  - Kernel radius r_i
  - Parameter $\alpha \in (0, 1)$ for controlling the shrinking
- The radius for the next iteration is:

$$r_{i+1}^2 = \frac{i + \alpha}{i + 1} r_i^2$$

See [Knaus & Zwicker 2011] for derivation

Convergence Analysis





Images courtesy of C. Knaus and M. Zwicker

Convergence Analysis

- What value for alpha?
  - Low value = fast shrinking
  - High value = slow shrinking
  - Convergence achieved for $\alpha \in (0, 1)$, but
- $\alpha = 2/3$ minimizes the MSE

Algorithm

- Step 1:
  - Photon tracing: emit, scatter, store photons
- Step 2:
  - Trace camera paths
  - Evaluate radiance estimate using radius r_i
- Display running average
- Compute new radius $r_{i+1}^2 = \frac{i+\alpha}{i+1} r_i^2$ and repeat...

Trivially parallelizable by iteration

Progressive Radius Reduction

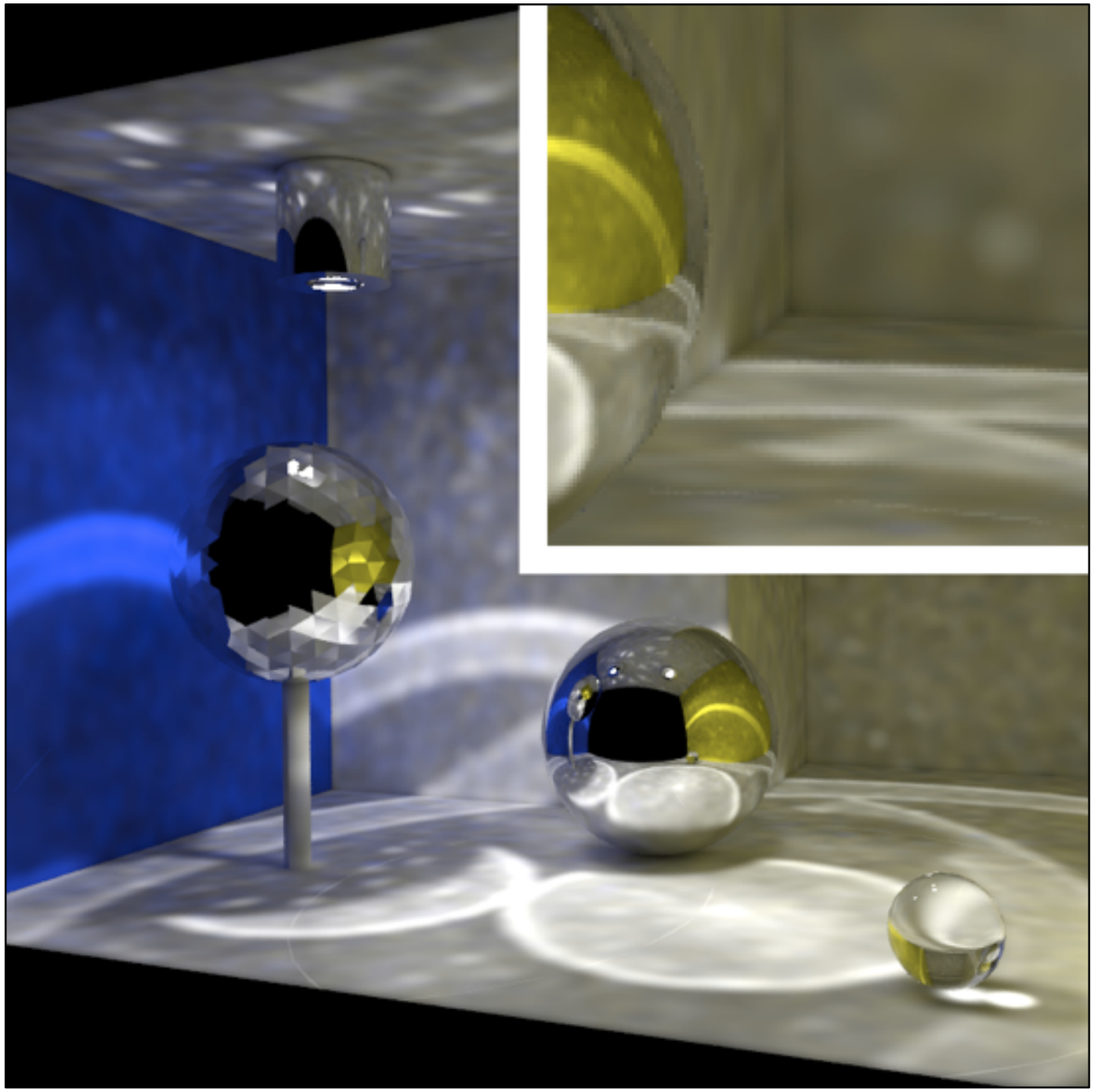


Image 1, $r = 20$

Progressive Radius Reduction

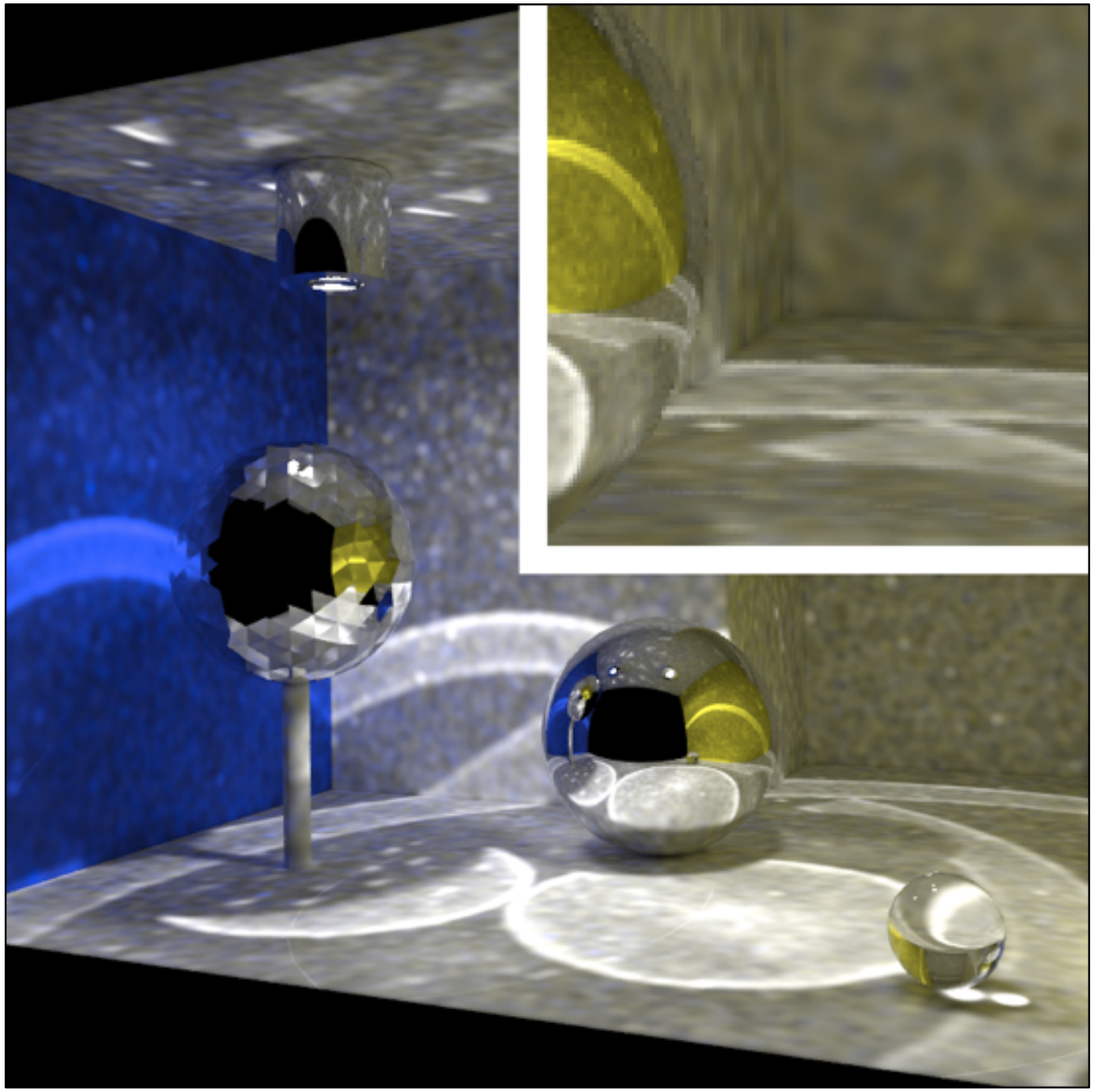


Image 10, $r = 11.87$

Progressive Radius Reduction

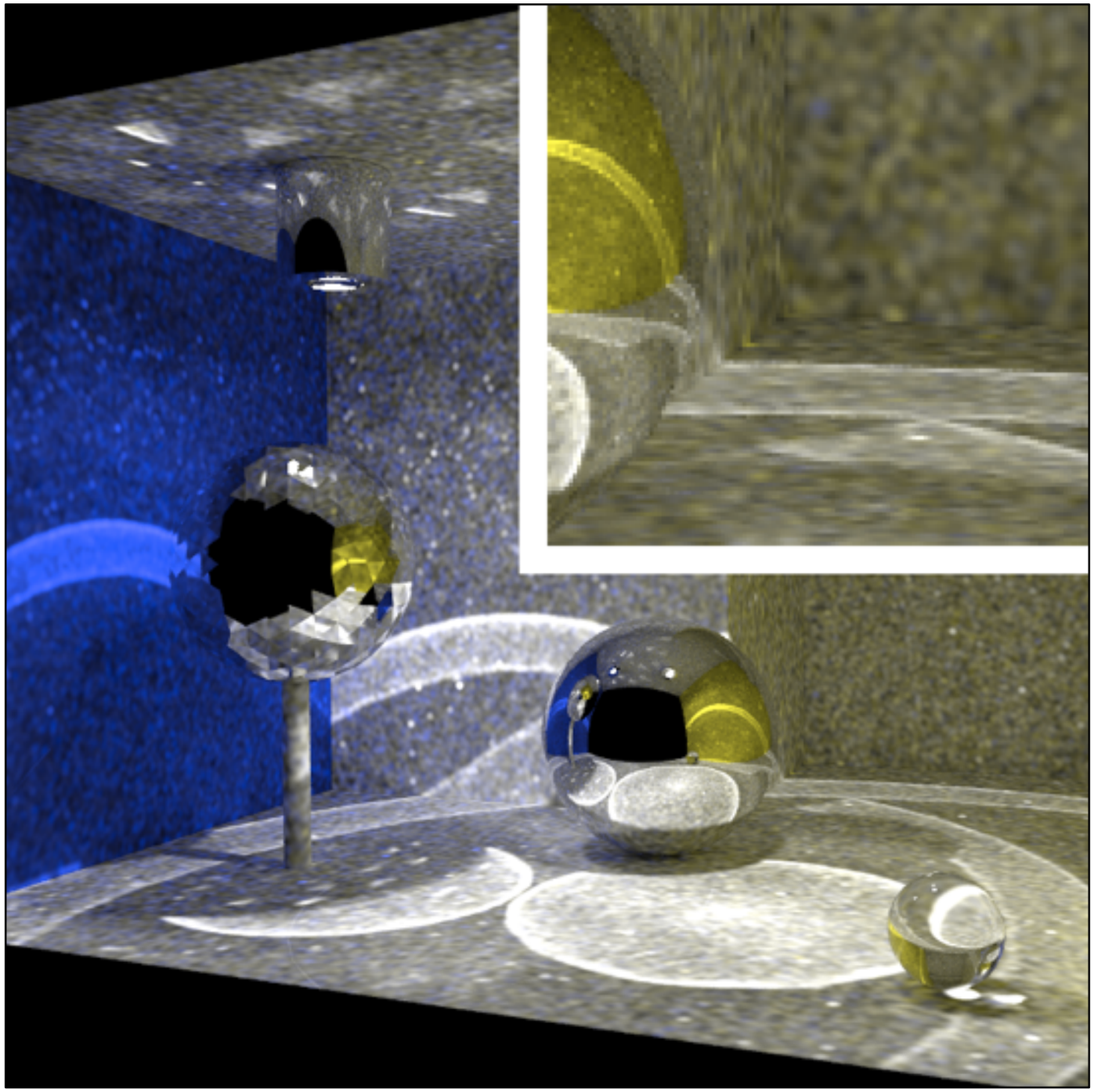


Image 100, $r = 6.71$

Progressive Radius Reduction

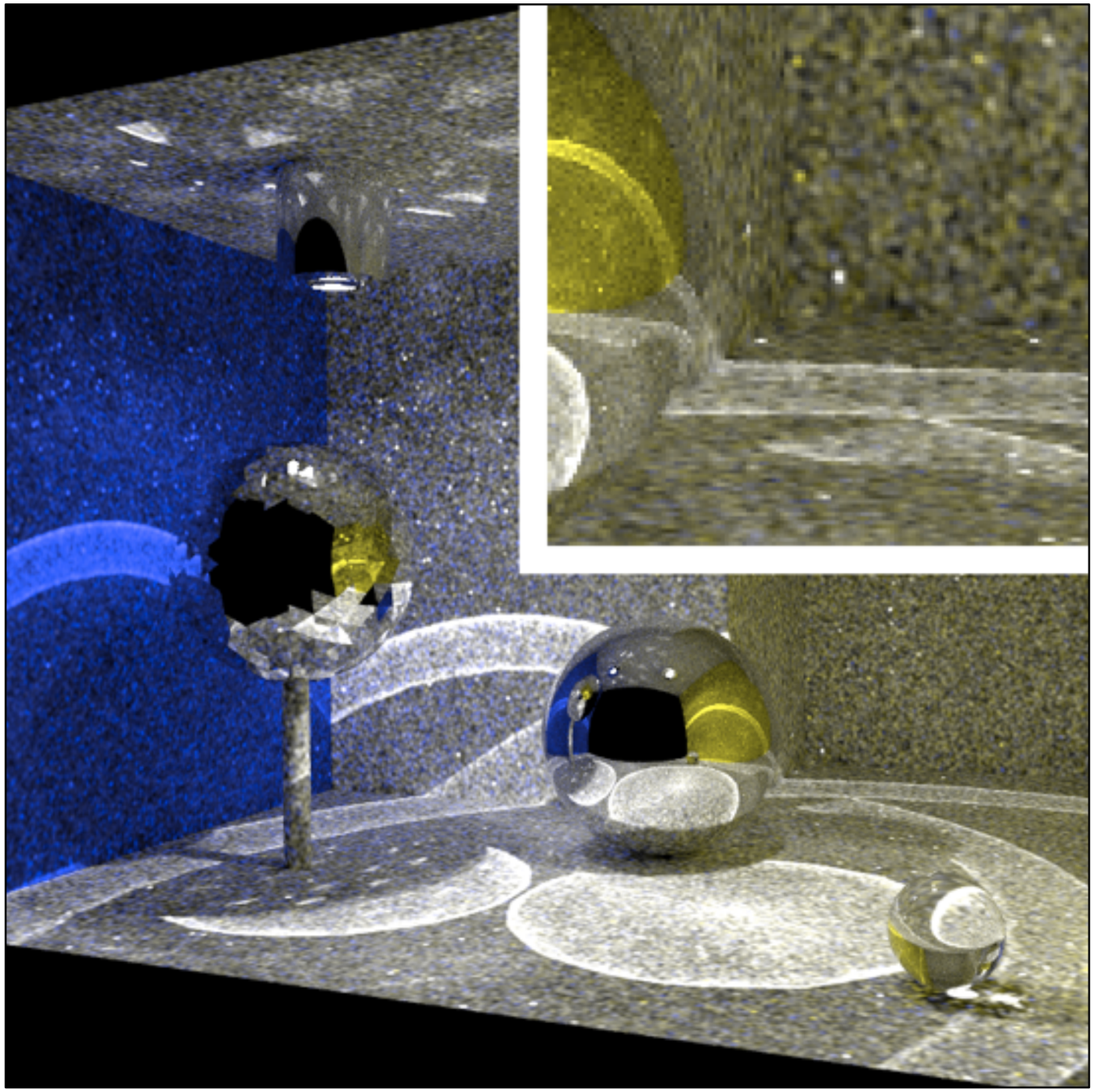


Image 1000, $r = 3.78$

Running Average

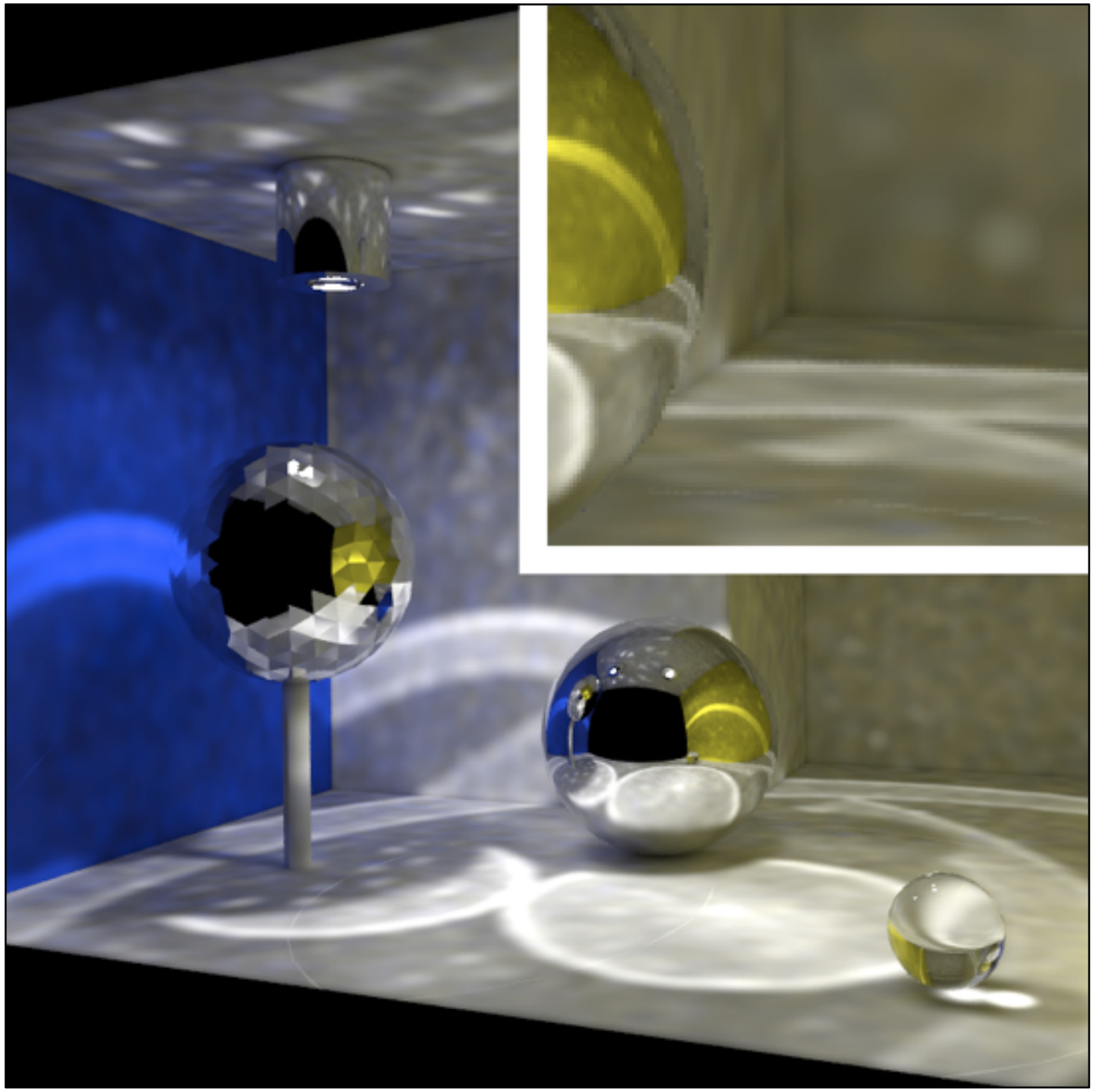
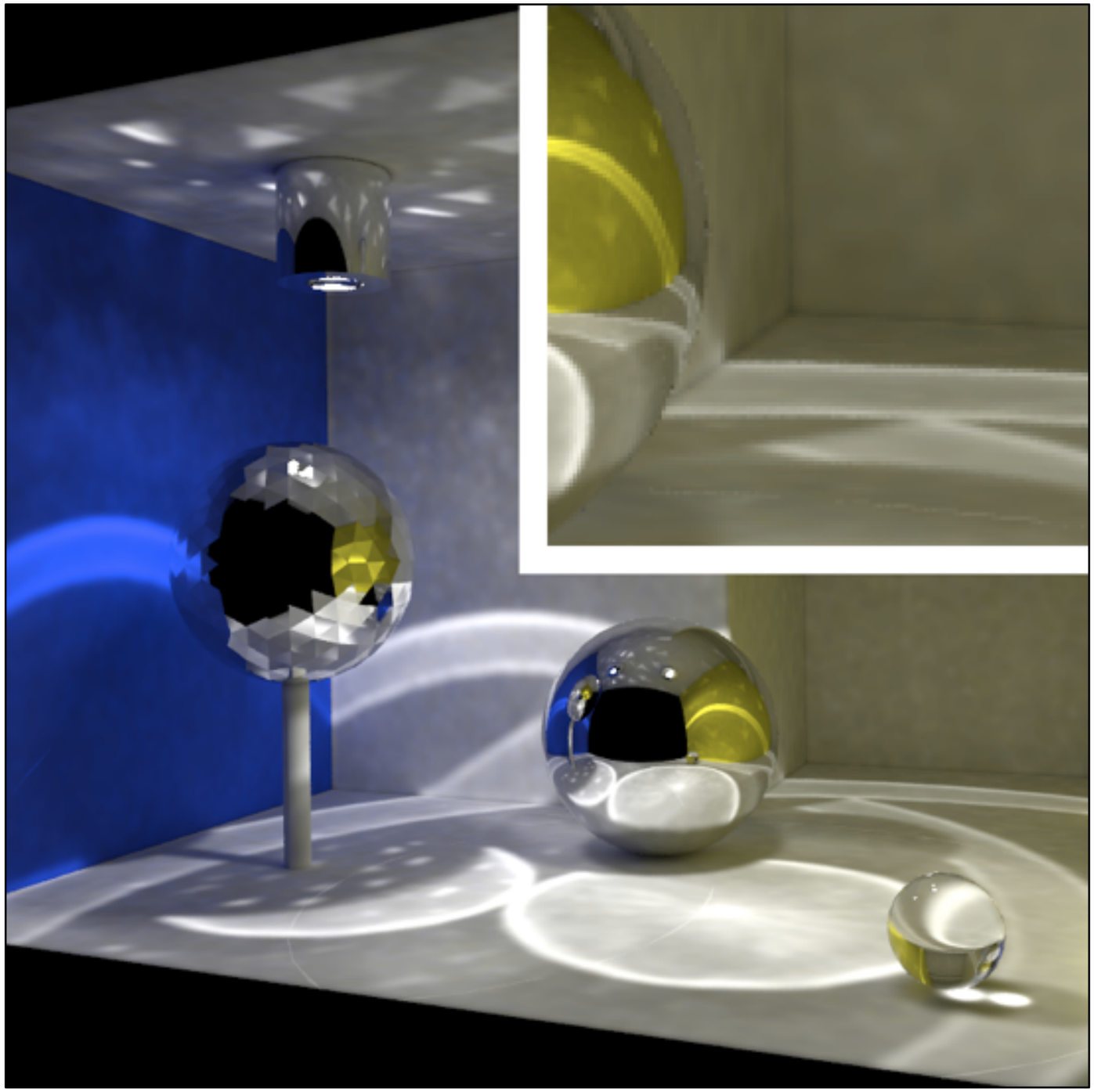


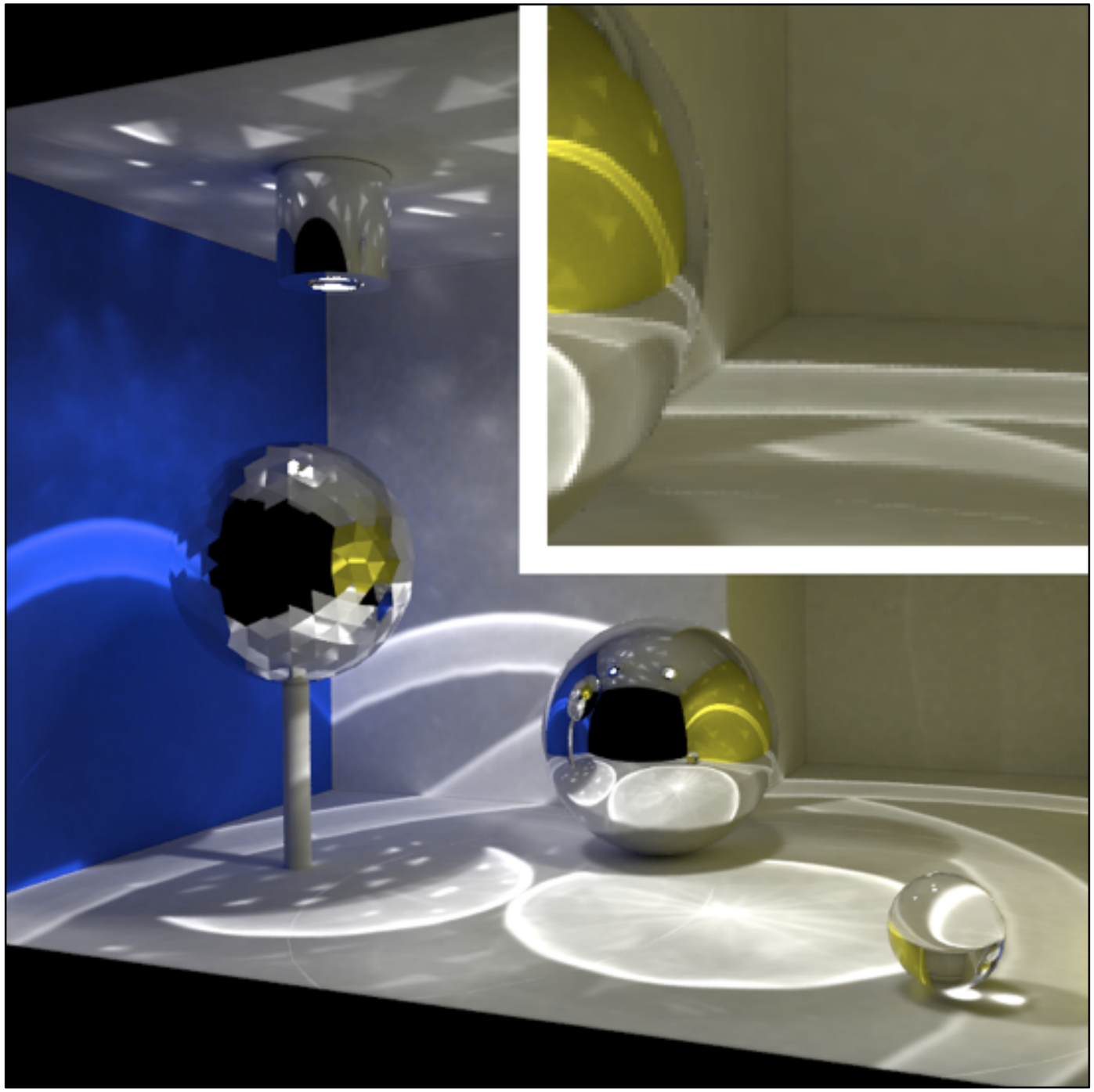
Image 1

Running Average



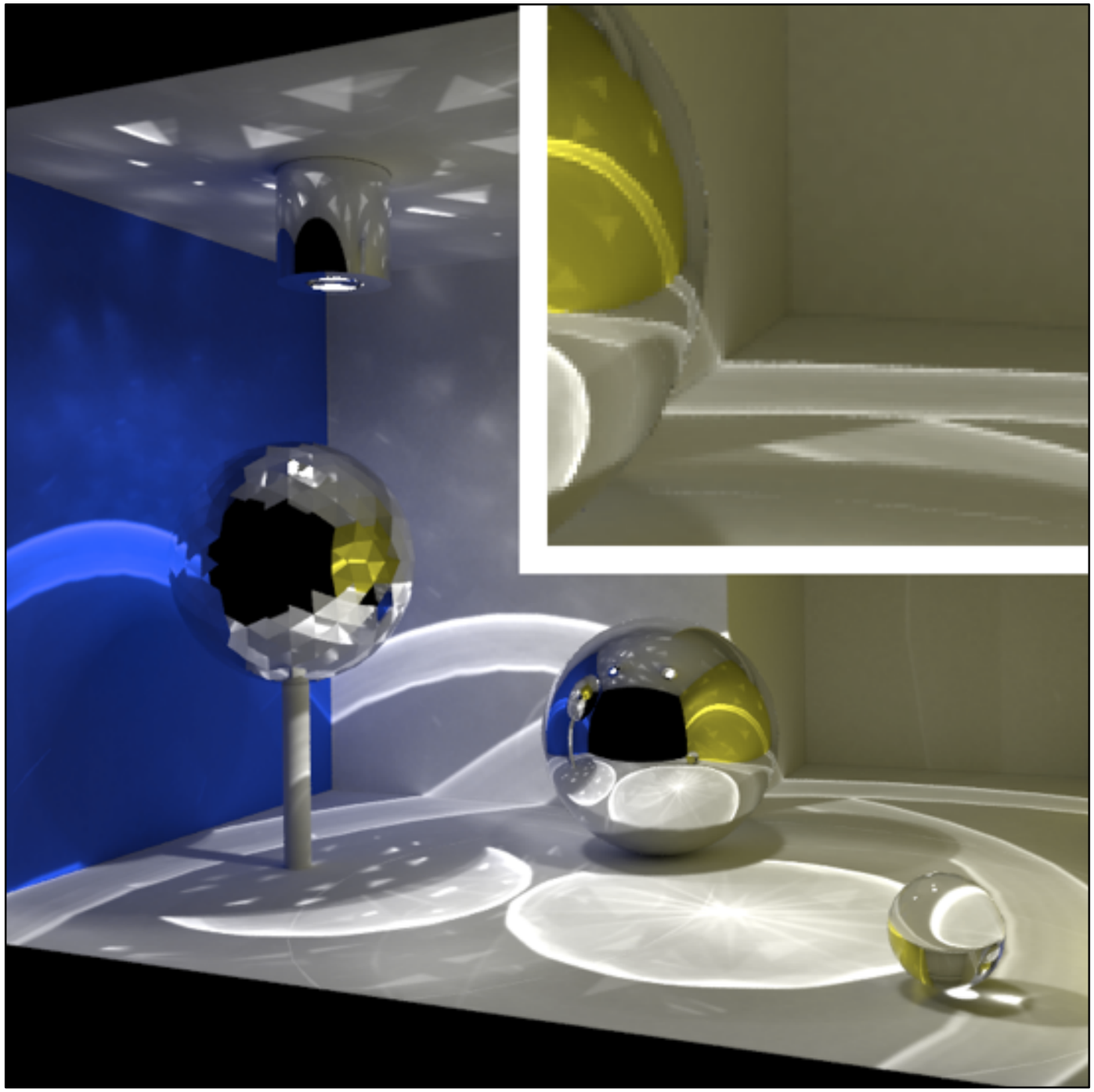
Average of Images 1-10

Running Average



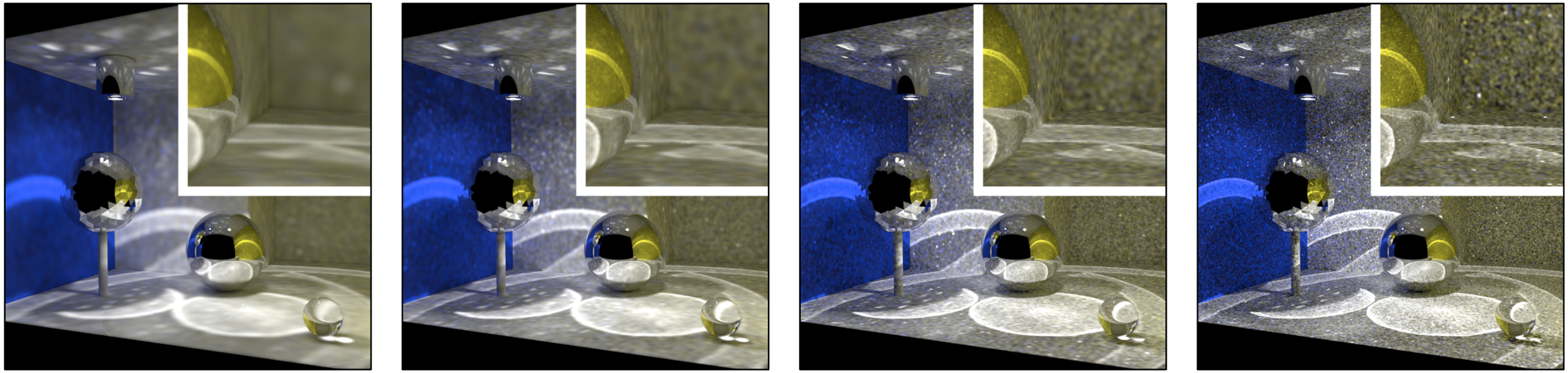
Average of Images 1-100

Running Average

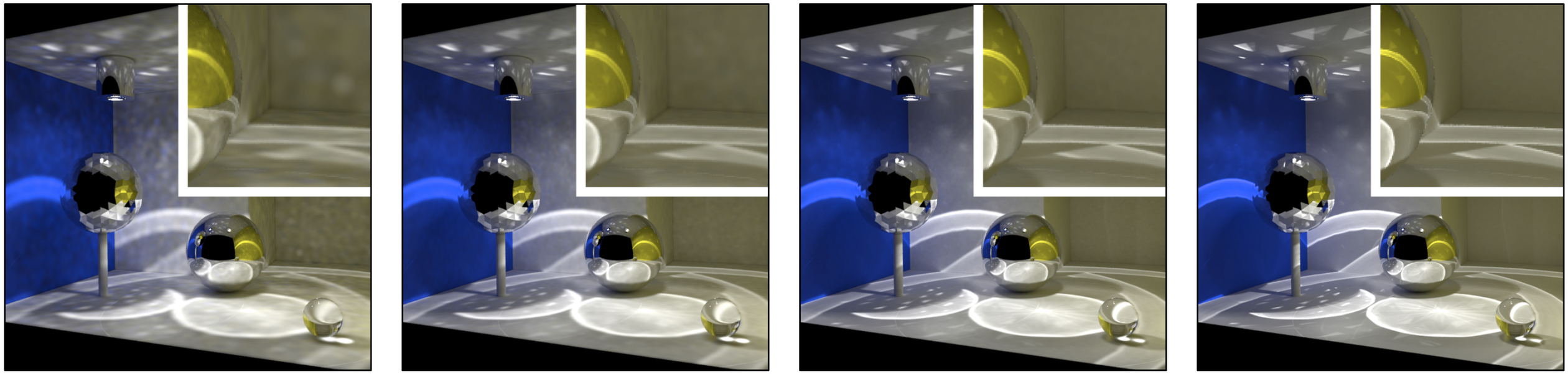


Average of Images 1-1000

Results



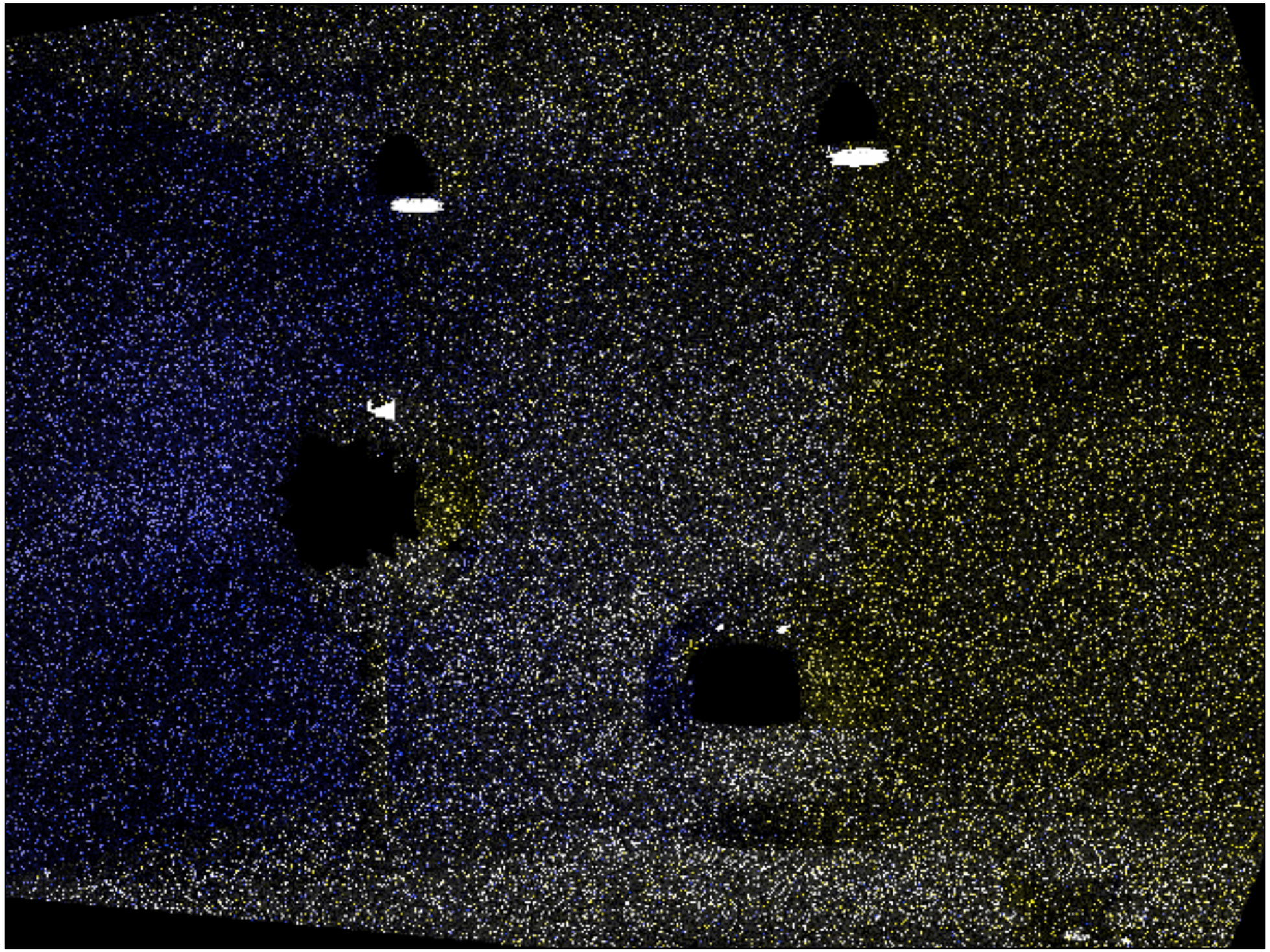
Individual iterations



Running average

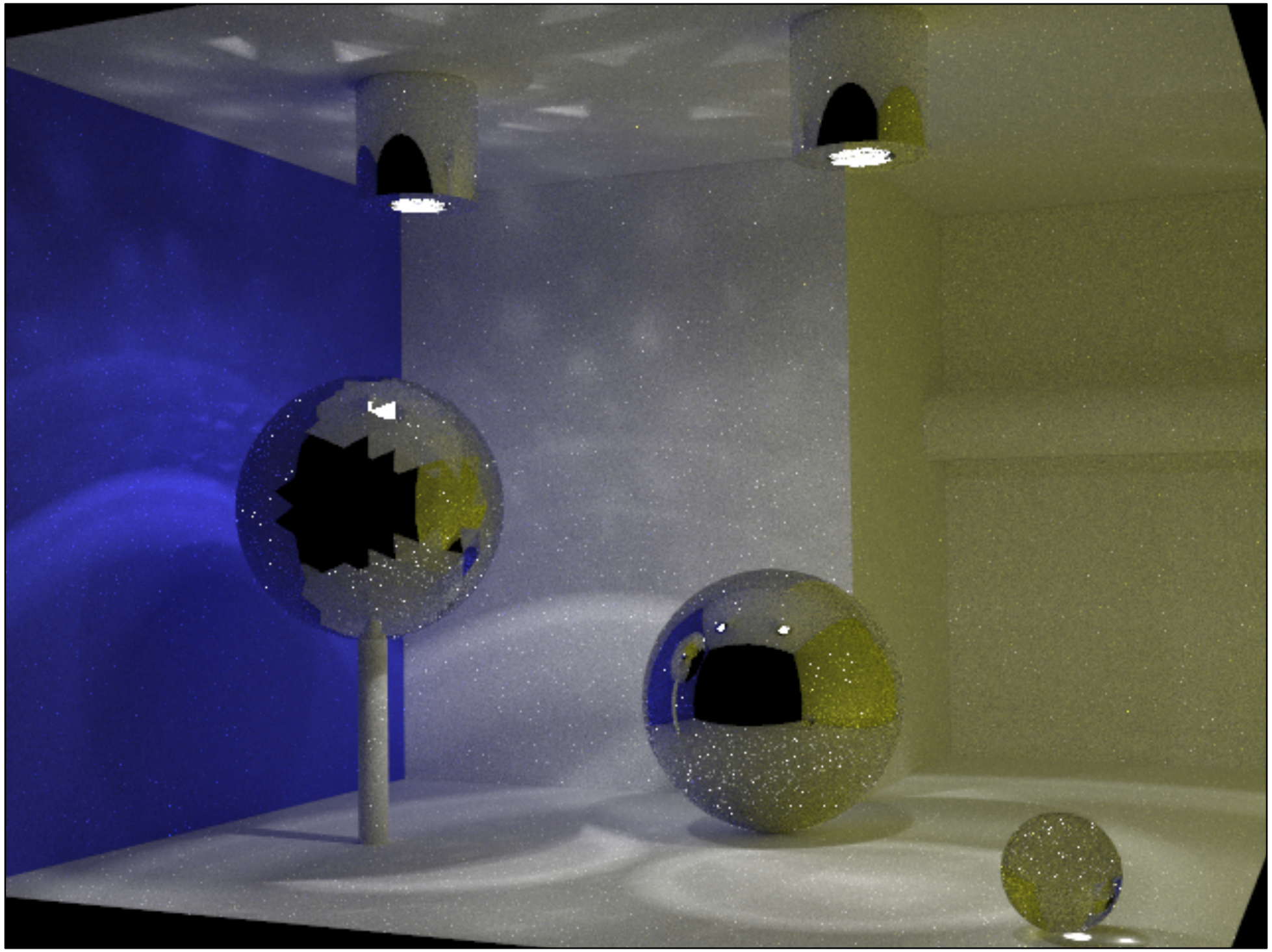
Images courtesy of C. Knaus and M. Zwicker

Path Tracing



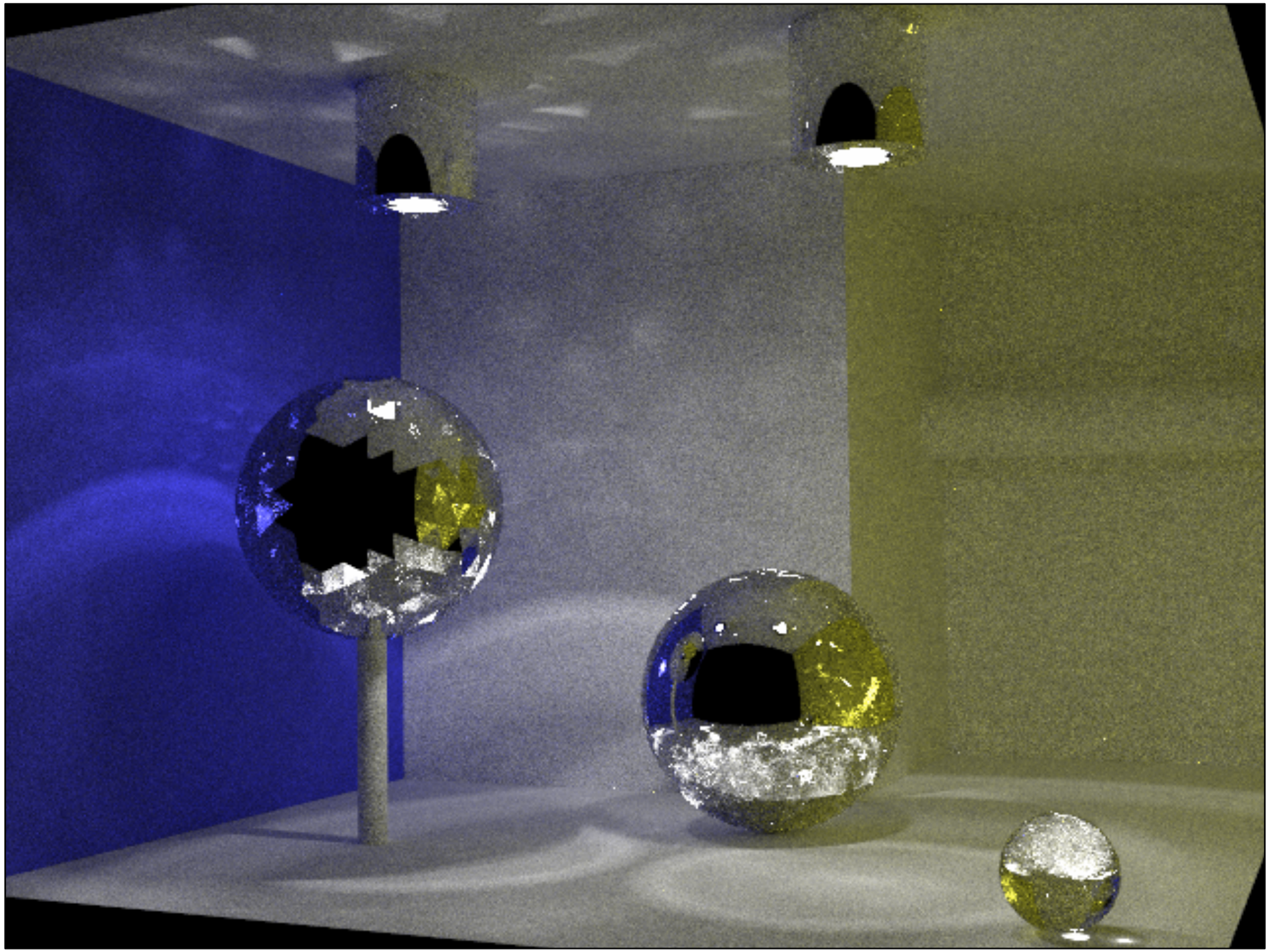
Images courtesy of C. Knaus and M. Zwicker

Bidirectional Path Tracing



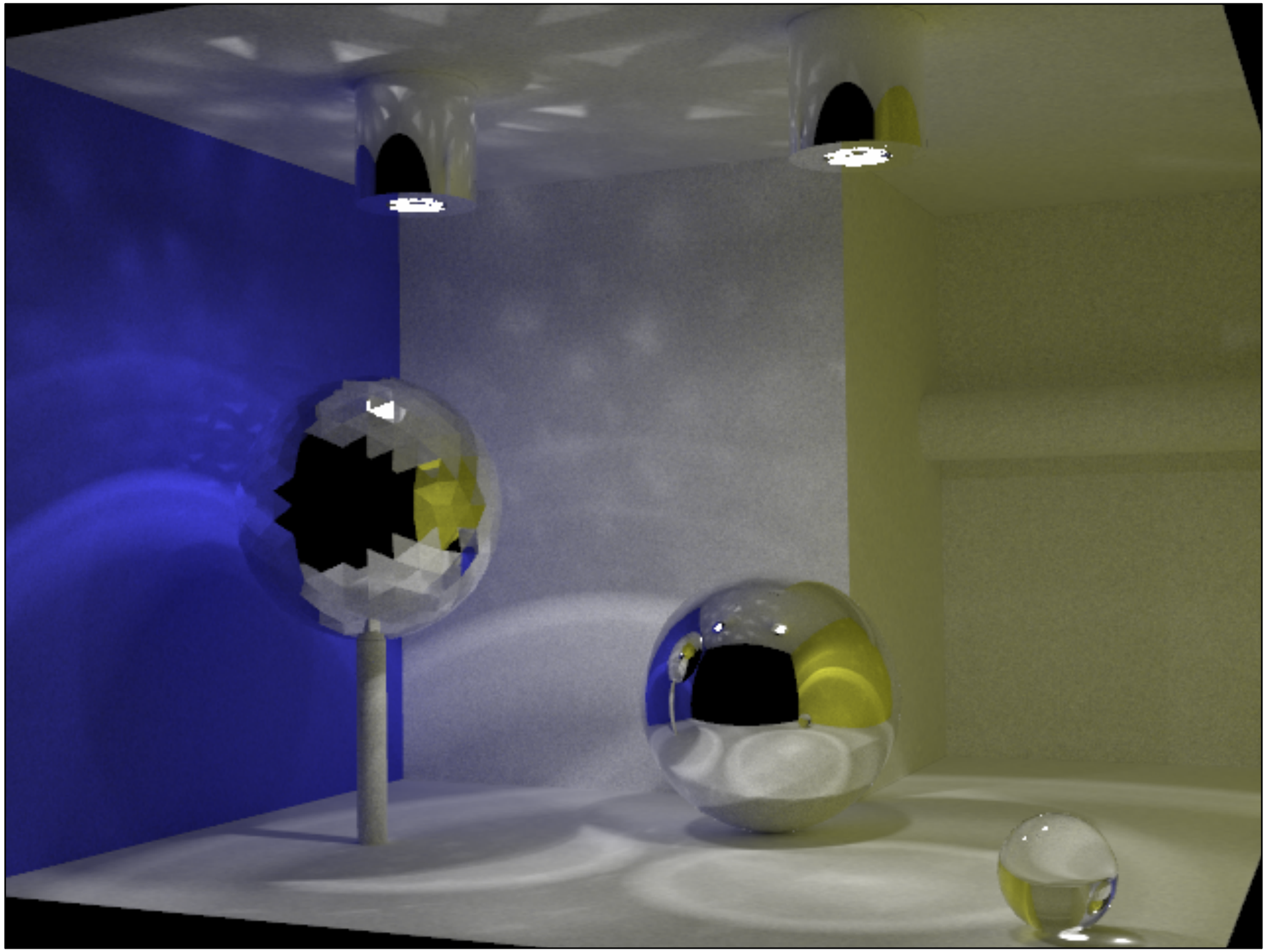
Images courtesy of T. Hachisuka

Metropolis Light Transport



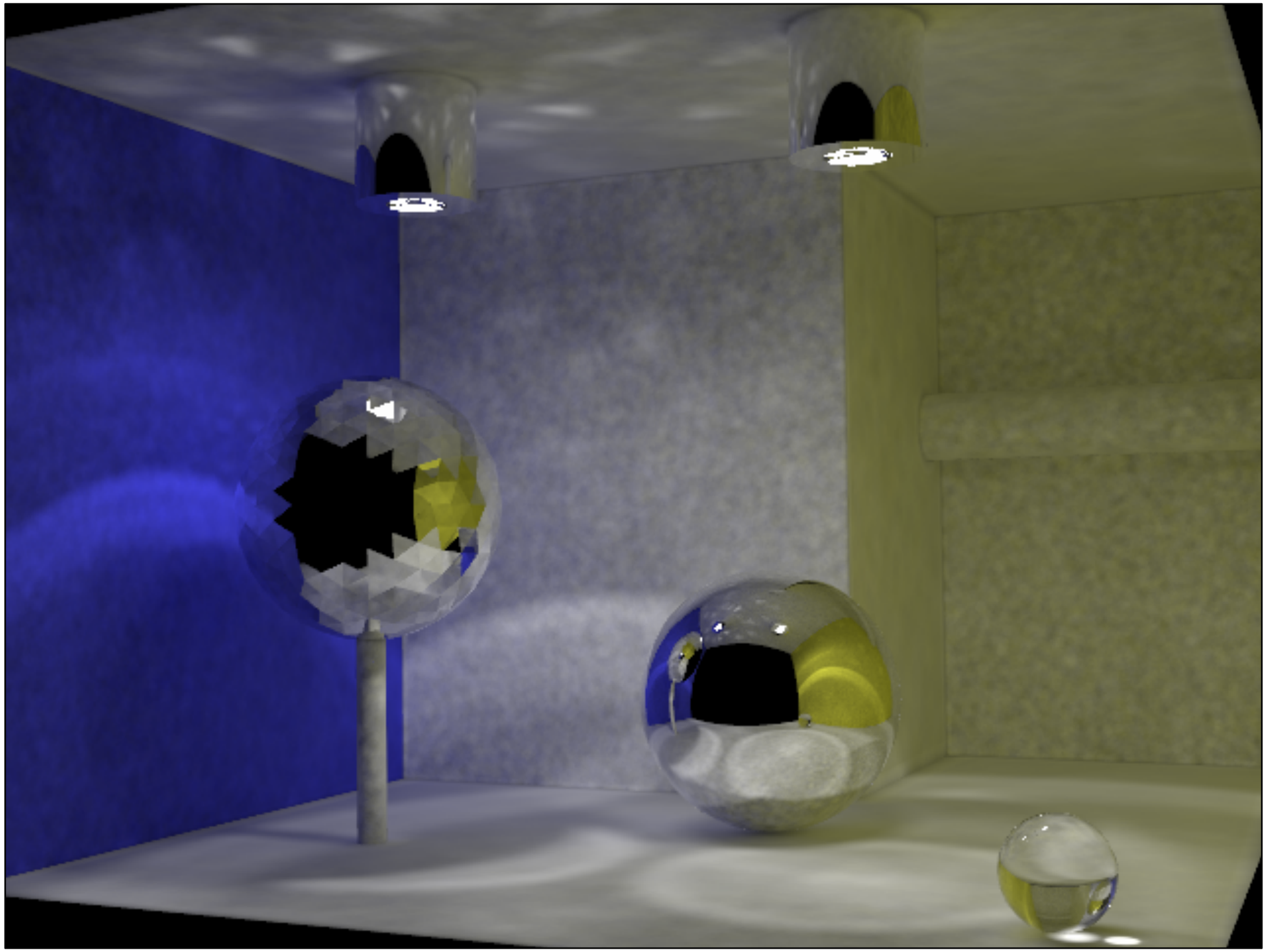
Images courtesy of T. Hachisuka

Progressive Photon Mapping



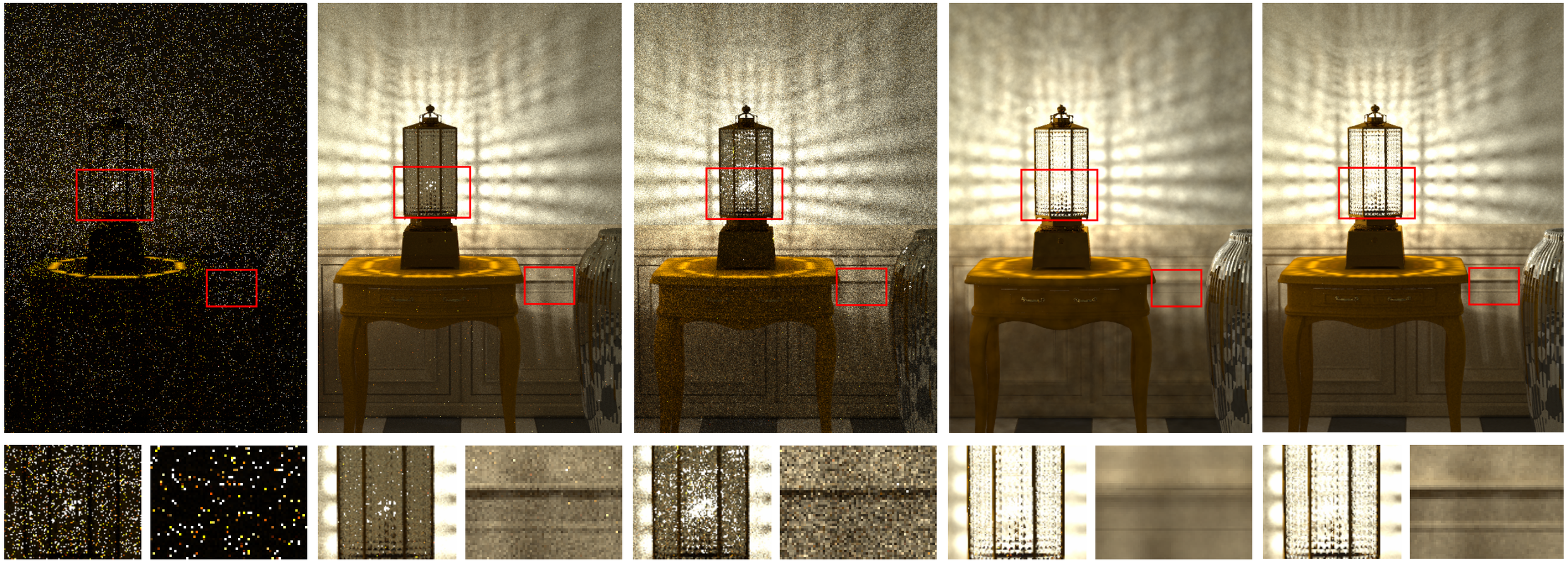
Images courtesy of T. Hachisuka

Photon Mapping

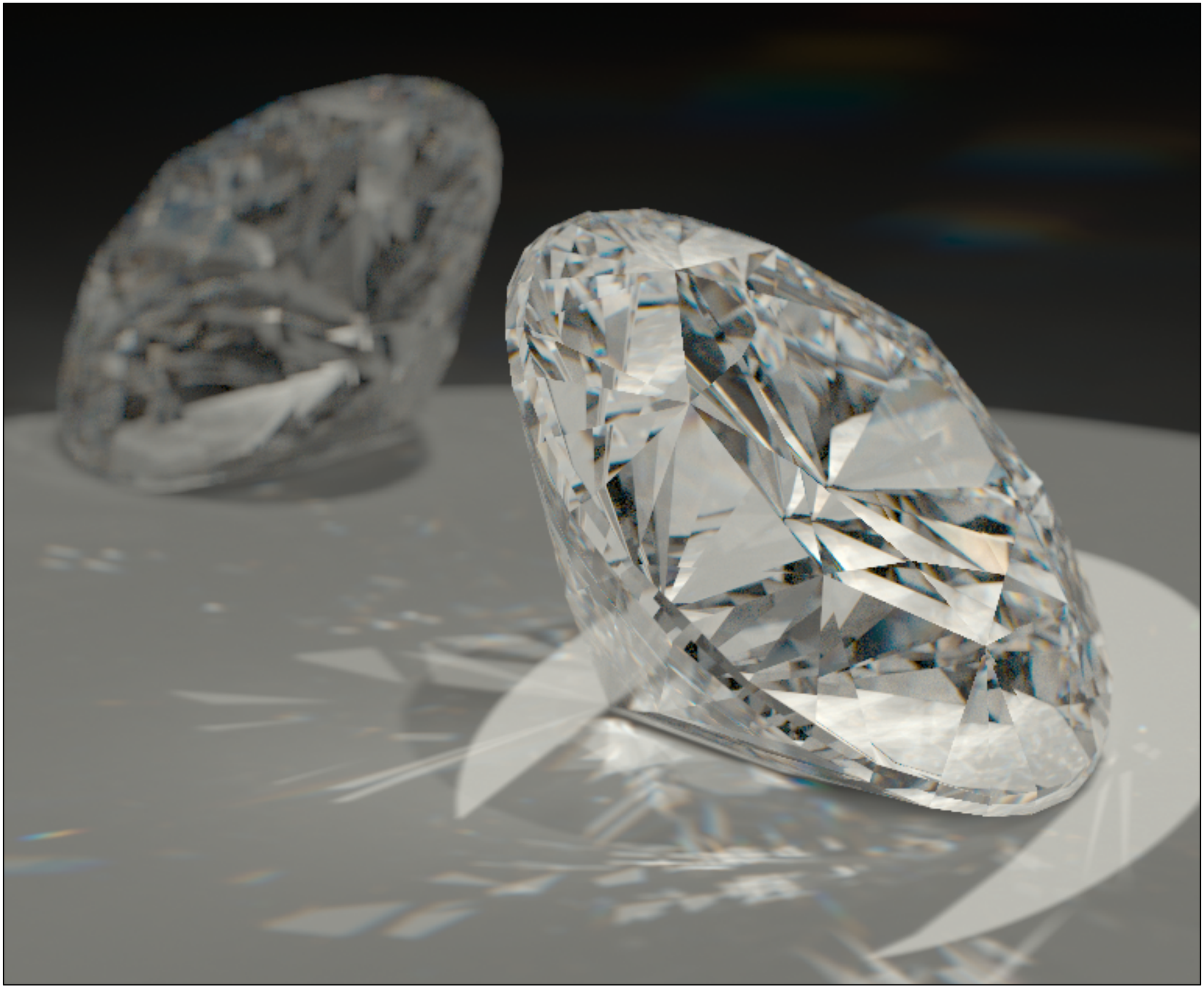


Images courtesy of T. Hachisuka

Glass Lantern

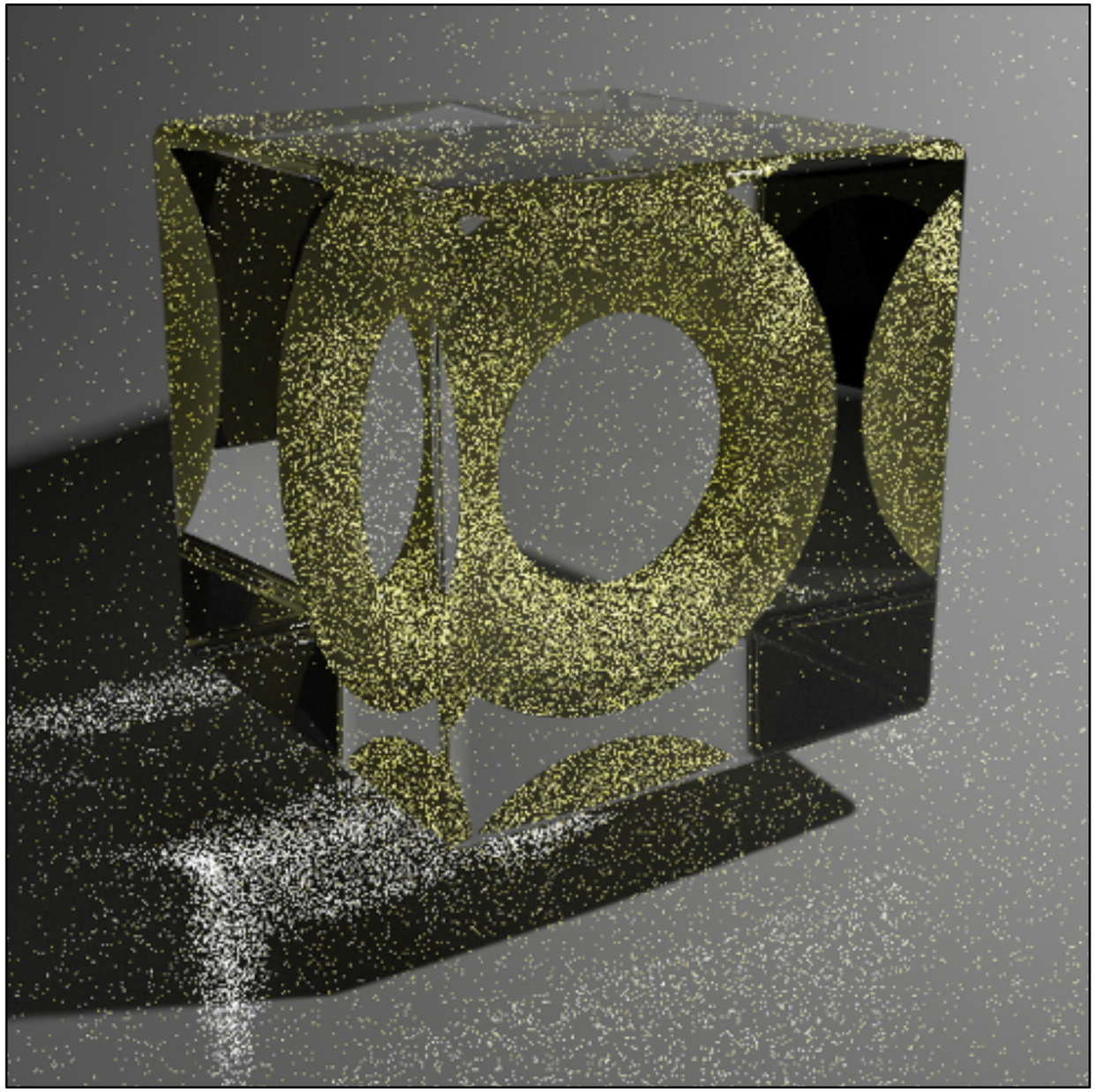


Path tracing | Bidirectional path tracing | Metropolis light transport | Photon mapping | Progressive photon mapping

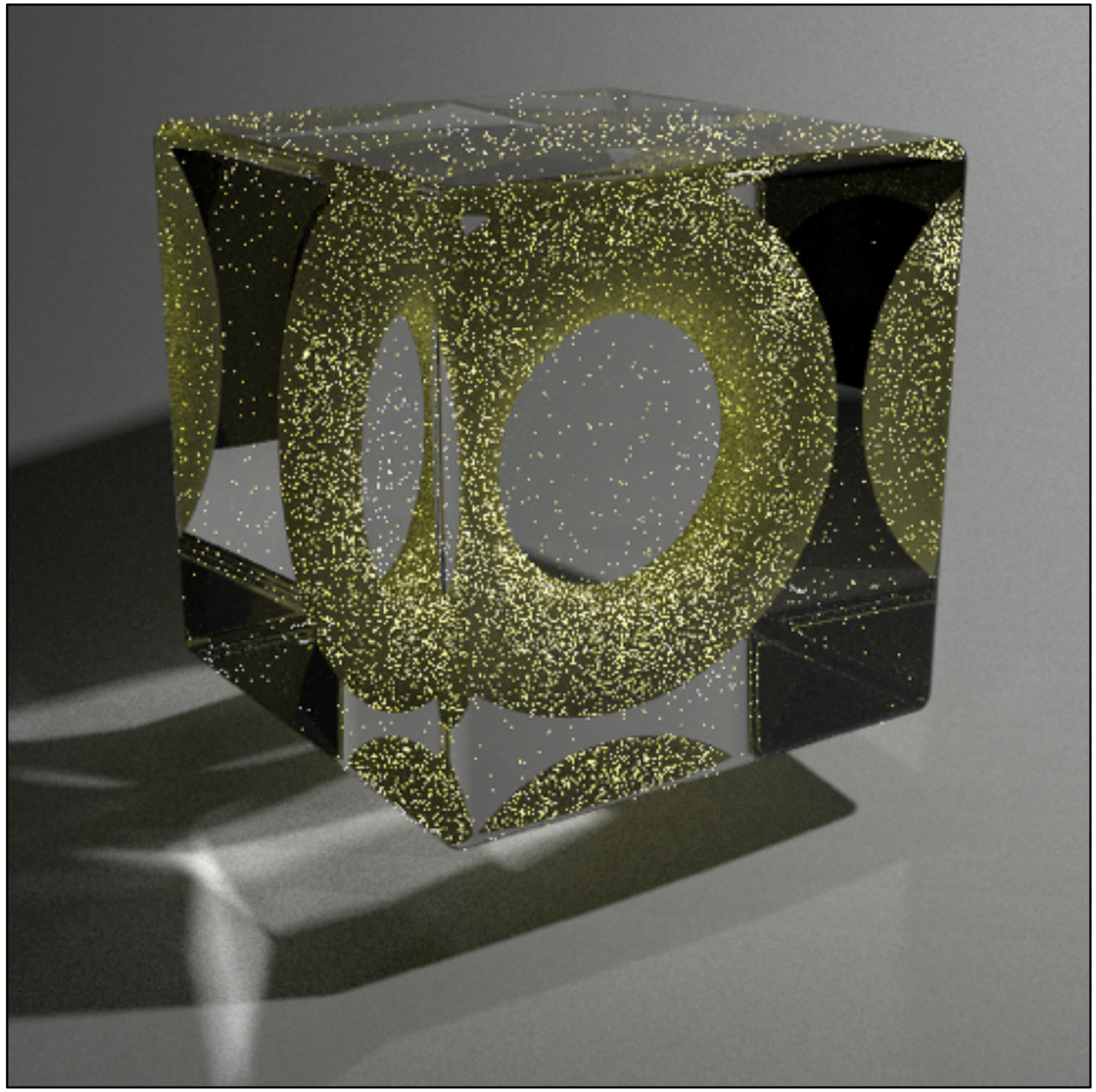


Images courtesy of C. Knaus and M. Zwicker

Torus in Cube (LS+D*+E)

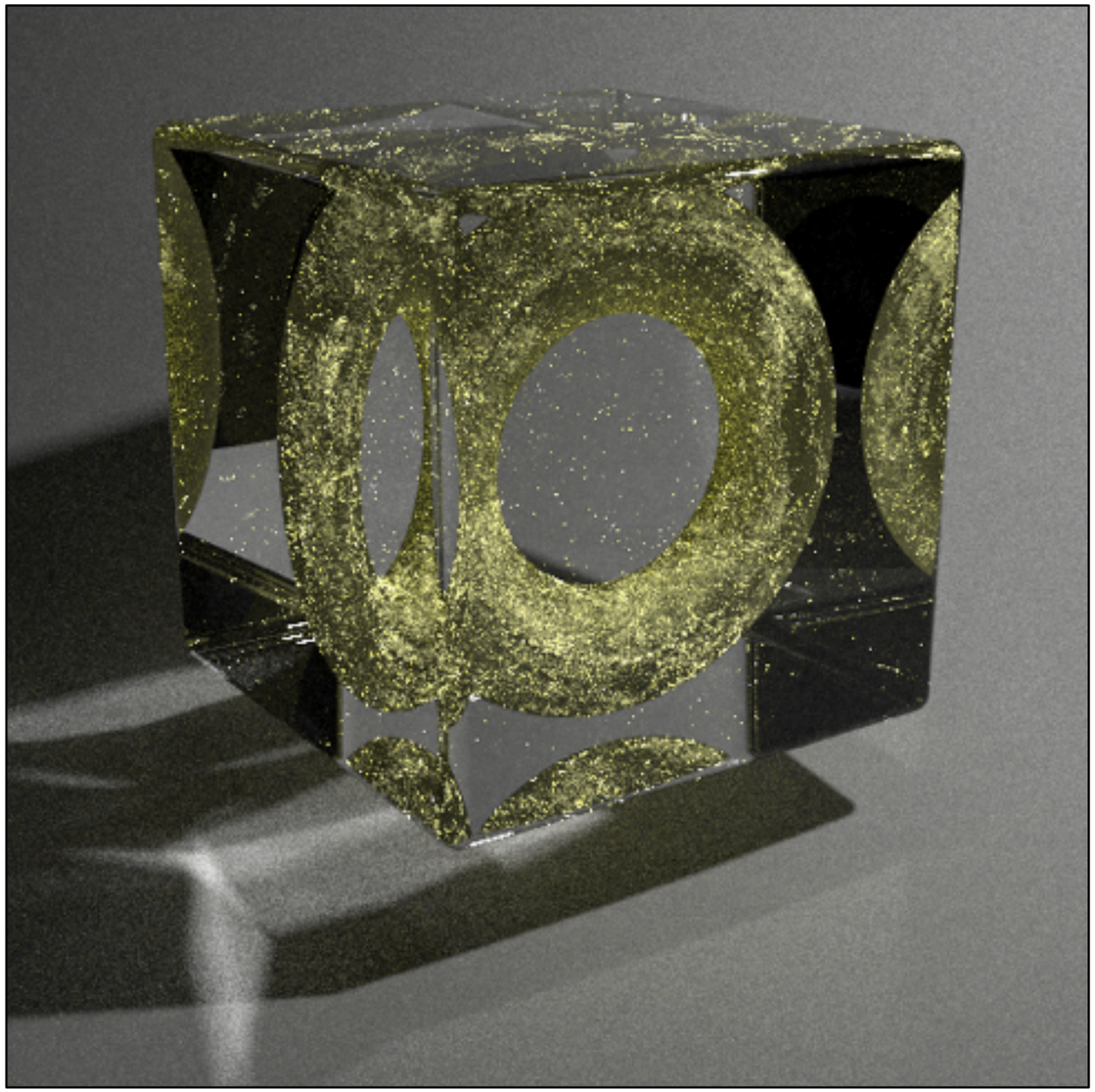


Path Tracing

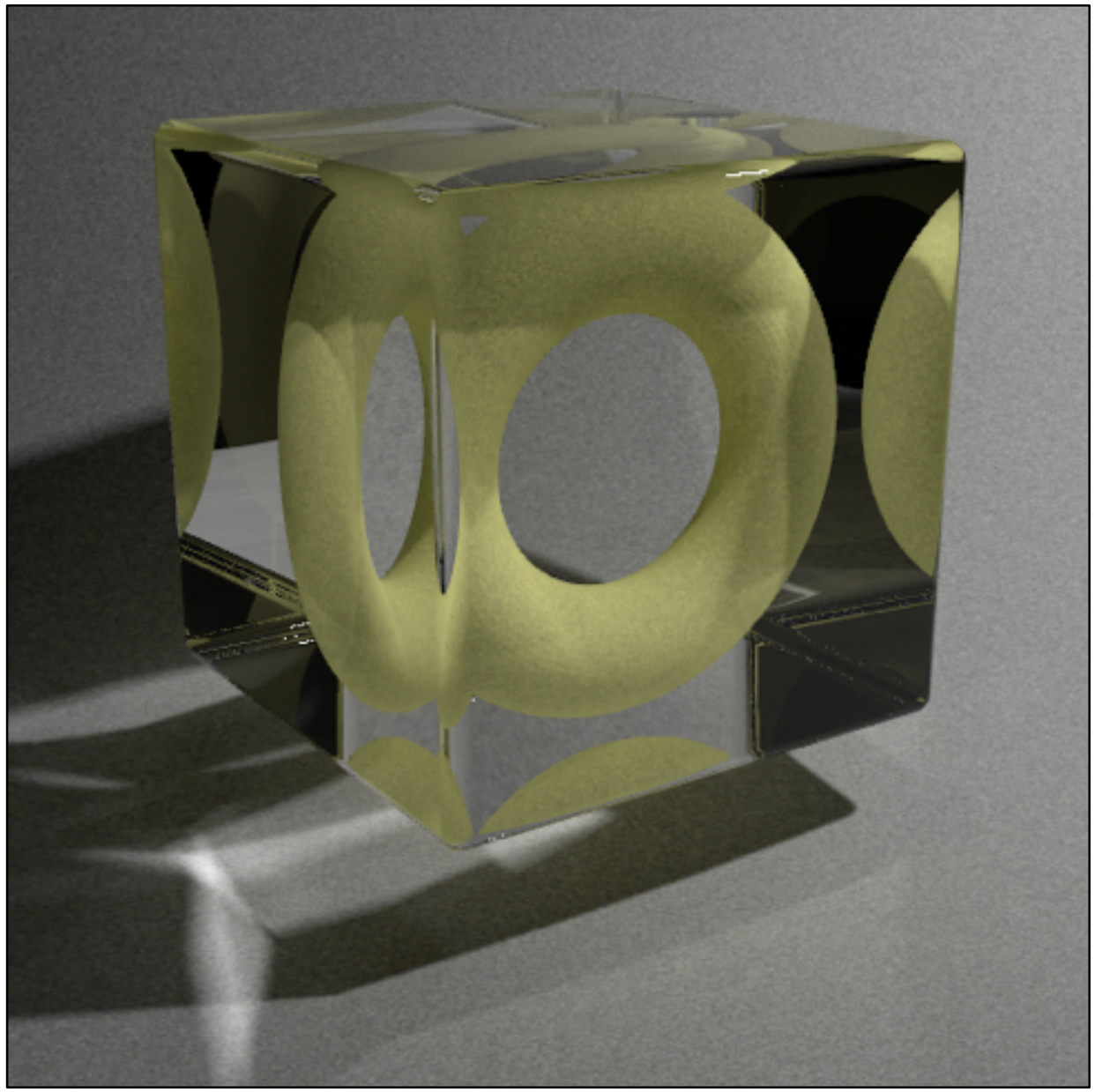


Bidirectional Path Tracing

Torus in Cube (LS+D*+E)



Metropolis Light Transport

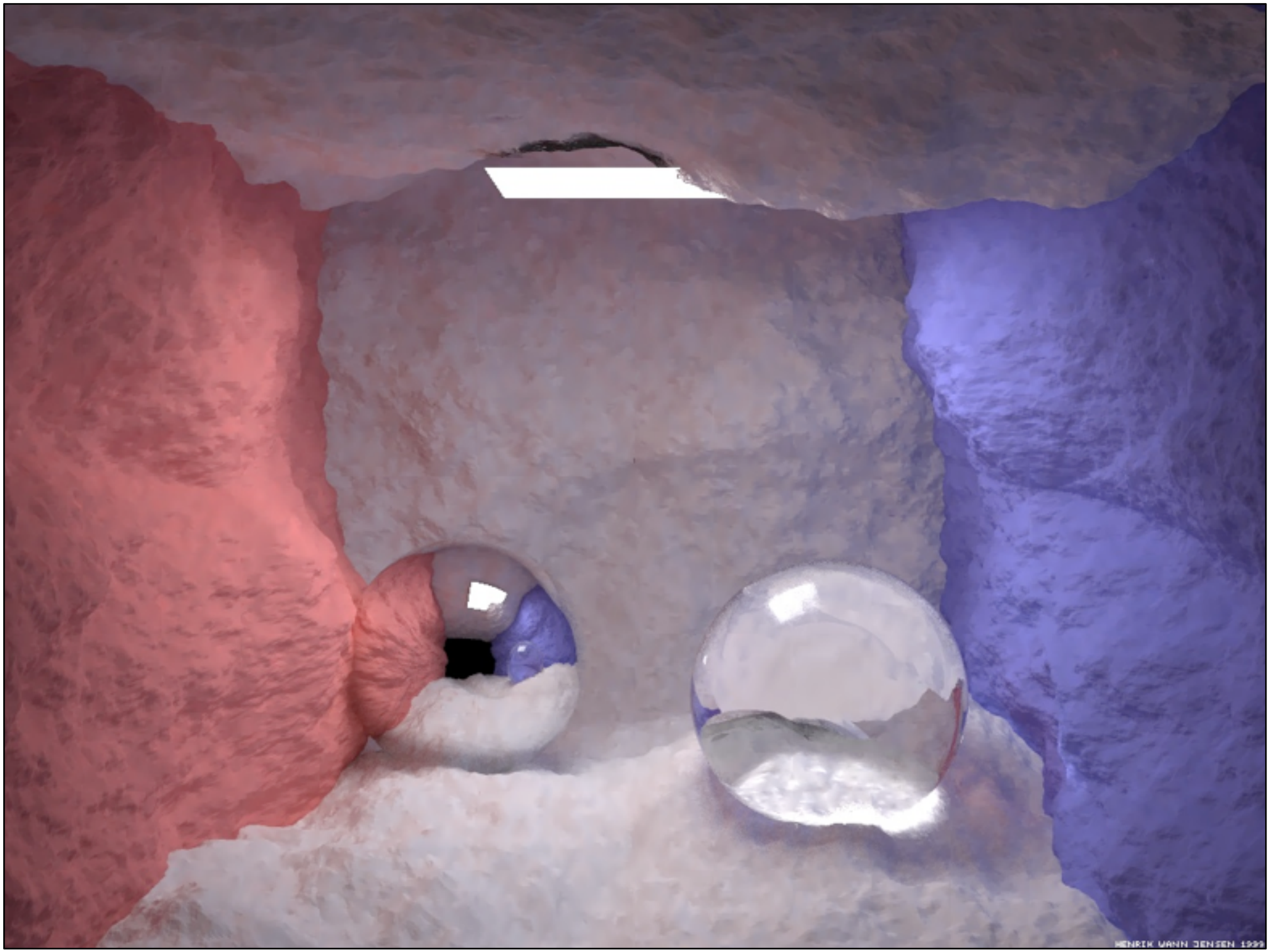


Progressive Photon Mapping

Progressive PM - Summary

- Reduces memory footprint
  - Converges without requiring infinite memory
- Renders progressively (user-friendly)
- Data structure does not need to be as sophisticated
- No need to bother using a caustic map, just use a single photon map for everything

Visual Break

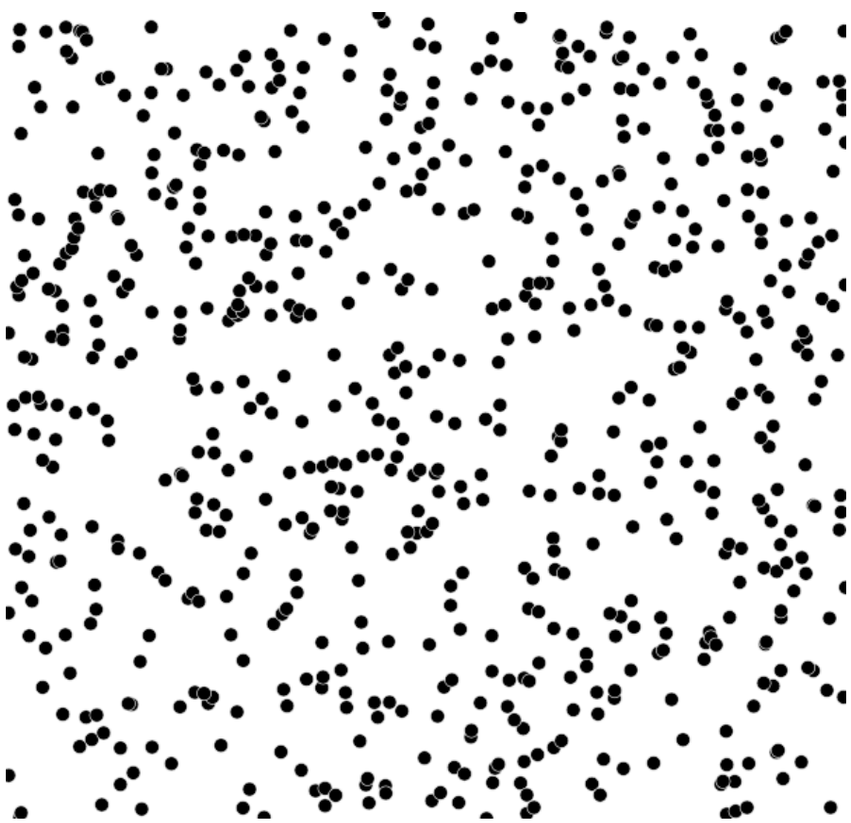


Photon Relaxation

Photon Relaxation

[Spencer and Jones 2009]

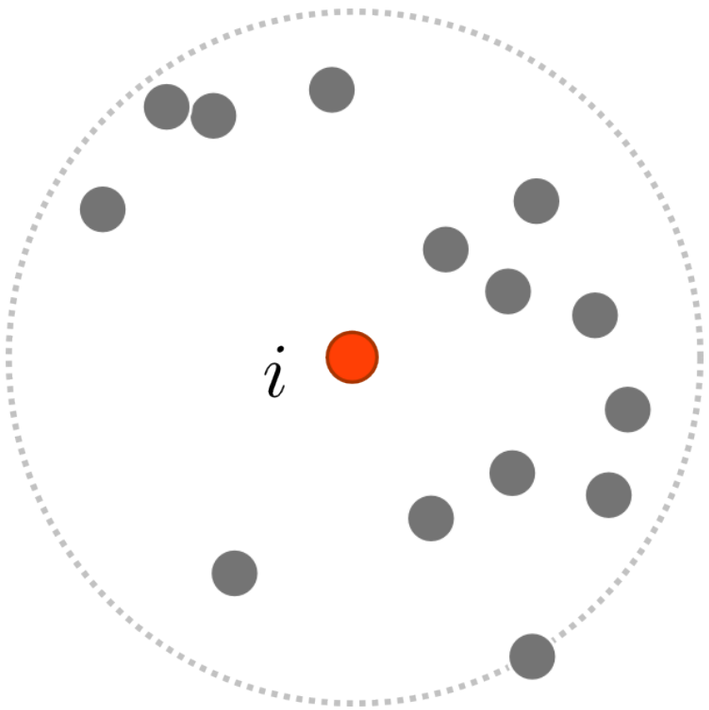
- Photon maps suffer from high point discrepancy due to:
  - stochastic processes (e.g. BRDF sampling)
  - tracing incoherence



Photon distribution

Point Repulsion

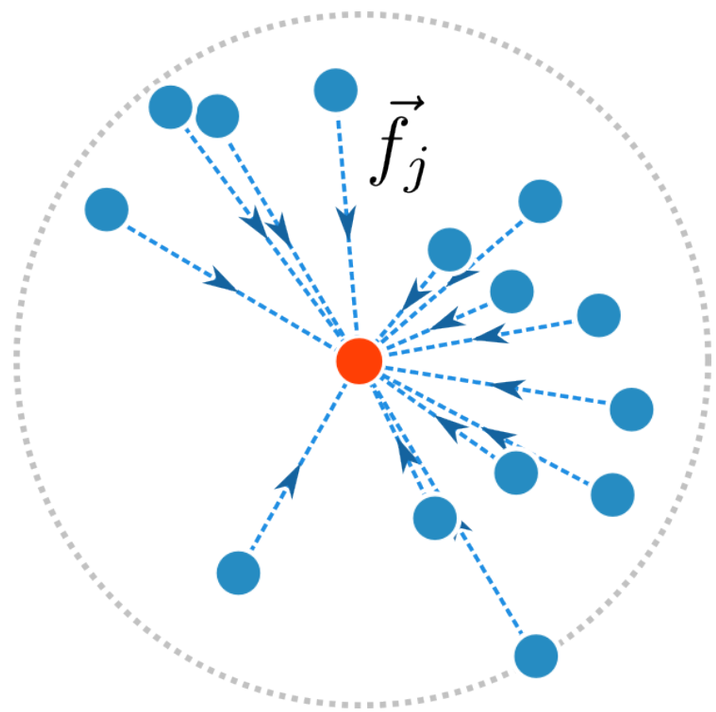
- Goal: minimize local discrepancy
- Iterative repulsion



For each photon,
find *k* nearest neighbors

Point Repulsion

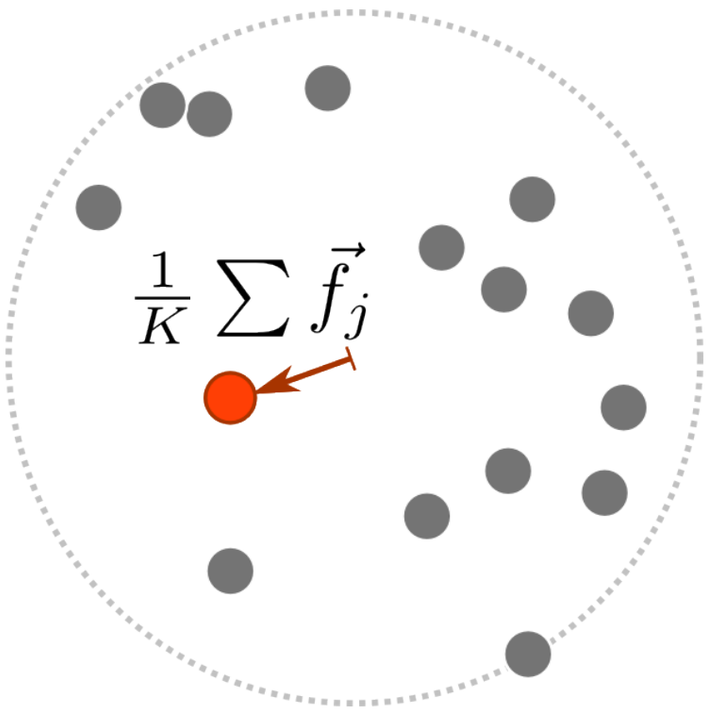
- Goal: minimize local discrepancy
- Iterative repulsion



Compute individual repulsion forces

Point Repulsion

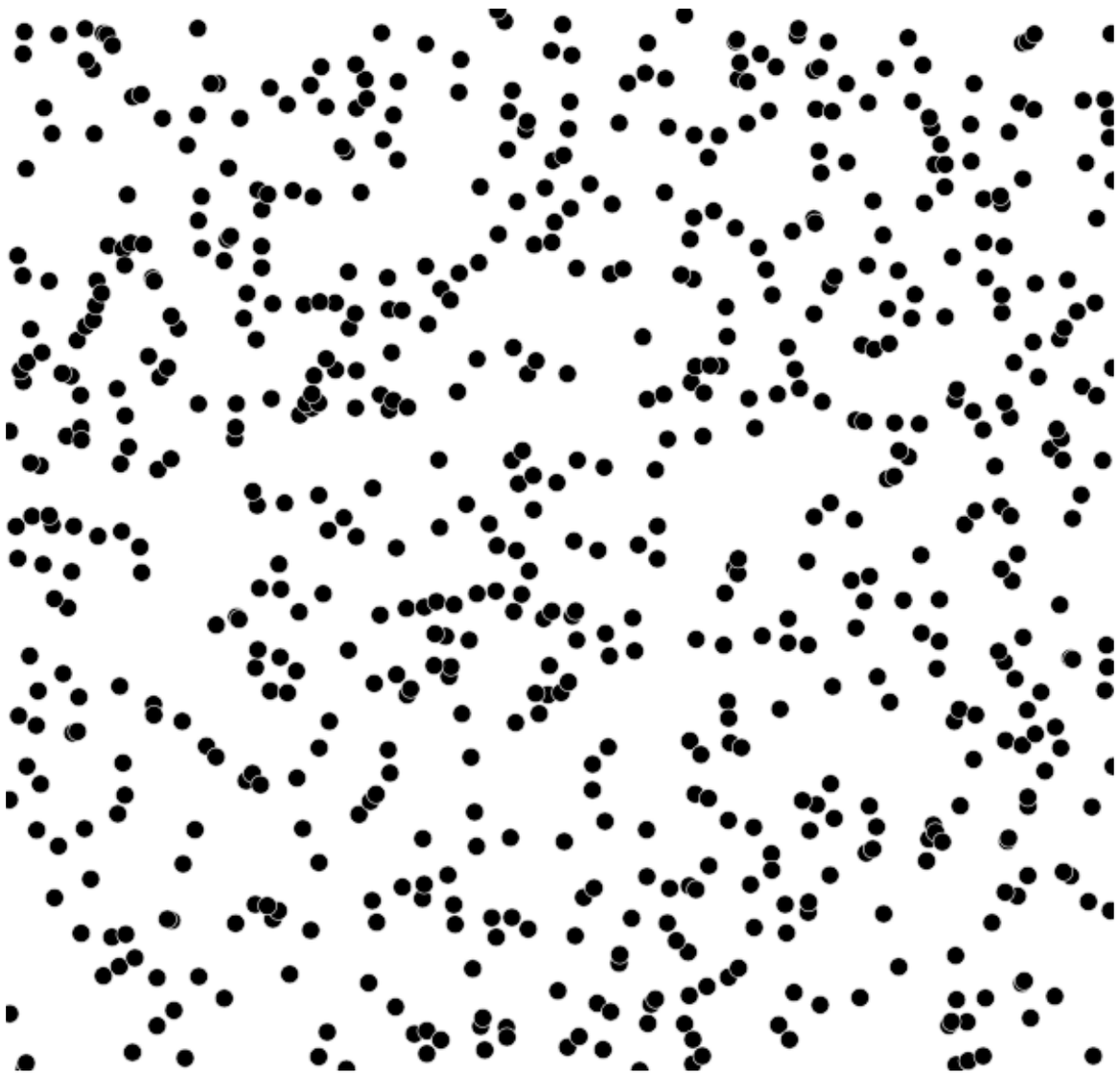
- Goal: minimize local discrepancy
- Iterative repulsion



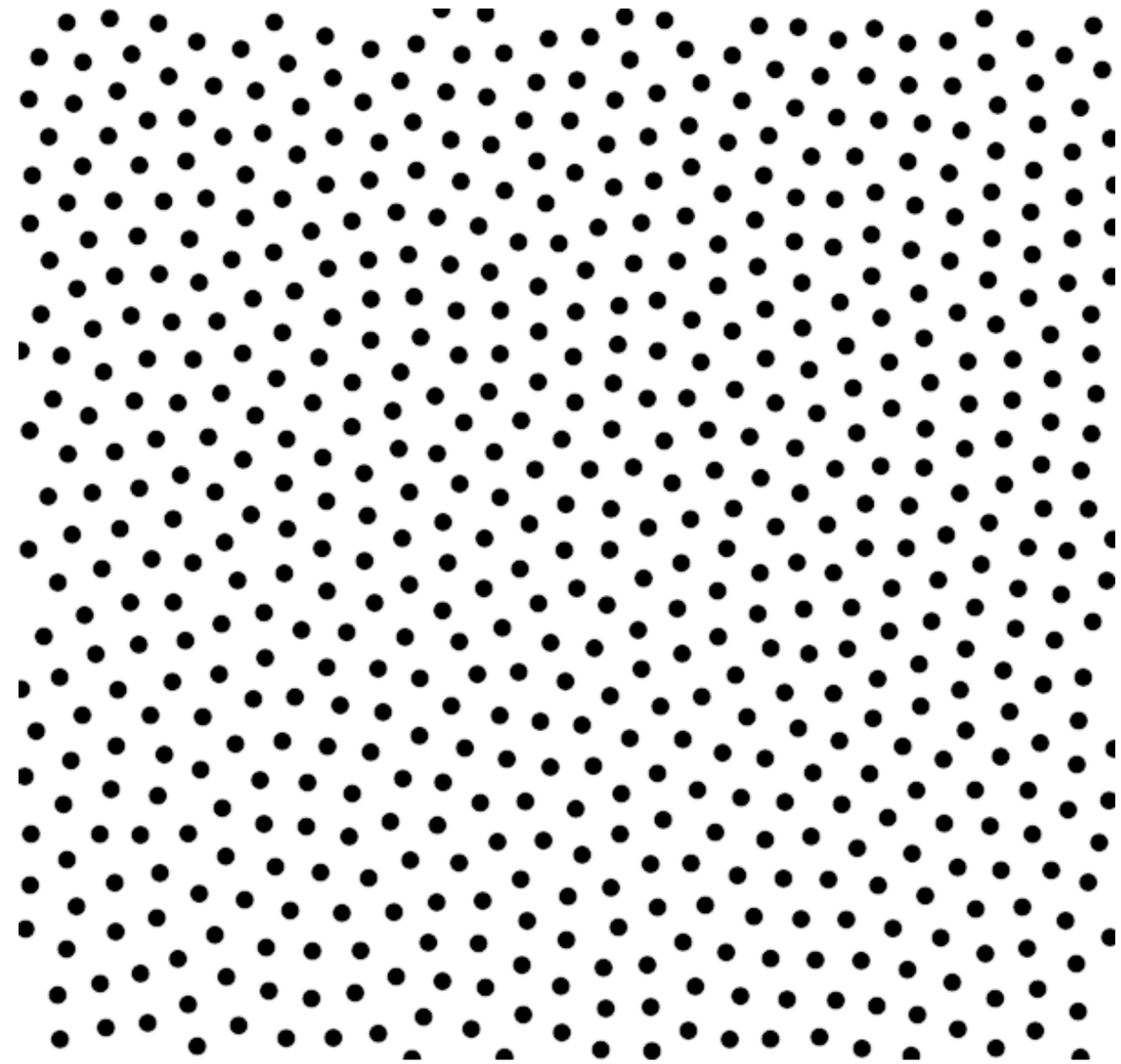
Apply mean force

Photon Relaxation

Initial photon distribution



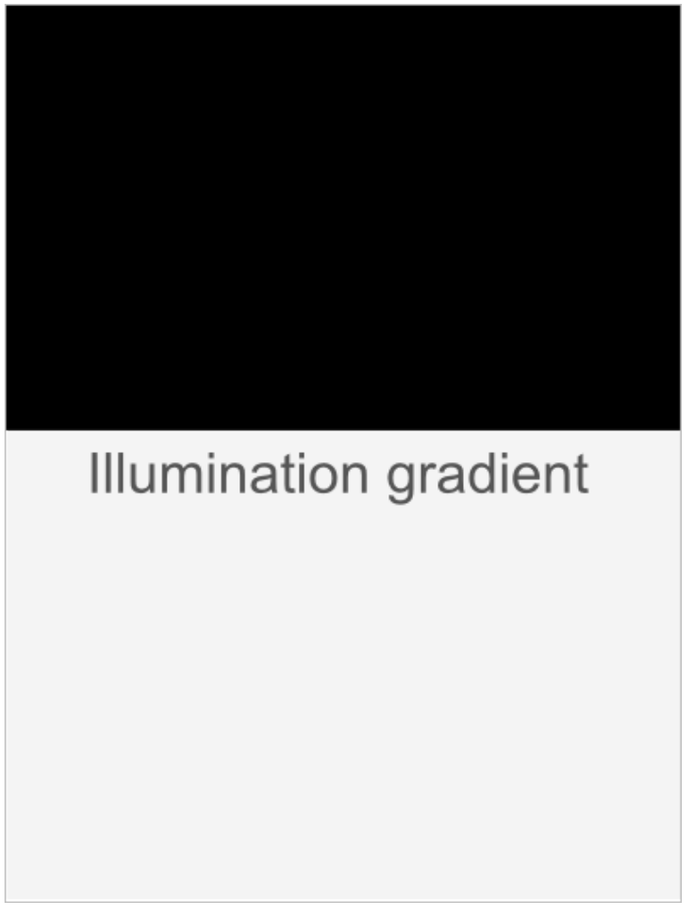
After relaxation



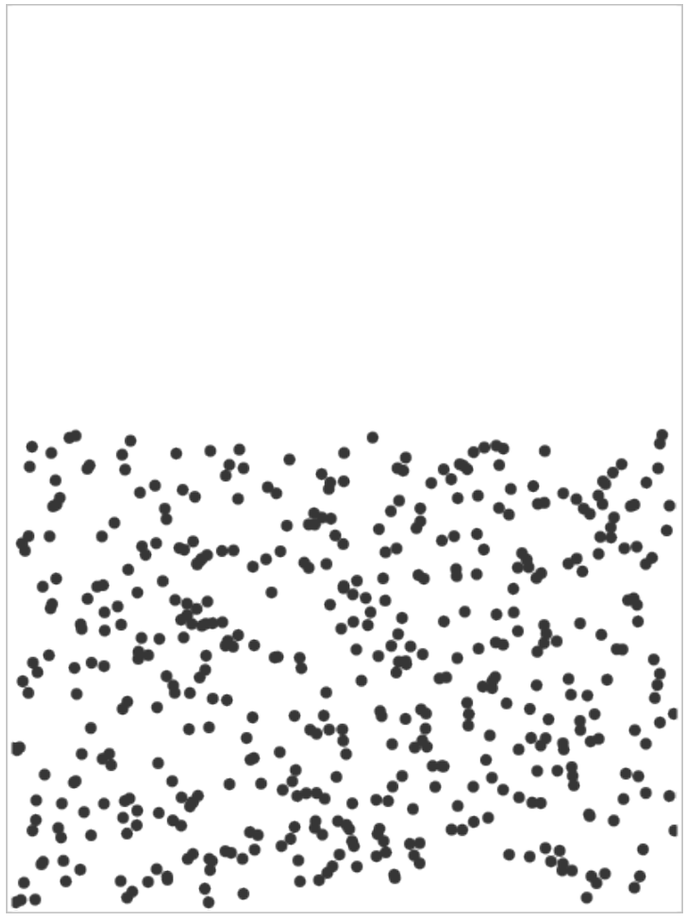
Images courtesy of B. Spencer

Photon Relaxation Issues

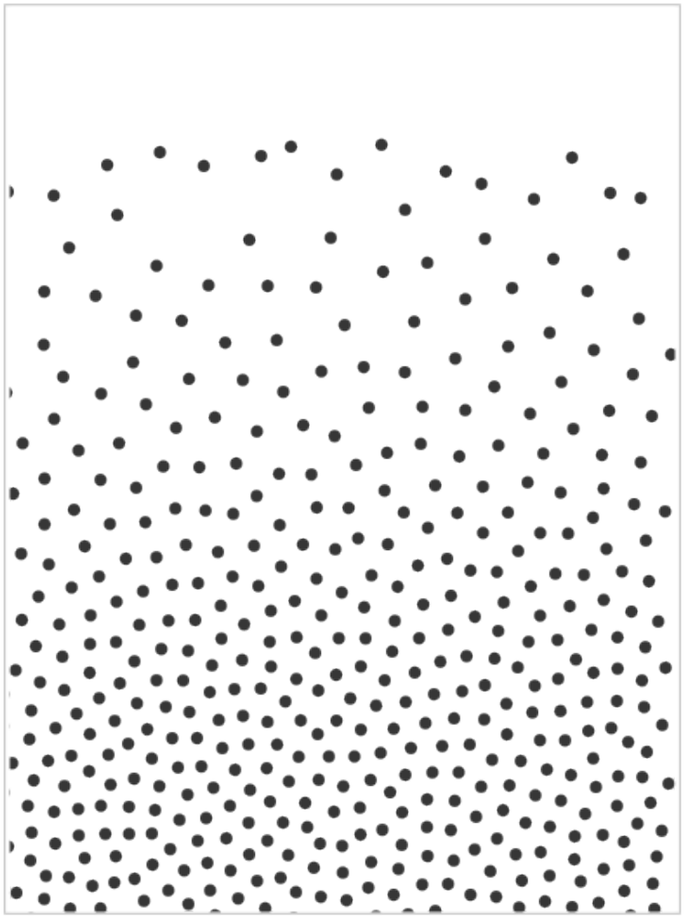
Illumination



Initial distribution

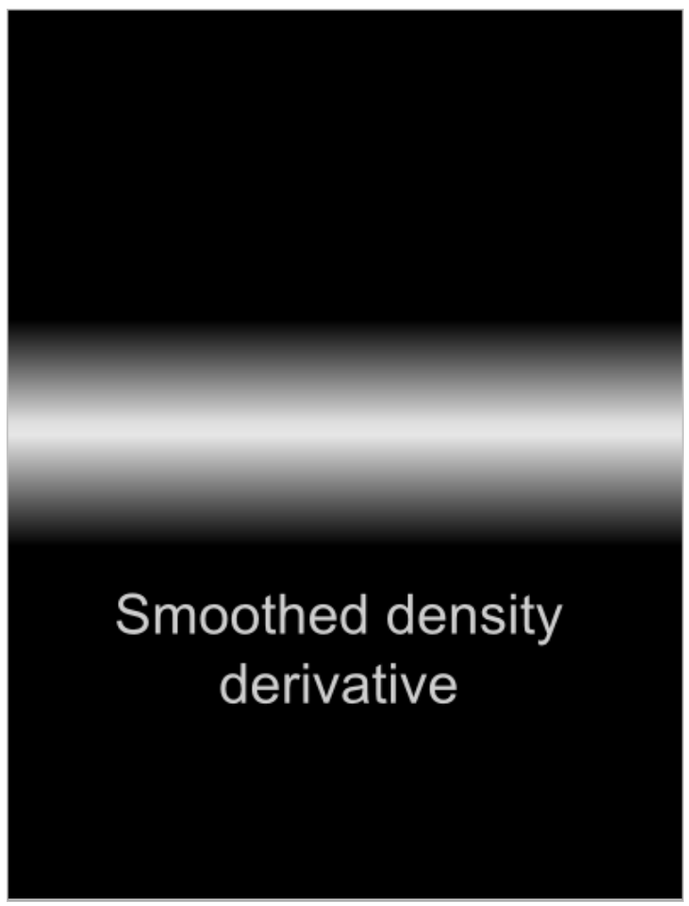


After relaxation

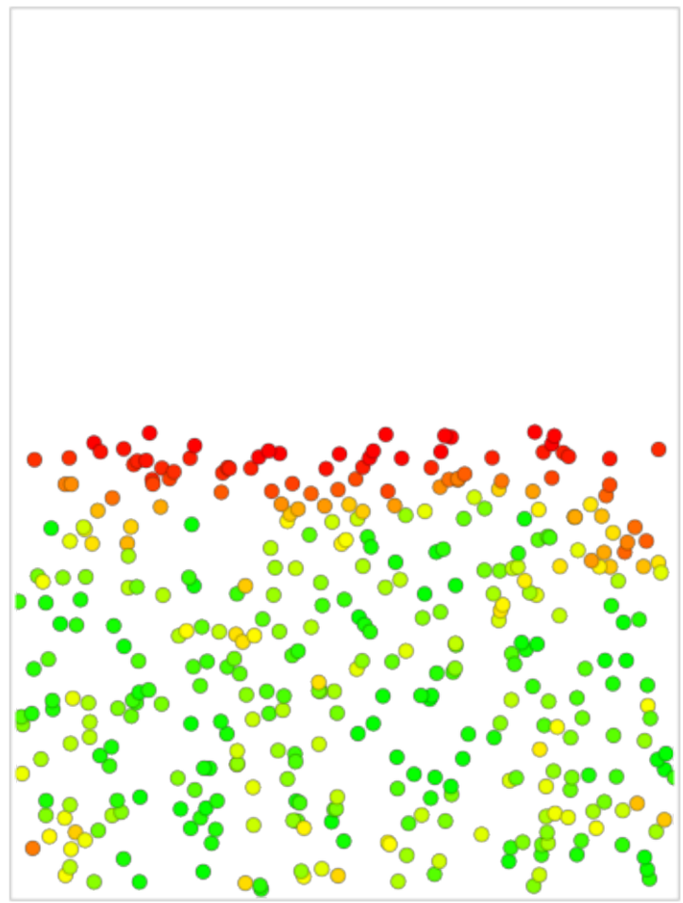


Photon Relaxation Issues

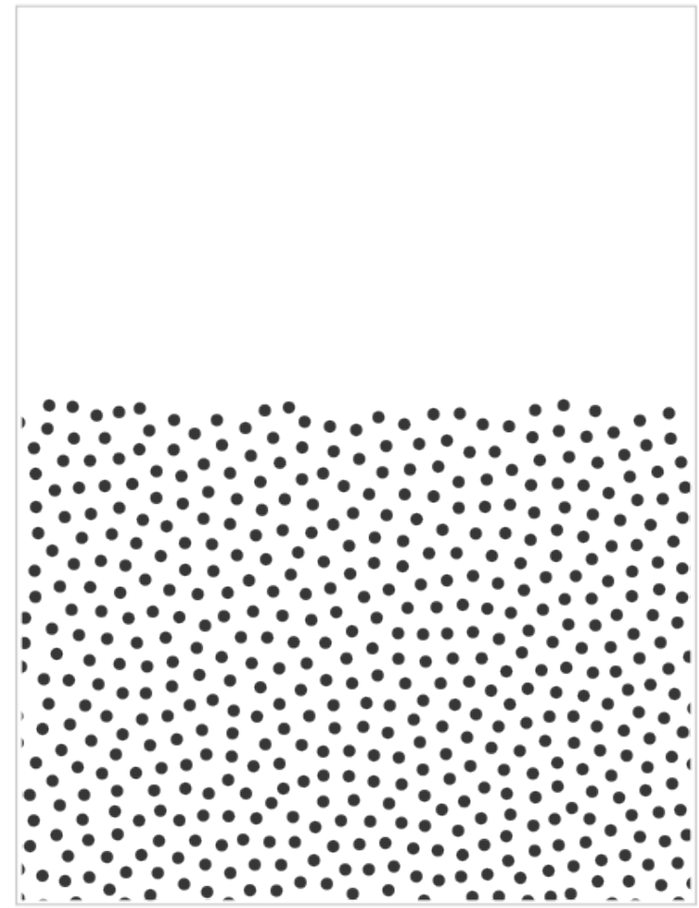
Illumination gradient



Constraints



After relaxation



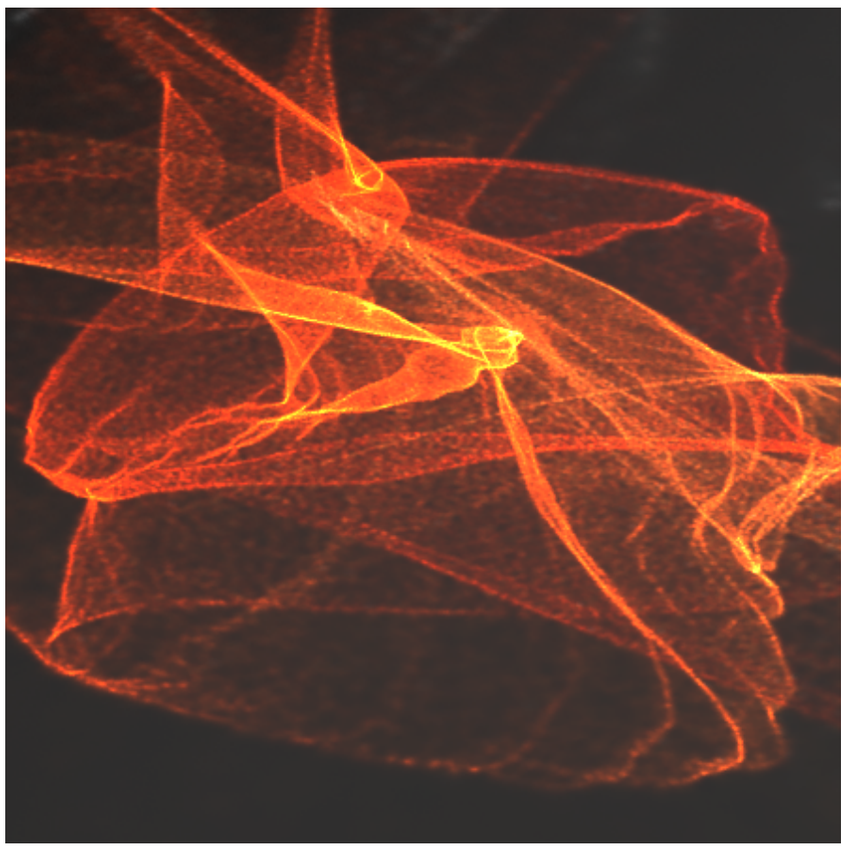
Images courtesy of B. Spencer

Photon Relaxation

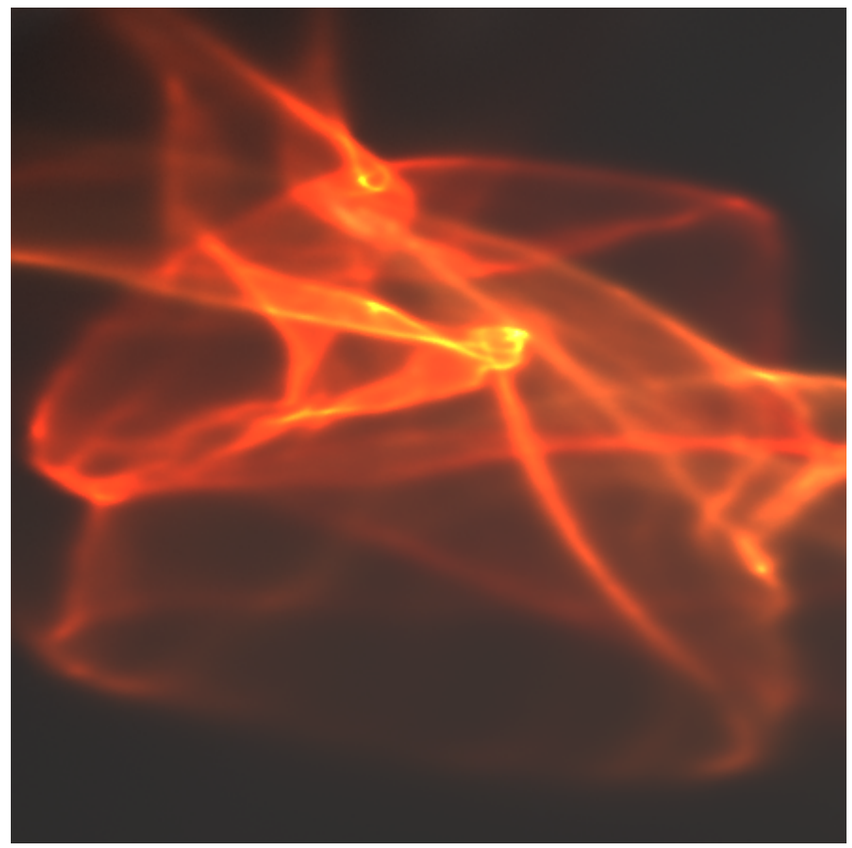
Images courtesy of B. Spencer

Photon Relaxation

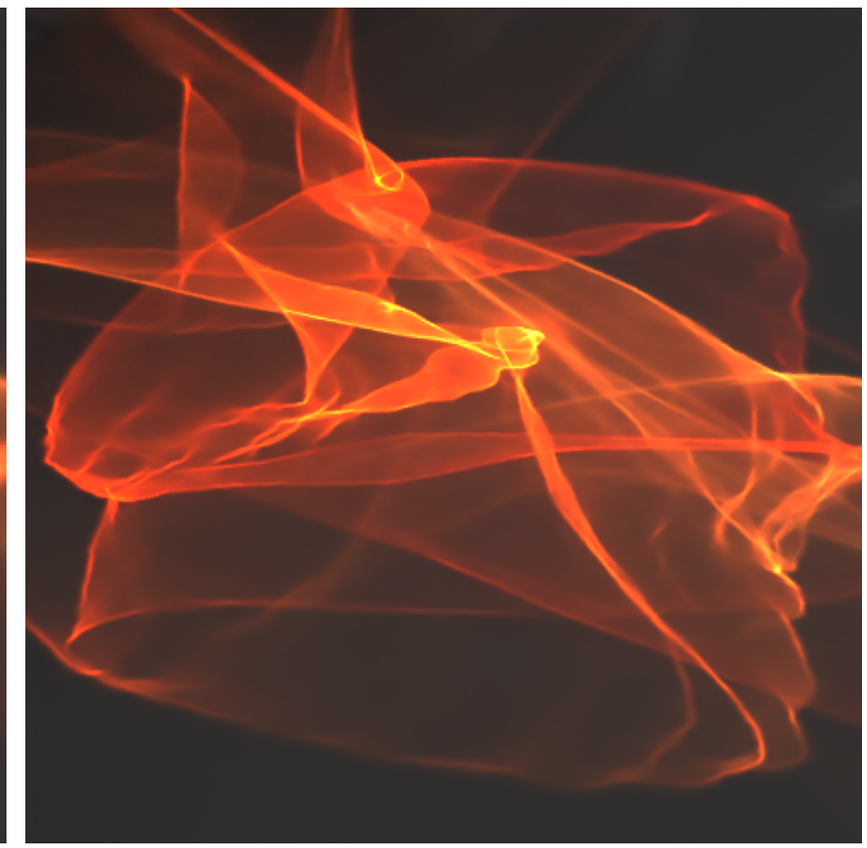
Initial distribution



Relaxation,
no constraints



Relaxation
with constraints



Images courtesy of B. Spencer

Photon Relaxation

Images courtesy of B. Spencer

Photon Relaxation

Photon Relaxation

Photon Mapping in Industry

PIXAR's RenderMan

Luxrender
GPL PHYSICALLY BASED RENDERER

NVIDIA®
mental ray®

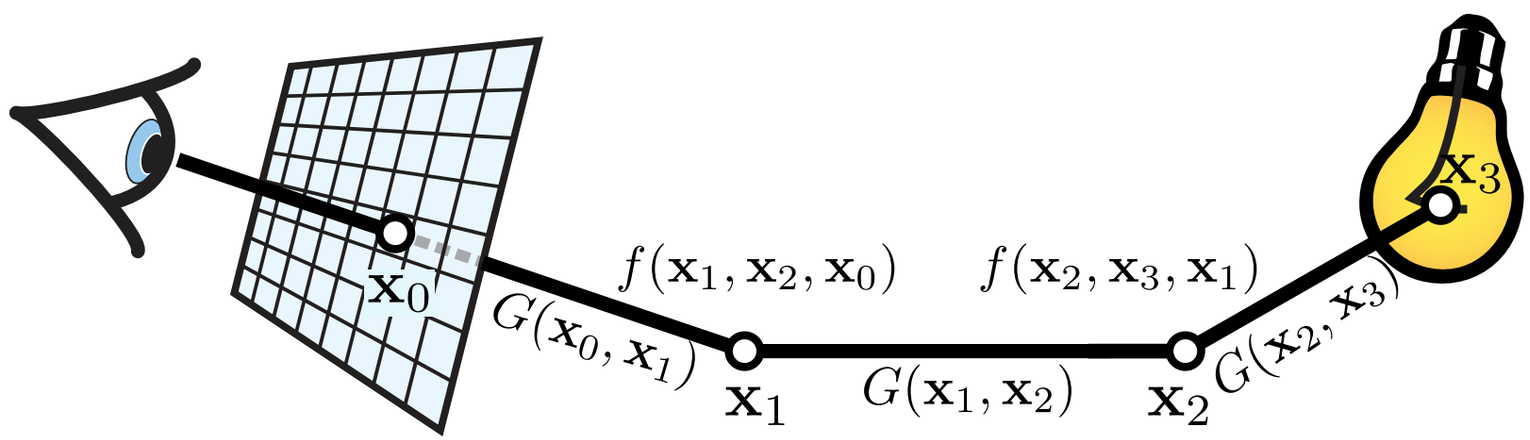
v·ray

Questions?

Recall: Path Integral Measurement Eq.

$$I_j = \int_{\mathcal{P}} W_e(\mathbf{x}_0, \mathbf{x}_1) L_e(\mathbf{x}_k, \mathbf{x}_{k-1}) T(\bar{\mathbf{x}})\, d\bar{\mathbf{x}}$$

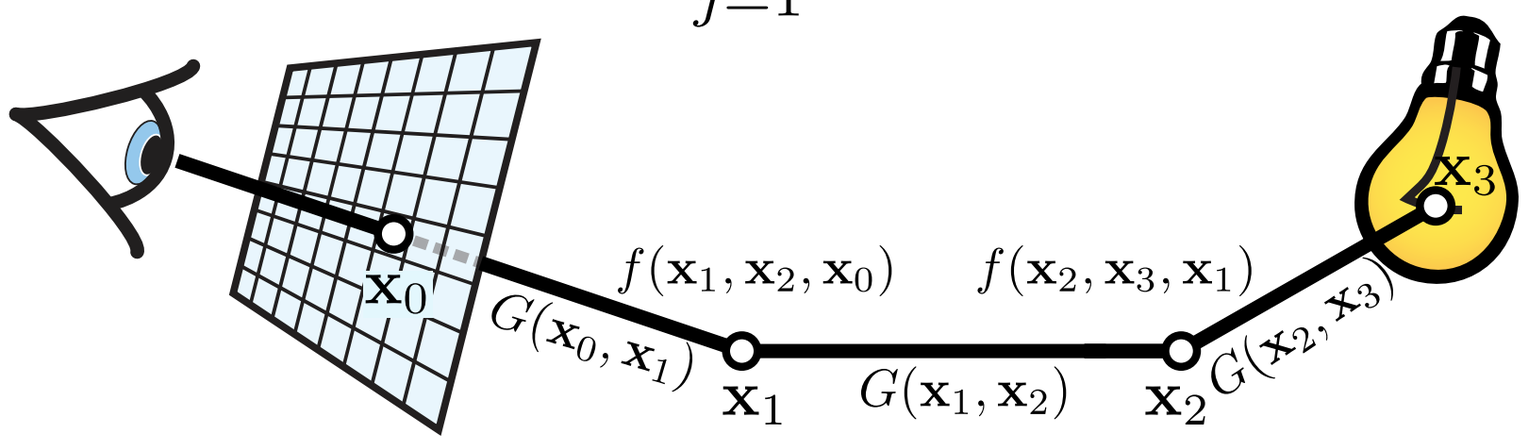
path throughput $T(\bar{\mathbf{x}}) = G(\mathbf{x}_0, \mathbf{x}_1) \prod_{j=1}^{k-1} f(\mathbf{x}_j, \mathbf{x}_{j+1}, \mathbf{x}_{j-1}) G(\mathbf{x}_j, \mathbf{x}_{j+1})$



Recall: Path Integral Measurement Eq.

$$I_j = \int_{\mathcal{P}} W_e(\mathbf{x}_0, \mathbf{x}_1) L_e(\mathbf{x}_k, \mathbf{x}_{k-1}) T(\bar{\mathbf{x}})\, d\bar{\mathbf{x}}$$

path throughput $T(\bar{\mathbf{x}}) = G(\mathbf{x}_0, \mathbf{x}_1) \prod_{j=1}^{k-1} f(\mathbf{x}_j, \mathbf{x}_{j+1}, \mathbf{x}_{j-1}) G(\mathbf{x}_j, \mathbf{x}_{j+1})$



- Monte Carlo estimator:

$$I \approx \frac{W_e(\mathbf{x}_0, \mathbf{x}_1) T(\mathbf{x}_0, \cdots, \mathbf{x}_k) L_e(\mathbf{x}_k, \mathbf{x}_{k-1})}{\underbrace{p(\mathbf{x}_0, \cdots, \mathbf{x}_k)}_{\text{joint PDF of path vertices}}}$$

Photon Mapping

$$I \approx \frac{W_e(\mathbf{x}_0, \mathbf{x}_1) T(\mathbf{x}_0, \cdots, \mathbf{x}_k) L_e(\mathbf{x}_k, \mathbf{x}_{k-1})}{p(\mathbf{x}_0, \cdots, \mathbf{x}_k)}$$

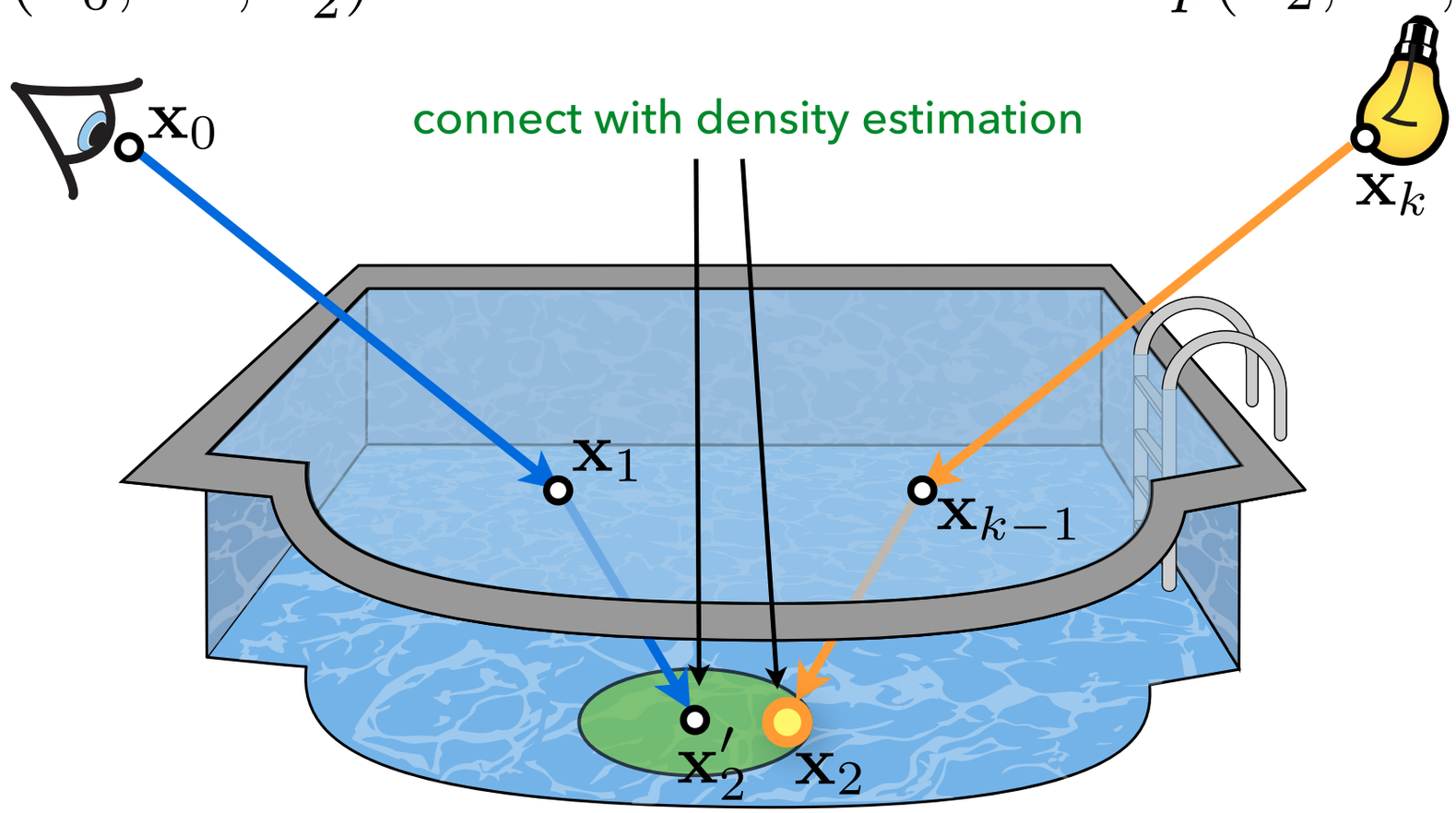
split path contribution into two parts

eye subpath

$$\frac{W_e(\mathbf{x}_0, \mathbf{x}_1) T(\mathbf{x}_0, \cdots, \mathbf{x}_2')}{p(\mathbf{x}_0, \cdots, \mathbf{x}_2')}$$

light subpath/photon "power"

$$\Phi_p = \frac{T(\mathbf{x}_2, \cdots, \mathbf{x}_k) L_e(\mathbf{x}_k, \mathbf{x}_{k-1})}{p(\mathbf{x}_2, \cdots, \mathbf{x}_k)}$$



VPLs

$$I \approx \frac{W_e(\mathbf{x}_0, \mathbf{x}_1) T(\mathbf{x}_0, \cdots, \mathbf{x}_k) L_e(\mathbf{x}_k, \mathbf{x}_{k-1})}{p(\mathbf{x}_0, \cdots, \mathbf{x}_k)}$$

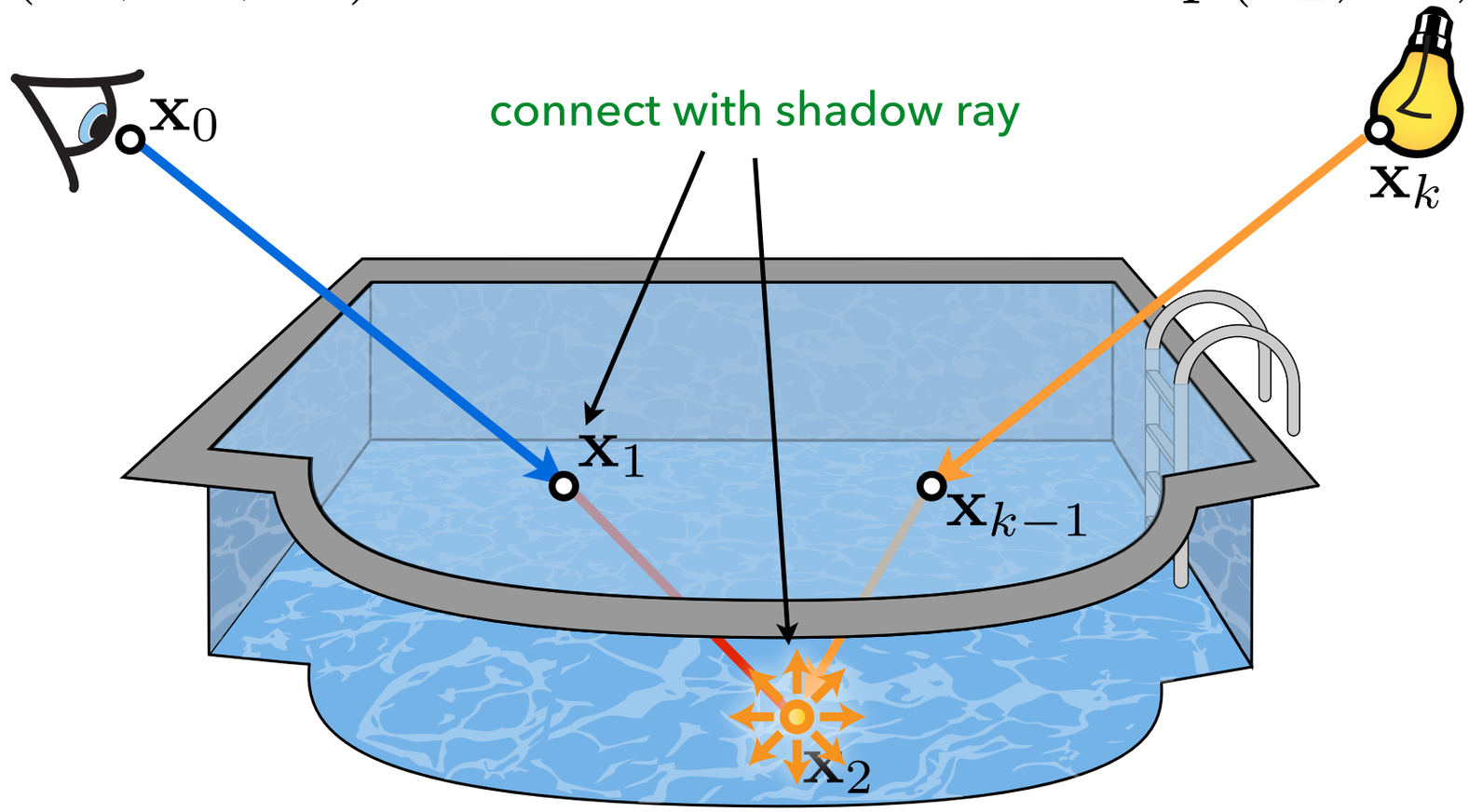
split path contribution into two parts

eye subpath

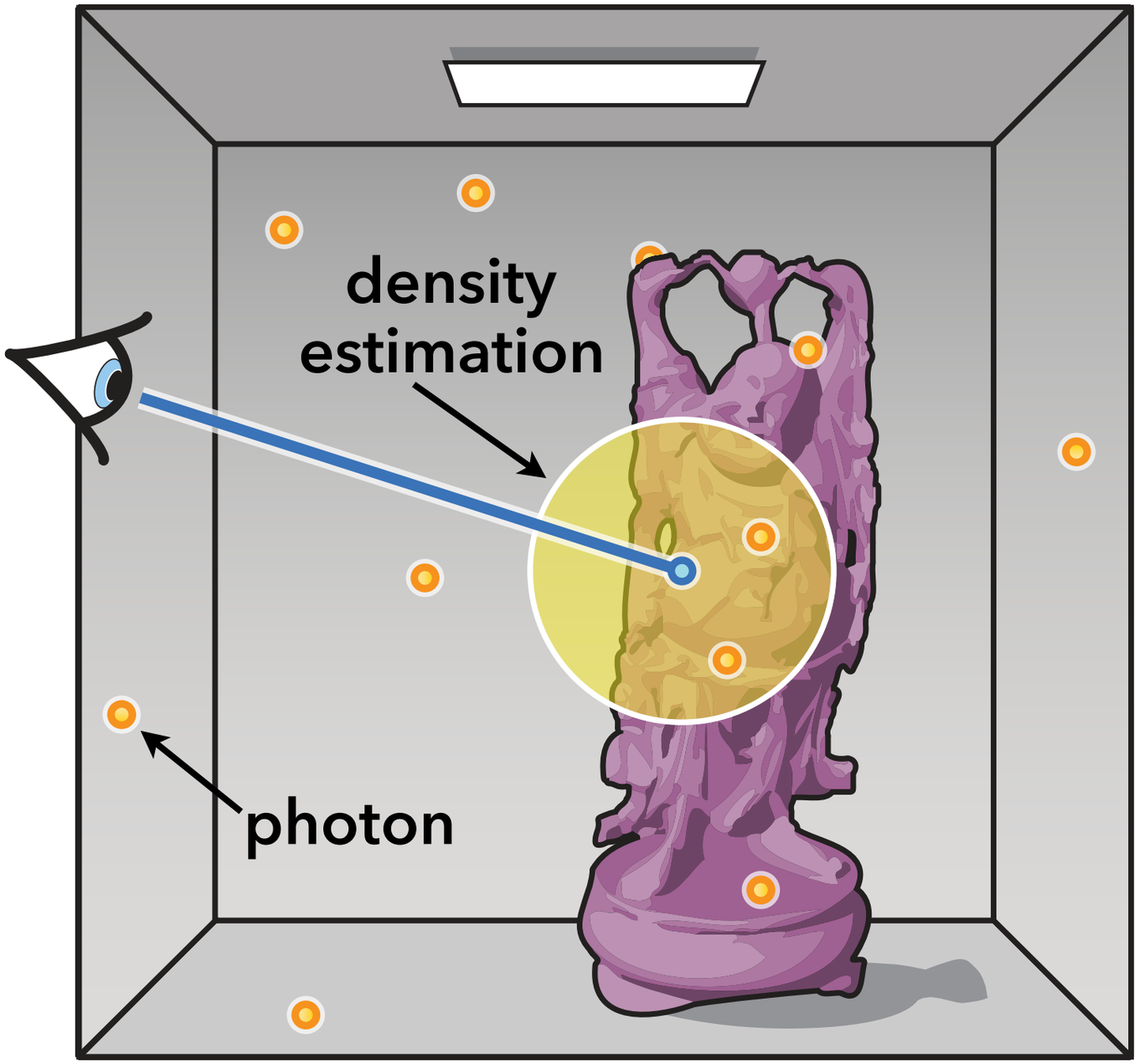
$$\frac{W_e(\mathbf{x}_0, \mathbf{x}_1) T(\mathbf{x}_0, \cdots, \mathbf{x}_1)}{p(\mathbf{x}_0, \cdots, \mathbf{x}_1)}$$

light subpath/VPL "power"

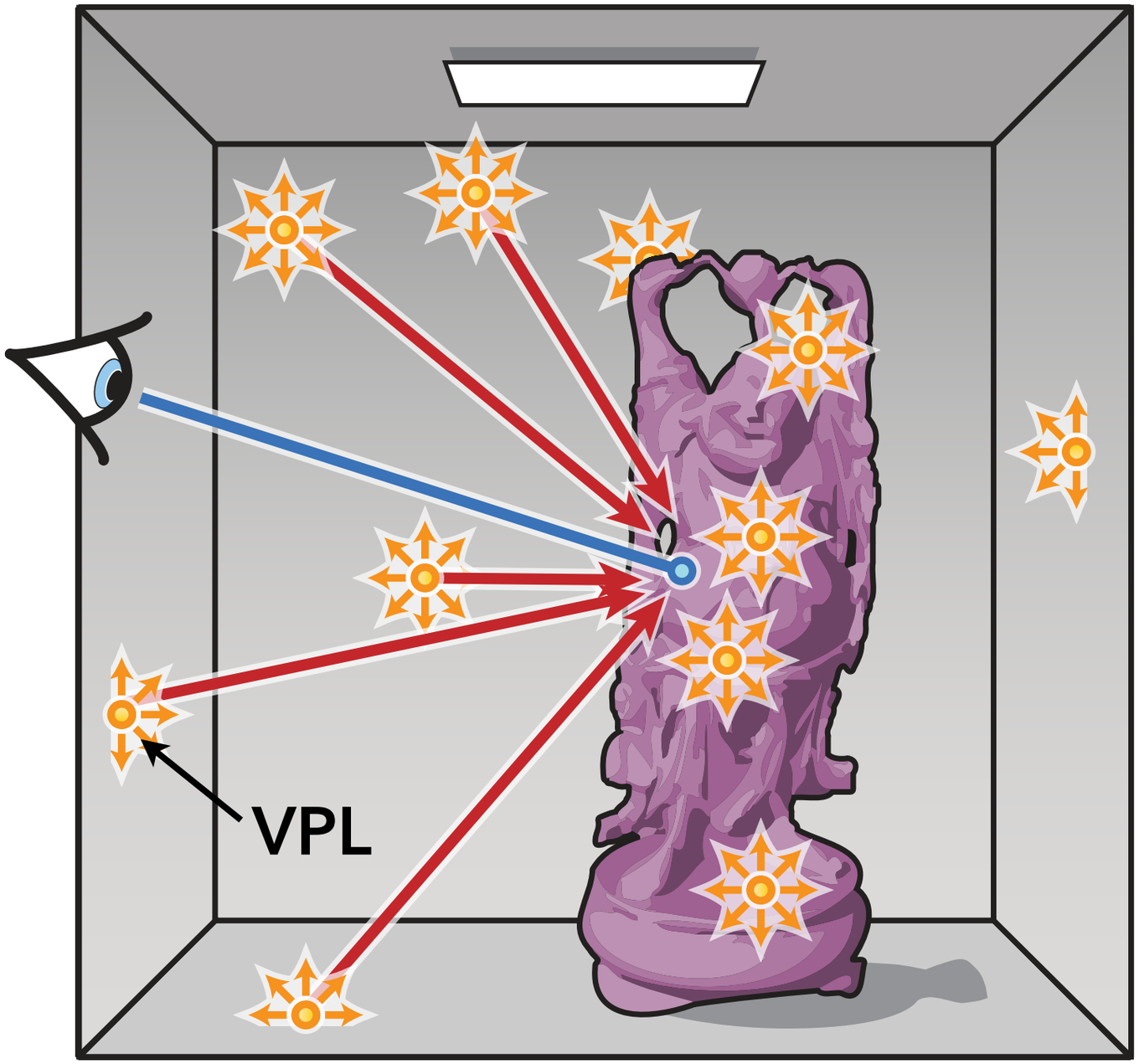
$$\Phi_p = \frac{T(\mathbf{x}_2, \cdots, \mathbf{x}_k) L_e(\mathbf{x}_k, \mathbf{x}_{k-1})}{p(\mathbf{x}_2, \cdots, \mathbf{x}_k)}$$



Photon Mapping vs. VPLs



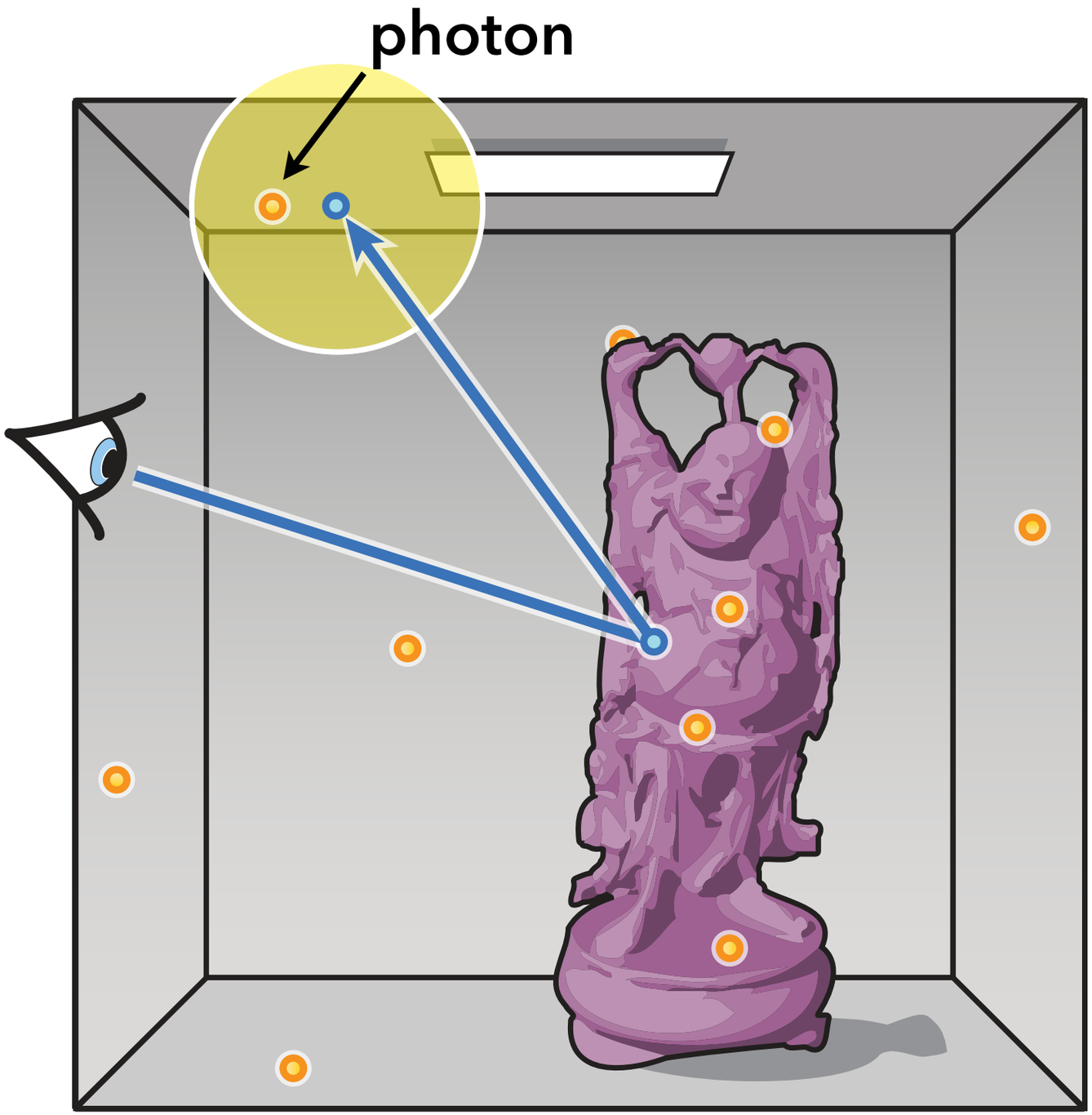
Photon Mapping vs. VPLs



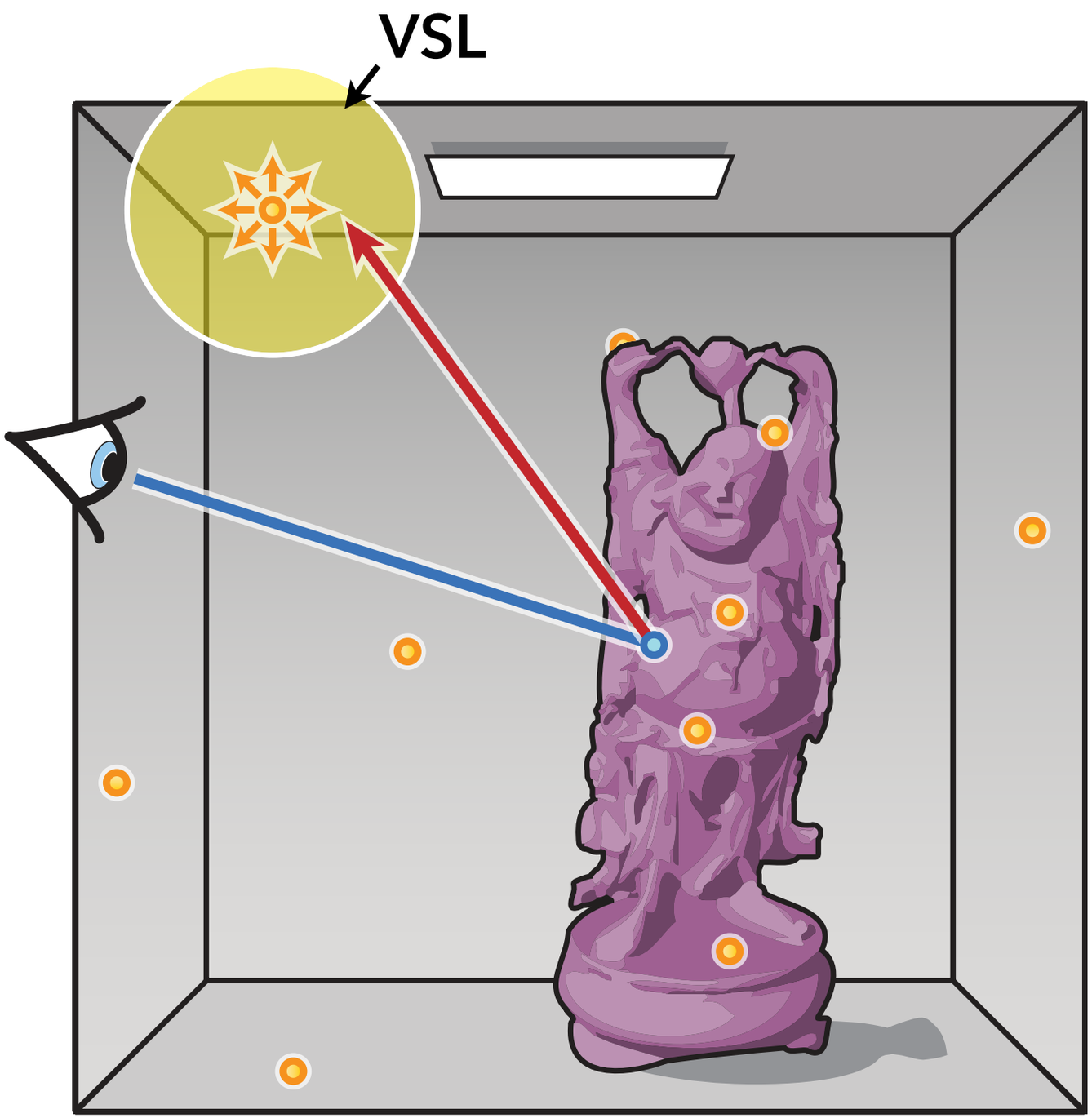
Photon Mapping vs. VPLs

- Light transport?
  - with identical light paths, VPLs yield one-bounce longer paths (thanks to the shadow ray)
  - VPLs great for smooth indirect illumination, photon mapping great for sharp illumination features
- Biased vs. Unbiased?
  - Photon mapping is biased but consistent
  - VPLs are theoretically unbiased, but most practical implementations are not even consistent

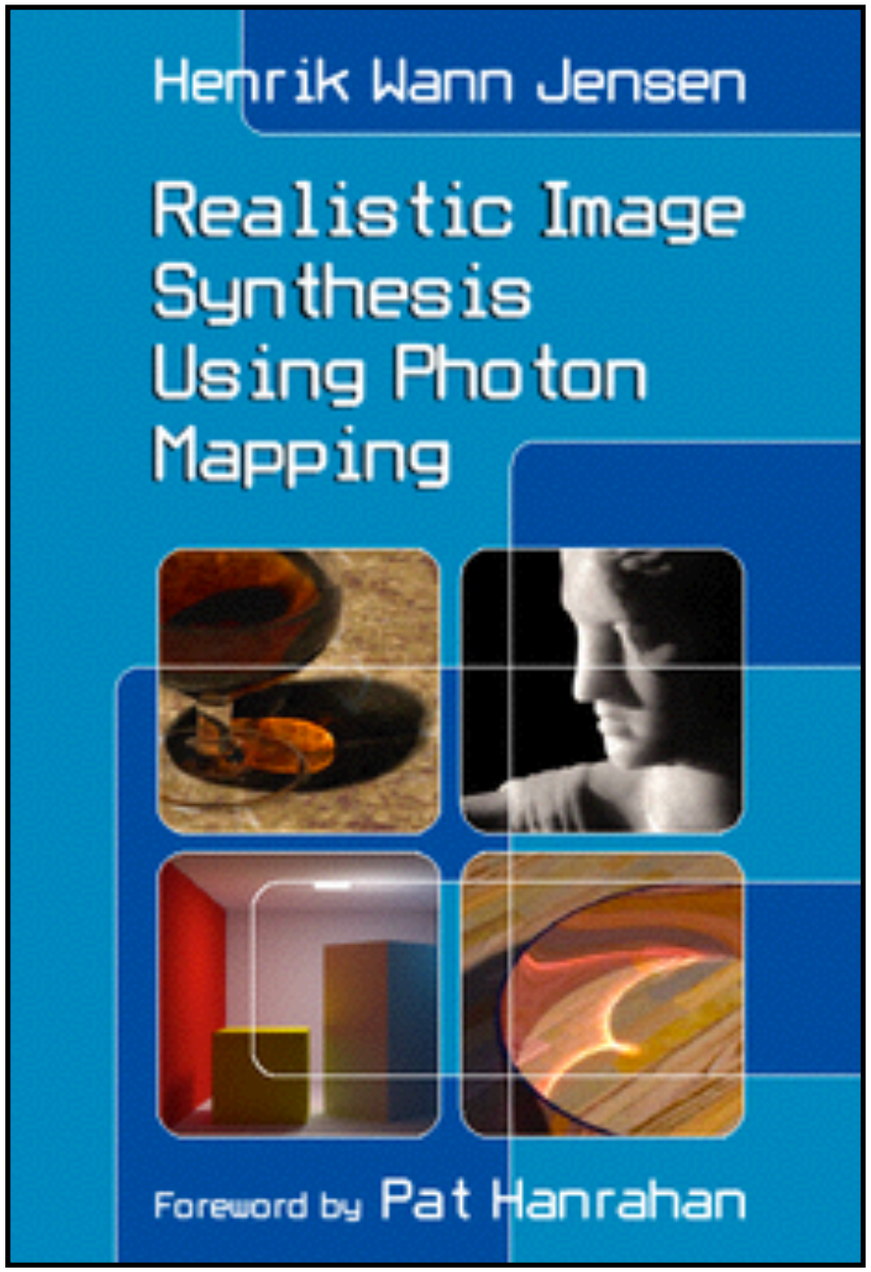
PM with Final Gather vs. VSLs



PM with Final Gather vs. VSLs



PM with Final Gather vs. VSLs

- Light transport:
  - identical
- Differences:
  - connections to light paths are constructed differently
  - only subtle differences in the approximations

More On Photon Mapping