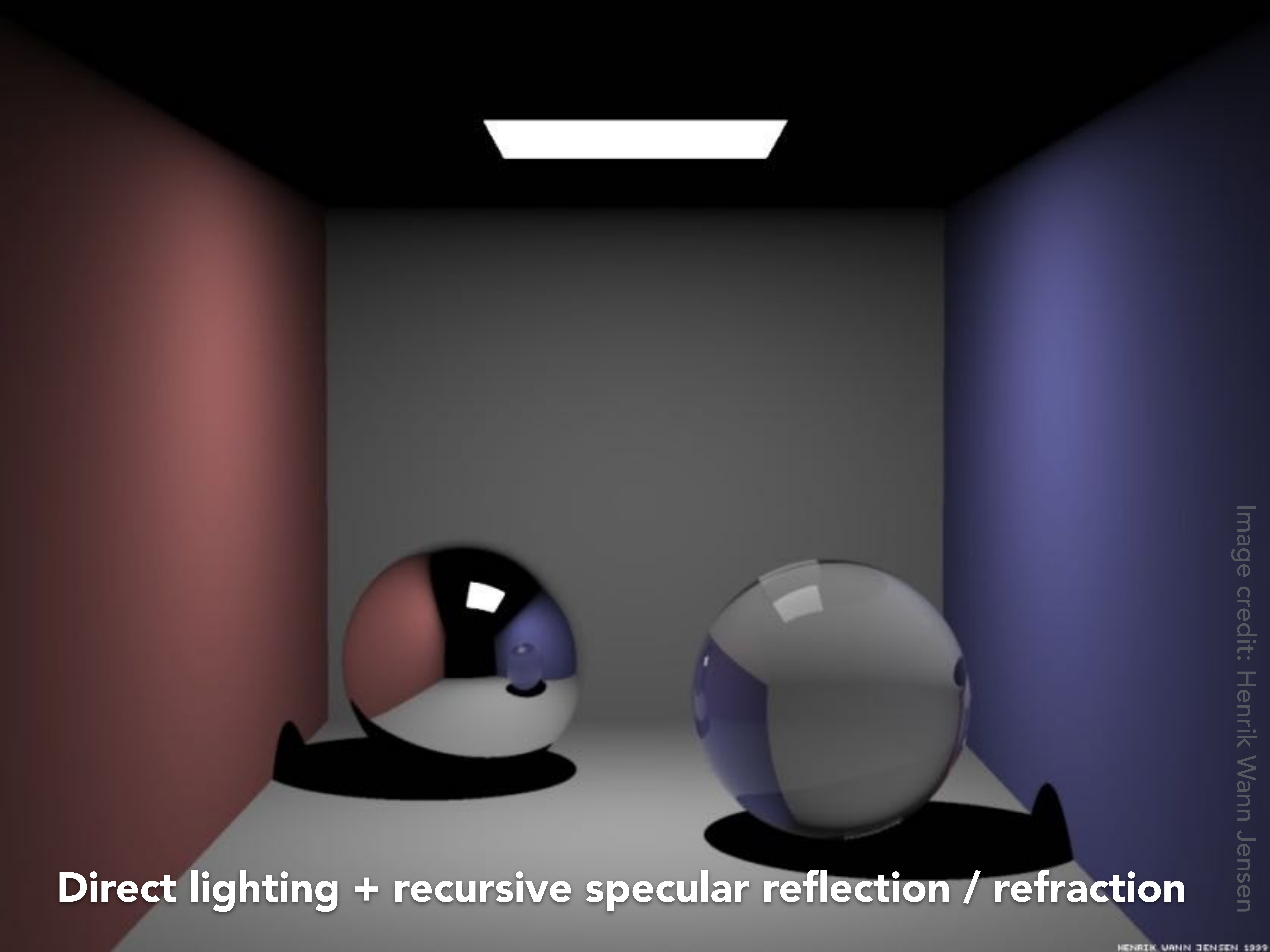
Lecture 13:

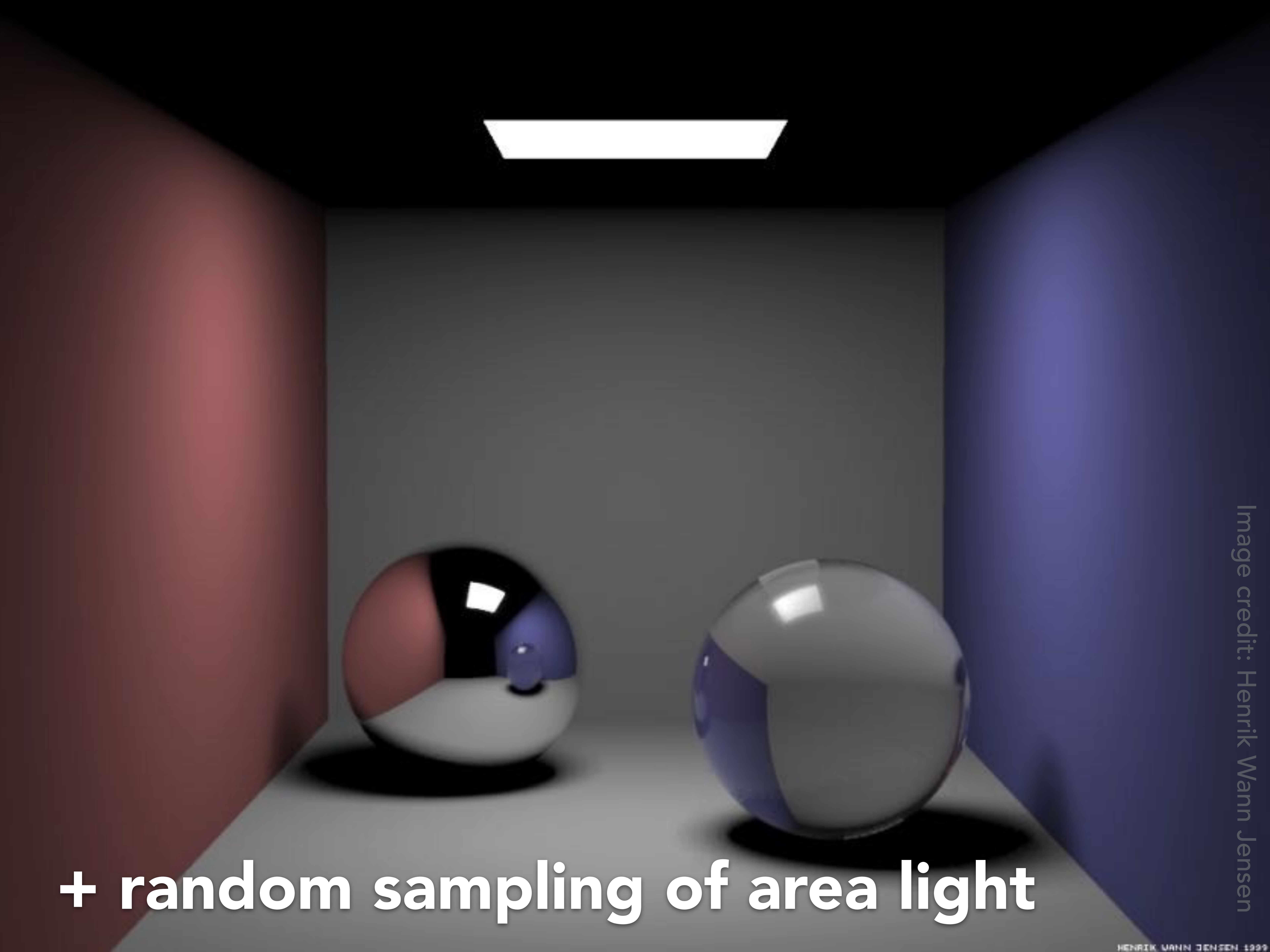
Global Illumination & Path Tracing

Computer Graphics and Imaging
UC Berkeley CS184/284A



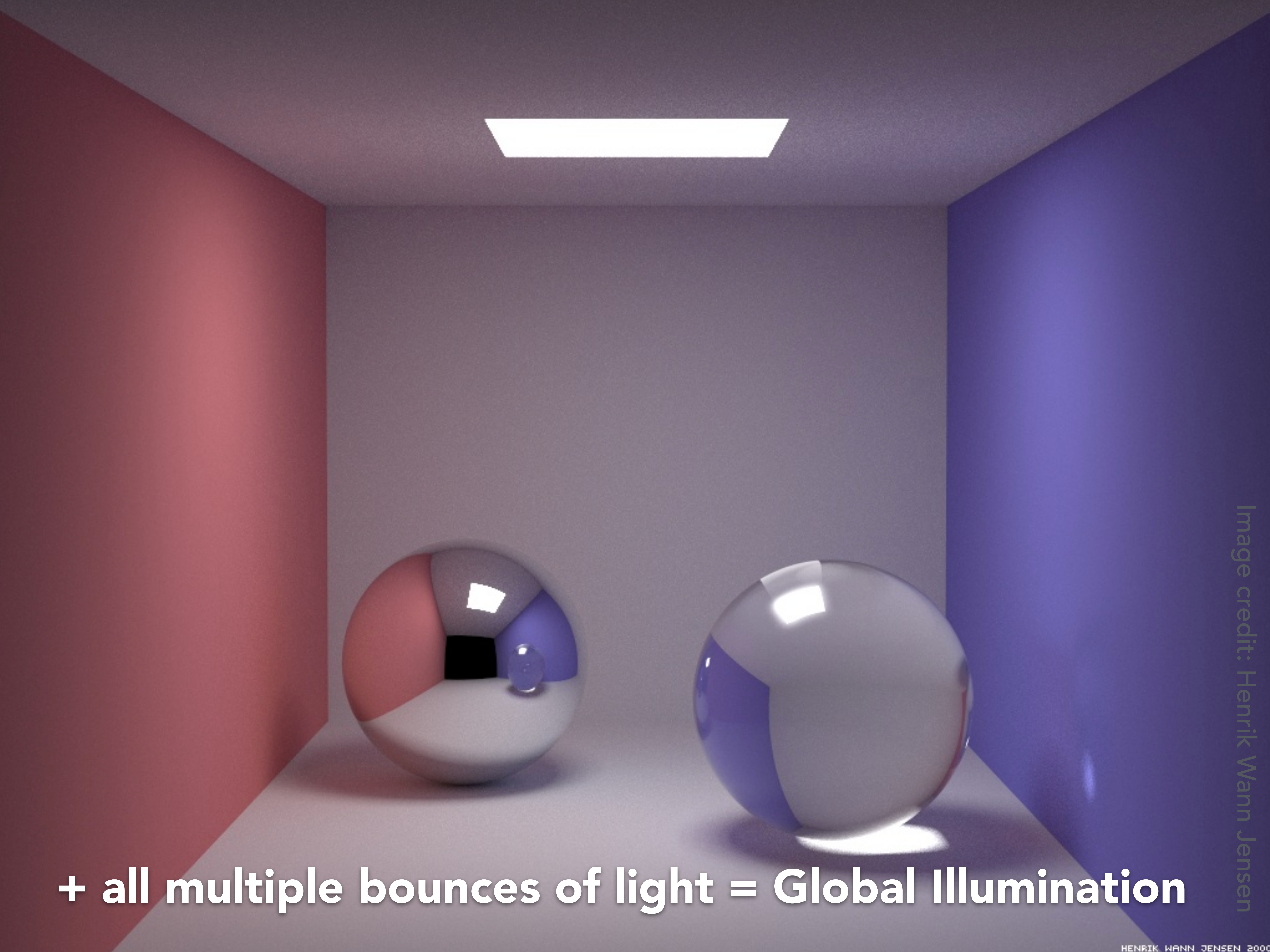
Direct lighting + recursive specular reflection / refraction

Image credit: Henrik Wann Jensen



+ random sampling of area light

Image credit: Henrik Wann Jensen

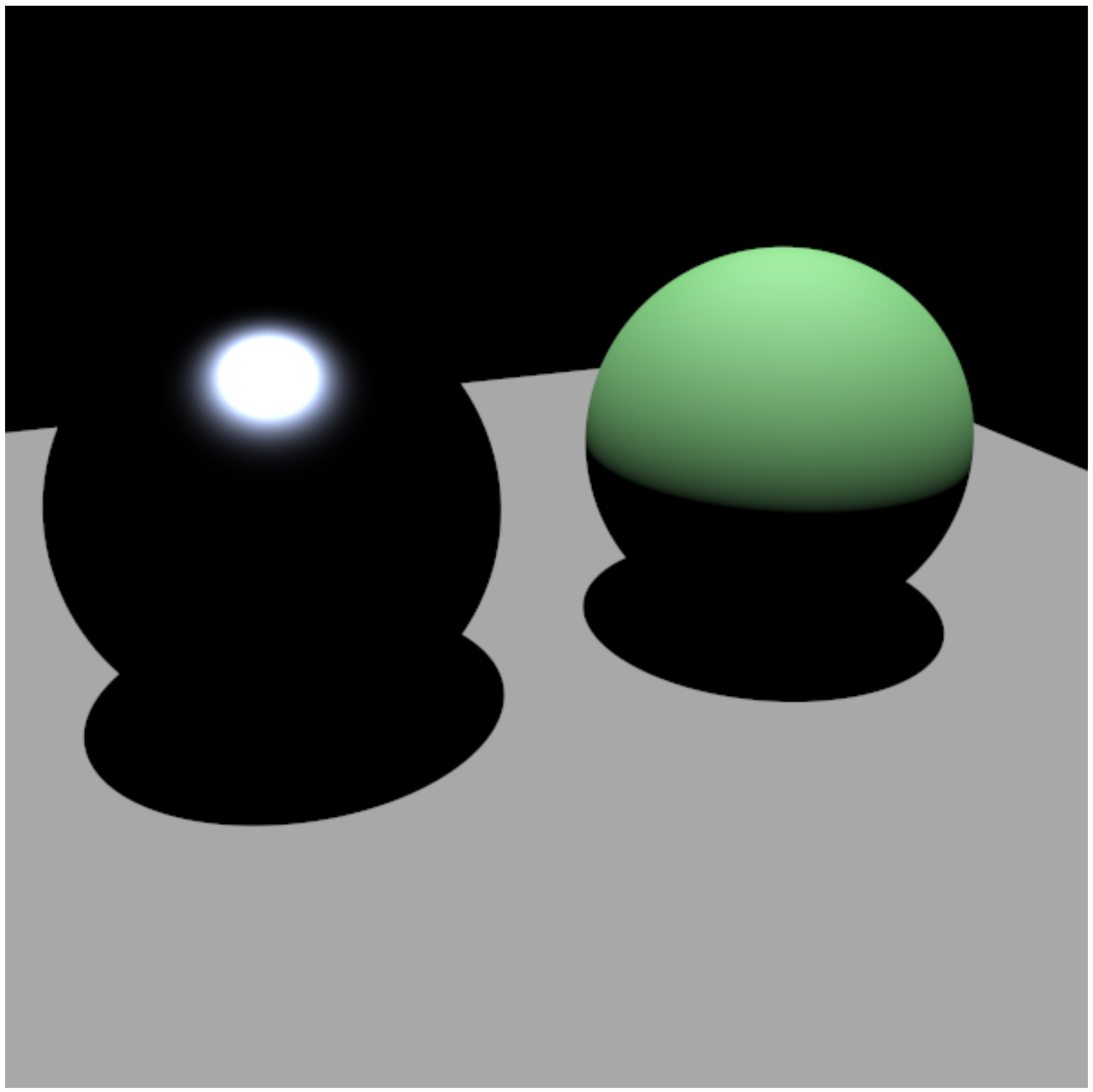

+ all multiple bounces of light = Global Illumination
Image credit: Henrik Wann Jensen
HENRIK WANN JENSEN 2000

Cornell Box – Photograph vs Rendering

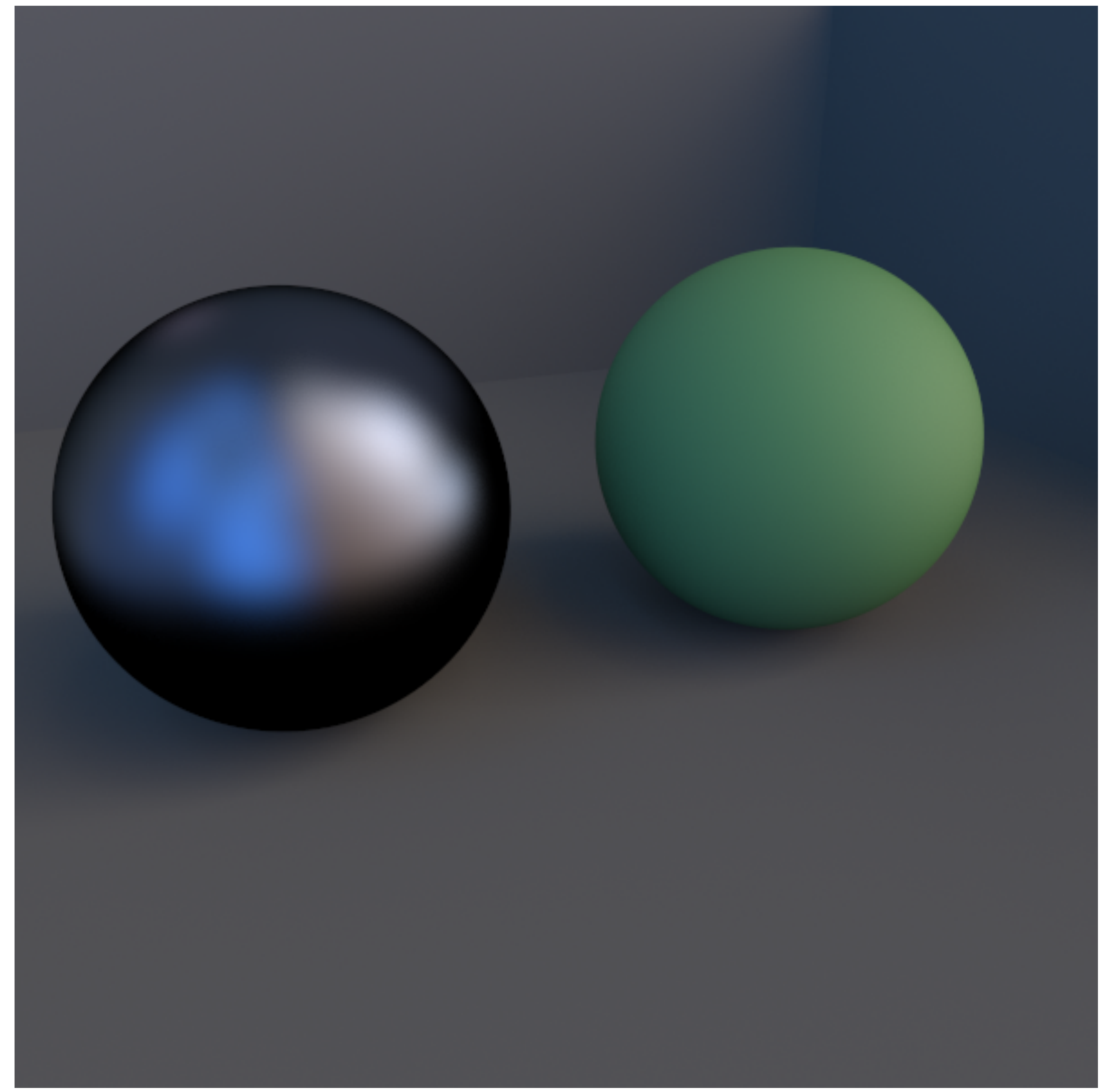
 

Photograph (CCD) vs. global illumination rendering

Visual Richness from Complex Lighting



Point Light



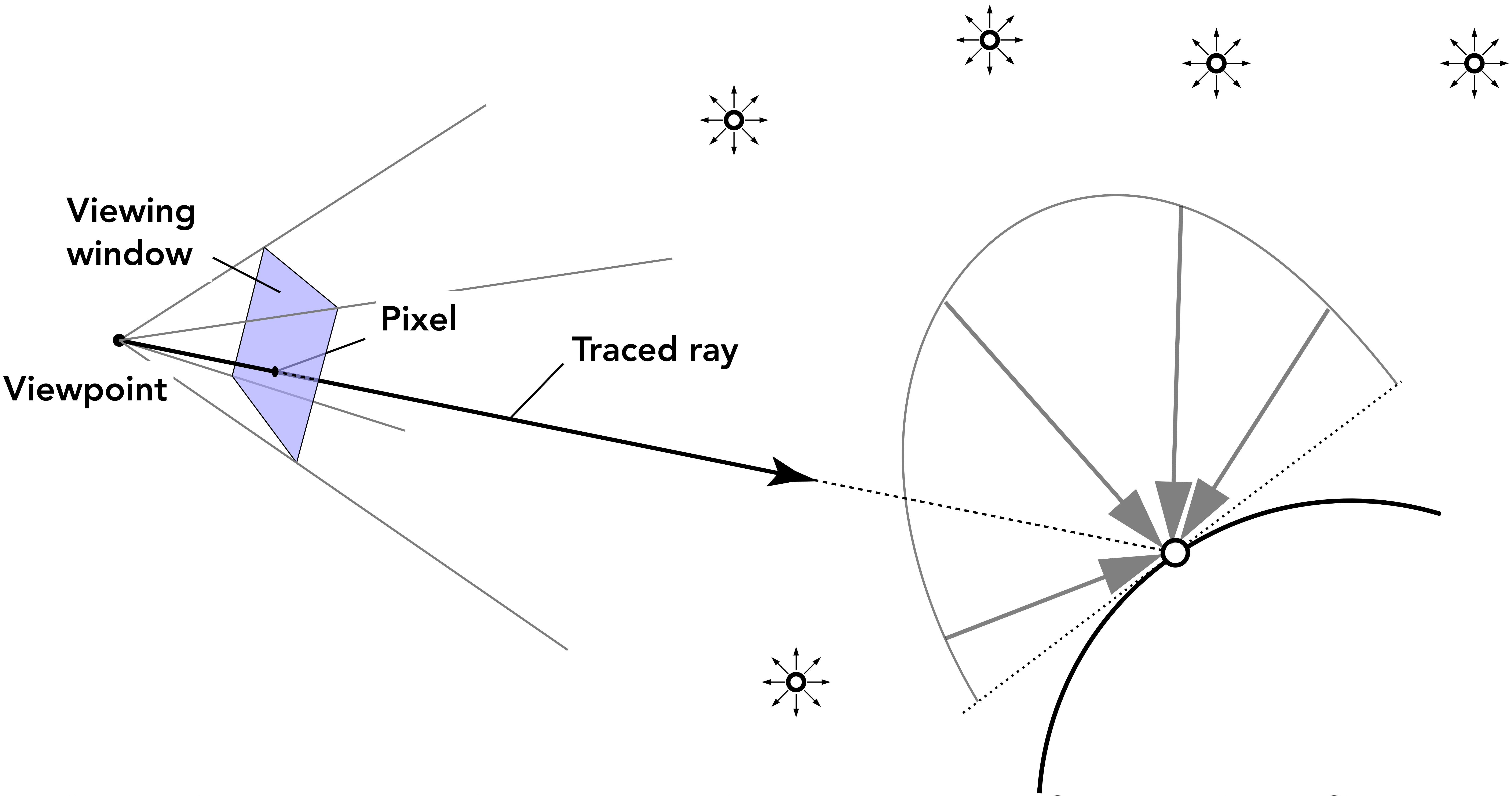
Environment Map Lighting

Visual Richness from Indirect Lighting

Visual Richness from Complex Materials

Credit: Bertrand Benoit. "Sweet Feast," 2009. [Blender /VRay]

Ray Tracer Samples Radiance Along A Ray



The light entering the pixel is the sum total of the light reflected off the surface into the ray's (reverse) direction

Mini-Intro To Material Reflection

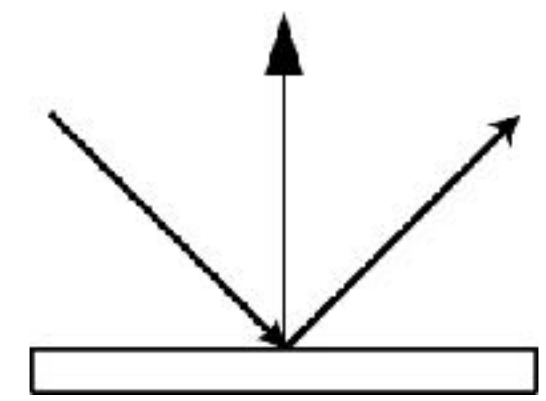
Reflection

Definition: reflection is the process by which light incident on a surface interacts with the surface such that it leaves on the incident (same) side without change in frequency

Categories of Reflection Functions

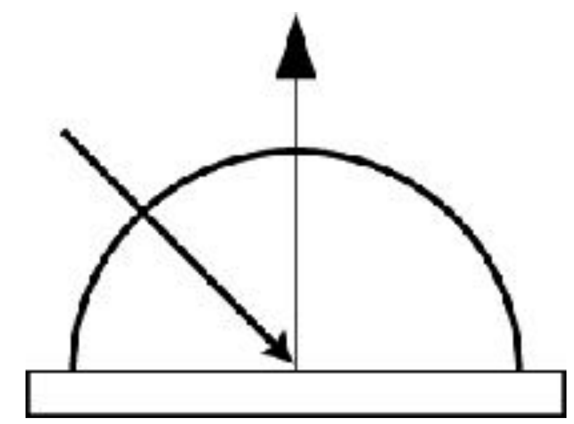
Ideal specular

- Perfect mirror reflection



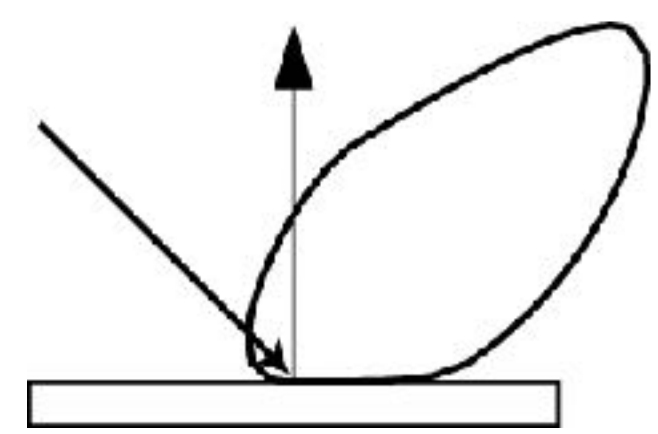
Ideal diffuse

- Equal reflection in all directions



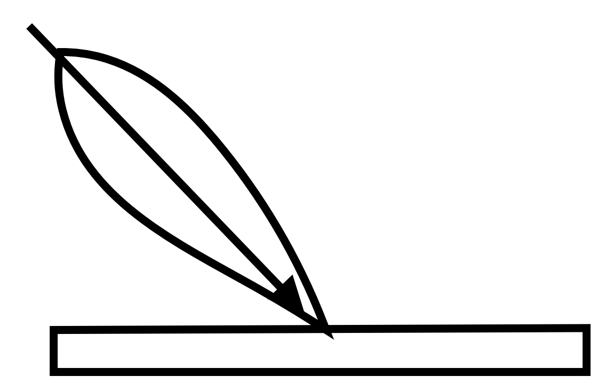

Glossy specular

- Majority of light reflected near mirror direction



Retro-reflective

- Light reflected back towards light source




Diagrams illustrate how light from incoming direction is reflected in various outgoing directions.

Materials: Mirror

Materials: Diffuse

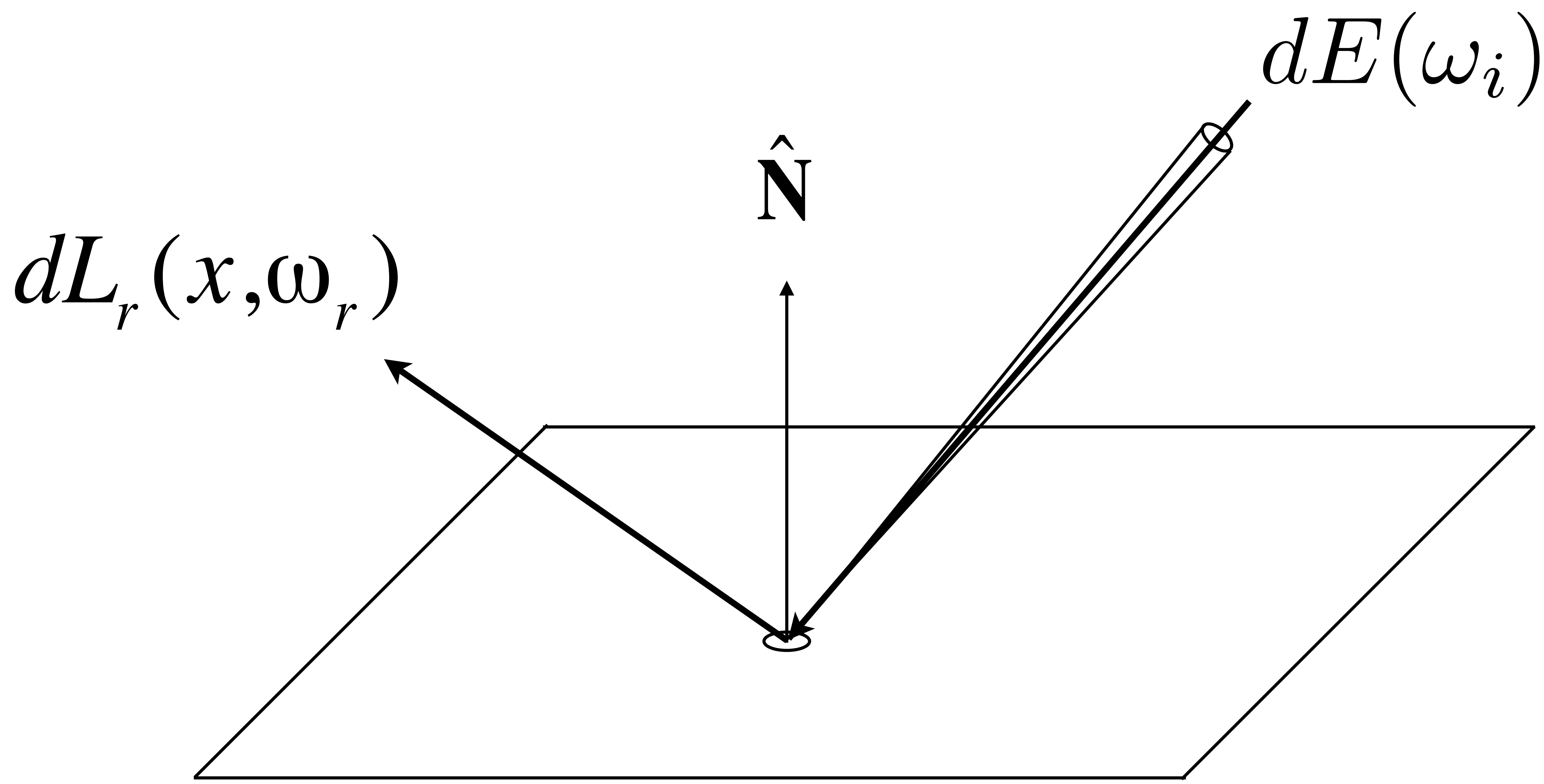
Materials: Gold

Materials: Plastic

Materials: Red Semi-Gloss Paint

Materials: Ford Mystic Lacquer Paint

Reflection at a Point

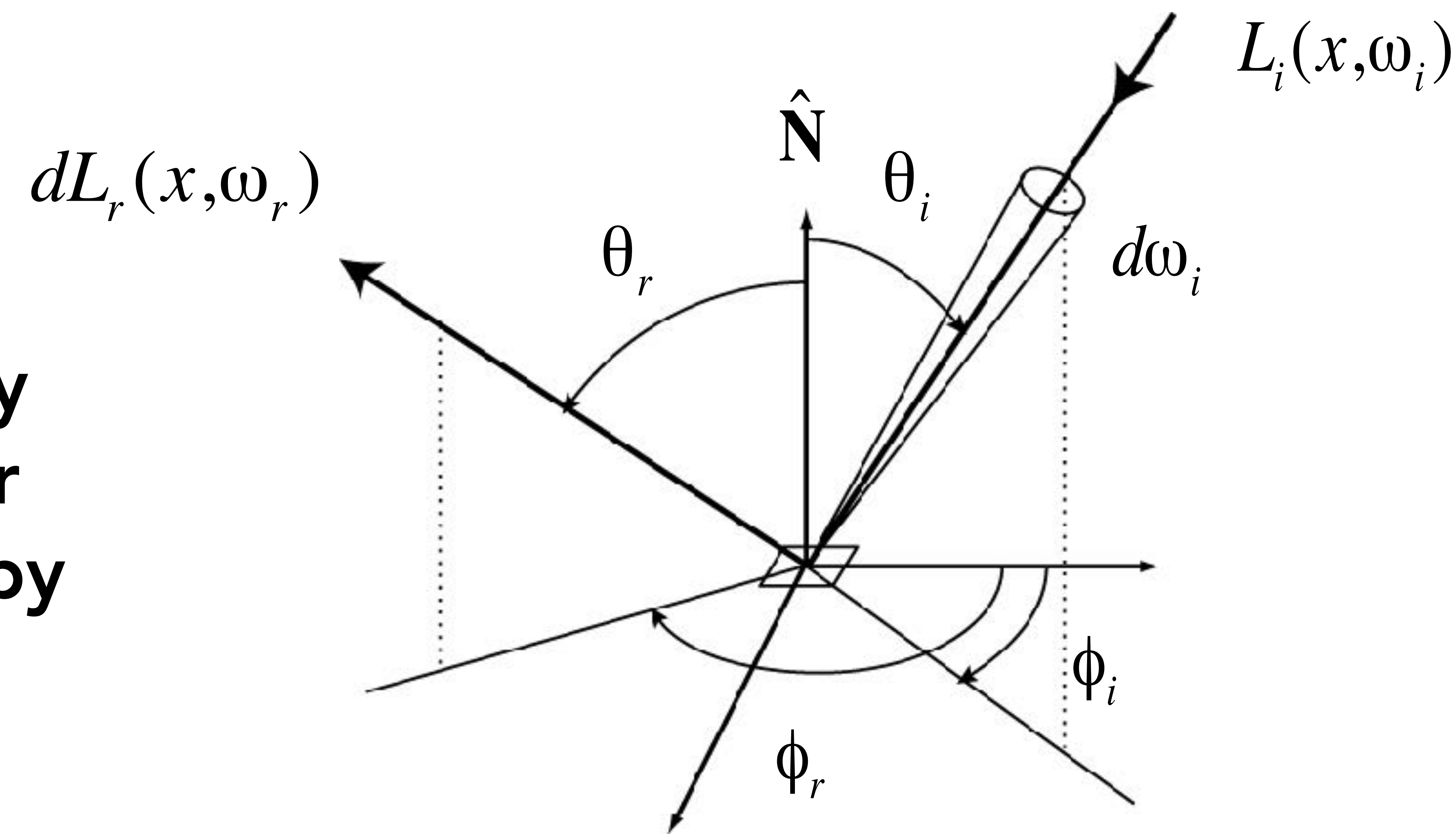


Differential irradiance incoming: $dE(\omega_i) = L(\omega_i)\, cos\, \theta_i \, d\omega_i$

Differential radiance exiting (due to $dE(\omega_i)$) $dL_r(\omega_r)$

BRDF

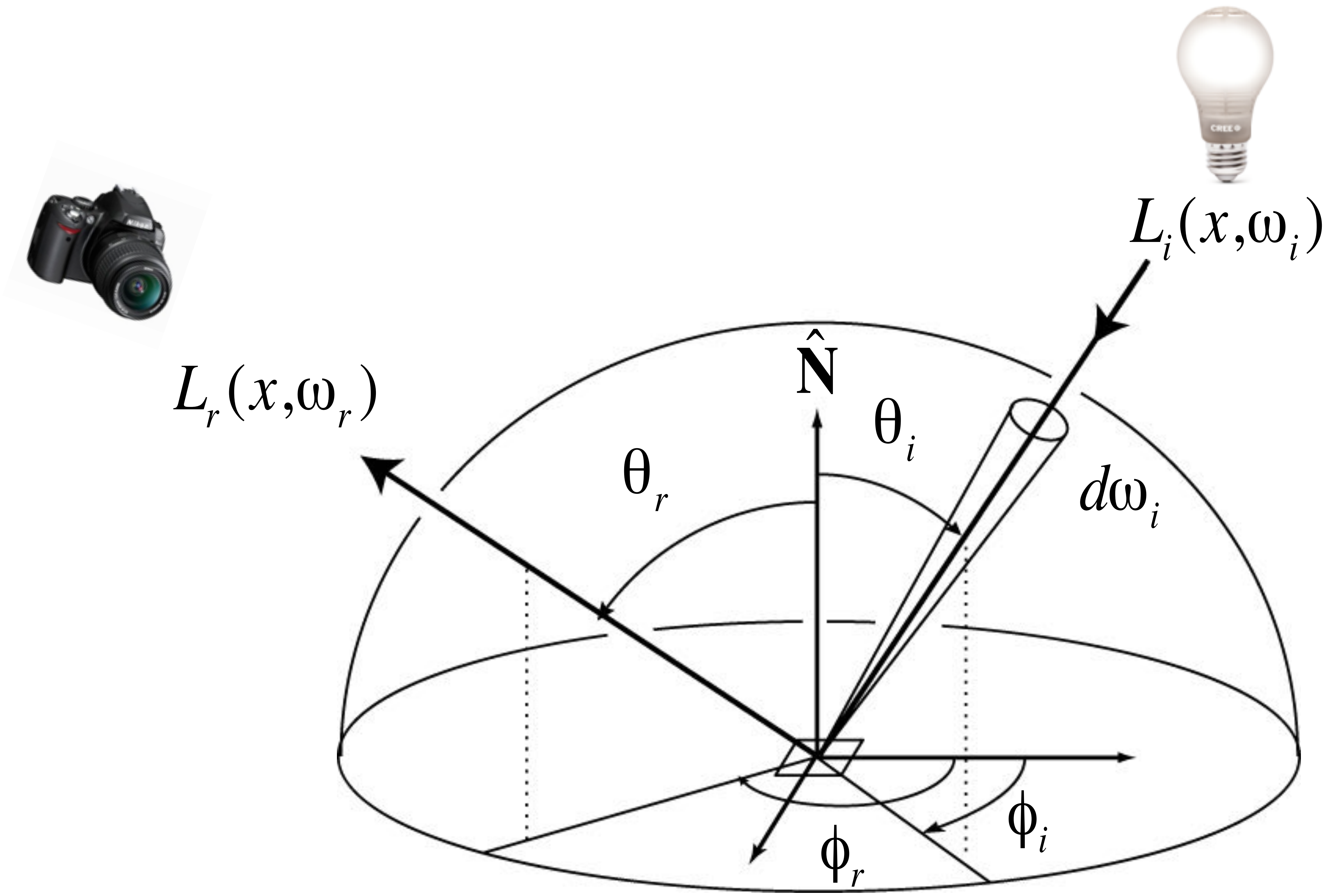
Definition: The bidirectional reflectance distribution function (BRDF) represents how much light is reflected into each outgoing direction ω_r from each incoming direction



NB: ω_i points away from surface rather than into surface, by convention.

$$f_r(\omega_i \to \omega_r) = \frac{\mathrm{d}L_r(\omega_r)}{\mathrm{d}E_i(\omega_i)} = \frac{\mathrm{d}L_r(\omega_r)}{L_i(\omega_i)\cos\theta_i\,\mathrm{d}\omega_i} \quad \left[\frac{1}{\mathrm{sr}}\right]$$

The Reflection Equation



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

Solving the Reflection Equation

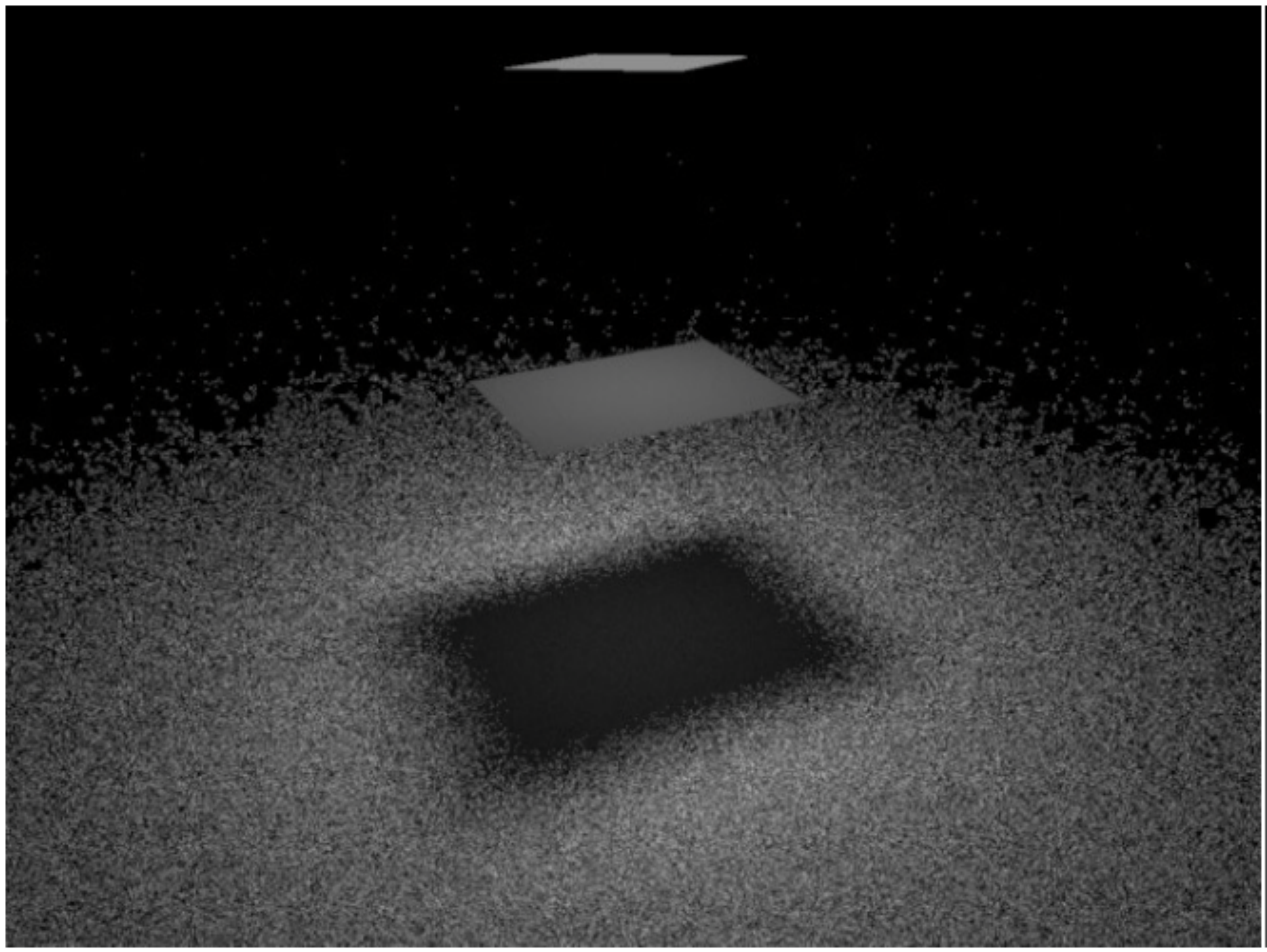
$$L_r(\mathrm{p}, \omega_r) = \int_{H^2} f_r(\mathrm{p}, \omega_i \to \omega_r)\, L_i(\mathrm{p}, \omega_i)\, \cos\theta_i \,\mathrm{d}\omega_i$$

Monte Carlo estimate:

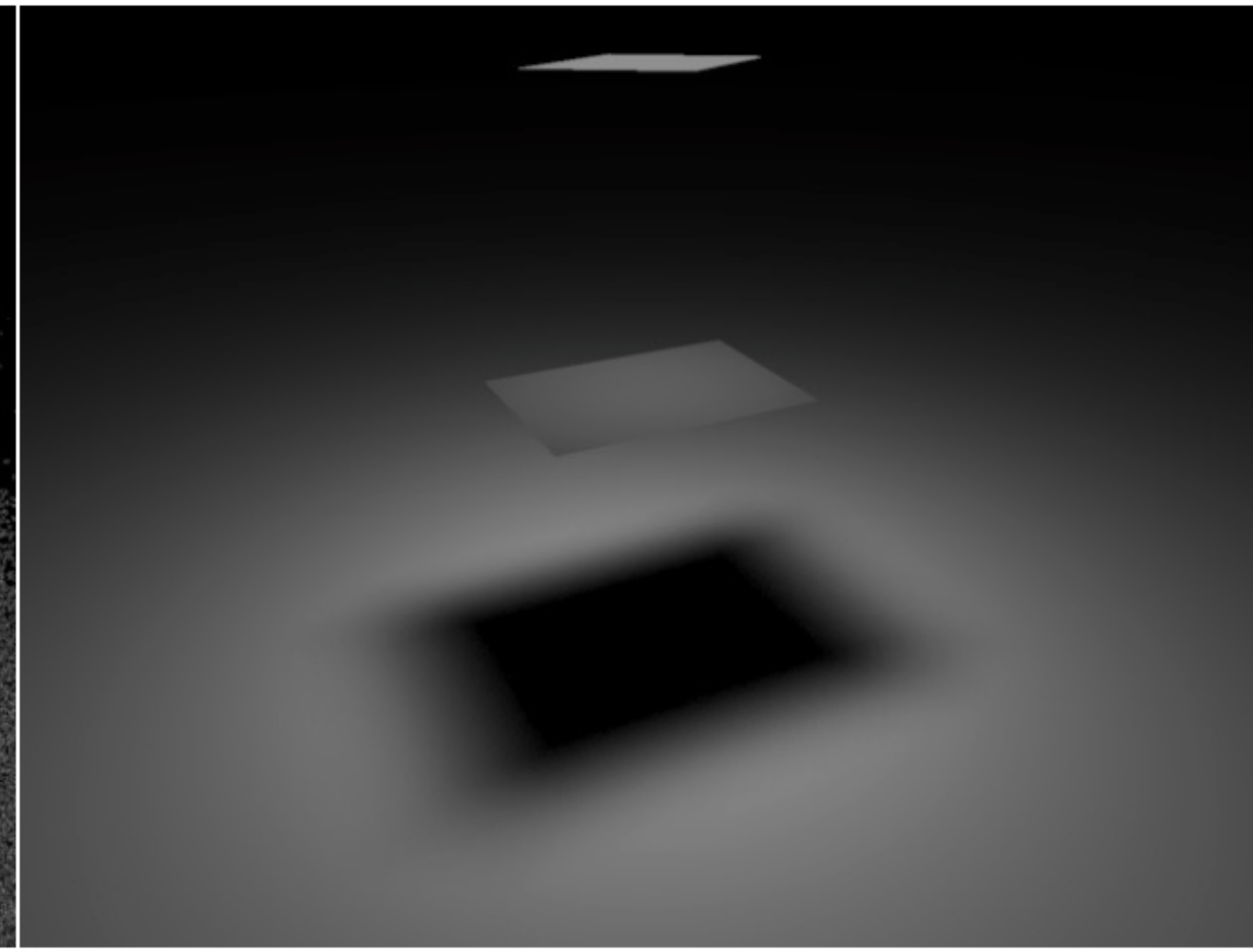
- **Generate directions ω_j sampled from some distribution $p(\omega)$**
- **Choices for $p(\omega)$**
  - **Uniformly sample hemisphere**
  - **Importance sample BRDF (proportional to BRDF)**
  - **Importance sample lights (sample position on lights)**
- **Compute the estimator**

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

Recall: Hemisphere vs Light Sampling



Sample hemisphere uniformly



Sample points on light

Direct Lighting Pseudocode (Uniform Random Sampling)

```
DirectLightingSampleUniform(x, wo)
   wi = hemisphere.sampleUniform();   // uniform random sampling
   pdf = 1.0 / (2 * pi);

   if (scene.shadowIntersection(x, wi))       // Shadow ray
      return 0;
   else
      L = lights.radiance(intersect(x,wi), -wi);
      return L * x.brdf(wi, wo) * costheta / pdf;
```

Direct Lighting Pseudocode (Importance Sampling of BRDF)

```
DirectLightingSampleBRDF(x, wo)
  wi, pdf = x.brdf.sampleDirection();        // Imp. Sample BRDF

  if (scene.shadowIntersection(x, wi))       // Shadow ray
    return 0;
  else
    L = lights.radiance(intersect(x, wi), -wi);
    return L * x.brdf(wi, wo) * costheta / pdf;
```

Direct Lighting Pseudocode (Importance Sampling of Lights)

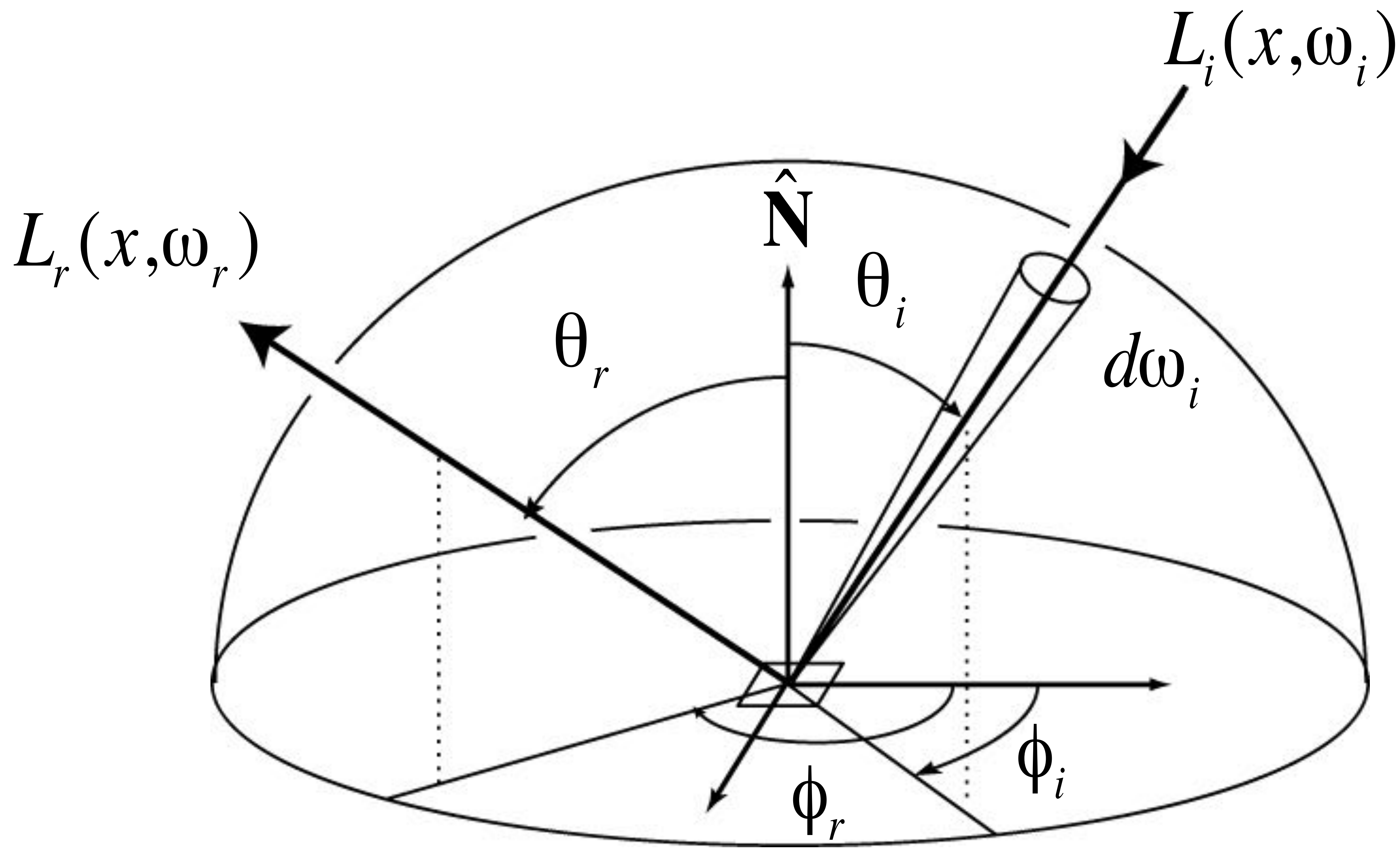
```
DirectLightingSampleLights(x, wo)
   L, wi, pdf = lights.sampleDirection(x);   // Imp. sampl lights

   if (scene.shadowIntersection(x, wi))      // Shadow ray
      return 0;
   else
      return L * x.brdf(wi, wo) *costheta / pdf;

// Note: only one random sample over all lights.
// Assignment 3-1 asks you to, alternatively, loop over
// multiple lights and take multiple samples
```

Global Illumination:
Deriving the Rendering Equation

Recall: Reflection Equation

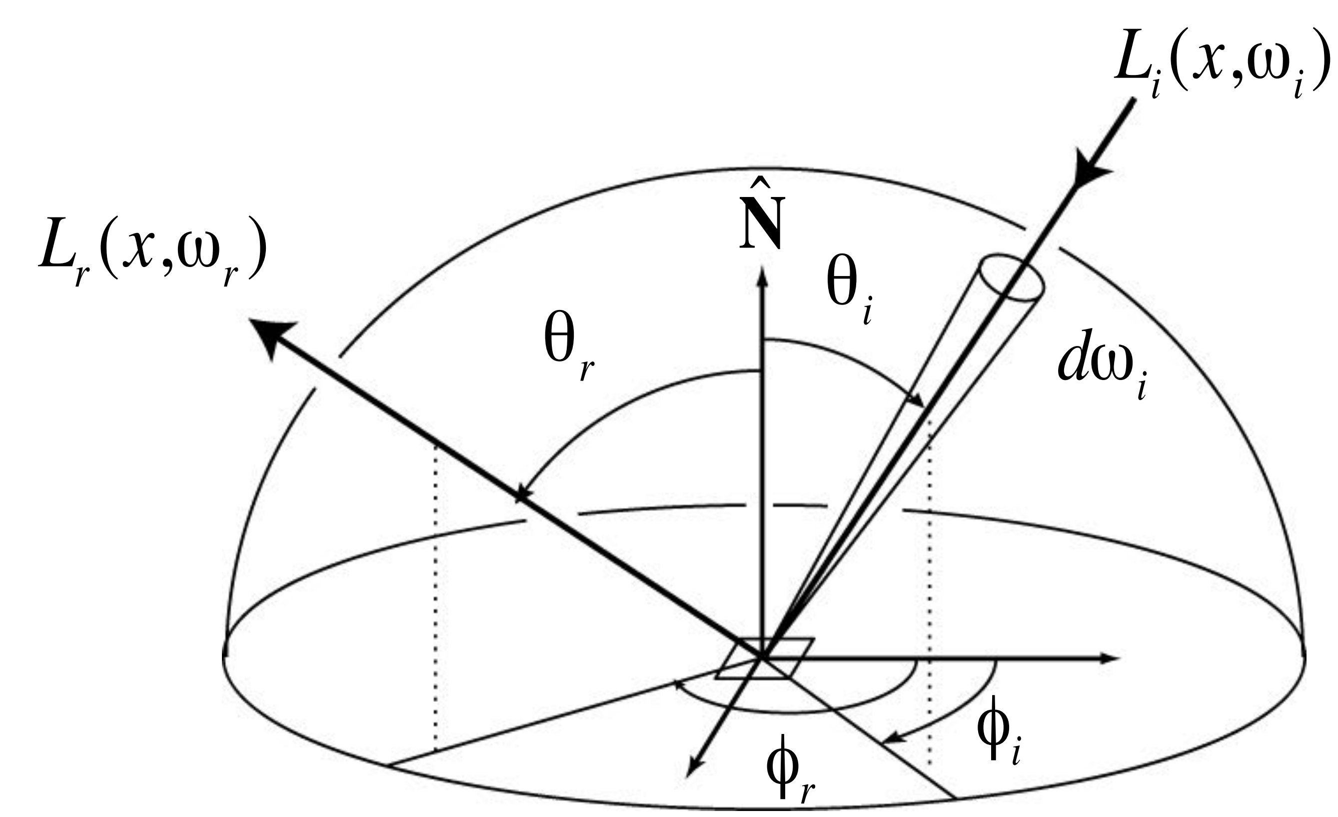


$$L_r(\mathrm{p}, \omega_r) = \int_{H^2} f_r(\mathrm{p}, \omega_i \rightarrow \omega_r)\, L_i(\mathrm{p}, \omega_i)\, \cos\theta_i\, \mathrm{d}\omega_i$$

Challenge: This is Actually A Recursive Equation

Reflected radiance depends on incoming radiance

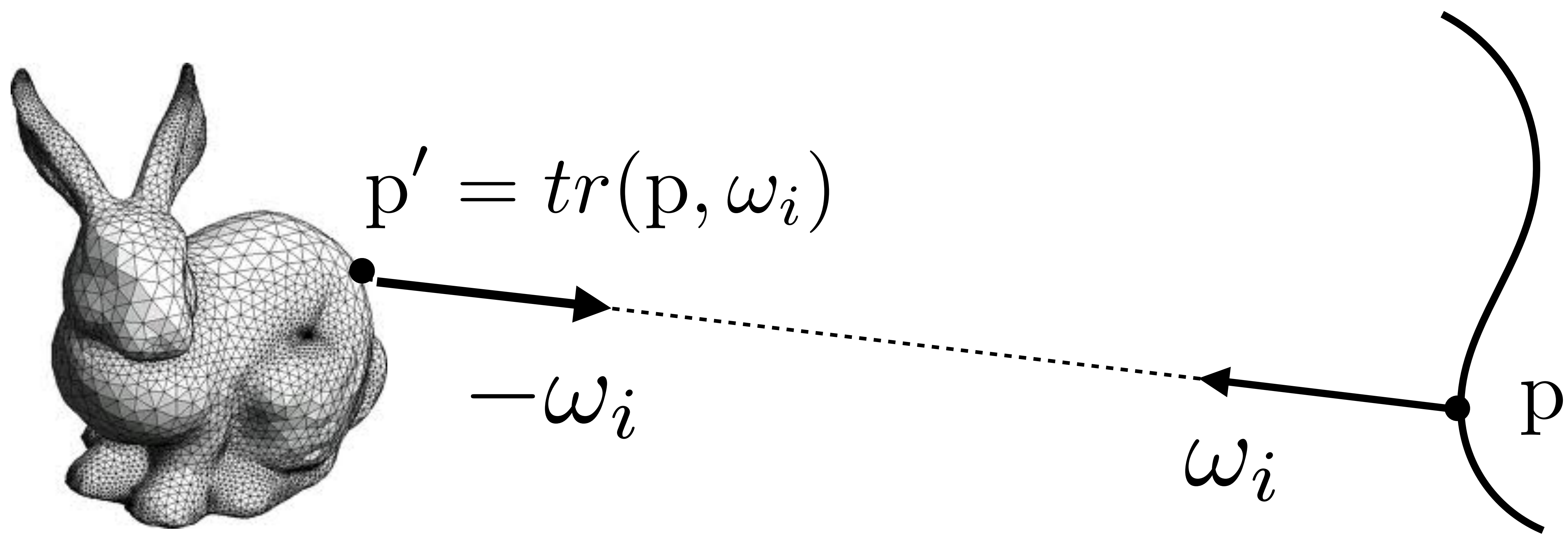
$$\boxed{L_r(\mathrm{p}, \omega_r)} = \int_{H^2} f_r(\mathrm{p}, \omega_i \to \omega_r) \boxed{L_i(\mathrm{p}, \omega_i)} \cos\theta_i \, \mathrm{d}\omega_i$$



But incoming radiance depends on reflected radiance (at another point in the scene)

Transport Function & Radiance Invariance

Definition: the Transport Function, $tr(\mathrm{p}, \omega)$, returns the first surface intersection point in the scene along ray (p, ω)



Radiance invariance along rays: $L_o(tr(\mathrm{p}, \omega_i), -\omega_i) = L_i(\mathrm{p}, \omega_i)$

"Radiance arriving at p from direction ω_i is equal to the radiance leaving p' in direction $-\omega_i$"

The Rendering Equation

Re-write the reflection equation:

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

Using the transport function: $L_i(\mathrm{p}, \omega_i) = L_o(tr(\mathrm{p}, \omega_i), -\omega_i)$

The Rendering Equation

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

Note: recursion is now explicit

How to solve?

Light Transport Operators

Operators Are Higher-Order Functions

Functions:

$$f, g : (x, \omega) \to \mathbb{R}$$

Operators are higher-order functions:

$$P : ((x, \omega) \to \mathbb{R}) \to ((x, \omega) \to \mathbb{R})$$

$$P(f) = g$$

- Take a function and transform it into another function

Linear Operators

- Linear operators act on functions like matrices act on vectors

$$h(x) = (L(f))(x)$$

- They are linear in that:

$$L(af + bg) = aL(f) + bL(g)$$

- Examples of linear operators:

$$H(f)(x) = \int h(x, x')\, f(x')\, \mathrm{d}x'$$

$$D(f)(x) = \frac{\delta f}{\delta x}(x)$$

Light Transport Functions & Operators

- Emitted radiance function (all surface points & outgoing directions) $L_e(\mathrm{p}, \omega)$

- Incoming/outgoing reflected radiance (all surface points & in/out directions) $L_i(\mathrm{p}, \omega), \;\; L_o(\mathrm{p}, \omega)$

- Transport function - returns the first scene intersection point along given ray $tr(\mathrm{p}, \omega)$

- Reflection operator:

$$R(g)(\mathrm{p}, \omega_o) \equiv \int_{H^2} f_r(\mathrm{p}, \omega_i \rightarrow \omega_o) \, g(\mathrm{p}, \omega_i) \; \cos\theta_i \, \mathrm{d}\omega_i$$

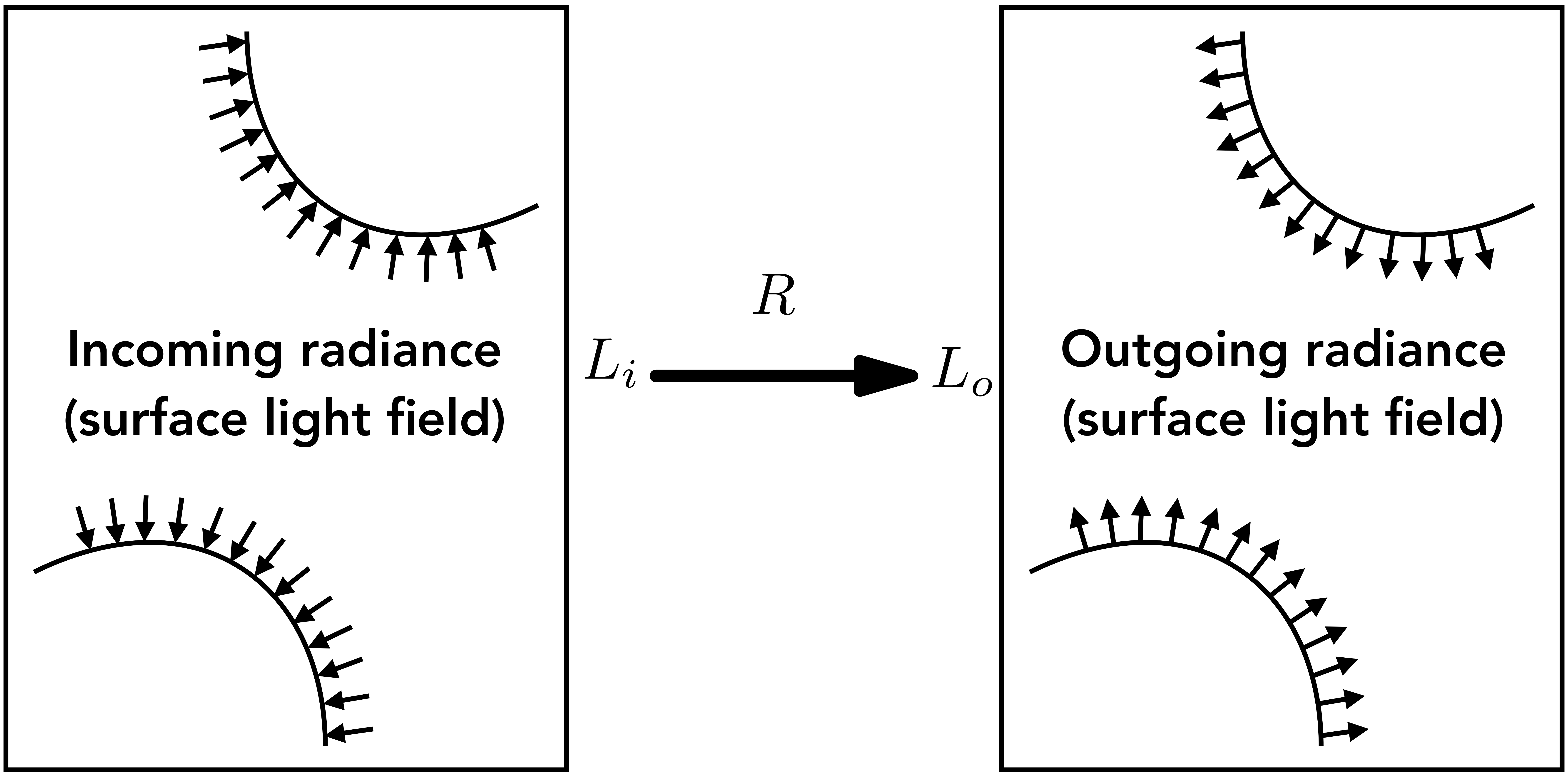
$$R(L_i) = L_o$$

- Transport operator:

$$T(f)(\mathrm{p}, \omega_o) \equiv f(tr(\mathrm{p}, \omega), -\omega)$$

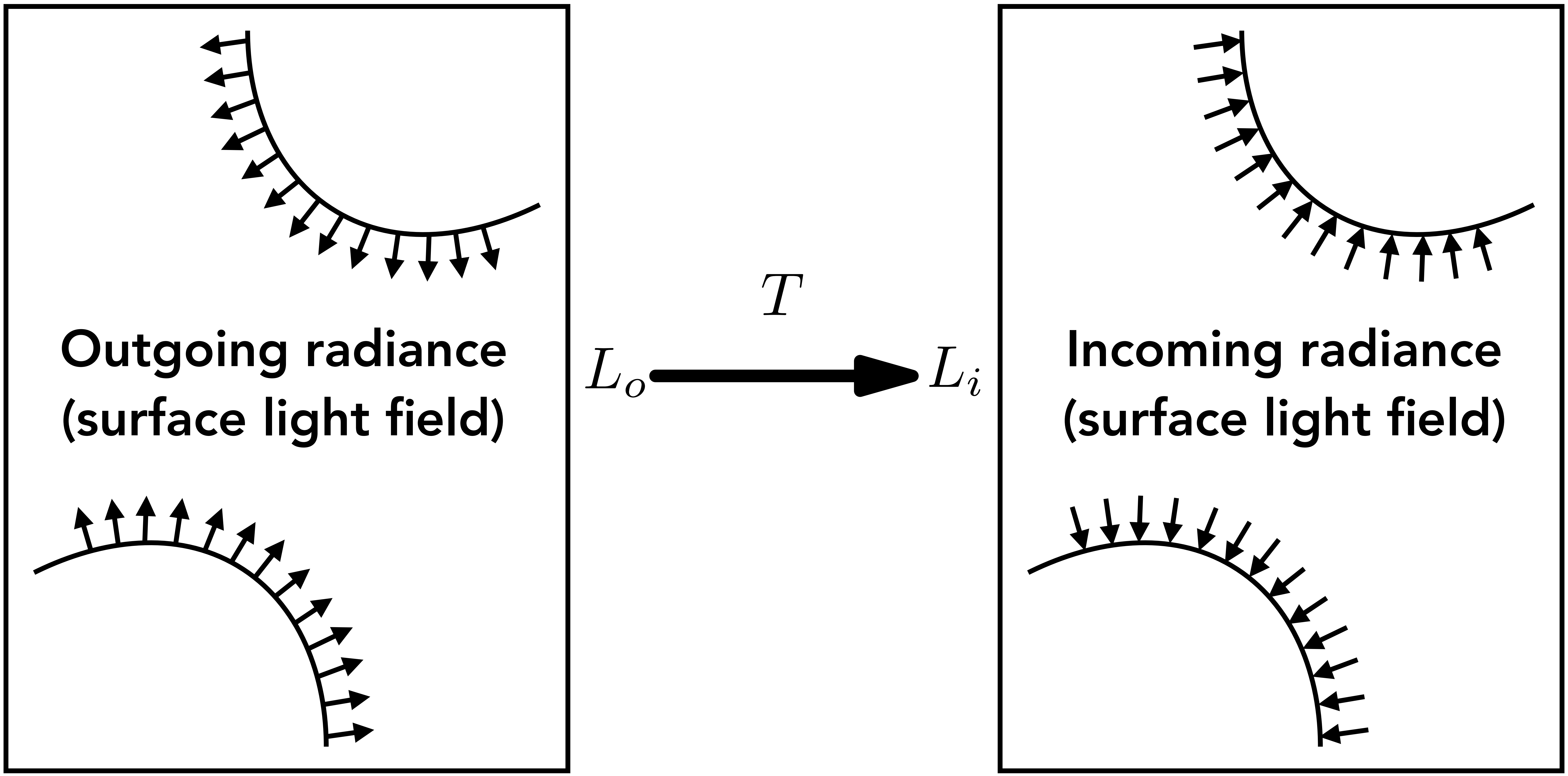
$$T(L_o) = L_i$$

Reflection Operator



$$R(g)(\mathrm{p}, \omega_o) \equiv \int_{H^2} f_r(\mathrm{p}, \omega_i \to \omega_o)\, g(\mathrm{p}, \omega_i)\, \cos\theta_i \,\mathrm{d}\omega_i$$

Transport Operator



$$T(f)(\mathrm{p}, \omega_o) \equiv f(tr(\mathrm{p}, \omega_o), -\omega_o)$$

$$T(L_o) = L_i$$

Rendering Equation in Operator Notation

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

$$L_o = L_e + (R \circ T)(L_o)$$

Define full one-bounce light transport operator: $K = R \circ T$

$$\boxed{L_o = L_e + K(L_o)}$$

Solving the Rendering Equation

Solving the Rendering Equation

- Rendering equation:

$$L = L_e + K(L)$$

$$(I - K)(L) = L_e$$

***L* is outgoing reflected**

- Solution desired:

$$L = (I - K)^{-1}(L_e)$$

- How to solve?

Solution Intuition

For scalar functions, recall:

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + \cdots$$

$$\text{converges for } -1 < x < 1$$

Similarly, for operators, it is true that

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

(Neumann series)

$$\text{converges for } ||K|| < 1$$

where $||K|| < 1$ means that the "energy" of the radiance function decreases after applying K. This is intuitively true for valid scene models based on energy dissipation (though not trivial to prove, see Veach & Guibas).

Formal Solution

Neumann series:

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

Check:

$$\begin{aligned}
&(I-K) \circ (I-K)^{-1} \\
&= (I-K) \circ (I + K + K^2 + K^3 + \cdots) \\
&= (I + K + K^2 + \cdots) - (K + K^2 + \cdots) \\
&= I
\end{aligned}$$

Again, energy dissipation makes it possible to show that the series converges.

Rendering Equation Solution

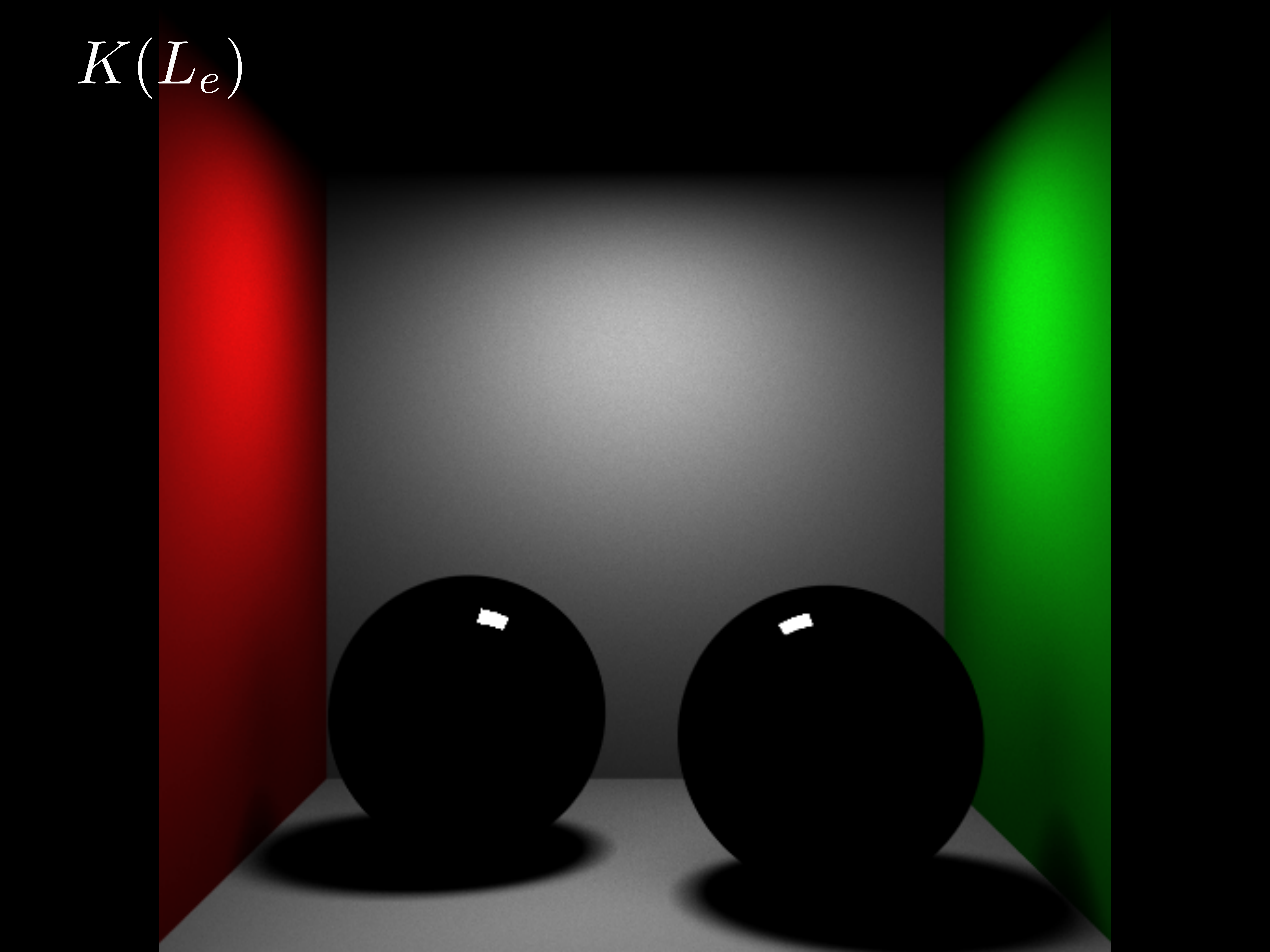
$$
\begin{aligned}
L &= (I - K)^{-1}(L_e) \\
&= (I + K + K^2 + K^3 + \cdots)(L_e) \\
&= L_e + K(L_e) + K^2(L_e) + K^3(L_e) + \cdots
\end{aligned}
$$

Emitted **1-bounce** **2-bounce** **3-bounce**

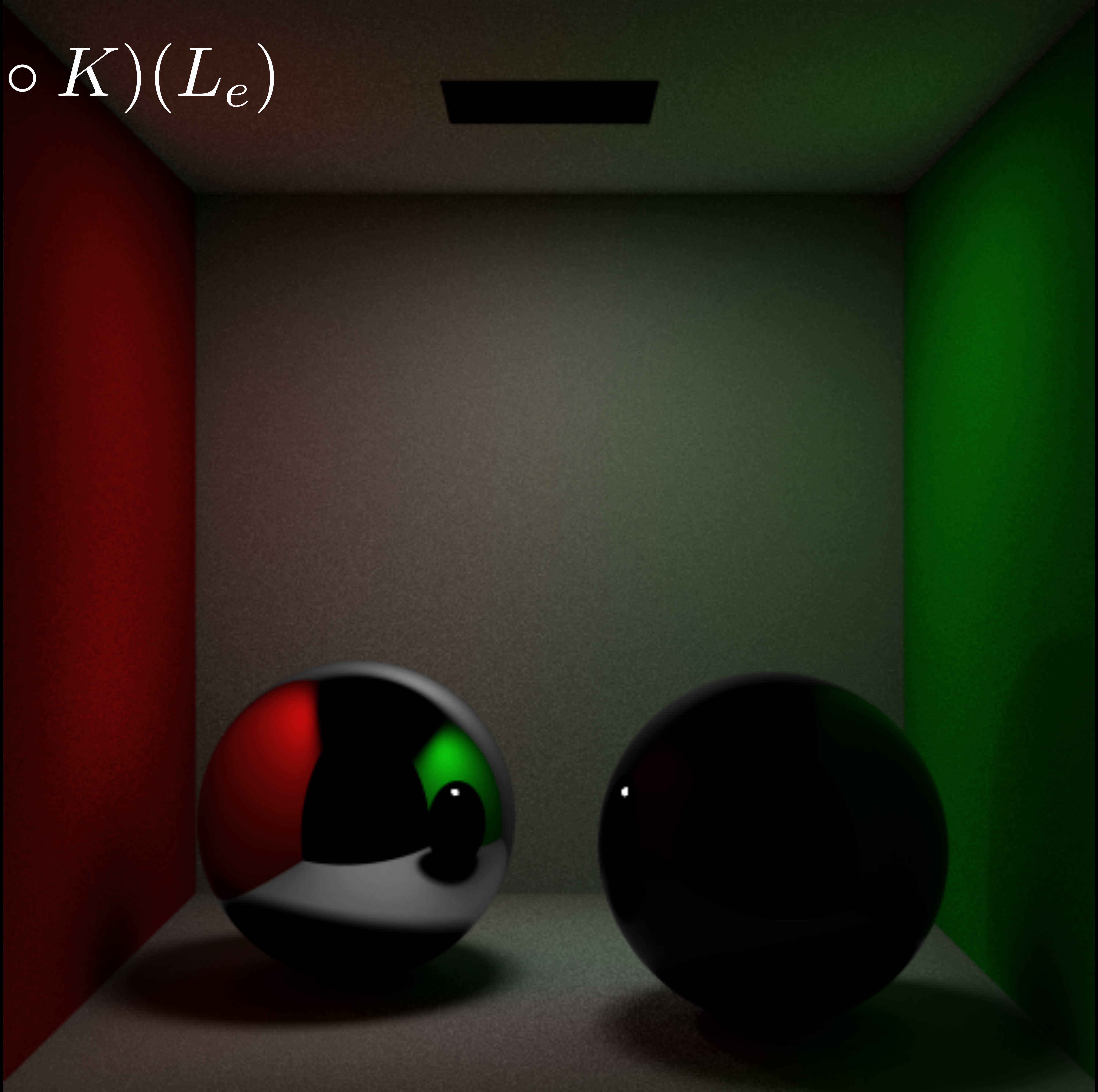
Intuitive: Sum of successive bounces of light

This calculates the steady-state surface light field over the scene.

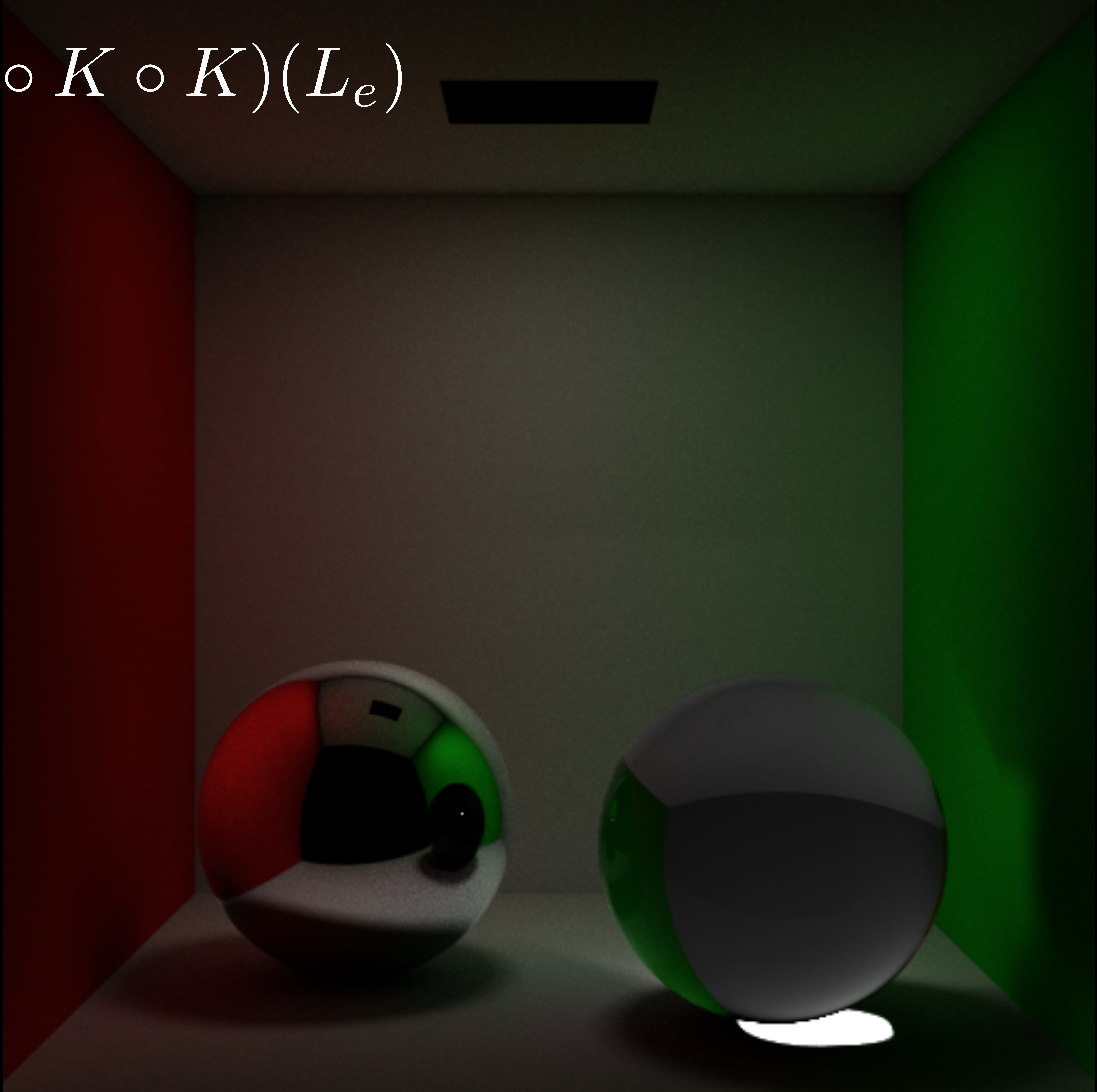
L_e


$K(L_e)$

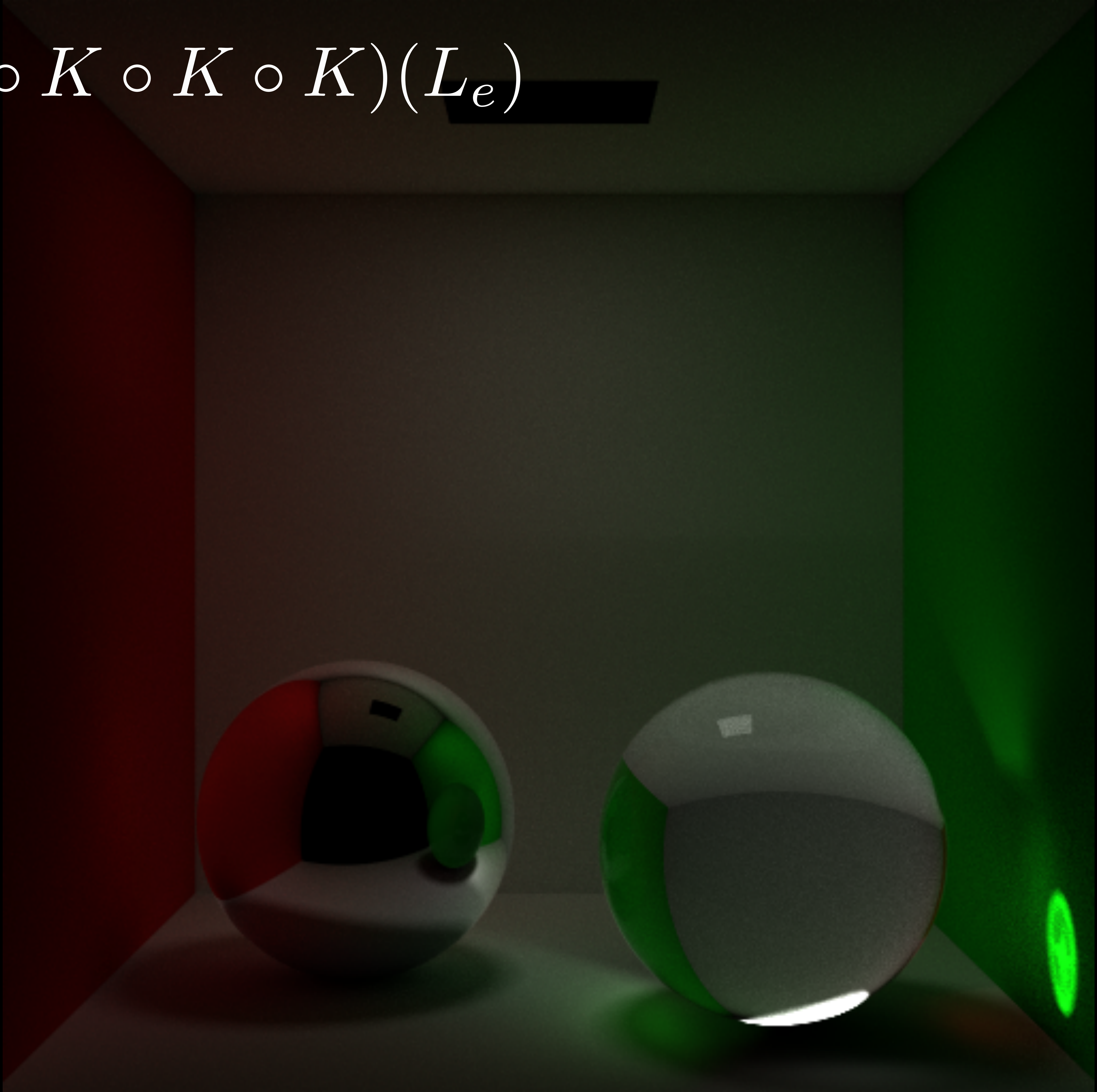
$(K \circ K)(L_e)$



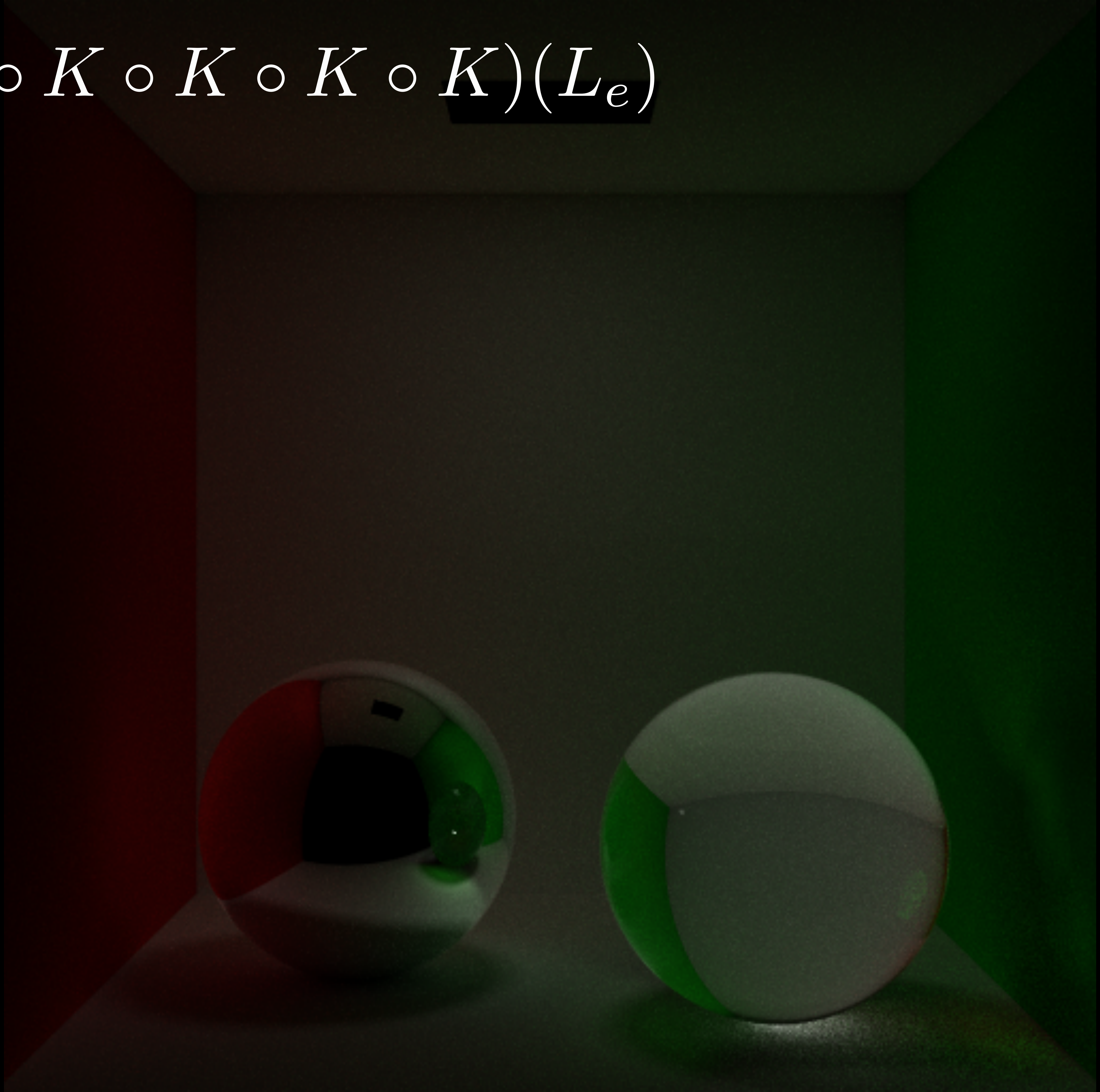
$$(K \circ K \circ K)(L_e)$$



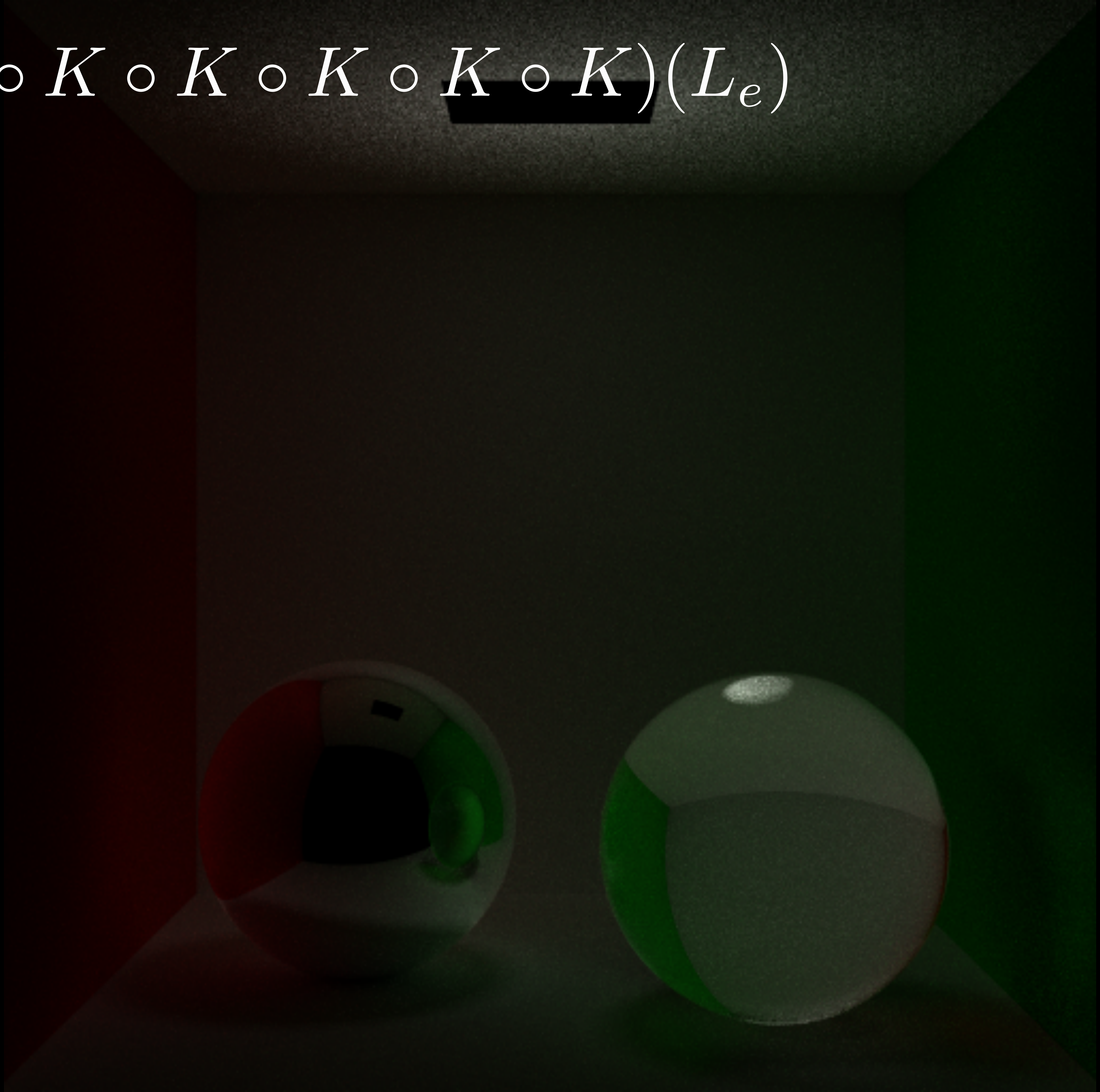
$$(K \circ K \circ K \circ K)(L_e)$$



$$(K \circ K \circ K \circ K \circ K)(L_e)$$

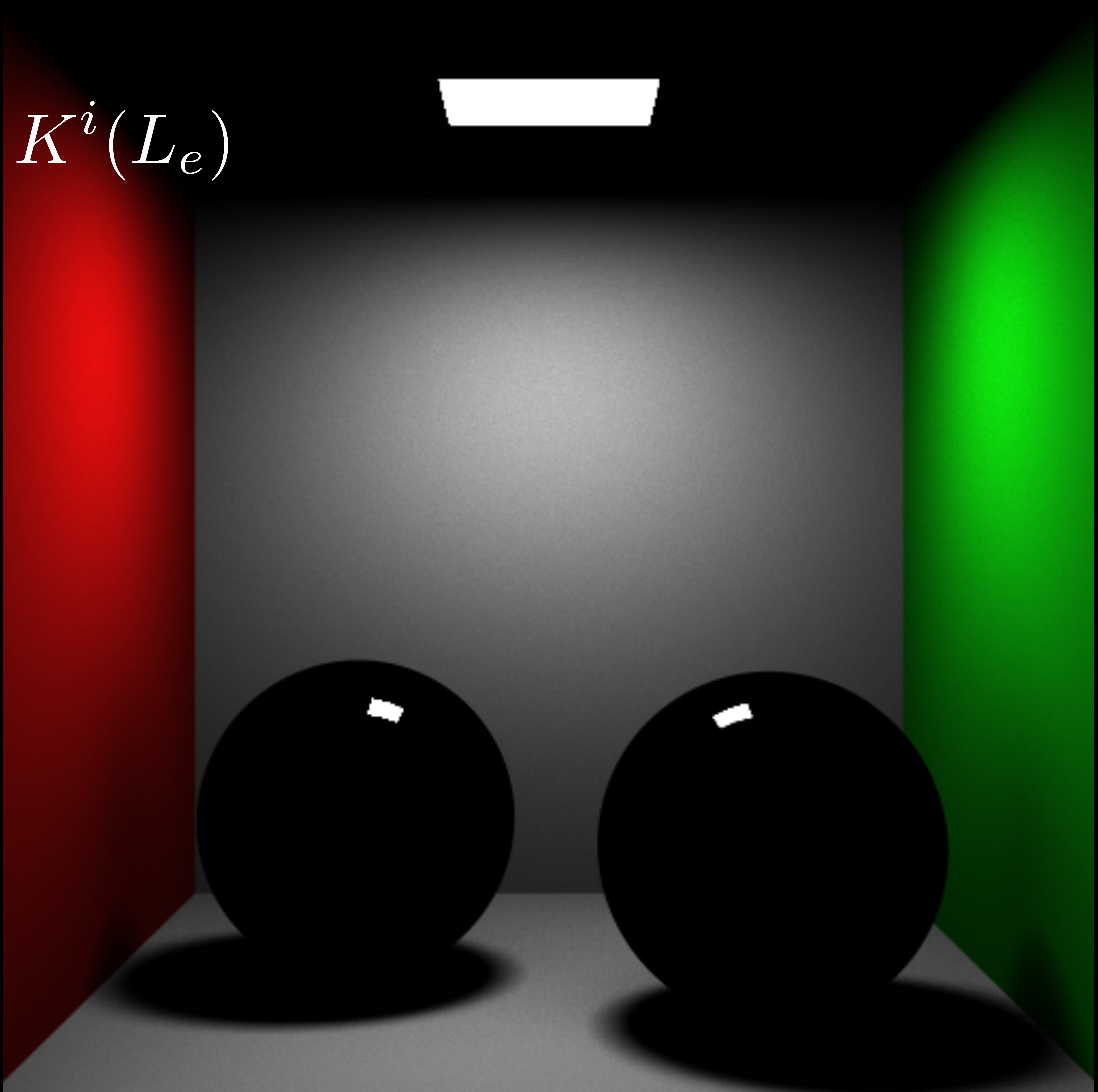


$$(K \circ K \circ K \circ K \circ K \circ K)(L_e)$$

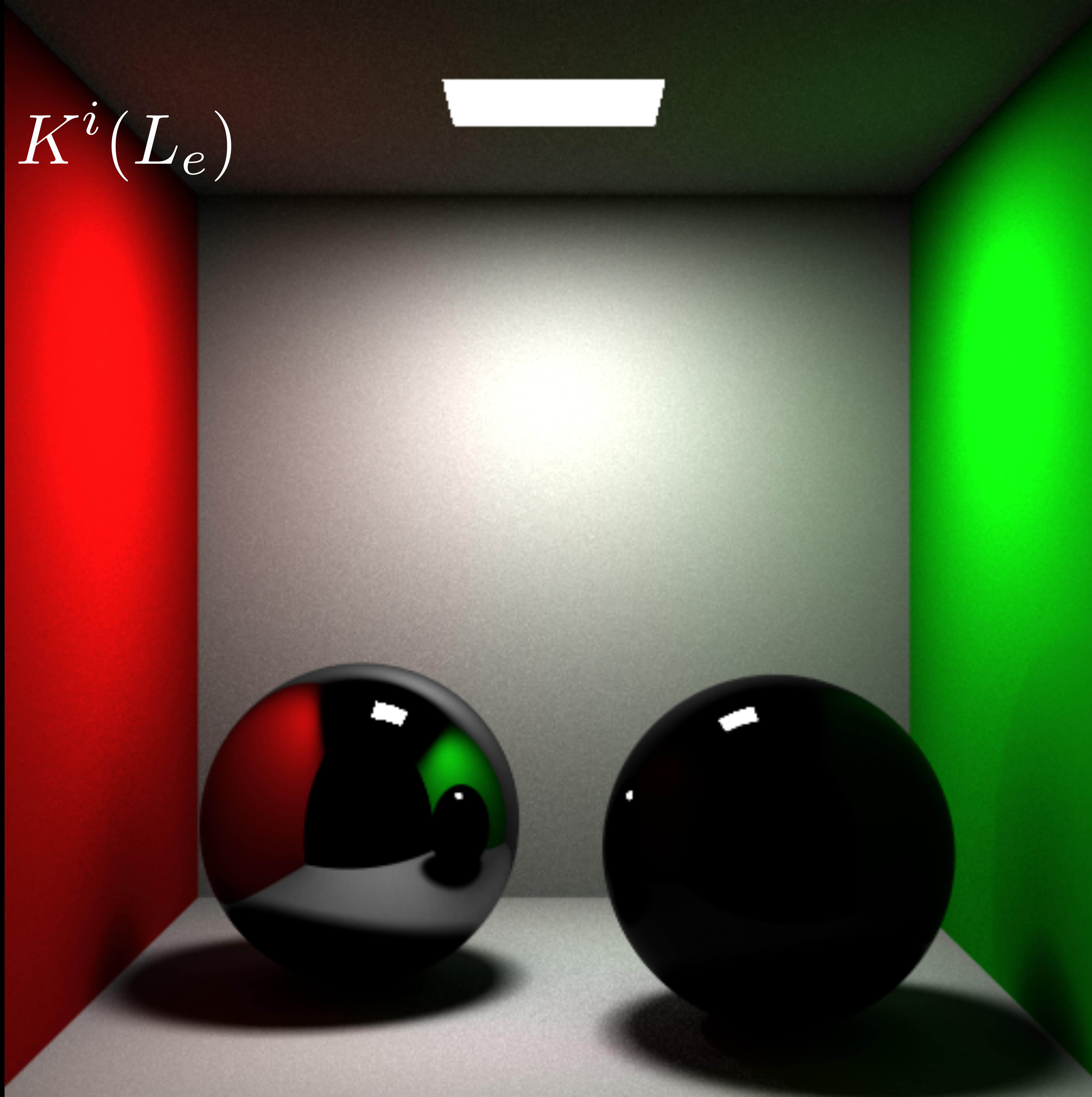


$$\sum_{i=0}^{0} K^i(L_e)$$

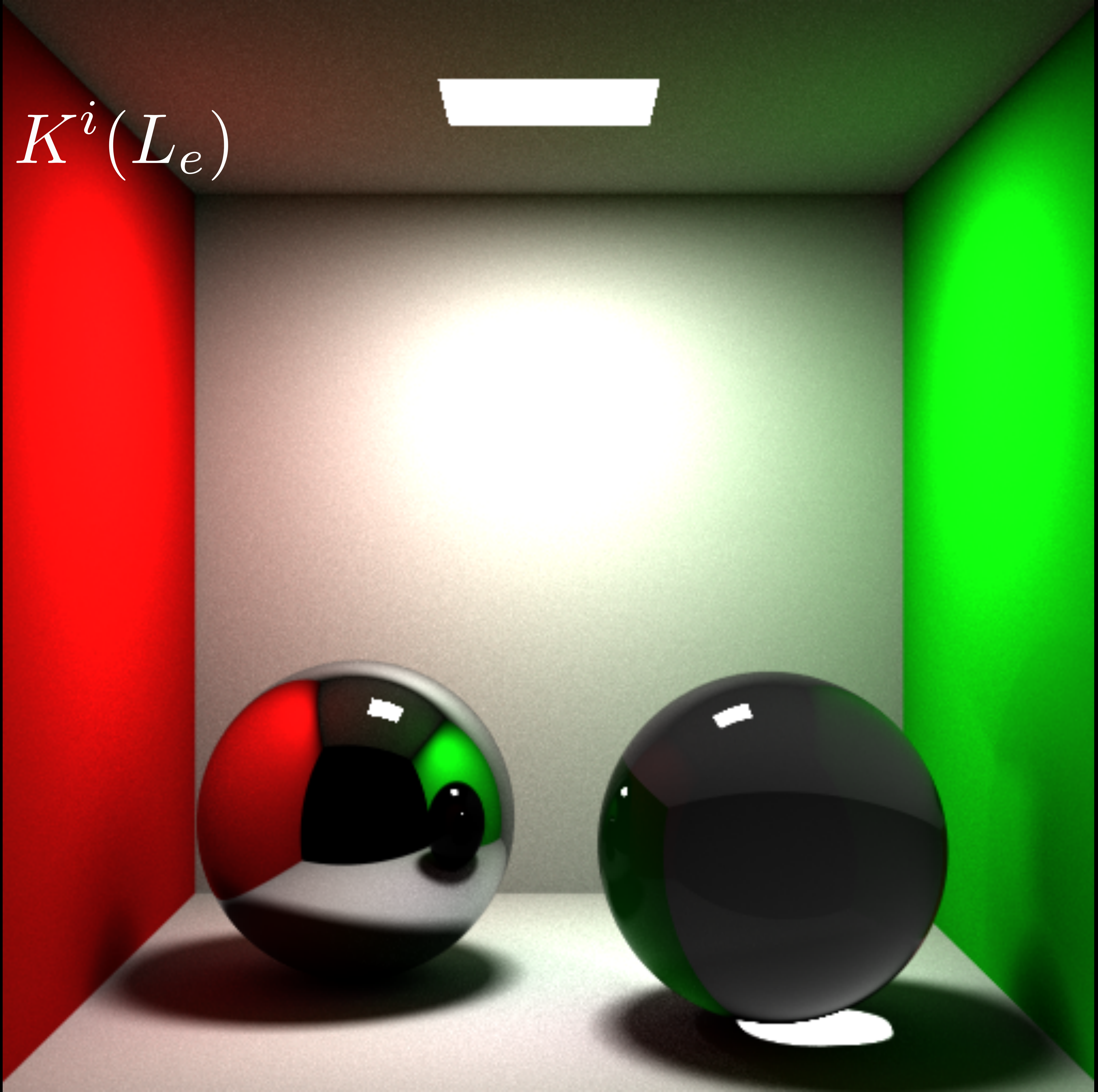
$$\sum_{i=0}^{1} K^i(L_e)$$



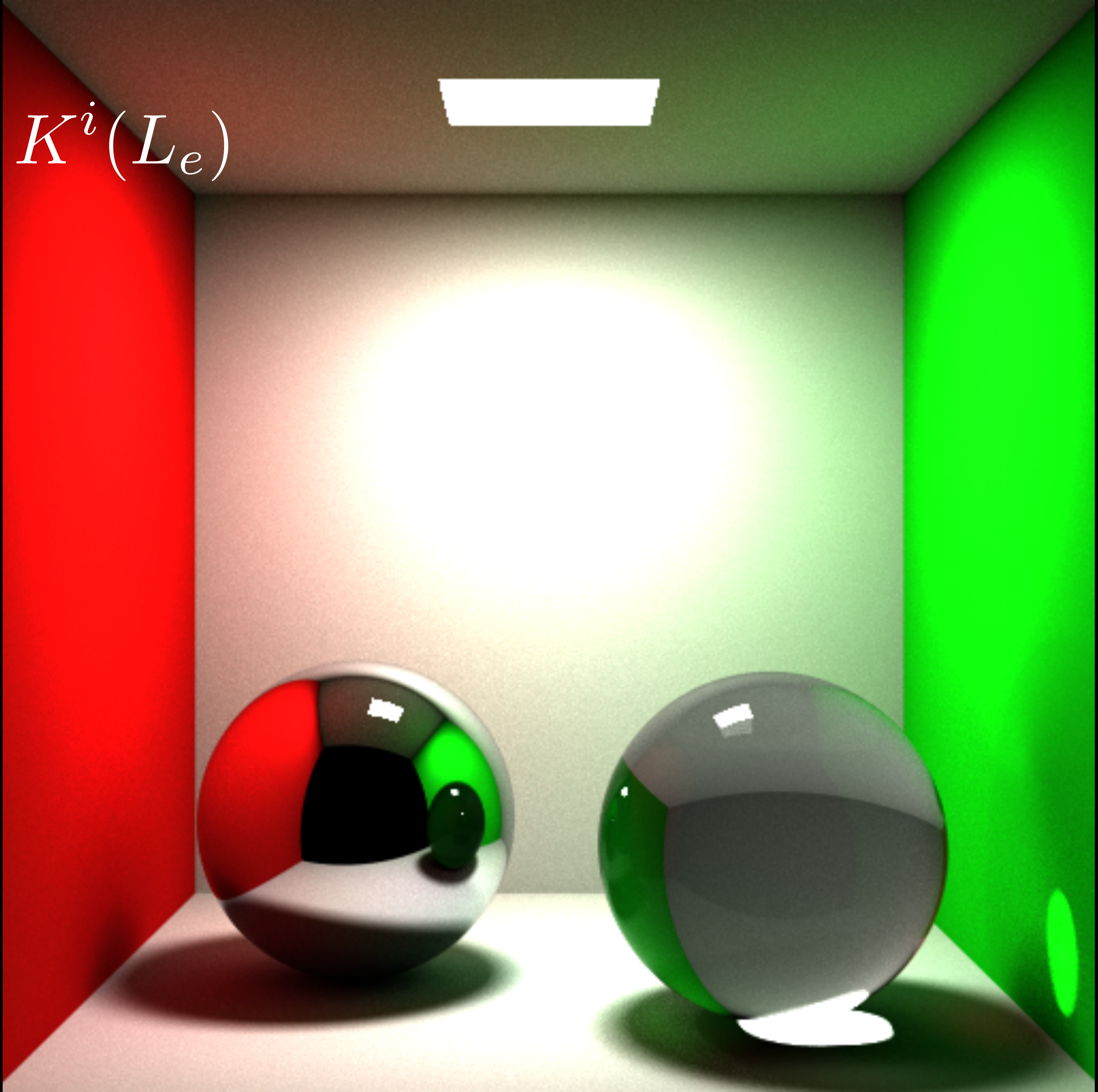
$$\sum_{i=0}^{2} K^i(L_e)$$



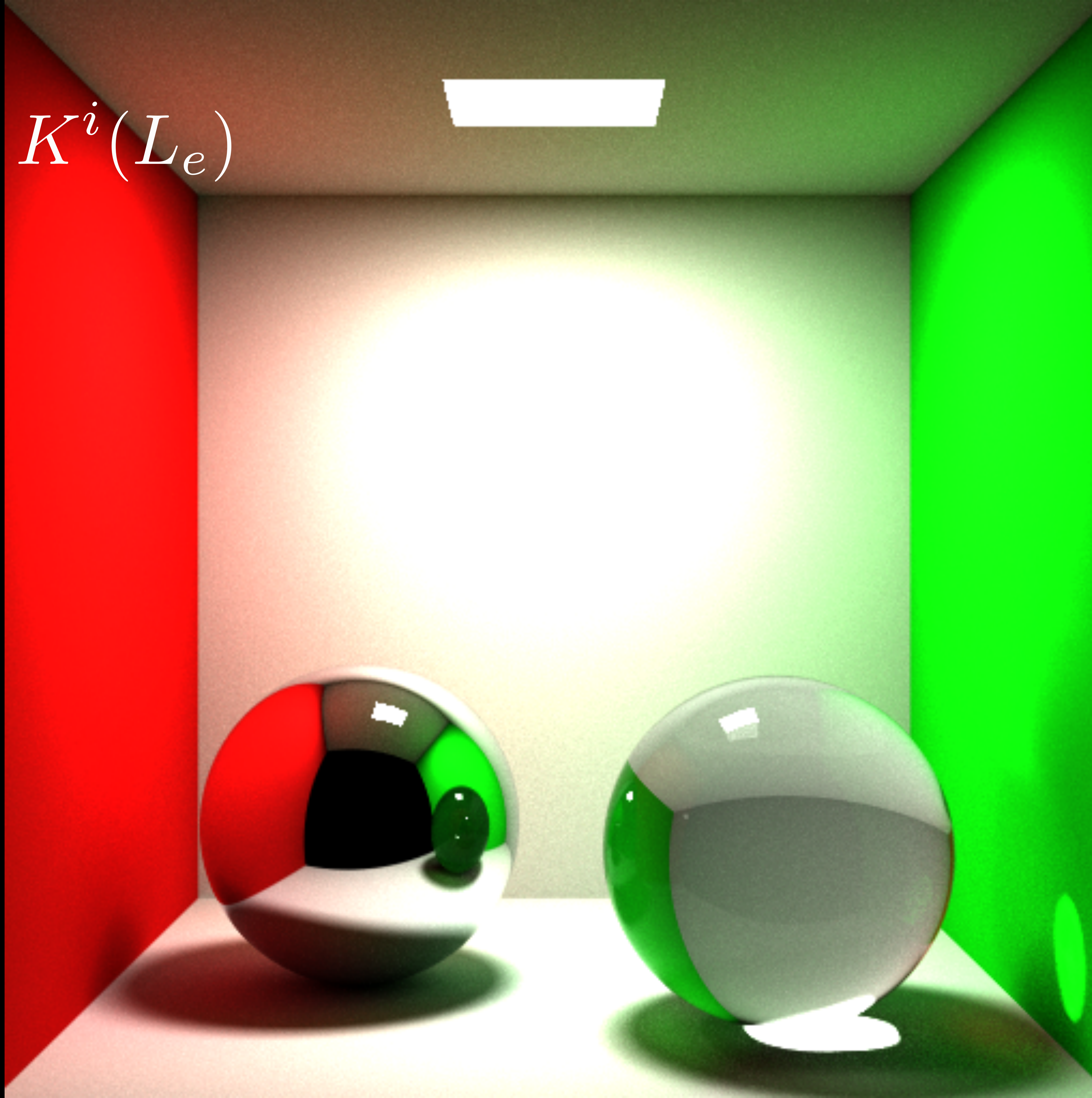
$$\sum_{i=0}^{3} K^i(L_e)$$



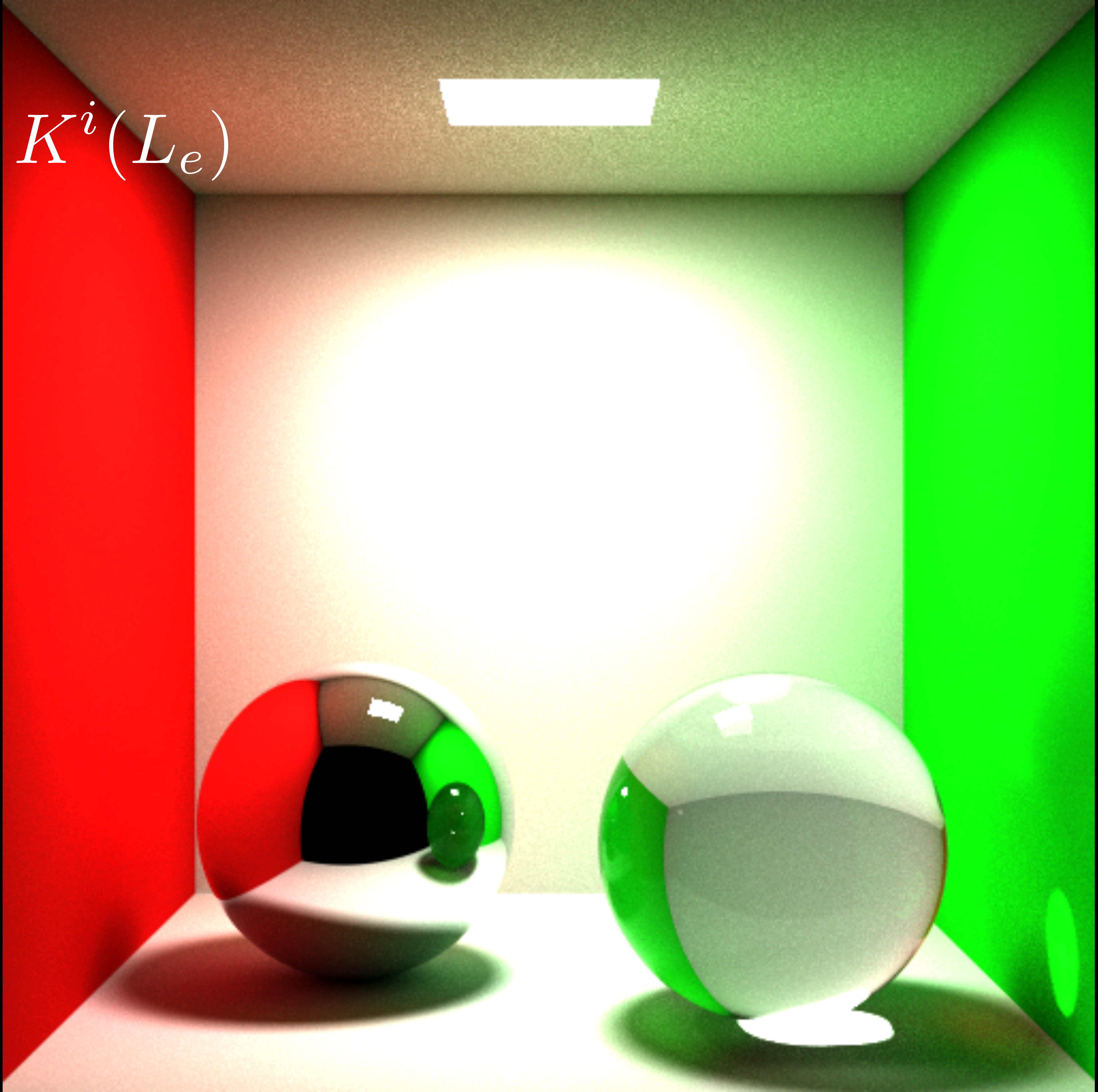
$$\sum_{i=0}^{4} K^i(L_e)$$



$$\sum_{i=0}^{5} K^i(L_e)$$



$$\sum_{i=0}^{6} K^i(L_e)$$




p
Direct illumination


p
One-bounce global illumination


p
Two-bounce global illumination

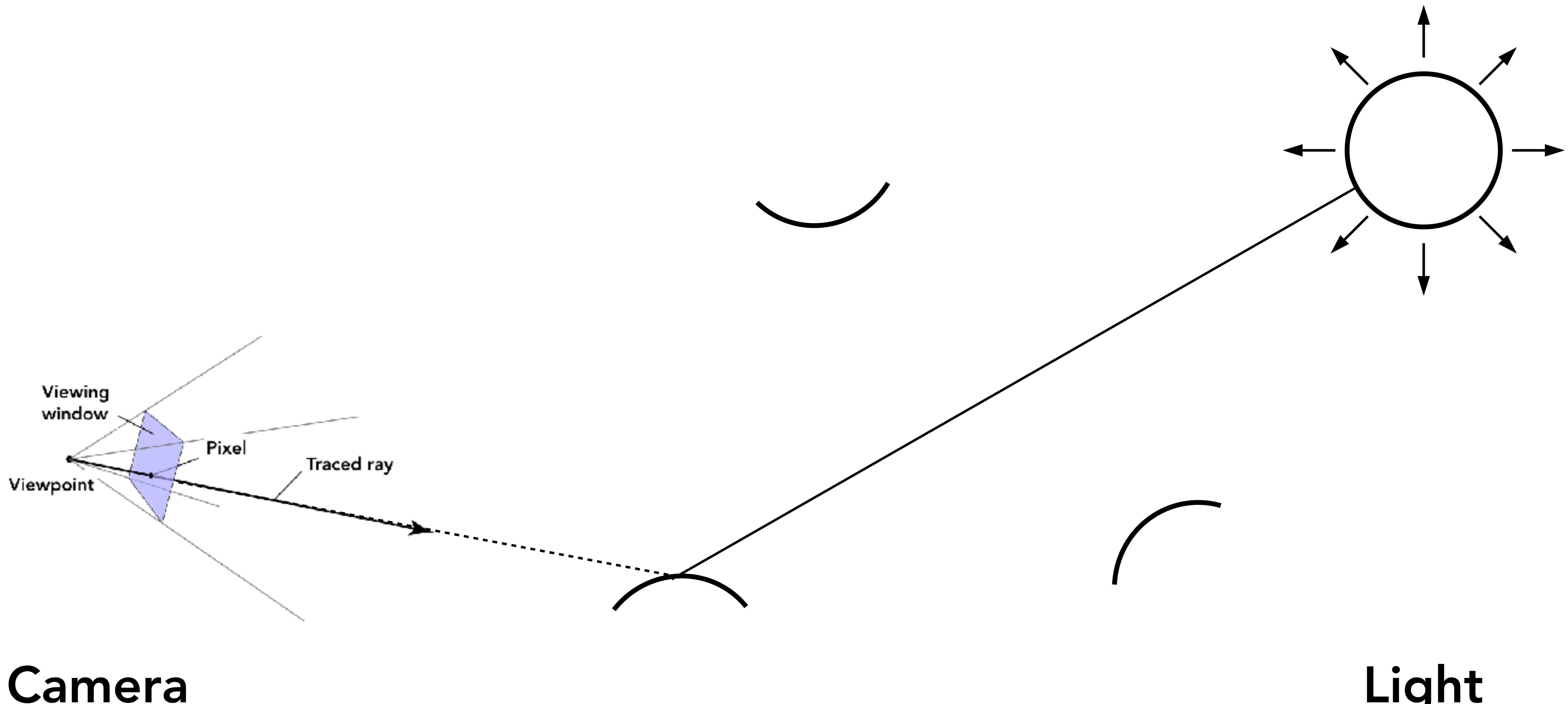

p
Four-bounce global illumination


p
Eight-bounce global illumination

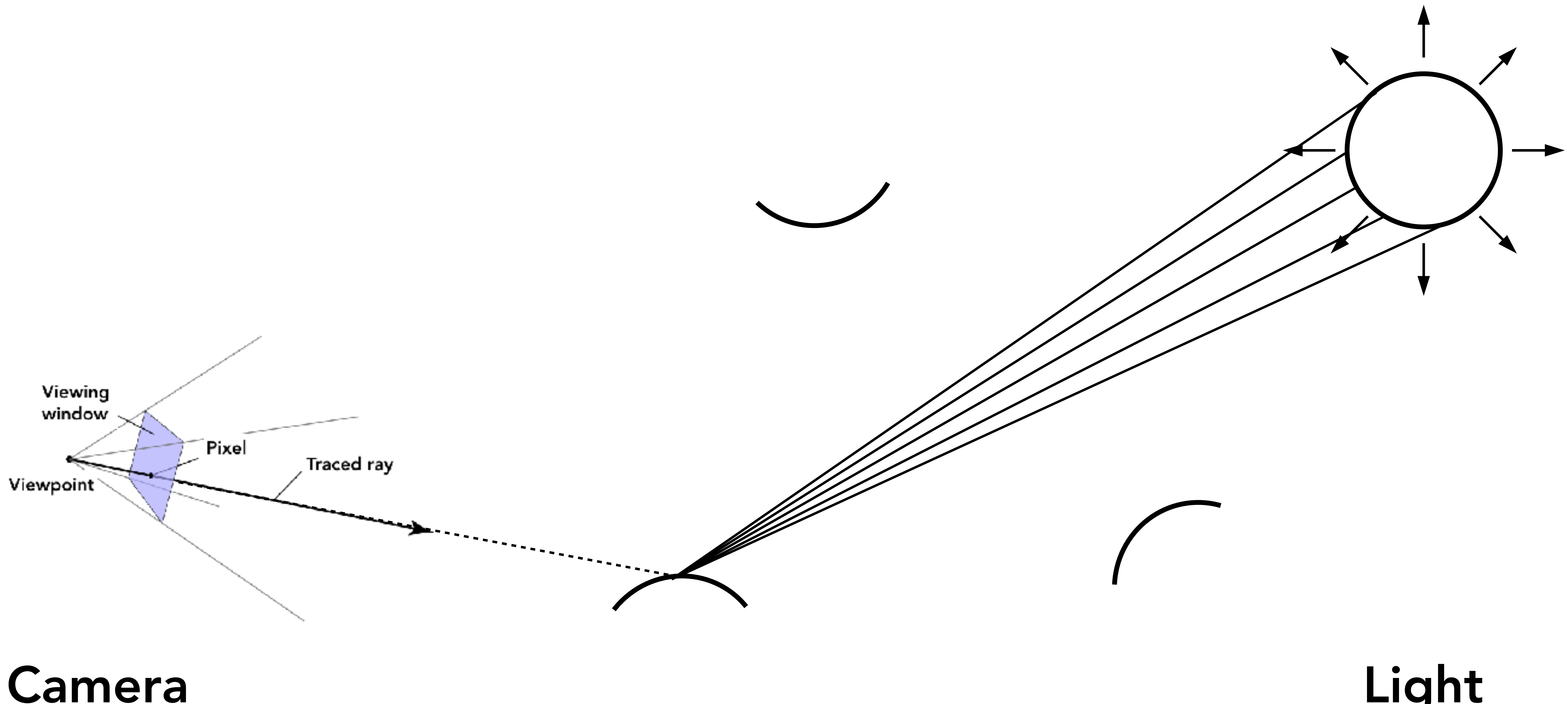

p
Sixteen-bounce global illumination

Light Paths

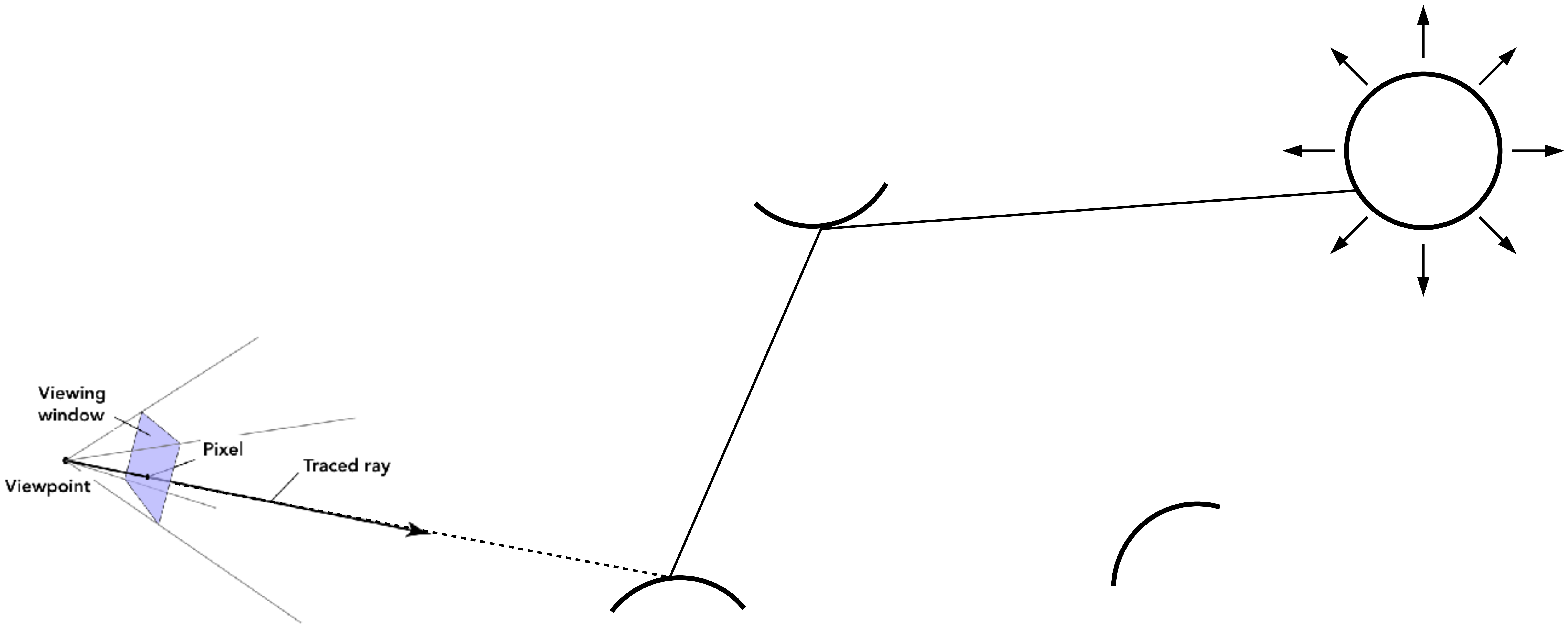
1-Bounce Path Connecting Ray to Light



1-Bounce Paths Connecting Ray to Light



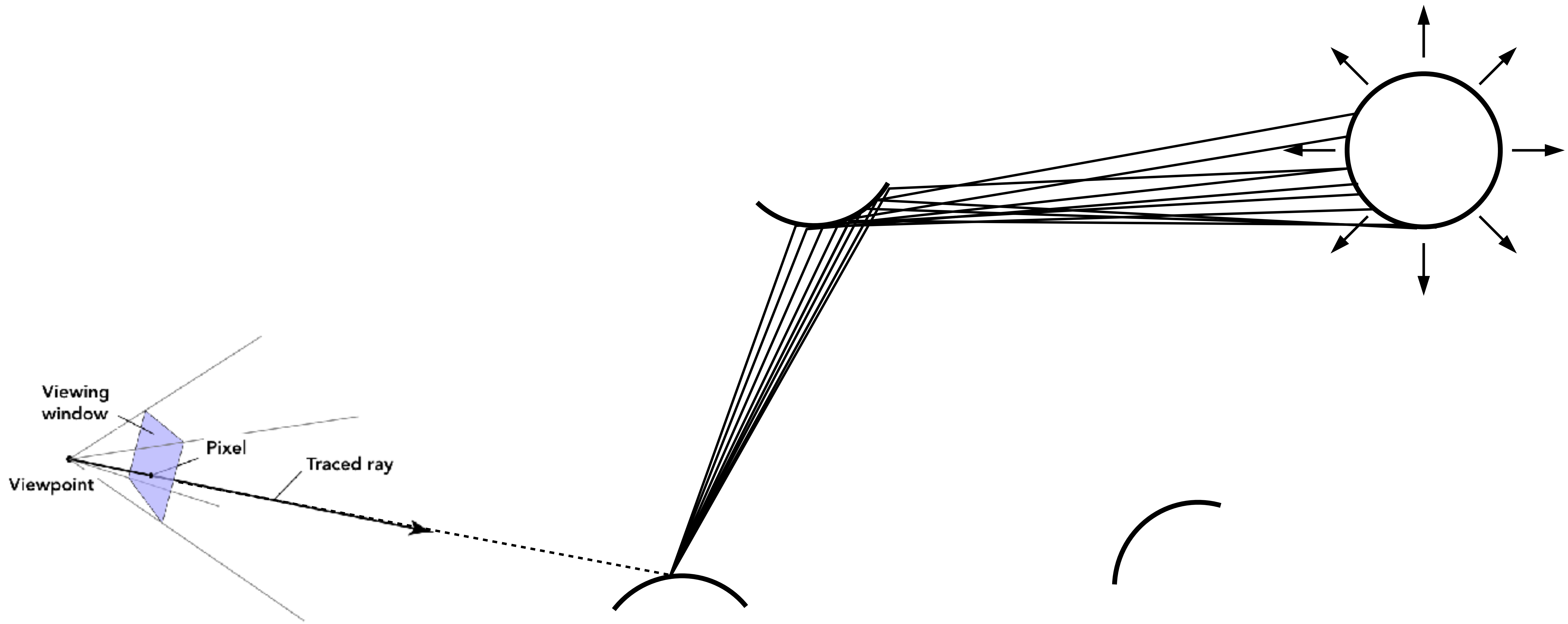
2-Bounce Path Connecting Ray to Light



Camera

Light

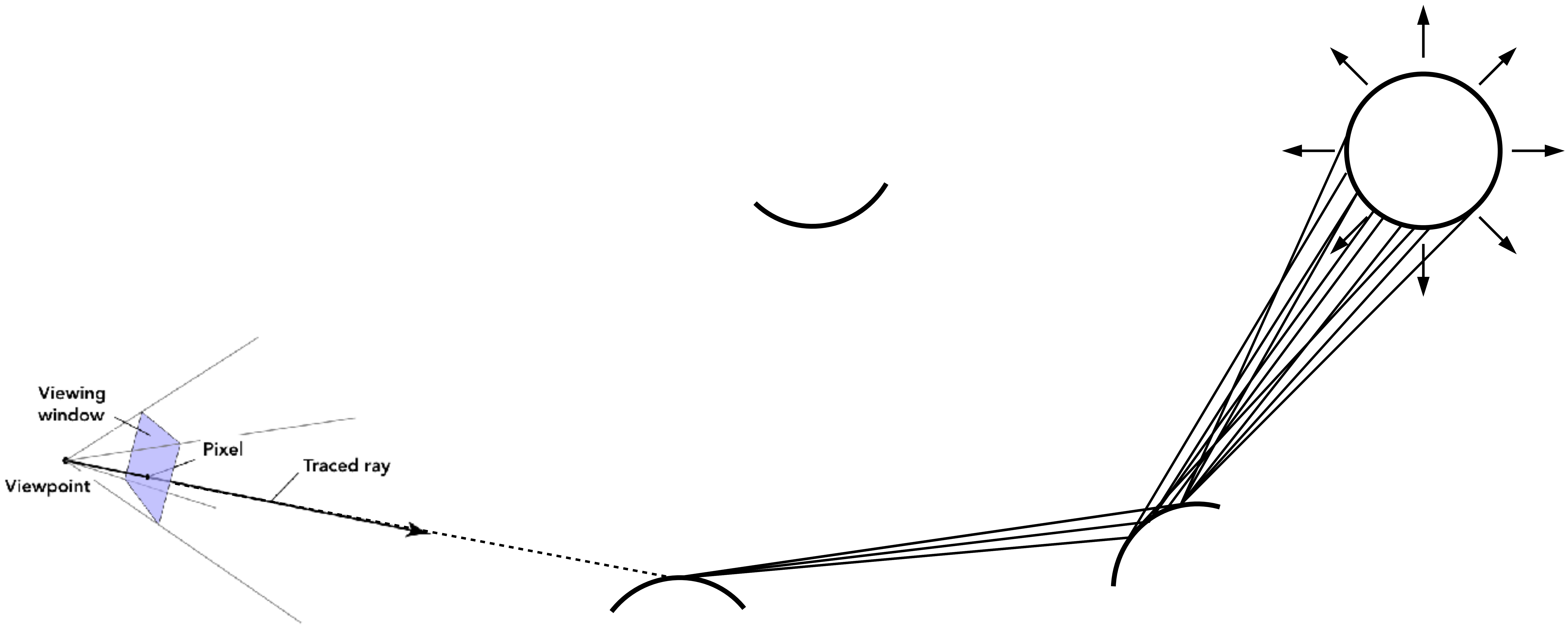
2-Bounce Paths Connecting Ray to Light



Camera

Light

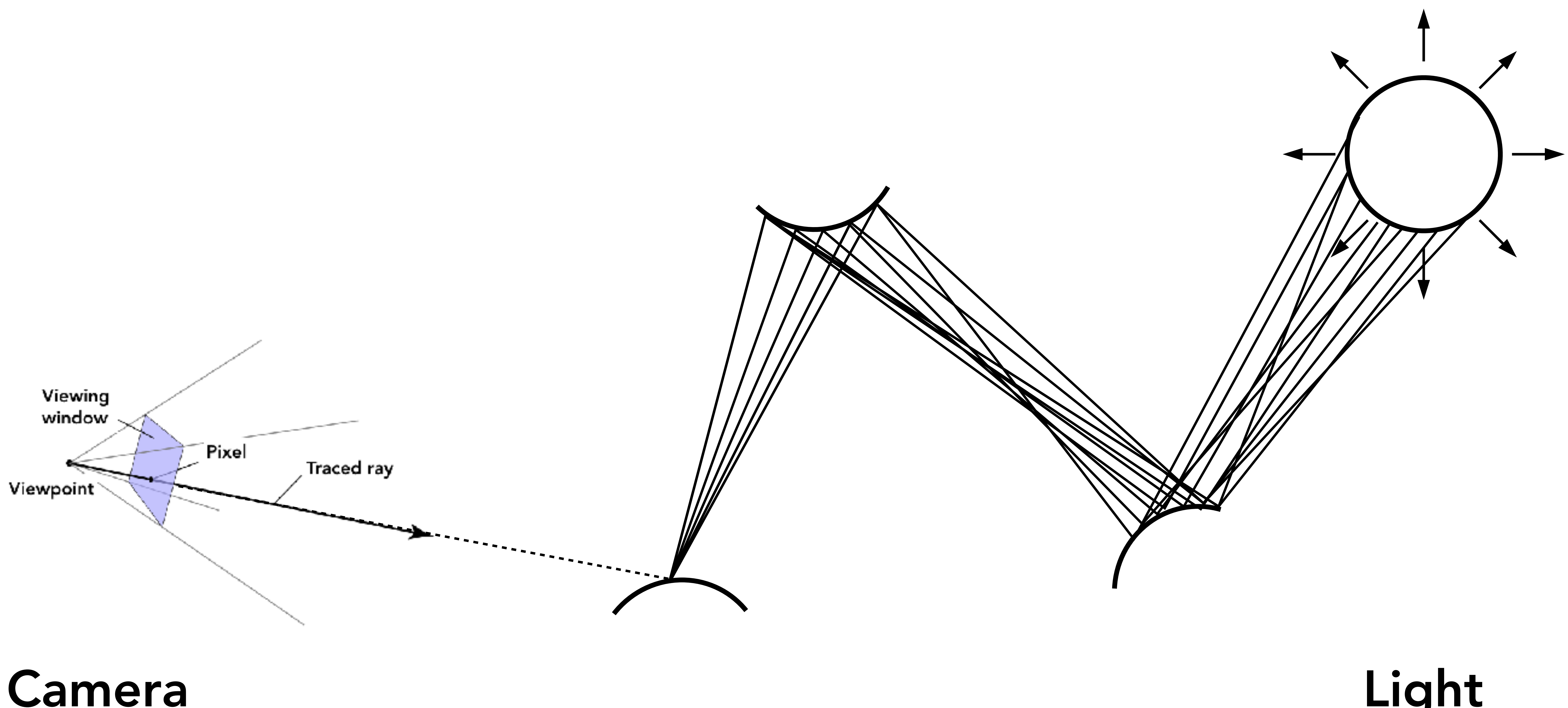
2-Bounce Paths Connecting Ray to Light



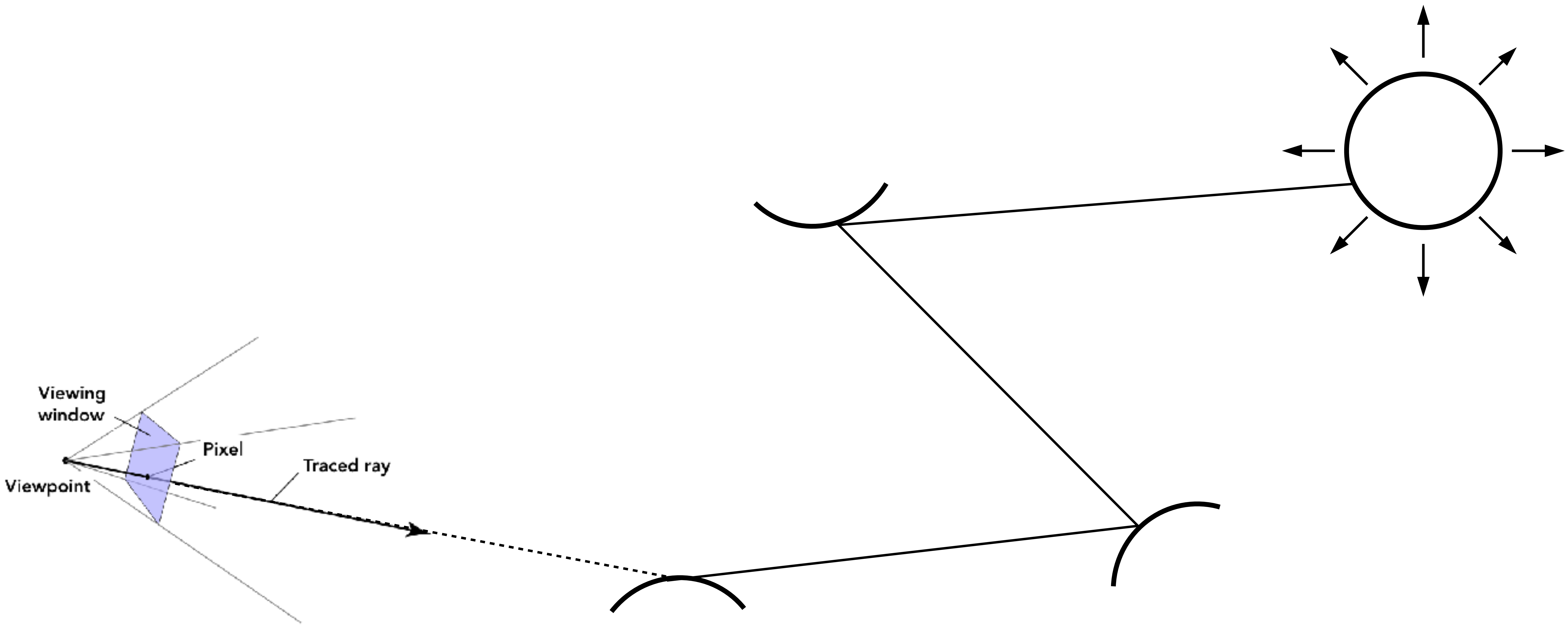
Camera

Light

3-Bounce Paths Connecting Ray to Light



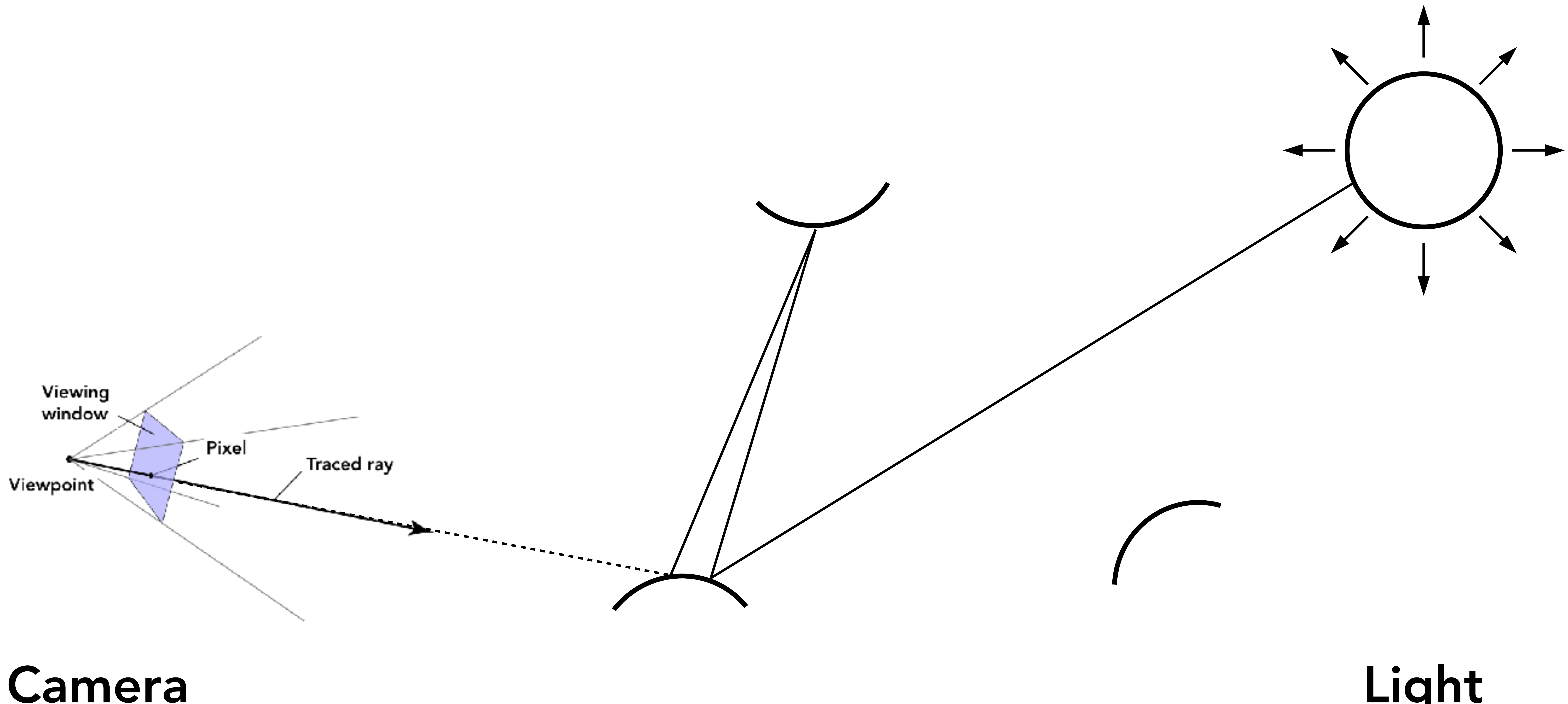
3-Bounce Path Connecting Ray to Light



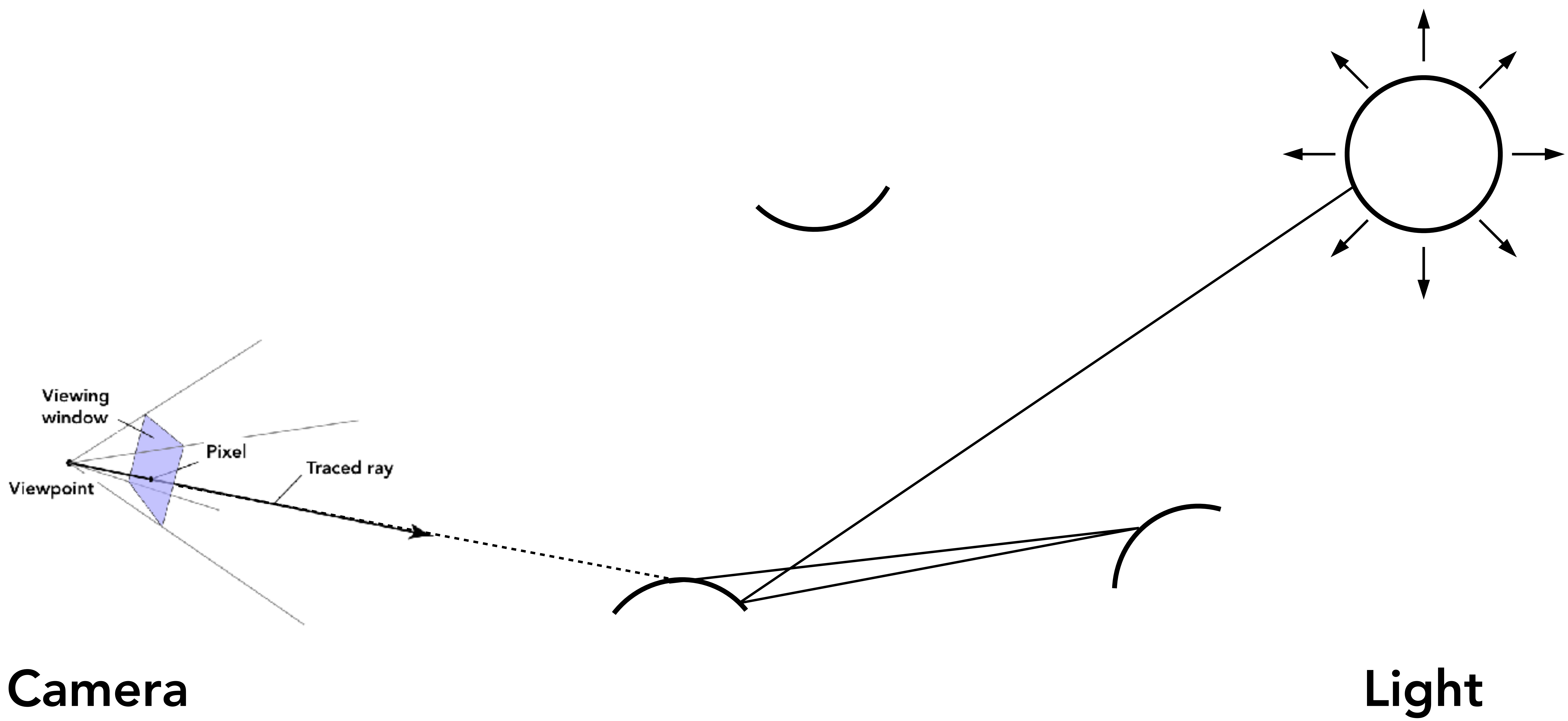
Camera

Light

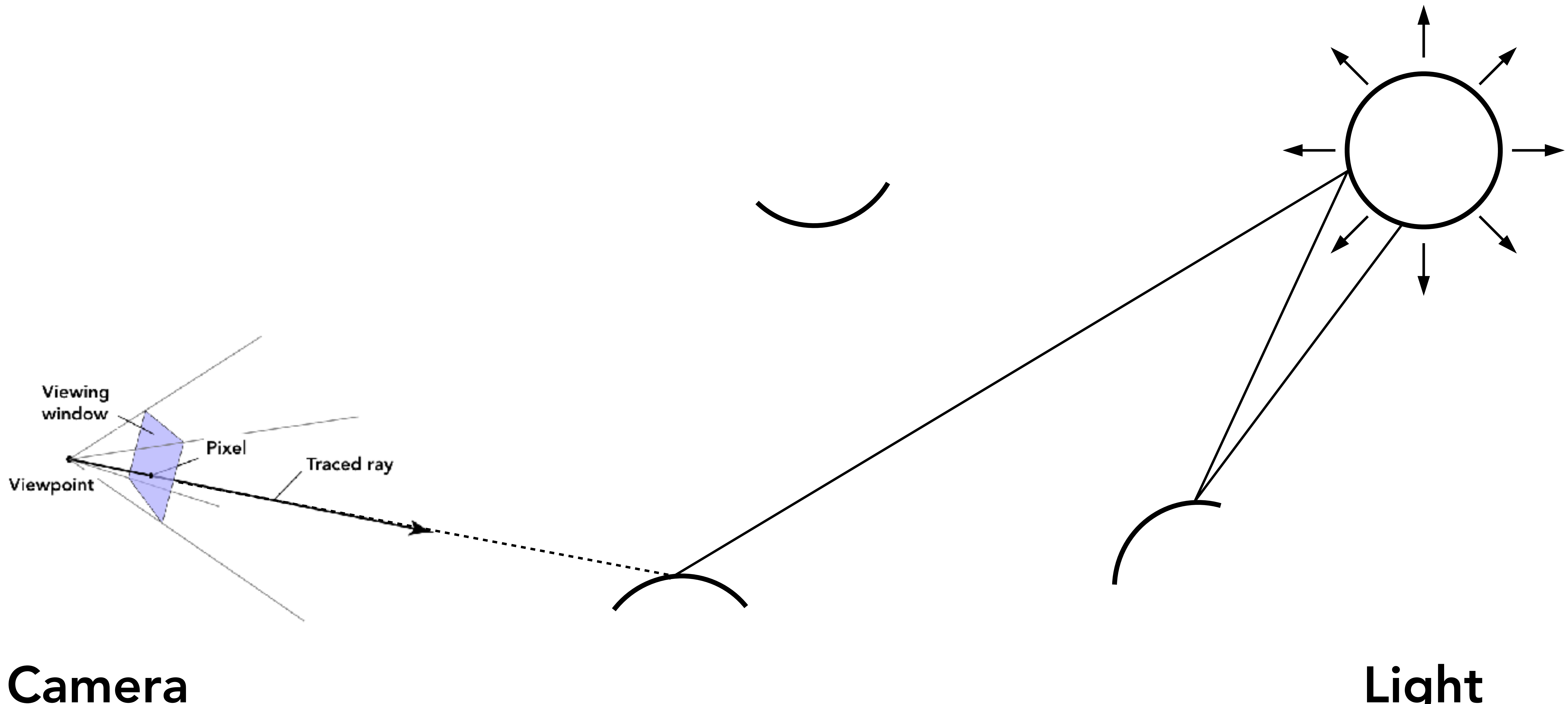
3-Bounce Path Connecting Ray to Light



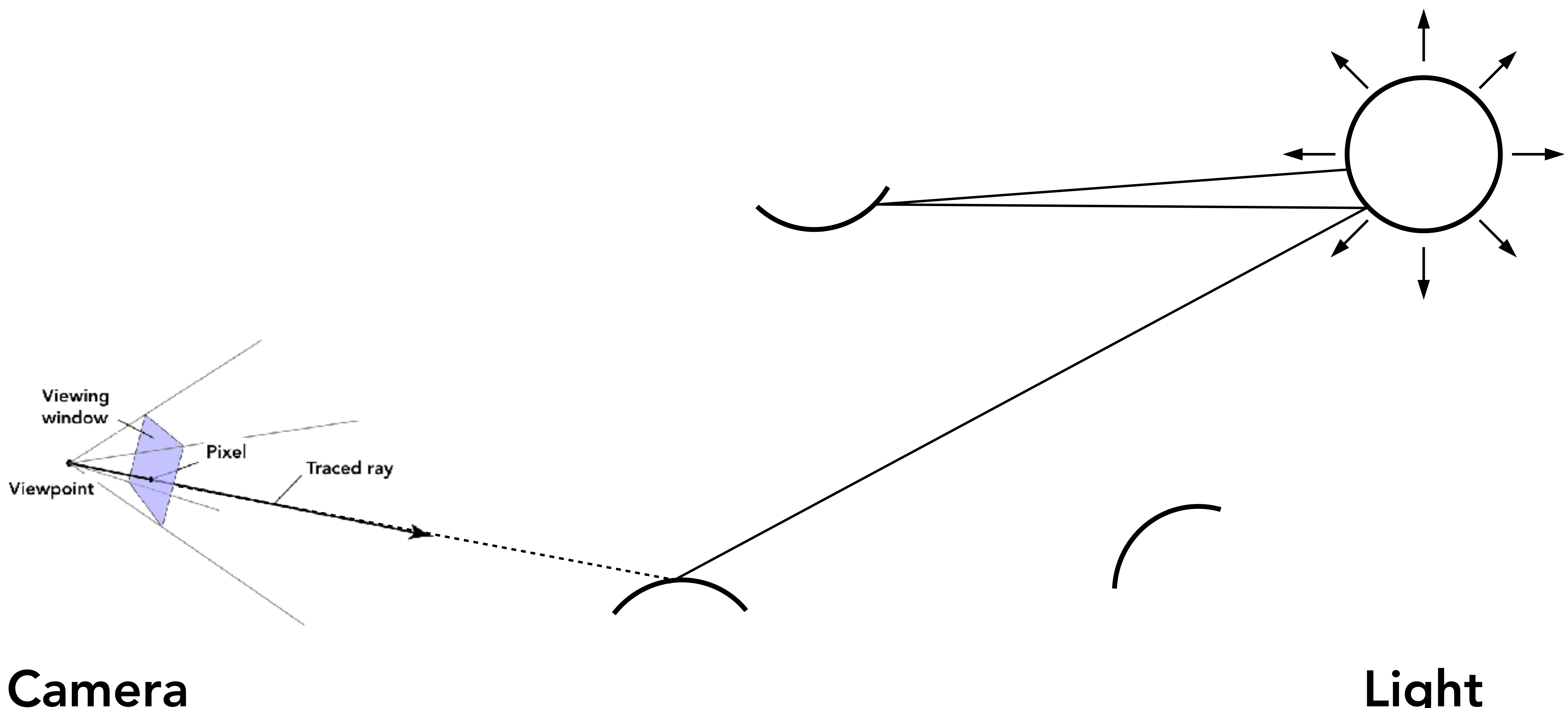
3-Bounce Path Connecting Ray to Light



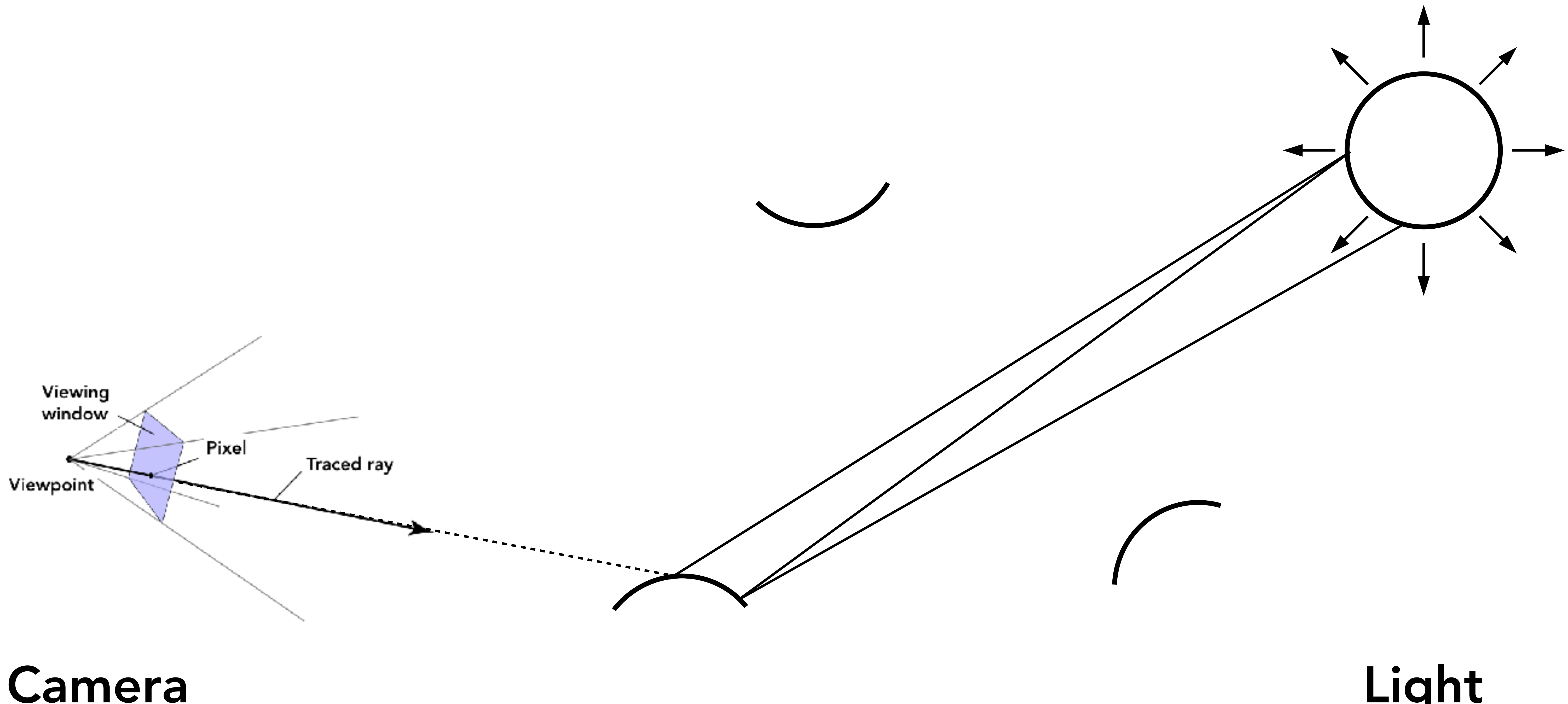
3-Bounce Path Connecting Ray to Light



3-Bounce Path Connecting Ray to Light



3-Bounce Path Connecting Ray to Light



Discussion: Global Illumination Rendering

Sum over all paths of all lengths

Challenges, discuss:

- How to generate all possible paths?
- How to sample space of paths efficiently?

Attendance Time

If you are seated in class, go to this form and sign in:

- https://tinyurl.com/184lecture

Notes:

- Time-stamp will be taken when you submit form. Do it now, won't count later.
- Don't tell friends outside class to fill it out now, because we will audit at some point in semester.
- Failing audit will have large negative consequence. You don't need to, because you have an alternative!

Sum Over Paths

Try 1: Monte Carlo Sum over Paths

```
EstRadianceIn(x, ω)
   p = intersectScene(x, ω);
   L = p.emittedLight(-ω);
   ωi, pdf = p.brdf.sampleDirection();
   L += EstRadianceIn(p, ωi) * p.brdf(ωi, -ω) * costheta / pdf;
   return L;
```

- Note:
  - Importance sampling BRDF
  - Infinite recursion!

Problem: Infinite Bounces of Light

How to integrate over infinite dimensions?

- Note: if energy dissipates, contribution of higher bounces decreases exponentially

Idea: just use N bounces

- Problem: biased! No matter how many Monte Carlo samples, never see light taking N+1 to infinity bounces

Idea: probabilistic termination?

- Non-zero probability of sampling paths of arbitrarily high number of bounces
- Surprisingly, can design this to be unbiased — this is called <u>Russian Roulette</u>

Russian Roulette: Unbiased Random Termination

New estimator: evaluate original estimator with probability p_{rr}, reweighted. Otherwise ignore.

$$\text{Let } X_{\mathrm{rr}} = \begin{cases} \frac{X}{p_{\mathrm{rr}}}, & \text{with probability } p_{\mathrm{rr}} \\ 0, & \text{otherwise} \end{cases}$$

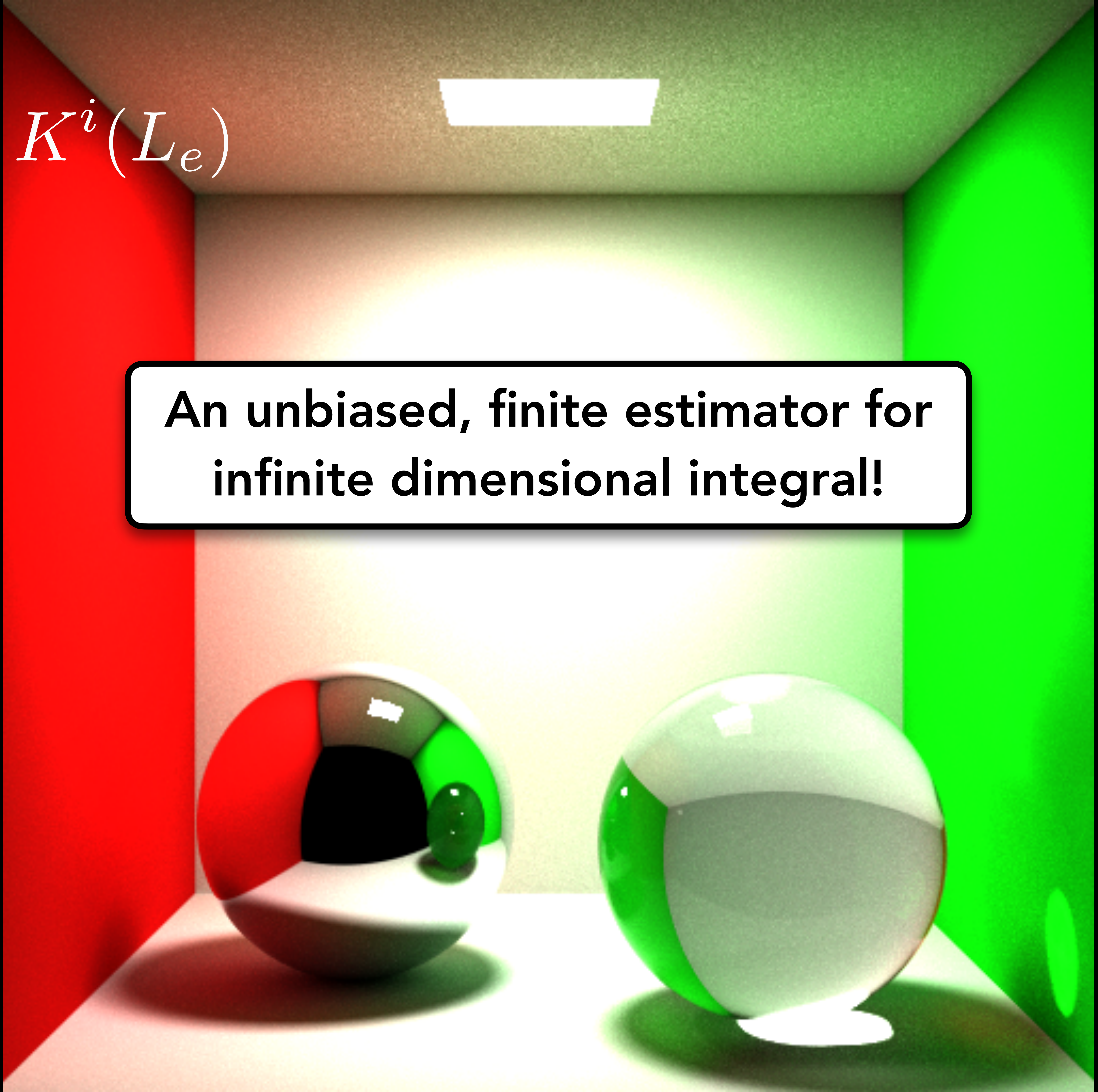
Same expected value as original estimator:

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

Want to choose p_{rr} considering Monte Carlo efficiency

- Terminate if expensive and/or low contribution
- In path tracing, expensive to recursively trace path. Increase termination probability if brdf is low in next bounce direction

$$\sum_{i=0}^{\infty} K^i(L_e)$$



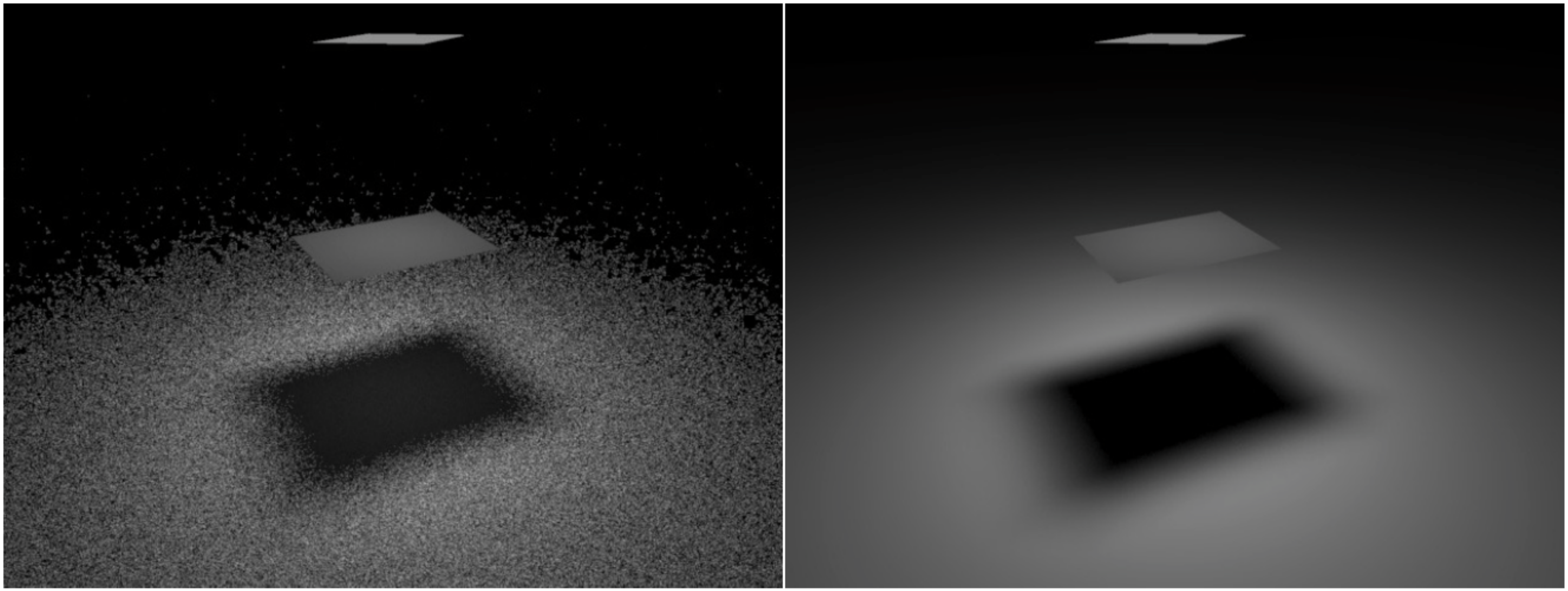
An unbiased, finite estimator for infinite dimensional integral!

Try 2: Russian Roulette Monte Carlo over Paths

```
EstRadianceIn(x, ω)
   p = intersectScene(x, ω);
   L = p.emittedLight(-ω);
   wi, pdf = p.brdf.sampleDirection();
   cpdf = continuationProbability(p.brdf, wi);
   if (random01() < cpdf)                          // Russian Roulette
      L += EstRadianceIn(p, wi)                    // Recursion
              * p.brdf(wi, -ω) * costheta / pdf / cpdf;
   return L;


// Unbiased, computation terminates, but still extremely noisy!
```

Recall: Importance Sampling



Solid angle sampling | Light area sampling

Path Tracing

Path Tracing Overview

Terminate paths randomly with Russian Roulette

Partition the recursive radiance evaluation. At each point on light path

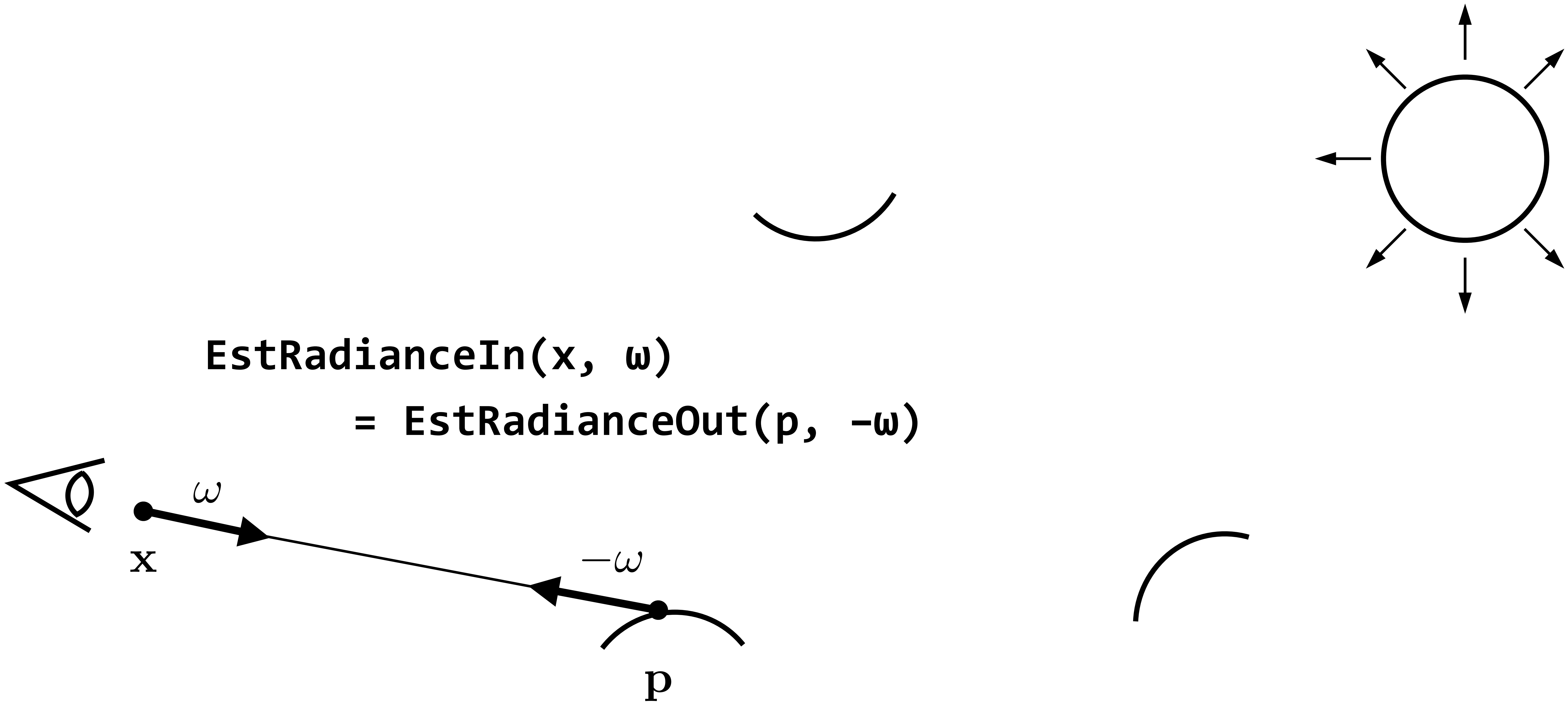
- Direct lighting – non-recursive, importance sample lights
- Indirect lighting – recursive, importance sample BRDF

Monte Carlo estimate for each partition separately

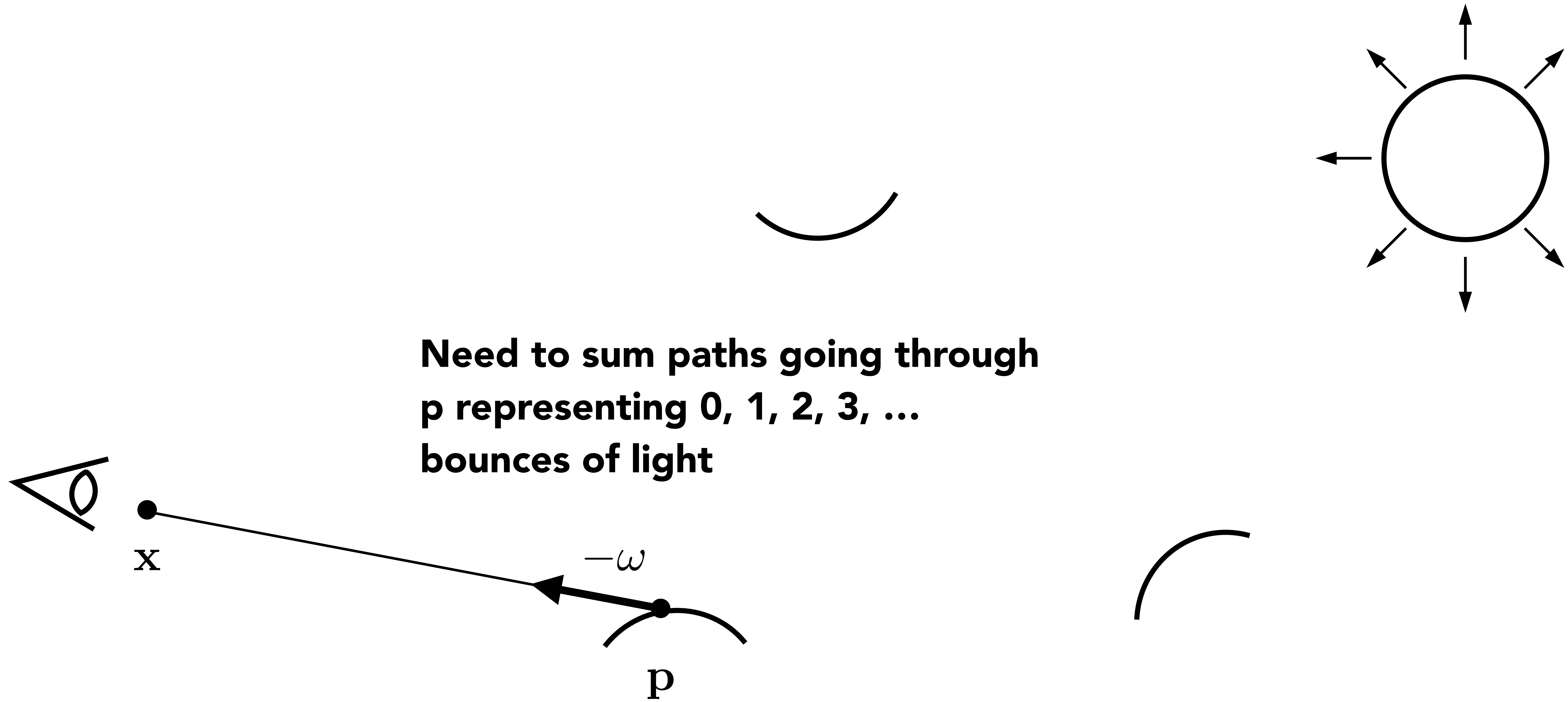
- Possible to take just one sample for each
- Assume: 100s - 1000s of paths sampled per pixel

Partitioning the Rendering Equation

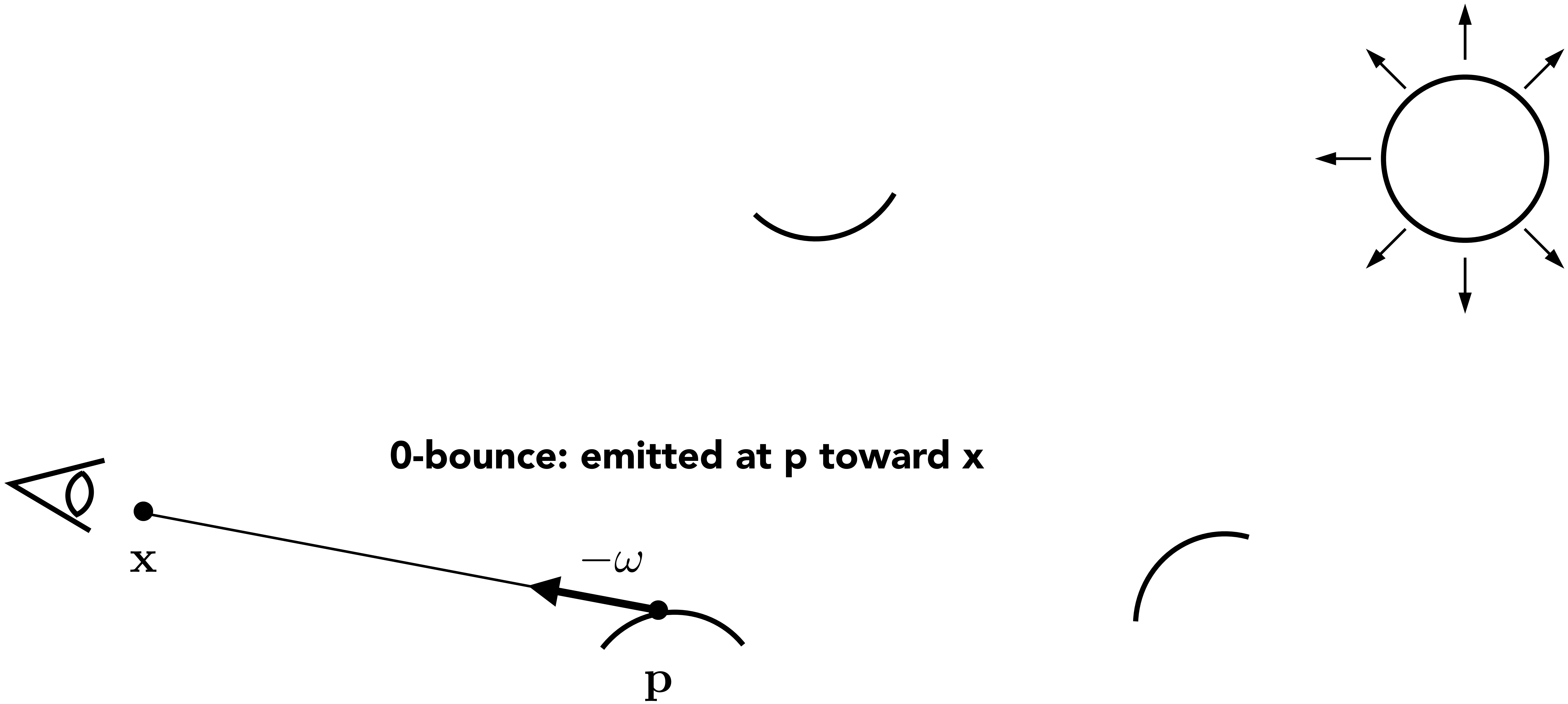
```
EstRadianceIn(x, ω)
      = EstRadianceOut(p, −ω)
```



Partitioning the Rendering Equation



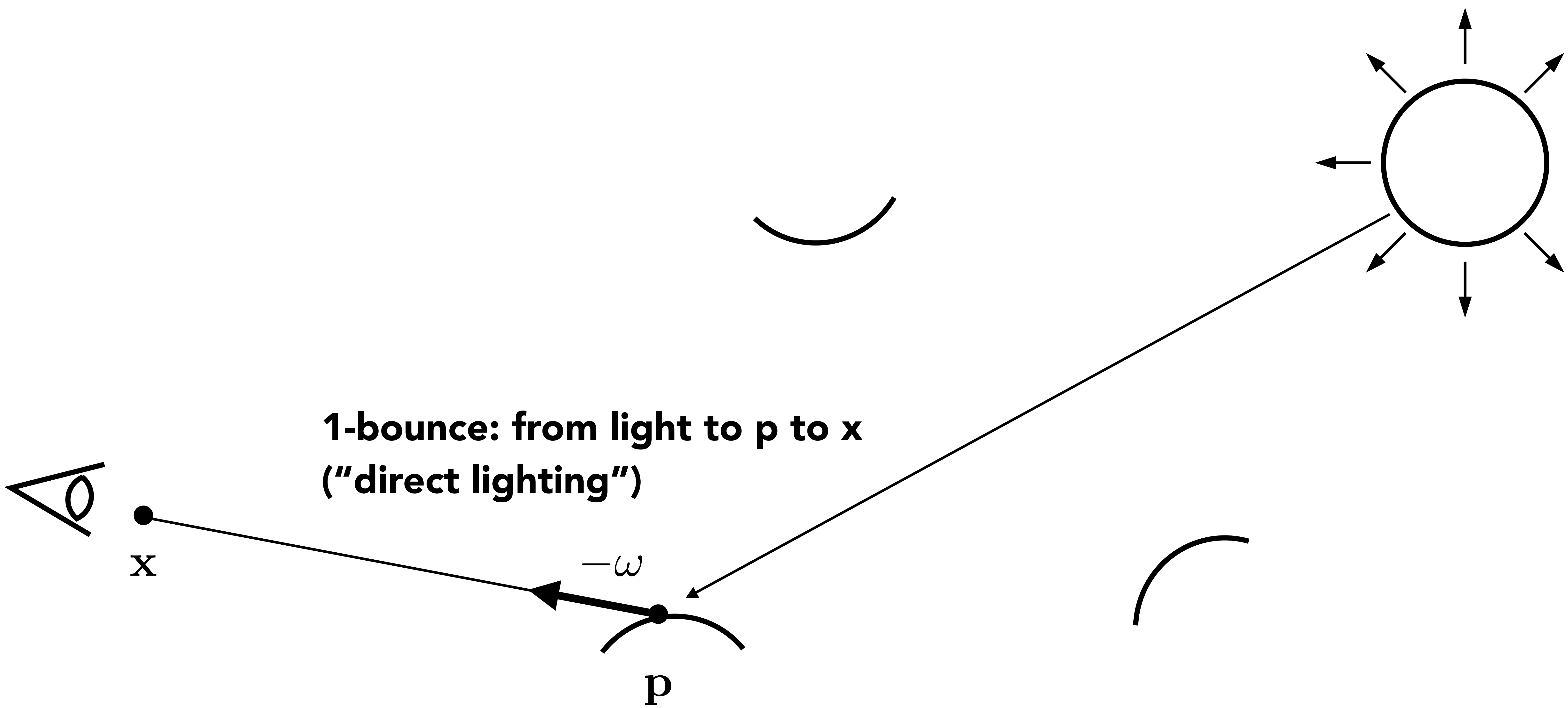
Partitioning the Rendering Equation



At p, consider light contributions from paths of varying bounce-length

- 0-bounce: light emitted from p (p is on a light source)

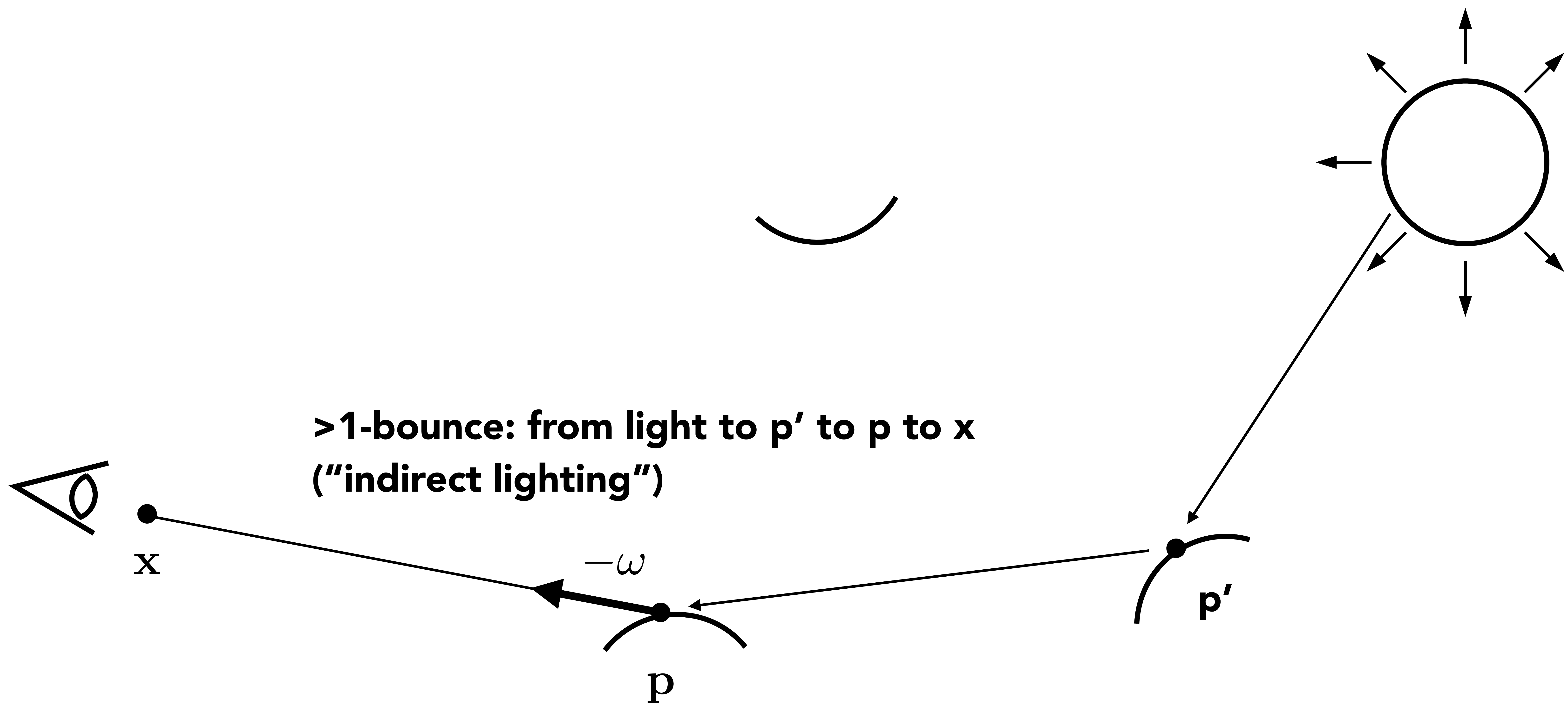
Partitioning the Rendering Equation



At p, consider light contributions from paths of varying bounce-length

- 0-bounce: light emitted from p (p is on a light source)
- 1-bounce: from light to p to x ("direct illumination")

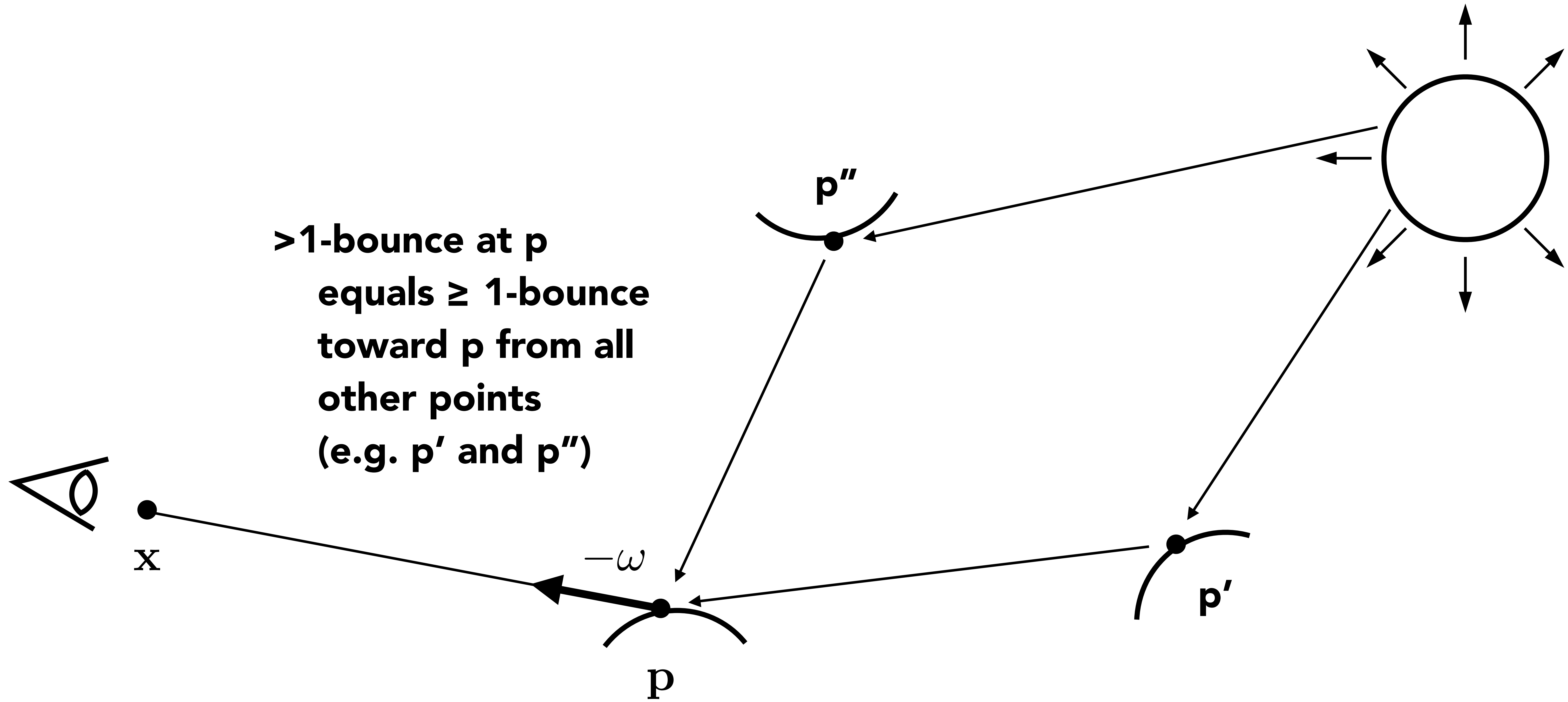
Partitioning the Rendering Equation



At p, consider light contributions from paths of varying bounce-length

- 0-bounce: light emitted from p (p is on a light source)
- 1-bounce: from light to p to x ("direct illumination")
- >1-bounce: from light to at least one other point to p to x ("indirect illumination")

Consider Evaluation of >1 Bounce of Light



At p, consider light contributions from paths of varying bounce-length

- 0-bounce: light emitted from p (p is on a light source)
- 1-bounce: from light to p to x ("direct illumination")
- >1-bounce: from light to at least one other point to p to x ("indirect illumination")

Path Tracing Pseudocode

```
EstRadianceIn(x, ω)                  // incoming at x from dir ω
   p = intersectScene(x, ω);
   return ZeroBounceRadiance(p, -ω)
          + AtLeastOneBounceRadiance(p, -ω);

ZeroBounceRadiance(p, wo)            // outgoing at p in dir ω
   return p.emittedLight(wo);
```

Path Tracing Pseudocode

```
AtLeastOneBounceRadiance(p, wo)                    // out at p, dir wo
  L = OneBounceRadiance(p, wo);                    // direct illum

  wi, pdf = p.brdf.sampleDirection();              // Imp. sampling
  p’ =  intersectScene(p, wi);
  cpdf = continuationProbability(p.brdf, wi, wo);
  if (random01() < cpdf)                           // Russ. Roulette
    L += AtLeastOneBounceRadiance(p’, -wi)  // Recursive est. of
    * p.brdf(wi, wo) * costheta / pdf / cpdf;// indirect illum
  return L;

OneBounceRadiance(p, wo)                           // out at p, dir wo
   return DirectLightingSampleLights(p, wo); // direct illum
```

Direct Lighting Pseudocode (Lights)

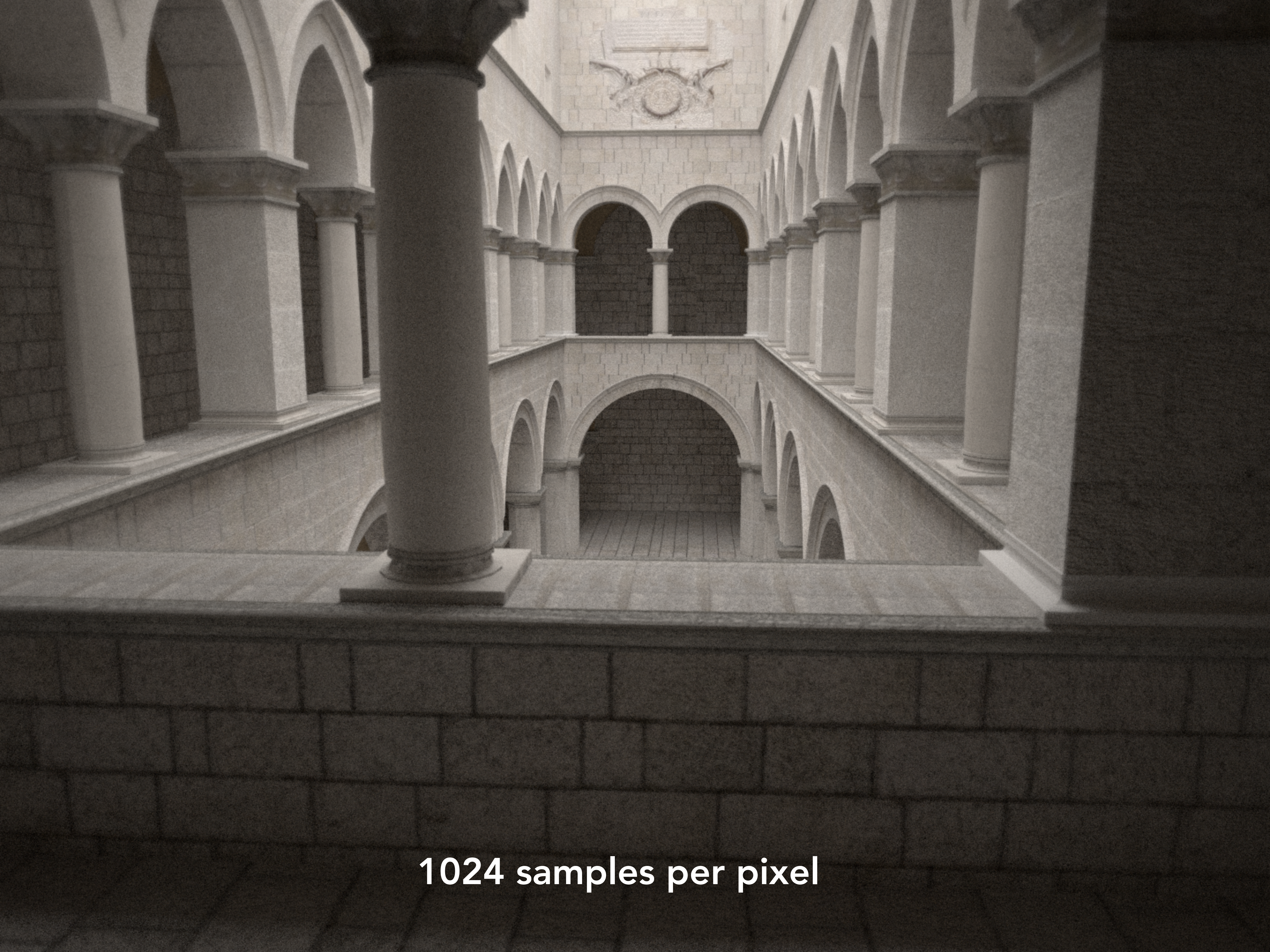
```
DirectLightingSamplingLights(p, wo)
   L, wi, pdf = lights.sampleDirection(p);   // Imp. sampling

   if (scene.shadowIntersection(p, wi))      // Shadow ray
      return 0;
   else
      return L * p.brdf(wi, wo) * costheta / pdf;

// Note: only one random sample over all lights.
// Assignment 3-A asks you to, alternatively, loop over
// multiple lights and take multiple samples (later slide)
```

One sample per pixel

32 samples per pixel



1024 samples per pixel

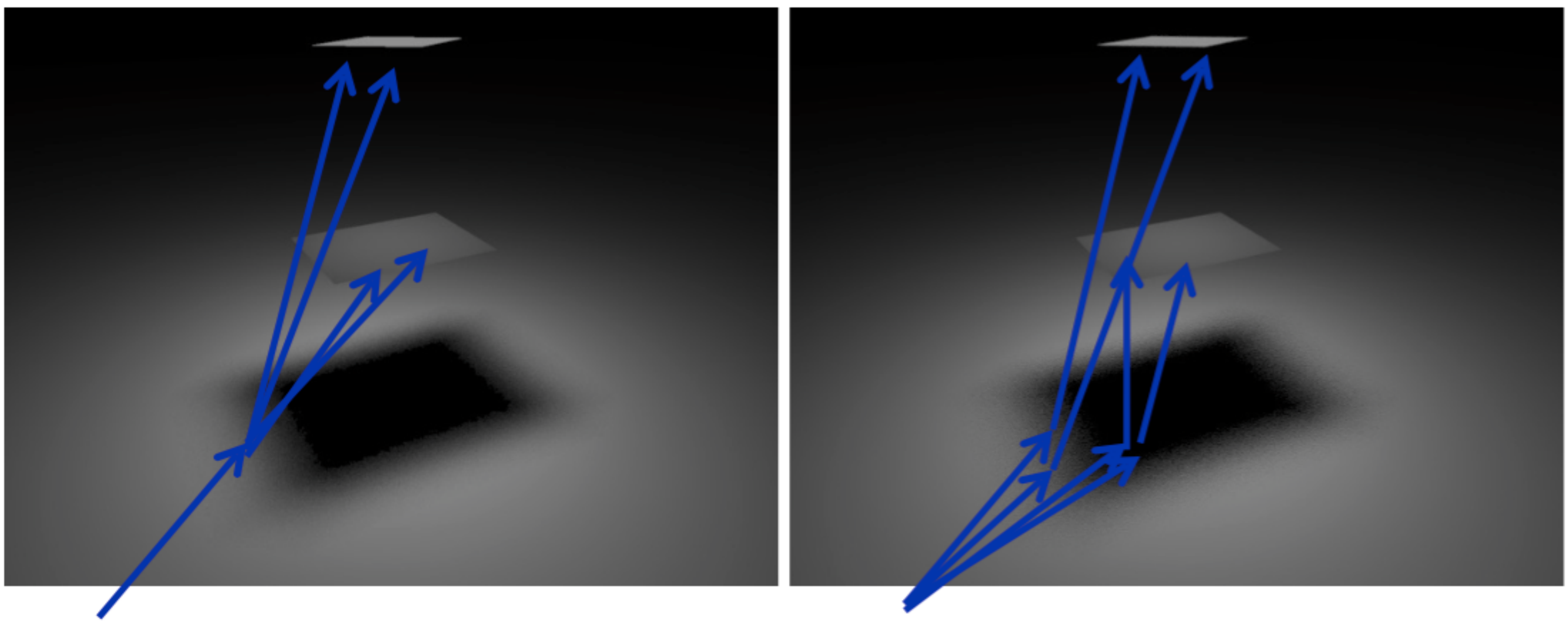
Summary of Intuition on Global Illumination & Path Tracing

Summary of Intuition on G.I. & P.T.

- Operator notation leads to insight that solution is adding successive bounces of light
- Trace N paths through a pixel, sample radiance
- Build paths by recursively tracing to next surface point and choosing a random reflection direction. At each surface, sum emitted light and reflected light
- How to terminate paths? We use Russian Roulette to kill probabilistically.
- How to reduce noise? Use importance sampling in choosing random direction. Two ways: importance sample the lights, and importance sample the BRDF.

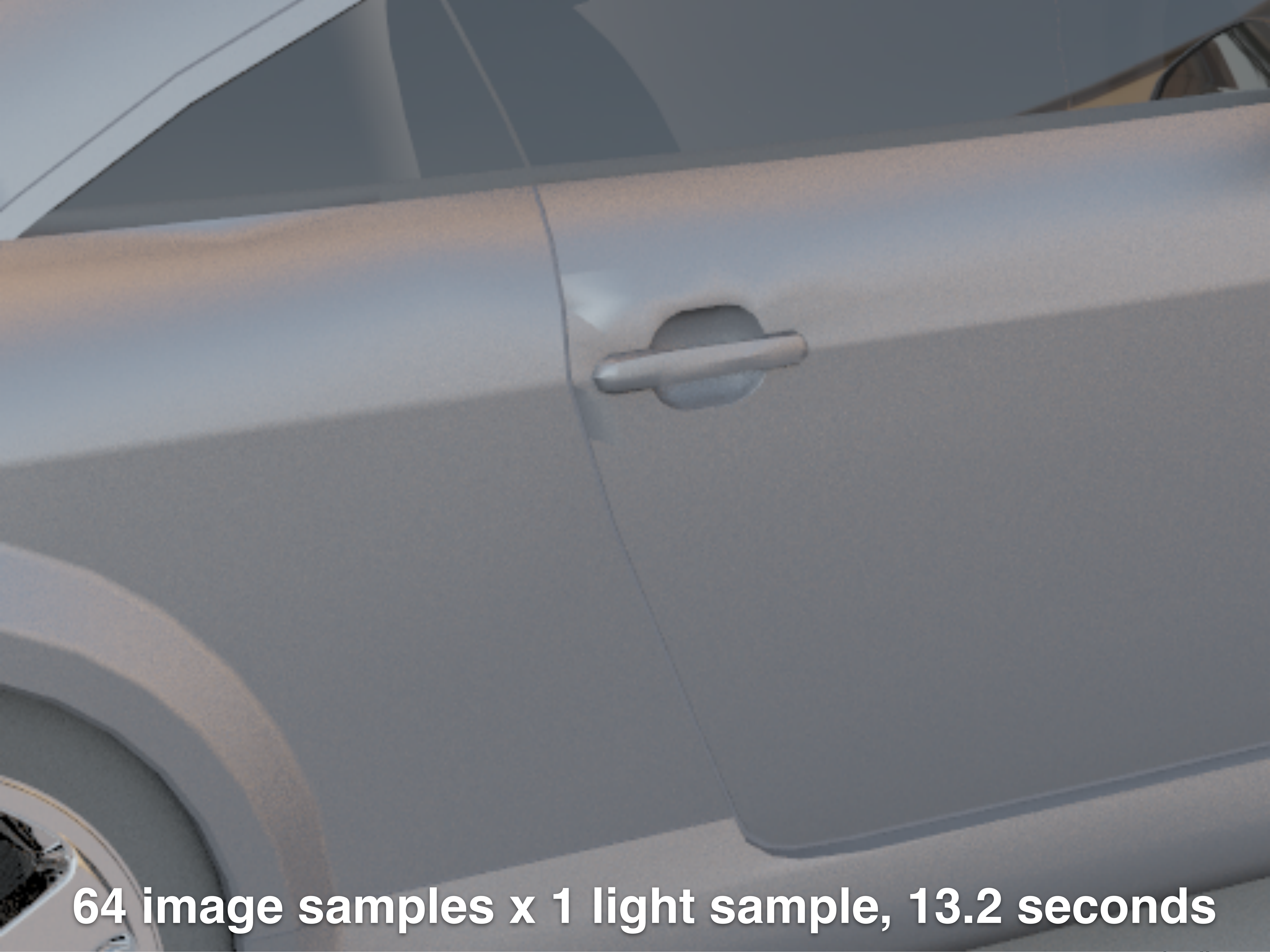
Implementation Notes

Paths vs Trees

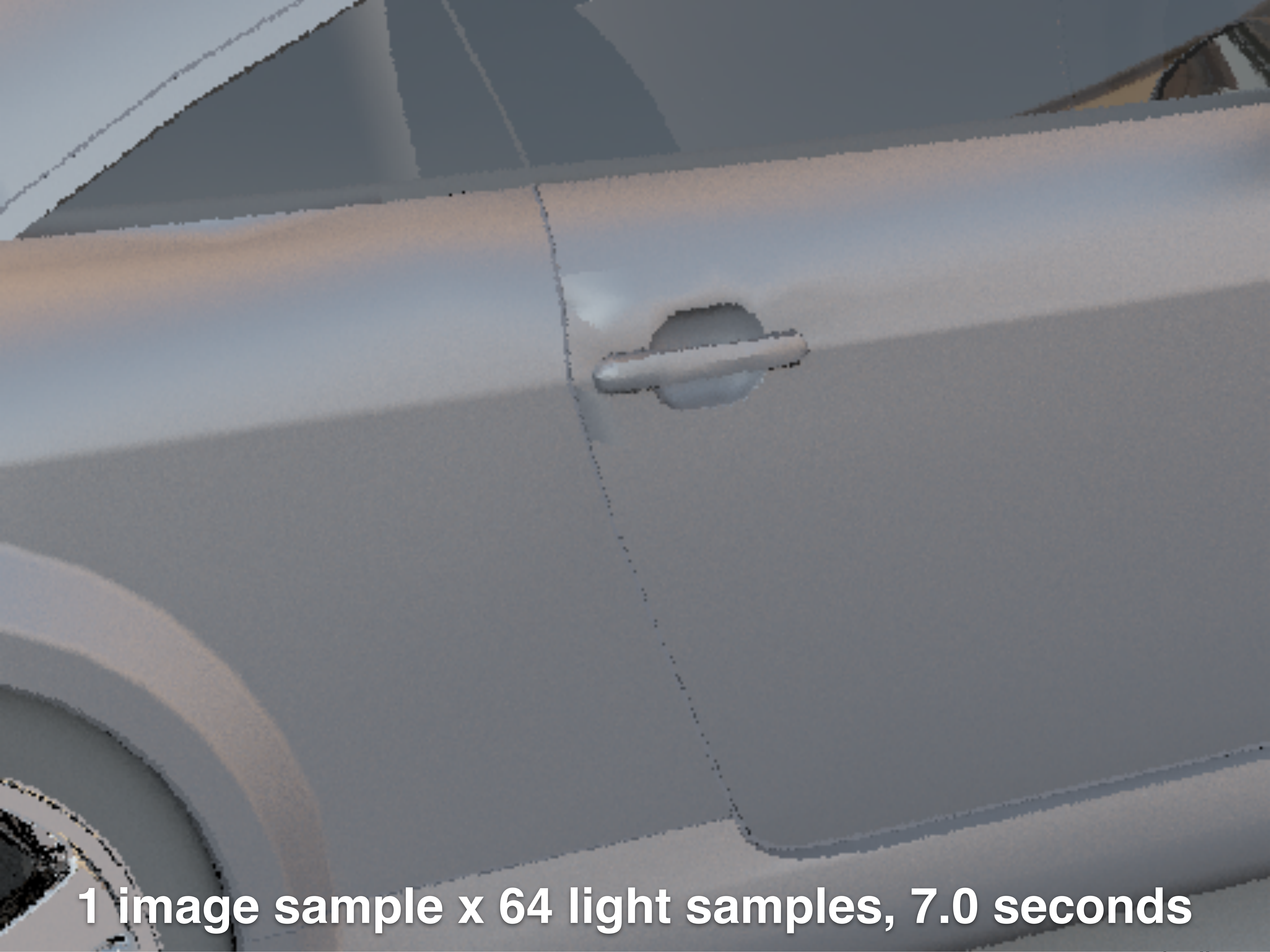


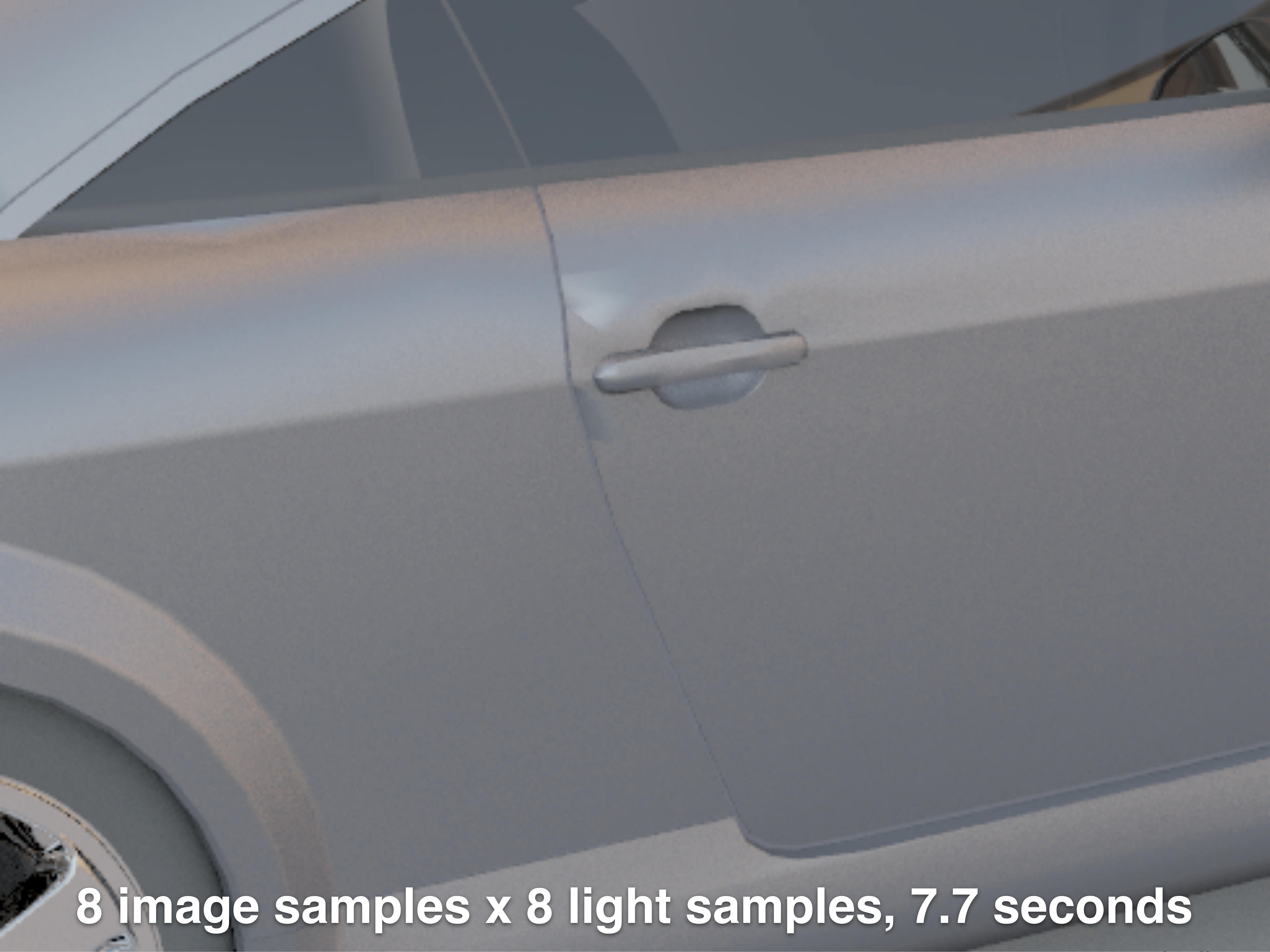
4 eye rays per pixel
16 shadow rays per eye ray
(68 ray traces per pixel)

64 eye rays per pixel
1 shadow ray per eye ray
(128 ray traces per pixel)



64 image samples x 1 light sample, 13.2 seconds


1 image sample x 64 light samples, 7.0 seconds



8 image samples x 8 light samples, 7.7 seconds

Multiple Light Sources

Consider multiple lights in direct lighting estimate

One strategy:

- Loop over all N lights, sum Monte-Carlo estimates for each light
- For each light: compute Monte Carlo estimate with M samples taken with importance sampling

Needs N * M samples

This is what the assignment asks you to implement.

Multiple Light Sources (Single Sample)

Consider random sampling of multiple lights with a single sample

- Randomly choose light i, with probability pi
- Randomly sample over that light's directions, with probability pL
- Probability of choosing sample is (pi * pL)
- Weight the lighting calculation by 1/(pi * pL)
- Is this estimator unbiased? Yes!
- How would you importance sample intelligently?

Can of course average N such samples

Point Lights / Ideal Specular Materials – Issues

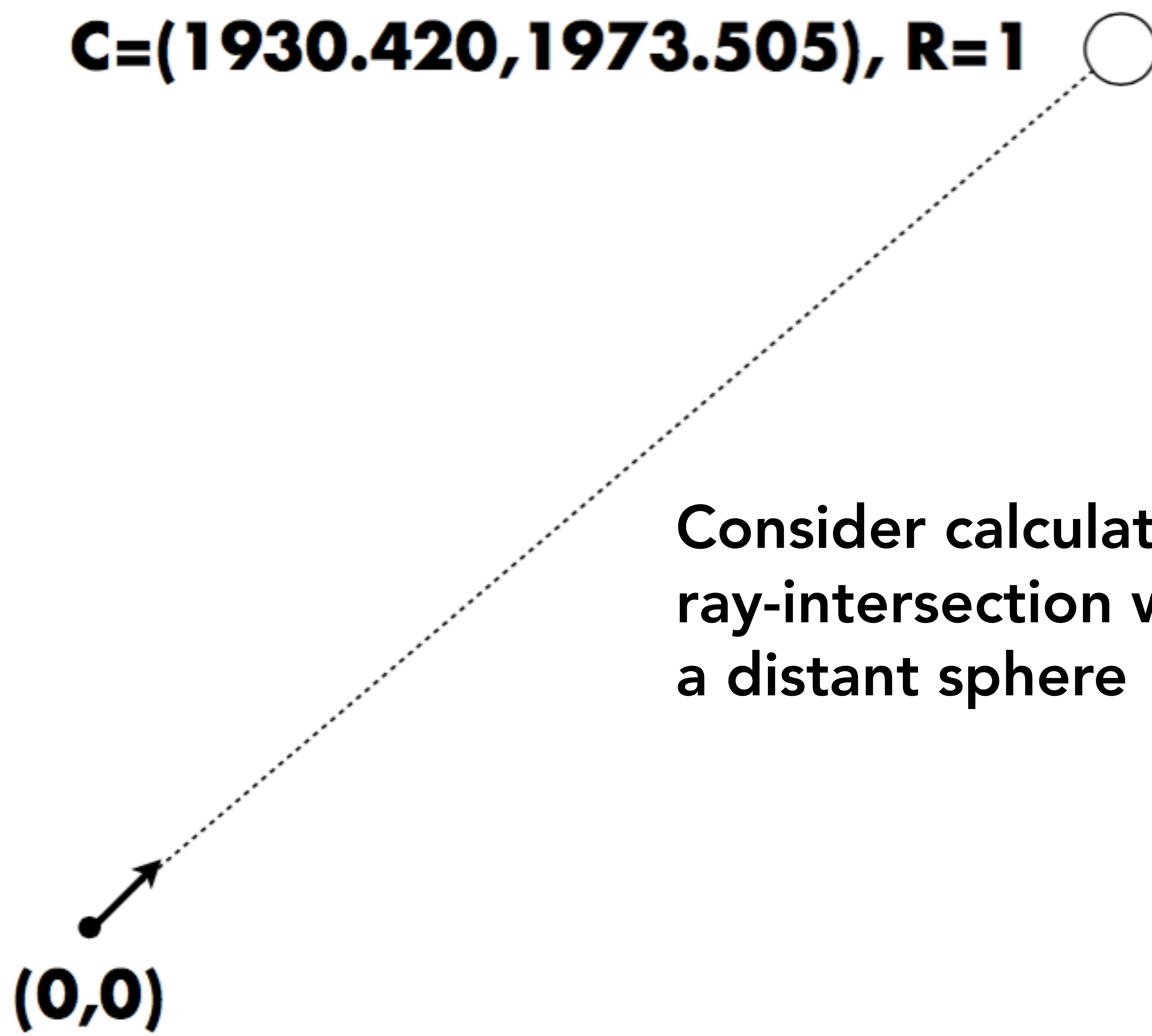
Sampling problems

- When sampling directions randomly, we have zero probability of matching exact direction of a point light or mirror reflection / specular refraction

Remedy

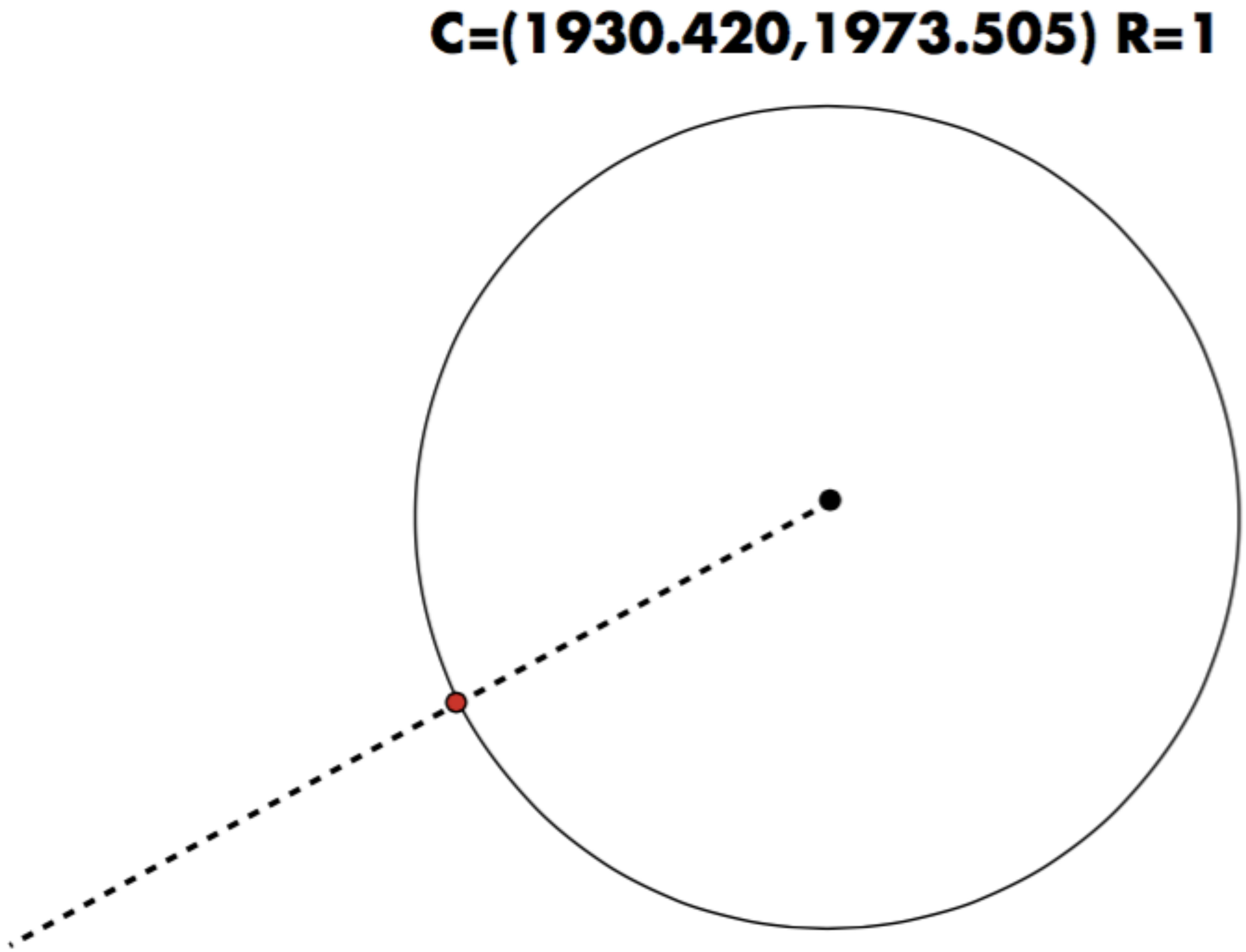
- In direct lighting, importance sample point lights by generating a single sample pointing directly at the light (only one sample needed)
- In indirect lighting, importance sample specular BRDFs by generating a sample point directly along the specular refraction / transmission direction

Numerical Precision Issues



Consider calculating ray-intersection with a distant sphere

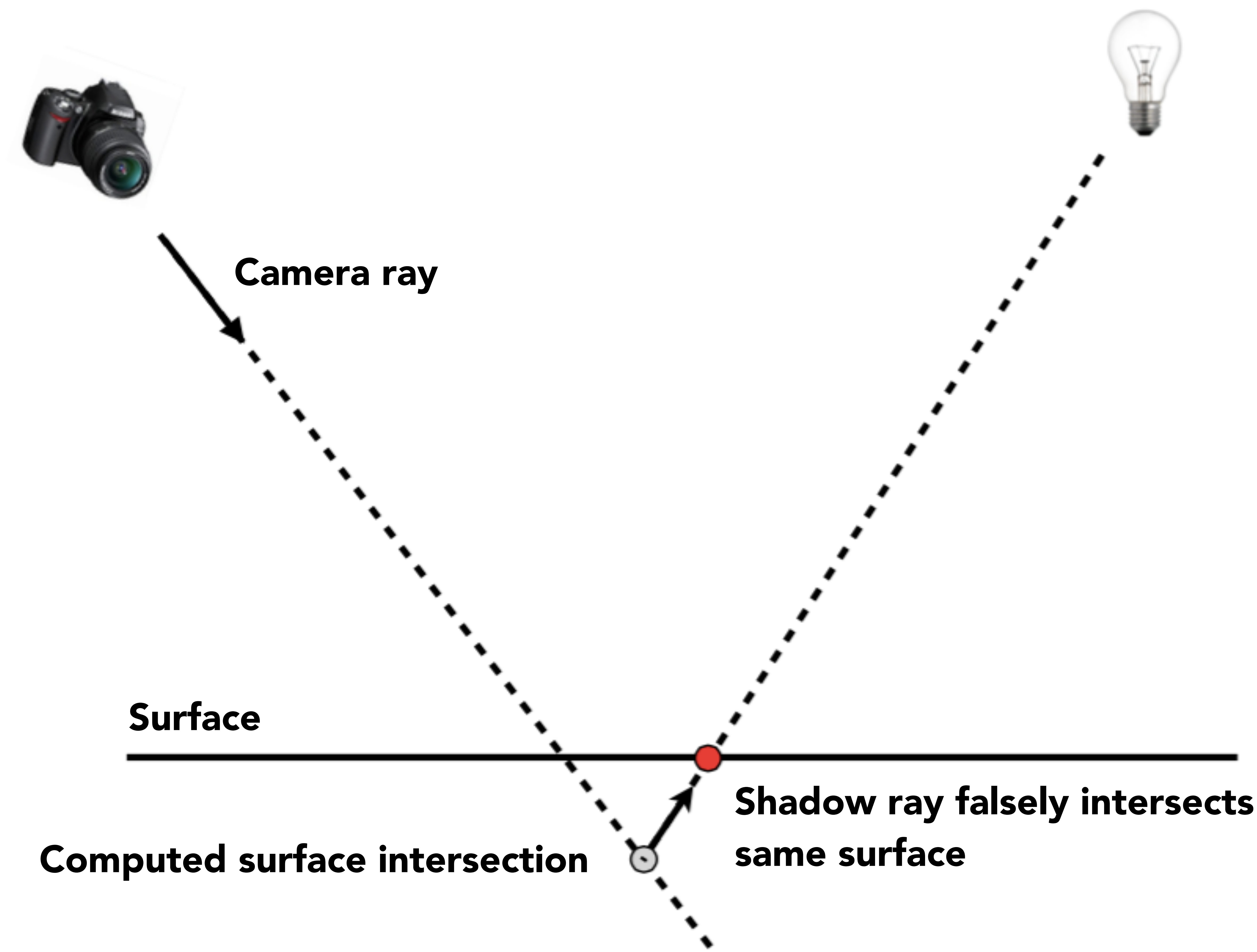
Numerical Precision Issues

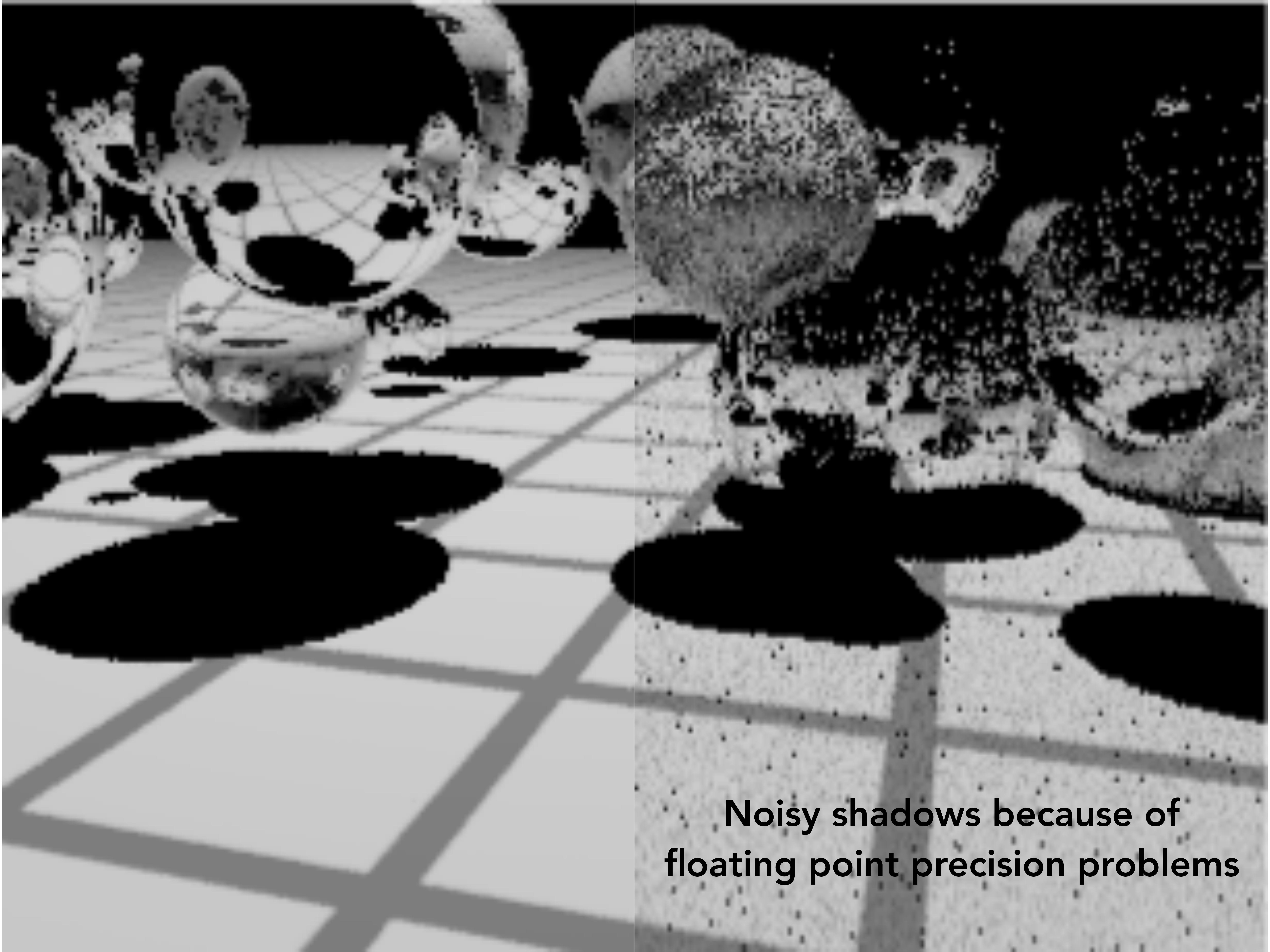


True Intersection: (1929.7203..., 1972.7897...)

Computed Intersection: (1930.4196..., 1973.5054...)

Noisy Shadows



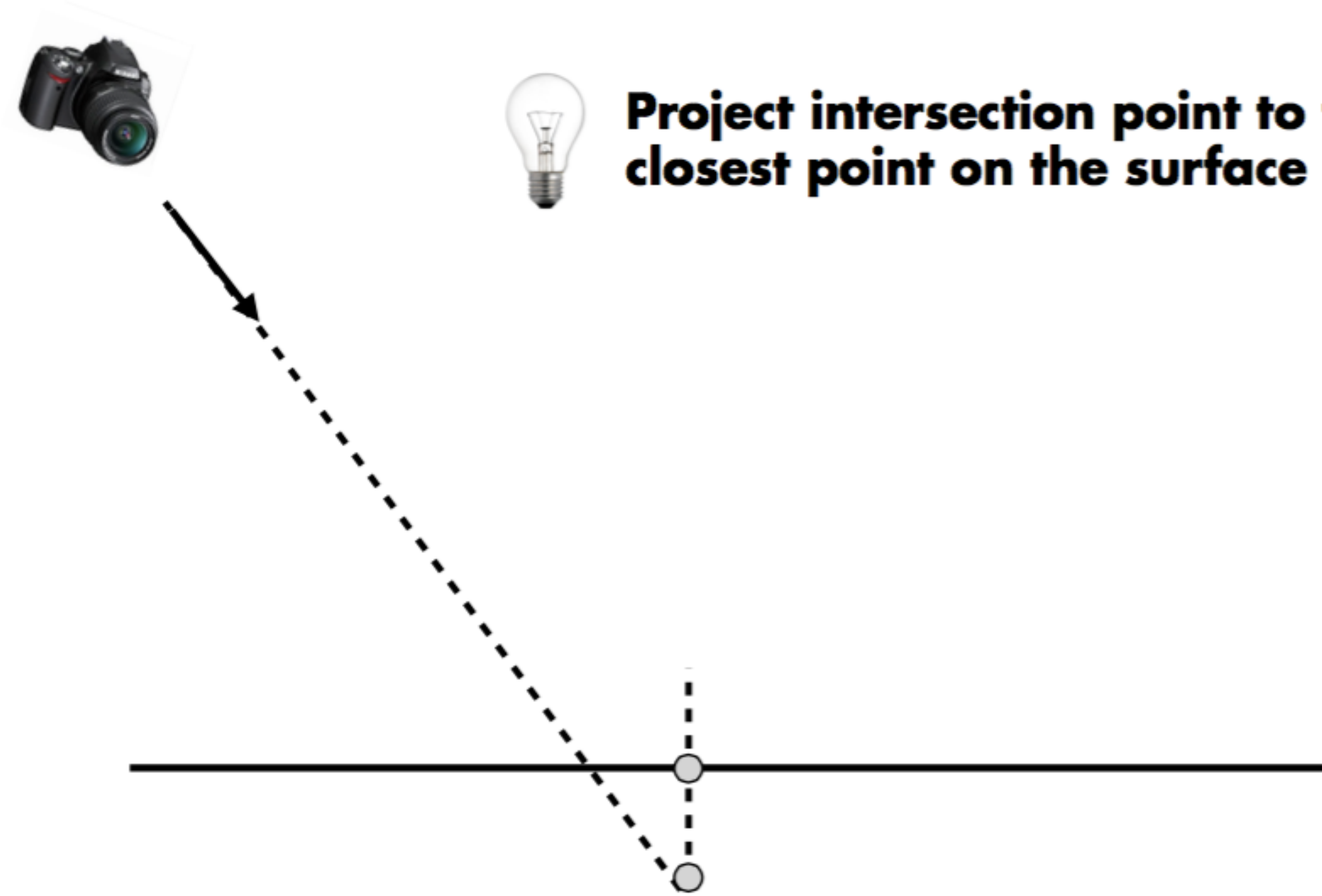

Noisy shadows because of
floating point precision problems

Floating-Point Precision Remedies

1. double (fp64) rather than float (fp32)
   - 53-bits of precision instead of 24-bits
   - Increase memory footprint
2. Ignore re-intersection with the last object hit
   - Only works for flat objects (e.g. triangles)
   - No help if model has coincident triangles
3. Offset origin along ray to ignore close intersections
   - Hard to choose offset that isn't too small or too big

Remedy: Project Intersection Point to Surface

Project intersection point to the closest point on the surface



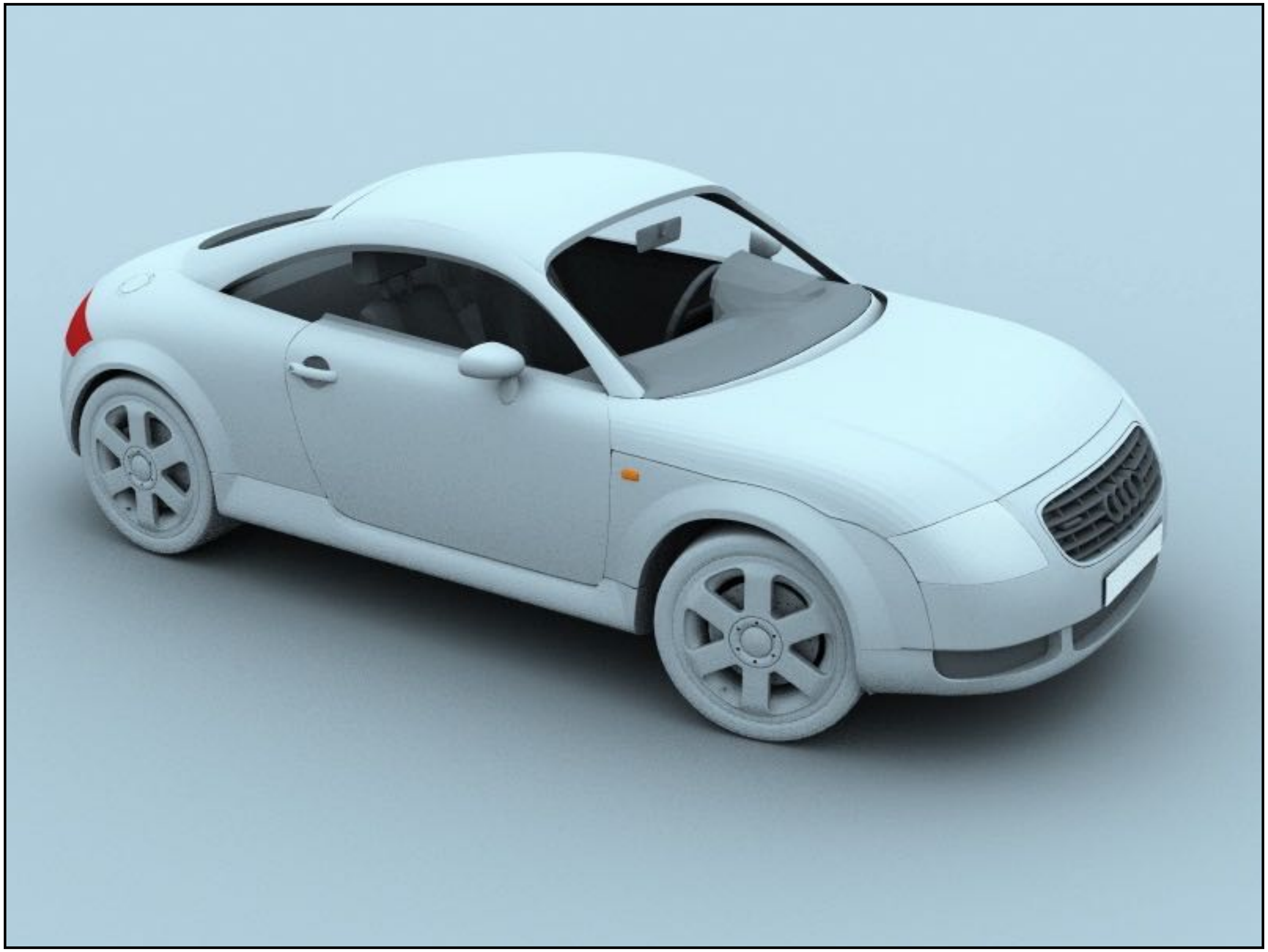
Good Scenes for Path Tracing (Diffuse, Sky Lighting)

M. Fajardo, Arnold Path Tracer

Good Scenes for Path Tracing (Diffuse, Sky Lighting)

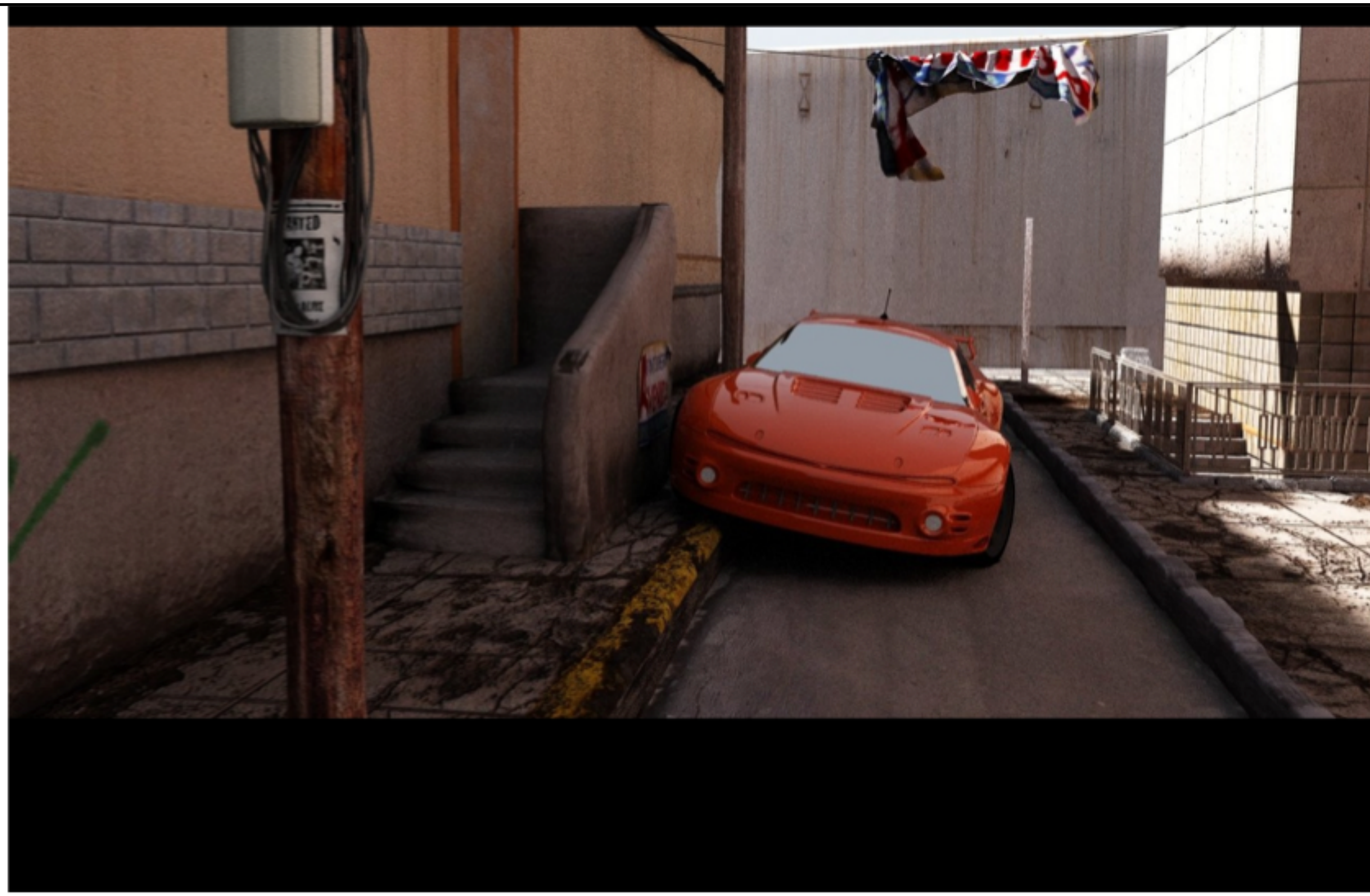
M. Fajardo, Arnold Path Tracer

Good Scenes for Path Tracing (Diffuse, Sky Lighting)



M. Fajardo, Arnold Path Tracer

Good Scenes for Path Tracing (Diffuse, Sky Lighting)

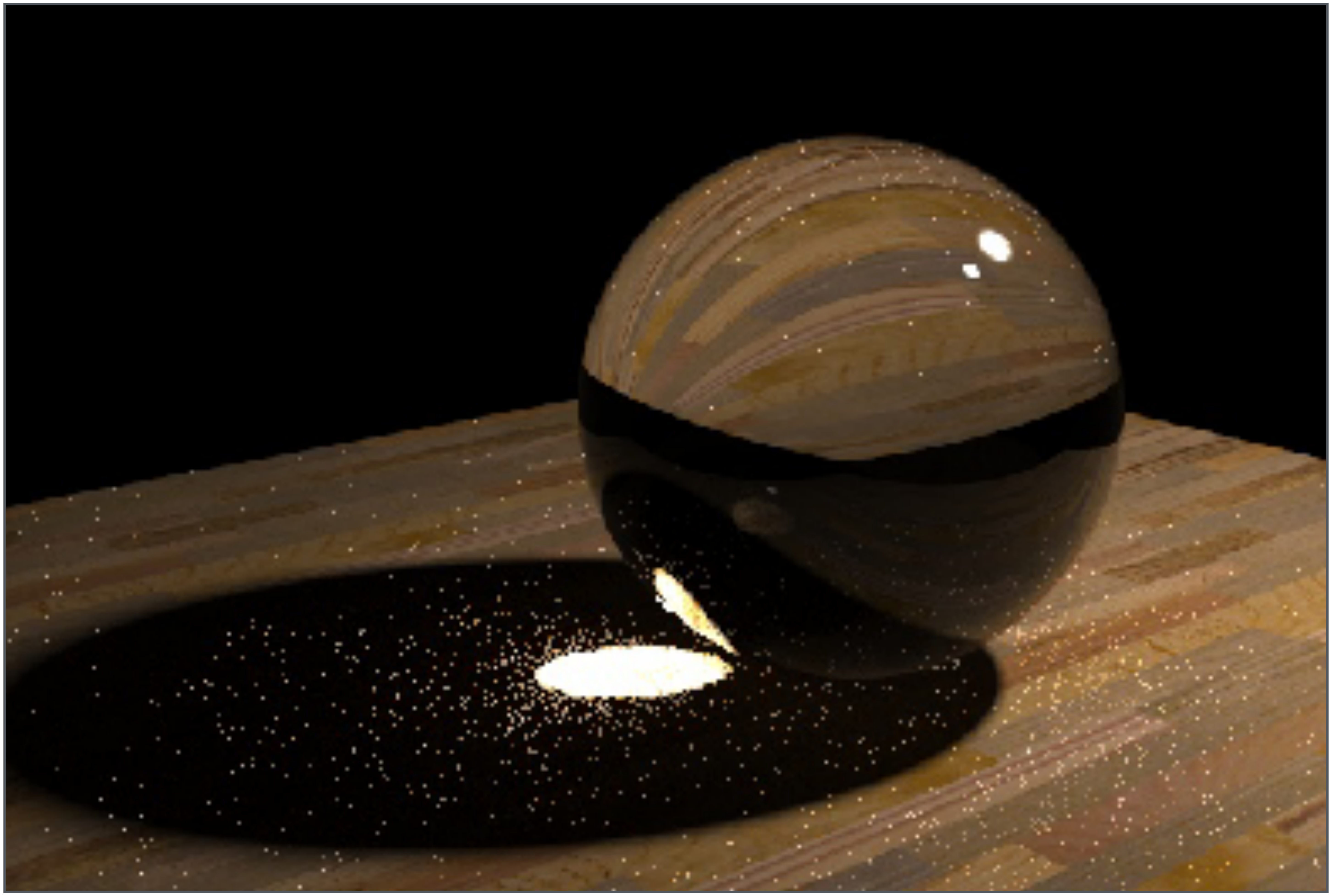


Street scene 1

1536x654, 16 paths/pixel, 2 bounces, 250,000 faces, 18 min., dual PIII-800

M. Fajardo, Arnold Path Tracer

A Challenging Scene for Path Tracing – Why?



Henrik Wann Jensen

1000 paths / pixel

Things to Remember

Global illumination challenge: recursive light transport

Reflection & rendering equations, operator notation

Neumann solution of rendering equation

- Sum successive bounces of light, infinite series

Path tracing

- Russian Roulette for unbiased finite estimate of infinite series (infinite dimensional integral)
- Partition into direct and indirect illumination
- Importance sampling of lighting and BRDF

Acknowledgments

Thanks to Matt Pharr, Pat Hanrahan and Kayvon Fatahalian for many of these slides. Thanks also to Steve Marschner for the path tracer code progression sequence.

Thanks to Weilun Sun for rendering the Cornell Box with successive bounces of light.

Thanks to Ben Mildenhall for suggestions to improve many slides.