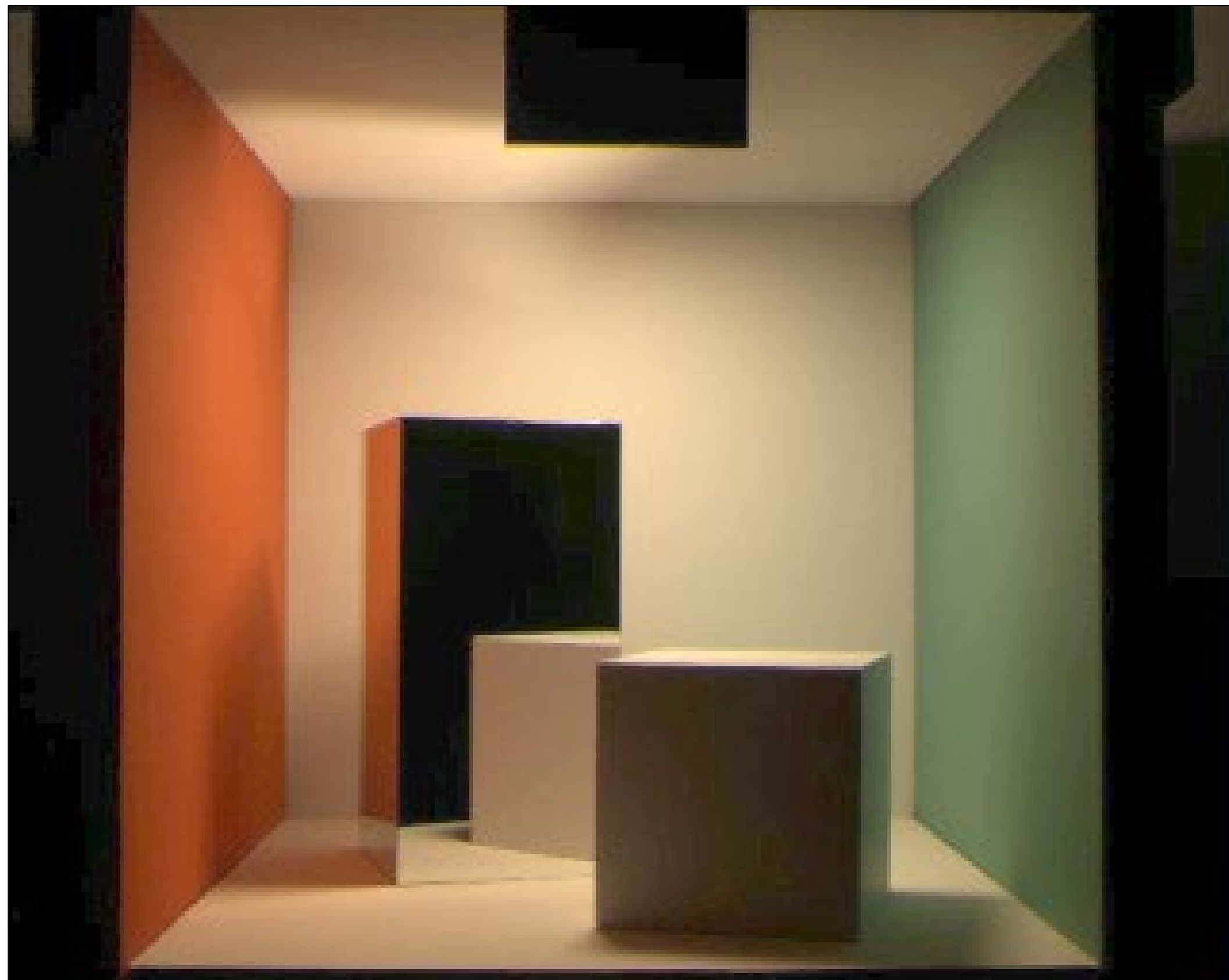


Rendering equation and path tracing



15-468, 15-668, 15-868
Physics-based Rendering
Spring 2023, Lecture 11

Course announcements

- Take-home quiz 6 posted, due Tuesday 3/14 at 23:59.
- Programming assignment 3 posted, due Friday 3/17 at 23:59.
 - How many of you have looked at/started/finished it?
 - Any questions?
- Double recitation today.

Overview of today's lecture

- Rendering equation.
- Path tracing with next-event estimation.

Slide credits

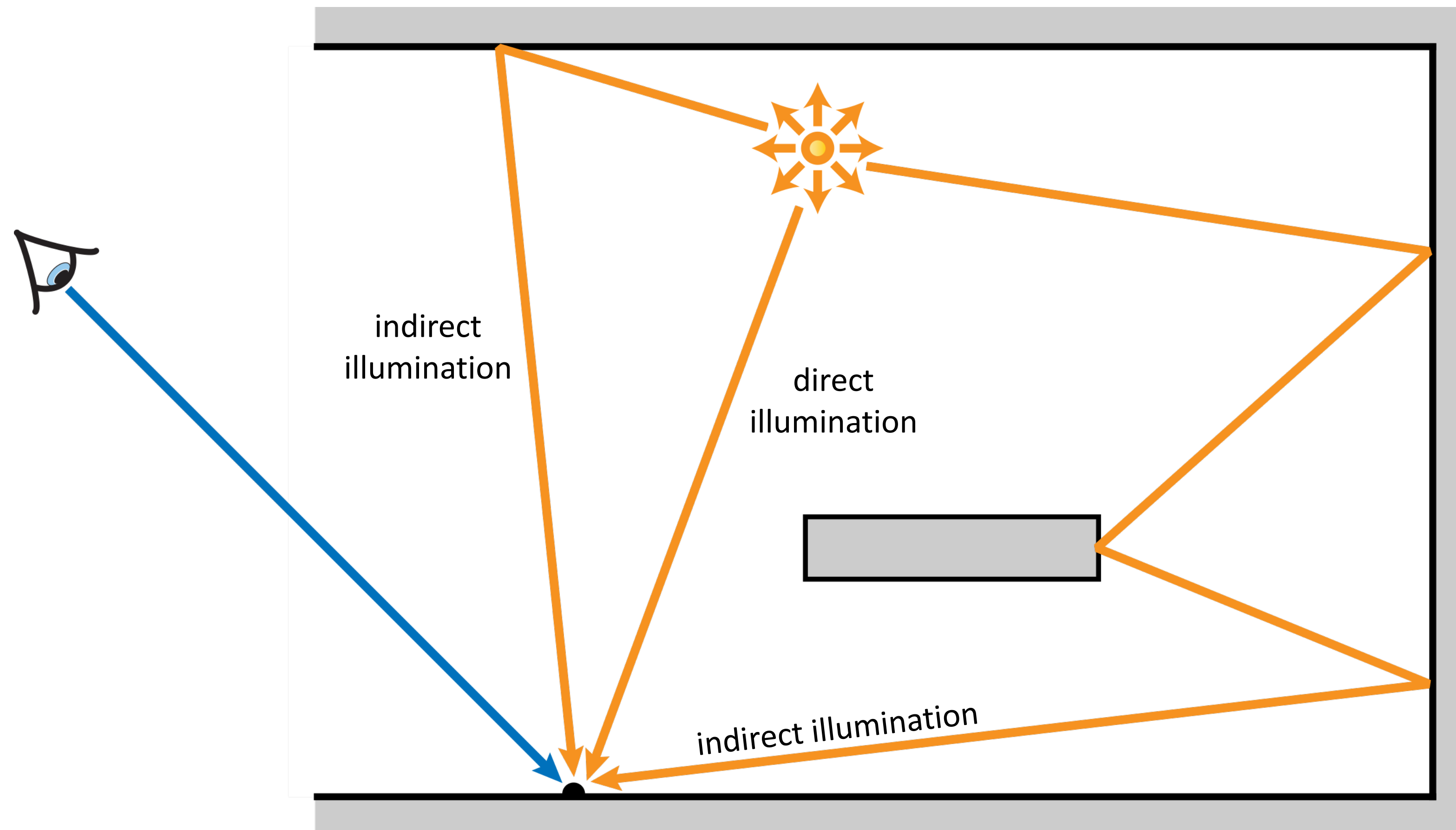
Most of these slides were directly adapted from:

- Wojciech Jarosz (Dartmouth).

Direct vs. Indirect Illumination

Where does L_i
“come from”?

$$L_r(\mathbf{x}, \vec{\omega}_r) = \int_{H^2} f_r(\mathbf{x}, \vec{\omega}_i, \vec{\omega}_r) L_i(\mathbf{x}, \vec{\omega}_i) \cos \theta_i d\vec{\omega}_i$$



Direct vs. Indirect Illumination

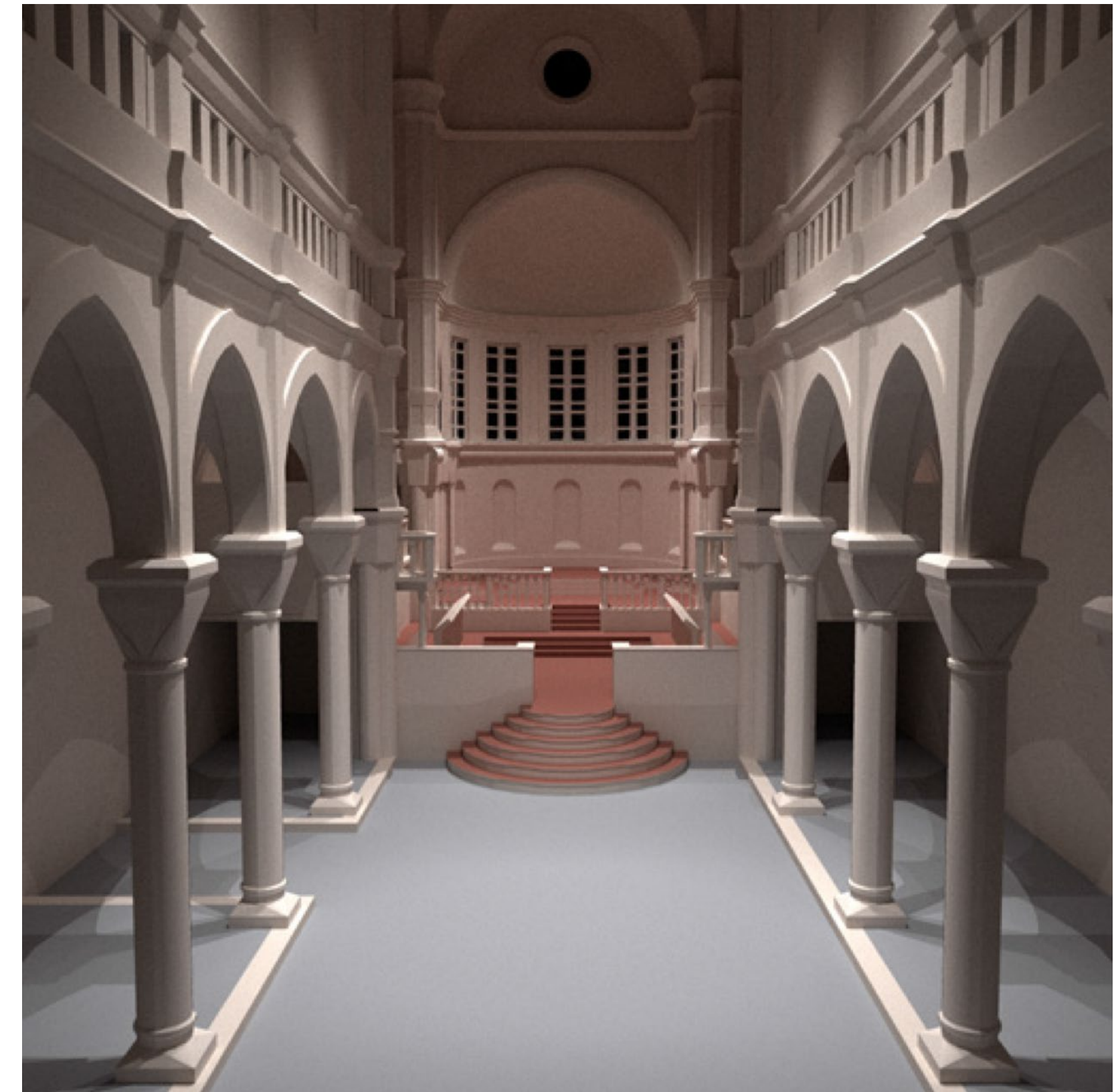
Direct illumination



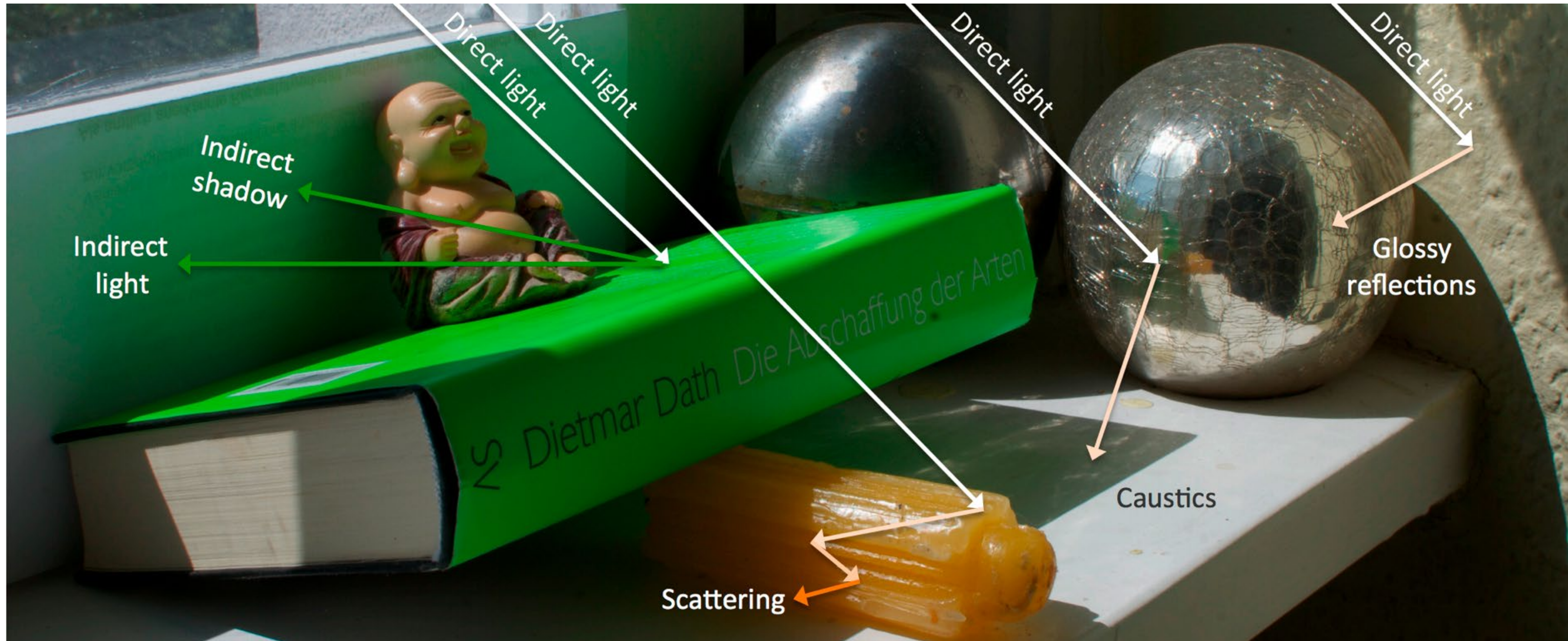
Indirect illumination



Direct + indirect illumination

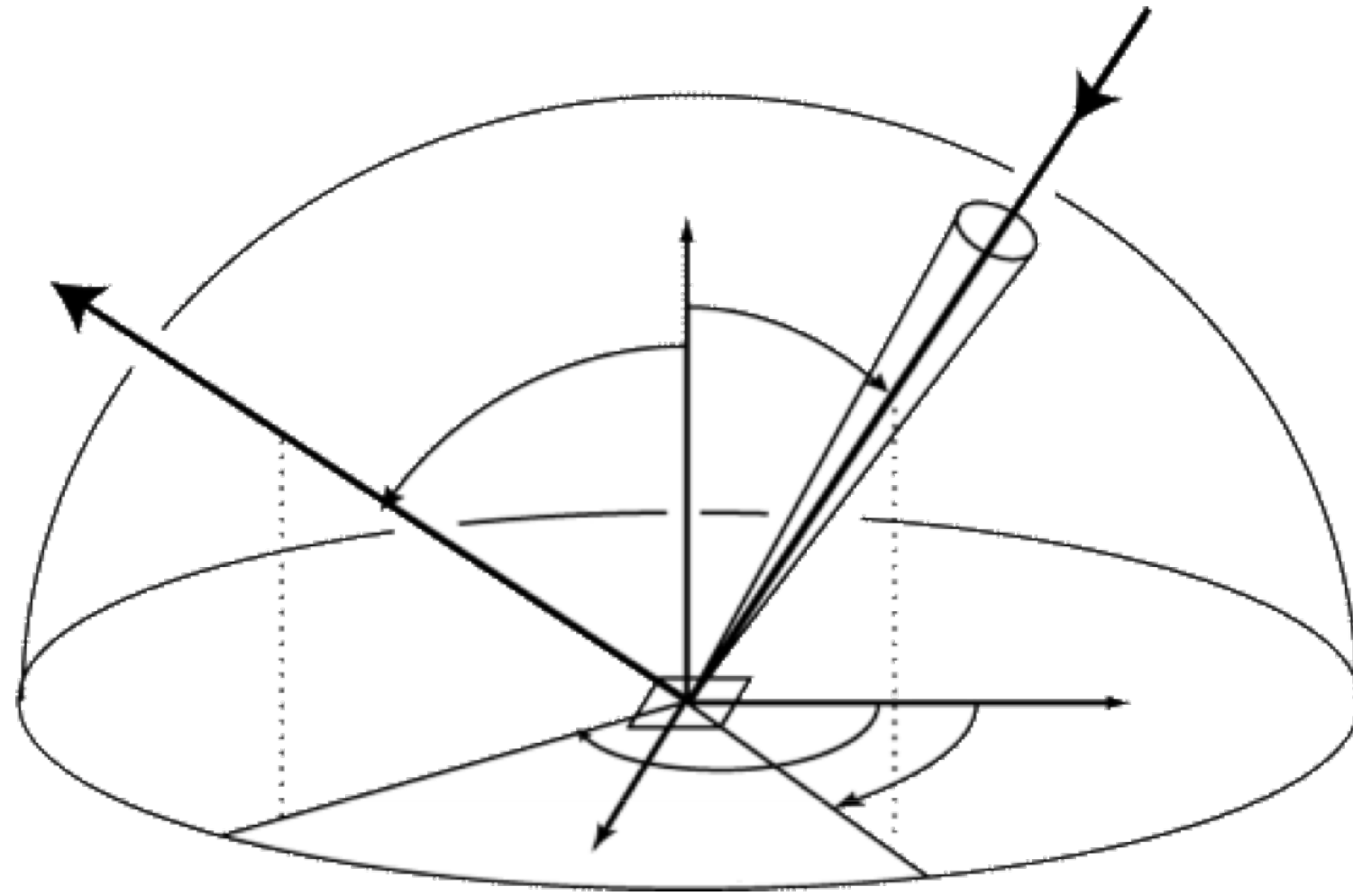


All-in-One!



Reflection Equation

Reflected radiance is the weighted integral of incident radiance



$$L_r(\mathbf{x}, \vec{\omega}_r) = \int_{H^2} f_r(\mathbf{x}, \vec{\omega}_i, \vec{\omega}_r) L_i(\mathbf{x}, \vec{\omega}_i) \cos \theta_i d\vec{\omega}_i$$

Rendering Equation

James Kajiya, “The Rendering Equation.”
SIGGRAPH 1986.

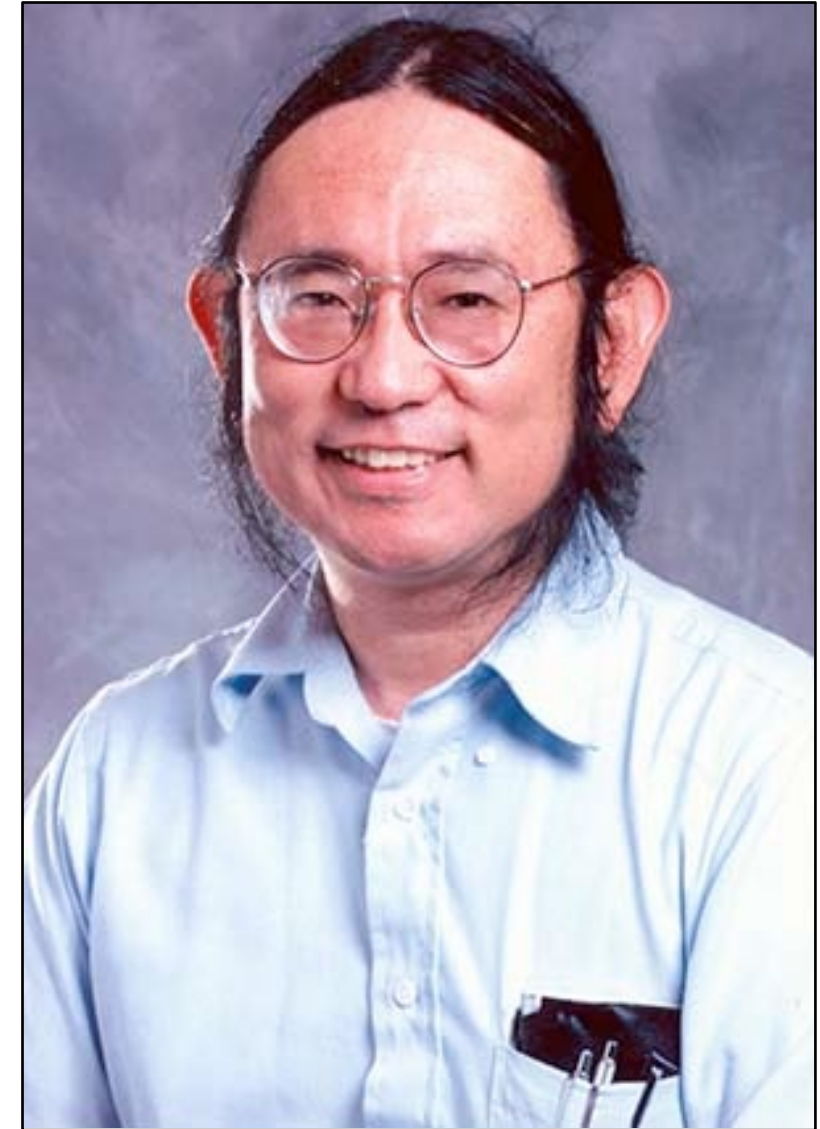
Energy equilibrium:

$$L_o(\mathbf{x}, \vec{\omega}_o) = L_e(\mathbf{x}, \vec{\omega}_o) + L_r(\mathbf{x}, \vec{\omega}_o)$$

↑
outgoing

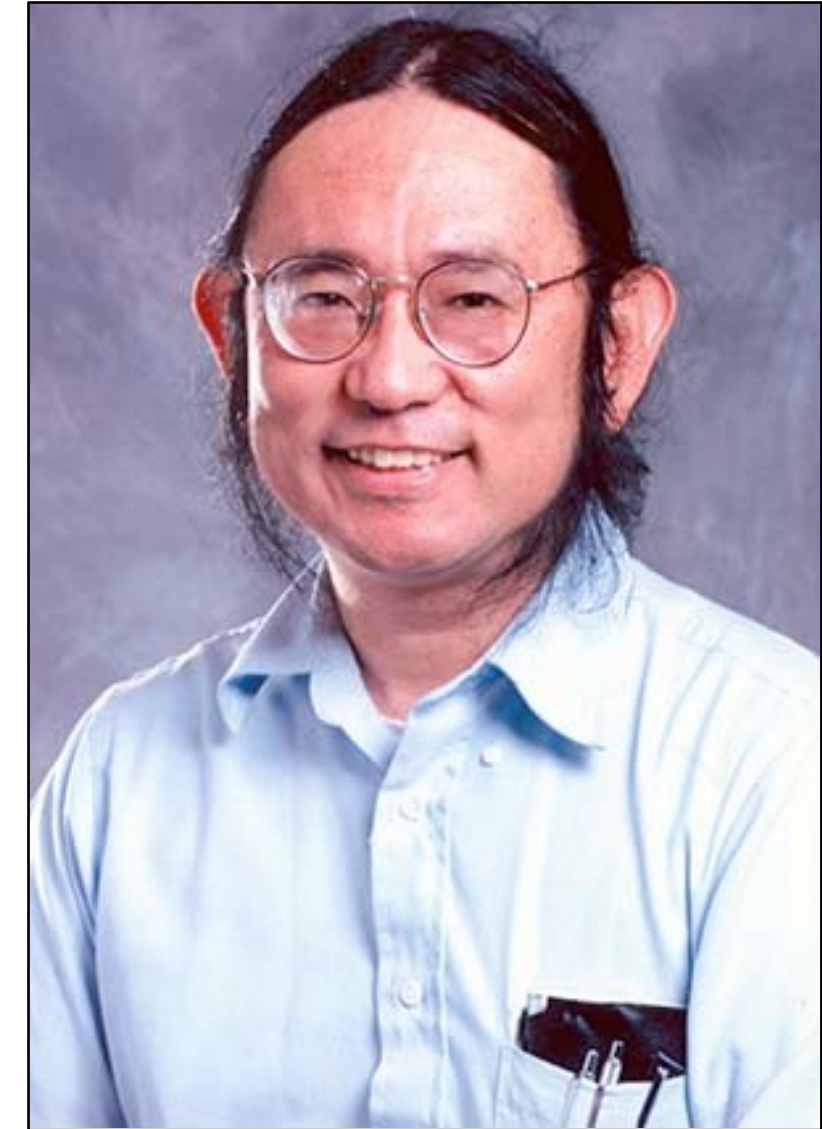
↑
emitted

↑
reflected



Rendering Equation

James Kajiya, "The Rendering Equation."
SIGGRAPH 1986.



Energy equilibrium:

$$\boxed{L_o}(\mathbf{x}, \vec{\omega}_o) = L_e(\mathbf{x}, \vec{\omega}_o) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}_i, \vec{\omega}_o) \boxed{L_i}(\mathbf{x}, \vec{\omega}_i) \cos \theta_i d\vec{\omega}_i$$

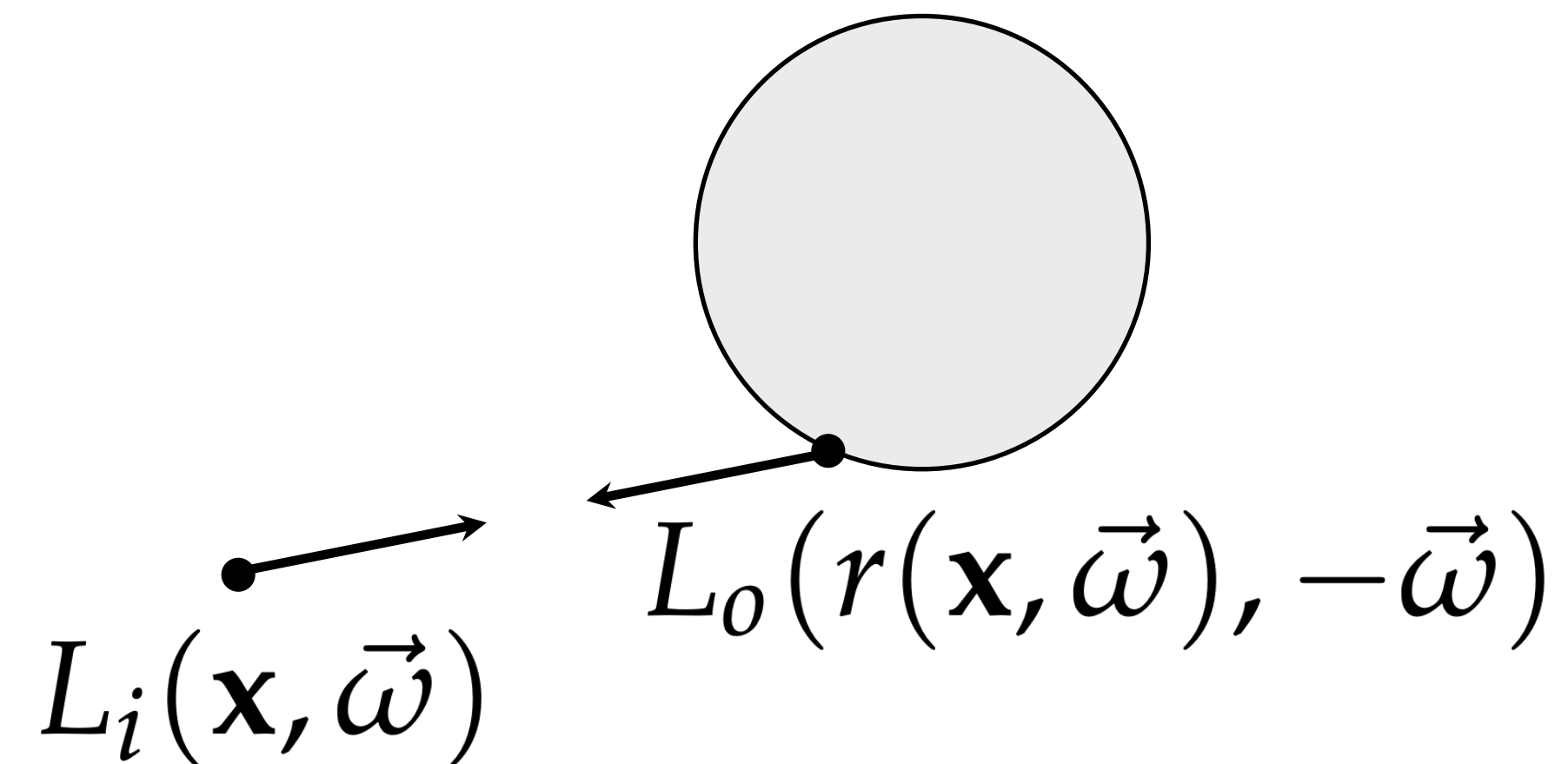
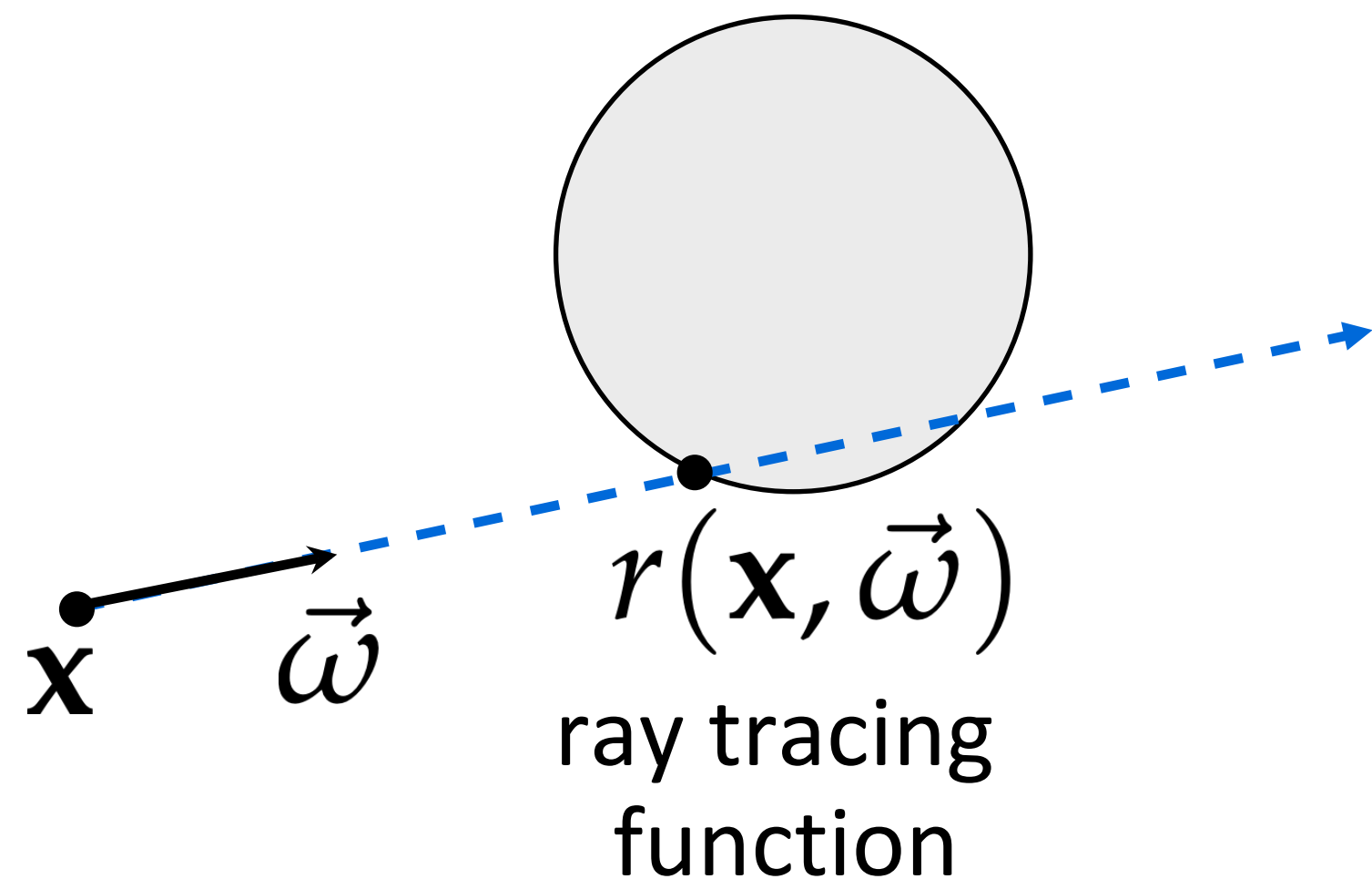
outgoing emitted reflected

Light Transport

In free-space/vacuum, radiance is constant along rays

We can relate incoming radiance to outgoing radiance

$$L_i(\mathbf{x}, \vec{\omega}) = L_o(r(\mathbf{x}, \vec{\omega}), -\vec{\omega})$$



Rendering Equation

ray tracing
function

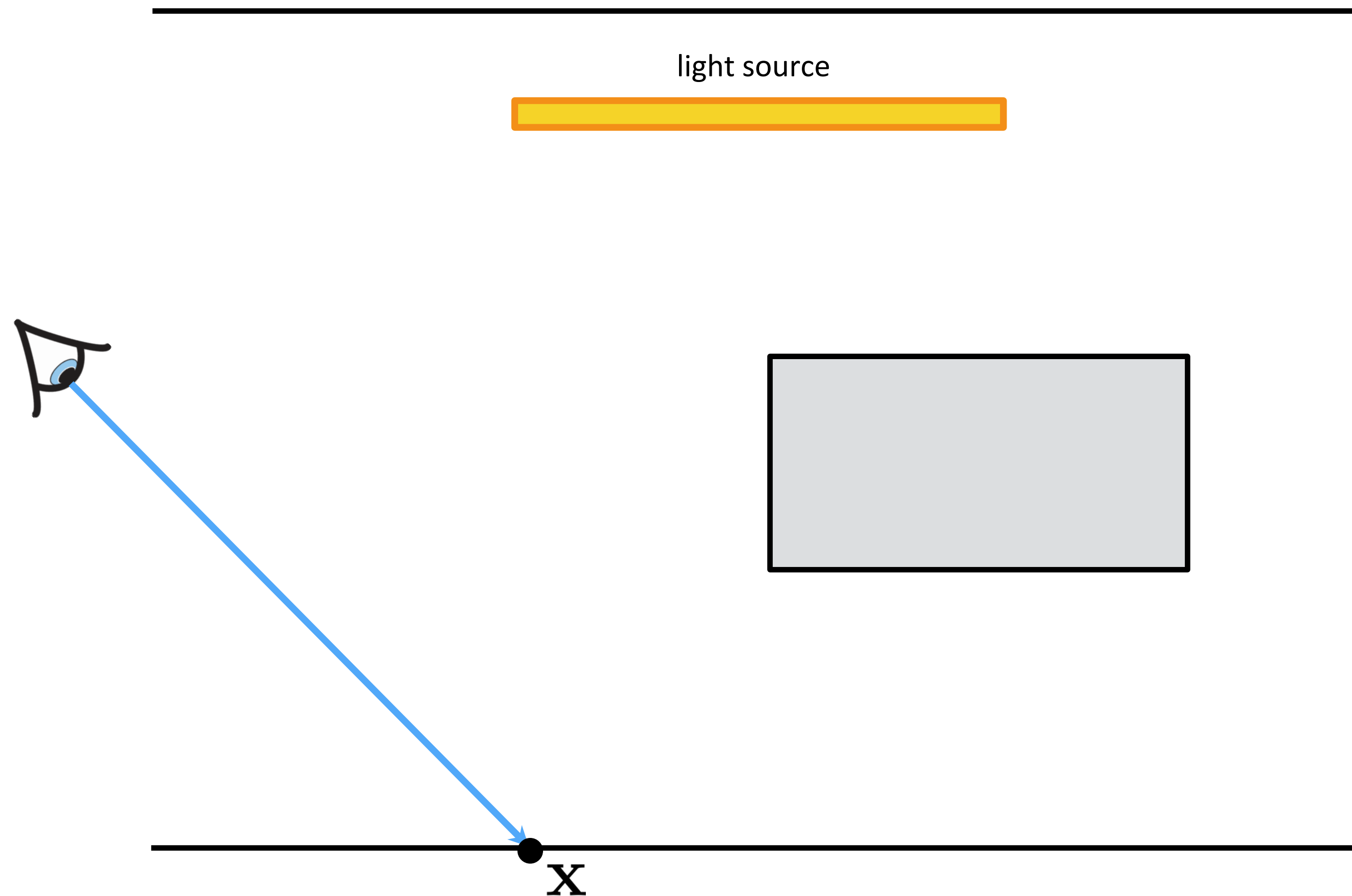
$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos \theta' d\vec{\omega}'$$

Only outgoing radiance on both sides

- we drop the “o” subscript
- Fredholm equation of the second kind (recursive)
- Extensive operator-theoretic study (that we will not cover here, but great reading group material)

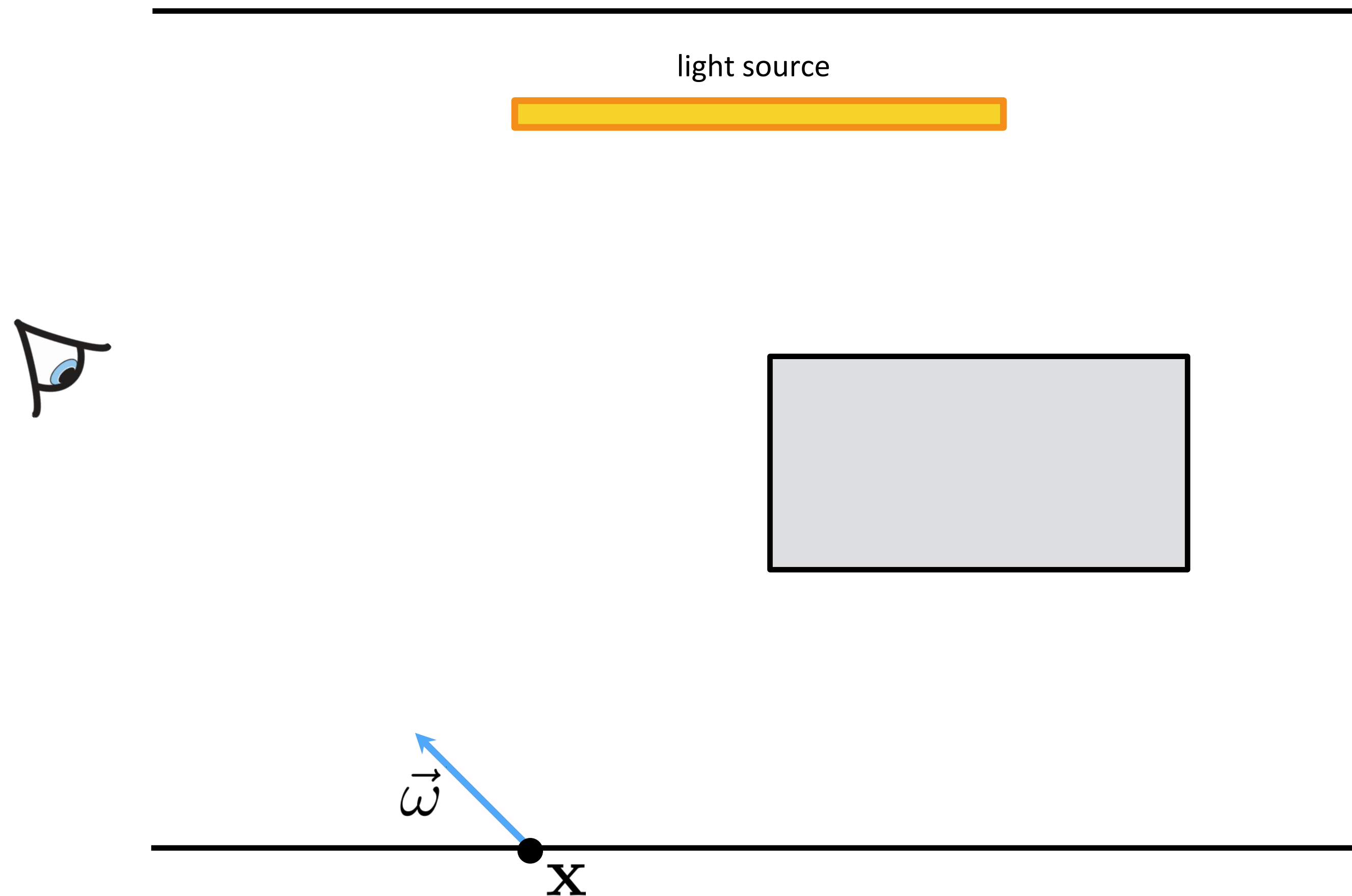
Rendering Equation

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos \theta' d\vec{\omega}'$$



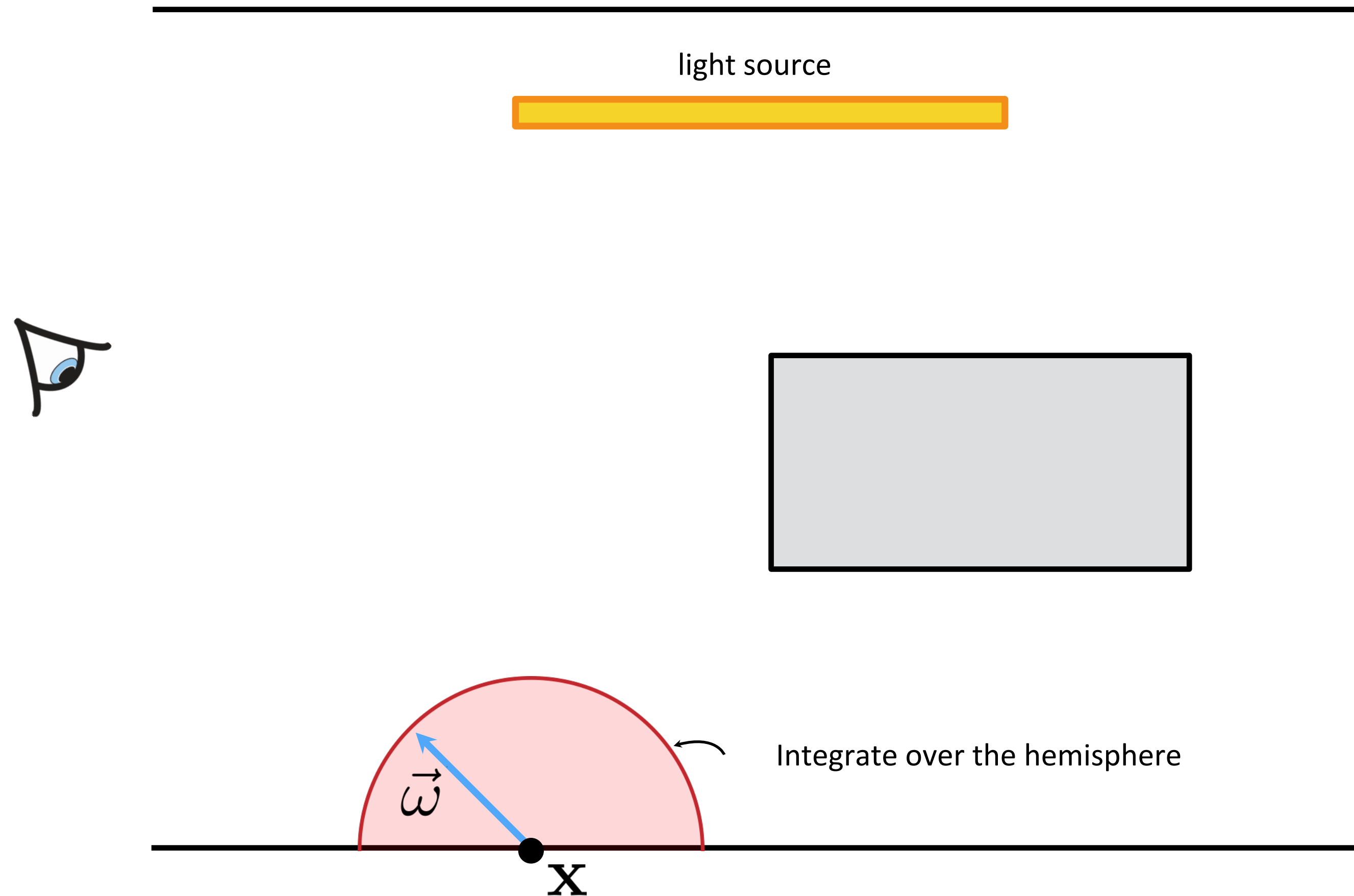
Rendering Equation

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos \theta' d\vec{\omega}'$$



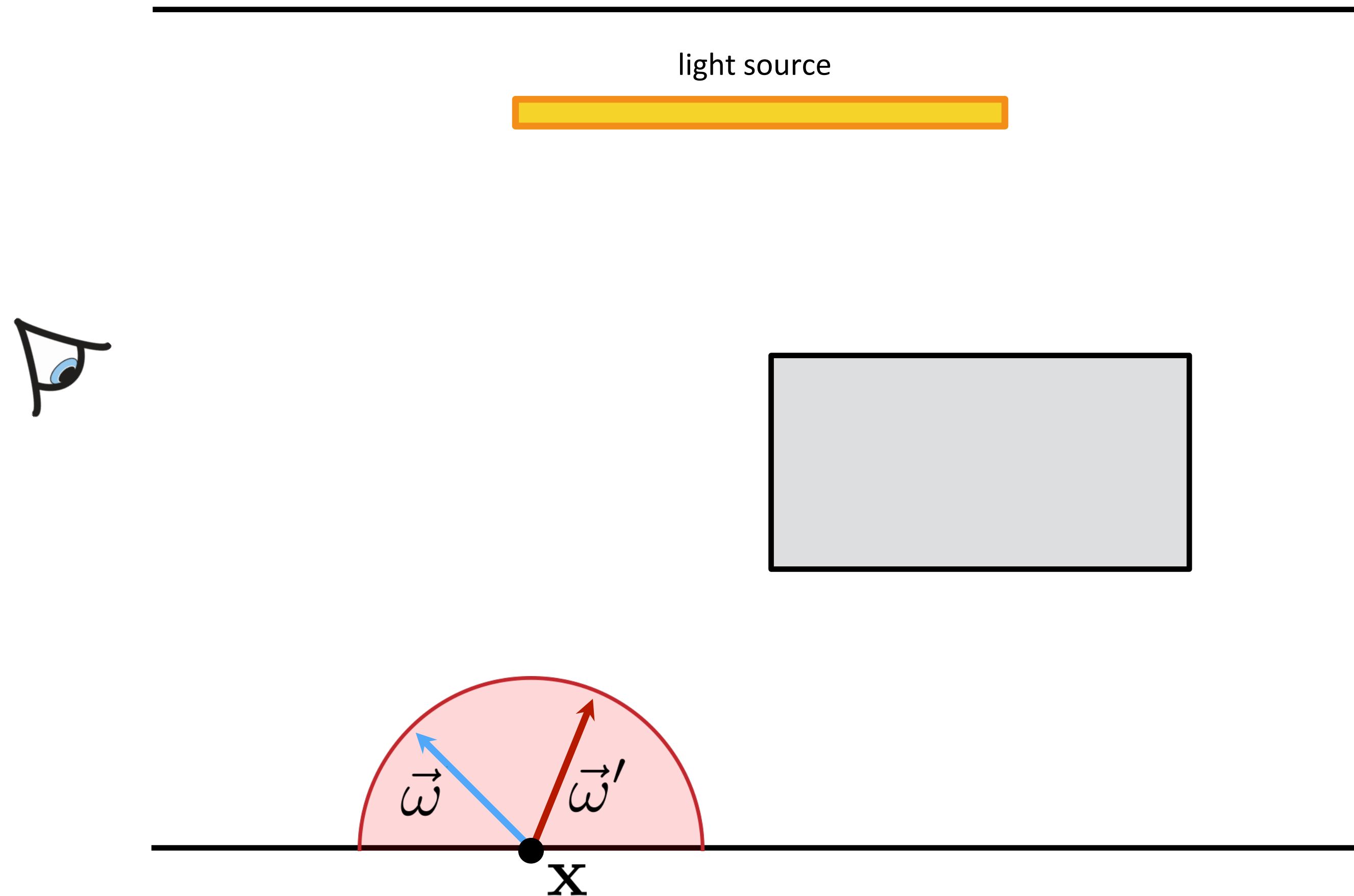
Rendering Equation

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos \theta' d\vec{\omega}'$$



Rendering Equation

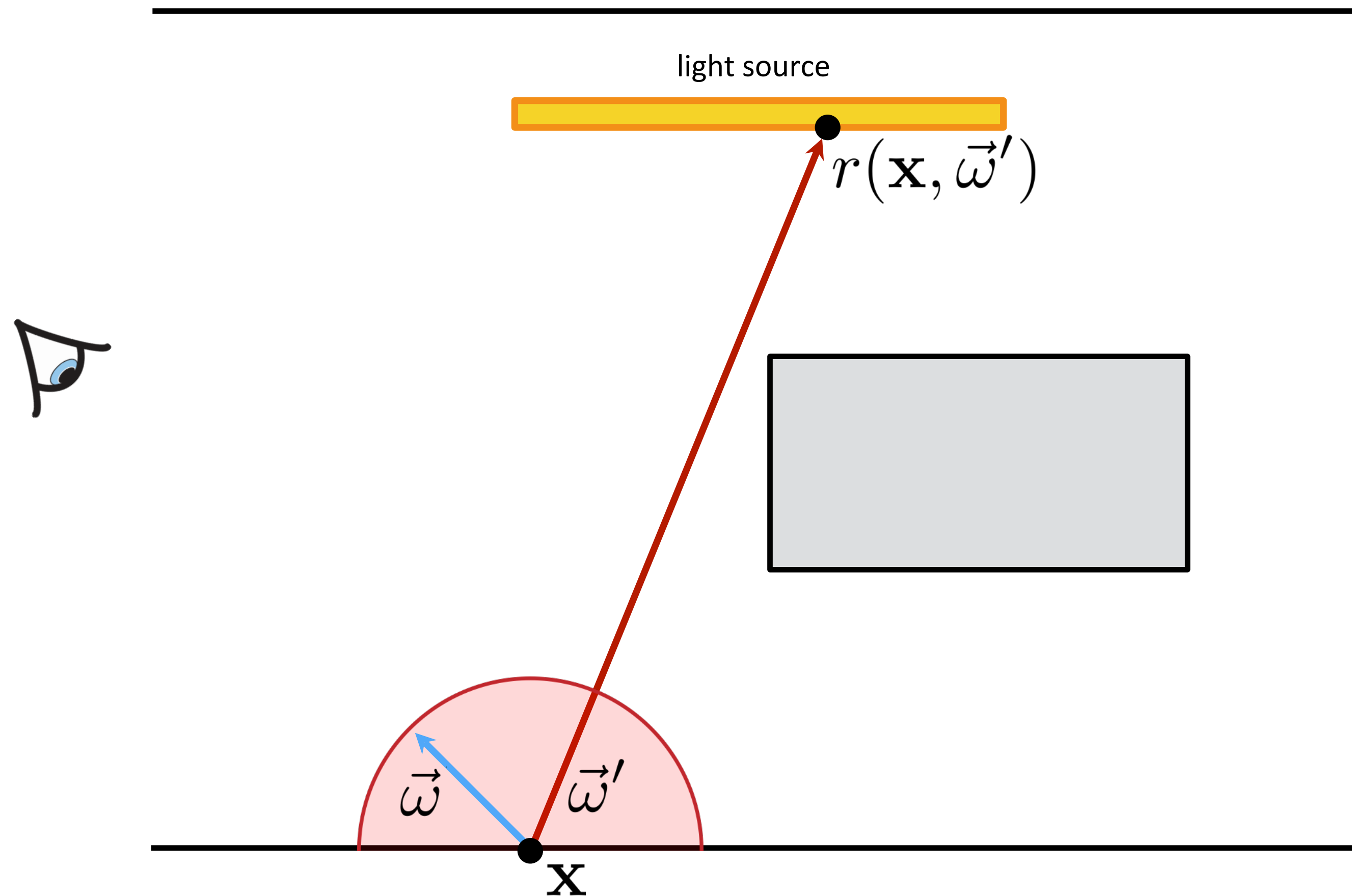
$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos \theta' d\vec{\omega}'$$



Rendering Equation

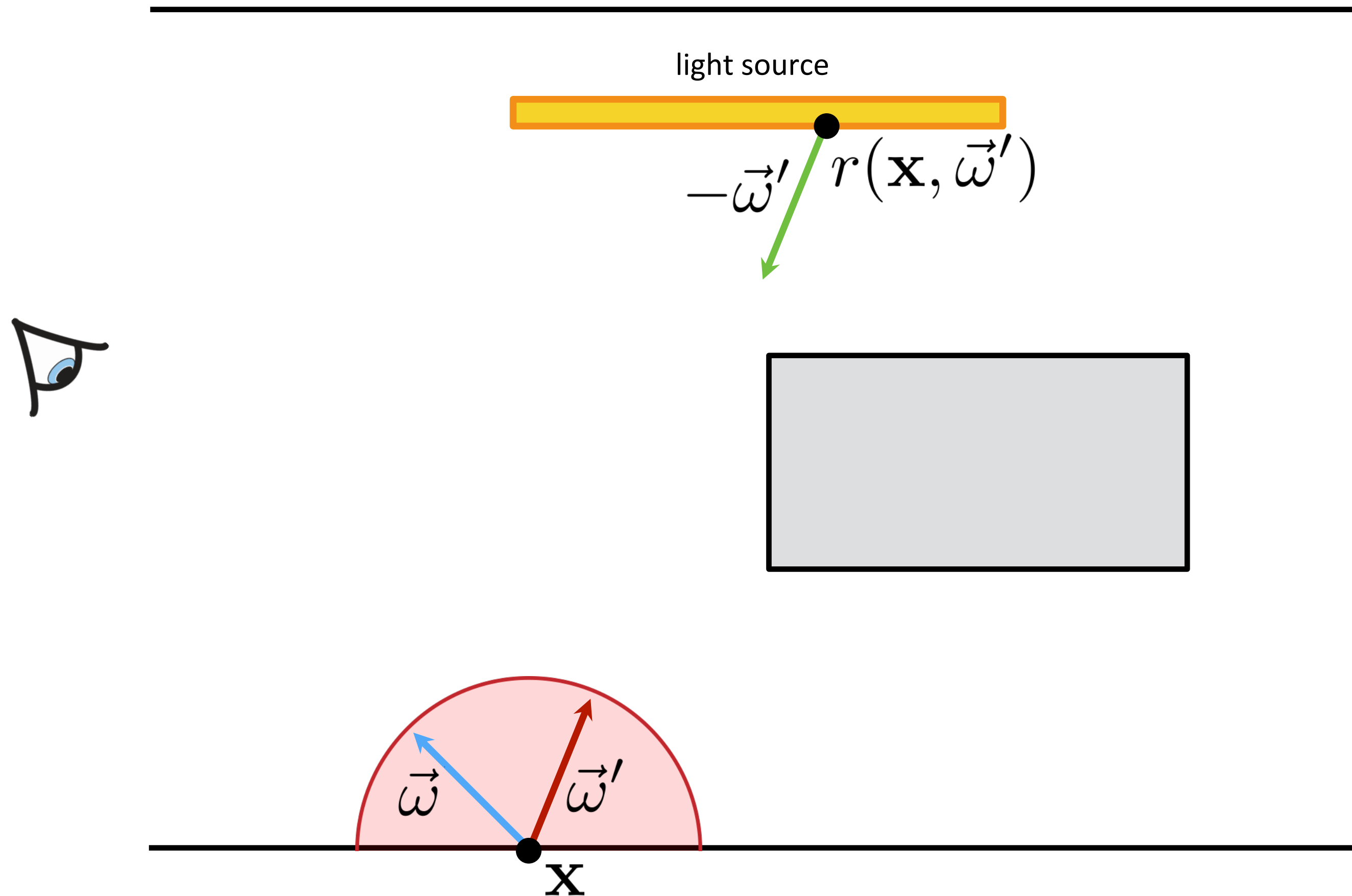
ray tracing
function

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos \theta' d\vec{\omega}'$$



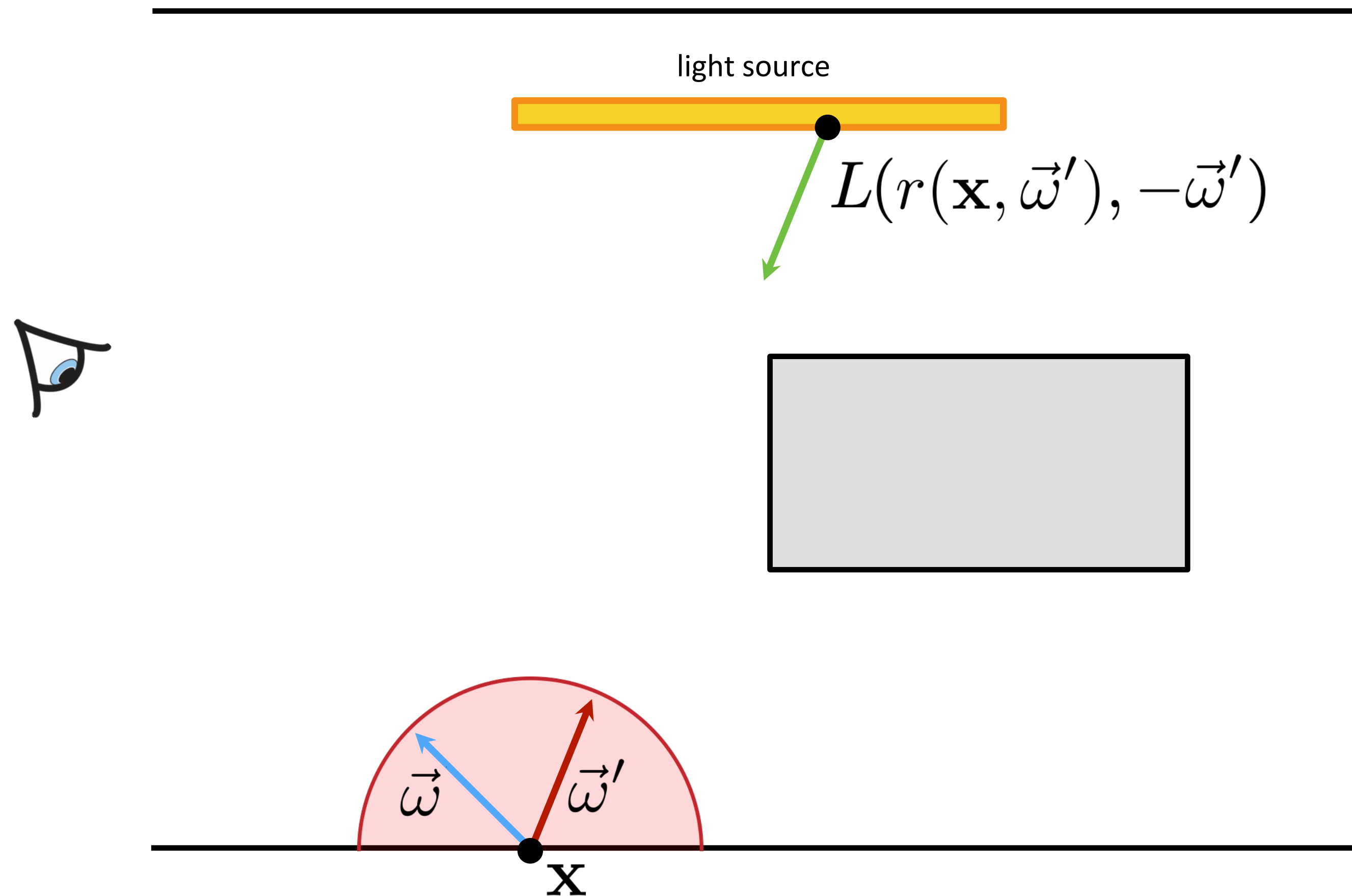
Rendering Equation

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos \theta' d\vec{\omega}'$$



Rendering Equation

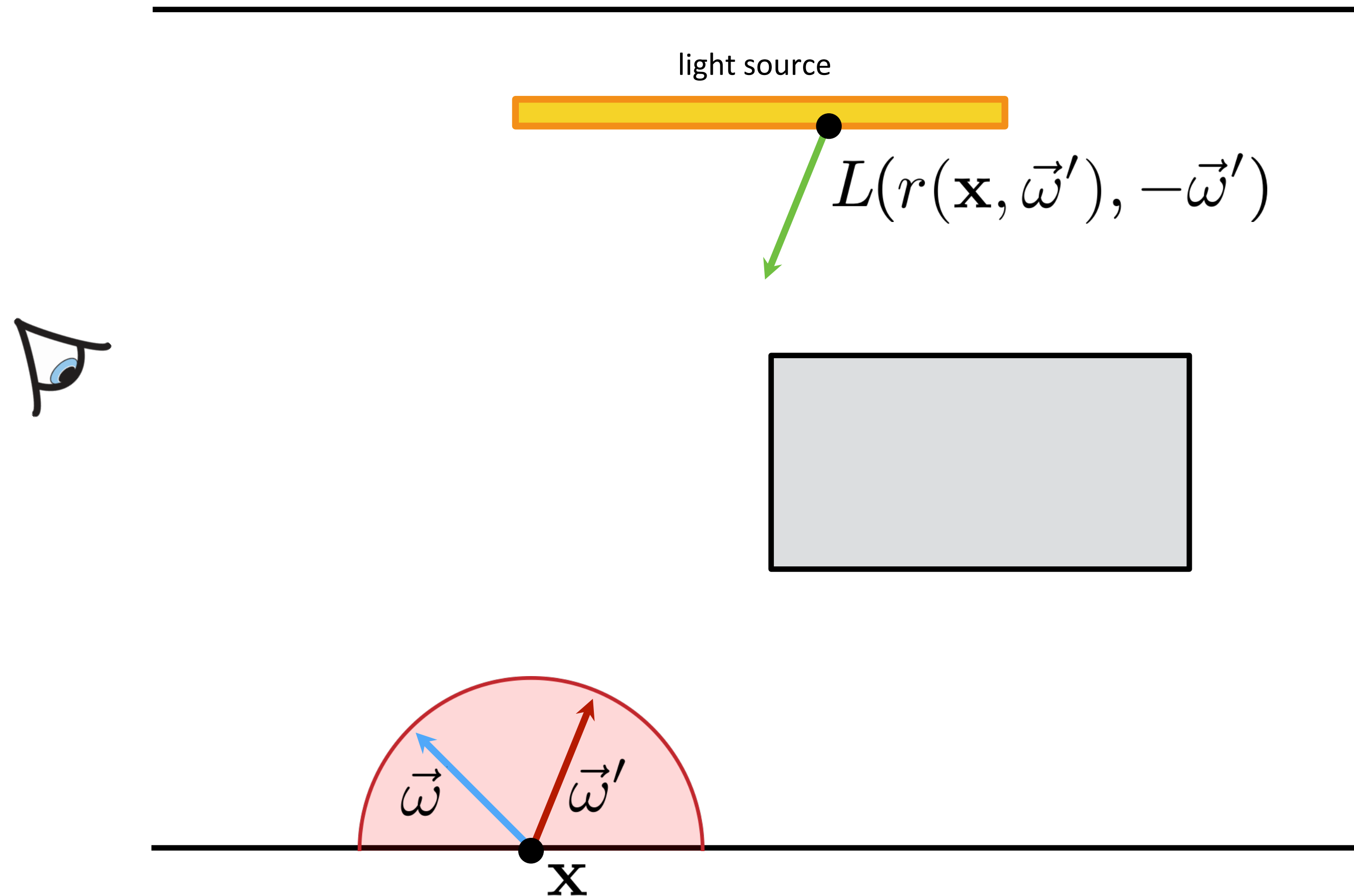
$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos \theta' d\vec{\omega}'$$



Rendering Equation

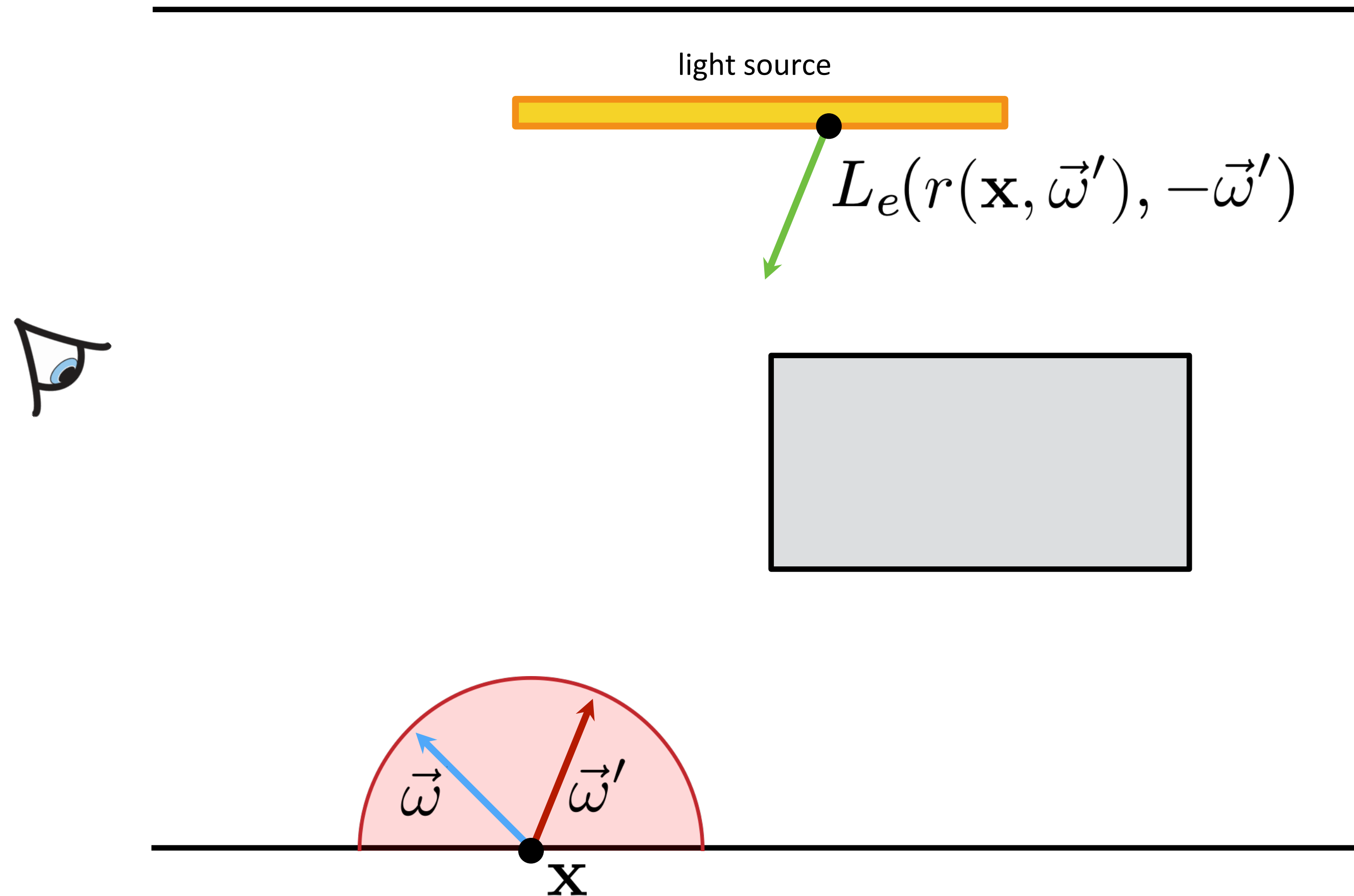
$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos \theta' d\vec{\omega}'$$

recursion



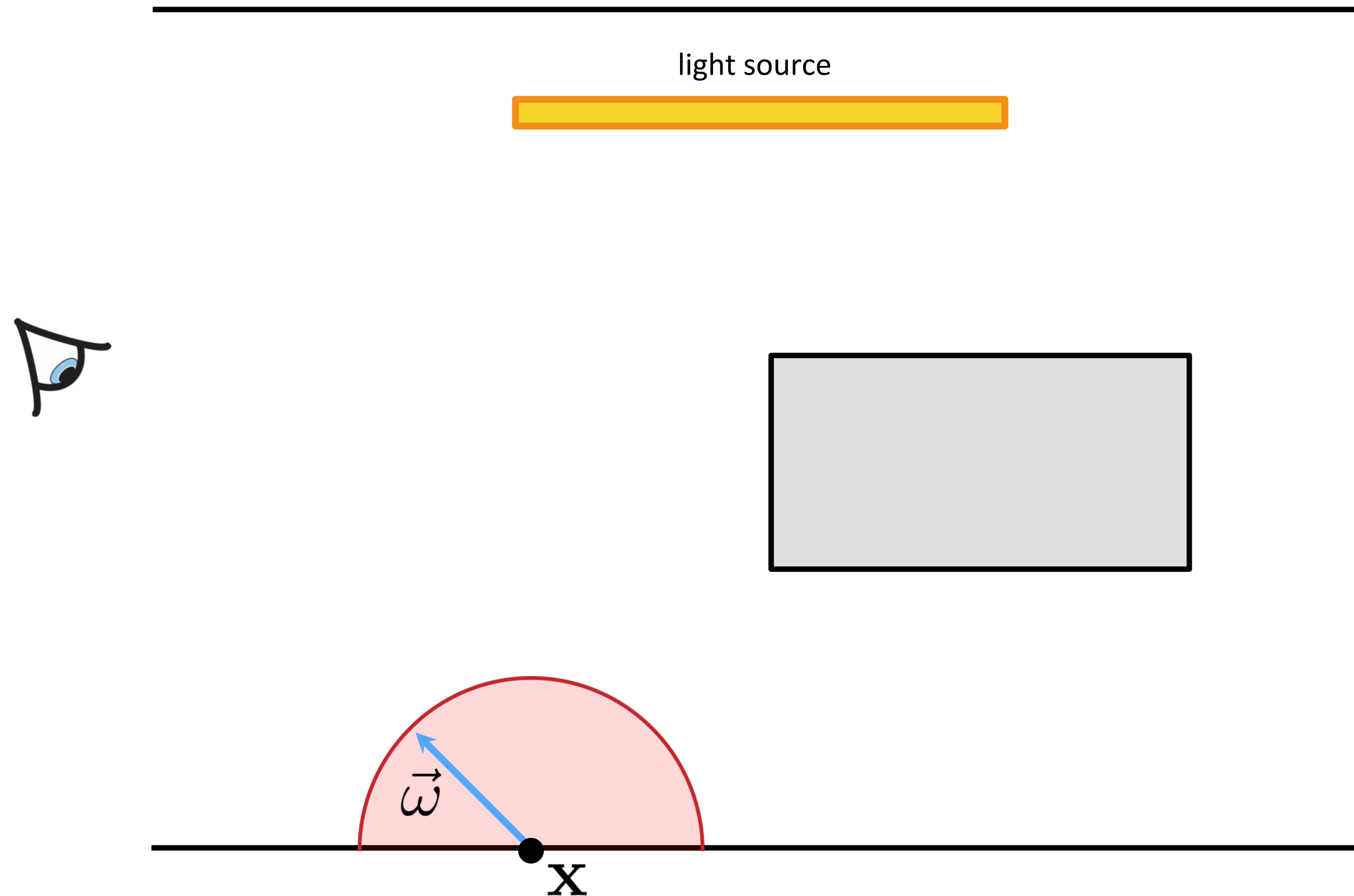
Rendering Equation

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos \theta' d\vec{\omega}'$$



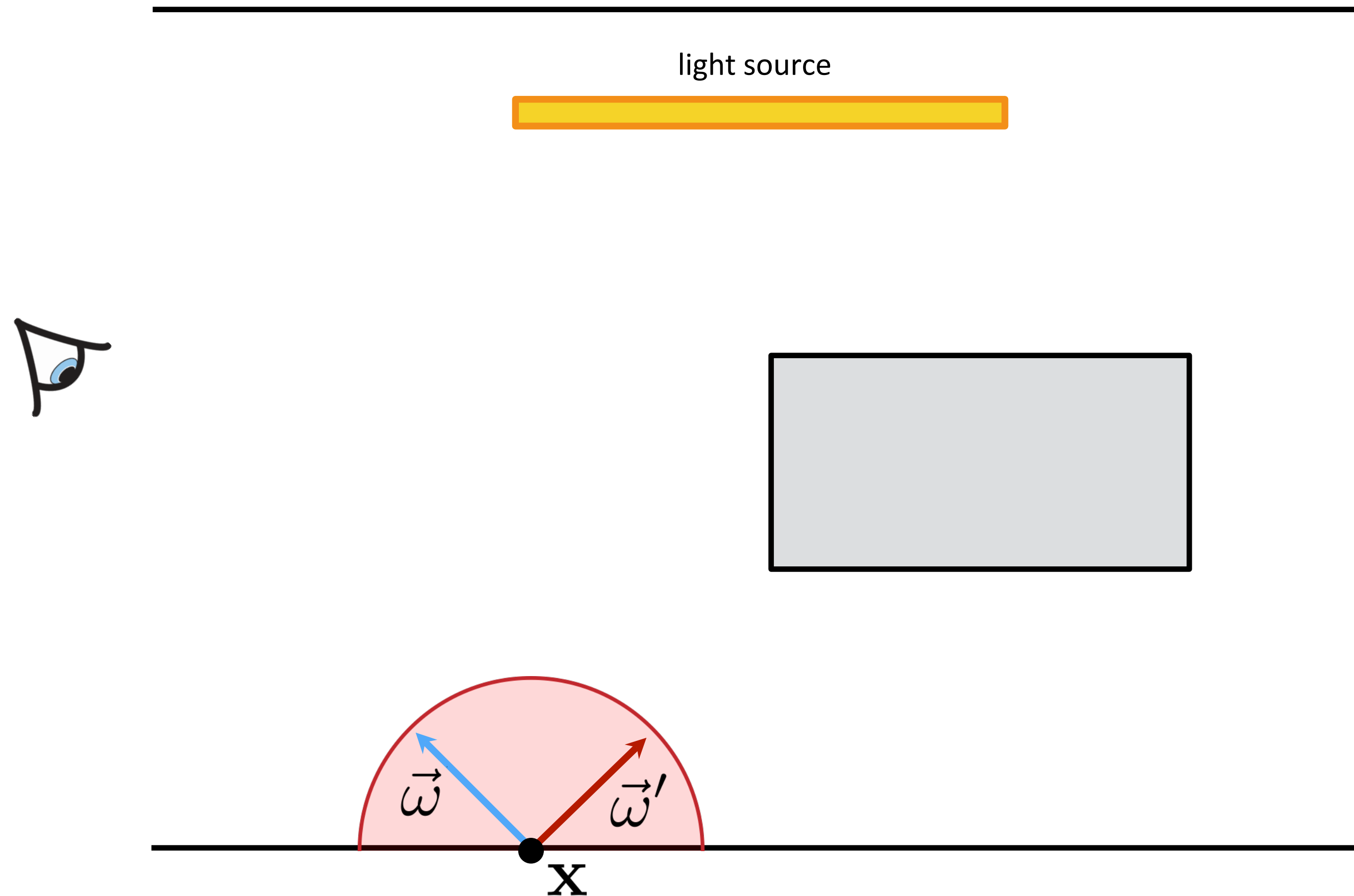
Rendering Equation

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos \theta' d\vec{\omega}'$$



Rendering Equation

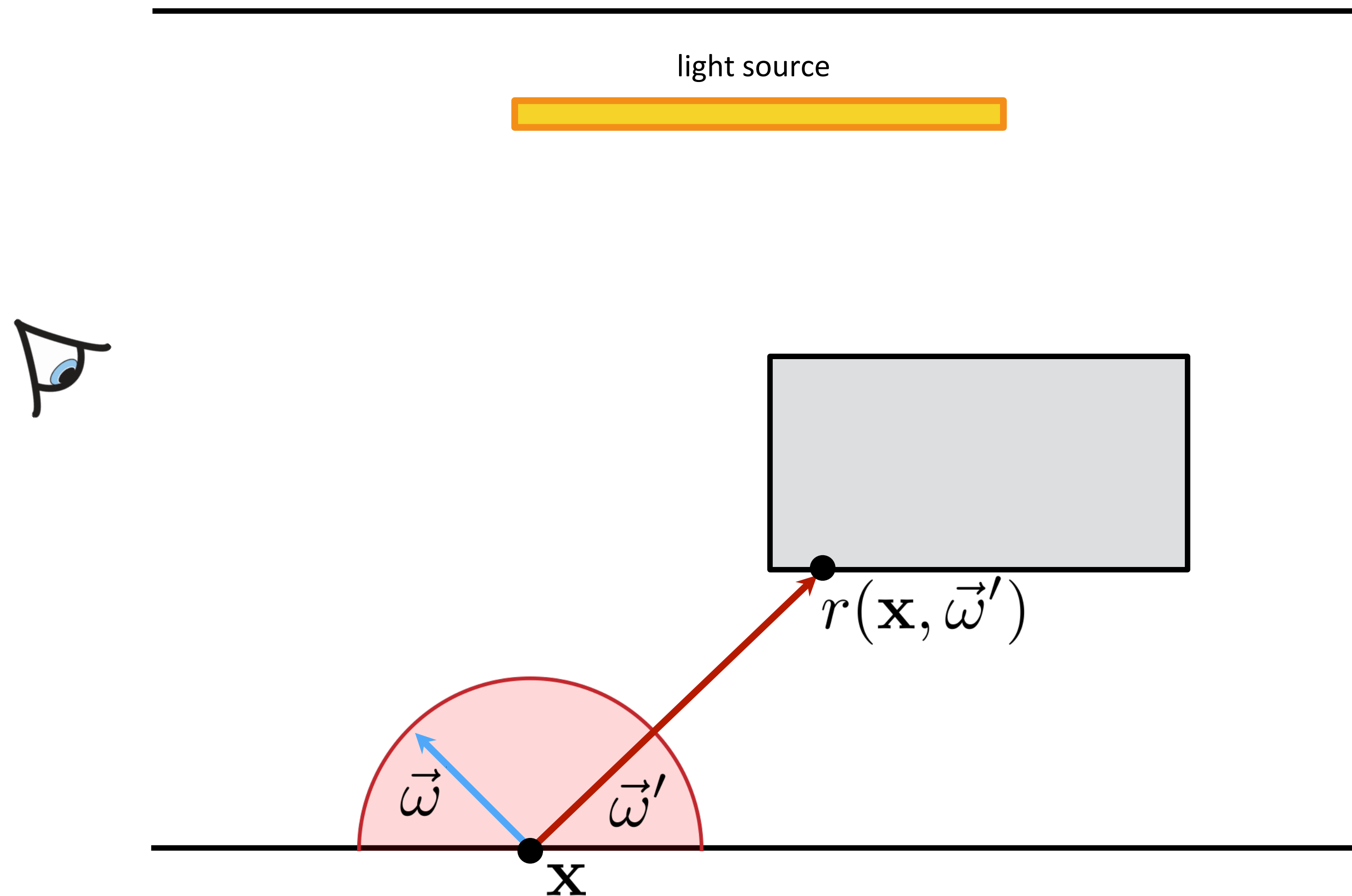
$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos \theta' d\vec{\omega}'$$



Rendering Equation

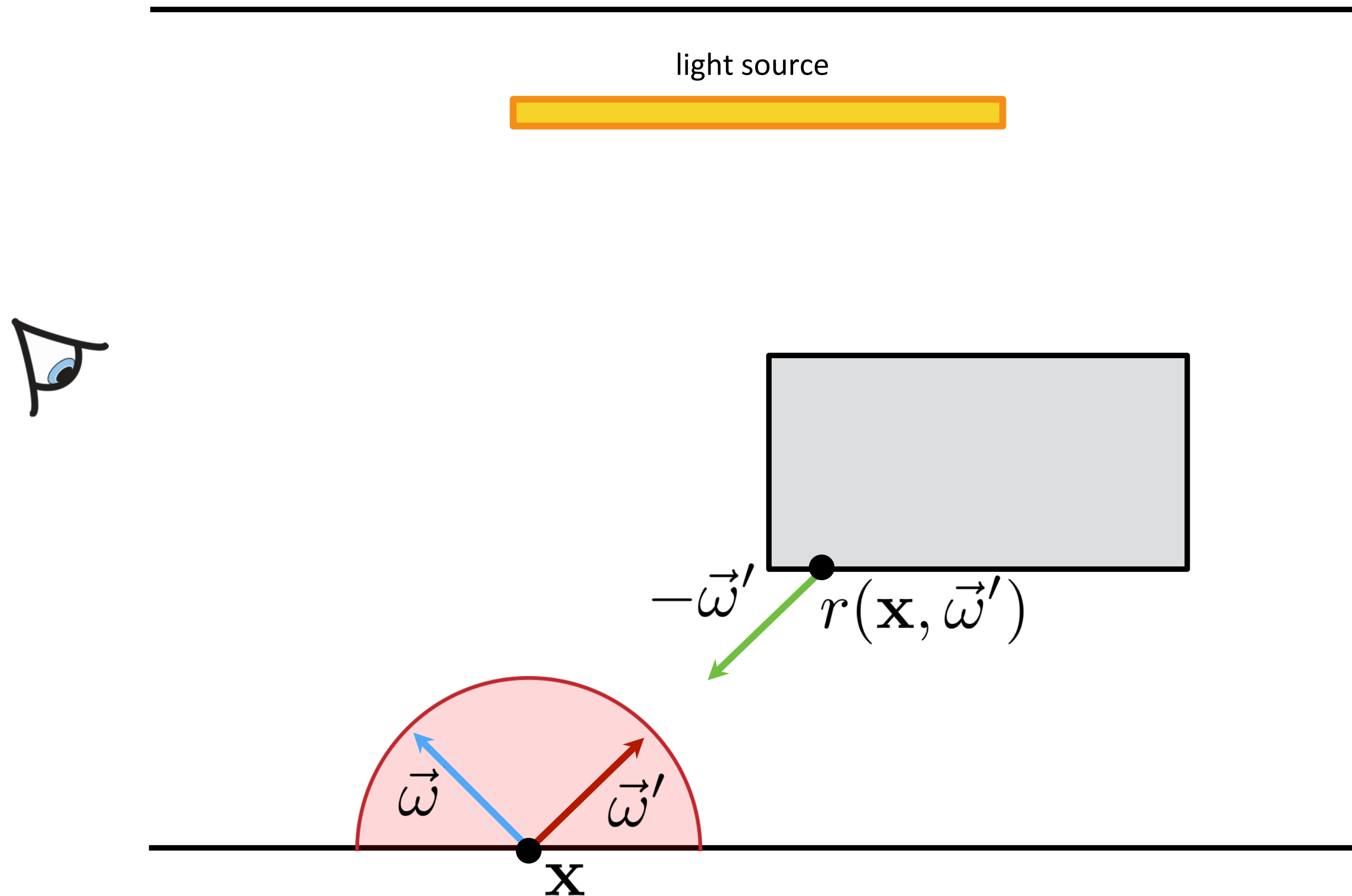
ray tracing
function

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos \theta' d\vec{\omega}'$$



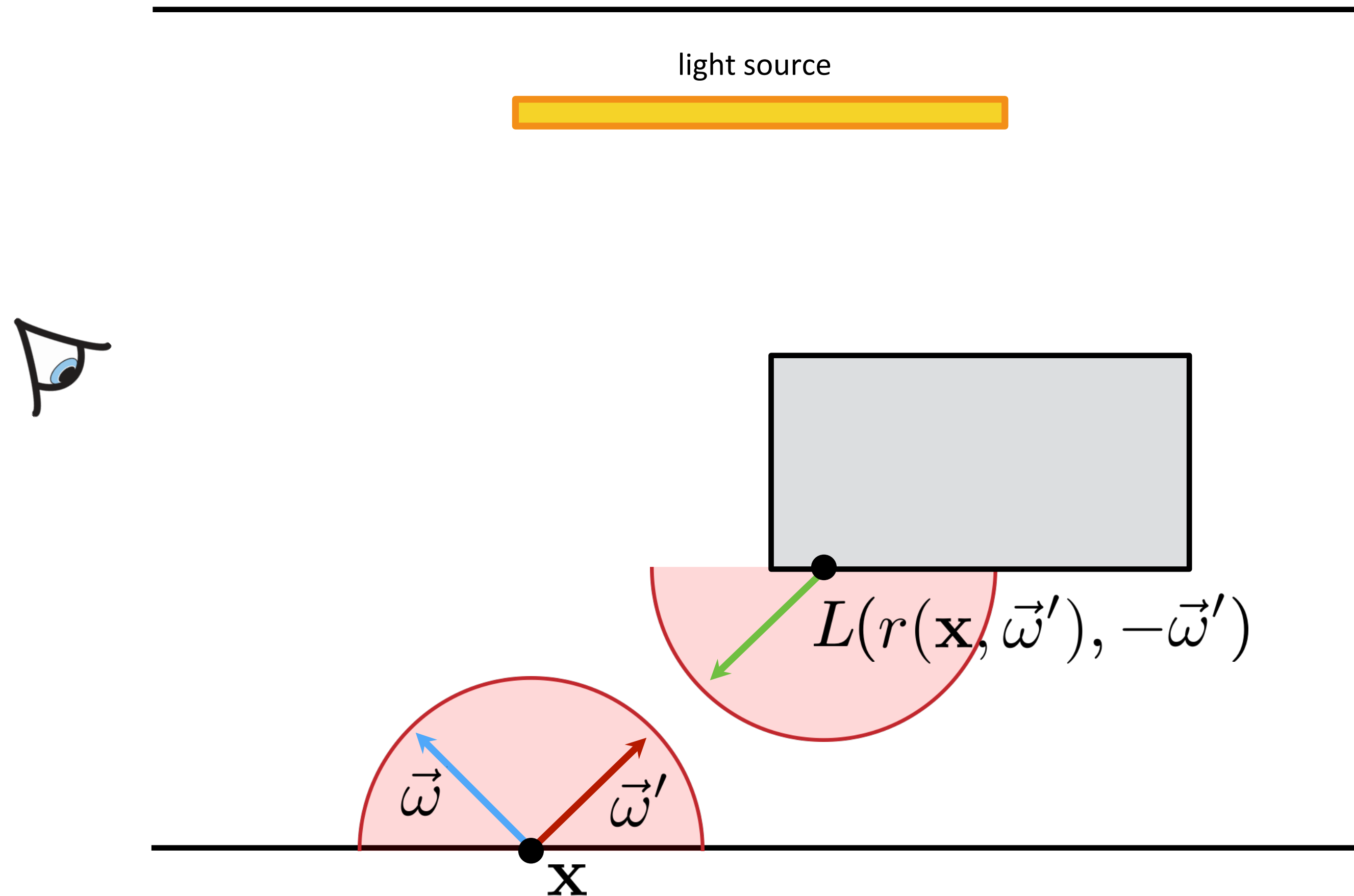
Rendering Equation

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos \theta' d\vec{\omega}'$$



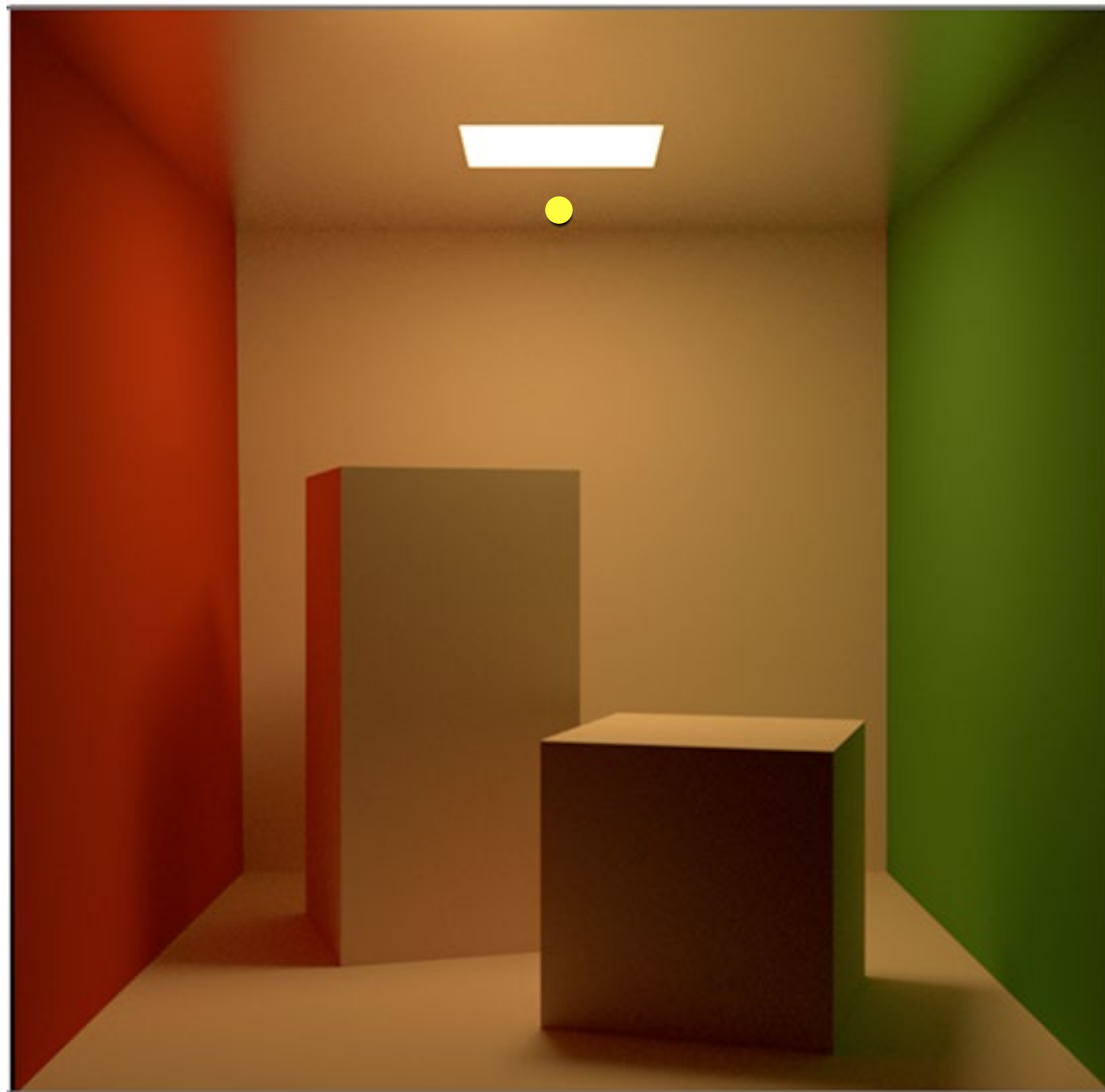
Rendering Equation

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) \underbrace{L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}')}_{\text{recursion}} \cos \theta' d\vec{\omega}'$$



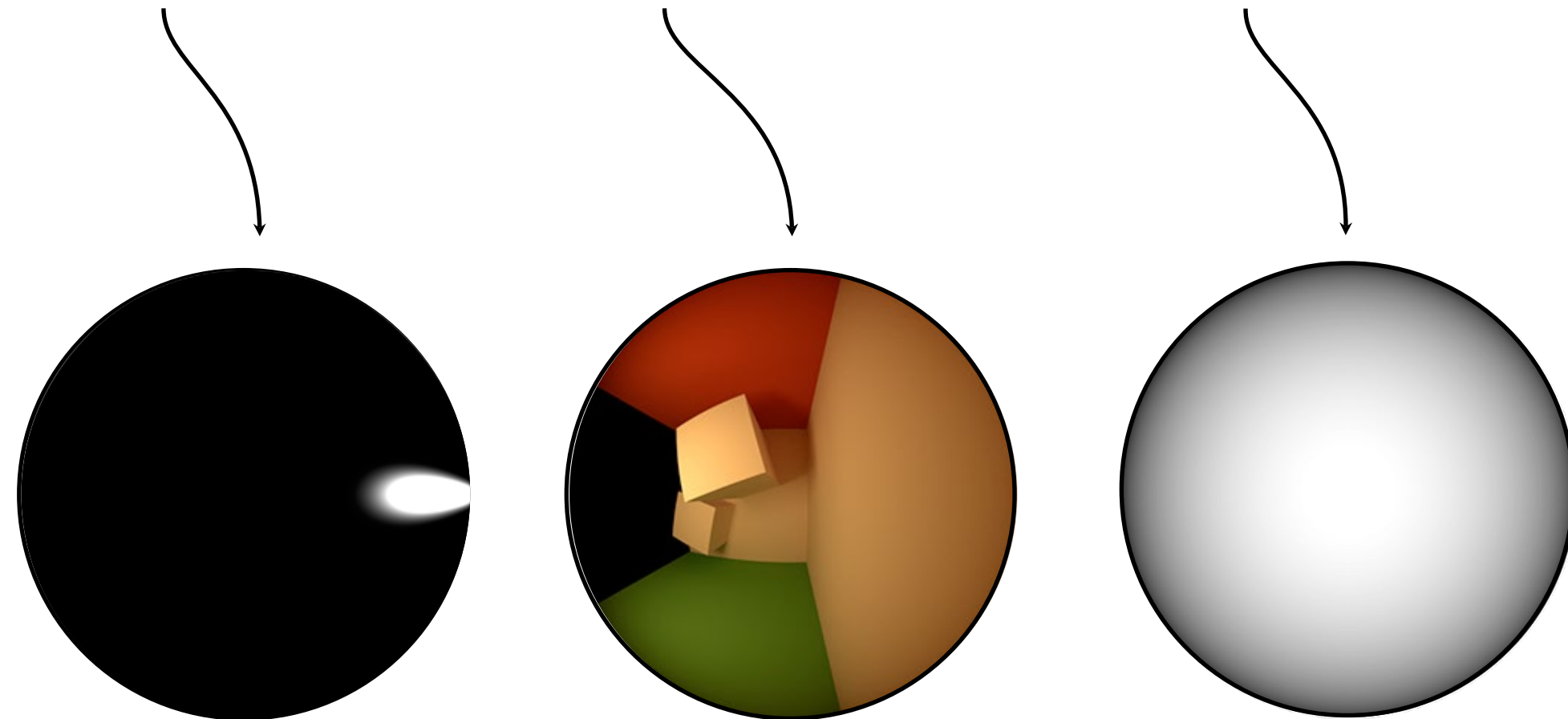
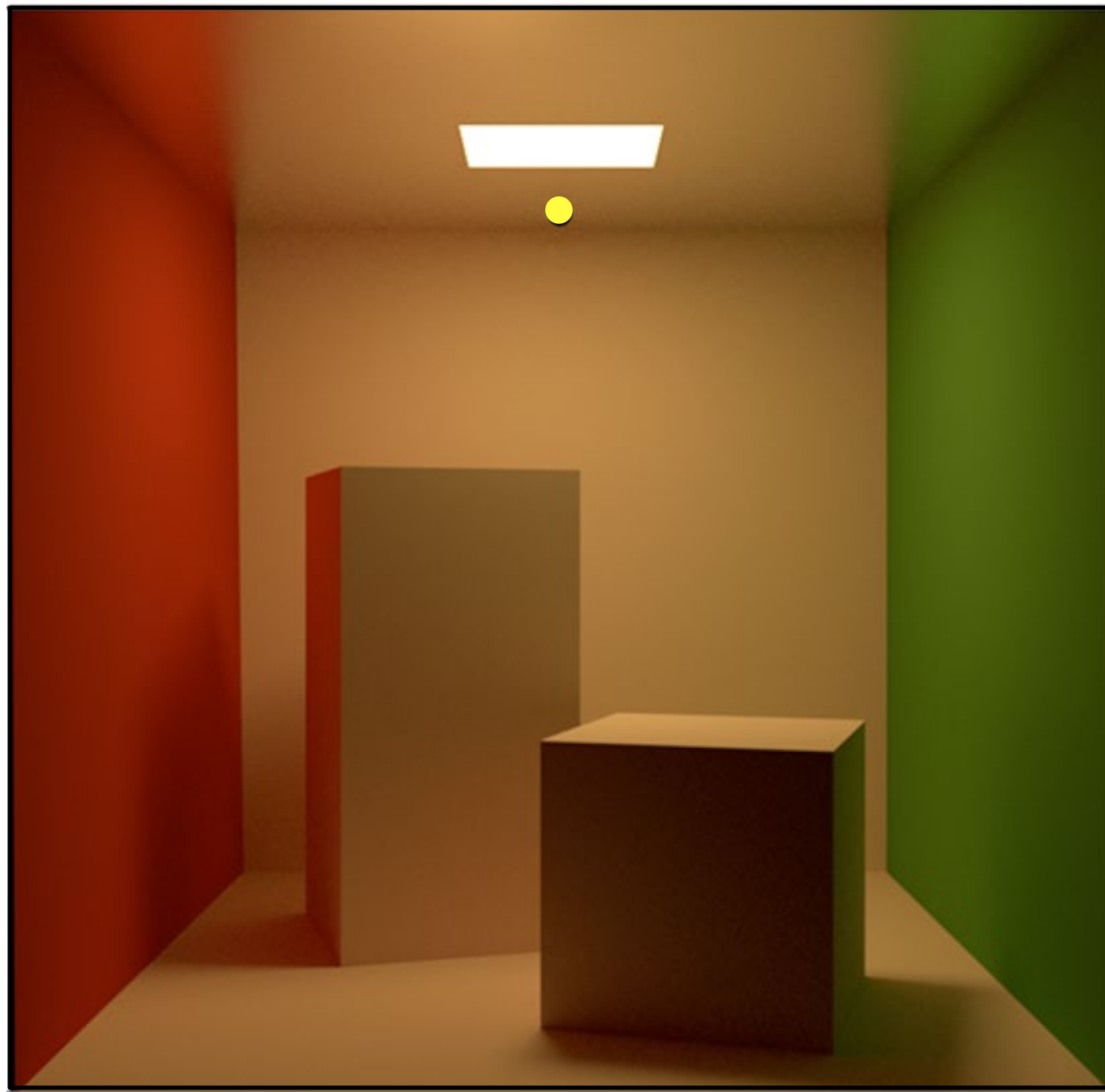
Rendering Equation

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos \theta' d\vec{\omega}'$$



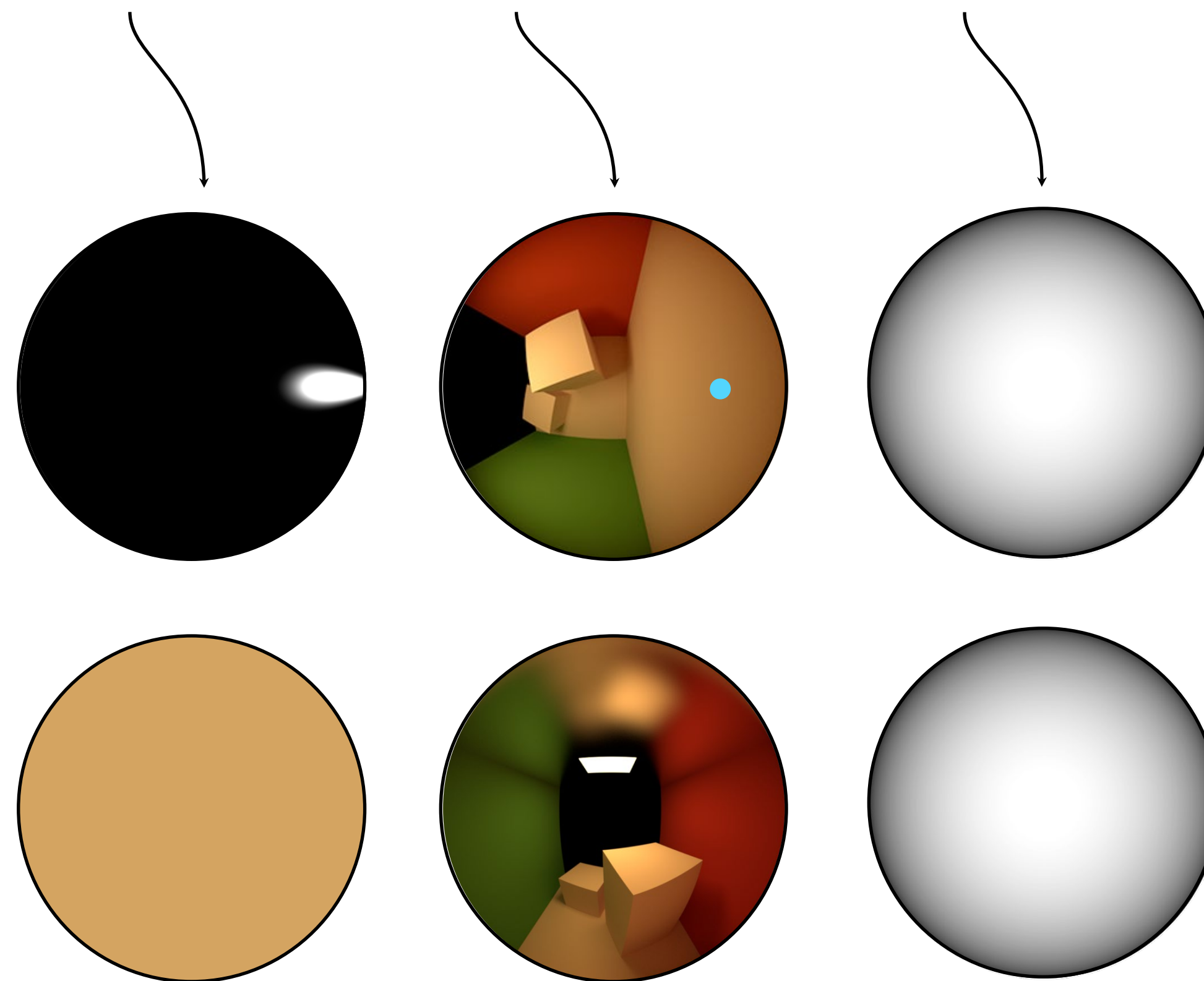
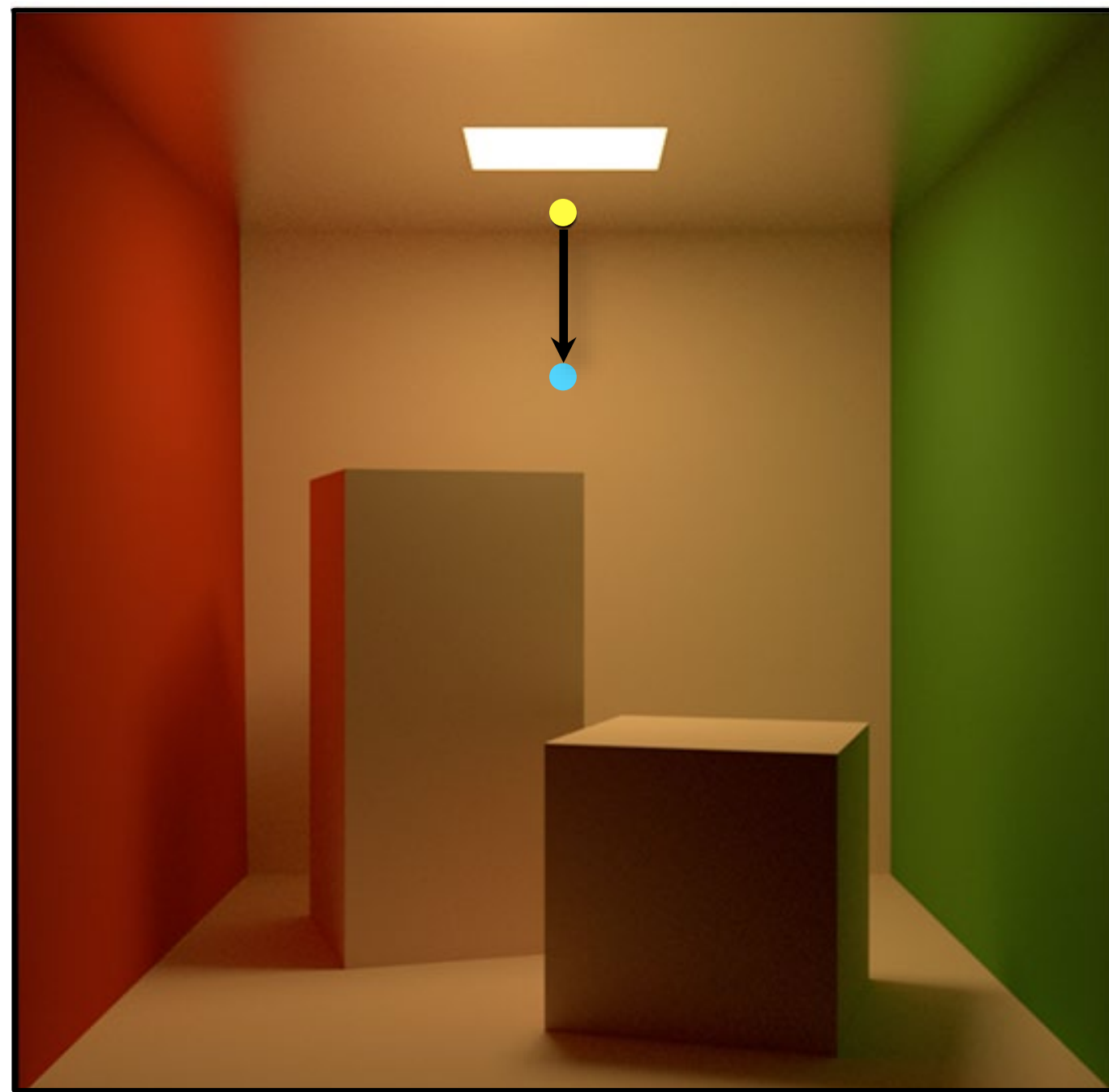
Rendering Equation

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos \theta' d\vec{\omega}'$$



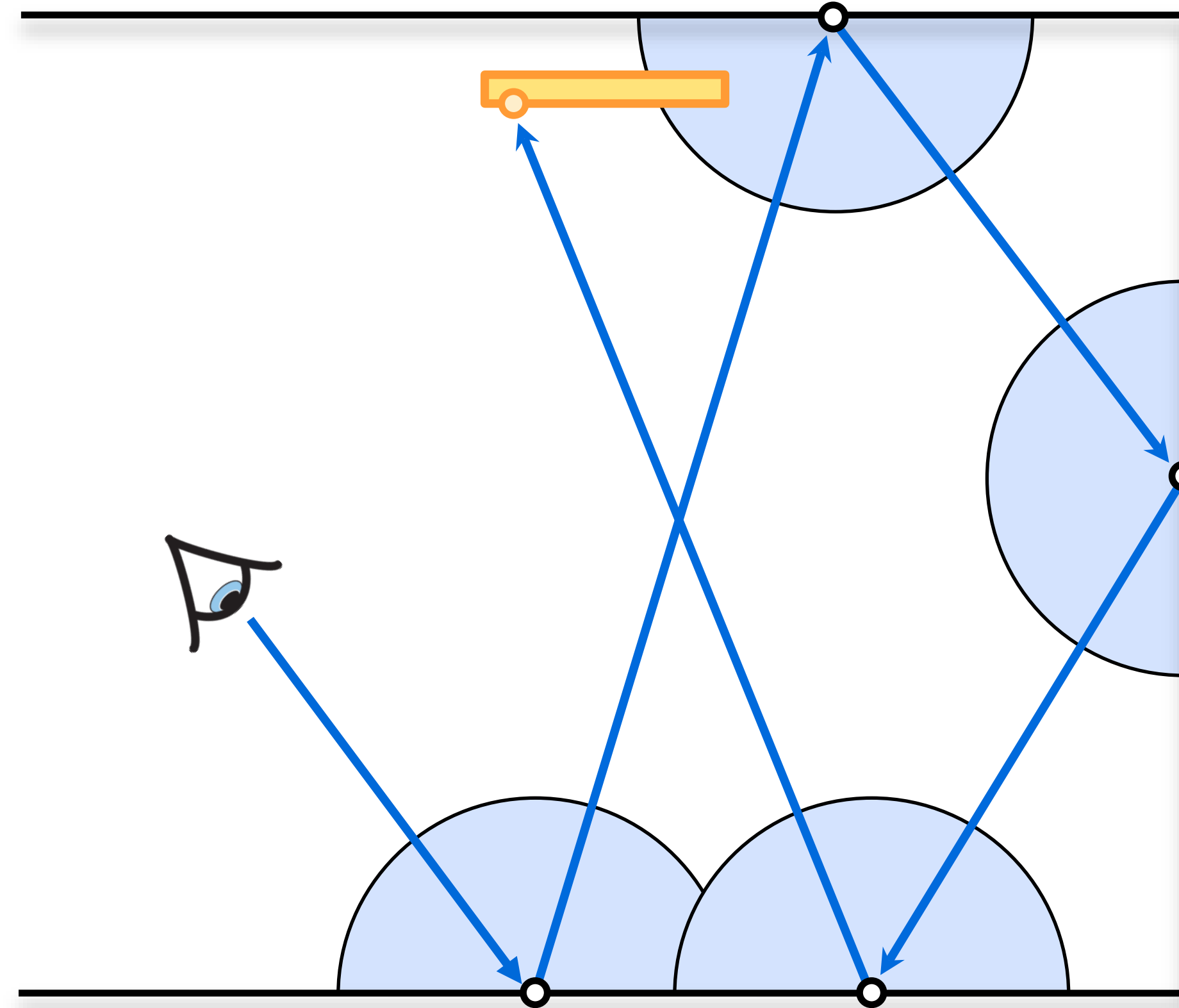
Rendering Equation

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos \theta' d\vec{\omega}'$$



Path Tracing

Path Tracing



$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\vec{\mathbf{x}}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos \theta' d\vec{\omega}'$$

$$L(\mathbf{x}, \vec{\omega}) \approx L_e(\mathbf{x}, \vec{\omega}) + \frac{f_r(\vec{\mathbf{x}}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos \theta'}{p(\vec{\omega}')}$$

Path Tracing Algorithm

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + L_r(\mathbf{x}, \vec{\omega})$$

Color color(Point \mathbf{x} , Direction ω , int moreBounces):

if not moreBounces:

return $L_e(\mathbf{x}, -\omega)$

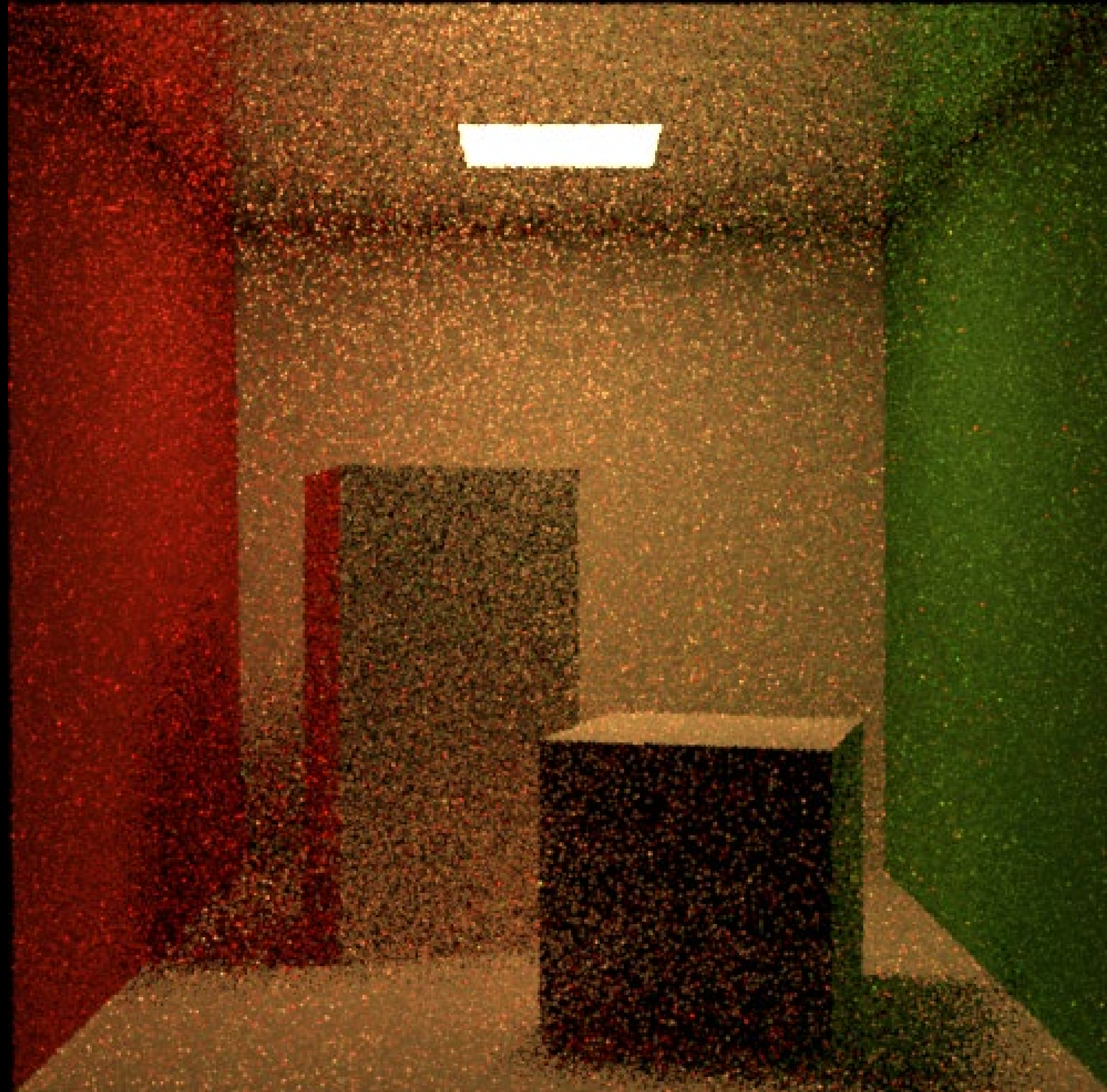
// sample recursive integral

ω' = sample from BRDF

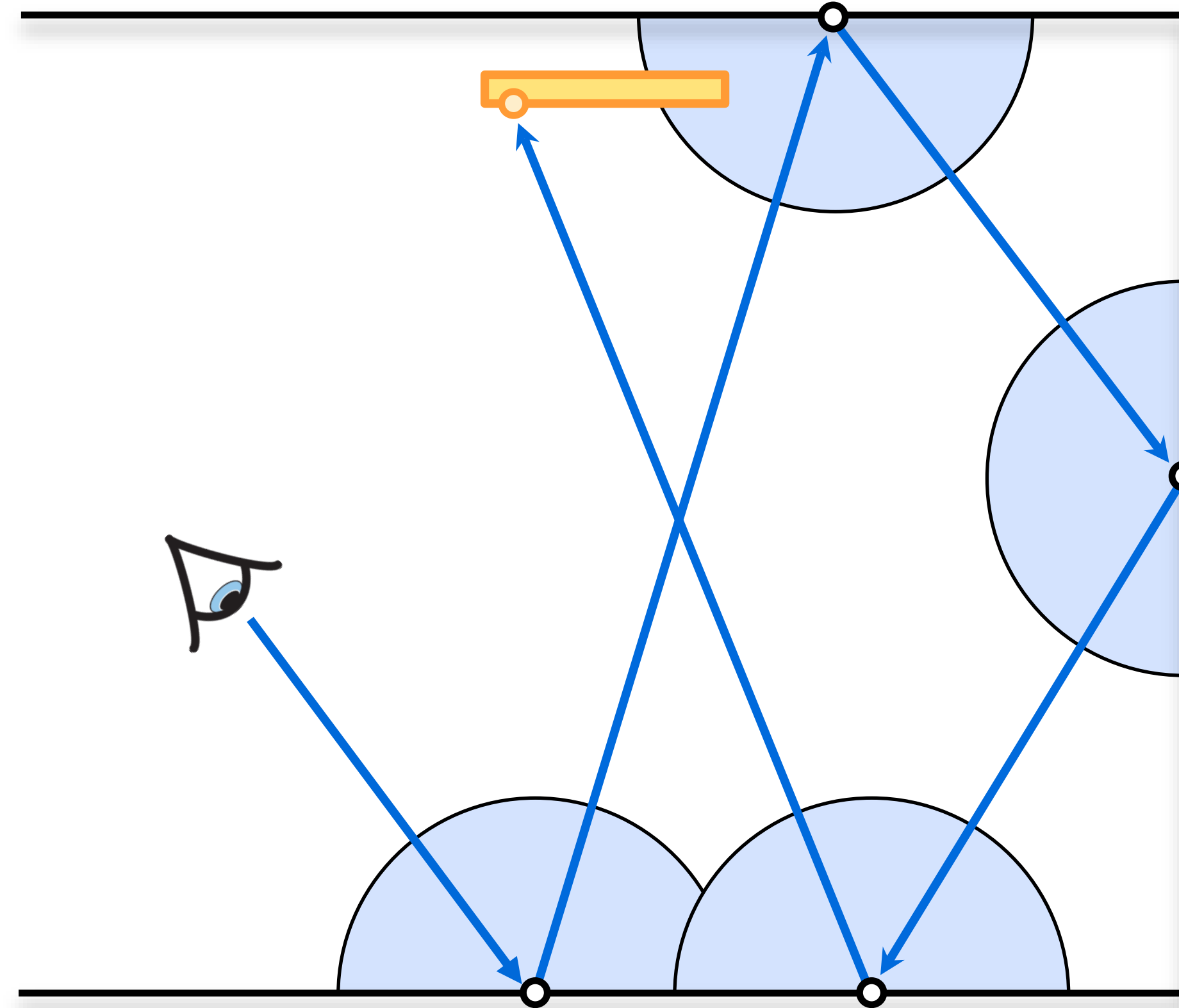
return $L_e(\mathbf{x}, -\omega) + \text{BRDF} * \text{color}(\text{trace}(\mathbf{x}, \omega'), \text{moreBounces}-1) * \text{dot}(\mathbf{n}, \omega') / \text{pdf}(\omega')$

Path Tracing with Shadow Rays

1 path/pixel



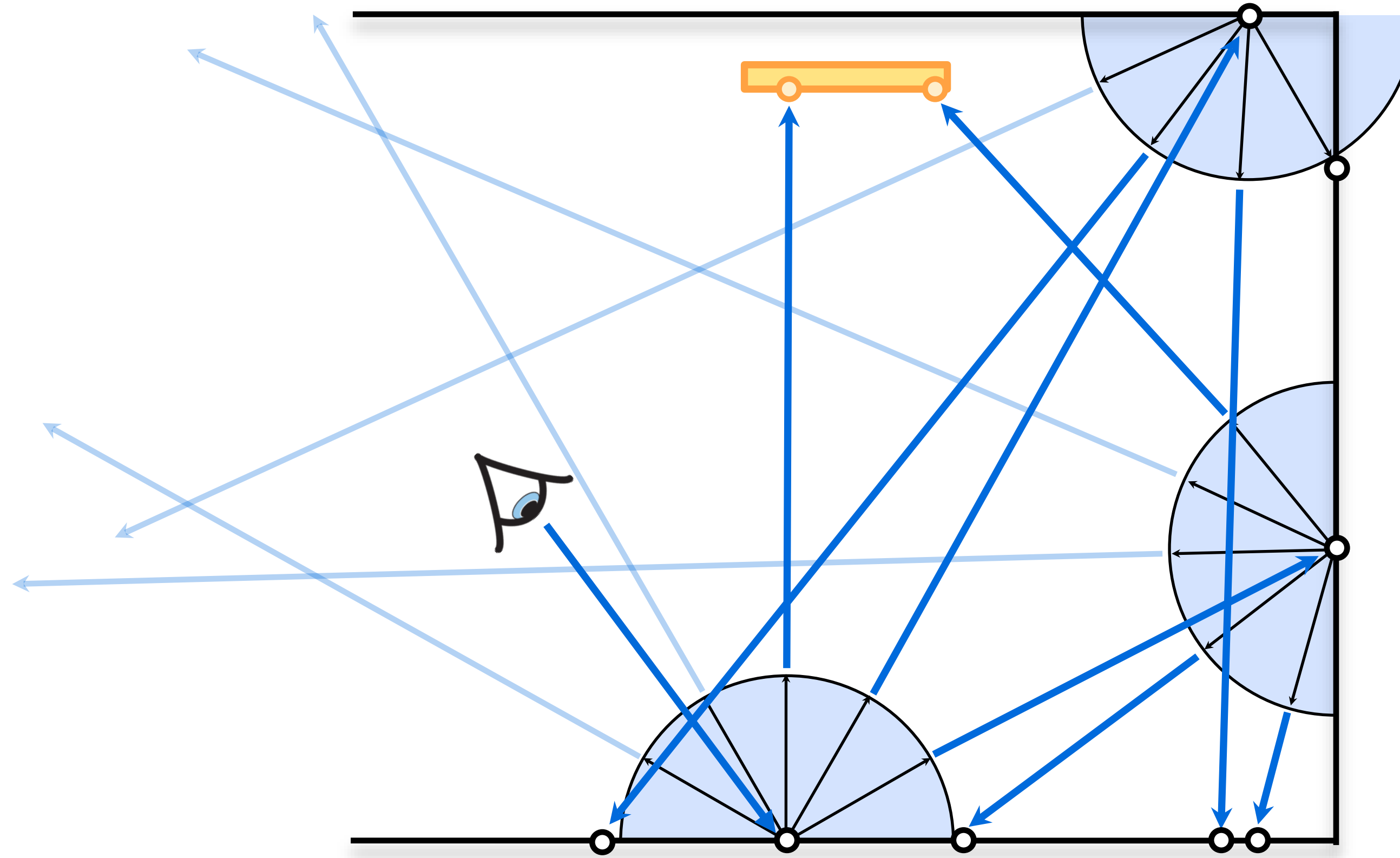
Path Tracing



$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\vec{\mathbf{x}}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos \theta' d\vec{\omega}'$$

$$L(\mathbf{x}, \vec{\omega}) \approx L_e(\mathbf{x}, \vec{\omega}) + \frac{f_r(\vec{\mathbf{x}}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos \theta'}{p(\vec{\omega}')}$$

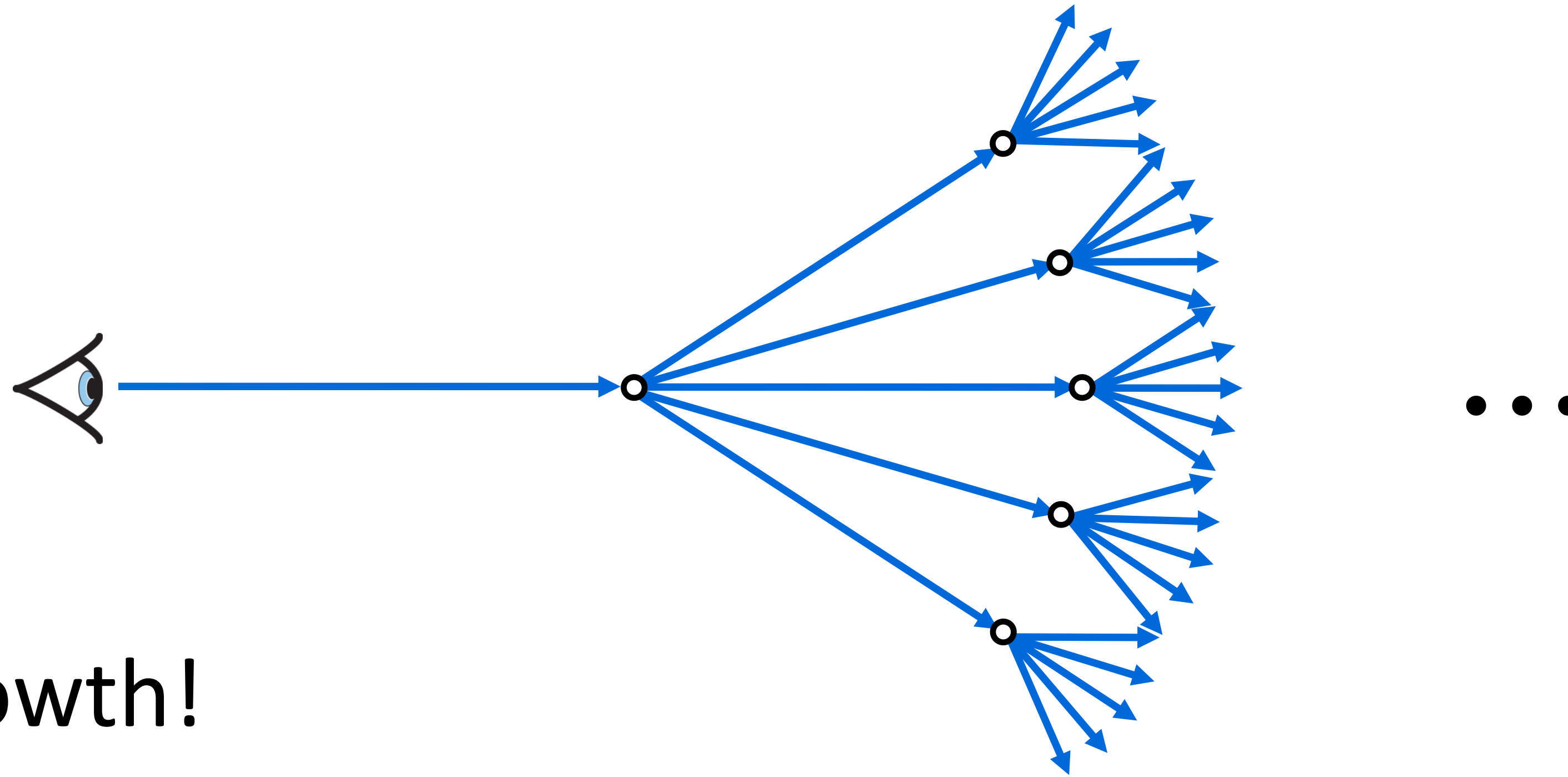
Improving quality: the wrong way



$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\vec{\mathbf{x}}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos \theta' d\vec{\omega}'$$

$$L(\mathbf{x}, \vec{\omega}) \approx L_e(\mathbf{x}, \vec{\omega}) + \frac{1}{N} \sum_{k=1}^N \frac{f_r(\vec{\mathbf{x}}, \vec{\omega}'_k, \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'_k), -\vec{\omega}'_k) \cos \theta'_k}{p(\vec{\omega}'_k)}$$

The problem



Exponential growth!

3-bounce contributes less than 1-bounce transport, but we estimate it with 25× as many samples!

Improving quality

Just shoot more rays/pixel

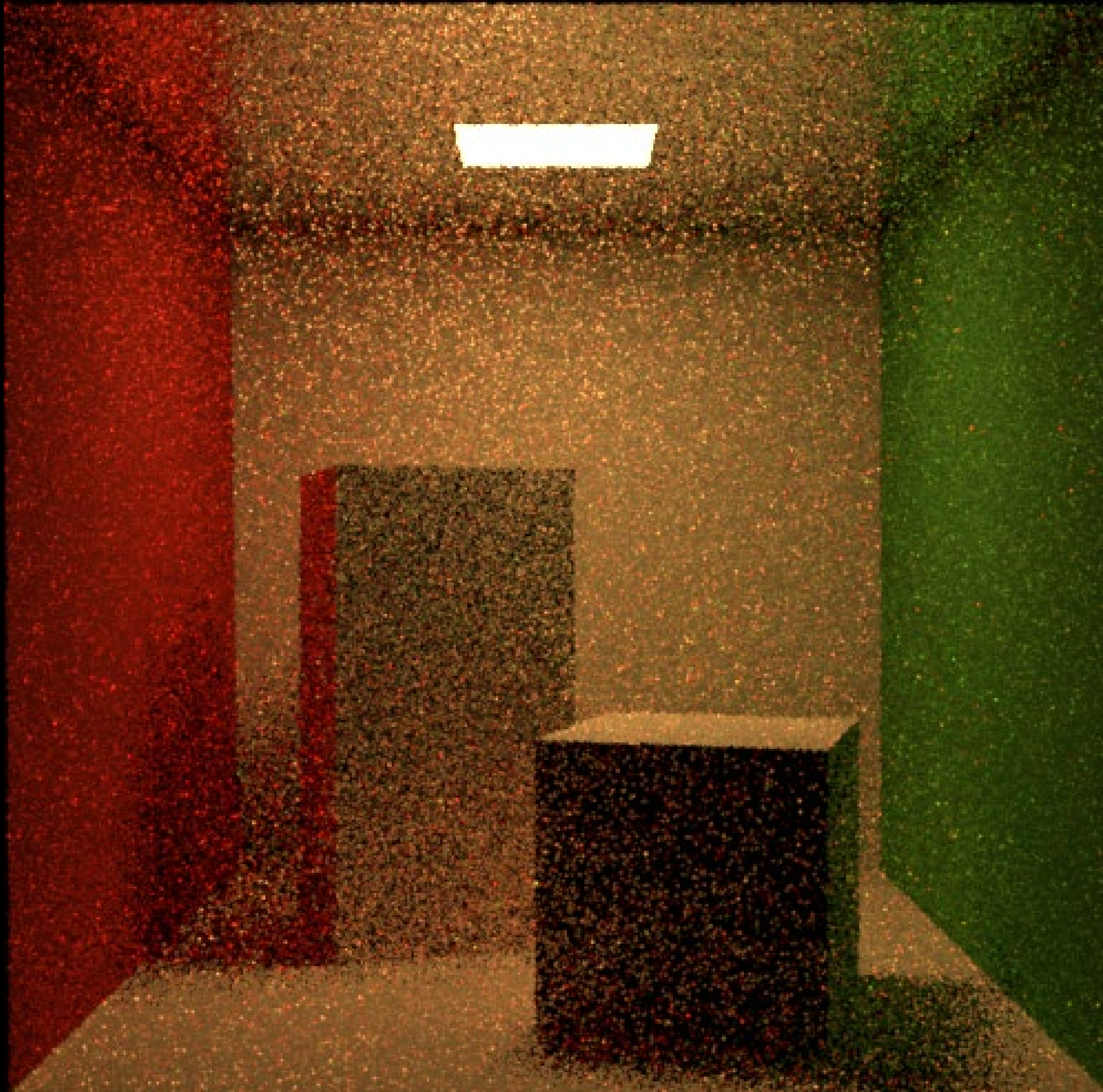
- avoid exponential growth: make sure not to branch!

Each ray will start a new **path**

We can achieve antialiasing/depth of field/motion blur at the same time “for free”!

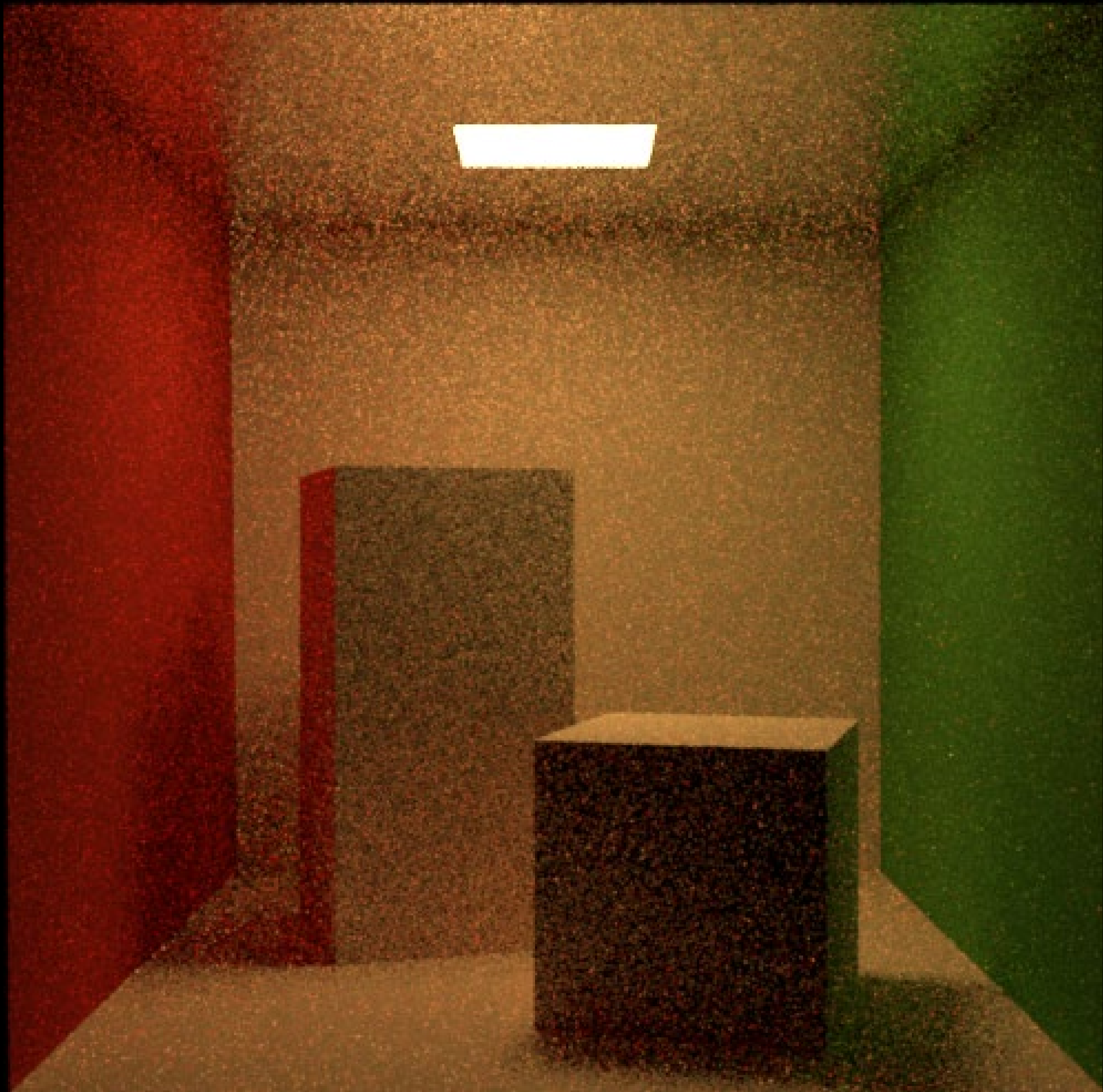
Path Tracing with Shadow Rays

1 path/pixel



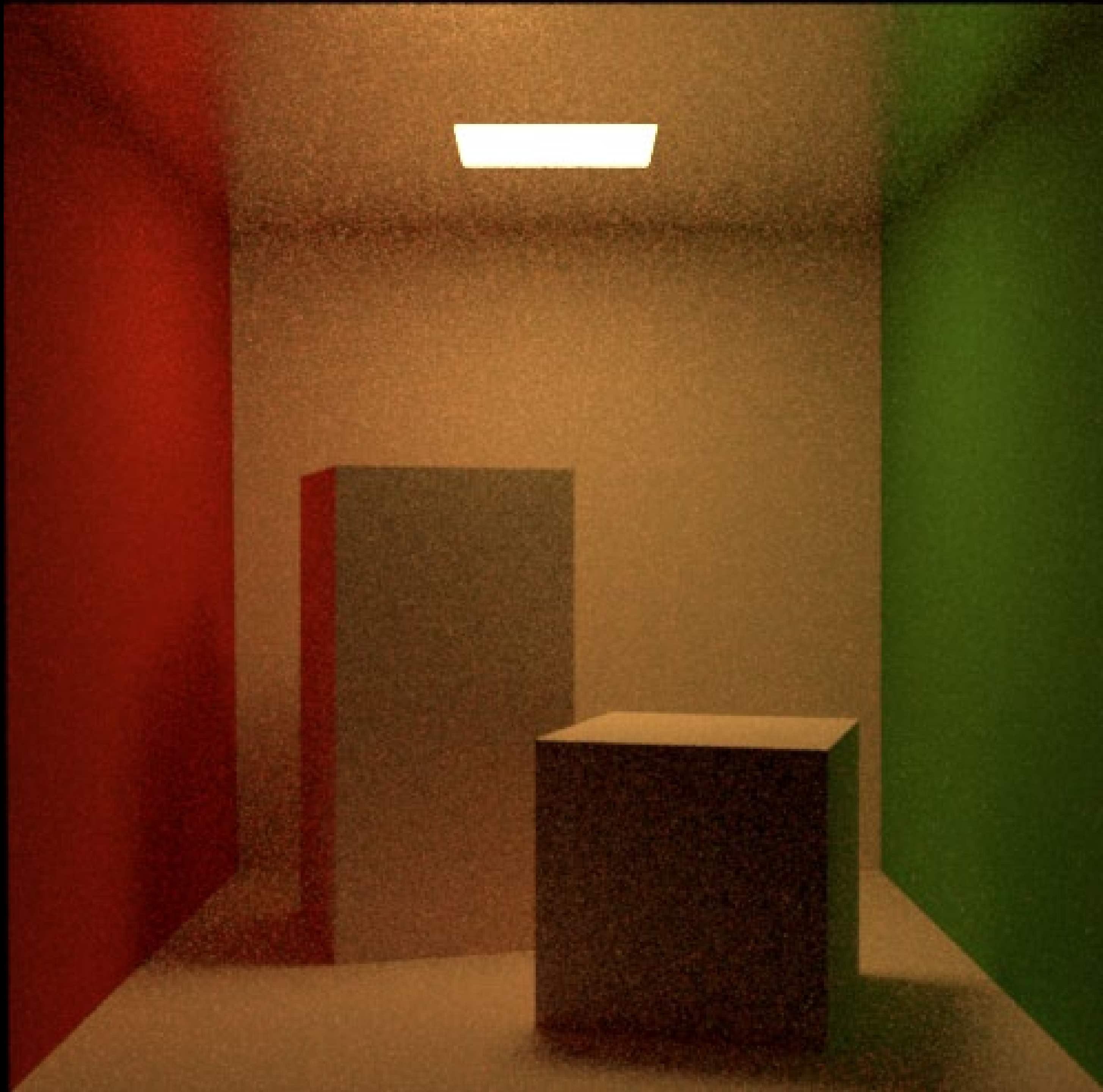
Path Tracing with Shadow Rays

4 paths/pixel



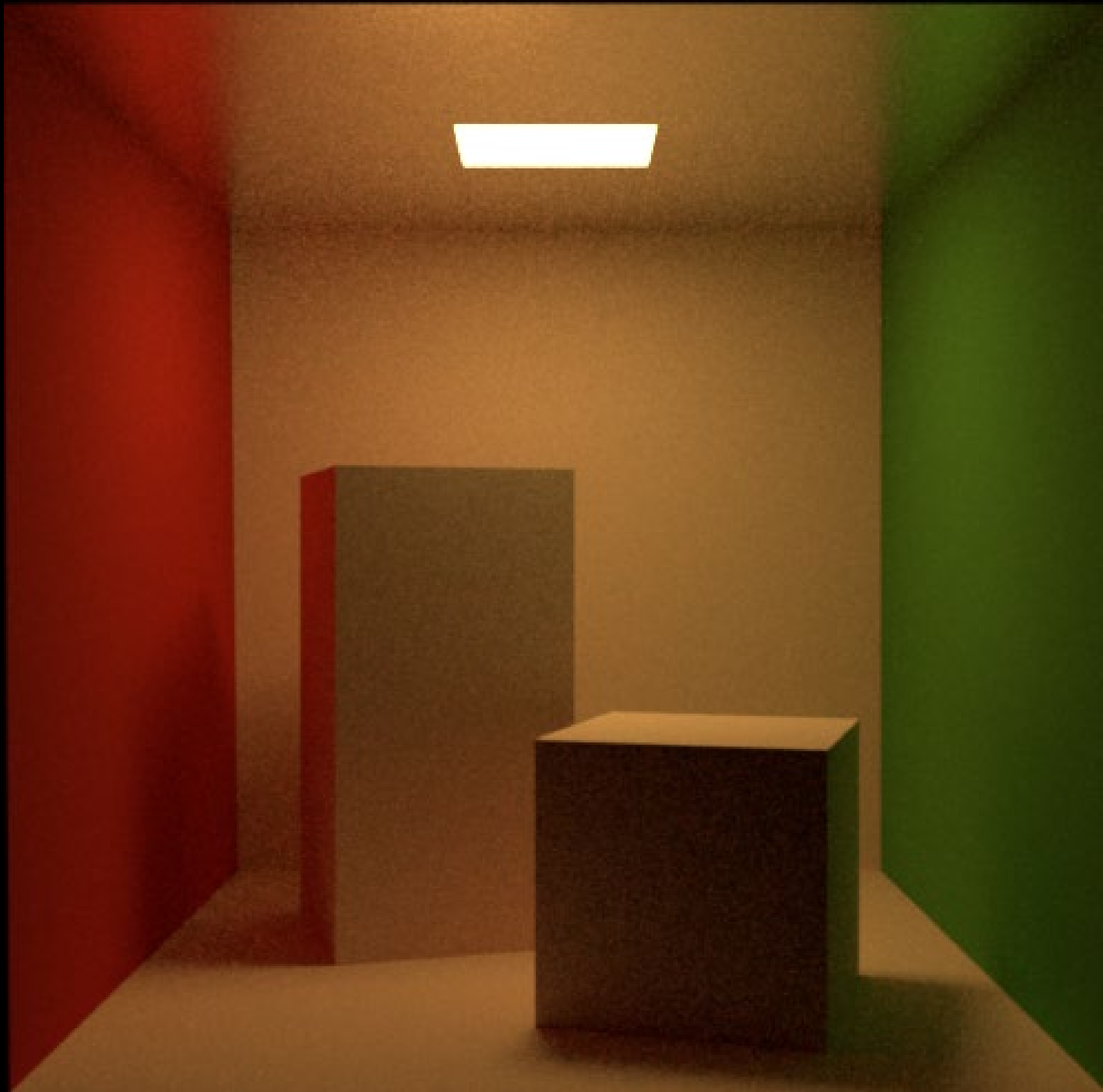
Path Tracing with Shadow Rays

16 paths/pixel



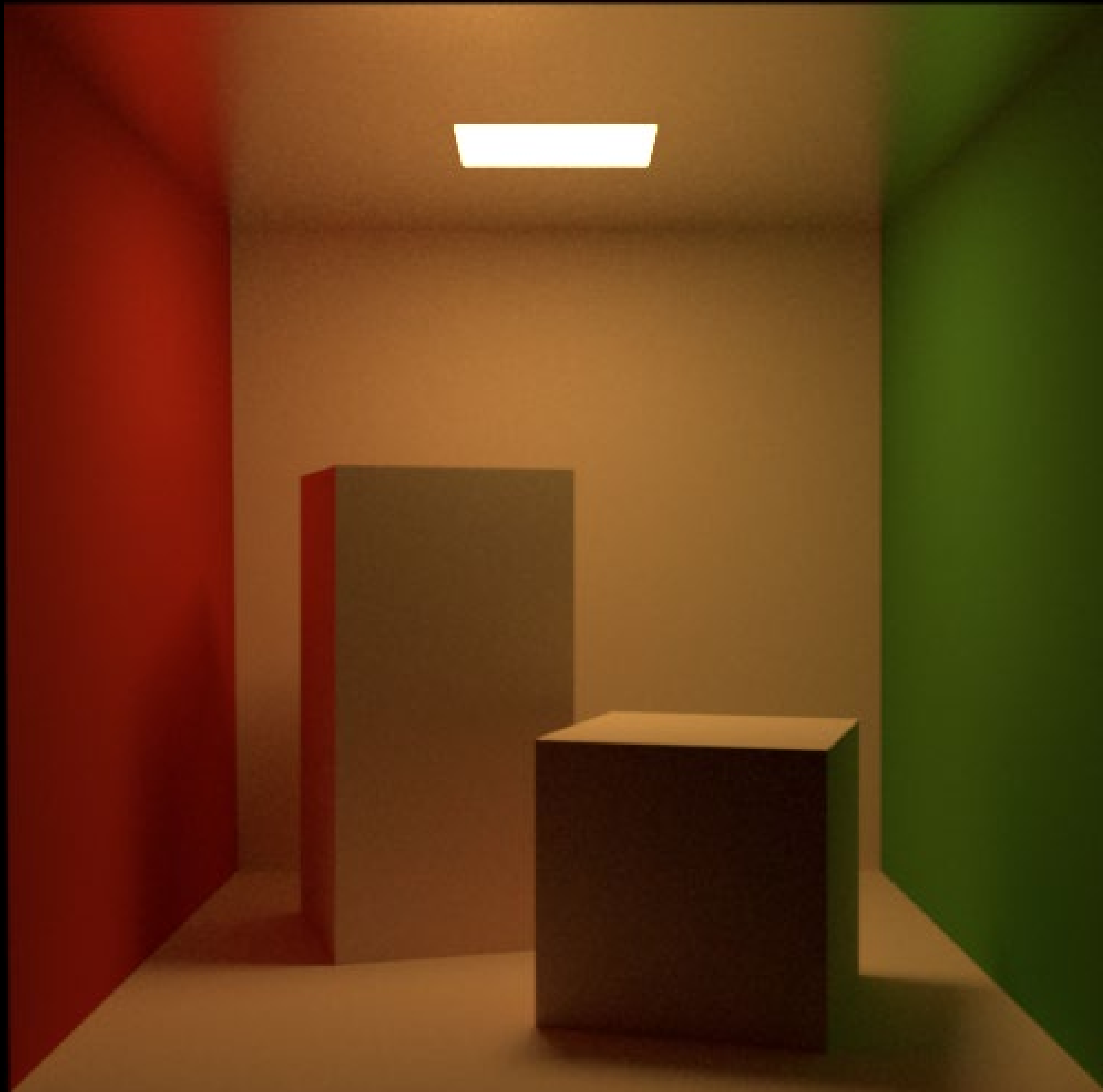
Path Tracing with Shadow Rays

64 paths/pixel



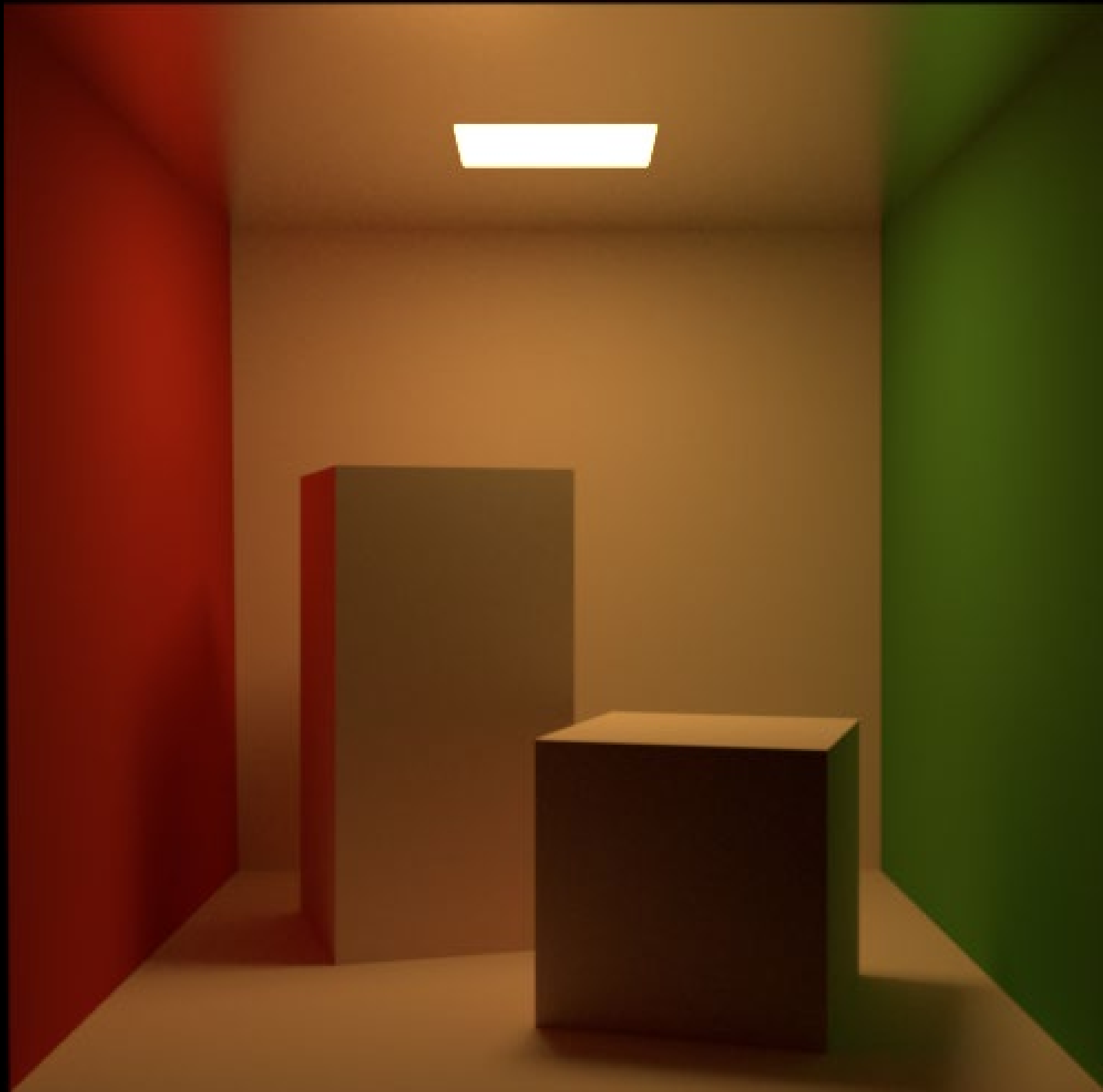
Path Tracing with Shadow Rays

256 paths/pixel



Path Tracing with Shadow Rays

1024 paths/pixel



When do we stop recursion?

Truncating at some fixed depth introduces *bias*

Solution: Russian roulette

Russian Roulette

Probabilistically terminate the recursion

New estimator: evaluate original estimator X with probability P (but reweighted), otherwise return zero:

$$X_{\text{rr}} = \begin{cases} \frac{X}{P} & \xi < P \\ 0 & \text{otherwise} \end{cases}$$

Unbiased: same expected value as original estimator:

$$E[X_{\text{rr}}] = P \cdot \left(\frac{E[X]}{P} \right) + (1 - P) \cdot 0 = E[X]$$

Russian Roulette

This will actually increase variance!

- but it will improve efficiency if P is chosen so that samples that are expensive, but are likely to make a small contribution, are skipped

You are already doing this

- probabilistic absorption in BSDF (instead of scattering)

Partitioning the Integrand

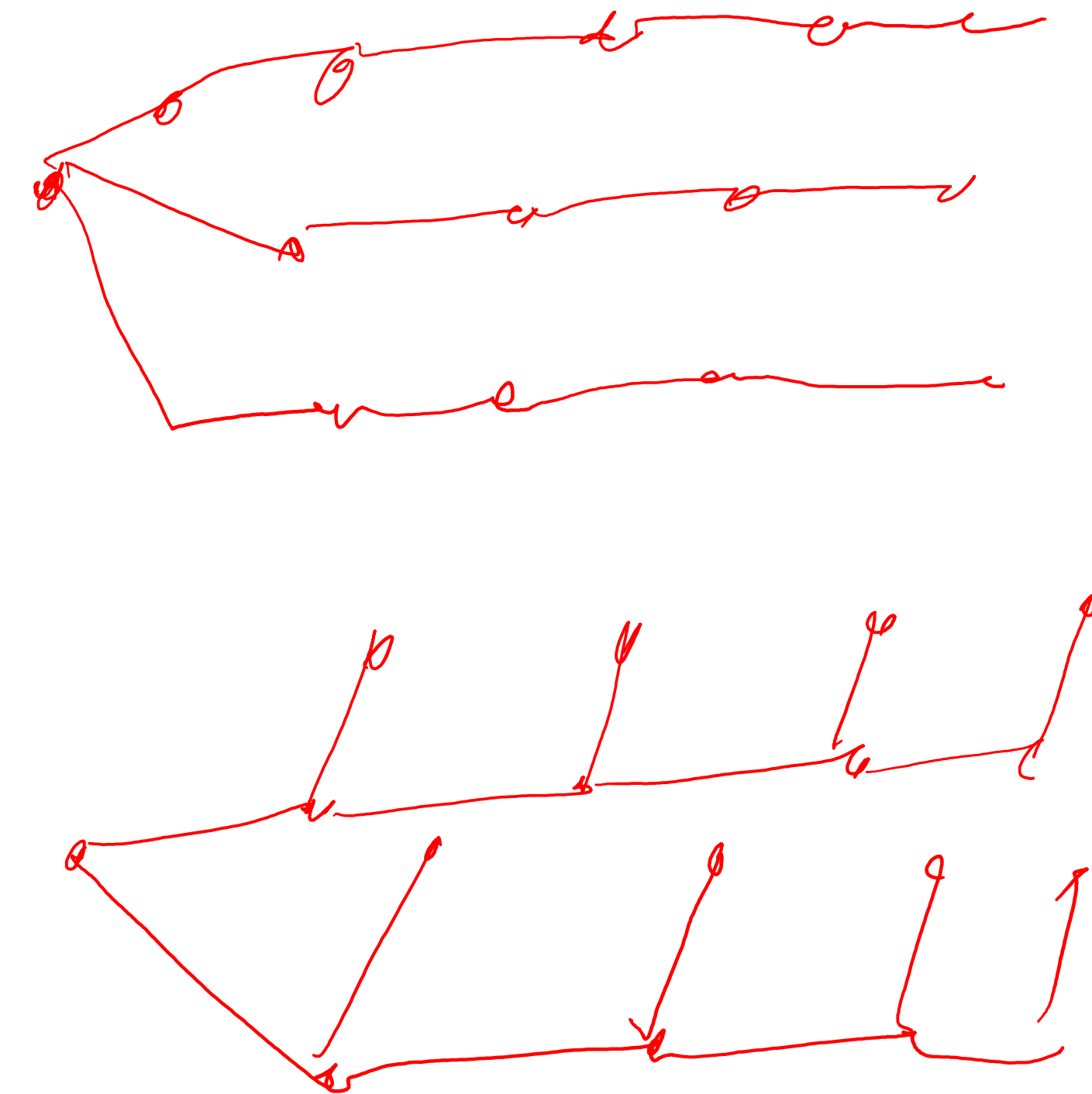
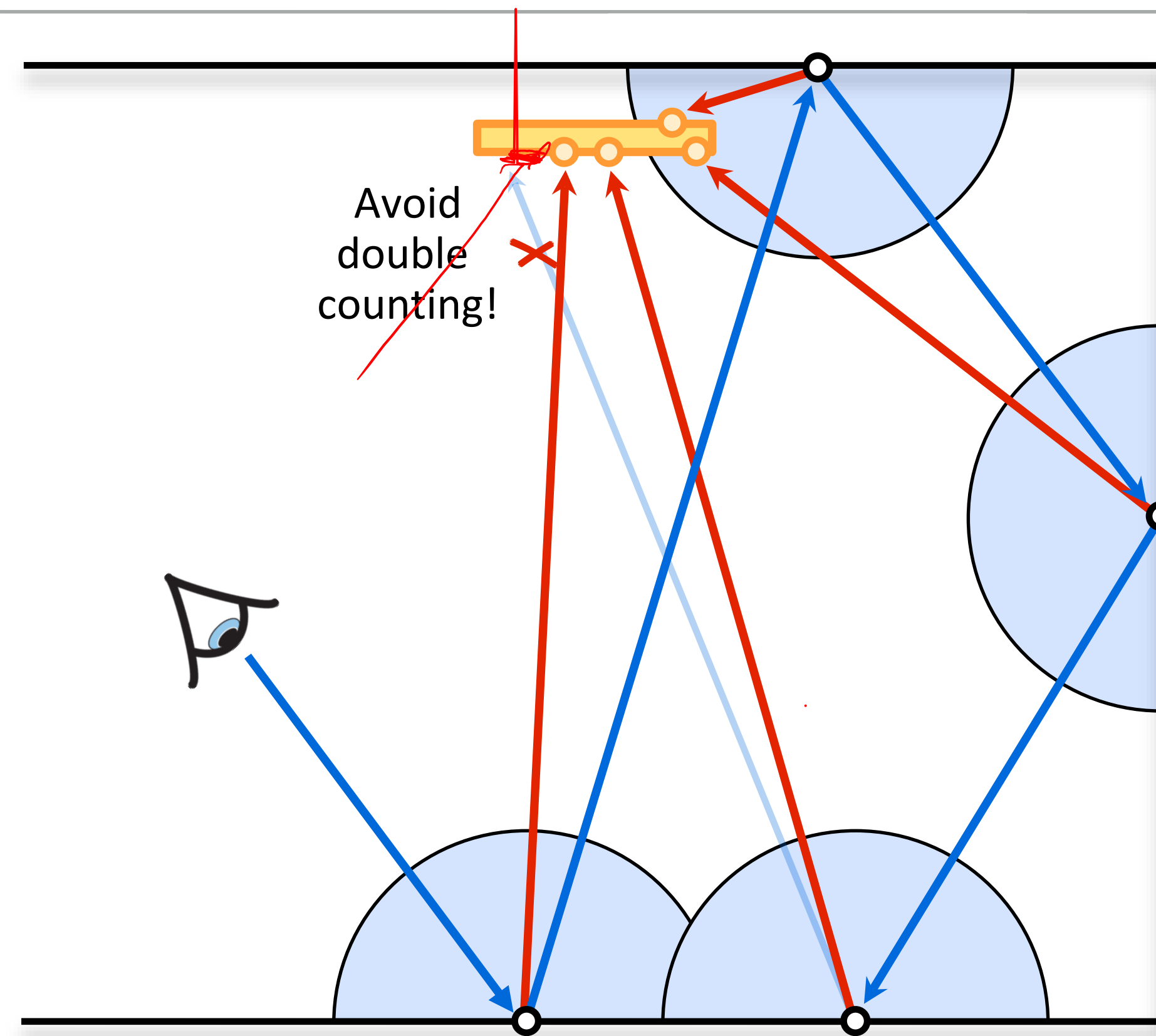
Direct illumination: sometimes better estimated by sampling emissive surfaces

Let's estimate direct illumination separately from indirect illumination, then add the two

- i.e., shoot shadow rays (direct) and gather rays (indirect)
- be careful *not to double-count!*

Also known as ***next-event estimation (NEE)***

Path Tracing with NEE



$$L(\mathbf{x}, \vec{\omega}) = L_e + \int_{A_e} \dots L_e(\mathbf{x} \leftarrow \mathbf{x}') \dots dA_e(\mathbf{x}') + \int_{H^2 \setminus A_e} \dots L(\mathbf{x}, \vec{\omega}') \dots d\vec{\omega}'$$

Path Tracing Algorithm with NEE

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + L_{\text{dir}}(\mathbf{x}, \vec{\omega}) + L_{\text{ind}}(\mathbf{x}, \vec{\omega})$$

Color color(Point \mathbf{x} , Direction ω , int moreBounces) :

if not moreBounces:
return L_e ;

double counting!

// next-event estimation: compute L_{dir} by sampling the light

ω_1 = sample from light

$L_{\text{dir}} = \text{BRDF} * \text{color}(\text{trace}(\mathbf{x}, \omega_1), 0) * \text{dot}(\mathbf{n}, \omega_1) / \text{pdf}(\omega_1)$

// compute L_{ind} by sampling the BSDF

ω_2 = sample from BSDF;

$L_{\text{ind}} = \text{BSDF} * \text{color}(\text{trace}(\mathbf{x}, \omega_2), \text{moreBounces}-1) * \text{dot}(\mathbf{n}, \omega_2) / \text{pdf}(\omega_2)$

return $L_e + L_{\text{dir}} + L_{\text{ind}}$

Path Tracing Algorithm with NEE

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + L_{\text{dir}}(\mathbf{x}, \vec{\omega}) + L_{\text{ind}}(\mathbf{x}, \vec{\omega})$$

Color color(Point \mathbf{x} , Direction ω , int moreBounces, bool includeLe):

$L_e = \text{includeLe} ? L_e(\mathbf{x}, -\omega) : \text{black}$

if not moreBounces:

return L_e

// next-event estimation: compute L_{dir} by sampling the light

$\omega_1 = \text{sample from light}$

$L_{\text{dir}} = \text{BRDF} * \text{color}(\text{trace}(\mathbf{x}, \omega_1), 0, \text{true}) * \text{dot}(\mathbf{n}, \omega_1) / \text{pdf}(\omega_1)$

// compute L_{ind} by sampling the BSDF

$\omega_2 = \text{sample from BSDF}$

$L_{\text{ind}} = \text{BSDF} * \text{color}(\text{trace}(\mathbf{x}, \omega_2), \text{moreBounces}-1, \text{false}) * \text{dot}(\mathbf{n}, \omega_2) / \text{pdf}(\omega_2)$

return $L_e + L_{\text{dir}} + L_{\text{ind}}$

Questions?

We should really be using MIS or mixture sampling

Naive Path Tracing

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + L_r(\mathbf{x}, \vec{\omega})$$

Color color(Point \mathbf{x} , Direction ω , int moreBounces):

if not moreBounces:

return $L_e(\mathbf{x}, -\omega)$

// sample recursive integral

ω' = sample from BRDF

return $L_e(\mathbf{x}, -\omega) + \text{BRDF} * \text{color}(\text{trace}(\mathbf{x}, \omega'), \text{moreBounces}-1) * \text{dot}(\mathbf{n}, \omega') / \text{pdf}(\omega')$

Path Tracing with mixture sampling

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + L_r(\mathbf{x}, \vec{\omega})$$

Color color(Point \mathbf{x} , Direction ω , int moreBounces):

if not moreBounces:

return $L_e(\mathbf{x}, -\omega)$

// sample recursive integral

ω' = sample from mixture PDF

return $L_e(\mathbf{x}, -\omega) + \text{BRDF} * \text{color}(\text{trace}(\mathbf{x}, \omega'), \text{moreBounces}-1) * \text{dot}(\mathbf{n}, \omega') / \text{pdf}(\omega')$

Path Tracing Algorithm with NEE

color trace(Point x , Direction ω , int moreBounces, **bool includeLe**):

get scene intersection x , and normal n

$L_e = \text{includeLe} ? L_e(x, -\omega) : \text{black}$

if not moreBounces:

return L_e

// next-event estimation: compute L_{dir} by sampling the light

$\omega_1 = \text{sample from light}$

$L_{\text{dir}} = \text{BRDF} * \text{trace}(x, \omega_1, 0, \text{true}) * \text{dot}(n, \omega_1) / \text{pdf}(\omega_1)$

// compute L_{ind} by sampling the BSDF

$\omega_2 = \text{sample from BSDF}$

$L_{\text{ind}} = \text{BSDF} * \text{trace}(x, \omega_2, \text{moreBounces}-1, \text{false}) * \text{dot}(n, \omega_2) / \text{pdf}(\omega_2)$

return $L_e + L_{\text{dir}} + L_{\text{ind}}$

Path Tracing Algorithm with NEE+MIS

color trace(Point x , Direction ω , int moreBounces, float L_e weight):

get scene intersection x , and normal n

$L_e = L_e\text{weight} * L_e(x, -\omega)$

if not moreBounces:

return L_e

// next-event estimation: compute L_{dir} by sampling the light

$\omega_1 = \text{sample from light}$

$L_{dir} = \text{BRDF} * \text{trace}(x, \omega_1, 0, \text{mis-weight}_1) * \text{dot}(n, \omega_1) / \text{pdf}(\omega_1)$

// compute L_{ind} by sampling the BSDF

$\omega_2 = \text{sample from BSDF}$

$L_{ind} = \text{BSDF} * \text{trace}(x, \omega_2, \text{moreBounces}-1, \text{mis-weight}_2) * \text{dot}(n, \omega_2) / \text{pdf}(\omega_2)$

return $L_e + L_{dir} + L_{ind}$

Path Tracing on 99 Lines of C++

```
1. #include <math.h> // smallpt, a Path Tracer by Kevin Beason, 2008
2. #include <stdlib.h> // Make : g++ -O3 -fopenmp smallpt.cpp -o smallpt
3. #include <stdio.h> // Remove "-fopenmp" for g++ version < 4.2
4. struct Vec { // Usage: time ./smallpt 5000 && xv image.ppm
5.     double x, y, z; // position, also color (r,g,b)
6.     Vec(double x_=0, double y_=0, double z_=0){ x=x_; y=y_; z=z_; }
7.     Vec operator+(const Vec &b) const { return Vec(x+b.x,y+b.y,z+b.z); }
8.     Vec operator-(const Vec &b) const { return Vec(x-b.x,y-b.y,z-b.z); }
9.     Vec operator*(double b) const { return Vec(x*b,y*b,z*b); }
10.    Vec mult(const Vec &b) const { return Vec(x*b.x,y*b.y,z*b.z); }
11.    Vec& norm(){ return *this = *this * (1/sqrt(x*x+y*y+z*z)); }
12.    double dot(const Vec &b) const { return x*b.x+y*b.y+z*b.z; } // cross:
13.    Vec operator%(Vec&b){return Vec(y*b.z-z*b.y,z*b.x-x*b.z,x*b.y-y*b.x);}
14. };
15. struct Ray { Vec o, d; Ray(Vec o_, Vec d_) : o(o_), d(d_) {} };
16. enum Refl_t { DIFF, SPEC, REFR }; // material types, used in radiance()
17. struct Sphere {
18.     double rad; // radius
19.     Vec p, e, c; // position, emission, color
20.     Refl_t refl; // reflection type (DIFFuse, SPECular, REFRactive)
21.     Sphere(double rad_, Vec p_, Vec e_, Vec c_, Refl_t refl_):
22.         rad(rad_), p(p_), e(e_), c(c_), refl(refl_) {}
23.     double intersect(const Ray &r) const { // returns distance, 0 if nohit
24.         Vec op = p-r.o; // Solve t^2*d.d + 2*t*(o-p).d + (o-p).(o-p)-R^2 = 0
25.         double t, eps=1e-4, b=op.dot(r.d), det=b*b-op.dot(op)+rad*rad;
26.         if (det<0) return 0; else det=sqrt(det);
27.         return (t=b-det)>eps ? t : ((t=b+det)>eps ? t : 0);
28.     }
29. };
30. Sphere spheres[] = { //Scene: radius, position, emission, color, material
31.     Sphere(1e5, Vec( 1e5+1,40.8,81.6), Vec(),Vec(.75,.25,.25),DIFF), //Left
32.     Sphere(1e5, Vec(-1e5+99,40.8,81.6),Vec(),Vec(.25,.25,.75),DIFF), //Rght
33.     Sphere(1e5, Vec(50,40.8, 1e5), Vec(),Vec(.75,.75,.75),DIFF), //Back
34.     Sphere(1e5, Vec(50,40.8,-1e5+170), Vec(),Vec(), DIFF), //Frnt
35.     Sphere(1e5, Vec(50, 1e5, 81.6), Vec(),Vec(.75,.75,.75),DIFF), //Botm
36.     Sphere(1e5, Vec(50,-1e5+81.6,81.6),Vec(),Vec(.75,.75,.75),DIFF), //Top
37.     Sphere(16.5,Vec(27,16.5,47), Vec(),Vec(1,1,1)*.999, SPEC), //Mirr
38.     Sphere(16.5,Vec(73,16.5,78), Vec(),Vec(1,1,1)*.999, REFR), //Glas
39.     Sphere(600, Vec(50,681.6-.27,81.6),Vec(12,12,12), Vec(), DIFF) //Lite
40. };
41. inline double clamp(double x){ return x<0 ? 0 : x>1 ? 1 : x; }
42. inline int toInt(double x){ return int(pow(clamp(x),1/2.2)*255+.5); }
43. inline bool intersect(const Ray &r, double &t, int &id){
44.     double n=sizeof(spheres)/sizeof(Sphere), d, inf=t=1e20;
45.     for(int i=int(n);i--;) if((d=spheres[i].intersect(r))&&d<t){t=d;id=i;}
46.     return t<inf;
47. }
48. Vec radiance(const Ray &r, int depth, unsigned short *Xi){
49.     double t; // distance to intersection
50.     int id=0; // id of intersected object
51.     if (!intersect(r, t, id)) return Vec(); // if miss, return black
52.     const Sphere &obj = spheres[id]; // the hit object
53.     Vec x=r.o+r.d*t, n=(x-obj.p).norm(), nl=n.dot(r.d)<0?n:n*-1, f=obj.c;
54.     double p = f.x>f.y && f.x>f.z ? f.x : f.y>f.z ? f.y : f.z; // max refl
55.     if (++depth>5) if (erand48(Xi)<p) f=f*(1/p); else return obj.e; //R.R.
56.     if (obj.refl == DIFF){ // Ideal DIFFUSE reflection
57.         double r1=2*M_PI*erand48(Xi), r2=erand48(Xi), r2s=sqrt(r2);
58.         Vec w=nl, u=((fabs(w.x)>.1?Vec(0,1):Vec(1))%w).norm(), v=w%u;
59.         Vec d = (u*cos(r1)*r2s + v*sin(r1)*r2s + w*sqrt(1-r2)).norm();
60.         return obj.e + f.mult(radiance(Ray(x,d),depth,Xi));
61.     } else if (obj.refl == SPEC) // Ideal SPECULAR reflection
62.         return obj.e + f.mult(radiance(Ray(x,r.d-n*2*n.dot(r.d)),depth,Xi));
63.     Ray reflRay(x, r.d-n*2*n.dot(r.d)); // Ideal dielectric REFRACTION
64.     bool into = n.dot(nl)>0; // Ray from outside going in?
65.     double nc=1, nt=1.5, nnt=into?nc/nt:nt/nc, ddn=r.d.dot(nl), cos2t;
66.     if ((cos2t=1-nnt*nnt*(1-ddn*ddn))<0) // Total internal reflection
67.         return obj.e + f.mult(radiance(reflRay,depth,Xi));
68.     Vec tdir = (r.d*nnt - n*((into?-1):1)*(ddn*nnt+sqrt(cos2t))).norm();
69.     double a=nt-nc, b=nt+nc, R0=a*a/(b*b), c = 1-(into?-ddn:tdir.dot(n));
70.     double Re=R0+(1-R0)*c*c*c*c*c, Tr=1-Re, P=.25+.5*Re, RP=Re/P, TP=Tr/(1-P);
71.     return obj.e + f.mult(depth>2 ? (erand48(Xi)<P ? // Russian roulette
72.         radiance(reflRay,depth,Xi)*RP:radiance(Ray(x,tdir),depth,Xi)*TP) :
73.         radiance(reflRay,depth,Xi)*Re+radiance(Ray(x,tdir),depth,Xi)*Tr);
74. }
75. int main(int argc, char *argv[]){
76.     int w=1024, h=768, samps = argc==2 ? atoi(argv[1])/4 : 1; // # samples
77.     Ray cam(Vec(50,52,295.6), Vec(0,-0.042612,-1).norm()); // cam pos, dir
78.     Vec cx=Vec(w*.5135/h), cy=(cx%cam.d).norm()**.5135, r, *c=new Vec[w*h];
79.     #pragma omp parallel for schedule(dynamic, 1) private(r) // OpenMP
80.     for (int y=0; y<h; y++){ // Loop over image rows
81.         fprintf(stderr, "\rRendering (%d spp) %5.2f%%", samps*4, 100.*y/(h-1));
82.         for (unsigned short x=0, Xi[3]={0,0,y*y*y}; x<w; x++) // Loop cols
83.             for (int sy=0, i=(h-y-1)*w+x; sy<2; sy++) // 2x2 subpixel rows
84.                 for (int sx=0; sx<2; sx++, r=Vec()){ // 2x2 subpixel cols
85.                     for (int s=0; s<samps; s++){
86.                         double r1=2*erand48(Xi), dx=r1<1 ? sqrt(r1)-1: 1-sqrt(2-r1);
87.                         double r2=2*erand48(Xi), dy=r2<1 ? sqrt(r2)-1: 1-sqrt(2-r2);
88.                         Vec d = cx*( ( (sx+.5 + dx)/2 + x)/w - .5) +
89.                             cy*( ( (sy+.5 + dy)/2 + y)/h - .5) + cam.d;
90.                         r = r + radiance(Ray(cam.o+d*140,d.norm()),0,Xi)*(1./samps);
91.                     } // Camera rays are pushed ^^^^ forward to start in interior
92.                     c[i] = c[i] + Vec(clamp(r.x),clamp(r.y),clamp(r.z))**.25;
93.                 }
94.             }
95.         FILE *f = fopen("image.ppm", "w"); // Write image to PPM file.
96.         fprintf(f, "P3\n%d %d\n%d\n", w, h, 255);
97.         for (int i=0; i<w*h; i++)
98.             fprintf(f,"%d %d %d ", toInt(c[i].x), toInt(c[i].y), toInt(c[i].z));
99.     }
```


directions for making pictures using numbers
 (explained using only the ten hundred words people use most often)

