Rendering equation and path tracing


15-468, 15-668, 15-868
Physics-based Rendering
Spring 2024, Lecture 11

http://graphics.cs.cmu.edu/courses/15-468

Course announcements

- All quizzes up to TQ4 graded on Canvas!

Overview of today’s lecture

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

Slide credits

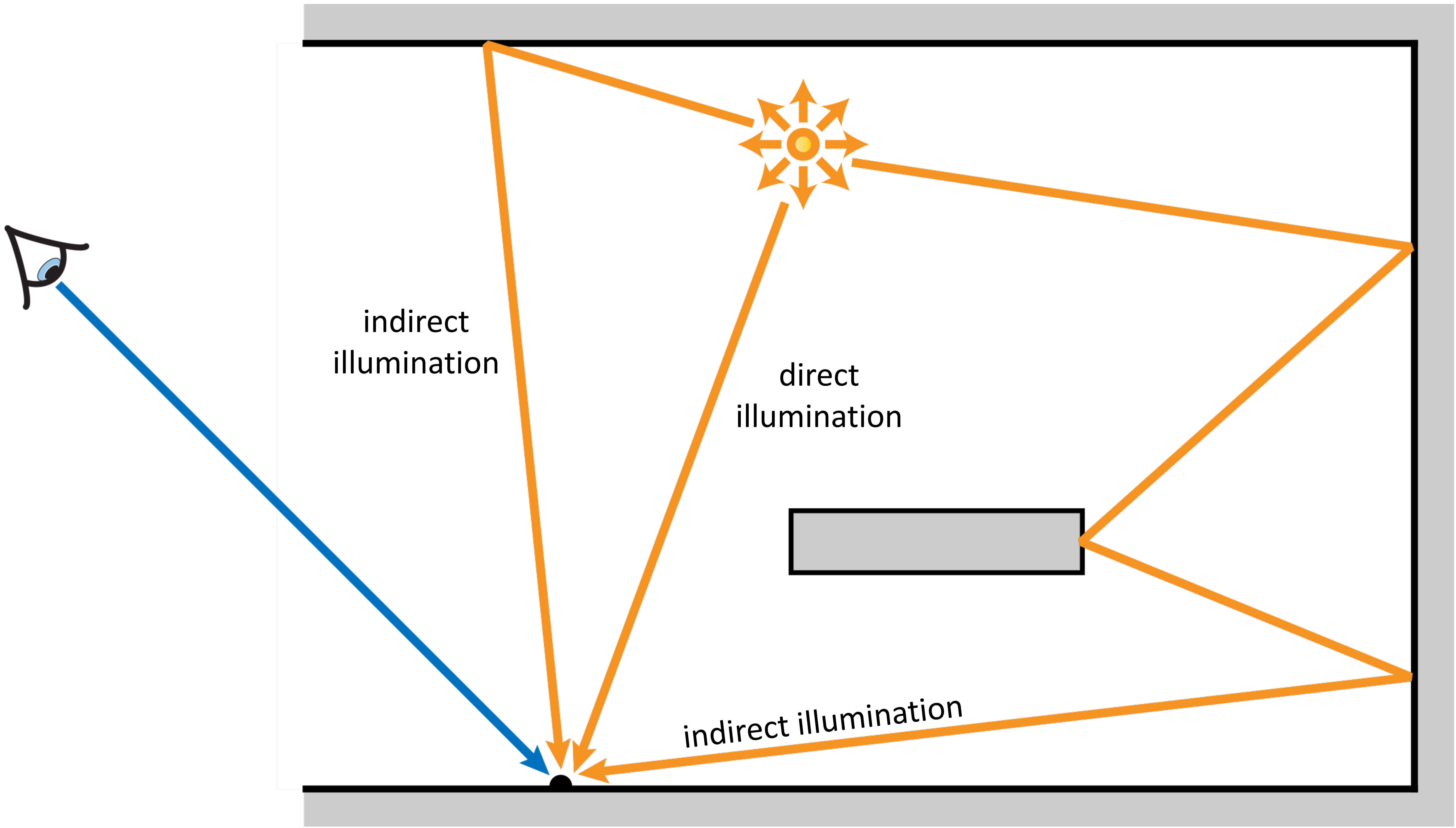
Most of these slides were directly adapted from:

- Wojciech Jarosz (Dartmouth).

Direct vs. Indirect Illumination

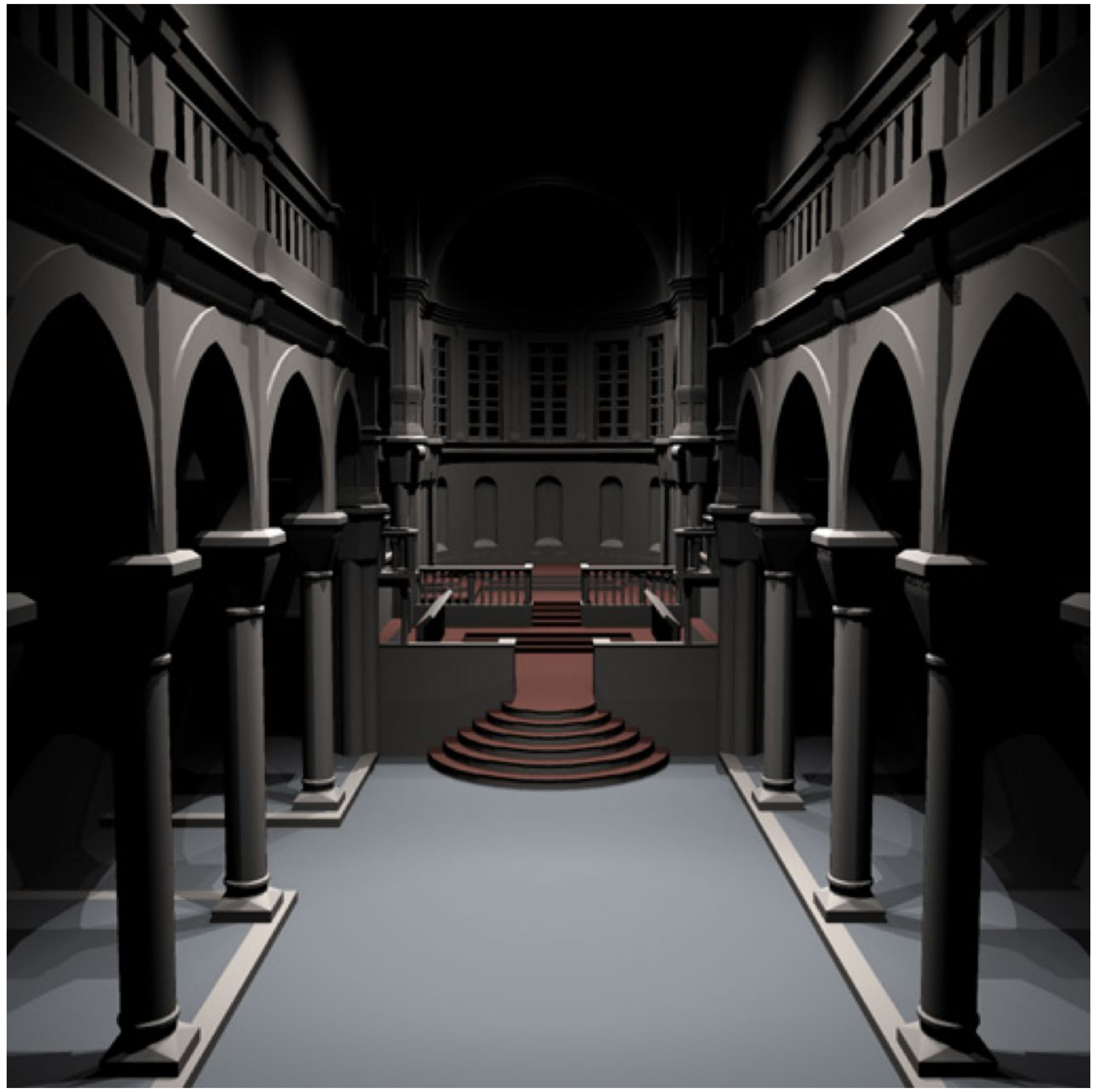
Where does L_i "come from"?

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



Direct vs. Indirect Illumination

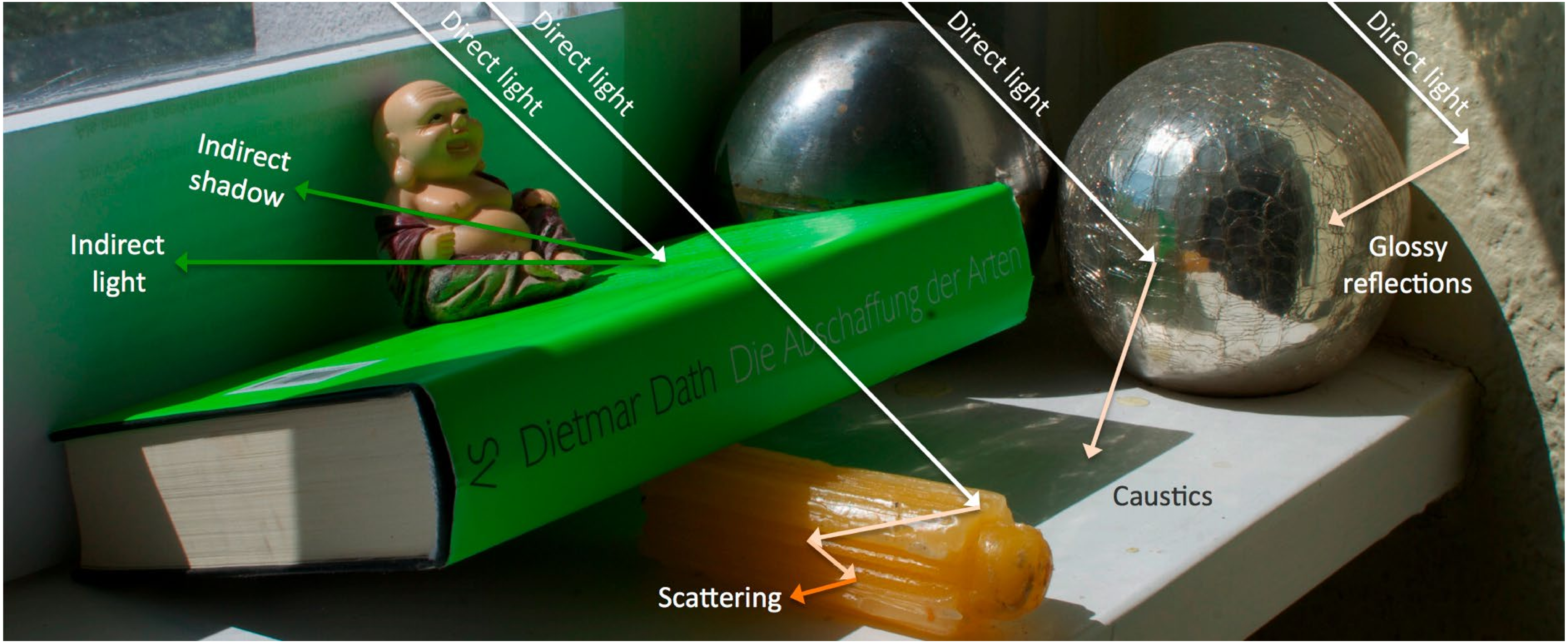
Direct illumination



Indirect illumination

Direct + indirect illumination

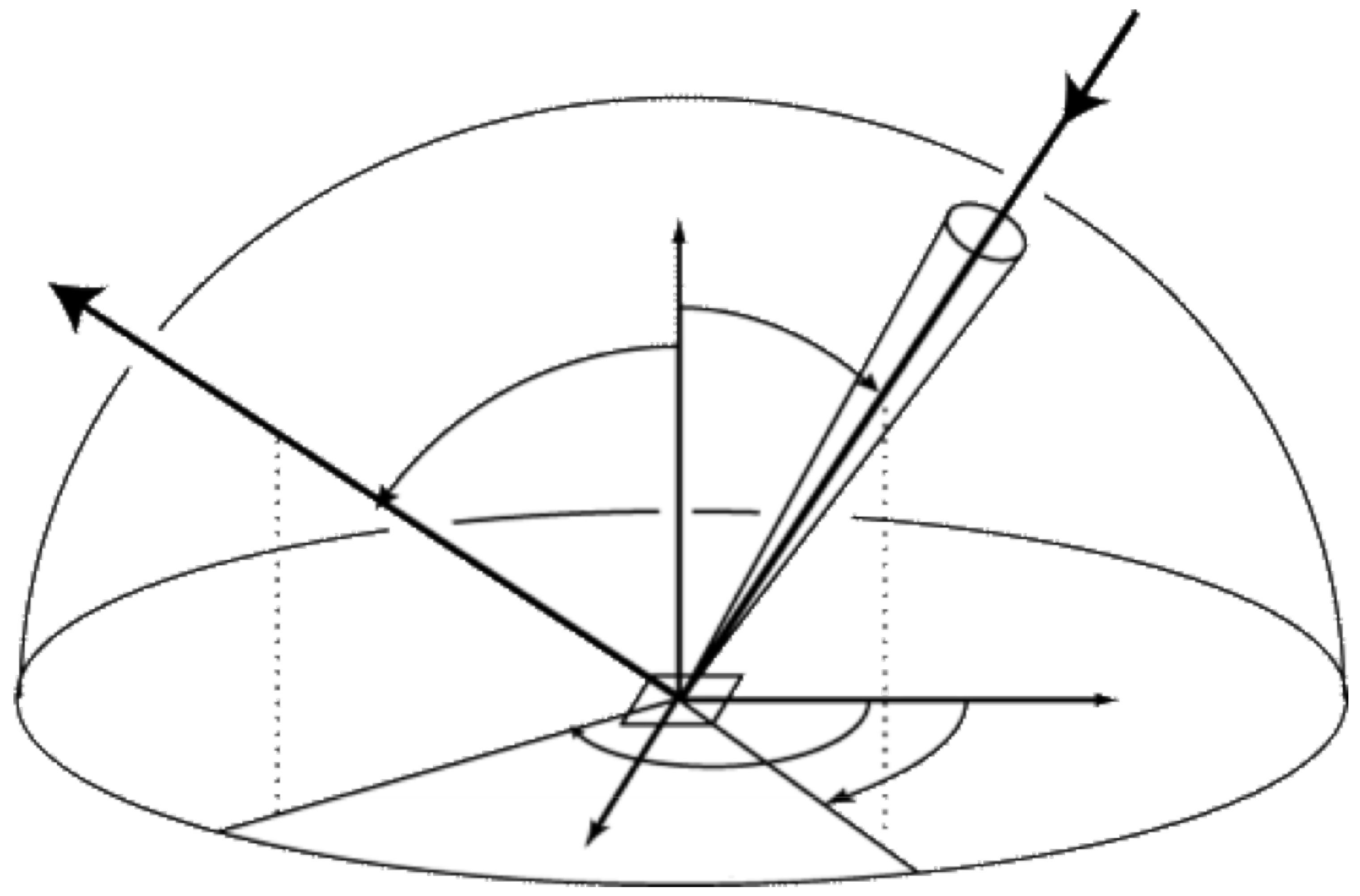
All-in-One!



Ritschel et al. [2012]

Reflection Equation

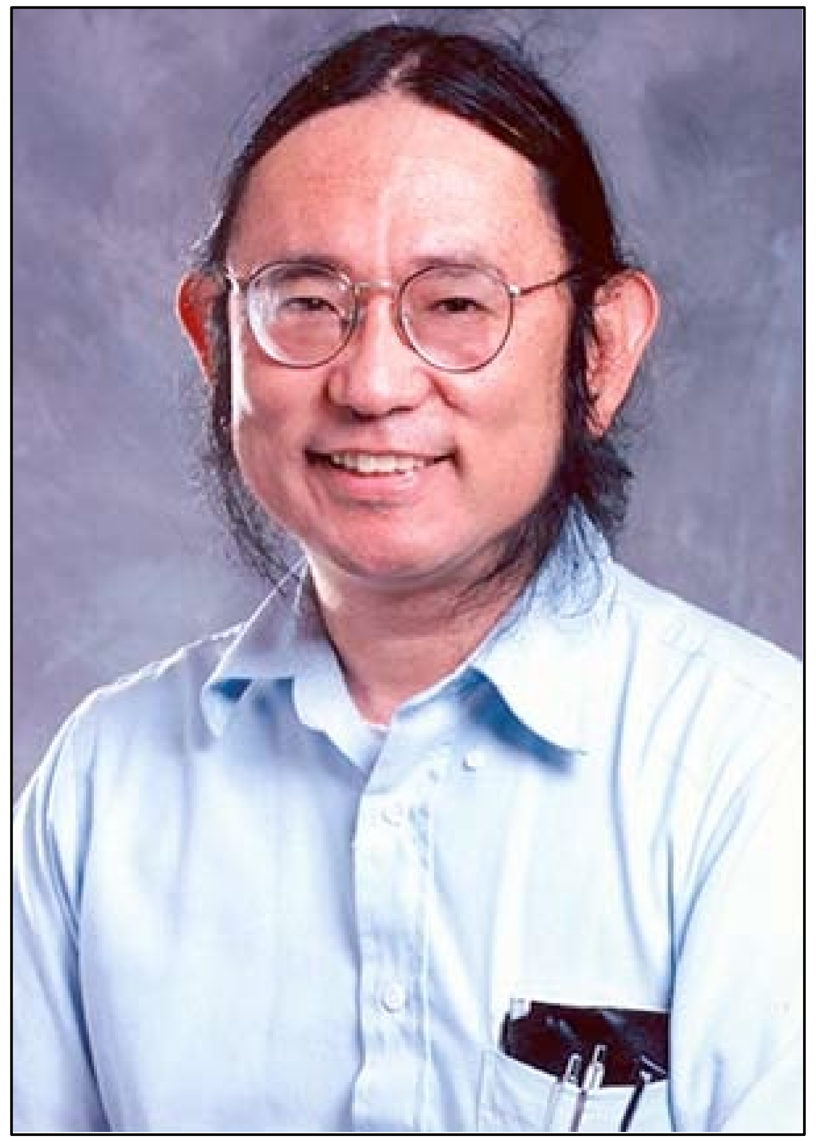
Reflected radiance is the weighted integral of incident radiance



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

Rendering Equation

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



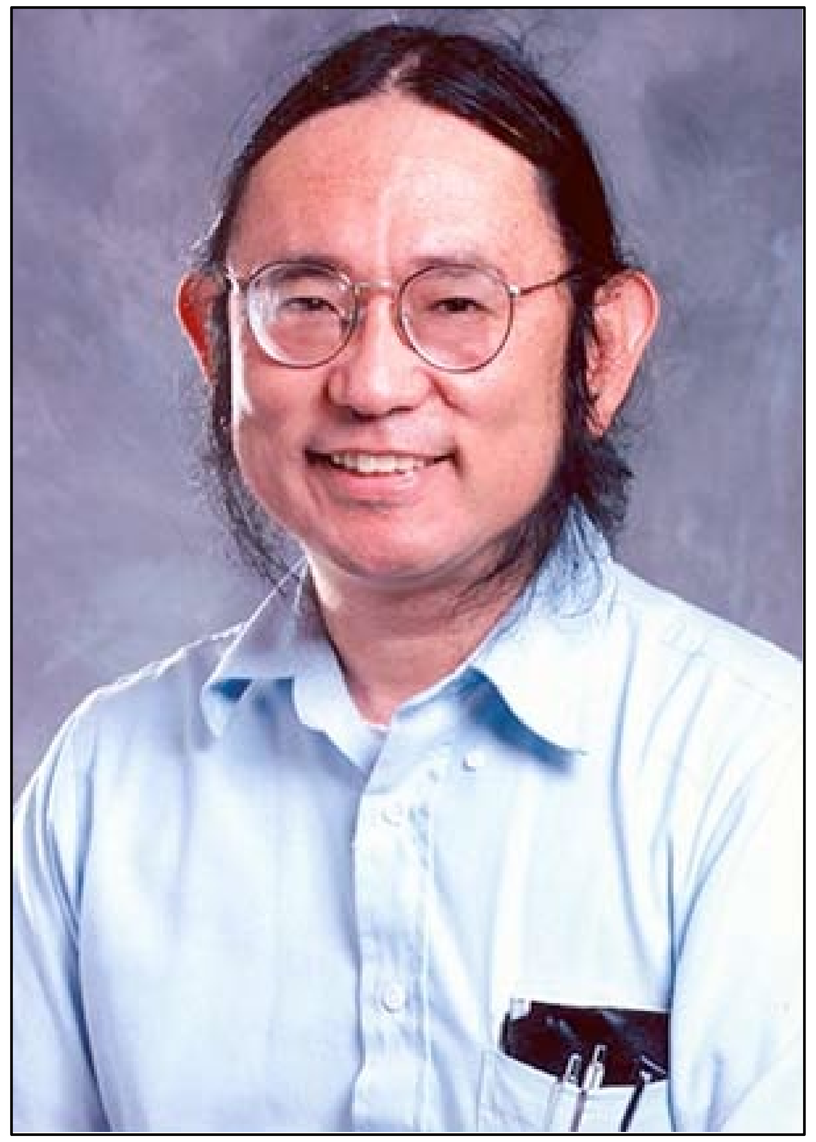
Energy equilibrium:

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

outgoing emitted reflected

Rendering Equation

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



Energy equilibrium:

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

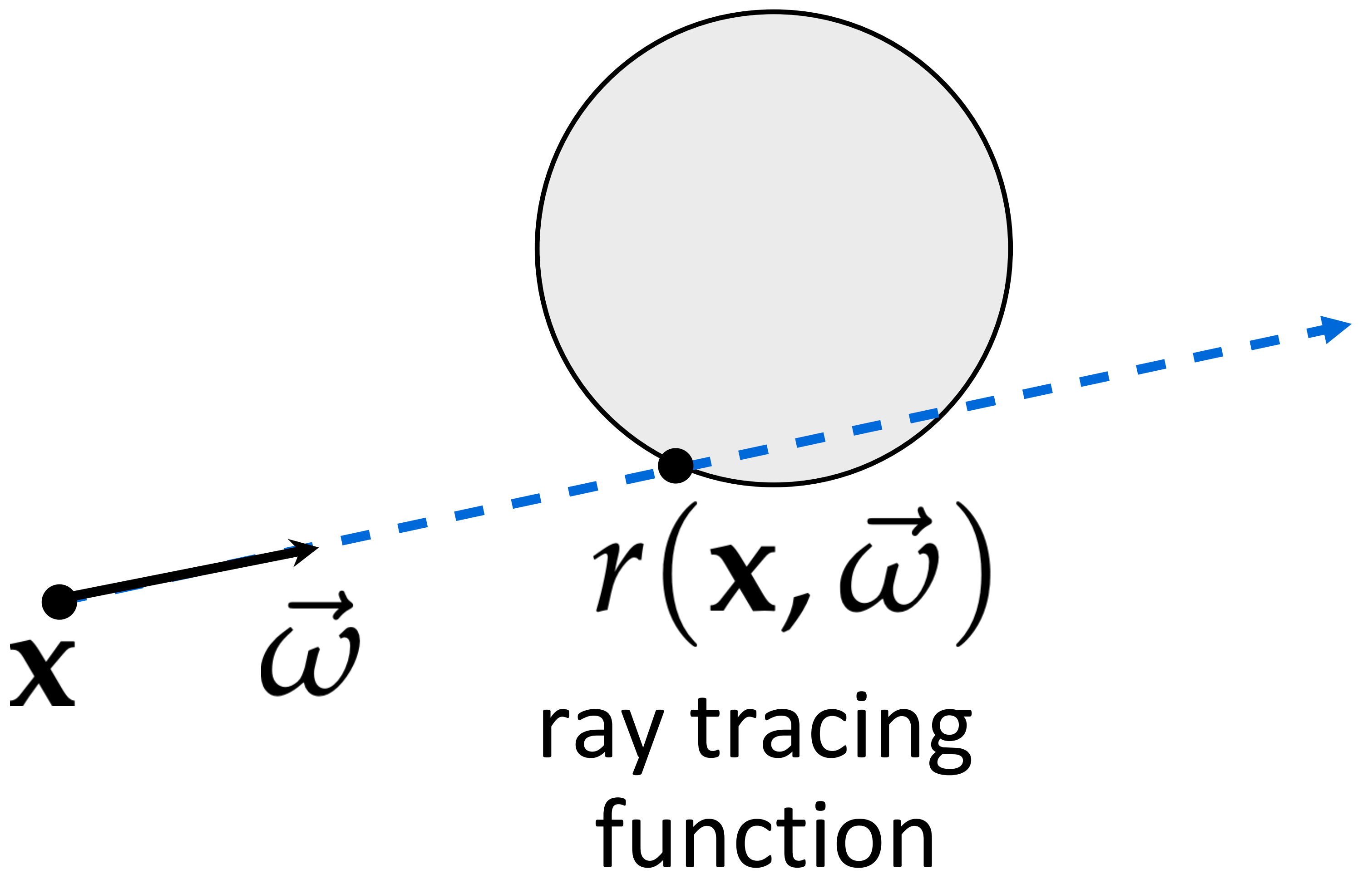
outgoing — emitted — reflected

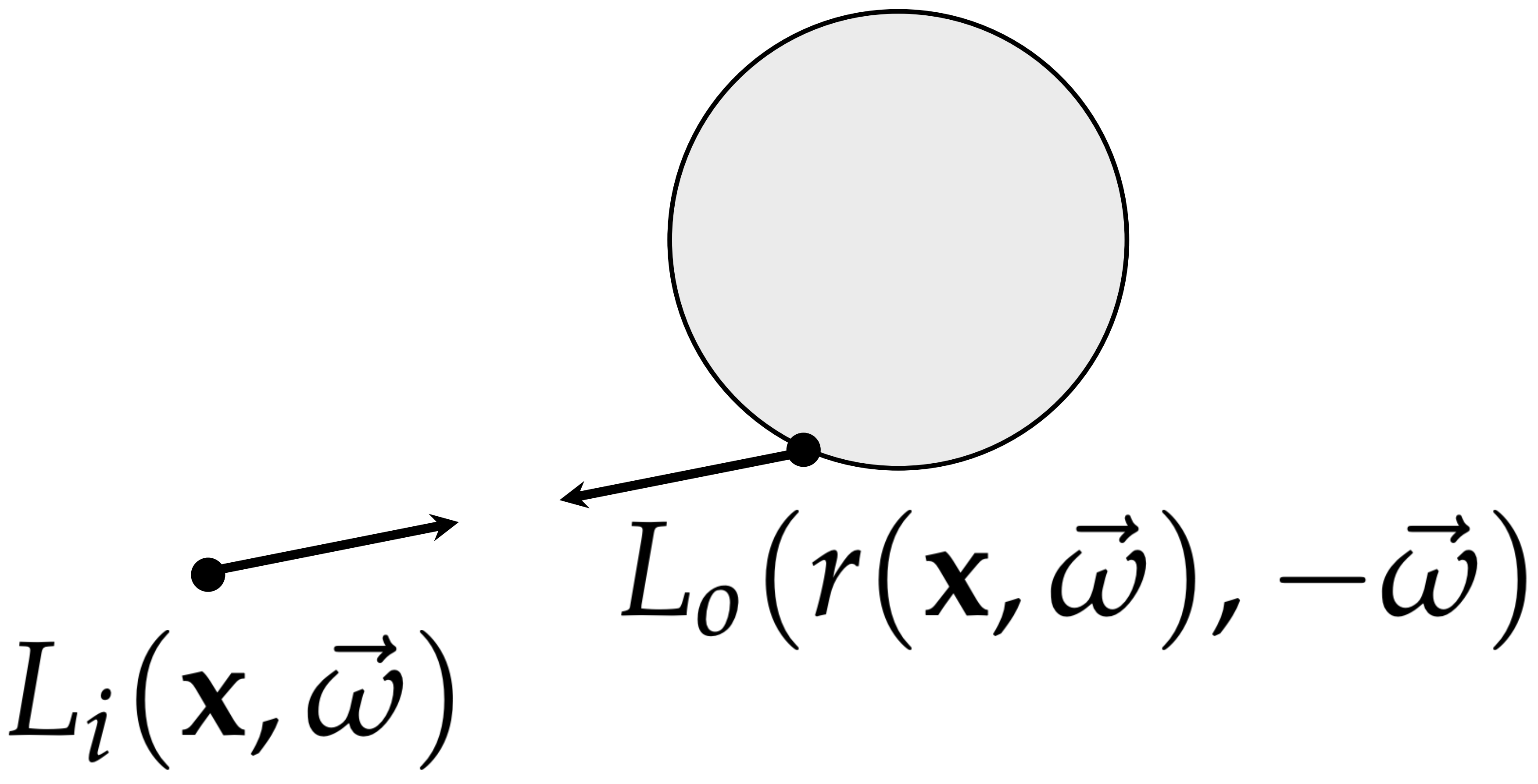
Light Transport

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

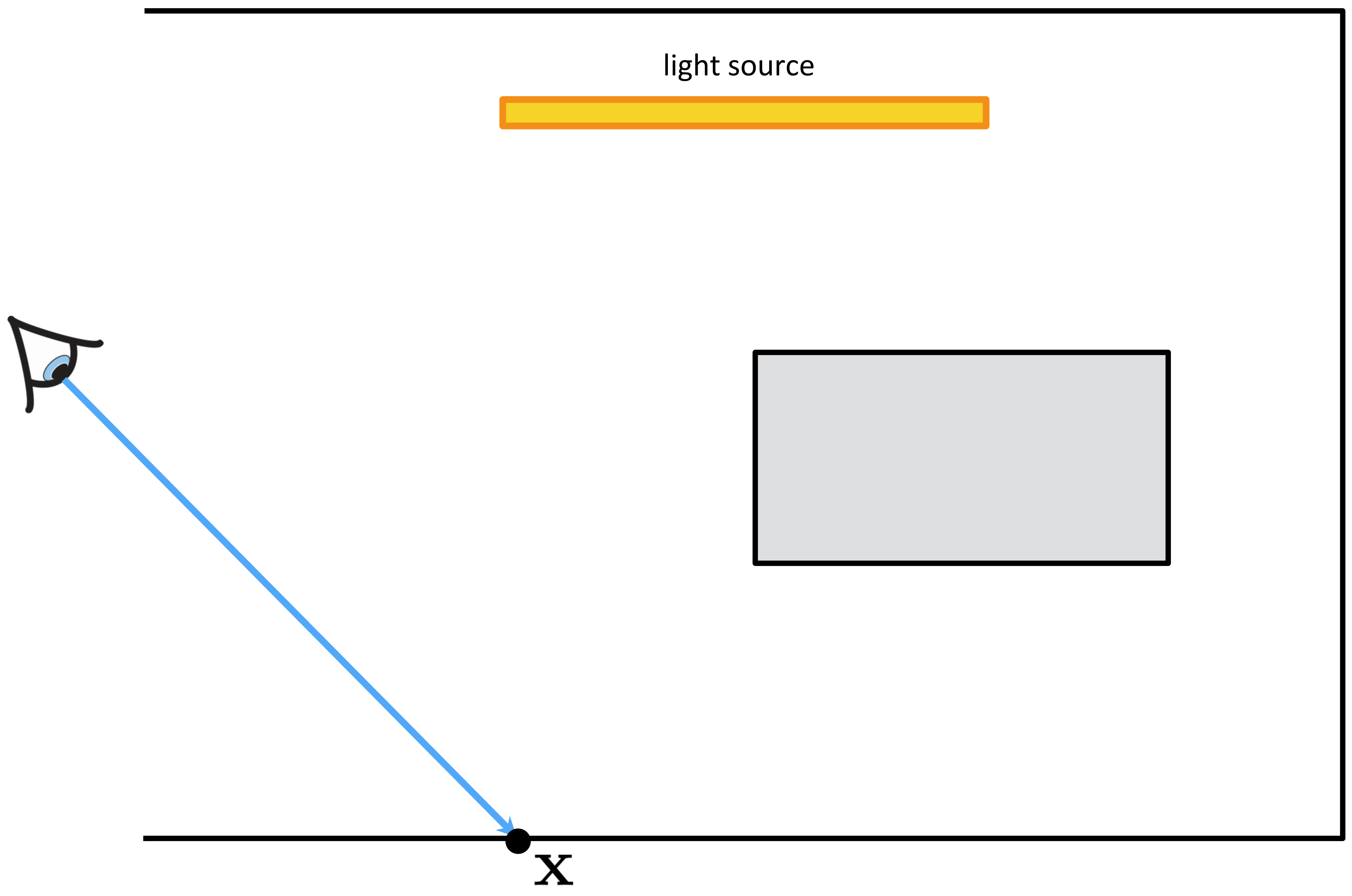
$$L(\mathbf{x},\vec{\omega}) = L_e(\mathbf{x},\vec{\omega}) + \int_{H^2} f_r(\mathbf{x},\vec{\omega}',\vec{\omega}) L(r(\mathbf{x},\vec{\omega}'),-\vec{\omega}') \cos\theta' \mathrm{d}\vec{\omega}'$$

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)

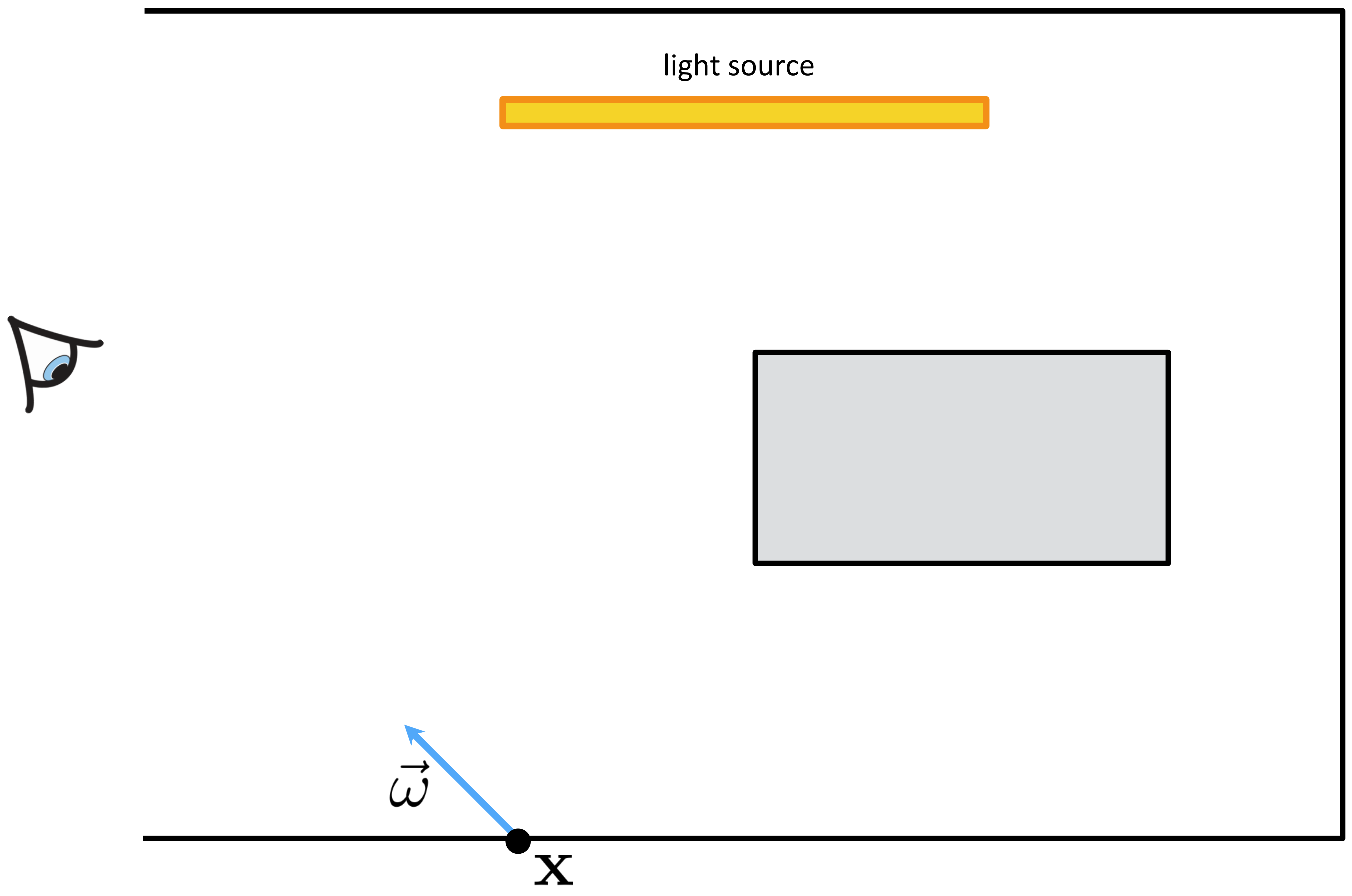
Rendering Equation

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



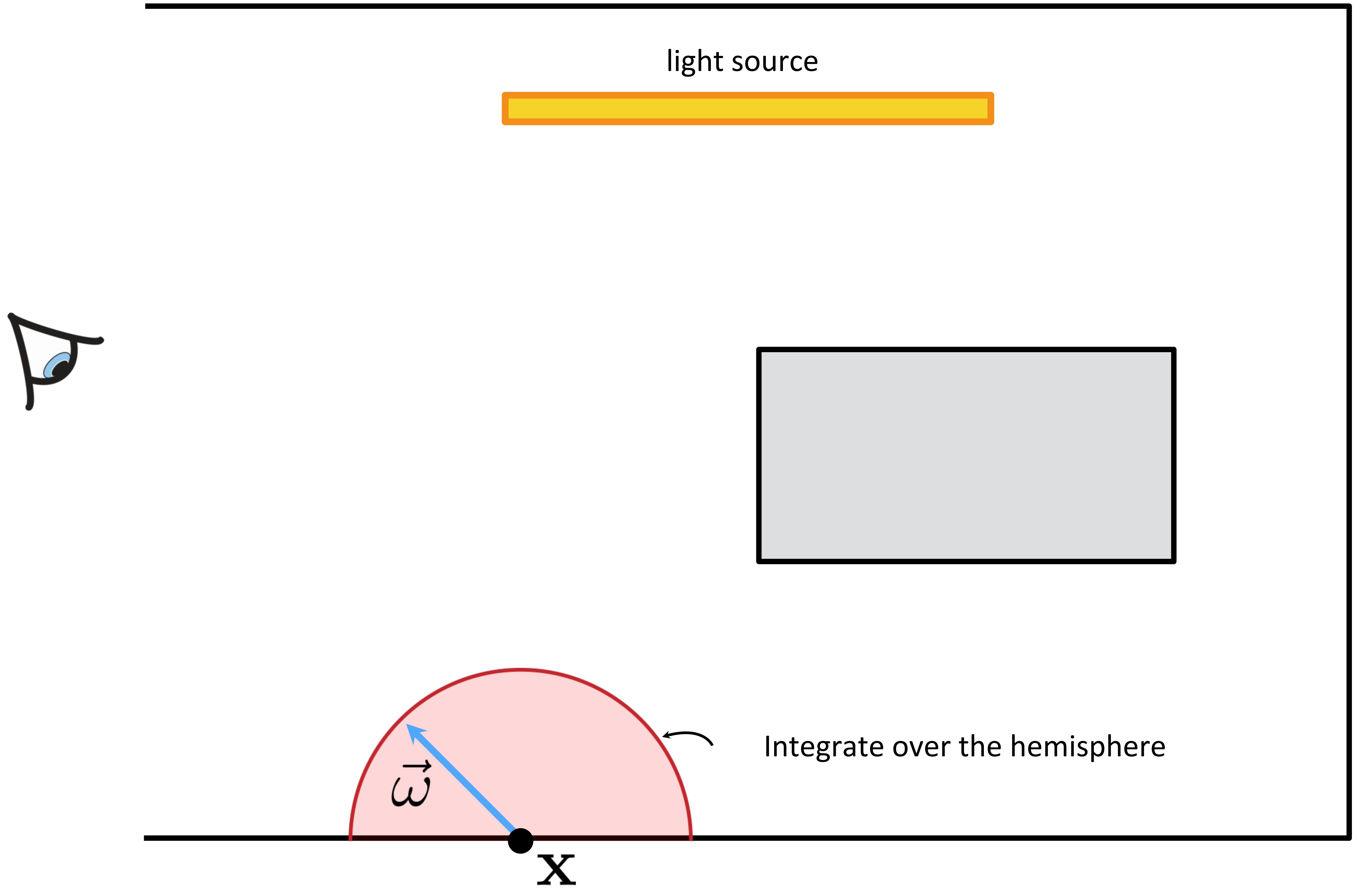
Rendering Equation

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



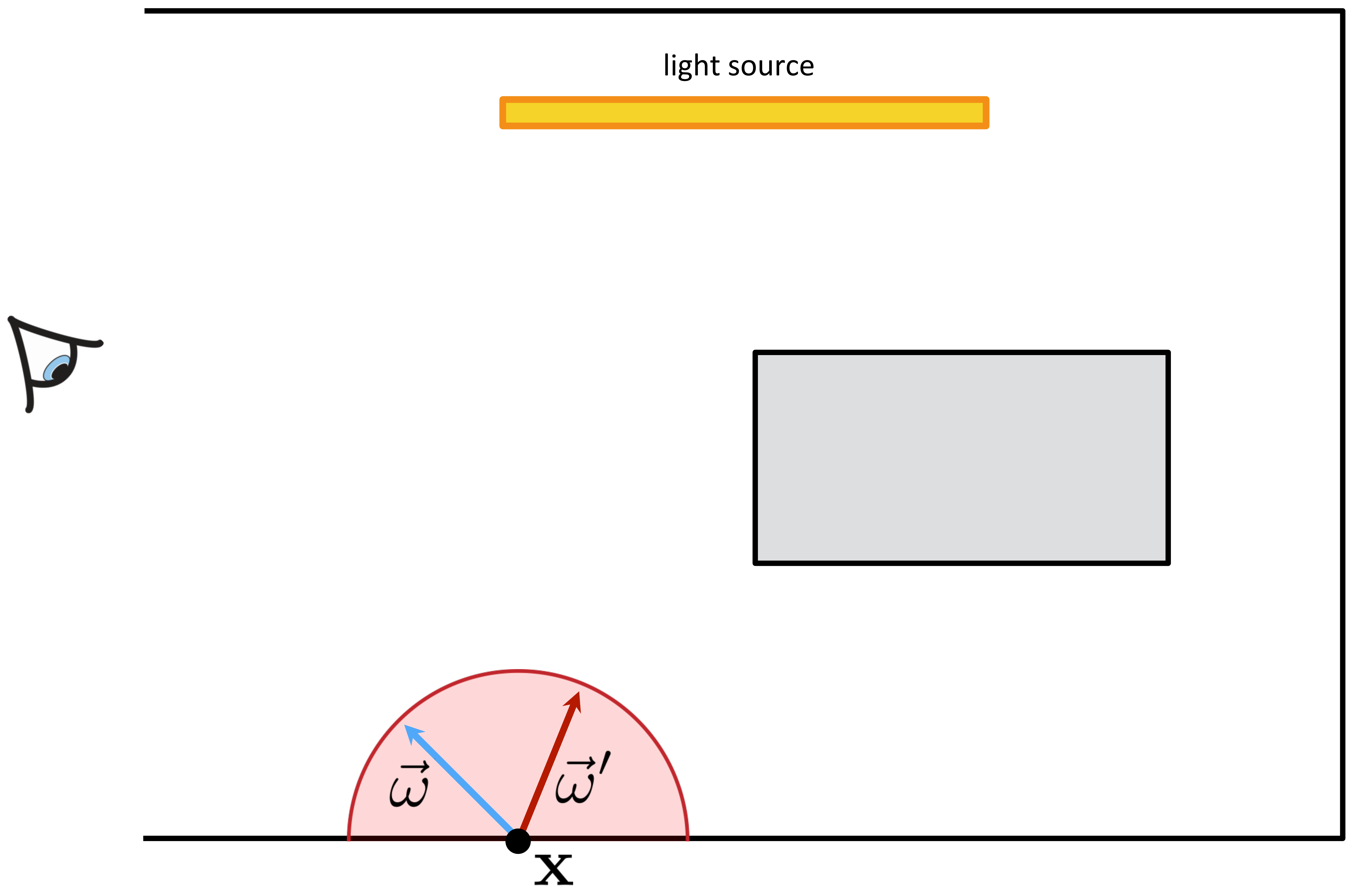
Rendering Equation

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



Rendering Equation

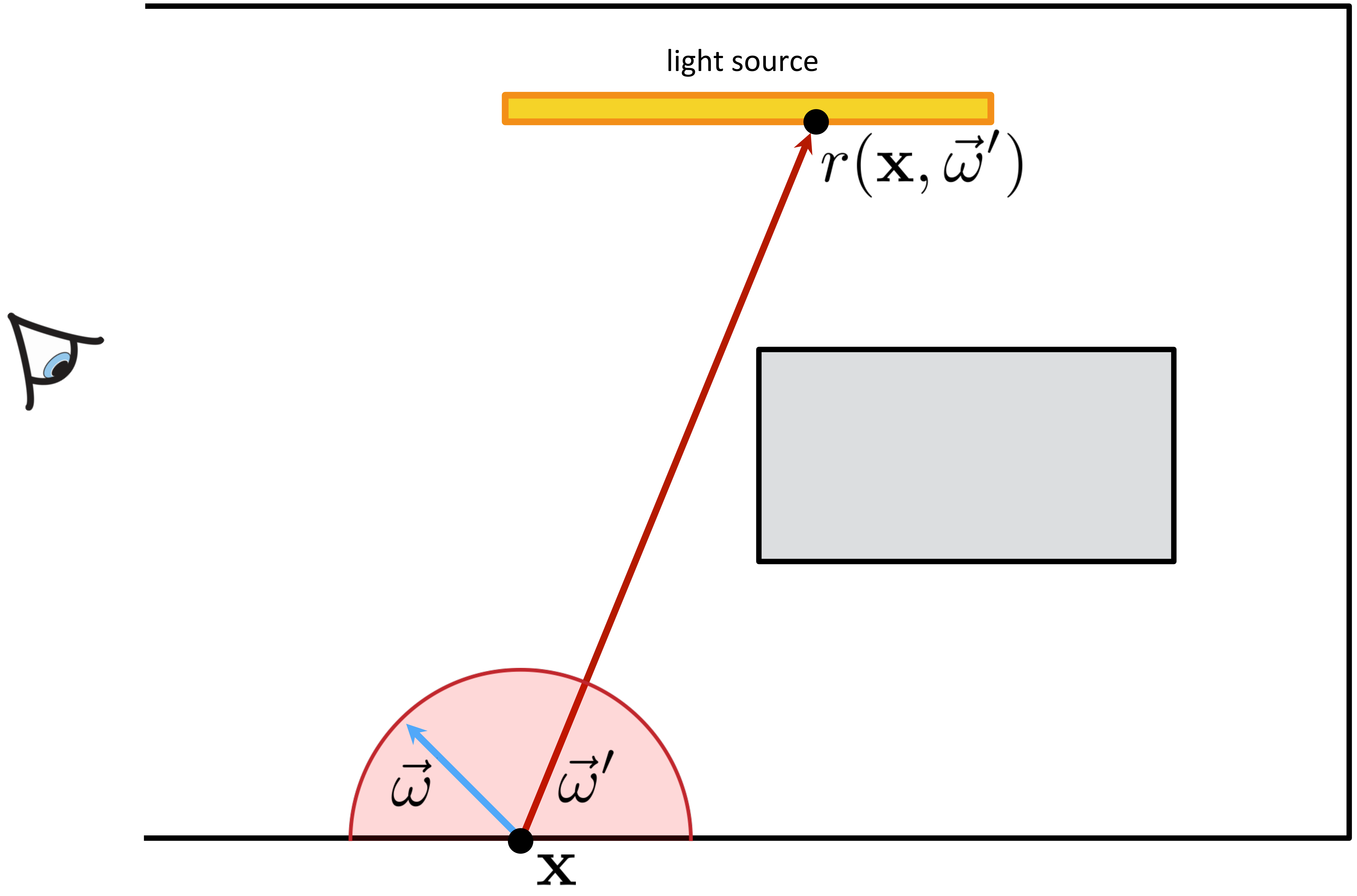
$$L(\mathbf{x},\vec{\omega}) = L_e(\mathbf{x},\vec{\omega}) + \int_{H^2} f_r(\mathbf{x},\vec{\omega}',\vec{\omega})L(r(\mathbf{x},\vec{\omega}'),-\vec{\omega}')\cos\theta'\,\mathrm{d}\vec{\omega}'$$



Rendering Equation

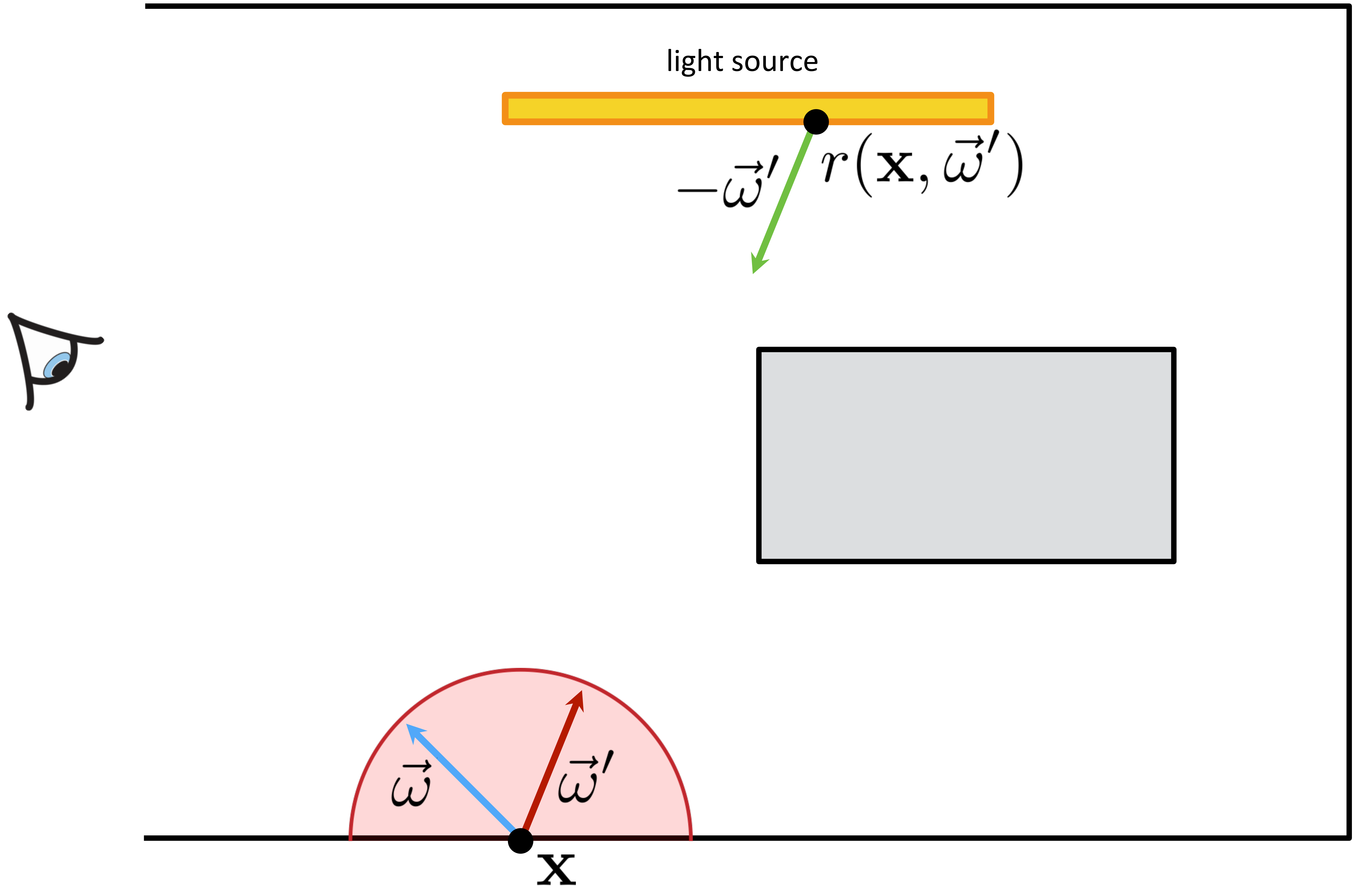
ray tracing function

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



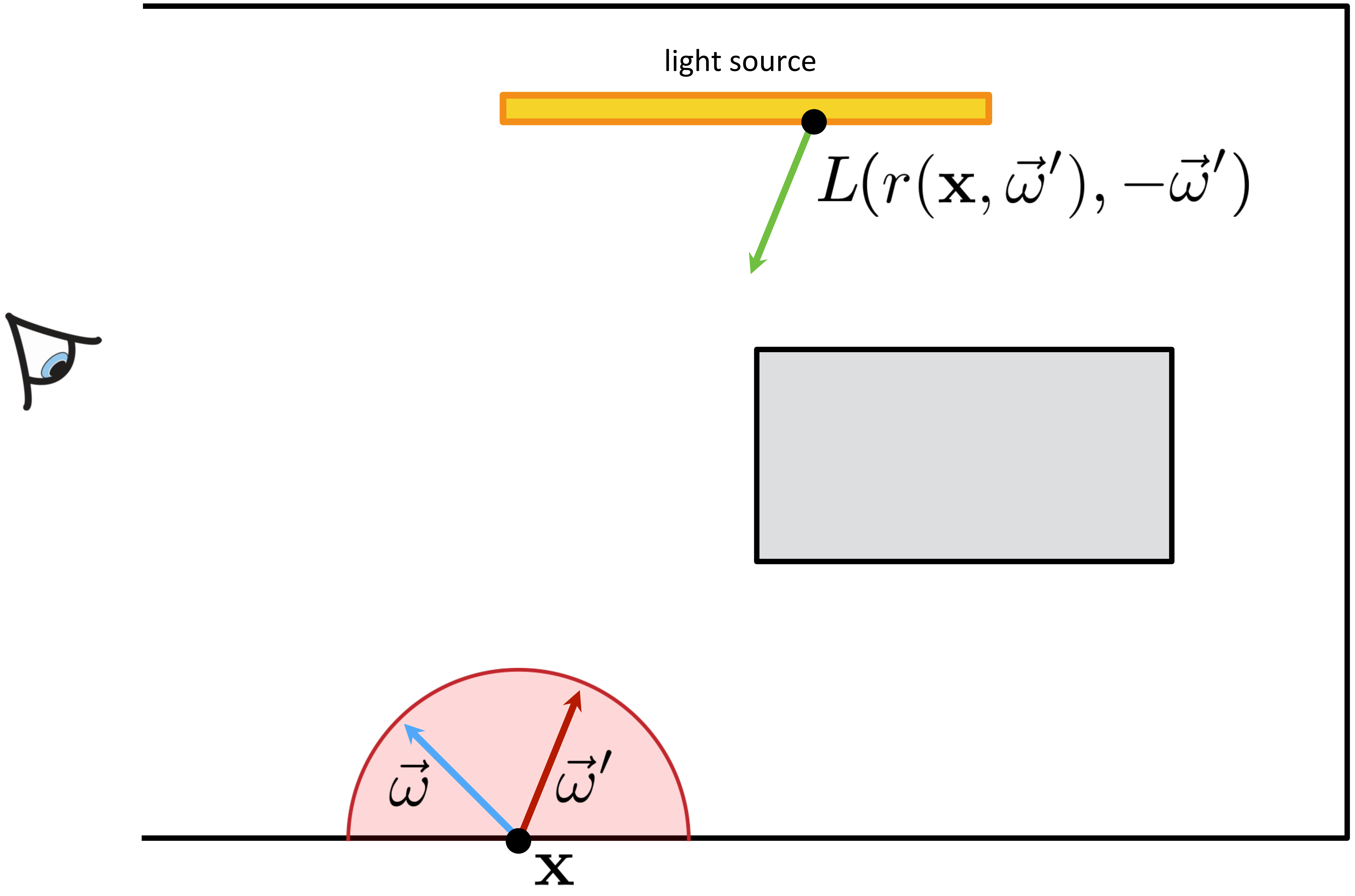
Rendering Equation

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



Rendering Equation

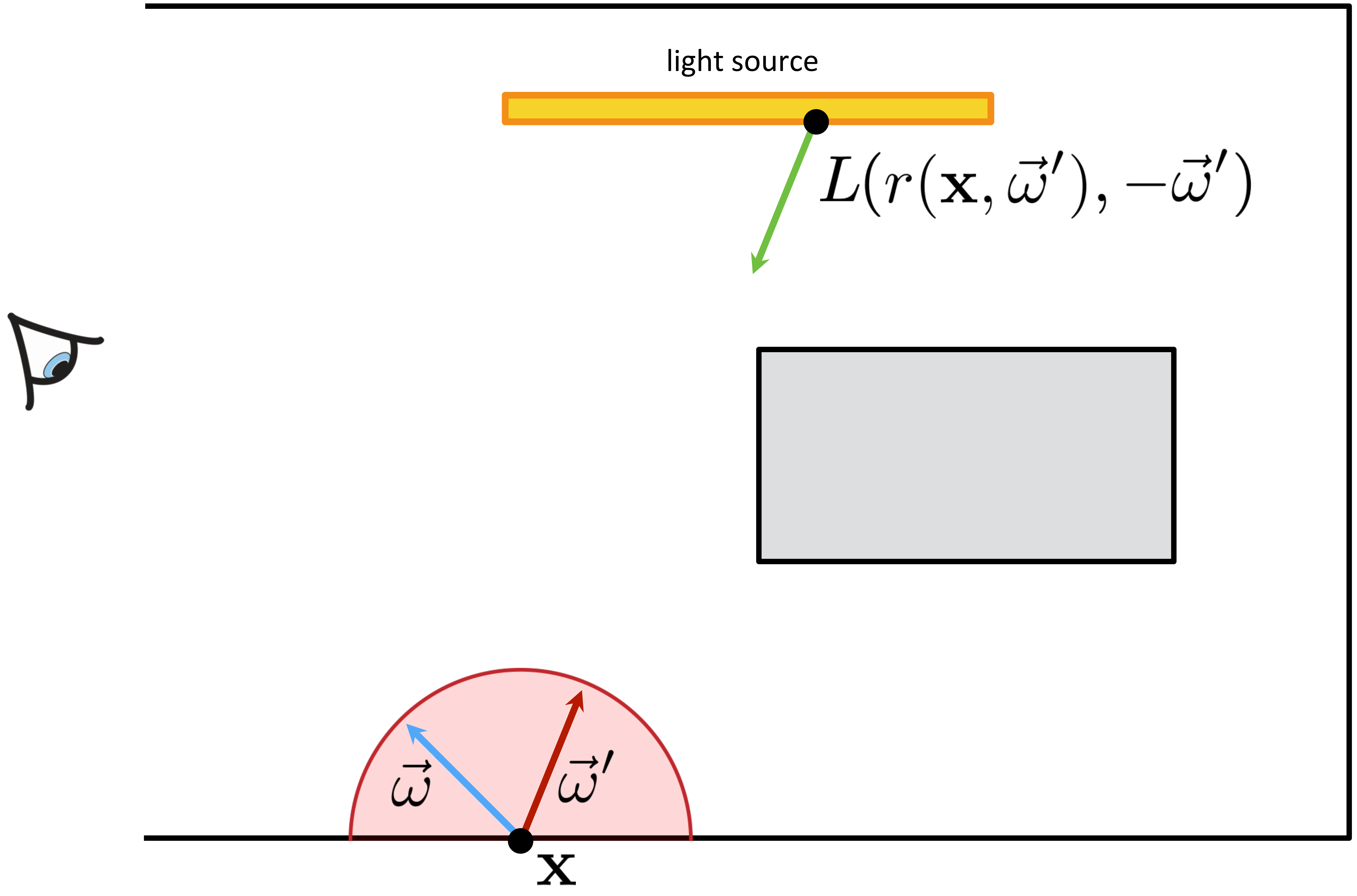
$$L(\mathbf{x},\vec{\omega}) = L_e(\mathbf{x},\vec{\omega}) + \int_{H^2} f_r(\mathbf{x},\vec{\omega}',\vec{\omega}) L(r(\mathbf{x},\vec{\omega}'),-\vec{\omega}') \cos\theta' \mathrm{d}\vec{\omega}'$$



Rendering Equation

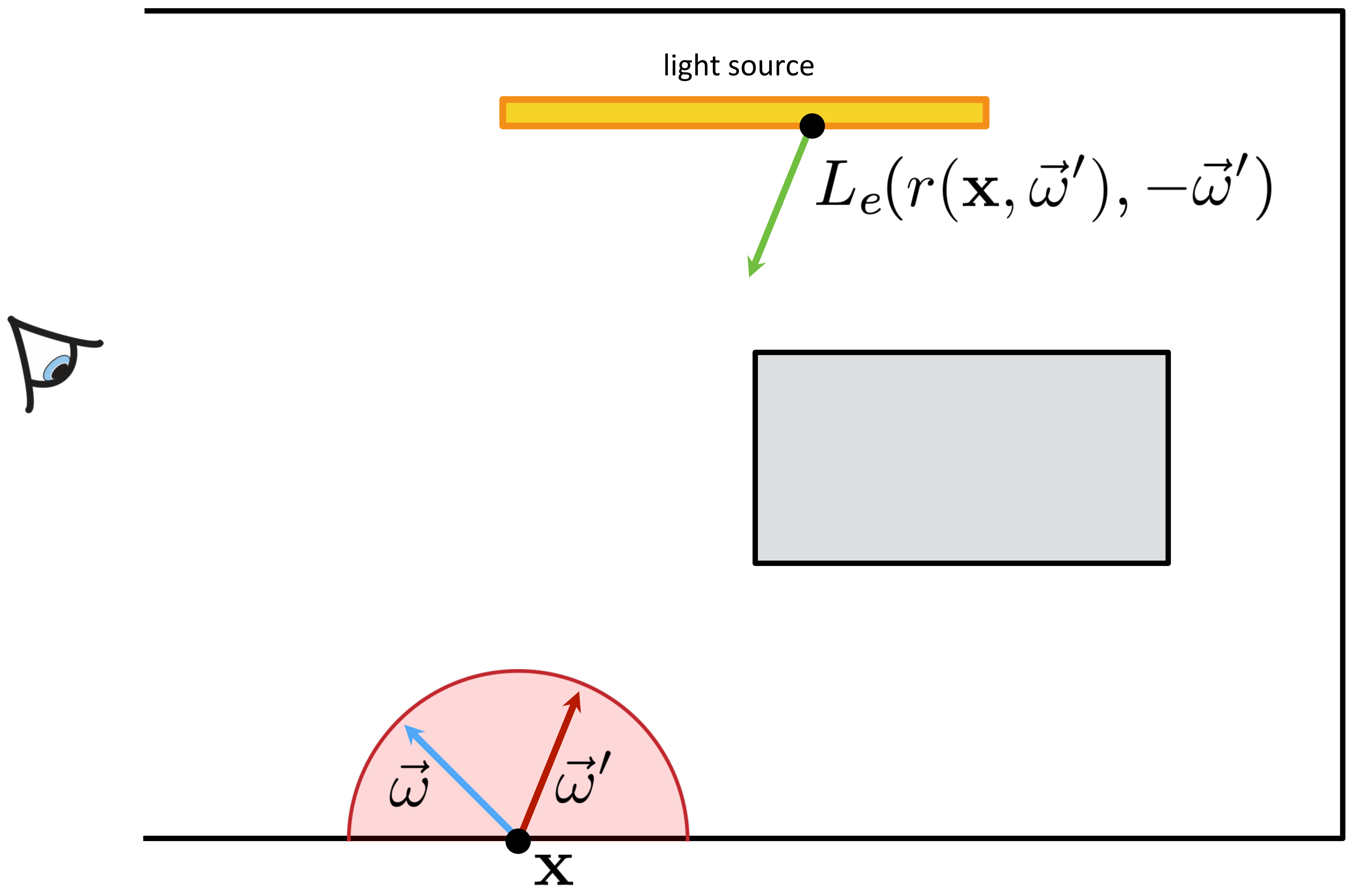
$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos\theta' \mathrm{d}\vec{\omega}'$$

recursion



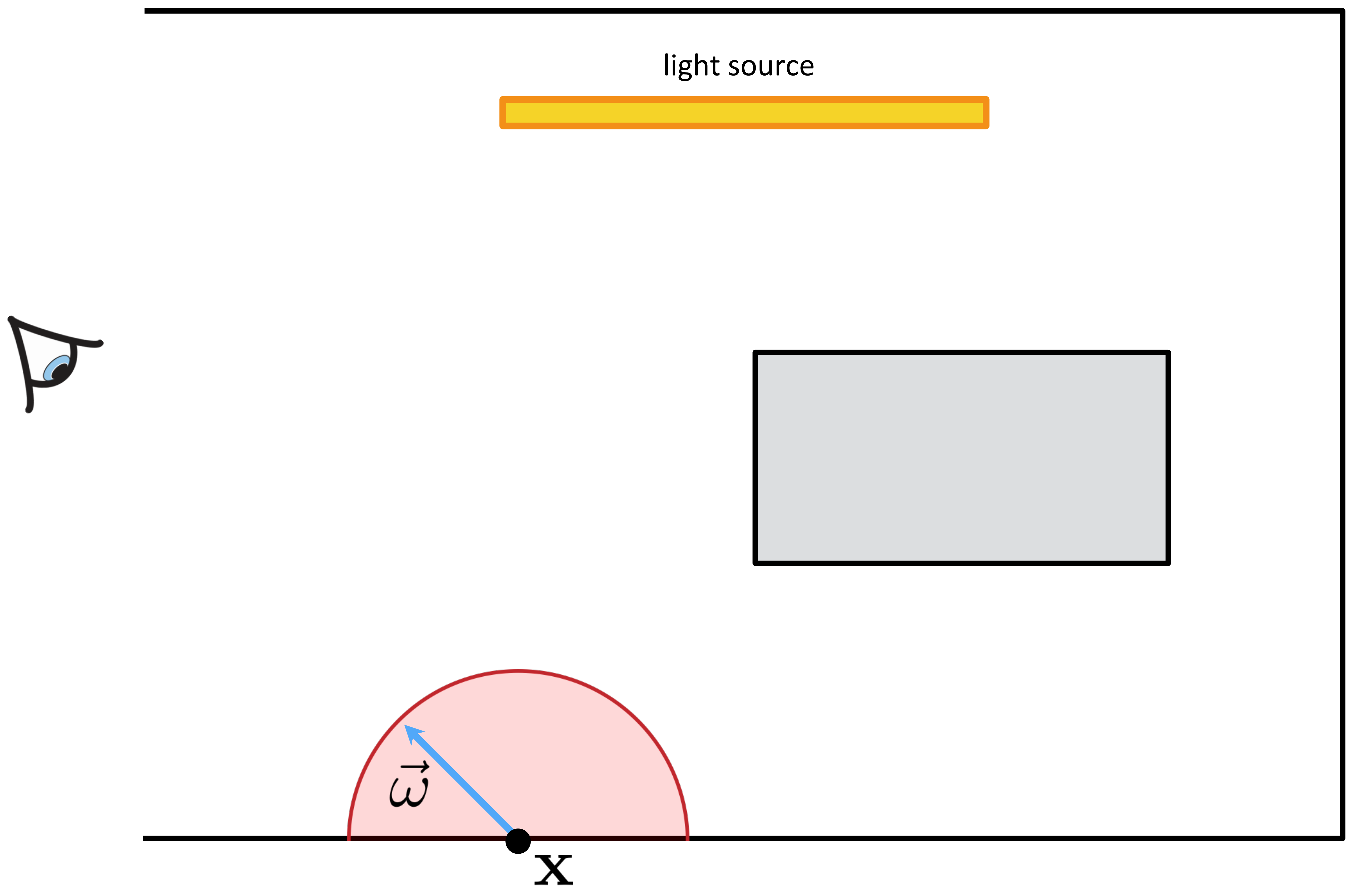
Rendering Equation

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



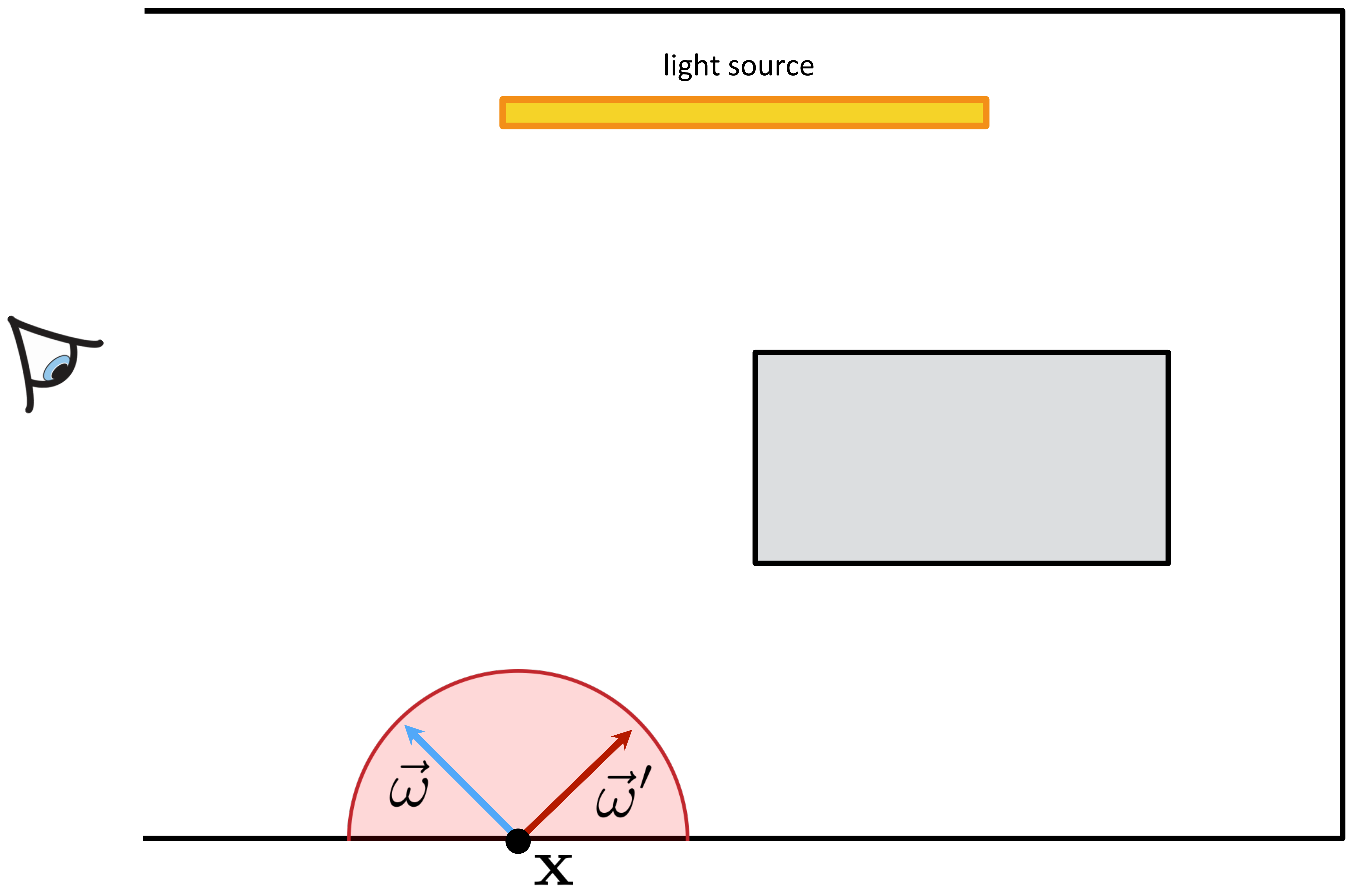
Rendering Equation

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



Rendering Equation

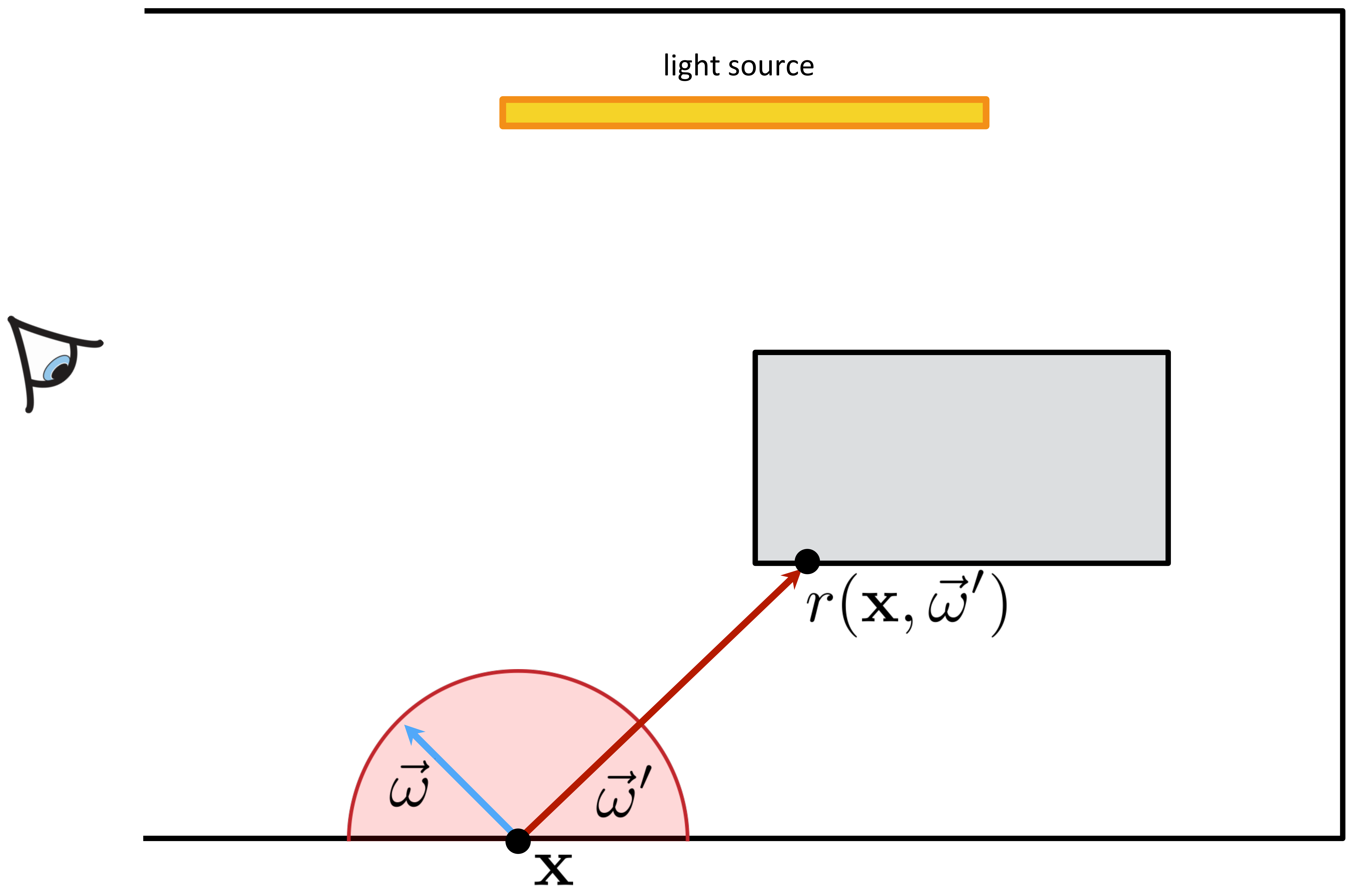
$$L(\mathbf{x},\vec{\omega}) = L_e(\mathbf{x},\vec{\omega}) + \int_{H^2} f_r(\mathbf{x},\vec{\omega}',\vec{\omega})L(r(\mathbf{x},\vec{\omega}'),-\vec{\omega}')\cos\theta'\,\mathrm{d}\vec{\omega}'$$



Rendering Equation

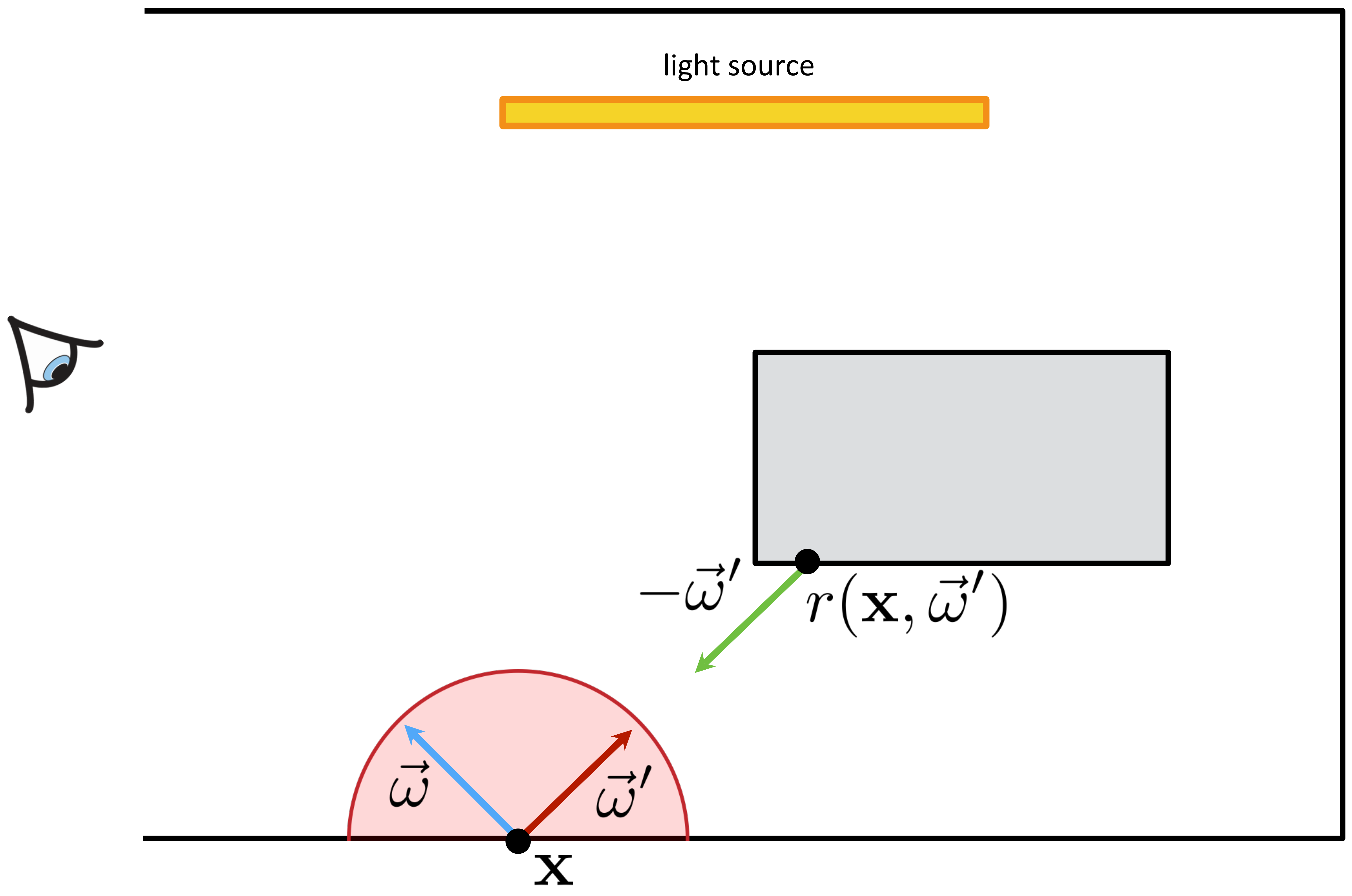
ray tracing function

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



Rendering Equation

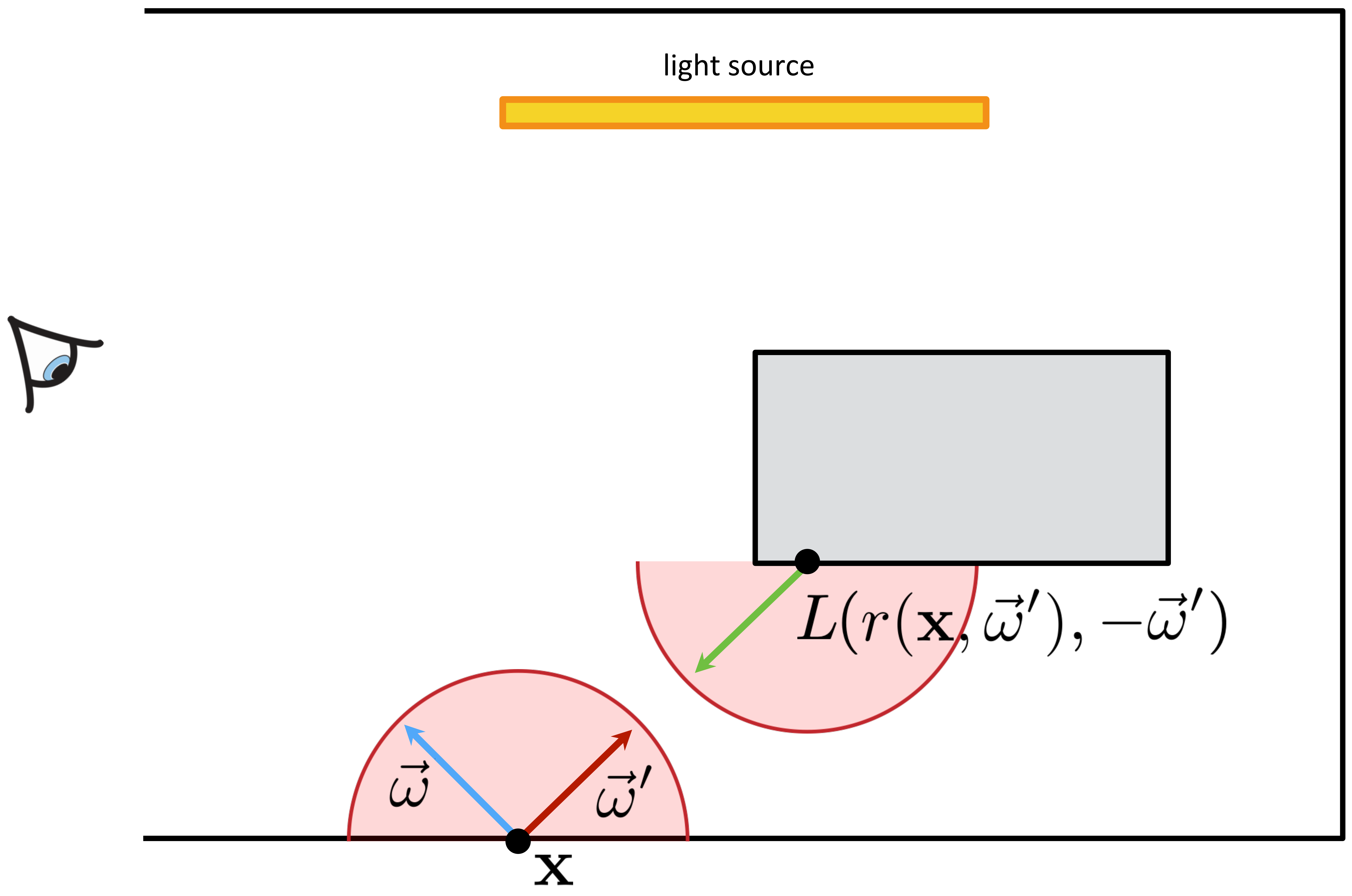
$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos\theta' \mathrm{d}\vec{\omega}'$$



Rendering Equation

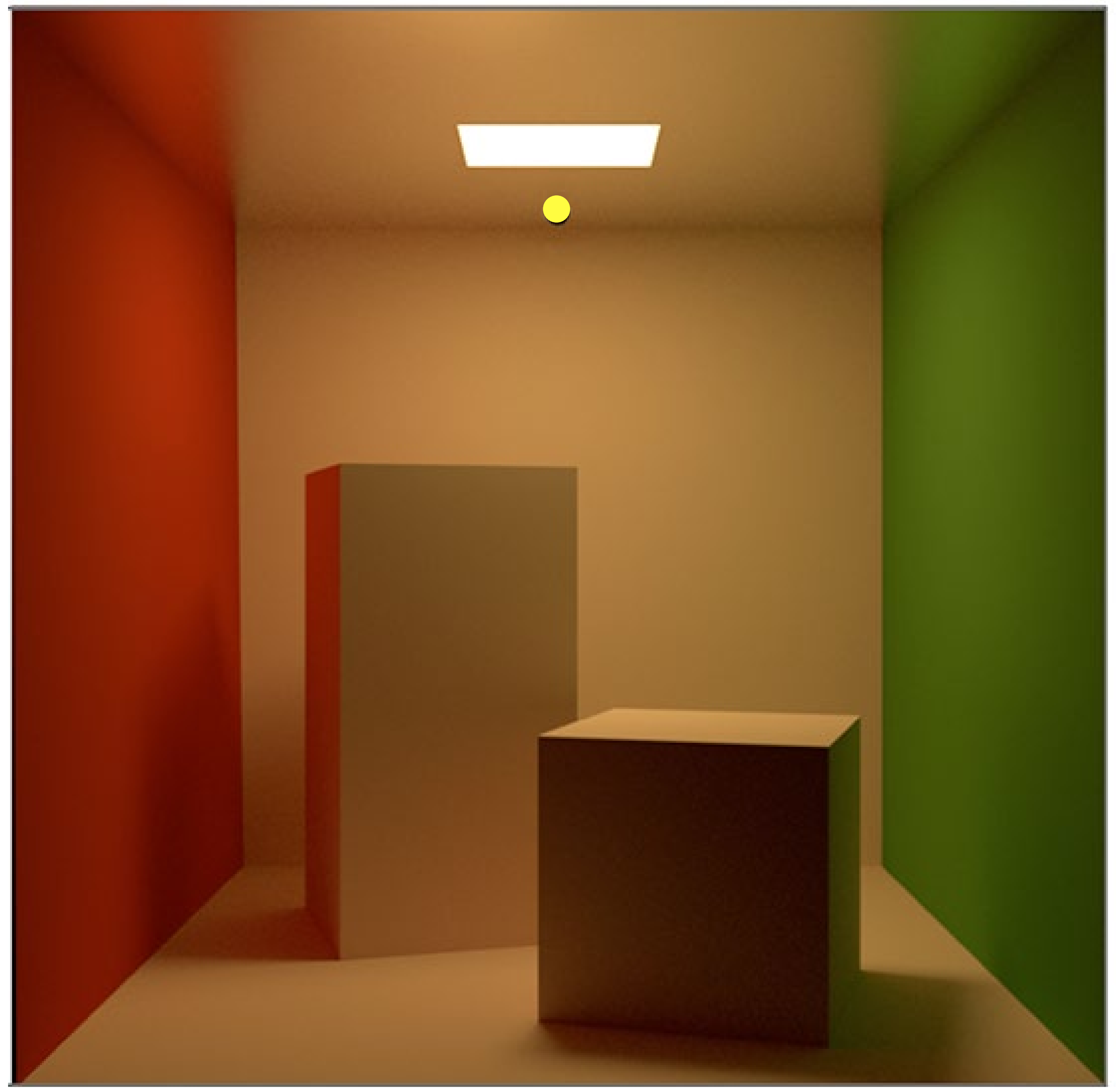
$$L(\mathbf{x},\vec{\omega}) = L_e(\mathbf{x},\vec{\omega}) + \int_{H^2} f_r(\mathbf{x},\vec{\omega}',\vec{\omega})L(r(\mathbf{x},\vec{\omega}'),-\vec{\omega}')\cos\theta' \mathrm{d}\vec{\omega}'$$

recursion



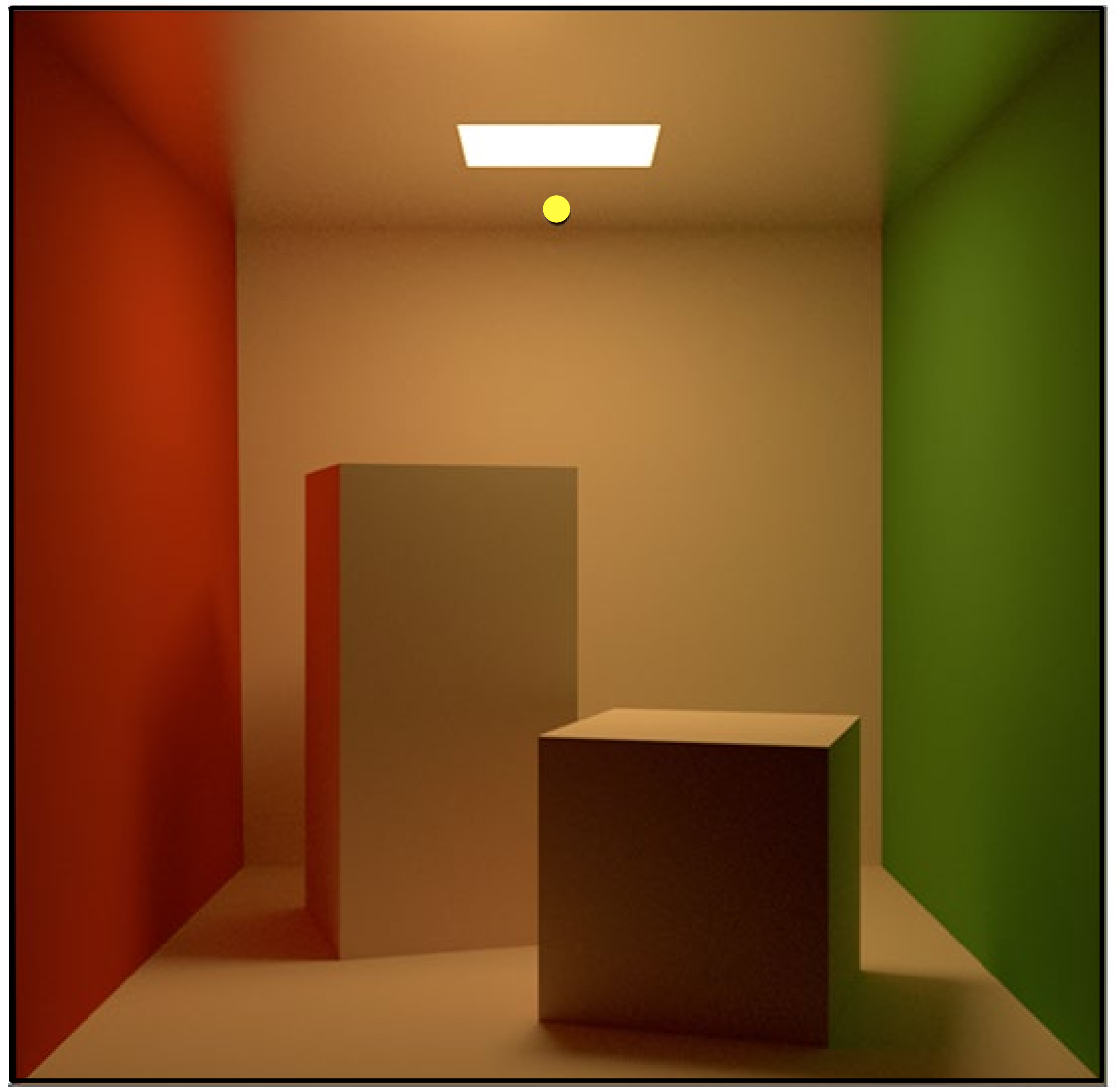
Rendering Equation

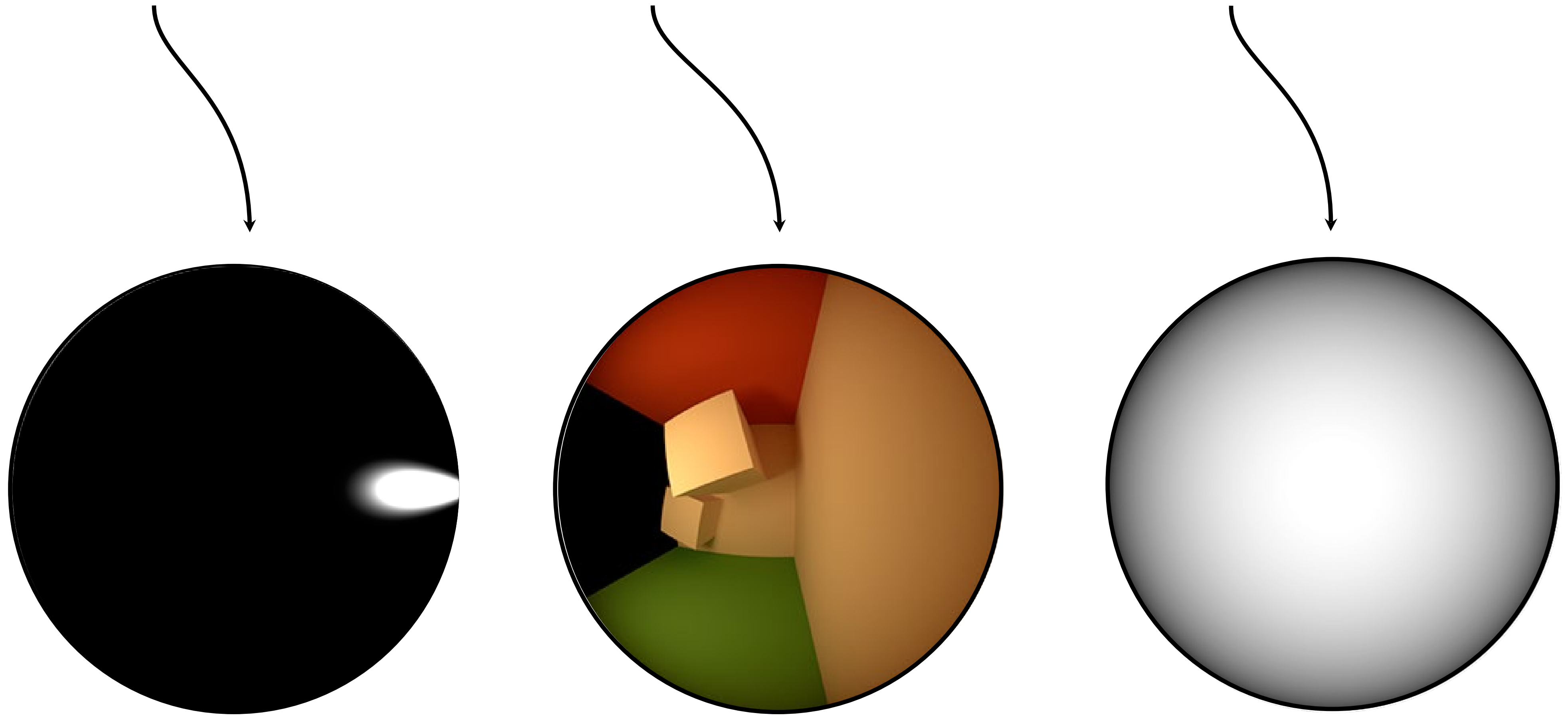
$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos\theta' \mathrm{d}\vec{\omega}'$$



Rendering Equation

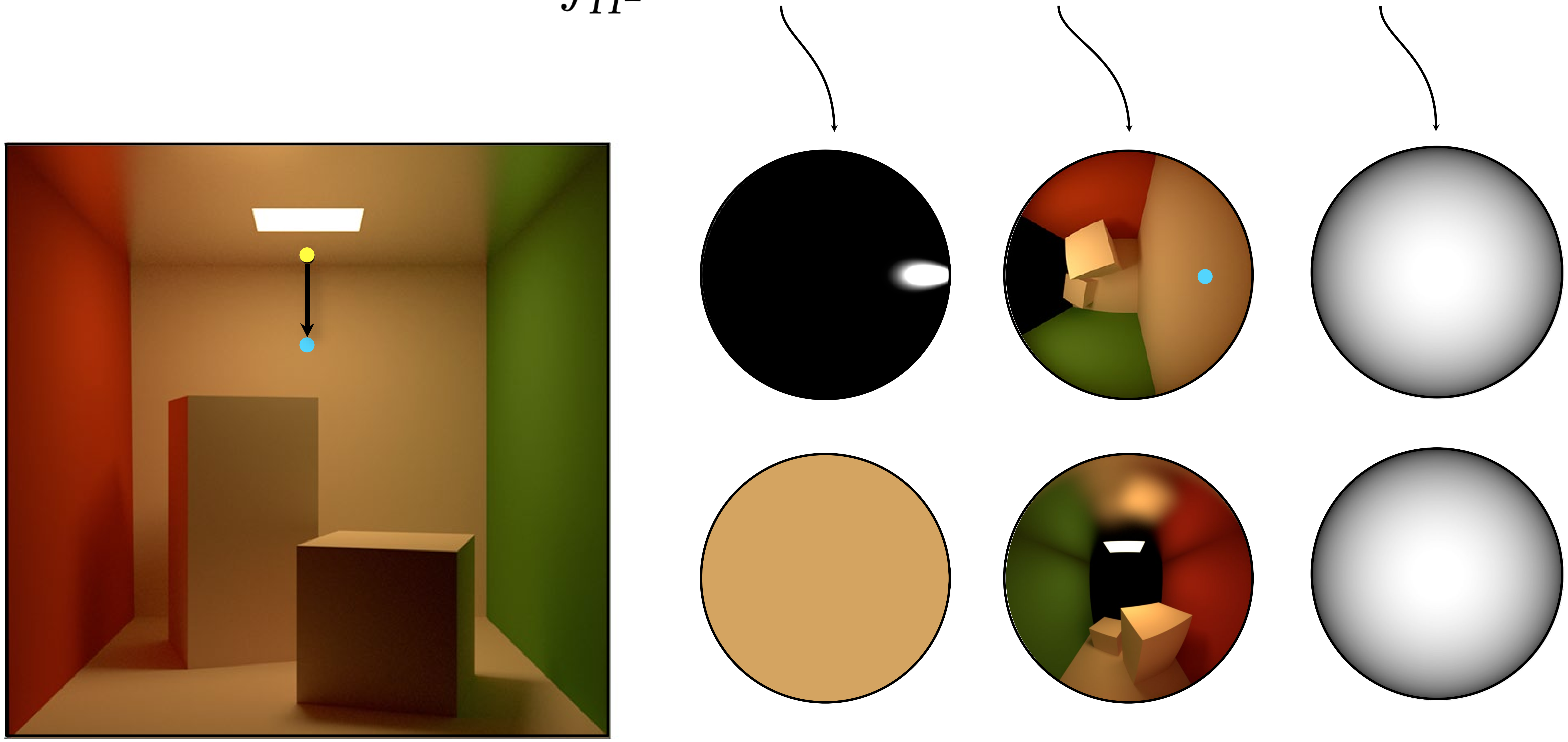
$$L(\mathbf{x},\vec{\omega}) = L_e(\mathbf{x},\vec{\omega}) + \int_{H^2} f_r(\mathbf{x},\vec{\omega}',\vec{\omega})L(r(\mathbf{x},\vec{\omega}'),-\vec{\omega}')\cos\theta'\mathrm{d}\vec{\omega}'$$





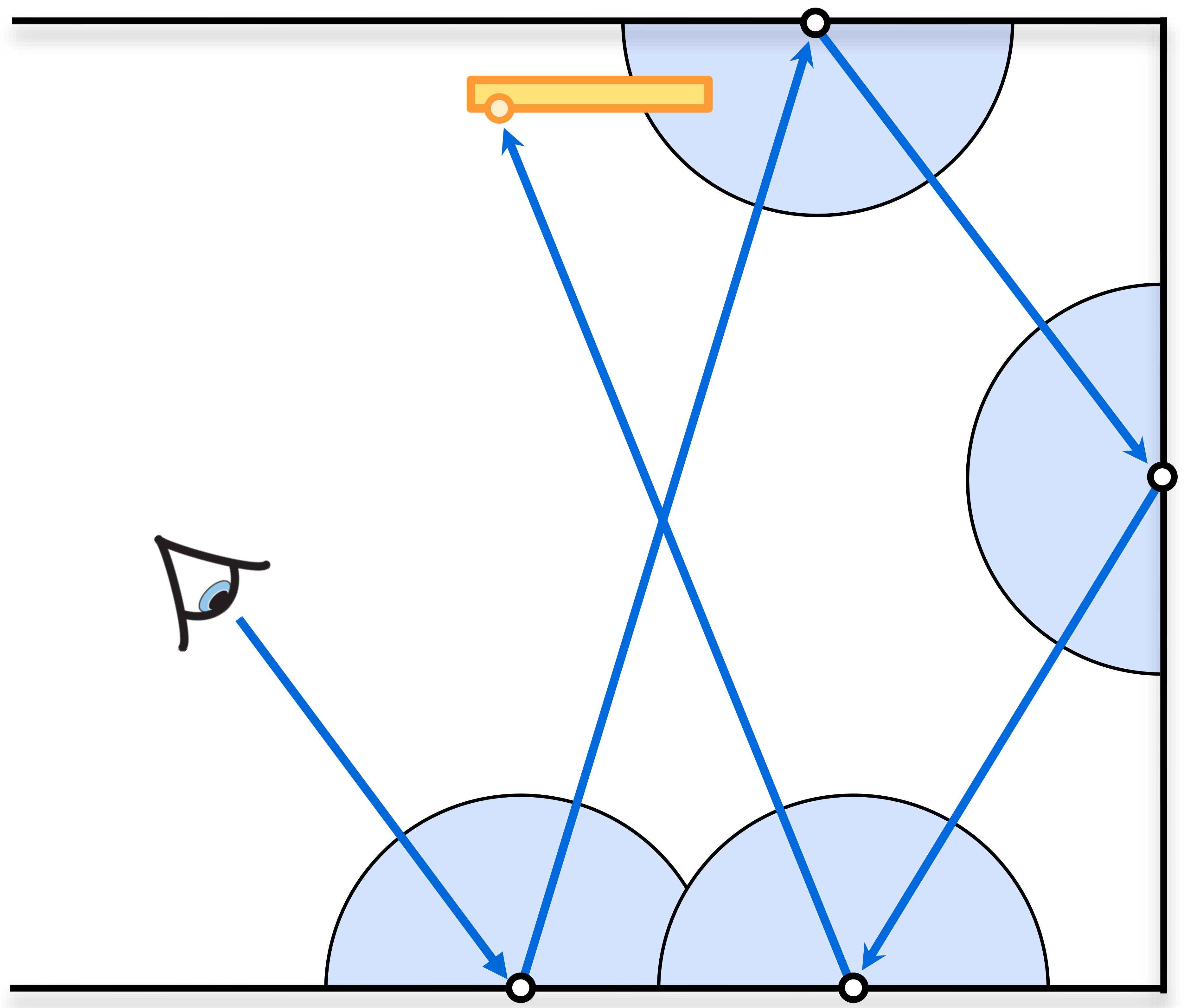
Rendering Equation

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



Path Tracing

Path Tracing



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

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

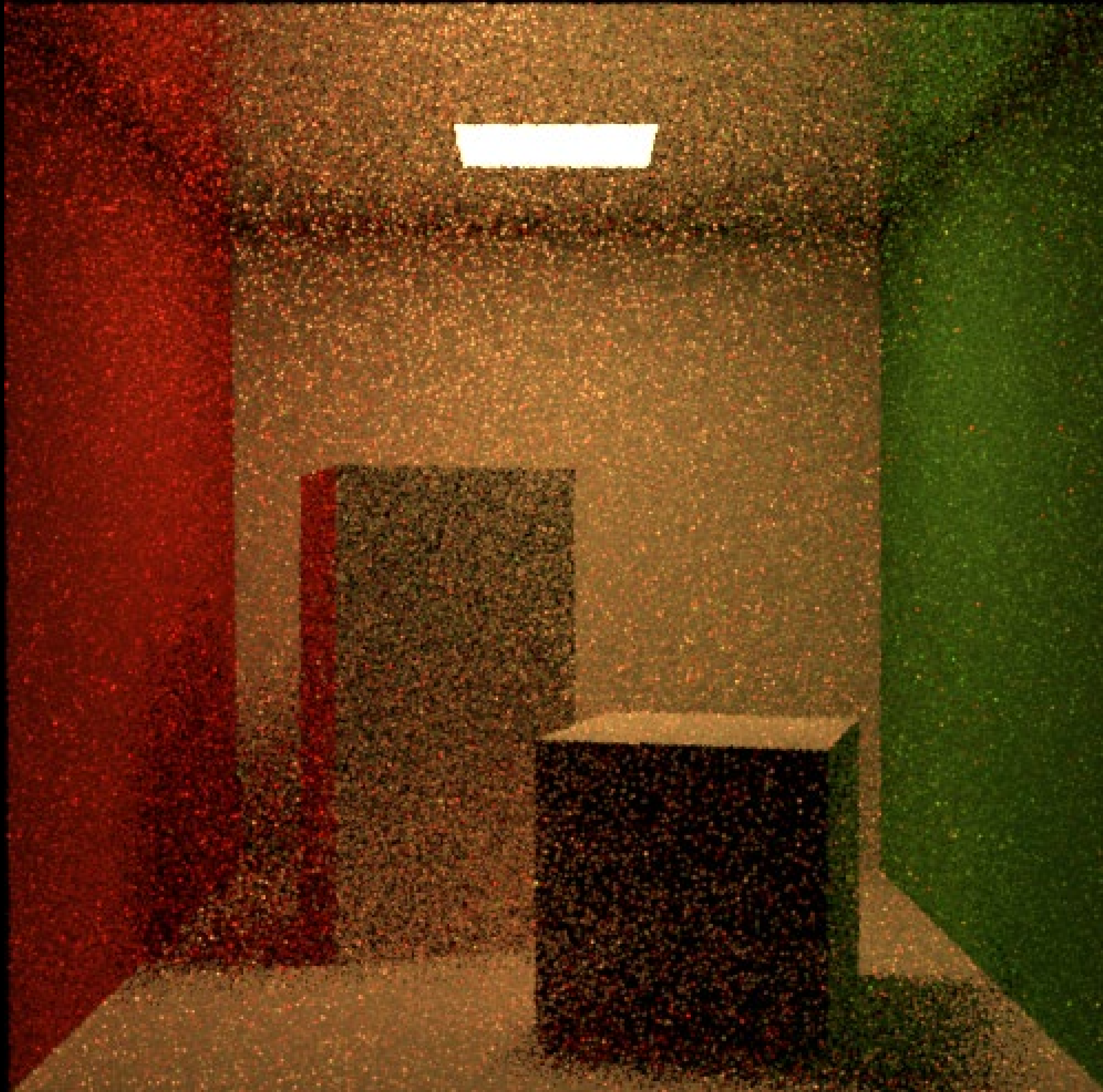
Path Tracing Algorithm

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

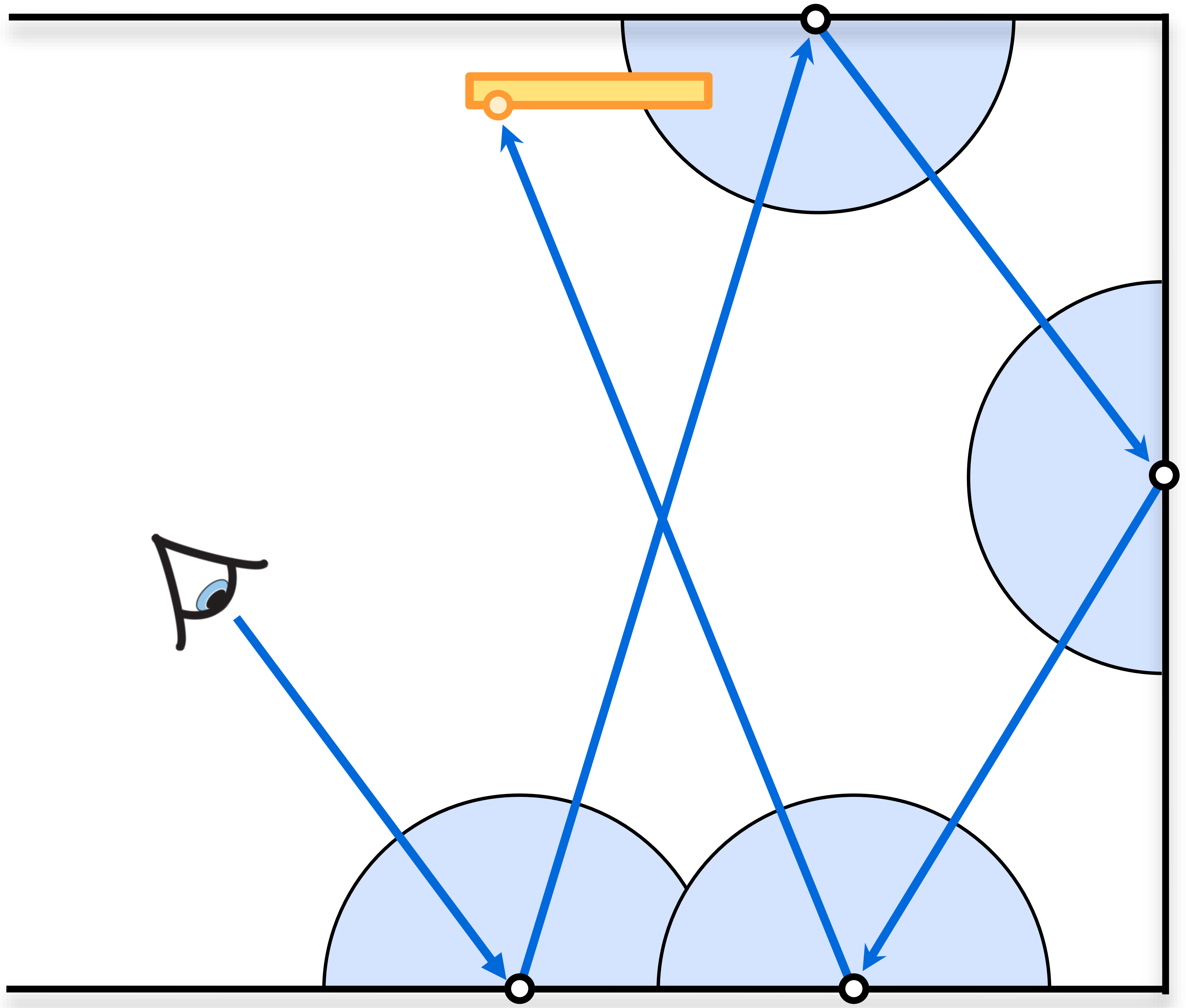
```
Color color(Point x, Direction ω, int moreBounces):

    if not moreBounces:
        return Le(x,-ω)

    // sample recursive integral
    ω' = sample from BRDF
    return Le(x,-ω) + BRDF * color(trace(x, ω'), moreBounces-1) * dot(n, ω') / pdf(ω')
```

Path Tracing with Shadow Rays

1 path/pixel



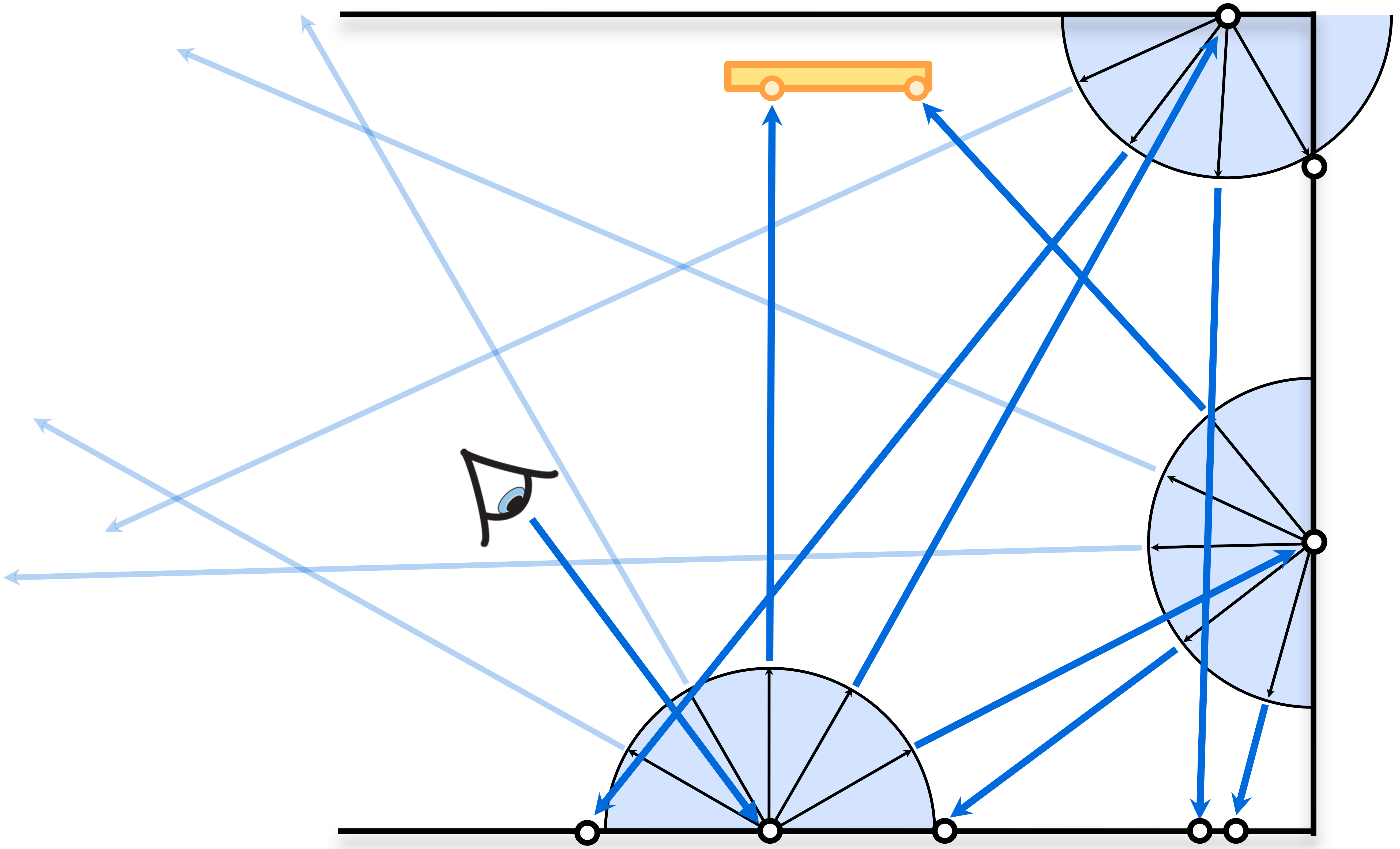
Path Tracing



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

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

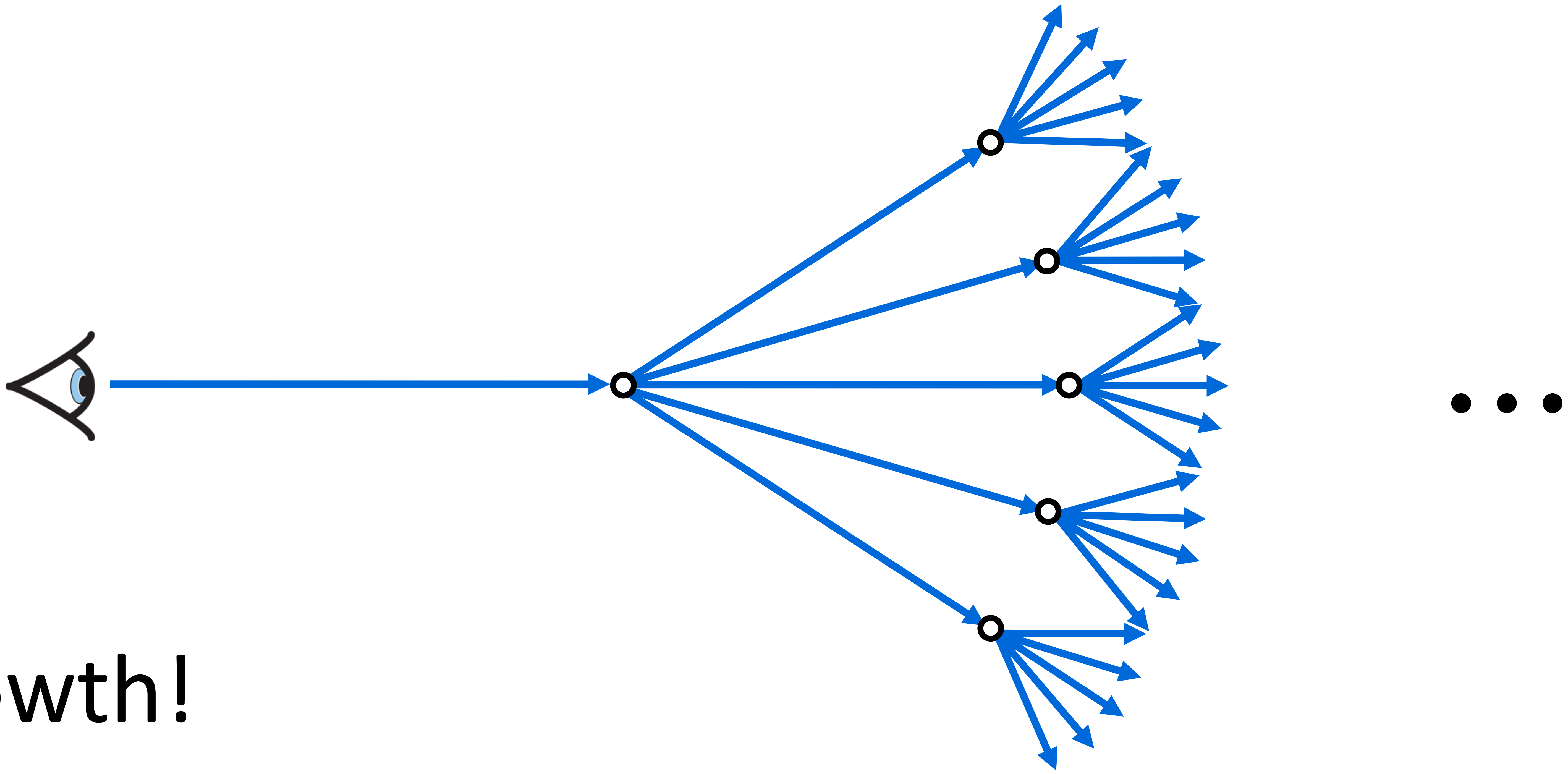
Improving quality: the wrong way



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

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

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

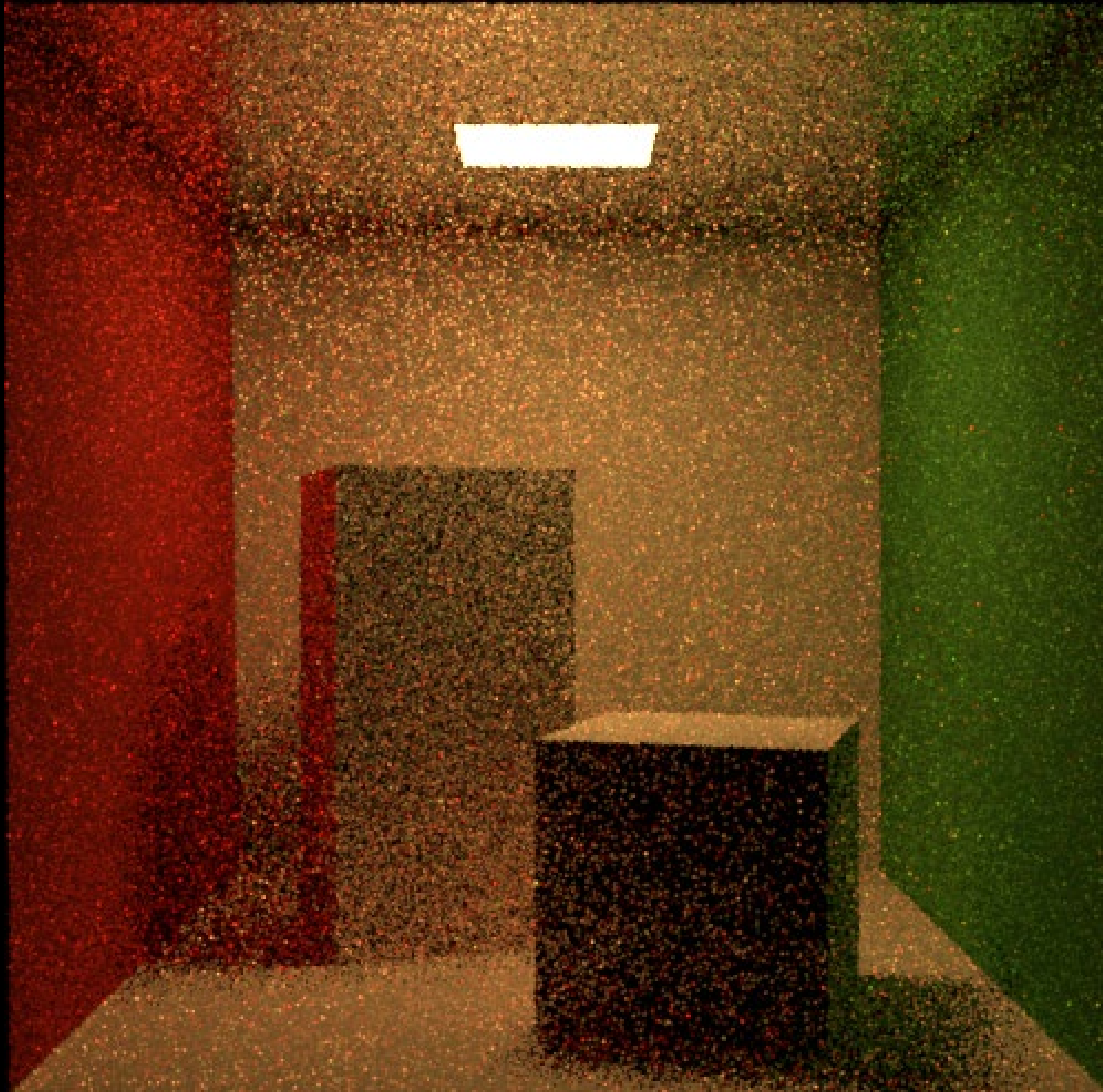
- avoid exponential growth: make sure not to branch!

Each ray will start a new **path**

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

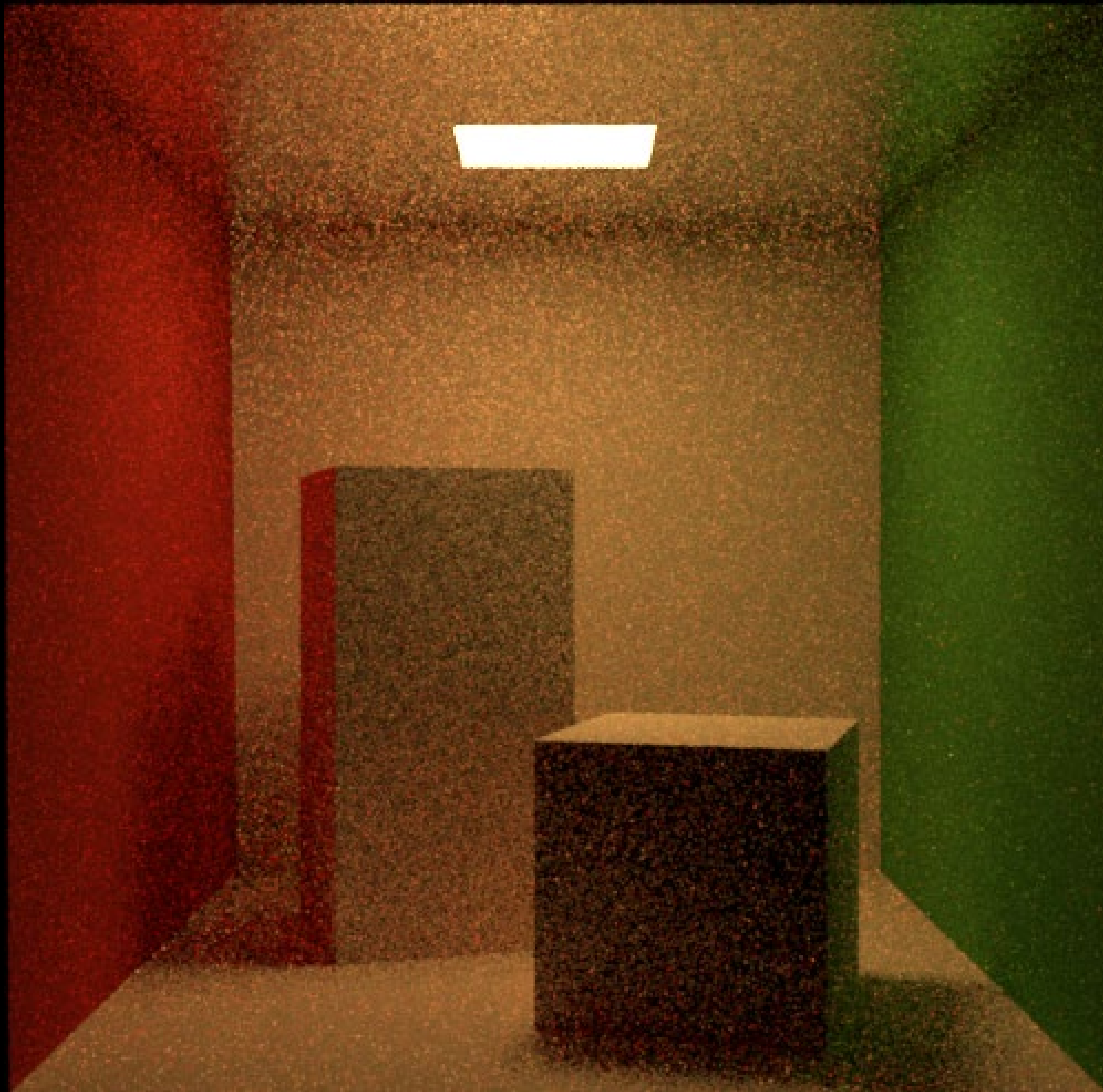
Path Tracing with Shadow Rays

1 path/pixel



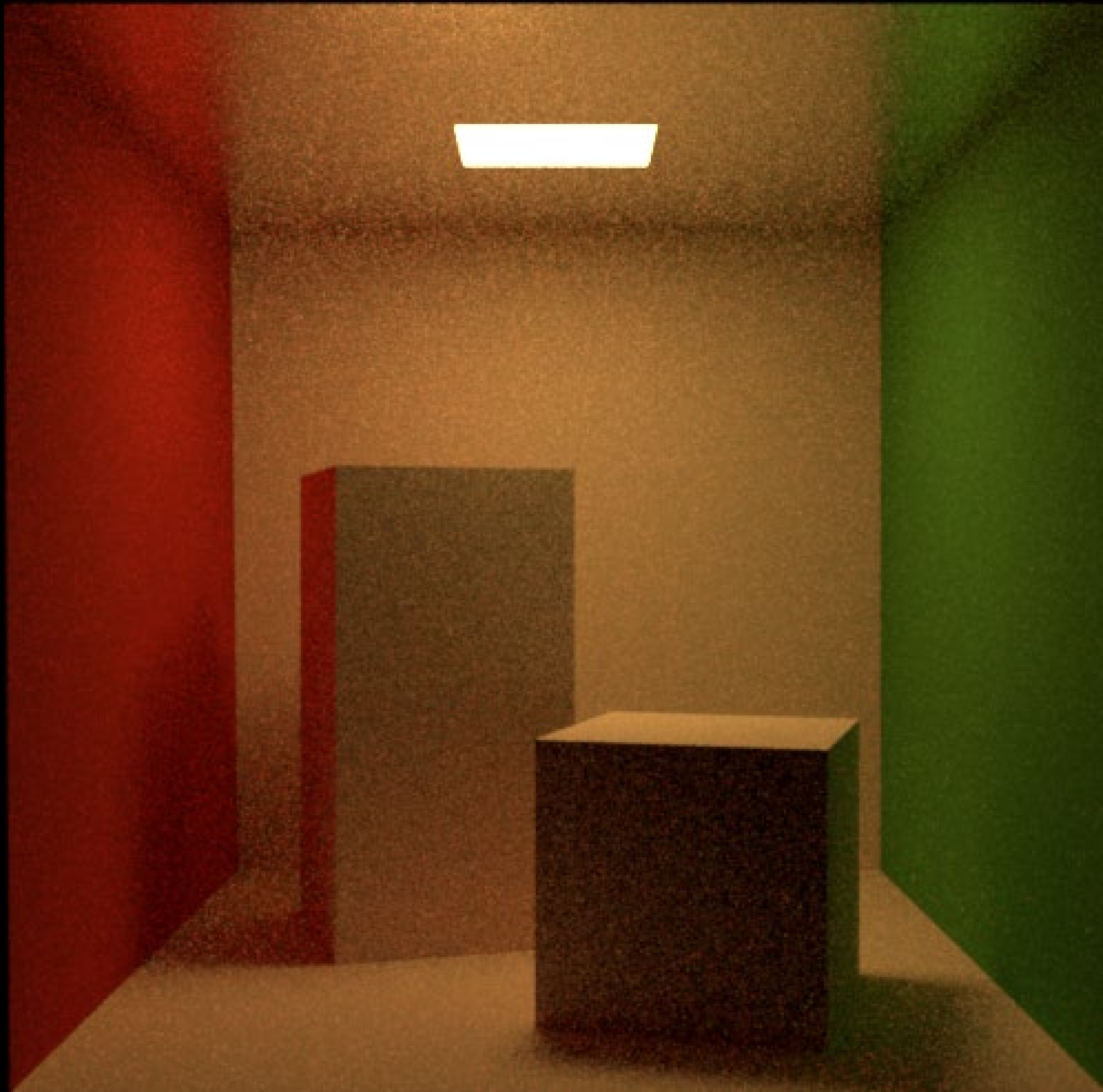
Path Tracing with Shadow Rays

4 paths/pixel



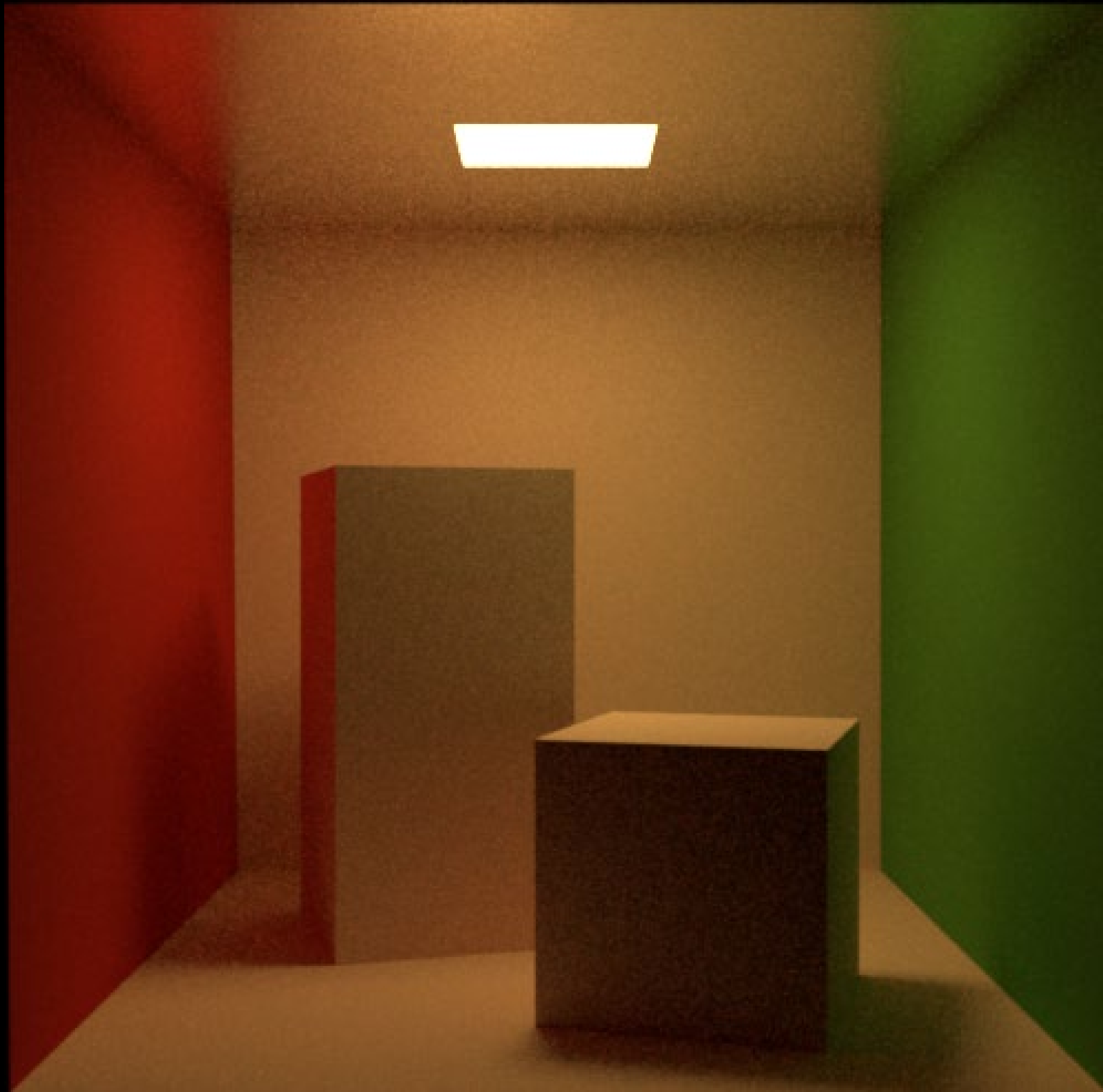
Path Tracing with Shadow Rays

16 paths/pixel



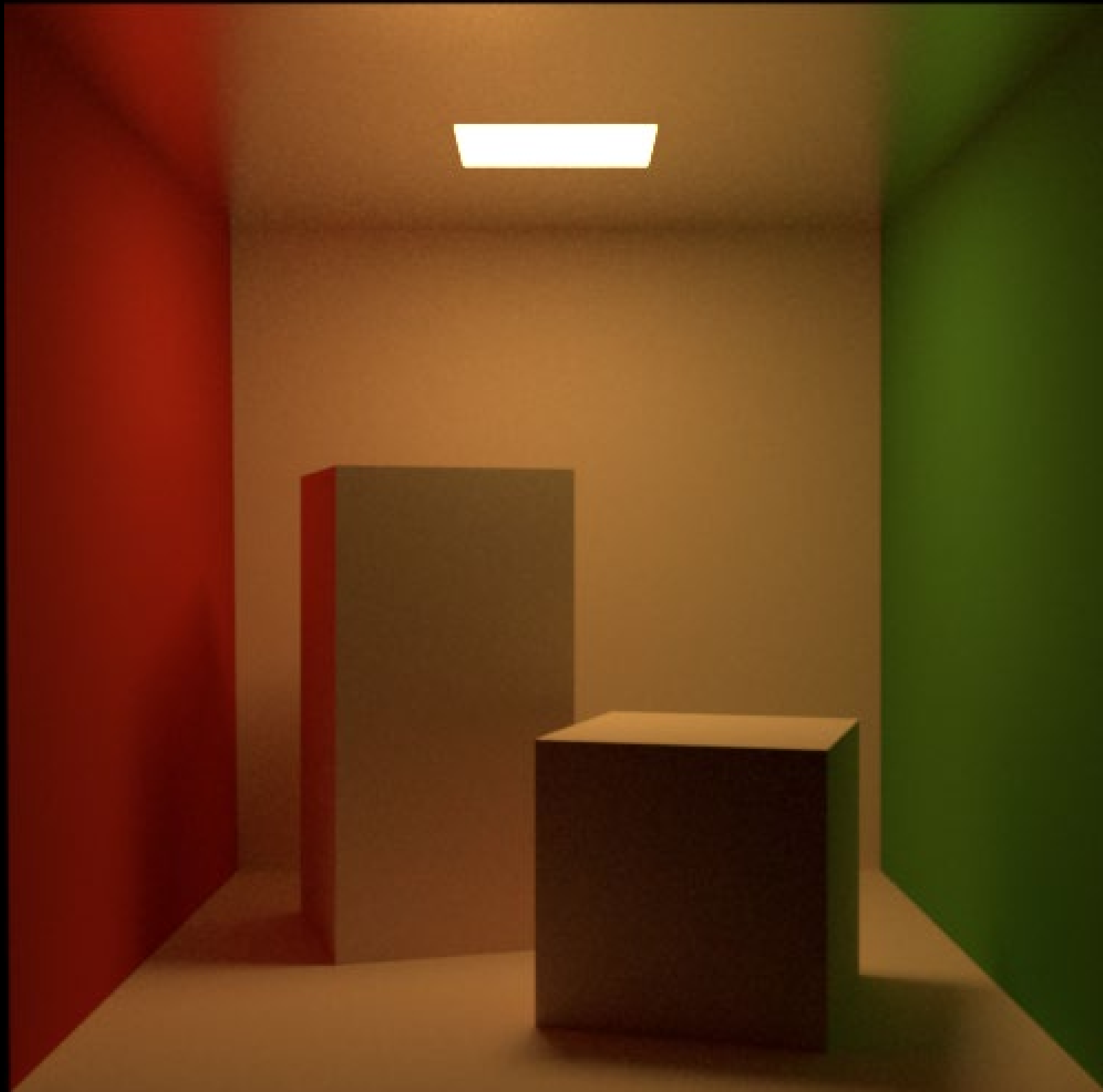
Path Tracing with Shadow Rays

64 paths/pixel



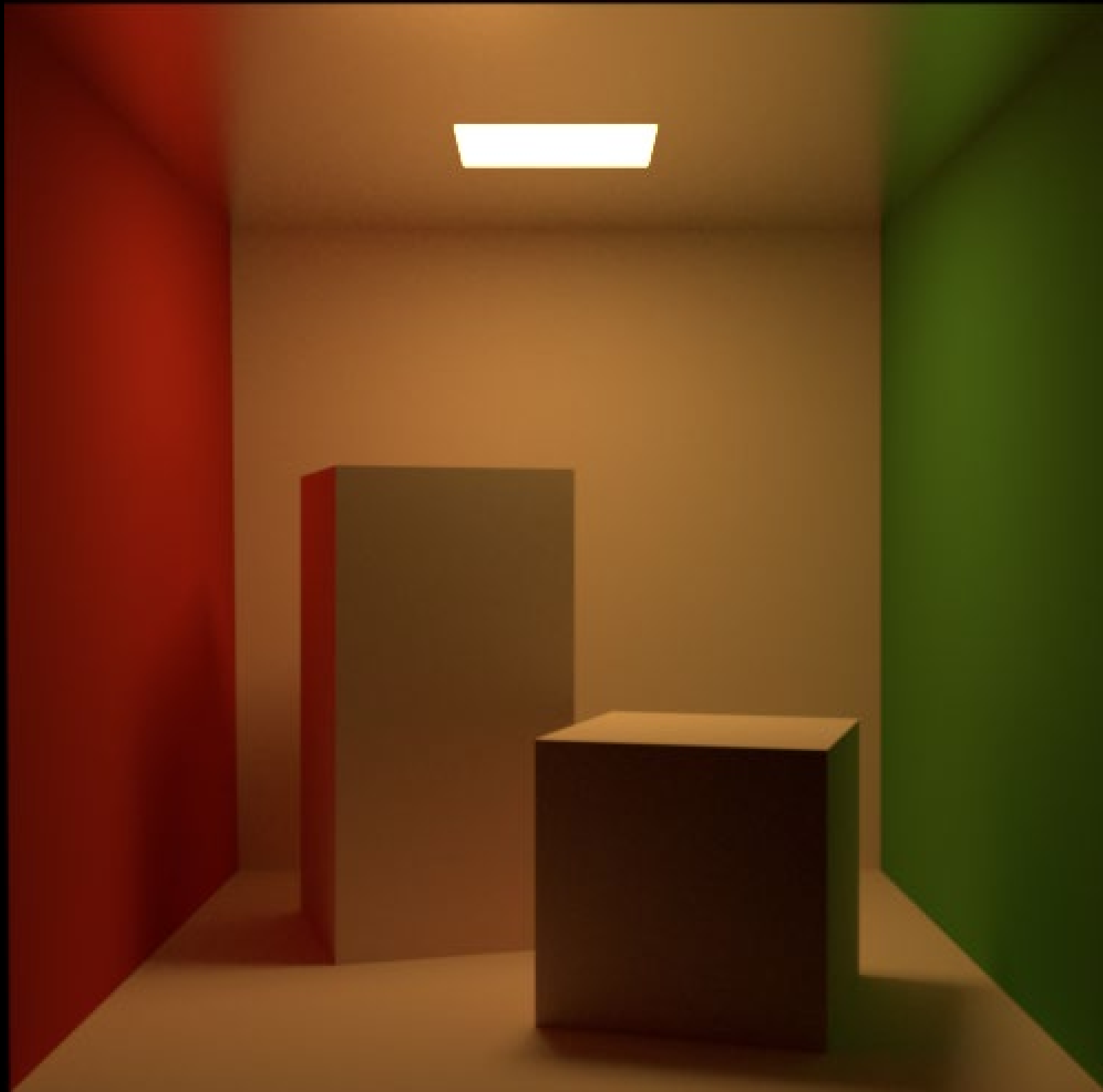
Path Tracing with Shadow Rays

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:

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

Unbiased: same expected value as original estimator:

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

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

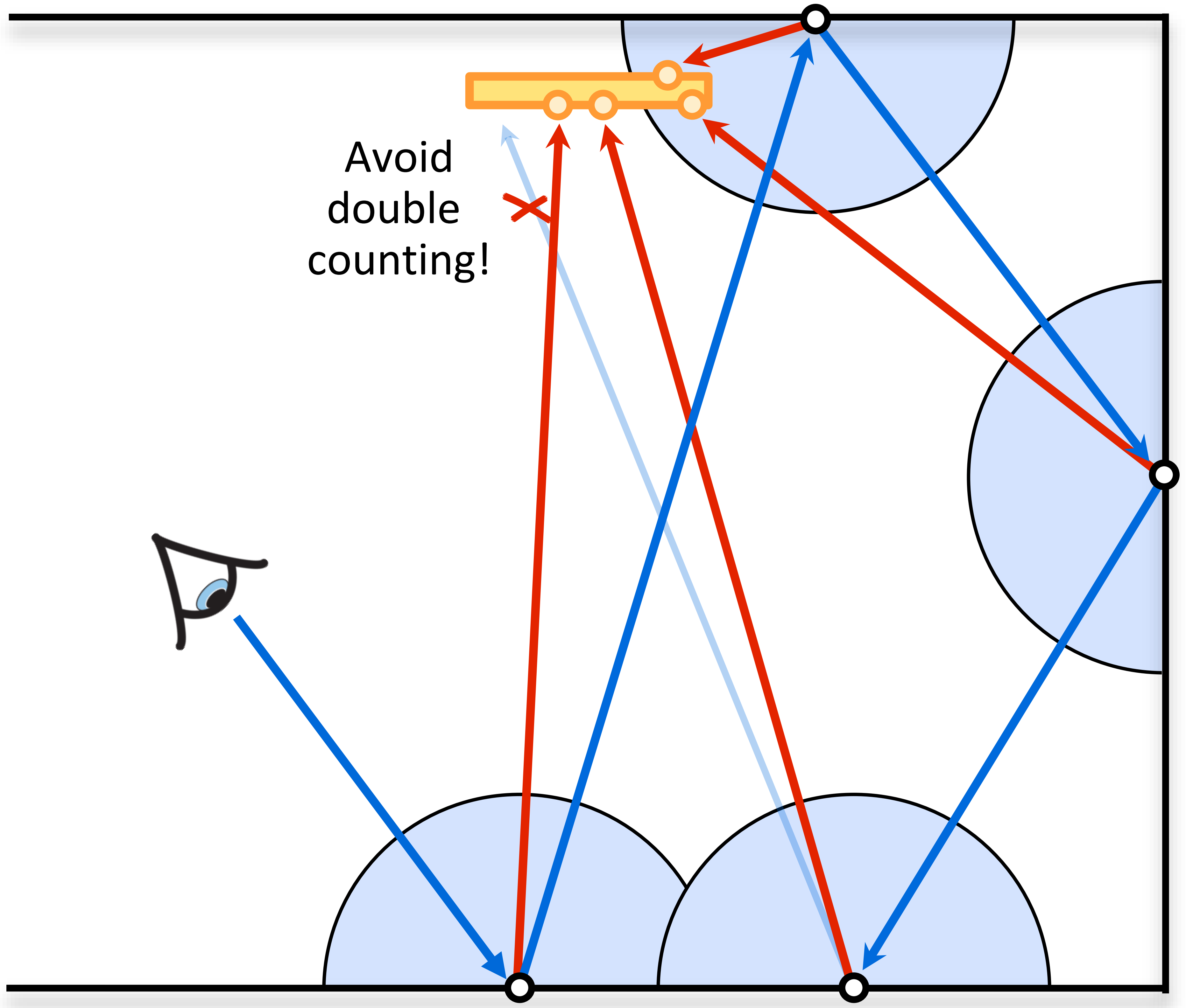
Direct illumination: sometimes better estimated by sampling emissive surfaces

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 <u>*not to double-count*</u>!

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

Path Tracing with NEE



$$L(\mathbf{x},\vec{\omega}) = L_\mathrm{e} + \int_{A_\mathrm{e}} \cdots L_\mathrm{e}(\mathbf{x} \leftarrow \mathbf{x}') \cdots \mathrm{d}A_\mathrm{e}(\mathbf{x}') + \int_{H^2 \setminus A_\mathrm{e}} \cdots L(\mathbf{x},\vec{\omega}') \cdots \mathrm{d}\vec{\omega}'$$

Path Tracing Algorithm with NEE

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

```
Color color(Point x, Direction ω, int moreBounces):

    if not moreBounces:
        return Le;

    // next-event estimation: compute Ldir by sampling the light
    ω1 = sample from light
    Ldir = BRDF * color(trace(x, ω1), 0) * dot(n, ω1) / pdf(ω1)

    // compute Lind by sampling the BSDF
    ω2 = sample from BSDF;
    Lind = BSDF * color(trace(x, ω2), moreBounces-1) * dot(n, ω2) / pdf(ω2)

    return Le + Ldir + Lind
```

double counting!

Path Tracing Algorithm with NEE

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

```
Color color(Point x, Direction ω, int moreBounces, bool includeLe):

    Le = includeLe ? Le(x,-ω) : black

    if not moreBounces:
        return Le

    // next-event estimation: compute Ldir by sampling the light
    ω1 = sample from light
    Ldir = BRDF * color(trace(x, ω1), 0, true) * dot(n, ω1) / pdf(ω1)

    // compute Lind by sampling the BSDF
    ω2 = sample from BSDF
    Lind = BSDF * color(trace(x, ω2), moreBounces-1, false) * dot(n, ω2) / pdf(ω2)

    return Le + Ldir + Lind
```

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 x, Direction ω, int moreBounces):

    if not moreBounces:
        return Le(x,-ω)

    // sample recursive integral
    ω' = sample from ~~BRDF~~
    return Le(x,-ω) + BRDF * color(trace(x, ω'), moreBounces-1) * dot(n, ω') / pdf(ω')
```

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 x, Direction ω, int moreBounces):

    if not moreBounces:
        return Le(x,-ω)

    // sample recursive integral
    ω' = sample from mixture PDF
    return Le(x,-ω) + BRDF * color(trace(x, ω'), moreBounces-1) * dot(n, ω') / pdf(ω')
```

Path Tracing Algorithm with NEE

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

    get scene intersection x, and normal n

    Le = includeLe ? Le(x,-ω) : black

    if not moreBounces:
        return Le

    // next-event estimation: compute Ldir by sampling the light
    ω1 = sample from light
    Ldir = BRDF * trace(x, ω1, 0, true) * dot(n, ω1) / pdf(ω1)

    // compute Lind by sampling the BSDF
    ω2 = sample from BSDF
    Lind = BSDF * trace(x, ω2, moreBounces-1, false) * dot(n, ω2) / pdf(ω2)

    return Le + Ldir + Lind
```

Path Tracing Algorithm with NEE+MIS

```
color trace(Point x, Direction ω, int moreBounces, float Leweight):

    get scene intersection x, and normal n

    Le = Leweight * Le(x,-ω)

    if not moreBounces:
        return Le

    // next-event estimation: compute Ldir by sampling the light
    ω1 = sample from light
    Ldir = BRDF * trace(x, ω1, 0, mis-weight1) * dot(n, ω1) / pdf(ω1)

    // compute Lind by sampling the BSDF
    ω2 = sample from BSDF
    Lind = BSDF * trace(x, ω2, moreBounces-1, mis-weight2) * dot(n, ω2) / pdf(ω2)

    return Le + Ldir + Lind
```

Path Tracing on 99 Lines of C++

```cpp
#include <math.h>   // smallpt, a Path Tracer by Kevin Beason, 2008
#include <stdlib.h> // Make : g++ -O3 -fopenmp smallpt.cpp -o smallpt
#include <stdio.h>  //        Remove "-fopenmp" for g++ version < 4.2
struct Vec {        // Usage: time ./smallpt 5000 && xv image.ppm
  double x, y, z;                  // position, also color (r,g,b)
  Vec(double x_=0, double y_=0, double z_=0){ x=x_; y=y_; z=z_; }
  Vec operator+(const Vec &b) const { return Vec(x+b.x,y+b.y,z+b.z); }
  Vec operator-(const Vec &b) const { return Vec(x-b.x,y-b.y,z-b.z); }
  Vec operator*(double b) const { return Vec(x*b,y*b,z*b); }
  Vec mult(const Vec &b) const { return Vec(x*b.x,y*b.y,z*b.z); }
  Vec& norm(){ return *this = *this * (1/sqrt(x*x+y*y+z*z)); }
  double dot(const Vec &b) const { return x*b.x+y*b.y+z*b.z; } // cross:
  Vec operator%(Vec&b){return Vec(y*b.z-z*b.y,z*b.x-x*b.z,x*b.y-y*b.x);}
};
struct Ray { Vec o, d; Ray(Vec o_, Vec d_) : o(o_), d(d_) {} };
enum Refl_t { DIFF, SPEC, REFR };  // material types, used in radiance()
struct Sphere {
  double rad;       // radius
  Vec p, e, c;      // position, emission, color
  Refl_t refl;      // reflection type (DIFFuse, SPECular, REFRactive)
  Sphere(double rad_, Vec p_, Vec e_, Vec c_, Refl_t refl_):
    rad(rad_), p(p_), e(e_), c(c_), refl(refl_) {}
  double intersect(const Ray &r) const { // returns distance, 0 if nohit
    Vec op = p-r.o; // Solve t^2*d.d + 2*t*(o-p).d + (o-p).(o-p)-R^2 = 0
    double t, eps=1e-4, b=op.dot(r.d), det=b*b-op.dot(op)+rad*rad;
    if (det<0) return 0; else det=sqrt(det);
    return (t=b-det)>eps ? t : ((t=b+det)>eps ? t : 0);
  }
};
Sphere spheres[] = {//Scene: radius, position, emission, color, material
  Sphere(1e5, Vec( 1e5+1,40.8,81.6), Vec(),Vec(.75,.25,.25),DIFF),//Left
  Sphere(1e5, Vec(-1e5+99,40.8,81.6),Vec(),Vec(.25,.25,.75),DIFF),//Rght
  Sphere(1e5, Vec(50,40.8, 1e5),     Vec(),Vec(.75,.75,.75),DIFF),//Back
  Sphere(1e5, Vec(50,40.8,-1e5+170), Vec(),Vec(),           DIFF),//Frnt
  Sphere(1e5, Vec(50, 1e5, 81.6),    Vec(),Vec(.75,.75,.75),DIFF),//Botm
  Sphere(1e5, Vec(50,-1e5+81.6,81.6),Vec(),Vec(.75,.75,.75),DIFF),//Top
  Sphere(16.5,Vec(27,16.5,47),       Vec(),Vec(1,1,1)*.999, SPEC),//Mirr
  Sphere(16.5,Vec(73,16.5,78),       Vec(),Vec(1,1,1)*.999, REFR),//Glas
  Sphere(600, Vec(50,681.6-.27,81.6),Vec(12,12,12),  Vec(), DIFF) //Lite
};
inline double clamp(double x){ return x<0 ? 0 : x>1 ? 1 : x; }
inline int toInt(double x){ return int(pow(clamp(x),1/2.2)*255+.5); }
inline bool intersect(const Ray &r, double &t, int &id){
  double n=sizeof(spheres)/sizeof(Sphere), d, inf=t=1e20;
  for(int i=int(n);i--;) if((d=spheres[i].intersect(r))&&d<t){t=d;id=i;}
  return t<inf;
}
Vec radiance(const Ray &r, int depth, unsigned short *Xi){
  double t;                               // distance to intersection
  int id=0;                               // id of intersected object
  if (!intersect(r, t, id)) return Vec(); // if miss, return black
  const Sphere &obj = spheres[id];        // the hit object
  Vec x=r.o+r.d*t, n=(x-obj.p).norm(), nl=n.dot(r.d)<0?n:n*-1, f=obj.c;
  double p = f.x>f.y && f.x>f.z ? f.x : f.y>f.z ? f.y : f.z; // max refl
  if (++depth>5) if (erand48(Xi)<p) f=f*(1/p); else return obj.e; //R.R.
  if (obj.refl == DIFF){                  // Ideal DIFFUSE reflection
    double r1=2*M_PI*erand48(Xi), r2=erand48(Xi), r2s=sqrt(r2);
    Vec w=nl, u=((fabs(w.x)>.1?Vec(0,1):Vec(1))%w).norm(), v=w%u;
    Vec d = (u*cos(r1)*r2s + v*sin(r1)*r2s + w*sqrt(1-r2)).norm();
    return obj.e + f.mult(radiance(Ray(x,d),depth,Xi));
  } else if (obj.refl == SPEC)            // Ideal SPECULAR reflection
    return obj.e + f.mult(radiance(Ray(x,r.d-n*2*n.dot(r.d)),depth,Xi));
  Ray reflRay(x, r.d-n*2*n.dot(r.d));     // Ideal dielectric REFRACTION
  bool into = n.dot(nl)>0;                // Ray from outside going in?
  double nc=1, nt=1.5, nnt=into?nc/nt:nt/nc, ddn=r.d.dot(nl), cos2t;
  if ((cos2t=1-nnt*nnt*(1-ddn*ddn))<0)    // Total internal reflection
    return obj.e + f.mult(radiance(reflRay,depth,Xi));
  Vec tdir = (r.d*nnt - n*((into?1:-1)*(ddn*nnt+sqrt(cos2t)))).norm();
  double a=nt-nc, b=nt+nc, R0=a*a/(b*b), c = 1-(into?-ddn:tdir.dot(n));
  double Re=R0+(1-R0)*c*c*c*c*c,Tr=1-Re,P=.25+.5*Re,RP=Re/P,TP=Tr/(1-P);
  return obj.e + f.mult(depth>2 ? (erand48(Xi)<P ?   // Russian roulette
    radiance(reflRay,depth,Xi)*RP:radiance(Ray(x,tdir),depth,Xi)*TP) :
    radiance(reflRay,depth,Xi)*Re+radiance(Ray(x,tdir),depth,Xi)*Tr);
}
int main(int argc, char *argv[]){
  int w=1024, h=768, samps = argc==2 ? atoi(argv[1])/4 : 1; // # samples
  Ray cam(Vec(50,52,295.6), Vec(0,-0.042612,-1).norm()); // cam pos, dir
  Vec cx=Vec(w*.5135/h), cy=(cx%cam.d).norm()*.5135, r, *c=new Vec[w*h];
#pragma omp parallel for schedule(dynamic, 1) private(r)       // OpenMP
  for (int y=0; y<h; y++){                       // Loop over image rows
    fprintf(stderr,"\rRendering (%d spp) %5.2f%%",samps*4,100.*y/(h-1));
    for (unsigned short x=0, Xi[3]={0,0,y*y*y}; x<w; x++)   // Loop cols
      for (int sy=0, i=(h-y-1)*w+x; sy<2; sy++)     // 2x2 subpixel rows
        for (int sx=0; sx<2; sx++, r=Vec()){        // 2x2 subpixel cols
          for (int s=0; s<samps; s++){
            double r1=2*erand48(Xi), dx=r1<1 ? sqrt(r1)-1: 1-sqrt(2-r1);
            double r2=2*erand48(Xi), dy=r2<1 ? sqrt(r2)-1: 1-sqrt(2-r2);
            Vec d = cx*( ( (sx+.5 + dx)/2 + x)/w - .5) +
                    cy*( ( (sy+.5 + dy)/2 + y)/h - .5) + cam.d;
            r = r + radiance(Ray(cam.o+d*140,d.norm()),0,Xi)*(1./samps);
          } // Camera rays are pushed ^^^^^ forward to start in interior
          c[i] = c[i] + Vec(clamp(r.x),clamp(r.y),clamp(r.z))*.25;
        }
  }
  FILE *f = fopen("image.ppm", "w");         // Write image to PPM file.
  fprintf(f, "P3\n%d %d\n%d\n", w, h, 255);
  for (int i=0; i<w*h; i++)
    fprintf(f,"%d %d %d ", toInt(c[i].x), toInt(c[i].y), toInt(c[i].z));
}
```

smallpt by Kevin Beason

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

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega f_r(x, \omega_i \rightarrow \omega_o)\, L_i(x, \omega_i)\, (\omega_i \cdot n)\, d\omega_i$$

position on the stuft

direction towards the eye

light that leaves the position on the stuff and reaches the eye

light made by the stuff (sometimes because it is very hot)

light can be added said a man who sat under a tree many years ago

the answer to how much light from an interesting direction that will keep going in the direction towards the eye, after hitting stuff at the position (this is easy for mirrors, not so easy for everything else)

the light that comes from an interesting direction towards the position on the stuff

how much the light becomes less bright because the stuff leans away from the interesting direction

for lots of interesting directions inside half a ball facing up from the stuff, add up all the answers in between

this idea came from http://xkcd.com/1133/

@levork