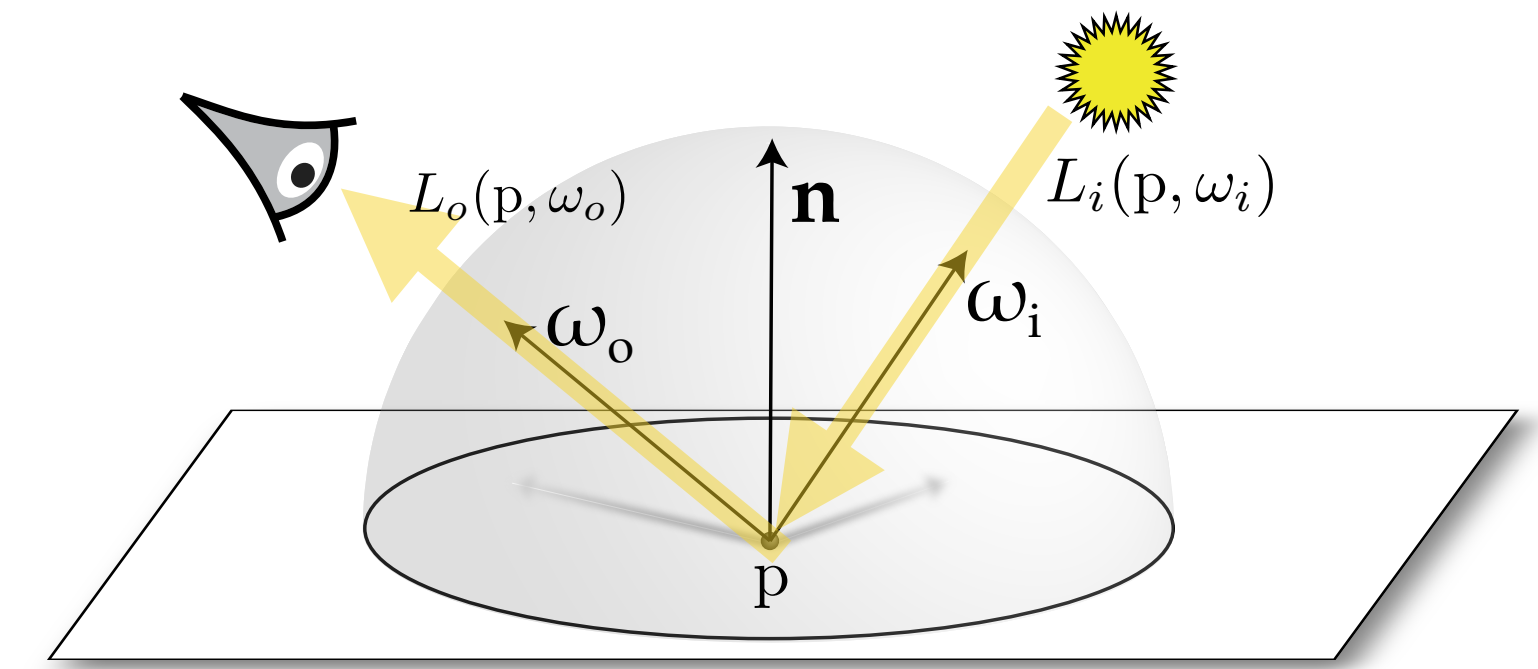


Lecture 12:

Global Illumination using Path Tracing

Computer Graphics: Rendering, Geometry, and Image Manipulation
Stanford CS248A, Winter 2026

Last time: Monte Carlo estimate for the reflection equation



Reflection equation in terms of integration over solid angles:

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

Monte Carlo estimate via sampling solid angle according to the PDF: $p(\omega)$

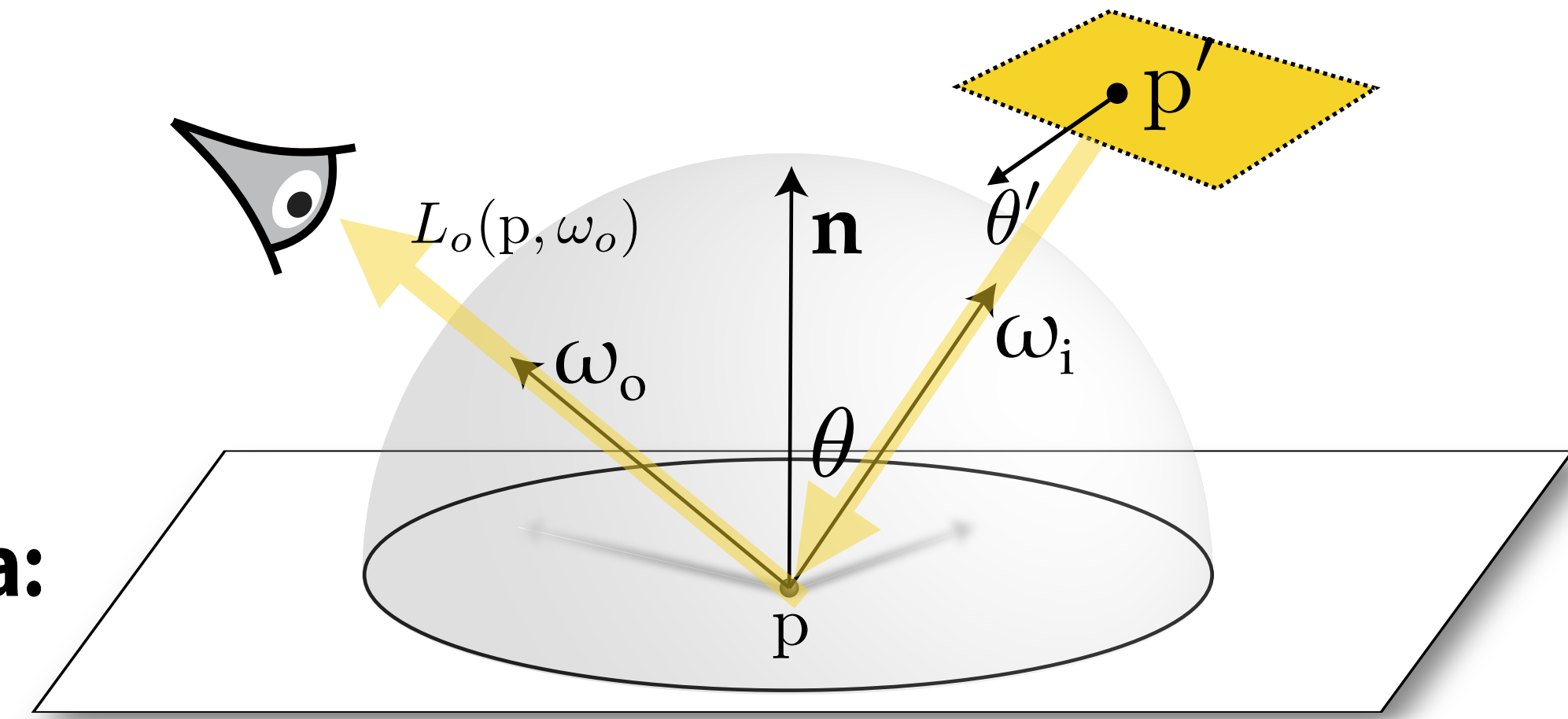
Sample: $\omega_j \sim p(\omega)$

Estimate:
$$\frac{1}{N} \sum_{j=1}^N \frac{f_r(p, \omega_j \rightarrow \omega_o) L_i(p, \omega_j) \cos \theta_j}{p(\omega_j)}$$

Last time: Monte Carlo estimate for the reflection equation

Assume area light source emitting radiance L in all directions.

Reflection equation in terms of integration over one light source area:



$$L_o(p, \omega_o) = \int_{A'} f_r(p, (p' - p) \rightarrow \omega_o) L \frac{\cos \theta \cos \theta'}{|p - p'|^2} dA$$

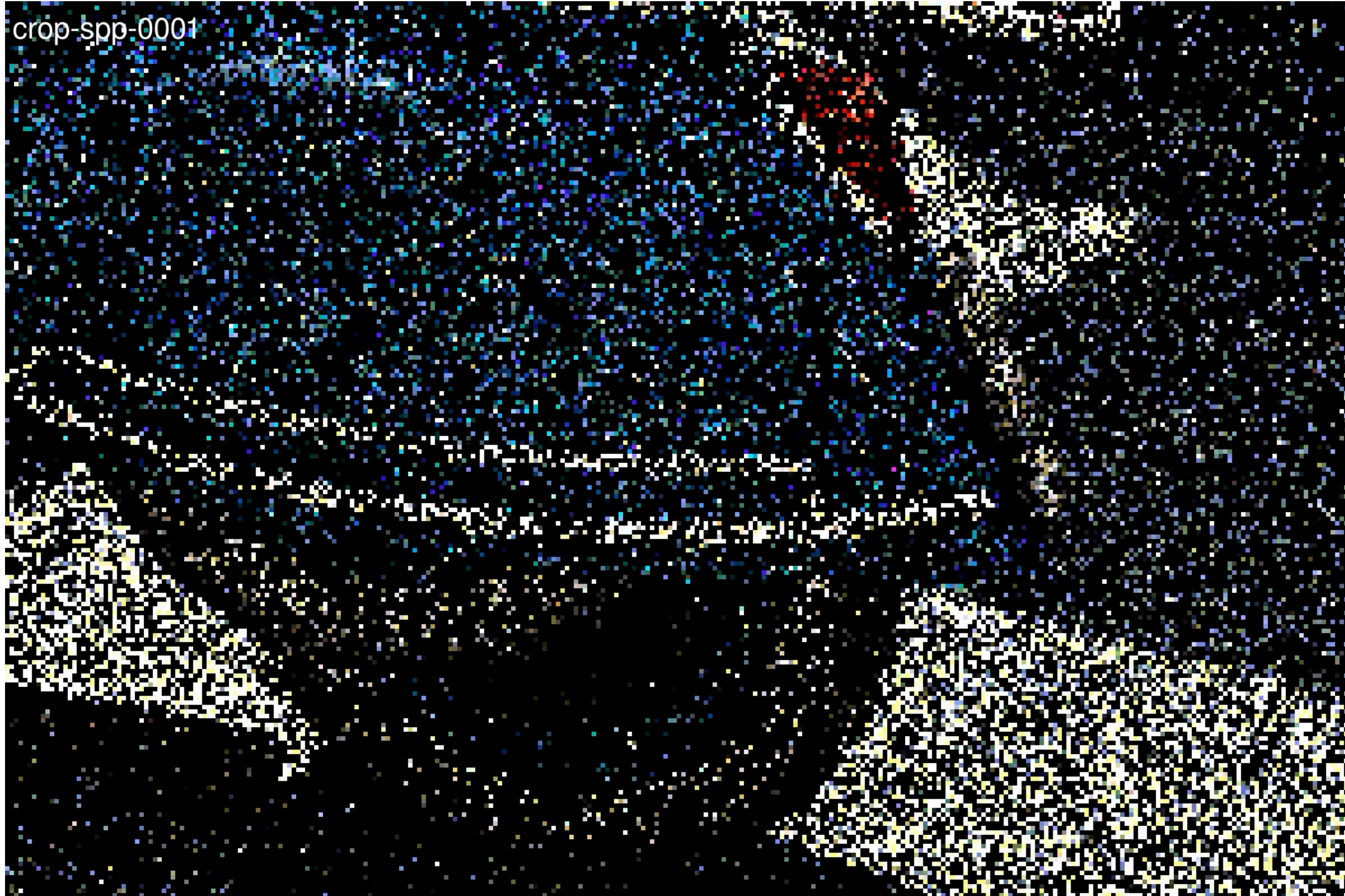
Monte Carlo estimate via sampling points on lights:

Sample:

$$p_j \sim p(p)$$

Estimate:
$$\frac{1}{N} \sum_{j=1}^N \frac{f_r(p, (p' - p) \rightarrow \omega_o) L \cos \theta \cos \theta'}{p(p_j) |p - p'|^2}$$

Review: Why does this picture look like this?



For video see: <http://pharr.org/matt/assets/bistro-spp.mp4>

Major themes from last time

- Monte Carlo: draw samples from integration domain according to $p(x)$
- Evaluate estimator at all samples
- In estimation, average of estimators is the integral we wish to compute
- Importance sampling: biasing sampling towards parts of the domain where integrand is large reduces variance in the Monte Carlo estimate

- **Consider:** $\int_{\Omega^2} f_r(p, \omega_i \rightarrow \omega_o) L_i(p, \omega_i) \cos \theta_i d\omega_i$

- **E.g., shape $p(\omega)$ to be proportional to:** $\cos \theta_i$

If no priors of lighting/BRDF known, or if lighting environment and/or BRDF are largely constant

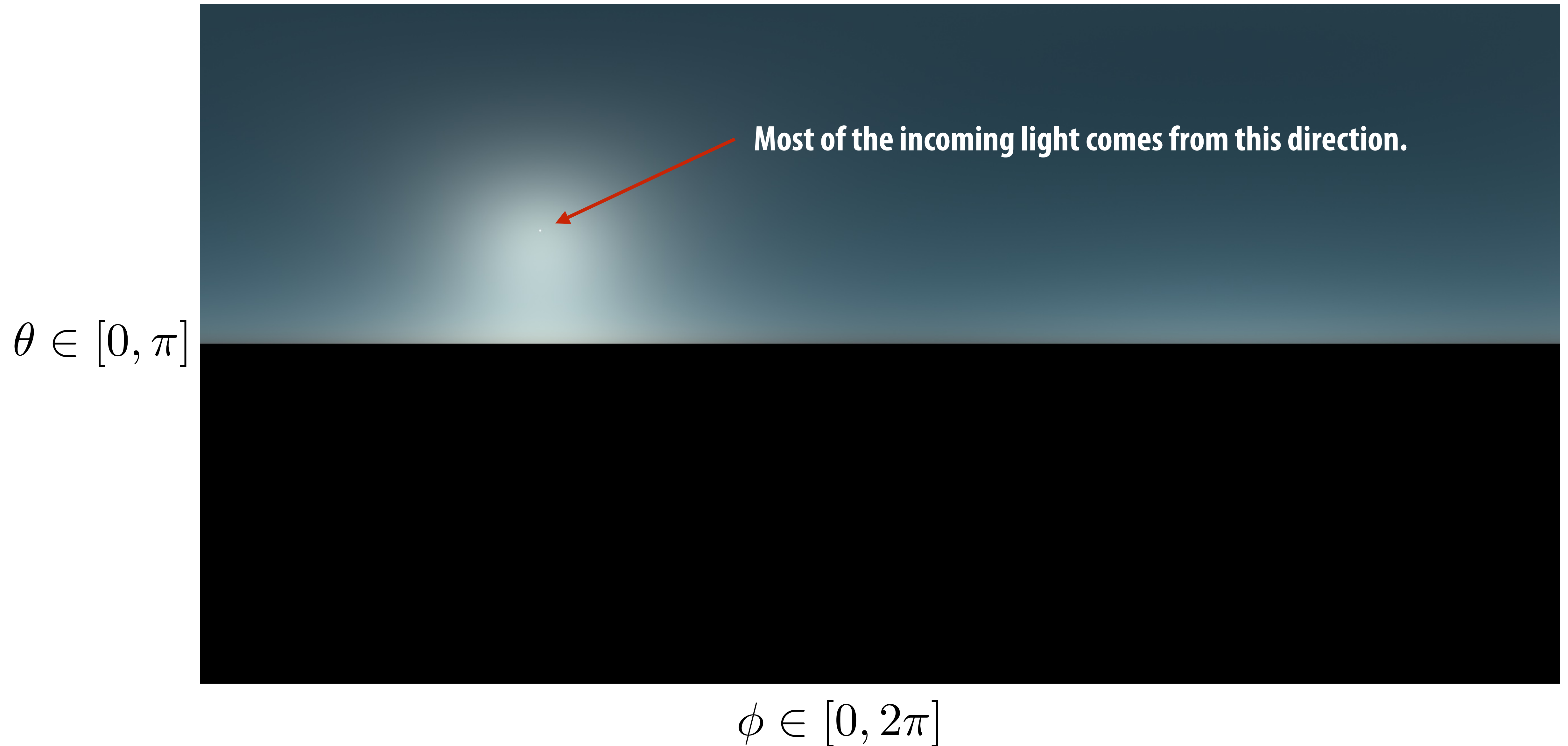
$L_i(p, \omega_i)$

If priors on the lighting are known: eg., environment light sampling, bias samples toward powerful lights in many-light setting

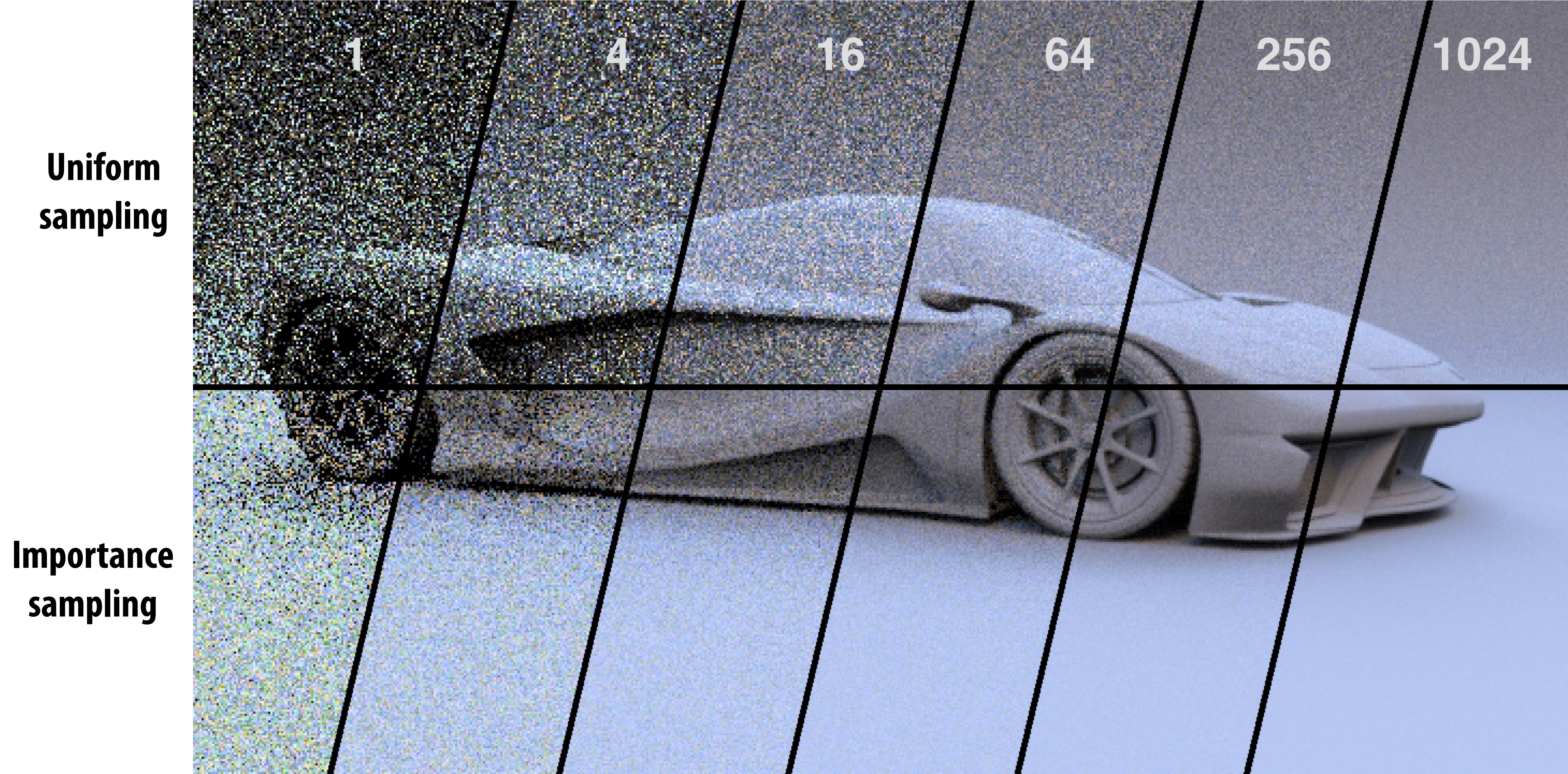
$f_r(p, \omega_i \rightarrow \omega_o)$

If BRDF is highly peaked, like a very glossy surface

Review: example of lighting environment with peaked distribution

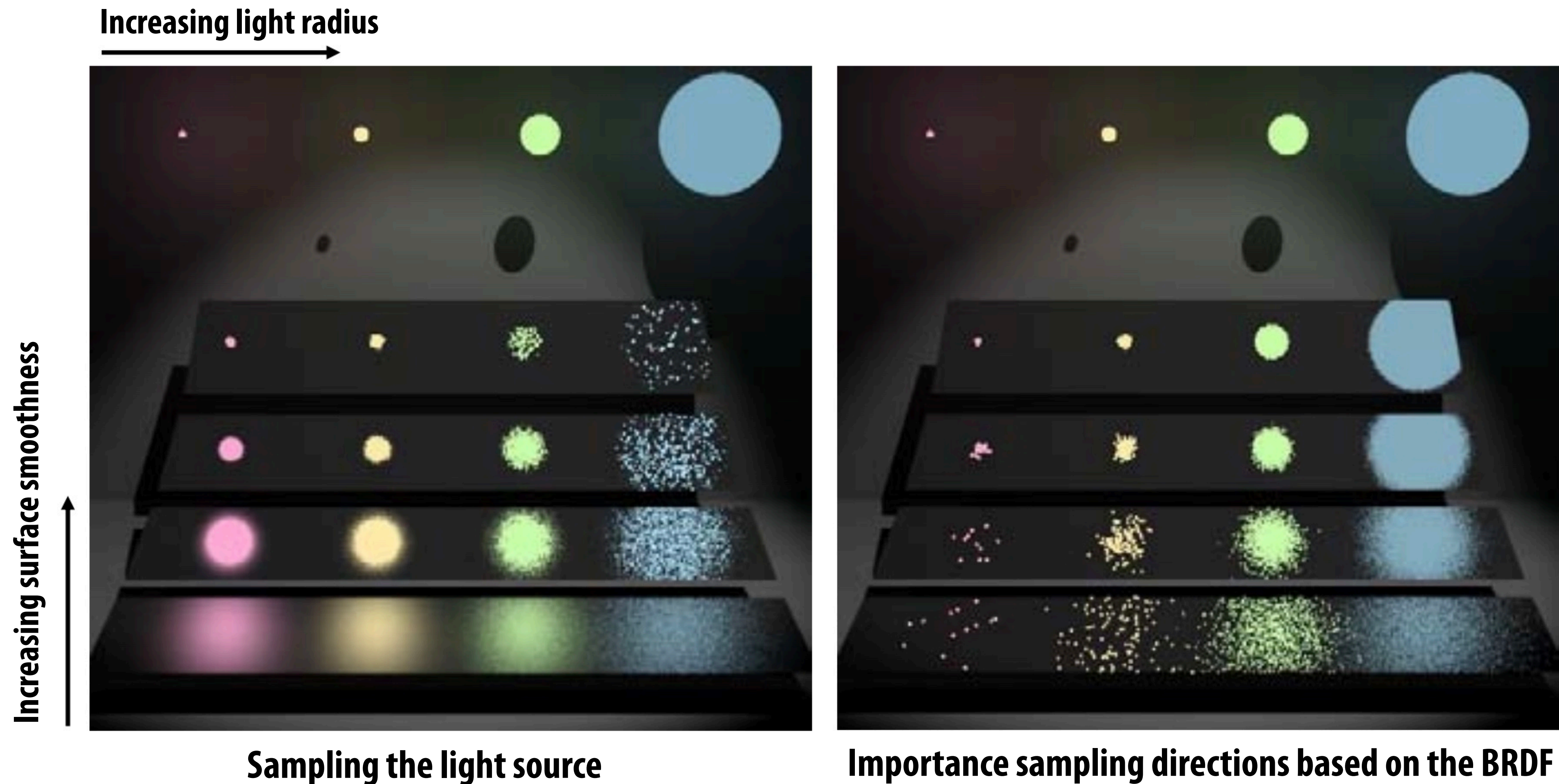


Importance sampling reduces variance in reflection estimate



Can you explain this figure?

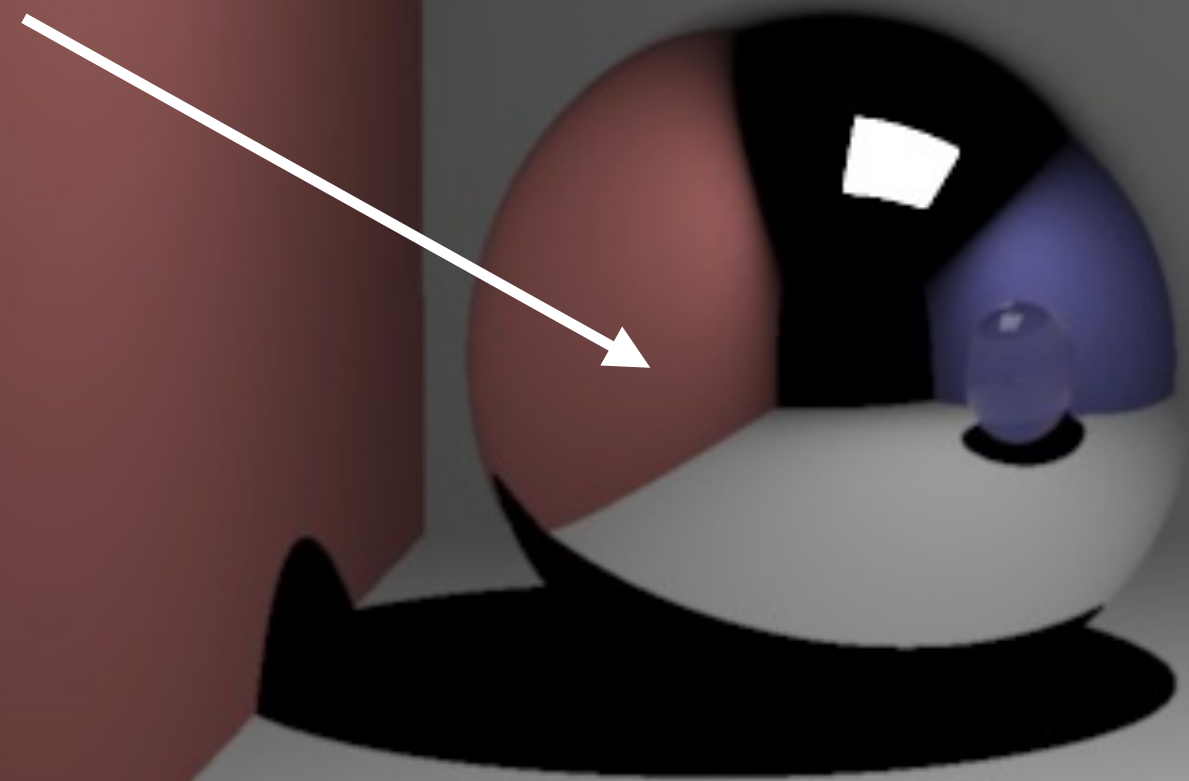
It depicts reflection of a spherical light source off surfaces of increasing smoothness (nearly a mirror in the back)



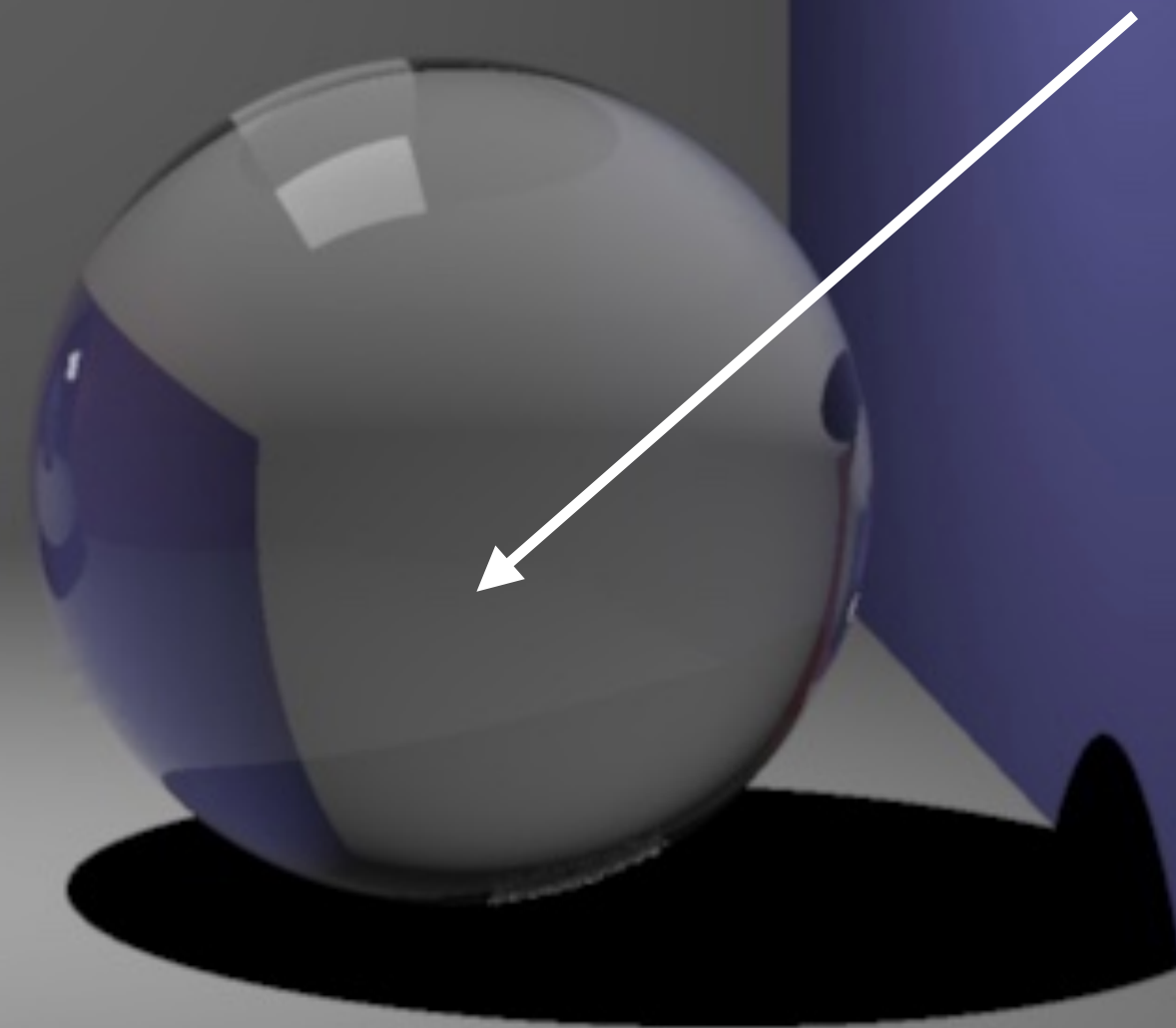
What about an image like this?

Direct illumination + reflection + transparency

Perfect mirror:



Glass



Basic recursive ray tracing for a perfect mirror

```
Spectrum TraceRay(Ray ray) {
    Intersection isect = scene->Intersect(ray);
    BSDF bsdf = isect.GetBSDF();
    Vector3f wo = -ray.d;

    if (bsdf->isMirror()) { // handle case of perfect mirror surface
        Vector3d wi = computeMirrorReflDir(isect.N, wo);
        if (ray.depth == MAX_DEPTH)
            return Spectrum(0.0);
        else
            return TraceRay(Ray(isect.P, wi, ray.depth+1));
    } else {
        // non-mirror case, compute reflectance due to unshadowed
        // area source of radiance L

        Spectrum Lo;
        for (int j=0 to NUM_LIGHT_SAMPLES) {
            // sample point on light source to get: p_j, pdf(p_j)

            // compute MC estimator...

            Lo += estimator / NUM_LIGHT_SAMPLES;
        }
    }
}
```

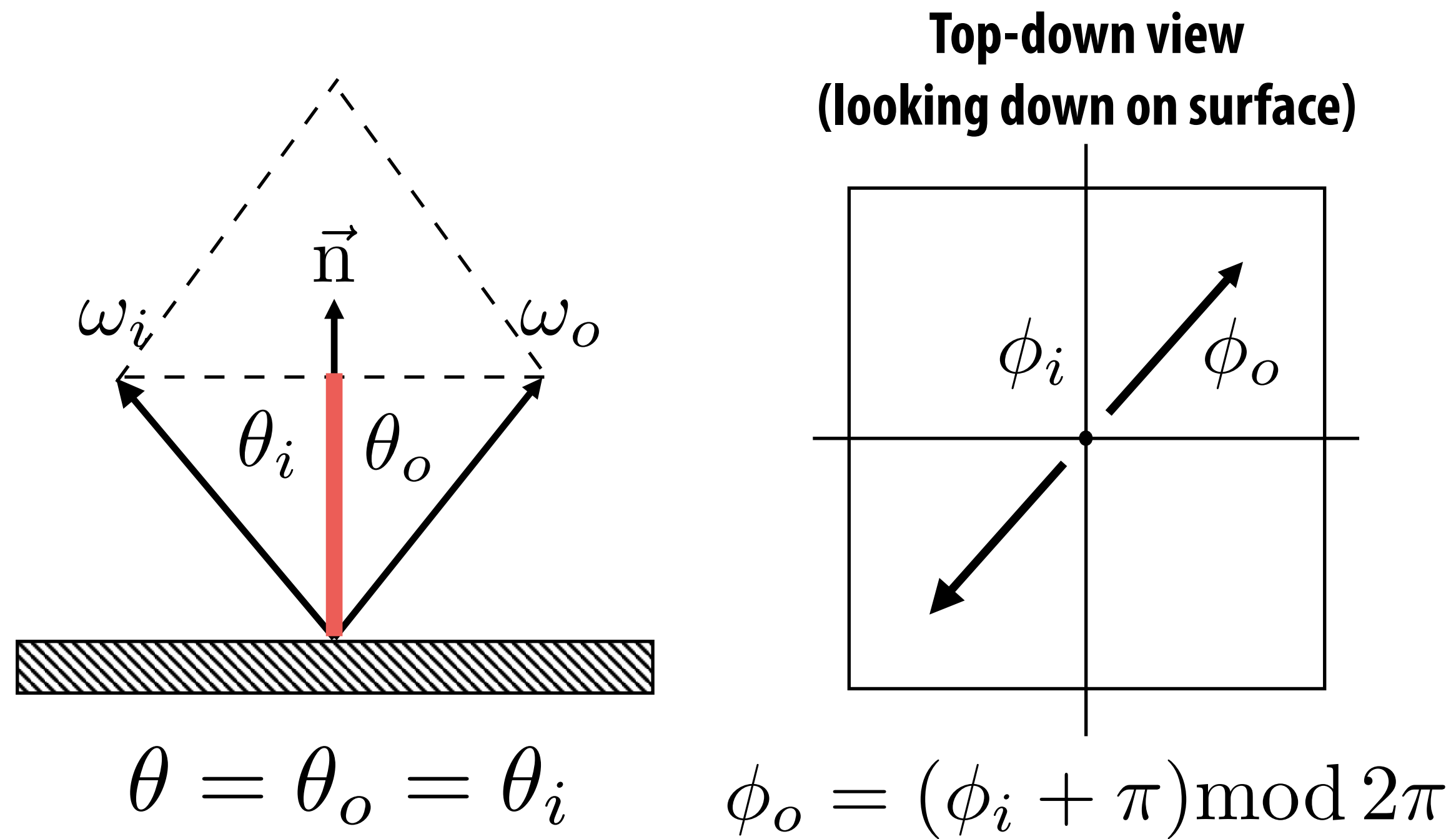
Specular mirror reflectance

Reflectance due to direct lighting

$$\frac{f_r(p, (p' - p) \rightarrow \omega_o) L \cos \theta \cos \theta'}{p(p_j) |p - p'|^2}$$

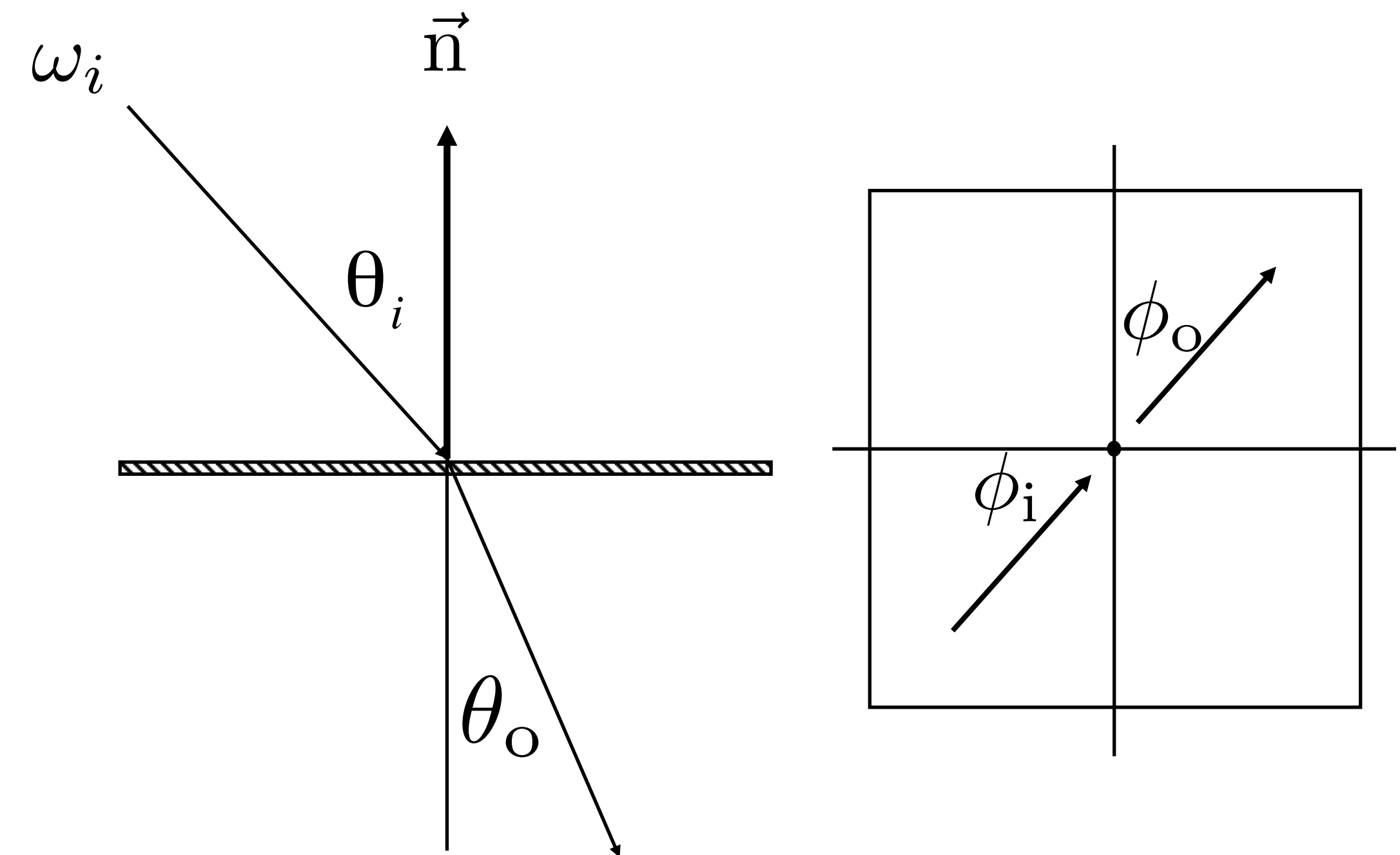
Recall: computing mirror reflection and transmission directions

Reflected ray:

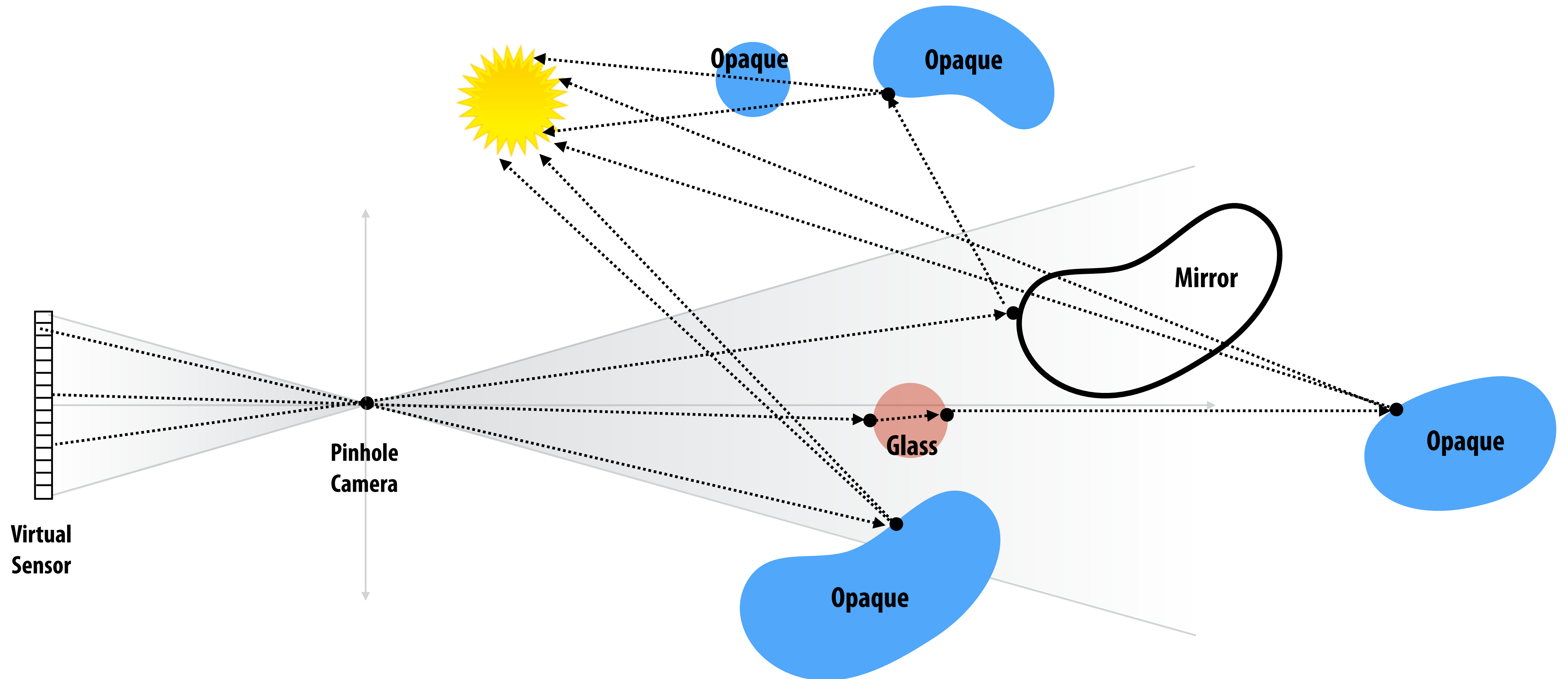


$$\omega_o = -\omega_i + 2(\omega_i \cdot \vec{n})\vec{n}$$

Refracted ray: (transmission)



Basic recursive ray tracing



Whitted's recursive ray tracer

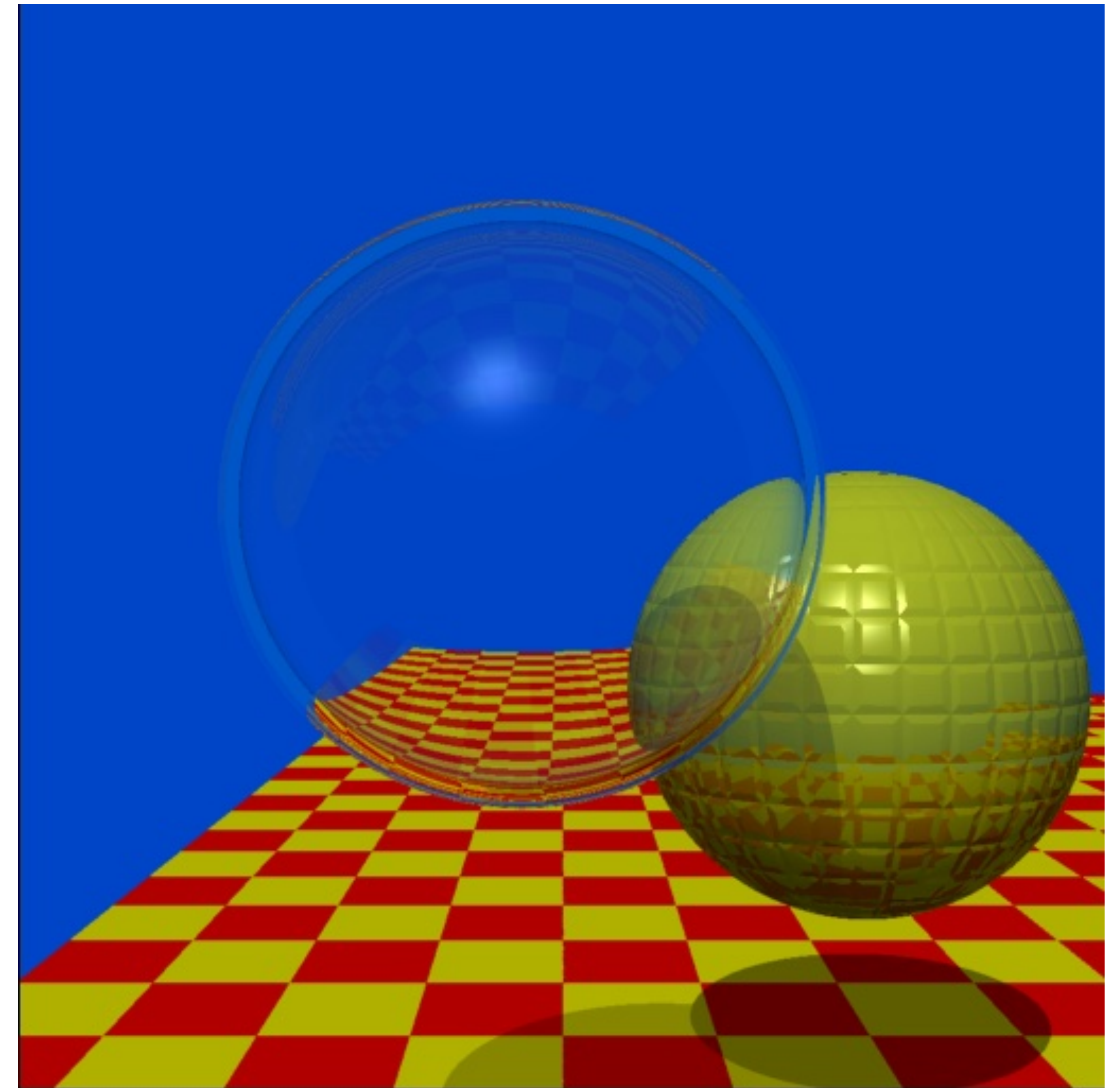
“An improved Illumination model for shaded display”

T. Whitted, CACM 1980

- 1. Always send ray to the light source (unless glass or mirror)**
- 2. Recursively generate reflected rays (mirror) and transmitted rays (glass)**

Time:

- VAX 11/780 (1979) 74m**
- PC (2006) 6s**
- GPU (2012) 1/30s**



Spheres and Checkerboard
T. Whitted, 1979

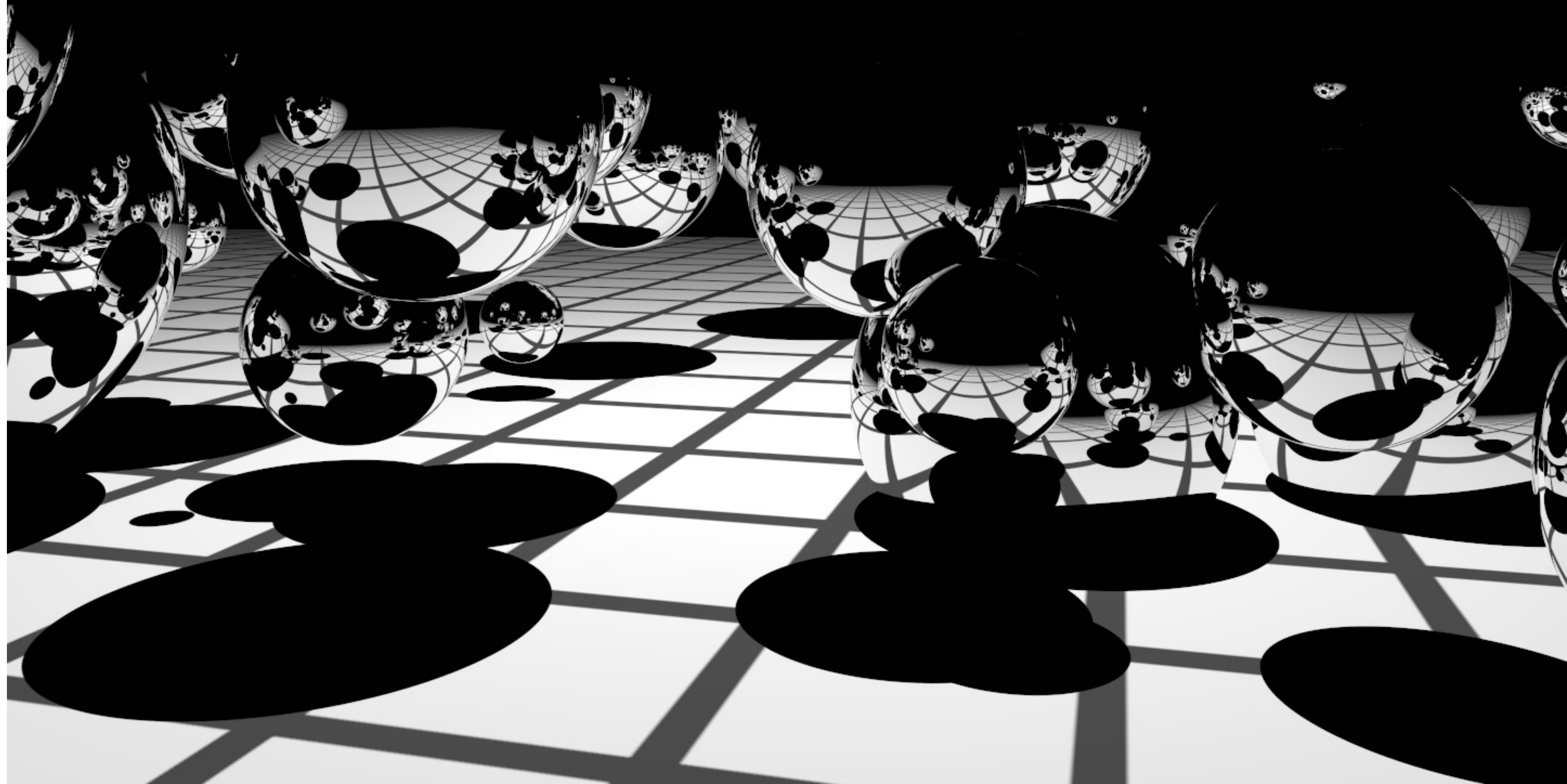
Mirror, depth 1



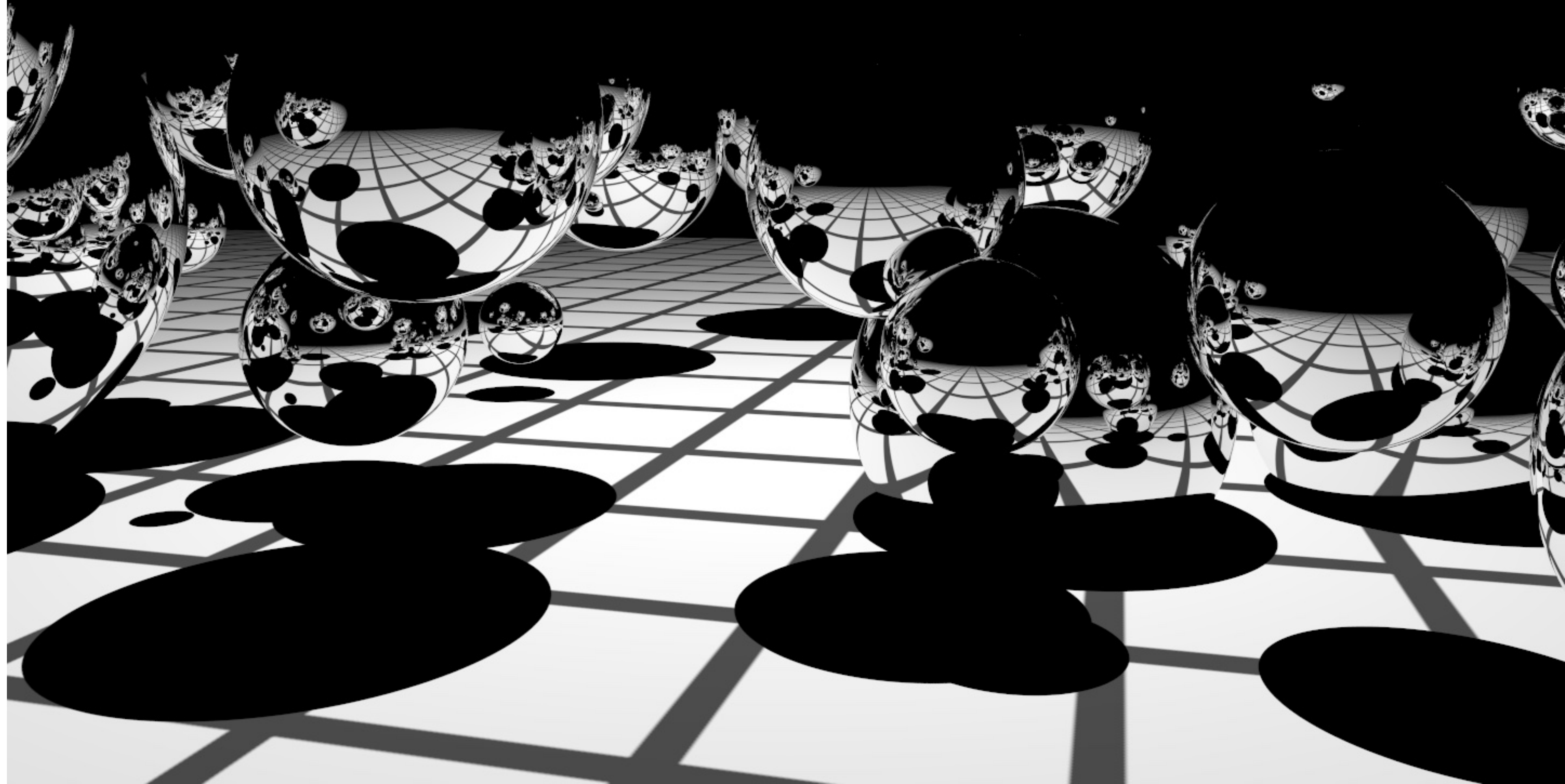
Mirror, depth 2



Mirror, depth 3



Mirror, depth 10



Glass, depth 1



Glass, depth 2



Glass, depth 3



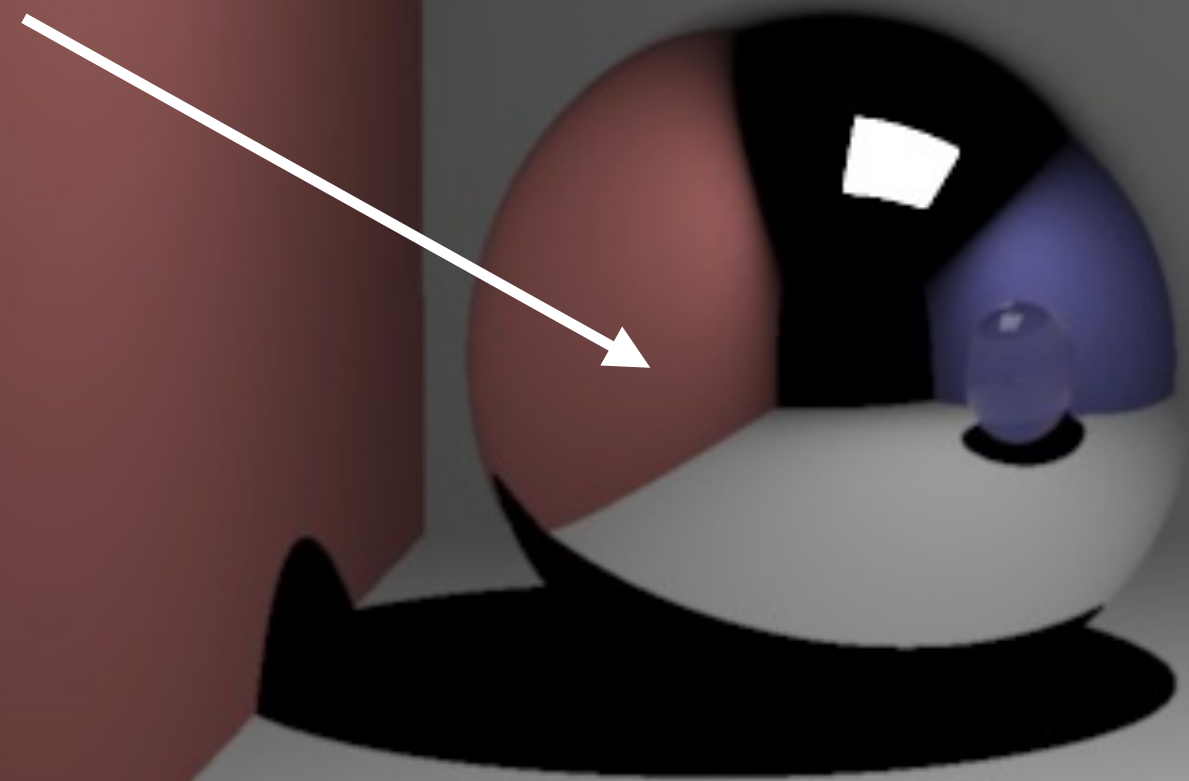
Glass, depth 10



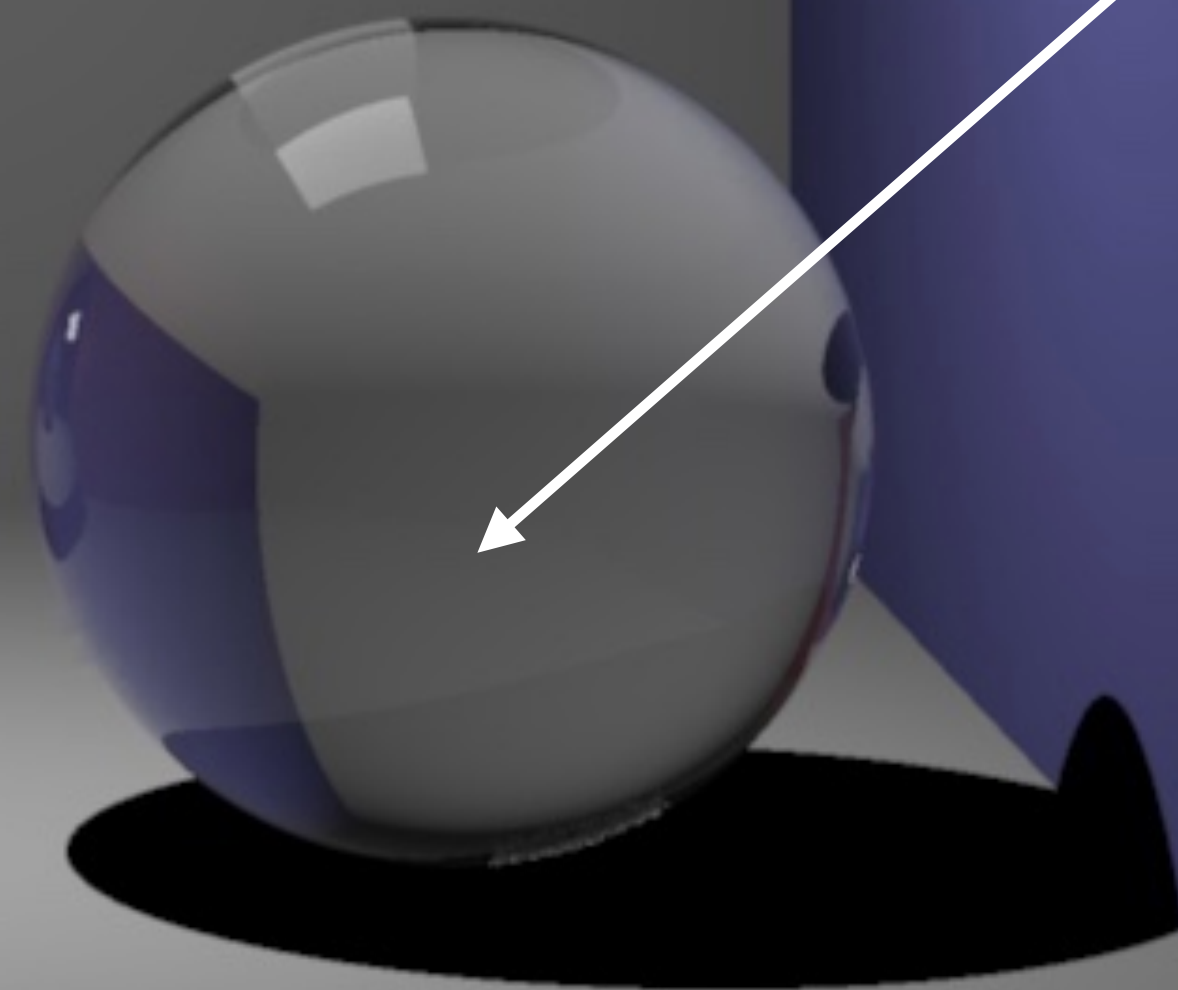
Direct illumination + reflection + transparency

(Assuming point light source)

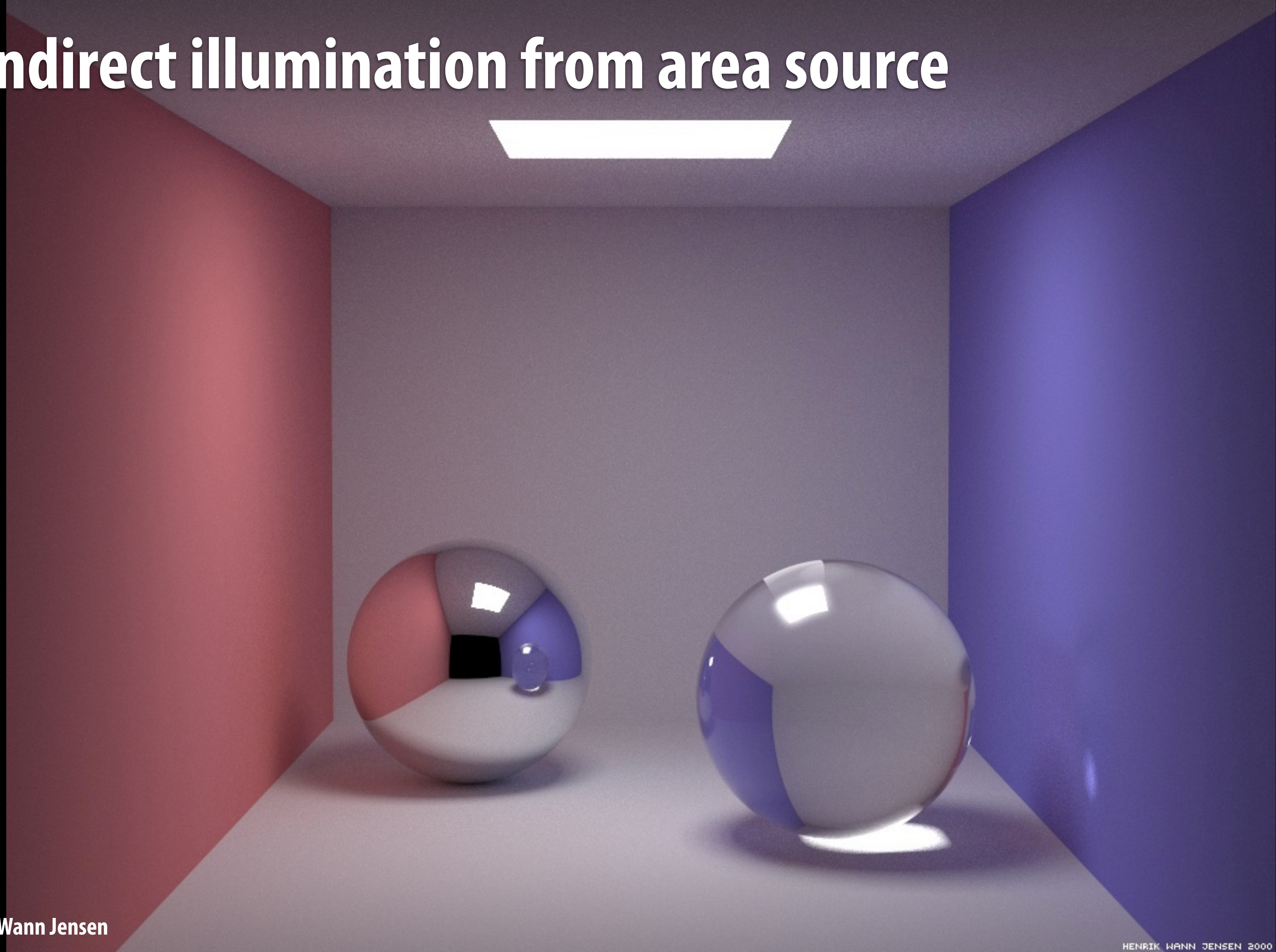
Perfect mirror:



Glass



With indirect illumination from area source



Direct illumination only



• *p*

With indirect illumination



Importance of indirect illumination



Importance of indirect illumination



Importance of indirect illumination



Scene breakdown: by bounce

0 bounces (visible light sources)



1 bounce (direct illumination)



2 bounces



3 bounces



4 bounces



8 bounces



Energy balance: surface balance equation

$$[\text{outgoing}] = [\text{emitted}] + [\text{incoming}] - [\text{absorbed}]$$

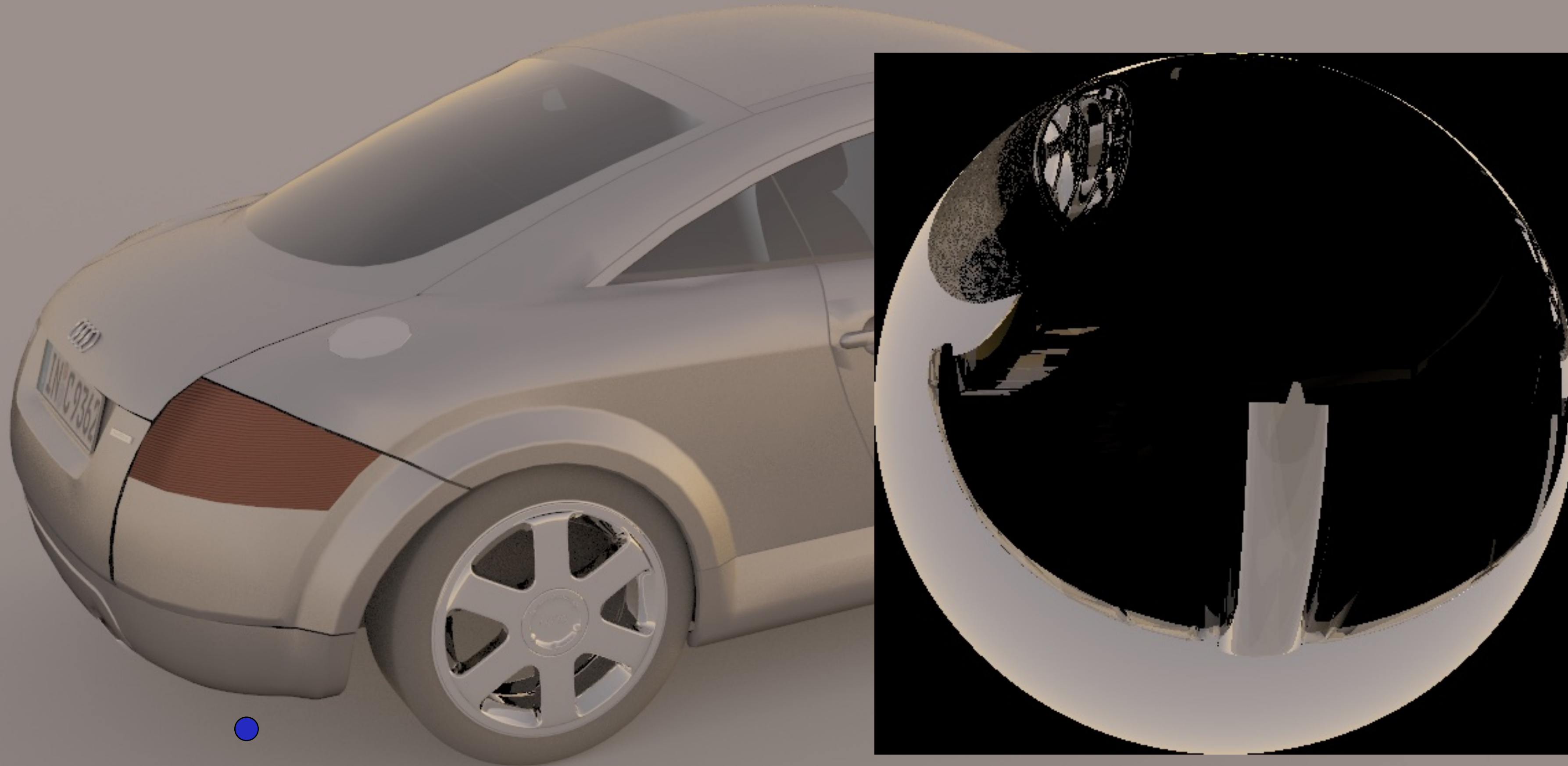
$$[\text{reflected}] = [\text{incoming}] - [\text{absorbed}]$$

$$[\text{outgoing}] = [\text{emitted}] + [\text{reflected}]$$

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

The problem: need to know incident radiance!

Incident radiance function

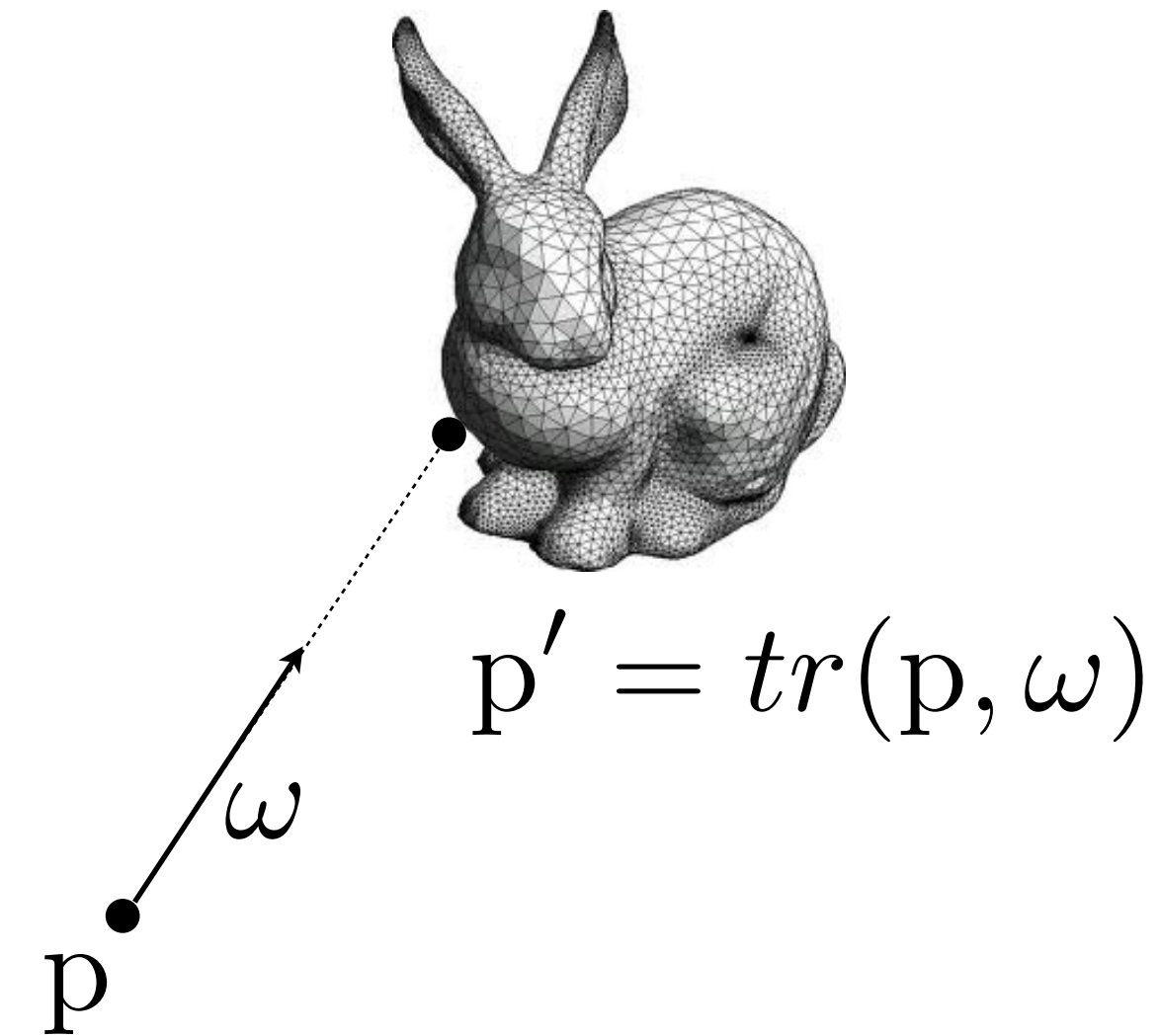


The rendering equation

Radiance invariance along rays:

$$L_i(p, \omega_i) = L_o(tr(p, \omega_i), -\omega_i)$$

“Radiance arriving at p from direction ω_i
is the same as radiance leaving p' from direction $-\omega_i$.”



Rewrite incident radiance in terms of exitant radiance at 1st visible surface:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{H^2} \underset{\substack{\uparrow \\ \text{Light scattering}}}{f_r(p, \omega_i \rightarrow \omega_o)} L_o(\underset{\substack{\uparrow \\ \text{Light transport}}}{tr(p, \omega_i)}, -\omega_i) \cos \theta_i d\omega_i$$

The rendering equation

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

The rendering equation: area form

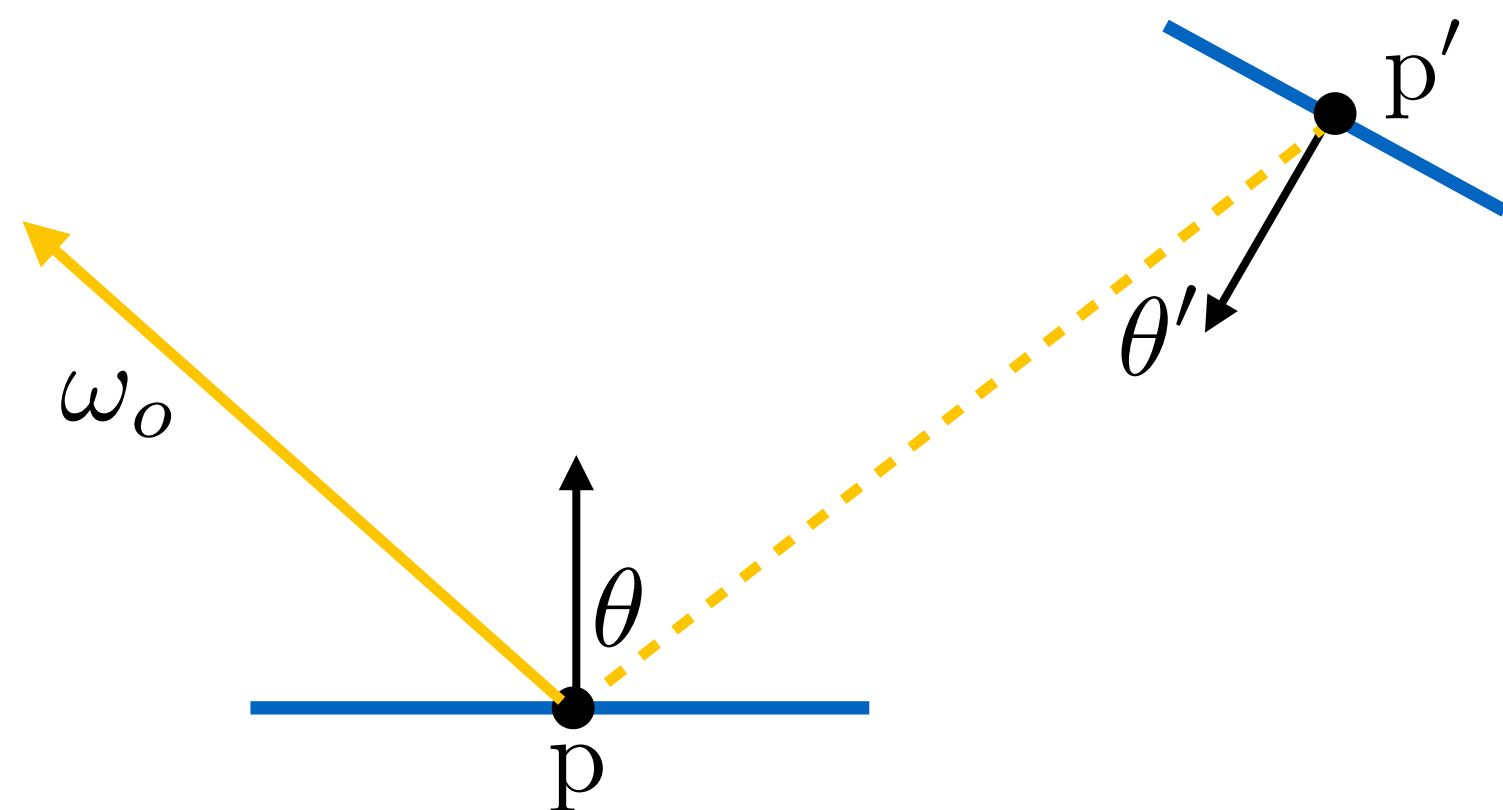
■ Can rewrite rendering equation as an integral over surface area of objects in the scene

- **Apply change of variables** $d\omega = \frac{\cos \theta}{r^2} dA$
- **Introduce binary visibility function:** $V(p \leftrightarrow p')$

$$L_o(p, \omega_o) = L_e(p, \omega_o) +$$

$$\int_A f_r(p, \underbrace{(p' - p)}_{\omega_i} \rightarrow \omega_o) \underbrace{L_o(p', (p - p'))}_{-\omega_i} \cos \theta V(p \leftrightarrow p') \frac{\cos \theta'}{|p - p'|^2} dp'$$

$$= L_e(p, \omega_o) + \int_A f_r(p, (p' - p) \rightarrow \omega_o) L_o(p', (p - p')) G(p \leftrightarrow p') dp'$$



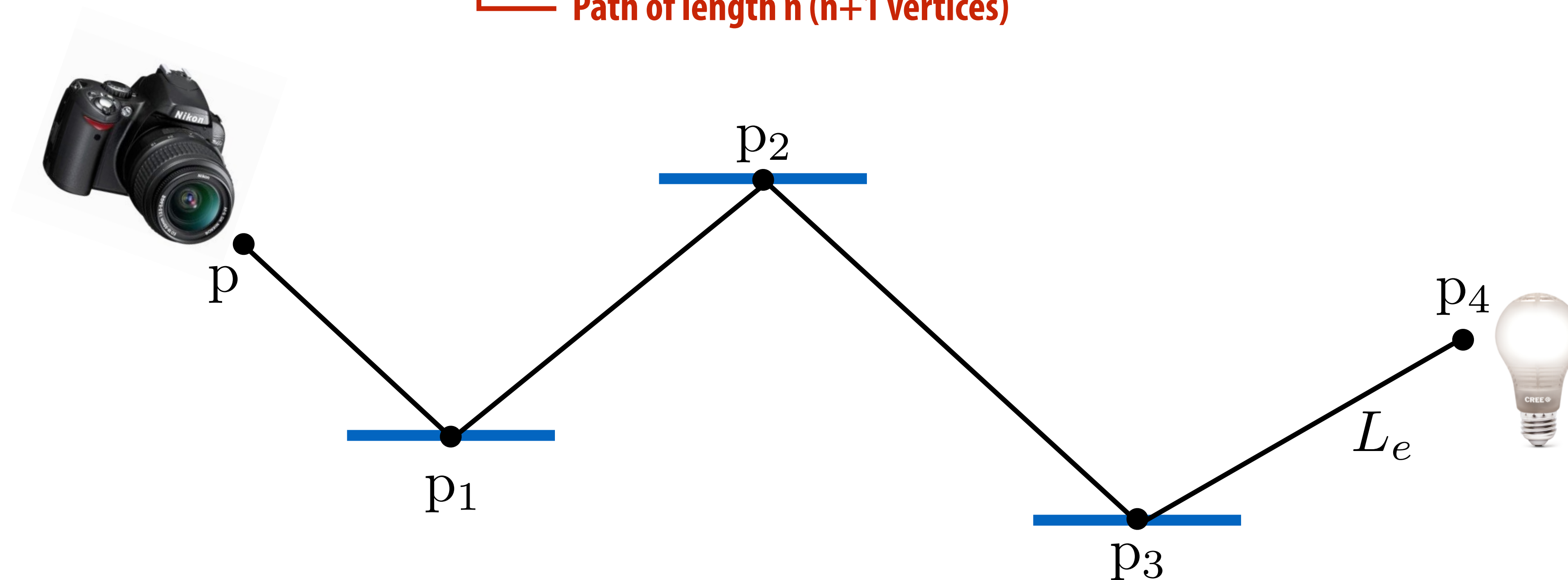
$$G(p \leftrightarrow p') = V(p \leftrightarrow p') \frac{\cos \theta \cos \theta'}{|p - p'|^2}$$

The rendering equation as a sum over paths

$$\begin{aligned}
 L_o(p_1 \rightarrow p) = & L_e(p_1 \rightarrow p) \quad \text{← Path of length 1} \\
 & + \int_A L_e(p_2 \rightarrow p_1) f(p_2 \rightarrow p_1 \rightarrow p) G(p_1 \leftrightarrow p_2) dA(p_2) \quad \text{← Paths of length 2} \\
 & + \int_A \int_A L_e(p_3 \rightarrow p_2) f(p_3 \rightarrow p_2 \rightarrow p_1) G(p_2 \leftrightarrow p_3) f(p_2 \rightarrow p_1 \rightarrow p) G(p_1 \leftrightarrow p_2) dA(p_3) dA(p_2) \\
 & + \dots \quad \text{Paths of length 3}
 \end{aligned}$$

$$L_o(p_1 \rightarrow p) = \sum_{n=1}^{\infty} P(\bar{p}_n)$$

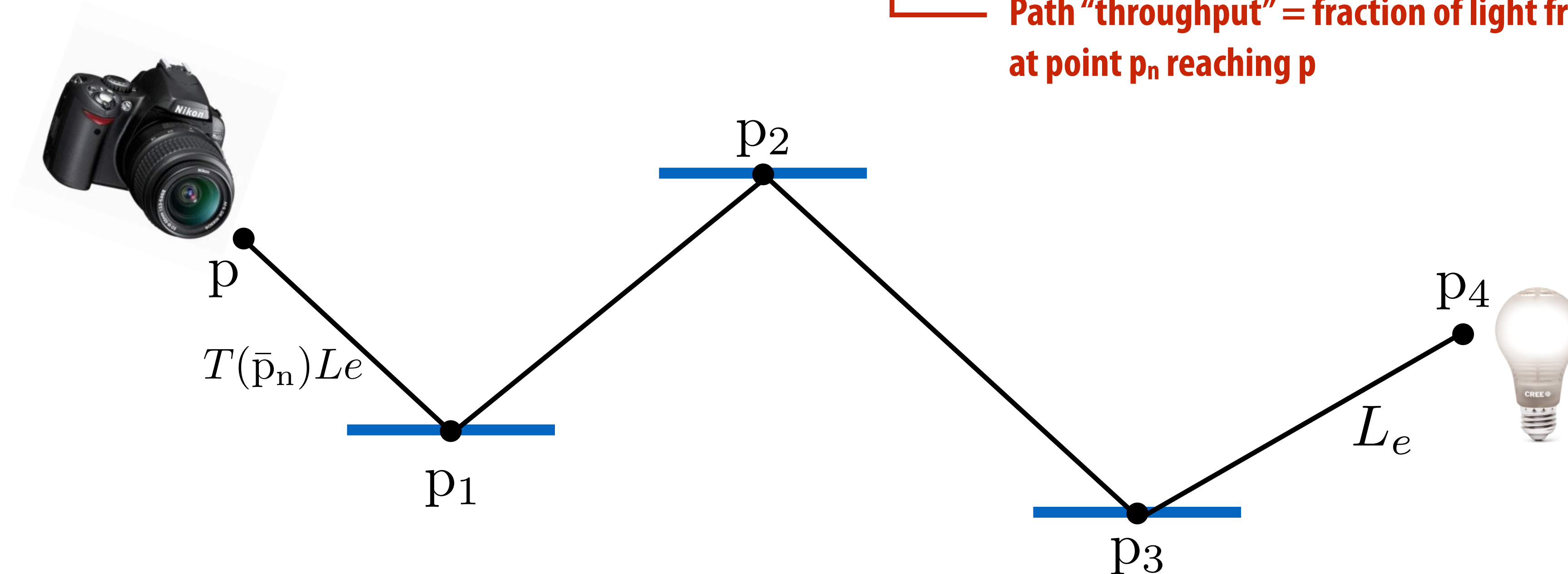
↙ Energy reaching p from all paths of length n
 ↗ Path of length n (n+1 vertices)



The rendering equation as a sum over paths

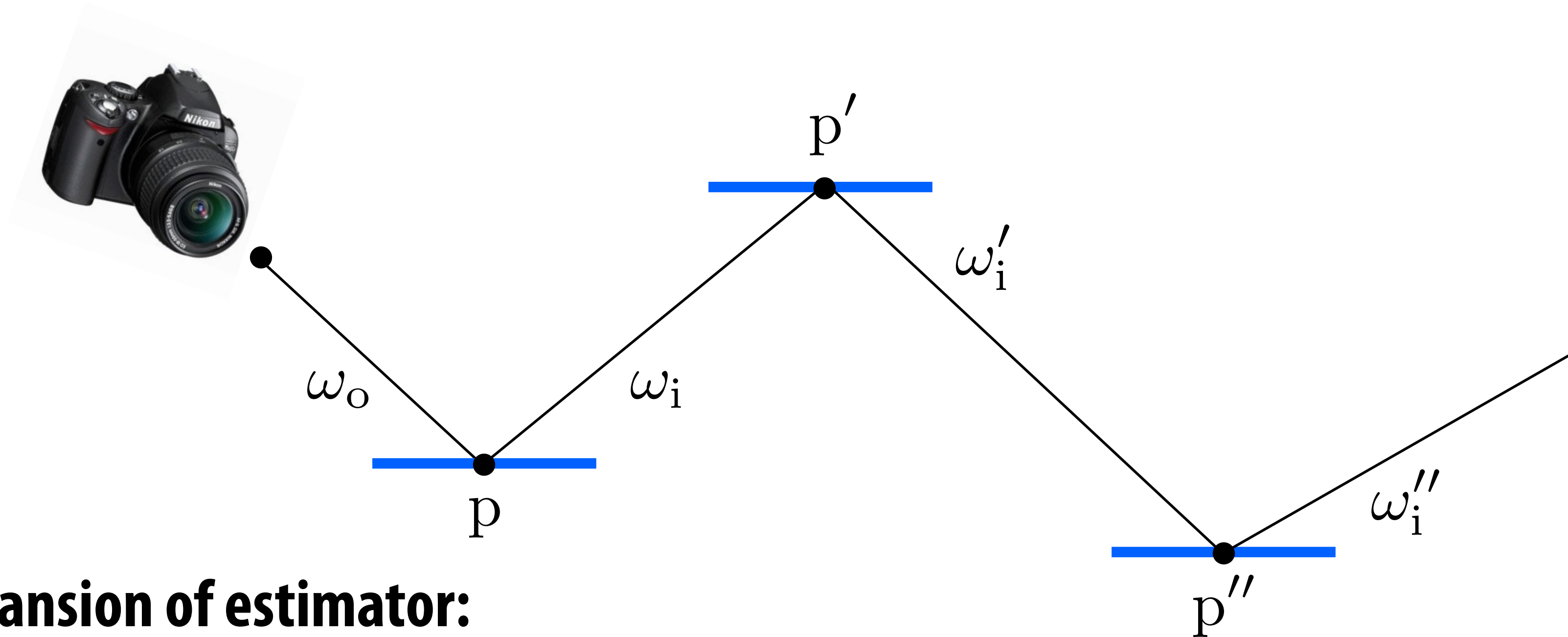
$$\begin{aligned}
 P(\bar{p}_n) &= \int_A \int_A \int_A \cdots \int_A L_e(p_n \rightarrow p_{n-1}) \\
 &\times \left(\prod_{i=1}^{n-1} f(p_{i+1} \rightarrow p_i \rightarrow p_{i-1}) G(p_{i+1} \leftrightarrow p_i) \right) dA(p_2) \cdots dA(p_n) \\
 &= \int_A \int_A \int_A \cdots \int_A L_e(p_n \rightarrow p_{n-1}) T(\bar{p}_n) dA(p_2) \cdots dA(p_n)
 \end{aligned}$$

Path "throughput" = fraction of light from light source at point p_n reaching p



How do we sample paths?

Idea: generate random path incrementally starting at camera



Recursive expansion of estimator:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \frac{f_r(\omega_o, \omega_i) \cos \theta_i}{p(\omega_i)} L_o(p', -\omega_i)$$

$$= L_e + \frac{f_r \cos \theta_i}{p(\omega_i)} \left[L_e(p', -\omega_i) + \frac{f_r(-\omega_i, \omega'_i) \cos \theta'_i}{p(\omega'_i)} L_o(p'', \omega''_i) \right]$$

$= \dots$

Basic recursive path tracing

```
Spectrum PathLo(Ray ray) {  
    Intersection isect = scene->Intersect(ray);  
    BSDF bsdf = isect.GetBSDF();  
    Vector3f wo = -ray.d, wi;  
    Float pdf;  
    Spectrum fr = bsdf.Sample_f(wo, &wi, &pdf);  
  
    return isect.Le(wo) +  
        fr * PathLo(Ray(isect.P, wi)) * Dot(wi, isect.N) / pdf;  
}
```

Note how over the entire path the indirect illumination is modulated by product of probabilities of individual path segments.

Kajiya, 1986



Figure 6. A sample image. All objects are neutral grey. Color on the objects is due to caustics from the green glass balls and color bleeding from the base polygon.

One sample per pixel



32 samples per pixel



1024 samples per pixel

Another example:



16 paths/pixel



64 paths/pixel



256 paths/pixel



1024 paths/pixel



4096 paths/pixel



The postcard-sized path tracer

```
#include <stdlib.h> // card > pixar.ppm
#include <stdio.h>
#include <math.h>
#define R return
#define O operator
typedef float F; typedef int I; struct V{F x,y,z; V(F v=0){x=y=z=v;} V(F a,F b,F c=0){x=a;y=b;z=c;} V O+(V r){R V(x+r.x,y+r.y,z+r.z);} V O*(V r){R V(x*r.x,y*r.y,z*r.z);} F O%(V r){R x*r.x+y*r.y+z*r.z;} V O!(){R*this*(1/sqrtf(*this*this));}}; F L(F l,F r){R l<r?l:r;} F U(){R(F)rand()/RAND_MAX;} F B(V p,V l,V h){l=p+l*-1;h=h+p*-1;R-L(L(L(l.x,h.x),L(l.y,h.y)),L(l.z,h.z));} F S(V p,I&m){F d=1\
e9;V f=p;f.z=0;char l[]="505_5W9W5_9_COC_AOEOA_E_IOQ_I_QOUOY_Y_]OWW[WaOa_aW\
eWa_e_cWiO";for(I i=0;i<60;i+=4){V b=V(l[i]-79,l[i+1]-79)*.5,e=V(l[i+2]-79,l[i+3]-79)*.5+b*-1,o=f+(b+e*L(-L((b+f*-1)%e/(e*e),0),1))*-1;d=L(d,o%o);}d=sq\
rtf(d);V a[]={V(-11,6),V(11,6)};for(I i=2;i--;){V o=f+a[i]*-1;d=L(d,o.x>0?f\
absf(sqrtf(o%o)-2):(o.y+=o.y>0?-2:2,sqrtf(o%o)));}d=powf(powf(d,8)+powf(p.z,8),.125)-.5;m=1;F r=L(-L(B(p,V(-30,-.5,-30),V(30,18,30)),B(p,V(-25,17,-25),V(25,20,25))),B(V(fmodf(fabsf(p.x),8),p.y,p.z),V(1.5,18.5,-25),V(6.5,20,25)))\
;if(r<d)d=r,m=2;F s=19.9-p.y;if(s<d)d=s,m=3;R d;}I M(V o,V d,V&h,V&n){I m,s=0;F t=0,c;for(;t<100;t+=c)if((c=S(h=o+d*t,m))<.01||++s>99)R n=!V(S(h+V(.01,0),s)-c,S(h+V(0,.01),s)-c,S(h+V(0,0,.01),s)-c),m;R 0;}V T(V o,V d){V h,n,r,t=1,l(!V(.6,.6,1));for(I b=3;b--;){I m=M(o,d,h,n);if(!m)break;if(m==1){d=d+n*(n%d*-2);o=h+d*.1;t=t*.2;}if(m==2){F i=n%1,p=6.283185*U(),c=U(),s=sqrtf(1-c),g=n.z<0?-1:1,u=-1/(g+n.z),v=n.x*n.y*u;d=V(v,g+n.y*n.y*u,-n.y)*(cosf(p)*s)+V(1+g*n.x*n.x*u,g*v,-g*n.x)*(sinf(p)*s)+n*sqrtf(c);o=h+d*.1;t=t*.2;if(i>0&&M(h+n*.1,l,h,n)==3)r=r+t*V(500,400,100)*i;}if(m==3){r=r+t*V(50,80,100);break;}}R r;}I main(){I w=960,h=540,s=16;V e(-22,5,25),g=!(V(-3,4,0)+e*-1),l=!V(g.z,0,-g.x)*(1./w),u(g.y*l.z-g.z*l.y,g.z*l.x-g.x*l.z,g.x*l.y-g.y*l.x);printf("P\
6 %d %d 255 ",w,h);for(I y=h;y--;)for(I x=w;x--;){V c;for(I p=s;p--;)c=c+T(e,! (g+l*(x-w/2+U())+u*(y-h/2+U())));c=c*(1./s)+14./241;V o=c+1;c=V(c.x/o.x,c.y/o.y,c.z/o.z)*255;printf("%c%c%c",(I)c.x,(I)c.y,(I)c.z);}}// Andrew Kensler
```



[Andrew Kensler]

[Decyphering the postcard sized path tracer, Fabien Sanglard]

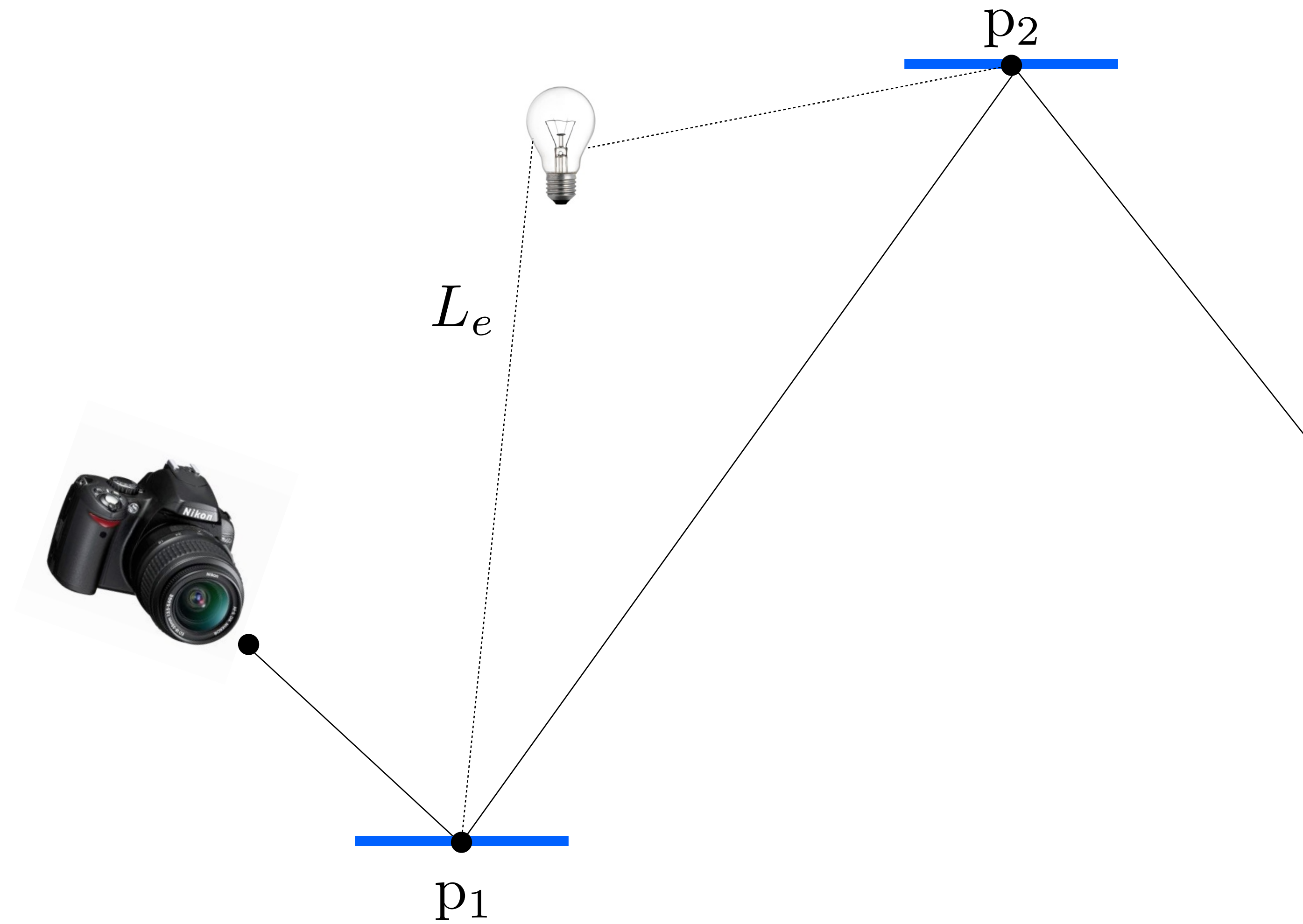
Partitioning the rendering equation

$$L_i(\mathbf{p}, \omega_i) = L_{i,d}(\mathbf{p}, \omega_i) + L_{i,i}(\mathbf{p}, \omega_i)$$

- **Incident direct illumination:** $L_{i,d}(\mathbf{p}, \omega_i)$
 - Sample lights+BRDFs
- **Incident indirect illumination:** $L_{i,i}(\mathbf{p}, \omega_i)$
 - Recursive evaluation of rendering eqn.

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

Path tracing



Path tracing: sampling indirect illumination

$$\int_{H^2} f_r(\omega_i \rightarrow \omega_o) L_{i,i}(\mathbf{p}, \omega_i) \cos \theta_i d\omega_i$$

- **Sample incoming direction from some distribution (e.g. proportional to BRDF):** $\omega_i \sim p(\omega)$
- **Recursively call path tracing function to compute incident indirect radiance**

- **Estimator:**

$$\frac{f_r(\omega_i \rightarrow \omega_o) L_{i,i}(\mathbf{p}, \omega_i) \cos \theta_i}{p(\omega_i)}$$

$$\frac{f_r(\omega_i \rightarrow \omega_o) L_o(tr(\mathbf{p}, \omega_i), -\omega_i) \cos \theta_i}{p(\omega_i)}$$

Path tracing: recursive

$$L_o(p, \omega_o) = L_e(p, \omega_o) +$$

$$\int_{H^2} f_r(\omega_i \rightarrow \omega_o) L_{i,d}(p, \omega_i) \cos \theta_i d\omega_i +$$

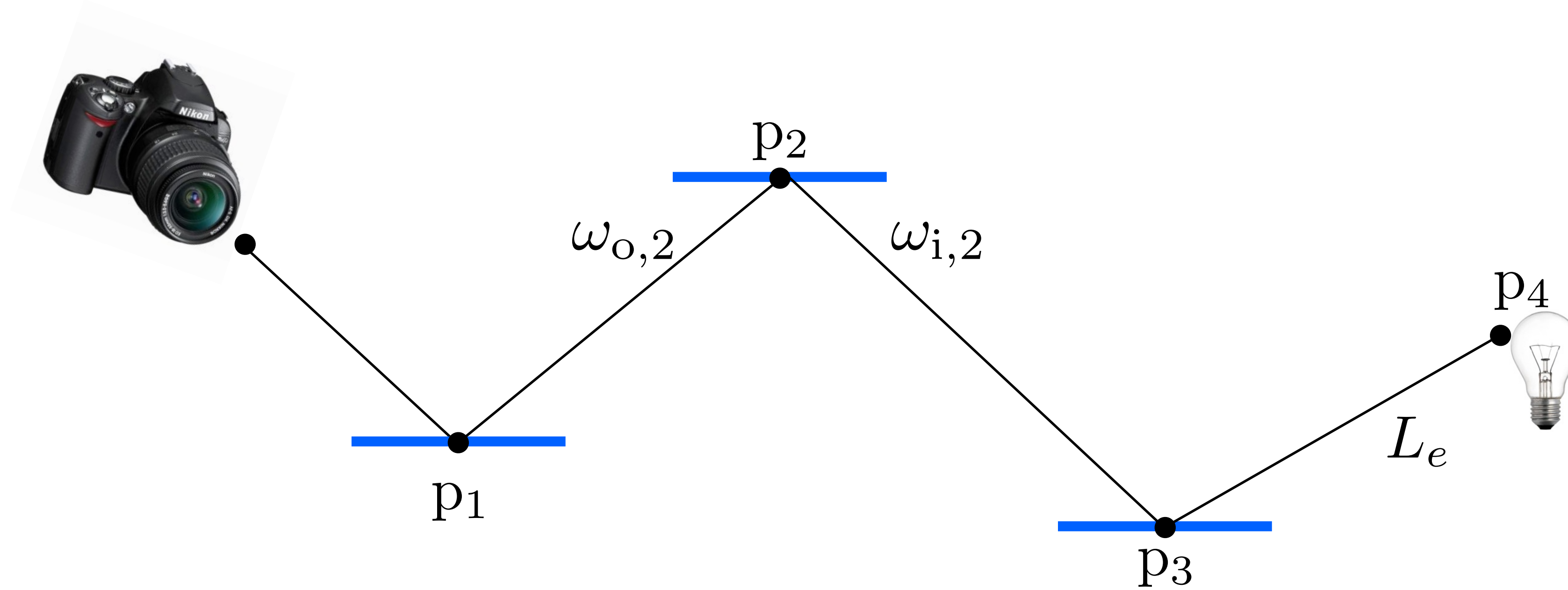
$$\int_{H^2} f_r(\omega_i \rightarrow \omega_o) L_o(tr(p, \omega_i), -\omega_i) \cos \theta_i d\omega_i$$

```
Spectrum PathLo(Ray ray) {  
    Intersection isect = scene->Intersect(ray);  
    BSDF bsdf = isect.GetBSDF();  
    Vector3f wo = -ray.d;  
  
    Spectrum Ld = ReflFromDirectLighting(bsdf, wo);  
  
    Spectrum fr = bsdf.Sample_f(wo, &wi, &pdf);  
    return (ray.depth == 0 ? isect.Le(wo) : 0.) + Ld +  
           fr * PathLo(Ray(isect.P, wi, ray.depth+1)) * Dot(wi, isect.N) / pdf;  
}
```

Reflectance due to incident
direct lighting

Reflectance due to incident
indirect lighting

Path contribution



$$\beta(\bar{p}) = \prod_j \frac{f_r(p_j, \omega_{o,j}, \omega_{i,j}) \cos \theta_{i,j}}{p(\omega_{i,j})}$$

$$L_i = \sum \beta(\bar{p}) L_e$$

Path tracing: iterative version

```
Spectrum PathLo(Ray ray) {  
    Spectrum Lo = 0, beta = 1;  
    int depth = 0;  
    while (true) {  
        Intersection isect = scene->Intersect(ray);  
        Vector3f wo = -ray.d;  
        if (depth == 0) Lo += isect.Le(wo);  
        BSDF brdf = isect.GetBSDF();  
  
        Lo += beta * ReflFromDirectLighting(brdf, wo);  
  
        Spectrum fr = brdf.Sample_f(wo, &wi, &pdf);  
        beta *= fr * Dot(wi, isect.N) / pdf;  
        depth++;  
        ray = Ray(isect.P, wi);  
    }  
    return Lo;  
}
```

What's the problem with this implementation?

Russian Roulette

- Define path termination probability q
- Randomly terminate path based on q ;
 - for surviving paths, scale contribution by $\frac{1}{1 - q}$
- Russian roulette gives expectation:

$$(1 - q)E \left[\frac{X}{1 - q} \right] + qE[0] = E[X]$$

Russian Roulette

```
Spectrum PathLo(Ray ray) {  
    Spectrum Lo = 0, beta = 1;  
    int depth = 0;  
    while (true) {  
        Intersection isect = scene->Intersect(ray);  
        Vector3f wo = -ray.d;  
        if (depth == 0) Lo += isect.Le(wo);  
  
        BSDF bsdf = isect.GetBSDF();  
        Lo += beta * ReflFromDirectLighting(bsdf, wo);  
  
        Spectrum fr = bsdf.Sample_f(wo, &wi, &pdf);  
        beta *= fr * Dot(wi, isect.N) / pdf;  
  
        float q = 0.25;  
        if (UniformFloat() < q) break;  
        else beta /= (1-q);  
        depth++;  
        ray = Ray(isect.P, wi);  
    }  
    return Lo;  
}
```

Recall the estimator:

$$\frac{f_r(\omega_i \rightarrow \omega_o) L_o(tr(p, \omega_i), -\omega_i) \cos \theta_i}{p(\omega_i)}$$

P(choosing ω_i | not_terminating) P(not terminating)

Improving Russian Roulette

- Recurring principle: It's best to avoid spending computation on samples that make a small contribution
- How do you know a sample will make a small contribution before you evaluate it?
- Recall: $L_i = \beta(\bar{p}) L_e$
- $\beta(\bar{p})$ is a reasonable proxy for expected contribution

Russian Roulette: better

```
Spectrum PathLo(Ray ray) {  
    Spectrum Lo = 0, beta = 1;  
    int depth=0;  
    while (true) {  
        Intersection isect = scene->Intersect(ray);  
        Vector3f wo = -ray.d;  
        if (depth == 0) Lo += isect.Le(wo);  
  
        BSDF bsdf = isect.GetBSDF();  
        Lo += beta * ReflFromDirectLighting(bsdf, wo);  
  
        Spectrum fr = bsdf.Sample_f(wo, &wi, &pdf);  
        beta *= fr * Dot(wi, isect.N) / pdf;  
  
        Float q = 1 - beta.MaxComponentValue();  
        if (UniformFloat() < q) break;  
        beta /= 1 - q;  
        depth++;  
        ray = Ray(isect.P, wi);  
    }  
    return Lo;  
}
```

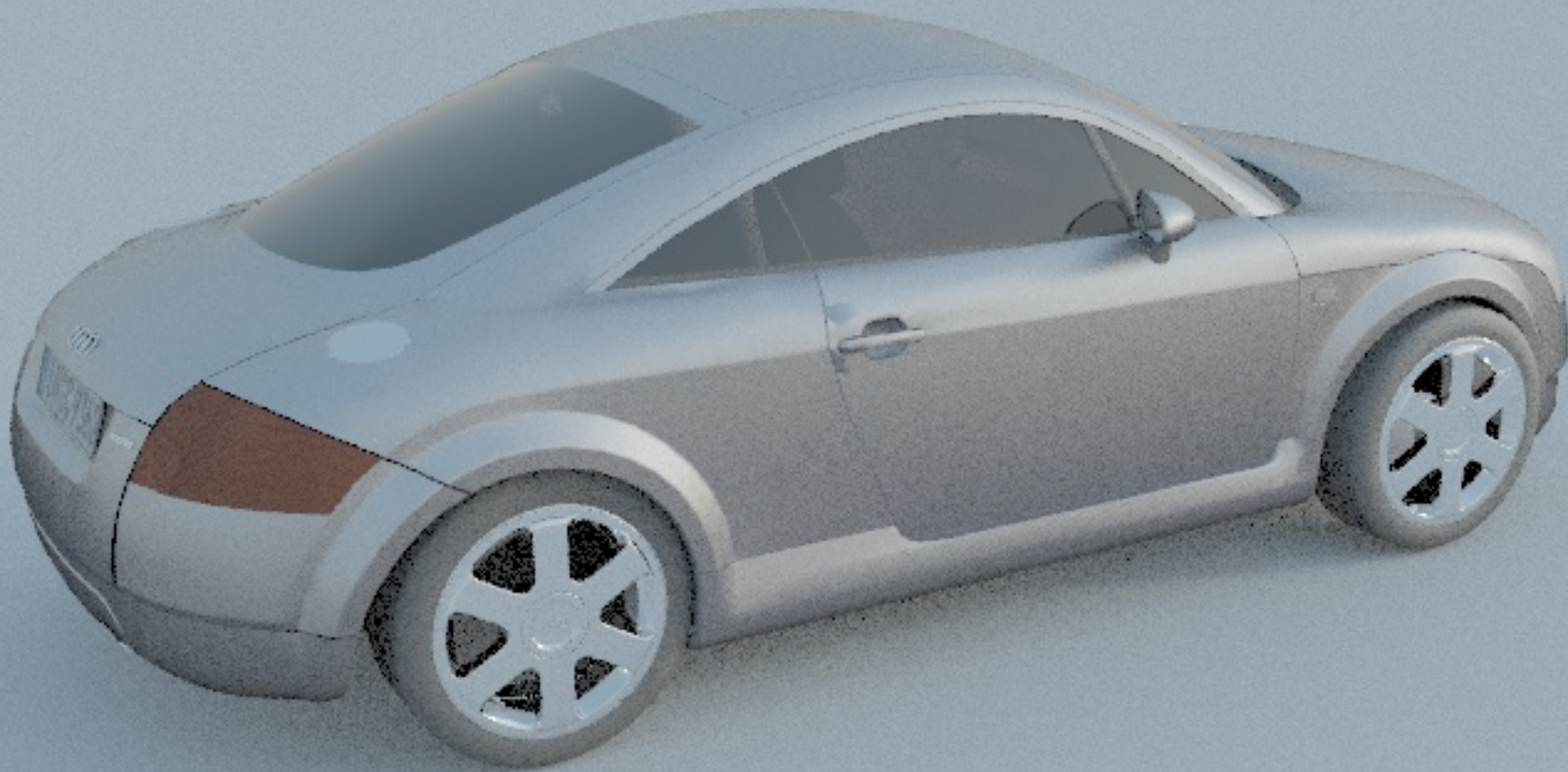
High path throughput yields low
termination probability

$$\text{Efficiency} \propto \frac{1}{\text{Variance} \times \text{Cost}}$$



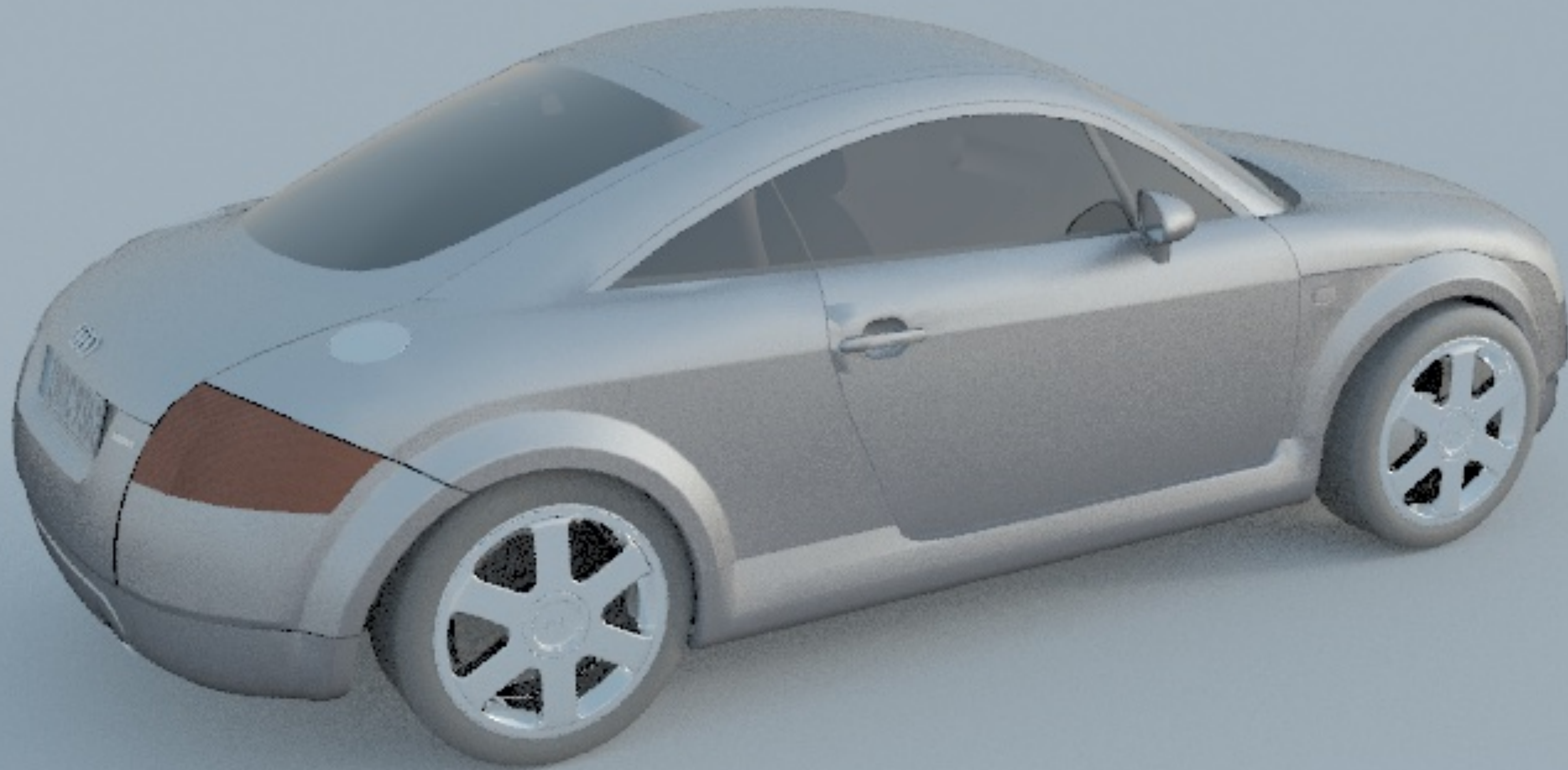
Terminate 90%: 2.5 seconds
MSE 0.0124, MC Efficiency 32.90

$$\text{Efficiency} \propto \frac{1}{\text{Variance} \times \text{Cost}}$$



Terminate proportional to path contribution: 2.9 seconds
MSE 0.00413, MC Efficiency 84.76

$$\text{Efficiency} \propto \frac{1}{\text{Variance} \times \text{Cost}}$$



No Russian Roulette: 3.9 seconds
MSE 0.00379, MC Efficiency 67.4

Advanced topic: path guiding

- Recall the MC estimator:

$$\frac{f_r(\omega_i \rightarrow \omega_o) L_o(tr(p, \omega_i), -\omega_i) \cos \theta_i}{p(\omega_i)}$$

- Regular path tracing: sample
(or something similar to it)

$$\omega_i \sim f_r(\omega_i \rightarrow \omega_o)$$

- But: really want to sample $\propto f_r L_o \cos \theta$

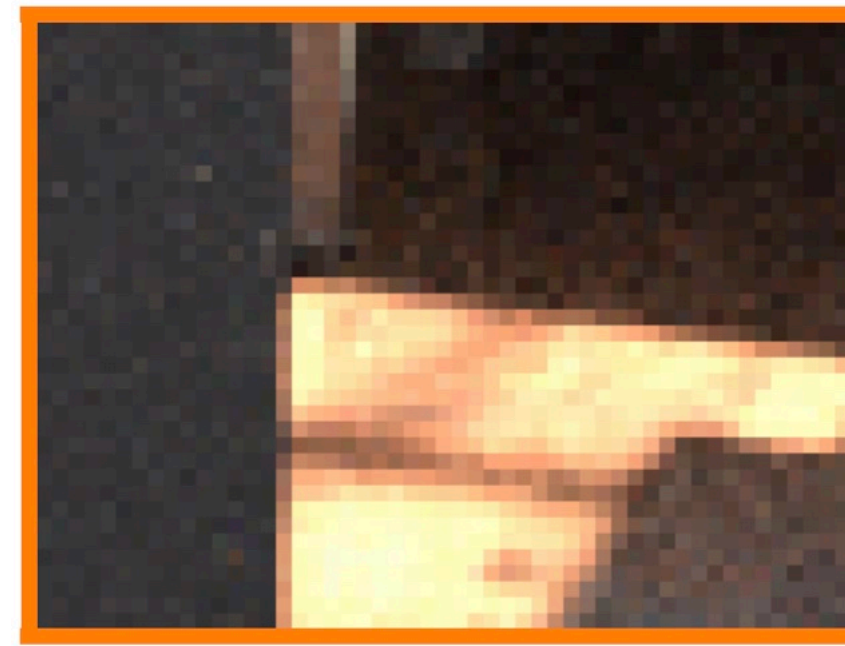
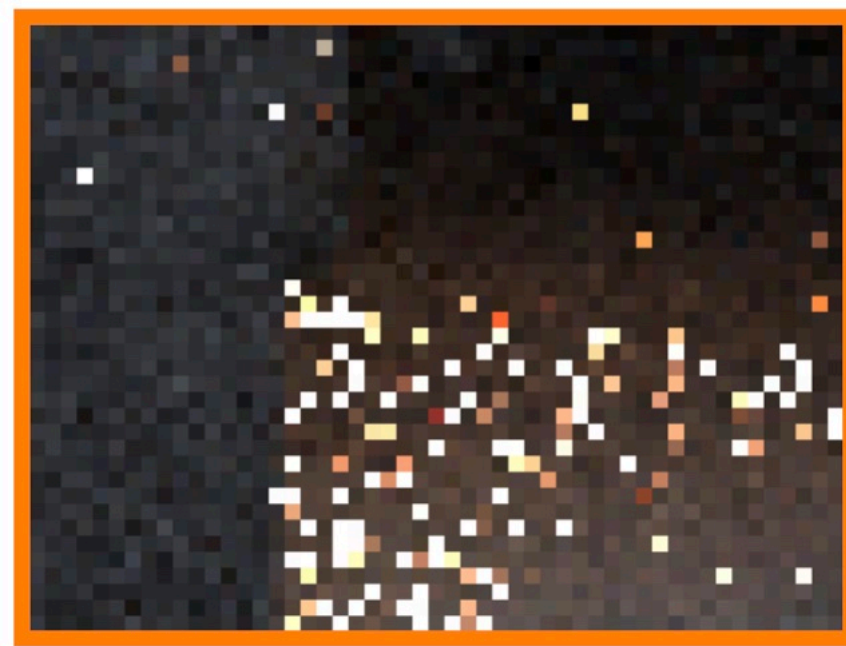
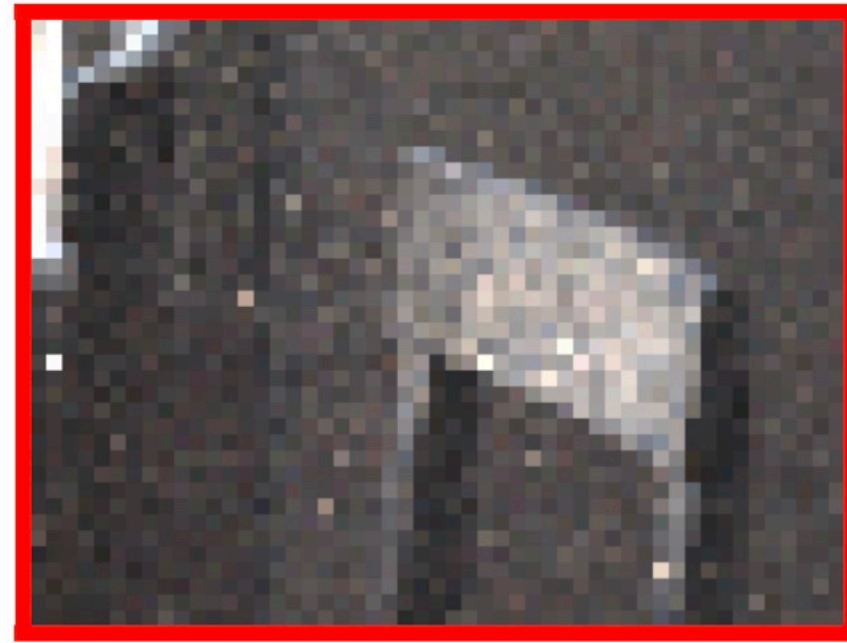
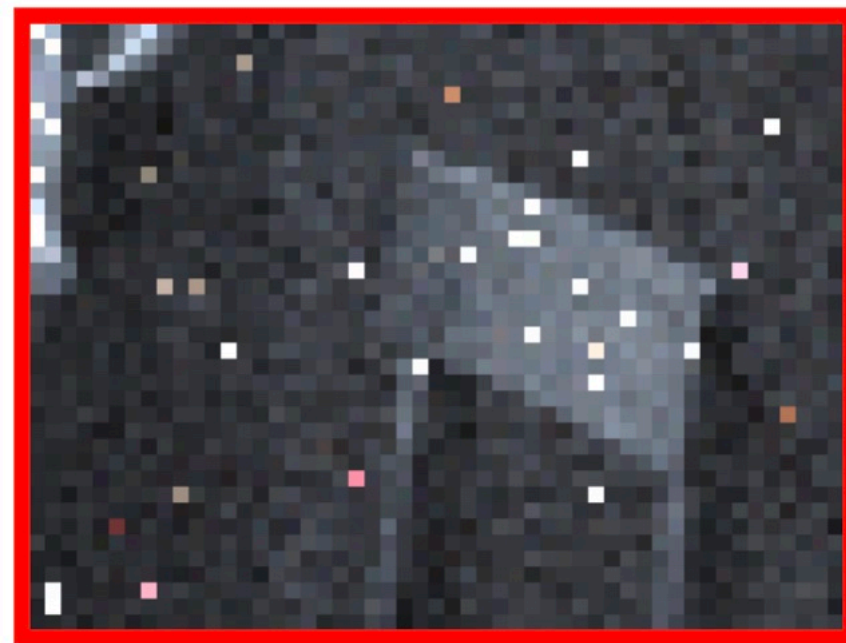
- Idea: learn the distribution of light in the scene to guide sampling

Advanced topic: path guiding

Use results from prior paths to
influence choice of future paths.



[Müller et al. 2018]

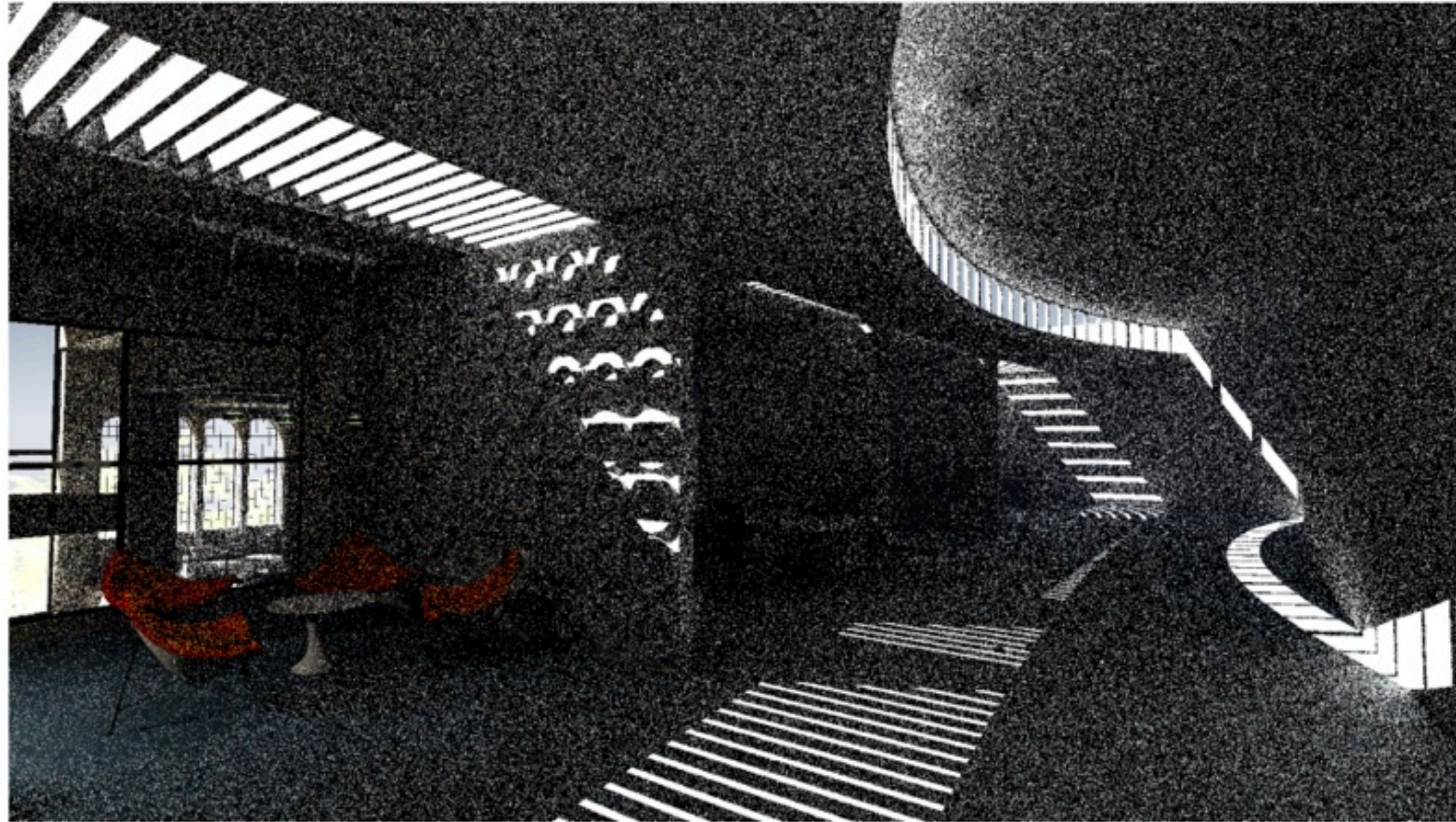


Baseline

PPG

Neural

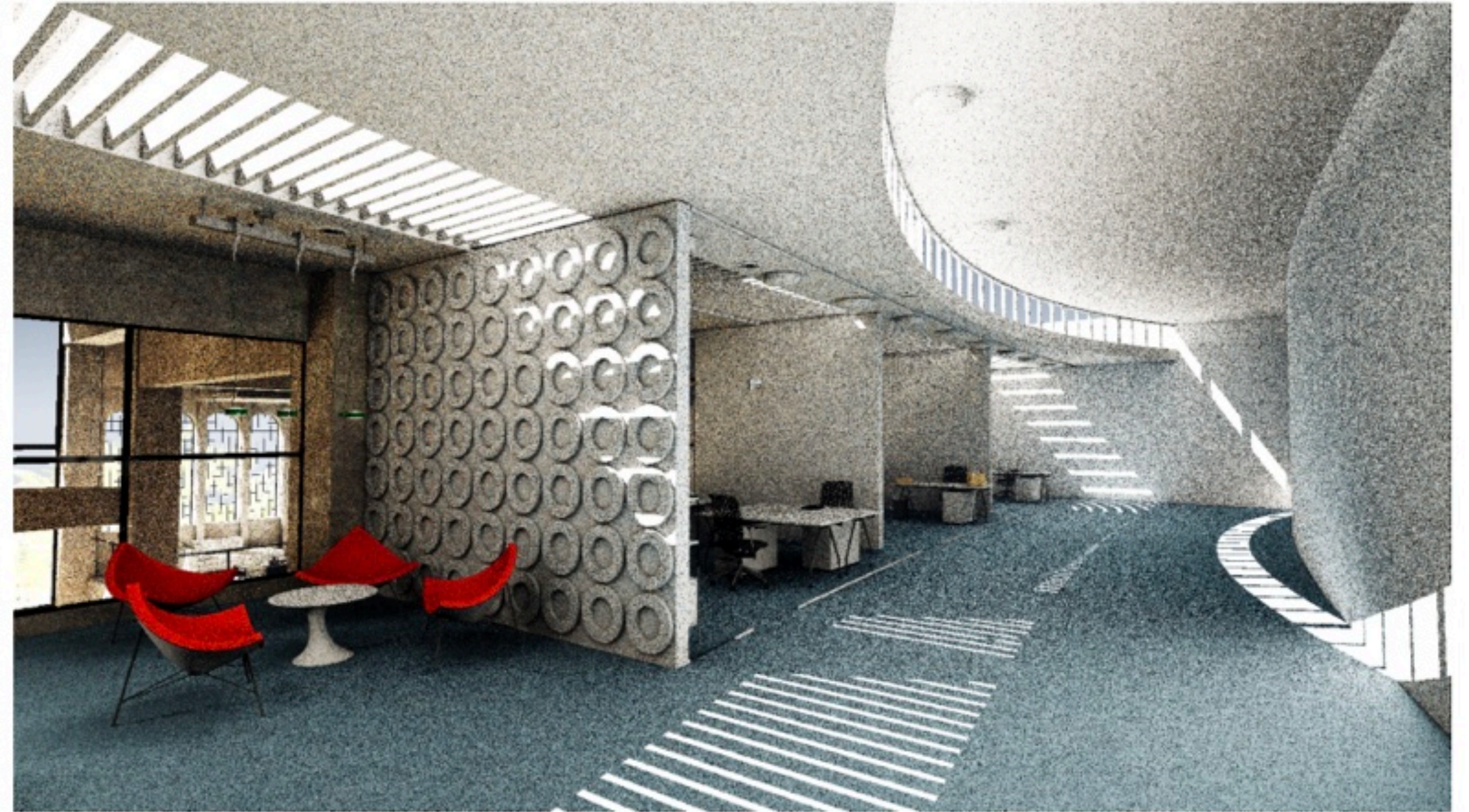
Recent work: caching/reusing good paths



Path traced: 1 path/pixel (8 ms/frame)

Key idea: cache good paths, reuse good paths found from prior frames or for prior pixels in same frame

[Ouyang et al. 2021]



Path traced: 1 path/pixel using ReSTIR GI (8.9 ms/frame)



High sample count path traced "ground truth"

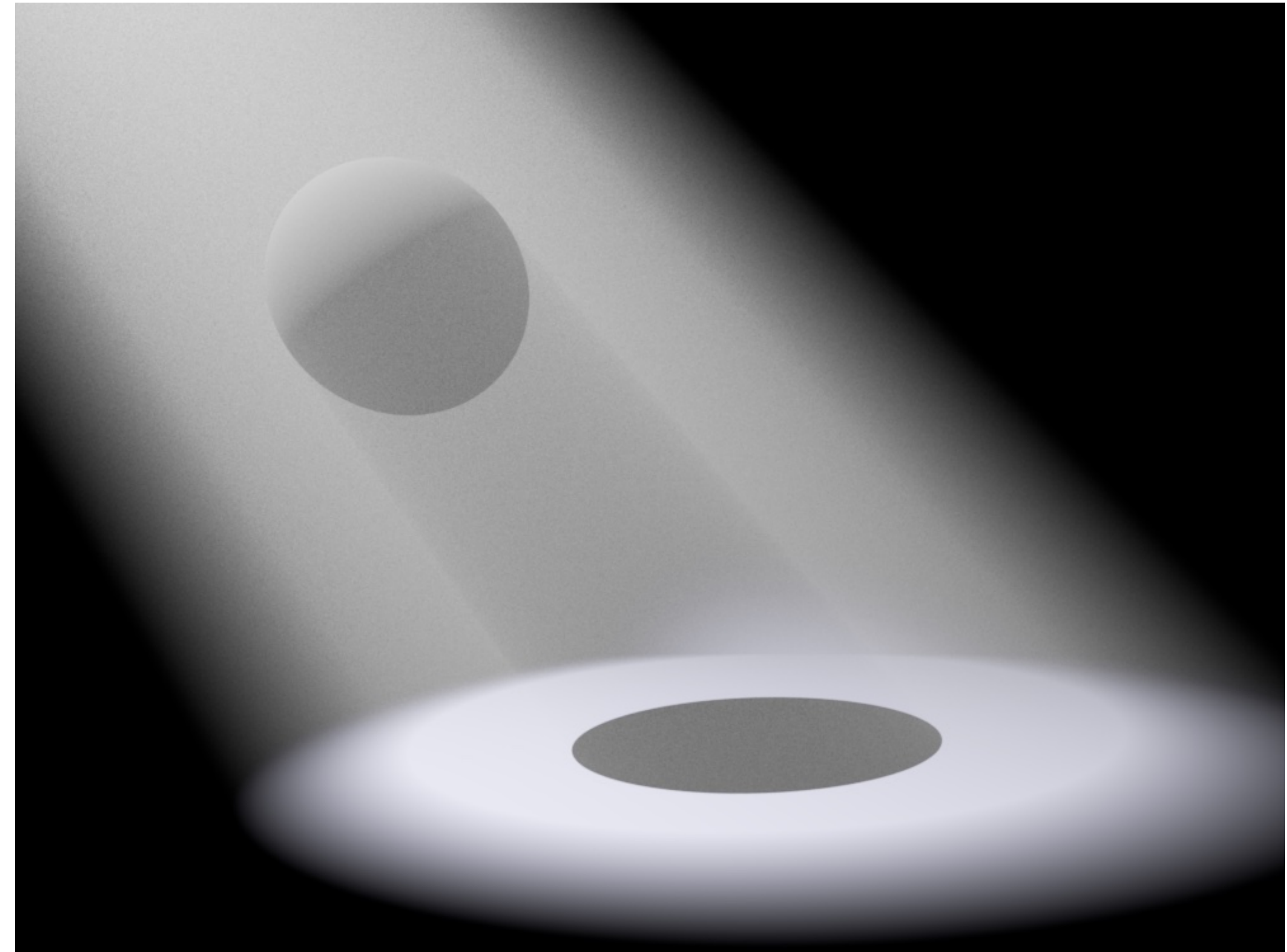
Path tracing summary

- **Path tracing: sample paths through scene (from the domain of all possible paths)**
 - **Unbiased estimator of solution to the full rendering equation (provided paths are correctly weighted by their probabilities... incorrect PDFs are a common source of bugs)**
- **Simple, elegant, brute force! ;-)**
- **Efficiency comes from intelligent biasing of paths toward the most “important” paths.**
 - **But that’s a topic for a more advanced rendering course!**
- **Next time: modern techniques (and hacks!) for real-time performance**

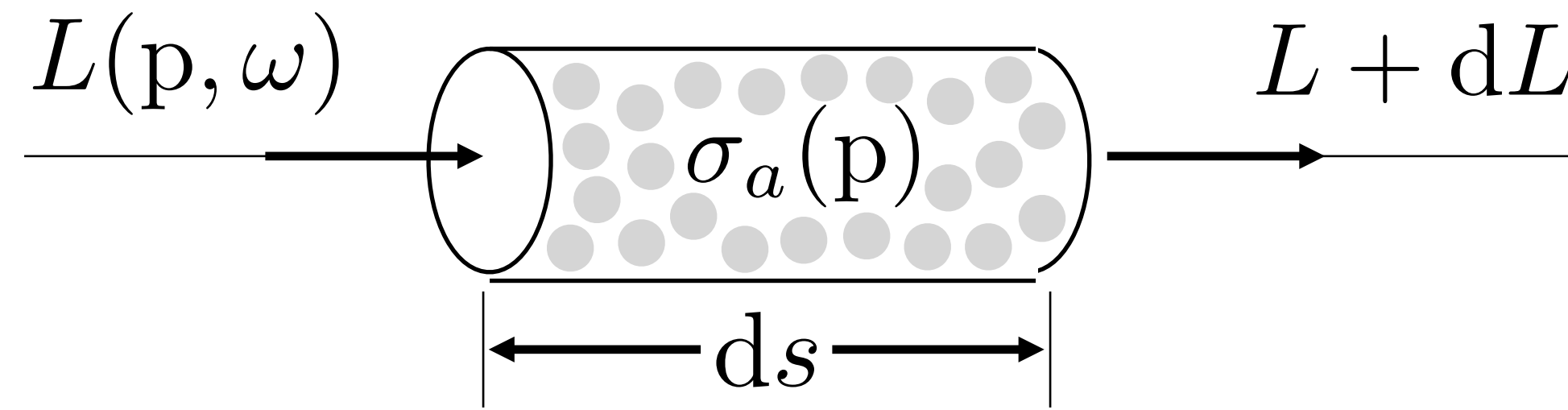
(Only if time)

**Now that you have property understanding of path tracing,
we can talk about volume rendering with more rigor**

Volumetric effects



Absorption in a volume (we saw this before)



$$\mathbf{p} = (x, y, z)$$

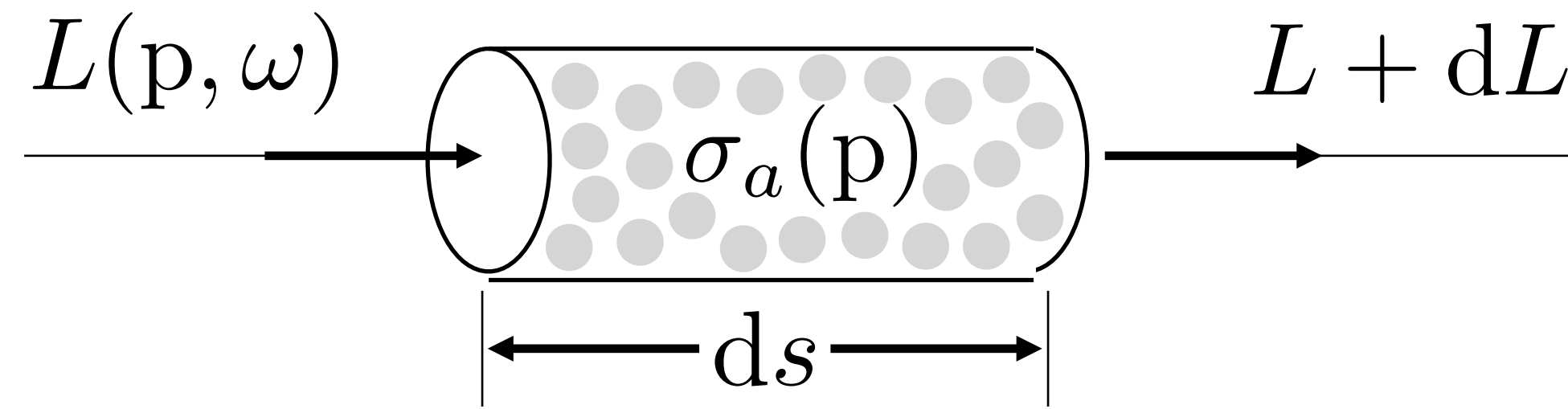
$$\omega = (\phi, \theta)$$

$$dL(p, \omega) = -\sigma_a(p) L(p, \omega) ds$$

$$\frac{dL(p, \omega)}{ds} = -\sigma_a(p) L(p, \omega)$$

- $L(p, \omega)$ radiance along a ray from p in direction ω
- Absorption cross section at point in space: $\sigma_a(p)$
 - Probability of being absorbed per unit length
 - Units: 1/distance

Absorption in a volume



$$\mathbf{p} = (x, y, z)$$

$$\omega = (\phi, \theta)$$

$$\frac{dL(p, \omega)}{L(p, \omega)} = -\sigma_a(p) ds$$

$$L(\mathbf{p} + s\omega, \omega) = e^{-\int_0^s \sigma_a(\mathbf{p} + s'\omega) ds'} L(\mathbf{p}, \omega) = T(s) L(\mathbf{p}, \omega)$$

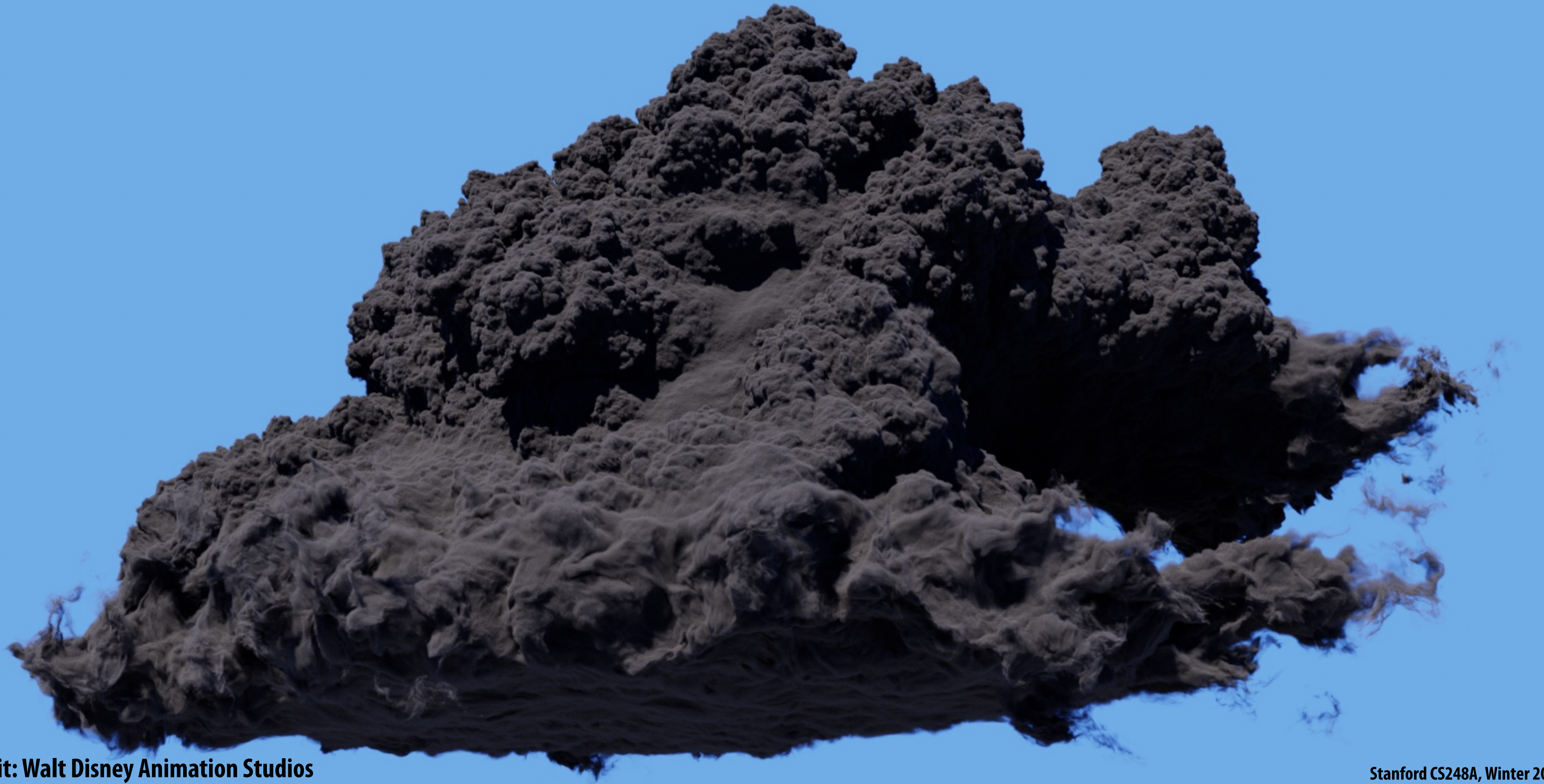
Transmittance:

$$T(s) = e^{-\int_0^s \sigma_a(\mathbf{p} + s'\omega, \omega) ds'}$$

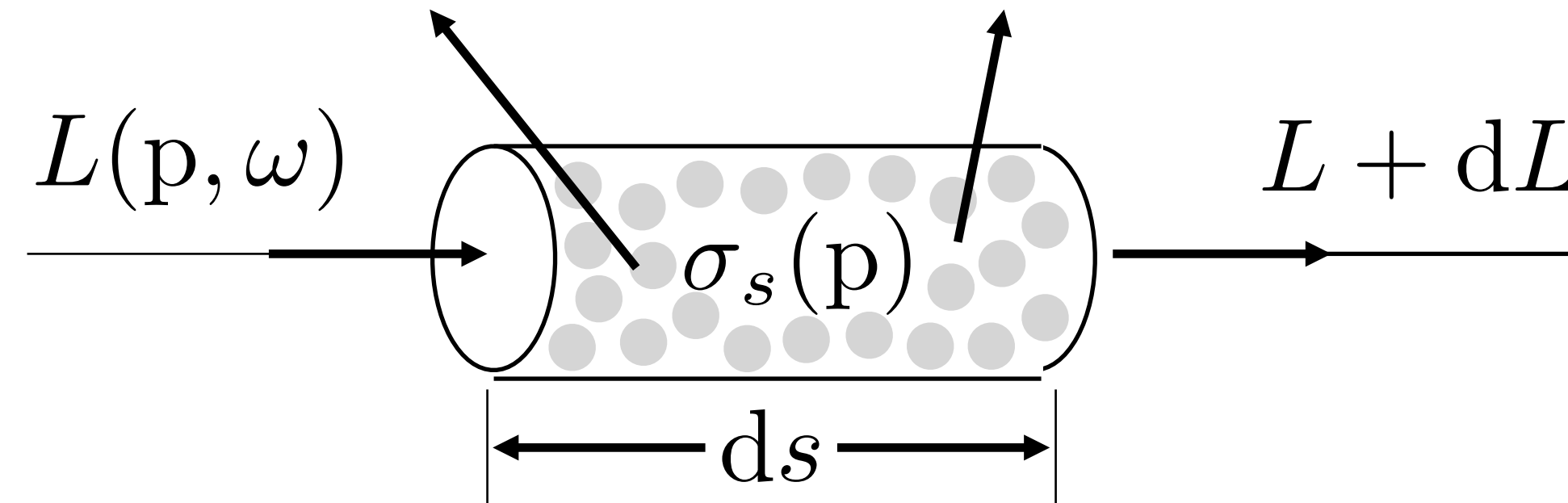
Absorption: lower density



Absorption: higher density



Out scattering



$$dL(p, \omega) = -\sigma_s(p) L(p, \omega) ds$$

- **Scattering cross section at point in space: σ_s**
 - **Probability of being scattered per unit length**
 - **Units: 1/distance**

Absorption and out scattering diminish radiance

Total cross section:

$$\sigma_t = \sigma_a + \sigma_s$$

$$dL(p, \omega) = -\sigma_t(p) L(p, \omega) ds$$

$$L(p + s\omega, \omega) = T(s) L(p, \omega)$$

Where total transmittance is:

$$T(s) = e^{-\int_0^s \sigma_t(p + s'\omega) ds'} = e^{-\tau(s)}$$

$$\tau(s) = \int_0^s \sigma_t(p + s'\omega) ds'$$

“Optical distance” (from absorption and scattering)

Ray marching to compute transmittance (you know well!)

Step through volume in small steps

Given “camera ray” from point $\mathbf{o}$ in direction $\mathbf{w}$...

$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{w}$$

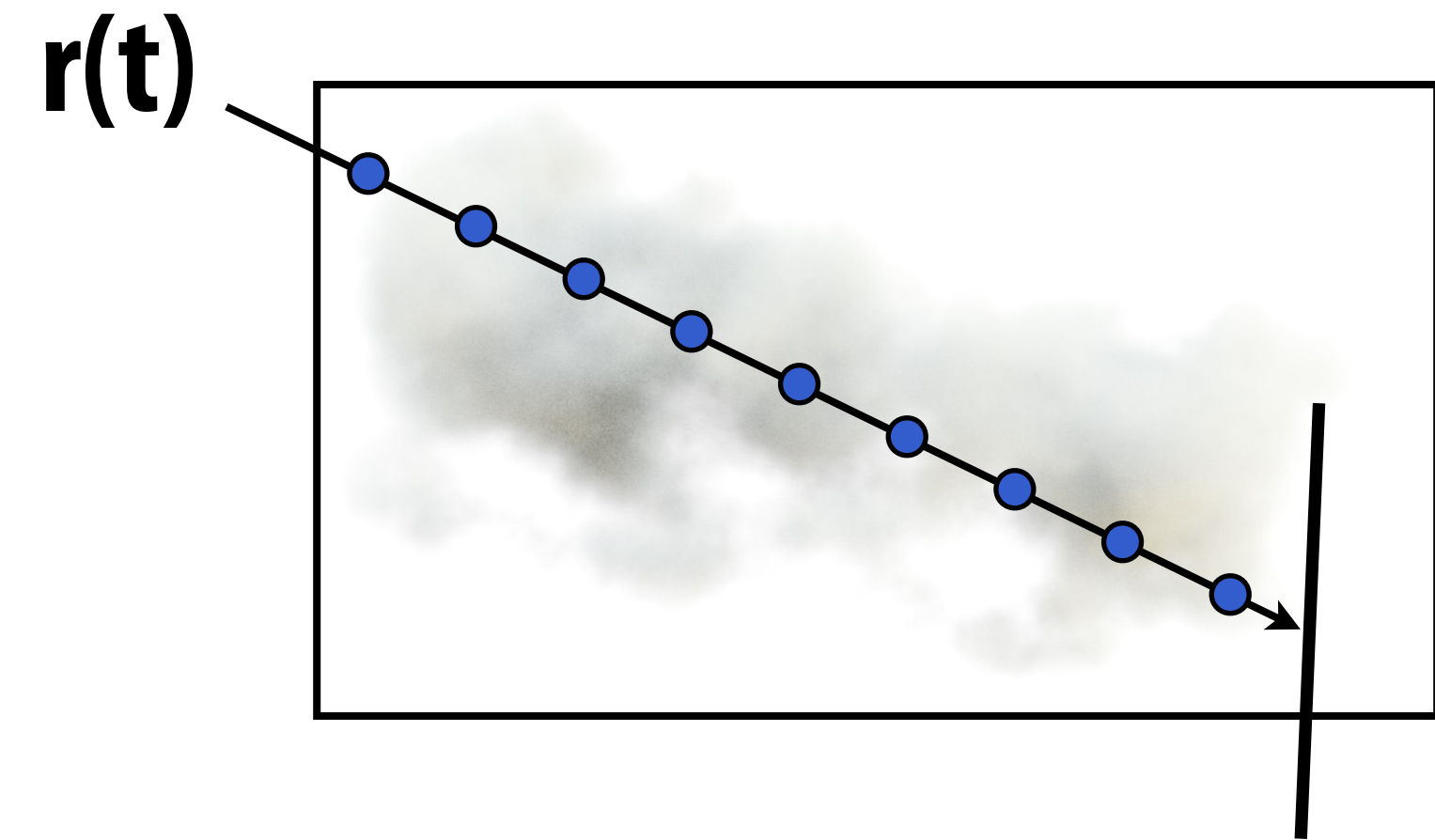
And volume with density

$$\sigma(\mathbf{p})$$

Estimate optical thickness as:

$$\tau(s) \approx \frac{s}{N} \sum_i^N \sigma_t(\mathbf{p}_i)$$

$$\mathbf{p}_i = \mathbf{o} + \frac{i + 0.5}{N} \mathbf{w}$$



Trying to approximate this integral

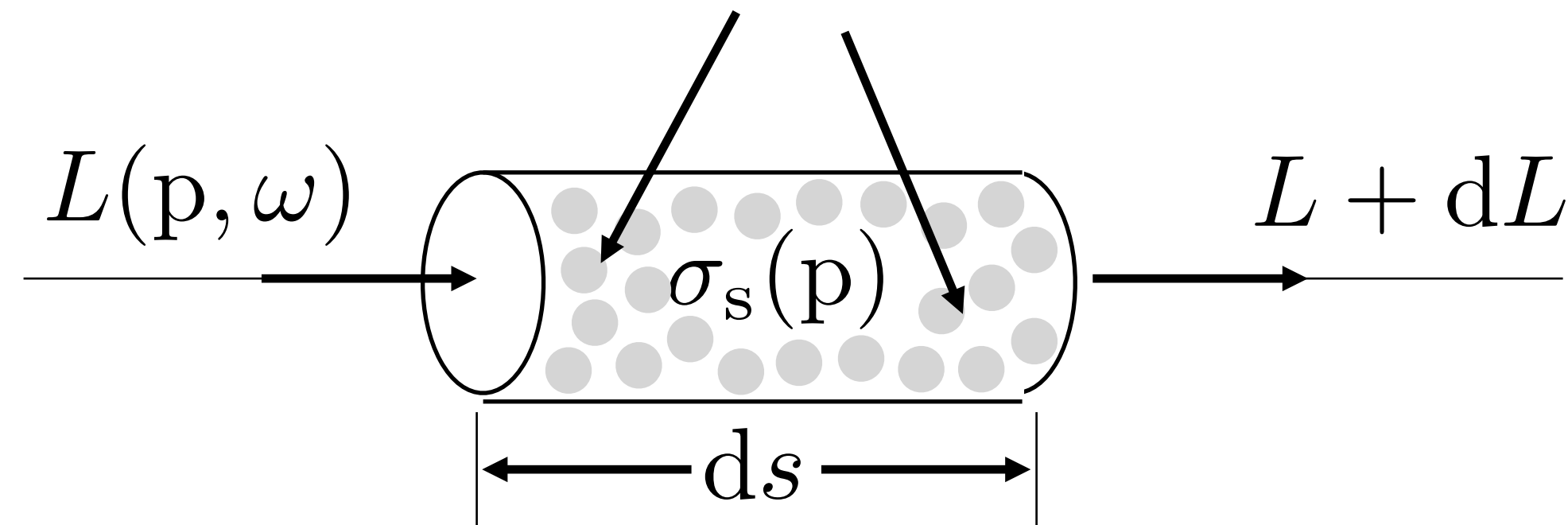
$$\tau(s) = \int_0^s \sigma_t(\mathbf{p} + s'\mathbf{w}) ds'$$

To compute:

$$T(s) = e^{-\tau(s)}$$

In scattering

- Light going in other directions also scatters into the direction ω
- In scattering increases radiance along ω



$$S(p, \omega) = \sigma_s(p) \int_{S^2} p(\omega' \rightarrow \omega) L(p, \omega') d\omega'$$

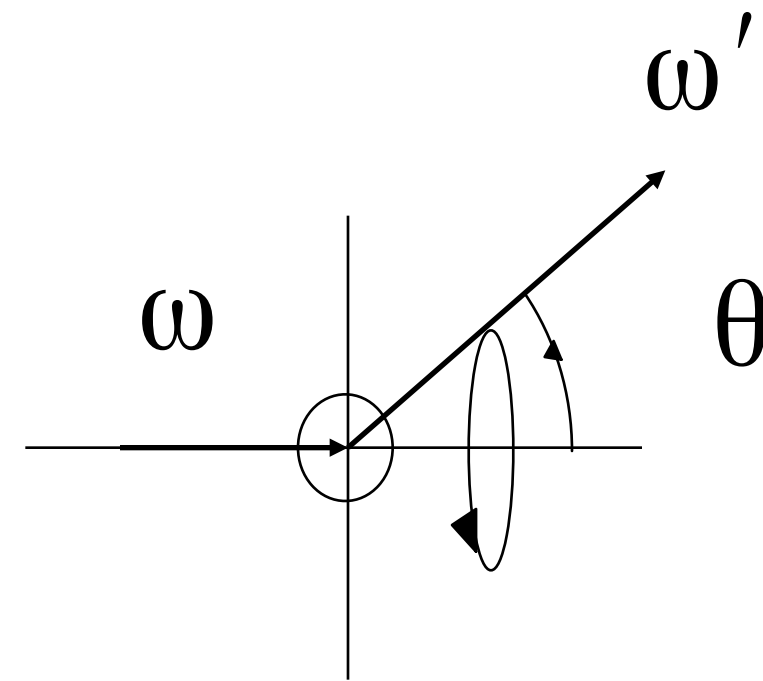
Phase function: $p(\omega' \rightarrow \omega)$

Energy conservation: $\int_{S^2} p(\omega' \rightarrow \omega) d\omega' = 1$

Phase Functions

■ Phase angle

$$\cos \theta = \omega \cdot \omega'$$



■ Phase functions

- Isotropic:

$$p(\cos \theta) = \frac{1}{4\pi}$$

- Rayleigh:

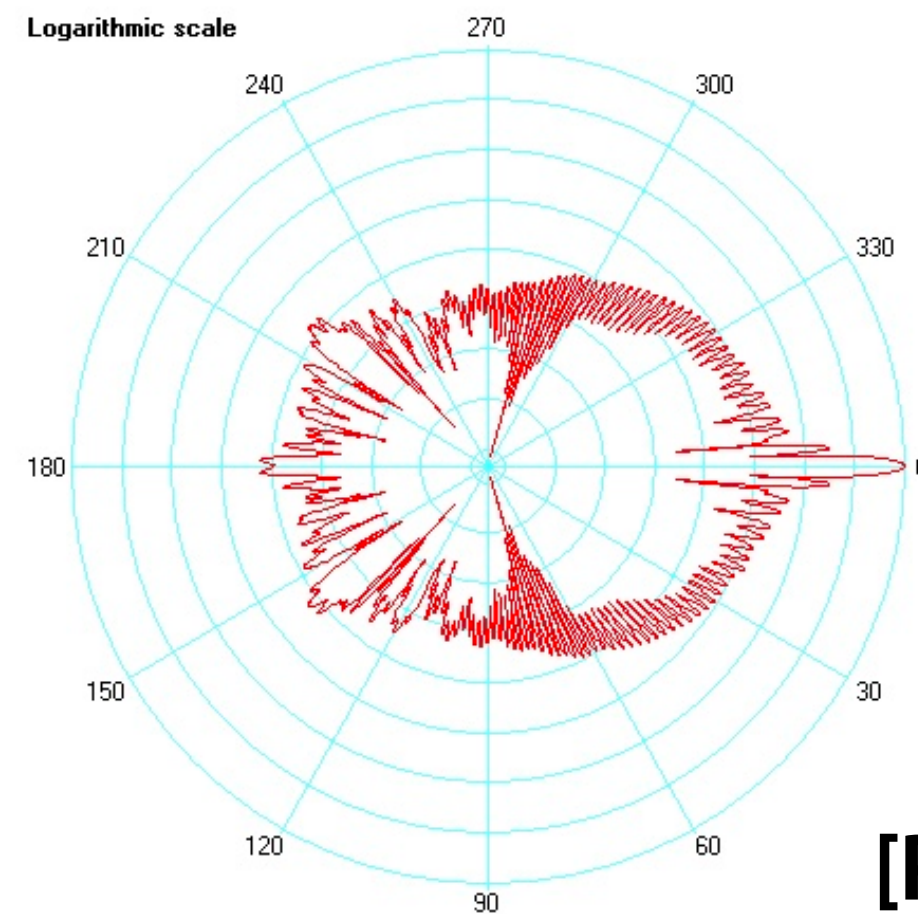
$$p(\cos \theta) = \frac{3}{4}(1 + \cos^2 \theta)$$

with

$$\sigma_s \propto \frac{1}{\lambda^4}$$

Note: probability of scattering is function of wavelength

- Mie:



[Philip Laven]

Rayleigh Scattering: Blue Sky, Red Sunset



From Greenler: Rainbows, Halos, and Glories

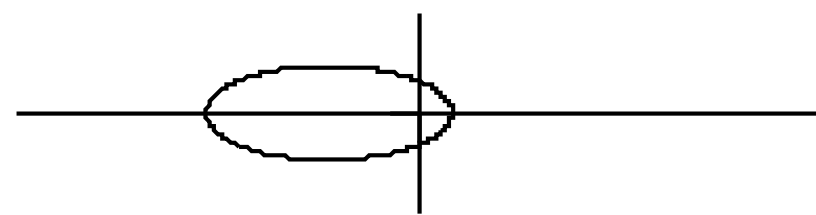
Henyey-Greenstein Phase Function

- Empirical phase function

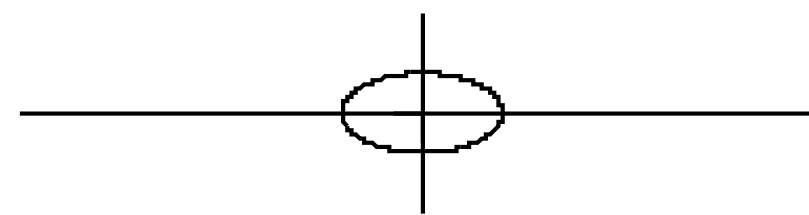
$$p(\cos \theta) = \frac{1}{4\pi} \frac{1 - g^2}{(1 + g^2 - 2g \cos \theta)^{3/2}}$$

- Average phase angle g :

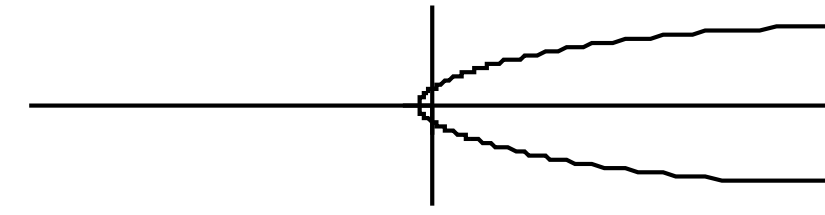
$$g = 2\pi \int_0^{2\pi} p(\cos \theta) \cos \theta \sin \theta \, d\theta$$



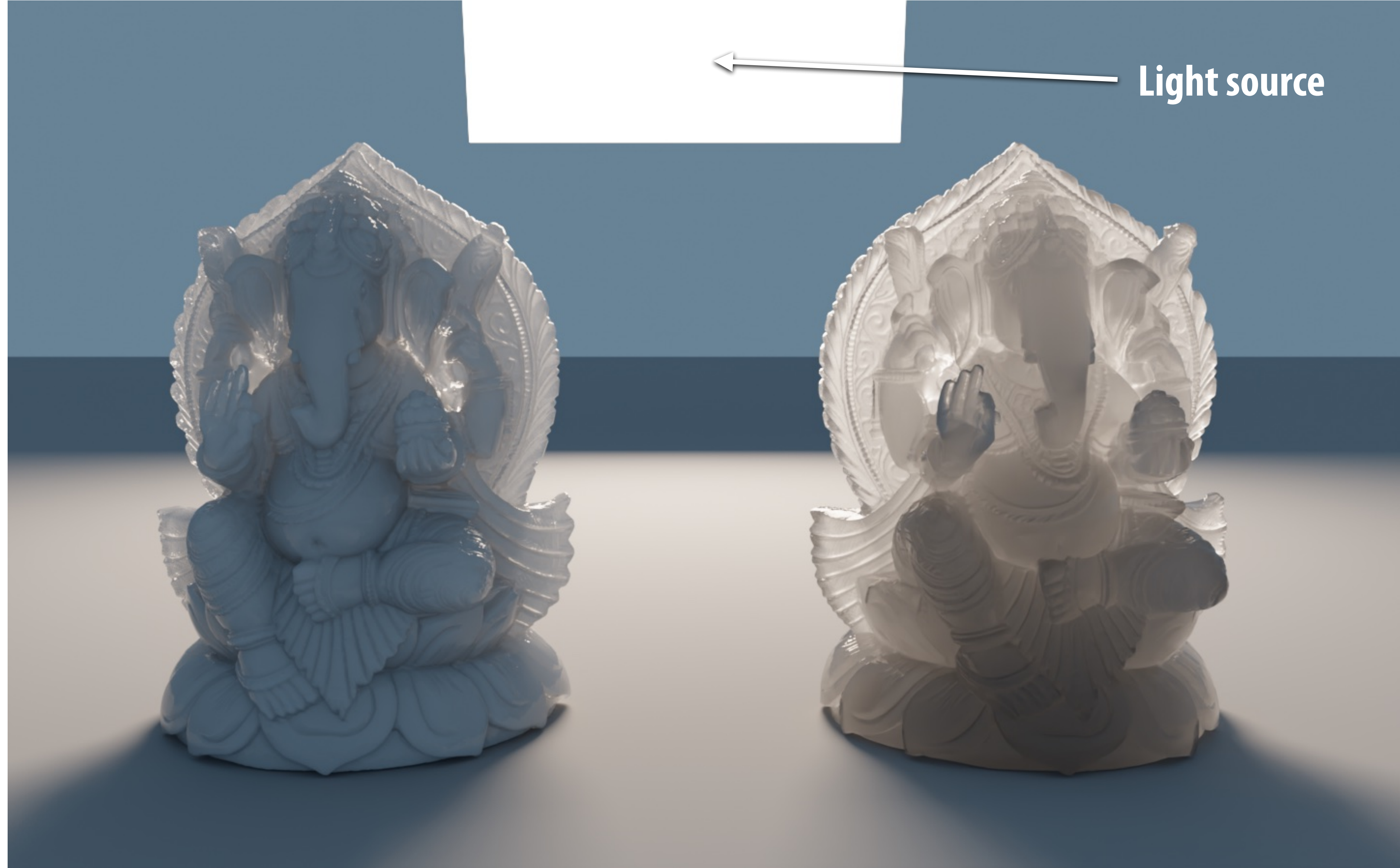
$$g = -0.3$$



$$g = 0$$



$$g = 0.6$$



**More Backward
Scattering**

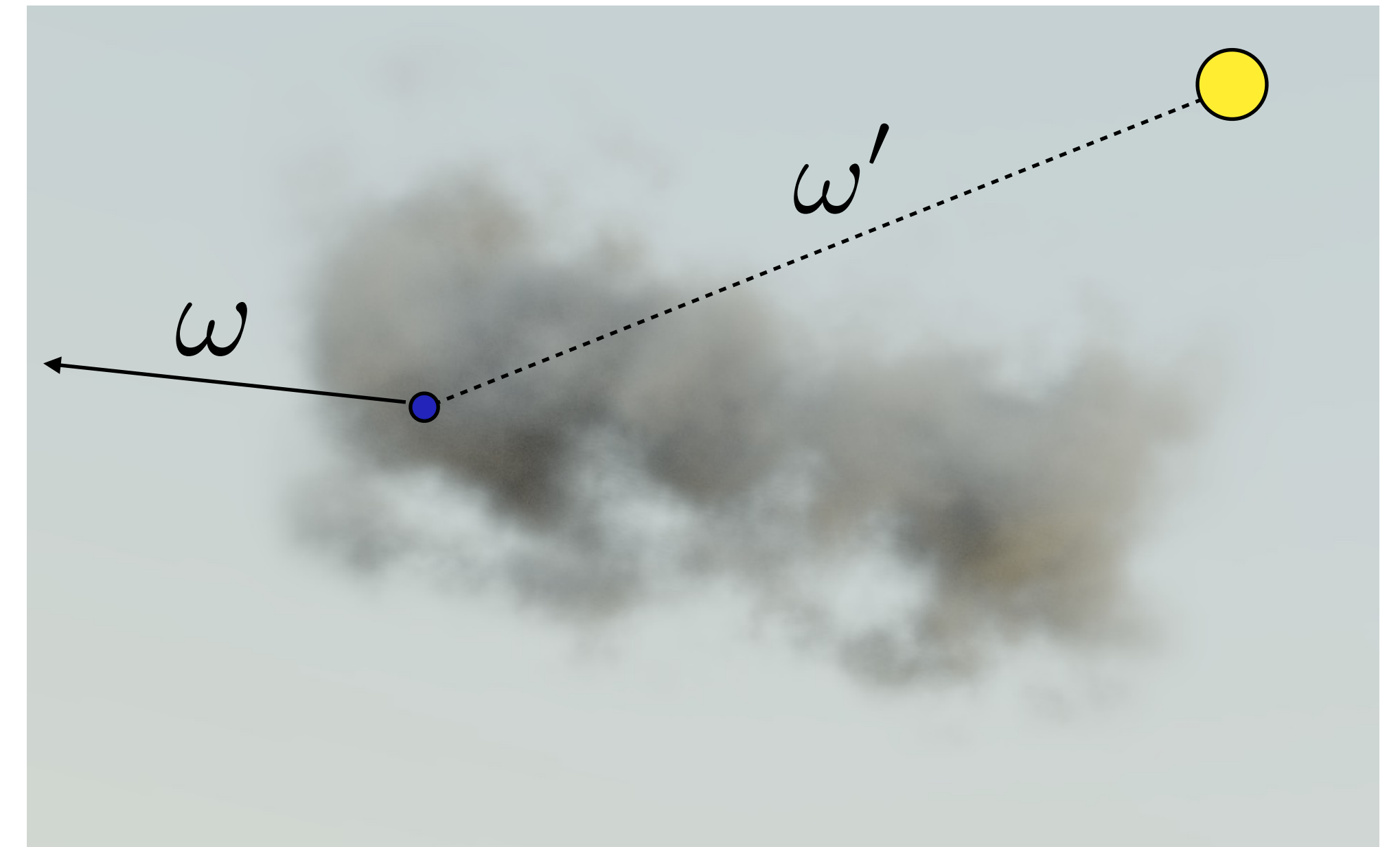
**More Forward
Scattering**

Direct illumination in a volume

- Can treat like direct illumination on a surface
- e.g., sample light sources (or from phase function's distribution)

$$S_d(p', \omega) = \sigma_s(p') \int_{S^2} p(\omega' \rightarrow \omega) L_d(p', \omega') d\omega'$$

- But computing direct lighting is now much more expensive
- Why? (Hint: requires more than a shadow ray)

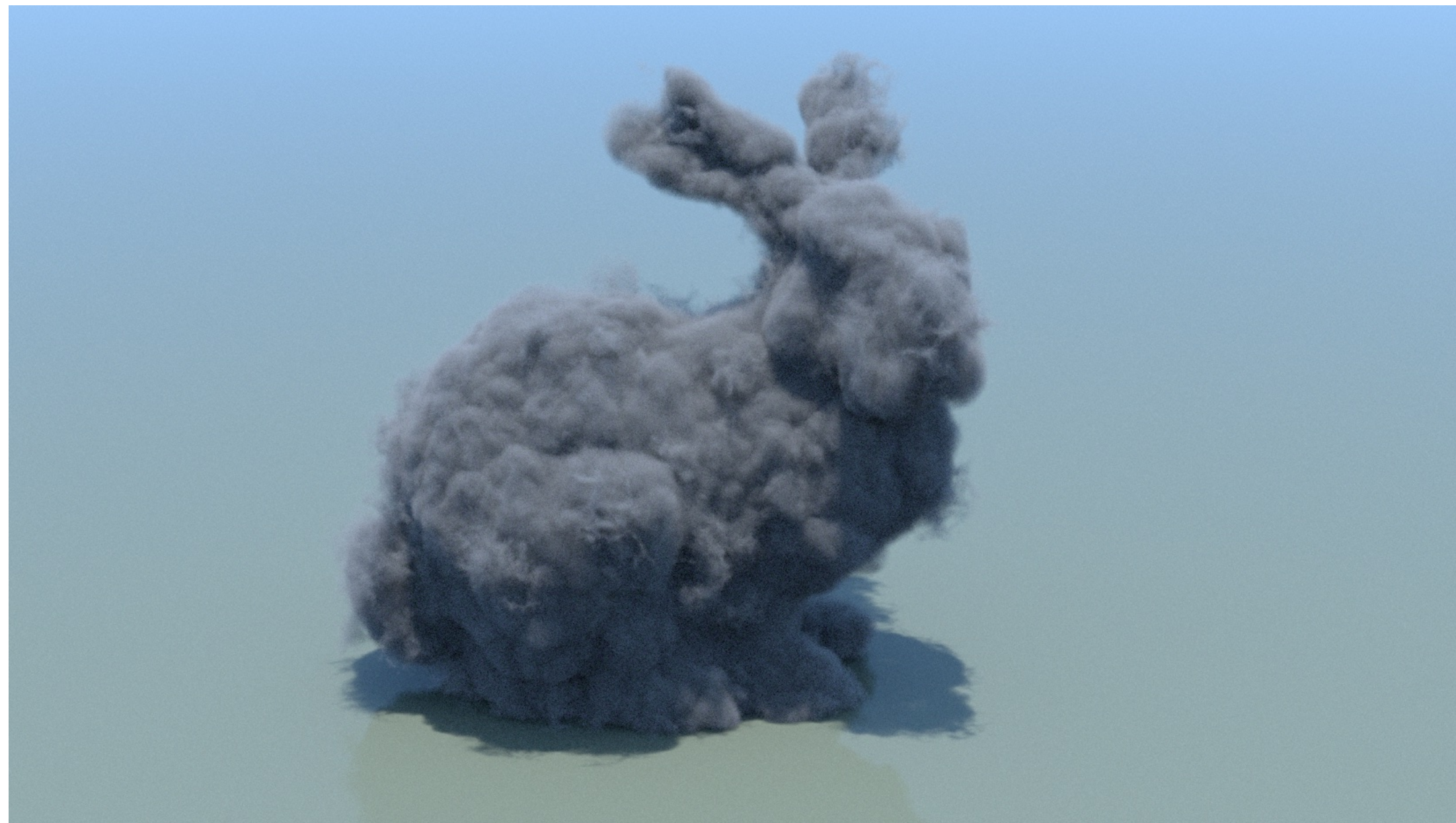


Single scattering

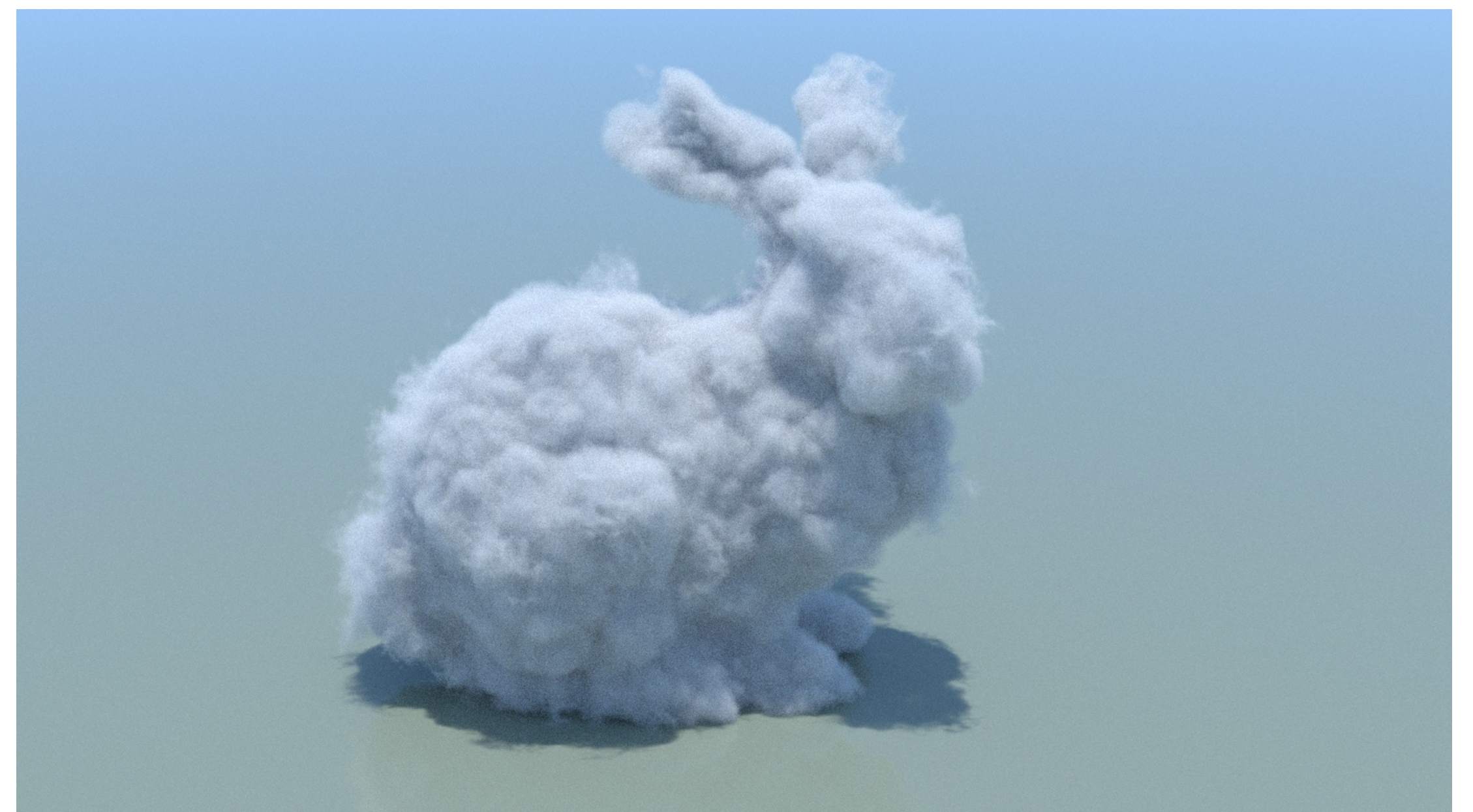


Multiple scattering (not discussed today)

- Appearance of a volume is not just due to a single scattering event (light directly from a light source scattering in the direction of eye)
- Light scatters many times in the volume before existing in the direction of the eye, so need to account for all these light paths
- Advanced rendering topic: Monte Carlo estimate of multiple scattering events (“volume rendering equation”)

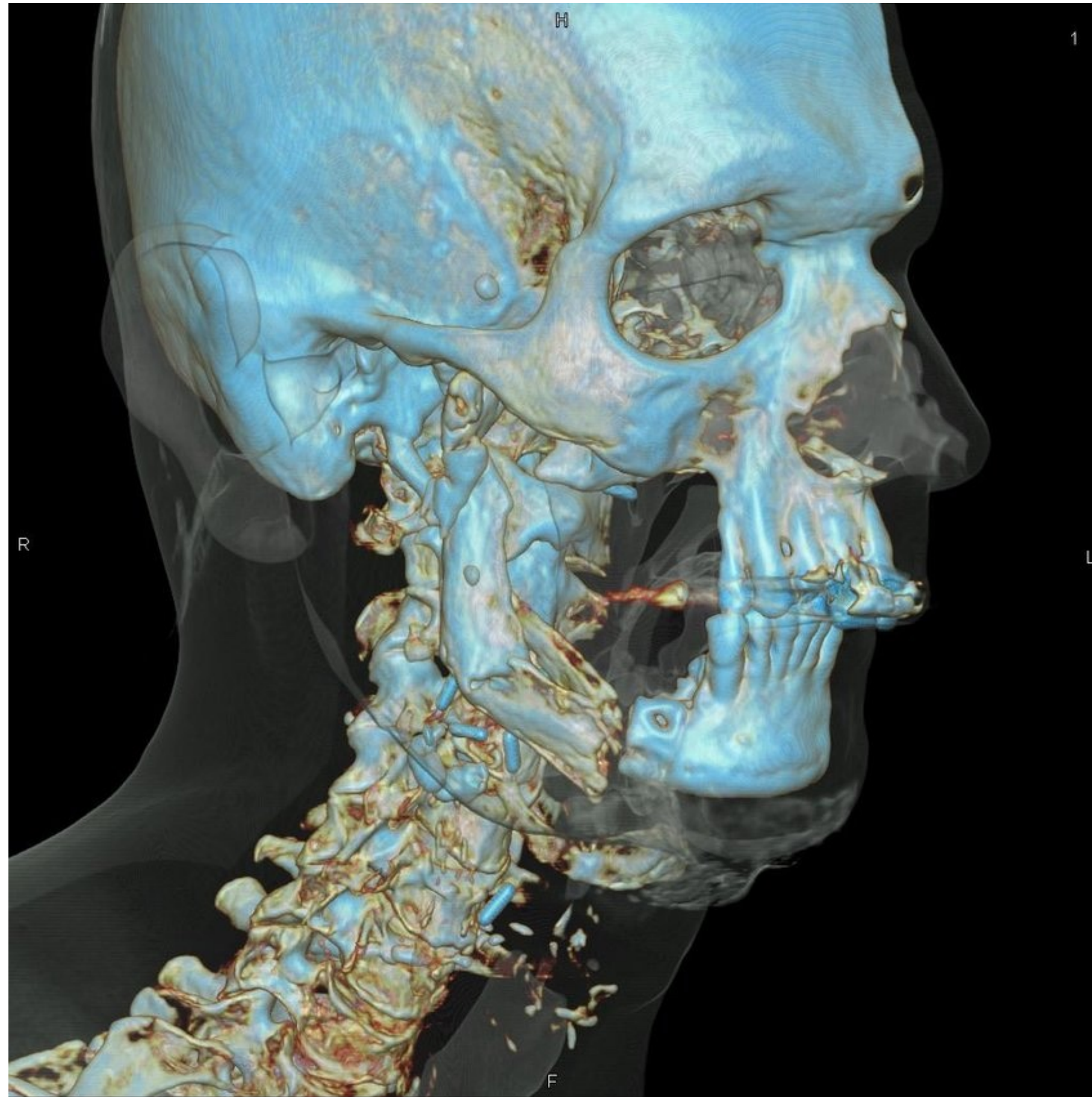


Single scattering



Multiple scattering

Your assignment 2 “volume renderer” assumes that lighting is “baked into” the volume representation. You are not simulating illumination of a volume.



Volume rendered CT scan

Volume density and “color” at all points in space.

$\sigma(p)$

$c(p, \omega) = c(x, y, z, \phi, \theta)$ The reflectance off surface at point p in direction ω



**Volume rendered scene
(Mildenhall et al.)**