Path Tracing I

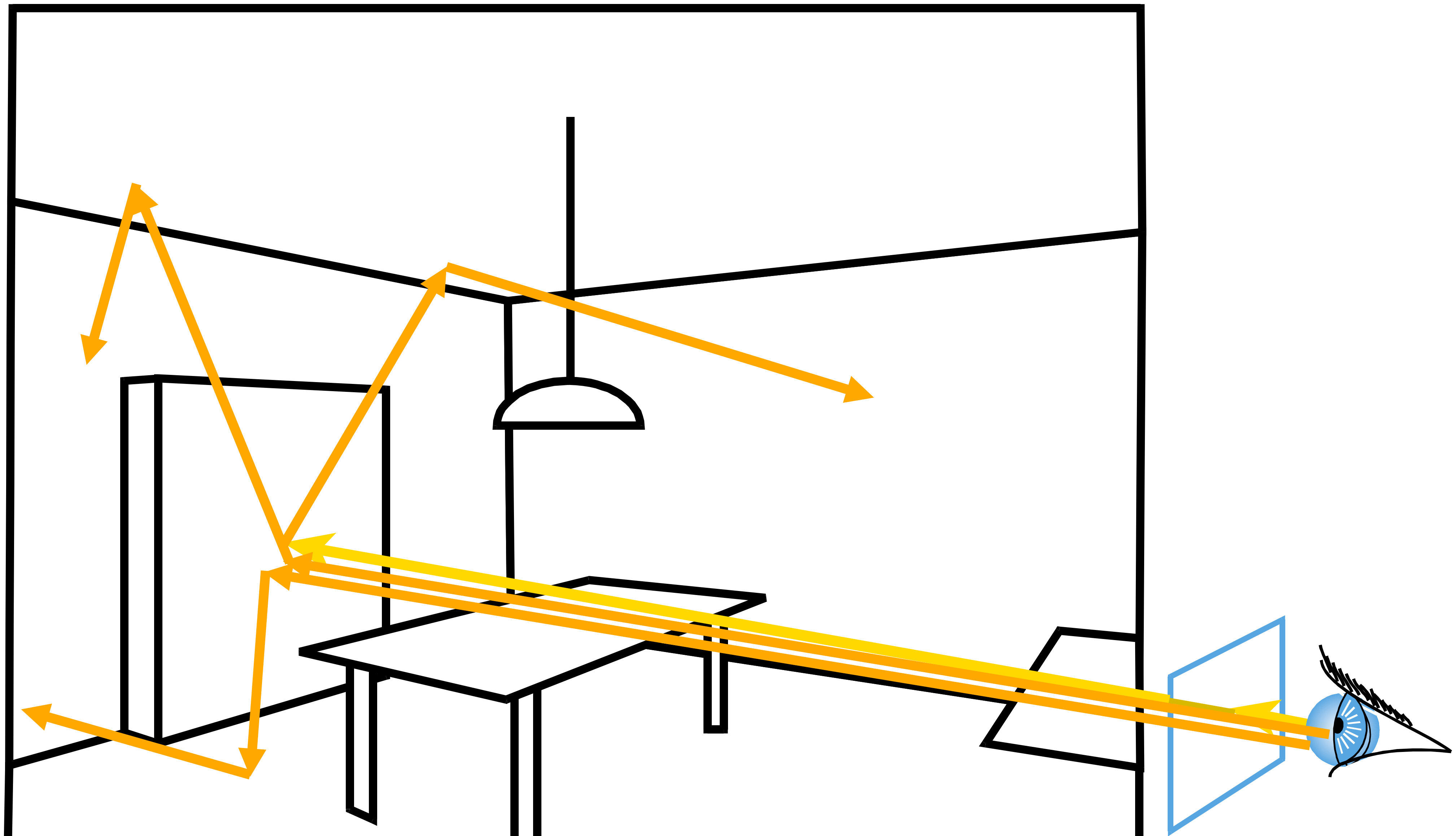
Bridage 2 real-time path tracer

Aalto CS-E5520 Spring 2023
Jaakko Lehtinen

Monte Carlo Path Tracing

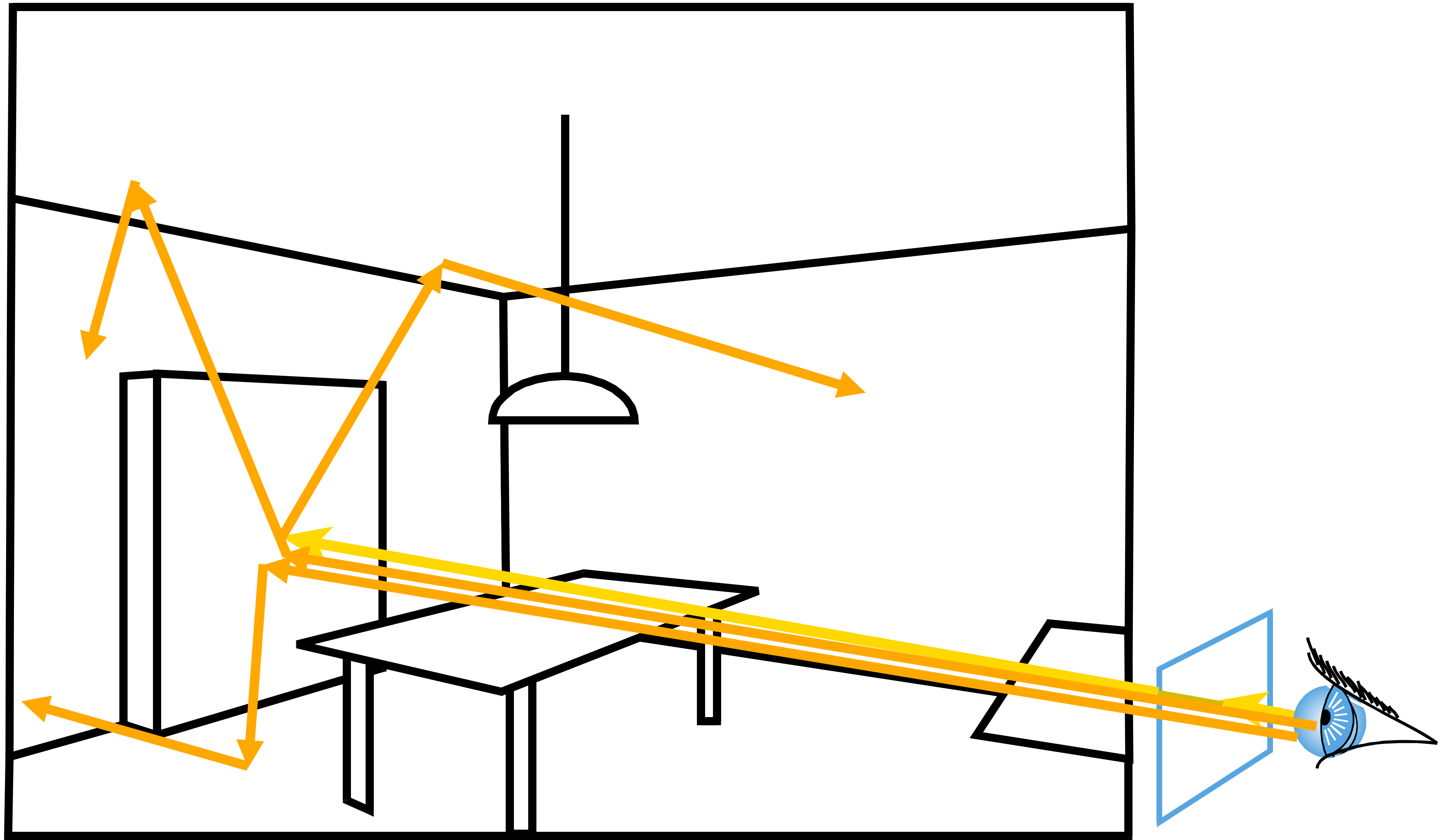
- Recursively estimate the rendering equation

$$L_{\text{out}}(x, \mathbf{v}) = \int_{\Omega} L_{\text{in}}(x, \mathbf{l})\, f_r(x, \mathbf{l} \to \mathbf{v})\, \cos\theta_{\text{in}}\, \text{d}\mathbf{l} + E_{\text{out}}(x, \mathbf{v})$$



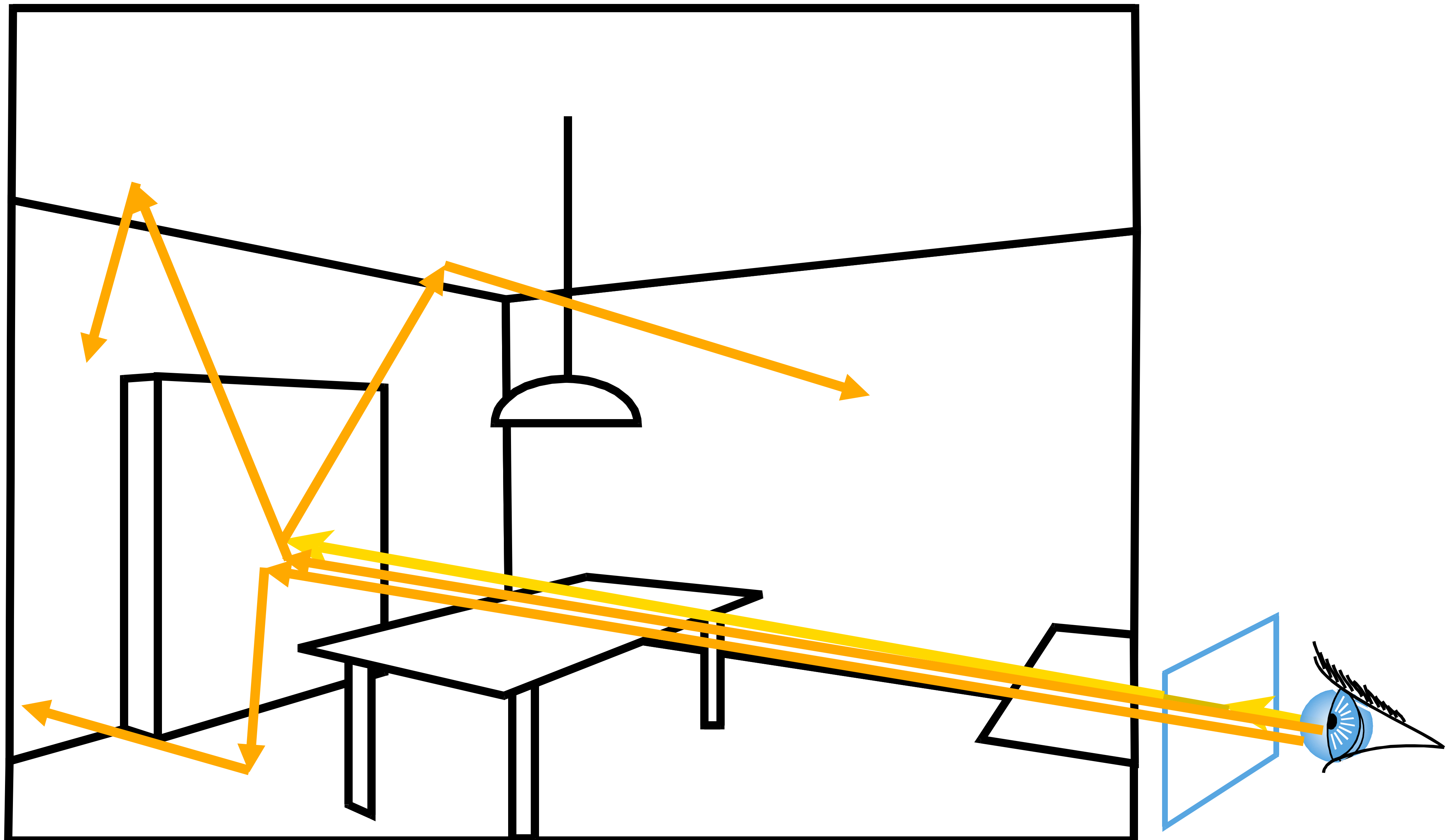
Monte Carlo Path Tracing

- Trace only one secondary ray per recursion
  - Otherwise number of rays explodes!
- But send many primary rays per pixel (antialiasing)



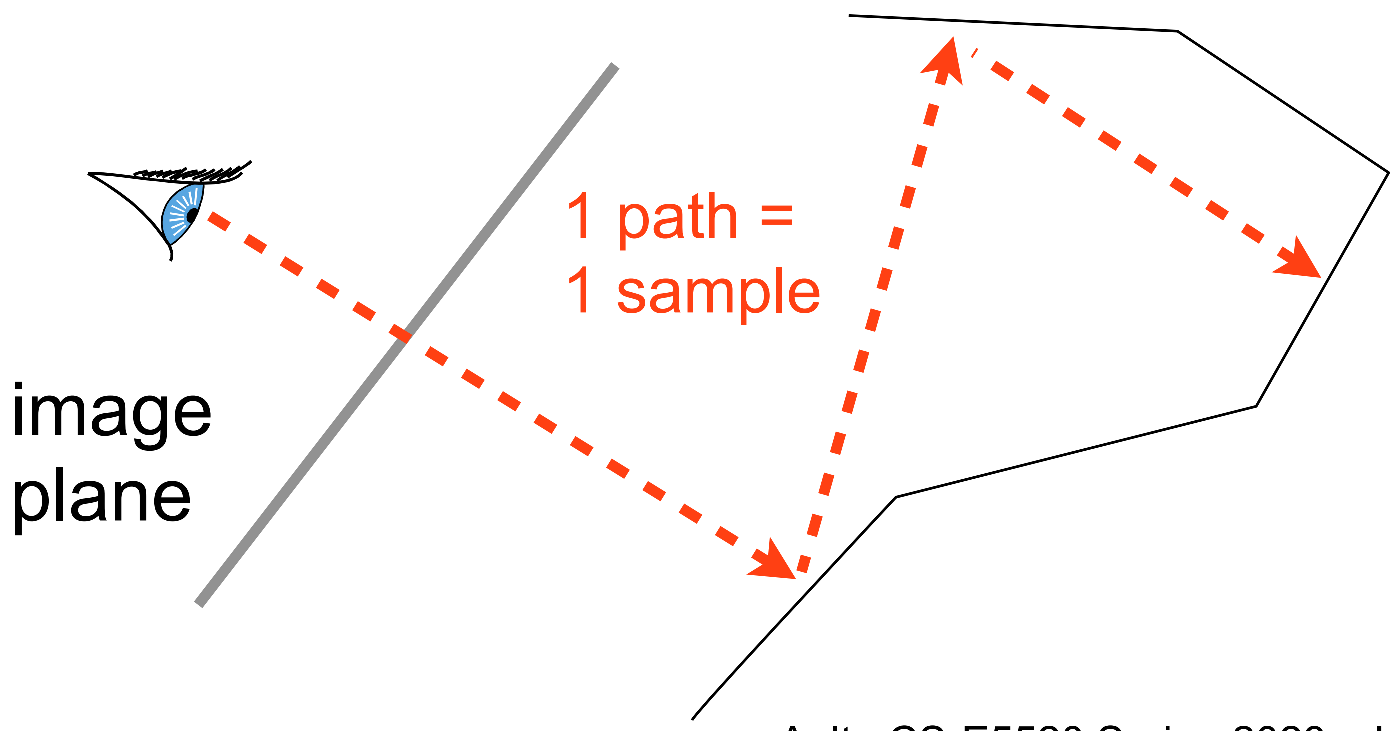
Monte Carlo Path Tracing

- Trace only one secondary ray per recursion
  - Otherwise number of rays explodes!
- But send many primary rays per pixel (antialiasing)



Monte Carlo Path Tracing

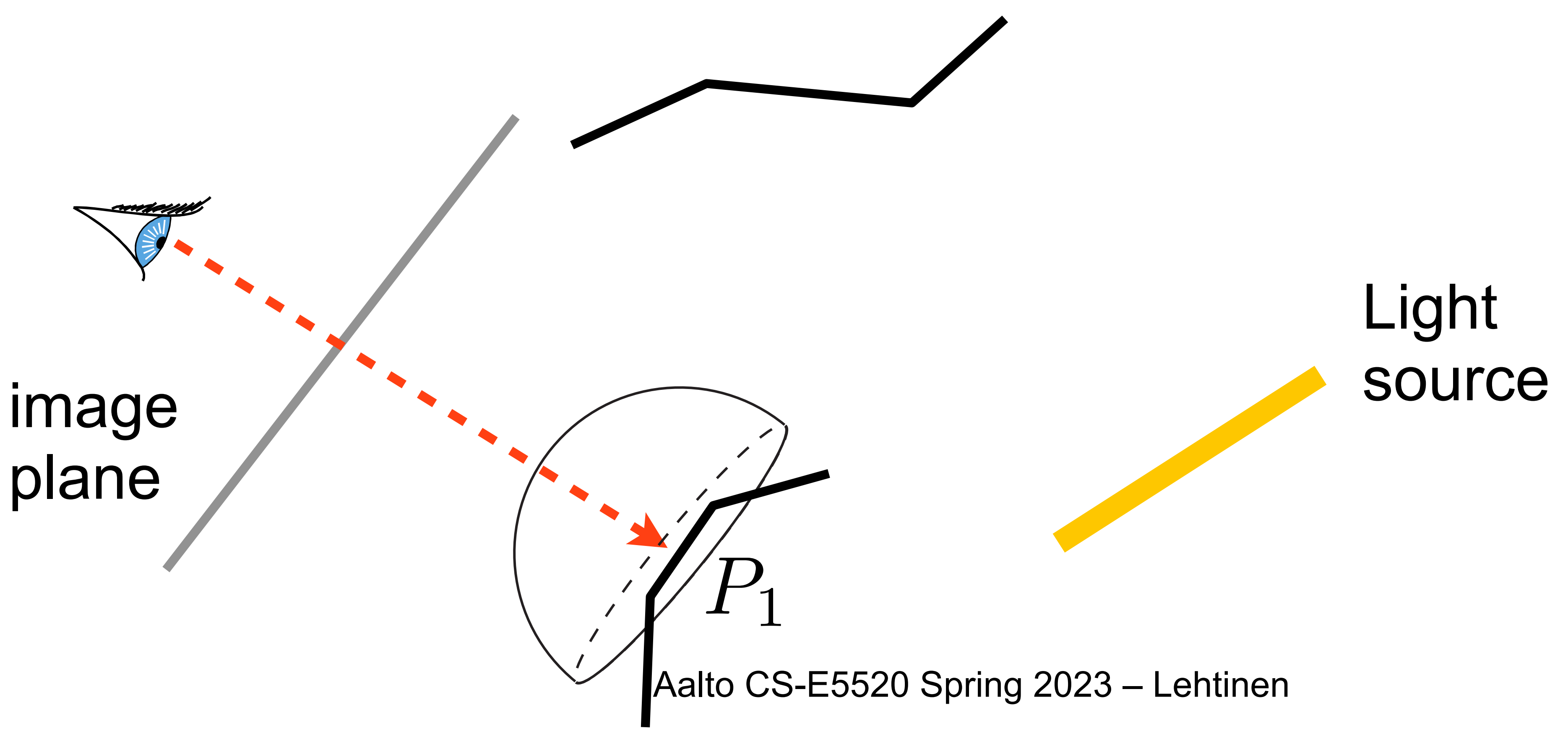
- The idea is just the same as before with AO+filter
  - Instead of thinking about nested integrals over hemispheres at each bounce, let's think of one integral over the Cartesian product of all the hemispheres
  - For n bounces, the domain is $\text{screen} \times \underbrace{\Omega \times \ldots \times \Omega}_{n \text{times}}$
  - Each sample is a *path = sequence of rays*



Example: 1 Indirect Bounce

(Without pixel filter, for clarity!)

- What is the radiance leaving P_1 towards the eye after it has taken precisely one bounce off other surfaces after leaving the light source?

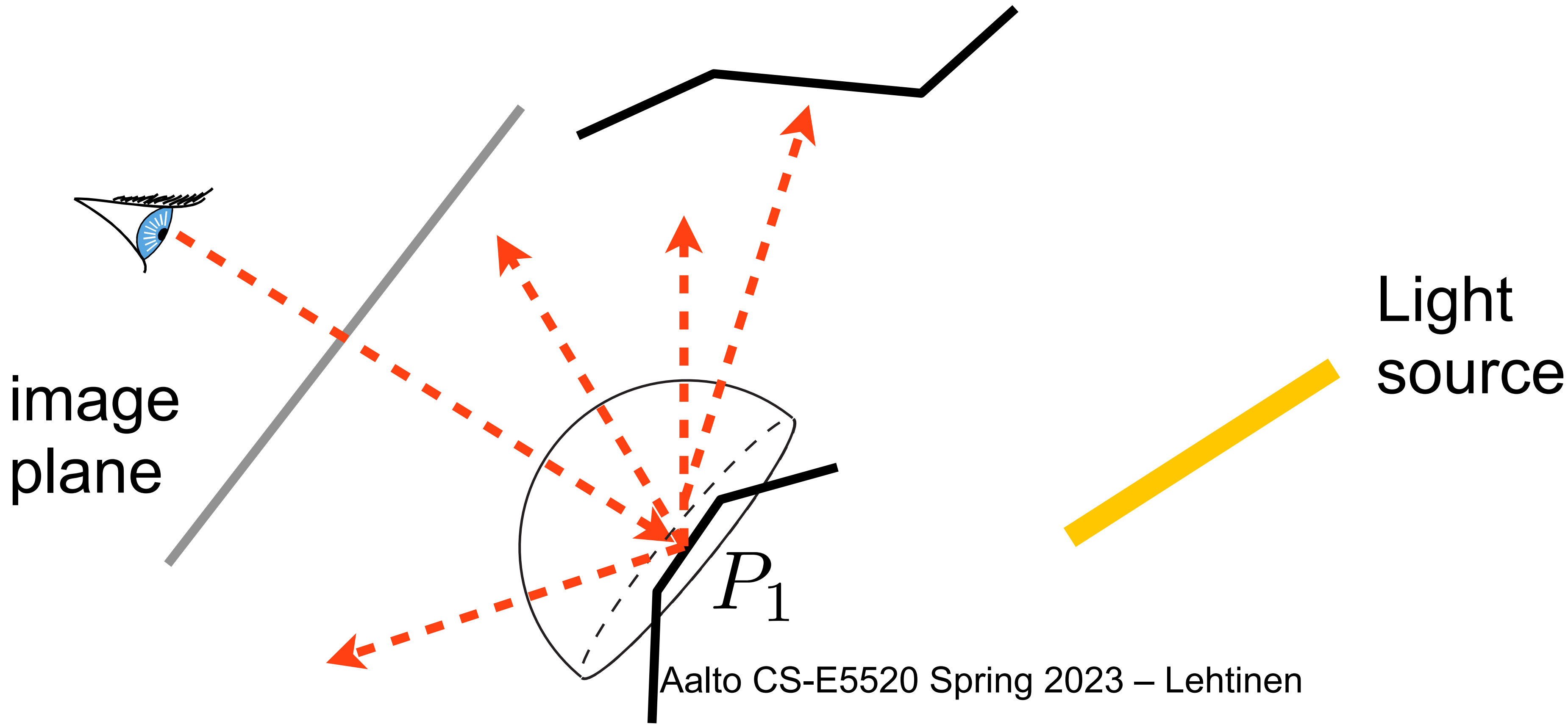


Example: 1 Indirect Bounce

(Without pixel filter, for clarity!)

- Nested version (P_1, P_2 are ray hit points)

$$L_2(x,y) = \int_{\Omega(P_1)} L(P_1 \leftarrow \omega_1)\, f_r(P_1,\, \omega_1 \rightarrow \text{eye})\, \cos\theta_1 \, \mathrm{d}\omega_1$$

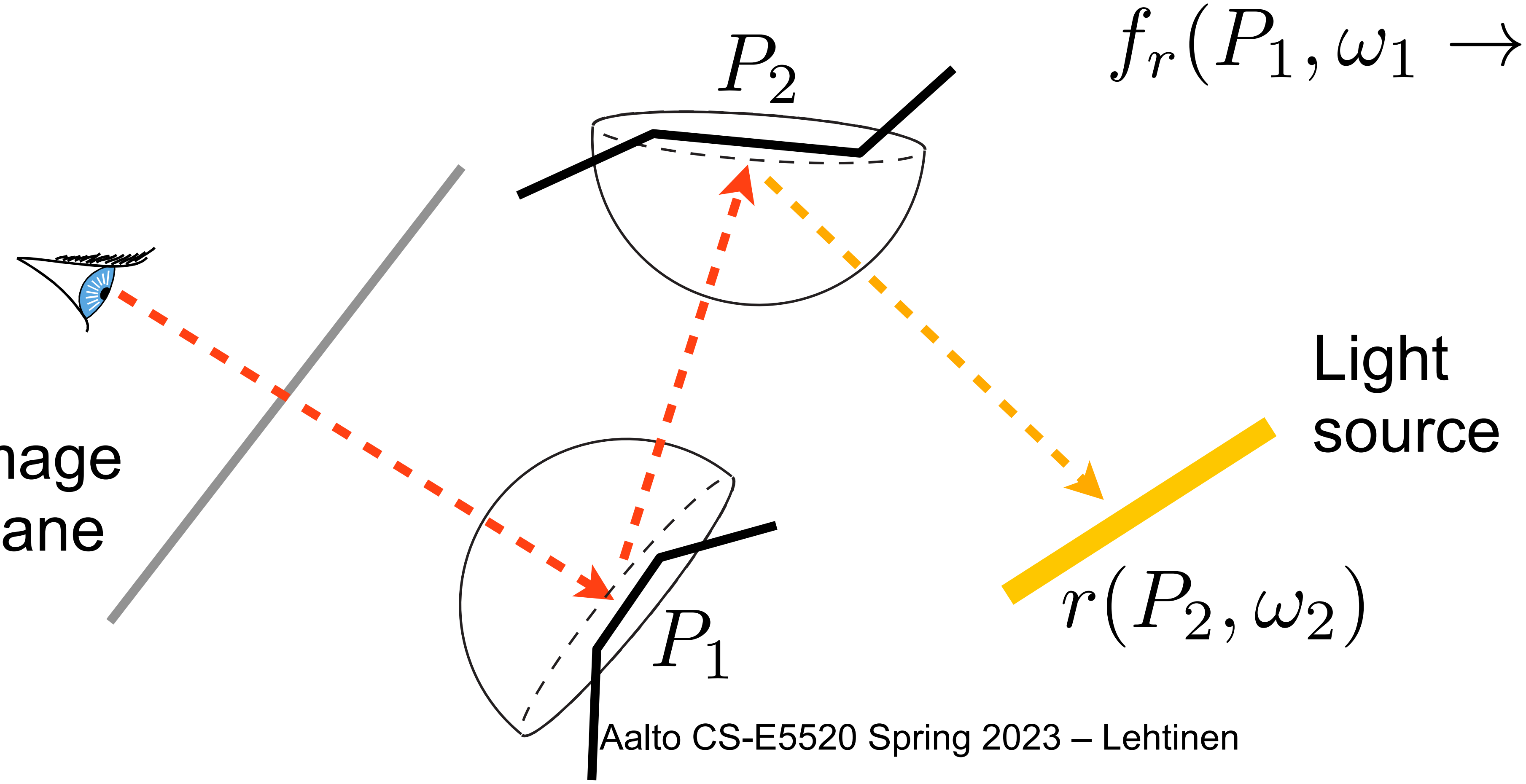


Example: 1 Indirect Bounce

(Without pixel filter, for clarity!)

- Nested version (P_1, P_2 are ray hit points)

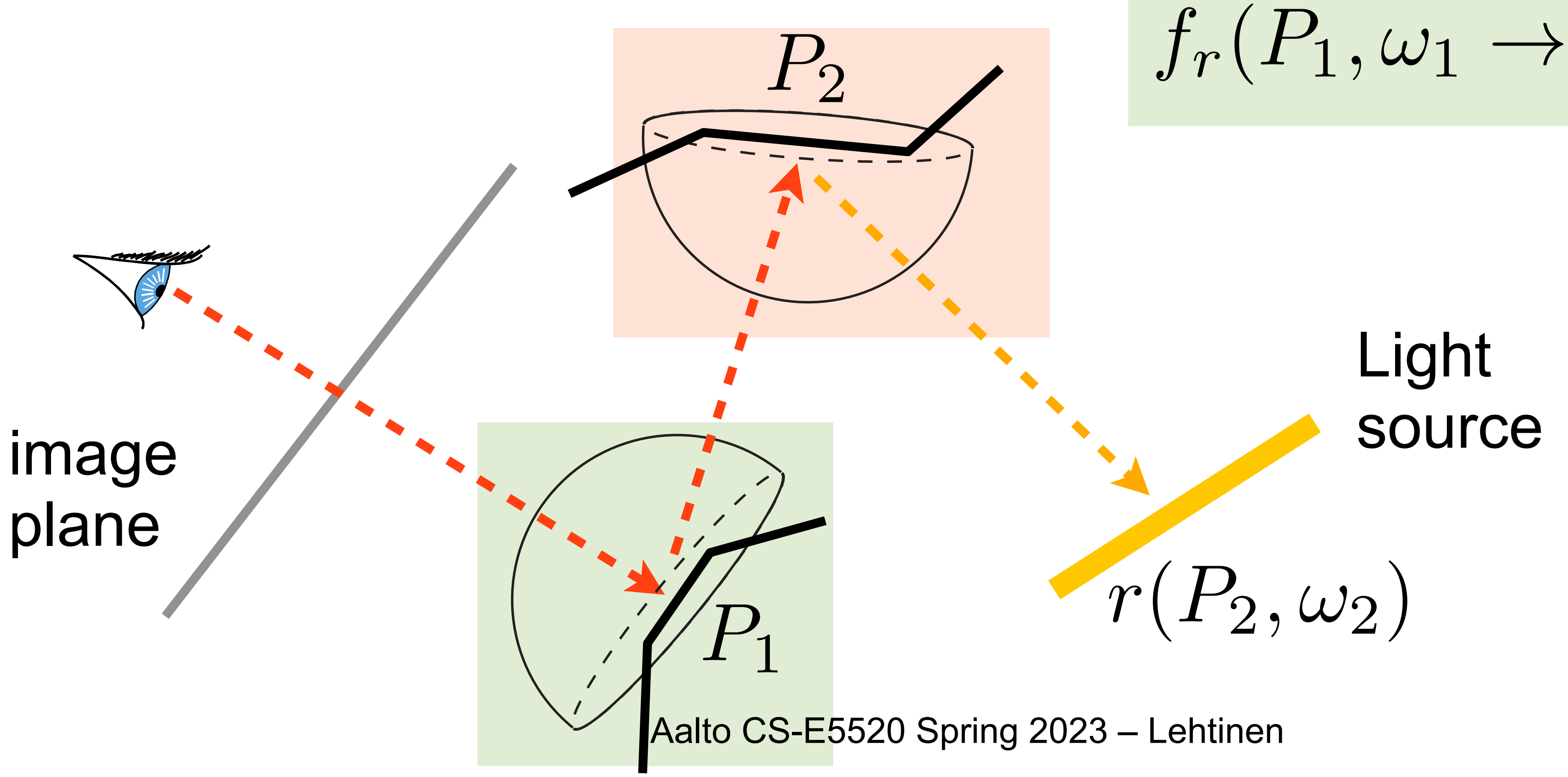
$$L_2(x,y) = \int_{\Omega(P_1)} \overbrace{\left[\int_{\Omega(P_2)} E(r(P_2,\omega_2) \to P_2) f_r(P_2, \omega_2 \to -\omega_1) \cos\theta_2 \mathrm{d}\omega_2\right]}^{L(P_1 \leftarrow \omega_1)} f_r(P_1, \omega_1 \to \text{eye}) \cos\theta_1 \mathrm{d}\omega_1$$



Example: 1 Indirect Bounce

- Nested version (P_1, P_2 are ray hit points)

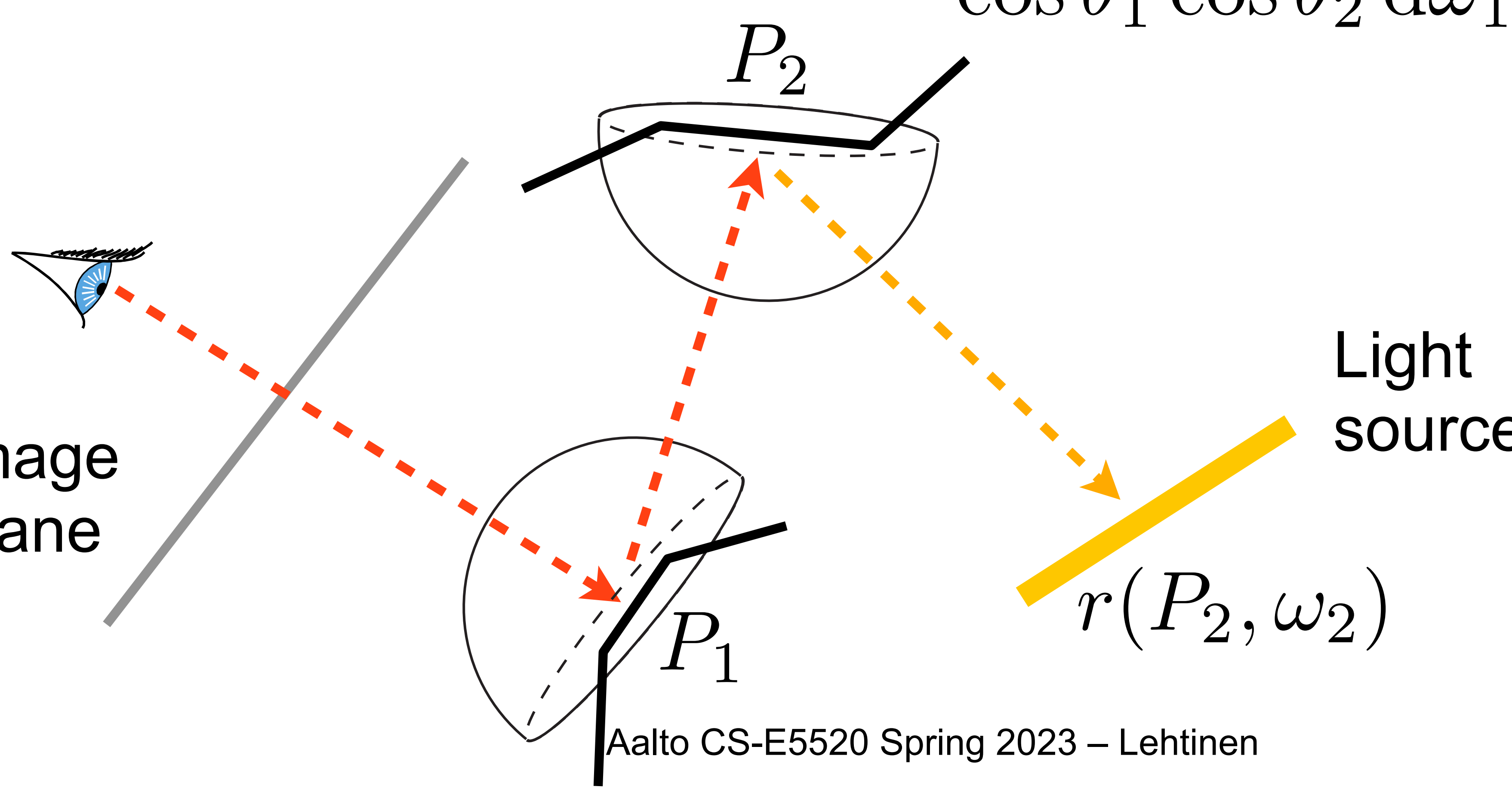
$$L_2(x,y) = \int_{\Omega(P_1)} \overbrace{\left[\int_{\Omega(P_2)} E(r(P_2,\omega_2) \rightarrow P_2) f_r(P_2,\omega_2 \rightarrow -\omega_1) \cos\theta_2 \mathrm{d}\omega_2 \right]}^{L(P_1 \leftarrow \omega_1)} f_r(P_1,\omega_1 \rightarrow \mathrm{eye}) \cos\theta_1 \mathrm{d}\omega_1$$



Example: 1 Indirect Bounce $P_2 = r(P_1, \omega_1)$

- Flat version, 4D integral

$$L_2(x,y) = \int_{\Omega(P_1)\times\Omega(P_2)} E(r(P_2,\omega_2) \to P_2) \times f_r(P_2, \omega_2 \to -\omega_1) f_r(P_1, \omega_1 \to \text{eye}) \times \cos\theta_1 \cos\theta_2 \, \mathrm{d}\omega_1 \mathrm{d}\omega_2$$



This really is just as simple as going from two nested 1D integrals to a 2D area integral!

Full Solution

- The full solution is a sum over paths of all lengths

$$L(x,y) = \sum_{i=0}^{\infty} L_i(x,y), \qquad \text{with } L_0(x,y) = E(P_1 \leftarrow \text{eye})$$

- Notice how we've "unwrapped" the recursive rendering equation into a sum of terms
  - n bounce lighting is an integral over $\text{screen} \times \underbrace{\Omega \times \ldots \times \Omega}_{n \text{times}}$
  - This is the same as directly evaluating the terms of the Neumann series E + TE + TTE + ...

Sampling Paths

- "Local path sampling" proceeds bounce to bounce, always importance sampling according to local BRDF
- That is, for each sample (path):
  - First sample screen (x, y), then trace ray
  - At primary hit, choose outgoing direction ω_1 , trace ray
  - At secondary hit, choose outgoing direction ω_2
  - Apply local PDFs at each step.. justification below

- Denote the full path $\bar{x} = (x, y, \omega_1, \omega_2, \ldots)$
  - Then $p(\bar{x}) = p(x, y)\, p(\omega_1)\, p(\omega_2) \ldots$
  - (This assumes independent choices at each bounce)
  - Easy to implement

Brute Force Path Tracing, Eye Part

$$L(x \rightarrow \mathbf{v}) = \int_{\Omega} L(x \leftarrow \mathbf{l})\, f_r(x, \mathbf{l} \rightarrow \mathbf{v})\, \cos\theta \, \mathrm{d}\mathbf{l}$$

$$+ \; E(x \rightarrow \mathbf{v})$$

```
for each pixel
 Lout = 0, w=0
 for i=1 to #samples
  generate xi,yi inside pixel with p(x,y)
  ray_i = generatecameraray(xi,yi)
  Lout += f(xi,yi) * trace(ray_i)/p(x,y)
  w += f(xi,yi)/p(x,y)
 endfor
 L(pixel) = Lout/w
endfor
```

(Assuming, for simplicity, that only one pixel filter is nonzero. Look back a few slides for full treatment.)

Brute Force Path Tracing

$$L(x \to \mathbf{v}) = \int_\Omega L(x \leftarrow \mathbf{l})\, f_r(x, \mathbf{l} \to \mathbf{v})\, \cos\theta\, \mathrm{d}\mathbf{l}$$

$$+ \; E(x \to \mathbf{v})$$

```
trace(ray)
 hit = intersect(scene, ray)
 result = emission(hit,-dir(ray))  // 0 if no light
 // sample outgoing direction
 [w,pdf] = sampleReflection(hit,dir(ray))
 // recursively estimate incoming radiance, apply BRDF
 result += BRDF(hit,-dir(ray),w)*
           cos(theta)*
           trace(ray(hit,w))/pdf
 return result
```

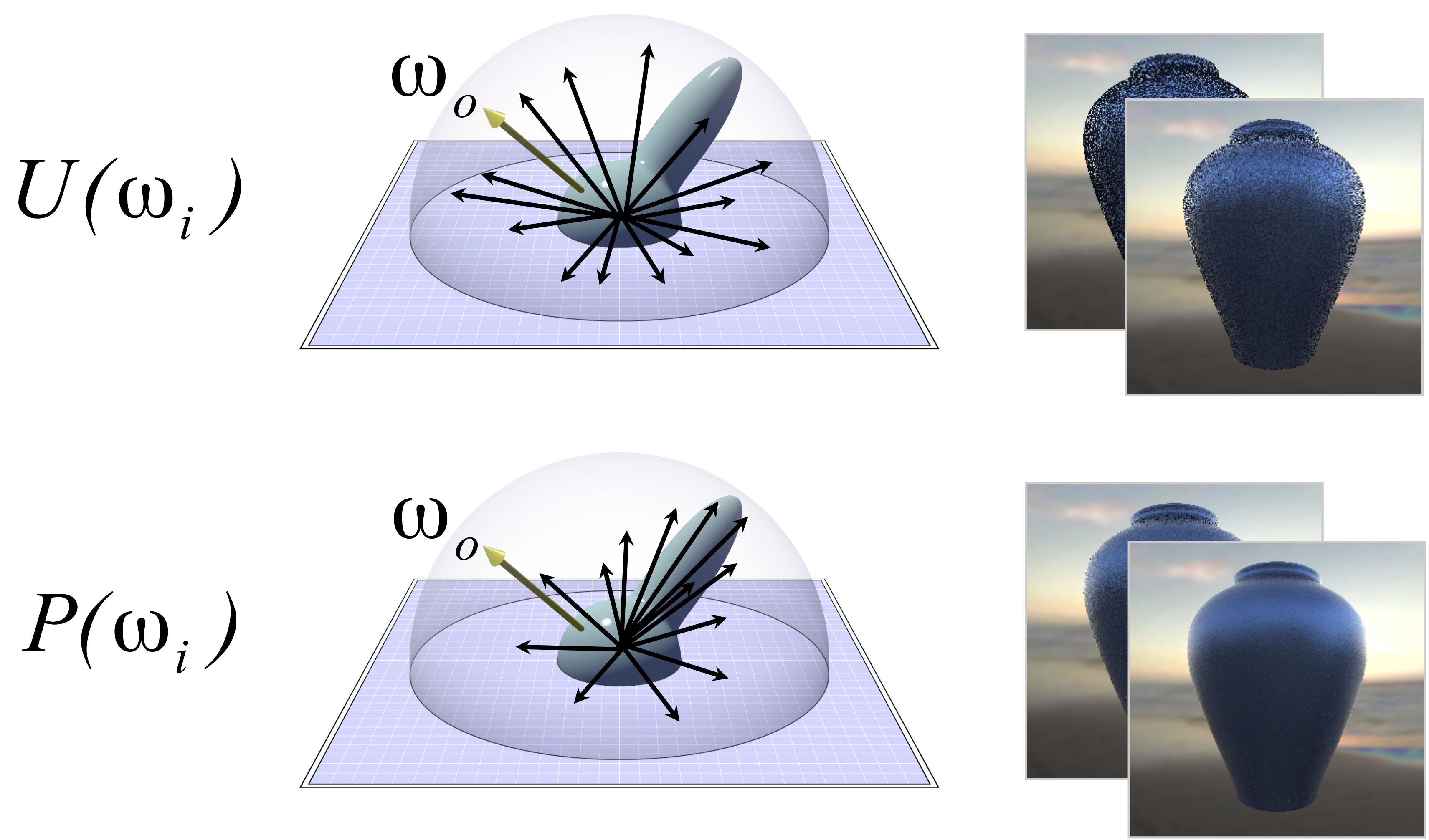
Brute Force Path Tracing

$$L(x \rightarrow \mathbf{v}) = \int_\Omega L(x \leftarrow \mathbf{l})\, f_r(x, \mathbf{l} \rightarrow \mathbf{v})\, \cos\theta\, \mathrm{dl}$$
$$+ E(x \rightarrow \mathbf{v})$$

```
trace(ray)
 hit = intersect(scene, ray)
 result = emission(hit,-dir(ray))  // 0 if no light
 // sample outgoing direction
 [w,pdf] = sampleReflection(hit,dir(ray))
 // recursively estimate incoming radiance, apply BRDF
 result += BRDF(hit,-dir(ray),w)*
           cos(theta)*
           trace(ray(hit,w))/pdf
 return result
// when we apply the PDF like this, we are implicitly
// multiplying them for all bounces like shown before
```

Notes

- `sampleReflection()` chooses a direction with which to estimate reflectance integral for indirect part
  - I.e. importance sample according to BRDF

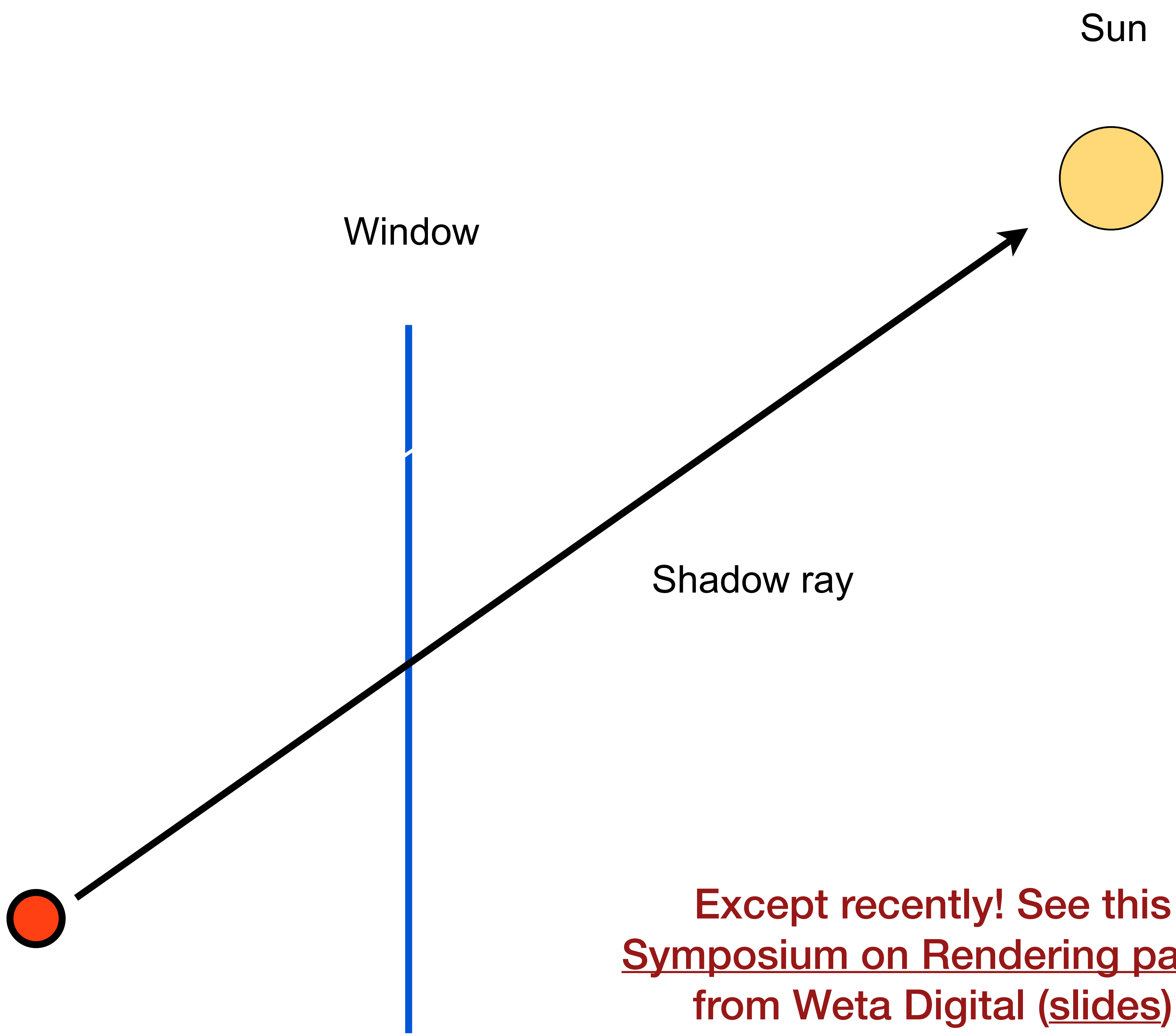


Jason Lawrence

Why “Brute Force”?

- We’re waiting for the sampler to hit the light on its own
  - Often not a good idea
  - But sometimes we can’t do too much else
  - Think of an architectural model where all the light comes through several specular bounces through windows
- In simple cases we can help by adding an explicit direct light sampling step to each bounce

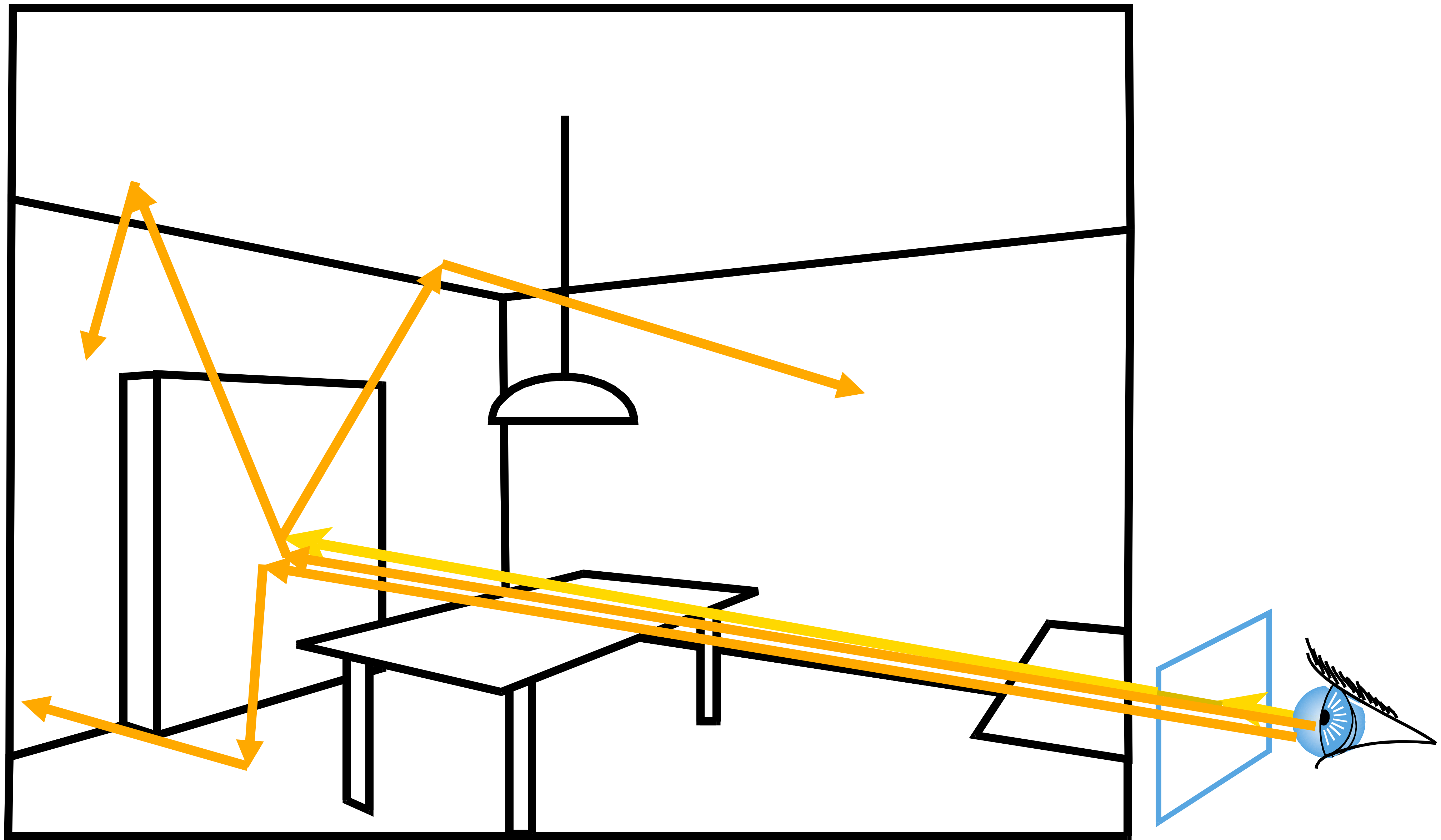
~~This Doesn't Work!~~



Except recently! See this [Symposium on Rendering paper]() from Weta Digital ([slides]())

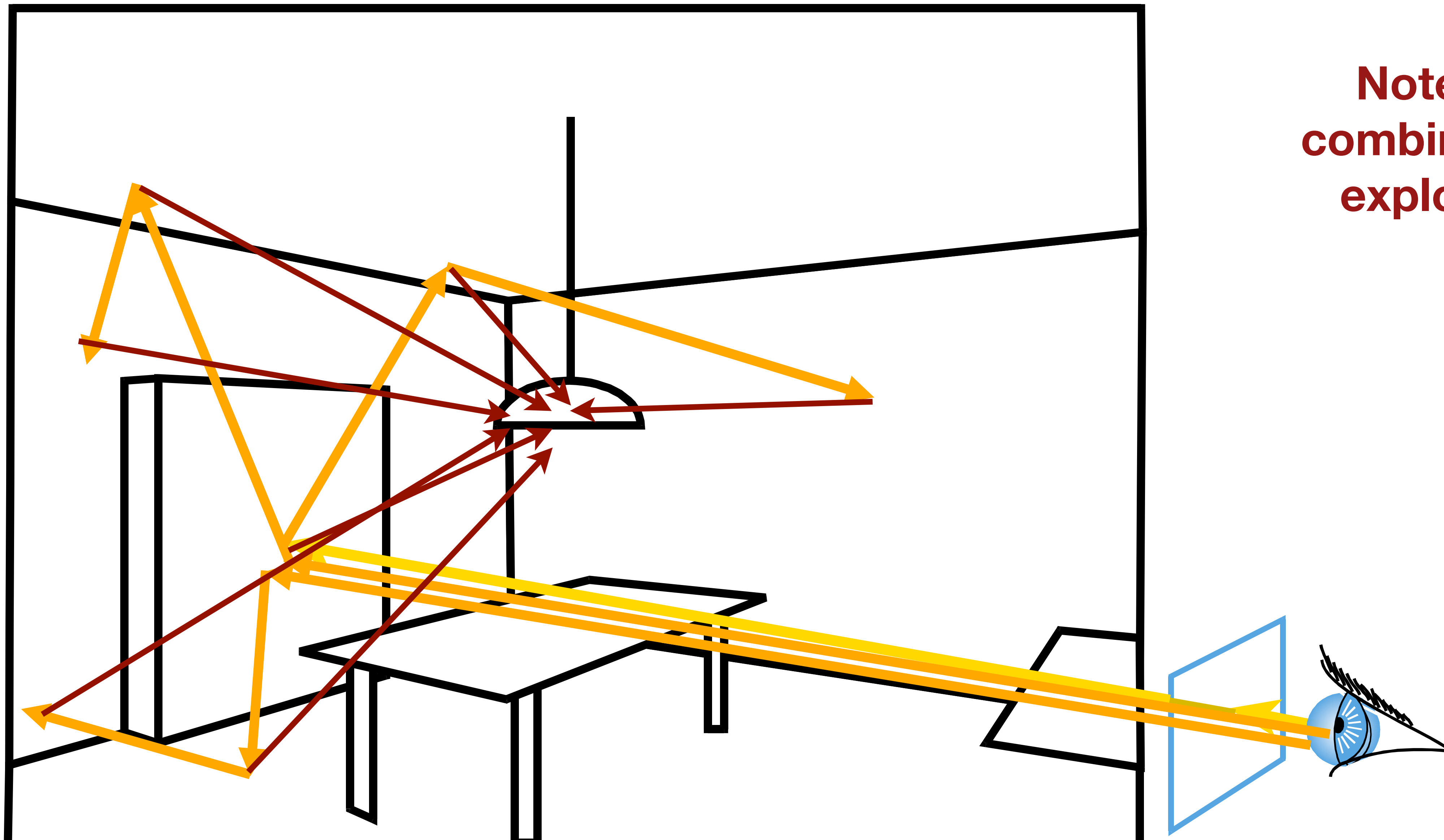
Brute Force Path Tracing

- Trace only one secondary ray per recursion
  - Otherwise number of rays explodes!
- But send many primary rays per pixel (antialiasing)



Path Tracing w/ Light Sampling

- At each hit, also sample a light and shoot a shadow ray
- The standard way of doing path tracing
- Also called "next event estimation"



Note: No combinatorial explosion!

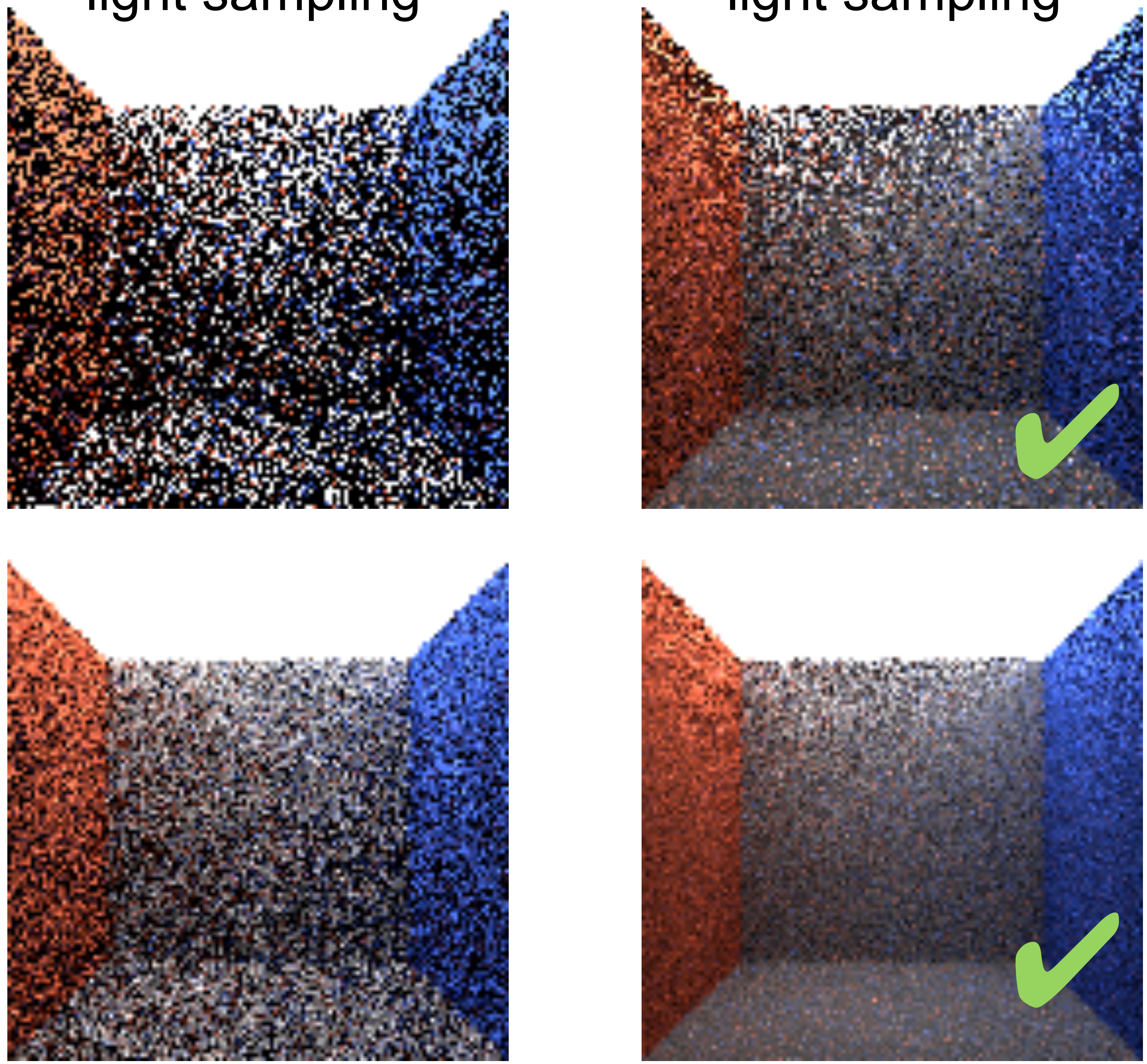
Importance of Sampling the Light

Without explicit light sampling

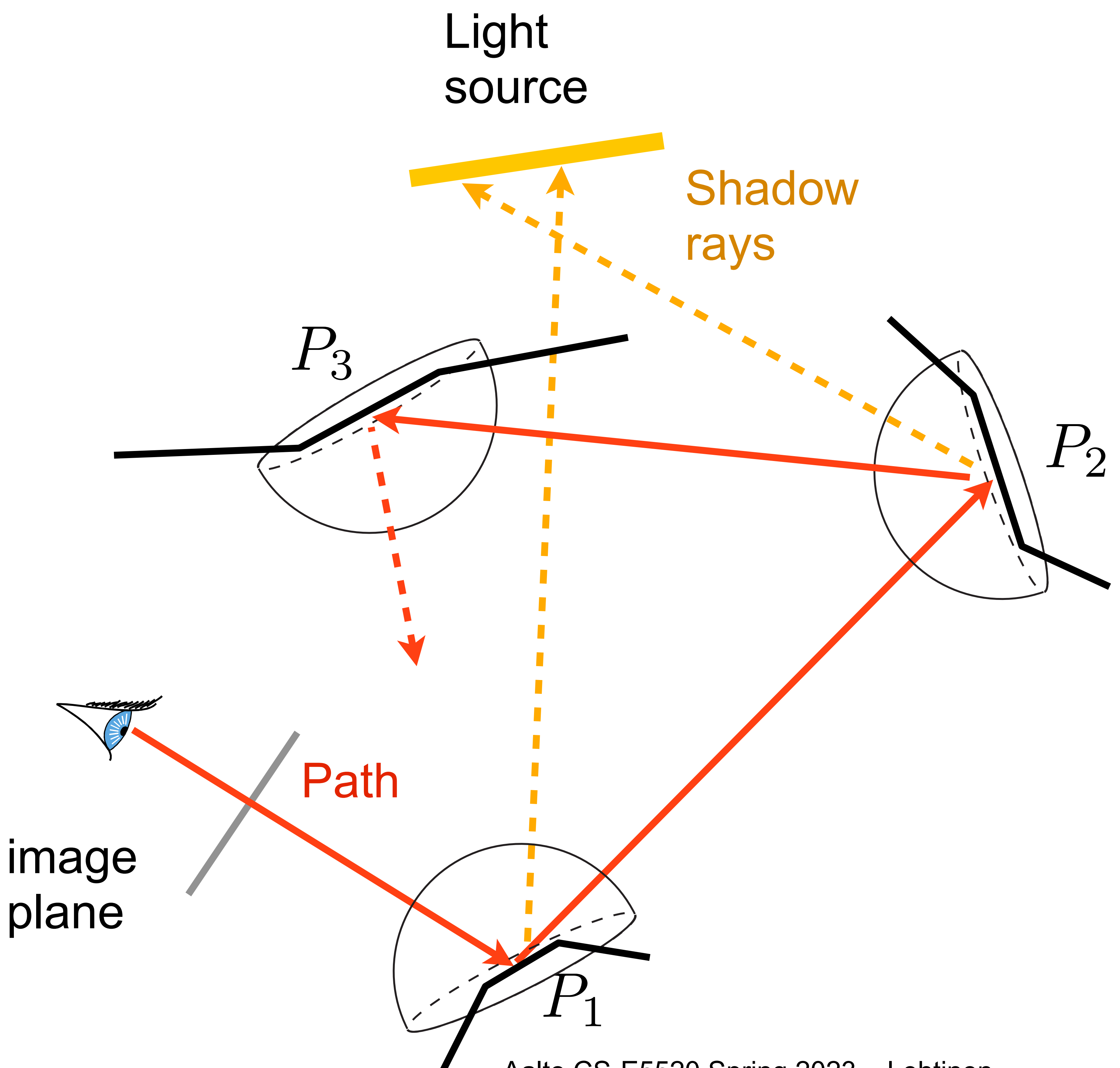
With explicit light sampling

1 path per pixel

4 paths per pixel



Path Tracing w/ Light Sampling



Interpretation of Shadow Rays

- Recall: the full lighting solution is a sum over paths of all lengths

$$L(x,y) = \sum_{i=0}^{\infty} L_i(x,y), \qquad \text{with } L_0(x,y) = E(P_1 \leftarrow \text{eye})$$

- Notice how we've "unwrapped" the recursive rendering equation into a sum of terms
  - n bounce lighting is an integral over $\text{screen} \times \underbrace{\Omega \times \ldots \times \Omega}_{n \text{times}}$ (brute force PT)
  - But now we've replaced the final hemisphere with lights by solid-angle-to-area conversion: $\text{screen} \times \omega \times \omega \ldots \times \text{lights}$

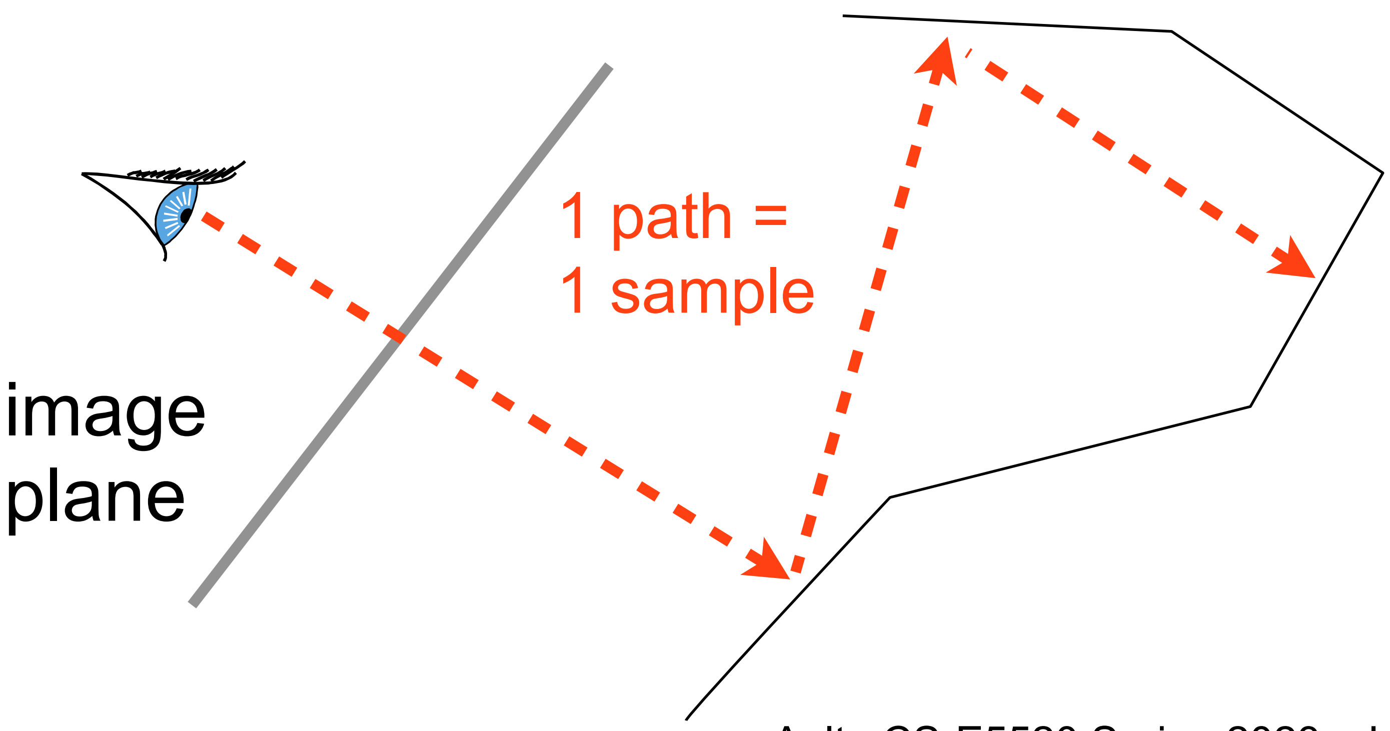
A Different Parameterization

- In hemisphere form, the domain for n bounces is

$$\text{screen} \times \underbrace{\Omega \times \ldots \times \Omega}_{n \text{times}}$$

- For shadow ray sampling, it is

$$\text{screen} \times \underbrace{\Omega \times \ldots \Omega}_{n-1 \text{ times}} \times \text{light area}$$



Path Tracing Pseudocode

$$L(x \rightarrow \mathbf{v}) = \int_\Omega L(x \leftarrow \mathbf{l})\, f_r(x, \mathbf{l} \rightarrow \mathbf{v})\, \cos\theta\, \mathrm{d}\mathbf{l}$$

$$+ \; E(x \rightarrow \mathbf{v})$$

```
trace(ray)
 hit = intersect(scene, ray)
 if ray is from camera // only add "very direct" light here
  result = emission(hit,-dir(ray))
 [y,pdf1] = sampleLightsource()     // pick shadow ray dest.
 // G(hit,y) contains the usual cosine/r^2 of the
 // hemisphere-to-area variable change
 result += V(hit,y)*E(y,y->hit)*BRDF*cos*G(hit,y)/pdf1
 [w,pdf] = sampleReflection(hit,dir(ray)) // like before
 result += BRDF(hit,-dir(ray),w)*
           cos(theta)*
           trace(ray(hit,w))/pdf
 return result
```

Notes 2

- `sampleLightsource`() picks a point on the light source and evaluates its PDF
  - You're doing this in the first part of your radiosity assignment
  - ..and we saw this already on the first MC lecture
  - We're (again) applying the solid angle-to-area variable change (i.e. we're integrating over the surface of the light source)

- When you have multiple light sources, you pick *one* at random, and build this into the PDF
  - Simple: just multiply the light source $p(y)$ with the probability of picking that particular light source

Picking Lights

- It makes sense to importance sample the light you pick
- E.g. doesn’t make sense to sample dim, far-away lights as often as bright, nearby ones!

One Small Problem

One Small Problem

- Yes, it doesn’t terminate if you just keep going
  - Fortunately, there’s still something we can do!

Russian Roulette

- The usual MC estimate is $E\{\frac{f(x)}{p(x)}\}_p$
  - f/p is a random variable because x is a random variable

Russian Roulette

- The usual MC estimate is $E\{\frac{f(x)}{p(x)}\}_p$
  - f/p is a random variable because x is a random variable
- Let's multiply this by another specially constructed random variable R
  - R(x)=0 with probability $\alpha(x)$, and $R = 1/(1-\alpha)$ otherwise
  - Also assume α and x are uncorrelated (independent). Then:

$$E\{\frac{R \cdot f(x)}{p(x)}\} = E\{R\}\, E\{\frac{f(x)}{p(x)}\} = E\{\frac{f(x)}{p(x)}\}$$

This step requires independence

Russian Roulette: What is Going On?

- R(x)=0 with probability $\alpha(x)$, and $R = 1/\alpha$ otherwise

$$E\{\frac{R \cdot f(x)}{p(x)}\} = E\{R\}\, E\{\frac{f(x)}{p(x)}\} = E\{\frac{f(x)}{p(x)}\}$$

- *We've given ourselves permission to sometimes replace the value of the integrand with zero without introducing bias to the result*
  - When we don't set it to zero, we multiply the result by $1/\alpha$
- This means, for instance, that we can probabilistically terminate light paths without tracing them to infinity

Path Tracing w/ RR

$$L(x \rightarrow \mathbf{v}) = \int_{\Omega} L(x \leftarrow \mathbf{l})\, f_r(x, \mathbf{l} \rightarrow \mathbf{v})\, \cos\theta \,\mathrm{d}\mathbf{l}$$

$$+\ E(x \rightarrow \mathbf{v})$$

```
trace(ray)
 hit = intersect(scene, ray)
 if ray is from camera // only add "very direct" light here
  result = emission(hit,-dir(ray))
 [y,pdf1] = sampleLightsource()      // pick shadow ray dest.
 result += E(y,y->hit)*BRDF*cos*G(hit,y)/pdf1
 [w,pdf] = sampleReflection(hit,dir(ray))
 // russian roulette with alpha=0.5
 terminate = uniformrandom() < 0.5
 if !terminate
  result += BRDF(hit,-dir(ray),w)*
            cos(theta)*
            trace(ray(hit,w))/pdf/0.5   // 1/0.5 =mult. by 2!
 return result
```

“Path Space”

- Earlier we wrote n-bounce lighting as a simultaneous integral over n hemispheres
- We can just as well integrate over surfaces instead
  - We just need to add in the geometry terms like before
    - $1/r^2$, visibility, the other cosine
- The space of paths of length n is then simply

$$\underbrace{S \times \ldots \times S}_{n \text{ times}}$$

with S being the set of 2D surfaces of the scene

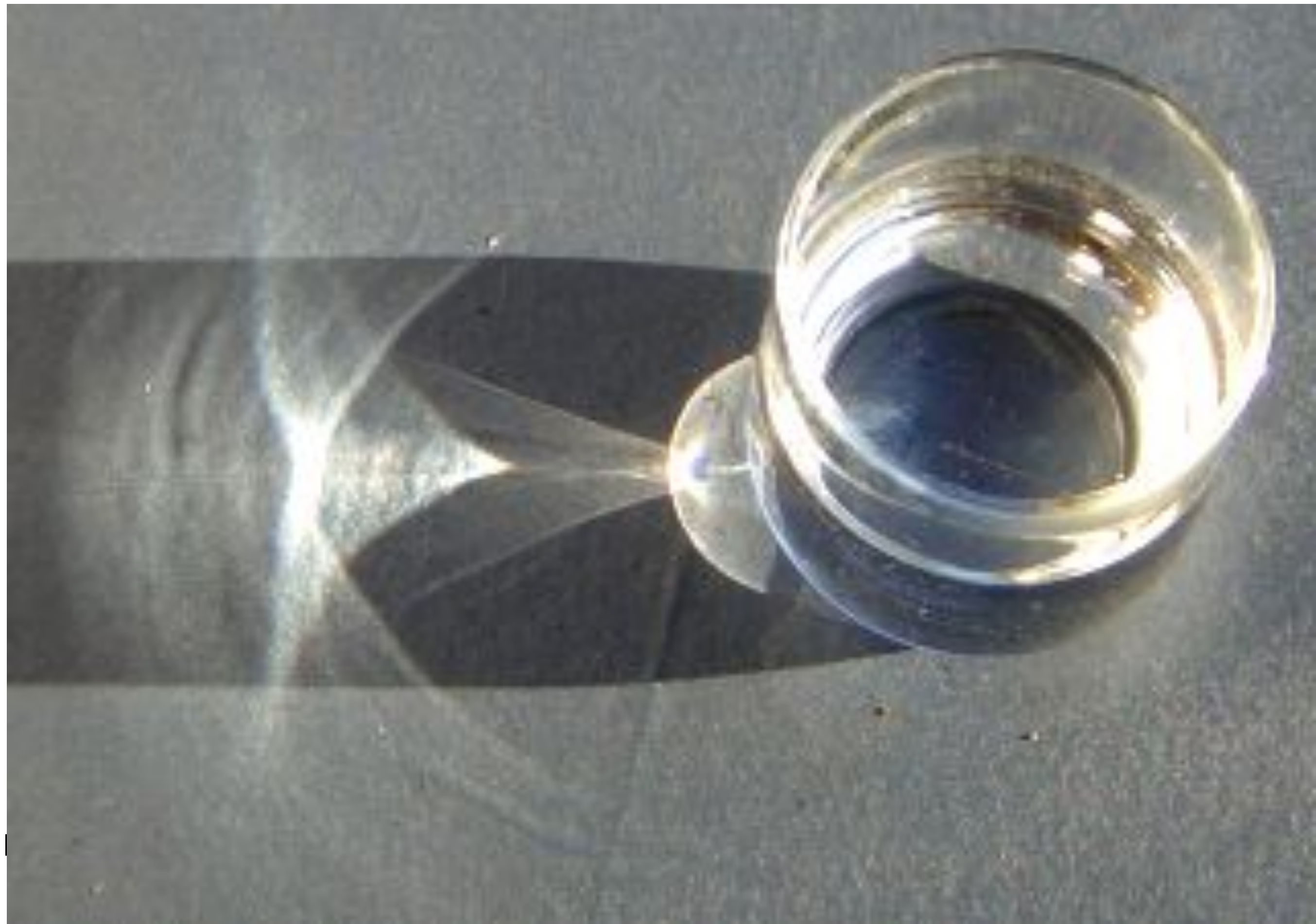
- See Eric Veach’s PhD

Bigger Picture

- We are shooting rays from the camera, propagating them along, and kind of hoping we will find light
  - Actively try to hit it by the light source samples

- What about more difficult cases?
  - In a *caustic*, the light propagates through a series of specular refractions and reflections before hitting a diffuse surface

wikipedia



Problem With Caustics

- All we can do is shoot shadow rays towards the light
  - Not very helpful here!

To be continued…