Aalto CS-E5520 Spring 2024

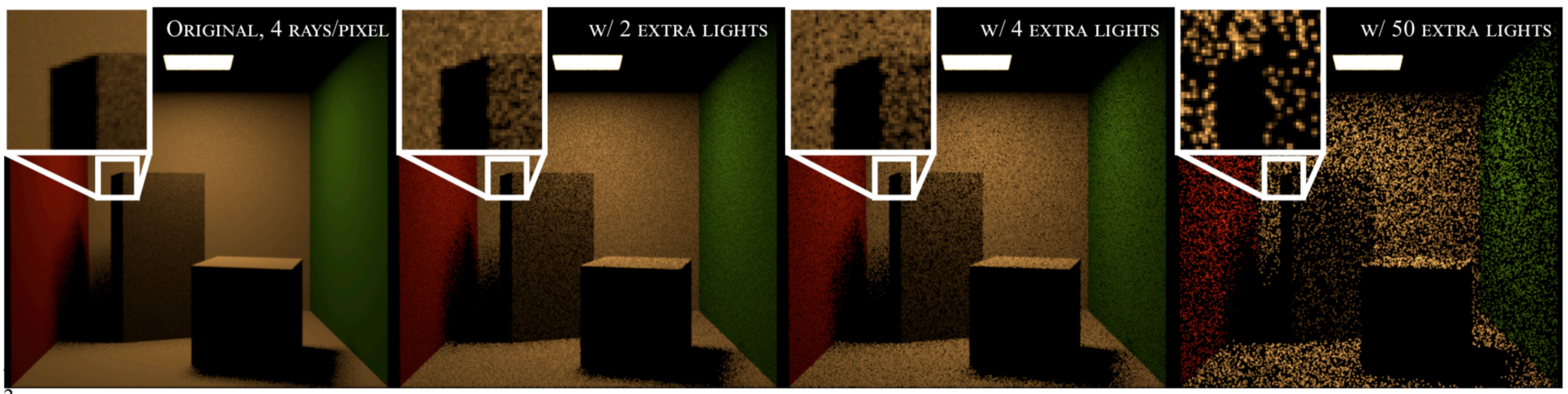
Jaakko Lehtinen

Beyond Path Tracing: Bidirectional and Further

Mansfields Shoe Factory, c. 1900

WebUrbanist

These Should All Be the Same Image..



(Extra lights added next to the box so that all light from them is blocked by the walls)

Path Tracing w/ RR (Recap)

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

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

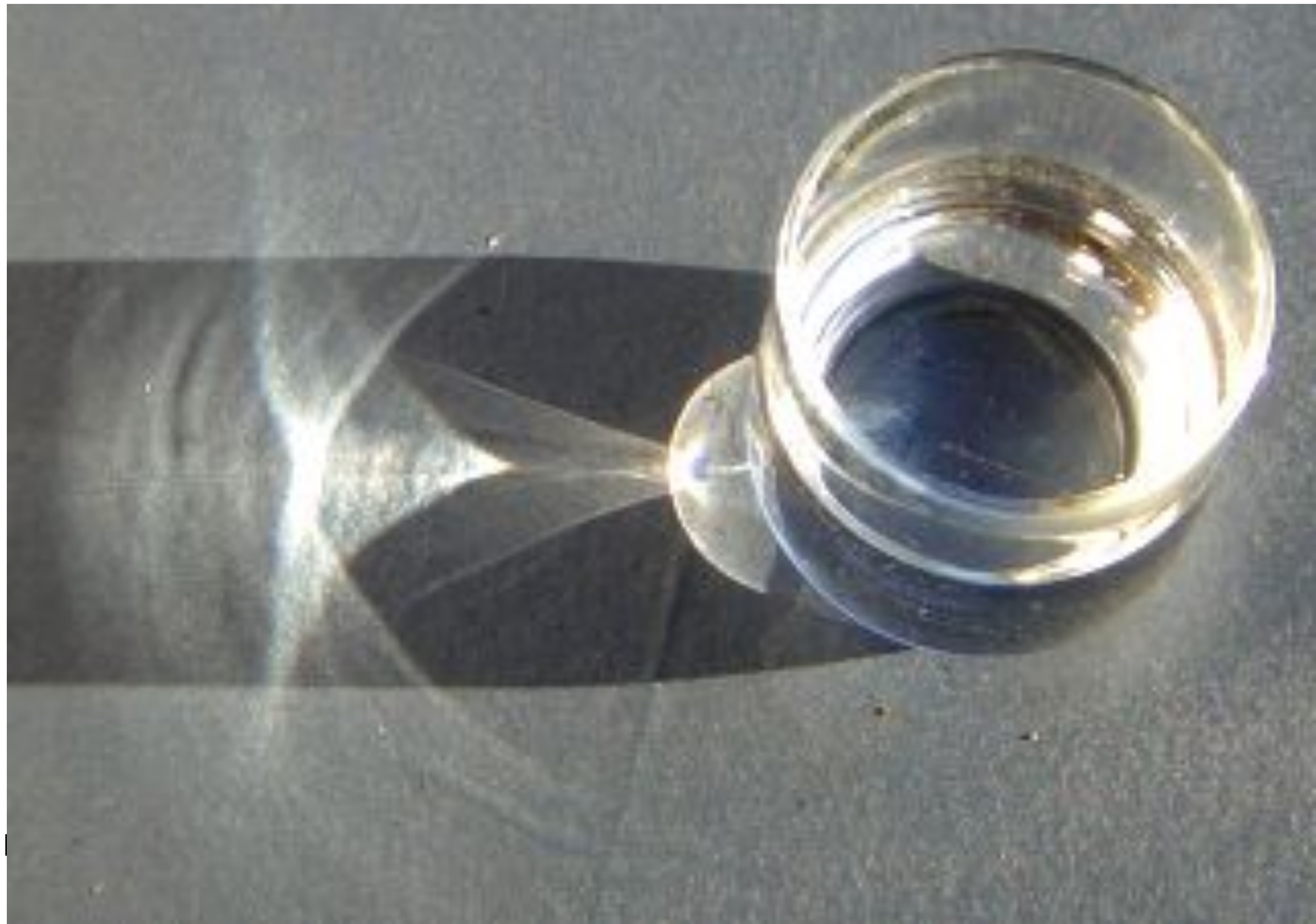
```
trace(ray)
 hit = intersect(scene, ray)
 if ray is from camera // only add "very direct" light here
  result = emission(hit,-dir(ray))
 [y,pdf1] = sampleLightsource()      // pick shadow ray dest.
 result += E(y,y->hit)*BRDF*cos*G(hit,y)/pdf1
 [w,pdf] = sampleReflection(hit,dir(ray))
 // russian 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
```

Bigger Picture

- In Path Tracing, we shoot rays from the camera, propagate them along, and kind of hope we'll find light
  - Actively try to hit it by the light source samples

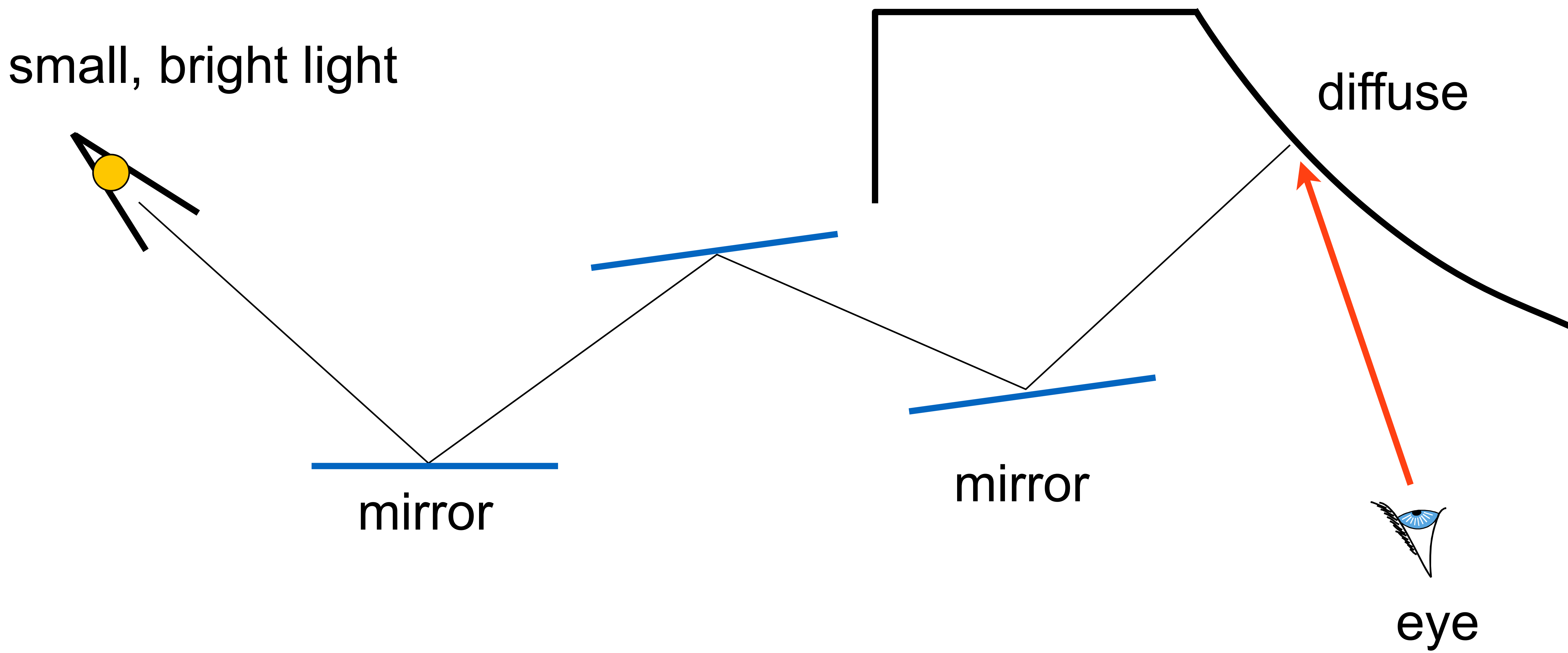
- What about more difficult cases?
  - In a *caustic*, the light propagates through a series of specular refractions and reflections before hitting a diffuse surface

wikipedia



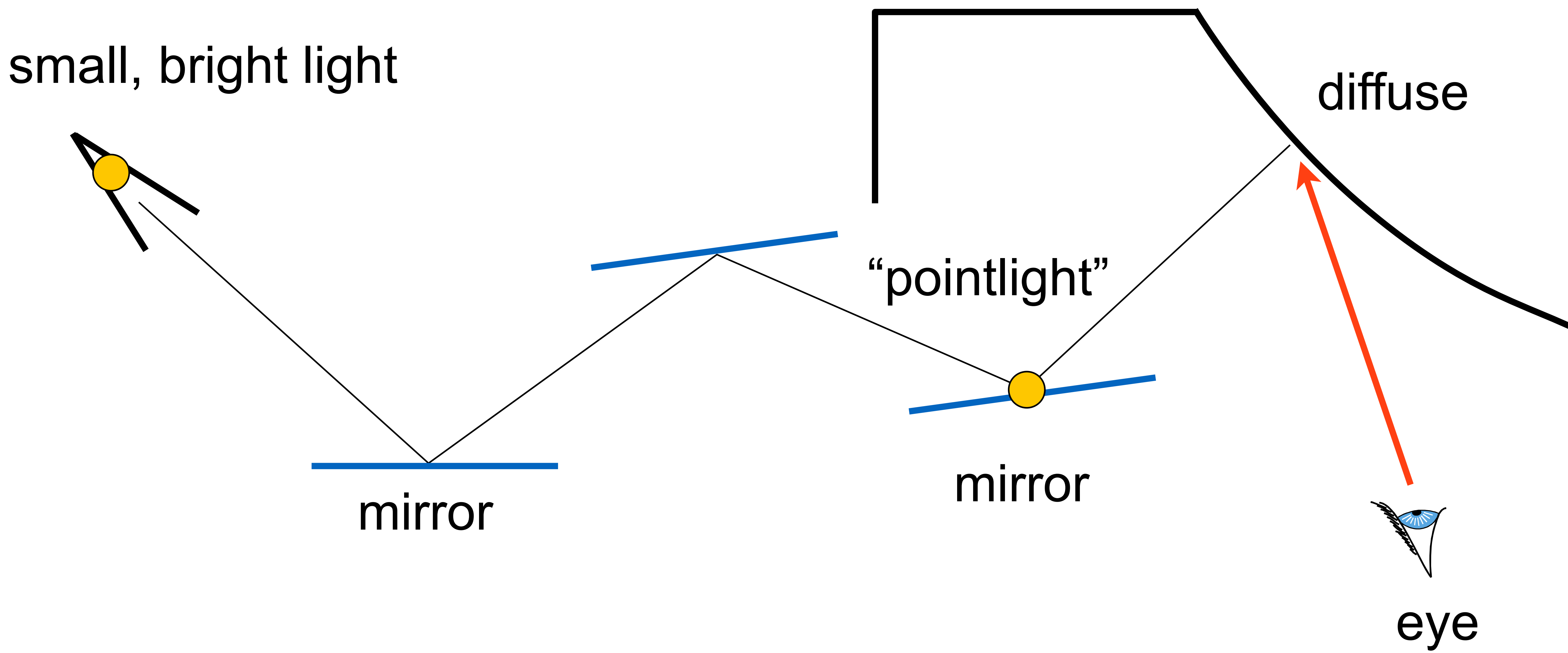
Problem With Caustics

- Think of an almost pointlike light shining through a sequence mirrors onto a diffuse receiver



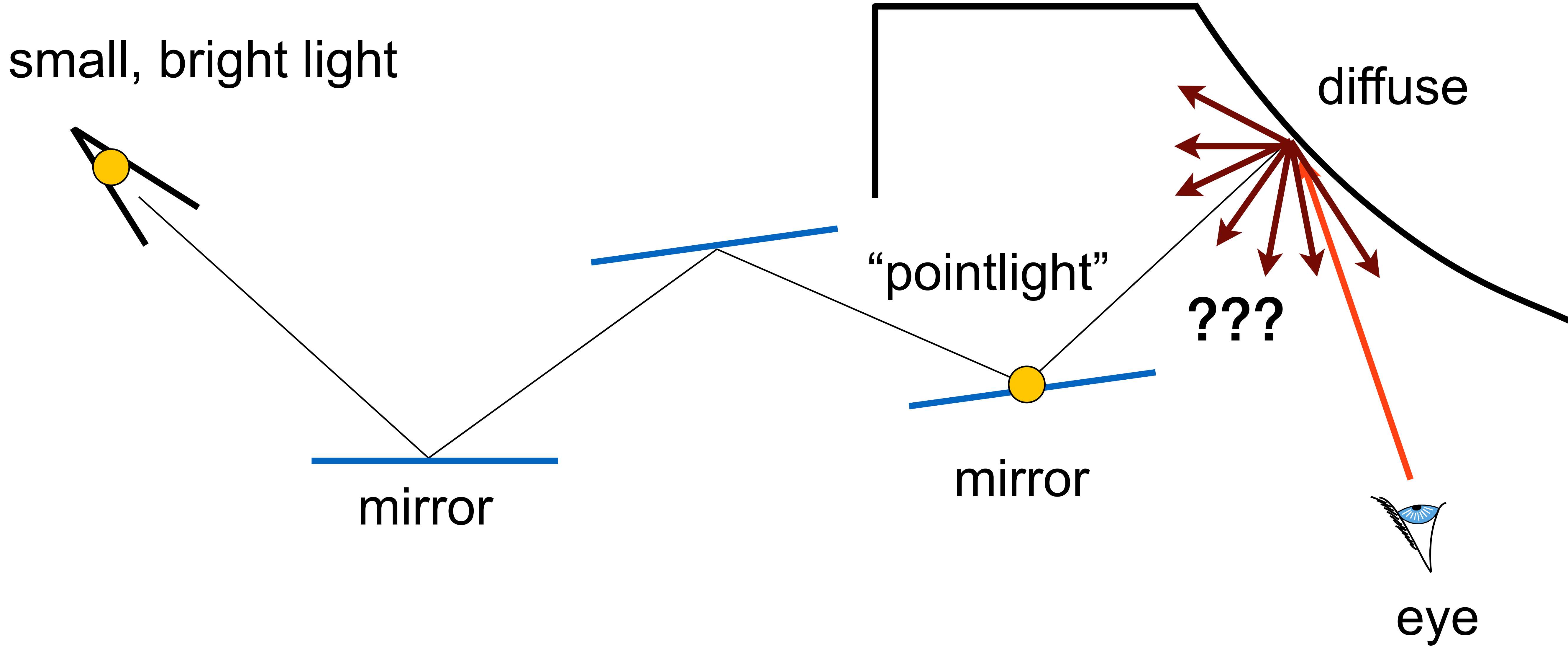
Problem With Caustics

- The point hit by the eye ray effectively sees a pointlight in the direction of the last mirror



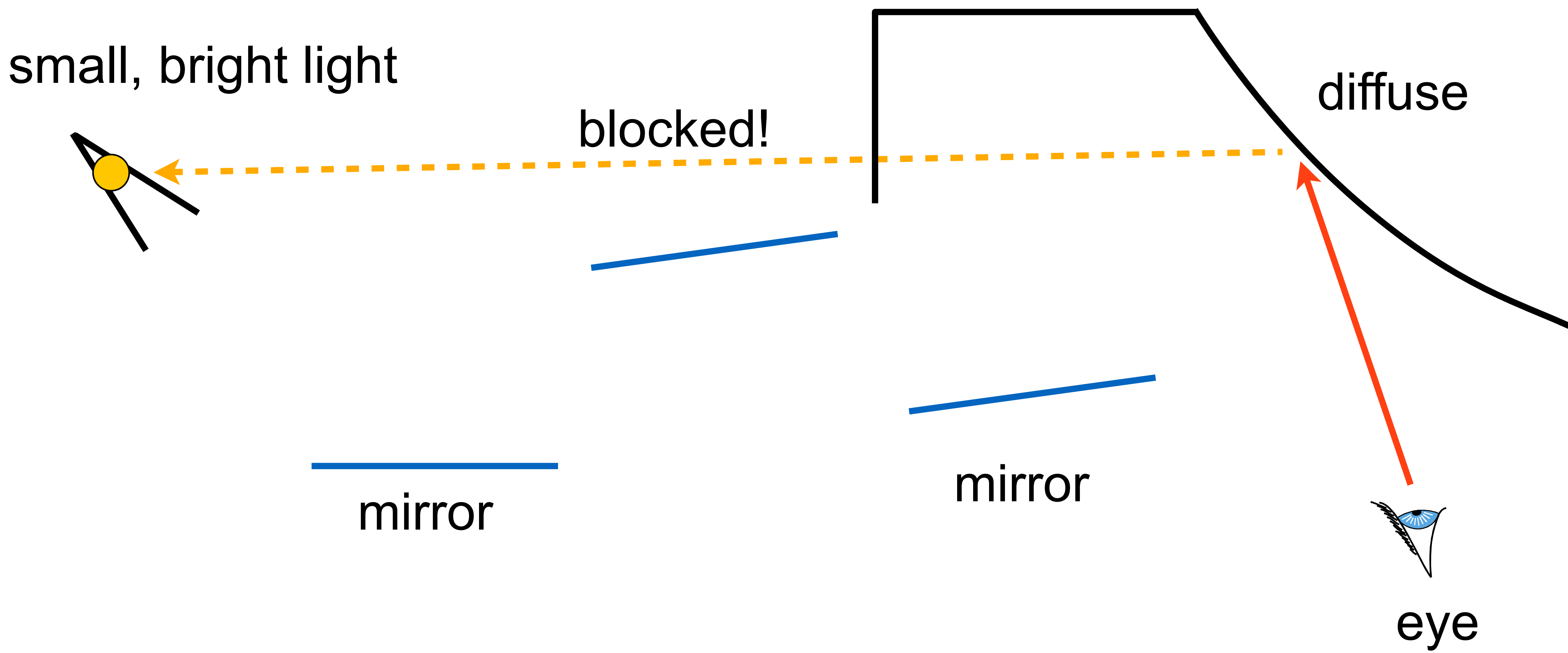
Problem With Caustics

- The point hit by the eye ray effectively sees a pointlight in the direction of the last mirror
  - *Does the cosine importance sampler know that..?*

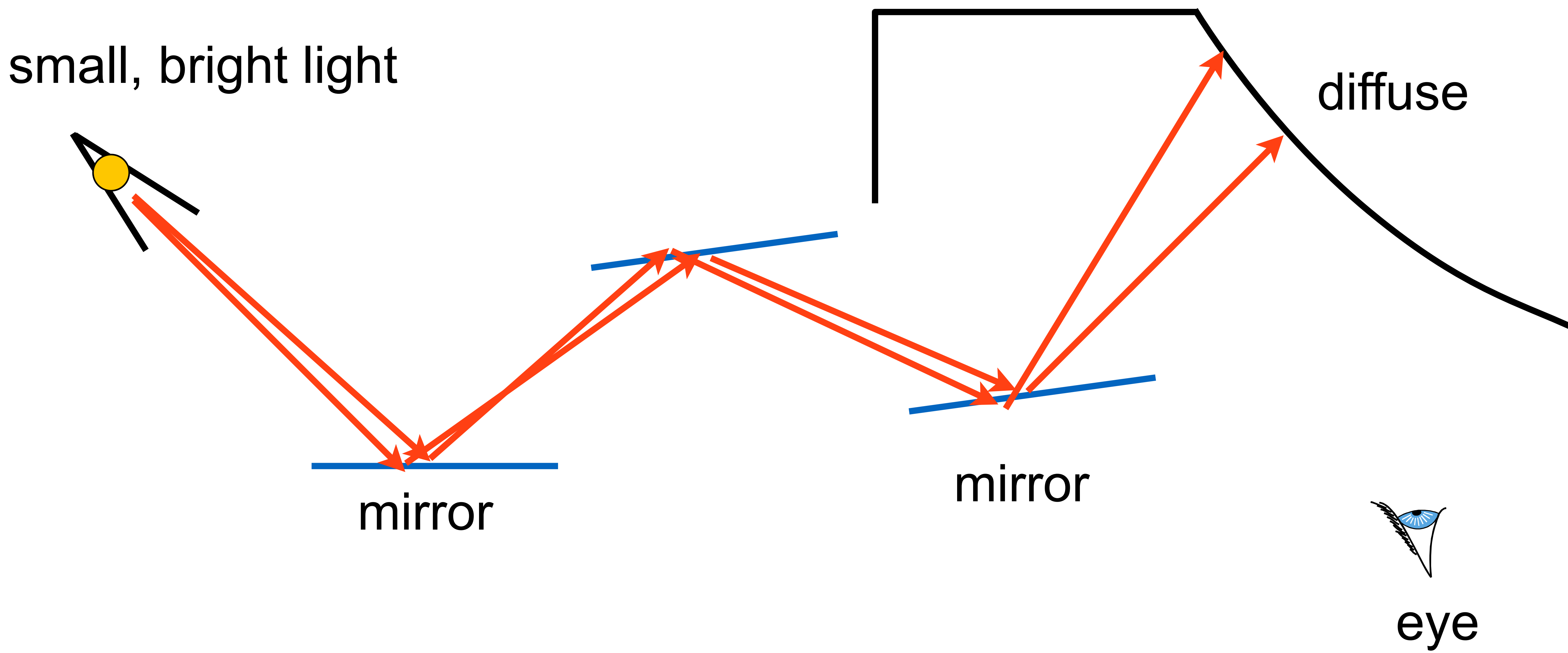


Problem With Caustics

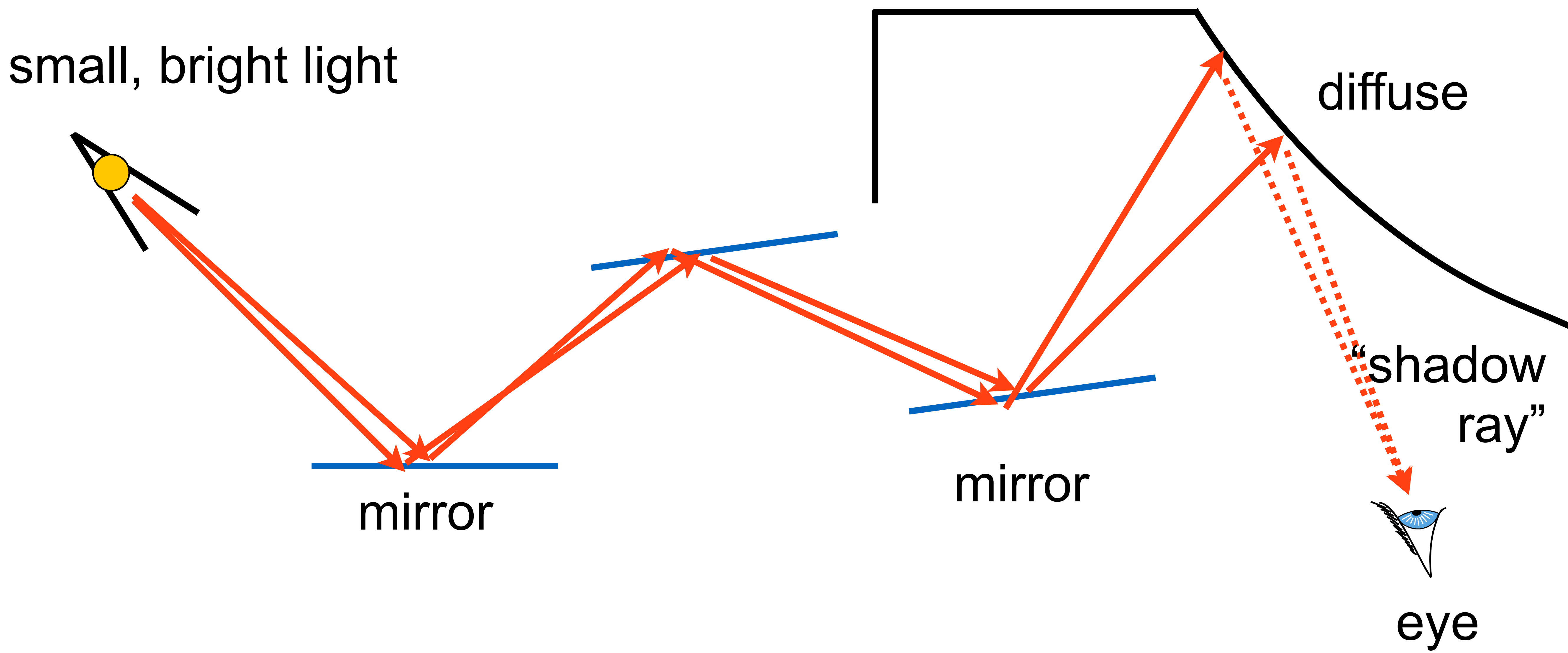
- All we can do is shoot shadow rays towards the light
  - Not helpful here!



What if…

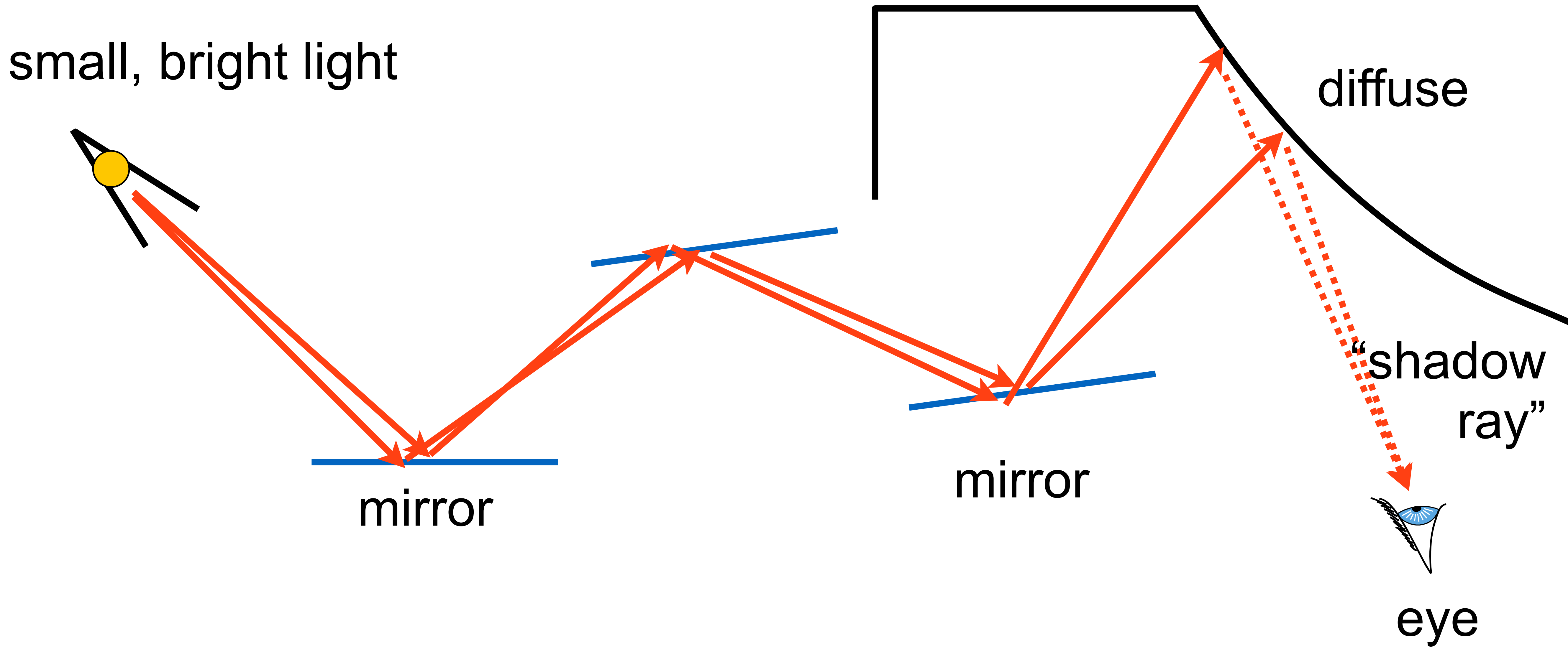


What if…



“Light Tracing” = reverse path tracing

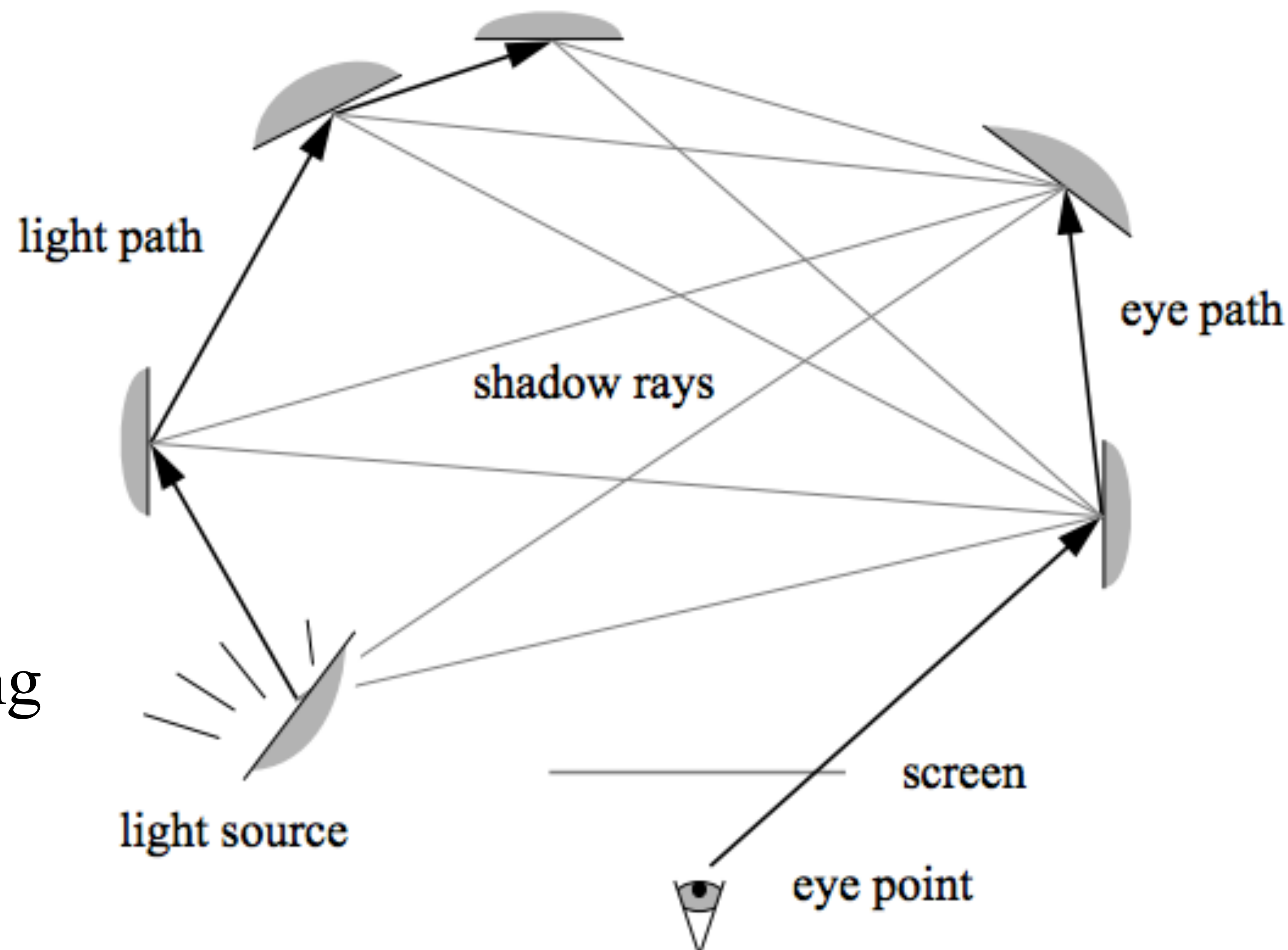
- “Shadow rays” towards camera from all path vertices, much like regular shadow rays



Bidirectional Path Tracing (BDPT)

- Veach and Guibas 95, Lafortune and Willems 93

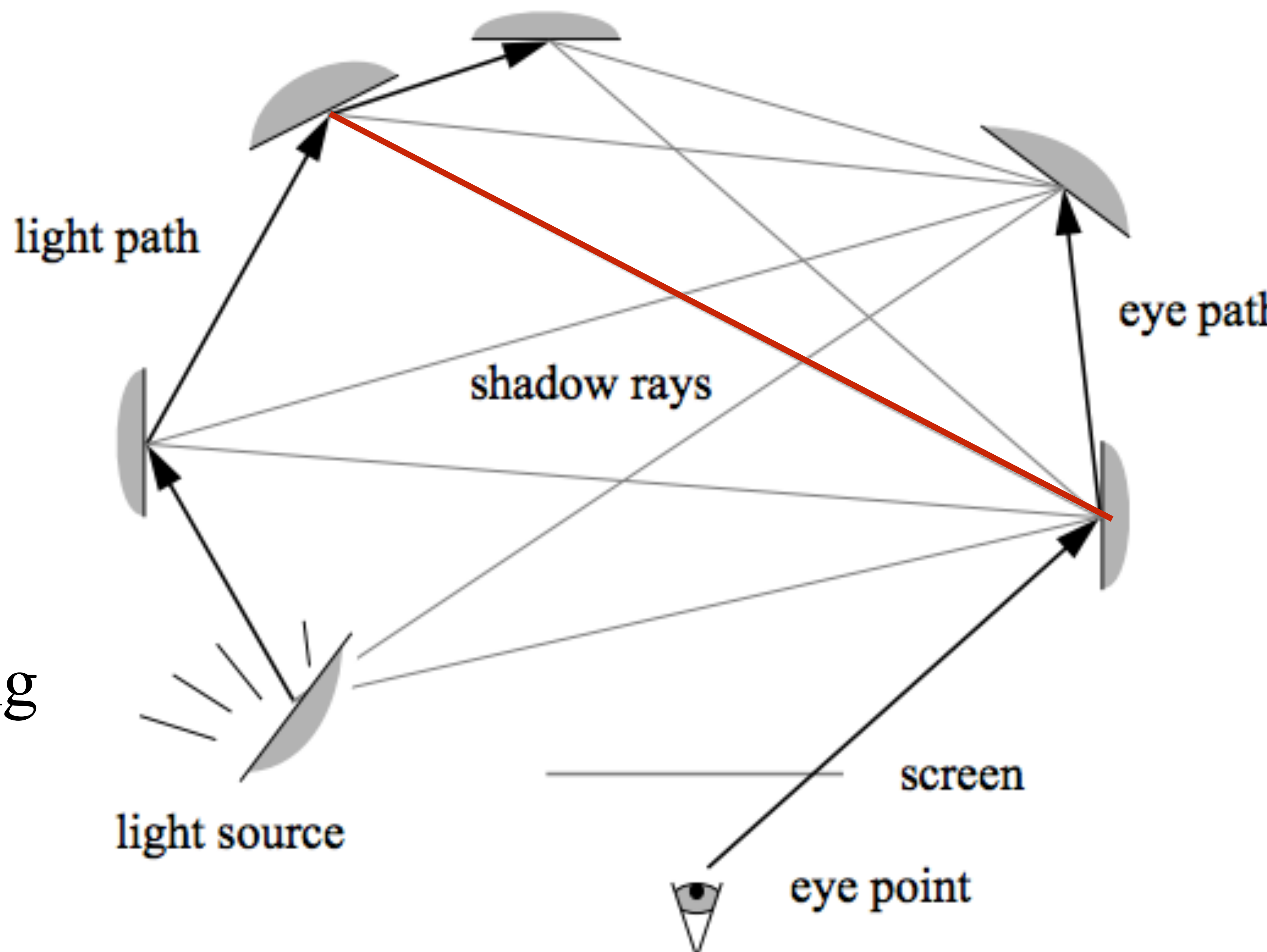
- Shoot a path from light
- Shoot another from the eye
- Connect the two
  - Use multiple importance sampling
- Best exposition in Veach's thesis



Bidirectional Path Tracing (BDPT)

- Veach and Guibas 95, Lafortune and Willems 93

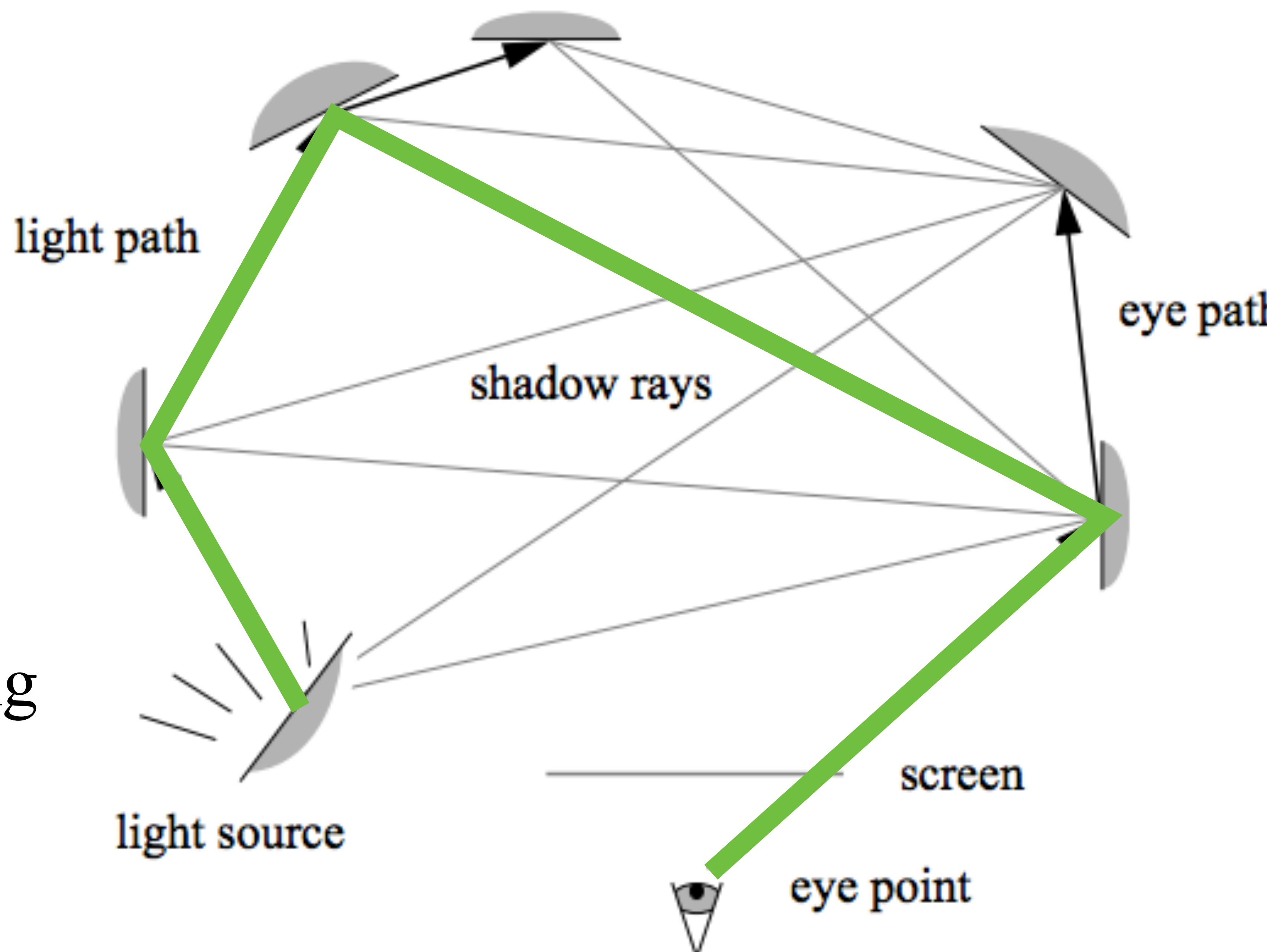
- Shoot a path from light
- Shoot another from the eye
- Connect the two
  - Use multiple importance sampling
- Best exposition in Veach's thesis



Bidirectional Path Tracing (BDPT)

- Veach and Guibas 95, Lafortune and Willems 93

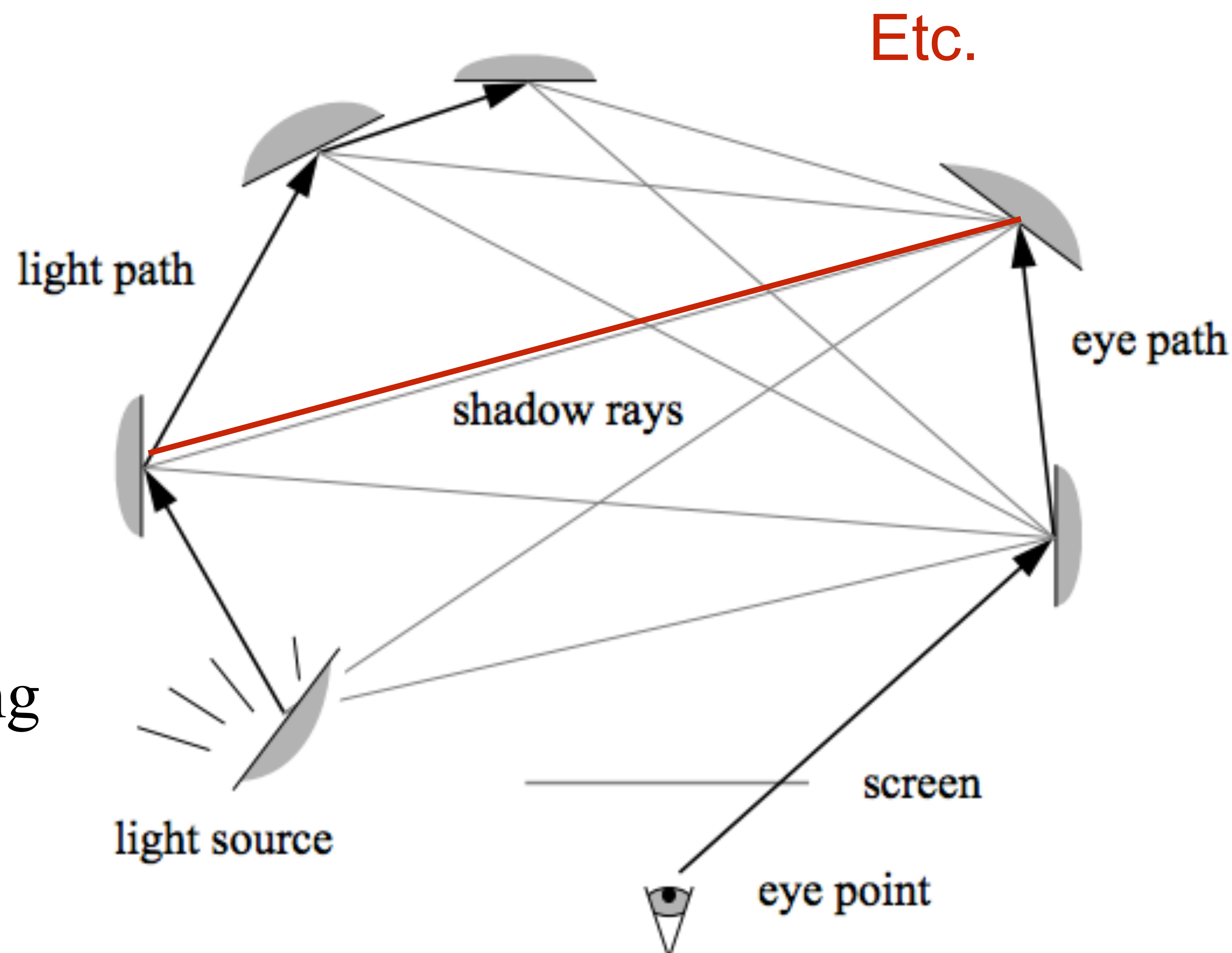
- Shoot a path from light
- Shoot another from the eye
- Connect the two
  - Use multiple importance sampling
- Best exposition in Veach's thesis



Bidirectional Path Tracing (BDPT)

- Veach and Guibas 95, Lafortune and Willems 93

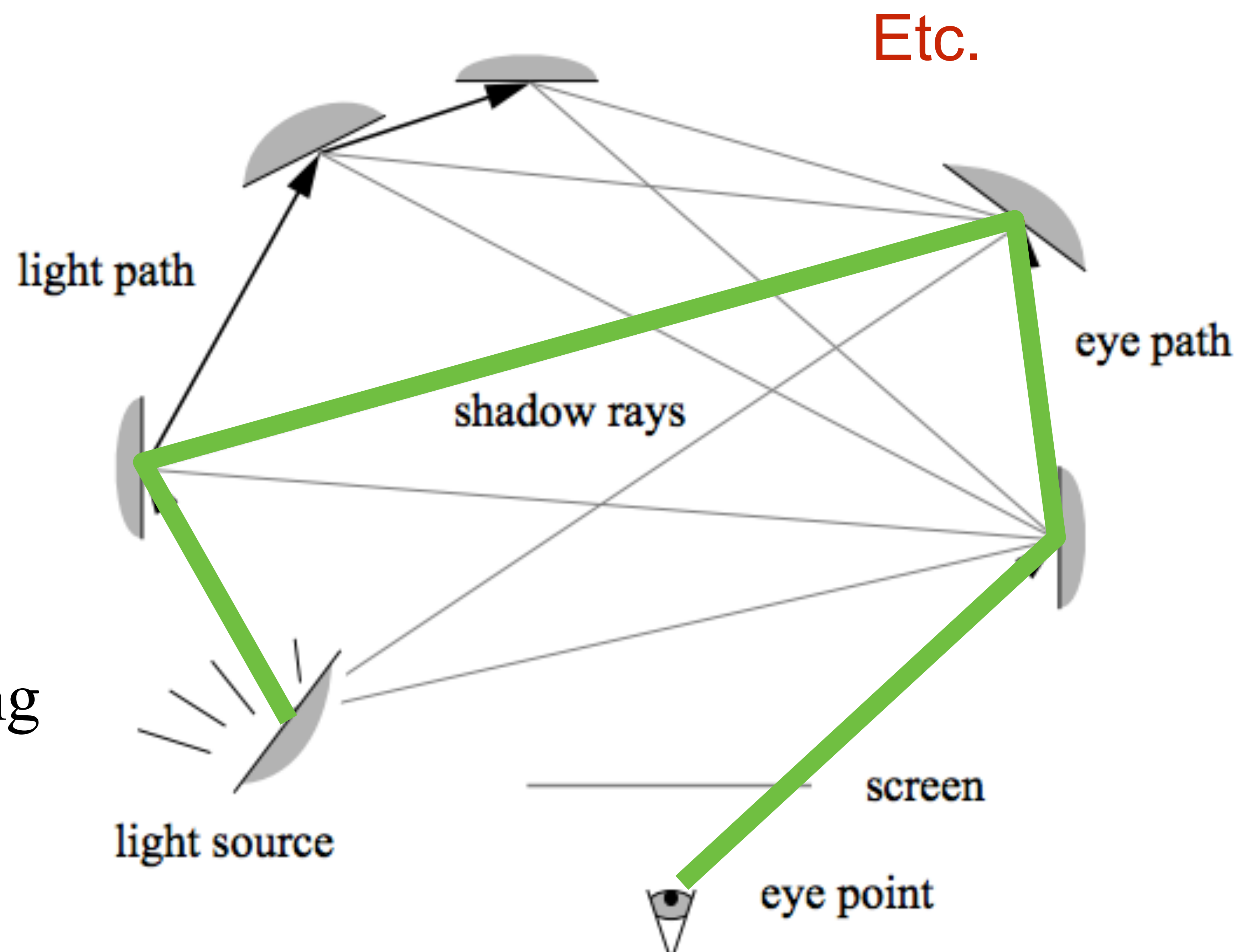
- Shoot a path from light
- Shoot another from the eye
- Connect the two
  - Use multiple importance sampling
- Best exposition in Veach's thesis



Bidirectional Path Tracing (BDPT)

- Veach and Guibas 95, Lafortune and Willems 93

- Shoot a path from light
- Shoot another from the eye
- Connect the two
  - Use multiple importance sampling
- Best exposition in Veach's thesis



BDPT Advantages

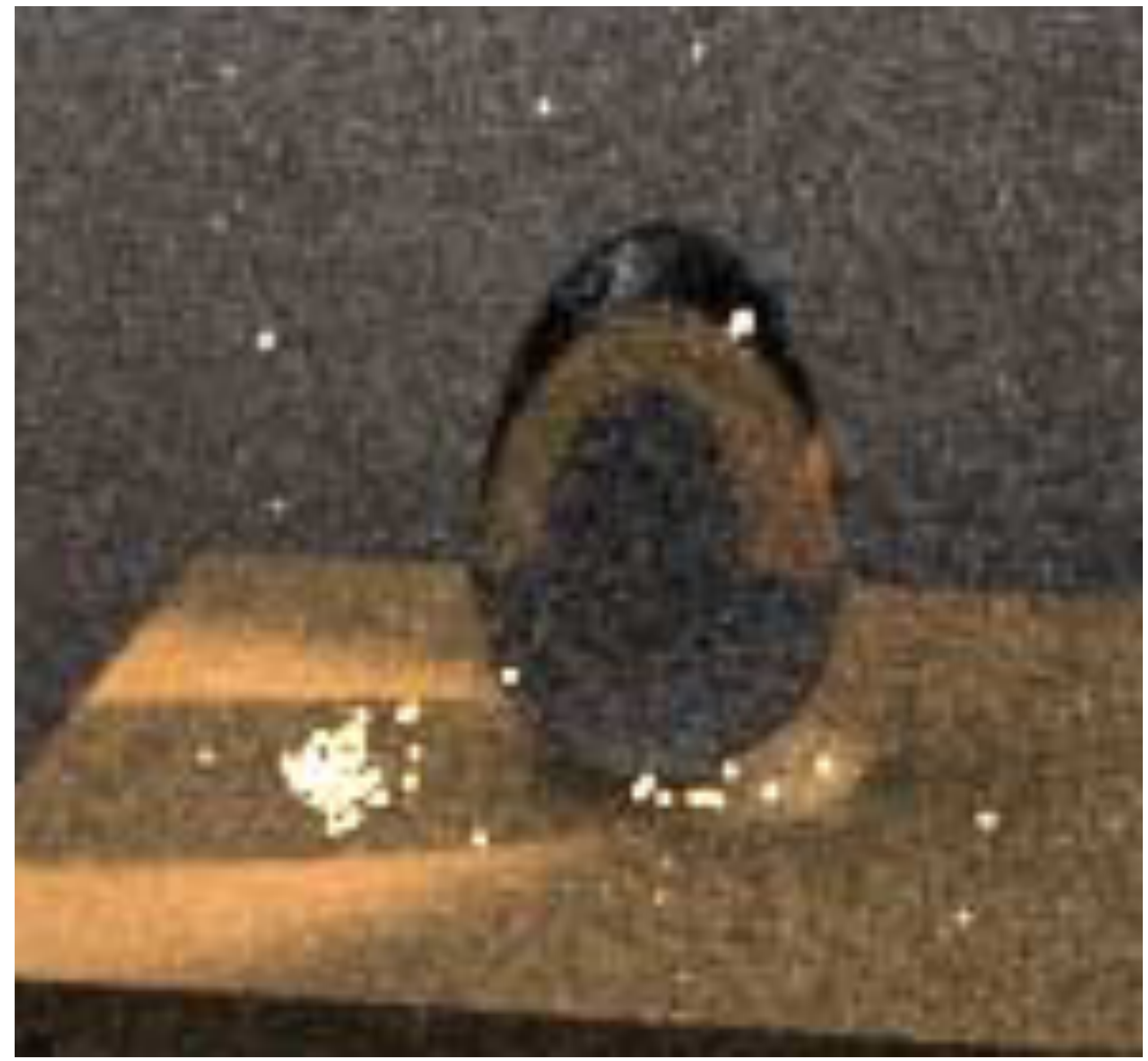
- As the starting example shows, sometimes it's easy to sample from the light...
- ...at other times, such as when you are looking at a scene through many specular interactions, eye sampling is better
- BDPT with Multiple Importance Sampling is a principled way of combining the benefits from both
  - See Veach's 95 MIS paper (again)

- See Dietger van Antwerpen's PhD thesis for a GPU implementation

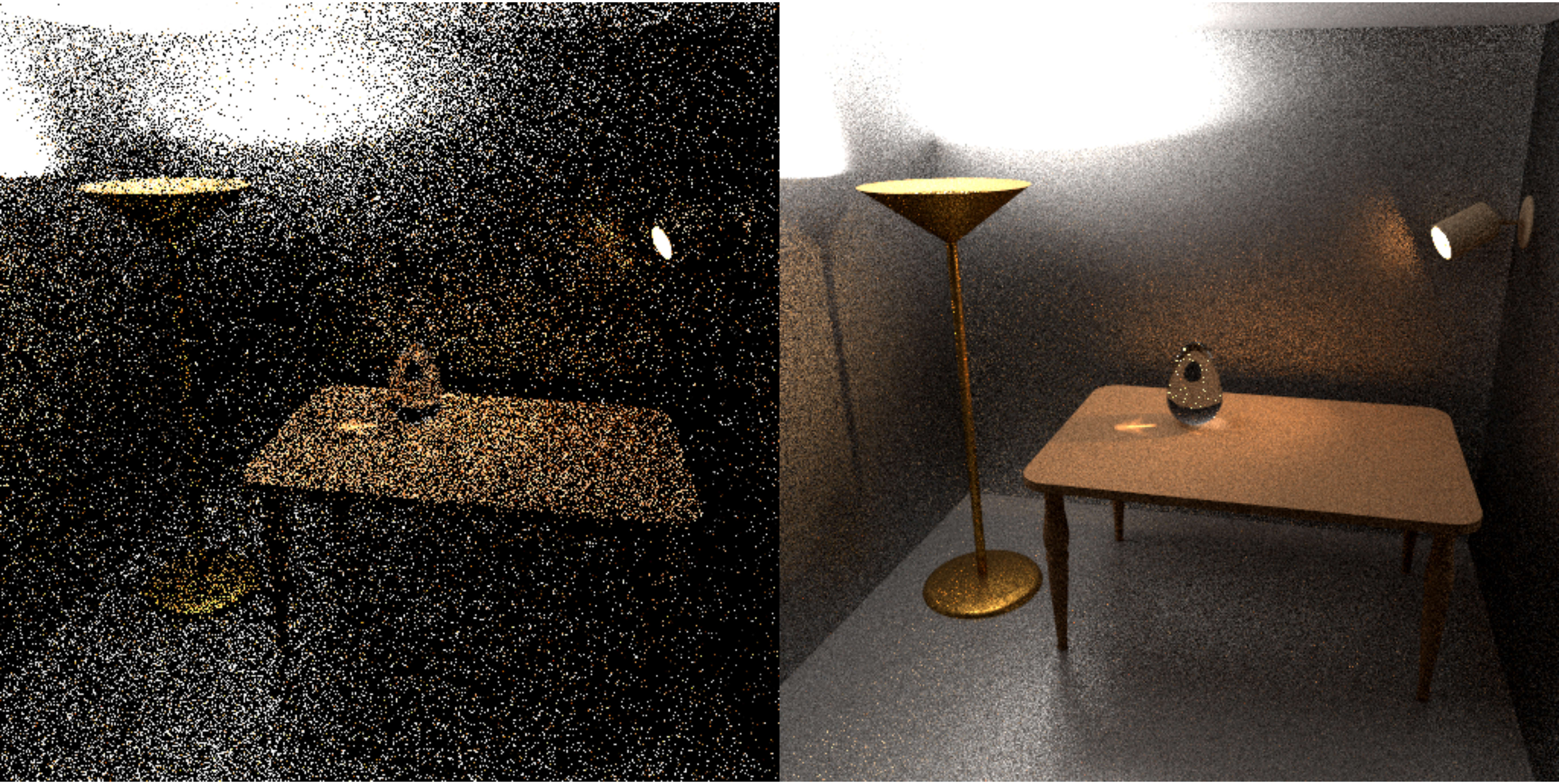
BDPT cont'd

- It's still just integrating over paths, just with more complex PDFs used for generating the paths
  - This is non-trivial; you have to be able to enumerate all the ways a given path can be constructed
  - E.g. a path with 4 segments can be constructed
    - Entirely from the eye
    - Entirely from the light
    - One segment from light, three from camera
    - Two from each
    - Three segments from light, one from camera
  - And each have their own PDFs
- We'll come to this soon!

BDPT Caustics vs. PT



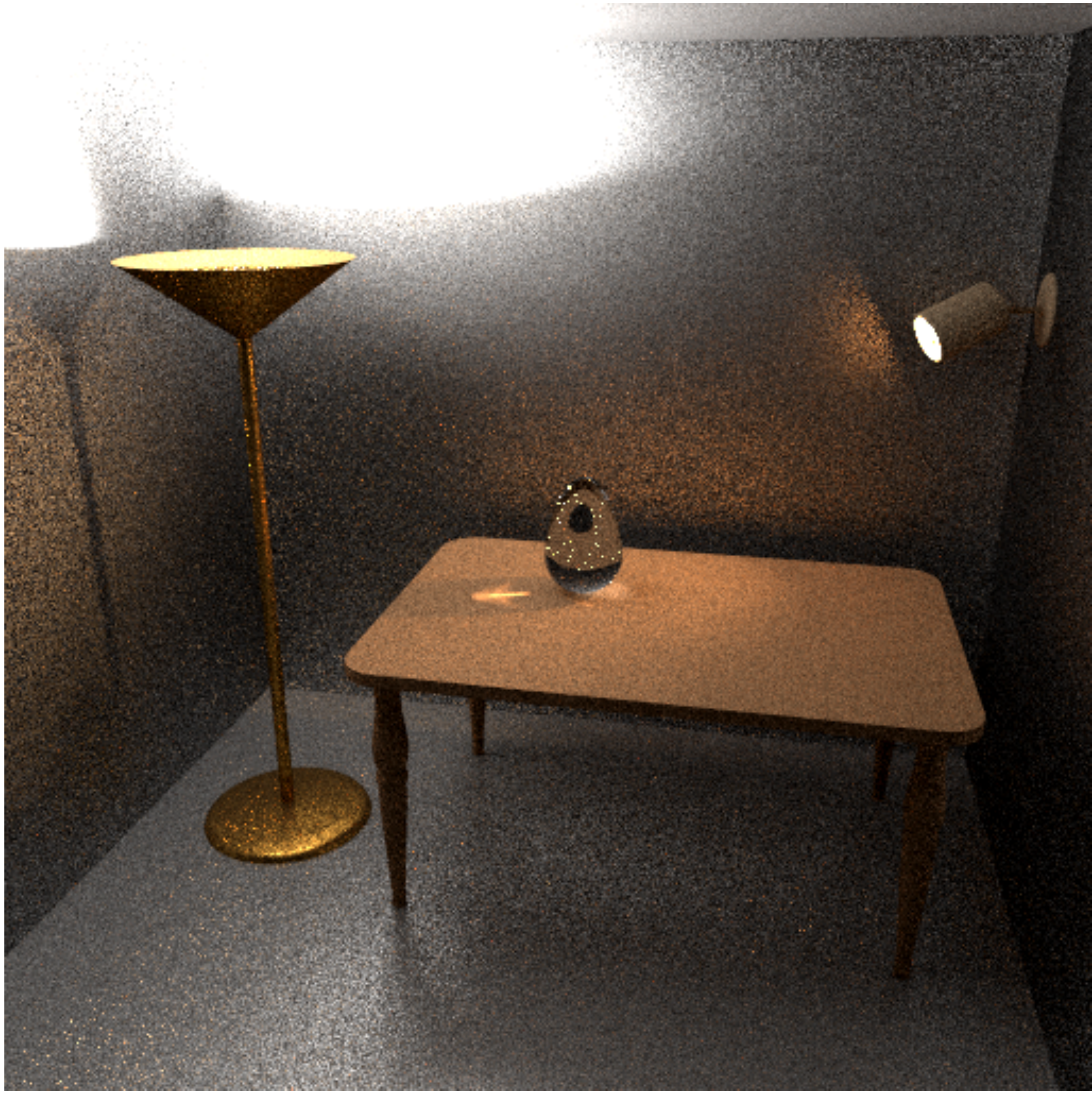
PT vs. BDPT, 5s Time Budget



Images: Dietger van Antwerpen

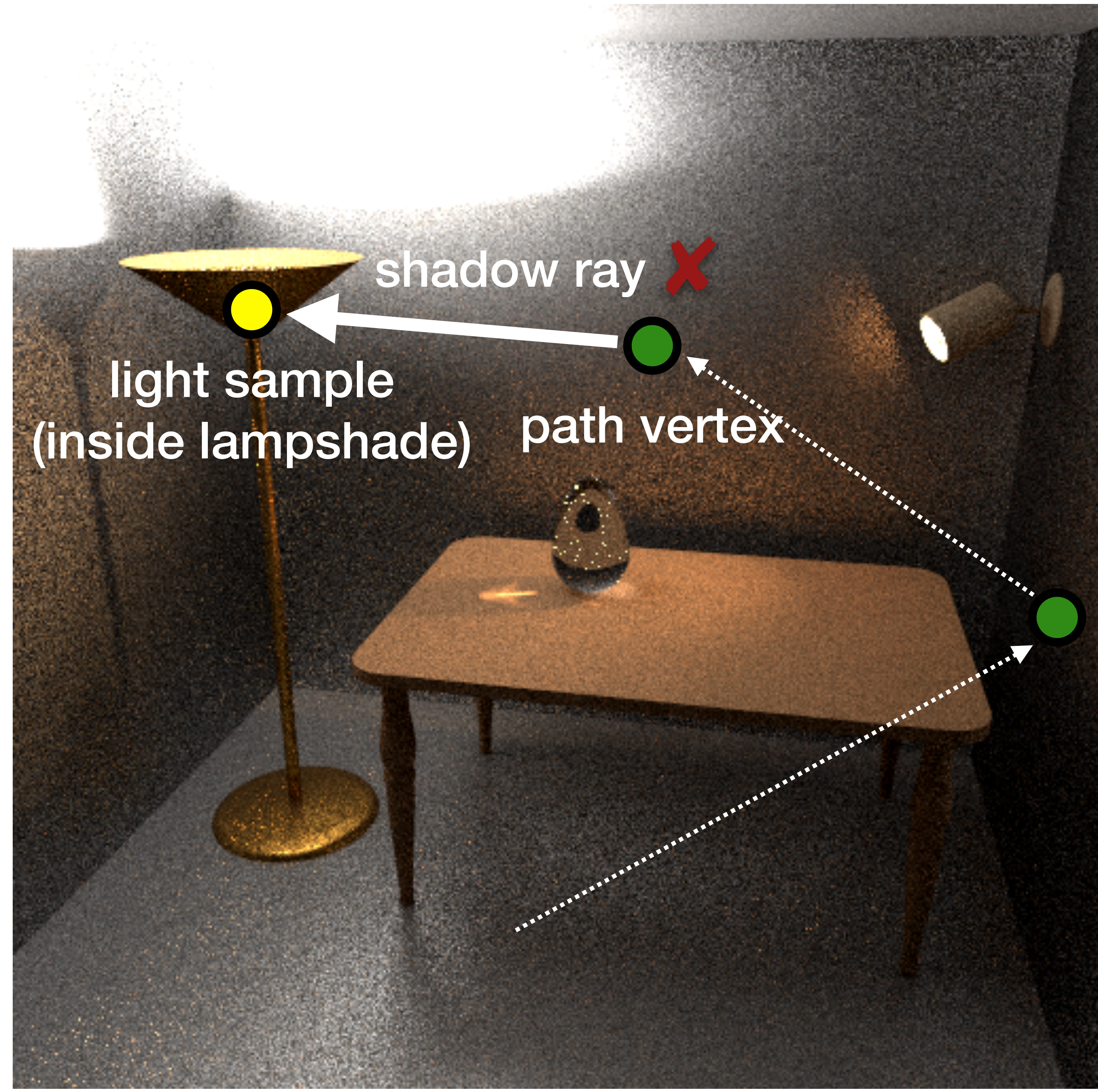
Why Does BDPT Do So Much Better?

- Most of the light is indirect, reflected from the lighting fixtures
  - The actual emitting sources are small
  - Effectively, the fixtures ("varjostin") turn the direct lights into small indirect sources
  - And as you remember, the path tracer can't deal with those!
    - Shadow rays only towards emitters



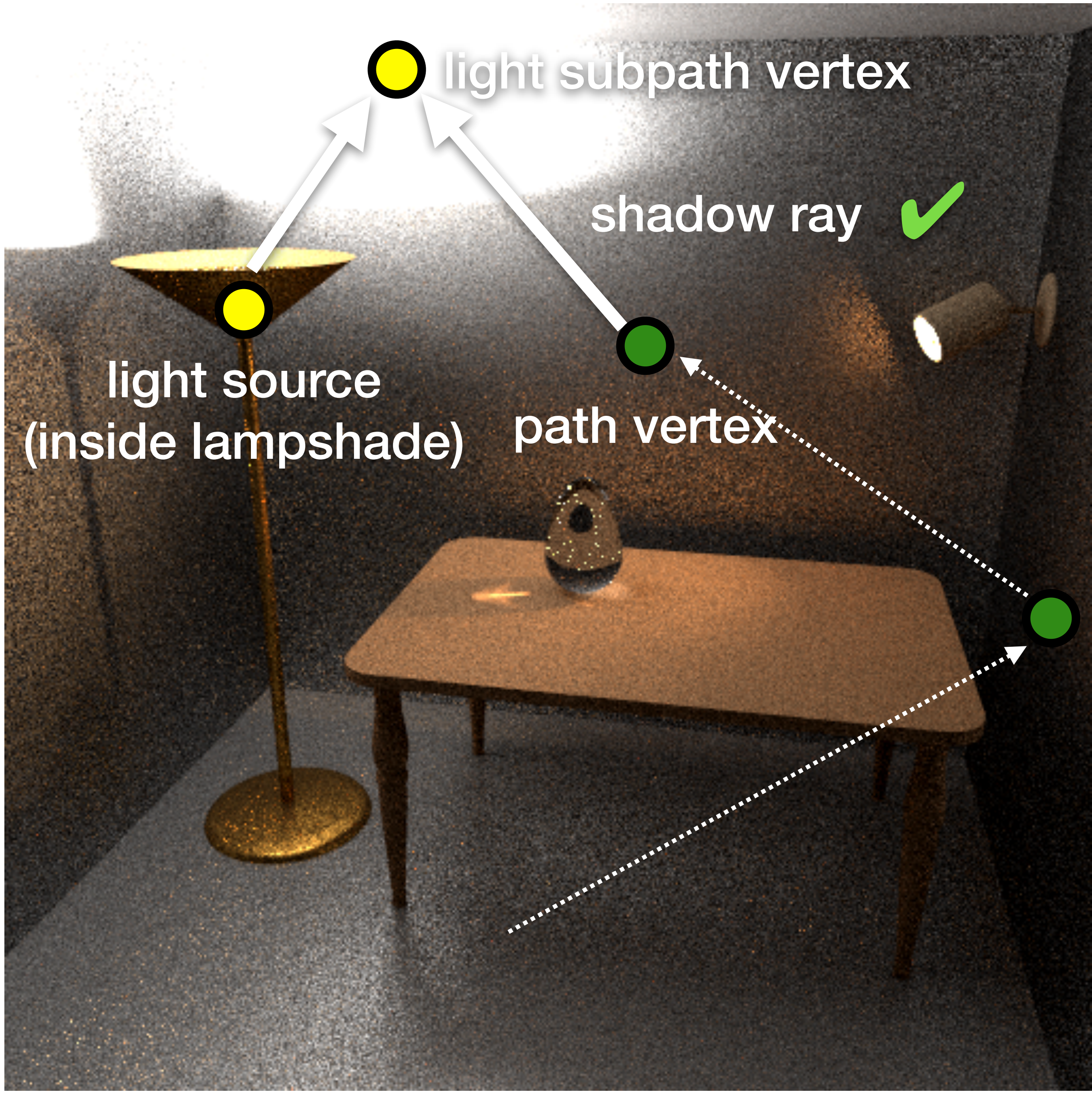
Why Does BDPT Do So Much Better?

- Most of the light is indirect, reflected from the lighting fixtures
  - The actual emitting sources are small
  - Effectively, the fixtures ("varjostin") turn the direct lights into small indirect sources
  - And as you remember, the path tracer can't deal with those!
    - Shadow rays only towards emitters

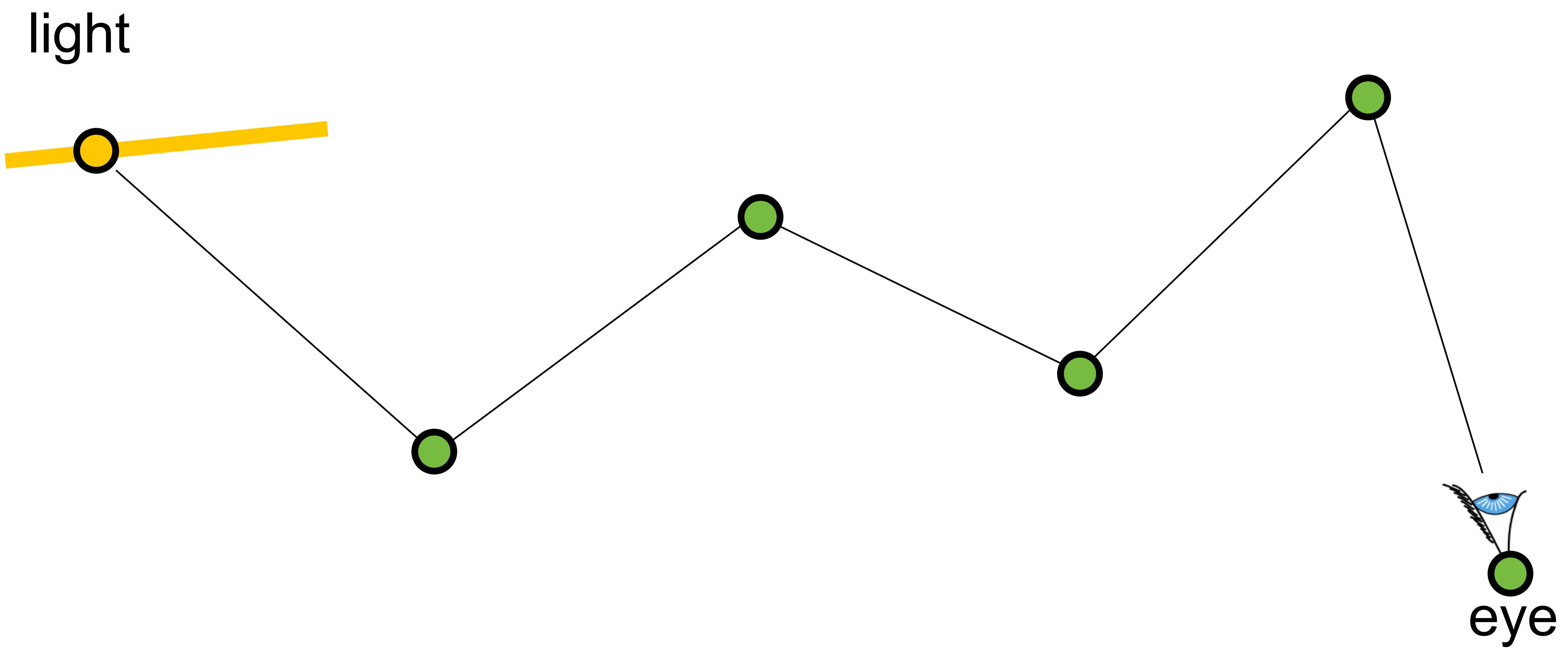


Why Does BDPT Do So Much Better?

- Most of the light is indirect, reflected from the lighting fixtures
  - The actual emitting sources are small
  - Effectively, the fixtures ("varjostin") turn the direct lights into small indirect sources
  - And as you remember, the path tracer can't deal with those!
    - Shadow rays only towards emitters

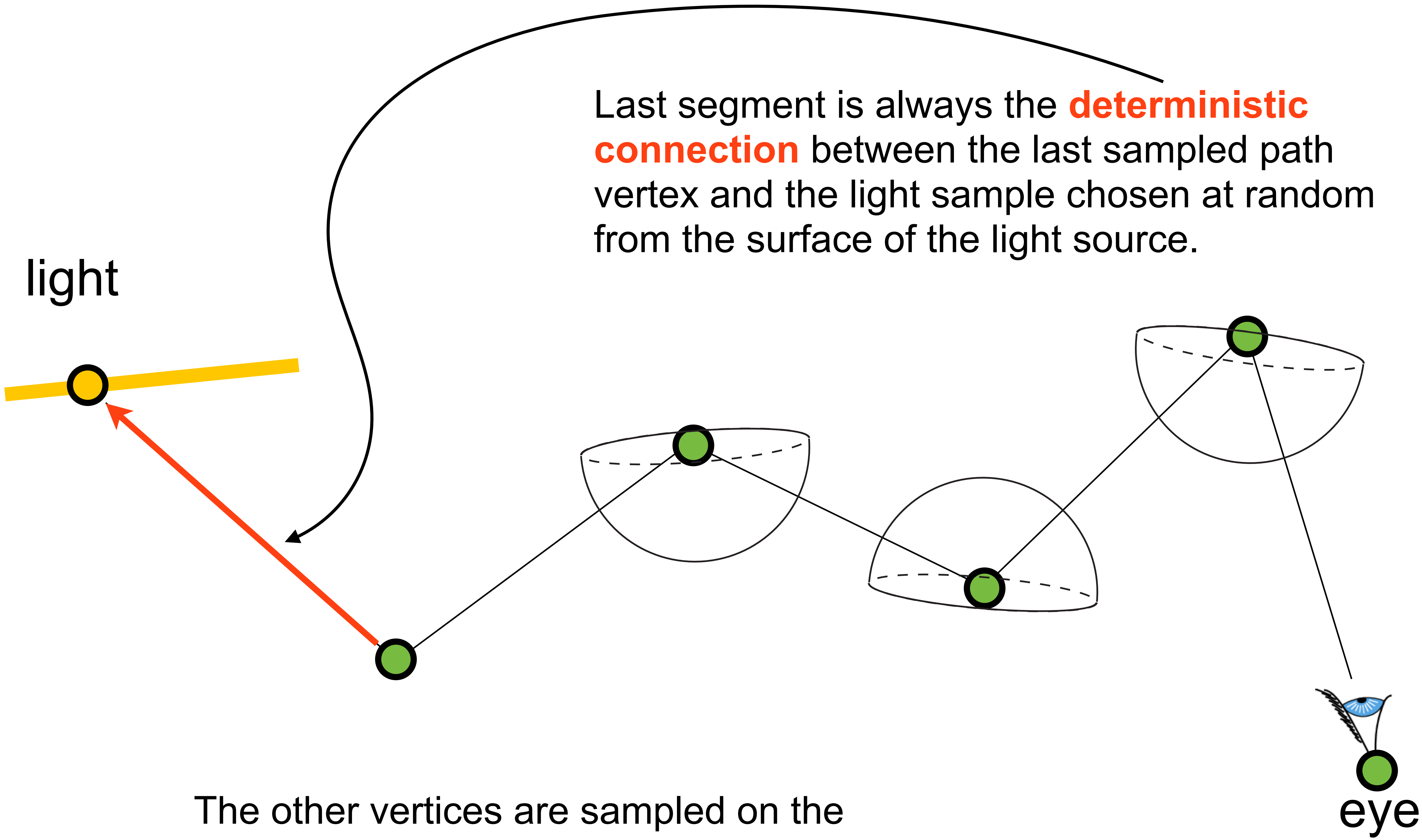


Questions?


light
eye

Regular Path Tracing

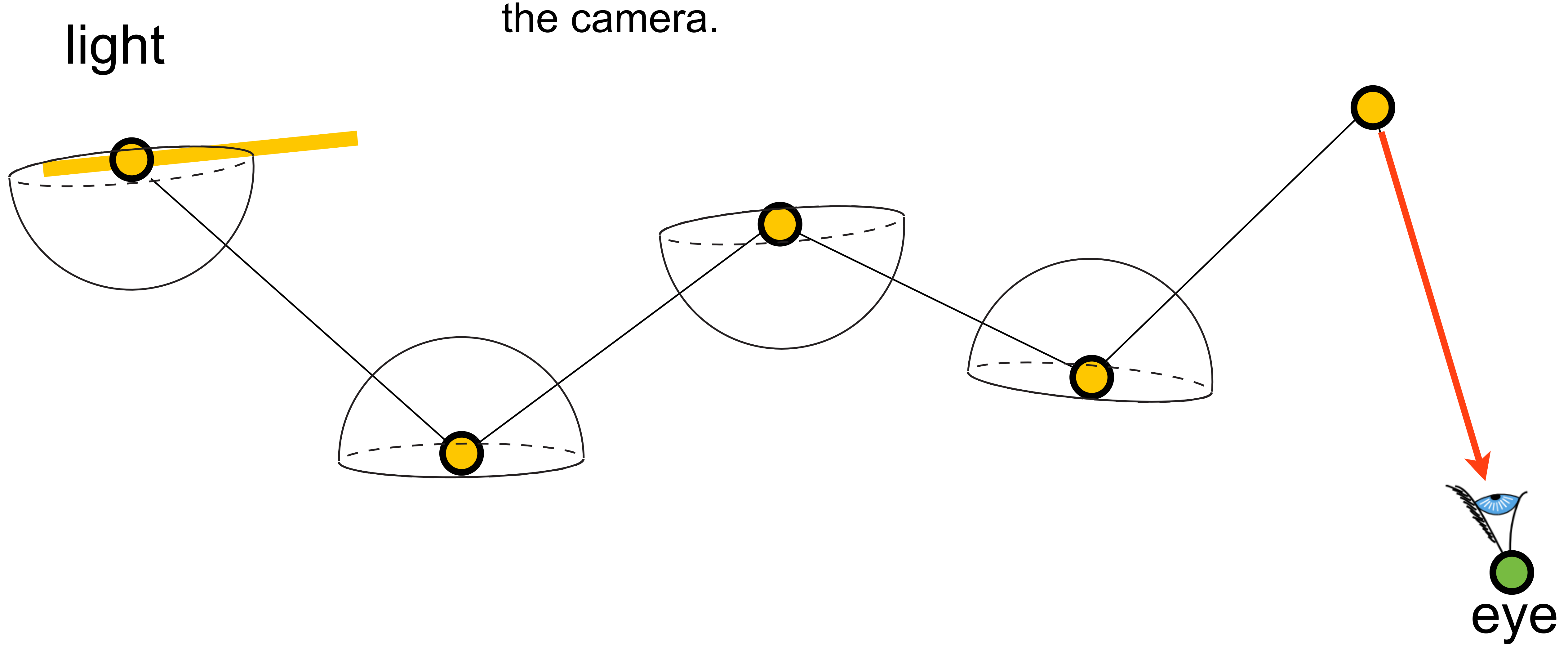
Last segment is always the **deterministic connection** between the last sampled path vertex and the light sample chosen at random from the surface of the light source.



The other vertices are sampled on the hemisphere at the previous vertex.

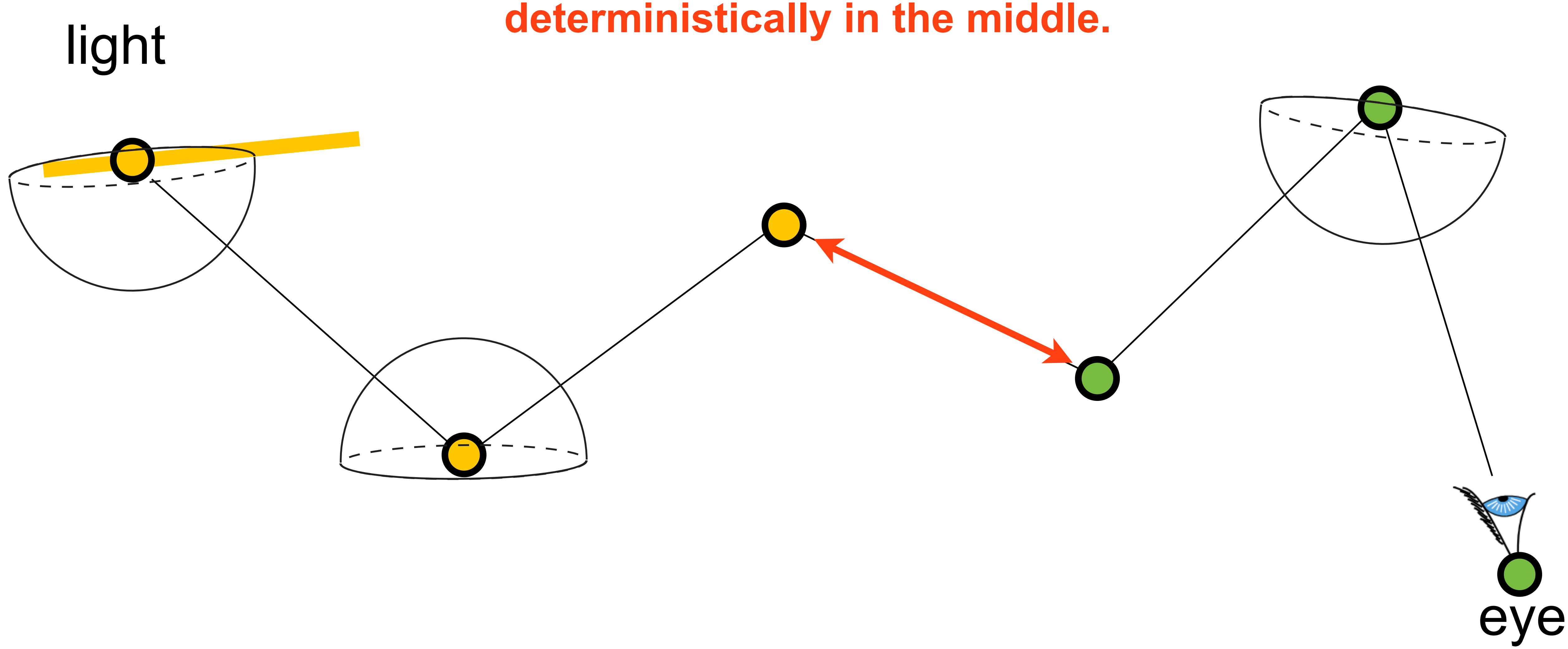
Sampling From the Light

The same exact path can be sampled also by constructing a chain from the light, and connecting its endpoint **deterministically** to the camera.



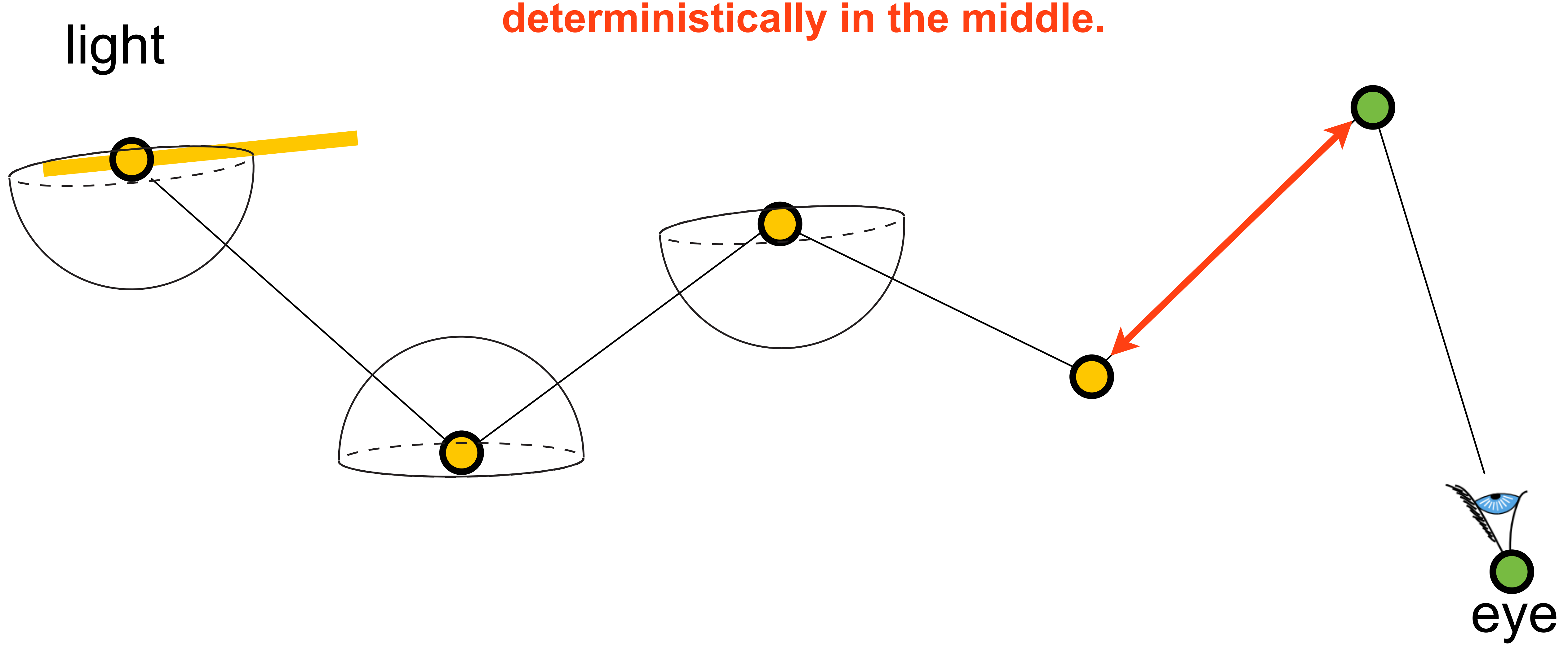
Bidirectional Sampling

The same exact path can be sampled also by constructing one chain from the light, another from the camera, and **connecting them deterministically in the middle.**



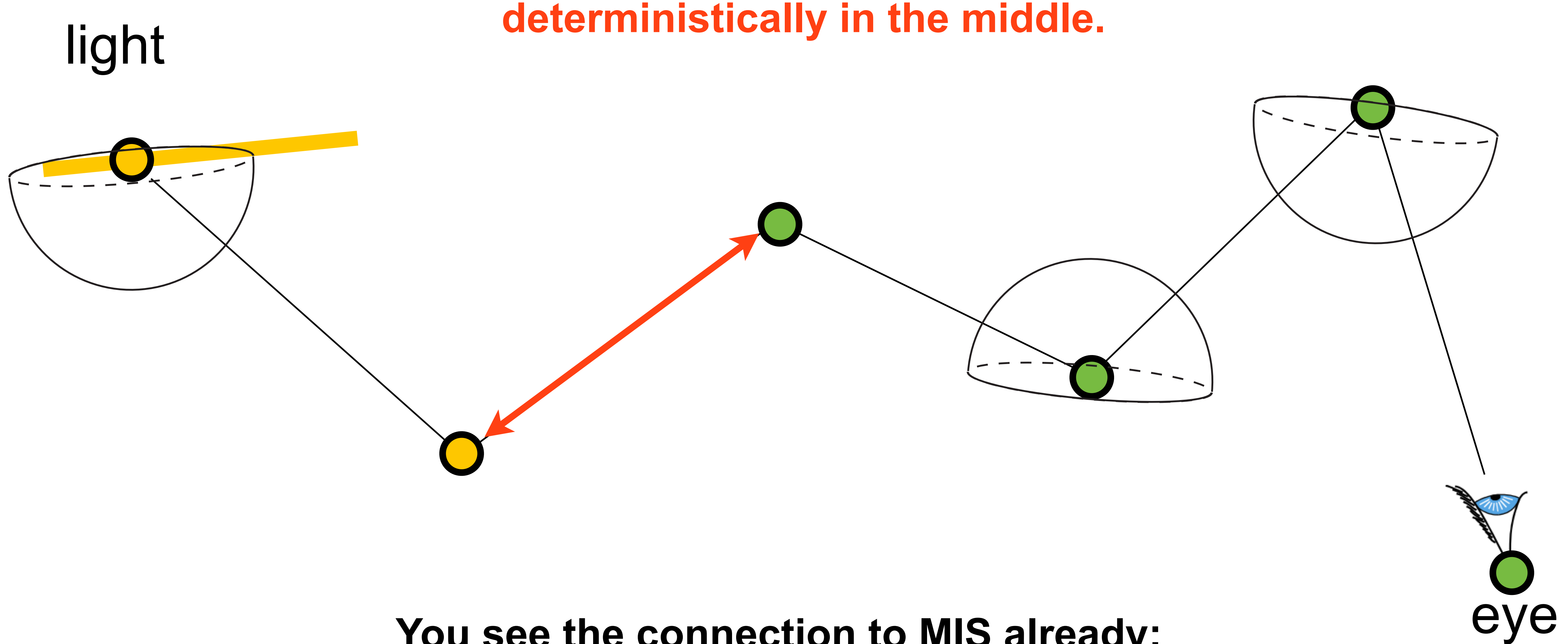
Bidirectional Sampling, Take Two

The same exact path can be sampled also by constructing one chain from the light, another from the camera, and **connecting them deterministically in the middle.**



Bidirectional Sampling, Take Three

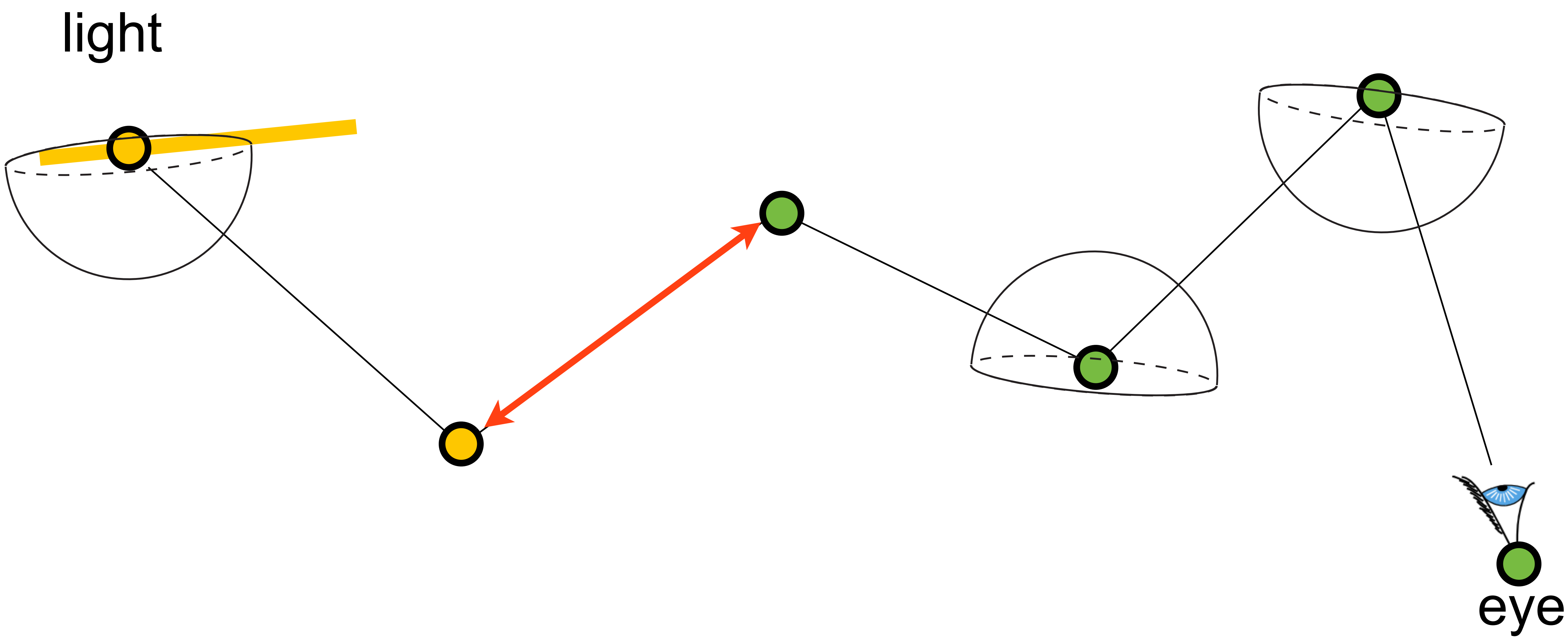
The same exact path can be sampled also by constructing one chain from the light, another from the camera, and **connecting them deterministically in the middle.**



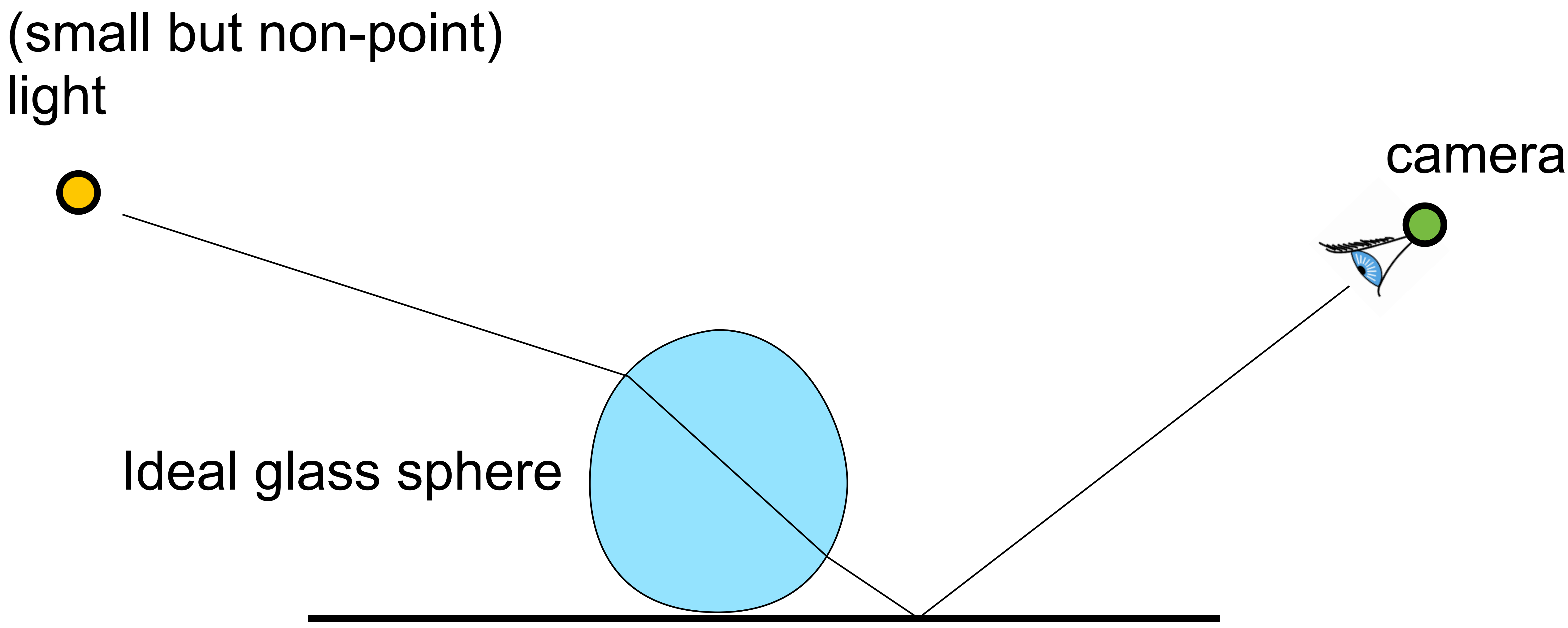
You see the connection to MIS already: multiple ways of sampling the same path.

MIS

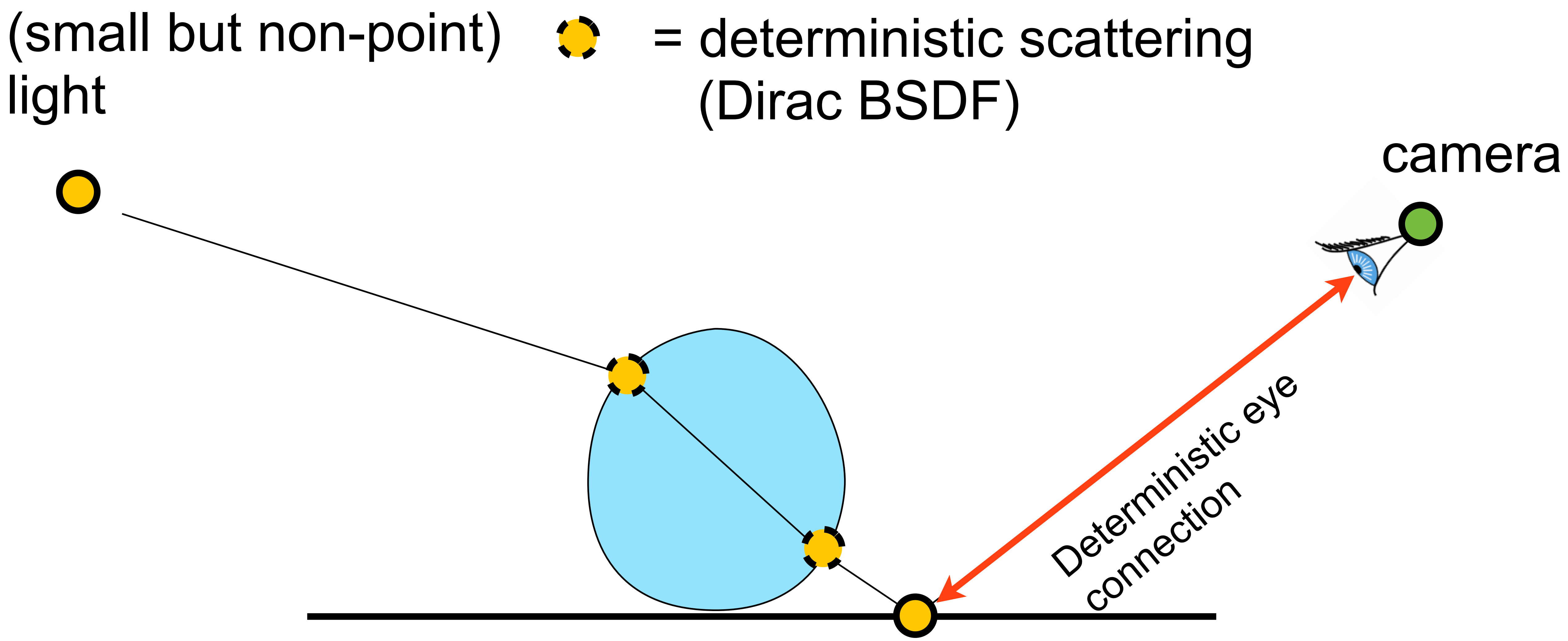
All these variants have a different PDF.



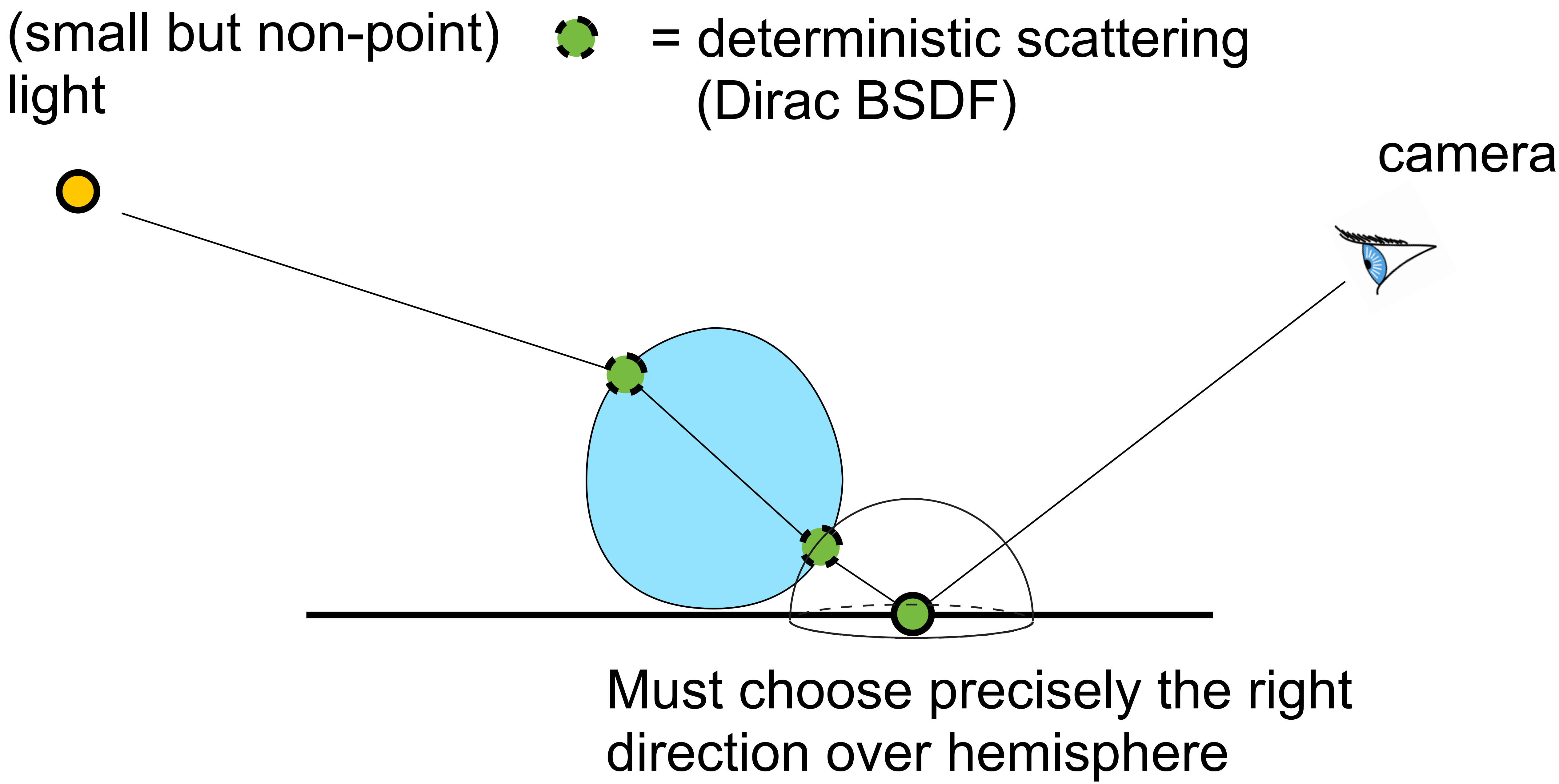
Example: Caustic Path



Easy To Sample From The Light

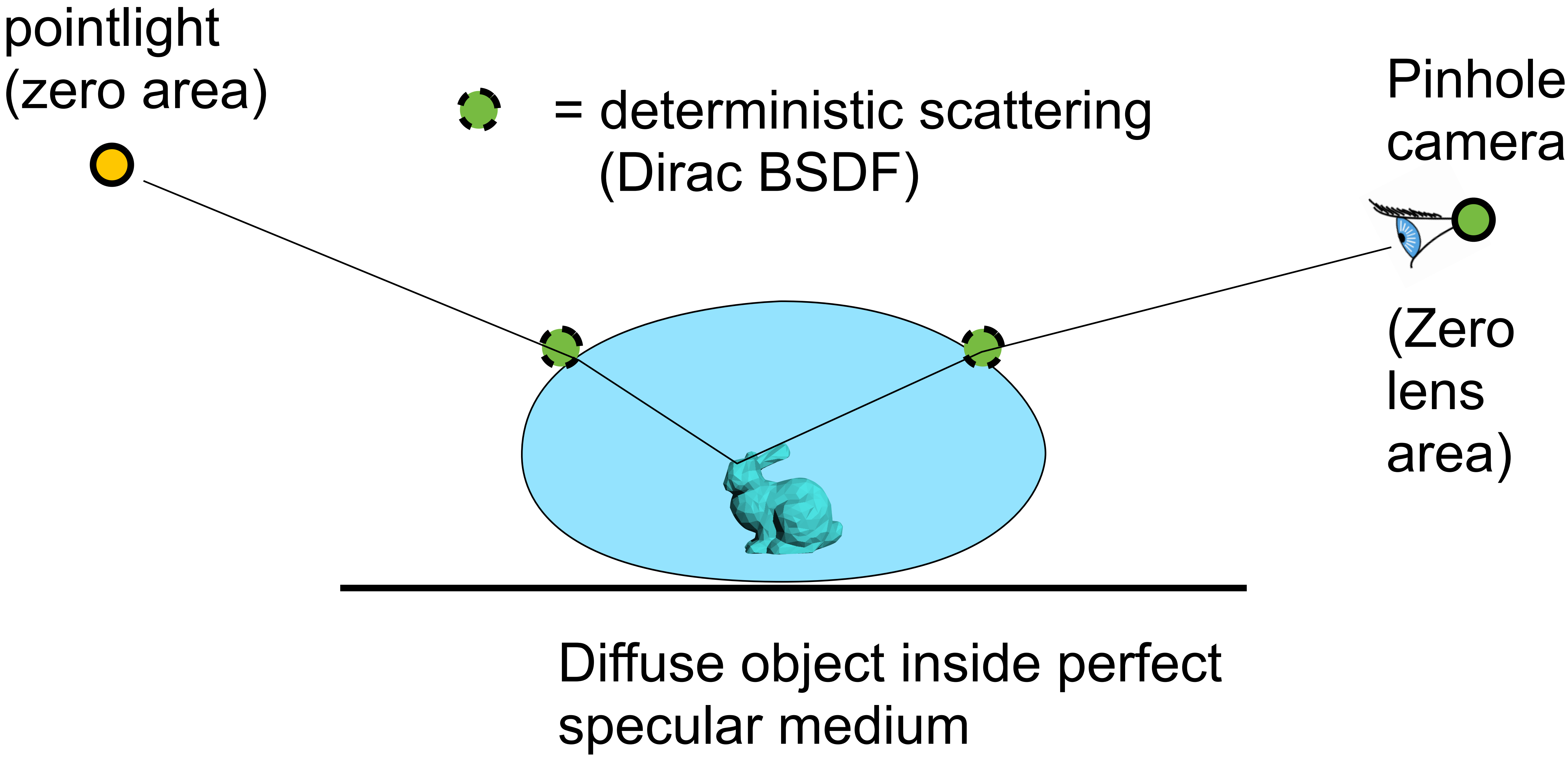


Almost Impossible The Other Way



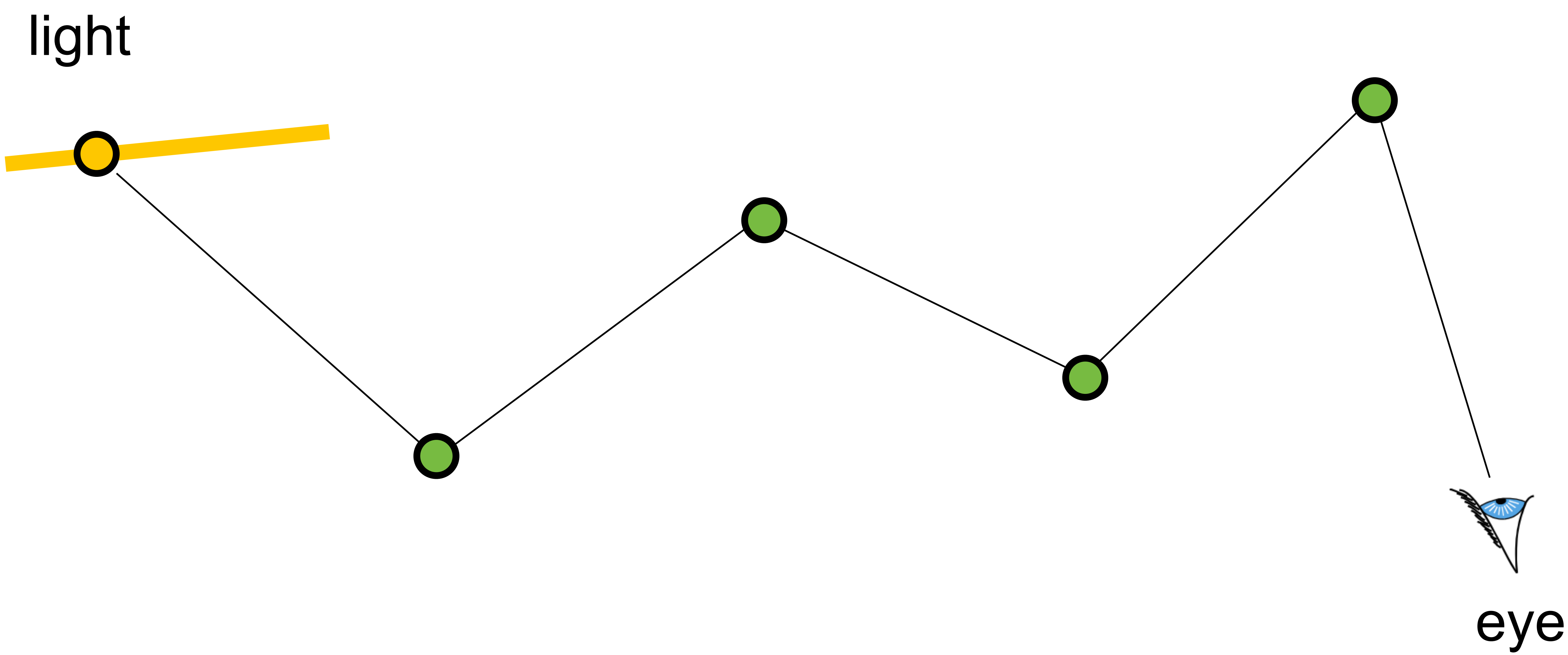
Some Paths **ARE** impossible!

- With a point light and pinhole camera, how do you sample this path sequentially? You don't!



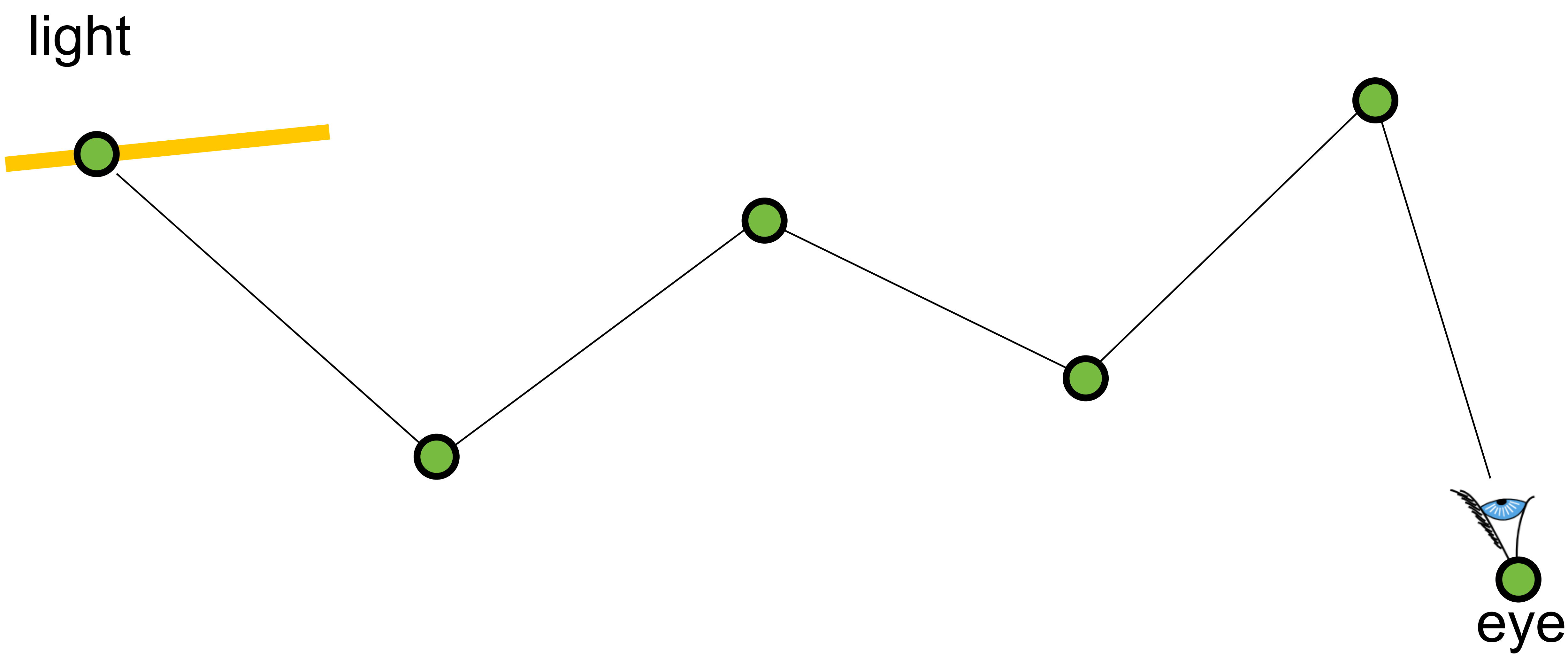
k, m -samplers [Veach 1995]

- To sample a path with k segments, there are generally k+2 different ways to sample it (here k=5)
  - $m = 0, \ldots, k+1$ vertices from the light



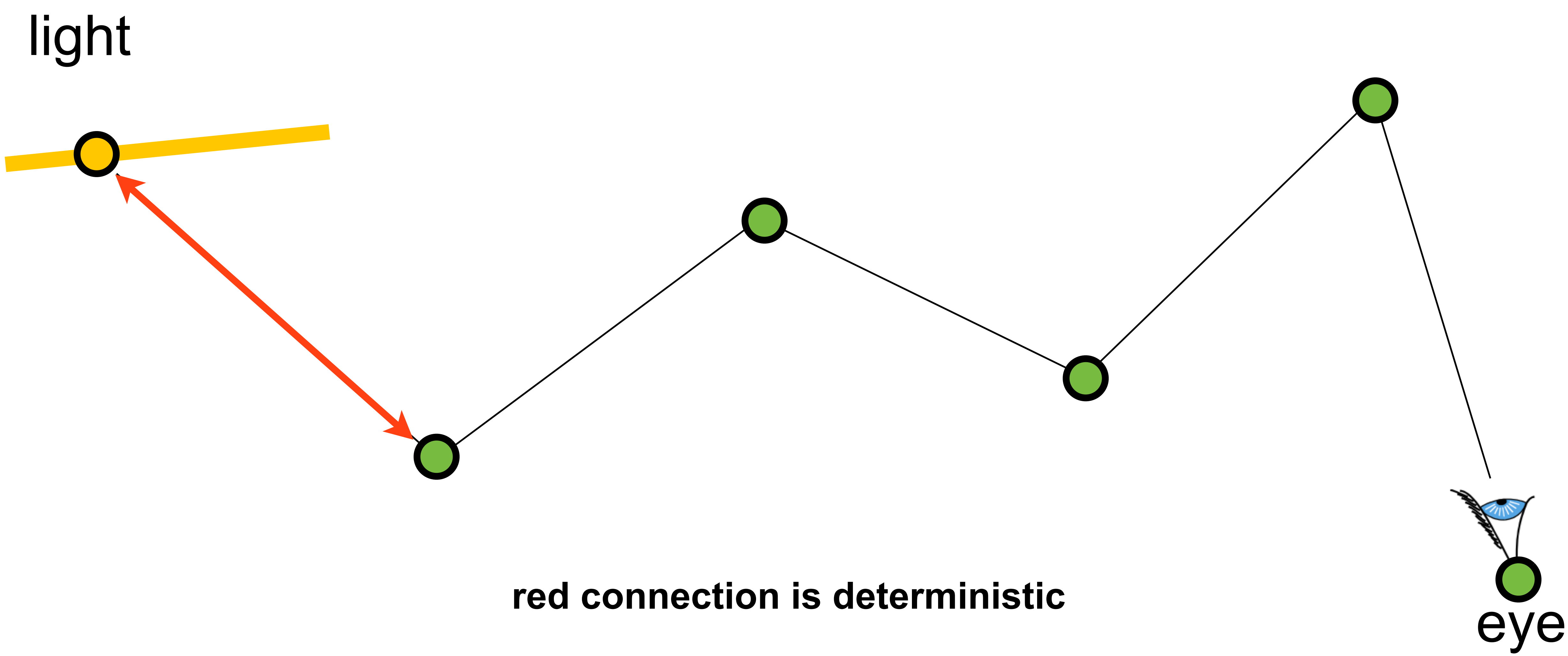
k=5, m=0

- All vertices from the camera, no shadow rays
  - Wait to hit light by chance = brute force path tracing
  - *Impossible with pointlights*



k=5, m=1

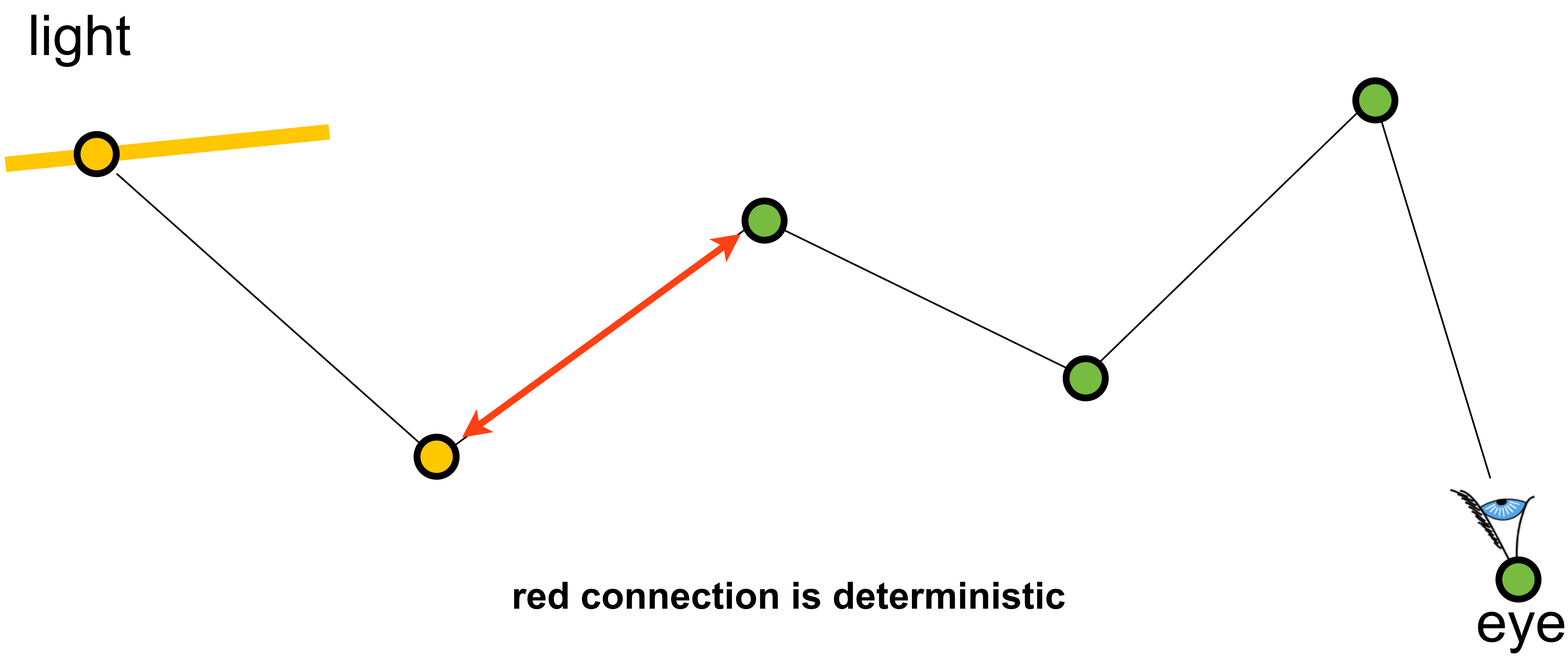
- Pick point on light source, sample rest of vertices from camera, connect last camera vertex to only light vertex
  - This is regular path tracing with shadow rays



red connection is deterministic

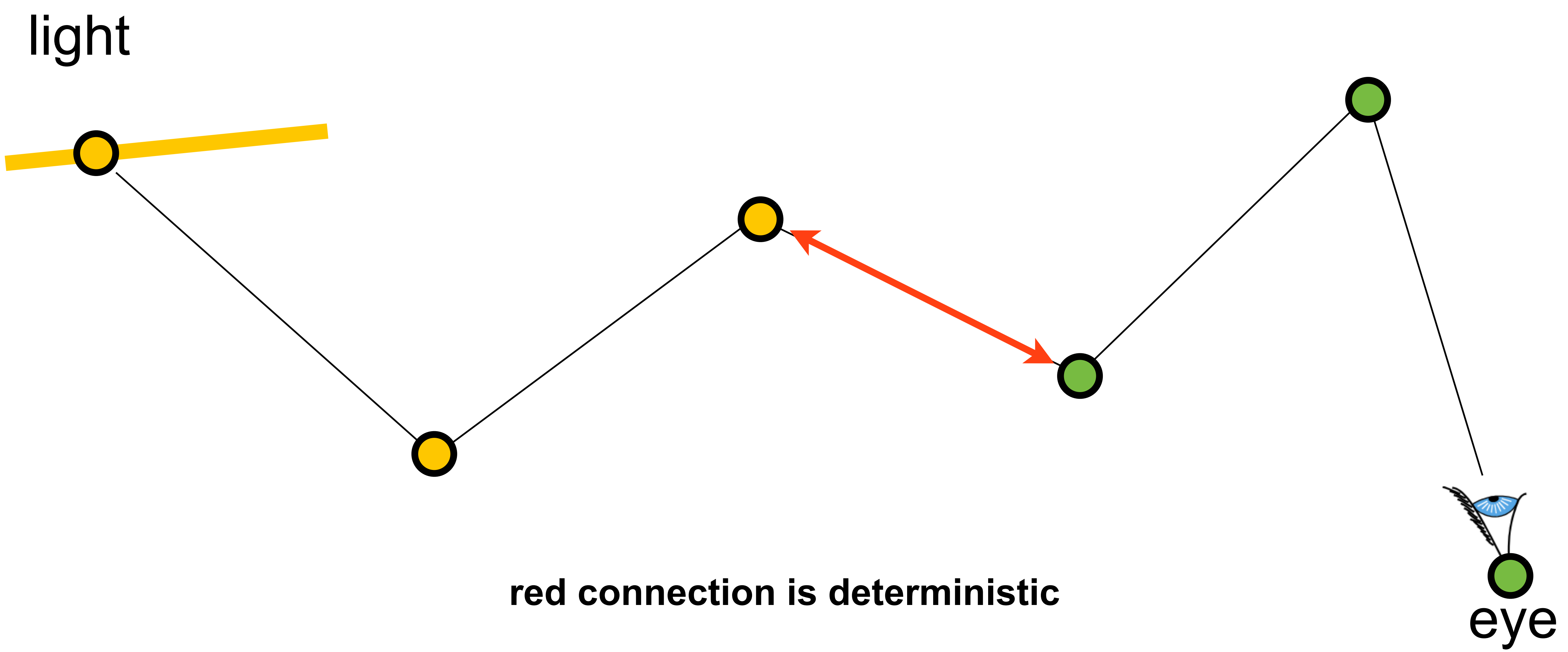
k=5, m=2

- Two vertices from light: Pick point on light source, pick random direction, trace ray



k=5, m=3

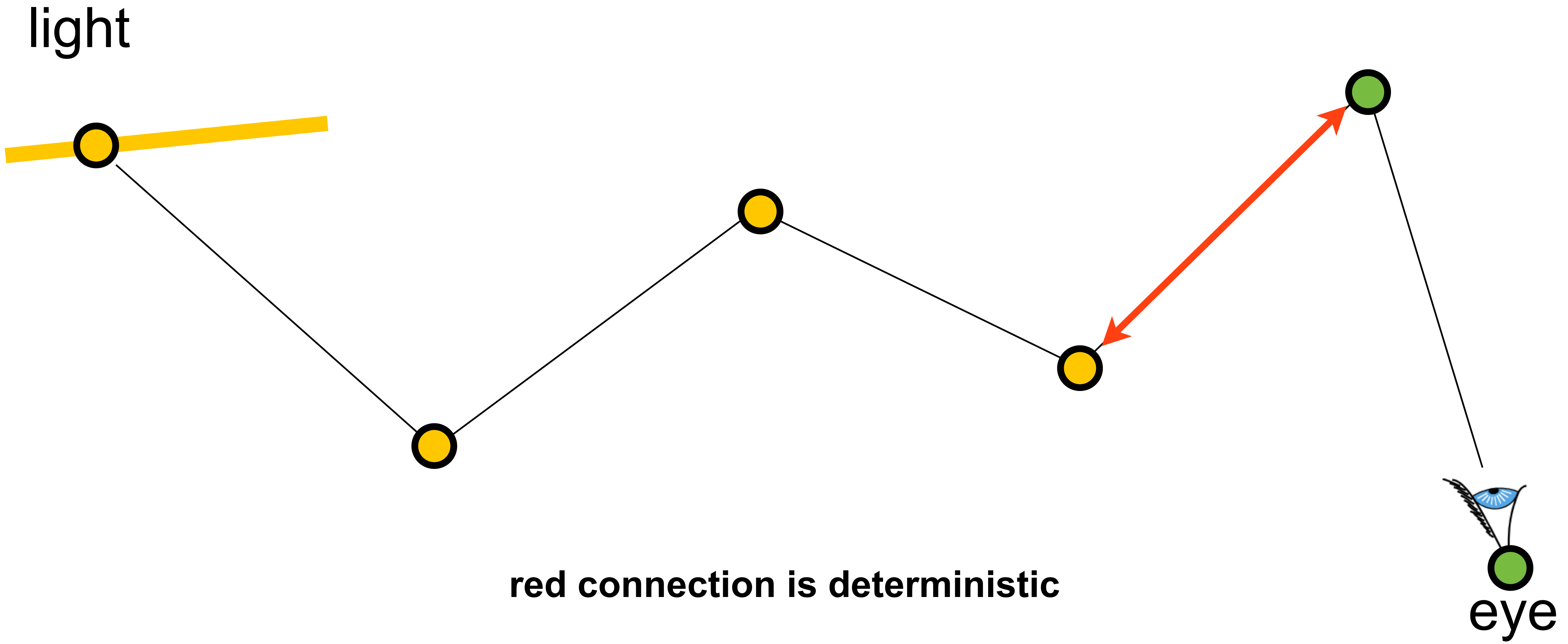
- Shoot one indirect bounce from light



red connection is deterministic

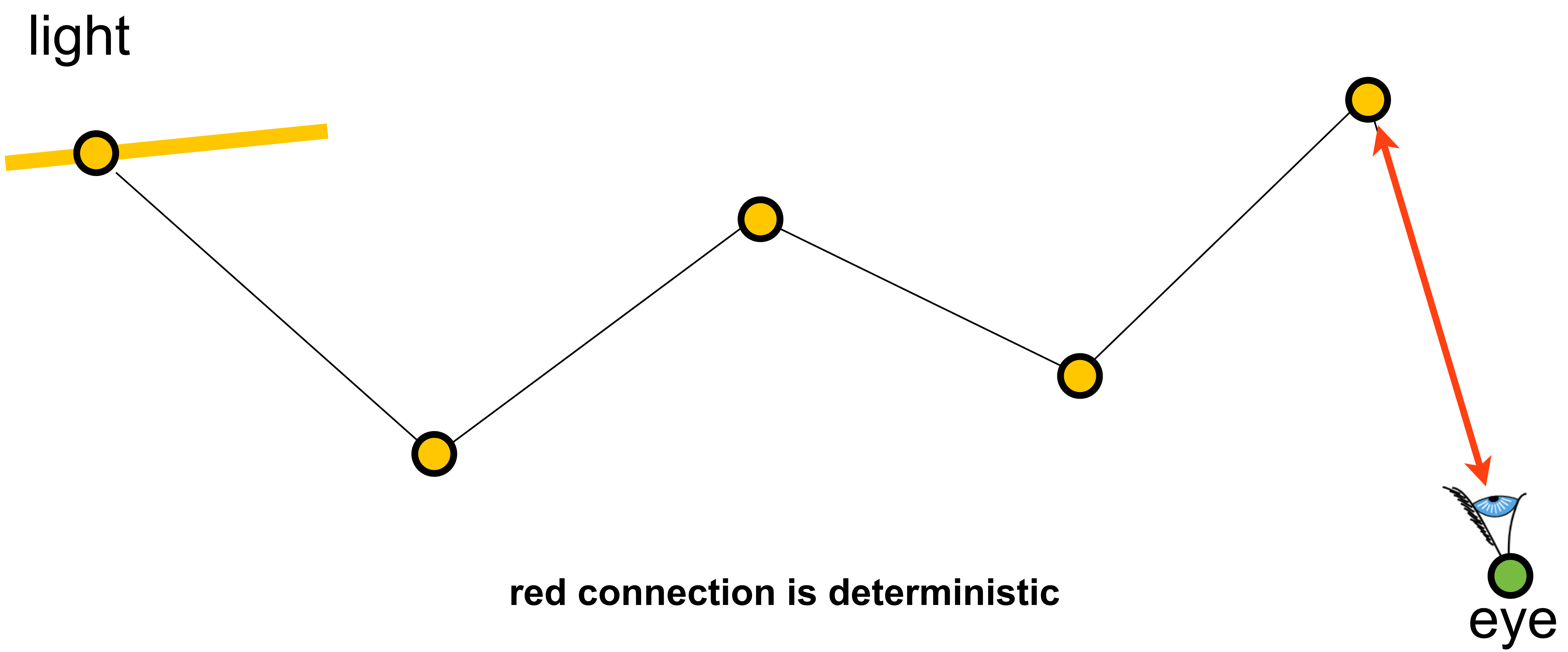
k=5, m=4

- Two indirect bounces from light



k=5, m=5

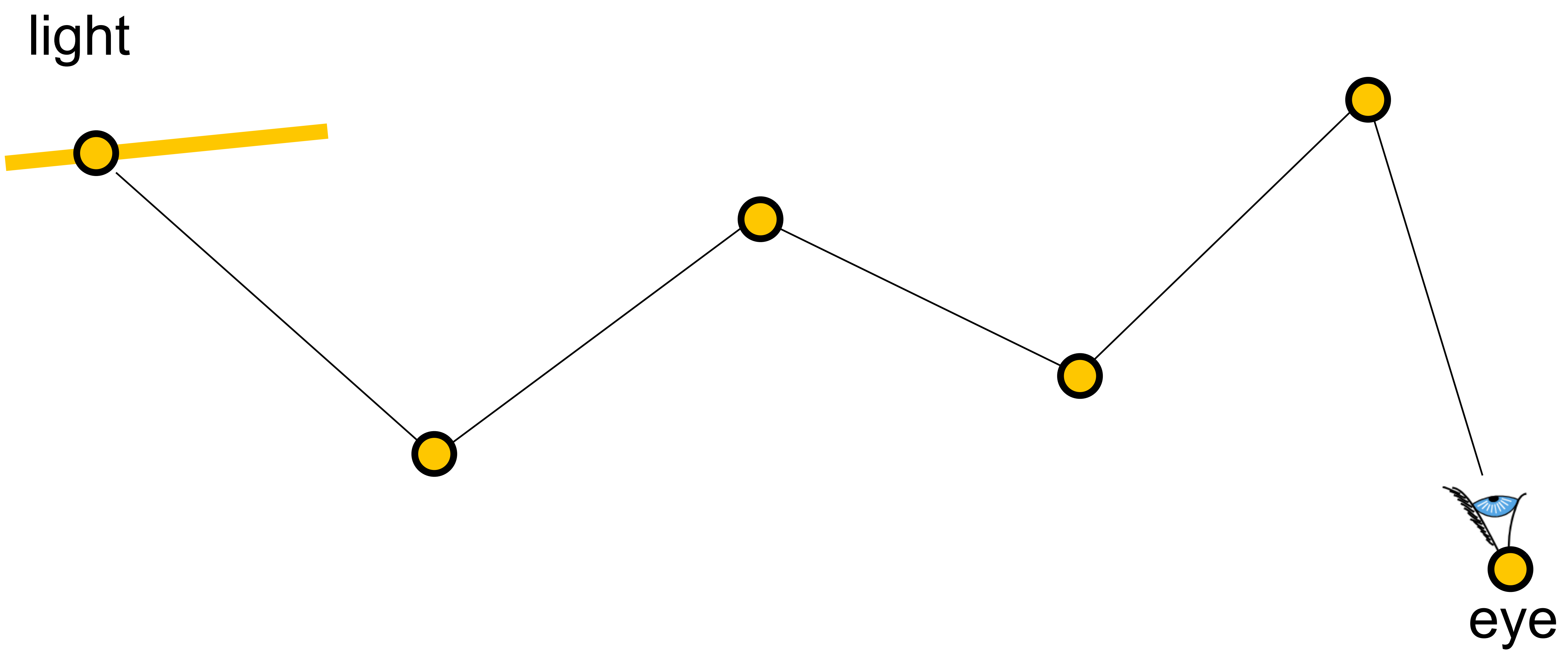
- Three light bounces, no camera rays at all!



red connection is deterministic

k=5, m=6

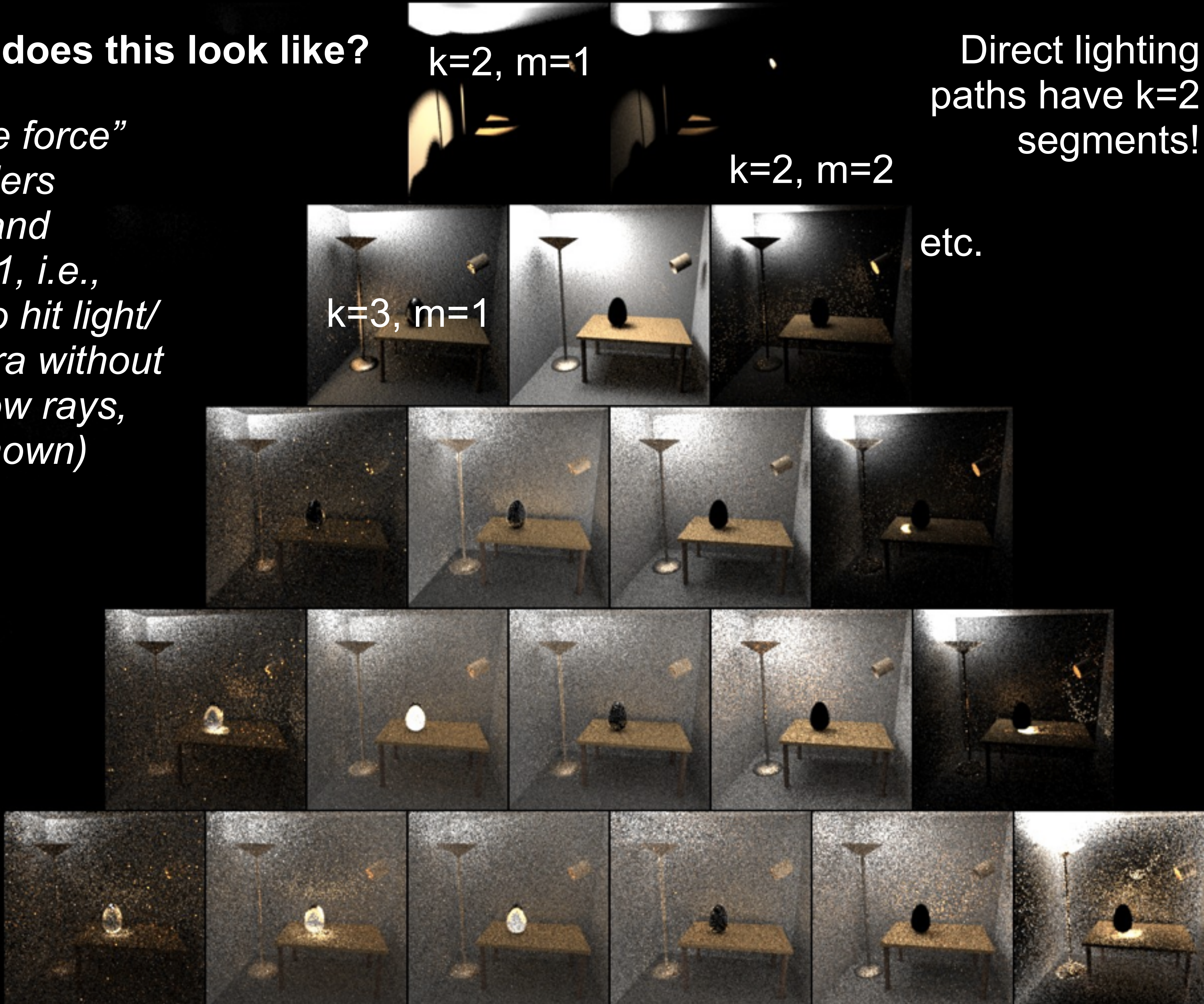
- Wait to hit camera by chance!
  - *Impossible with a pointlike camera without finite aperture!*
  - Symmetric to the brute force path tracing case m=0



What does this look like?

("brute force" samplers m=0 and m=k+1, i.e., wait to hit light/ camera without shadow rays, not shown)

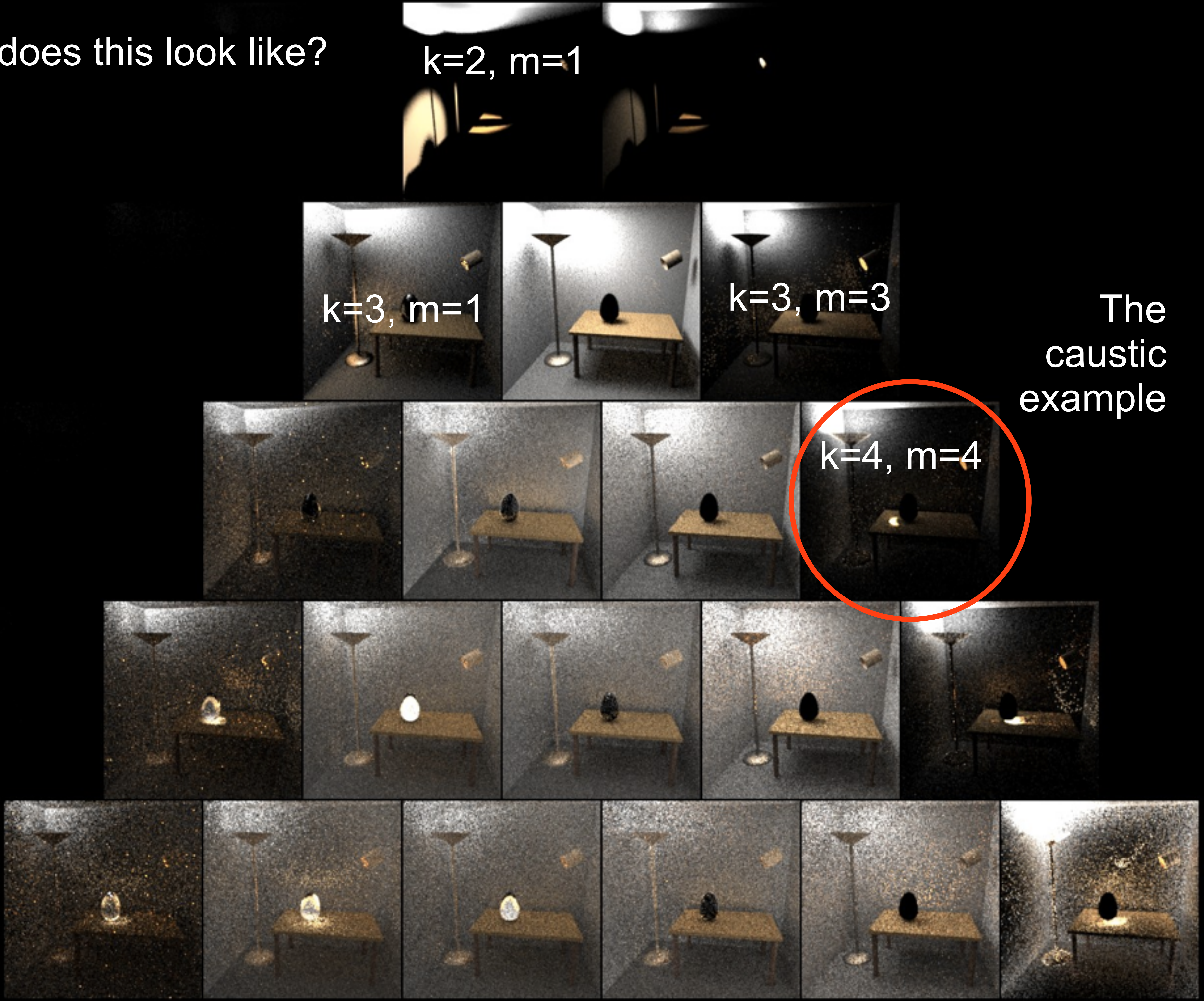
Direct lighting paths have k=2 segments!



What's Going On

- Each picture is the contribution of one particular k, m sampler to the final picture, weighted by the corresponding MIS weight
- (Also further bounces have been scaled up so that you can see something)

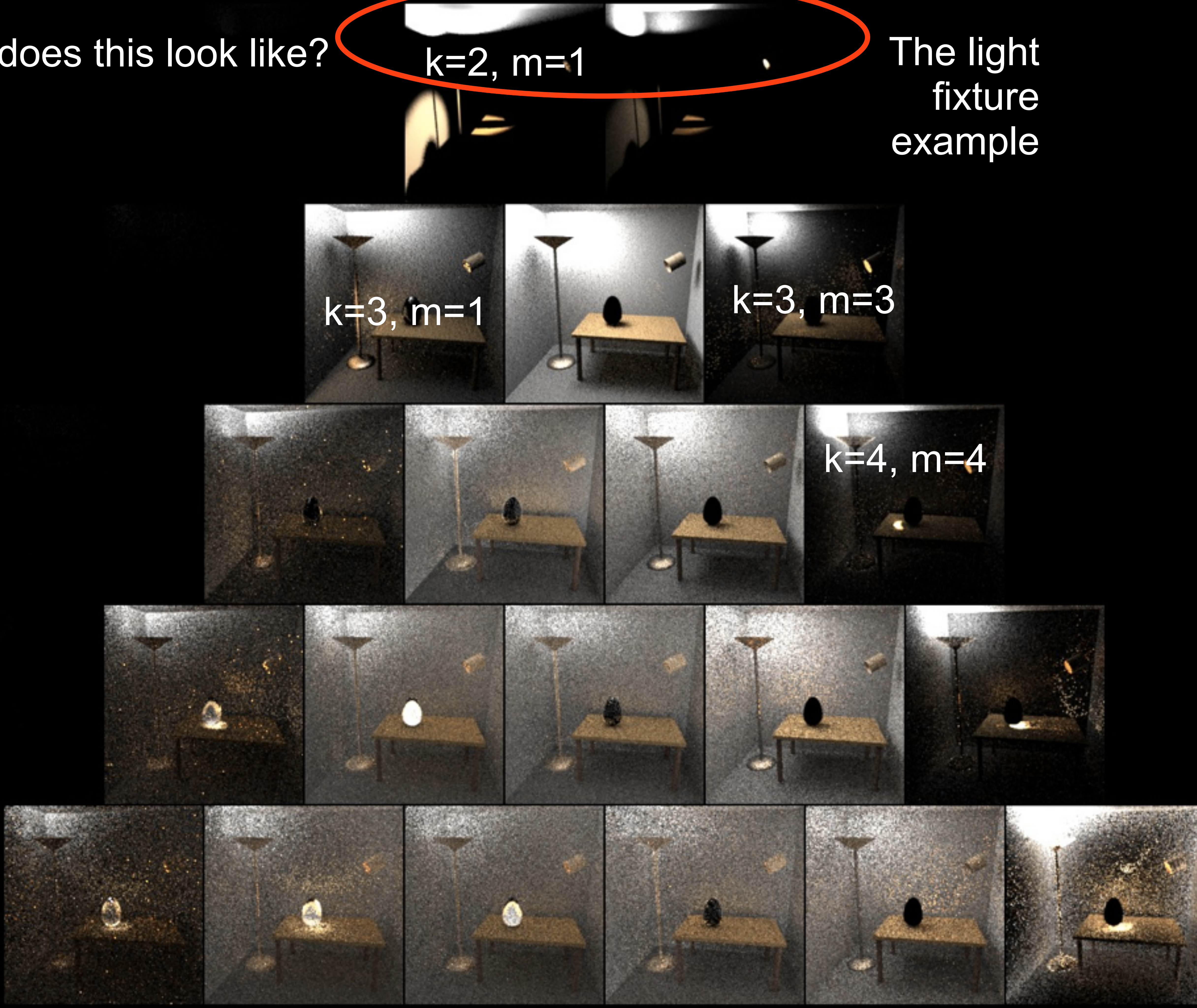
What does this look like?



The caustic example

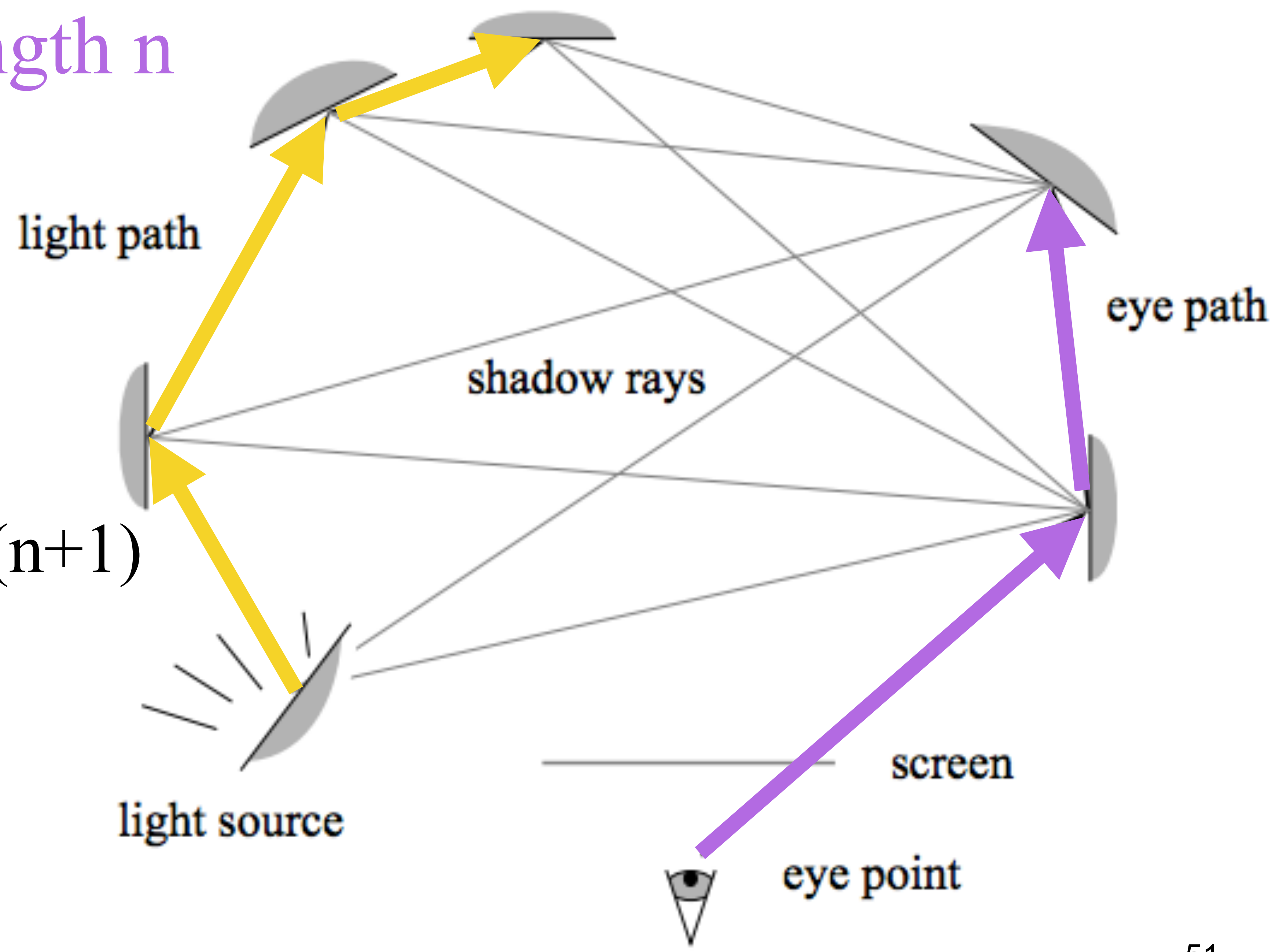
What does this look like?

The light fixture example



BDPT Recipe

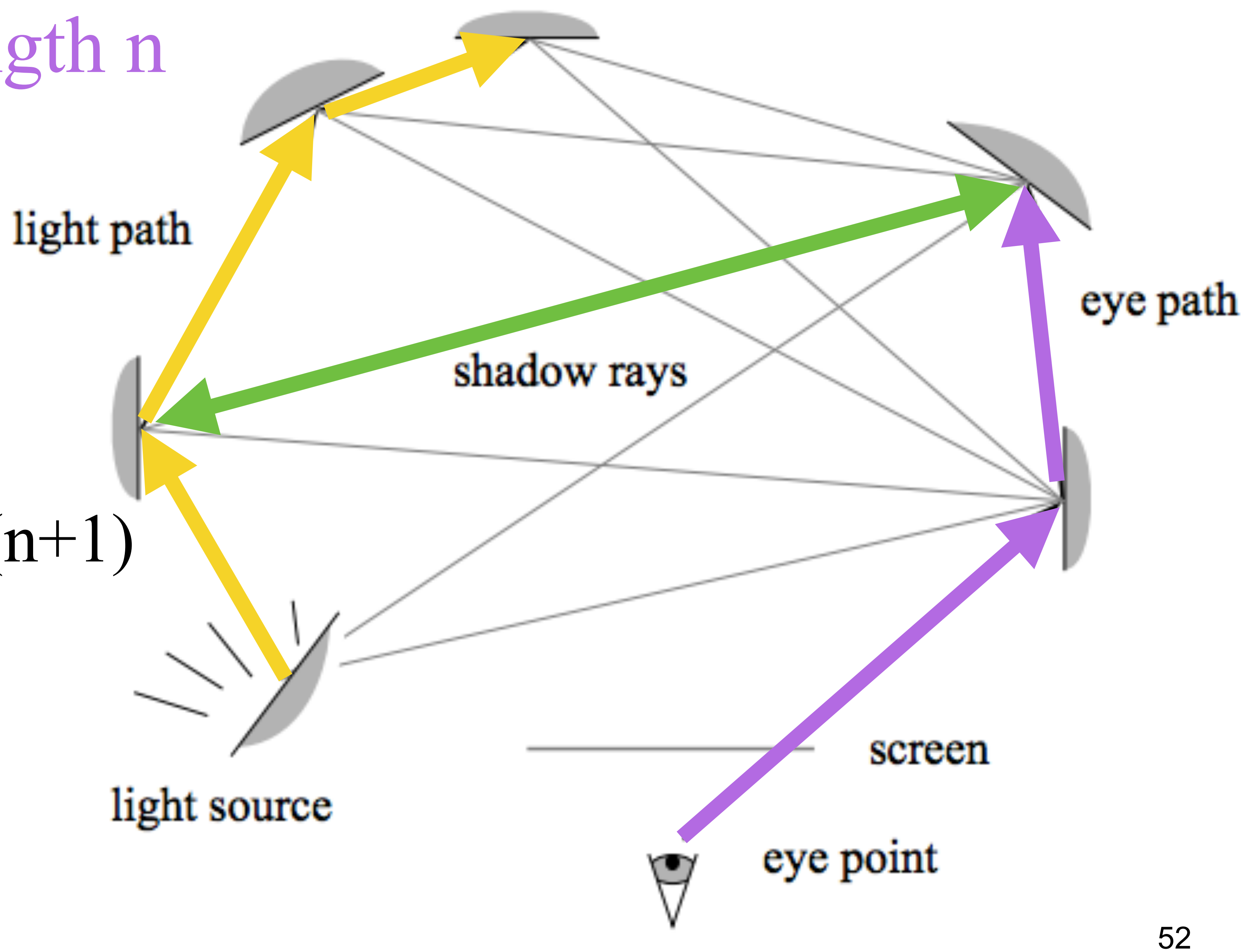
- Shoot a path of length m from light
- Shoot a path of length n from the eye
- Connect the two in all the possible ways
  - This forms (m+1)*(n+1) separate paths



BDPT Recipe

k=4

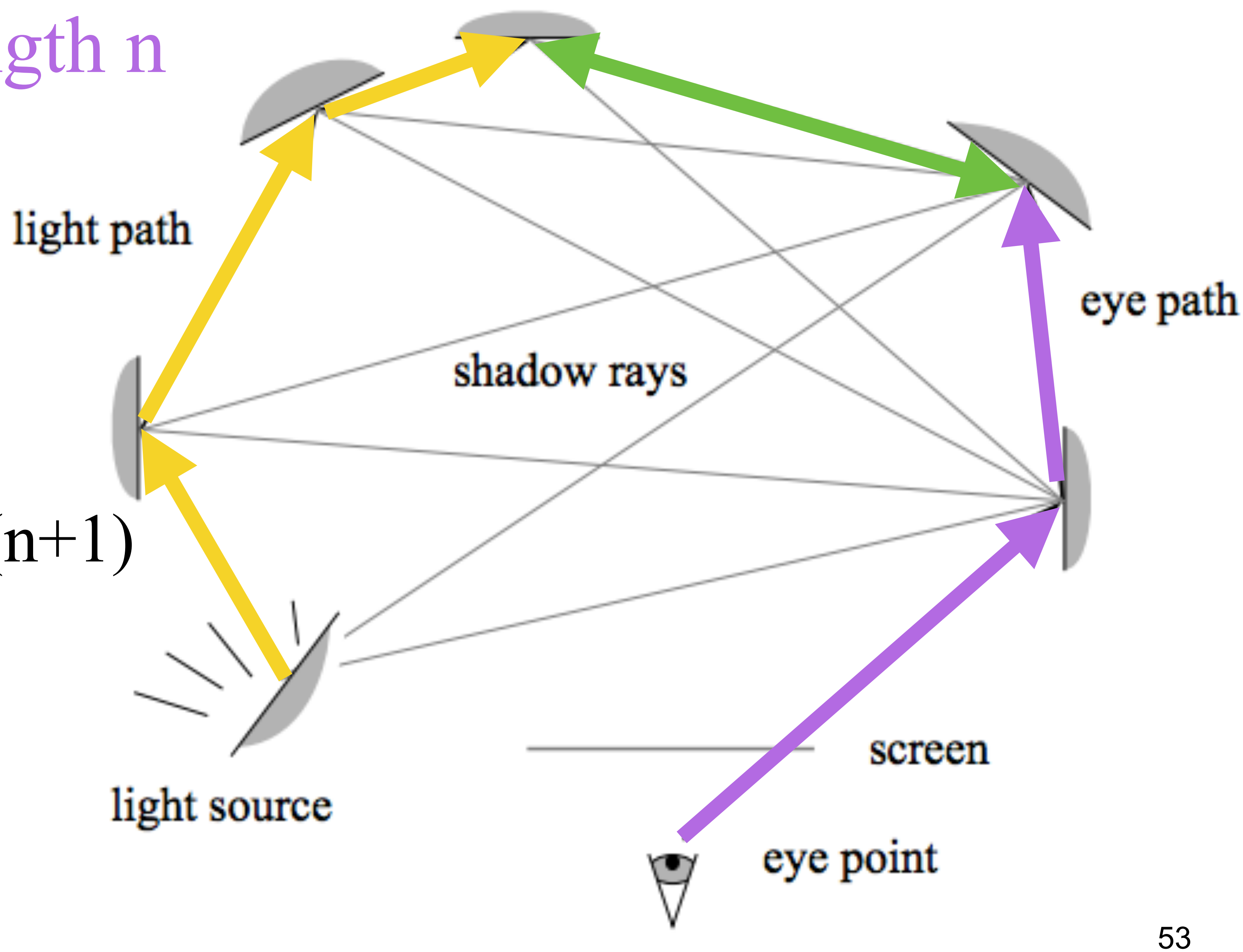
- Shoot a path of length m from light
- Shoot a path of length n from the eye
- Connect the two in all the possible ways
  - This forms (m+1)*(n+1) separate paths



BDPT Recipe, Outer Loop

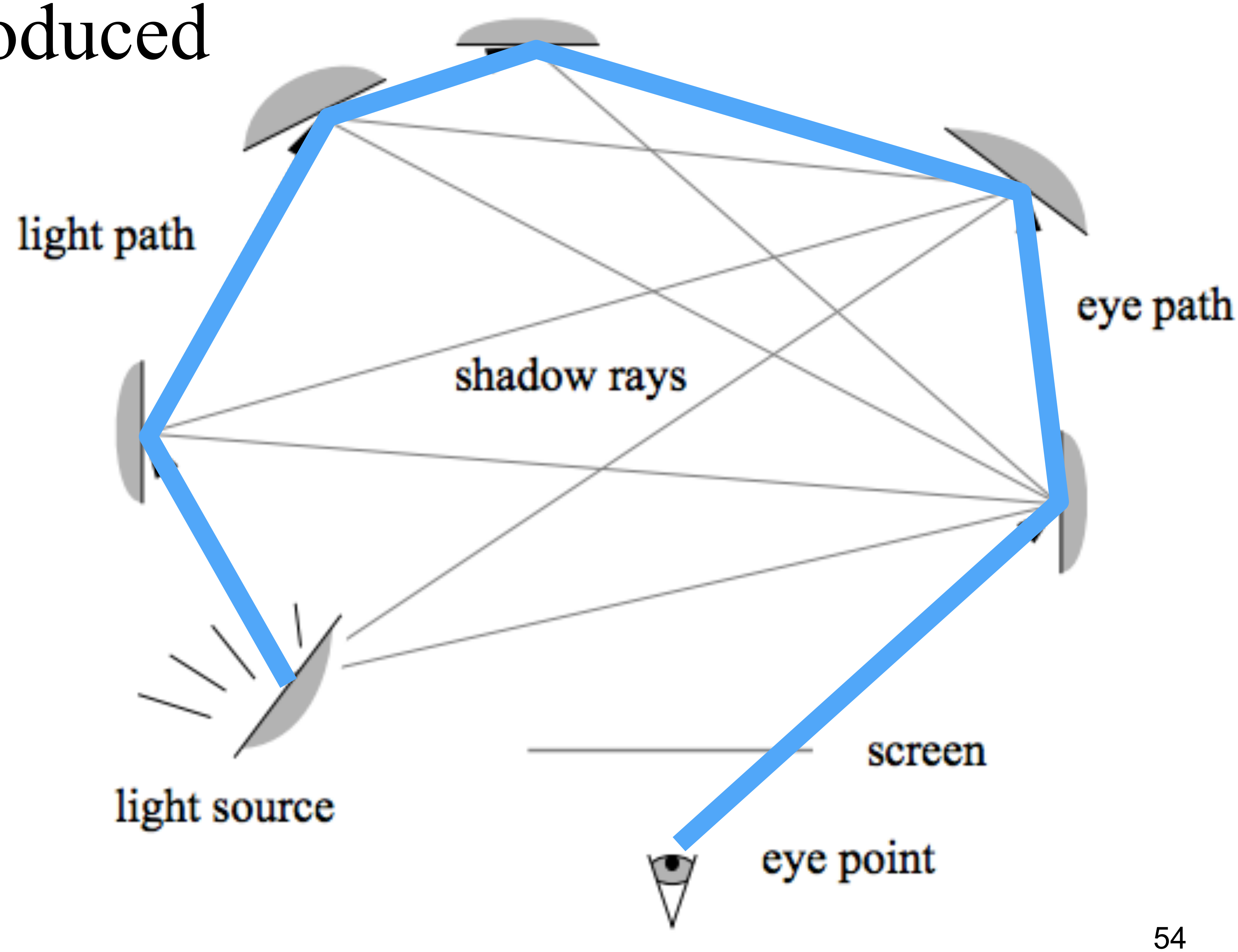
k=6

- Shoot a path of length m from light
- Shoot a path of length n from the eye
- Connect the two in all the possible ways
  - This forms (m+1)*(n+1) separate paths



BDPT Recipe, Inner Loop

- For each path x thus formed, iterate over all m+1 possible k, m that *could have* produced the path
  - Compute $p_{k,m}(x)$



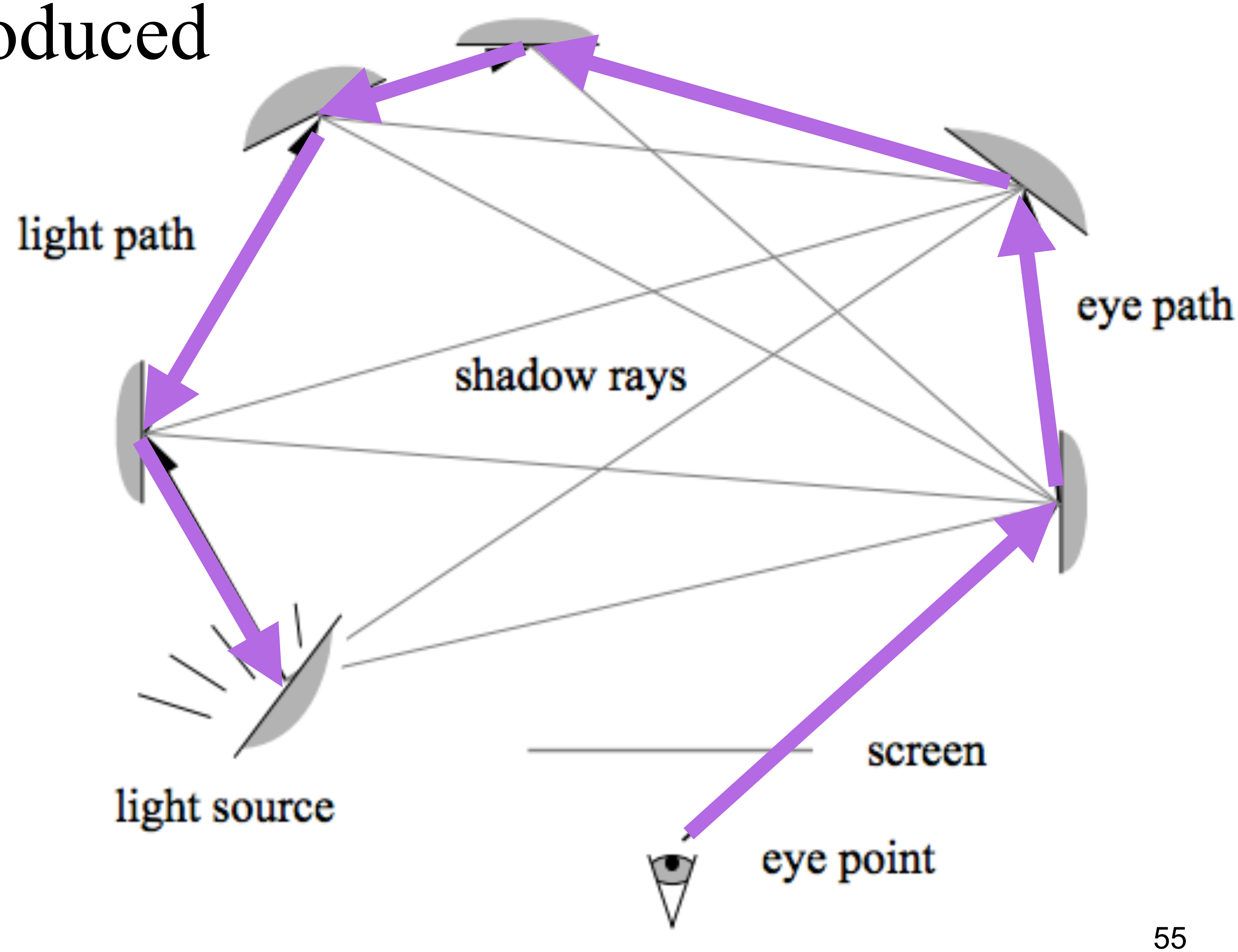
BDPT Recipe, Inner Loop

eye sampler

light sampler

deterministic

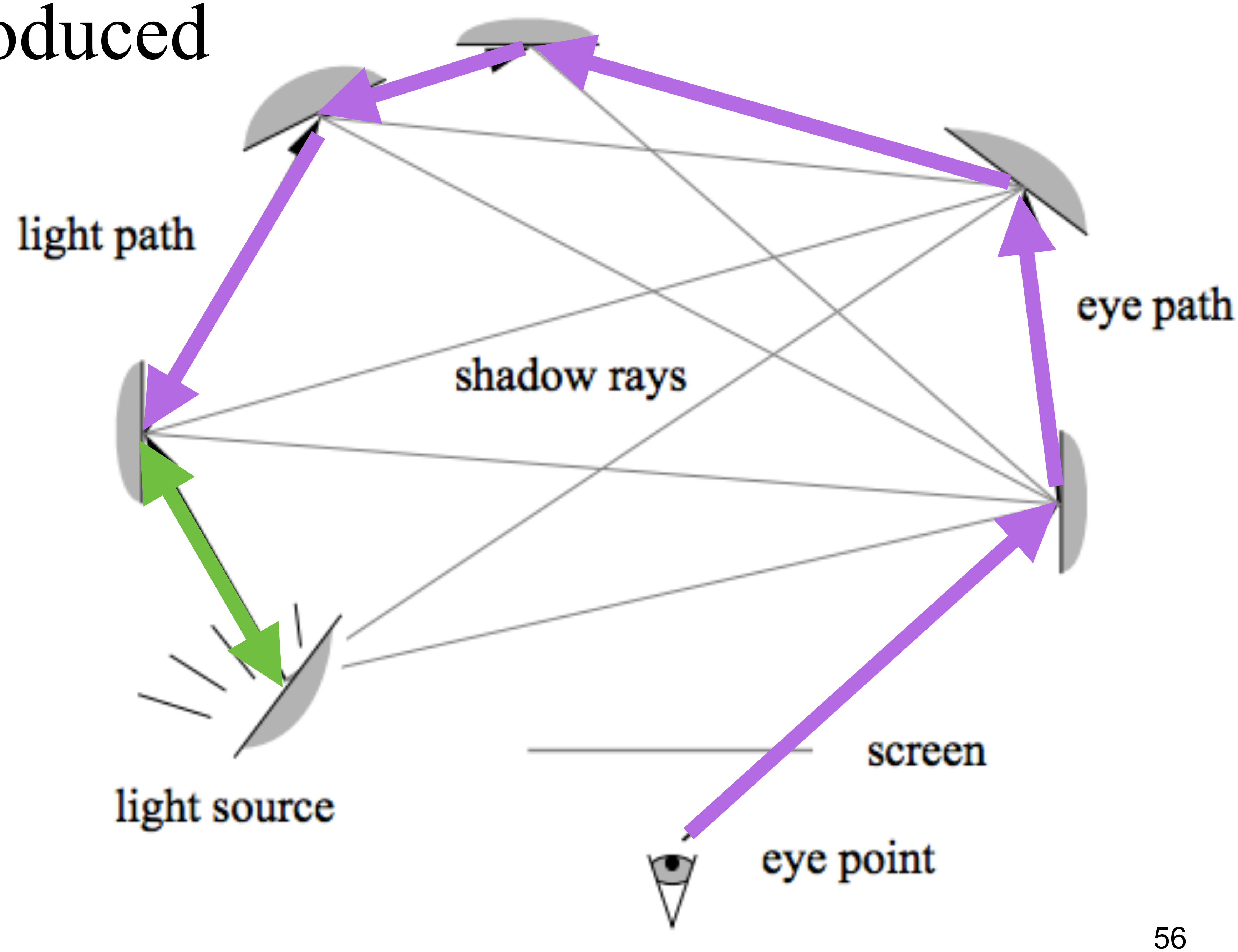
- For each path x thus formed, iterate over all m+1 possible k, m that *could have* produced the path
  - Compute $p_{k,m}(x)$



BDPT Recipe, Inner Loop

eye sampler
light sampler
deterministic

- For each path x thus formed, iterate over all m+1 possible k, m that *could have* produced the path
  - Compute $p_{k,m}(x)$



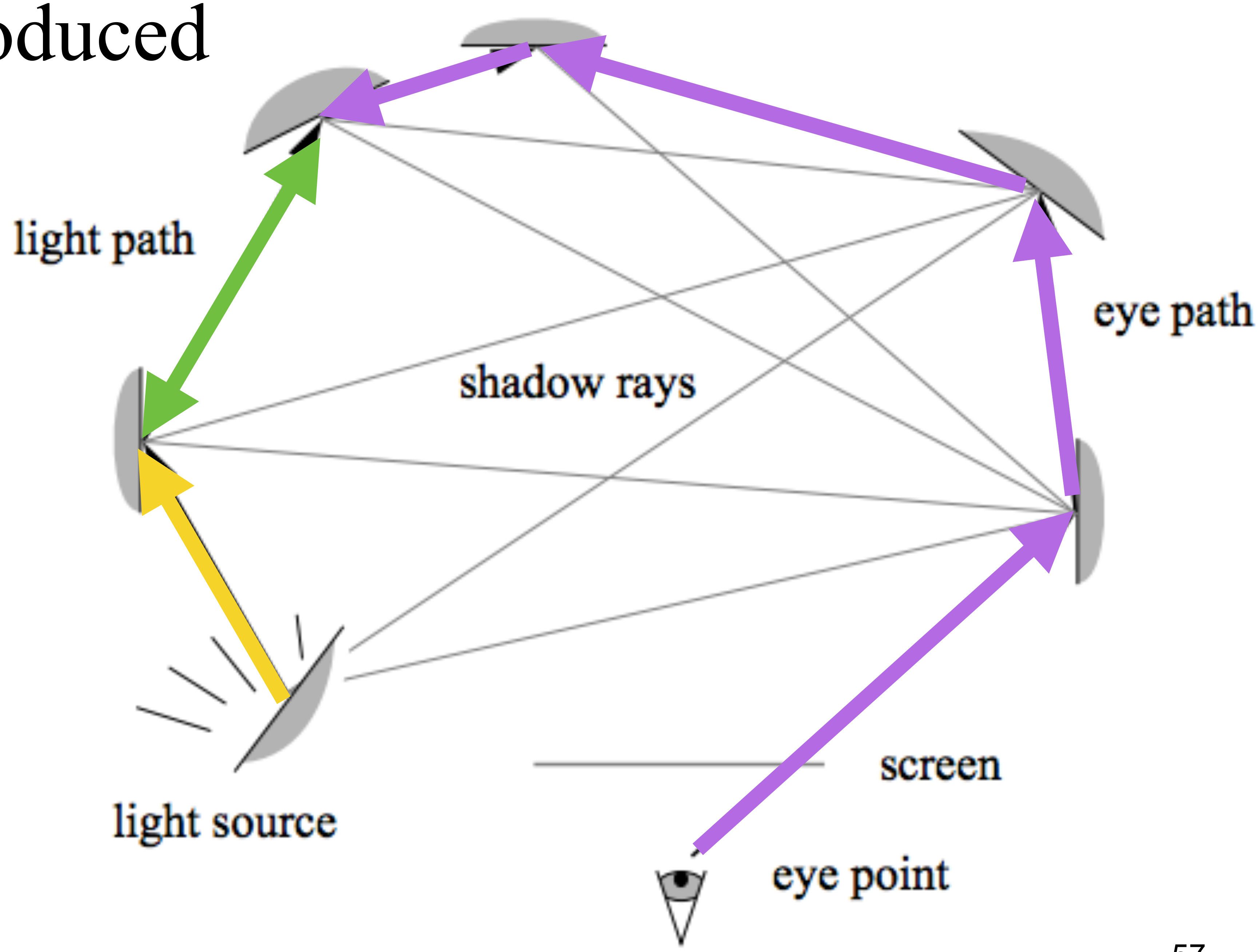
BDPT Recipe, Inner Loop

eye sampler

light sampler

deterministic

- For each path x thus formed, iterate over all m+1 possible k, m that *could have* produced the path
  - Compute $p_{k,m}(x)$



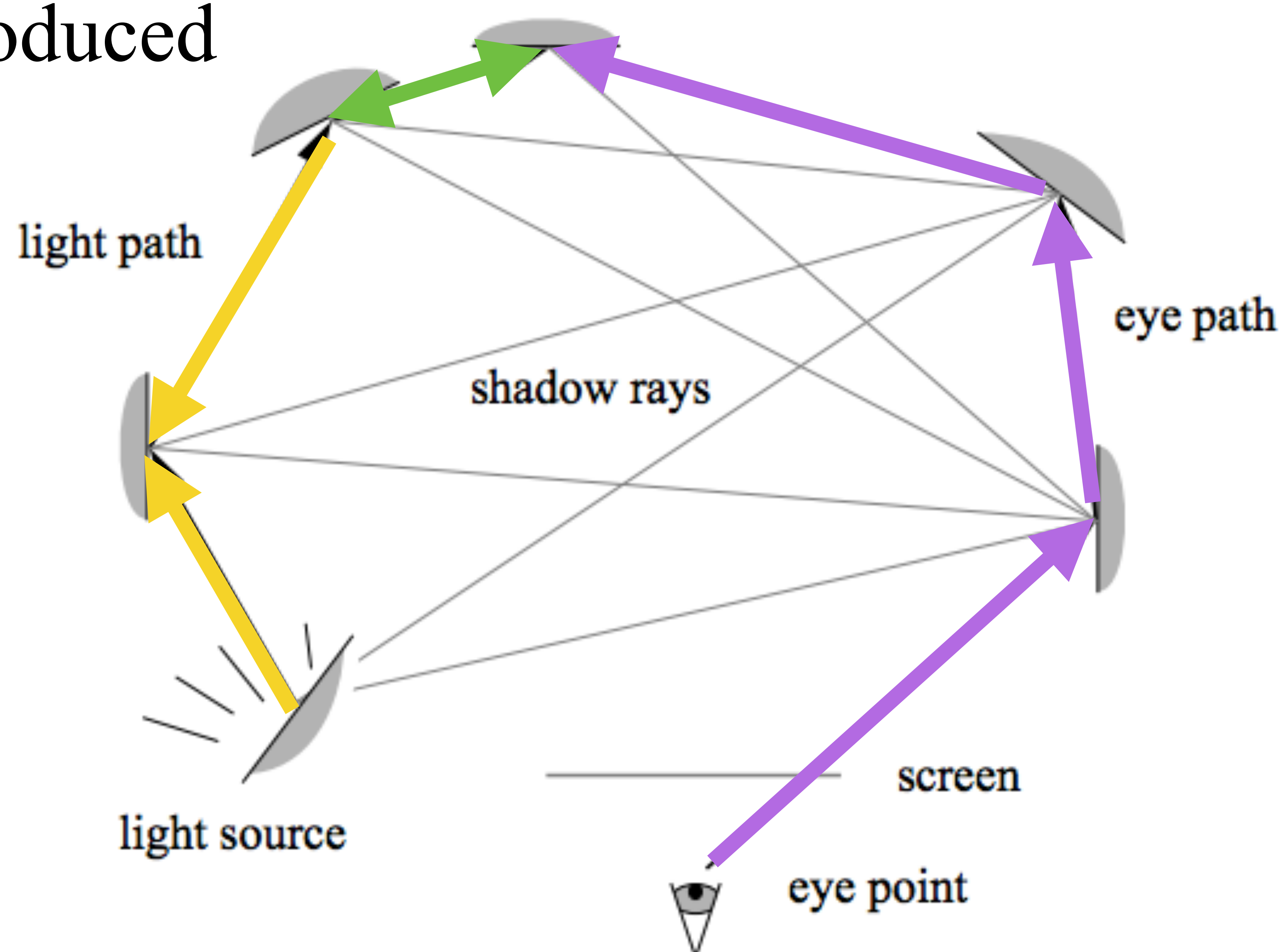
BDPT Recipe, Inner Loop

eye sampler

light sampler

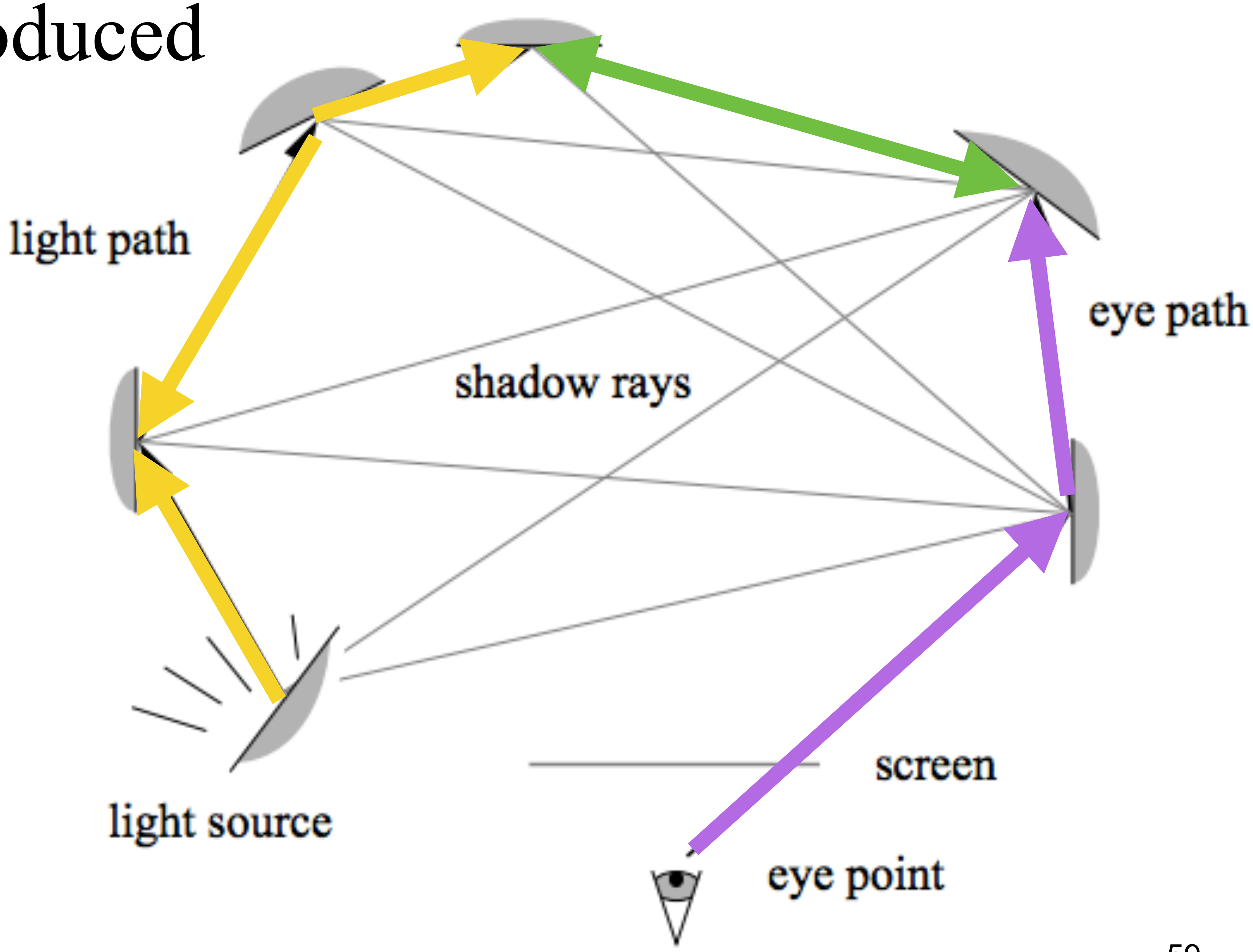
deterministic

- For each path x thus formed, iterate over all m+1 possible k, m that *could have* produced the path
  - Compute $p_{k,m}(x)$



BDPT Recipe, Inner Loop

- For each path x thus formed, iterate over all m+1 possible k, m that *could have* produced the path
  - Compute $p_{k,m}(x)$



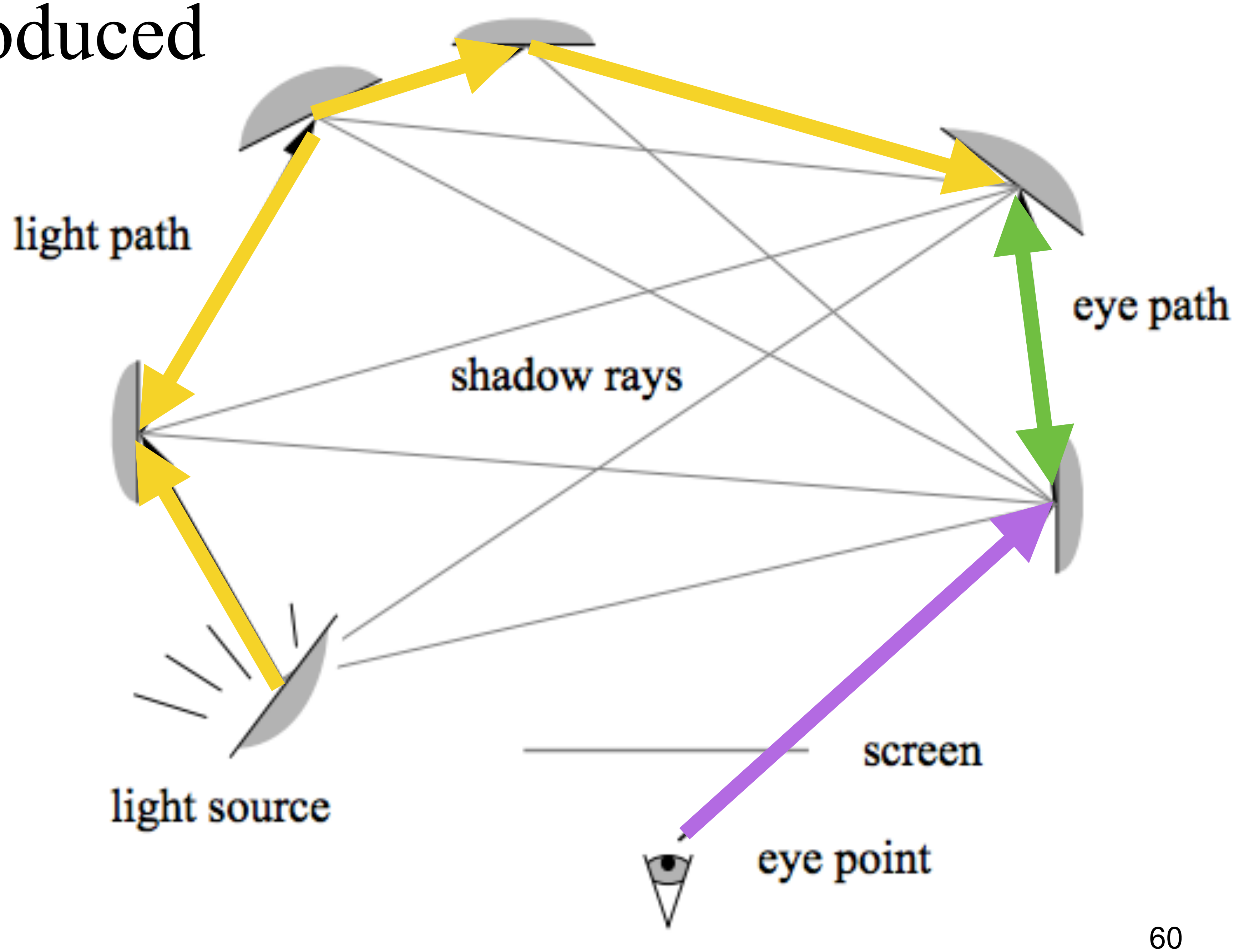
BDPT Recipe, Inner Loop

eye sampler

light sampler

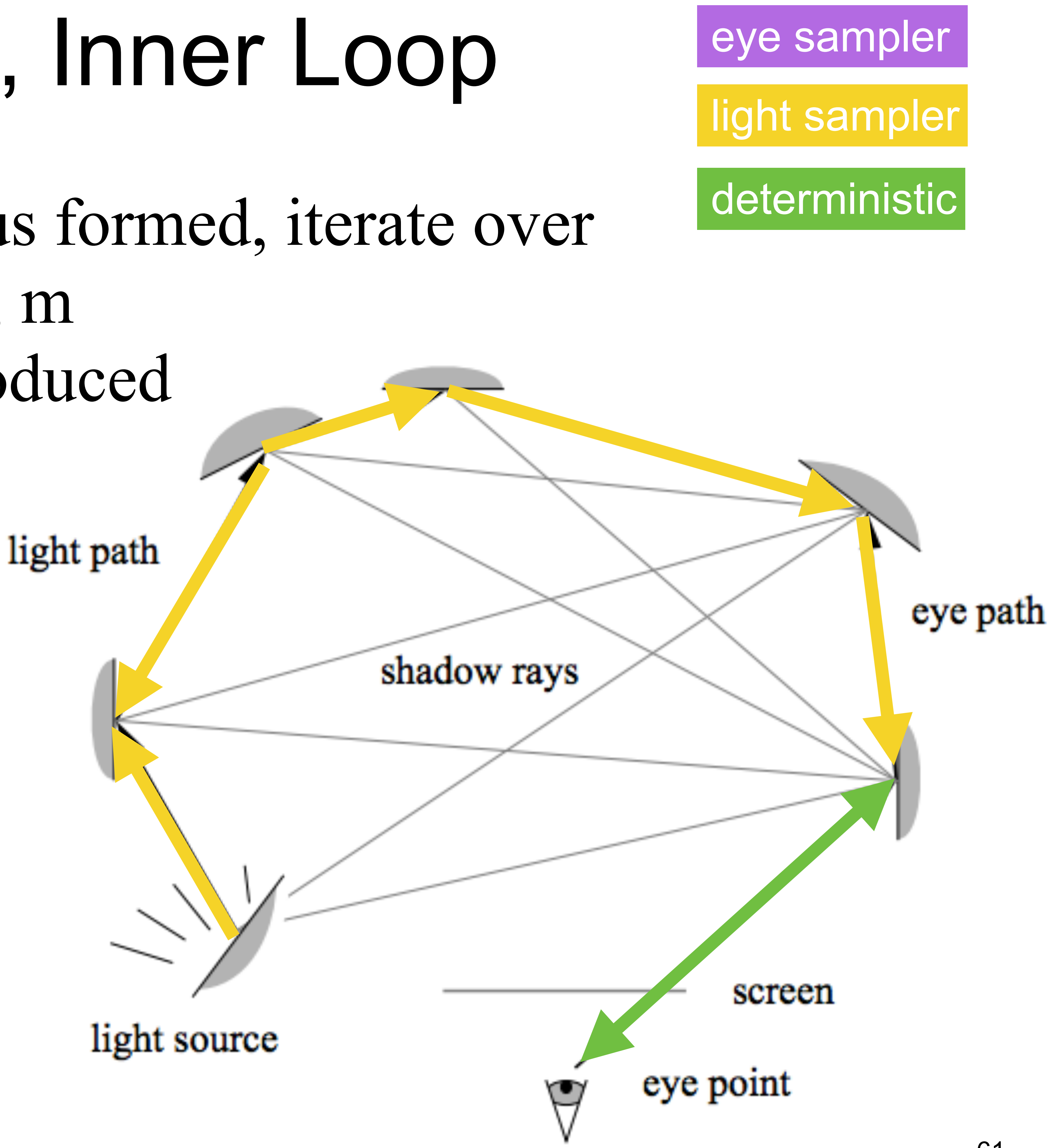
deterministic

- For each path x thus formed, iterate over all m+1 possible k, m that *could have* produced the path
  - Compute $p_{k,m}(x)$



BDPT Recipe, Inner Loop

- For each path x thus formed, iterate over all m+1 possible k, m that *could have* produced the path
  - Compute $p_{k,m}(x)$



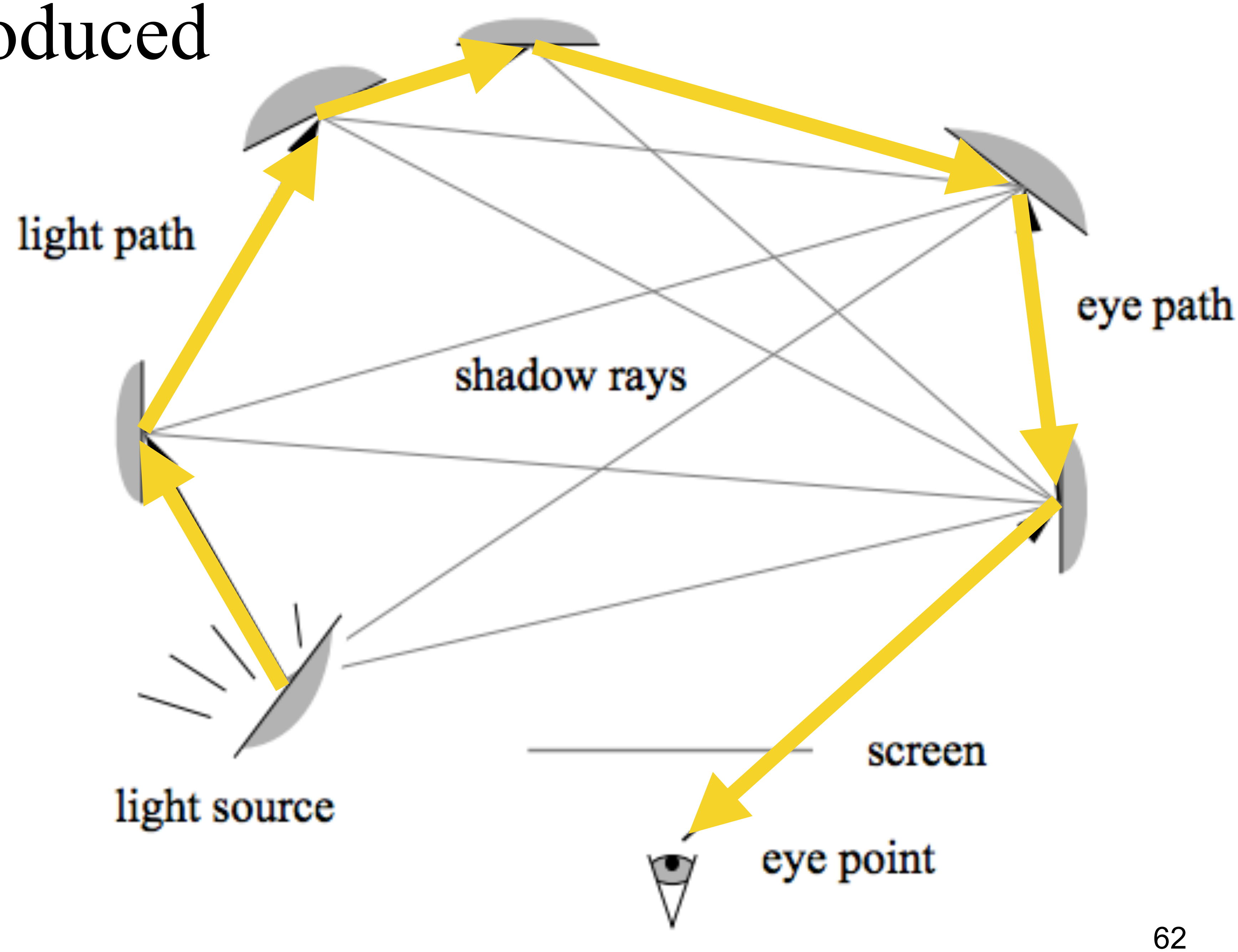
BDPT Recipe, Inner Loop

eye sampler

light sampler

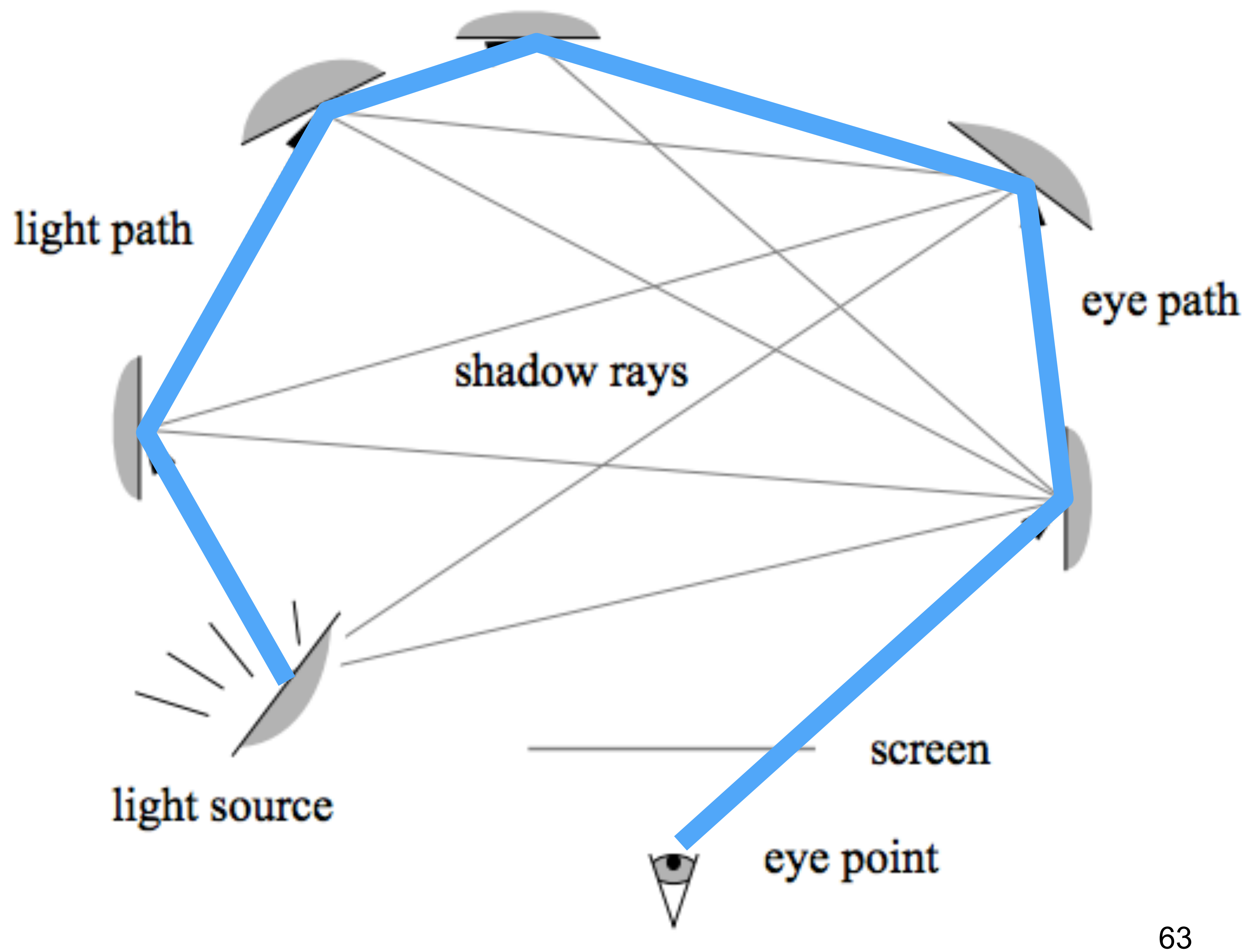
deterministic

- For each path x thus formed, iterate over all m+1 possible k, m that *could have* produced the path
  - Compute $p_{k,m}(x)$



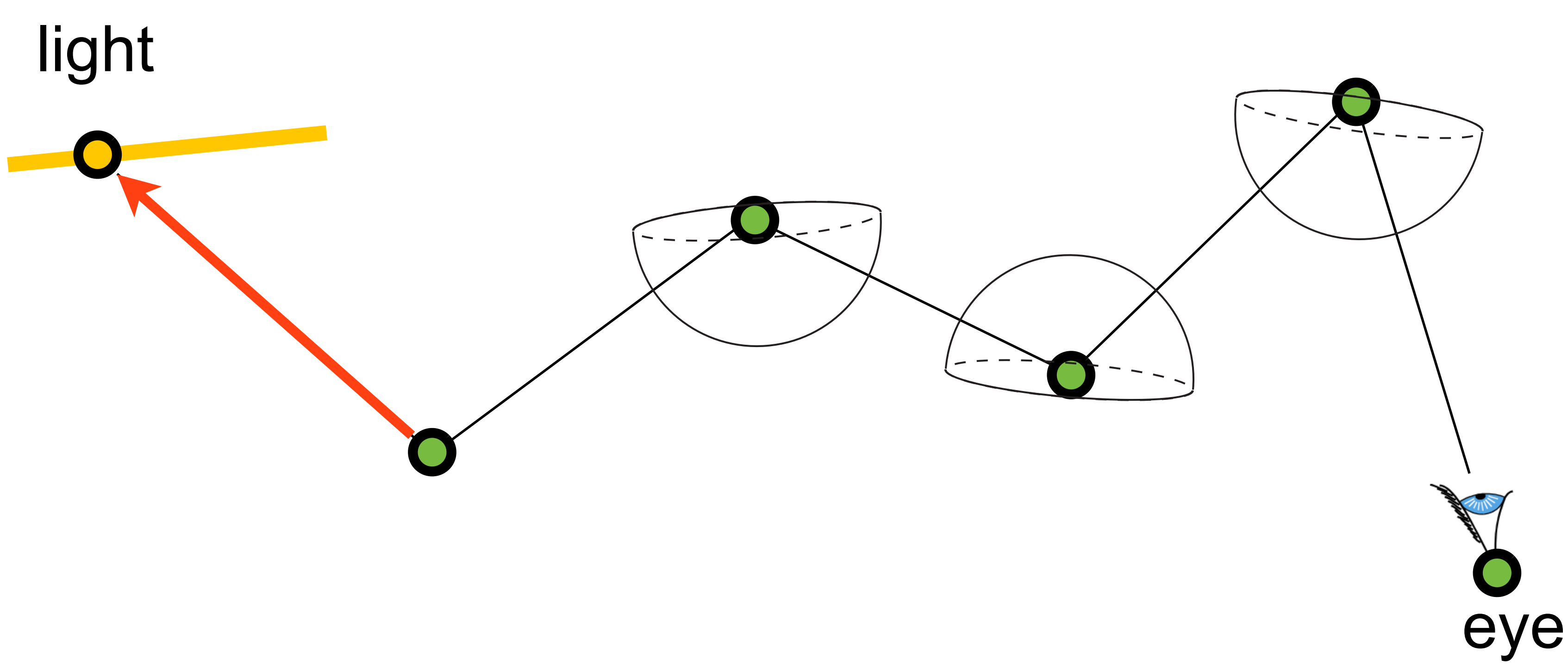
BDPT Recipe, Inner Loop

- Now, can compute $\bar{p}(x)$ according to the MIS recipe from all the $p_{k,m}(x)$
- Done?



One More Thing

- The different samplers produce paths that "live" in different spaces
  - E.g. regular path tracer's paths of length k live on $\Omega^{k-1} \times L$ where omega is the (hemi)sphere and L is surface of the light
  - Paths sampled in other ways "live" in a different space!



One More Thing

- To be able to sum PDFs in MIS, must convert all paths to a common space/parameterization/measure
- We pick a measure that converts all solid angles to corresponding areas
  - This is called the *area-product measure* (Veach)
  - Also, pretty synonymously, we call it *path space*
  - Simple: apply the change of variables from solid angle to area (we already know how)

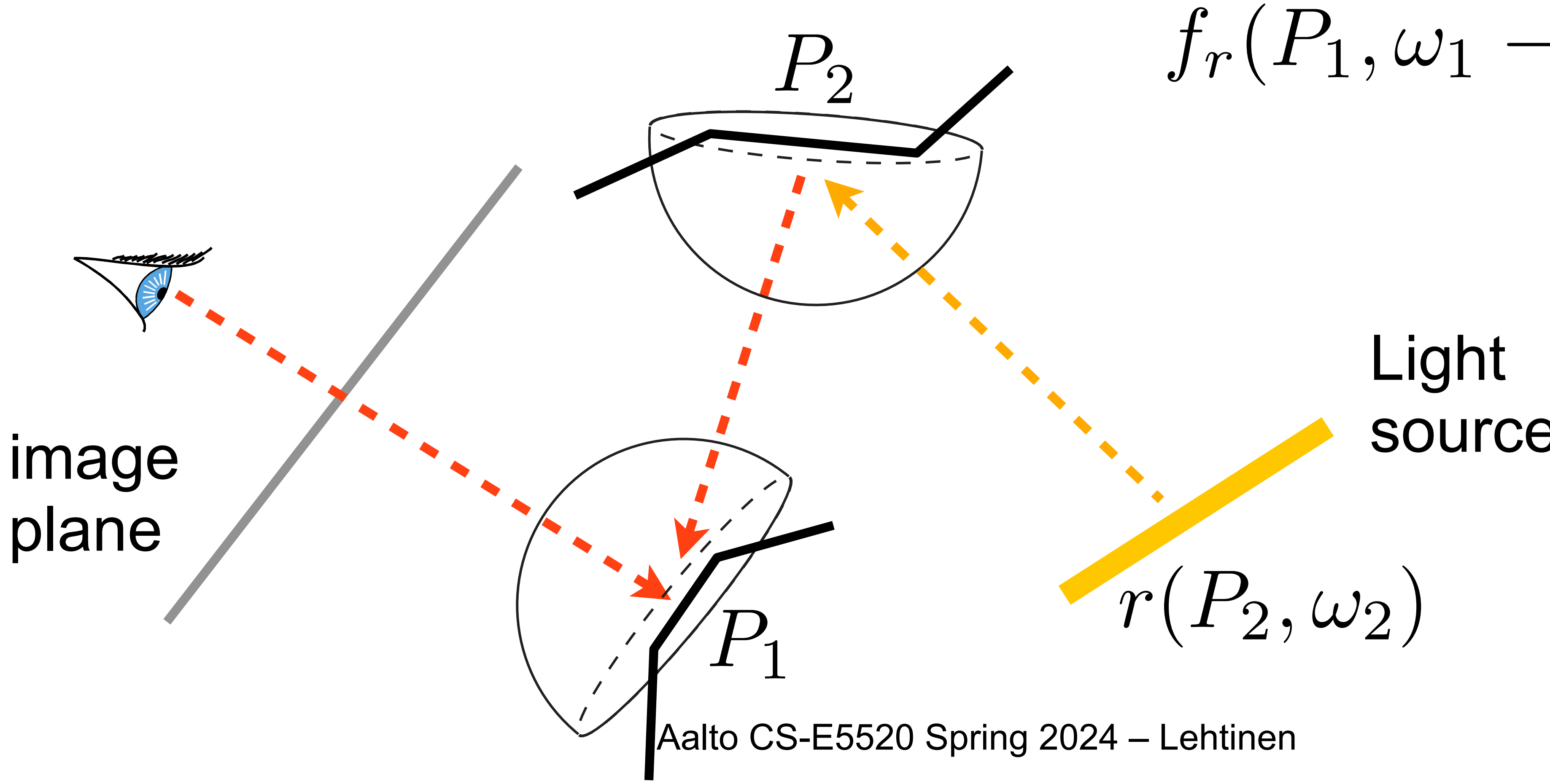
Veach's "Path Space"

- Earlier we wrote n-bounce lighting as a simultaneous integral over n hemispheres

Flashback: 1 Indirect Bounce

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

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



Veach's "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

Rendering Equation Solid Angle form

$$L_o(x \to \omega_o) = E(x \to \omega_o) + \int_\Omega L(x \leftarrow \omega_i)\, f_r(x, \omega_i \to \omega_o)\, \cos\theta \, \mathrm{d}\omega_i$$

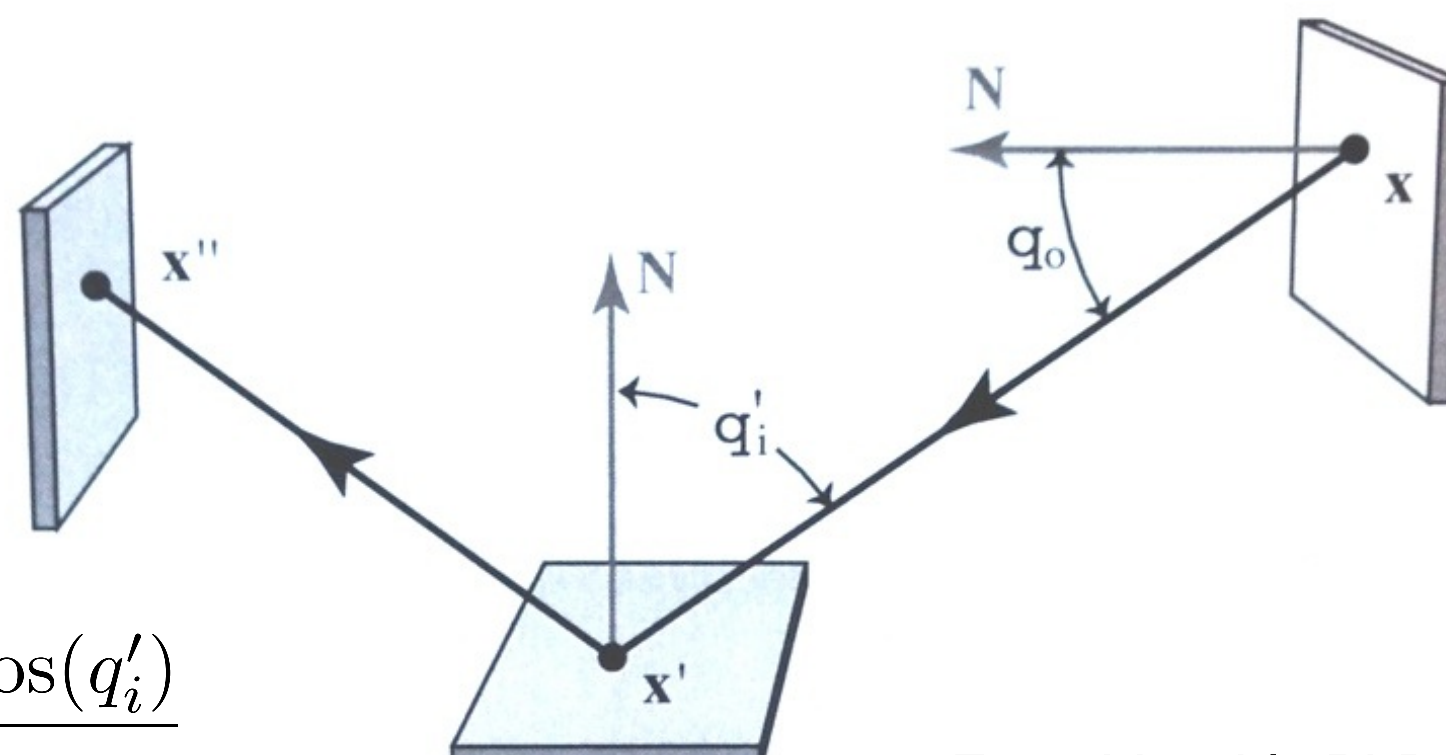
- Now let's apply the change of variables from the 2nd lecture to convert solid angle to area..

$$\mathrm{d}\omega = \frac{\cos\theta}{r^2}\mathrm{d}A$$

Rendering Equation, Area Form

- Radiance from *x* reflected by *x*' towards *x*''

$$L_o(x' \to x'') = E(x' \to x'') + \int_S L(x \to x')\, f_r(x \to x' \to x'')\, G(x \leftrightarrow x')\, \mathrm{d}A(x)$$



$$G(x \leftrightarrow x') = \frac{V(x \leftrightarrow x')\cos(q_o)\cos(q_i')}{\|x - x'\|^2}$$

From Veach's PhD

The G Term

- Absorbs the familiar cosine/r^2 solid-angle-to-area change of variables, and the incident cosine.

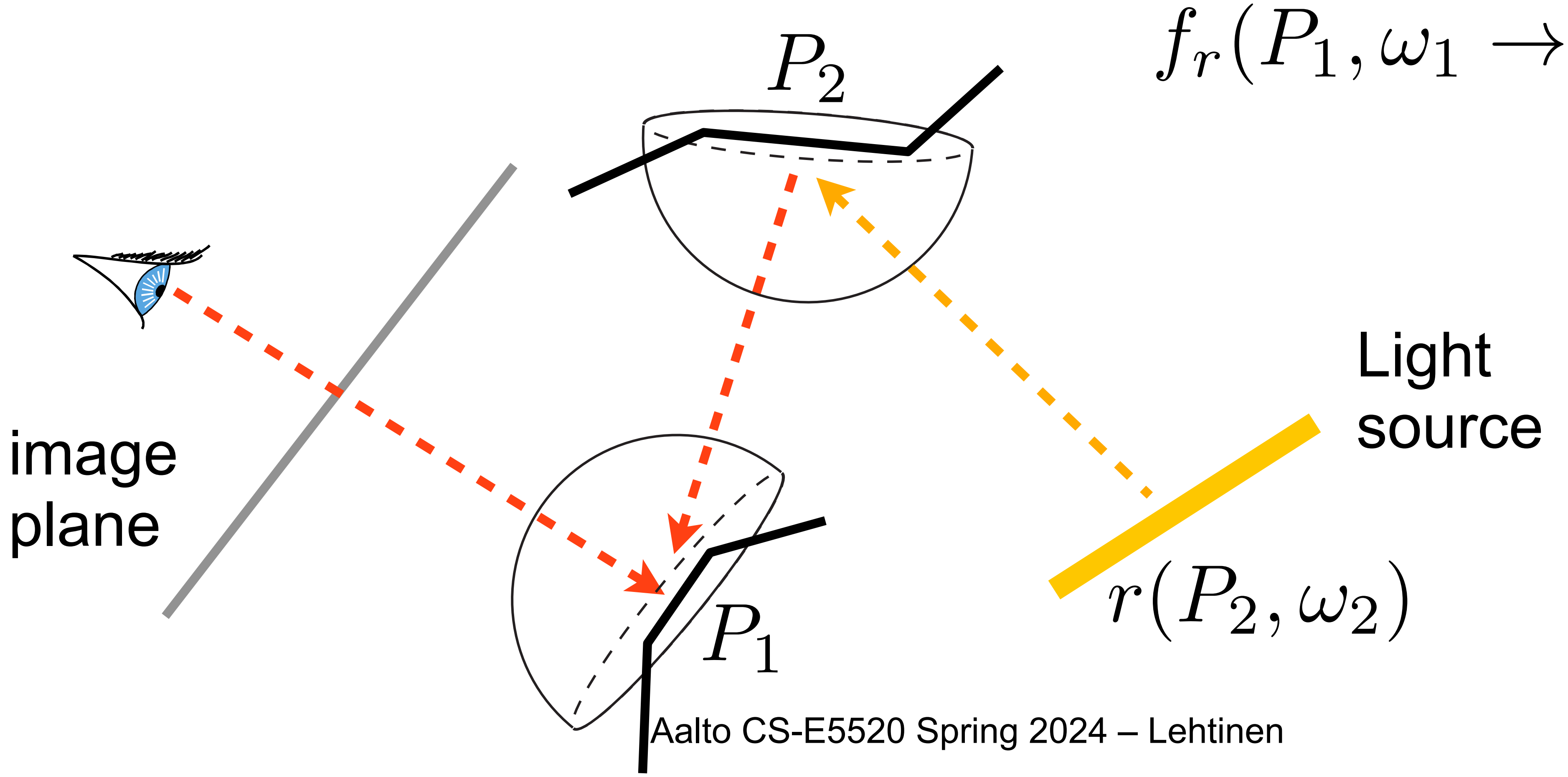
$$G(x \leftrightarrow x') = \frac{V(x \leftrightarrow x') \cos(q_o) \cos(q_i')}{\|x - x'\|^2}$$

- (Standard clamping / absolute values apply to cosines)

1 Indirect Bounce, Hemispheres

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

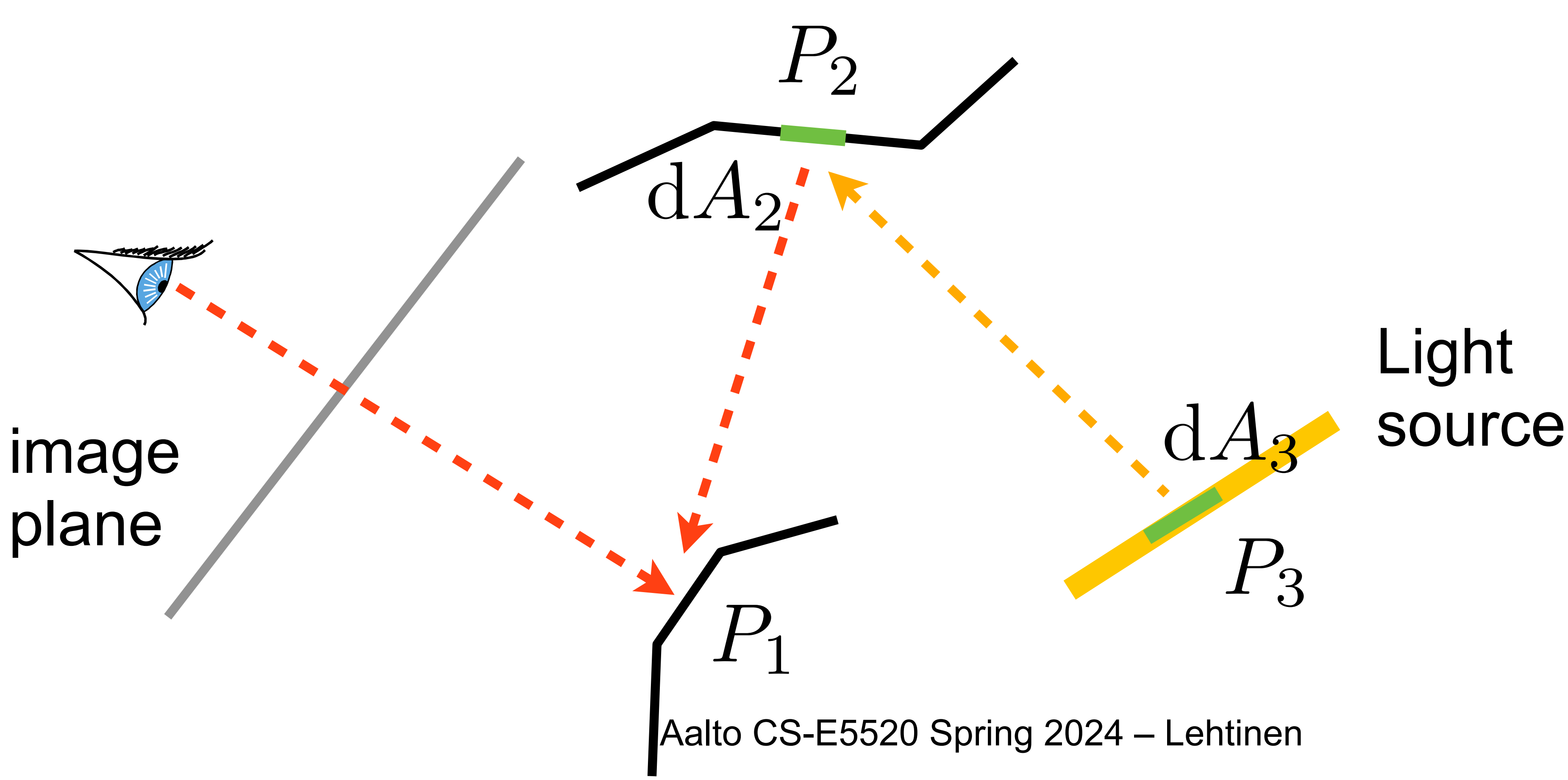
$$L_2(x,y) = \int_{\Omega(P_1)} \overbrace{\left[\int_{\Omega(P_2)} E(r(P_2,\omega_2) \to P_2) f_r(P_2, \omega_2 \to -\omega_1) \cos\theta_2 \mathrm{d}\omega_2 \right]}^{L(P_1 \leftarrow \omega_1)} f_r(P_1, \omega_1 \to \text{eye}) \cos\theta_1 \mathrm{d}\omega_1$$



1 Indirect Bounce, Area-Product

$$L_2(x,y) = \int_S \int_S E(P_3 \to P_2) G(P_3 \leftrightarrow P_2) f_r(P_3 \to P_2 \to P_1) \times$$

$$G(P_2 \leftrightarrow P_1) f_r(P_2 \to P_1 \to \text{camera}) \mathrm{d}A(P_3) \mathrm{d}A(P_2)$$



Must Convert all Paths to This Form

- Only then able to apply MIS
- Won't go into 100% gritty detail
- See the MIS paper and Veach's PhD thesis (Chap. 8, 10), again, for all the details
- One important detail: the local sampling densities are defined in terms of solid angles (e.g. cosine-weighted sampling)
  - Must also convert the PDFs, not just path throughput
  - Fortunately, very similar to the change of variables:

$$p(\mathbf{x}) = p(\omega)\frac{\mathrm{d}\omega}{\mathrm{d}A} = p(\omega)\frac{\cos\theta}{r^2}$$

One Last Detail

- Veach treats the sensor as being part of the scene, i.e., he writes the pixel filter etc. using the same path space formalism
  - The "pixel" is just another path vertex
  - This *is* a good idea if you consider physical camera models
  - If you attempt BDPT, I suggest you handle the camera/sensor as a special case and do not attempt the full generality
  - However, when evaluating the PDFs, you will have to account for the screen-to-visible surface change of variables
    - Not super hard.

Getting the details right is *hard*…

- Not in principle, but in practice.

- Therefore…

Aether

- Anderson, Li, Lehtinen, Durand, SIGGRAPH 2017
- Aether is a domain-specific language that takes care of both discrete and continuous probabilities in light transport simulation

- You write the sampling code that generates the paths, and Aether automatically provides you with the associated *pdf*() function
  - Can be used on other paths as well, so supports MIS directly
- Built around reusable *strategies*
  - sample point on lens, sample BRDF, sample light source…

Example

Aether Makes Extending Algorithms Easier

```
// Create camera subpath
RandomSequence<Vertex> camPath;
// Append the camera position
camPath.Append(sampCamPos, uniDist);
// Append the primary intersection point for pixel (x, y)
camPath.Append(sampCamDir, uniDist, raycaster, x, y);
// Extend by successive BSDF sampling until max depth
for(; camPath.Size() <= maxDepth;) {
  camPath.Append(sampBSDF, uniDist, raycaster);
}
camPath.Sample();

Spectrum Li(0);
for (int length = 2; length < camPath.Size(); length++) {
  // BSDF sampled path
  auto bsdfPath = camPath.Slice(0, length + 1);
  // BSDF sampled path + direct light sampling
  auto directPath = camPath.Slice(0, length);
  // Direct sampling the light
  directPath.Append(sampEmtDirect, uniDist, emitters);
  directPath.Sample();
  // Combine bsdf path and direct path
  // Returns a list of paths with their MIS weights
  auto combinedList =
    combine<PowerHeuristic>(bsdfPath, directPath);
  // Sum up the contributions
  for (const auto &combined : combinedList) {
    const auto &path = combined.sequence;
    Li += combined.weight *
      (integrand(path) / path.Pdf());
  }
}
return Li;
```

Path Tracer

```
RandomSequence<Vertex> camPath;
// ... sample camera subpath as in the path tracer

// Create emitter subpath
RandomSequence<Vertex> emtPath;
// Randomly sample a light and a position on the light
emtPath.Append(sampEmtPos, uniDist, emitters);
// Sample direction from emitter and intersect with scene
emtPath.Append(sampEmtDir, uniDist, raycaster);
for(; emtPath.Size() <= maxDepth;) {
  emtPath.Append(sampBSDF, uniDist, raycaster);
}
emtPath.Sample();

// Combine subpaths
for (int length = 2; length <= maxDepth + 1; length++) {
  // Collect paths with specified length
  std::vector<RandomSequence<Vertex>> paths;
  for (int camSize = 0; camSize < length; camSize++) {
    const int emtSize = length - camSize;
    // Slice the subpaths and connect them together
    auto camSlice = camPath.Slice(0, camSize);
    auto emtSlice = emtPath.Slice(0, emtSize);
    paths.push_back(camSlice.Concat(reverse(emtSlice)));
  }
  // Combine bsdf path and direct path
  // Returns a list of paths with their MIS weights
  auto combinedList = combine<PowerHeuristic>(paths);
  for (const auto &combined : combinedList) {
    const auto &path = combined.sequence;
    // Compute w*f/p and splats contribution
    film->Record(project(path),
      combined.weight * (integrand(path) / path.Pdf()));
  }
}
```

Bidirectional

```
// segment contains two vertices of the 'portal edge'
// We assume segment never hits the sensor, but it
// could hit the emitter
RandomSequence<Vertex> segment;
// ...
std::vector<RandomSequence<Vertex>> paths;
for (int camSize = 1; camSize < length; camSize++) {
  const int emtSize = length - camSize;
  // Tri-directional subpath
  if (camSize > 1 && emtSize >= 1) {
    // Shorten the sensor and emitter subpaths by 1
    auto camSlc = camPath.Slice(0, camSize - 1);
    auto emtSlc = emtPath.Slice(0, emtSize - 1);
    // Replace with segment
    paths.push_back(
      camSlc.Concat(segment).Concat(reverse(emtSlc)));
  }
  // Shorten the sensor subpaths by 2
  if (sensorSubpathSize > 2) {
    auto camSlc = camPath.Slice(0, camSize - 2);
    auto emtSlc = emtPath.Slice(0, emtSize);
    paths.push_back(
      camSlc.Concat(segment).Concat(reverse(emtSlc)));
  }
  // Shorten the emitter subpaths by 2
  if (emitterSubpathSize >= 2) {
    auto camSlc = camPath.Slice(0, camSize);
    auto emtSlc = emtPath.Slice(0, emtSize - 2);
    paths.push_back(
      camSlc.Concat(segment).Concat(reverse(emtSlc)));
  }
  // Slice and concat without the segment
  auto camSlc = camPath.Slice(0, camSize);
  auto emtSlc = emtPath.Slice(0, emtSize);
  paths.push_back(camSlc.Concat(reverse(emtSlc)));
}
// ...combine the paths as in BDPT
auto combinedList = combine<PowerHeuristic>(paths);
// ...
```

Tridirectional

Example

Aether Makes Extending Algorithms Easier

```cpp
// Create camera subpath
RandomSequence<Vertex> camPath;
// Append the camera position
camPath.Append(sampCamPos, uniDist);
// Append the primary intersection point for pixel (x, y)
camPath.Append(sampCamDir, uniDist, raycaster, x, y);
// Extend by successive BSDF sampling until max depth
for(; camPath.Size() <= maxDepth;) {
  camPath.Append(sampBSDF, uniDist, raycaster);
}
camPath.Sample();

Spectrum Li(0);
for (int length = 2; length < camPath.Size(); length++) {
  // BSDF sampled path
  auto bsdfPath = camPath.Slice(0, length + 1);
  // BSDF sampled path + direct light sampling
  auto directPath = camPath.Slice(0, length);
  // Direct sampling the light
  directPath.Append(sampEmtDirect, uniDist, emitters);
  directPath.Sample();
  // Combine bsdf path and direct path
  // Returns a list of paths with their MIS weights
  auto combinedList =
    combine<PowerHeuristic>(bsdfPath, directPath);
  // Sum up the contributions
  for (const auto &combined : combinedList) {
    const auto &path = combined.sequence;
    Li += combined.weight *
      (integrand(path) / path.Pdf());
  }
}
return Li;
```

Path Tracer

Code on github!
(See project page)

```cpp
RandomSequence<Vertex> camPath;
// ... sample camera subpath as in the path tracer

// Create emitter subpath
RandomSequence<Vertex> emtPath;
// Randomly sample a light and a position on the light
emtPath.Append(sampEmtPos, uniDist, emitters);
// Sample direction from emitter and intersect with scene
emtPath.Append(sampEmtDir, uniDist, raycaster);
for(; emtPath.Size() <= maxDepth;) {
  emtPath.Append(sampBSDF, uniDist, raycaster);
}
emtPath.Sample();

// Combine subpaths
for (int length = 2; length <= maxDepth + 1; length++) {
  // Collect paths with specified length
  std::vector<RandomSequence<Vertex>> paths;
  for (int camSize = 0; camSize <= length; camSize++) {
    const int emtSize = length - camSize;
    // Slice the subpaths and connect them together
    auto camSlice = camPath.Slice(0, camSize);
    auto emtSlice = emtPath.Slice(0, emtSize);
    paths.push_back(camSlice.Concat(reverse(emtSlice)));
  }
  // Combine bsdf path and direct path
  // Returns a list of paths with their MIS weights
  auto combinedList = combine<PowerHeuristic>(paths);
  for (const auto &combined : combinedList) {
    const auto &path = combined.sequence;
    // Compute w*f/p and splats contribution
    film->Record(project(path),
      combined.weight * (integrand(path) / path.Pdf()));
  }
}
```

Bidirectional

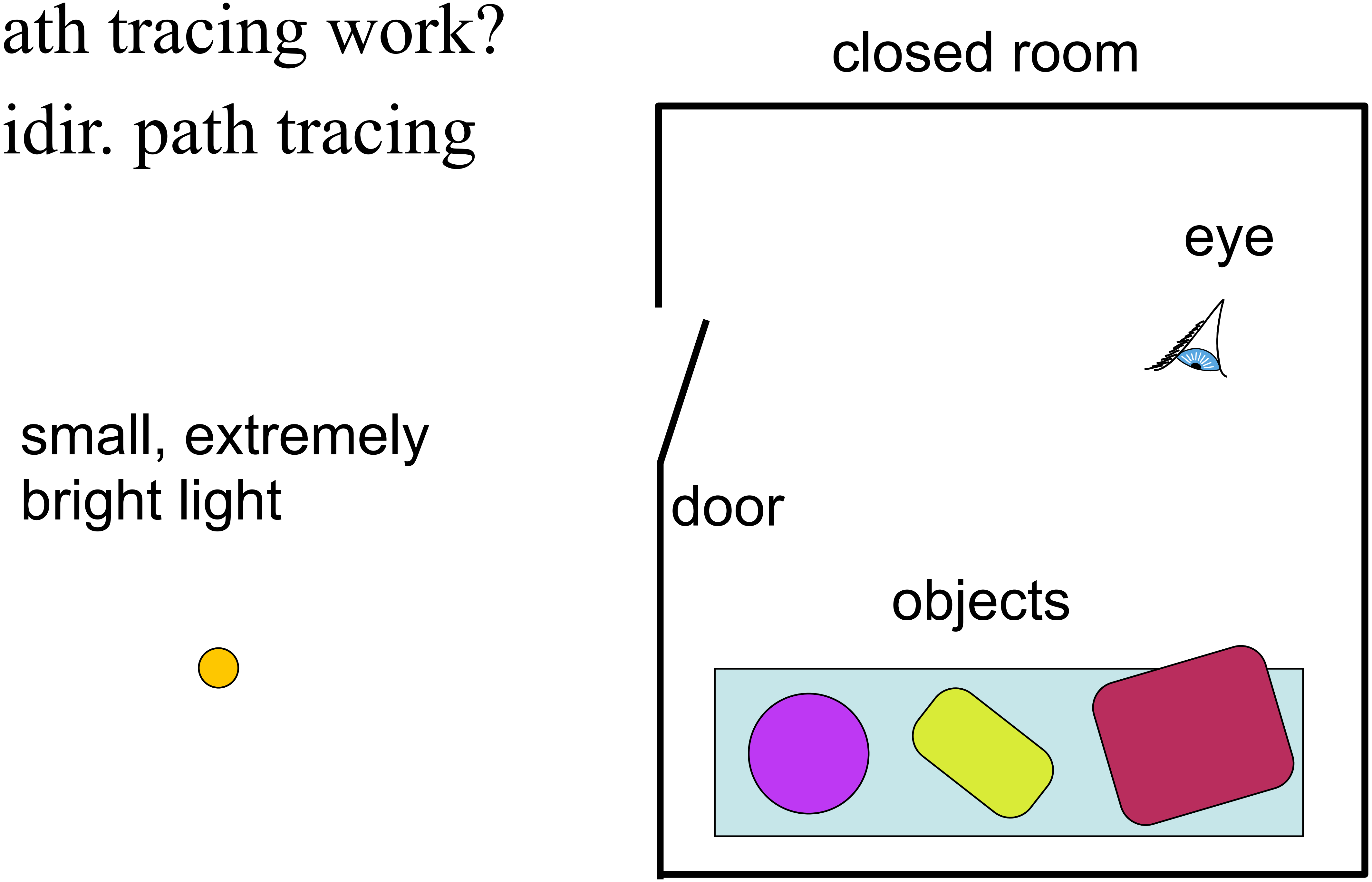
```cpp
// segment contains two vertices of the 'portal edge'
// We assume segment never hits the sensor, but it
// could hit the emitter
RandomSequence<Vertex> segment;
// ...
std::vector<RandomSequence<Vertex>> paths;
for (int camSize = 1; camSize < length; camSize++) {
  const int emtSize = length - camSize;
  // Tri-directional subpath
  if (camSize > 1 && emtSize >= 1) {
    // Shorten the sensor and emitter subpaths by 1
    auto camSlc = camPath.Slice(0, camSize - 1);
    auto emtSlc = emtPath.Slice(0, emtSize - 1);
    // Replace with segment
    paths.push_back(
      camSlc.Concat(segment).Concat(reverse(emtSlc)));
  }
  // Shorten the sensor subpaths by 2
  if (sensorSubpathSize > 2) {
    auto camSlc = camPath.Slice(0, camSize - 2);
    auto emtSlc = emtPath.Slice(0, emtSize);
    paths.push_back(
      camSlc.Concat(segment).Concat(reverse(emtSlc)));
  }
  // Shorten the emitter subpaths by 2
  if (emitterSubpathSize >= 2) {
    auto camSlc = camPath.Slice(0, camSize);
    auto emtSlc = emtPath.Slice(0, emtSize - 2);
    paths.push_back(
      camSlc.Concat(segment).Concat(reverse(emtSlc)));
  }
  // Slice and concat without the segment
  auto camSlc = camPath.Slice(0, camSize);
  auto emtSlc = emtPath.Slice(0, emtSize);
  paths.push_back(camSlc.Concat(reverse(emtSlc)));
}
// ...combine the paths as in BDPT
auto combinedList = combine<PowerHeuristic>(paths);
// ...
```

Tridirectional

Miguel Angel Bermudez Pinon, rendered using Maxwell

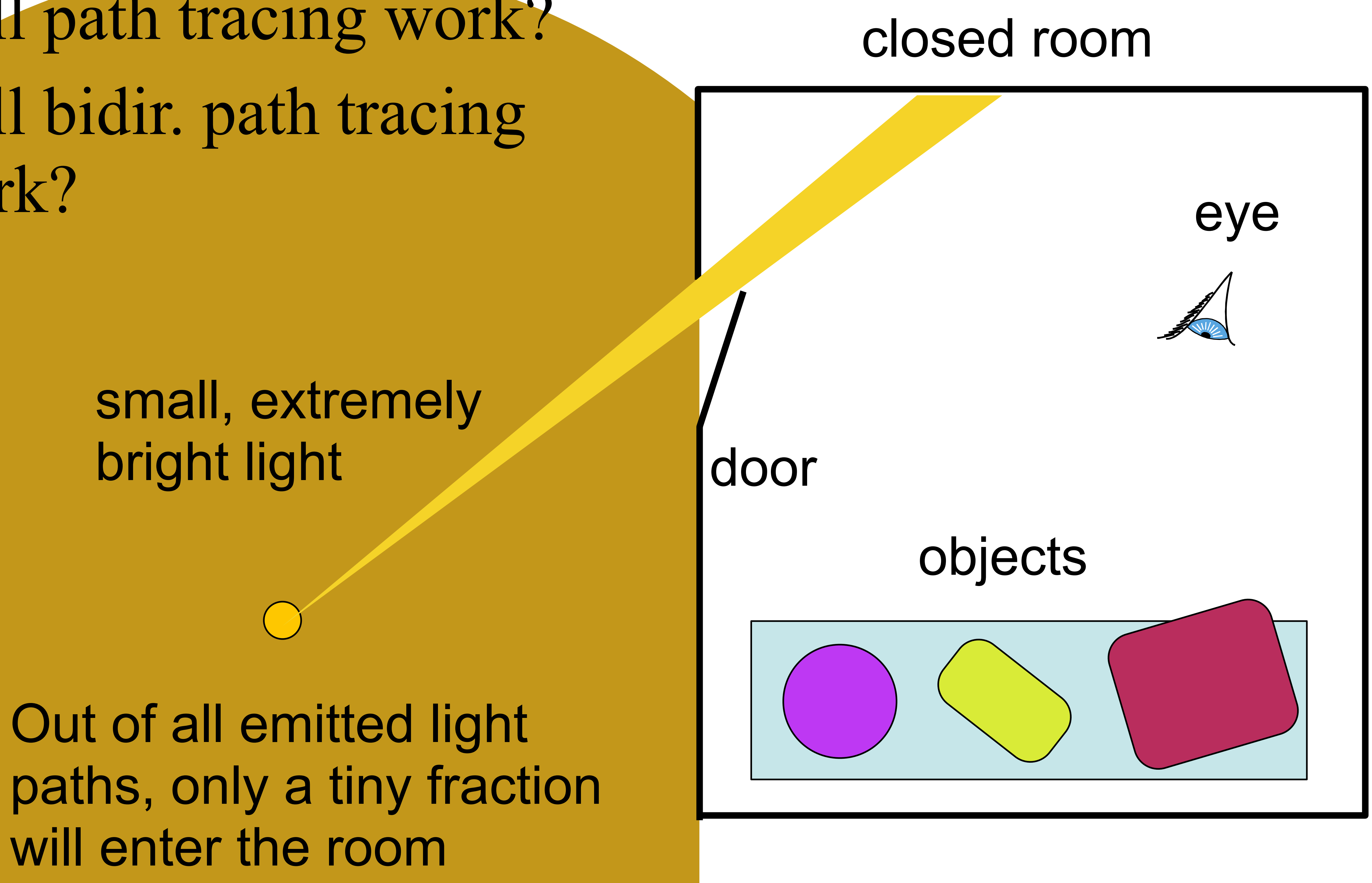
Well Is That the End, Then...?

- Will path tracing work?
- Will bidir. path tracing work?



Well Is That the End, Then...?

- Will path tracing work?
- Will bidir. path tracing work?

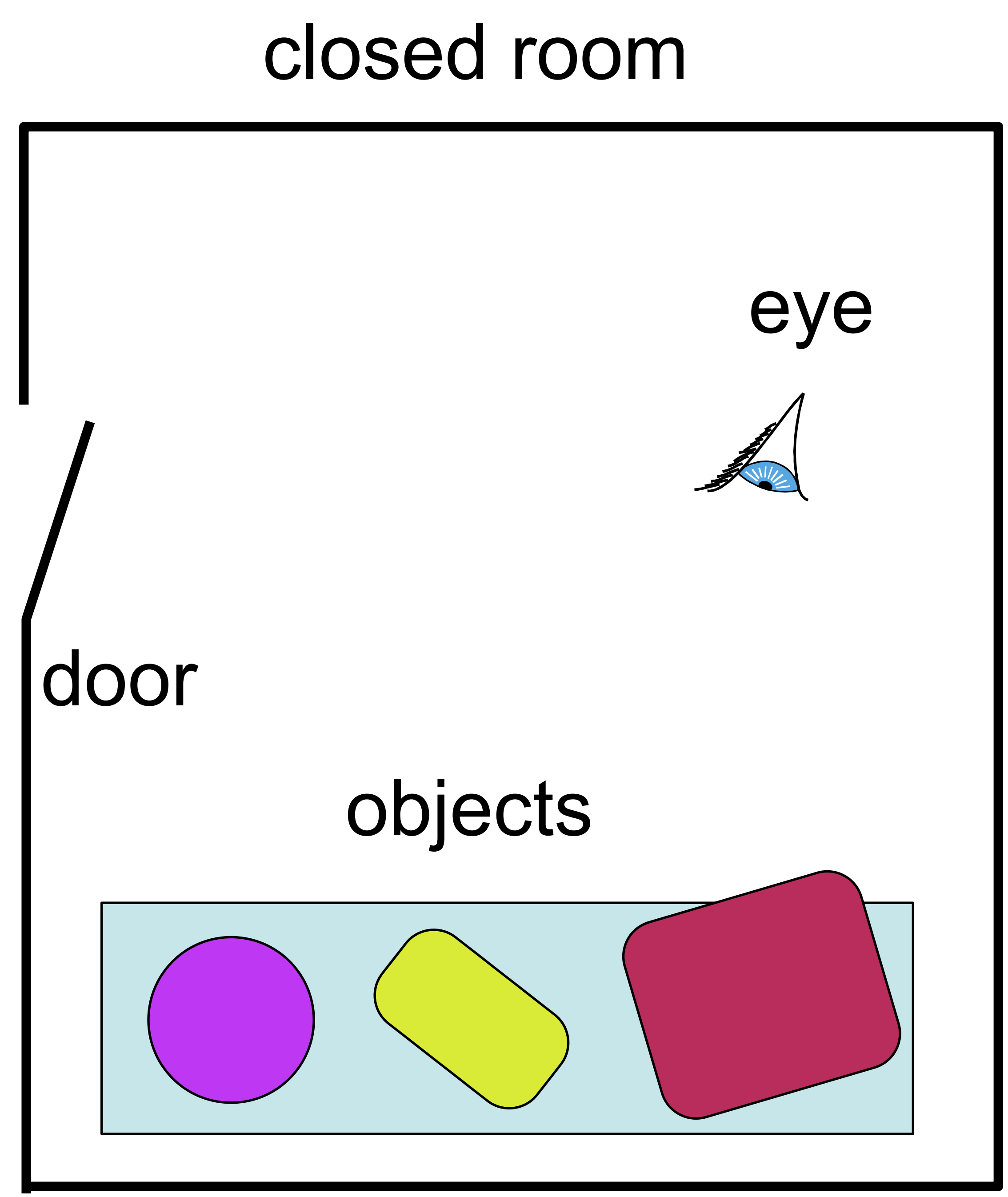


Might Look Like This

Eric Veach

Why Is This Hard?

- To reach the sensor, the light must enter the room through a narrow slit
  - Makes light sampling inefficient: how does the light know to shoot through the slit?
- And for eye rays to hit the light source, they must also find their way through the opening



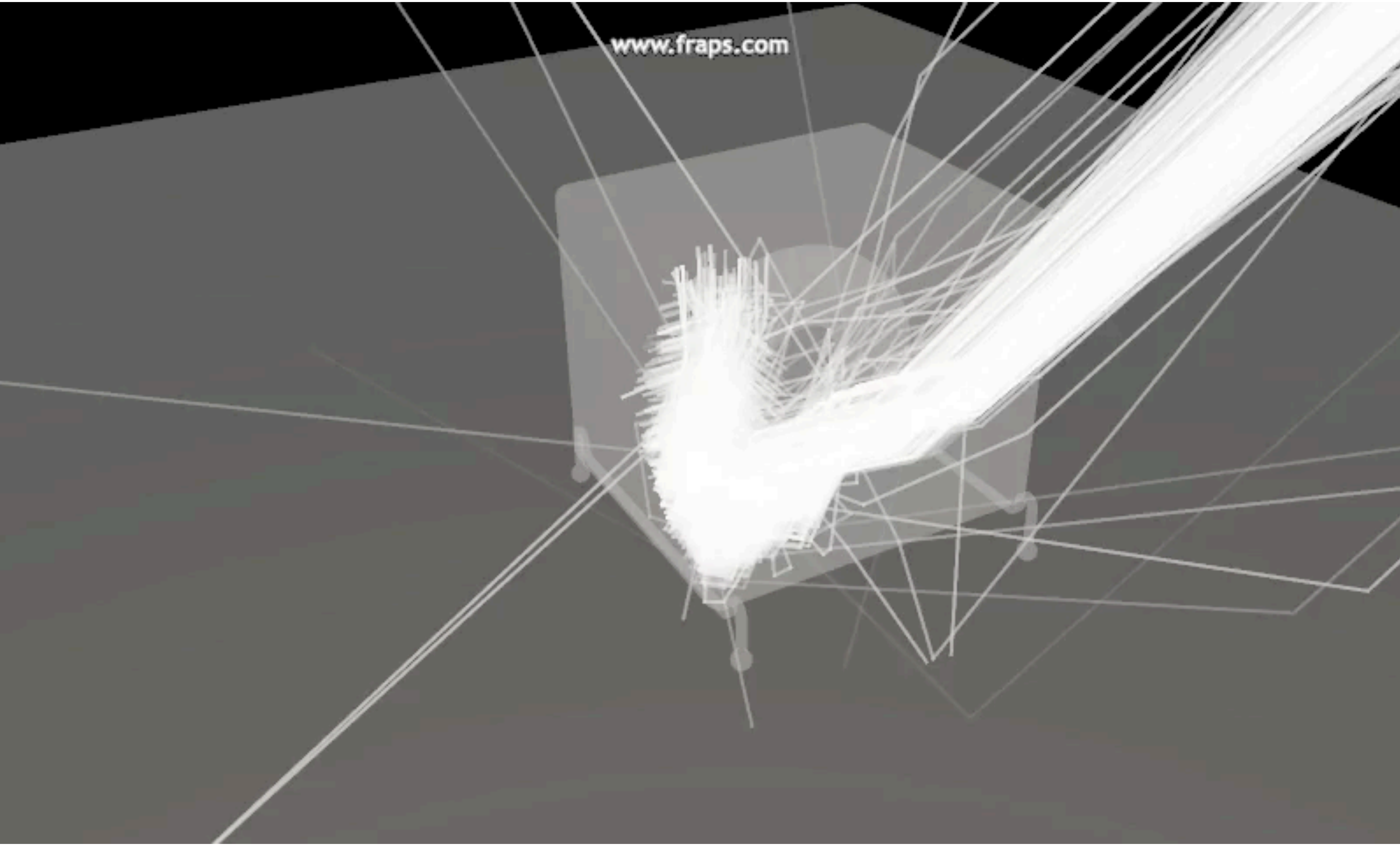
Metropolis Light Transport (MLT)

- Basic intuition: once we get lucky and find a light-carrying path, let's explore its local neighborhood to find others as well

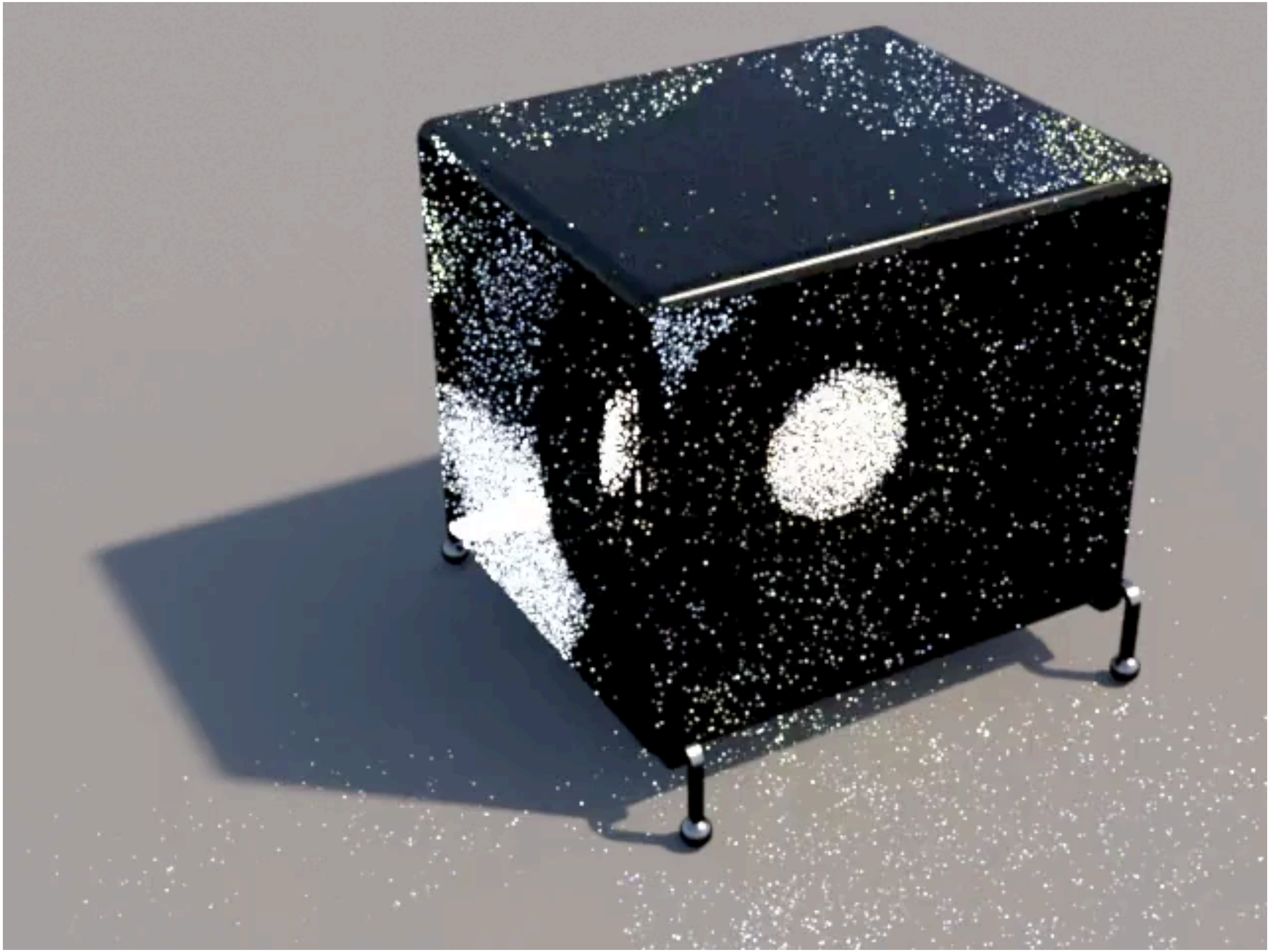
- Veach and Guibas 97

Metropolis Light Transport (MLT)

- Basic intuition: once we get lucky and find a light-carrying path, let's explore its local neighborhood to find others as well
- Basic idea
  - 0. start from a random light path X_0
  - 1. accumulate contribution of path X_i to image
  - 2. propose a new path X' near X_i
  - 3. compare the light throughput of X_i and X', and set X_{i+1}=X' if it carries more light, or sometimes even if it doesn't (if not, set $X_{i+1}=X_i$, that is, stay put)
  - 4. go to 1
- Sounds easy, right?



Markus Kettunen



Metropolis Light Transport

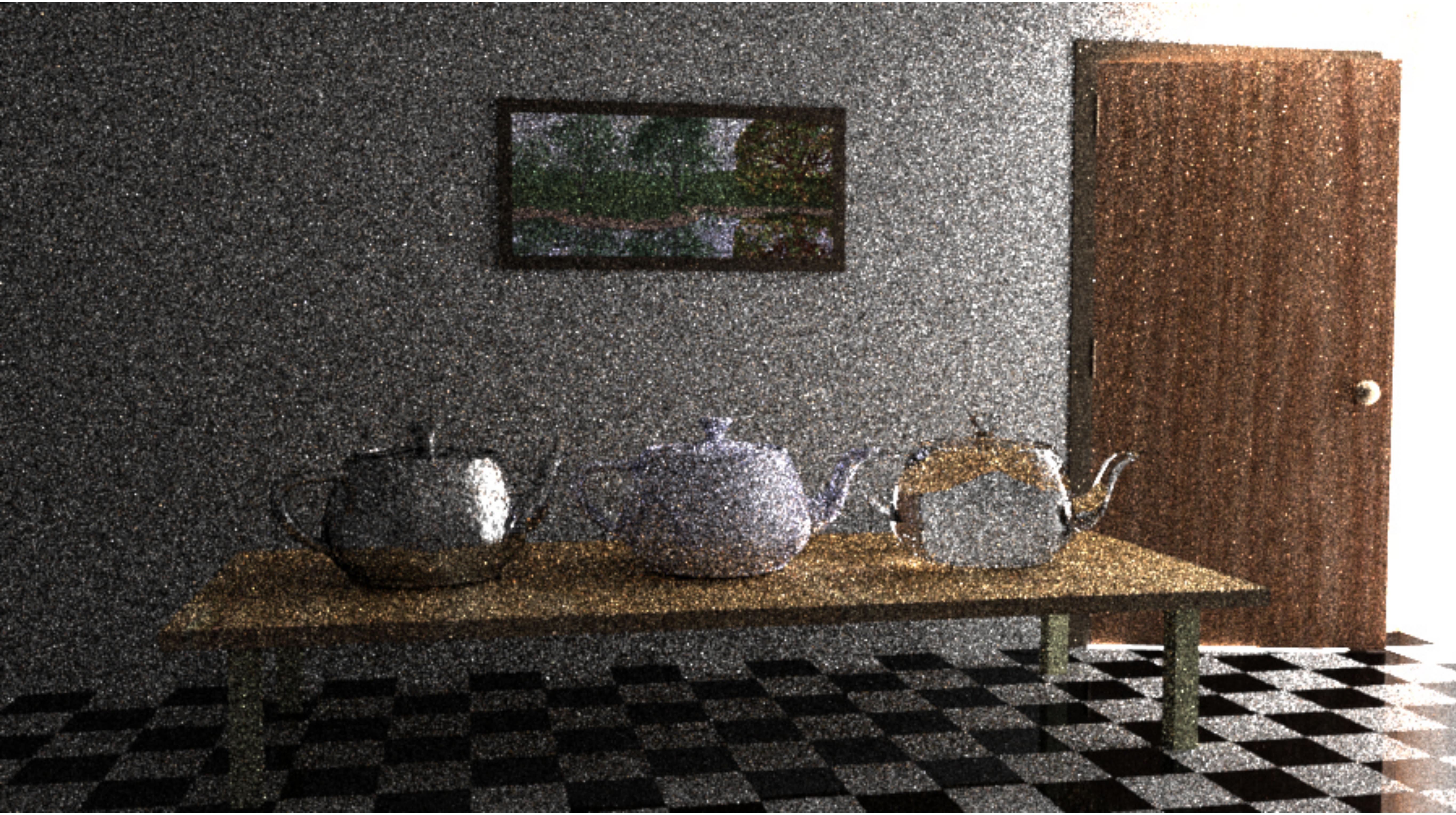
- Mathematically, the paths X_i form a Markov Chain that is distributed according to the light carrying power
  - Works on "path space" defined earlier
  - We'll leave it at that for now
- Truth is it's not very easy in practice
  - The devil is in the details: computing the acceptance probability is involved
  - There has been *no implementation* since Veach..
  - ..except now! Mitsuba by Wenzel Jakob has an open implementation, plus tons of other rendering algorithms
    - **Highly recommended** to download and play around with!

Metropolis Light Transport

- Mathematically, the paths X_i form a Markov Chain that is distributed according to the light carrying power
  - Works on "path space" defined earlier
  - We'll leave it at that for now
- Truth is it's not very easy in practice
  - The devil is in the details: computing the acceptance probability is involved
  - There has been *no implementation* since Veach..
  - ..except now! Mitsuba by W... impl... ...ering algorithms
    - ... to download and play around with!

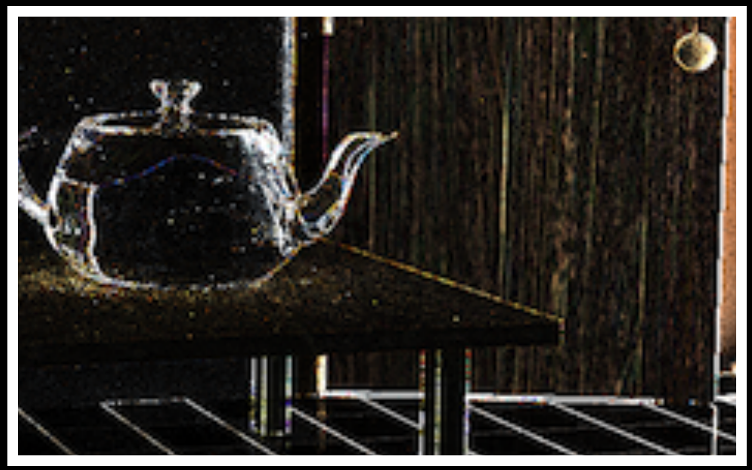
And Aether helps here, too!

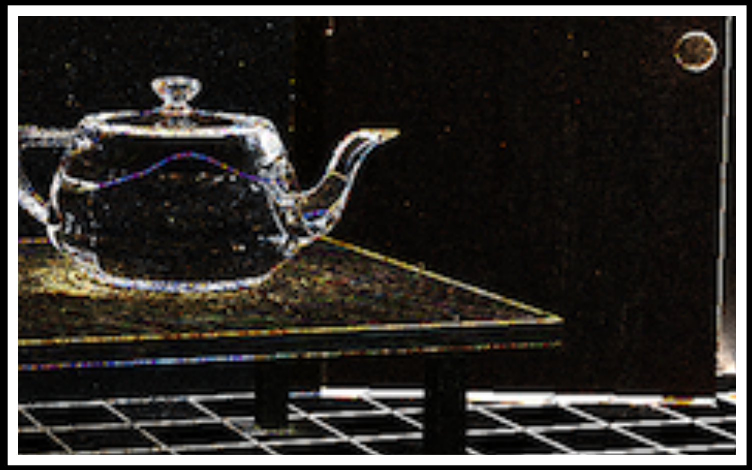
BDPT



Metropolis, Equal Time

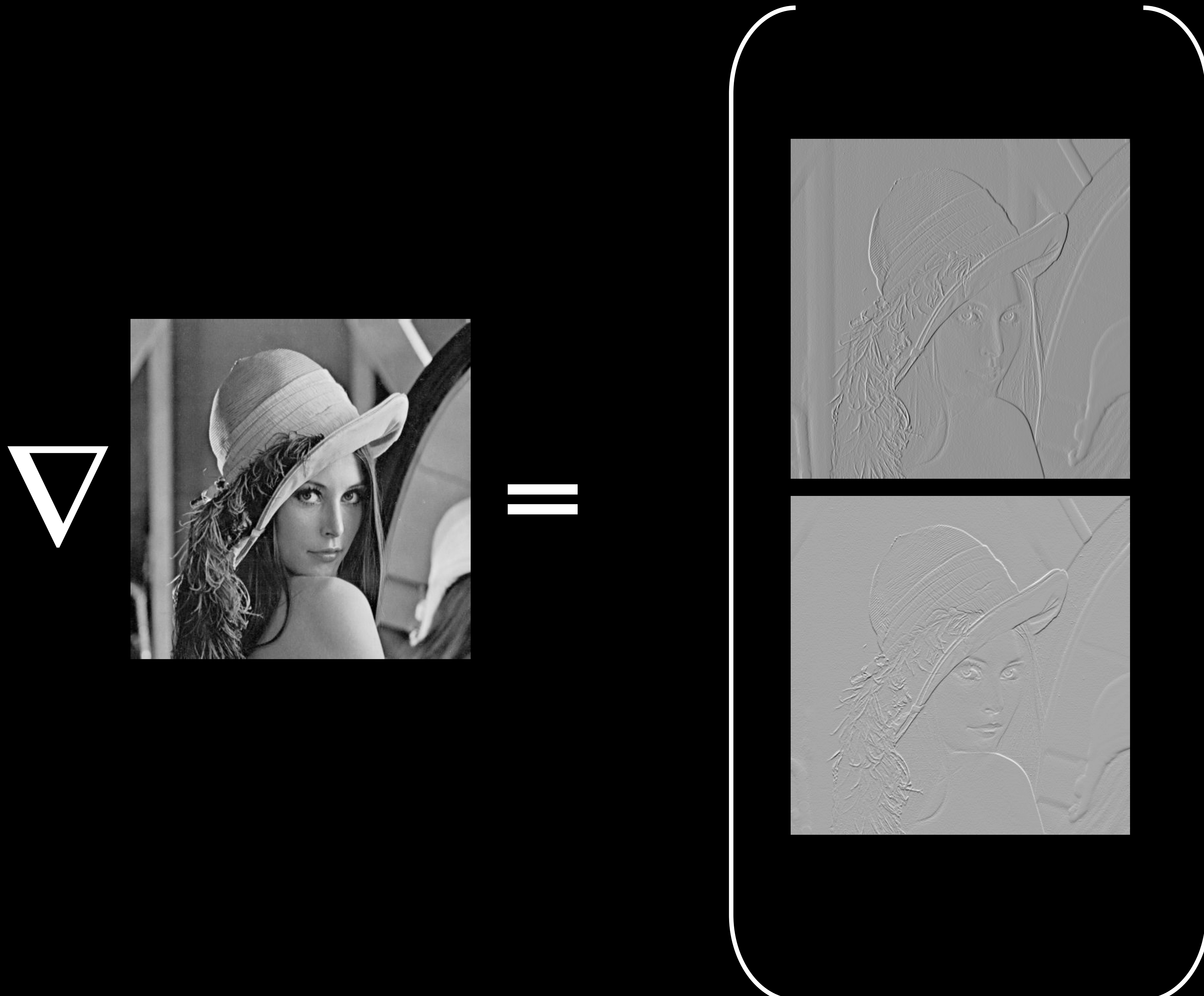
Gradient-Domain MLT (Lehtinen et al. 2013)

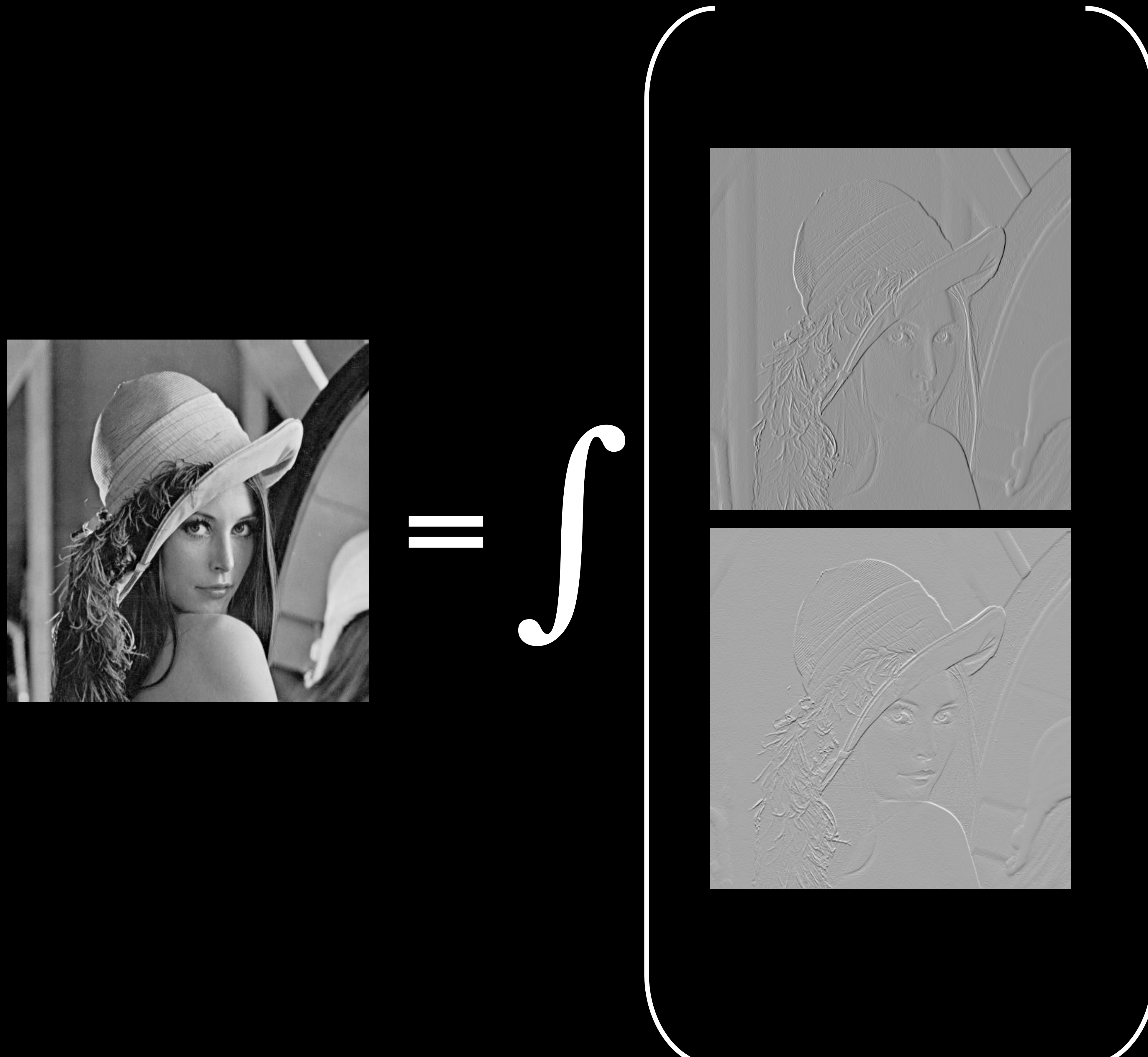

Horizontal Difference


Vertical Difference


Poisson Solver
(produces image, given gradient)


Final Image





New State of The Art

- Kettunen, Härkönen, Lehtinen, SIGGRAPH 2019 (Open Access from ACM)

Deep Convolutional Reconstruction For Gradient-Domain Rendering

MARKUS KETTUNEN, Aalto University
ERIK HÄRKÖNEN, Aalto University
JAAKKO LEHTINEN, Aalto University and Nvidia

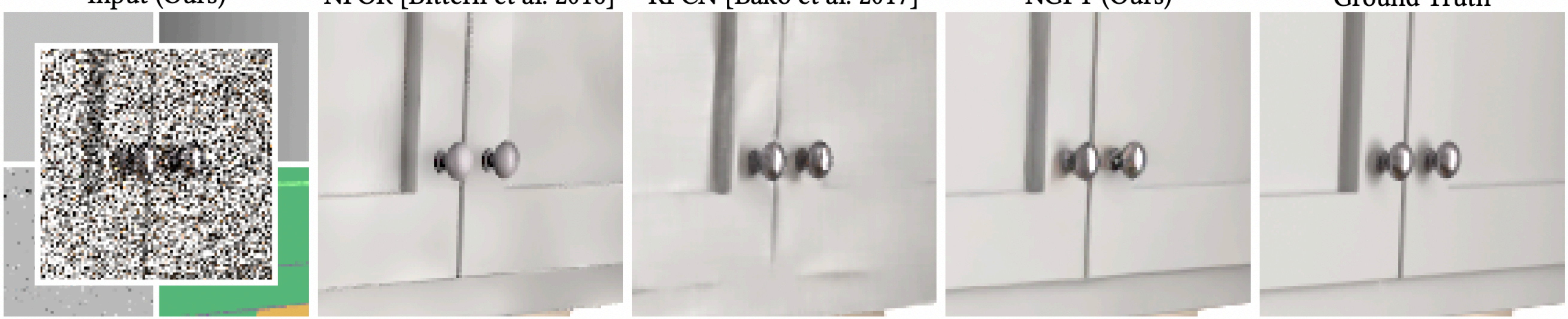


Fig. 1. Comparison of the primal-domain denoisers NFOR [Bitterli et al. 2016] and KPCN [Bako et al. 2017] to our gradient-domain reconstruction NGPT from very noisy equal-time inputs (8 samples for ours and 20 for others). Generally outperforming the comparison methods, our results show that gradient sampling is useful also in the context of non-linear neural image reconstruction, often resolving e.g. shadows better than techniques that do not make use of gradients.

Kelemen Metropolis

- Computation of the transition probabilities in path space is so hard to get right that an easier formulation was developed by Kelemen et al.
  - Path space with unlimited bounces is infinite dimensional
  - Kelemen maps paths from the surfaces to an infinite unit hypercube instead (just a change of variables)
  - Computations become a lot simpler, easier to implement
  - Unfortunately, results not as good, as recently demonstrated by Mitsuba

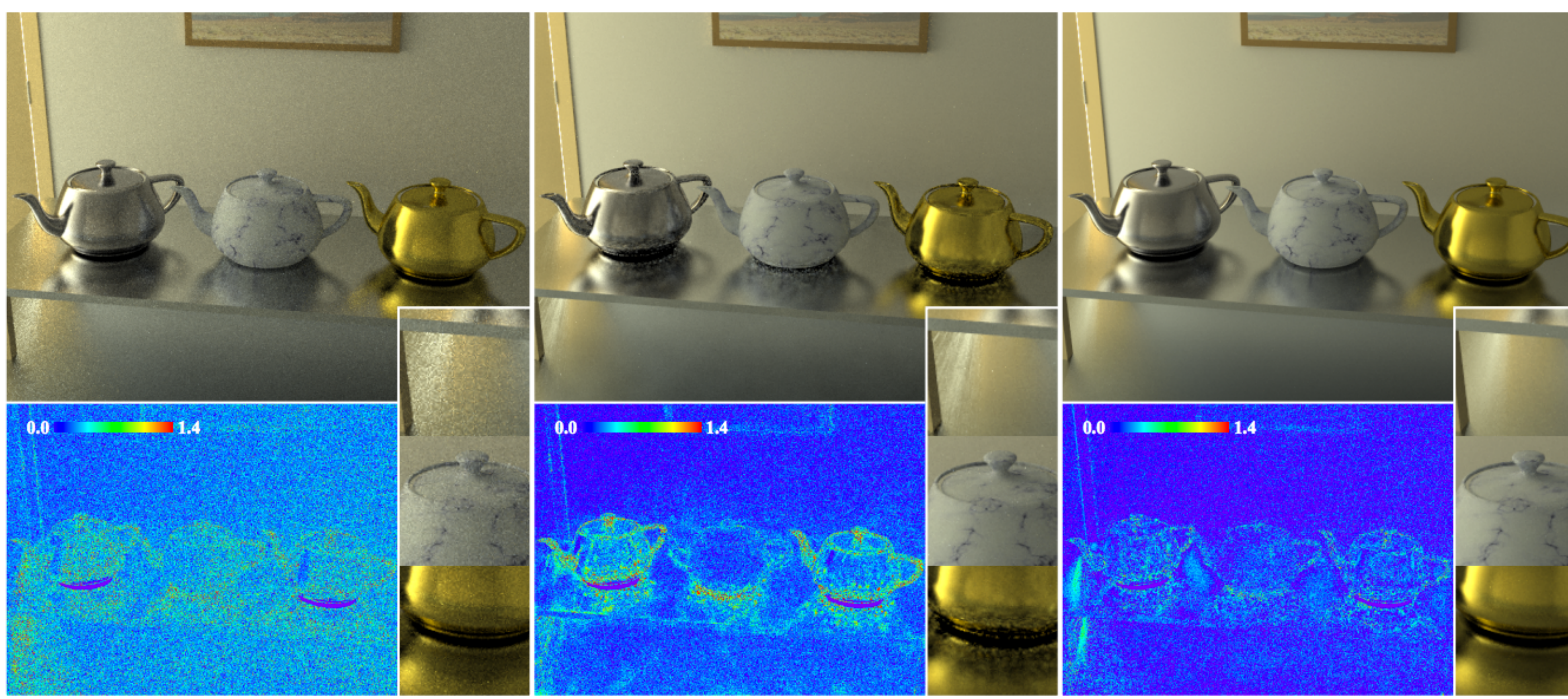
  - Now called Primary Sample Space MLT (PSSMLT)

Better Kelemen MLT

- Multiplexed Metropolis Light Transport (Hachisuka, Kaplanyan, Dachsbacher, SIGGRAPH 2013)
  - Combines MIS and Primary Sample Space

Multiplexed Metropolis Light Transport

Toshiya Hachisuka[1] Anton S. Kaplanyan[2] Carsten Dachsbacher[2]
[1]Aarhus University [2]Karlsruhe Institute of Technology



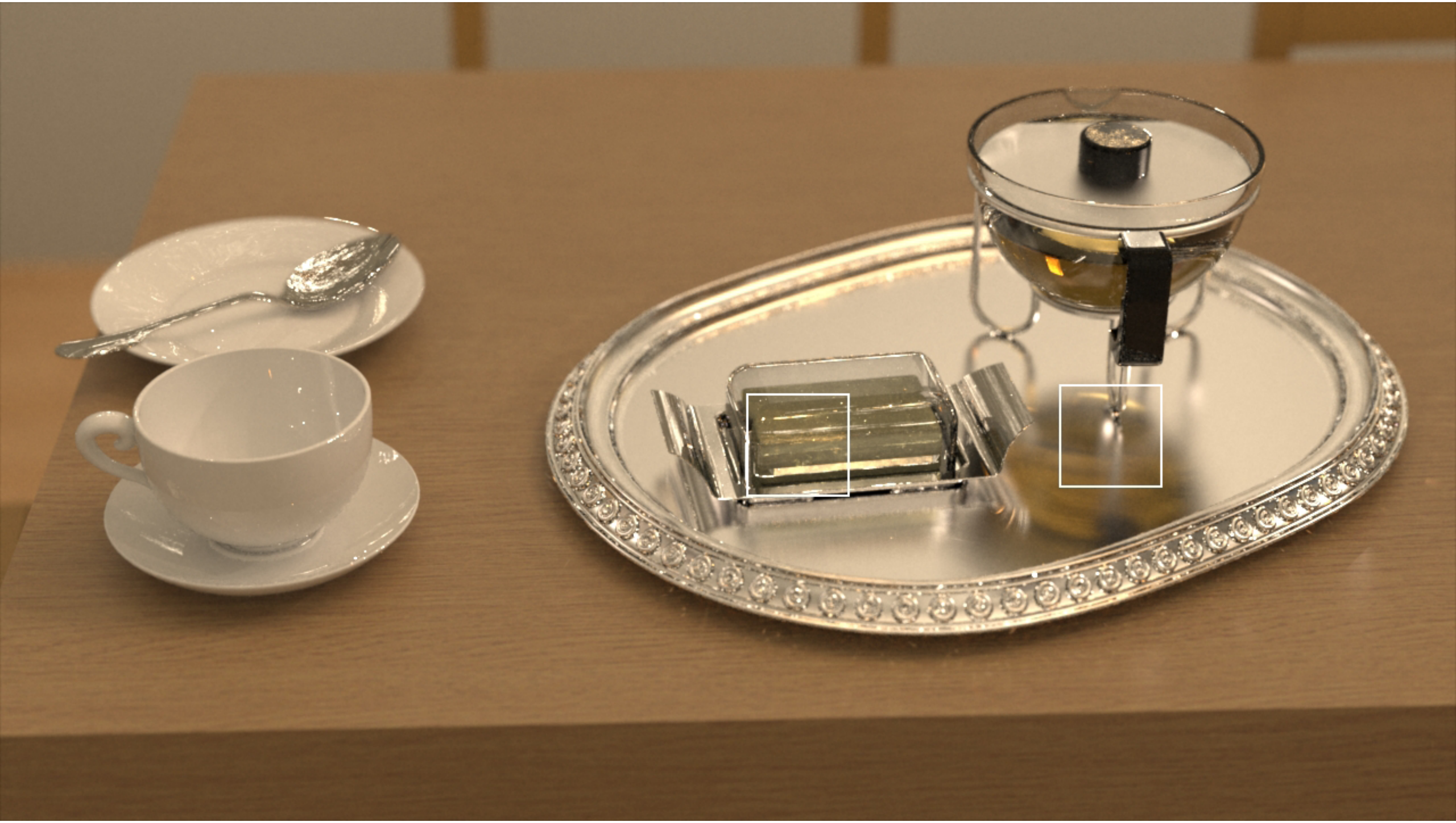
PSSMLT (RMSE: 0.073906) Original MLT (RMSE: 0.042226) MMLT (RMSE: 0.041381)

Manifold Exploration, ERPT

- For extremely difficult cases, Wenzel Jakob devised a technique for locally exploring neighborhoods in caustics (Jakob and Marschner, SIGGRAPH 2012)
  - First big extension to MLT in a decade
  - **Watch the video on the linked page!**

- Energy Redistribution Path Tracing or ERPT (Cline et al. SIGGRAPH 2005) is a variant of MLT that runs very short chains for local exploration

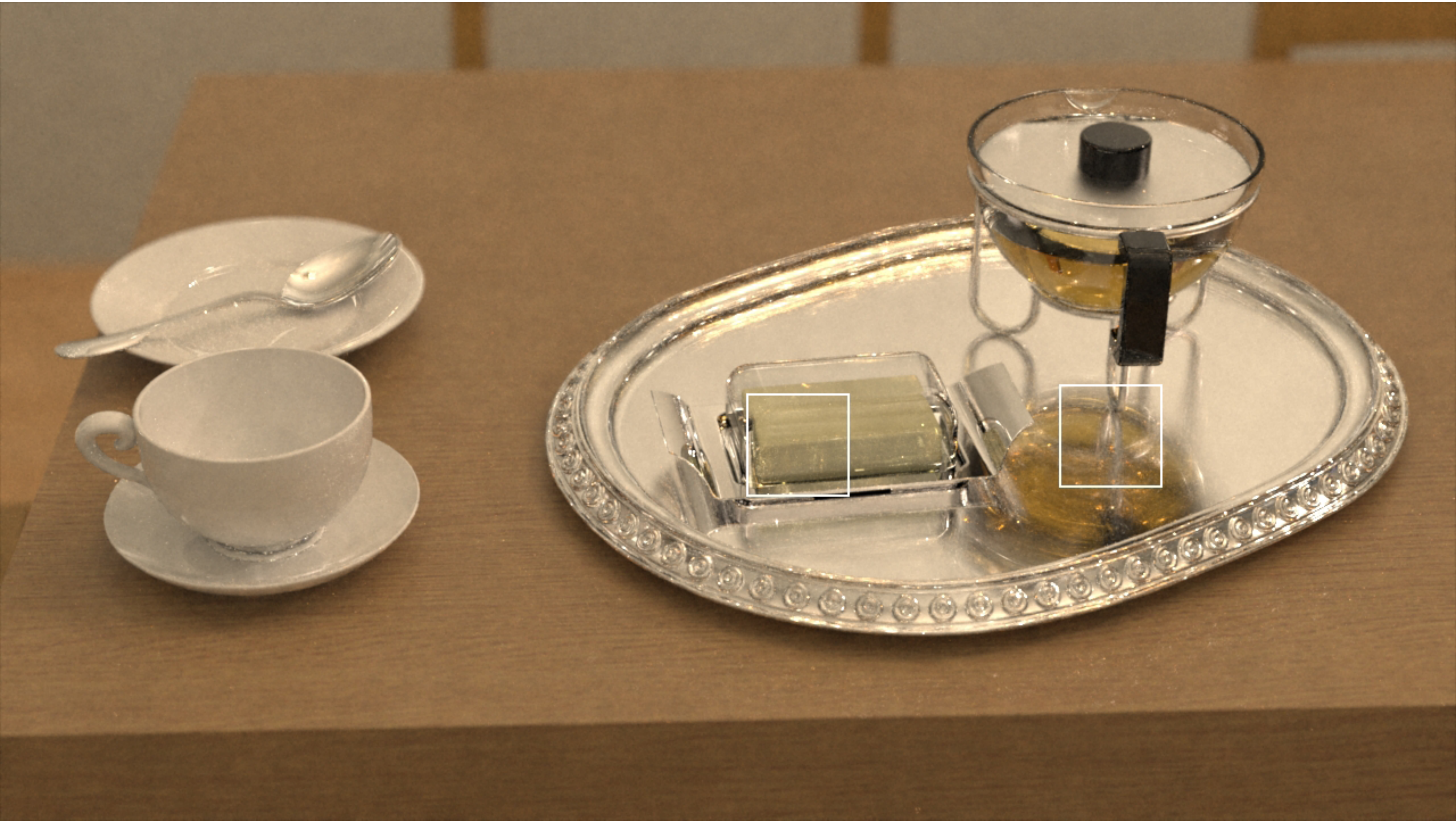
MLT Result w/o Manifolds



Text

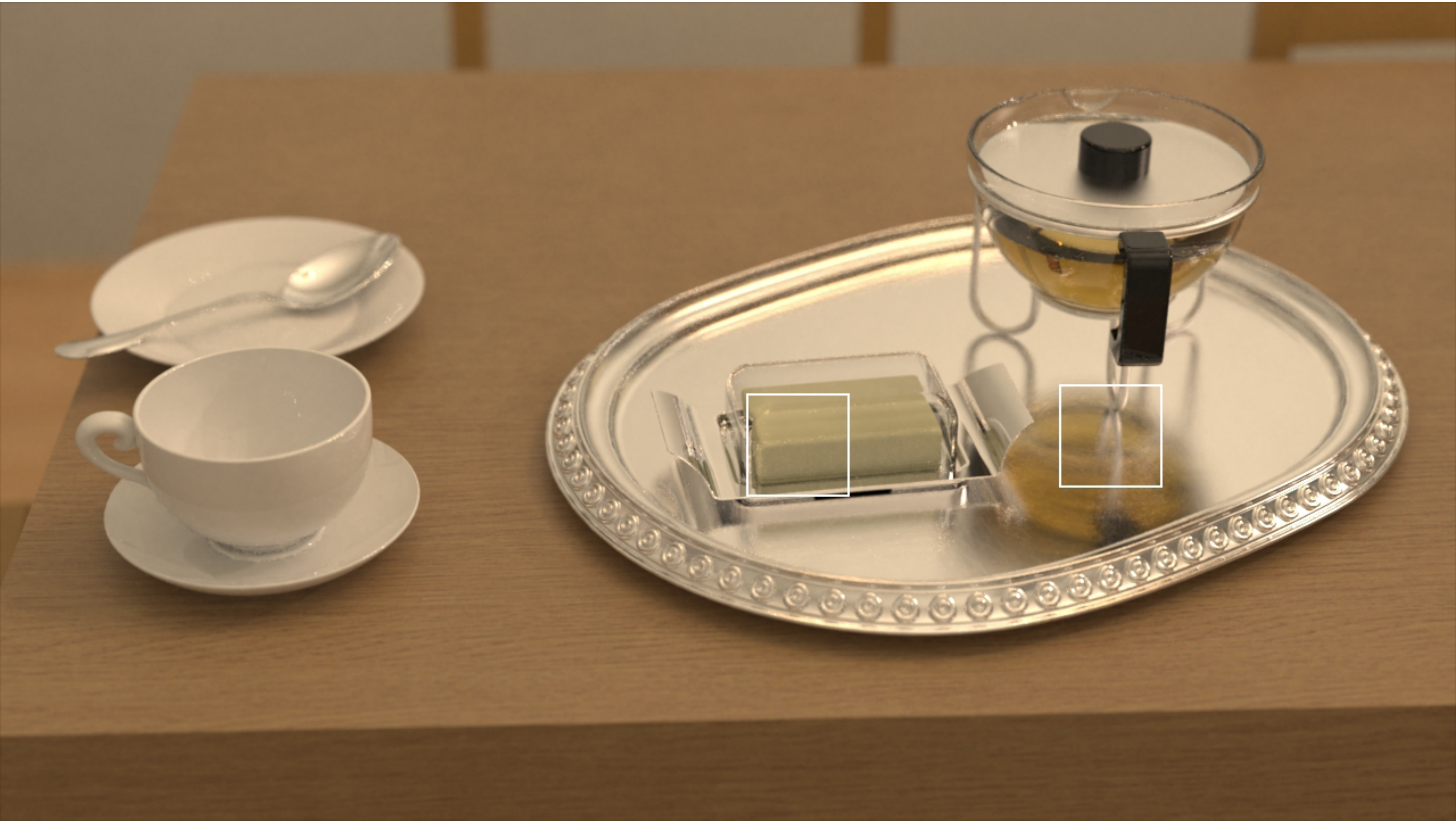
Jakob and Marschner

Manifold Exploration Result (eq.time)



Jakob and Marschner

Reference

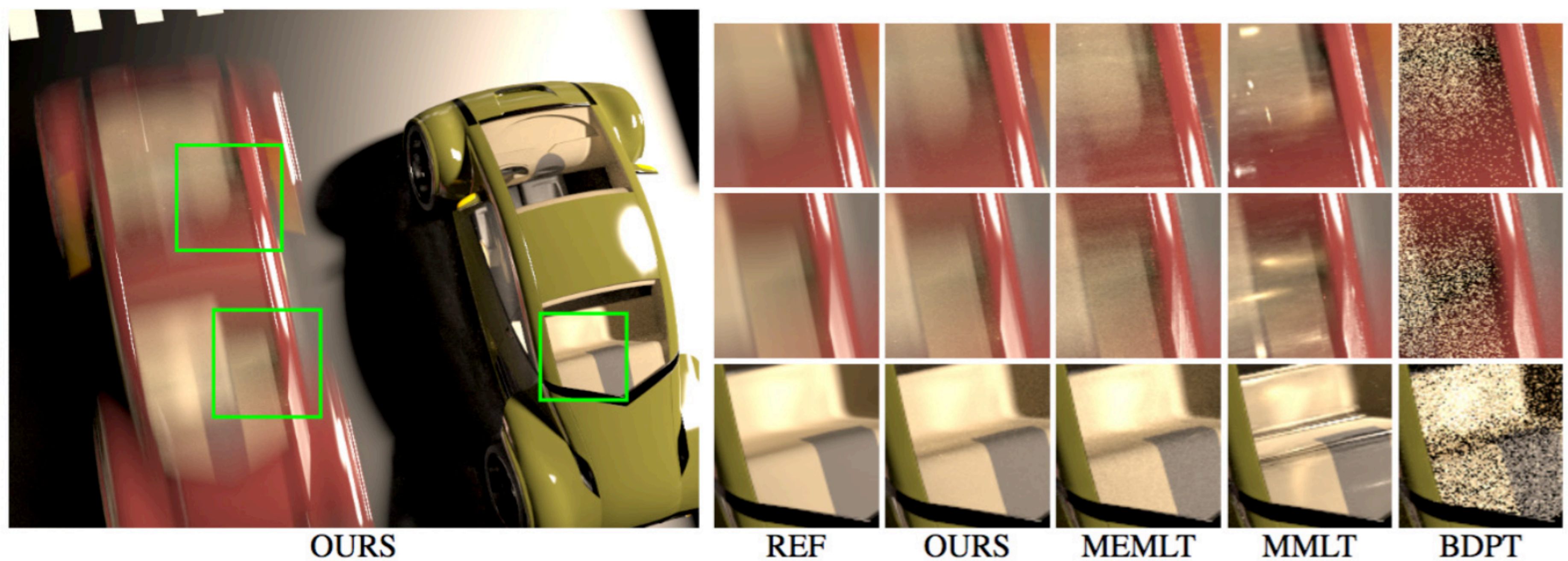


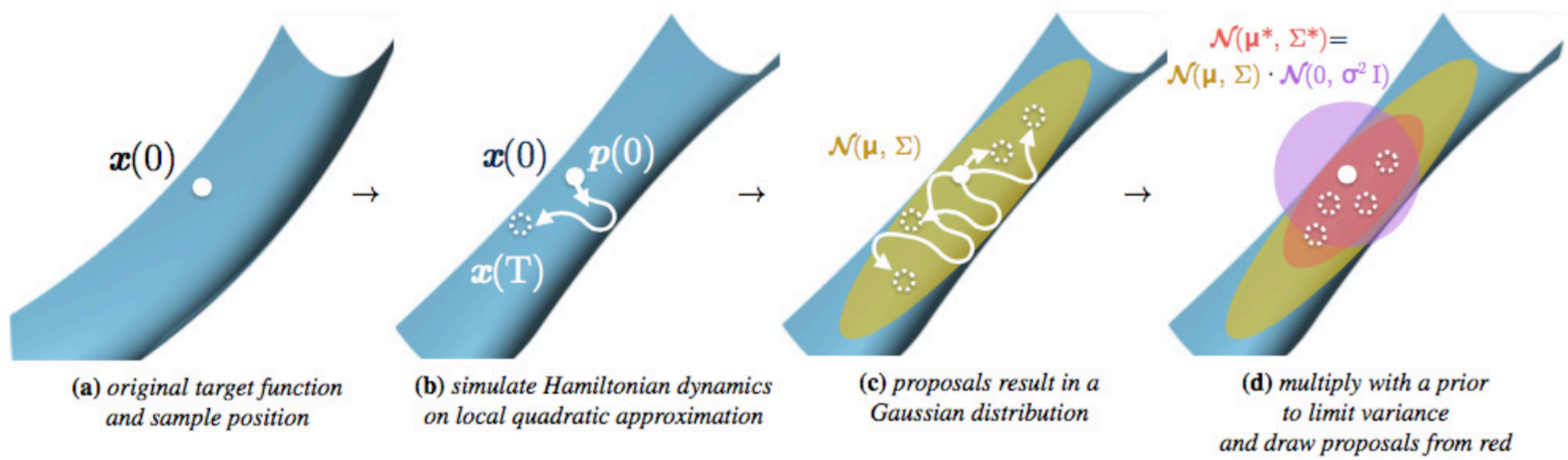
The Light Source in Prev. Picture

Jakob and Marschner

Hamiltonian Monte Carlo

- Li, Lehtinen, Ramamoorthi, Jakob, Durand, SIGGRAPH Asia 2015
- See great talk by Tzu-Mao





Comparisons

- In very difficult situations, Metropolis rocks..
- ..but in simpler, easy transport situations such as diffuse GI with no hard cases like bright indirect sources, PT/BDPT blow it out of the water

- Why? The samples produced by Metropolis are bad
  - Not stratified, not low discrepancy
  - **If you can stratify MLT, come talk to me and we'll make you famous :)**

Closing Remarks for Path Sampling

- (Bidirectional) Path tracing, MLT are *unbiased*
  - Means they will give you the correct answer *on average*
- E.g. radiosity is not unbiased
  - Has systematic error *(what kind..?)*
  - But it is *consistent*, meaning it will converge to the correct solution when you refine the mesh and compute radiosities with better and better sampling