TIEA311
Tietokonegrafiikan perusteet
kevät 2019

(“Principles of Computer Graphics” – Spring 2019)

TIEA311 Tietokonegrafiikan perusteet – kevät 2019 (“Principles of Computer Graphics” – Spring 2019)

Adapted from: *Wojciech Matusik*, and *Frédo Durand*: 6.837 Computer Graphics. Fall 2012. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu/.

Frontpage of the local course version, held during Spring 2019 at the Faculty of Information technology, University of Jyväskylä:
`http://users.jyu.fi/~nieminen/tgp19/`

TIEA311 - Additional material

In 2018, we went to light-speed for these slides.

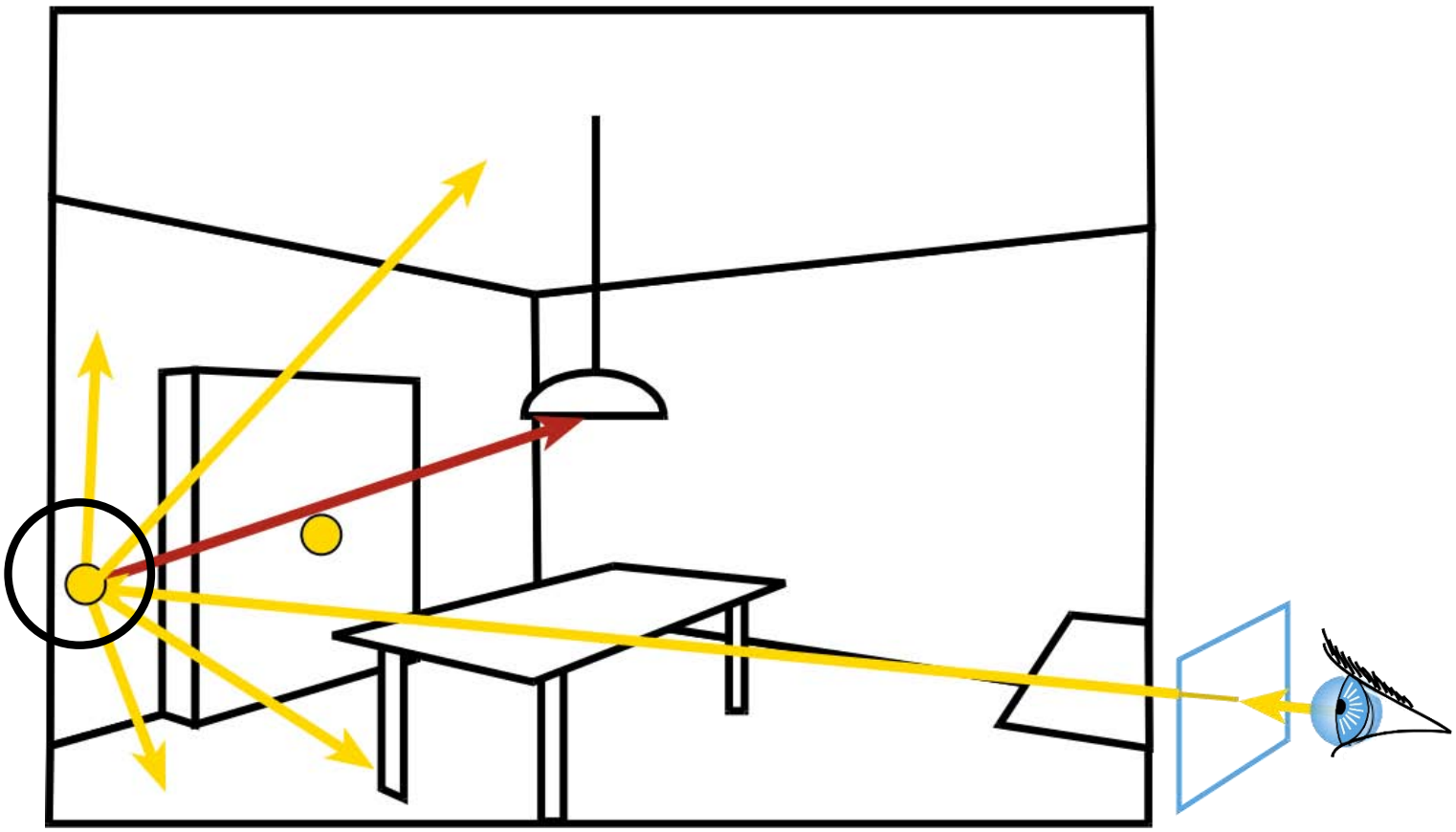
They are “nice to know” if you decide to continue with computer graphics or related research topics. Many pointers to related fields in mathematics, statistics, and numerics.

Global Illumination and Monte Carlo

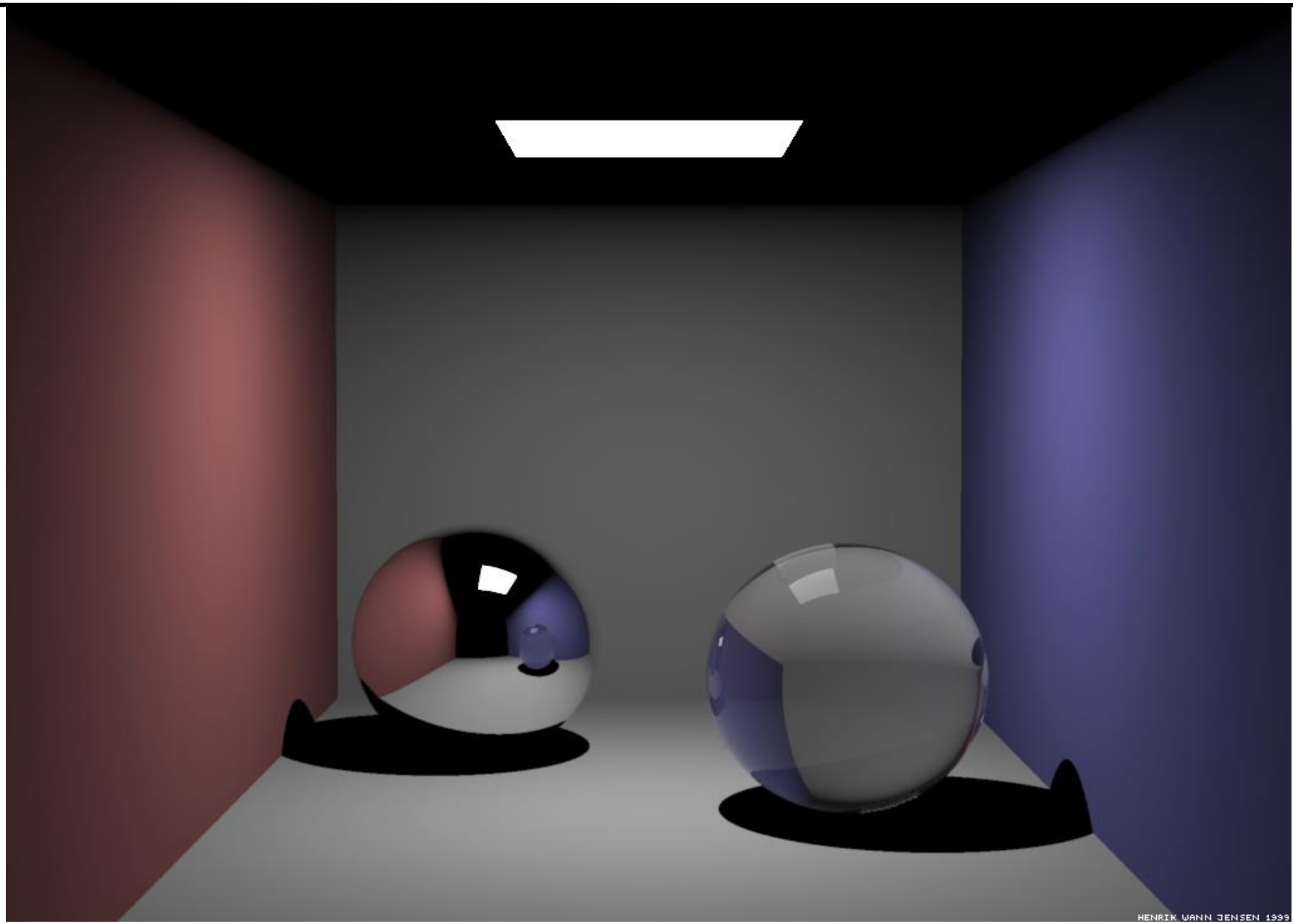
MIT EECS 6.837 Computer Graphics
Wojciech Matusik
with many slides from Fredo Durand and Jaakko Lehtinen

Global Illumination

- So far, we've seen only direct lighting (red here)
- We also want indirect lighting
  - Full integral of all directions (multiplied by BRDF)
  - In practice, send tons of random rays

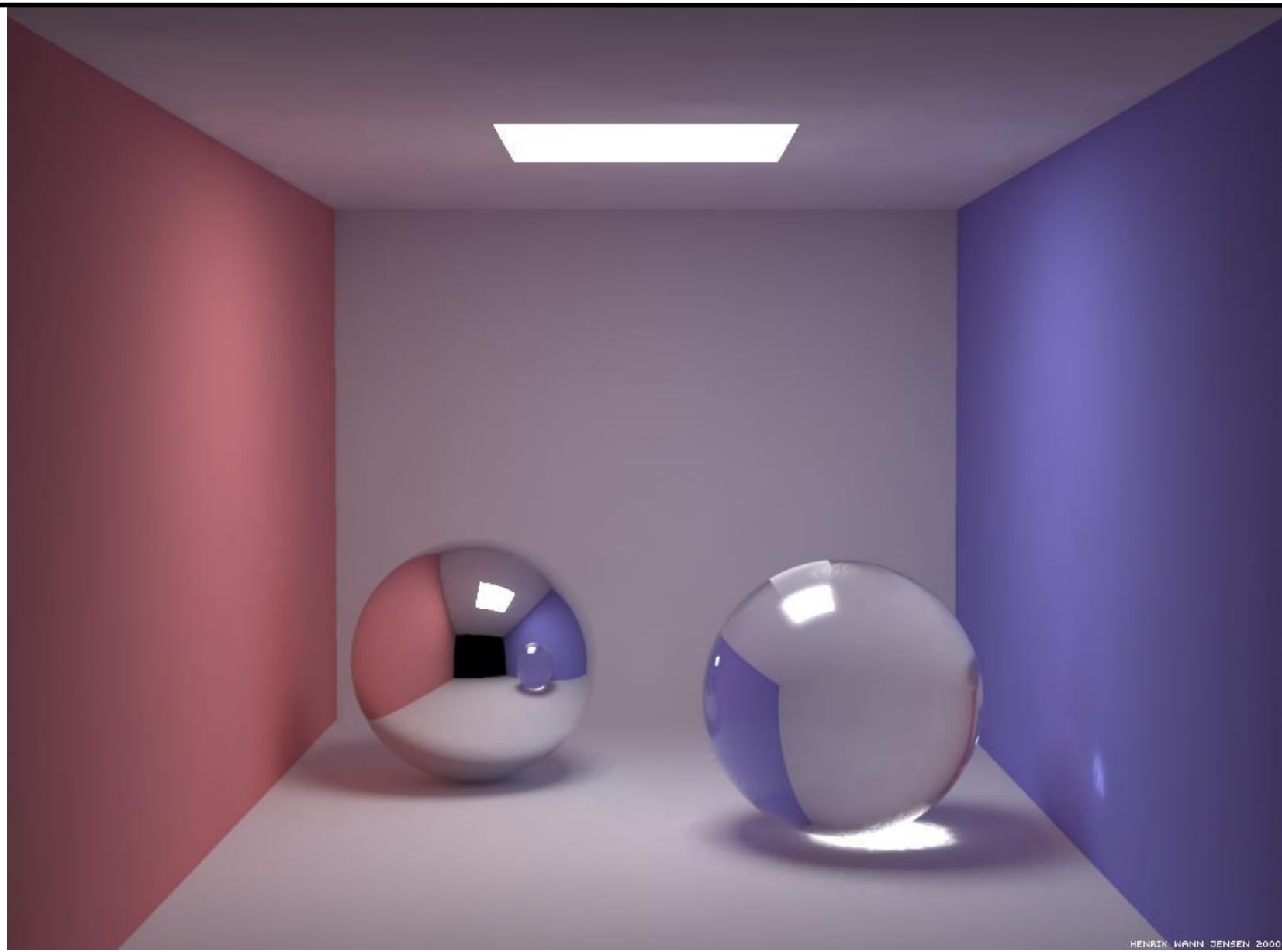


Direct Illumination



Courtesy of Henrik Wann Jensen. Used with permission.

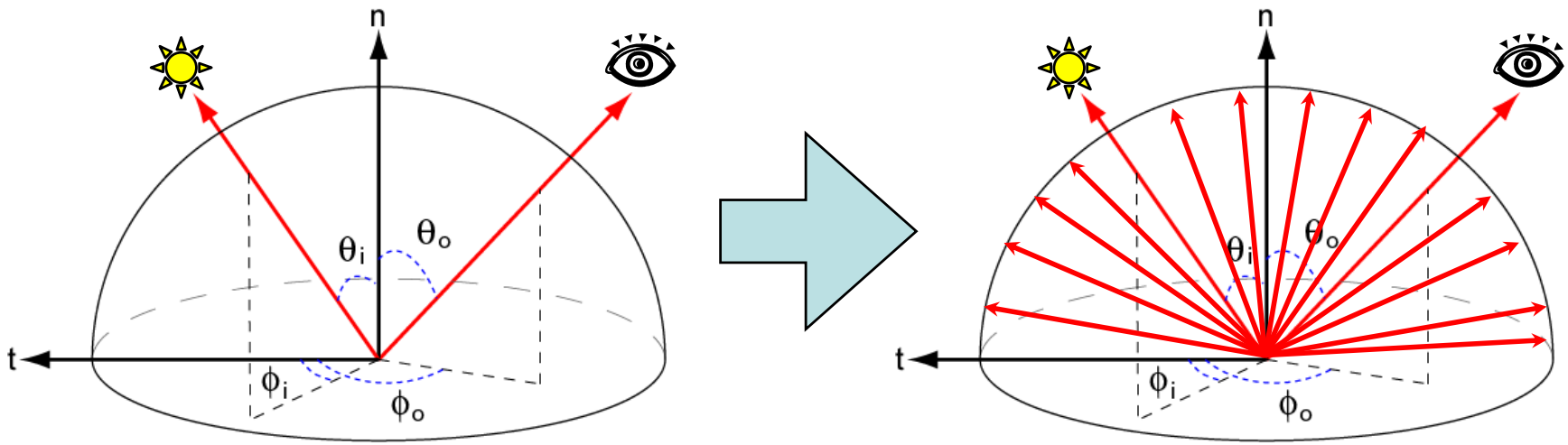
Global Illumination (with Indirect)



Courtesy of Henrik Wann Jensen. Used with permission.

Global Illumination

- So far, we only used the BRDF for point lights
  - We just summed over all the point light sources
- BRDF also describes how indirect illumination reflects off surfaces
  - Turns summation into integral over hemisphere
  - As if *every* direction had a light source



Reflectance Equation, Visually

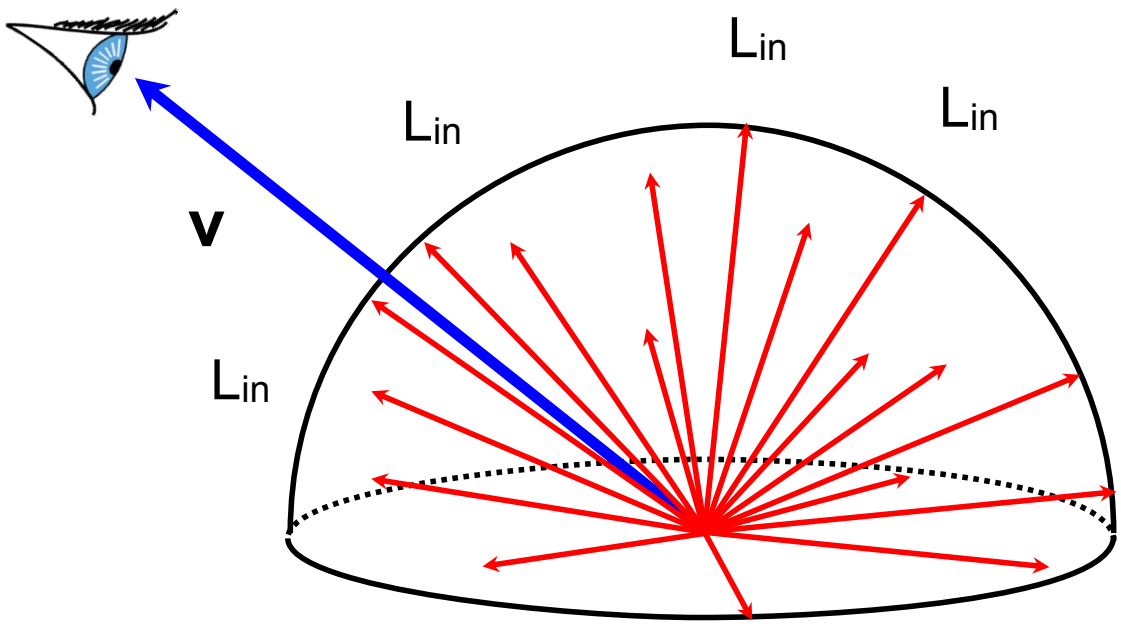
$$L_{\text{out}}(x, \boldsymbol{v}) = \int_{\Omega} L_{\text{in}}(\boldsymbol{l})\, f_r(x, \boldsymbol{l}, \boldsymbol{v})\, \cos\theta \, \mathrm{d}l$$

outgoing light to direction v

incident light from direction omega

the BRDF

cosine term

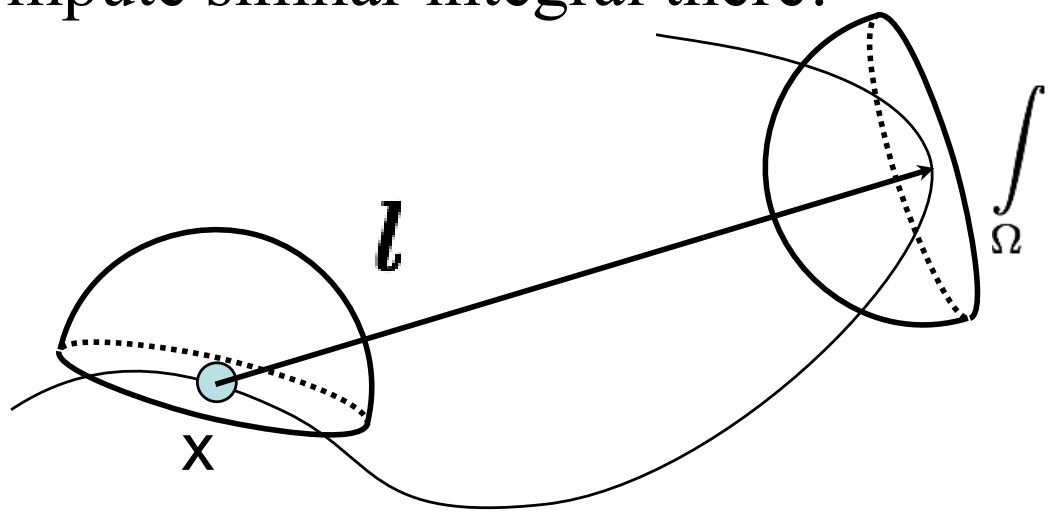


Sum (integrate) over every direction on the hemisphere, modulate incident illumination by BRDF

The Reflectance Equation

$$L_{\text{out}}(x, \boldsymbol{v}) = \int_{\Omega} L_{\text{in}}(\boldsymbol{l})\, f_r(x, \boldsymbol{l}, \boldsymbol{v})\, \cos\theta \, \mathrm{d}l$$

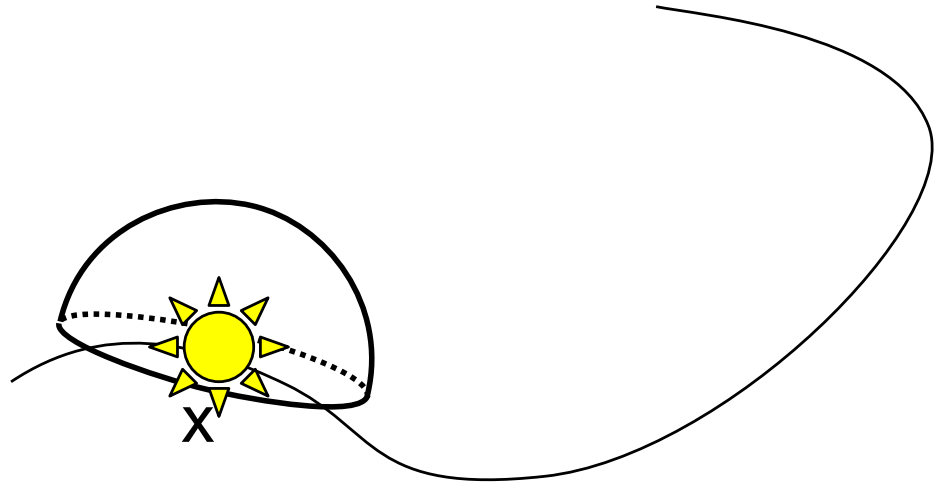
- Where does L_{in} come from?
  - It is the light reflected towards x from the surface point in direction ***l*** ==> must compute similar integral there!
    - Recursive!



The Rendering Equation

$$L_{\text{out}}(x, \boldsymbol{v}) = \int_{\Omega} L_{\text{in}}(\boldsymbol{l})\, f_r(x, \boldsymbol{l}, \boldsymbol{v})\, \cos\theta \, \mathrm{d}l + E_{\text{out}}(x, \boldsymbol{v})$$

- Where does L_{in} come from?
  - It is the light reflected towards x from the surface point in direction ***l*** ==> must compute similar integral there!
    - Recursive!
  - AND if x happens to be a light source, we add its contribution directly



The Rendering Equation

$$L_{\text{out}}(x, \boldsymbol{v}) = \int_{\Omega} L_{\text{in}}(\boldsymbol{l})\, f_r(x, \boldsymbol{l}, \boldsymbol{v})\, \cos\theta \, \mathrm{d}l + E_{\text{out}}(x, \boldsymbol{v})$$

- The rendering equation describes the appearance of the scene, including direct and indirect illumination
  - An “integral equation”, the unknown solution function L is both on the LHS and on the RHS inside the integral
    - Must either discretize or use Monte Carlo integration
  - Originally described by Kajiya and Immel et al. in 1986
  - More on 6.839
    - Also, see book references towards the end

The Rendering Equation

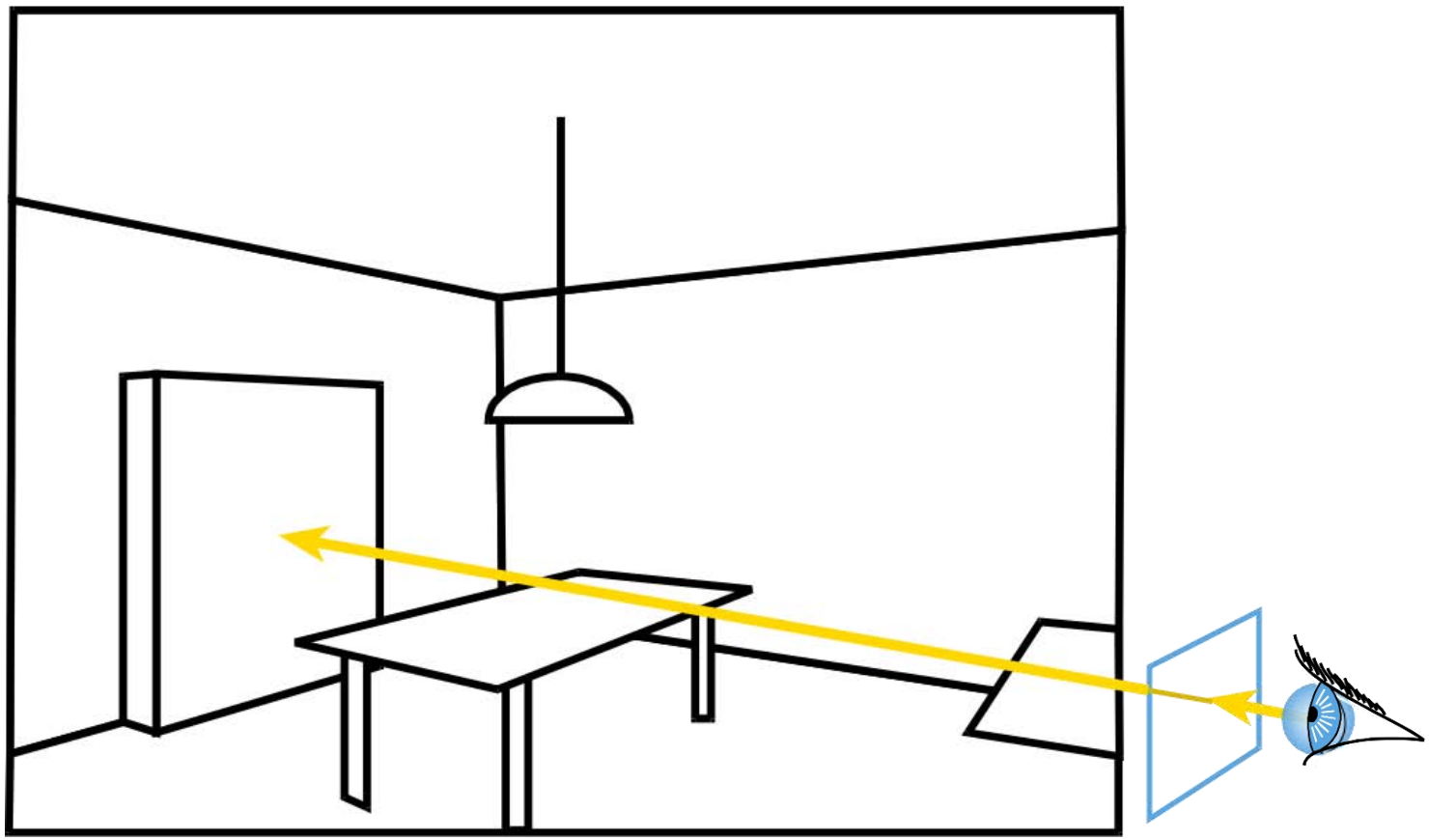
- Analytic solution is usually impossible
- Lots of ways to solve it approximately
- Monte Carlo techniques use random samples for evaluating the integrals
  - We'll look at some simple method in a bit...
- Finite element methods discretize the solution using basis functions (again!)
  - Radiosity, wavelets, precomputed radiance transfer, etc.

How To Render Global Illumination?

Lehtinen et al. 2008

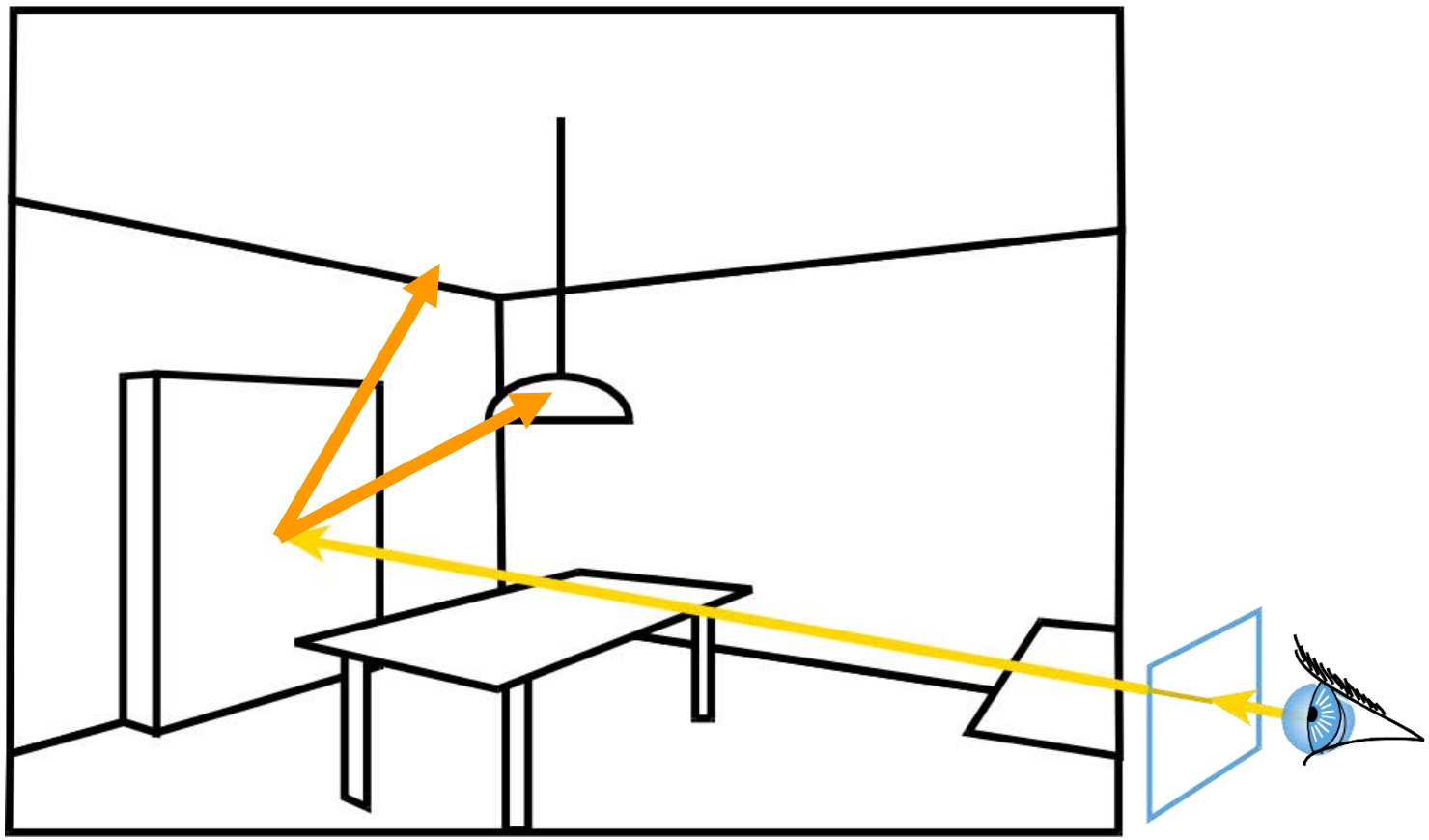
Ray Casting

- Cast a ray from the eye through each pixel



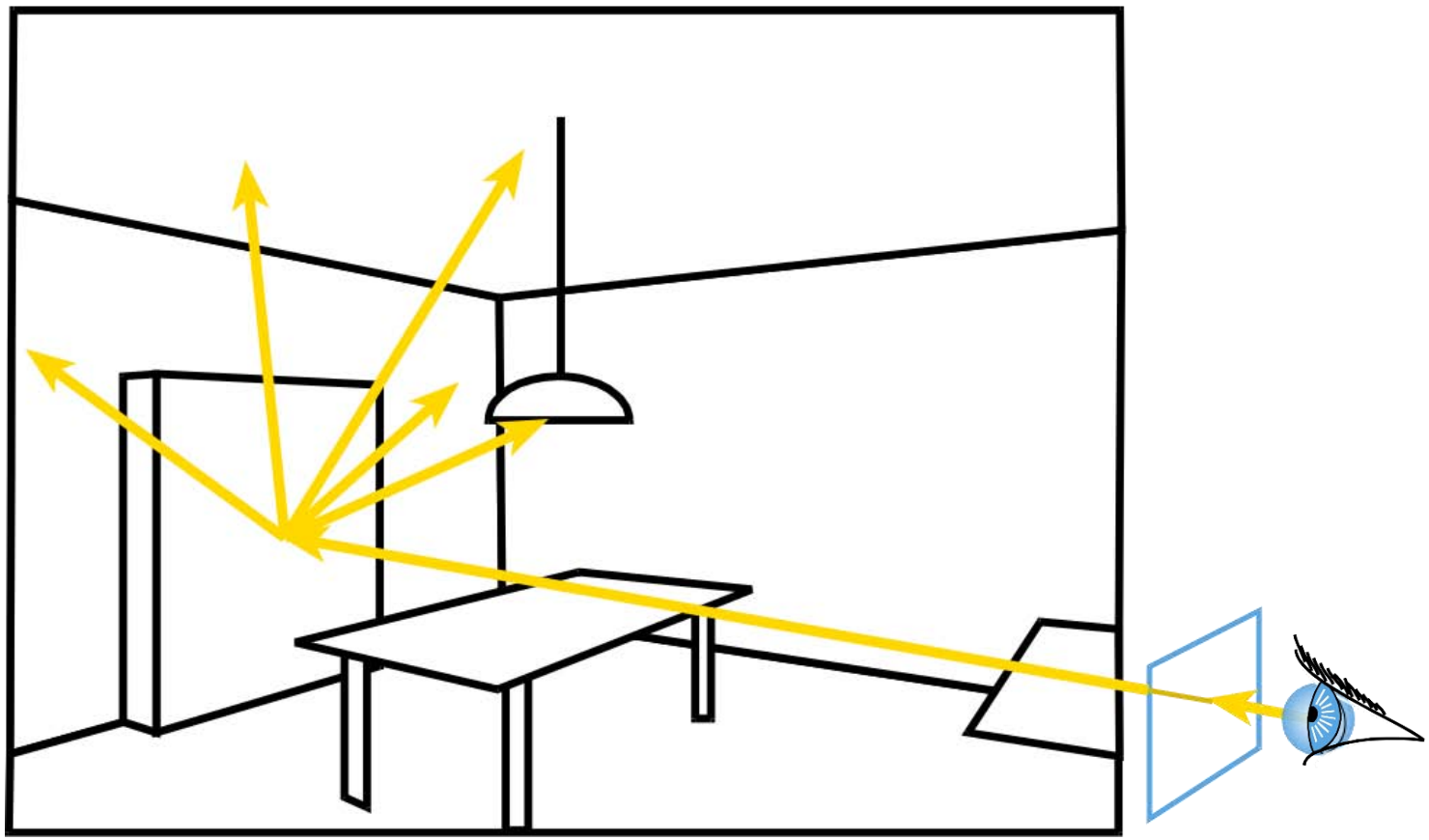
Ray Tracing

- Cast a ray from the eye through each pixel
- Trace secondary rays (shadow, reflection, refraction)



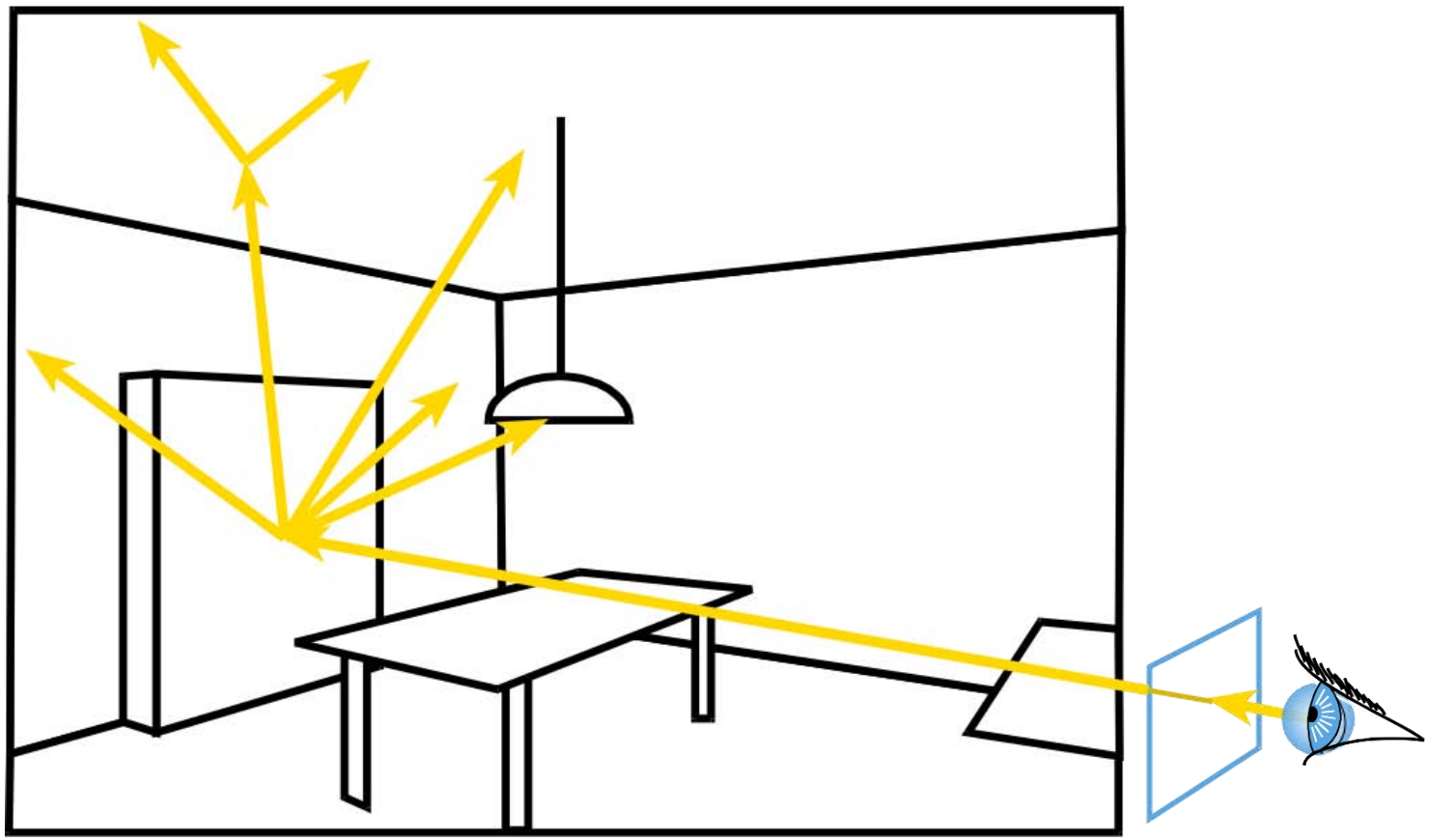
“Monte-Carlo Ray Tracing”

- Cast a ray from the eye through each pixel
- Cast random rays from the hit point to evaluate hemispherical integral using random sampling



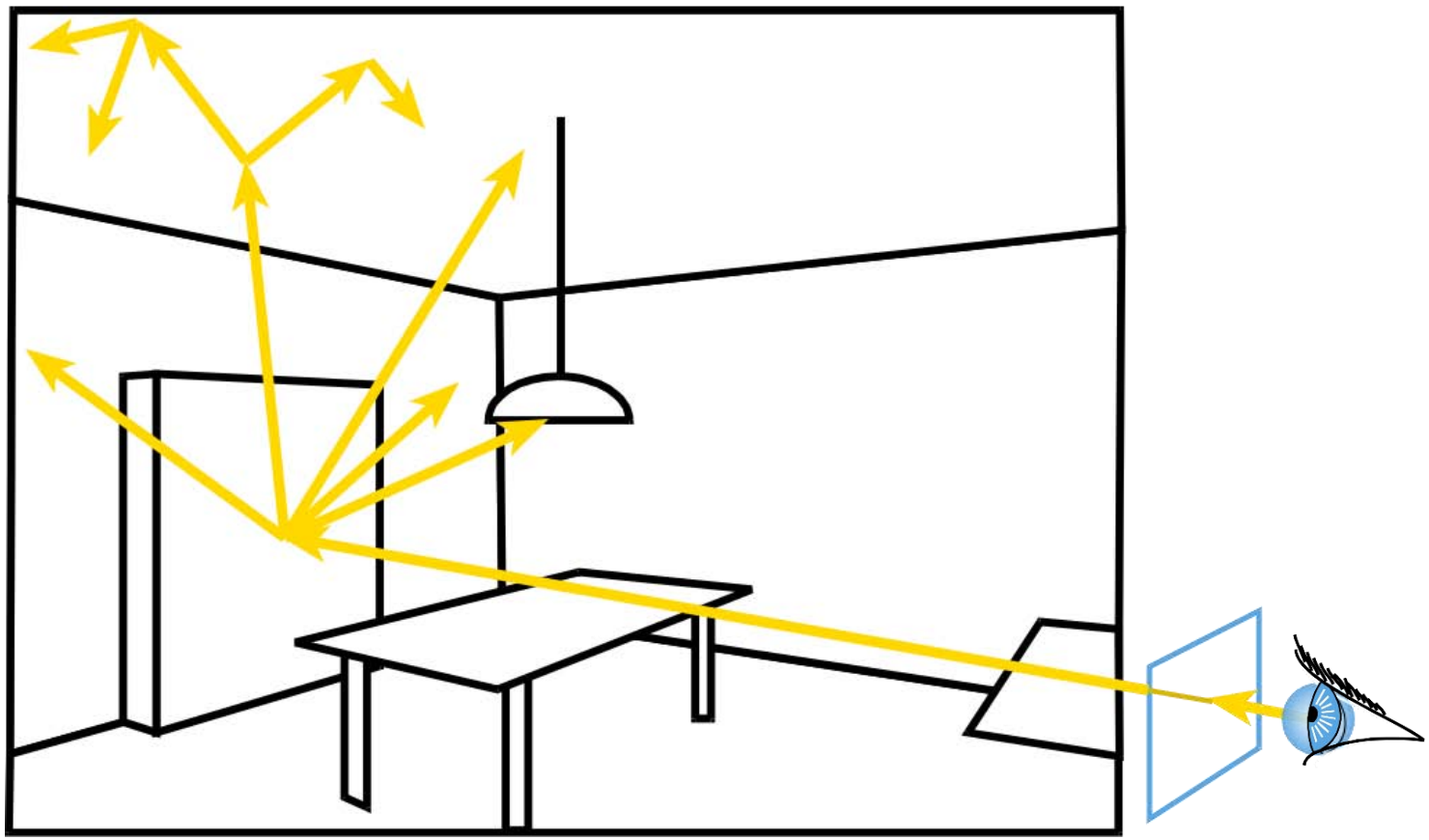
“Monte-Carlo Ray Tracing”

- Cast a ray from the eye through each pixel
- Cast random rays from the visible point
- Recurse



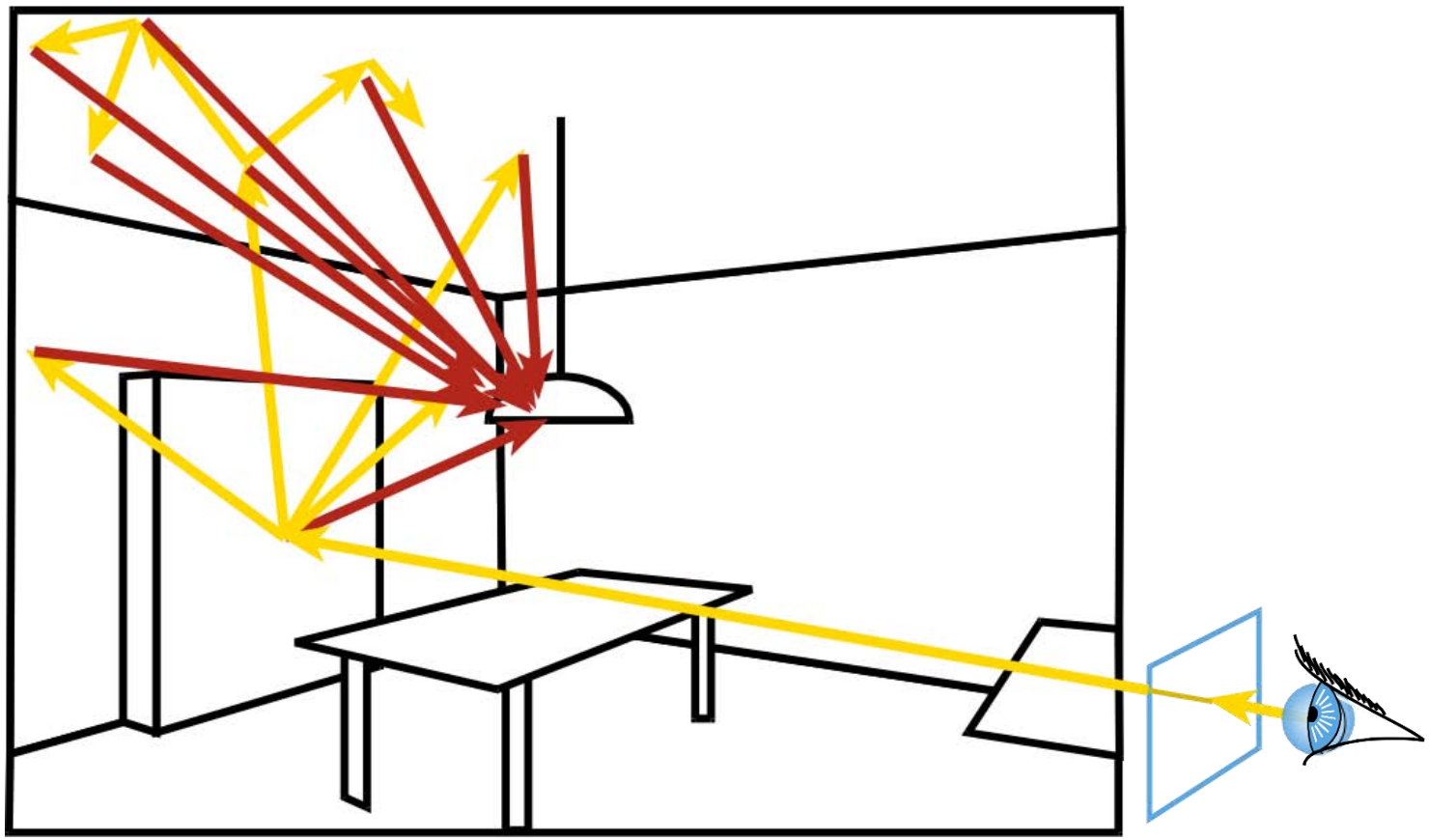
“Monte-Carlo Ray Tracing”

- Cast a ray from the eye through each pixel
- Cast random rays from the visible point
- Recurse

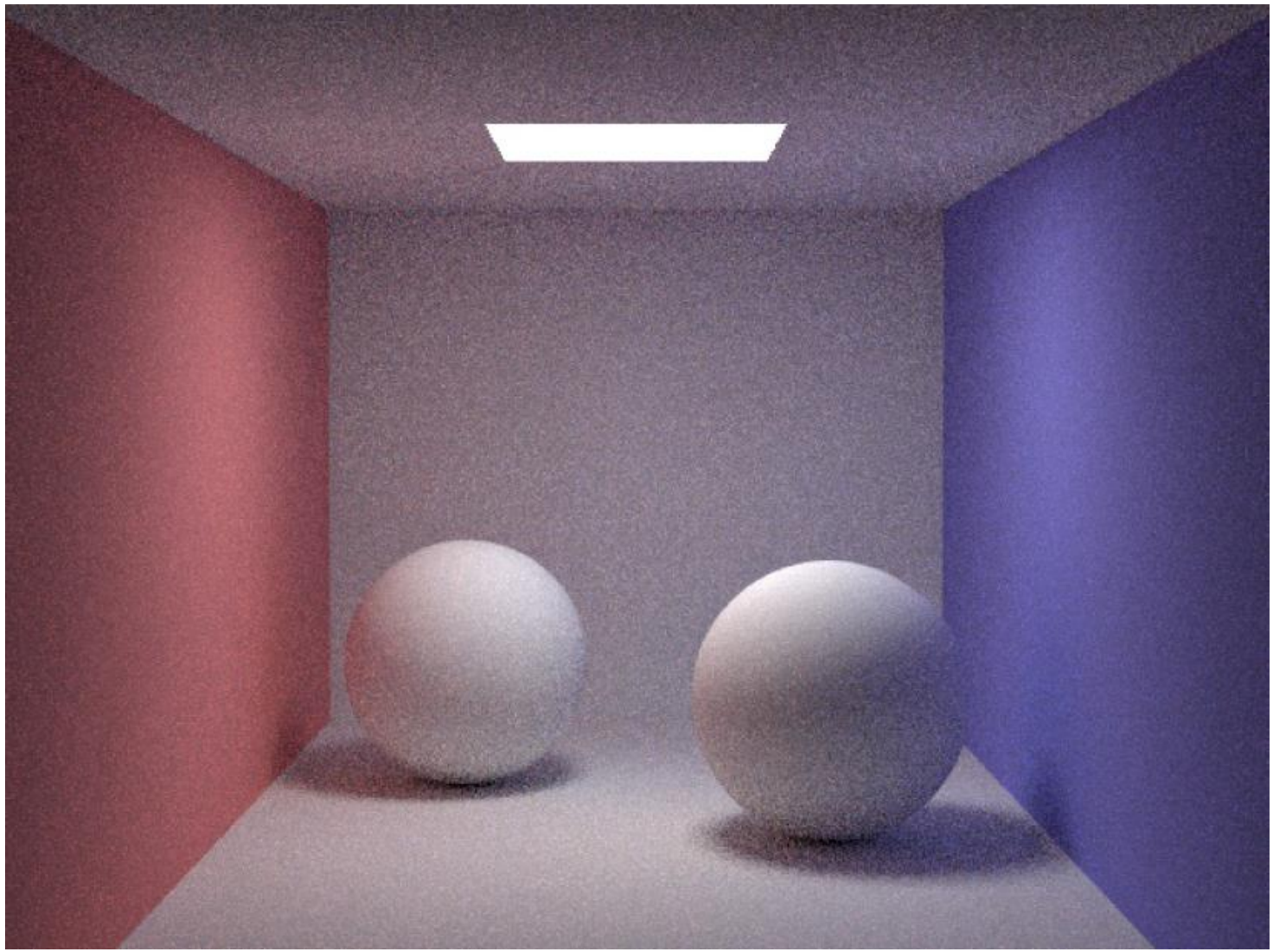


“Monte-Carlo Ray Tracing”

- Systematically sample light sources at each hit
  - Don’t just wait the rays will hit it by chance



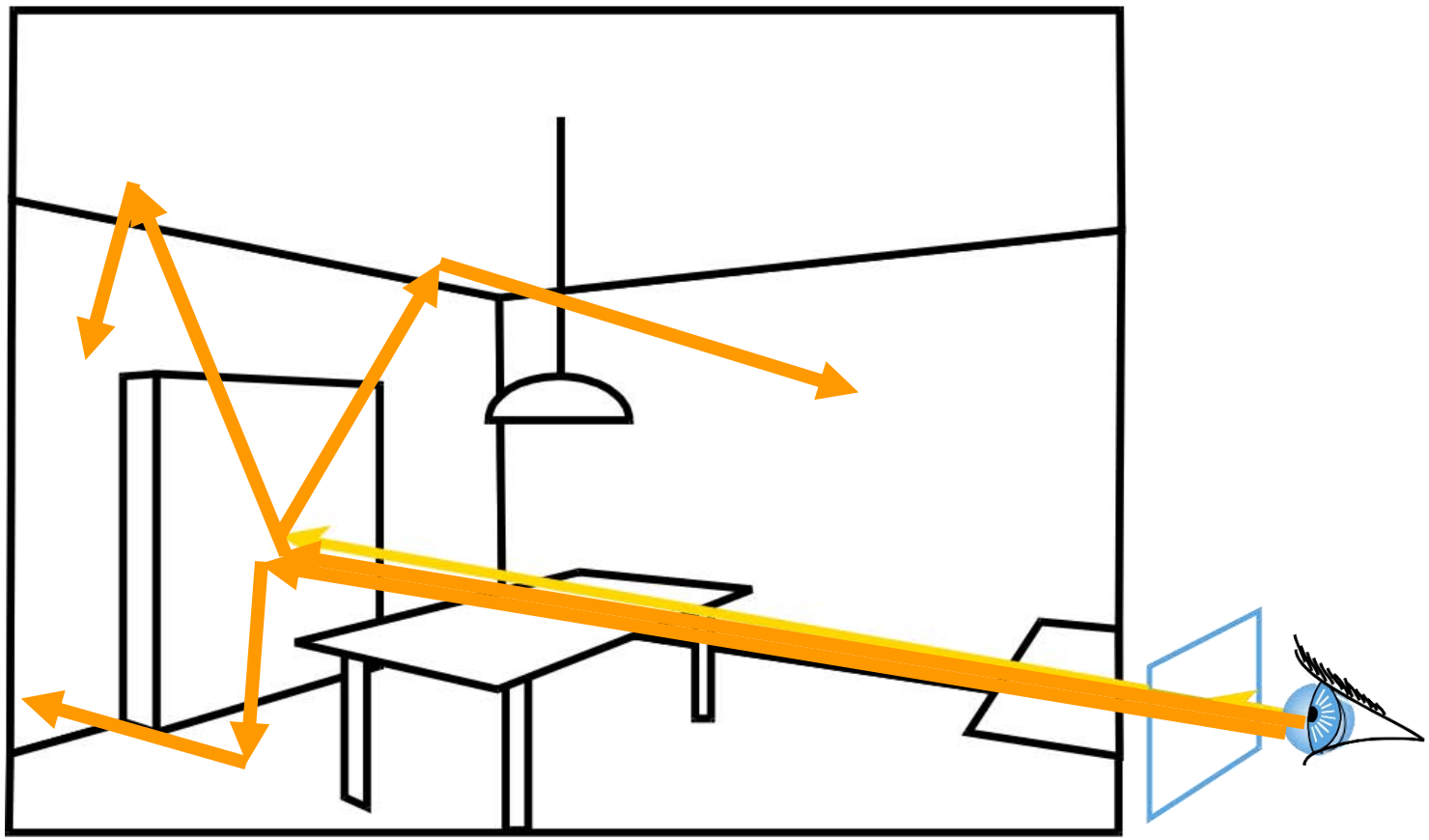
Results



Courtesy of Henrik Wann Jensen. Used with permission.

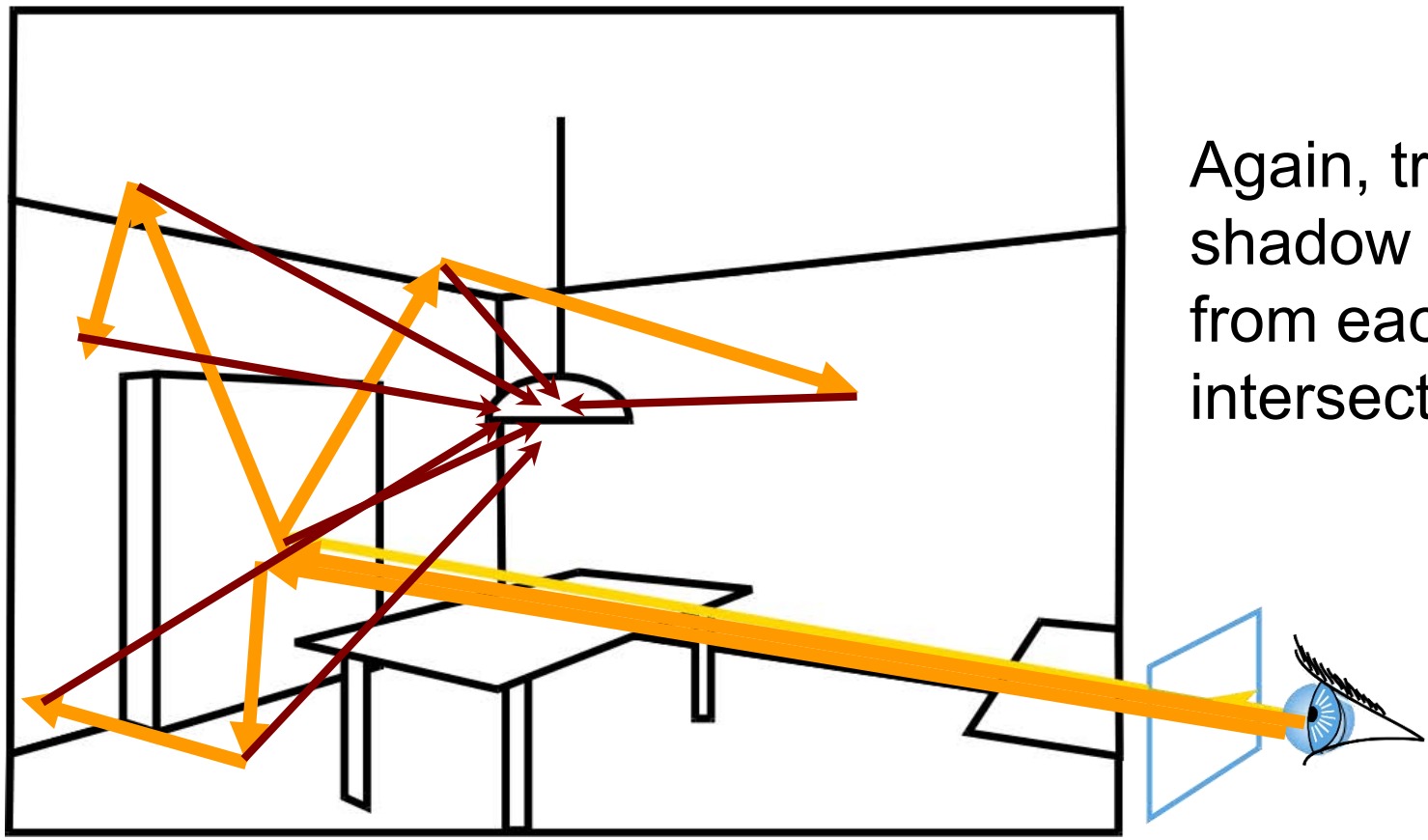
Monte Carlo Path Tracing

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



Monte Carlo Path Tracing

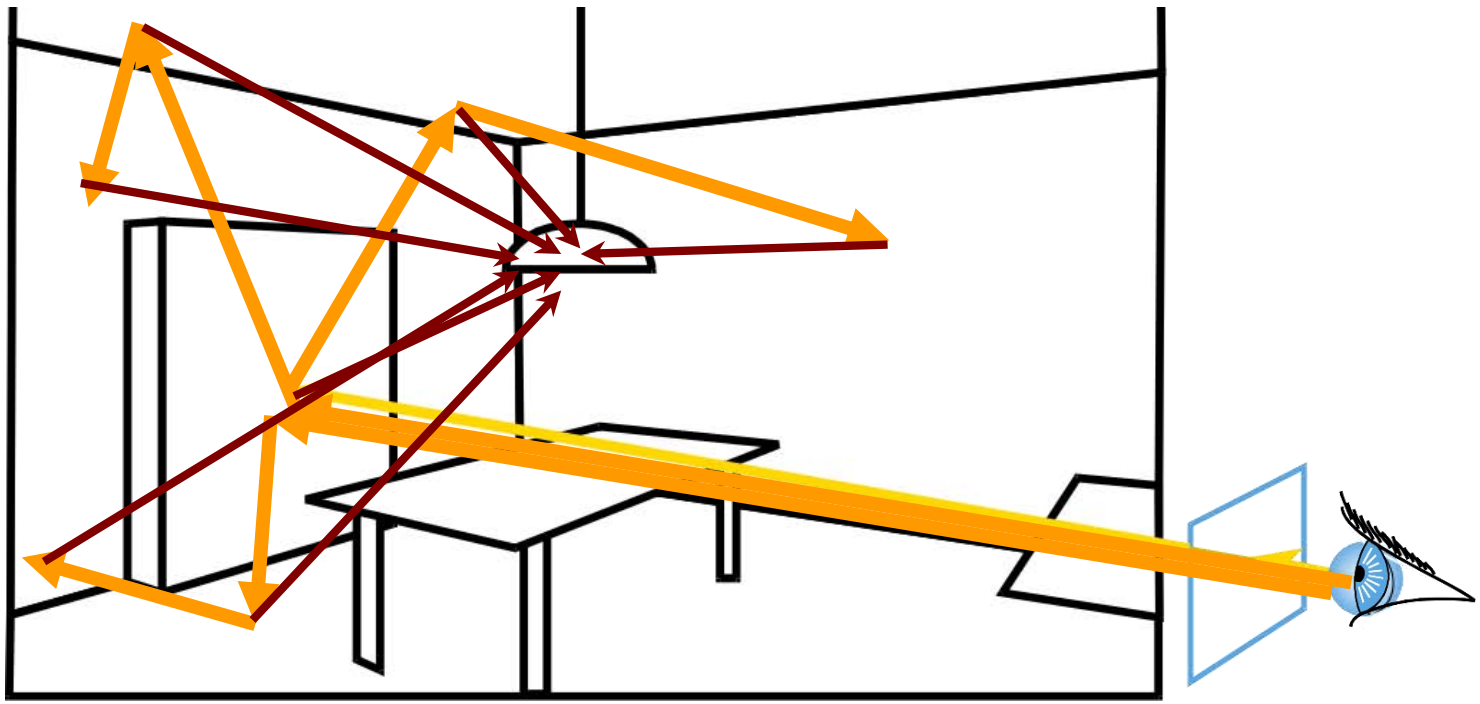
- Trace only one secondary ray per recursion
  - Otherwise number of rays explodes!
- But send many primary rays per pixel (antialiasing)



Again, trace shadow rays from each intersection

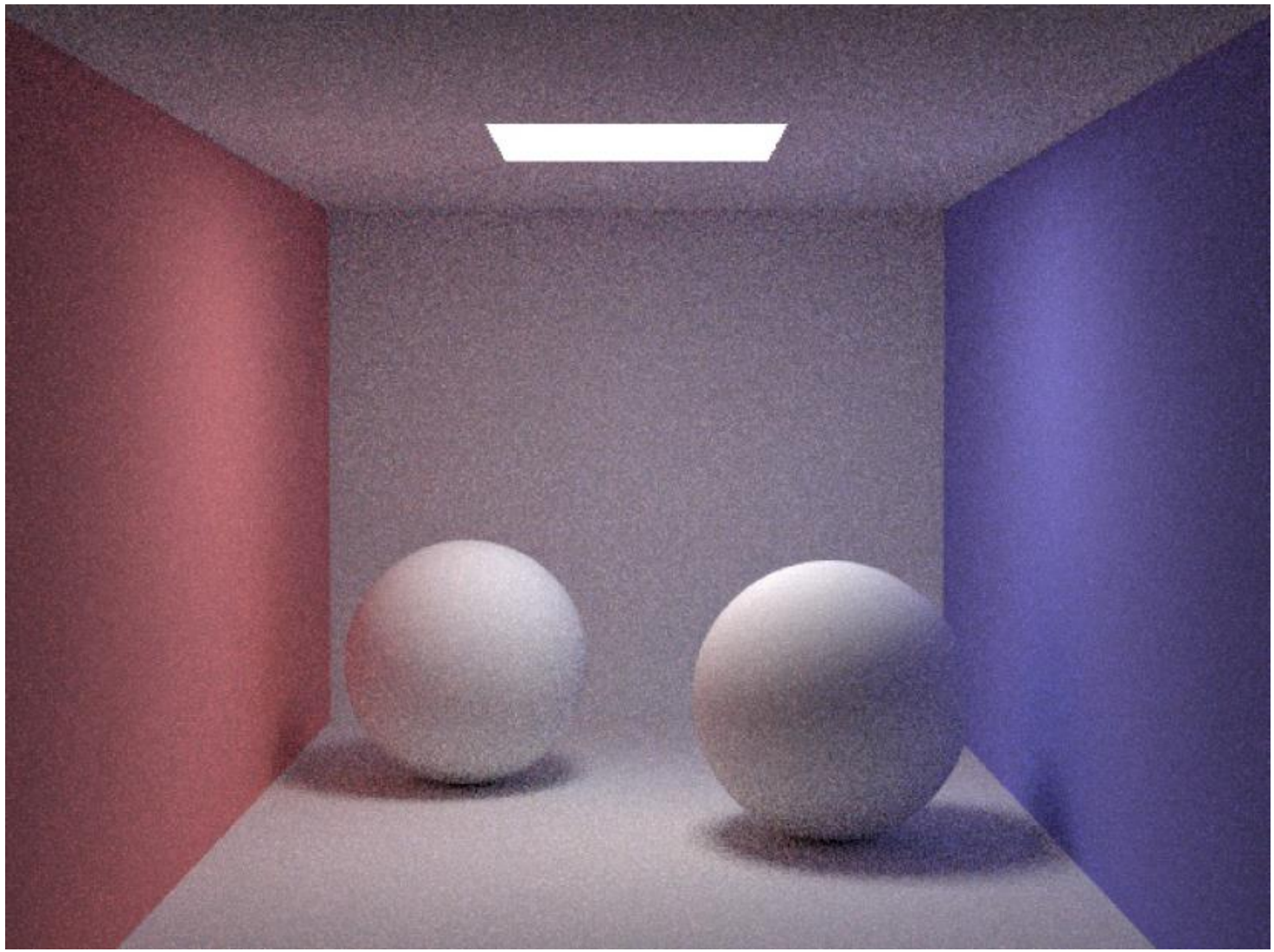
Monte Carlo Path Tracing

- We shoot one path from the eye at a time
  - Connect every surface point on the way to the light by a shadow ray
  - We are randomly sampling the space of all possible light paths between the source and the camera



Path Tracing Results

- 10 paths/pixel

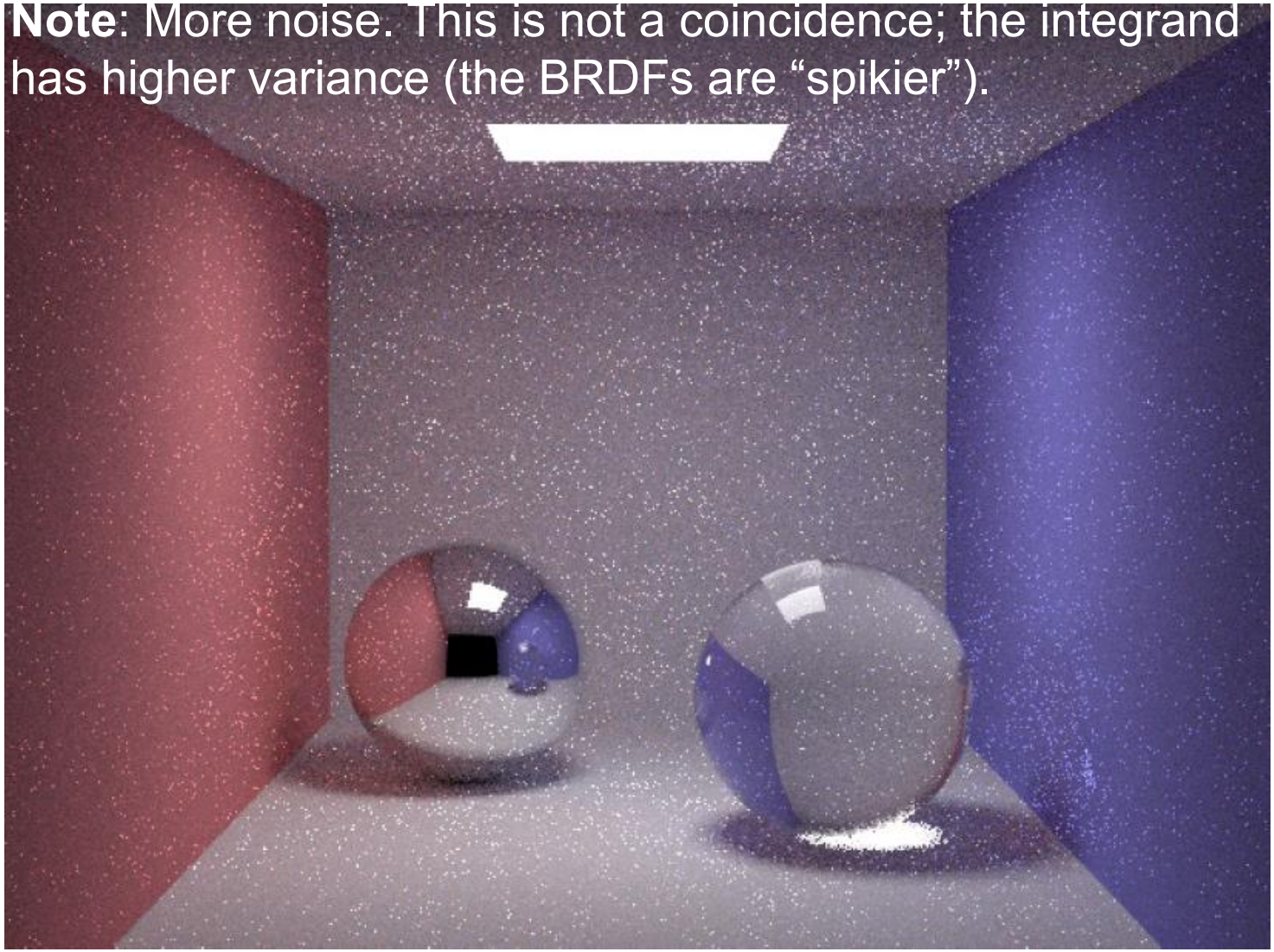


Henrik Wann Jensen

Courtesy of Henrik Wann Jensen. Used with permission.

Path Tracing Results: Glossy Scene

- 10 paths/pixel

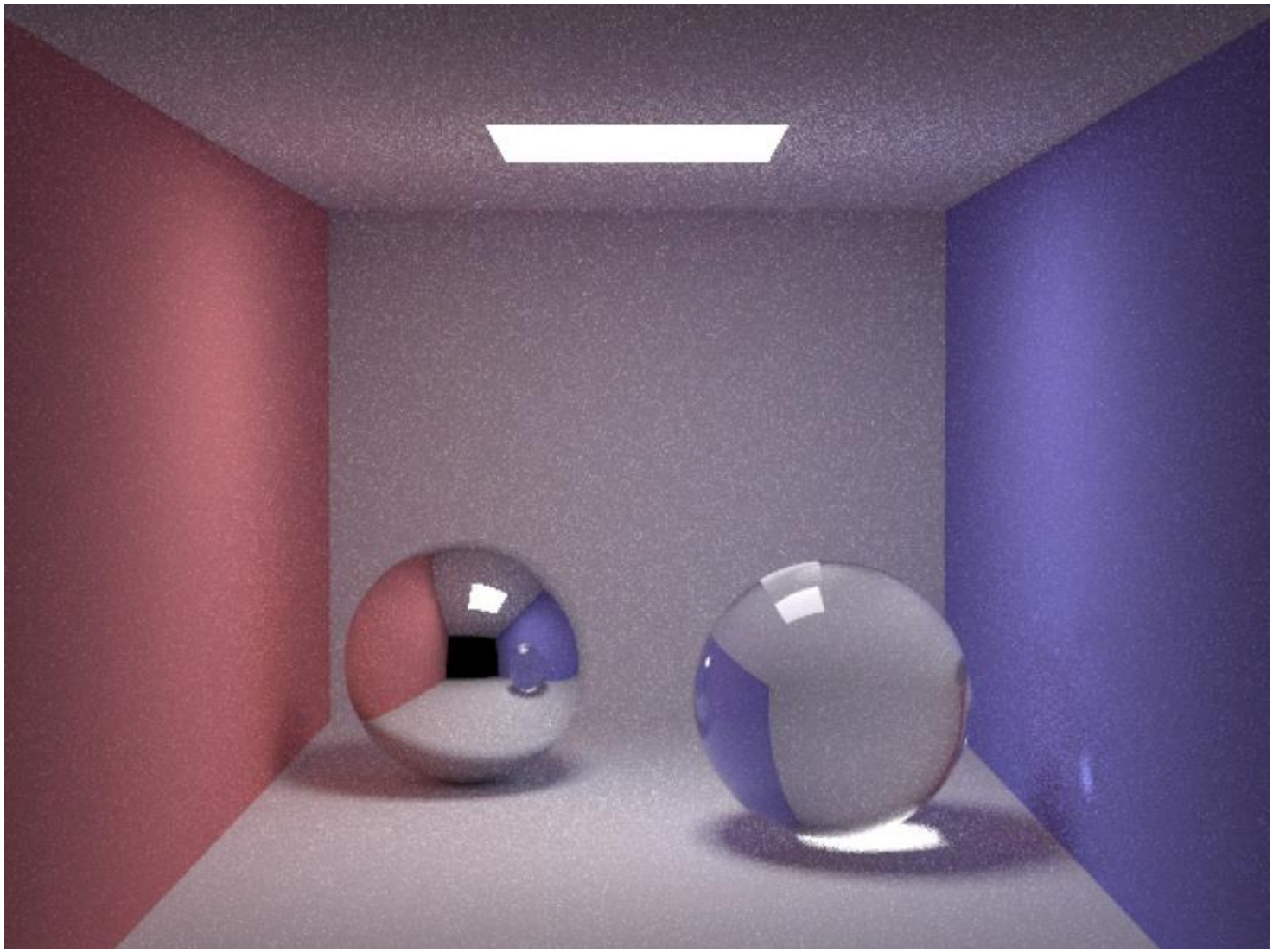


Henrik Wann Jensen

Courtesy of Henrik Wann Jensen. Used with permission.

Path Tracing Results: Glossy Scene

- 100 paths/pixel



Henrik Wann Jensen

Courtesy of Henrik Wann Jensen. Used with permission.

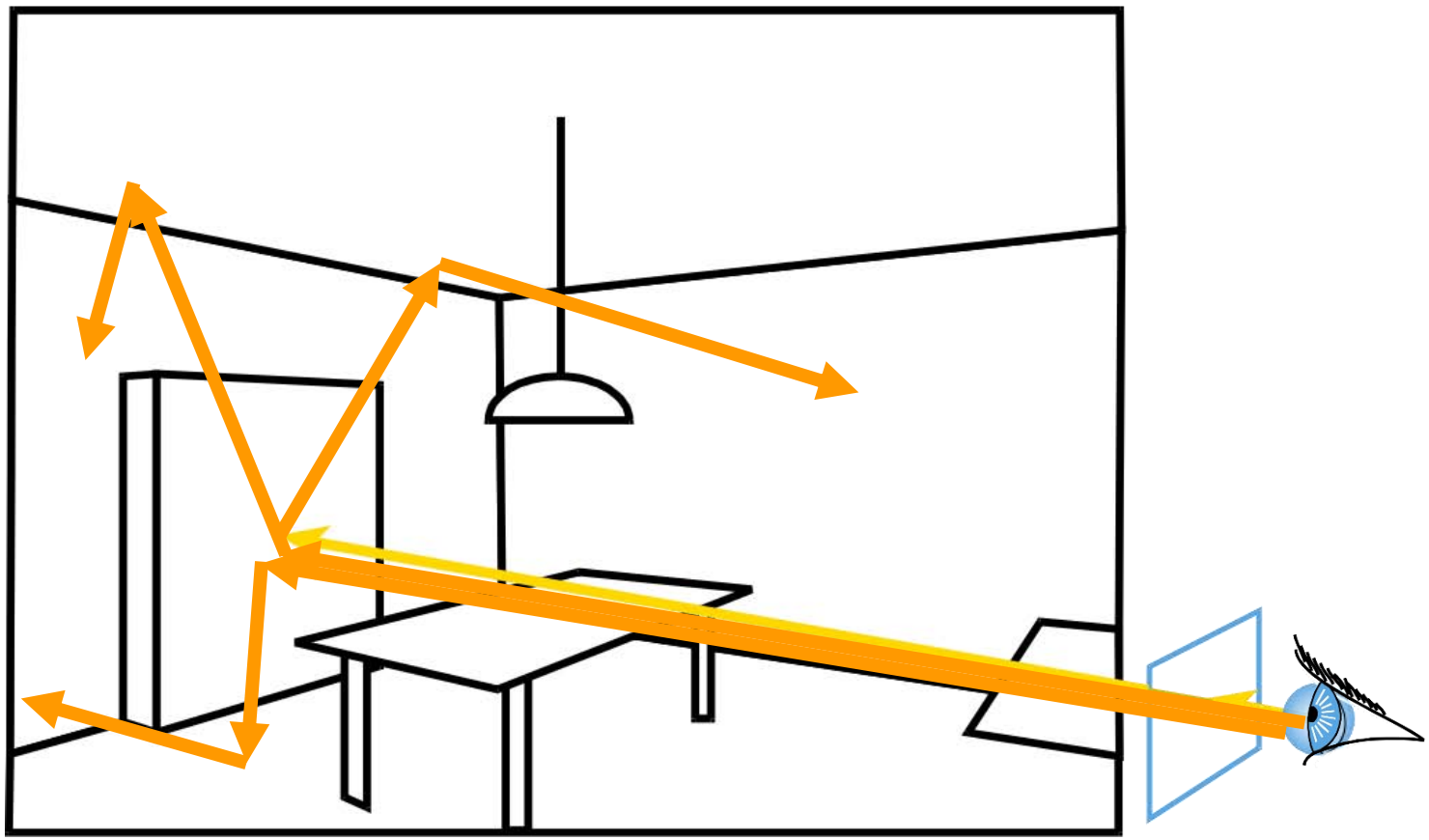
Demo

- http://madebyevan.com/webgl-path-tracing/

Image removed due to copyright restrictions. Please see the above link for further details.

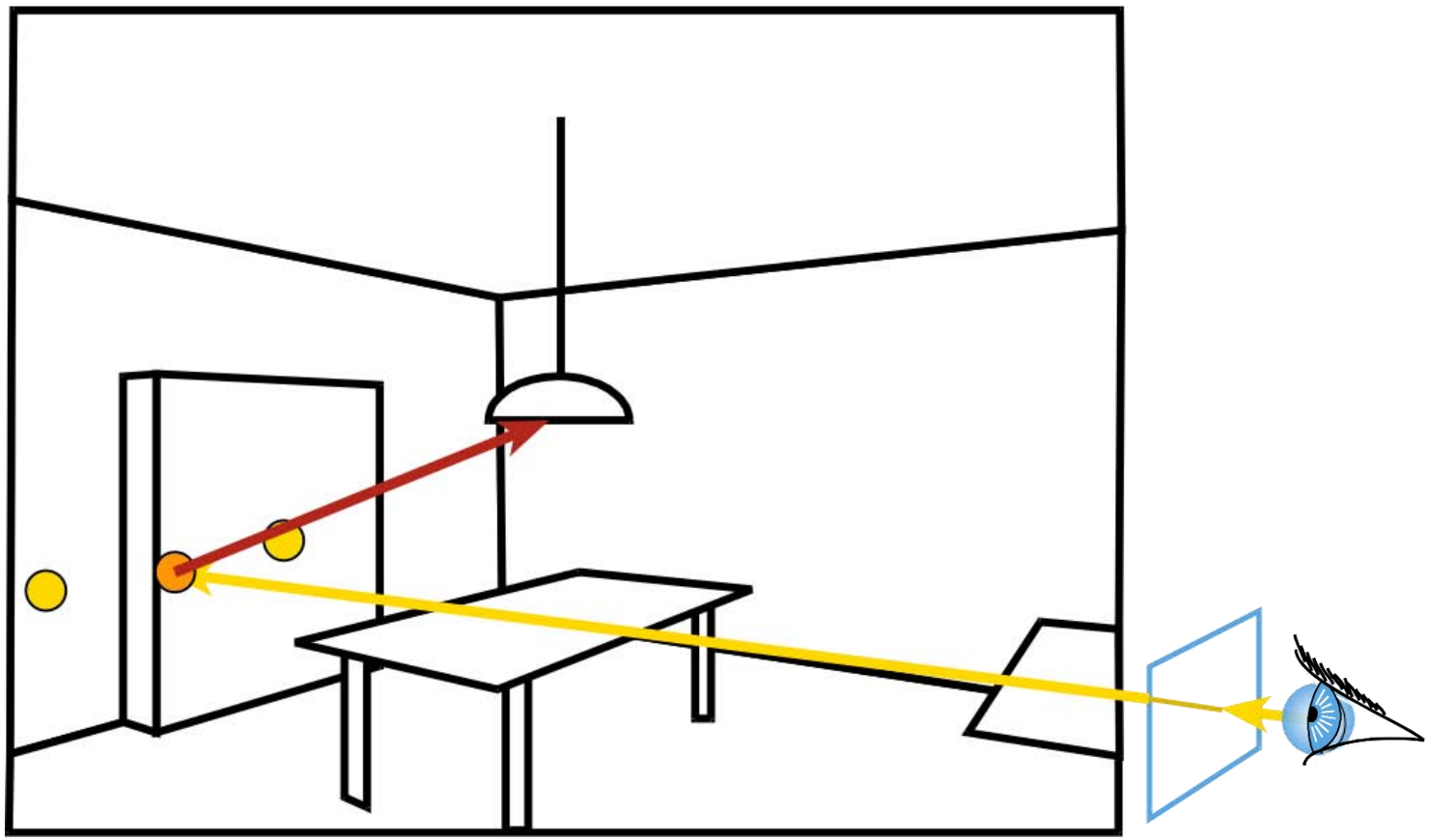
Path Tracing is costly

- Needs tons of rays per pixel!



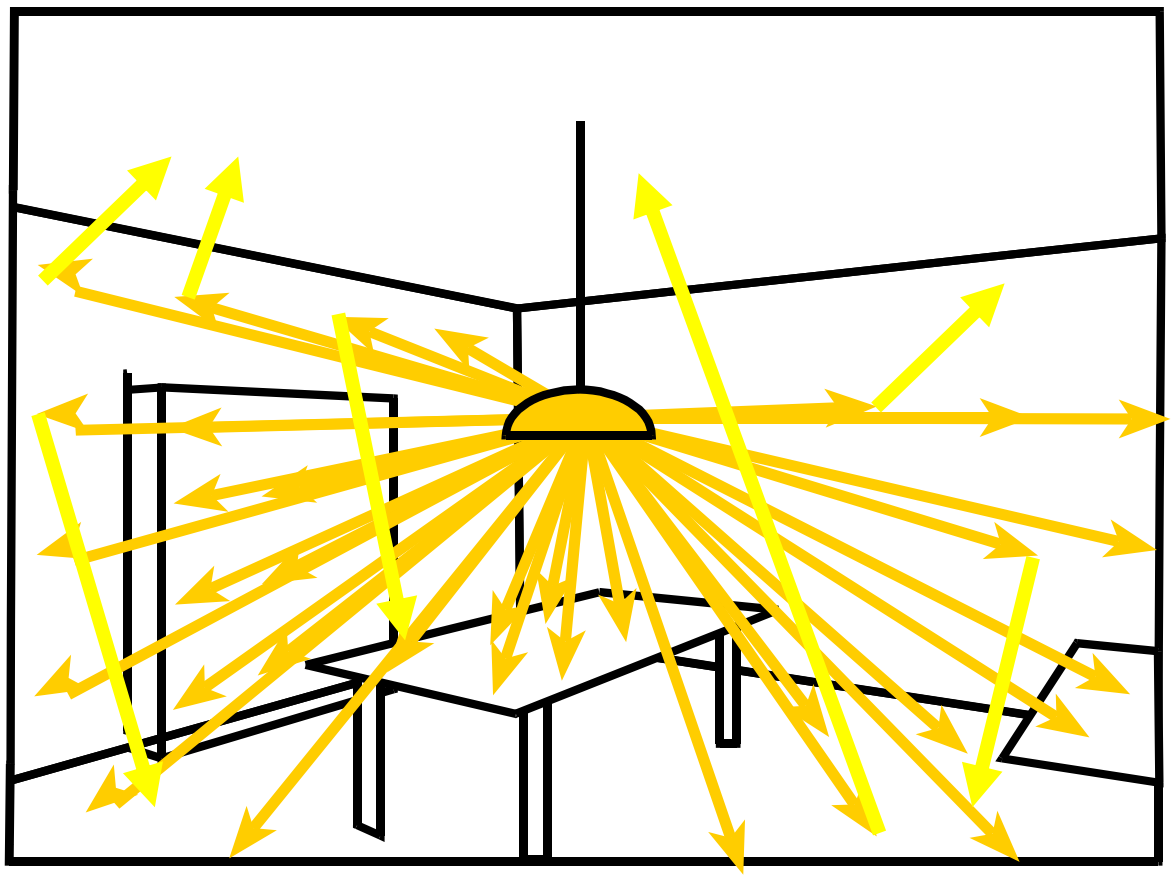
Irradiance Caching

- Store the indirect illumination
- Interpolate existing cached values
- But do full calculation for direct lighting



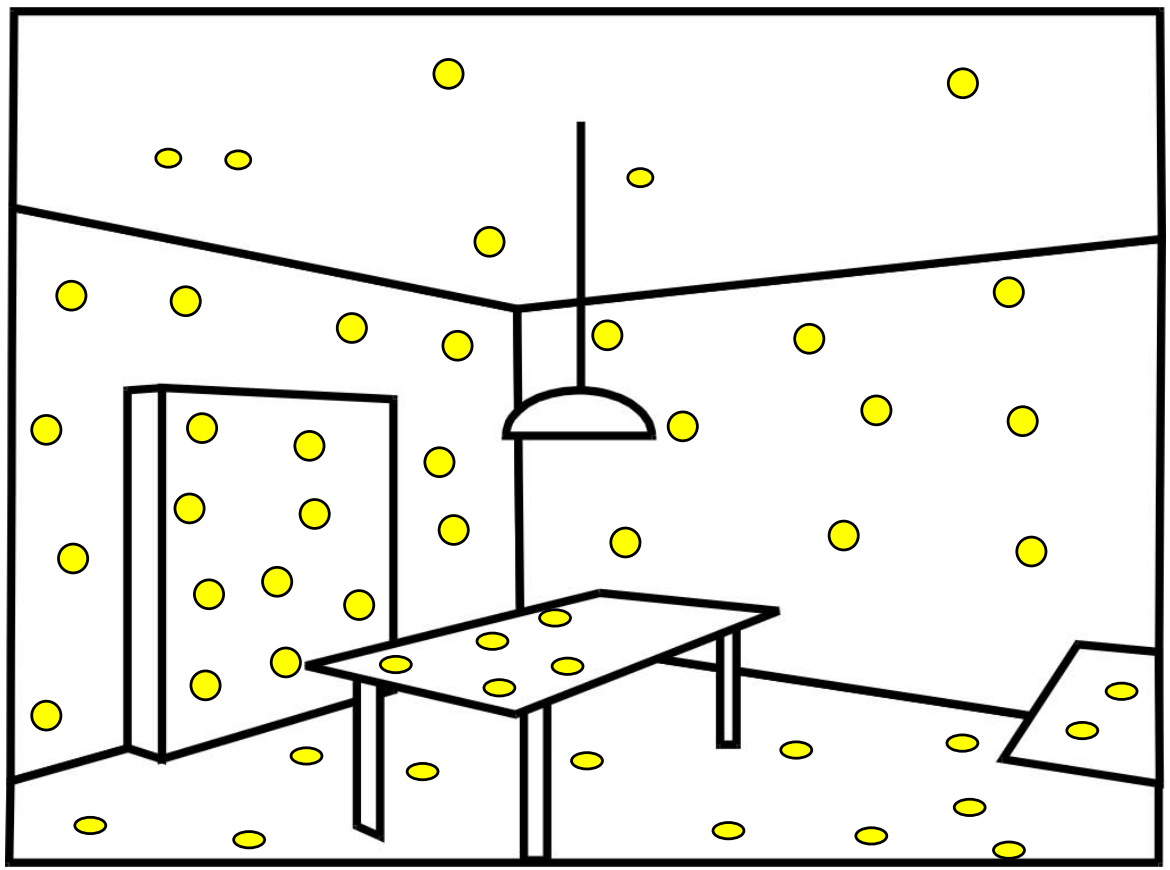
Photon Mapping

- Preprocess: cast rays from light sources, let them bounce around randomly in the scene
- Store “photons”



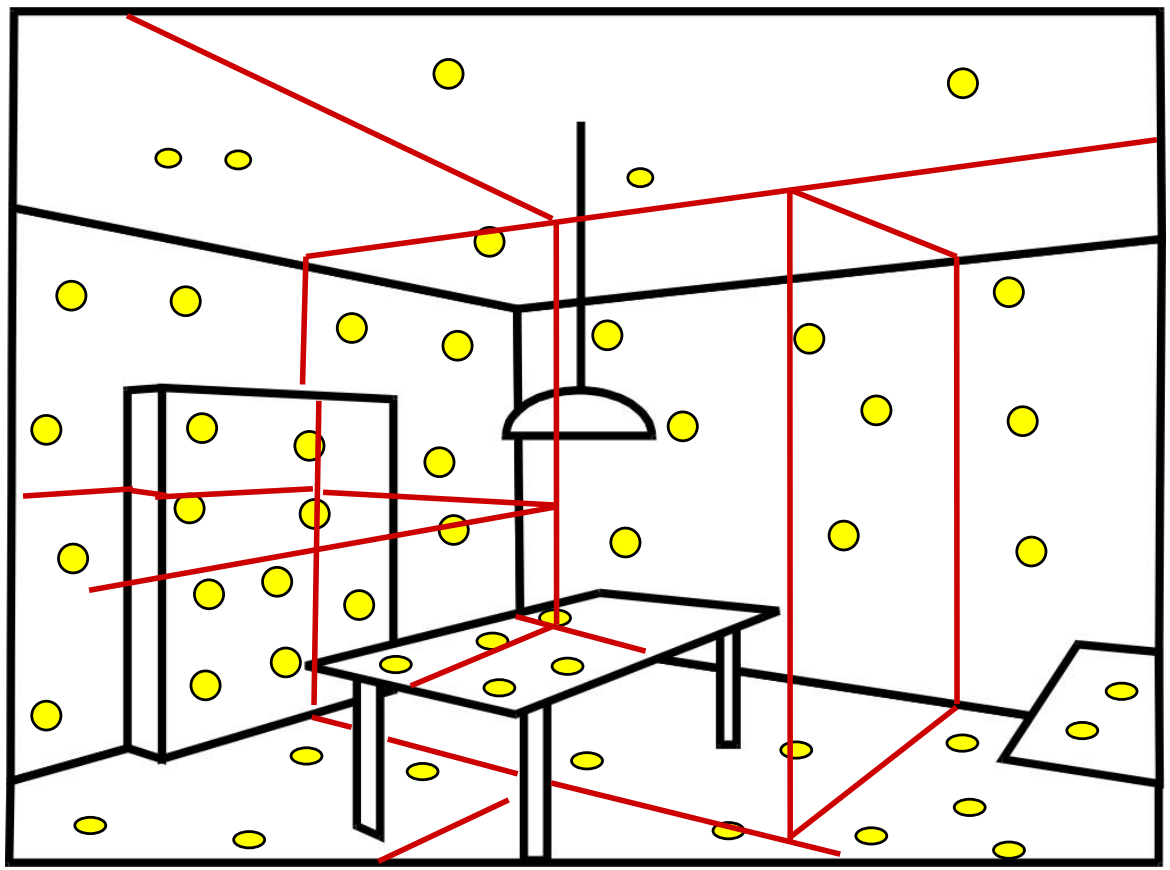
Photon Mapping

- Preprocess: cast rays from light sources
- Store photons (position + light power + incoming direction)



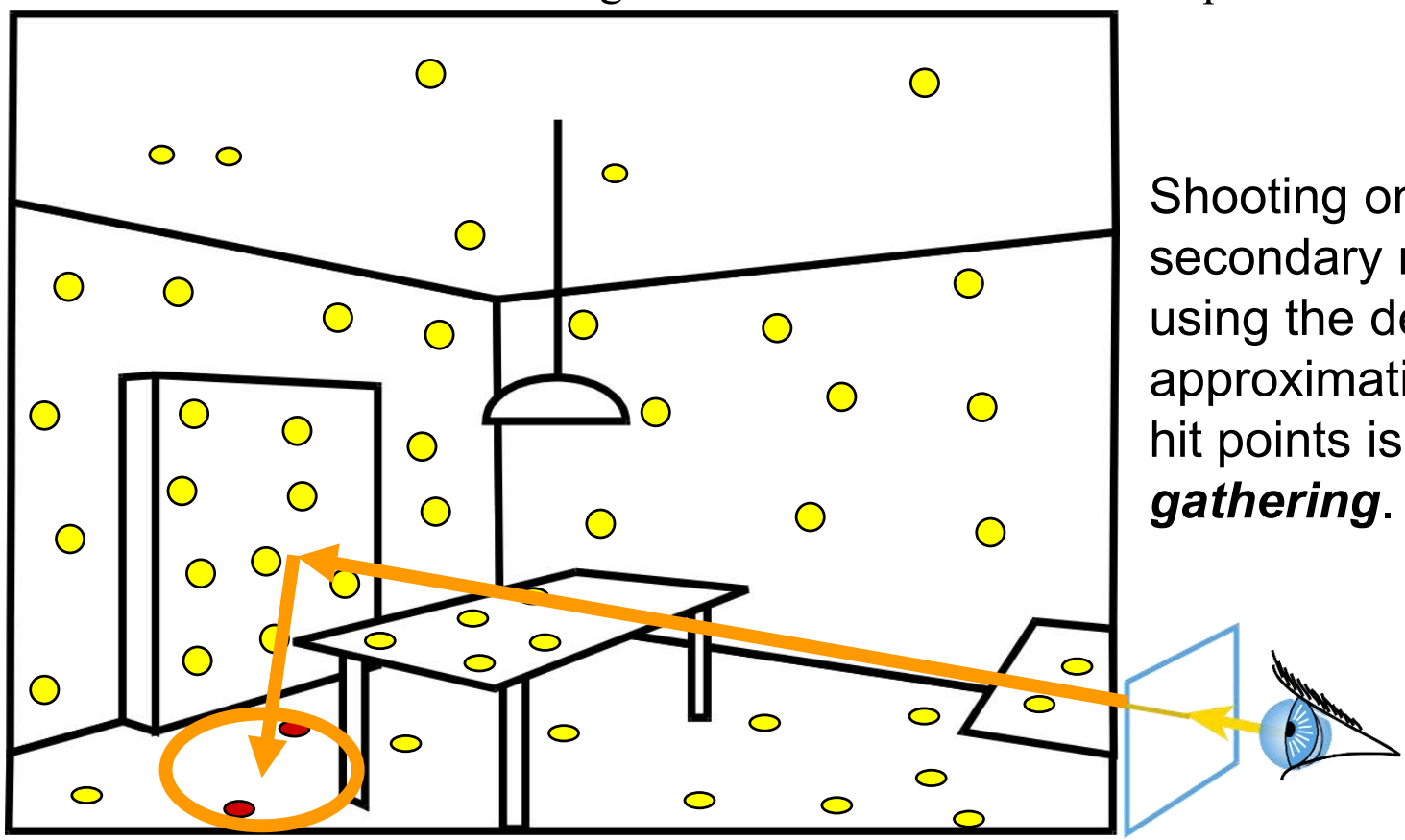
The Photon Map

- Efficiently store photons for fast access
- Use hierarchical spatial structure (kd-tree)



Photon Mapping - Rendering

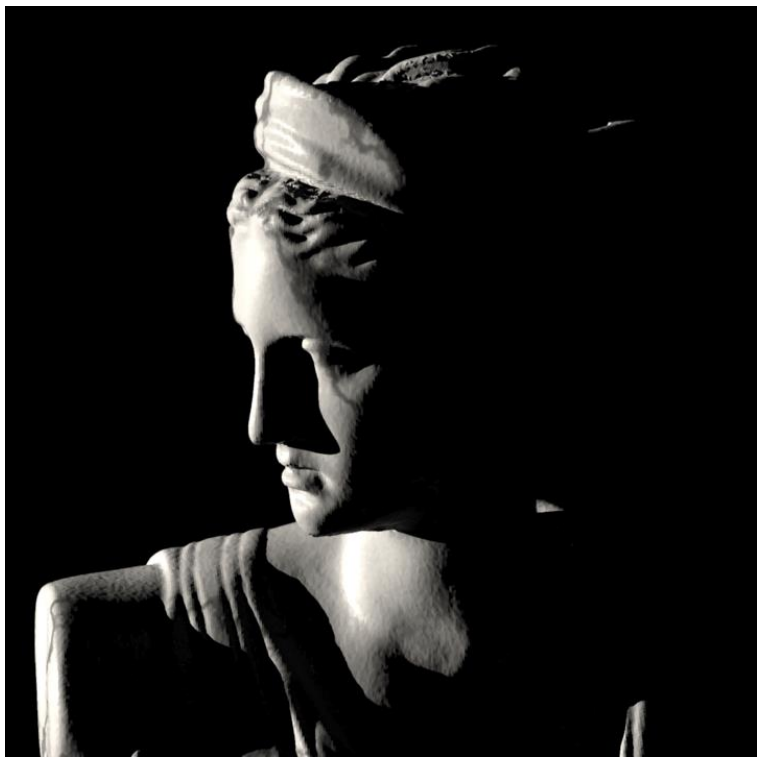
- Cast primary rays
- For secondary rays
  - reconstruct irradiance using adjacent stored photon
  - Take the k closest photons
- Combine with irradiance caching and a number of other techniques



Shooting one bounce of secondary rays and using the density approximation at those hit points is called ***final gathering***.

More Global Illumination Coolness

- Many materials exhibit *subsurface scattering*
  - Light doesn't just reflect off the surface
  - Light enters, scatters around, and exits at another point
  - Examples: Skin, marble, milk



Images: Jensen et al.

Courtesy of Henrik Wann Jensen. Used with permission.

That Was Just the Beginning

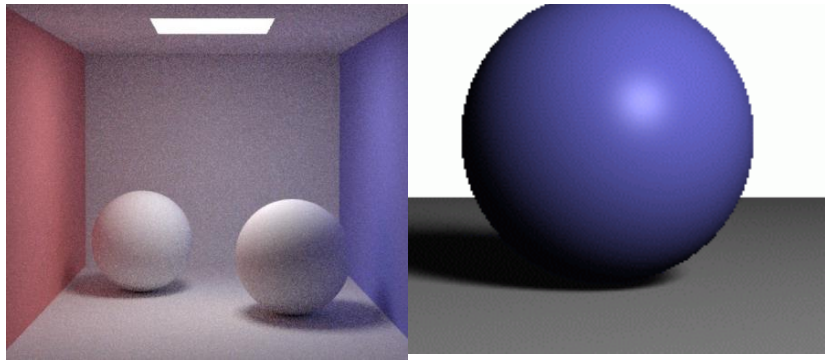
- Tons and tons of other Monte Carlo techniques
  - Bidirectional Path Tracing
    - Shoot random paths not just from camera but also light, connect the path vertices by shadow rays
  - Metropolis Light Transport
- And Finite Element Methods
  - Use basis functions instead of random sampling
  - Radiosity (with hierarchies & wavelets)
  - Precomputed Radiance Transfer

- This would warrant a class of its own!

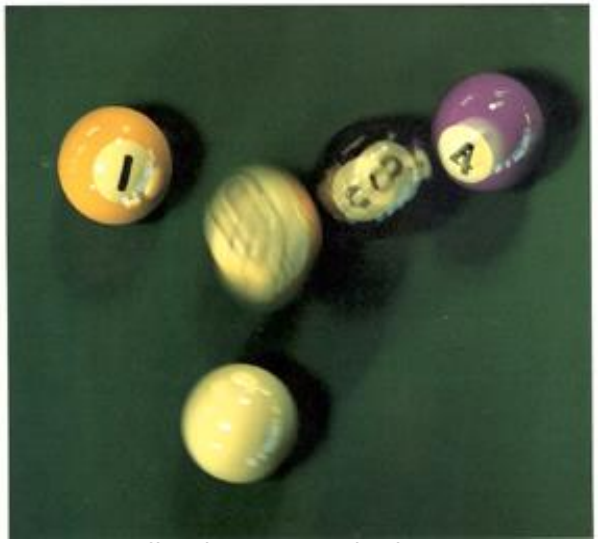
What Else Can We Integrate?

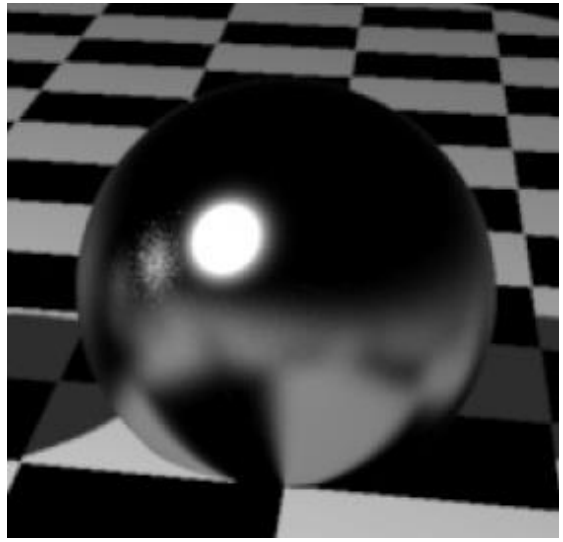
- Pixel: antialiasing
- Light sources: Soft shadows
- Lens: Depth of field
- Time: Motion blur
- BRDF: glossy reflection
- (Hemisphere: indirect lighting)

$$\int\int\int\int\int L(x, y, t, u, v)dx\,dy\,dt\,du\,dv$$



Courtesy of Henrik Wann Jensen. Used with permission.

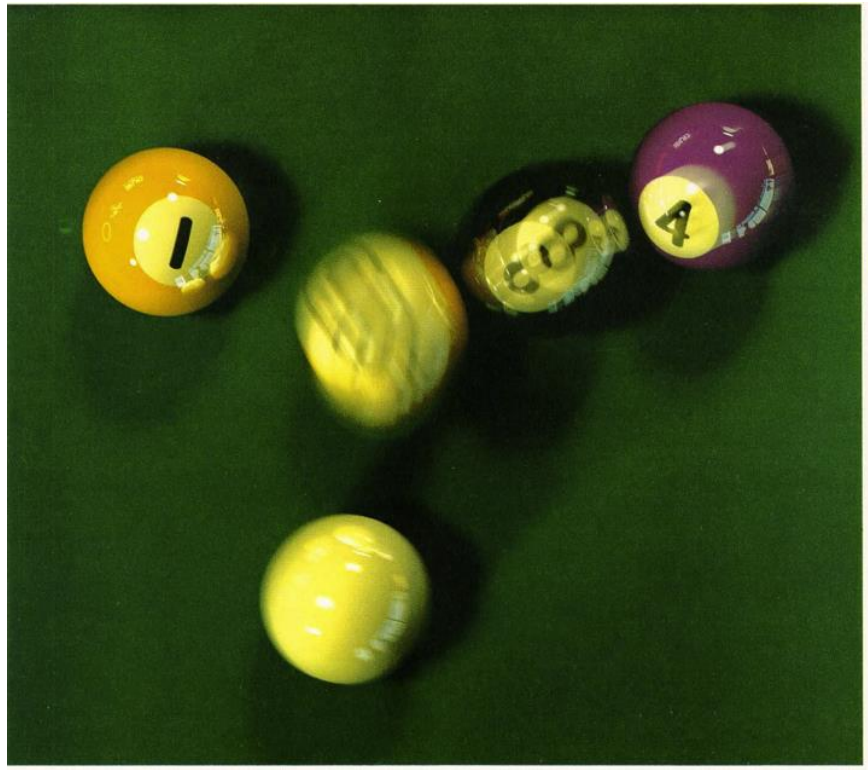




Domains of Integration

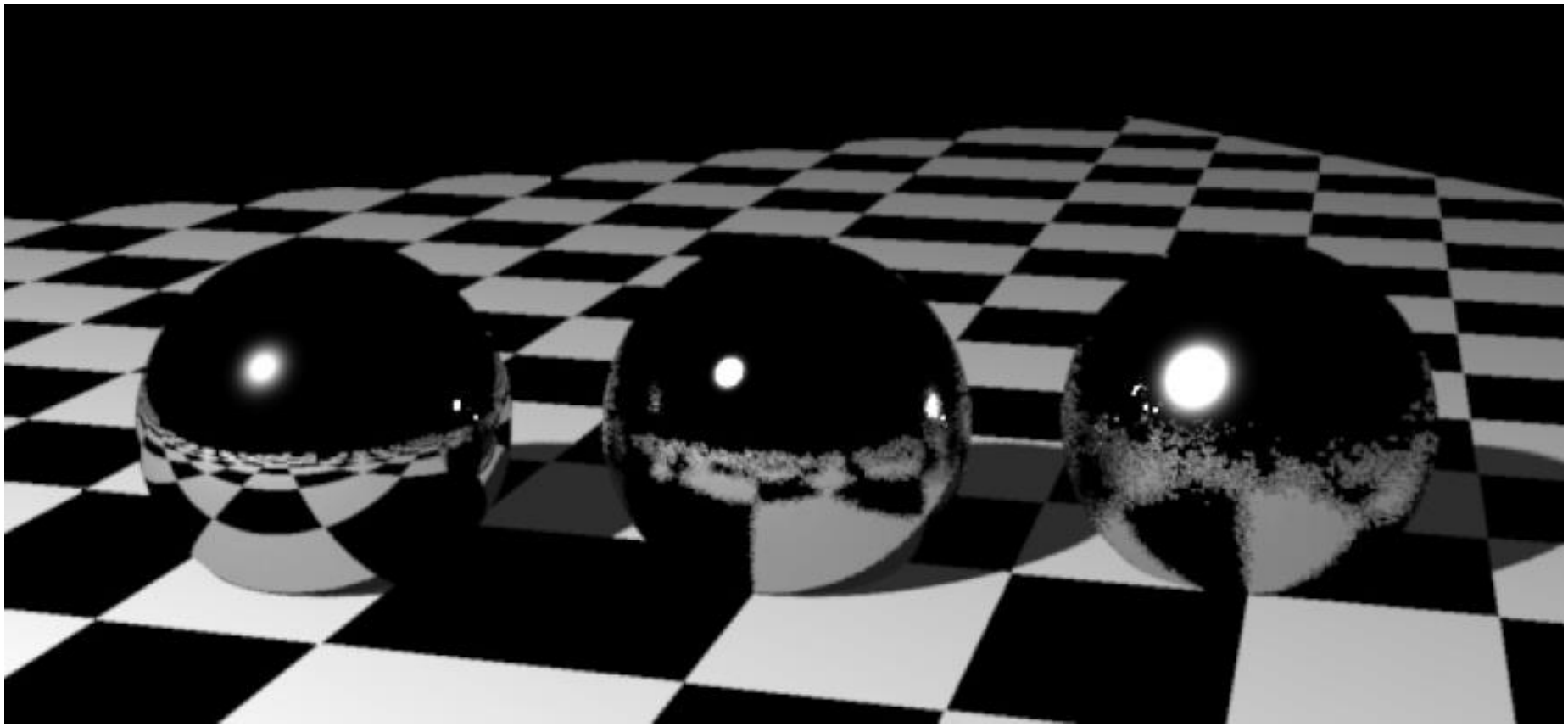
- Pixel, lens (Euclidean 2D domain)
  - Antialiasing filters, depth of field
- Time (1D)
  - Motion blur
- Hemisphere
  - Indirect lighting
- Light source
  - Soft shadows

Famous motion blur image
from Cook et al. 1984



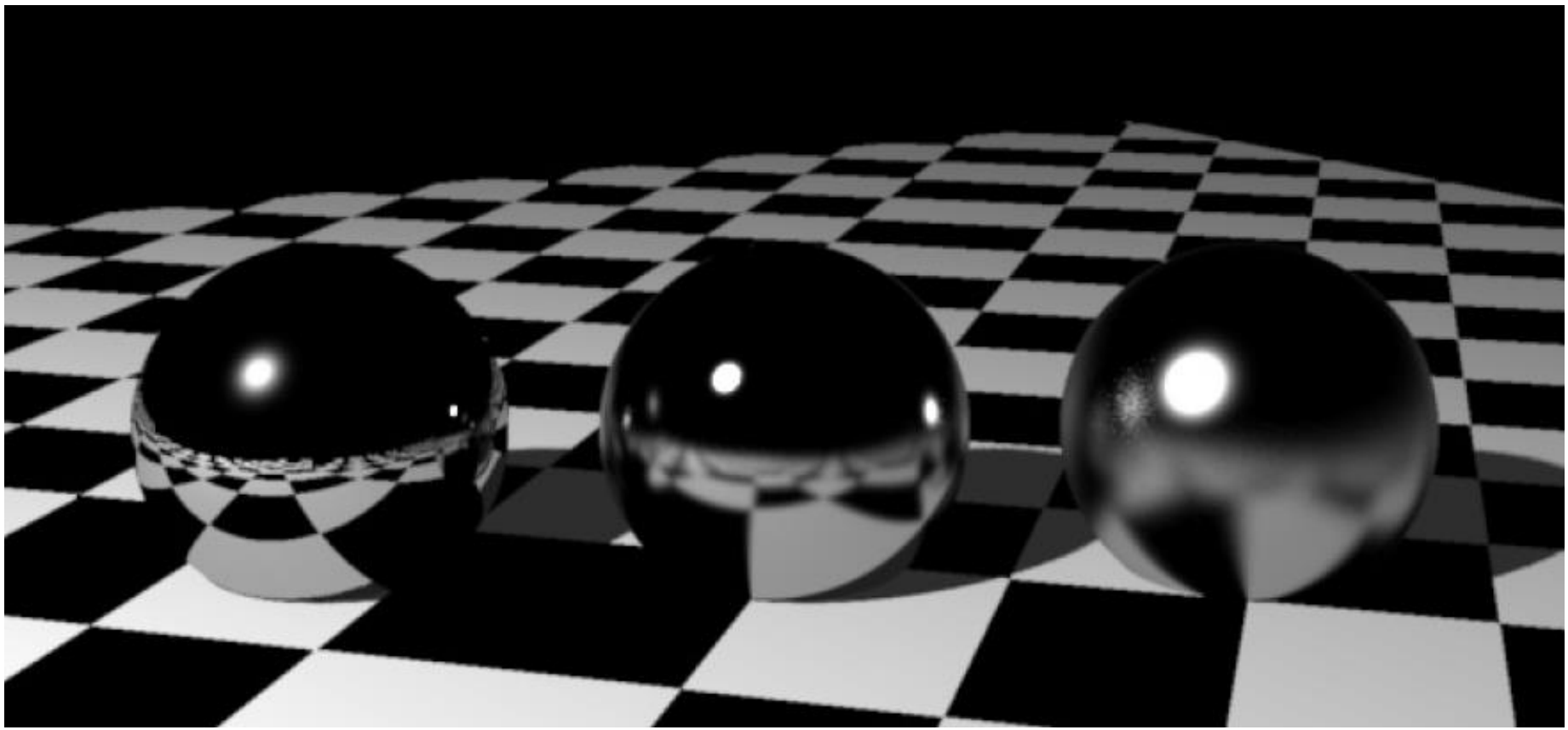
Motivational Eye Candy

- Rendering glossy reflections
- Random reflection rays around mirror direction
  - 1 sample per pixel



Motivational Eye Candy

- Rendering glossy reflections
- Random reflection rays around mirror direction
  - 256 samples per pixel



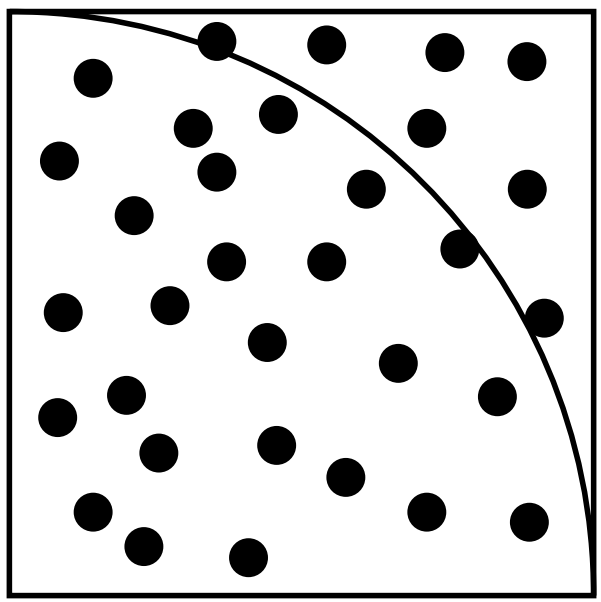
Monte Carlo Integration

$$\int_S f(x)\,\mathrm{d}x \approx \frac{\mathrm{Vol}(S)}{N}\sum_{i=1}^{N} f(x_i)$$

- S is the integration domain
  - Vol(S) is the volume (measure) of S
- $\{x_i\}$ are independent uniform random points in S
- The integral is the average of *f* times the volume of S
- Variance is proportional to 1/N
  - Avg. error is proportional 1/sqrt(N)
  - To halve error, need 4x samples

Monte Carlo Computation of π

- The probability is π /4
- Count the inside ratio n = # inside / total # trials
- $\pi \approx n * 4$
- The error depends on the number or trials



Demo

```
def piMC(n):
   success = 0
   for i in range(n):                    x=random.random()
                  y=random.random()
                  if x*x+y*y<1: success = success+1        return
4.0*float(success)/float(n)
```

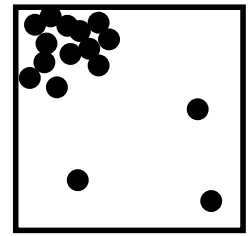
Questions?

- Images by Veach and Guibas, SIGGRAPH 95

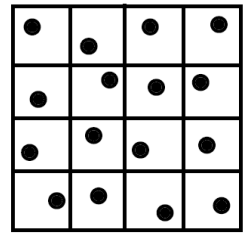
Naïve sampling strategy

Optimal sampling strategy

Stratified Sampling



- With uniform sampling, we can get unlucky
  - E.g. all samples clump in a corner
  - If we don't know anything of the integrand, we want a relatively uniform sampling
    - Not regular, though, because of aliasing!



- To prevent clumping, subdivide domain Ω into non-overlapping regions Ω_i
  - Each region is called a *stratum*
- Take one random sample per Ω_i

For Further Information...

- 6.839!
- Eric Veach's PhD dissertation
  http://graphics.stanford.edu/papers/veach_thesis/

- Physically Based Rendering
  by Matt Pharr, Greg Humphreys

TIEA311 - Today in Jyväskylä

- If you got interested, you may want to check the whole “Lecture 18” from the original course material.
- Much more about integration on our courses about numerics. (Some years of math studies is a prerequisite)
- More about stratified sampling on our courses about data mining (exactly the same methods used for some data mining / machine learning methods)
- Perfect goals for thesis projects! Our faculty has a long history in related numerical methods and theory, so staff resources for thesis supervision should be quite easy to find.