Anti-aliasing and Monte Carlo Path Tracing

Brian Curless
CSE 457
Autumn 2017

Reading

Required:

- Marschner and Shirley, Section 13.4 (online handout)

Further reading:

- Pharr, Jakob, and Humphreys, Physically Based Ray Tracing: From Theory to Implementation, Chapter 13
- A. Glassner. An Introduction to Ray Tracing. Academic Press, 1989.
- Robert L. Cook, Thomas Porter, Loren Carpenter. “Distributed Ray Tracing.” Computer Graphics (Proceedings of SIGGRAPH 84). *18 (3)*. pp. 137-145. 1984.
- James T. Kajiya. “The Rendering Equation.” Computer Graphics (Proceedings of SIGGRAPH 86). *20 (4)*. pp. 143-150. 1986.

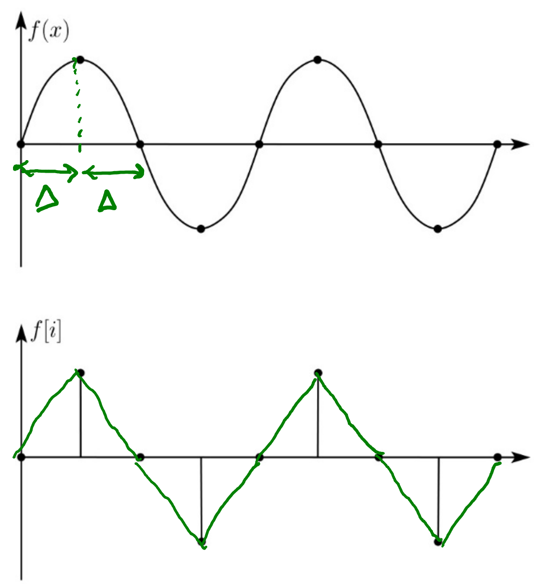
Aliasing

Ray tracing is a form of sampling and can suffer from annoying visual artifacts...

Consider a continuous function $f(x)$. Now sample it at intervals Δ to give $f[i] = \text{quantize}[f(i\,\Delta)]$.

Q: How well does $f[i]$ approximate $f(x)$?

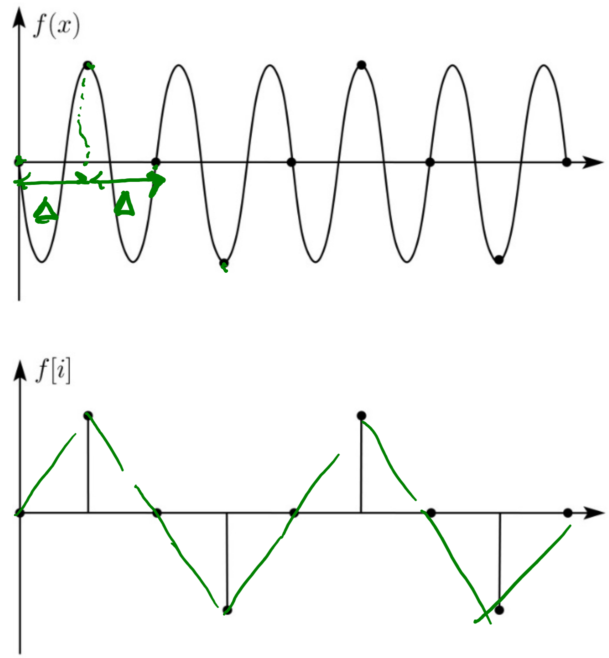
Consider sampling a sinusoid:



In this case, the sinusoid is reasonably well approximated by the samples.

Aliasing (con't)

Now consider sampling a higher frequency sinusoid

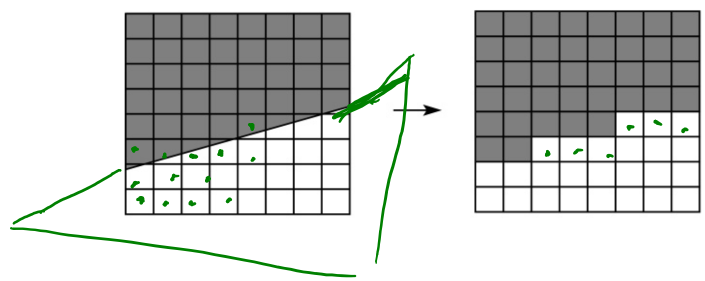


We get the exact same samples, so we seem to be approximating the first lower frequency sinusoid again.

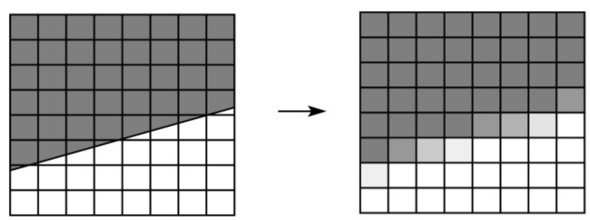
We say that, after sampling, the higher frequency sinusoid has taken on a new "alias", i.e., changed its identity to be a lower frequency sinusoid.

Aliasing and anti-aliasing in rendering

One of the most common rendering artifacts is the "jaggies". Consider rendering a white polygon against a black background:



We would instead like to get a smoother transition:

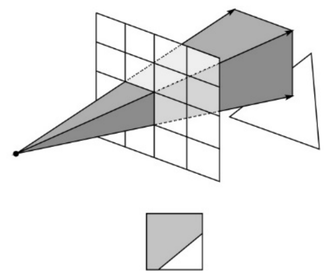


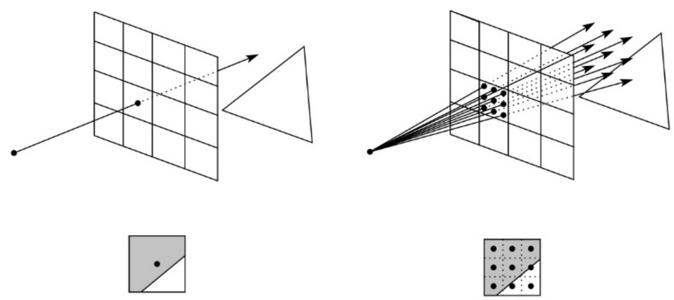
Anti-aliasing is the process of removing high frequencies *before* they cause aliasing.

In a renderer, computing the average color within a pixel is a good way to anti-alias. How exactly do we compute the average color?

Antialiasing in a ray tracer

We would like to compute the average intensity in the neighborhood of each pixel.





When casting one ray per pixel, we are likely to have aliasing artifacts.

To improve matters, we can cast more than one ray per pixel and average the result.

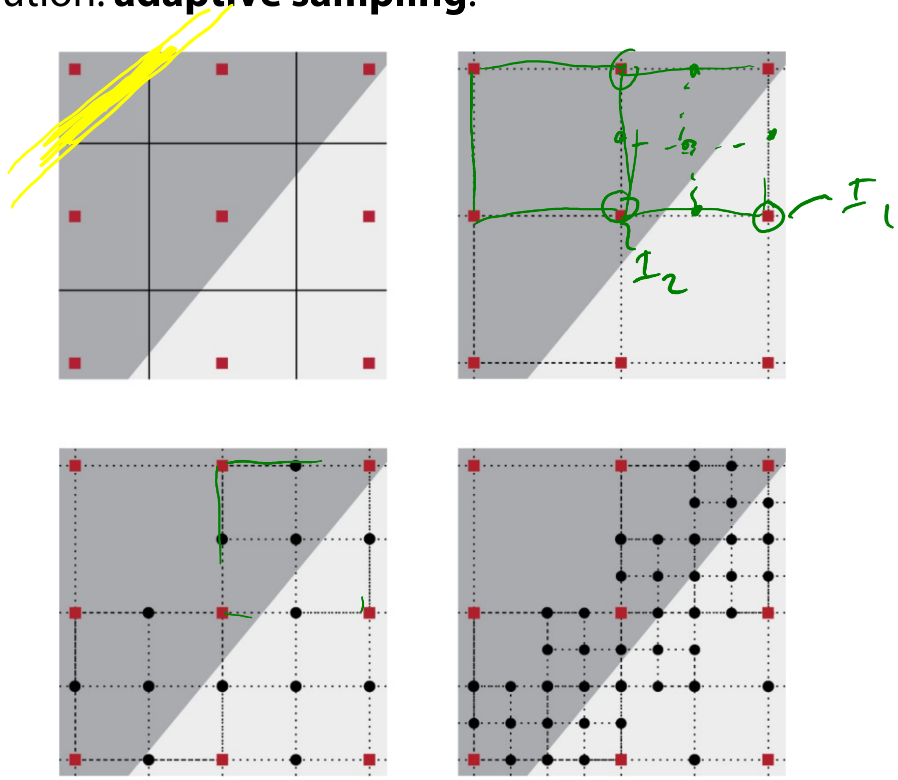
A.k.a., **super-sampling and averaging down**.

Antialiasing by adaptive sampling

Casting many rays per pixel can be unnecessarily costly. If there are no rapid changes in intensity at the pixel, maybe only a few samples are needed.

$\|I_1 - I_2\| > thresh$

Solution: **adaptive sampling**.



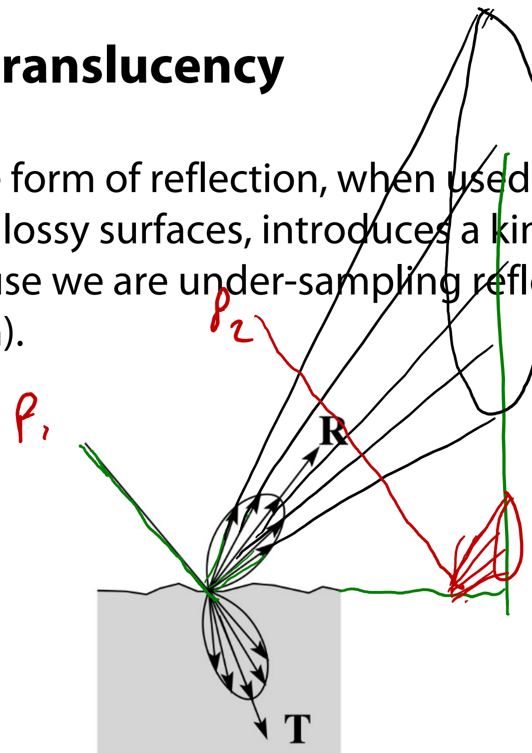
$$\frac{\|I_1 - I_2\|}{\frac{1}{2}\|I_1 + I_2\|}$$

Q: When do we decide to cast more rays in a particular area?

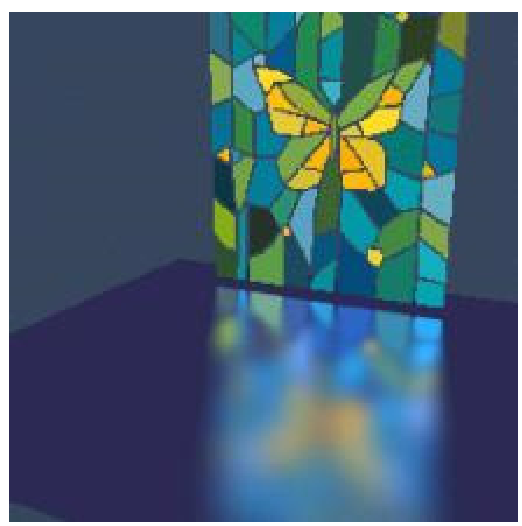
Gloss and translucency

The mirror-like form of reflection, when used to approximate glossy surfaces, introduces a kind of aliasing, because we are under-sampling reflection (and refraction).

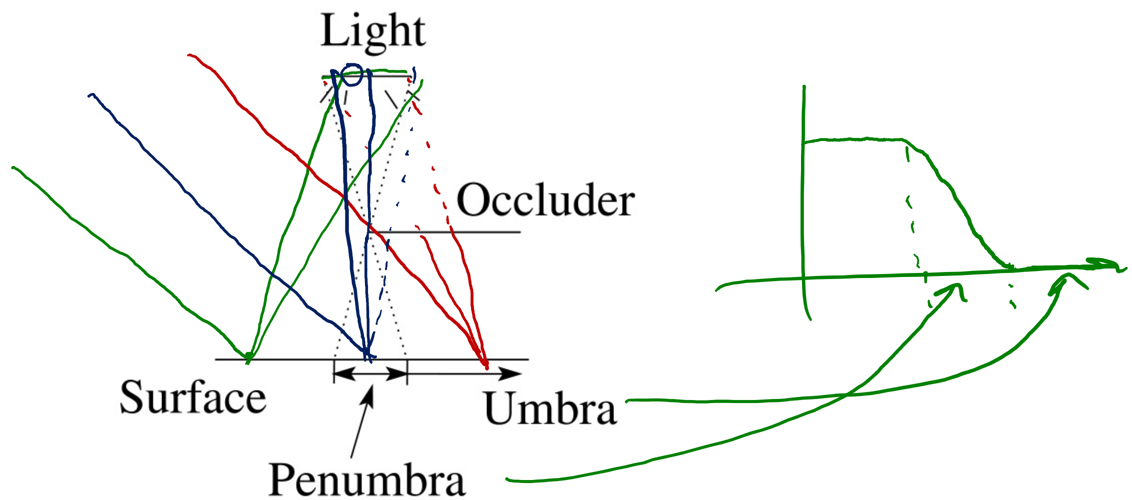
For example:



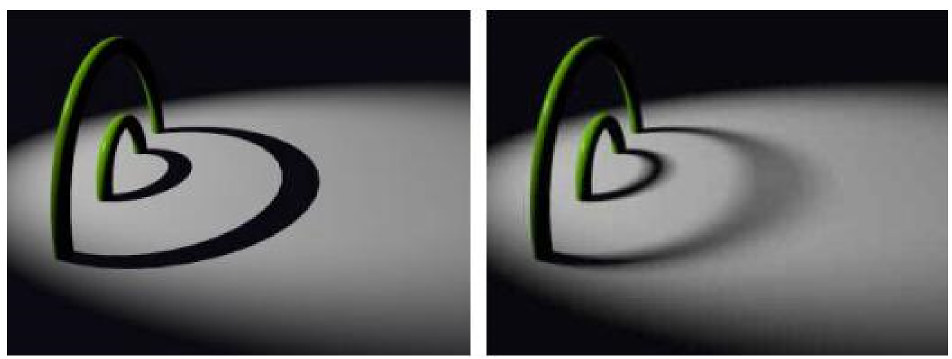
Distributing rays over reflection directions gives:



Soft shadows

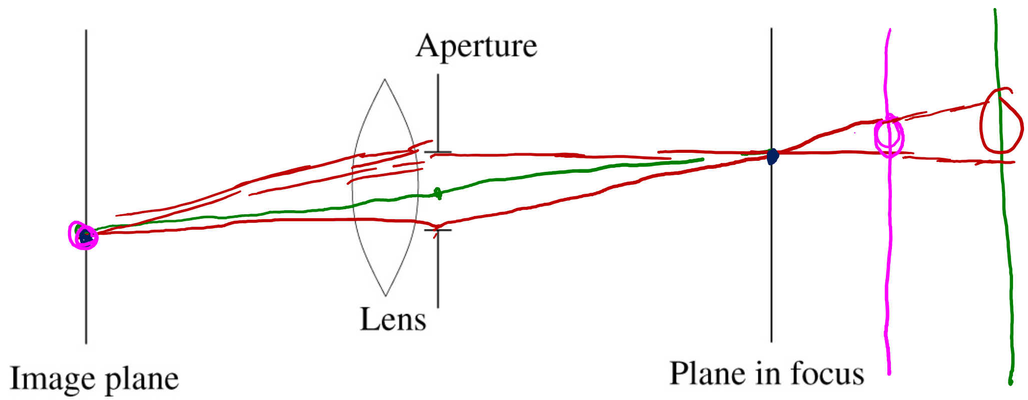


Distributing rays over light source area gives:



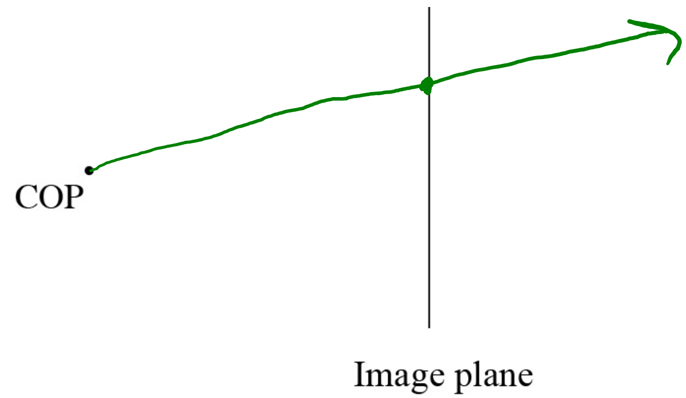
Depth of field

To simulate a camera, we can model the refraction of light through a lens. This will give us a "depth of field" effect: objects close to the in-focus plane are sharp, and the rest is blurry.



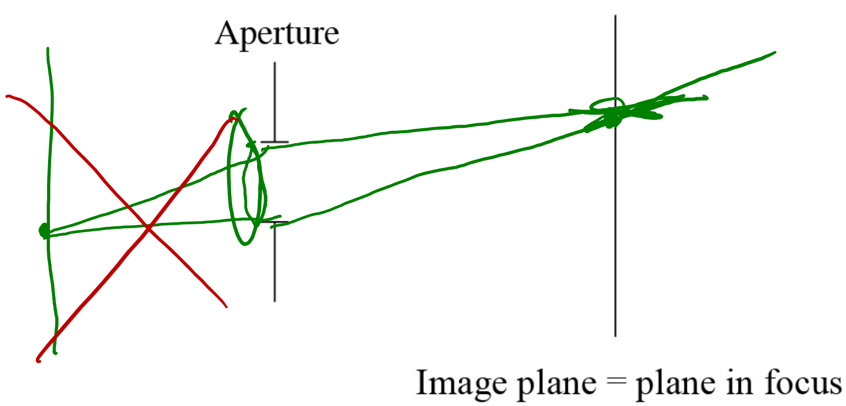
Depth of field (cont'd)

This is really similar to the pinhole camera model:



But now:

- Put the image plane at the depth you want to be in focus.
- Treat the aperture as multiple COPs (samples across the aperture).
- For each pixel, trace multiple viewing/primary rays for each COP and average the results.



Motion blur

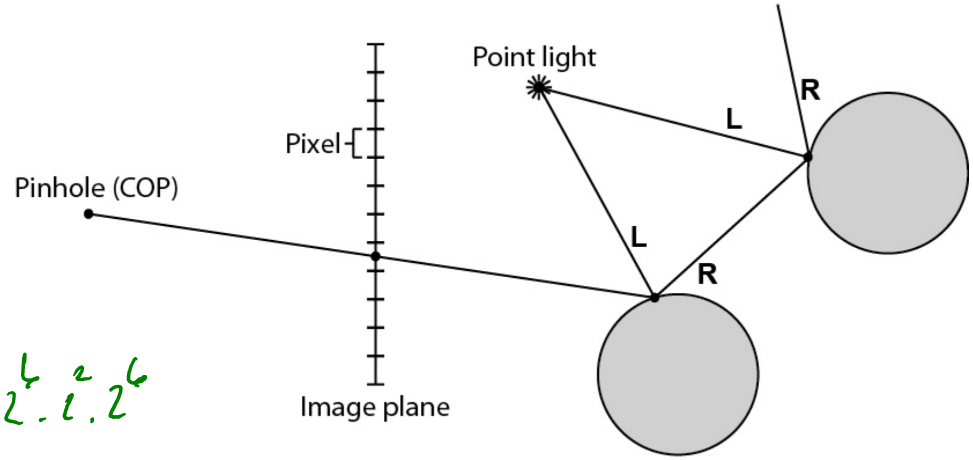
Distributing rays over time gives:

How can we use super-sampling and averaging down to get motion blur?

Naively improving Whitted ray tracing

Consider Whitted vs. a brute force approach with anti-aliasing, depth of field, area lights, gloss…

Whitted ray tracing



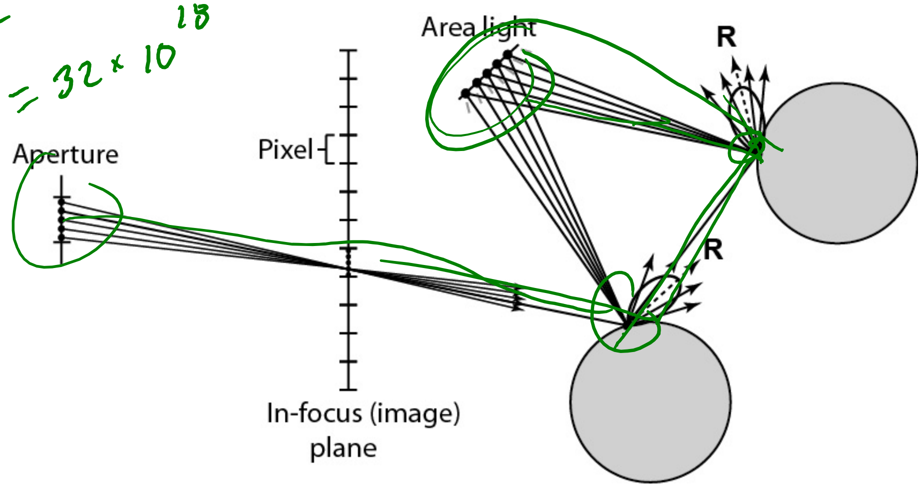
$10^6 \cdot 10^6 \cdot 2^6 \cdot 2^6 \cdot 2^6 \cdot 2^2 \cdot 2^6$

$= 10^{12} \cdot 2^{26}$

$= 10^{12} \cdot 2^6 (2^{10})^2$

$= 10^{12} \cdot 2^6 \cdot 10^6 = 32 \times 10^{18}$

Brute force, advanced ray tracing



Advanced ray tracing has:

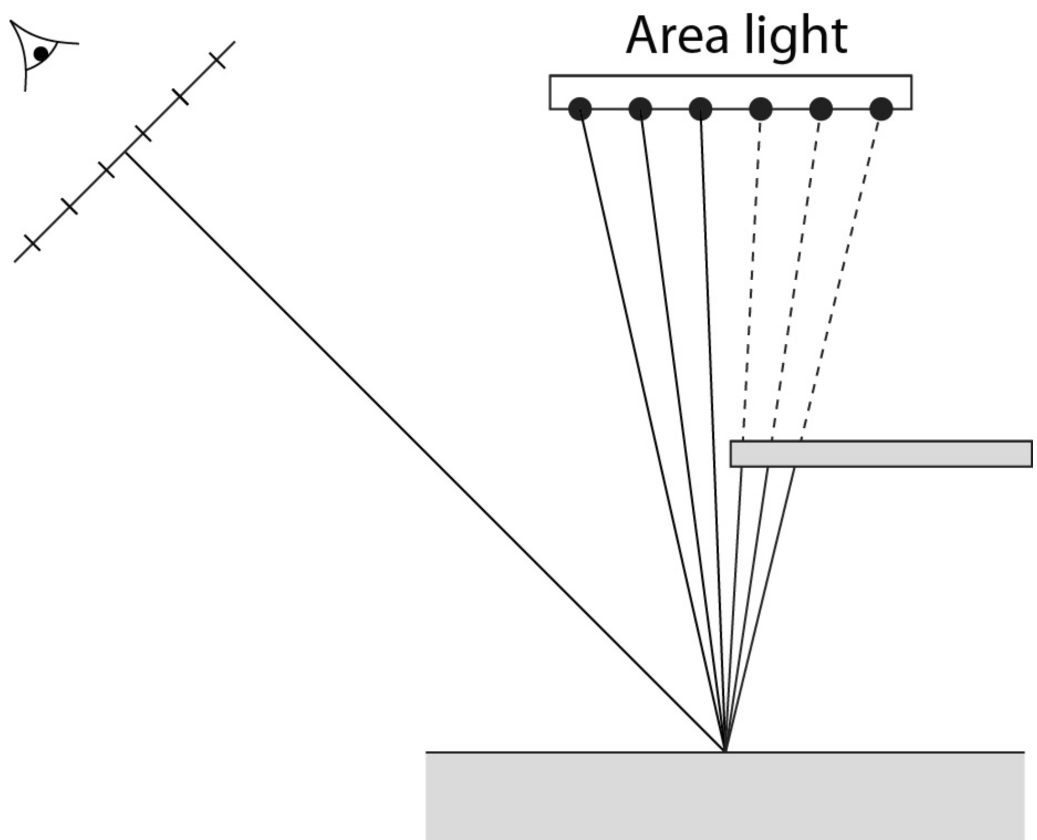
- m x m pixels
- k x k supersampling
- a x a sampling of camera aperture
- n primitives
- ℓ area light sources
- s x s sampling of each area light source
- r x r rays cast recursively per intersection (gloss/translucency)
- d is average ray path length

Asymptotic # of intersection tests $\approx O(m^2 k^2 a^2 n (\ell s^2 + r^2 (\ell s^2 + r^2 (\ldots)))$

For m =1,000, $k = a = s = r = 8$, n = 1,000,000, $\ell = 4$, $d = 8$ … very expensive!!

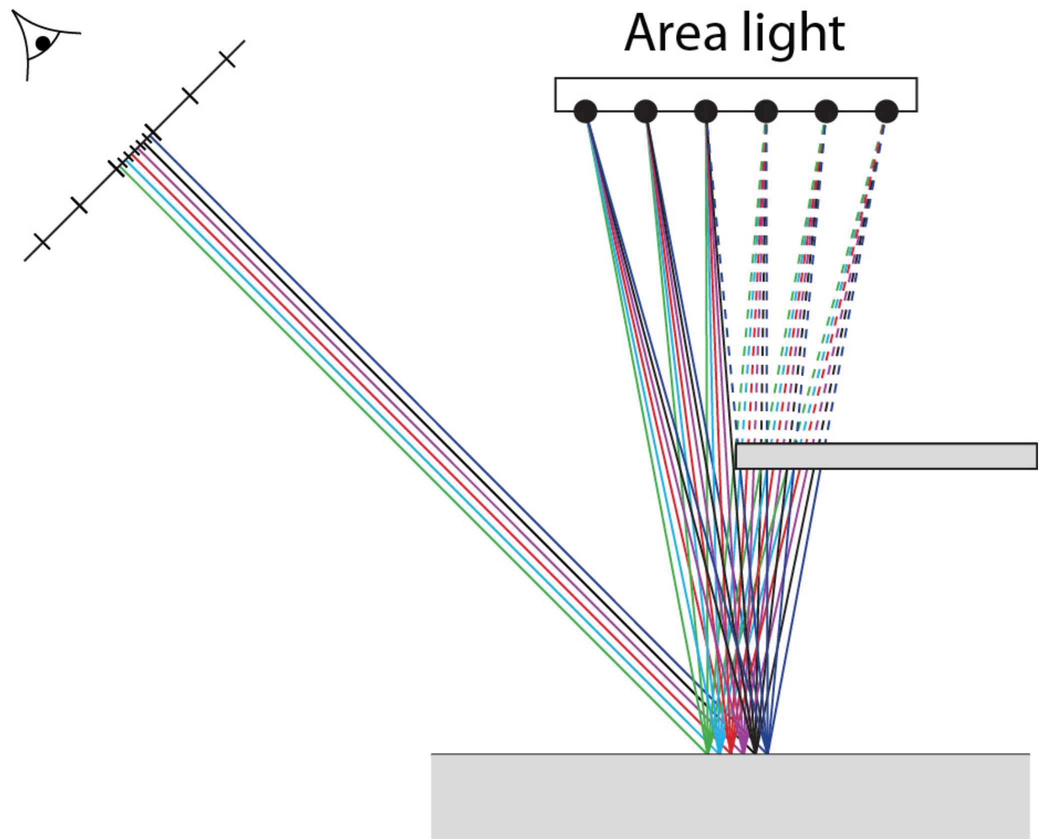
Penumbra revisited

Let's revisit the area light source…



We can trace a ray from the viewer through a pixel, but now when we hit a surface, we cast rays to samples on the area light source.

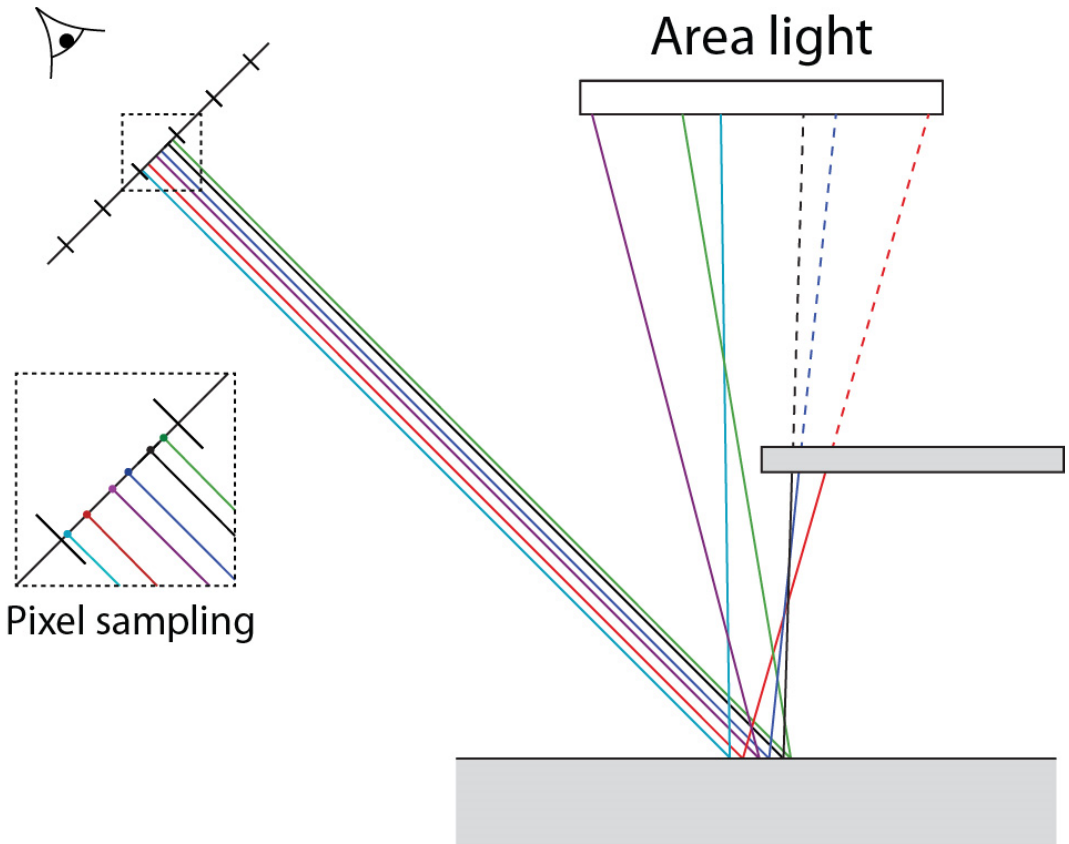
Penumbra revisited



We should anti-alias to get best looking results.

Whoa, this is a lot of rays…just for one pixel!!

Penumbra revisited



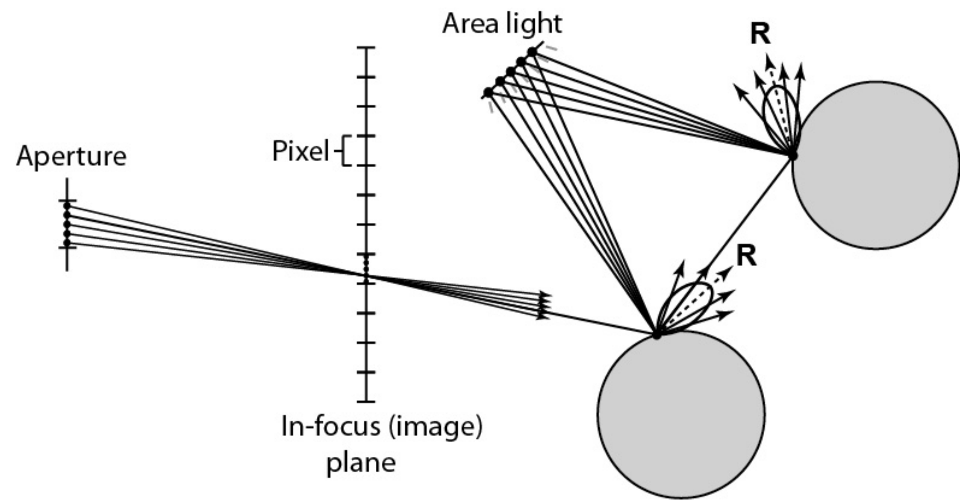
We can get a similar result with **much** less computation:

- Choose random location within a pixel, trace ray.
- At first intersection, choose random location on area light source and trace shadow ray.
- Continue recursion as with Whitted, but always choose random location on area light for shadow ray.

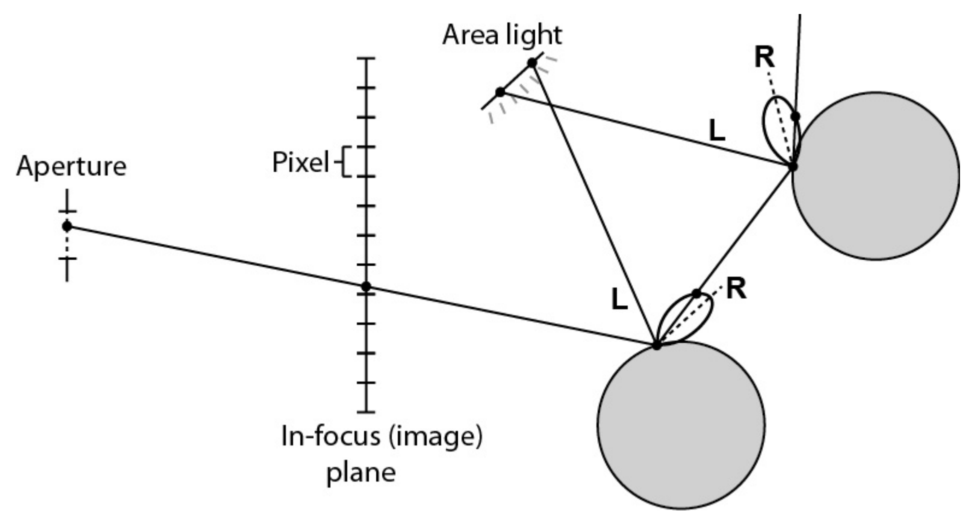
Monte Carlo Path Tracing vs. Brute Force

We can generalize this idea to do random sampling for each viewing ray, shadow ray, reflected ray, etc. This approach is called **Monte Carlo Path Tracing** (MCPT).

Brute force, advanced ray tracing



Monte Carlo path tracing

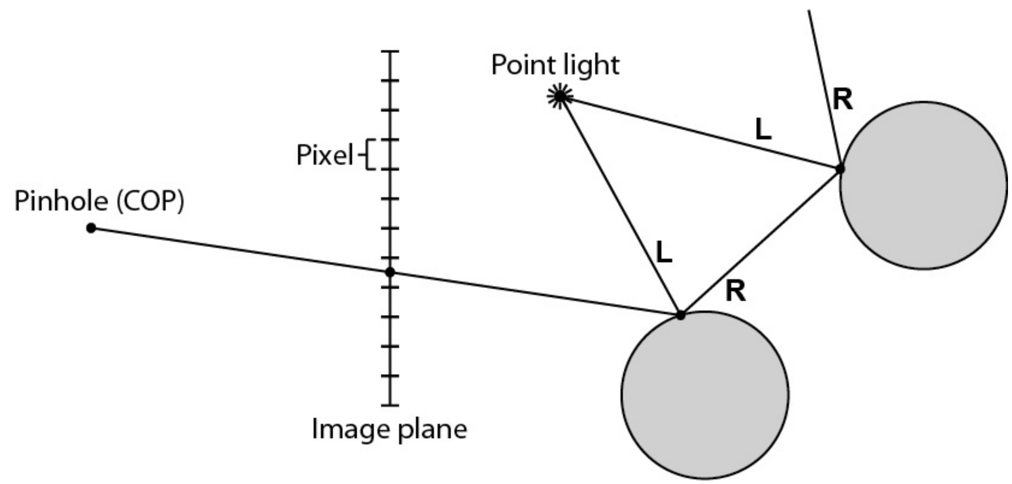


MCPT vs. Whitted

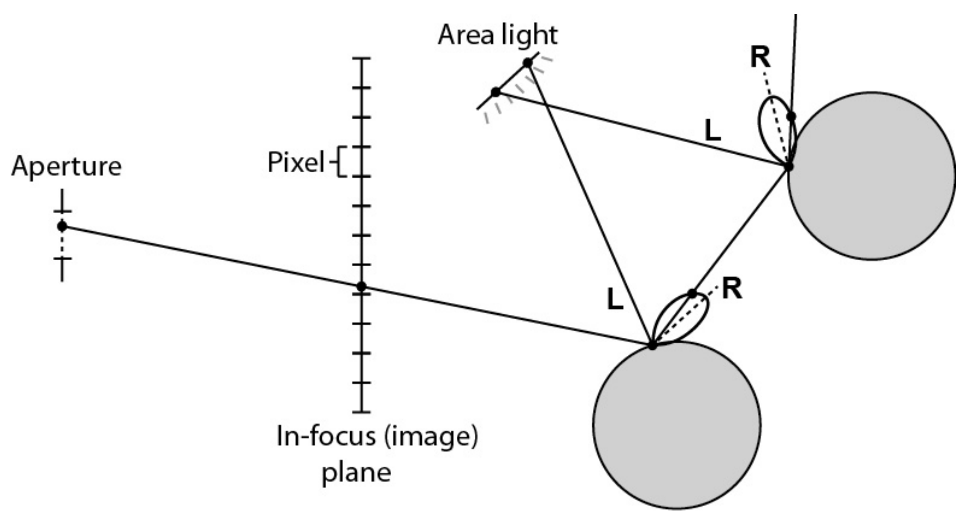
Q: For a fixed number of rays per pixel, does MCPT trace more total rays than Whitted?

Q: Does MCPT give the same answer every time?

Whitted ray tracing



Monte Carlo path tracing



Ray tracing as integration

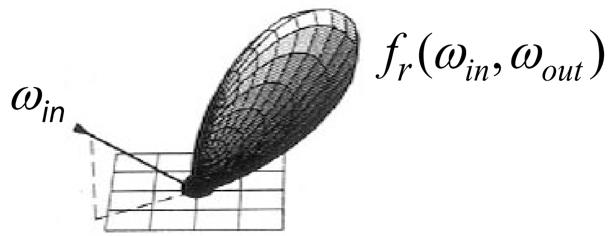
Ray tracing amounts to estimating a multi-dimensional integral at each pixel. The integration is over:

- the pixel area
- the aperture
- each light source
- all diffuse/glossy reflections (recursively)

MCPT images are often noisy. We can reduce noise by being smarter about which rays we cast…

Intergration over reflection

Integration over diffuse/glossy reflections is at the heart of rendering. Recall that the BRDF tells us how incoming light will scatter into outgoing directions:



By reciprocity, we can replace ω_{in} on the left side above with ω_{out}, and treat the function $f_r(\omega_{in}, \omega_{out})$ as the "sensitivity" to different incoming directions.

To compute the total light for an outgoing direction, we integrate all incoming directions:

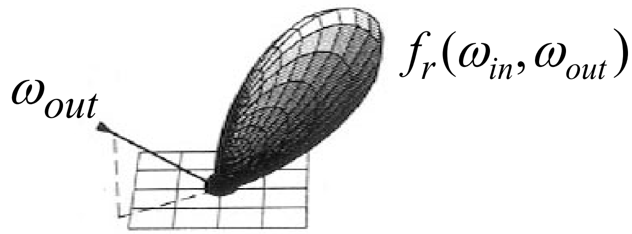
$$I(\omega_{out}) = \int_H I(\omega_{in}) f_r(\omega_{in}, \omega_{out}) \left(\omega_{in} \cdot \mathbf{N}\right) d\omega_{in}$$

To integrate in with MCPT, when considering reflection recurstion, we could just:

- Cast a ray in a (uniformly) random direction
- Weight the result by $f_r(\omega_{in}, \omega_{out})\left(\omega_{in} \cdot \mathbf{N}\right)$

Importance sampling of reflection

For a given BRDF:



again the surface reflection equation is:

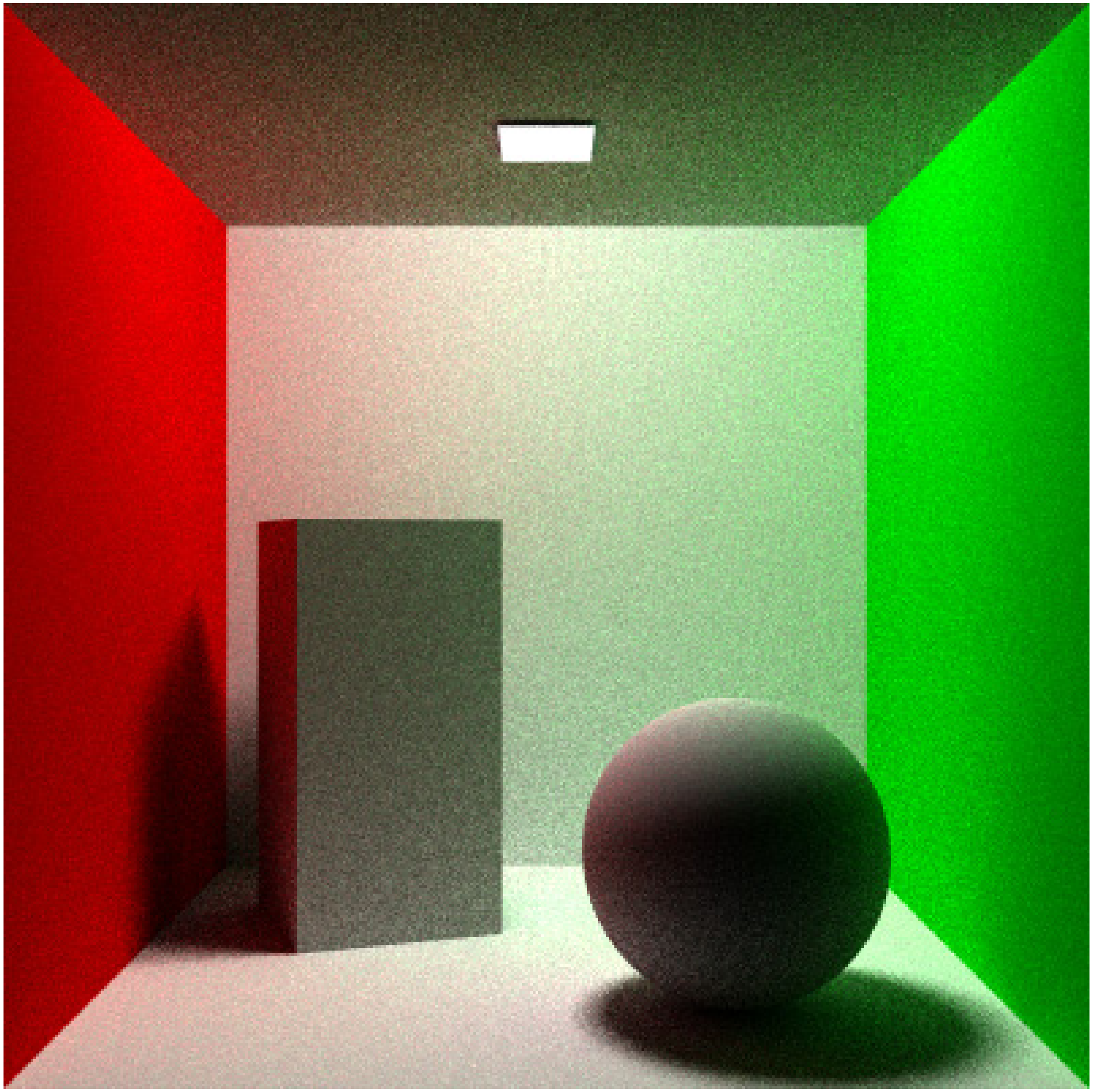
$$I(\omega_{out}) = \int_H I(\omega_{in}) f_r(\omega_{in}, \omega_{out}) (\omega_{in} \cdot \mathbf{N}) \, d\omega_{in}$$

With importance sampling:

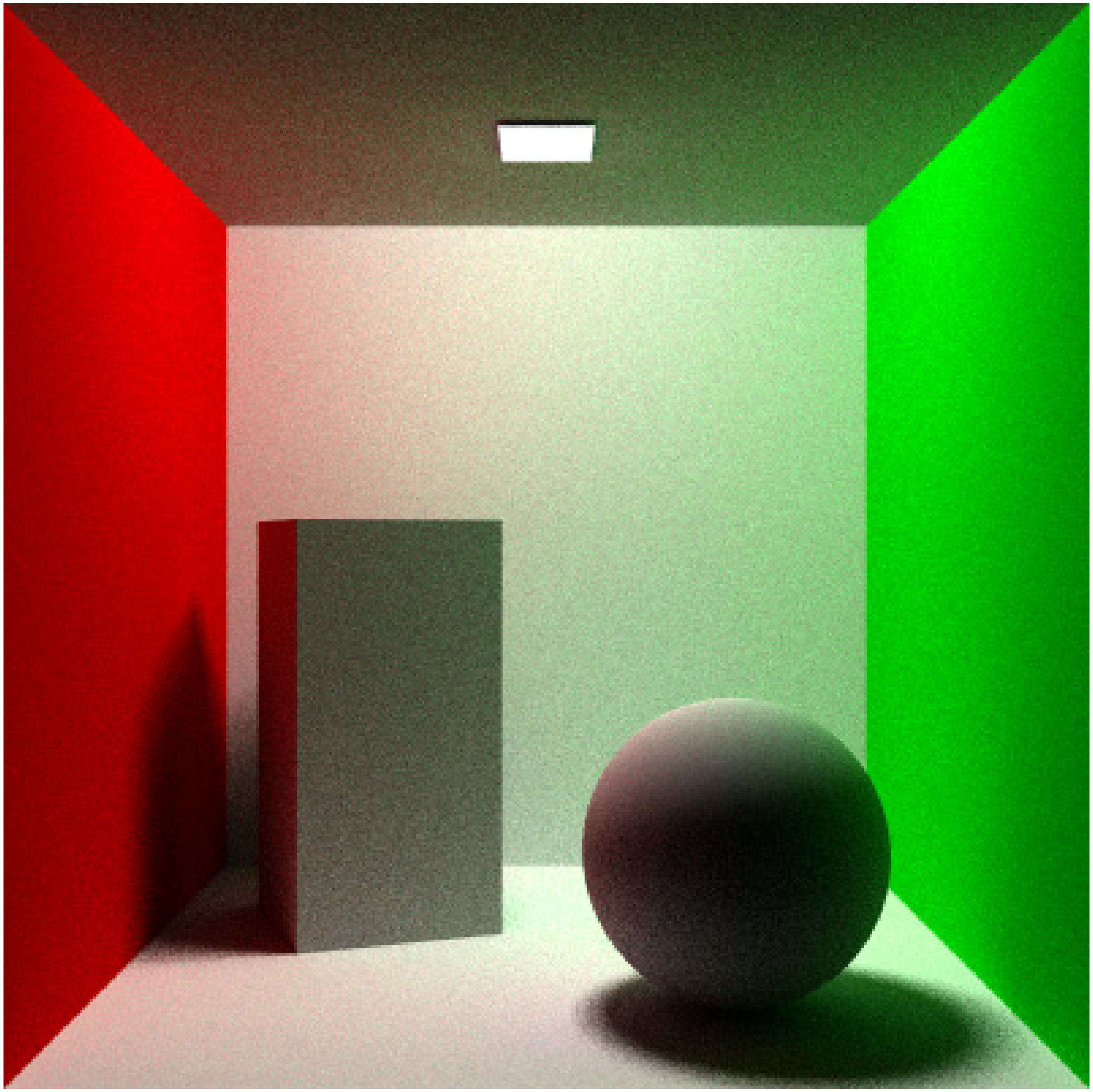
- Cast a ray in a direction drawn from a distribution $p(\omega_{in})$ that is large where the BRDF is large.
- Weight the ray by: $f_r(\omega_{in}, \omega_{out}) (\omega_{in} \cdot \mathbf{N}) / p(\omega_{in})$

Ideally, the distribution is proportional to the BRDF:

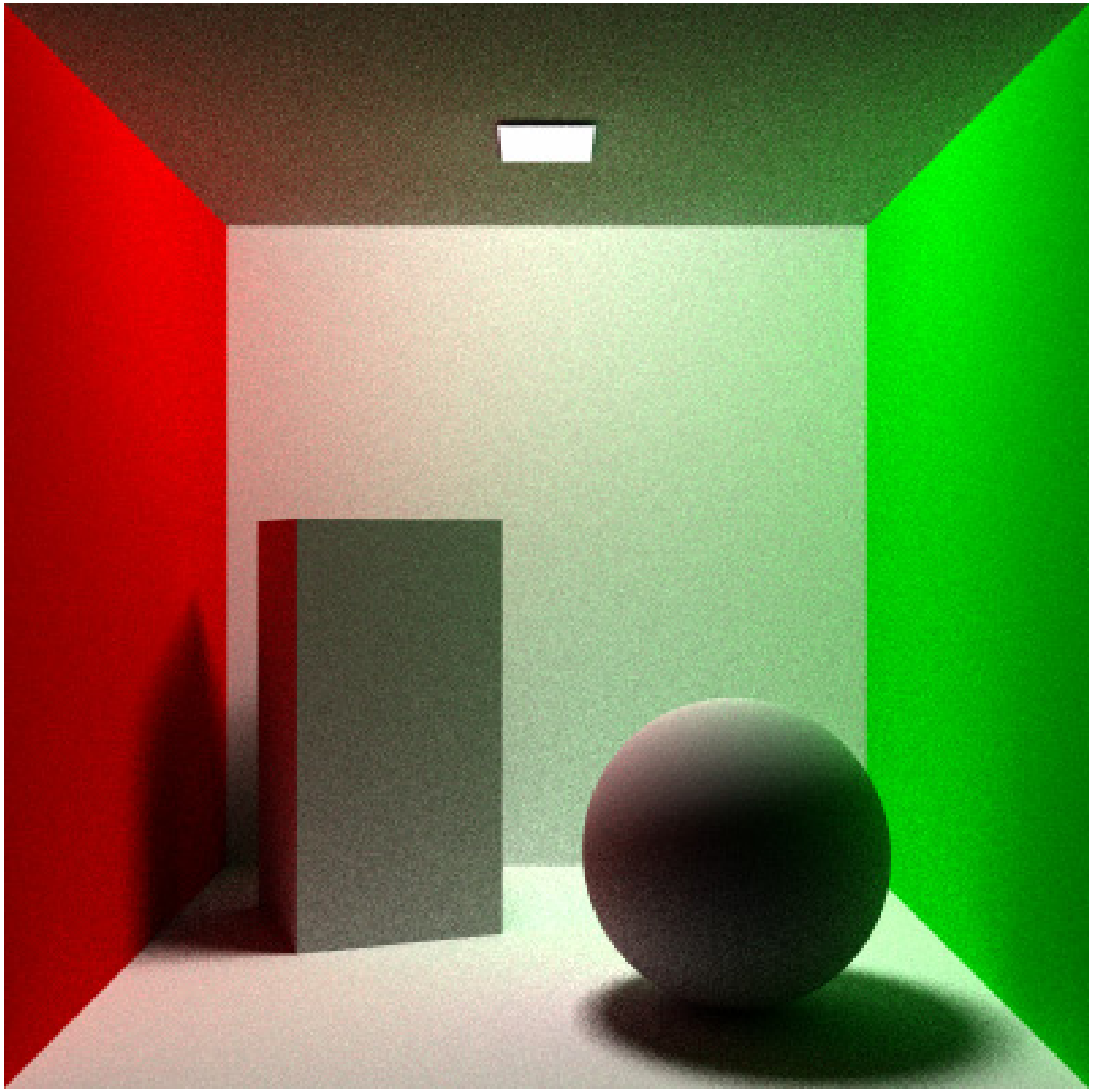
$$p(\omega_{in}) \sim f_r(\omega_{in}, \omega_{out}) (\omega_{in} \cdot \mathbf{N})$$



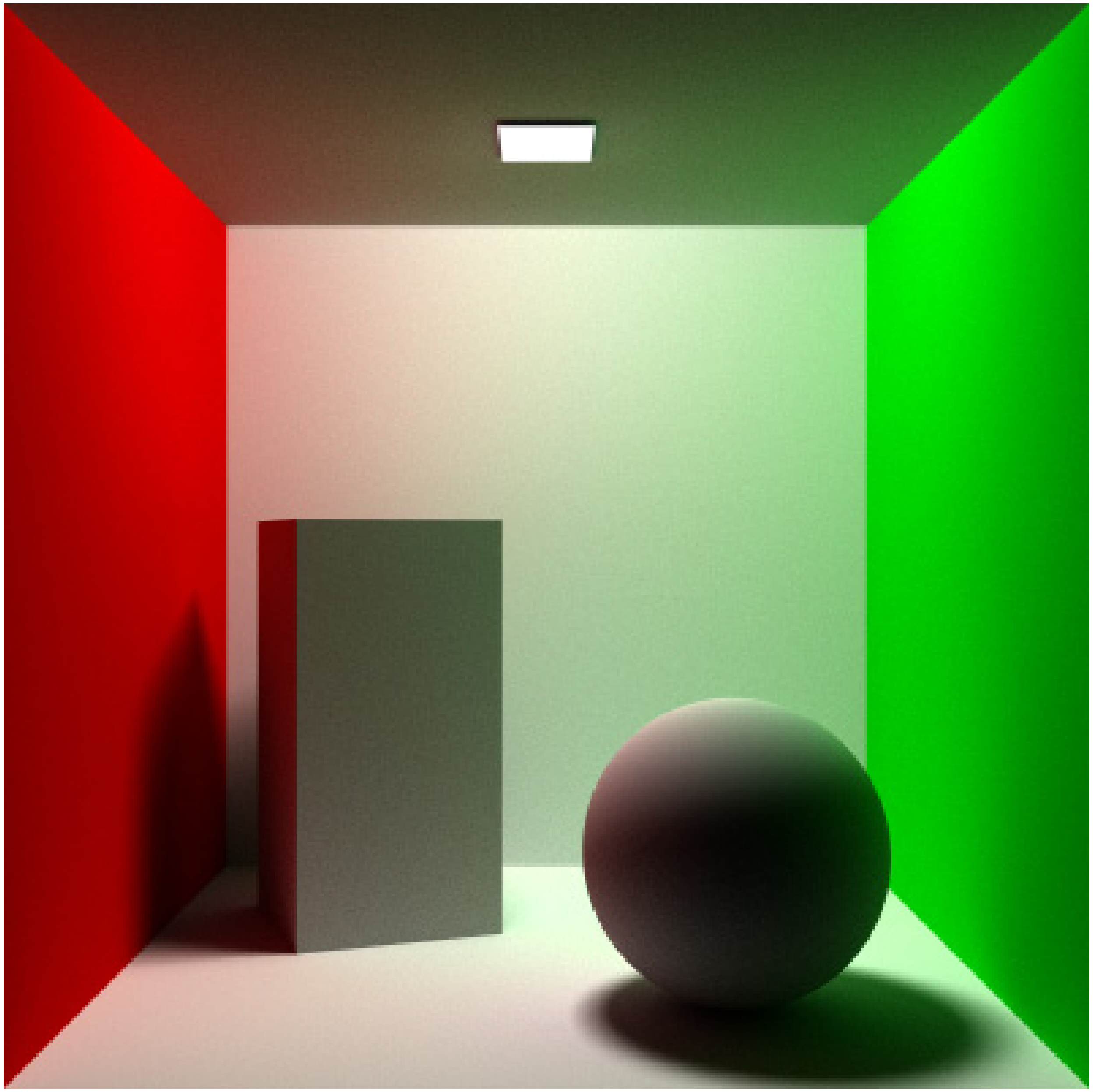
100 rays/pixel **without** importance sampling



100 rays/pixel **with** importance sampling

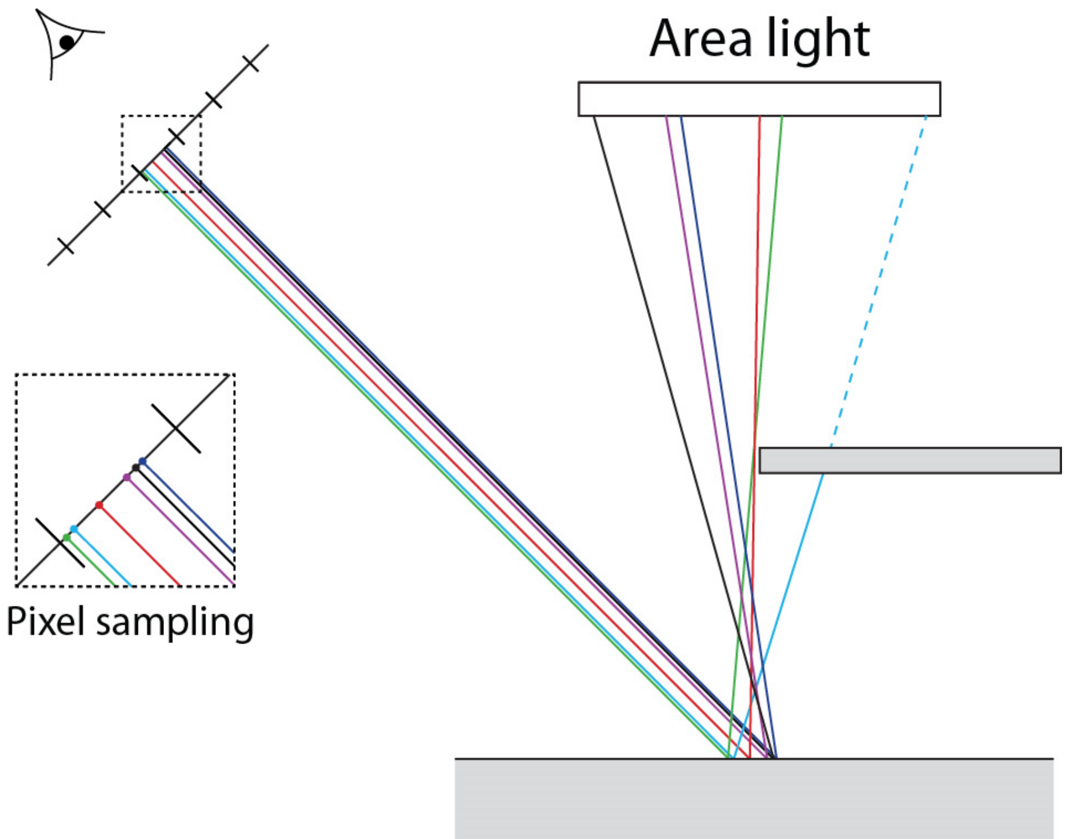


200 rays/pixel **without** importance sampling

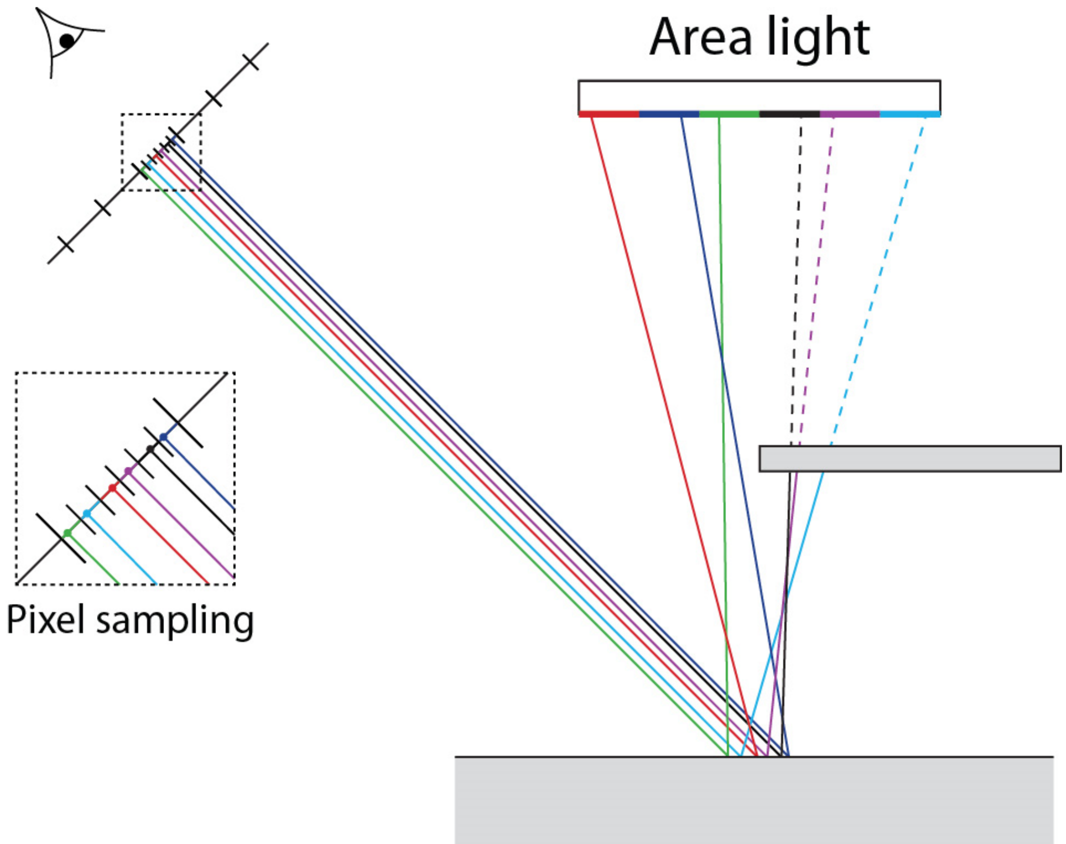


900 rays/pixel **with** importance sampling

Penumbra revisited: clumped samples



Penumbra: stratified sampling

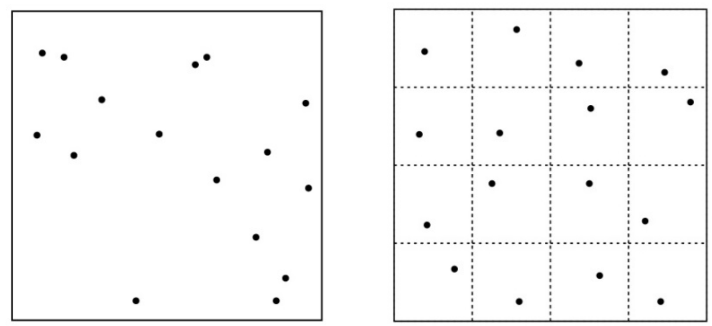


Stratified sampling gives a better distribution of samples:

- Break pixel and light source into *regions*.
- Choose random locations within each region.
- Trace rays through/to those jittered locations.

Stratified sampling of a 2D pixel

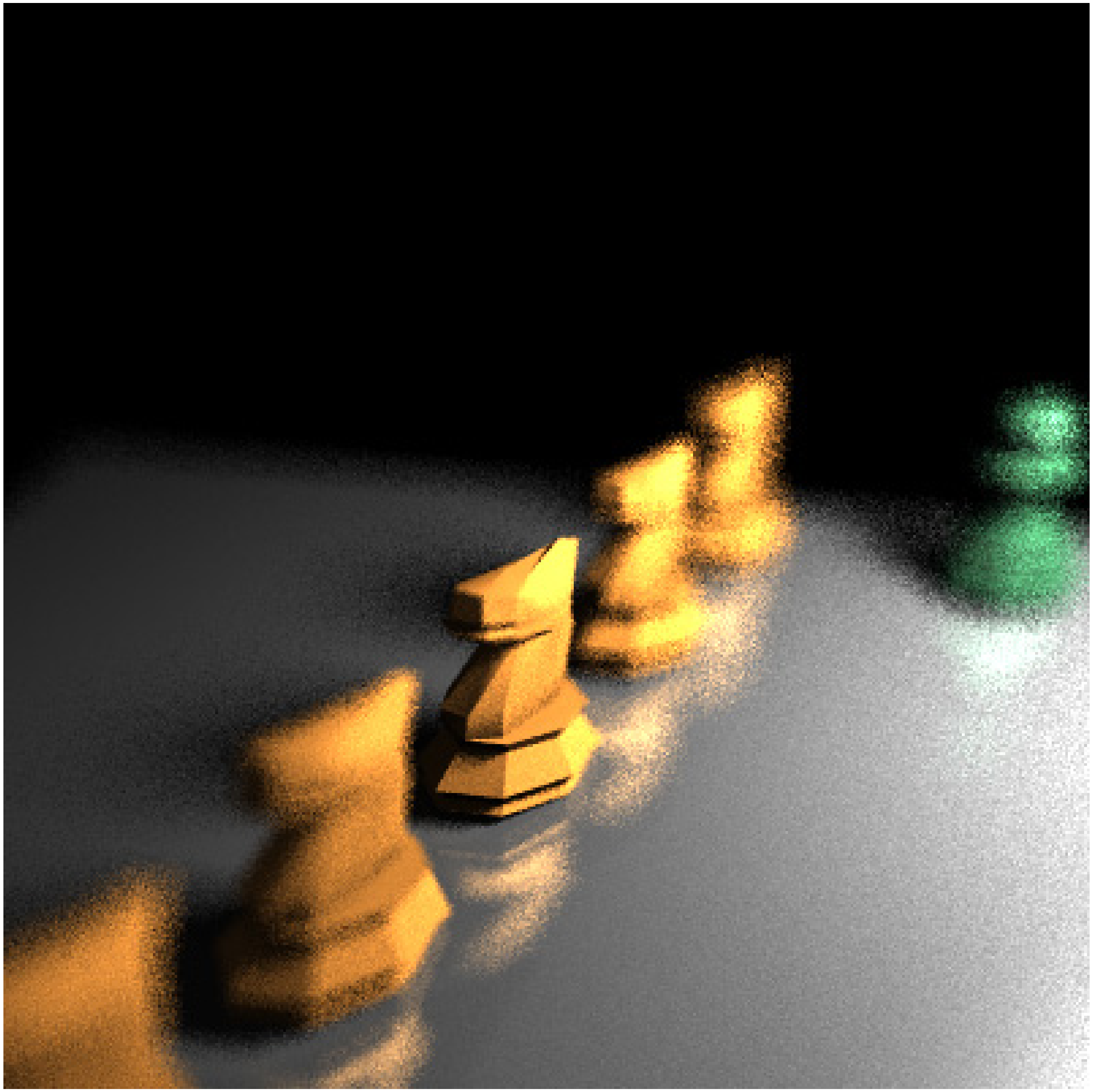
Here we see pure uniform vs. stratified sampling over a 2D pixel (here 16 rays/pixel):



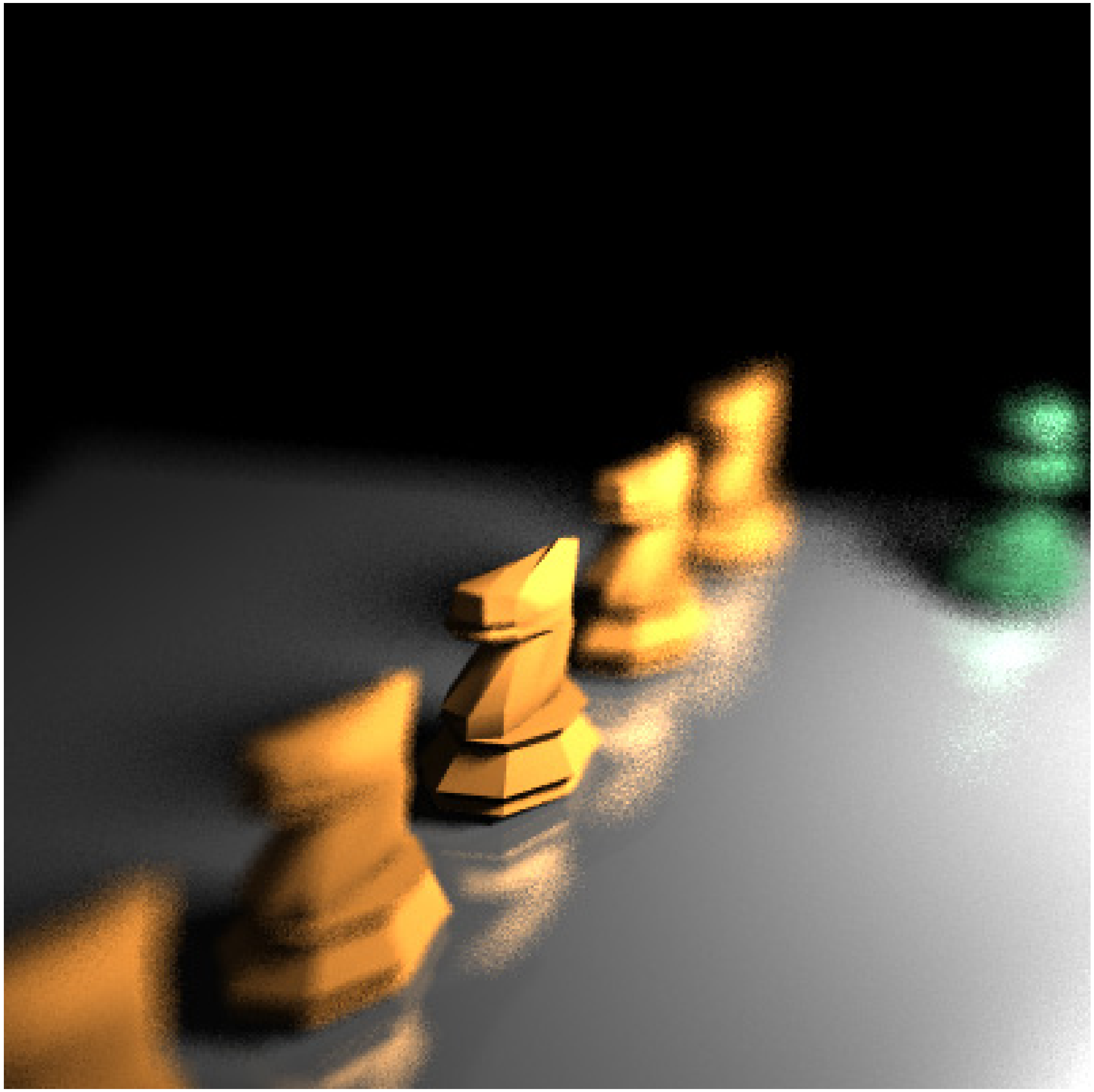
Random Stratified

The stratified pattern on the right is also sometimes called a **jittered** sampling pattern.

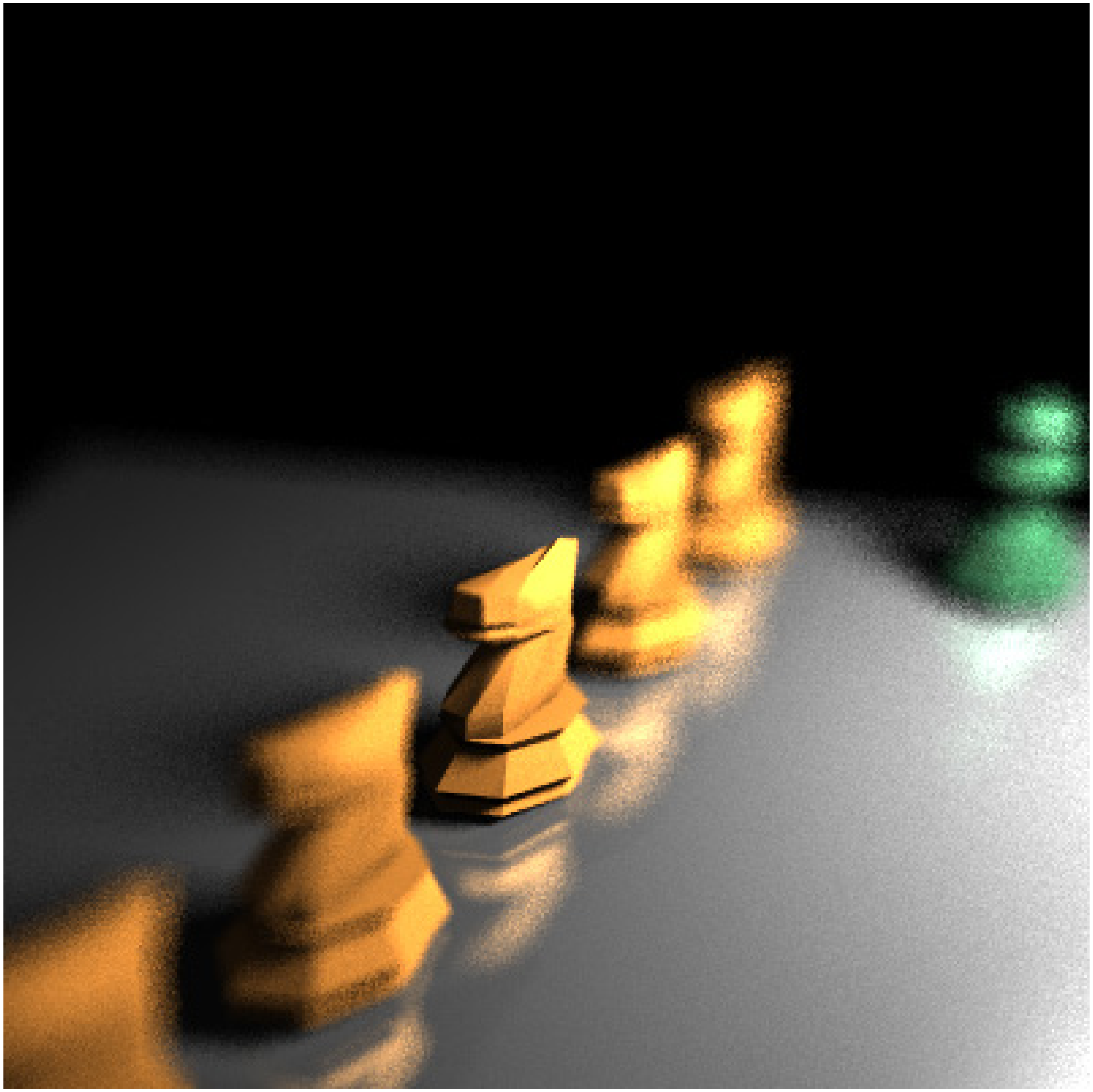
Similar grids can be constructed over the camera aperture, light sources, and diffuse/glossy reflection directions.



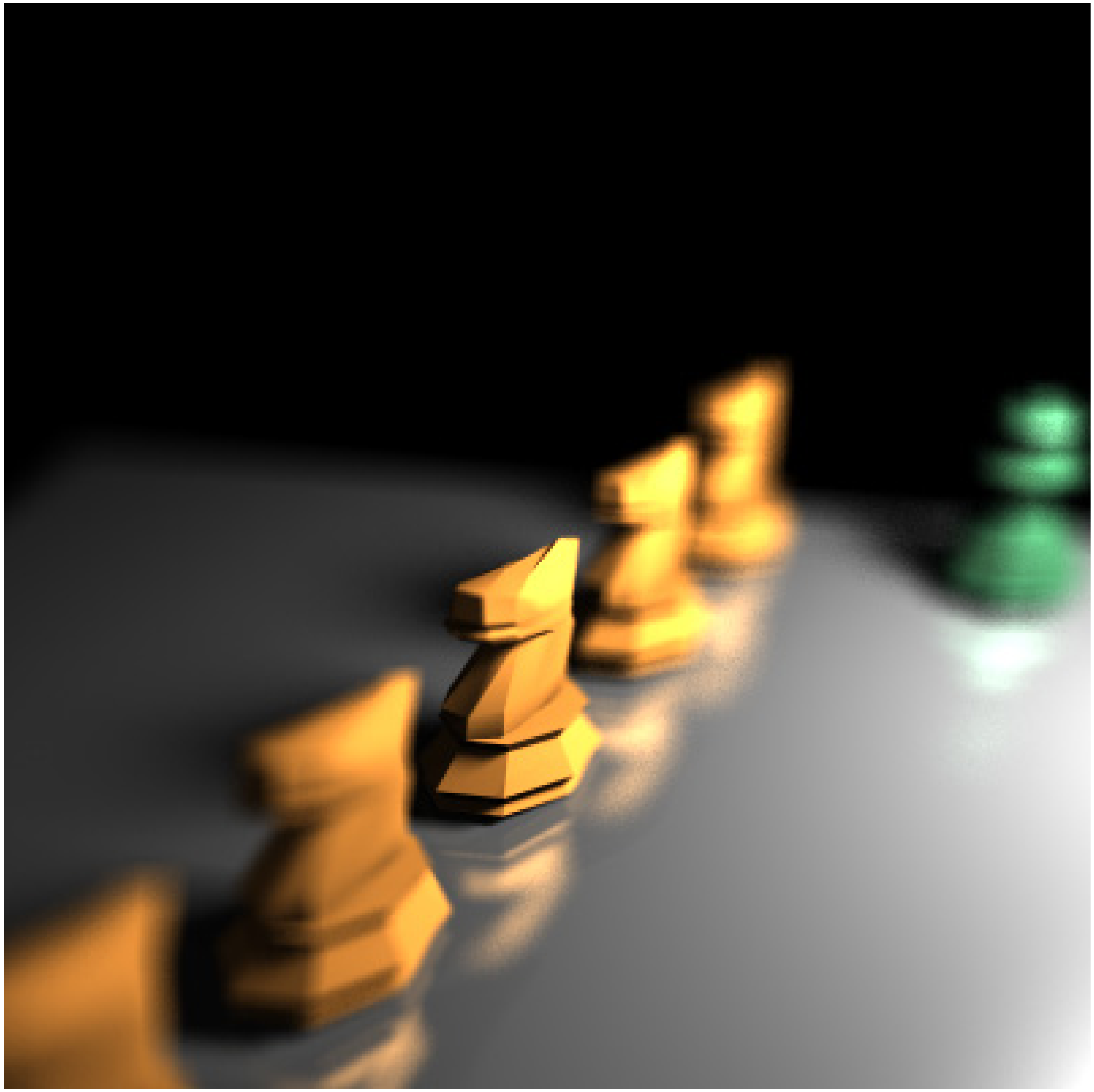
25 rays/pixel **without** stratified sampling



25 rays/pixel **with** stratified sampling



64 rays/pixel **without** stratified sampling



400 rays/pixel **with** stratified sampling

Summary

What to take home from this lecture:

- The meanings of all the boldfaced terms.
- An intuition for what aliasing is.
- How to reduce aliasing artifacts in a ray tracer
- The limitations of Whitted ray tracing (no glossy surfaces, etc.)
- The main idea behind Monte Carlo path tracing and what effects it can simulate (glossy surfaces, etc.)