Anti-aliasing and Monte Carlo Path Tracing

Adriana Schulz

CSE 457

Fall 2020

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.

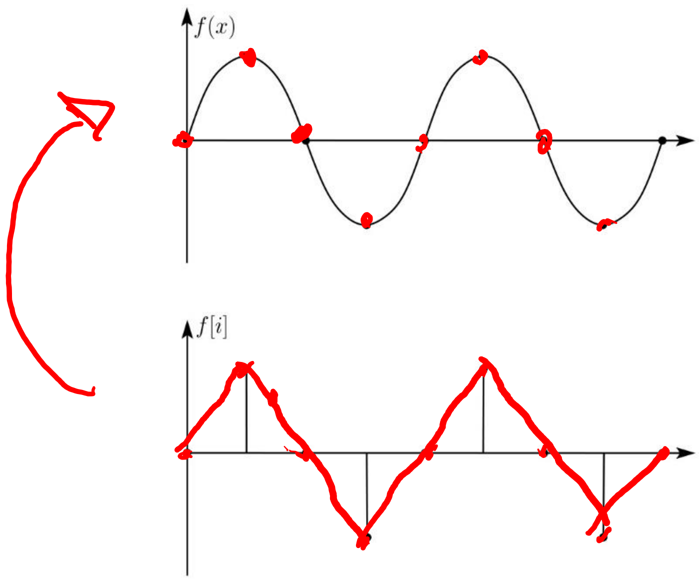
Aliasing

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 $f(x)$?

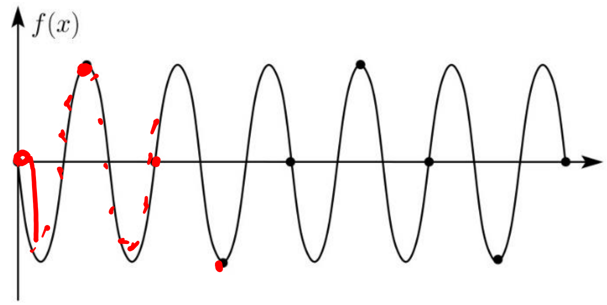
Consider sampling a sinusoid:

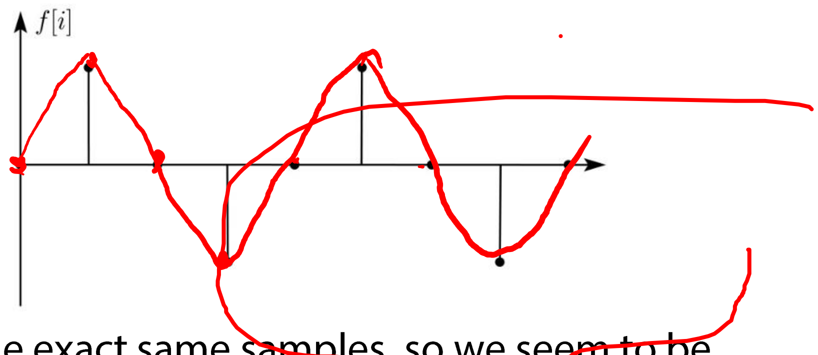


In this case, the sinusoid is reasonably well approximated by the samples.

Aliasing (con't)

Now consider sampling a higher frequency sinusoid



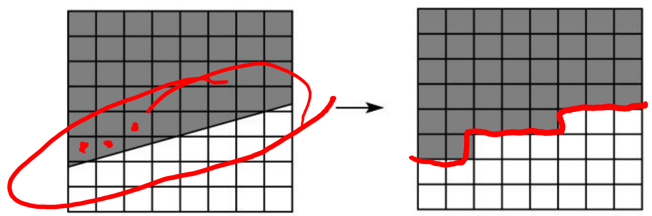


We get the exact same samples, so we seem to be approximating the first lower frequency sinusoid again.

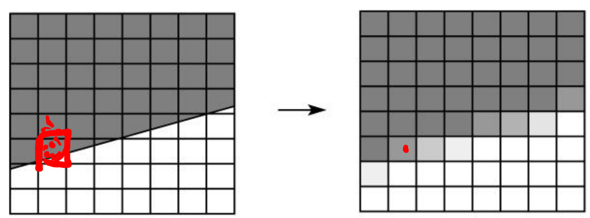
We say that, after sampling, the higher frequency sinusoid has taken on a new "alias", i.e., changed its identity to be a lower frequency sinusoid.

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:

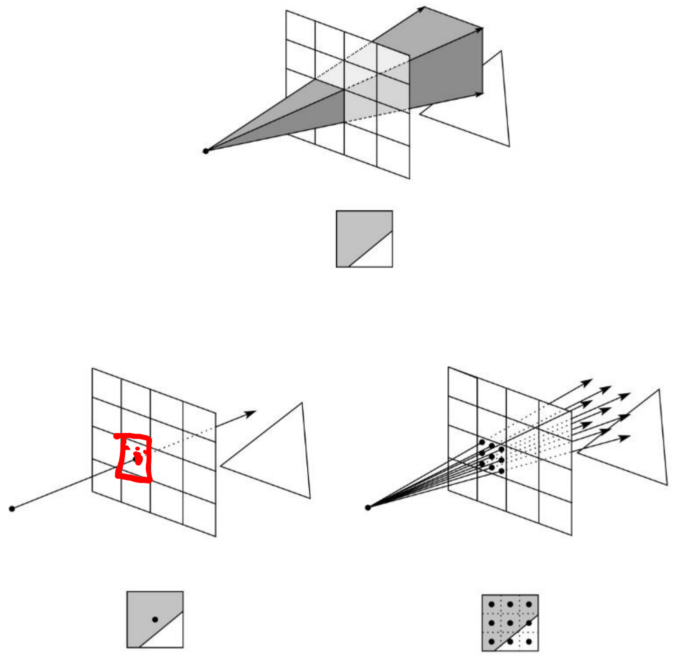


Anti-aliasing is the process of removing high frequencies *before* they cause aliasing.

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.

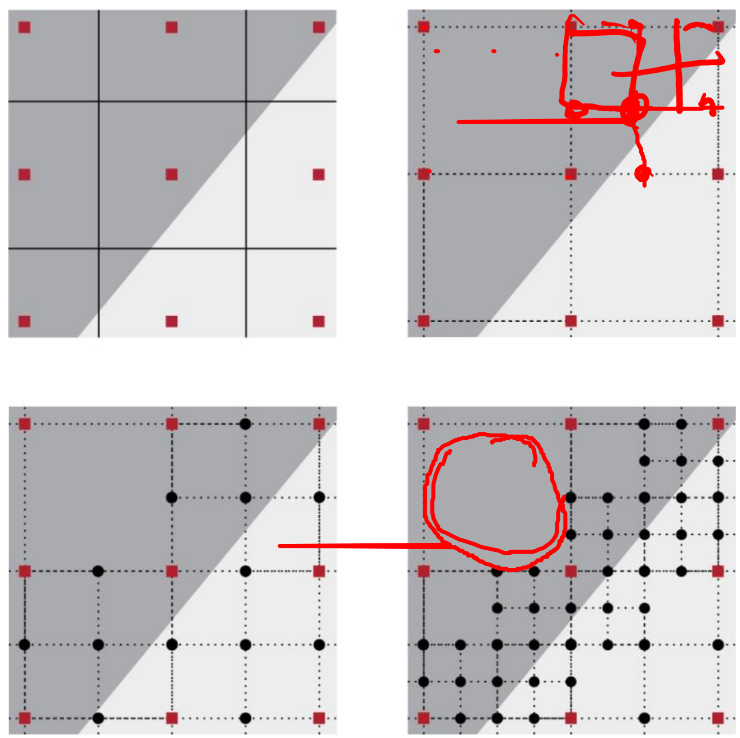
To improve matters, we can cast more than one ray per pixel and average the result.

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.

Solution: **adaptive sampling**.

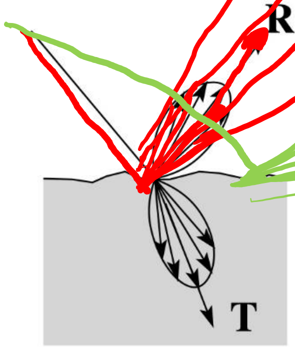


Q: When do we decide to cast more rays in a particular area?

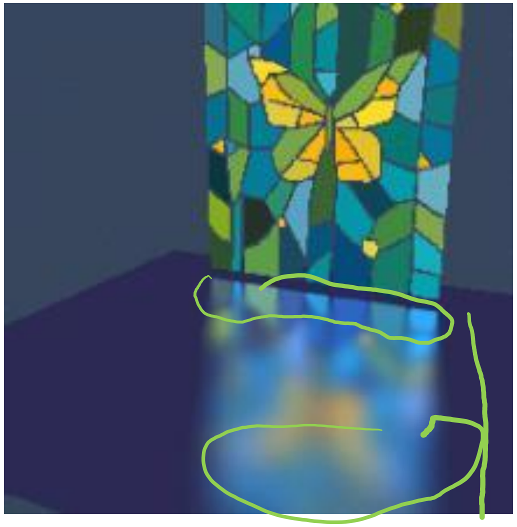
Gloss and translucency

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).

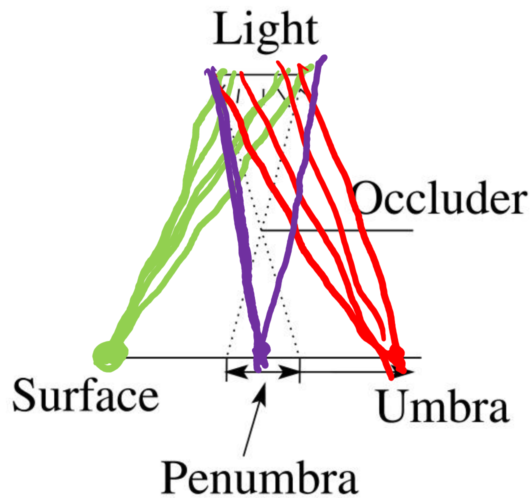
For example:

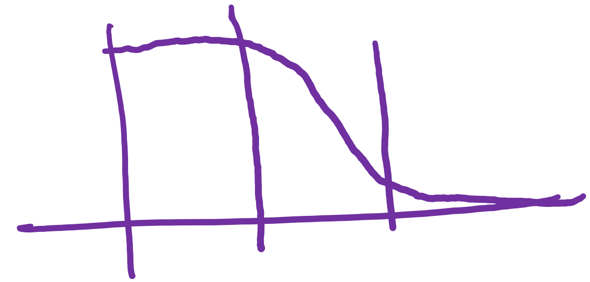


Distributing rays over reflection directions gives:

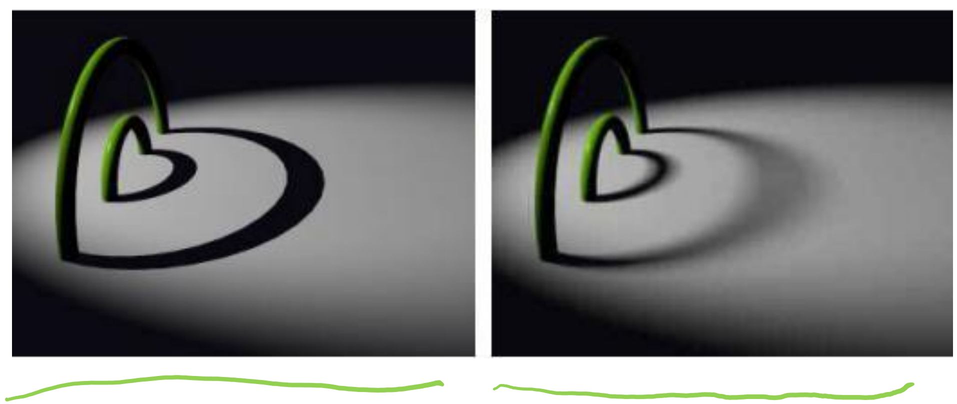


Soft shadows



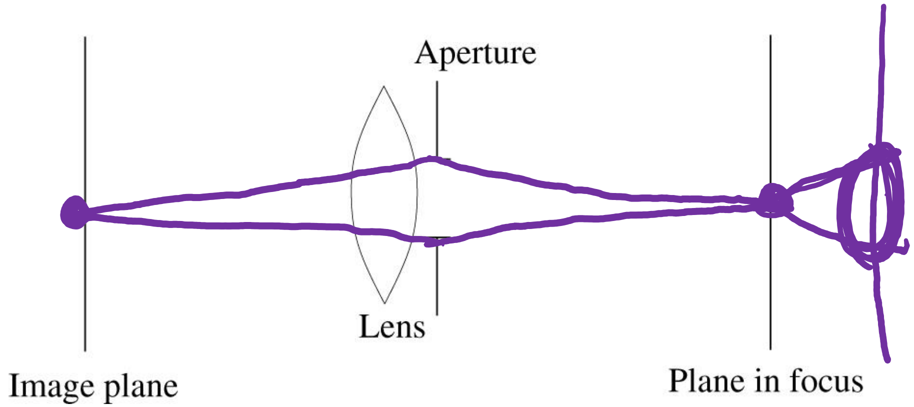


Distributing rays over light source area gives:



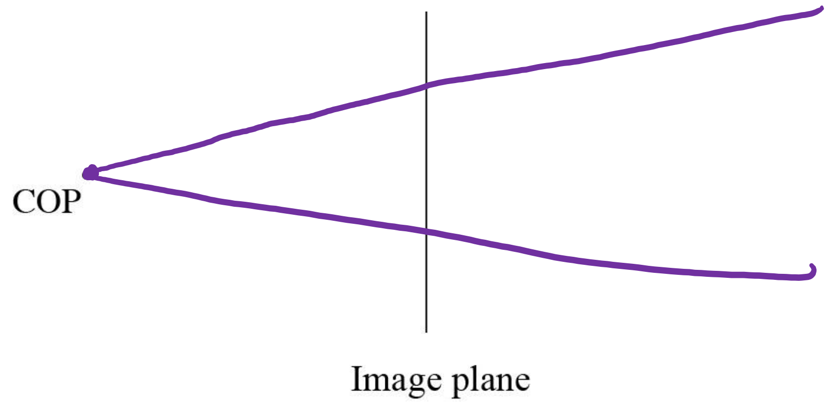
Depth of field

To simulate depth of field, we can model the refraction of light through a lens. Objects close to the in-focus plane are sharp, and the rest is blurry.



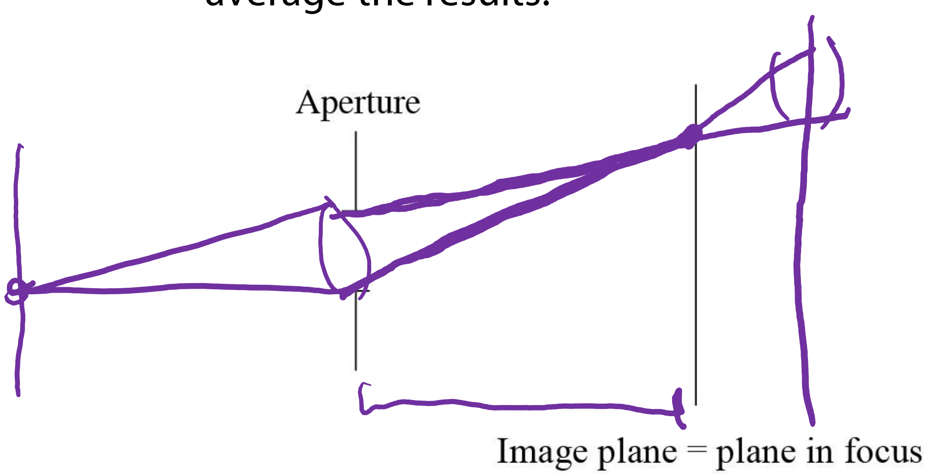
Depth of field (cont'd)

Neat idea: we can think of this as a generalization of the graphics pinhole model:



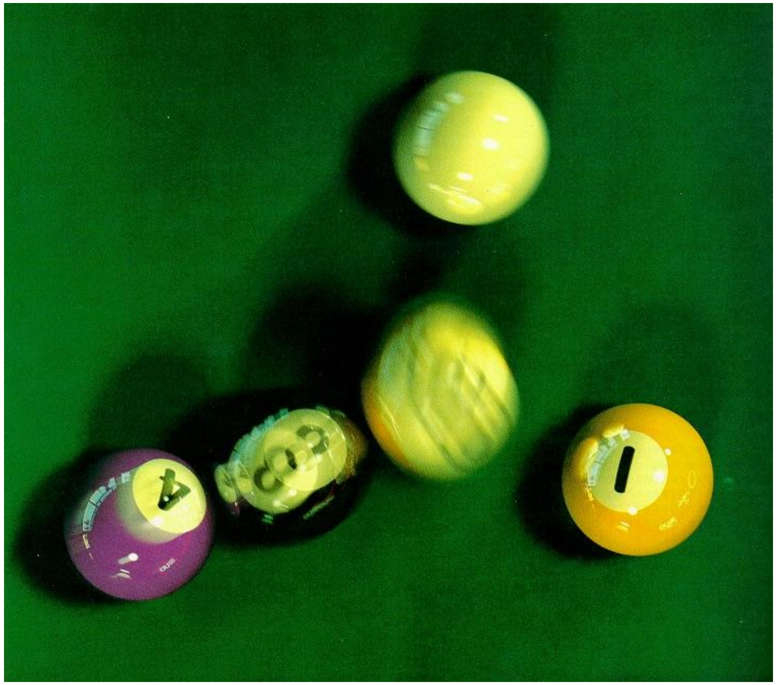
But now:

- 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:

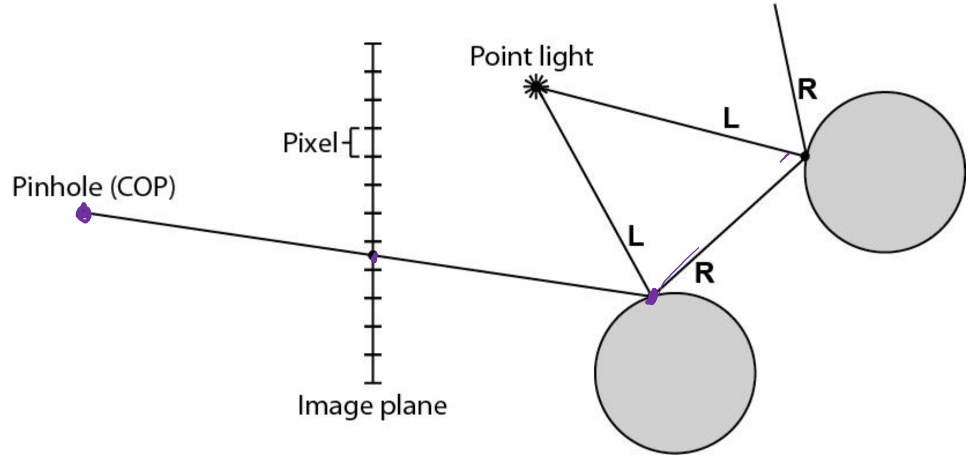


How can we use super-sampling and averaging down to get motion blur?

Naively improving Whitted ray tracing

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

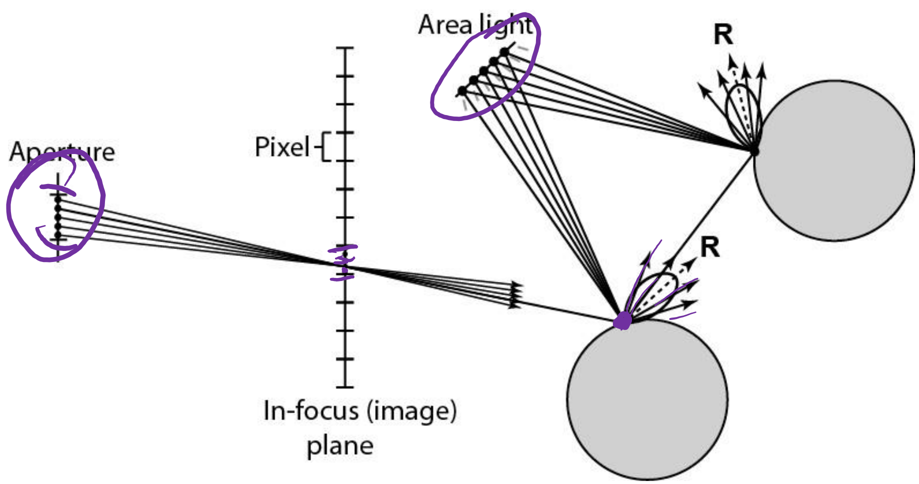
Whitted ray tracing



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

Brute force, advanced ray tracing

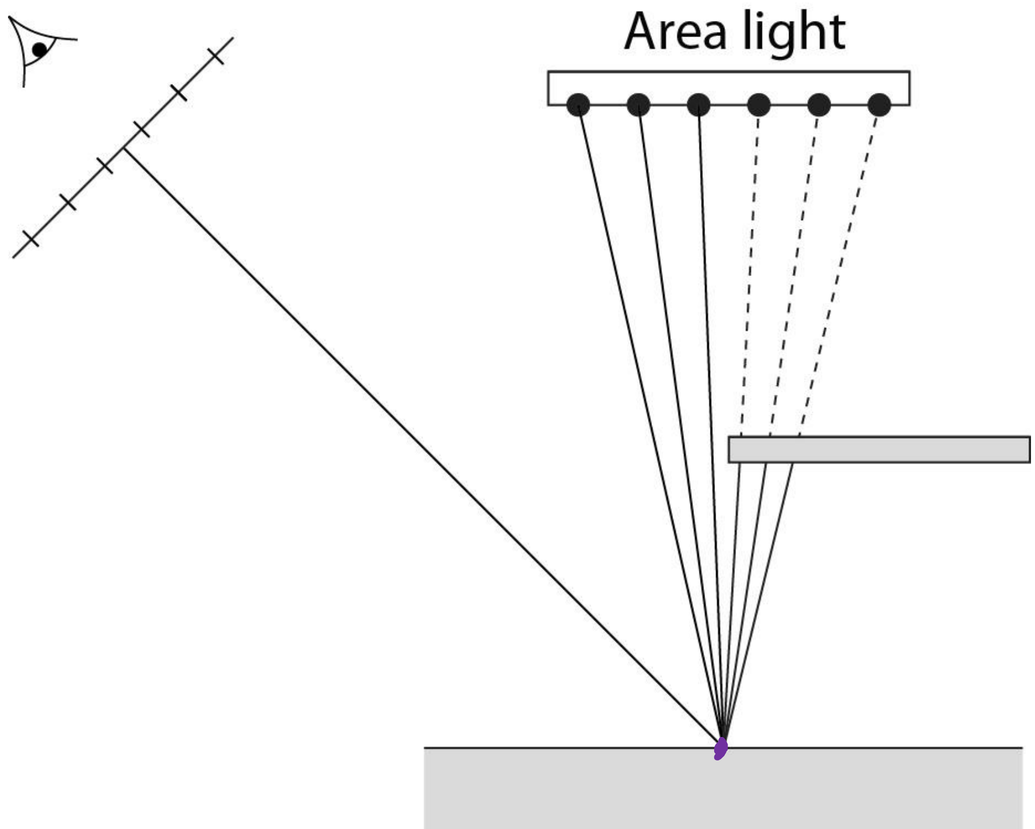


Asymptotic # of intersection tests ≈ $O(n\, m^2 k^2 a^2 (\ell s^2 + r^2(\ell s^2 + r^2(\ldots)))$

For m = 1,000, $k = a = s = r = 8$, n = 1,000,000, $\ell = 4$, $d = 8$ … very expensive!! 10^{18}

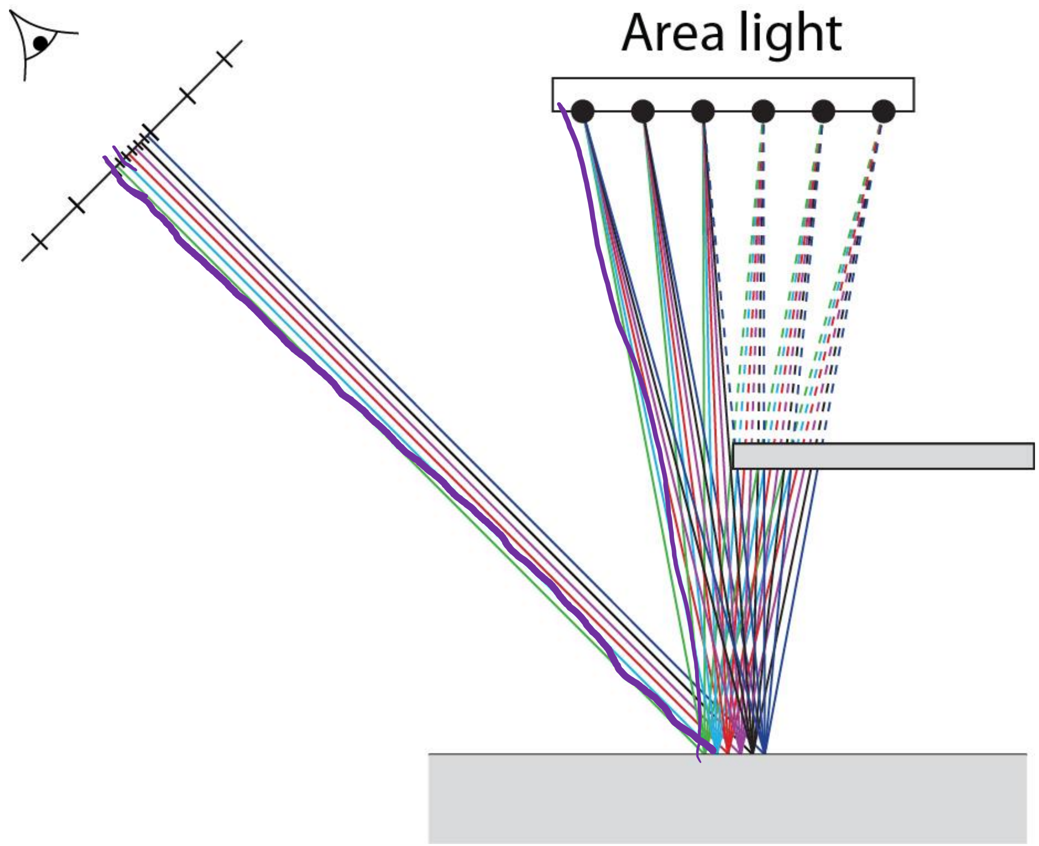
Penumbra revisited

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.

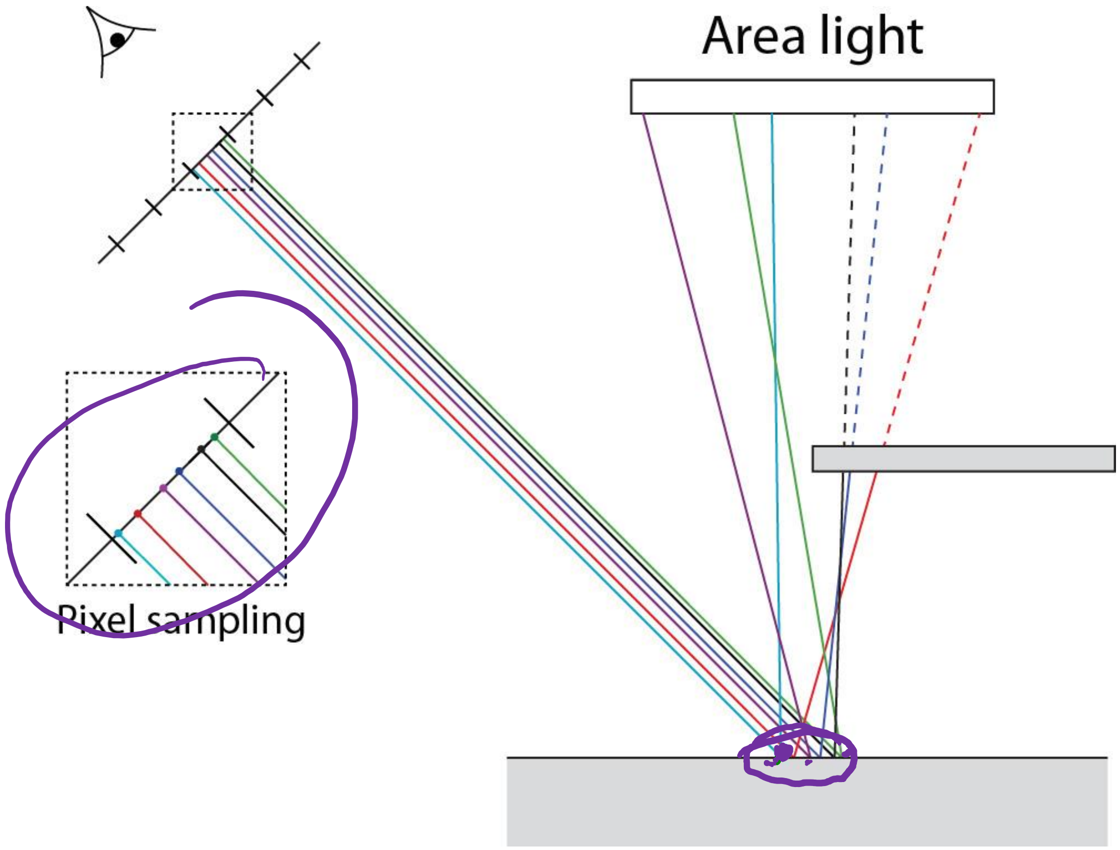
Penumbra revisited



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:

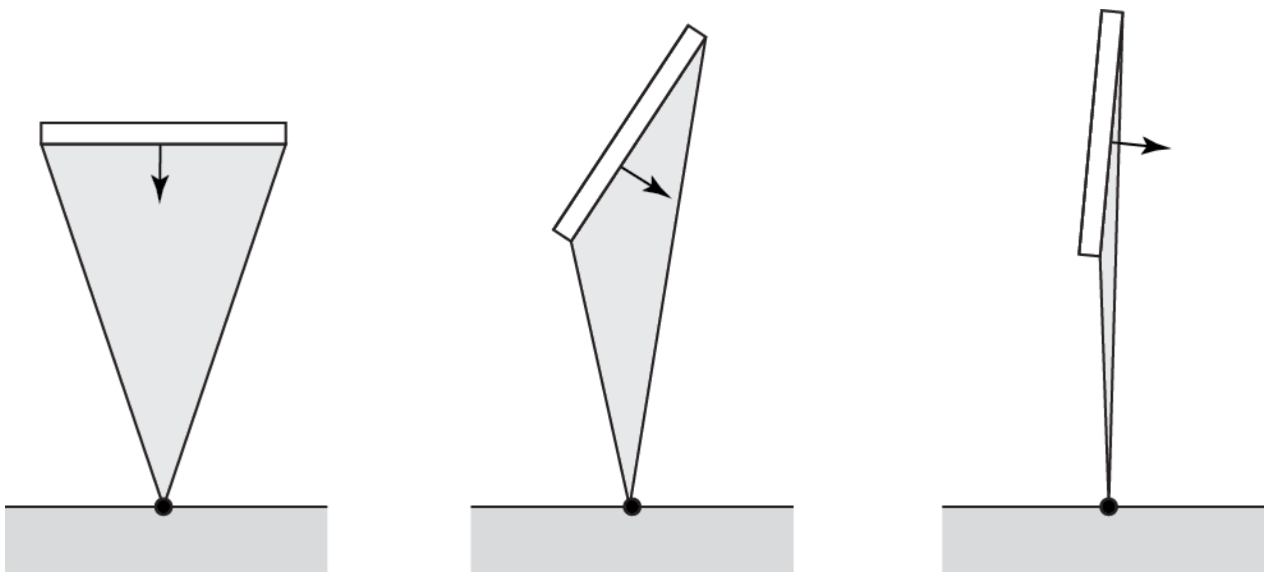
- Choose random location within a pixel, trace ray.
- At first intersection, choose random location on area light source and trace shadow ray.
- Continue recursion as with Whitted, but always choose random location on area light for shadow ray.

Orientation of an area light

One important detail for area lights…

As an area light tilts away away from a scene point, less of the light is “visible” to that scene point, which means that less light reaches the point.

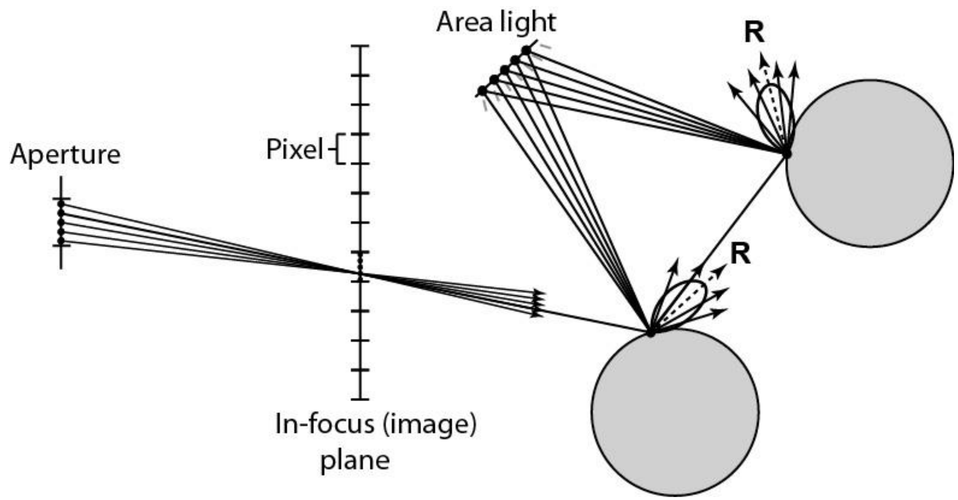
Thus, we attenuate the contribution of the light by the cosine of the angle of the light and the direction to a point on the surface.



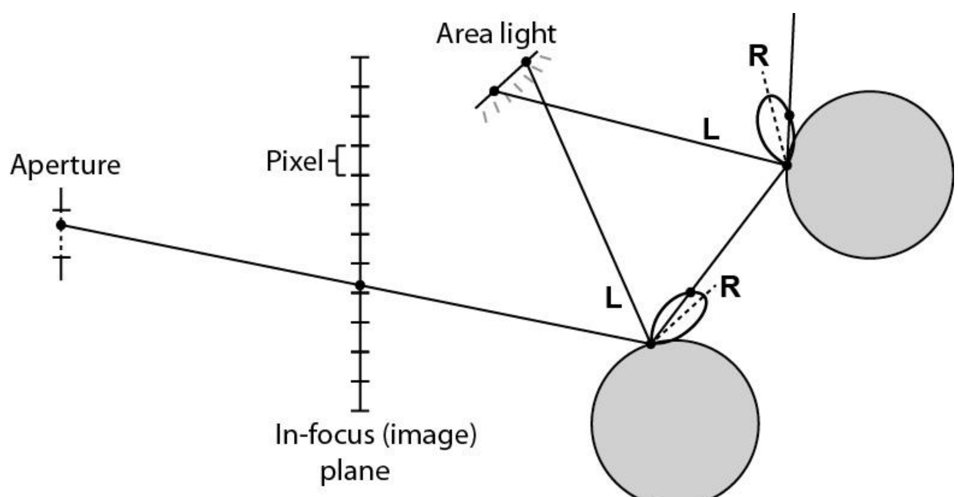
Monte Carlo Path Tracing vs. Brute Force

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

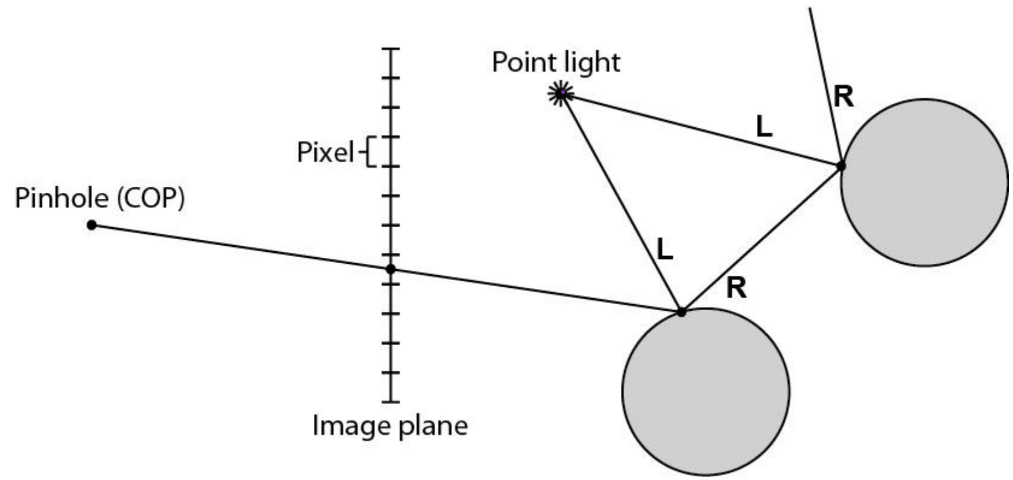


MCPT vs. Whitted

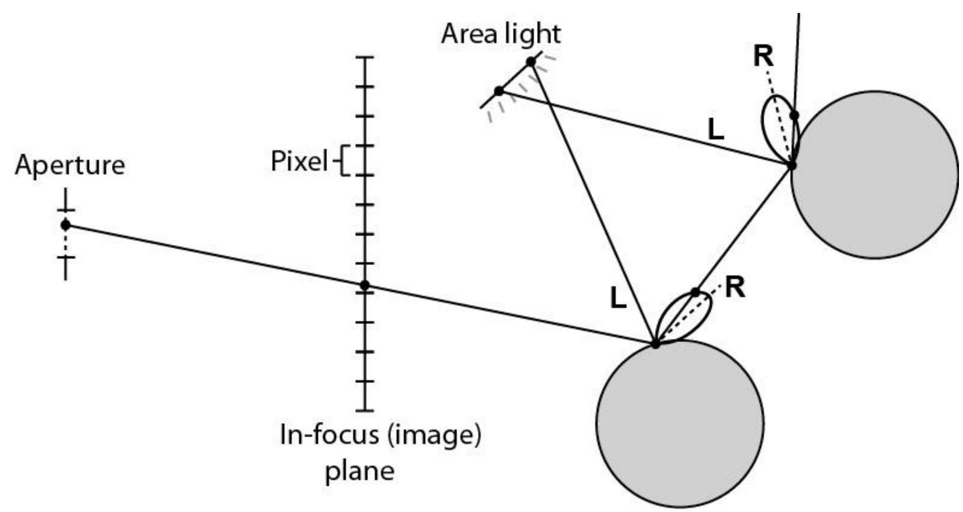
Q: For a fixed number of rays per pixel, does MCPT trace more total rays than Whitted?

Q: Does MCPT give the same answer every time?

Whitted ray tracing



Monte Carlo path tracing



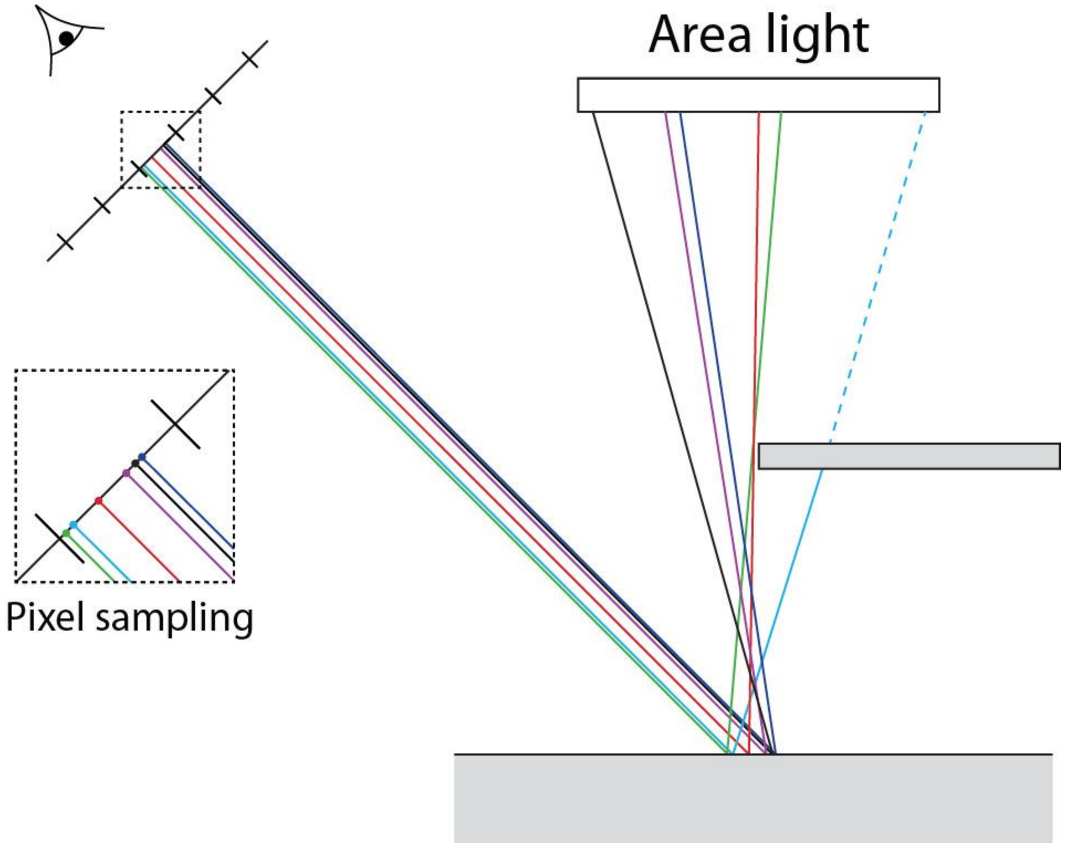
Ray tracing as integration

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

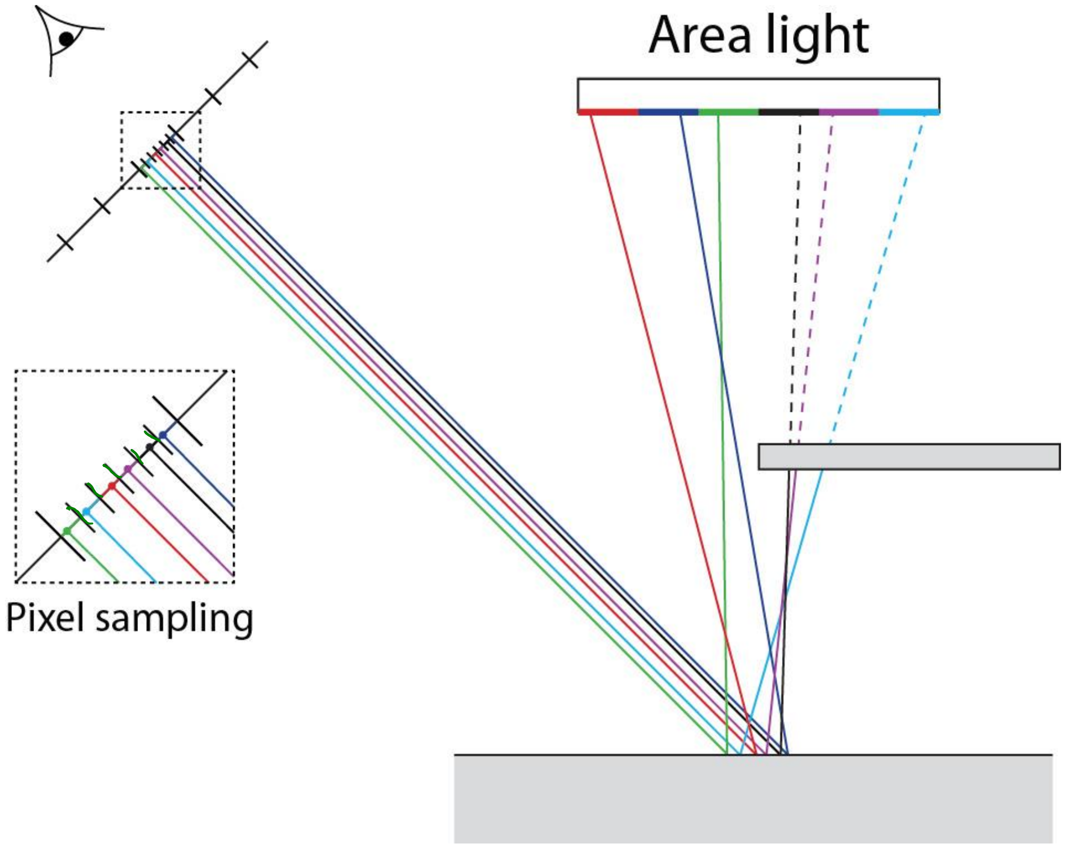
- the pixel area
- the aperture
- each light source
- all diffuse/glossy reflections (recursively)

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

Penumbra revisited: clumped samples



Penumbra: stratified sampling

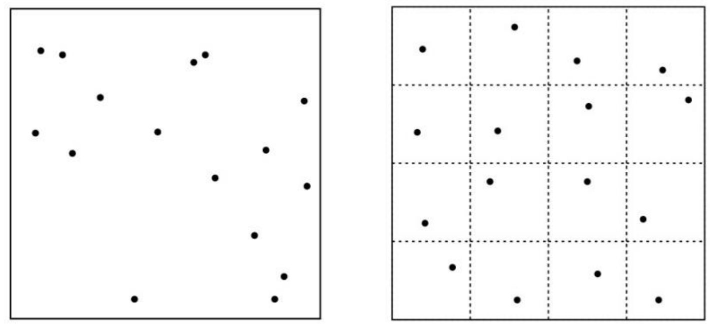


Stratified sampling gives a better distribution of samples:

- 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):

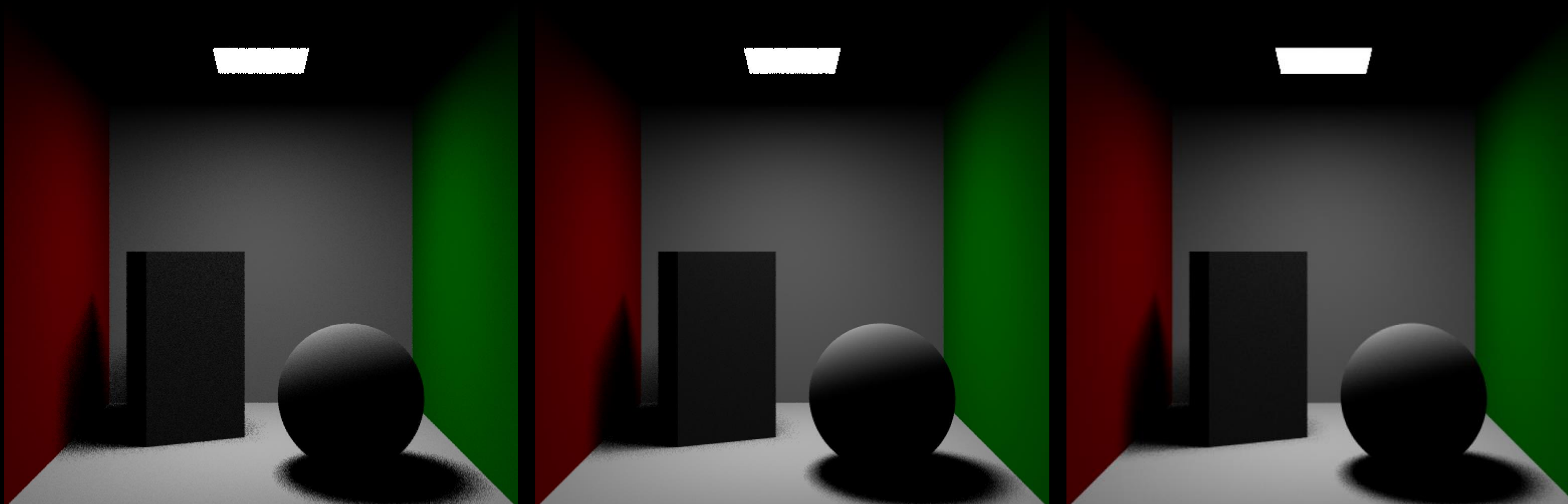


Random   Stratified

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.

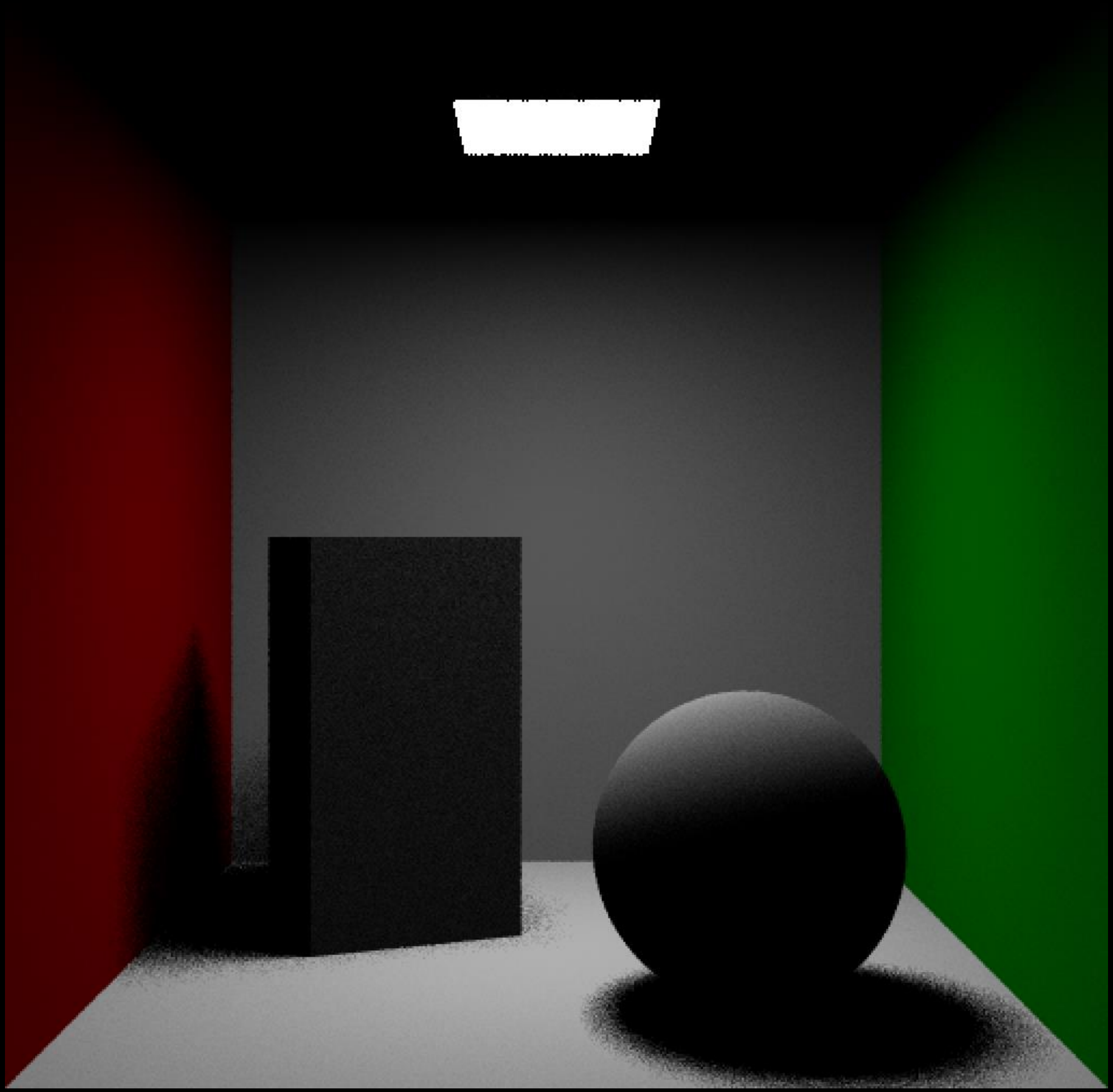
Stratified sampling of an area light



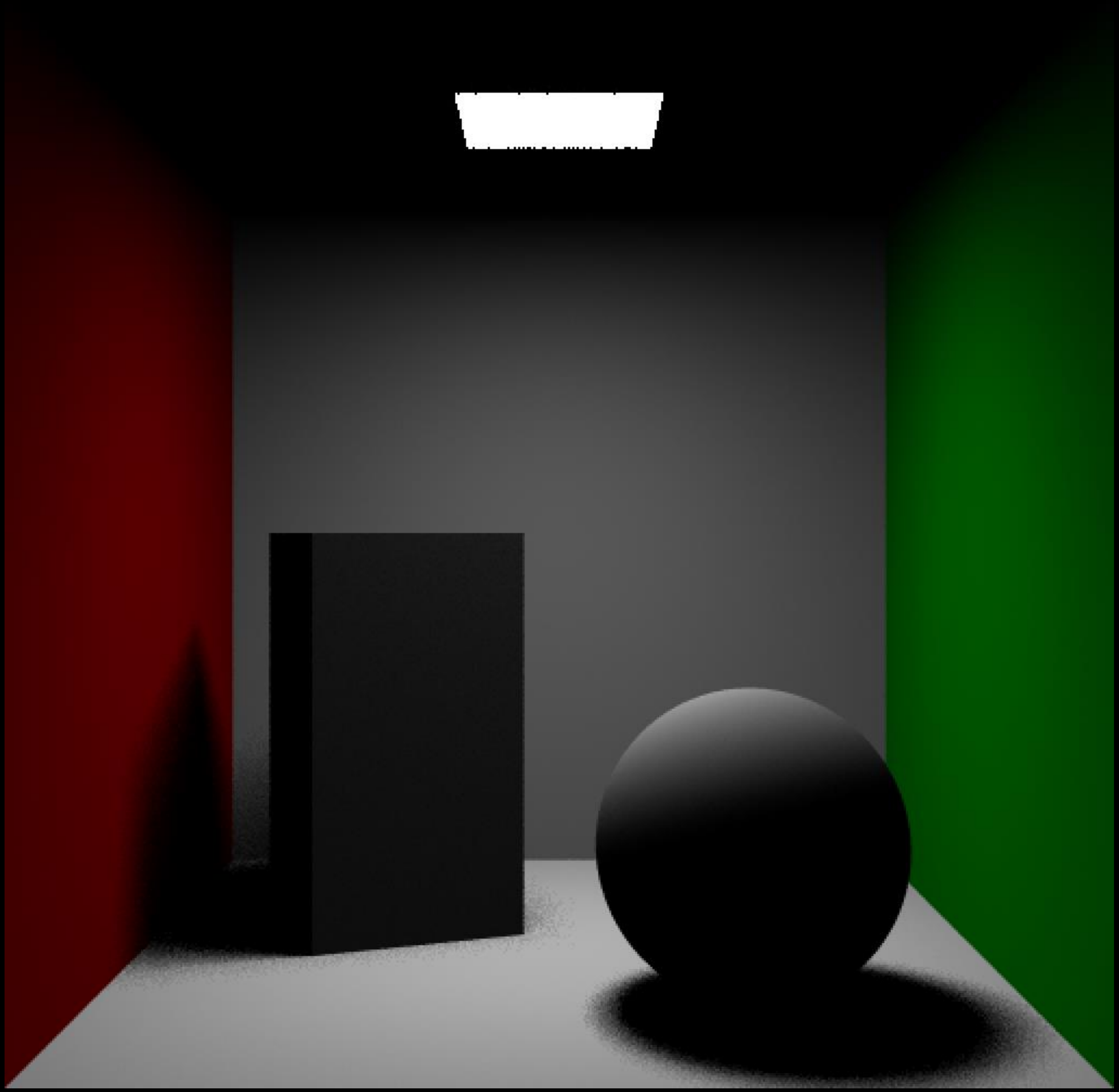
16 rays/pixel
uniform sampling

16 rays/pixel
stratified sampling

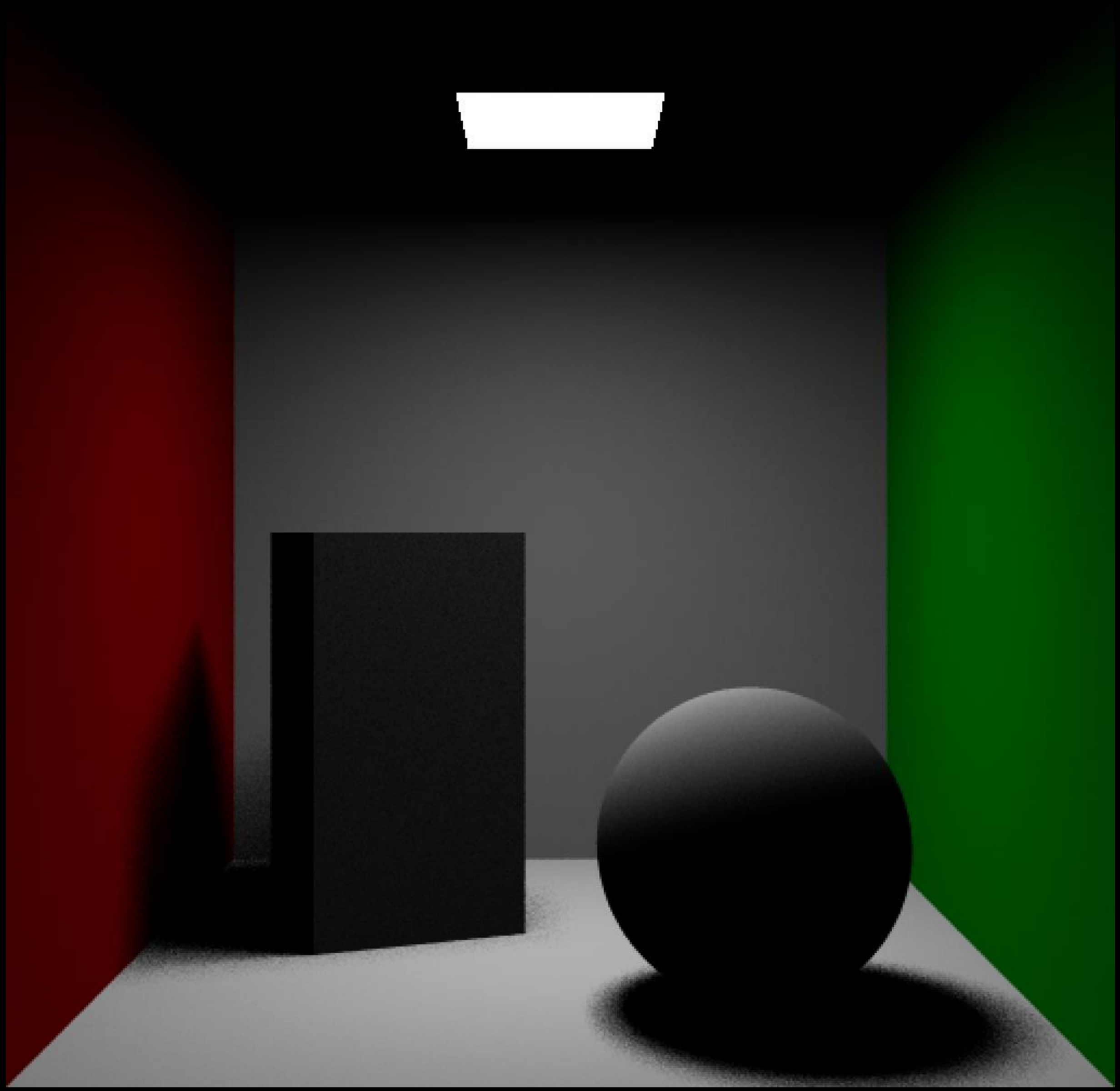
64 rays/pixel
uniform sampling



16 rays/pixel **uniform** sampling



16 rays/pixel **stratified** sampling



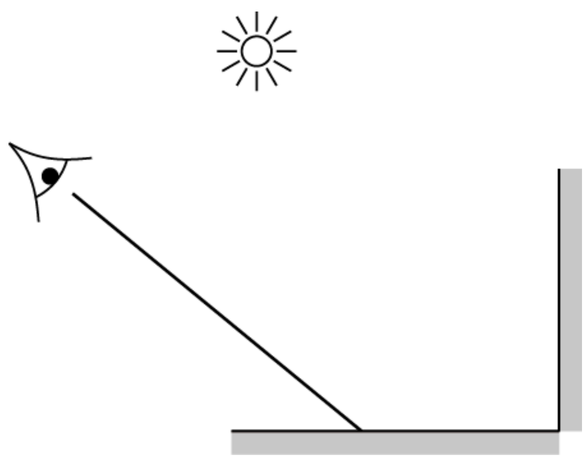
64 rays/pixel **uniform** sampling

Integration over reflection

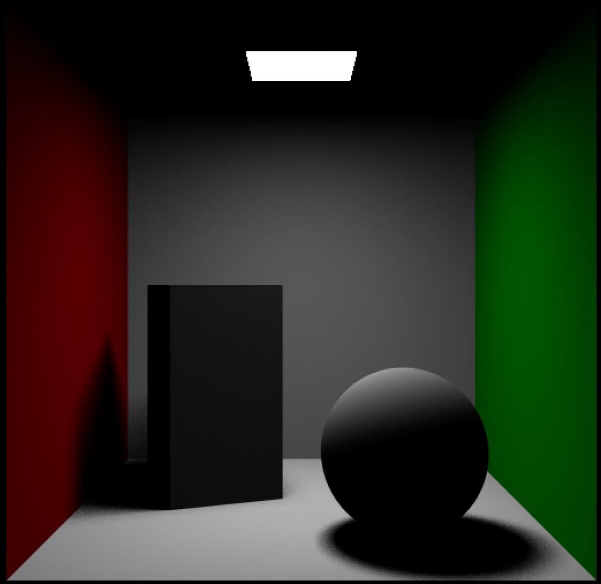
As described earlier, we can also reflect rays in directions away from ideal reflection.

An extreme case is diffuse reflection. The idea is that we:

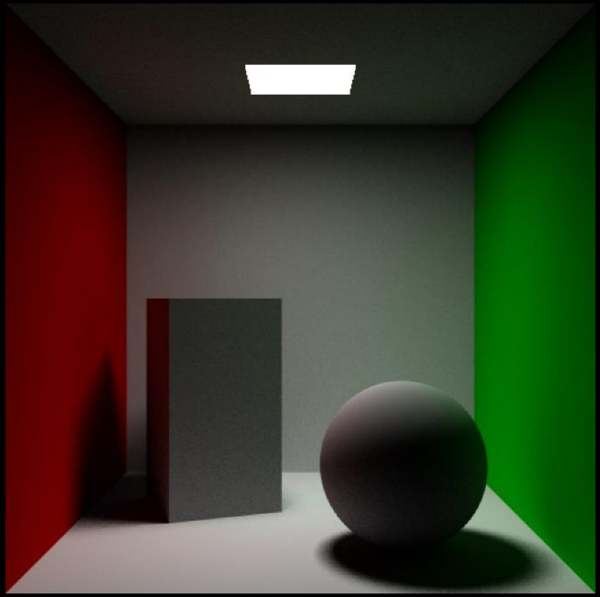
1. Hit a surface.
2. Choose a random direction for reflection.
3. Treat the returning ray value as a directional light.
4. Shade with that returned indirect light, as well as with direct lighting.
5. Return the result.



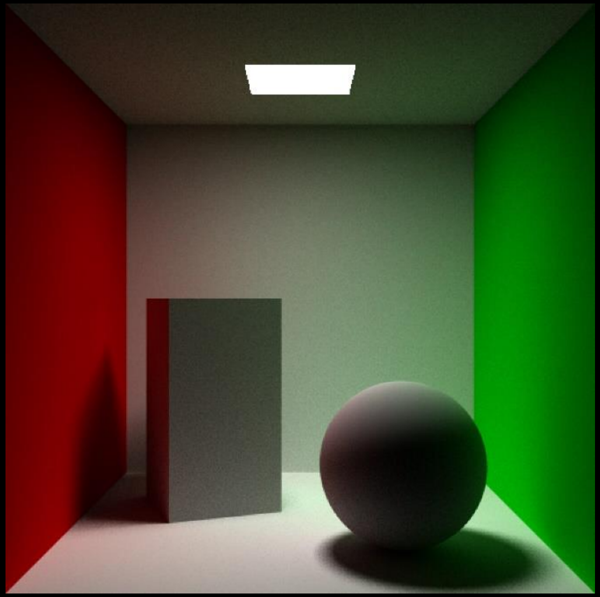
Diffuse interreflection



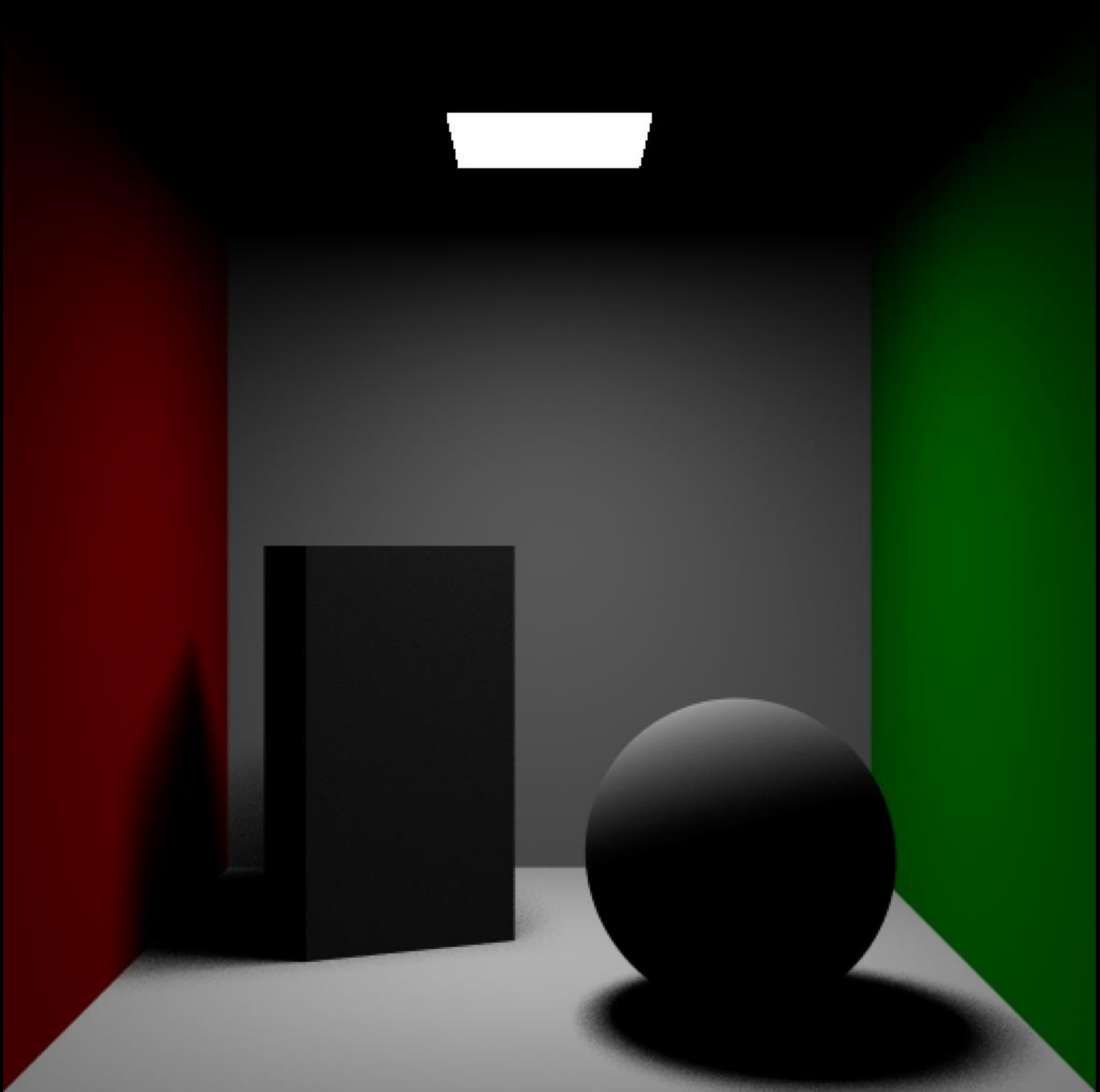
Depth 0

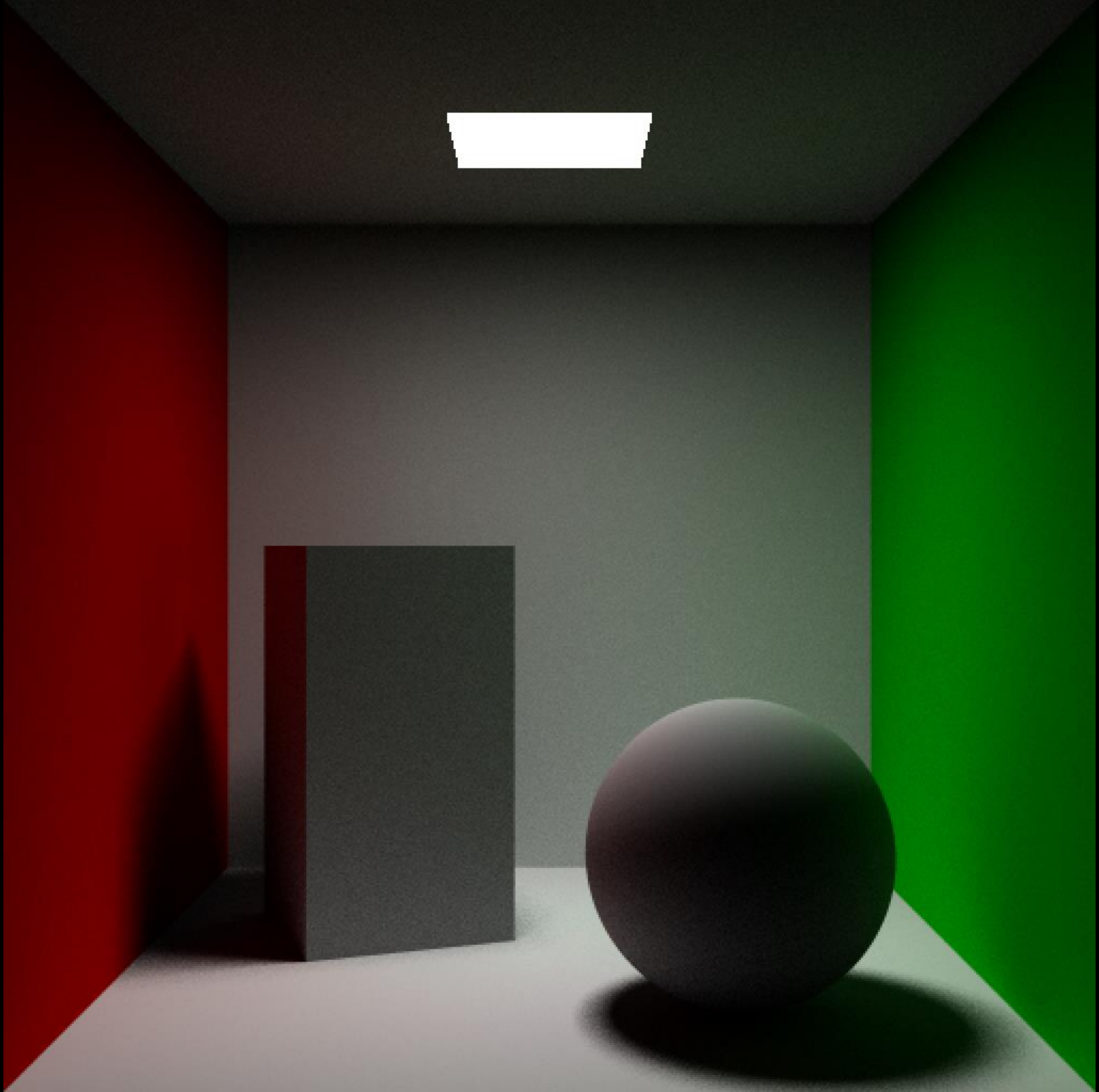


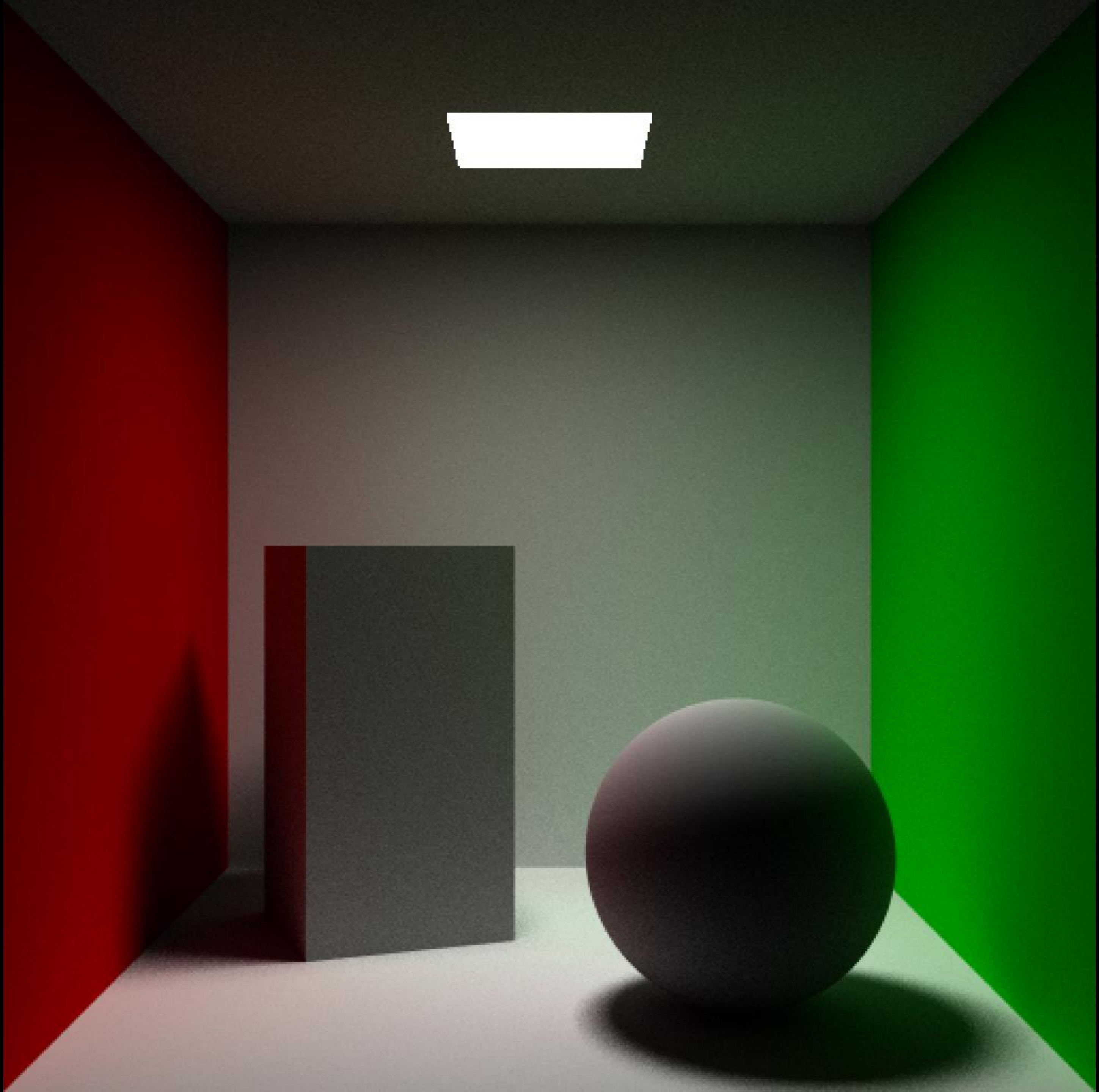
Depth 1

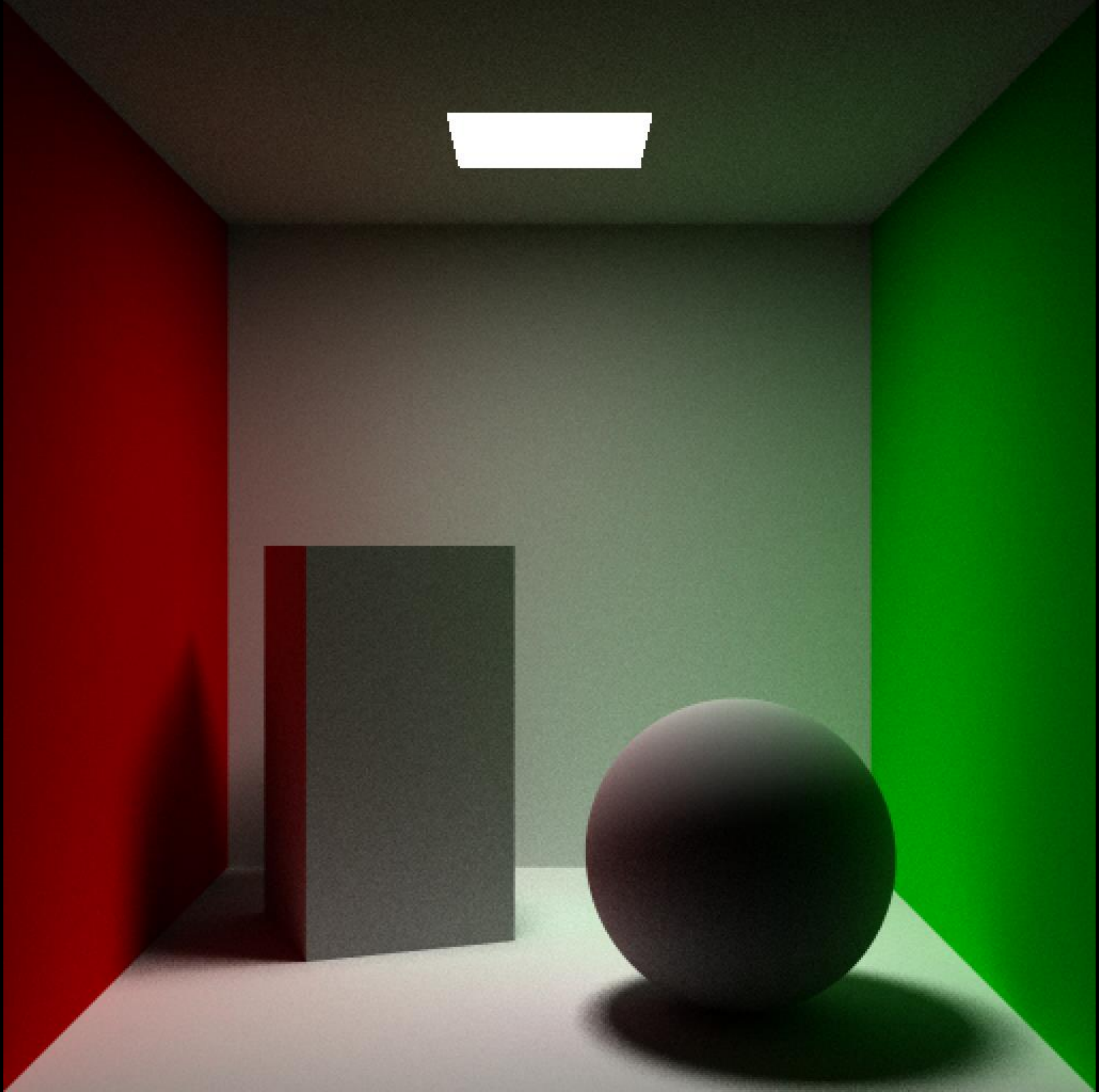


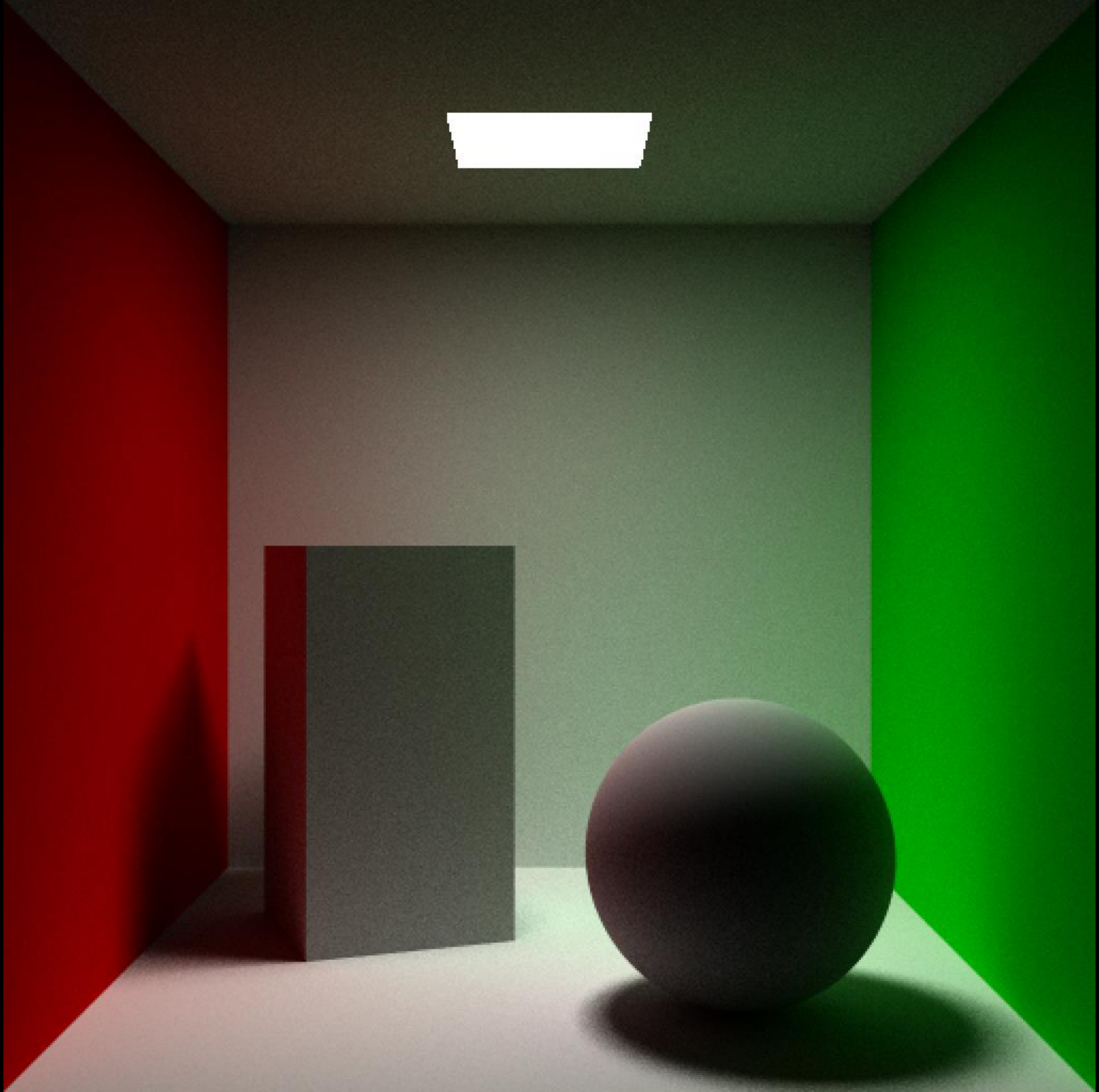
Depth 5

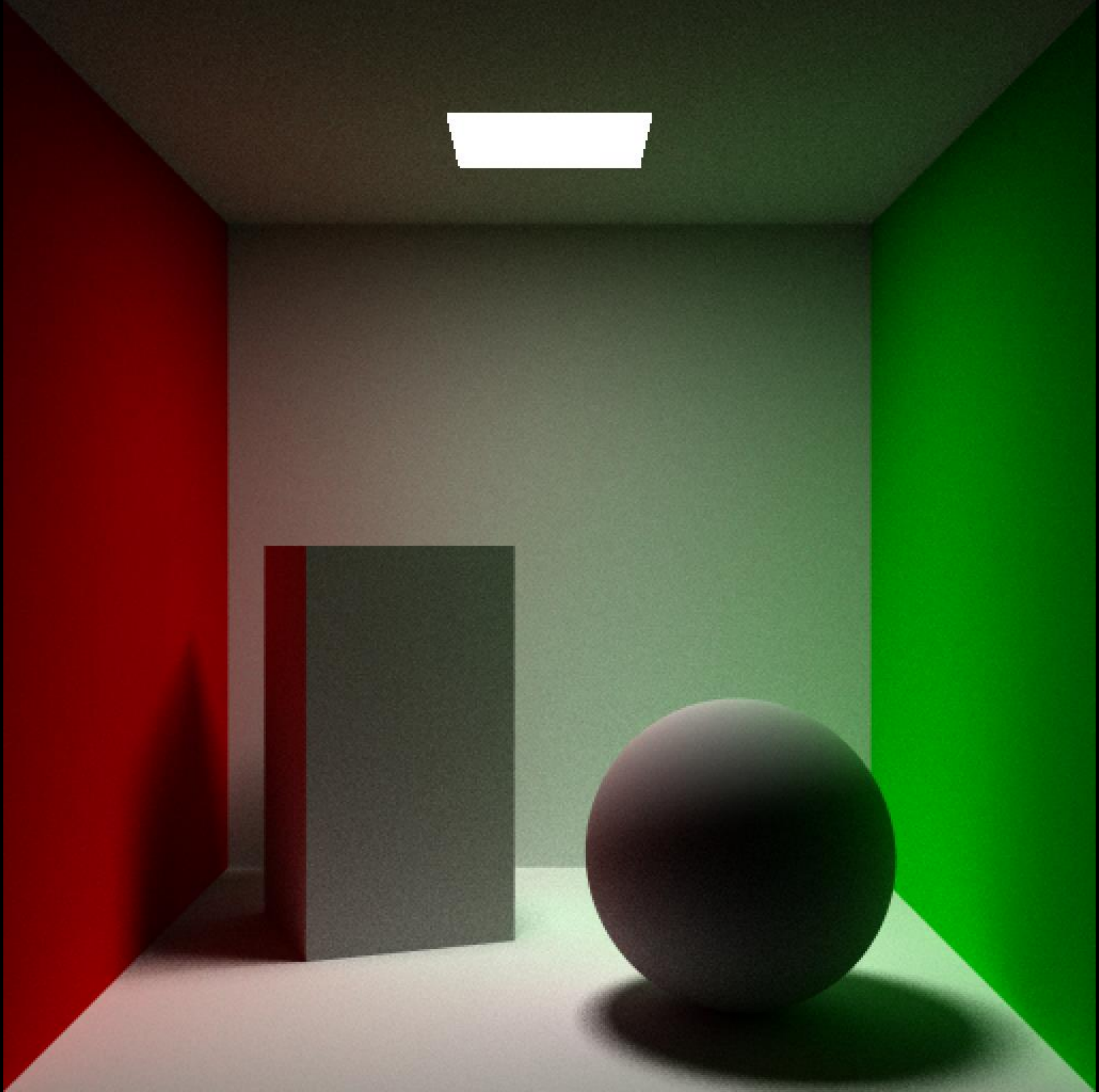








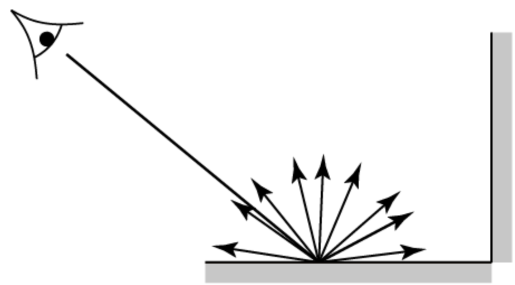




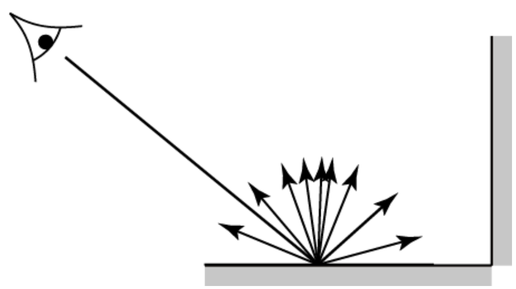
Importance sampling

Originally, we said we would choose a random direction for diffuse reflection. Whatever comes back, we will weight it by $\cos\theta$, where θ is the angle between the normal and the reflection direction.

Let's look at a bunch of uniformly random directions. Are they equally important?

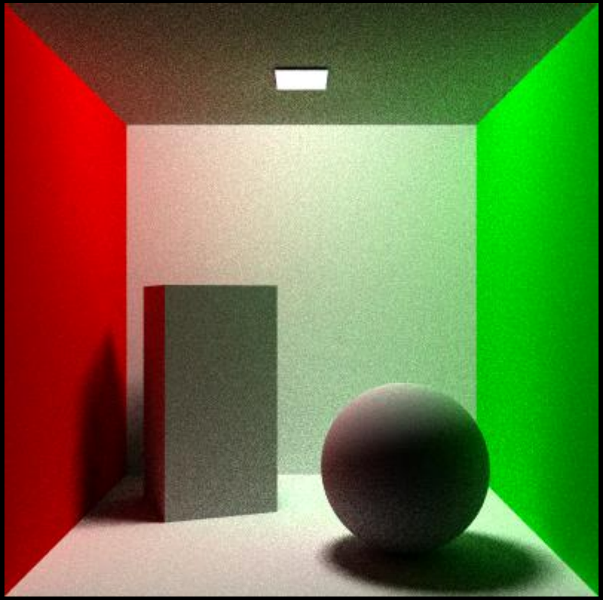


Instead, we could choose to reflect more rays in directions where $\cos\theta$ is greater, and fewer where it is smaller:

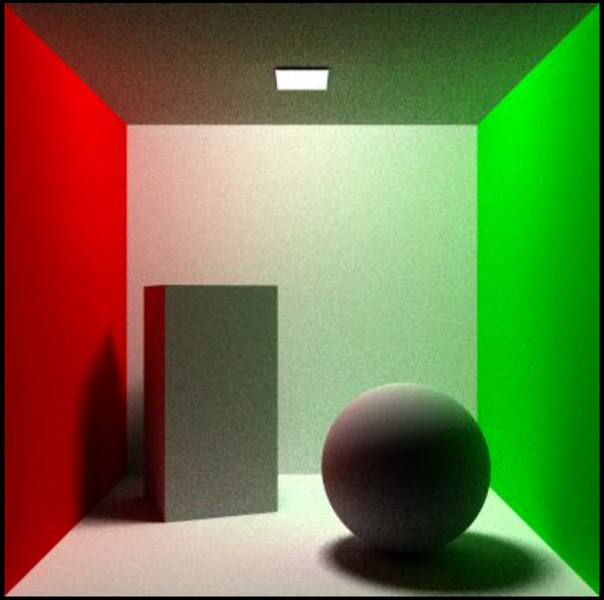


This is called importance sampling. In fact, if we choose the reflection direction q from a probability distribution function $p(\theta) \sim \cos(\theta)$, then we don't actually have to weight the rays at all!

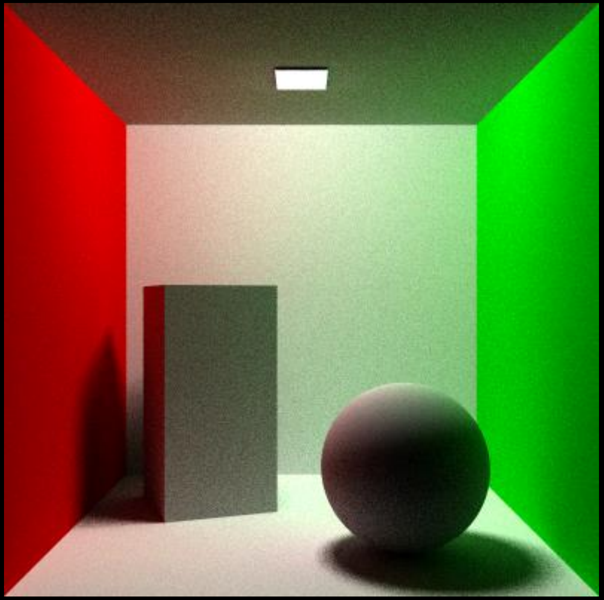
Importance sampling



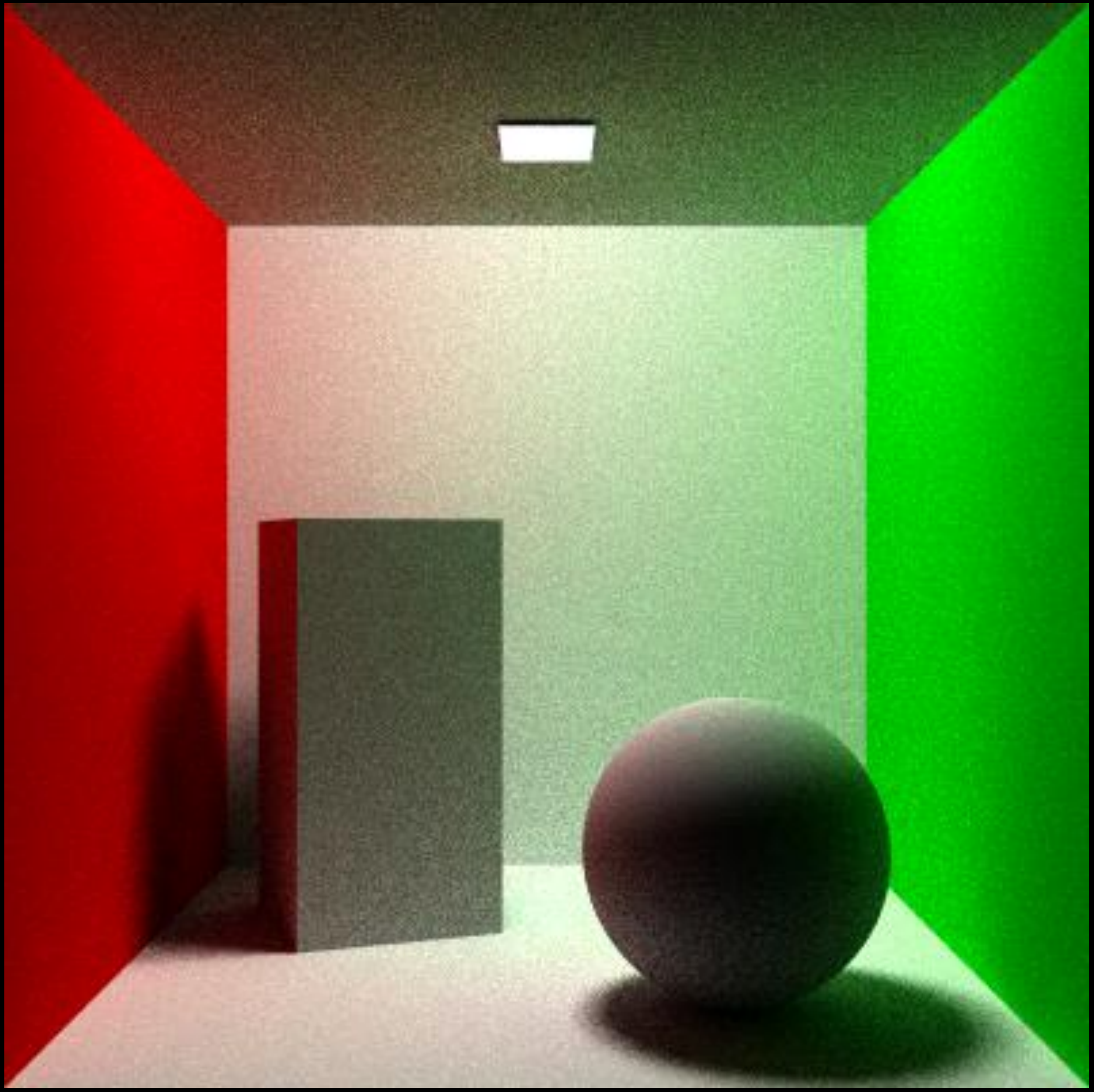
100 rays/pixel
without
importance sampling



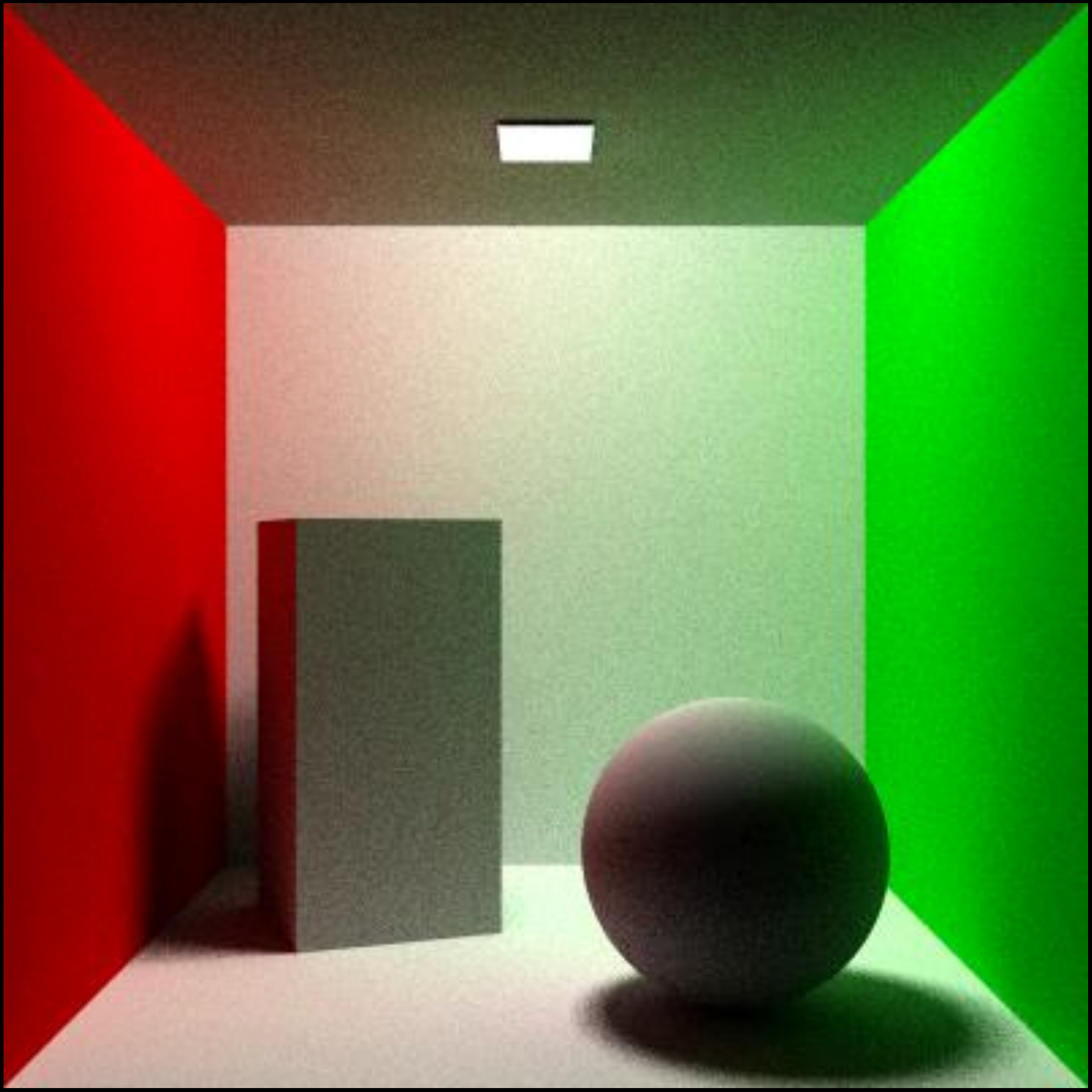
100 rays/pixel
with
importance sampling



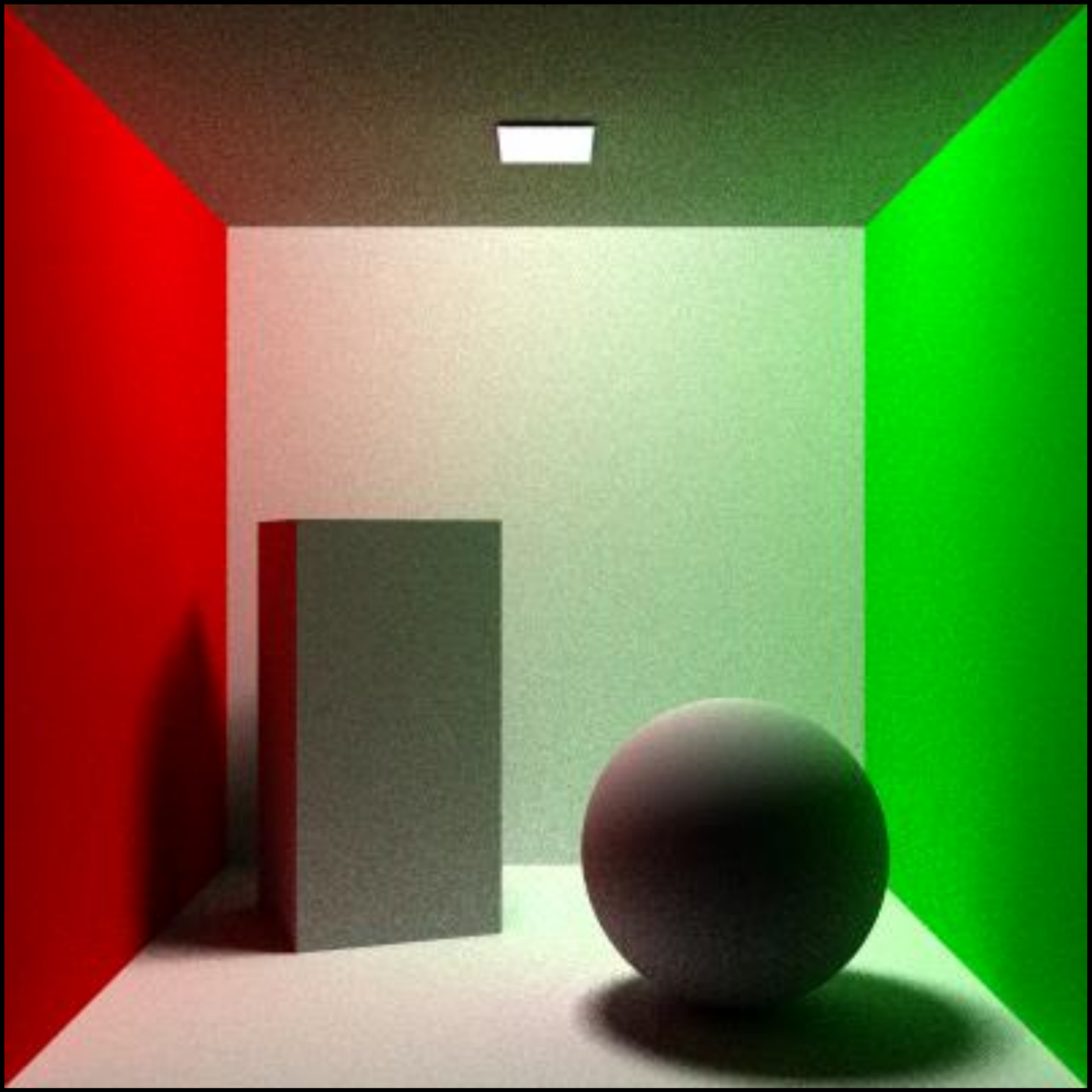
200 rays/pixel
without
importance sampling



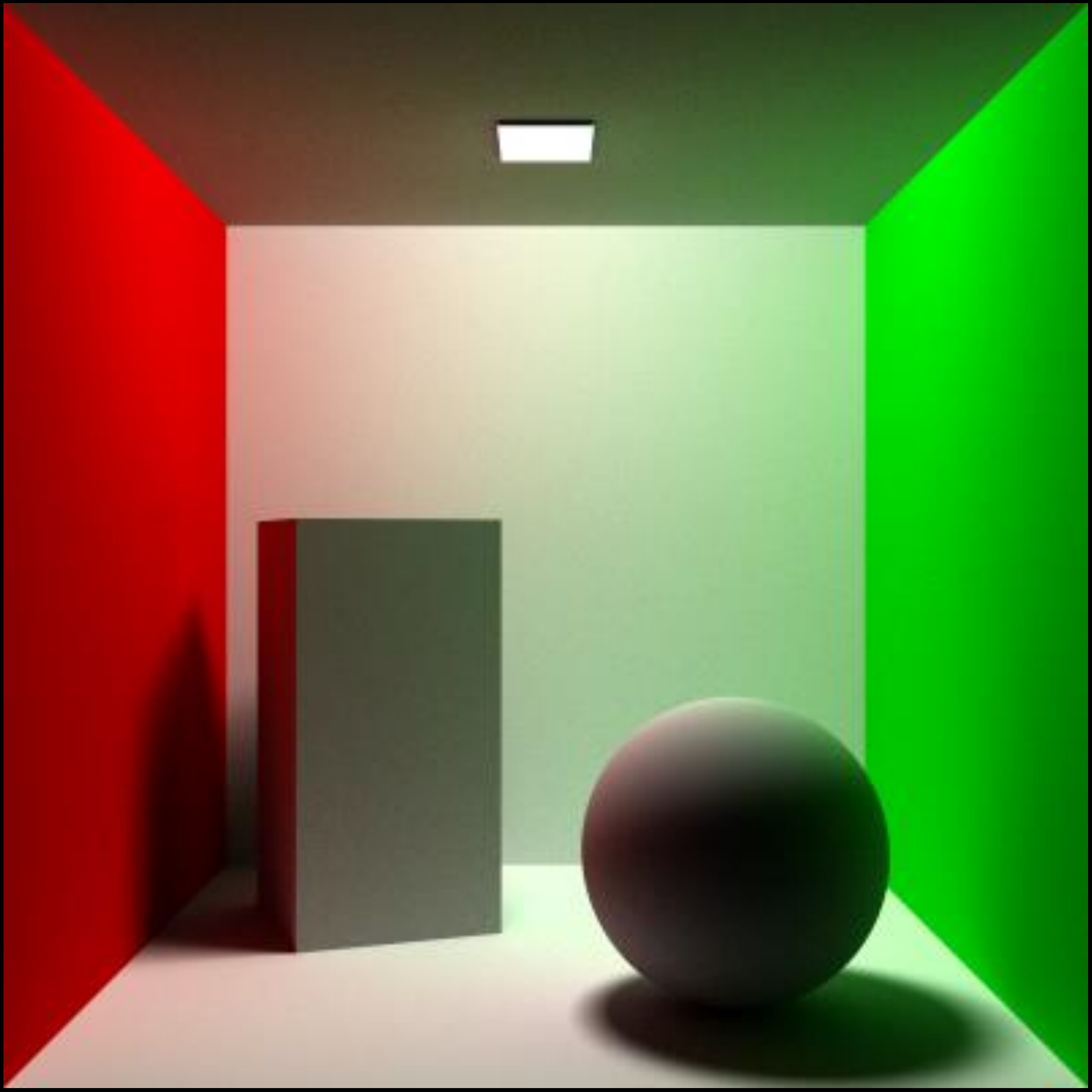
100 rays/pixel **without** importance sampling



100 rays/pixel **with** importance sampling



200 rays/pixel **without** importance sampling



900 rays/pixel **with** importance sampling

Another fancy render…

Area light, glossy and diffuse reflection, depth of field (1024 rays/pixel)

MCPT for beginners

If you want to try out MCPT, it is not as hard as you might think.

Try it with simple sampling strategies:

- Choose one of the effects you really like, add more if you have time.
- Skip stratification.
- If you do diffuse, don't do importance to begin with, just weight by the shading equation (normal dot with reflected ray direction).
- If you do glossy, you do need some kind of importance sampling.
    - Try simple perturbations around the reflection direction; the more random perturbation you allow, the blurrier the reflections.
    - The amount of perturbation is ideally determined by the specular exponent for Phong shading (bigger exponent means less perturbation).
- Throw a lot of rays per pixel!

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.)