Anti-aliasing and Monte Carlo Path Tracing

Brian Curless
CSE 557
Autumn 2017

Reading

Required:

- Marschner and Shirley, Section 13.4 (online handout)
- Pharr, Jakob, and Humphreys, Physically Based Ray Tracing: From Theory to Implementation, Chapter 13 (online handout)

Further reading:

- A. Glassner. An Introduction to Ray Tracing. Academic Press, 1989. [In the lab.]
- 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.

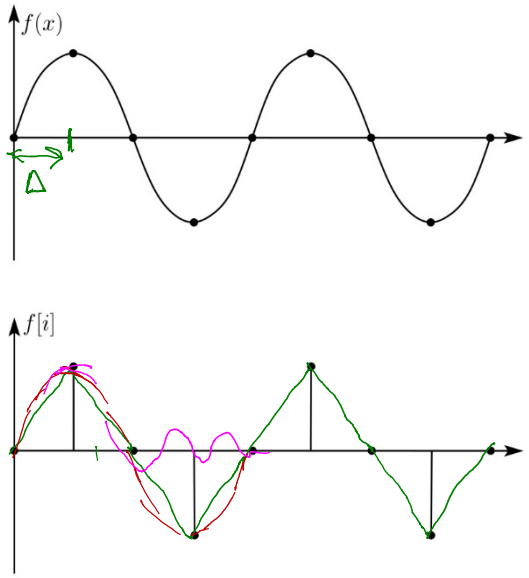
Aliasing

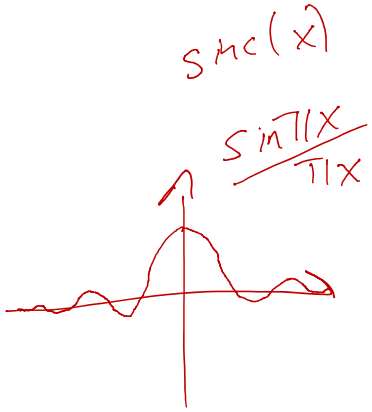
Ray tracing is a form of sampling and can suffer from annoying visual artifacts...

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:

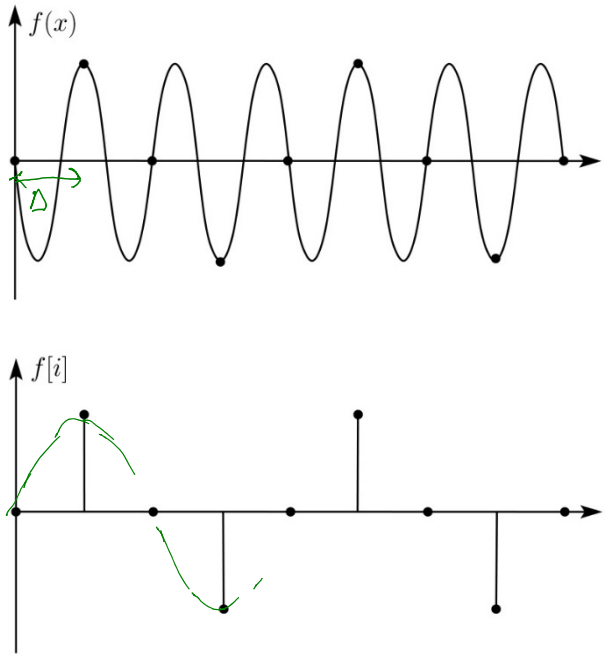




In this case, the sinusoid is reasonably well approximated by the samples.

Aliasing (con't)

Now consider sampling a higher frequency sinusoid

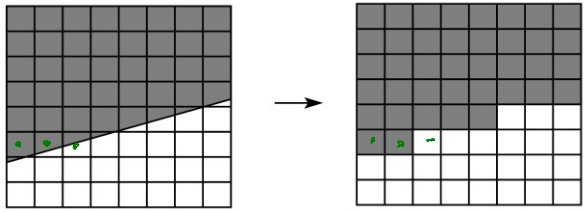


We get the exact same samples, so we seem to be approximating the first lower frequency sinusoid again.

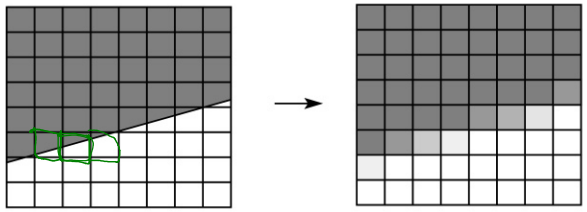
We say that, after sampling, the higher frequency sinusoid has taken on a new "alias", i.e., changed its identity to be a lower frequency sinusoid.

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:

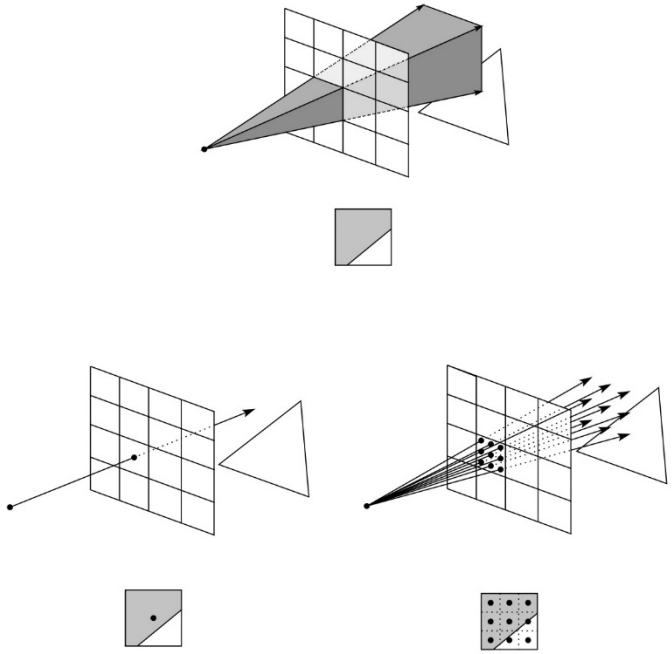


Anti-aliasing is the process of removing high frequencies *before* they cause aliasing.

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.

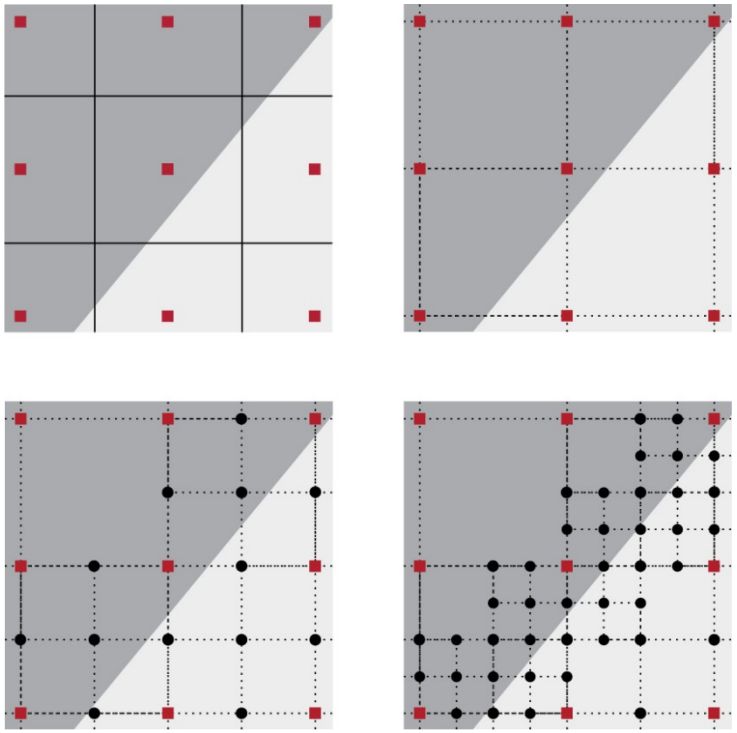
To improve matters, we can cast more than one ray per pixel and average the result.

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**.

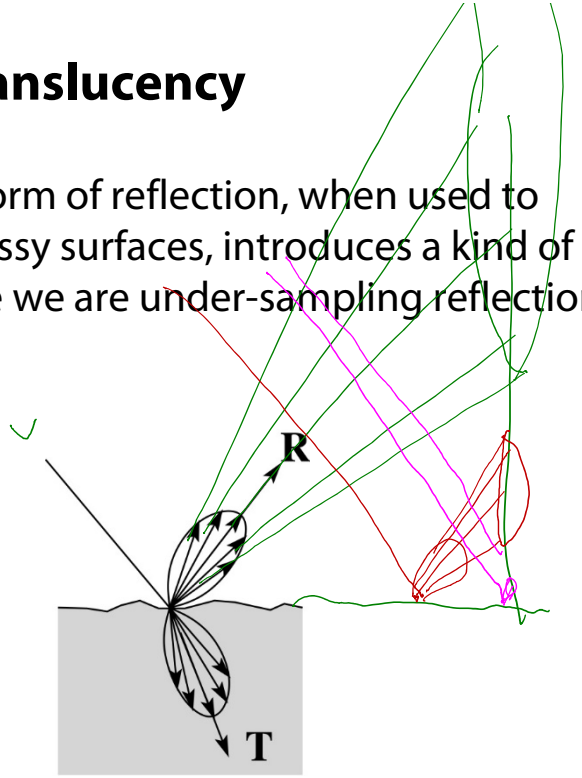


Q: When do we decide to cast more rays in a particular area?

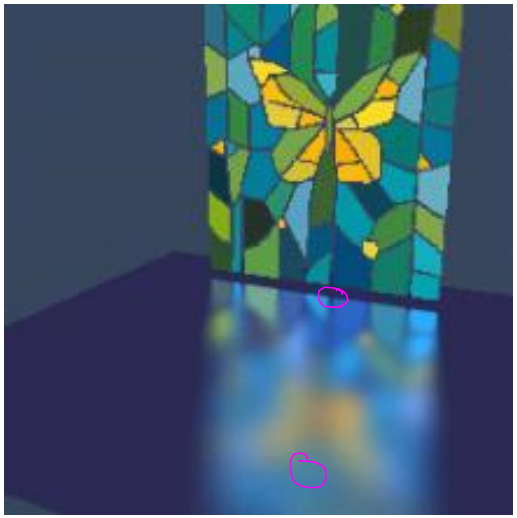
Gloss and translucency

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).

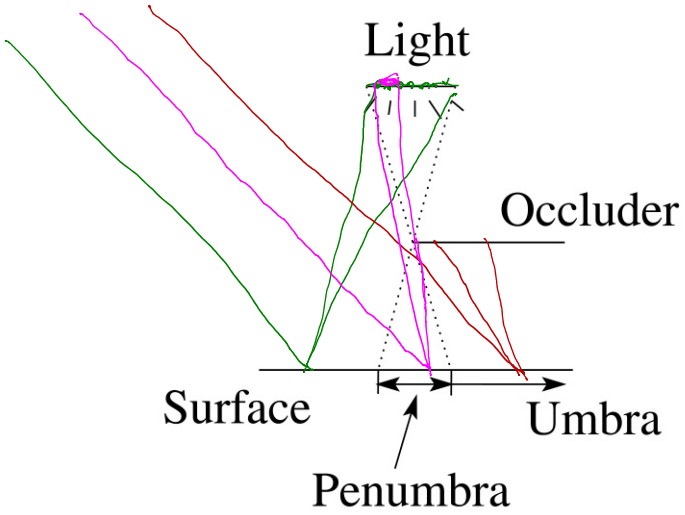
For example:

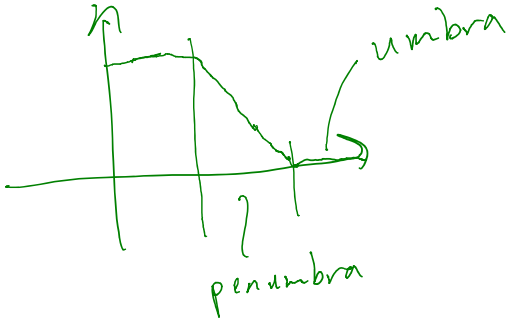


Distributing rays over reflection directions gives:

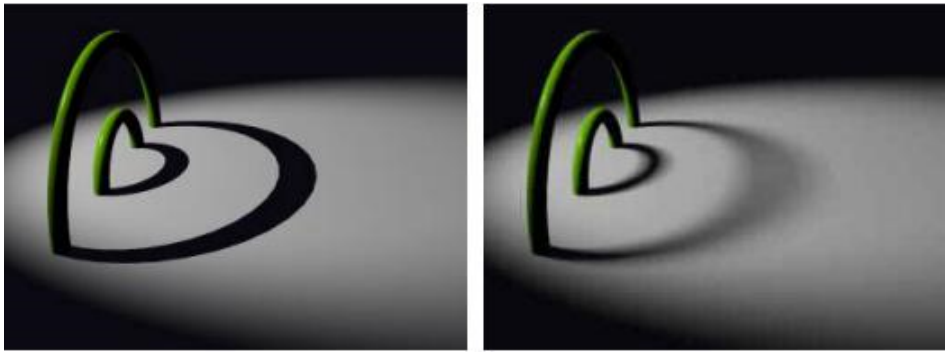


Soft shadows



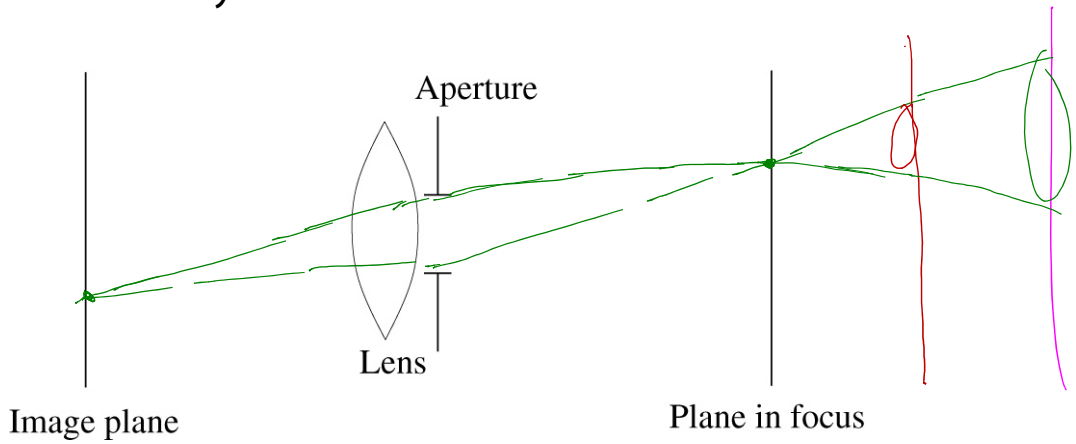


Distributing rays over light source area gives:



Depth of field

To simulate 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:

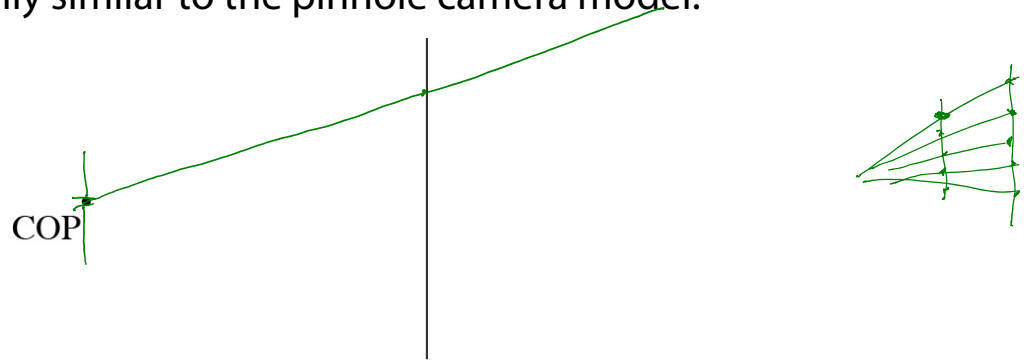


Image plane

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.

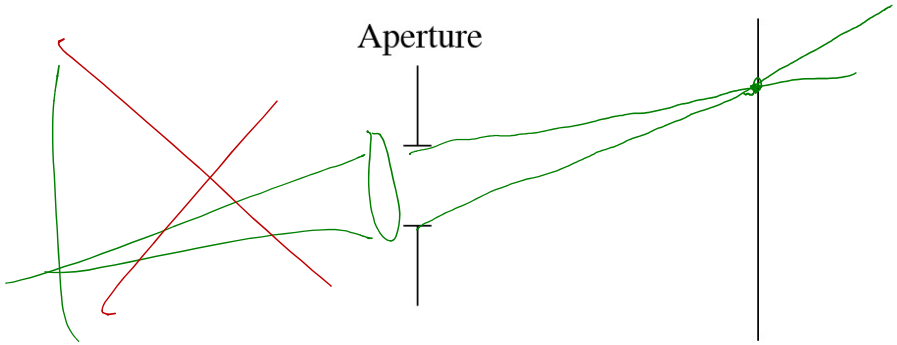


Image plane = plane in focus

Motion blur

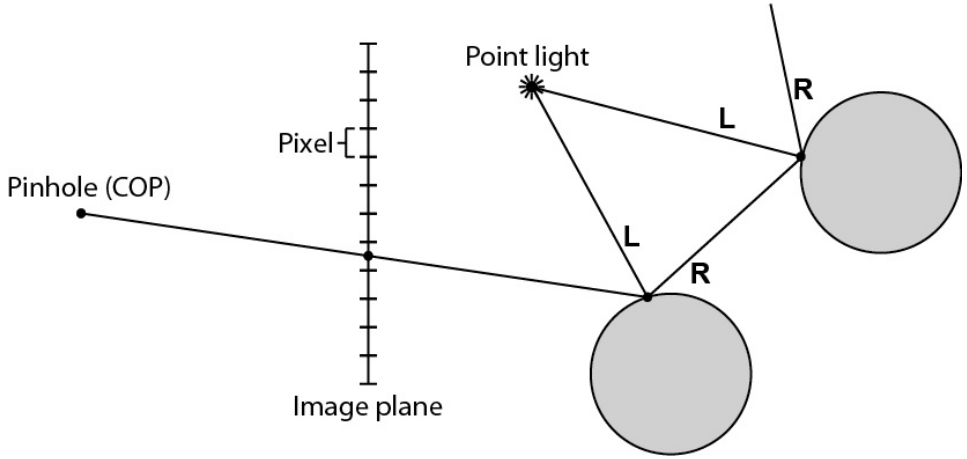
Distributing rays over time gives:

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

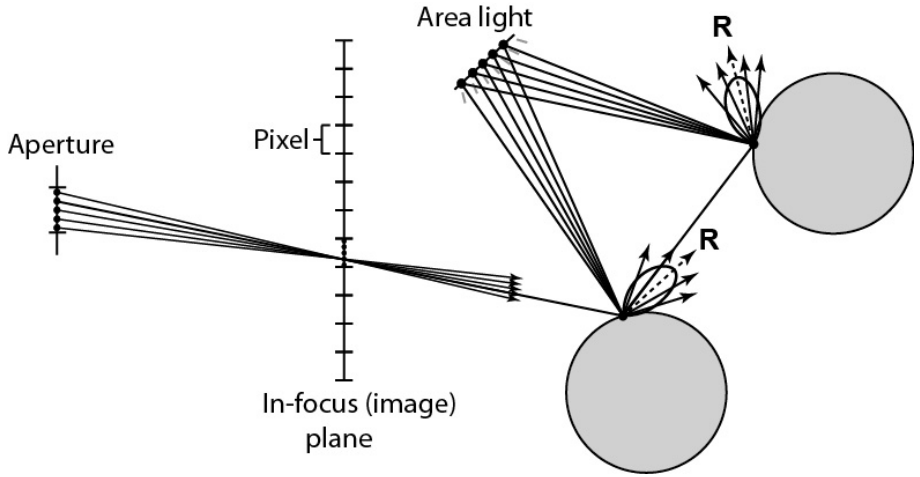
Naively improving Whitted ray tracing

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

Whitted ray tracing



Brute force, advanced ray tracing



Advanced ray tracing has:

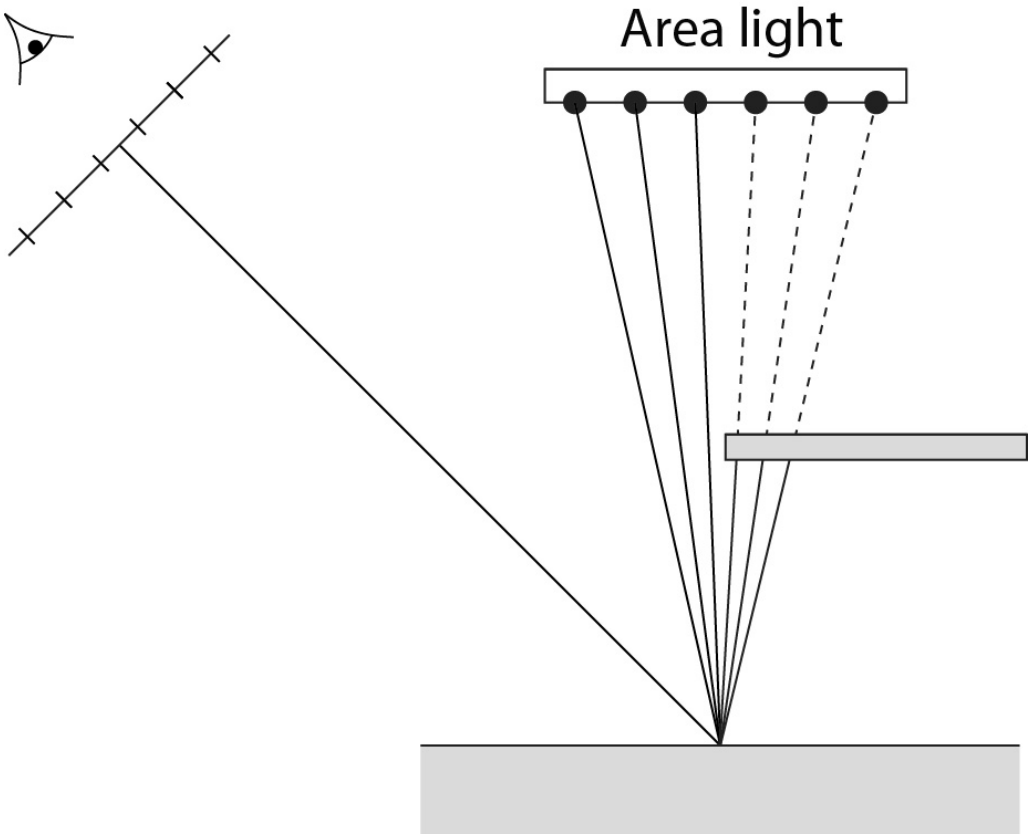
- m x m pixels
- k x k supersampling
- a x a sampling of camera aperture
- n primitives
- ℓ area light sources
- s x s sampling of each area light source
- r x r rays cast recursively per intersection (gloss/translucency)
- d is average ray path length

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

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

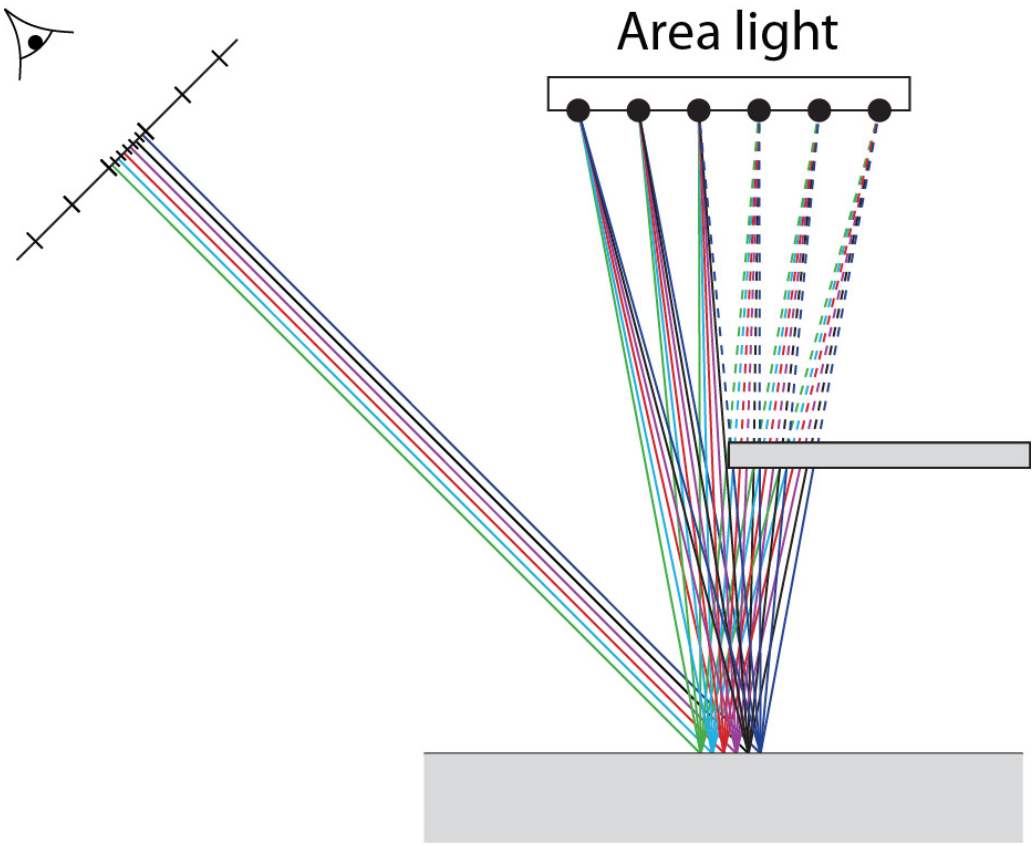
Penumbra revisited

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.

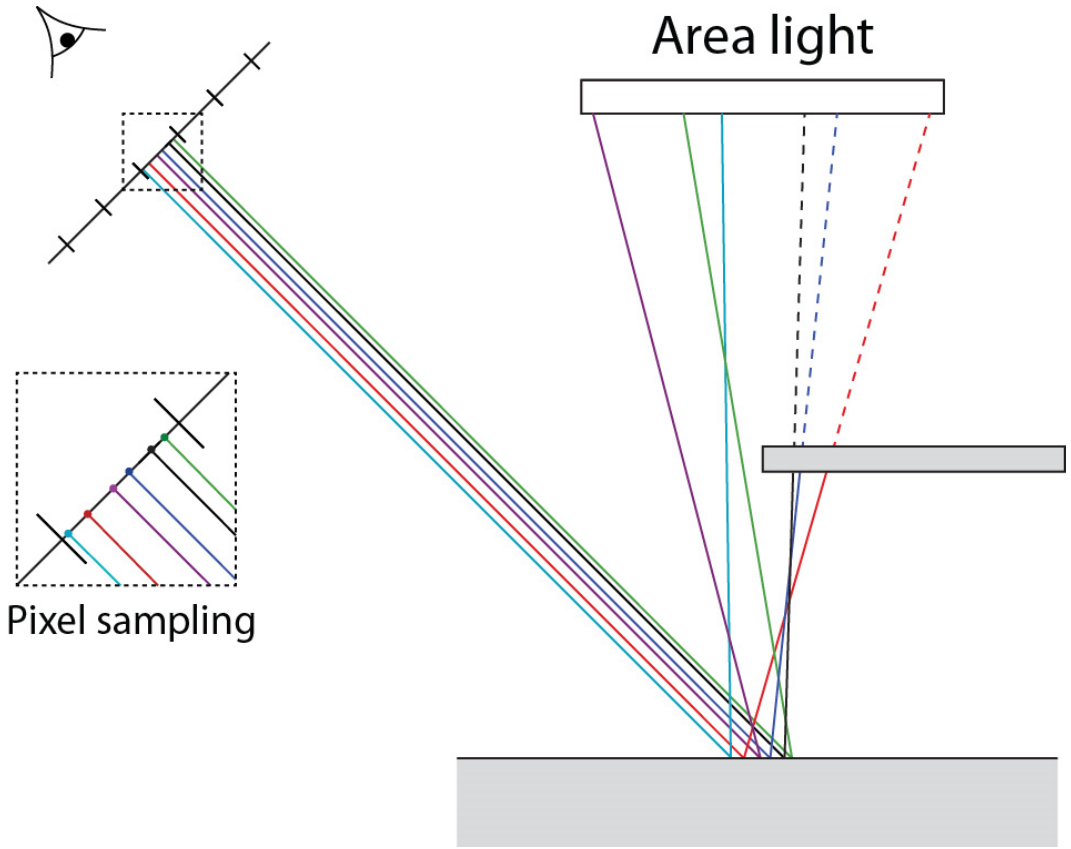
Penumbra revisited



We should anti-alias to get best looking results.

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

Penumbra revisited



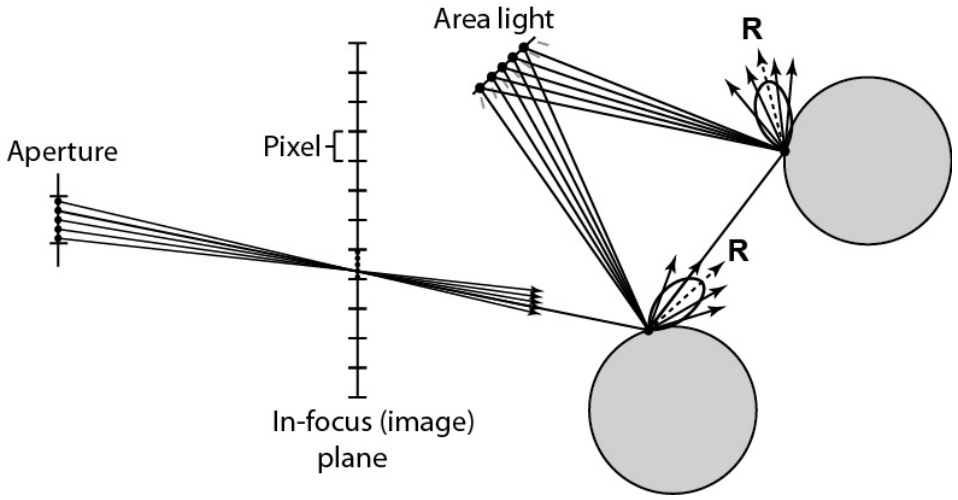
We can get a similar result with **much** less computation:

- 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.

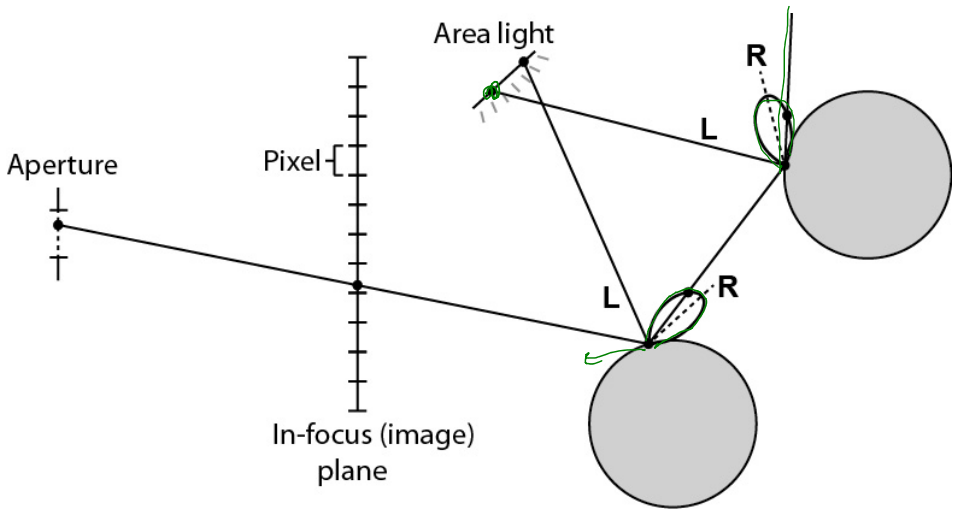
Monte Carlo Path Tracing vs. Brute Force

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

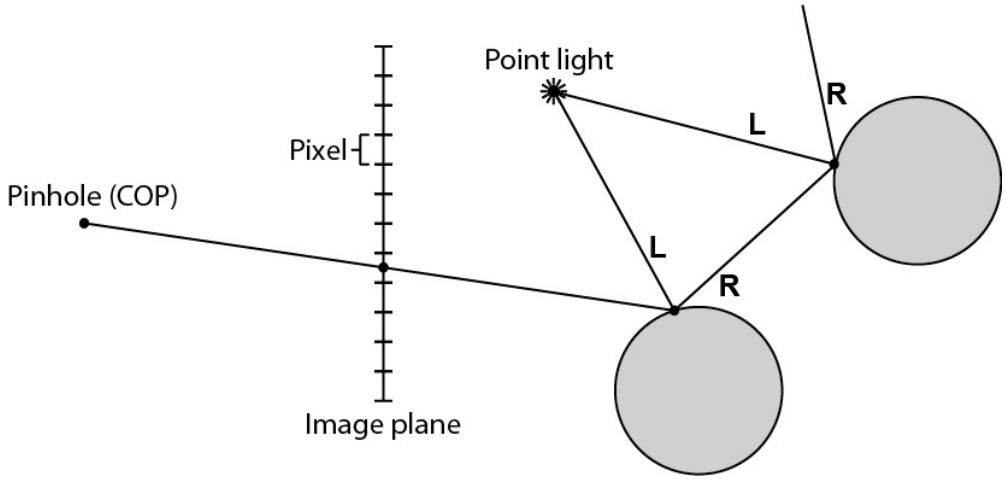


MCPT vs. Whitted

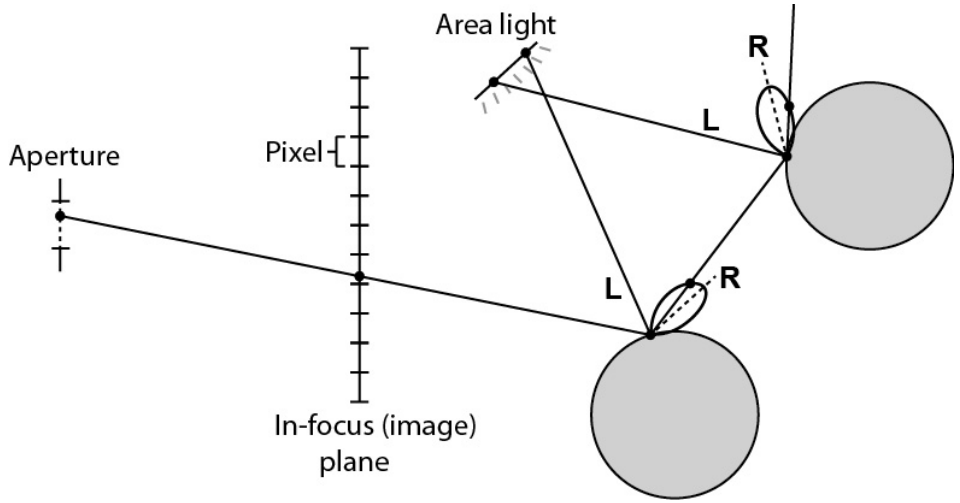
Q: For a fixed number of rays per pixel, does MCPT trace more total rays than Whitted?

Q: Does MCPT give the same answer every time?

Whitted ray tracing



Monte Carlo path tracing

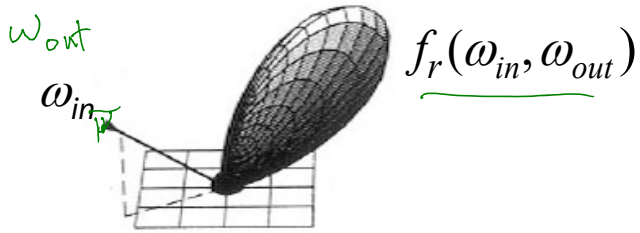


Ray tracing as integration

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)

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:



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

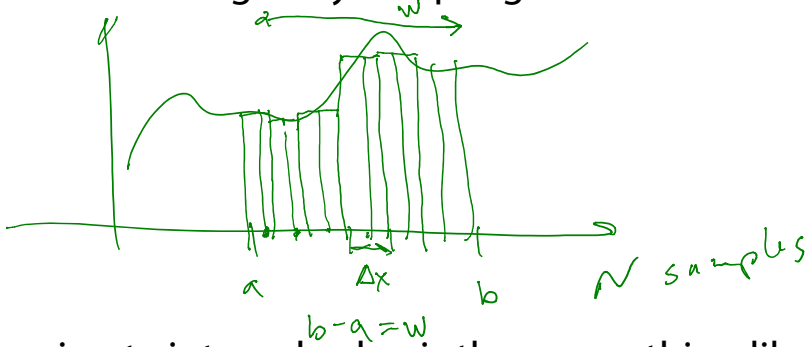
$$I(\omega_{out}) = \int_H I(\omega_{in}) f_r(\omega_{in}, \omega_{out}) (\omega_{in} \cdot \mathbf{N}) \, d\omega_{in}$$

Approximating integrals

Let's say we want to compute the integral of a function:

$$F = \int_a^b f(x)\,dx$$

If $f(x)$ is not known analytically (or is not easy to integrate), but can be readily evaluated, then we can approximate the integral by sampling.



$\Delta x = \frac{w}{N}$

Our approximate integral value is then something like:

$$F \approx \sum_{i=1}^{N} f(i\Delta x)\Delta x = \frac{w}{N} \sum_{i=1}^{N} f(i\Delta x)$$

where we have sampled N times at spacing Δx.

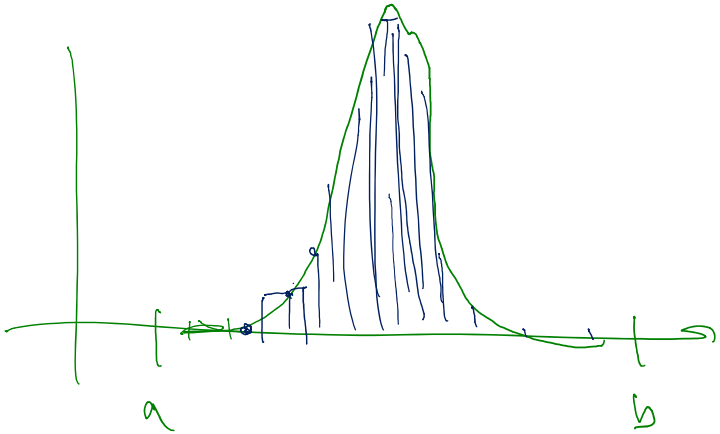
Evaluating an integral in this manner is called **quadrature**.

Q: How many samples do we need if we have integrals over d dimensions, and we sample N times in each dimension?

N^d

Approximating integrals (cont'd)

We can also do uneven sampling:



Our approximate integral value is then something like:

$$F \approx \sum_{i=1}^{N} f(x_i)\Delta x_i$$

where we have sampled N times at variable spacing Δx_i.

We can think of the Δx_i as weights on the samples.

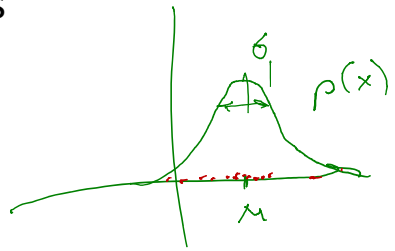
Q: Which x_i should we pick? where $f(x)$ is large

Q: When the x_i are more closely spaced, do they get larger or smaller weight?

A stochastic approach

A good approach to distributing samples unevenly across many dimensions is to do so **stochastically**.

Quick review first… For 1D, let's say the position in x is a random variable X, which is distributed according to $p(x)$, a probability density function (non-negative, integrates to unity). Then:



$$\mu = E[X] = \int_{-\infty}^{\infty} x\, p(x)\, dx$$

$$\sigma^2 = V[X] = E\left[(x - E[x])^2\right] = E\left[(x-\mu)^2\right]$$

Suppose we don't know $p(x)$, but we can collect N samples $\{X_i\}$ drawn from $p(x)$.

How would we estimate the mean from the samples?

$$\frac{1}{N}\sum x_i$$

Suppose we repeated the process, drawing N samples and re-computing the mean. Will we get the exact same answer?

Sample mean

This sample mean is itself a new random variable:

$$\frac{1}{N}\sum_{i=1}^{N} X_i$$

where each of the X_i are independent with pdf $p(x)$.

Ideally, the expected value of this random variable actually is the expected value of the original one, X.

It's easy to show that:

$$E[cX] = cE[X]$$

$$E[X+Y] = E[X] + E[Y]$$

where X and Y are **independent** random variables.

The $\{X_i\}$ are **independent and identically distributed** (**i.i.d.**) leading to expected sample mean:

$$E\left[\frac{1}{N}\sum_{i=1}^{N} X_i\right] = \frac{1}{N} E\left[\sum X_i\right] = \frac{1}{N}\sum_{i=1}^{N} E[X_i] = \frac{1}{N} \cdot N E[X] = E[X]$$

(where $E[X_i] = E[X]$)

Sample variance

We noted that the sample mean does not give the same answer every time – it's a random variable.

How much does the answer vary? We'll compute its variance.

It's easy to show that:

$$V[cX] = c^2 \, \mathrm{V}[X]$$

$$V[X+Y] = V[X] + V[Y]$$

where X and Y are independent random variables.

For i.i.d. variables $\{X_i\}$, we can now compute the variance of the sample mean:

$$V\left[\frac{1}{N}\sum_{i=1}^{N} X_i\right] = \frac{1}{N^2} V\left[\sum x_i\right] = \frac{1}{N} \sum_{i=1}^{N} V[x_i] = \frac{1}{N} \cdot N \, V[x] = \frac{1}{N} V[x]$$

$V[x]$

What parameter can we control to reduce the variance?

N

How does standard deviation change when adjusting that parameter?

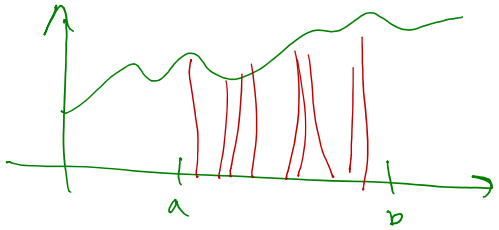
$\frac{1}{\sqrt{N}}$

Stochastic sampling as integration

Let's return to the integral we wish to compute:

$$F = \int_a^b f(x)\,dx$$

Suppose now we use a uniform pdf to sample the range $[a, b]$:



We can then compute the mean of these samples:

$$\frac{1}{N}\sum_{i=1}^{N} f\left(X_i\right)$$

Note that a function of a random variable is itself a random variable.

So, we have created yet another random variable, which is an average of random variables $\{ f(X_i) \}$.

What is its expected value…?

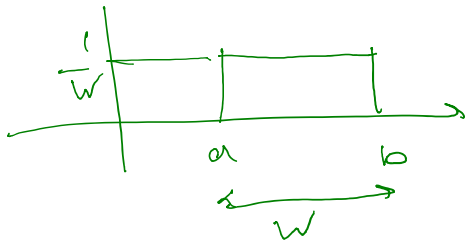
Stochastic sampling as integration (cont'd)

We can compute the expected value as:

$$E\left[\frac{1}{N}\sum_{i=1}^{N} f\left(X_i\right)\right] =$$

But what is the expected value of $f(X)$ where X is drawn from a uniform pdf over $[a, b]$:

$$p(x) = \begin{cases} \frac{1}{w} & \text{if } a \le x \le b \\ 0 & \text{else} \end{cases}$$



Then we have:

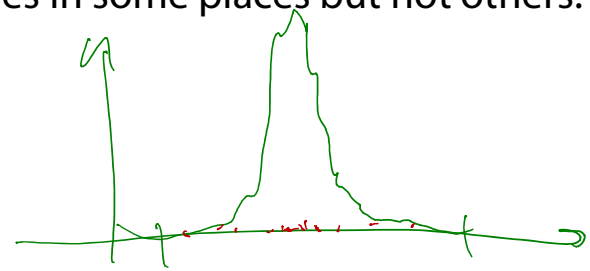
$$E\left[f\left(X\right)\right] = \int_{-\infty}^{\infty} f(x)p(x)dx$$

The variance of our integral estimate is:

$$V\left[\frac{1}{N}\sum_{i=1}^{N} f\left(X_i\right)\right] =$$

Monte Carlo integration

Suppose we are integrating a function that has large values in some places but not others.



How do we get more samples in the large value areas?

$p(x)$ pdf is big where $f(x)$ is big

Remember before that if our samples are more closely spaced, they should receive less weight. What function tells us how close spaced samples will be?

$p(x)$ $\frac{1}{p(x)}$ measures "spacing"

We can now estimate the integral as:

$$\frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)}$$

This is called **Monte Carlo integration**.

Monte Carlo integration (cont'd)

But is this sample mean really approximating the integral? Let's compute its expected value:

$$E\left[\frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)}\right]=$$

Now we need the expected value of $f(X)/p(X)$:

$$E\left[f\left(X\right)/\,p(X)\right]=$$

Bingo!

We can compute the variance of our integral estimator:

$$V\left[\frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)}\right]=$$

We want a low variance estimate. What variables and/or functions are under our control here?

Importance sampling

To reduce variance, we can now choose $p(x)$ to sample $f(x)$ well.

If $f(x)$ is non-negative, the optimal choice would be to set $p(x) \sim f(x)$.

$V\left[\frac{f(x)}{p(x)}\right] = 0$

$p(x) = kf(x)$

$k = \frac{1}{\int f(x)\,dx}$

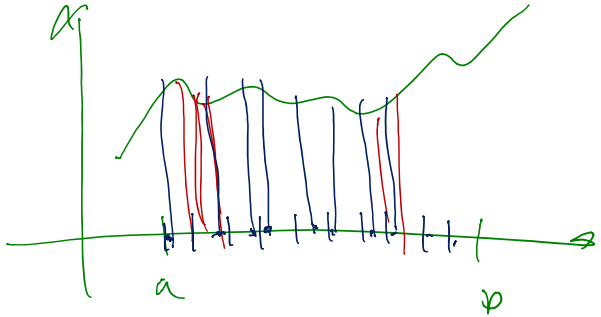
Why don't we just do that?

it requires knowledge of $\int f(x)\,dx$

Alternatively, we can use heuristics to guess where $f(x)$ will be large and choose $p(x)$ based on those heuristics. This approach is called **importance sampling**.

Stratified sampling

Another source of variance is "bad luck". Consider a uniform sampling of our function, where samples may clump:



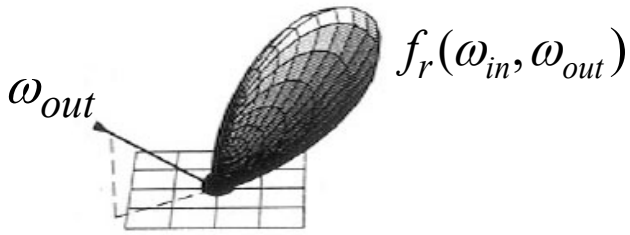
To reduce this problem, we can apply **stratified sampling**.

The idea is that, given your probability function:

- You can break it up into bins of equal probability area (i.e., equal likelihood).
- Then choose a sample from each bin.

Importance sampling of reflection

For a given BRDF:



again the surface reflection equation is:

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

Without importance sampling:

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

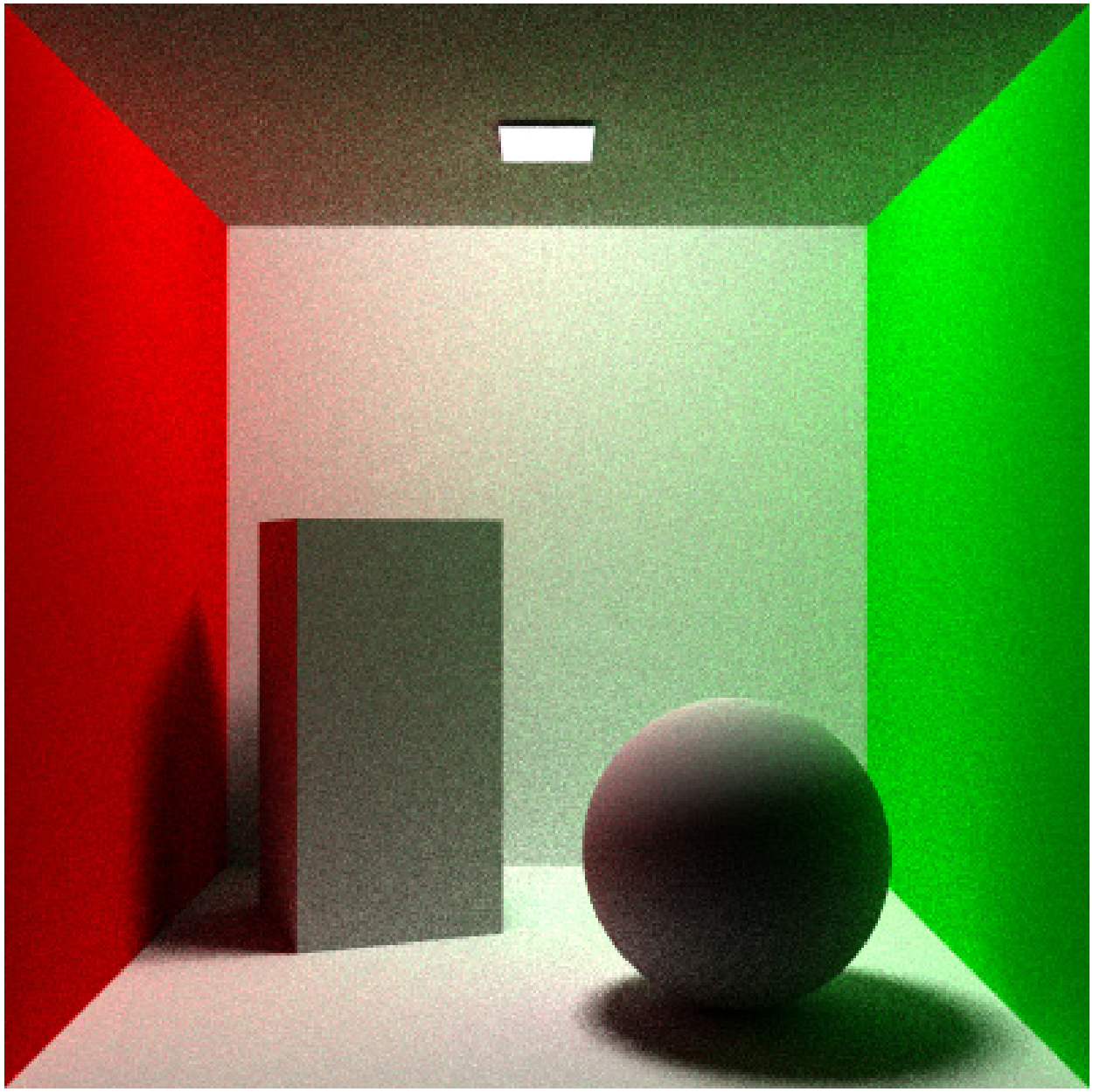
With importance sampling:

- 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: $f_r(\omega_{in}, \omega_{out})(\omega_{in} \cdot \mathbf{N}) / p(\omega_{in})$

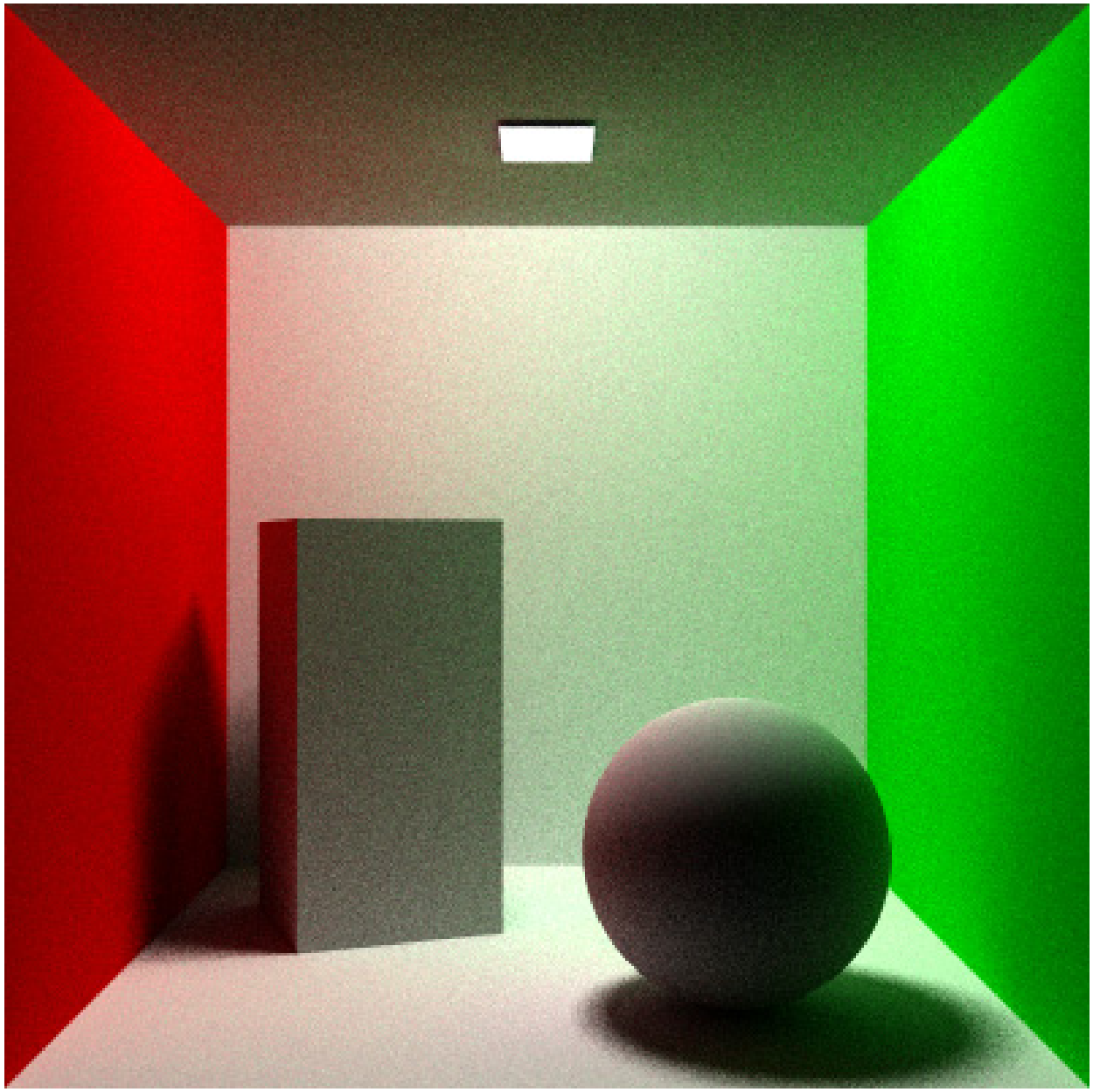
Ideally, the distribution is proportional to the BRDF:

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

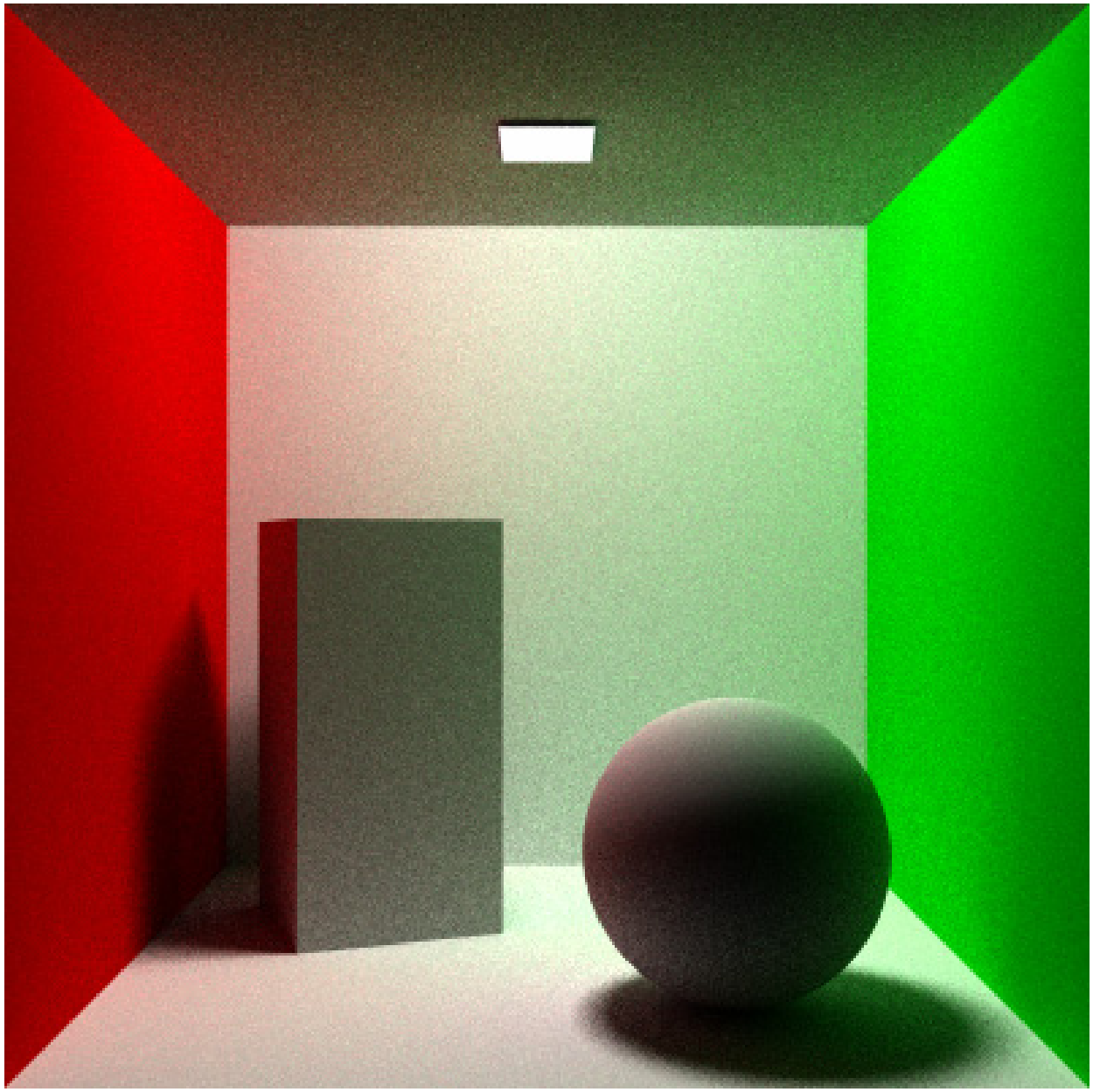
See Pharr handout for cosine weighted sampling of the hemisphere, very useful for diffuse reflection.



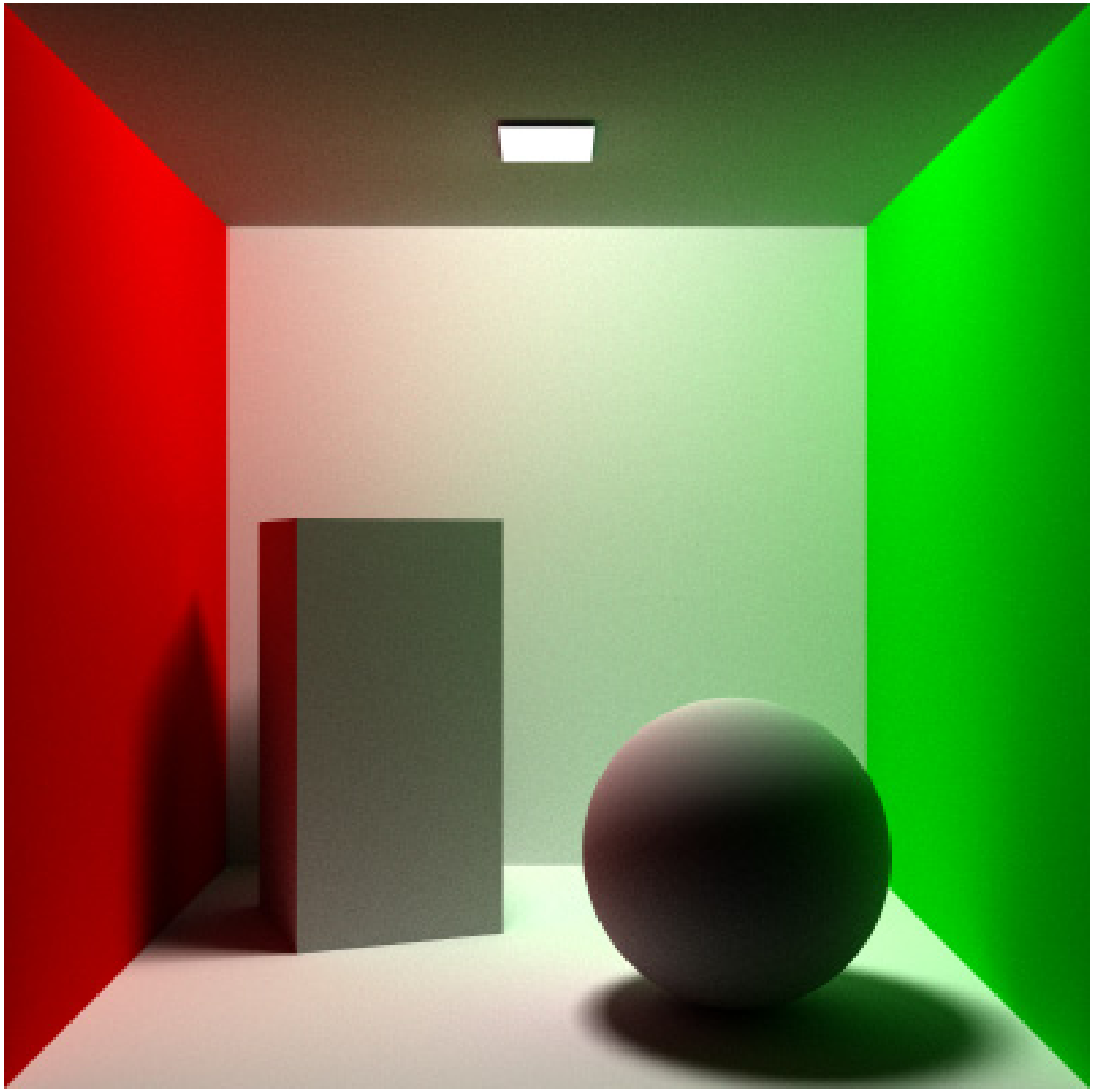
100 rays/pixel **without** importance sampling



100 rays/pixel **with** importance sampling

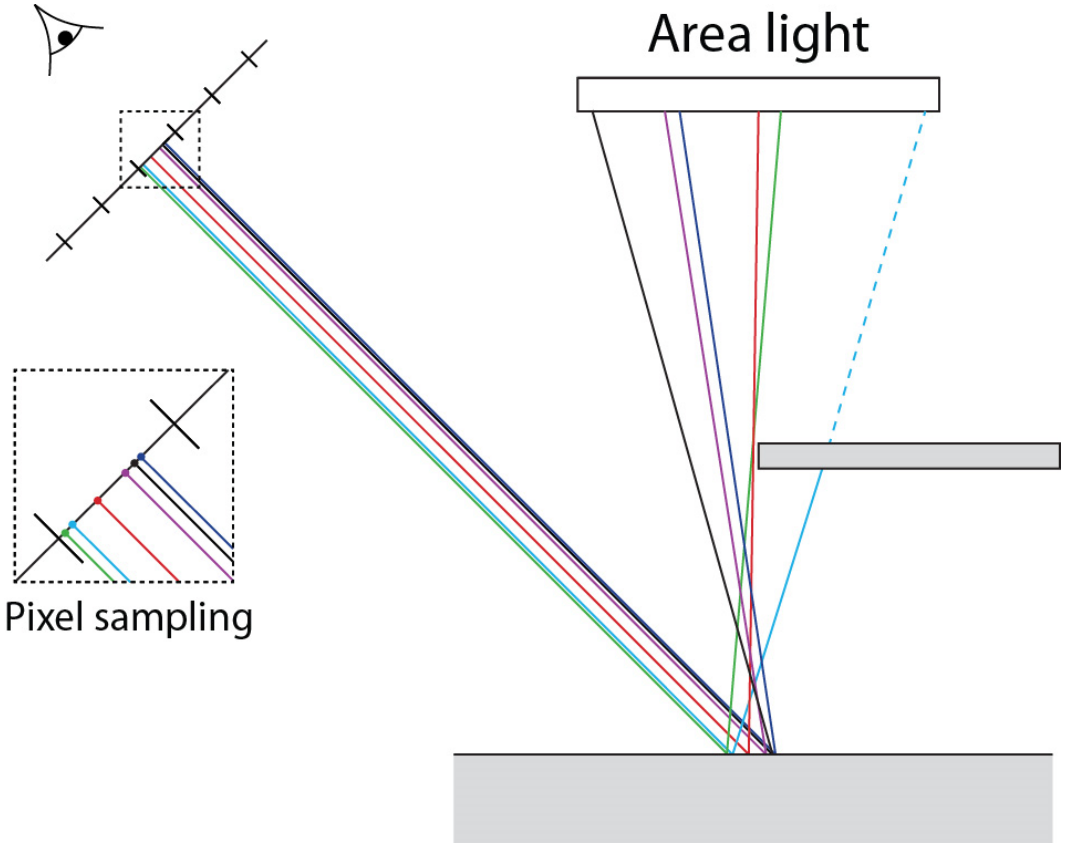


200 rays/pixel **without** importance sampling

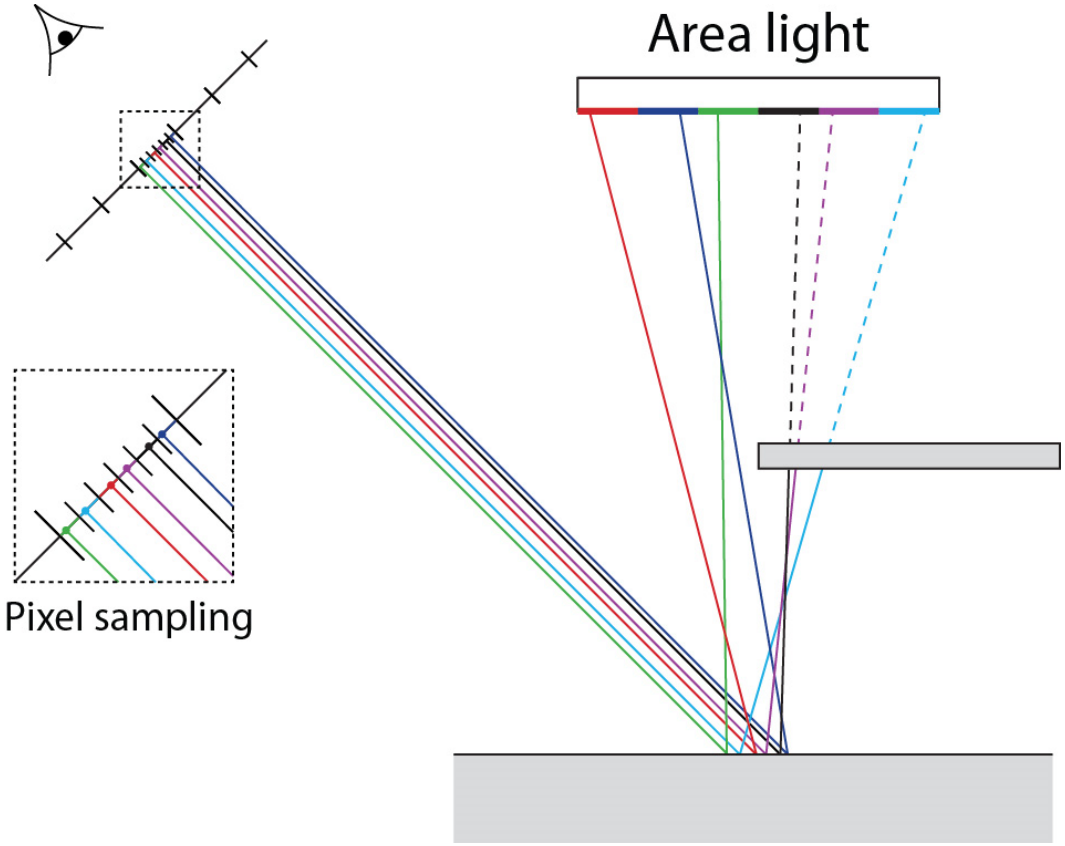


900 rays/pixel **with** importance sampling

Penumbra revisited: clumped samples



Penumbra: stratified sampling

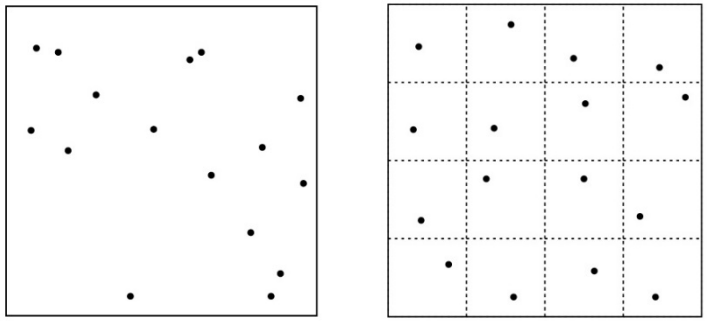


Stratified sampling gives a better distribution of samples:

- 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

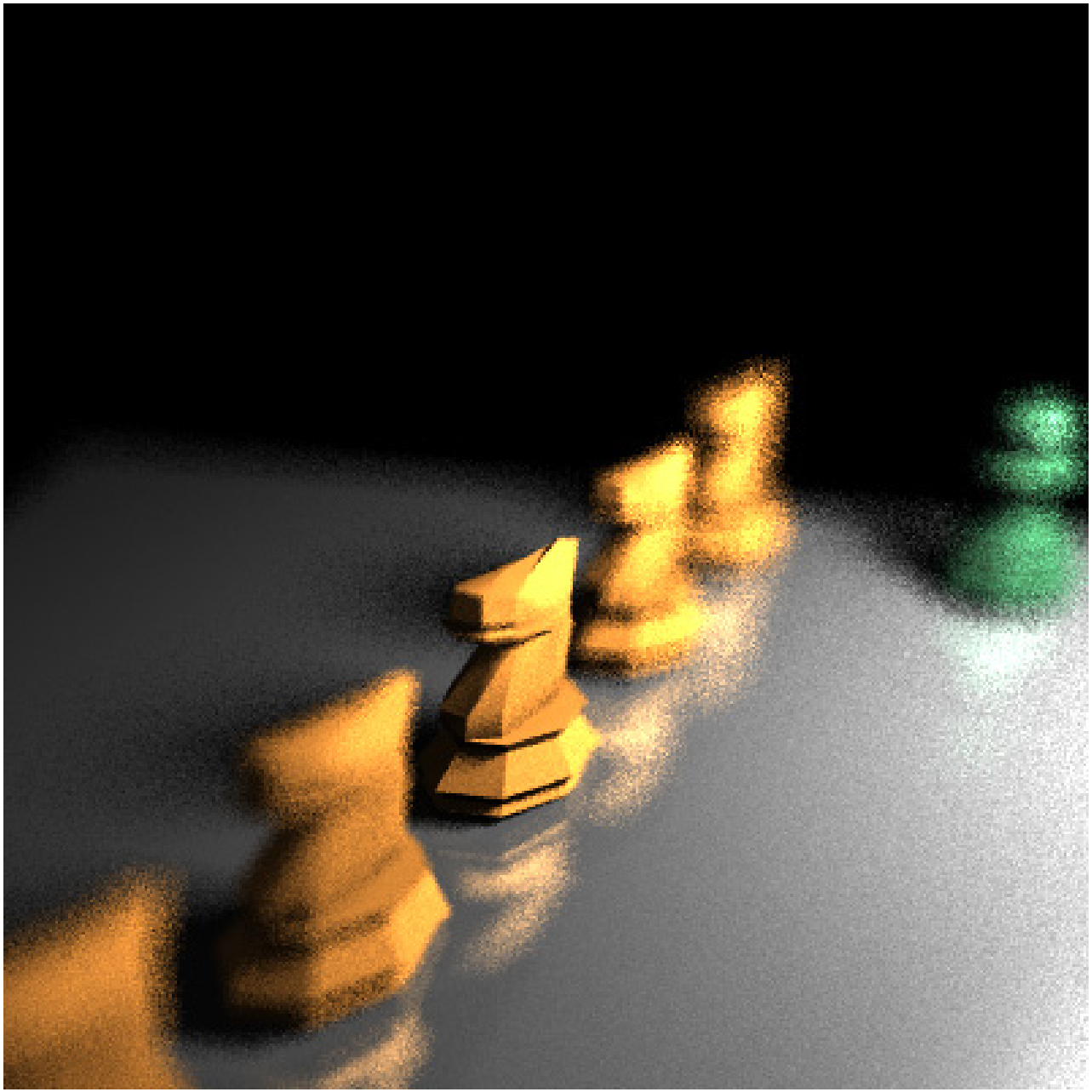
Here we see pure uniform vs. stratified sampling over a 2D pixel (here 16 rays/pixel):



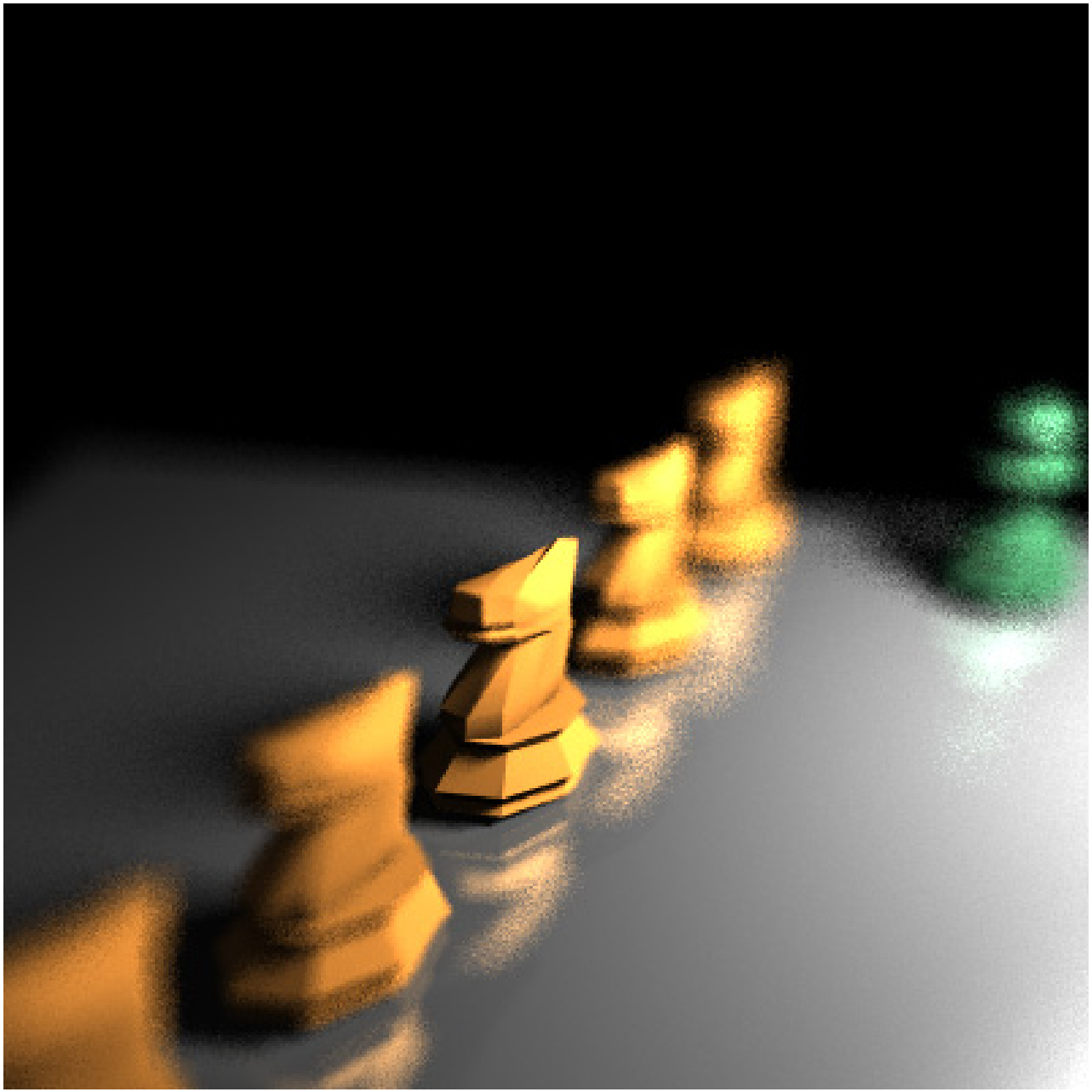
Random Stratified

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

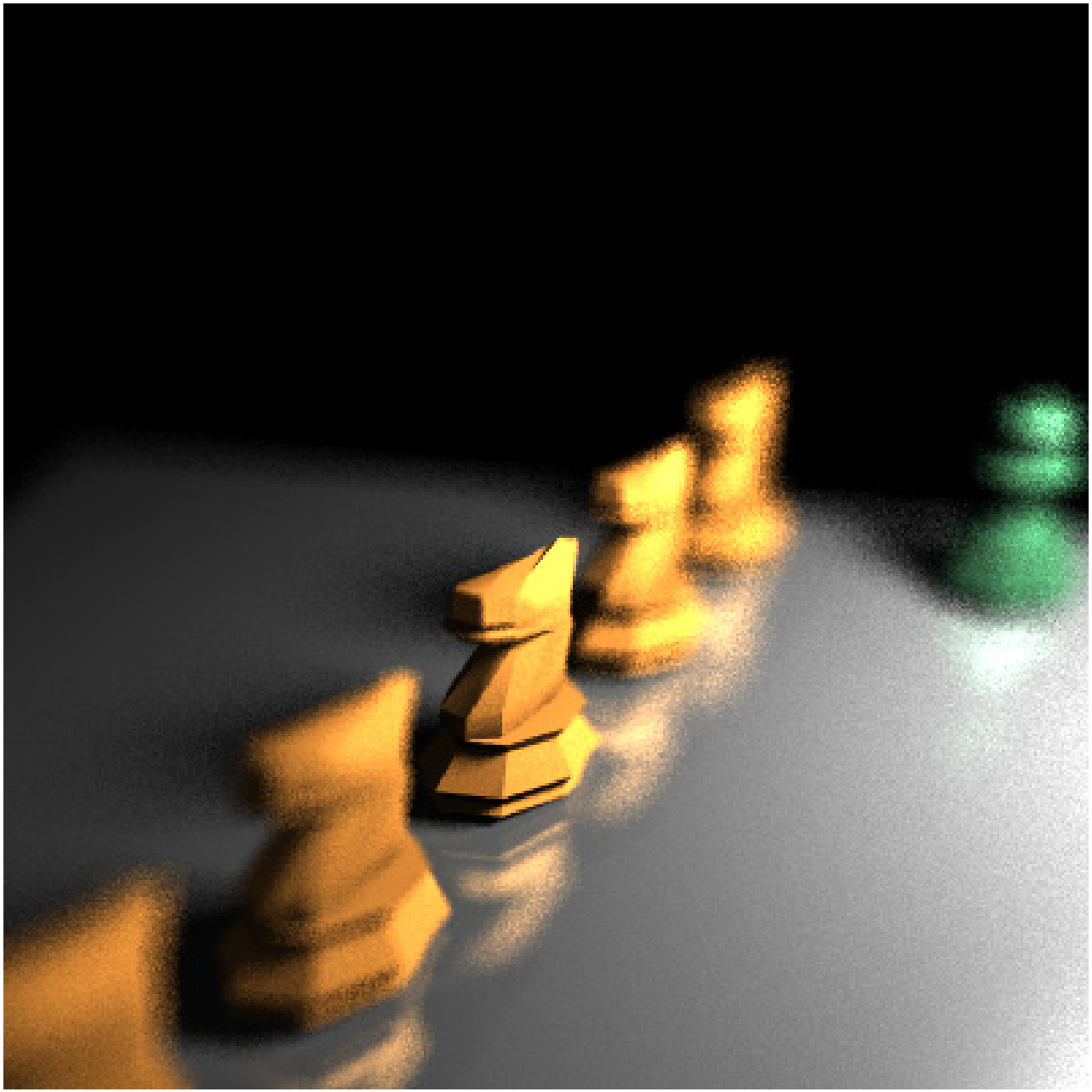
Similar grids can be constructed over the camera aperture, light sources, and diffuse/glossy reflection directions.



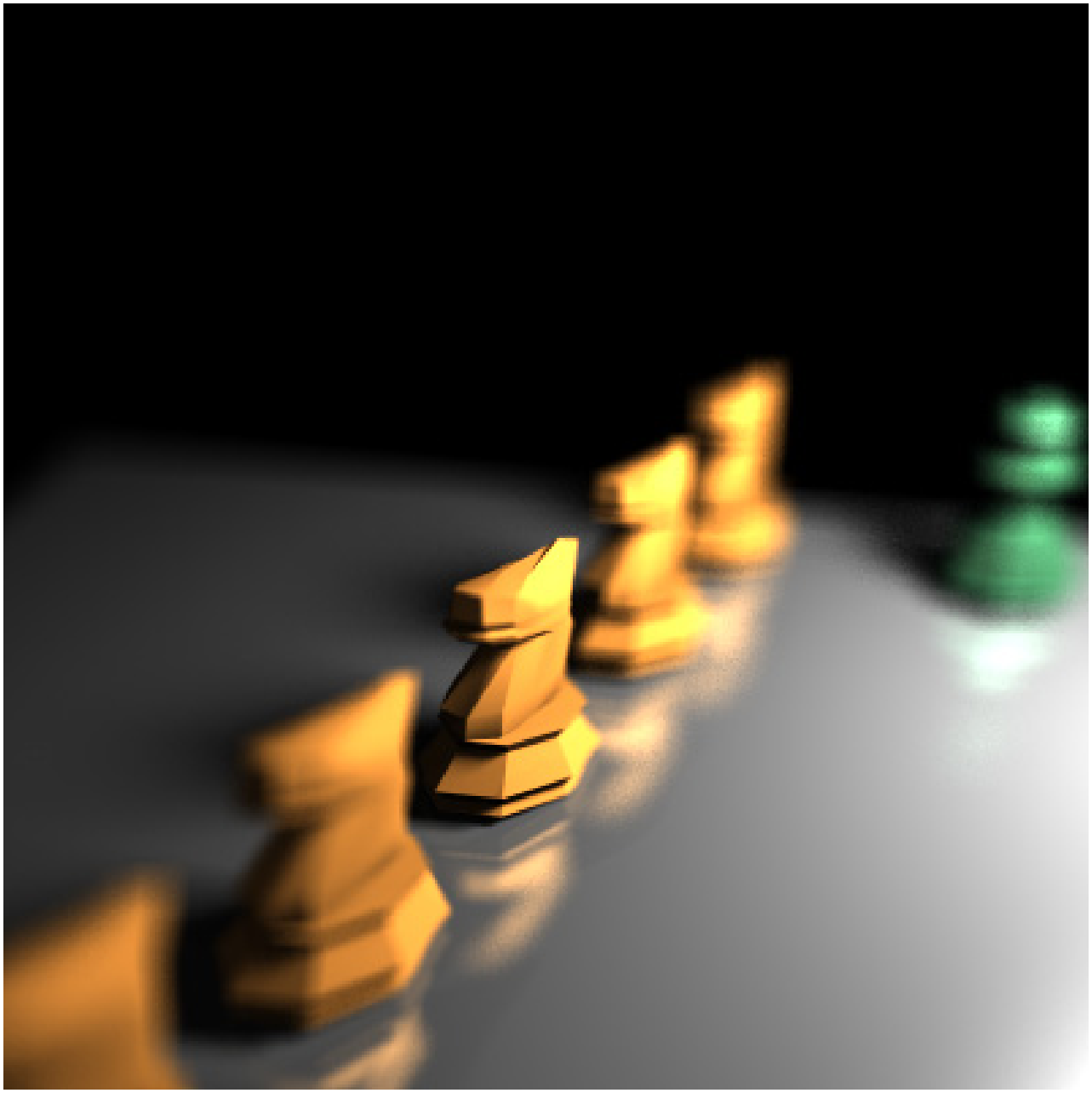
25 rays/pixel **without** stratified sampling



25 rays/pixel **with** stratified sampling



64 rays/pixel **without** stratified sampling



400 rays/pixel **with** stratified sampling