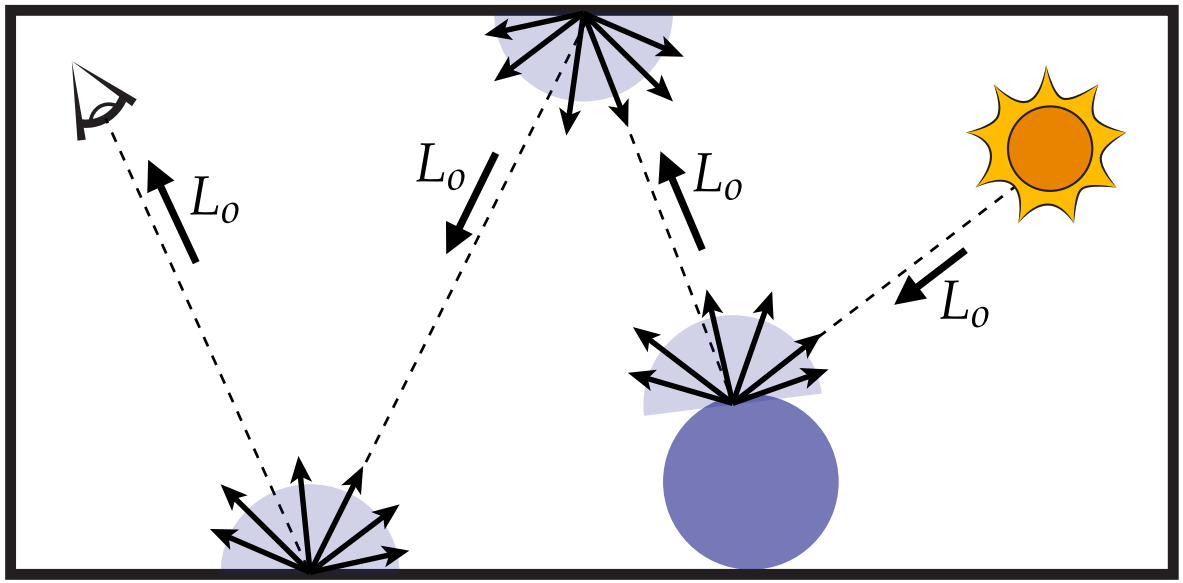
Lecture 13:

Monte Carlo Ray Tracing

Computer Graphics 2025

Fuzhou University - Computer Science

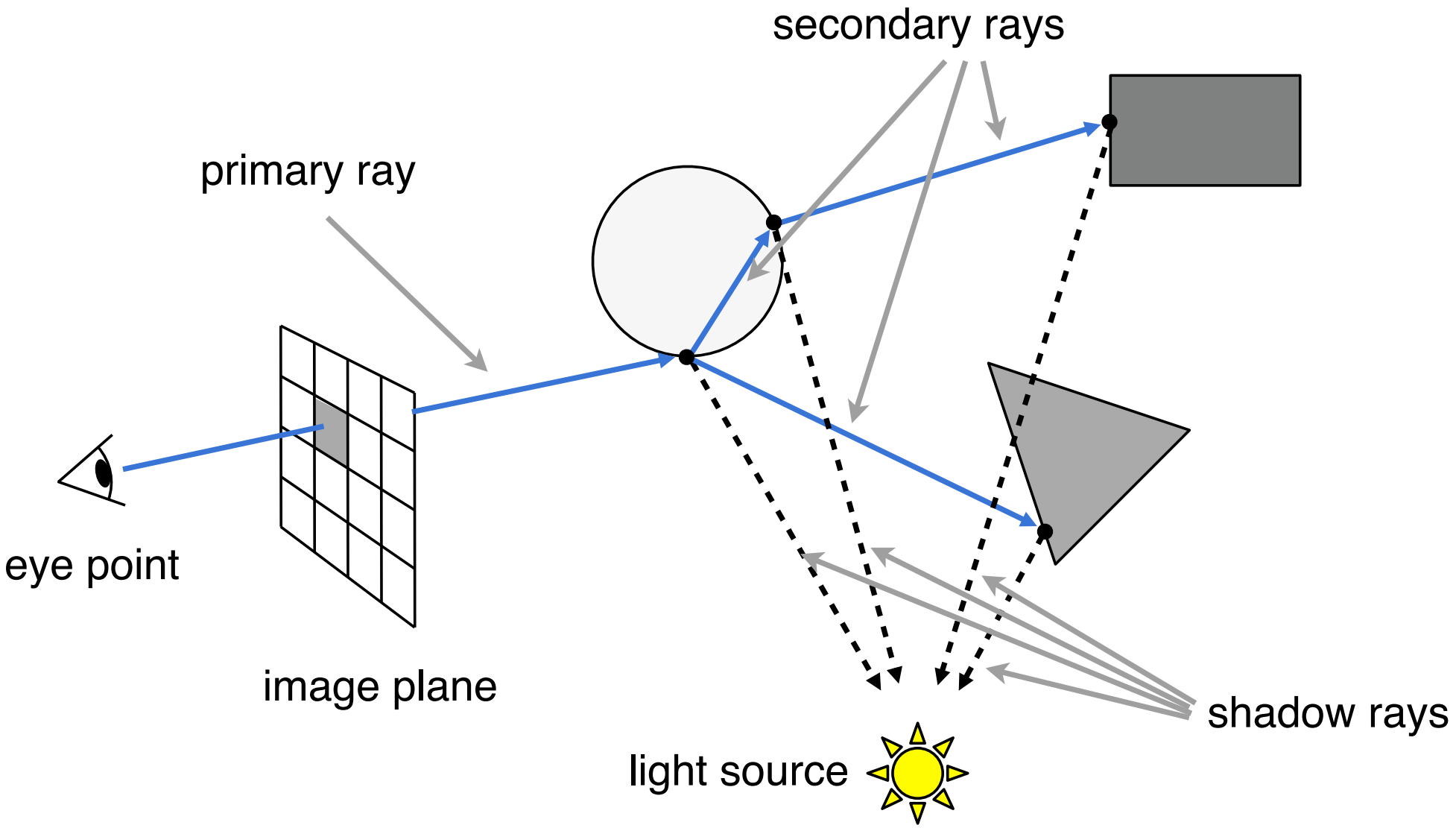
Recall Rendering Equation



$$L_o(\mathbf{p}, \omega_o) = L_e(\mathbf{p}, \omega_o) + \int_{\mathcal{H}^2} f_r(\mathbf{p}, \omega_i \rightarrow \omega_o) L_i(\mathbf{p}, \omega_i) \cos\theta \, d\omega_i$$

How can we possibly evaluate this integral?

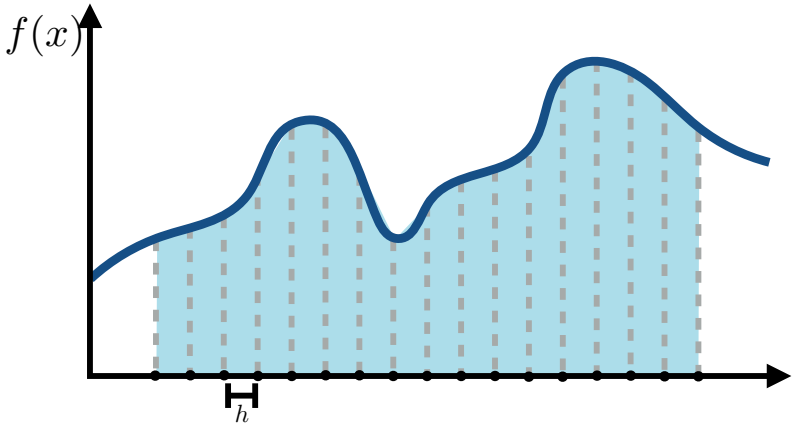
Recall Ray Tracing



Numerical Integration — Overview

- In graphics, many quantities we're interested in are naturally expressed as integrals (total brightness, total area, ...)
- For very, very simple integrals, we can compute the solution analytically
- For everything else, we have to compute a numerical approximation
- Basic idea:
  - integral is "area under curve"
  - sample the function at many points
  - integral is approximated as weighted sum

$$\int_0^1 \frac{1}{3}x^2\,dx = \left[x^3\right]_0^1 = 1$$



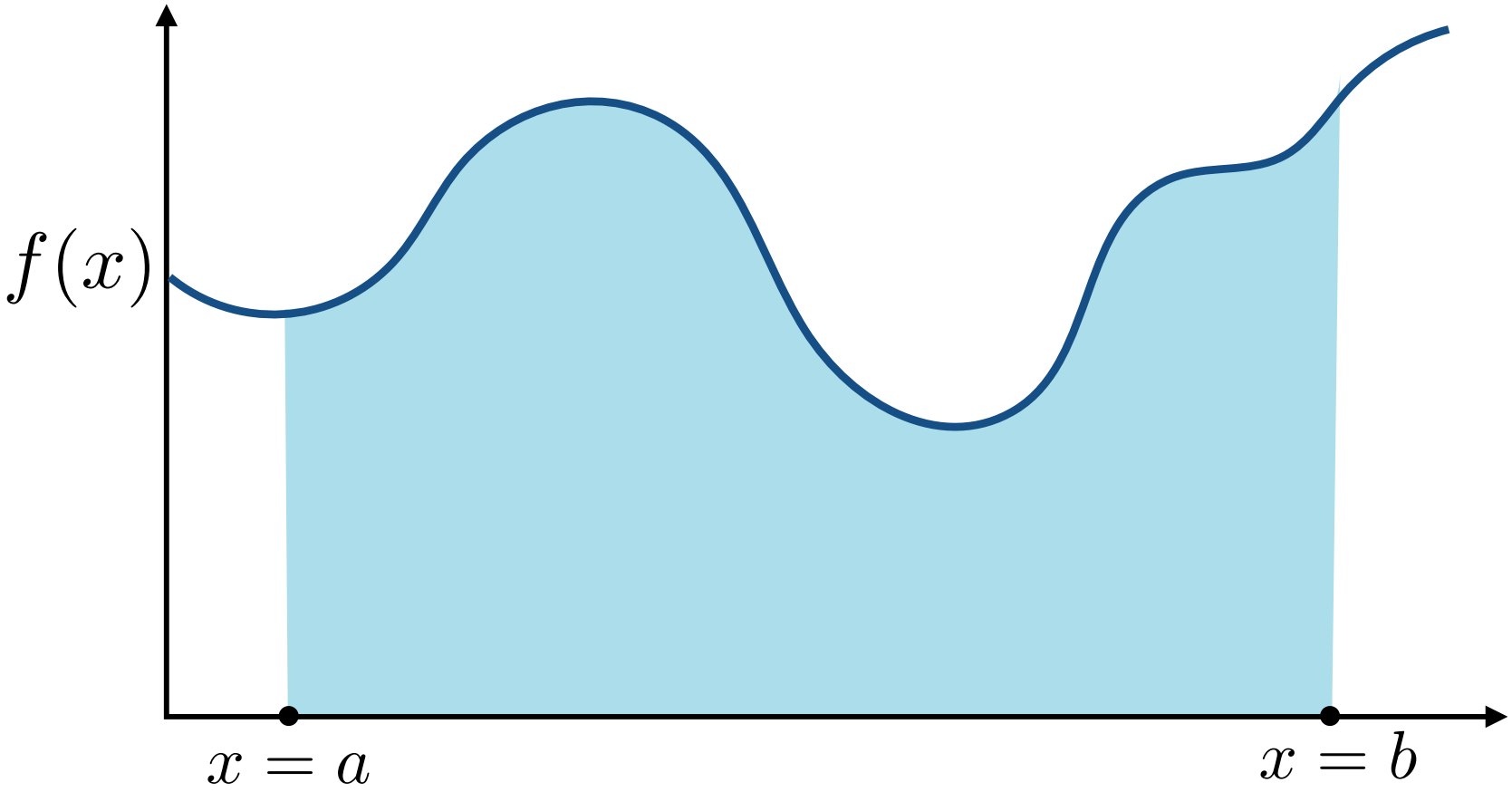
Rendering: What Are We Integrating?

- Recall this view of the world:

Want to "sum up" — i.e., integrate!
— light from all directions

Review: Integral as "area under curve"

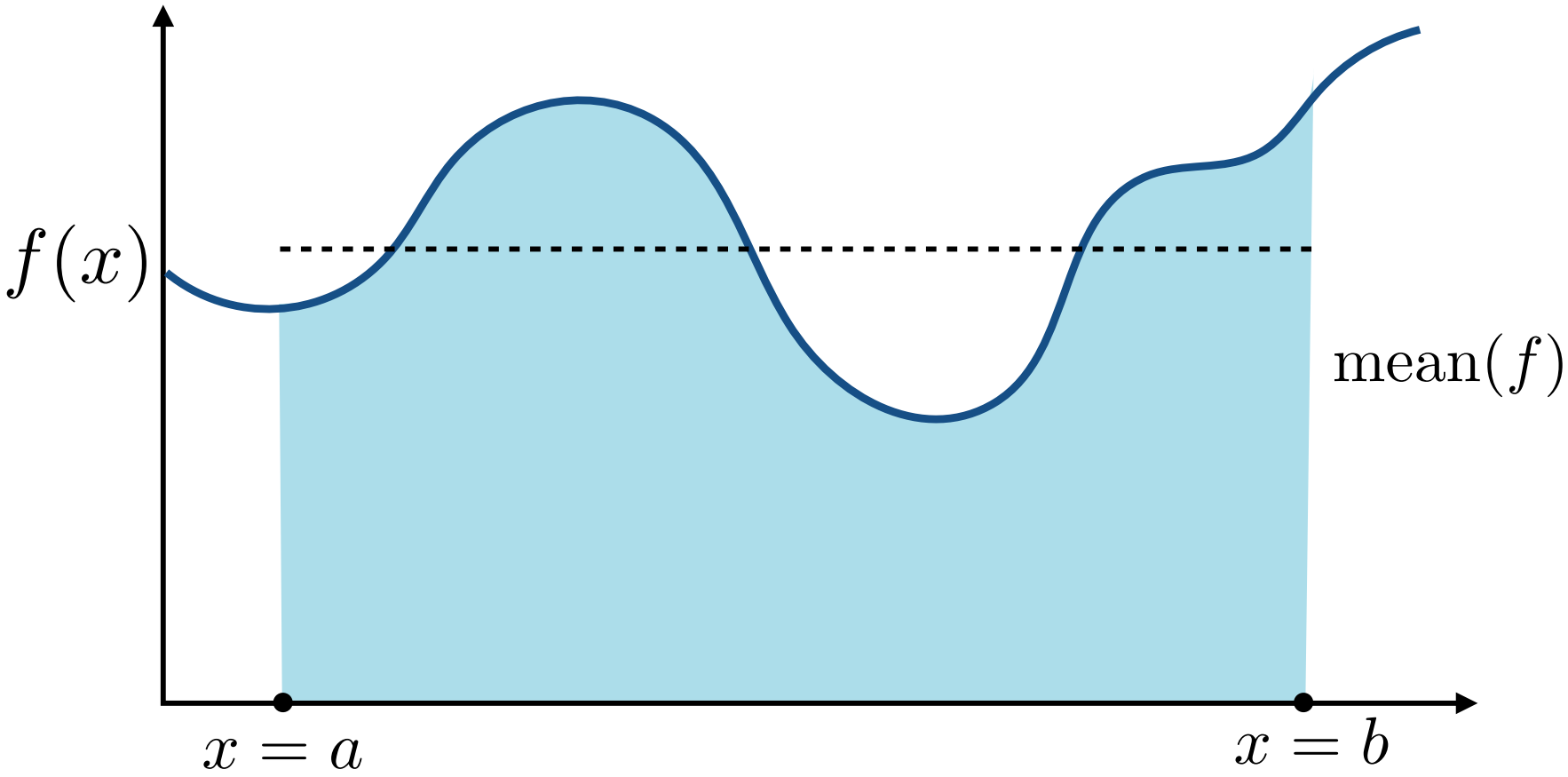
$$\int_a^b f(x)dx$$



Or: average value times size of domain

$$\int_a^b f(x)dx = (b-a)\text{mean}(f)$$

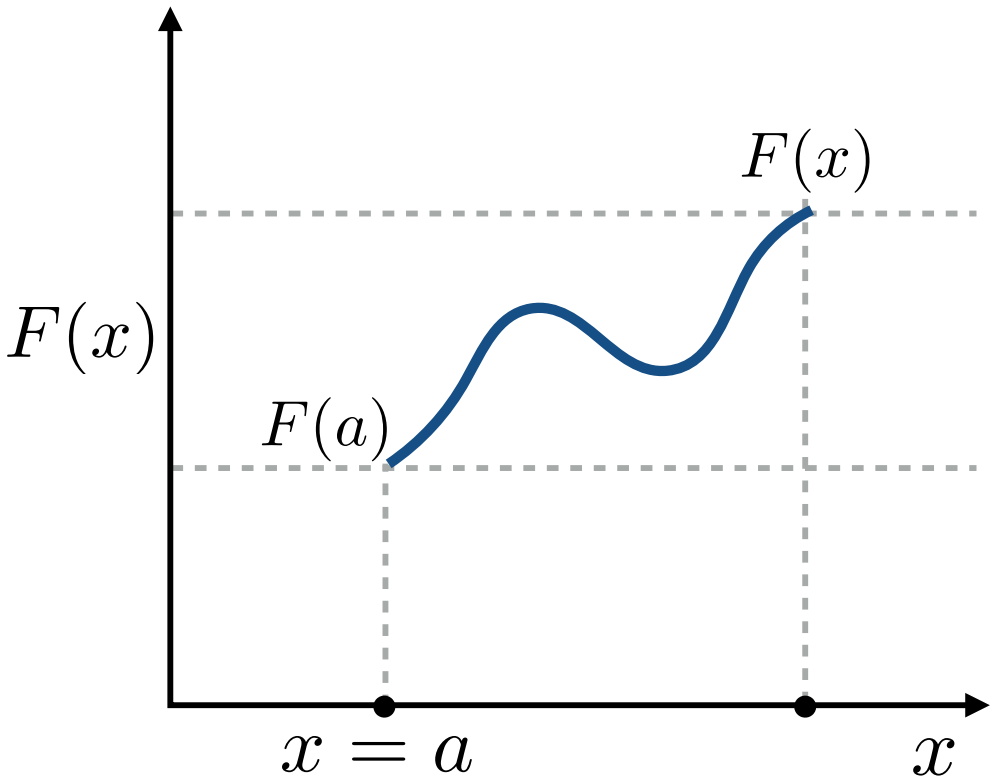
积分中值定理



Review: Fundamental Theorem of Calculus

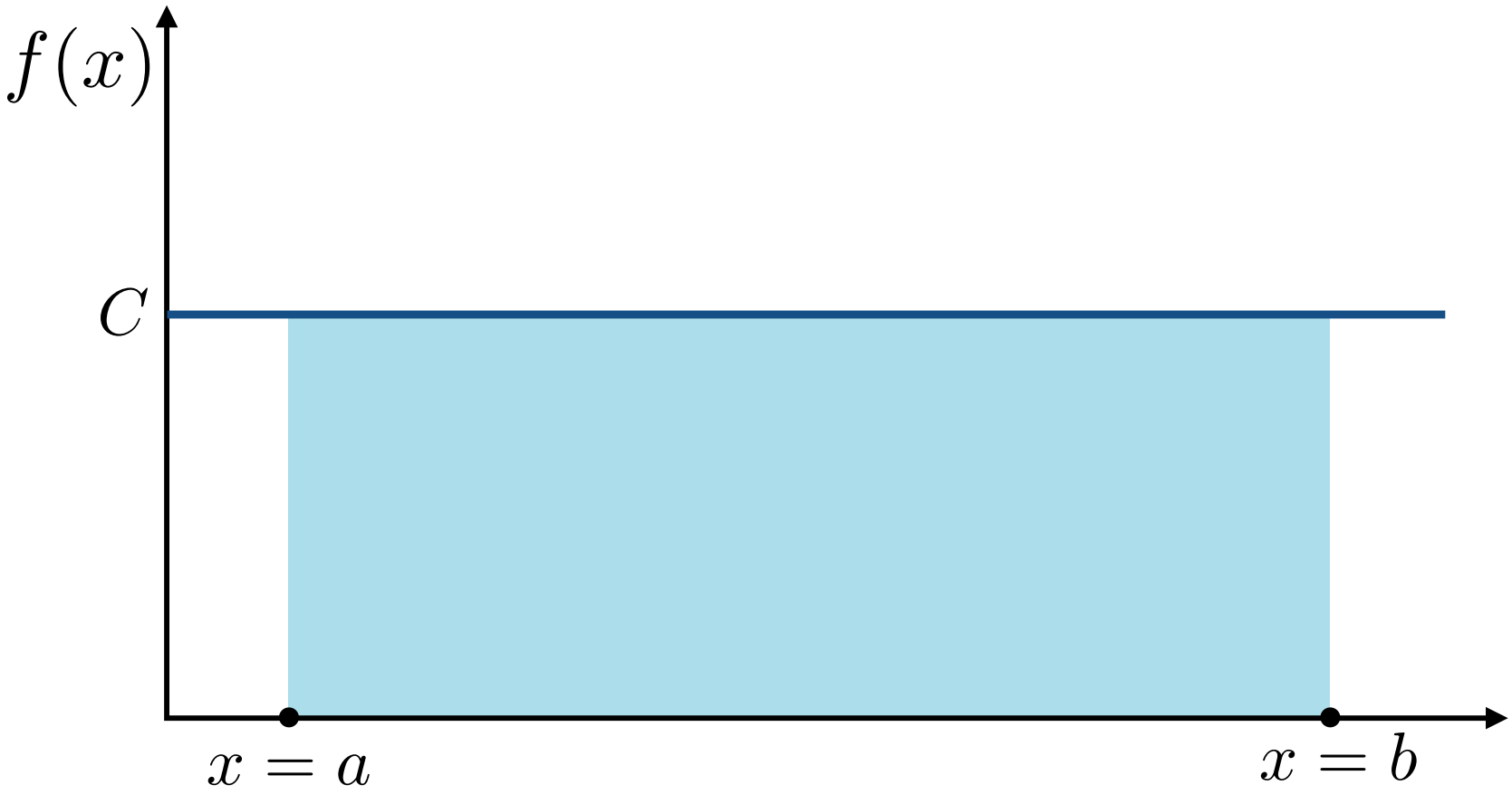
$$\int_a^b f(x)dx = F(b) - F(a)$$

$$f(x) = \frac{d}{dx}F(x)$$



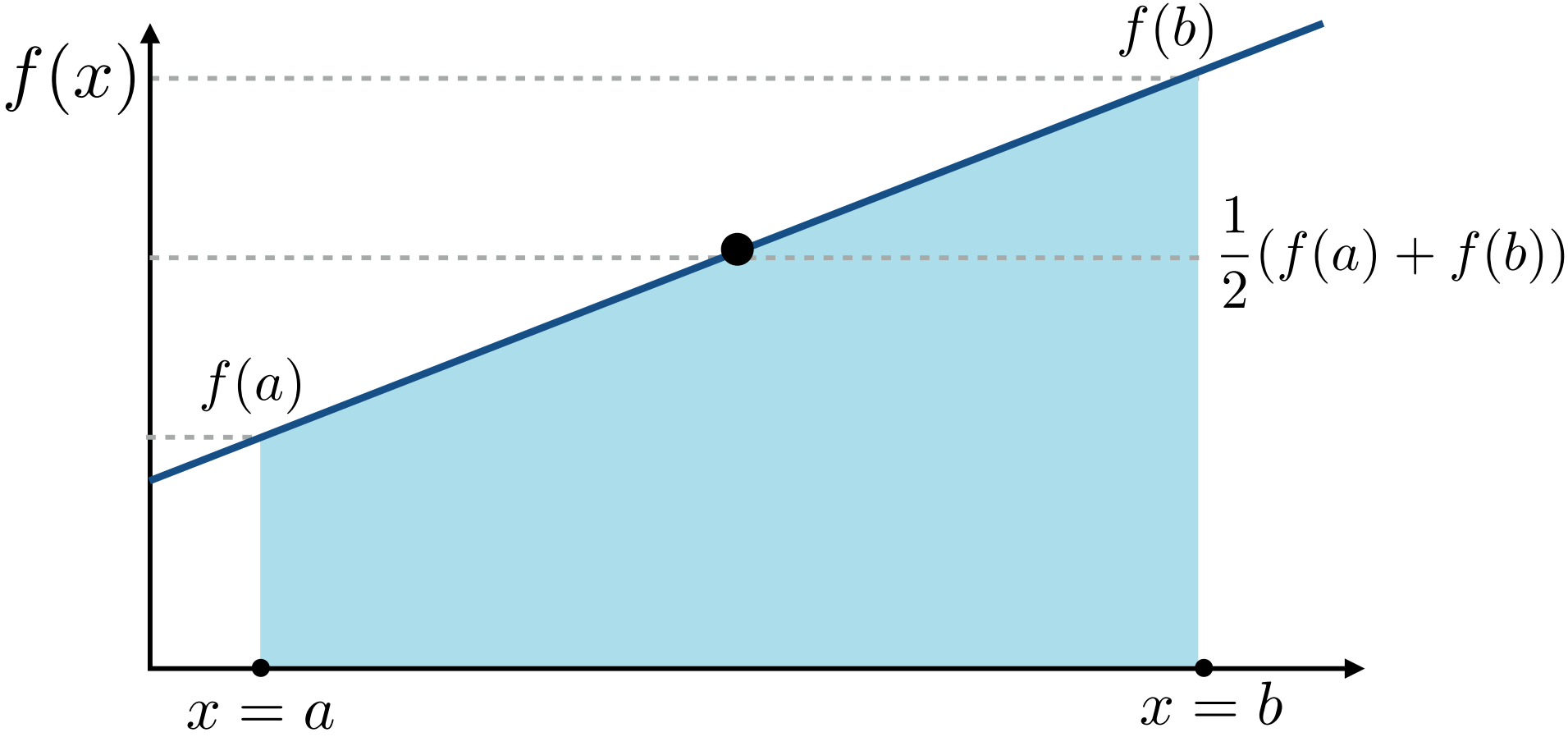
Simple Case: Constant Function

$$\int_a^b C dx = (b-a)C$$



Affine Function: $f(x) = cx + d$

$$\int_a^b f(x)dx = \frac{1}{2}(f(a) + f(b))(b - a)$$

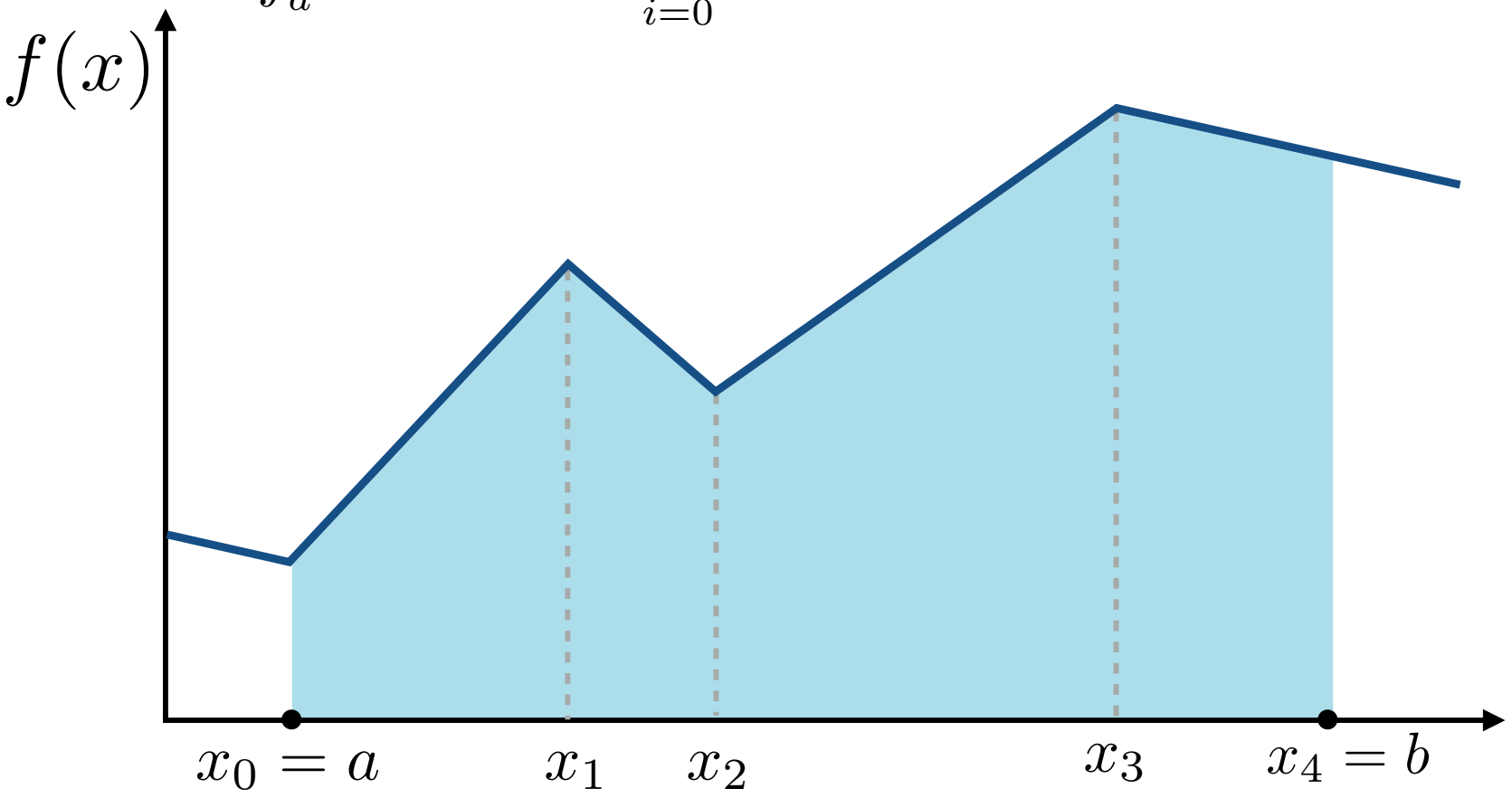


Need only one sample of the function

Piecewise Affine Function

For piecewise functions, just sum integral of each piece:

$$\int_a^b f(x)dx = \frac{1}{2}\sum_{i=0}^{n-1}(x_{i+1} - x_i)(f(x_i) + f(x_{i+1}))$$



Piecewise Affine Function

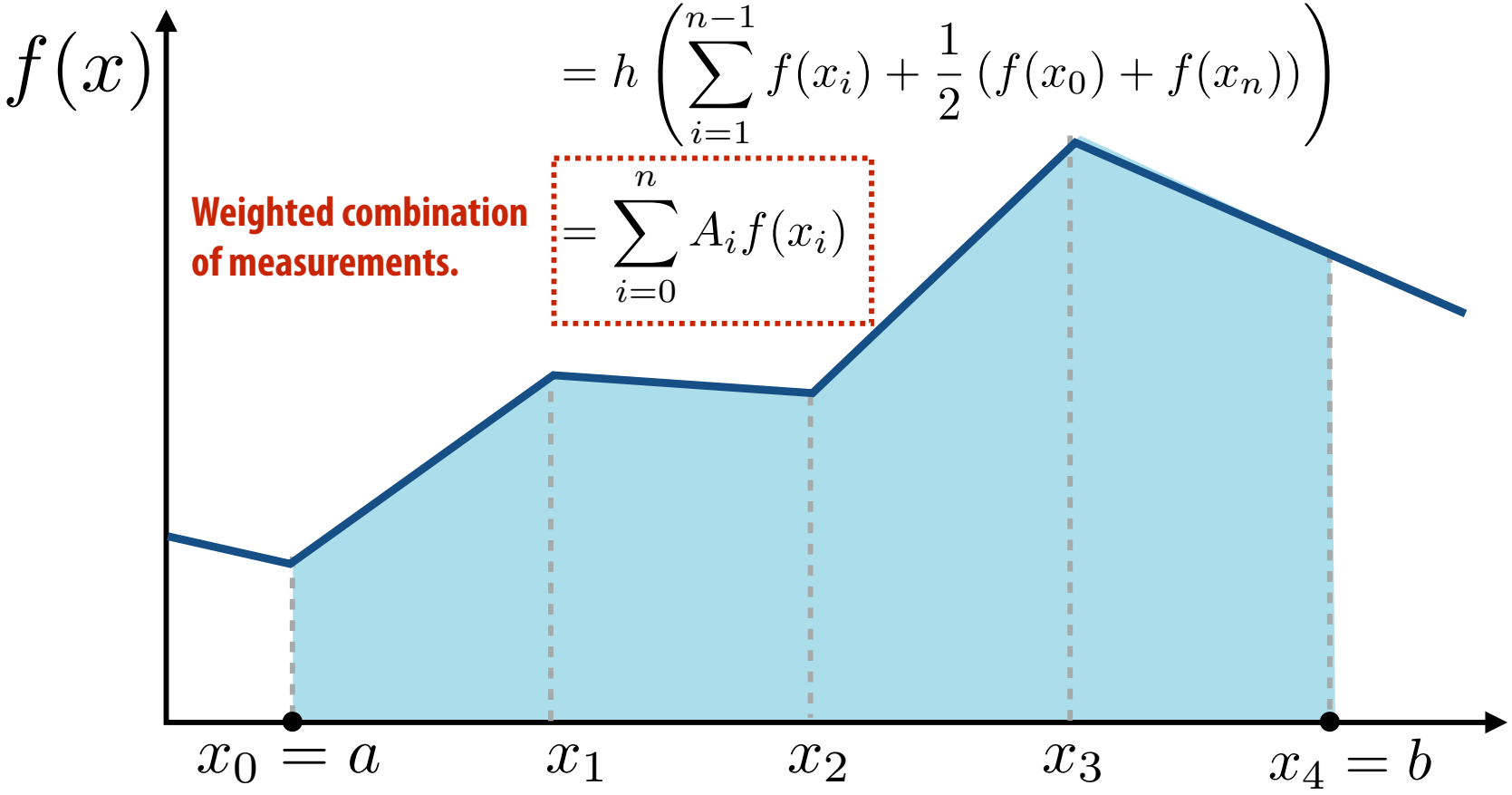
If N-1 segments are of equal length:

$$\int_a^b f(x)dx = \frac{h}{2}\sum_{i=0}^{n-1}(f(x_i) + f(x_{i+1}) \qquad h = \frac{b-a}{n-1}$$

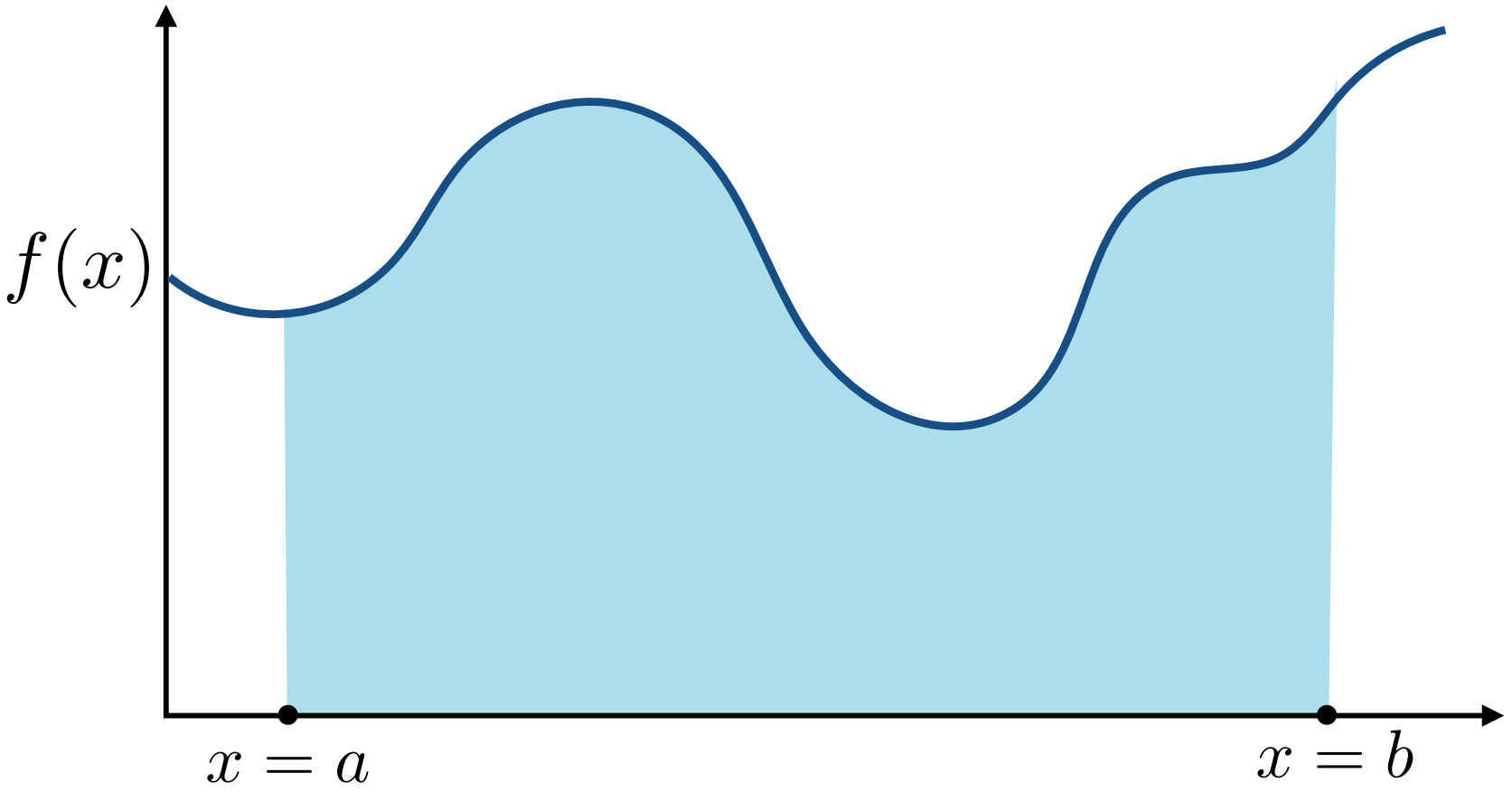
$$= h\left(\sum_{i=1}^{n-1} f(x_i) + \frac{1}{2}\left(f(x_0) + f(x_n)\right)\right)$$

$$= \sum_{i=0}^{n} A_i f(x_i)$$

Weighted combination of measurements.

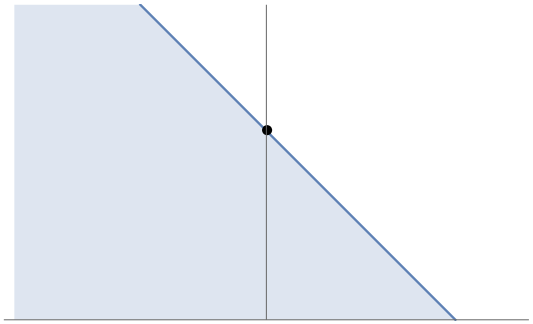


More General Polynomials?

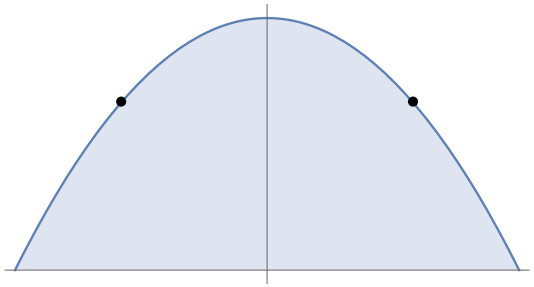


Gauss Quadrature 高斯求积

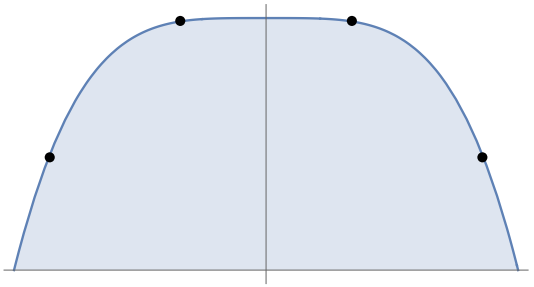
- For any polynomial of degree n, we can always obtain the exact integral by sampling at a special set of n points and taking a special weighted combination



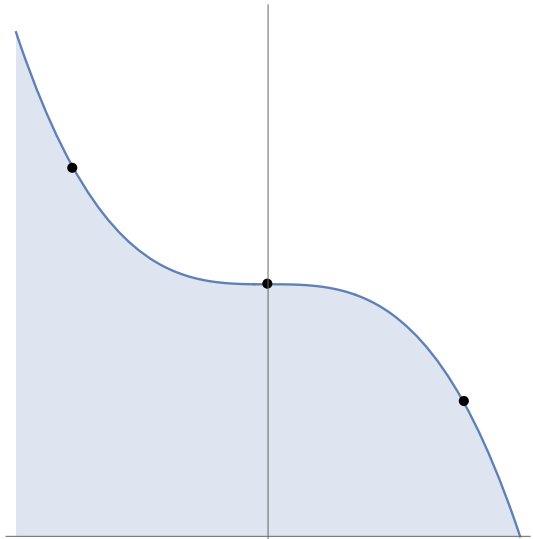
n=1



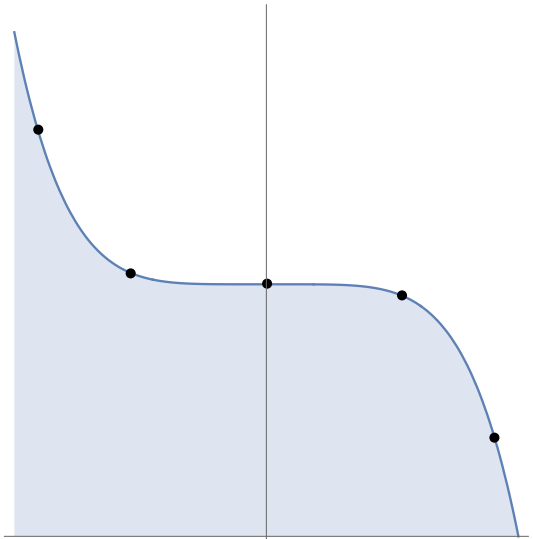
n=2



n=4



n=3



n=5

https://en.wikipedia.org/wiki/Gaussian_quadrature

Key idea so far:

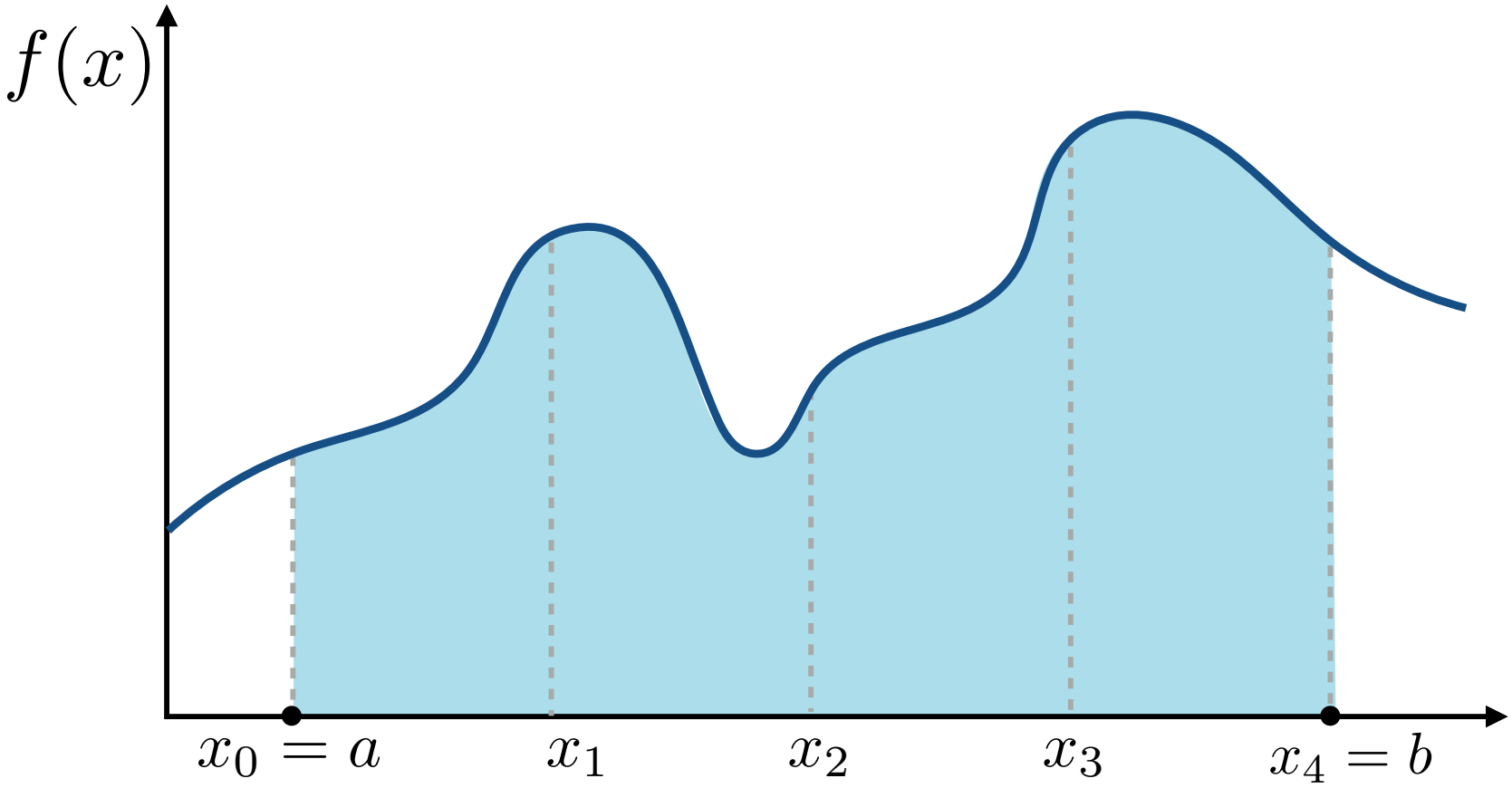
To approximate an integral, we need

(i) quadrature points, and

(ii) Weights for each point

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

Arbitrary Function $f(x)$?

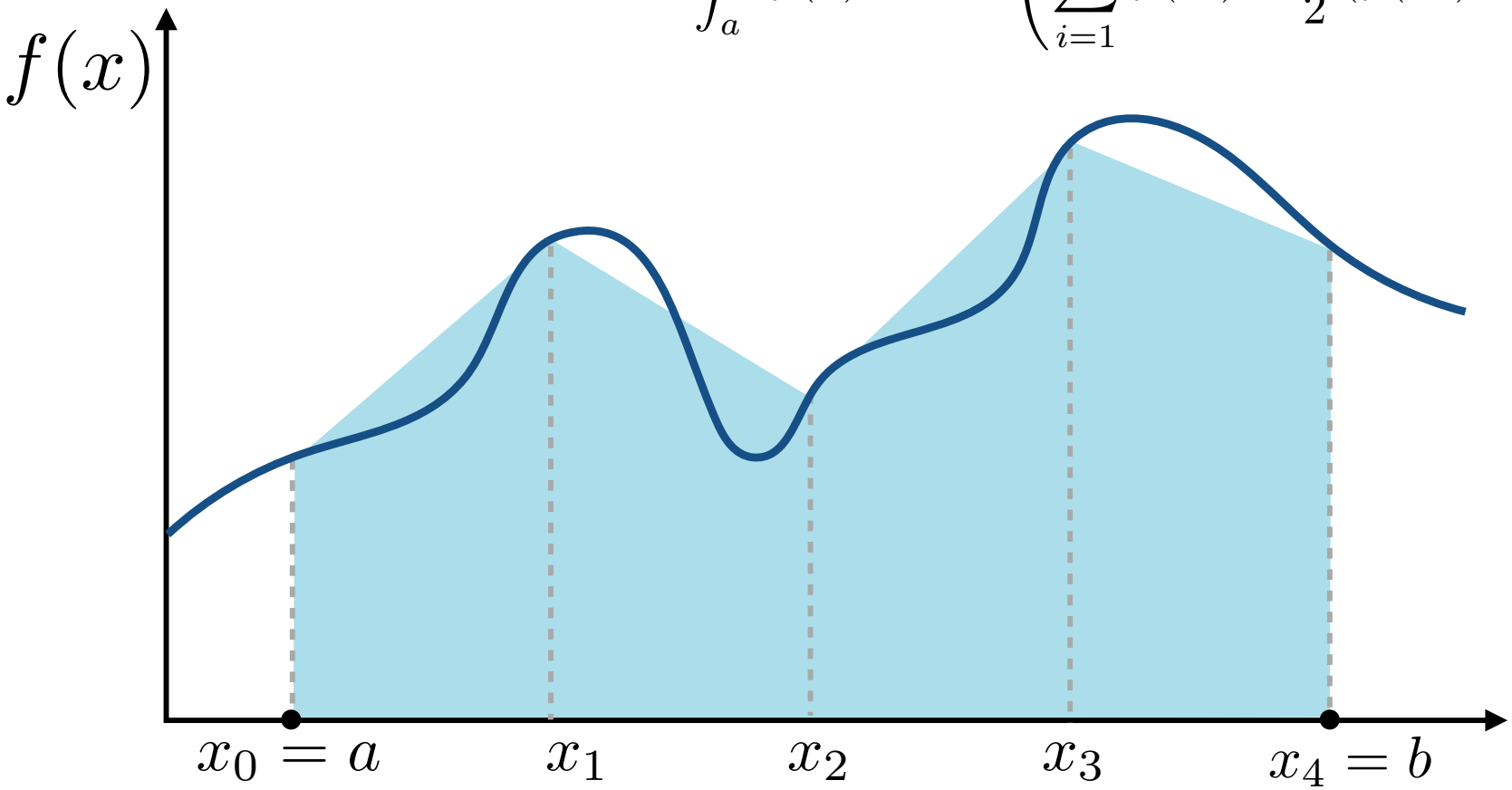


Trapezoid Rule 梯形法则

Approximate integral of $f(x)$ by assuming function is piecewise linear

For equal length segments: $h = \dfrac{b-a}{n-1}$

$$\int_a^b f(x)dx = h\left(\sum_{i=1}^{n-1} f(x_i) + \frac{1}{2}\left(f(x_0) + f(x_n)\right)\right)$$

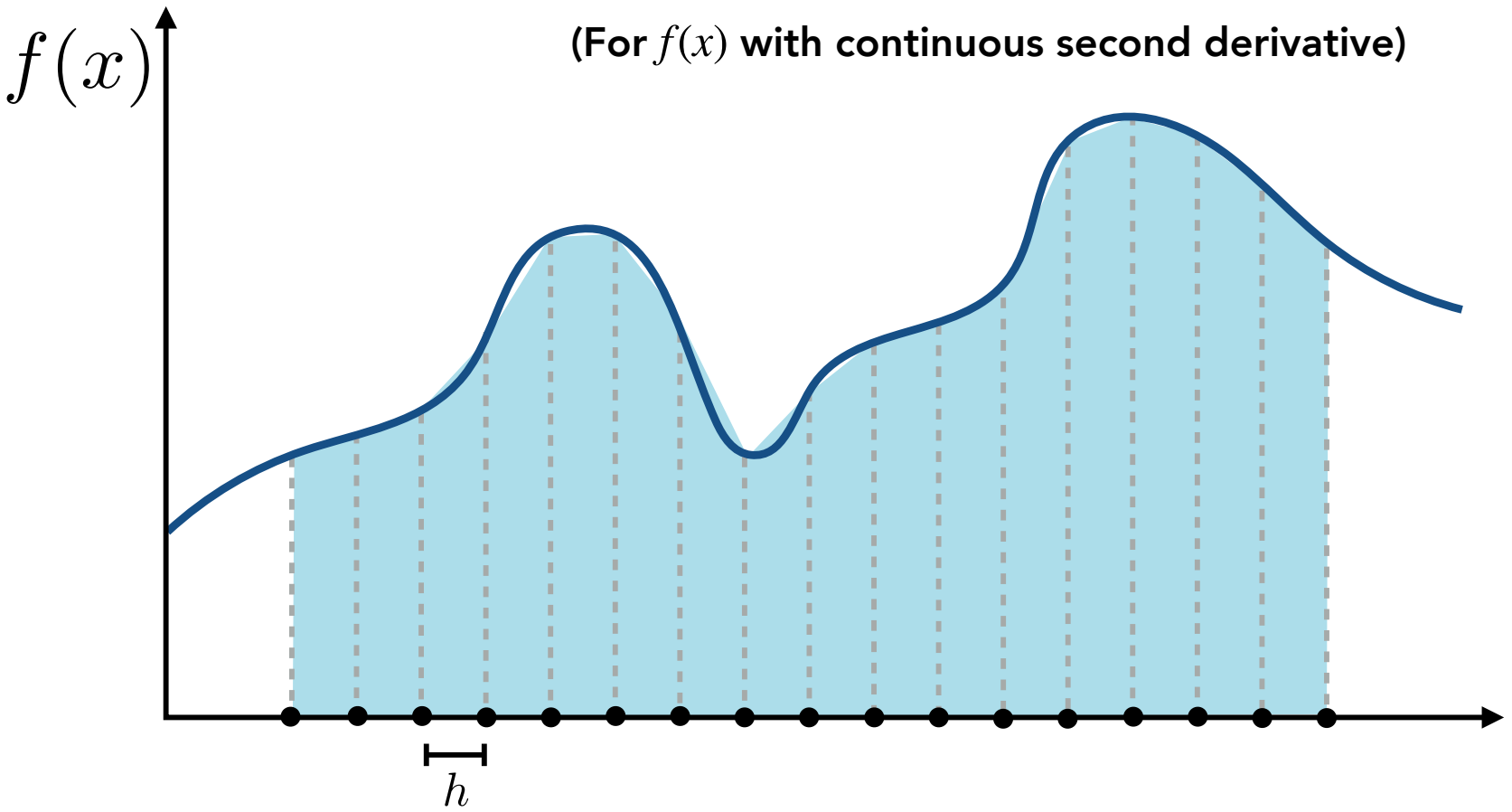


Trapezoid Rule

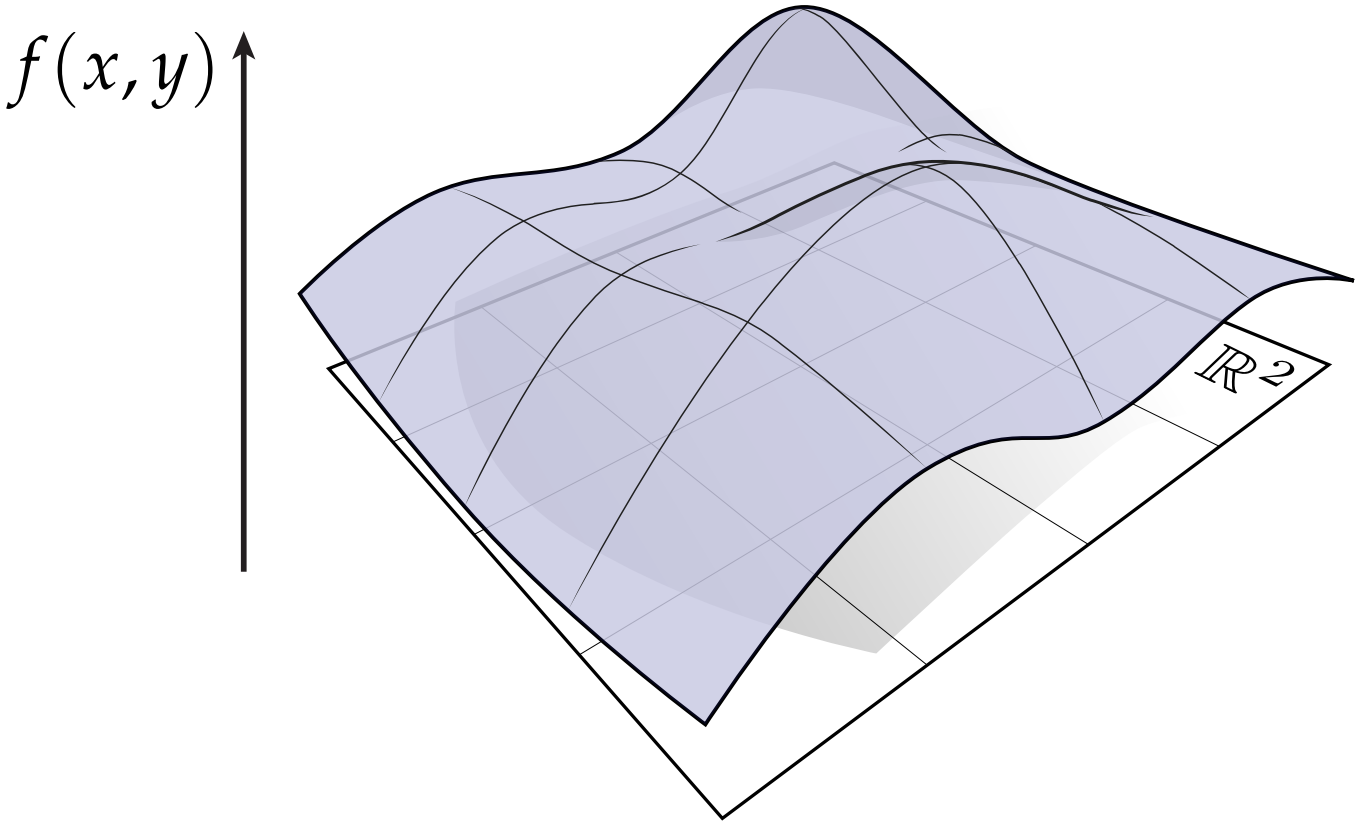
Consider cost and accuracy of estimate as $n \to \infty$ (or $h \to 0$)

Work: $O(n)$

Error can be shown to be: $O(h^2) = O(\frac{1}{n^2})$



What about a 2D function?



How should we approximate the area underneath?

Integration in 2D (2D Trapezoidal Rule)

Consider integrating $f(x, y)$ using the trapezoidal rule

(Apply rule twice: when integrating in x and in y)

$$\int_{a_y}^{b_y}\int_{a_x}^{b_x} f(x,y)dxdy = \int_{a_y}^{b_y} \left[O(h^2) + \sum_{i=0}^{n} A_i f(x_i, y)\right] dy$$

First application of rule

$$= O(h^2) + \sum_{i=0}^{n} A_i \int_{a_y}^{b_y} f(x_i, y)dy$$

$$= O(h^2) + \sum_{i=0}^{n} A_i \left[O(h^2) + \sum_{j=0}^{n} A_j f(x_i, y_j)\right]$$

Second application

$$= O(h^2) + \sum_{i=0}^{n}\sum_{j=0}^{n} A_i A_j f(x_i, y_j)$$

Errors add, so error still: $O(h^2)$

But work is now: $O(n^2)$

(*n* x *n* set of measurements)

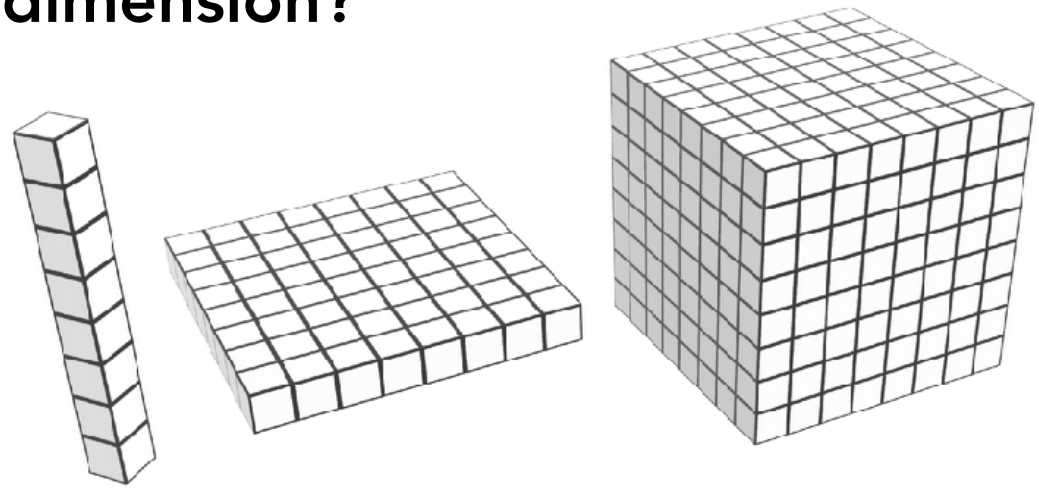
Must perform much more work in 2D to get same error bound on integral!

In K-D, let $N = n^k$

Error goes as: $O\left(\frac{1}{N^{2/k}}\right)$

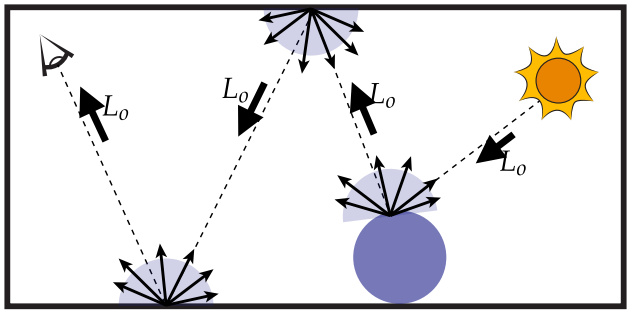
Curse of Dimensionality

- How much does it cost to apply the trapezoid rule as we go up in dimension?
  - 1D: $O(n)$
  - 2D: $O(n^2)$
  - ...
  - kD: $O(n^k)$



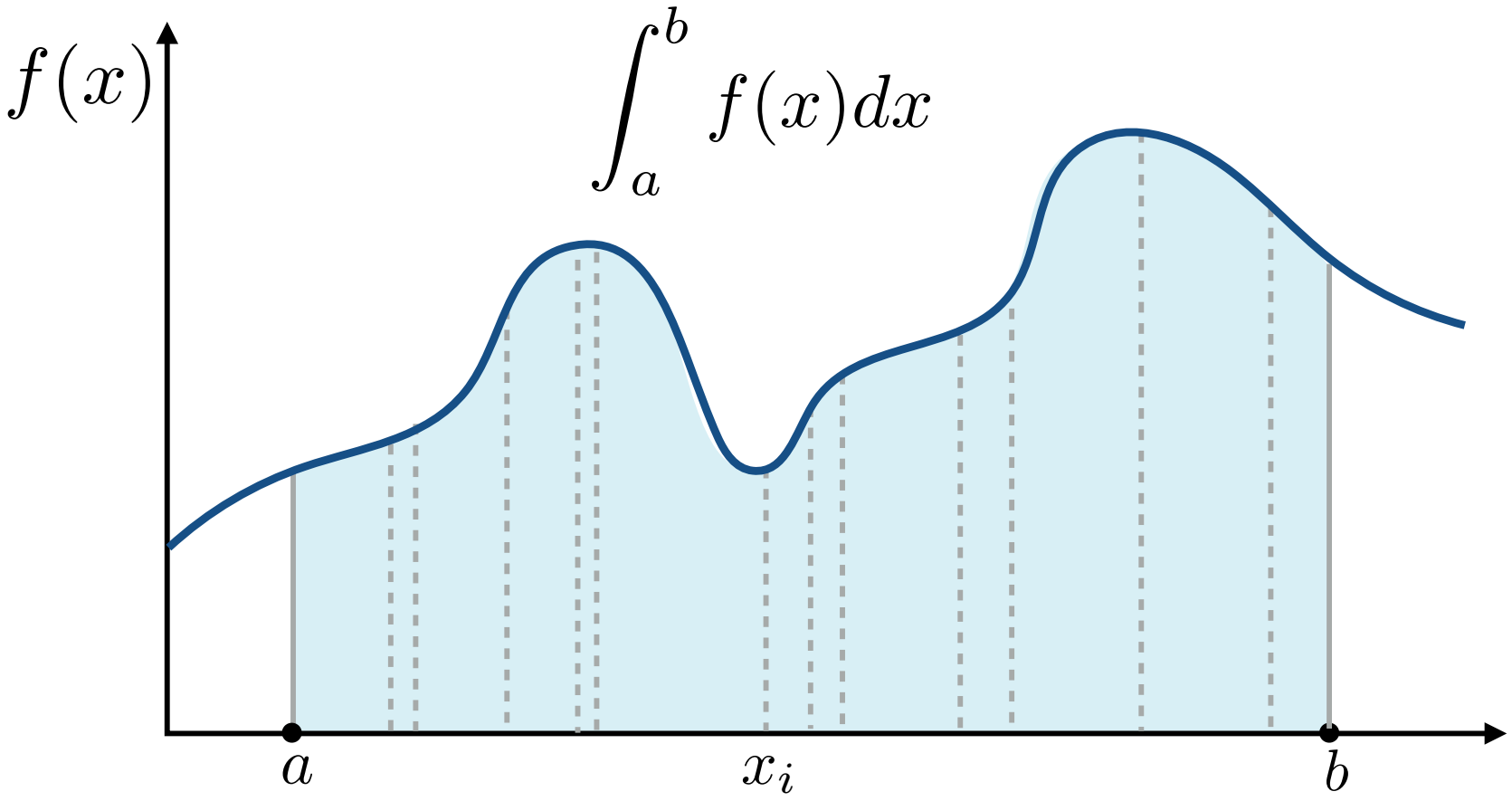
...

- For many problems in graphics (like rendering), k is very, very big (e.g., tens or hundreds or thousands)
- Applying trapezoid rule does not scale!
- Need a fundamentally different approach...



Monte Carlo Integration

Simple idea: estimate the integral of a function by averaging random samples of the function's value.



Monte Carlo Integration

Let us define the Monte Carlo estimator for the definite integral of given function $f(x)$

Definite integral $$\int_a^b f(x)dx$$

Random variable $$X_i \sim p(x)$$

Note: p(x) must be nonzero for all x where f(x) is nonzero

Monte Carlo estimator $$F_N = \frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)}$$

Review: Random Variables

X **Random variable. Represents a distribution of potential values**

$X \sim p(x)$ **Probability density function (PDF). Describes relative probability of a random process choosing value** x

Uniform PDF: all values over a domain are equally likely

e.g., for an unbiased die

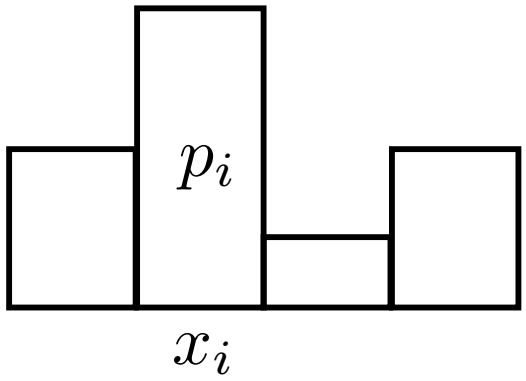
X **takes on values 1,2,3,4,5,6**

$$p(1) = p(2) = p(3) = p(4) = p(5) = p(6)$$

Discrete Probability Distributions

n discrete values x_i

With probability p_i



Requirements of a PDF:

$$p_i \geq 0$$

$$\sum_{i=1}^{n} p_i = 1$$

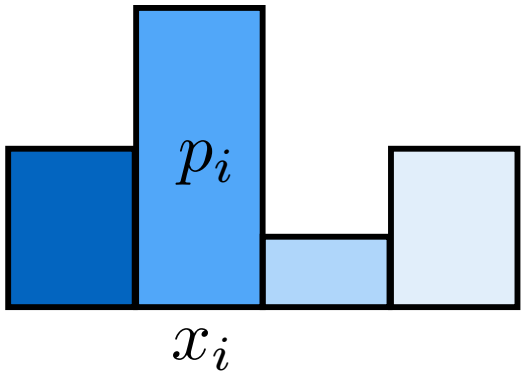
Six-sided die example: $p_i = \frac{1}{6}$

Think: p_i **is the probability that a random measurement of** X **will yield the value** x_i

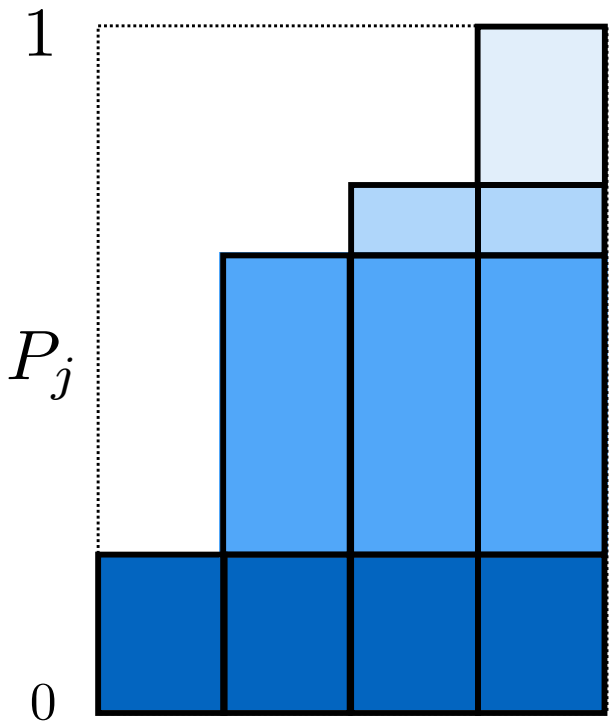
X **takes on the value** x_i **with probability** p_i

Cumulative Distribution Function (CDF)

(For a discrete probability distribution)



Cumulative PDF: $P_j = \sum_{i=1}^{j} p_i$

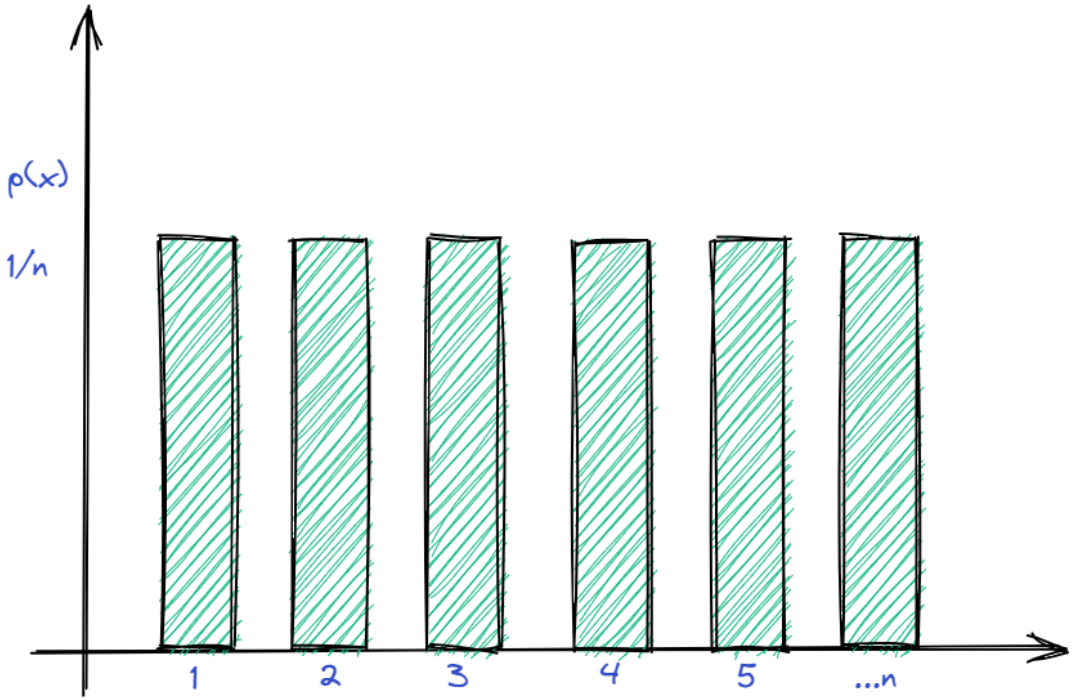


Where:

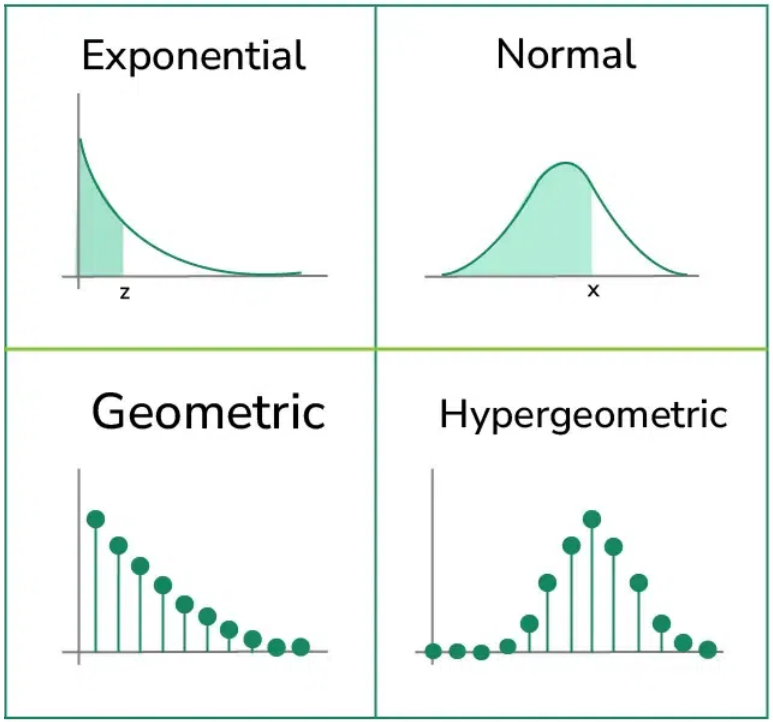
$$0 \leq P_i \leq 1$$

$$P_n = 1$$

How do we generate samples of a discrete random variable (with a known PDF?)



$u \sim \text{Uniform}(0, 1)$



Idea: From uniform random → any distribution

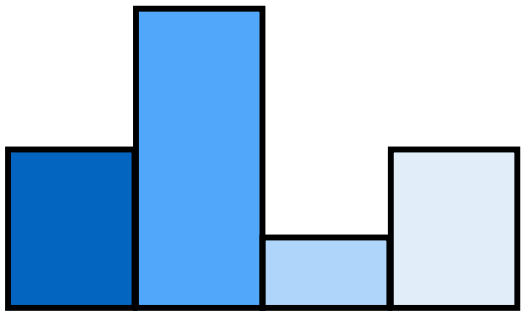
Sampling From Discrete Probability Distributions

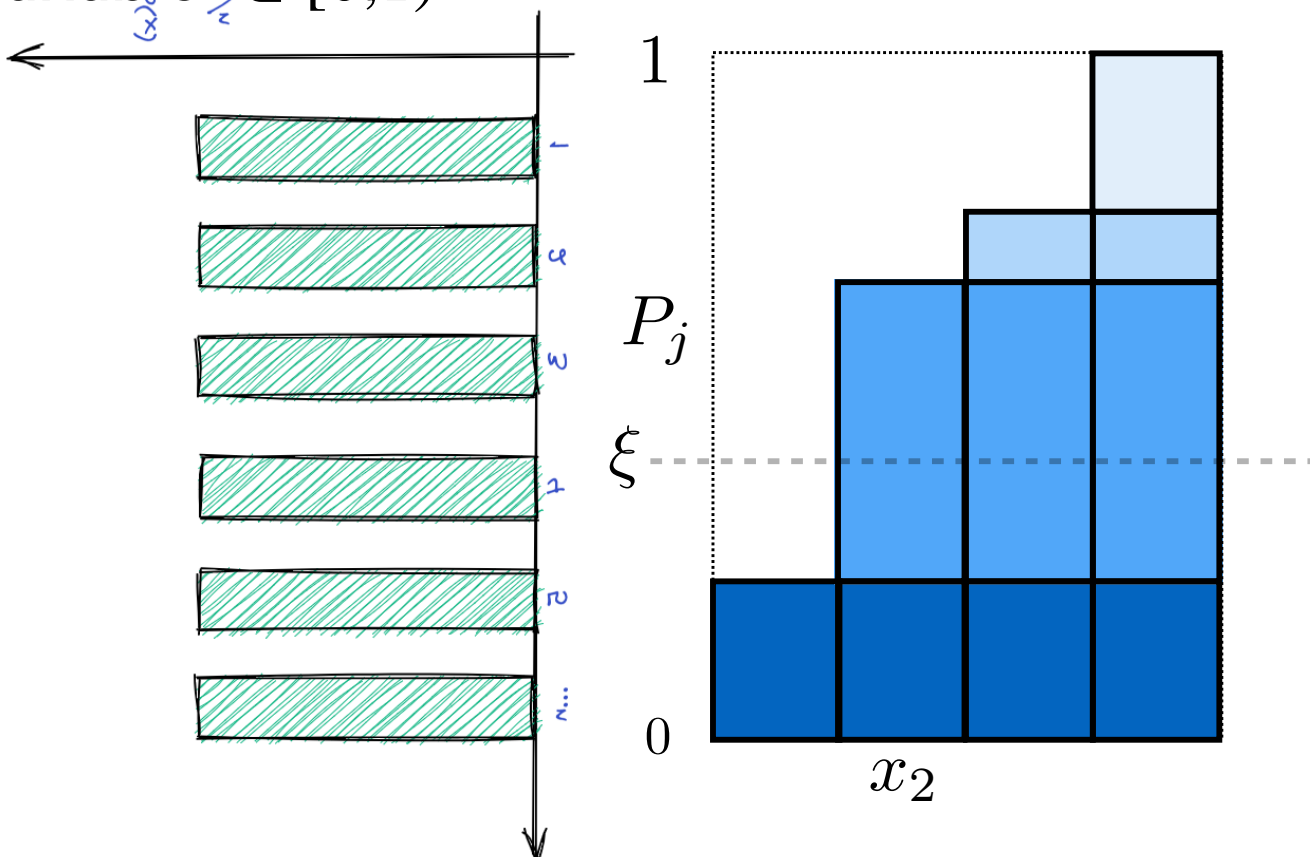
To randomly select an event,

Select x_i if

$$P_{i-1} < \xi \leq P_i$$

Uniform random variable $\in [0,1)$





Continuous Probability Distributions

PDF $p(x)$

$$p(x) \geq 0$$

CDF $P(x)$

$$P(x) = \int_0^x p(x)\,\mathrm{d}x$$

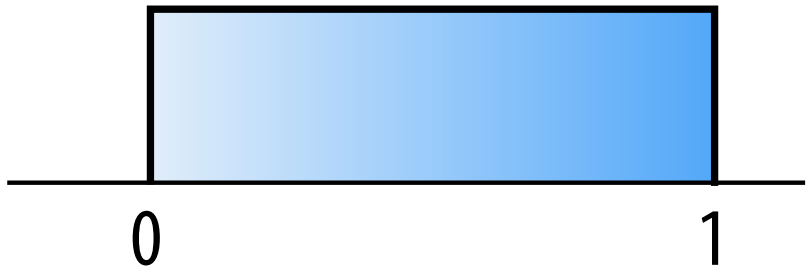
$$P(x) = \Pr(X < x)$$

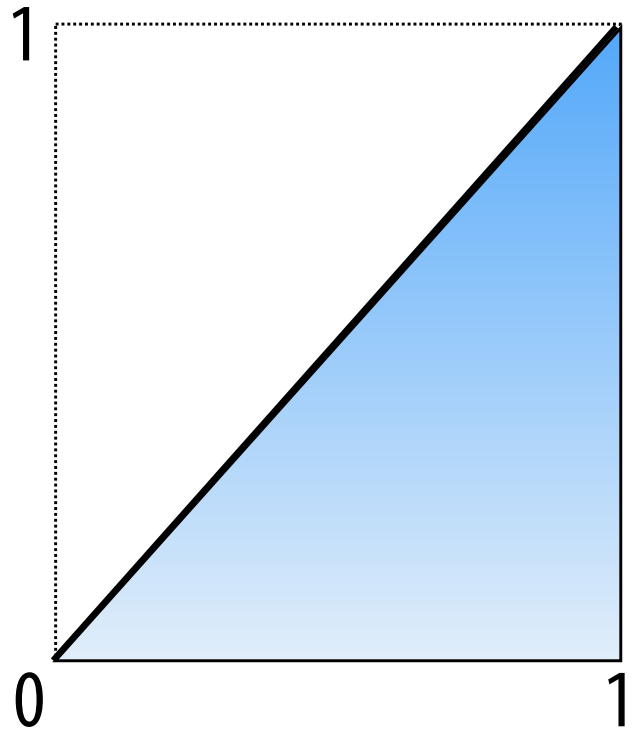
$$P(1) = 1$$

$$\Pr(a \leq X \leq b) = \int_a^b p(x)\,\mathrm{d}x$$

$$= P(b) - P(a)$$

Uniform distribution
(For random variable $X \in [0,1]$)





Sampling continuous random variables using the inversion method

Cumulative probability distribution function

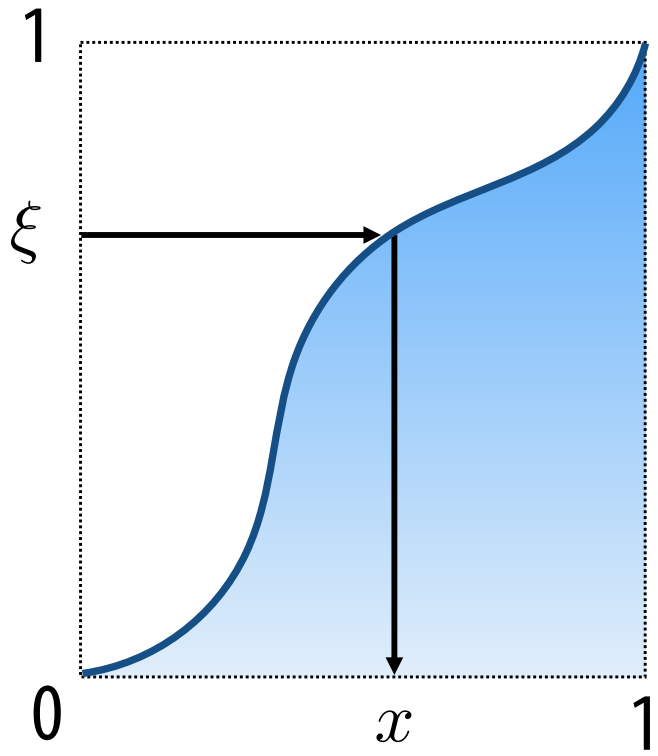
$$P(x) = \Pr(X < x)$$

Construction of samples:

Solve for $x = P^{-1}(\xi)$

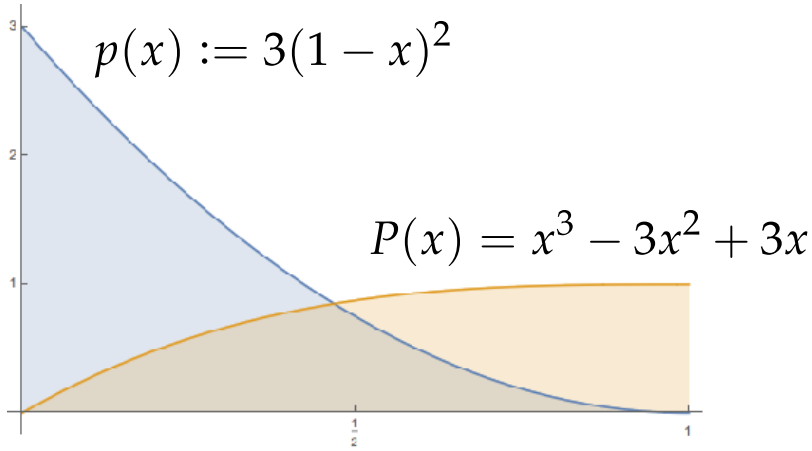
Must know the formula for:

1. The integral of $p(x)$
2. The inverse function $P^{-1}(x)$



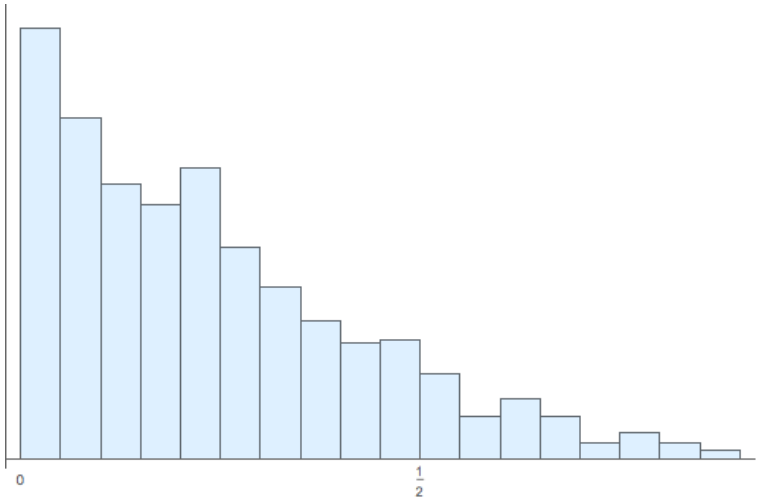
Example — Sampling Quadratic Distribution

- As a toy example, consider the simple probability distribution $p(x) = 3(1-x)^2$ over the interval [0,1]
- How do we pick random samples distributed according to $p(x)$?
- First, integrate probability distribution $p(x)$ to get cumulative distribution $P(x)$
- Invert $P(x)$ by solving $y = P(x)$ for x
- Finally, plug uniformly distributed random values y in [0,1] into this expression

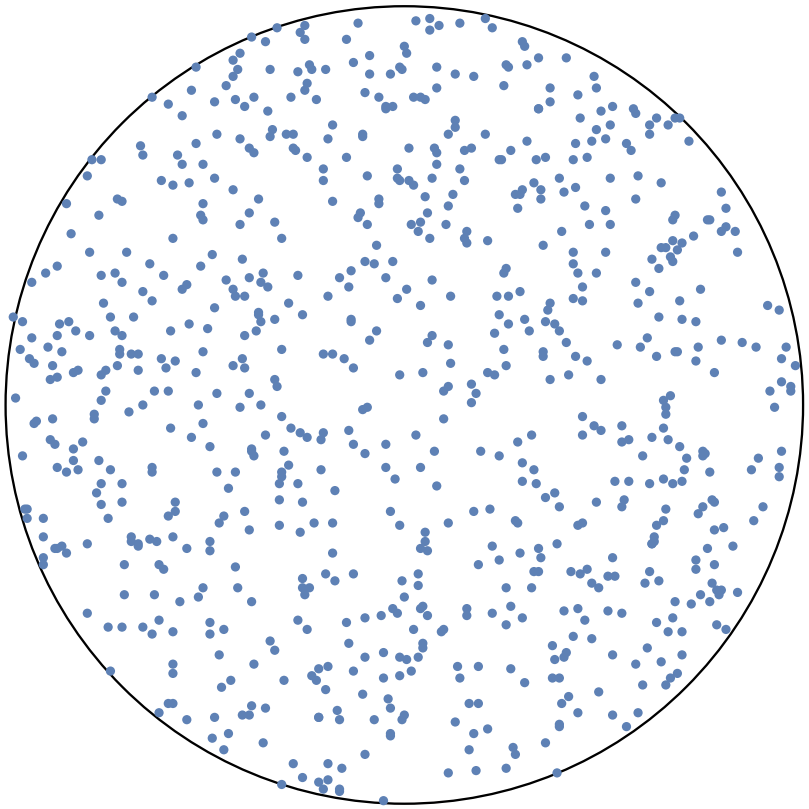


$$\int_0^s 3(1-x)^2\,dx = s^3 - 3s^2 + 3s$$

$$x = 1 - (1-y)^{\frac{1}{3}}$$



How do we uniformly sample the unit circle?

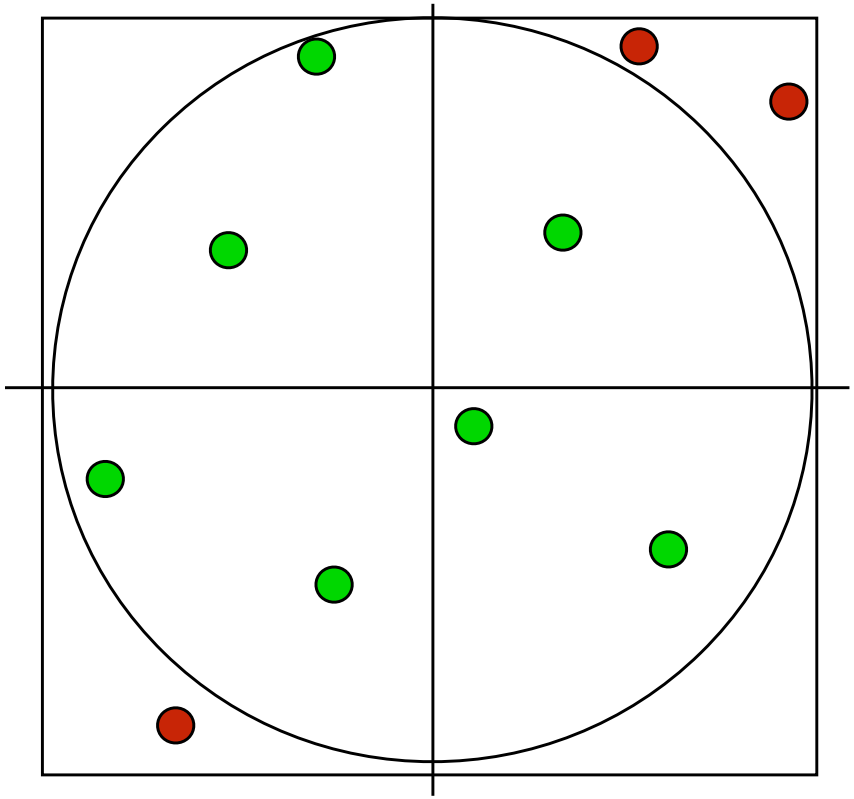


I.e., choose any point P=(px,py) in circle with equal probability

Uniform Sampling via Rejection Sampling

Pick uniform samples in square (easy)

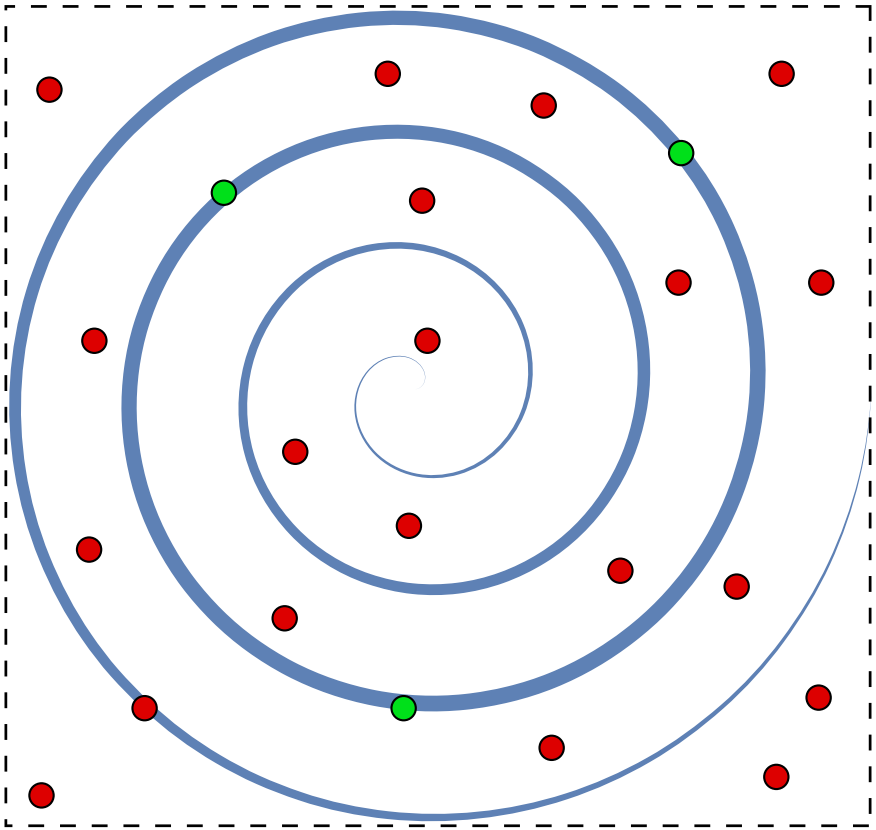
Then toss out any samples not in circle (easy)



Efficiency of technique: area of circle / area of square

Efficiency of via Rejection Sampling

- If the region we care about covers only a very small fraction of the region we've sampling, rejection is probability a bad idea:

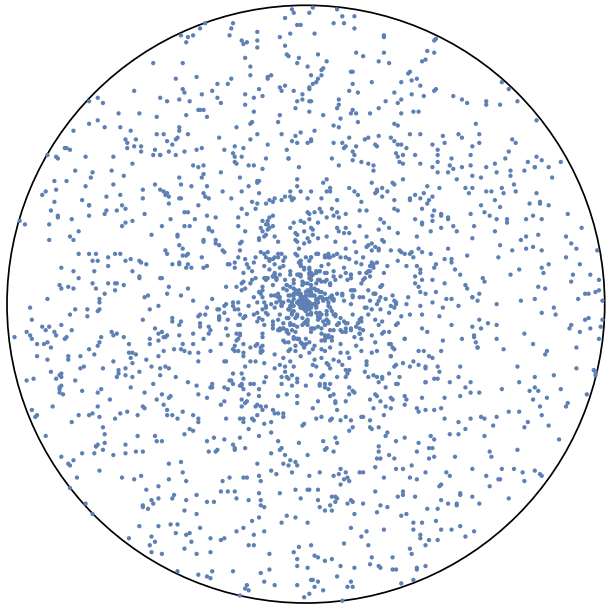


Smarter in this case to "warp" our random variables to follow the spiral.

Uniformly Sampling Unit Circle

- θ = uniform random angle between 0 and 2π
- r = uniform random radius between 0 and 1
- Return point: $(r\cos(\theta), r\sin(\theta))$

This algorithm does not produce the desire uniform sampling of the area of a circle. Why?

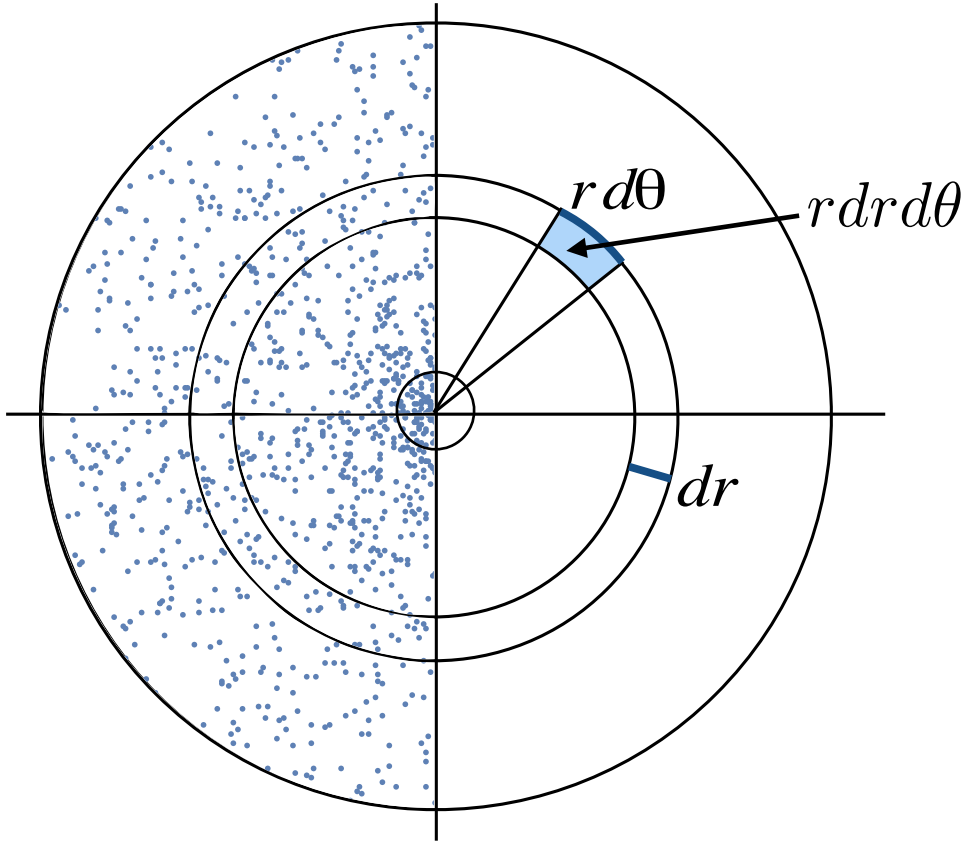


$$\theta = 2\pi\xi_1$$

$$r = \xi_2$$

Because sampling is not uniform in area!

Points farther from center of circle are less likely to be chosen



$$\theta = 2\pi\xi_1 \qquad r = \xi_2$$

So how should we pick samples? Well, think about how we integrate over a disk in polar coordinates…

Uniformly Sampling Unit Circle: via Inversion in 2D

$$\int_0^{2\pi}\int_0^1 p(r,\theta)\,\mathrm{d}r\,\mathrm{d}\theta = 1$$

$$A = \int_0^{2\pi}\int_0^1 r\,\mathrm{d}r\,\mathrm{d}\theta = \int_0^1 r\,\mathrm{d}r\int_0^{2\pi}\mathrm{d}\theta = \left(\frac{r^2}{2}\right)\Big|_0^1 \theta\Big|_0^{2\pi} = \pi$$

$$p(r,\theta) = \frac{r}{\pi}$$

$$p(r,\theta) = p(r)p(\theta) \longleftarrow r,\theta \text{ independent}$$

$$p(\theta) = \frac{1}{2\pi}$$

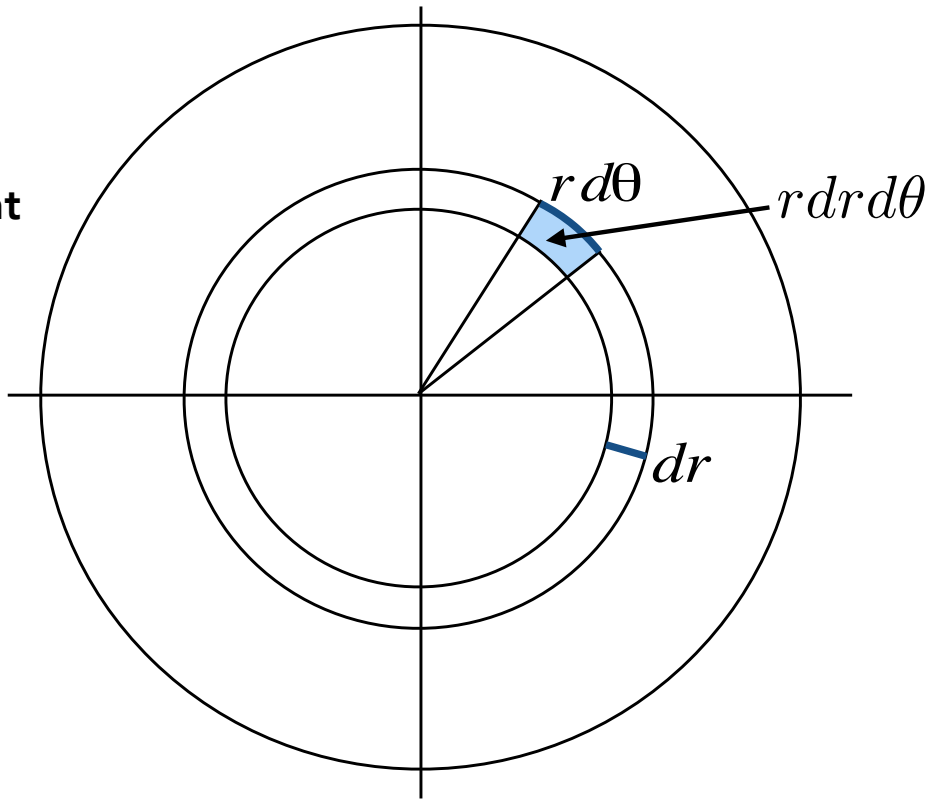
$$P(\theta) = \frac{1}{2\pi}\theta$$

$$p(r) = 2r$$

$$P(r) = r^2$$

$$\theta = 2\pi\xi_1$$

$$r = \sqrt{\xi_2}$$

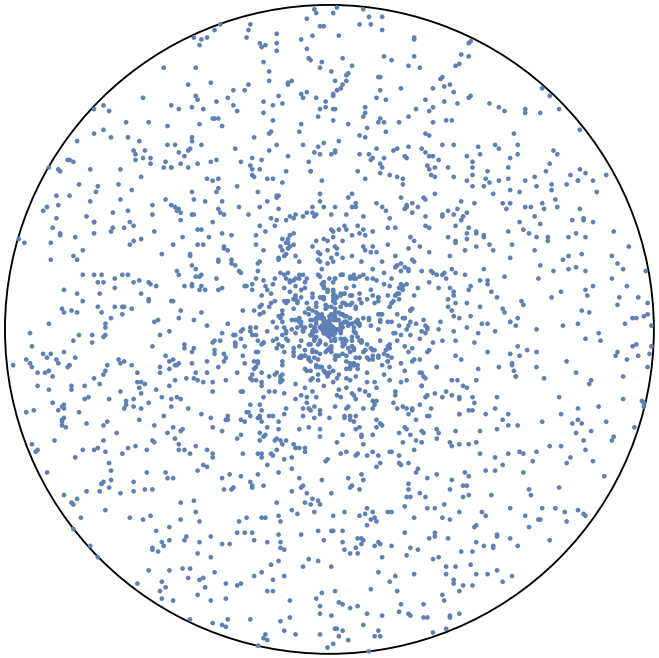


Uniformly Area Sampling of Unit Circle

WRONG

Probability is uniform;

Samples are not!



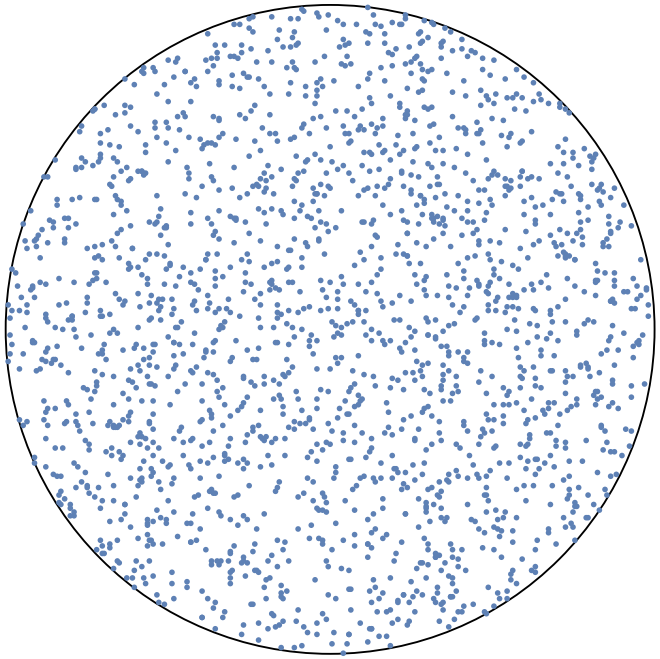
$$\theta = 2\pi\xi_1$$

$$r = \xi_2$$

RIGHT

Probability is nonuniform;

Samples are uniform!



$$\theta = 2\pi\xi_1$$

$$r = \sqrt{\xi_2}$$

Monte Carlo Integration

Let us define the Monte Carlo estimator for the definite integral of given function $f(x)$

Definite integral $$\int_a^b f(x)dx$$

Random variable $$X_i \sim p(x)$$

Note: p(x) must be nonzero for all x where f(x) is nonzero

Monte Carlo estimator $$F_N = \frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)}$$

Example: Basic Monte Carlo Estimator

The basic Monte Carlo estimator is a simple special case where we sample with a uniform random variable

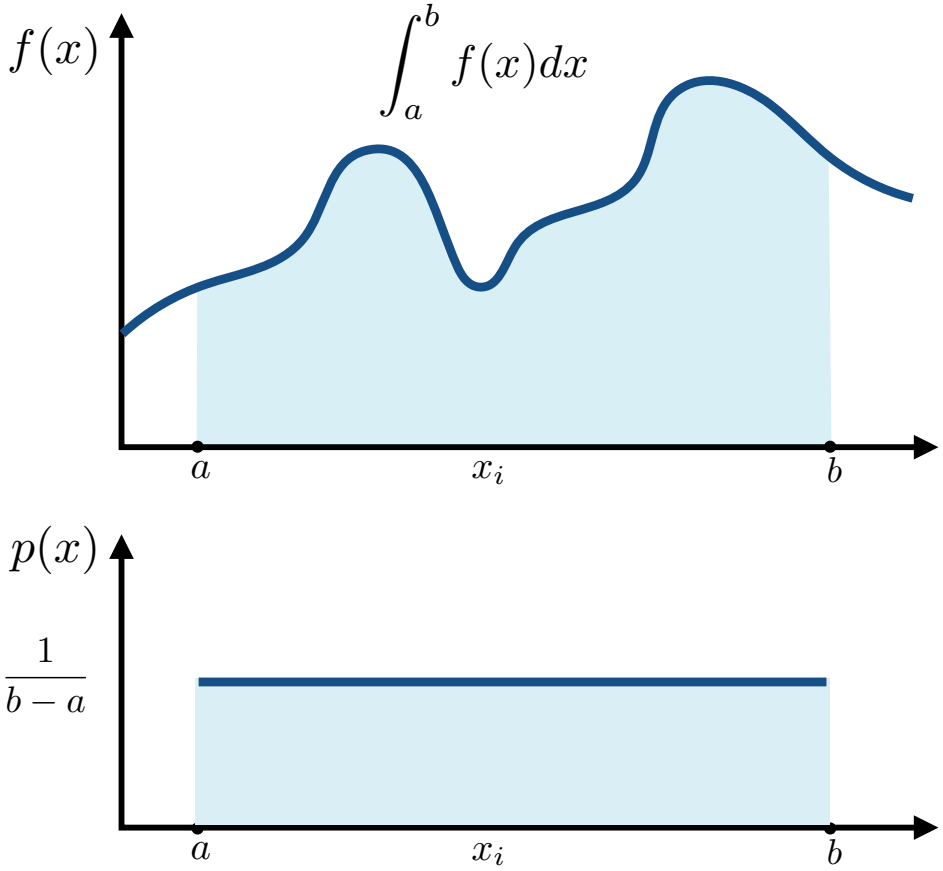
Uniform random variable

$$X_i \sim p(x) = C \quad \textbf{(constant)}$$

$$\int_a^b p(x)\,dx = 1$$

$$\implies \int_a^b C\,dx = 1$$

$$\implies C = \frac{1}{b-a}$$



Example: Basic Monte Carlo Estimator

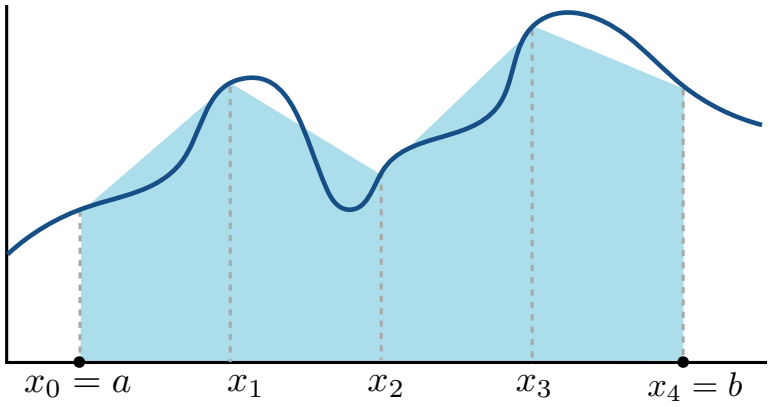
The basic Monte Carlo estimator is a simple special case where we sample with a uniform random variable

Basic Monte Carlo estimator (derivation)

$$F_N = \frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)} \quad \text{(MC Estimator)}$$

$$= \frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{1/(b-a)}$$

$$= \frac{b-a}{N}\sum_{i=1}^{N} f(X_i)$$



Unbiased Estimator

Definition: A randomized integral estimator is *unbiased* if its expected value is the desired integral.

Fact: the general and basic Monte Carlo estimators are unbiased

Expected Value

Intuition: what value does a random variable take, on average?

- E.g., consider a fair coin where heads=1, tails=0
- Equal probability of head & tails is 1/2 for both
- Expected value is then (1/2)*1+(1/2)*0=1/2

Expected value of random variable Y

Number of possible outcomes

$$E(Y) := \sum_{i=1}^{k} p_i y_i$$

Probability of ith outcome

Value of ith outcome

Properties of expected values:

$$E\left[\sum_i Y_i\right] = \sum_i E[Y_i]$$

$$E[aY] = aE[Y]$$

Proof That Monte Carlo Estimator Is Unbiased

$$
\begin{aligned}
E[F_N] &= E\left[\frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)}\right] \\
&= \frac{1}{N}\sum_{i=1}^{N} E\left[\frac{f(X_i)}{p(X_i)}\right] \\
&= \frac{1}{N}\sum_{i=1}^{N}\int_a^b \frac{f(x)}{p(x)}\,p(x)\mathrm{d}x \\
&= \frac{1}{N}\sum_{i=1}^{N}\int_a^b f(x)\,\mathrm{d}x \\
&= \int_a^b f(x)\,\mathrm{d}x
\end{aligned}
$$

Properties of expected values:

$$E\left[\sum_i Y_i\right] = \sum_i E[Y_i]$$

$$E[aY] = aE[Y]$$

The expected value of the Monte Carlo estimator is the desired integral.

Variance

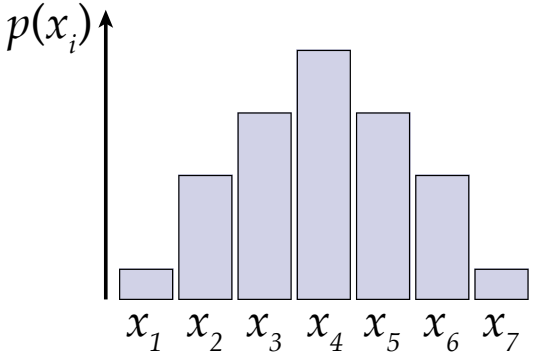
Intuition: how far are our samples from the average, on average?

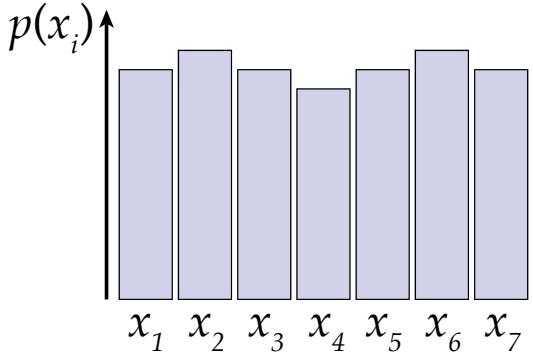
Definition

$$V[Y] = E[(Y - E[Y])^2]$$

$$\mathrm{Var}[F_N] = \frac{1}{N}\left[\int_a^b \frac{f(x)^2}{p(x)}\,dx - \left(\int_a^b f(x)\,dx\right)^2\right]$$

Q: which of these has higher variance?





方差的大小取决于f(x) 与采样分布p(x)的匹配程度。若p(x) 很小而 f(x) 大，方差增加

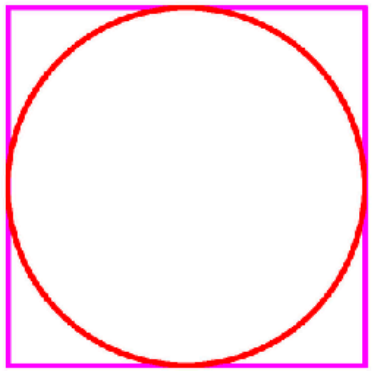
Law of Large Numbers 大数定律

- Important fact: for any random variable, the average value of N trials approaches the expected value as we increase N
- Decrease in variance is always linear in N:

$$V\left[\frac{1}{N}\sum_{i=1}^{N} Y_i\right] = \frac{1}{N^2}\sum_{i=1}^{N} V[Y_i] = \frac{1}{N^2} N\, V[Y] = \frac{1}{N} V[Y]$$

Consider computation of π :

nCoconuts	estimate of π
1	4.000000
10	3.200000
100	3.240000
1000	3.112000
10000	3.163600
100000	3.139520
1000000	3.141764



Properties of variance:

$$V[Y] = E[Y^2] - E[Y]^2$$

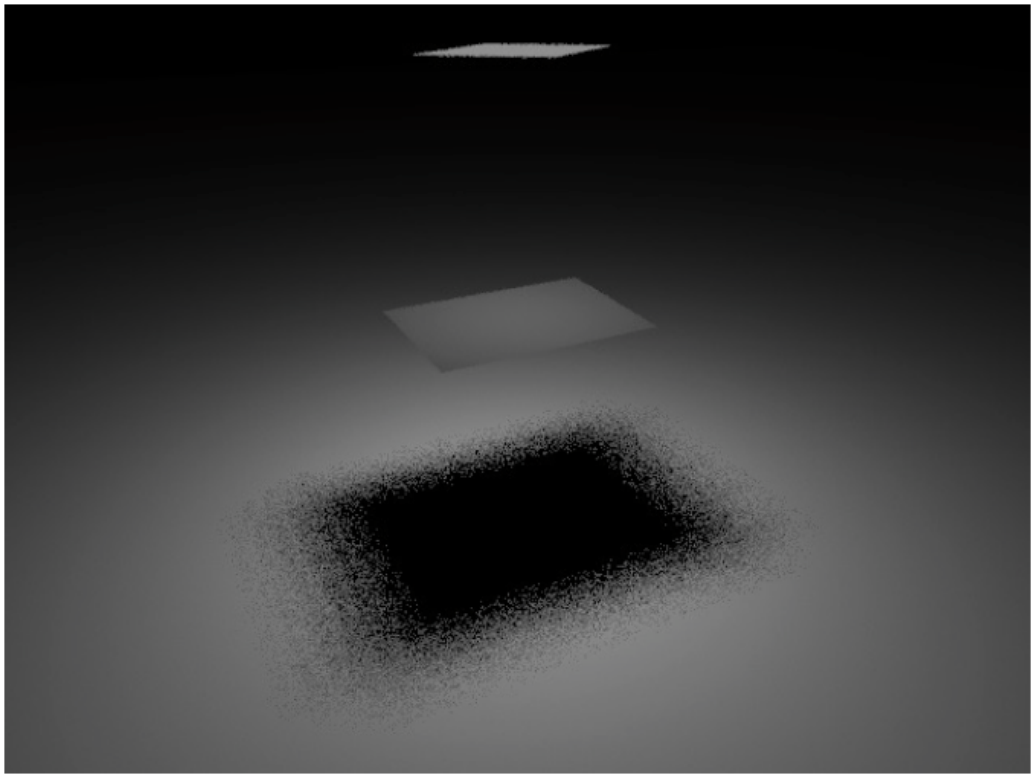
$$V\left[\sum_{i=1}^{N} Y_i\right] = \sum_{i=1}^{N} V[Y_i]$$

$$V[aY] = a^2\, V[Y]$$

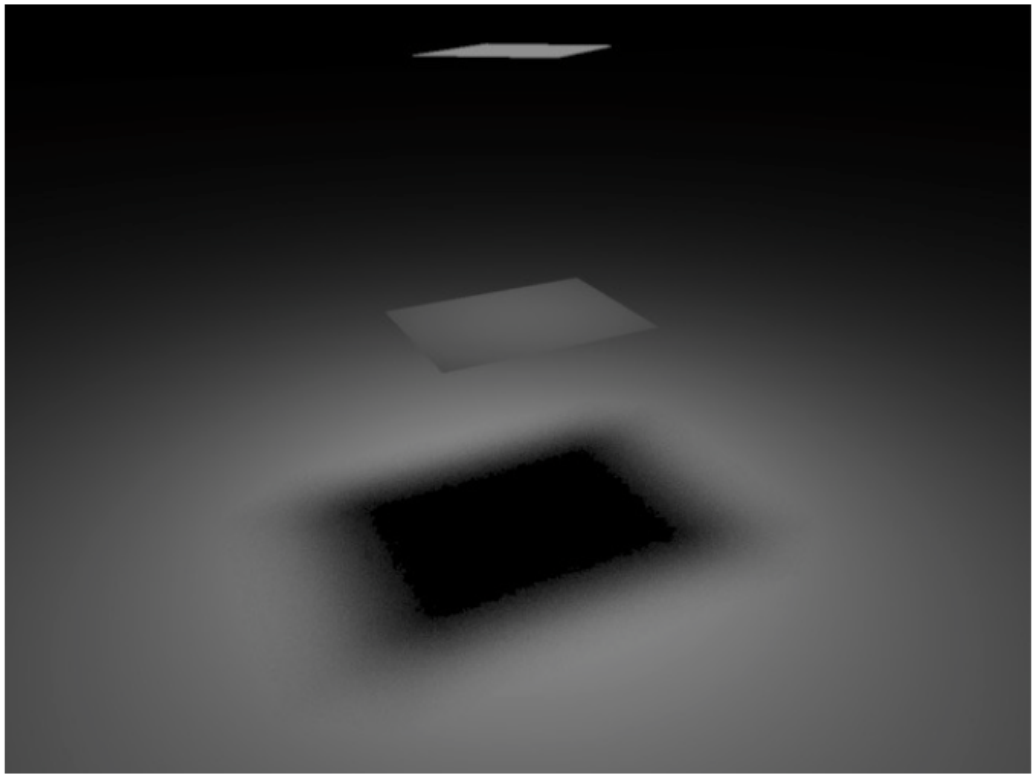
Q: Why is the law of large numbers important for Monte Carlo ray tracing?

A: No matter how hard the integrals are (complicated lighting, geometry, materials, etc.), can always get the right image by taking more samples.

More Random Samples Reduces Variance



1 shadow ray

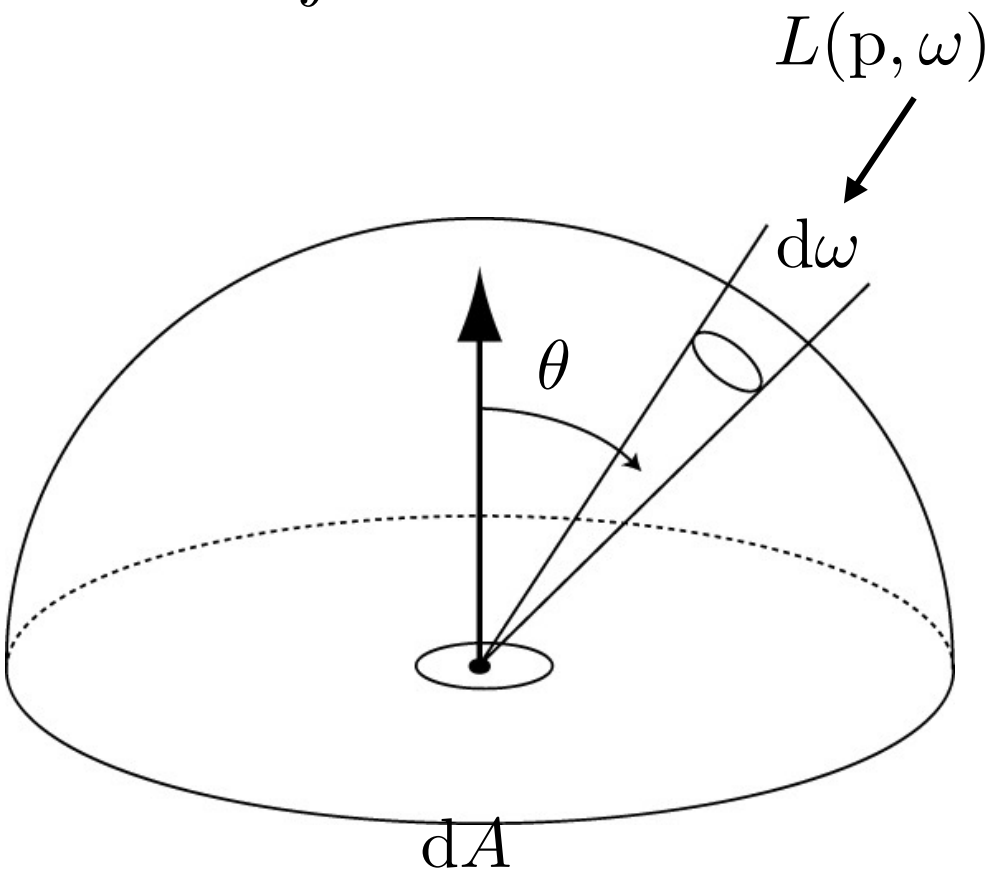


16 shadow rays

Example: Monte Carlo Estimate Of Direct Lighting Integral

Direct Lighting (Irradiance) Estimate

$$E(\mathrm{p}) = \int L(\mathrm{p}, \omega)\ \cos\theta\, \mathrm{d}\omega$$



Idea: sample directions over hemisphere uniformly in solid angle

Estimator:

$$X_i \sim p(\omega) \qquad p(\omega) = \frac{1}{2\pi}$$

$$Y_i = f(X_i)$$

$$Y_i = L(\mathrm{p}, \omega_i) cos\, \theta_i$$

$$F_N = \frac{2\pi}{N} \sum_{i=1}^{N} Y_i$$

Direct Lighting (Irradiance) Estimate

Sample directions over hemisphere uniformly in solid angle

$$E(\mathrm{p}) = \int L(\mathrm{p}, \omega)\ \cos\theta\, \mathrm{d}\omega$$

Given surface point *p*

Initialize Monte Carlo estimator F_N to 0

For each of N samples:

A ray tracer evaluates radiance along a ray

Generate random direction: ω_i

Compute incoming radiance L_i arriving at *p* from direction ω_i

Increment the Monte Carlo estimator: $F_N := F_N + \frac{2\pi}{N} L_i \cos\theta_i$

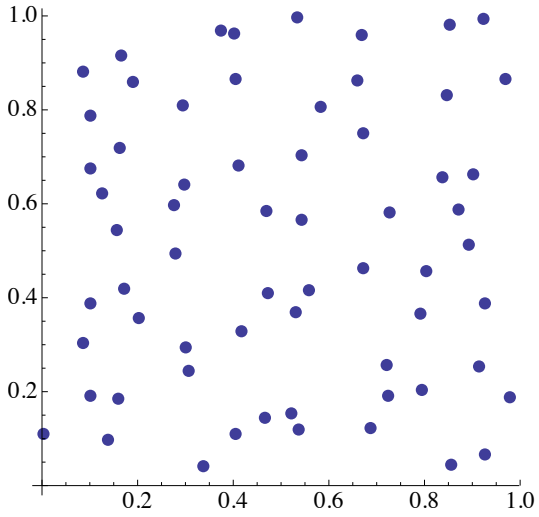
Random Direction: Picking Points on Unit Hemisphere

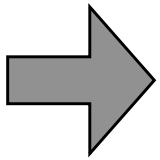
How do we uniformly sample directions from the hemisphere?

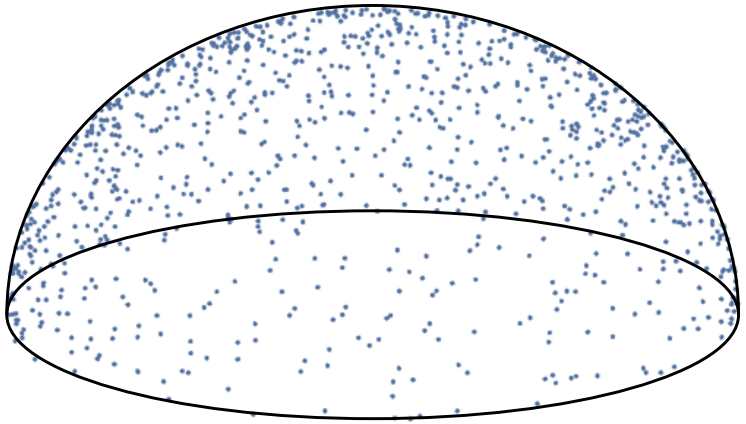
One way: use rejection sampling. (How?)

Another way: "warp" two values in [0,1] via the inversion method:

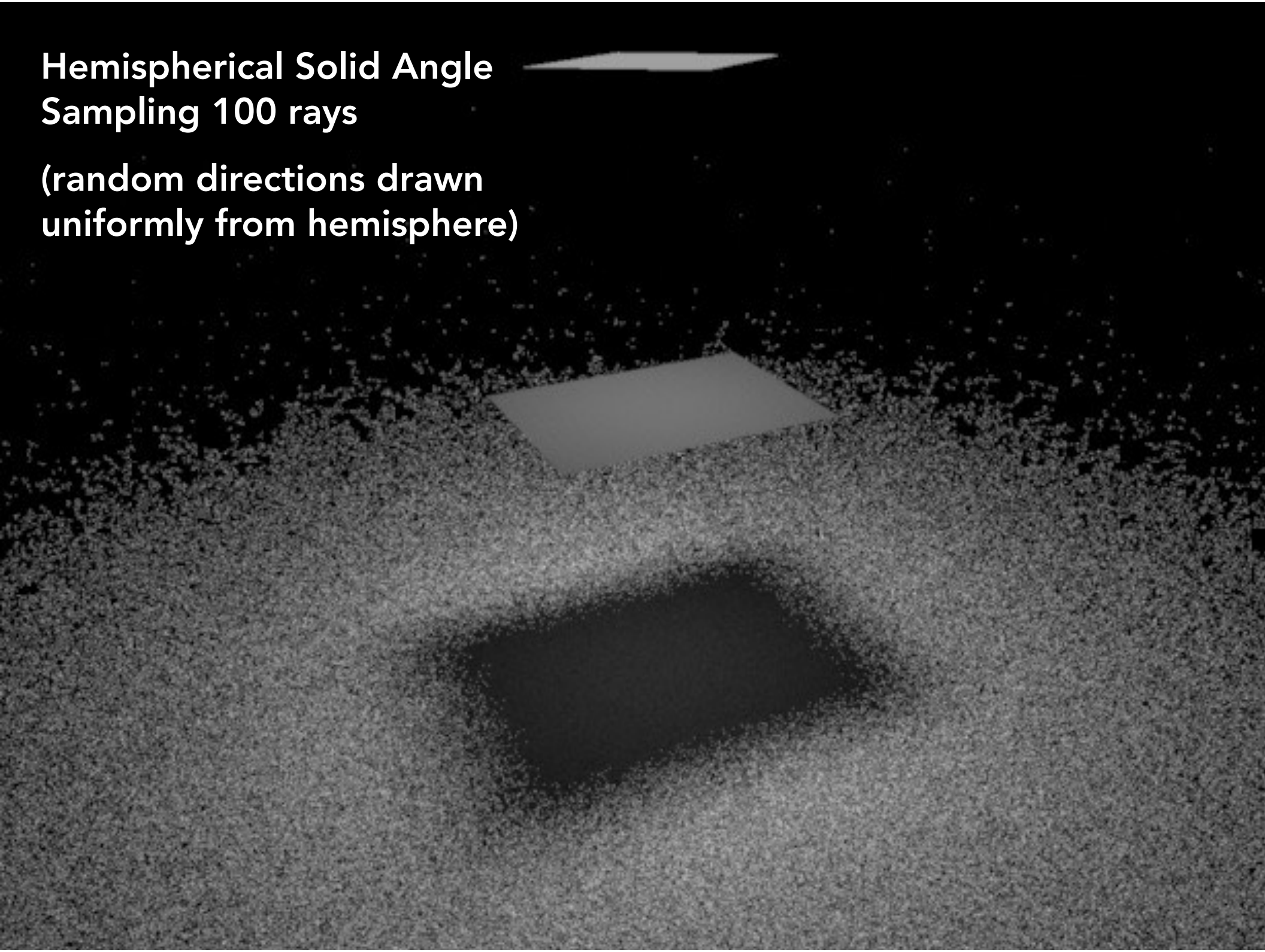
$$(\xi_1, \xi_2) = (\sqrt{1-\xi_1^2}\cos(2\pi\xi_2), \sqrt{1-\xi_1^2}\sin(2\pi\xi_2), \xi_1)$$

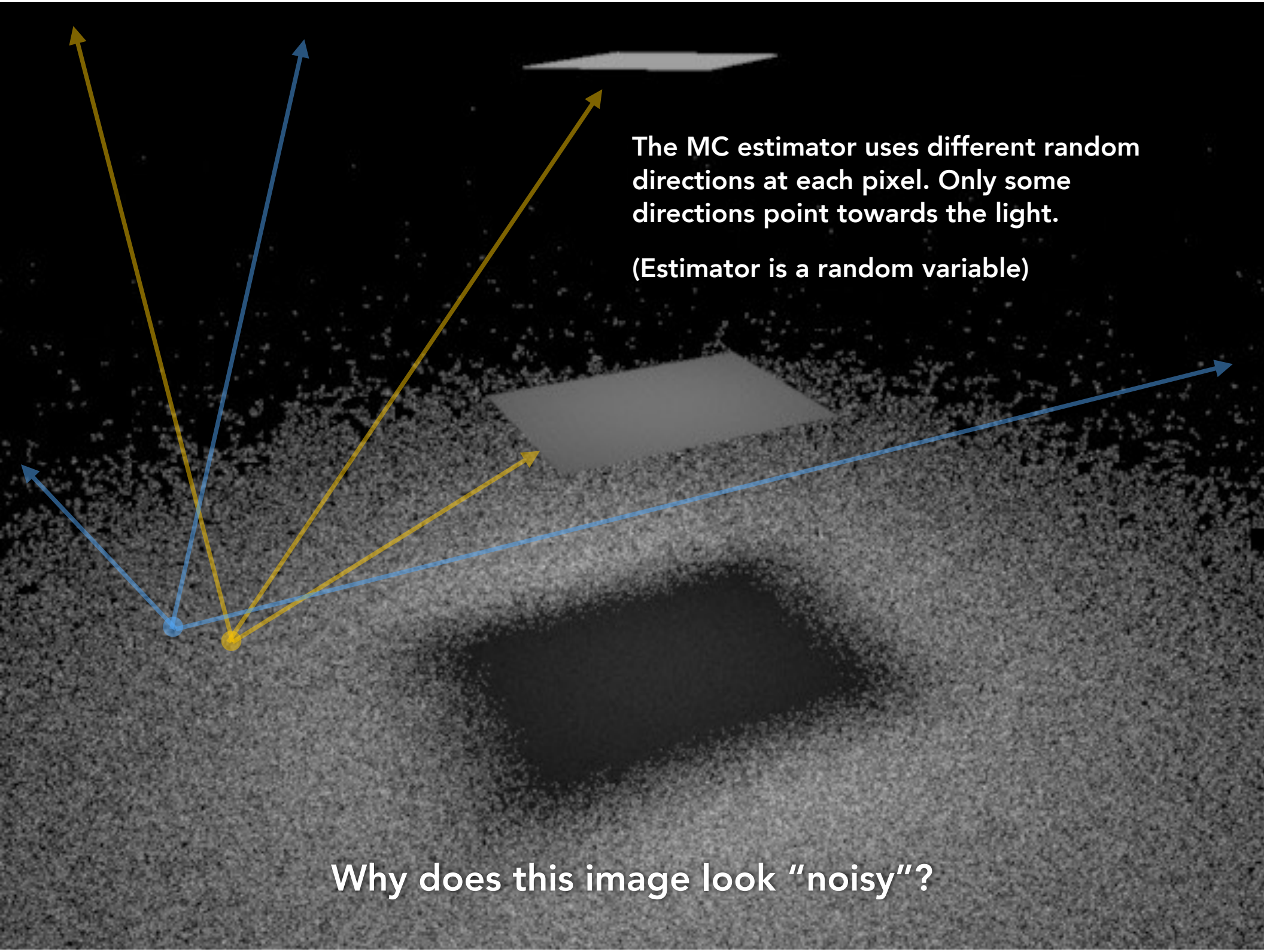






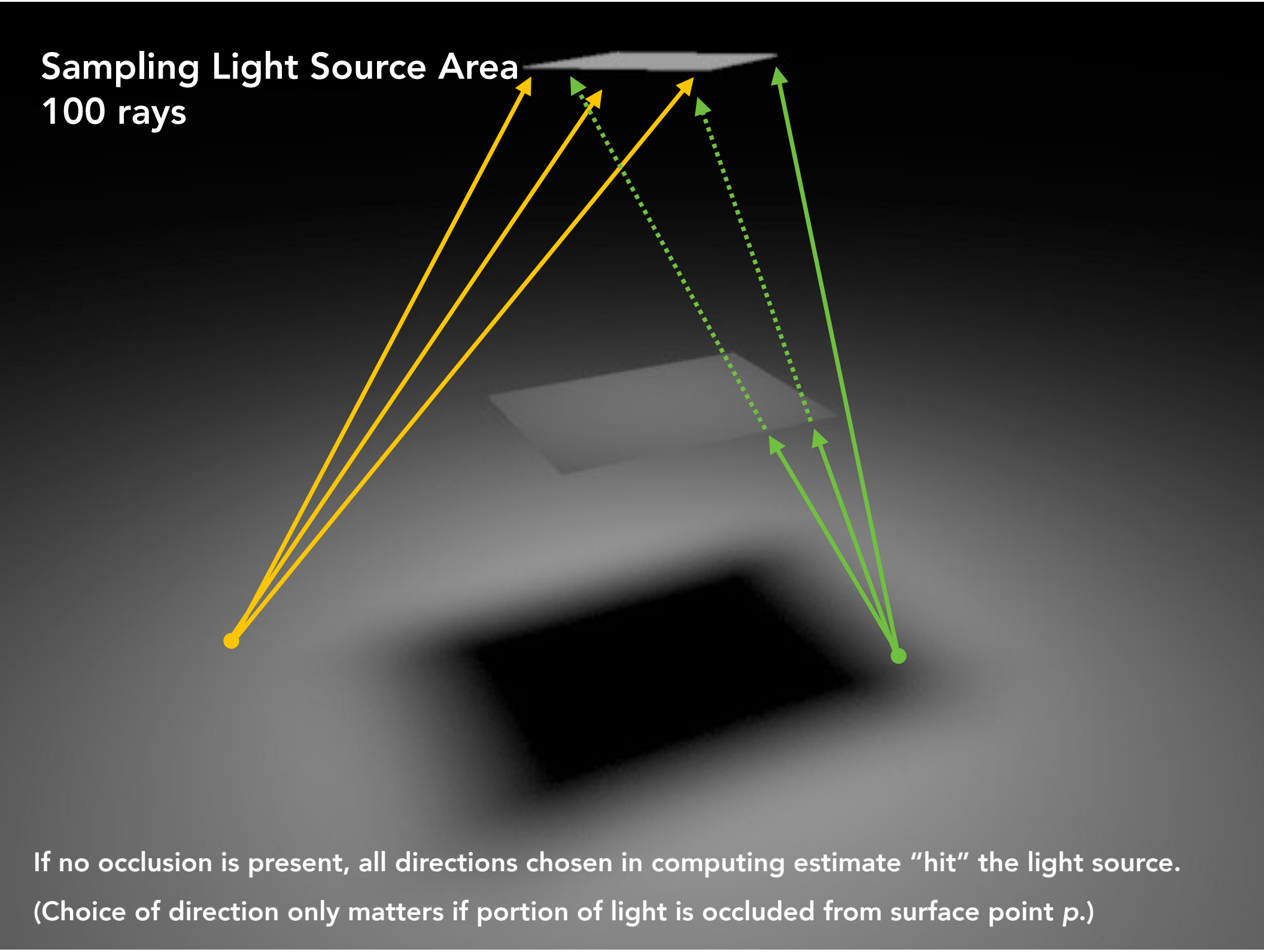
Exercise: derive from the inversion method


Hemispherical Solid Angle
Sampling 100 rays
(random directions drawn
uniformly from hemisphere)


The MC estimator uses different random directions at each pixel. Only some directions point towards the light.
(Estimator is a random variable)
Why does this image look “noisy”?

Observation: incoming radiance is zero for most directions in this scene

Idea: integrate only over the area of the light (directions where incoming radiance could be non-zero)

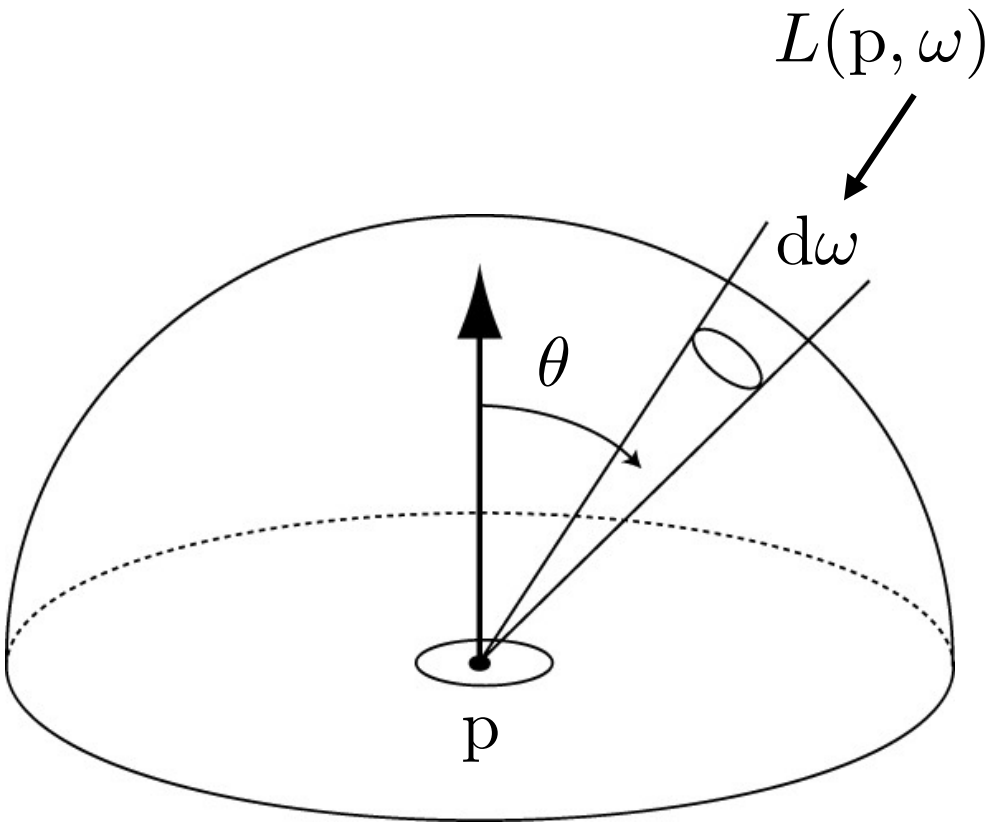


If no occlusion is present, all directions chosen in computing estimate "hit" the light source.

(Choice of direction only matters if portion of light is occluded from surface point *p*.)

Changing Basis of Integration: Sampling Hemisphere

$$E(\mathrm{p}) = \int L(\mathrm{p}, \omega)\, \cos\theta \, \mathrm{d}\omega$$

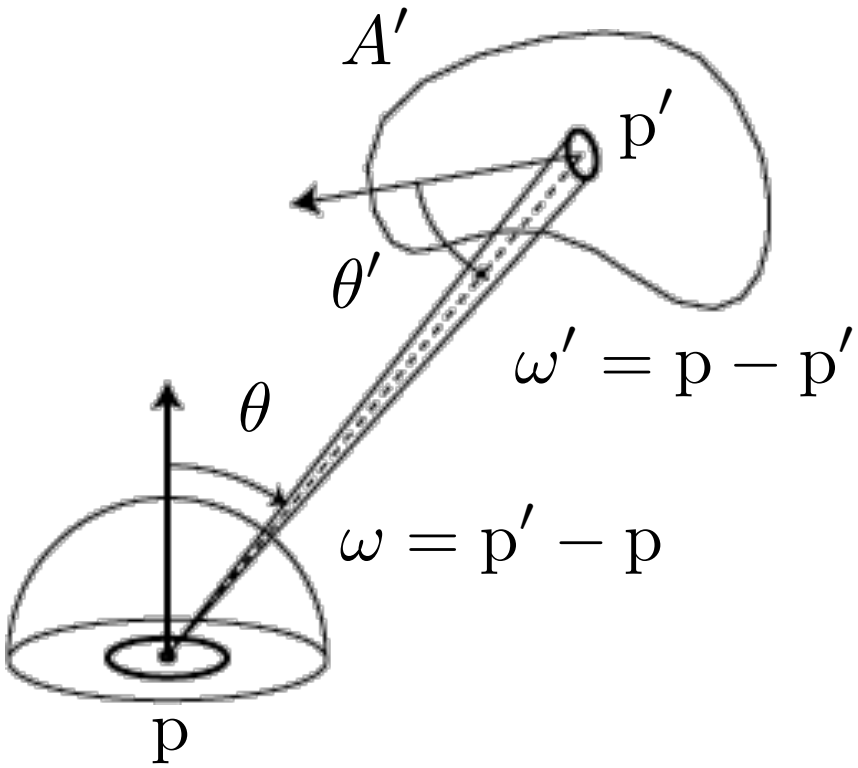


Changing Basis of Integration: Sampling Light Source Area

$$E(\mathrm{p}) = \int_{A'} L_o(\mathrm{p}', \omega')\, V(\mathrm{p}, \mathrm{p}')\, \frac{\cos\theta \cos\theta'}{|\mathrm{p} - \mathrm{p}'|^2}\, \mathrm{d}A'$$

Change of variables to integral over area of light

$$dw = \frac{dA' cos\,\theta'}{|\mathrm{p}' - \mathrm{p}|^2}$$

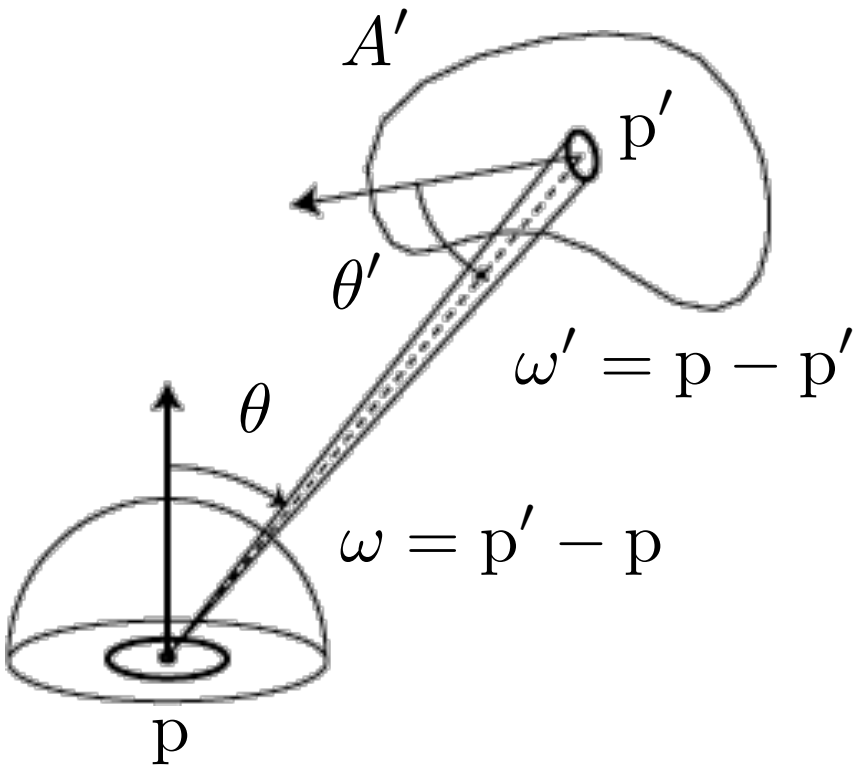


Binary visibility function:
1 if p′ is visible from p, 0 otherwise
(accounts for light occlusion)

Outgoing radiance from light point p, in direction w′ towards p

Monte Carlo Estimate by Sampling Light Source Area

$$E(\mathrm{p}) = \int_{A'} L_o(\mathrm{p}', \omega')\, V(\mathrm{p}, \mathrm{p}')\, \frac{\cos\theta \cos\theta'}{|\mathrm{p} - \mathrm{p}'|^2}\, \mathrm{d}A'$$



Randomly sample light source area A' (assume uniformly over area)

$$\int_{A'} p(\mathrm{p}')\, \mathrm{d}A' = 1$$

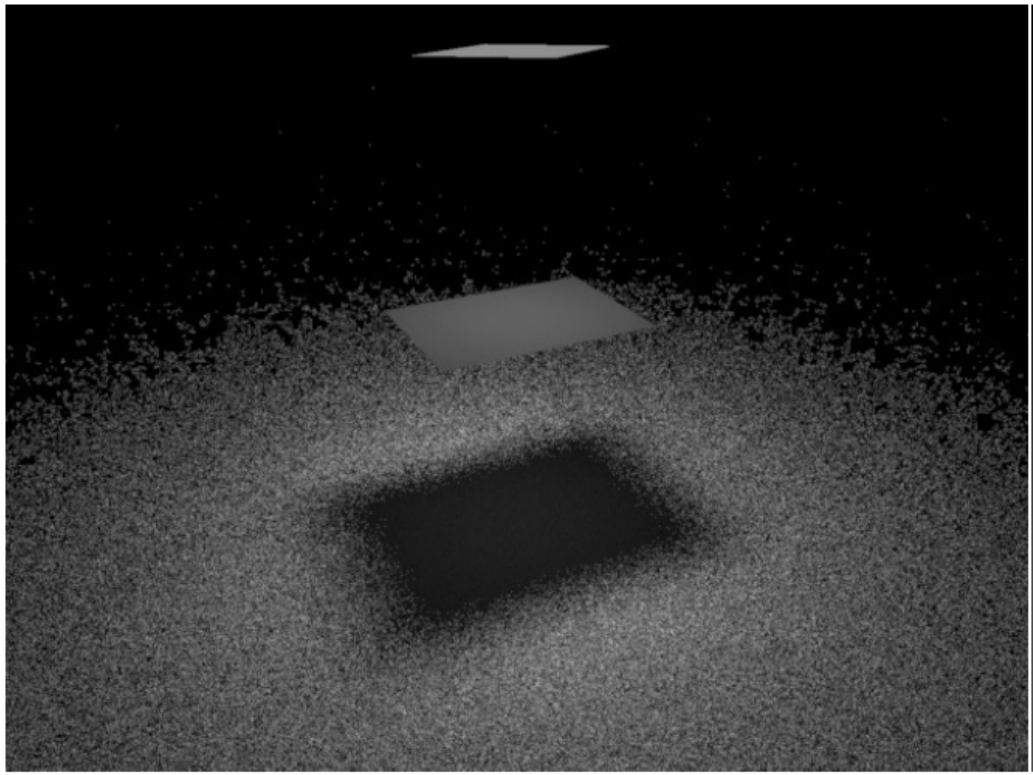
$$p(\mathrm{p}') = \frac{1}{A'}$$

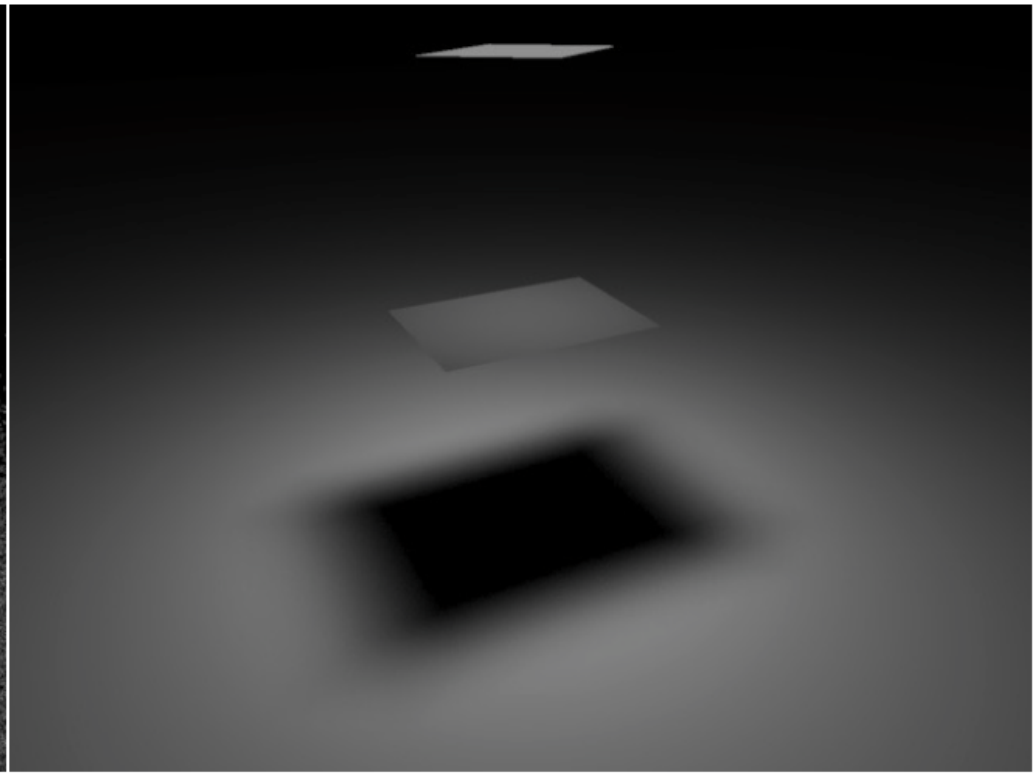
Monte Carlo Estimator

$$F_N = \frac{A'}{N} \sum_{i=1}^{N} Y_i$$

$$Y_i = L_o(\mathrm{p}'_i, \omega'_i)\, V(\mathrm{p}, \mathrm{p}'_i)\, \frac{\cos\theta_i \cos\theta'_i}{|\mathrm{p} - \mathrm{p}'_i|^2}$$

Solid Angle Sampling vs Light Area Sampling





Sampling solid angle

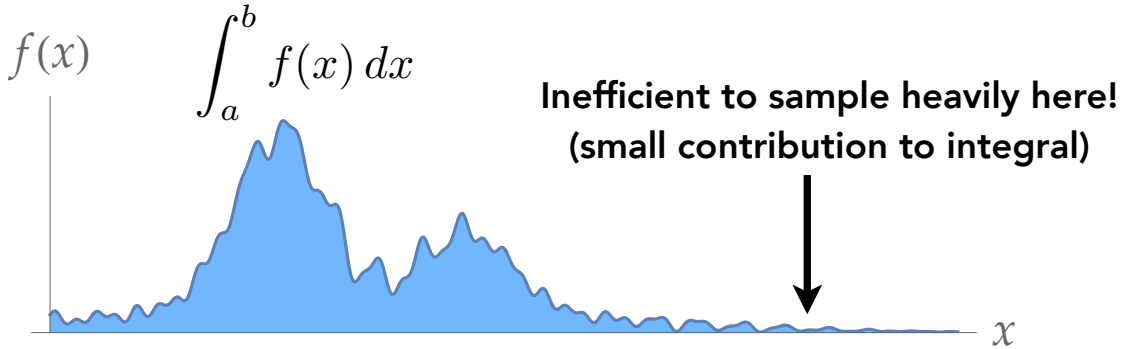
100 random directions on hemisphere

Sampling light source area

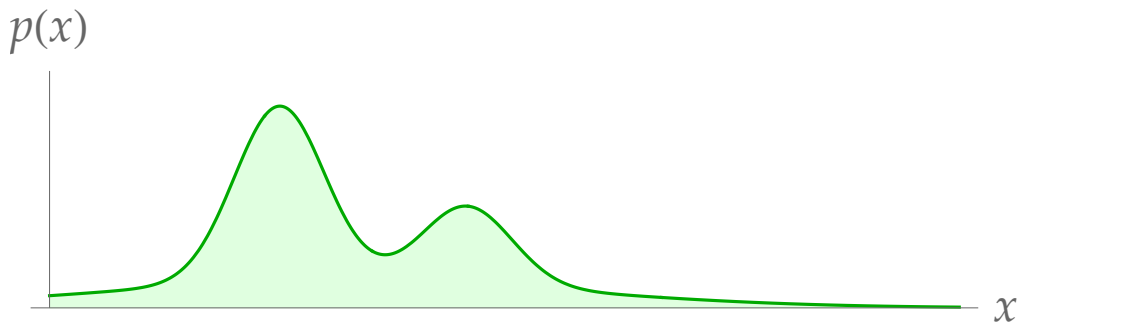
100 random points on area of light source

Importance Sampling

Simple idea: sample the integrand according to how much we expect it to contribute to the integral.



complicated integrand



our best guess for where the integrand is "big"

Note: p(x) must be non-zero where f(x) is non-zero

Basic Monte Carlo:

$$\frac{b-a}{N}\sum_{i=1}^{N} f(X_i)$$

(x_i are sampled *uniformly*)

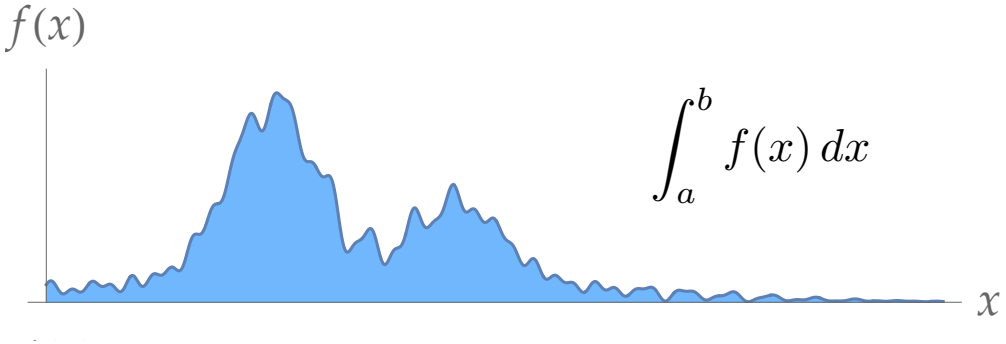
Importance-Sampled Monte Carlo:

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

(x_i are sampled proportional to *p*)

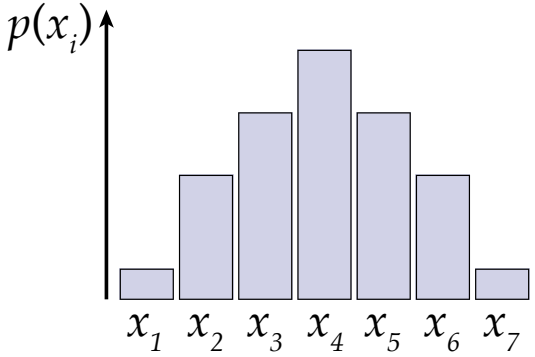
"If I sample x less frequently, each sample should count for more."

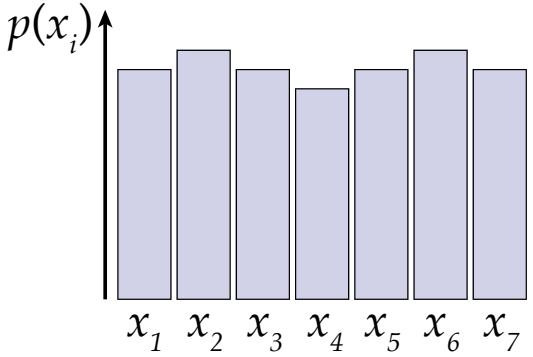
Importance Sampling Can Reduce Variance



$$\mathrm{Var}[F_N] = \frac{1}{N}\left[\int_a^b \frac{f(x)^2}{p(x)}\,dx - \left(\int_a^b f(x)\,dx\right)^2\right]$$

Q: which of these has higher variance?





Importance sampling reduces variance when your sampling distribution approximates the shape of the integrand.

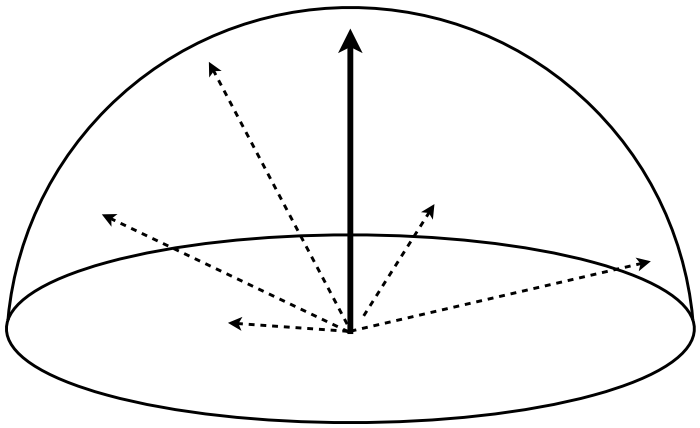
More Importance Sampling — Cosine-Weighted Sampling

Consider uniform hemisphere sampling in irradiance estimate:

$$f(\omega) = L_i(\omega)\cos\theta \qquad\qquad p(\omega) = \frac{1}{2\pi}$$

$$(\xi_1, \xi_2) = (\sqrt{1-\xi_1^2}\cos(2\pi\xi_2), \sqrt{1-\xi_1^2}\sin(2\pi\xi_2), \xi_1)$$

$$\int_\Omega f(\omega)\,\mathrm{d}\omega \approx \frac{1}{N}\sum_i^N \frac{f(\omega)}{p(\omega)} = \frac{1}{N}\sum_i^N \frac{L_i(\omega)\,\cos\theta}{1/2\pi} = \frac{2\pi}{N}\sum_i^N L_i(\omega)\cos\theta$$



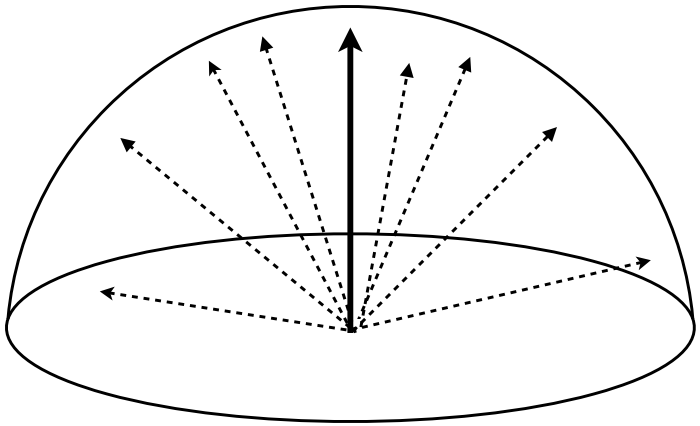
More Importance Sampling — Cosine-Weighted Sampling

Cosine-weighted hemisphere sampling in irradiance estimate:

$$f(\omega) = L_i(\omega)\cos\theta \qquad\qquad p(\omega) = \frac{\cos\theta}{\pi}$$

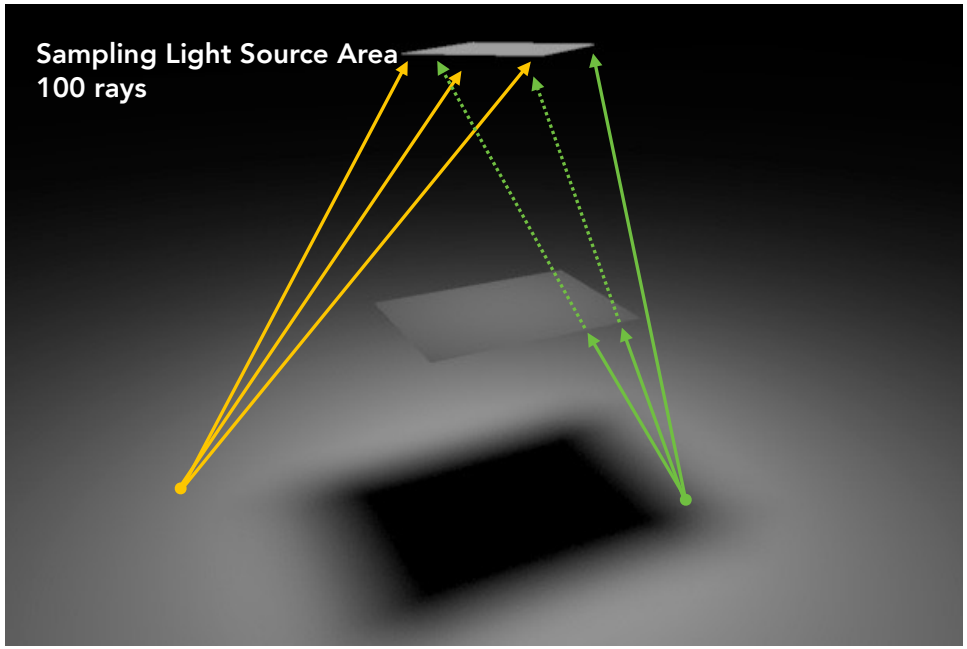
$$\int_\Omega f(\omega)\,\mathrm{d}\omega \approx \frac{1}{N}\sum_i^N \frac{f(\omega)}{p(\omega)} = \frac{1}{N}\sum_i^N \frac{L_i(\omega)\,\cos\theta}{\cos\theta/\pi} = \frac{\pi}{N}\sum_i^N L_i(\omega)$$

Idea: bias samples toward directions where cosθ is large (if L is constant, then these are the directions that contribute most)



So far we've considered light coming directly from light sources, scattered once.

How do we use Monte Carlo integration to get the final color values for each pixel?



Indirect Illumination (Irradiance) Estimate

$$L_r(\mathrm{p}, \omega_r) = \int_{H^2} f_r(\mathrm{p}, \omega_i \rightarrow \omega_r)\, L_i(\mathrm{p}, \omega_i)\, \cos\theta_i \, \mathrm{d}\omega_i$$

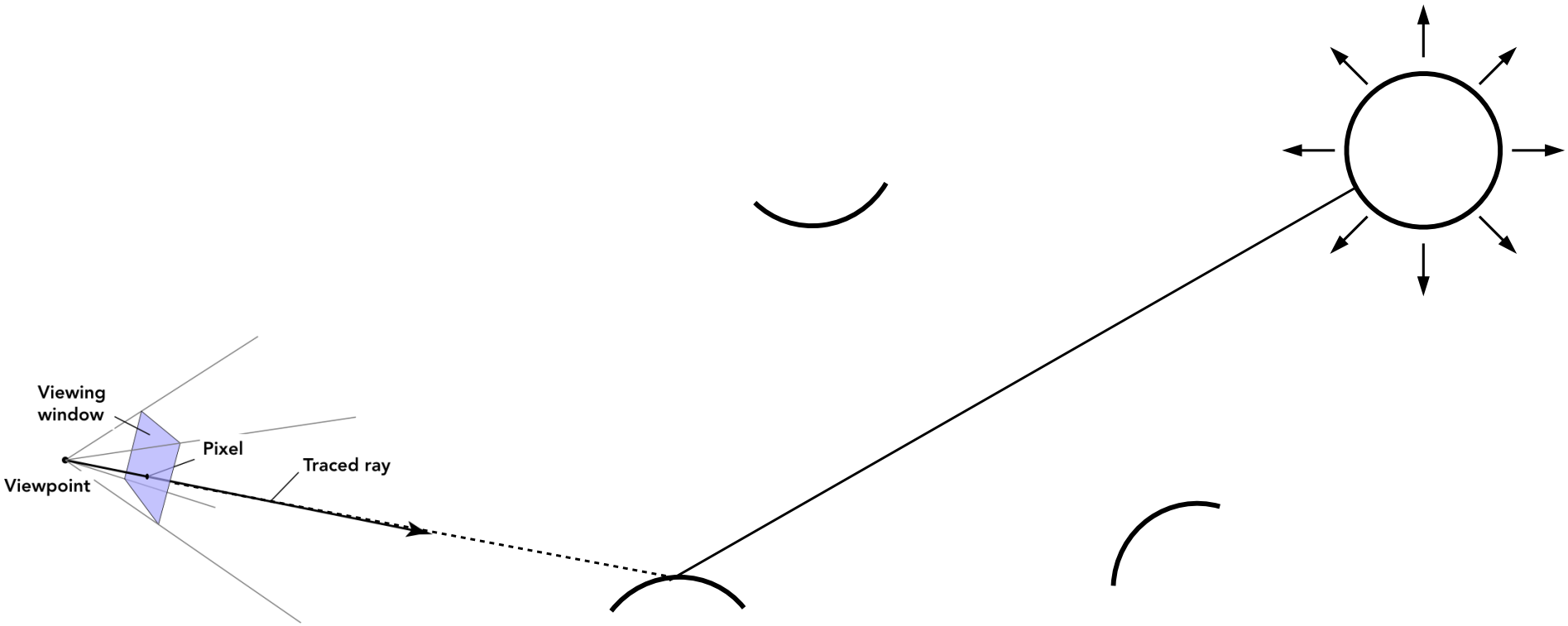
Monte Carlo estimate:

- **Generate directions ω_j sampled from some distribution $p(\omega)$**
- **Choices for $p(\omega)$**
  - **Uniformly sample hemisphere**
  - **Importance sample BRDF (proportional to BRDF)**
  - **Importance sample lights (sample position on lights)**
- **Compute the estimator**

$$\frac{1}{N} \sum_{j=1}^{N} \frac{f_r(\mathrm{p}, \omega_j \rightarrow \omega_r)\, L_i(\mathrm{p}, \omega_j)\, \cos\theta_j}{p(\omega_j)}$$

Light Paths

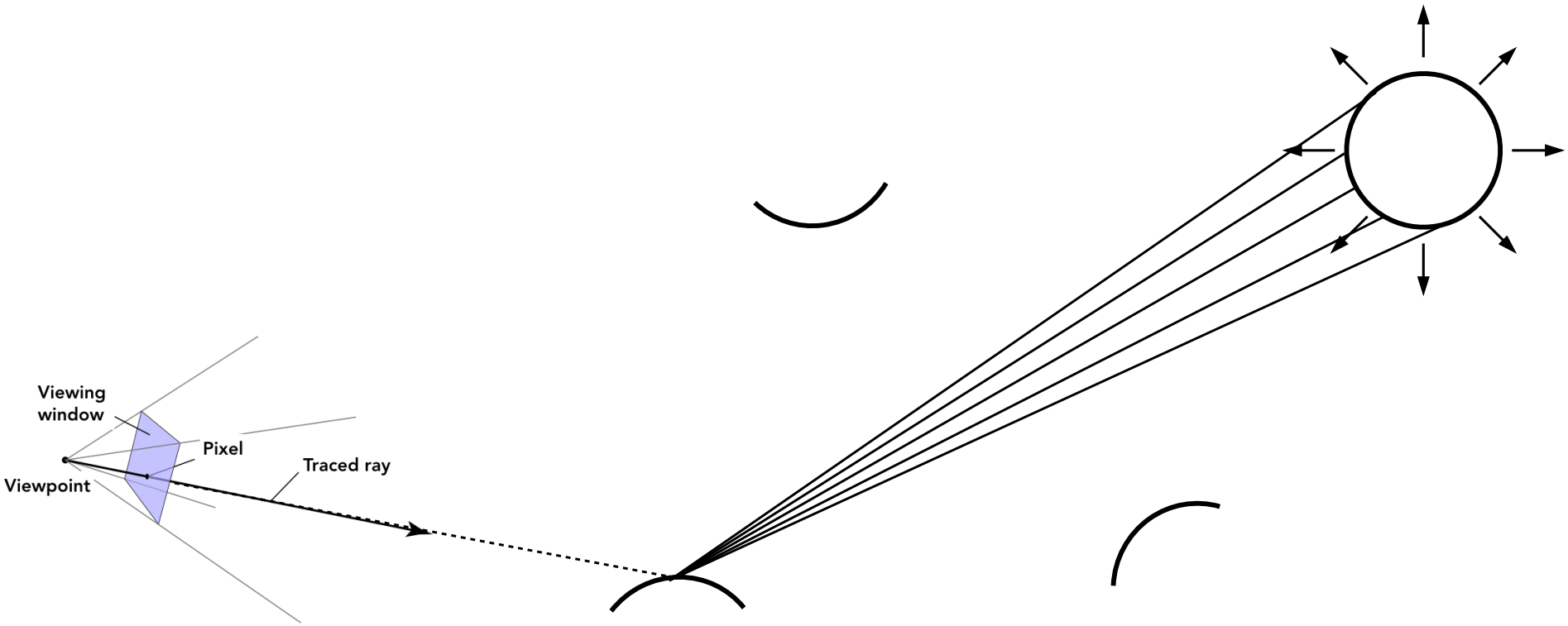
1-Bounce Path Connecting Ray to Light



Camera

Light

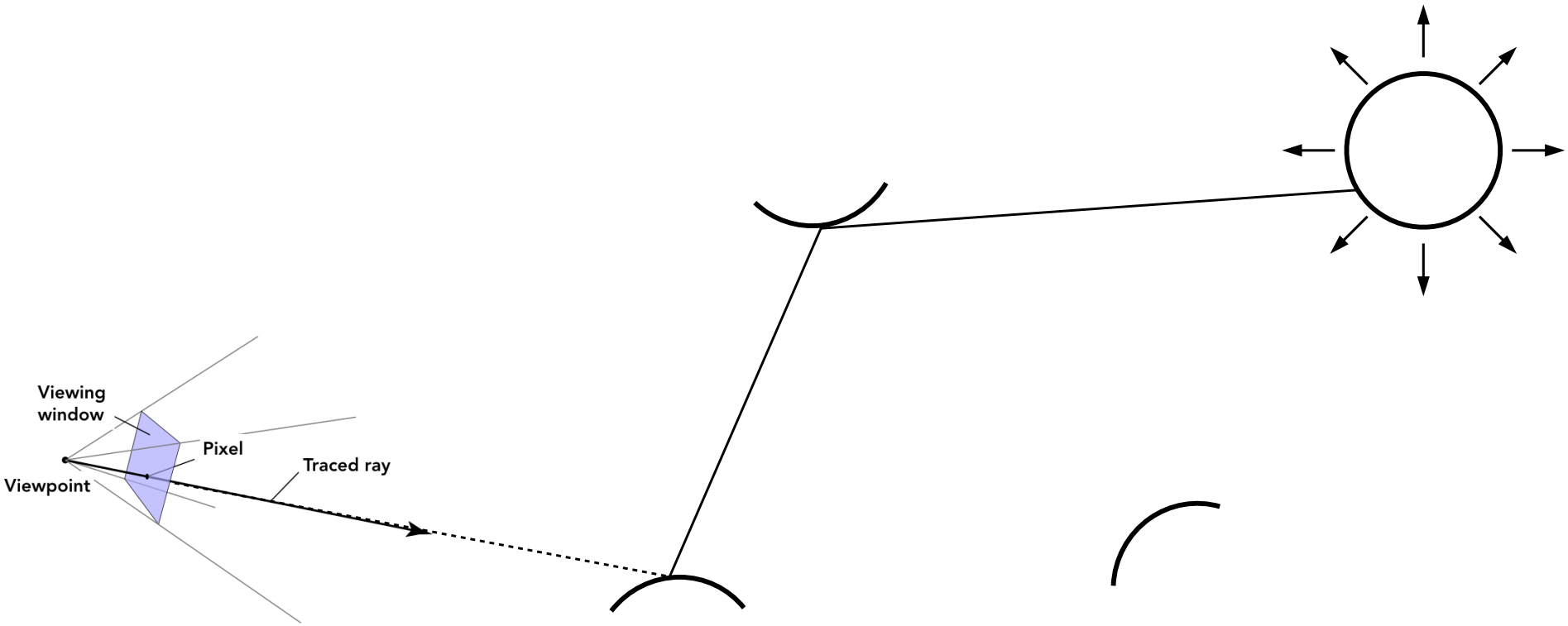
1-Bounce Paths Connecting Ray to Light



Camera

Light

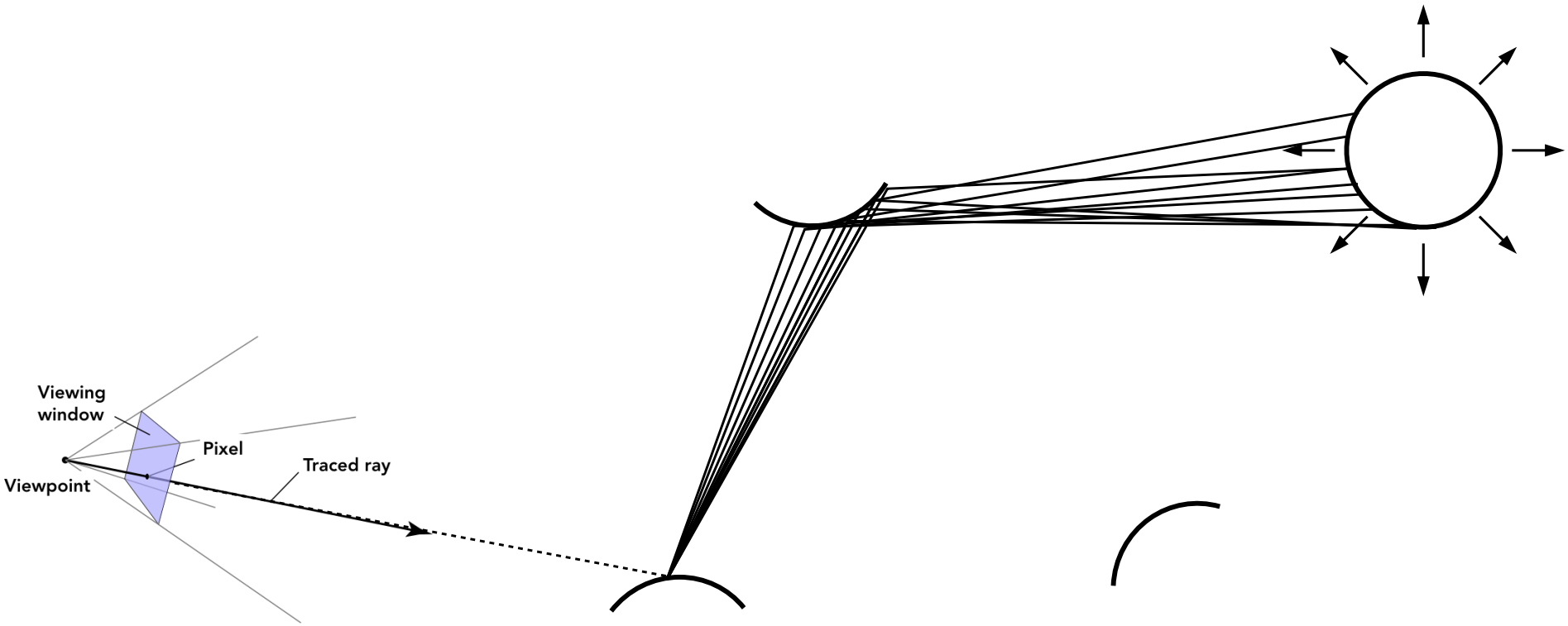
2-Bounce Path Connecting Ray to Light



Camera

Light

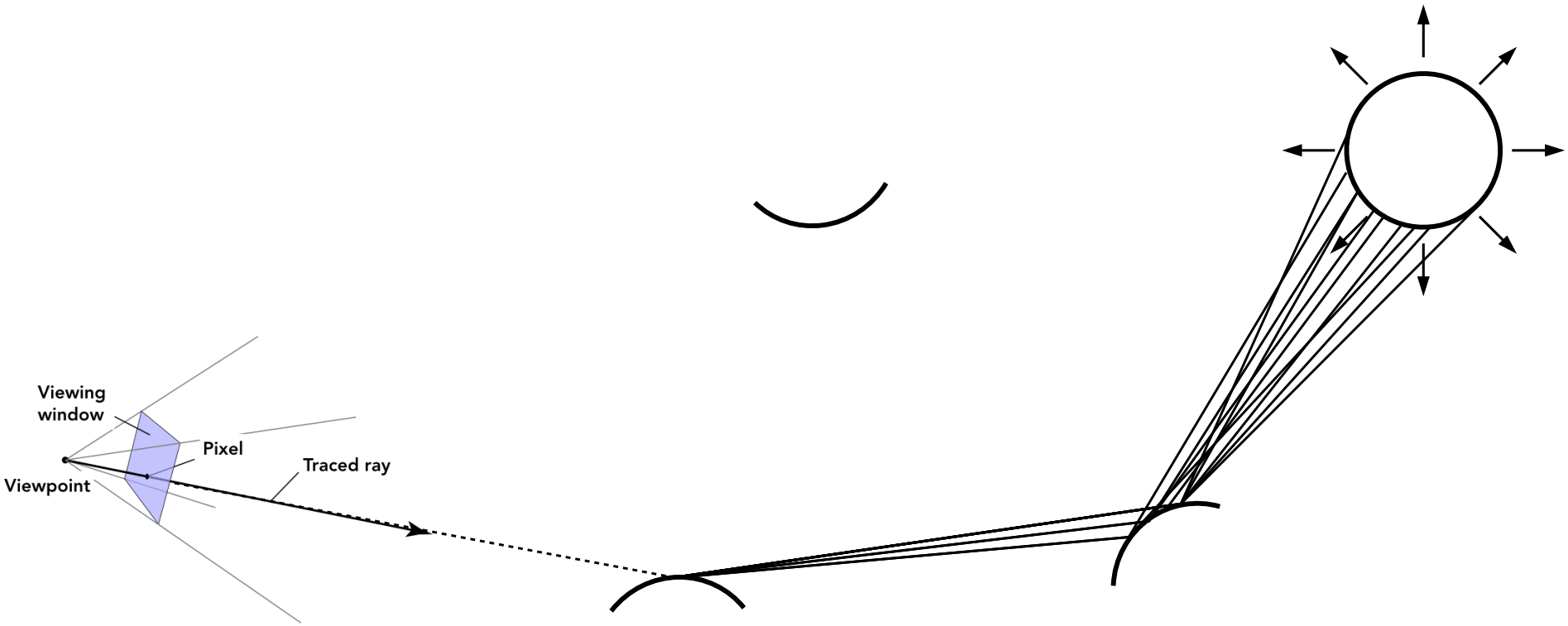
2-Bounce Paths Connecting Ray to Light



Camera

Light

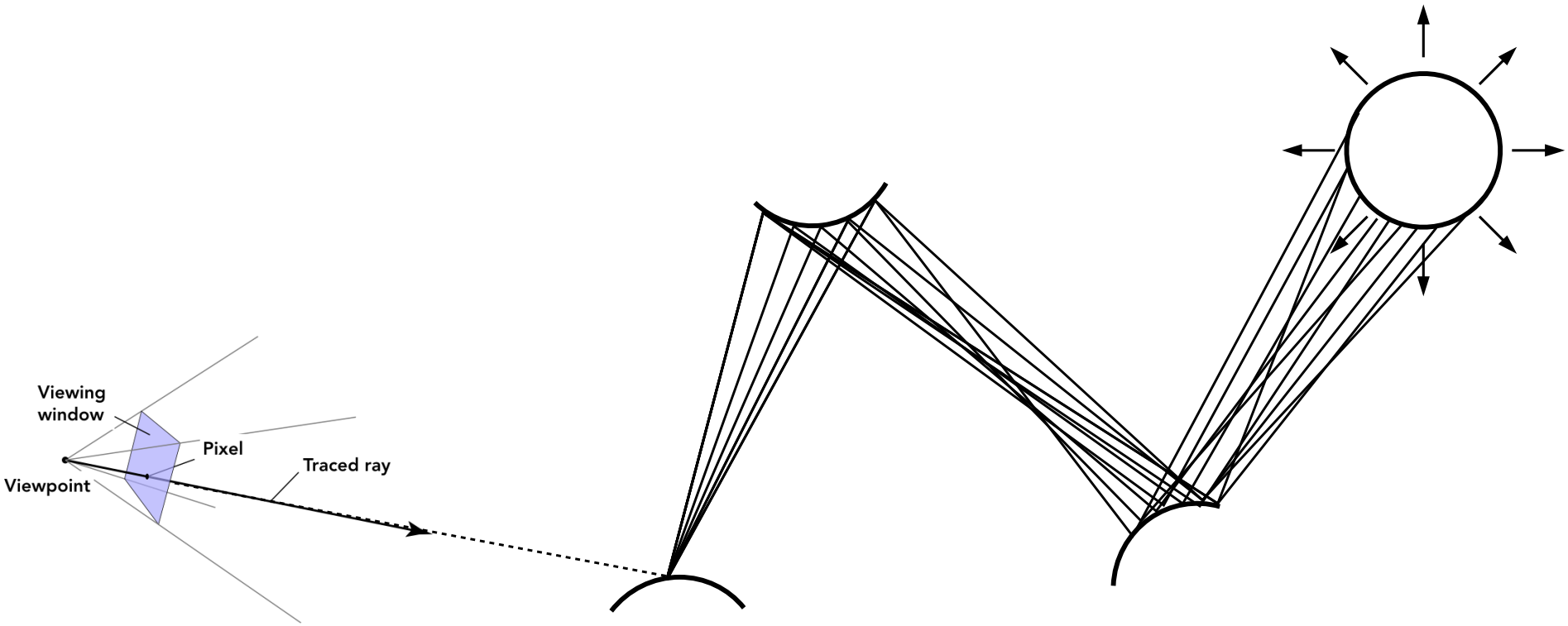
2-Bounce Paths Connecting Ray to Light



Camera

Light

3-Bounce Paths Connecting Ray to Light



Camera

Light

Global Illumination Rendering

Sum over all paths of all lengths

Direct illumination

One-bounce global illumination

Two-bounce global illumination

Four-bounce global illumination

Eight-bounce global illumination

Sixteen-bounce global illumination

Global Illumination Rendering

Sum over all paths of all lengths

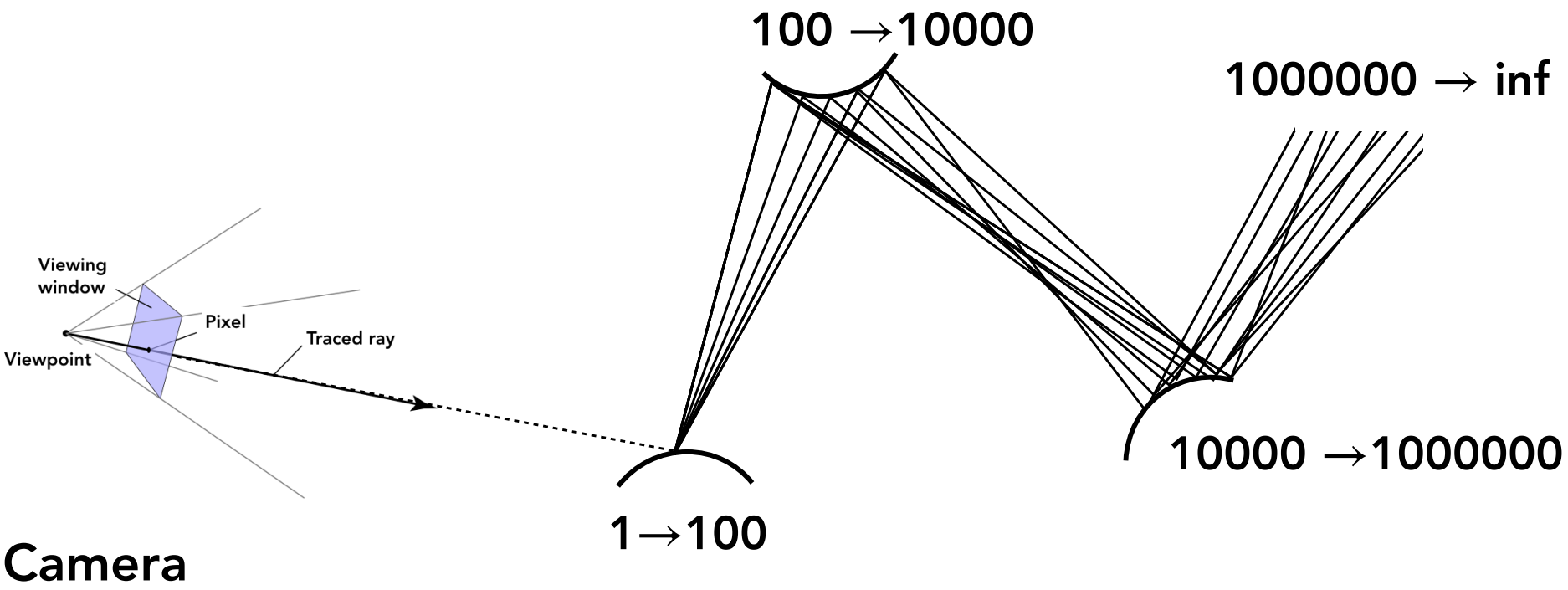
Challenges:

- How to generate all possible paths?
- How to sample space of paths efficiently?

Path Tracing Problem

Problem 1: Explosion of #rays as #bounces go up:

$$\#rays = N^{\#bounces}$$

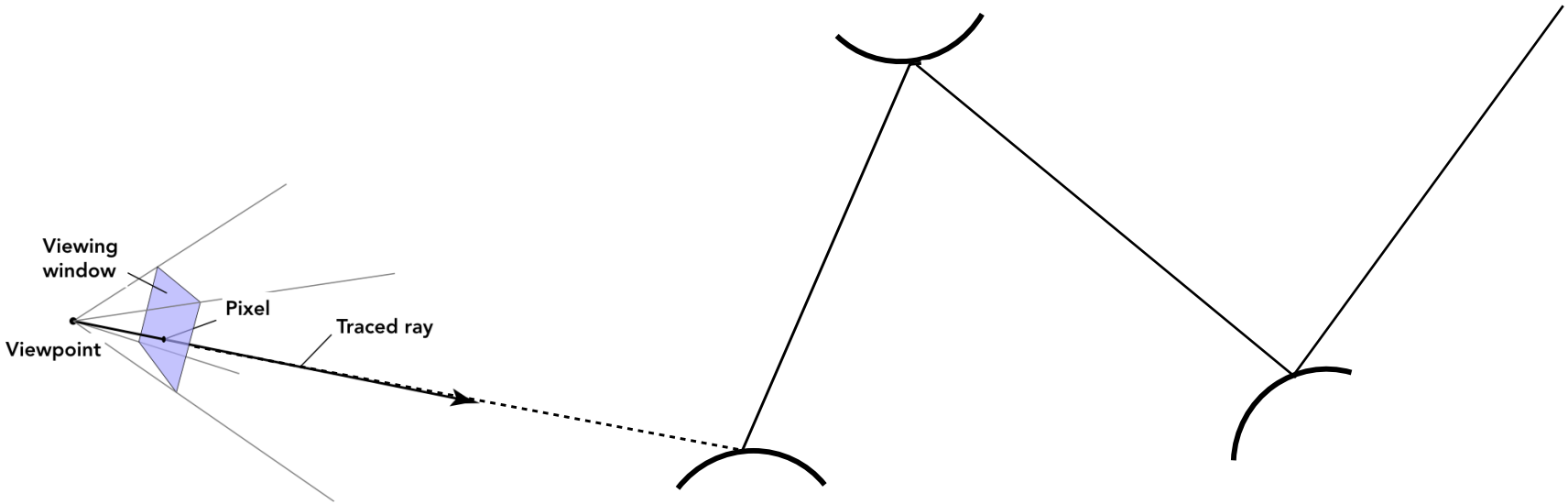


Key observation: #rays will not explode iff N = ?

Path Tracing Problem

Solution: only 1 ray is traced at each shading point

$$\#rays = N^{\#bounces}=1$$

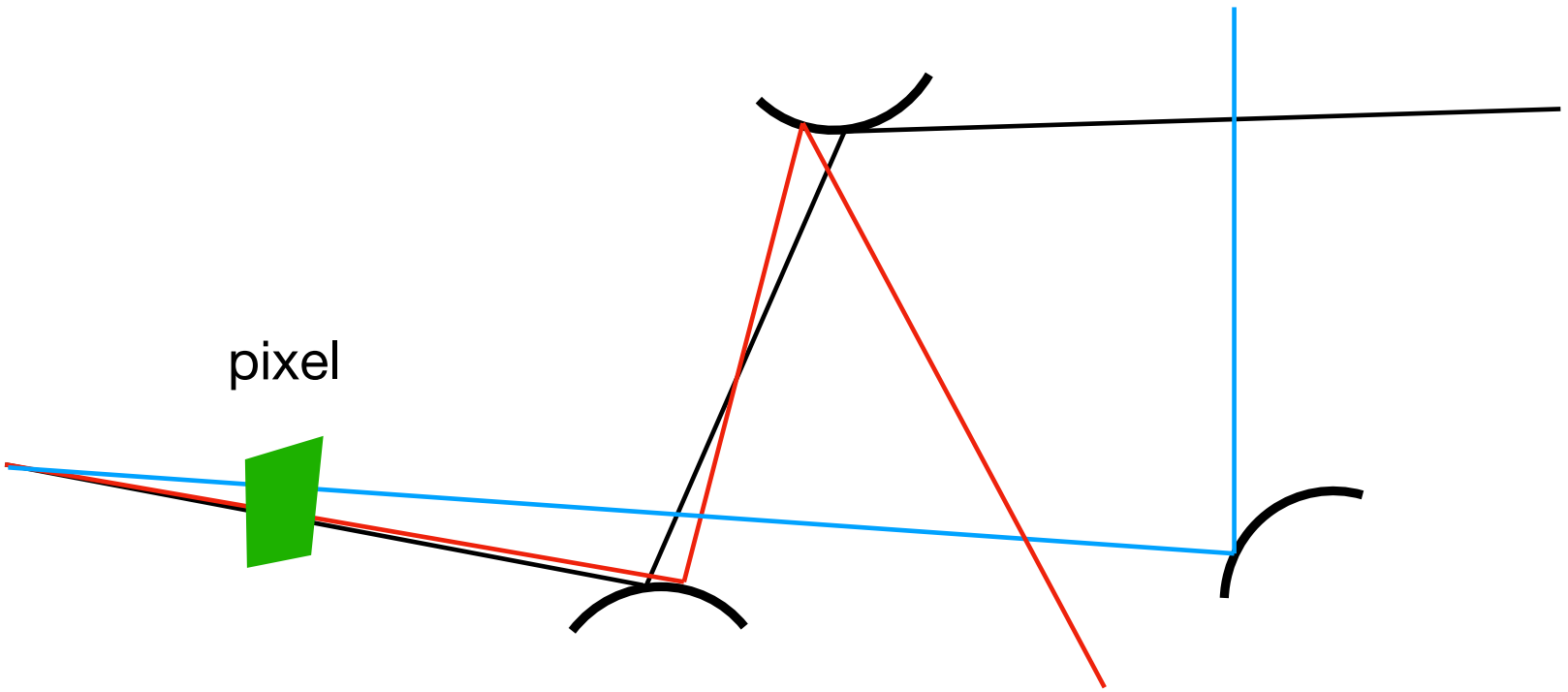



One sample (path) per pixel

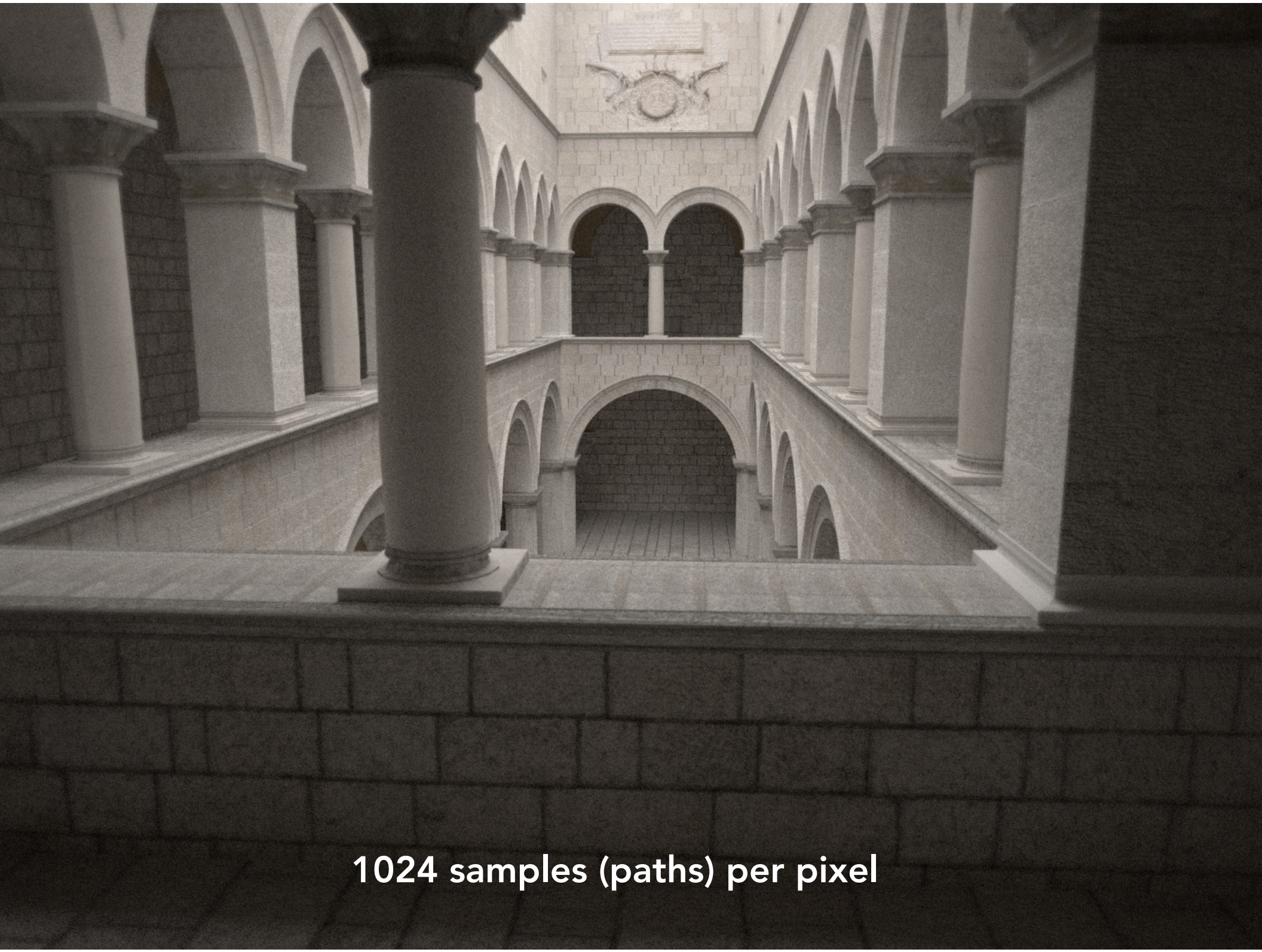
Path Tracing Problem

But, noisy rendering if only one ray is traced

No problem, just trace more paths through each pixel and average their radiance!




32 samples (paths) per pixel


1024 samples (paths) per pixel

Path Tracing Problem

Problem 2: Infinite Bounces of Light

#bounces $\rightarrow \infty$

How to integrate over infinite dimensions?

- Note: if energy dissipates, contributions from higher bounces decrease exponentially

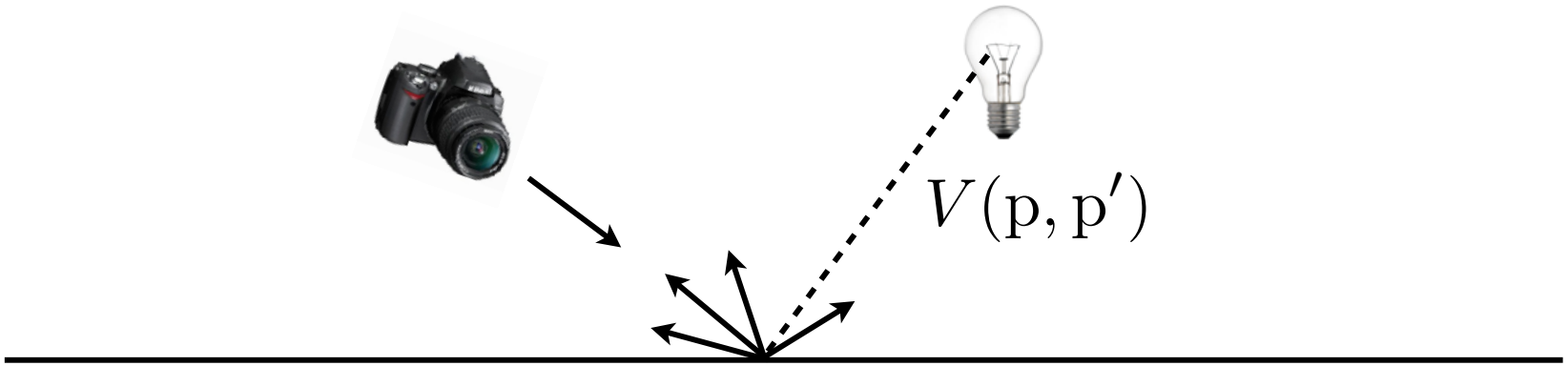
Idea: just use N bounces

- Problem: biased! No matter how many Monte Carlo samples, never see light taking N+1 to infinity bounces

Russian Roulette

- Idea: want to avoid spending time evaluating function for samples that make a small contribution to the final result
- Consider a low-contribution sample of the form:

$$L = \frac{f_r(\omega_i \rightarrow \omega_o)\, L_i(\omega_i)\, V(\mathrm{p}, \mathrm{p}')\, \cos\theta_i}{p(\omega_i)}$$



Russian Roulette

$$L = \frac{f_r(\omega_i \rightarrow \omega_o)\, L_i(\omega_i)\, V(\mathrm{p}, \mathrm{p}')\, \cos\theta_i}{p(\omega_i)}$$

$$\downarrow$$

$$L = \left[\frac{f_r(\omega_i \rightarrow \omega_o)\, L_i(\omega_i)\, \cos\theta_i}{p(\omega_i)}\right] V(\mathrm{p}, \mathrm{p}')$$

- If tentative contribution (in brackets) is small, total contribution to the image will be small regardless of $V(p, p')$
- Ignoring low-contribution samples introduces systematic error
  - No longer converges to correct value! (Exact integral)
- Instead, randomly discard low-contribution samples in a way that leaves estimator unbiased

Russian Roulette

- **New estimator: evaluate original estimator with probability p_{rr} , reweight. Otherwise ignore.**

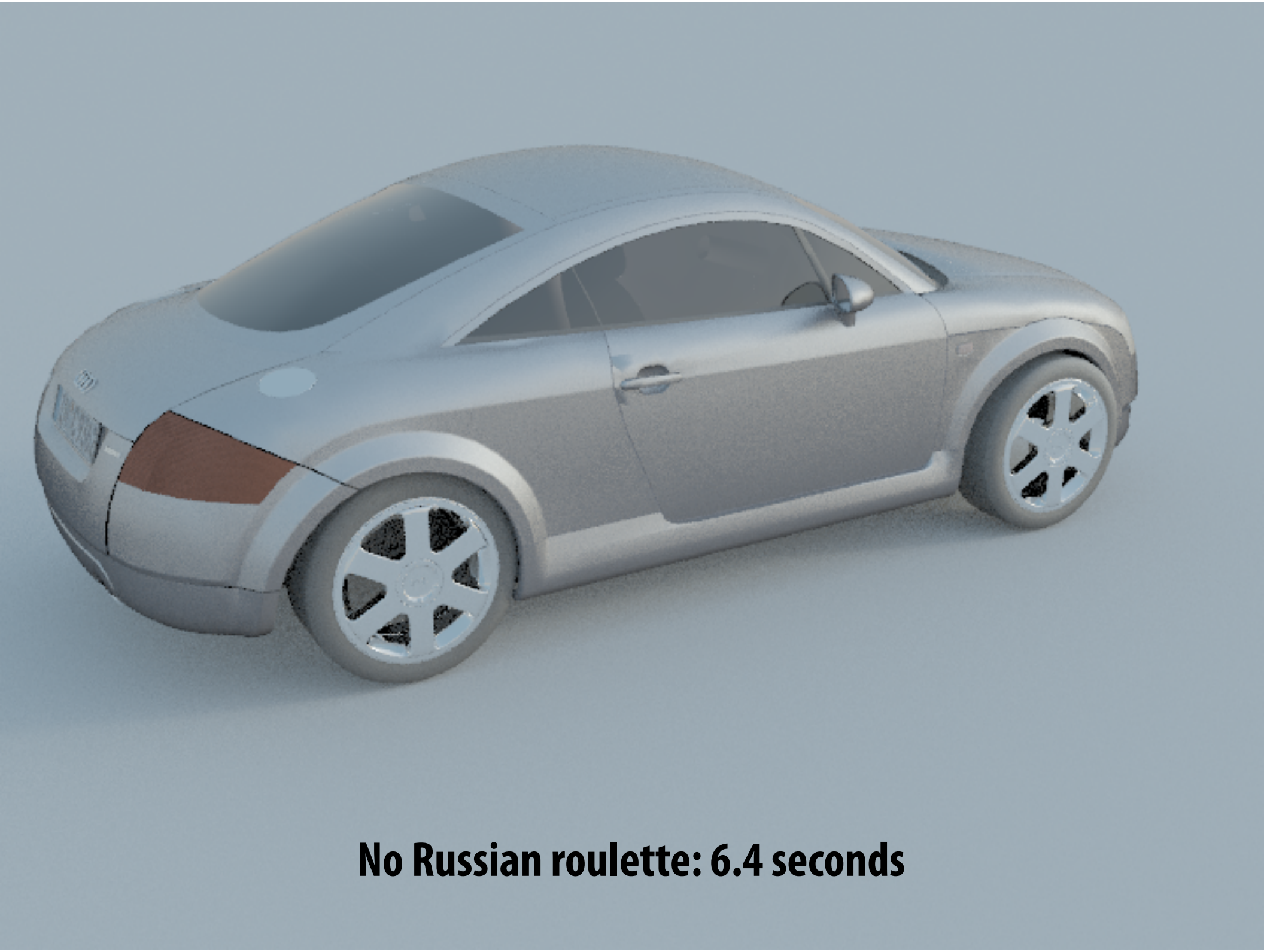
$$X' = \begin{cases} \dfrac{X}{p_{rr}}, & \text{with probability } p_{rr}, \\ 0, & \text{with probability } 1 - p_{rr}. \end{cases}$$

- **Sample expected value as original estimator:**

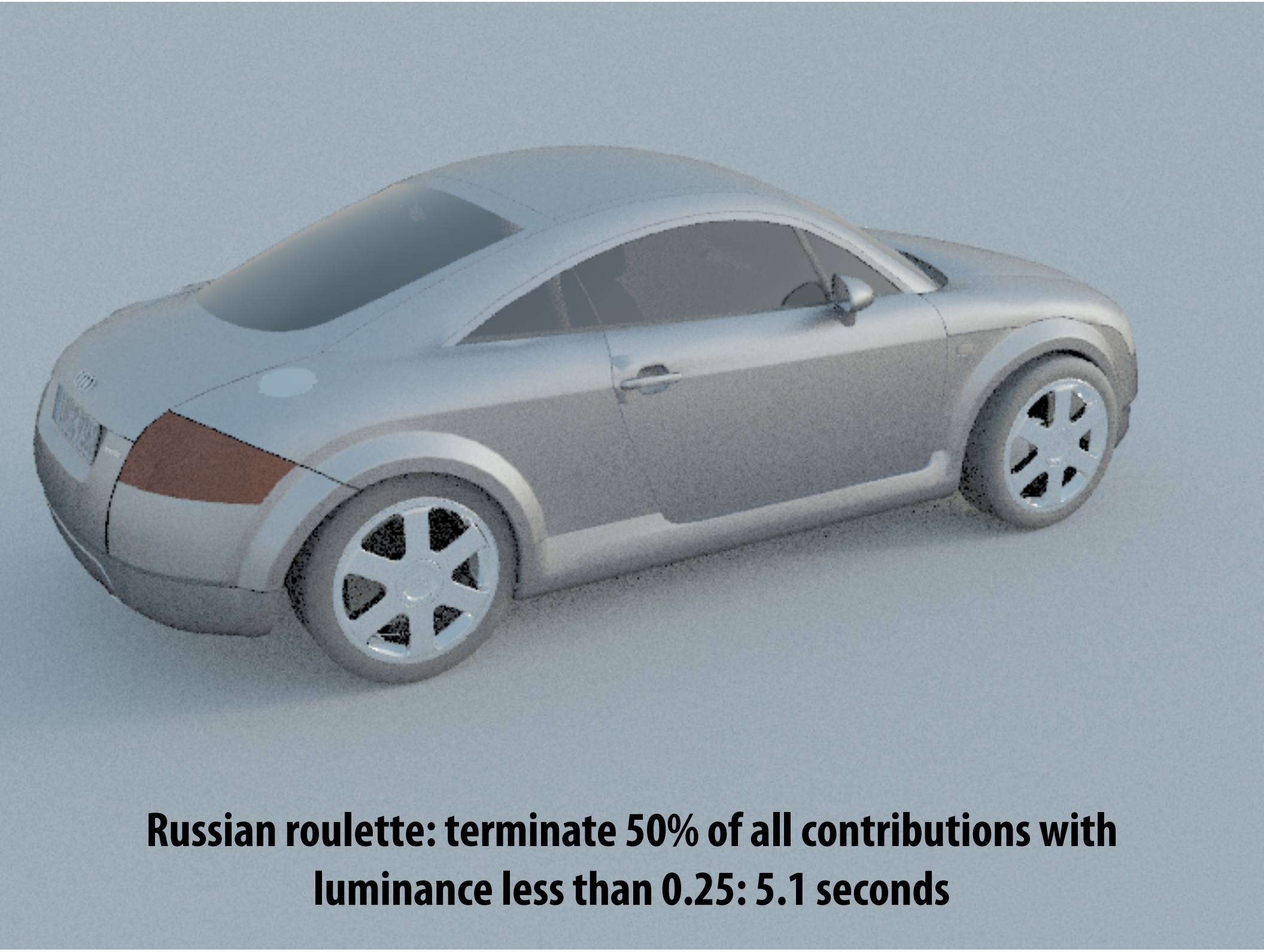
$$E[X'] = p_{rr} \cdot \frac{E[X]}{p_{rr}} + (1 - p_{rr}) \cdot 0 = E[X]$$

Unbiased!

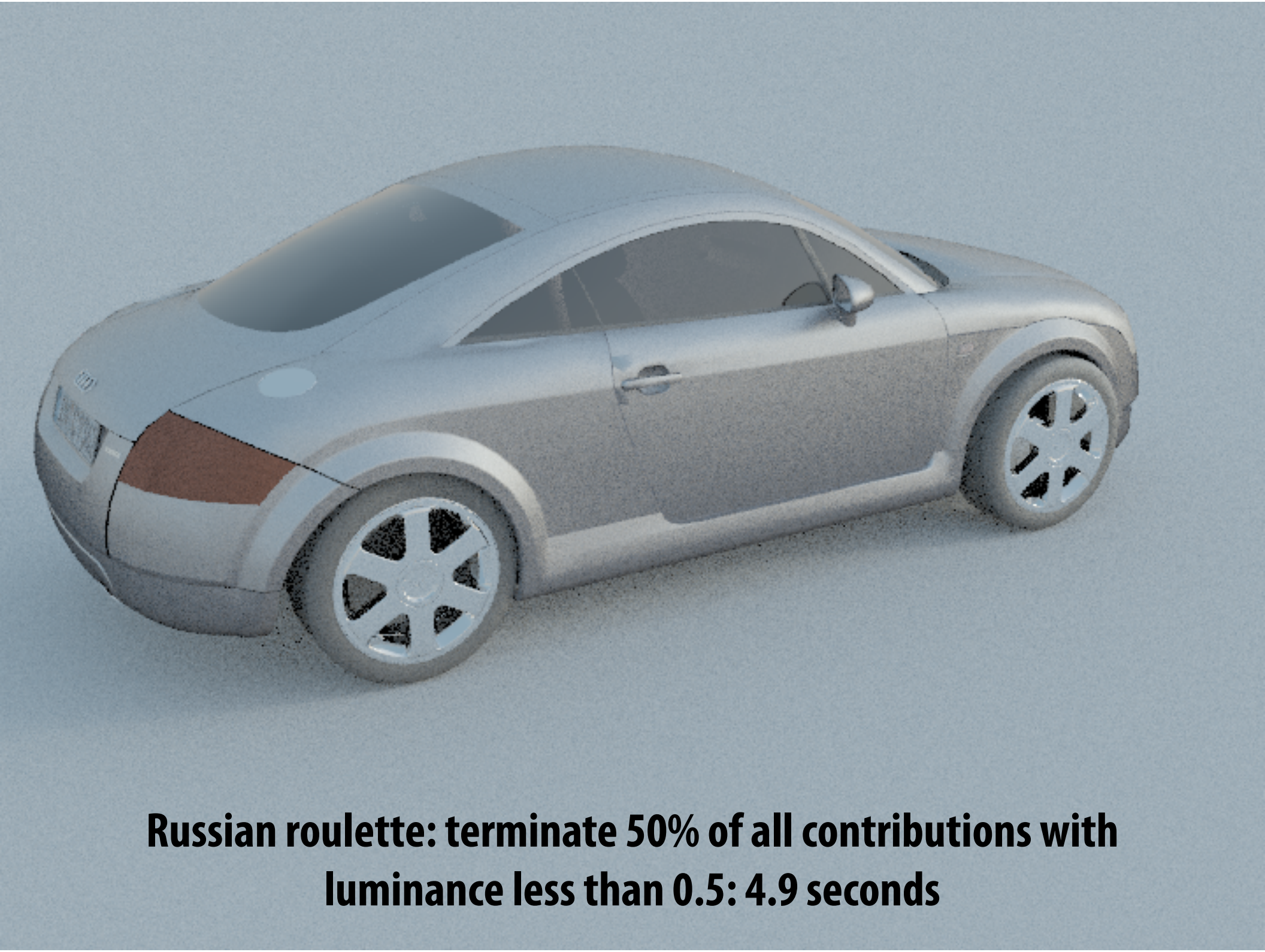
As $p_{rr} \downarrow$, the noise (variance) increases.



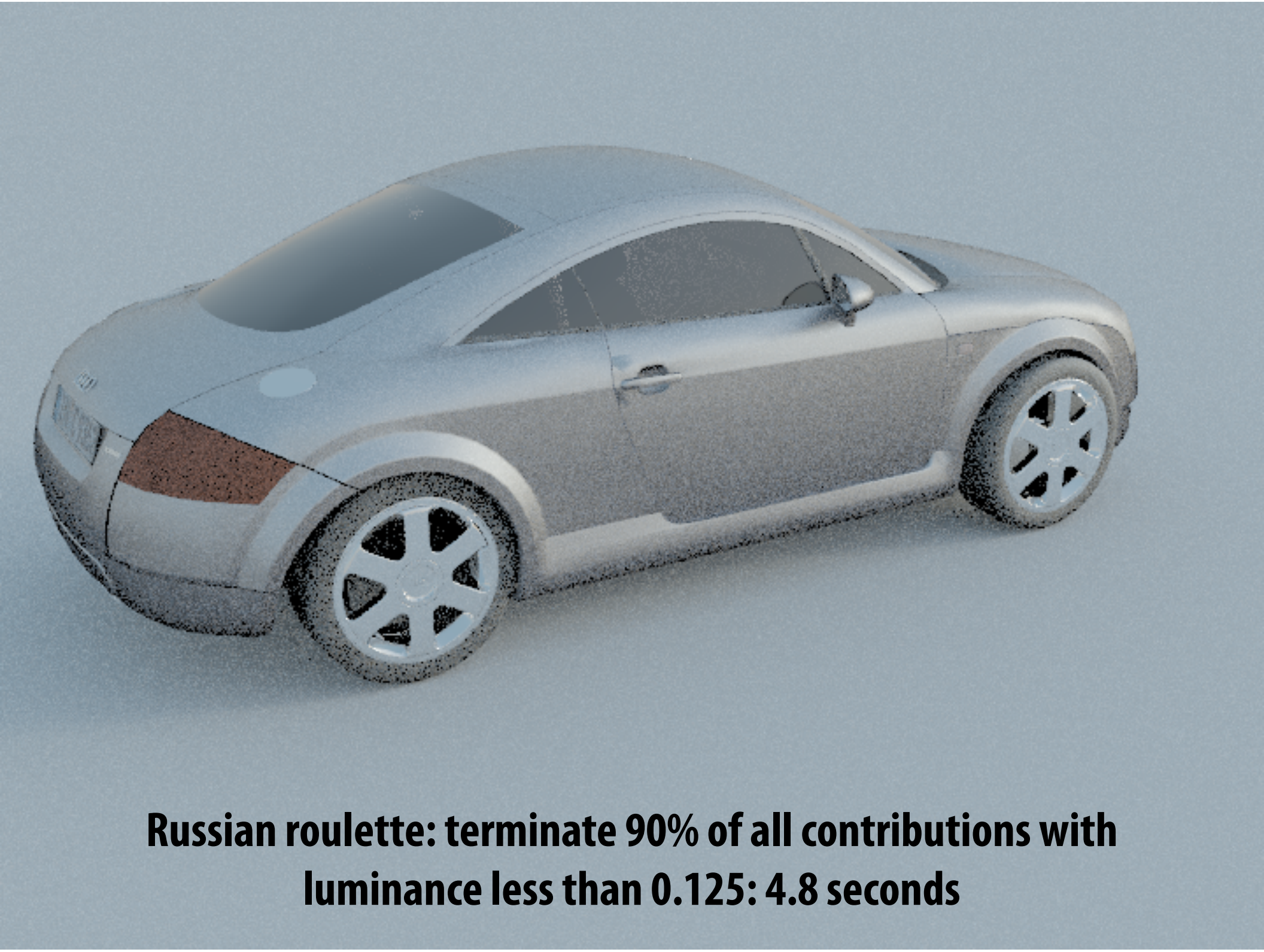
No Russian roulette: 6.4 seconds



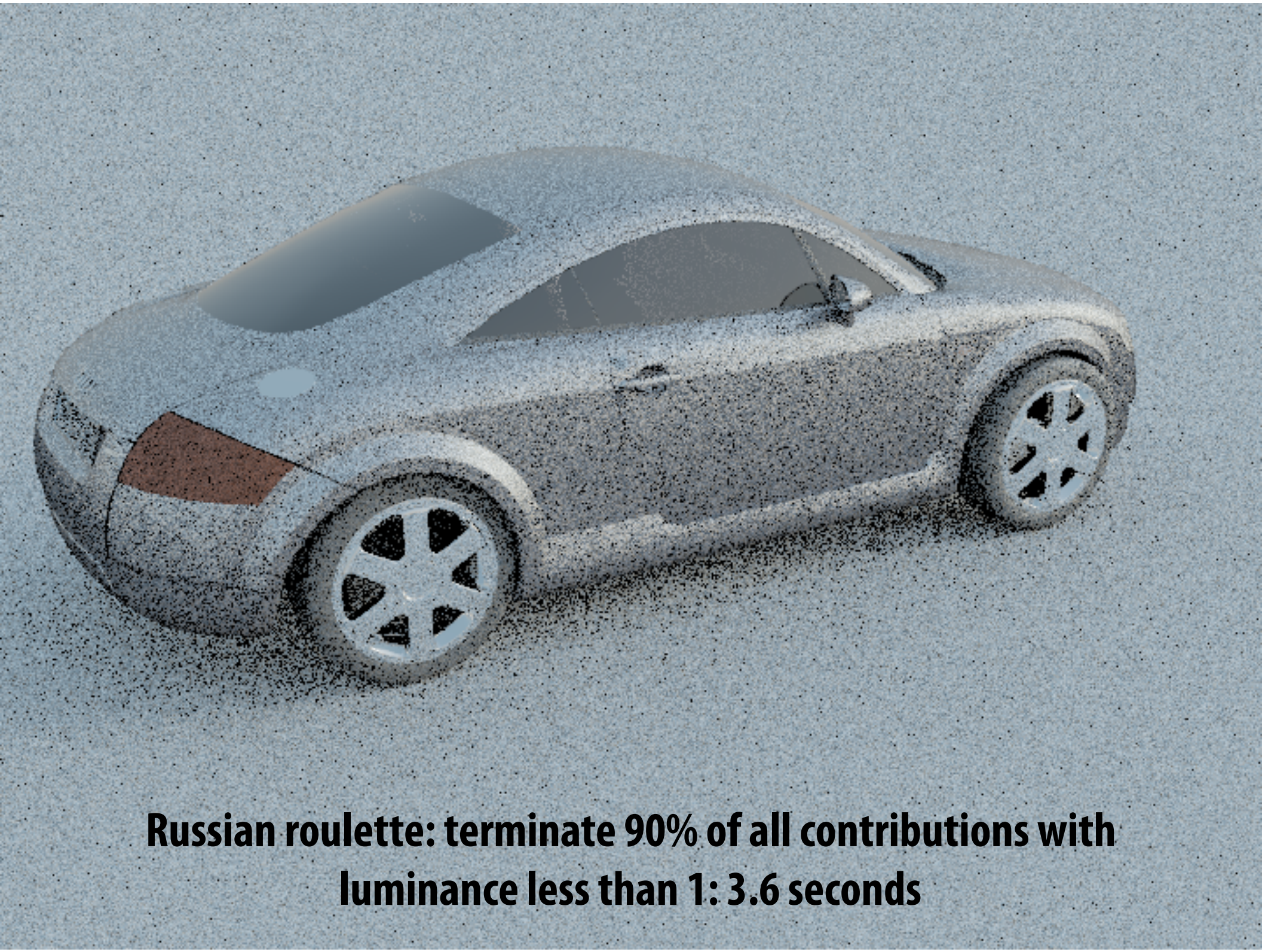
Russian roulette: terminate 50% of all contributions with luminance less than 0.25: 5.1 seconds



Russian roulette: terminate 50% of all contributions with luminance less than 0.5: 4.9 seconds



Russian roulette: terminate 90% of all contributions with luminance less than 0.125: 4.8 seconds


Russian roulette: terminate 90% of all contributions with
luminance less than 1: 3.6 seconds

Path Tracing Pseudo Code

Monte Carlo Estimator

$$F_N = \frac{A'}{N} \sum_{i=1}^{N} Y_i$$

$$Y_i = L_o(\mathrm{p}'_i, \omega'_i)\, V(\mathrm{p}, \mathrm{p}'_i)\, \frac{\cos\theta_i \cos\theta'_i}{|\mathrm{p} - \mathrm{p}'_i|^2}$$

```
shade(p, wo)
    # Contribution from the light source.
    Uniformly sample the light at x' (pdf_light = 1 / A)
    L_dir = L_i * f_r * cos θ * cos θ' / |x' - p|^2 / pdf_light

    # Contribution from other reflectors.
    L_indir = 0.0
    Test Russian Roulette with probability P_RR
    Uniformly sample the hemisphere toward wi (pdf_hemi = 1 / 2pi)
    Trace a ray r(p, wi)
    If ray r hit a non-emitting object at q
        L_indir = shade(q, -wi) * f_r * cos θ / pdf_hemi / P_RR

    Return L_dir + L_indir
```

$$\frac{1}{N} \sum_{j=1}^{N} \frac{f_r(\mathrm{p}, \omega_j \to \omega_r)\, L_i(\mathrm{p}, \omega_j)\, \cos\theta_j}{p(\omega_j)}$$

计算一条光路，光路上的每个点包括直接照明和间接照明两部分，光路随机终止

Path Tracing Overview

Terminate paths randomly with Russian Roulette

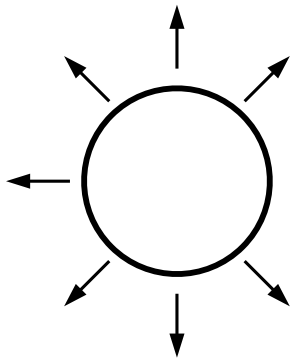
Partition the recursive radiance evaluation. At each point on light path

- Direct lighting – non-recursive, importance sample lights
- Indirect lighting – recursive, importance sample BRDF

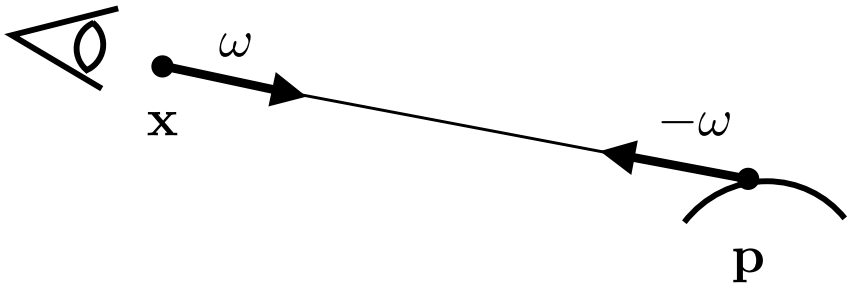
Monte Carlo estimate for each partition separately

- Possible to take just one sample for each
- Assume: 100s - 1000s of paths sampled per pixel

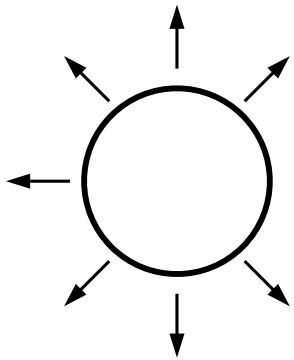
Partitioning the Rendering Equation



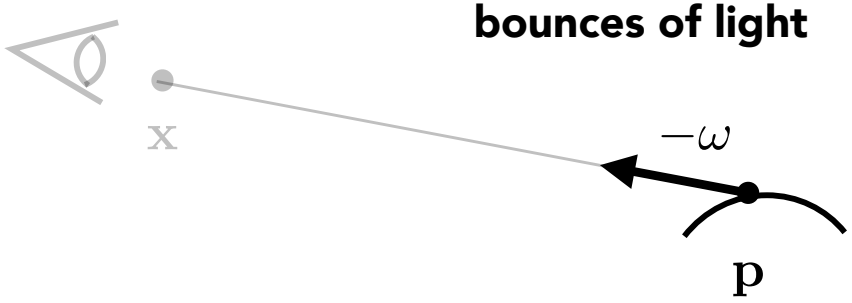
```
EstRadianceIn(x, ω)
      = EstRadianceOut(p, -ω)
```



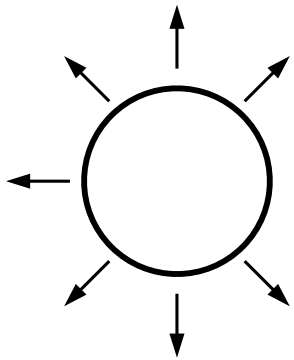
Partitioning the Rendering Equation

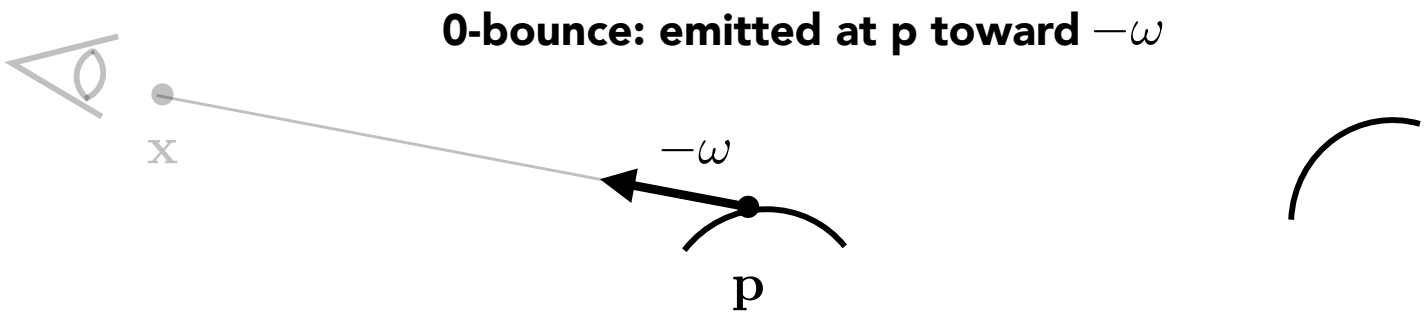


Need to sum paths going through p representing 0, 1, 2, 3, … bounces of light



Partitioning the Rendering Equation

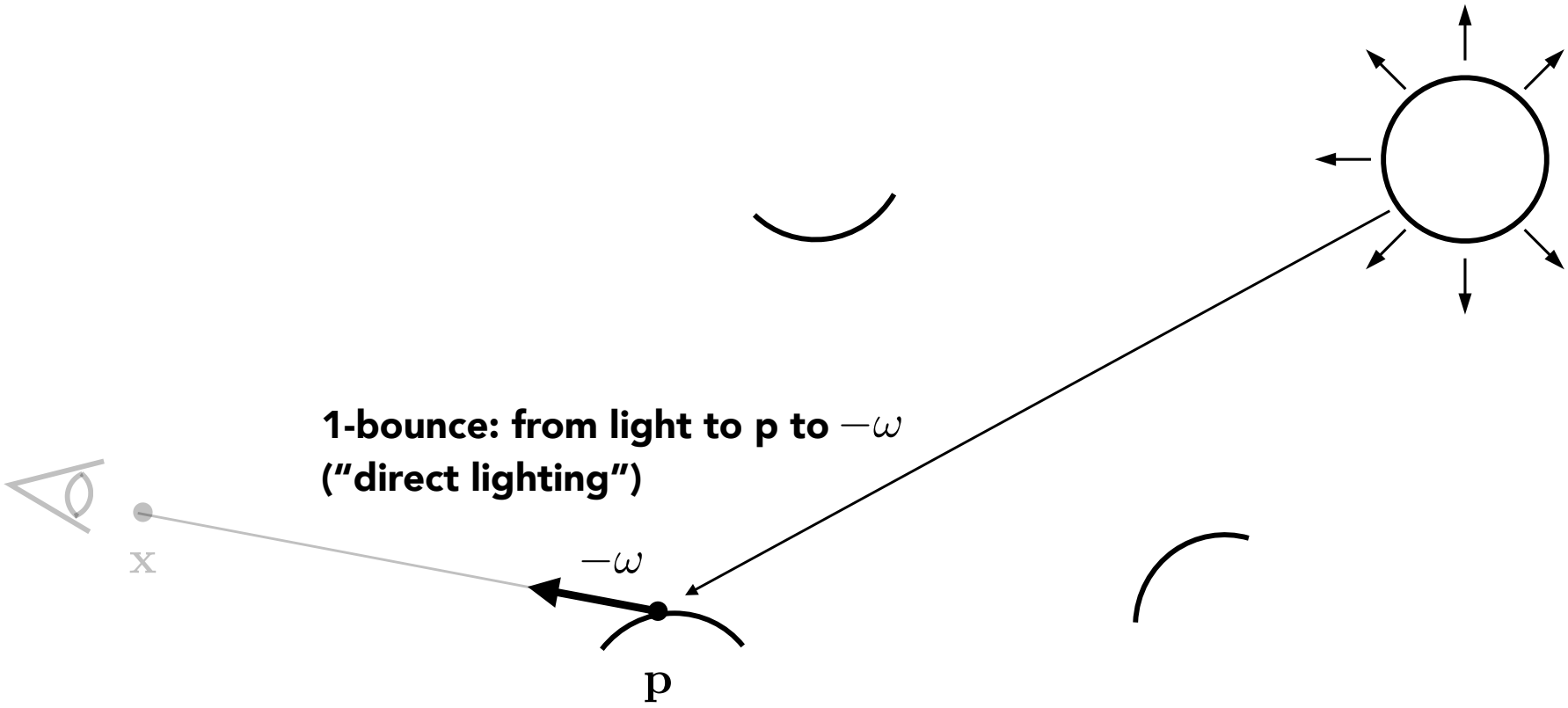




At p, consider light contributions from paths of varying bounce-length

- **0-bounce: light emitted from p (p is on a light source)**

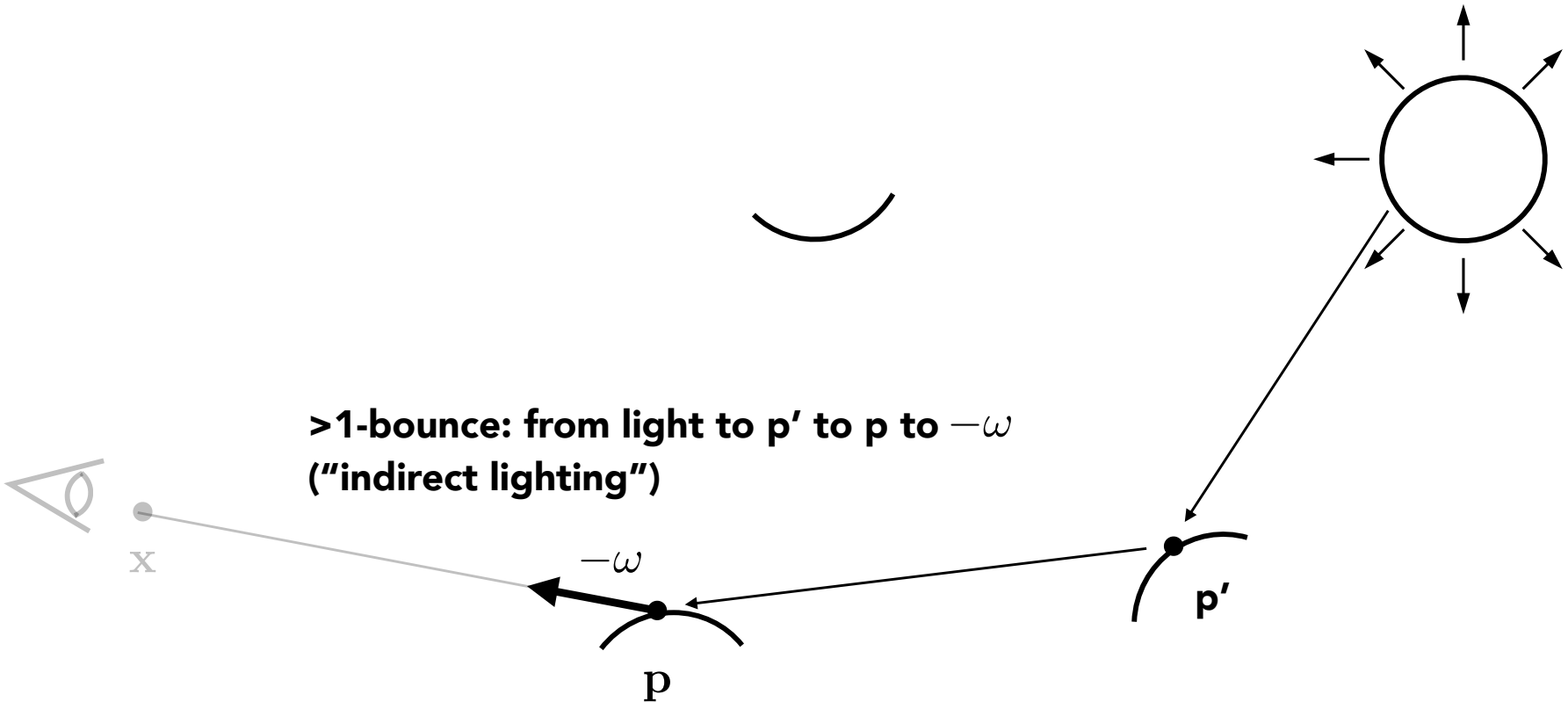
Partitioning the Rendering Equation



At p, consider light contributions from paths of varying bounce-length

- **0-bounce: light emitted from p (p is on a light source)**
- **1-bounce: from light to p to x ("direct illumination")**

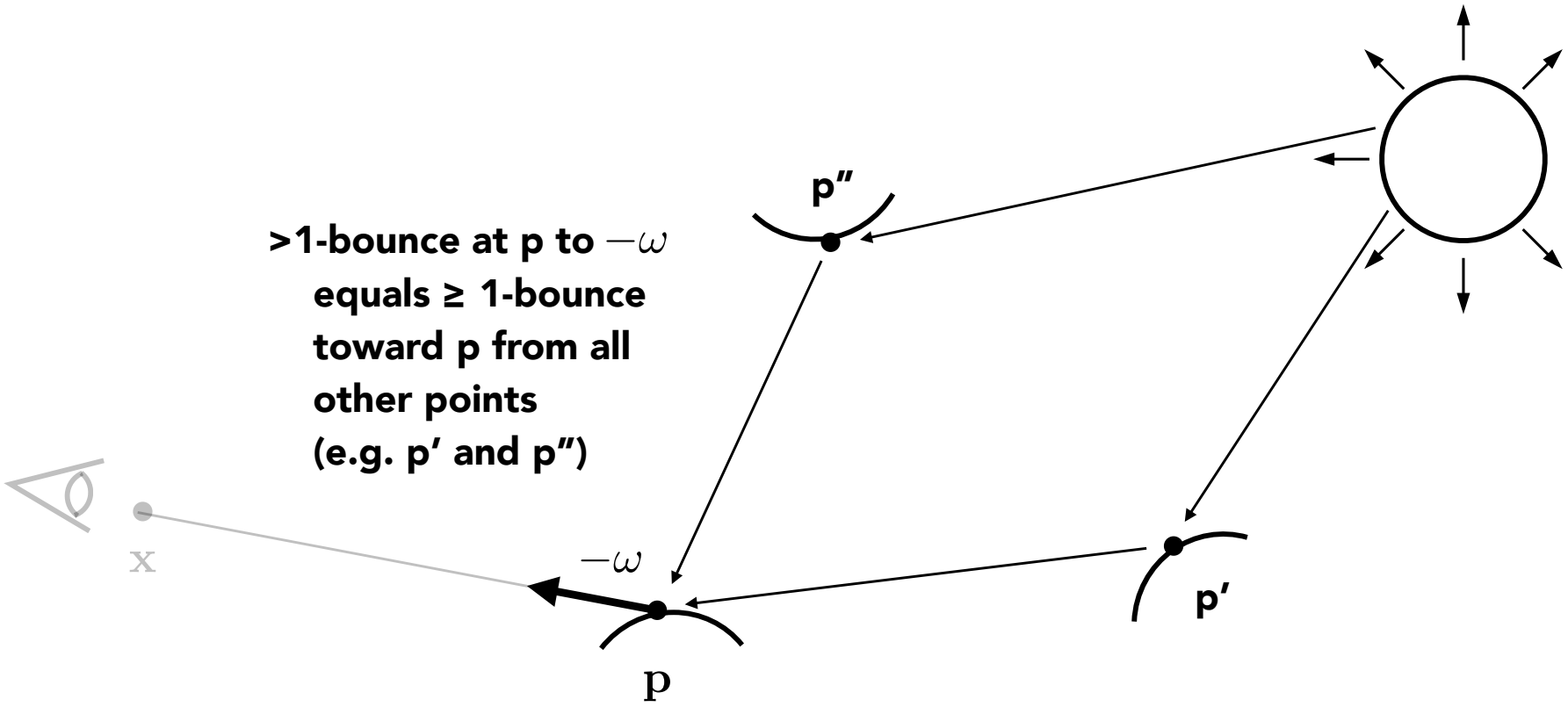
Partitioning the Rendering Equation



At p, consider light contributions from paths of varying bounce-length

- **0-bounce: light emitted from p (p is on a light source)**
- **1-bounce: from light to p to x ("direct illumination")**
- **>1-bounce: from light to at least one other point to p to x ("indirect illumination")**

Consider Evaluation of >1 Bounce of Light



At p, consider light contributions from paths of varying bounce-length

- 0-bounce: light emitted from p (p is on a light source)
- 1-bounce: from light to p to x (“direct illumination”)
- >1-bounce: from light to at least one other point to p to x (“indirect illumination”)

Path Tracing Pseudocode

```
EstRadianceIn(x, ω)                  // incoming at x from dir ω
   p = intersectScene(x, ω);
   return ZeroBounceRadiance(p, -ω)
          + AtLeastOneBounceRadiance(p, -ω);

ZeroBounceRadiance(p, ωo)            // outgoing at p in dir ωo
   return p.emittedLight(ωo);
```

Path Tracing Pseudocode

```
AtLeastOneBounceRadiance(p, ωo)                    // out at p, dir ωo
   L = OneBounceRadiance(p, ωo);                   // direct illum

   ωi, pdf = p.brdf.sampleDirection(ωo);            // Imp. sampling
   p’ =  intersectScene(p, ωi);
   cpdf = continuationProbability(p.brdf, ωi, ωo);
   if (random01() < cpdf)                          // Russ. Roulette
      L += AtLeastOneBounceRadiance(p’, -ωi)  // Recursive est. of
      * p.brdf(ωi, ωo) * costheta / pdf / cpdf;// indirect illum
   return L;

OneBounceRadiance(p, ωo)                           // out at p, dir ωo
    return DirectLightingSampleLights(p, ωo); // direct illum
```

Direct Lighting Pseudocode (Lights)

```
DirectLightingSamplingLights(p, wo)
   L, wi, pdf = lights.sampleDirection(p);    // Imp. sampling

   if (scene.shadowIntersection(p, wi))       // Shadow ray
      return 0;
   else
      return L * p.brdf(wi, wo) * costheta / pdf;

// Note: only one random sample over all lights.
// Assignment 3-1 asks you to, alternatively, loop over
// multiple lights and take multiple samples (later slide)
```

Is Monte Carlo Ray Tracing Correct?

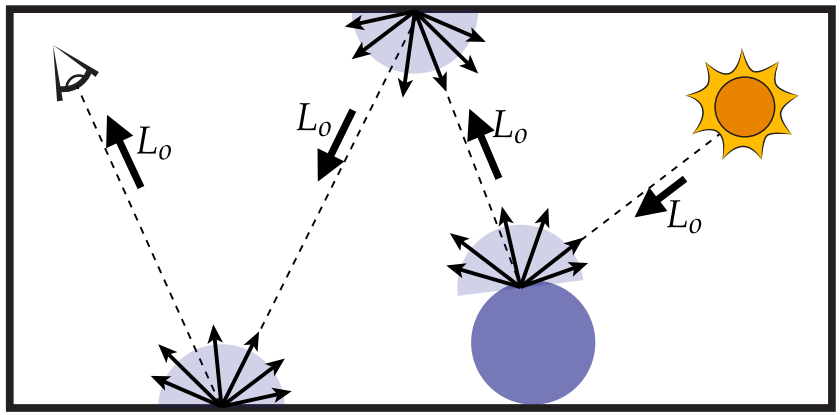
Yes, almost 100% correct, a.k.a. PHOTO-REALISTIC

Photo

Global Illumination
(Path Tracing)

Monte Carlo Ray Tracing — Summary

- Light hitting a point (e.g., pixel) described by rendering equation
  - Expressed as recursive integral
  - Can use Monte Carlo to estimate this integral
  - Need to be intelligent about how to sample!



$$L_o(\mathbf{p}, \omega_o) = L_e(\mathbf{p}, \omega_o) + \int_{\mathcal{H}^2} f_r(\mathbf{p}, \omega_i \rightarrow \omega_o) L_i(\mathbf{p}, \omega_i) \cos\theta \, d\omega_i$$