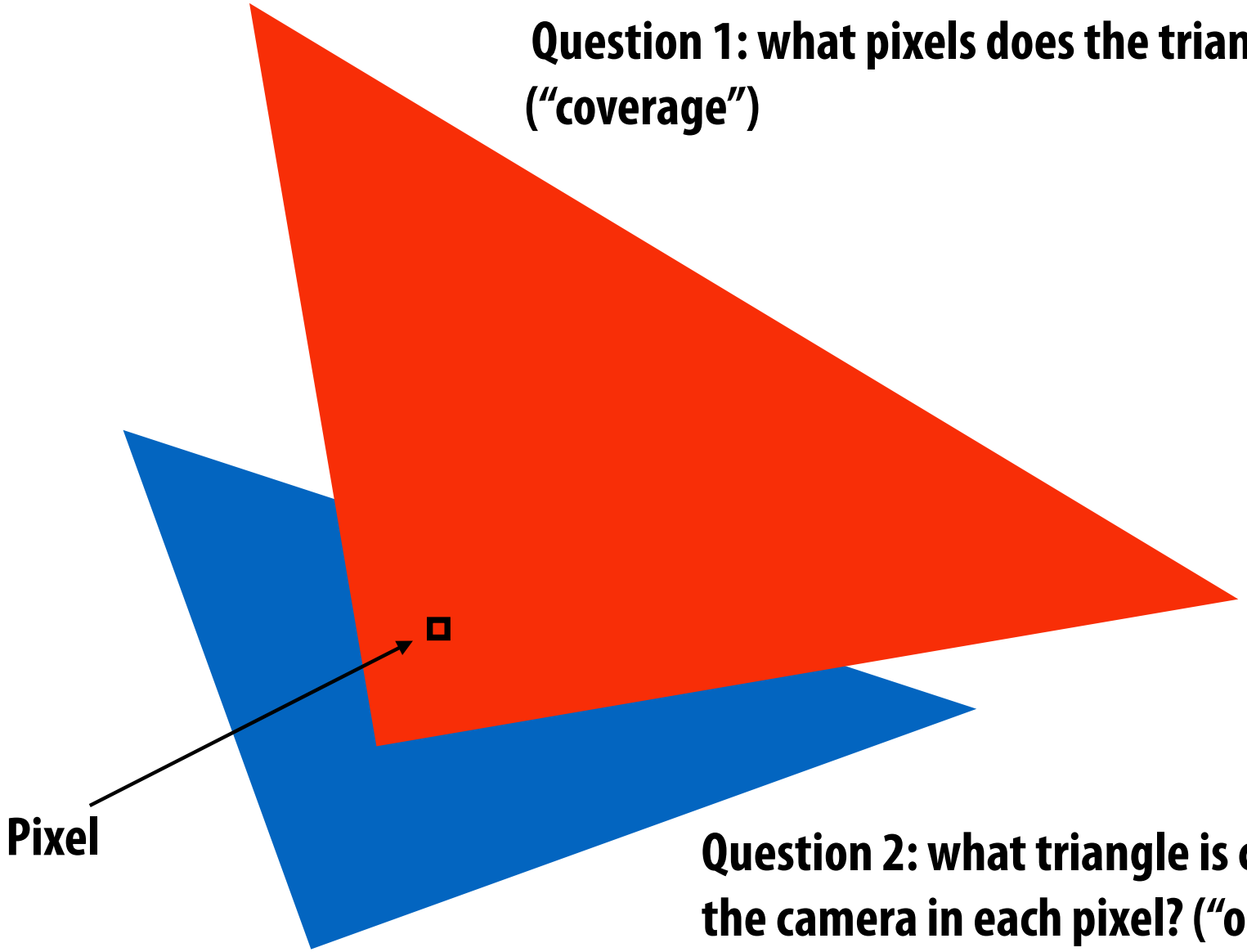
Lecture 2:

Drawing a Triangle (and an introduction to sampling)

Computer Graphics

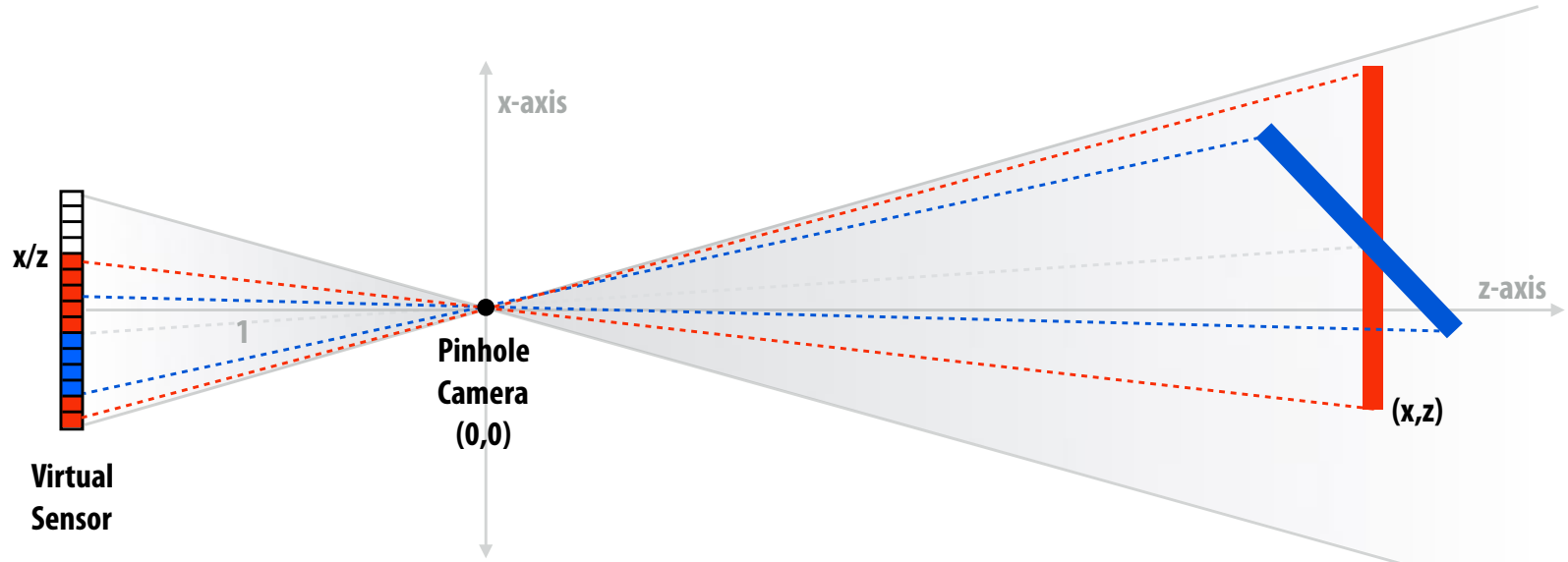
CMU 15-462/15-662, Spring 2016

Let's draw some triangles on the screen



The visibility problem

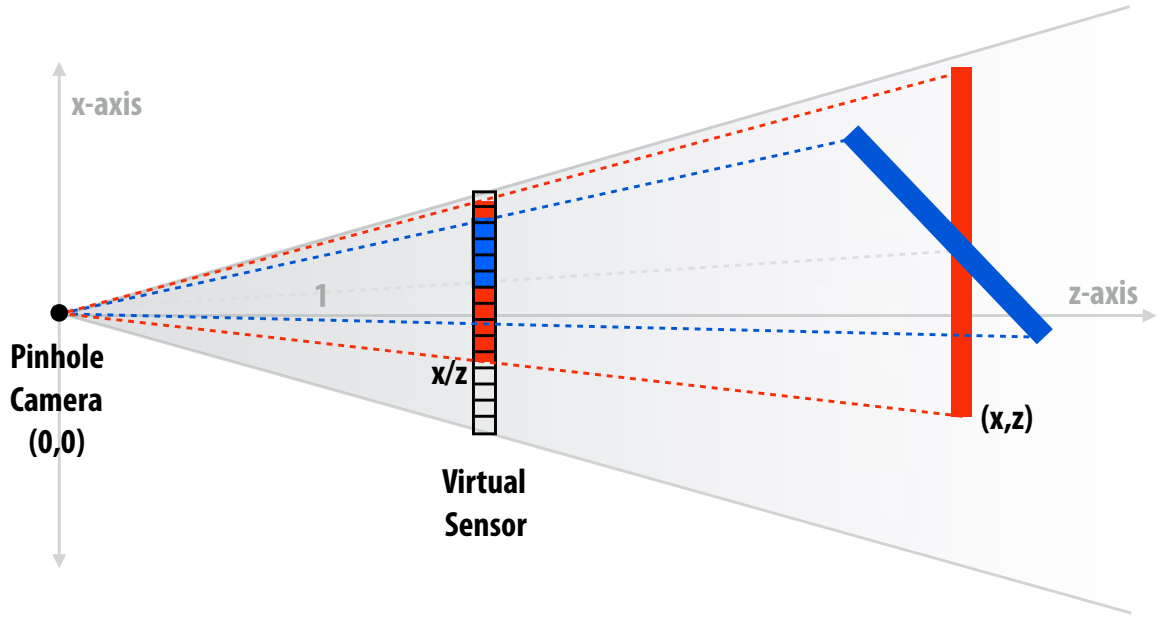
- **An informal definition: what scene geometry is visible within each screen pixel?**
  - What scene geometry projects into a screen pixel? (coverage)
  - Which geometry is visible from the camera at that pixel? (occlusion)



Recall perspective projection from last class

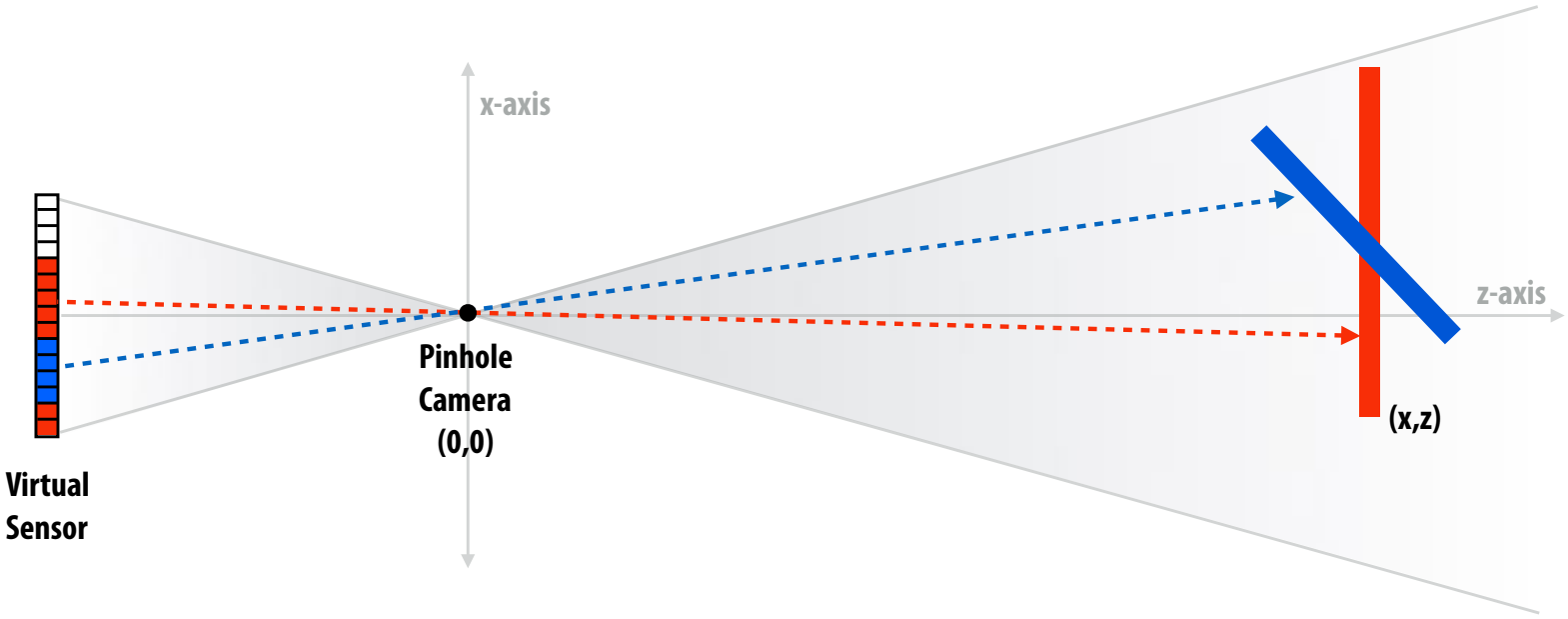
The visibility problem

- **An informal definition: what scene geometry is visible within each screen pixel?**
  - What scene geometry projects into a screen pixel? (coverage)
  - Which geometry is visible from the camera at that pixel? (occlusion)



The visibility problem (said differently)

- In terms of rays:
  - What scene geometry is hit by a ray from a pixel through the pinhole? (coverage)
  - What object is the first hit along that ray? (occlusion)

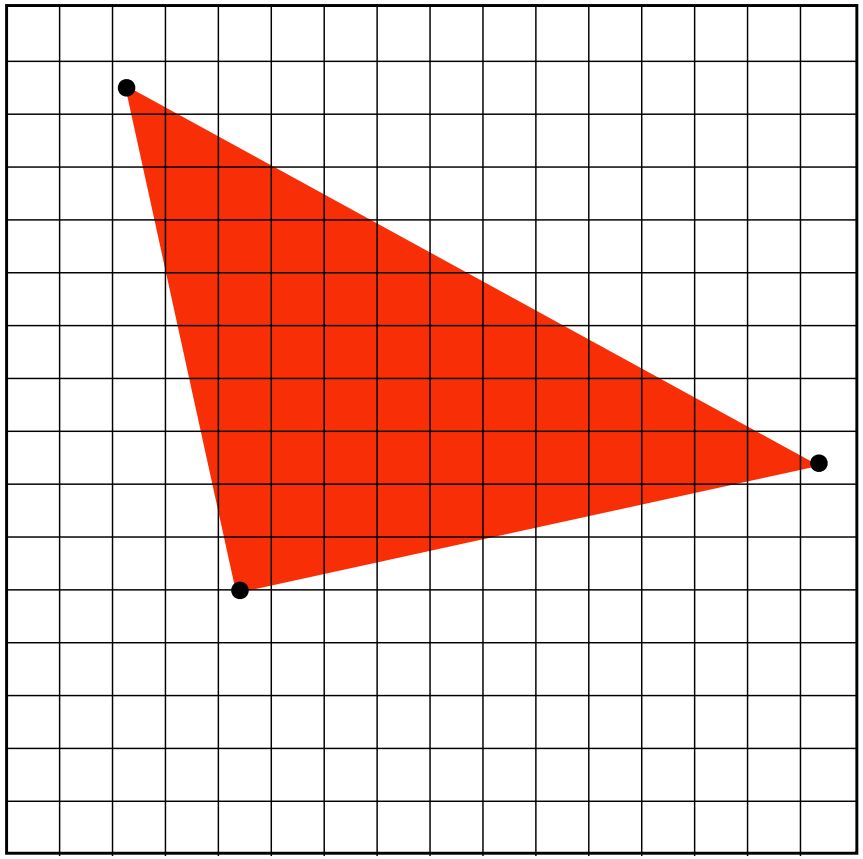


Hold onto this thought for later in the semester.

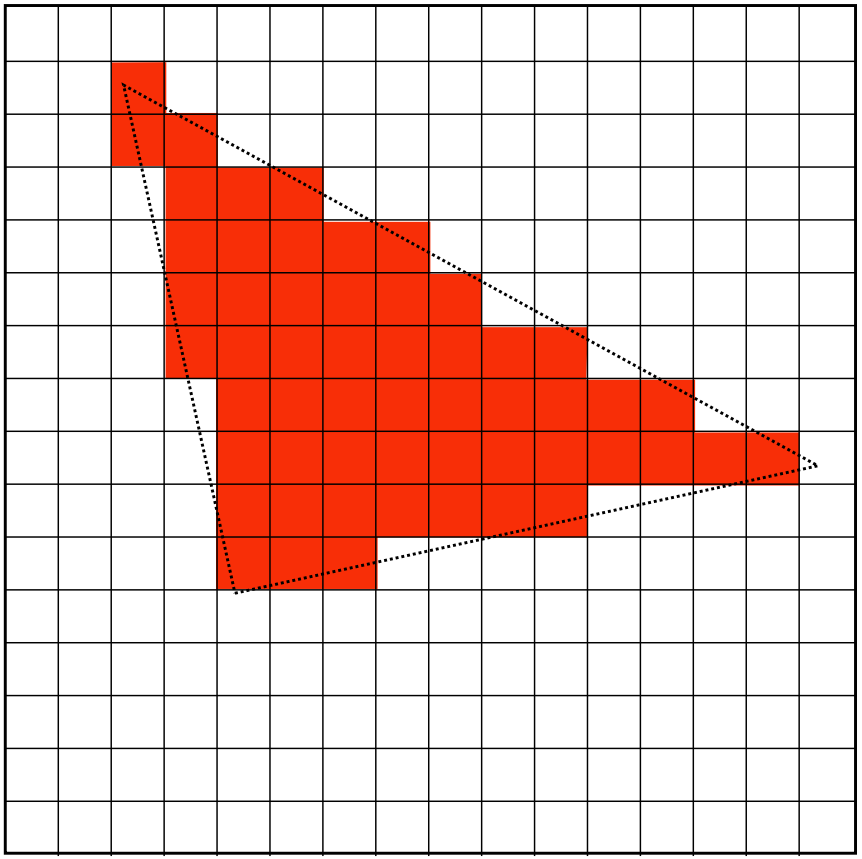
Computing triangle coverage

What pixels does the triangle overlap?

Input:
projected position of triangle vertices: P_0, P_1, P_2

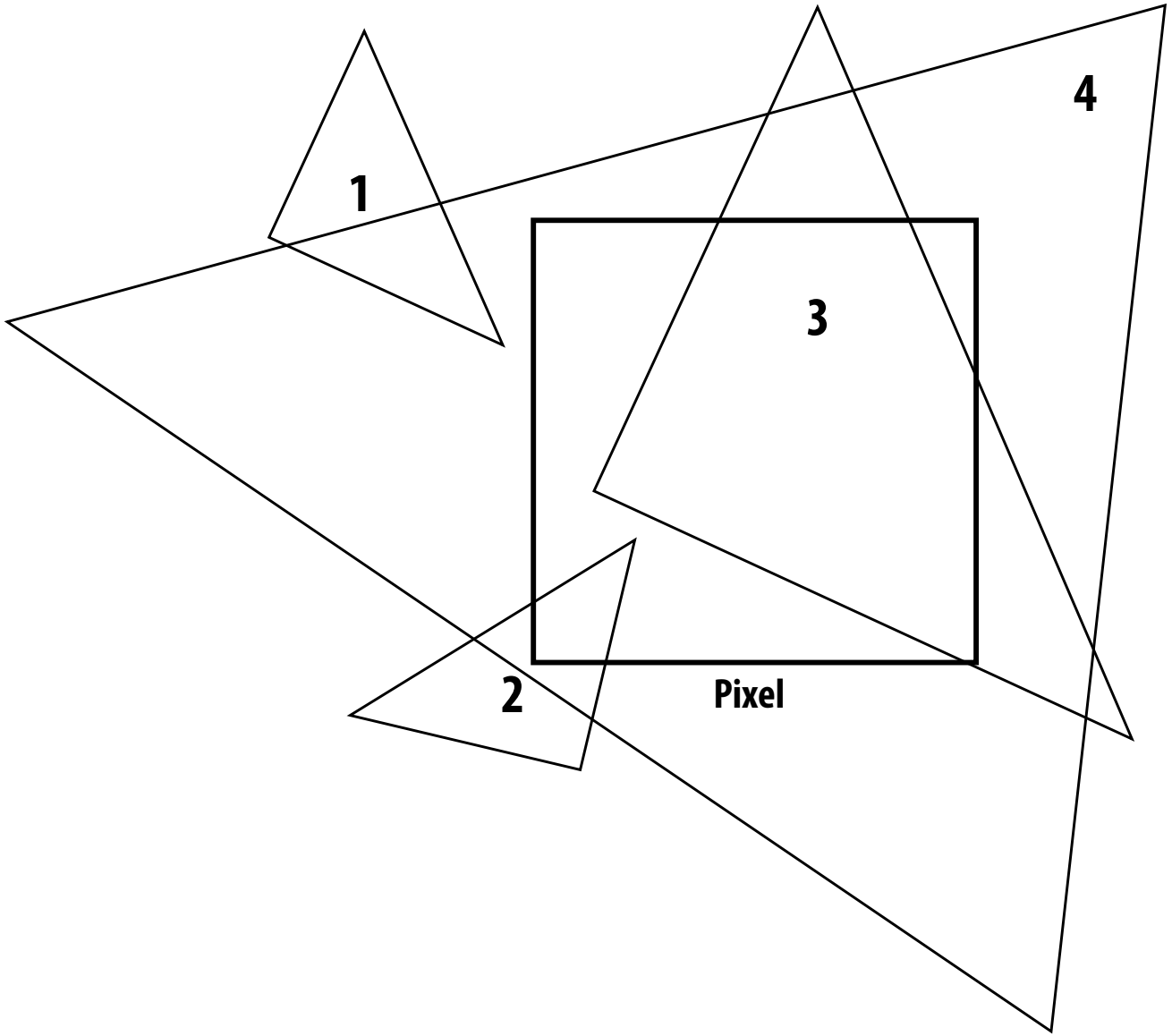


Output:
set of pixels "covered" by the triangle

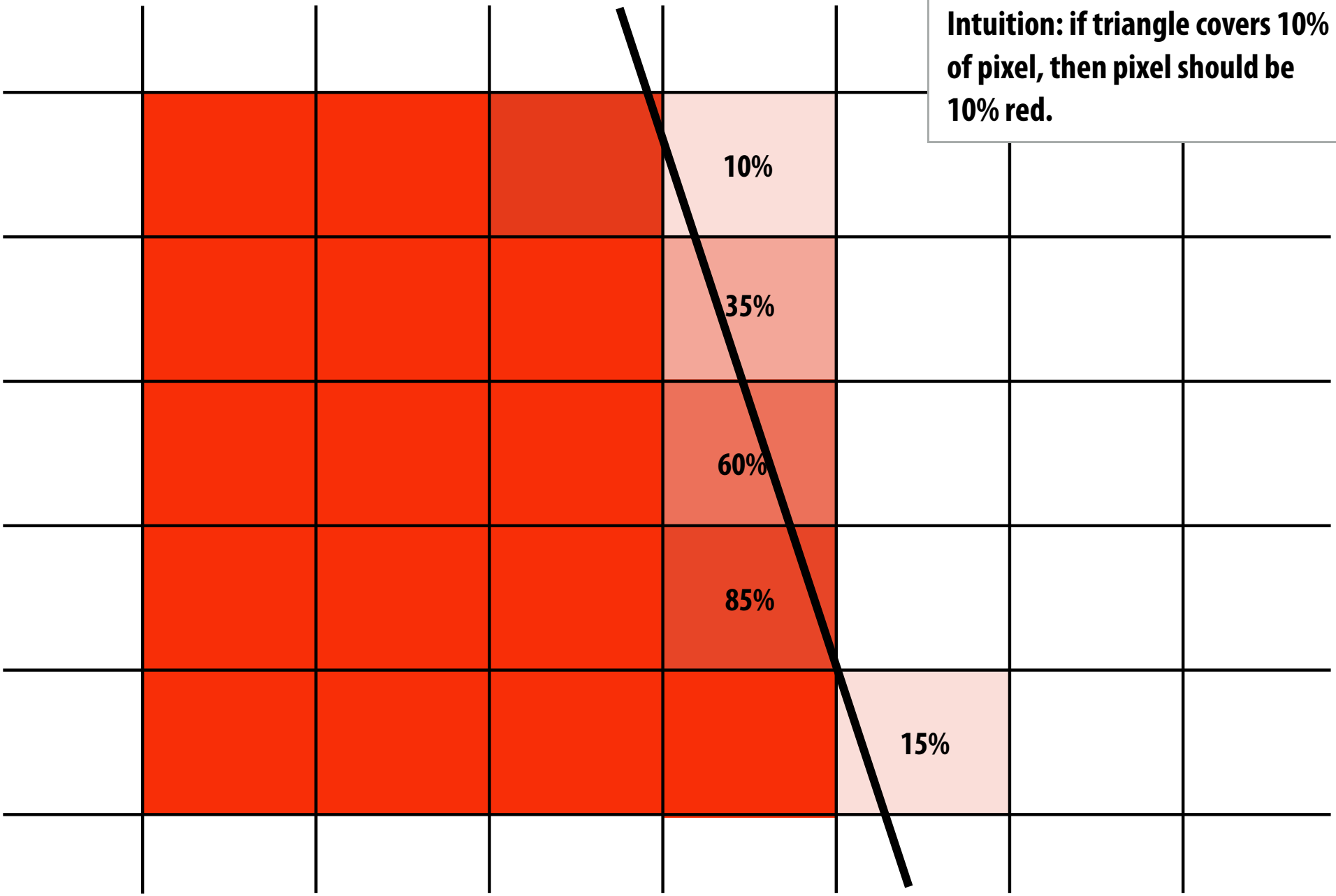


What does it mean for a pixel to be covered by a triangle?

Question: which triangles "cover" this pixel?

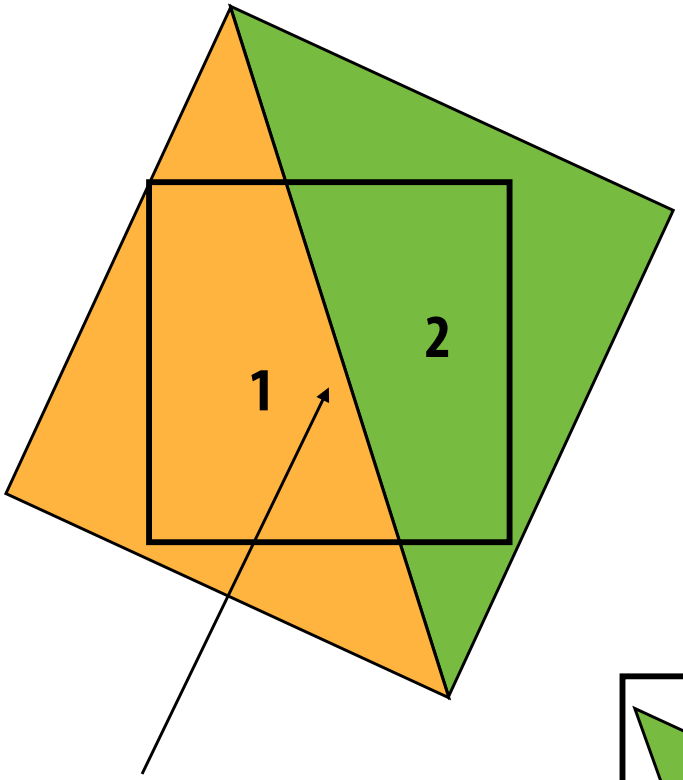


One option: compute fraction of pixel area covered by triangle, then color pixel according to this fraction.

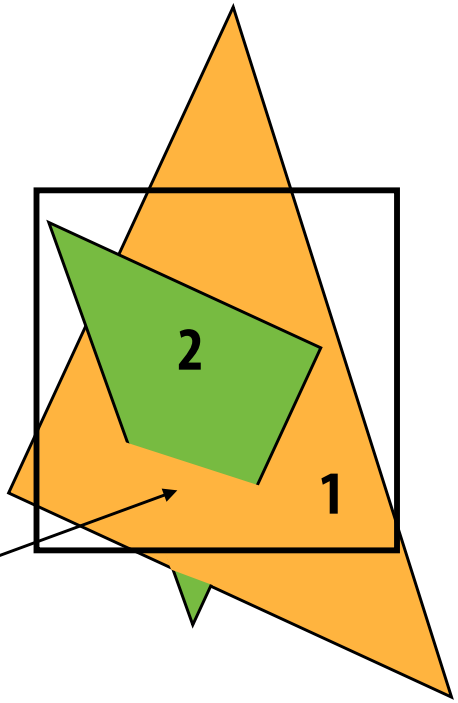


Intuition: if triangle covers 10% of pixel, then pixel should be 10% red.

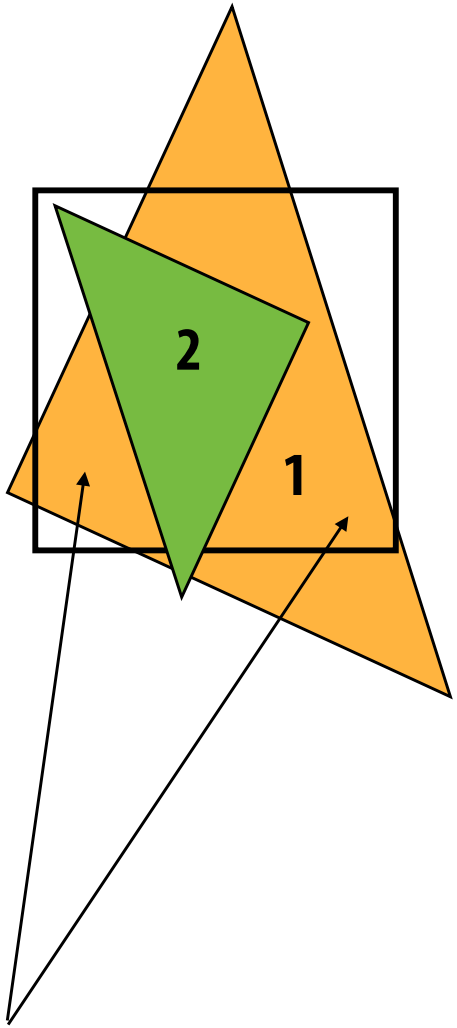
Analytical schemes get tricky when considering occlusion



Pixel covered by triangle 1, other half covered by triangle 2



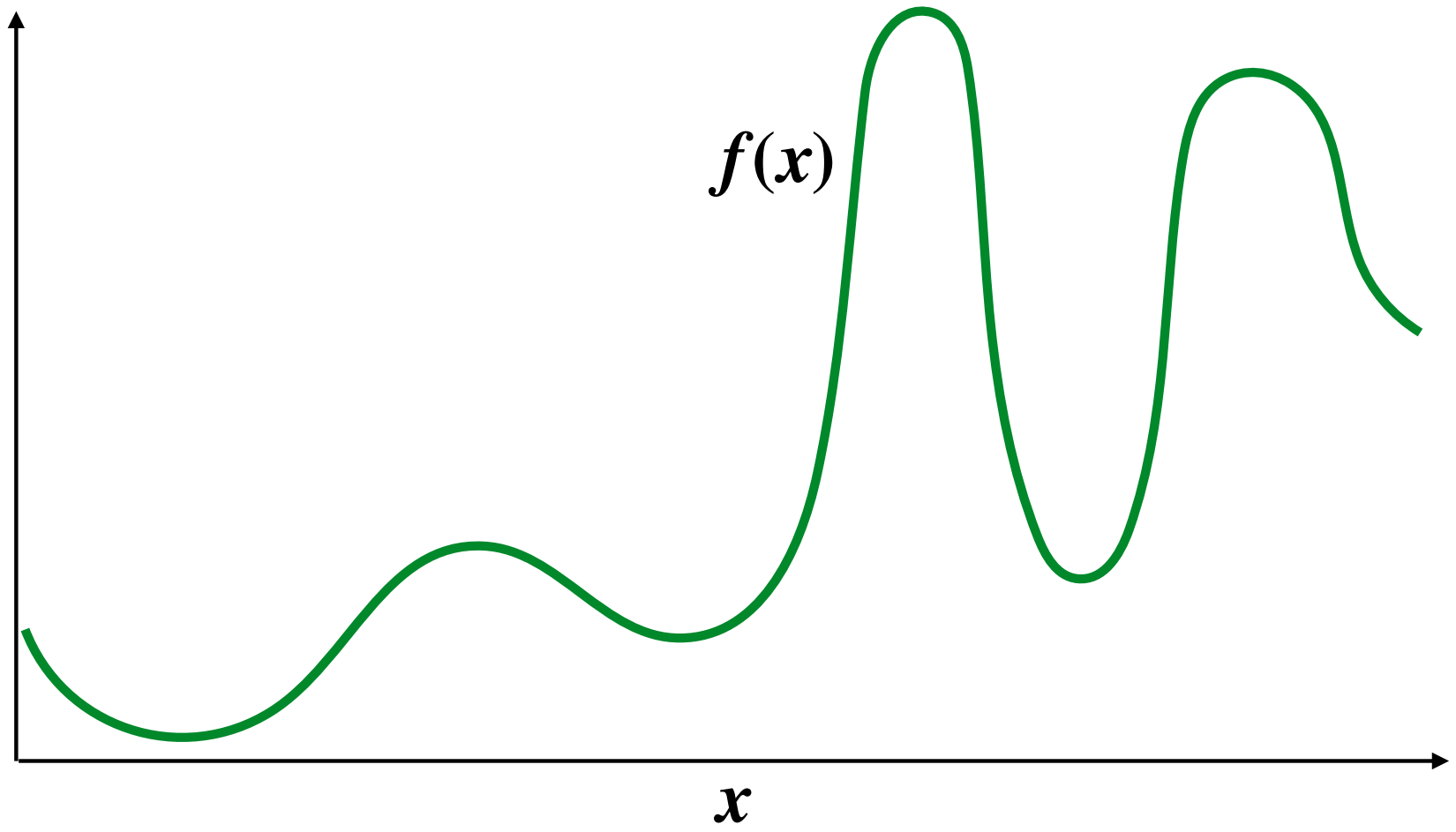
Interpenetration of triangles: even trickier



Two regions of triangle 1 contribute to pixel.
One of these regions is not even convex.

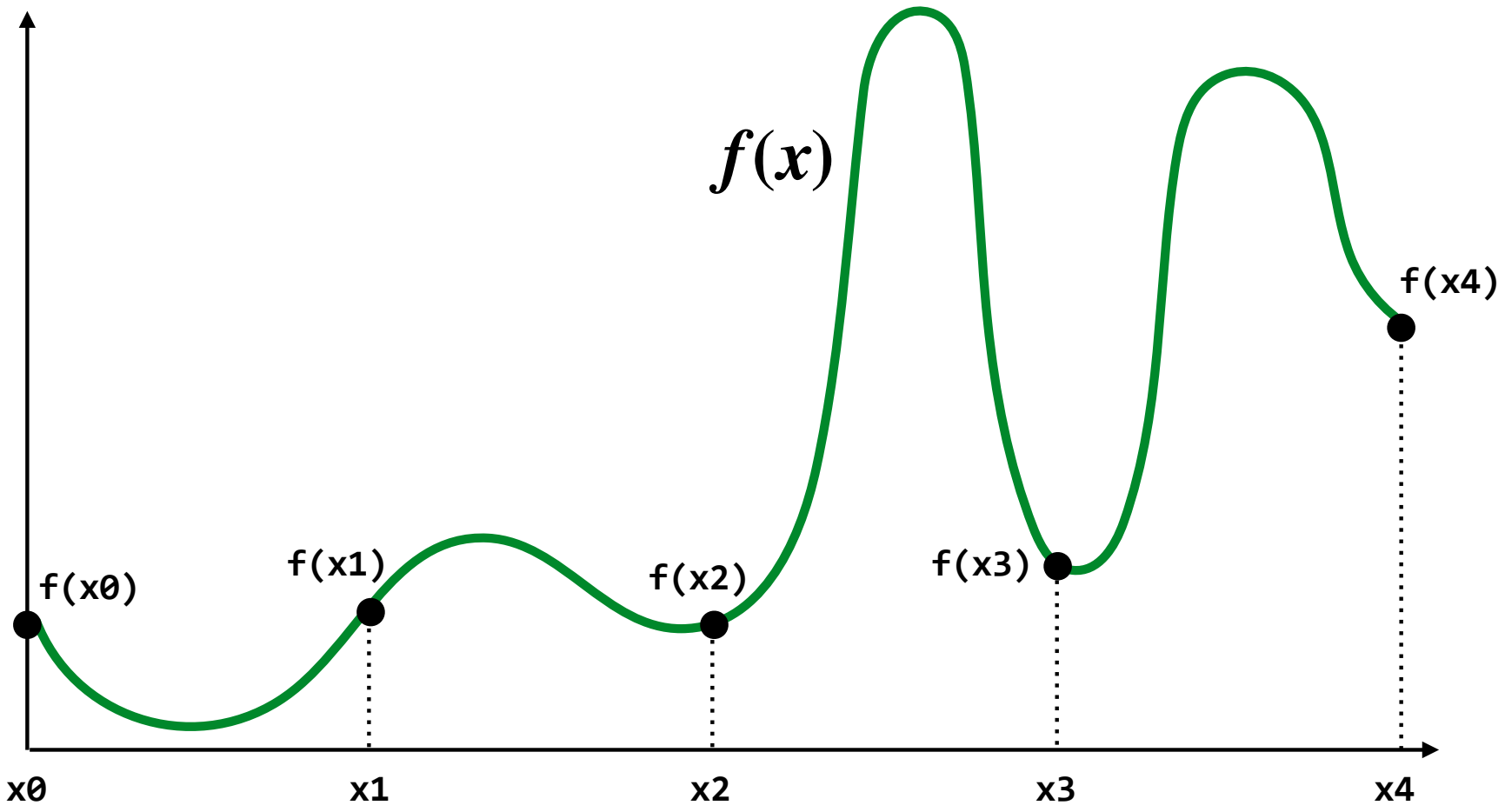
Sampling 101

1D signal



Sampling: taking measurements a signal

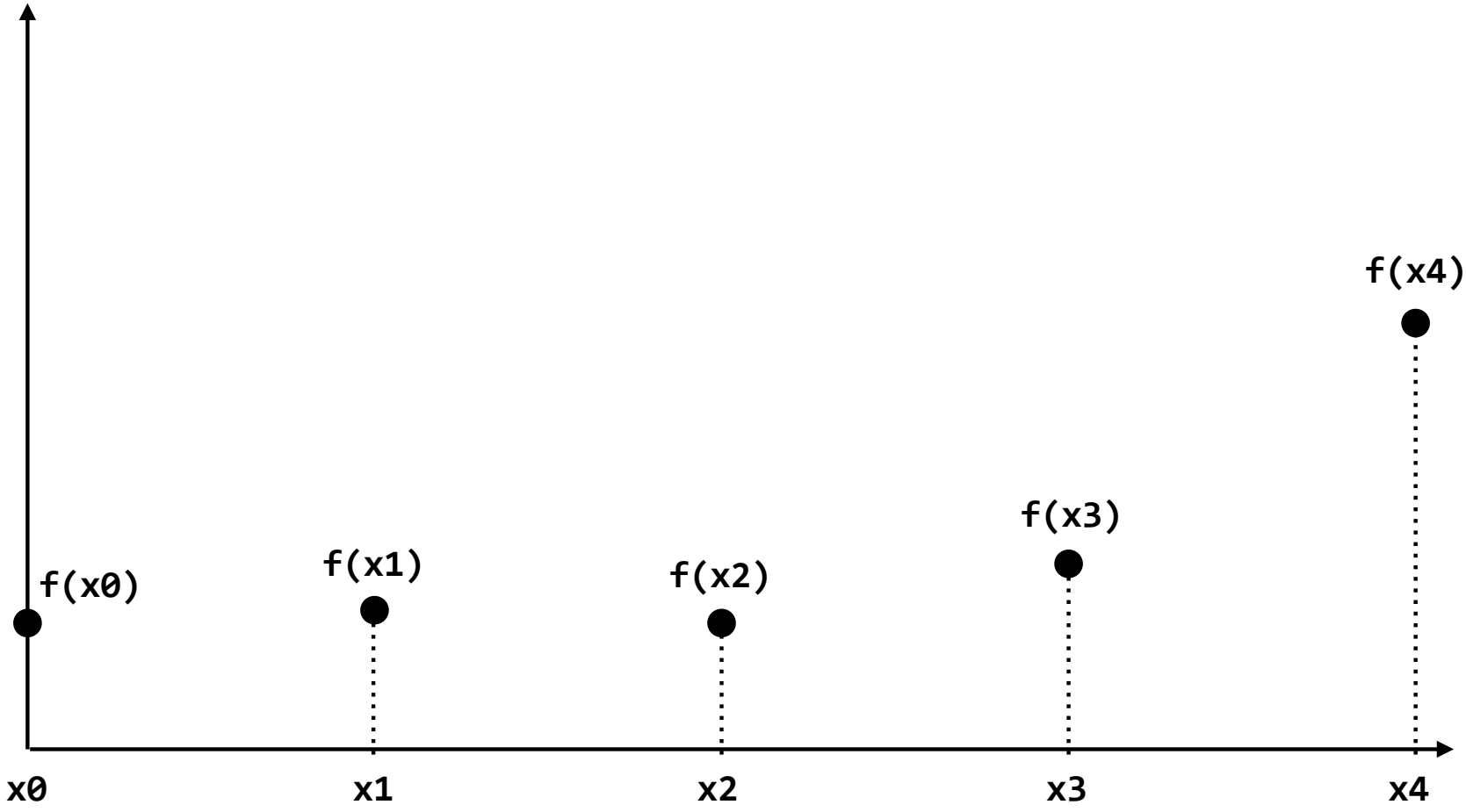
Below: 5 measurements ("samples") of $f(x)$



Audio file: stores samples of a 1D signal

Most consumer audio is sampled at 44.1 KHz

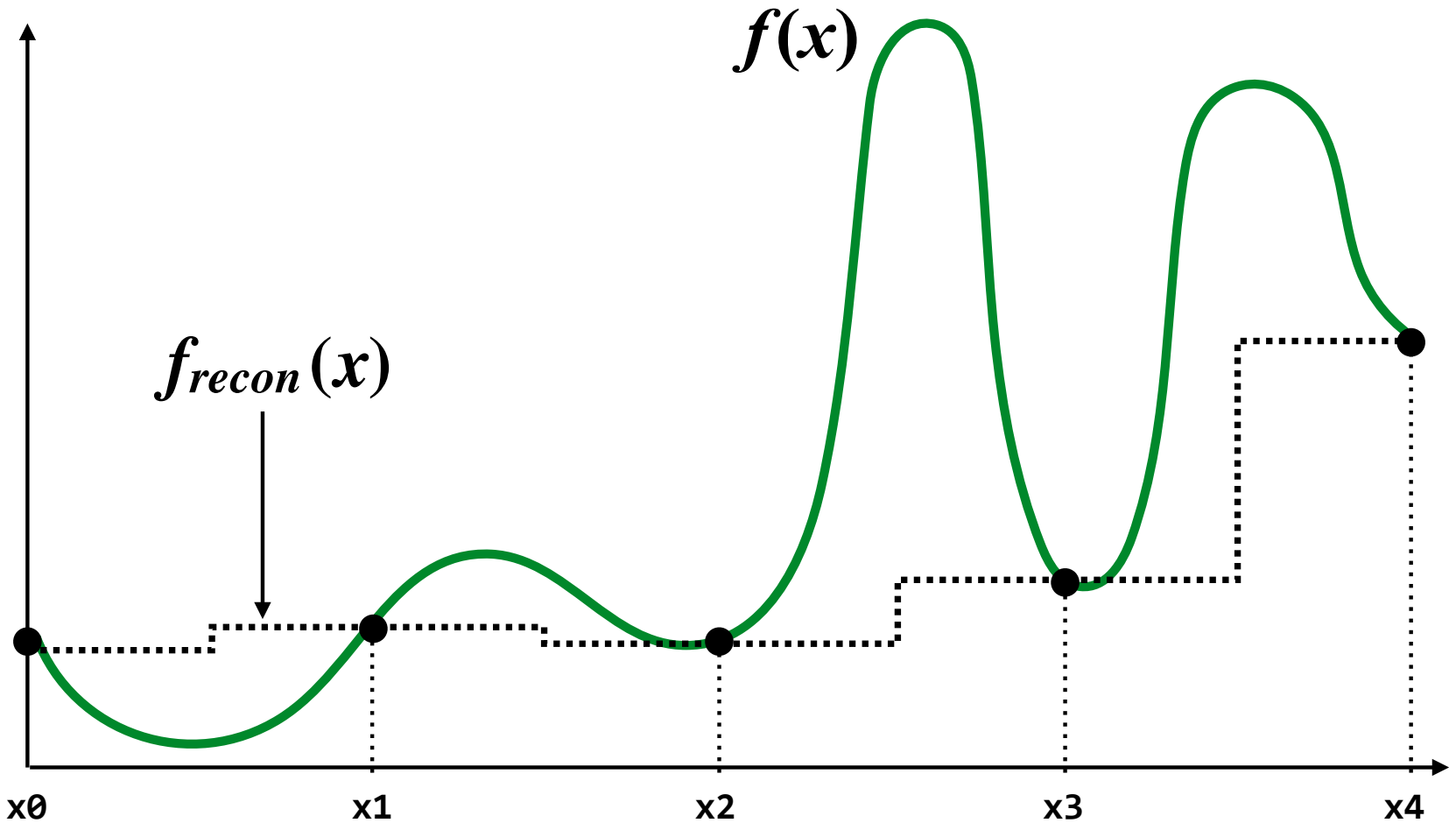
Reconstruction: given a set of samples, how might we attempt to reconstruct the original signal $f(x)$?



Piecewise constant approximation

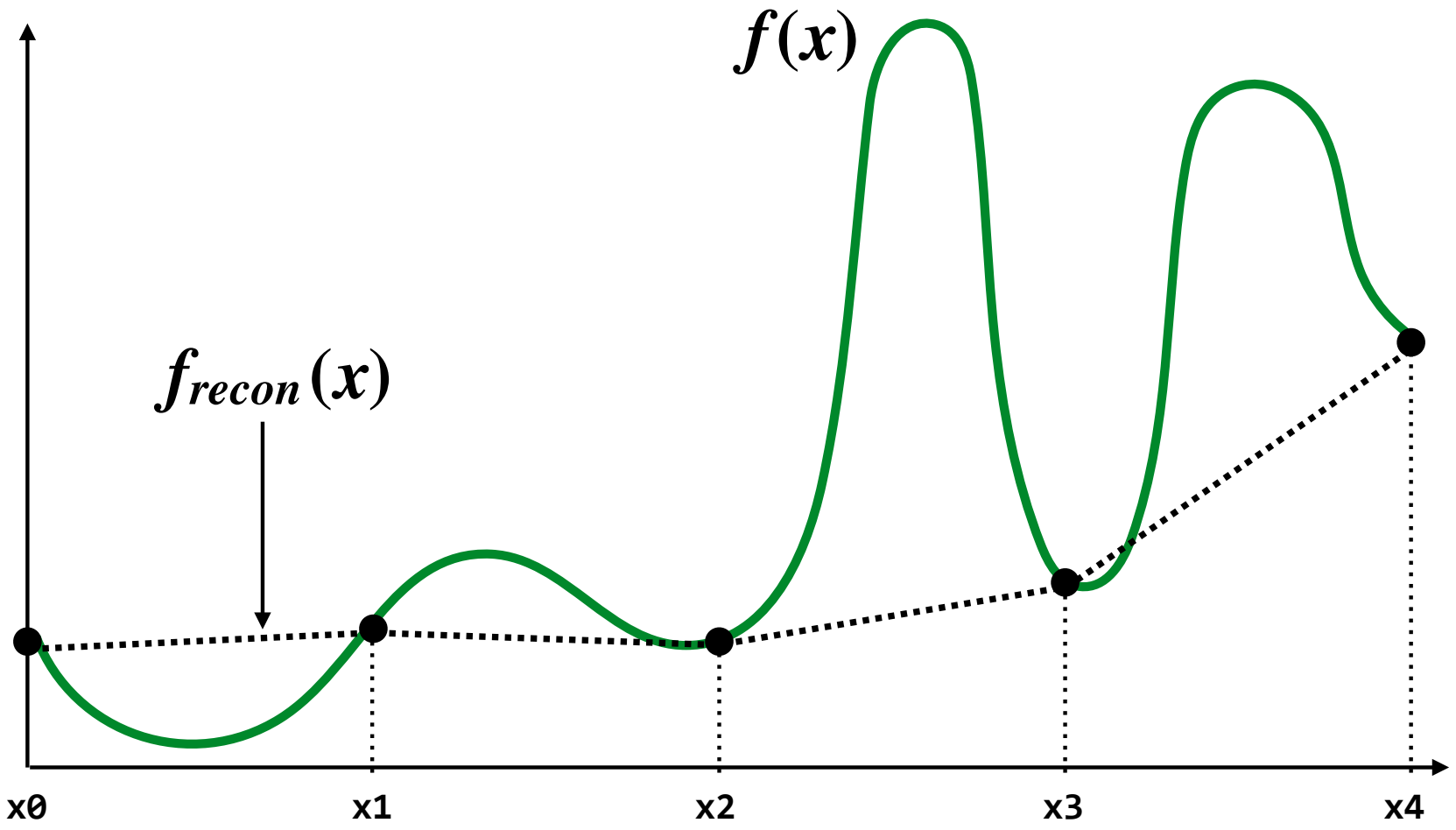
$f_{recon}(x)$ = value of sample closest to x

$f_{recon}(x)$ approximates $f(x)$

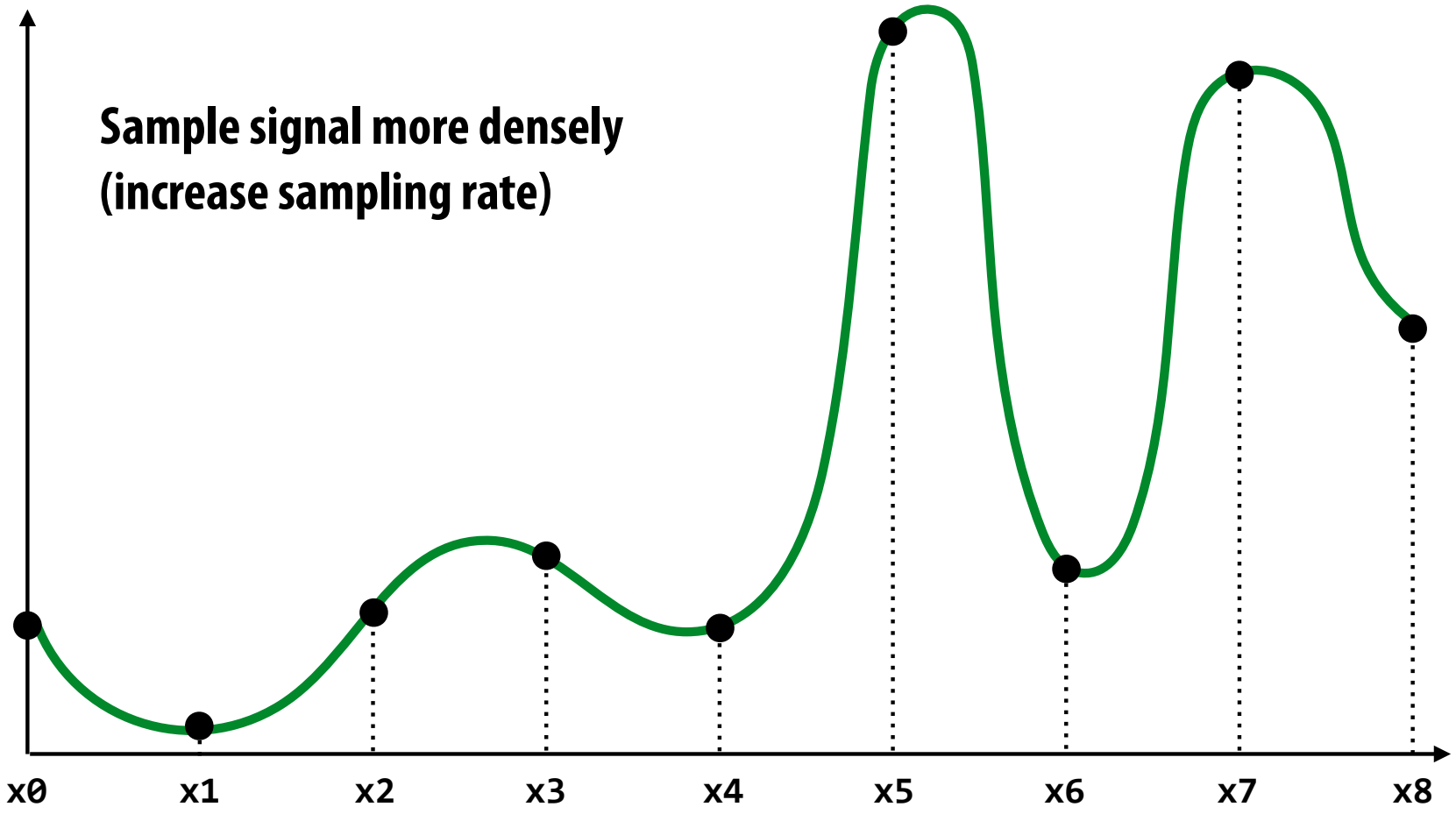


Piecewise linear approximation

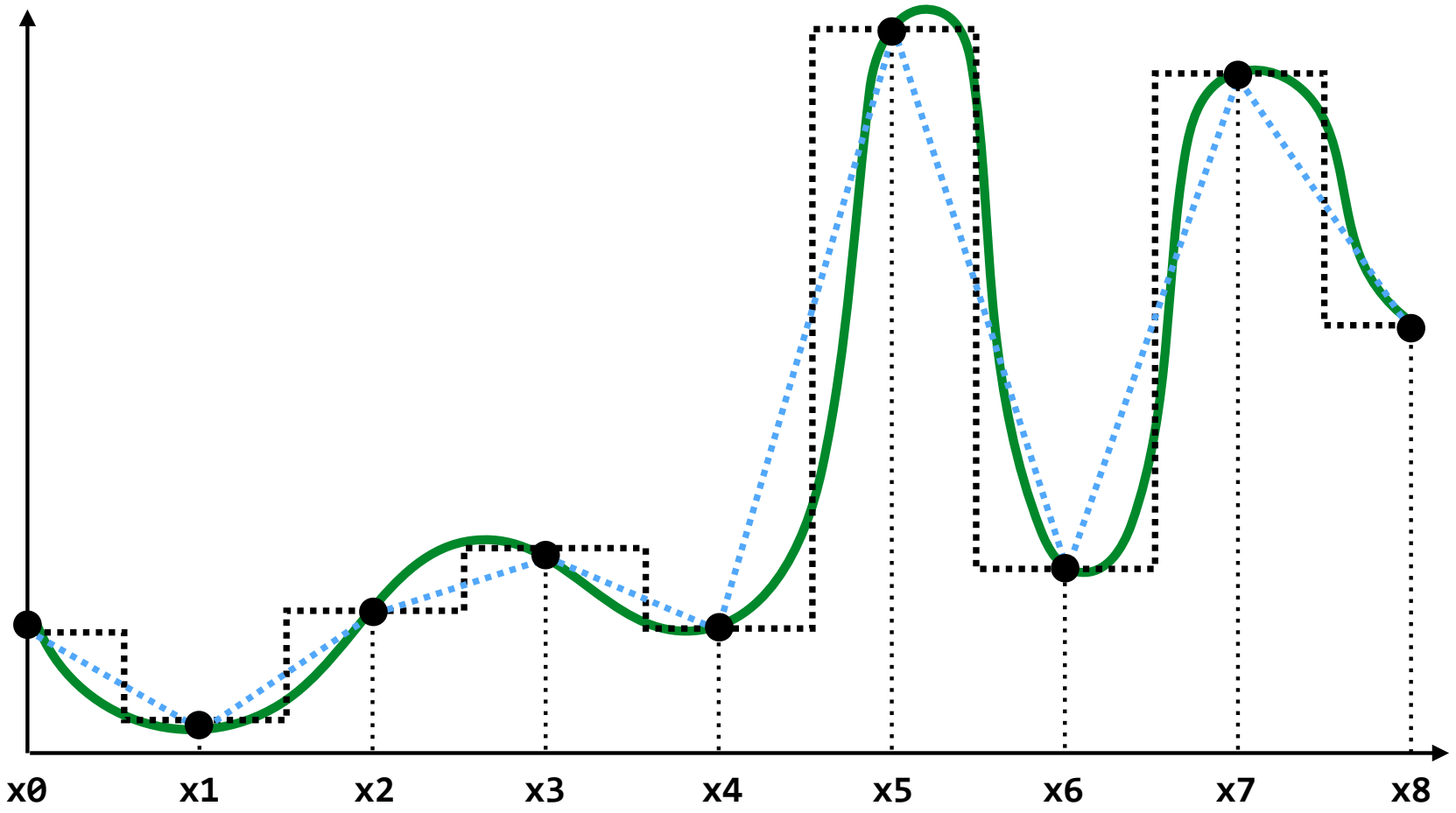
$f_{recon}(x)$ = linear interpolation between values of two closest samples to x



How can we represent the signal more accurately?



Reconstruction from denser sampling



= reconstruction via nearest

= reconstruction via linear interpolation

Mathematical representation of sampling

Consider the Dirac delta: $\delta(x)$

where for all $x \neq 0, \delta(x) = 0$ **and** $\int_{-\infty}^{\infty} \delta(x)dx = 1$

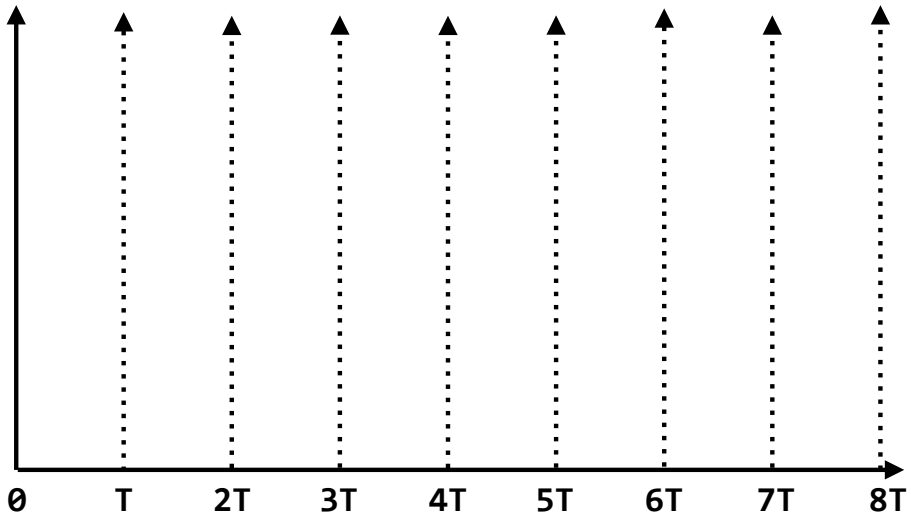
When applied to a function *f*, $\delta(x)$ acts to pull out the value of *f* at $x = 0$:

$$\int_{-\infty}^{\infty} f(x)\delta(x)dx = f(0)$$

Sampling function: $\text{Ш}_T(x)$

Consider a sequence of impulses with period T:

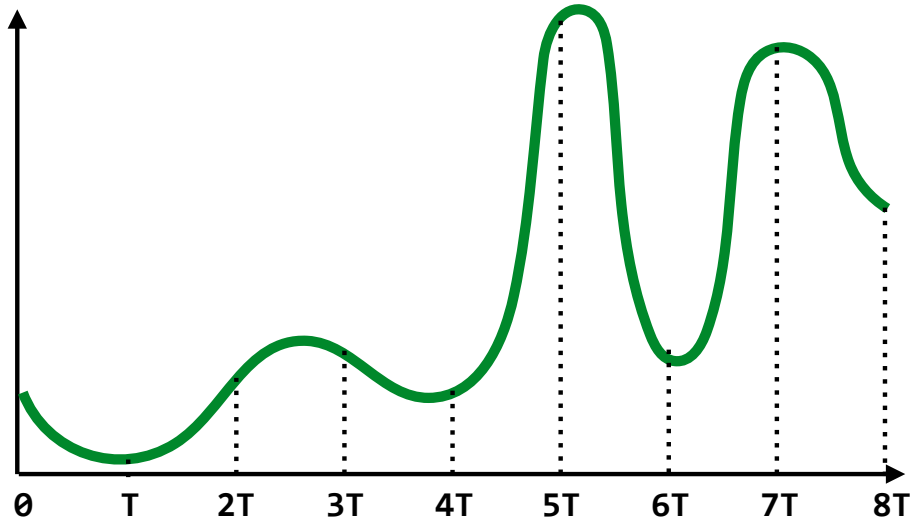
$$\text{Ш}_T(x) = T \sum_{i=-\infty}^{\infty} \delta(x - iT)$$

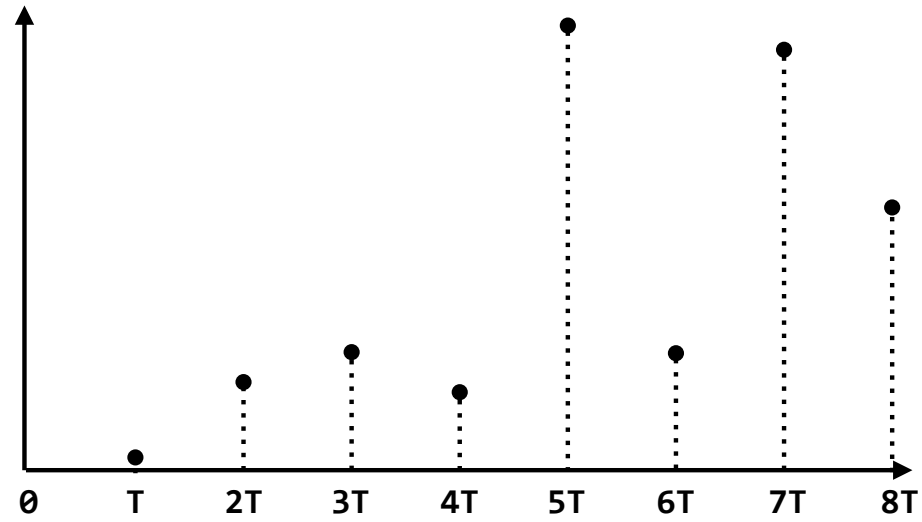


$$\text{Ш}_T(x) f(x) = T \sum_{i=-\infty}^{\infty} f(iT)\delta(x - iT)$$

So we can write the result of sampling as a product of $f(x)$ and a sequence of impulses centered around each sample point

$f(x)$





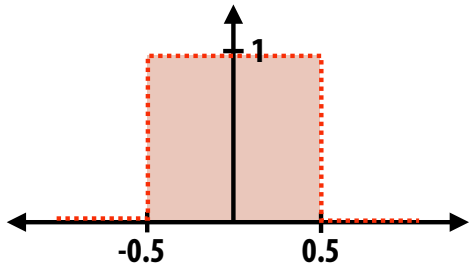
Convolution

$$(f * g)(x) = \int_{-\infty}^{\infty} f(y)g(x-y)dy$$

output signal — filter — input signal

It may be helpful to consider the effect of convolution with the simple unit-area "box" function:

$$f(x) = \begin{cases} 1 & |x| \leq 0.5 \\ 0 & otherwise \end{cases}$$



$$(f * g)(x) = \int_{-0.5}^{0.5} g(x-y)dy$$

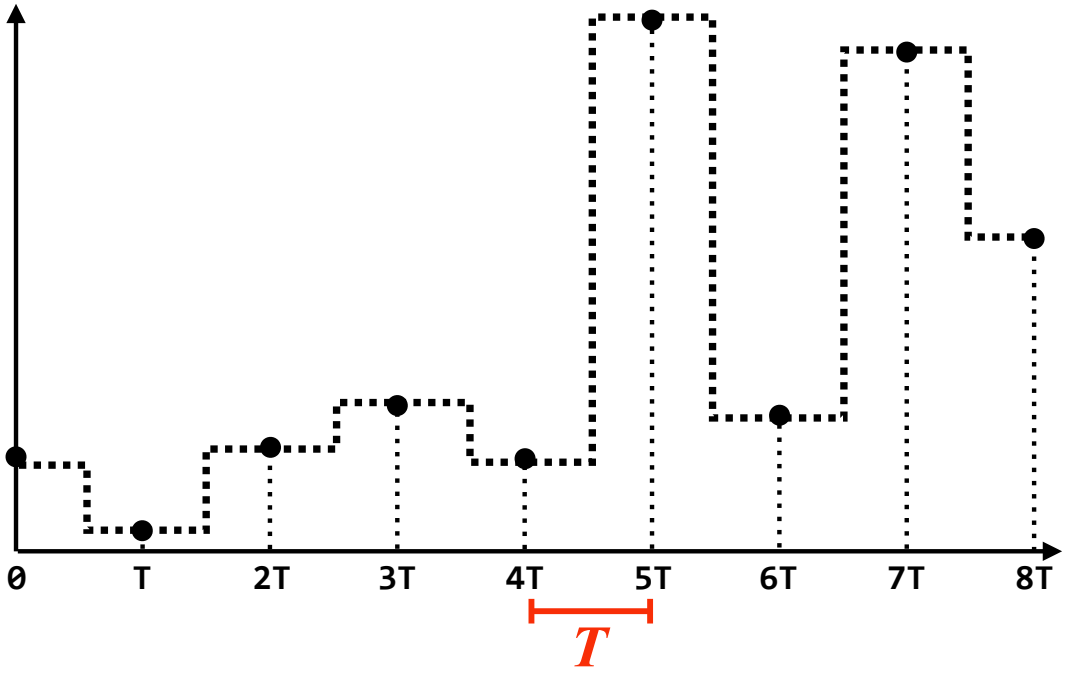
f * *g* is a "smoothed" version of *g*

Reconstruction as convolution (box filter)

Sampled signal:

(with period T)

$$g(x) = \text{Ш}_T(x) f(x) = T \sum_{i=-\infty}^{\infty} f(iT)\delta(x - iT)$$



Reconstruction filter:

(unit area box of width T)

$$h(x) = \begin{cases} 1/T & |x| \le T/2 \\ 0 & otherwise \end{cases}$$

Reconstructed signal:

(chooses nearest sample)

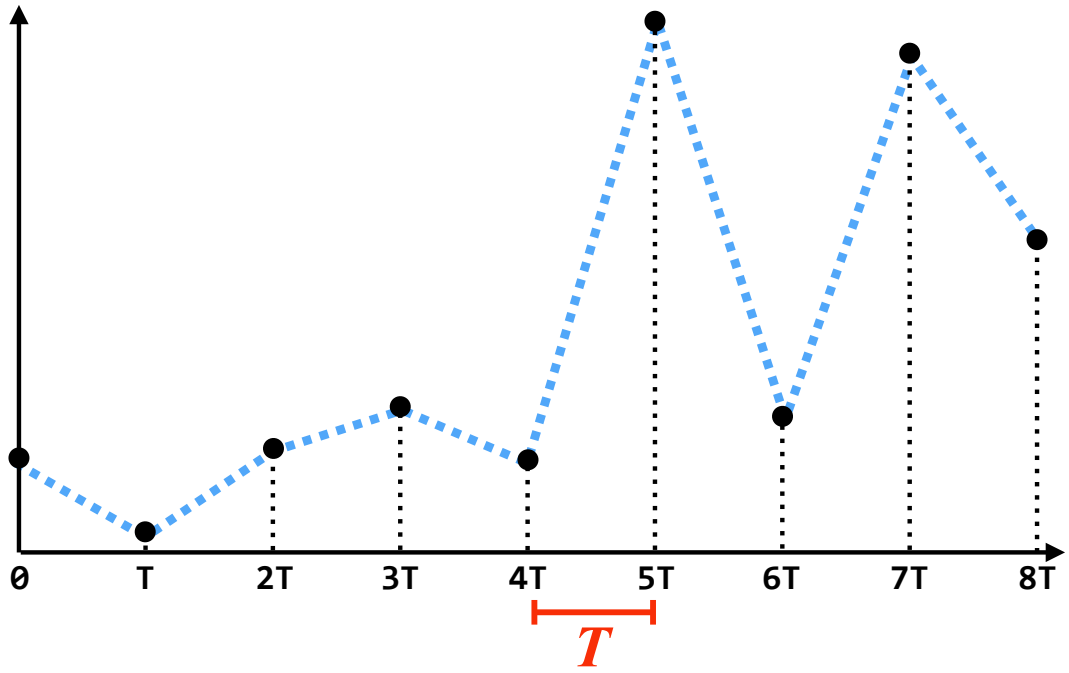
$$f_{recon}(x) = (h * g)(x) = T \int_{-\infty}^{\infty} h(y) \sum_{i=-\infty}^{\infty} f(iT)\delta(x - y - iT)dy = \int_{-T/2}^{T/2} \sum_{i=-\infty}^{\infty} f(iT)\delta(x - y - iT)$$

non-zero only for iT closest to x

Reconstruction as convolution (triangle filter)

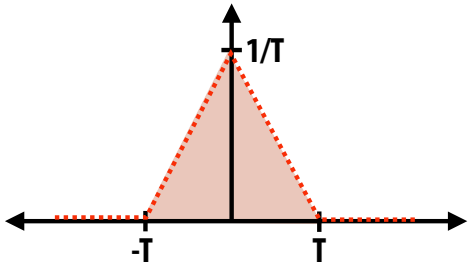
Sampled signal:
(with period T)

$$g(x) = \text{Ш}_T(x) f(x) = T \sum_{i=-\infty}^{\infty} f(iT)\delta(x - iT)$$



Reconstruction filter:
(unit area triangle of width T)

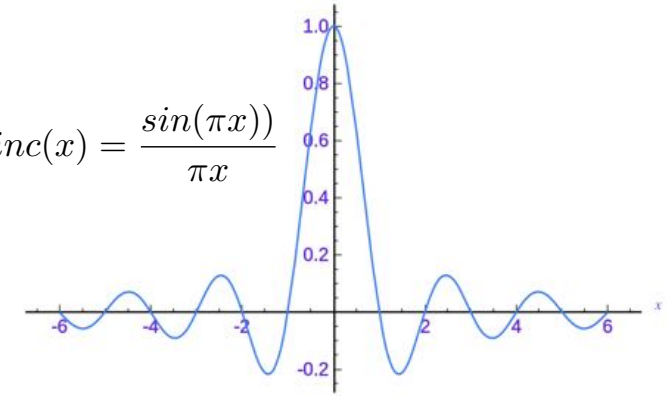
$$h(x) = \begin{cases} (1 - \frac{|x|}{T})/T & |x| \leq T \\ 0 & otherwise \end{cases}$$



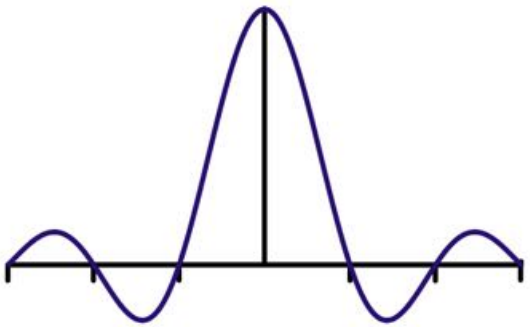
Summary

- Sampling = measurement of a signal
  - Represent signal as discrete set of samples
  - Mathematically described by multiplication by impulse train
- Reconstruction = generating signal from a discrete set of samples
  - Convolution of sampled signal with a reconstruction filter
  - Intuition: value of reconstructed function at any point in domain is a weighted combination of sampled values
  - We discussed simple box, triangle filters, but much higher quality filters exist

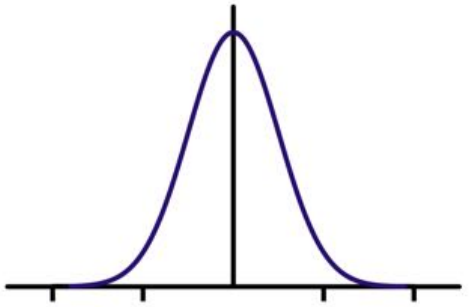
$$sinc(x) = \frac{sin(\pi x))}{\pi x}$$



Normalized sinc filter



Truncated sinc filter



Truncated gaussian filter

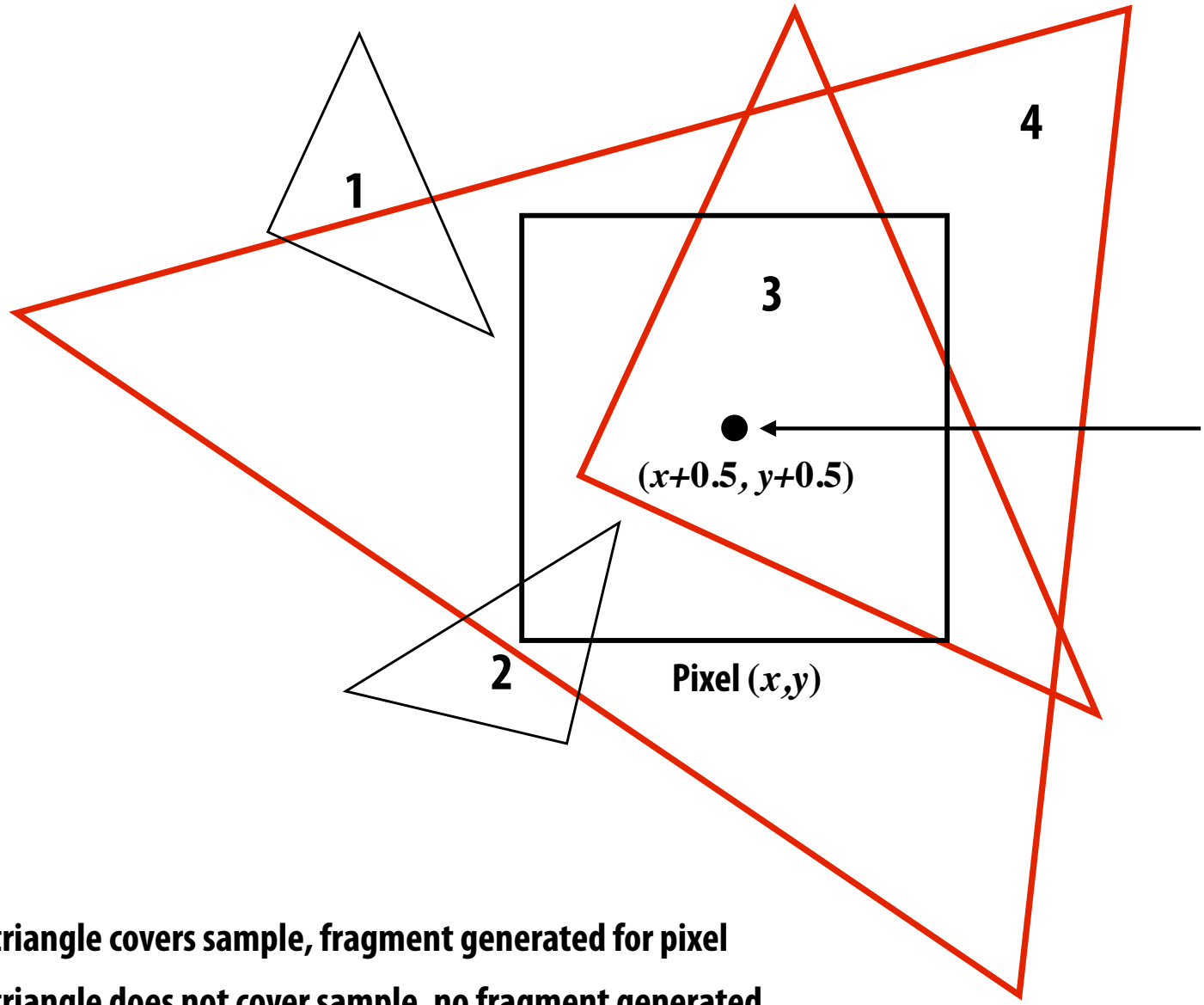
[Image credit: Wikipedia]

Now back to computing coverage

Think of coverage as a 2D signal

$$\text{coverage}(x,y) = \begin{cases} 1 & \text{if the triangle contains point } (x,y) \\ 0 & \text{otherwise} \end{cases}$$

Estimate triangle-screen coverage by sampling the binary function: `coverage(x,y)`



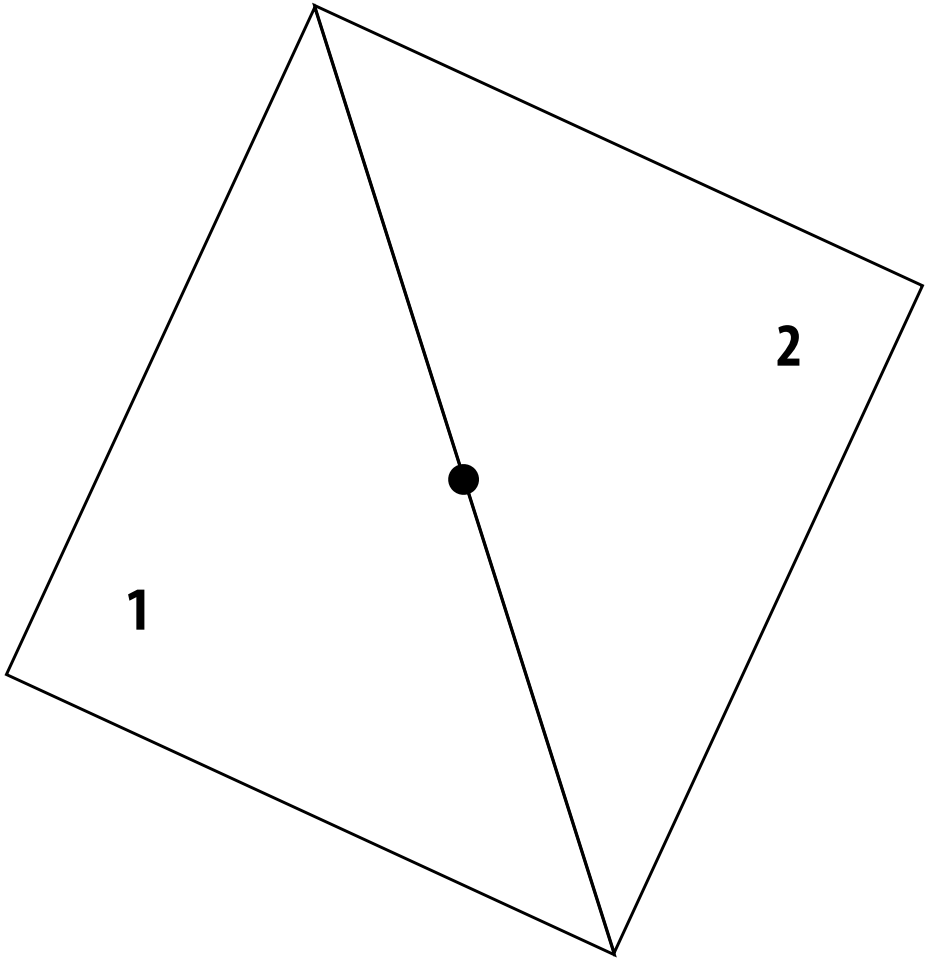
Example:
Here I chose the coverage sample point to be at a point corresponding to the pixel center.

= triangle covers sample, fragment generated for pixel

= triangle does not cover sample, no fragment generated

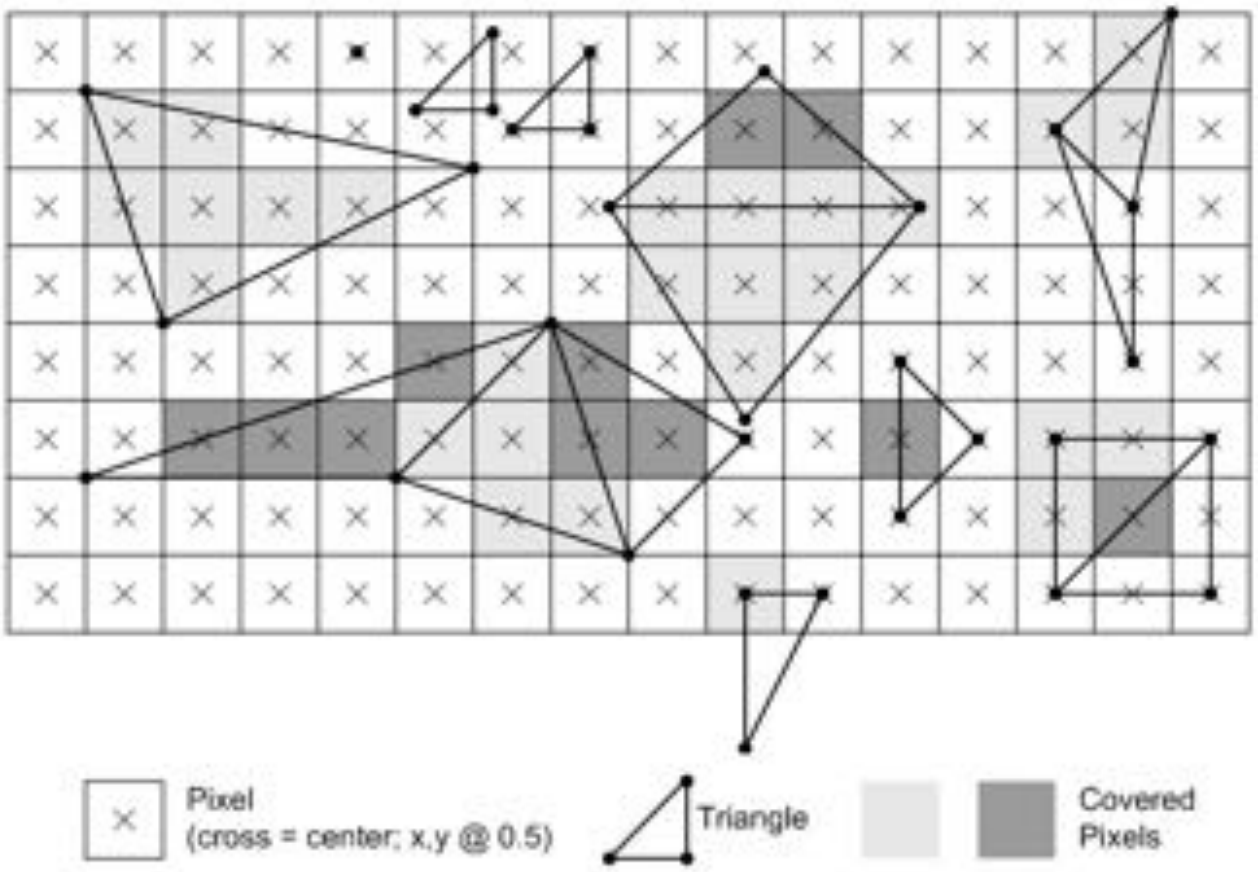
Edge cases (literally)

Is this sample point covered by triangle 1? or triangle 2? or both?



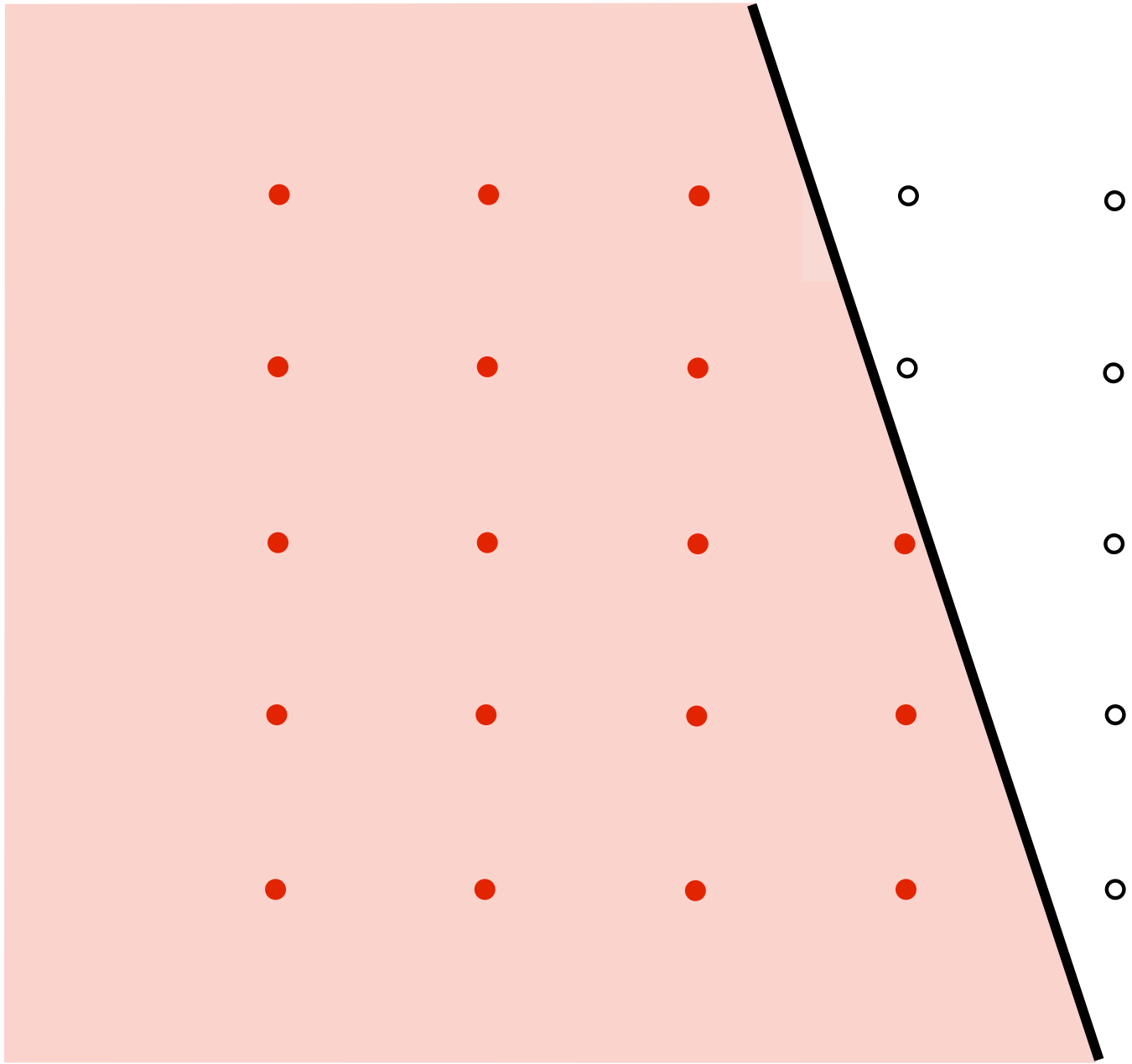
OpenGL/Direct3D edge rules

- When edge falls directly on a screen sample point, the sample is classified as within triangle if the edge is a "top edge" or "left edge"
  - Top edge: horizontal edge that is above all other edges
  - Left edge: an edge that is not exactly horizontal and is on the left side of the triangle. (triangle can have one or two left edges)

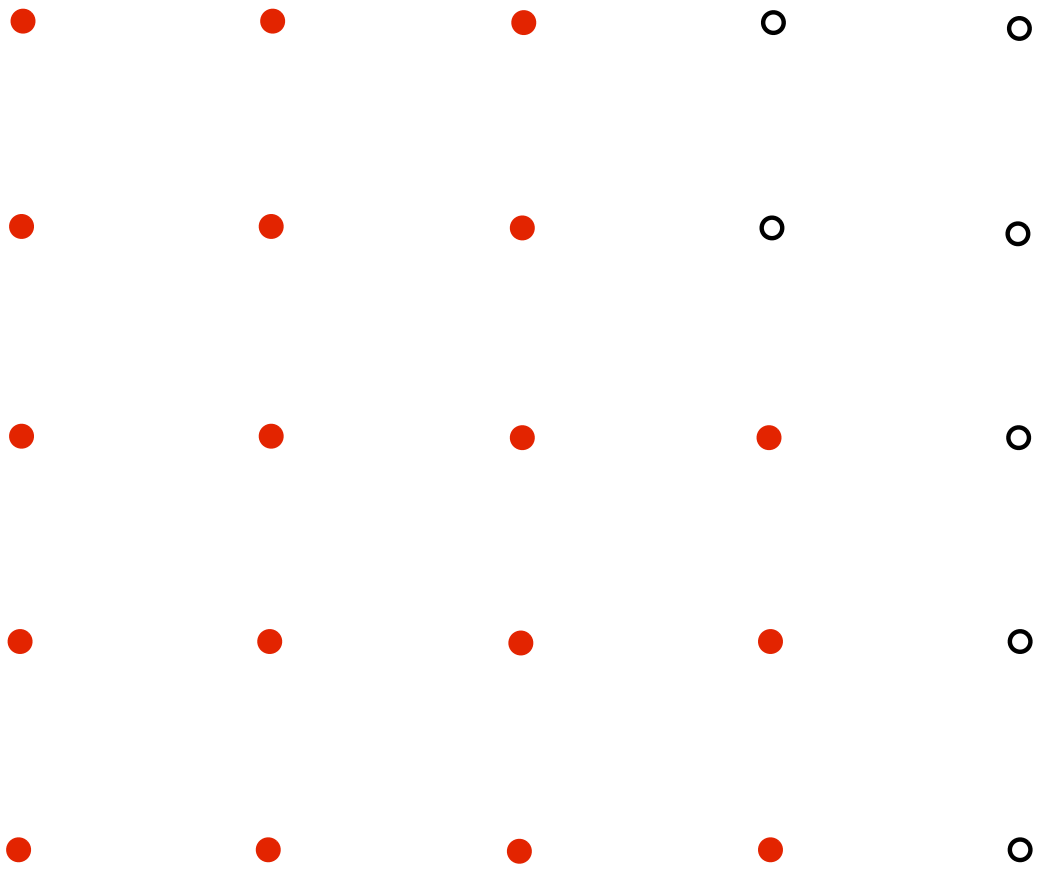


Source: Direct3D Programming Guide, Microsoft

Results of sampling triangle coverage

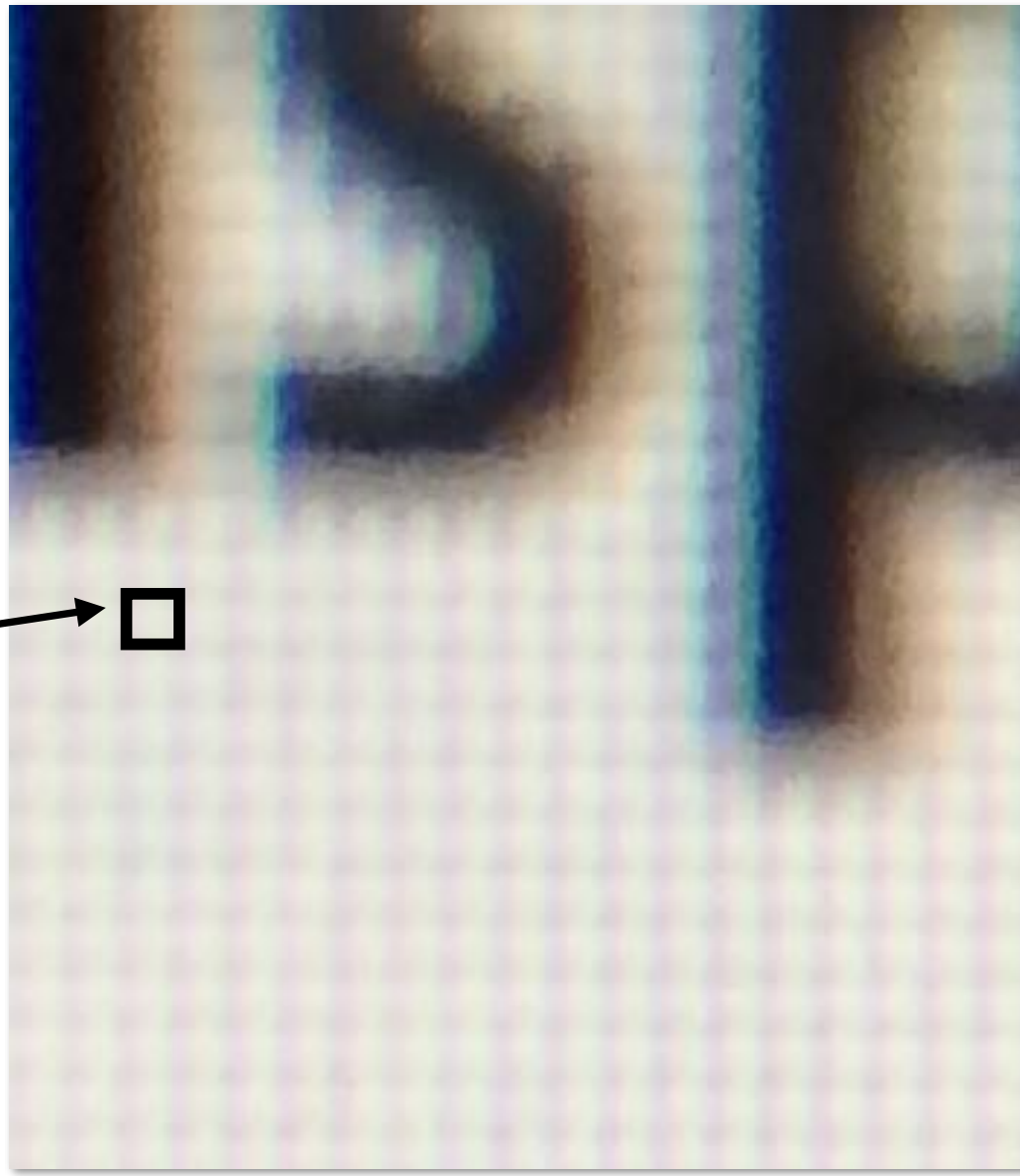


I have a sampled signal, now I want to display it on a screen



Pixels on a screen

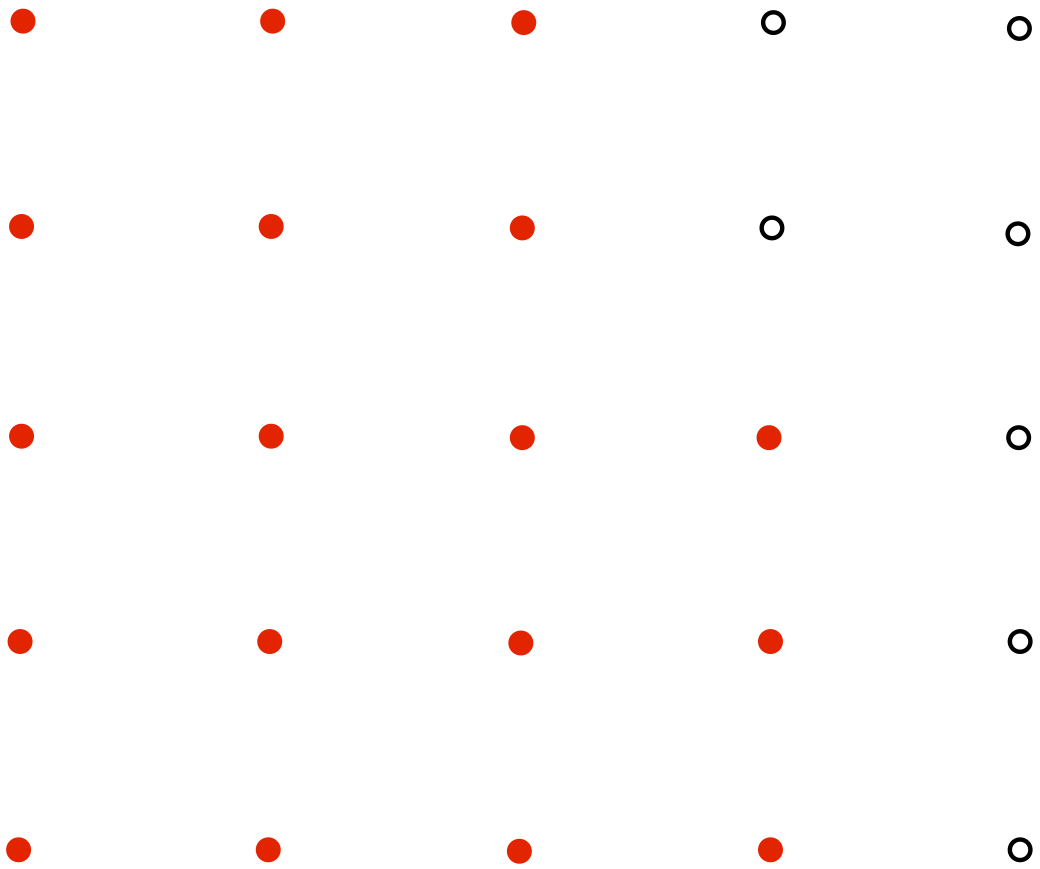
Each image sample sent to the display is converted into a little square of light of the appropriate color:
(a pixel = picture element)



LCD display pixel on my laptop

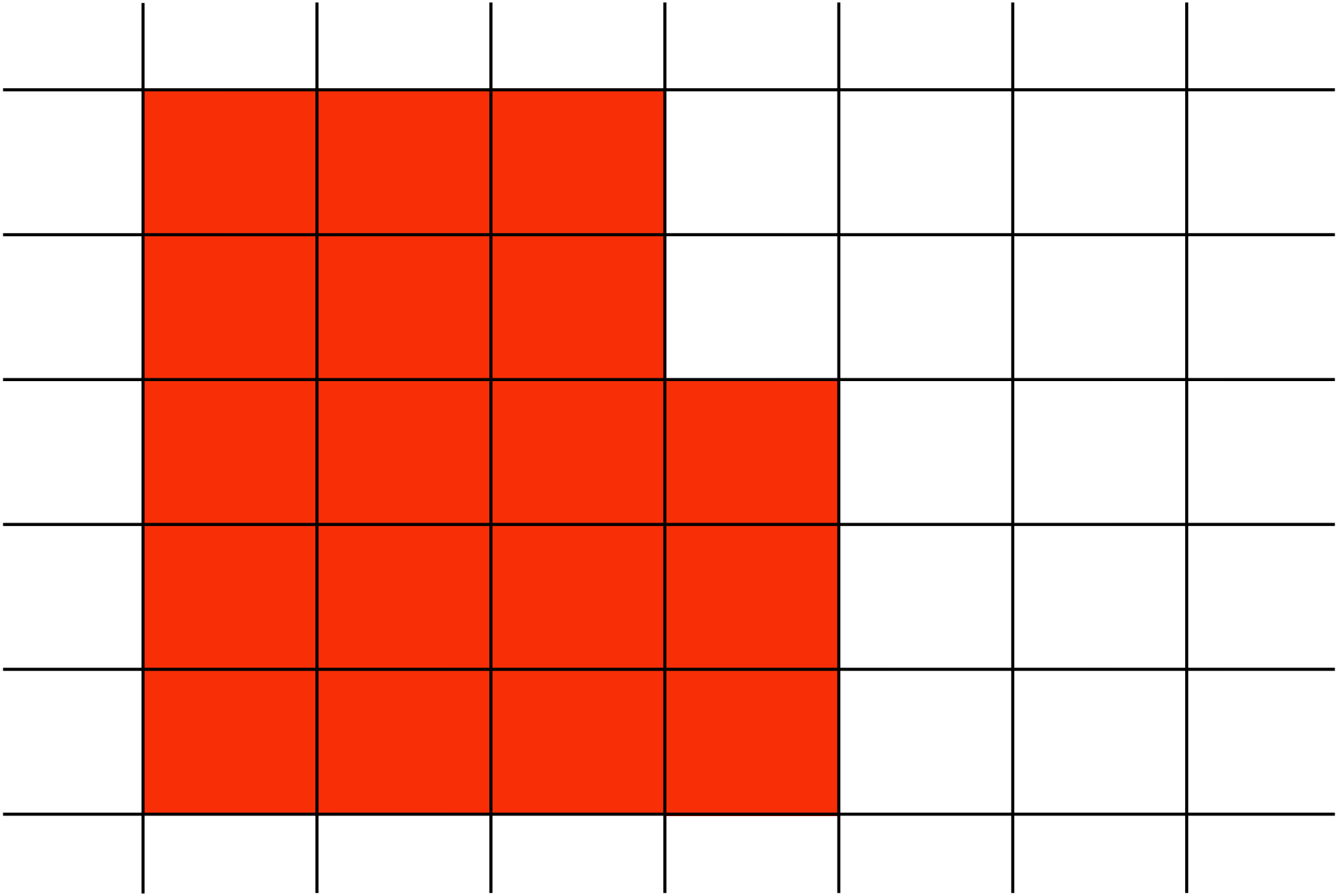
* Thinking of each LCD pixel as emitting a square of uniform intensity light of a single color is a bit of an approximation to how real displays work, but it will do for now.

So if we send the display this:



We see this when we look at the screen

(assuming a screen pixel emits a square of perfectly uniform intensity of light)

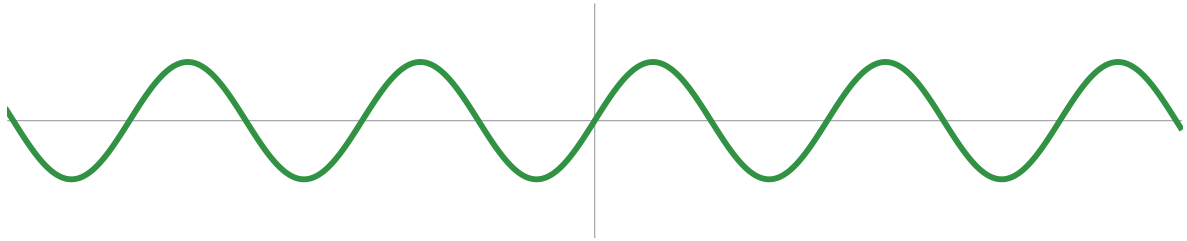


Recall: the real coverage signal was this

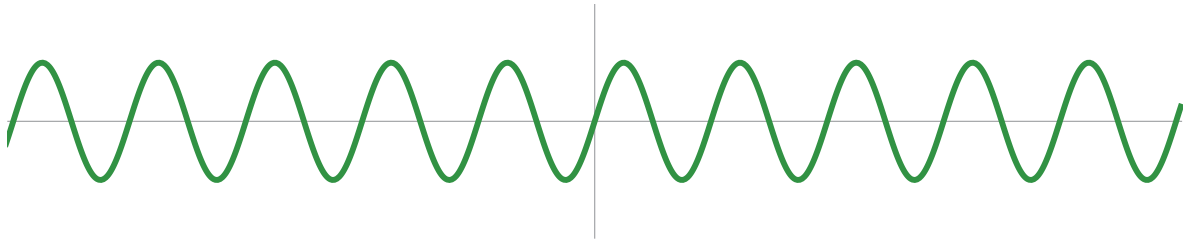
Aliasing

Representing sound as a superposition of frequencies

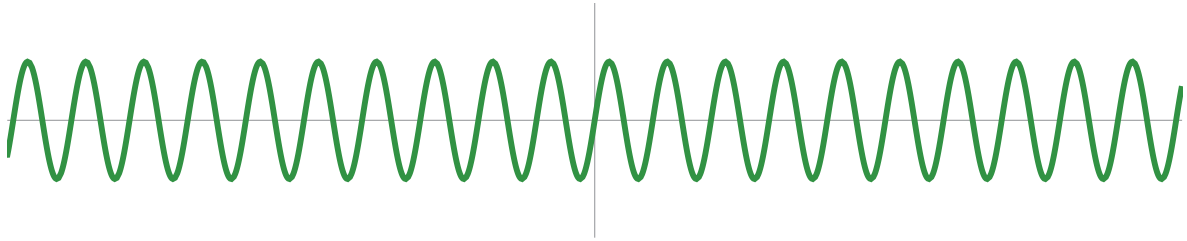
$f_1(x) = sin(\pi x)$



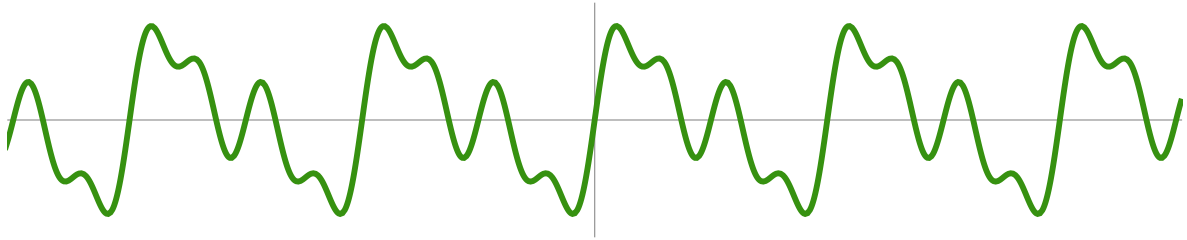
$f_2(x) = sin(2\pi x)$



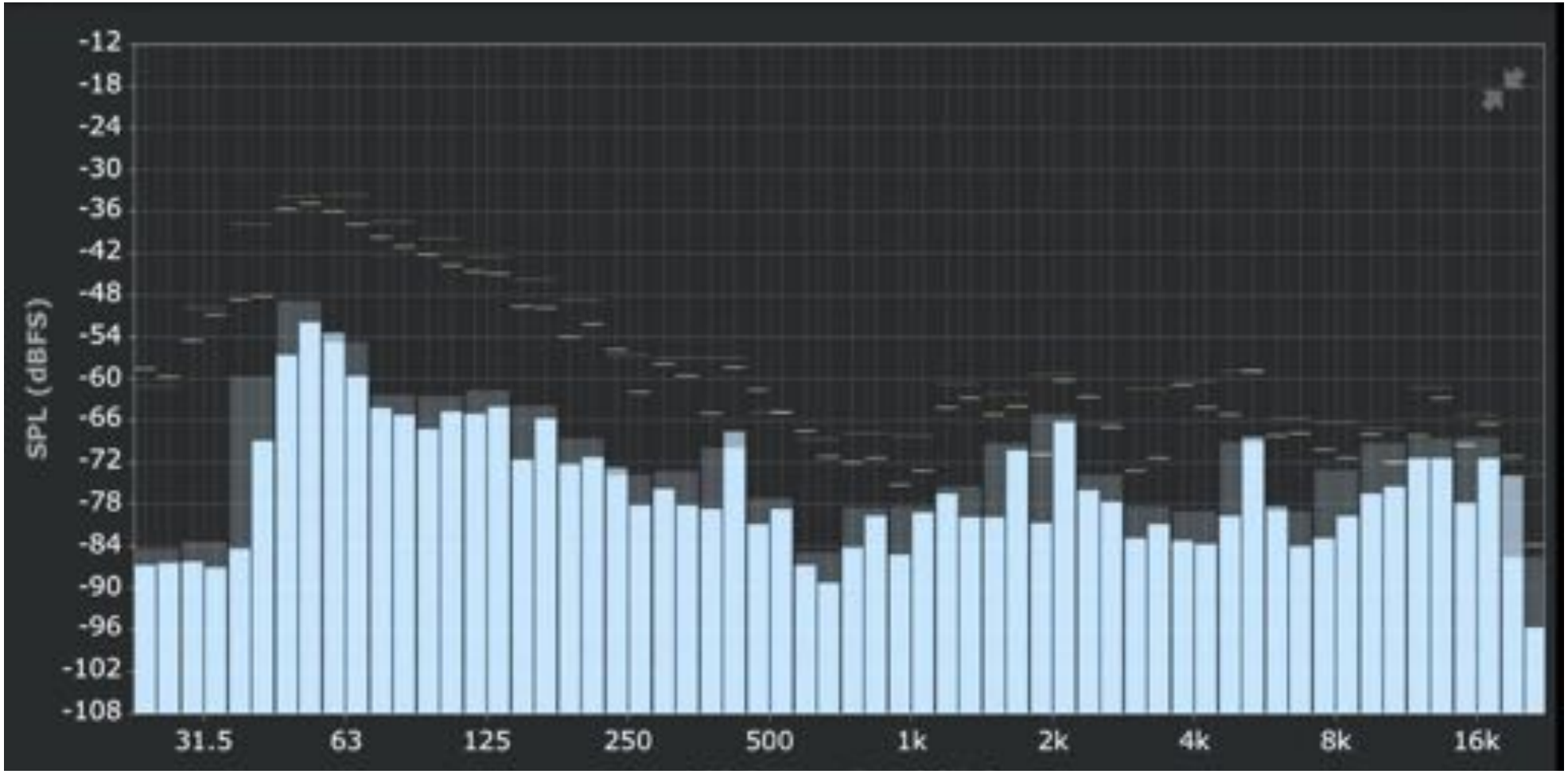
$f_4(x) = sin(4\pi x)$



$f(x) = f_1(x) + 0.75\, f_2(x) + 0.5\, f_4(x)$



Audio spectrum analyzer: representing sound as a sum of its constituent frequencies



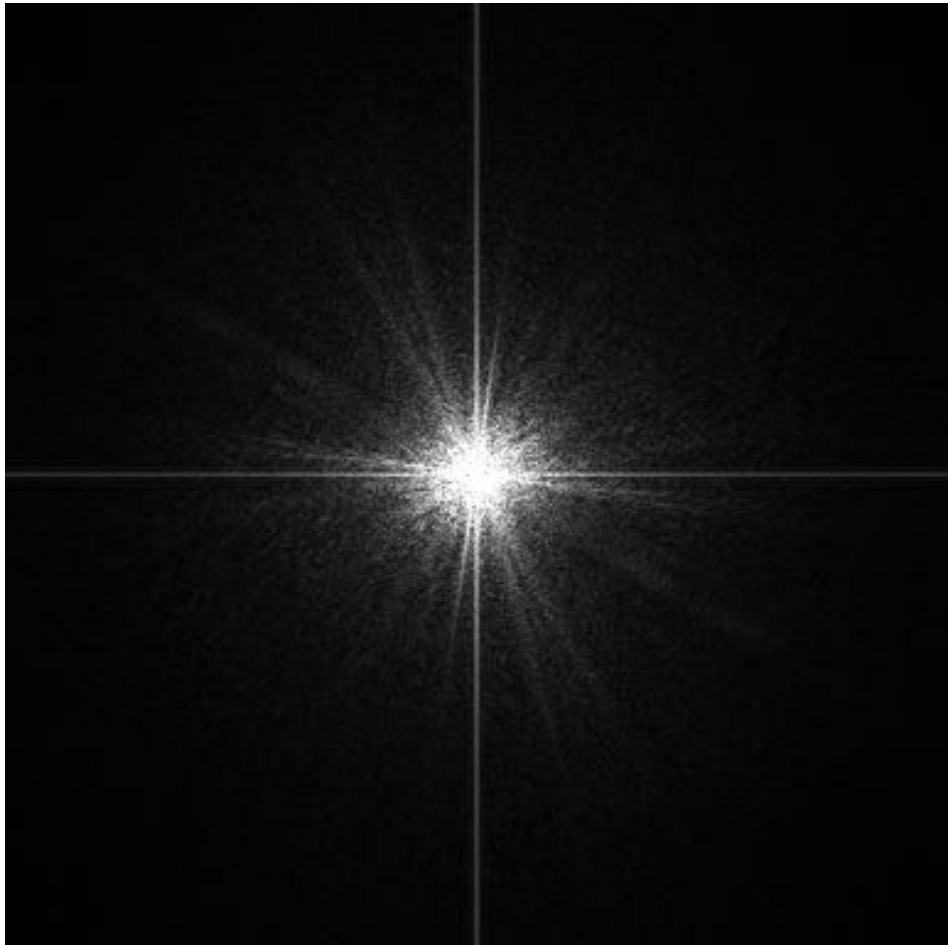
Intensity of low-frequencies (bass)

Intensity of high frequencies

Image credit: ONYX Apps

Visualizing the frequency content of images

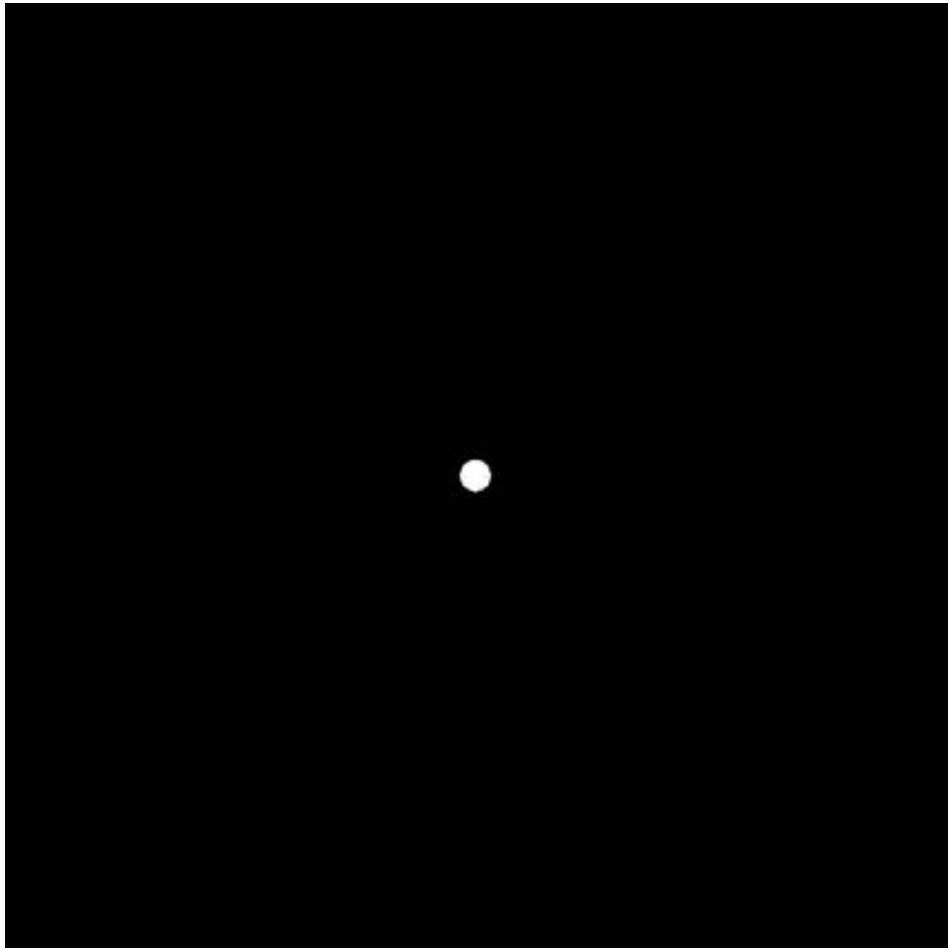
Spatial domain result



Spectrum

Low frequencies only (smooth gradients)

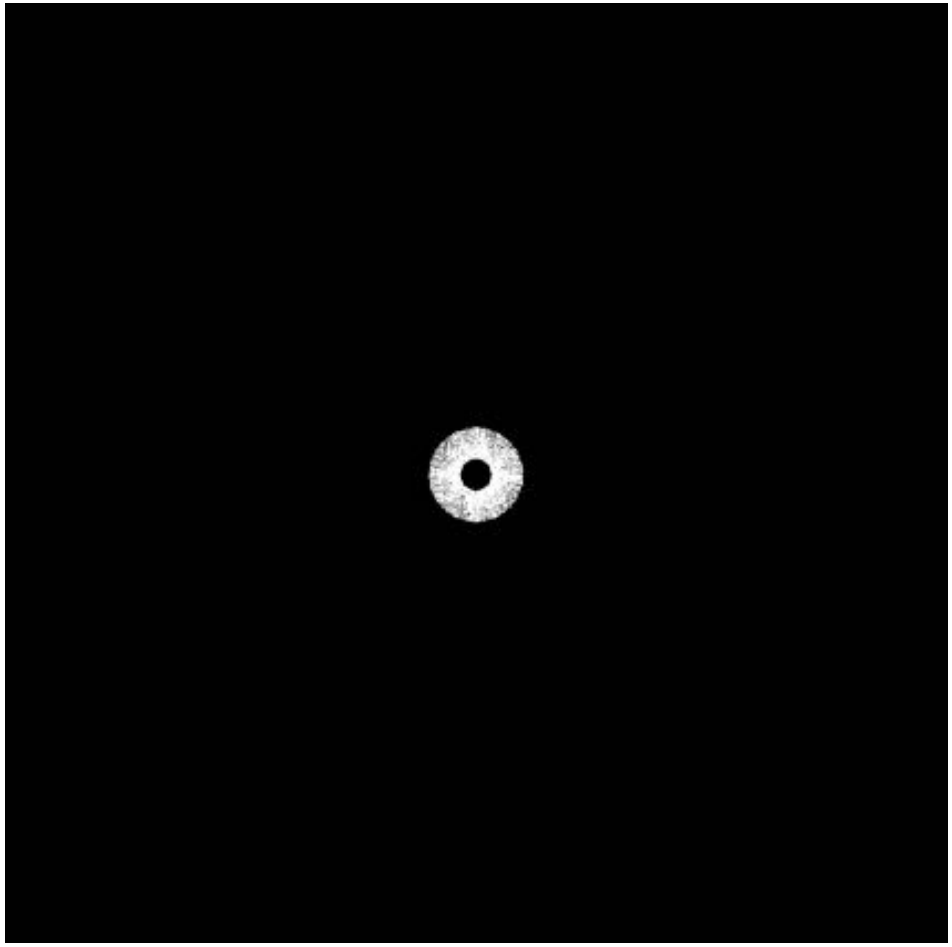
Spatial domain result



Spectrum (after low-pass filter)
All frequencies above cutoff have 0 magnitude

Mid-range frequencies

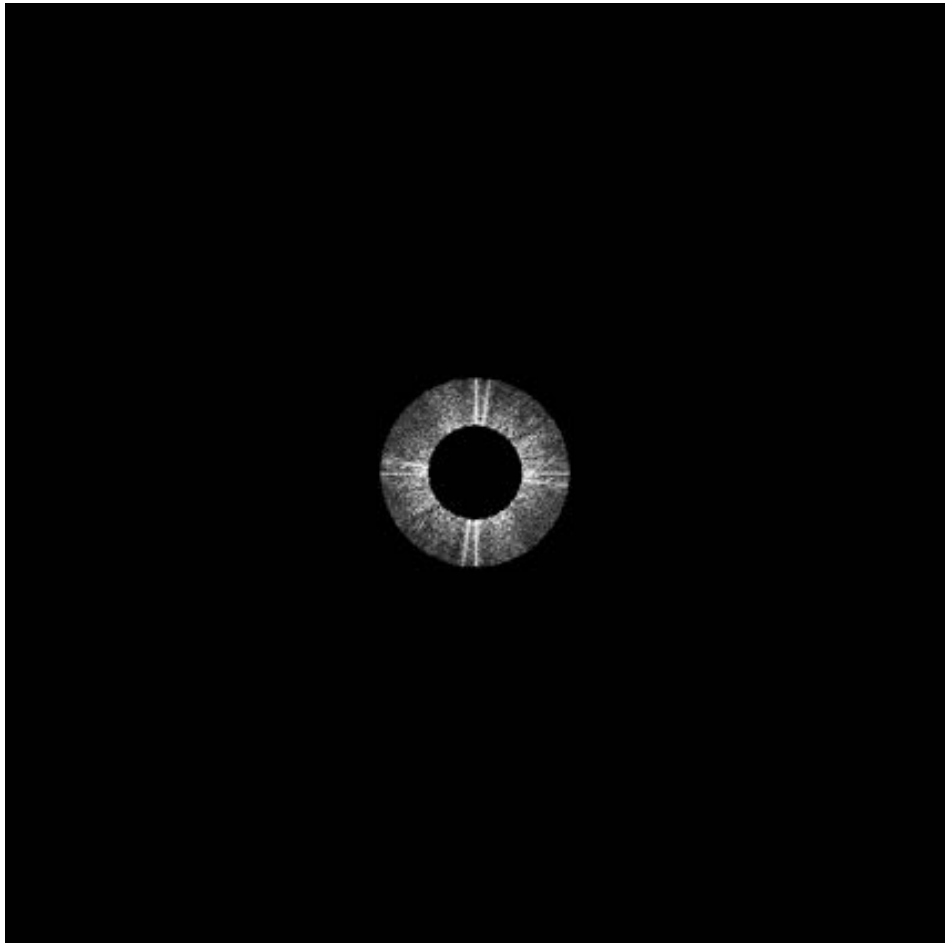
Spatial domain result



Spectrum (after band-pass filter)

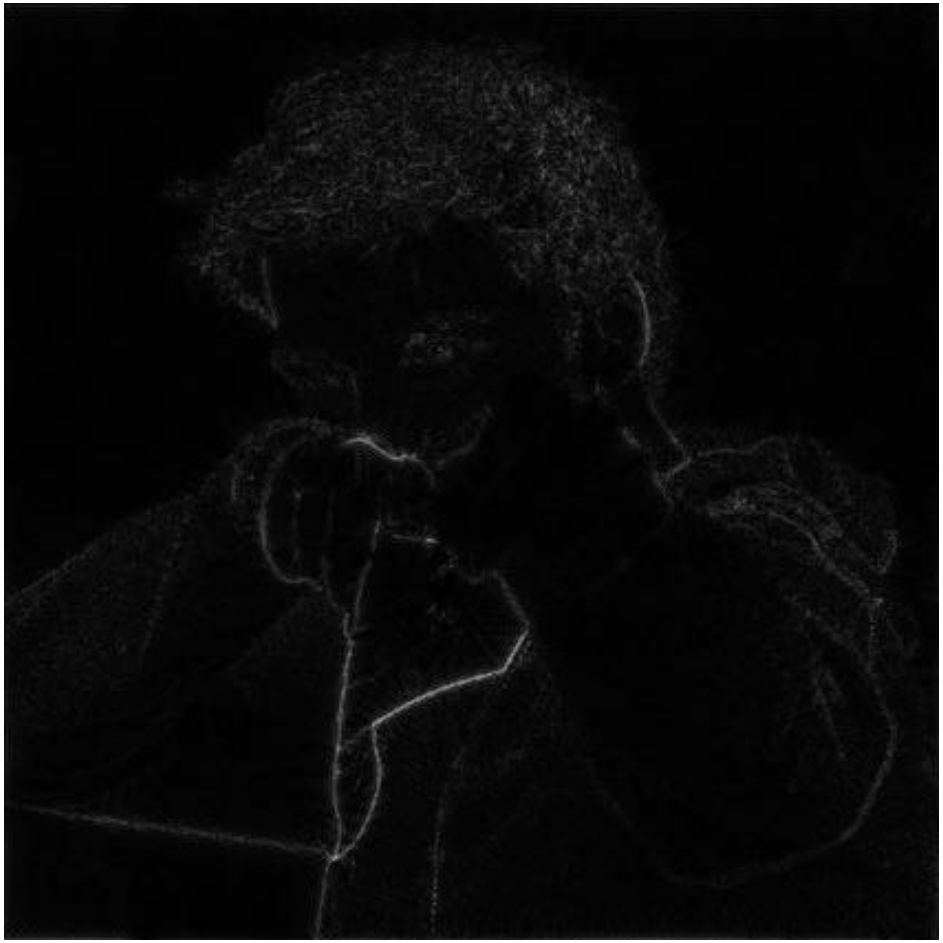
Mid-range frequencies

Spatial domain result

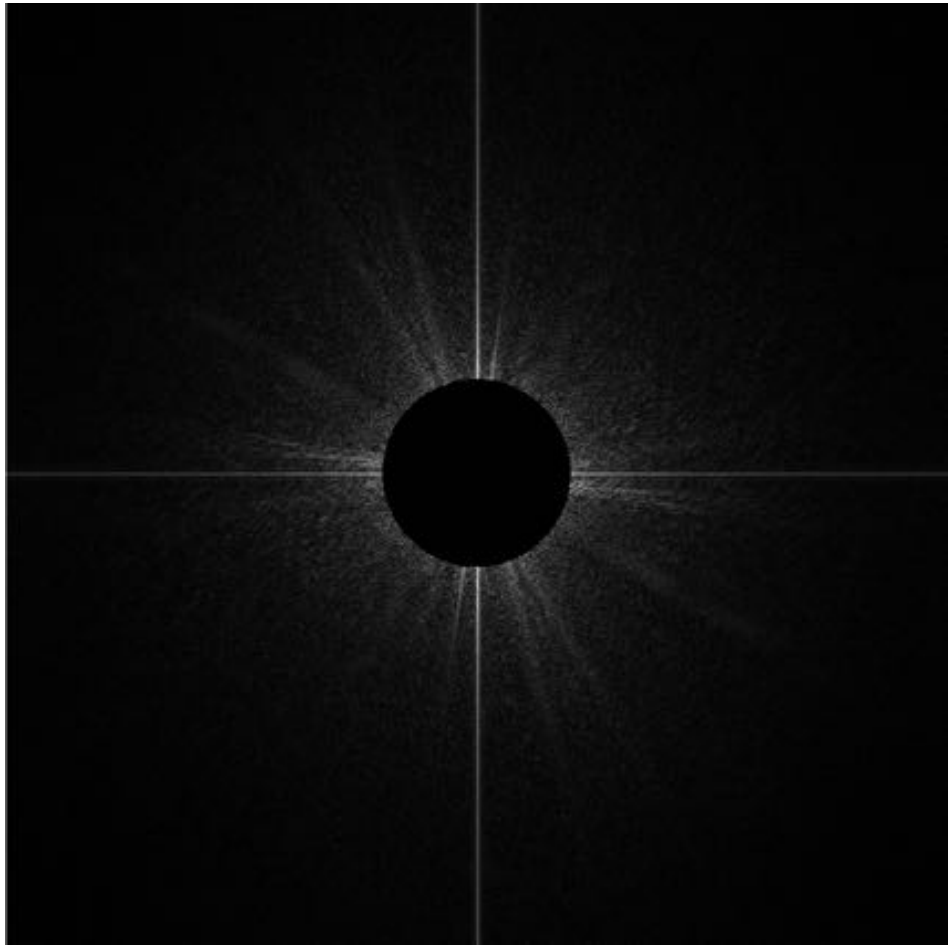


Spectrum (after band-pass filter)

High frequencies (edges)

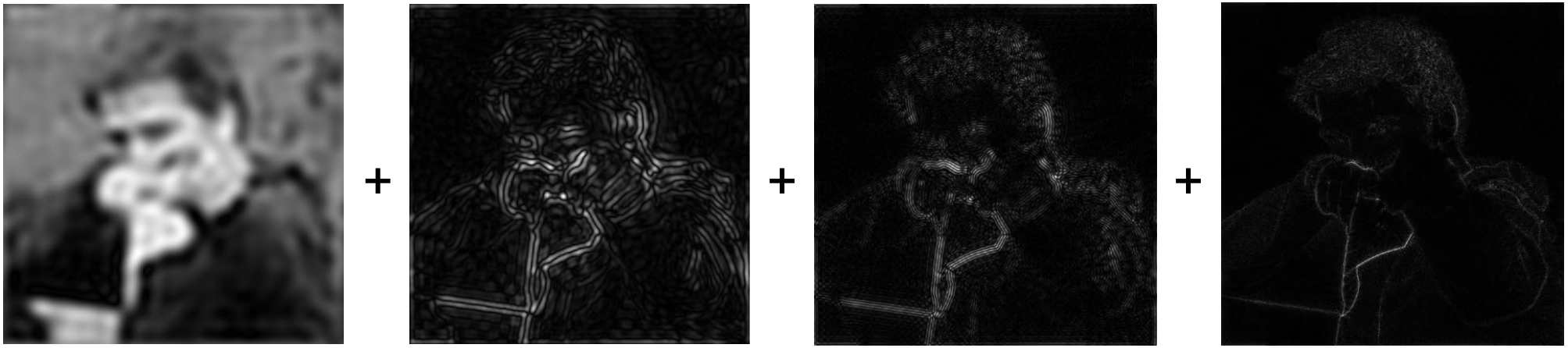


Spatial domain result
(strongest edges)

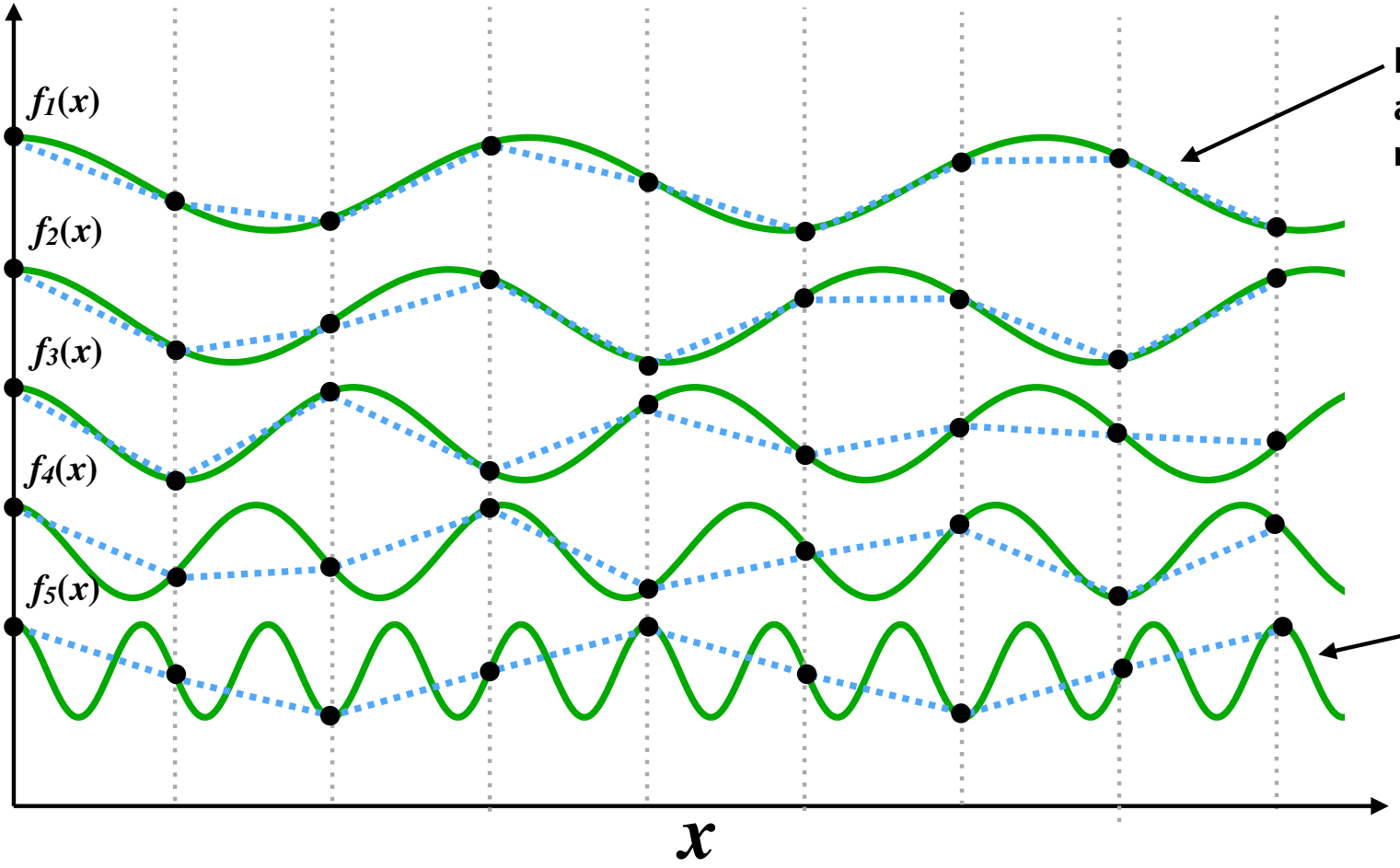


Spectrum (after high-pass filter)
All frequencies below threshold
have 0 magnitude

An image as a sum of its frequency components



1D example:
Undersampling high-frequency signals results in aliasing



Low-frequency signal: sampled adequately for accurate reconstruction

High-frequency signal is insufficiently sampled: reconstruction appears to be from a low frequency signal

"Aliasing": high frequencies in the original signal masquerade as low frequencies after reconstruction (due to undersampling)

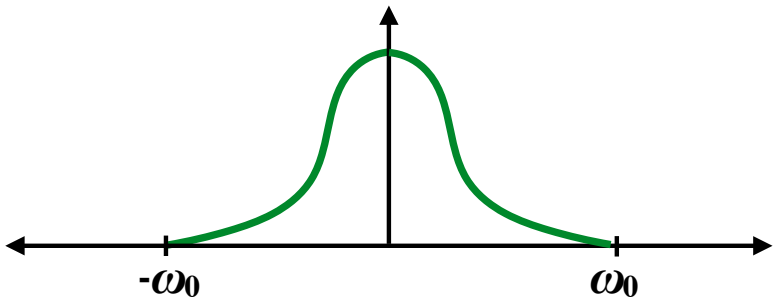
Temporal aliasing: wagon wheel effect

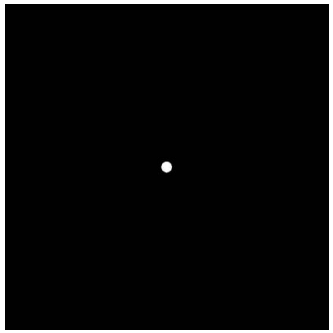
Camera's frame rate (temporal sampling rate) is too low for rapidly spinning wheel.

https://www.youtube.com/watch?v=VNftf5qLpiA

Nyquist-Shannon theorem

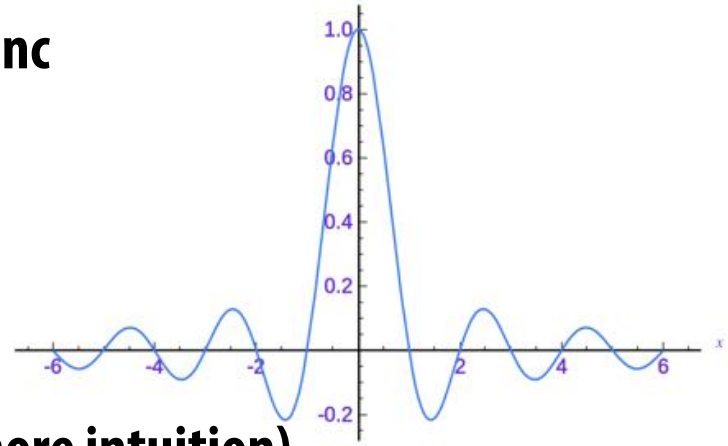
- Consider a band-limited signal: has no frequencies above ω_0
  - 1D: consider low-pass filtered audio signal
  - 2D: recall the blurred image example from a few slides ago





- The signal can be perfectly reconstructed if sampled with period $T = 1 / (2\omega_0)$
- And reconstruction is performed using a normalized sinc (ideal reconstruction filter with infinite extent)

$$sinc(x) = \frac{sin(\pi x))}{\pi x}$$

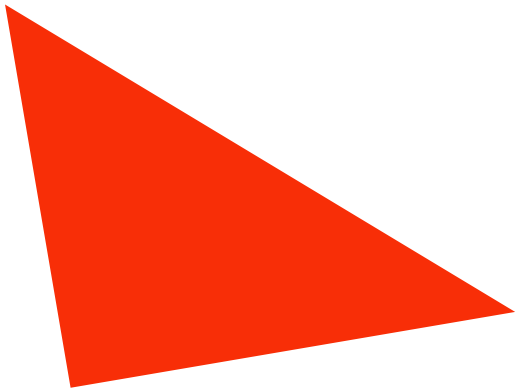


(See detailed explanation in suggested readings for more intuition)

Challenges of sampling-based approaches in graphics

- Our signals are not always band-limited in computer graphics. Why?

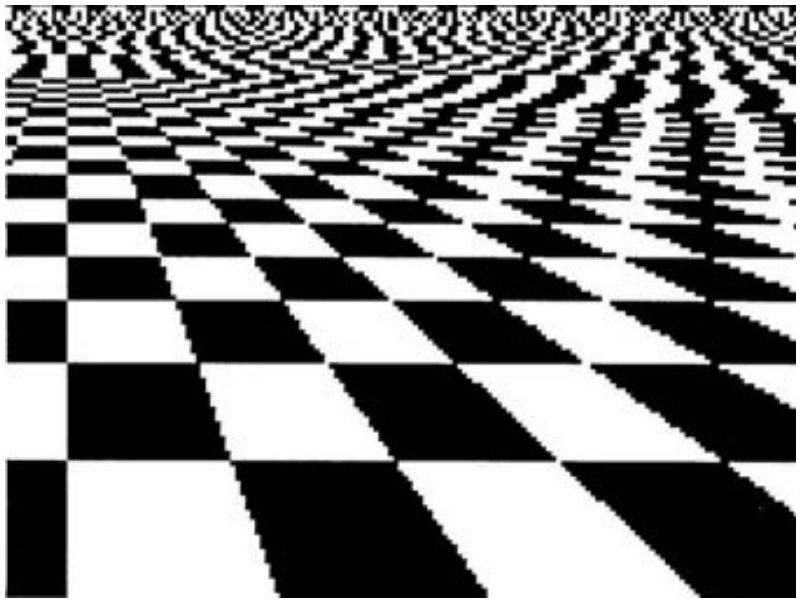
Hint:



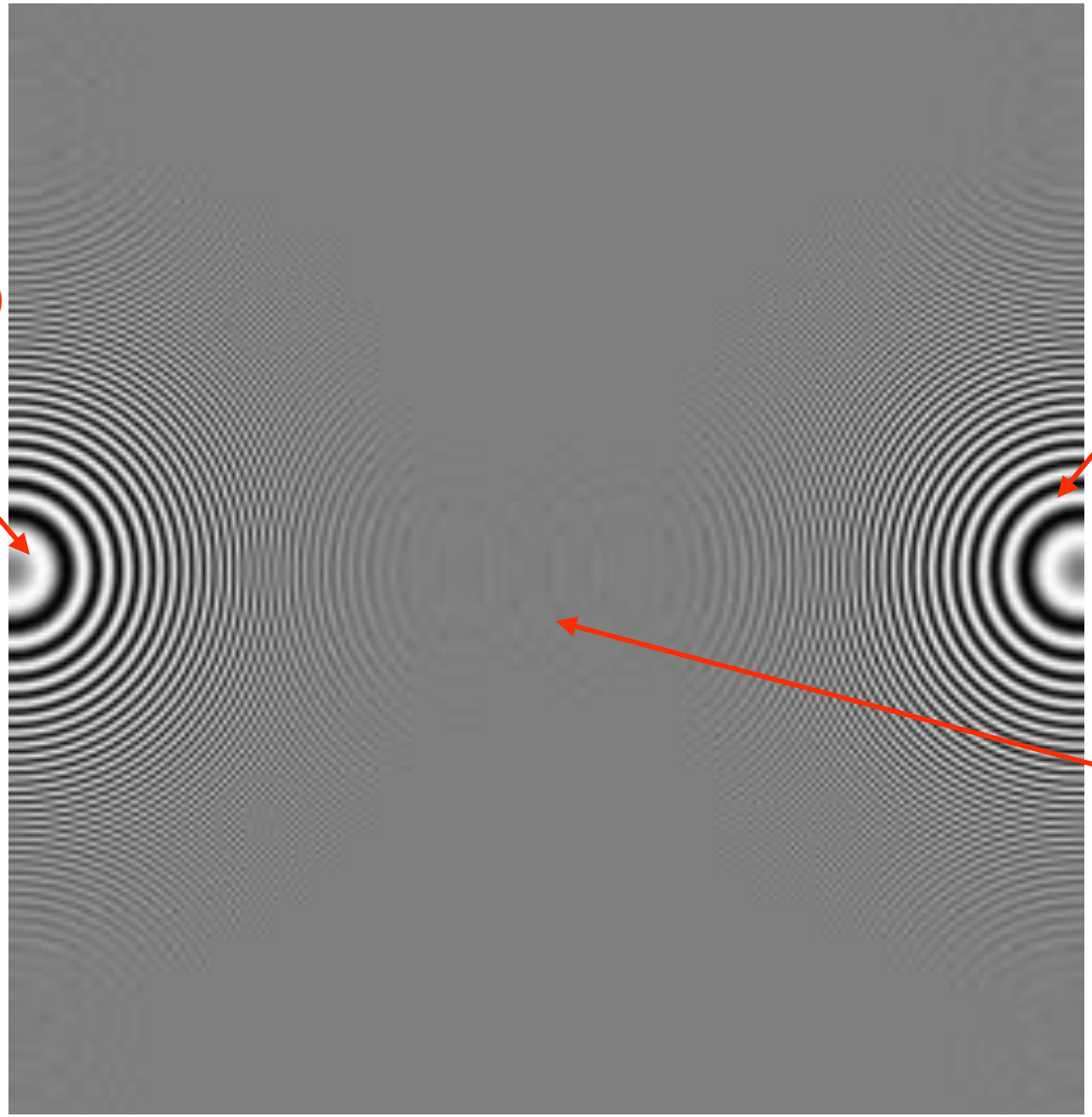
- Also, infinite extent of "ideal" reconstruction filter (sinc) is impractical for performant implementations. Why?

Aliasing artifacts in images

- **Undersampling high-frequency signals and the use of non-ideal resampling filters yields image artifacts**
  - **"Jaggies" in a single image**
  - **"Roping" or "shimmering" of images when animated**
  - **Moiré patterns in high-frequency areas of images**



Sampling a zone plate: $\sin(x^2 + y^2)$



Rings in center-left: Actual signal (low frequency oscillation)

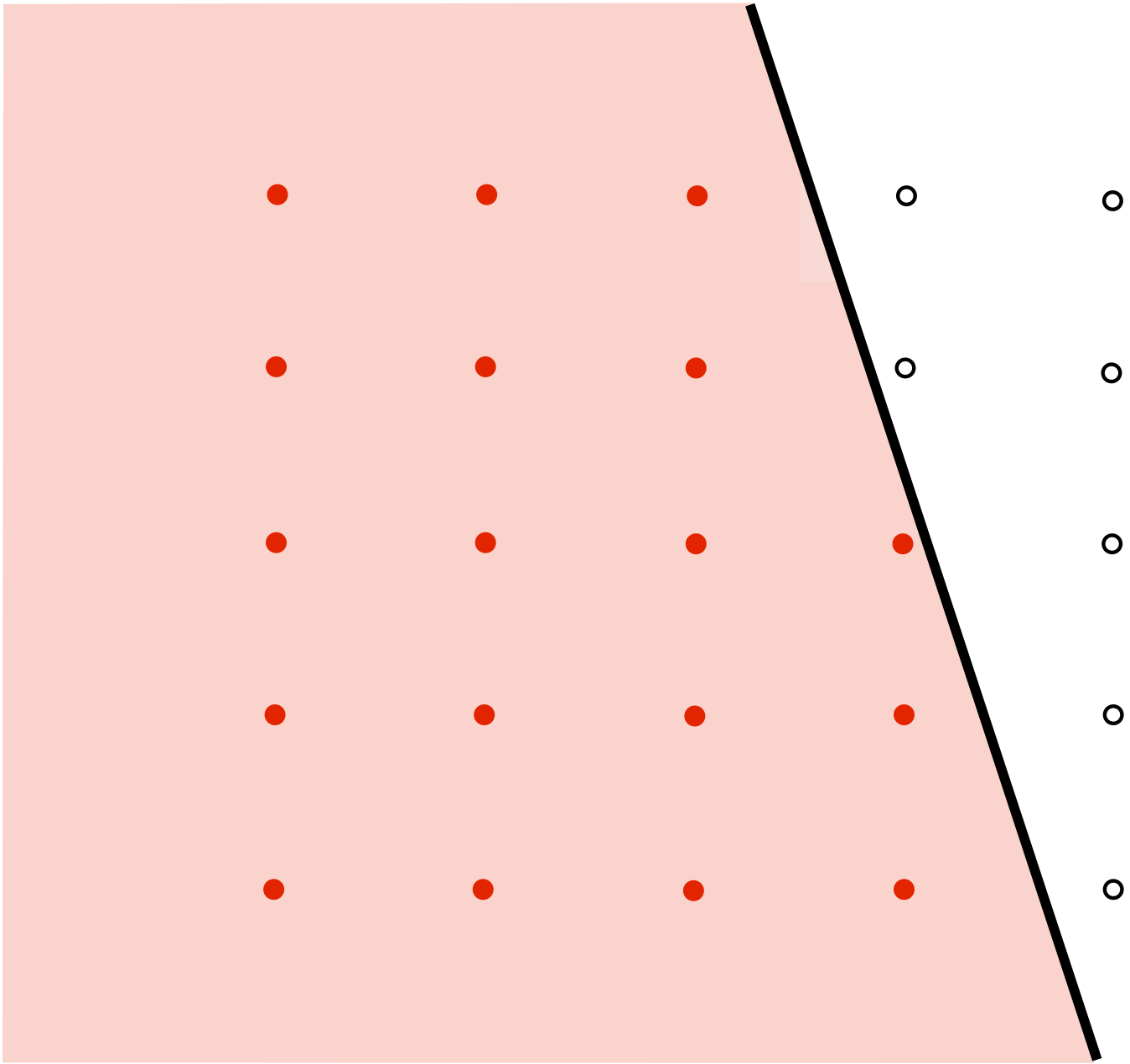
(0,0)

Rings on right: aliasing from undersampling high frequency oscillation and then resampling back to Keynote slide resolution

Middle: (interaction between actual signal and aliased resconstruction)

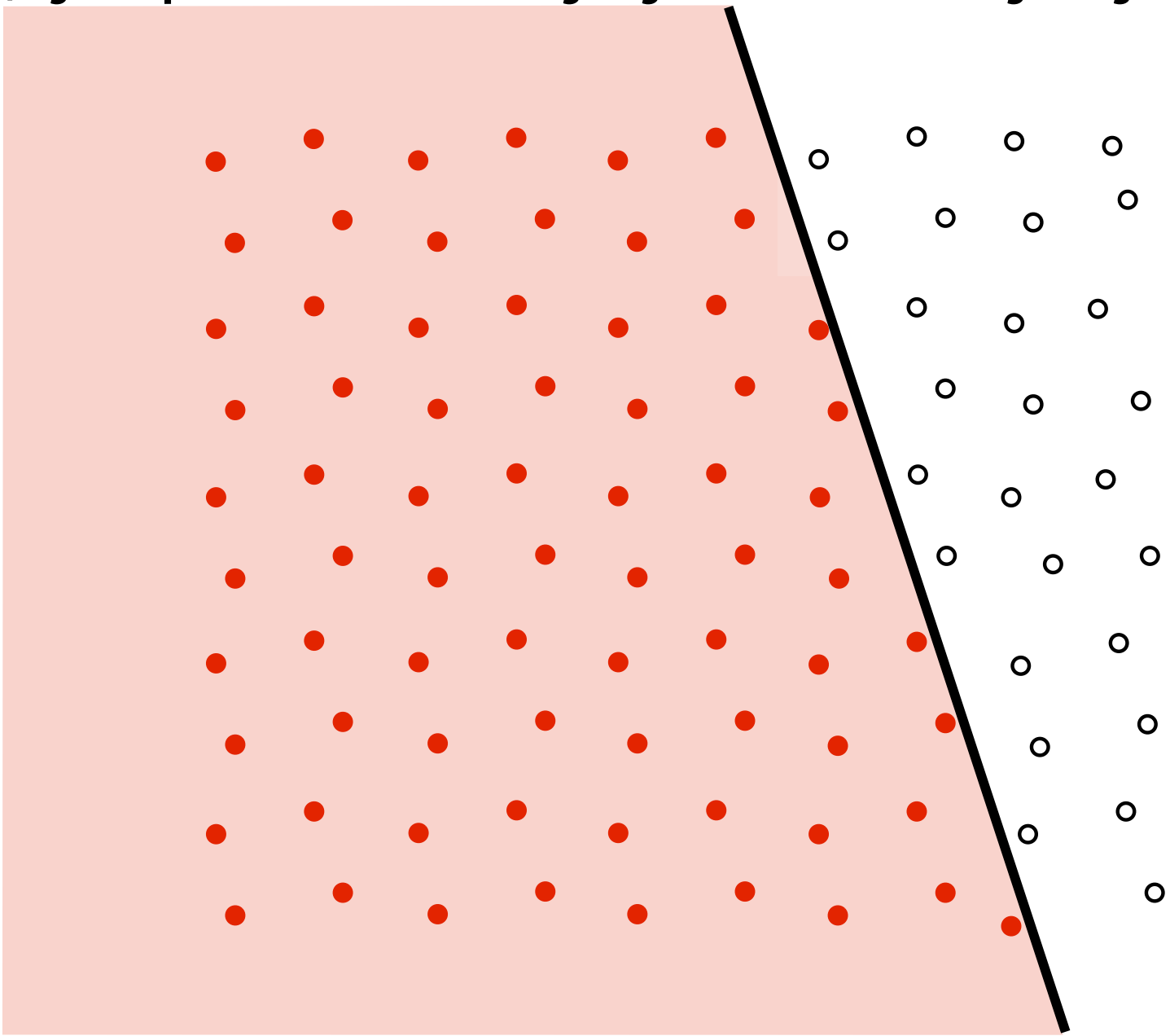
Figure credit: Pat Hanrahan and Bryce Summers

Initial coverage sampling rate (1 sample per pixel)



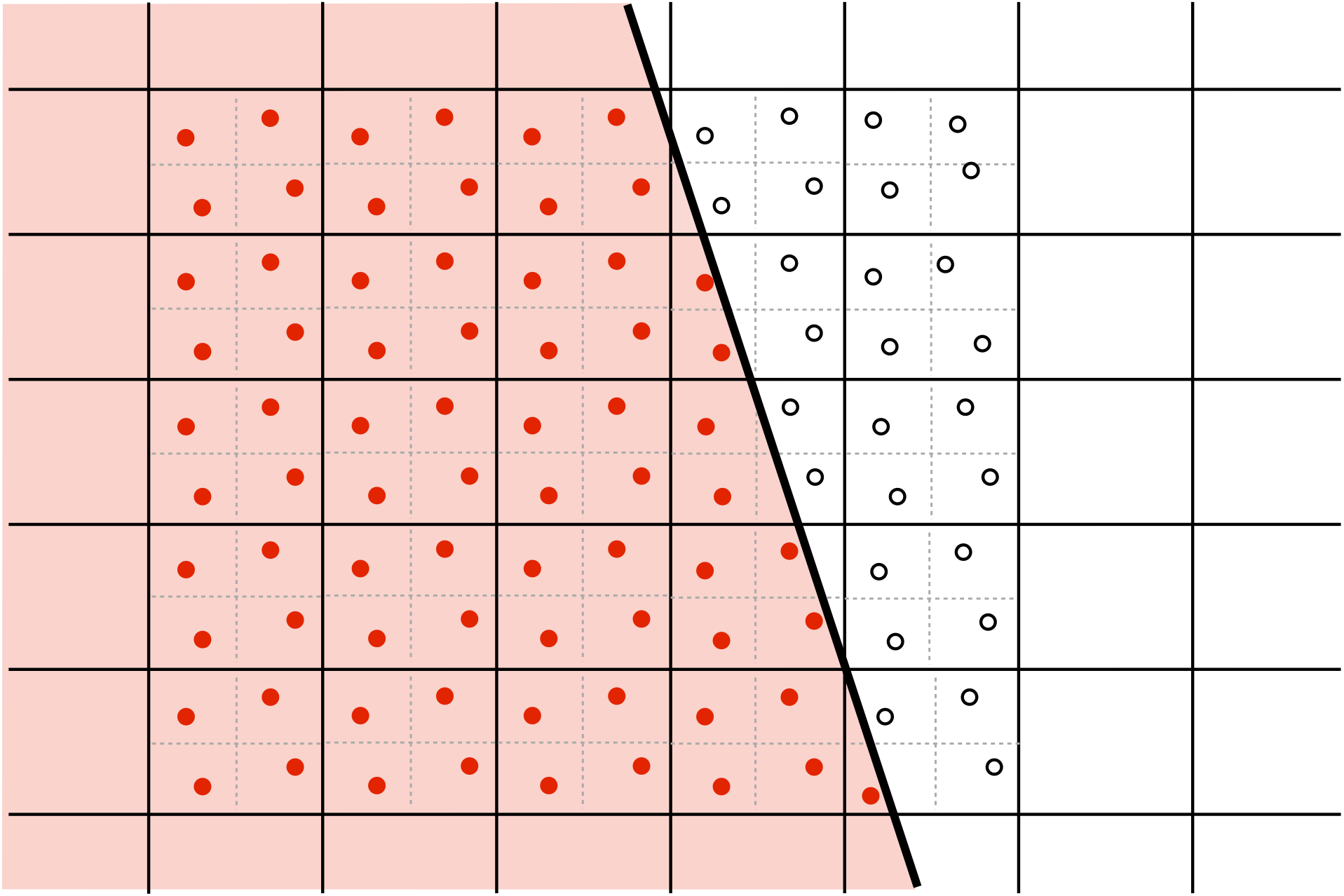
Increase density of sampling coverage signal

(high frequencies exist in coverage signal because of triangle edges)



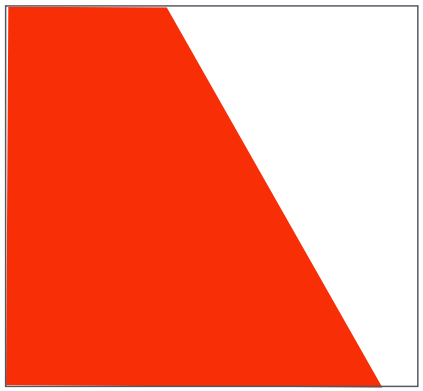
Supersampling

Example: stratified sampling using four samples per pixel

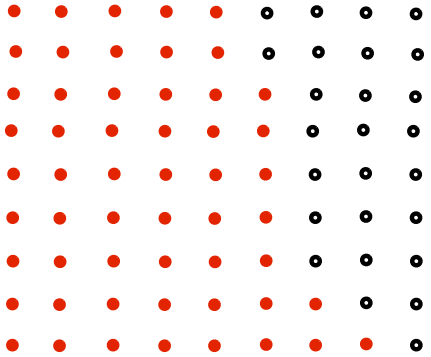


Resampling

Converting from one discrete sampled representation to another

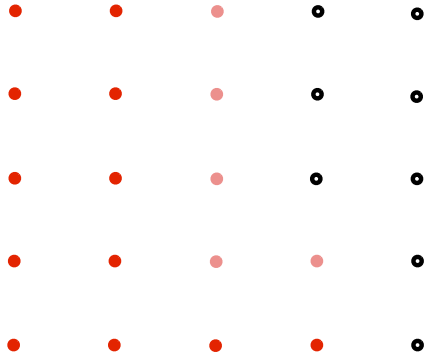


Original signal
(high frequency edge)



Dense sampling of
reconstructed signal

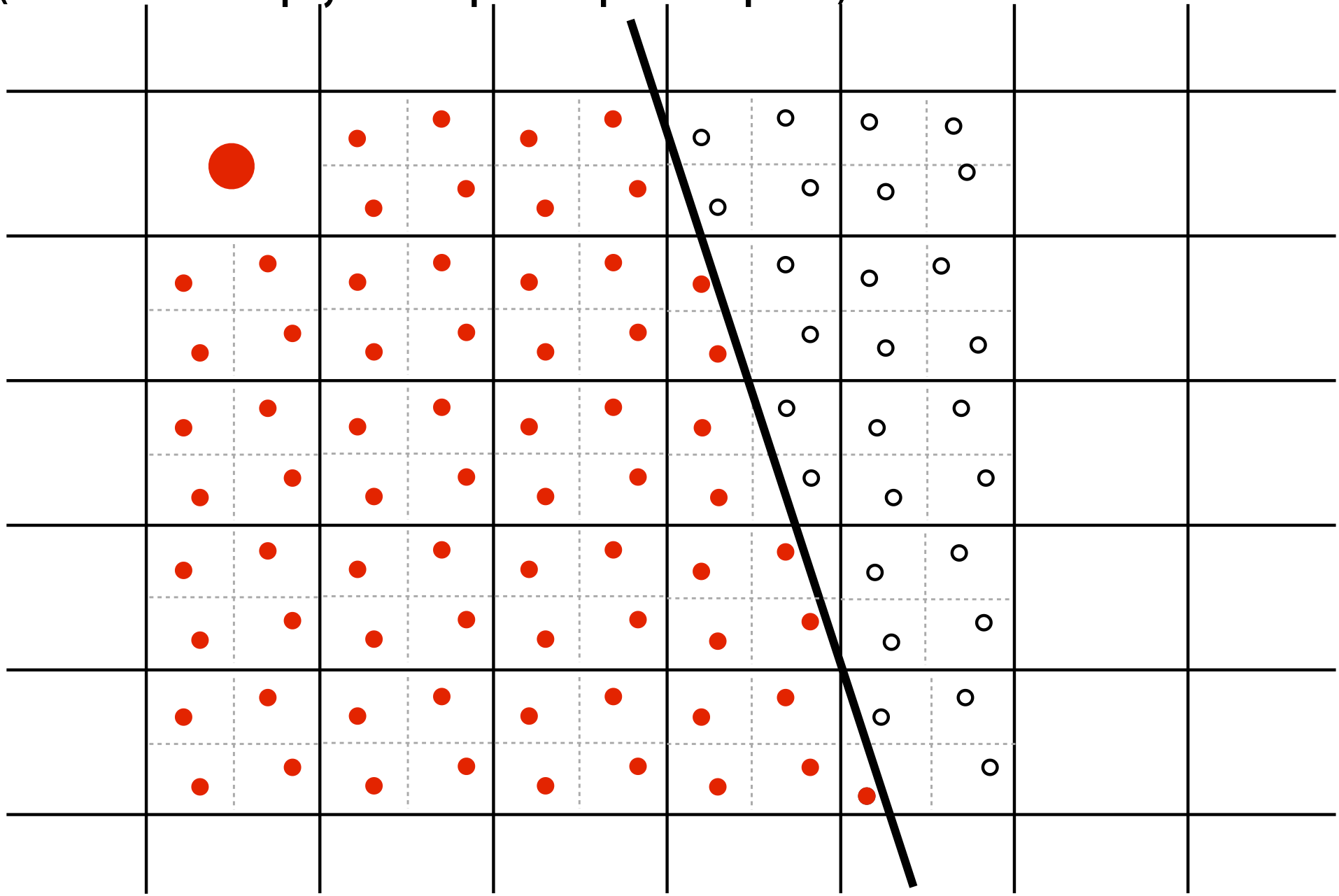
Reconstructed signal
(lacks high frequencies)



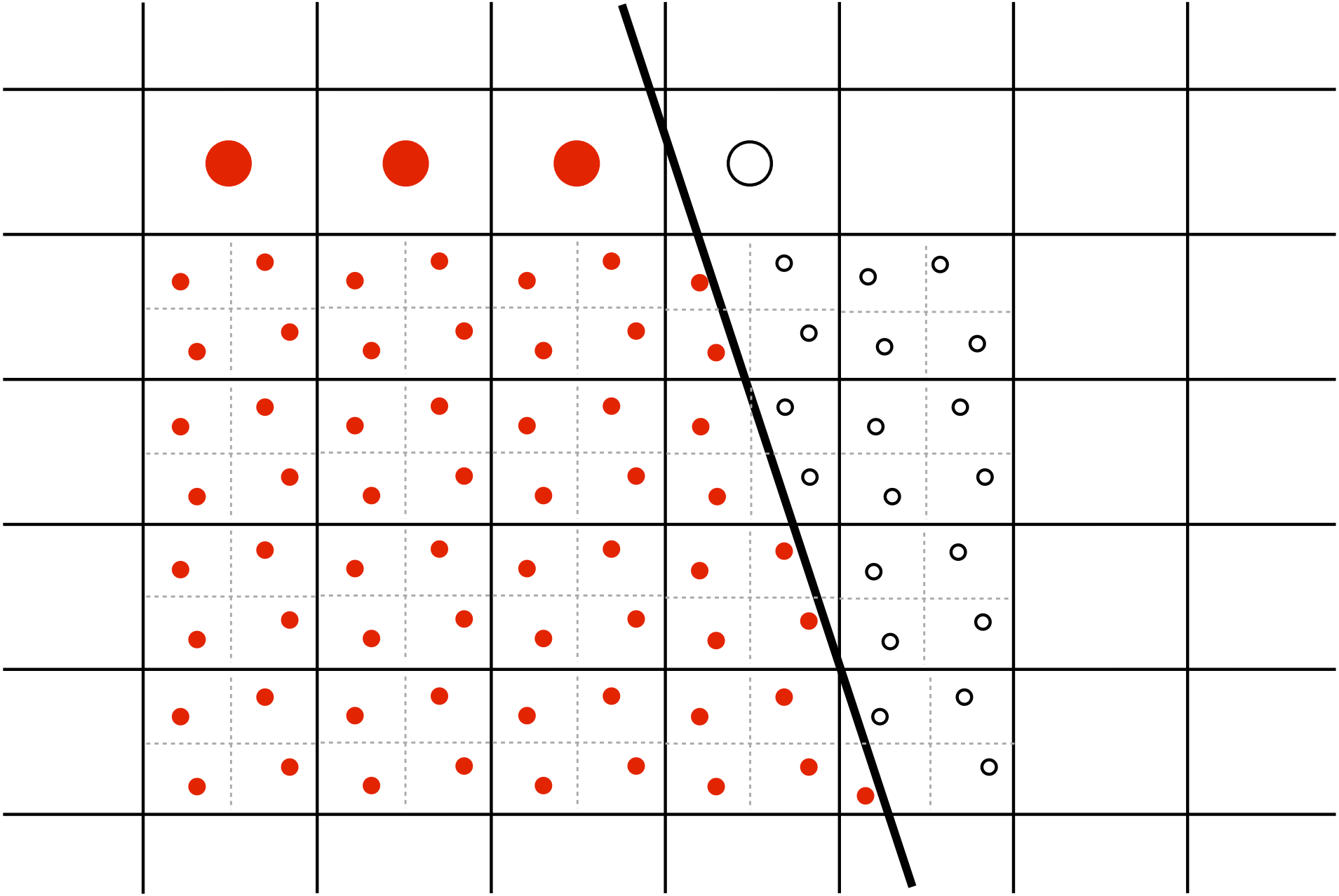
Coarsely sampled signal

Resample to display's pixel resolution

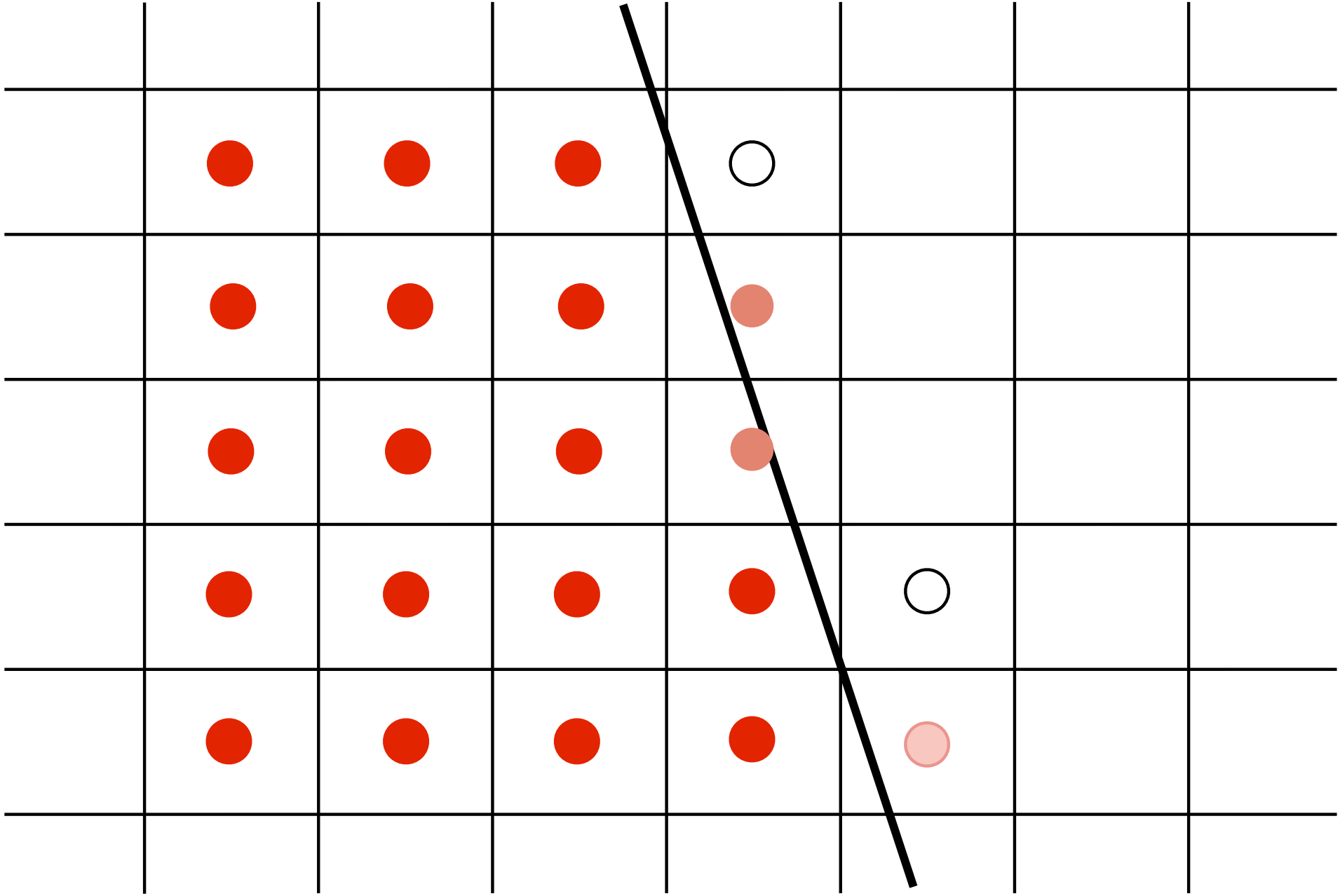
(Because a screen displays one sample value per screen pixel...)



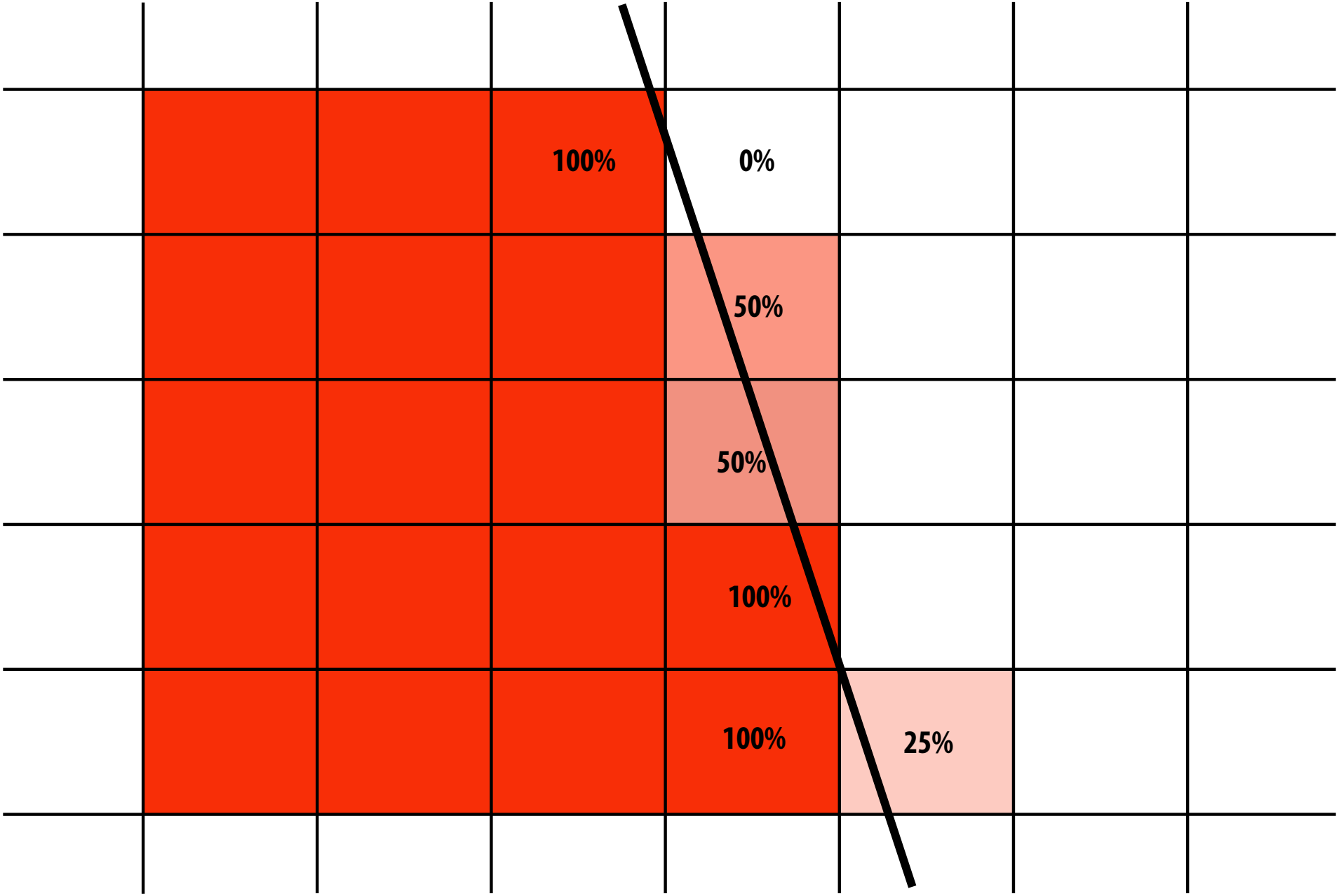
Resample to display's pixel rate (box filter)



Resample to display's pixel rate (box filter)



Displayed result (note anti-aliased edges)



Recall: the real coverage signal was this

Sampling coverage

- We want the light emitted from a display to be an accurate to match the ground truth signal: `coverage(x,y)`)

- Resampling a densely sampled signal (supersampled) integrates coverage values over the entire pixel region. The integrated result is sent to the display (and emitted by the pixel) so that the light emitted by the pixel is similar to what would be emitted in that screen region by an "infinite resolution display"

Sampling triangle coverage (evaluating coverage(x,y) for a triangle)

Point-in-triangle test

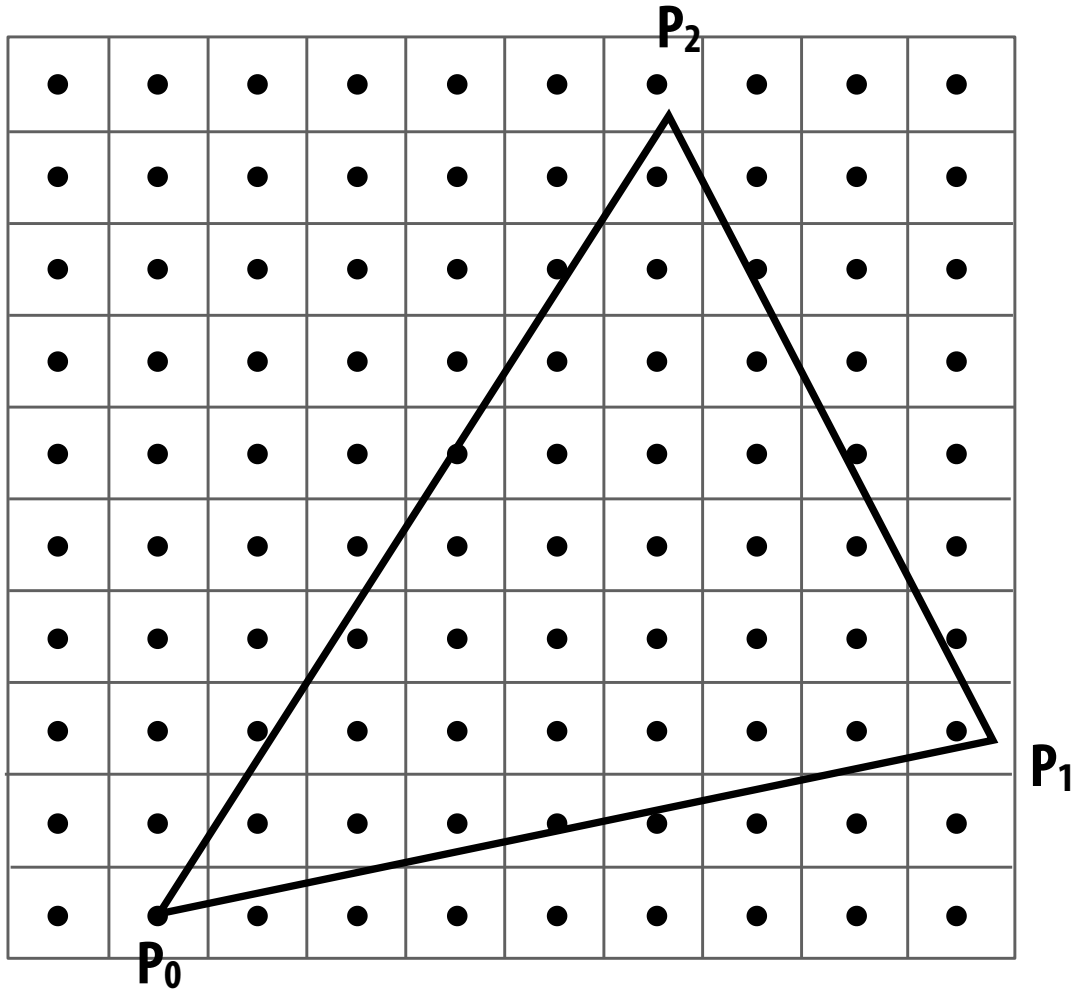
Compute triangle edge equations from projected positions of vertices

$P_i = (X_i, Y_i)$

$dX_i = X_{i+1} - X_i$
$dY_i = Y_{i+1} - Y_i$

$E_i(x, y) = (x - X_i)\, dY_i - (y - Y_i)\, dX_i$
$= A_i\, x + B_i\, y + C_i$

$E_i(x, y) = 0$: point on edge
> 0 : outside edge
< 0 : inside edge



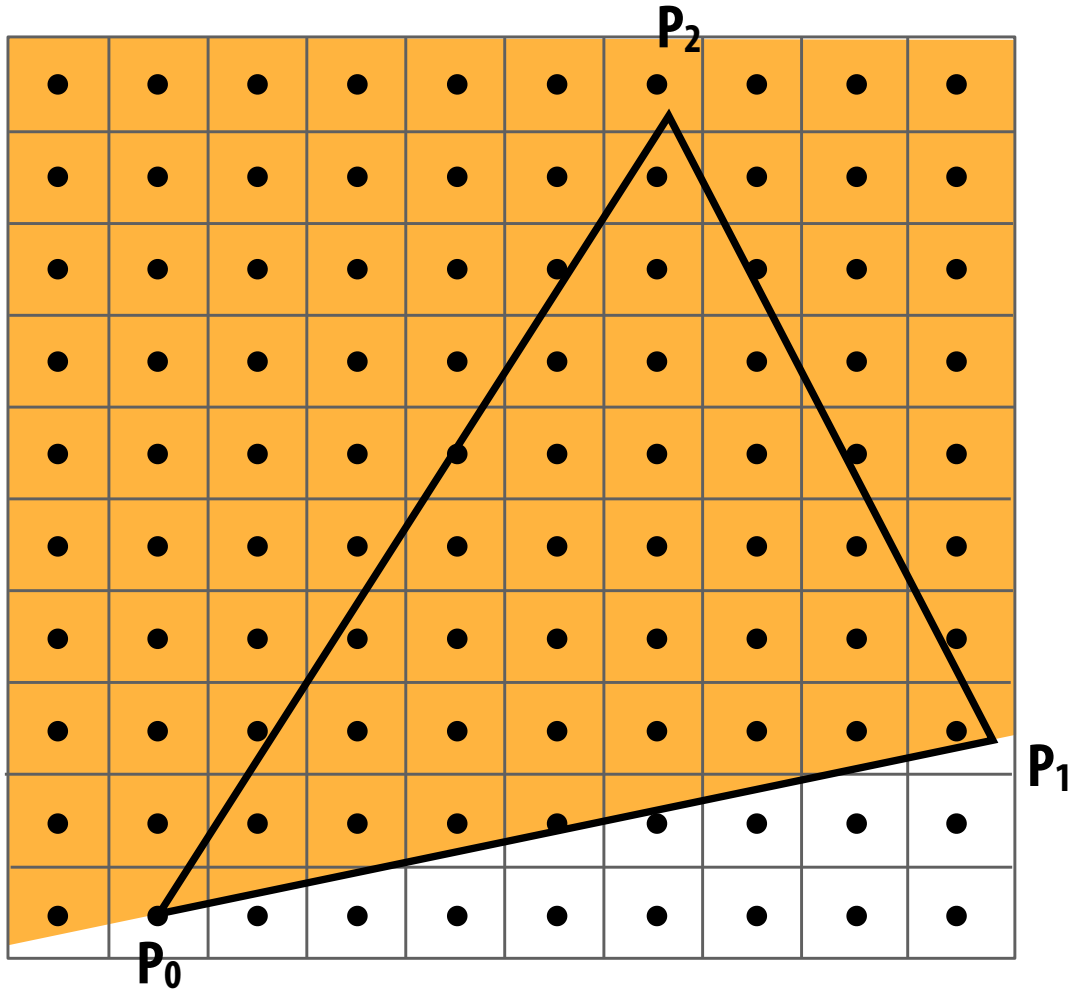
Point-in-triangle test

$P_i = (X_i, Y_i)$

$dX_i = X_{i+1} - X_i$
$dY_i = Y_{i+1} - Y_i$

$E_i(x, y) = (x - X_i)\, dY_i - (y - Y_i)\, dX_i$
$= A_i\, x + B_i\, y + C_i$

$E_i(x, y) = 0$: point on edge
> 0 : outside edge
< 0 : inside edge



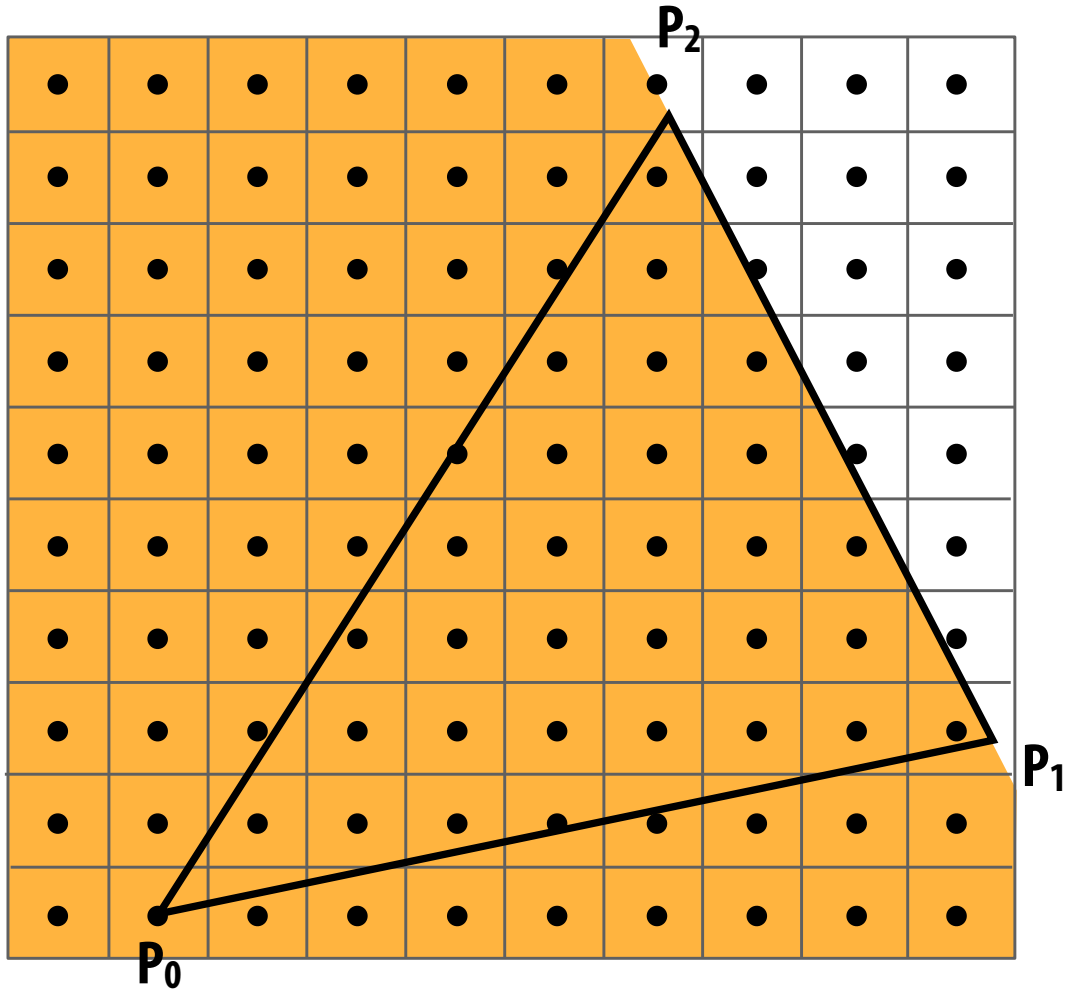
Point-in-triangle test

$P_i = (X_i, Y_i)$

$dX_i = X_{i+1} - X_i$
$dY_i = Y_{i+1} - Y_i$

$E_i(x, y) = (x - X_i)\, dY_i - (y - Y_i)\, dX_i$
$= A_i\, x + B_i\, y + C_i$

$E_i(x, y) = 0$: point on edge
> 0 : outside edge
< 0 : inside edge



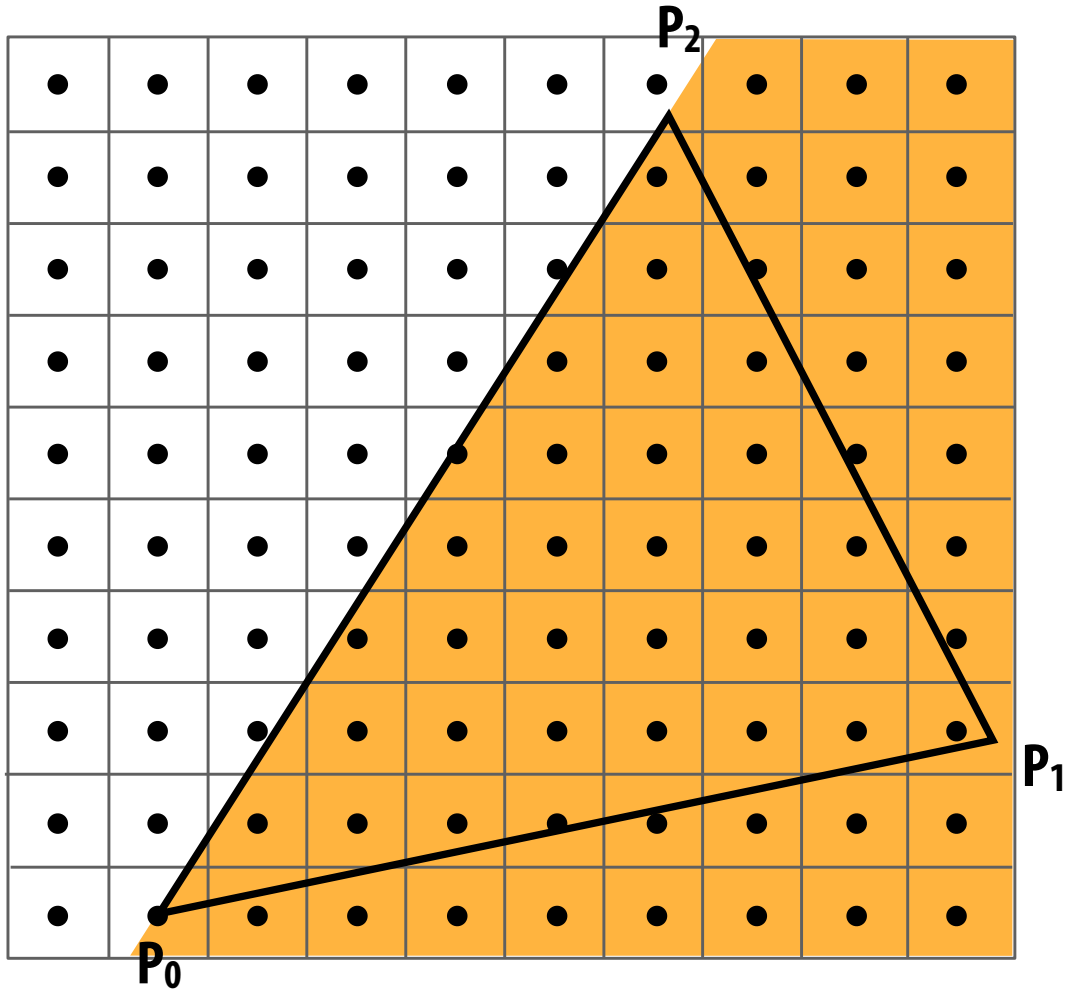
Point-in-triangle test

$P_i = (X_i, Y_i)$

$dX_i = X_{i+1} - X_i$
$dY_i = Y_{i+1} - Y_i$

$E_i(x, y) = (x - X_i)\, dY_i - (y - Y_i)\, dX_i$
$= A_i\, x + B_i\, y + C_i$

$E_i(x, y) = 0$: point on edge
> 0 : outside edge
< 0 : inside edge

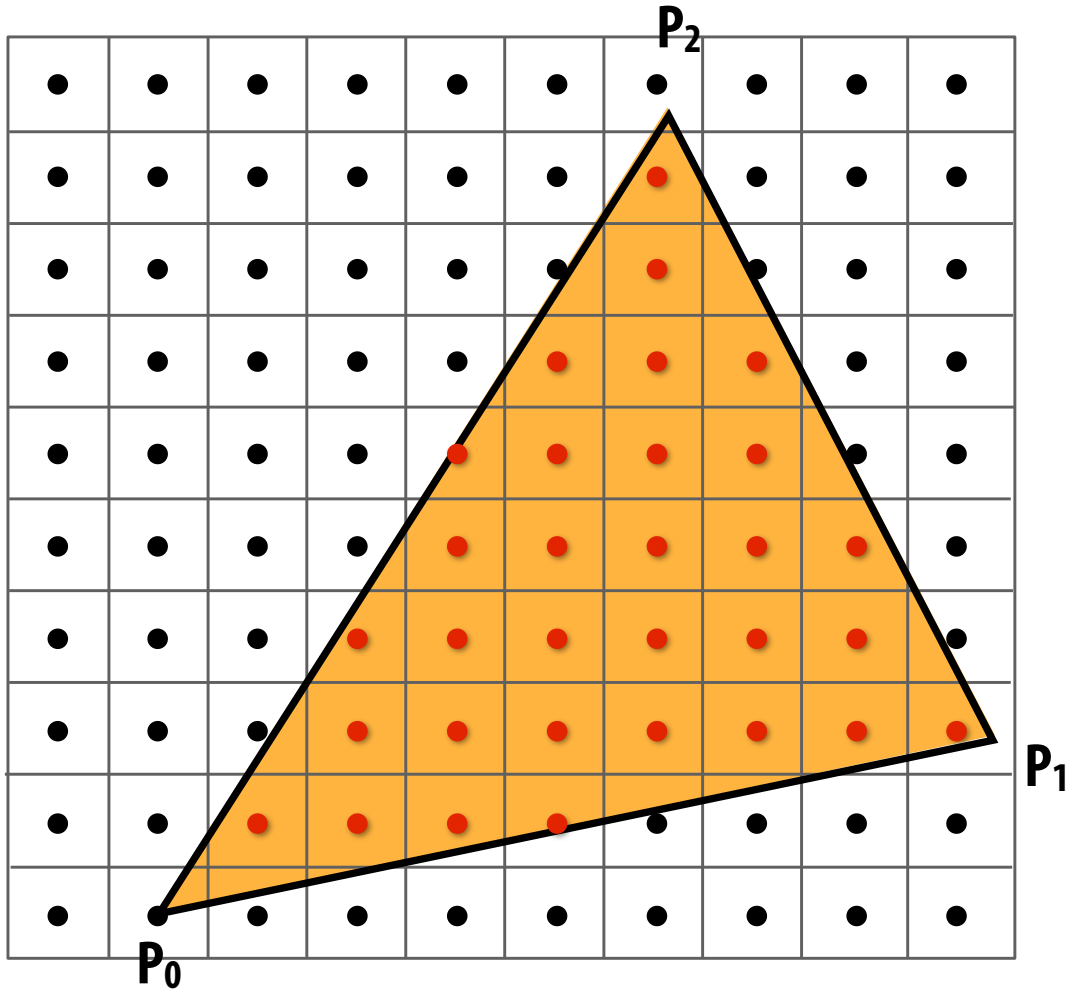


Point-in-triangle test

Sample point $s = (sx, sy)$ is inside the triangle if it is inside all three edges.

$$inside(sx, sy) =$$
$$E_0(sx, sy) < 0 \;\&\&$$
$$E_1(sx, sy) < 0 \;\&\&$$
$$E_2(sx, sy) < 0;$$

Note: actual implementation of $inside(sx,sy)$ involves ≤ checks based on the triangle coverage edge rules (see beginning of lecture)



Sample points inside triangle are highlighted red.

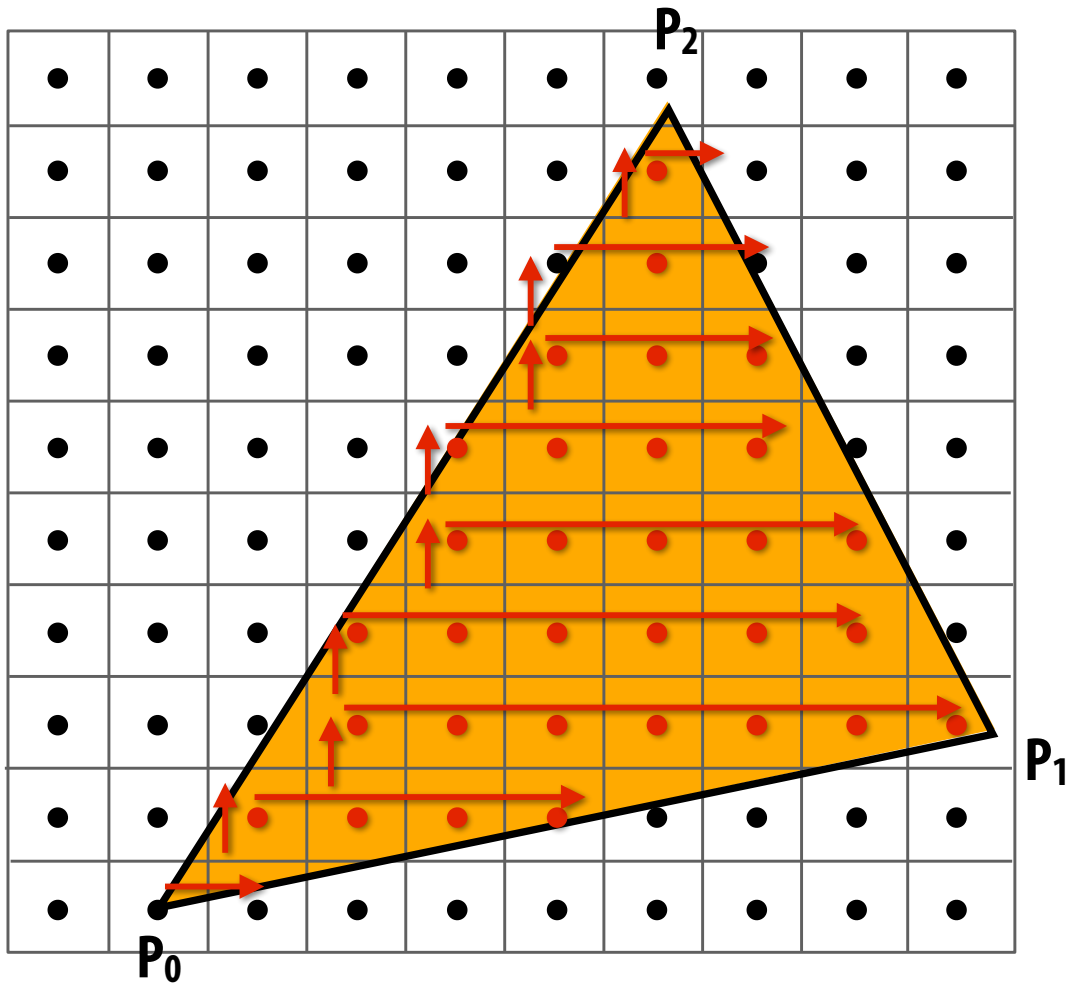
Incremental triangle traversal

$P_i = (X_i, Y_i)$

$dX_i = X_{i+1} - X_i$
$dY_i = Y_{i+1} - Y_i$

$E_i(x, y) = (x - X_i)\, dY_i - (y - Y_i)\, dX_i$
$= A_i\, x + B_i\, y + C_i$

$E_i(x, y) = 0$: point on edge
> 0 : outside edge
< 0 : inside edge



Efficient incremental update:

$dE_i(x+1,y) = E_i(x,y) + dY_i = E_i(x,y) + A_i$
$dE_i(x,y+1) = E_i(x,y) + dX_i = E_i(x,y) + B_i$

Incremental update saves computation:
Only one addition per edge, per sample test

Many traversal orders are possible: backtrack, zig-zag, Hilbert/Morton curves (locality maximizing)

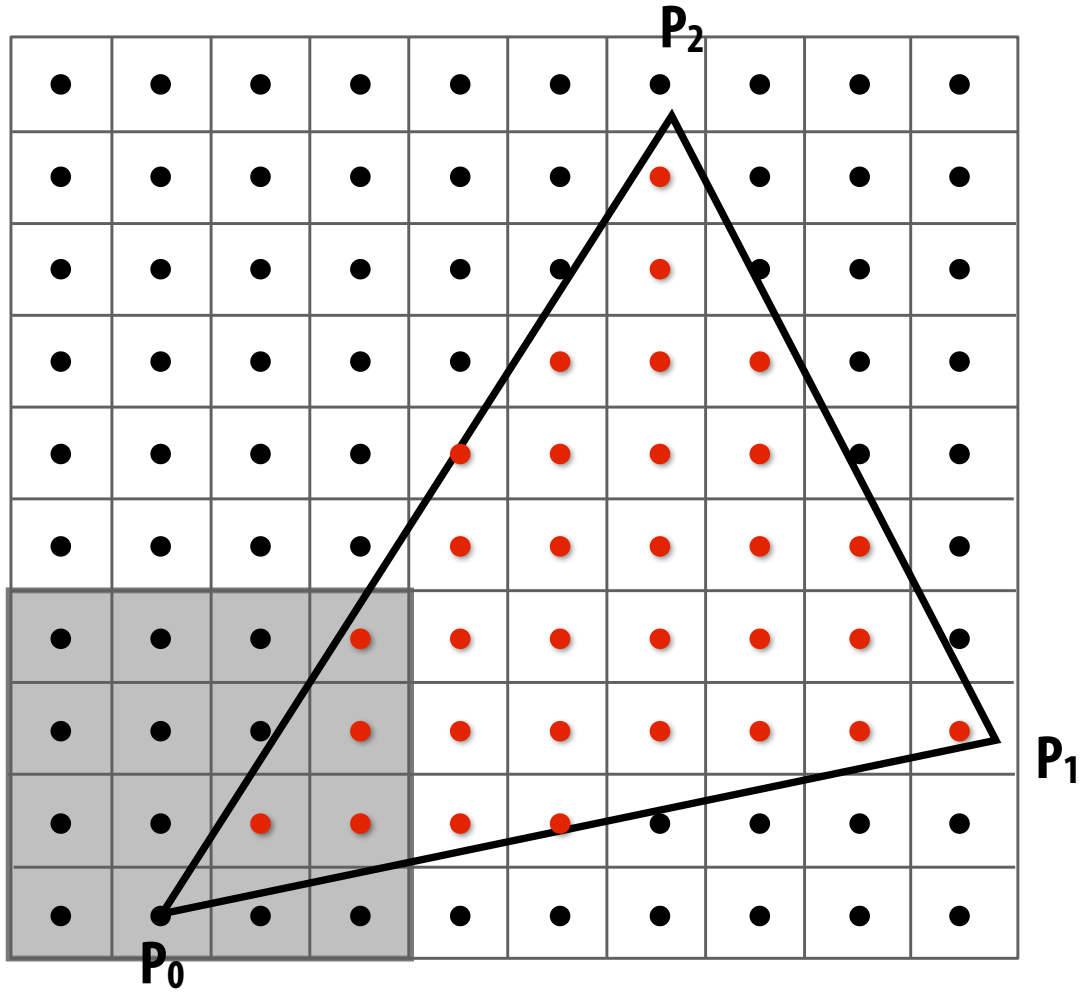
Modern approach: tiled triangle traversal

Traverse triangle in blocks

Test all samples in block against triangle in parallel

Advantages:

- Simplicity of wide parallel execution overcomes cost of extra point-in-triangle tests (most triangles cover many samples, especially when super-sampling coverage)
- Can skip sample testing work: entire block not in triangle ("early out"), entire block entirely within triangle ("early in")
- Additional advantaged related to accelerating occlusion computations (not discussed today)



All modern GPUs have special-purpose hardware for efficiently performing point-in-triangle tests

Summary

- **We formulated computing triangle-screen coverage as a sampling problem**
  - Triangle-screen coverage is a 2D signal
  - Undersampling and the use of simple (non-ideal) reconstruction filters may yield aliasing
  - In today's example, we reduced aliasing via supersampling
- **Image formation on a display**
  - When samples are 1-to-1 with display pixels, sample values are handed directly to display
  - When "supersampling", resample densely sampled signal down to display resolution
- **Sampling screen coverage of a projected triangle:**
  - Performed via three point-inside-edge tests
  - Real-world implementation challenge: balance conflicting goals of avoiding unnecessary point-in-triangle tests and maintaining parallelism in algorithm implementation

Further Reading

- A Pixel is Not A Little Square, Alvy Ray Smith, 1995
- A Nice Video Illustration of Temporal Aliasing. by Valvano and Yerraballi
- The Fourier Transform and It's Applications (Ch. 5) by B. Osgood (this is an outstanding reference)
- Stanford CS348B's Notes on Sampling. by Pat Hanrahan (also see Chapter 7.1 in the Physically Based Rendering book)

What you should know

1. How should we choose the correct color for a pixel? There is not an exact right answer. However, you should be able to discuss some of the issues involved.
2. What is aliasing, and what artifacts does it produce in our images and our animations?
3. One form of aliasing is where high frequencies masquerade as low frequencies. Give an example of this phenomenon.
4. Suppose we have a single red triangle displayed against a blue background. Does this scene contain high frequencies?
5. What does the Nyqvist-Shannon theorem tell us about how image frequencies relate to required sampling rate?
6. The practical solution on your graphics card for reducing aliasing (i.e., for antialiasing) is to take multiple samples per pixel and average to get pixel color. Try to use what we learned about sampling theory to explain as precisely as you can why taking multiple samples per pixel can reduce aliasing artifacts.
7. Be able to write an implicit representation of an edge given two points.
8. Be able to use the implicit edge representation to determine if a point is inside a triangle.

Practice Questions

1. Describe why sampling a signal more densely is one way to reduce aliasing. Is there a downside of increasing the rate at which a signal is sampled?
2. Consider a quadrilateral with the following vertex positions. (You should assume the vertices are connected in the order they are listed.)

P0=(1 , 1)

P1=(3 , 1.5)

P2=(2.5, 3.5)

P3=(1 , 3.75)

Derive the implicit edge equation for the edge between P0 and P1.

Assuming coverage sample points are uniformally distributed in the domain at half-integer coordinates (0.5, 0.5), (1.5, 0.5), etc., how many sample points are covered by the given quadrilateral? For simplicity, assume that a sample point on an edge is considered to be within the quadrilateral.