Computer Graphics
- Sampling & Antialiasing

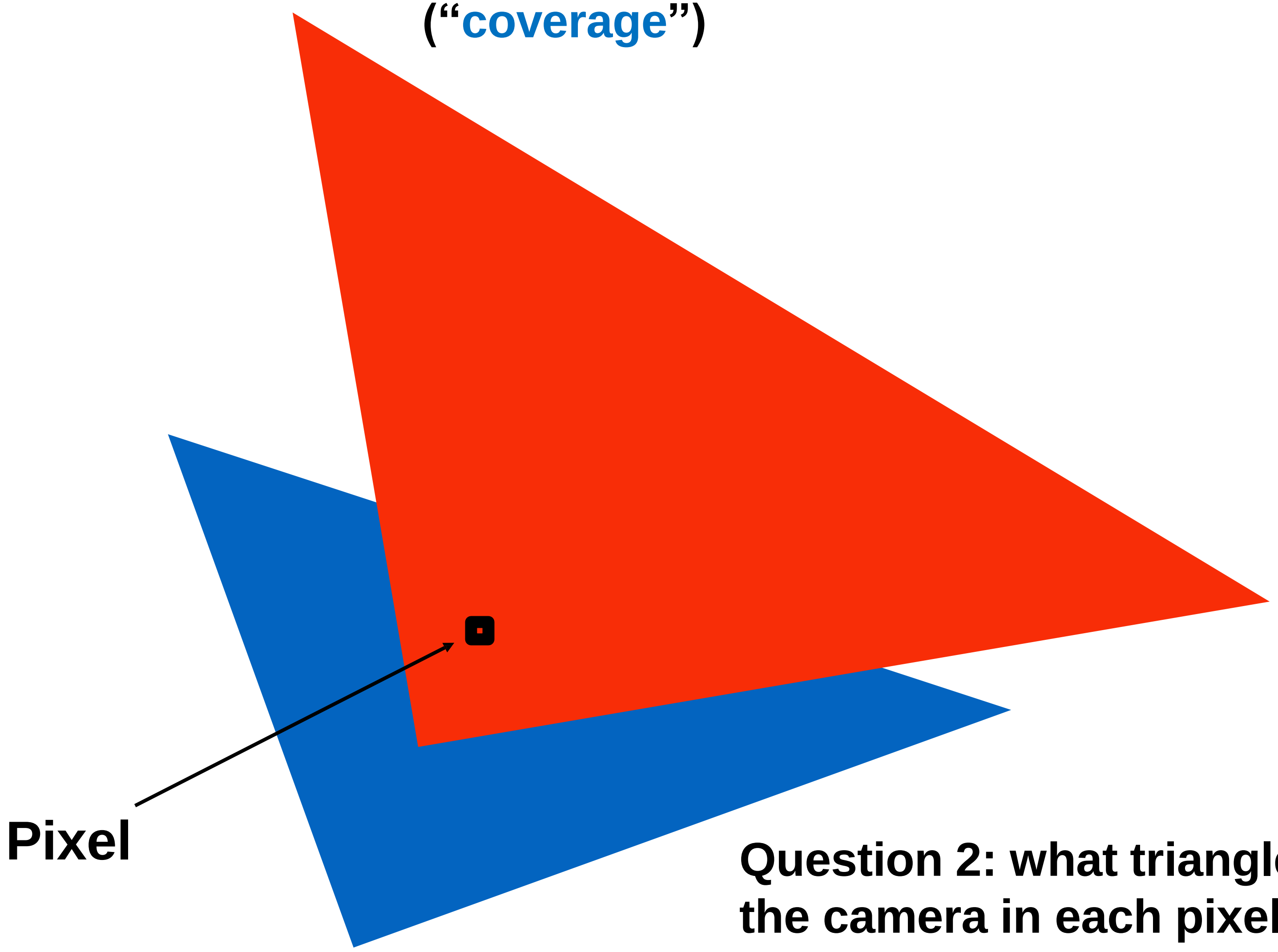
Junjie Cao @ DLUT

Spring 2017

http://jjcao.github.io/ComputerGraphics/

Let’s draw some triangles on the screen!

Question 1: what pixels does the triangle overlap? (“coverage”)

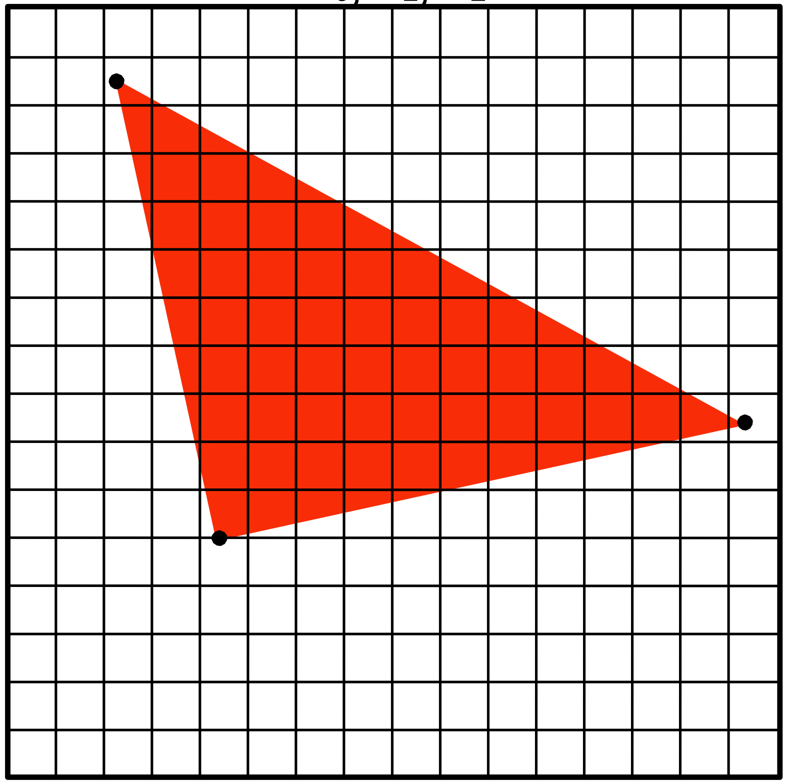


Question 2: what triangle is closest to the camera in each pixel? (“occlusion”)

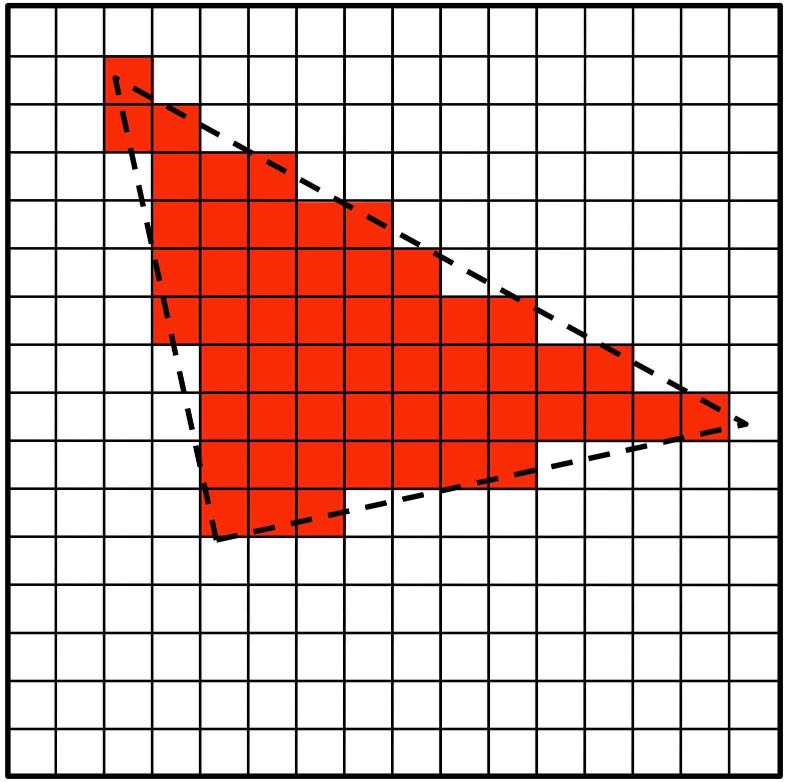
Computing triangle coverage

What pixels does the triangle overlap?

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



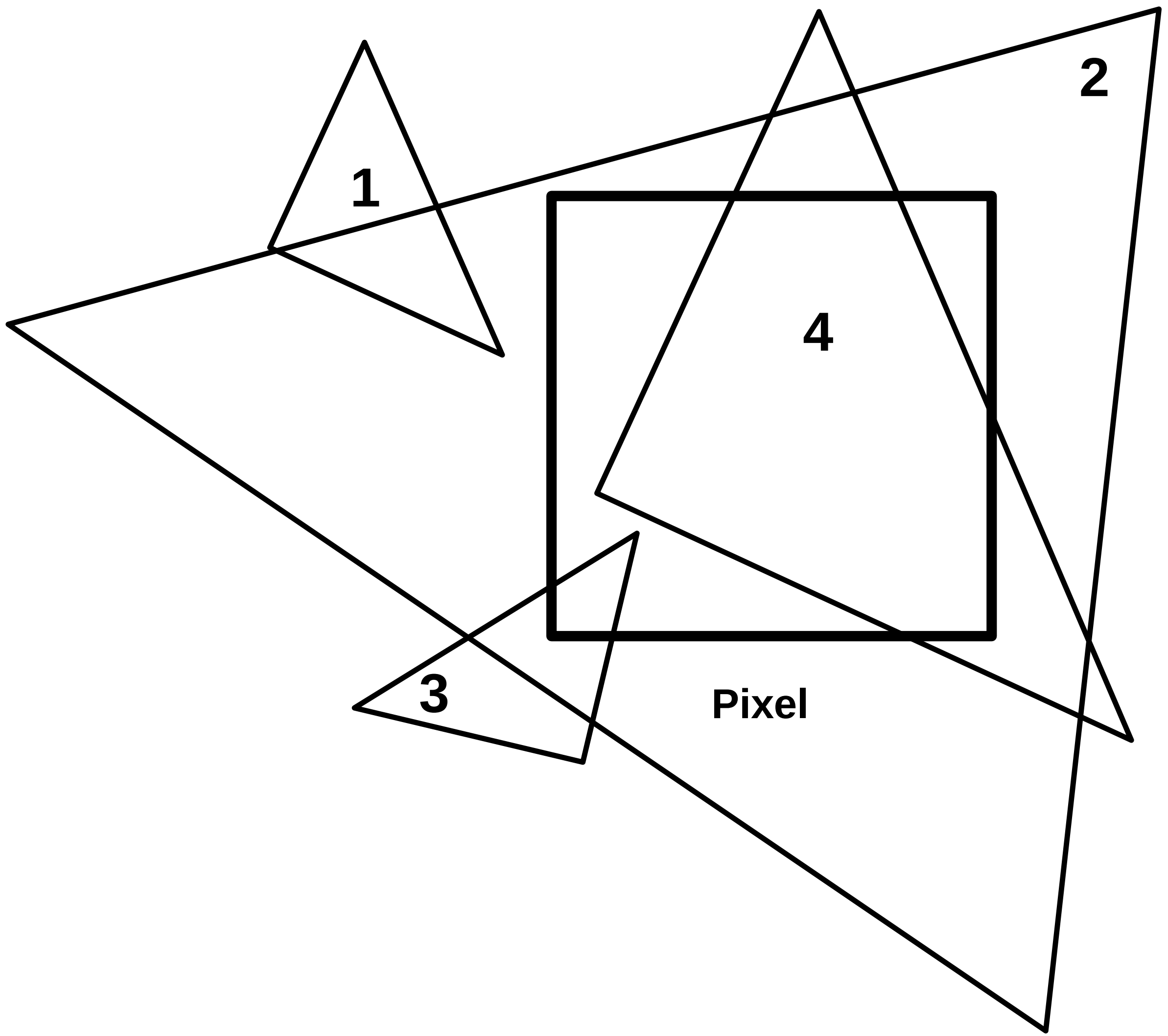
Output:
set of pixels "covered" by the triangle



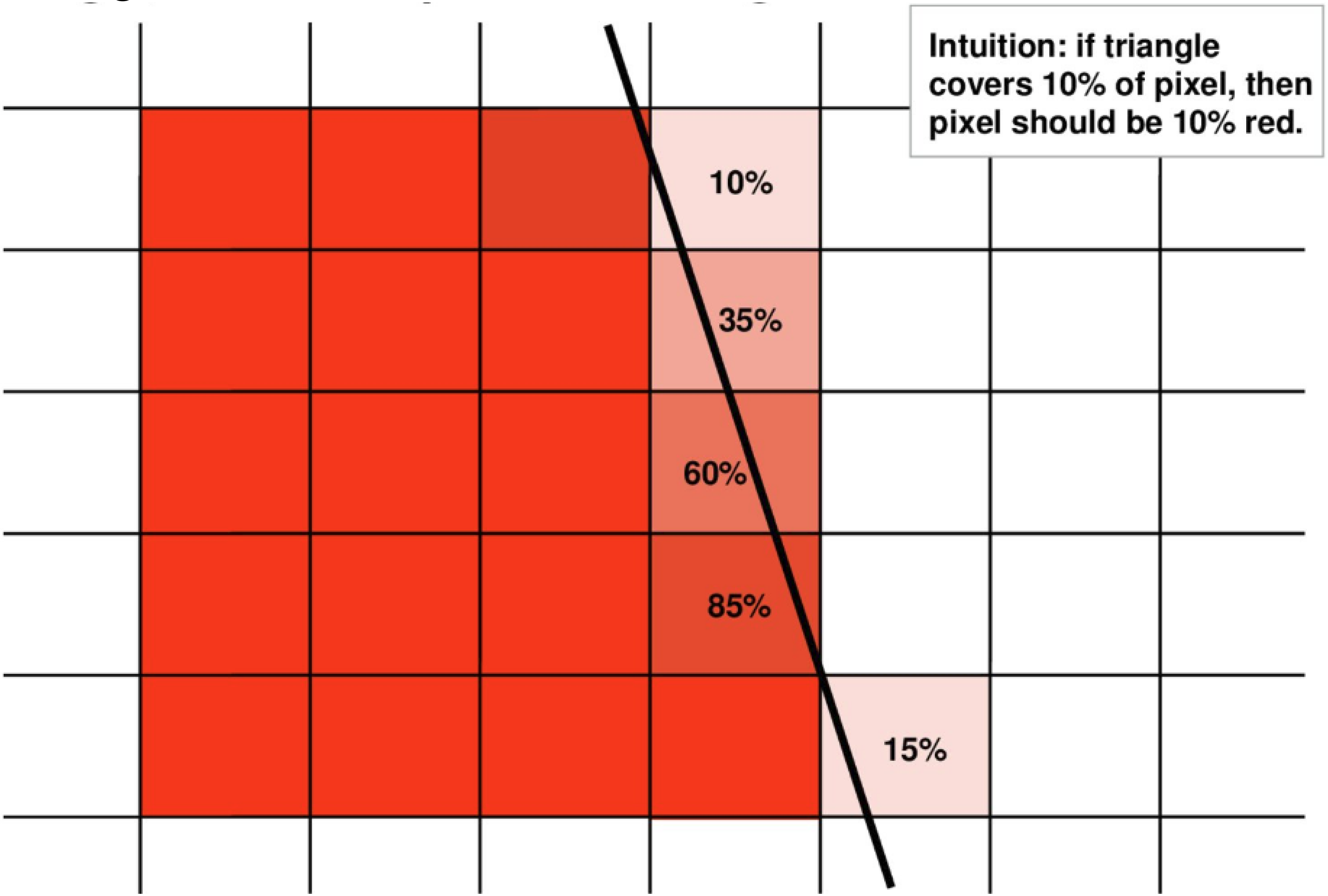
Note: need to represent a continuous signal using a discrete approximation!

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

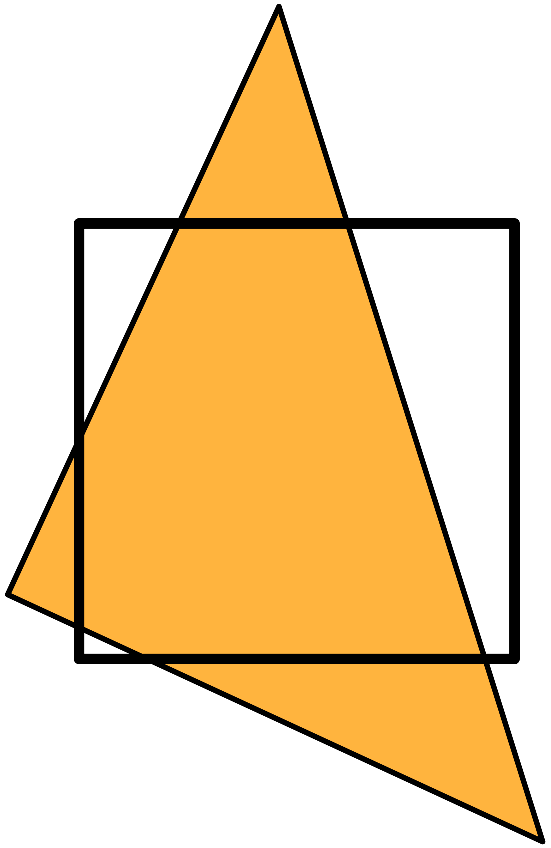
Question: which triangles "cover" this pixel?



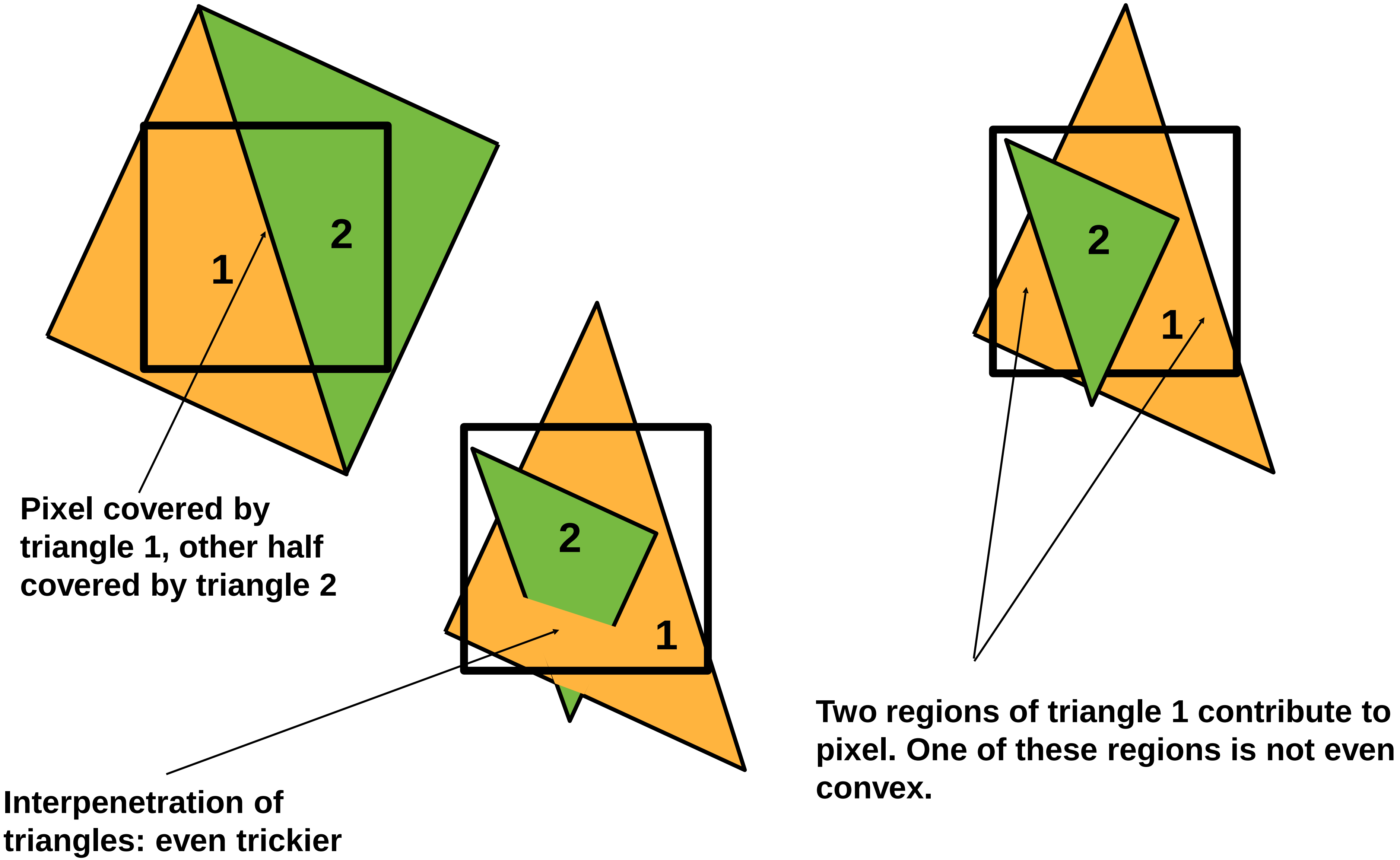
One option: compute fraction of pixel area covered by triangle, then color pixel according to this fraction.



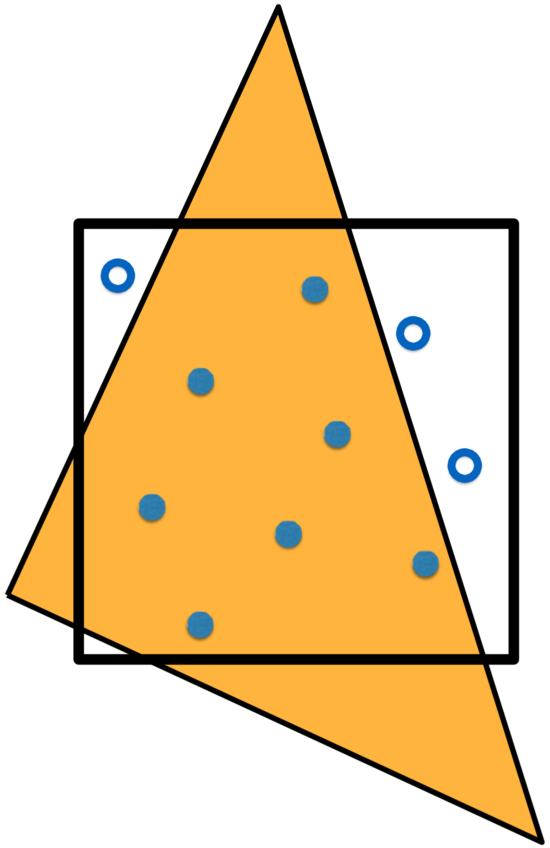
Computing amount of overlap?



Analytical schemes can get quite tricky, especially when considering interactions between multiple triangles



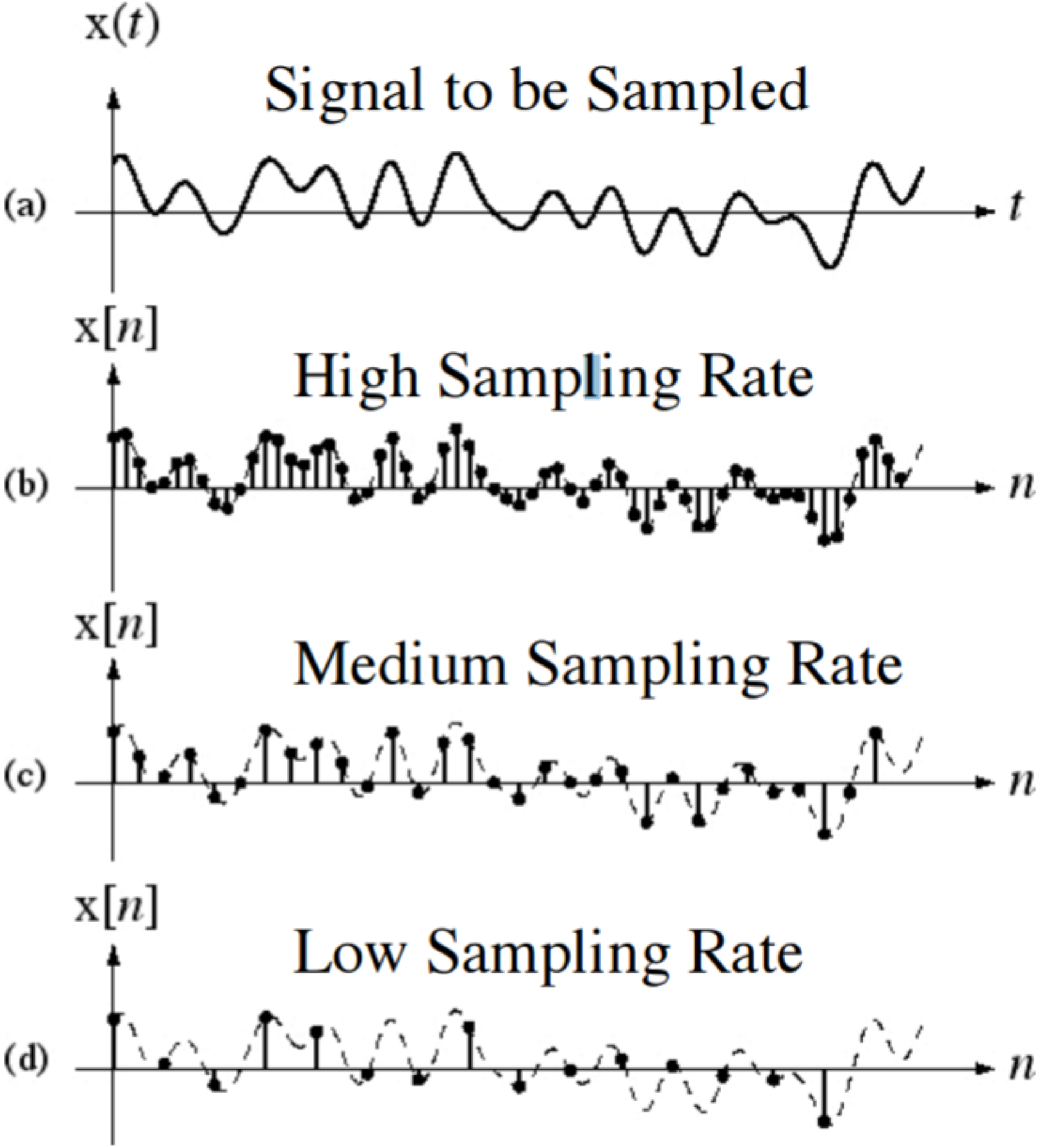
Estimating amount of overlap through sampling



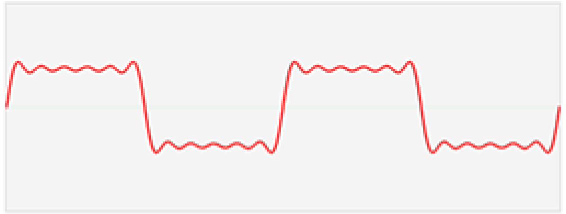
What is a principled approach to think about this process?

Sampling 101

Sampling


x(t)
Signal to be Sampled
(a)
t
x[n]
High Sampling Rate
(b)
n
x[n]
Medium Sampling Rate
(c)
n
x[n]
Low Sampling Rate
(d)
n

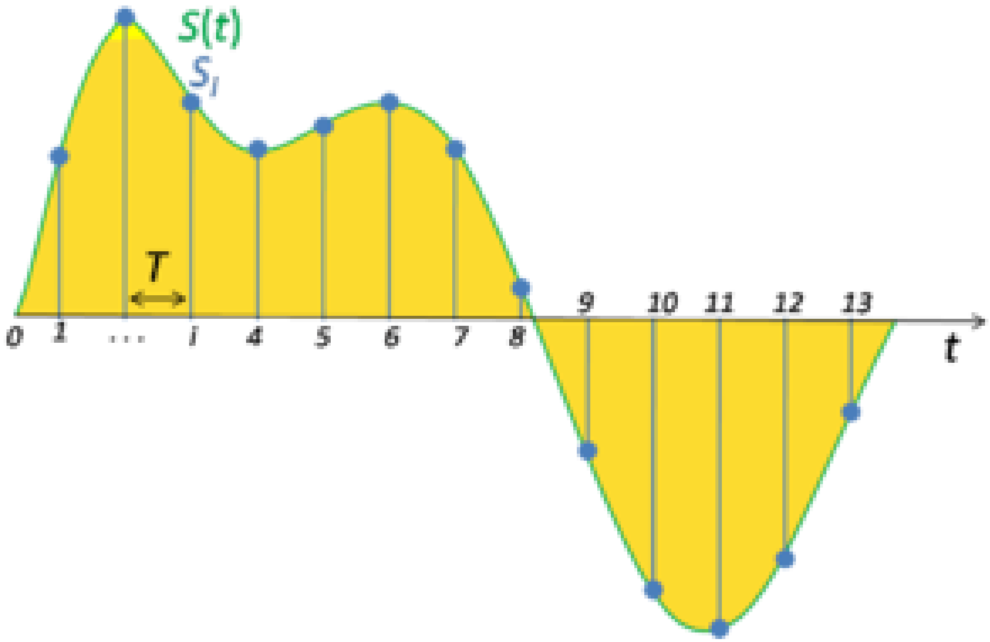
1D Temporal Signal and Sampling



Frequency	1 mHz (10^{-3} Hz)	1 Hz (10^{0} Hz)	1 kHz (10^{3} Hz)	1 MHz (10^{6} Hz)	1 GHz (10^{9} Hz)	1 THz (10^{12} Hz)
Period	1 ks (10^{3} s)	1 s (10^{0} s)	1 ms (10^{-3} s)	1 µs (10^{-6} s)	1 ns (10^{-9} s)	1 ps (10^{-12} s)

$$f = \frac{1}{T}$$

- T: **sampling period** or sampling interval



Audio file: stores samples of a 1D signal

Most consumer audio is sampled at 44.1 KHz

Video's temporal sampling rate is 24 frame per second, i.e. 24/60 Hz

Video's spatial sampling rate is:

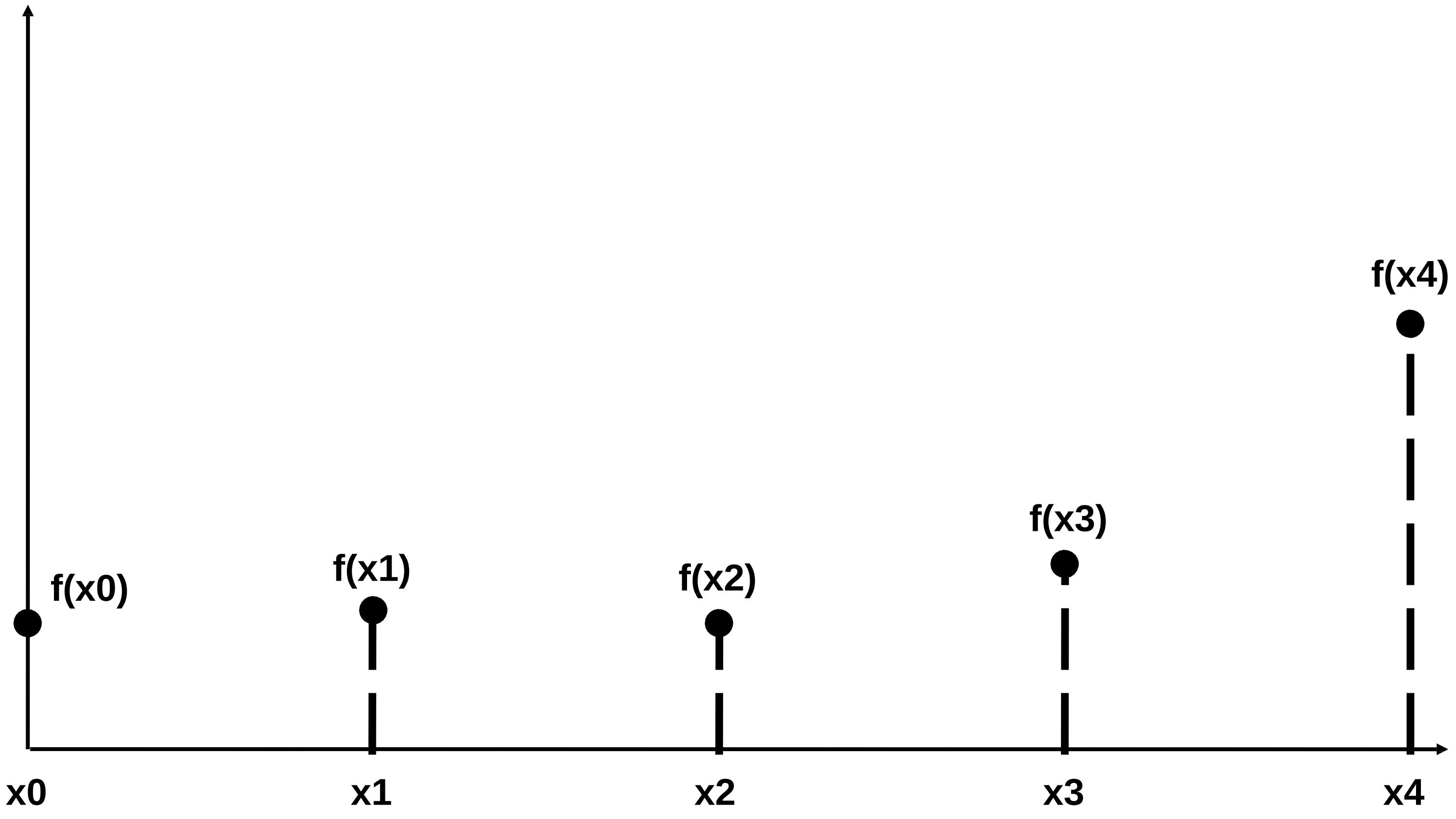
- 720p (1280×720 px; also called HD Ready)
- 1080p (1920×1080 px; also known as full HD)

Q: Why 44.1Khz?

Frequency	1 mHz (10^{-3} Hz)	1 Hz (10^{0} Hz)	1 kHz (10^{3} Hz)	1 MHz (10^{6} Hz)	1 GHz (10^{9} Hz)	1 THz (10^{12} Hz)
Period	1 ks (10^{3} s)	1 s (10^{0} s)	1 ms (10^{-3} s)	1 µs (10^{-6} s)	1 ns (10^{-9} s)	1 ps (10^{-12} s)

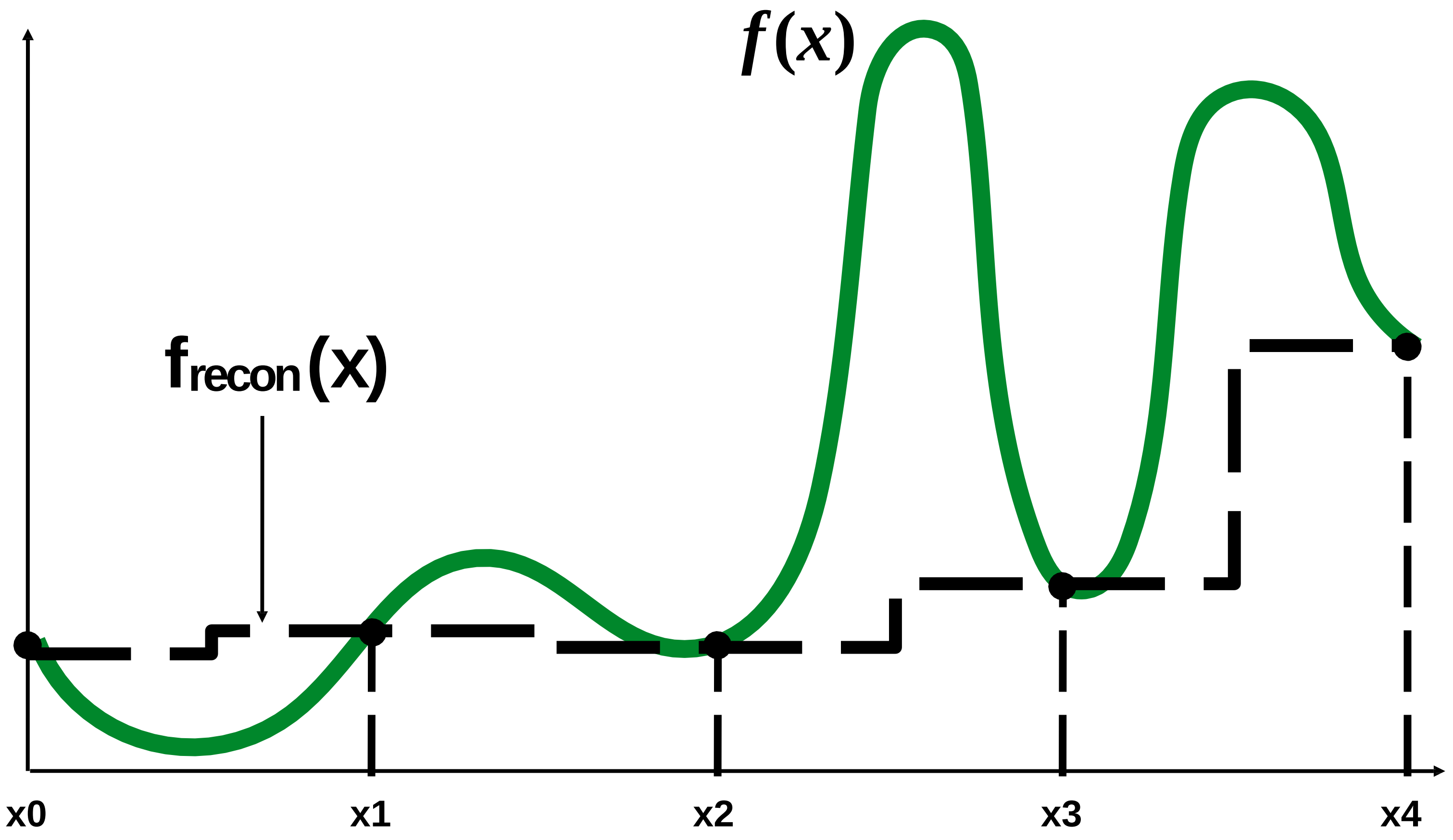
Reconstruction: from discrete to continuous (an interpolation problem)

$f_{recon}(x)$ is the reconstructed version of the original function $f(x)$



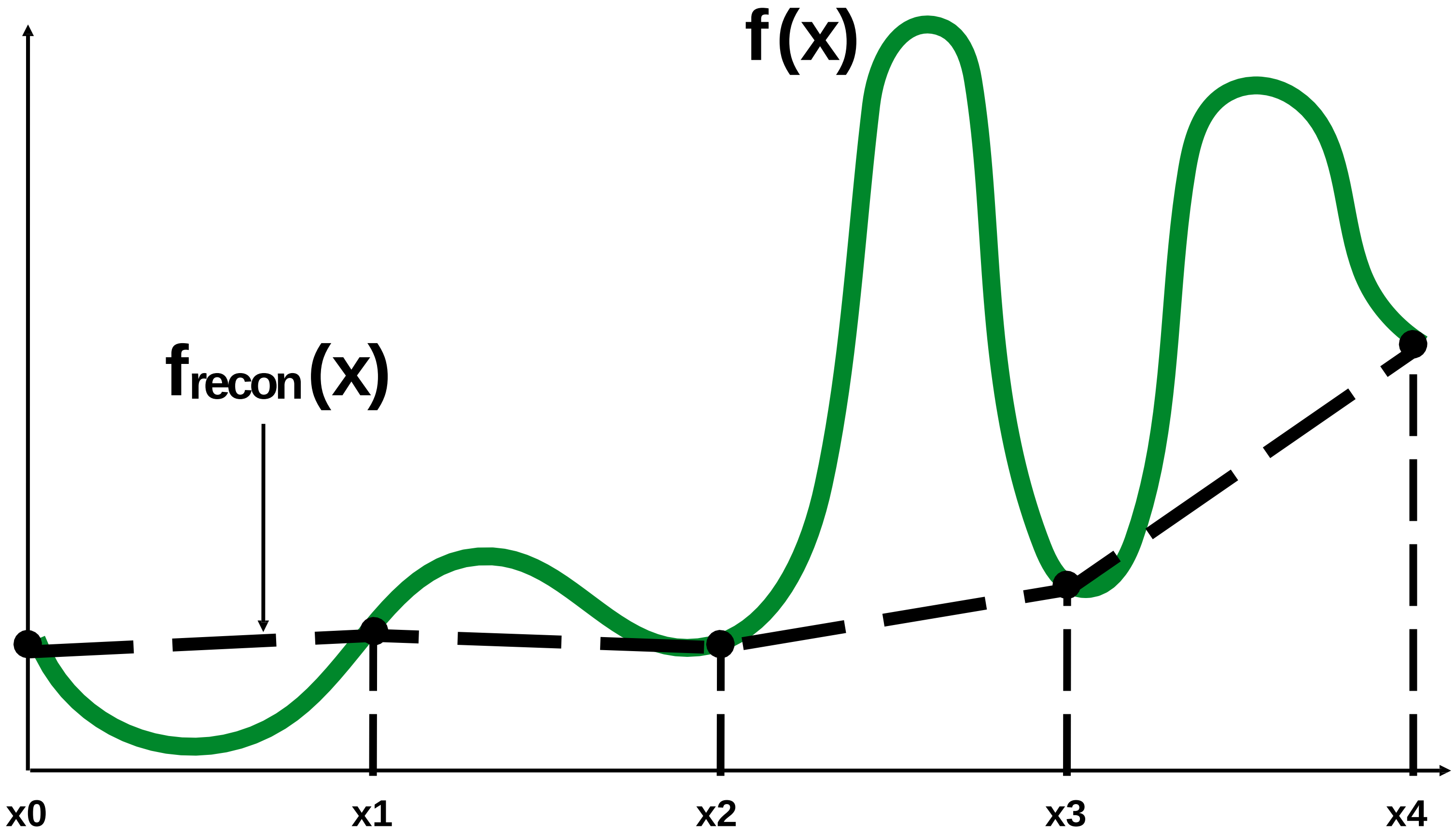
Piecewise constant approximation

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

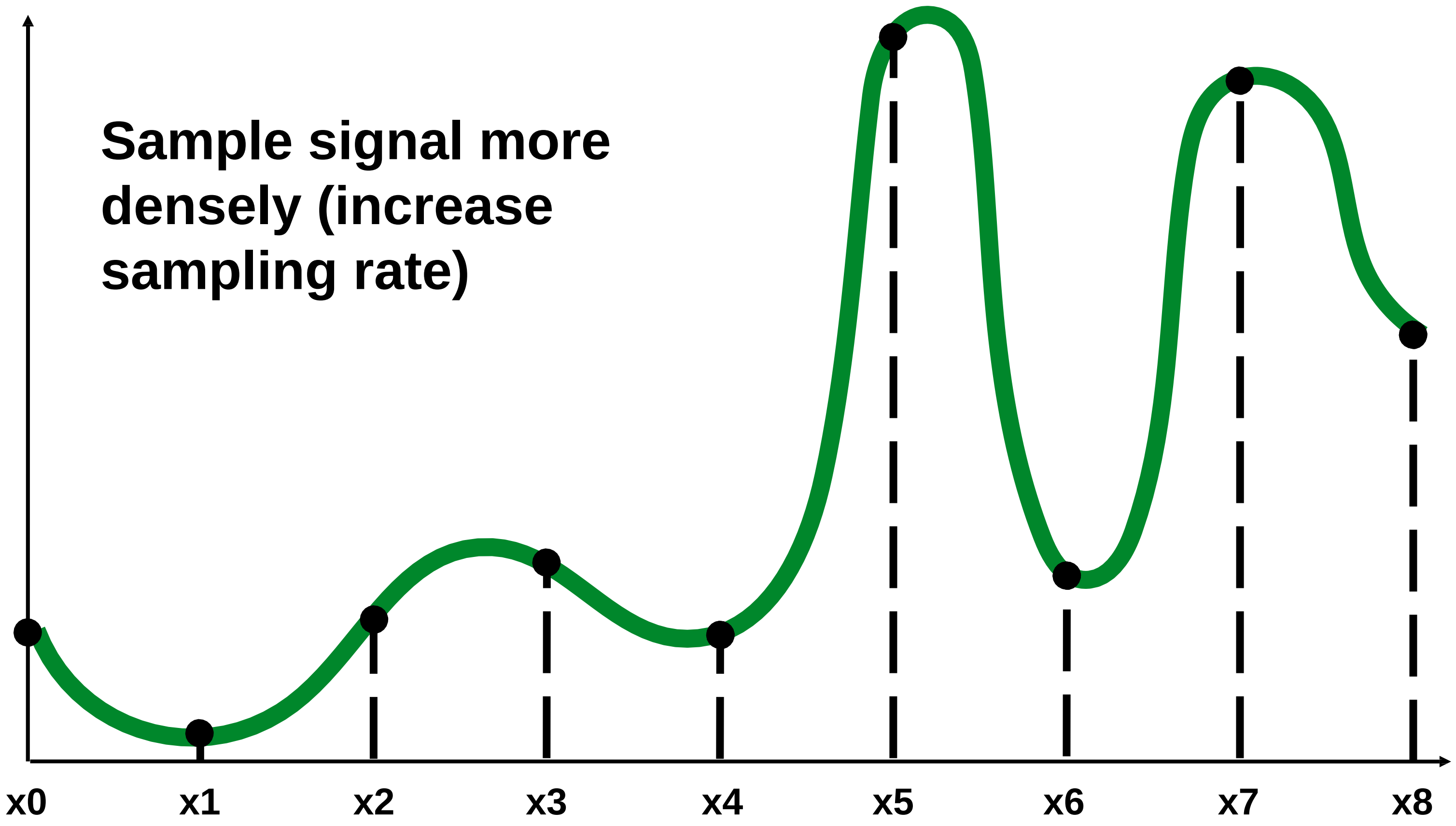


Piecewise linear approximation

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

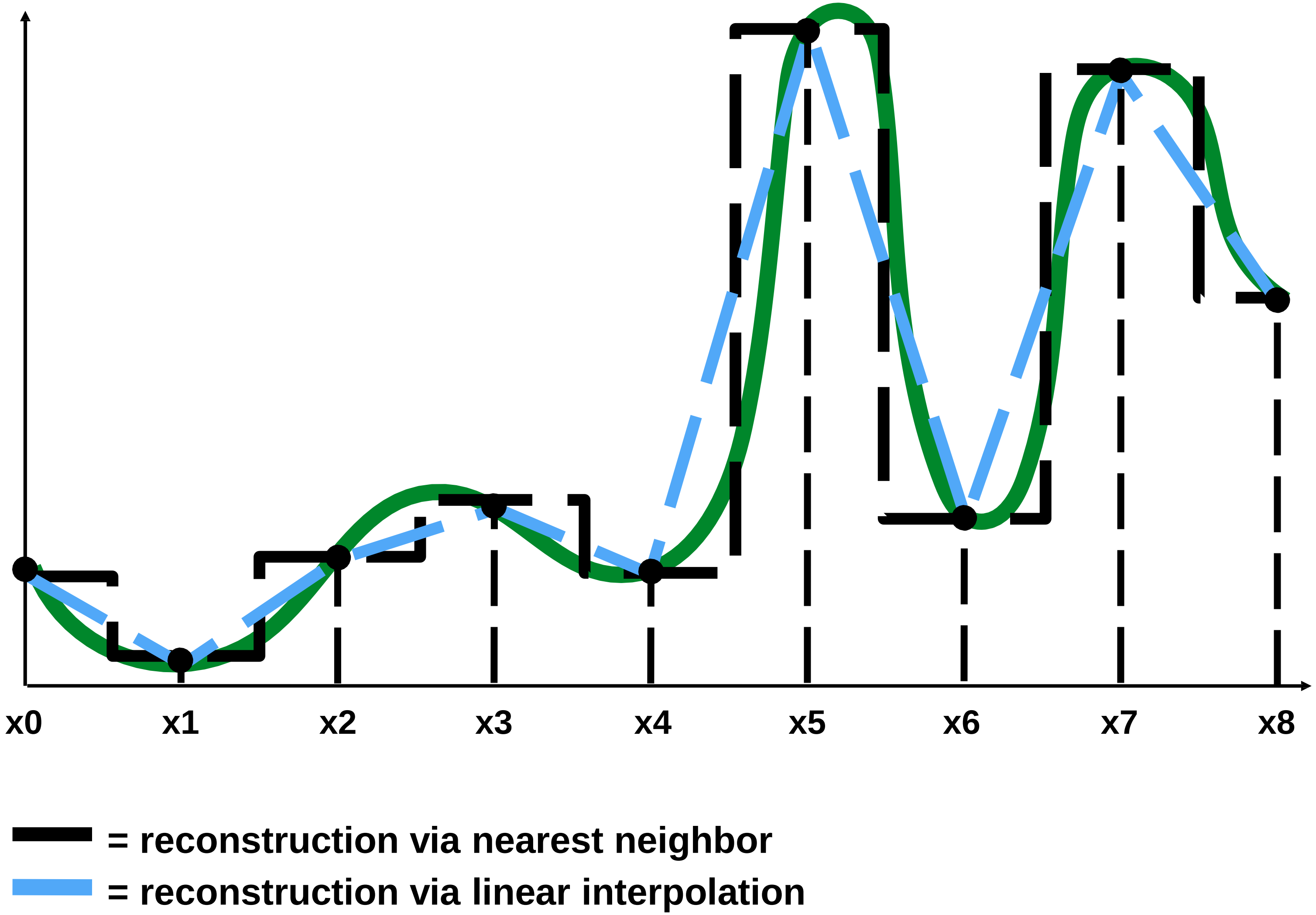


How can we reconstruct the signal more accurately?



Q: What does “increase sampling rate” mean for our problem?

Reconstruction from denser sampling



Sampling and Reconstruction

- As an aside
  - Sampling rate is obviously very important
    - Why limit it?

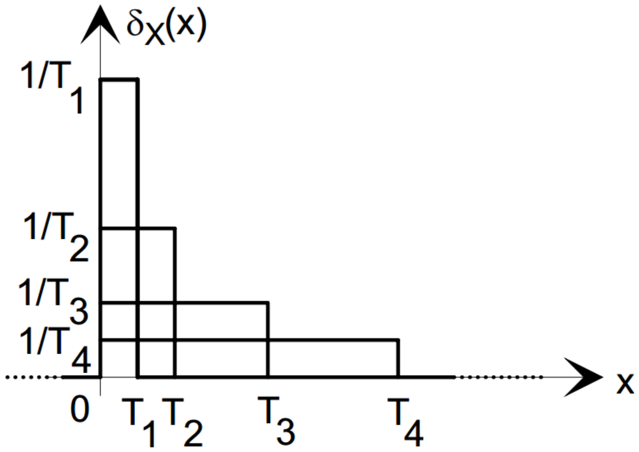
Mathematical representation of sampling

Consider the Dirac delta:

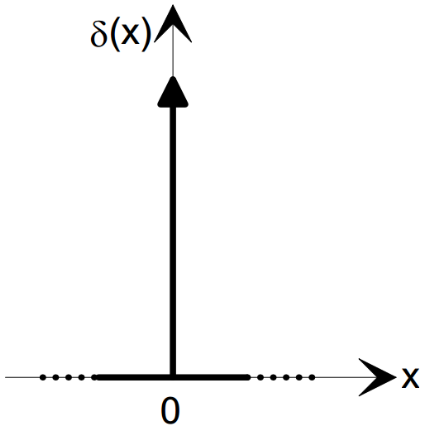
$$\delta(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ \text{undefined} & \text{at } x = 0 \end{cases}$$

s.t.

$$\int_{-\infty}^{\infty} \delta(x) dx = 1$$



Unit pulse functions

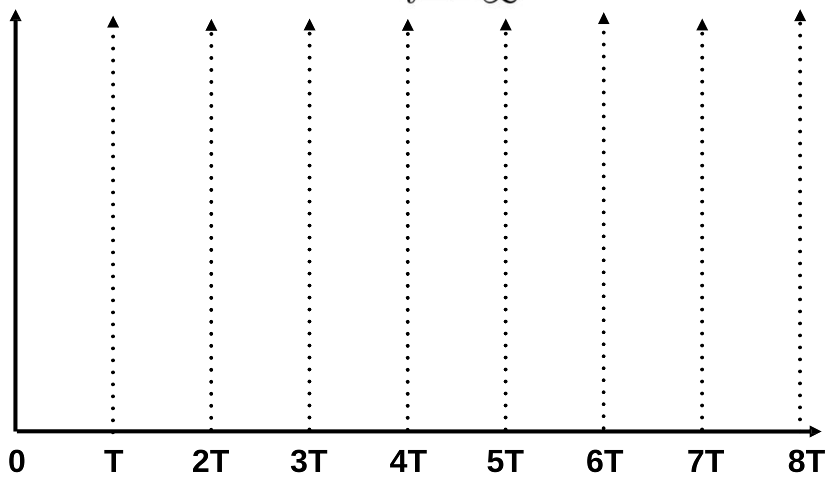


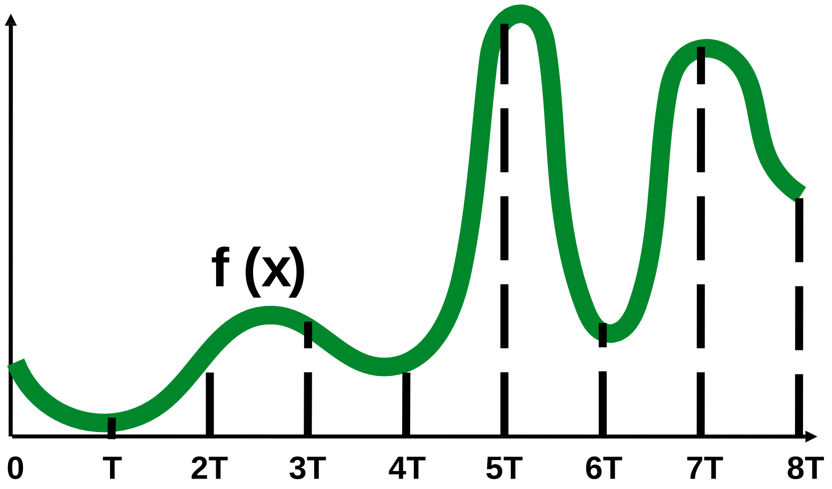
An Impulse

Sampling function

Consider a sequence of impulses with period T:

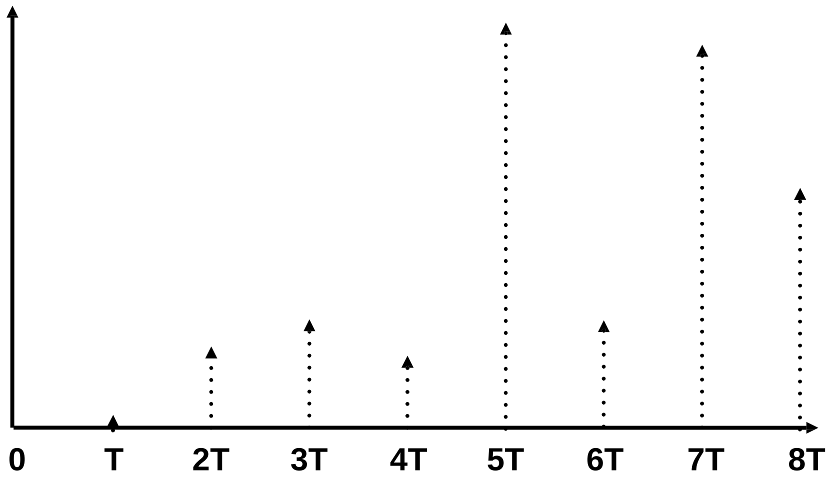
$$\text{Ш}_T(x) = T \sum_{i=-\infty}^{\infty} \delta(x - iT)$$





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

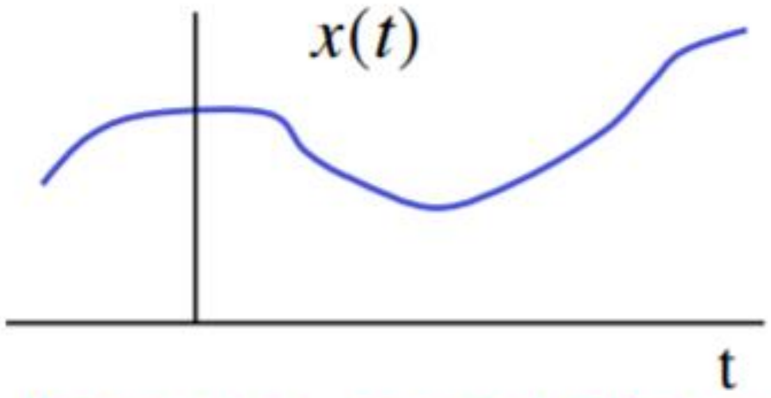
So we can write the result of sampling as a product of f(x) and a sequence of impulses centered around each sample point



Discrete-time Signals

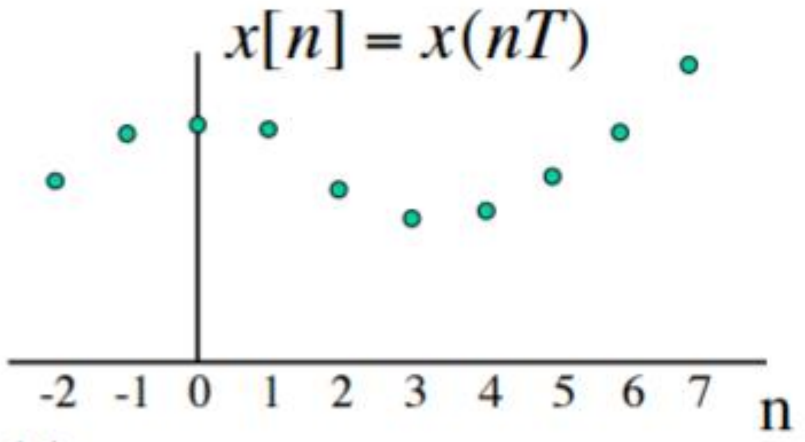
Continuous-time signal $x(t)$

math: real function of t



Discrete-time signal x_n

math: sequence

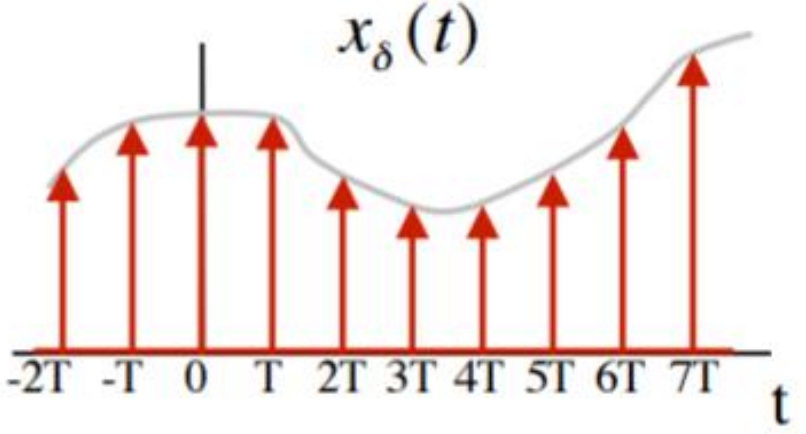


Impulse-sampled signal $x_\delta(t)$

math: train of impulses

$$x_\delta(t) = x(t)\sum_{n=-\infty}^{\infty}\delta(t-nT)$$

$$= \sum_{n=-\infty}^{\infty} x[n]\delta(t-nT)$$

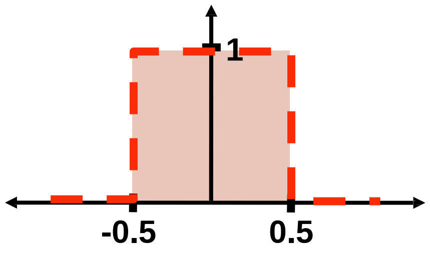


Reconstruction as convolution

$$\underbrace{(f * g)(x)}_{\text{reconstructed signal ("smooth" version of g)}} = \int_{-\infty}^{\infty} \underbrace{f(y)}_{\text{filter}} \underbrace{g(x-y)}_{\text{input signal (sampled signal)}} dy$$

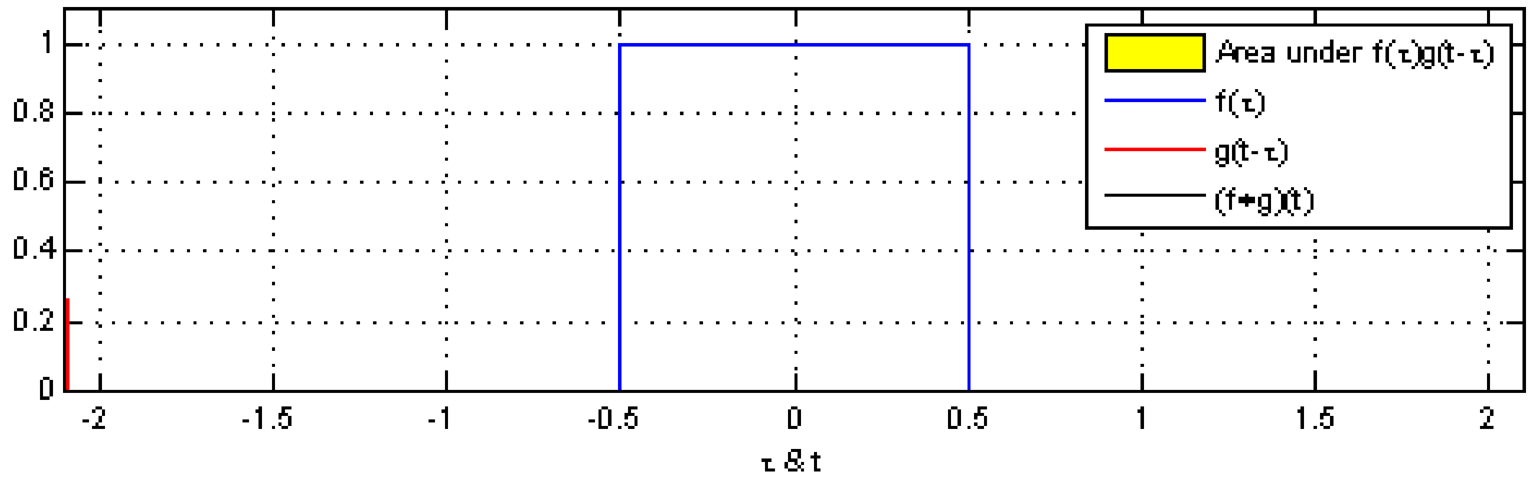
It may be helpful to consider the effect of convolution with the simple unit-area "box" function:

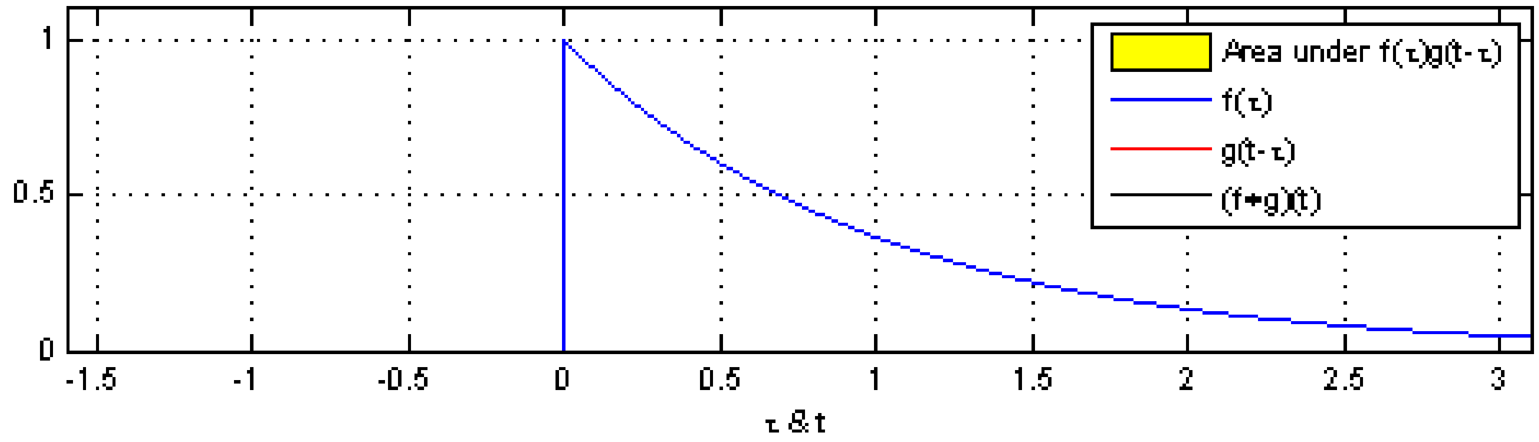
$$f(x) = \begin{cases} 1 & |x| \le 0.5 \\ 0 & otherwise \end{cases}$$



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

Convolution





Reconstruction as convolution (box filter)

Sampled signal: (with period T)

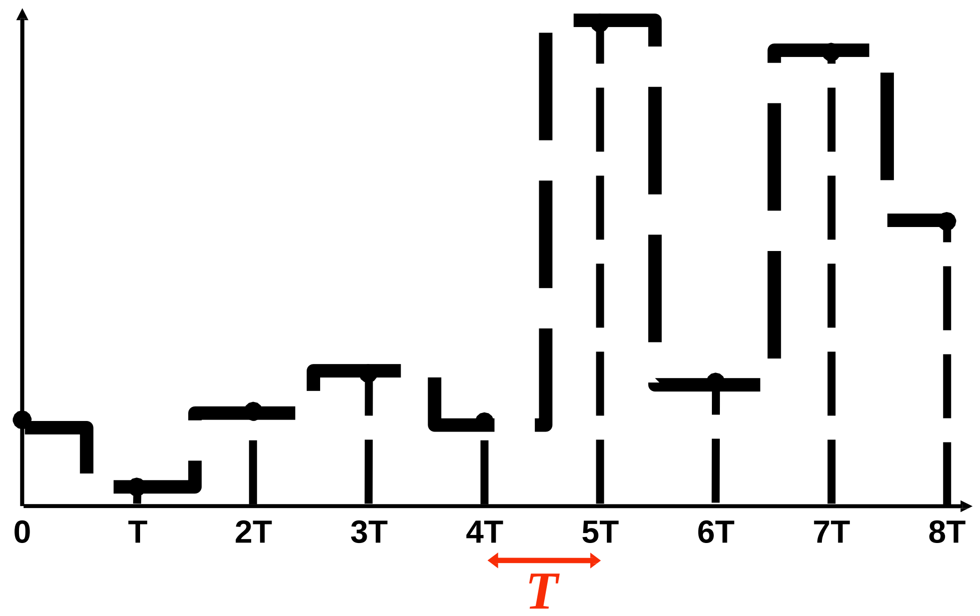
$$g(x) = Ш_T(x)f(x) = T\sum_{i=-\infty}^{\infty} f(iT)\delta(x - iT)$$

Reconstruction filter:
(unit area box of width T)

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

Reconstructed signal:

(nearest neighbor)



$$f_{recon}(x) = (h * g)(x) = T\int_{-\infty}^{\infty} h(y) \sum_{i=-\infty}^{\infty} \underbrace{f(iT)\delta(x - y - iT)dy}_{\text{non-zero only for iT closest to x}}$$

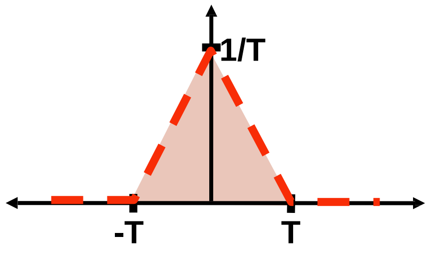
Reconstruction as convolution (triangle filter)

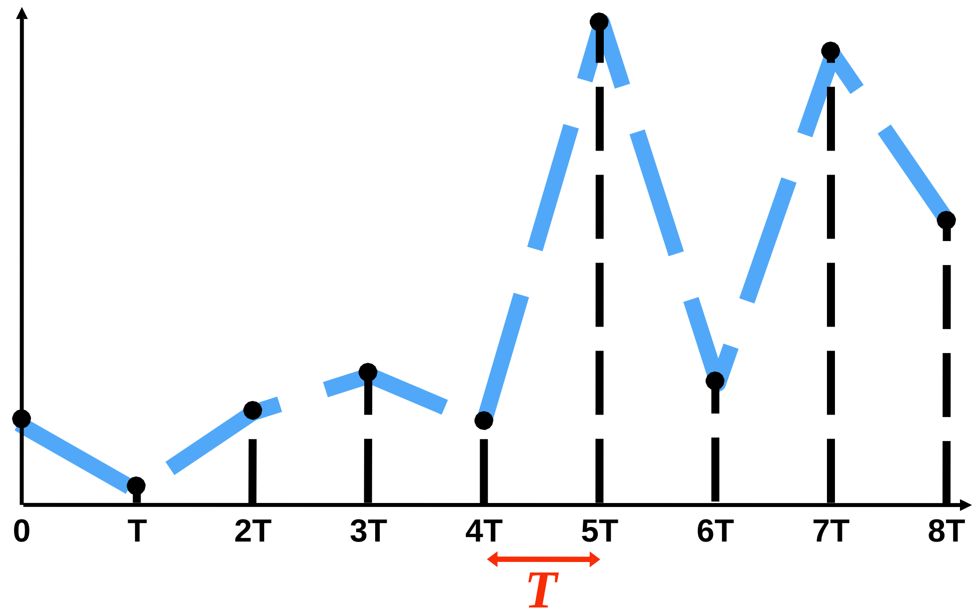
Sampled signal:
(with period T)

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

Reconstruction filter:
(unit area triangle of width T)

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





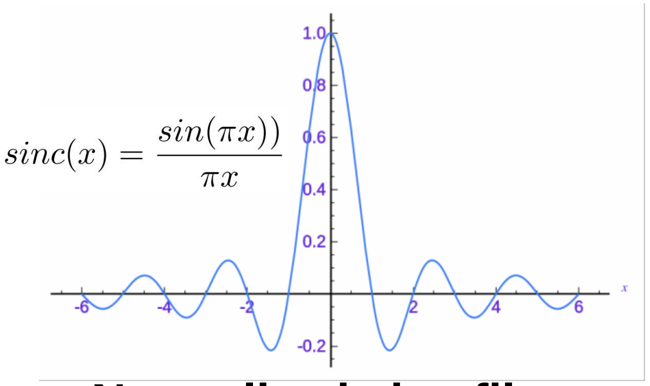
Reconstructed signal:

$$f_{recon}(x) = (h * g)(x) = \int_{-\infty}^{\infty} h(y) g(x - y) dy = \ldots$$

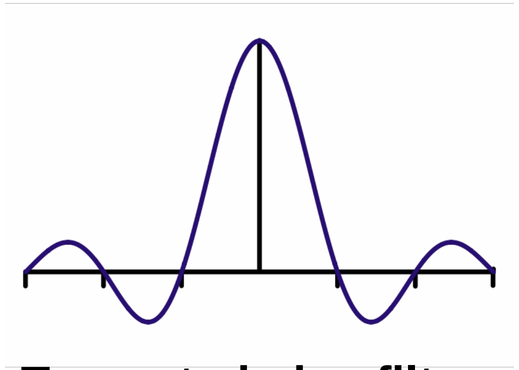
Summary

- Sampling = measurement of a signal
  - Represent signal as discrete set of samples
  - Mathematically described as multiplication by impulse train

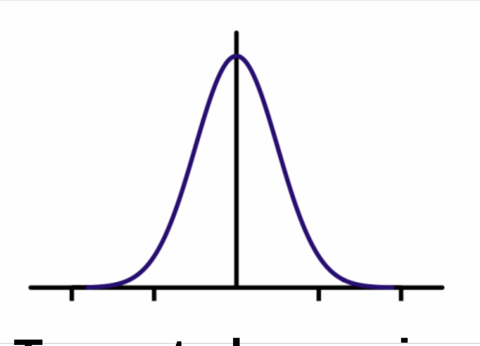
- Reconstruction = generating signal from a discrete set of samples
  - Convolution of sampled signal with a reconstruction filter
  - Intuition: value of reconstructed function at any point in domain is a combination of sampled values
  - We discussed simple box & triangle filters, but there are other, much higher quality filters



Normalized sinc filter



Truncated sinc filter



Truncated gaussian filter

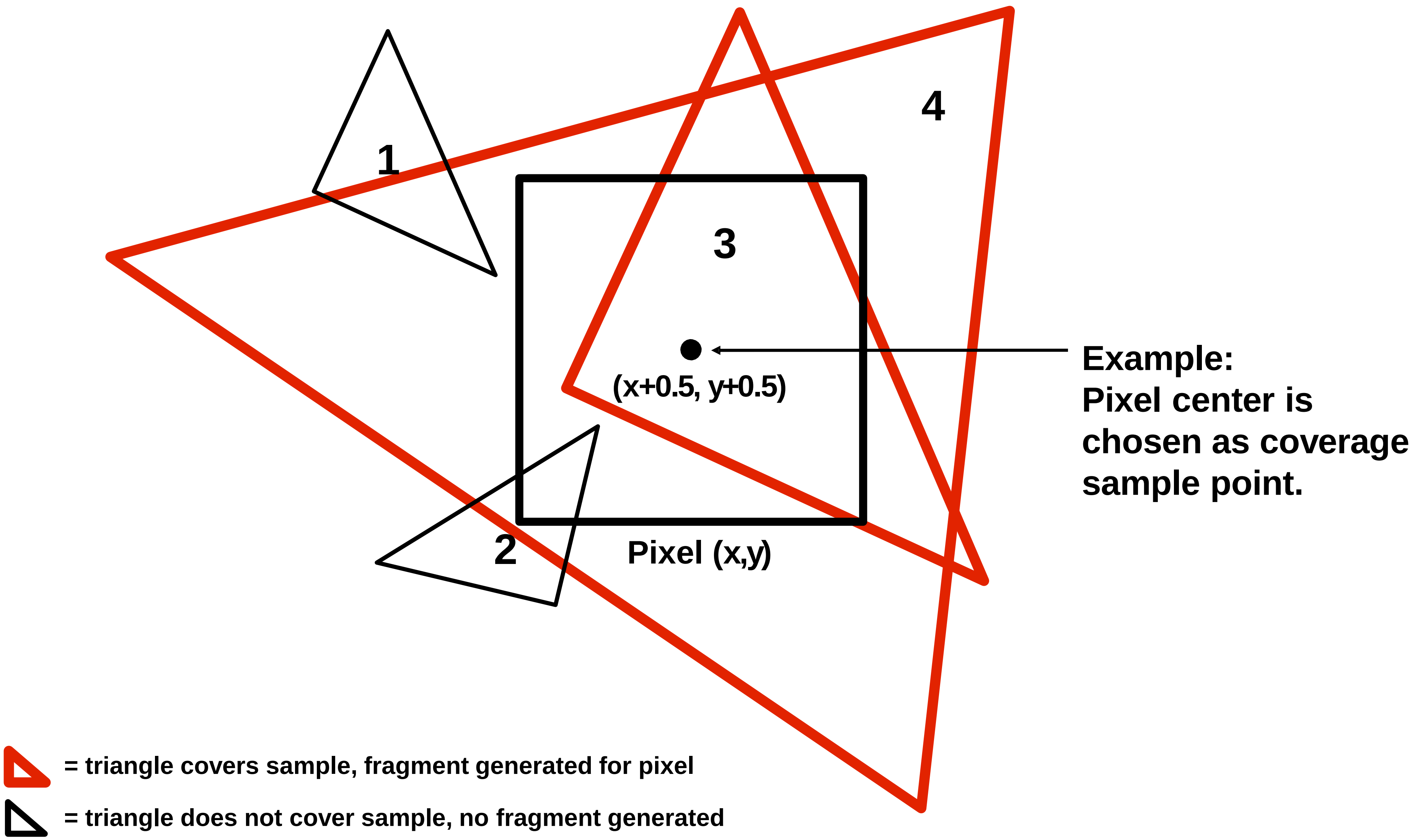
[Image credit: Wikipedia]

Now back to computing coverage

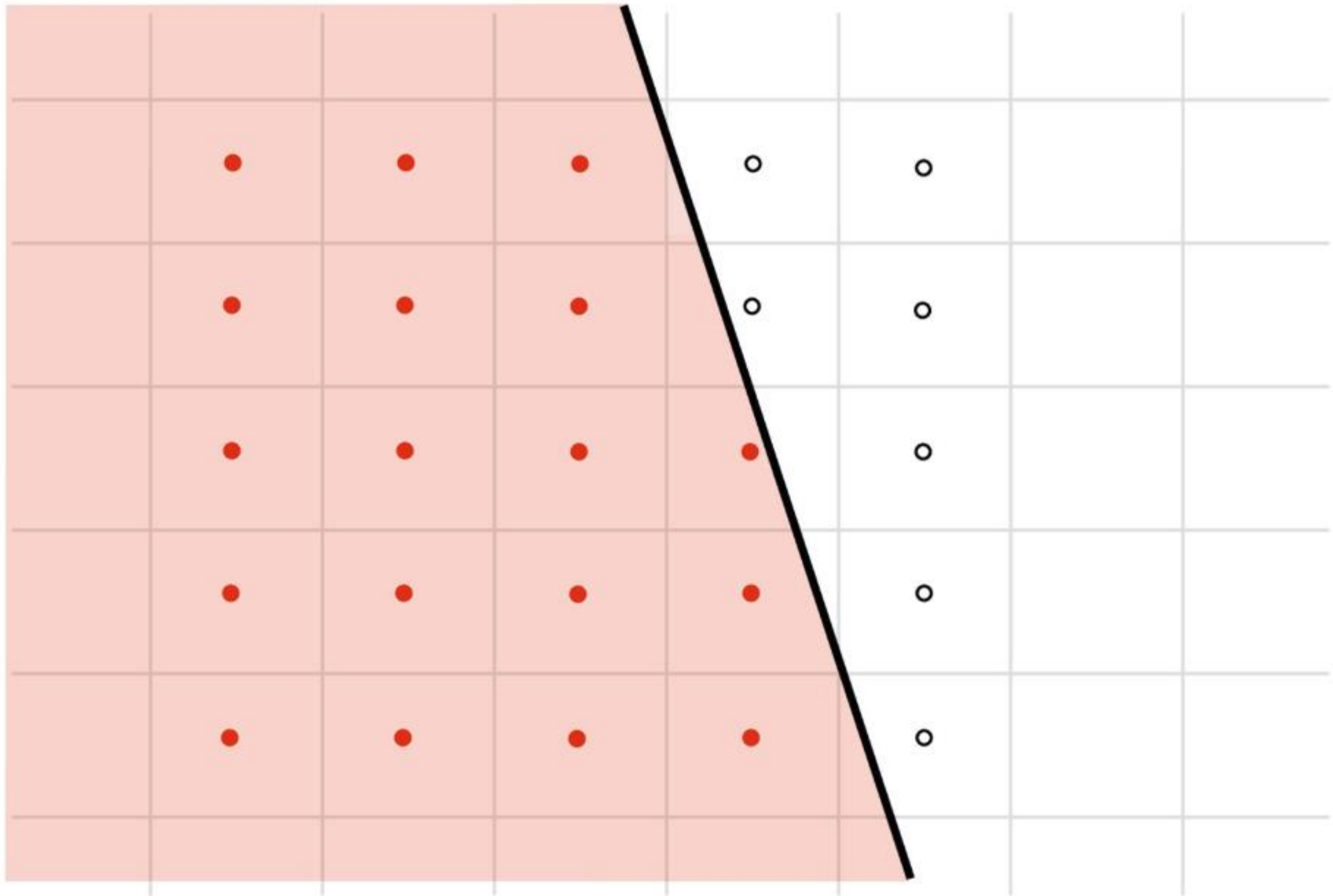
Think of coverage as a 2D signal

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

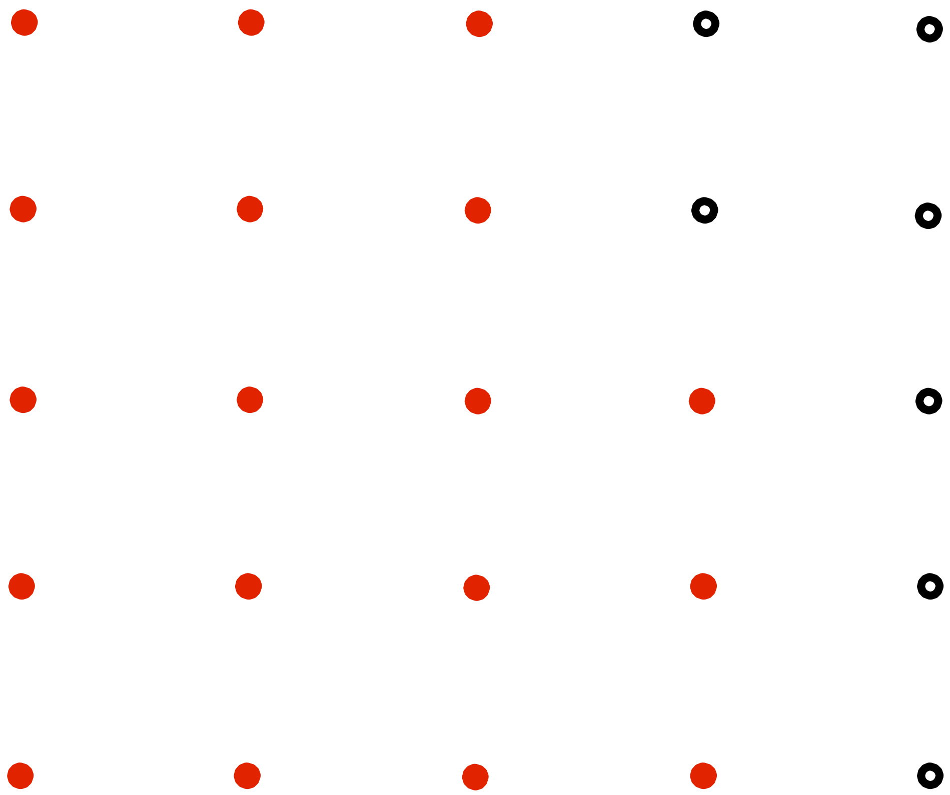
Estimate triangle-screen coverage by sampling the binary function: coverage(x,y)



Results of sampling triangle coverage



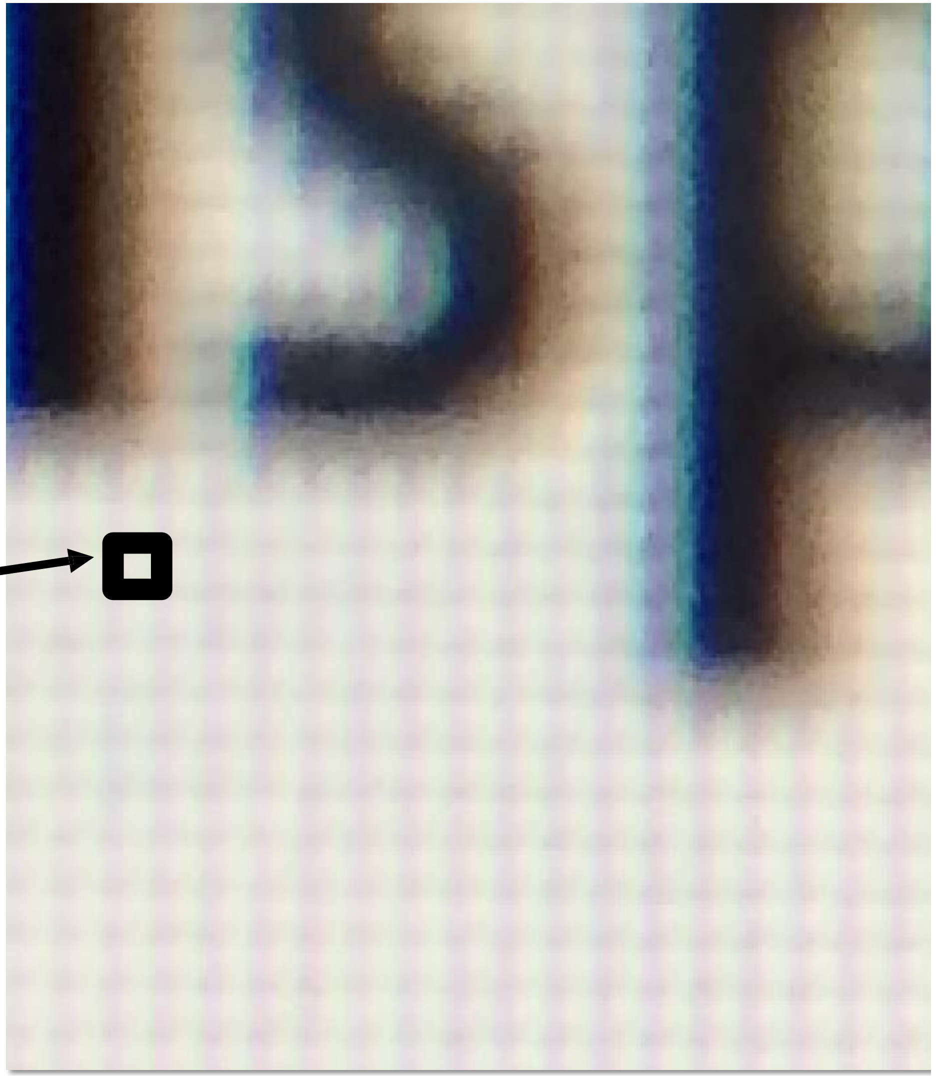
I have a sampled signal, now I want to display it on a screen



Pixels on a screen

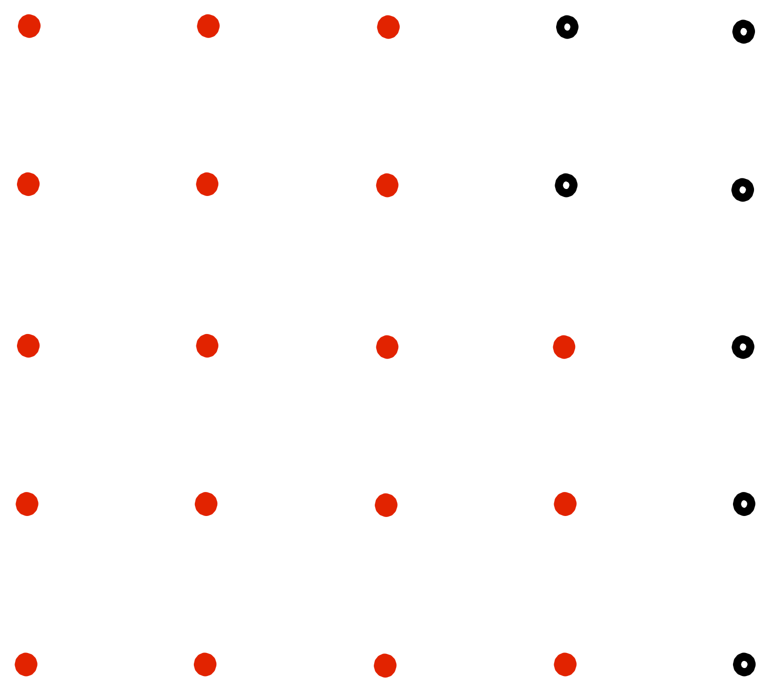
Each image sample sent to the display is converted into a little square of light of the appropriate color:
(a pixel = picture element)

LCD
display pixel

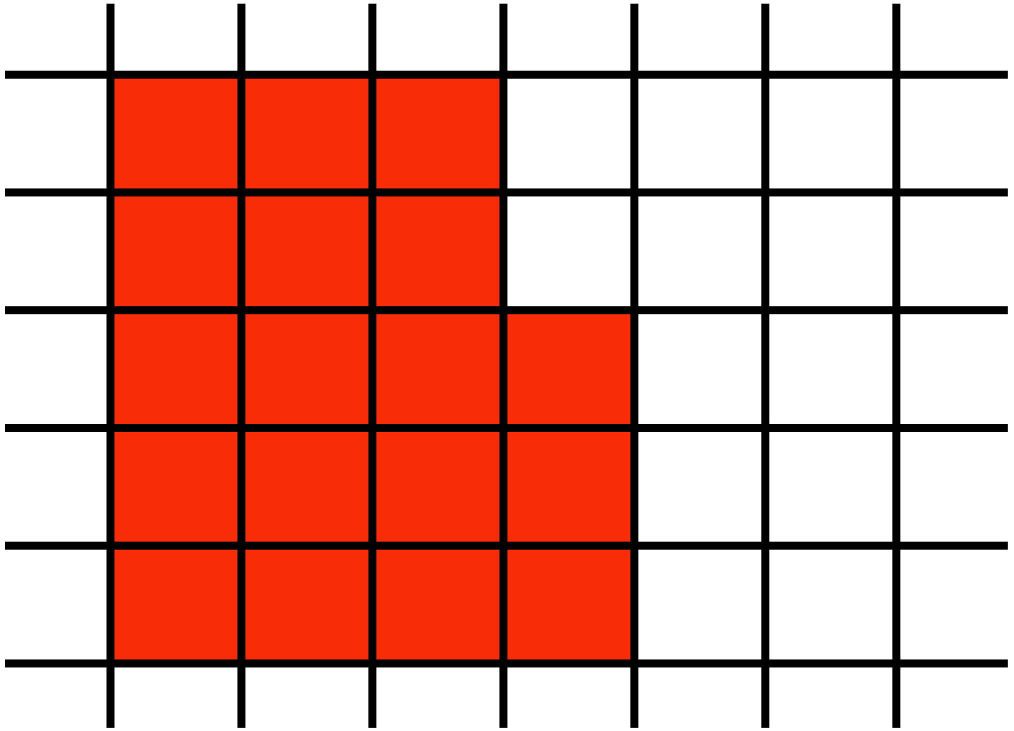


*** Thinking of each LCD pixel as emitting a square of uniform intensity light of a single color is a bit of an approximation to how real displays work, but it will do for now.**

So if we send the display this



We see this on the screen

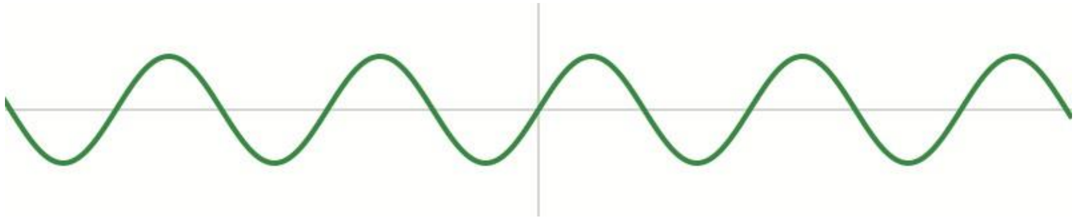


Recall: the real coverage signal

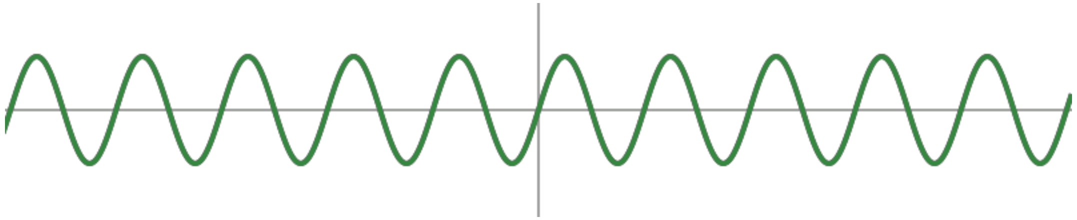
Aliasing

Representing signals as a superposition of frequencies

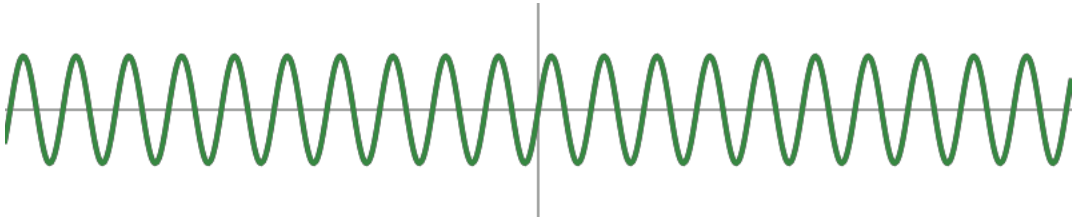
$f_1(x) = \sin(\pi x)$



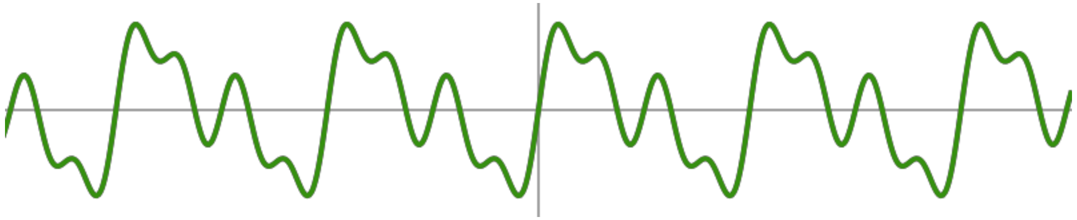
$f_2(x) = \sin(2\pi x)$



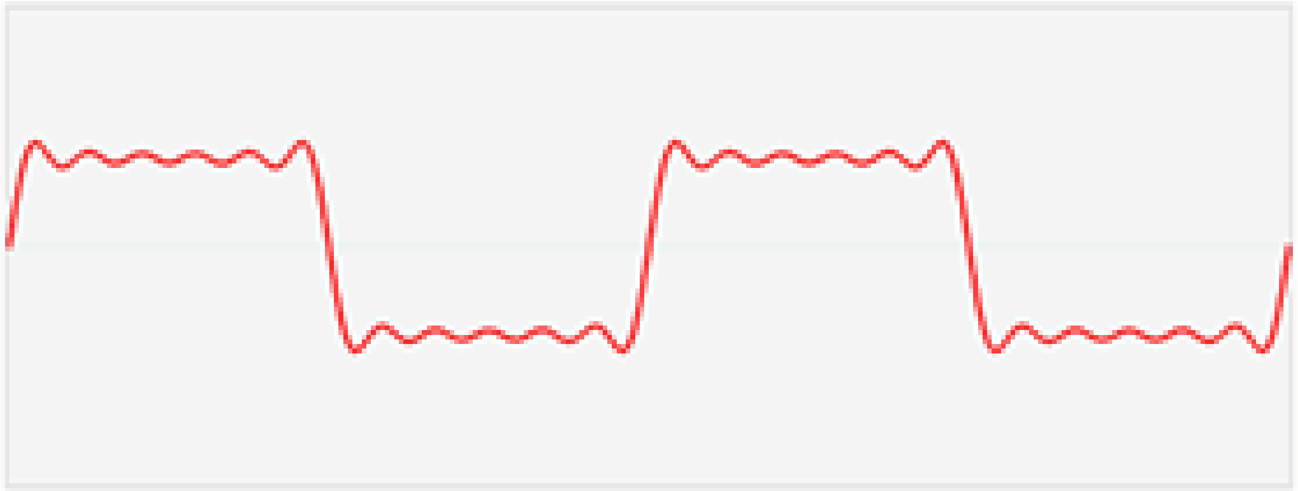
$f_4(x) = \sin(4\pi x)$



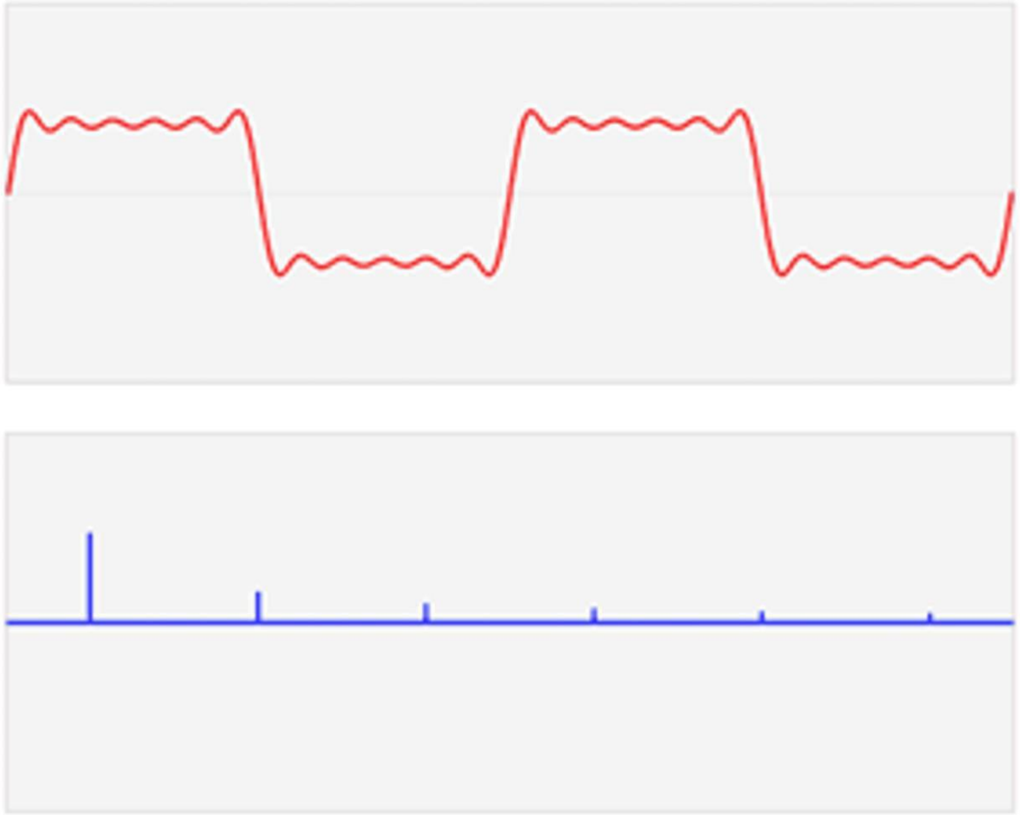
$f(x) = f_1(x) + 0.75\, f_2(x) + 0.5\, f_4(x)$



Representing signals as a superposition of frequencies



Representing signals as a superposition of frequencies

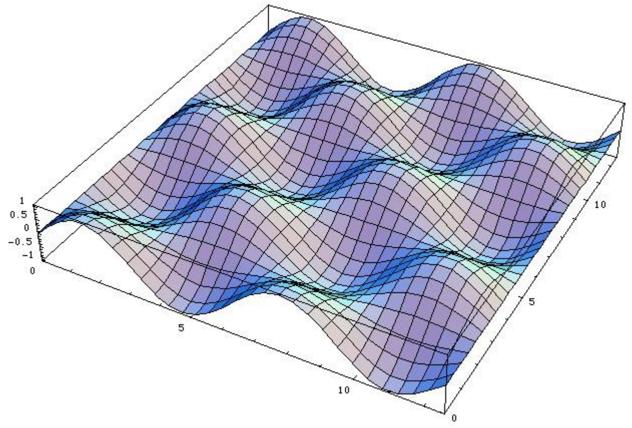


Spatial domain

Frequency domain

It’s exactly the same function!

Representing images (2D signals) as superposition of frequencies

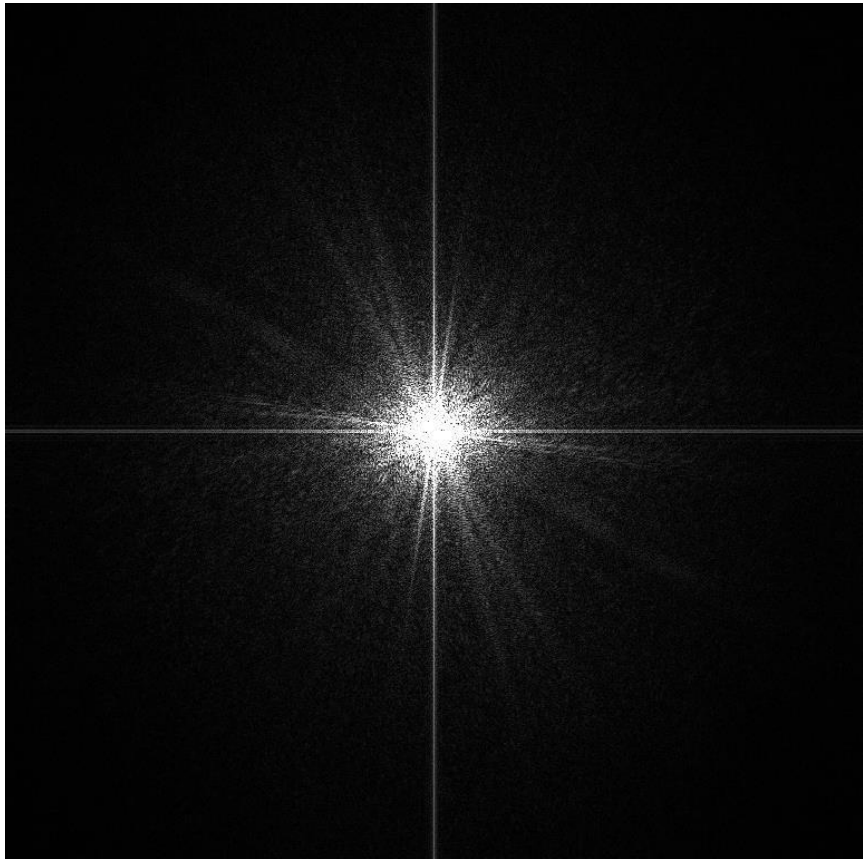


individual frequencies are 2D sinusoids

***(e.g.* f(x, y) = sin(aπx)* sin(bπx))**

Visualizing the frequency content of images

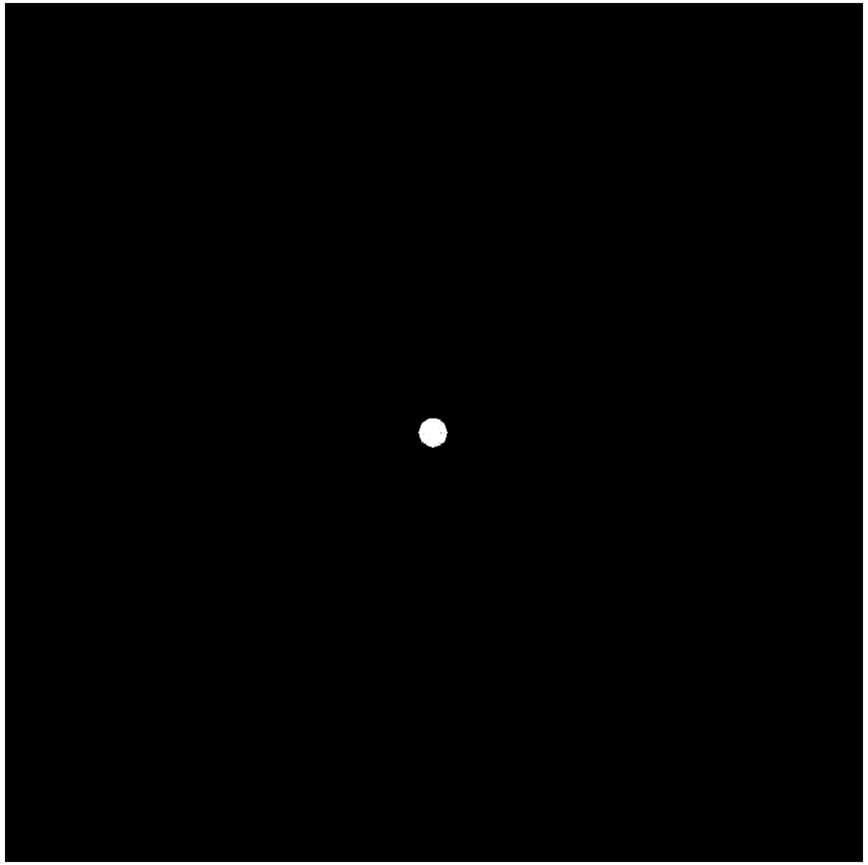
Spatial domain image



Frequency Domain Image

Low frequencies only

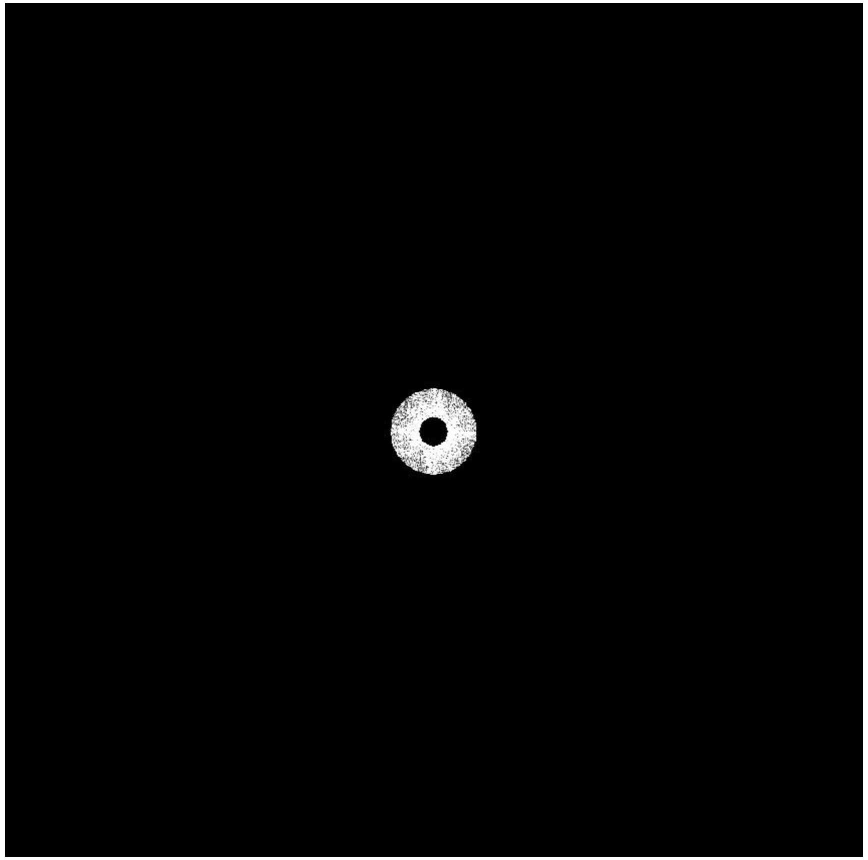
Spatial domain result



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

Mid-range frequencies

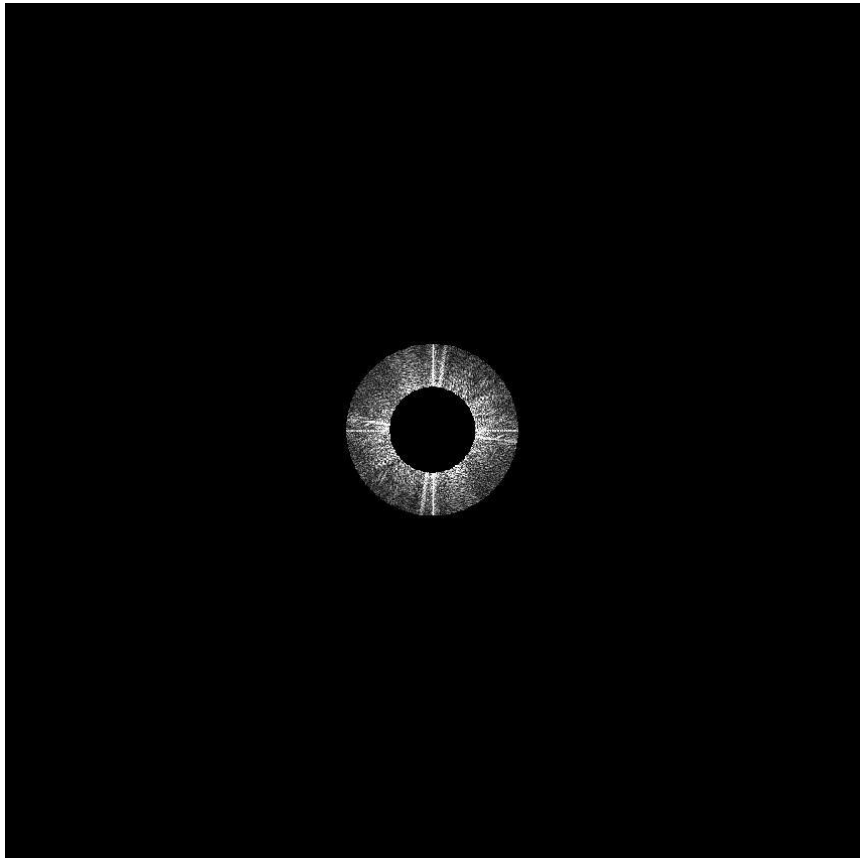
Spatial domain result



Spectrum (after band-pass filter)

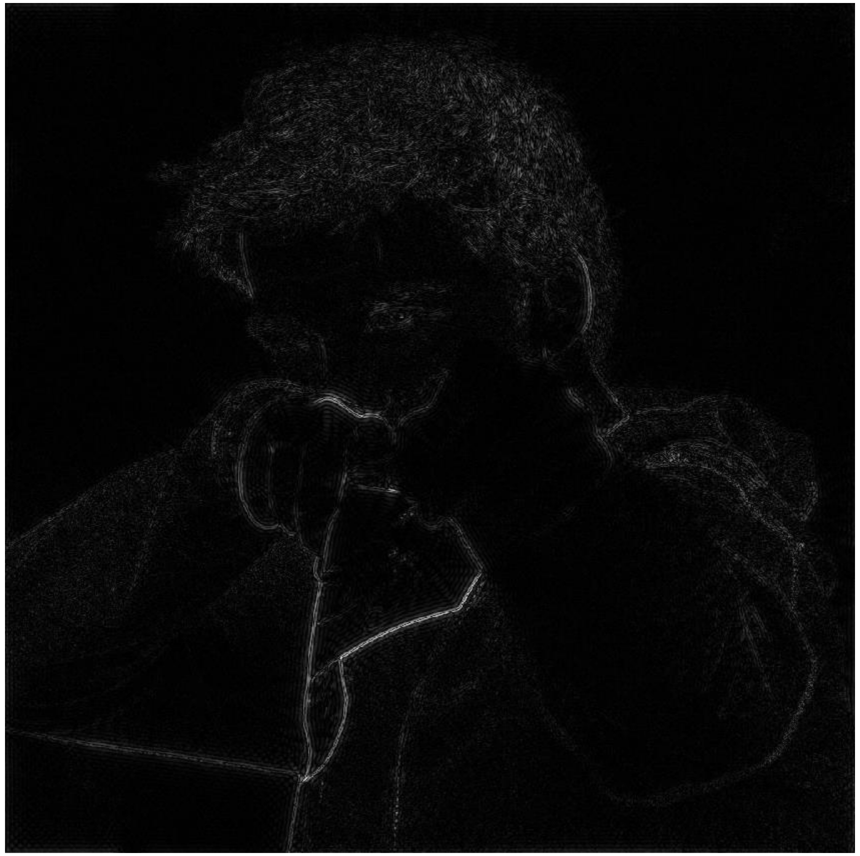
Mid-range frequencies

Spatial domain result

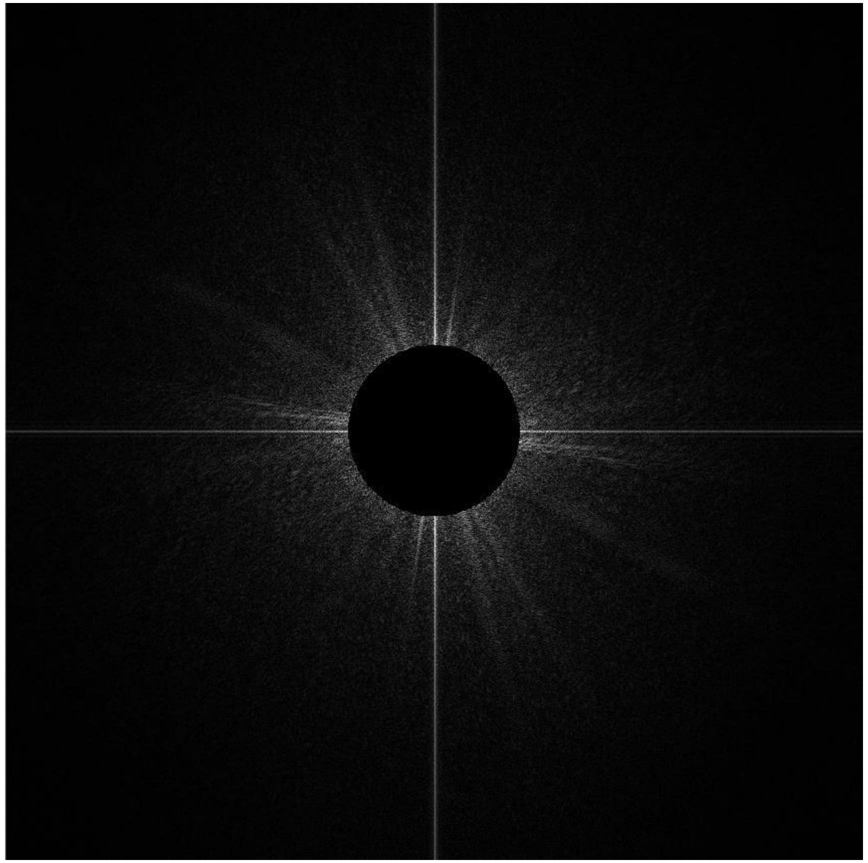


Spectrum (after band-pass filter)

High frequencies (edges)



Spatial domain result
(strongest edges)

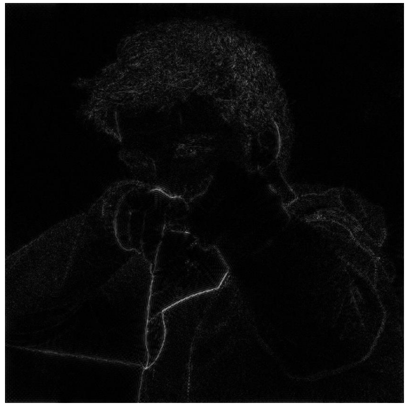


Spectrum (after high-pass filter)
All frequencies below threshold
have 0 magnitude

An image as a sum of its frequency components

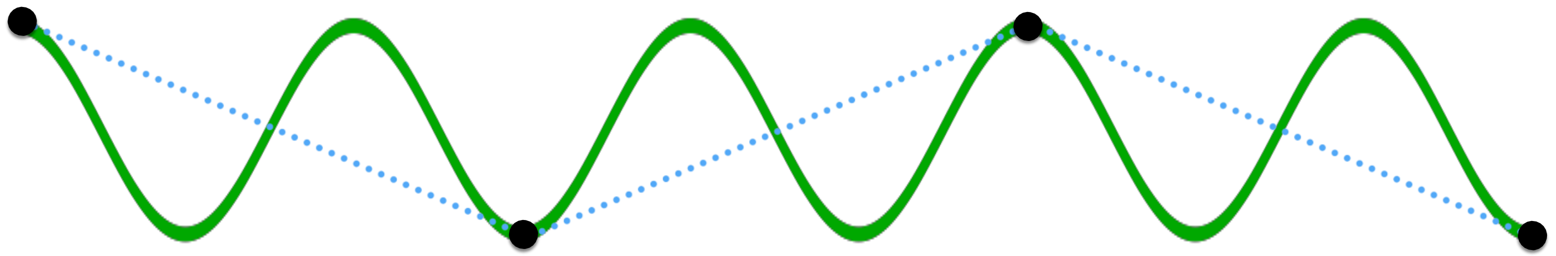

+

+

+


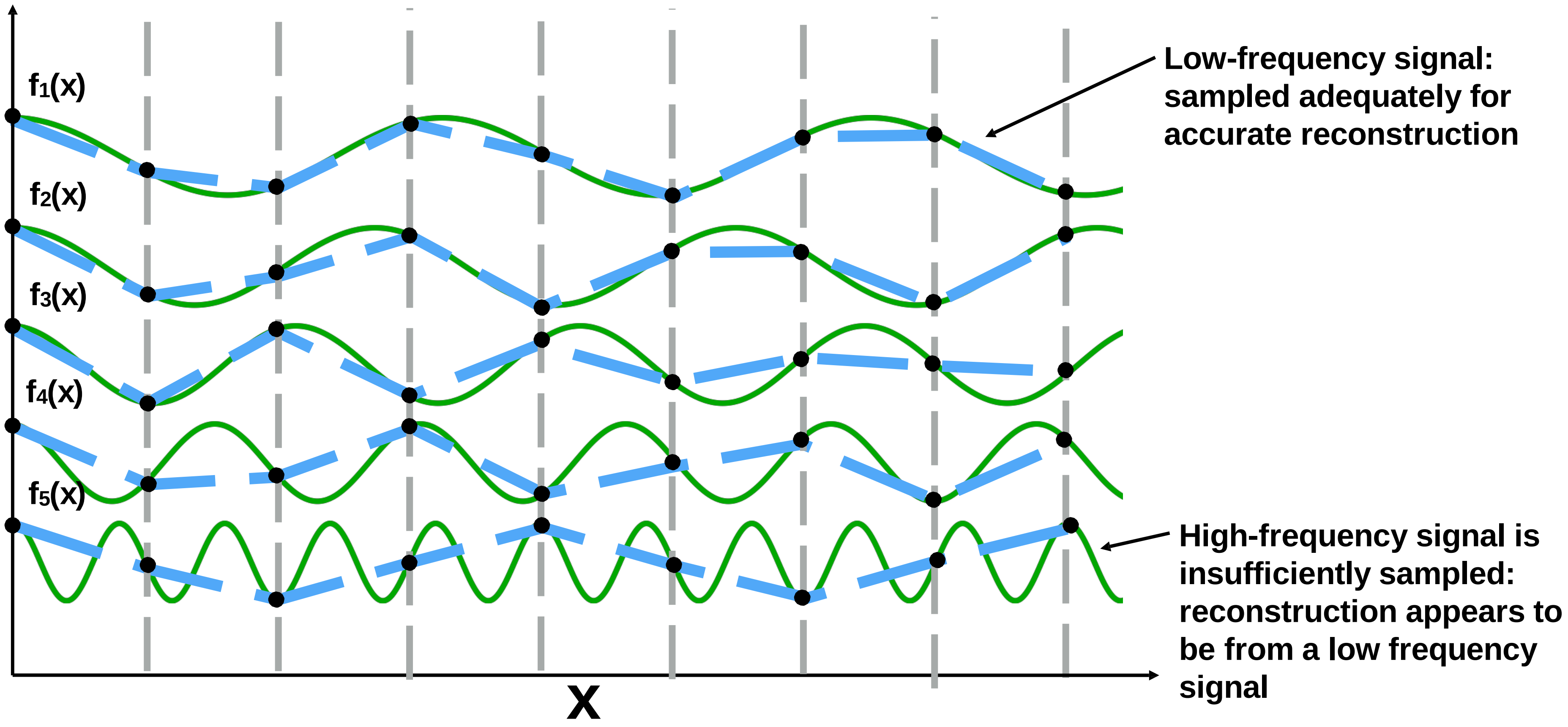
=


Back to 1D example: Sampling rate, high- frequency signals & aliasing



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

Back to 1D example: Sampling rate, high- frequency signals & aliasing



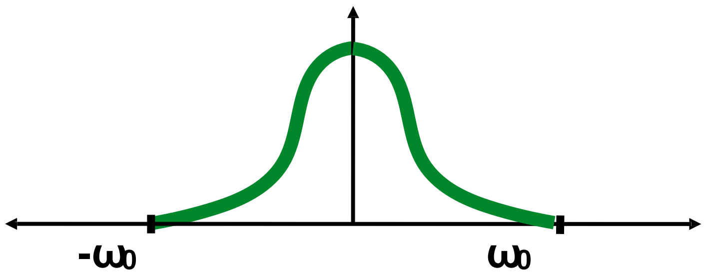
"Aliasing": high frequencies in the original signal masquerade as low frequencies after reconstruction (due to undersampling)

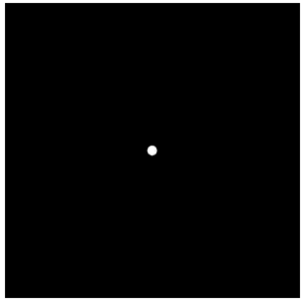
Sampling rate, high-frequency signals & aliasing

- **So, how densely should you be sampling?**

Nyquist-Shannon theorem

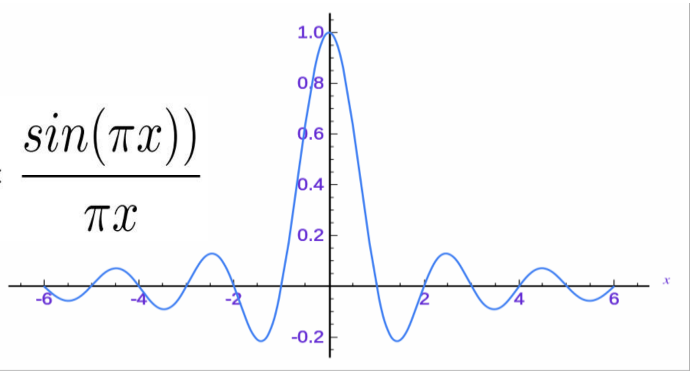
- Consider a band-limited signal: has no frequencies above ω_0
  - 1D: consider low-pass filtered audio signal
  - 2D: recall the blurred image example from a few slides ago





- The signal can be perfectly reconstructed if sampled with frequency f_s > $2\omega_0$
- And reconstruction is performed using a normalized sinc (ideal reconstruction filter with infinite extent)

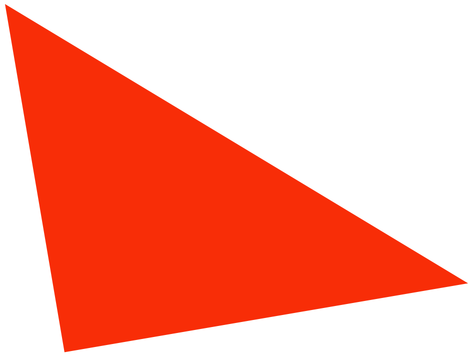
$$sinc(x) = \frac{sin(\pi x))}{\pi x}$$



Challenges of sampling-based approaches in graphics

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

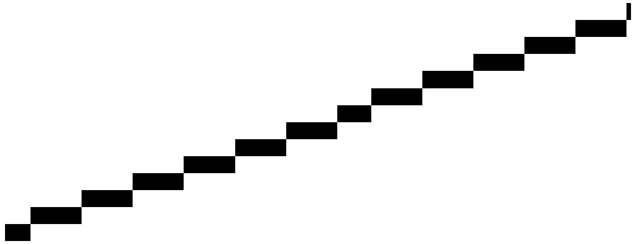
Hint:

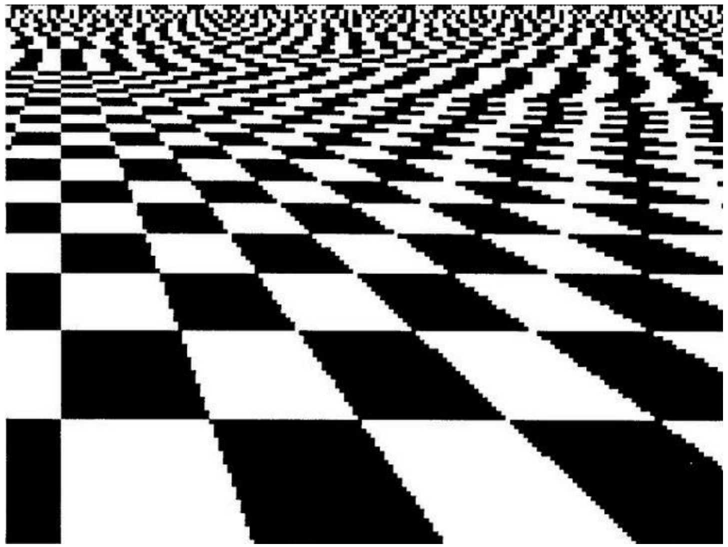


- **Also, infinite extent of “ideal” reconstruction filter (sinc) is impractical for performant implementations. Why?**

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**

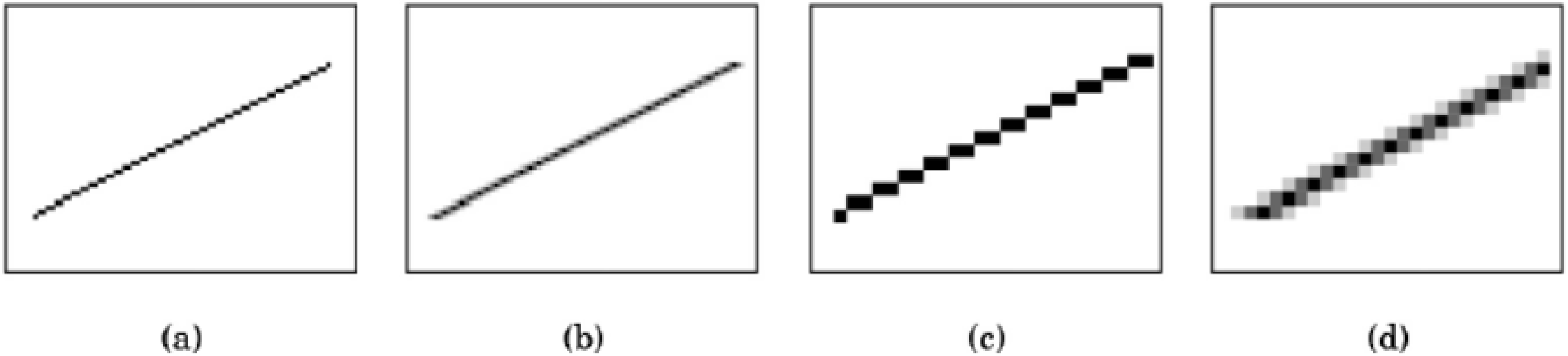




Aliasing: sample a continues image at grid points

Moiré patterns

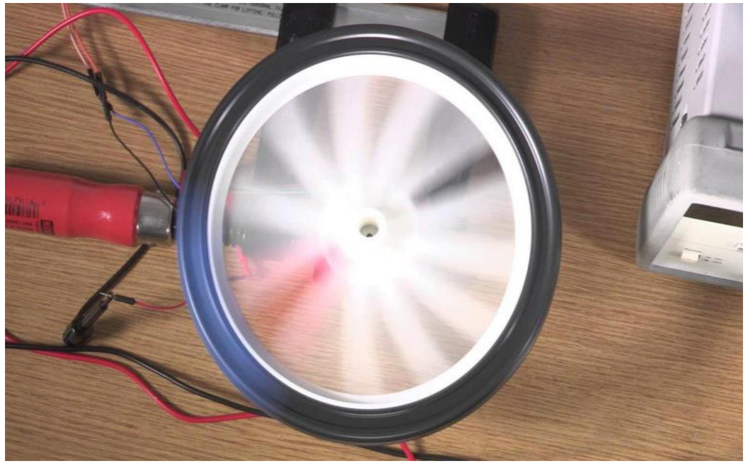
Antialiasing for Line Segments



- (c) is aliased, magnified
- (d) is antialiased, magnified

Temporal Aliasing

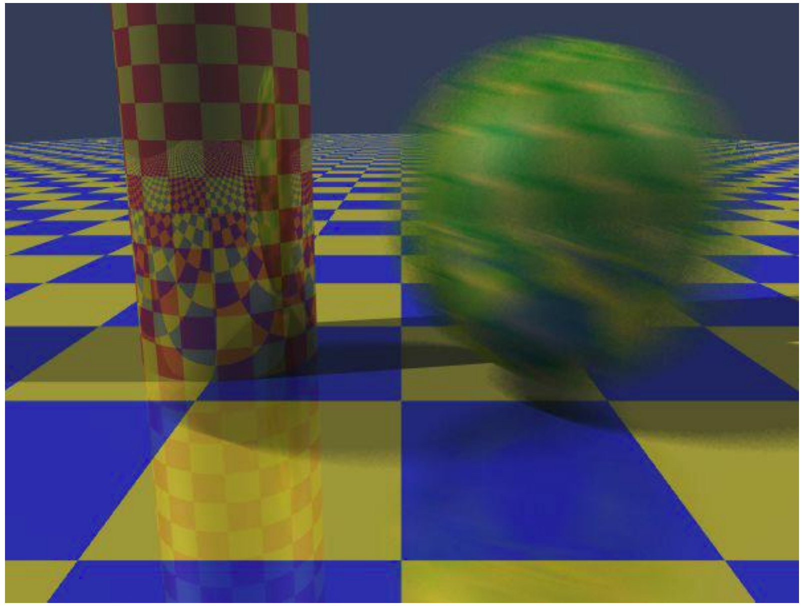
- Sampling rate is frame rate (30 Hz for video)
- Example: spokes of wagon wheel in movies



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

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

- Solution: supersample in time and average
  - Fast-moving objects are blurred
  - Happens automatically with real hardware (photo and video cameras)
  - Exposure time is important (shutter speed)
  - Effect is called **motion blur**



motion blur

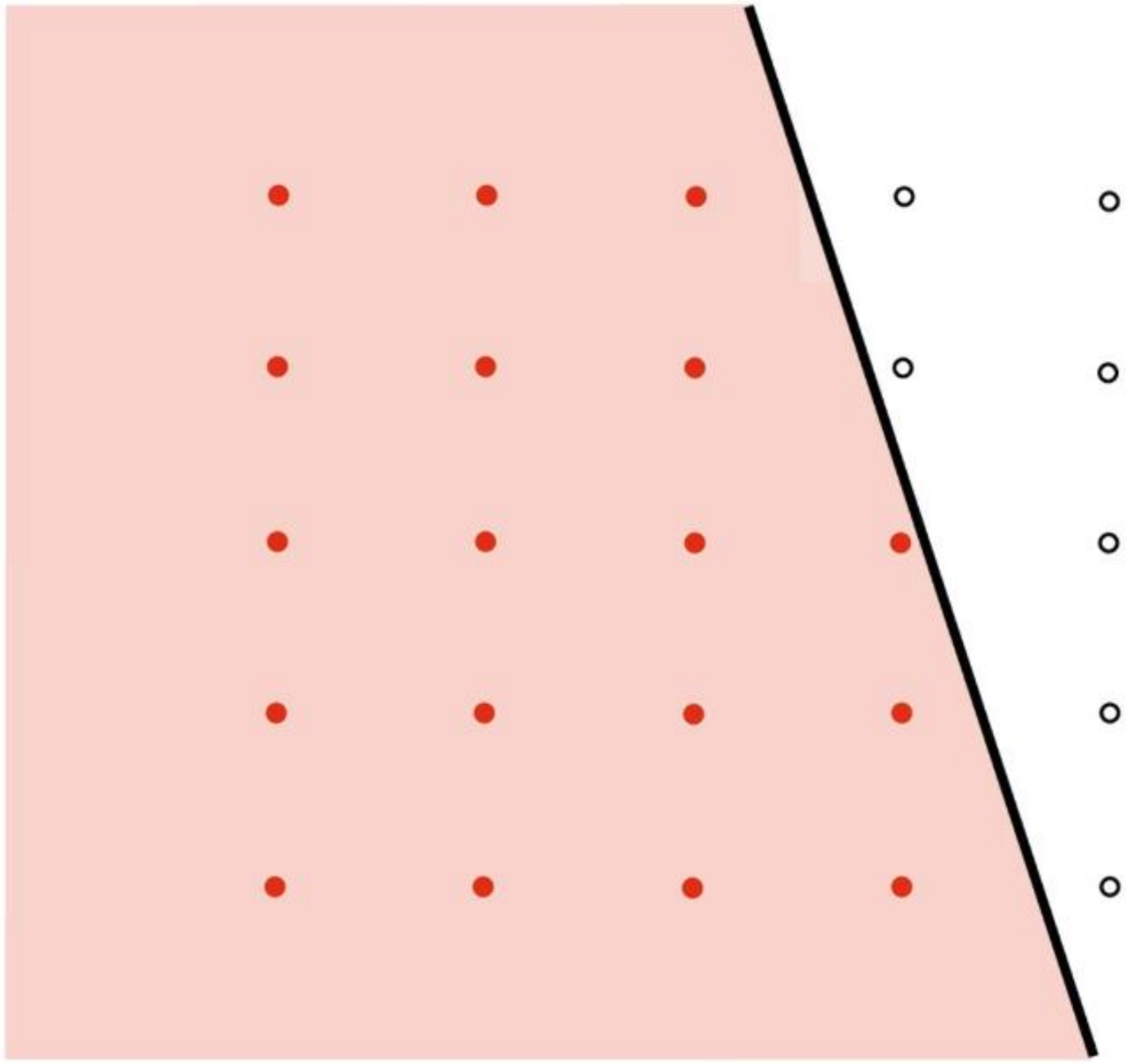
Motion Blur Example

- Achieved by stochastic sampling in time

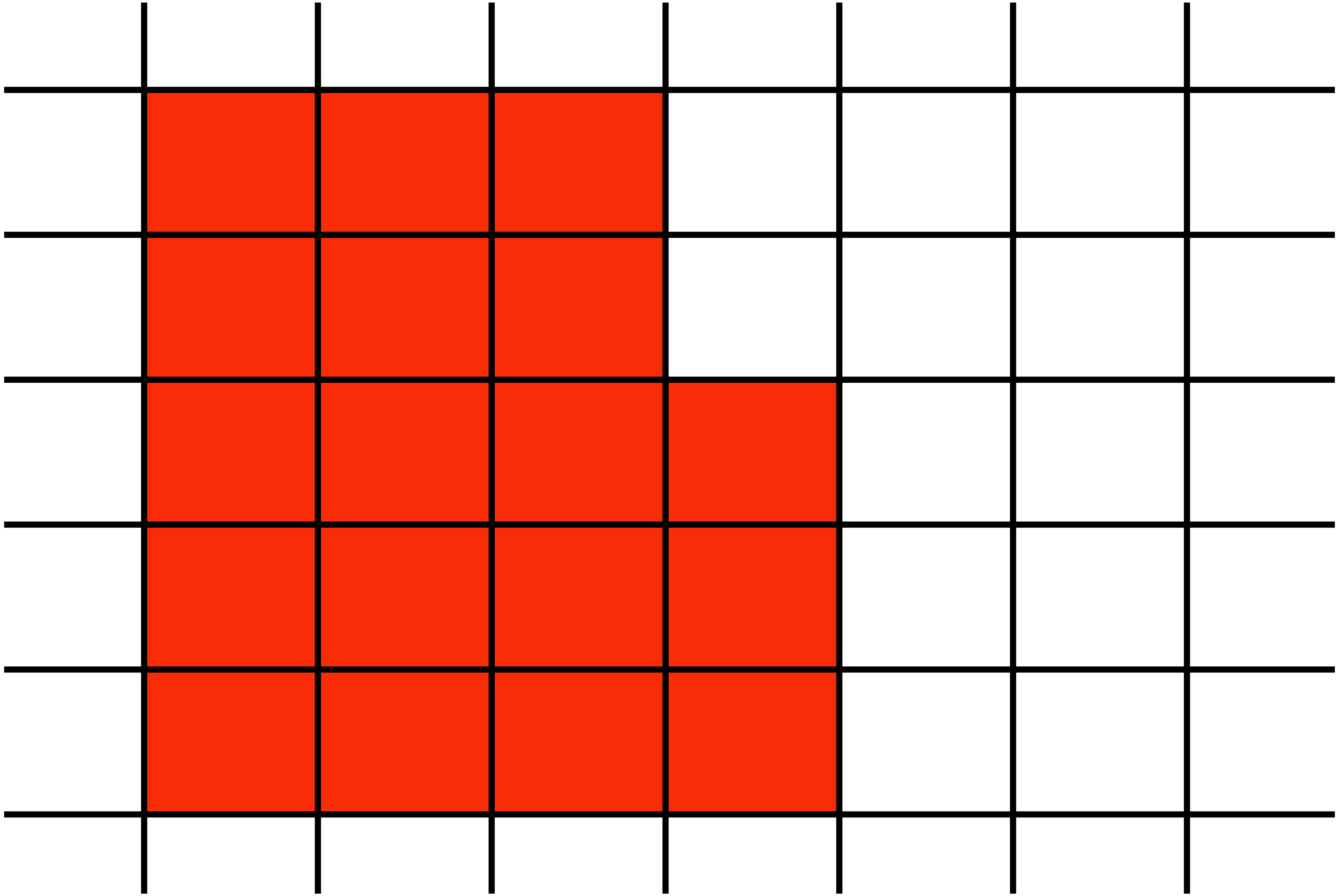
T. Porter, Pixar, 1984 16 samples / pixel / timestep

Recall: the real coverage signal

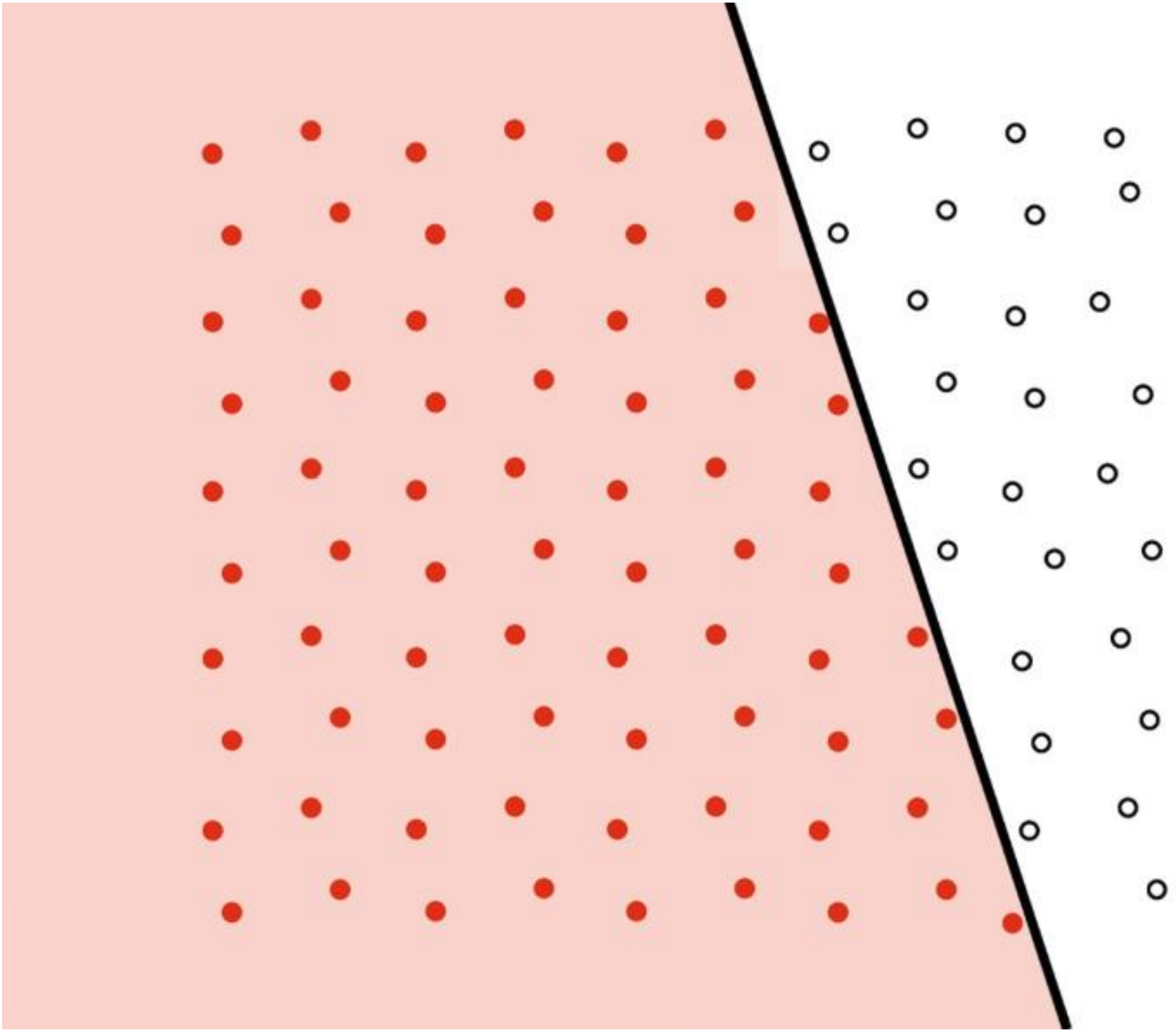
Initial coverage sampling rate (1 sample per pixel)



We see this on the screen

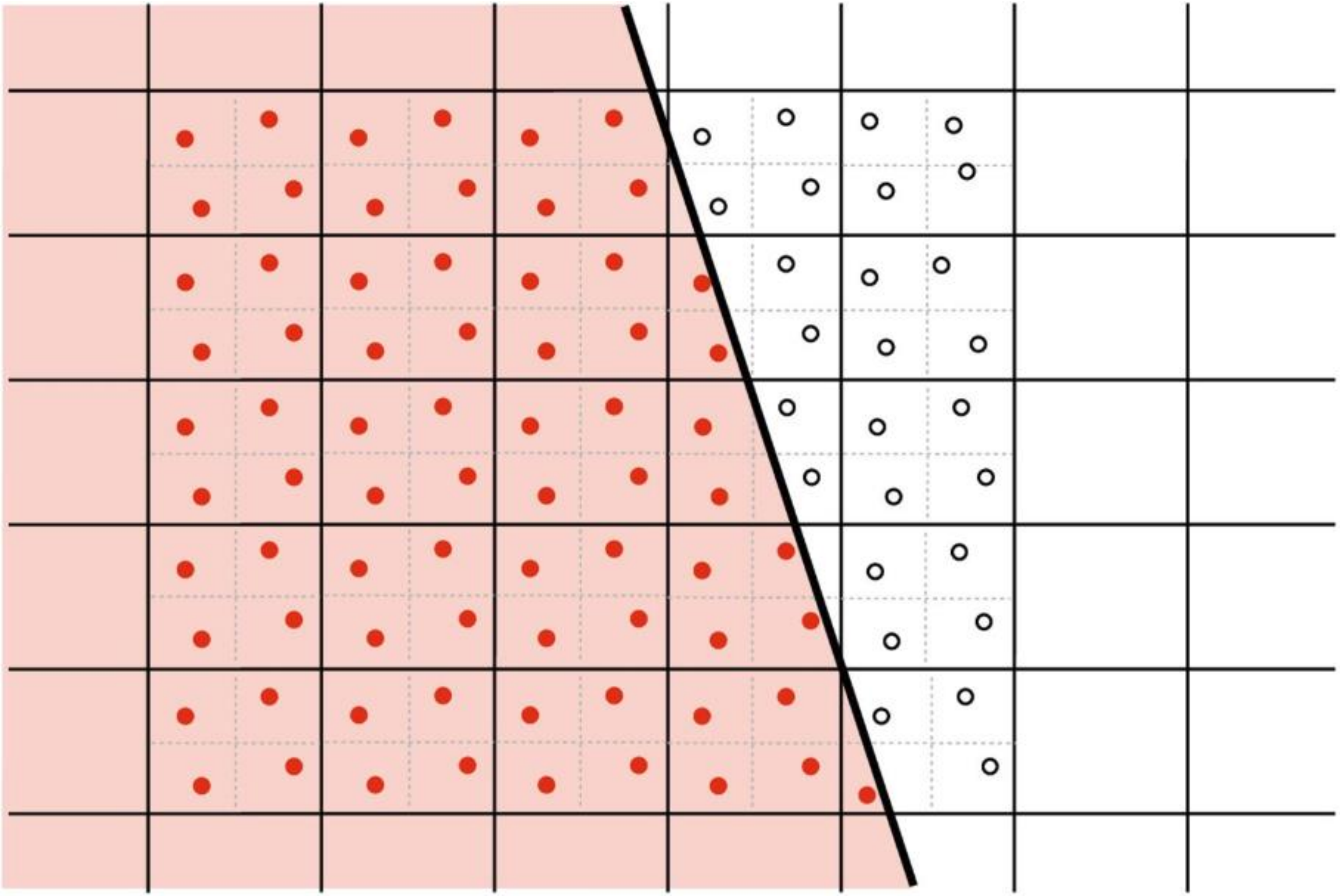


Increase density of sampling coverage signal
(high frequencies exist in original signal because of triangle edges)



Supersampling

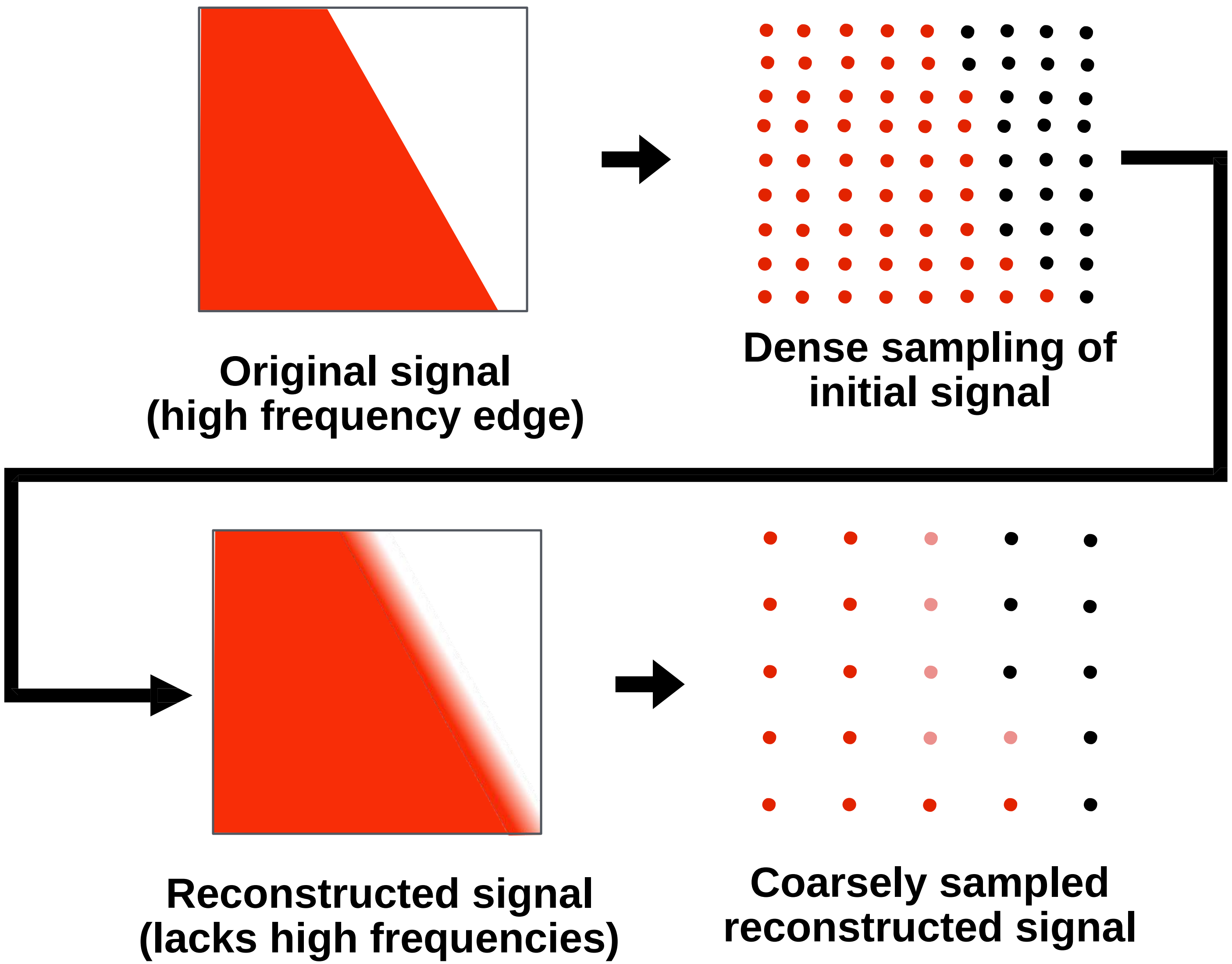
Example: stratified sampling using four samples per pixel



Ok, but now we have more samples than pixels!

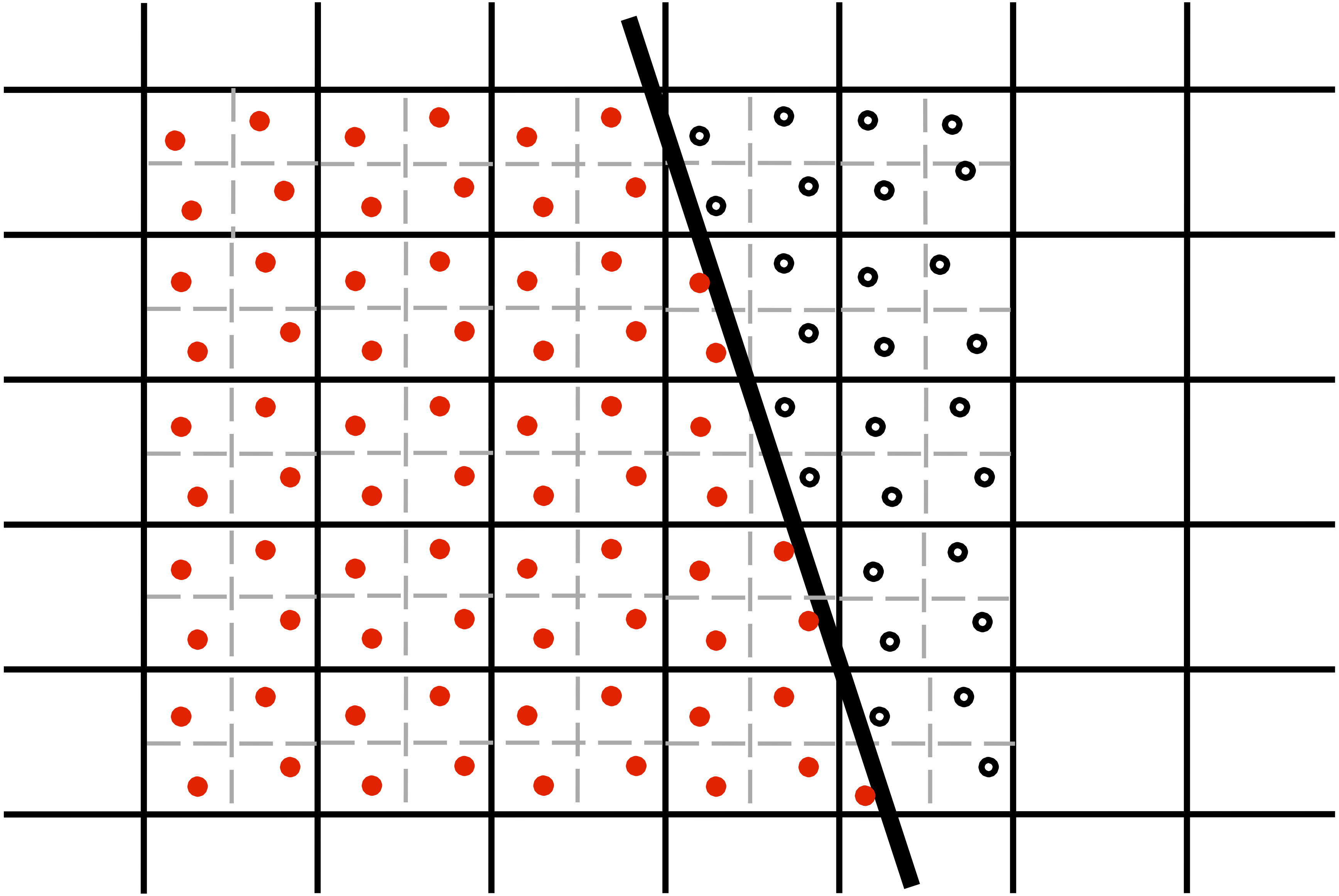
Resampling

Converting from one discrete sampled representation to another



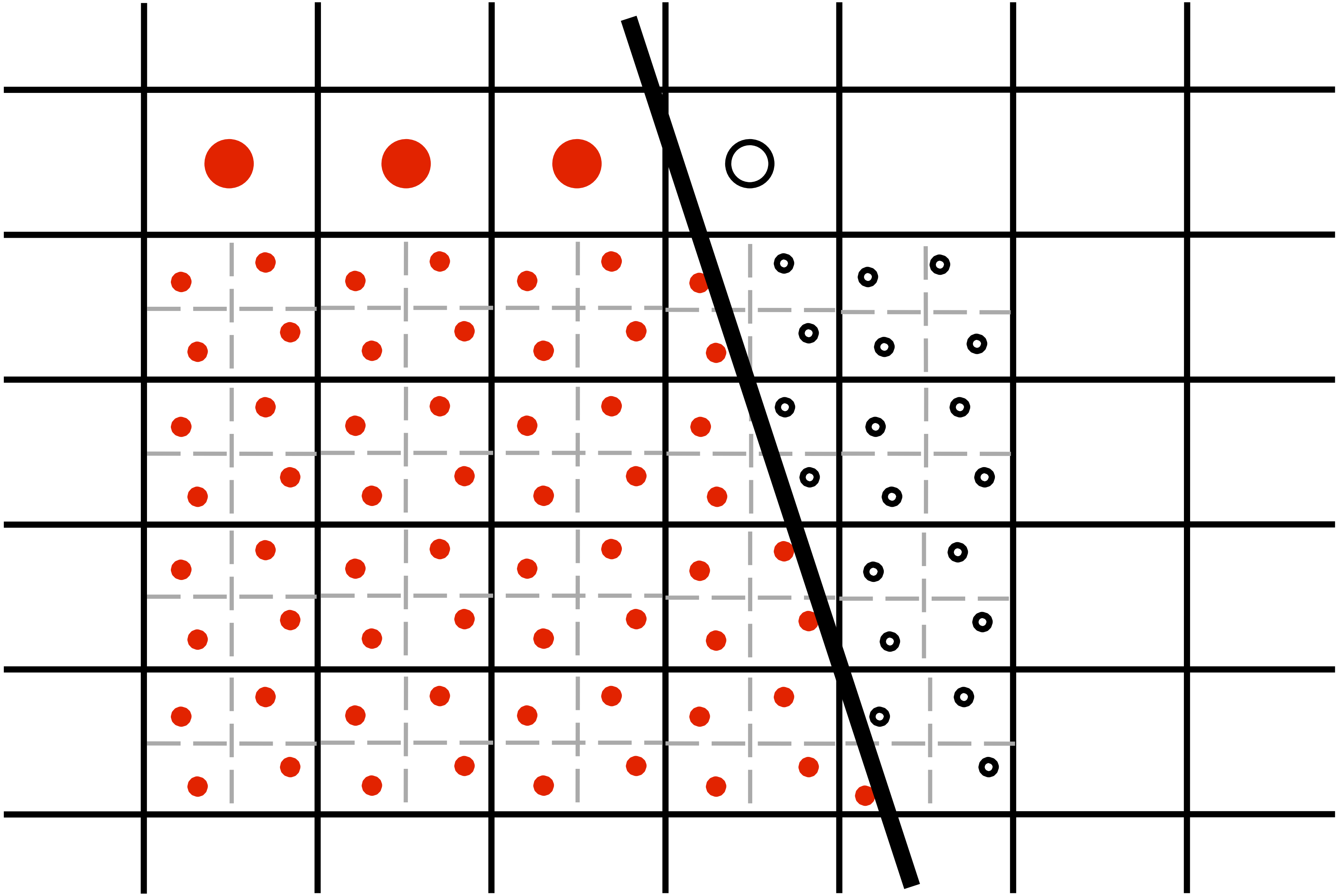
Resample to display's pixel resolution

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

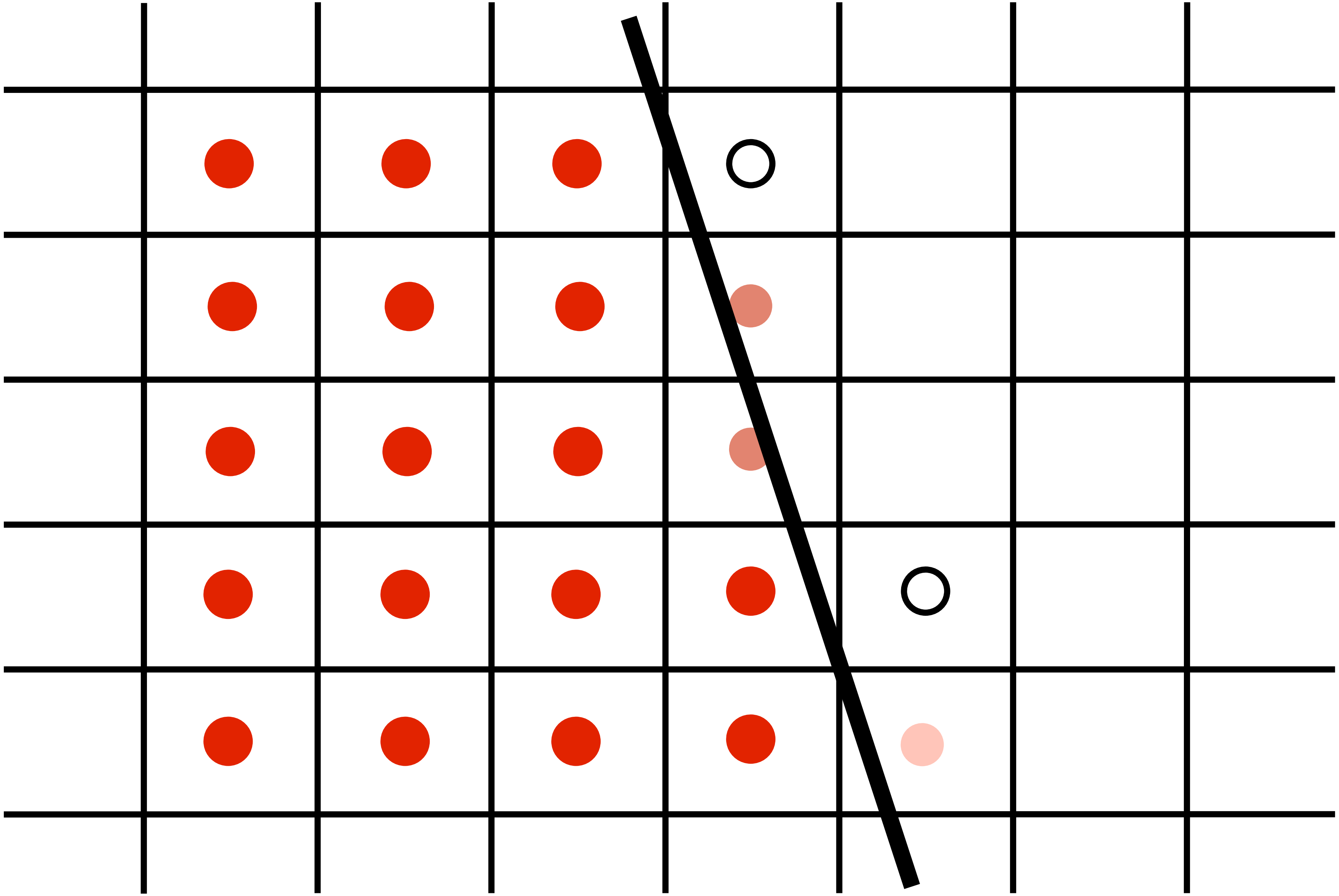


Resample to display's pixel resolution

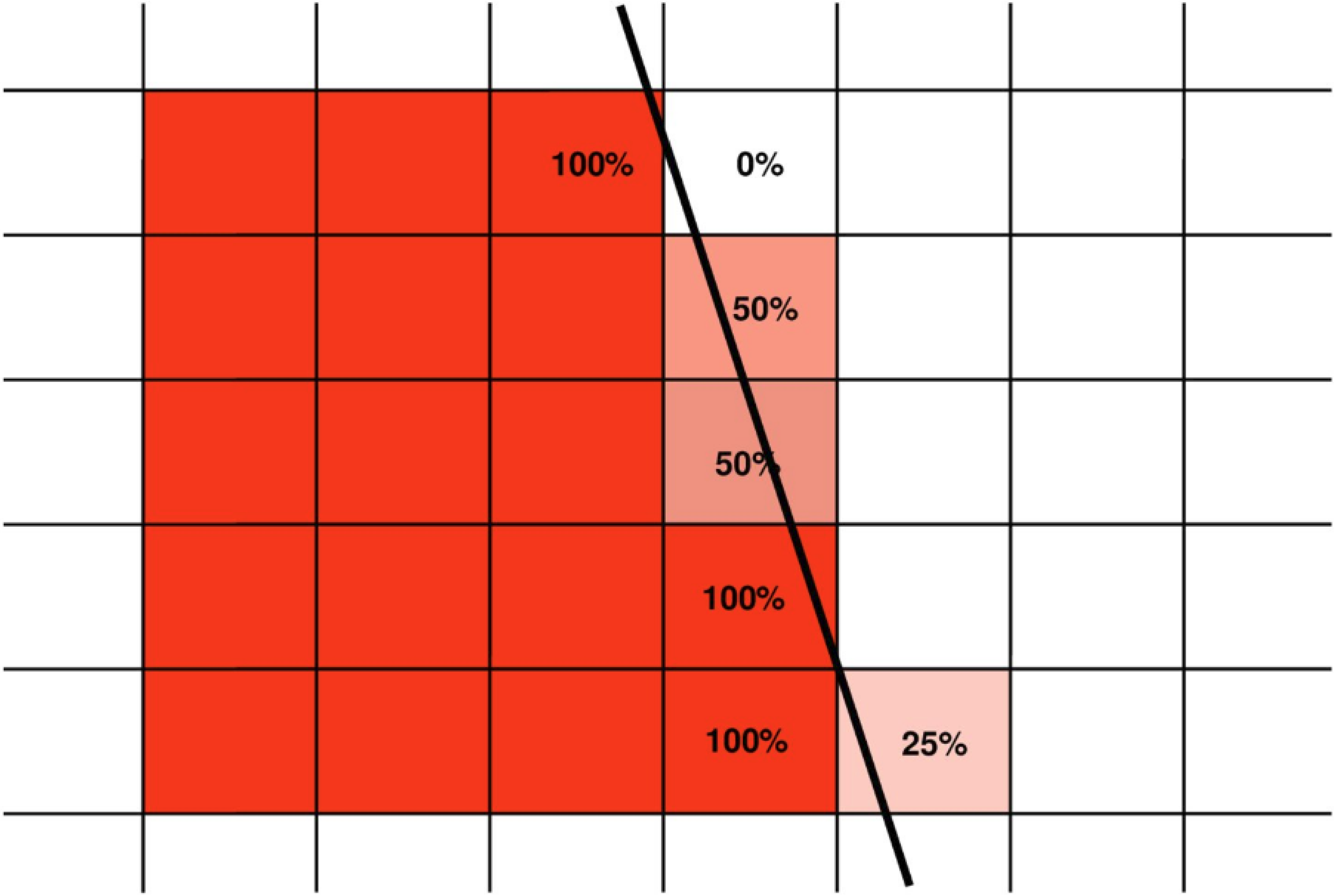
(Because a screen displays one sample value per screen pixel...)



Resample to display's pixel resolution

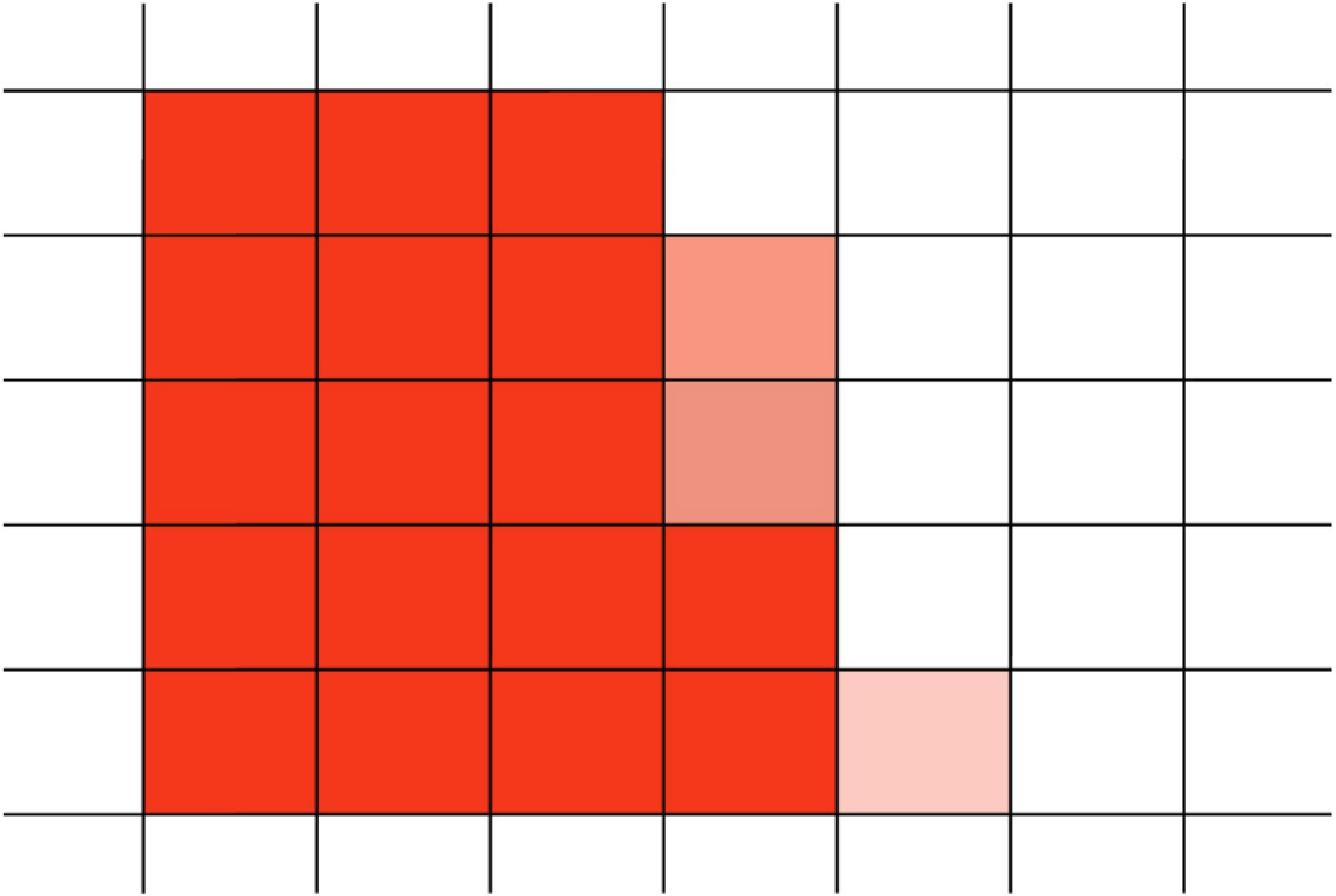


Displayed result (note anti-aliased edges)



Recall: the real coverage signal

Displayed result (note anti-aliased edges)



Pretty much as well as we can do without an “infinite resolution display”

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

Point-in-triangle test

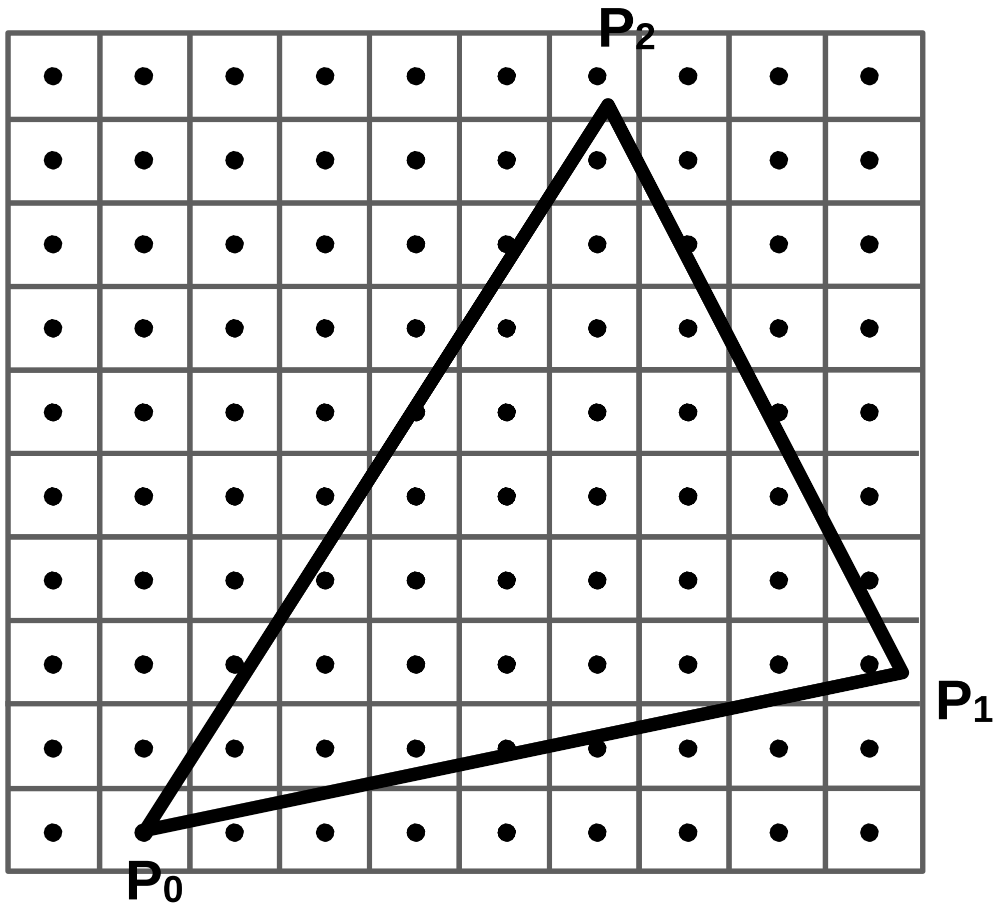
Compute triangle edge equations from projected positions of vertices

$P_i = (X_i, Y_i)$

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

$$\begin{aligned} E_i(x, y) &= (x - X_i)\, dY_i - (y - Y_i)\, dX_i \\ &= A_i\, x + B_i\, y + C_i \end{aligned}$$

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



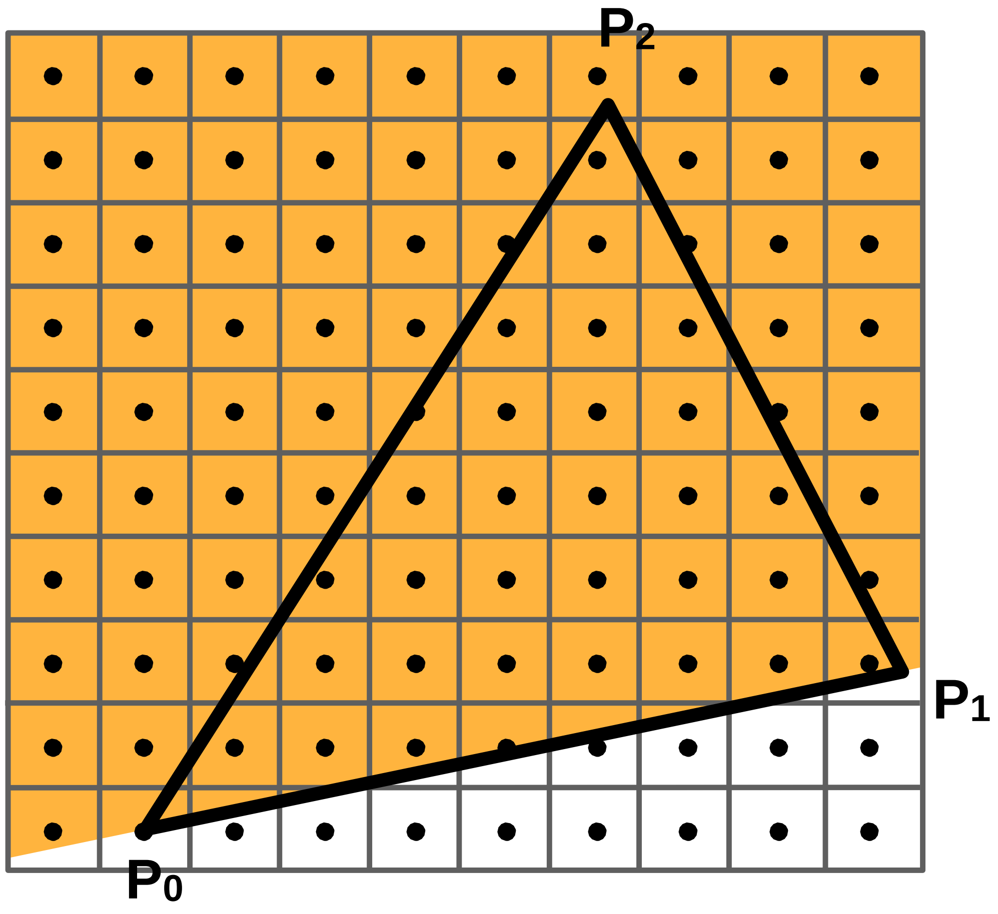
Point-in-triangle test

$P_i = (X_i, Y_i)$

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

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

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



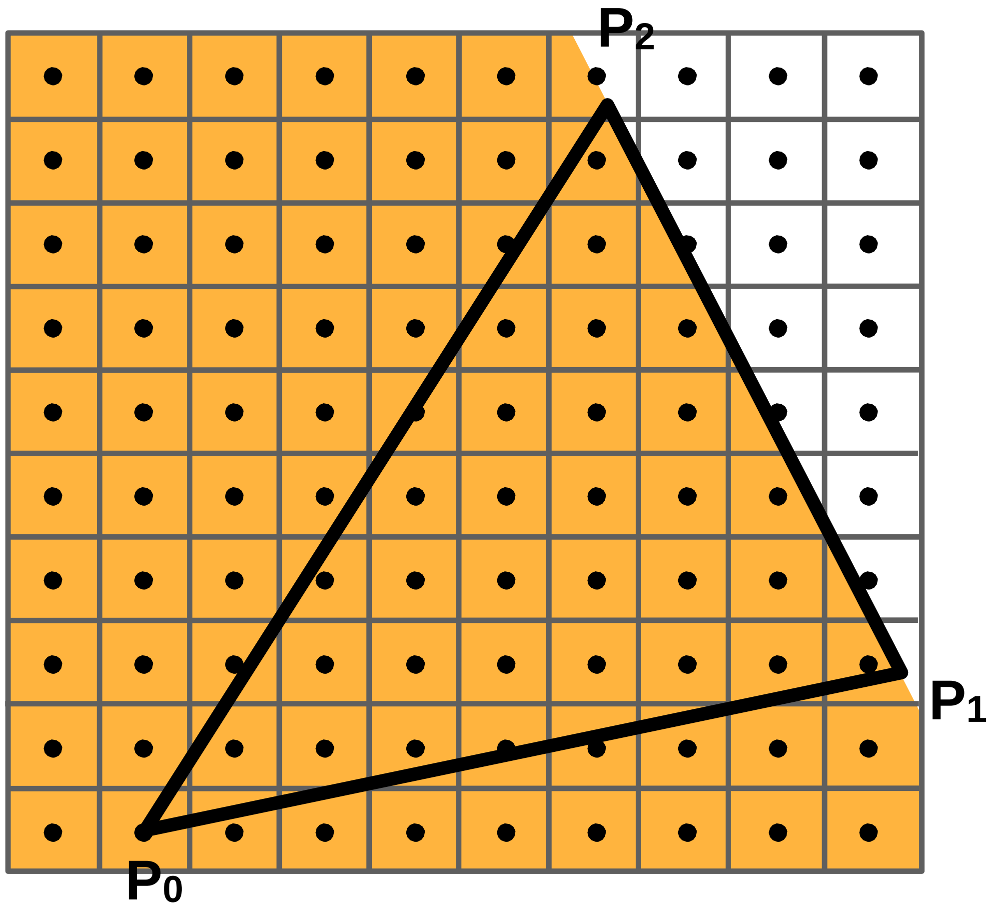
Point-in-triangle test

$P_i = (X_i, Y_i)$

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

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

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



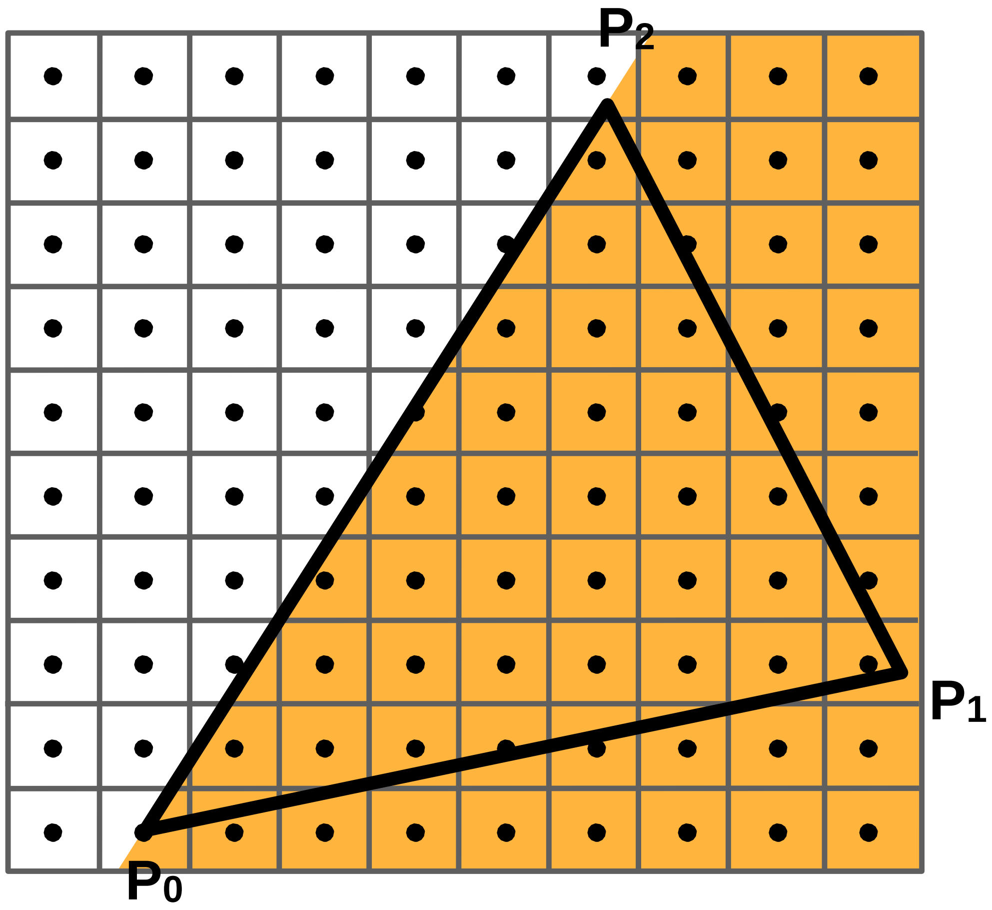
Point-in-triangle test

$P_i = (X_i, Y_i)$

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

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

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

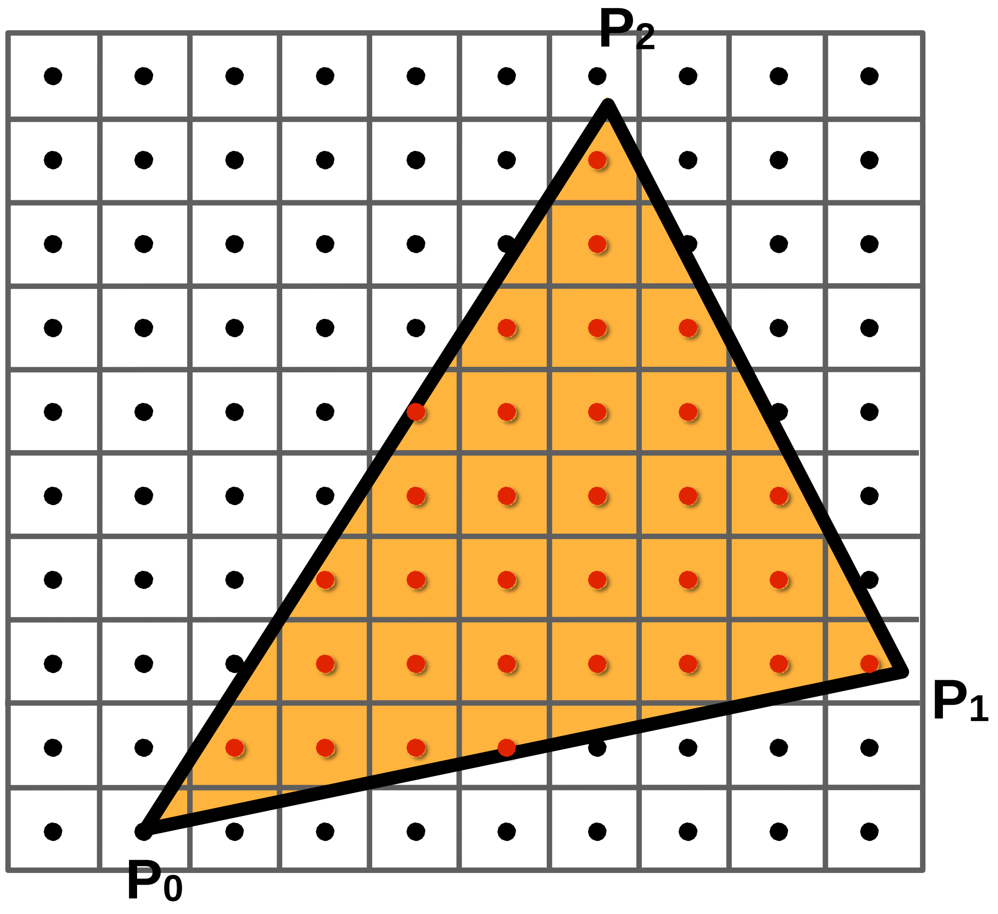


Point-in-triangle test

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 earlier slides)

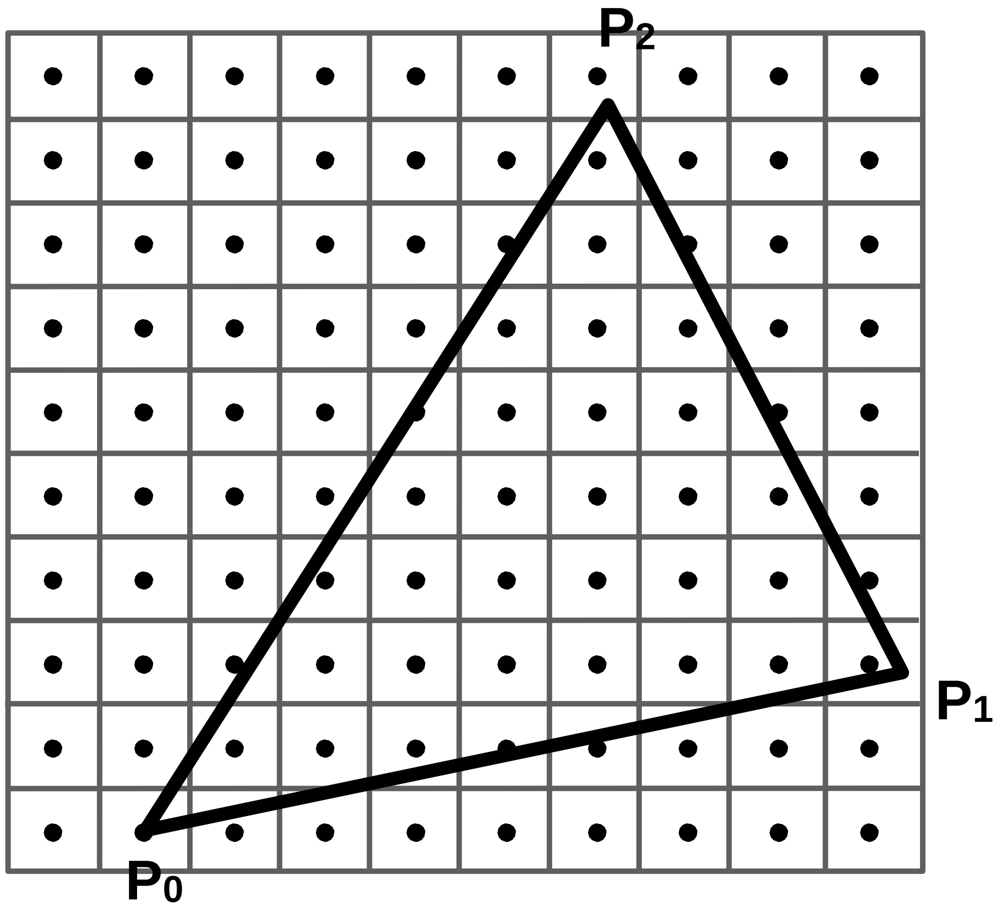


Sample points inside triangle are highlighted red.

Point-in-triangle test

Which points should we test?

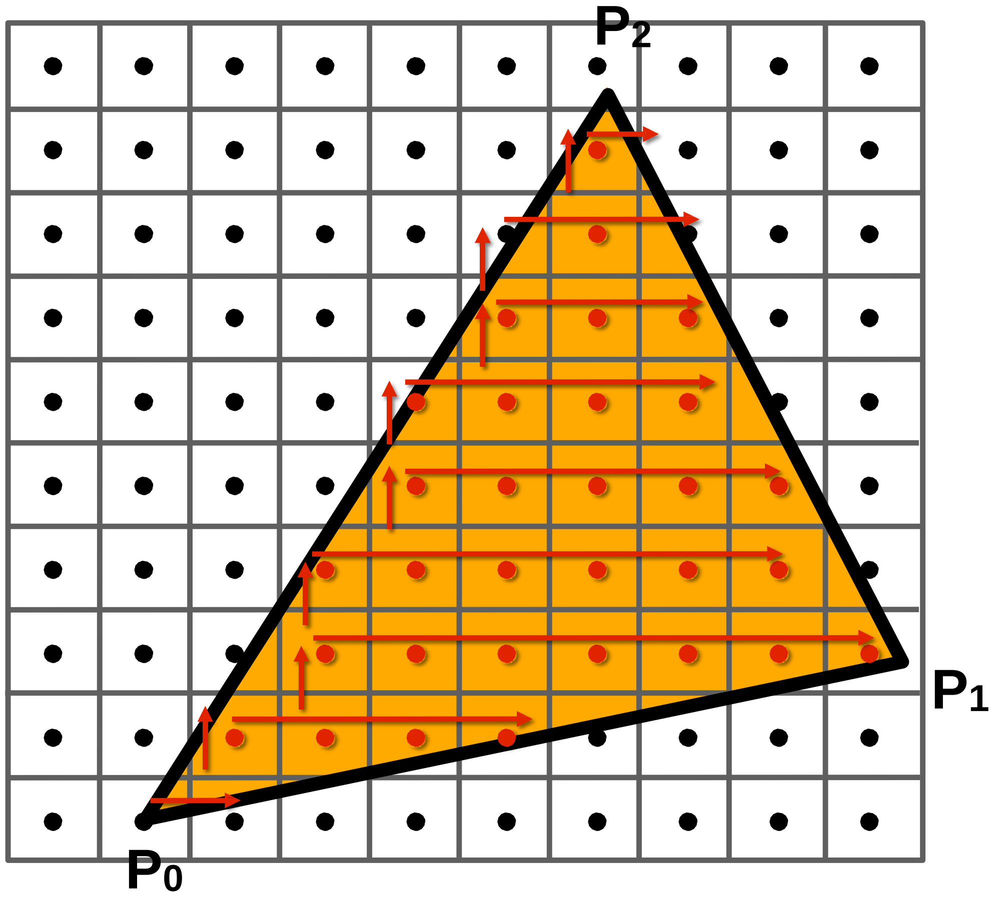
- **All of them?**
- **Points within bounding box?**



Incremental triangle traversal

Rather than testing all points on screen, traverse them incrementally

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



Modern approach: tiled triangle traversal

Traverse triangle in blocks

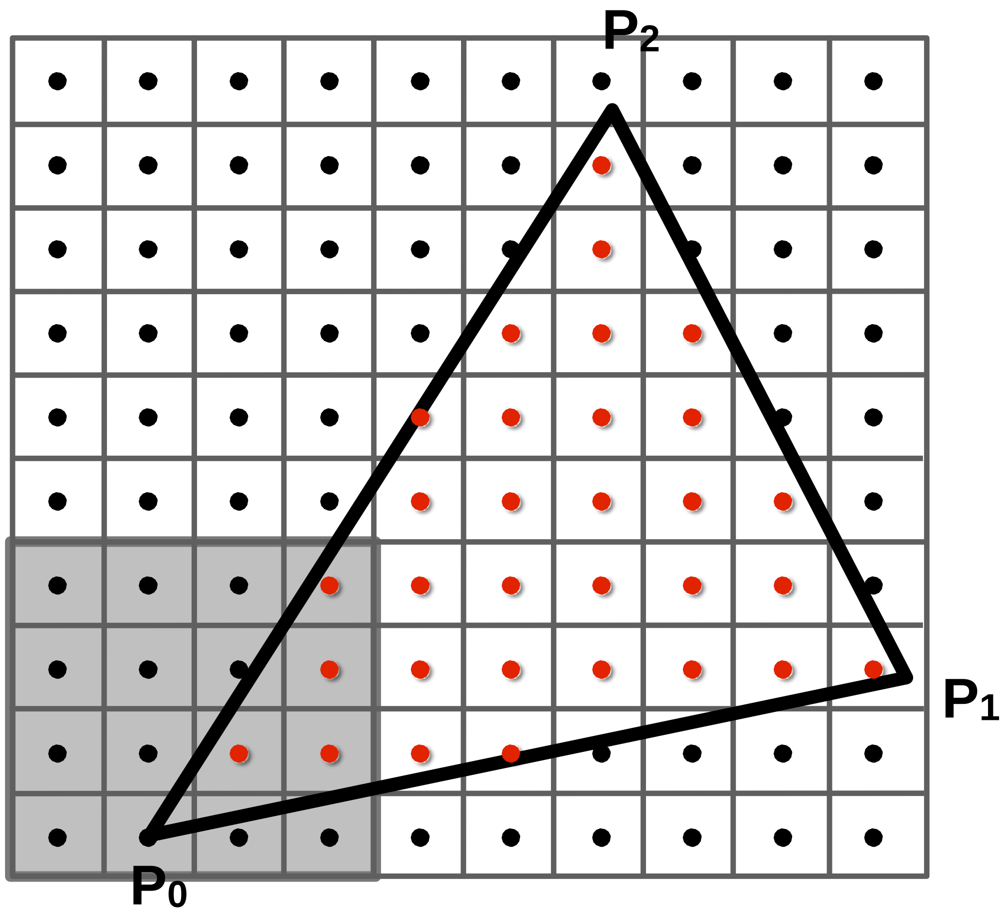
Test all samples in block against triangle in parallel

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 advantages 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