# Drawing a Triangle (and an introduction to sampling)

Computer Graphics
CMU 15-462/15-662, Fall 2016

# Announcements

- OpenGL tutorial, today, 5:00pm-6:30pm in GHC 6115
- Assignment 1 will be out tomorrow
- And now, onto drawing triangles…

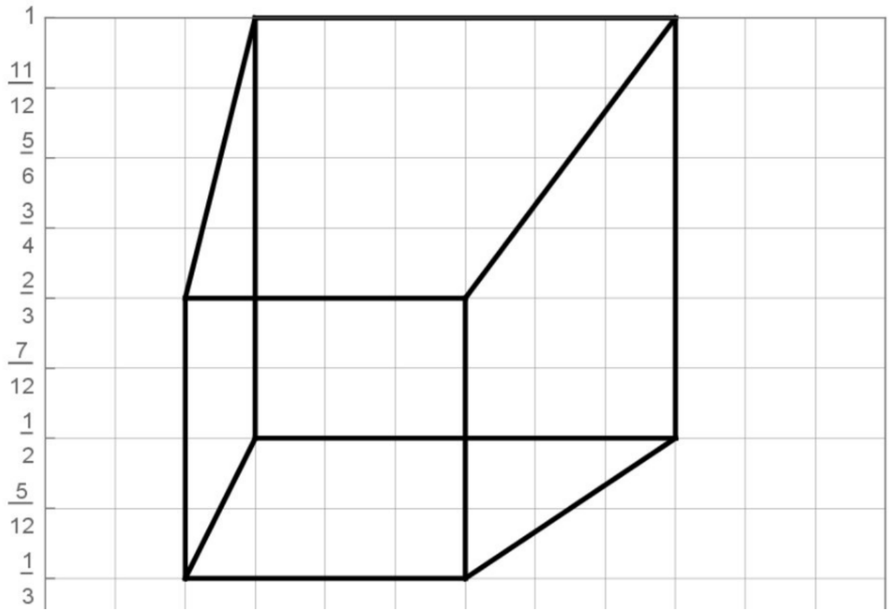
# Why should we care about triangles?


https://www.youtube.com/watch?v=4kOvYQW0AJM

# Why should we care about triangles?

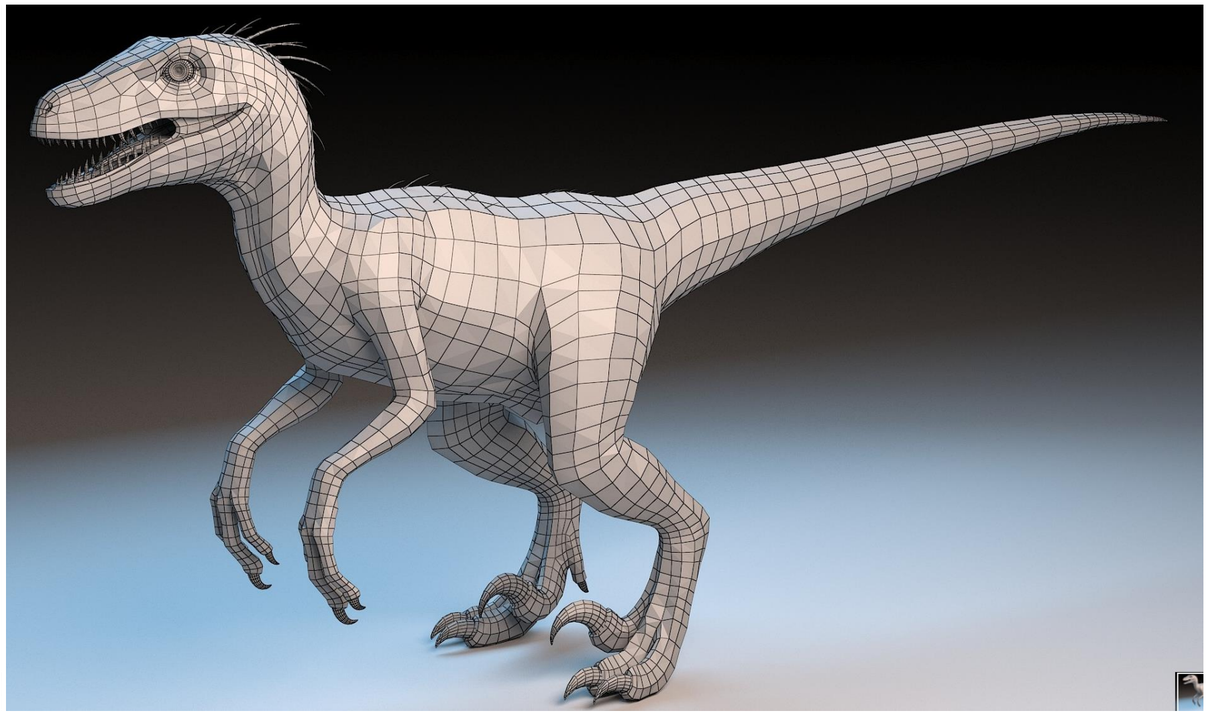

https://www.youtube.com/watch?v=4kOvYQW0AJM

# Why should we care about triangles?




**2D coordinates:**

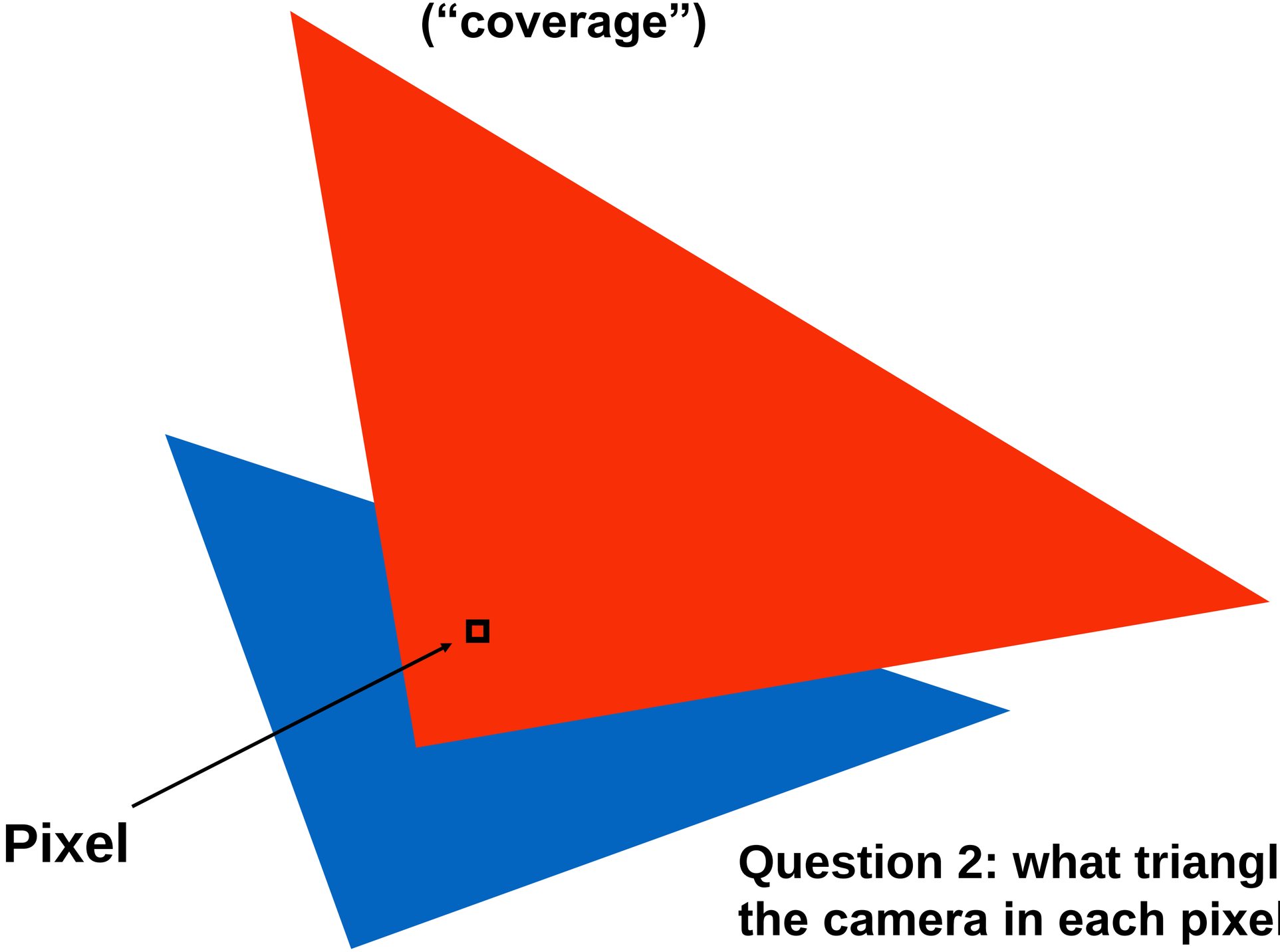
A: 1/4, 1/2
B: 3/4, 1/2
C: 1/4, 1
D: 3/4, 1
E: 1/6, 1/3
F: 1/2, 1/3
G: 1/6, 2/3
H: 1/2, 2/3



# If you know how to draw a triangle, you'll go a long way!

# Let’s draw some triangles on the screen!

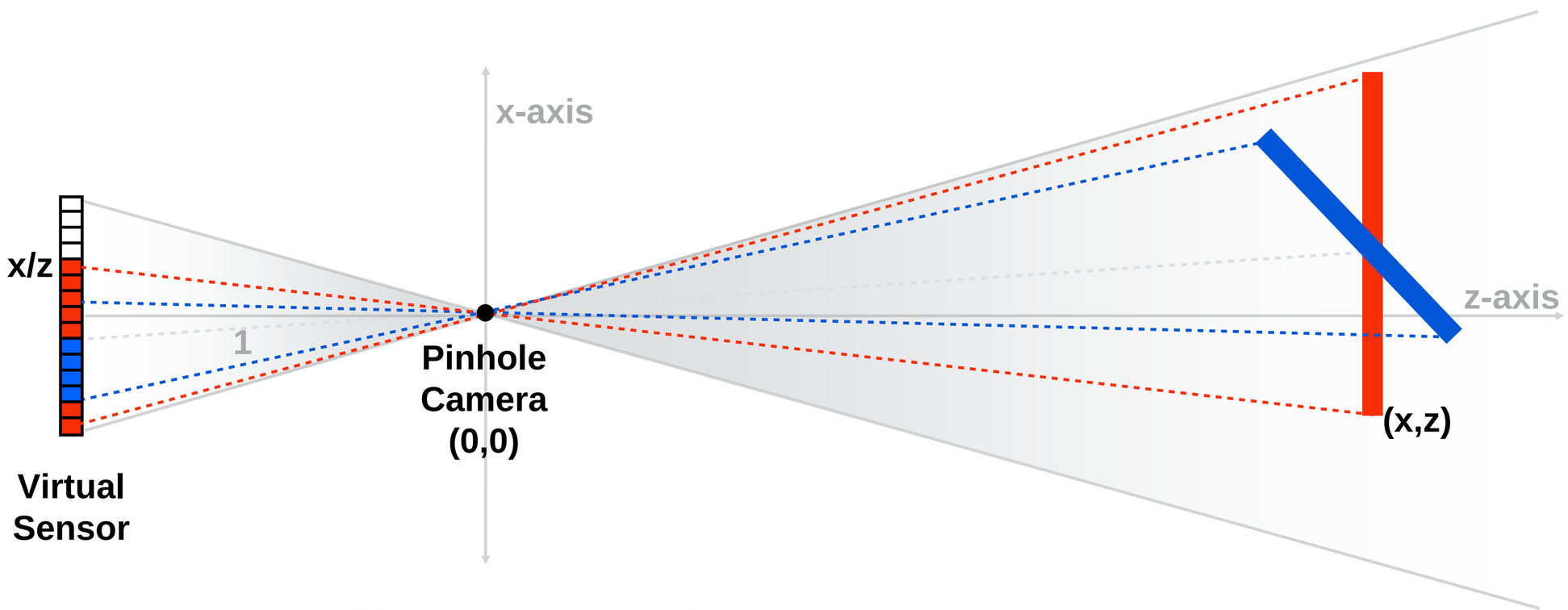
**Question 1: what pixels does the triangle overlap? (“coverage”)**




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

# The visibility problem

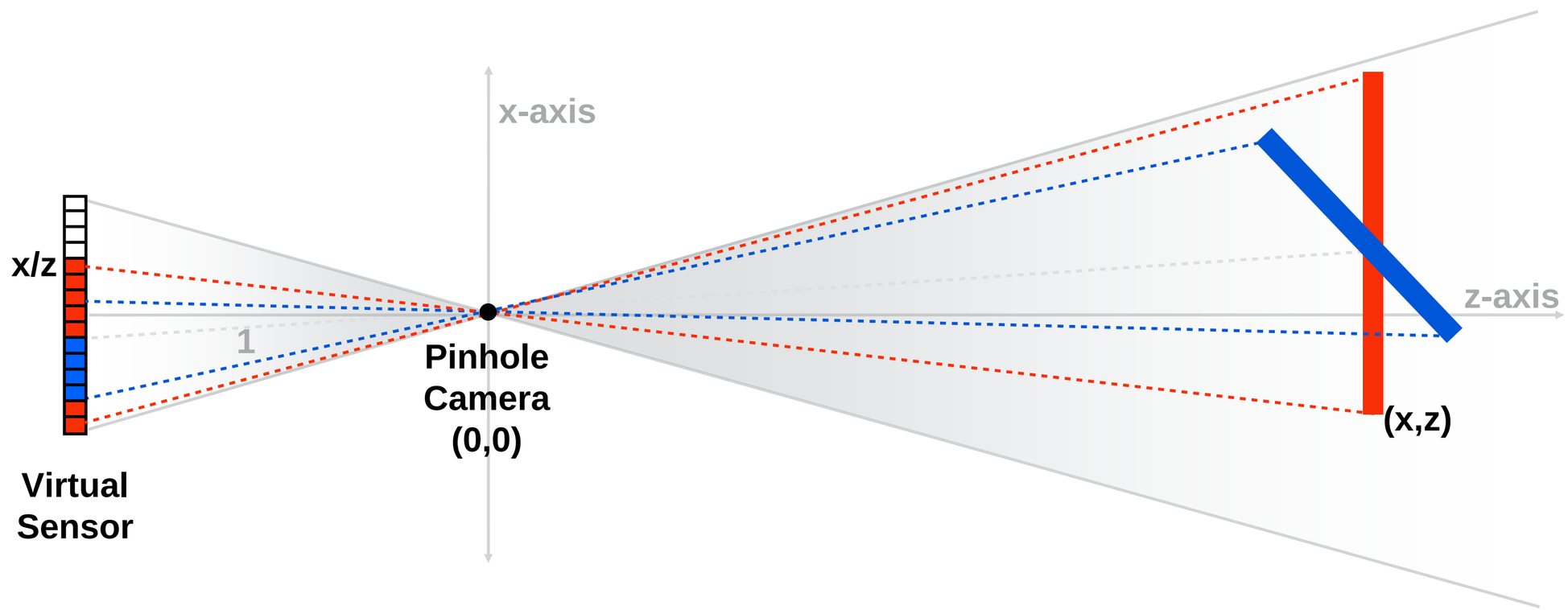
- **An informal definition: what scene geometry is visible within each screen pixel?**
  - **What scene geometry projects into a screen pixel? (coverage)**
  - **Which geometry is visible from the camera at that pixel? (occlusion)**




**Recall perspective projection from first class**

# The visibility problem

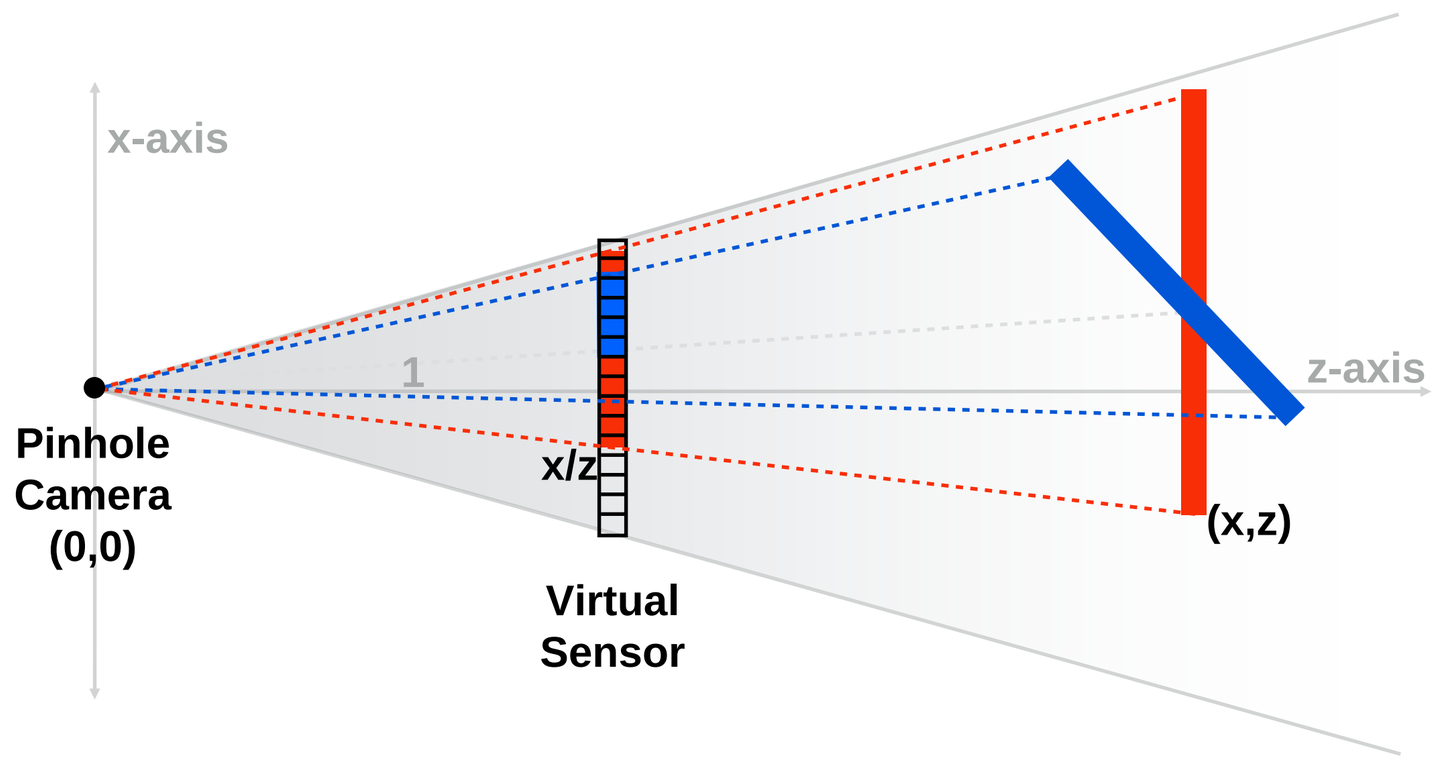
- **An informal definition: what scene geometry is visible within each screen pixel?**
  - **What scene geometry projects into a screen pixel? (coverage)**
  - **Which geometry is visible from the camera at that pixel? (occlusion)**




**Think about light bouncing around from source, to objects in scene through pinhole to the screen**

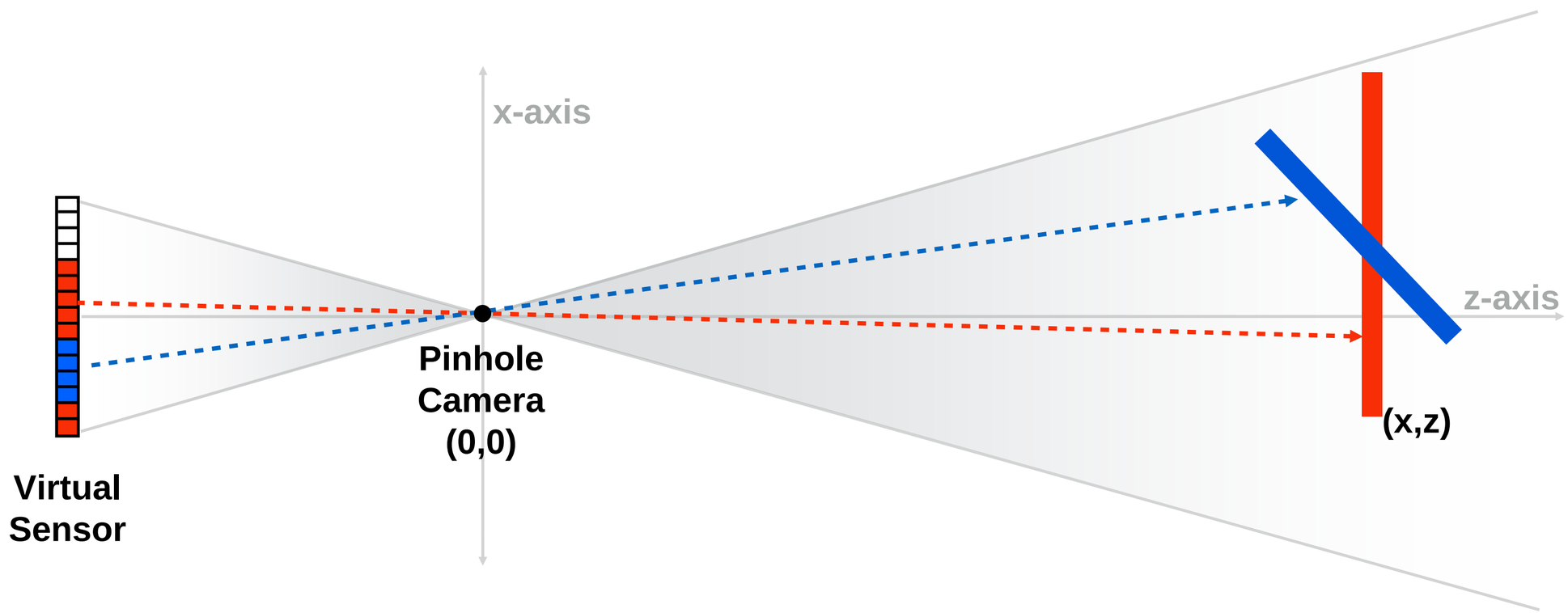
# The visibility problem

- **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)**



# 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)**

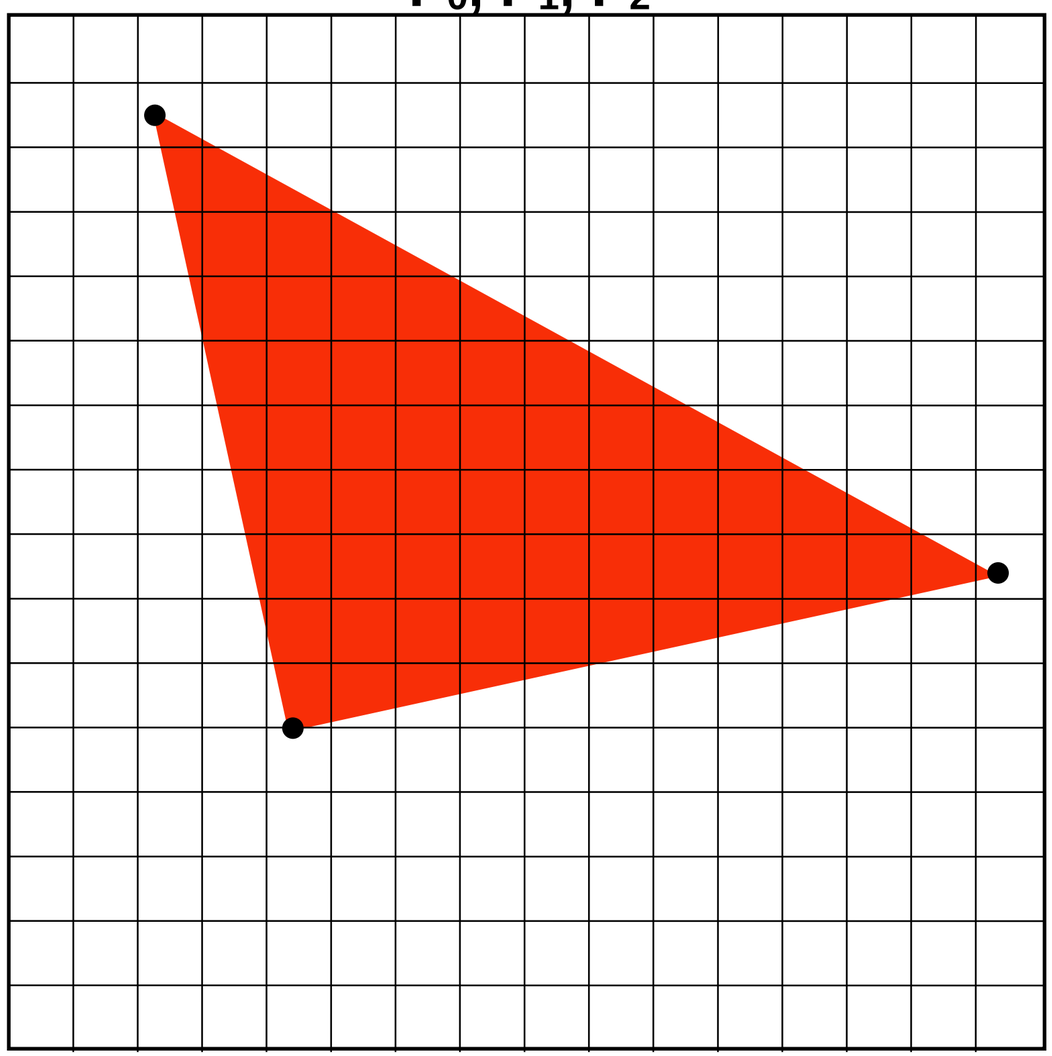



**Hold onto this thought for later in the semester.**

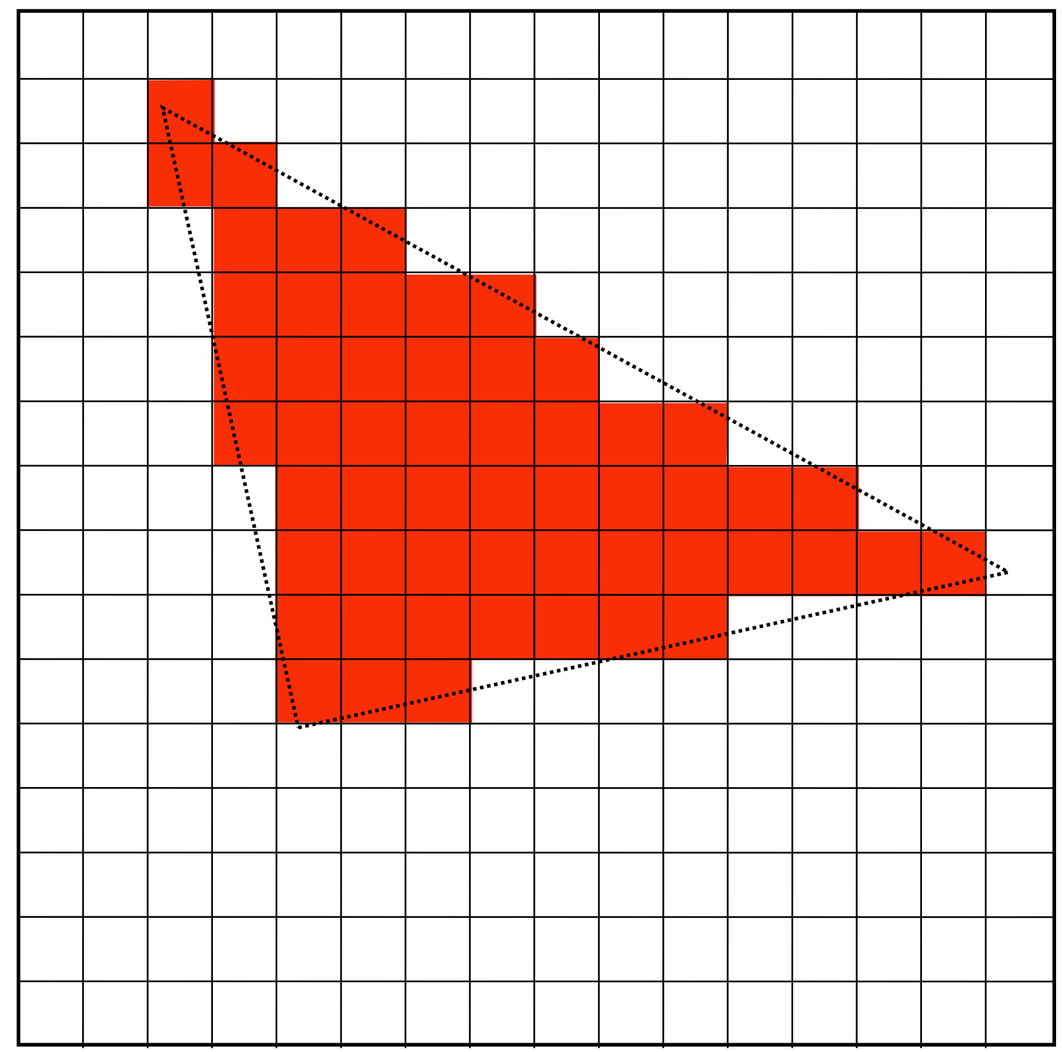
# Computing triangle coverage

## What pixels does the triangle overlap?

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



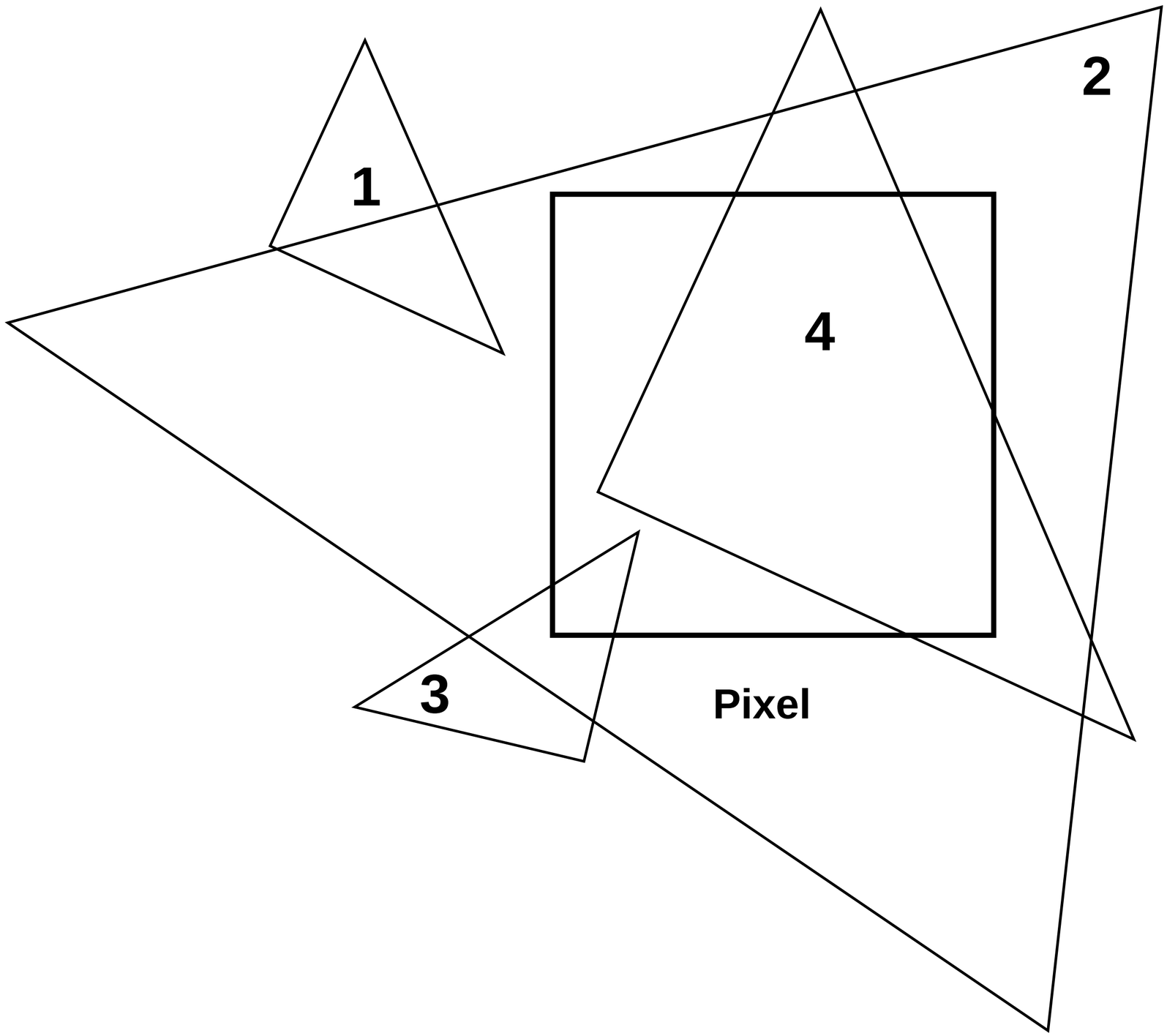
**Output:**
**set of pixels "covered" by the triangle**



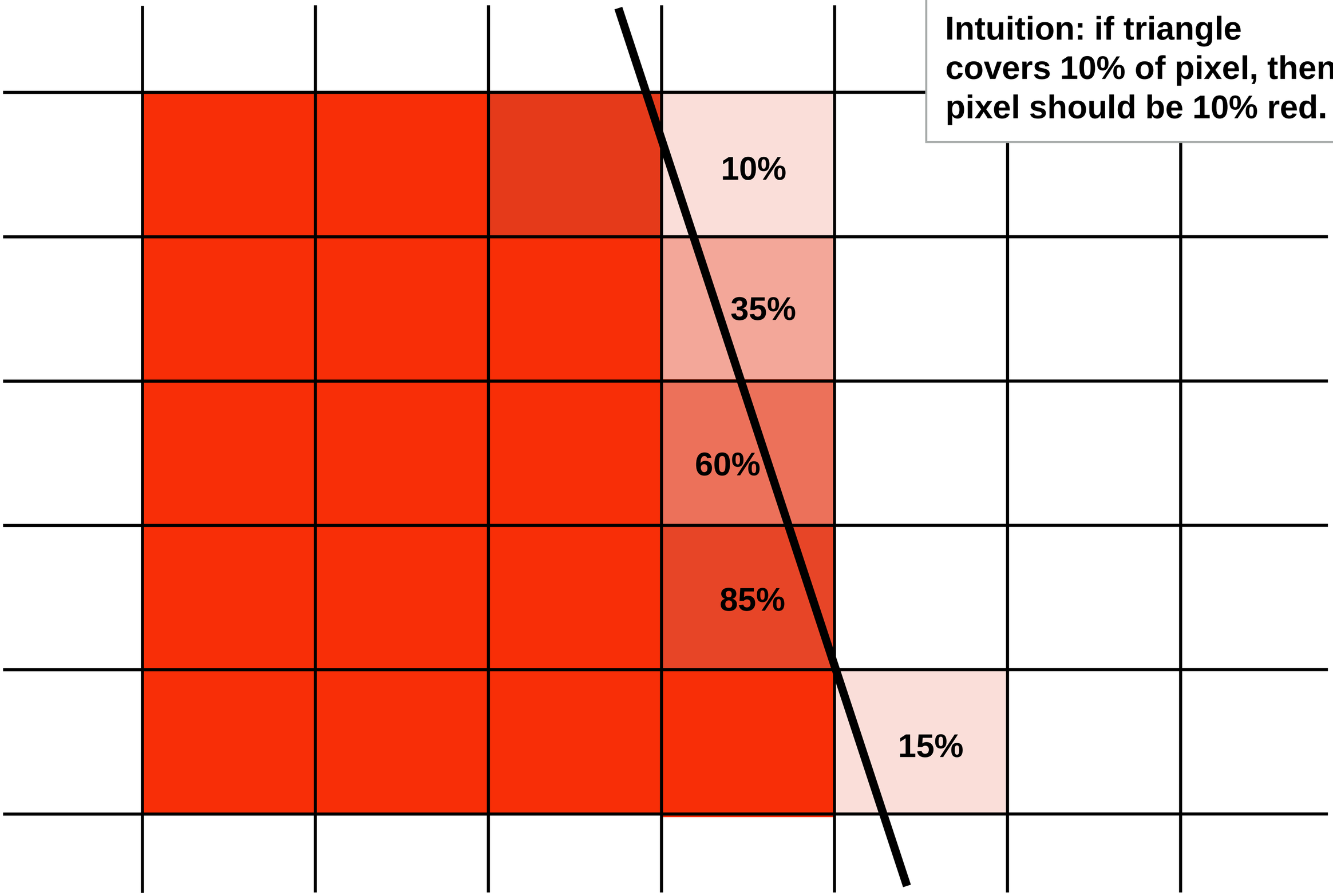
**Note: need to represent a continuous signal using a discrete approximation!**

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

**Question: which triangles "cover" this pixel?**

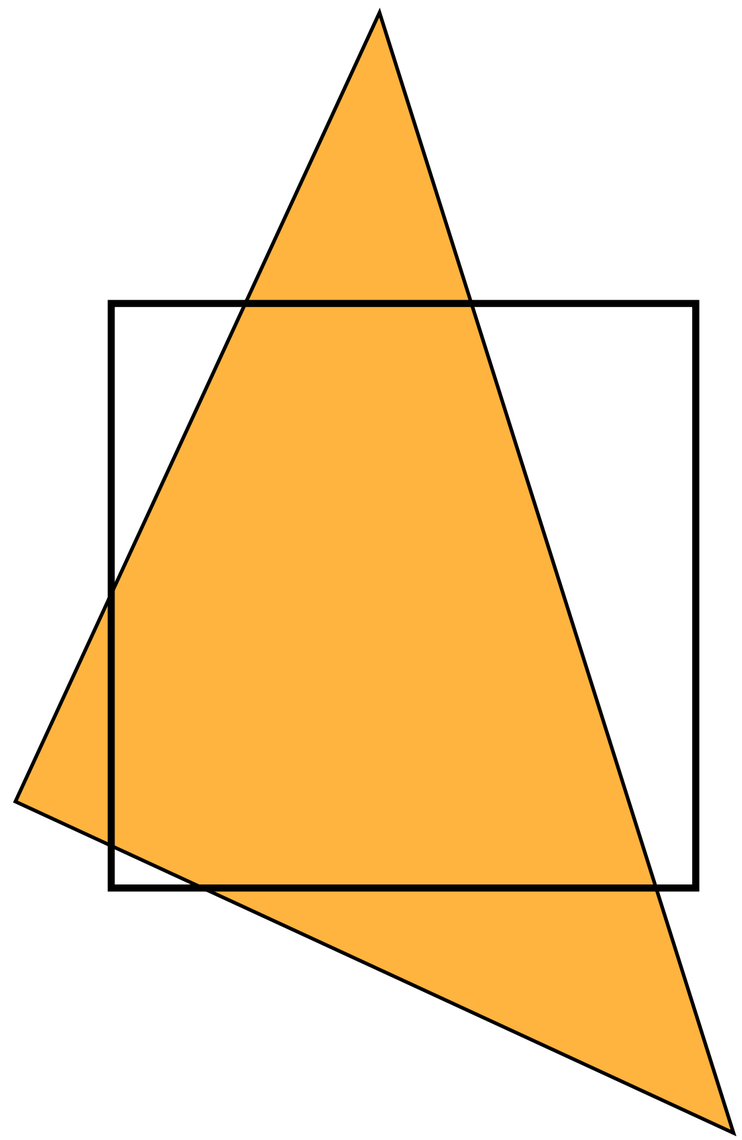


# One option: compute fraction of pixel area covered by triangle, then color pixel according to this fraction.

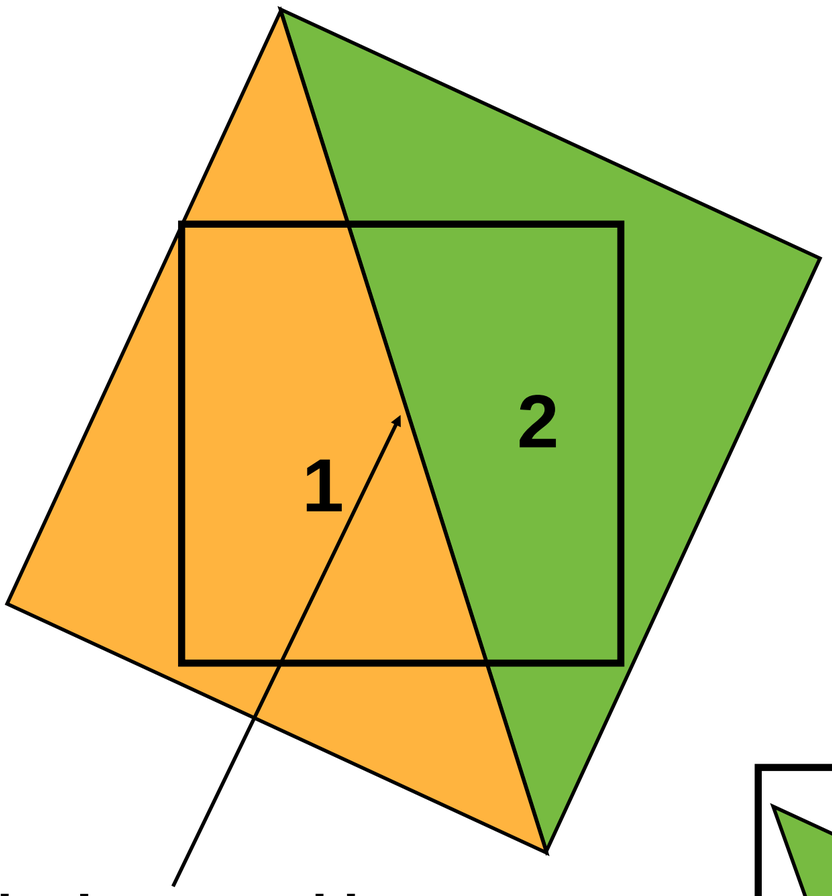



**Intuition: if triangle covers 10% of pixel, then pixel should be 10% red.**

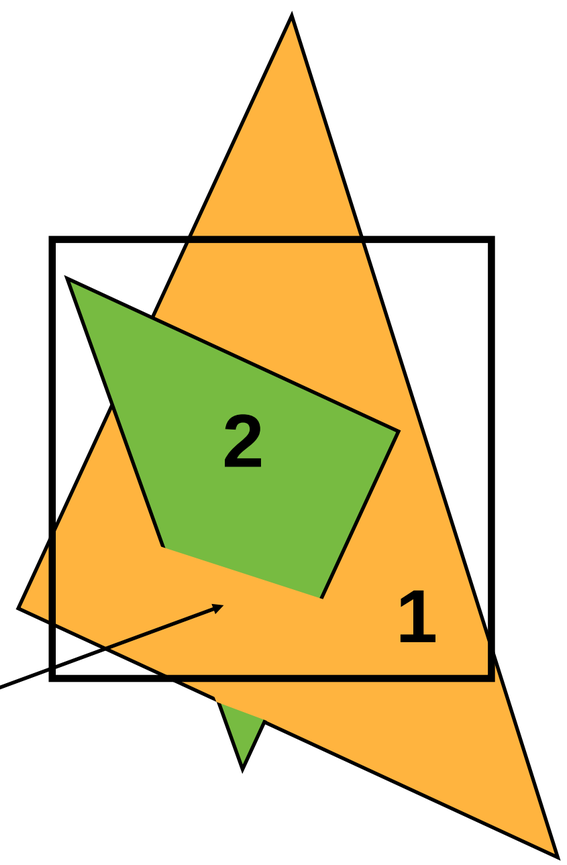
# Computing amount of overlap?



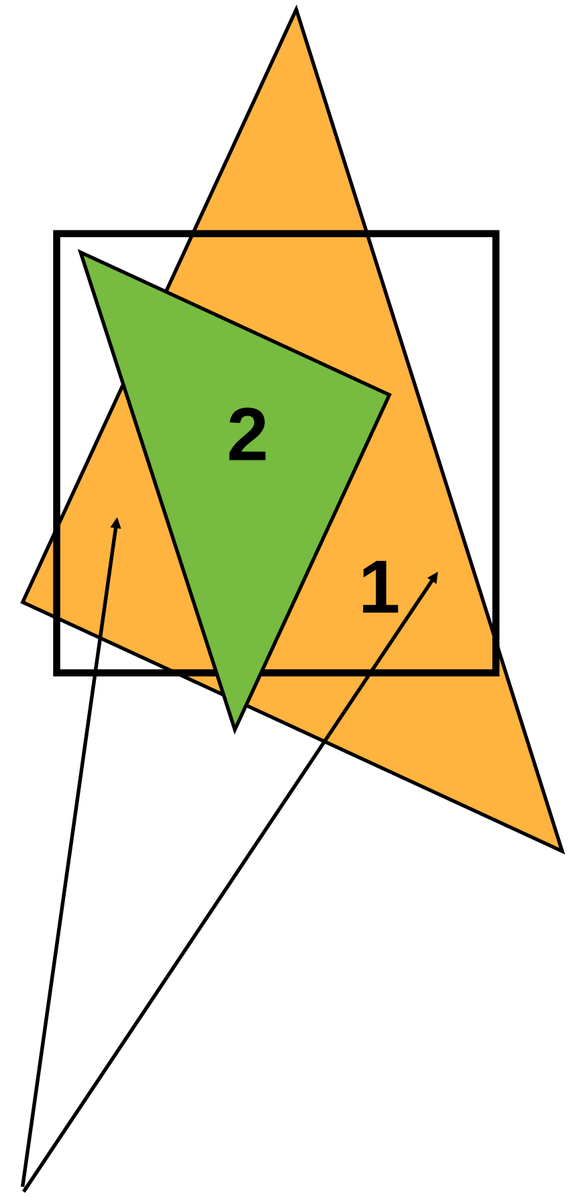
# Analytical schemes can get quite tricky, especially when considering interactions between multiple triangles




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

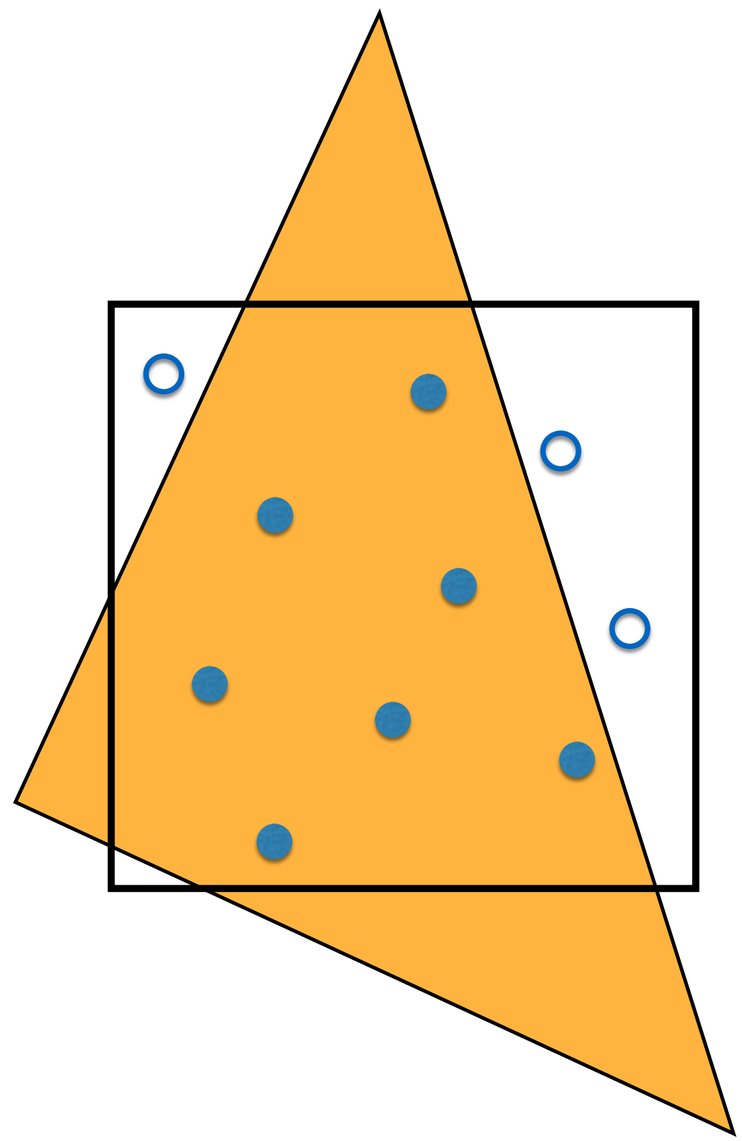



Interpenetration of triangles: even trickier




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

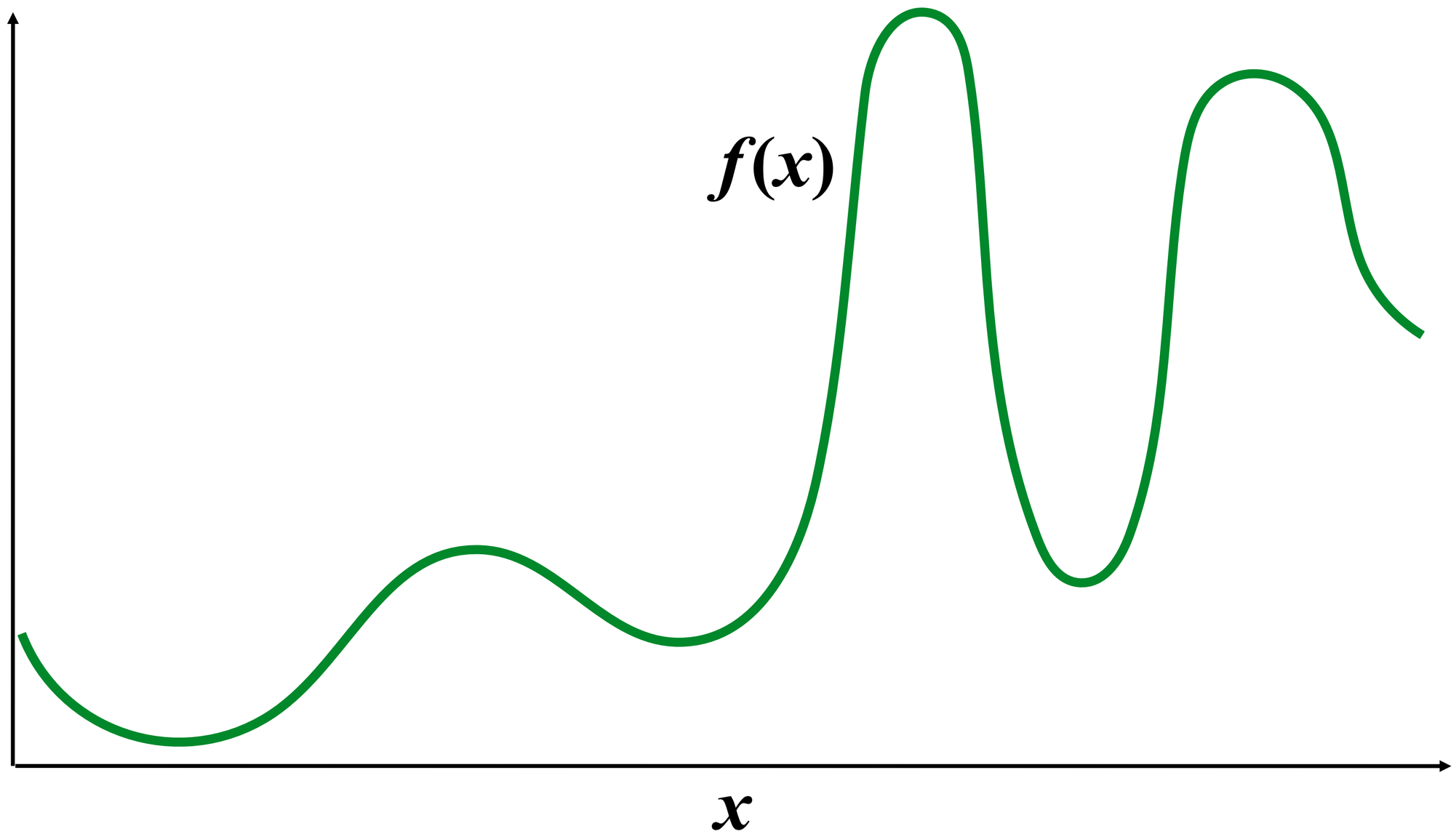
# Estimating amount of overlap through sampling



**What is a principled approach to think about this process?**

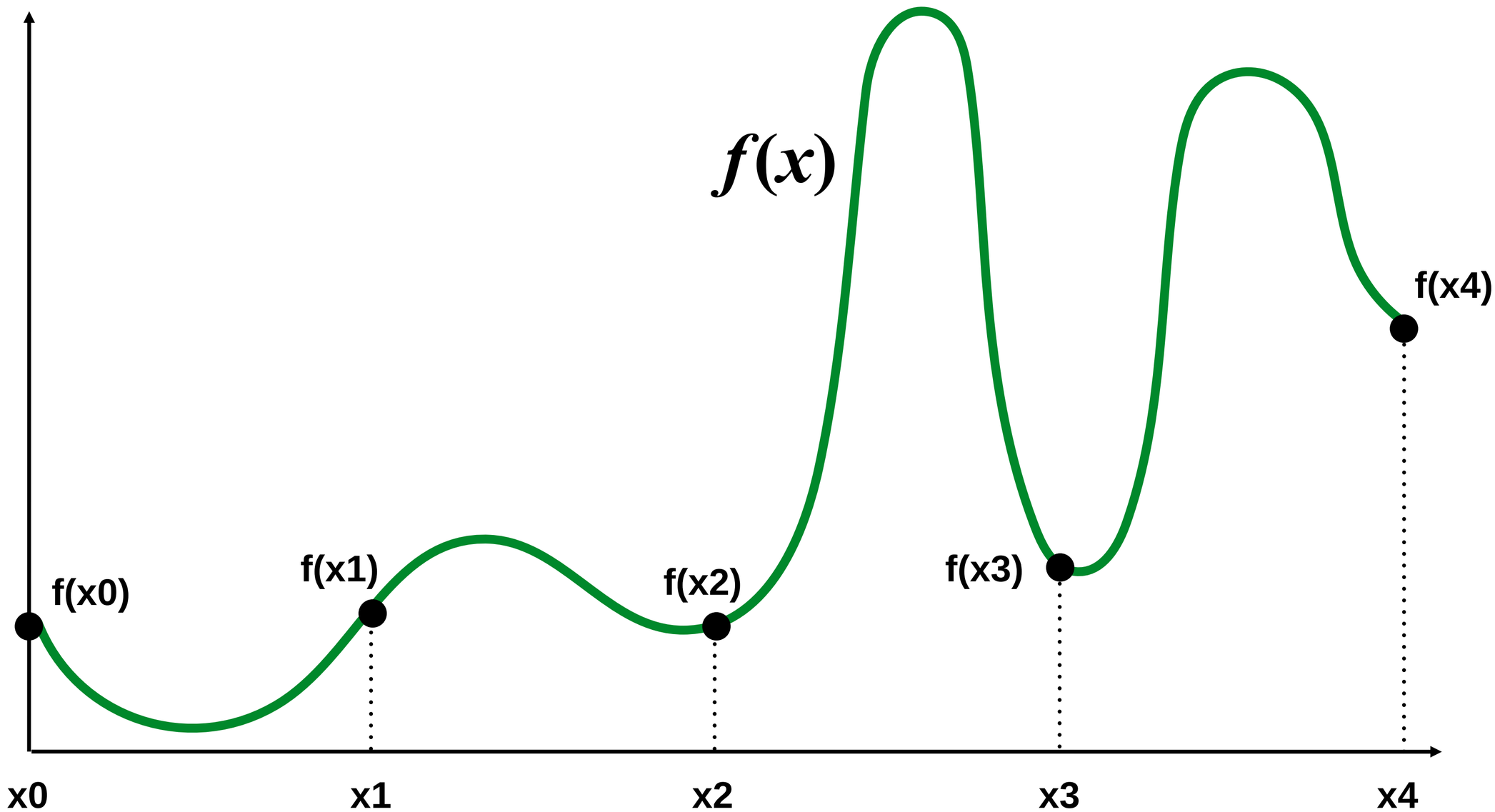
# Sampling 101

# 1D signal



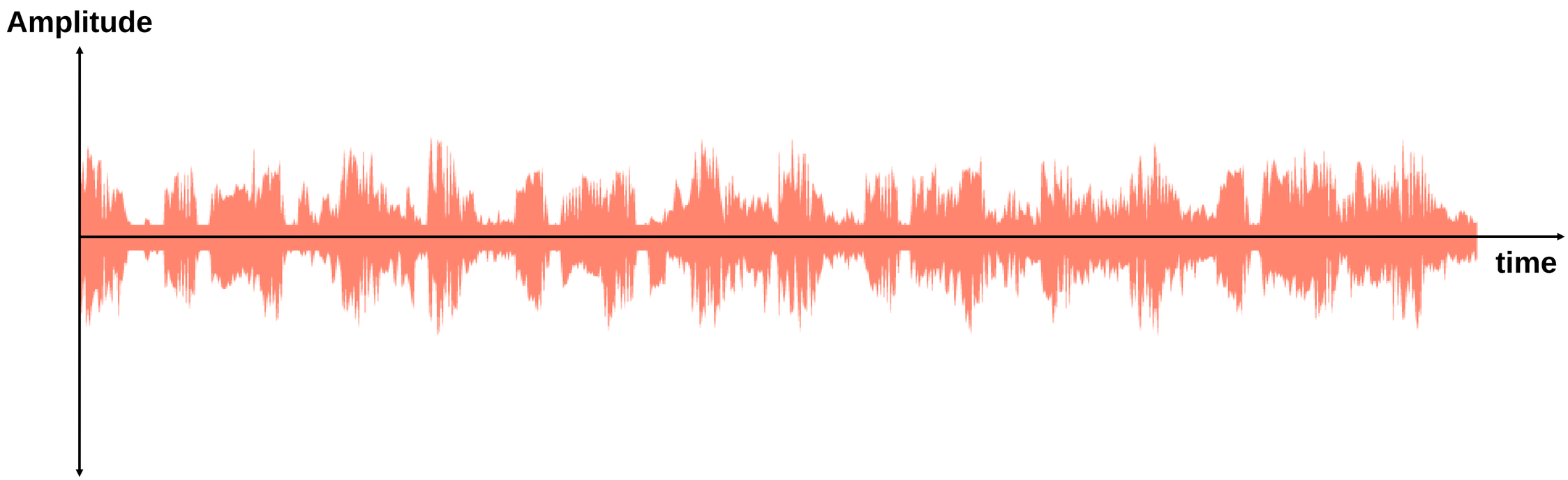
# Sampling: from continuous to discrete

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



# Audio file: stores samples of a 1D signal

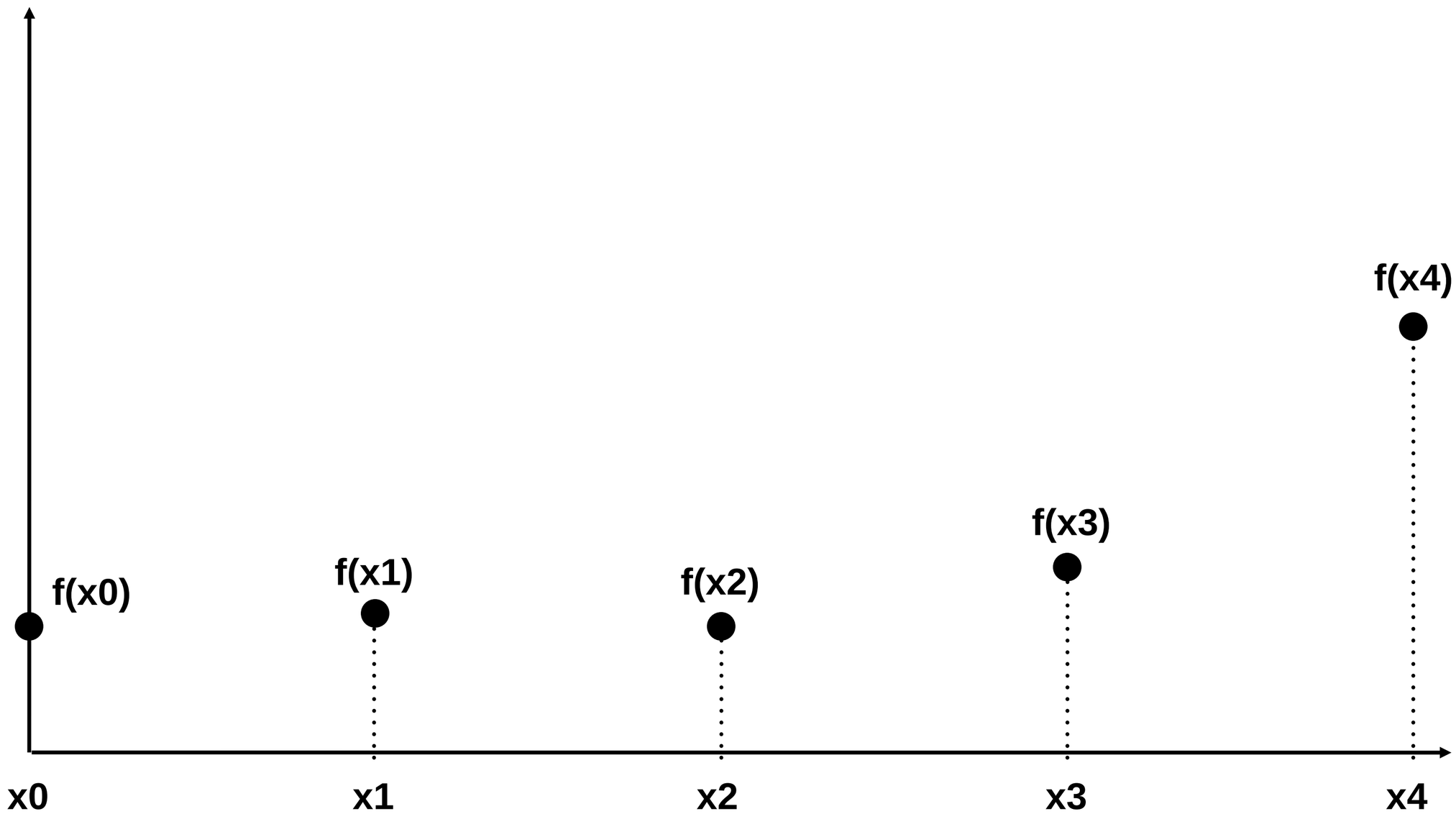
## Most consumer audio is sampled at 44.1 KHz




**Q: Why 44.1Khz?**

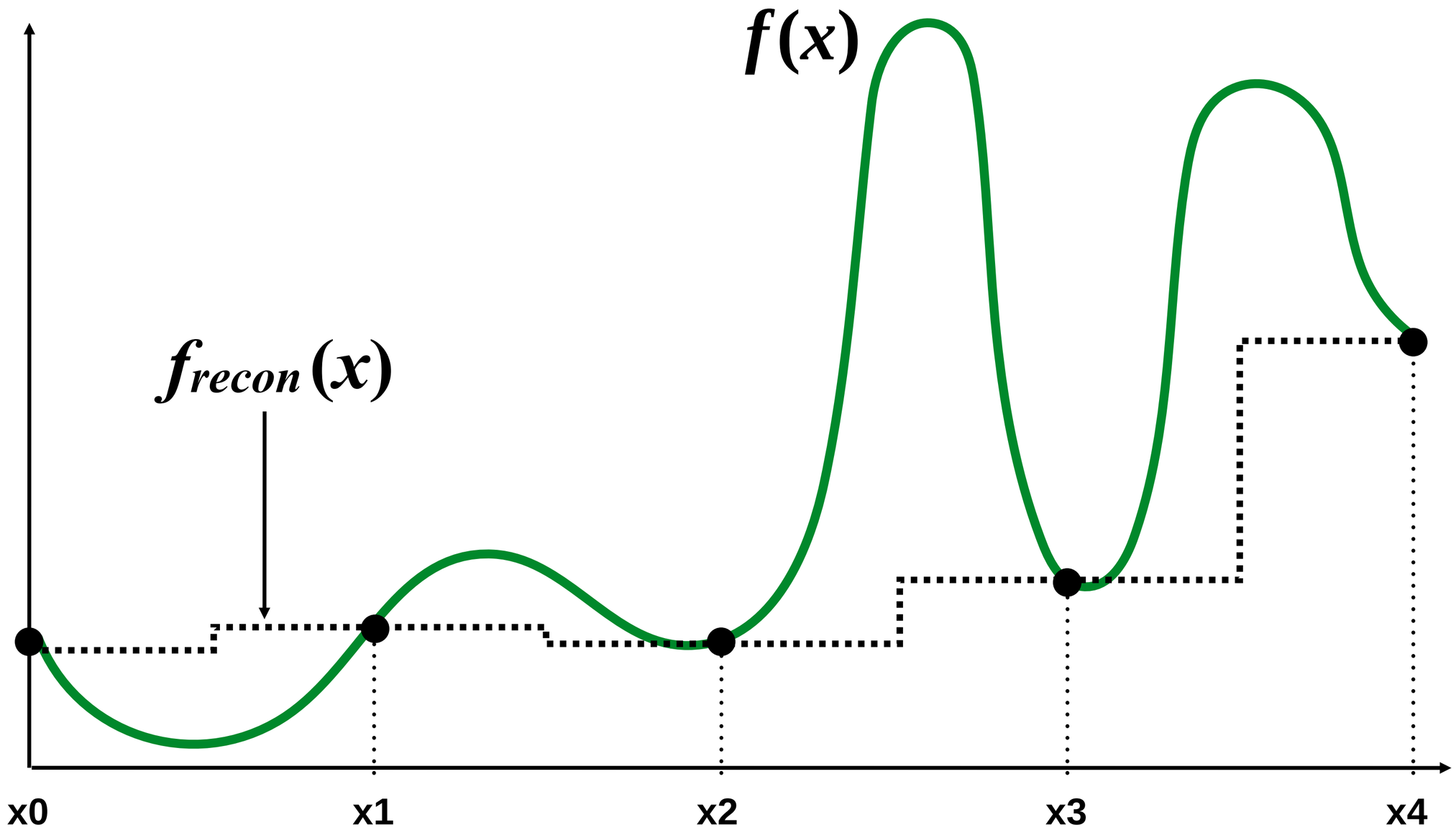
# Reconstruction: from discrete to continuous (an interpolation problem)

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



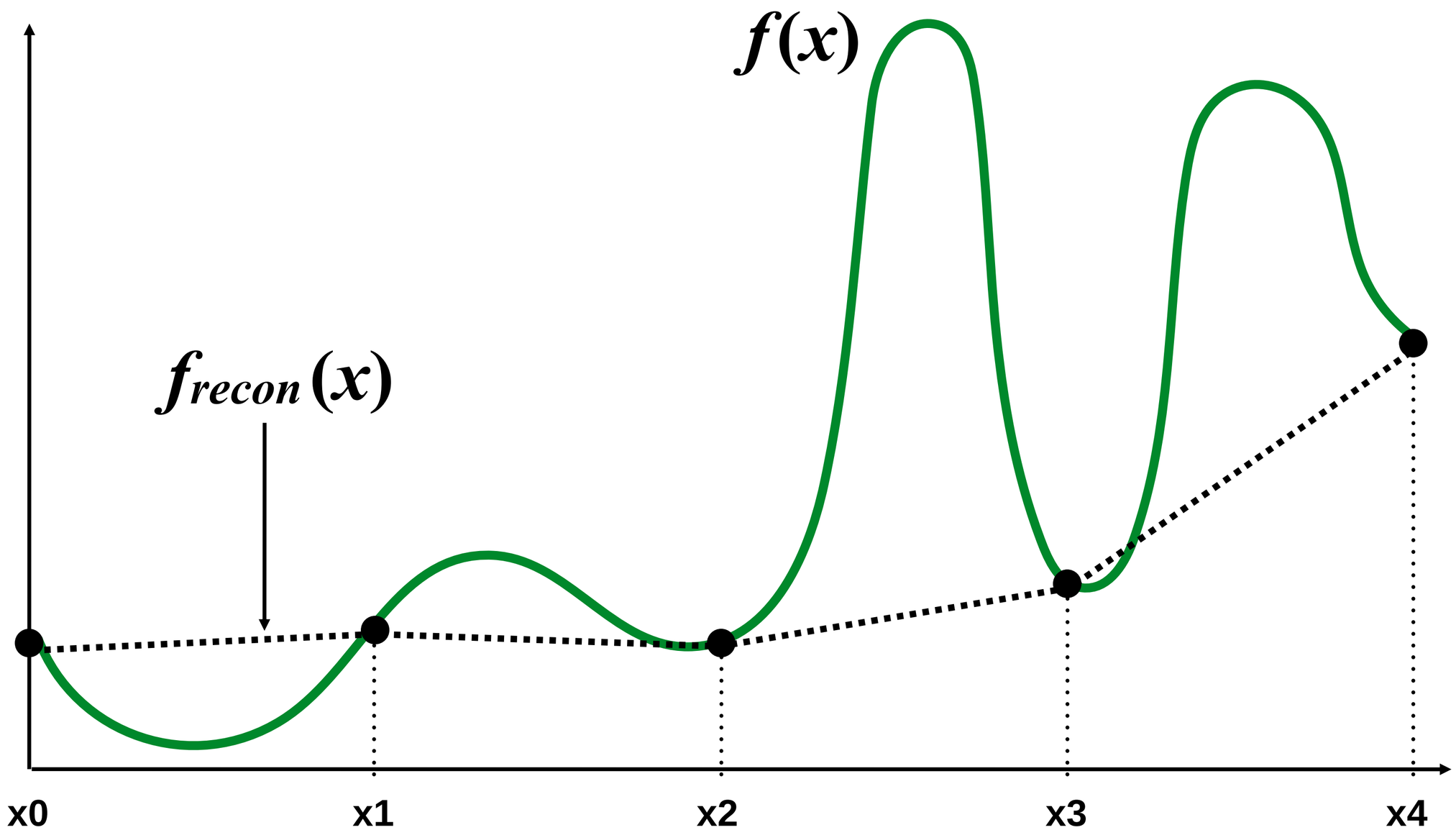
# Piecewise constant approximation

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

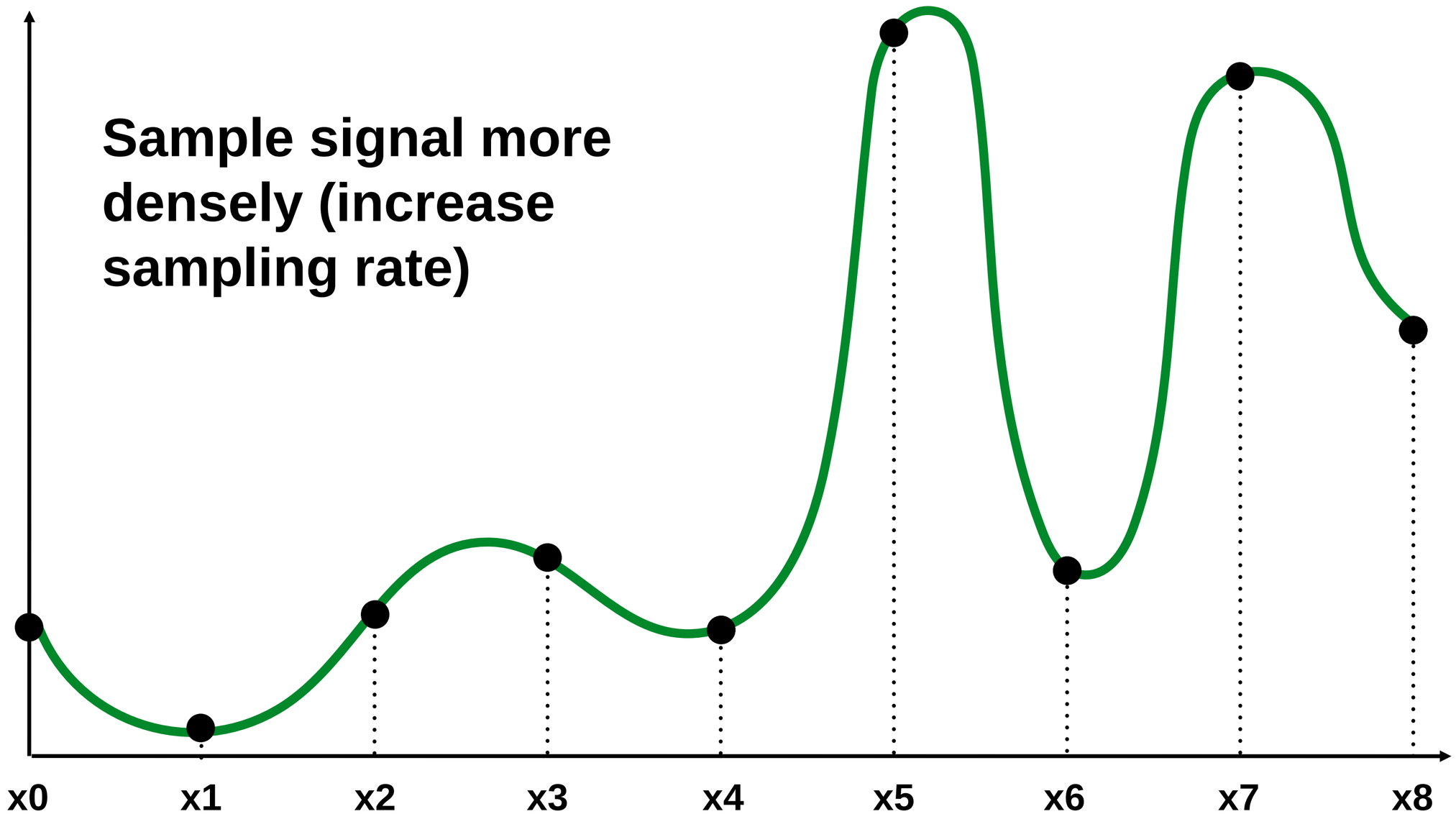


# Piecewise linear approximation

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

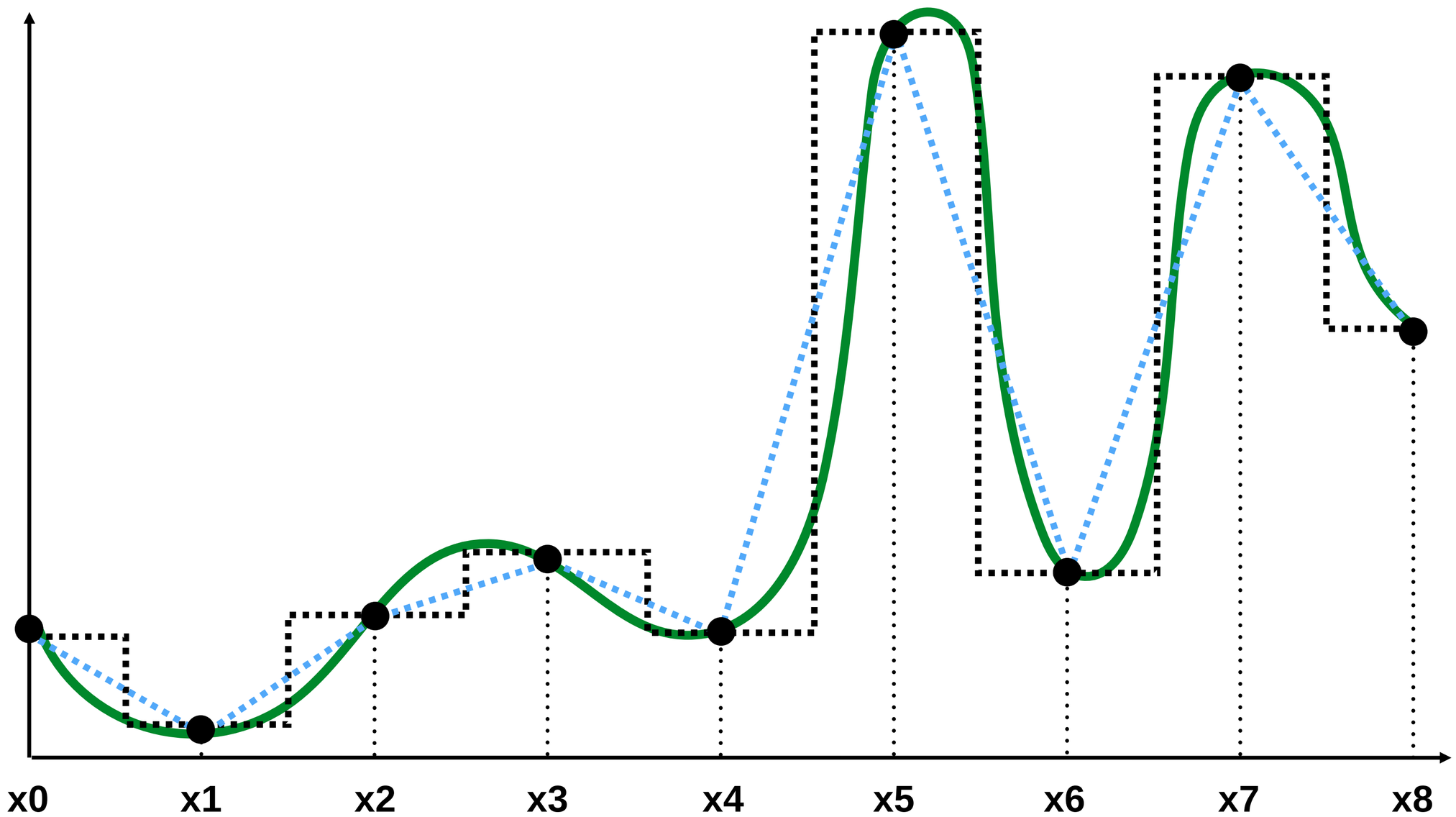


# How can we reconstruct the signal more accurately?




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

# Reconstruction from denser sampling




= reconstruction via nearest neighbor

= reconstruction via linear interpolation

# Sampling and Reconstruction

- As an aside
  - Sampling rate is obviously very important
    - Why limit it?
  - In general, what else might you worry about when sampling a signal?
    - Noise
    - Quantization
    - Impulse response

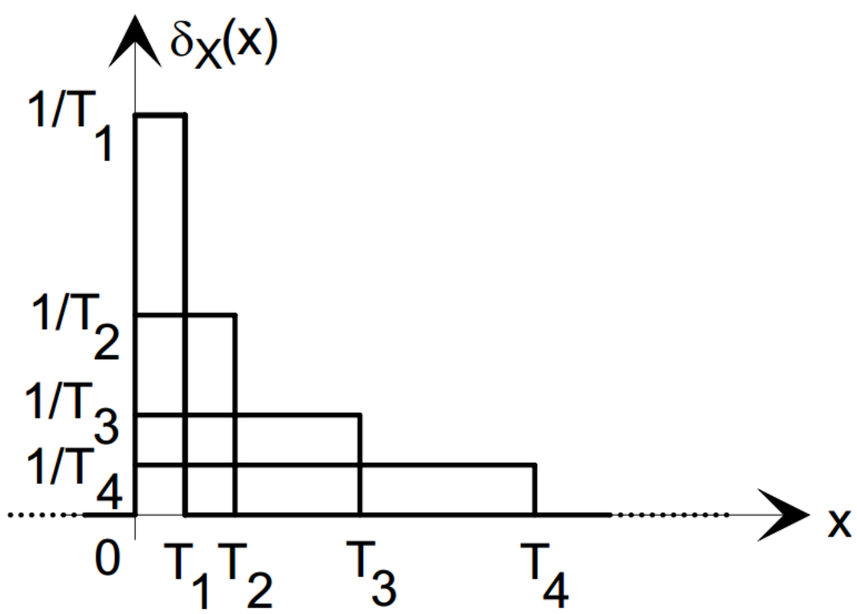
# Mathematical representation of sampling

**Consider the Dirac delta:**

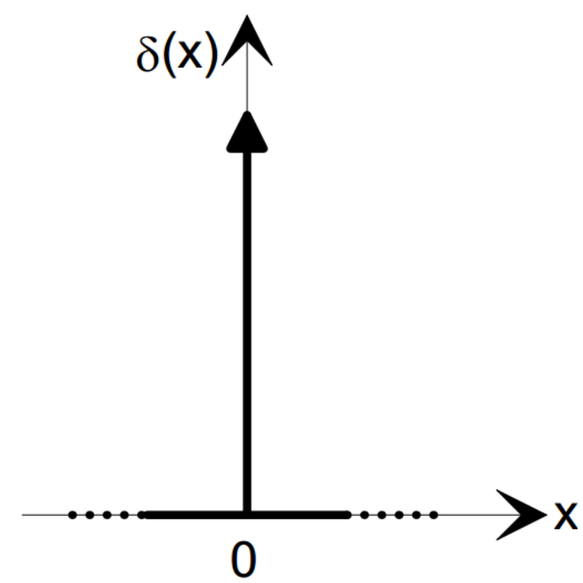
$$\delta(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ \text{undefined} & \text{at } x = 0 \end{cases}$$

**s.t.**

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




**Unit pulse functions**




**An Impulse**

# Mathematical representation of sampling

**The ‘sifting’ property of the Impulse:**

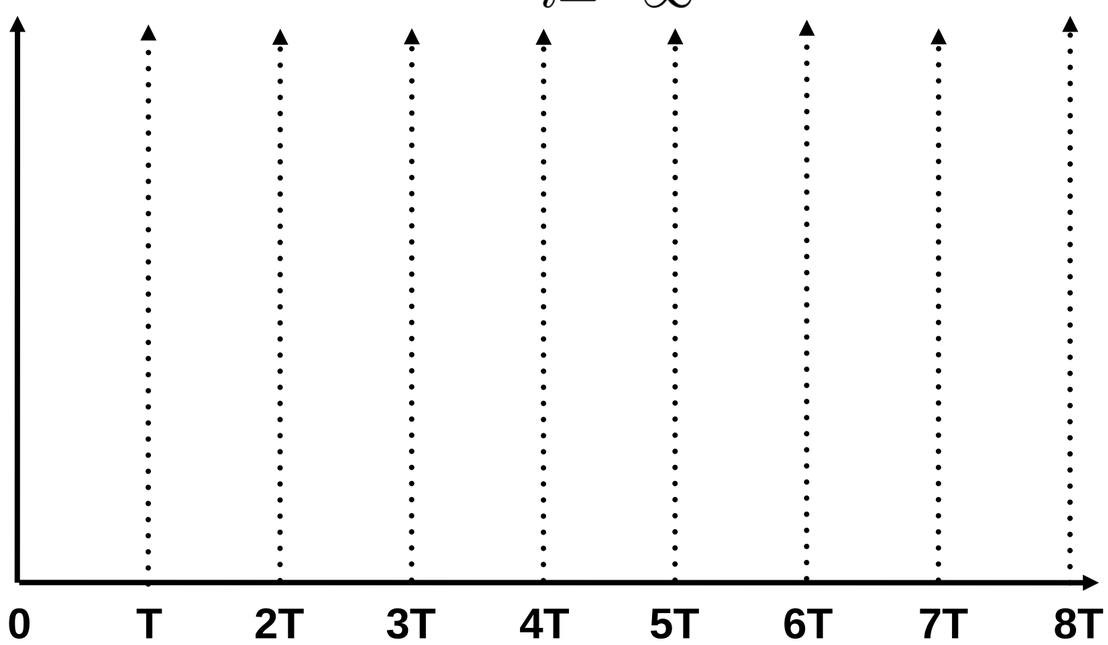
$$\int_{-\infty}^{\infty} f(x)\underbrace{\delta(x-a)}dx = f(a)$$

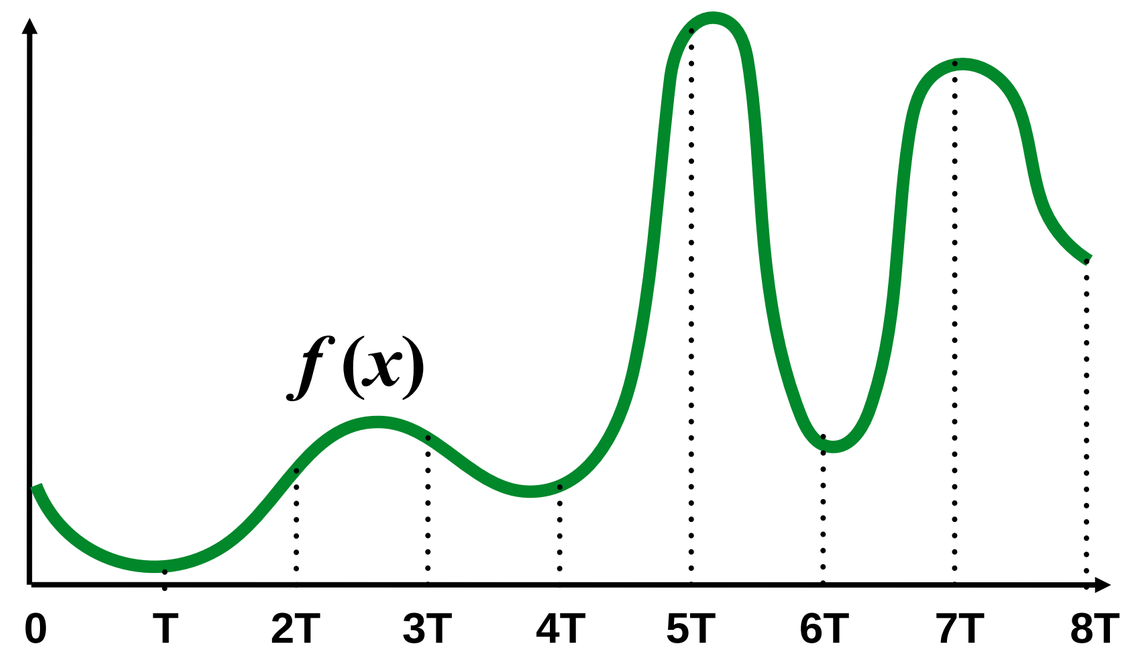
**Impulse occurring at** $x = a$

# Sampling function

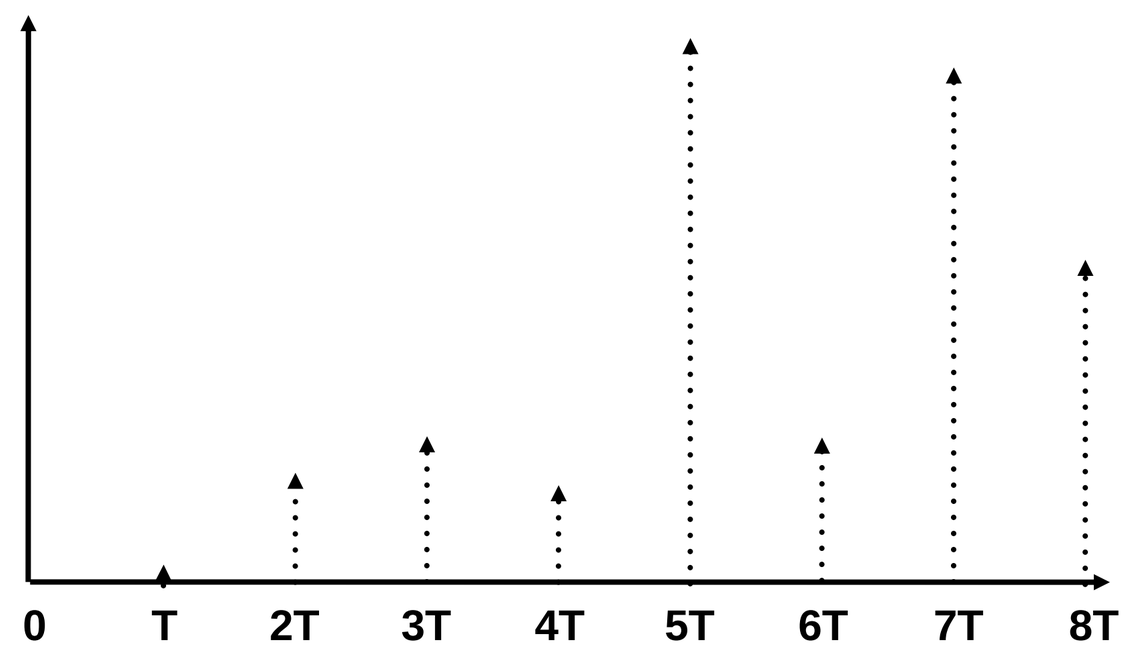
**Consider a sequence of impulses with period $T$:**

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




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




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

# Reconstruction as convolution

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

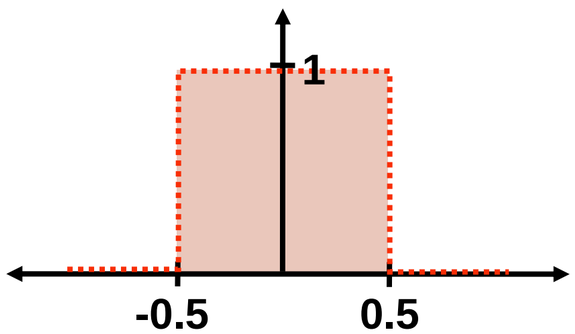
**reconstructed signal ("smooth" version of $g$)**

**filter**

**input signal (sampled signal)**

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

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




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

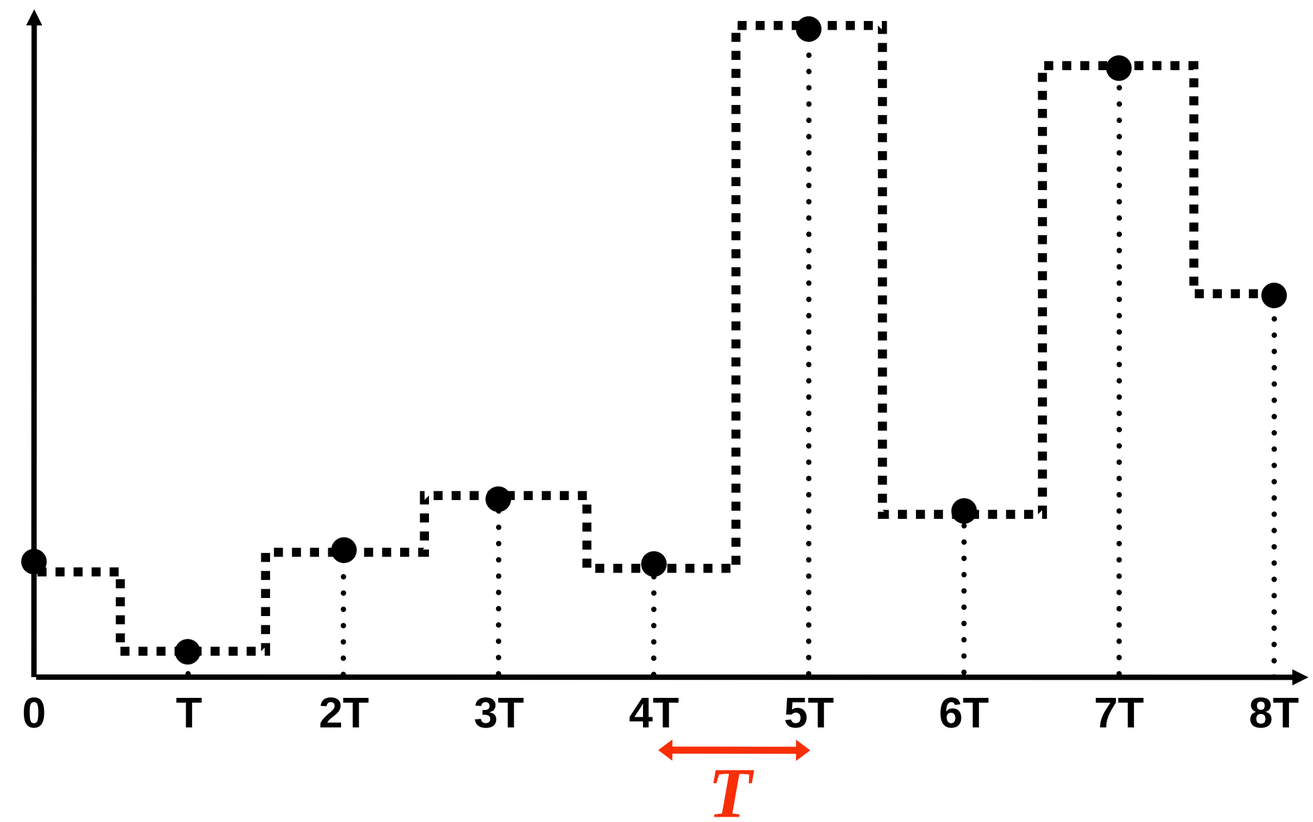
# Reconstruction as convolution (box filter)

**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)**

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




**Reconstructed signal:**
**(nearest neighbor)**

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

**non-zero only for $iT$ closest to $x$**

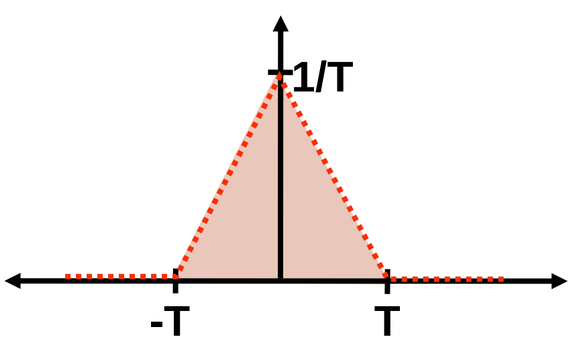
# Reconstruction as convolution (triangle filter)

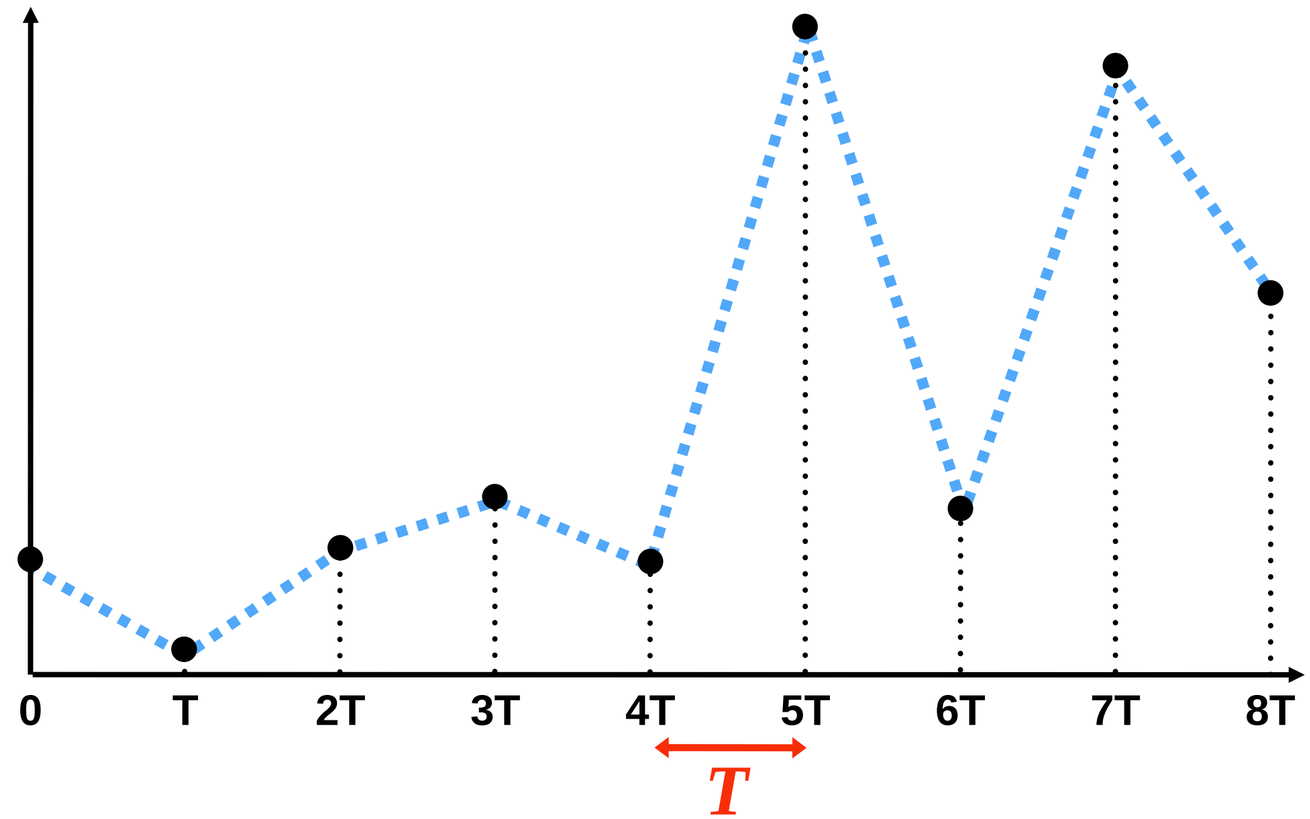
**Sampled signal:**
**(with period $T$)**

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

**Reconstruction filter:**
**(unit area triangle of width T)**

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



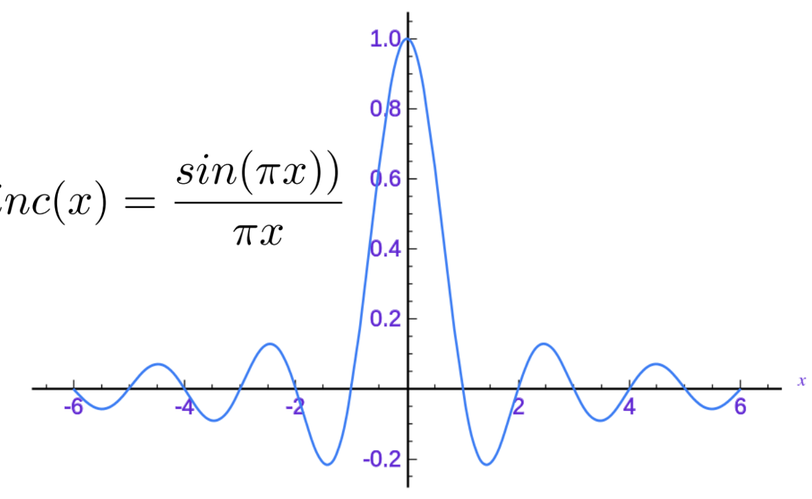

**Reconstructed signal:**

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

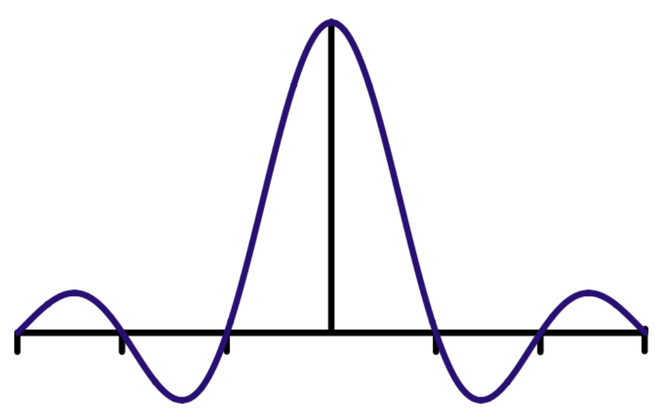
# Summary

- 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

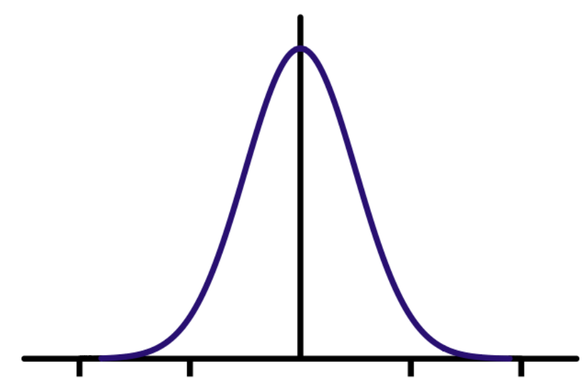
$$sinc(x) = \frac{sin(\pi x))}{\pi x}$$




Normalized sinc filter



Truncated sinc filter



Truncated gaussian filter

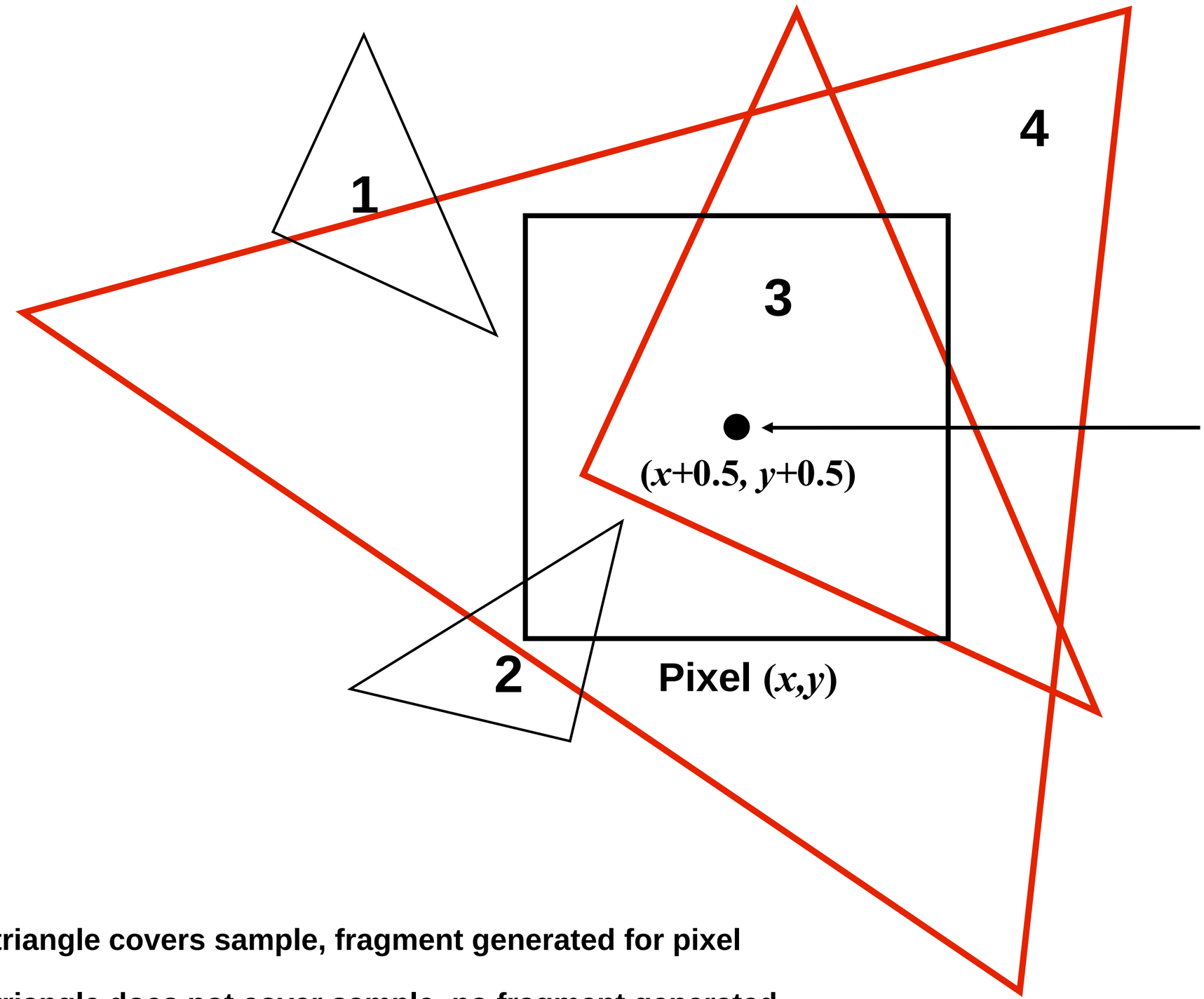
[Image credit: Wikipedia]

# 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:**
**Pixel center is chosen as coverage sample point.**

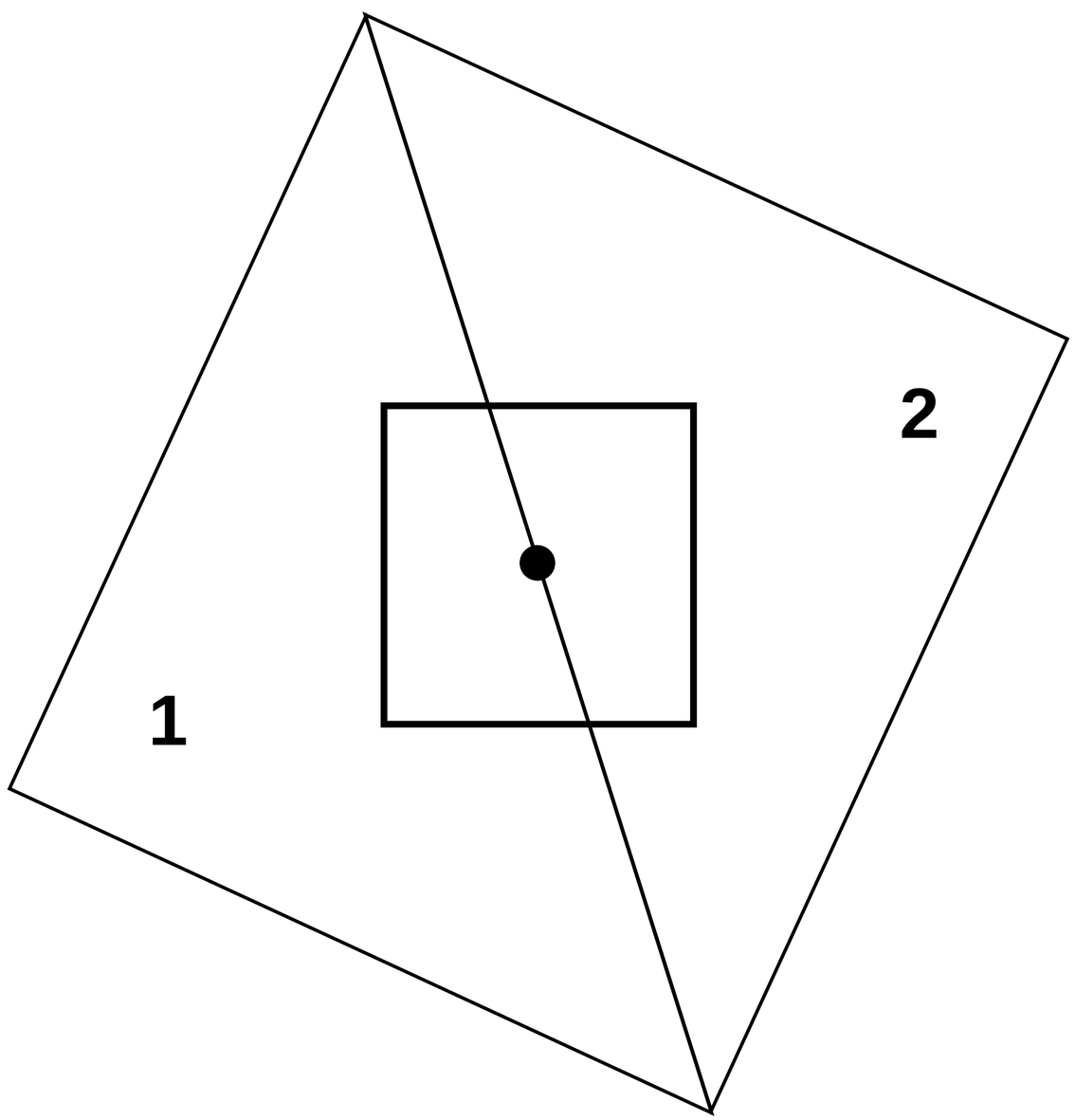
= triangle covers sample, fragment generated for pixel

= triangle does not cover sample, no fragment generated

# Edge cases (literally)

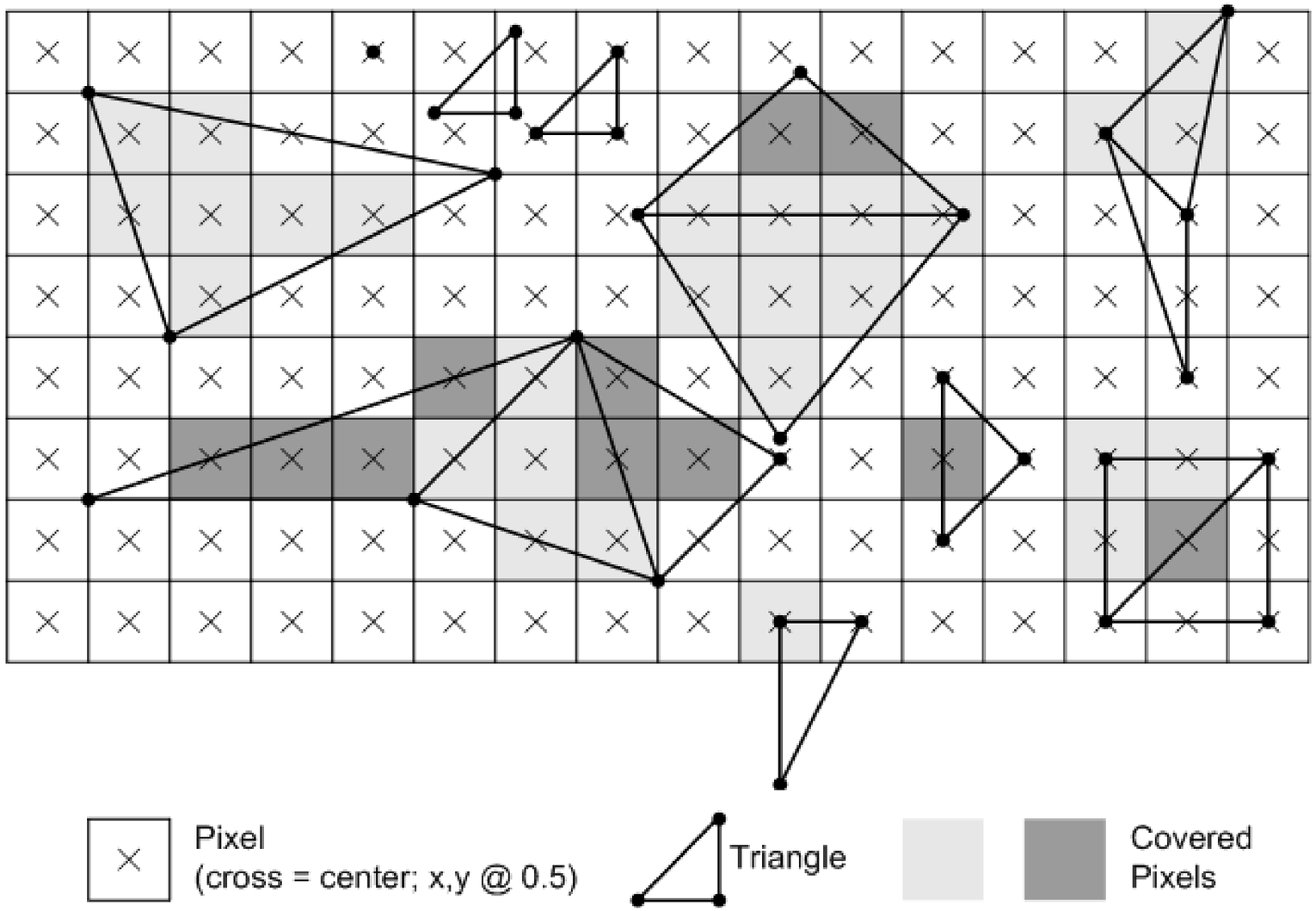
**Is this sample point covered by triangle 1? or triangle 2? or both? or none?**

**Why is it important to decide?**



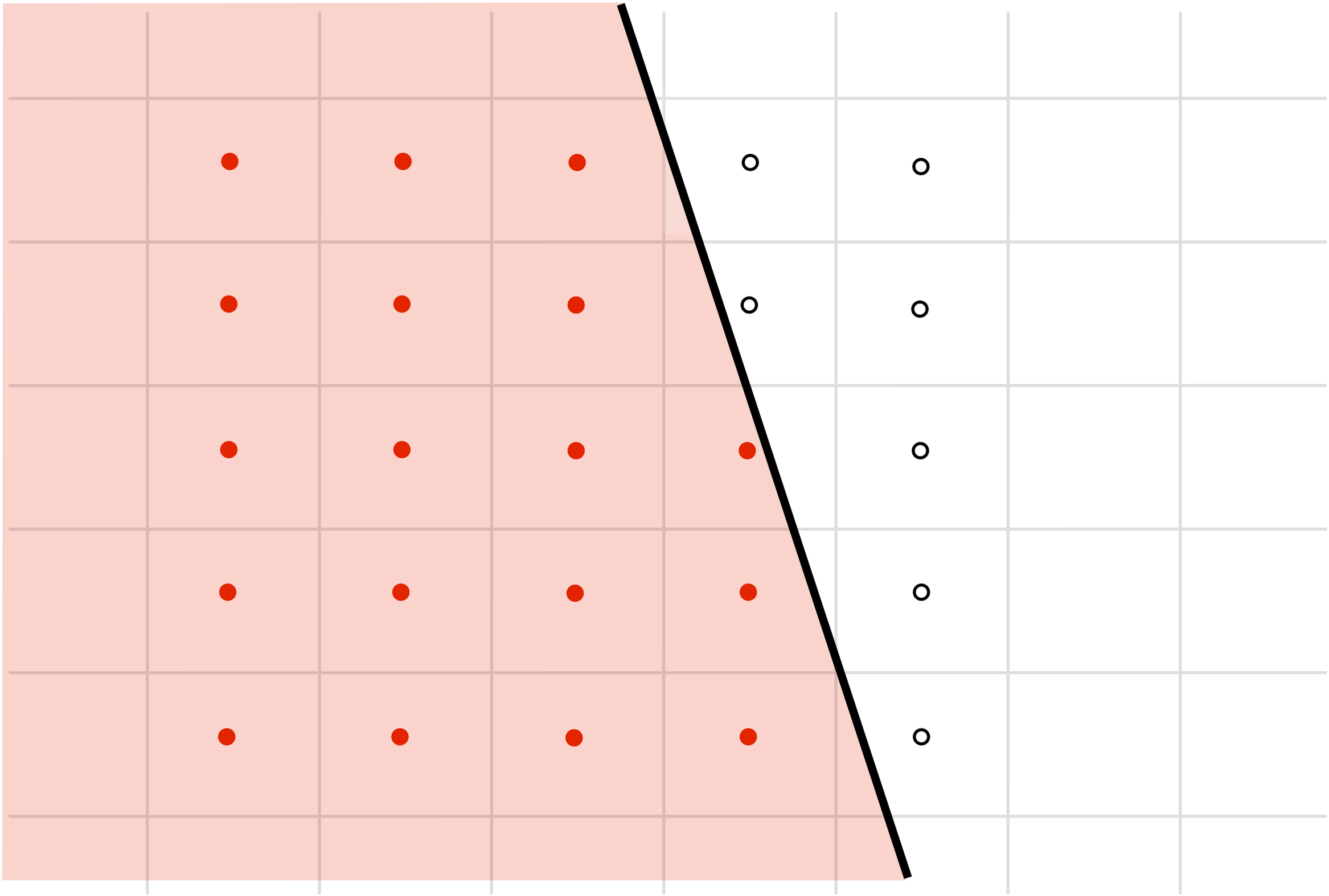
# OpenGL/Direct3D edge rules

- 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)

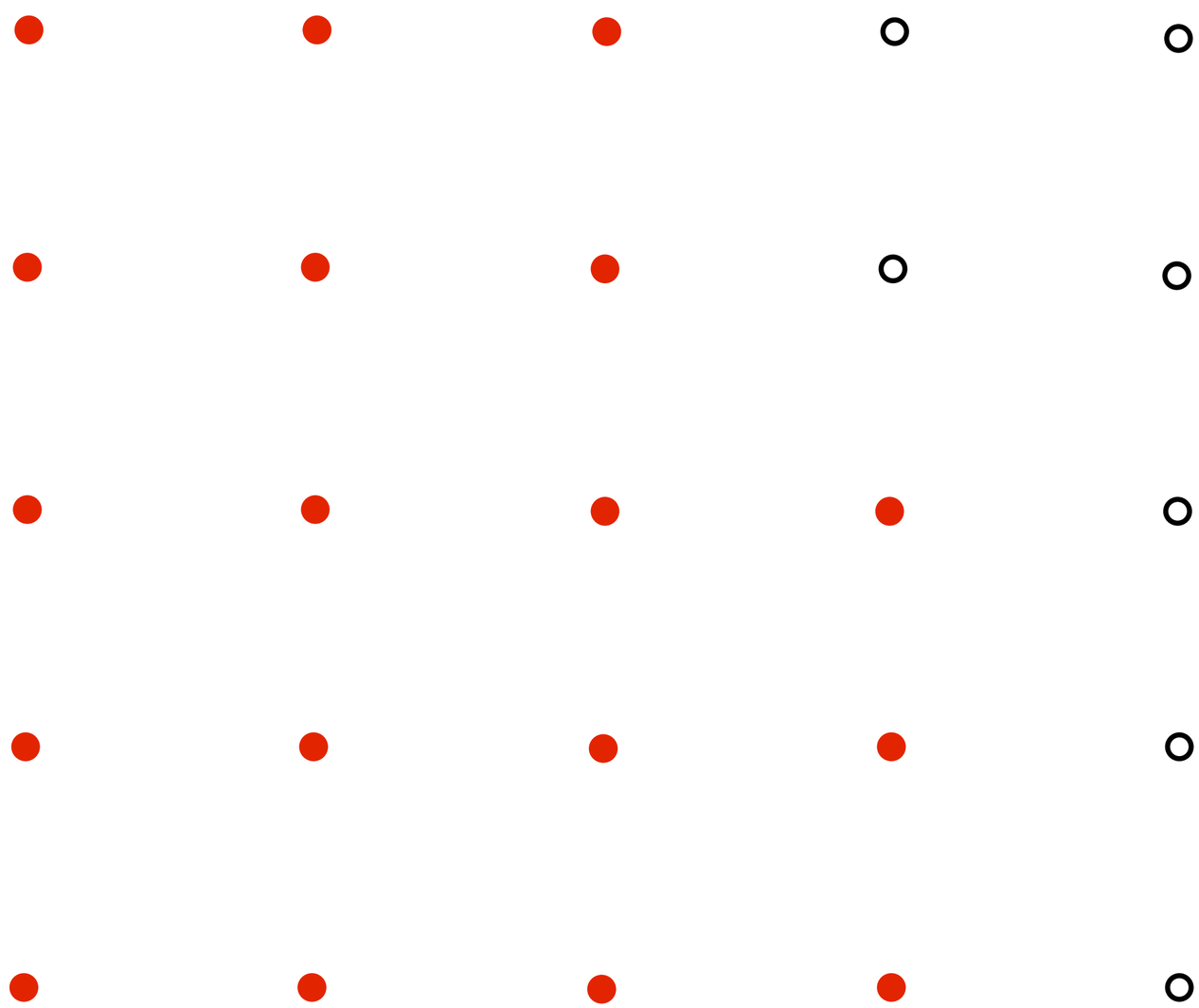



Source: Direct3D Programming Guide, Microsoft

# Results of sampling triangle coverage

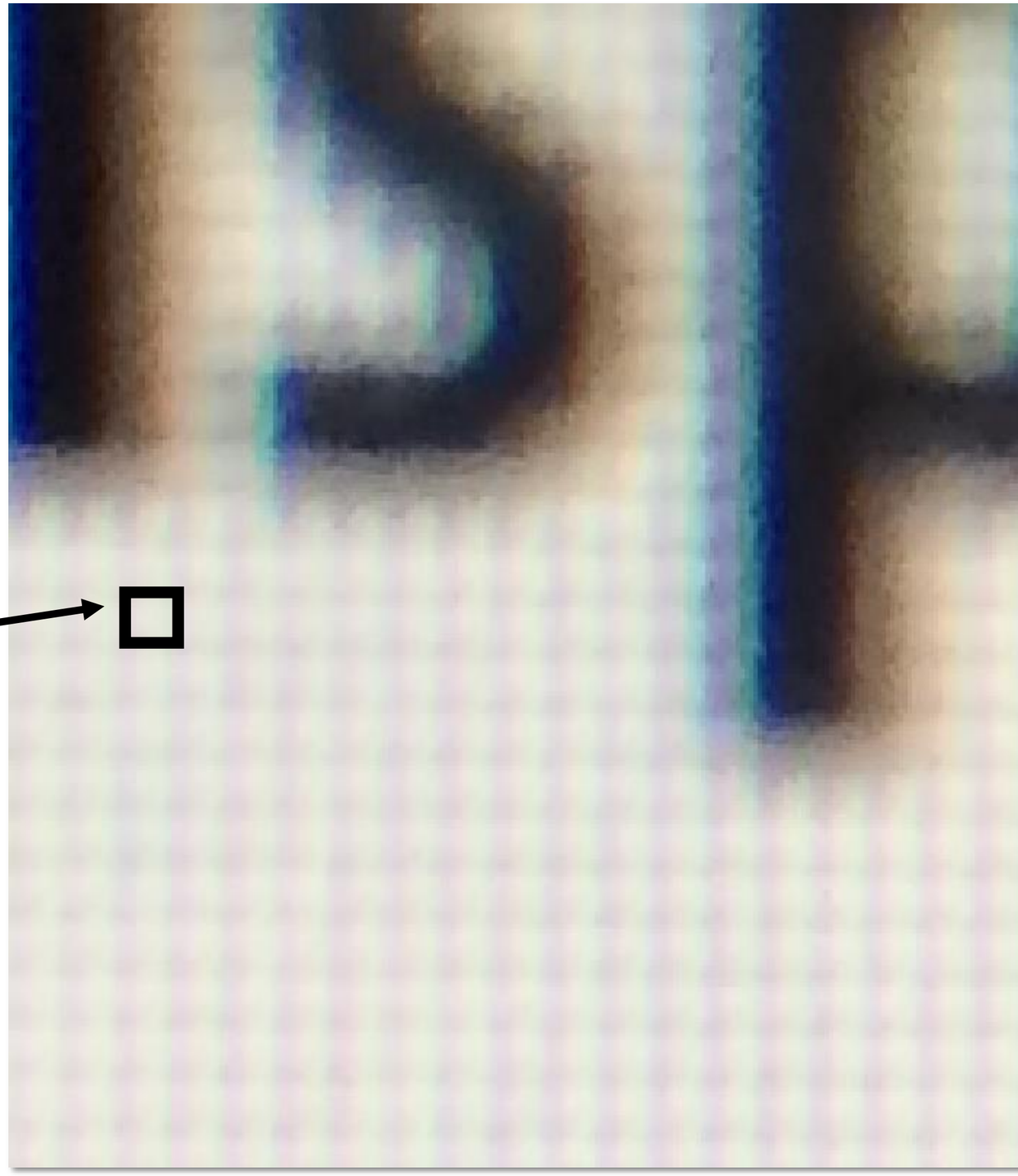


# I have a sampled signal, now I want to display it on a screen



# Pixels on a screen

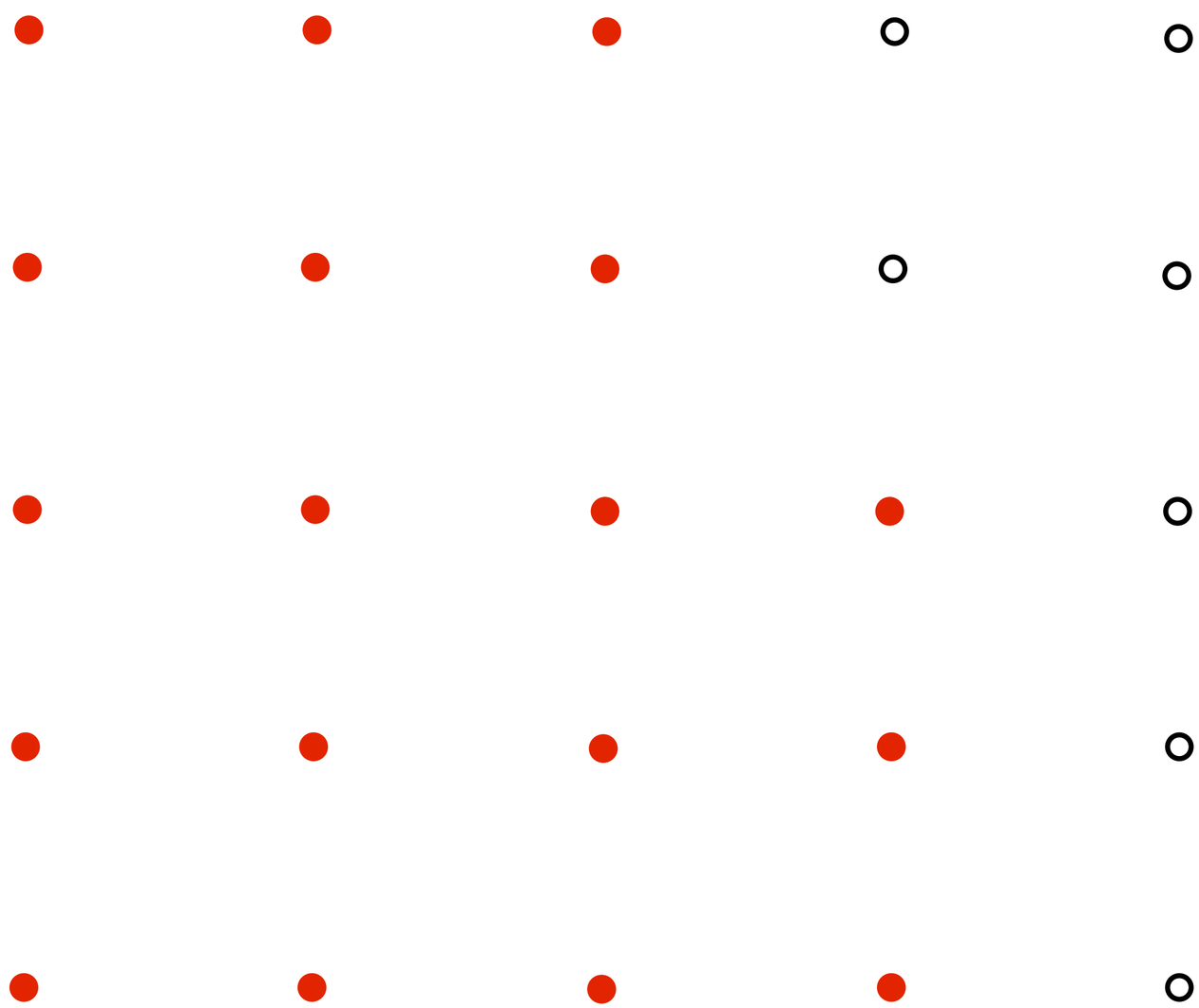
**Each image sample sent to the display is converted into a little square of light of the appropriate color:**
**(a pixel = picture element)**



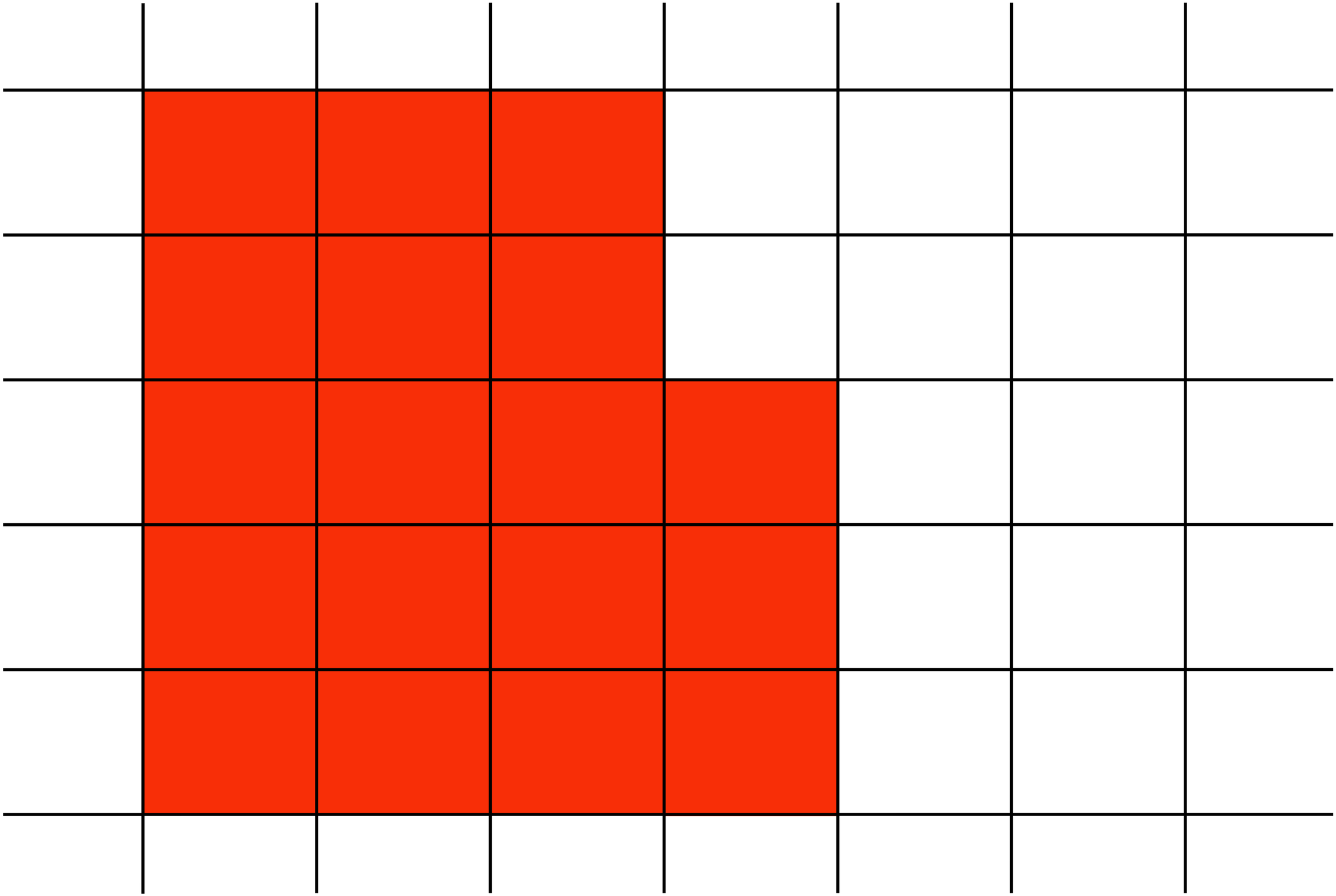
**LCD display pixel**

*** 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

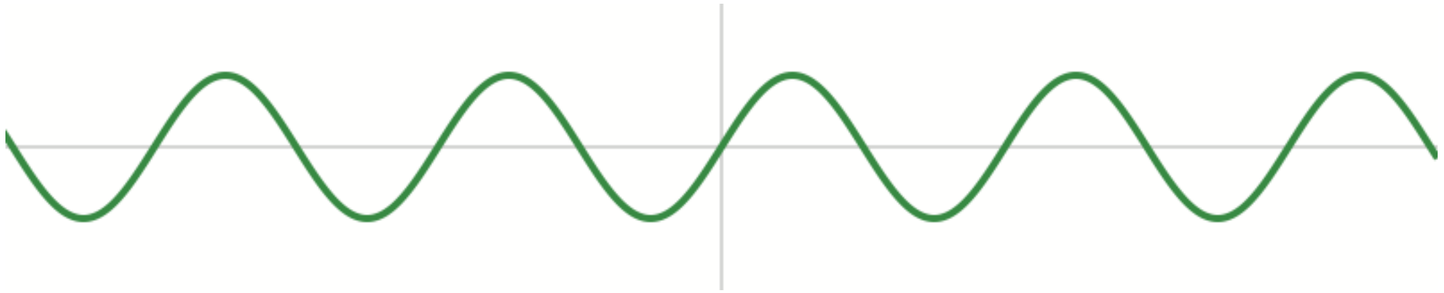


# Recall: the real coverage signal

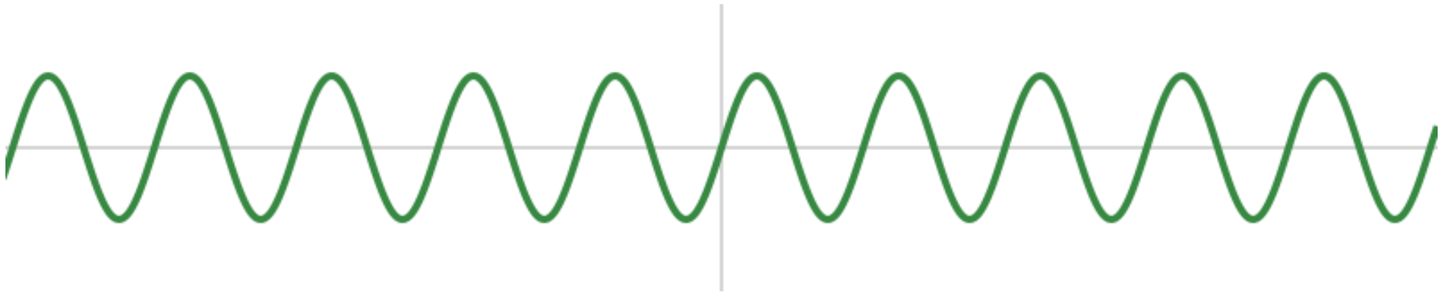
# Aliasing

# Representing signals as a superposition of frequencies

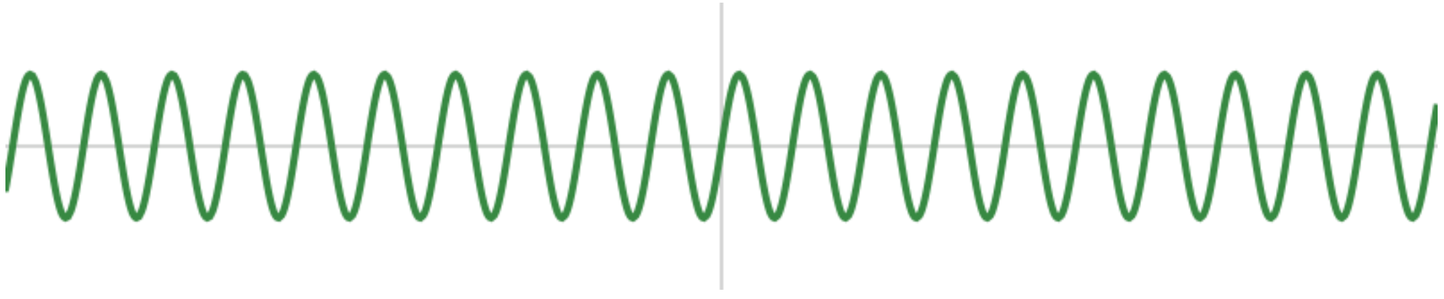
$f_1(x) = sin(\pi x)$



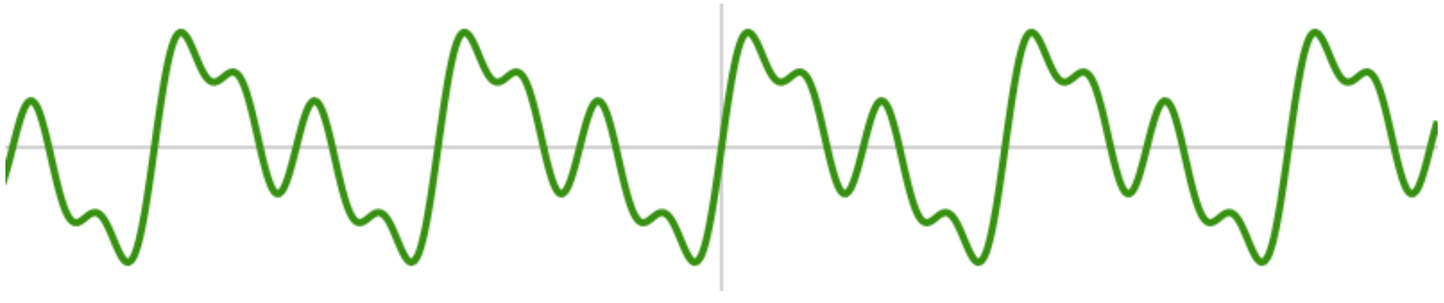
$f_2(x) = sin(2\pi x)$



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



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



# Representing signals as a superposition of frequencies

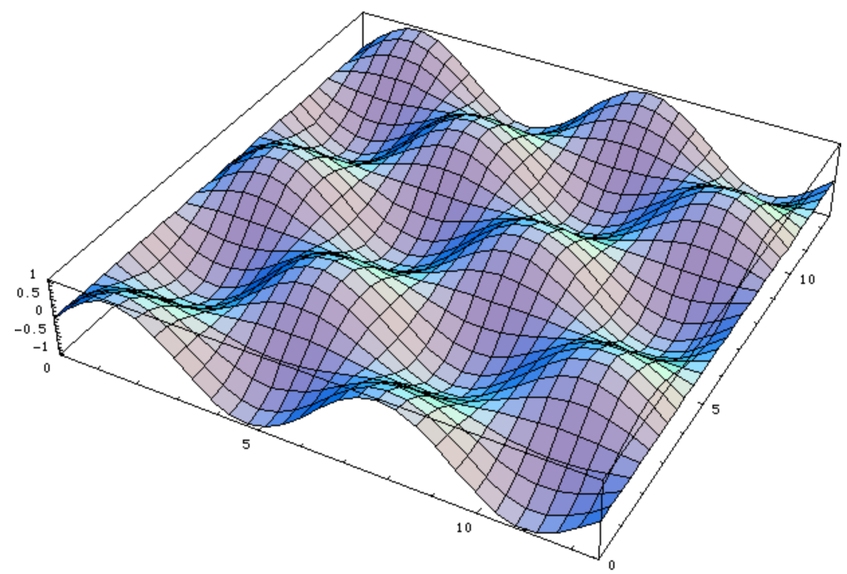
# Representing signals as a superposition of frequencies

Spatial domain

Frequency domain

It’s exactly the same function!

# Representing images (2D signals) as superposition of frequencies

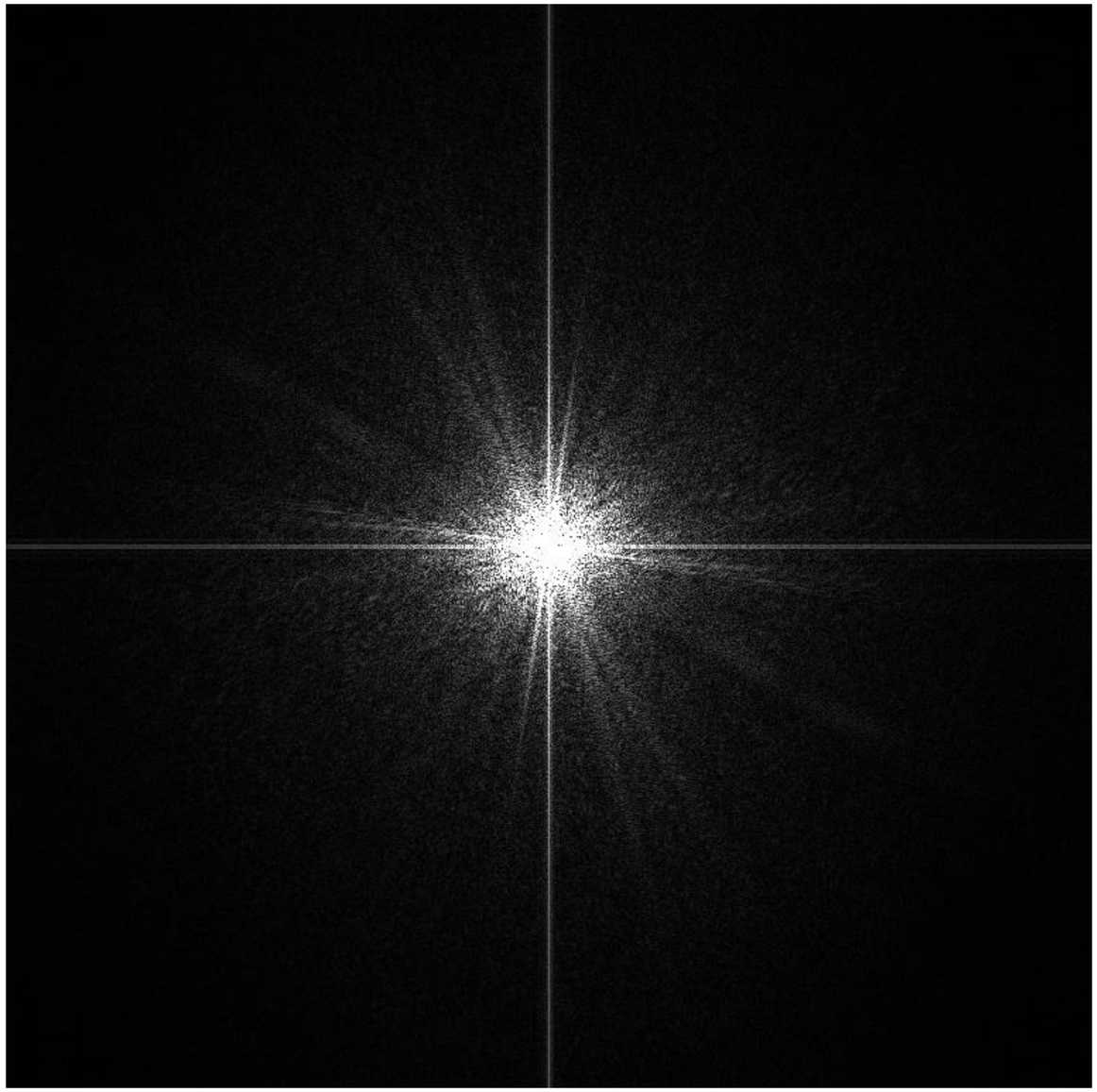



individual frequencies are 2D sinusoids

***(e.g.*** $f(x, y) = \sin(a\pi x) * \sin(b\pi x)$***)***

# Visualizing the frequency content of images

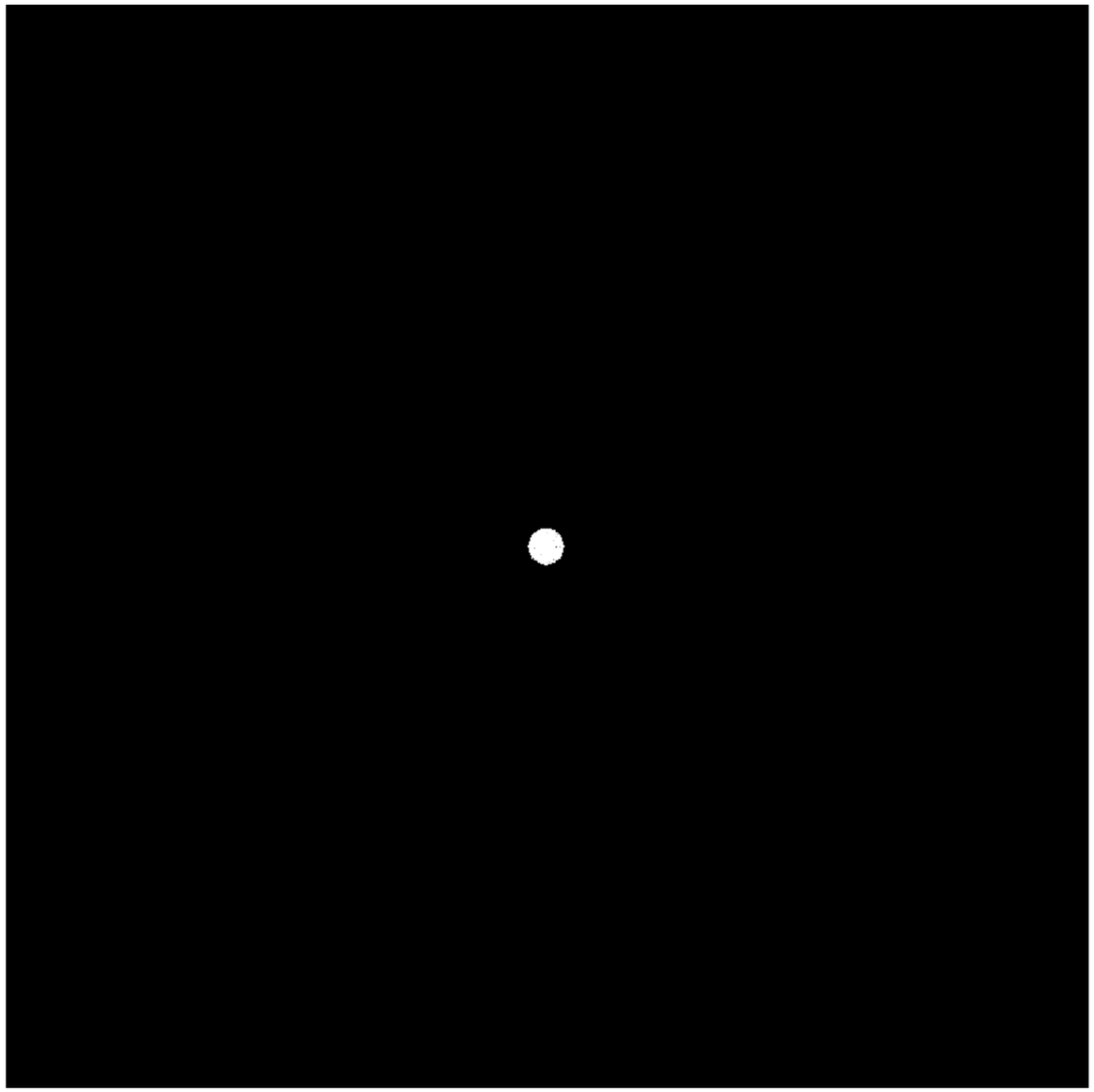
Spatial domain image



Frequency Domain Image

# Low frequencies only

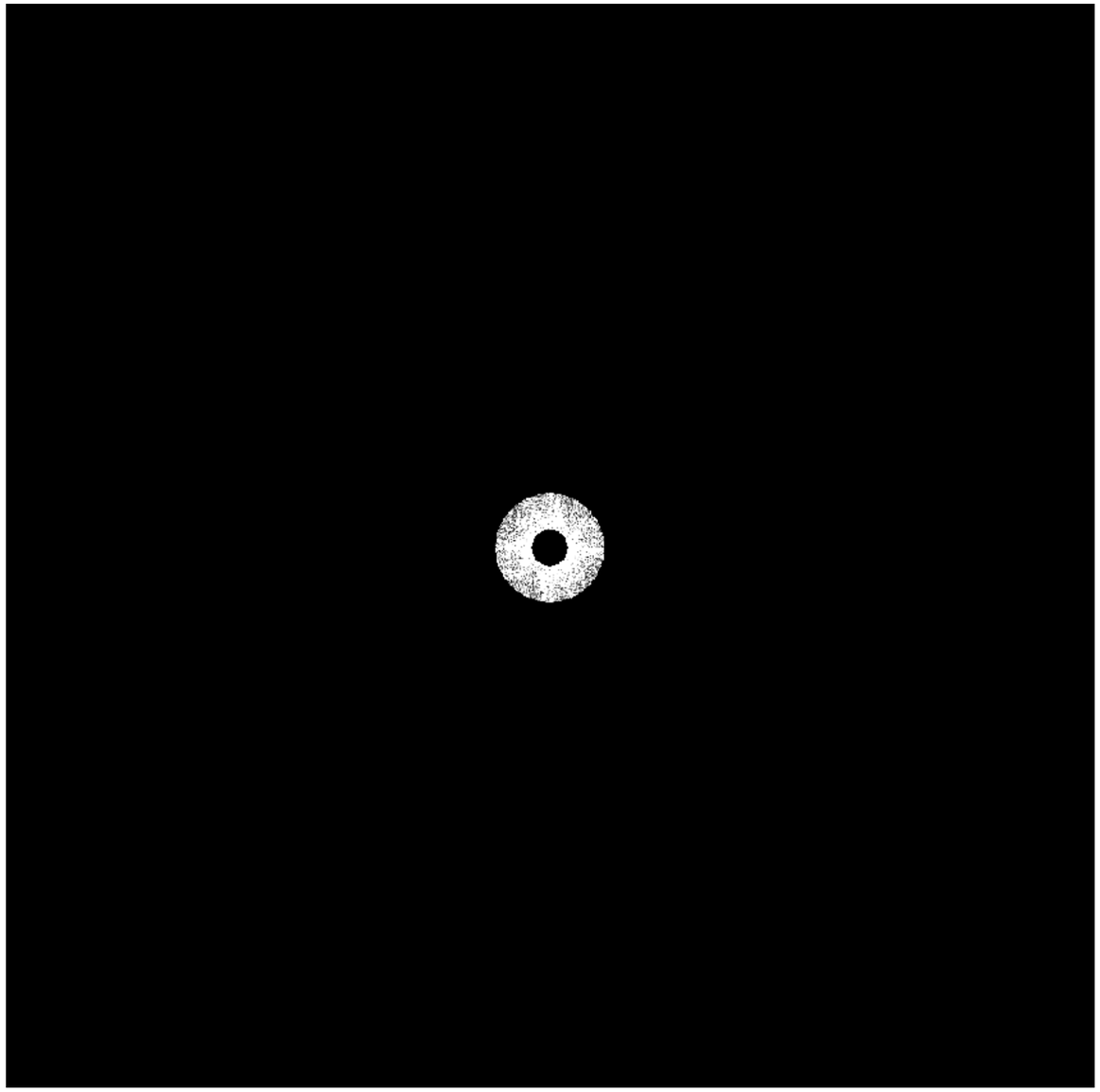
Spatial domain result



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

# Mid-range frequencies

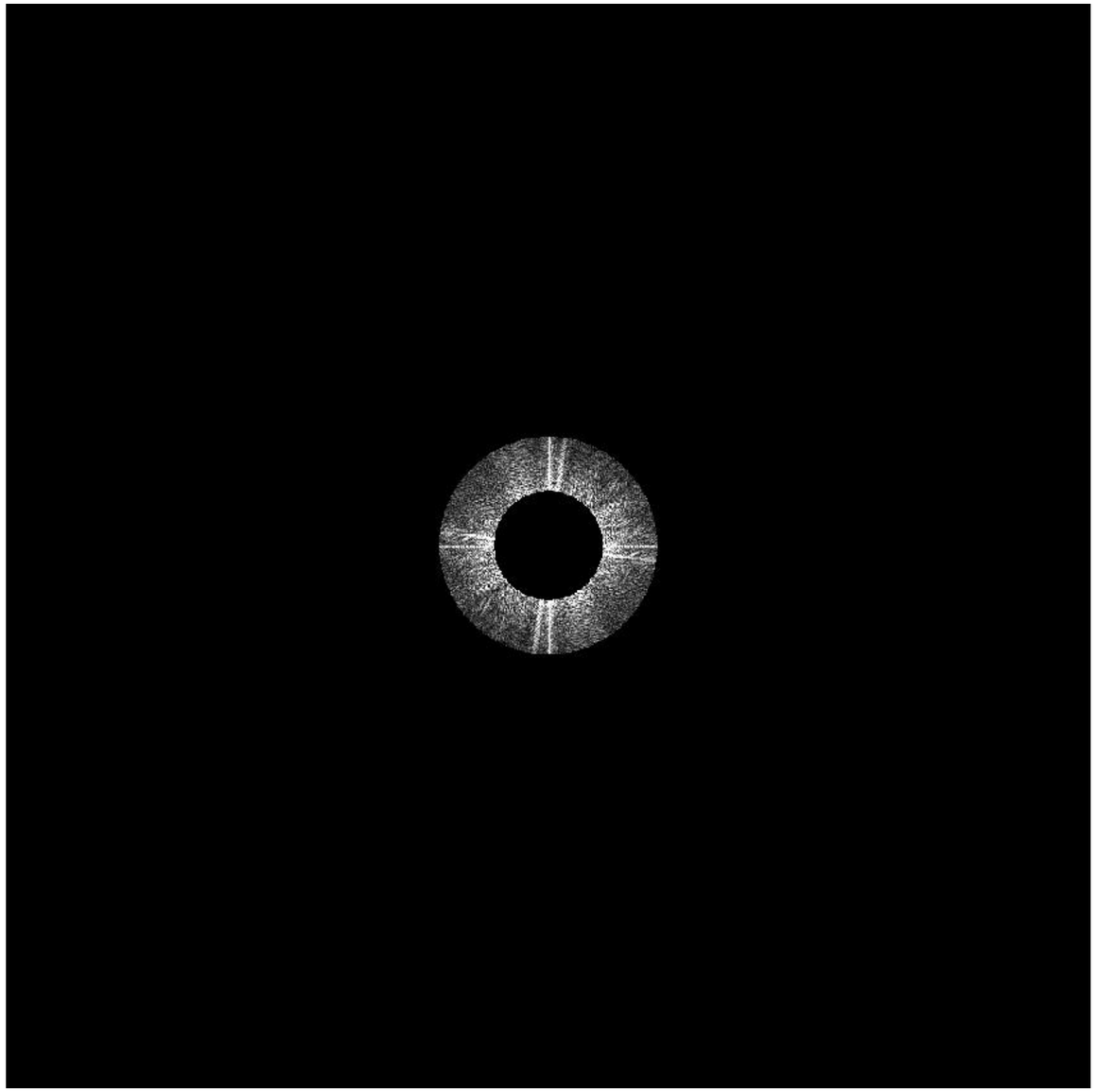
Spatial domain result



Spectrum (after band-pass filter)

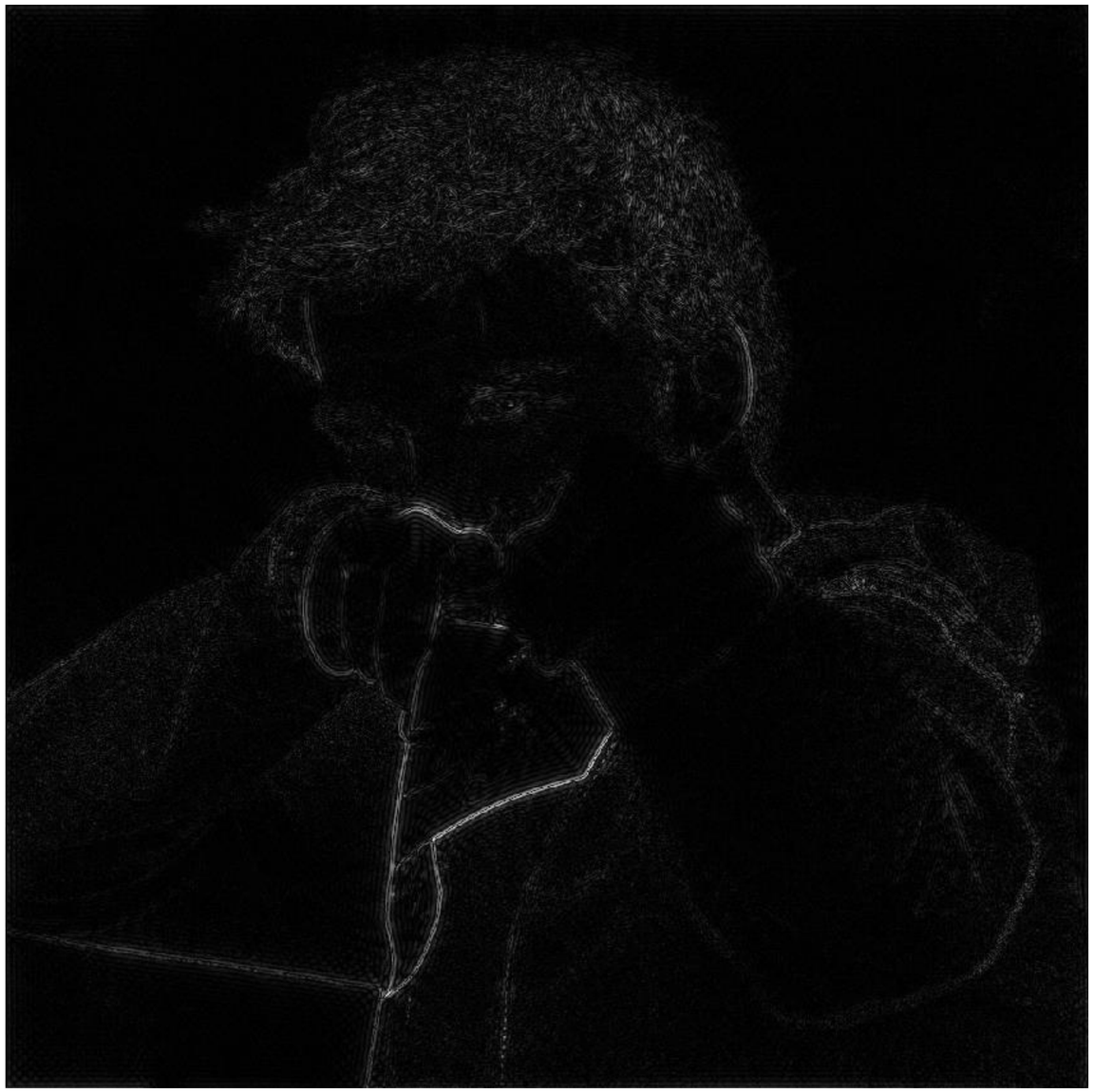
# Mid-range frequencies

Spatial domain result

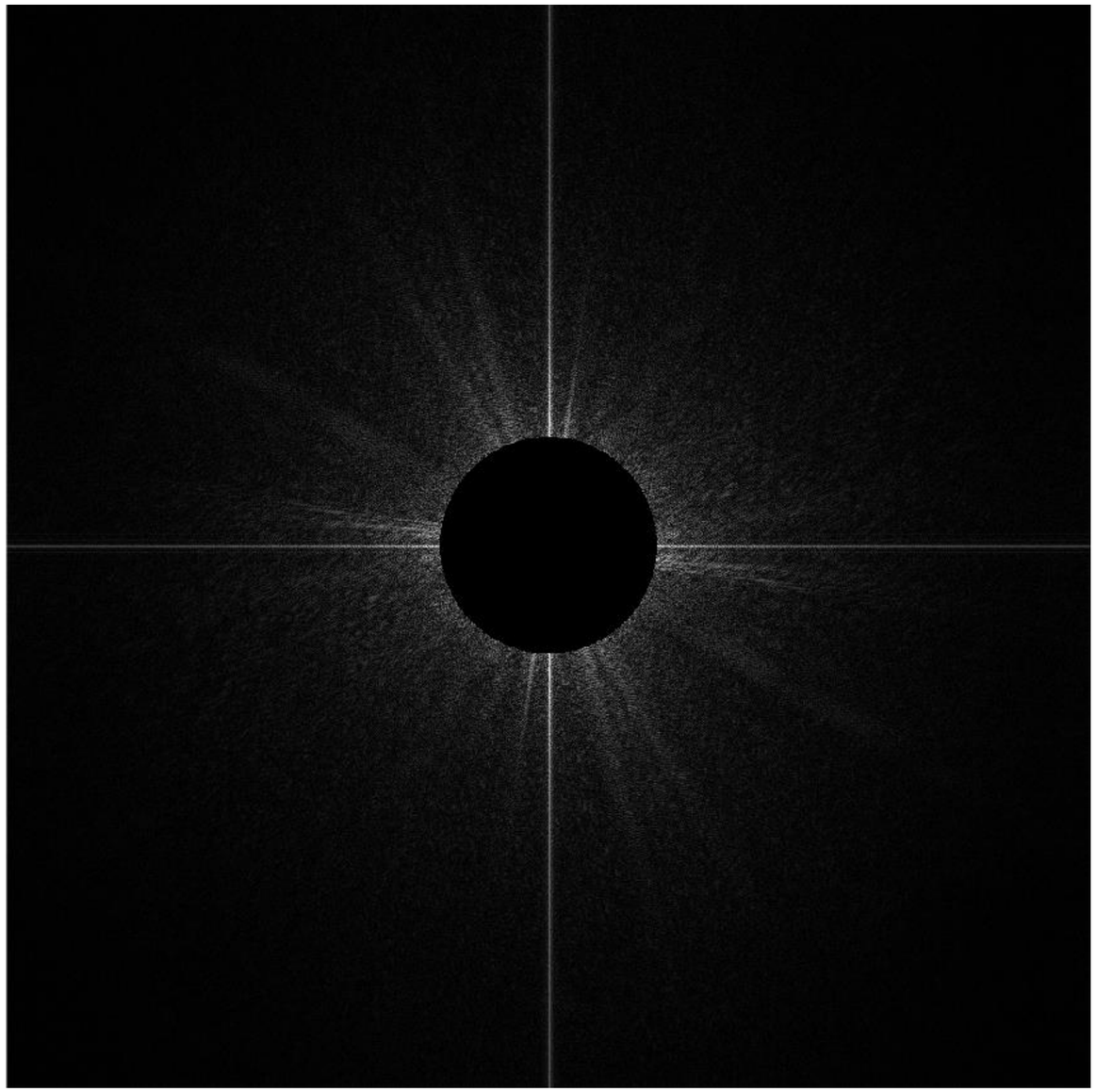


Spectrum (after band-pass filter)

# High frequencies (edges)



Spatial domain result
(strongest edges)



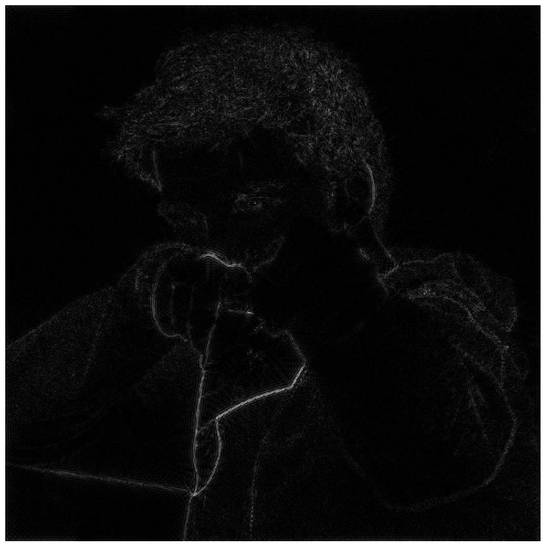
Spectrum (after high-pass filter)
All frequencies below threshold
have 0 magnitude

# An image as a sum of its frequency components

+

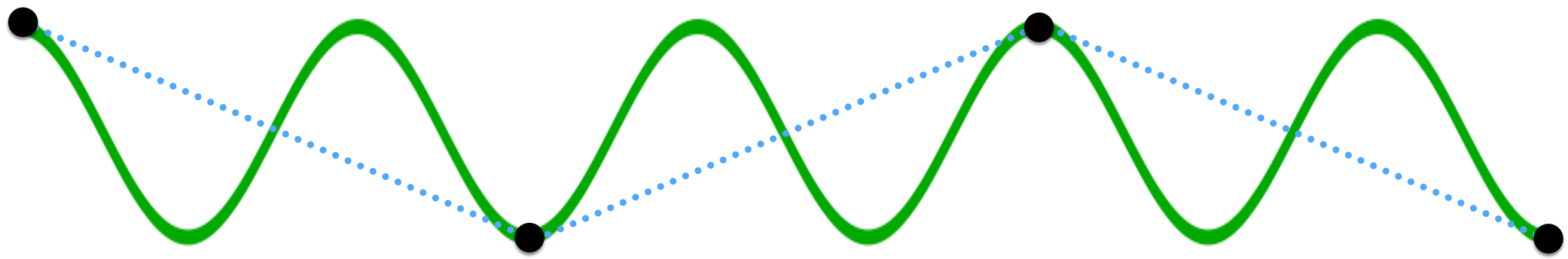
+

+



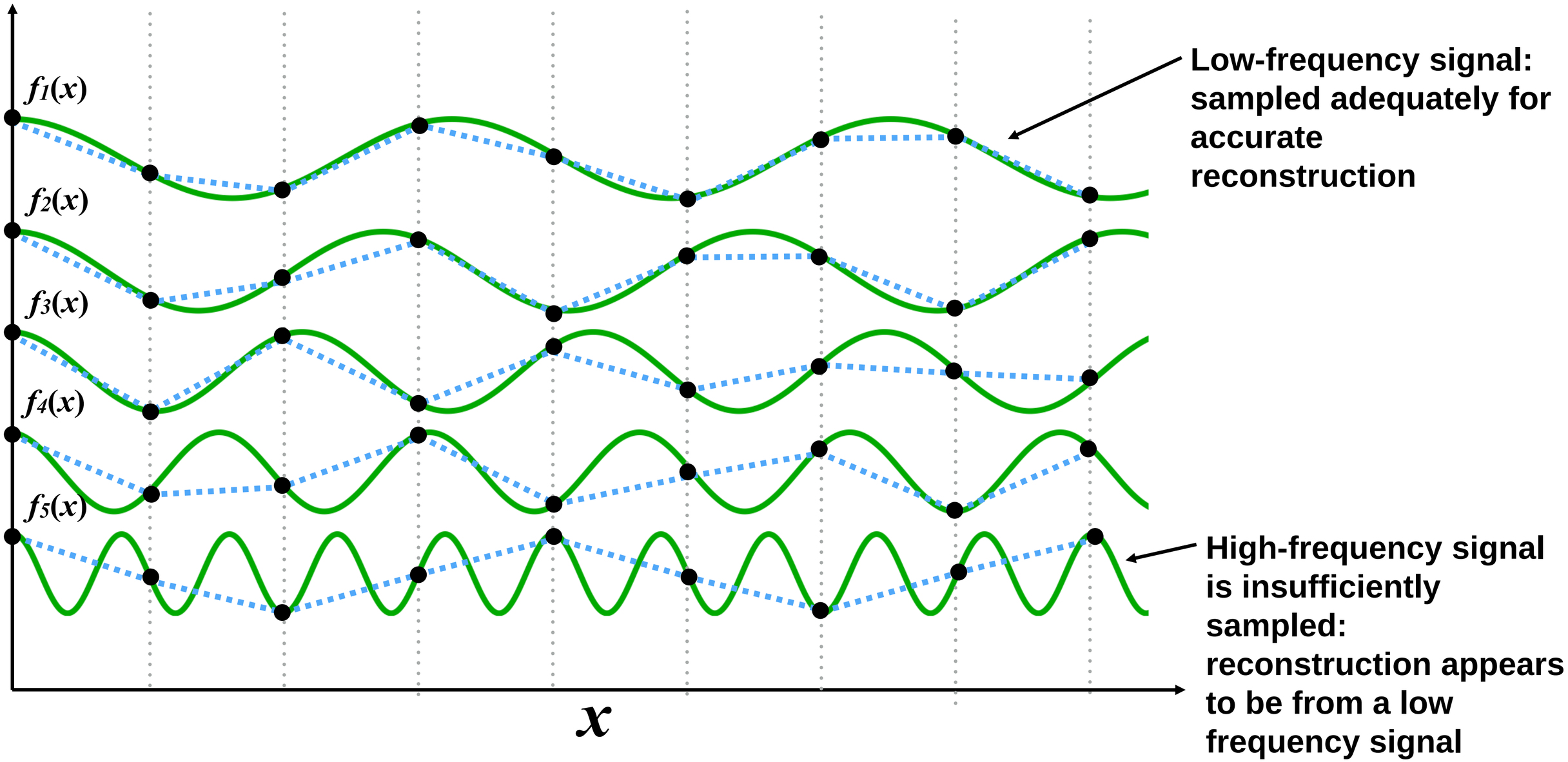
=

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



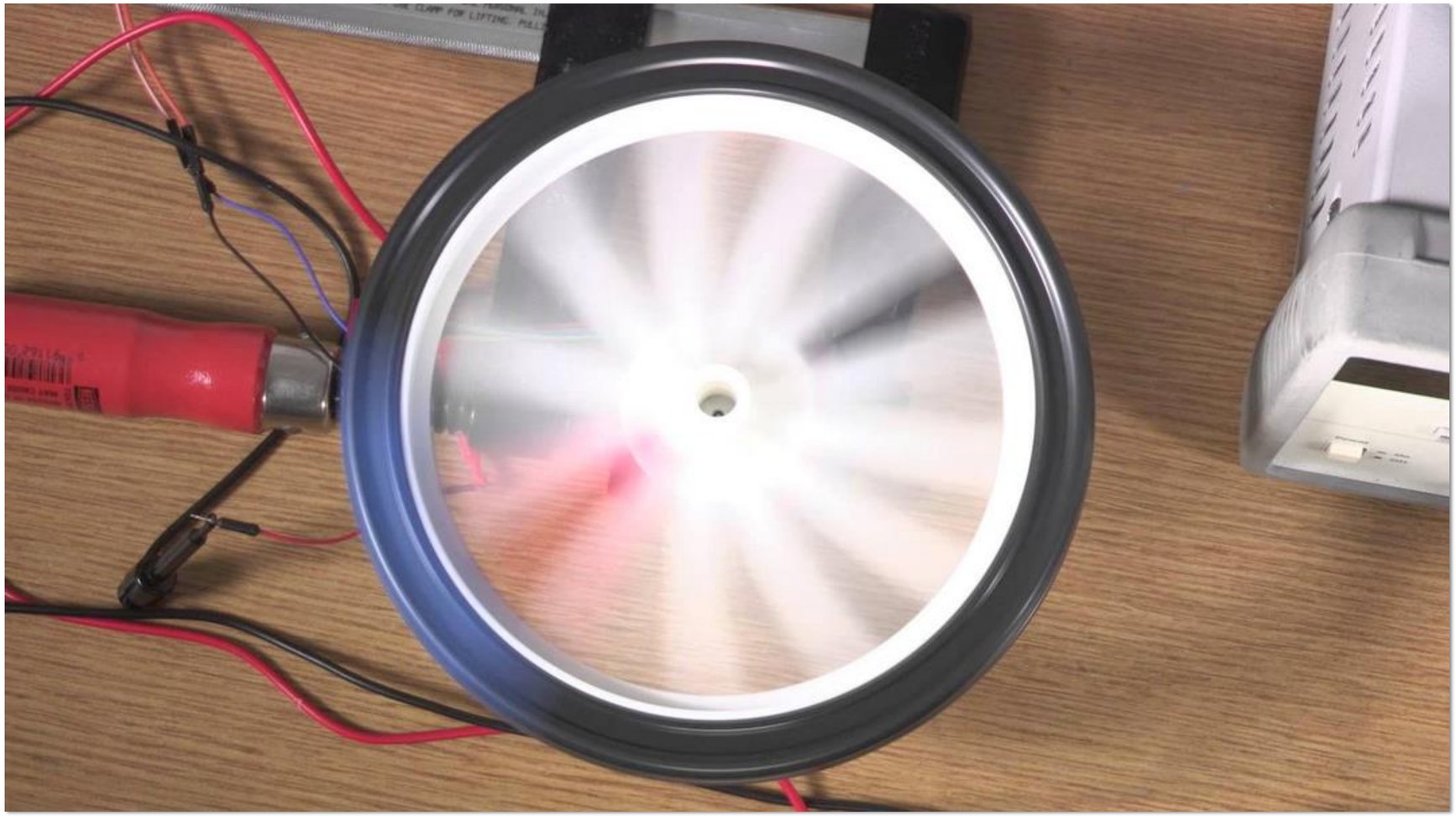
**“Aliasing”: high frequencies in the original signal masquerade as low frequencies after reconstruction (due to undersampling)**

# 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)**

# Temporal aliasing: wagon wheel effect



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

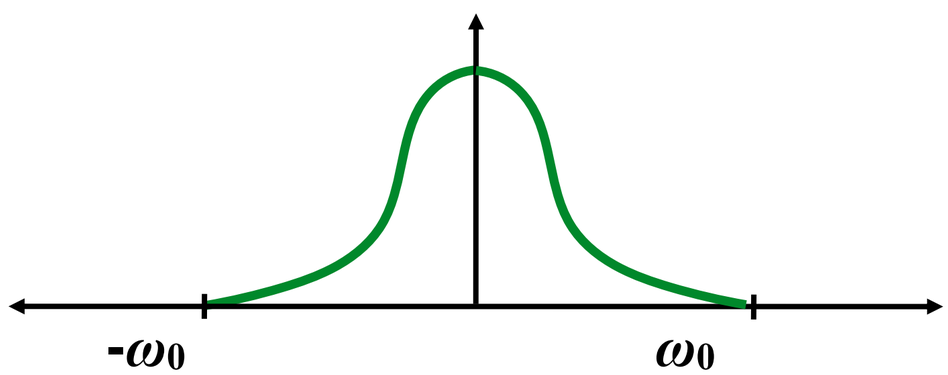
**https://www.youtube.com/watch?v=VNftf5qLpiA**

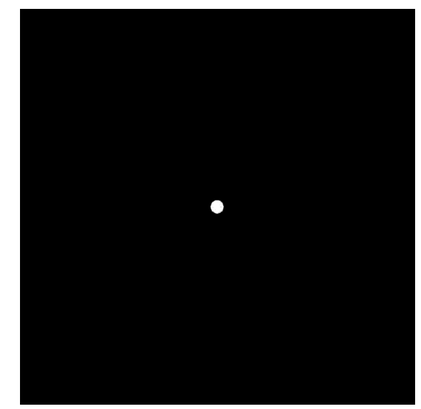
# Sampling rate, high-frequency signals & aliasing

- So, how densely should you be sampling?

# Nyquist-Shannon theorem

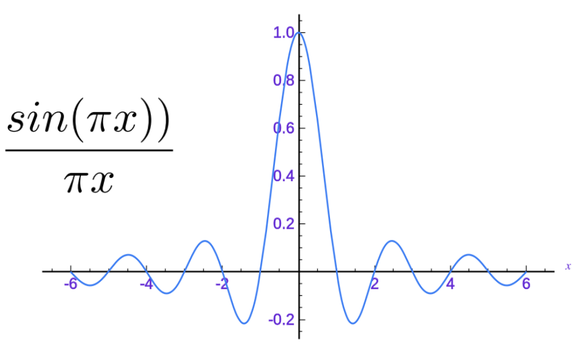
- Consider a band-limited signal: has no frequencies above $\omega_0$
  - 1D: consider low-pass filtered audio signal
  - 2D: recall the blurred image example from a few slides ago



- 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)

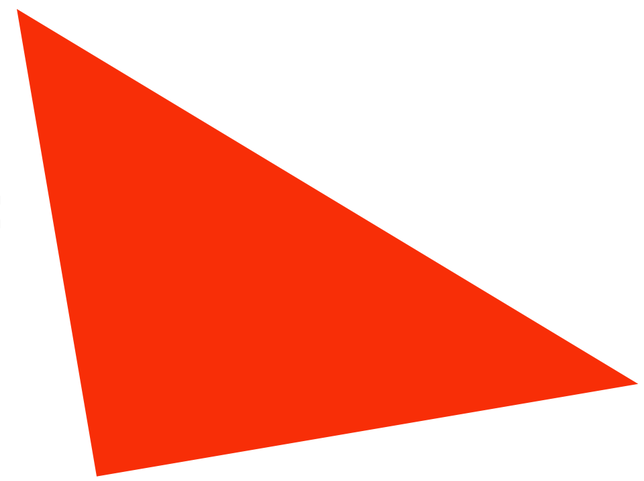
$$sinc(x) = \frac{sin(\pi x))}{\pi x}$$



# Challenges of sampling-based approaches in graphics

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

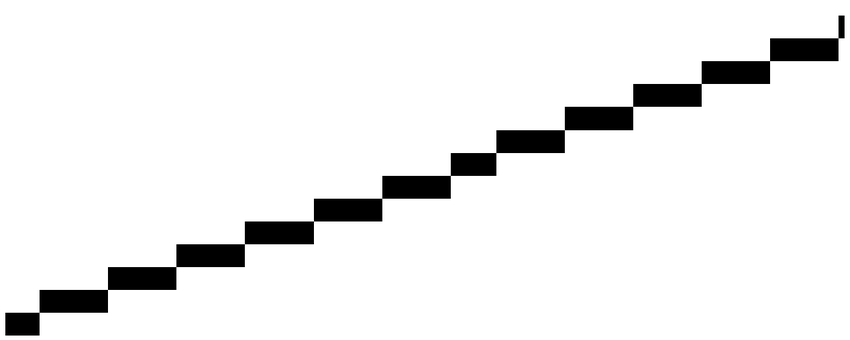
**Hint:**

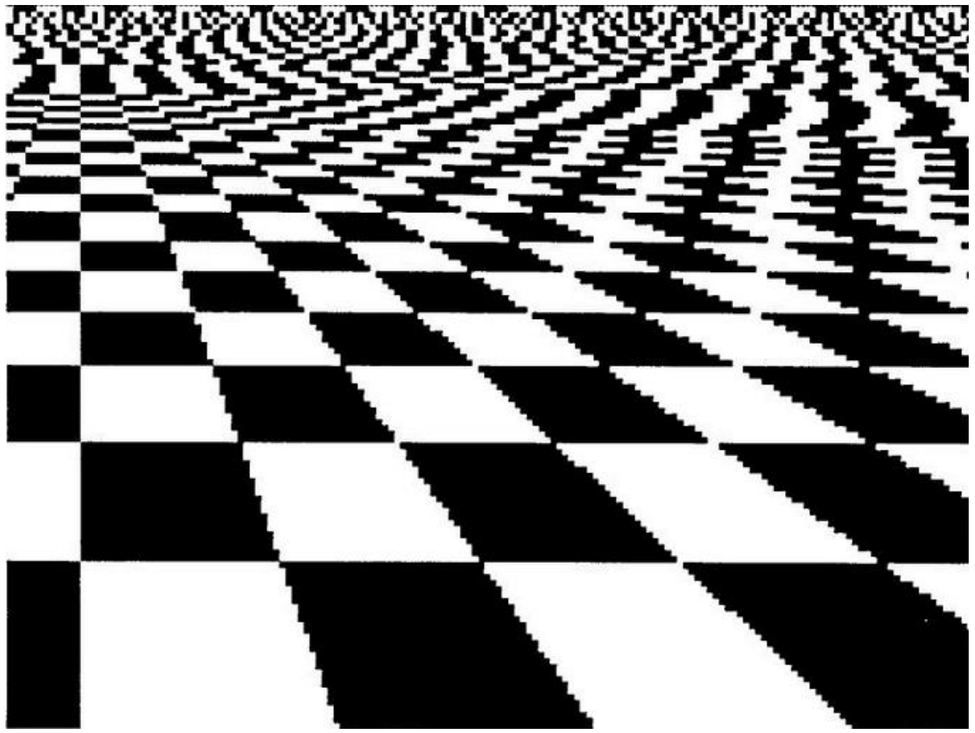


- **Also, infinite extent of “ideal” reconstruction filter (sinc) is impractical for performant implementations. Why?**

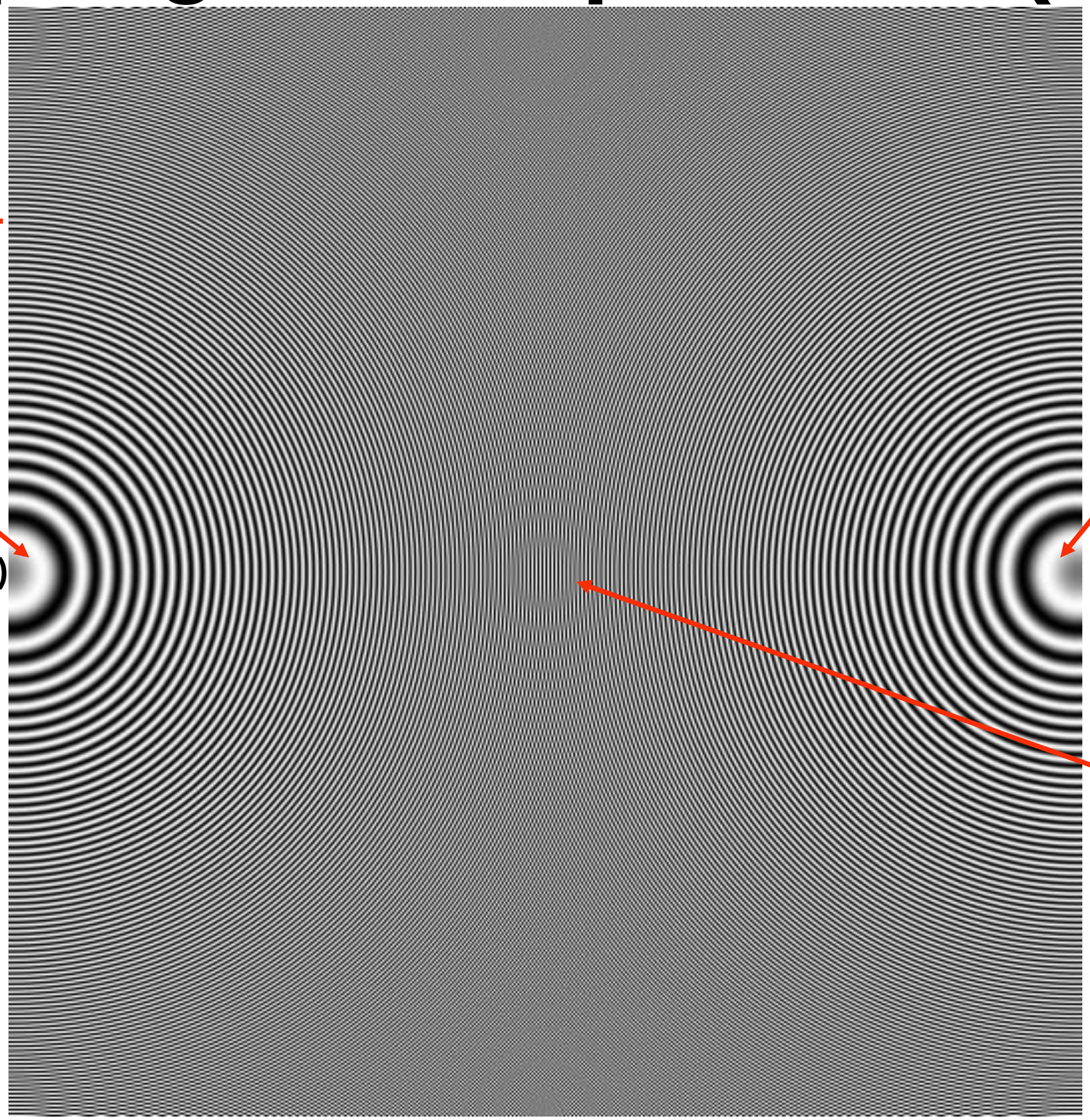
# Aliasing artifacts in images

- **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)

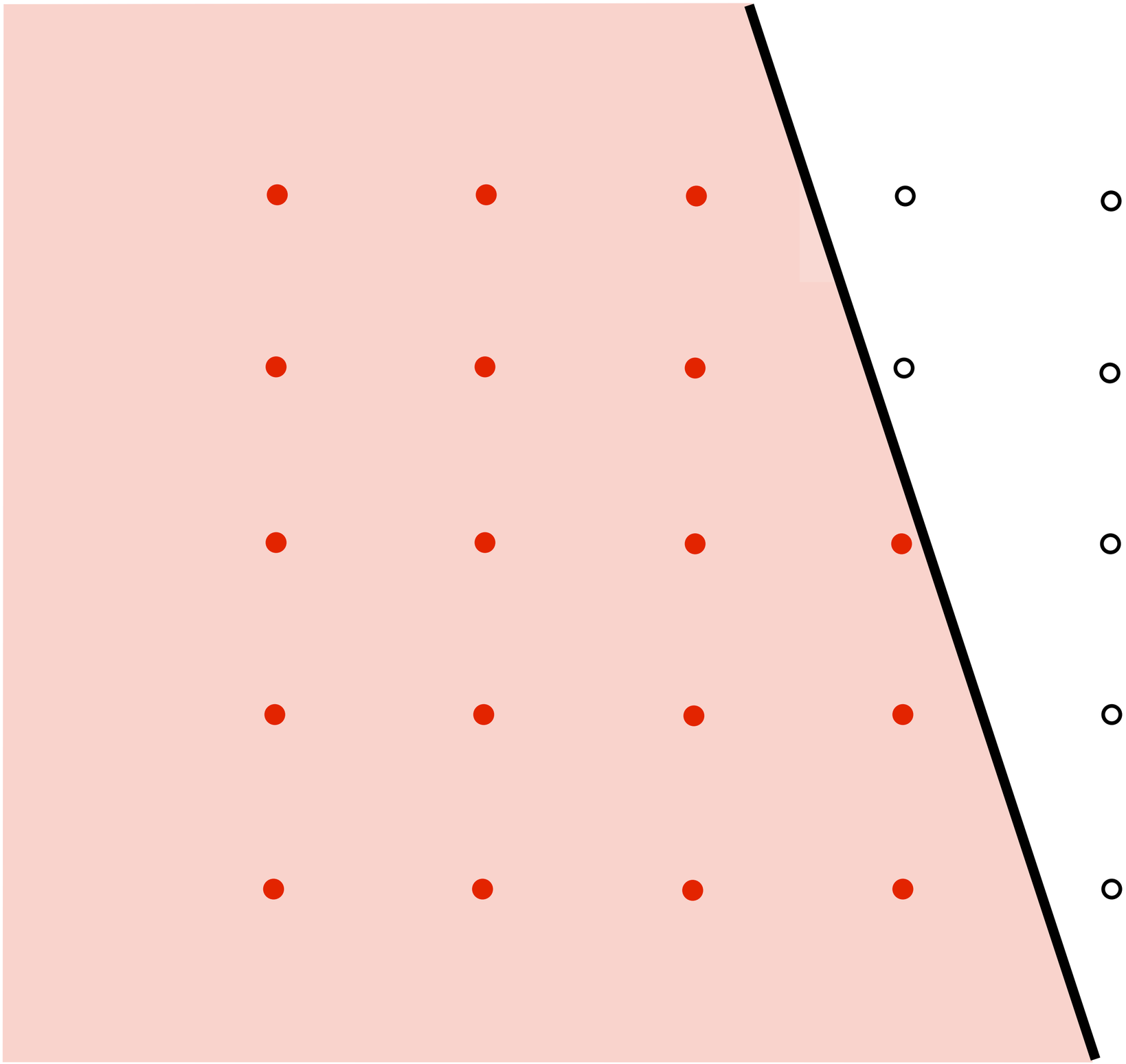
Rings on right: aliasing from undersampling high frequency oscillation and then resampling back to slide resolution

Middle: (interaction between actual signal and aliased resconstruction)

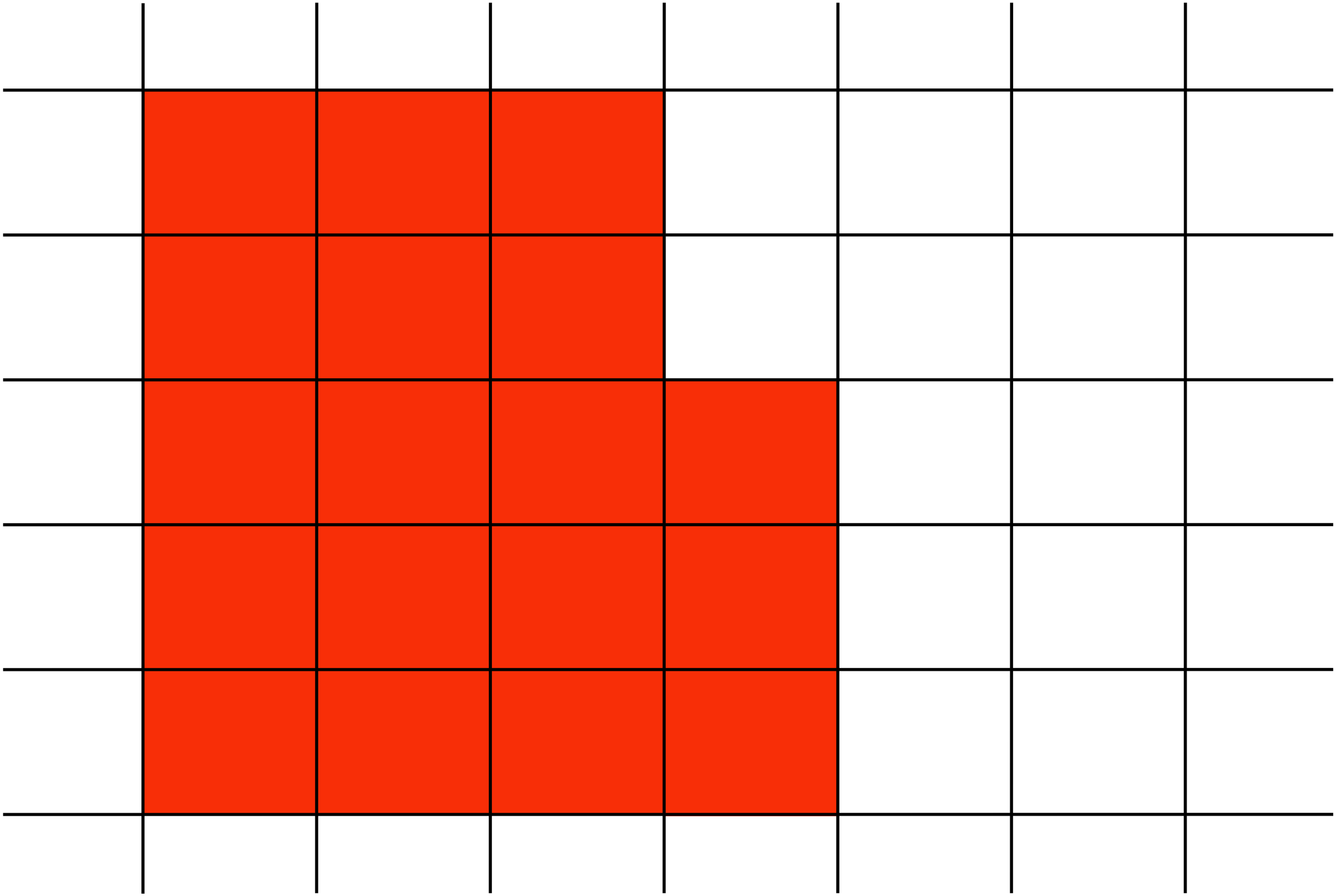
Figure credit: Pat Hanrahan and Bryce Summers

# Recall: the real coverage signal

# Initial coverage sampling rate (1 sample per pixel)

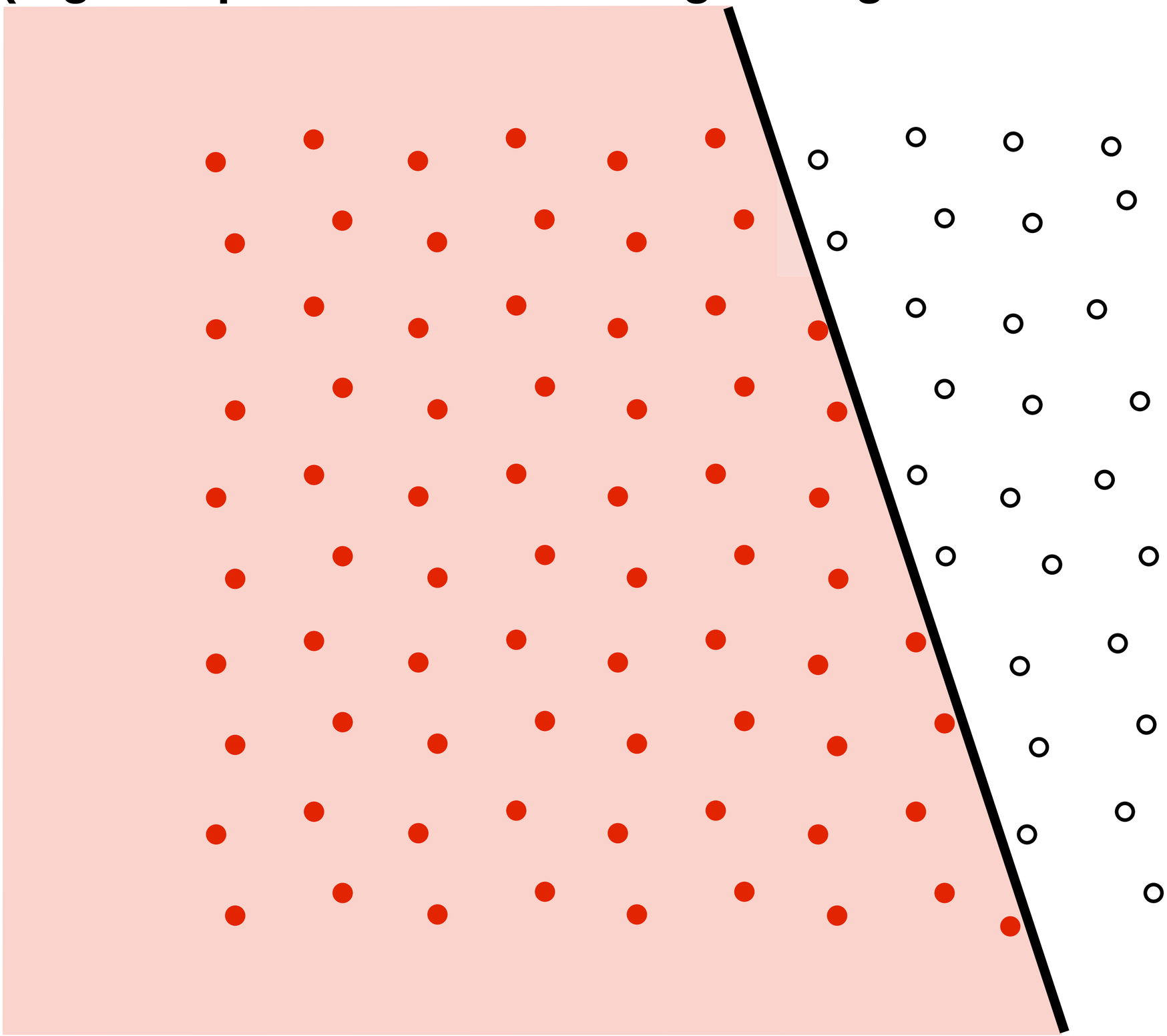


# We see this on the screen



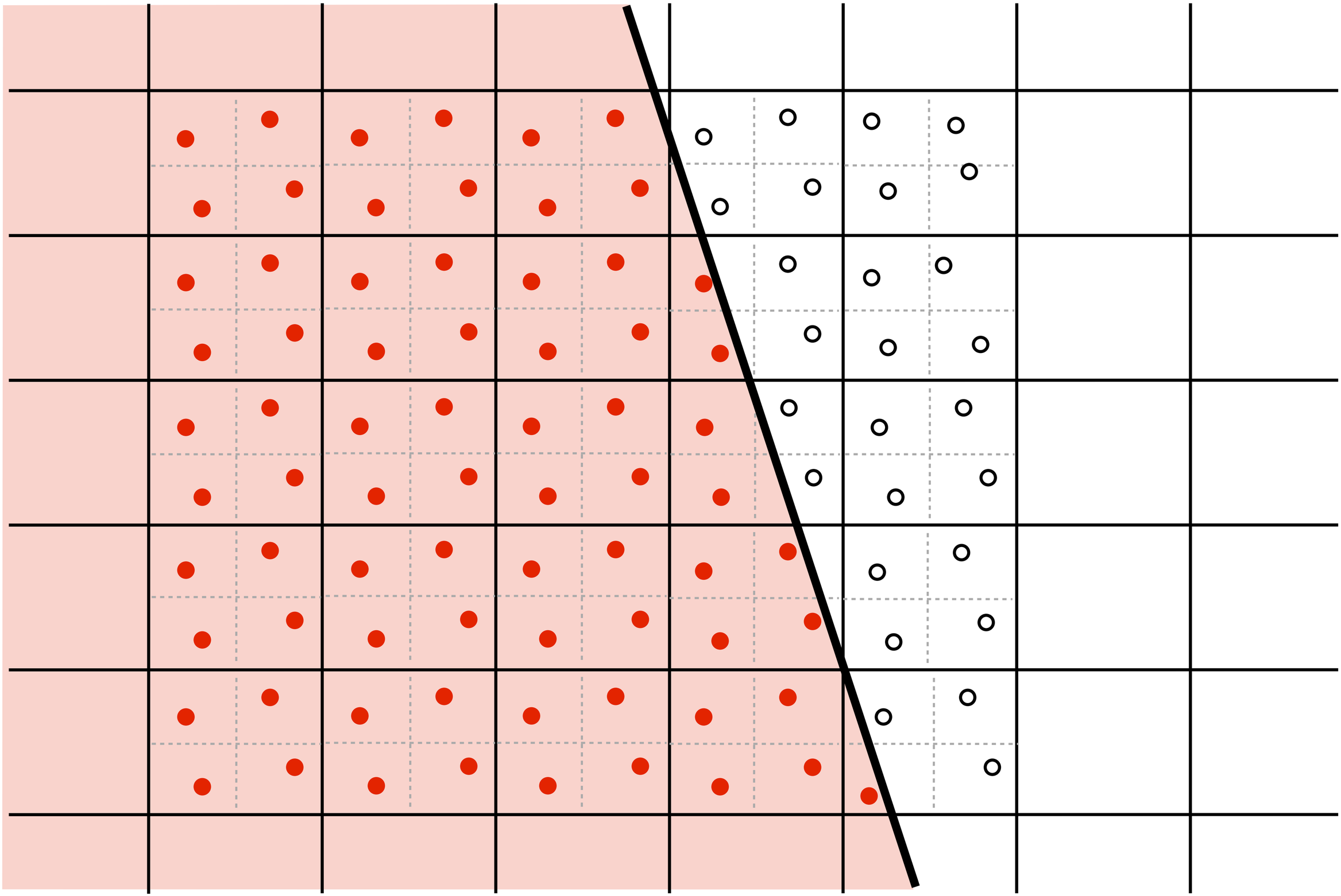
# Increase density of sampling coverage signal

## (high frequencies exist in original signal because of triangle edges)



# Supersampling

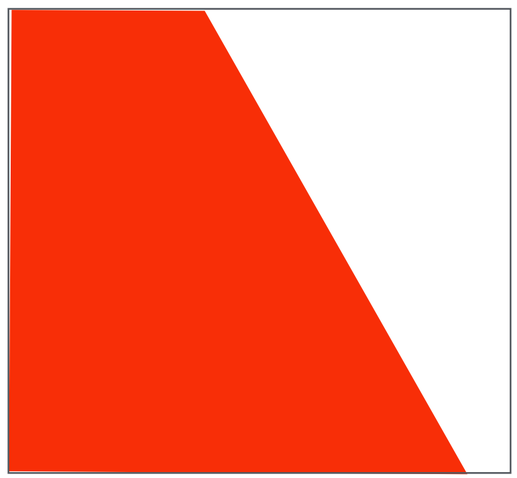
**Example: stratified sampling using four samples per pixel**



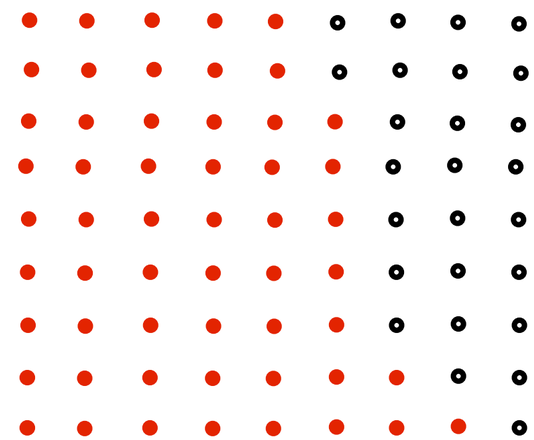
Ok, but now we have more samples than pixels!

# Resampling

## Converting from one discrete sampled representation to another

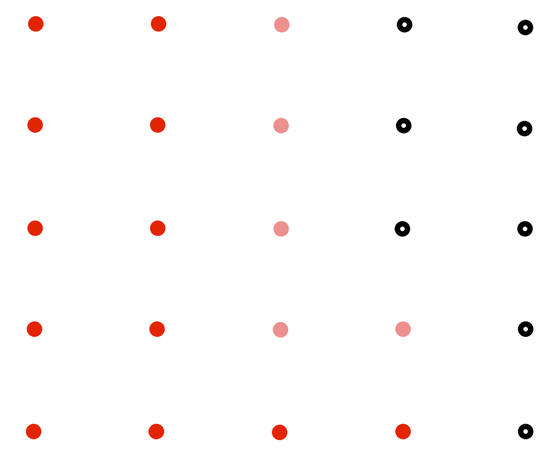


Original signal
(high frequency edge)



Dense sampling of
initial signal

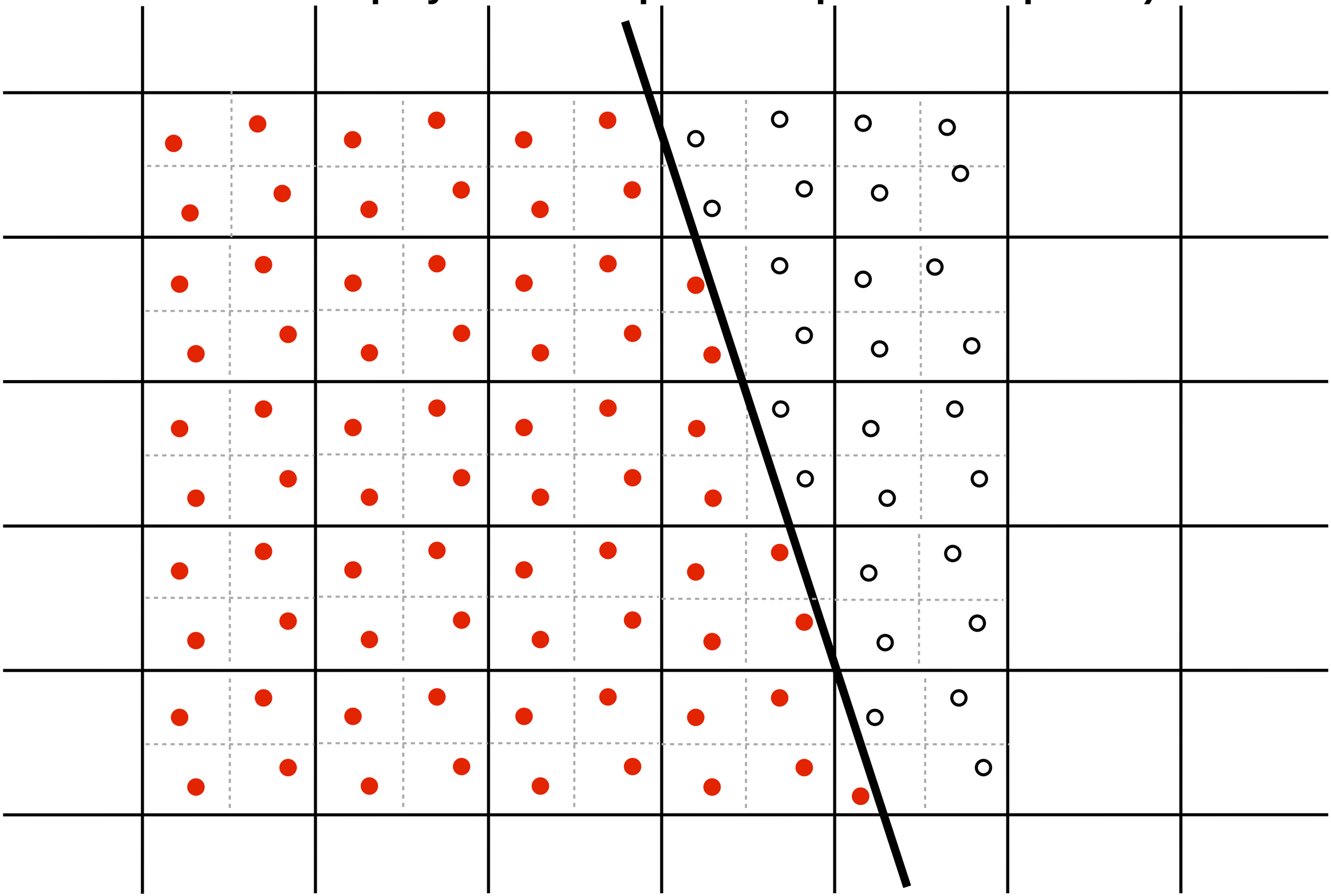
Reconstructed signal
(lacks high frequencies)



Coarsely sampled
reconstructed signal

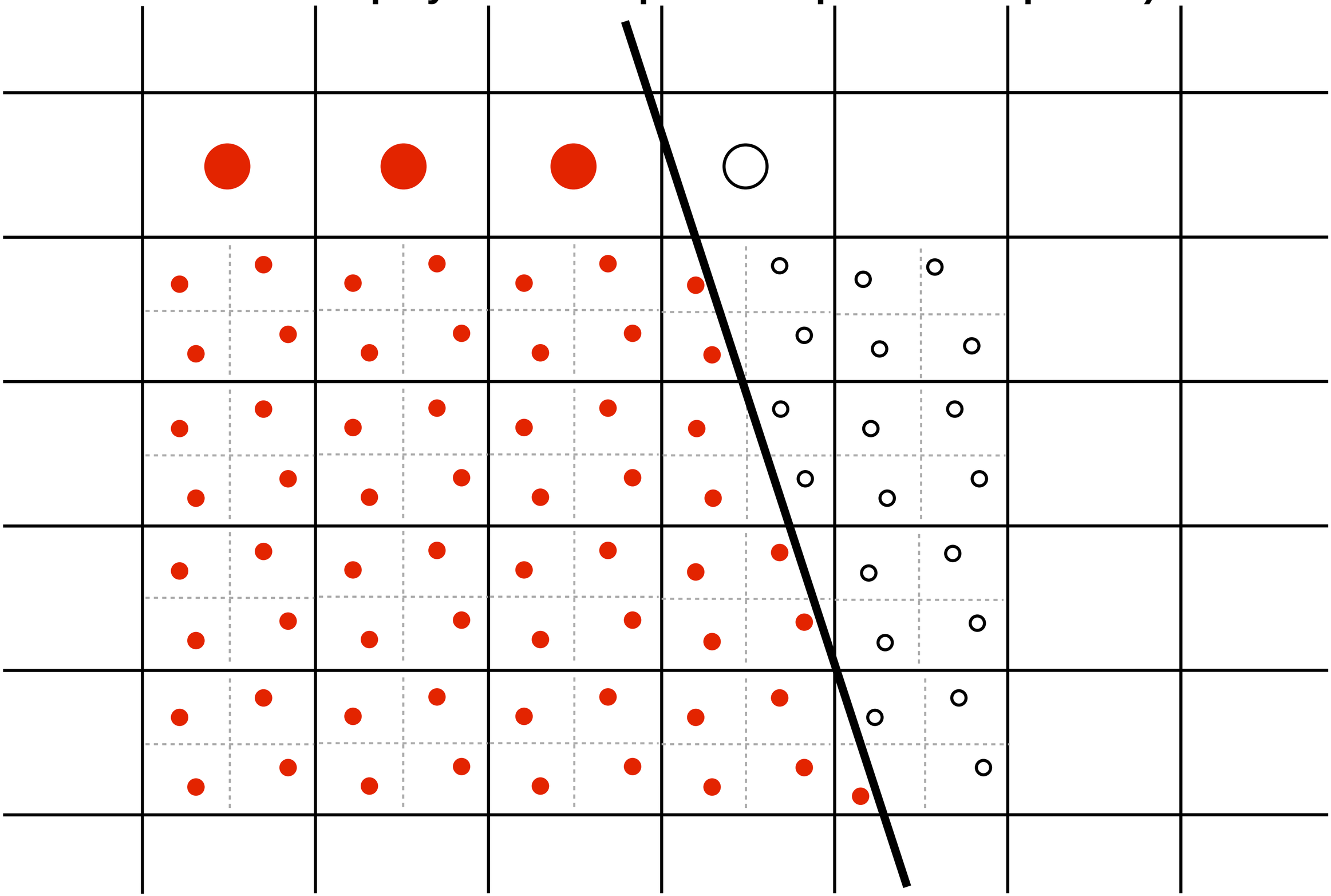
# Resample to display's pixel resolution

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

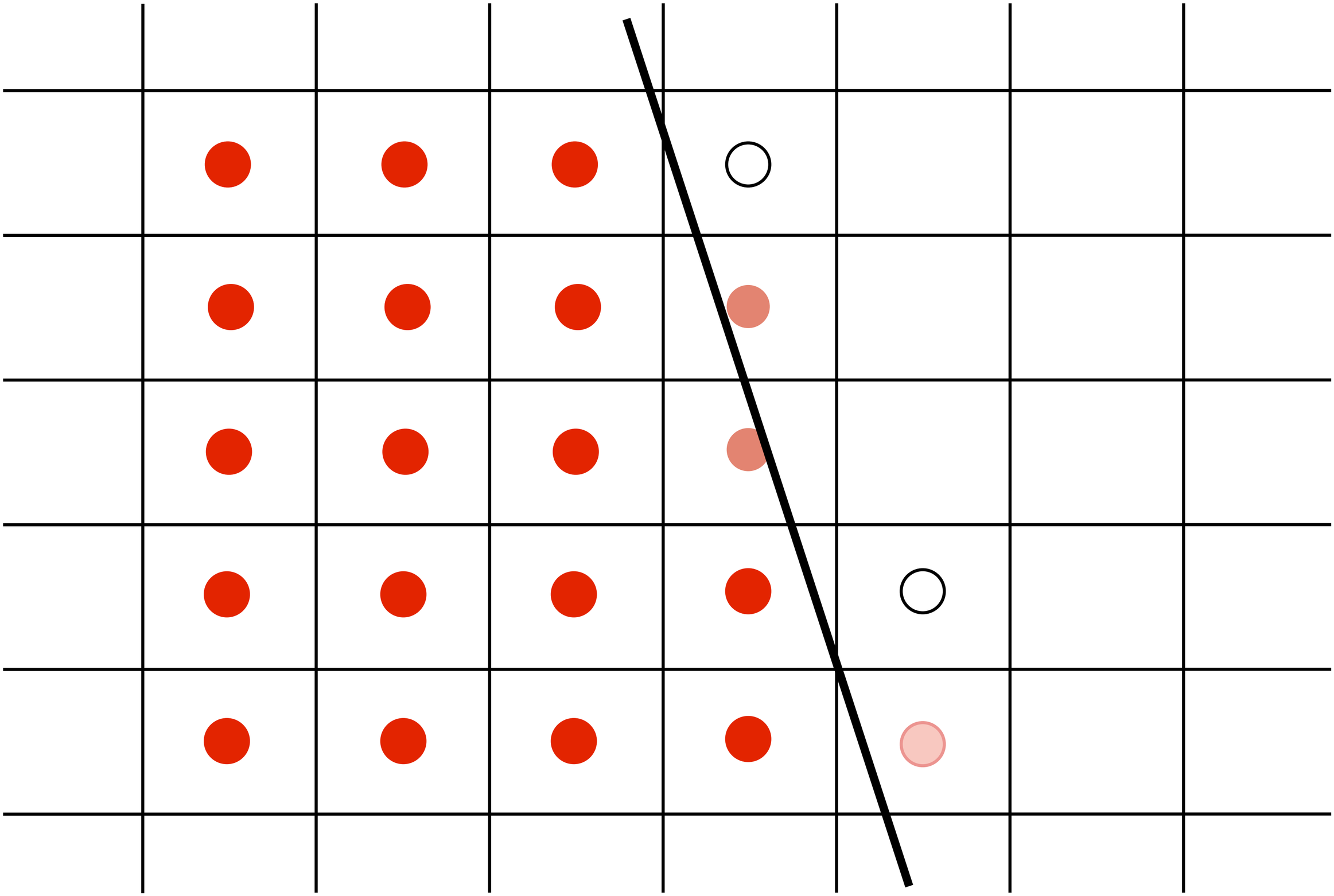


# Resample to display's pixel resolution

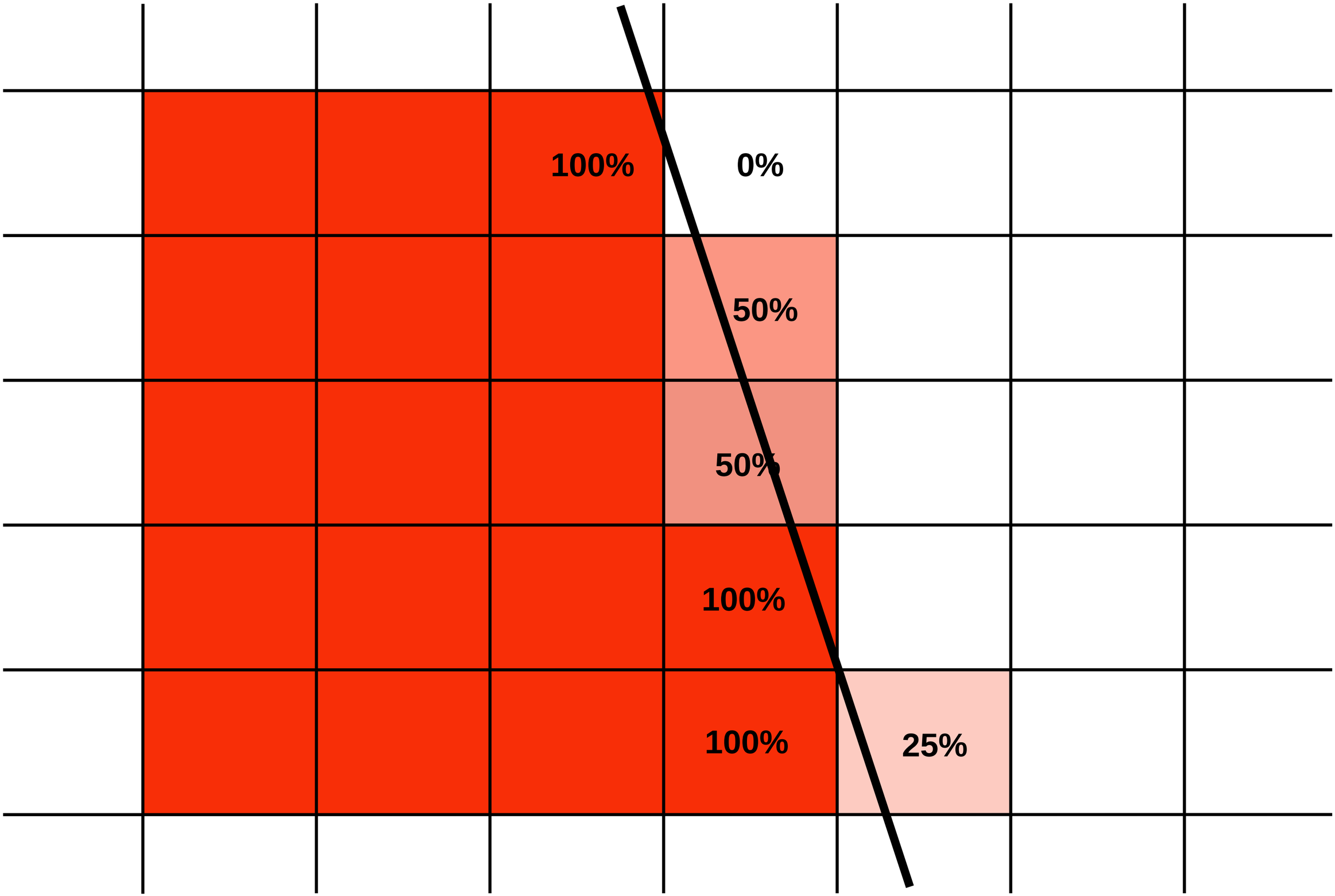
**(Because a screen displays one sample value per screen pixel...)**



# Resample to display's pixel resolution

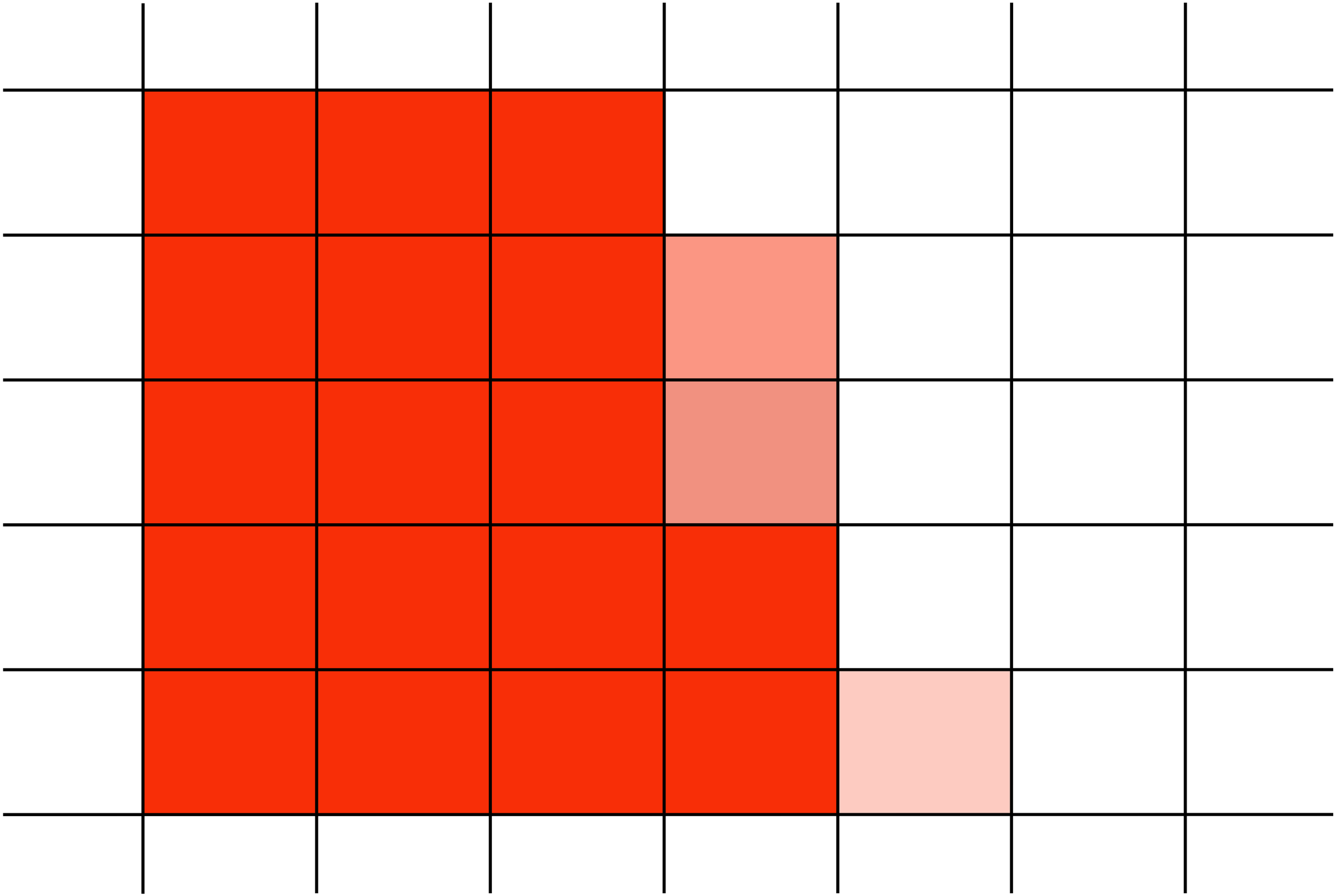


# Displayed result (note anti-aliased edges)



# 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

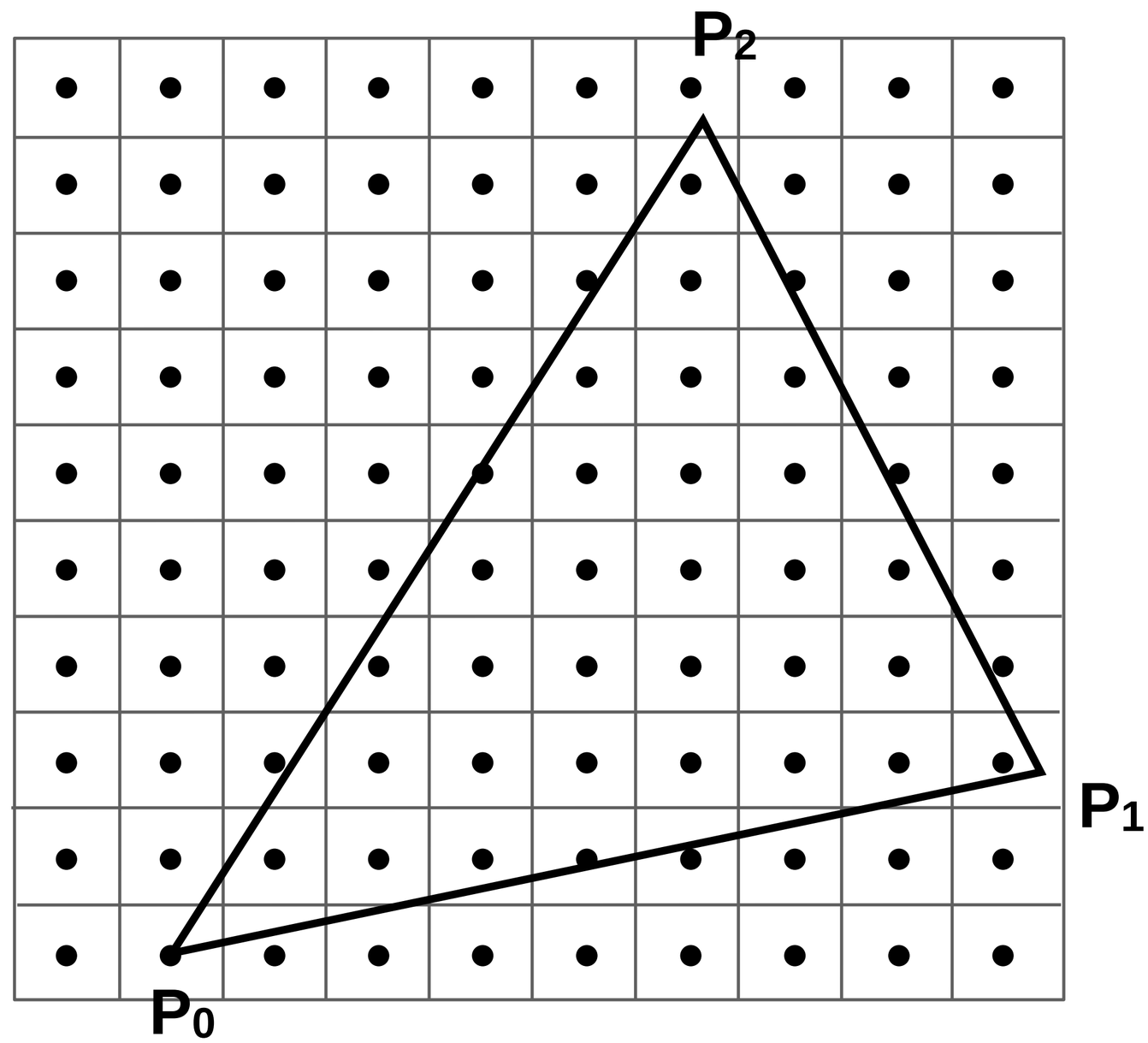
## Compute triangle edge equations from projected positions of vertices

$P_i = (X_i, Y_i)$

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

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

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



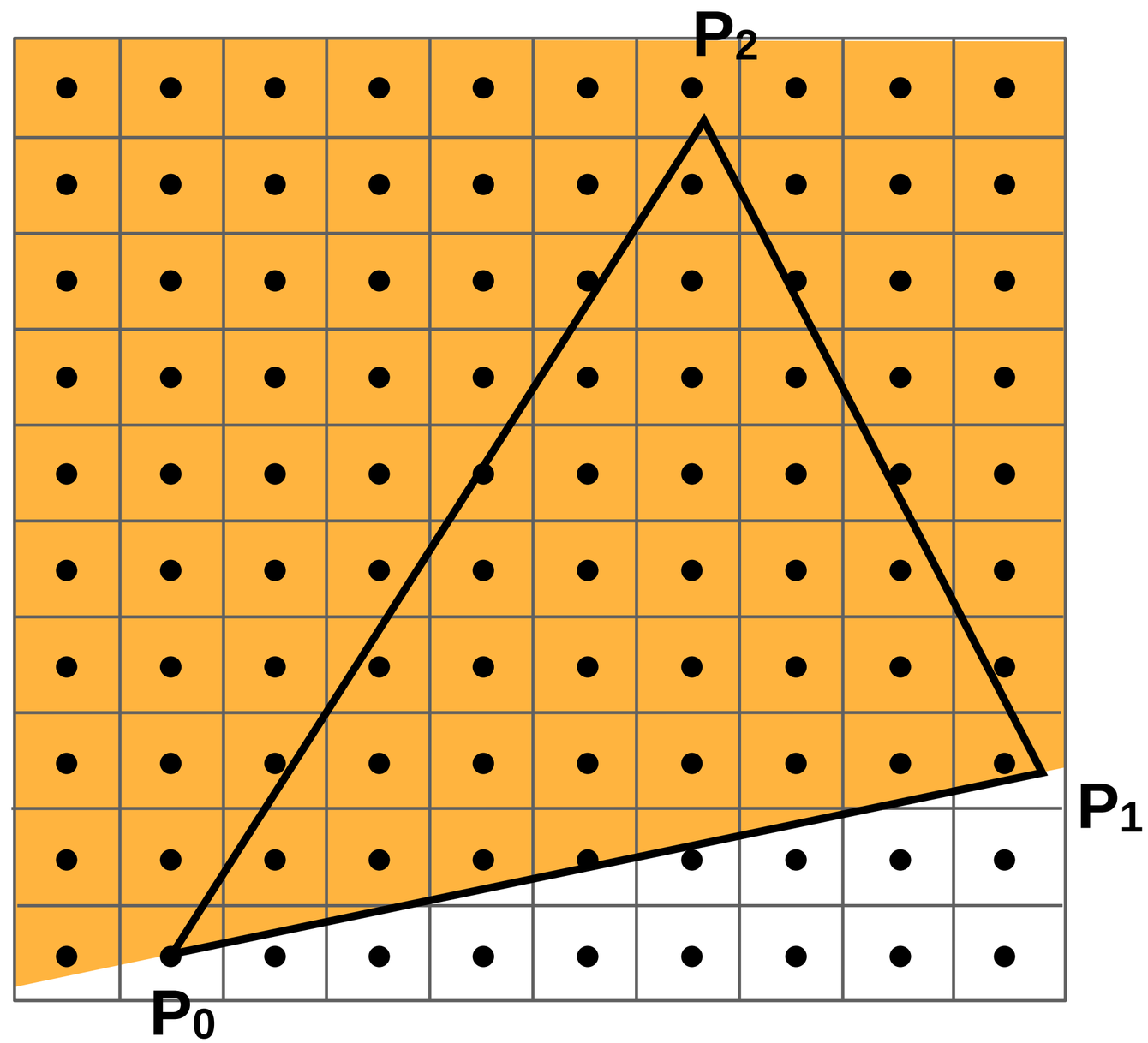
# Point-in-triangle test

$P_i = (X_i, Y_i)$

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

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

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



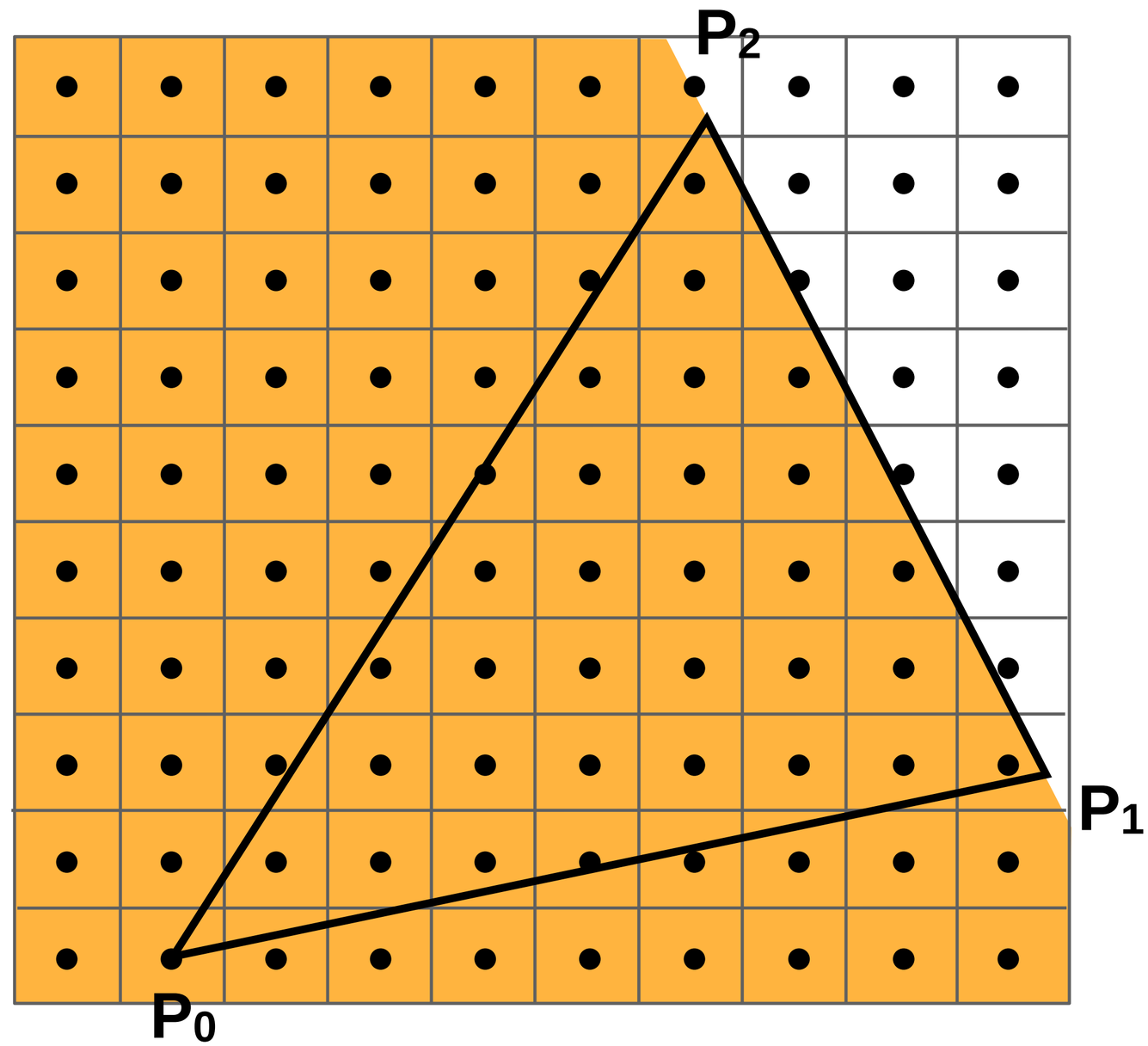
# Point-in-triangle test

$P_i = (X_i, Y_i)$

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

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

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



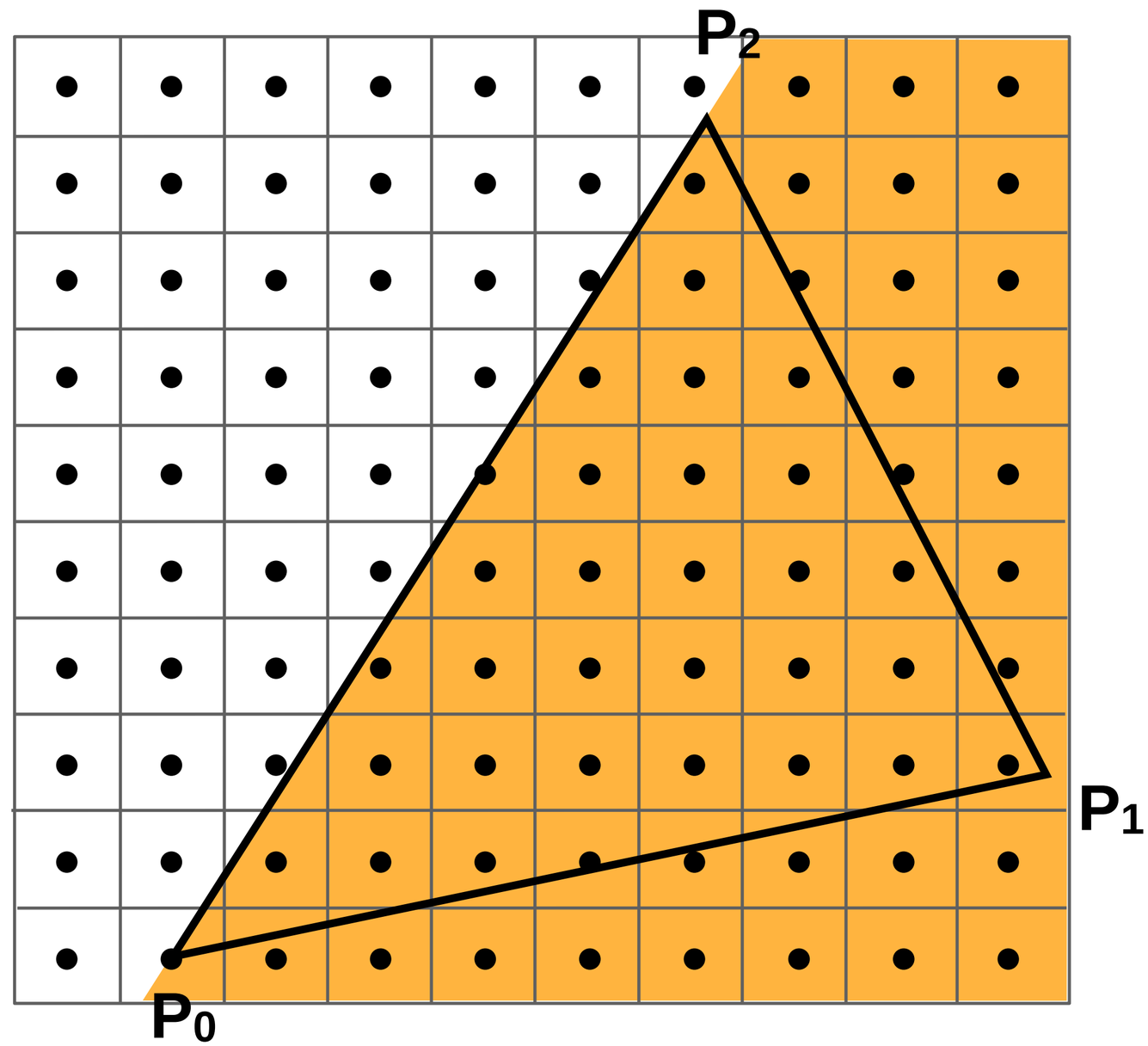
# Point-in-triangle test

$P_i = (X_i, Y_i)$

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

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

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

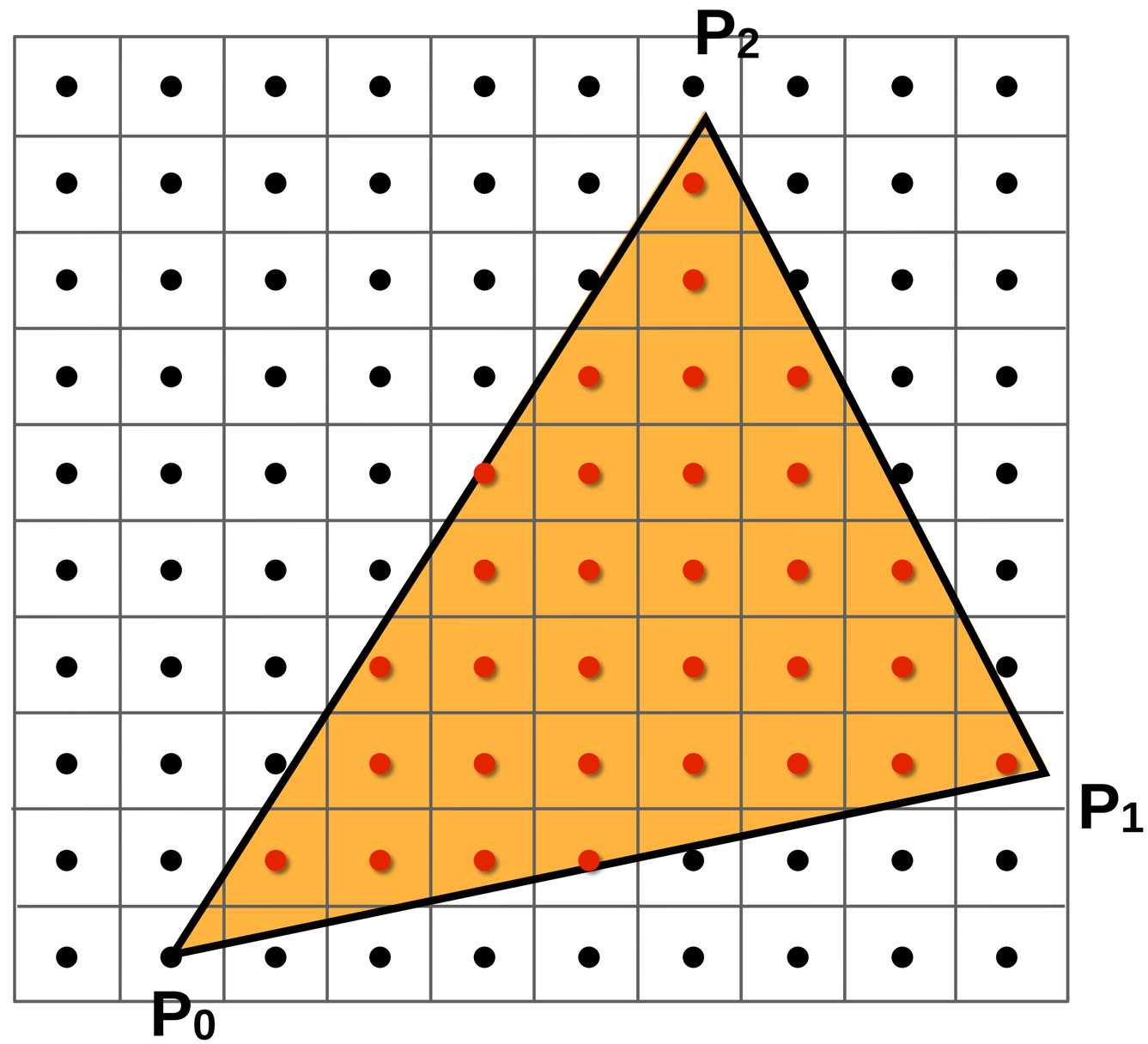


# Point-in-triangle test

**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)**

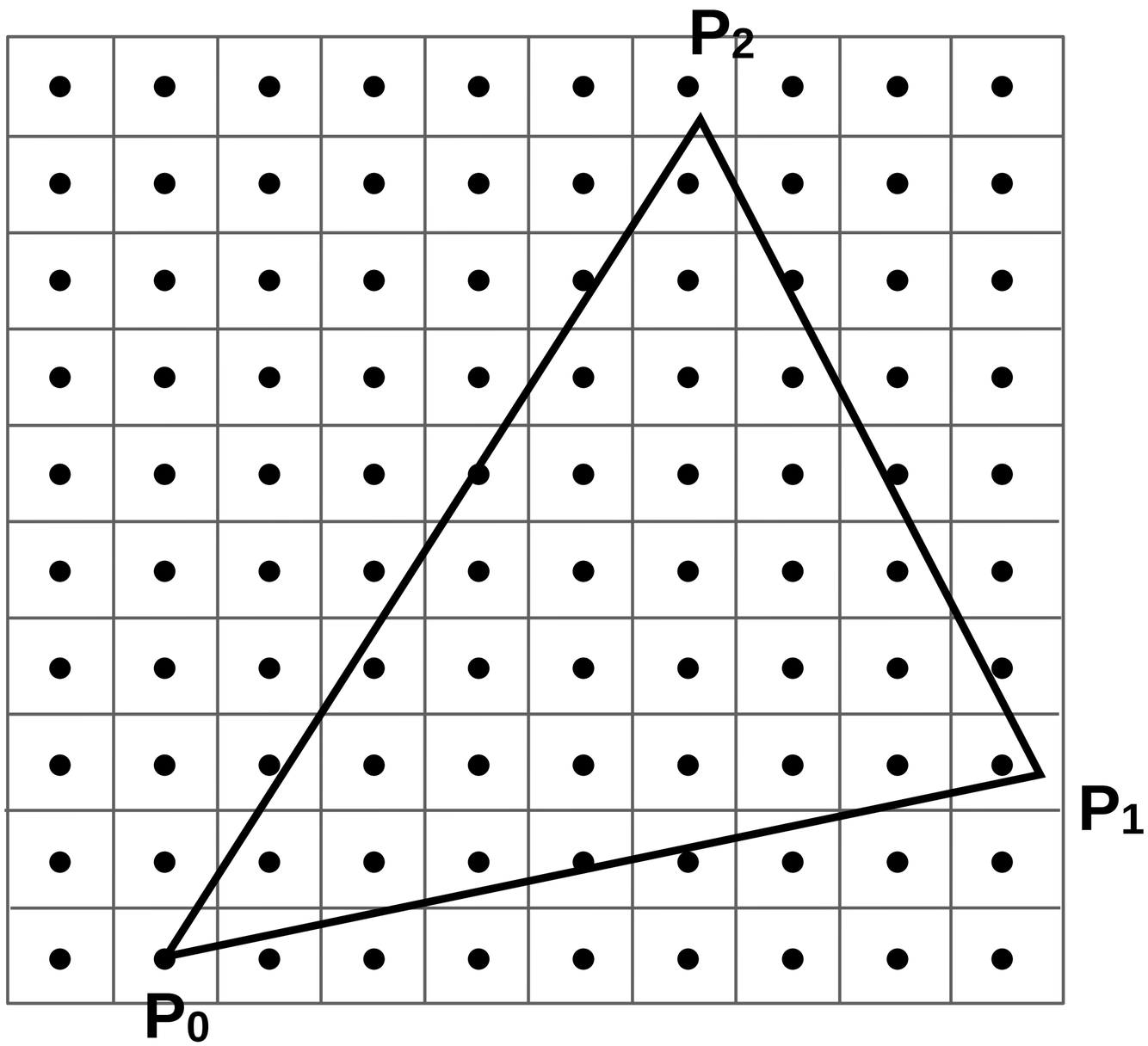



**Sample points inside triangle are highlighted red.**

# Point-in-triangle test

**Which points should we test?**

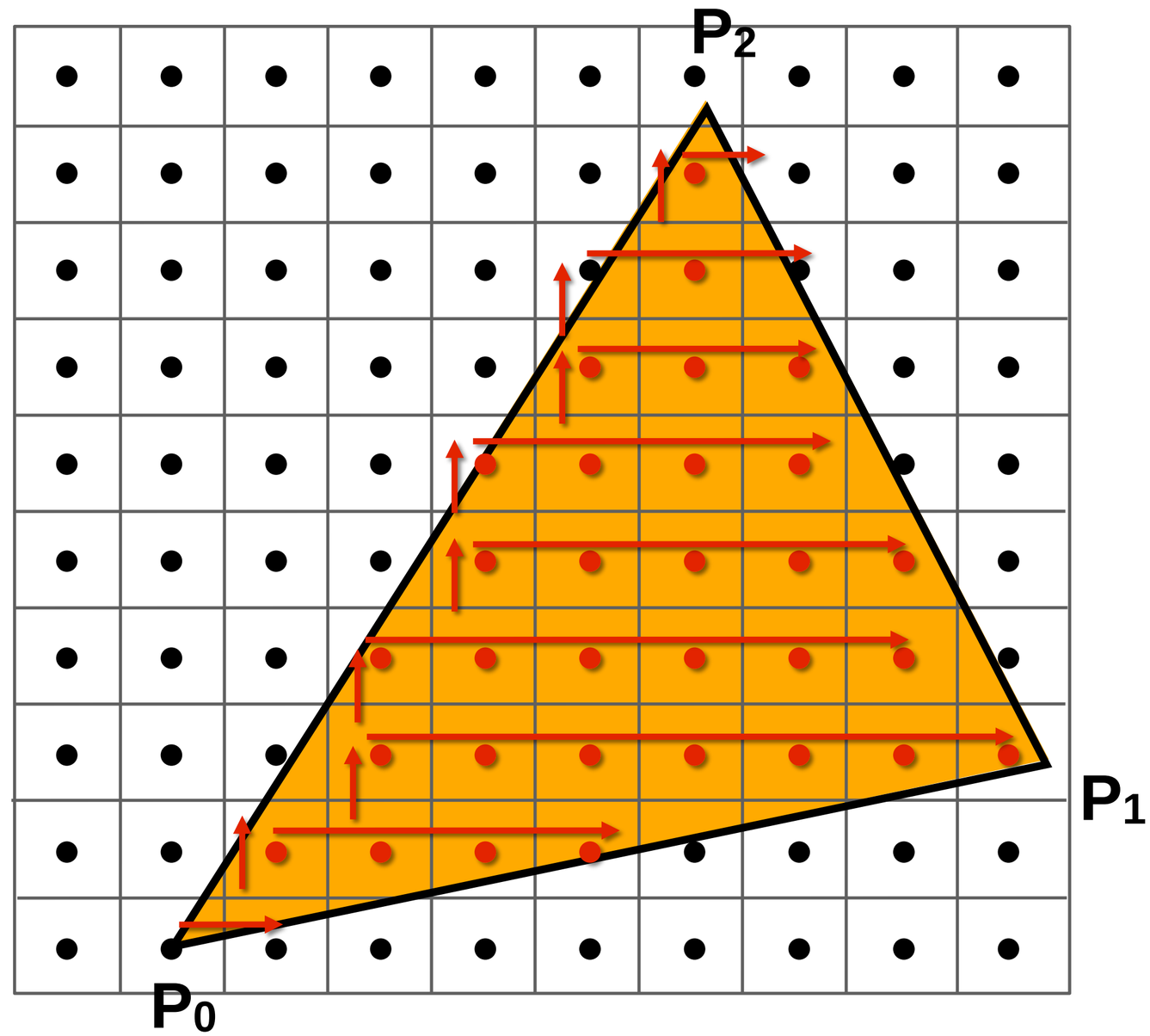
- **All of them?**
- **Points within bounding box?**



# 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)



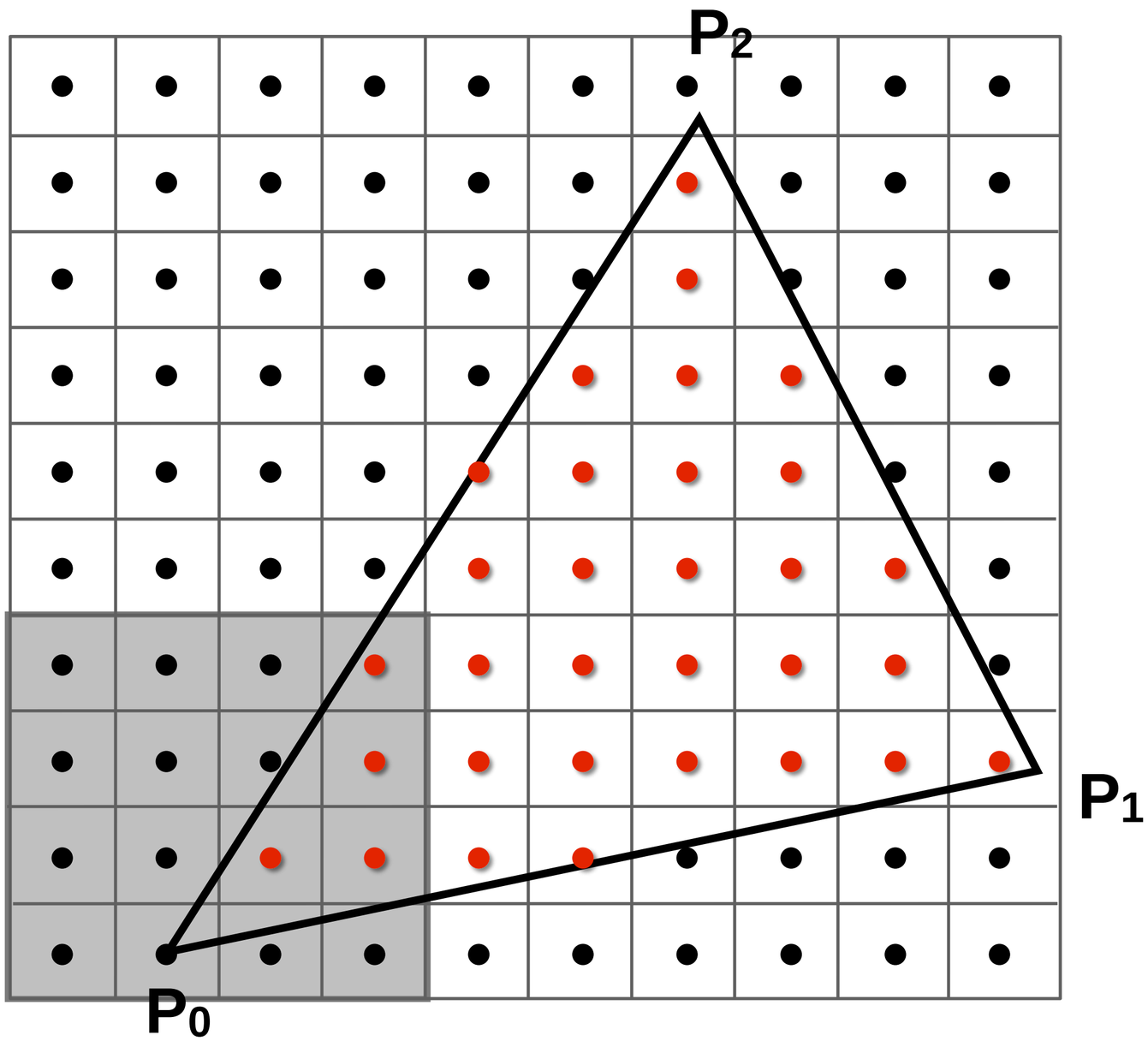
# Modern approach: tiled triangle traversal

**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

# Today’s quiz question

- **Will be posted on the web later today**