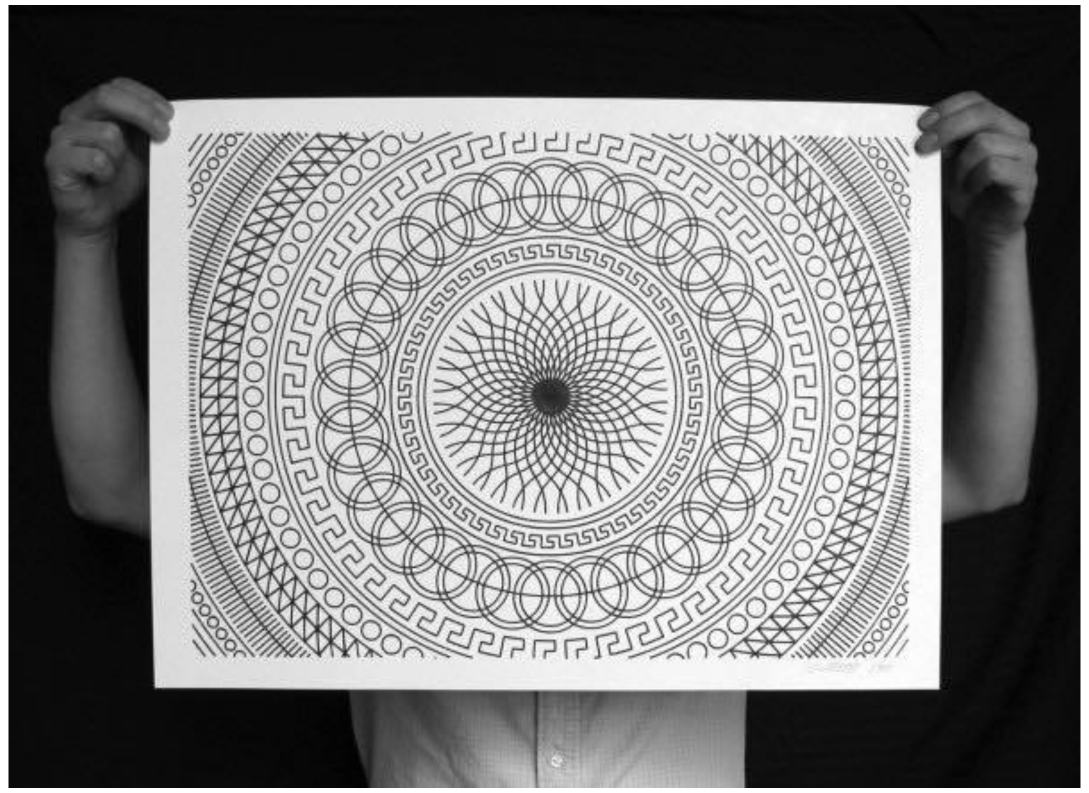
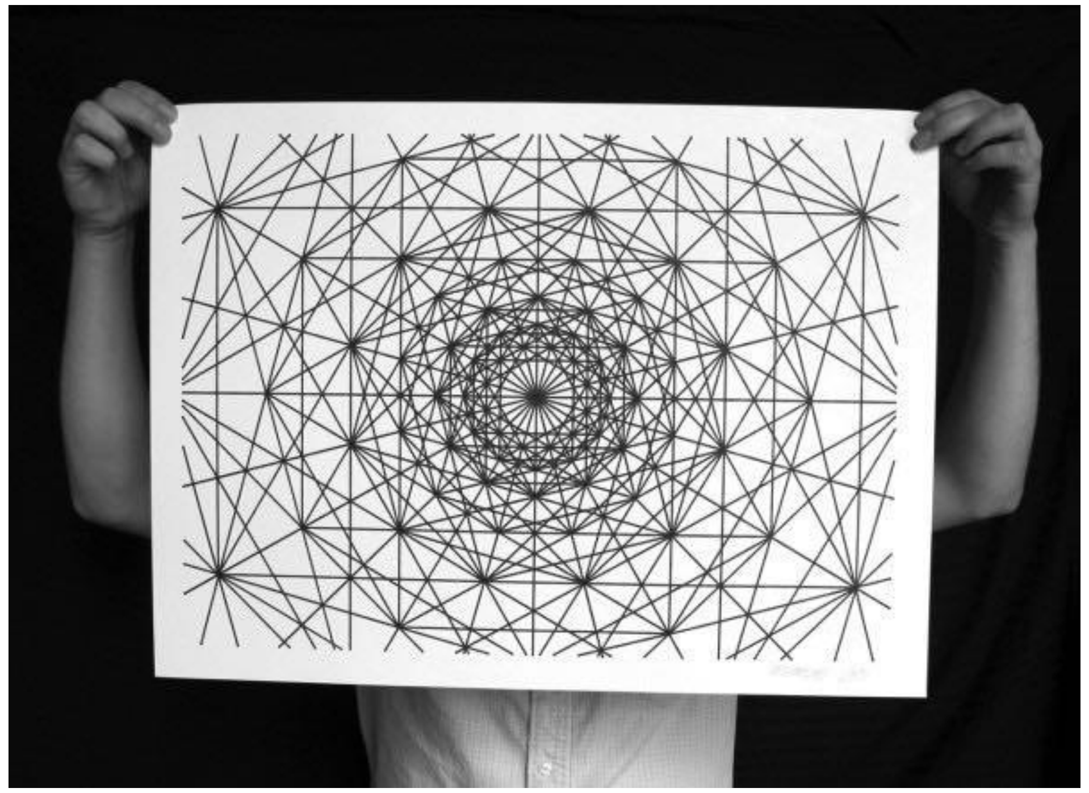
Week 8,9

Draw Lines and Triangles :

How do we draw lines on a computer?

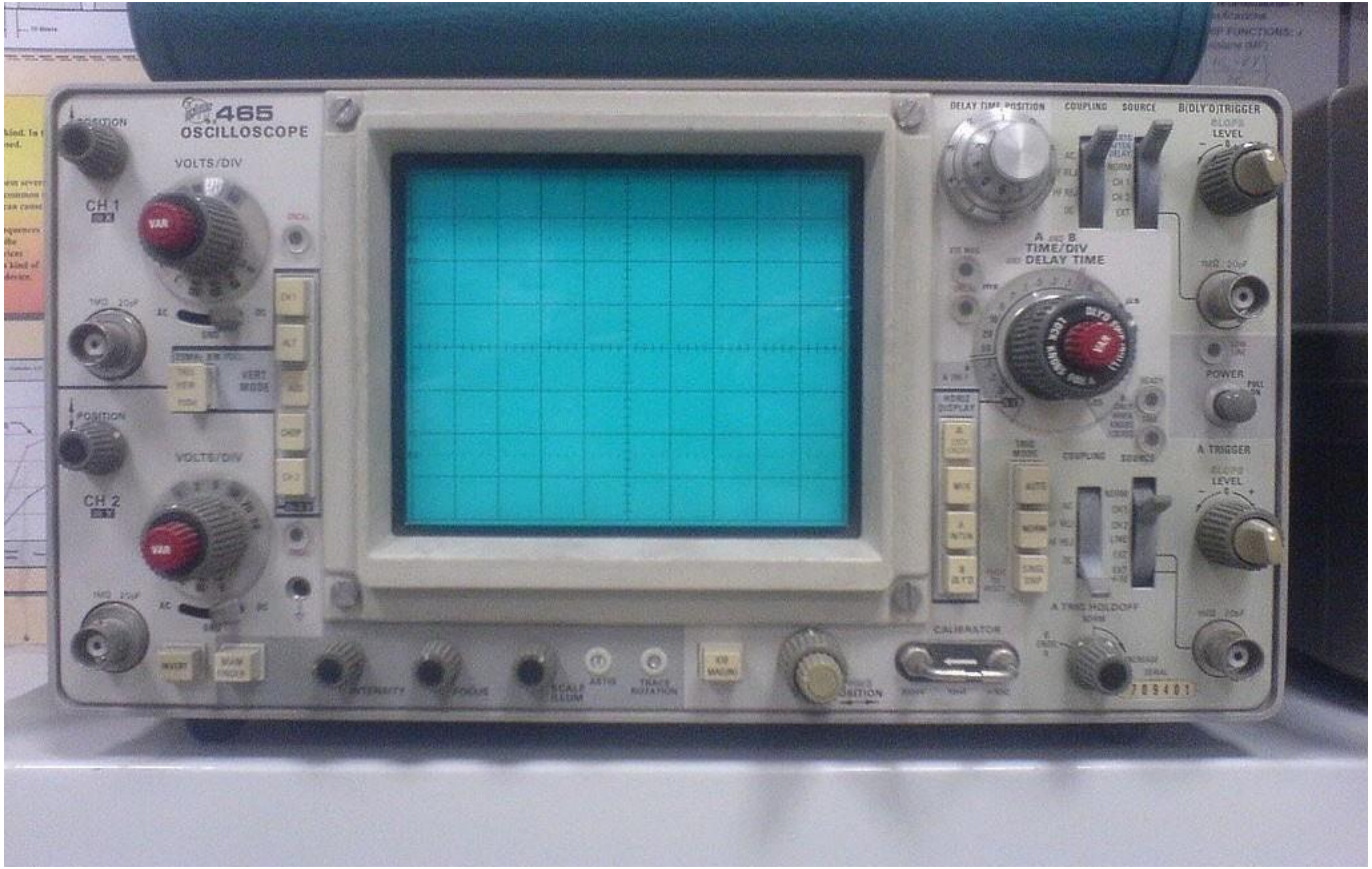
CNC sharpie drawing machine ;-)



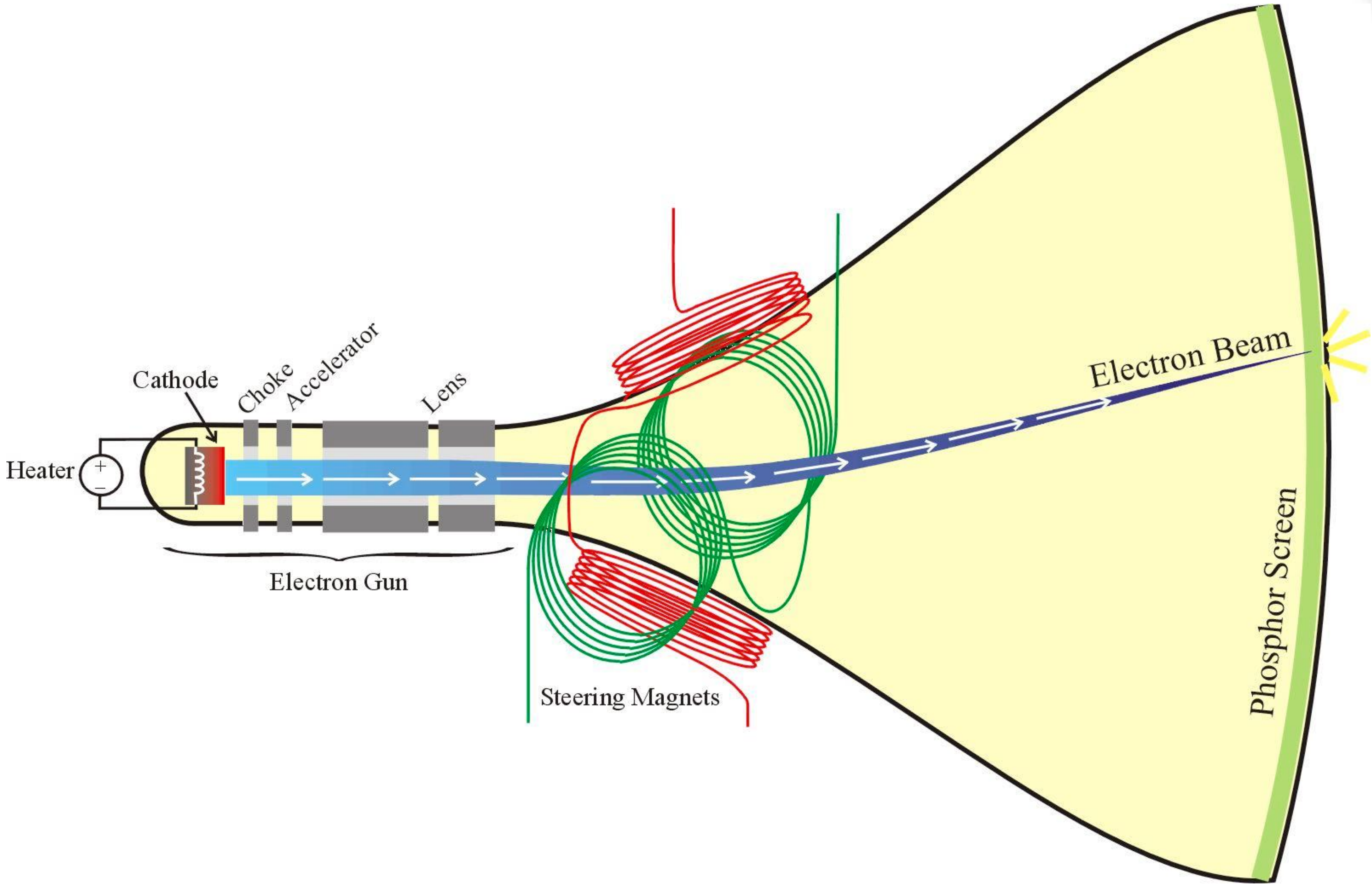


http://44rn.com/projects/numerically-controlled-poster-series-with-matt-w-moore/

Oscilloscope



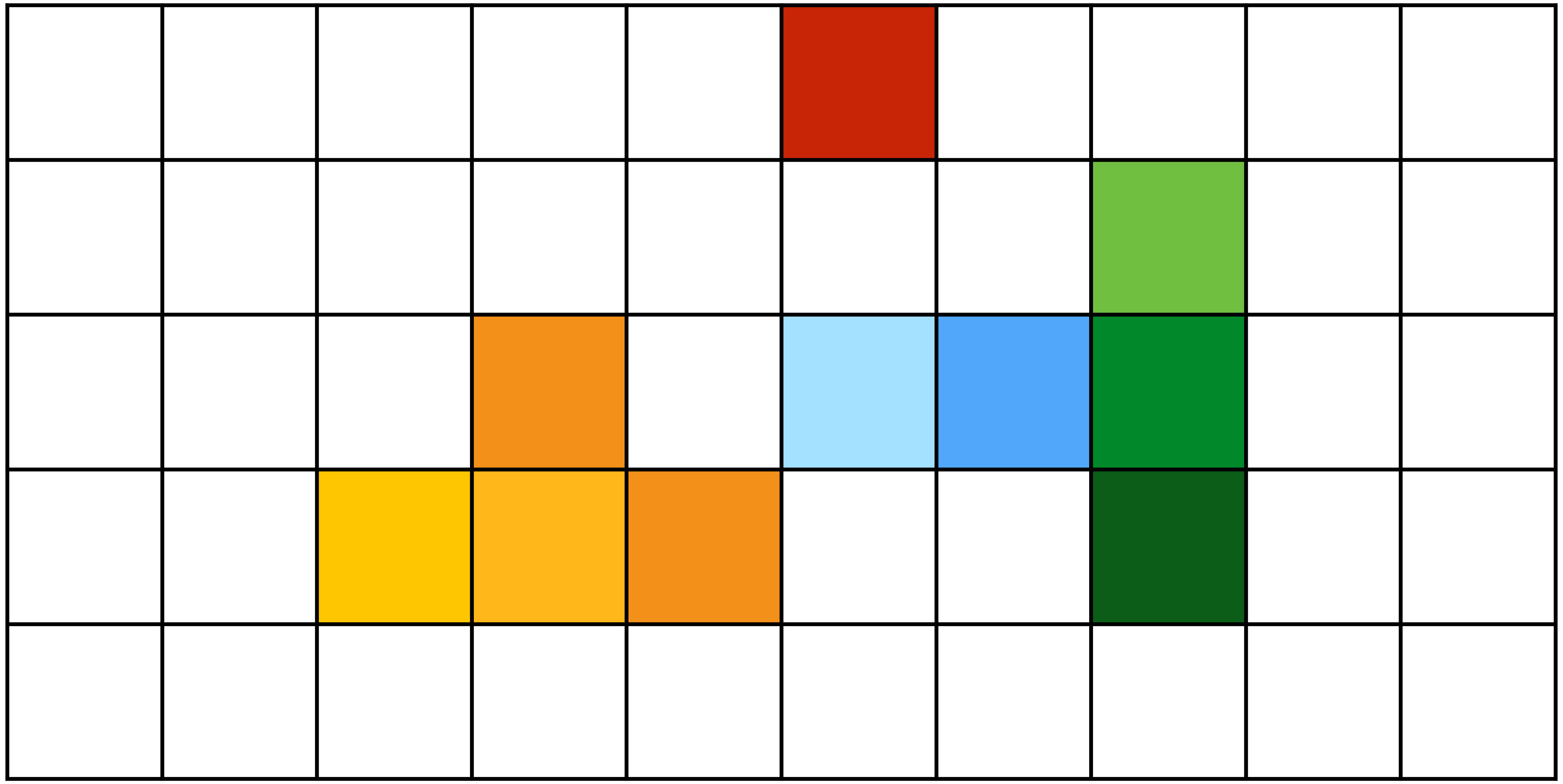
Cathode ray tube

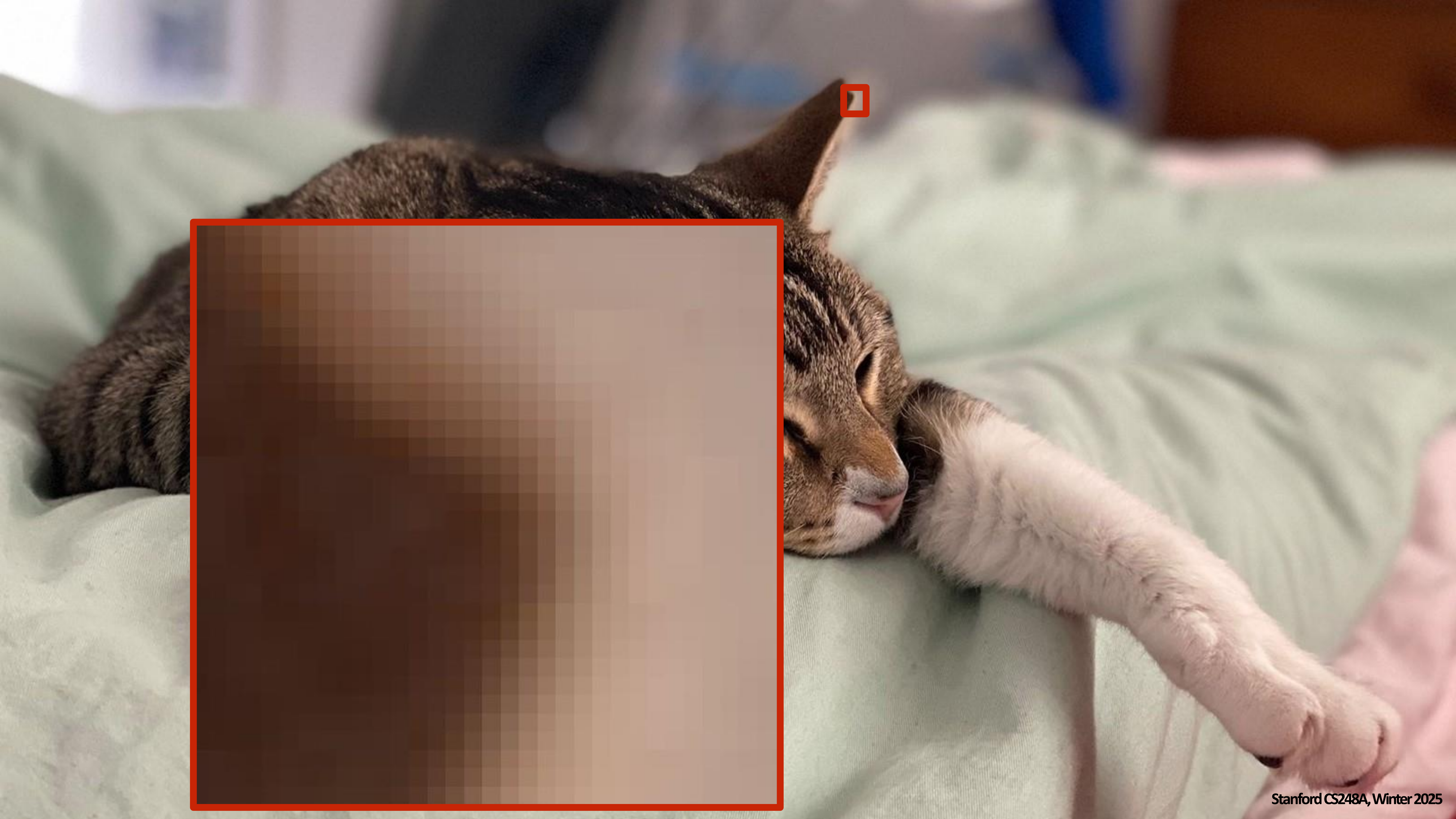


[Credit: http://propagation.ece.gatech.edu/ECE3025/tutorials/CathodeRayTube/CRToverview.htm]

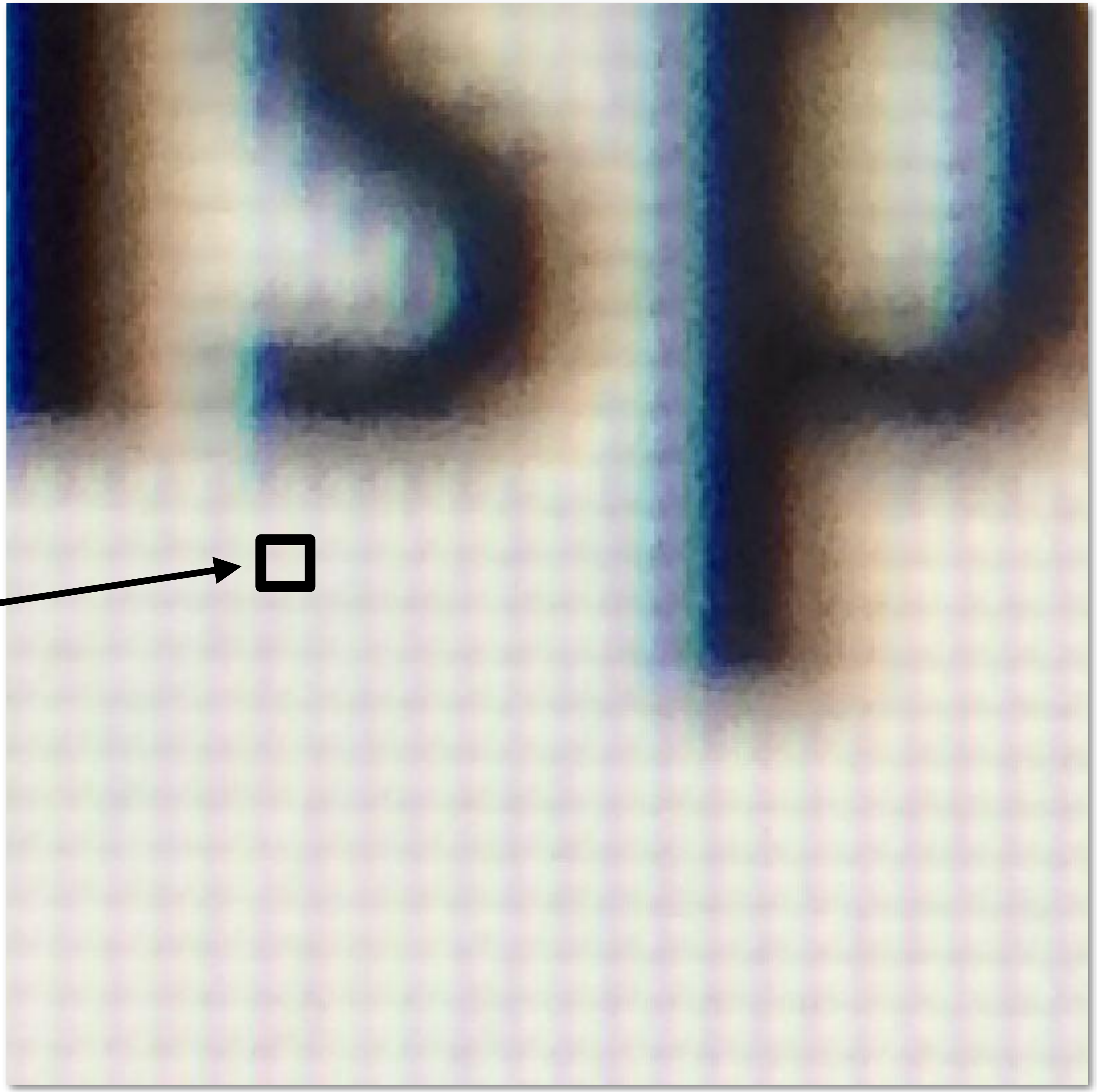
Output for a raster display

- Common abstraction of a raster display:
  - Image represented as a 2D grid of “pixels”
  - Each pixel can can take on a unique color value



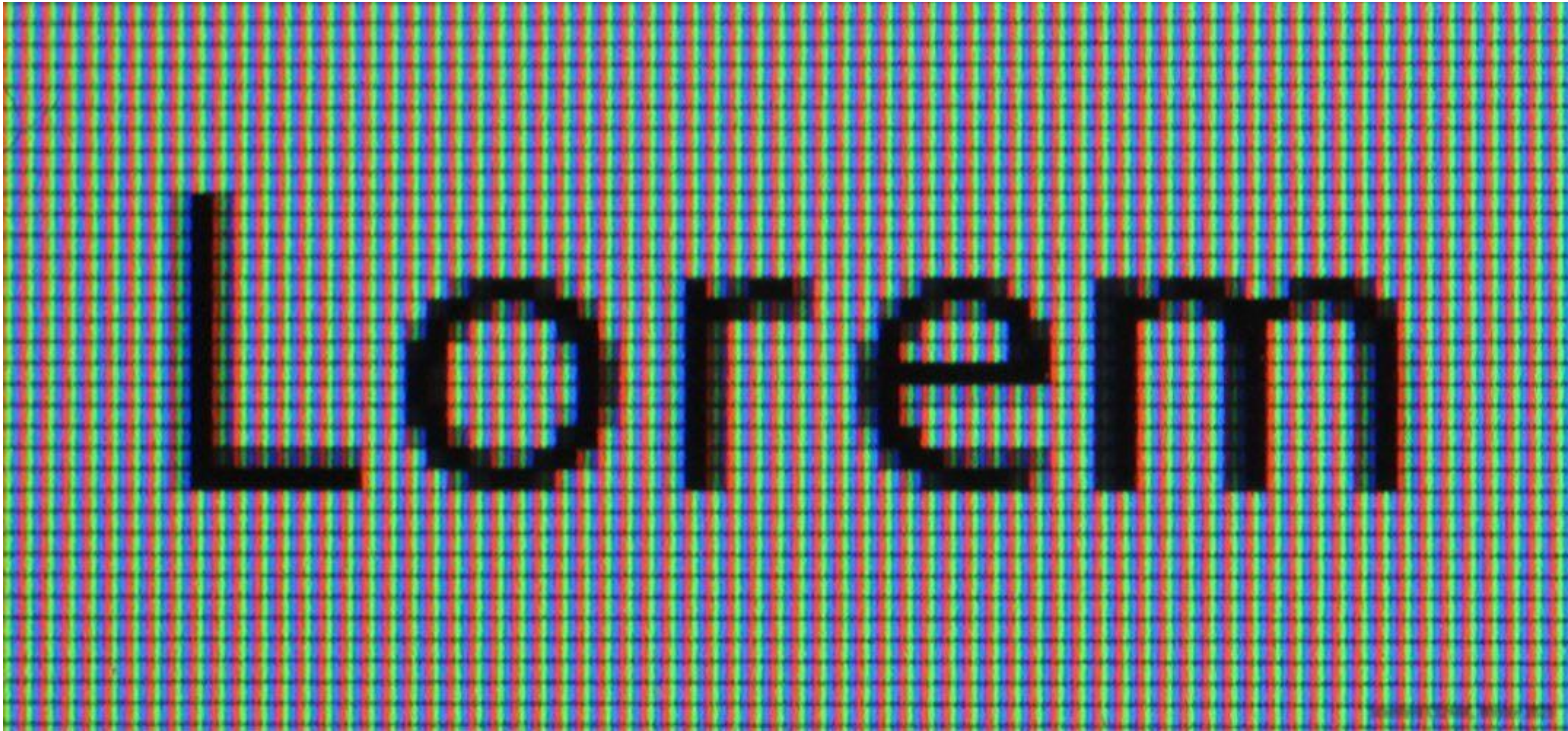


A raster display converts an image (a color value at each pixel) into emitted light

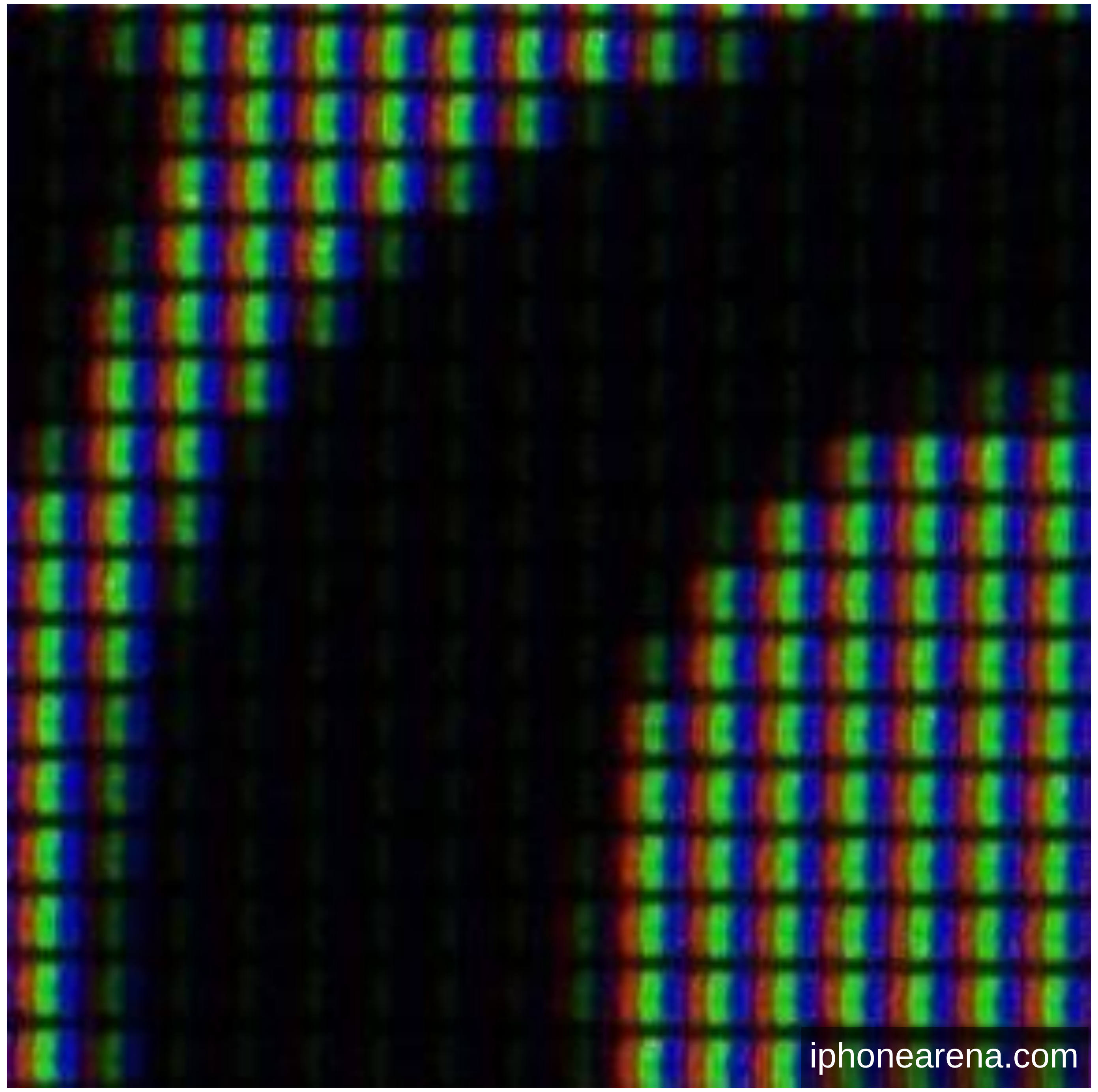


Display pixel on my laptop
(close up photo)

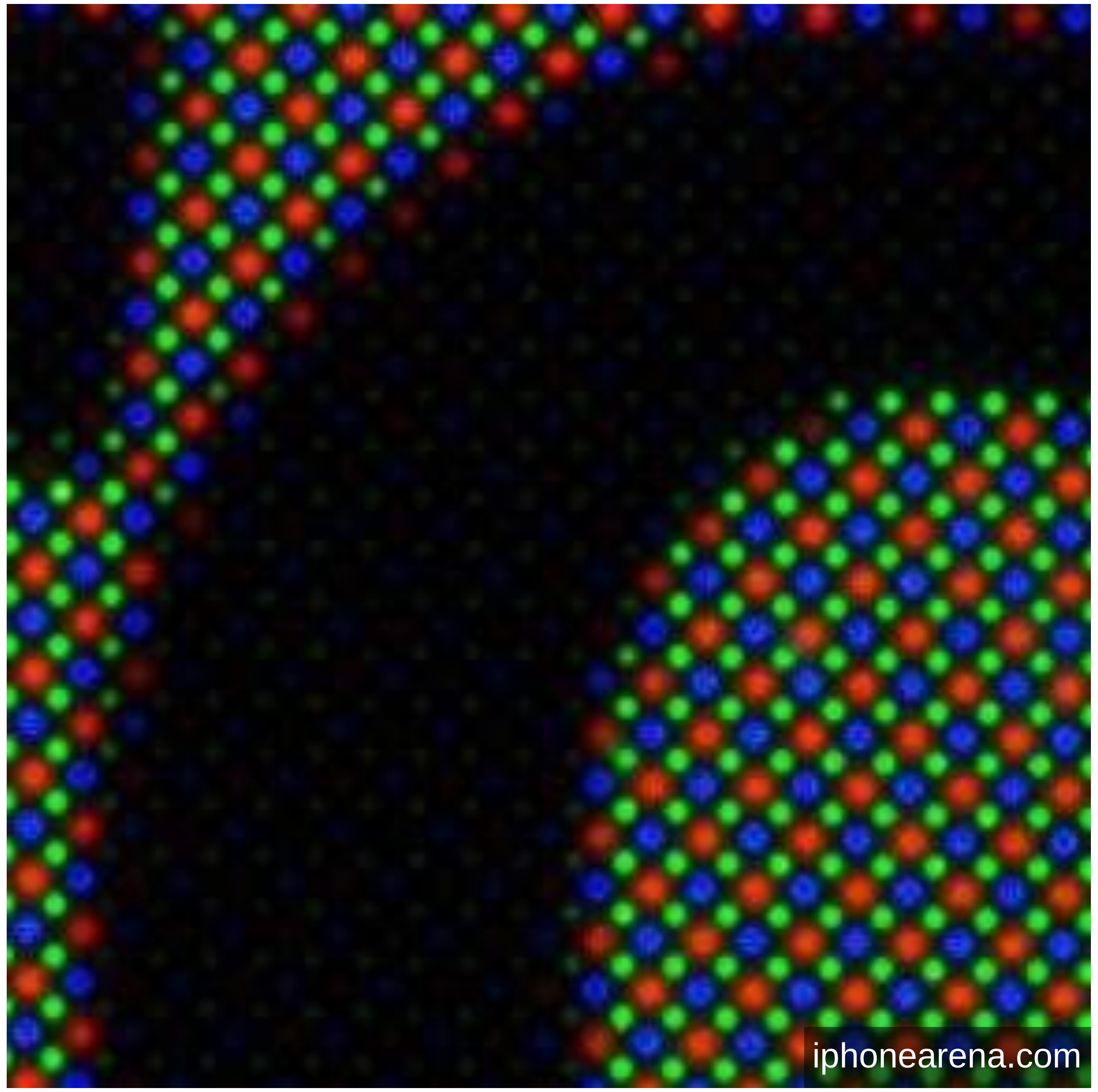
Close up photo of pixels on a modern display



LCD screen pixels (closeup)



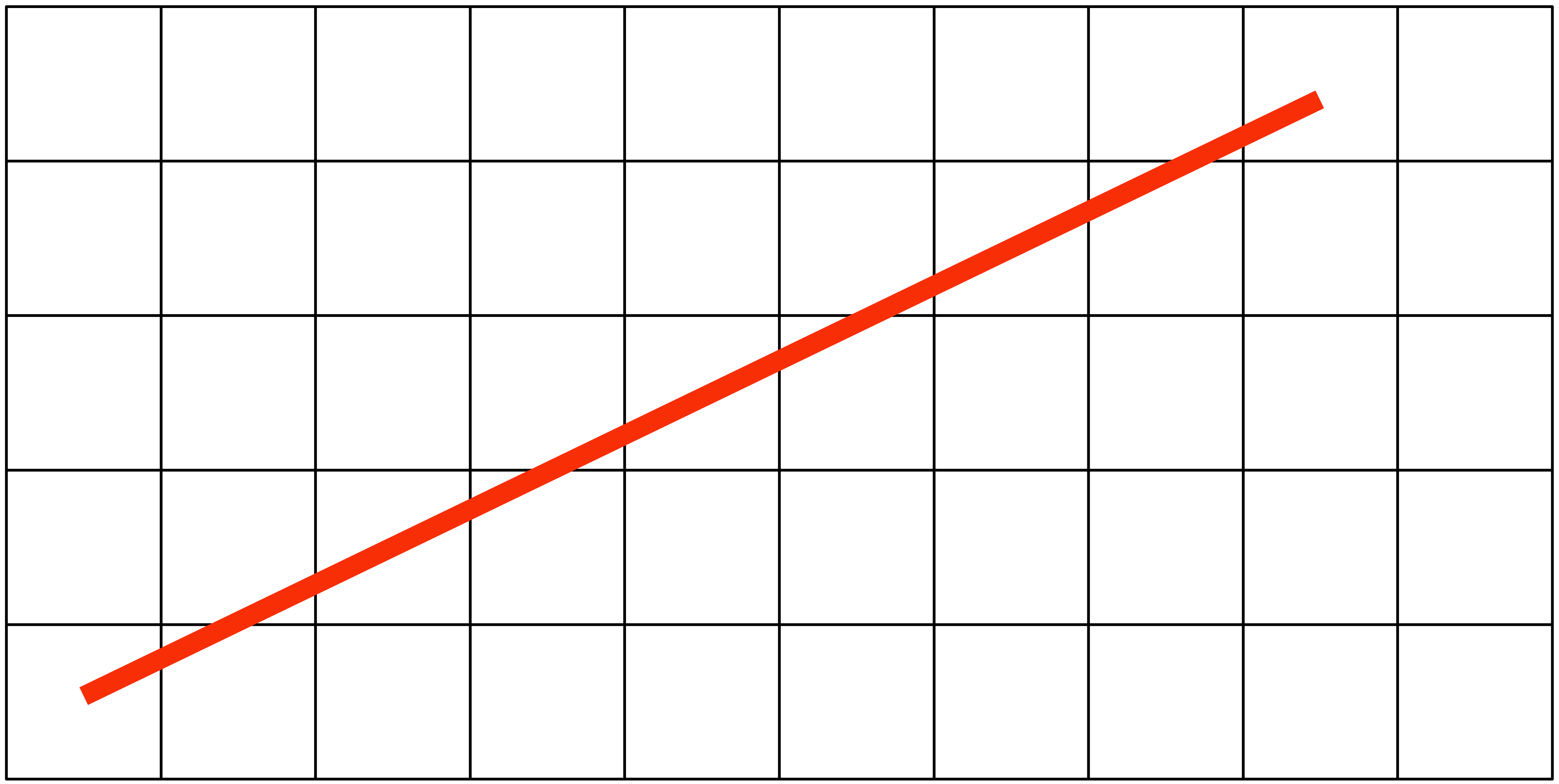
iPhone 6S



Galaxy S5

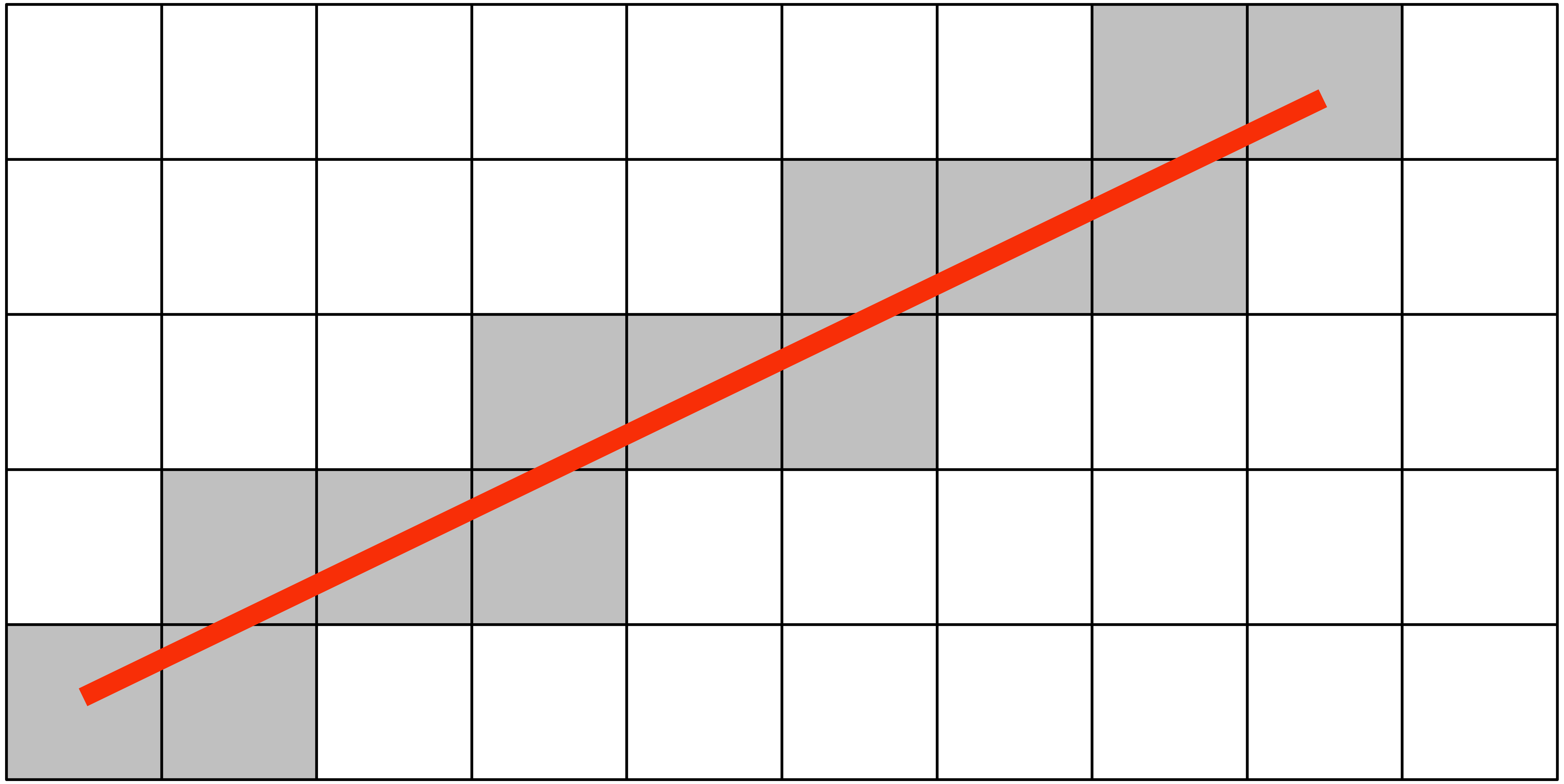
What pixels should we color in to depict a line?

"Rasterization": process of converting a continuous object (a line, a polygon, etc.) to a discrete representation on a "raster" grid (pixel grid)



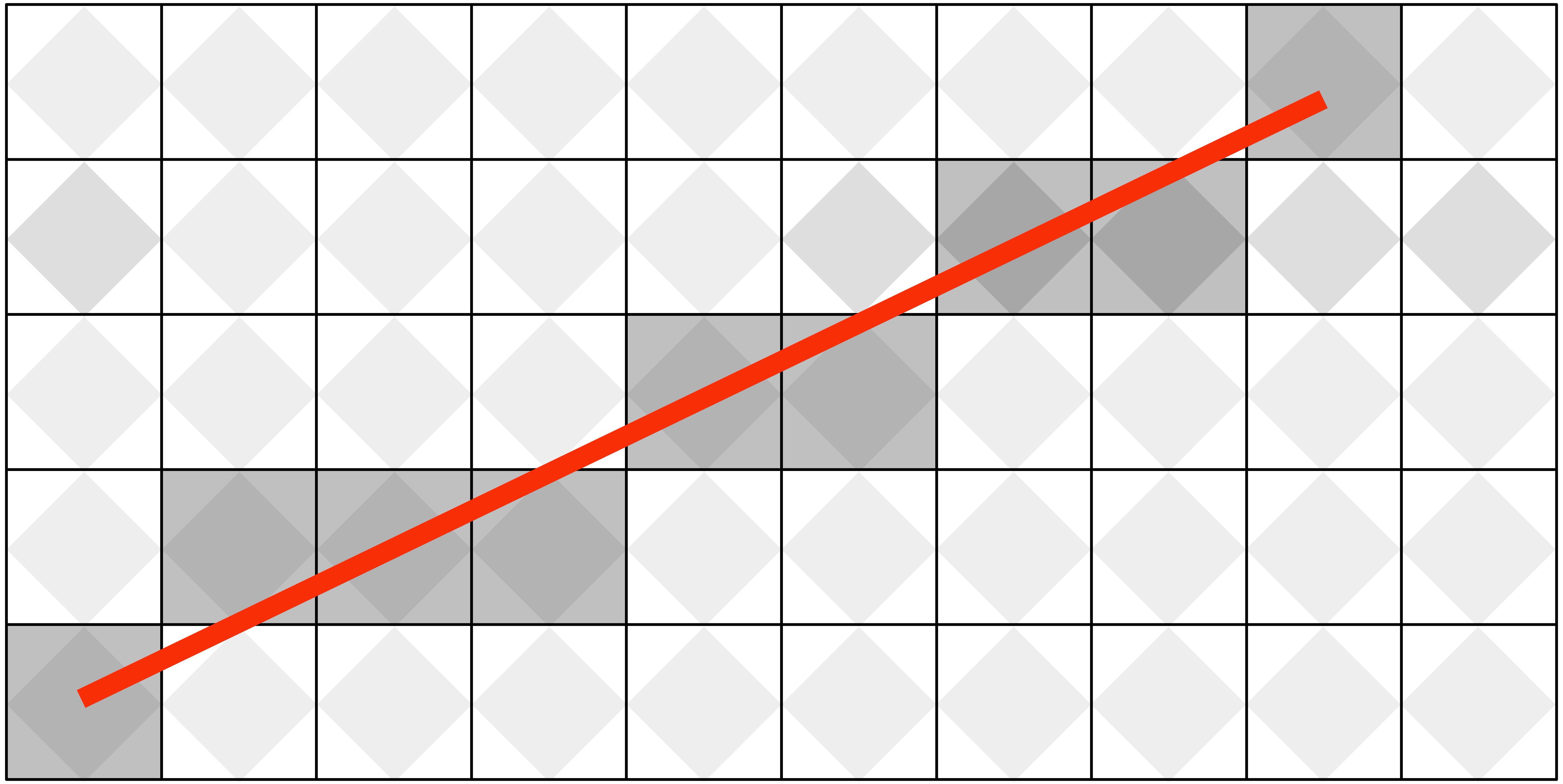
What pixels should we color in to depict a line?

Light up all pixels intersected by the line?



What pixels should we color in to depict a line?

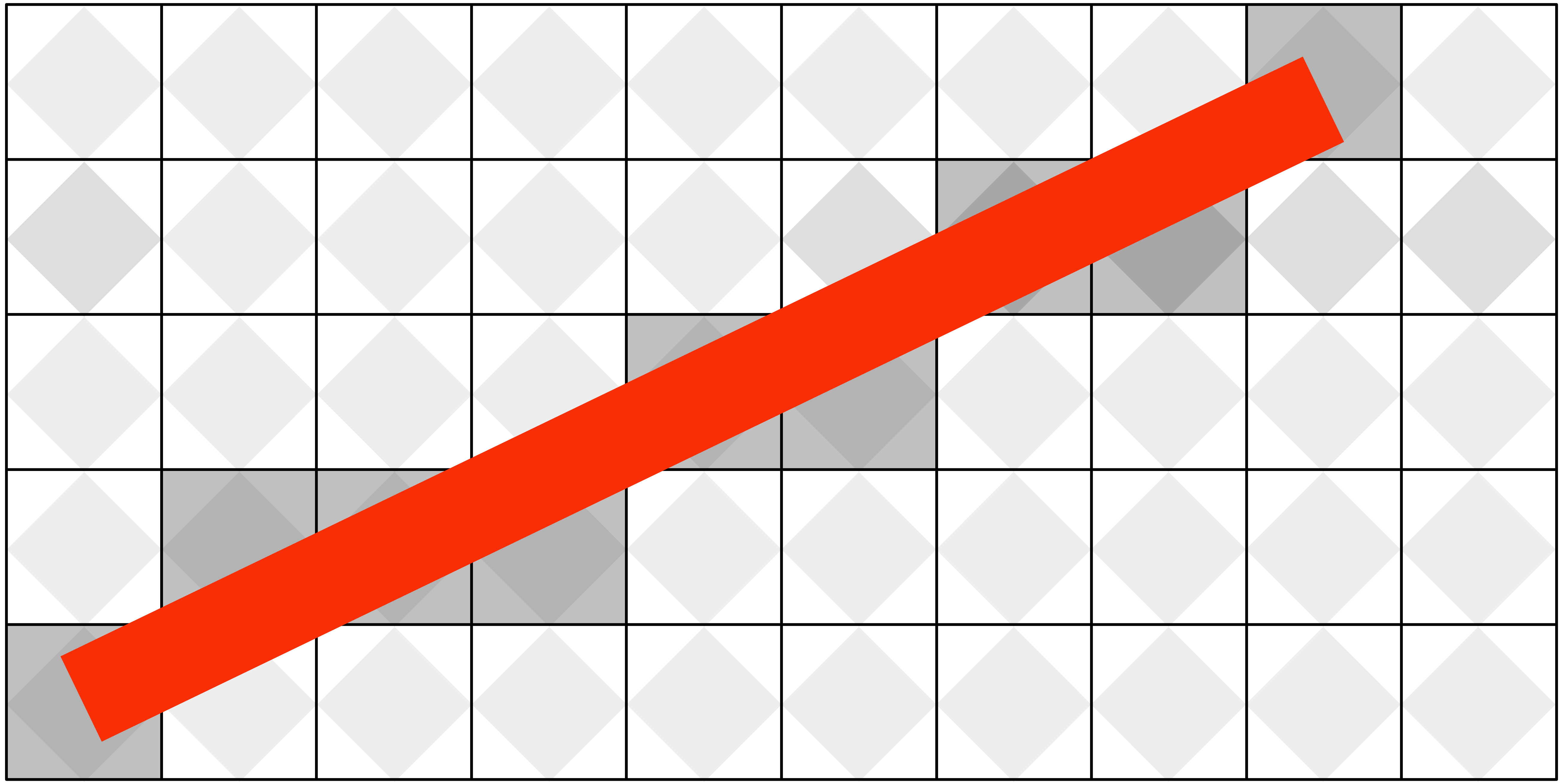
Diamond rule (used by modern GPUs):
light up pixel if line passes through associated diamond



What pixels should we color in to depict a line?

Is there a right answer?

(consider a drawing a “line” with thickness)



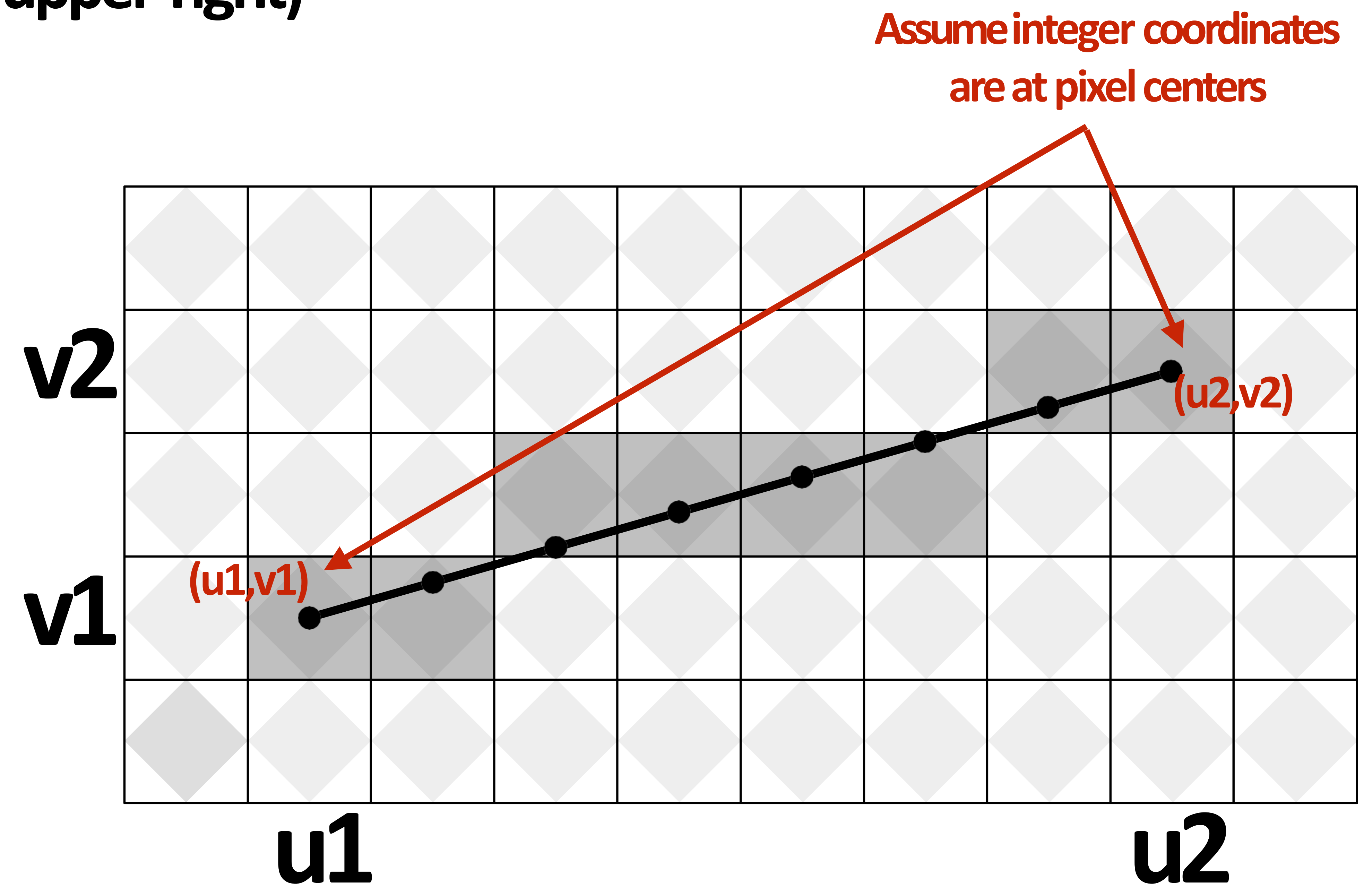
How do we find the pixels satisfying a chosen rasterization rule?

- Could check every single pixel in the image to see if it meets the condition…
  - $O(n^2)$ pixels in image vs. at most $O(n)$ “lit up” pixels
  - *Must* be able to do better!

Incremental line rasterization

- Let's say a line is represented with integer endpoints: (u1,v1), (u2,v2)
- Slope of line: s = (v2-v1) /(u2-u1)
- Consider an easy special case:
  - u1 < u2, v1 < v2 (line points toward upper-right)
  - 0 < s < 1 (more change in x than y)

```
v = v1;
for( u=u1; u<=u2; u++ )
{
   v += s;
   draw( u, round(v) )
}
```

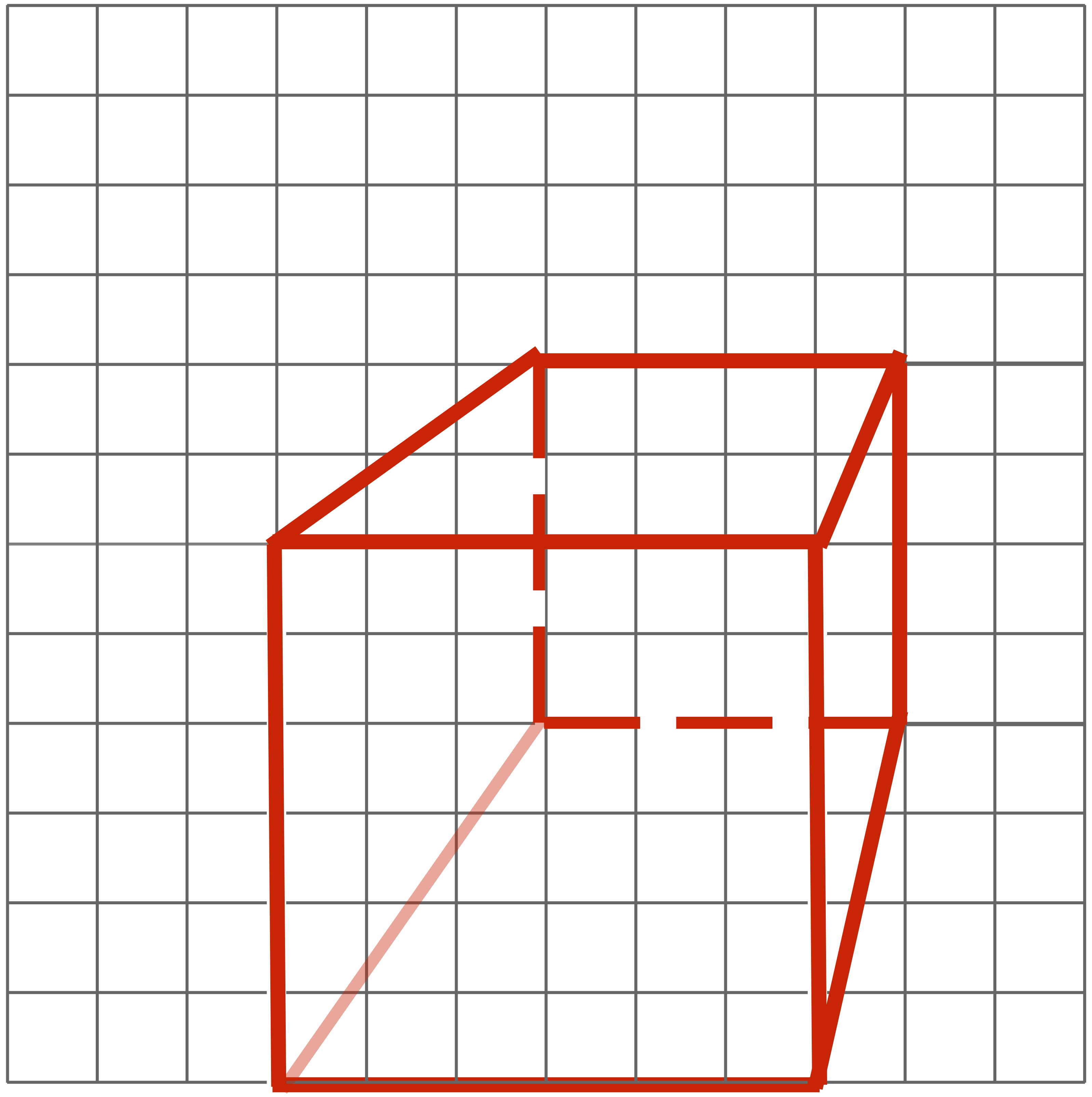


Common optimization: rewrite algorithm to use only integer arithmetic (Bresenham algorithm)

Line drawing of cube

We know how to compute to location of points in 3D on a 2D screen

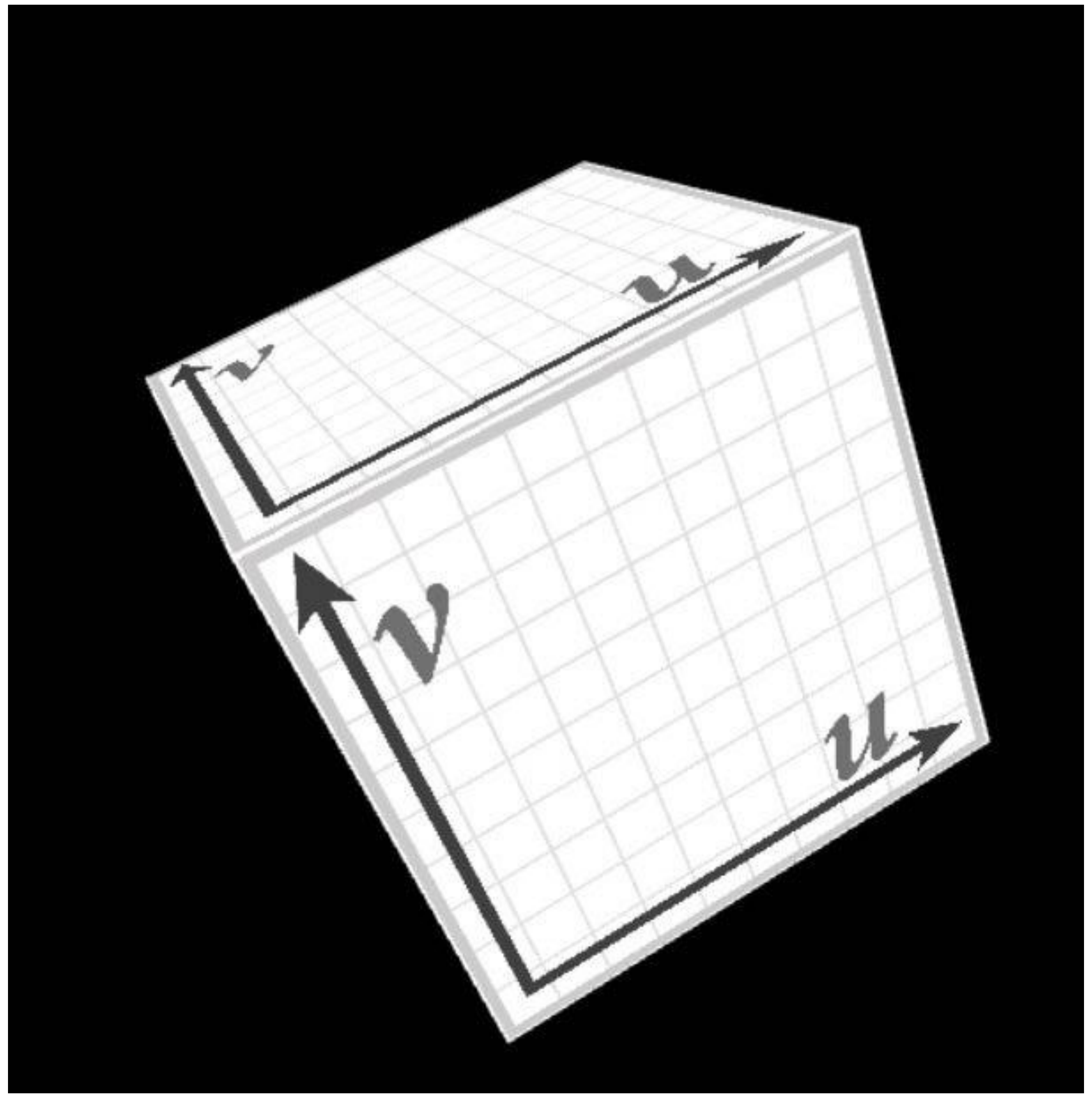
We know how to draw lines between those points.



We just rendered a simple line drawing of a cube.

But to render more realistic pictures
(or animations) we need a much richer model of the world.

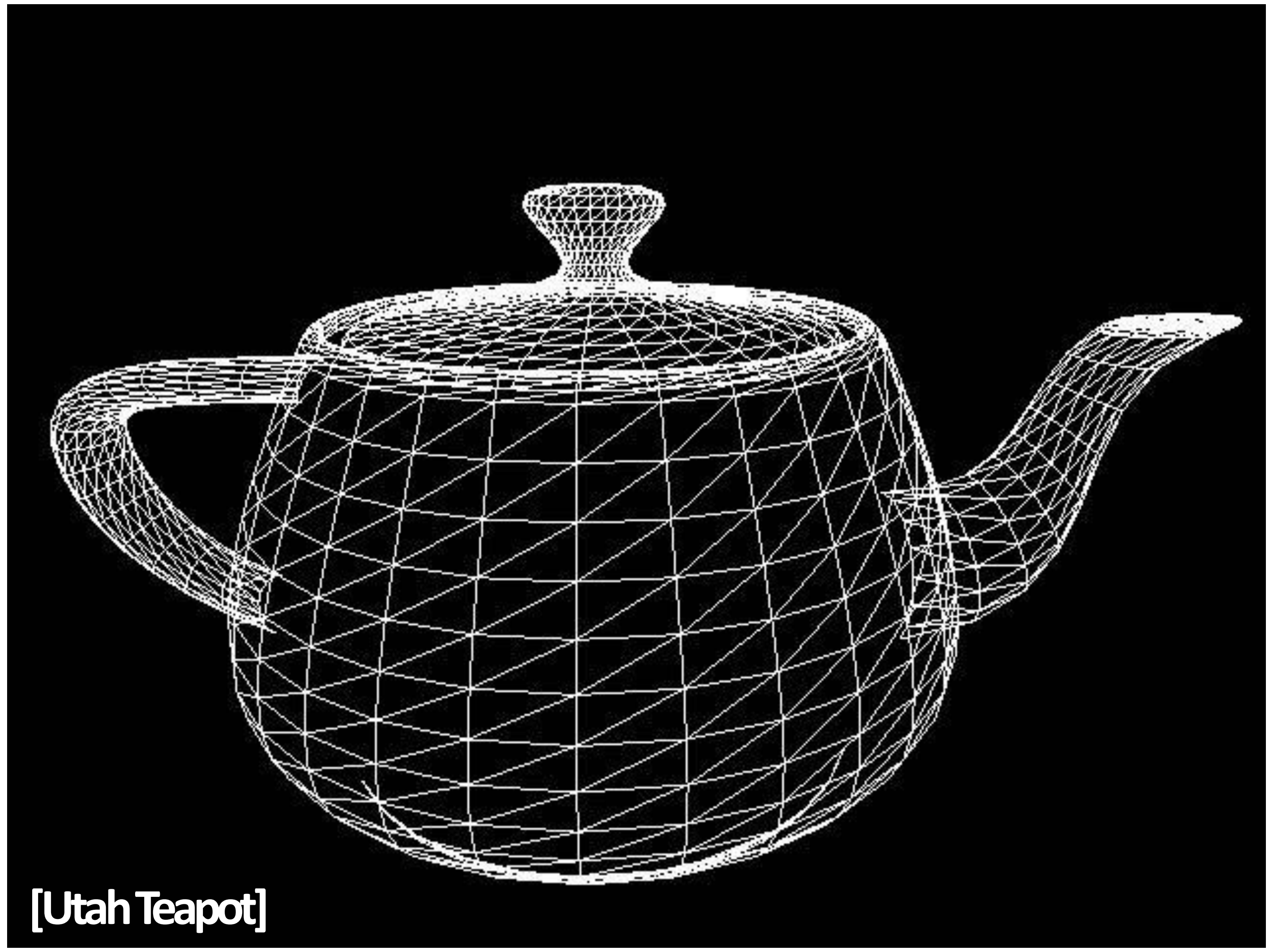
surfaces
materials
lights
cameras

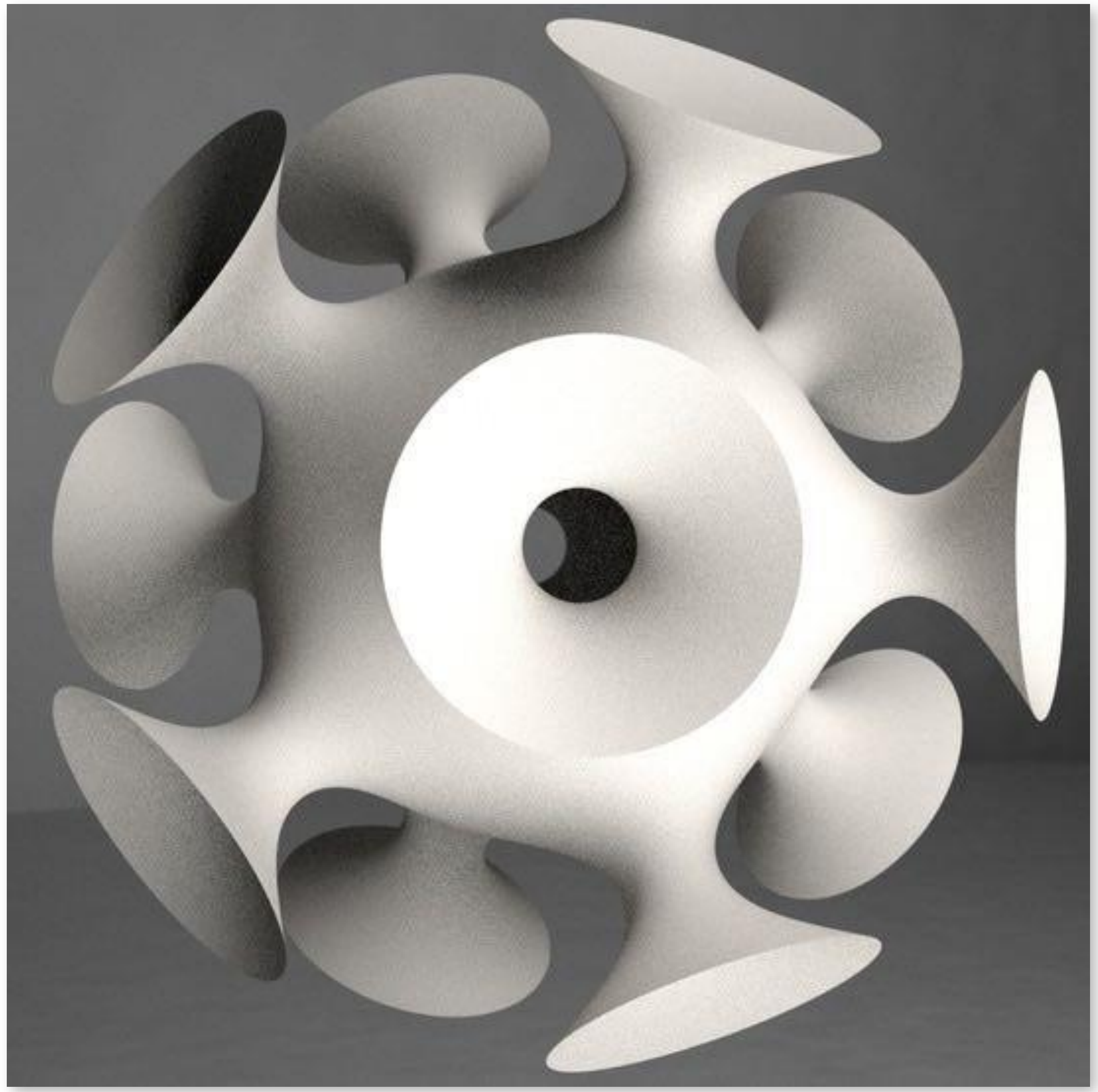


2D shapes

[Source: Batra 2015]

Complex 3D surfaces





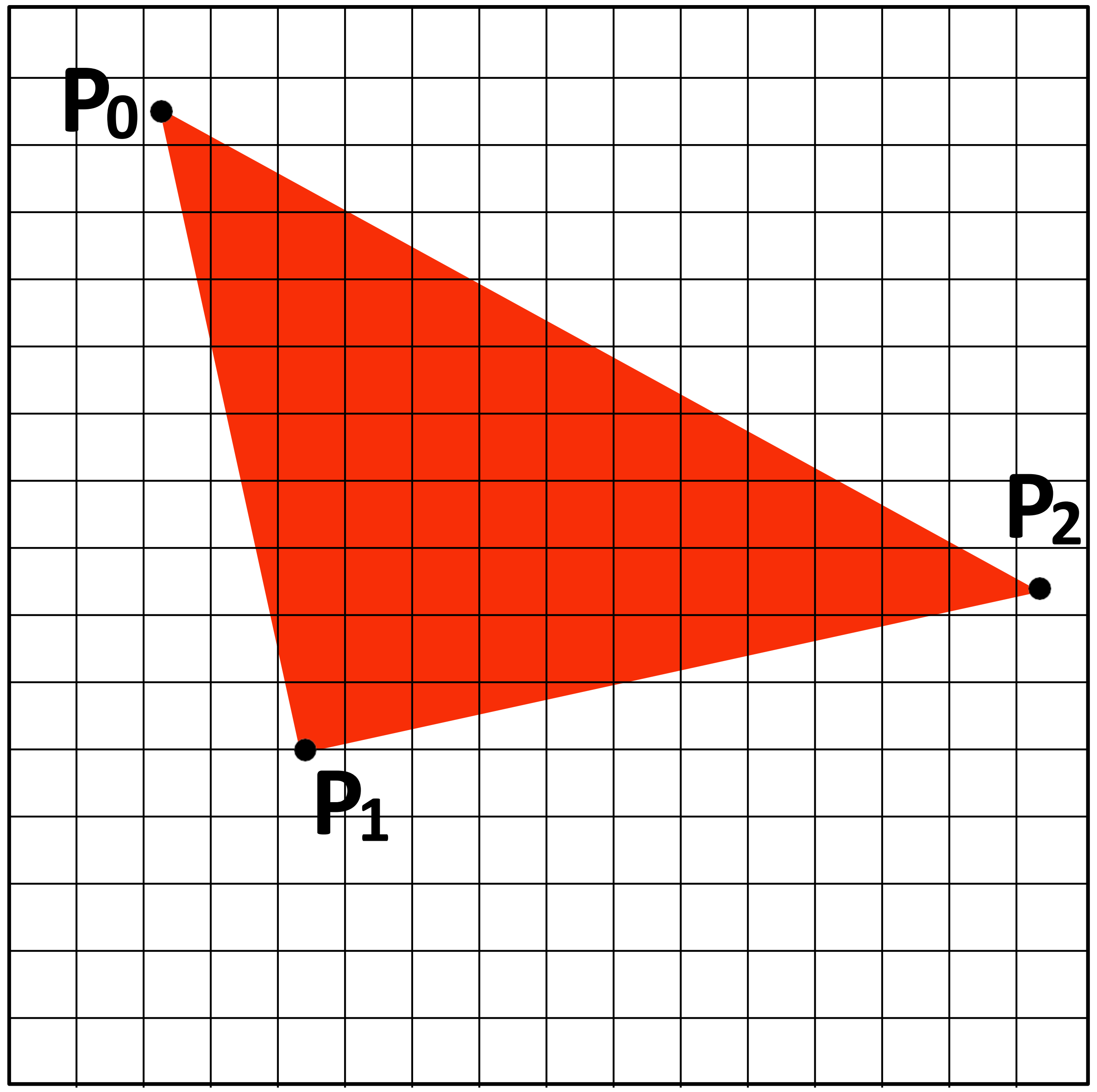
Platonic noid

We talked about drawing lines, what about triangles?

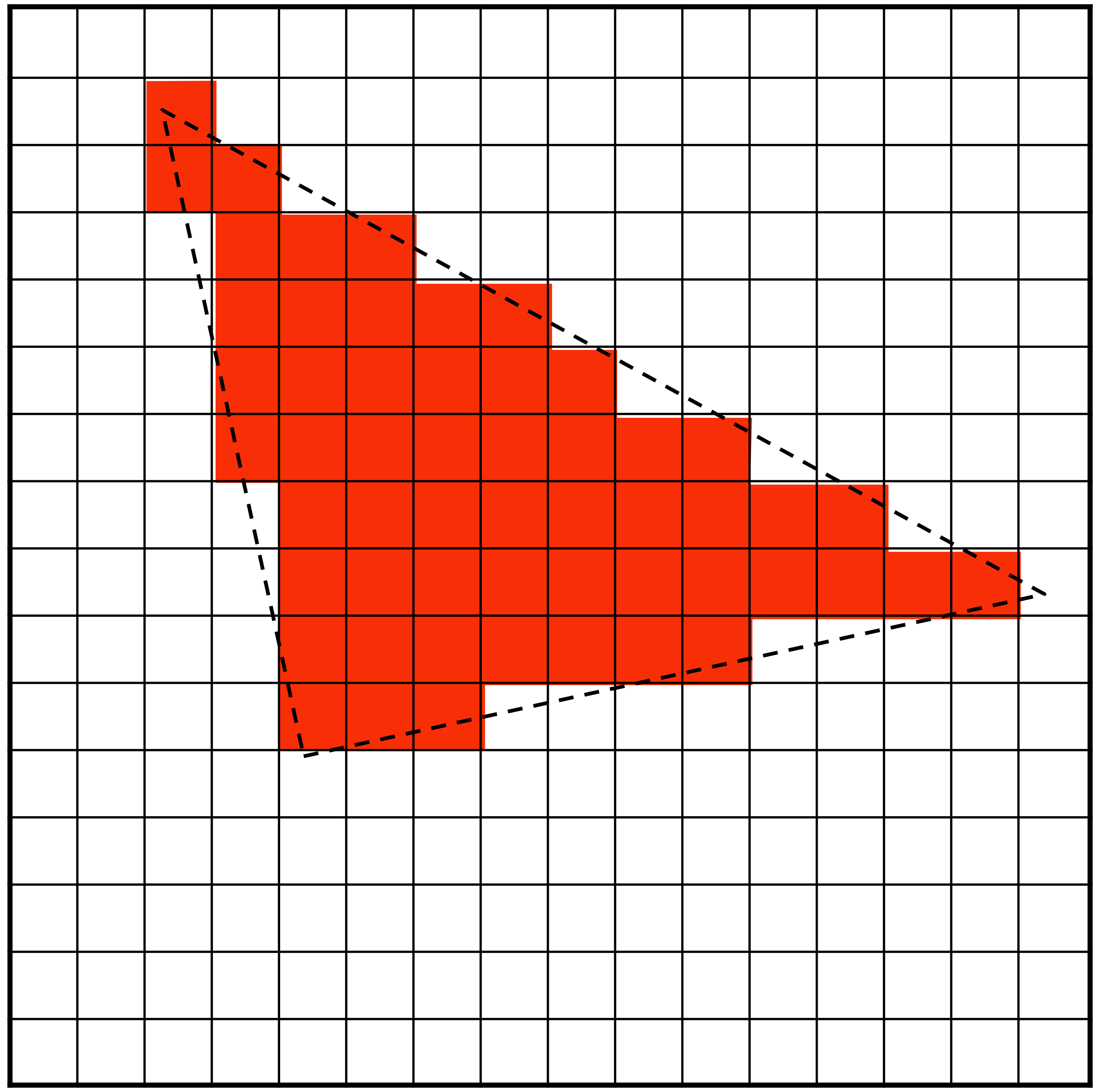
Drawing a triangle ("triangle rasterization")

(Converting a representation of a triangle into an image)

Input:
2D position of triangle vertices: P_0, P_1, P_2

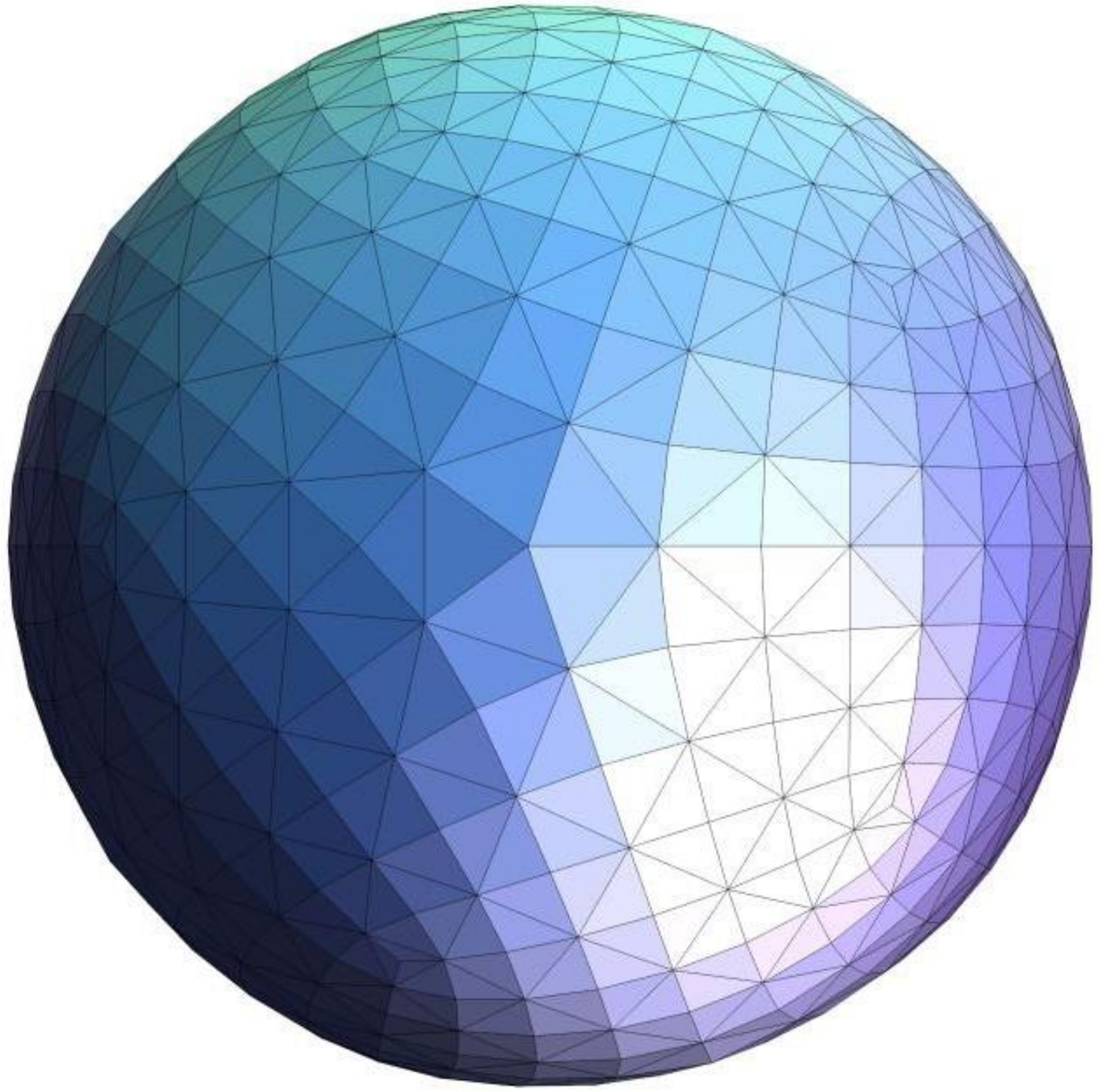


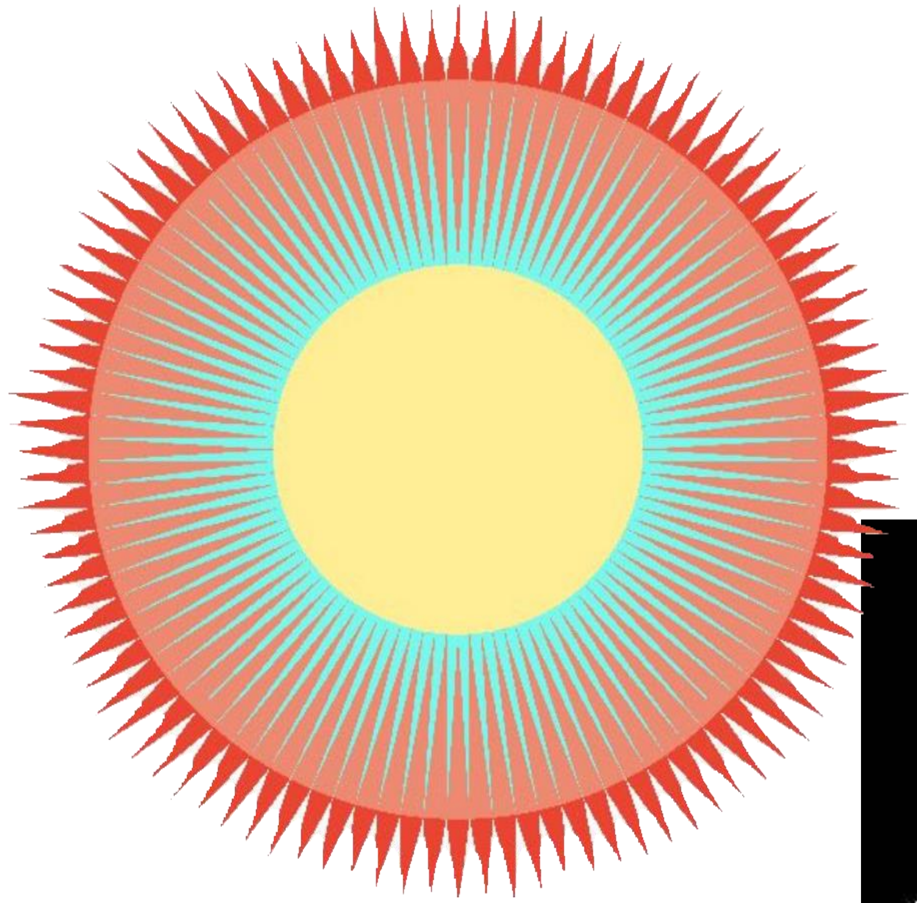
Output:
Set of pixels "covered" by the triangle

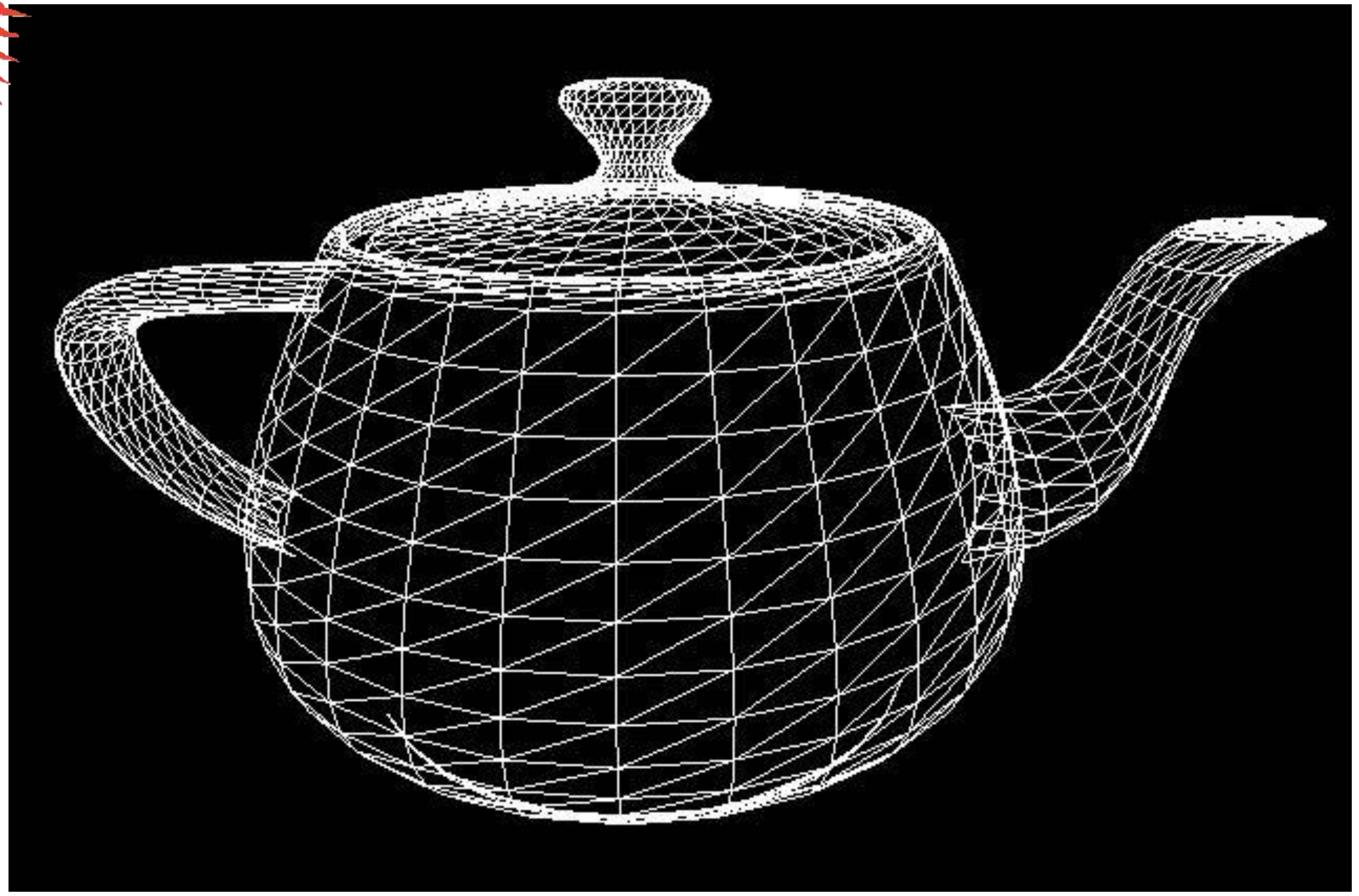


Why triangles?

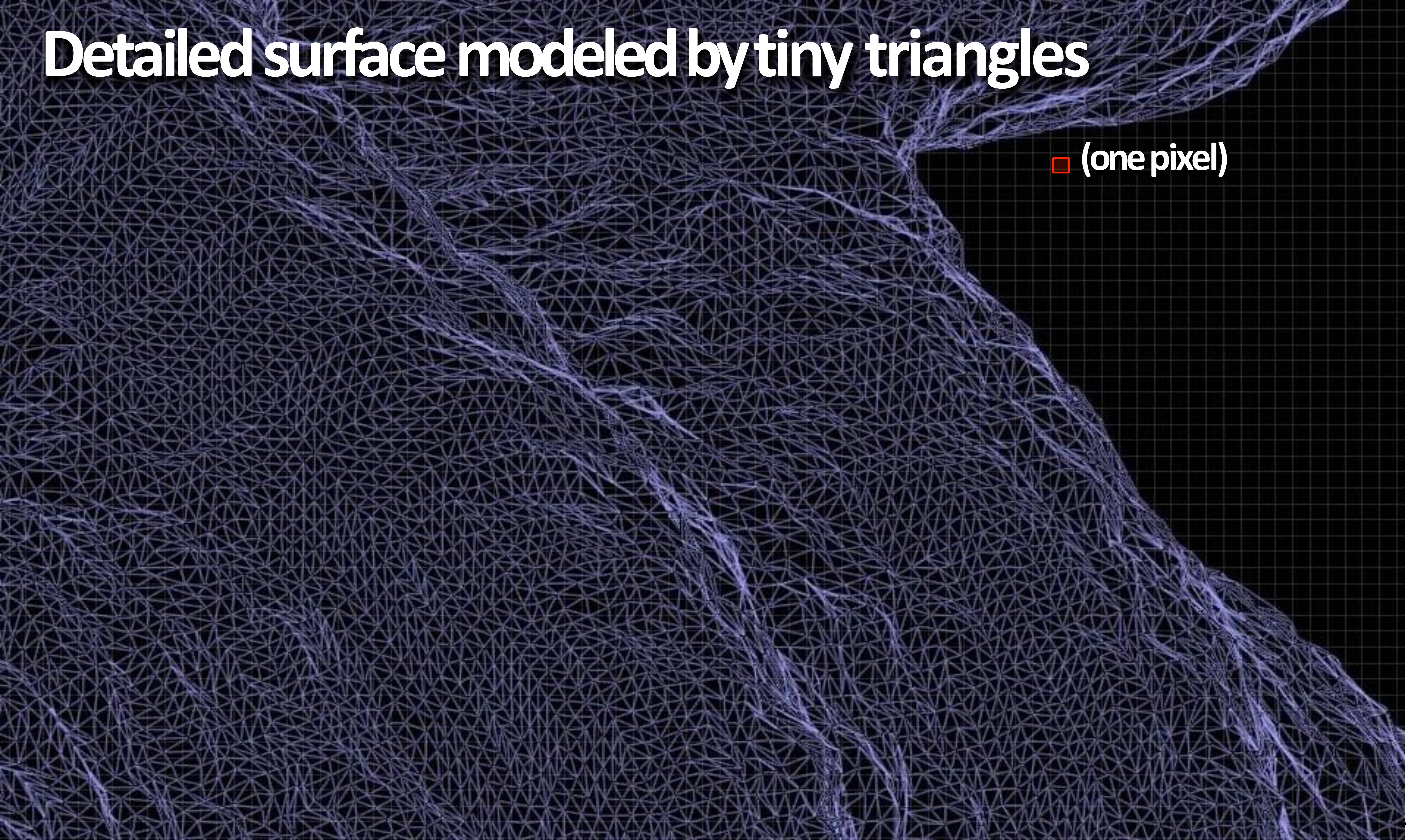
Triangles are a basic block for creating more complex shapes and surfaces



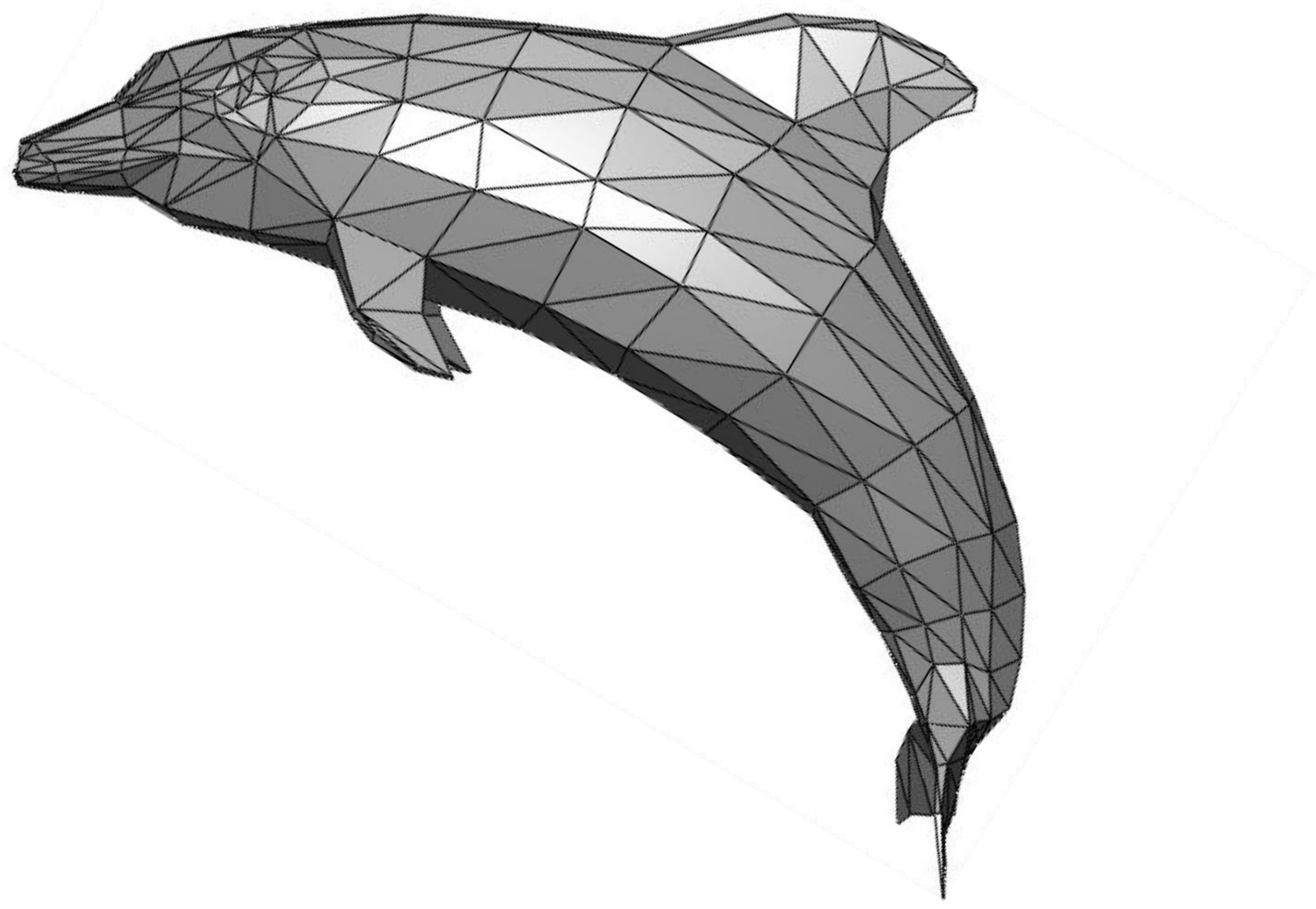




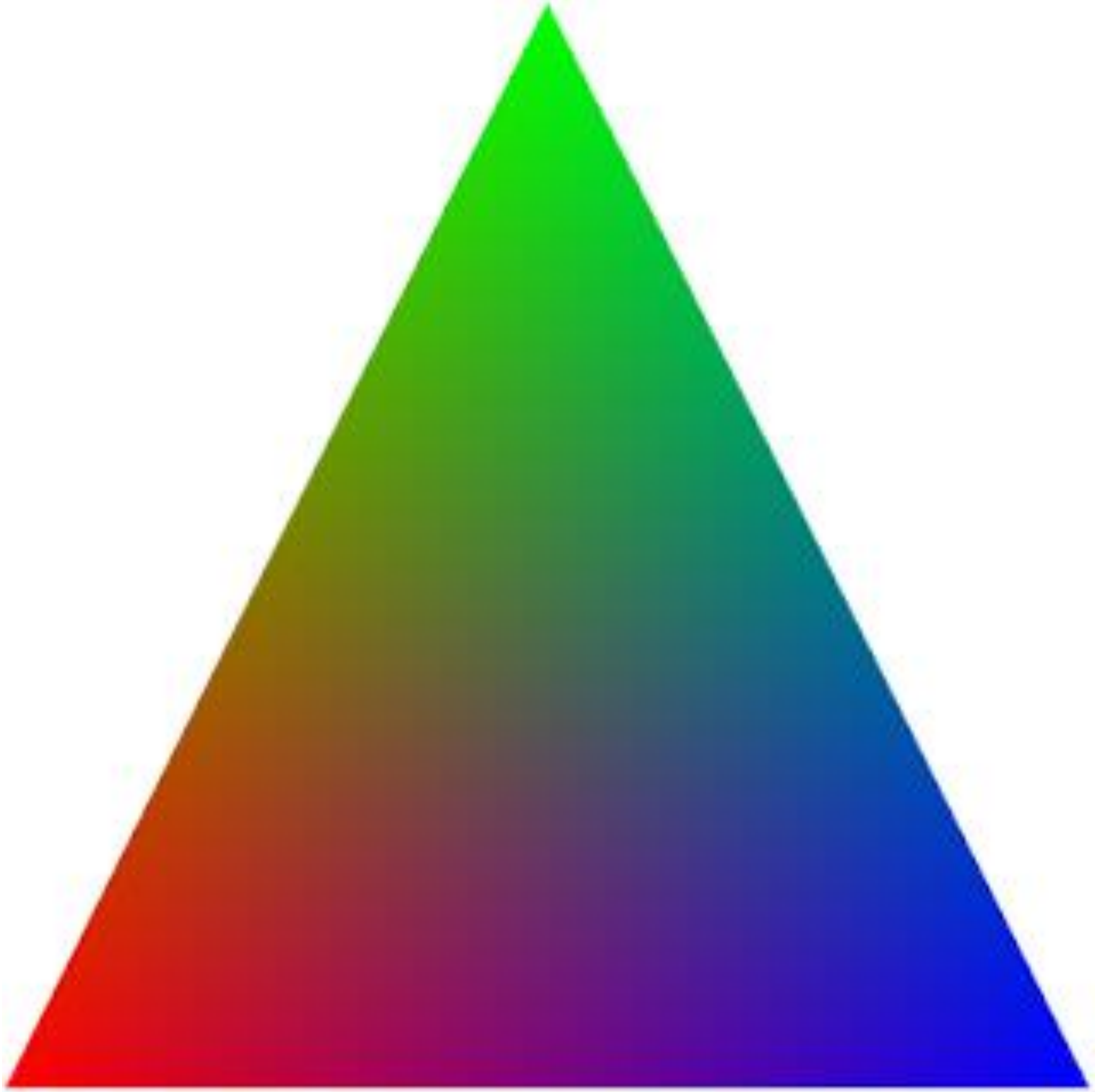
Detailed surface modeled by tiny triangles



Triangles - a fundamental primitive

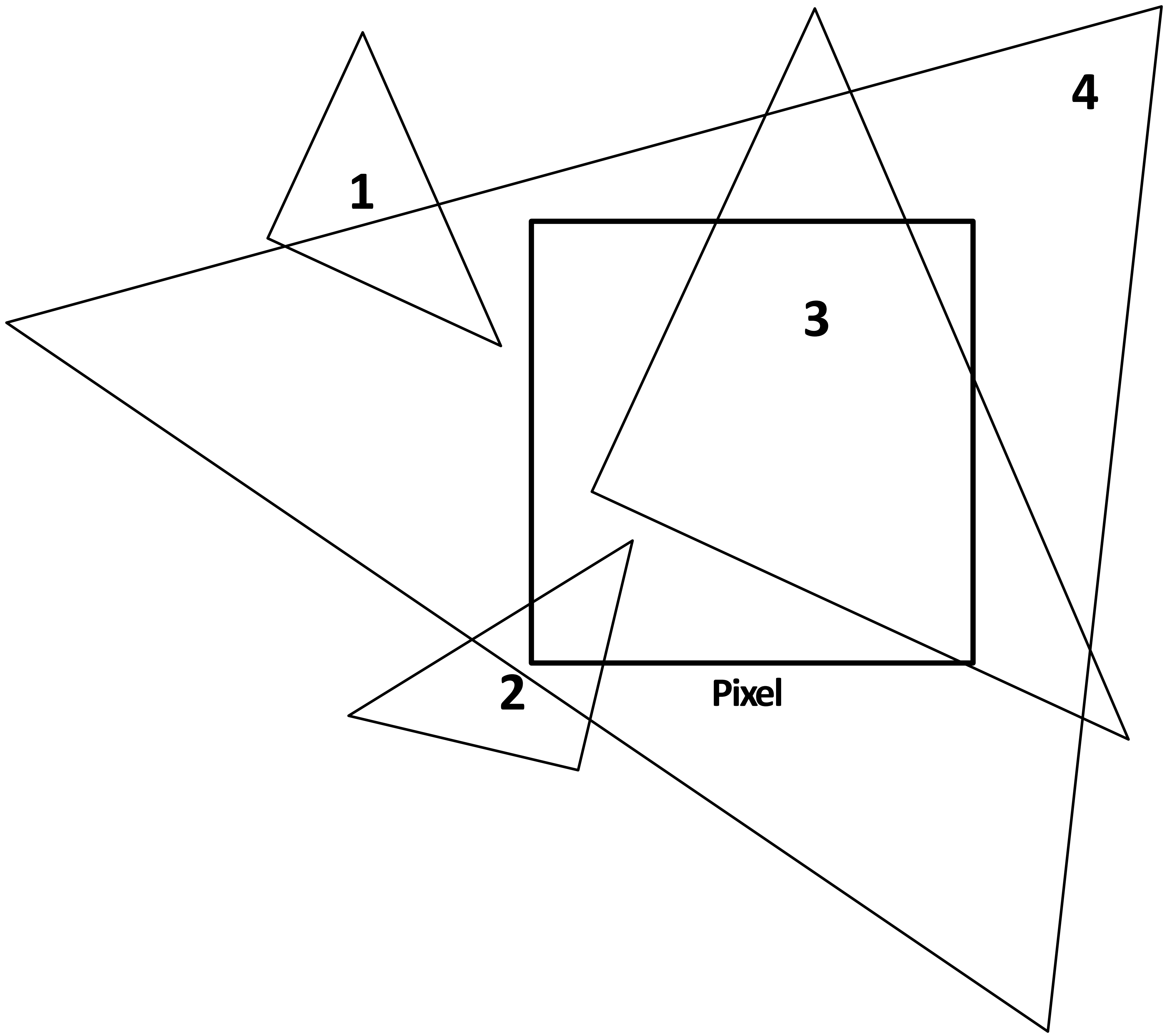


- Why triangles?
  - Most basic polygon
    - Can break up other polygons into triangles
    - Allows programs to optimize one implementation
  - Triangles have unique properties
    - Guaranteed to be planar
    - Well-defined interior
    - Well-defined method for interpolating values at vertices over triangle.



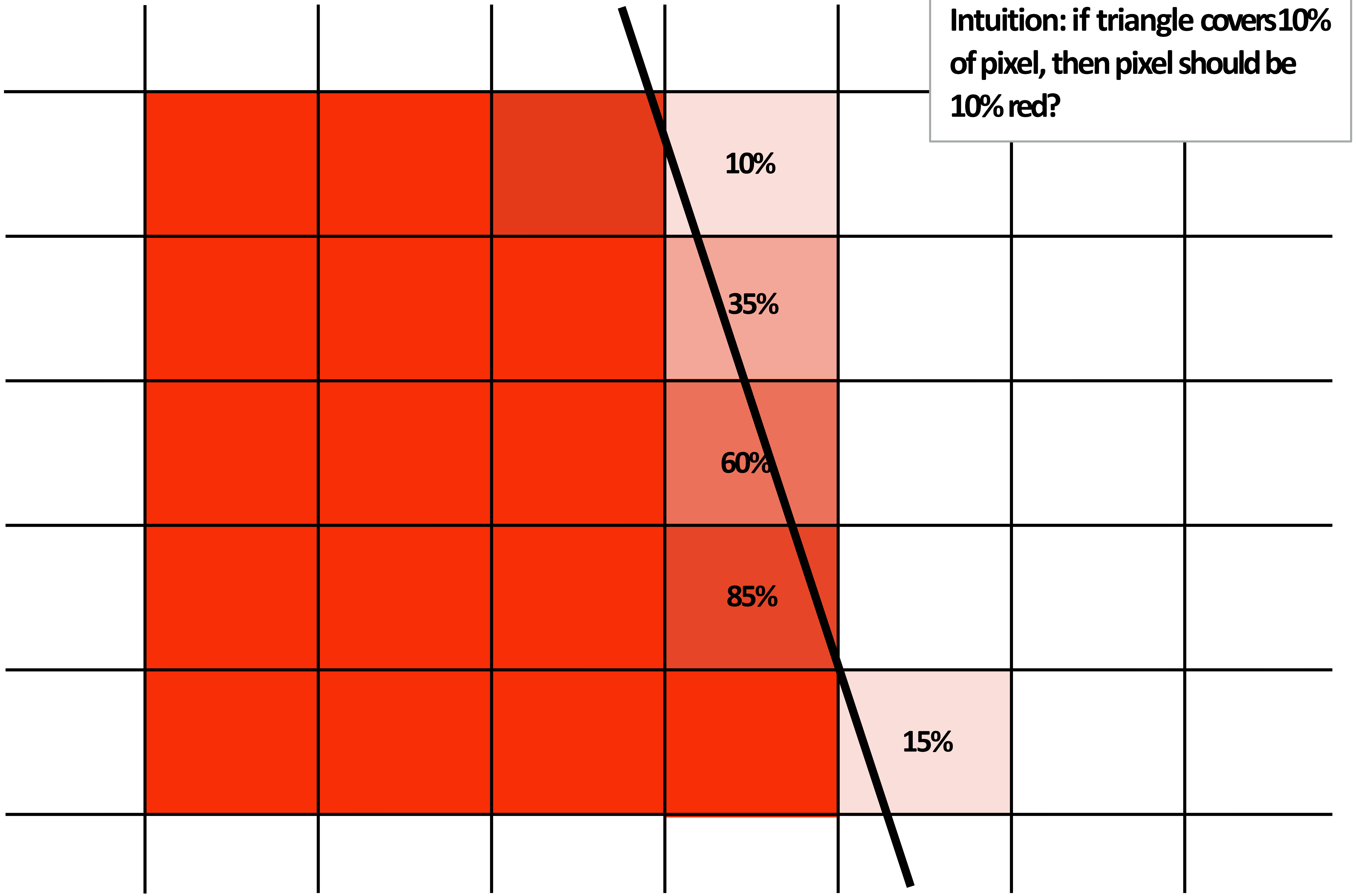
What does it mean for a pixel to be covered by a triangle?

Question: which triangles "cover" this pixel?

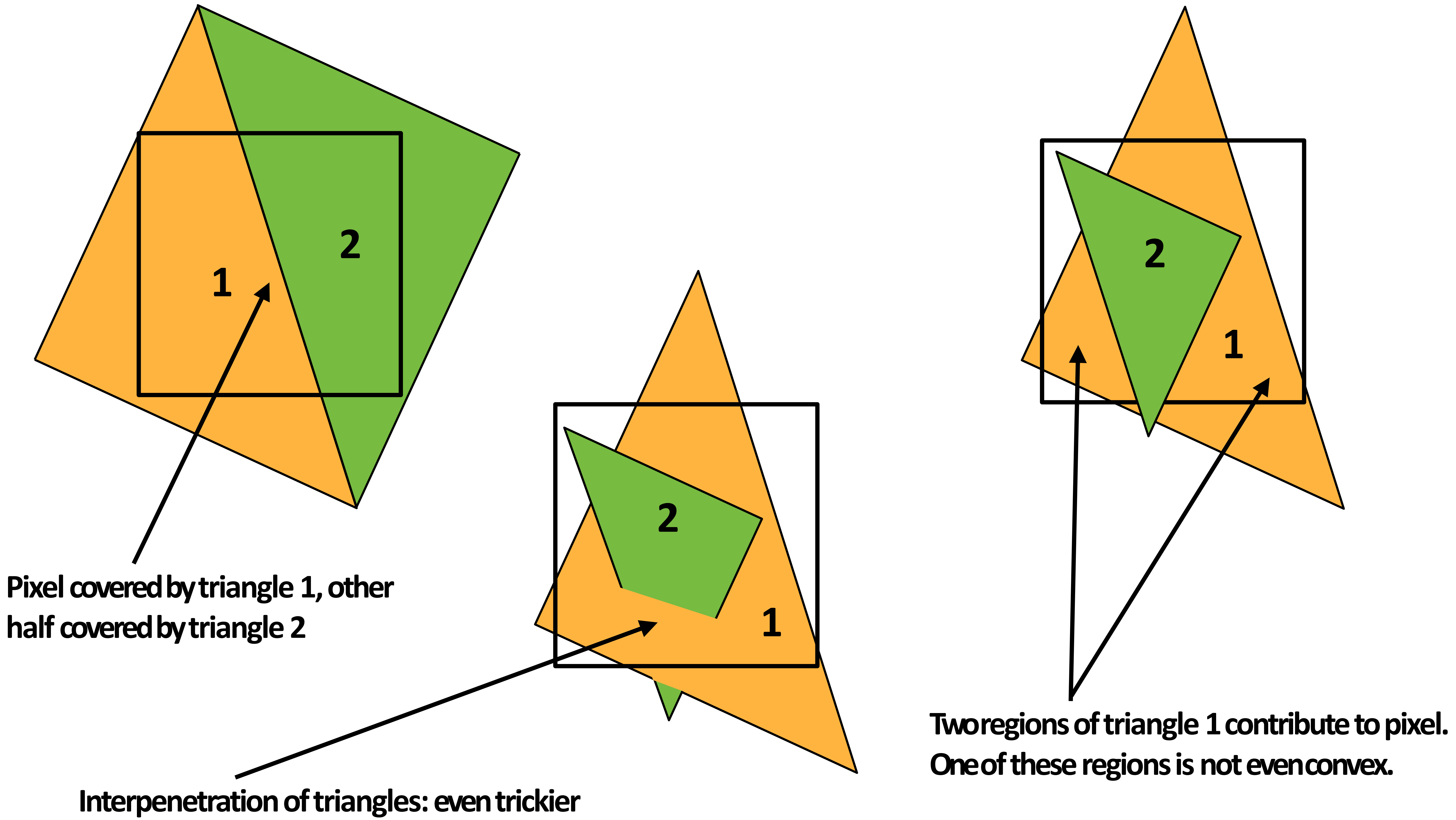


One option: compute fraction of pixel area covered by triangle, then color pixel according to this fraction.

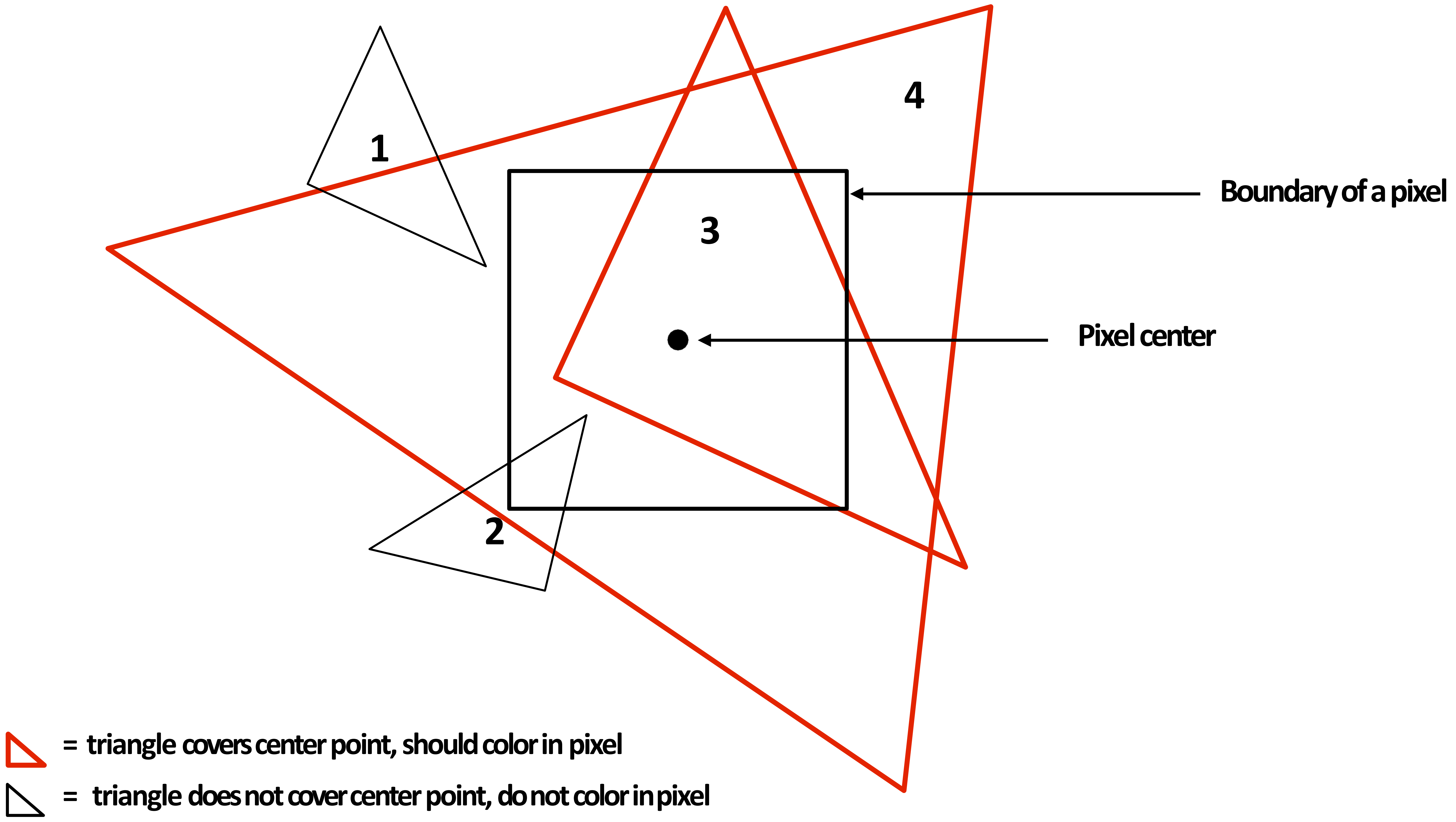
Intuition: if triangle covers 10% of pixel, then pixel should be 10% red?



Analytical coverage schemes get tricky when considering occlusion of one triangle by another

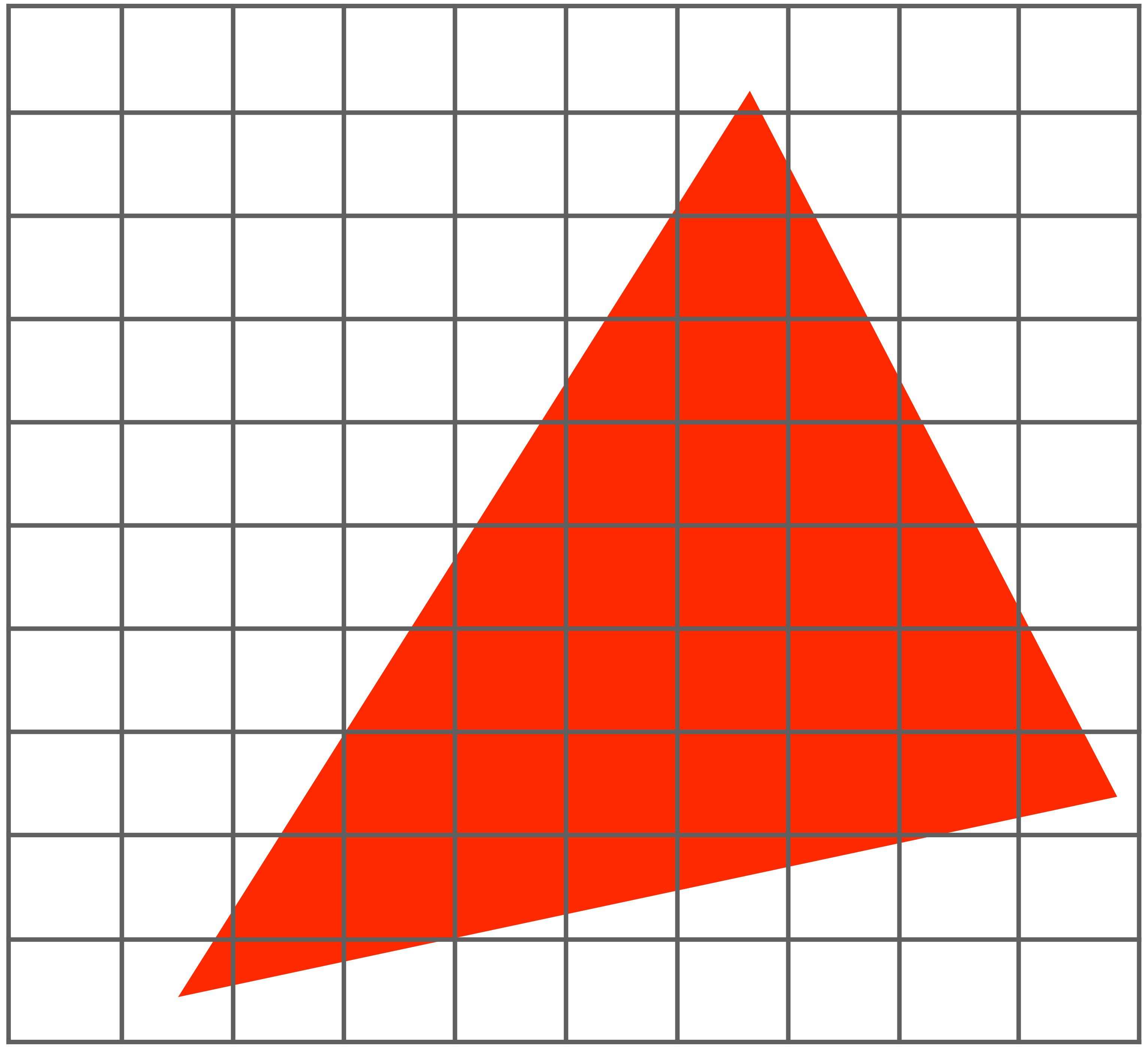


Idea: let's call a pixel "inside" the triangle if the pixel center is inside the triangle



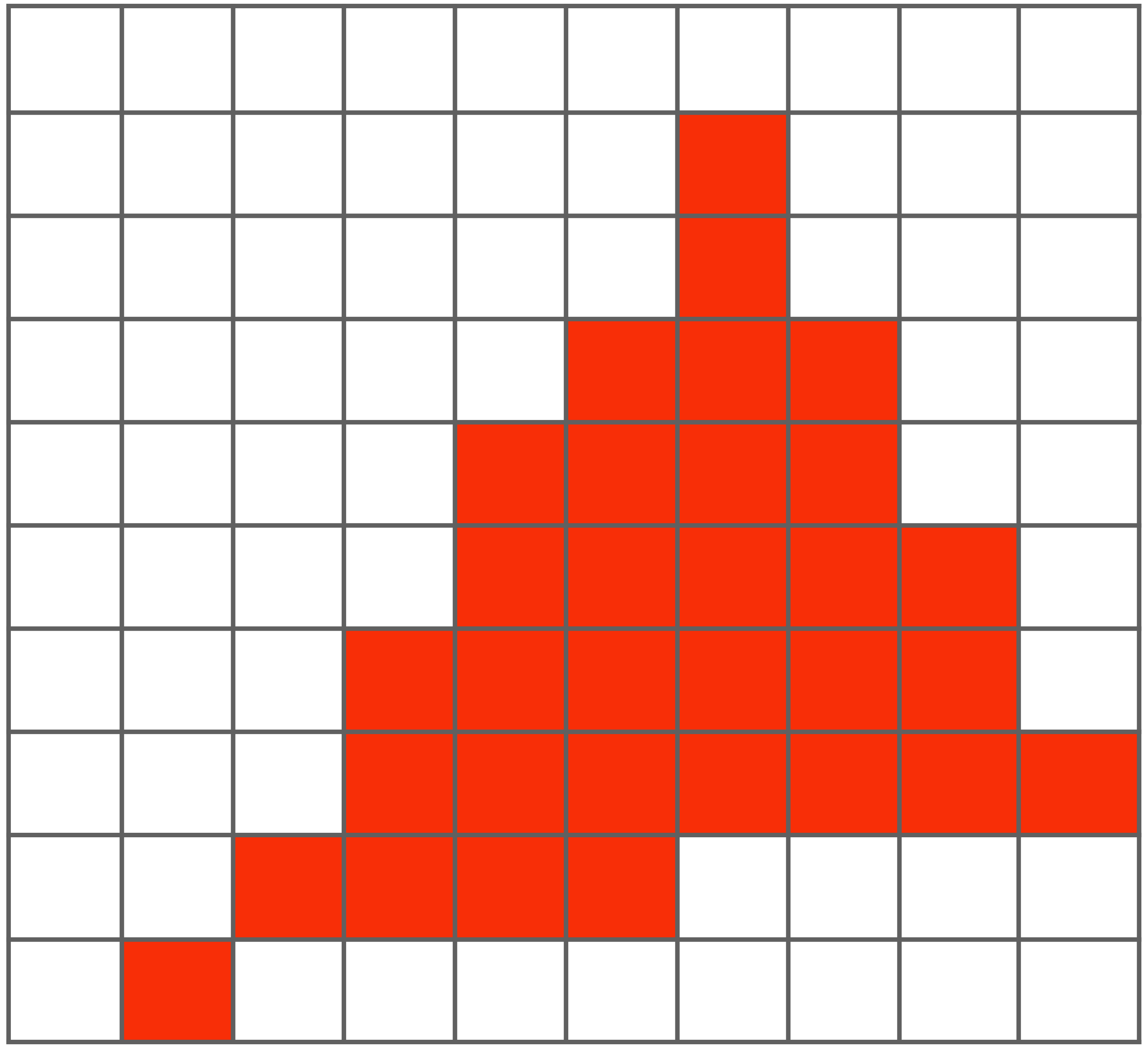
So here's our triangle...

(Overlaid over a pixel grid)



What's wrong with this picture?

(This is the result of rasterizing the triangle using our method)



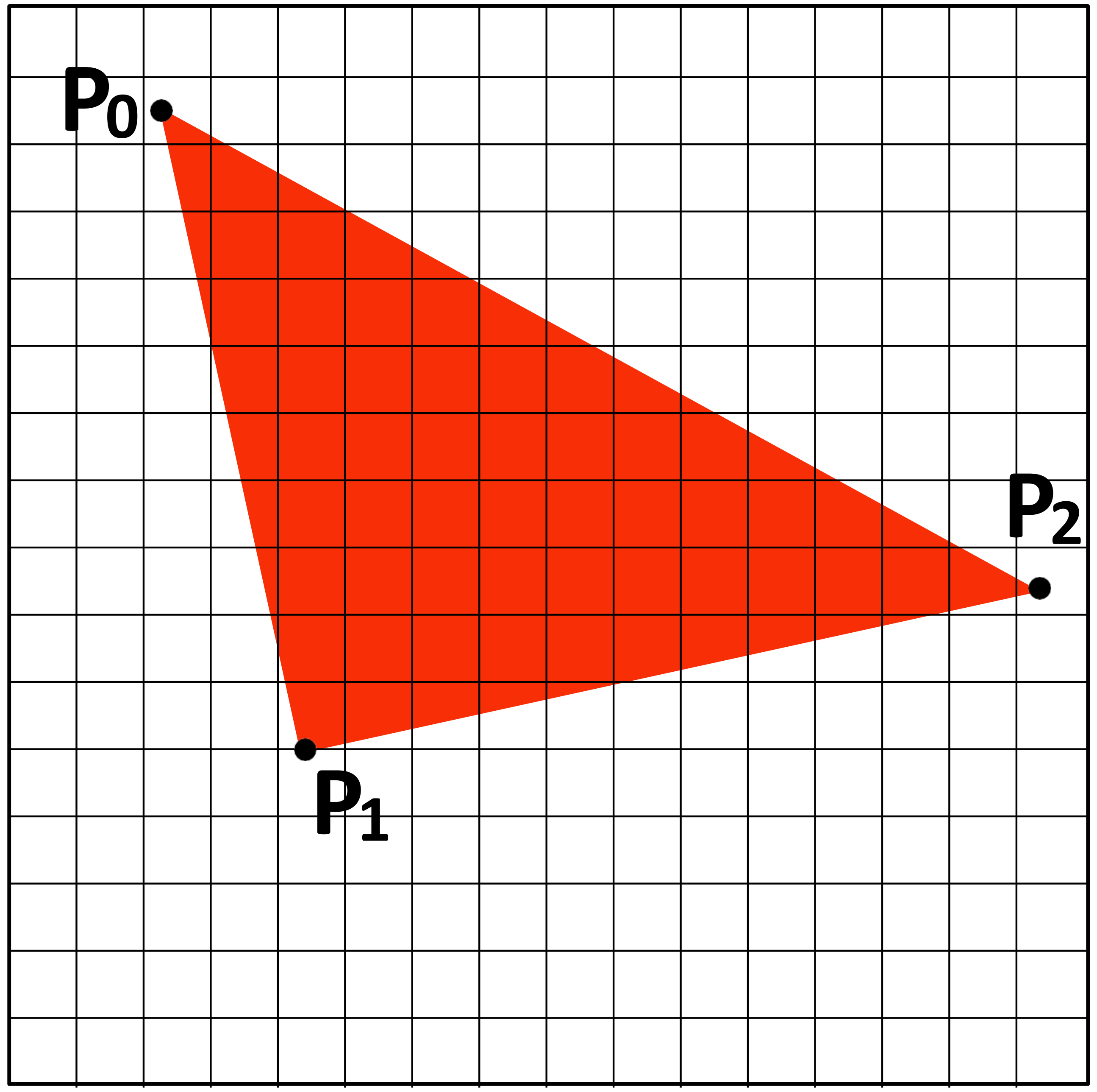
Jaggies!

drawing a triangle

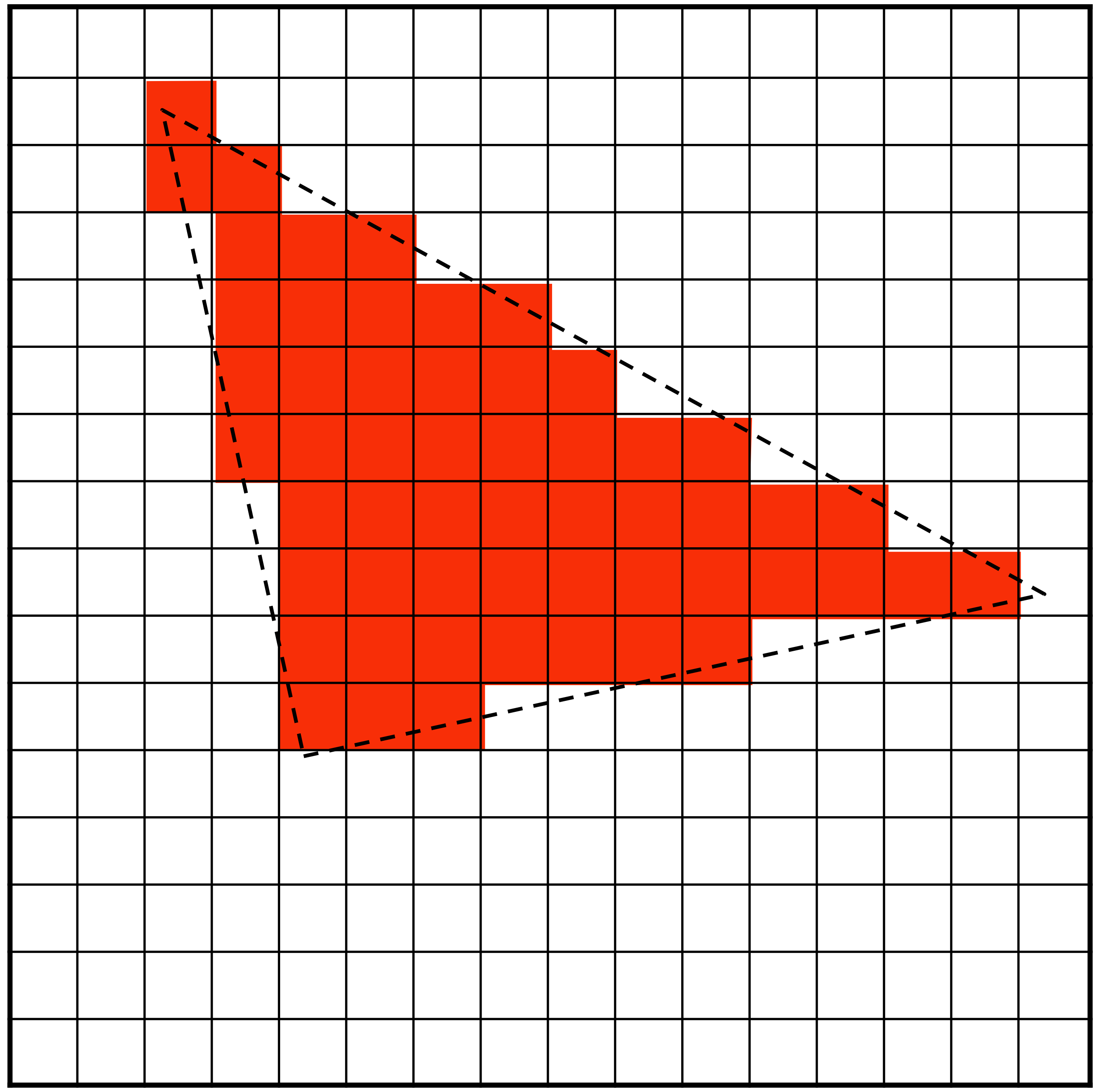
(Converting a representation of a triangle into an image)

"Triangle rasterization"

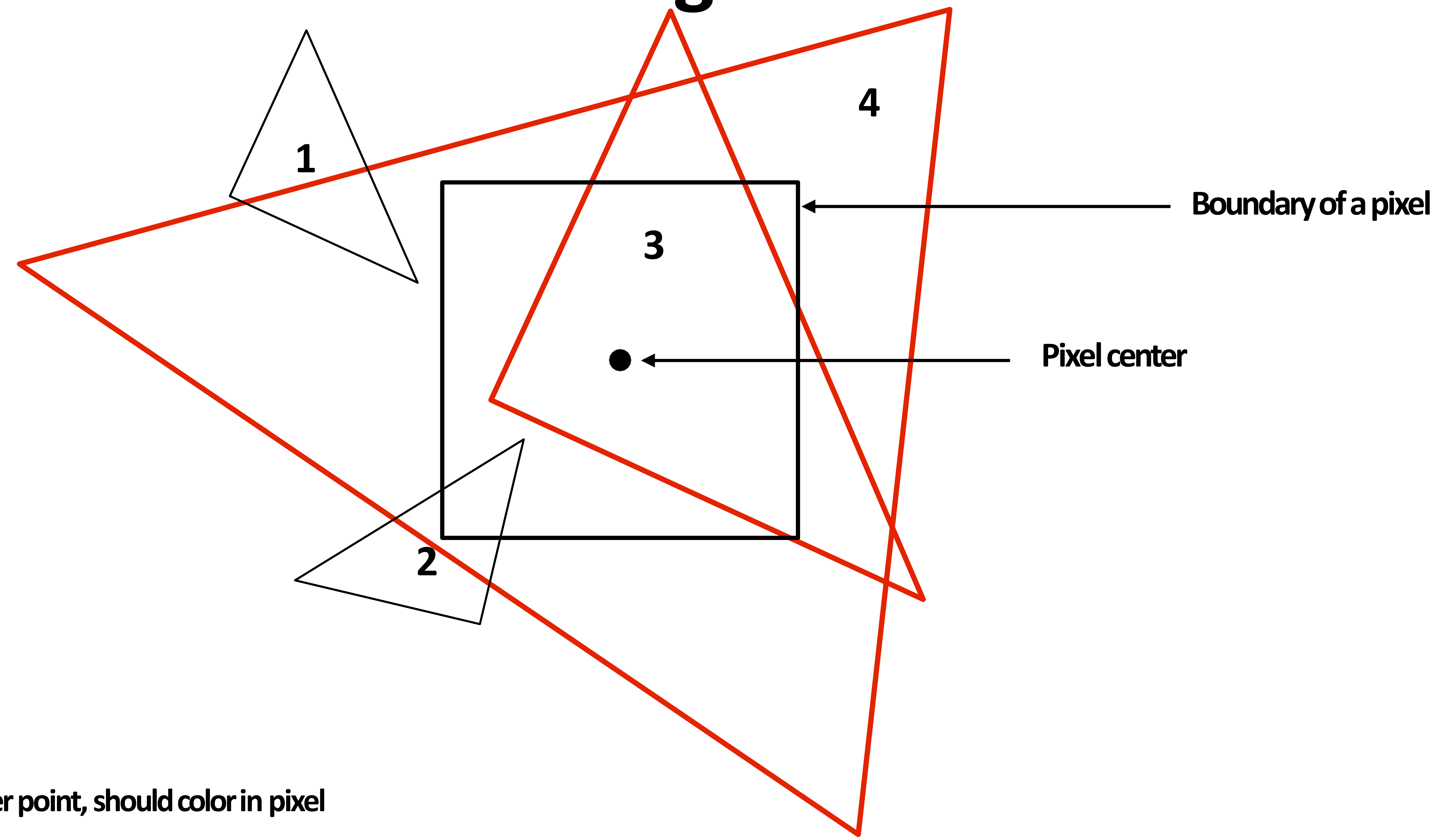
Input:
2D position of triangle vertices: P_0, P_1, P_2

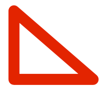


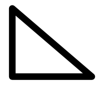
Output:
set of pixels "covered" by the triangle



Idea from last time: let's call a pixel "inside" the triangle if the pixel center is inside the triangle



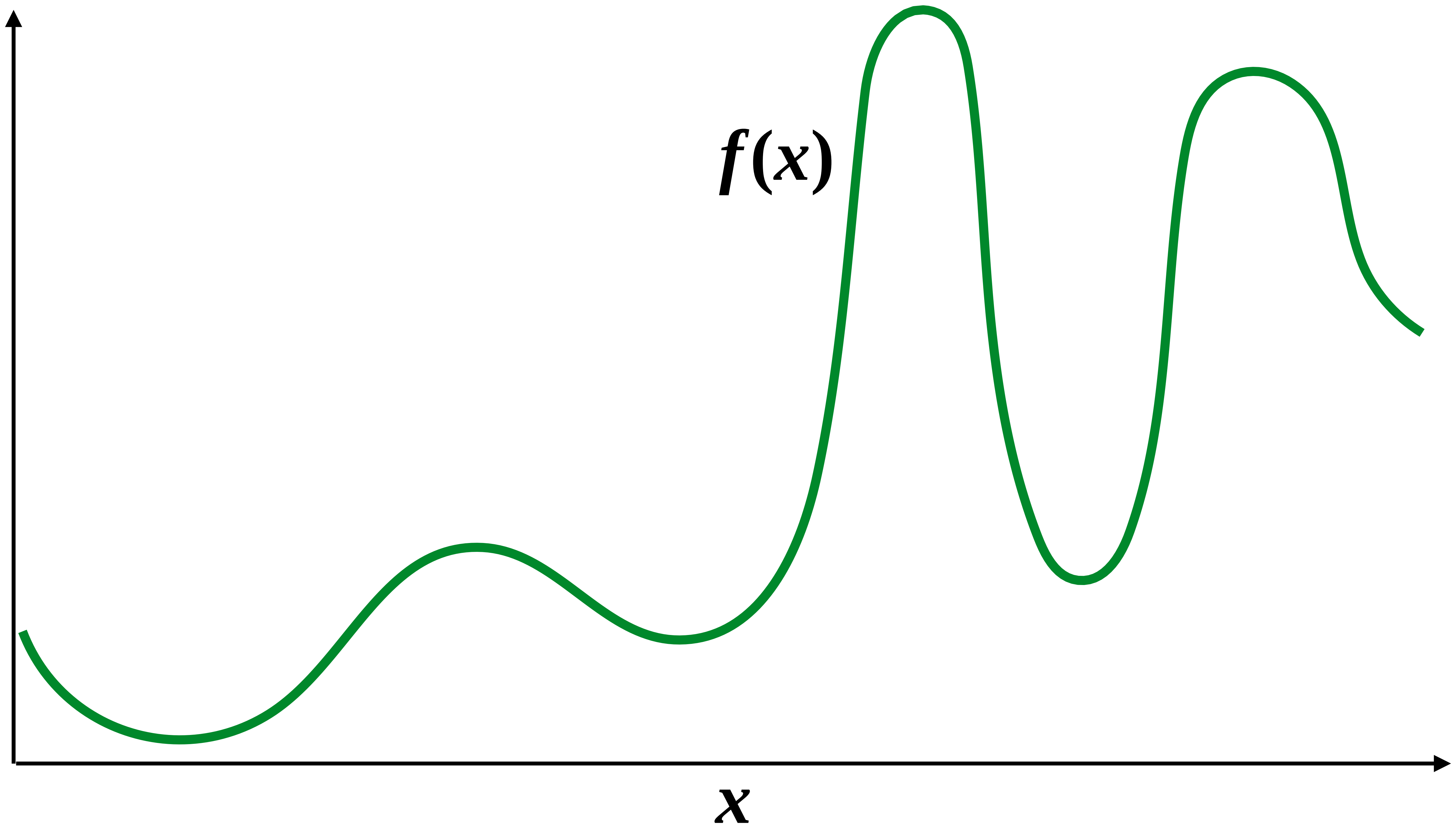
 = triangle covers center point, should color in pixel

 = triangle does not cover center point, do not color in pixel

Today we will draw triangles using a simple method:
point sampling
(testing whether a specific points are inside the triangle)

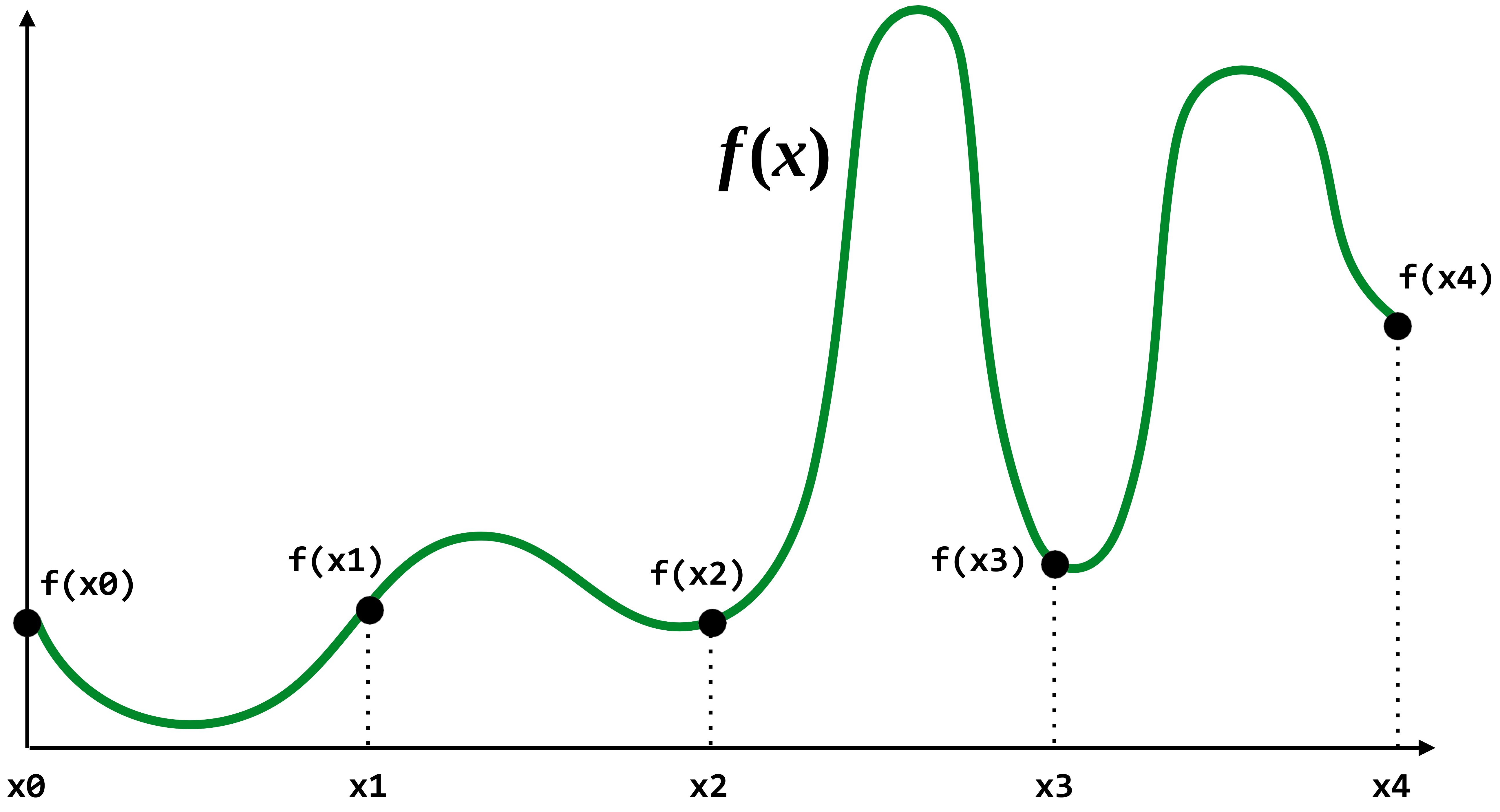
Before talking about sampling in 2D,
let's consider sampling in 1D first…

Consider a 1D signal: f(x)



Sampling: taking measurements of a signal

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



A discrete representation of f(x) is given by the samples $f(x_0)$, $f(x_1)$, $f(x_2)$, $f(x_3)$, $f(x_4)$

Audio file: stores samples of a 1D signal

Audio is often sampled at 44.1 KHz

Sampling a function

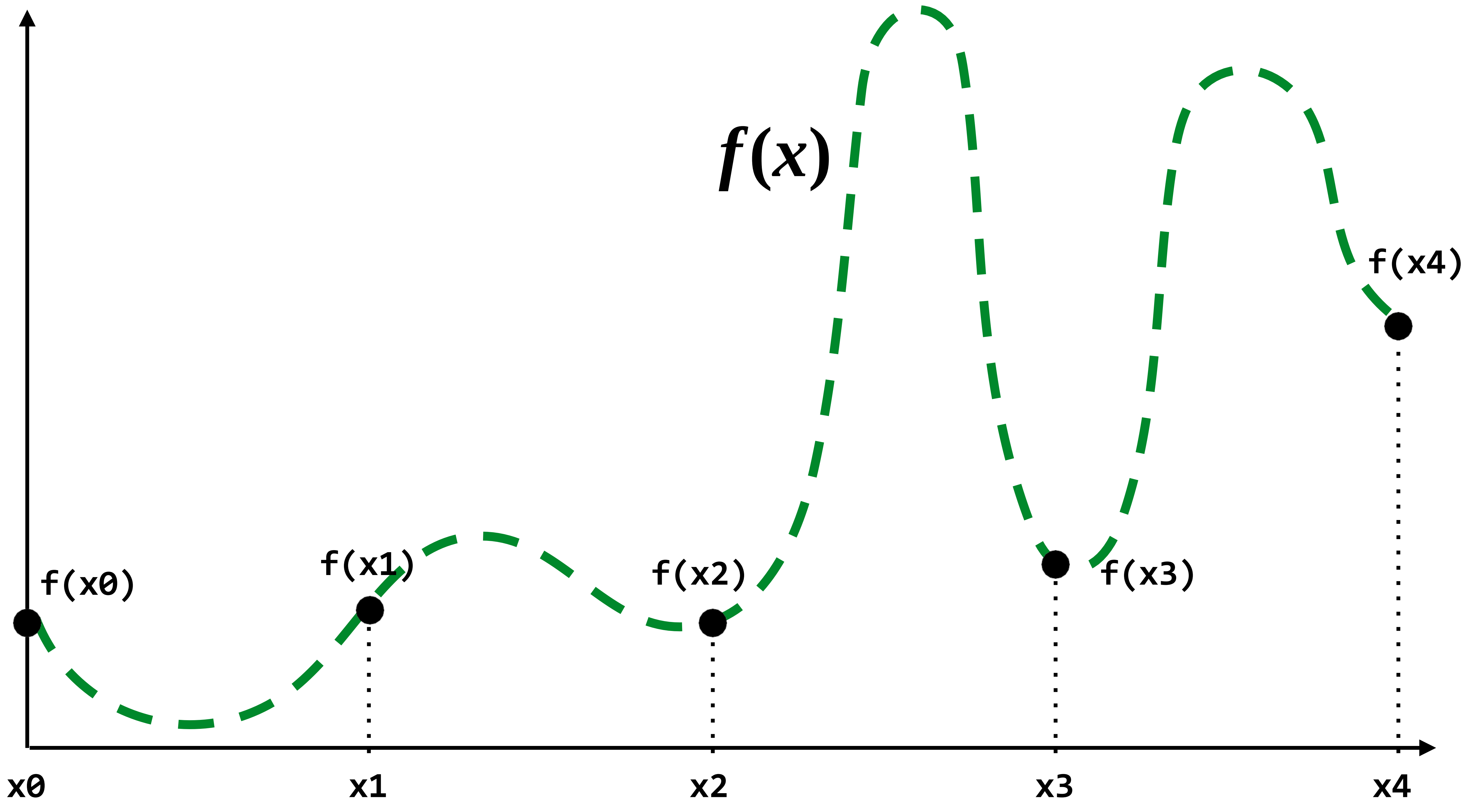
- Evaluating a function at a point is sampling the function's value

- We can discretize a function by periodic sampling

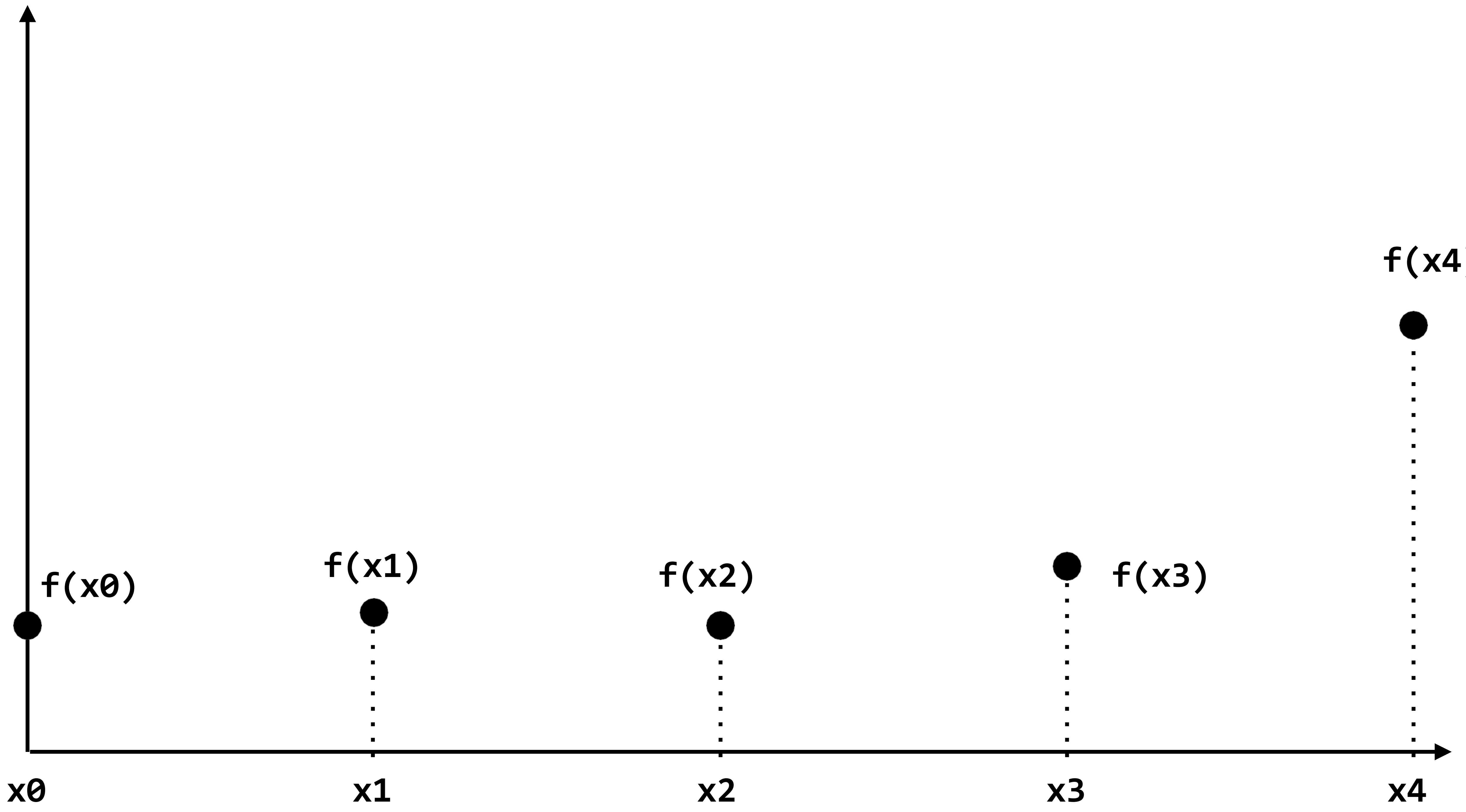
```
for(int x = 0; x < xmax; x++)
    output[x] = f(x);
```

- Sampling is a core idea in graphics. In this class we'll sample signals parameterized by: time (1D), area (2D), angle (2D), volume (3D), paths through a scene (infinite-D) etc ...

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



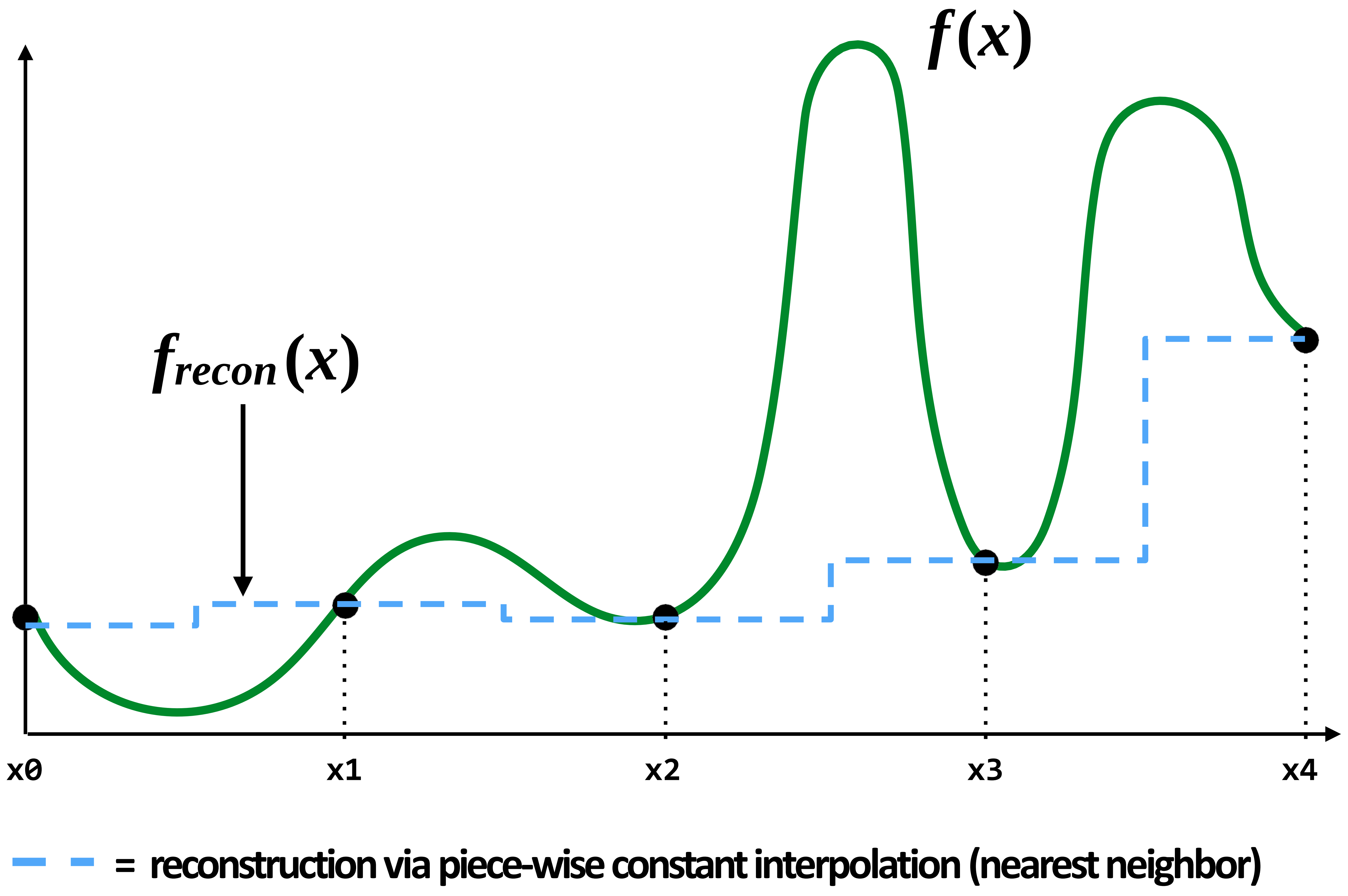
Reconstruction: given a set of samples, how might we attempt to reconstruct the original (continuous) signal $f(x)$?



Piecewise constant approximation

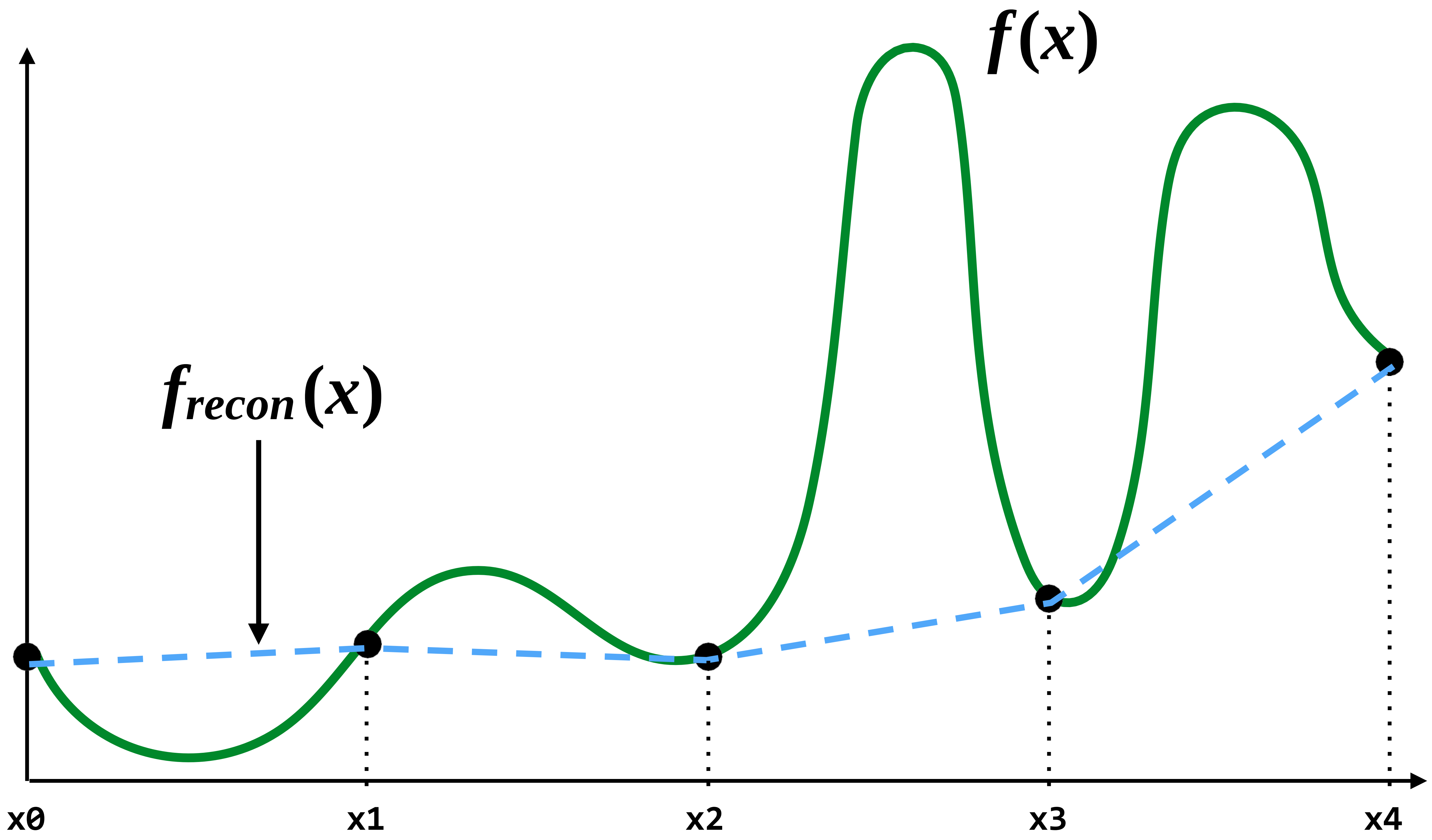
$f_{recon}(x)$ = value of sample closest to x

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



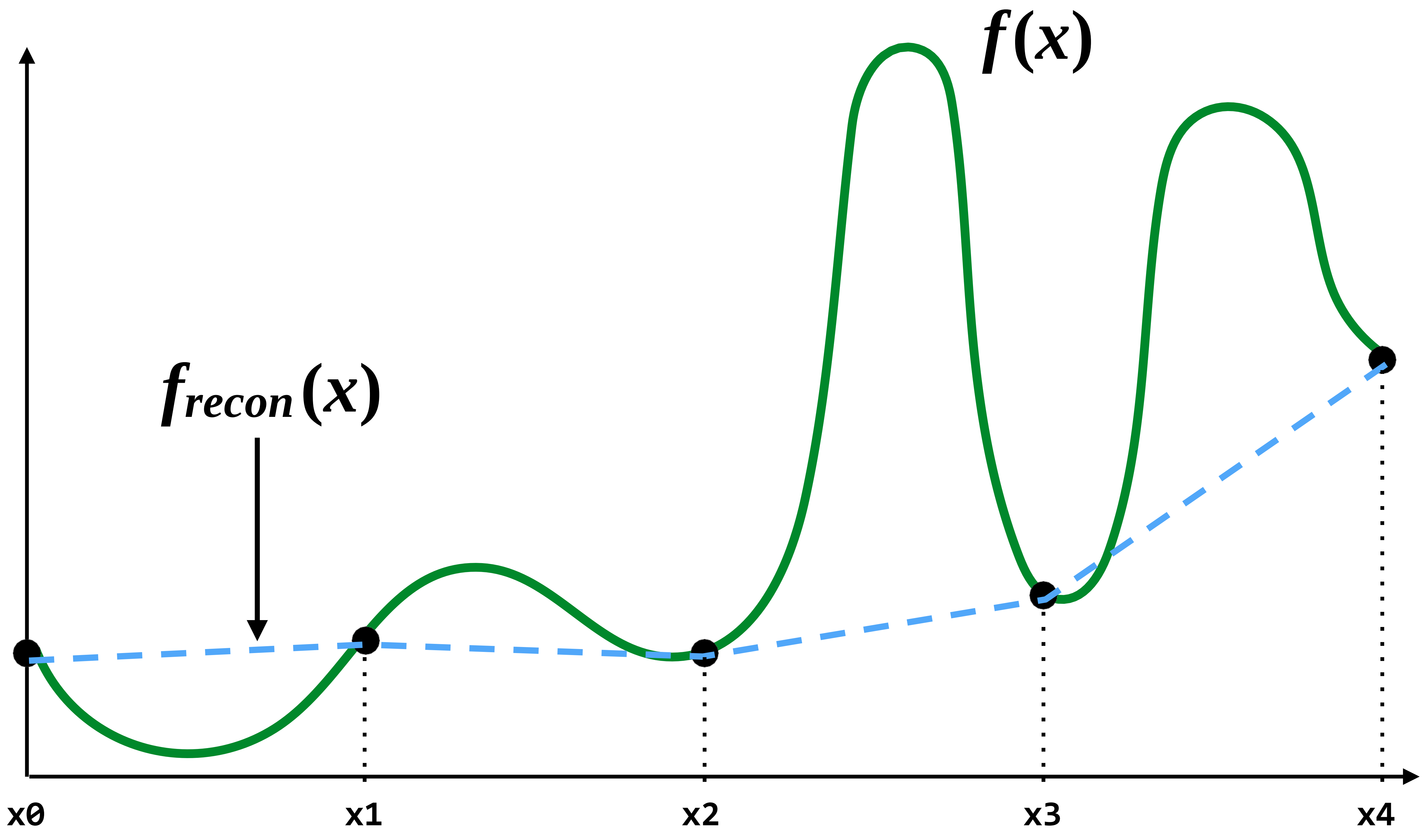
Piecewise linear approximation

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



= reconstruction via linear interpolation

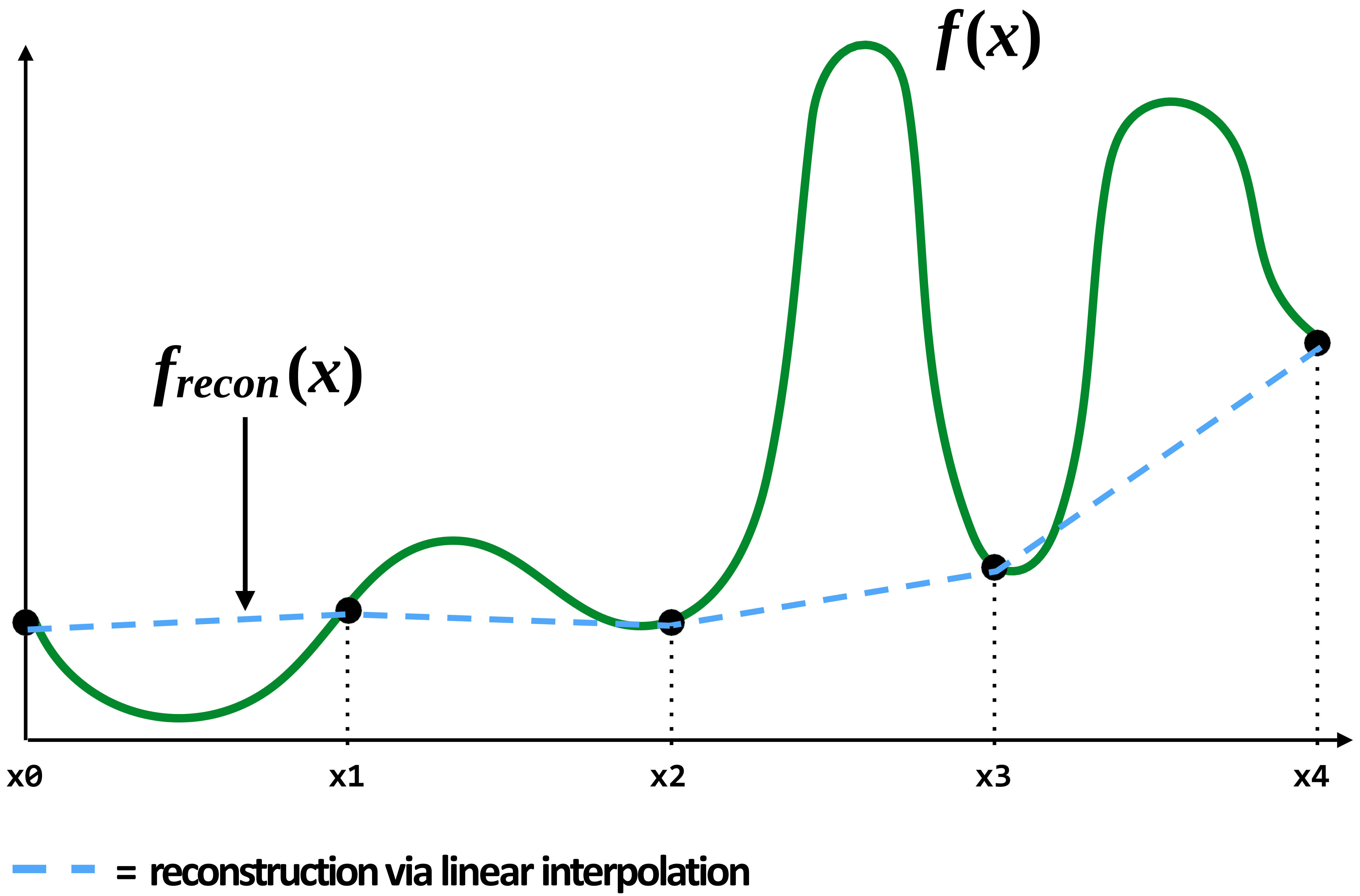
How can we represent the signal more accurately?



Answer: sample signal more densely (increase sampling rate)

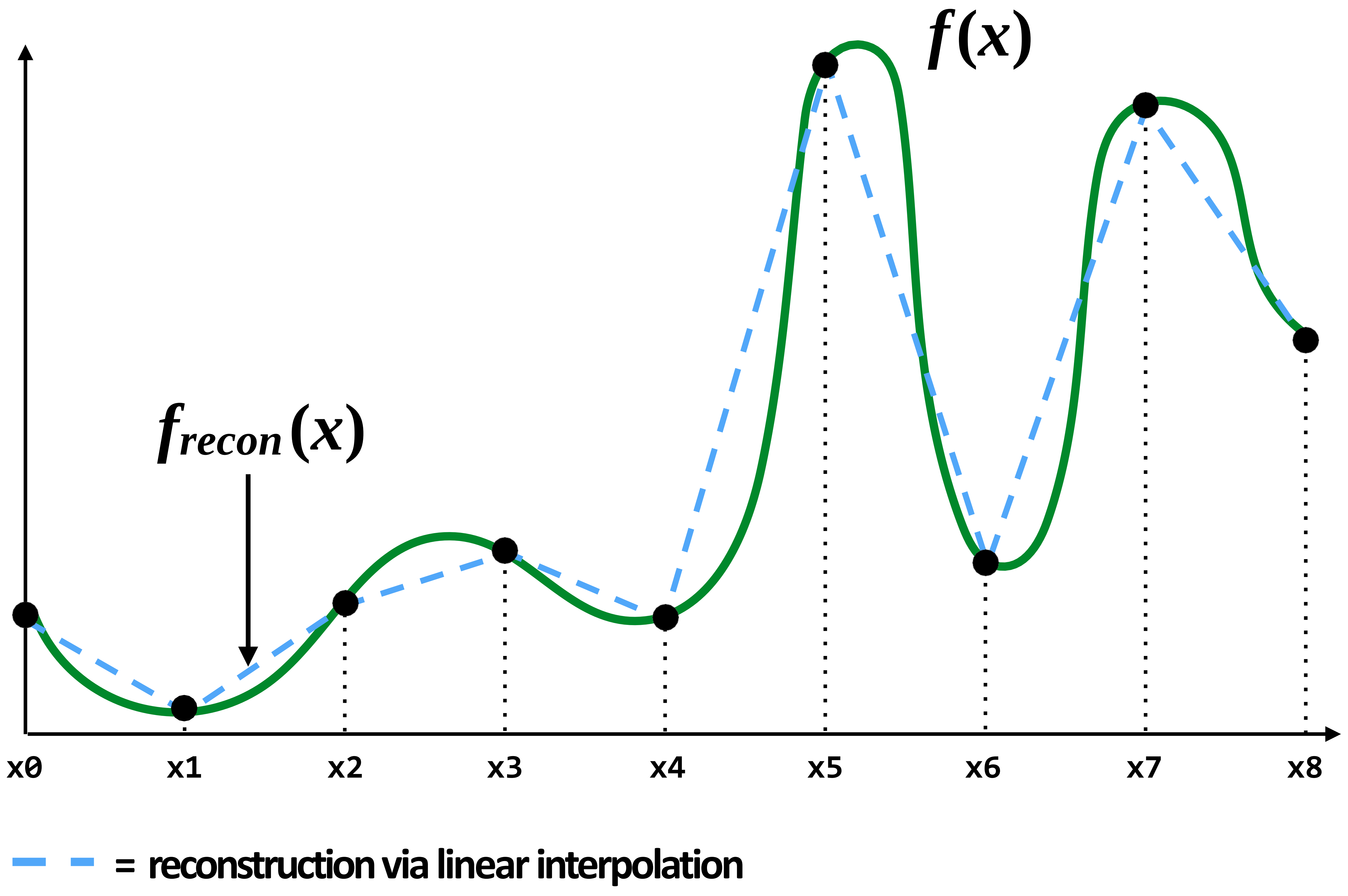
Reconstruction from sparse sampling

(5 samples)



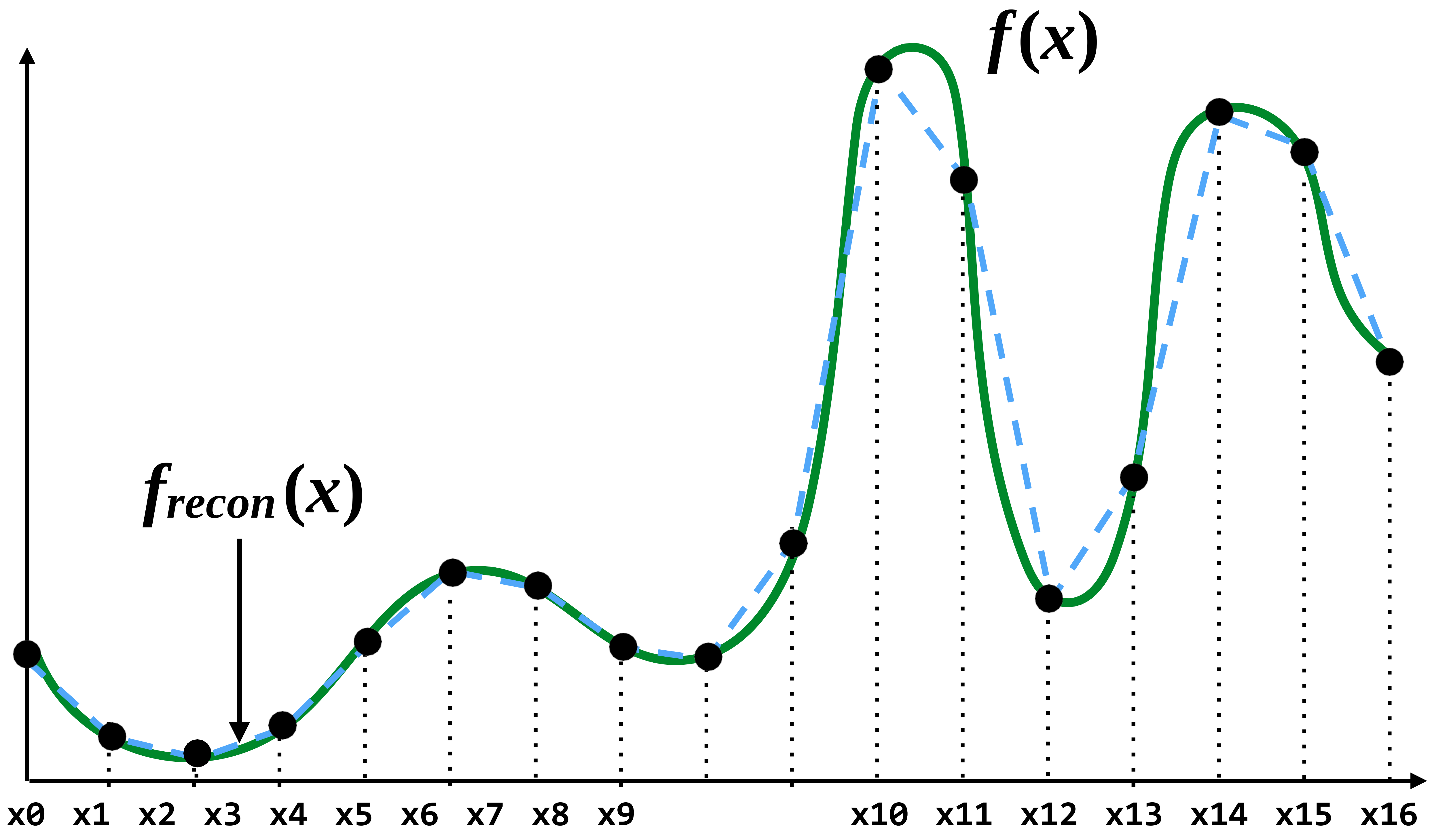
More accurate reconstructions result from denser sampling

(9 samples)



More accurate reconstructions result from denser sampling

(17 samples)



= reconstruction via linear interpolation

Drawing a triangle by 2D sampling

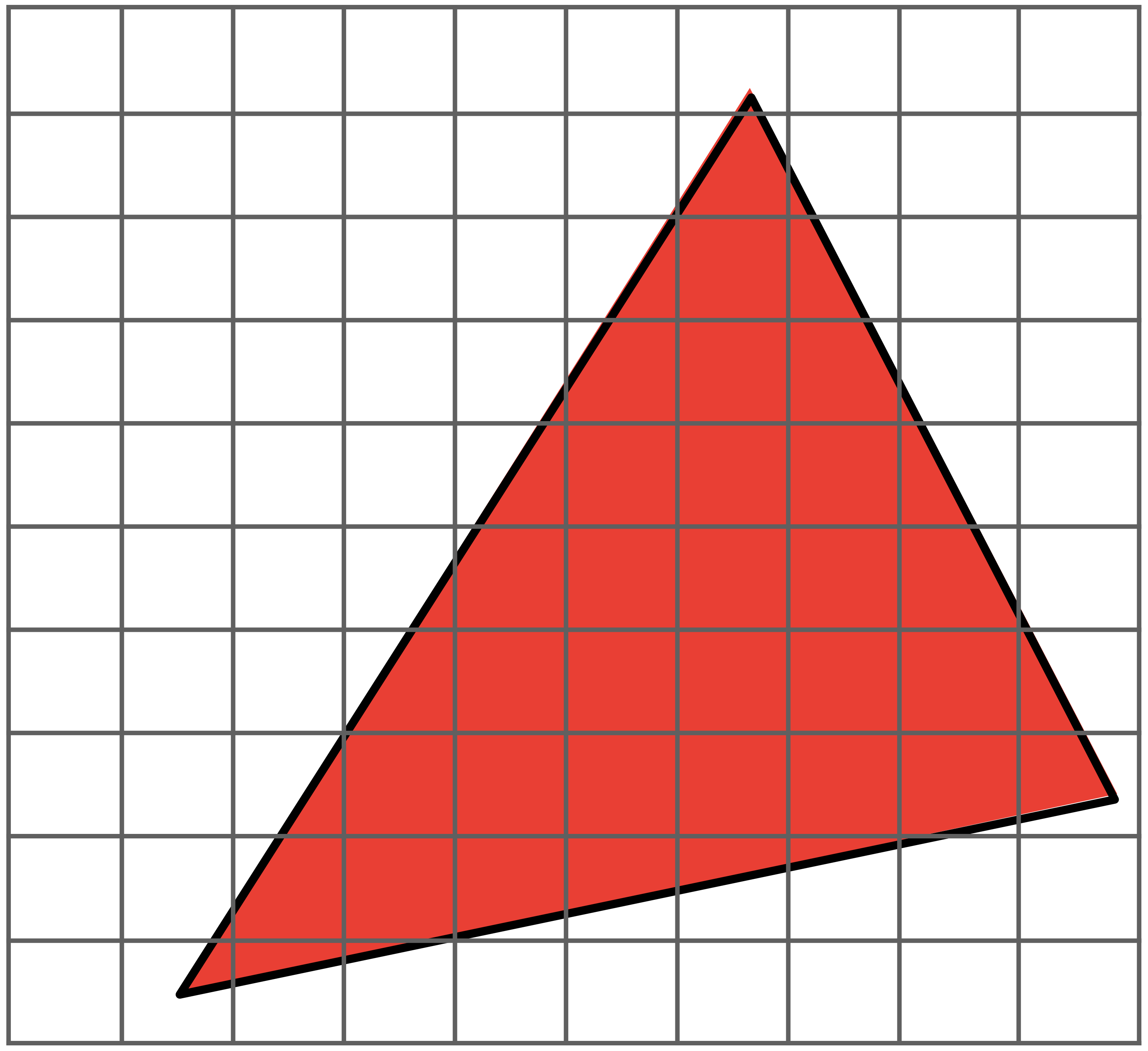


Image as a 2D matrix of pixels

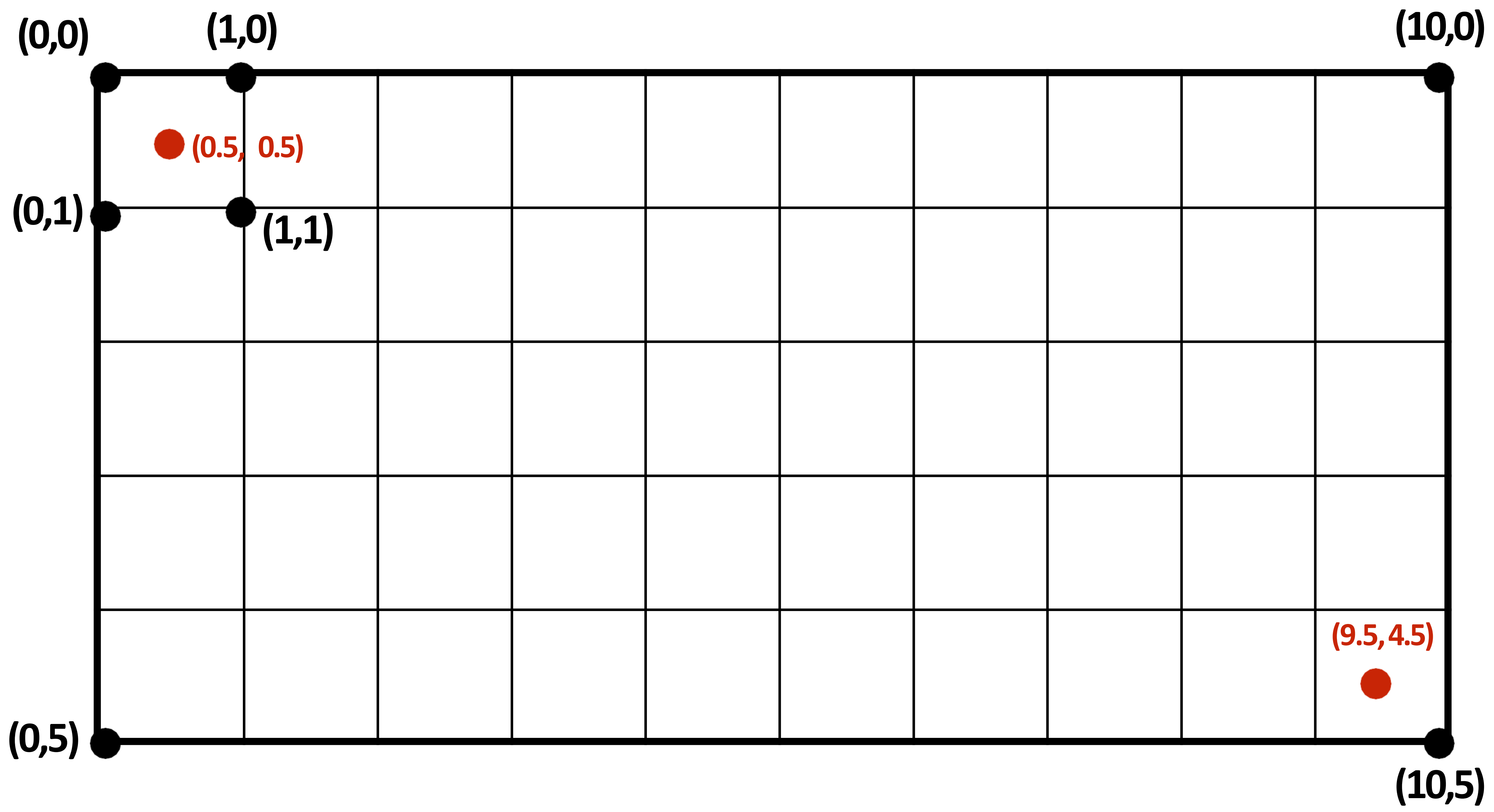
Here I'm showing a 10 x 5 pixel image

Identify pixel by its integer (x,y) coordinates

(0,0)	(1,0)								(9,0)
(0,1)	(1,1)								
(0,4)									(9,4)

Continuous coordinate space over image

Ok, now forget about pixels!

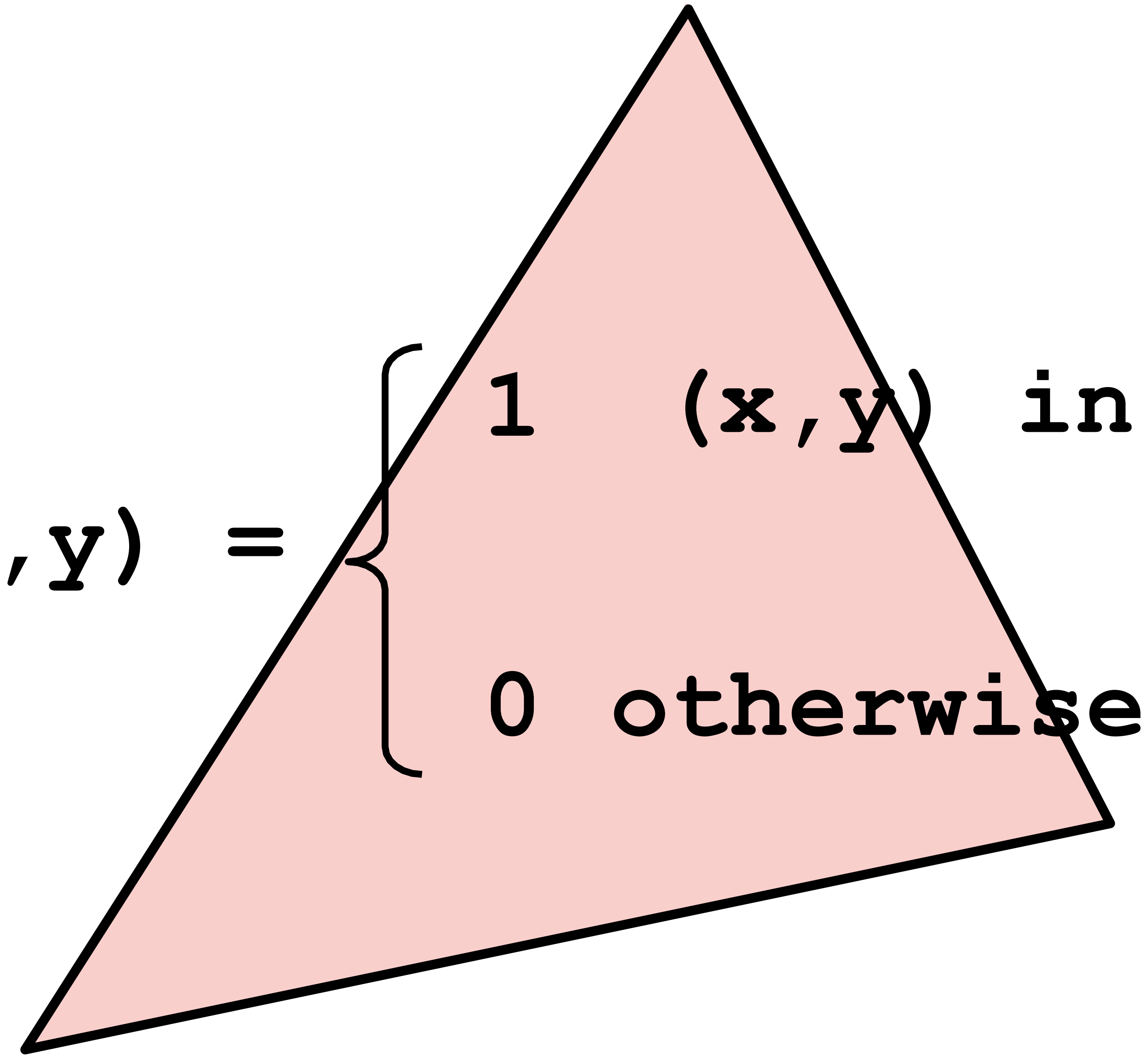


Continuous coordinate space over image

Ok, now forget about pixels!

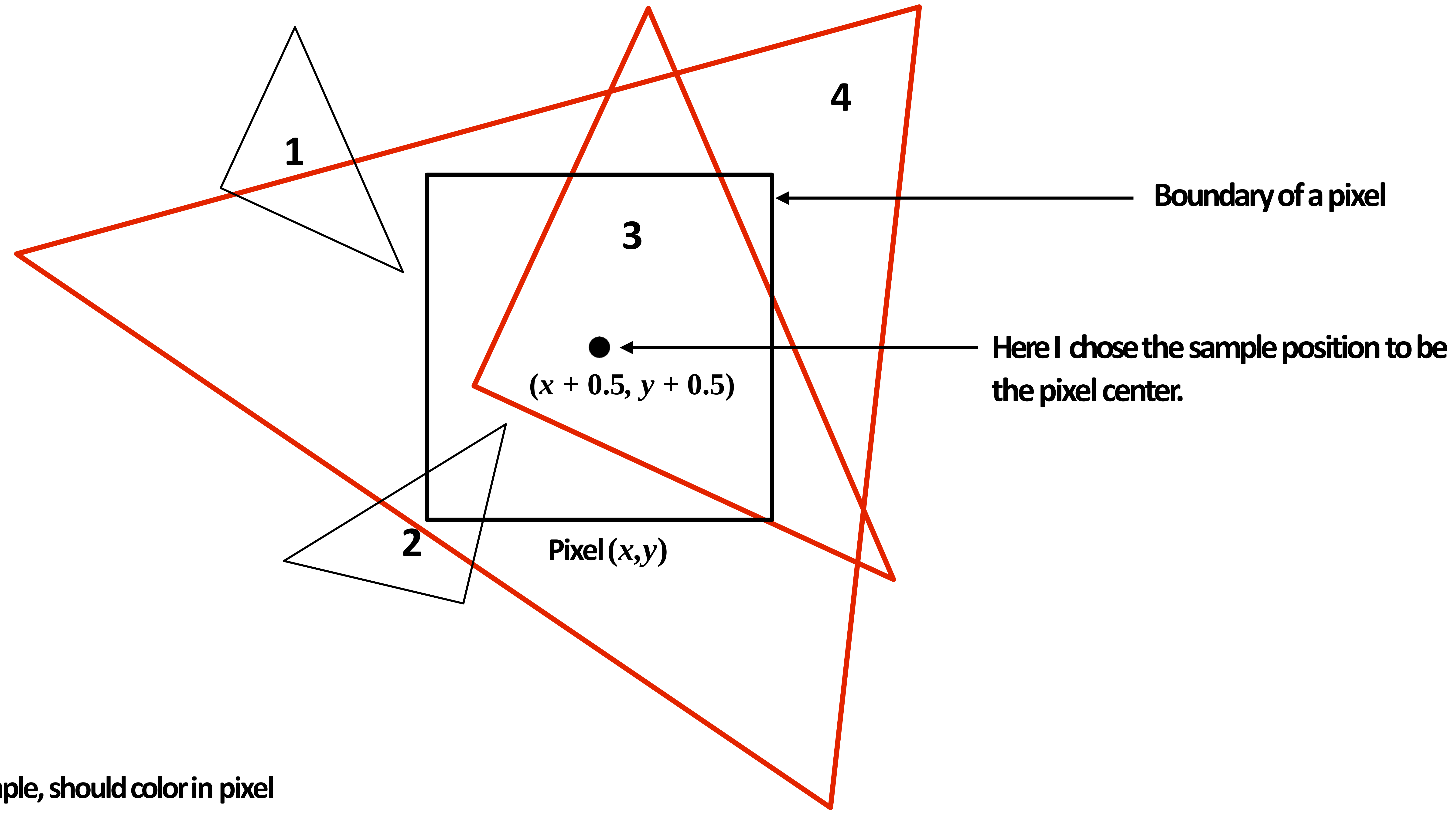
(I removed pixel boundaries from the figure to encourage you to forget about pixels!)

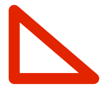
Define binary function: `inside(tri,x,y)`

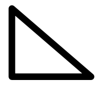


$$\texttt{inside(t,x,y)} = \begin{cases} \texttt{1} & \texttt{(x,y) in triangle t} \\ \texttt{0} & \texttt{otherwise} \end{cases}$$

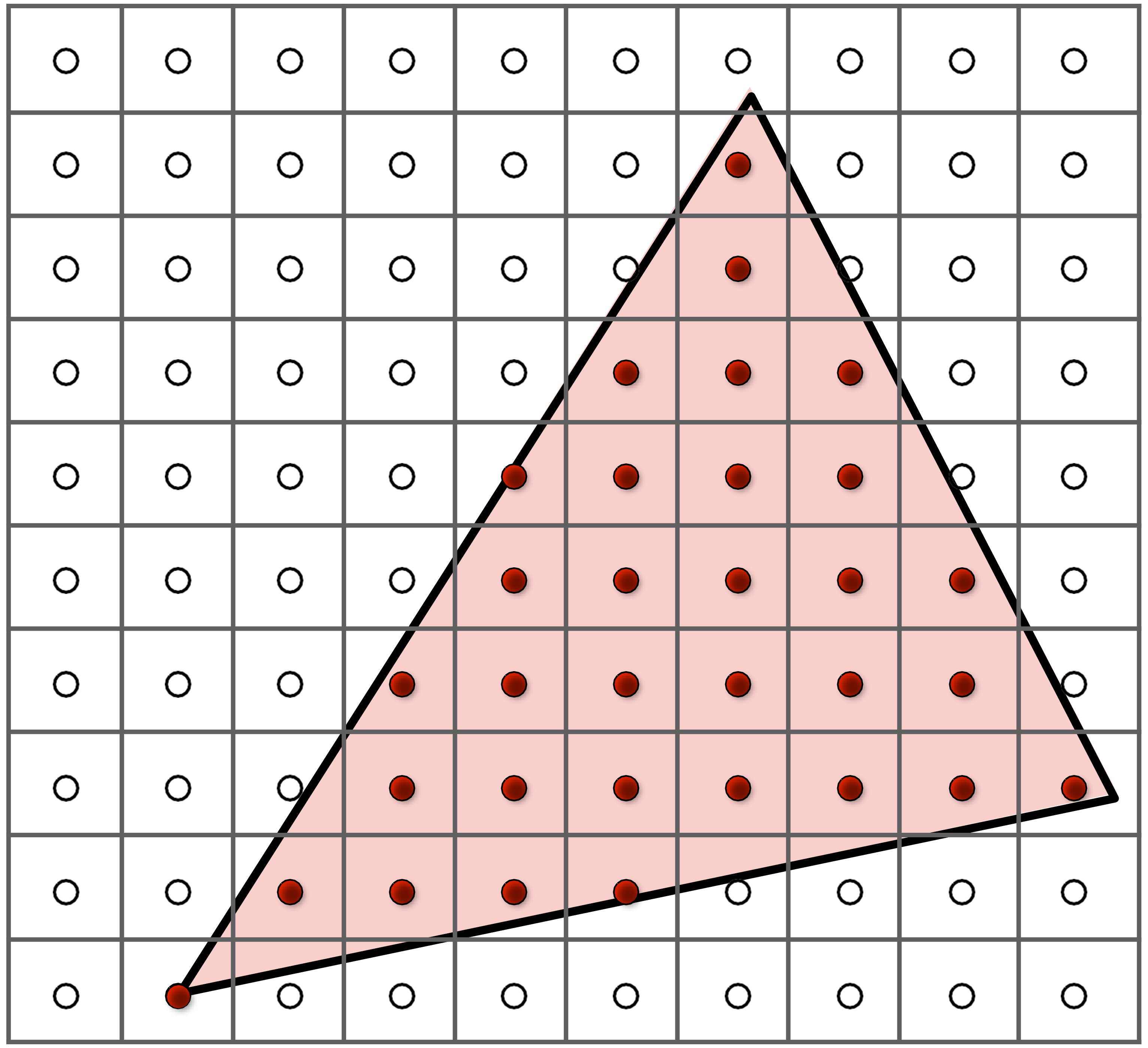
Sampling the binary function: `inside(tri,x,y)`




= triangle covers sample, should color in pixel


= triangle does not cover sample, do not color in pixel

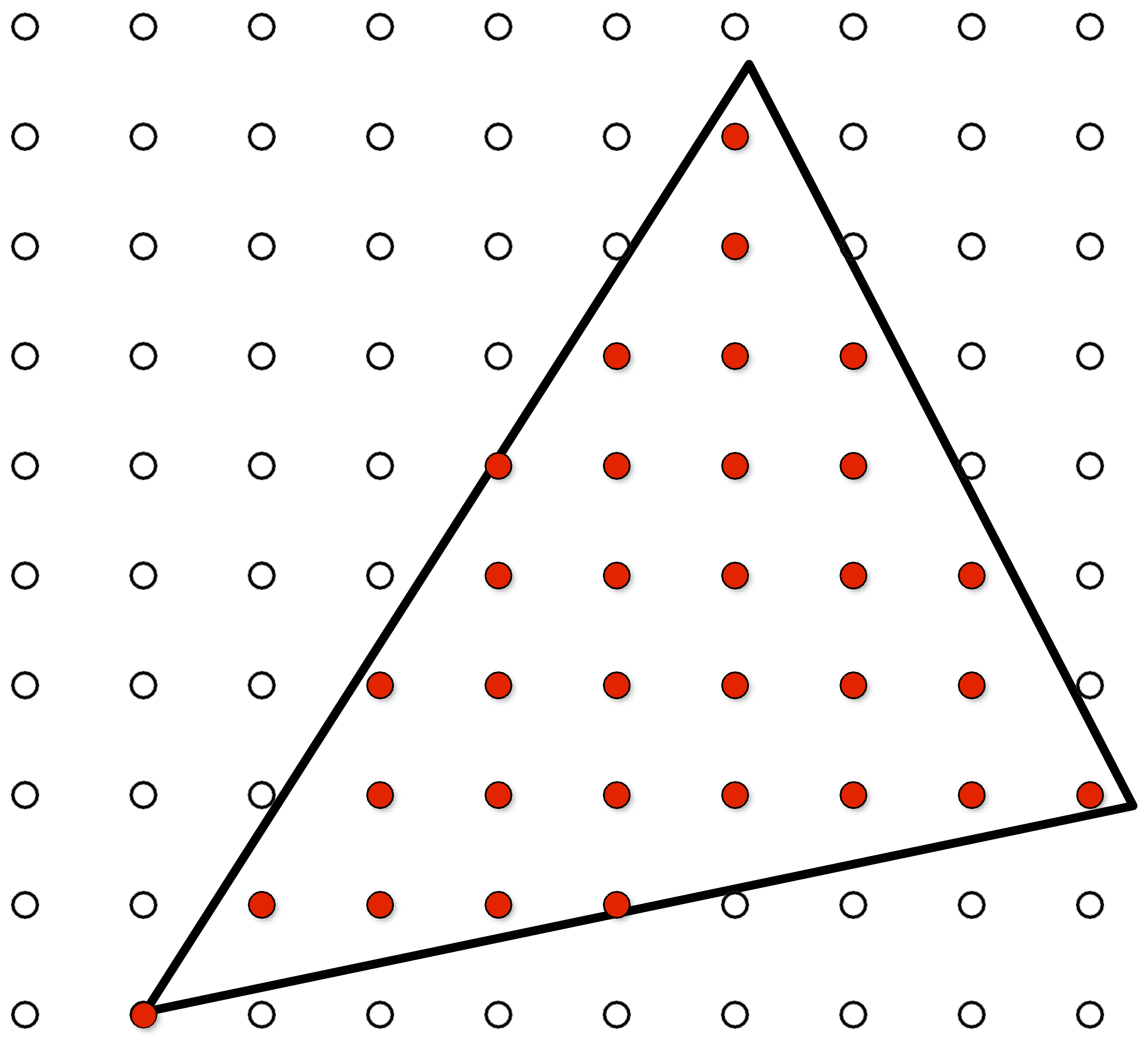
Sample coverage at pixel centers



Sample coverage at pixel centers

I only want you to think about evaluating triangle-point coverage!

NOT TRIANGLE-PIXEL OVERLAP!



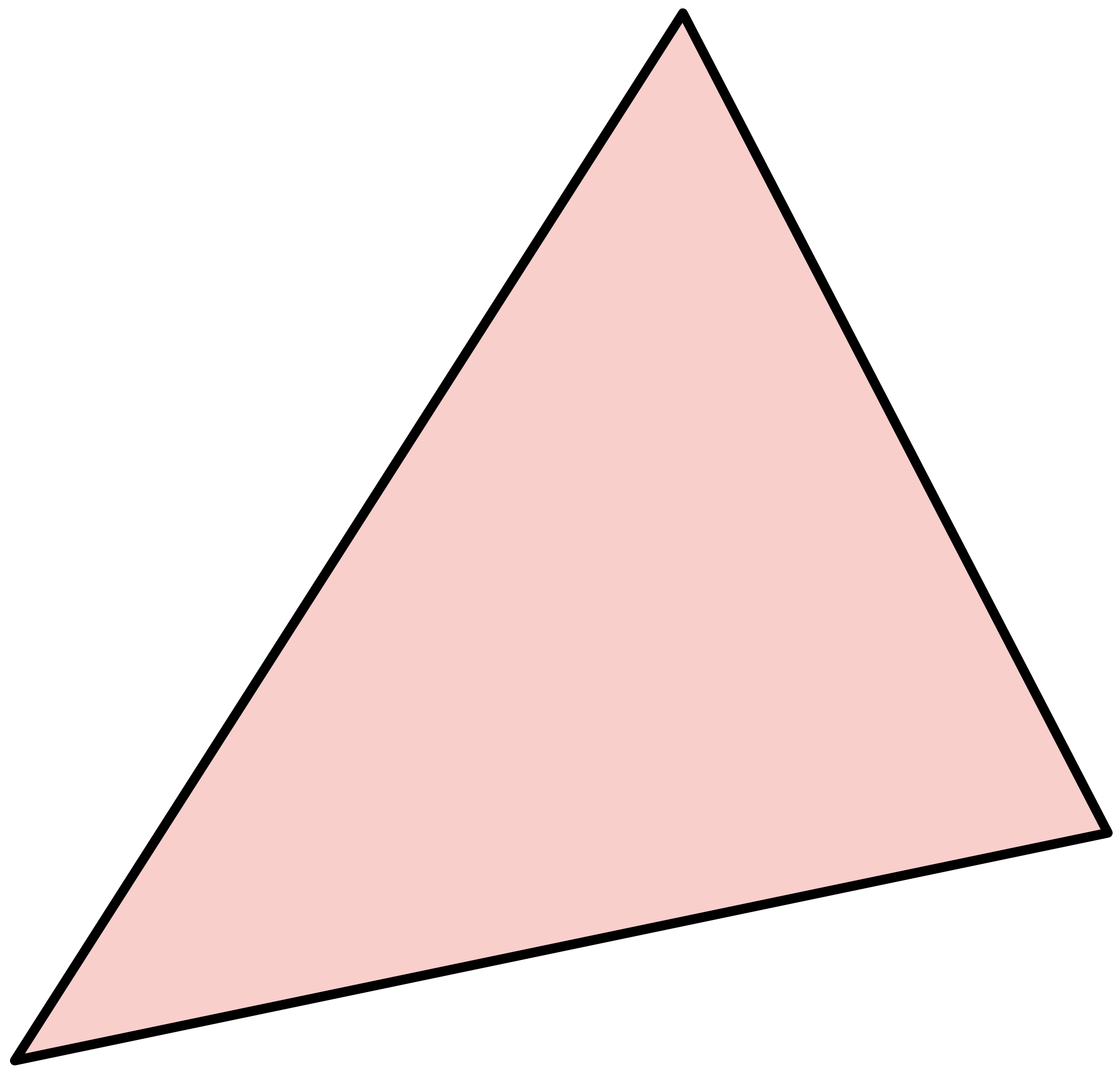
Rasterization = sampling a 2D binary function

- Rasterize triangle `tri` by sampling the function

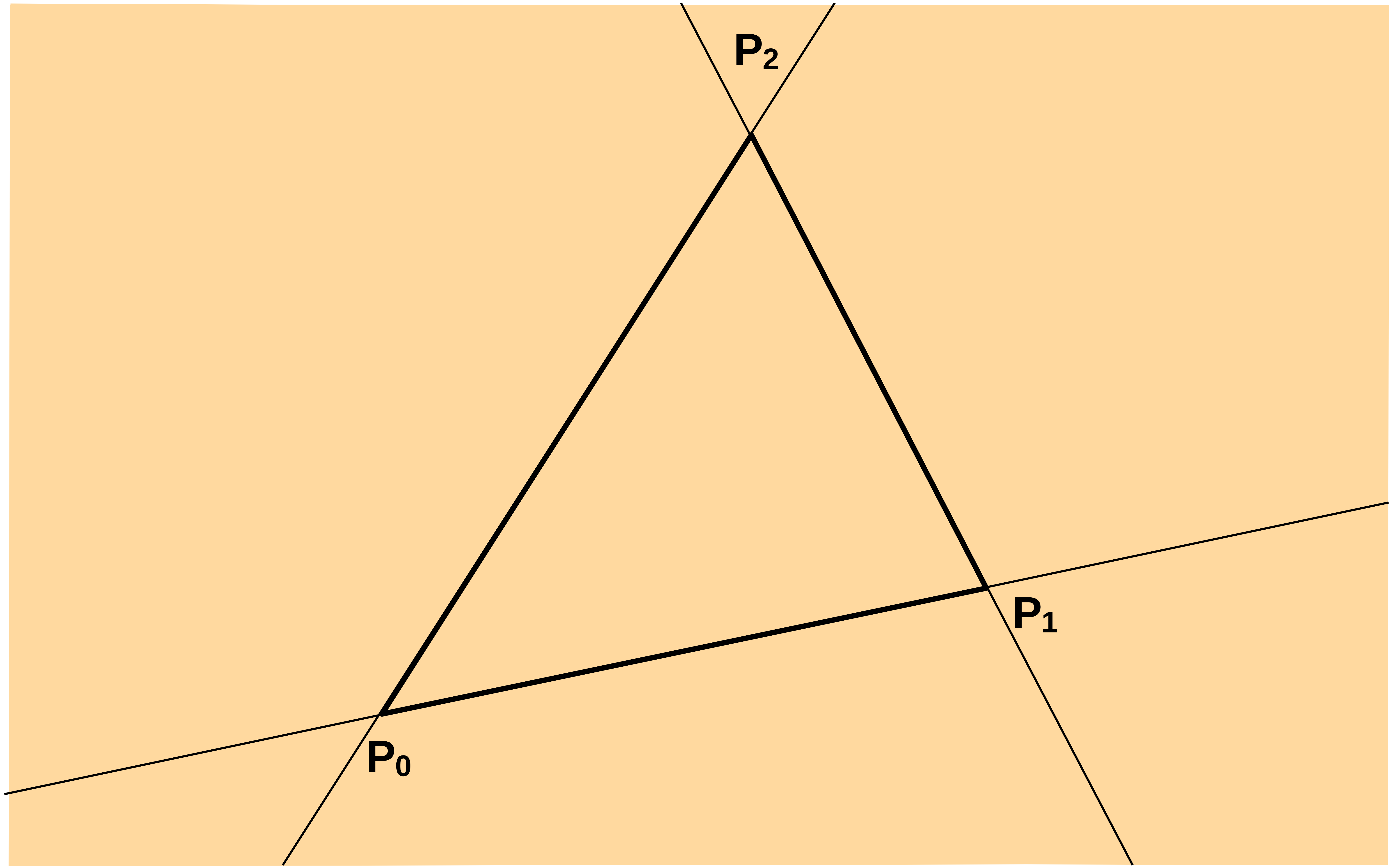
  `f(x,y) = inside(tri,x,y)`

```
for (int x = 0; x < xmax; x++)
  for (int y = 0; y < ymax; y++)
    image[x][y] = f(x + 0.5, y + 0.5);
```

Evaluating `inside(tri,x,y)`



Triangle = intersection of three half planes

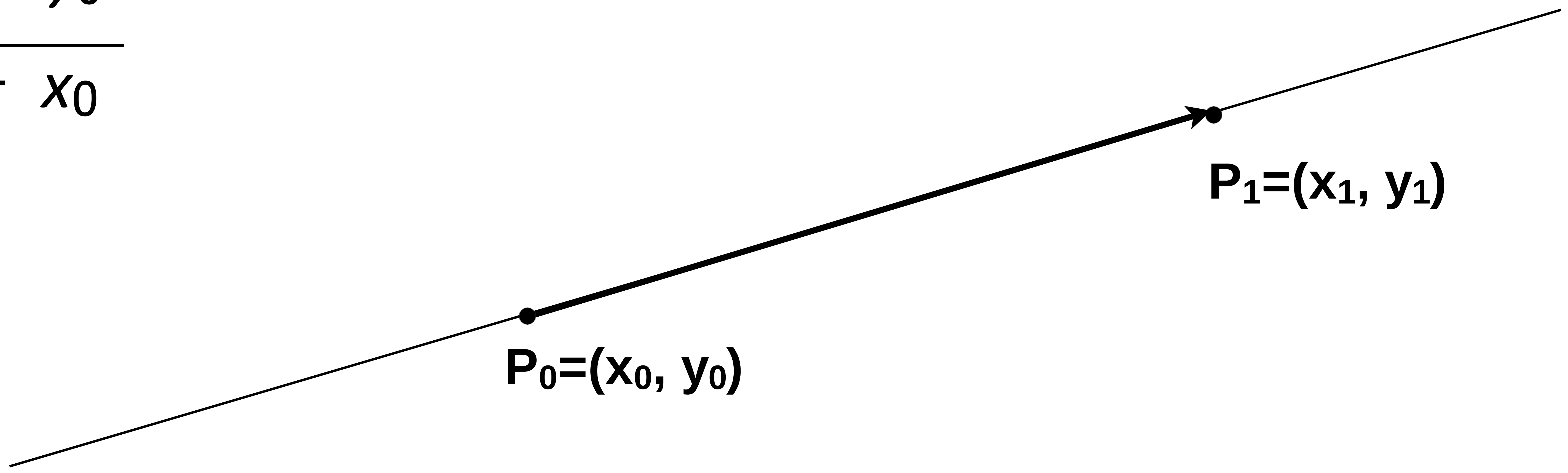


Point-slope form of a line

(You might have seen this in high school)

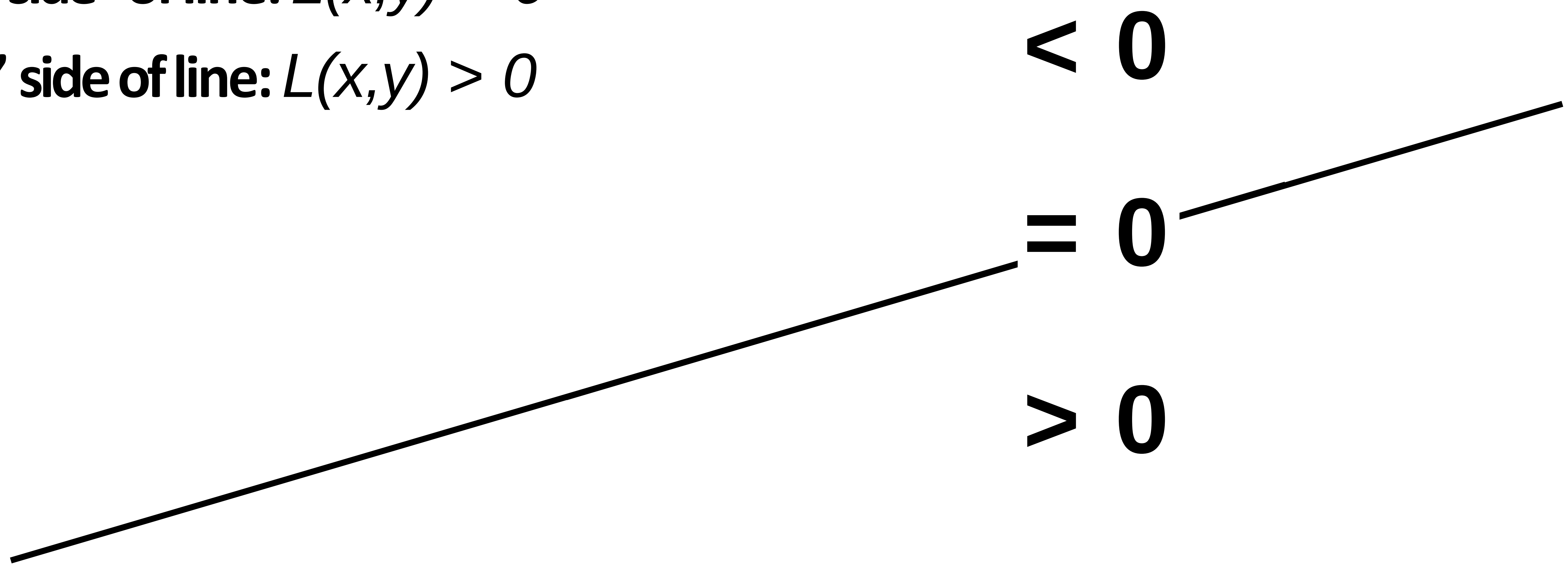
$$y - y_0 = m(x - x_0)$$

$$m = \frac{y_1 - y_0}{x_1 - x_0}$$

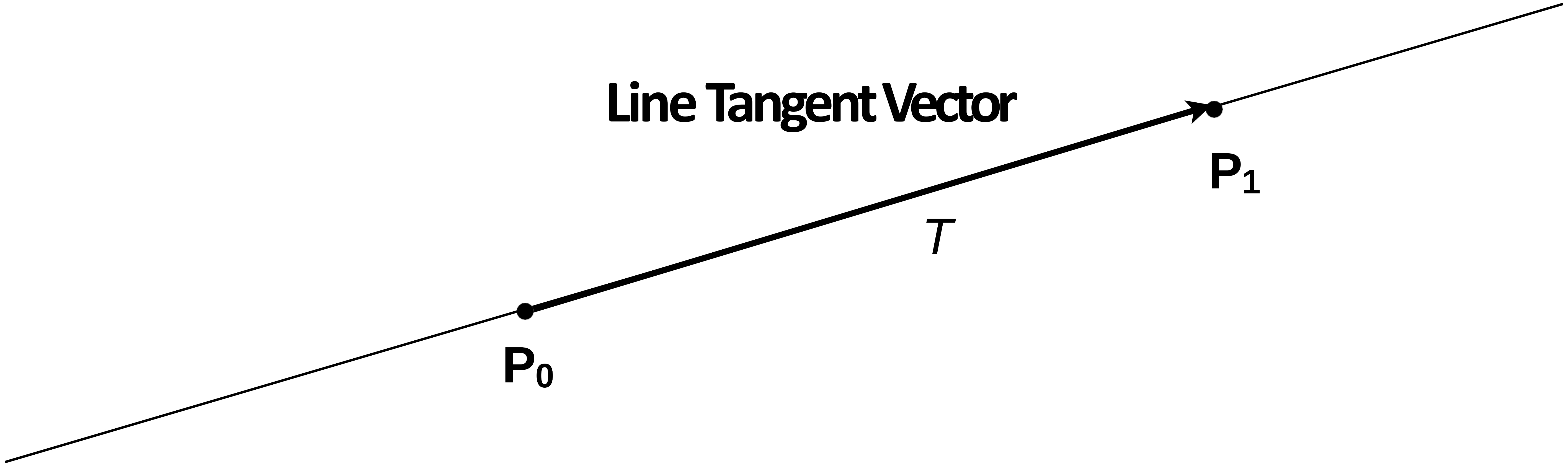


Each line defines two half-planes

- **Implicit line equation**
  - $L(x,y) = Ax + By + C$
  - **On the line:** $L(x,y) = 0$
  - **"Negative side" of line:** $L(x,y) < 0$
  - **"Positive" side of line:** $L(x,y) > 0$

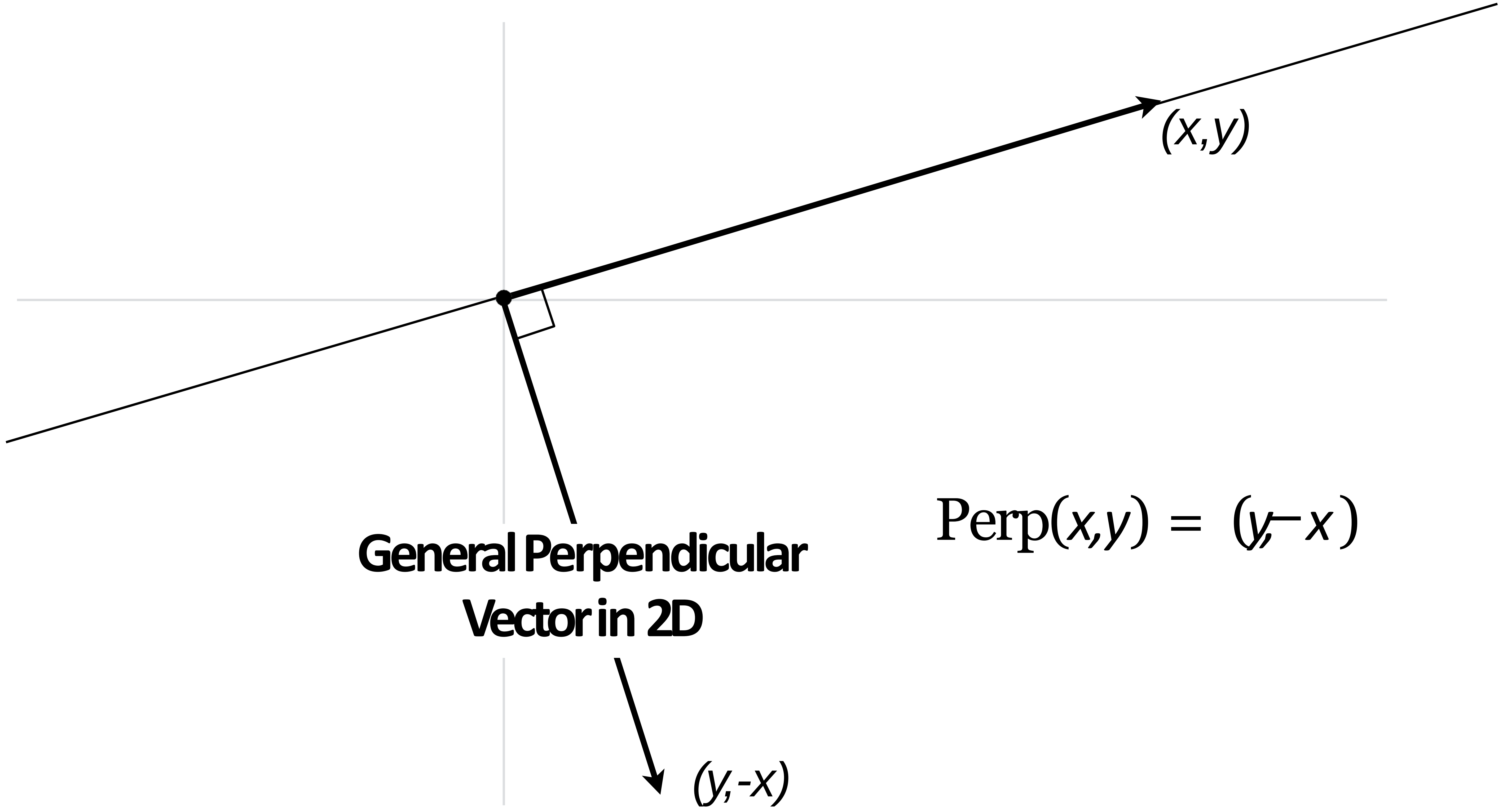


Line equation derivation



$$T = P_1 - P_0 = (x_1 - x_0, y_1 - y_0)$$

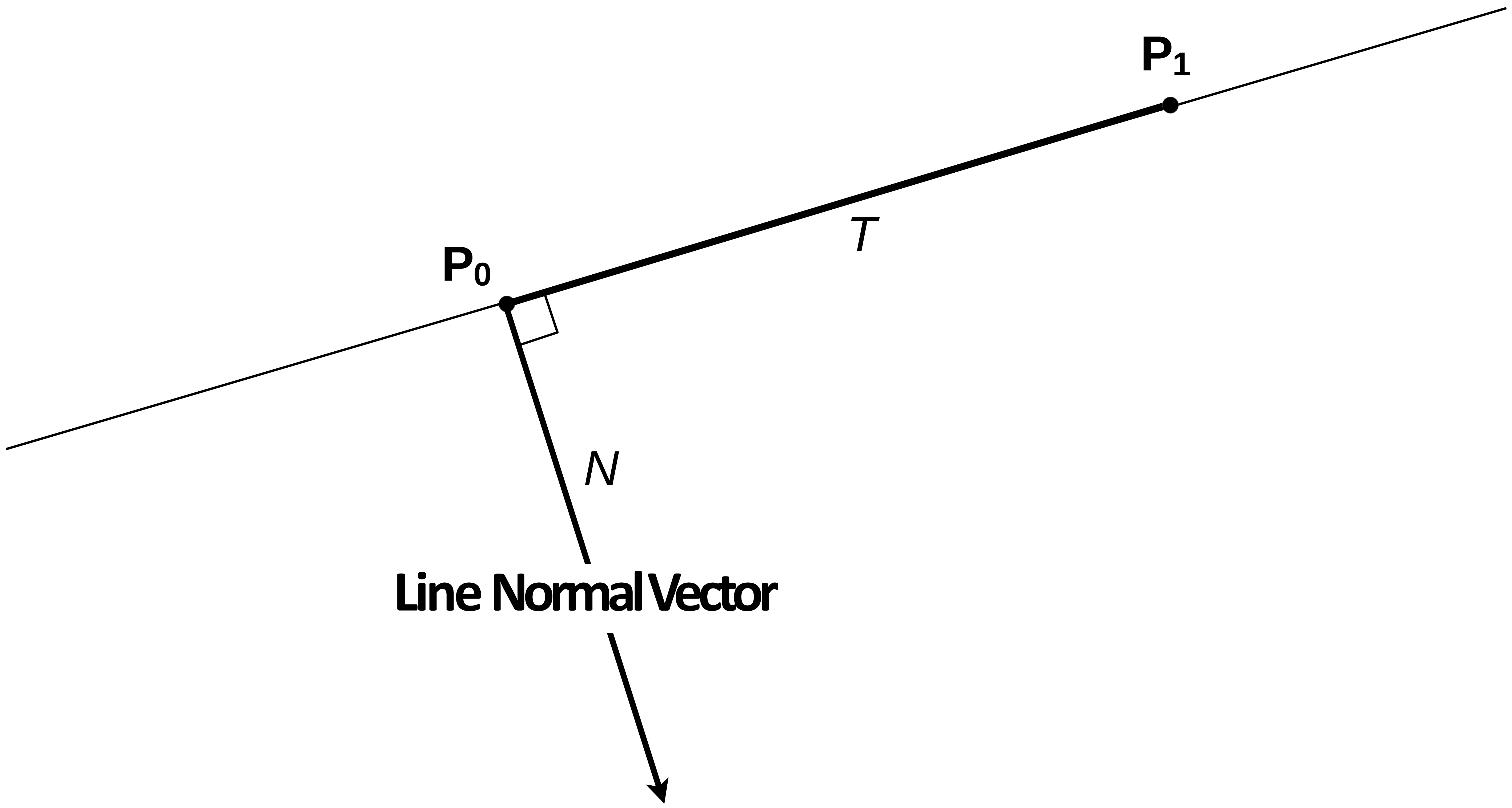
Line equation derivation



$$\mathrm{Perp}(x,y) = (y, -x)$$

Line equation derivation

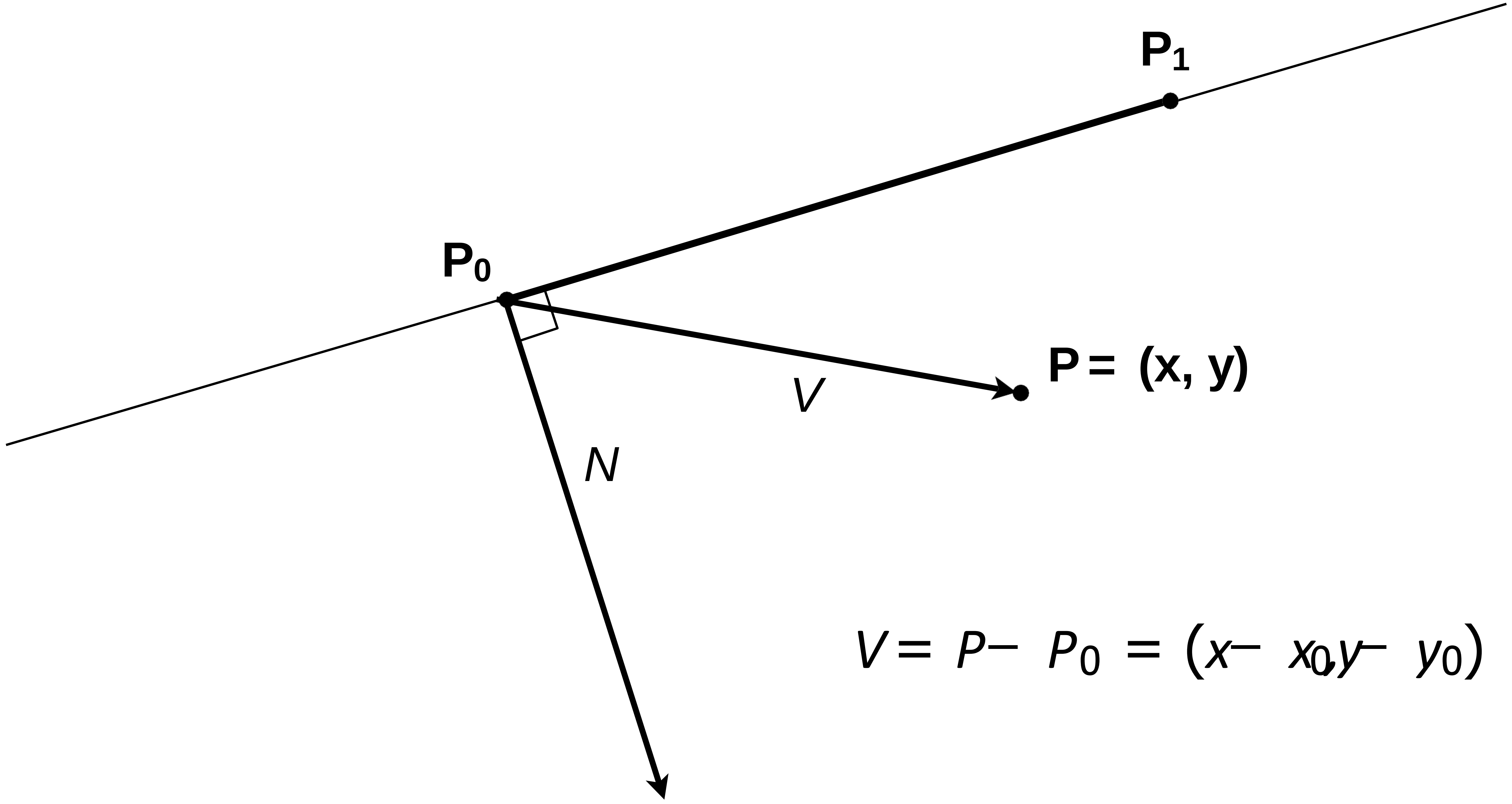
$$N = \mathrm{Perp}(T) = (y_1 - y_0, -(x_1 - x_0))$$



Line equation derivation

Now consider a point P=(x,y).

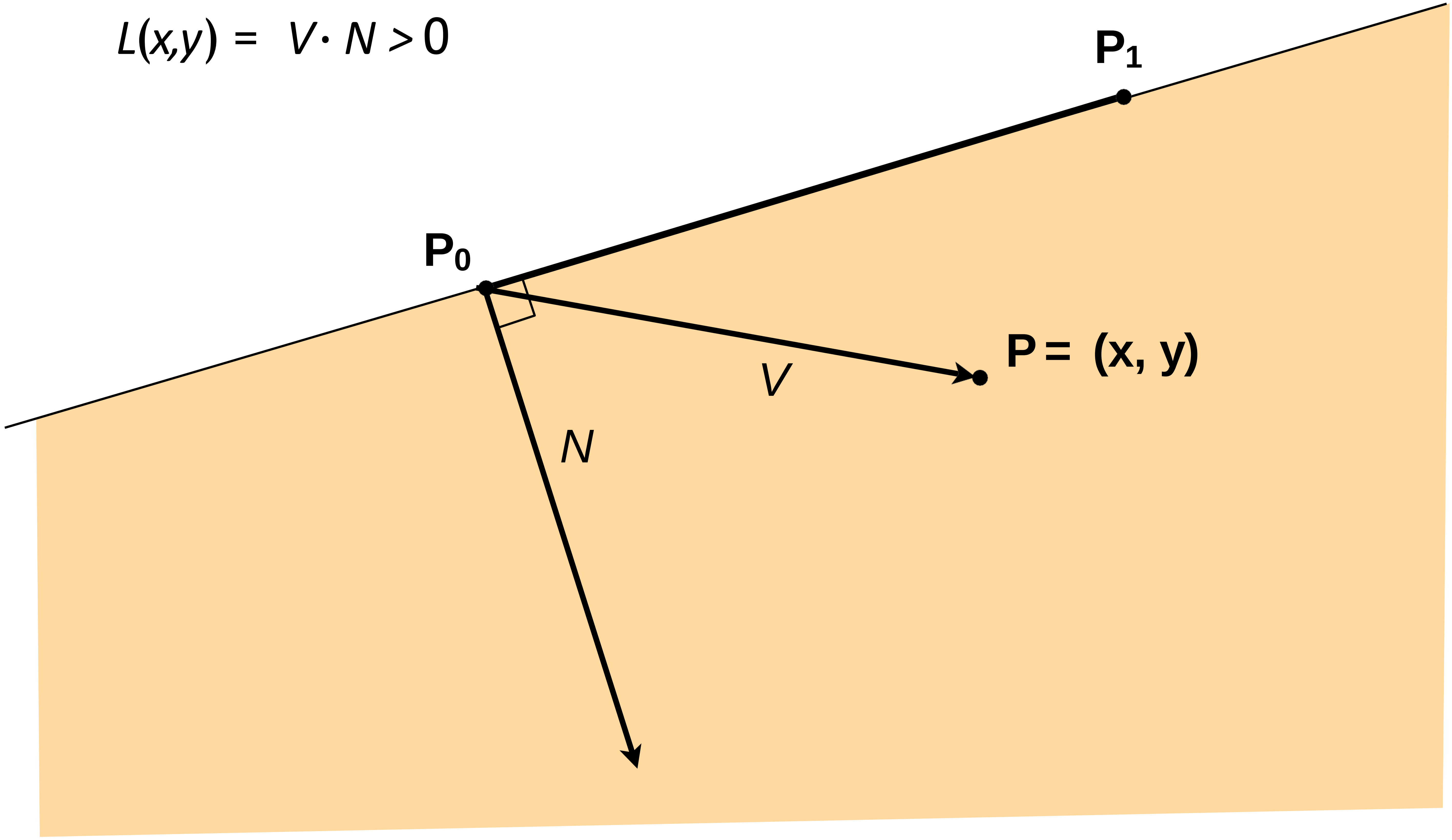
Which side of the line is it on?



$$V = P - P_0 = (x - x_0, y - y_0)$$

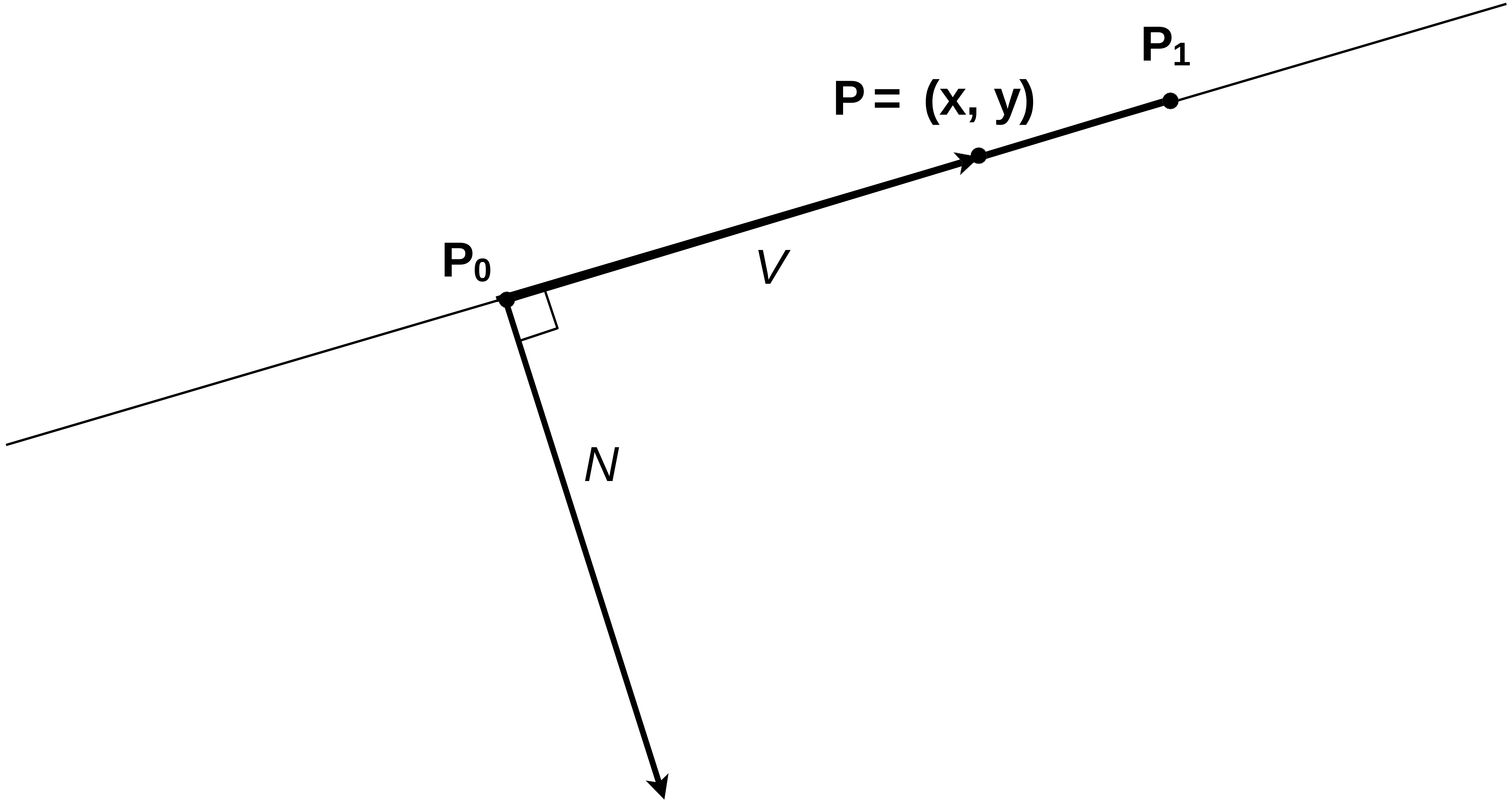
Line equation tests

$L(x,y) = V \cdot N > 0$

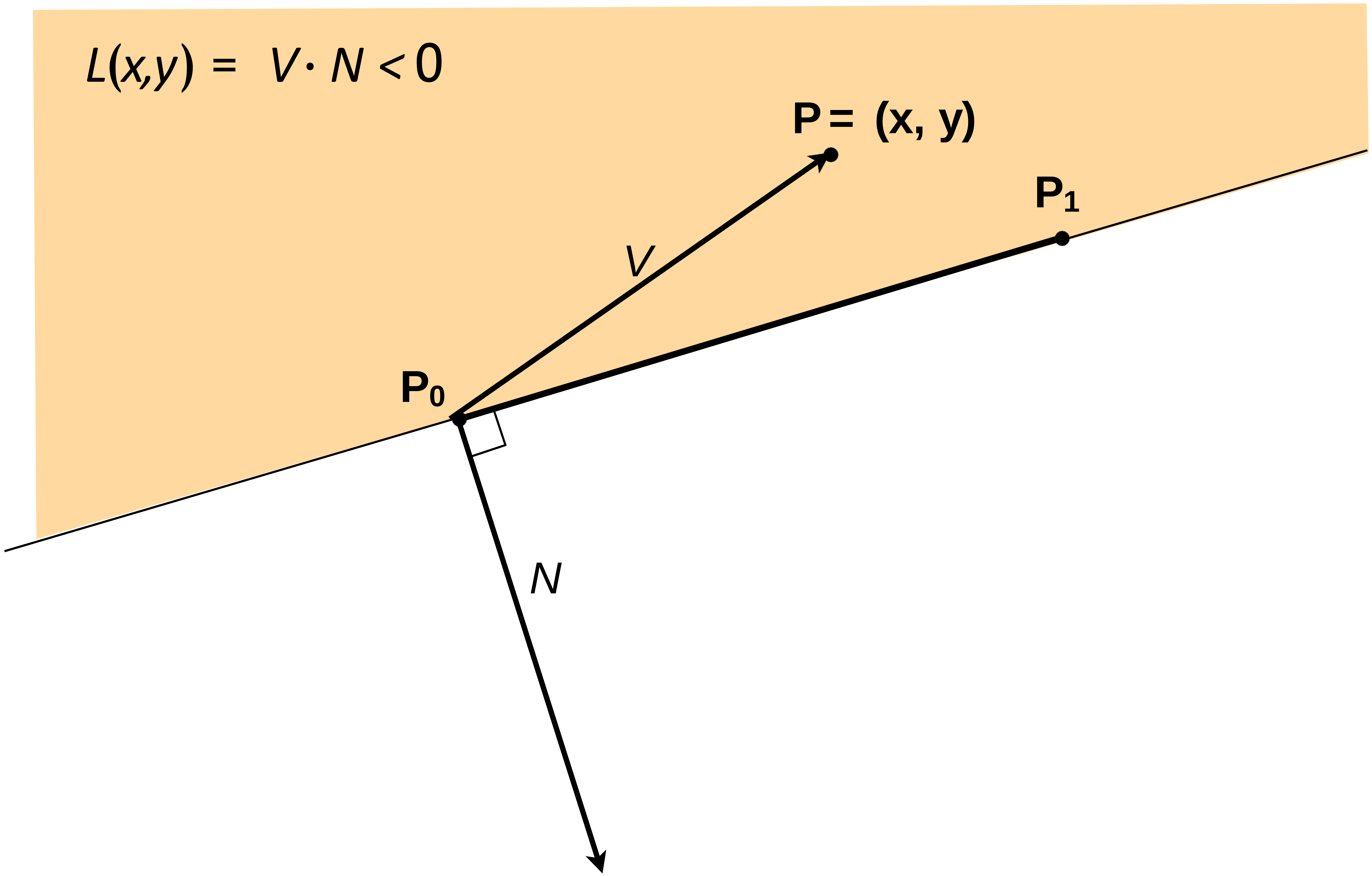


Line equation tests

$$L(x,y) = V \cdot N = 0$$

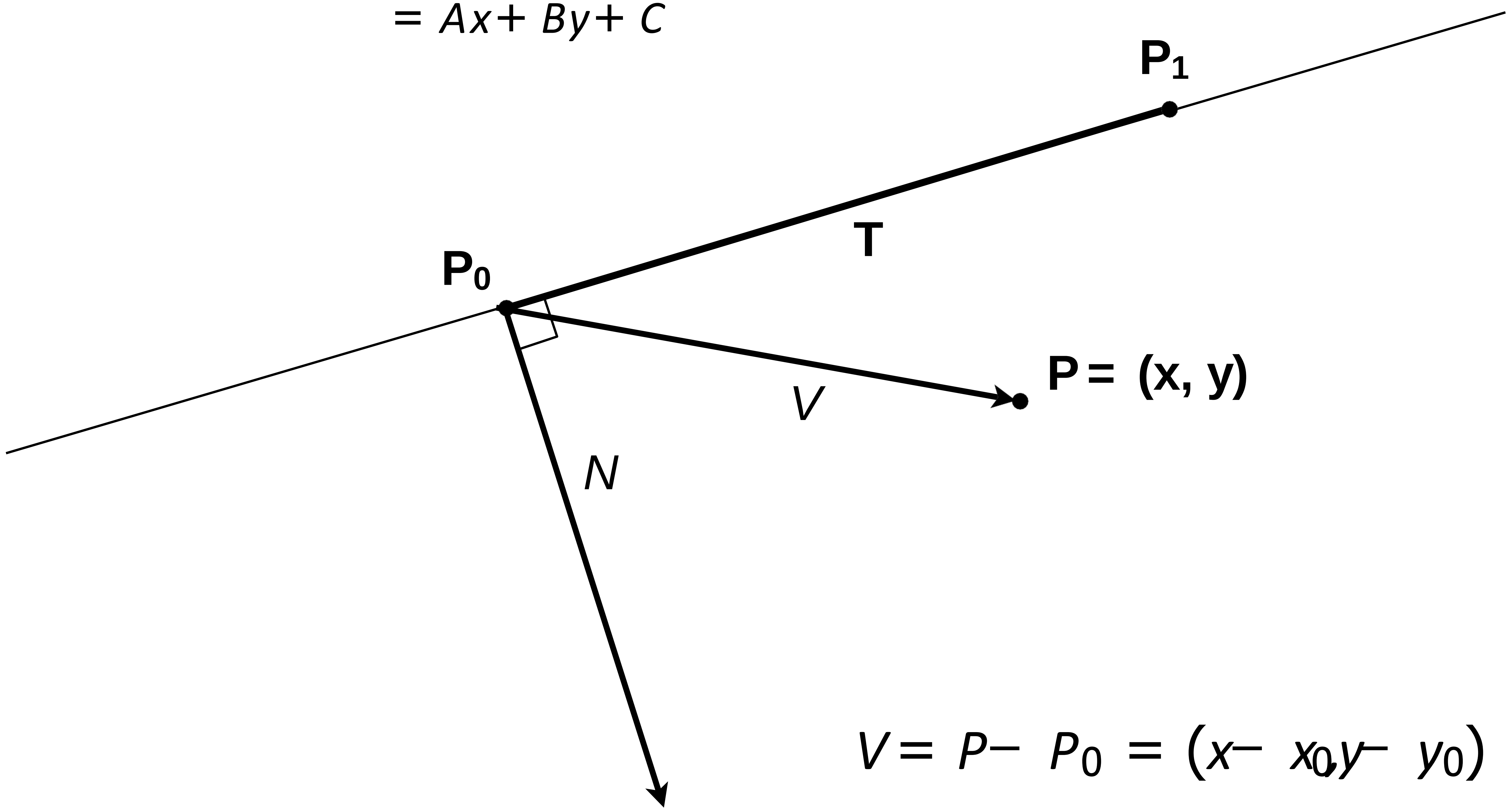


Line equation tests



Line equation derivation

$$L(x,y) = V \cdot N = -(y - y_0)(x_1 - x_0) + (x - x_0)(y_1 - y_0)$$
$$= (y_1 - y_0)x - (x_1 - x_0)y + y_0(x_1 - x_0) - x_0(y_1 - y_0)$$
$$= Ax + By + C$$



$$V = P - P_0 = (x - x_0, y - y_0)$$

$$N = \mathrm{Perp}(T) = (y_1 - y_0, -(x_1 - x_0))$$

Point-in-triangle test

$$P_i = (X_i, Y_i)$$

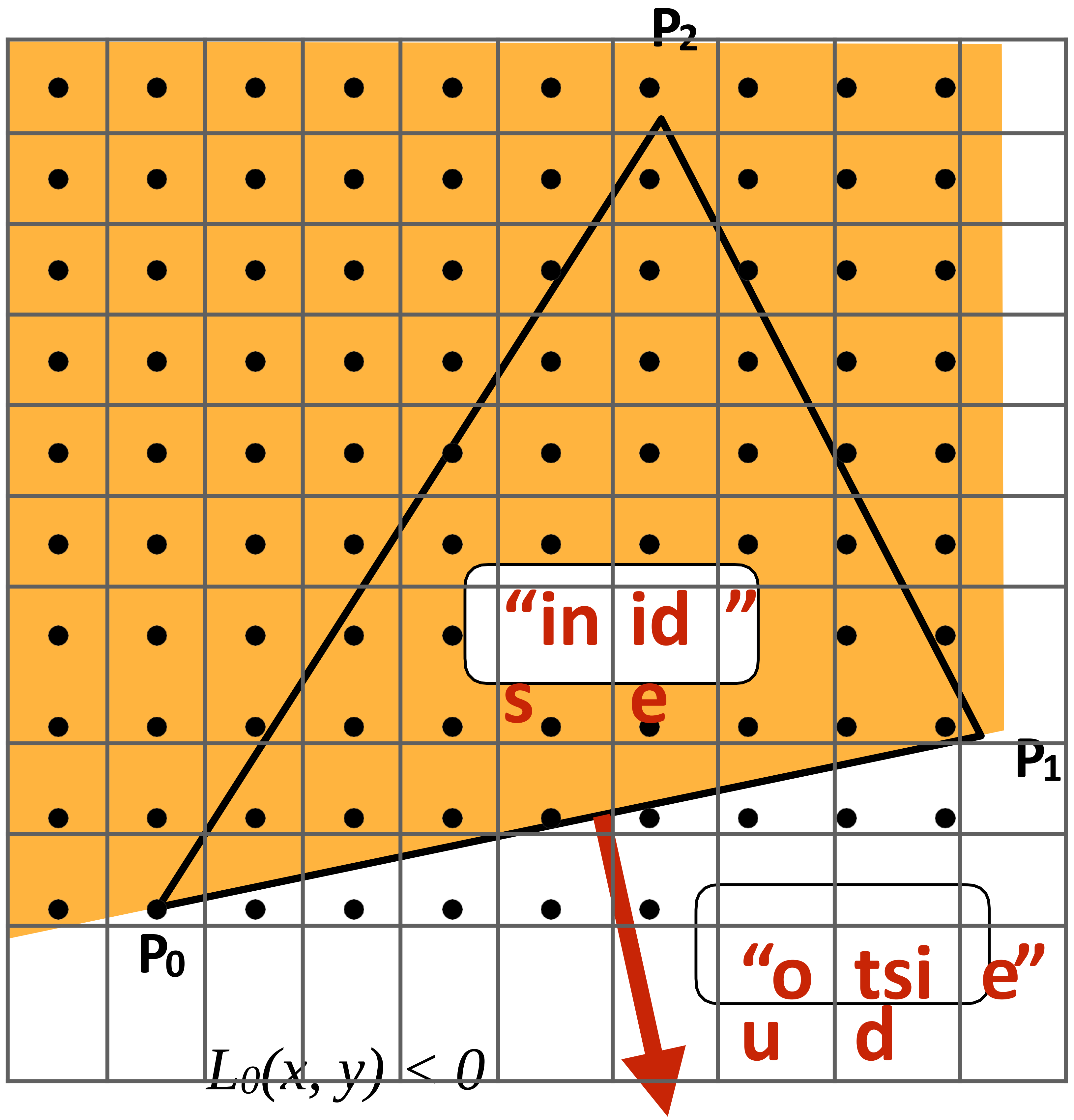
$$A_i = dY_i = Y_{i+1} - Y_i$$
$$B_i = -dX_i = X_i - X_{i+1}$$
$$C_i = Y_i\,(X_{i+1} - X_i) - X_i\,(Y_{i+1} - Y_i)$$

$$L_i\,(x, y) = A_i\,x + B_i\,y + C_i$$

$L_i\,(x, y) = 0$: point on edge

> 0 : outside edge

< 0 : inside edge



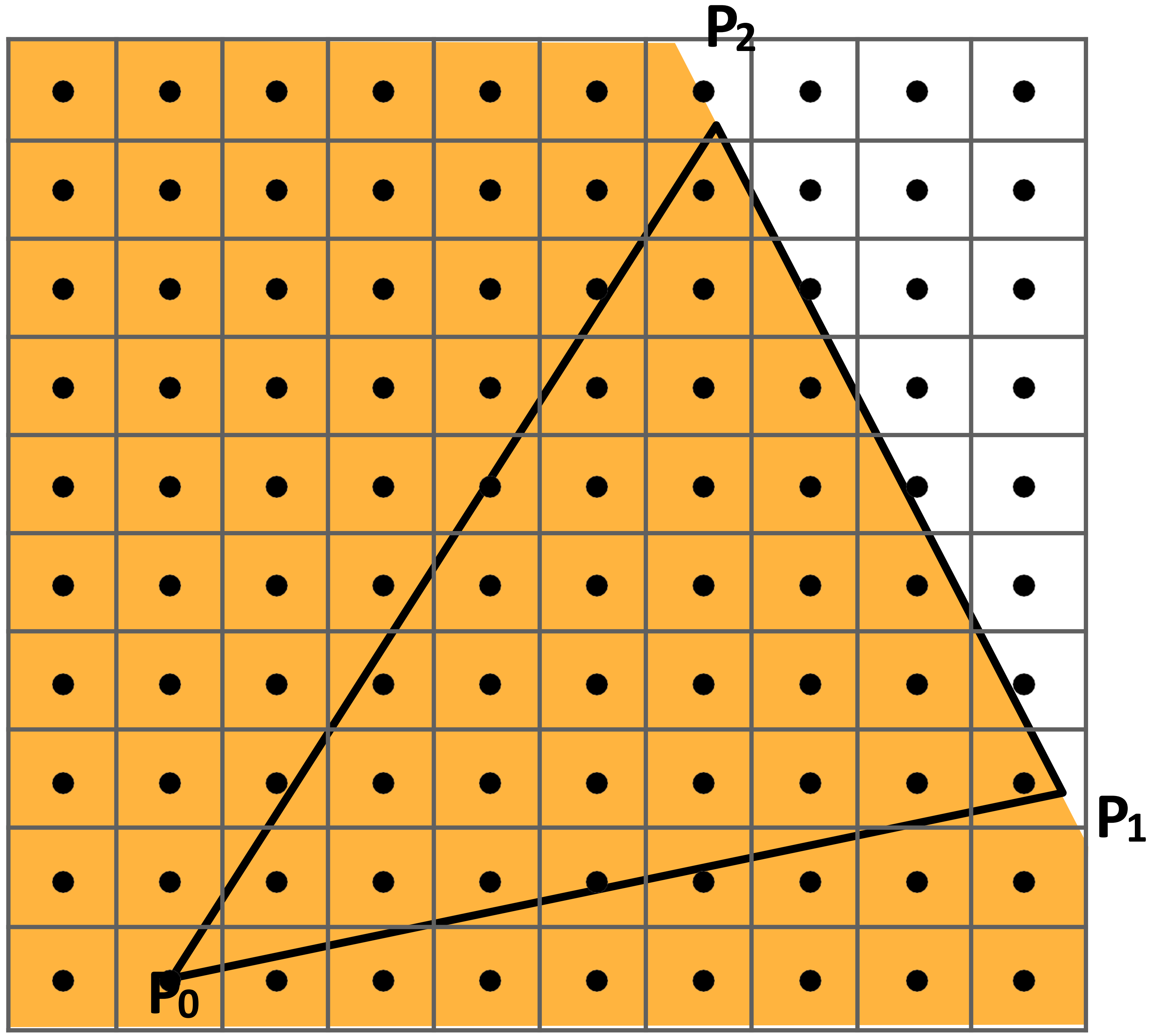
Point-in-triangle test

$P_i = (X_i, Y_i)$

$A_i = dY_i = Y_{i+1} - Y_i$
$B_i = -dX_i = X_i - X_{i+1}$
$C_i = Y_i\,(X_{i+1} - X_i) - X_i\,(Y_{i+1} - Y_i)$

$L_i\,(x, y) = A_i\,x + B_i\,y + C_i$

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



$L_1(x, y) < 0$

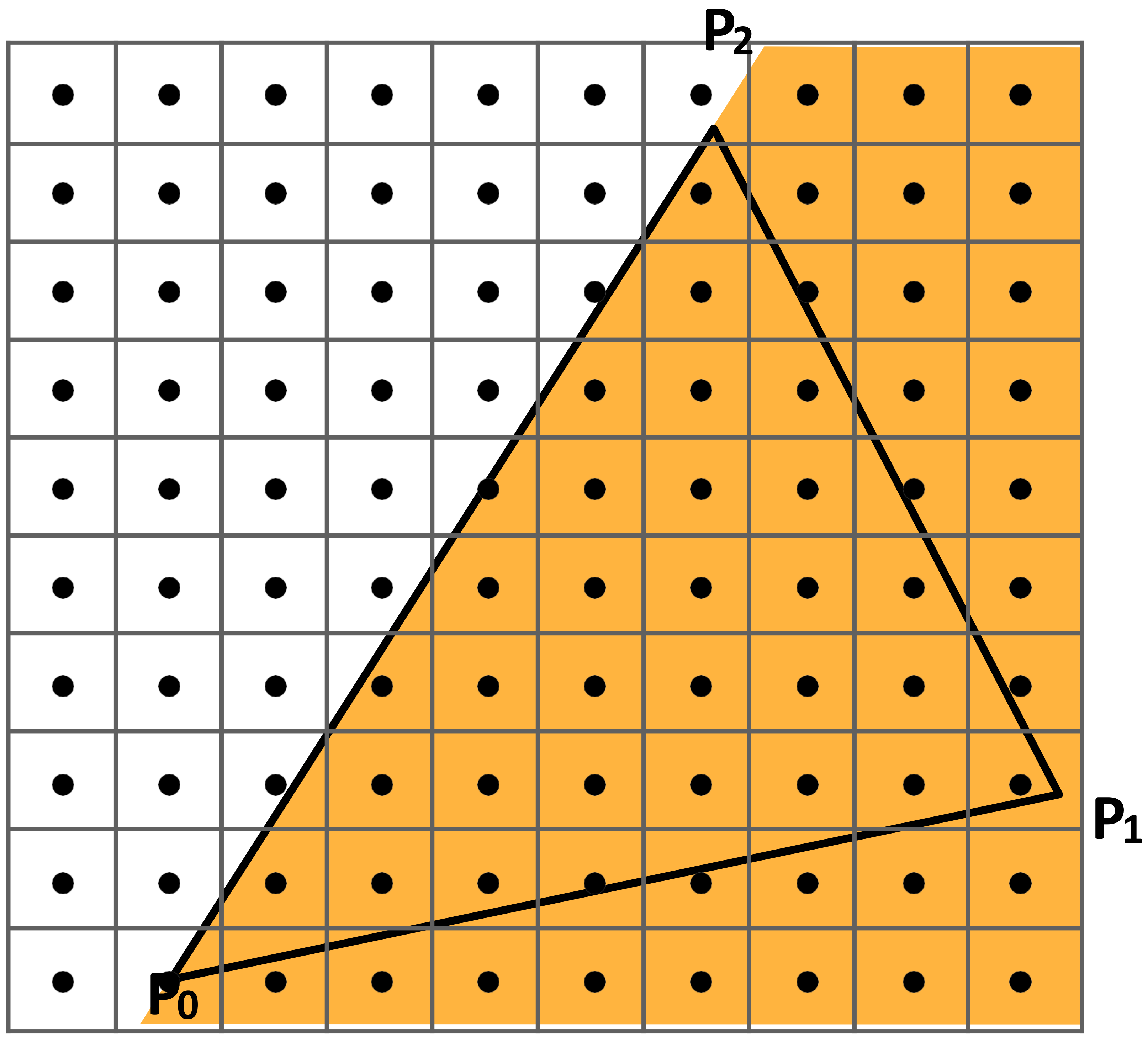
Point-in-triangle test

$P_i = (X_i, Y_i)$

$A_i = dY_i = Y_{i+1} - Y_i$
$B_i = -dX_i = X_i - X_{i+1}$
$C_i = Y_i (X_{i+1} - X_i) - X_i (Y_{i+1} - Y_i)$

$L_i (x, y) = A_i x + B_i y + C_i$

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



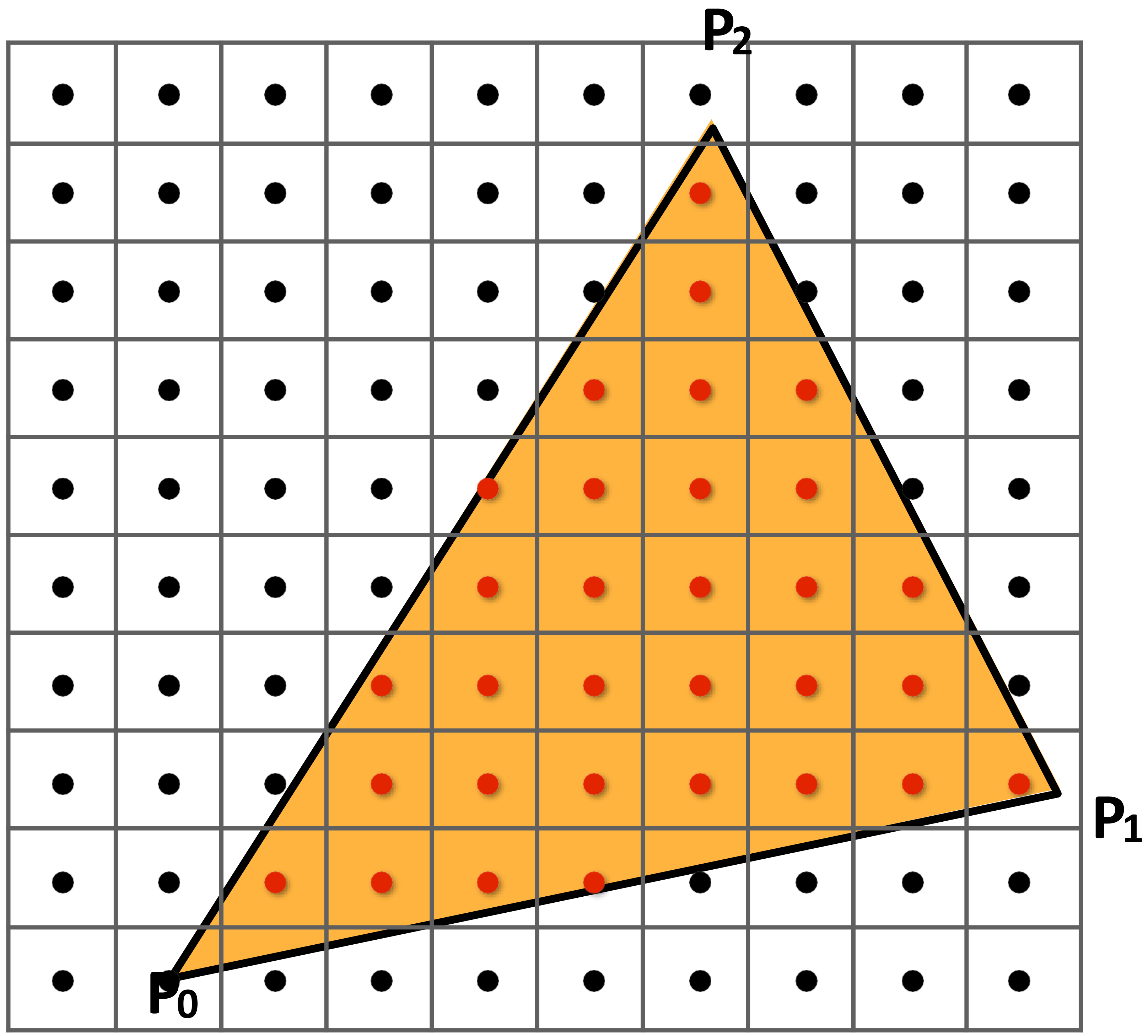
$L_2(x, y) < 0$

Point-in-triangle test

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

$$
\begin{aligned}
&inside(sx, sy) = \\
&\quad L_0(sx, sy) < 0 \;\&\& \\
&\quad L_1(sx, sy) < 0 \;\&\& \\
&\quad L_2(sx, sy) < 0
\end{aligned}
$$

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



Sample points inside triangle are highlighted red.