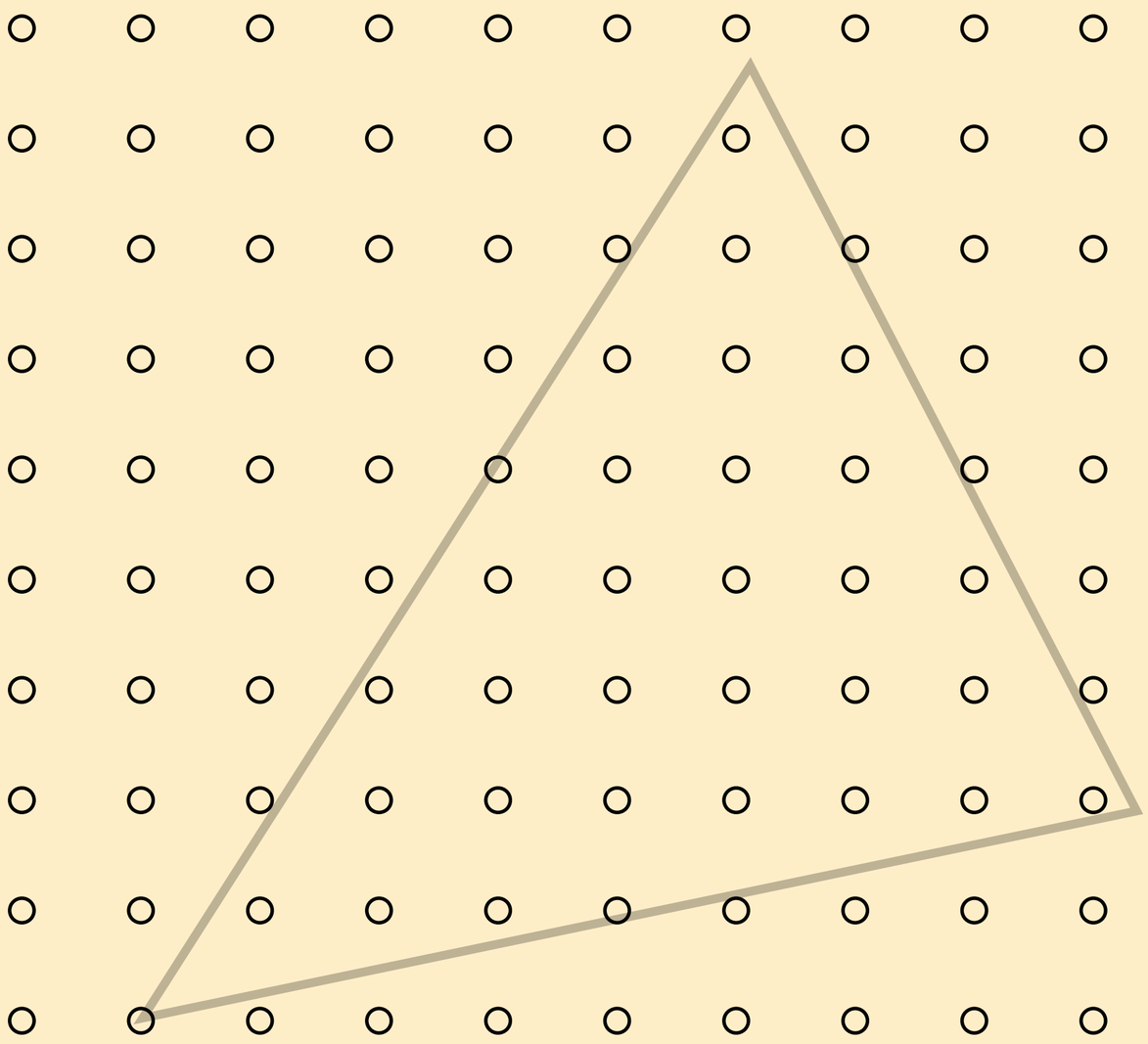
Discussion: What Value Should a Pixel Have?



Think about:

- Ideas for “higher quality” pixel formula?
- What are all the relevant factors?
- What’s right/wrong about point sampling?
- Why do jaggies look “wrong”?

We will discuss this later today!

Lecture 2:

Rasterization and Drawing Triangles

Computer Graphics and Imaging
UC Berkeley CS184/284

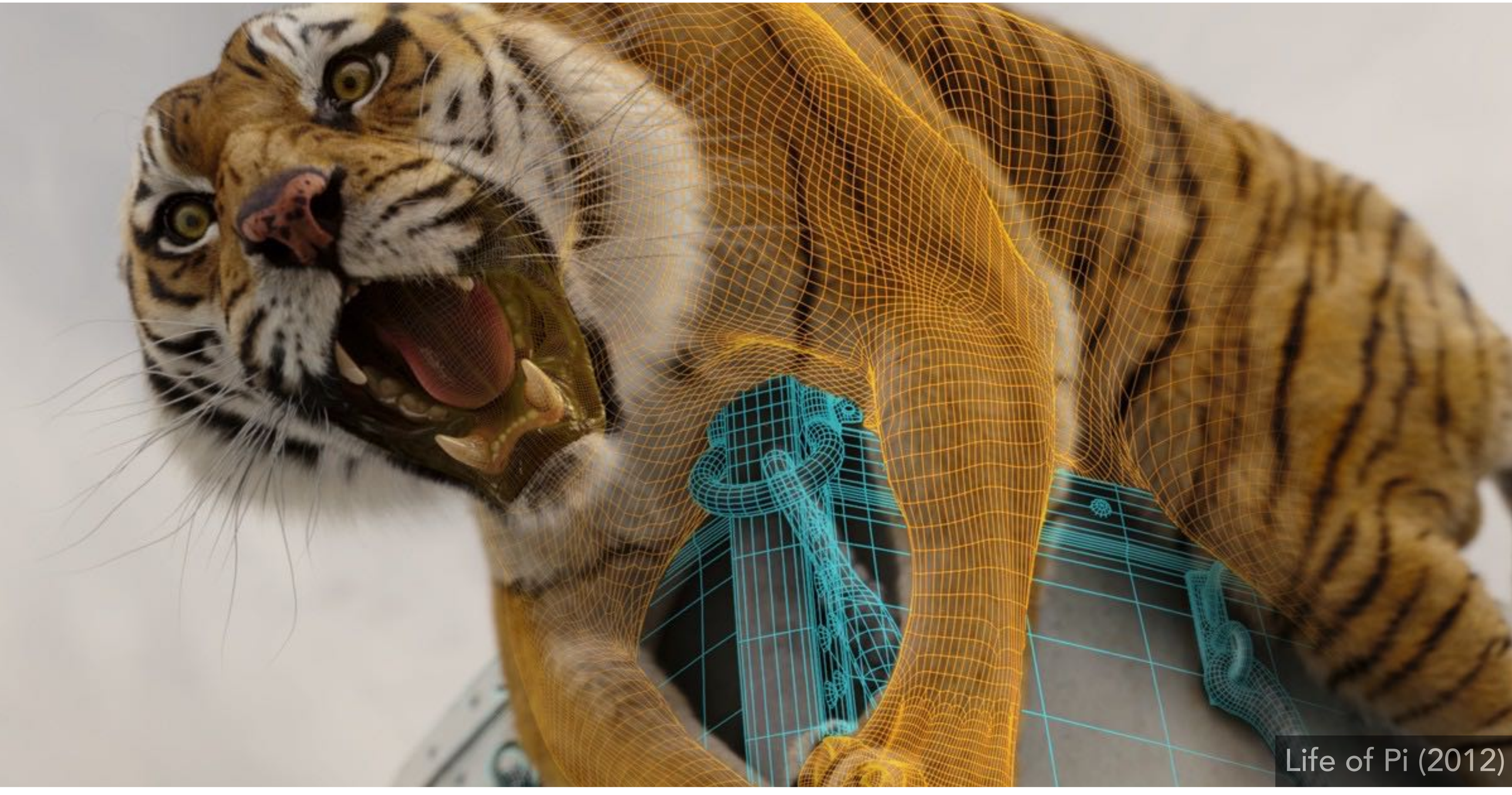
Announcements

- Gradescope Checkpoint clarification
  - You can re-submit as many times as you want until the deadline (Sunday 11:59 PM), but answers will not be shown until after deadline
  - Consult anything you want, but not other people (to ensure your *personal* understanding)
- Discord — information to join posted on Piazza
- Discussions start today
  - Try having IDE, etc. set up before discussion, but TAs are available to debug (also in OH)

Today's Agenda

- Review of lecture content (with a focus on connecting to Project 1)
- Discussion: What value should a pixel have?
- Taking any questions you may have from the video — please ask!
    - Also encouraged to make any comments ("I thought _____ was cool/unexpected/etc", "_____ also shows up in _____"...)
- If still time: demo project

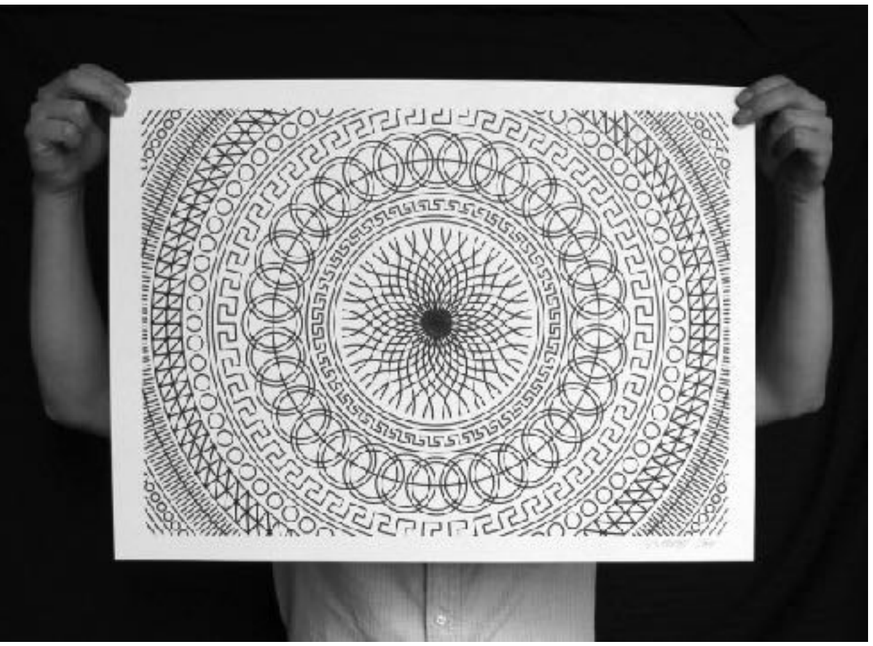
Today: Rasterizing Triangles into Pixels

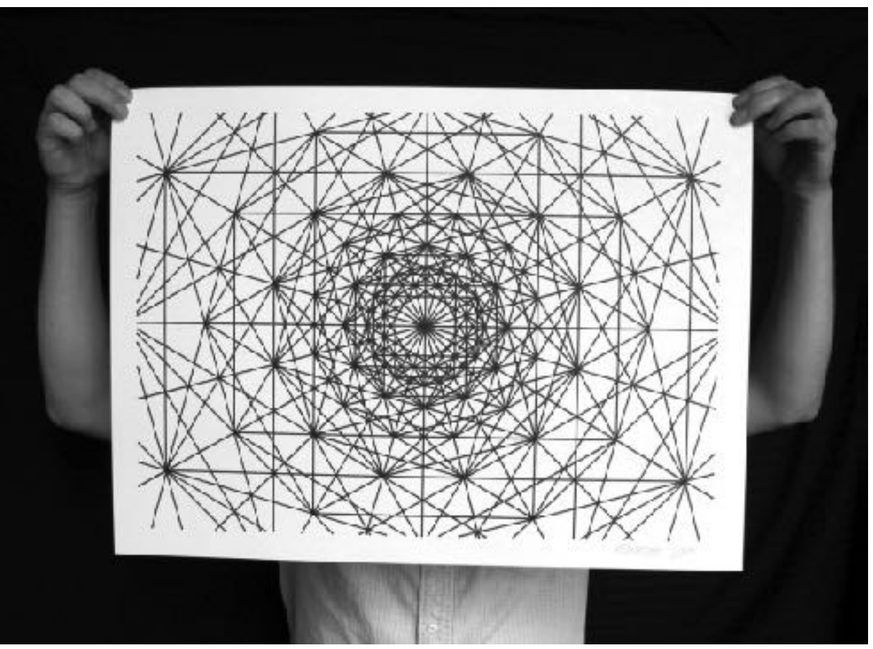


CS184/284A

Drawing Machines

CNC Sharpie Drawing Machine



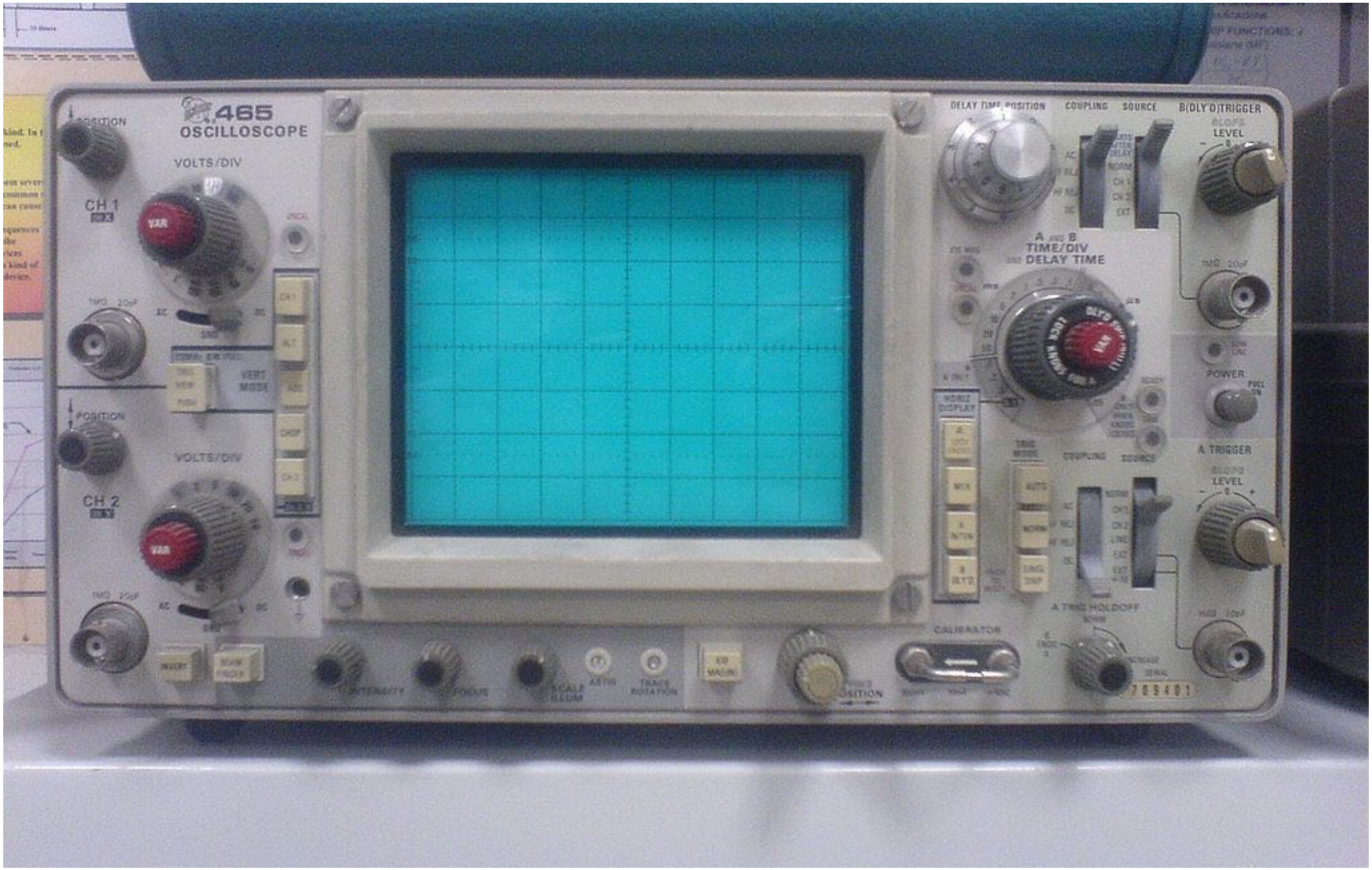


Aaron Panone with Matt W. Moore

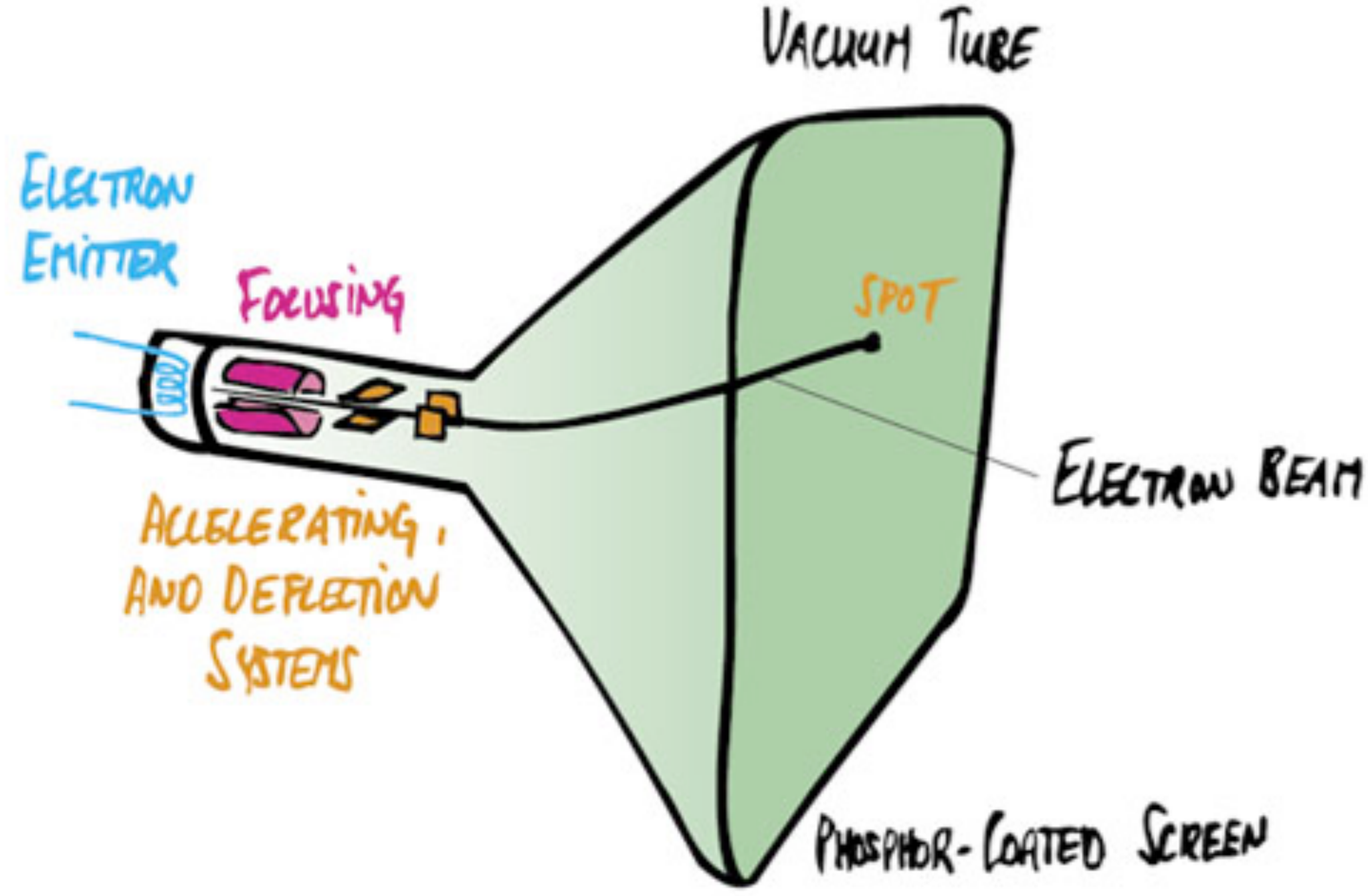
http://44rn.com/projects/numerically-controlled-poster-series-with-matt-w-moore/

Laser Cutters

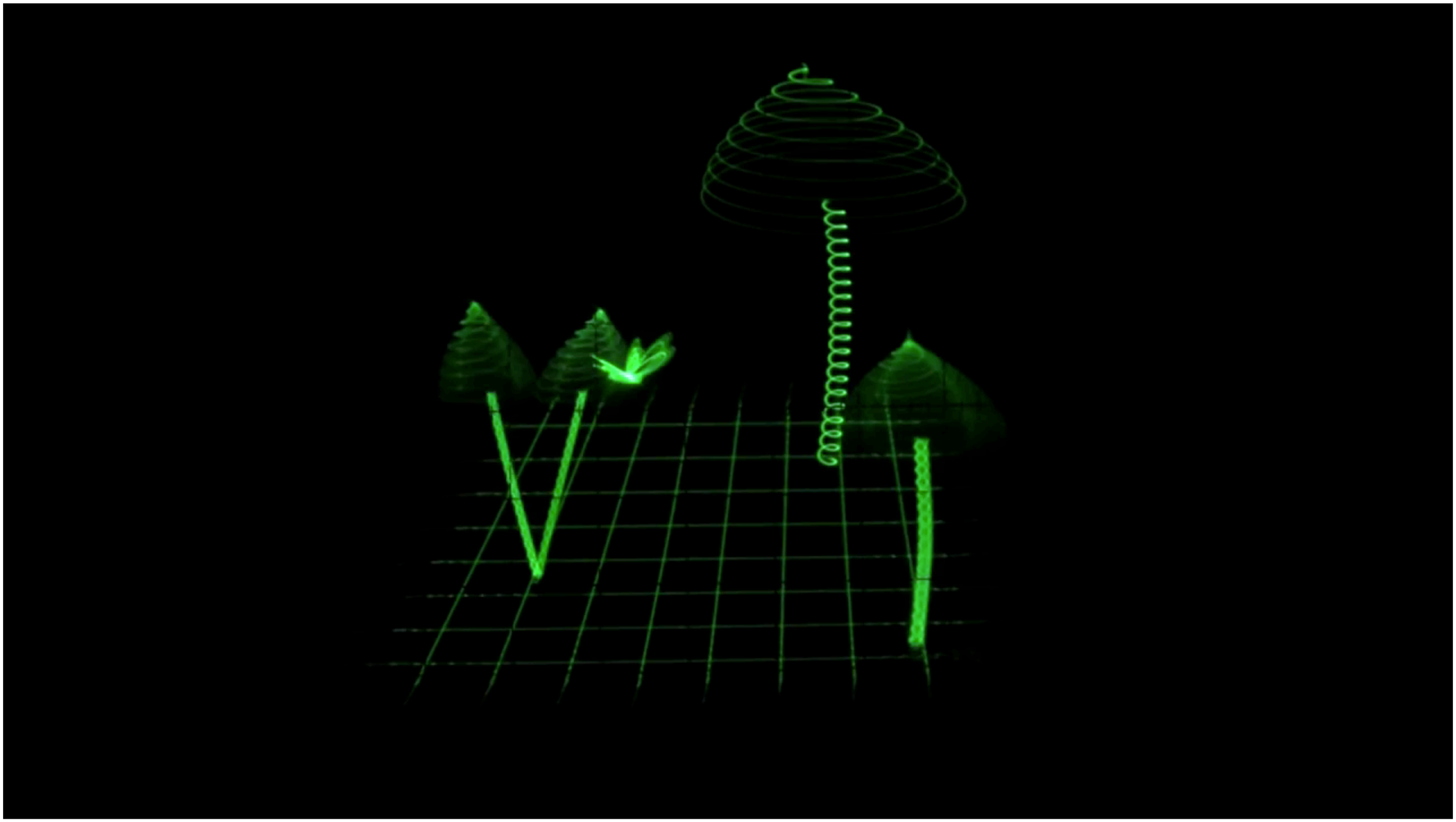
Oscilloscope



Cathode Ray Tube



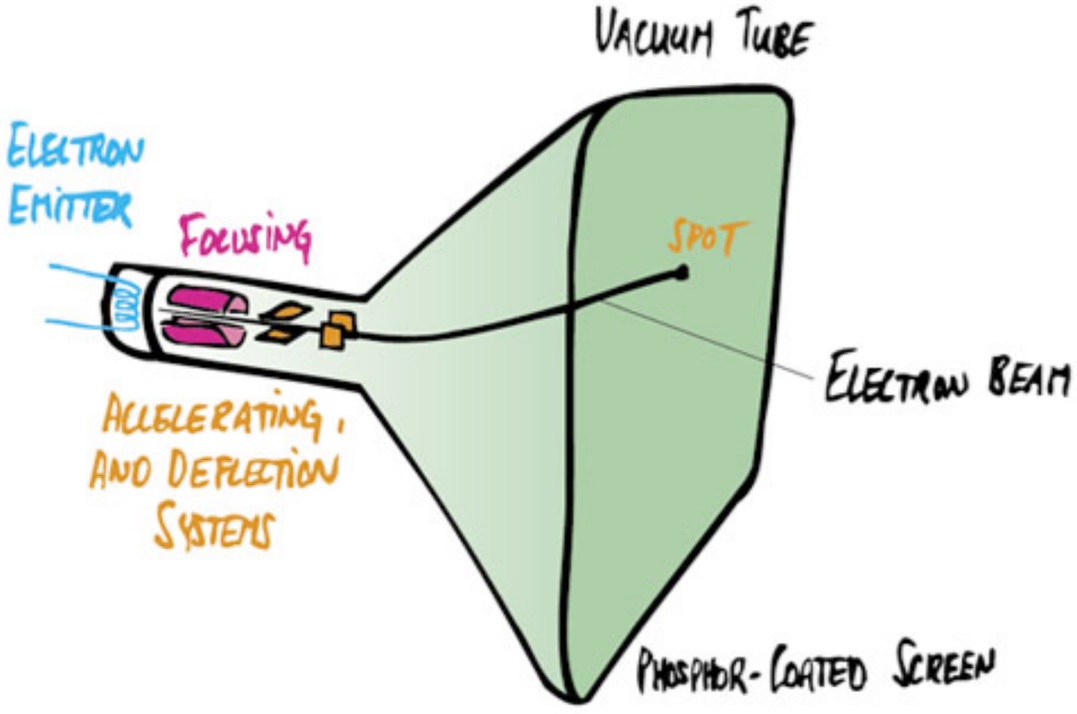
Oscilloscope Art



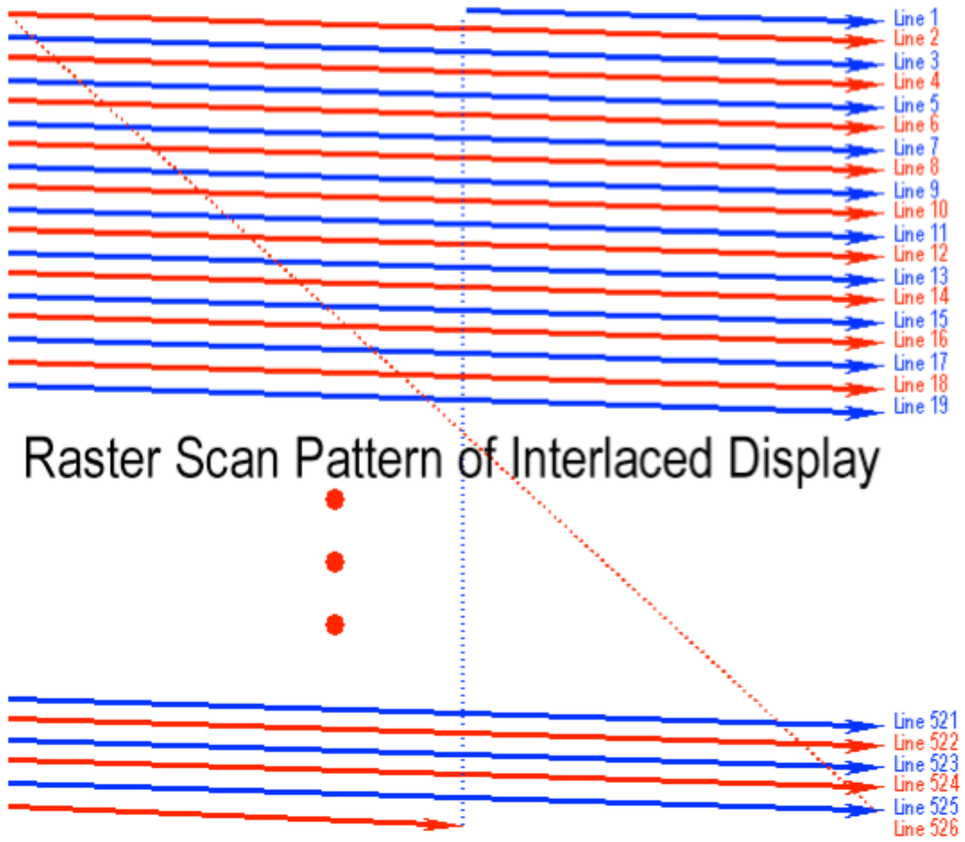
Jerobeam Fenderson

https://www.youtube.com/watch?v=rtR63-ecUNo

Television - Raster Display CRT



Cathode Ray Tube



Raster Scan
(modulate intensity)

Frame Buffer: Memory for a Raster Display

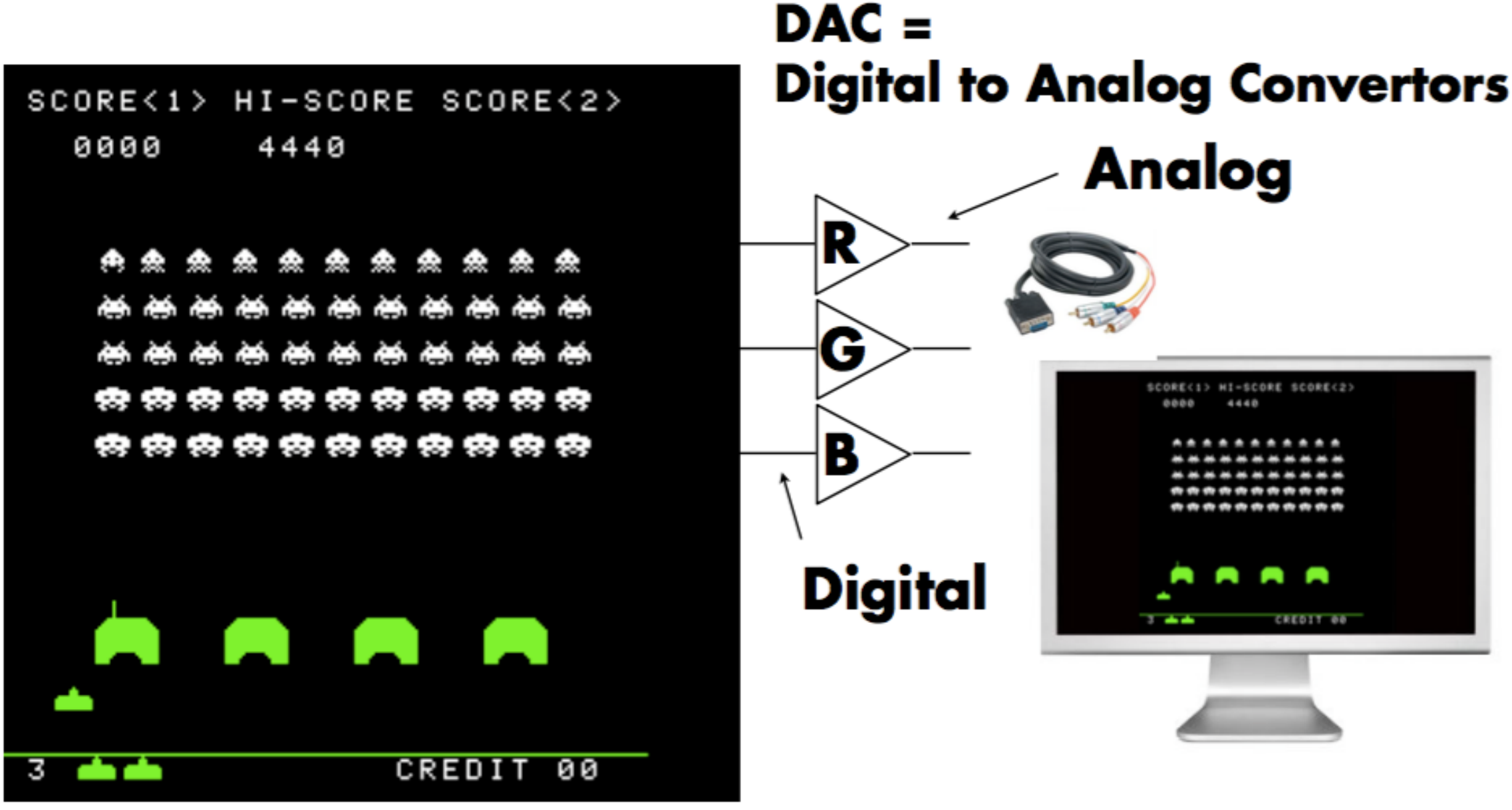


Image = 2D array of colors

Frame Buffer: Memory for a Raster Display

Note that there may be many other intermediate buffers!

Used to perform various calculations on data before writing to/resolving to/drawing to frame buffer.

Image = 2D array of colors —> An arbitrary 2D array of data could also be visualized as an image!

A Sampling of Different Raster Displays

Flat Panel Displays

Low-Res LCD Display

CS184/284A

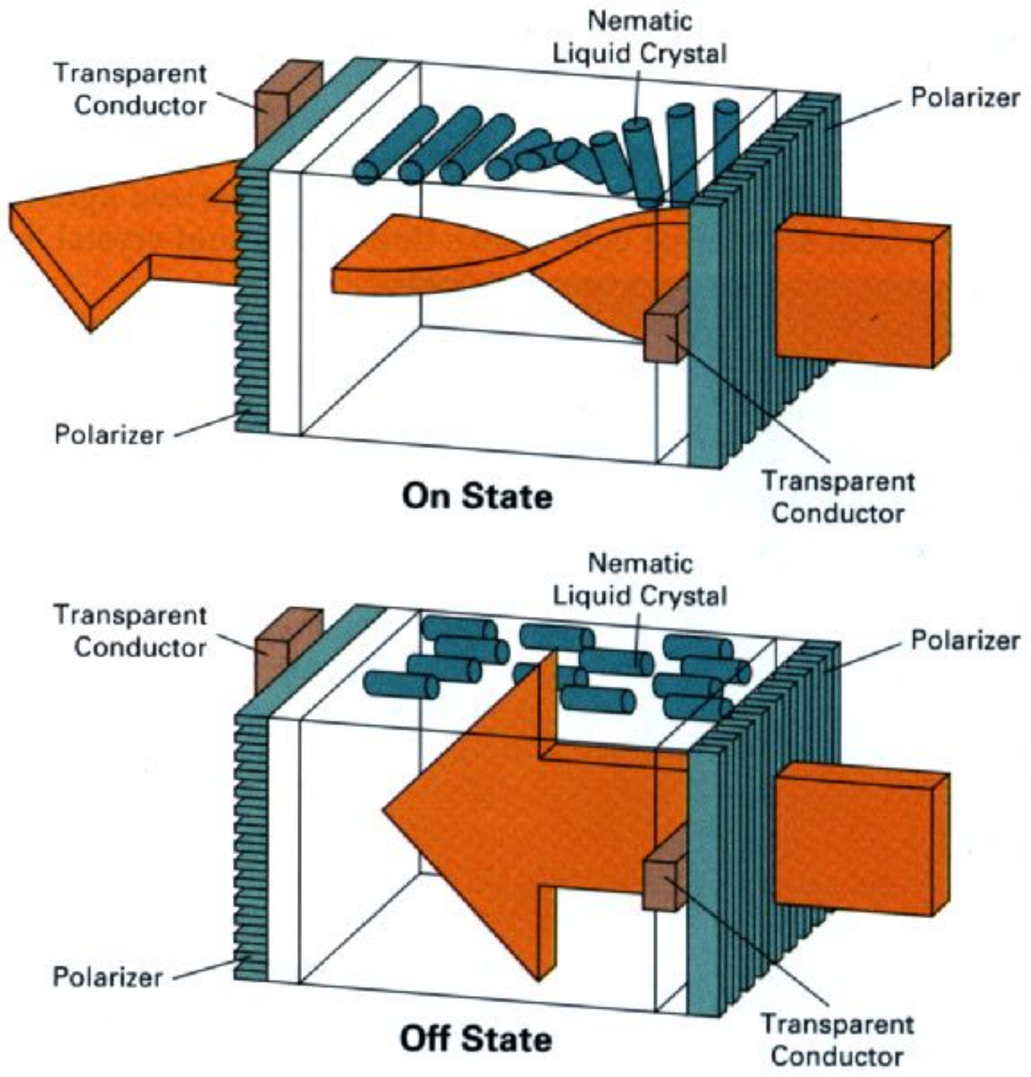
Color LCD, OLED, …

LCD (Liquid Crystal Display) Pixel

Principle: block or transmit light by twisting polarization

Illumination from backlight (e.g. fluorescent or LED)

Intermediate intensity levels by partial twist



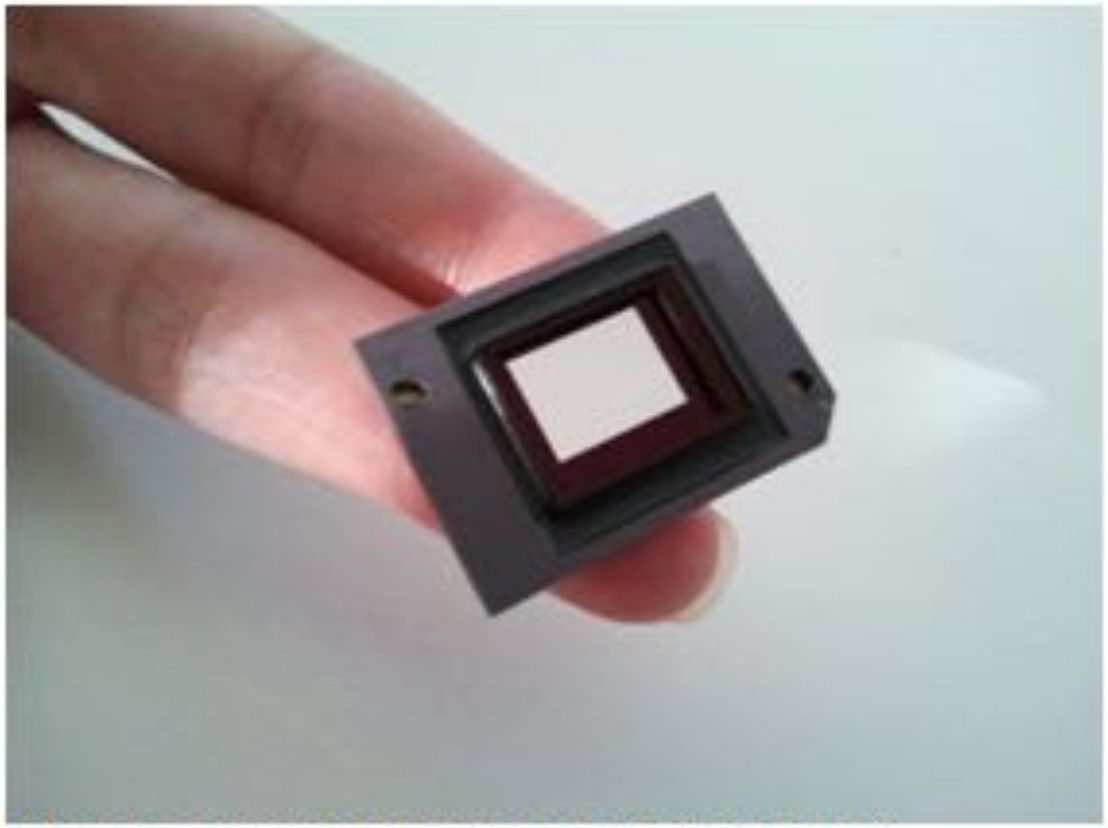
[H&B fig. 2-16]

LED Array Display

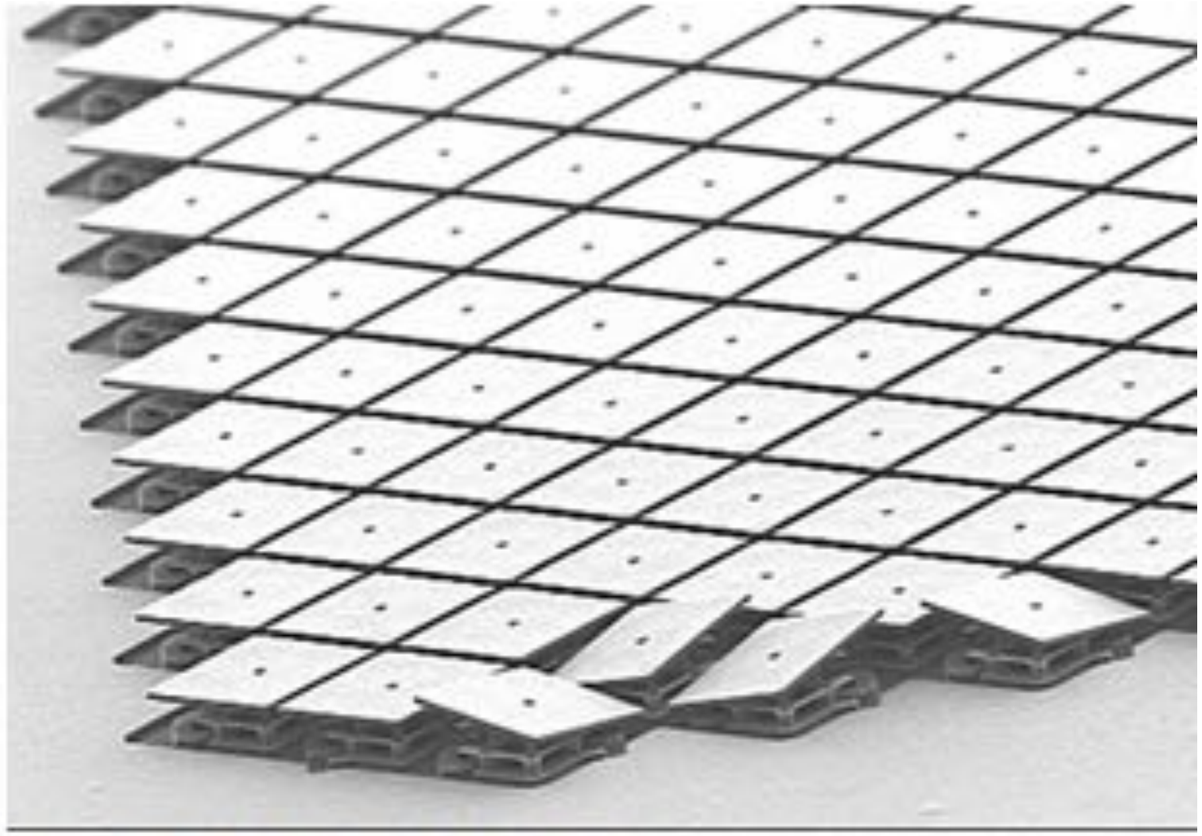
CS184/284A

Light emitting diode array

DMD Projection Display



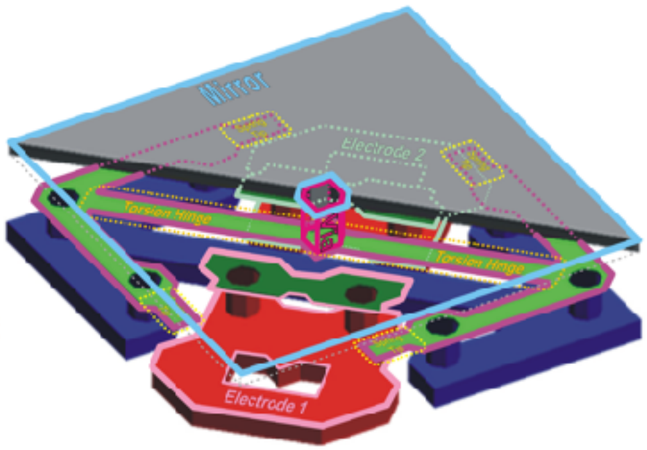
DIGITAL MICRO MIRROR DEVICE **(DMD)**
(**SLM** - Spatial Light Modulator)

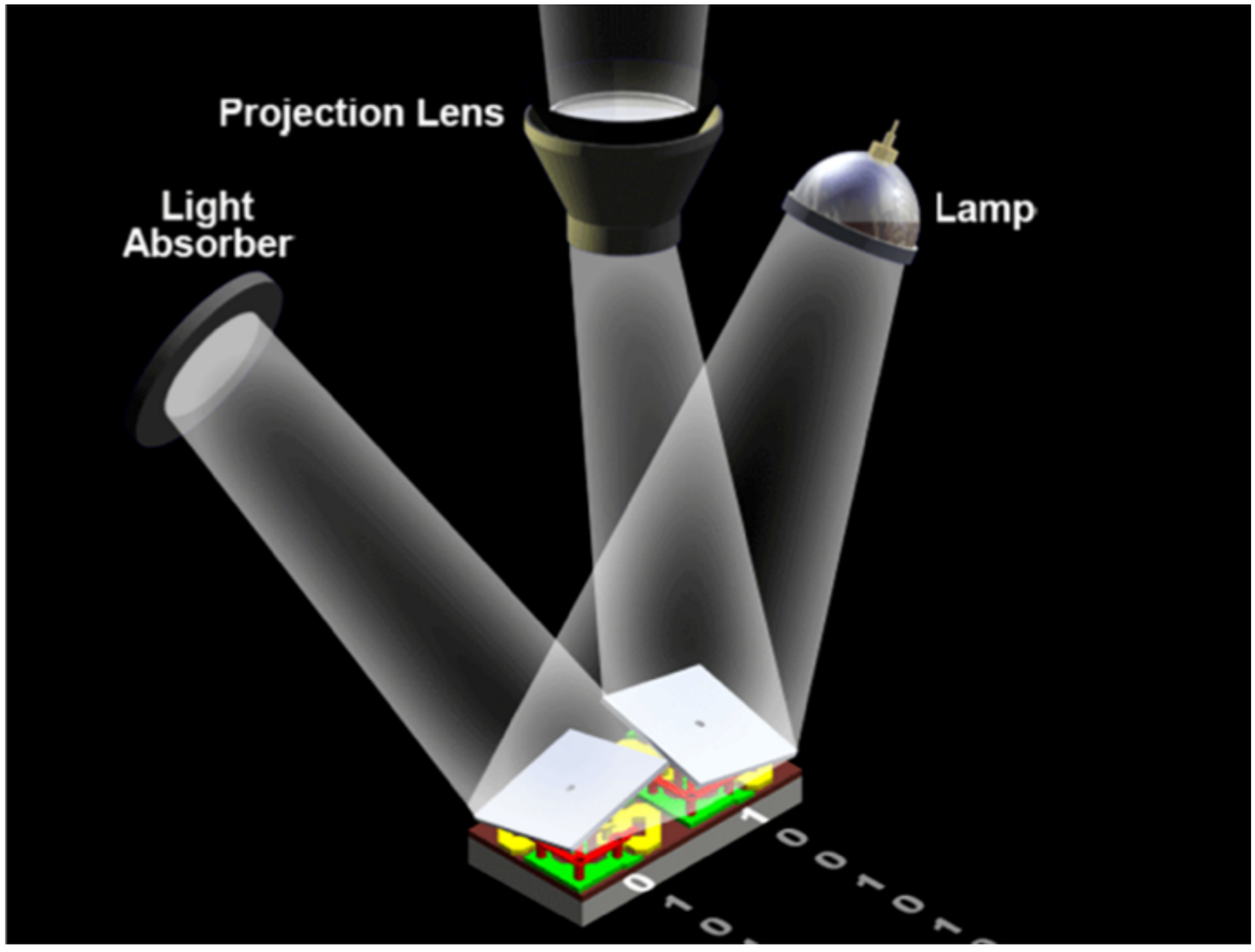


MICRO MIRRORS CLOSE UP

[Y.K. Rabinowitz; EKB Technologies

DMD Projection Display



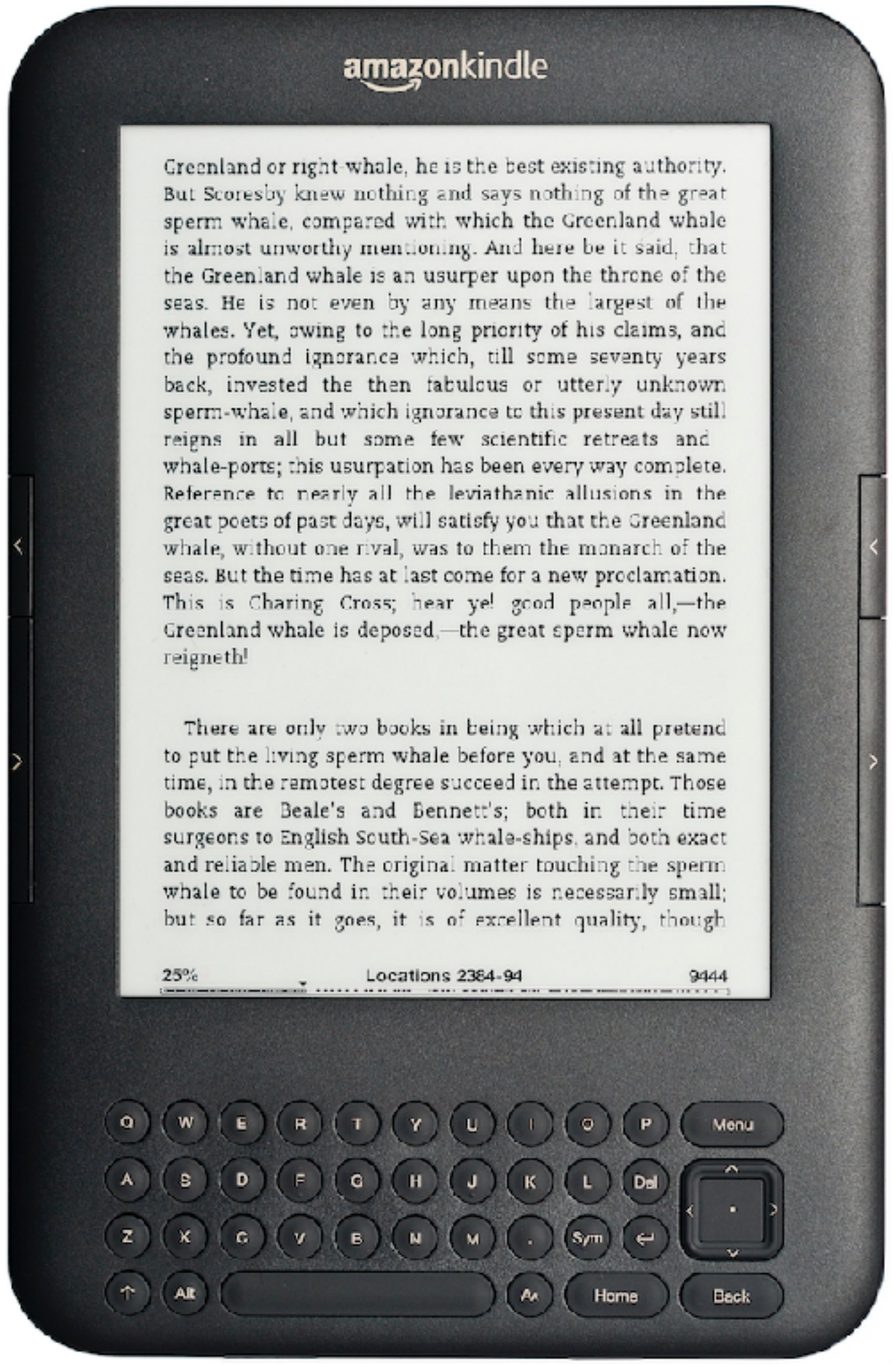


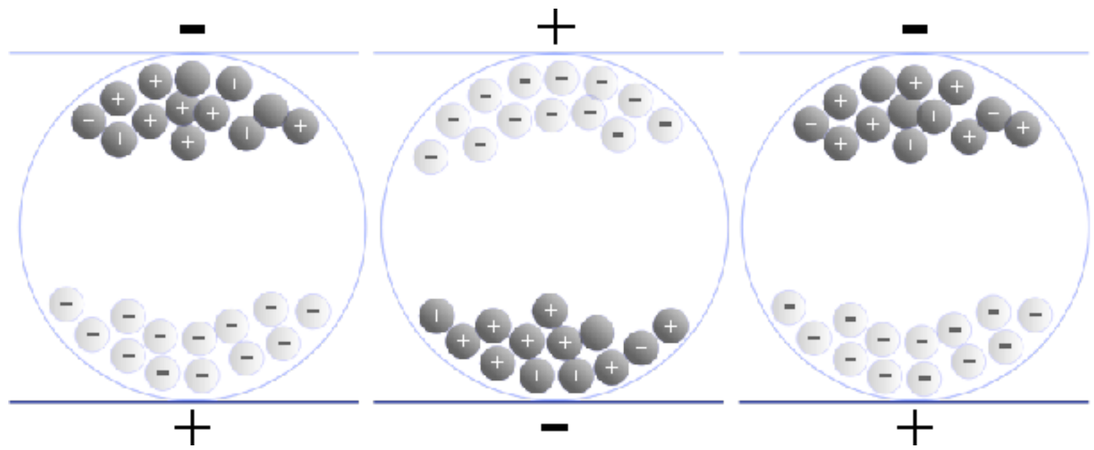
[Texas Instruments]

Array of micro-mirror pixels

DMD = Digital Micromirror Device

Electrophoretic (Electronic Ink) Display



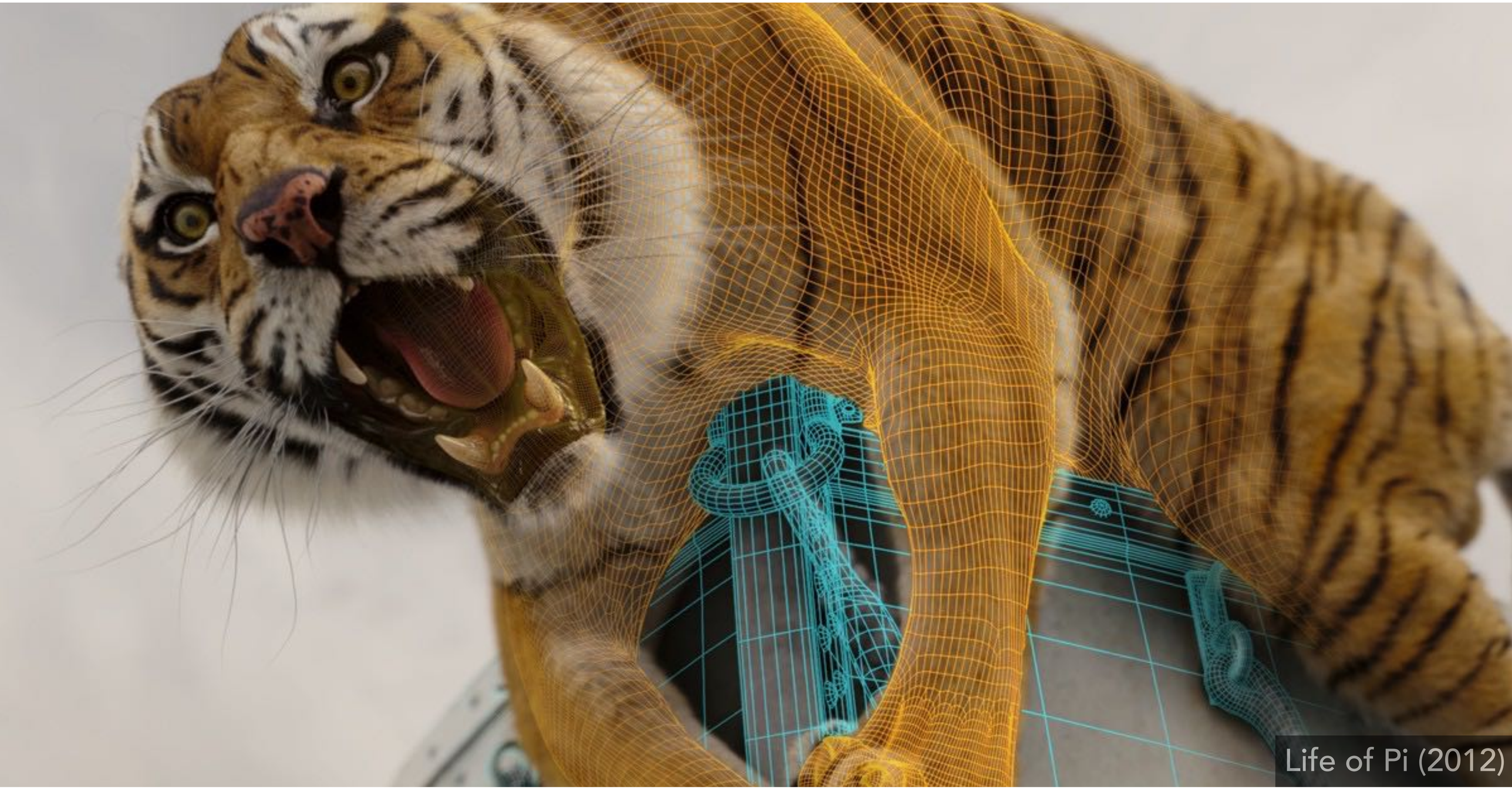


[Wikimedia Commons —Senarclens]

CS184/284A

Drawing to Raster Displays

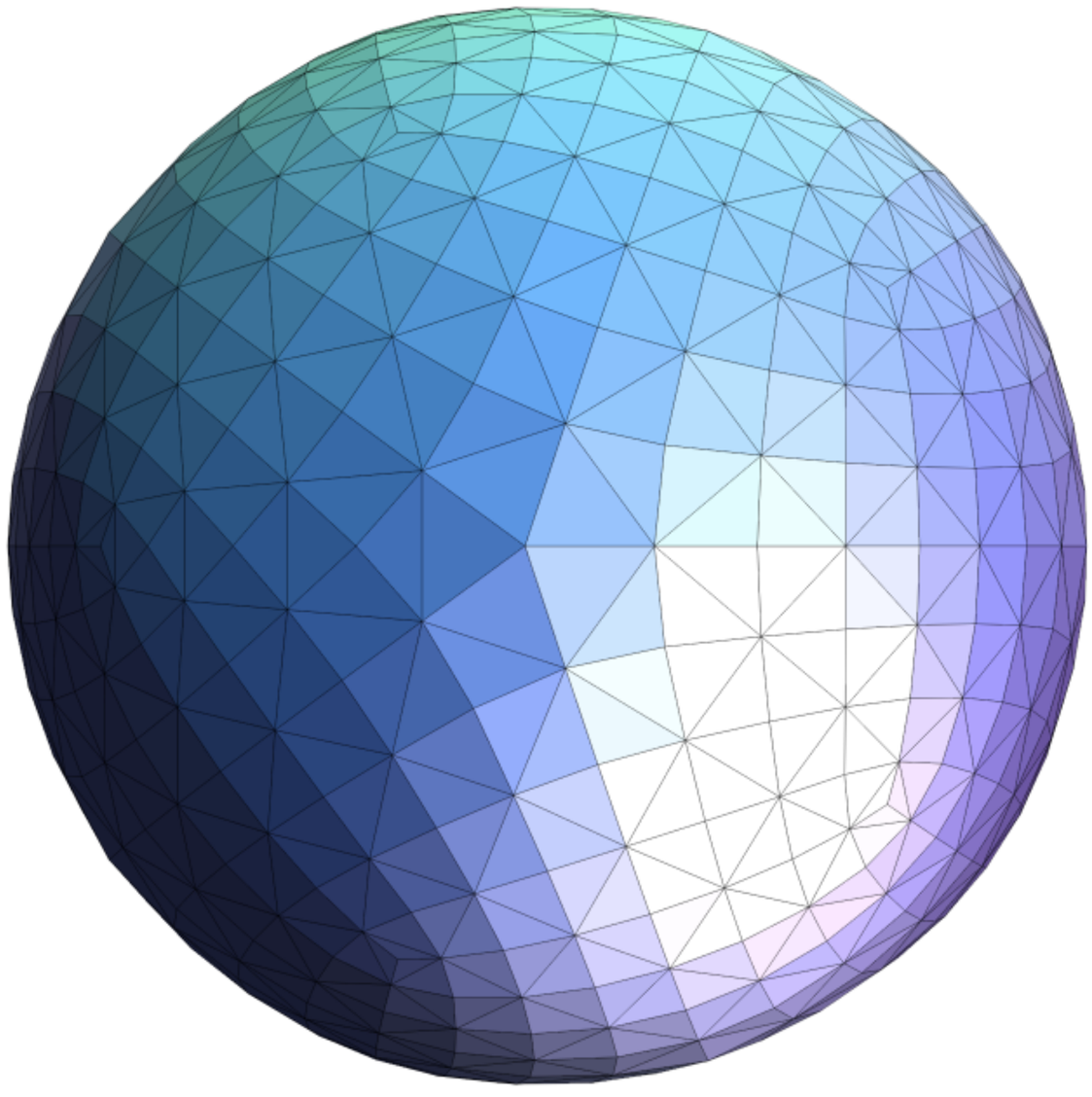
Polygon Meshes



Life of Pi (2012)

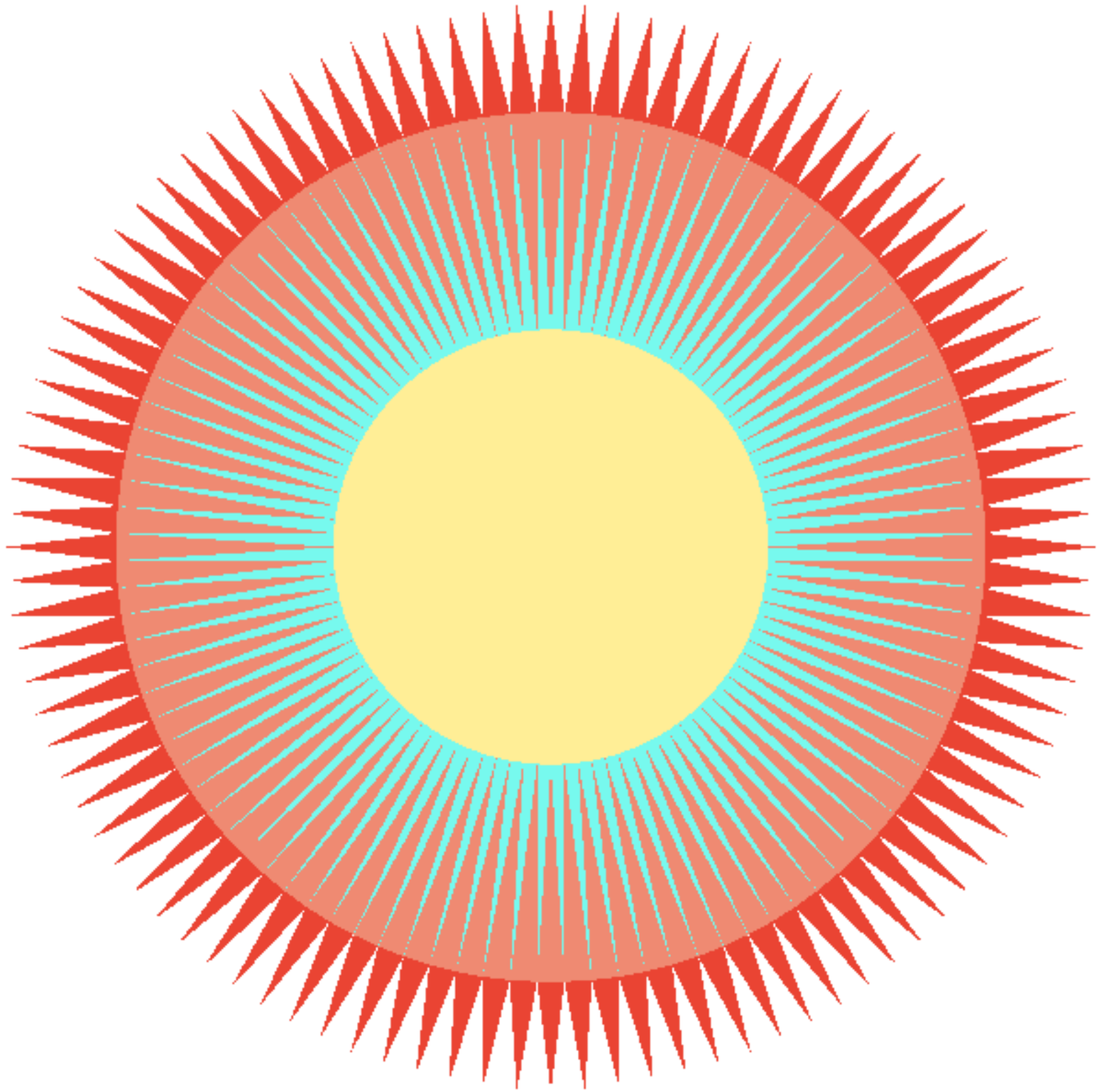
CS184/284A

Triangle Meshes



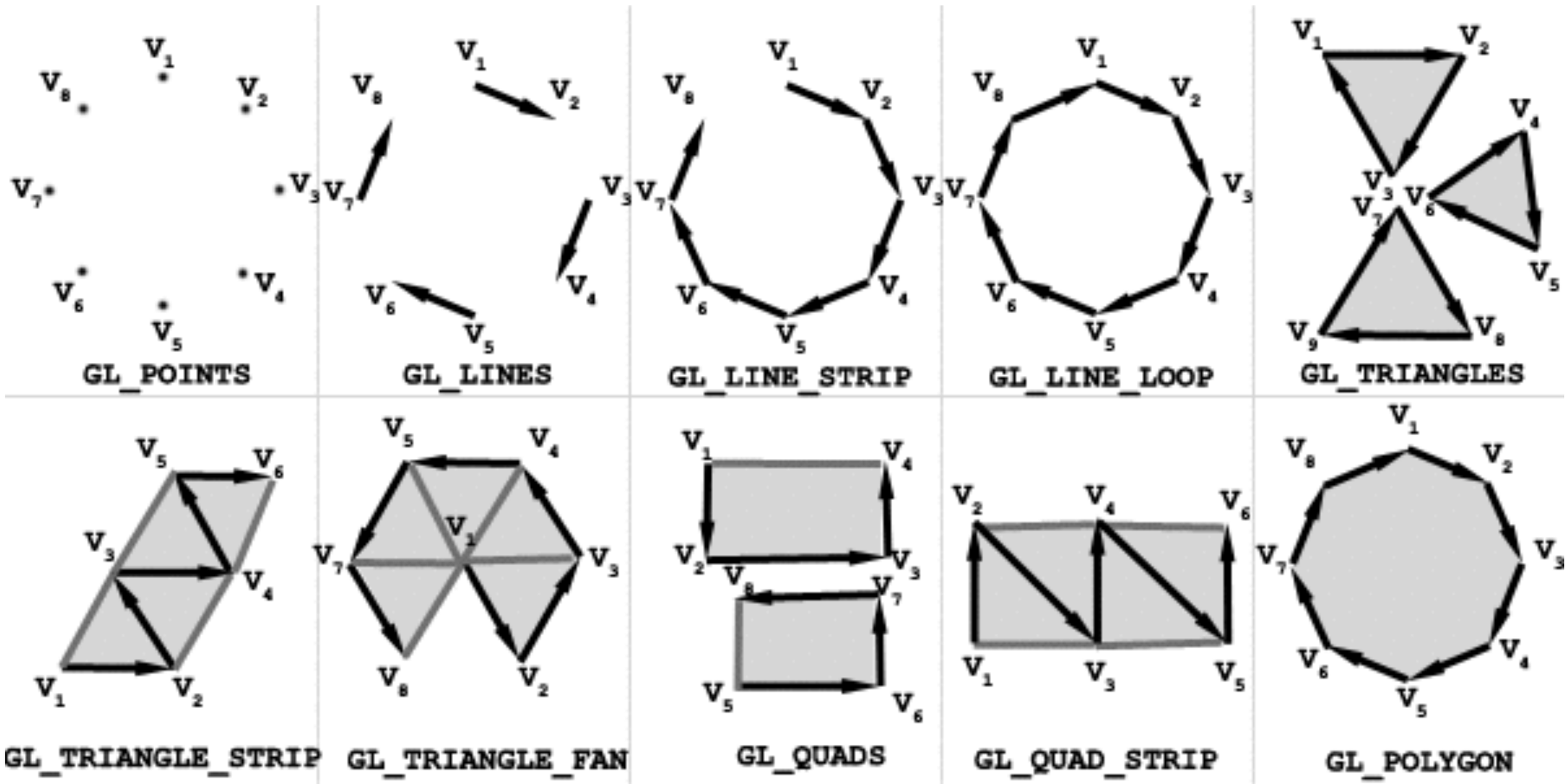
CS184/284A

Triangle Meshes



CS184/284A

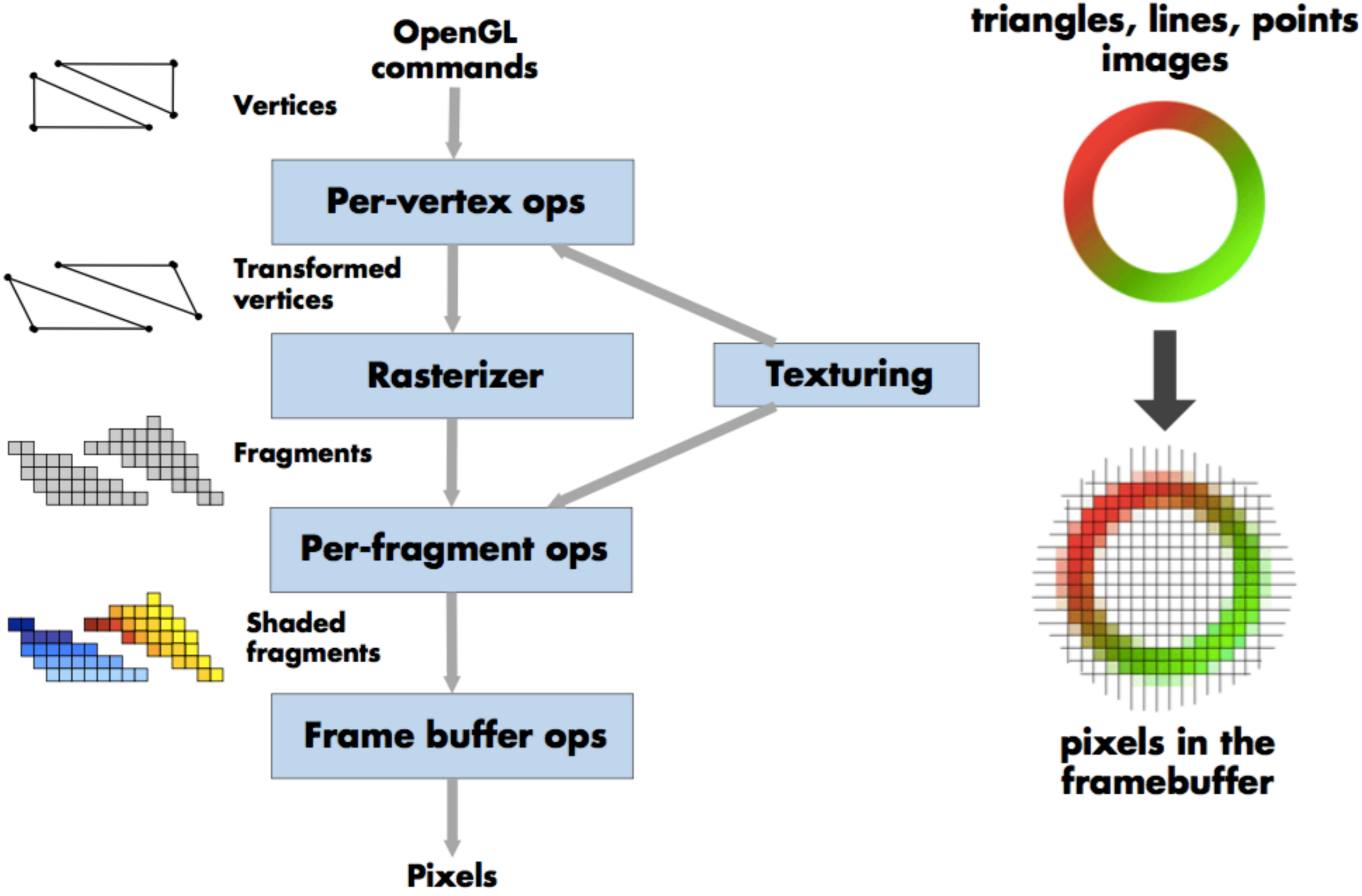
Shape Primitives



3dgep.com

Example shape primitives (OpenGL)

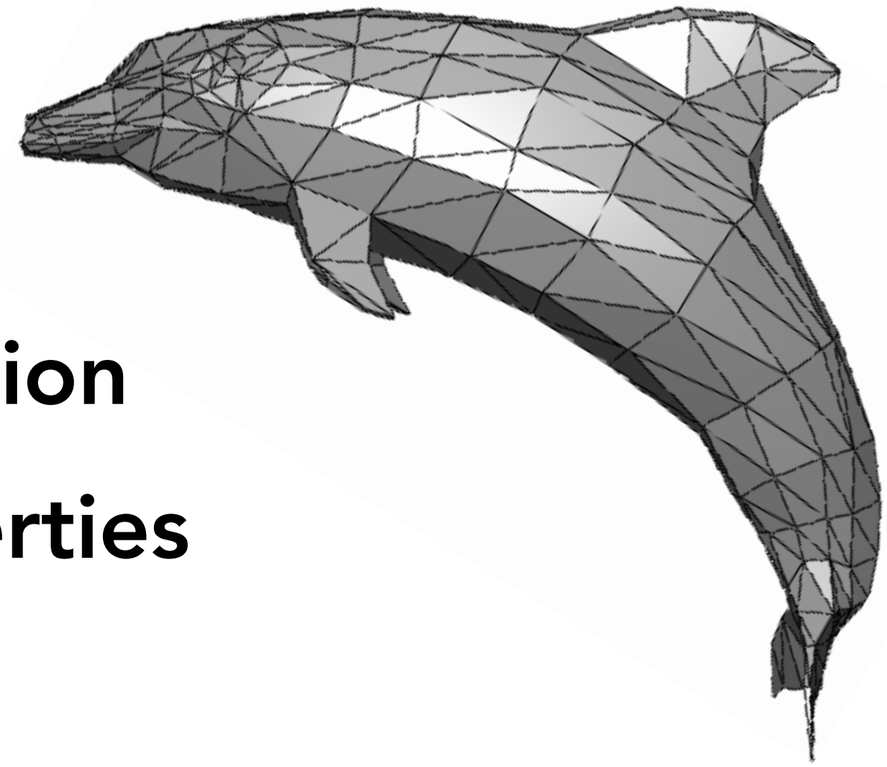
Graphics Pipeline = Abstract Drawing Machine



Triangles - Fundamental Area Primitive

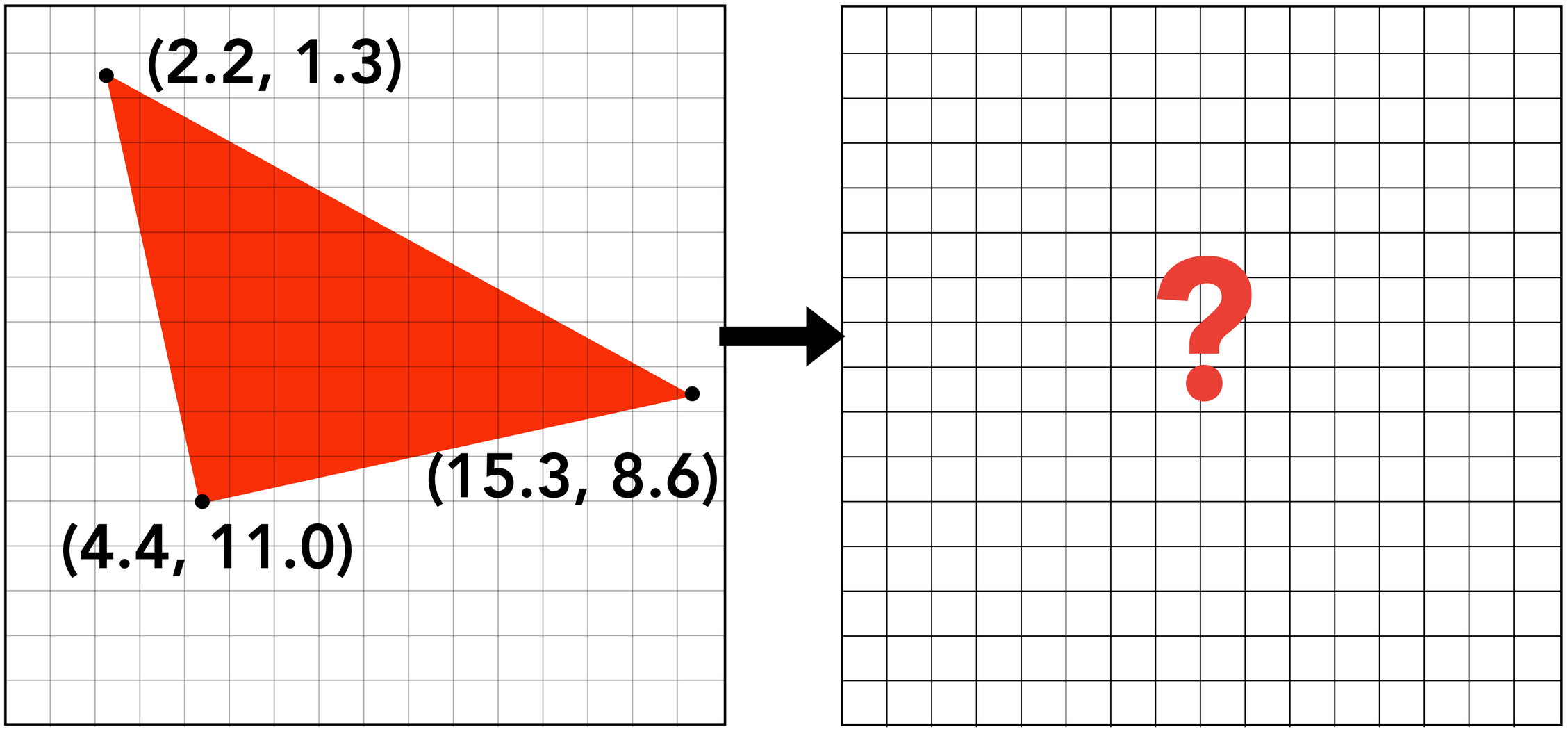
Why triangles?

- Most basic polygon
  - Break up other polygons
  - Optimize one implementation
- Triangles have unique properties
  - Guaranteed to be planar
  - Well-defined interior
  - Well-defined method for interpolating values at vertices over triangle (barycentric interpolation)



Drawing a Triangle To The Framebuffer (“Rasterization”)

What Pixel Values Approximate a Triangle?



Input: position of triangle vertices projected on screen

Output: set of pixel values approximating triangle

Today, Let's Start With A Simple Approach: Sampling

Sampling a Function

Evaluating a function at a point is sampling.

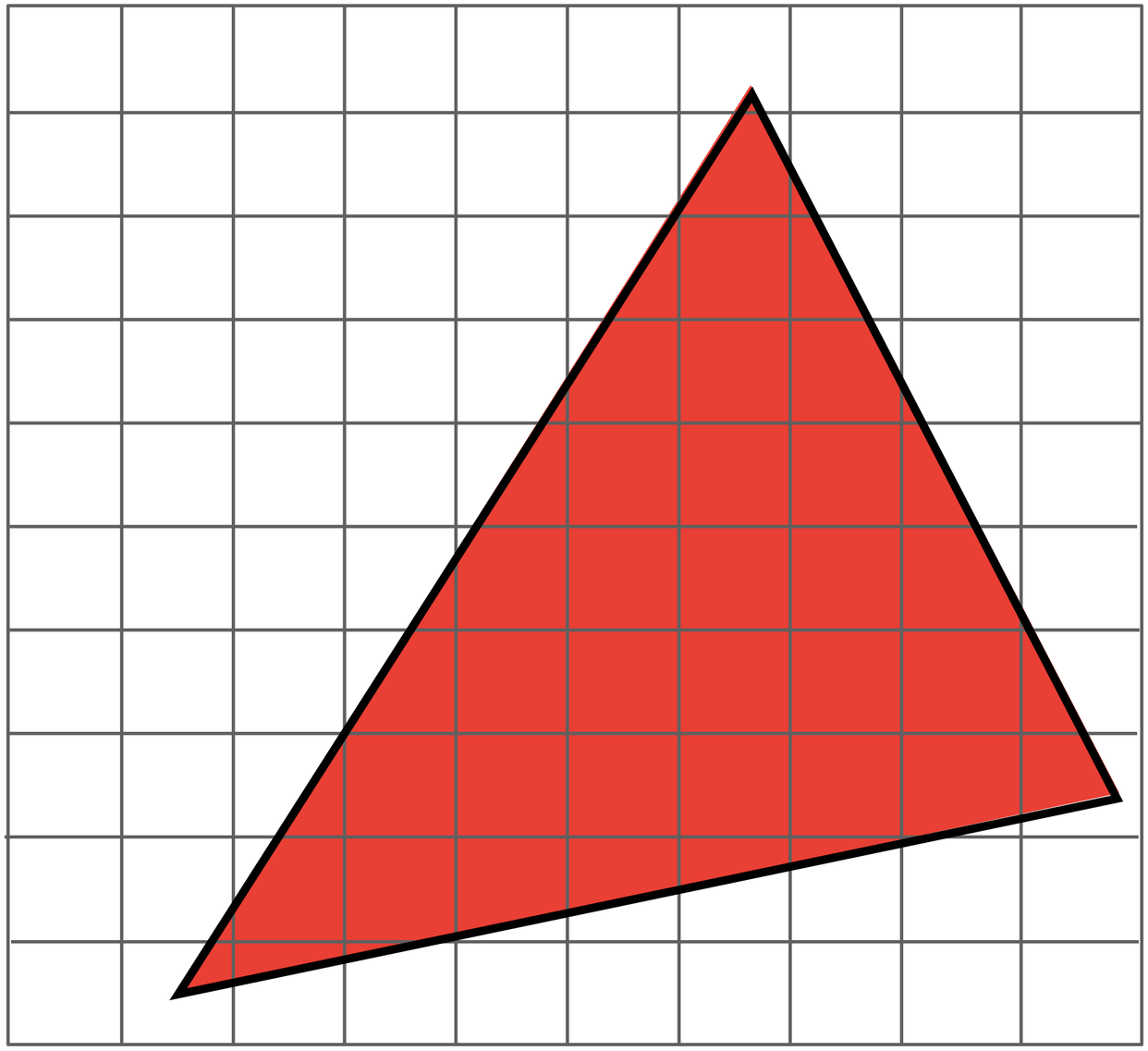
We can discretize a function by periodic sampling.

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

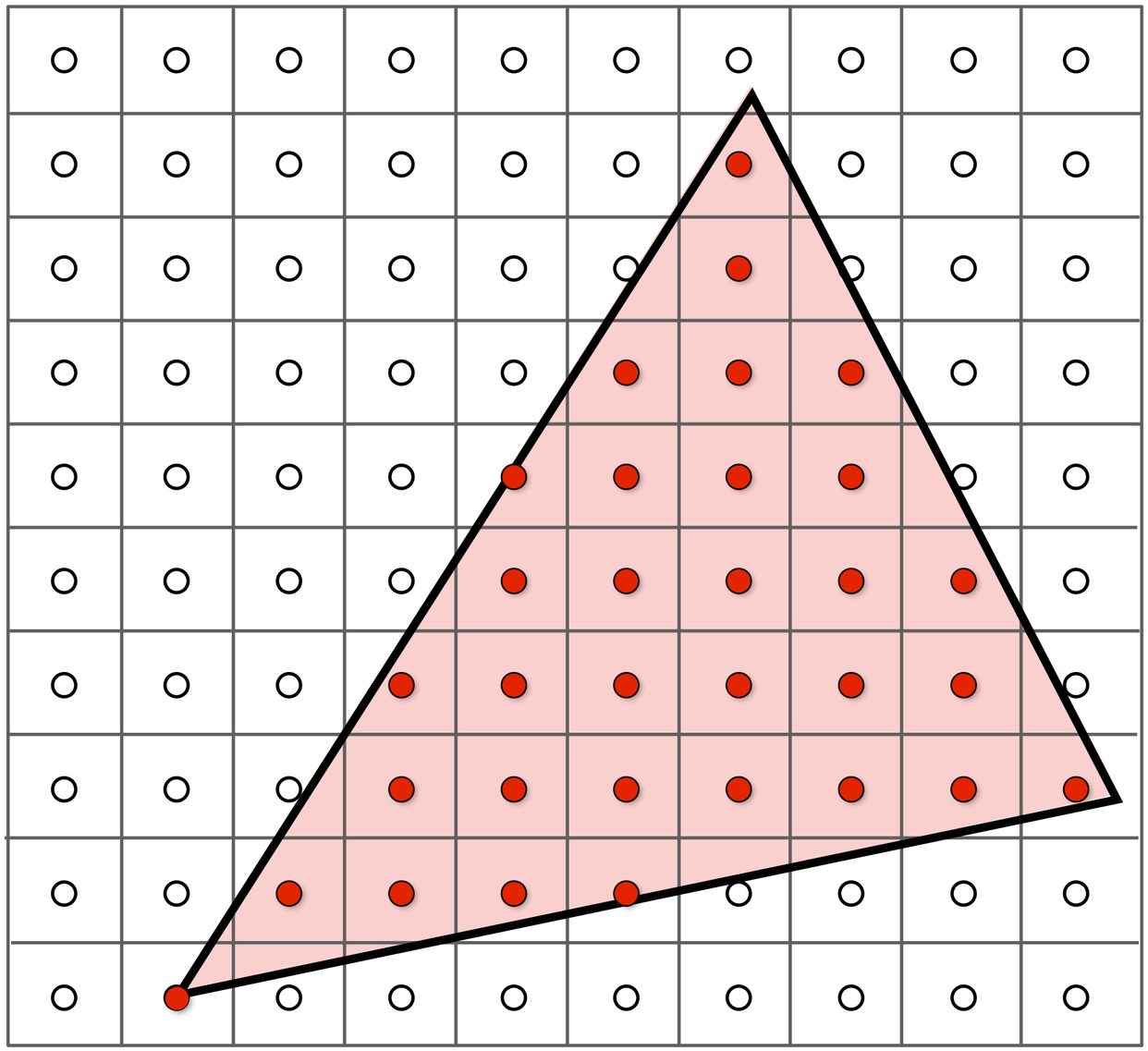
Sampling is a core idea in graphics. We'll sample time (1D), area (2D), angle (2D), volume (3D) …

We'll sample N-dimensional functions, even infinite dimensional functions.

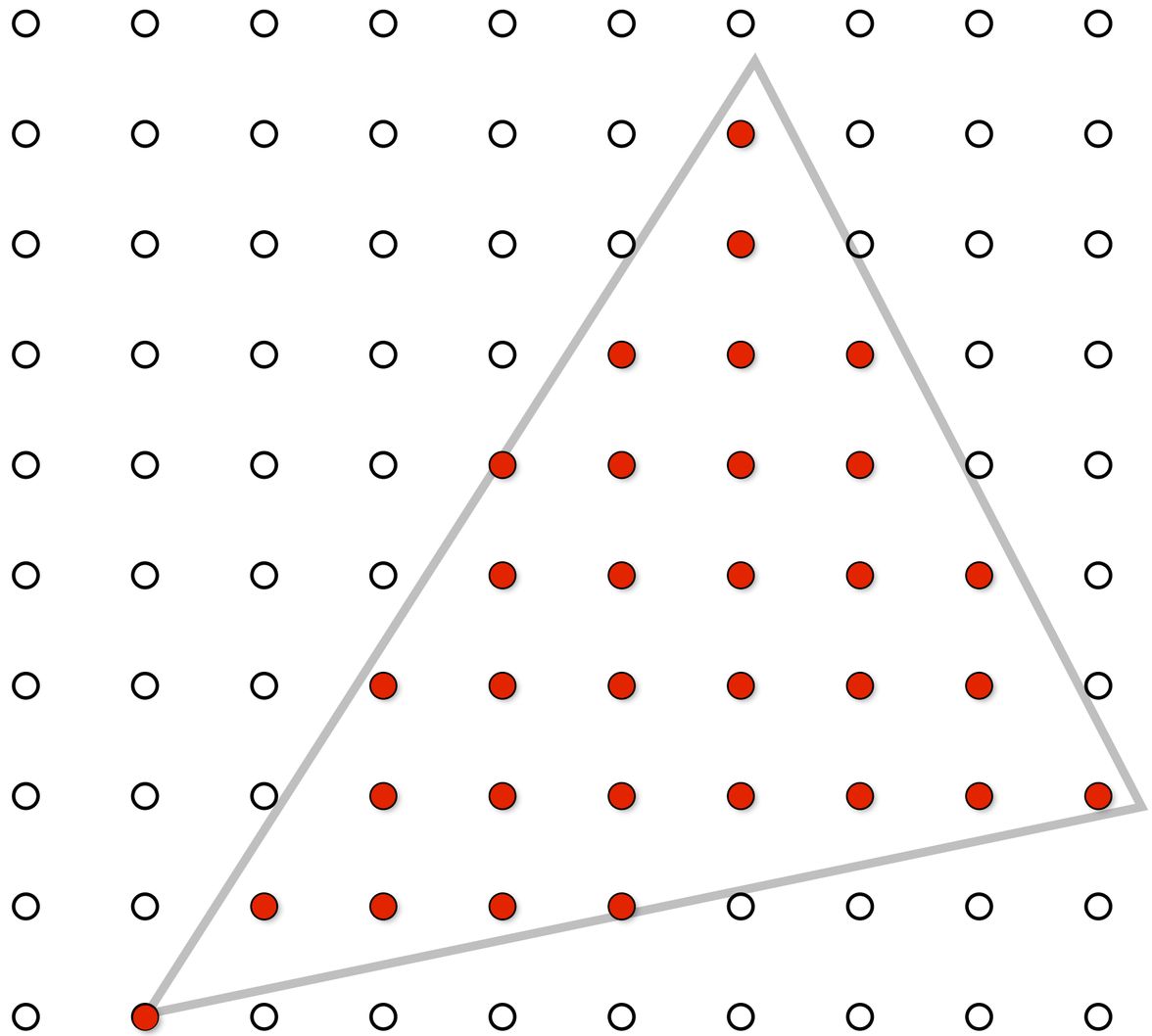
Let's Try Rasterization As 2D Sampling



Sample If Each Pixel Center Is Inside Triangle

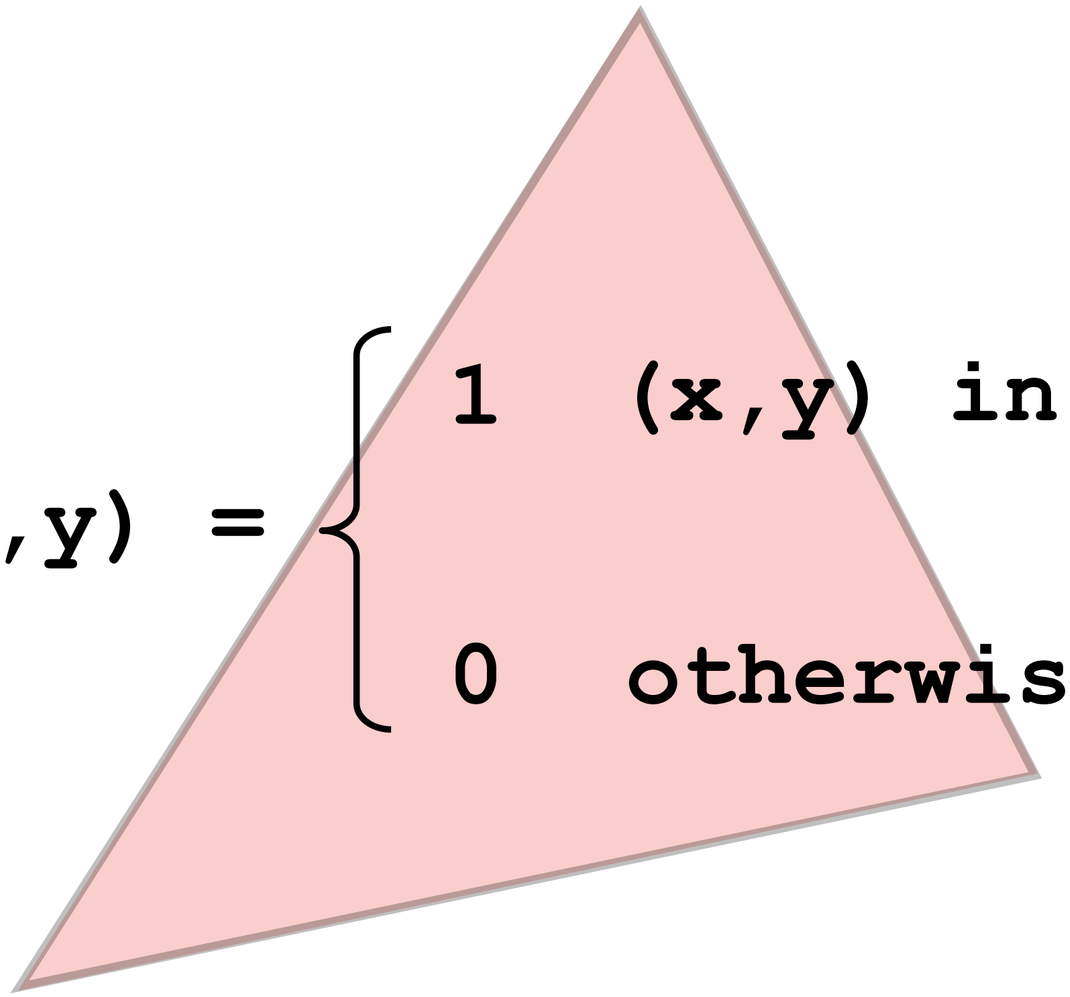


Sample If Each Pixel Center Is Inside Triangle



Define Binary Function: `inside(tri,x,y)`

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

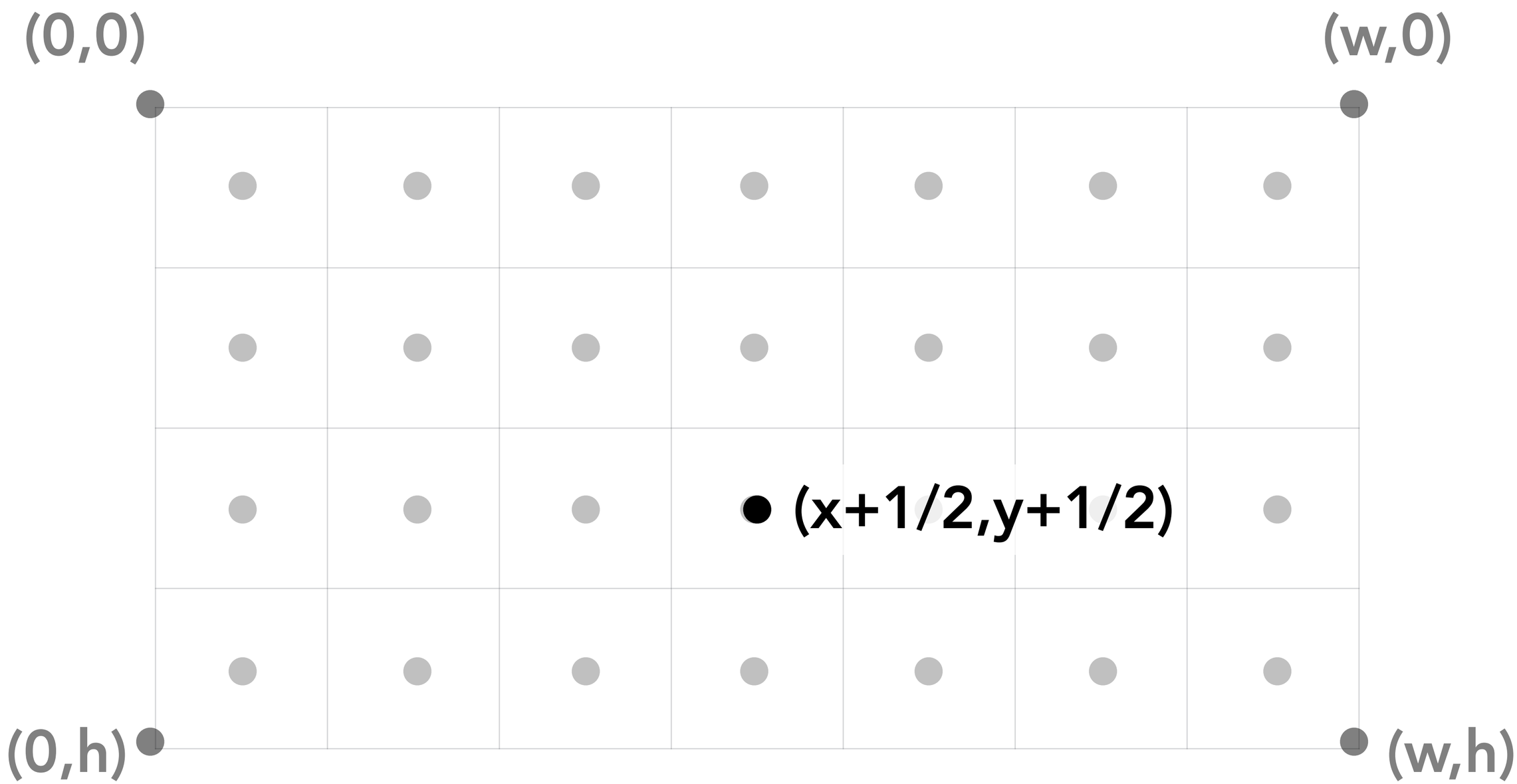


Rasterization = Sampling A 2D Indicator Function

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

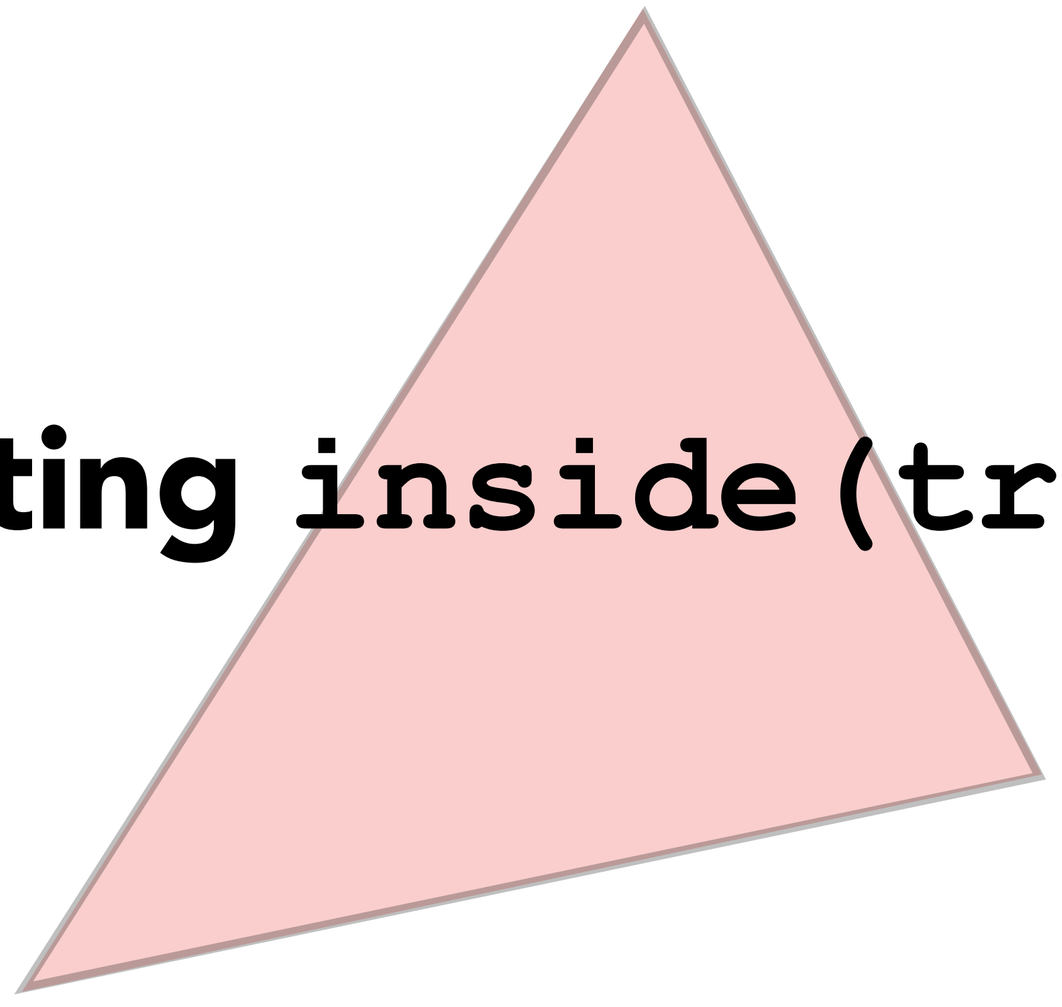
Rasterize triangle `tri` by sampling the function
`f(x,y) = inside(tri,x,y)`

Implementation Detail: Sample Locations

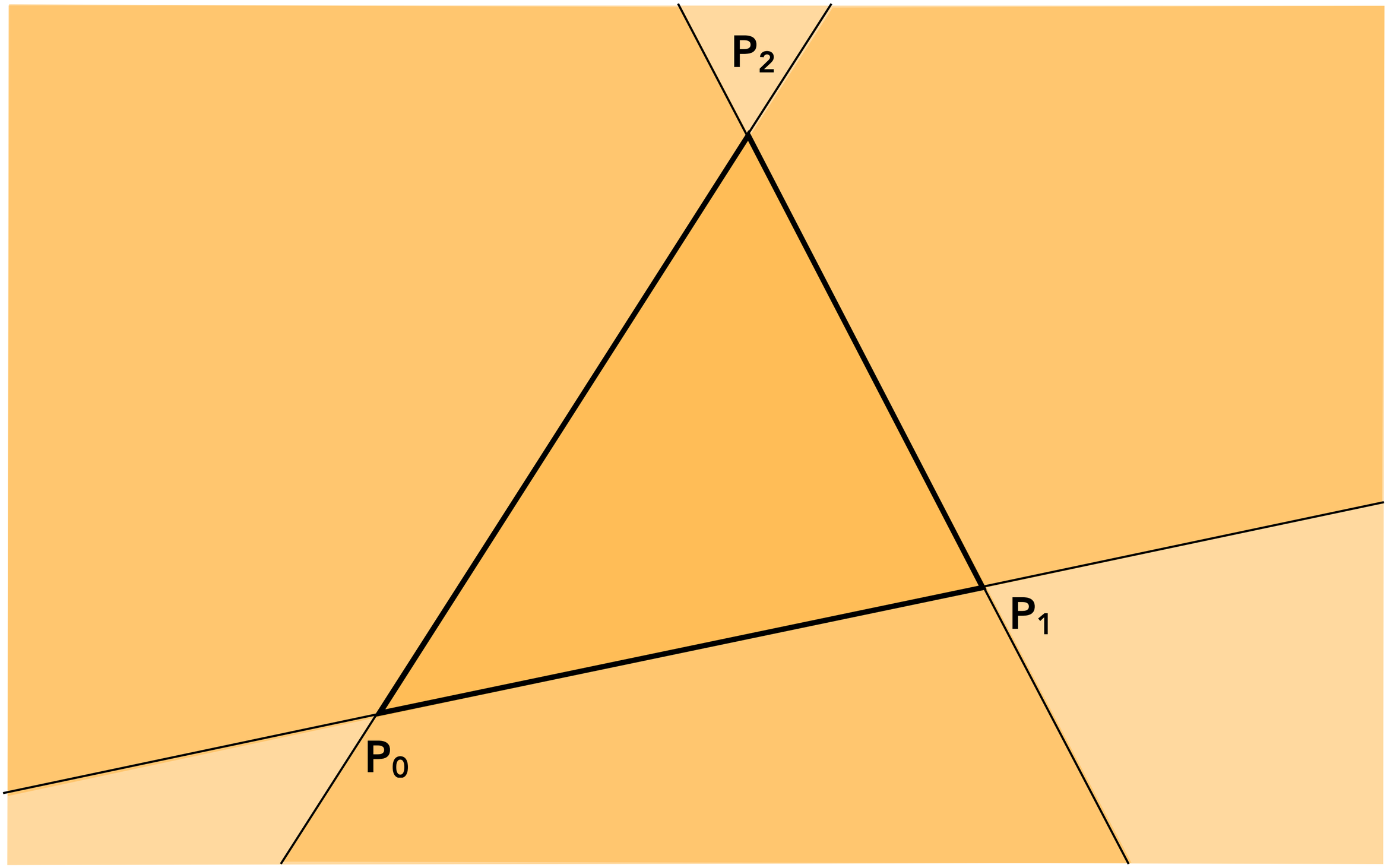


Sample location for pixel (x,y)

Evaluating `inside(tri,x,y)`



Triangle = Intersection of Three Half Planes

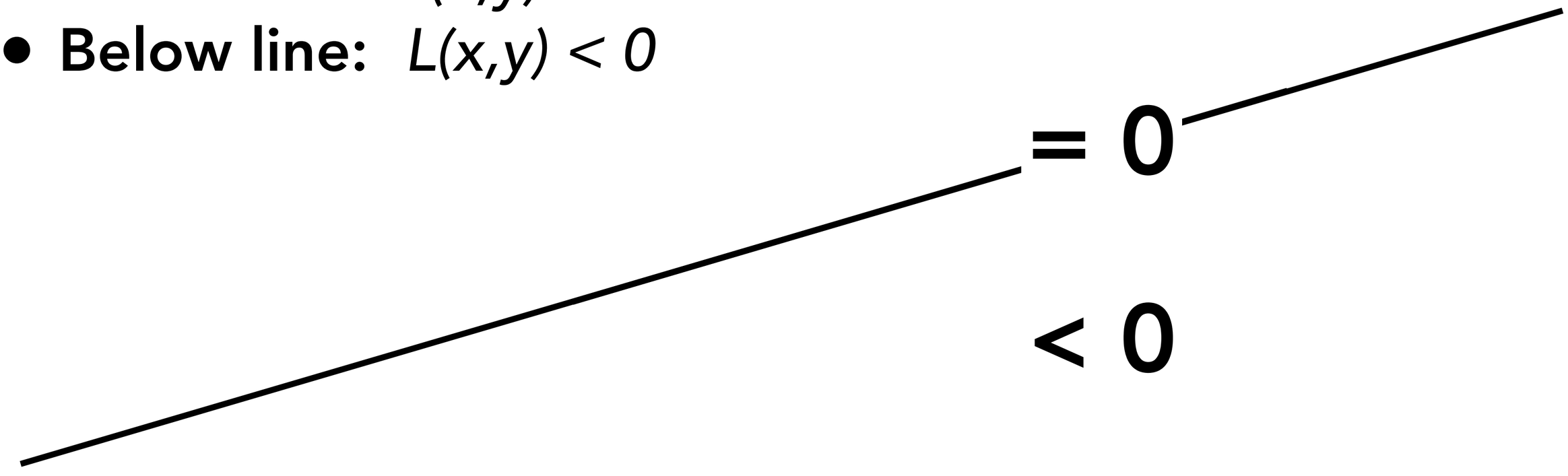


Each Line Defines Two Half-Planes

Implicit line equation

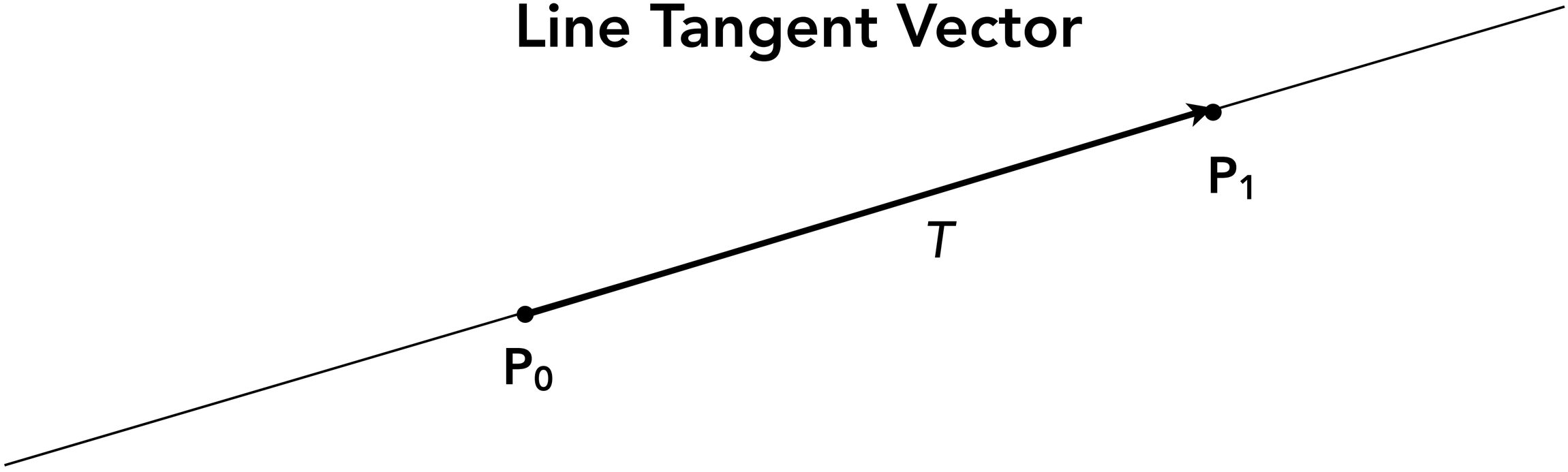
- $L(x,y) = Ax + By + C$

- **On line:** $L(x,y) = 0$
- **Above line:** $L(x,y) > 0$
- **Below line:** $L(x,y) < 0$



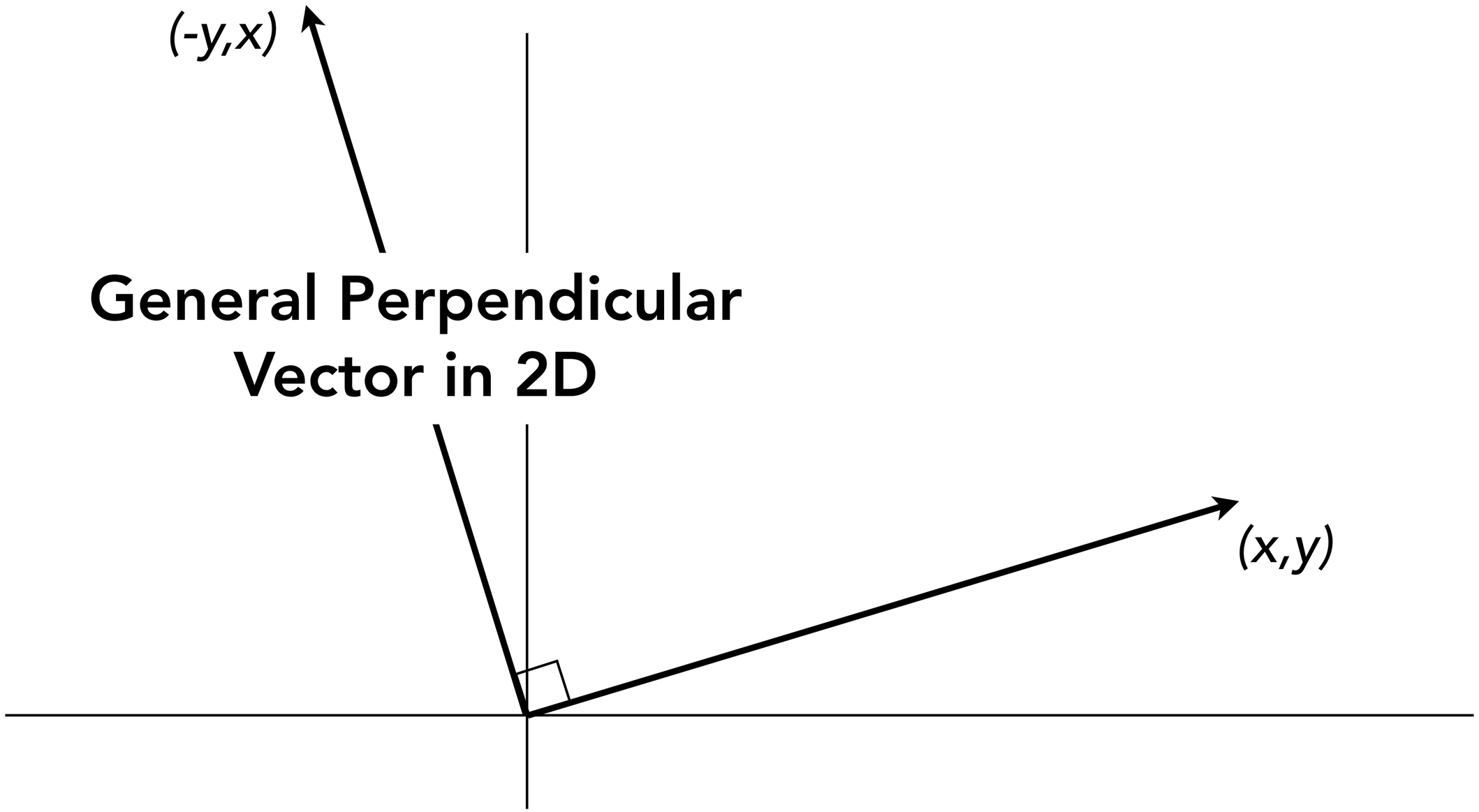
Line Equation Derivation

Line Tangent Vector



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

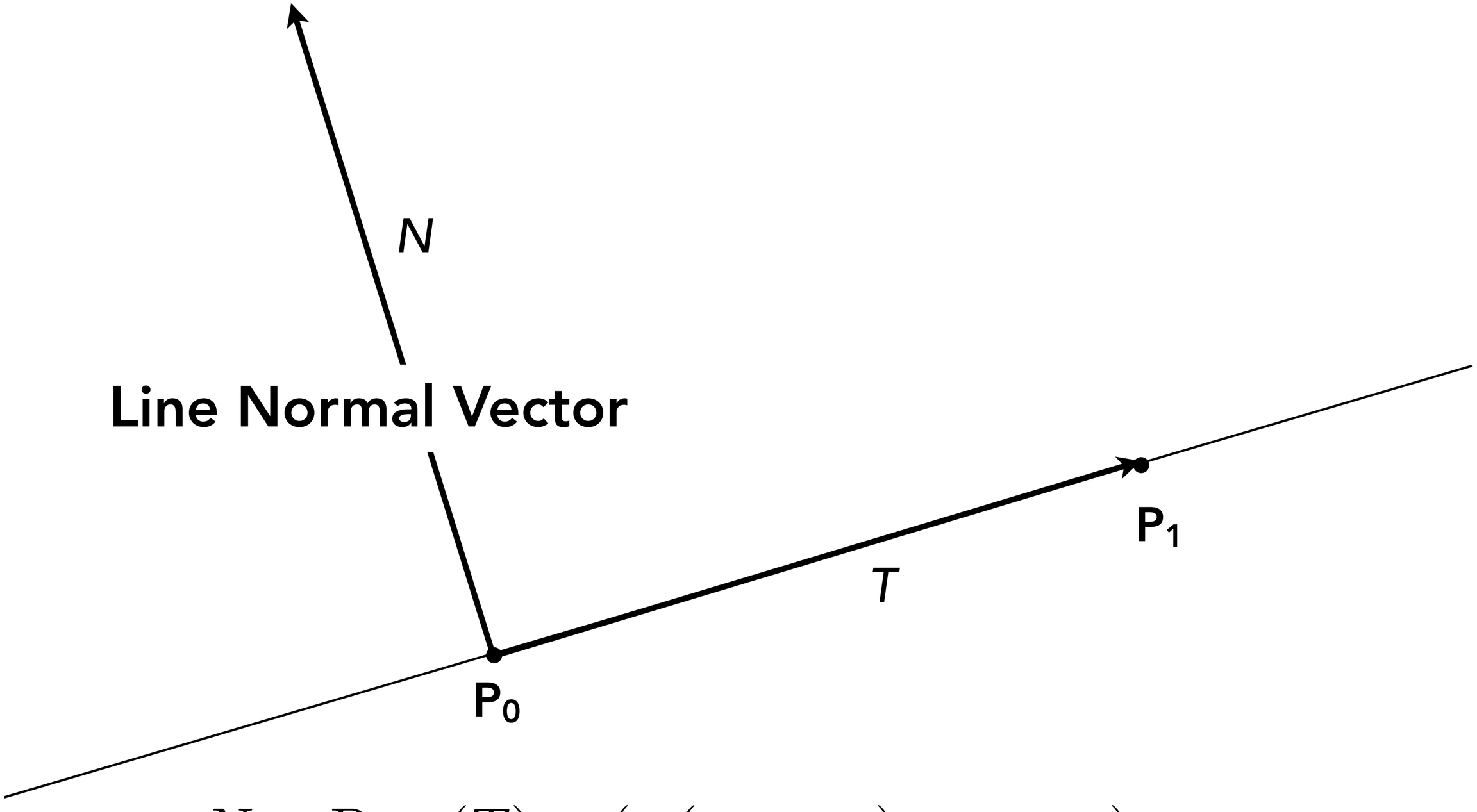
Line Equation Derivation



General Perpendicular Vector in 2D

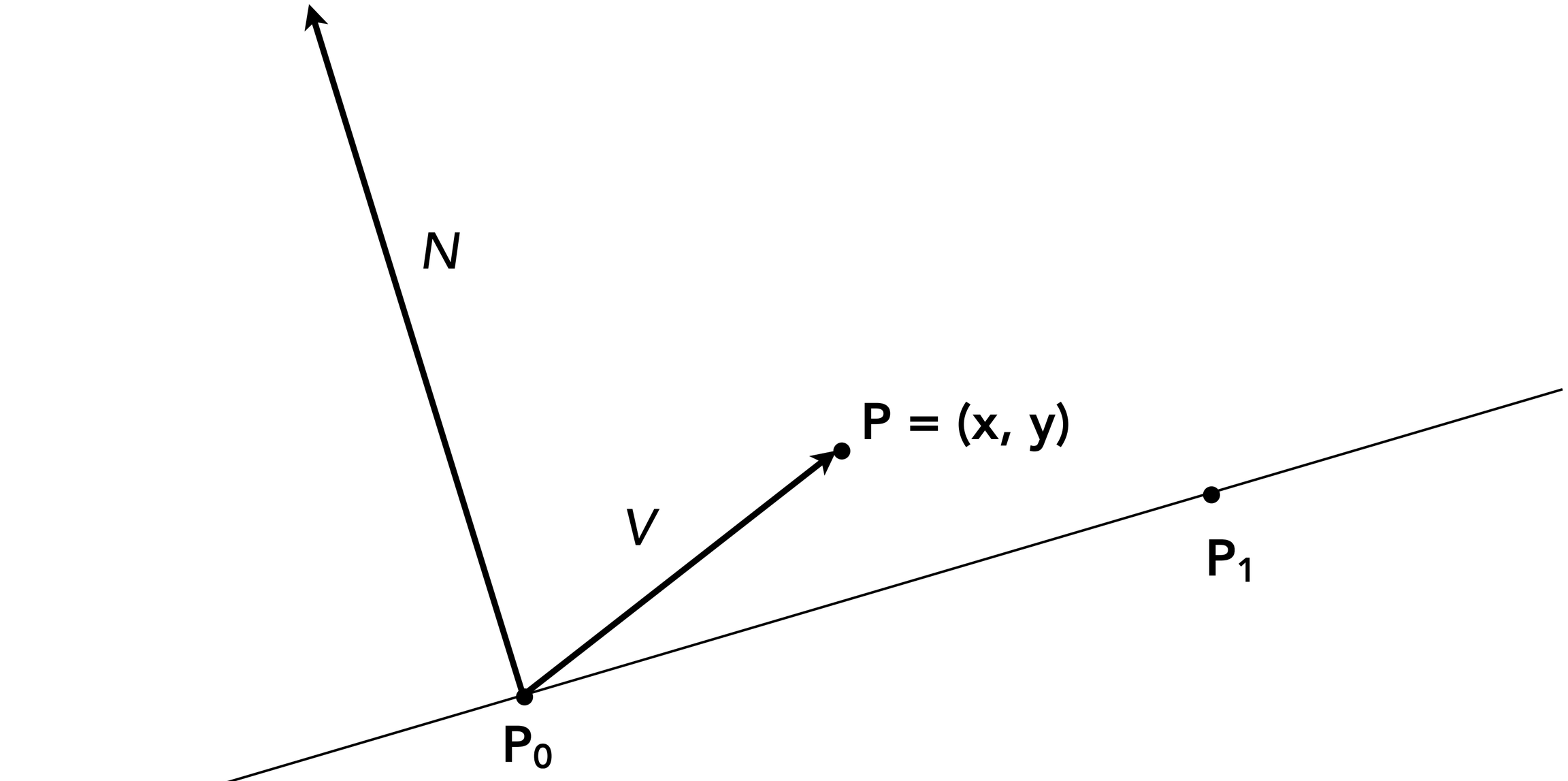
$$\text{Perp}(x, y) = (-y, x)$$

Line Equation Derivation



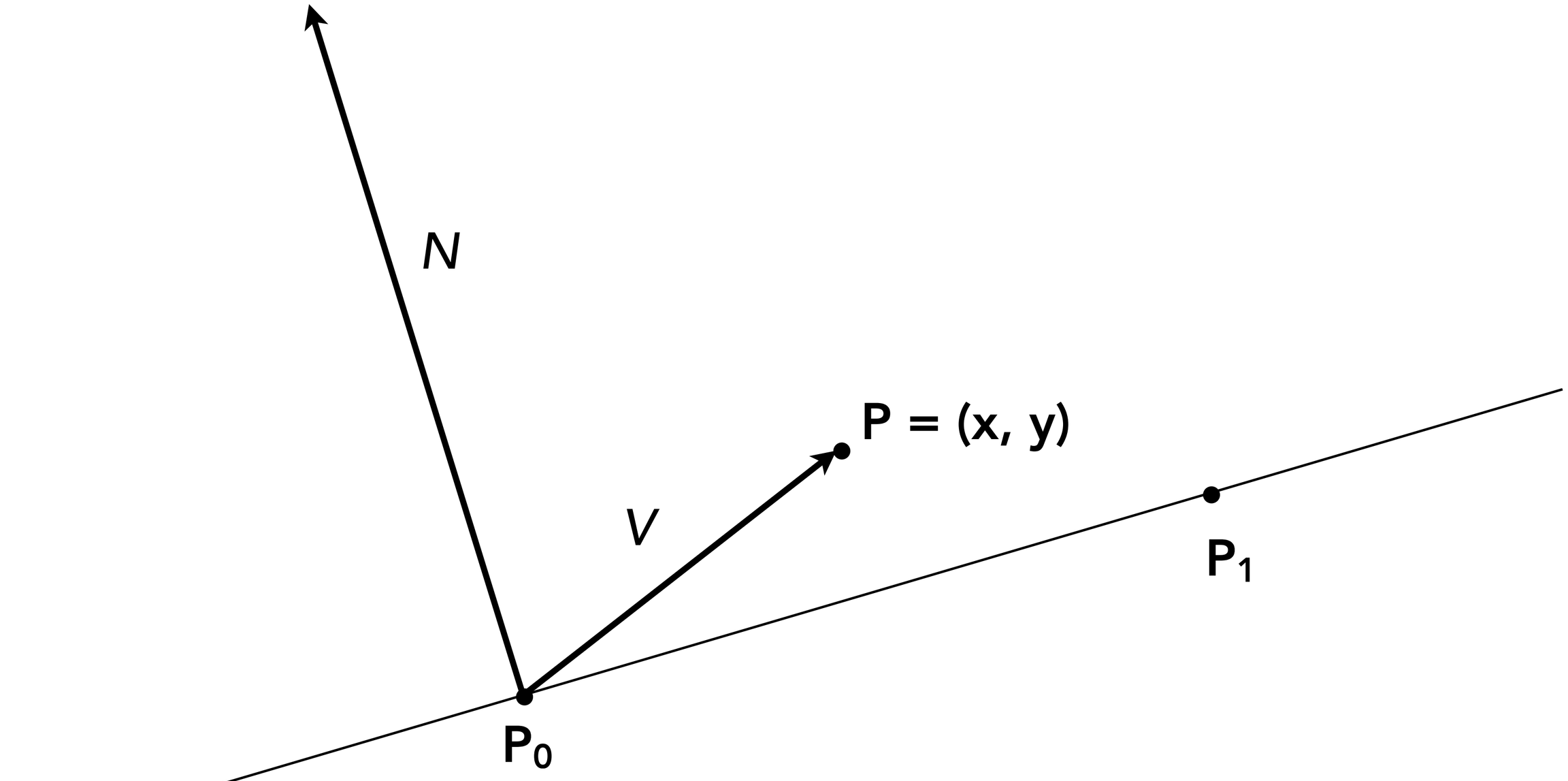
$$N = \text{Perp}(T) = (-(y_1 - y_0), x_1 - x_0)$$

Line Equation Derivation



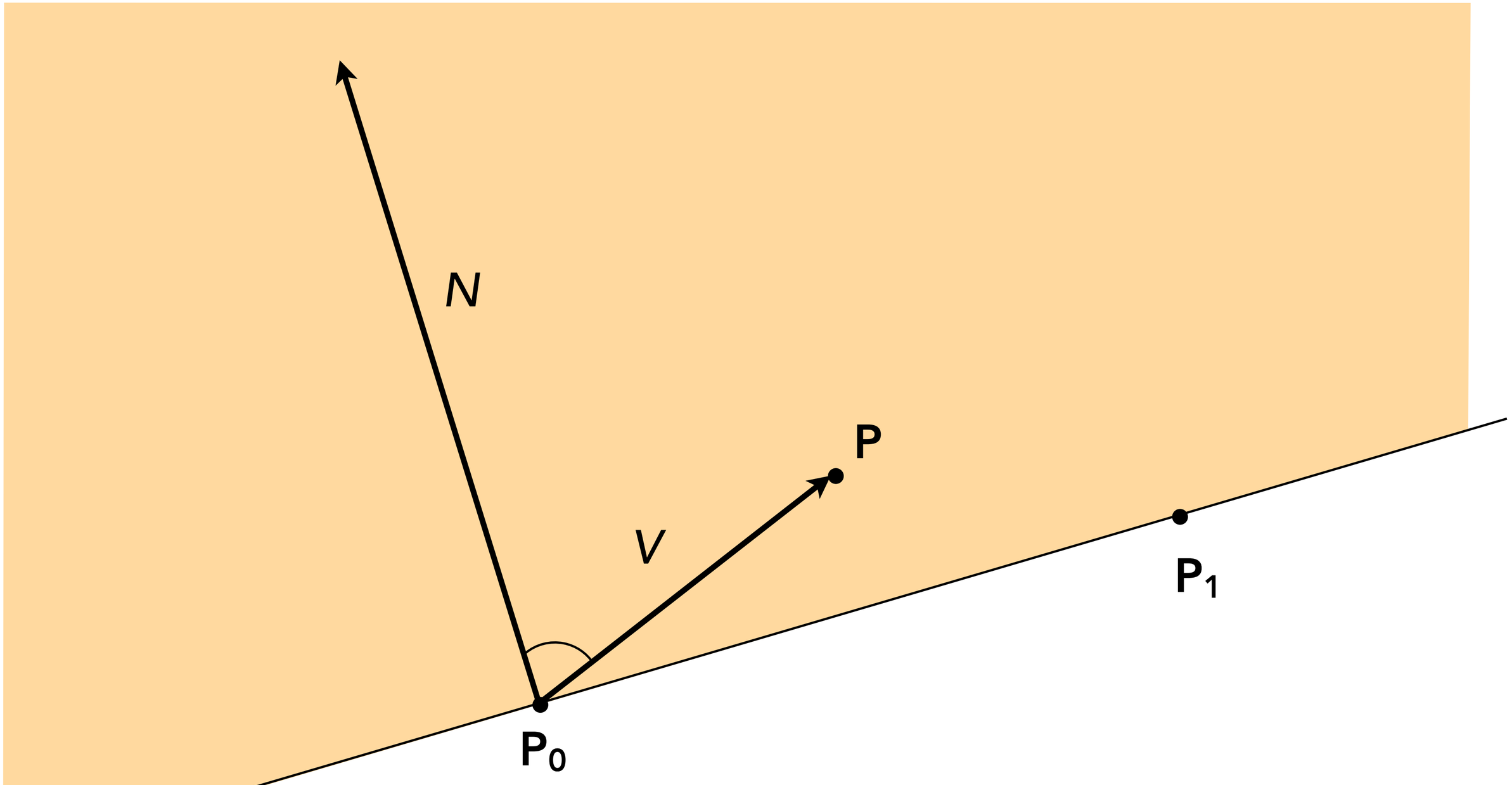
$$V = P - P_0 = (x - x_0, y - y_0)$$

Line Equation



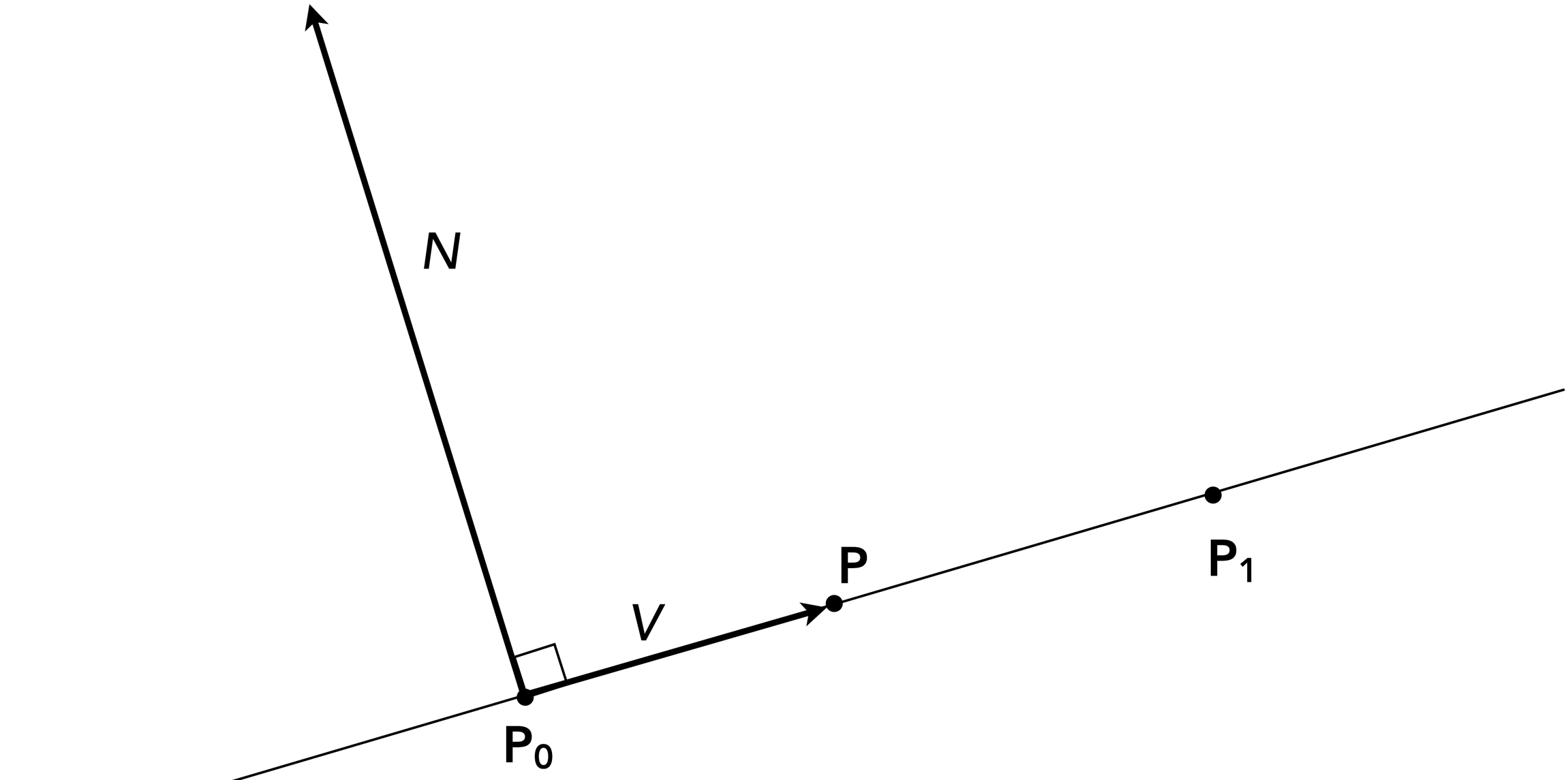
$$L(x,y) = V \cdot N = -(x - x_0)(y_1 - y_0) + (y - y_0)(x_1 - x_0)$$

Line Equation Tests



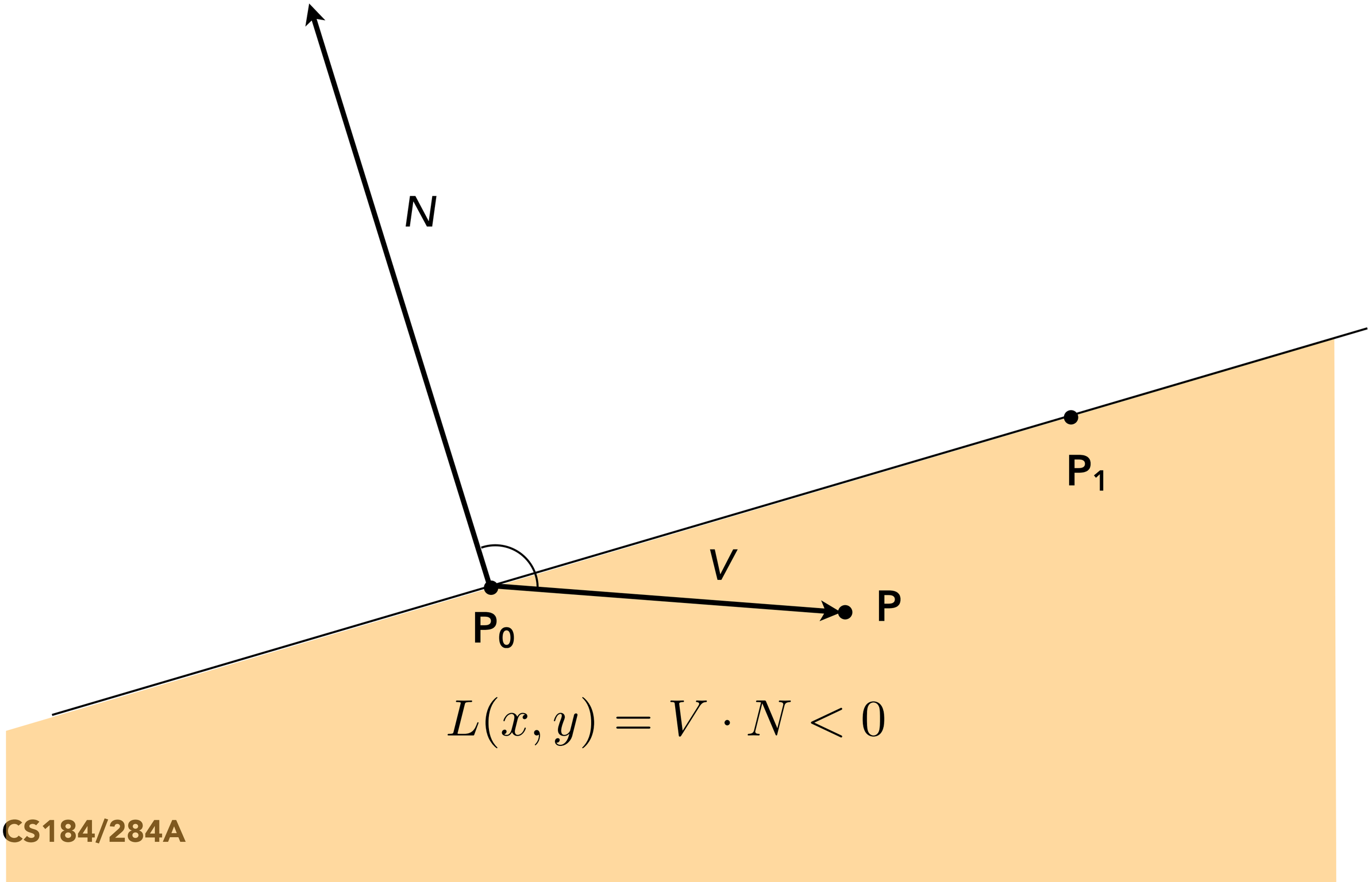
$$L(x, y) = V \cdot N > 0$$

Line Equation Tests



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

Line Equation Tests



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

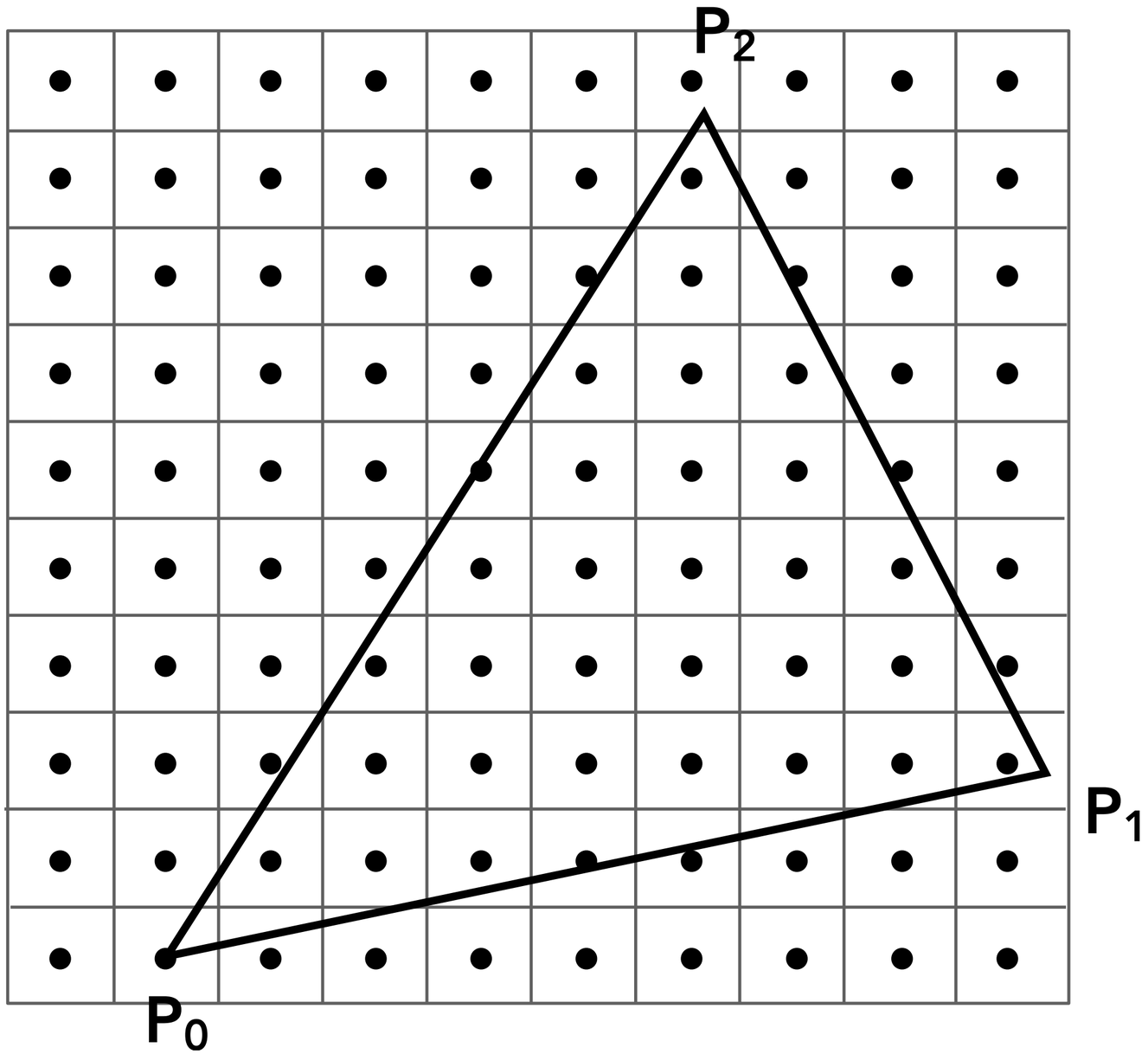
Point-in-Triangle Test: Three Line Tests

$P_i = (X_i, Y_i)$

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

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

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



Compute line equations from pairs of vertices

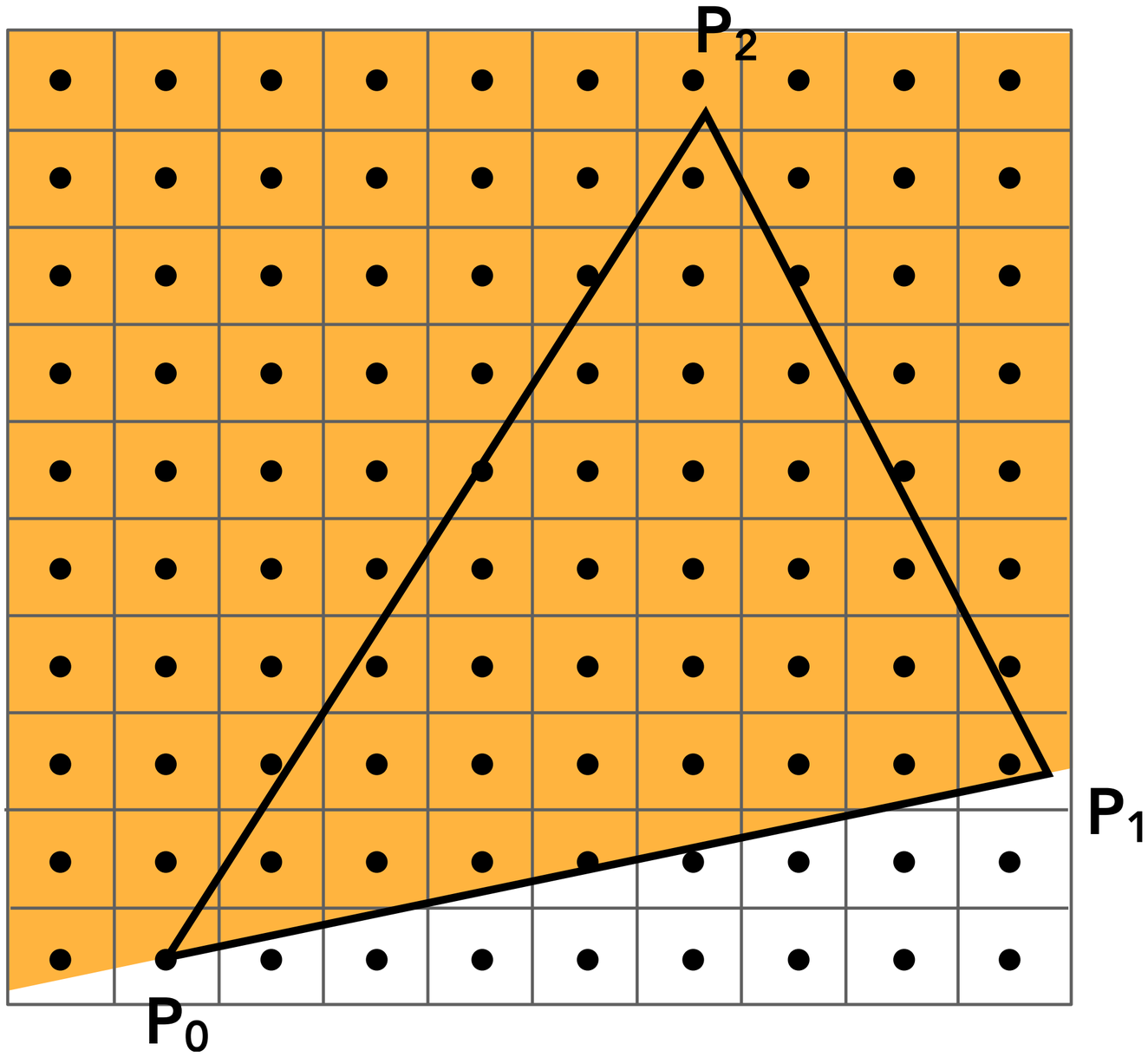
Point-in-Triangle Test: Three Line Tests

$P_i = (X_i, Y_i)$

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

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

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



$L_0(x, y) > 0$

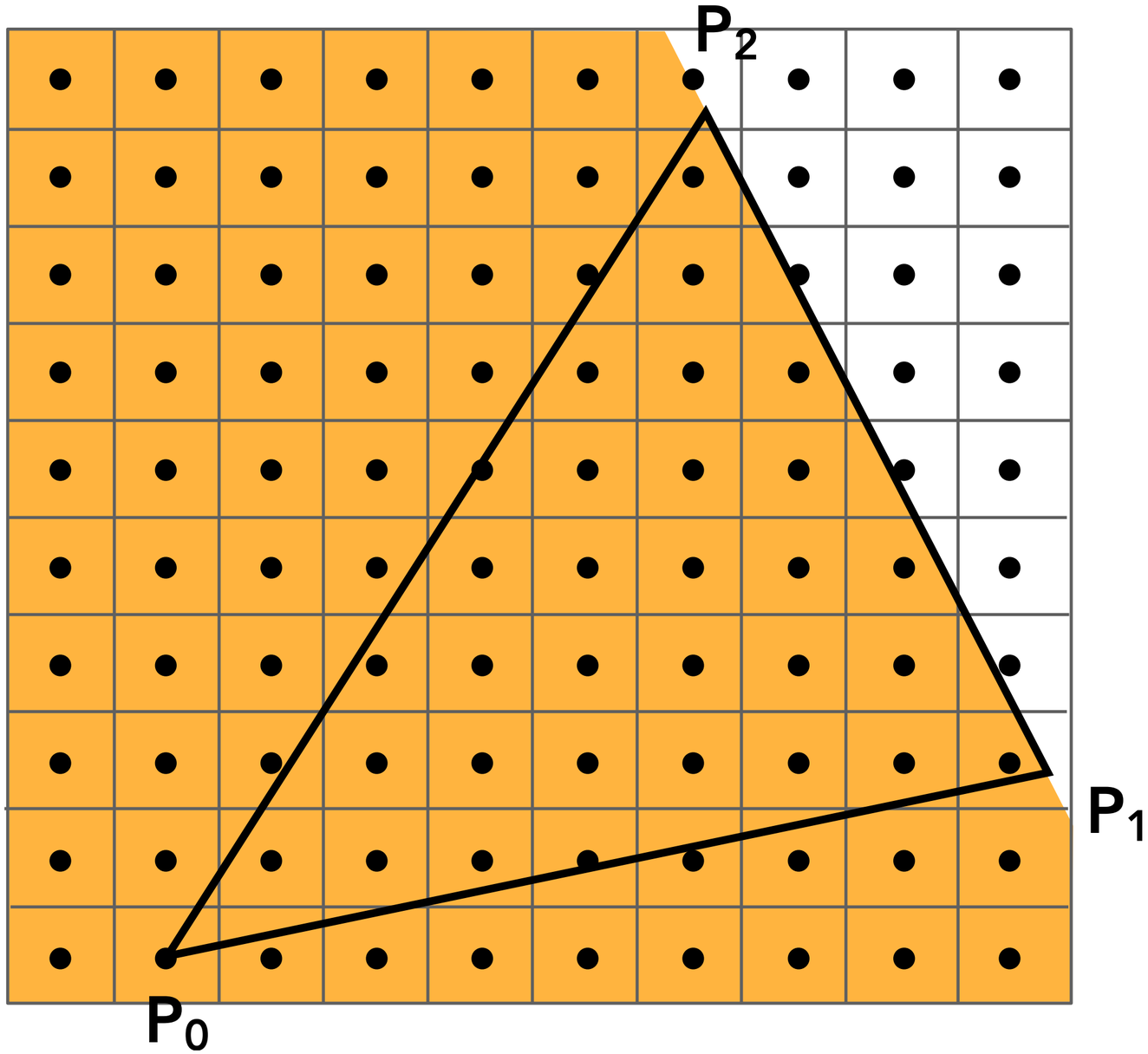
Point-in-Triangle Test: Three Line Tests

$P_i = (X_i, Y_i)$

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

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

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



$L_1(x, y) > 0$

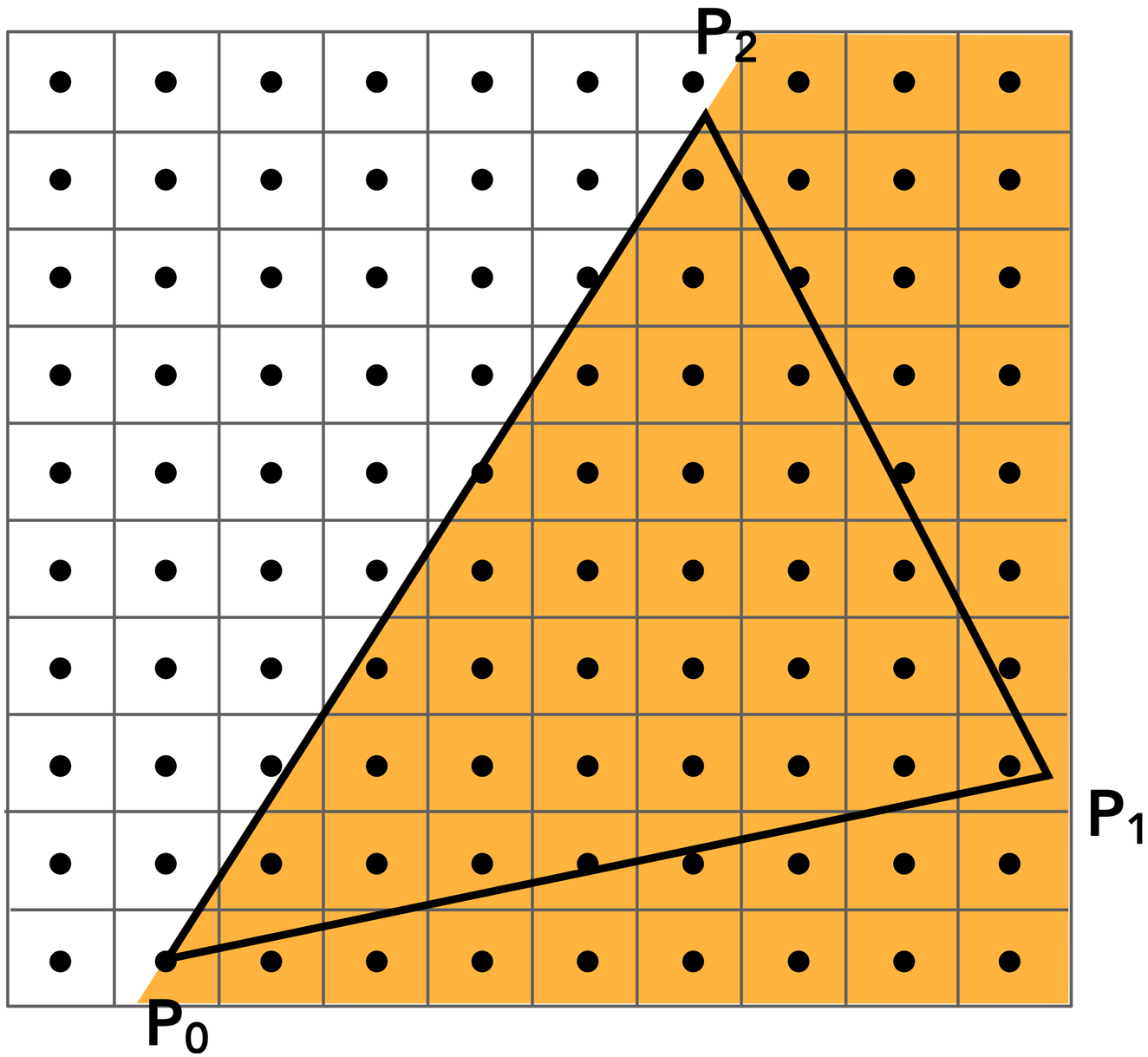
Point-in-Triangle Test: Three Line Tests

$P_i = (X_i, Y_i)$

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

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

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

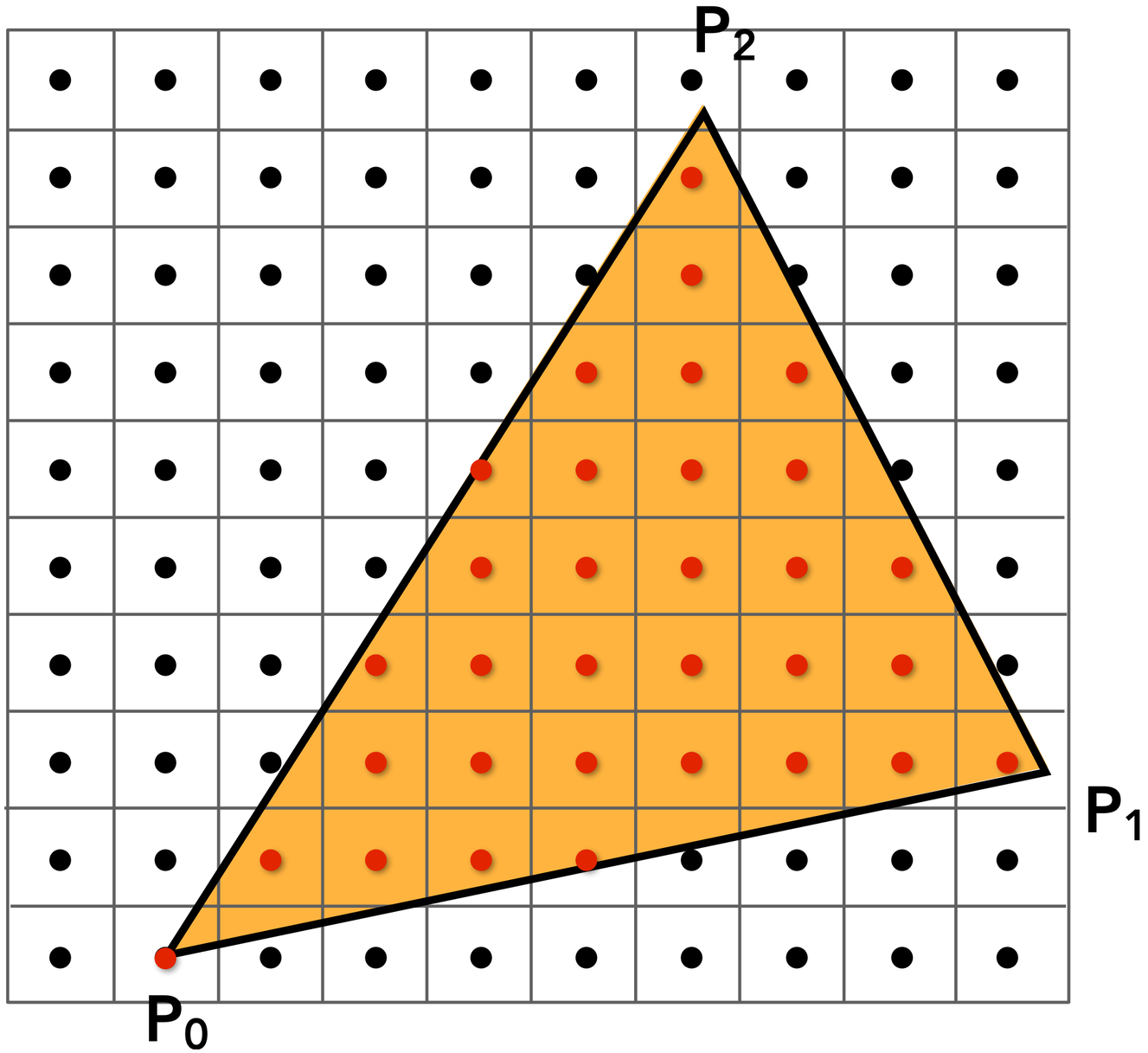


$L_2(x, y) > 0$

Point-in-Triangle Test: Three Line Tests

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

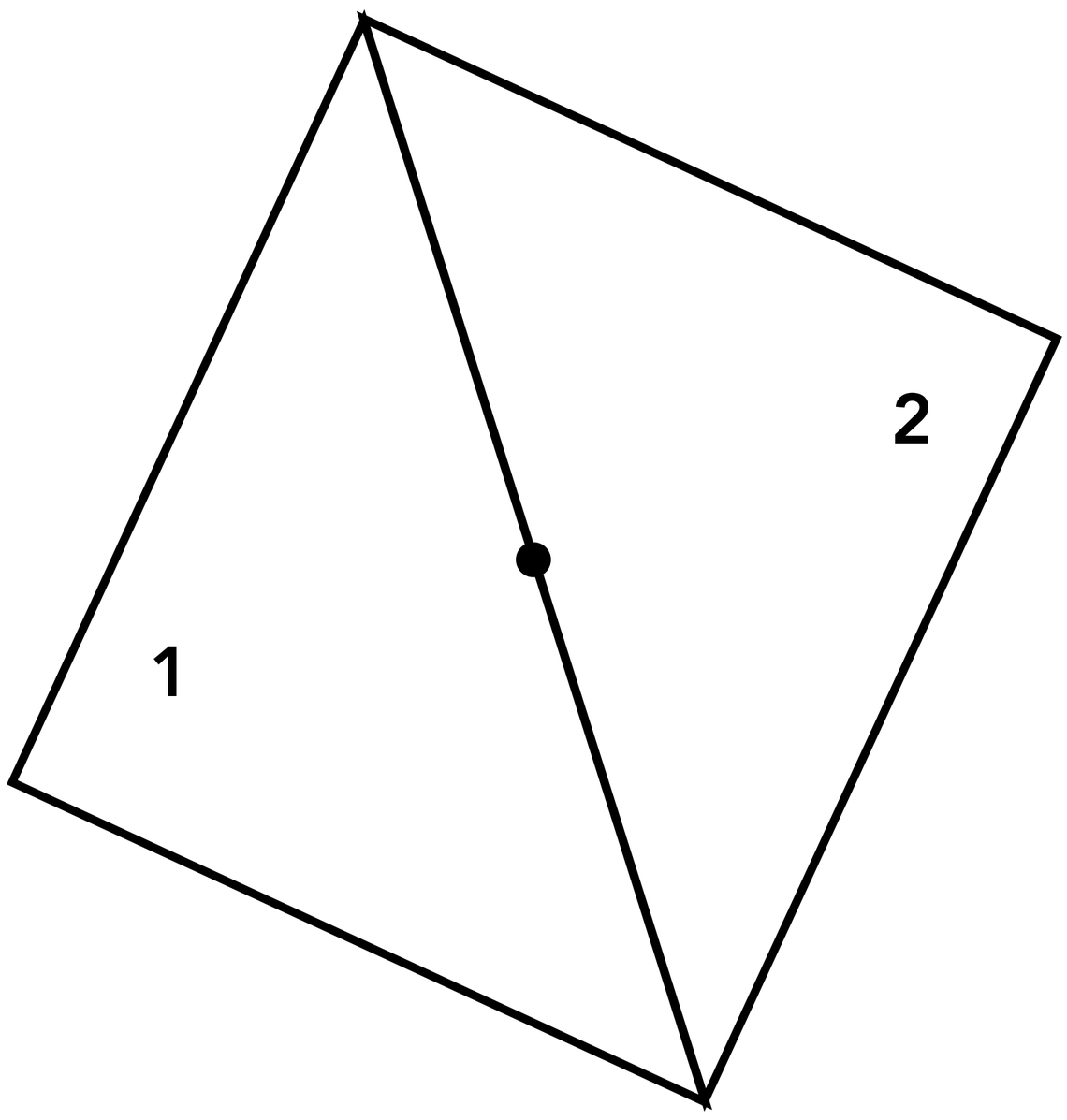
$$inside(sx, sy) =$$
$$L_0(sx, sy) > 0 \;\&\&$$
$$L_1(sx, sy) > 0 \;\&\&$$
$$L_2(sx, sy) > 0;$$



Note: actual implementation of $inside(sx,sy)$ **involves ≤ checks based on edge rules**

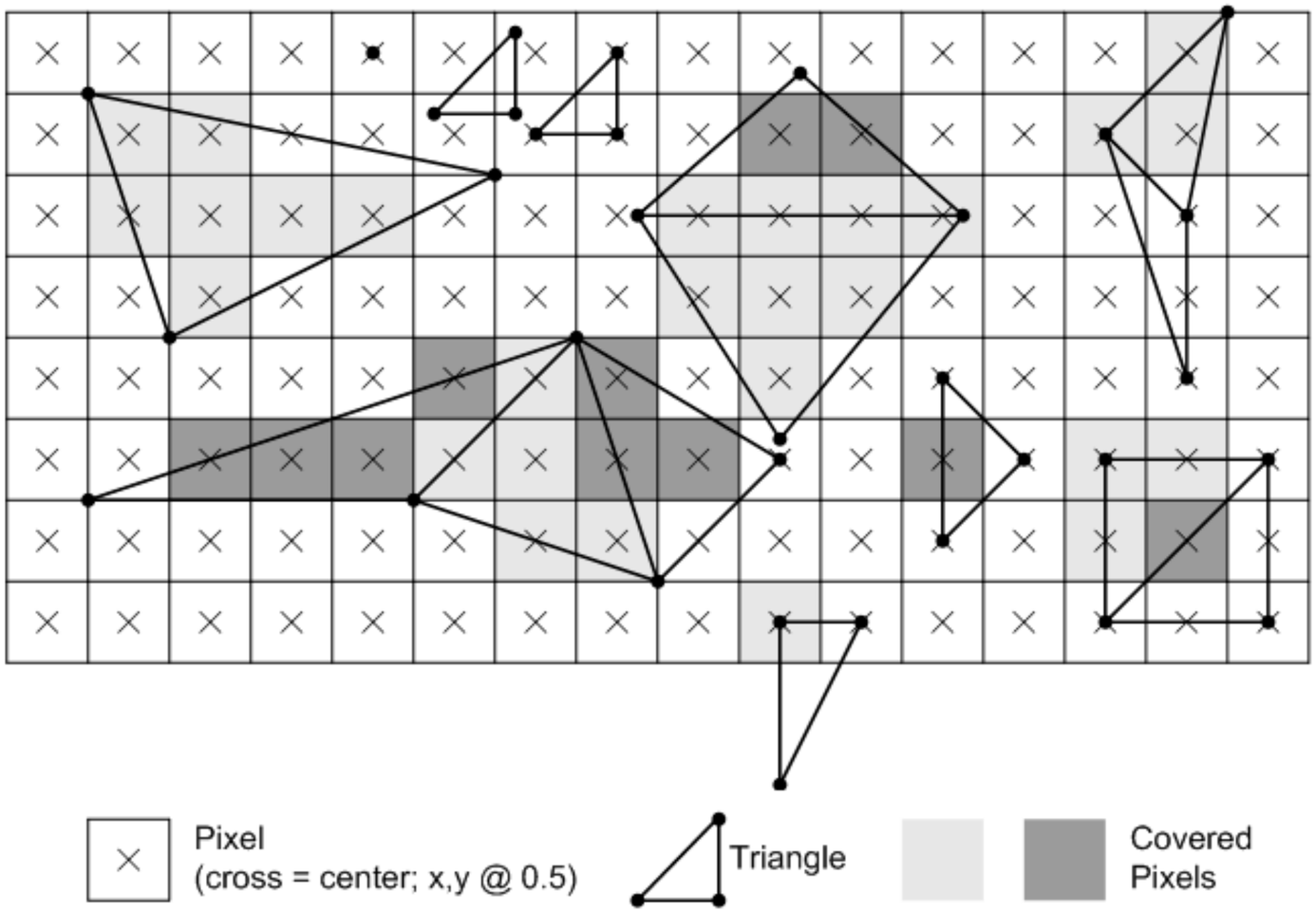
Edge Cases (Literally)

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



OpenGL/Direct3D Edge Rules

When sample point falls on an edge, the sample is classified as within triangle if the edge is a “top edge” or “left edge”

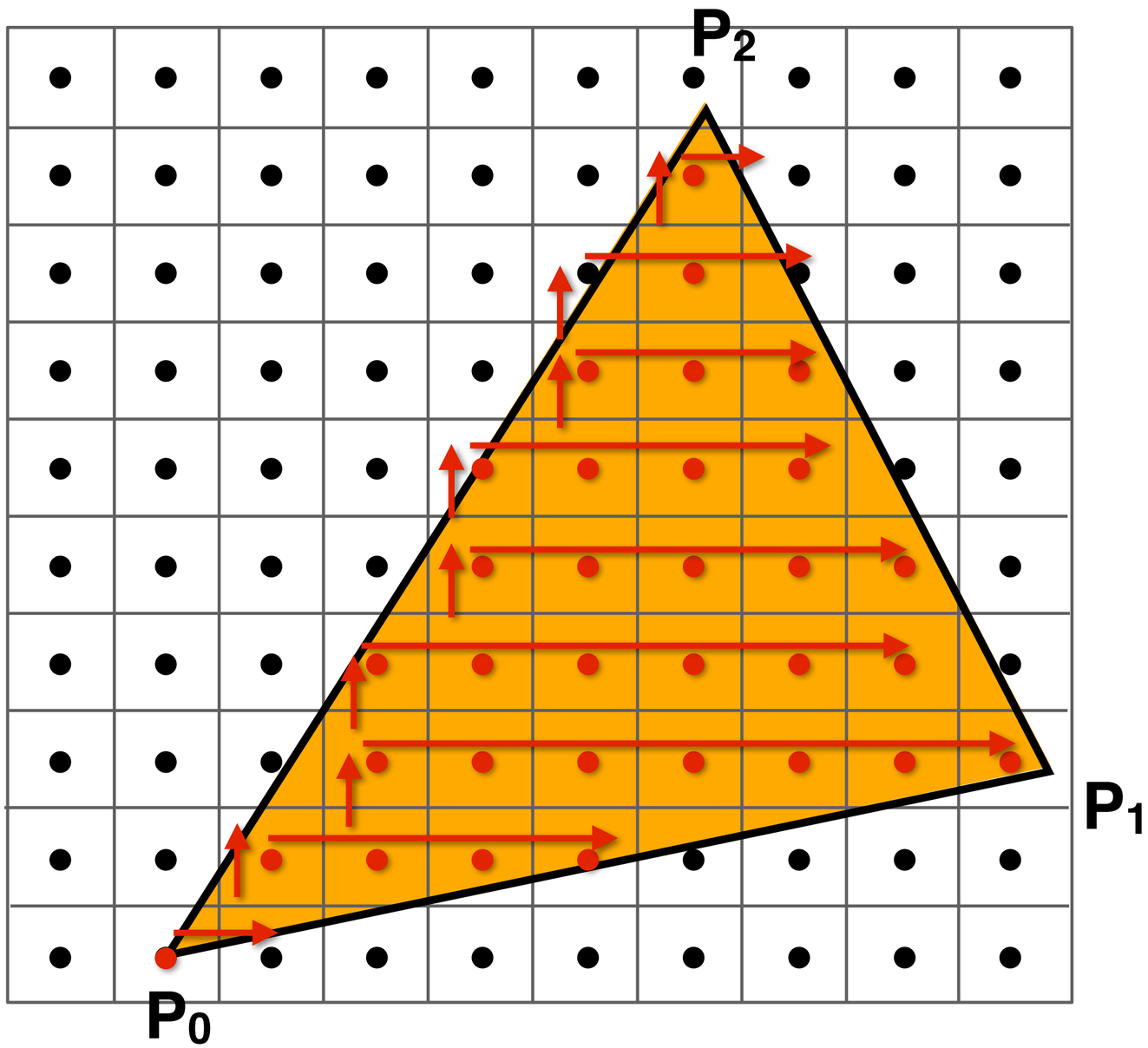


Top edge: horizontal edge that is above all other edges

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

Incremental Triangle Traversal (Faster?)



Modern Approach: Tiled Triangle Traversal

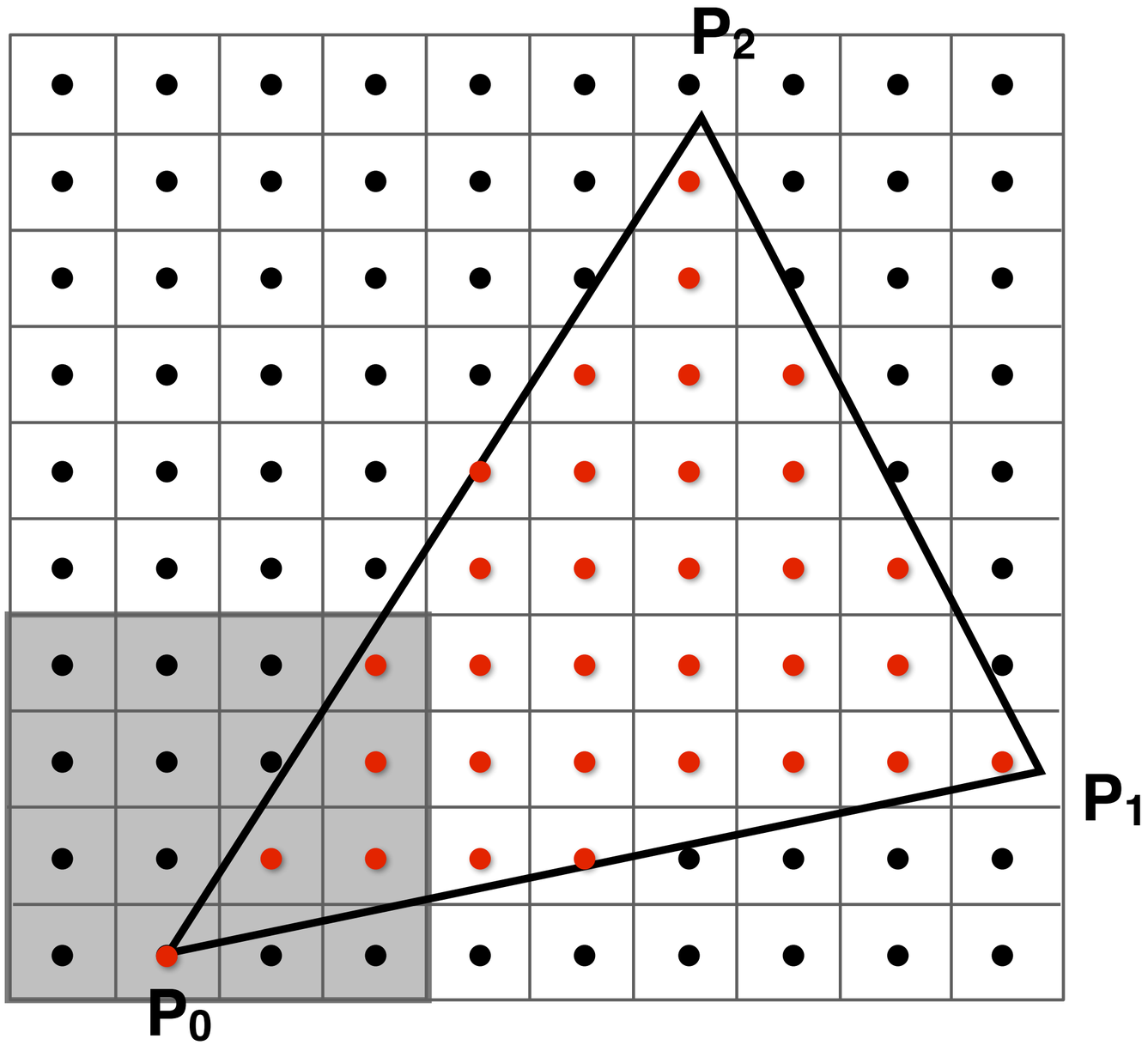
Traverse triangle in blocks

Test all samples in block 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)

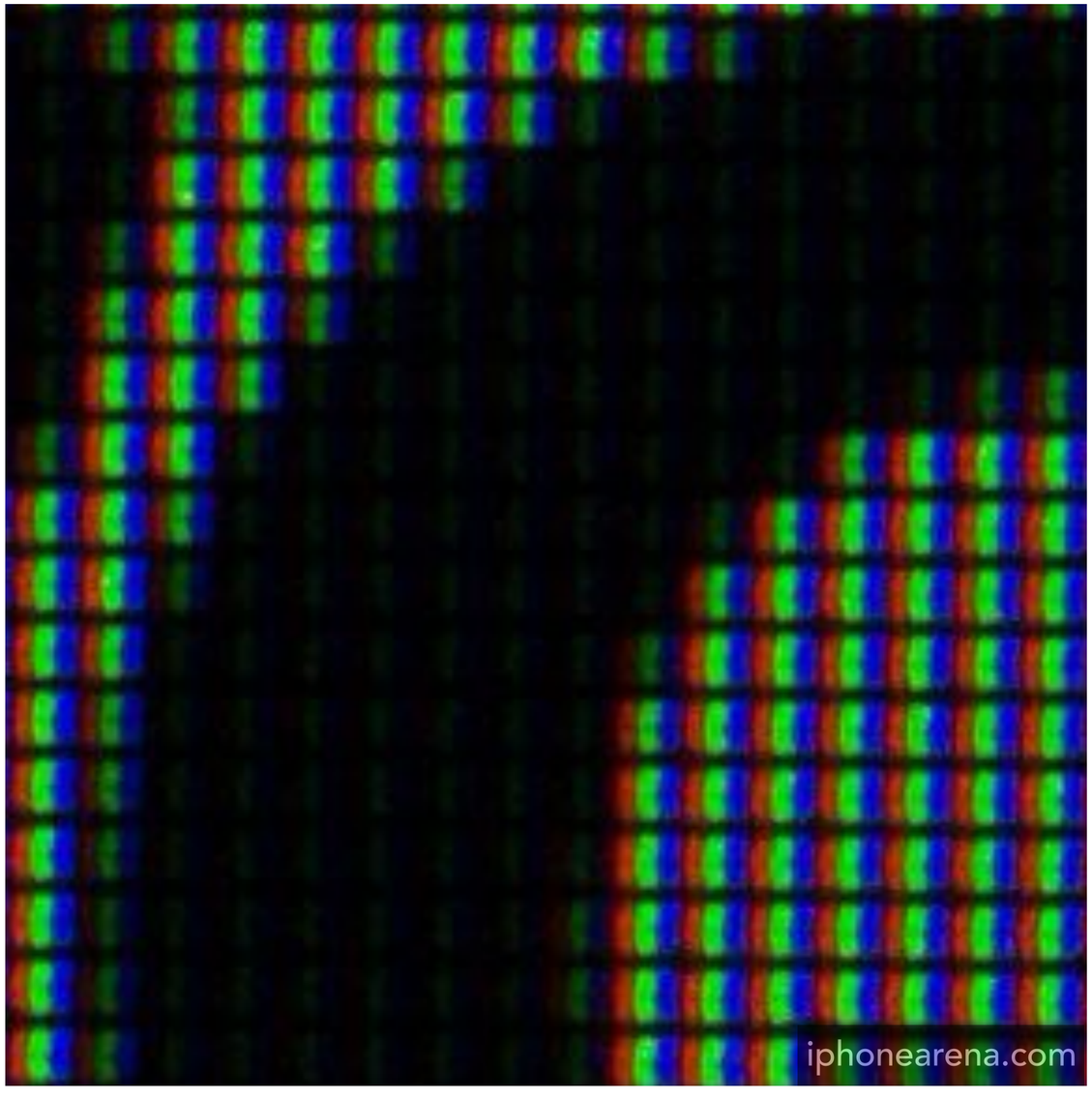
- Can skip sample testing work: entire block not in triangle (“early out”), entire block entirely within triangle (“early in”)



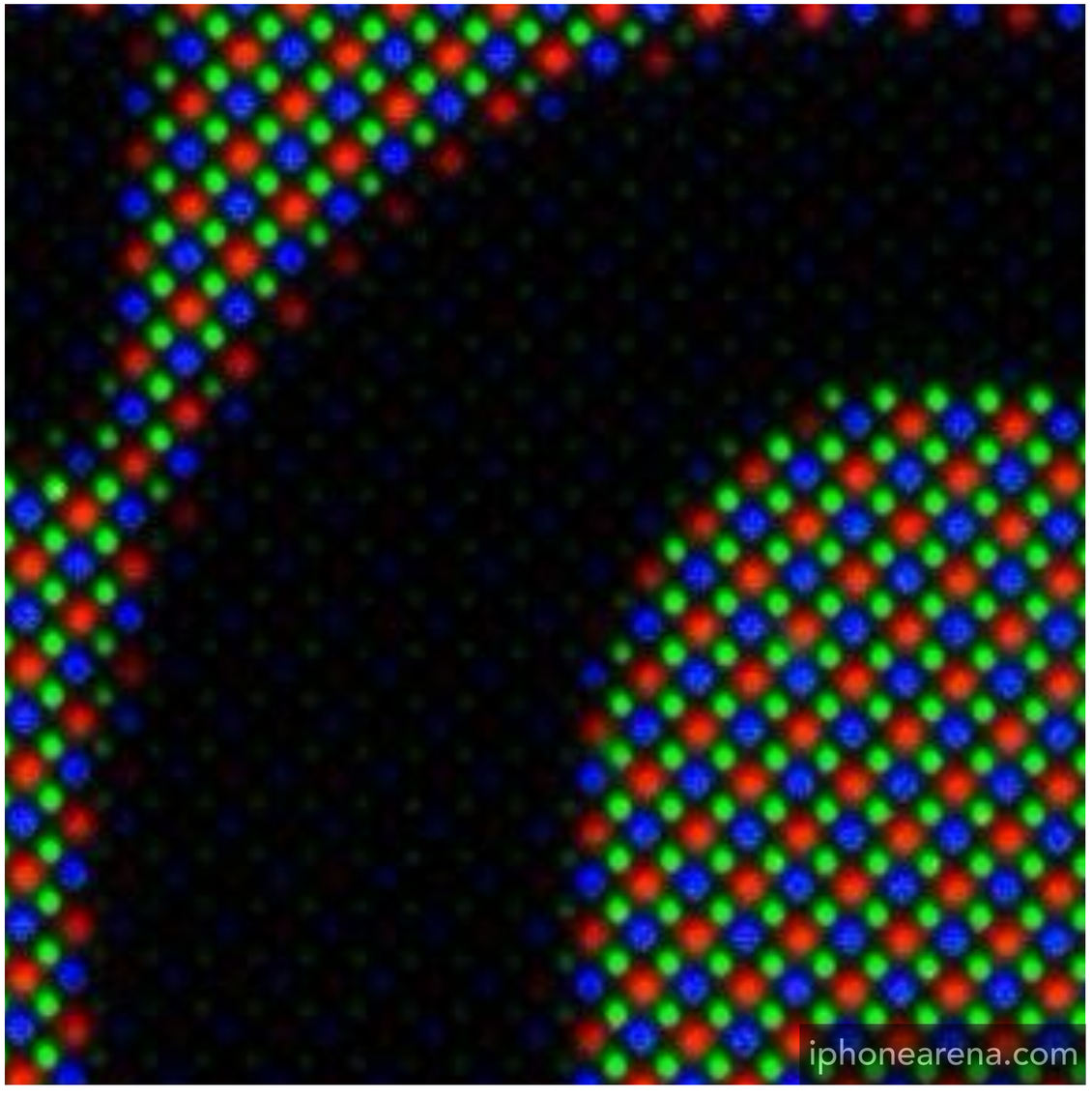
All modern GPUs have special-purpose hardware for efficient point-in-triangle tests

Signal Reconstruction on Real Displays

Real LCD Screen Pixels (Closeup)



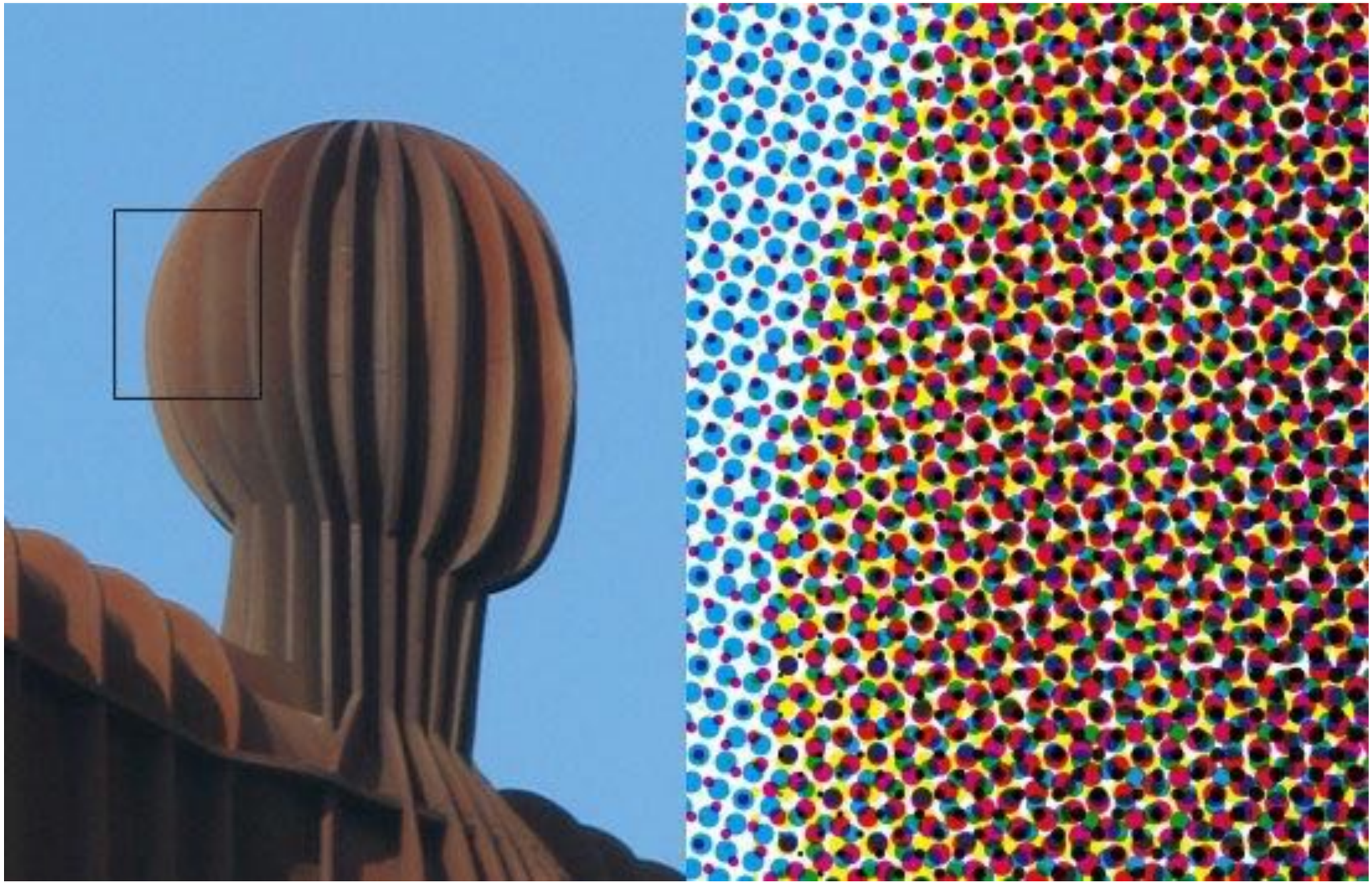
iPhone 6S



Galaxy S5

Notice R,G,B pixel geometry! But in this class, we will assume a colored square full-color pixel.

Aside: What About Other Display Methods?

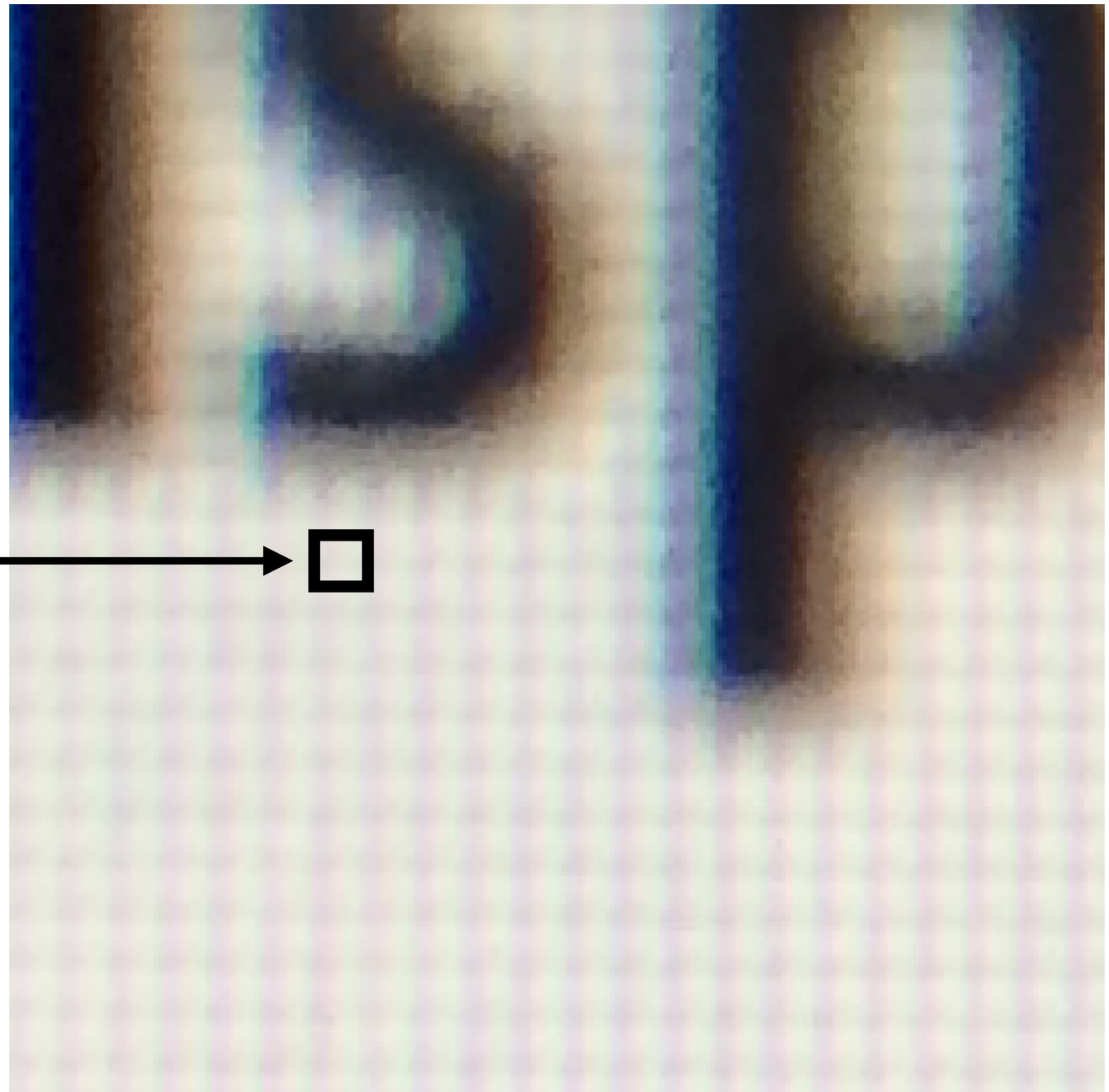


Color print: observe half-tone pattern

Assume Display Pixels Emit Square of Light

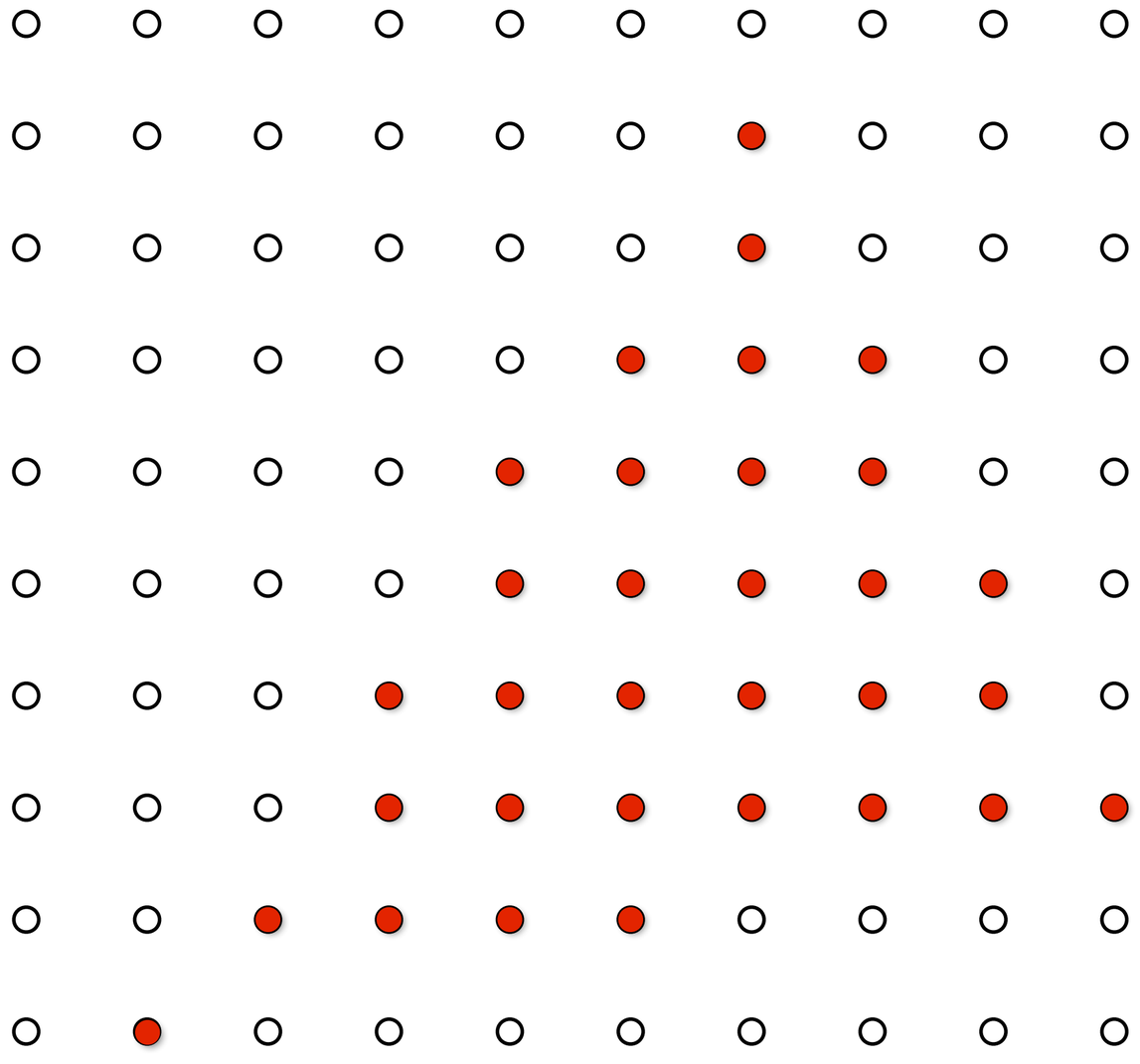
Each image sample sent to the display is converted into a little square of light of the appropriate color:
(a pixel = picture element)

LCD pixel on laptop

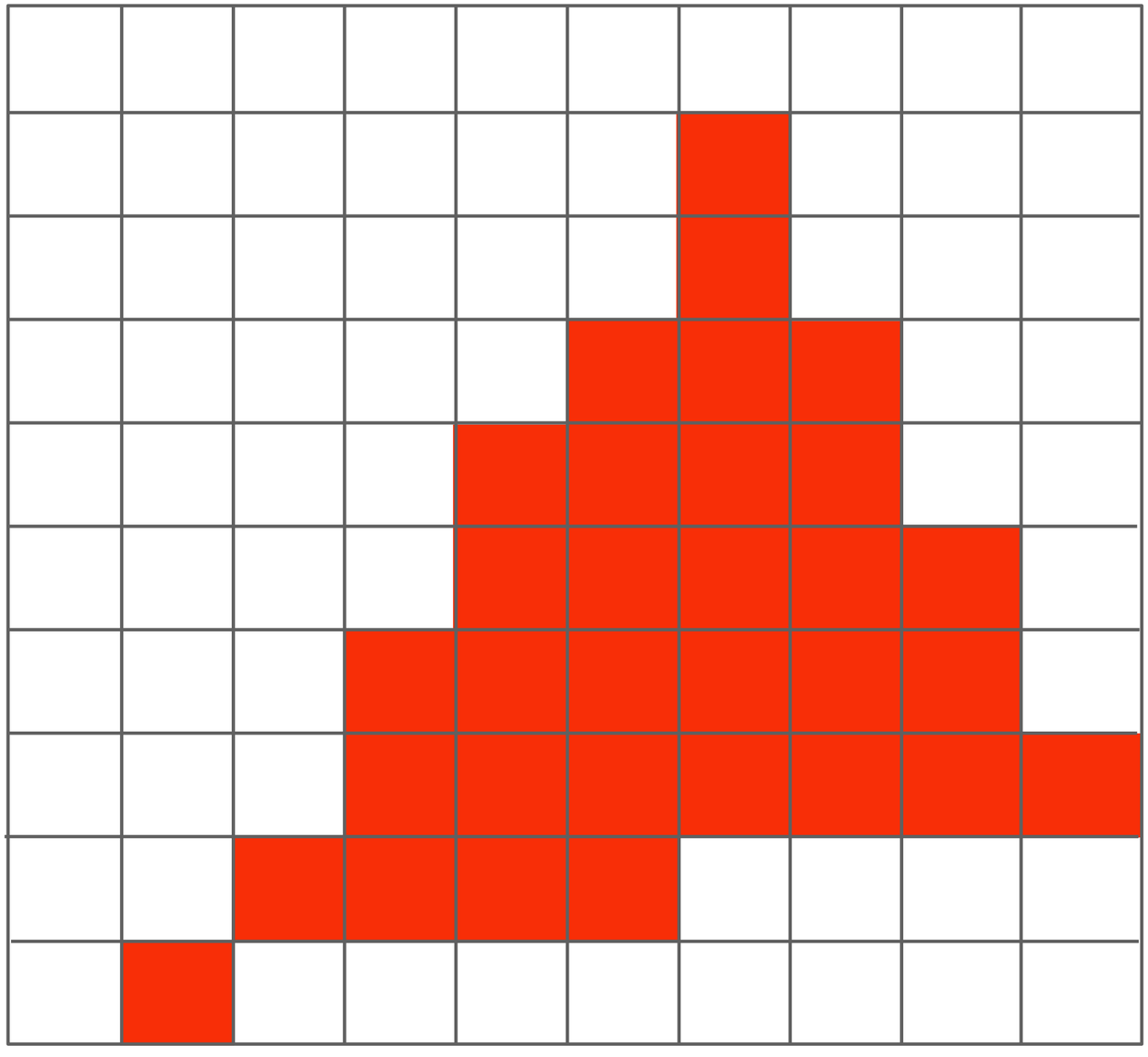


* LCD pixels do not actually emit light in a square of uniform color, but this approximation suffices for our current discussion

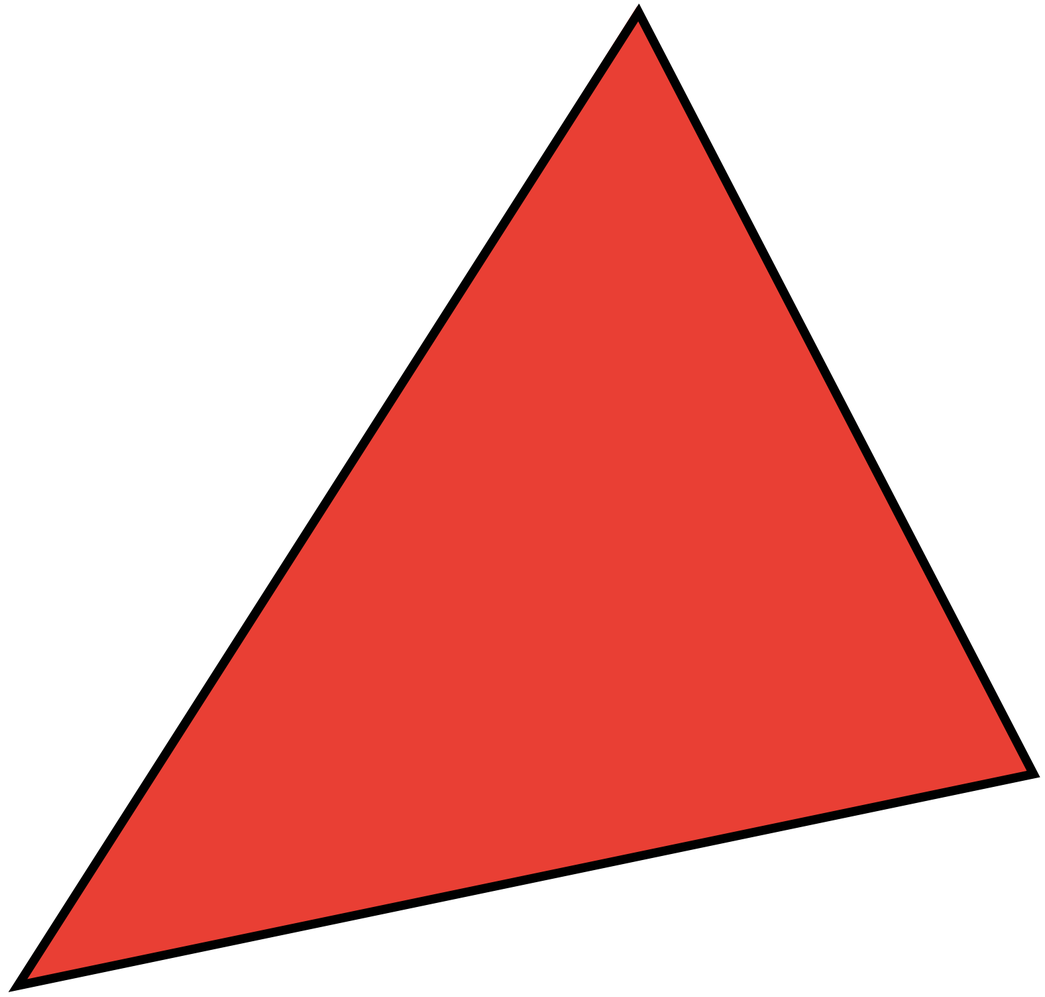
So, If We Send The Display This Sampled Signal



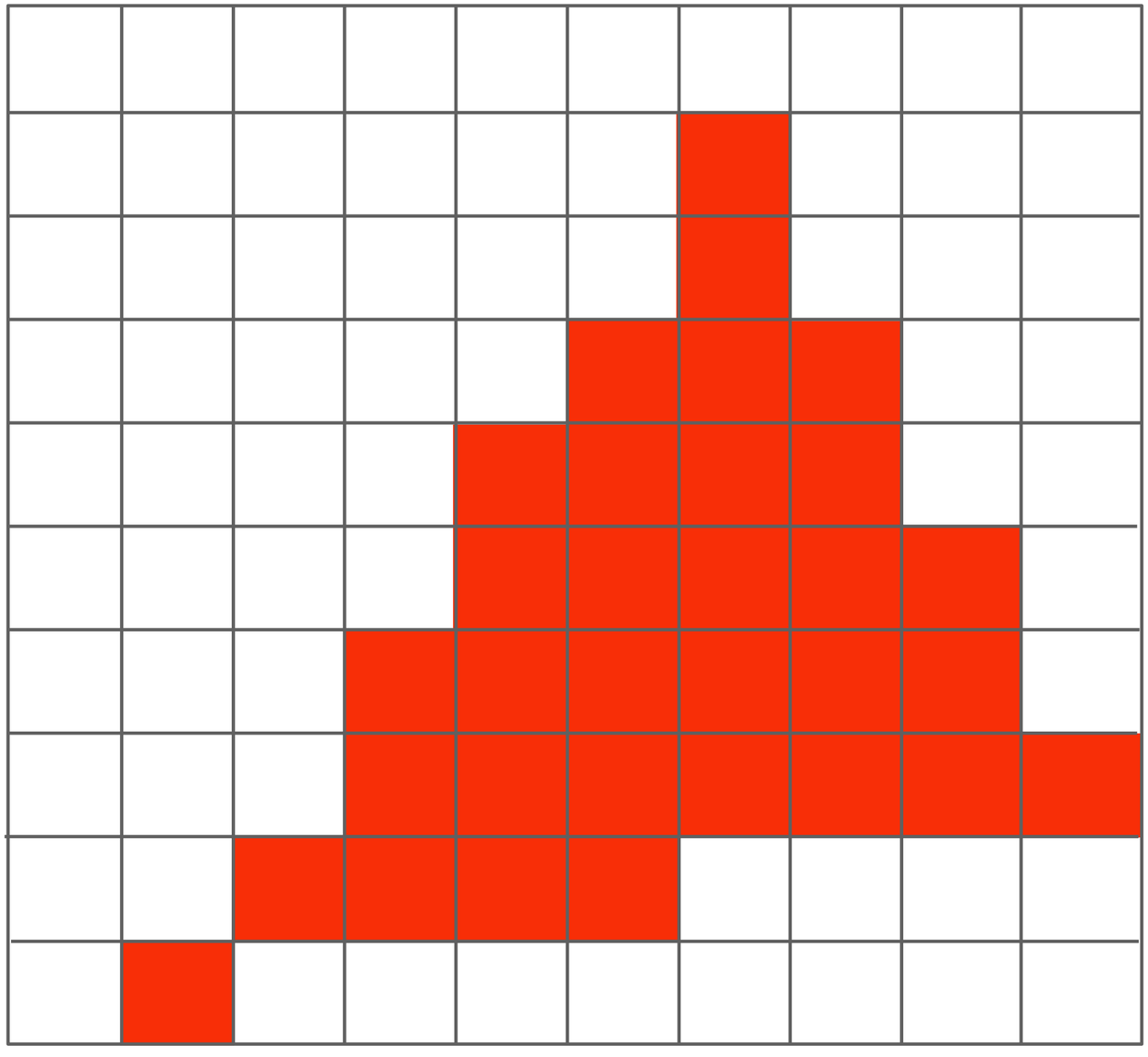
The Display Physically Emits This Signal



Compare: The Continuous Triangle Function

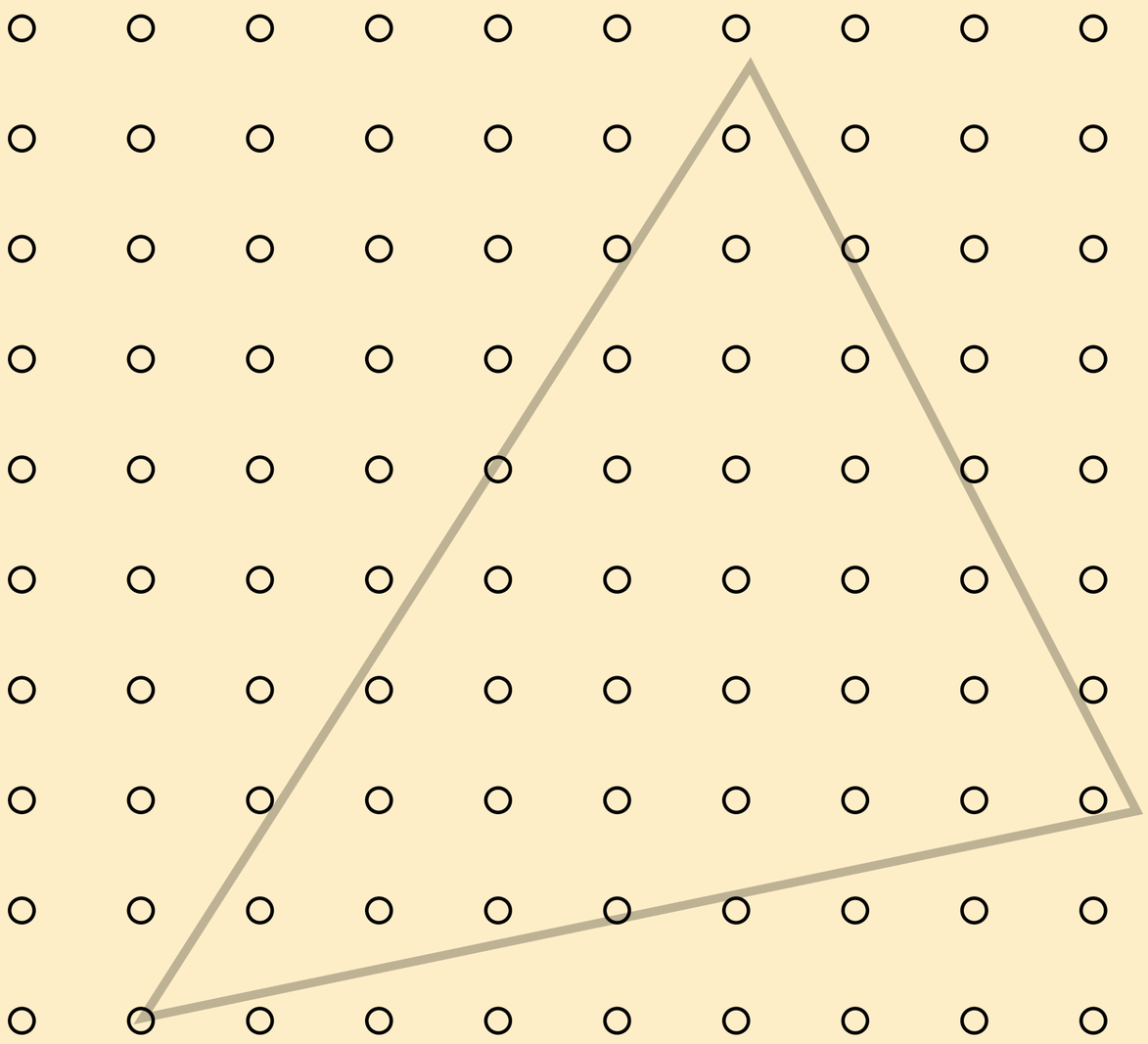


What's Wrong With This Picture?



Jaggies!

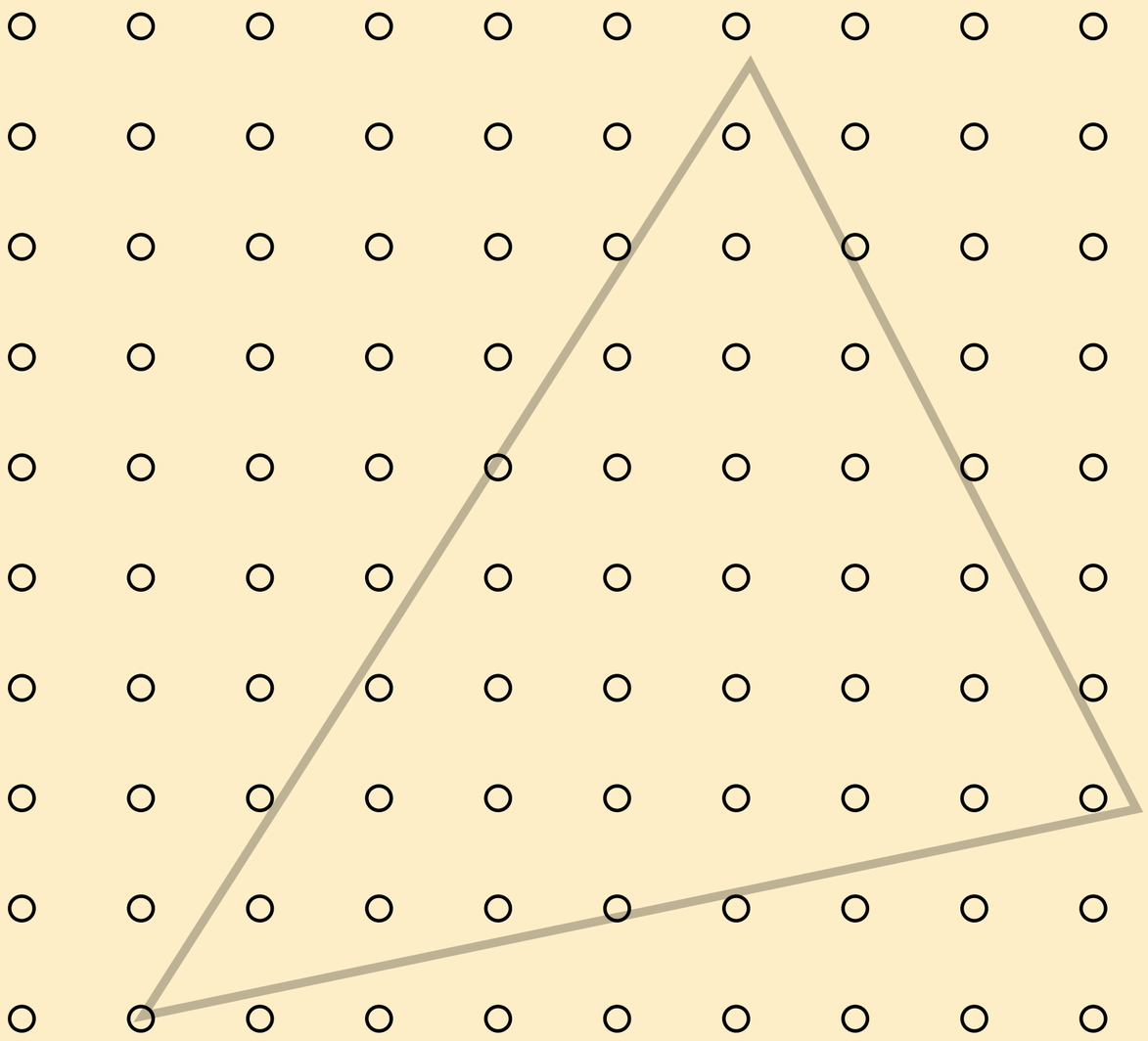
Discussion: What Value Should a Pixel Have?



Think about:

- Ideas for “higher quality” pixel formula?
- What are all the relevant factors?
- What’s right/wrong about point sampling?
- Why do jaggies look “wrong”?

Discussion: What Value Should a Pixel Have?

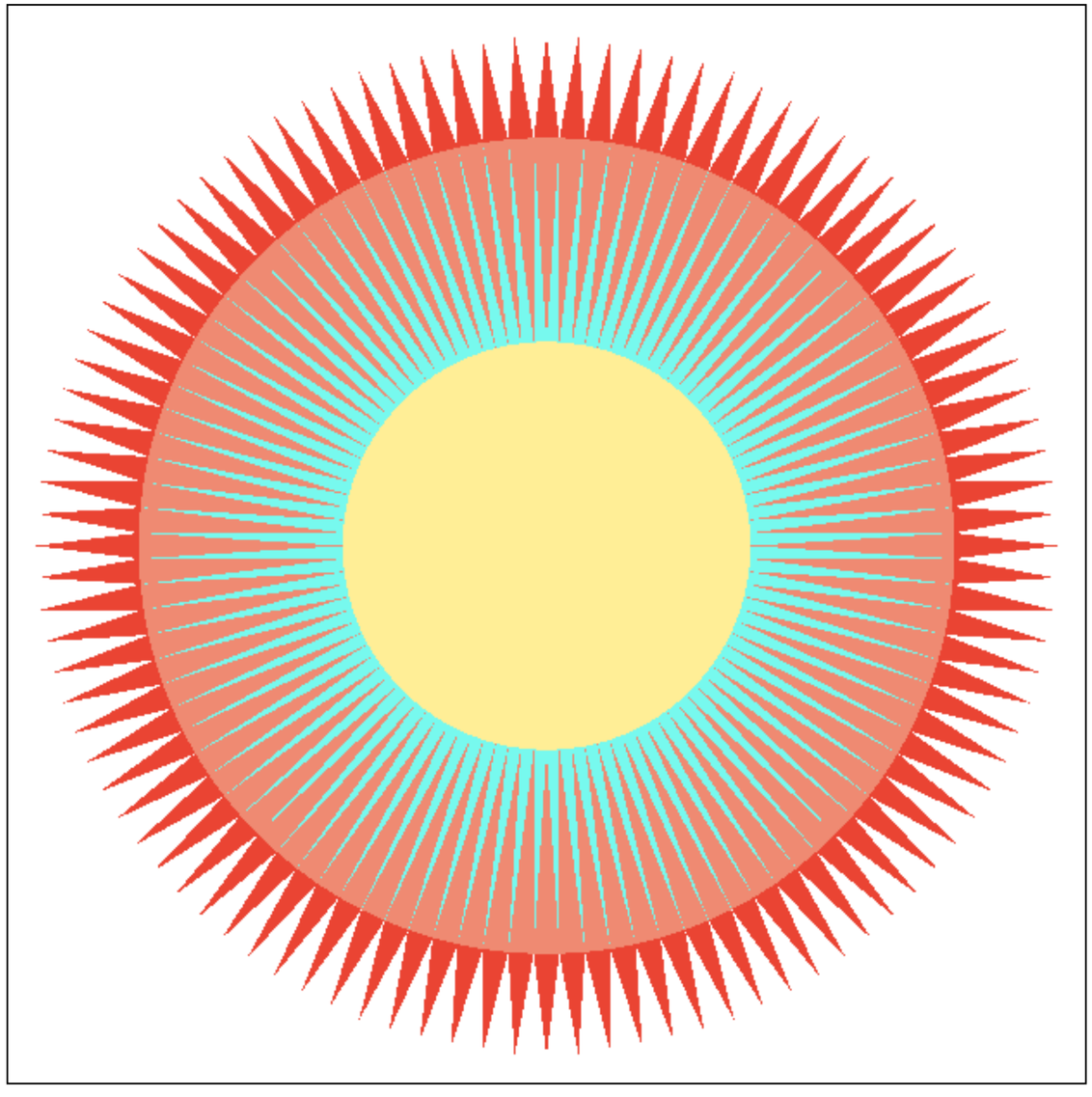


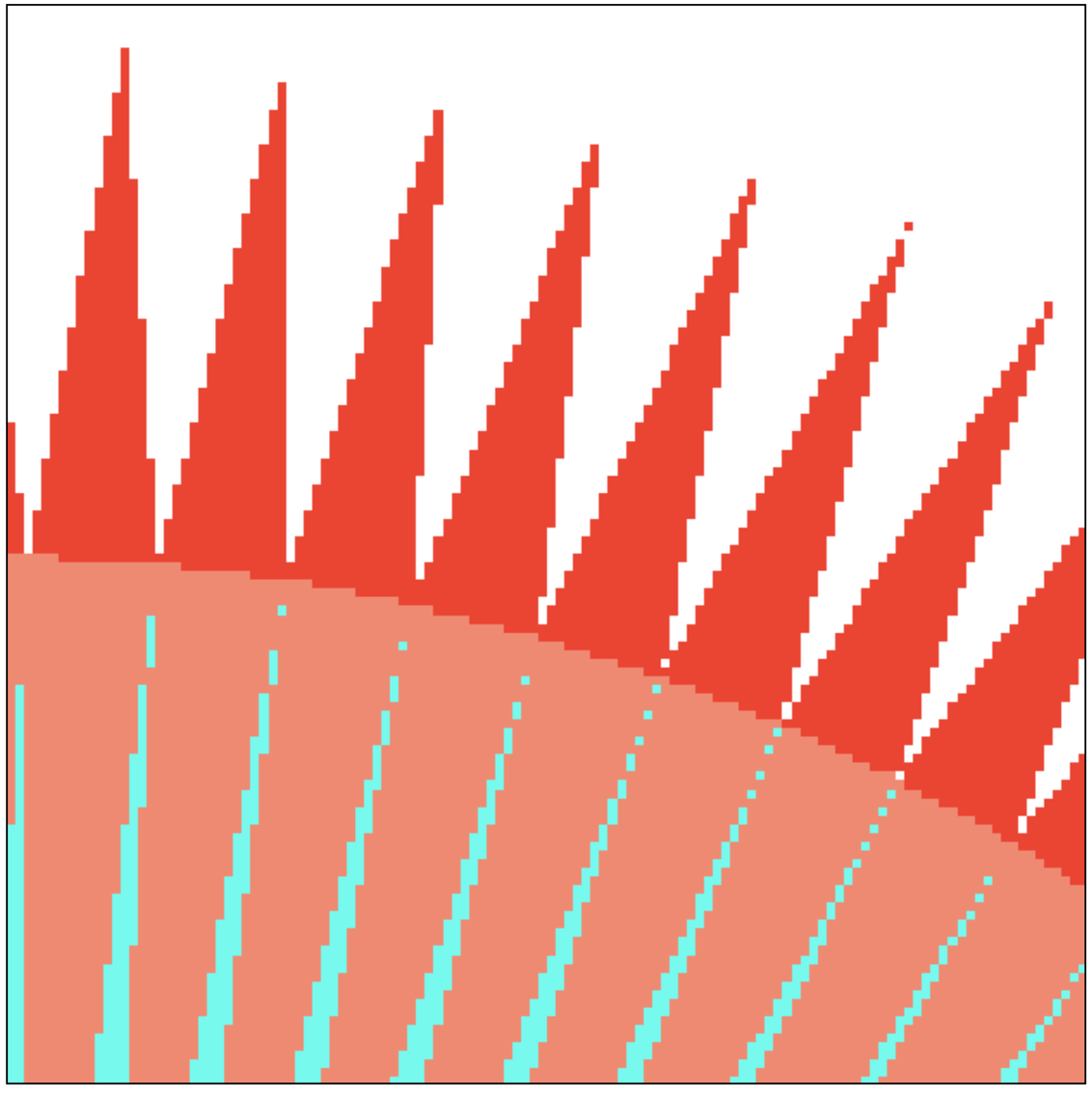
Filling any pixel that connects with the shape instead of checking if its center is inside

Color should be average color of the area the pixel occupies analytically (but that's complex, so sample more points in the pixel area to get a better average)

Having a higher pixel count would lead to less severe jaggies

Jaggies (Staircase Pattern)





Is this the best we can do?

Things to Remember

Drawing machines

- Many possibilities
- Why framebuffers and raster displays?
- Why triangles?

We posed rasterization as a 2D sampling process

- Test a binary function `inside(triangle,x,y)`
- Evaluate triangle coverage by 3 point-in-edge tests
- Finite sampling rate causes “jaggies” artifact (next time we will analyze in more detail)

Acknowledgments

Thanks to Kayvon Fatahalian, Pat Hanrahan, Mark Pauly and Steve Marschner for slide resources.

Many thanks to Ren Ng for lecture slides and pre-recorded lectures!