Lecture 2:

Digital Drawing

Computer Graphics 2025

Fuzhou University - Computer Science

Today's Topics

Drawing Machines

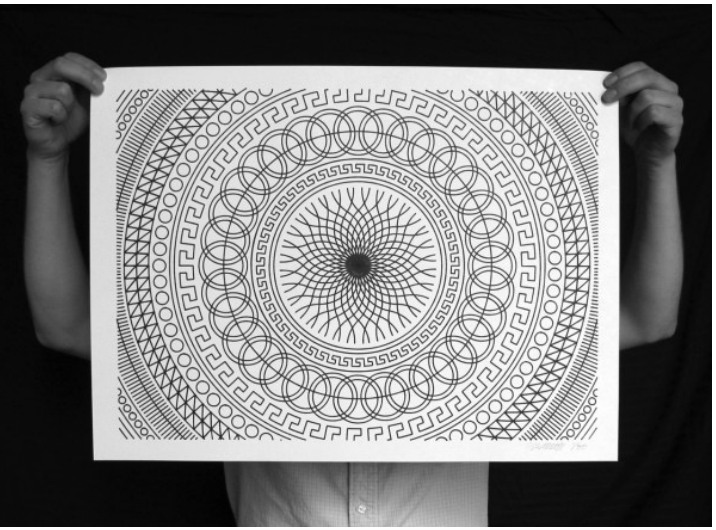
Drawing Triangles to Raster Displays

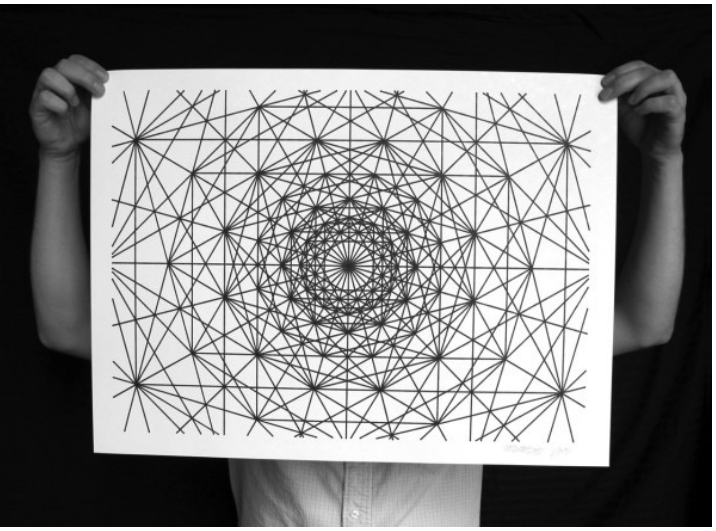
Signal Reconstruction on Real Displays

Sampling and Aliasing

Drawing Machines

CNC Sharpie Drawing Machine

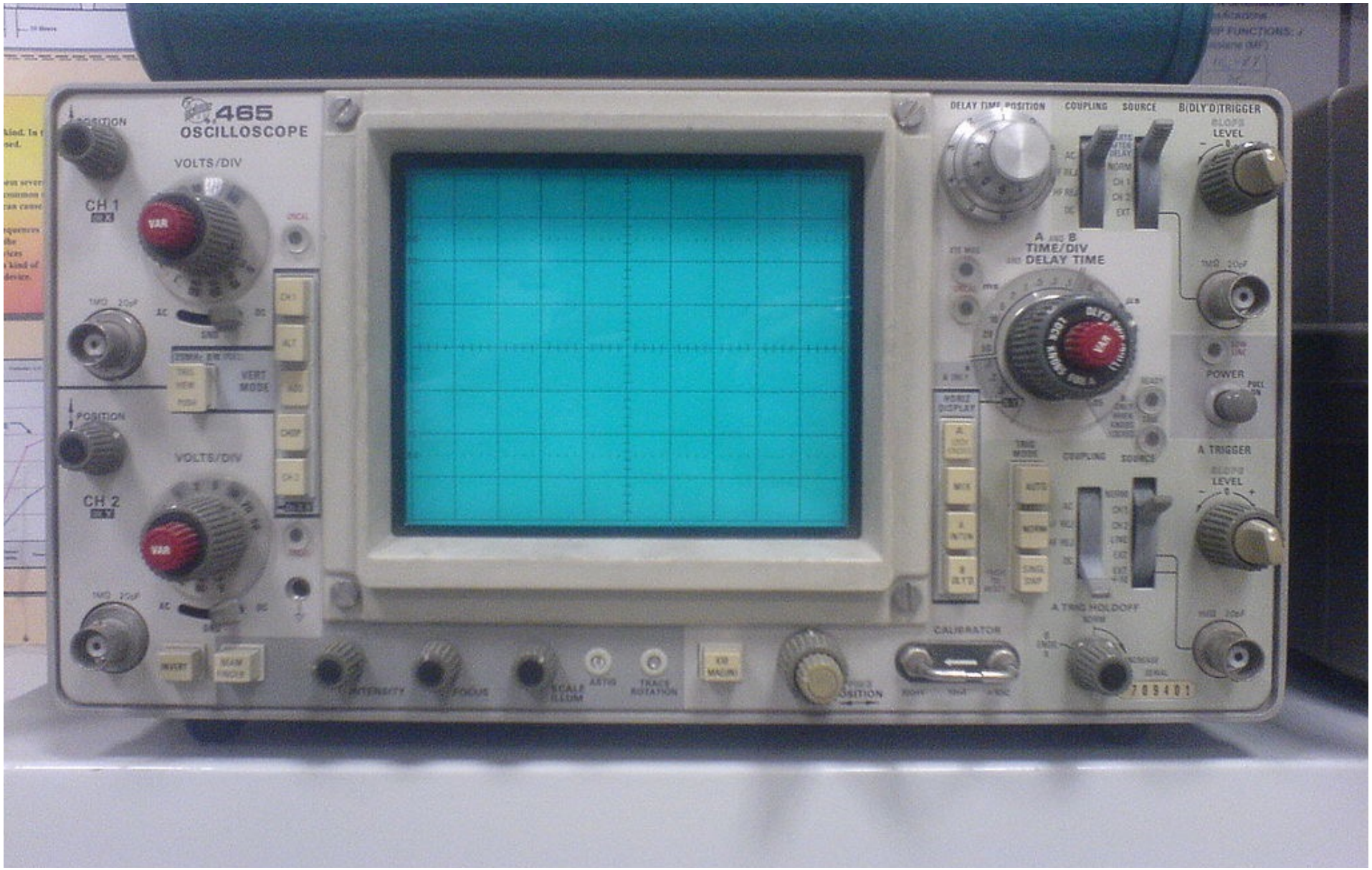




Aaron Panone with Matt W. Moore

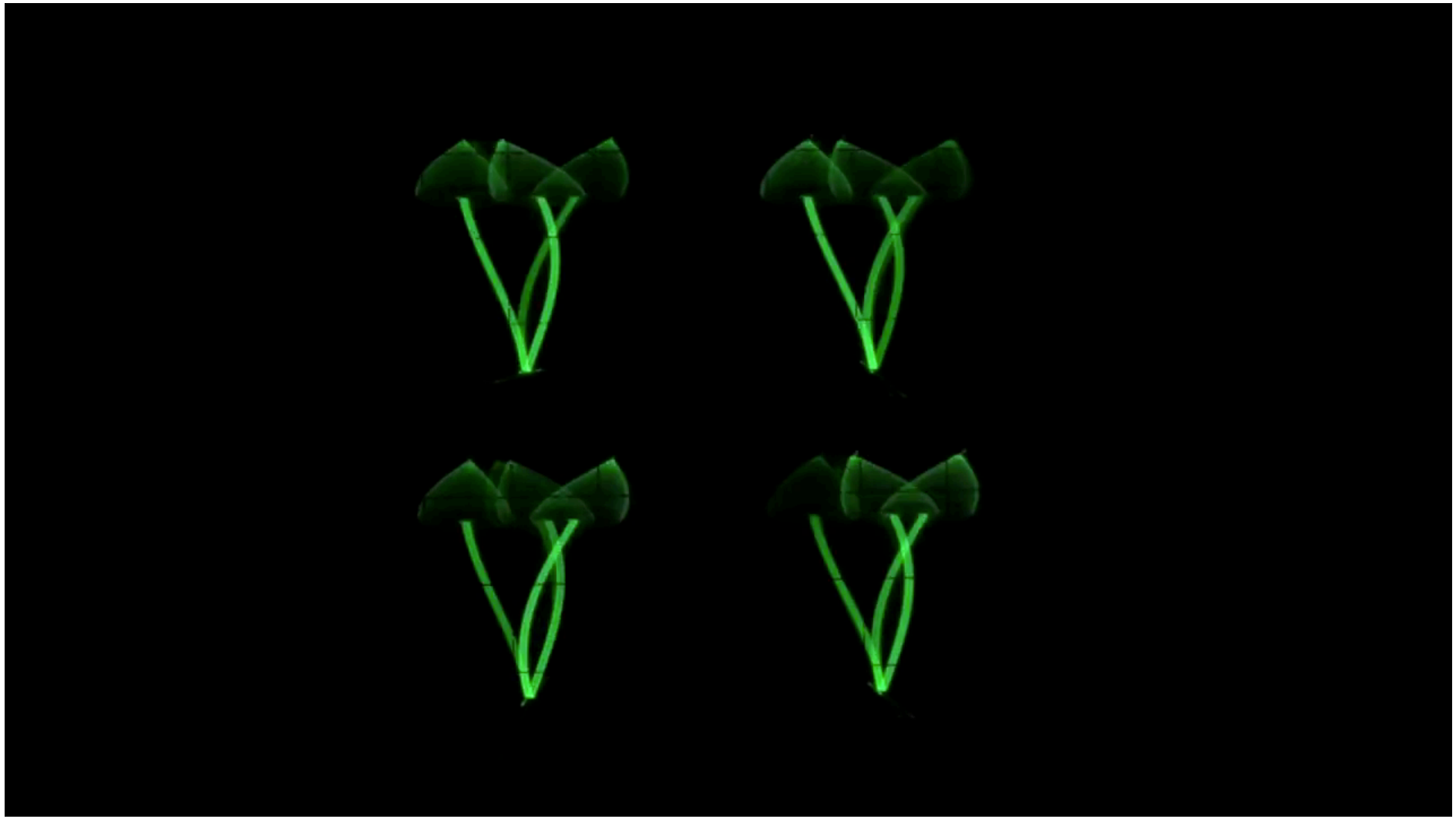
http://44rn.com/projects/numerically-controlled-poster-series-with-matt-w-moore/

Oscilloscope



示波器

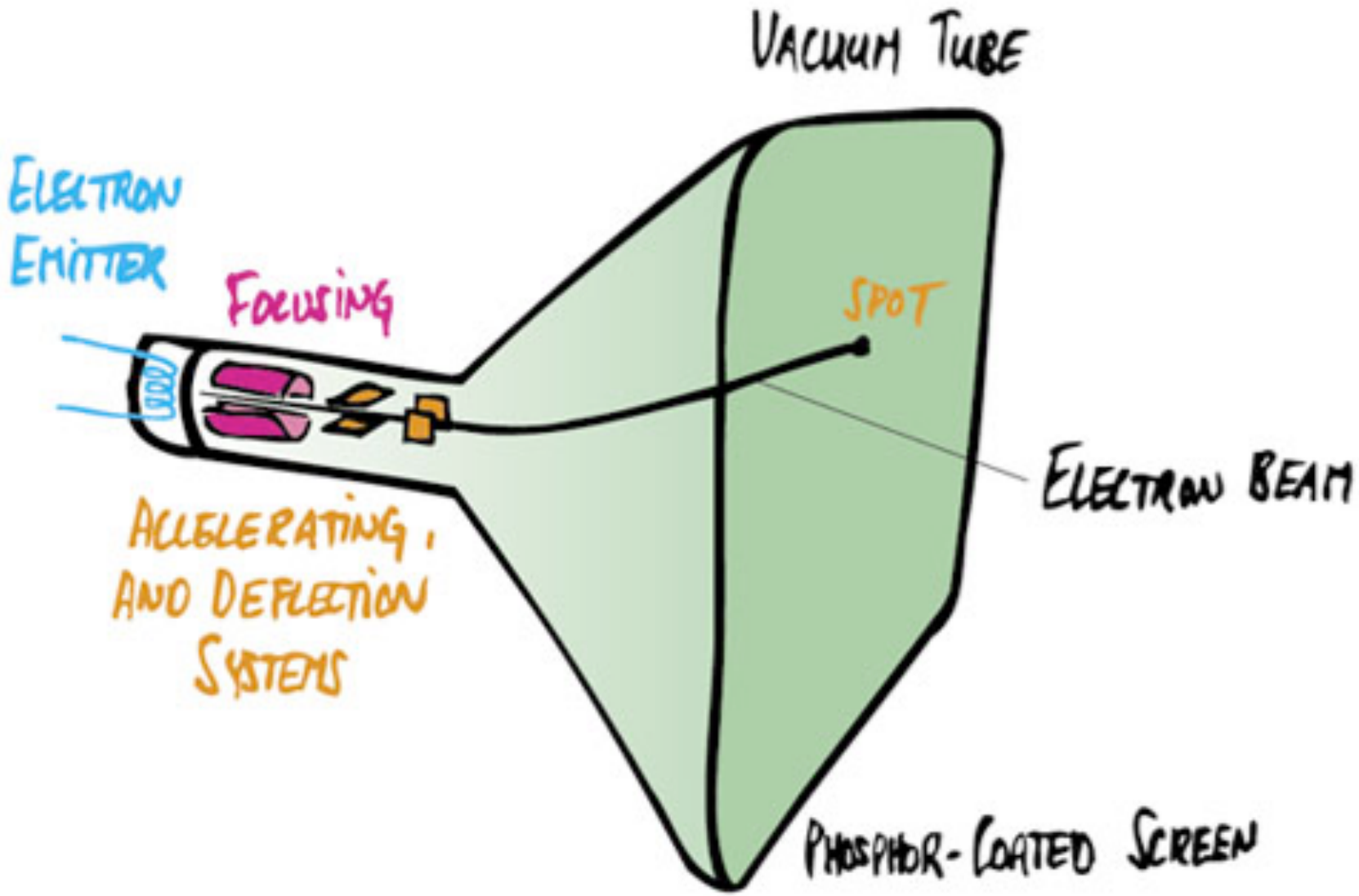
Oscilloscope Art



Jerobeam Fenderson

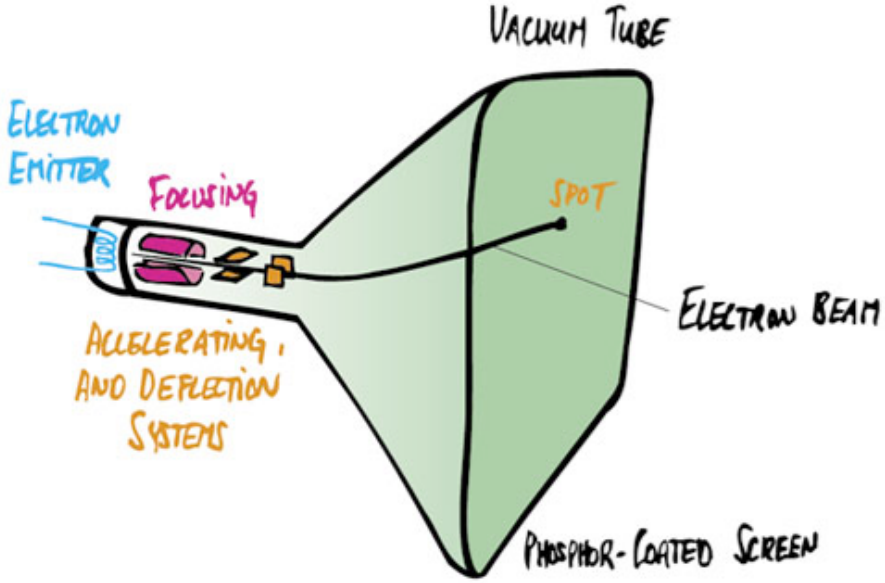
https://www.youtube.com/watch?v=rtR63-ecUNo

Cathode Ray Tube

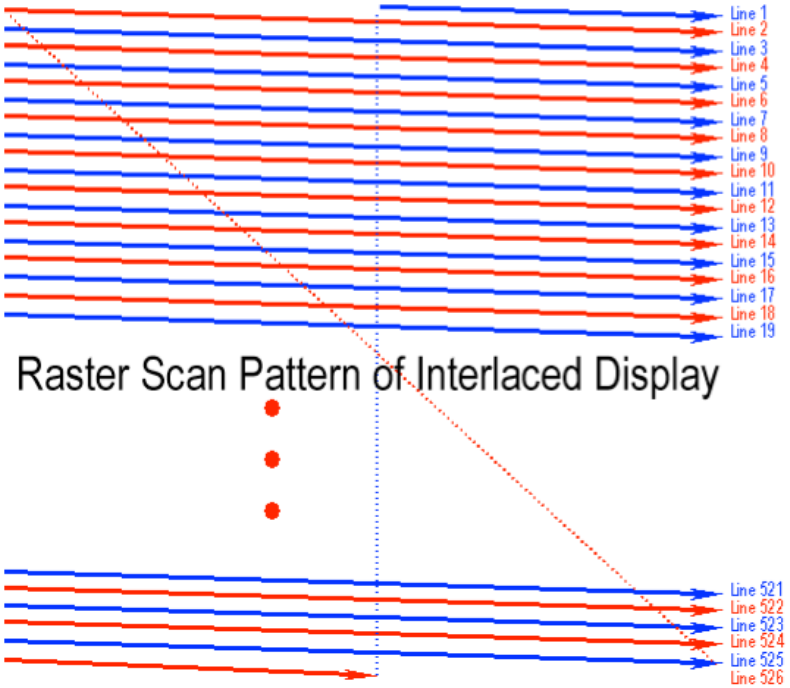


CRT: 阴极射线管

Television - Raster Display CRT



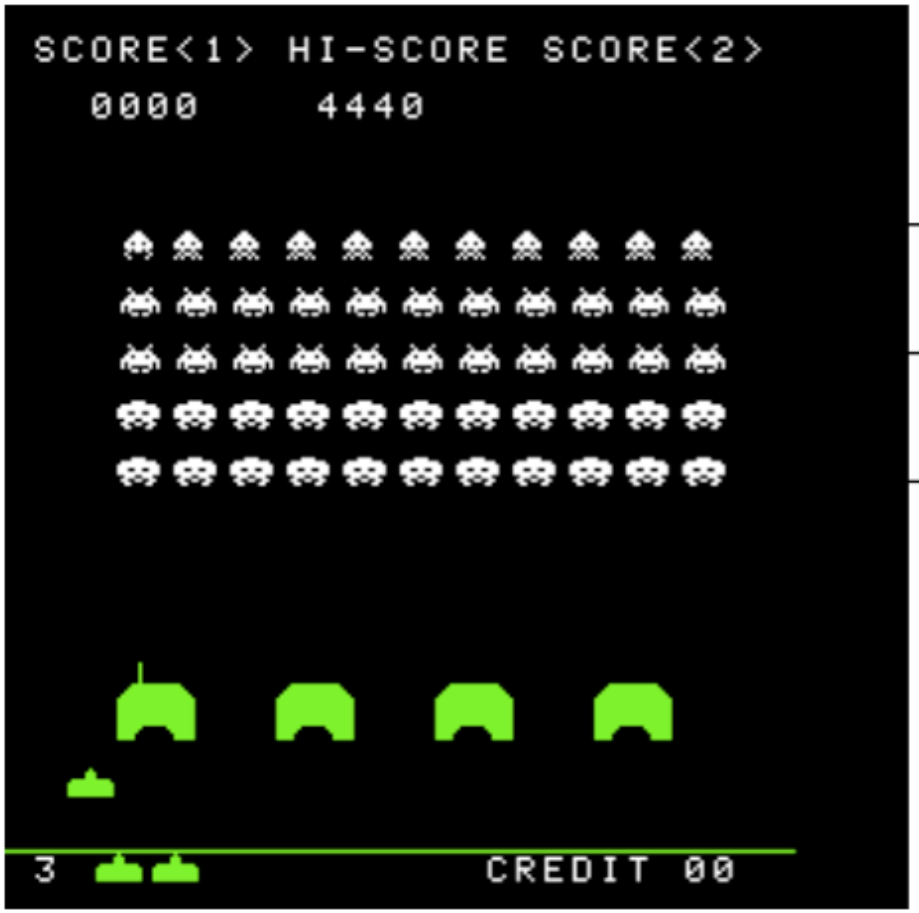
Cathode Ray Tube



Raster Scan
(modulate intensity)

光栅扫描

Frame Buffer: Memory for a Raster Display



DAC =
Digital to Analog Convertors

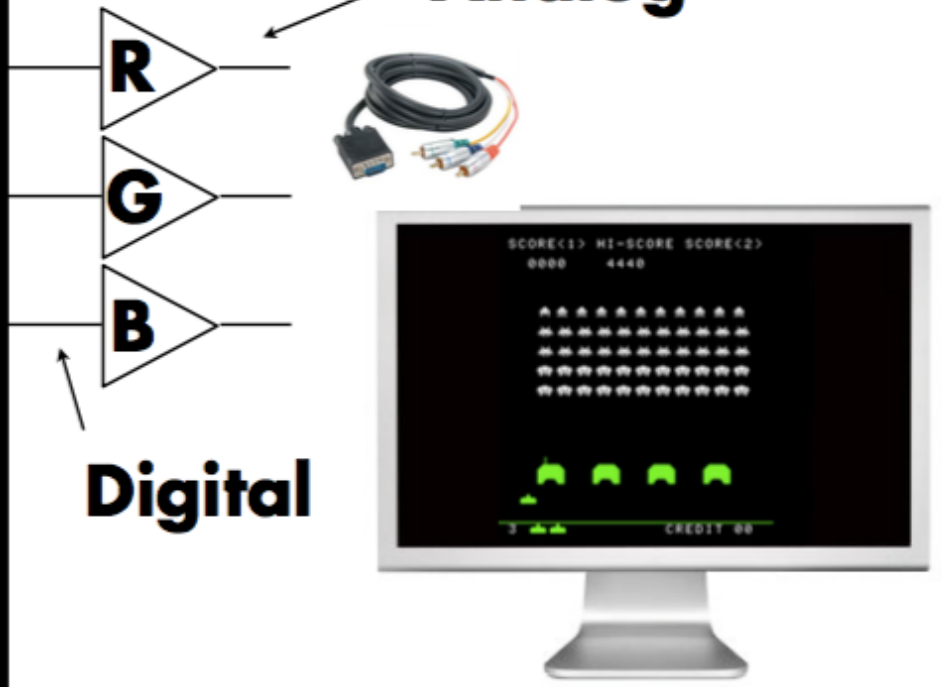


Image = 2D array of colors

A Sampling of Different Raster Displays

Flat Panel Displays

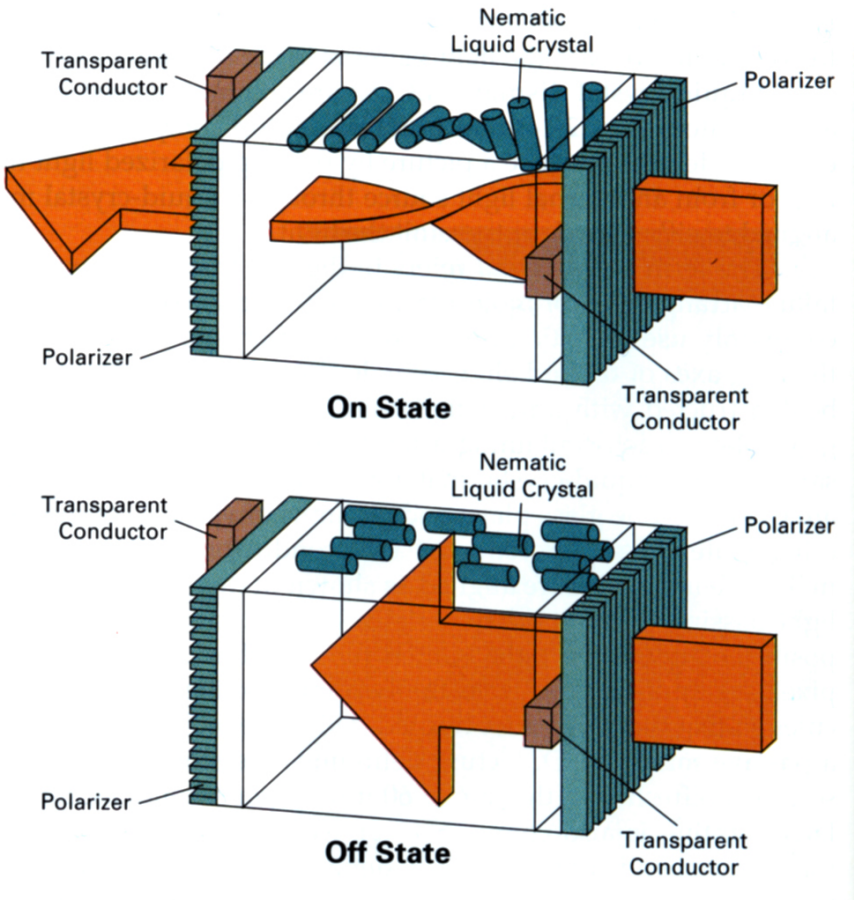
Low-Res LCD Display

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

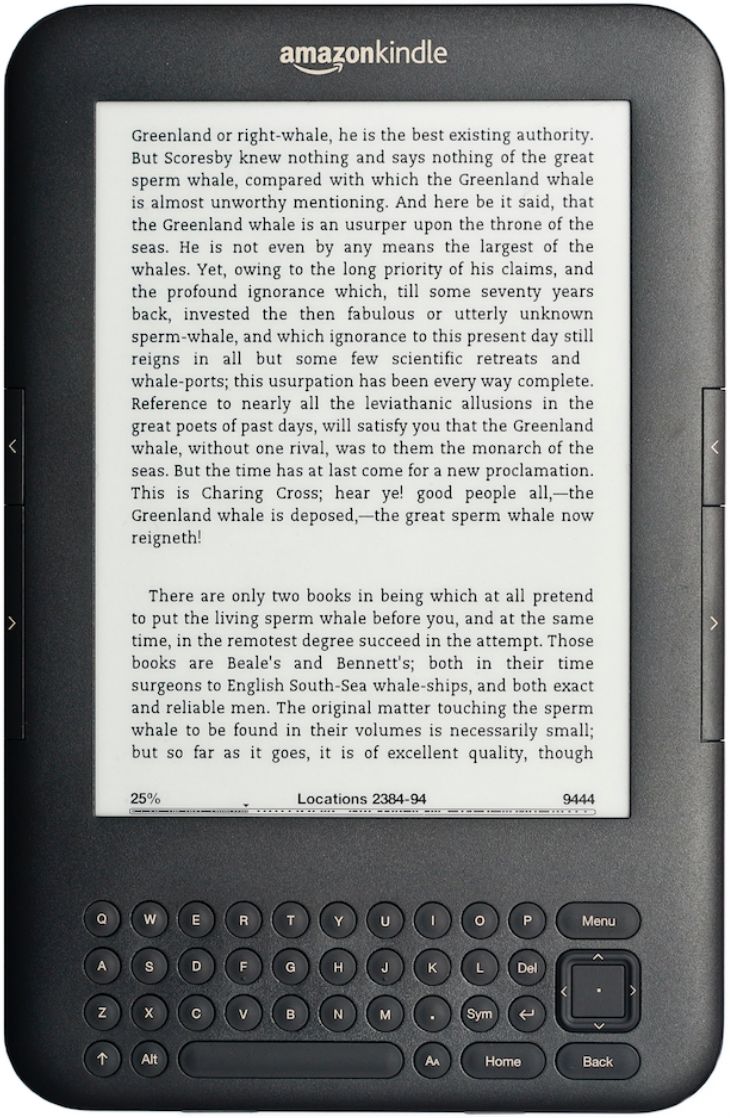


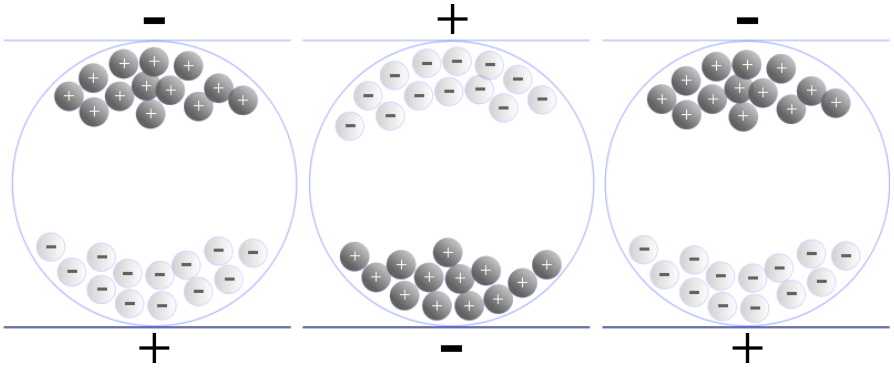
[H&B fig. 2-16]

LED Array Display

BAMPFA: LED Array Display

Electrophoretic (Electronic Ink) Display





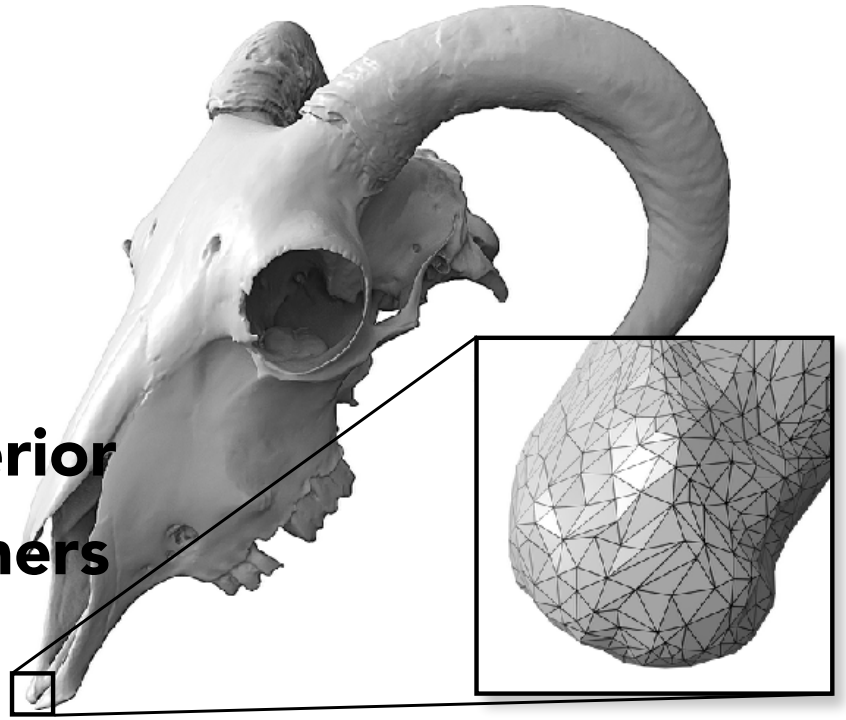
[Wikimedia Commons —Senarclens]

Drawing A Triangle to Raster Display

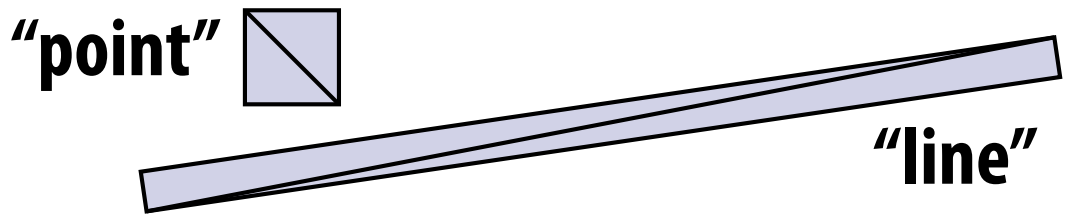
Polygon Meshes

Why triangles?

- Rasterization pipeline converts all primitives to triangles
  - even points and lines!
- Why?
  - can approximate any shape
  - always planar, well-defined interior
  - easy to interpolate data at corners
    - "barycentric coordinates"

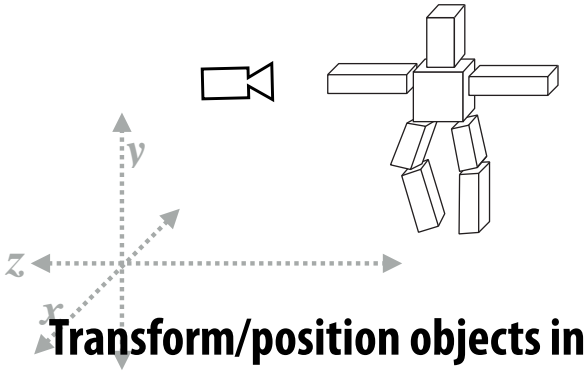


- Key reason: once everything is reduced to triangles, can focus on making an extremely well-optimized pipeline for drawing them

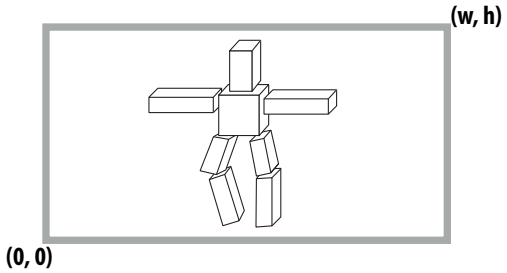


The Rasterization Pipeline

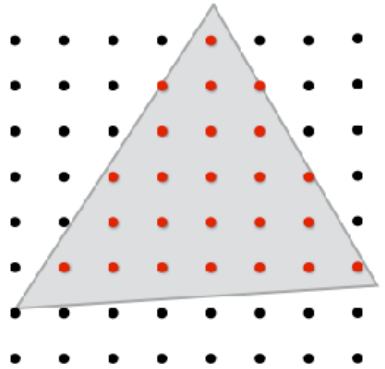
- Rough sketch of rasterization pipeline:



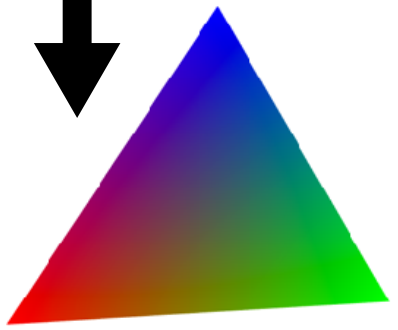
Transform/position objects in the world



Project objects onto the screen



Sample triangle coverage



Interpolate triangle attributes at covered samples

Sample texture maps / evaluate shaders

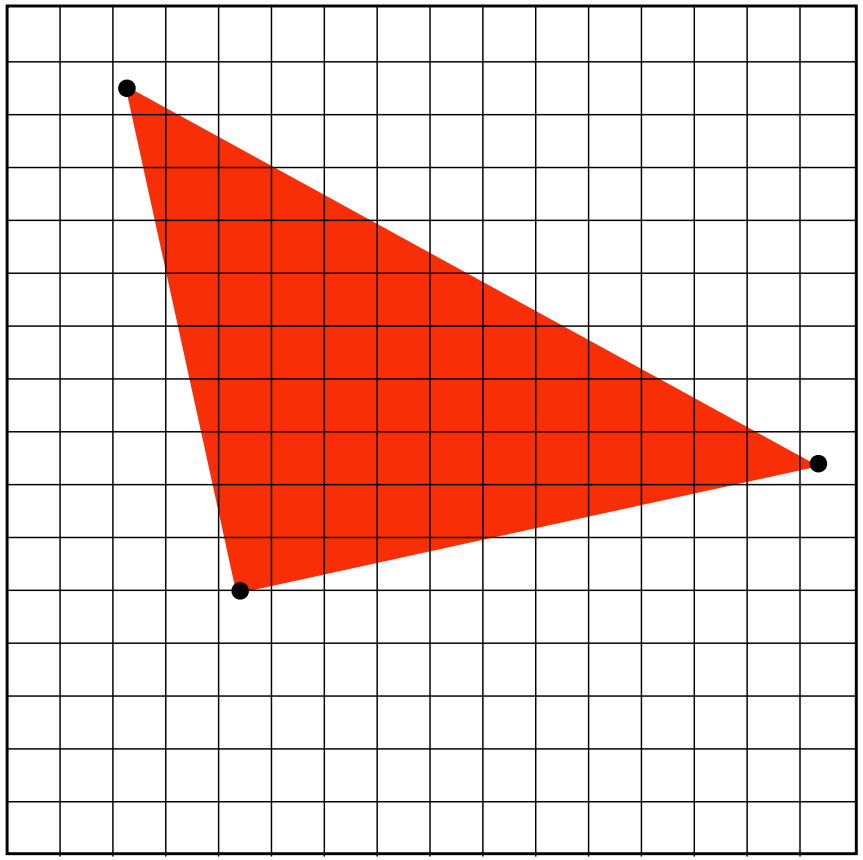
Combine samples into final image (depth, alpha, …)

- Reflects standard “real world” pipeline (OpenGL/Direct3D)

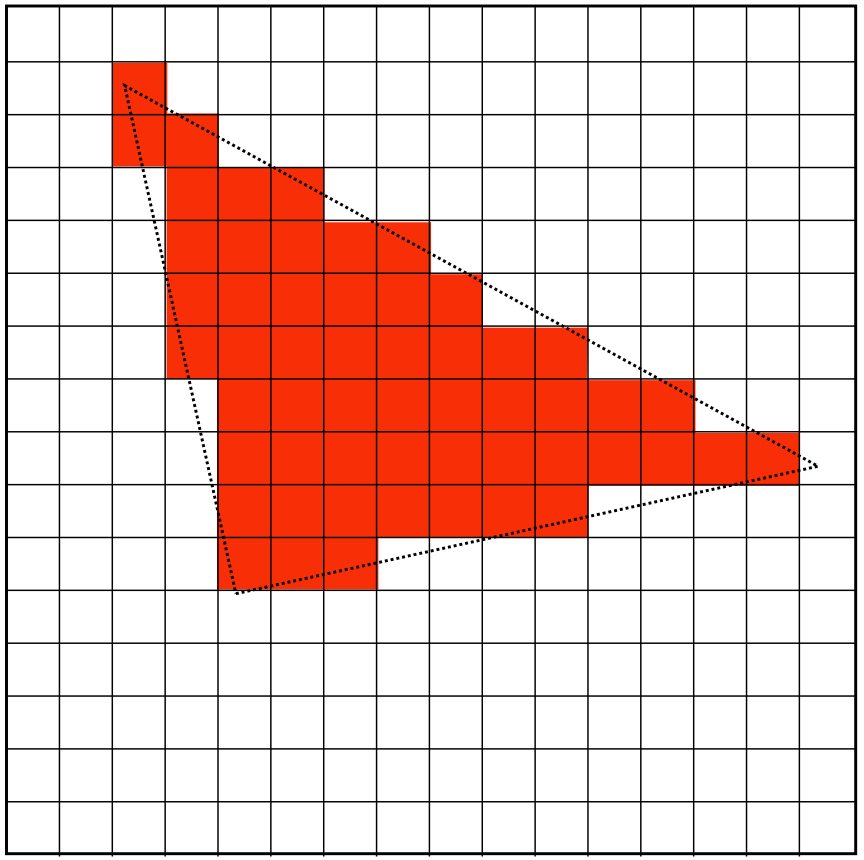
Computing Triangle Coverage

What pixels does the triangle overlaps?

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

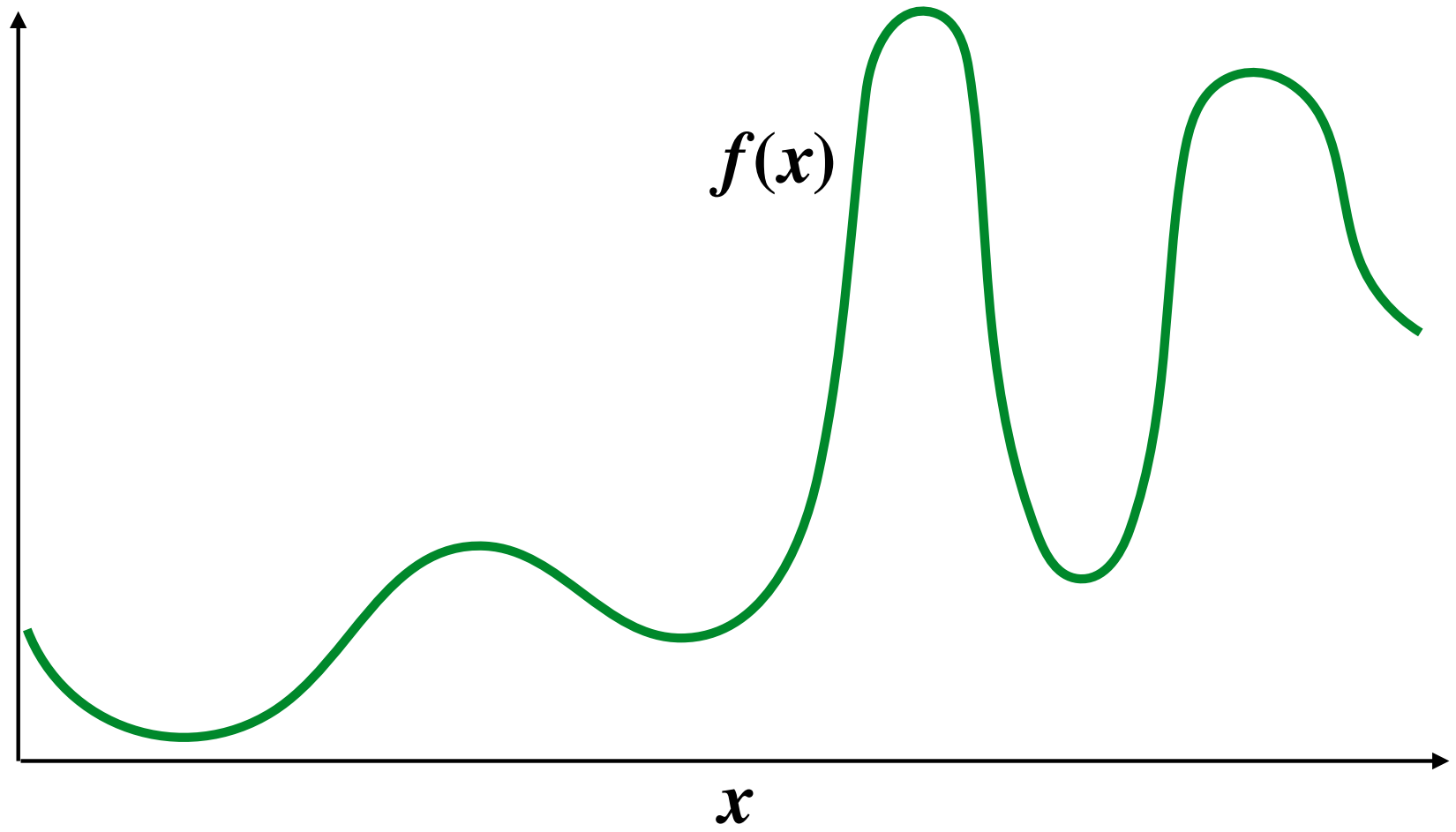


Output:
set of pixels "covered" by the triangle



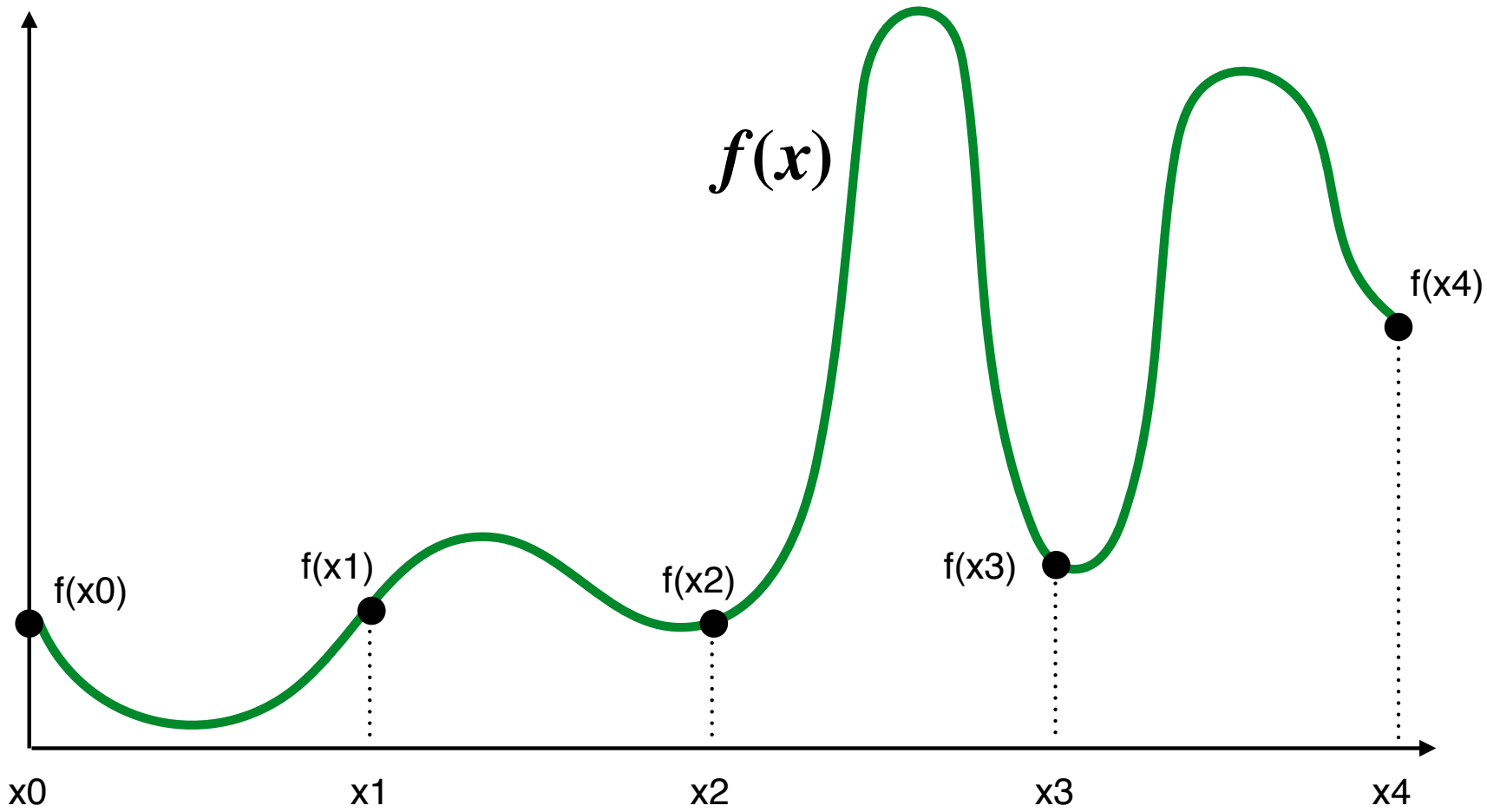
A Simple Approach: Sampling

Sampling a 1D Signal

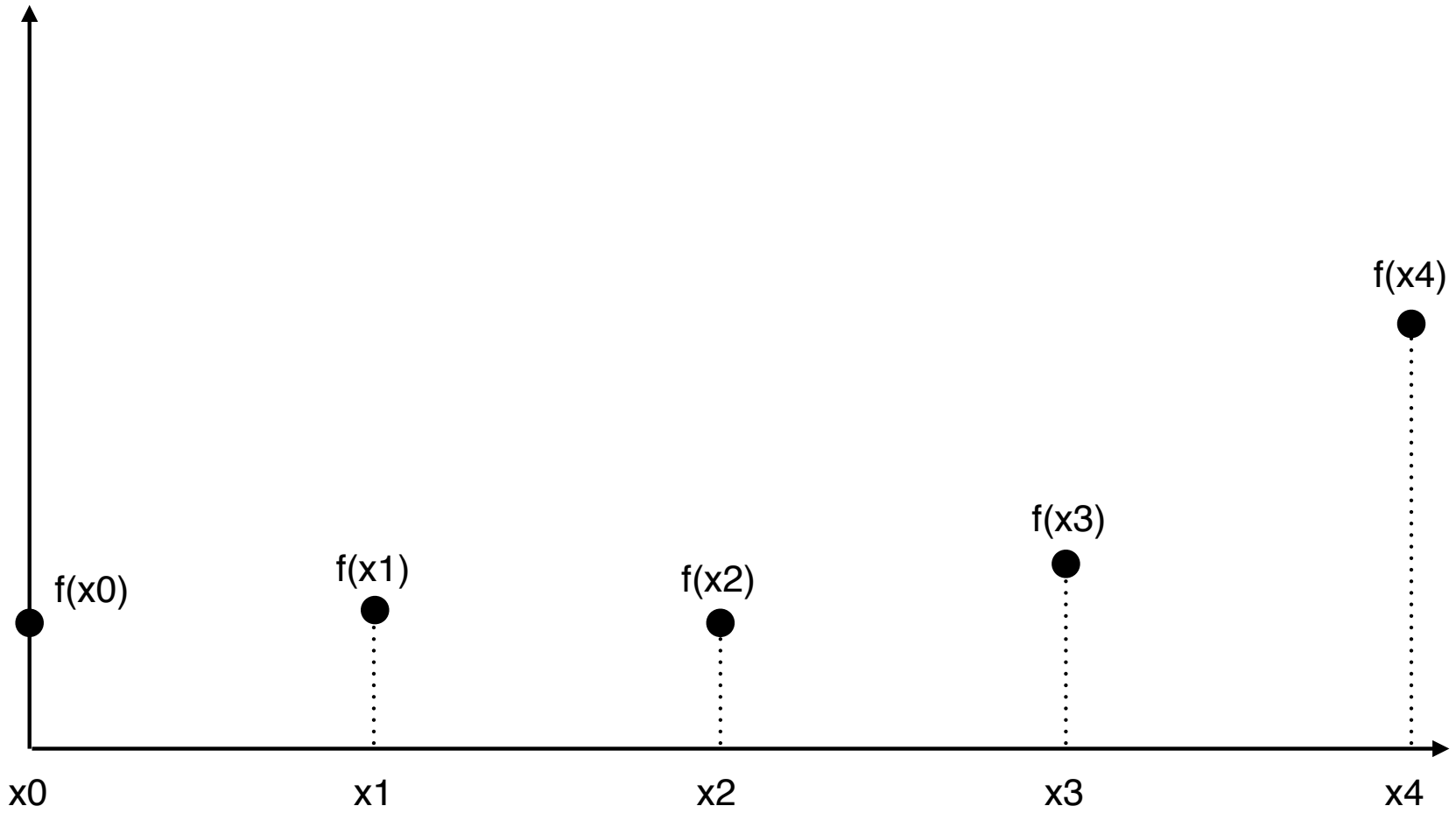


Sampling=taking measurements of a signal

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



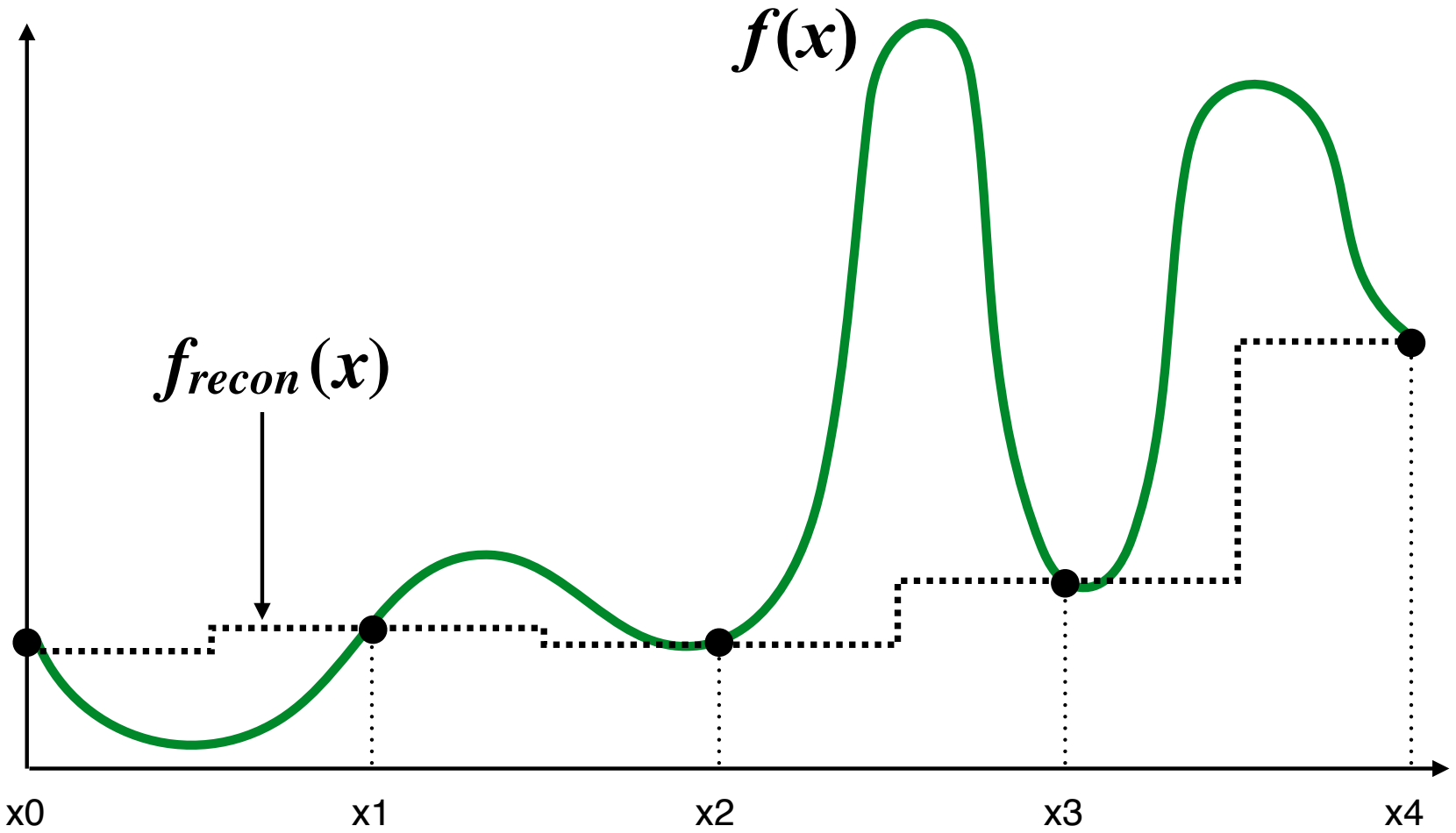
Reconstruction: given a set of samples, how might we attempt to reconstruct the original signal $f(x)$?



Piecewise constant approximation

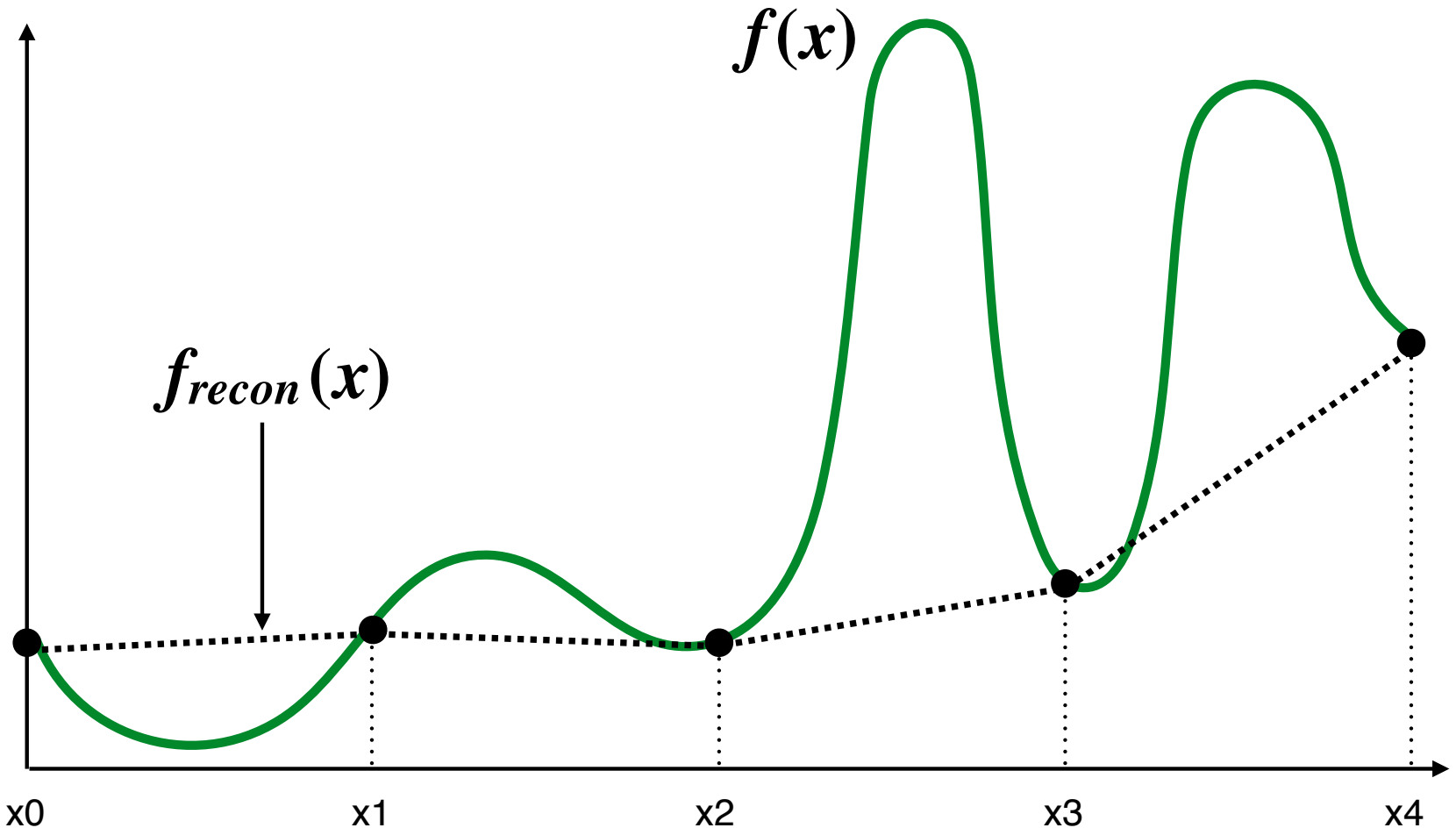
$f_{recon}(x)$ = value of sample closest to x

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

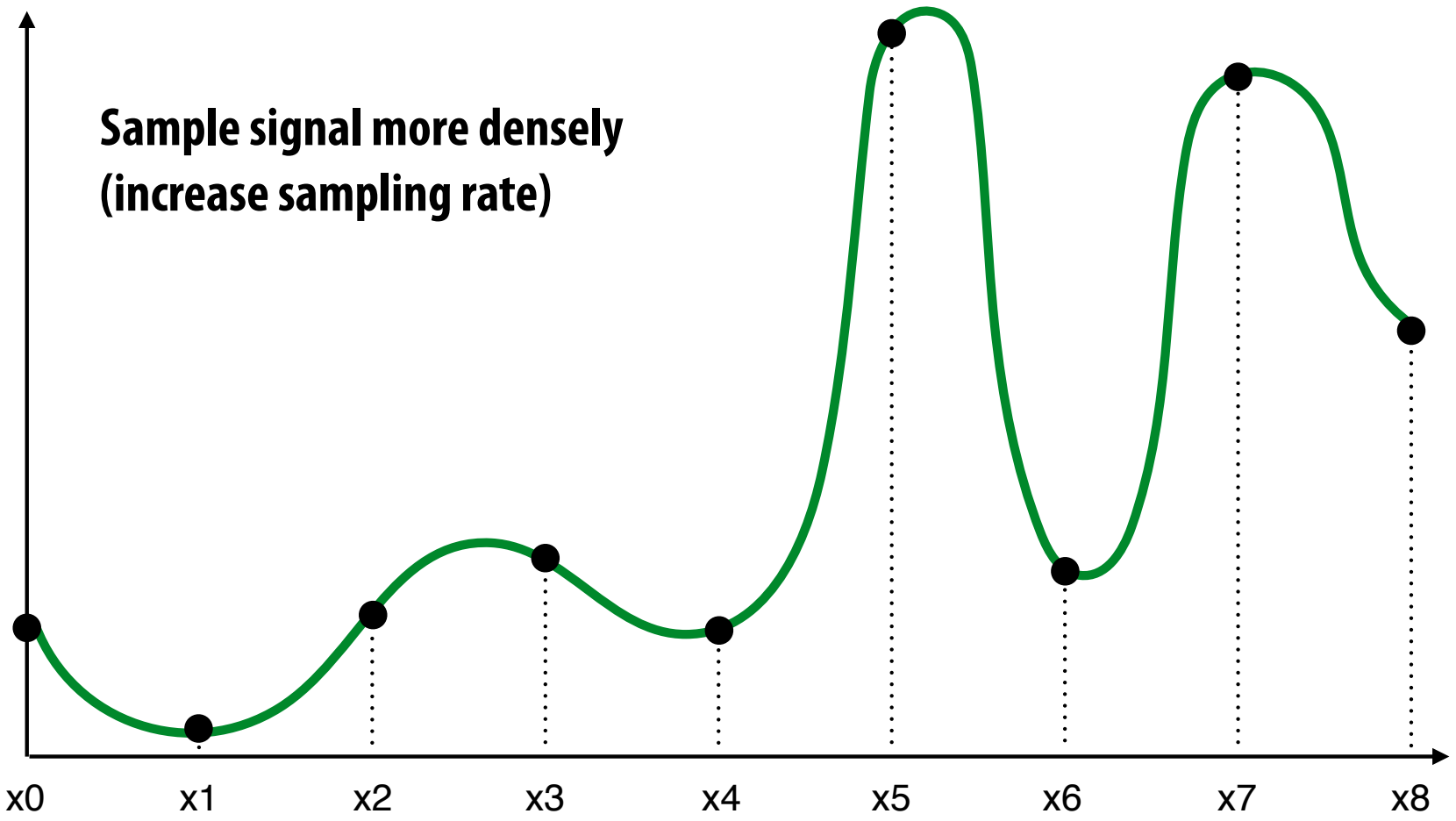


Piecewise linear approximation

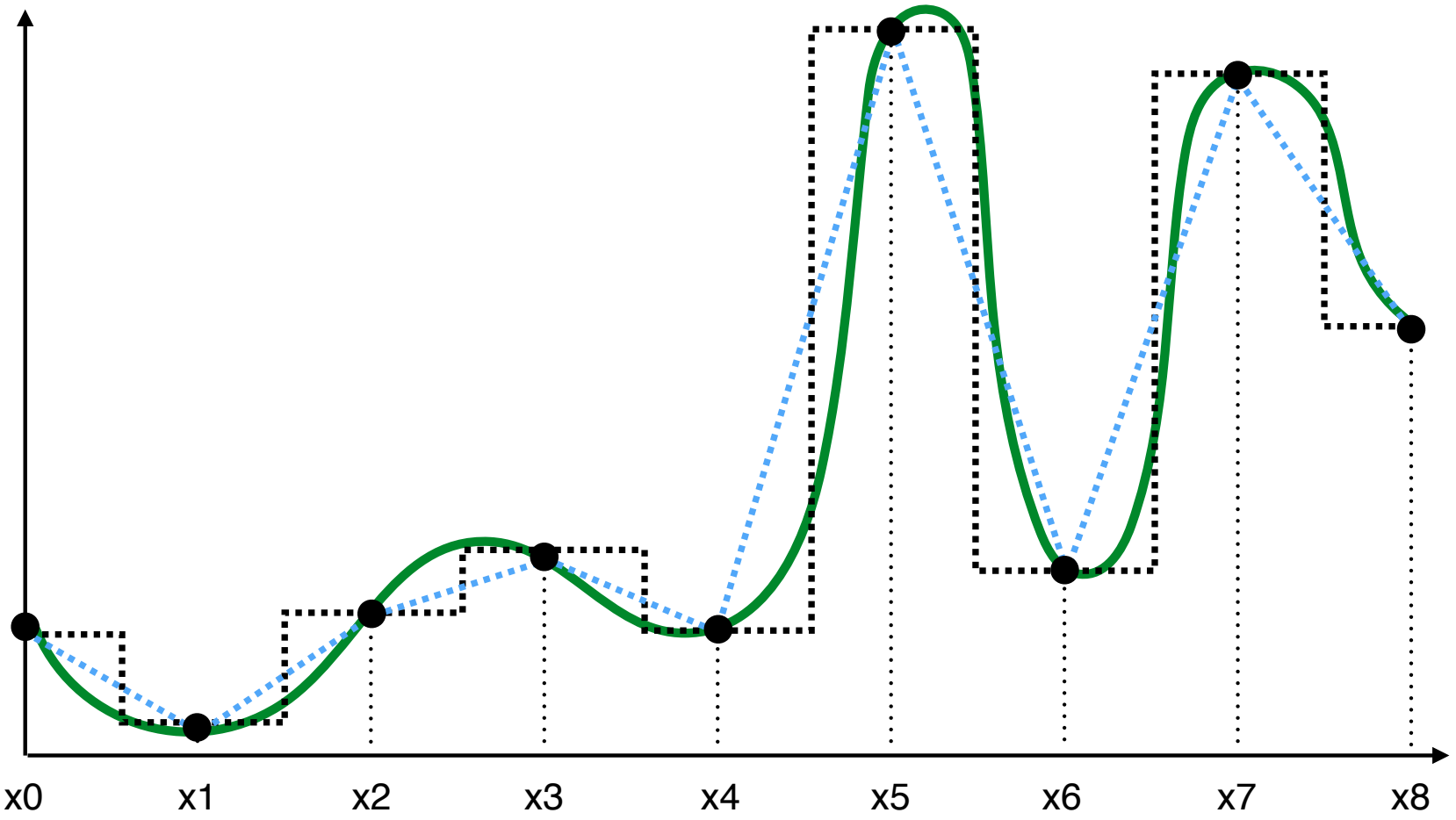
$f_{recon}(x)$ = linear interpolation between values of two closest samples to x



How can we represent the signal more accurately?



Reconstruction from denser sampling

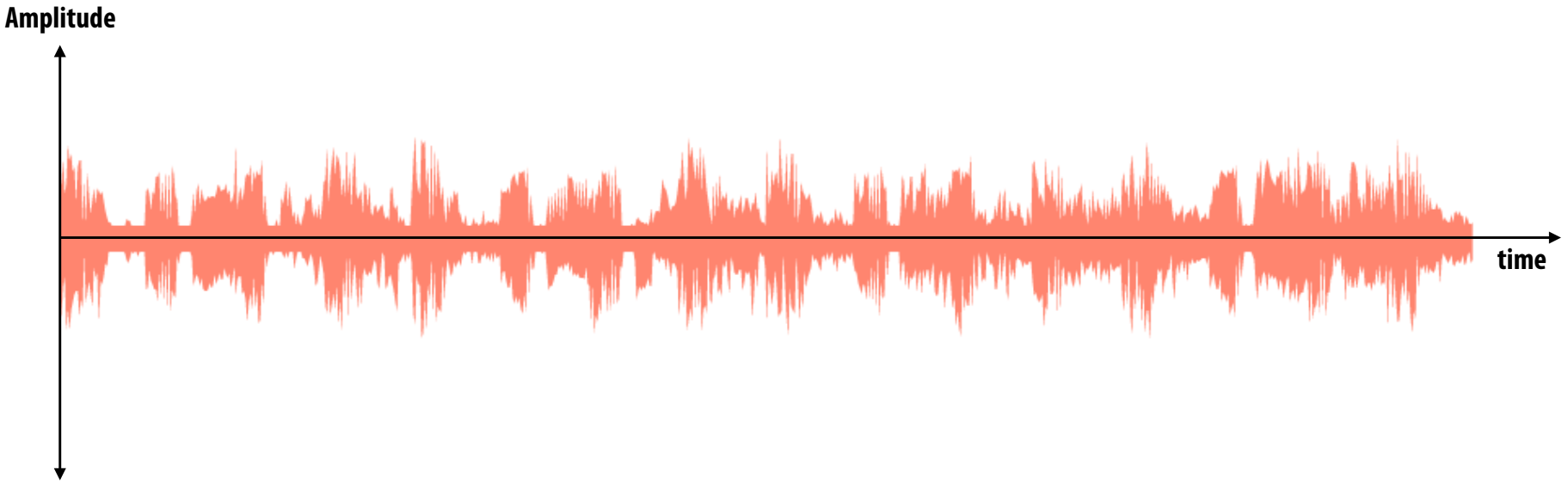


= reconstruction via nearest

= reconstruction via linear interpolation

Audio file: stores samples of a 1D signal

Sampled at 44.1 KHz (44100/second) that can restore the sound accurately.



2D Sampling & Reconstruction

- Basic story doesn't change much for images:
  - sample values measure image (i.e., signal) at sample points
  - apply interpolation/reconstruction filter to approximate image

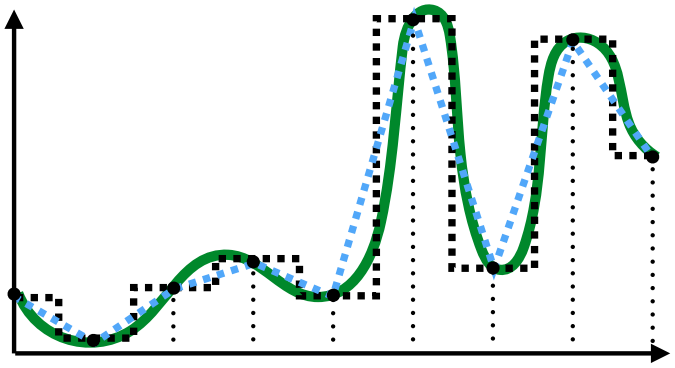
original

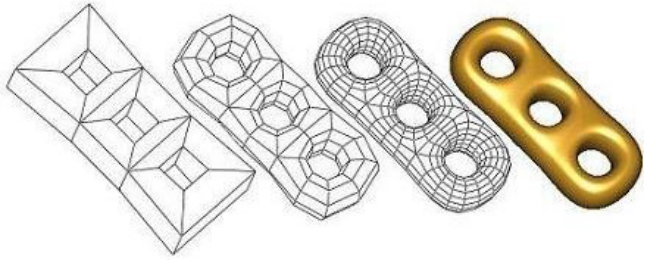
piecewise constant ("nearest neighbor")

piecewise *bi*-linear

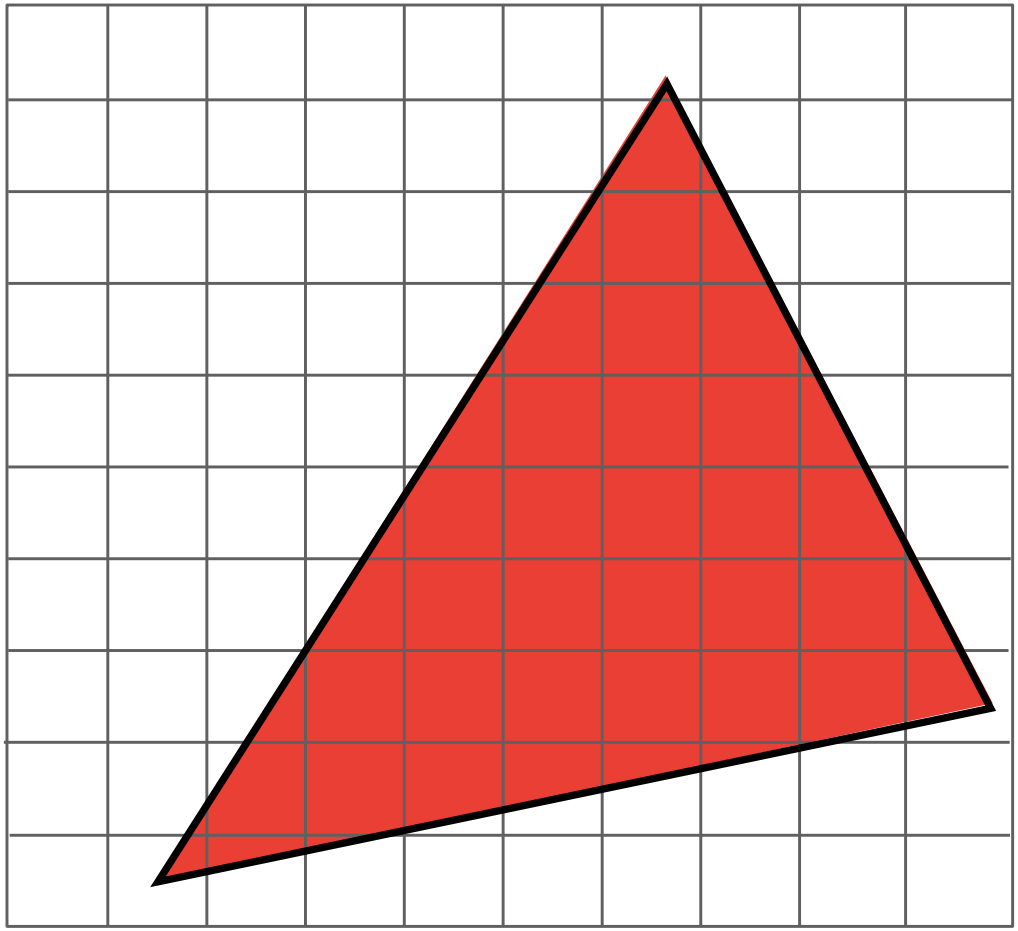
Sampling Summary

- Sampling = measurement of a signal
  - Encode signal as discrete set of samples
  - In principle, represent values at specific points (though hard to measure in reality!)
- Reconstruction = generating signal from a discrete set of samples
  - Construct a function that interpolates or approximates function values
  - E.g., piecewise constant/"nearest neighbor", or piecewise linear
  - Many more possibilities! For all kinds of signals (audio, images, geometry...)



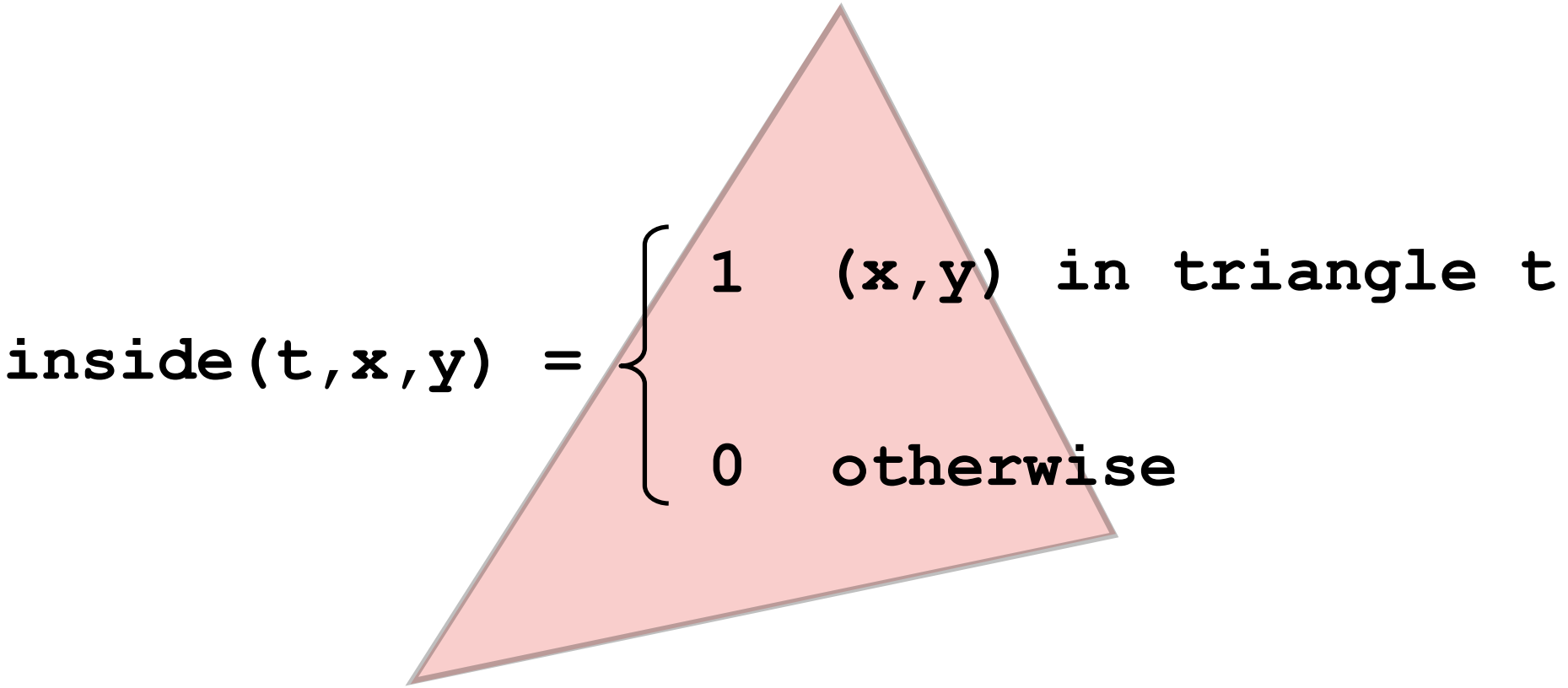


Let's Try Rasterization As 2D Sampling

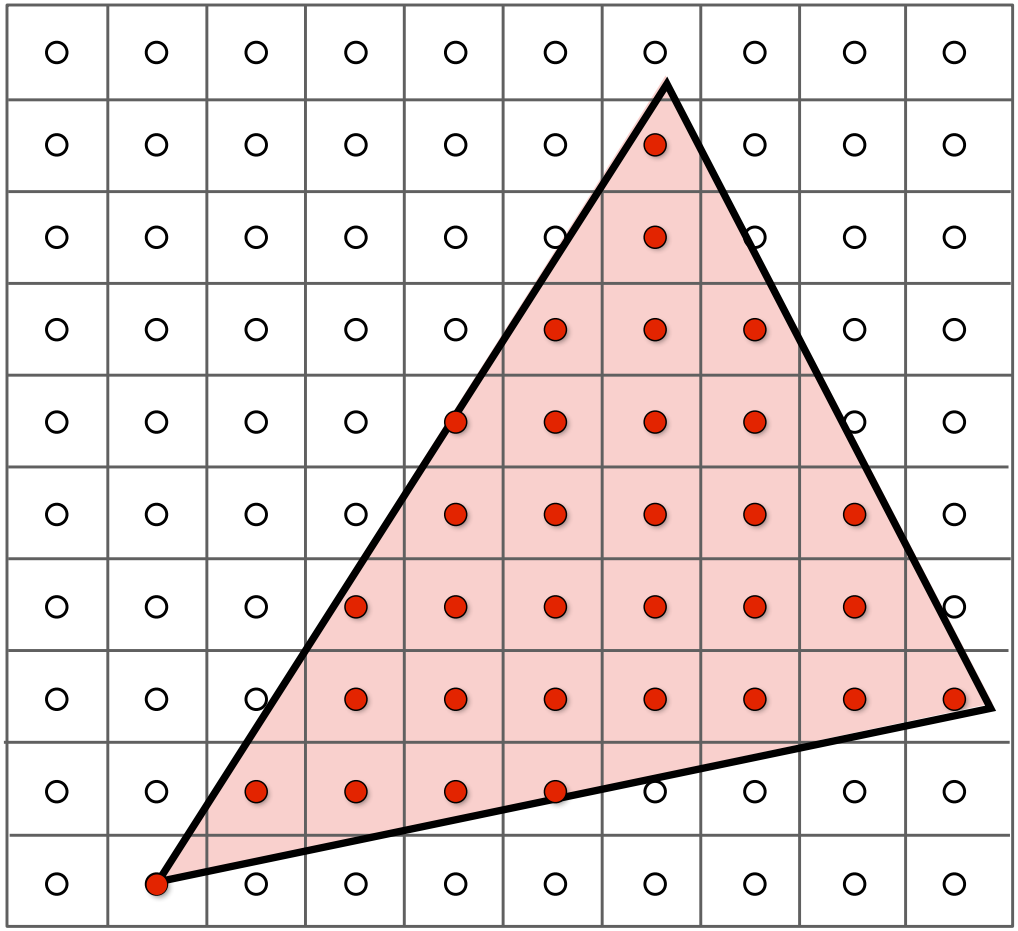


What function are we sampling?

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



Sample If Each Pixel Center Is Inside Triangle



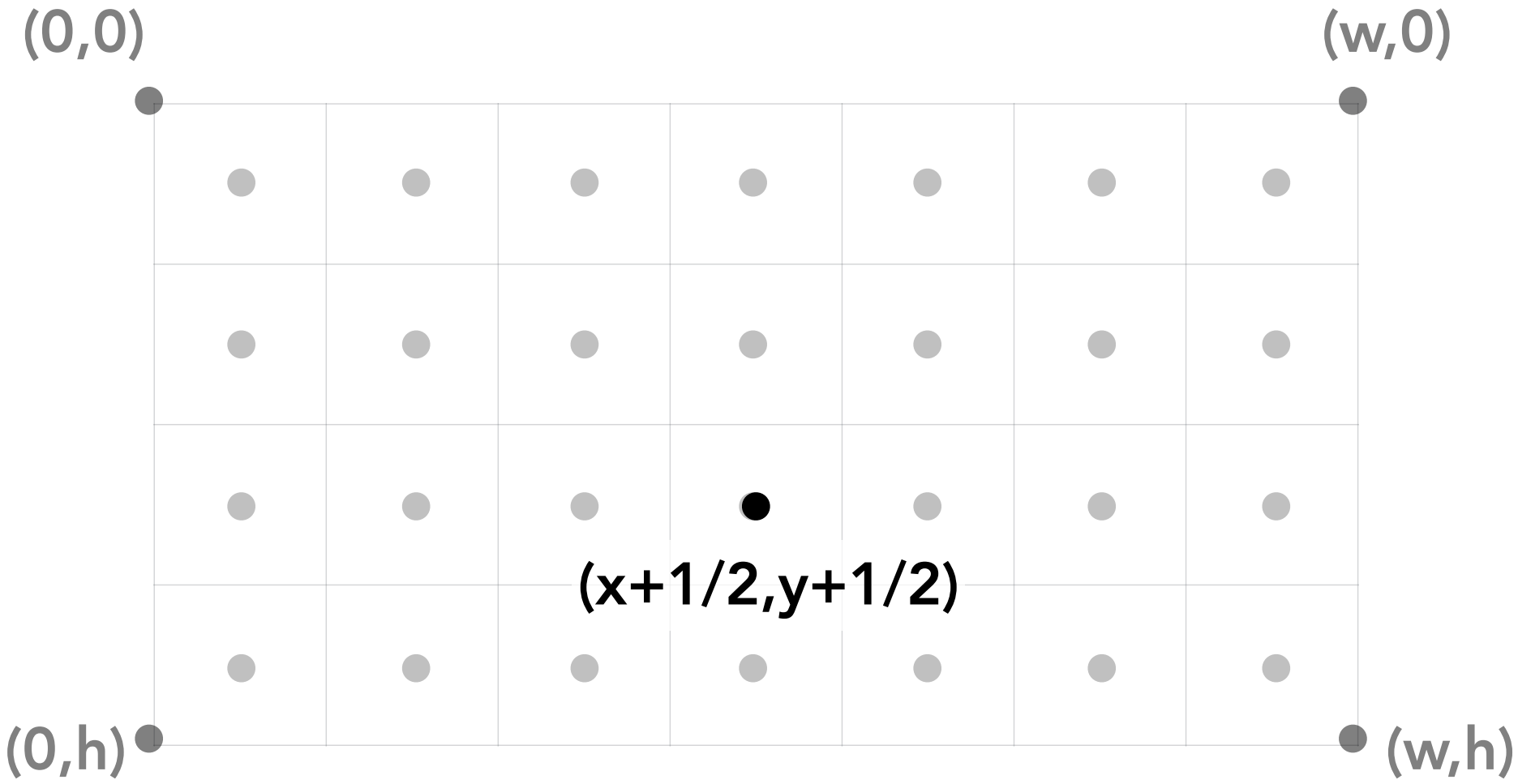
Rasterization = Sampling A 2D Indicator Function

```
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

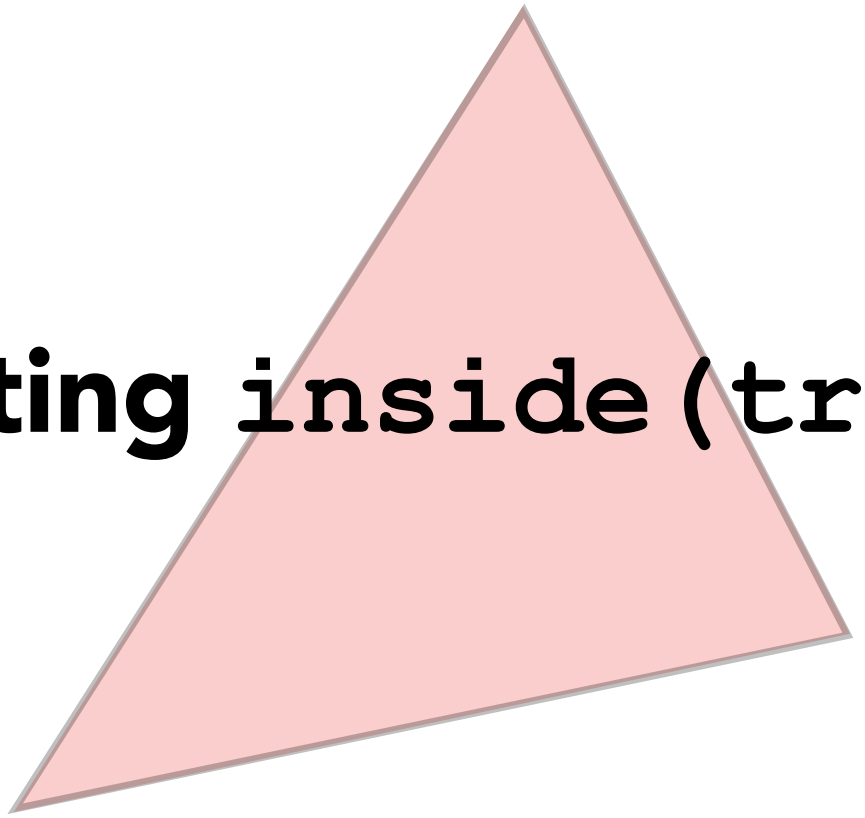
`f(x,y) = inside(tri,x,y)`

Implementation Detail: Sample Locations

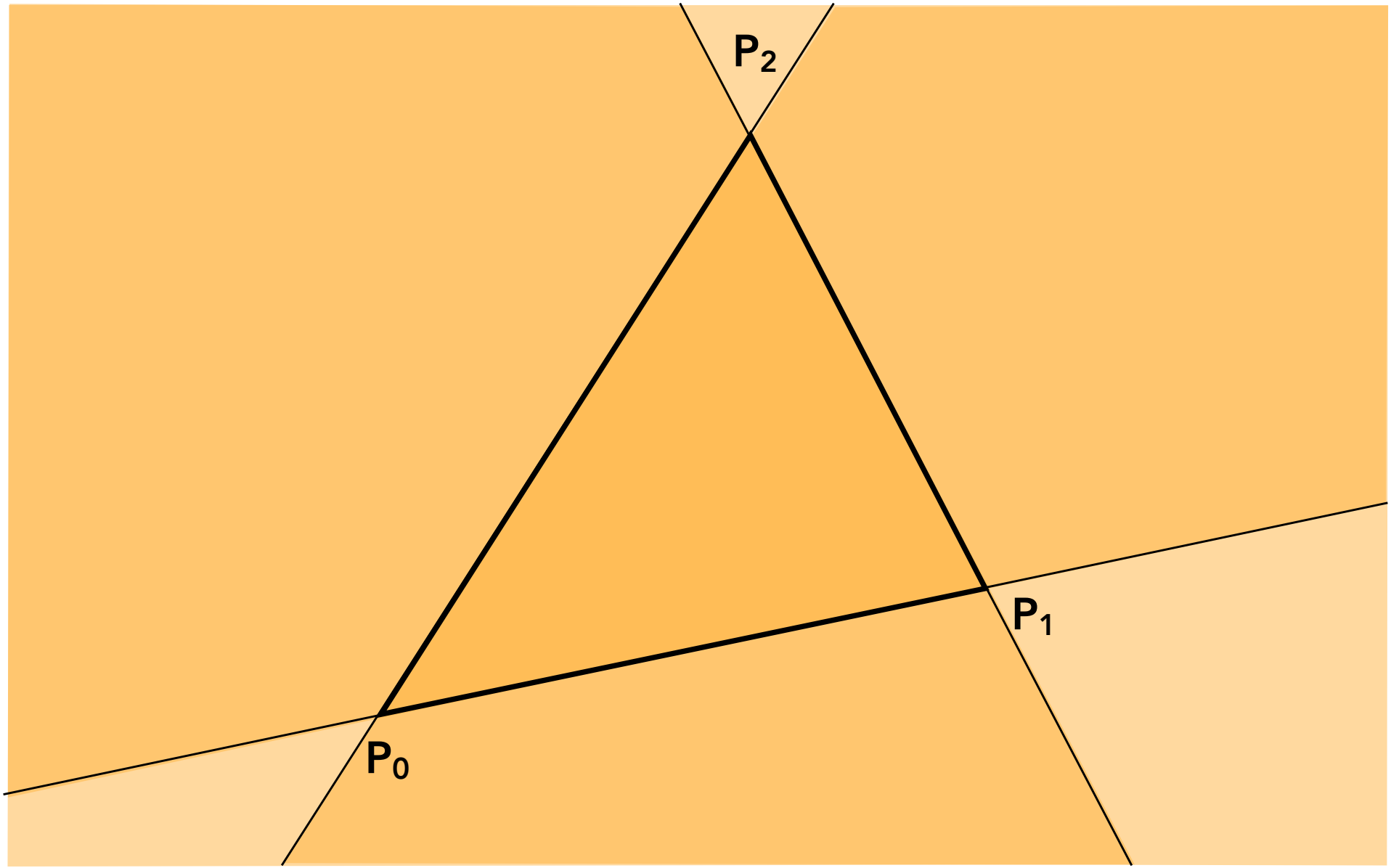


Sample location for pixel (x,y)

Evaluating `inside(tri,x,y)`



Triangle = Intersection of Three Half Planes

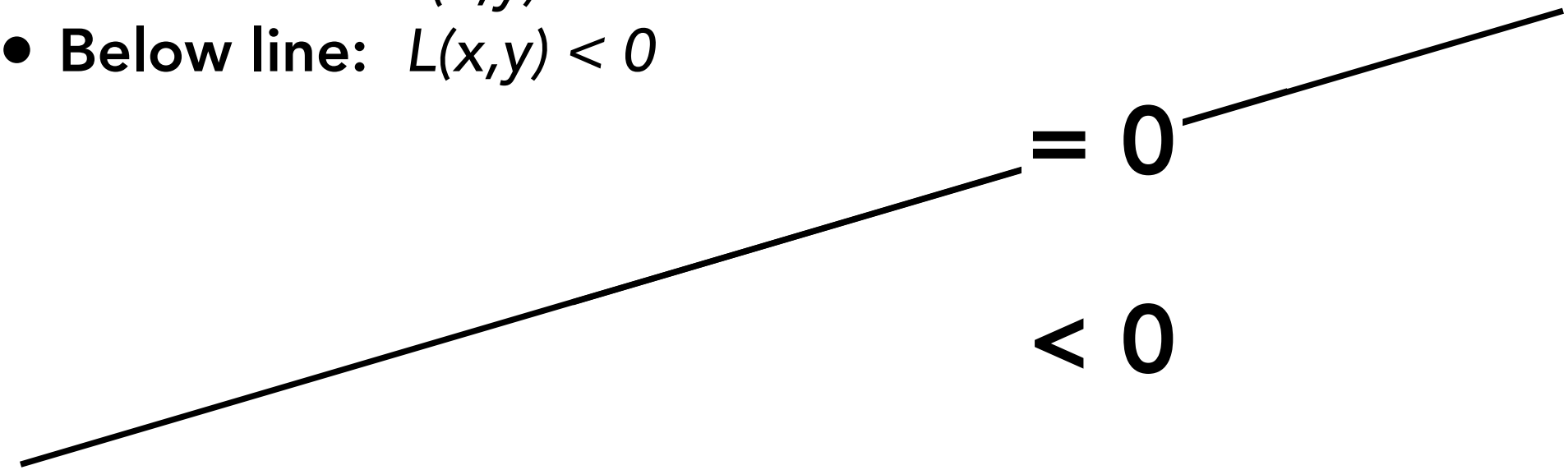


Each Line Defines Two Half-Planes

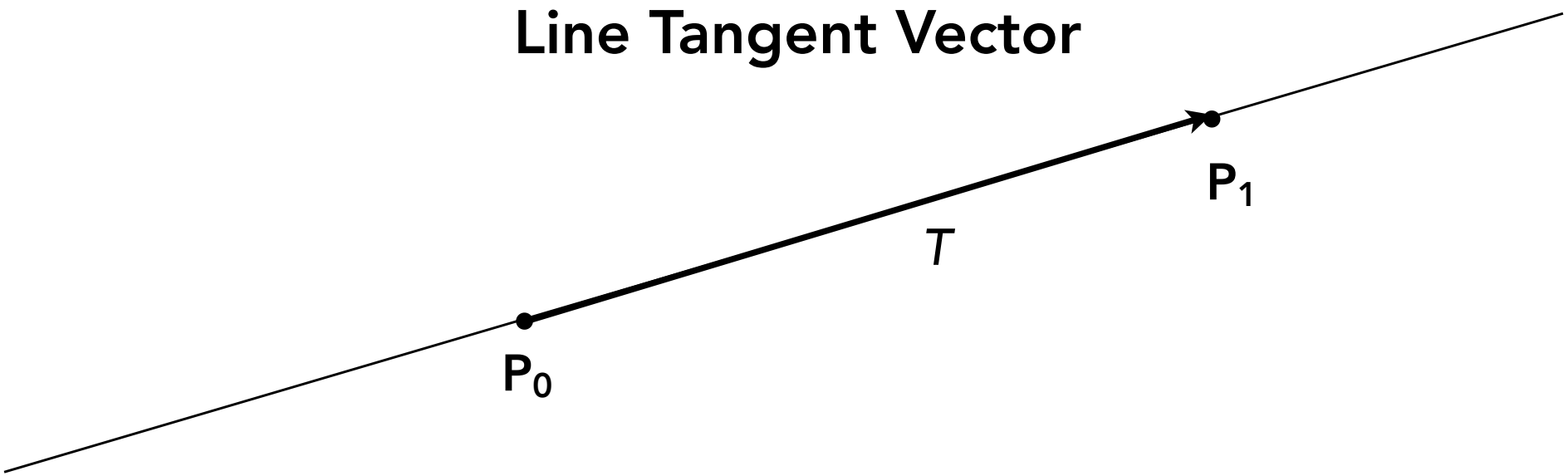
Implicit line equation

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

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

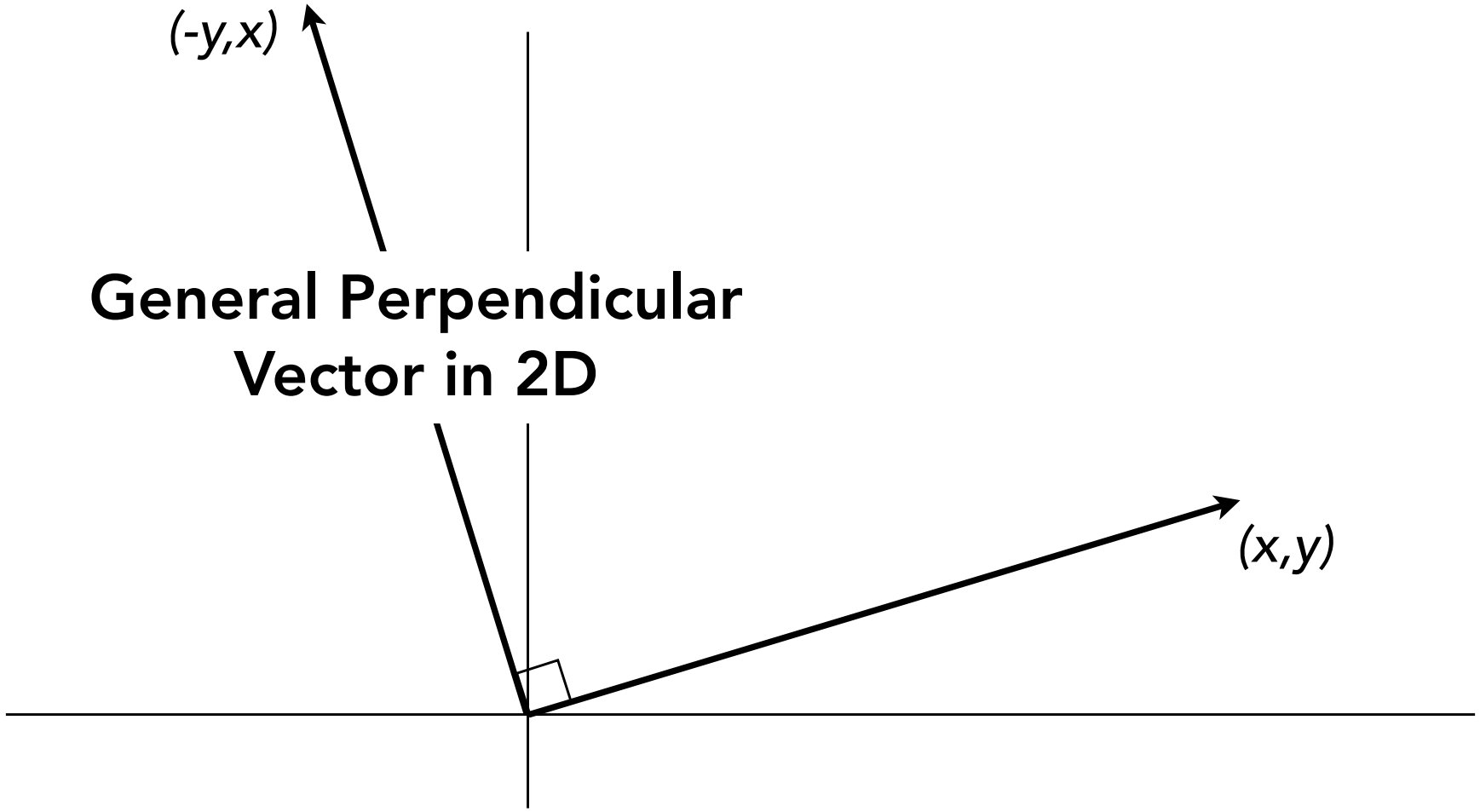


Line Equation Derivation



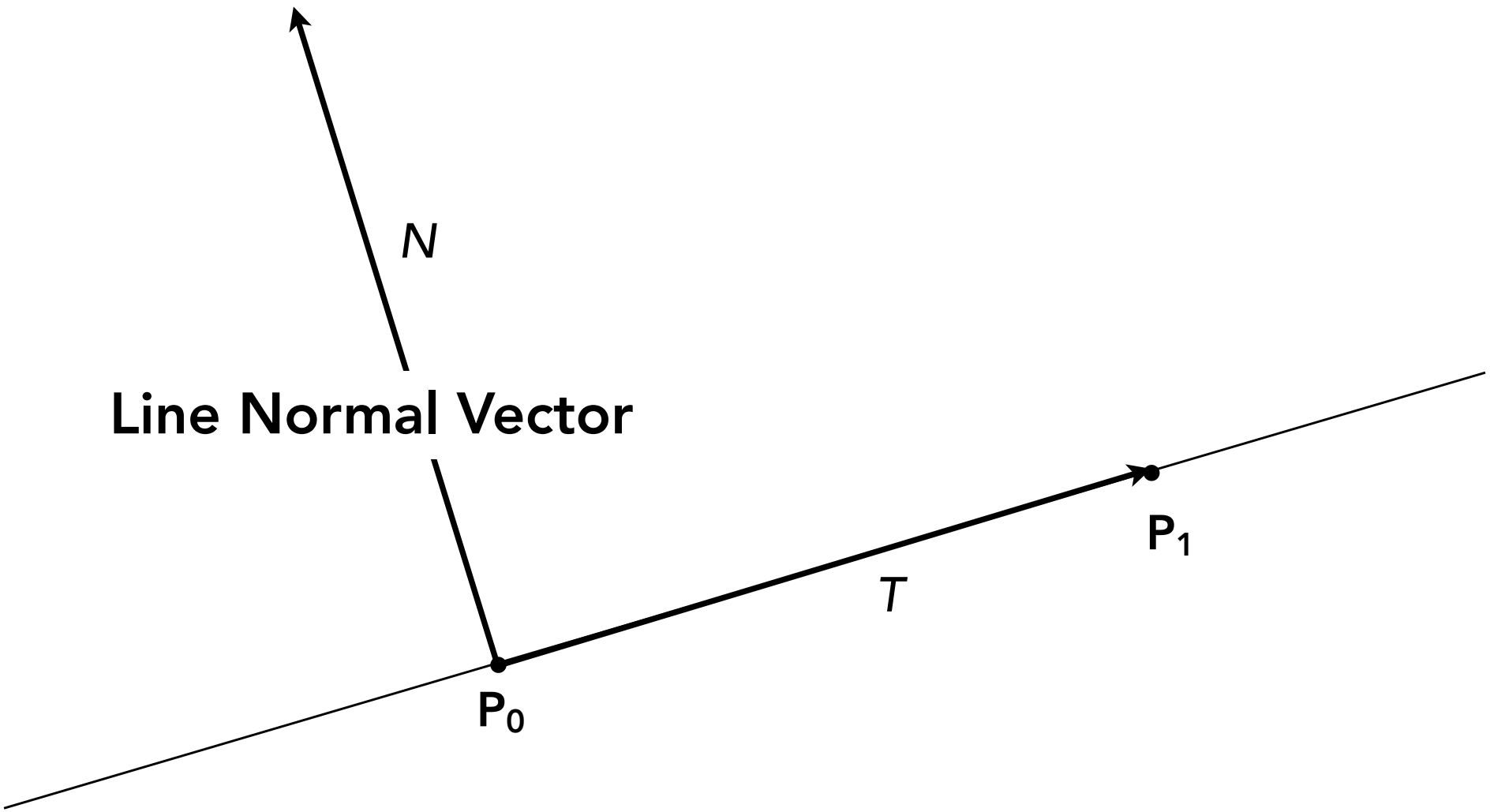
$$T = P_1 - P_0 = (x_1 - x_0, y_1 - y_0)$$

Line Equation Derivation



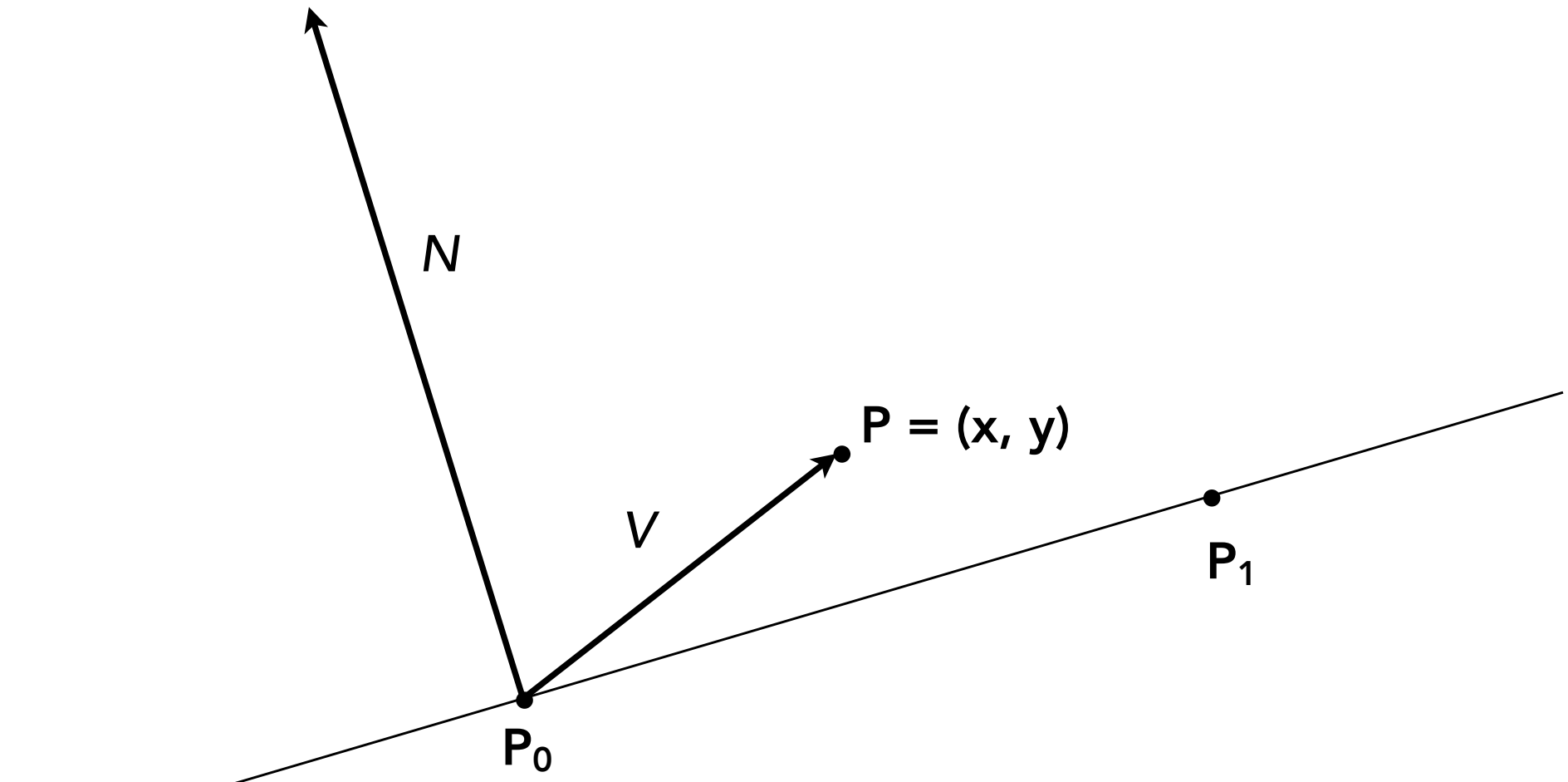
$$\mathrm{Perp}(x, y) = (-y, x)$$

Line Equation Derivation



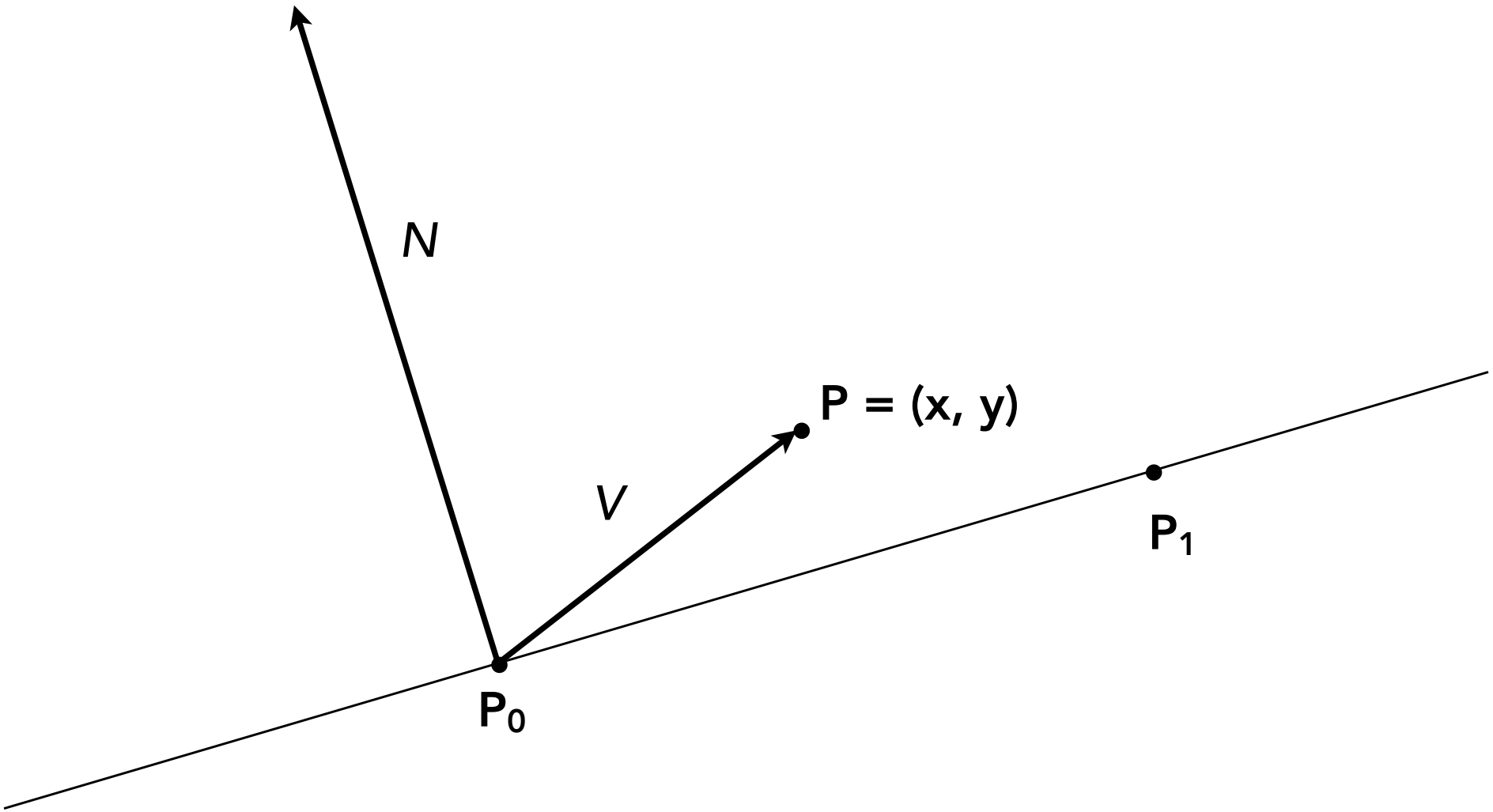
$$N = \text{Perp}(T) = (-(y_1 - y_0), x_1 - x_0)$$

Line Equation Derivation



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

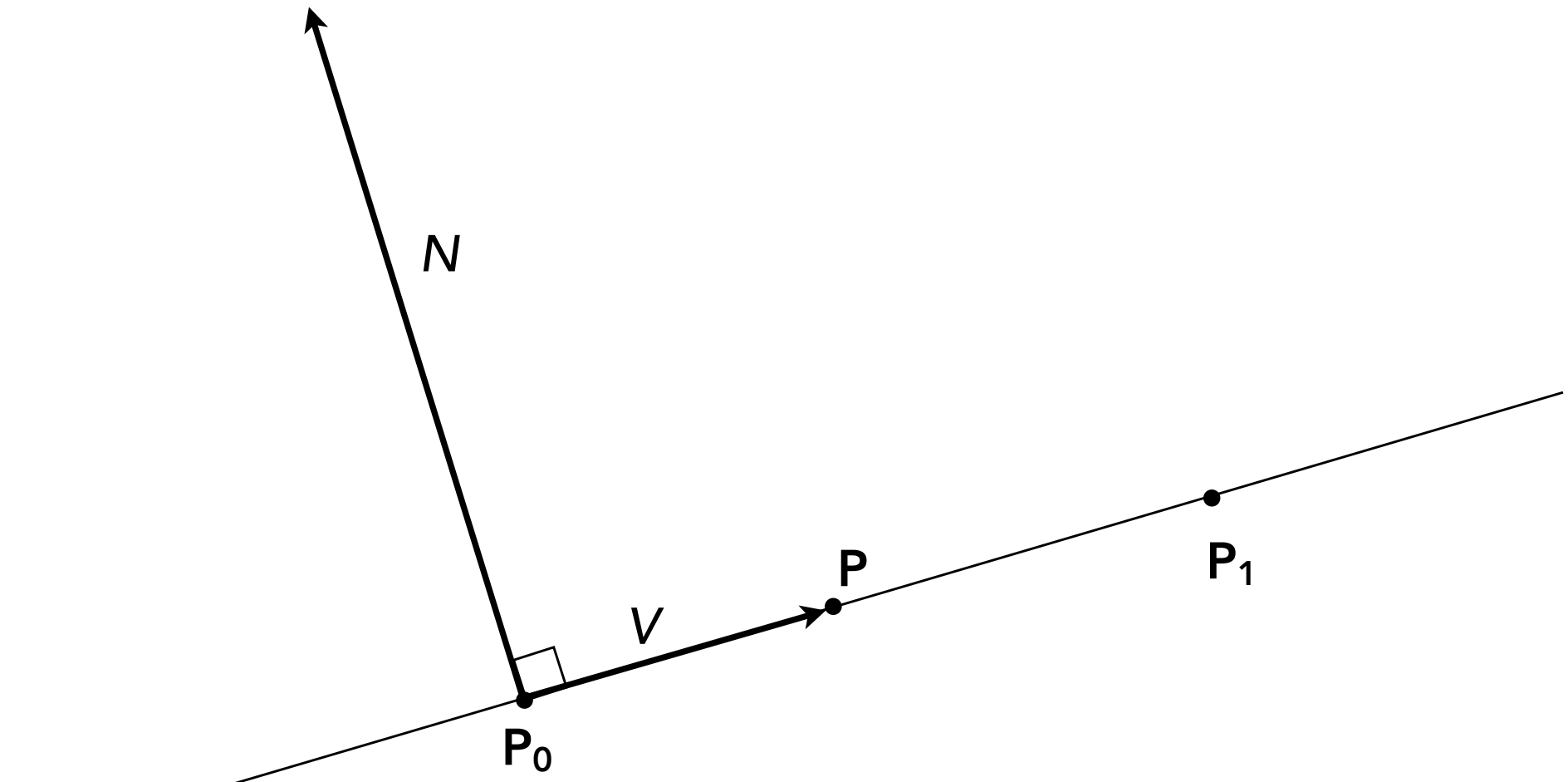
Line Equation



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

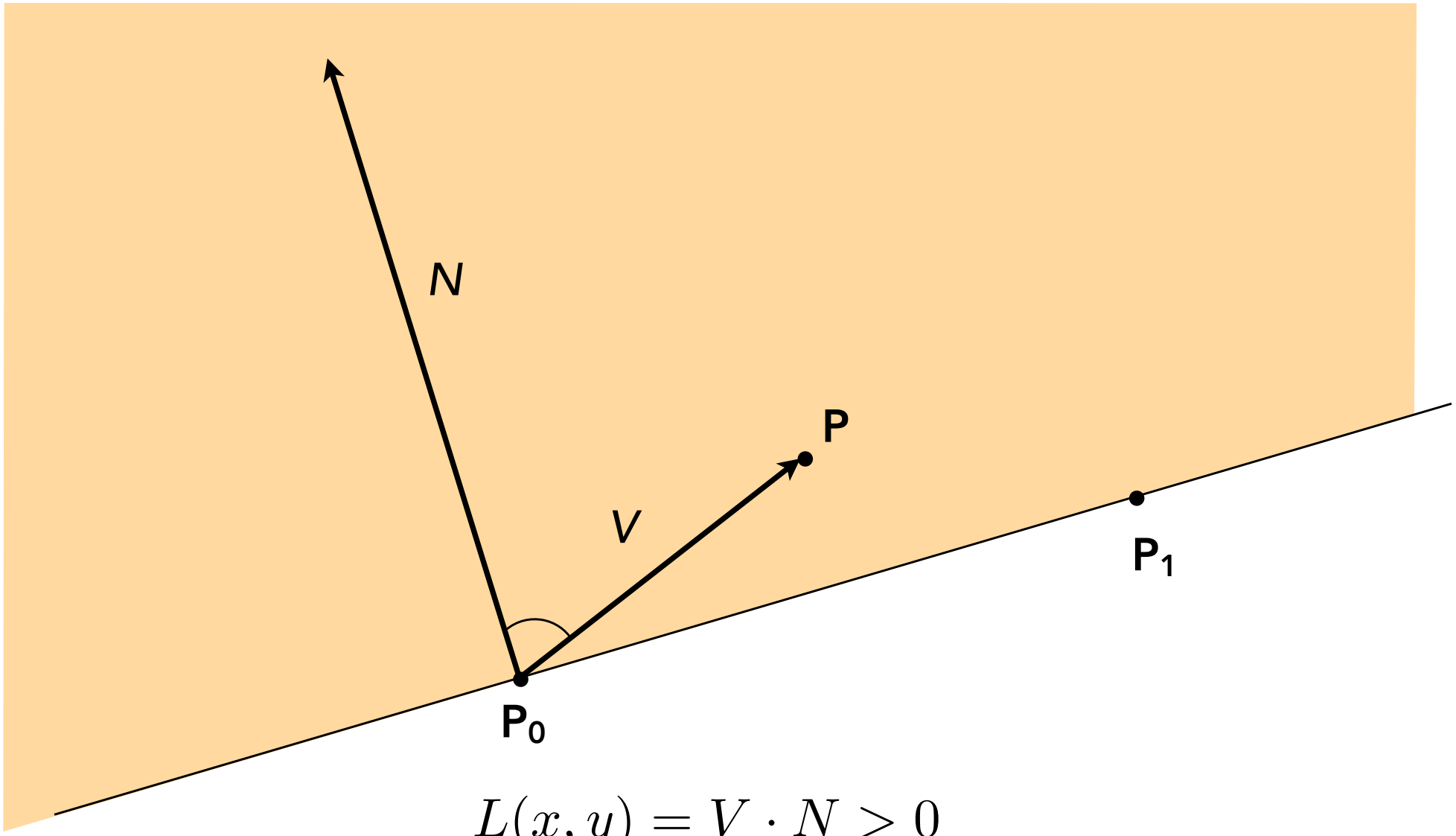
When you take the dot product of the normal vector and a point vector on the line, you get a scalar value that is constant for all points on the line, forming the implicit equation of the line.

Line Equation Tests



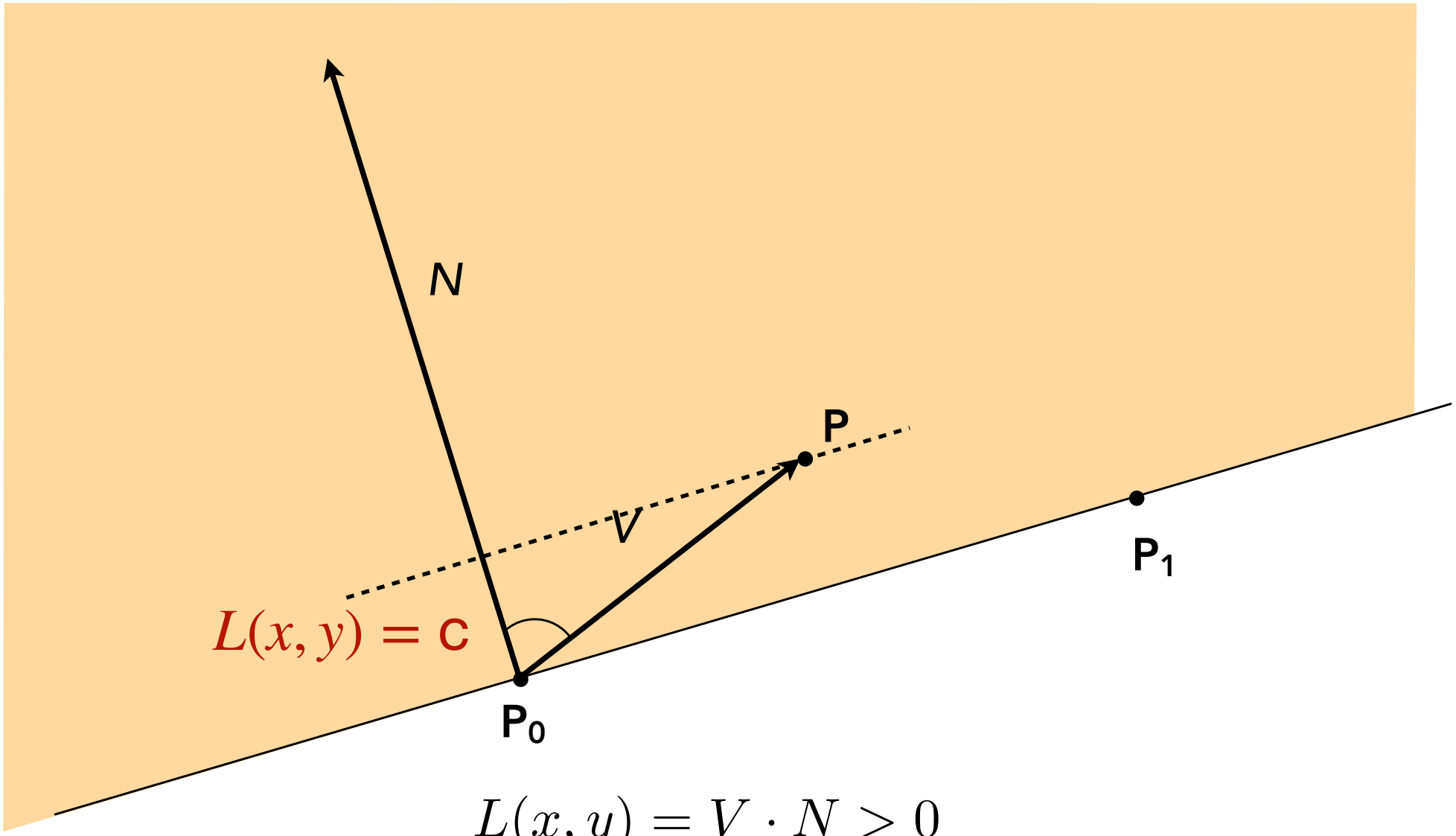
$$L(x, y) = V \cdot N = 0$$

Line Equation Tests



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

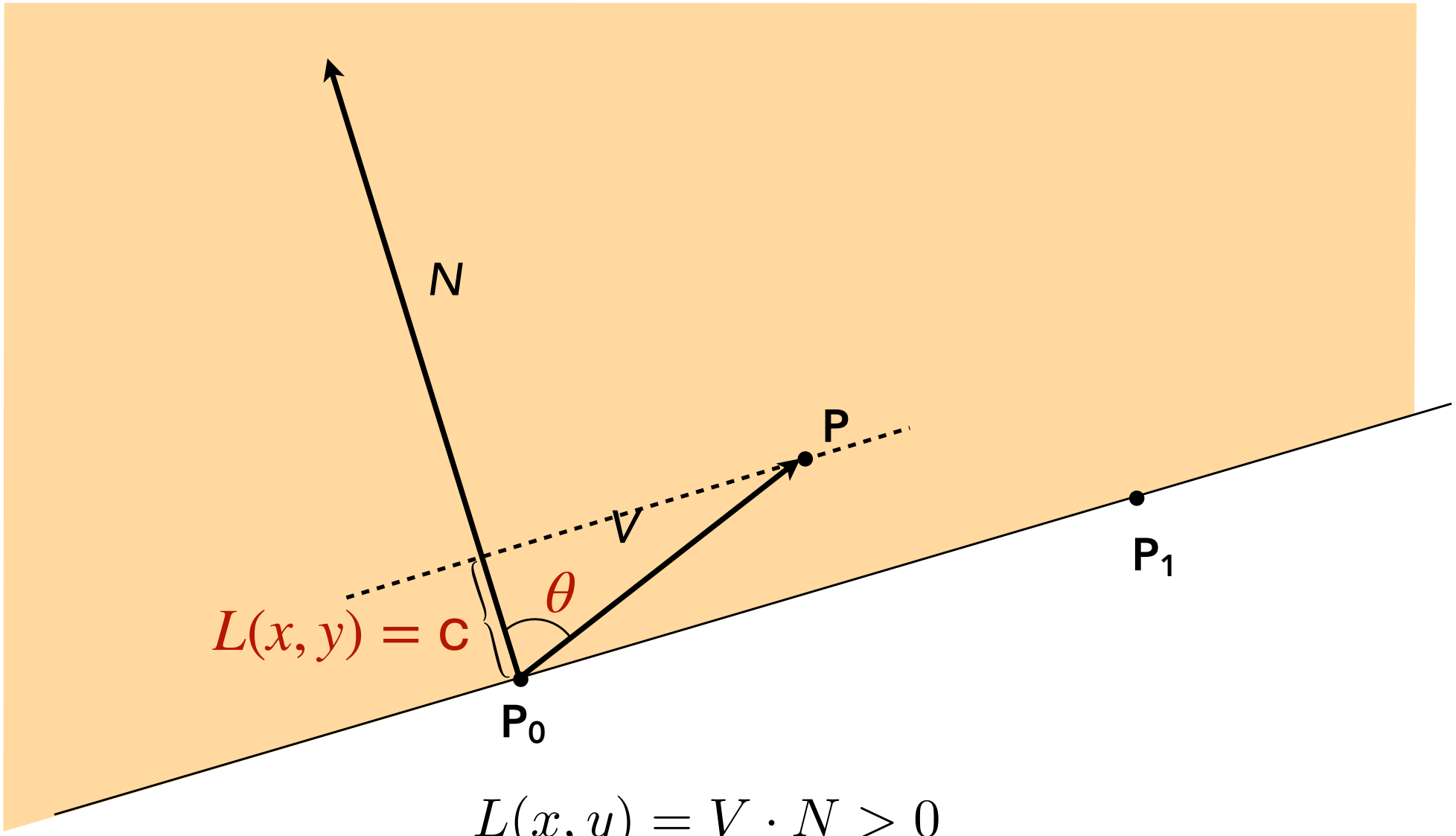
Line Equation Tests



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

When you take the dot product of the normal vector and a point vector on the line, you get a scalar value that is constant for all points on the line, forming the implicit equation of the line.

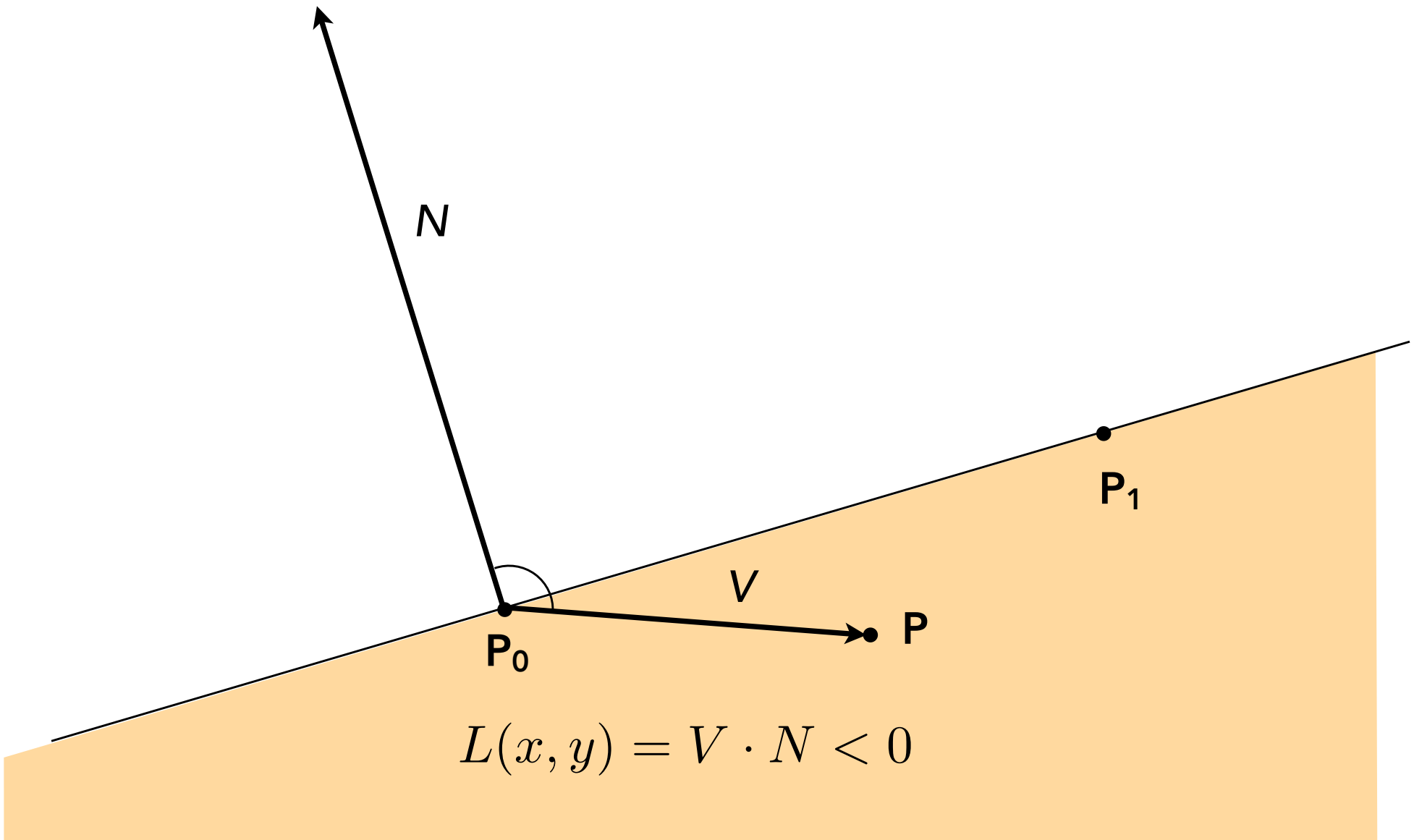
Line Equation Tests



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

$$V \cdot N = |V||N| cos(\theta)$$

Line Equation Tests



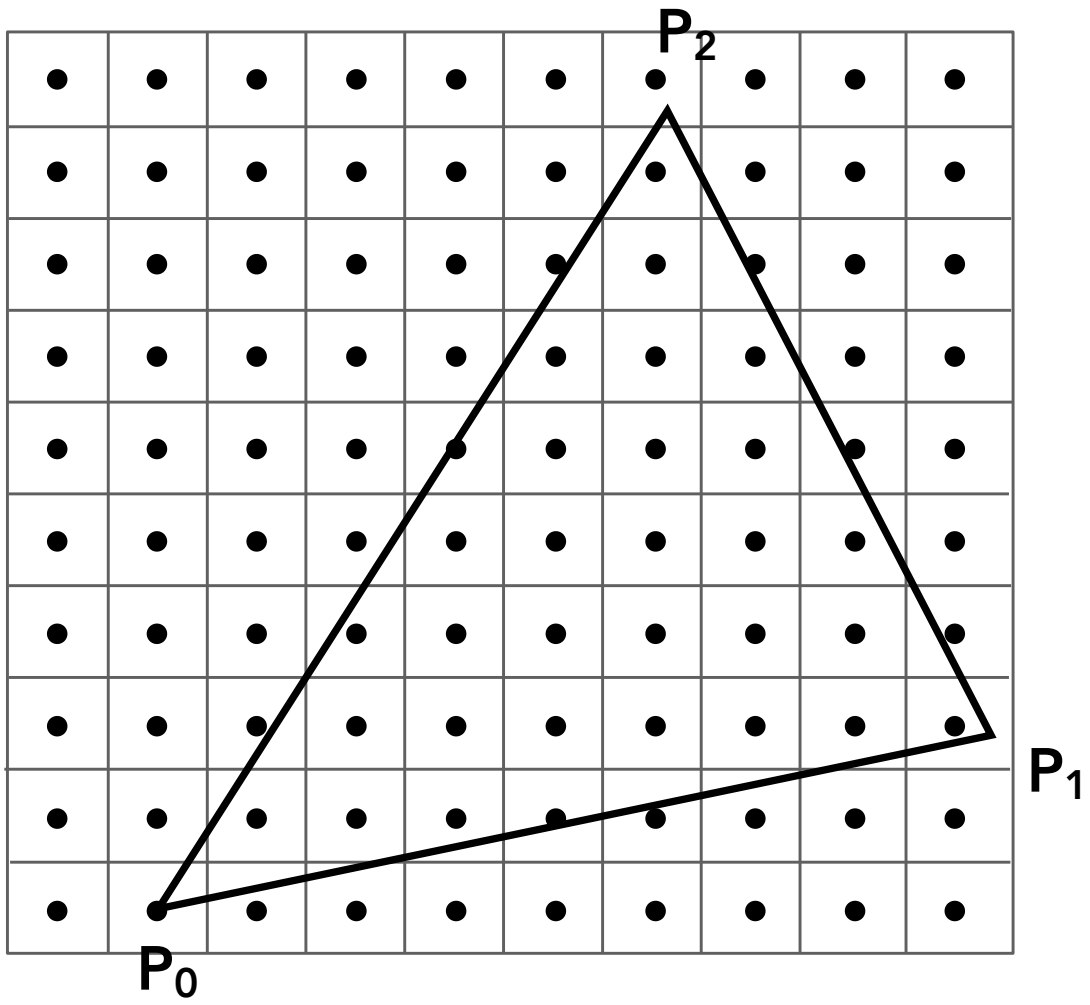
Point-in-Triangle Test: Three Line Tests

$P_i = (X_i, Y_i)$

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

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

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



Compute line equations from pairs of vertices

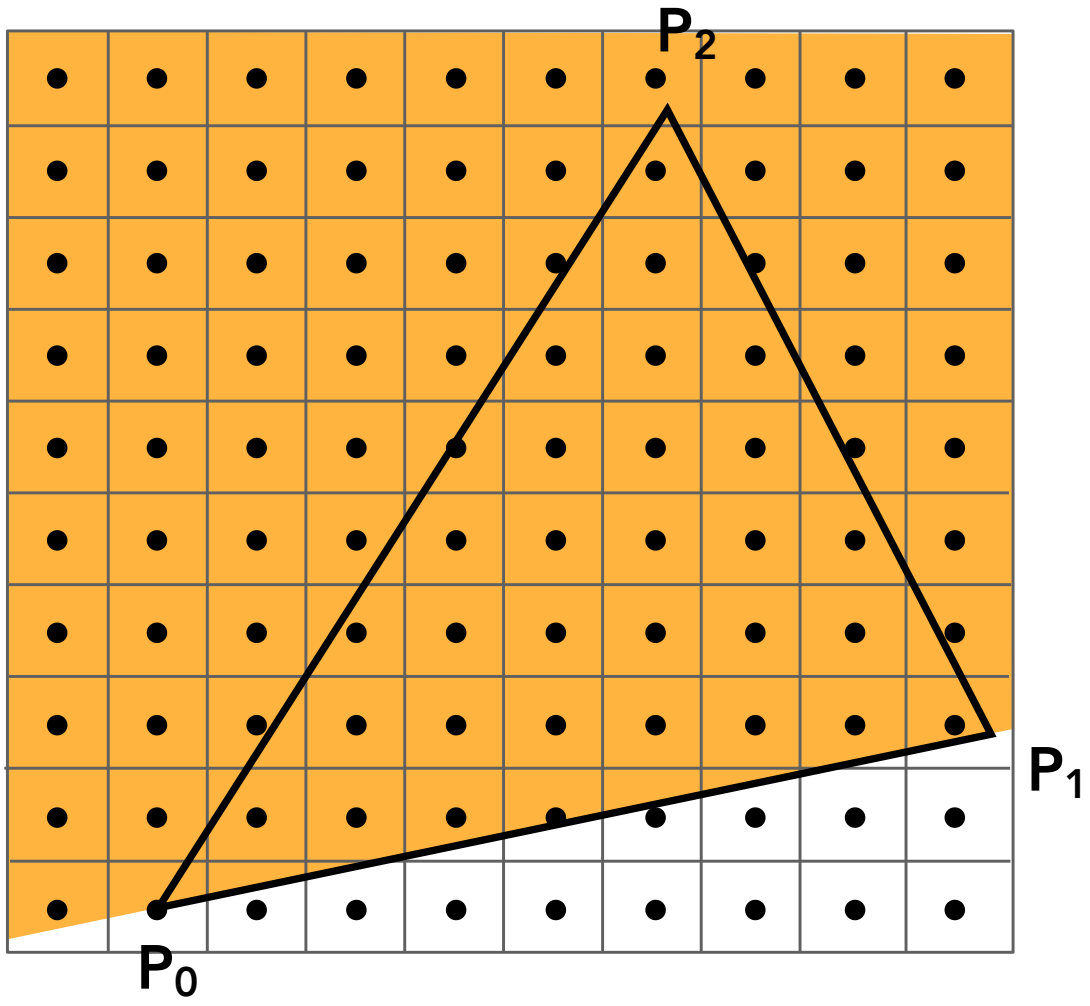
Point-in-Triangle Test: Three Line Tests

$P_i = (X_i, Y_i)$

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

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

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



$L_0(x, y) > 0$

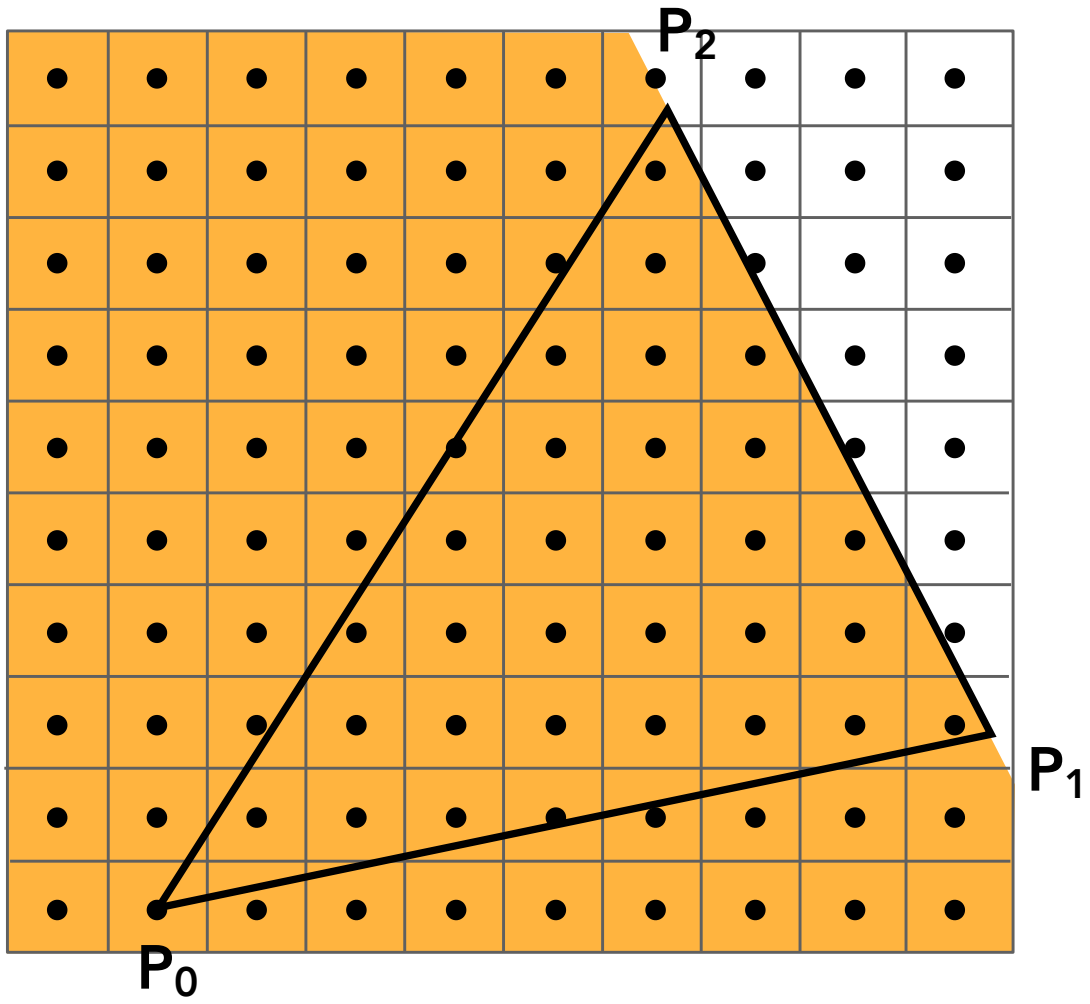
Point-in-Triangle Test: Three Line Tests

$P_i = (X_i, Y_i)$

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

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

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



$L_1(x, y) > 0$

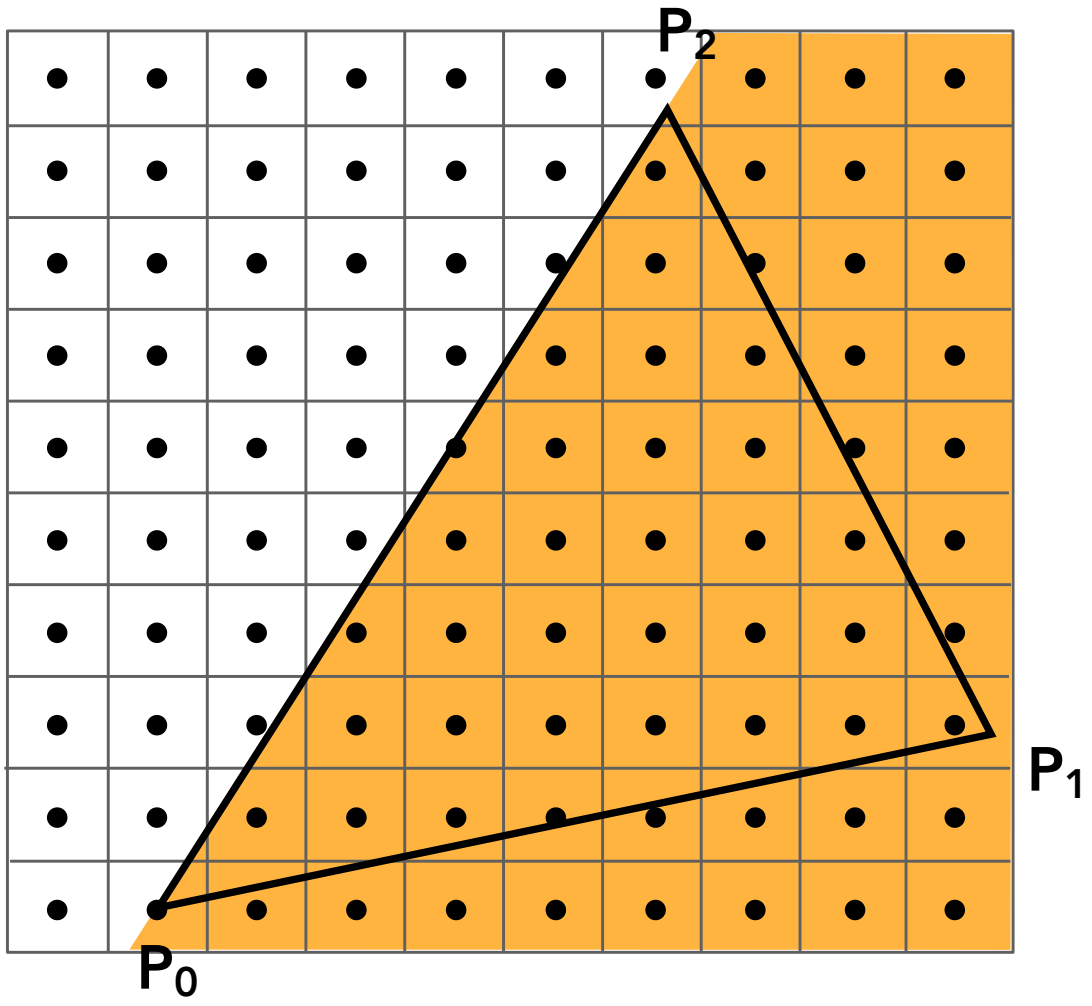
Point-in-Triangle Test: Three Line Tests

$P_i = (X_i, Y_i)$

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

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

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



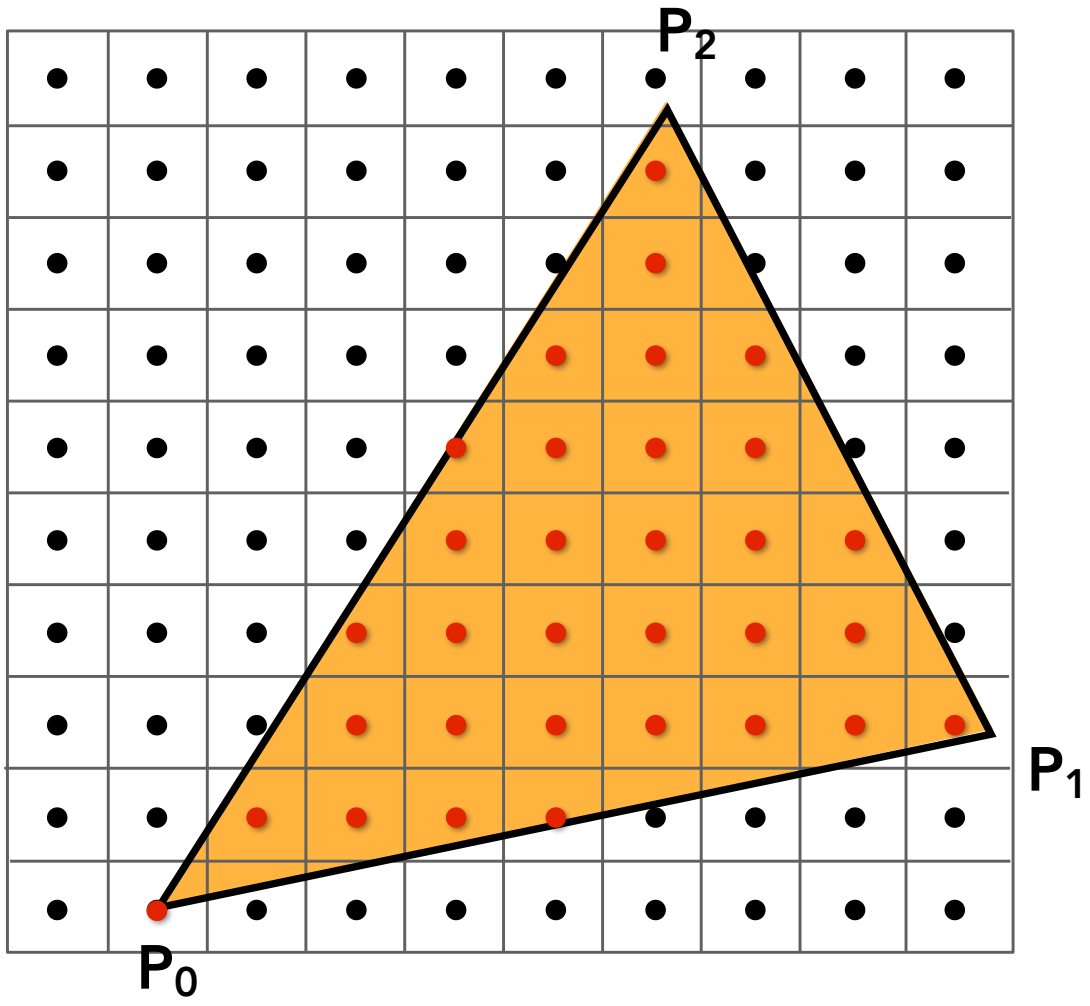
$L_2(x, y) > 0$

Point-in-Triangle Test: Three Line Tests

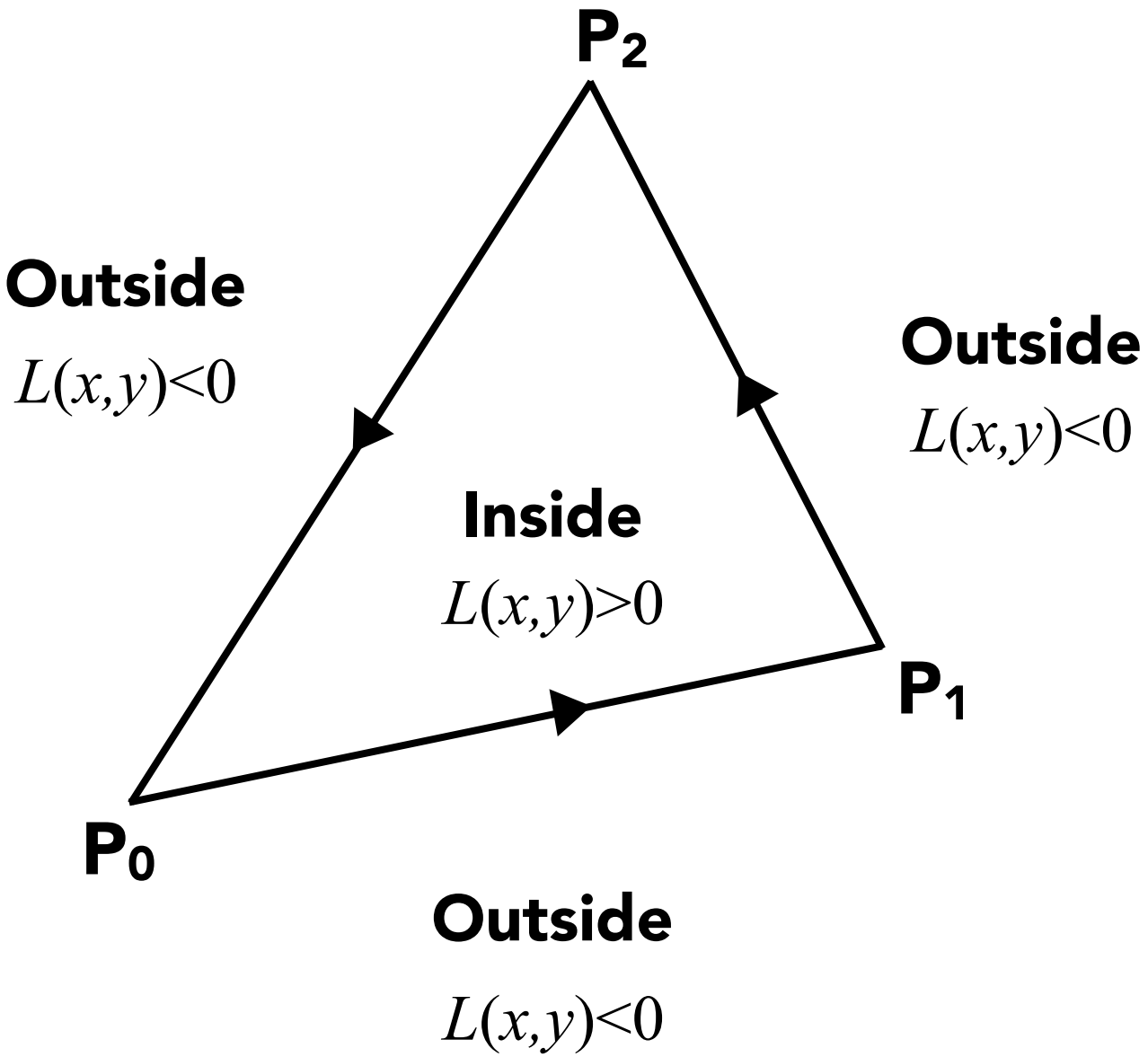
Sample point $s = (sx, sy)$ **is inside the triangle if it is inside all three lines.**

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

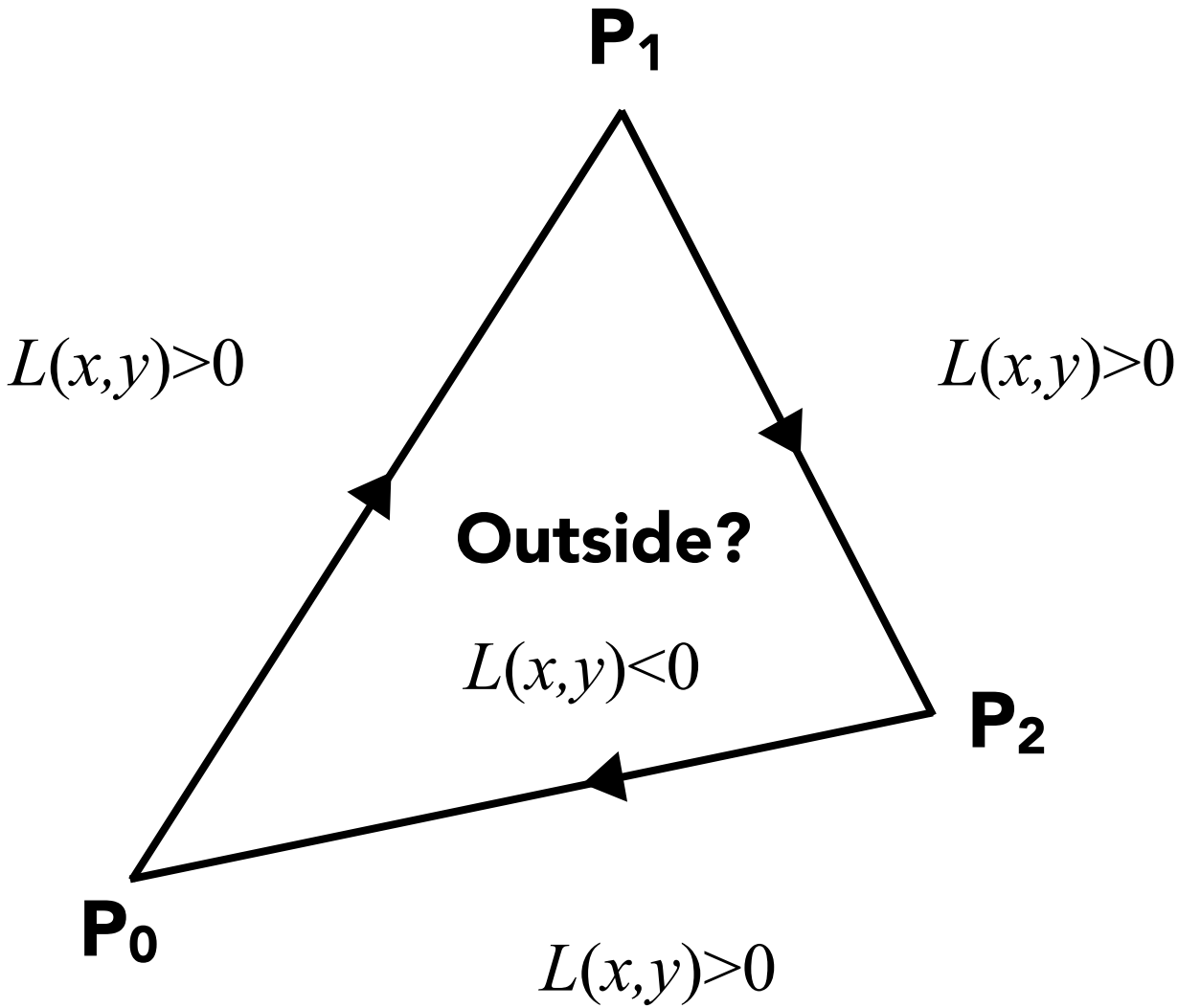
Note: actual implementation of $inside(sx,sy)$ **involves ≤ checks based on edge rules**



Triangle Orientation



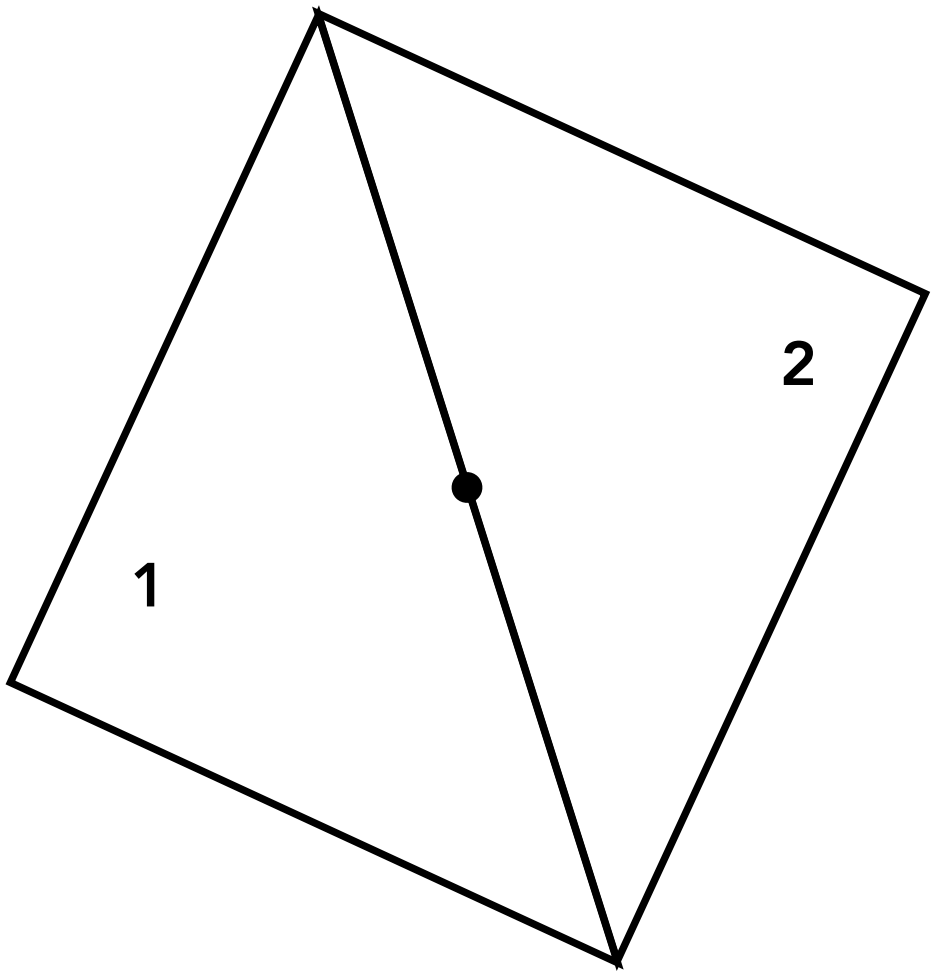
Triangle Orientation



Some Details

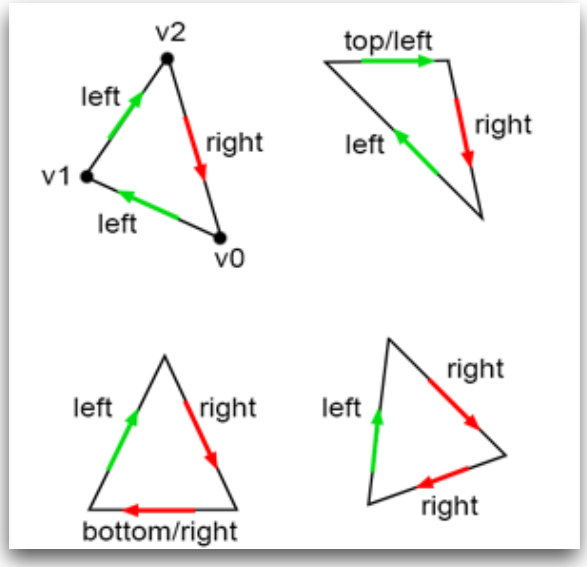
Edge Cases (Literally)

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



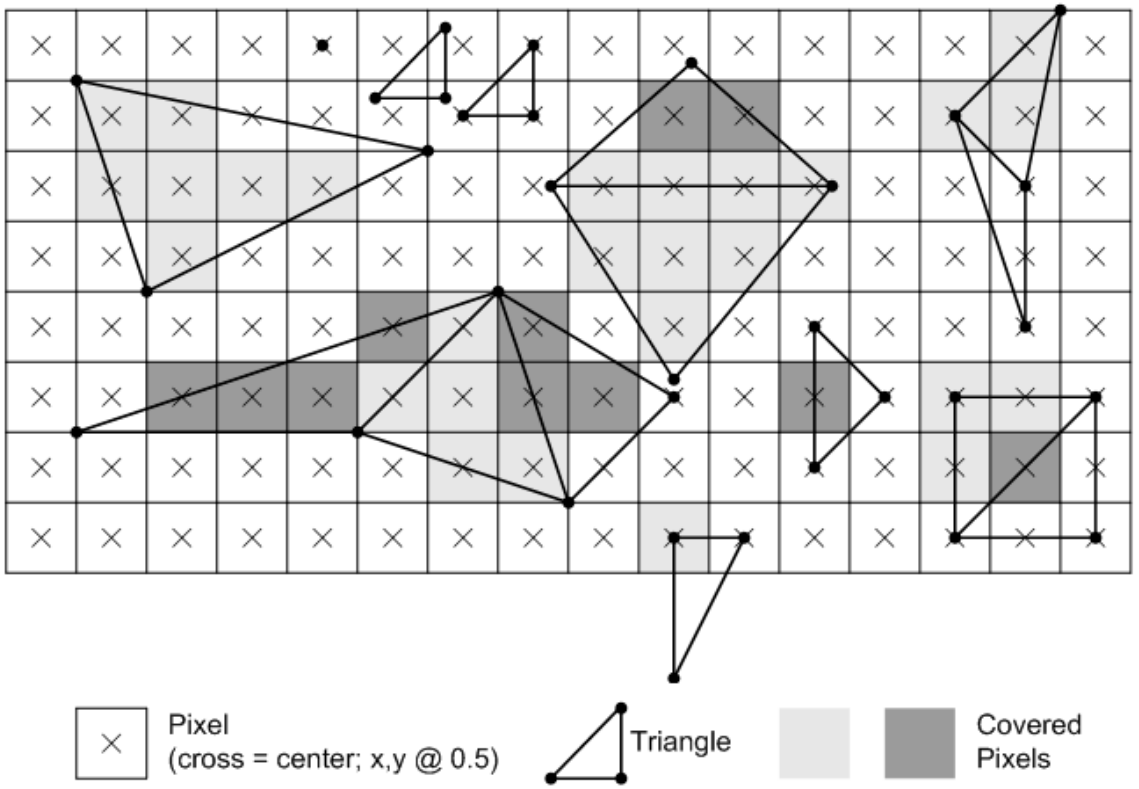
Rasterization Rules

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



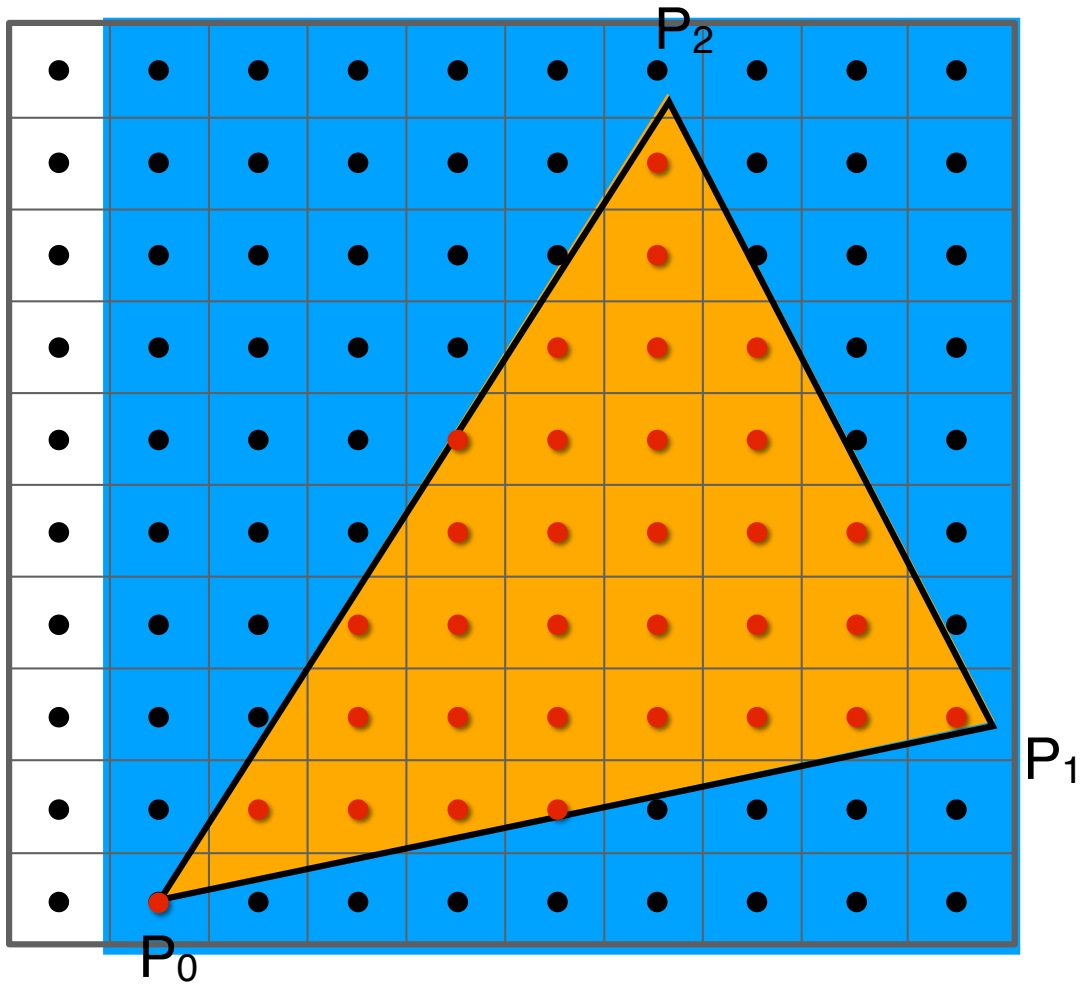
Top edge: horizontal edge that is above all other edges

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)



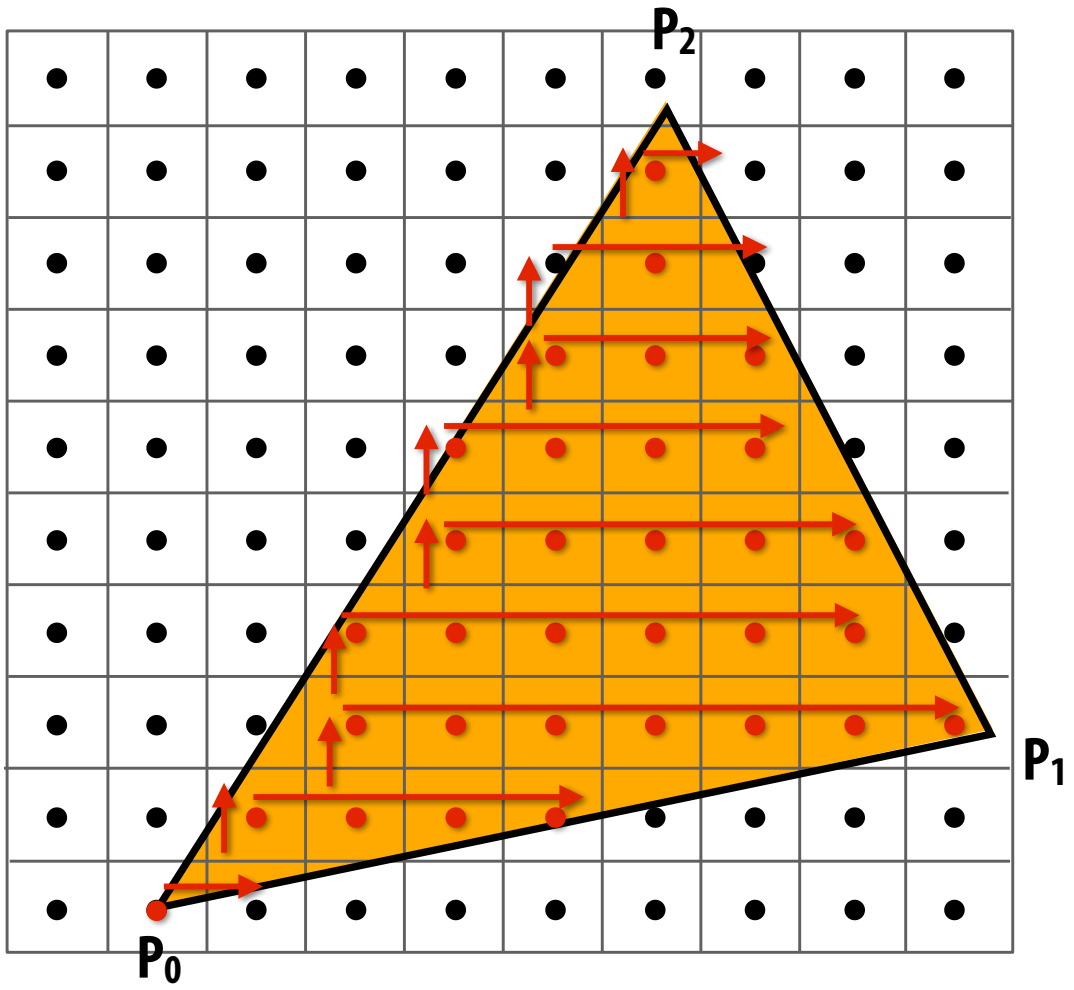
Source: Direct3D Programming Guide, Microsoft

Checking All Pixels on the Screen?



Use a Bounding Box!

Incremental Triangle Traversal (Faster?)



Modern Approach: Tiled Triangle Traversal

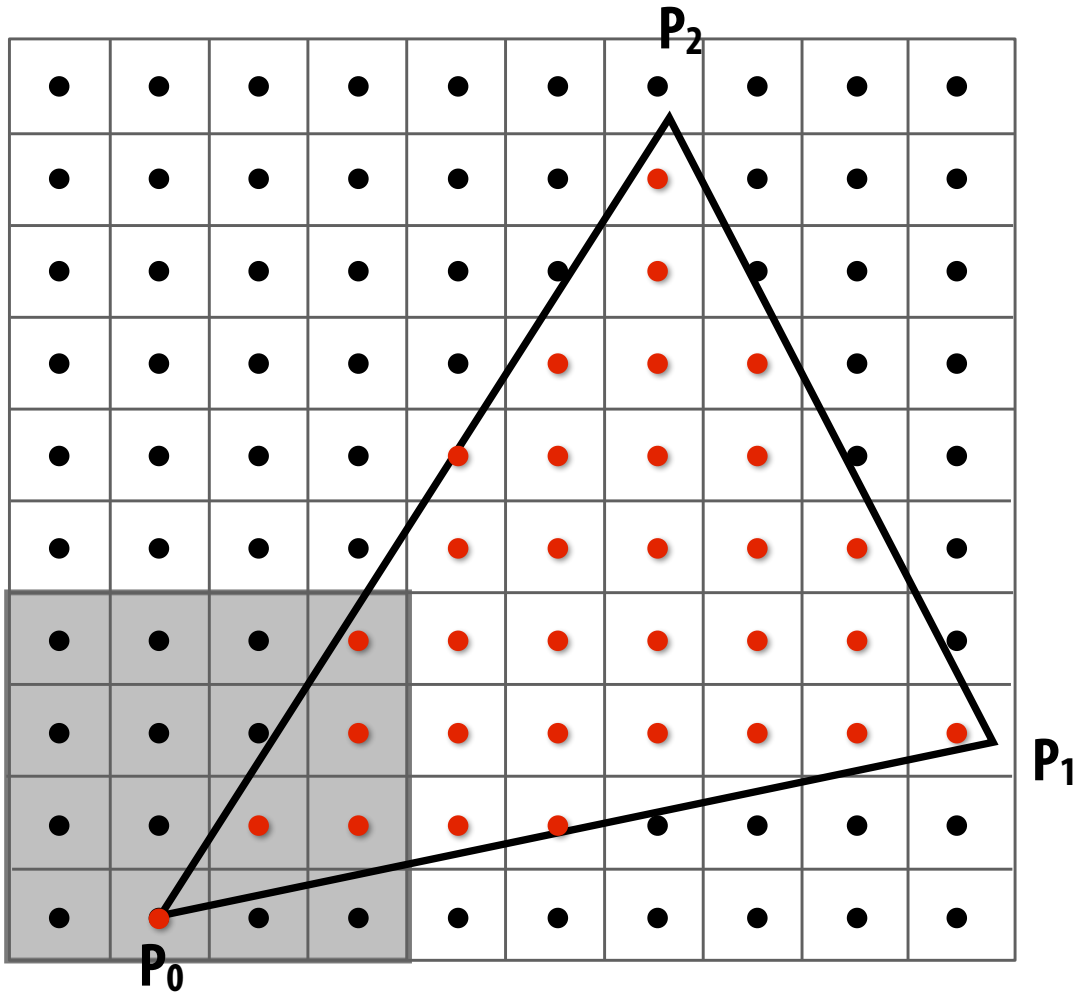
Traverse triangle in blocks

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

Today's Topics

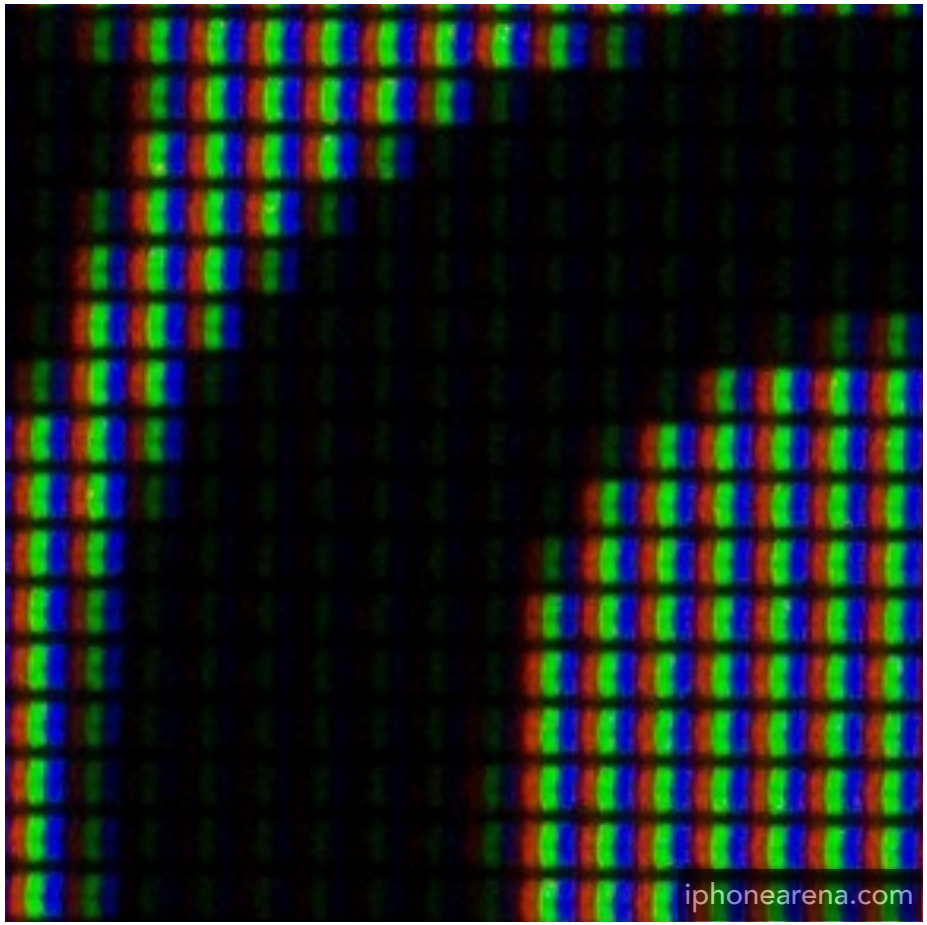
Drawing Machines

Drawing Triangles to Raster Displays

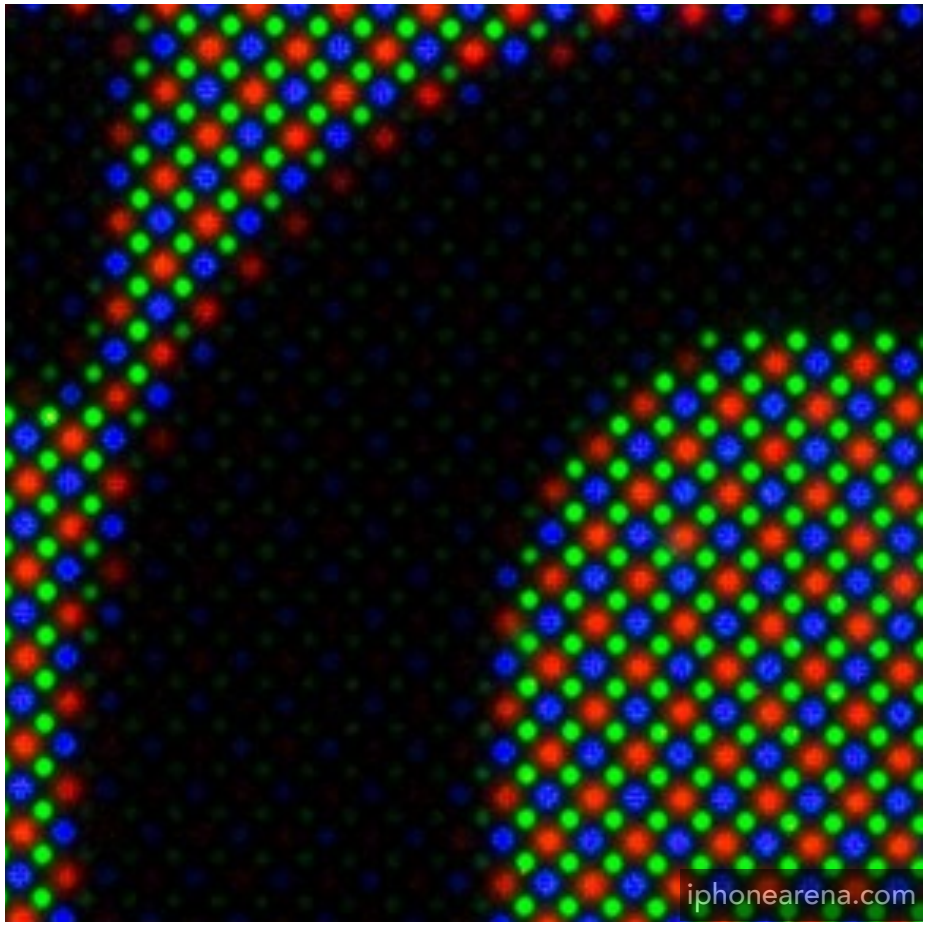
Signal Reconstruction on Real Displays

Sampling and Aliasing

Real LCD Screen Pixels (Closeup)



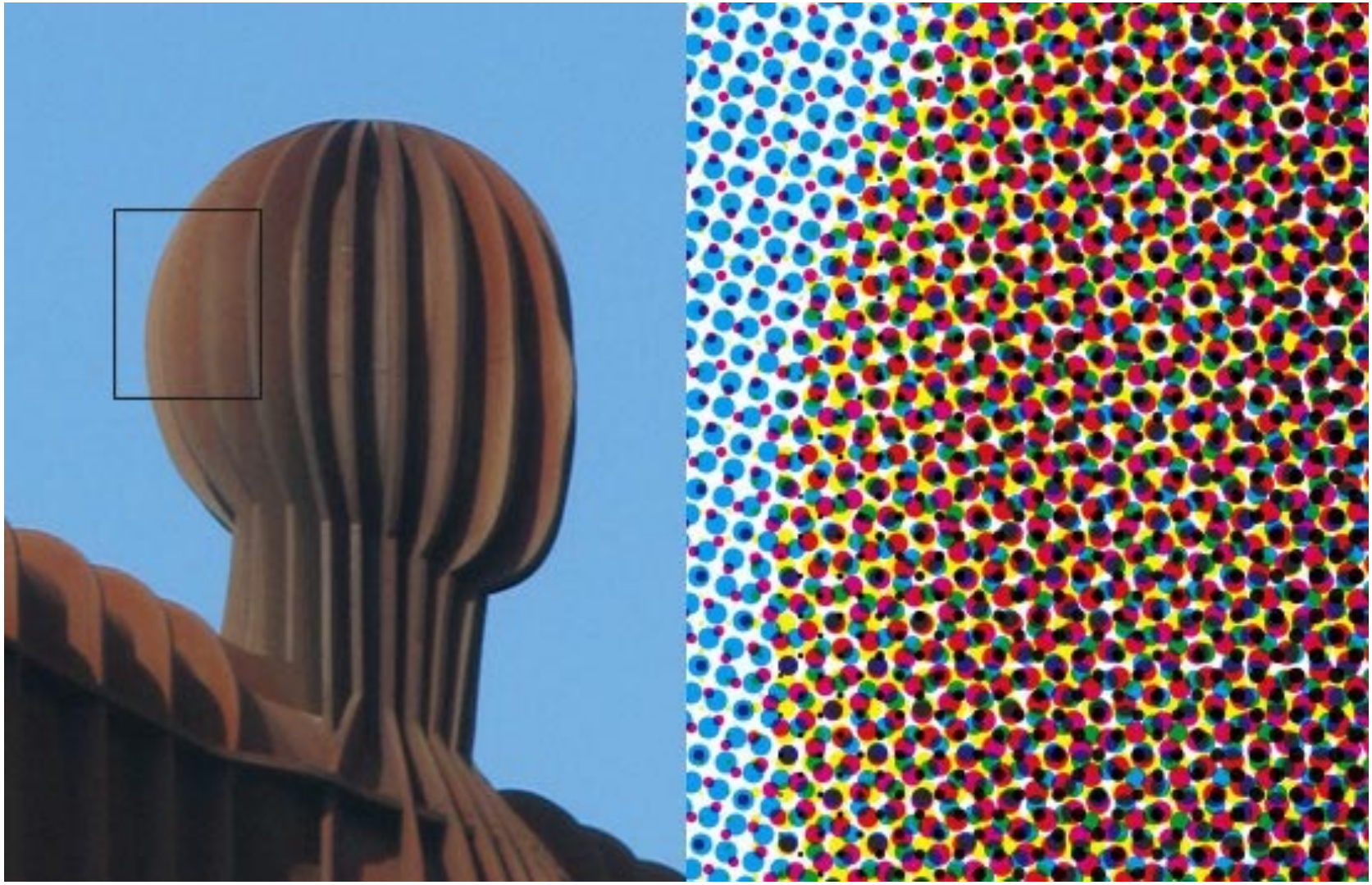
iPhone 6S



Galaxy S5

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

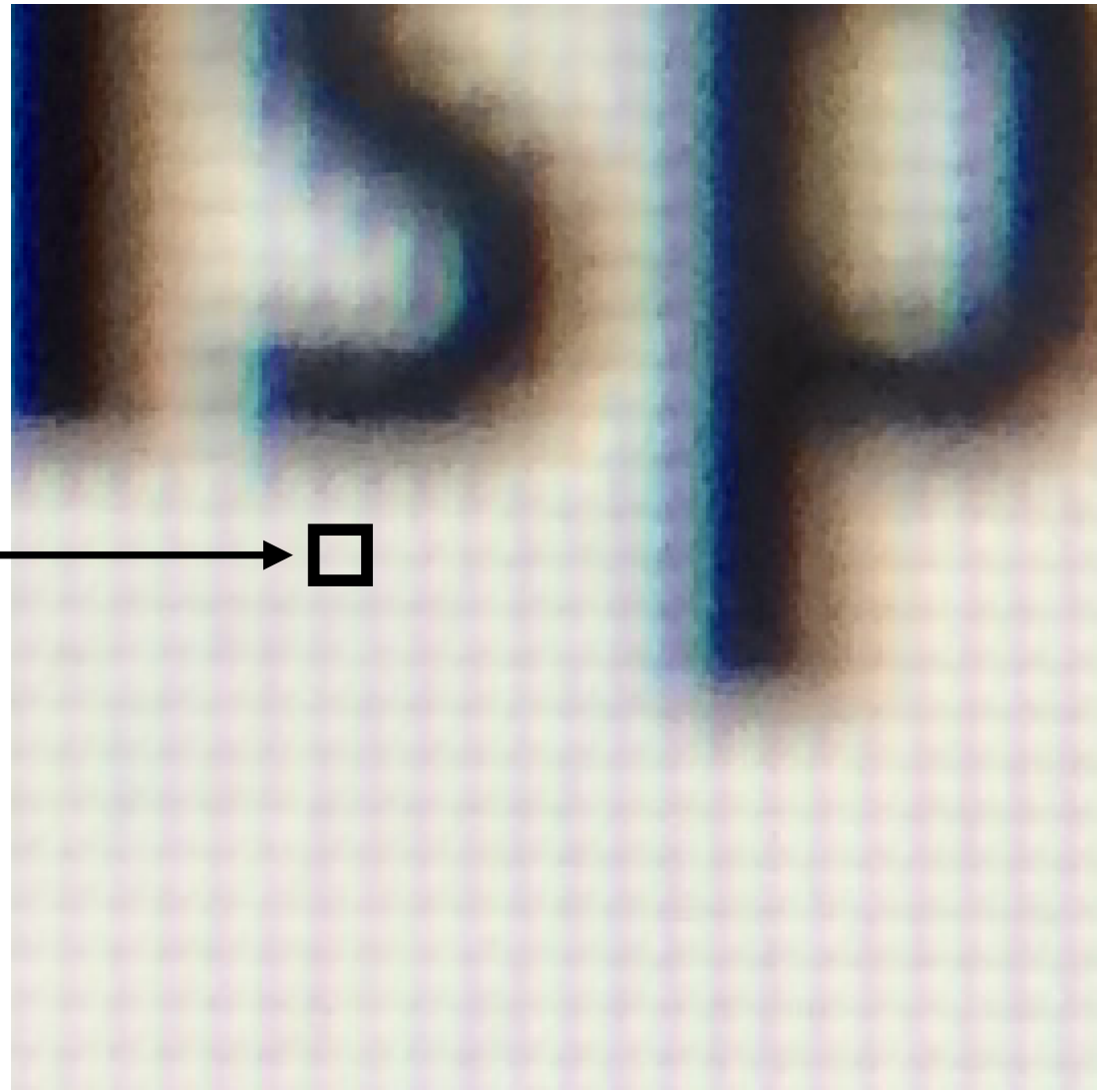
Aside: What About Other Display Methods?



Color print: observe half-tone pattern

Assume Display Pixels Emit Square of Light

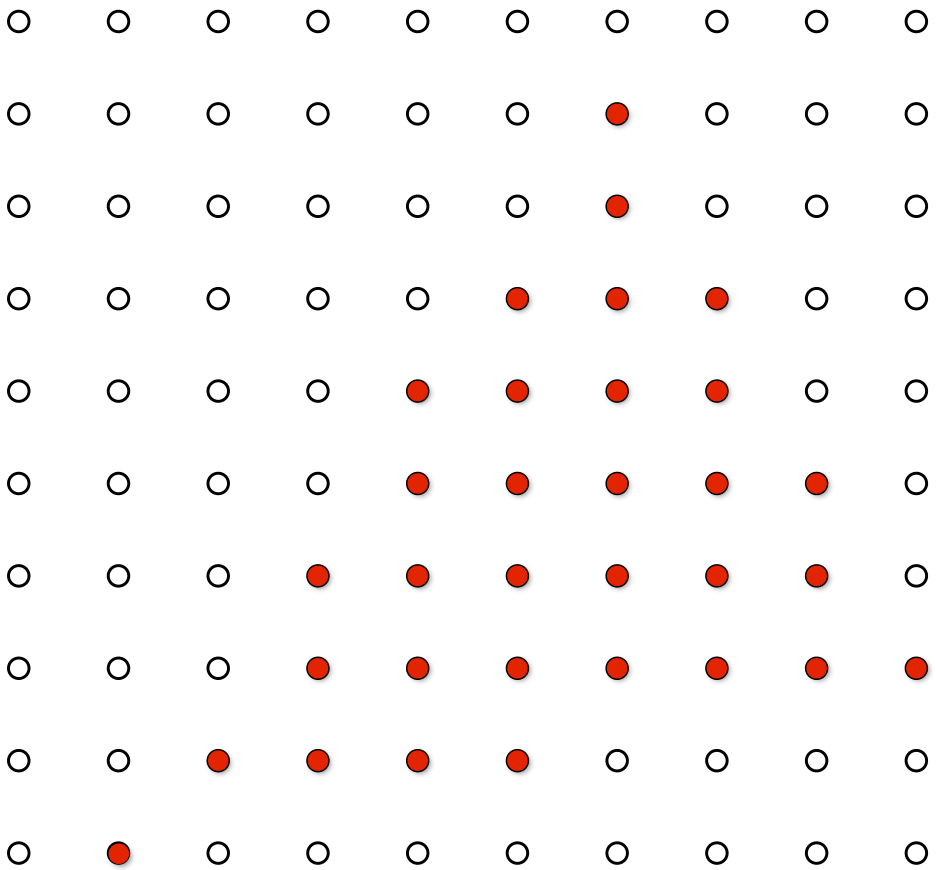
Each image sample sent to the display is converted into a little square of light of the appropriate color: (a pixel = picture element)



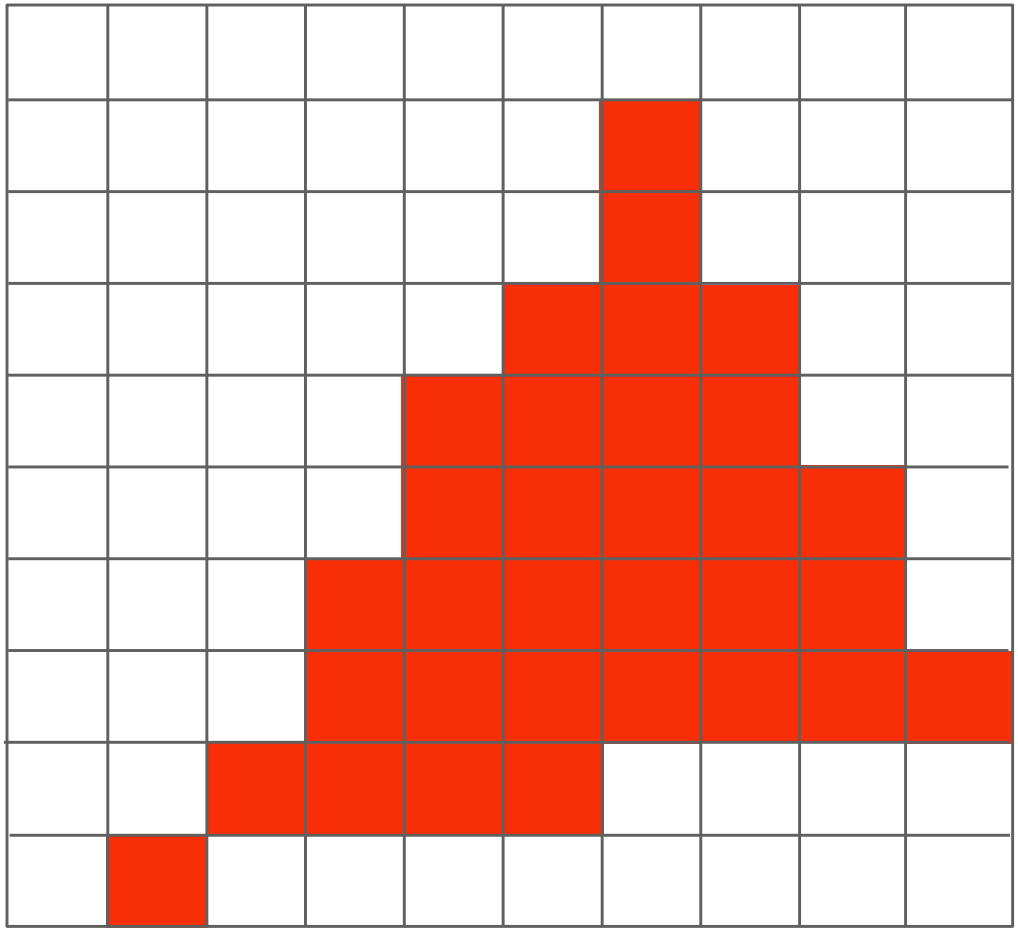
LCD pixel on laptop

* 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:

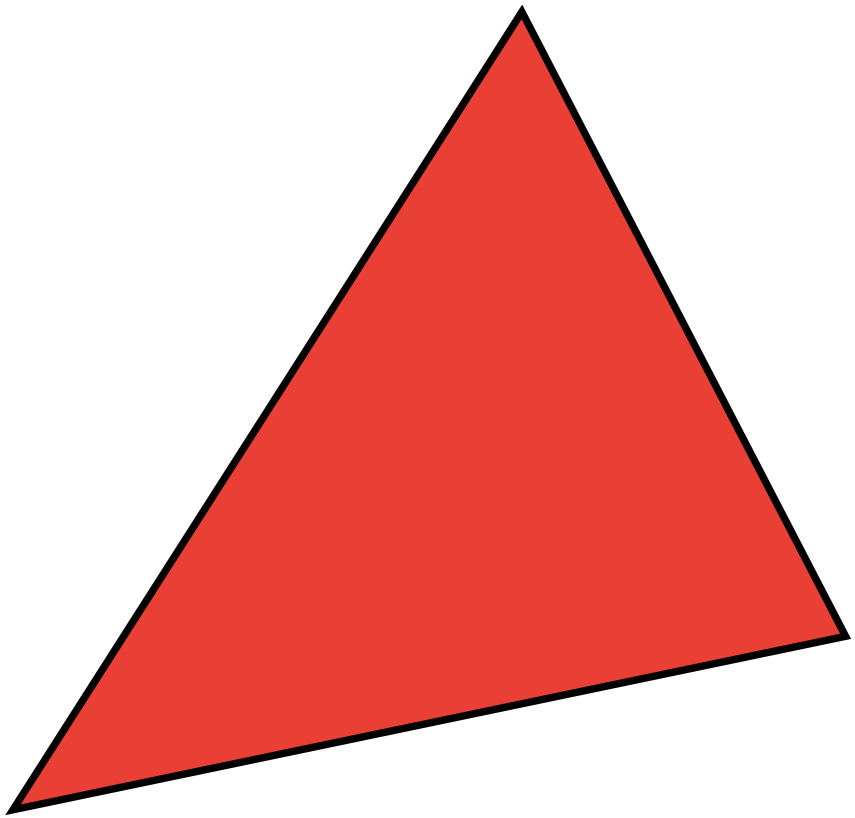


We see this:

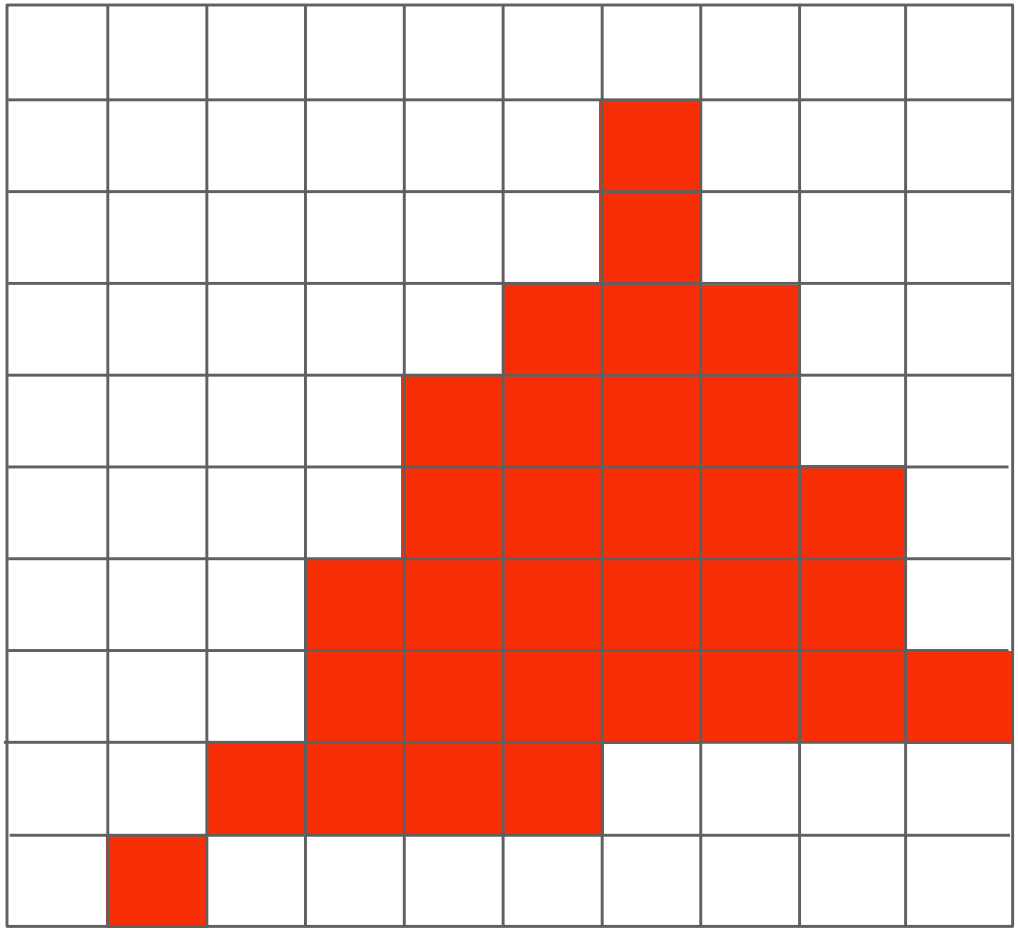


(assuming a screen pixel emits a square of perfectly uniform intensity of light)

Compare: The Continuous Triangle Function



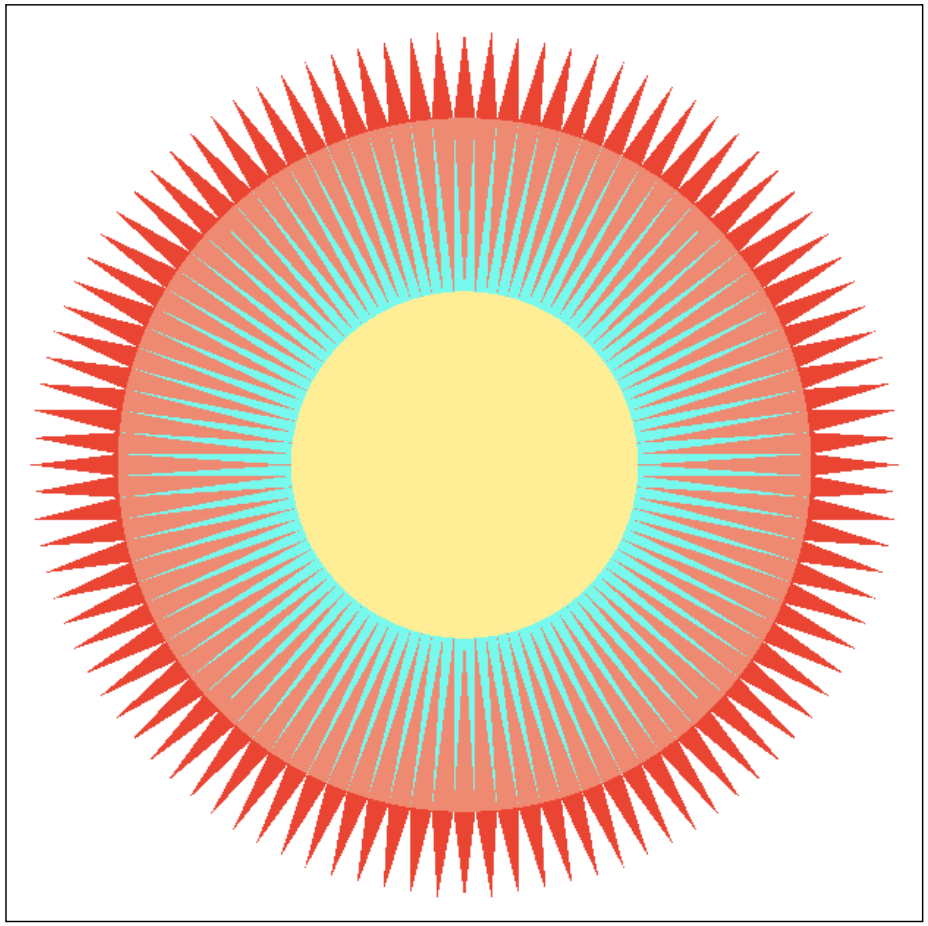
What's Wrong With This Picture?

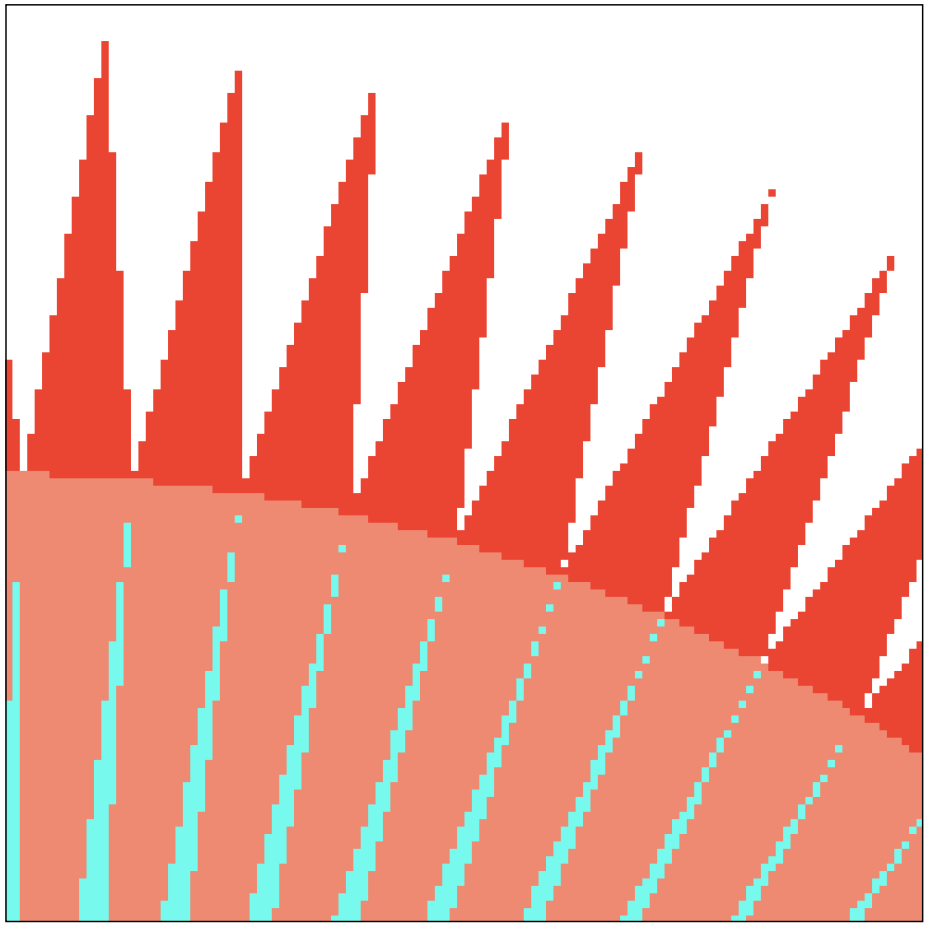


Jaggies!

锯齿

Jaggies (Staircase Pattern)





Is this the best we can do?

How can we avoid Jaggies?

Today's Topics

Drawing Machines

Drawing Triangles to Raster Displays

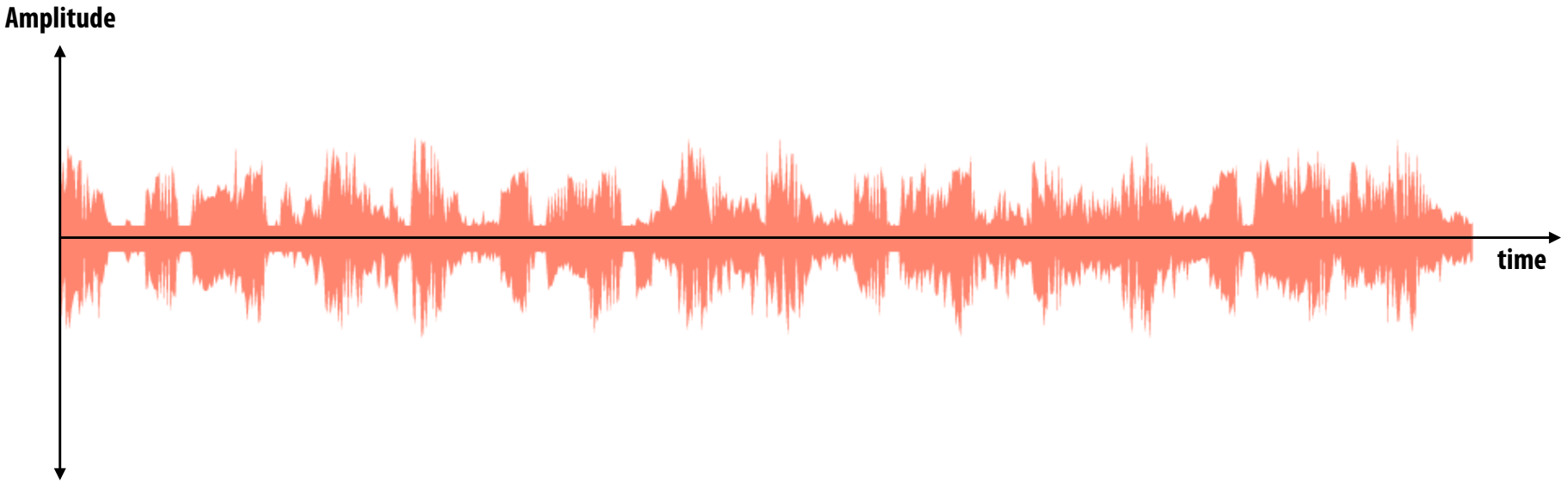
Signal Reconstruction on Real Displays

Sampling and Aliasing

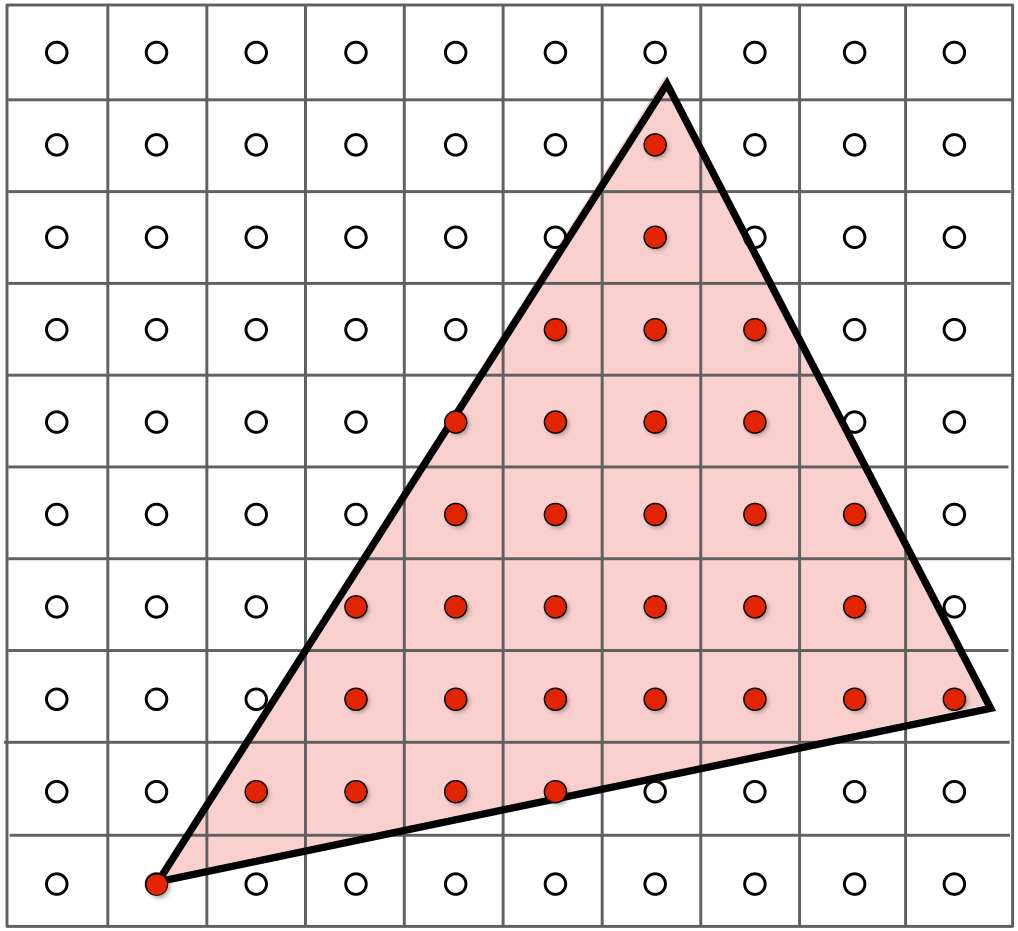
Sampling is Ubiquitous in Computer Graphics

Audio = sample1D position

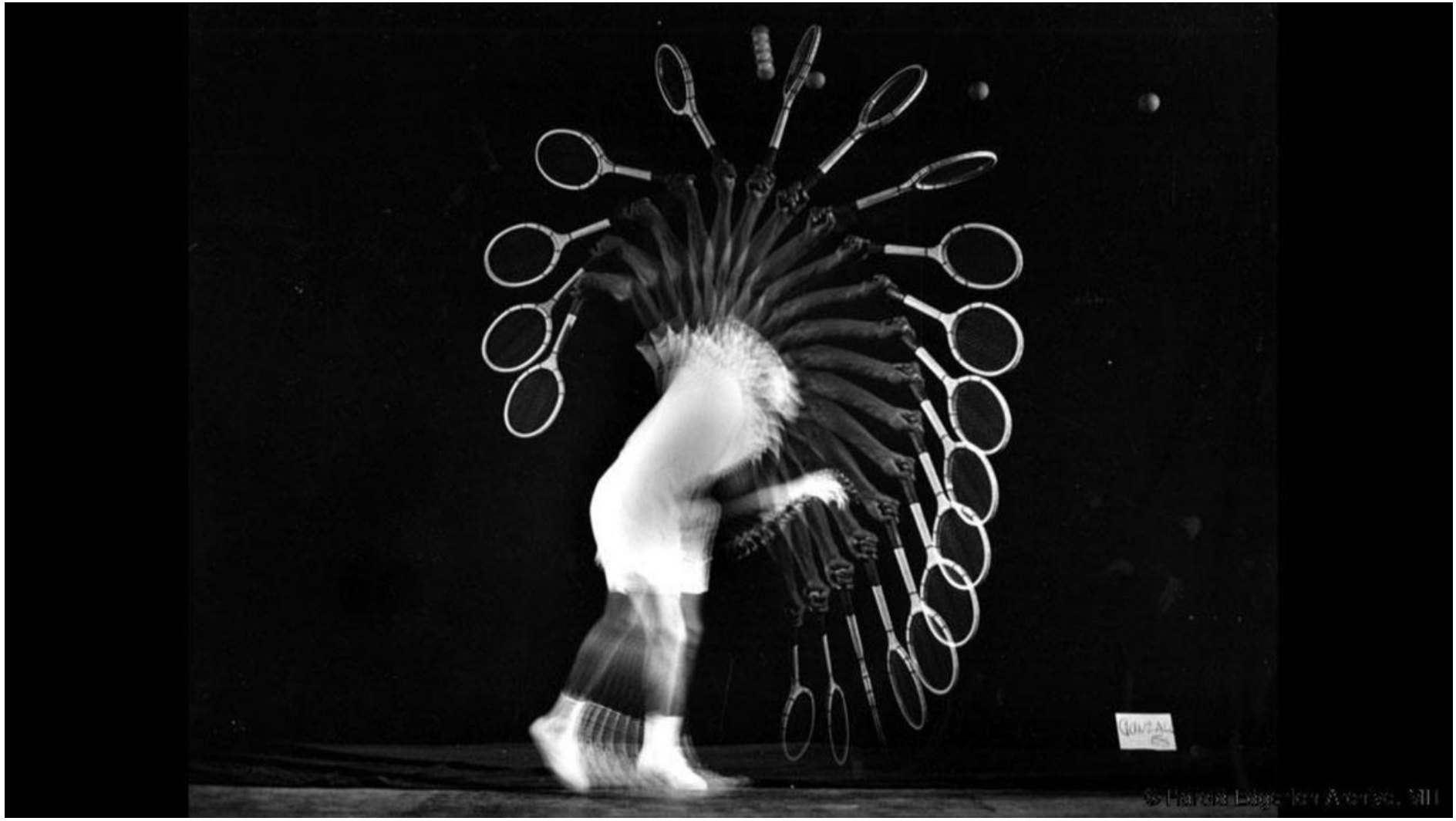
Sampled at 44.1 KHz (44100/second)



Rasterization = Sample 2D Positions

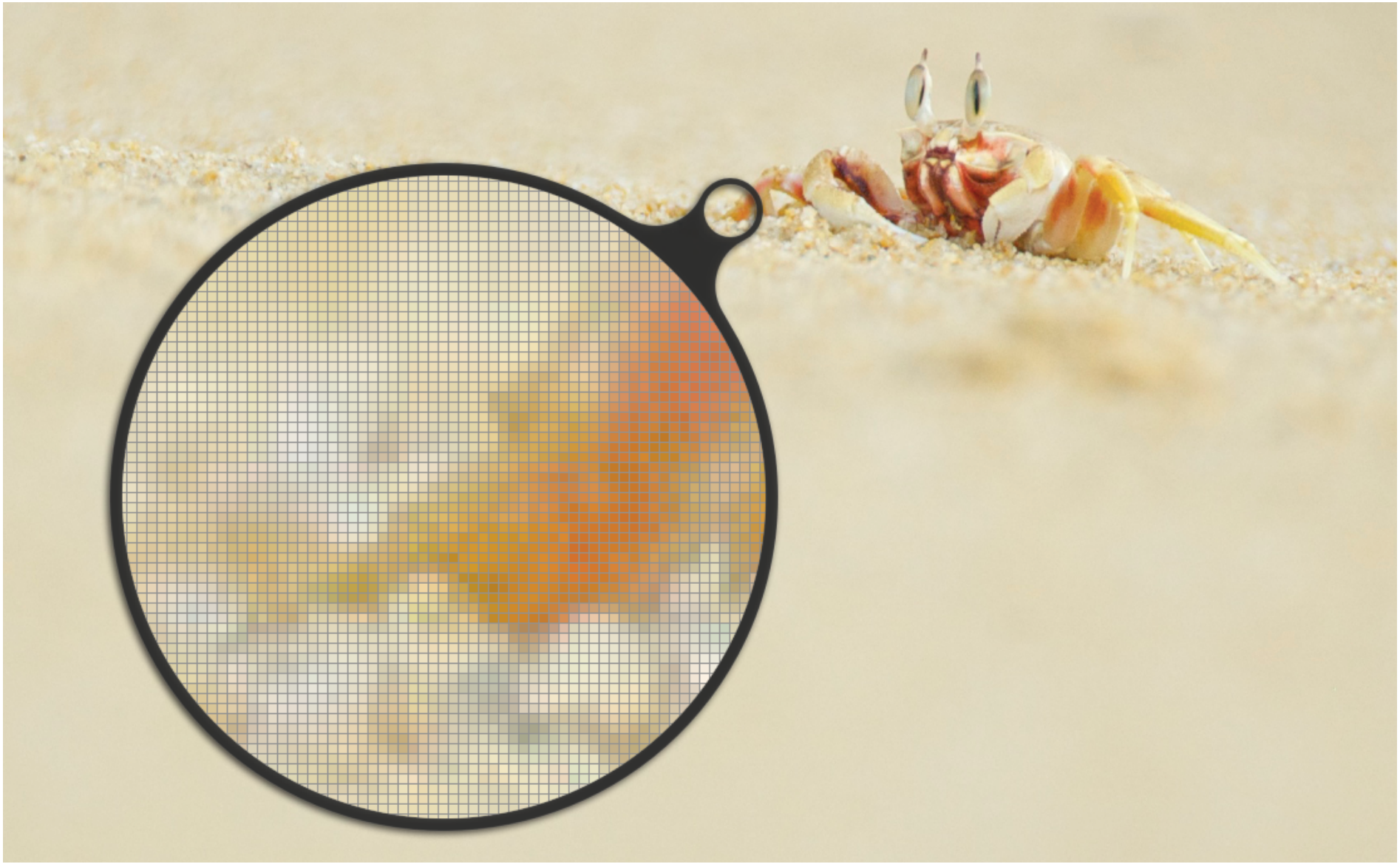


Video = Sample Time



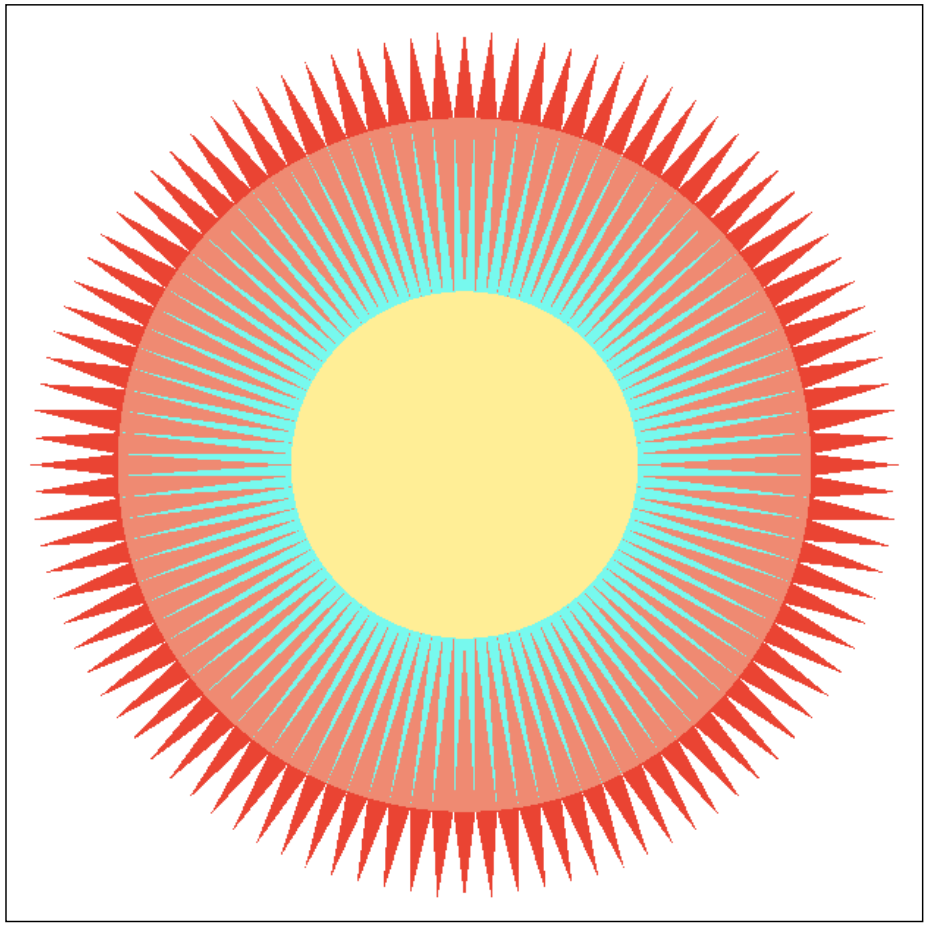
Harold Edgerton Archive, MIT

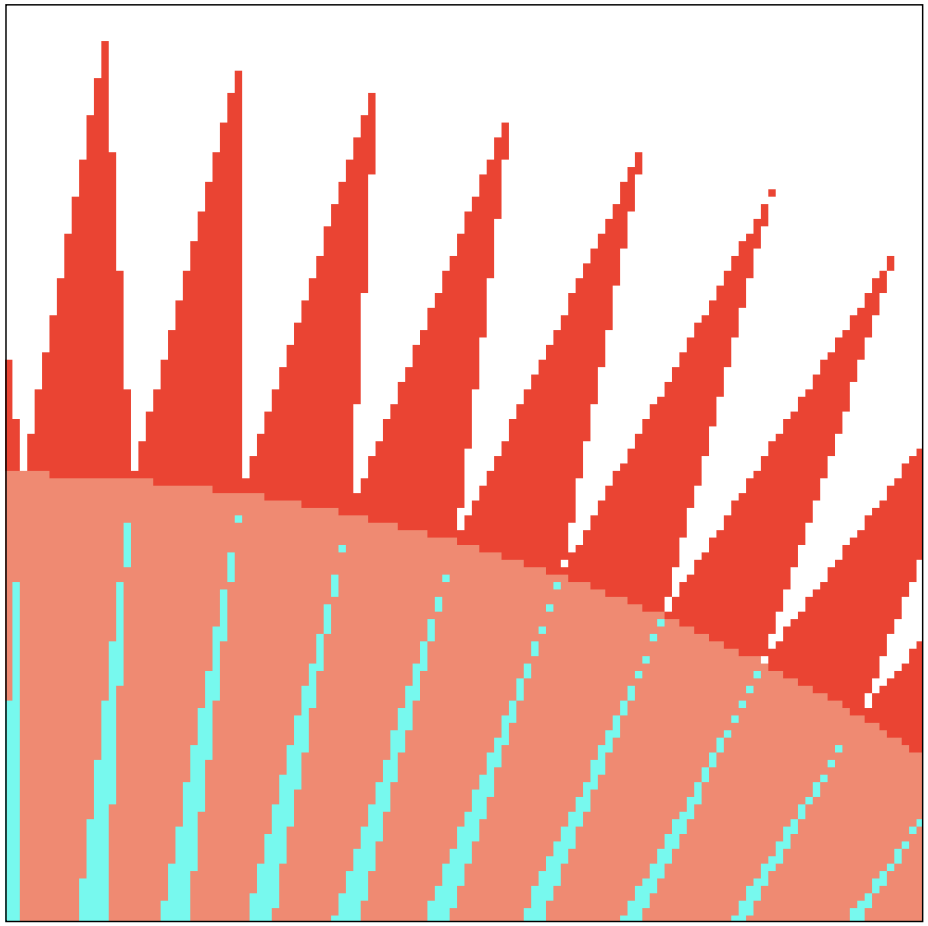
Photograph = Sample Image Sensor Plane



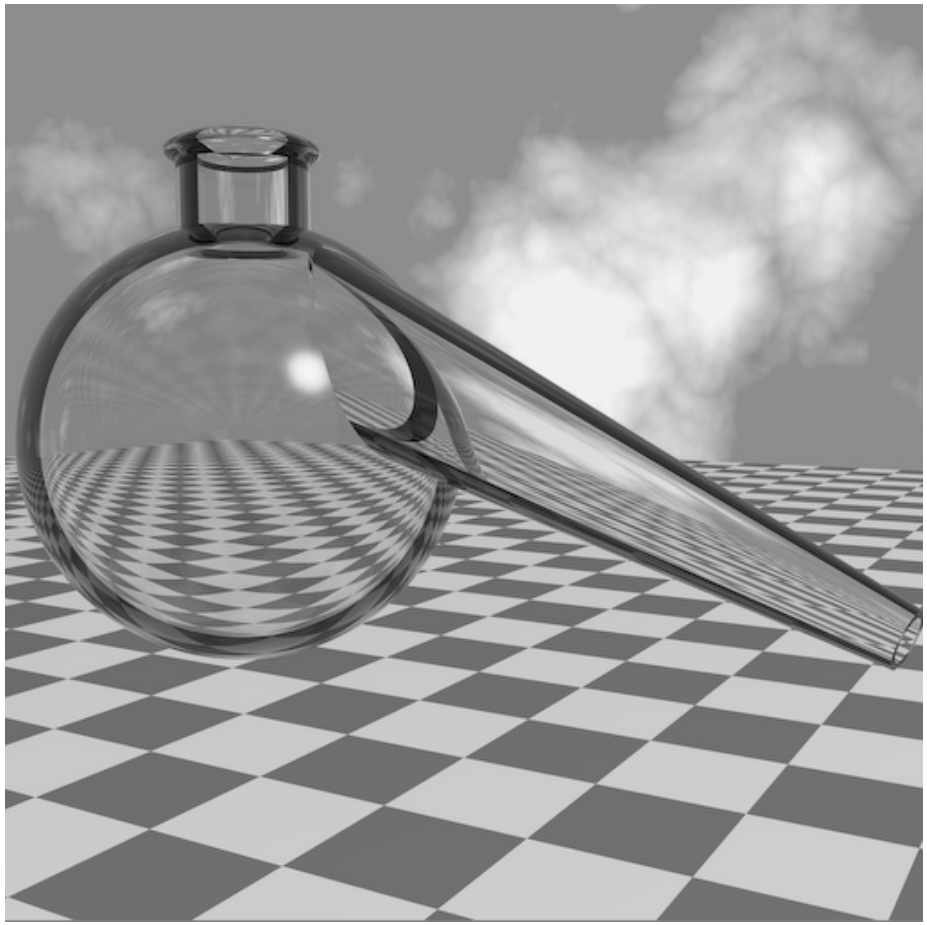
Sampling Artifacts in Graphics and Imaging

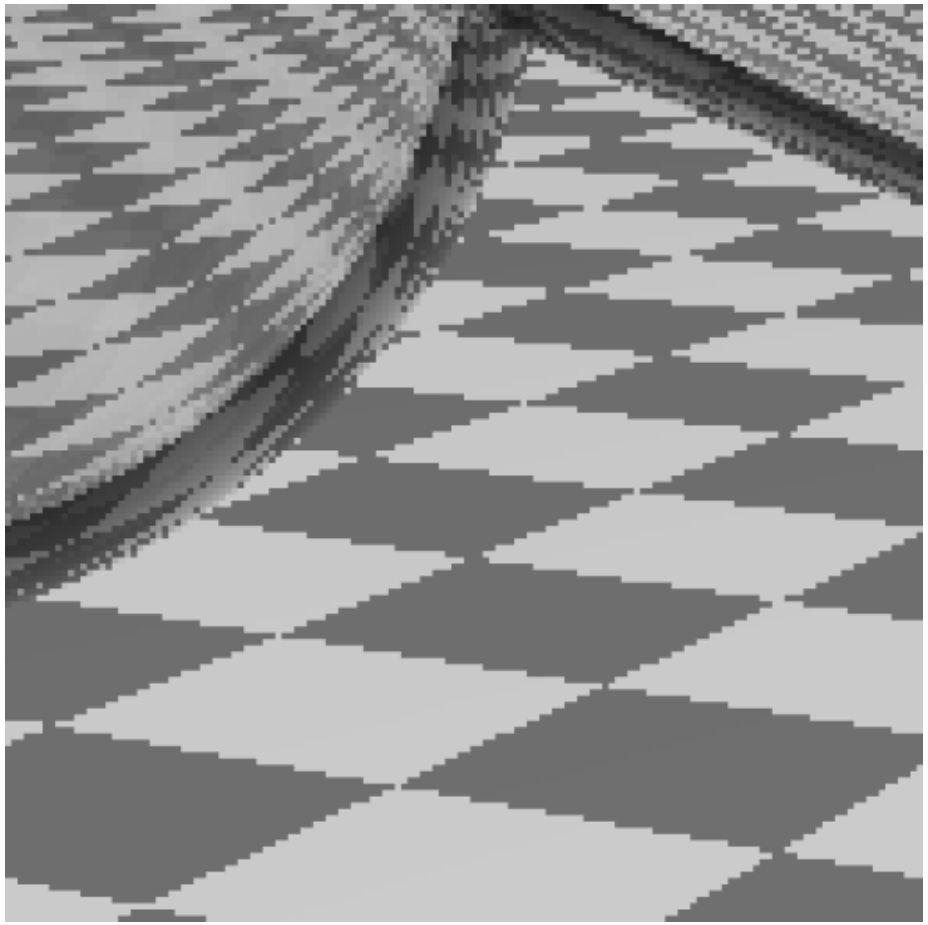
Jaggies (Staircase Pattern)





Jaggies (Staircase Pattern)





Retort by Don Mitchell

Moiré Patterns in Imaging

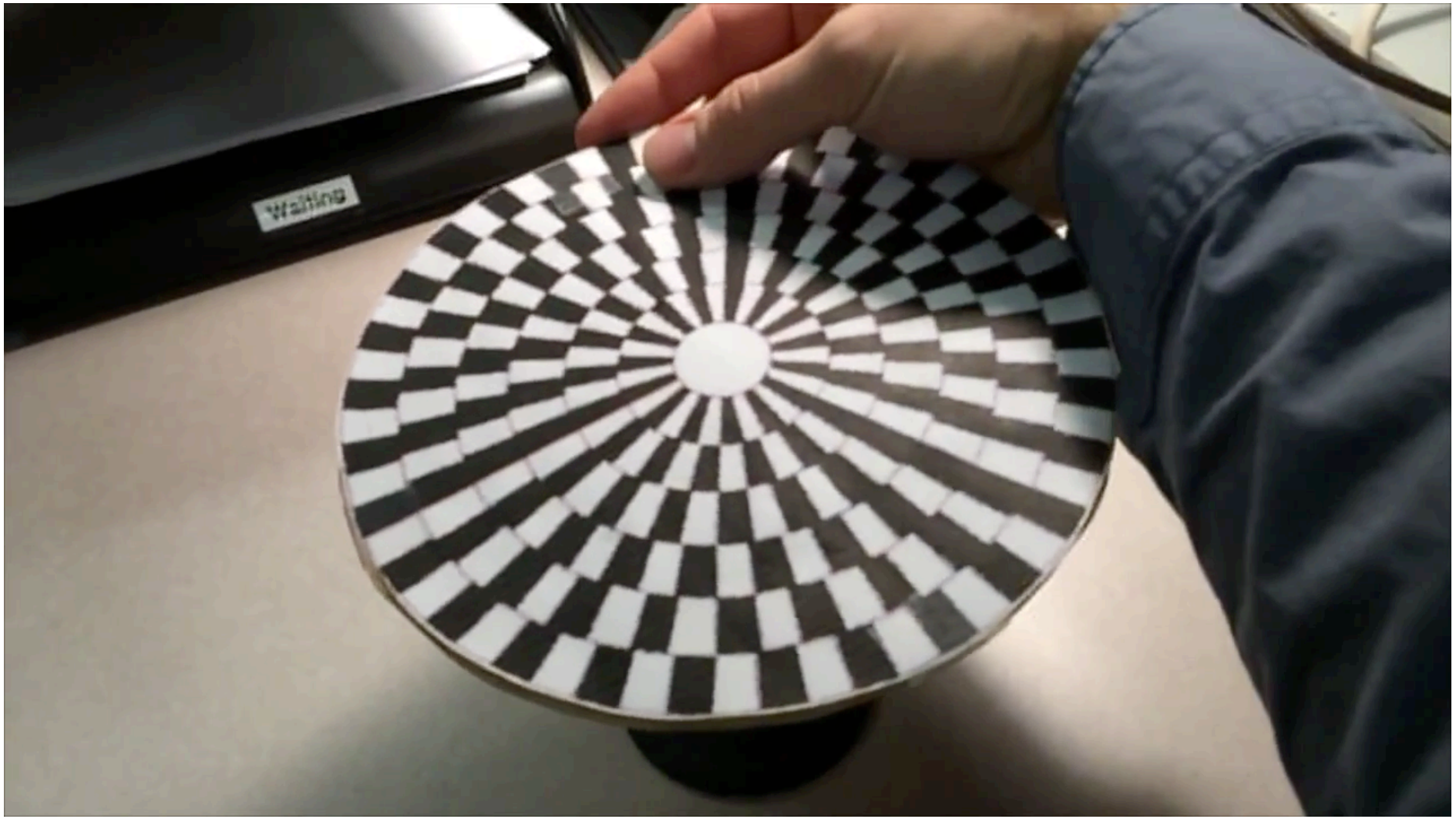
Read every sensor pixel

Skip odd rows and columns

lystit.com

Wagon Wheel Illusion (False Motion)

瓦格纳轮效应



Created by Jesse Mason, https://www.youtube.com/watch?v=QOwzkND_ooU

Sampling Artifacts in Computer Graphics

Artifacts due to sampling - “Aliasing”

- Jaggies – sampling in space
- Wagon wheel effect – sampling in time
- Moire – undersampling images (and texture maps)
- [Many more] …

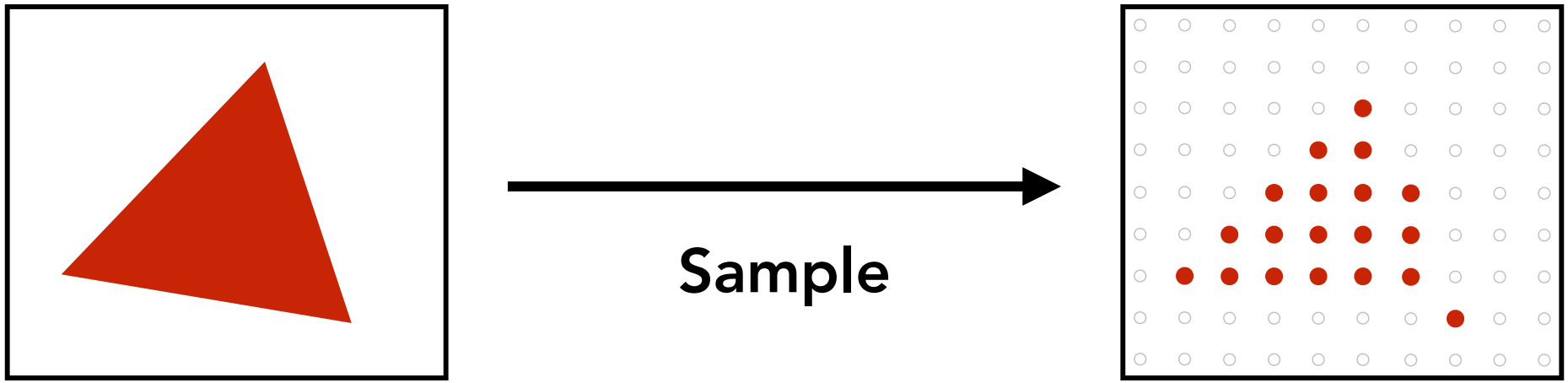
（Aliasing 翻译为混叠或走样，指信号变了样，深层意思是不同频率发生混叠）

We notice this in fast-changing signals (high frequency), when we sample too slowly

（通俗地讲，走样/混叠的原因是信号的变化速率大于采样频率）

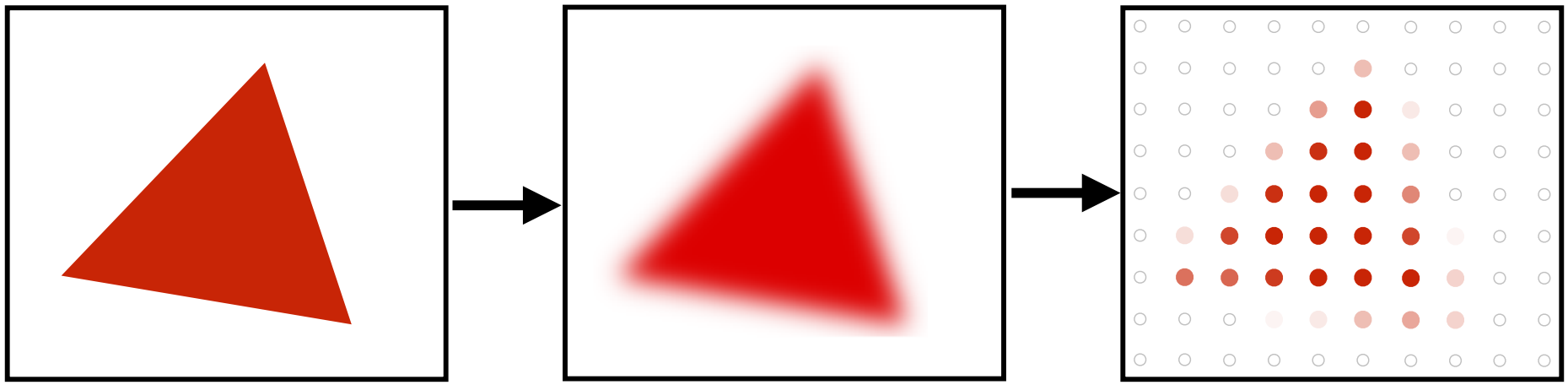
Antialiasing Idea: Filter Out High Frequencies Before Sampling

Rasterization: Point Sampling in Space



Note jaggies in rasterized triangle
where pixel values are pure red or white

Rasterization: Antialiased Sampling



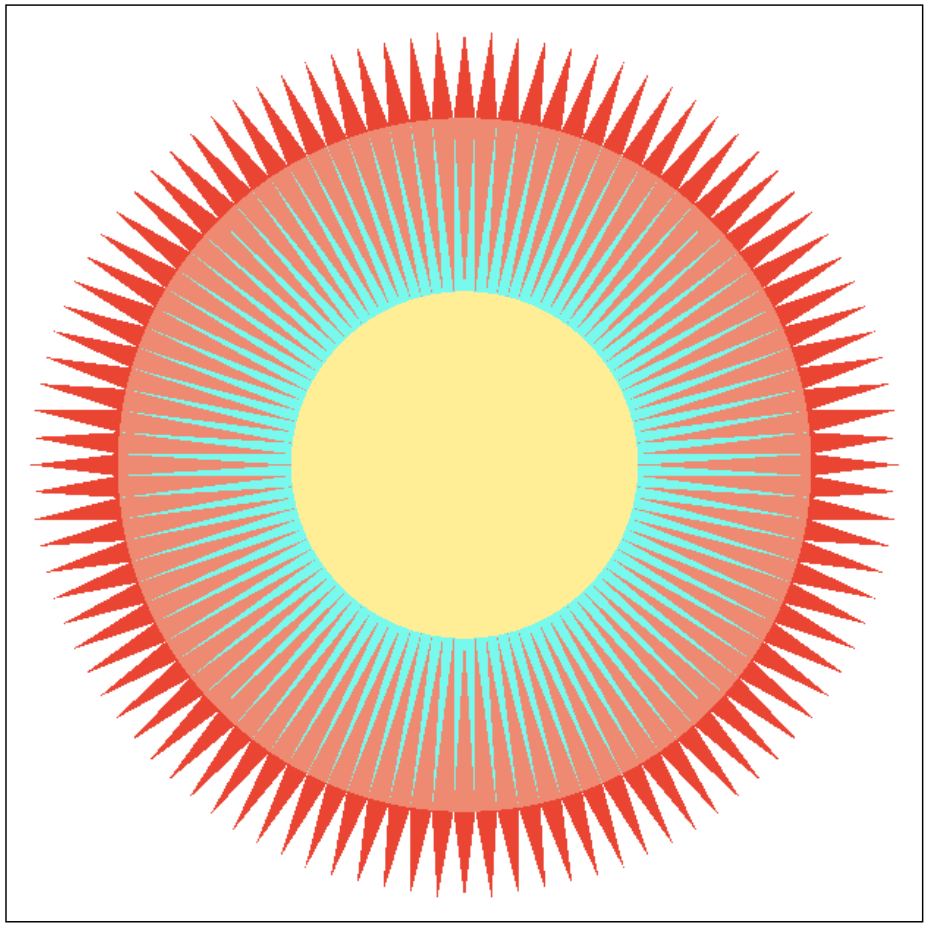
Pre-Filter

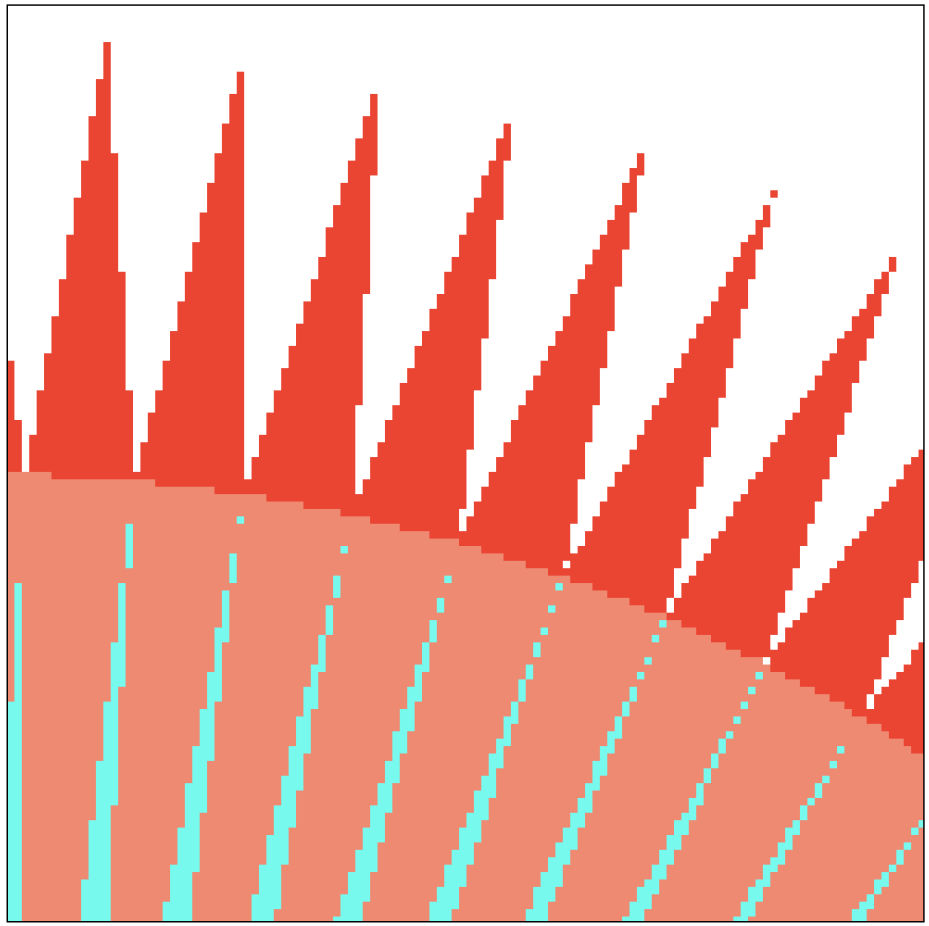
(remove high frequencies)

Sample

Note antialiased edges in rasterized triangle where pixel values take intermediate values

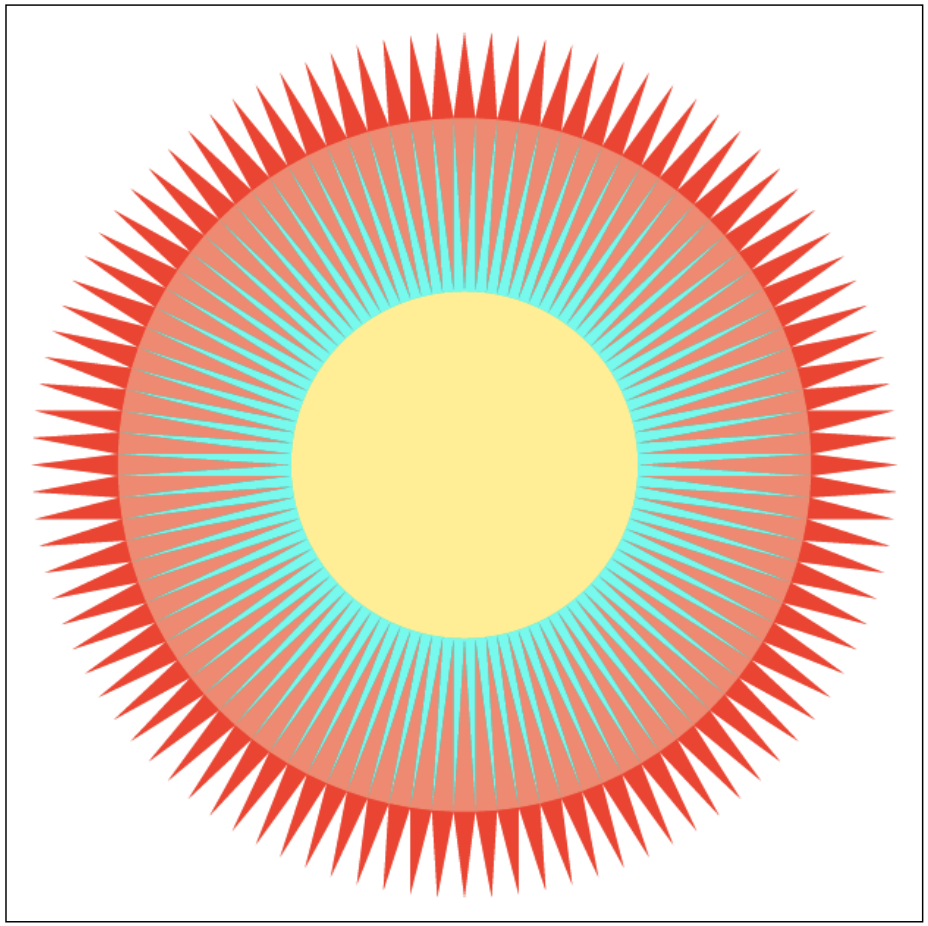
Point Sampling

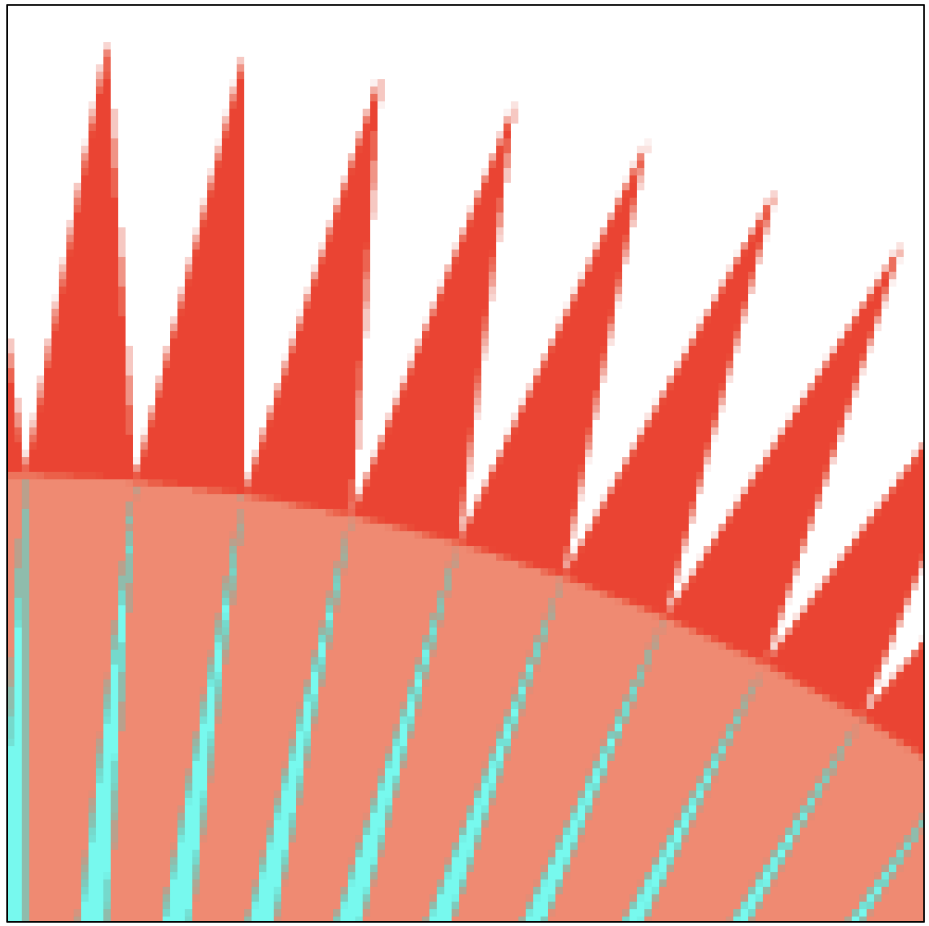




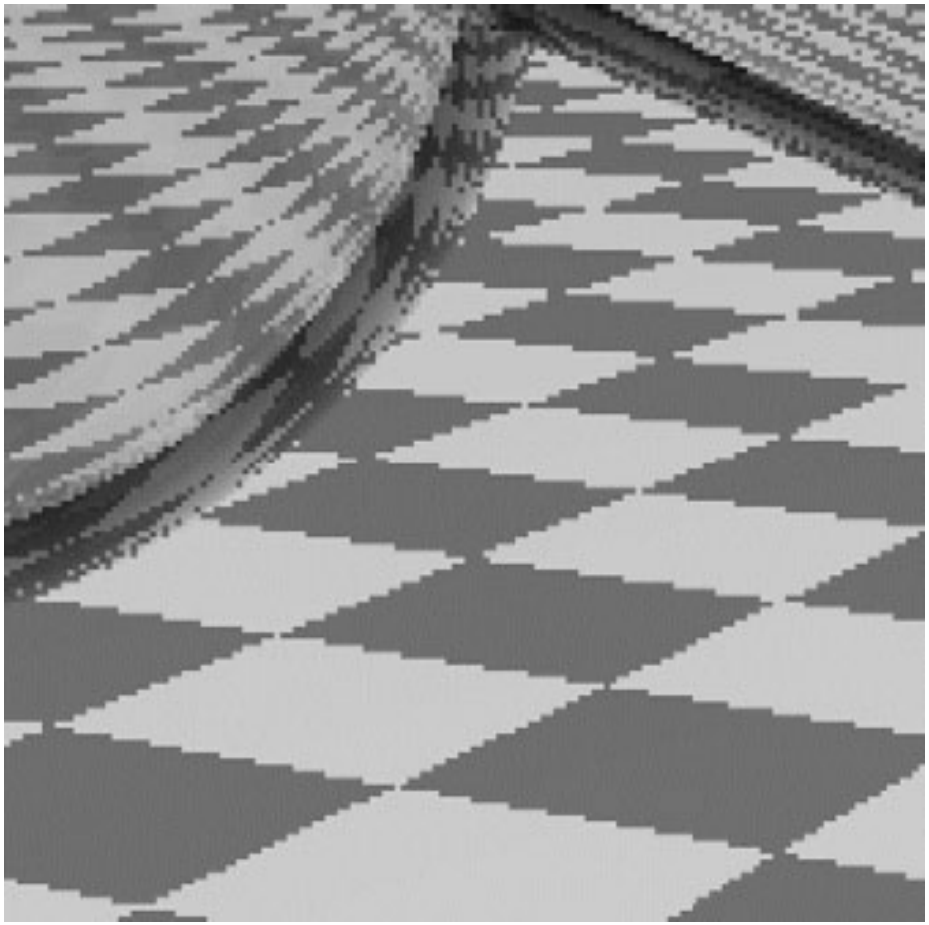
One sample per pixel

Antialiasing

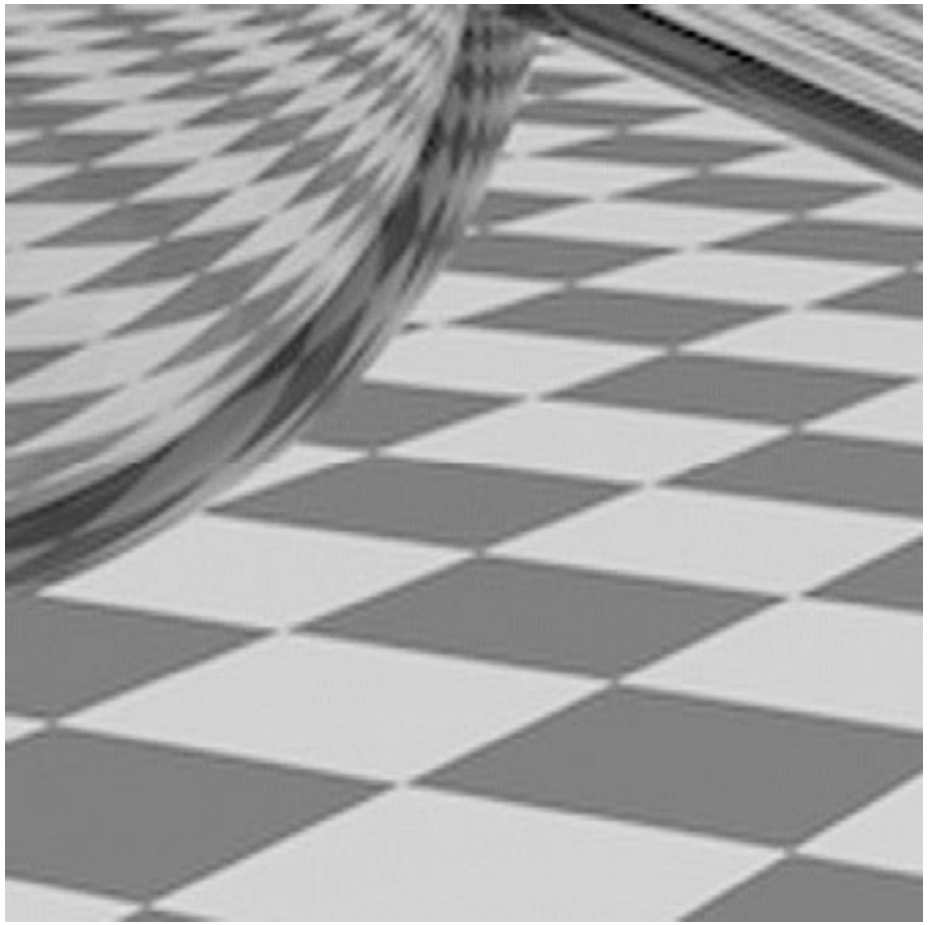




Point Sampling vs Antialiasing

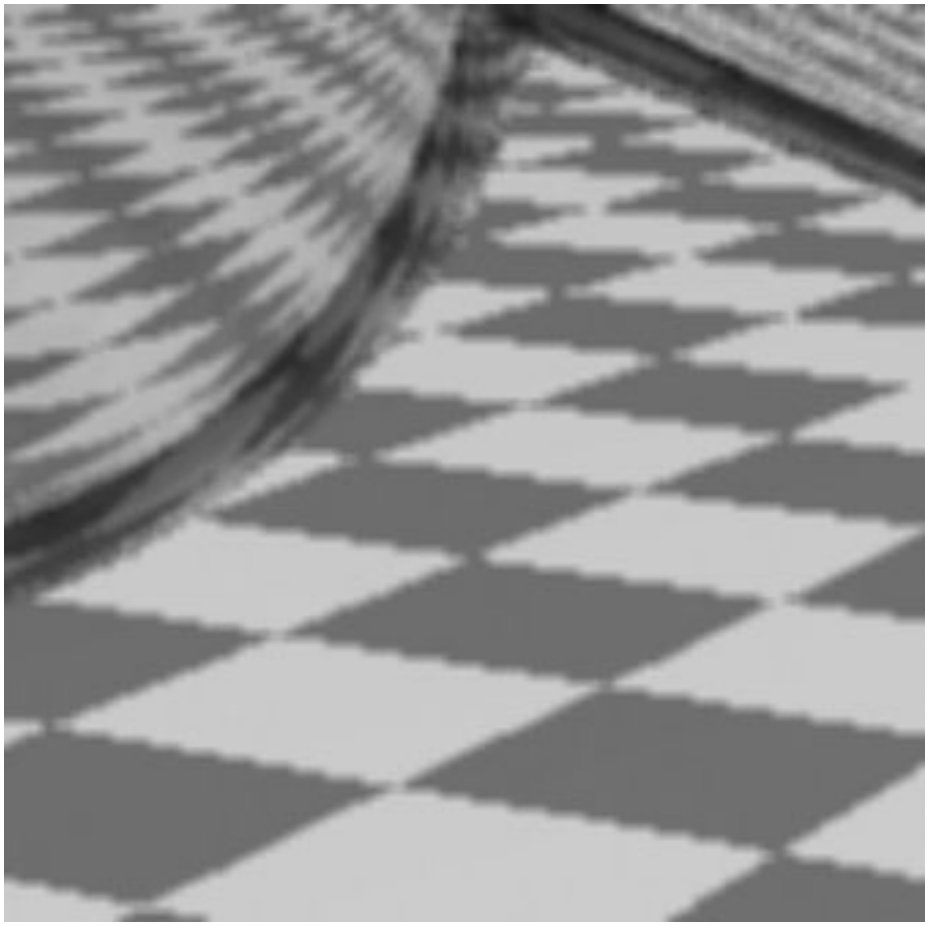


Jaggies

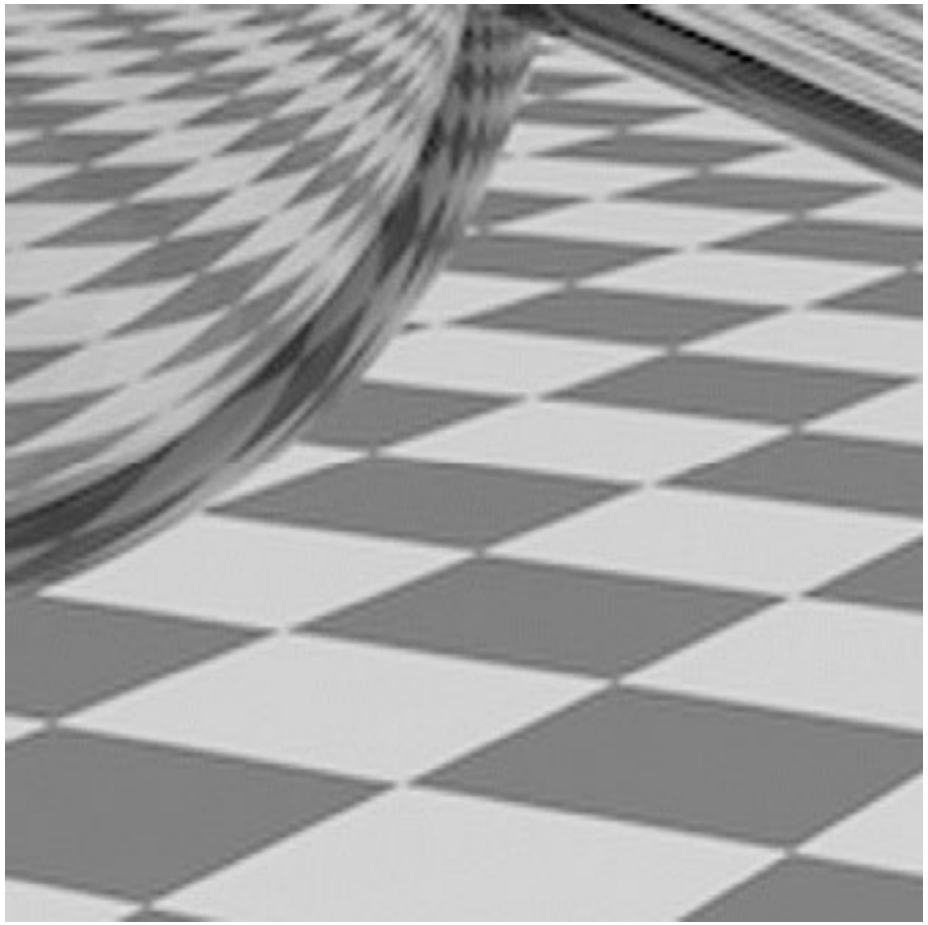


Pre-Filtered

Antialiasing vs Blurred Aliasing



Blurred Jaggies
(Sample then filter)



Pre-Filtered
(Filter then sample)

Next, we will dig into the reason why aliasing happened, and how to implement antialiasing rasterization algorithm.

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)

Things to Remember

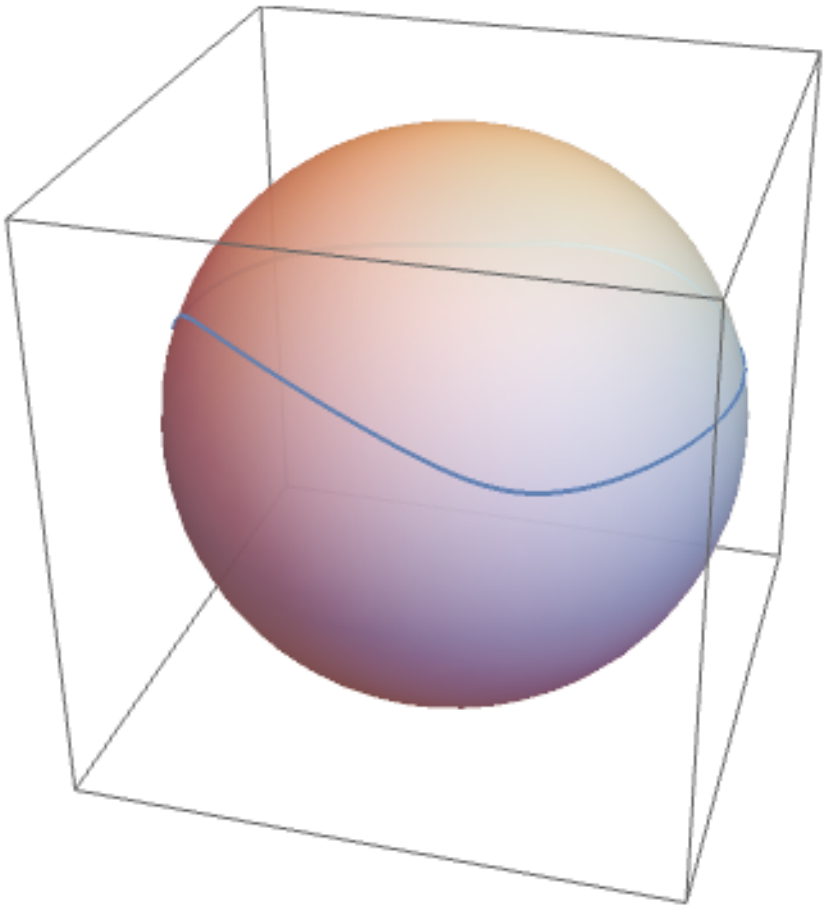
- Aliasing
  - Sampling causes aliasing (artifacts) due to the lower sampling rate compared to the original signal frequency
- Anti-aliasing
  - Pre-filtering removes high frequency from original signal before sampling

Bonus Slides: Orientation

Cut the sphere into two regions with a curve.

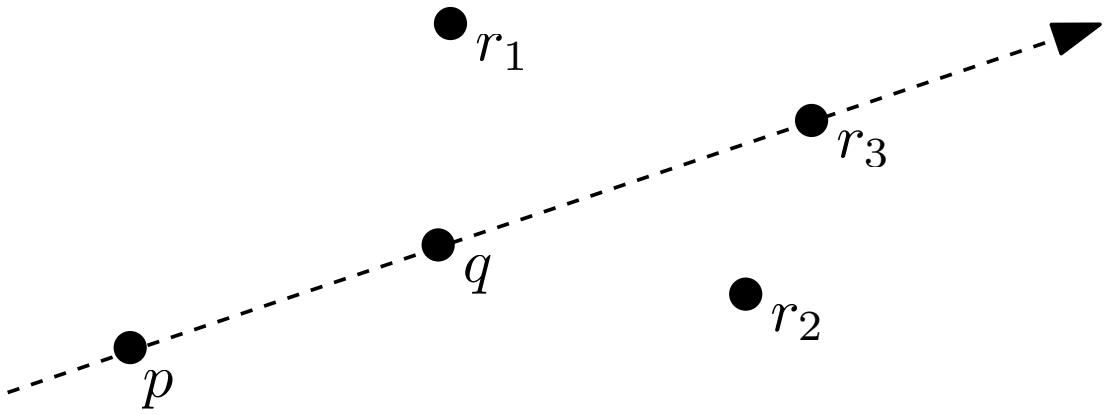
Q: which region is inside the curve?

A: it depends on the orientation of the boundary curve!



Orientation Test

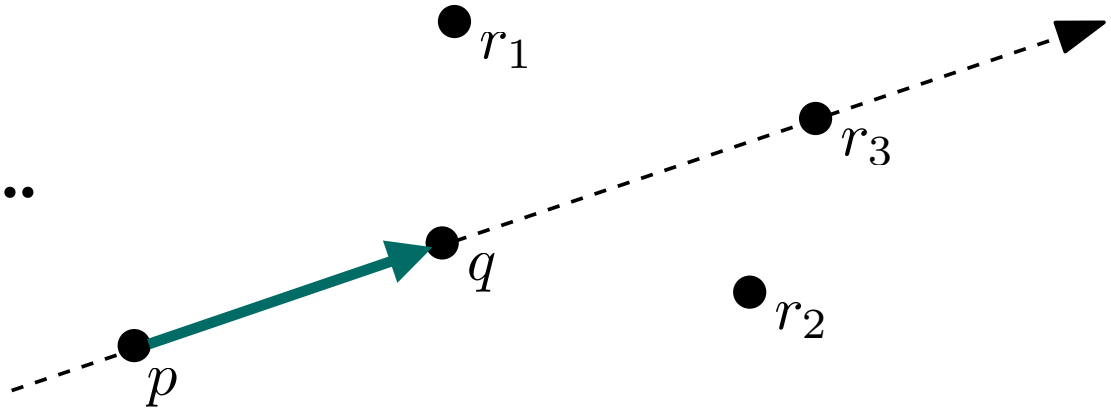
Given 3 points p, q, r in the plane, efficiently and robustly decide whether r lies to the left, to the right or on the oriented line pq.



Orientation Test

Given 3 points p, q, r in the plane, efficiently and robustly decide whether r lies to the left, to the right or on the oriented line pq.

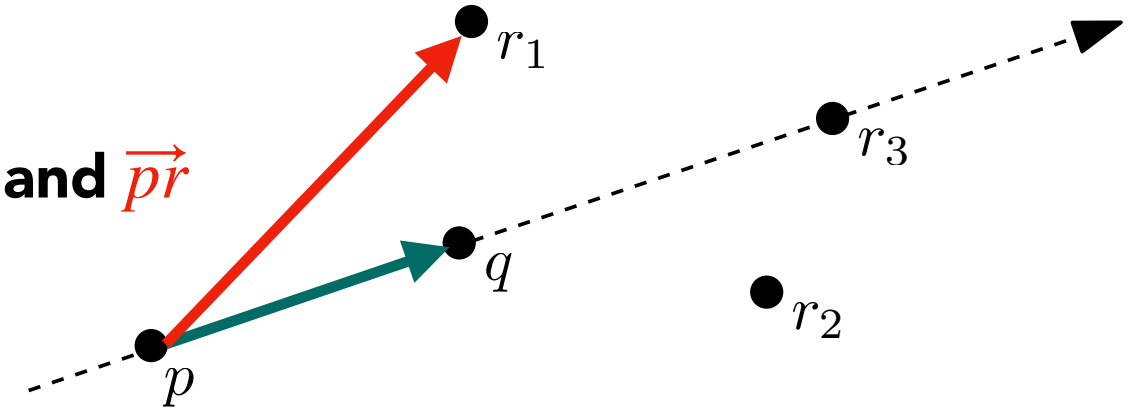
Consider the vectors $\overrightarrow{pq}$ ...



Orientation Test

Given 3 points p, q, r in the plane, efficiently and robustly decide whether r lies to the left, to the right or on the oriented line pq.

Consider the vectors $\overrightarrow{pq}$ and $\overrightarrow{pr}$

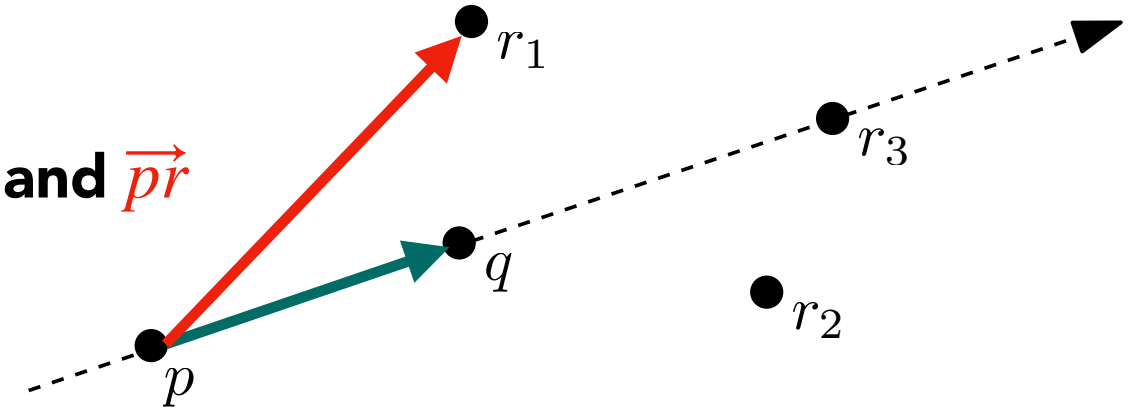


Orientation Test

Given 3 points p, q, r in the plane, efficiently and robustly decide whether r lies to the left, to the right or on the oriented line pq.

Consider the vectors $\overrightarrow{pq}$ and $\overrightarrow{pr}$

Compute $\overrightarrow{pq}$ x $\overrightarrow{pr}$



Point r lies on the line $pq \iff \begin{vmatrix} q_x - p_x & r_x - p_x \\ q_y - p_y & r_y - p_y \end{vmatrix} = 0$

Point r lies to the left of the oriented line $pq \iff \begin{vmatrix} q_x - p_x & r_x - p_x \\ q_y - p_y & r_y - p_y \end{vmatrix} > 0$

Point r lies to the right of the oriented line $pq \iff \begin{vmatrix} q_x - p_x & r_x - p_x \\ q_y - p_y & r_y - p_y \end{vmatrix} < 0$