NYU

Virtual and Augmented Reality

CS-GY 9223/CUSP-GX 6004

https://nyu-icl.github.io/courses/2022fall-vr-ar

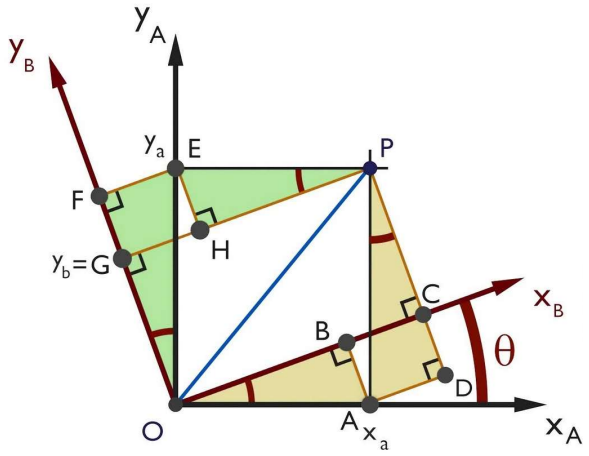
Prof. Qi Sun
qisun@nyu.edu | www.qisun.me

Logistics

- Assignment 1 Due Oct 12 Mid-night
  - Code + output + plots
  - Feel free to use any other language

- Today: last lecture today on background knowledge on viewing

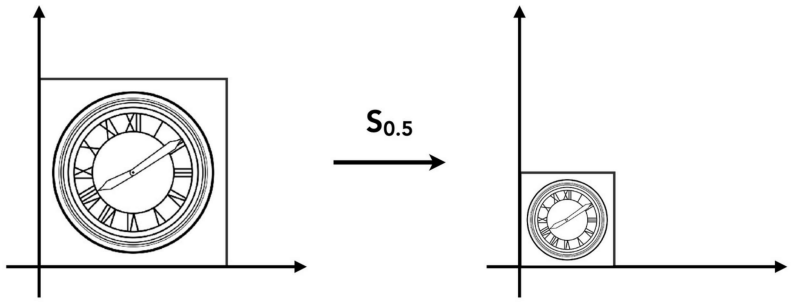
Review



- Vector/matrix/complex numbers

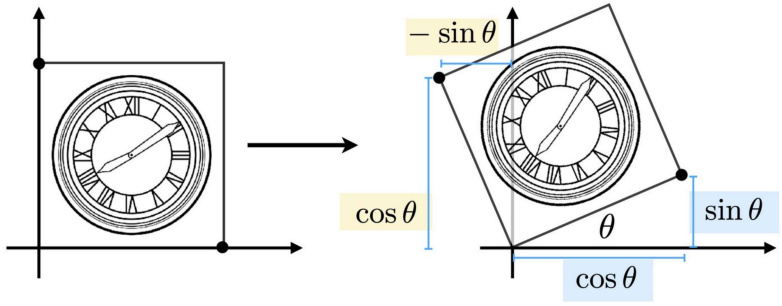
- 2D image transformation (represented by matrix)

Scale Matrix



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Rotation Matrix



$$\mathbf{R}_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

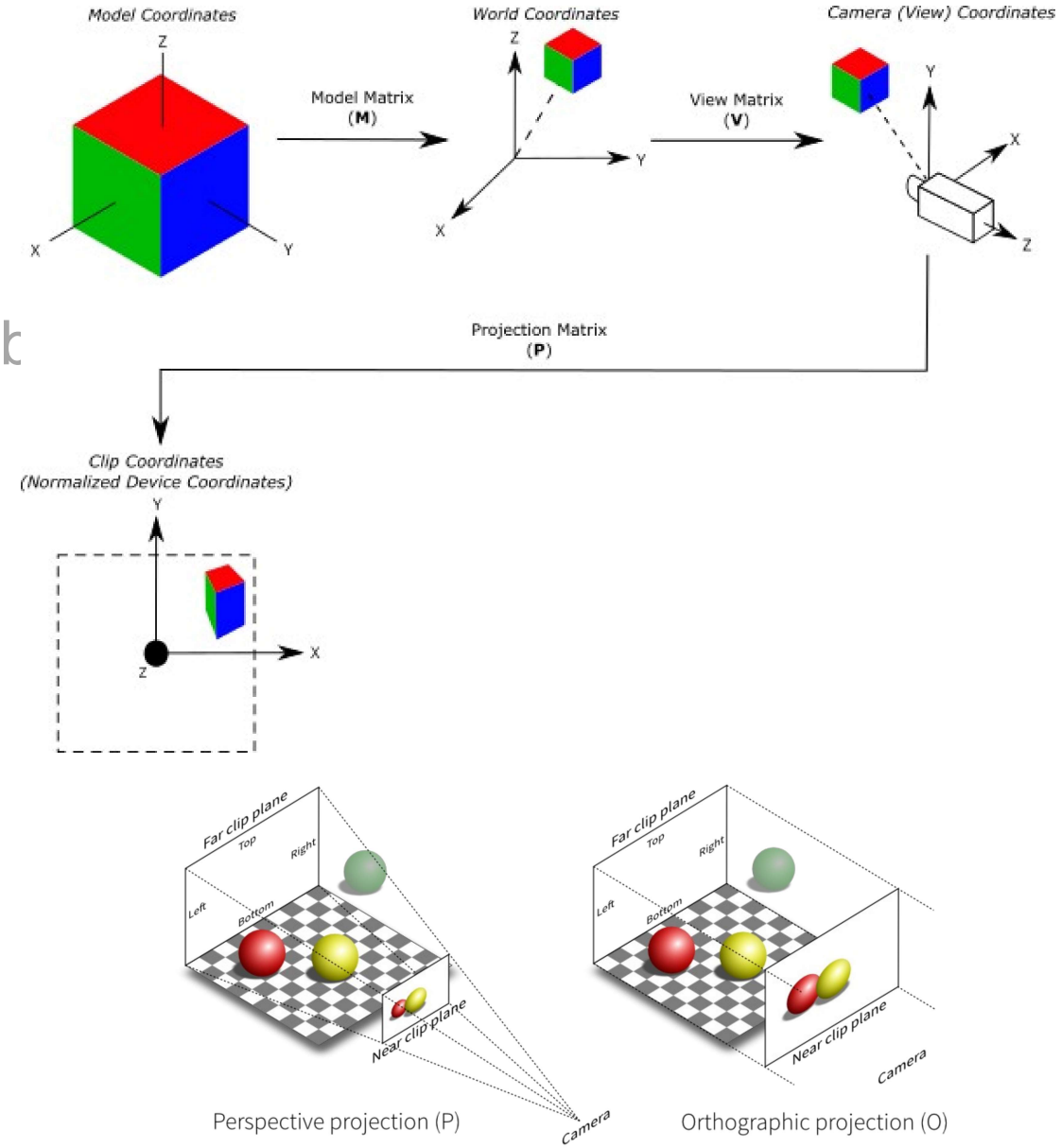
Review

- Vector/matrix/complex numbers
- 2D image transformation (represented by matrix)
- Homogeneous coordinate

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \qquad \begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \\ 1 \end{pmatrix}$$

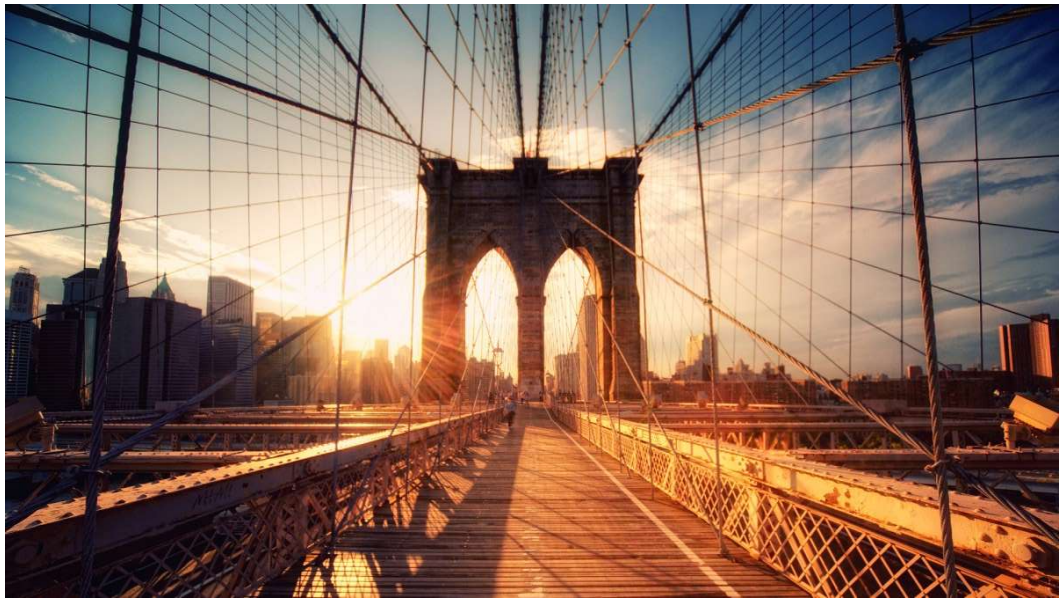
Review

- Vector/matrix/complex numb
- 2D image transformation
- Homogeneous coordinate
- 3D **M**odel-**V**iew-**P**rojection



Perspective Projection

- A property of homogeneous coordinates
  - (x, y, z, 1) and (kx, ky, kz, k) represent the same point (x, y, z) in 3D
  - e.g. (1, 0, 0, 1) and (2, 0, 0, 2) both represent (1, 0, 0)



Perspective Projection

- How?
  - First “squish” the frustum into a cuboid (n -> n, f -> f) ($M_{persp->ortho}$)
  - Do orthographic projection (M_{ortho})

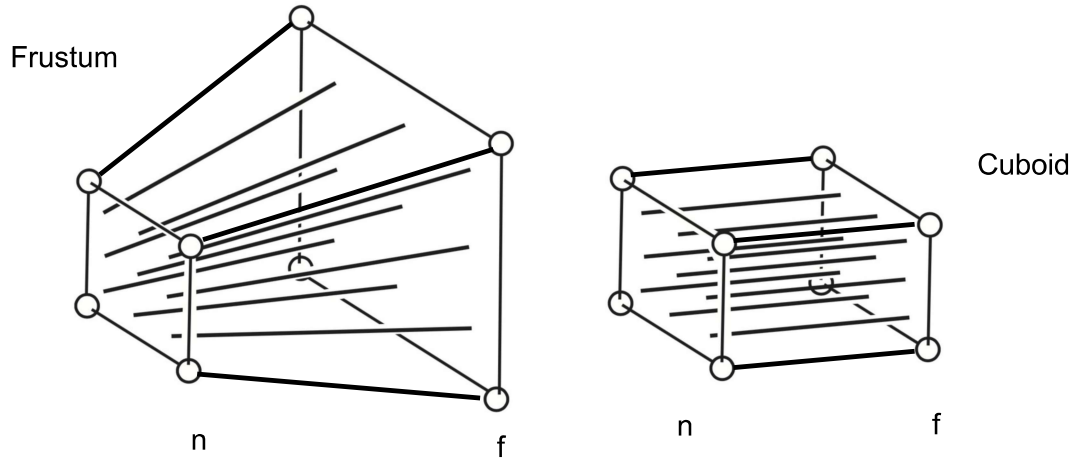
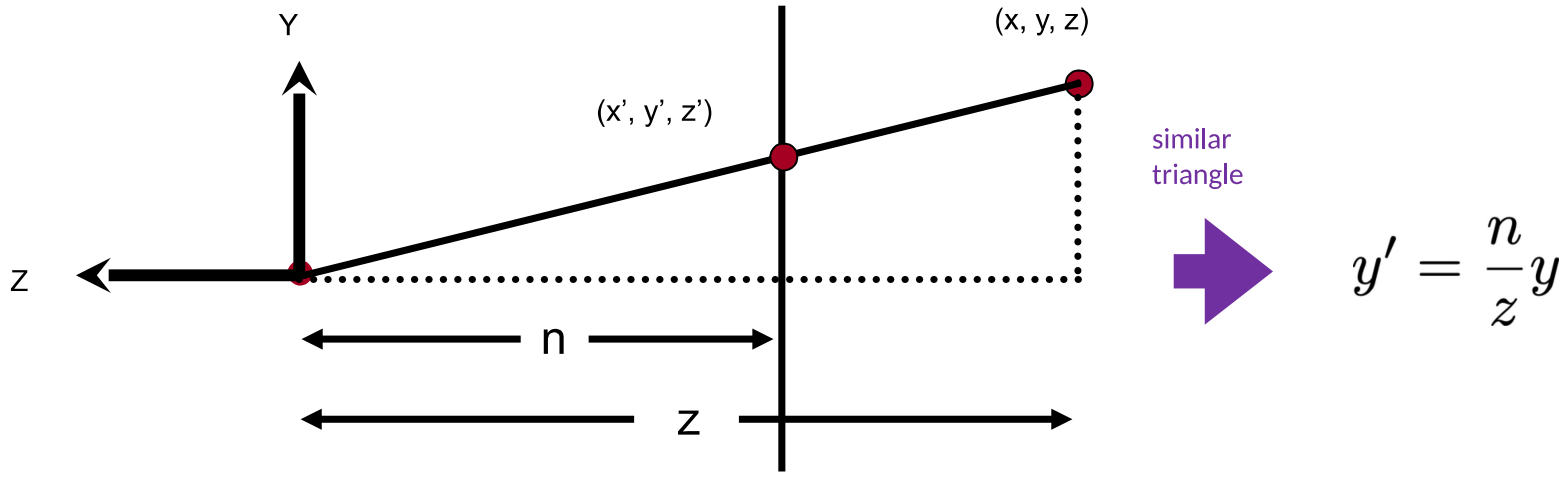


Fig. 7.13 from *Fundamentals of Computer Graphics, 4th Edition*

Perspective Projection

- In order to find a transformation
  - Find the relationship between transformed points (x', y', z') and the original points (x, y, z)



Perspective Projection

- In order to find a transformation
  - Find the relationship between transformed points (x', y', z') and the original points (x, y, z)

$$y' = \frac{n}{z}y \qquad x' = \frac{n}{z}x$$ (similar to y')

- In homogeneous coordinates,

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} nx/z \\ ny/z \\ \text{unknown} \\ 1 \end{pmatrix} \overset{\text{mult. by z}}{==} \begin{pmatrix} nx \\ ny \\ \text{still unknown} \\ z \end{pmatrix}$$

Perspective Projection

- So the “squish” (persp to ortho) projection does this

$$M^{(4\times4)}_{persp\to ortho}\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} nx \\ ny \\ \text{unknown} \\ z \end{pmatrix}$$

- Already good enough to figure out part of $M_{\text{persp->ortho}}$

$$M_{persp\to ortho} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ ? & ? & ? & ? \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Perspective Projection

- How to figure out the third row of $M_{persp->ortho}$
  - Any information that we can use?

$$M_{persp \to ortho} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ ? & ? & ? & ? \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

- Observation: the third row is responsible for z'
  - Any point on the near plane will not change
  - Any point's z on the far plane will not change

Perspective Projection

- Any point on the near plane will not change

$$M^{(4\times4)}_{persp\to ortho}\begin{pmatrix}x\\y\\z\\1\end{pmatrix}=\begin{pmatrix}nx\\ny\\\text{unknown}\\z\end{pmatrix}$$

replace z with n

$$\begin{pmatrix}x\\y\\n\\1\end{pmatrix}\Rightarrow\begin{pmatrix}x\\y\\n\\1\end{pmatrix}==\begin{pmatrix}nx\\ny\\n^2\\n\end{pmatrix}$$

- So the third row must be of the form (0 0 A B)

$$\begin{pmatrix}0 & 0 & A & B\end{pmatrix}\begin{pmatrix}x\\y\\n\\1\end{pmatrix}=n^2$$

Perspective Projection

- Besides, we have

$$\begin{pmatrix} 0 & 0 & A & B \end{pmatrix} \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} = n^2 \quad \Rightarrow \quad An + B = n^2$$

- Any point's z on the far plane will not change

$$\begin{pmatrix} 0 \\ 0 \\ f \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 \\ 0 \\ f \\ 1 \end{pmatrix} == \begin{pmatrix} 0 \\ 0 \\ f^2 \\ f \end{pmatrix} \quad \Rightarrow \quad Af + B = f^2$$

Perspective Projection

- Solve for A and B

$$An + B = n^2 \qquad A = n + f$$
$$Af + B = f^2 \qquad B = -nf$$

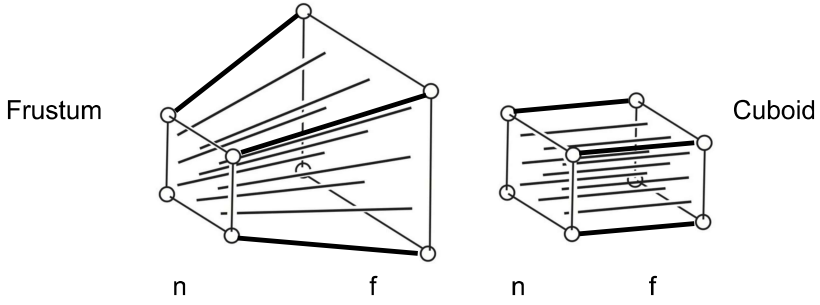
- Finally, every entry in $M_{persp->ortho}$ is known!

- What's next?
  - Do orthographic projection (M_{ortho}) to finish

$$M_{persp} = M_{ortho} M_{persp \rightarrow ortho}$$

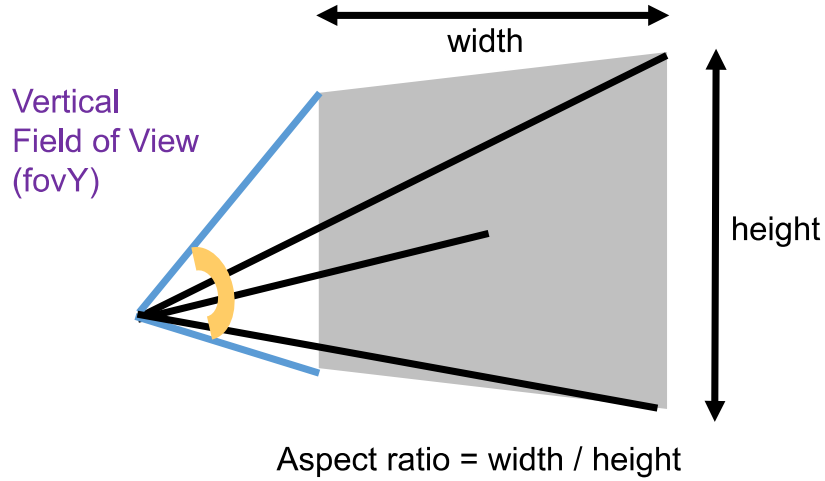
Perspective Projection

- What orthographic projection to perform?
  - Recall: just need the cuboid
  - [l, r] x [b, t] x [n, f]
  - n and f did not change
  - l, r, b, t are near plane's l, r, b, t



Perspective Projection

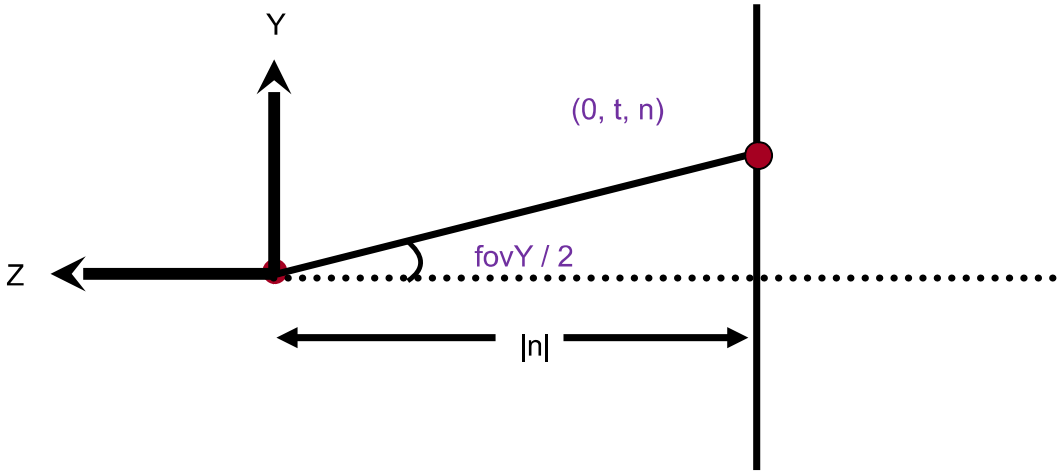
- What's near plane's l, r, b, t then?
  - If explicitly specified, good
  - Sometimes people prefer defining them with vertical **field-of-view** (fovY) and **aspect ratio**, and assume symmetry i.e. l = -r, b = -t



Slides courtesy: Prof. Ling-Qi Yar

Perspective Projection

- How to convert from fovY and aspect to l, r, b, t?
  - Trivial



$$\tan \frac{fovY}{2} = \frac{t}{|n|}$$

$$aspect = \frac{r}{t}$$

Finally : On-Scree Drawing

Drawing Machines and Laser Cutters

Consumer displays, however

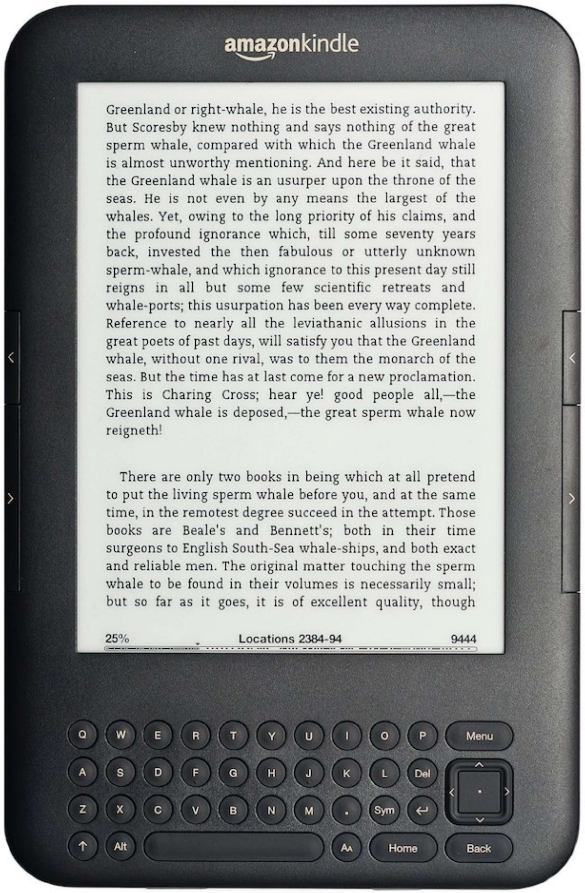
Low-Res LCD Display

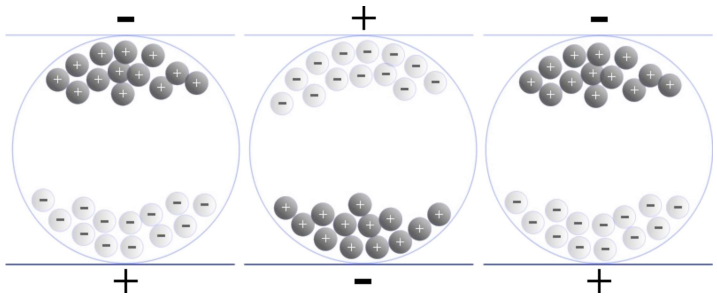
Color LCD, OLED, ...

LED Array Display

Light emitting diode array

Electrophoretic (Electronic Ink) Display

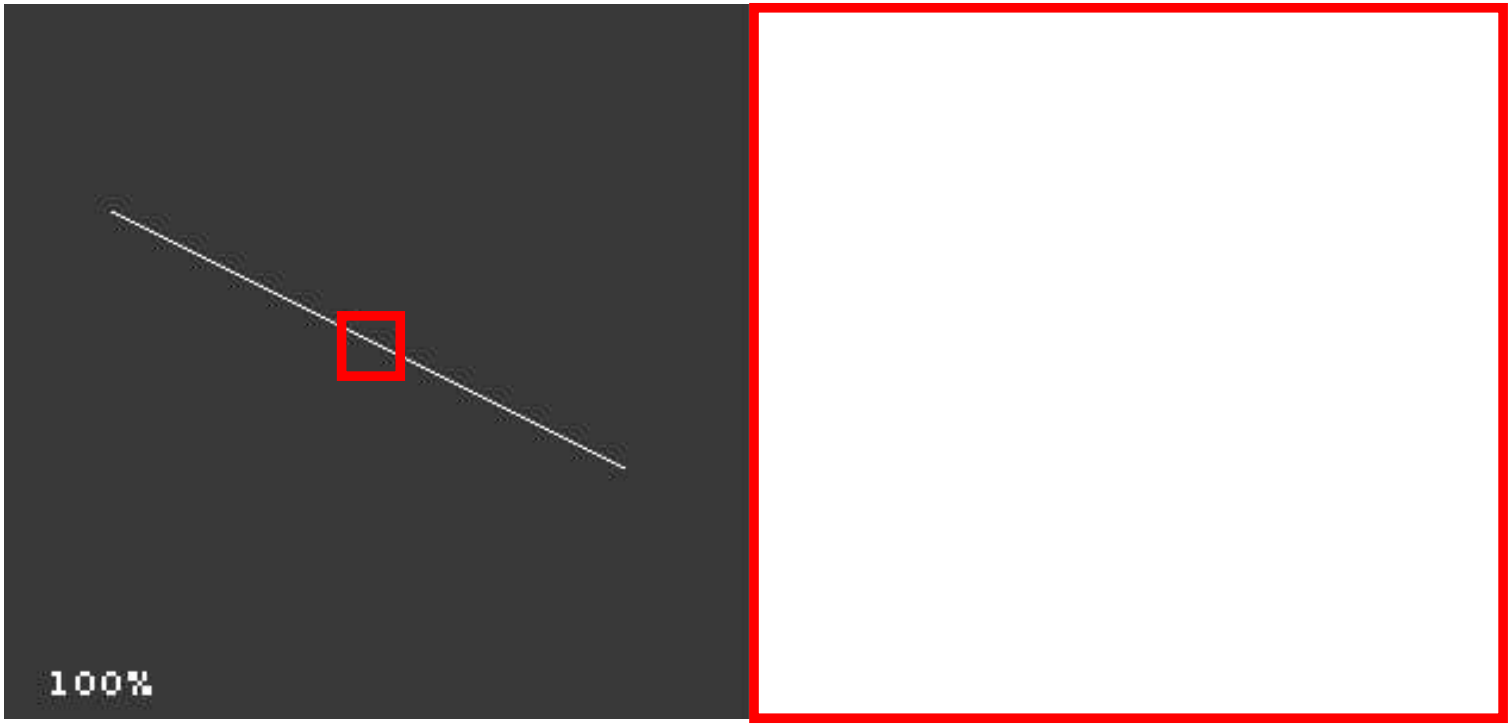




[Wikimedia Commons—Senarclens]

2
2

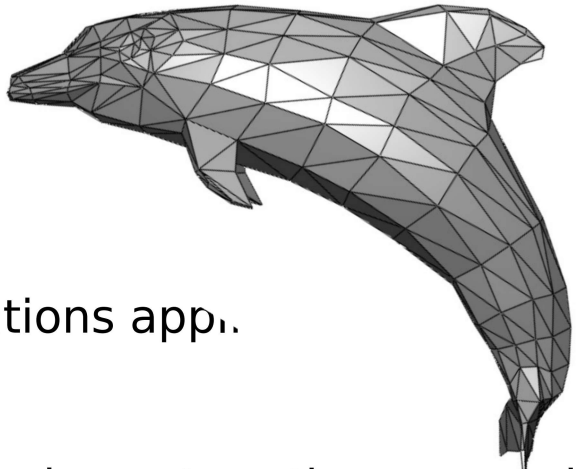
There is **NO** “straight line” on your screen



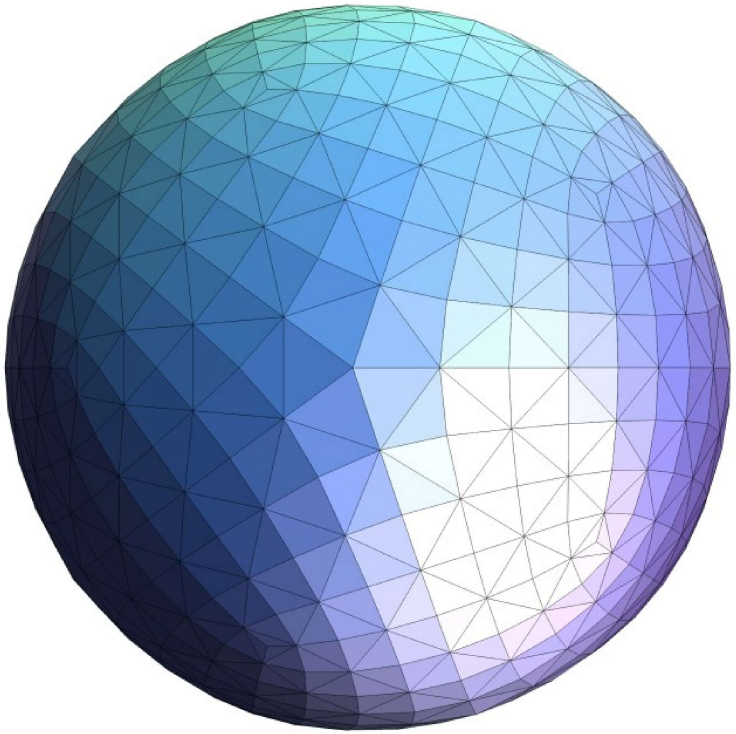
https://www.pixelbath.com/isometric-pixel-art-guide/chapter-2-basic-pixel

Triangles – The Simplest Primitives

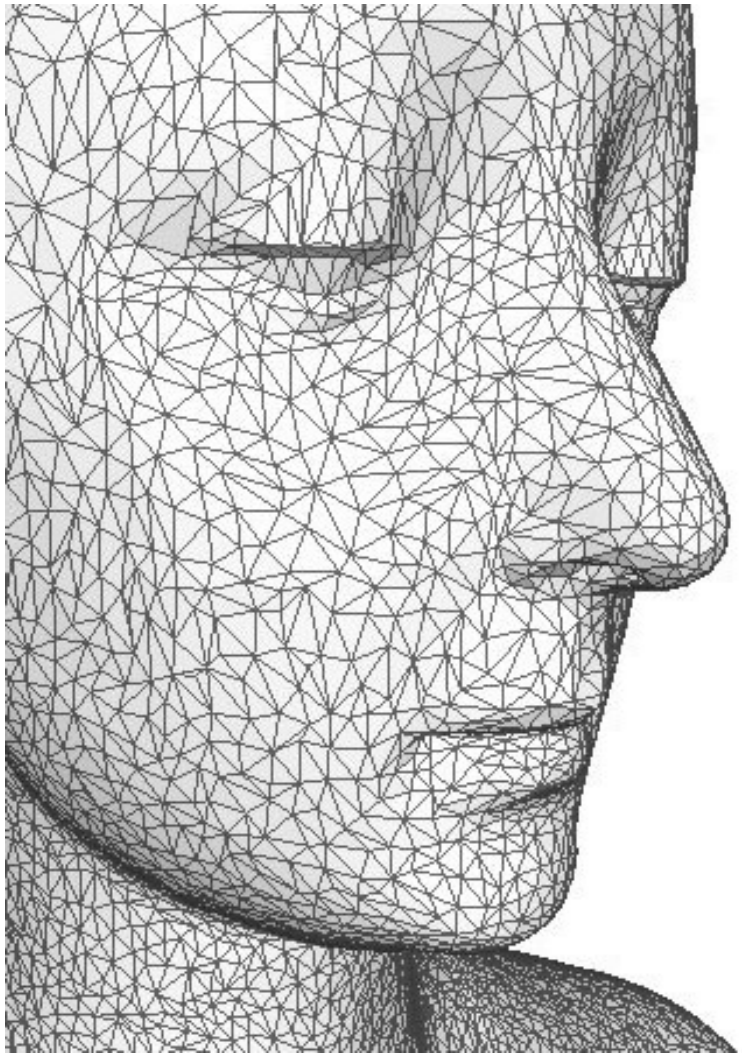
- Why?
  - Basic and LINEAR!
    - Can compose other shapes
    - Guaranteed to be planar – linear operations appl.
    - Well-defined interior
    - Well-defined method for interpolating values at vertices over triangle



Triangle Meshes

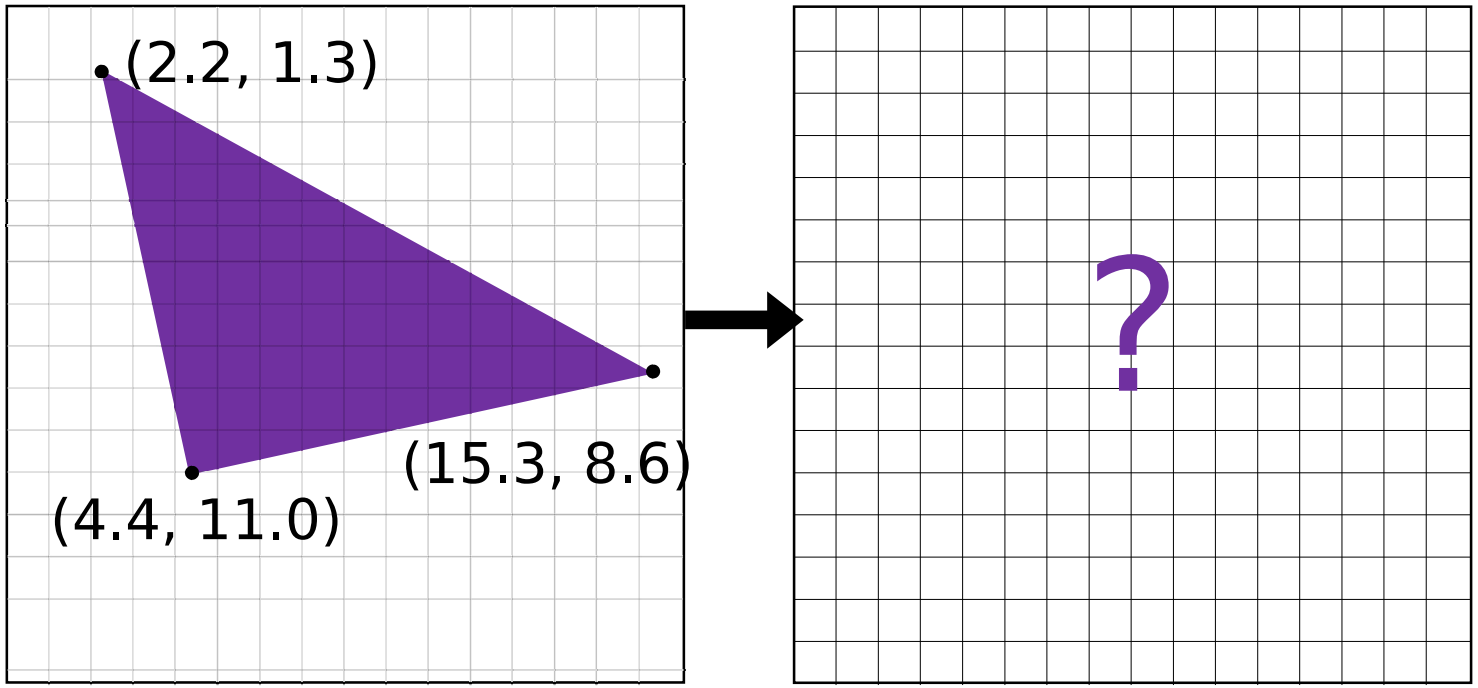


Triangle Meshes



Drawing a Triangle To The Screen (“Rasterization”)

Problem: how to draw this triangle on screen?



Input: position of triangle vertices projected (after MVP) on screen

Output: set of pixel values approximating triangle

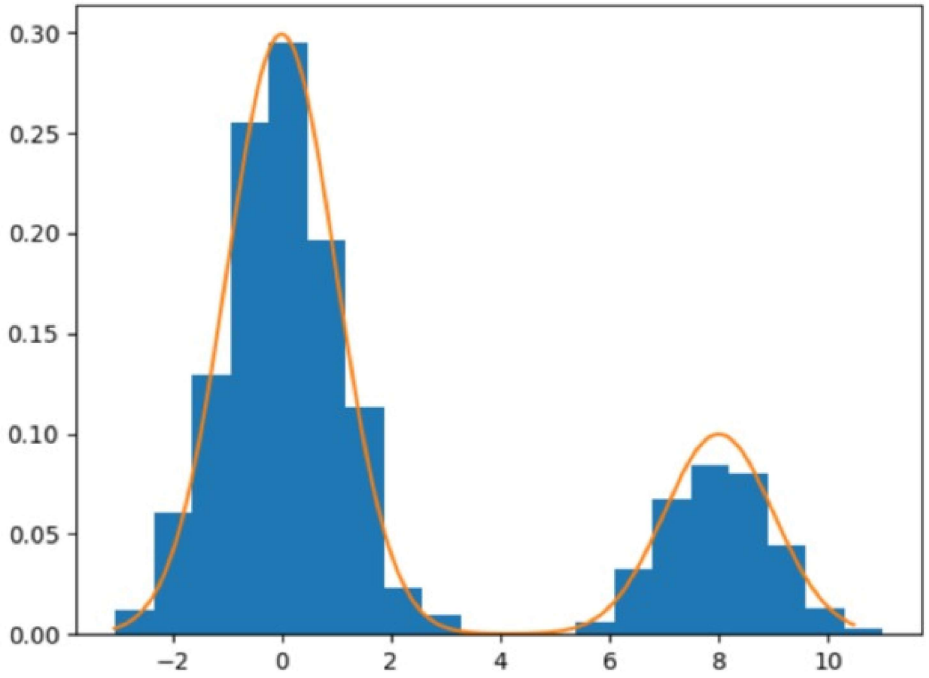
Slides courtesy: Prof. Ling-Qi Yar

Solution: Sampling

What is/Why Sampling

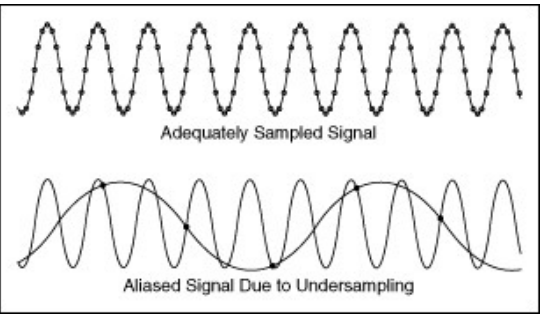
- Definition: evaluating a function at a point
- We can discretize a function by samp
- As simple as:

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

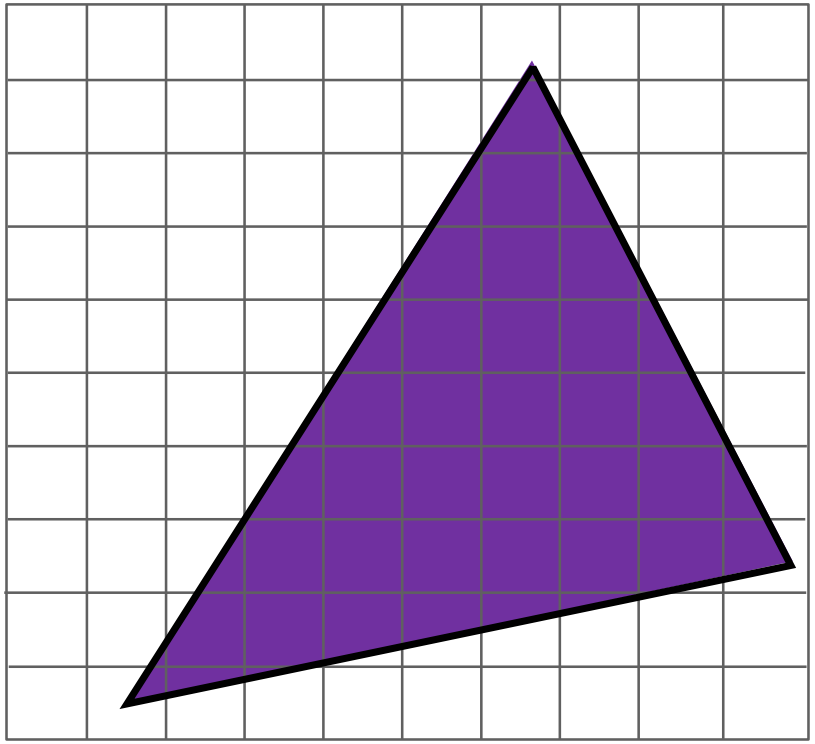


What is/Why Sampling

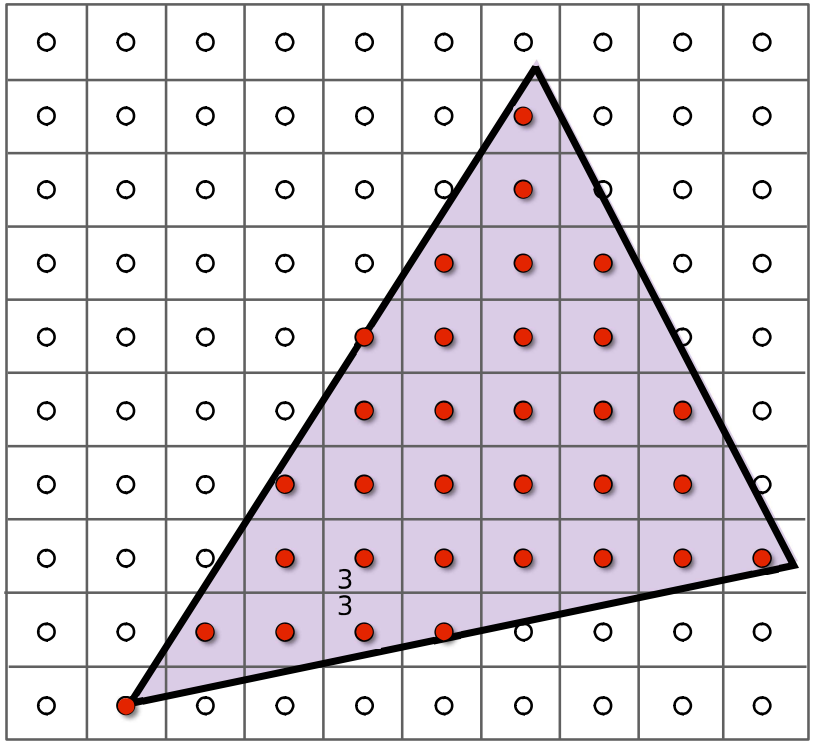
“If a function x(t) contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced 1/(2B) seconds apart.” -- Shannon Sampling Theorem



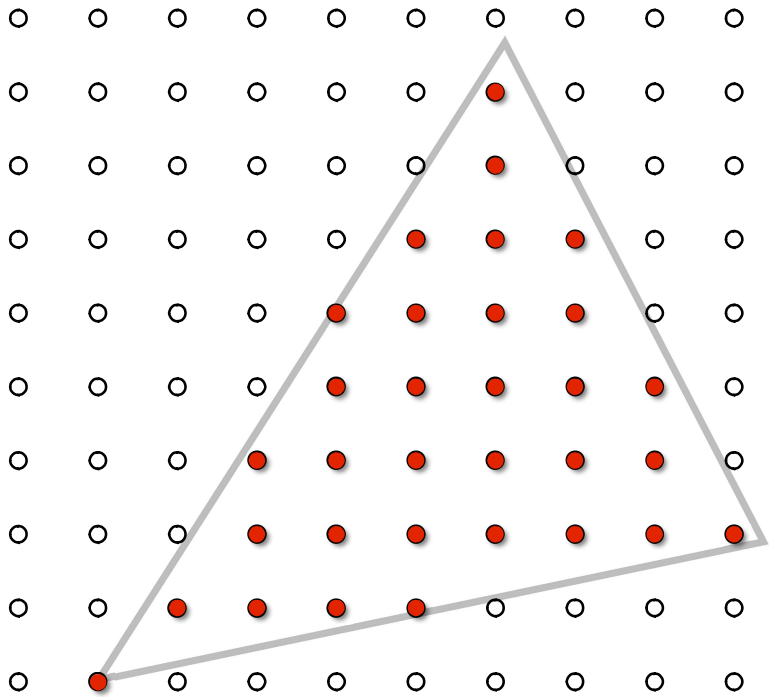
Let’s Try Rasterization As 2D Sampling



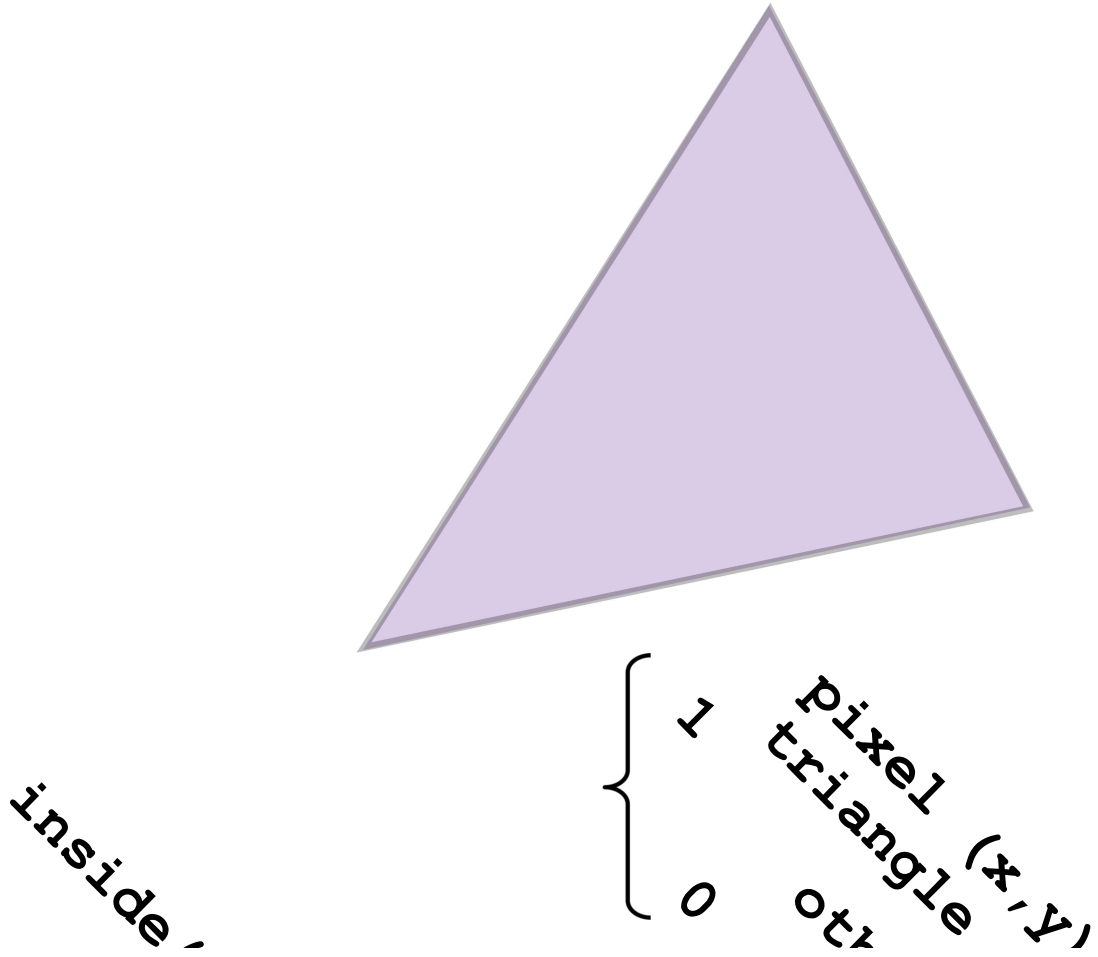
Is a Pixel Center Inside the Triangle?



Is a Pixel Center Inside the Triangle?



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

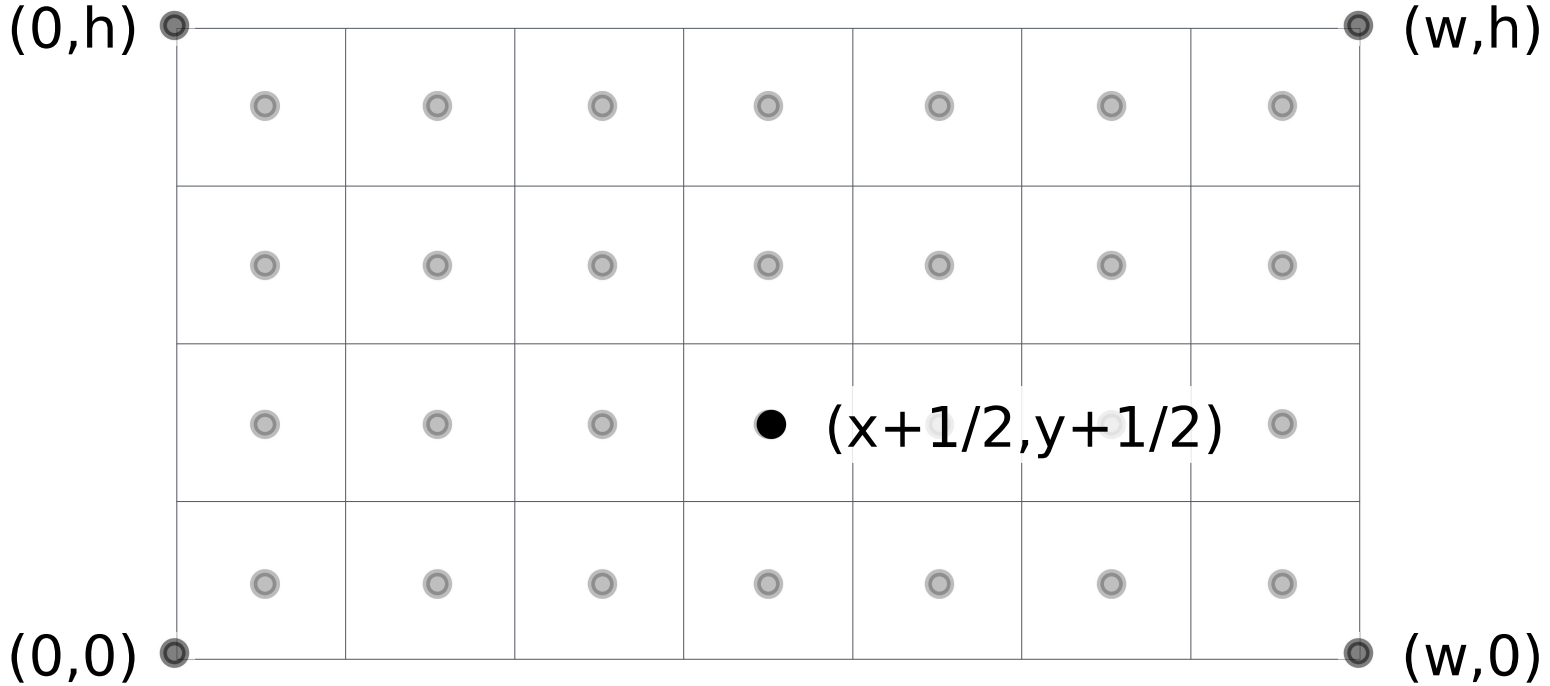


A Naïve Sampling/Rasterization

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

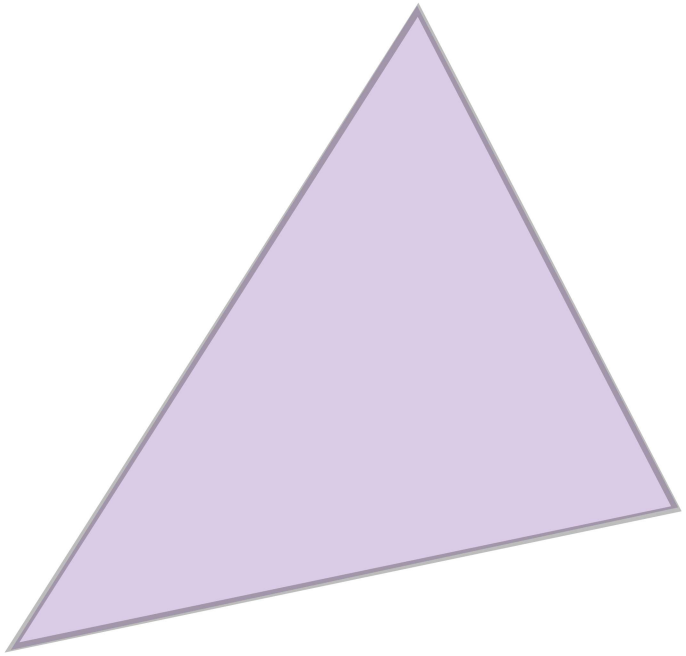
Rasterize triangle **tri** by sampling the function
f(x,y) = inside(tri,x,y)

Recall: Sample Locations

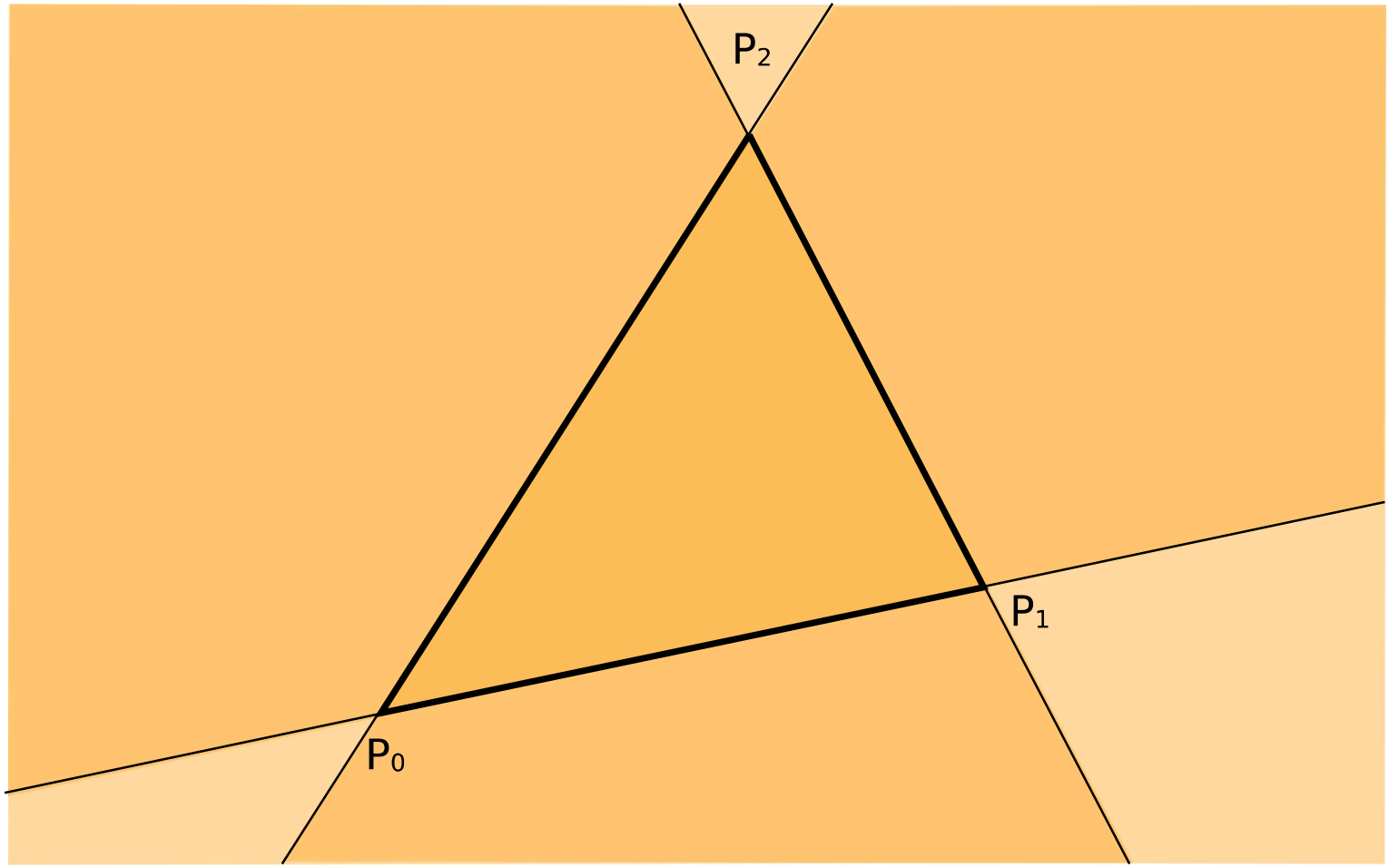


Sample location for pixel (x,y)

Evaluating `inside(tri,x,y)`



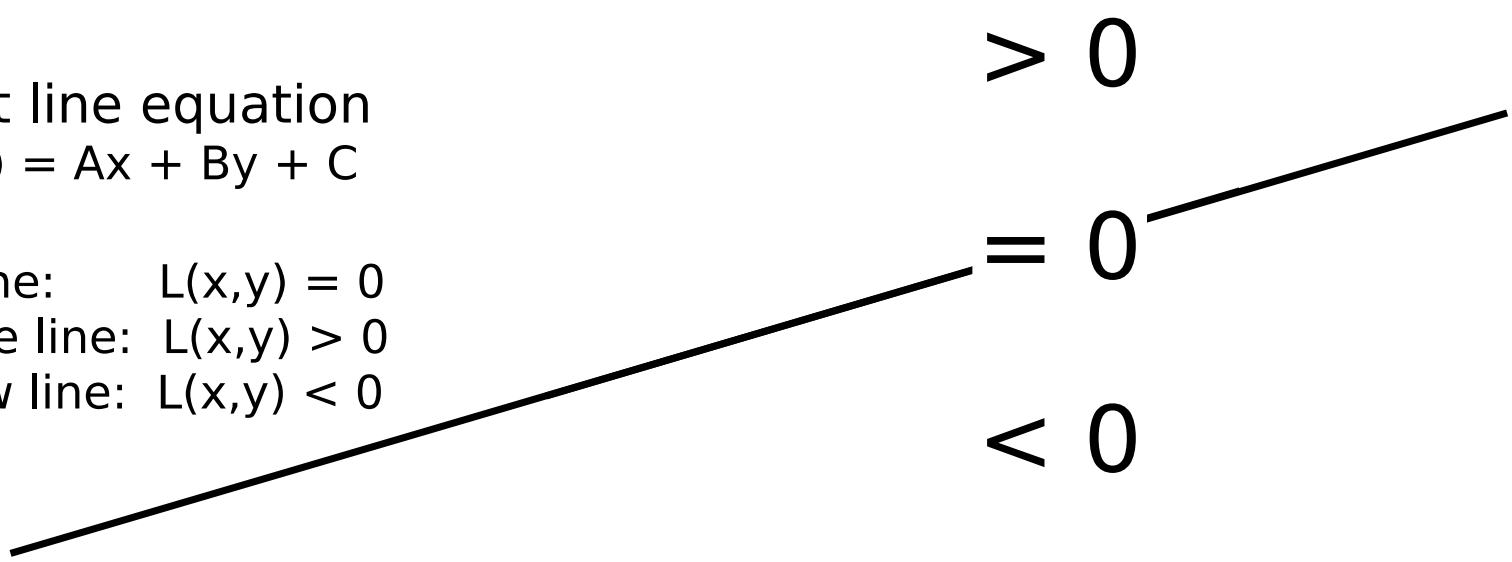
Triangle = Intersection of Three Half Planes



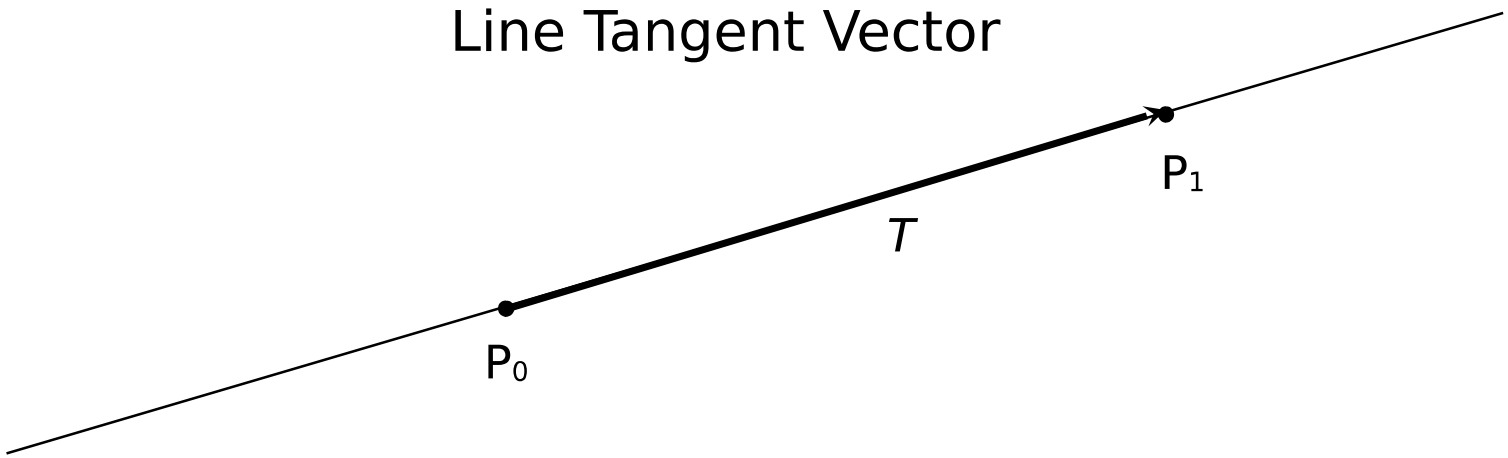
Each Line Defines Two Half-Planes

- 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

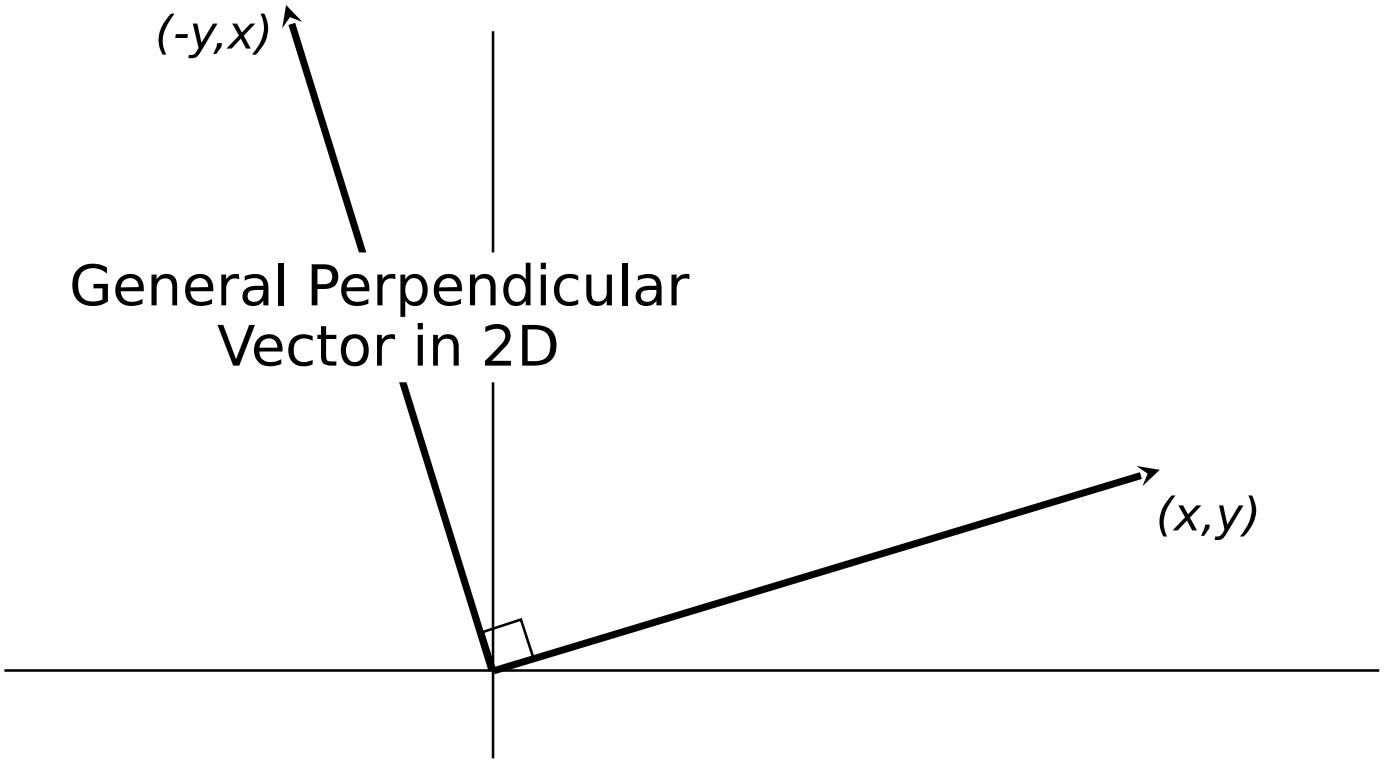


Line Equation Derivation



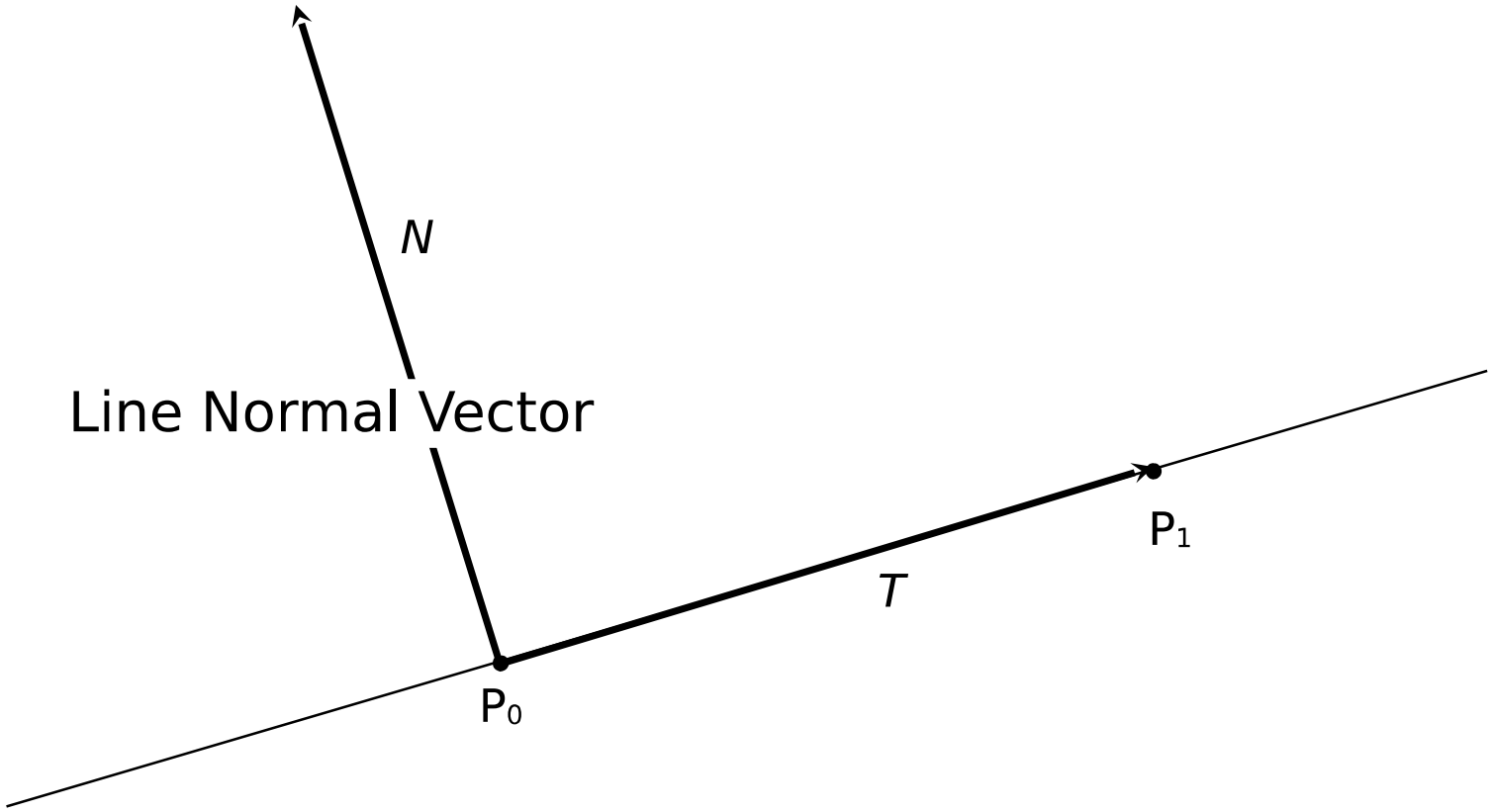
$$T = P_1 - P_0 = (x_1 - x_0, y_1 - y_0)$$

Line Equation Derivation



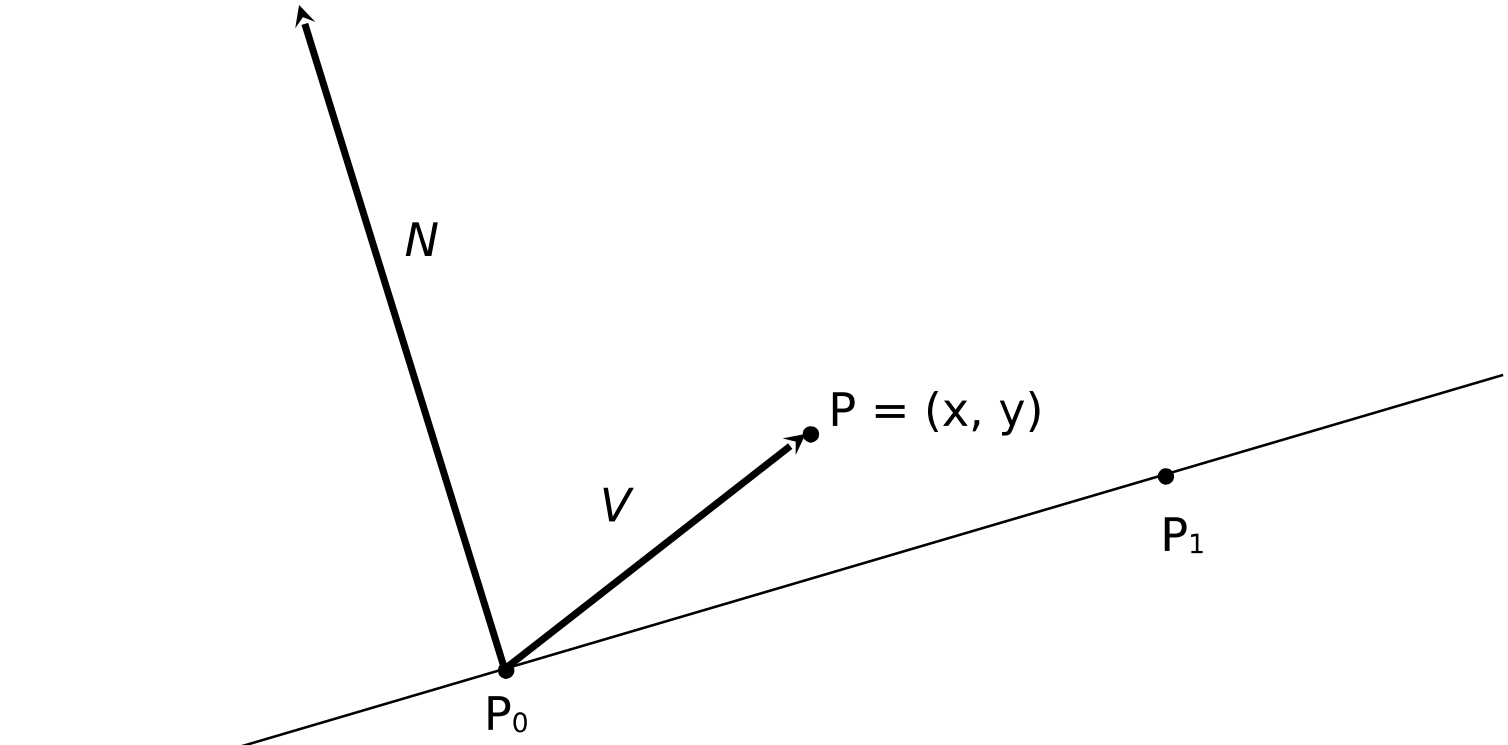
$$\text{Perp}(x, y) = (-y, x)$$

Line Equation Derivation



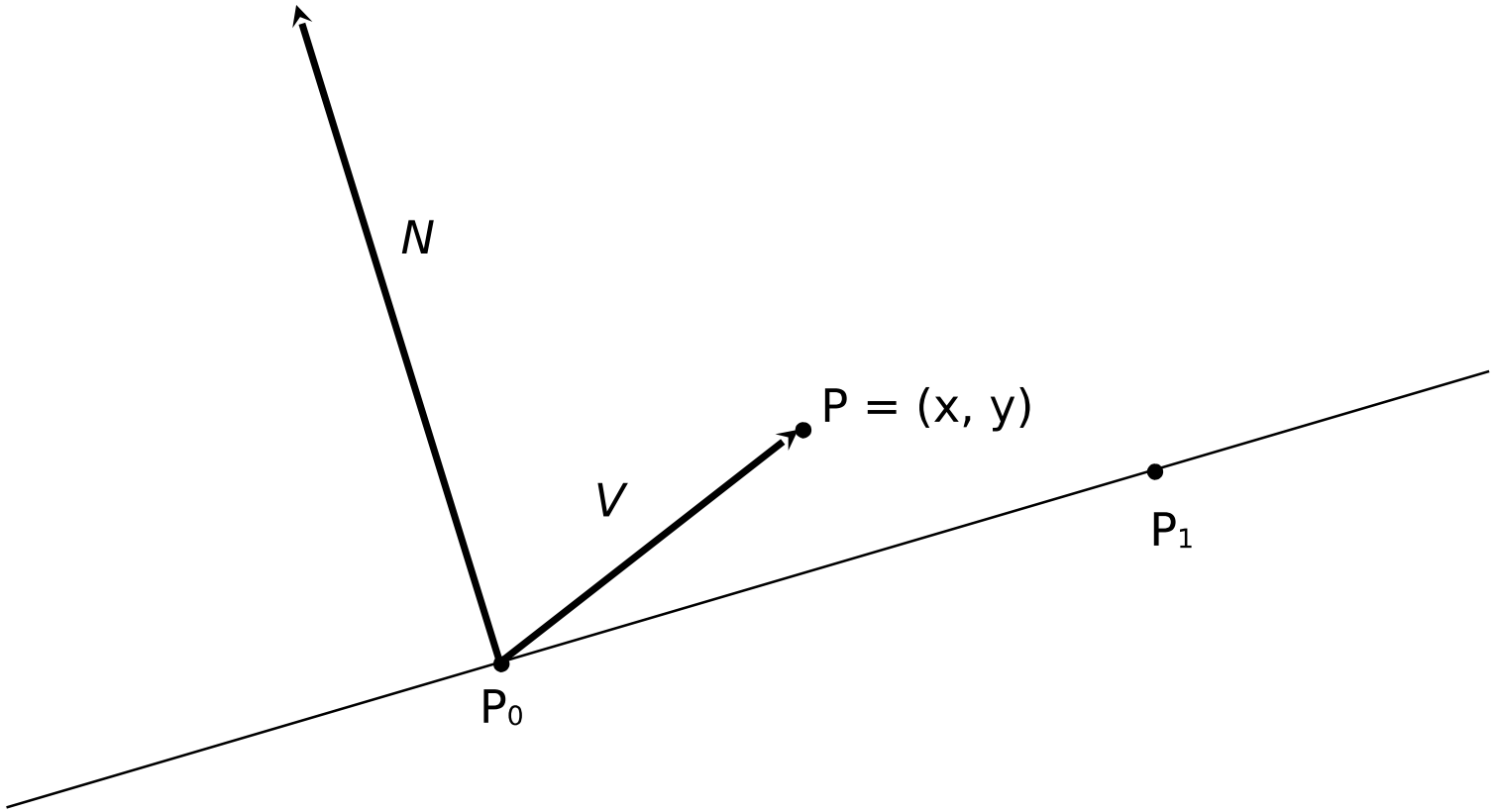
$$N = \mathrm{Perp}(T) = (-(y_1 - y_0), x_1 - x_0)$$

Line Equation Derivation



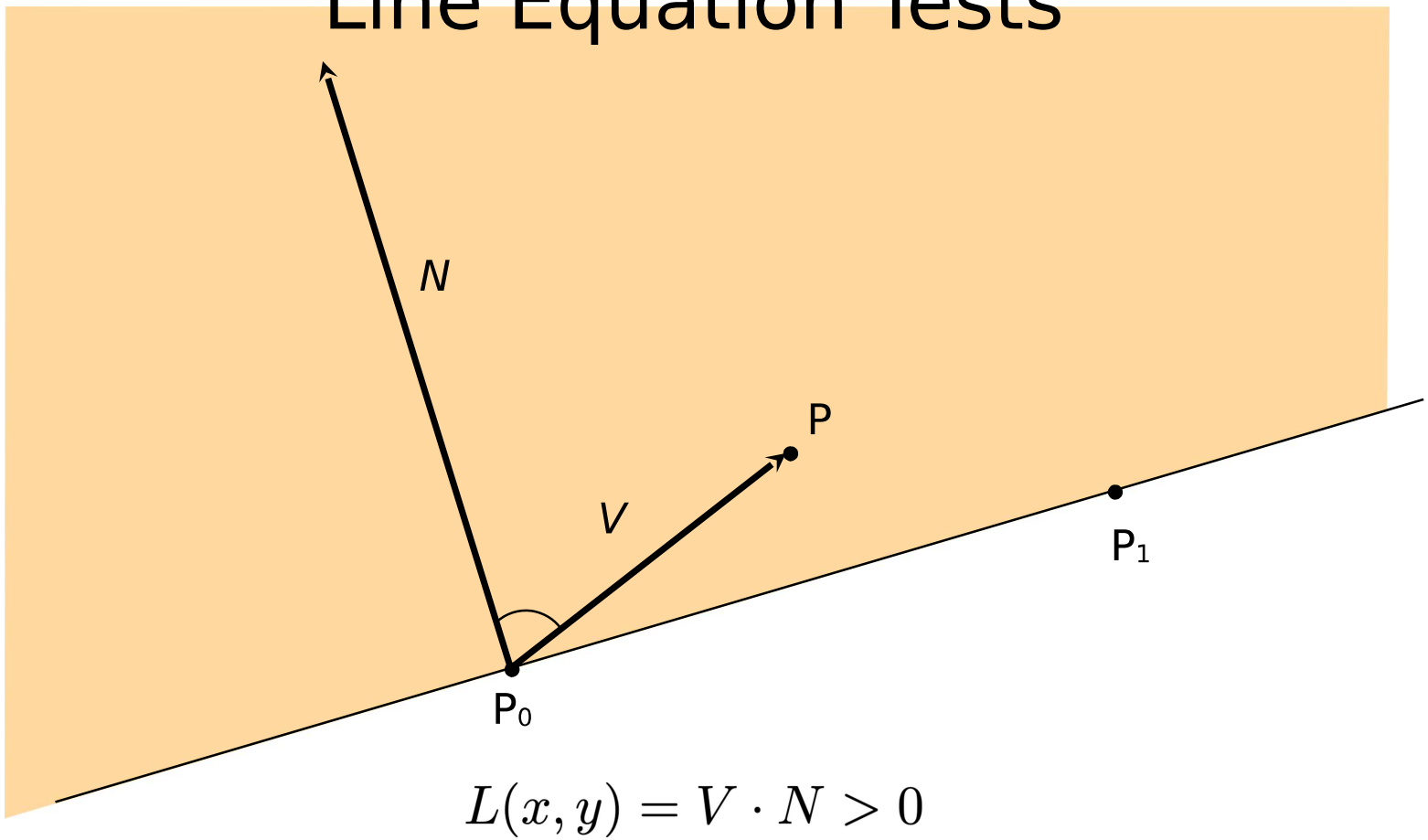
$$V = P - P_0 = (x - x_0, y - y_0)$$

Line Equation



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

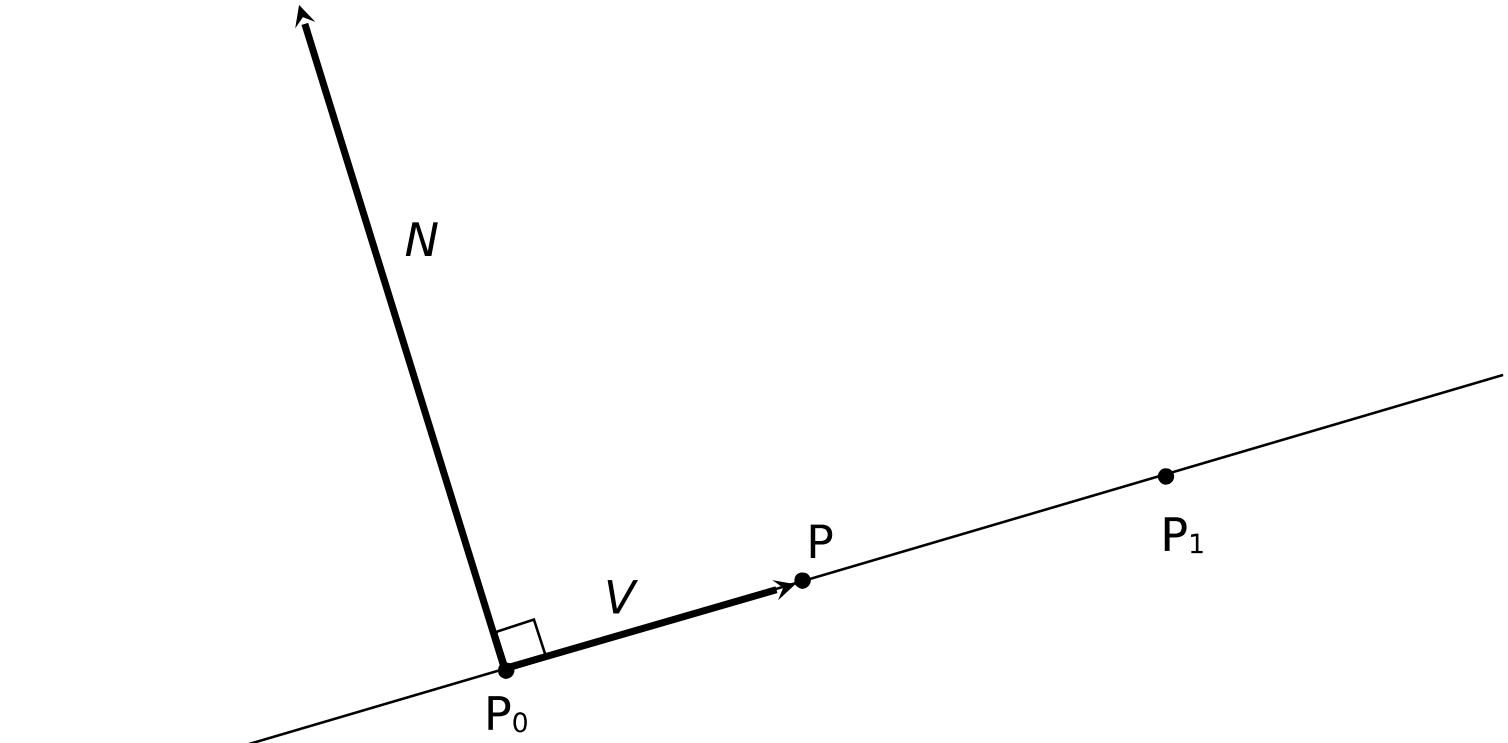
Line Equation Tests



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

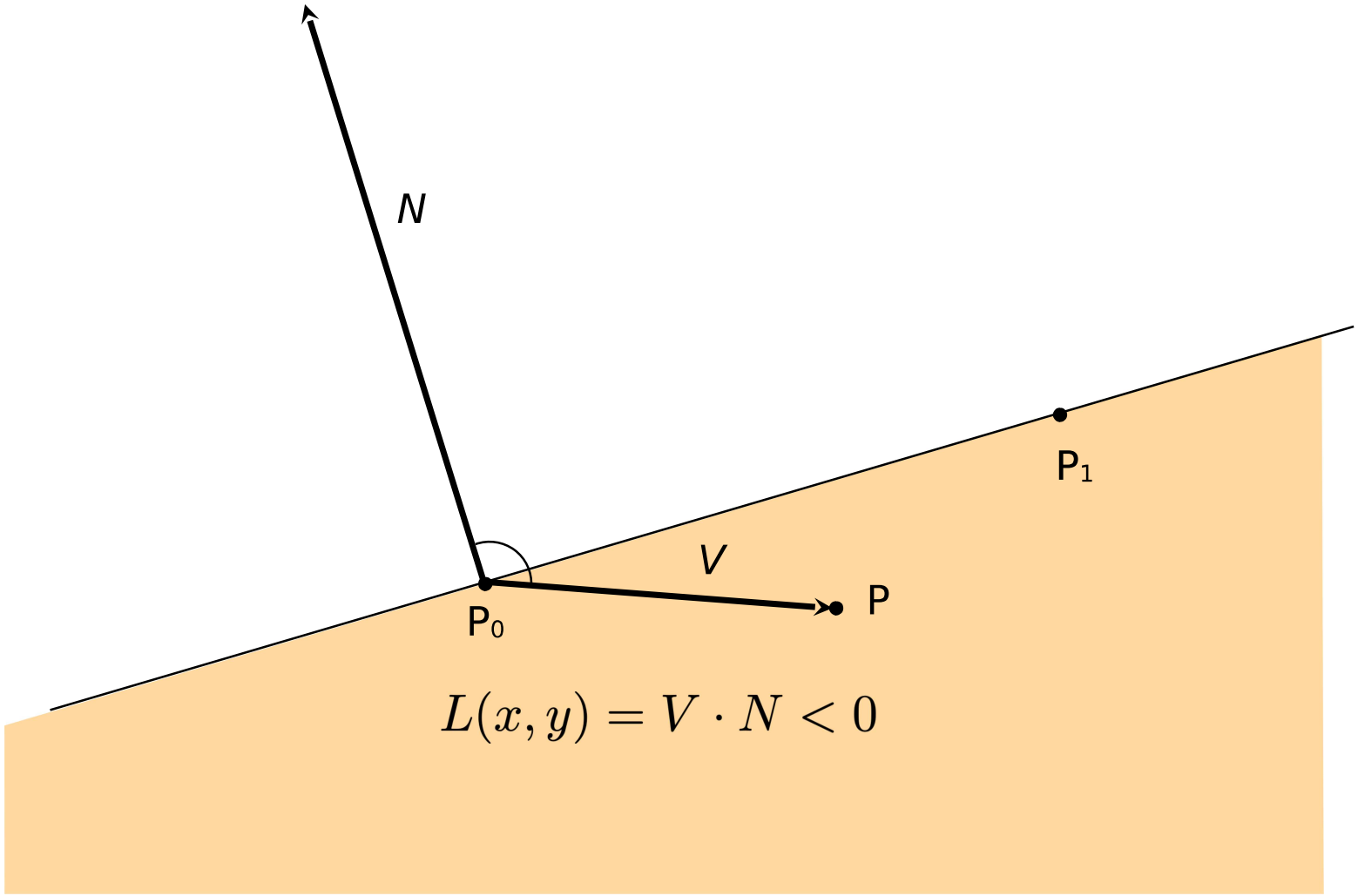
Careful: Orders matter!

Line Equation Tests



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

Line Equation Tests



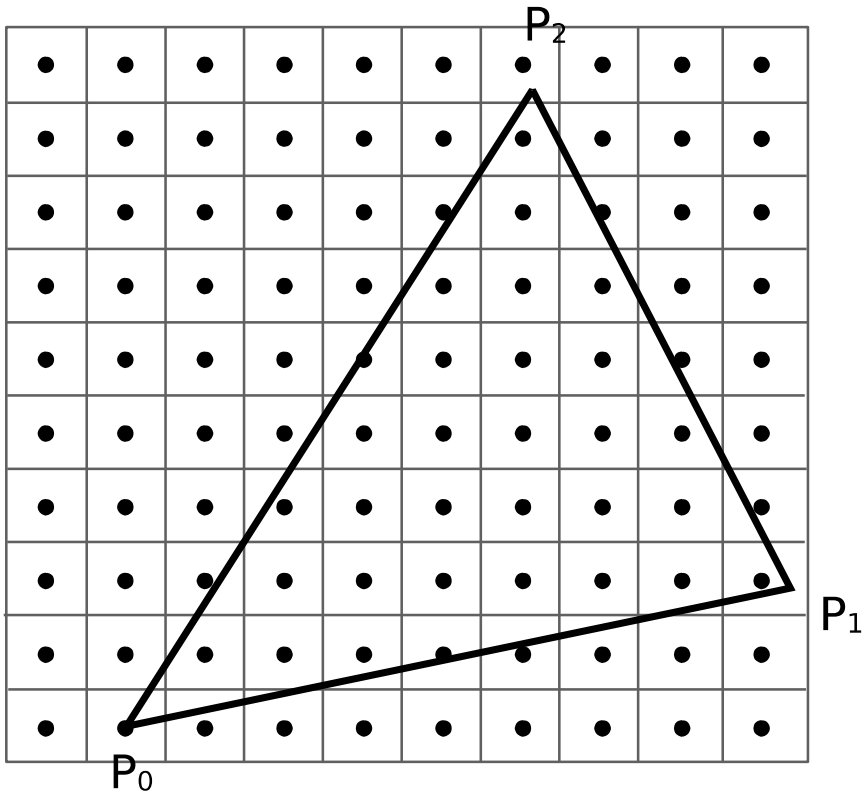
Point-in-Triangle Test: Three Line Tests

$P_i = (X_i, Y_i)$

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

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

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



Compute line equations from pairs of vertices

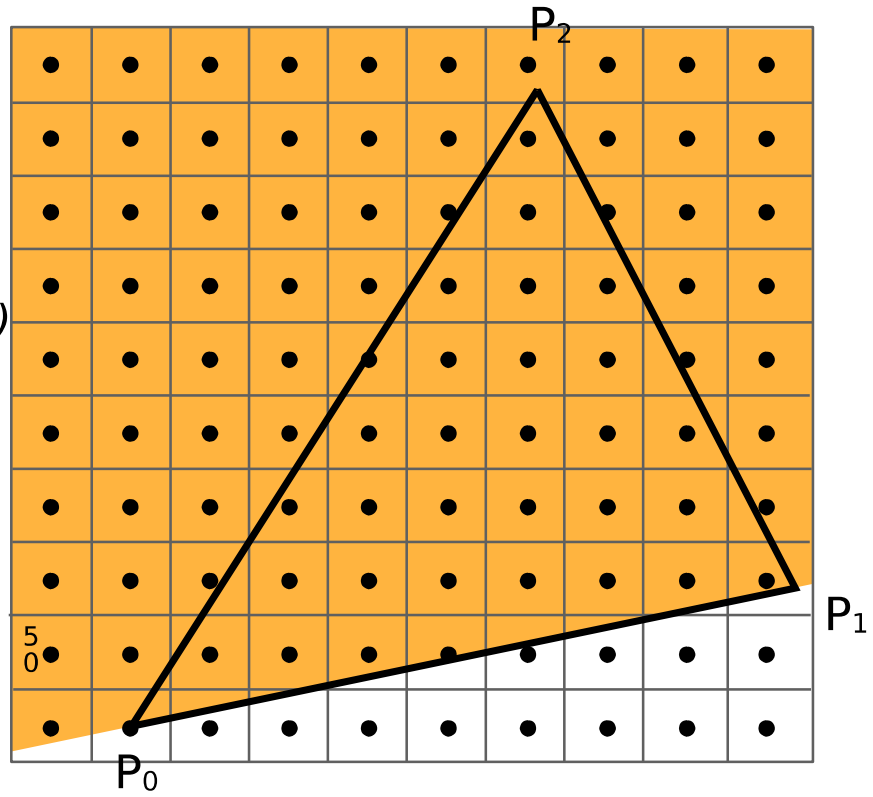
Point-in-Triangle Test: Three Line Tests

$P_i = (X_i, Y_i)$

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

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

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



$L_0(x, y) > 0$

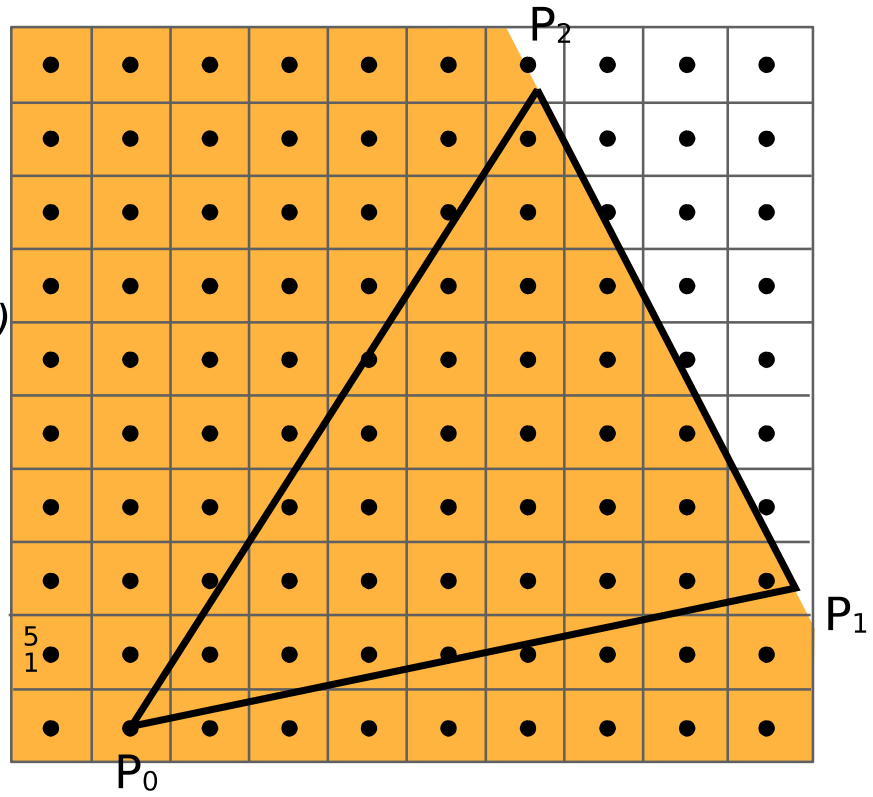
Point-in-Triangle Test: Three Line Tests

$P_i = (X_i, Y_i)$

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

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

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



$L_1(x, y) > 0$

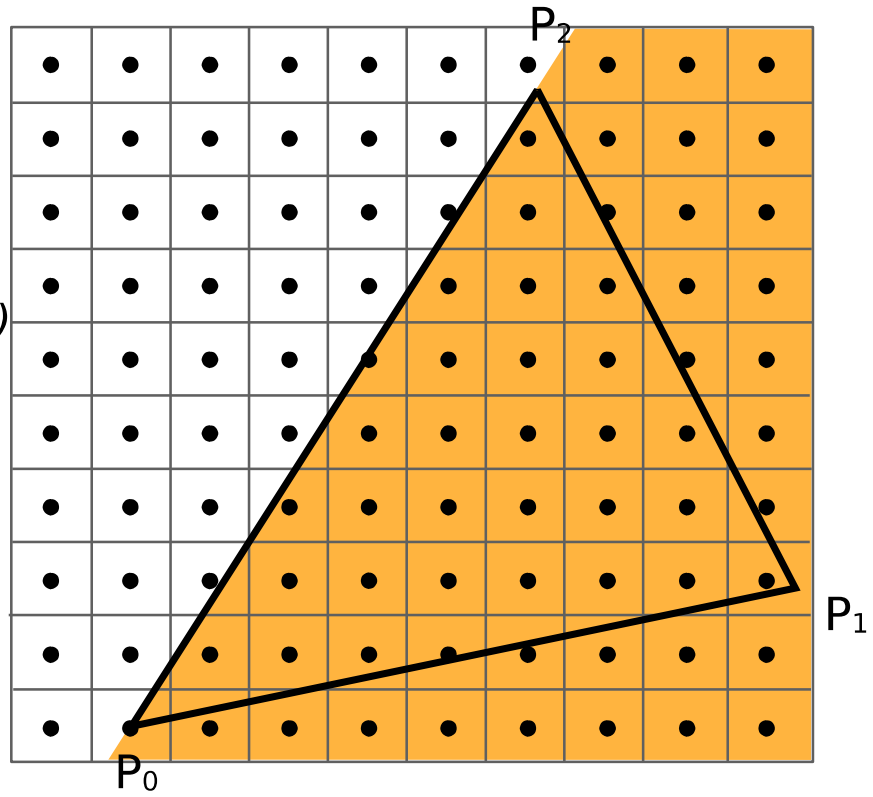
Point-in-Triangle Test: Three Line Tests

$P_i = (X_i, Y_i)$

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

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

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



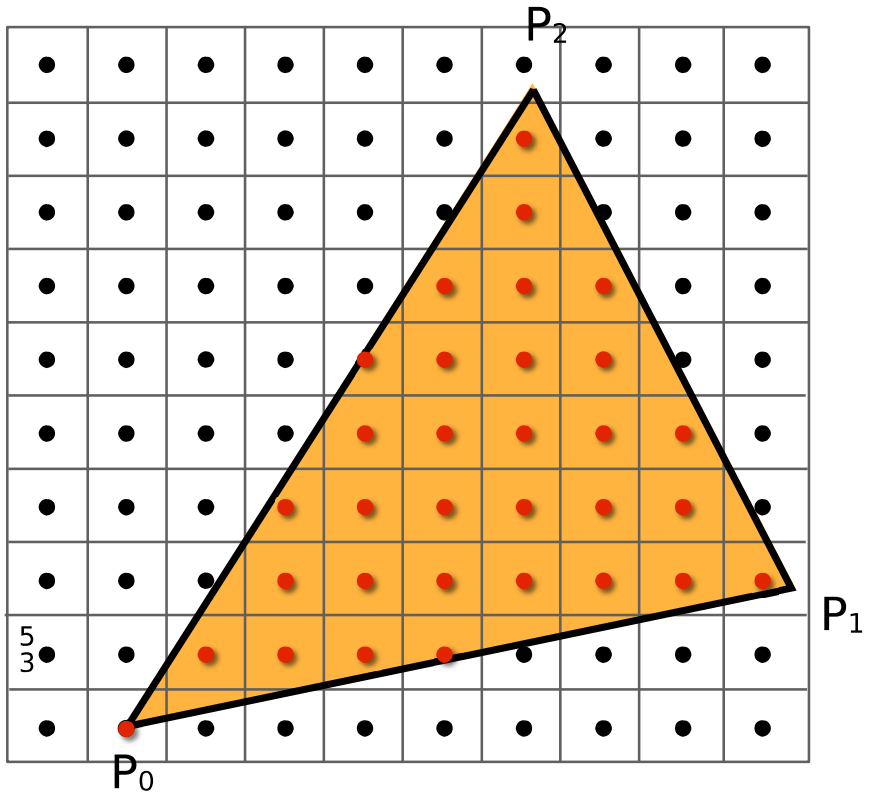
$L_2(x, y) > 0$

Point-in-Triangle Test: Three Line Tests

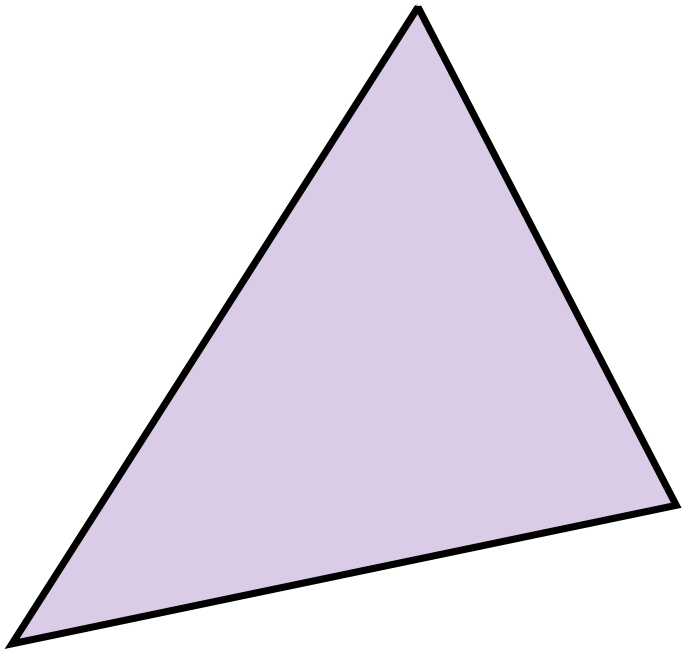
Sample point *s = (sx, sy)* is inside the triangle if it is inside all three lines.

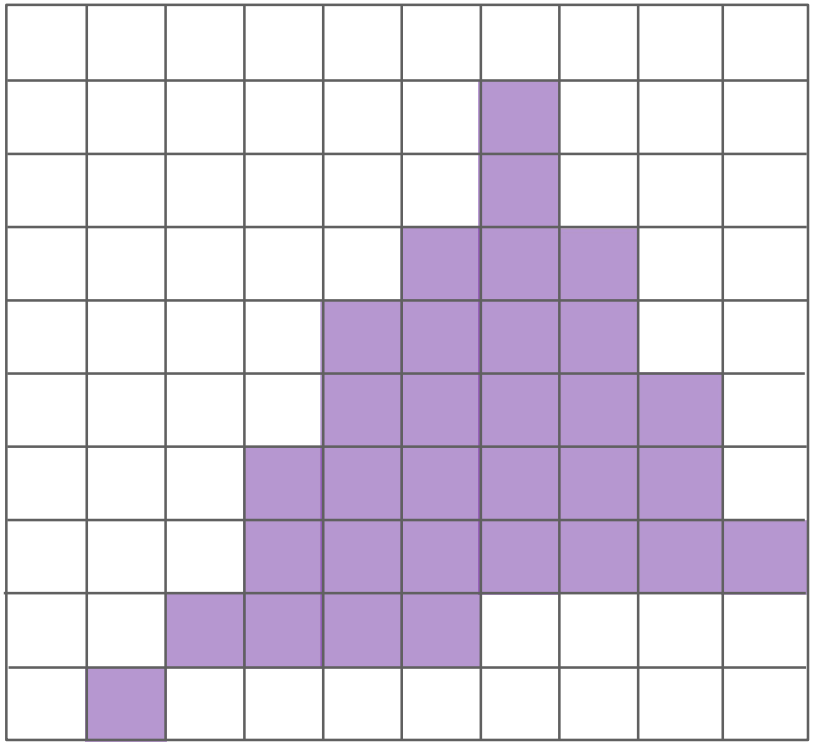
inside(sx, sy) =
L_0 (sx, sy) > 0 &&
L_1 (sx, sy) > 0 &&
L_2 (sx, sy) > 0;

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



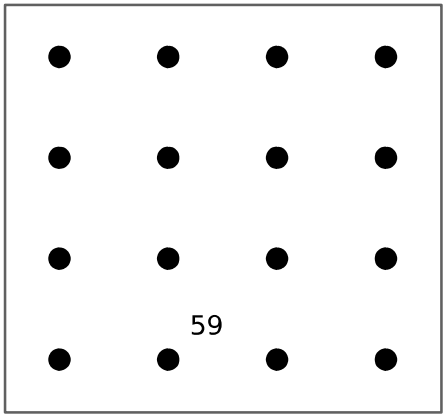
Super-Sampling and Anti-Aliasing





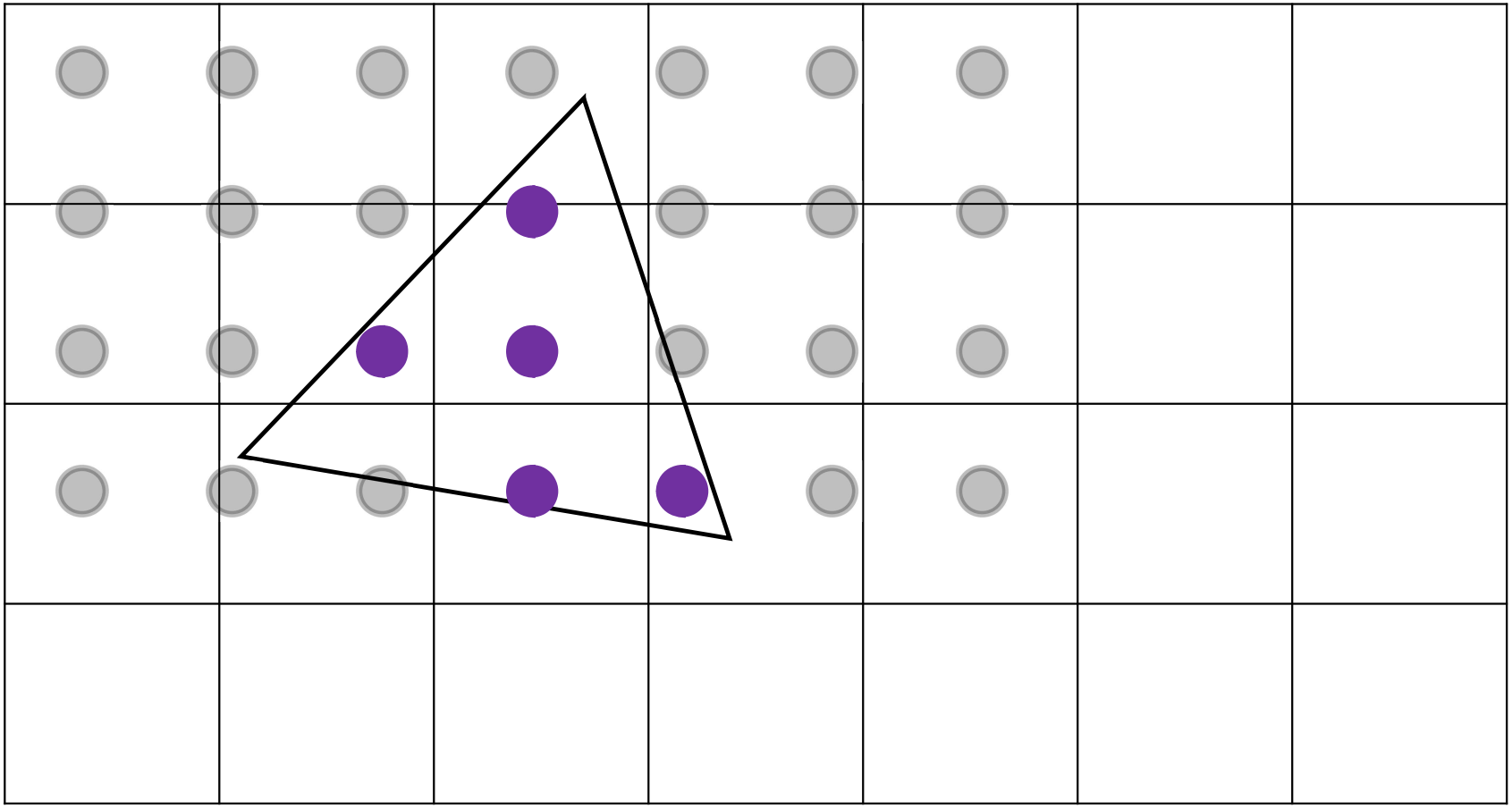
Supersampling

We can approximate the effect of the 1-pixel box filter by sampling multiple locations within a pixel and averaging their values:



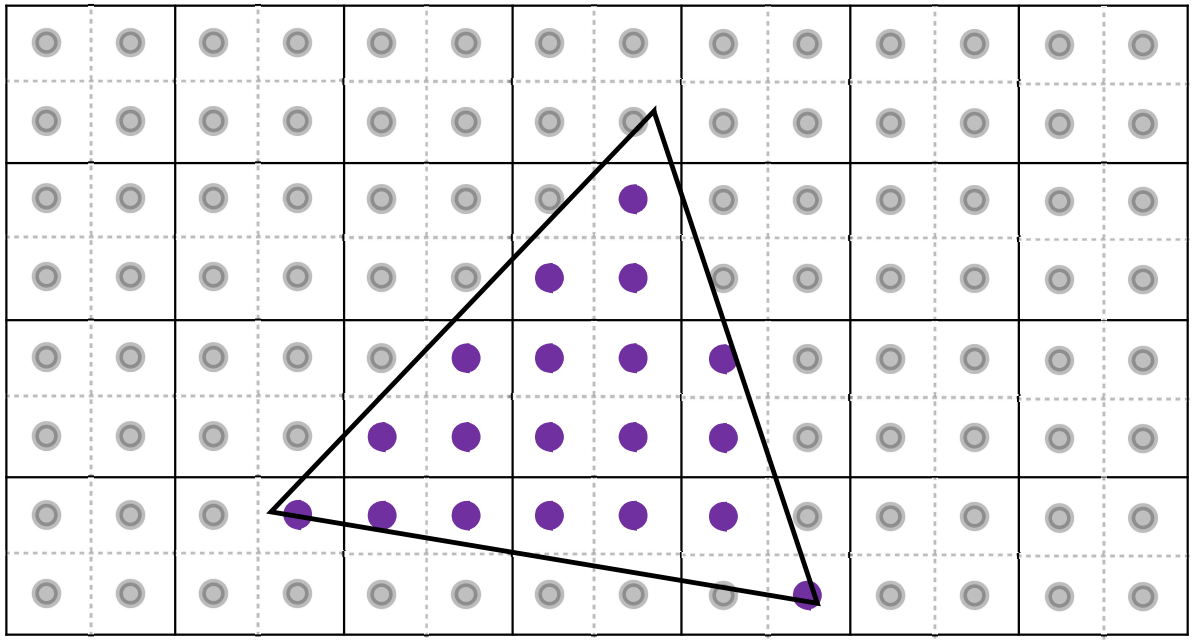
4x4 supersampling

Point Sampling: One Sample Per Pixel



Super-sampling: Step 1

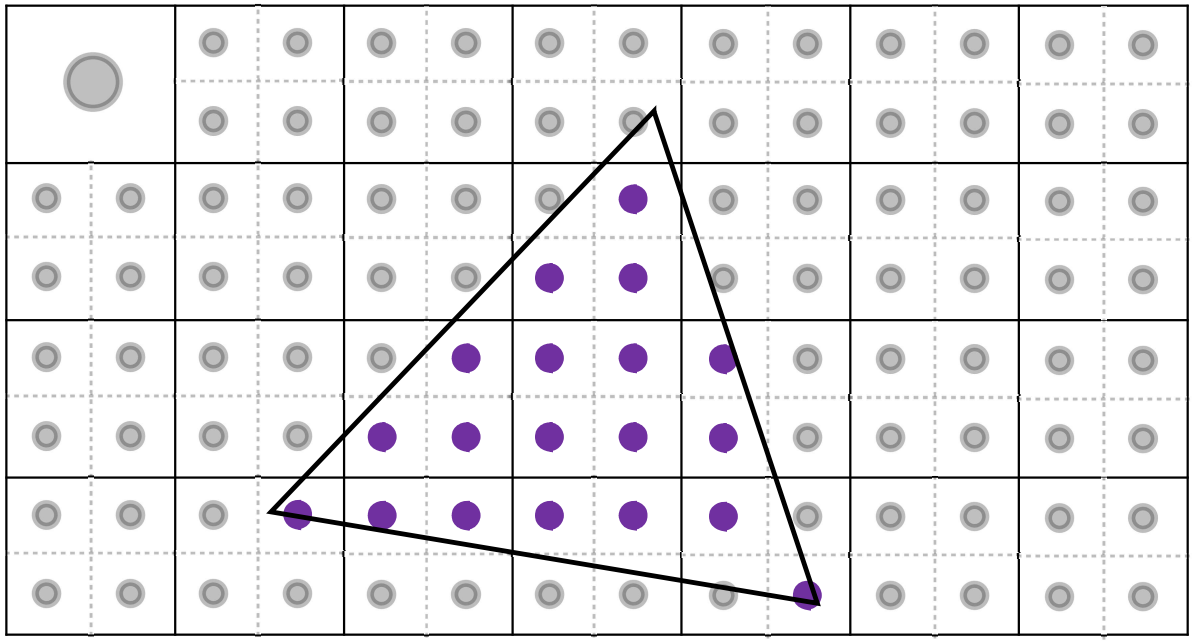
Take NxN samples in each pixel.



2x2 supersampling

Supersampling: Step 2

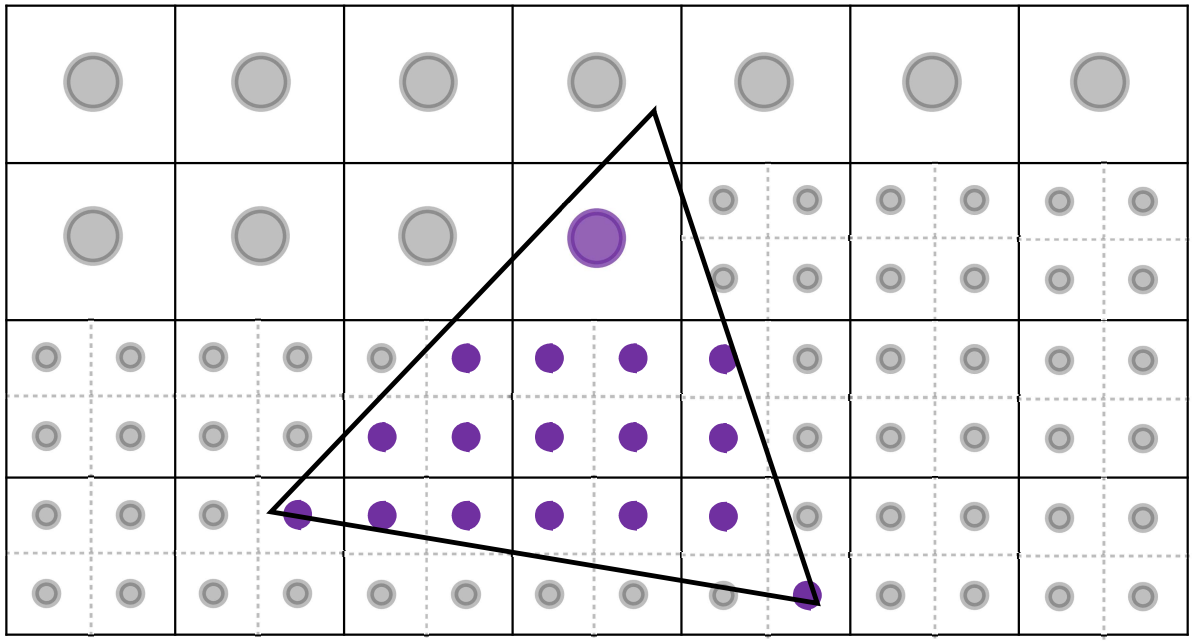
Average the NxN samples “inside” each pixel.



Averaging down

Supersampling: Step 2

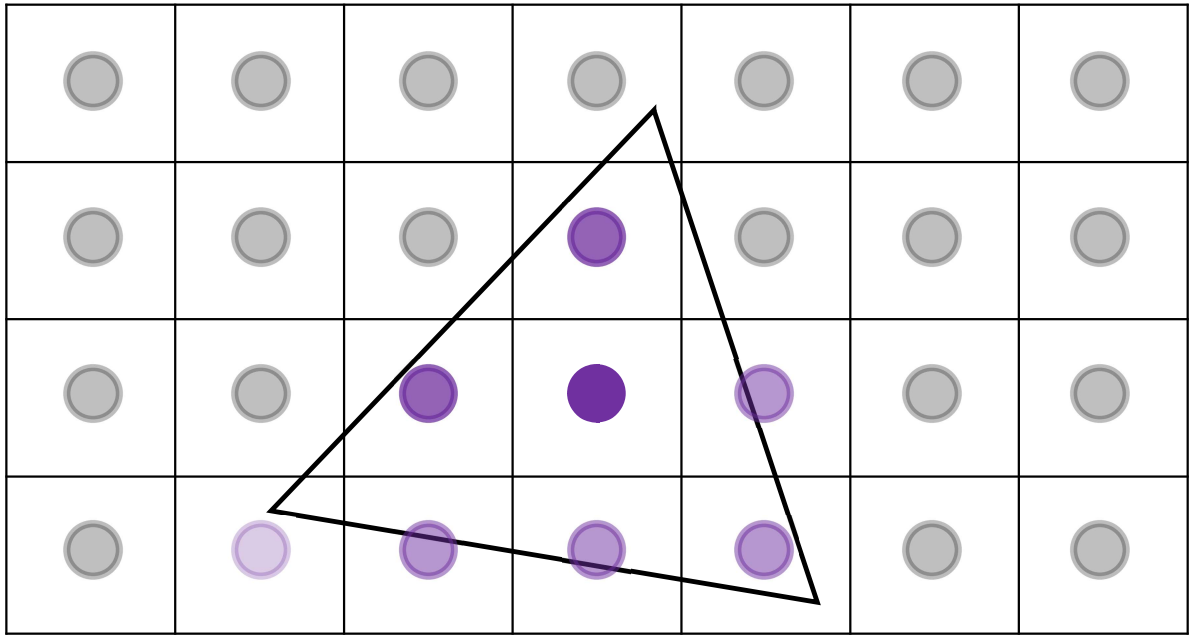
Average the NxN samples “inside” each pixel.



Averaging down

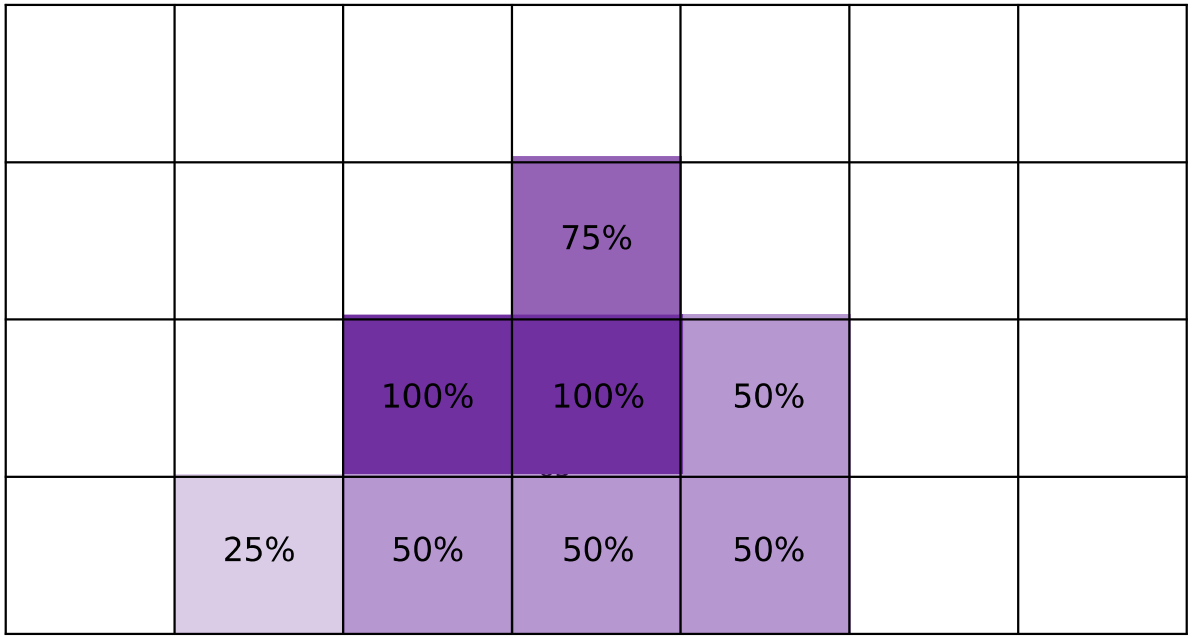
Supersampling: Step 2

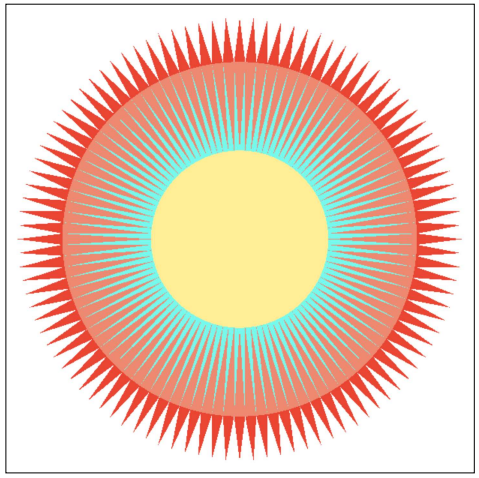
Average the NxN samples “inside” each pixel.

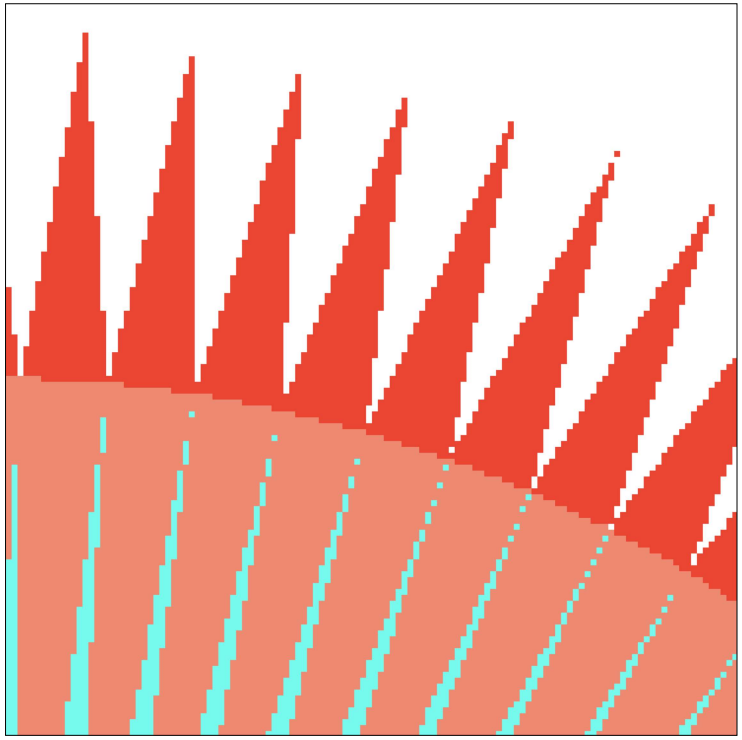


Supersampling: Result

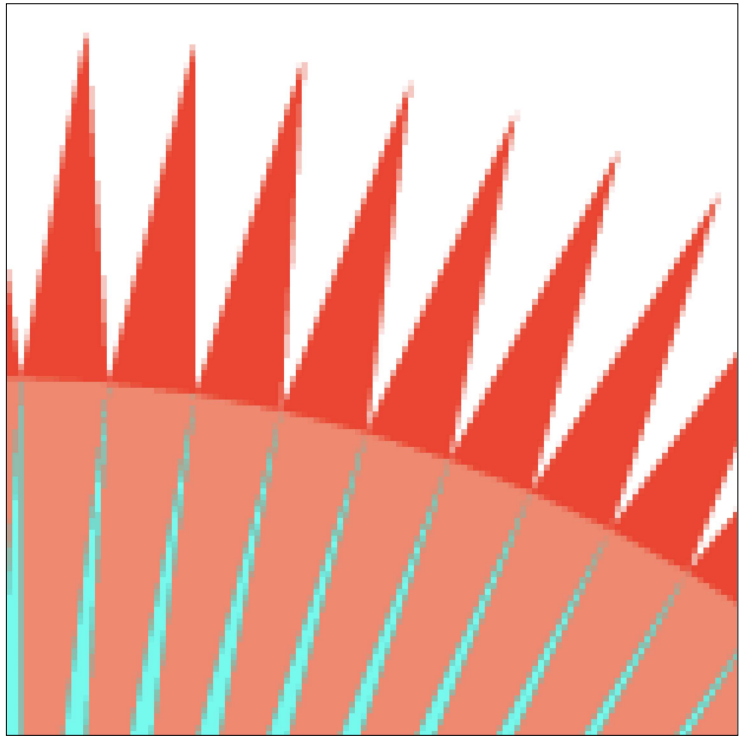
This is the corresponding signal emitted by the display







raw sampling



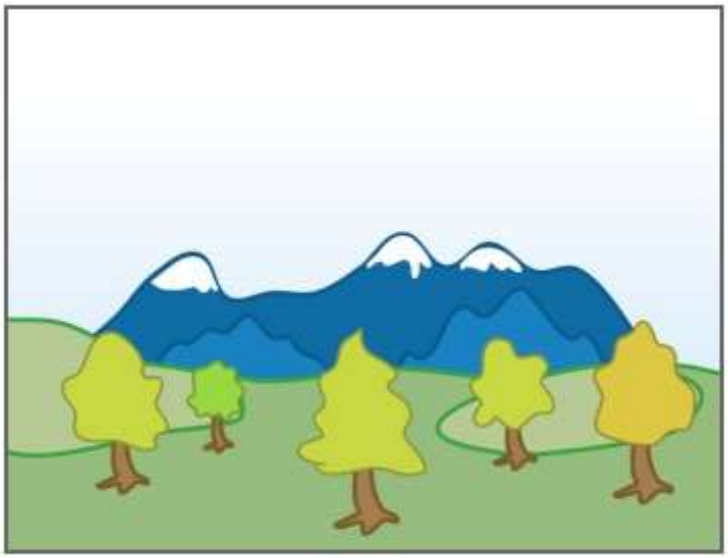
4x4 up-sampling


No AA
SSAA

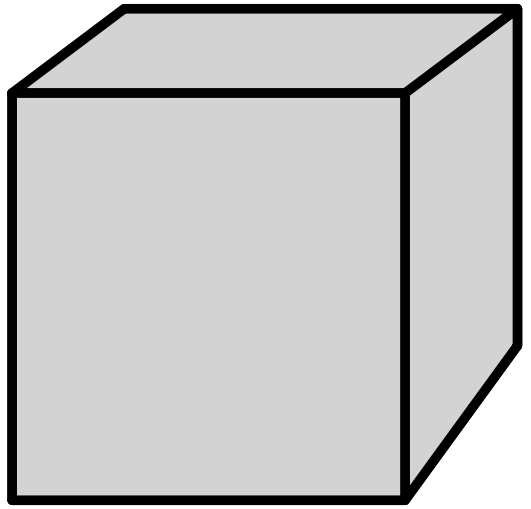
Visibility / Occlusion

Painter’s Algorithm

- Inspired by how painters paint
- Paint from back to front, overwrite in the framebuffer

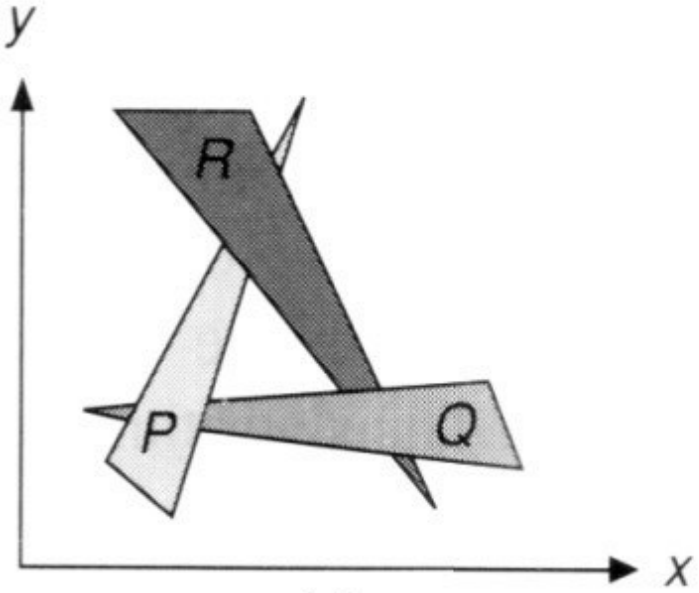


[Wikipedia]



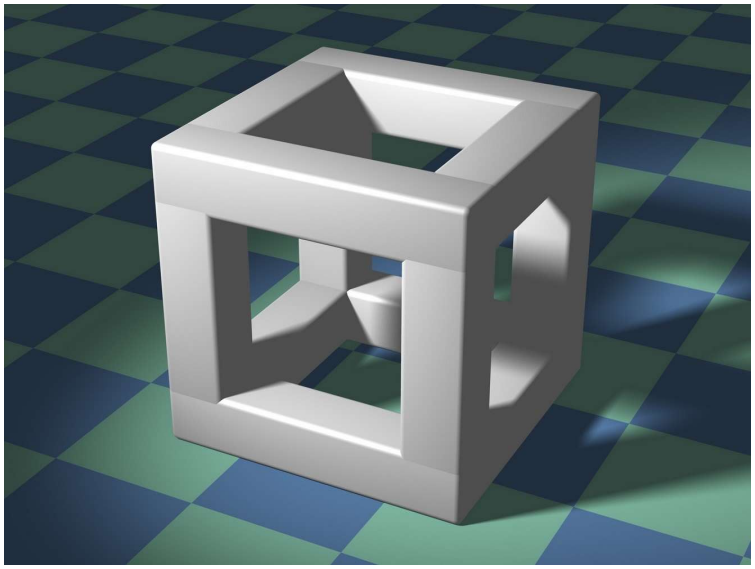
Painter's Algorithm

- Requires sorting in depth (O(n log n) for n triangles)
- Can have unresolvable depth order

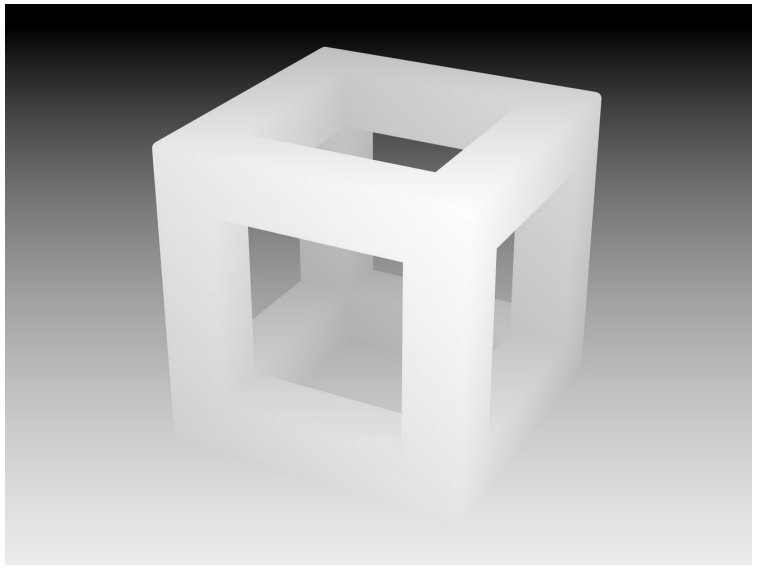


[Foley et al.]

Z-Buffer Example



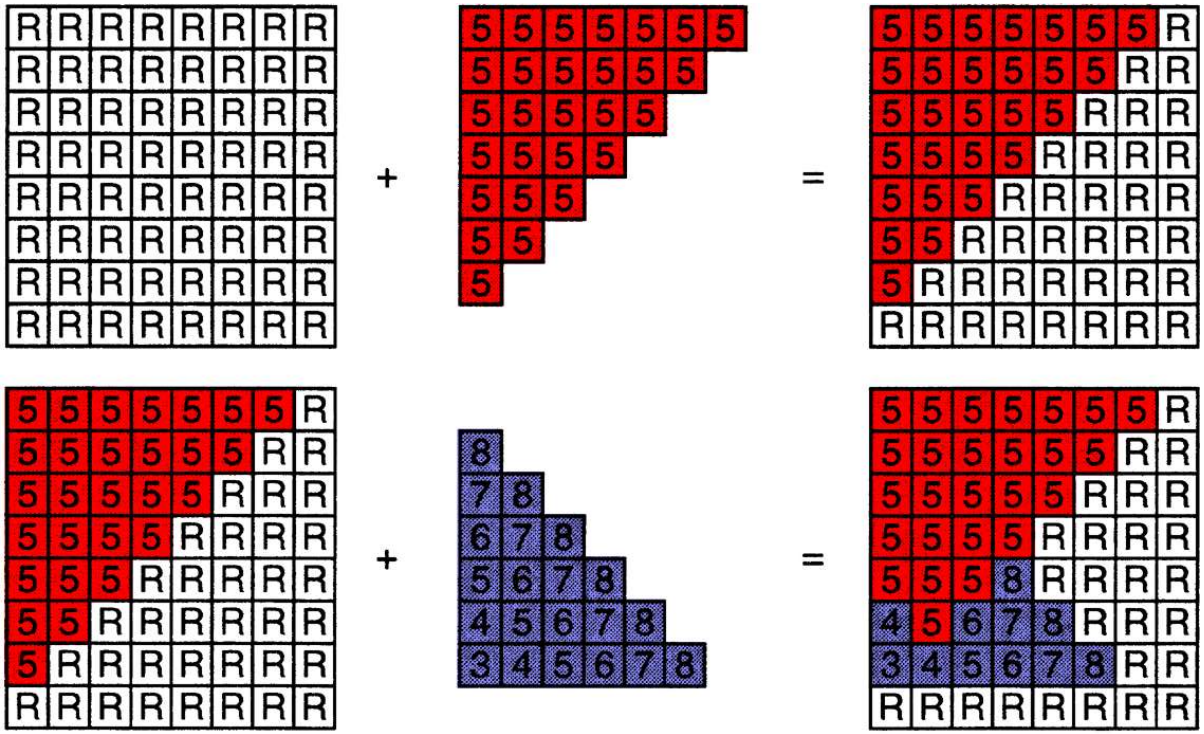
Rendering



Depth / Z buffer

Image credit: Dominic Alves, flickr.

Z-Buffer Algorithm



Graphics Pipeline = Abstract Drawing Machine