Lecture 1:

Course Introduction:
Welcome to Computer Graphics!

Computer Graphics: Rendering, Geometry, and Image Manipulation
Stanford CS248A, Winter 2026

Hi!

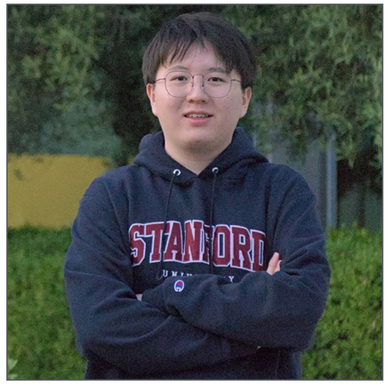
Josephine
the (Graphics) Cat

Kayvon Fatahalian

Tejan

Lvmin

Fangjun



Discussion:
Why *do you* want to study computer graphics?

Movies

Avatar: Fire and Ash (2025)

Computer games

This image is rendered in real-time on a modern GPU

Supercomputing for games

NVIDIA Founder's Edition RTX 4090 GPU

~ 82 TFLOPs fp32 *

*** Doesn't include additional 190 TFLOPS of ray tracing compute and 165 TFLOPS of fp15 DNN compute**

Specialized processors for performing graphics computations.

Virtual reality experiences

Augmented reality experiences

~11.4M visible pixels per panel
(28 Mpixel display)

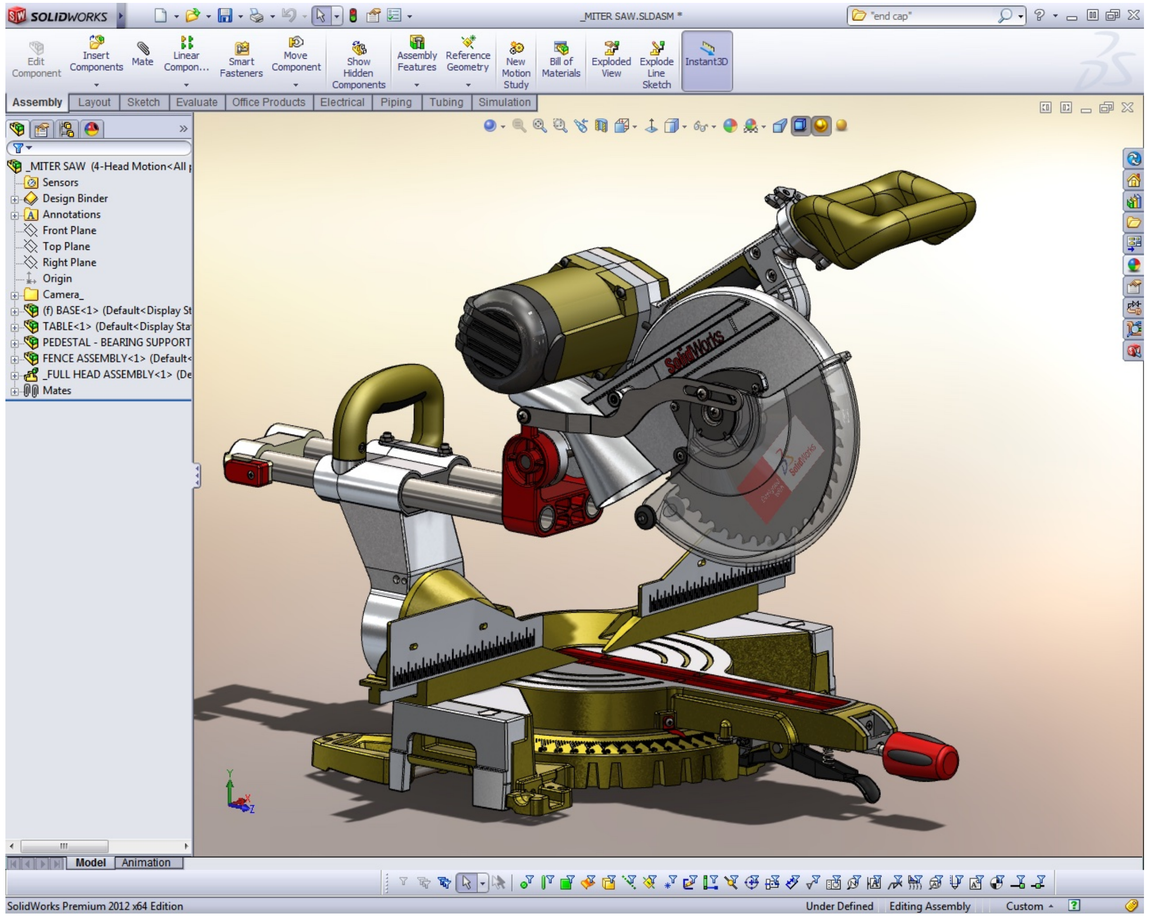
Apple Vision Pro

Digital illustration

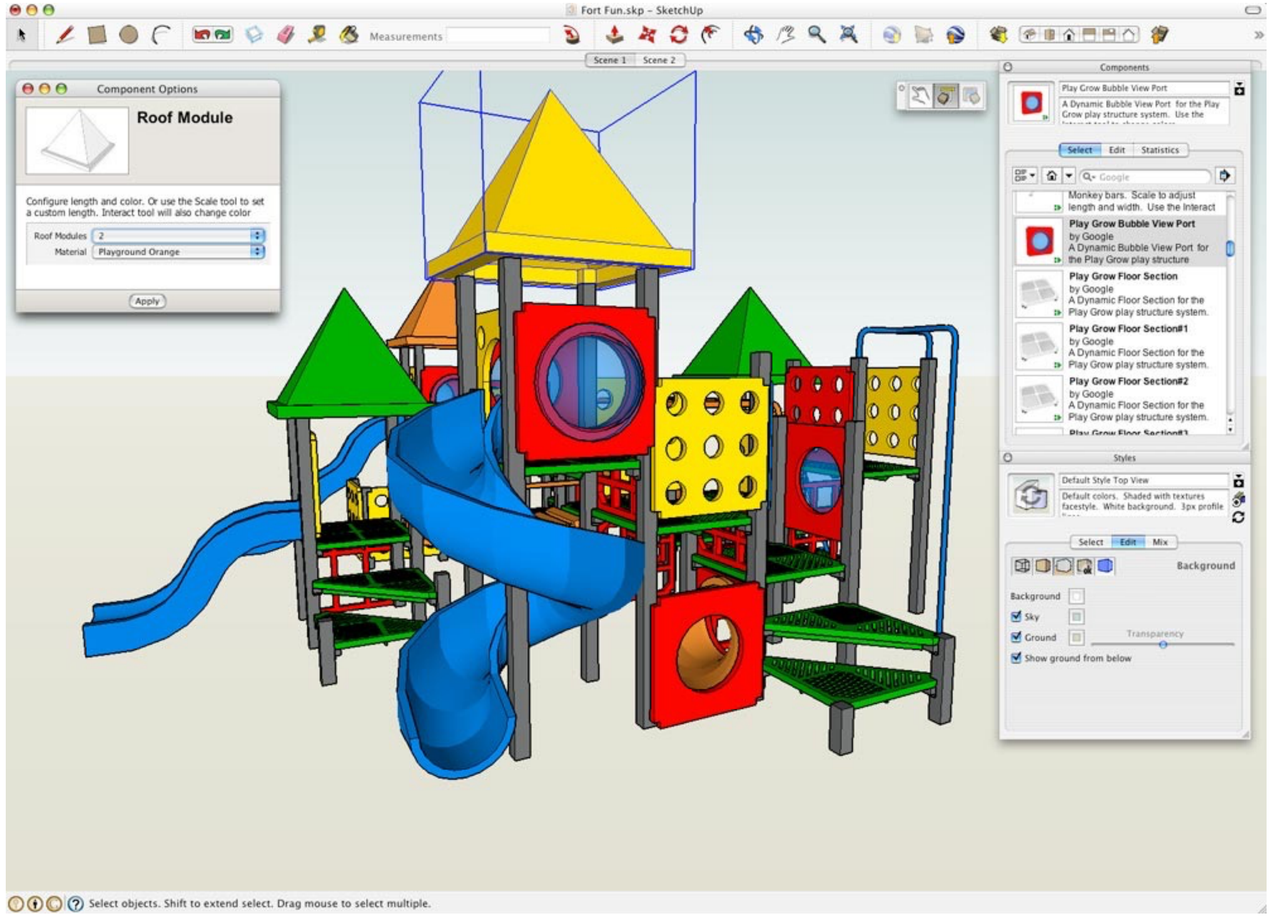
Meike Hakkart

http://maquenda.deviantart.com/art/Lion-done-in-illustrator-327715059

Computer aided design



SolidWorks



SketchUp

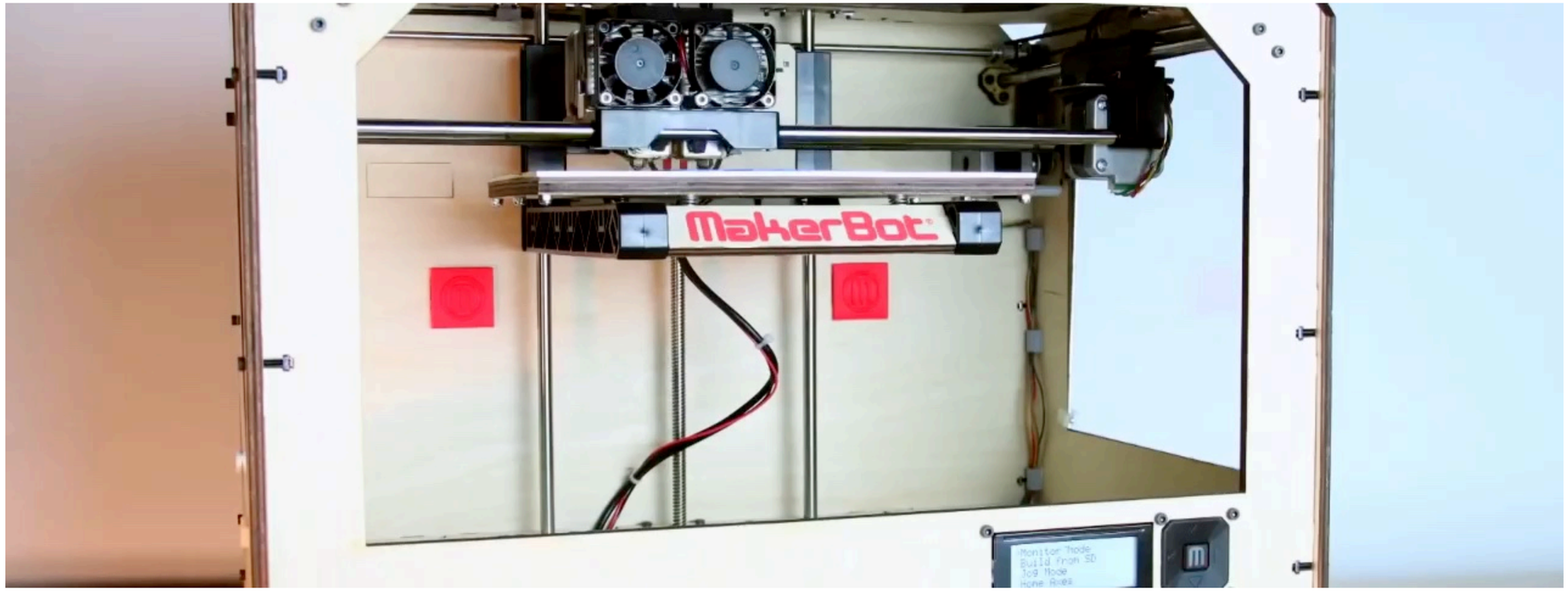
For mechanical, architectural, electronic, optical, …

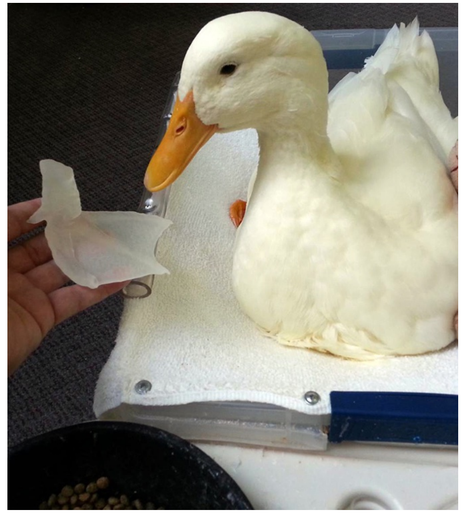
Product design and visualization

Ikea - 75% of catalog is rendered imagery (several years ago… likely a lot more now)

Architectural design

3D fabrication



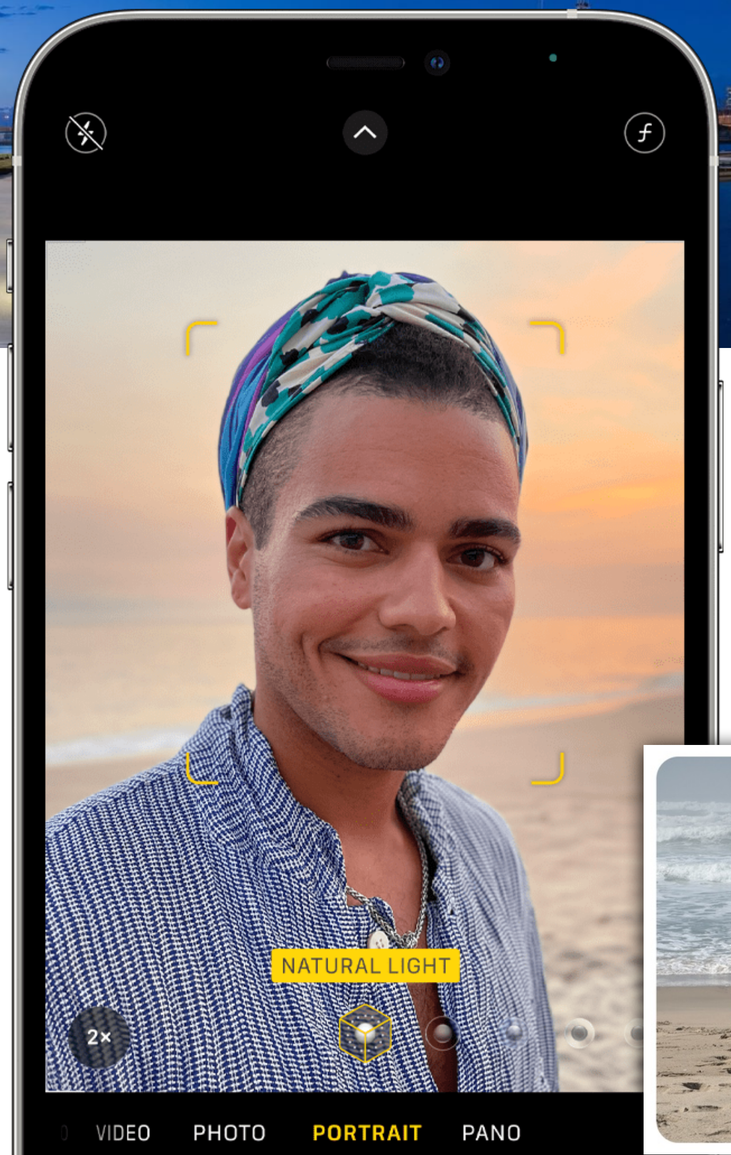


Modern graphical user interfaces

2D drawing and animation are ubiquitous in computing.
Typography, icons, images, transitions, transparency, …
(all rendered at high frame rate for rich experience)

Computational cameras

Panoramic stitching

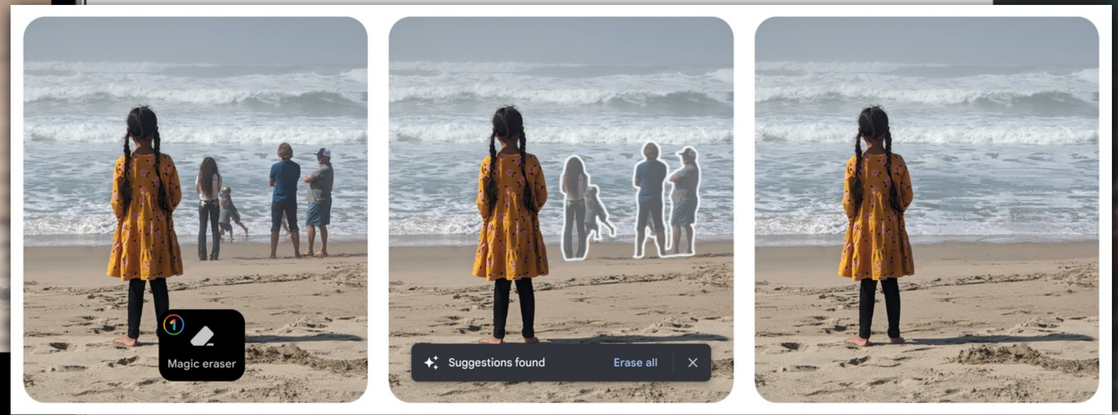


Portrait mode

(simulate effects of large aperture lens that is not possible to fit in a smartphone form factor)

Magic Eraser Feature

(Edit image by detecting objects and removing them)

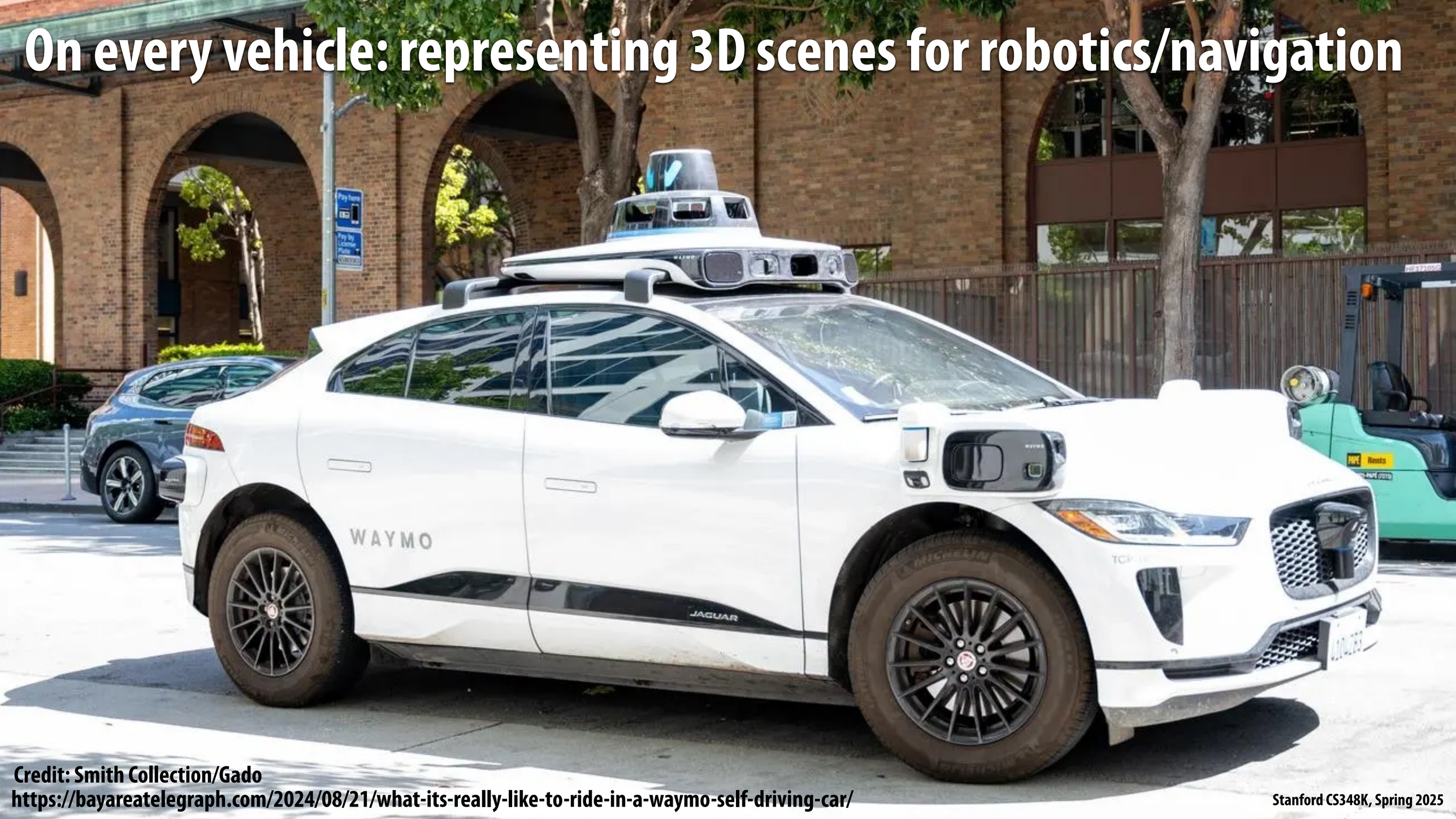


High dynamic range (HDR) photography

Turning sequences of photographs into 3D worlds

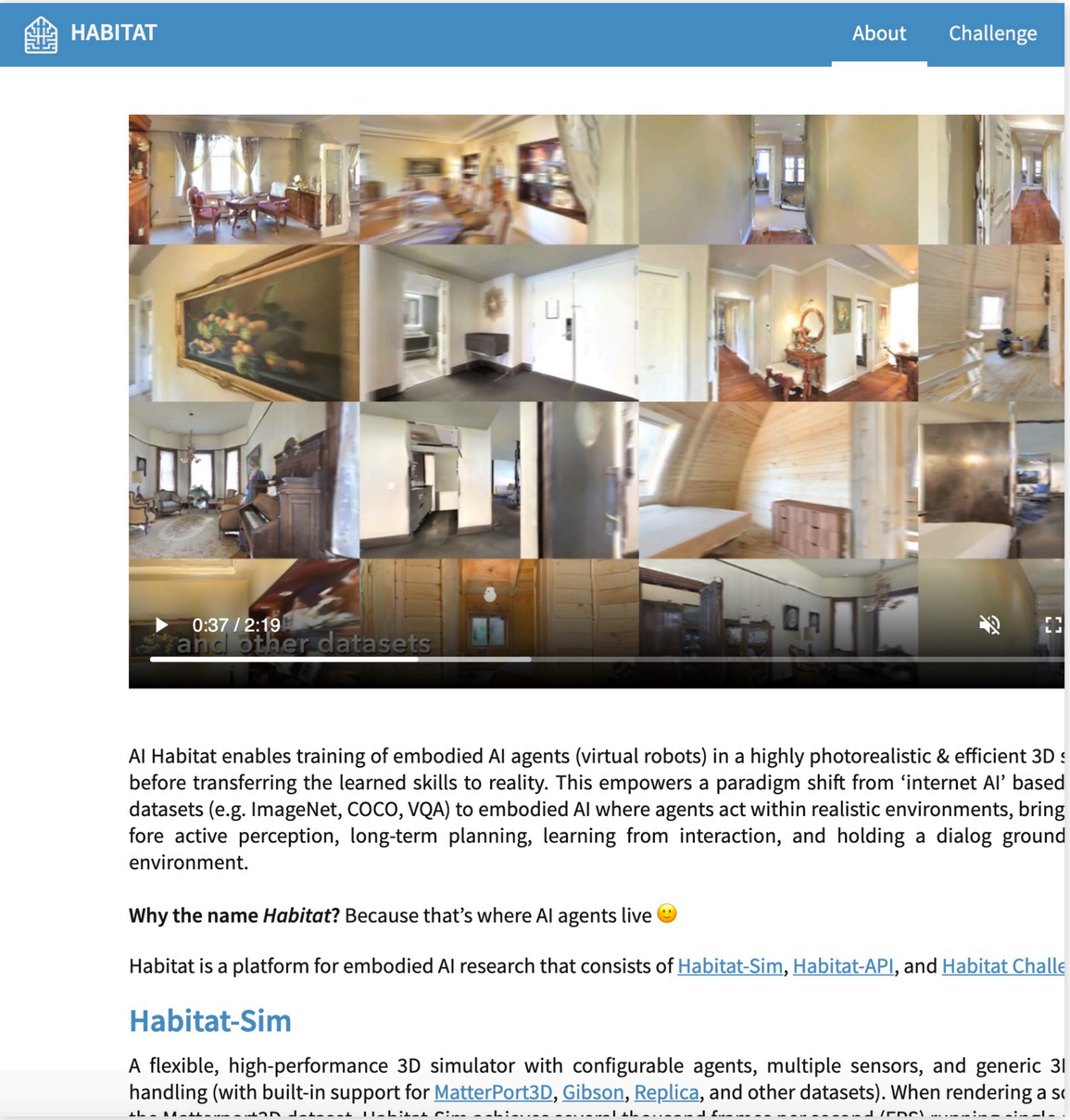
Kerbl et al. 2023

On every vehicle: representing 3D scenes for robotics/navigation

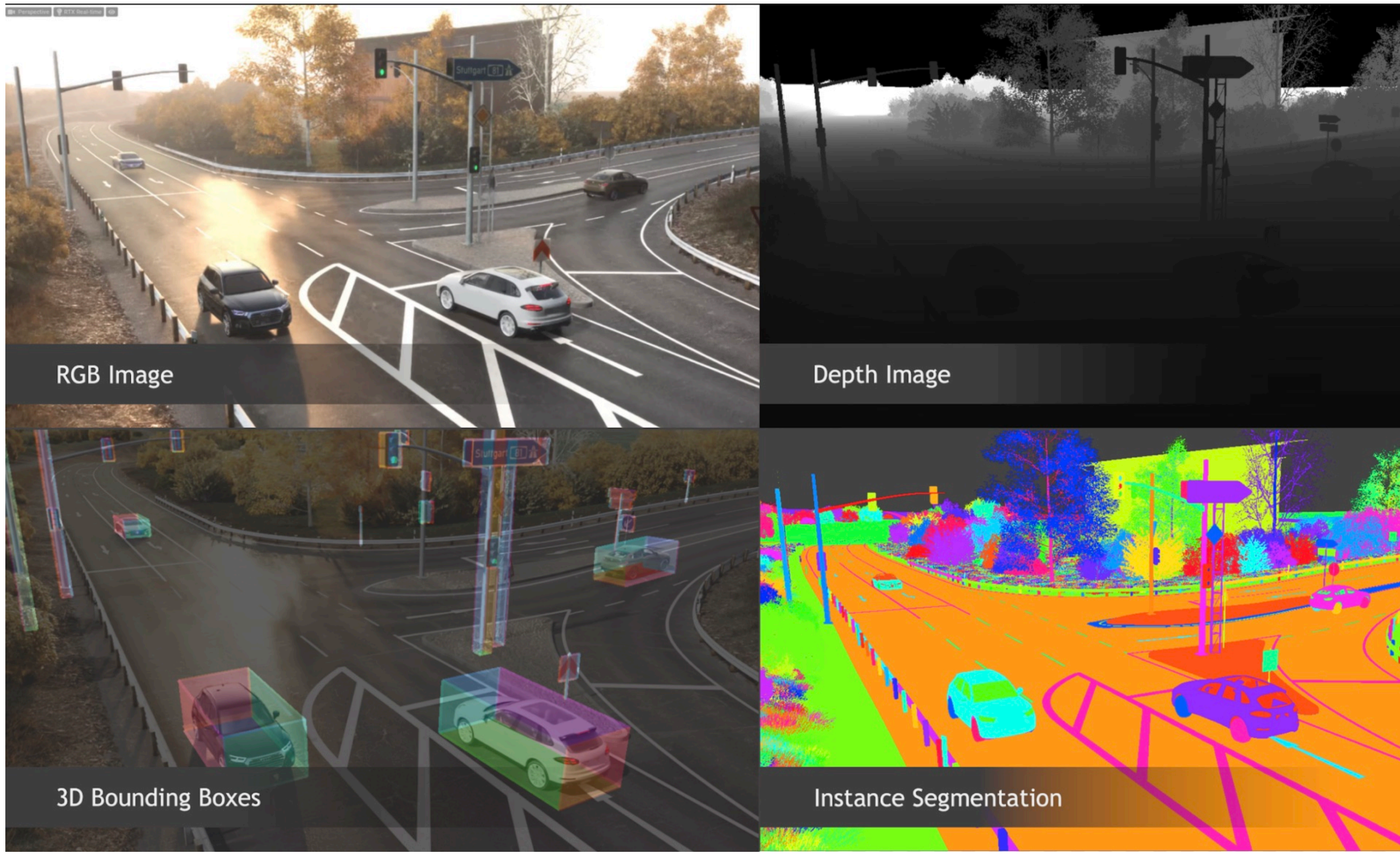


Credit: Smith Collection/Gado
https://bayareatelegraph.com/2024/08/21/what-its-really-like-to-ride-in-a-waymo-self-driving-car/

Graphics/simulation used for training ML models



AI Habitat:
simulator for training AI agents



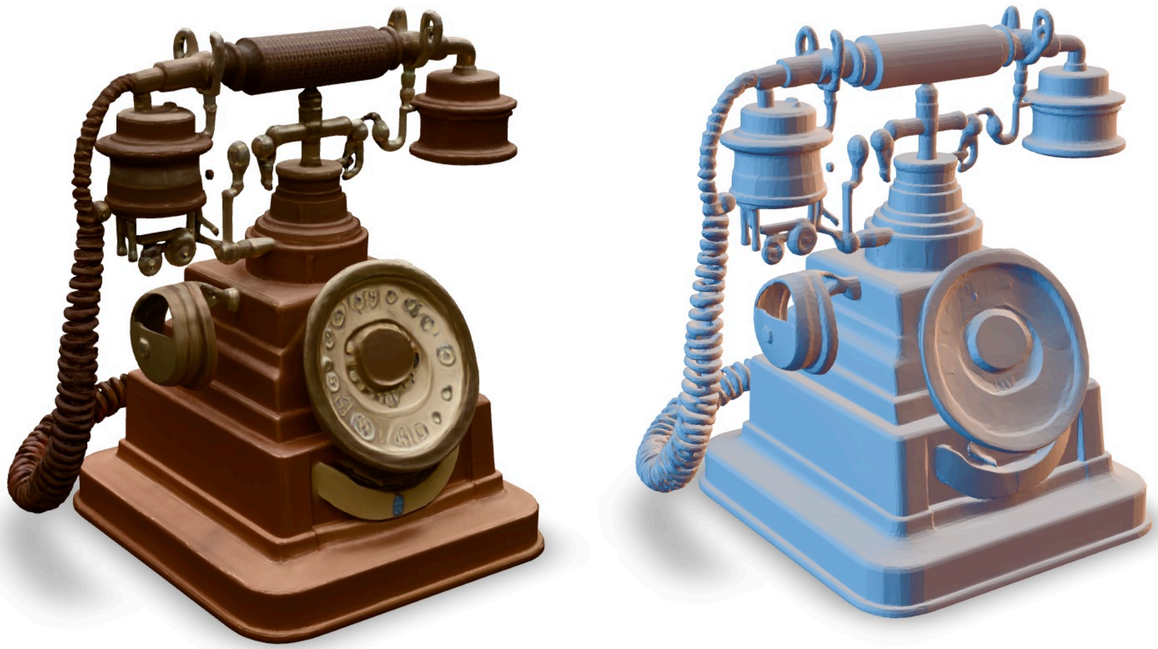
NV Drive Sim:
autonomous driving simulator

Transformative generative AI capabilities

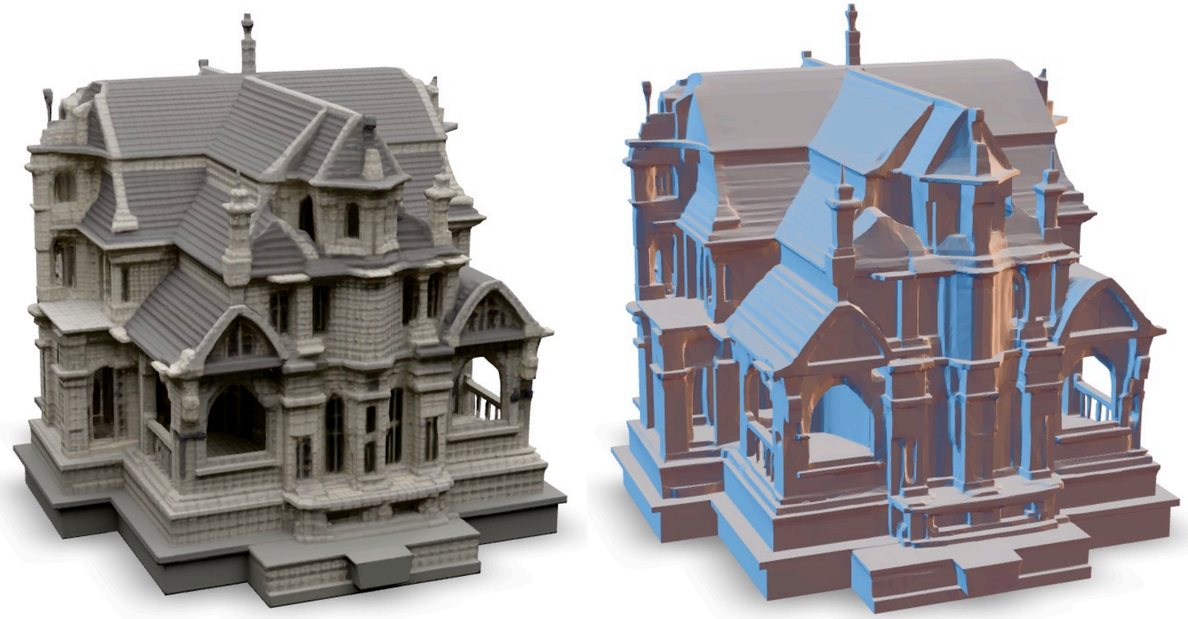
**"A bento box with rice,
edamame, ginger,
and sushi.
Top down view,
white background.
Sushi in right bin of bento box.
Edamame in top left."**

Emerging generative AI for creating textured 3D meshes

"Vintage copper rotary telephone with intricate detailing."



"A Victorian mansion made of stone bricks with ornate trim, bay windows, and a wraparound porch."



[Structured 3D Latents for Scalable and Versatile 3D Generation, Xiang et al. 2024]

Foundations of computer graphics

- All these applications demand *sophisticated* theory and systems

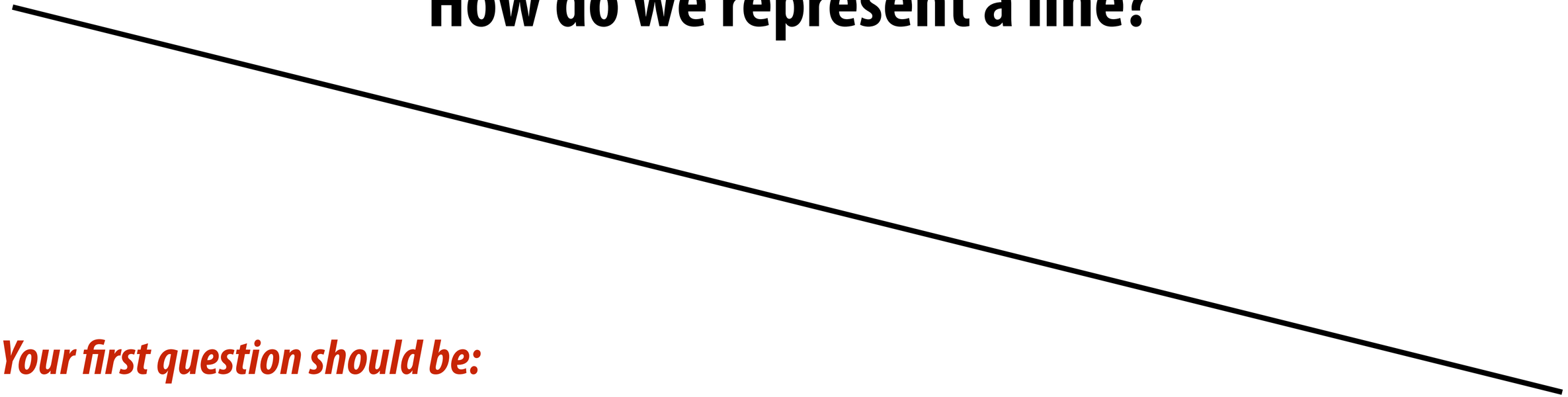
- Science and mathematics
  - Physics of light, color, optics
  - Math of curves, surfaces, geometry, perspective, …
  - Sampling
  - Machine learning and optimization

- Systems
  - Parallel, heterogeneous processing
  - Graphics-specific programming systems
  - Input/output devices

- Art and psychology
  - Perception: color, stereo, motion, image quality, …
  - Art and design: composition, form, lighting, …

Let's talk about representing shapes

How about something very simple…
How do we represent a line?

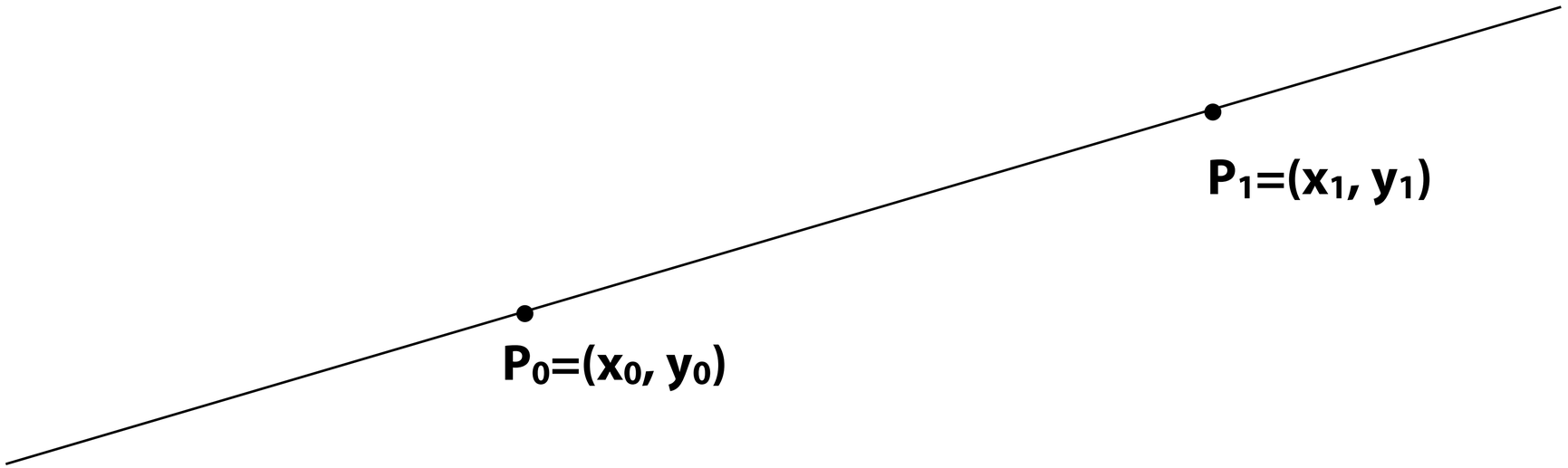


Your first question should be:
Well Kayvon… what operations do you want to do with lines?
(In other words… what's the task?)

Here are a few spatial reasoning tasks involving lines

- Enumerate points that are on the line L:
  - Given x coordinate of a point p on the line L has value x_0, what is the y coordinate of p
  - Give a list of points p' along the line L that are separated by distance D
- Is a given point p=(x,y) on the line L?
- How far is a point p from the line L?
- What side of the line L is point p on?
- Render a line L in an image

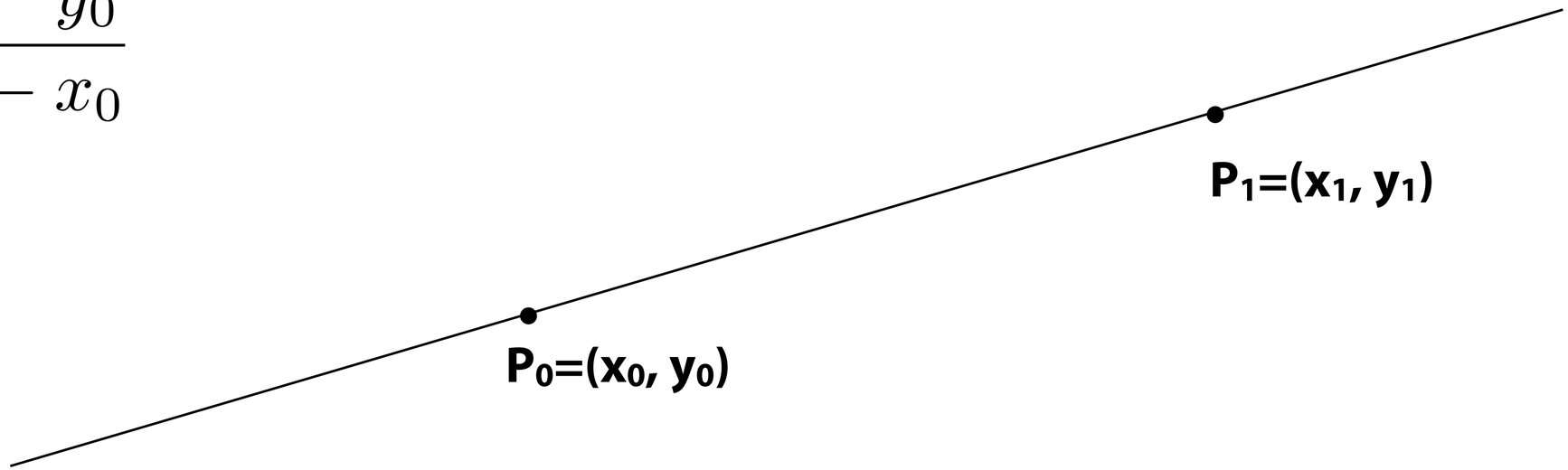
Given points P0 and P1, give a representation for the line connecting the two points.



Point-slope form of a line

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

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



Convenient representation for the following task:

- **Given points P_0 and P_1, give equation for the line connecting the two points.**

Slope-intercept form of a line

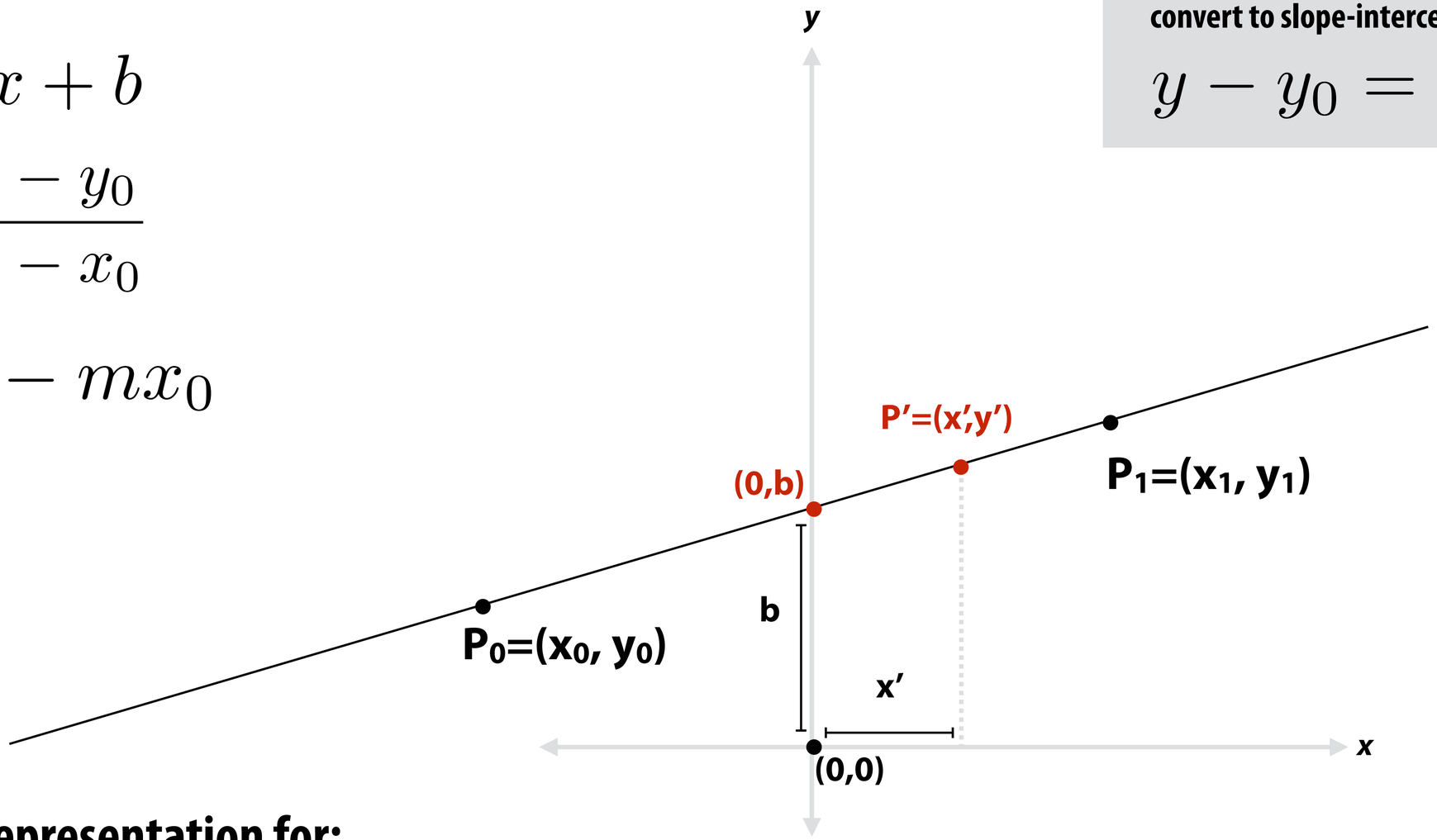
$$y = mx + b$$

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

$$b = y_0 - mx_0$$

Given point-slope form (below), how do you convert to slope-intercept form?

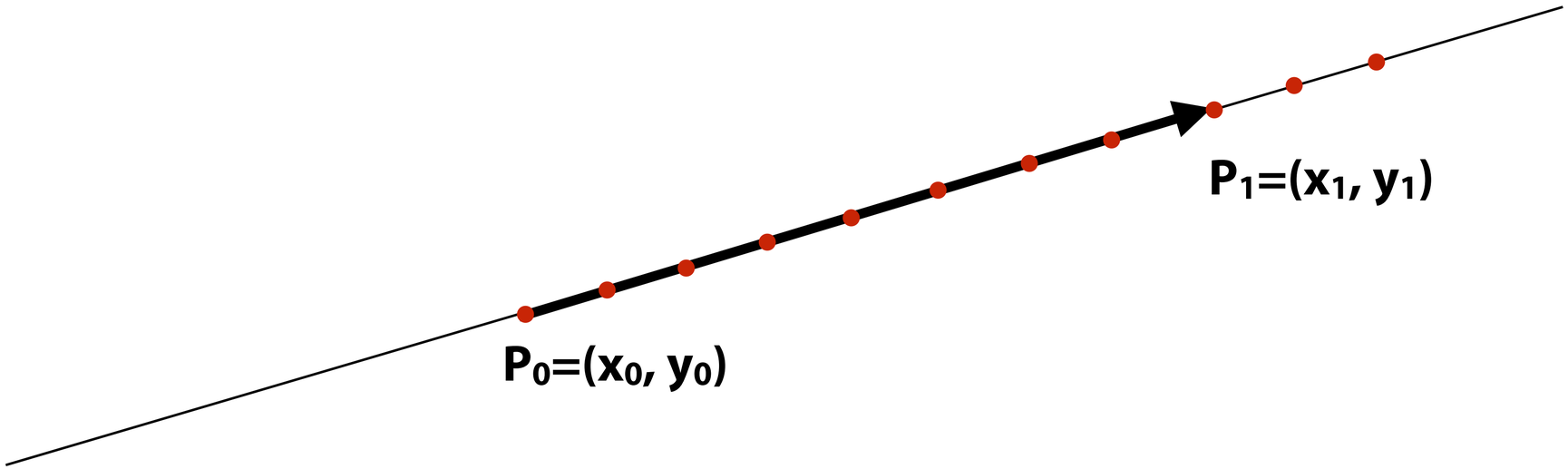
$$y - y_0 = m(x - x_0)$$



Convenient representation for:

- **Where does the line cross the Y axis? (At y=b!)**
- **Given point P' on the line located x' on the x axis, what's y-coordinate of P'?**

New task:
Enumerate points spaced by distance *d* on L starting at P_0 and moving toward P_1?



Parametric form of line

point that is distance *t* along line *L* from P_0.

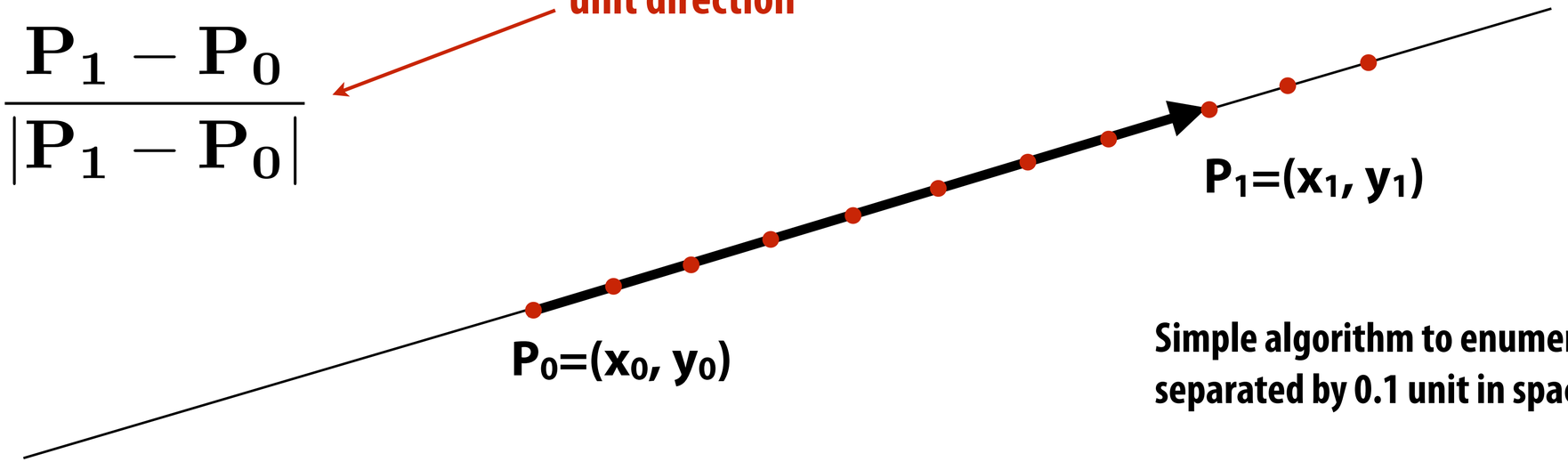
$$\mathbf{L}(t) = \mathbf{P_0} + t\mathbf{d}$$

unit direction

$$\mathbf{d} = \frac{\mathbf{P_1} - \mathbf{P_0}}{|\mathbf{P_1} - \mathbf{P_0}|}$$

This what's called an "explicit representation" of the line: Given a parameter (*t*) identifying a location on the line, the equation provides the point in space.

Note, the vector representation holds for lines in any dimensional space (2D, 3D, etc.)



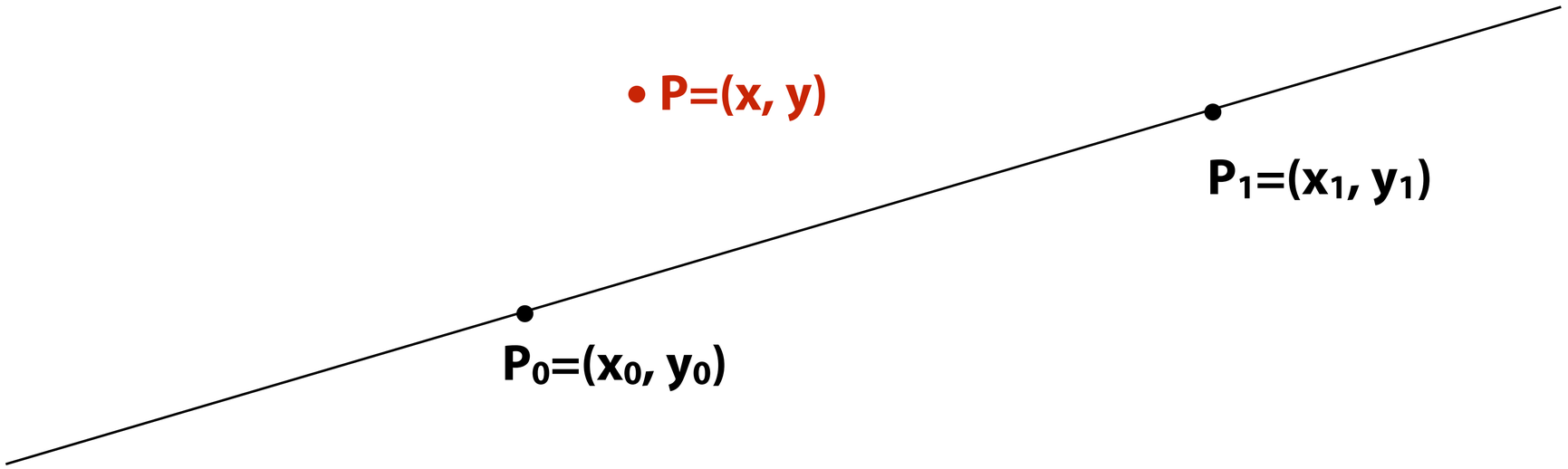
Simple algorithm to enumerate points on line separated by 0.1 unit in space:

```
let dt = 0.1
for (i =0 to N)
   evaluate L(i * dt)
```

Convenient representation for the following task:

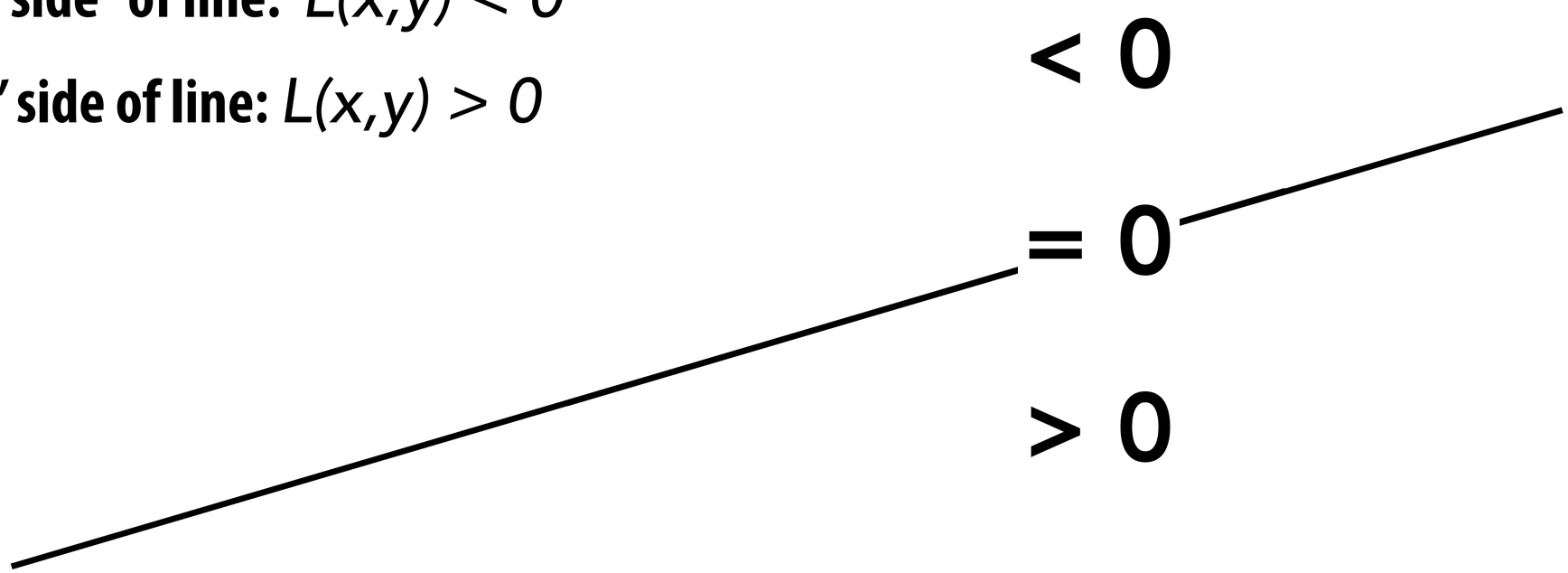
- **Enumerating points on the line given a distance from P_0**

New task:
Is a given point P=(x,y) on the line *L* defined by points P_0 and P_1?
If it's not, what side is it on?



Implicit form of a line

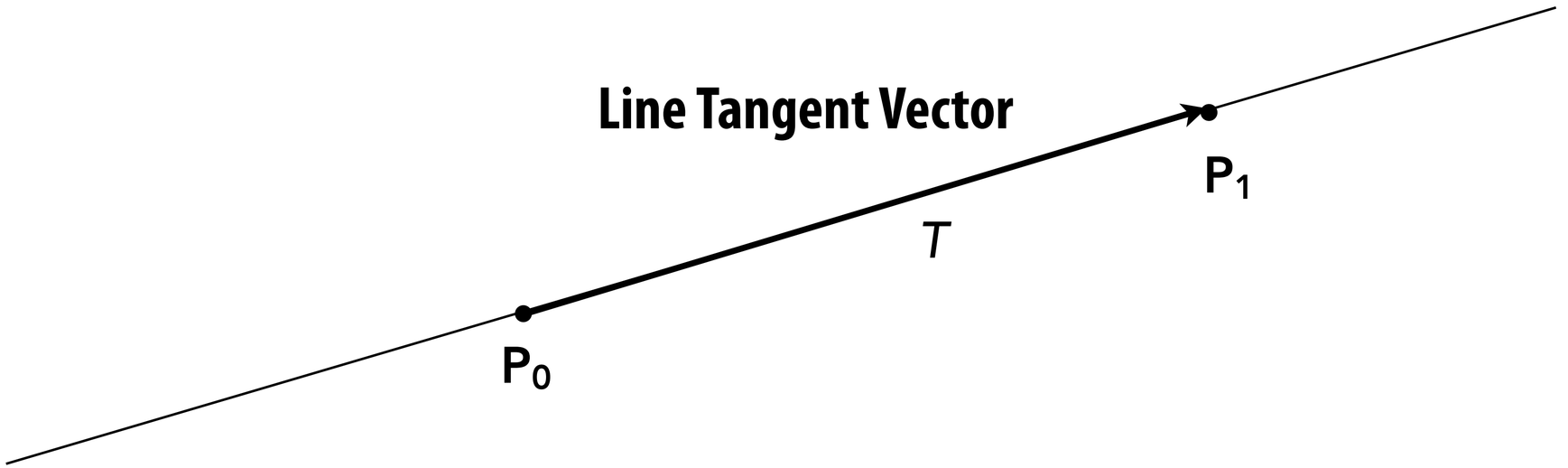
- **Implicit line equation**
  - $L(x,y) = Ax + By + C$
  - **On the line:** $L(x,y) = 0$
  - **"Negative side" of line:** $L(x,y) < 0$
  - **"Positive" side of line:** $L(x,y) > 0$



Convenient representation for the following task:

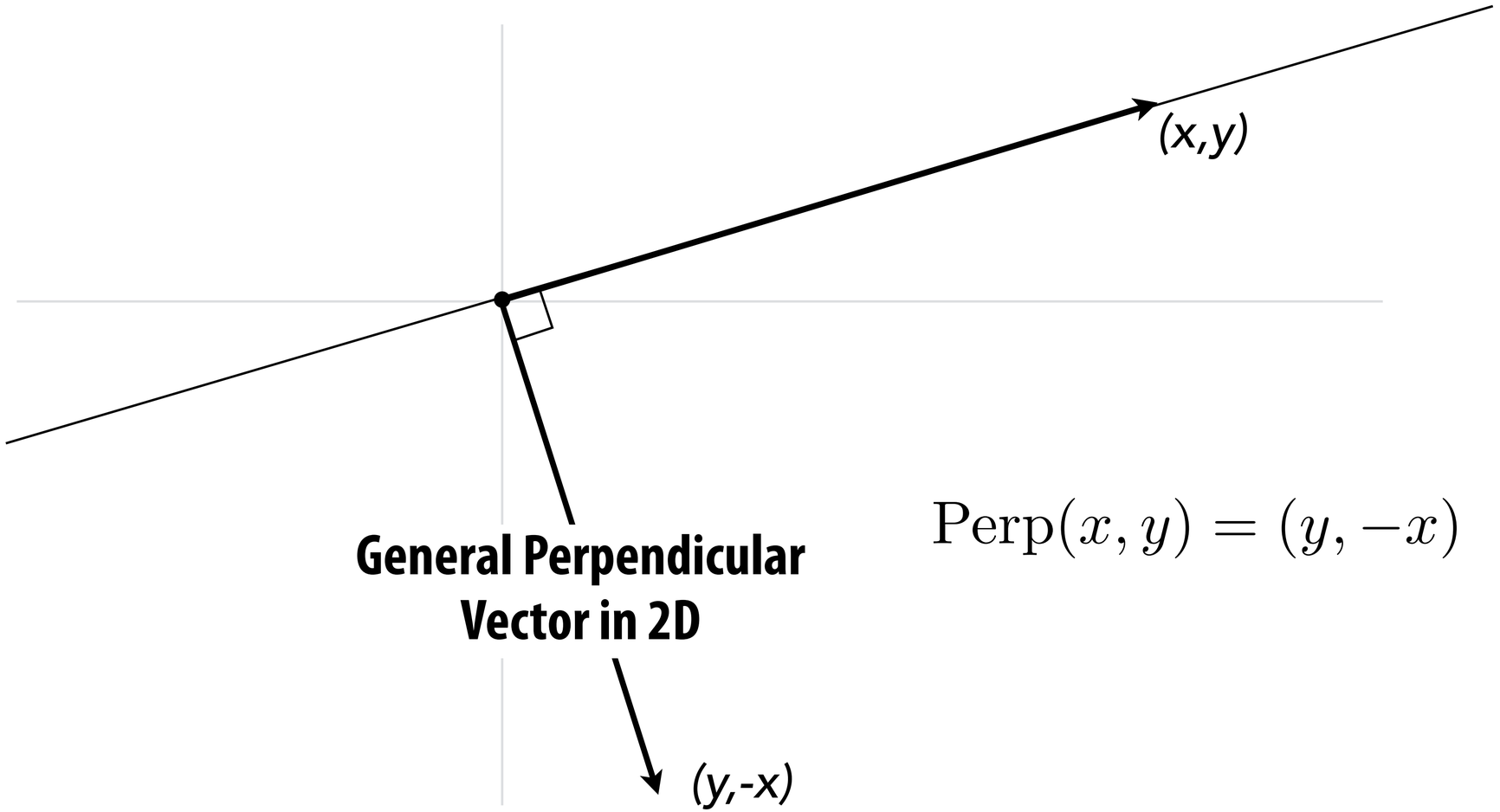
- **Determining if a given point *P* is on the line, or what side of the line the point is on.**

Line equation derivation



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

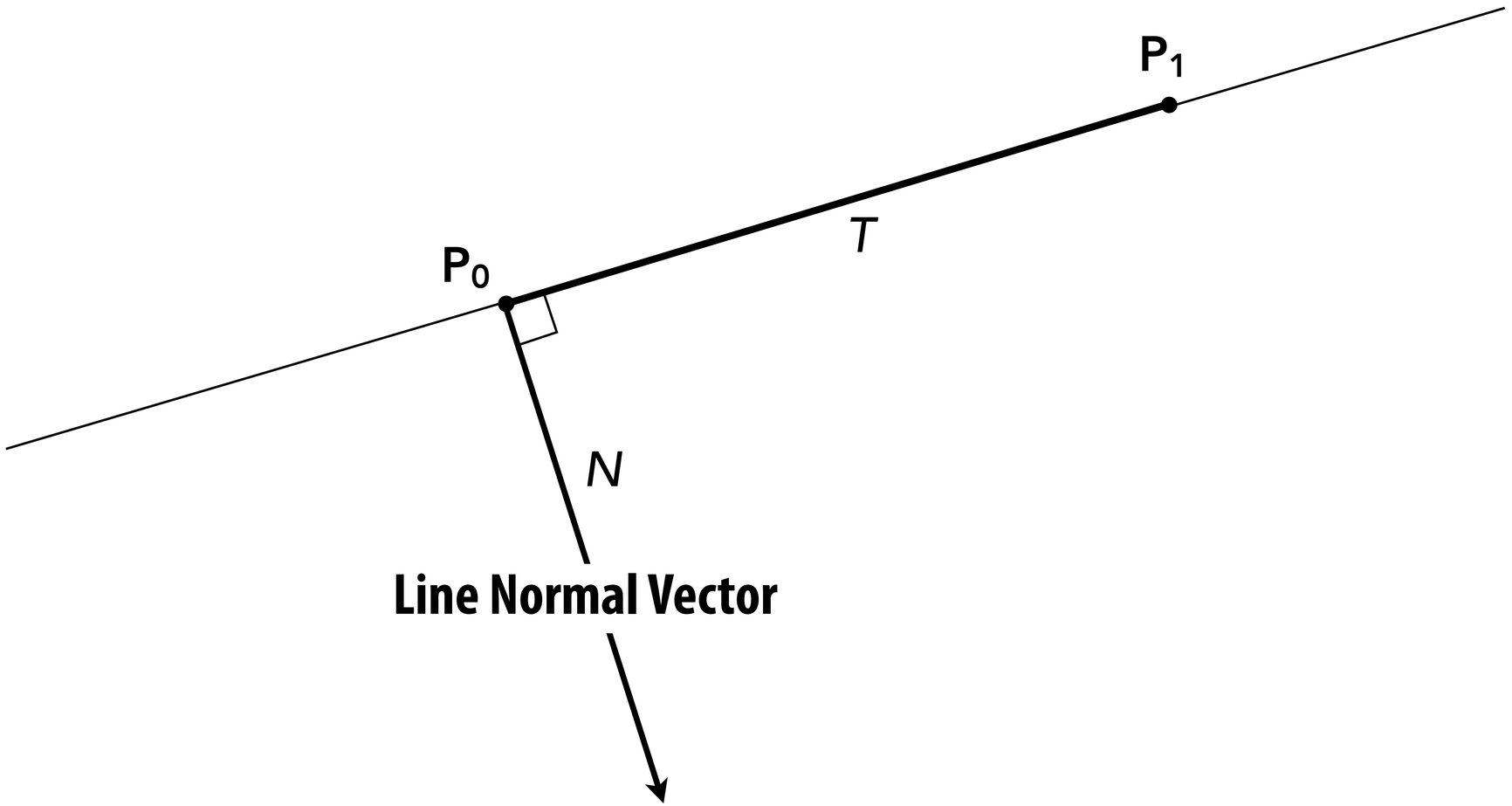
Line equation derivation



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

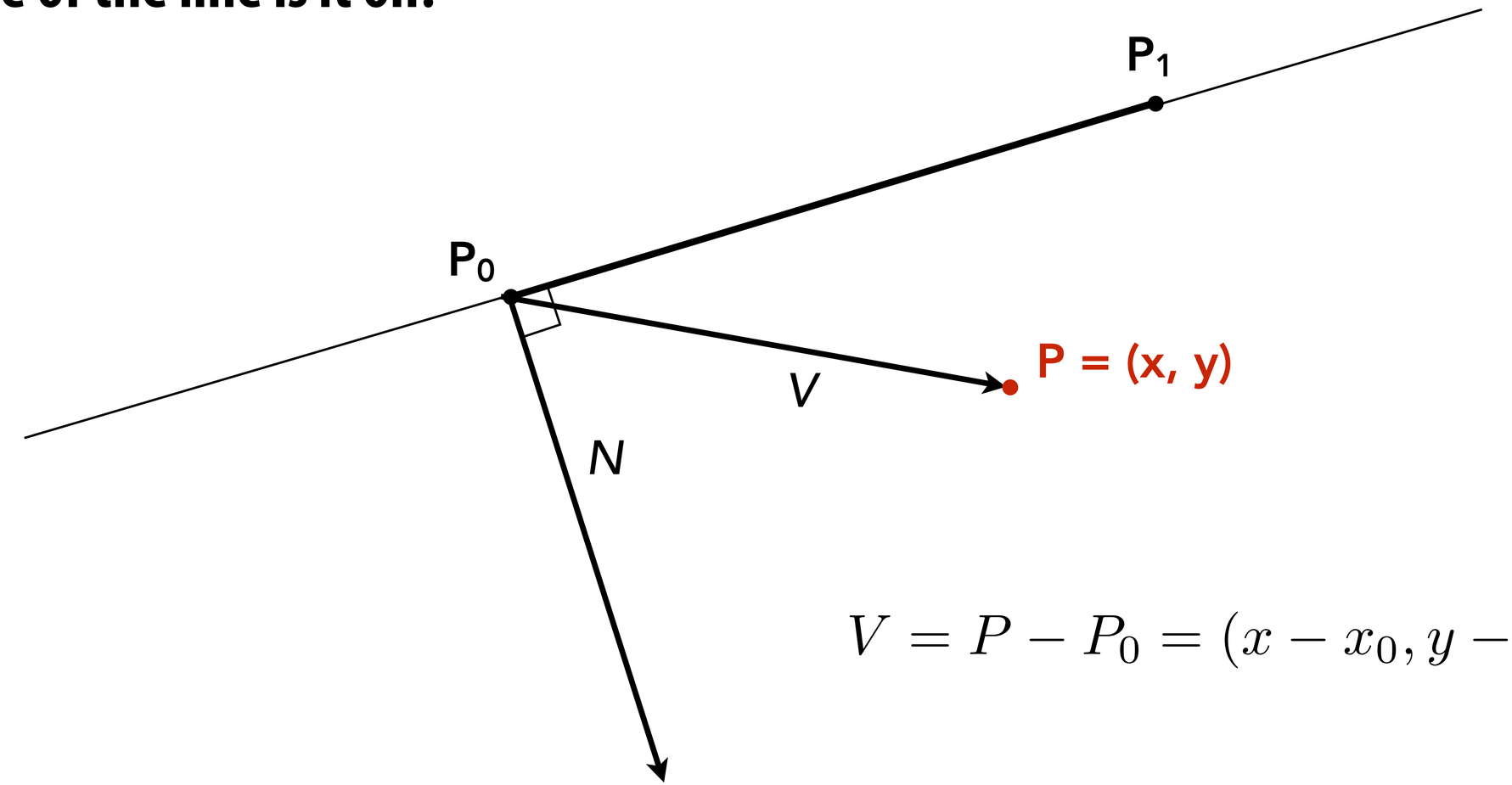
Line equation derivation

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



Line equation derivation

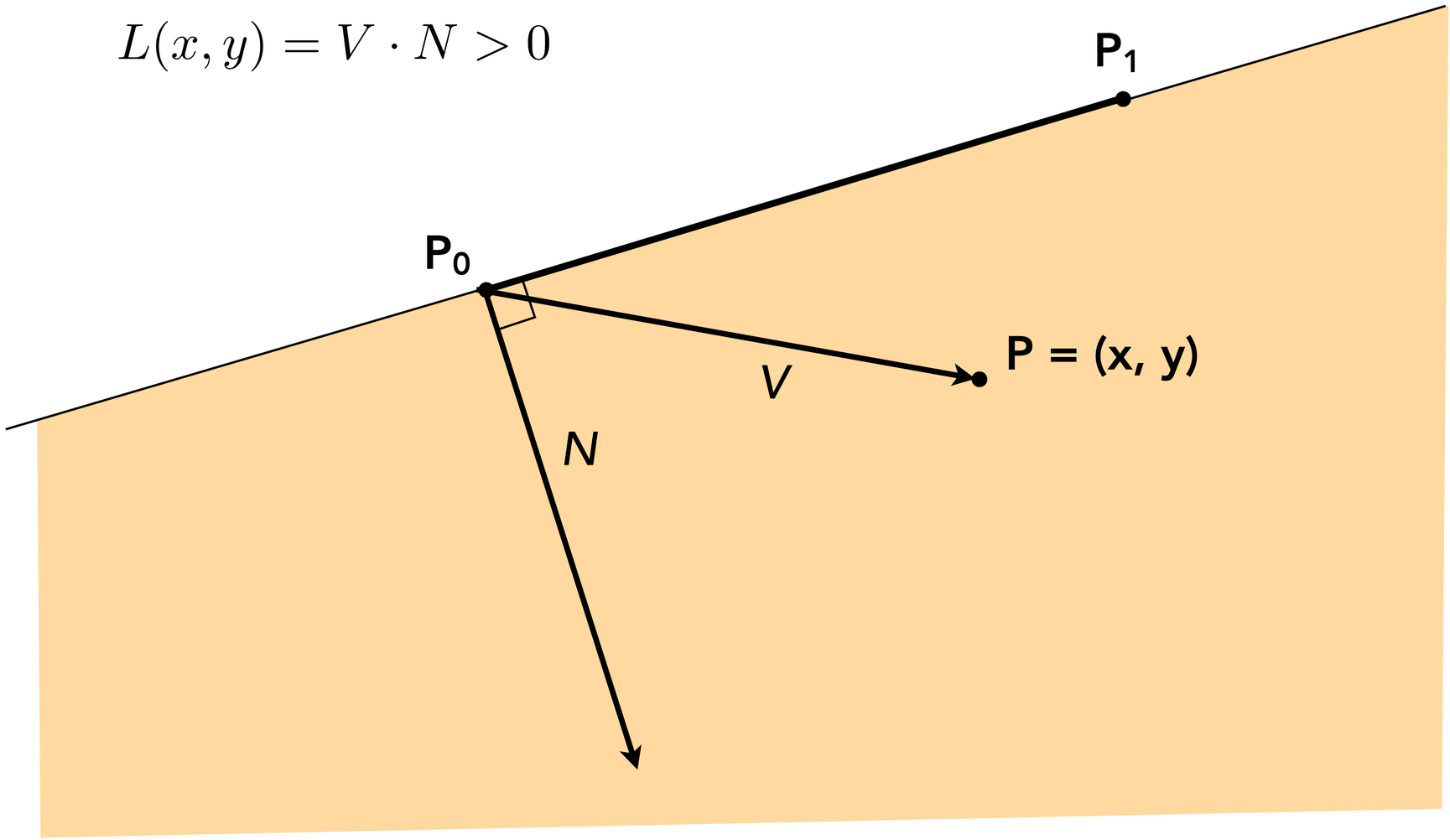
Now consider a point *P*=(x,y).
Which side of the line is it on?



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

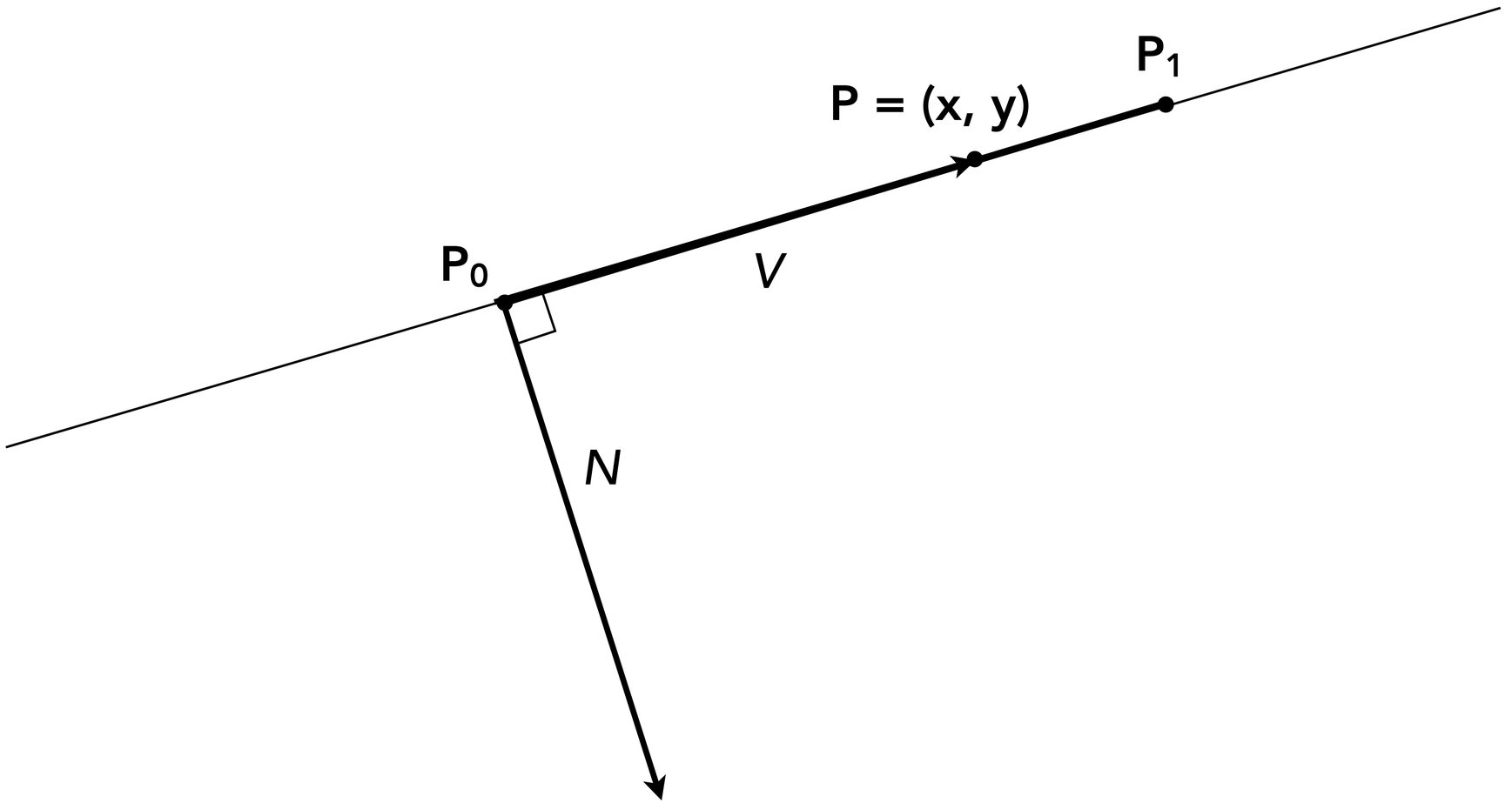
Line equation tests

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

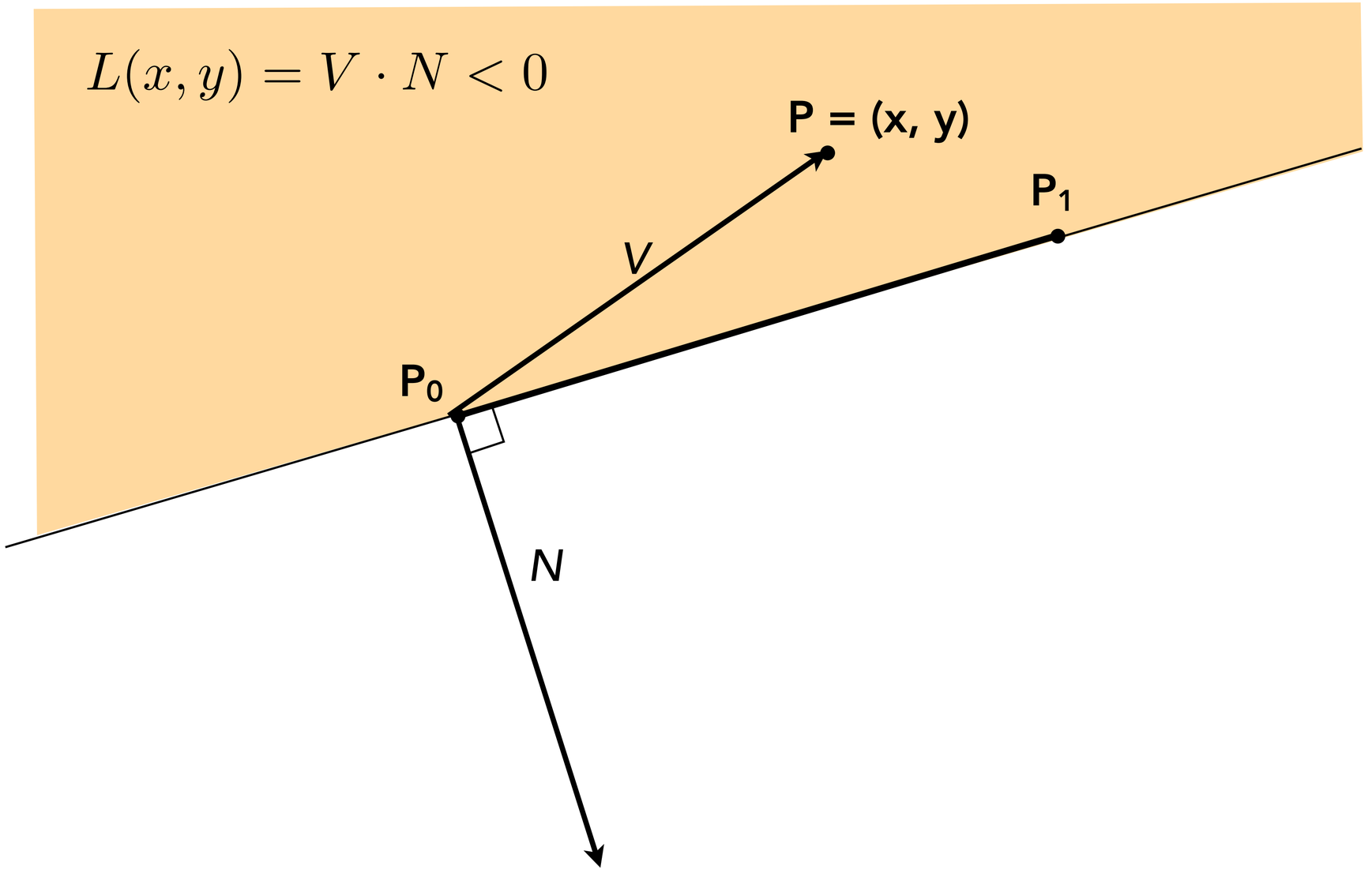


Line equation tests

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

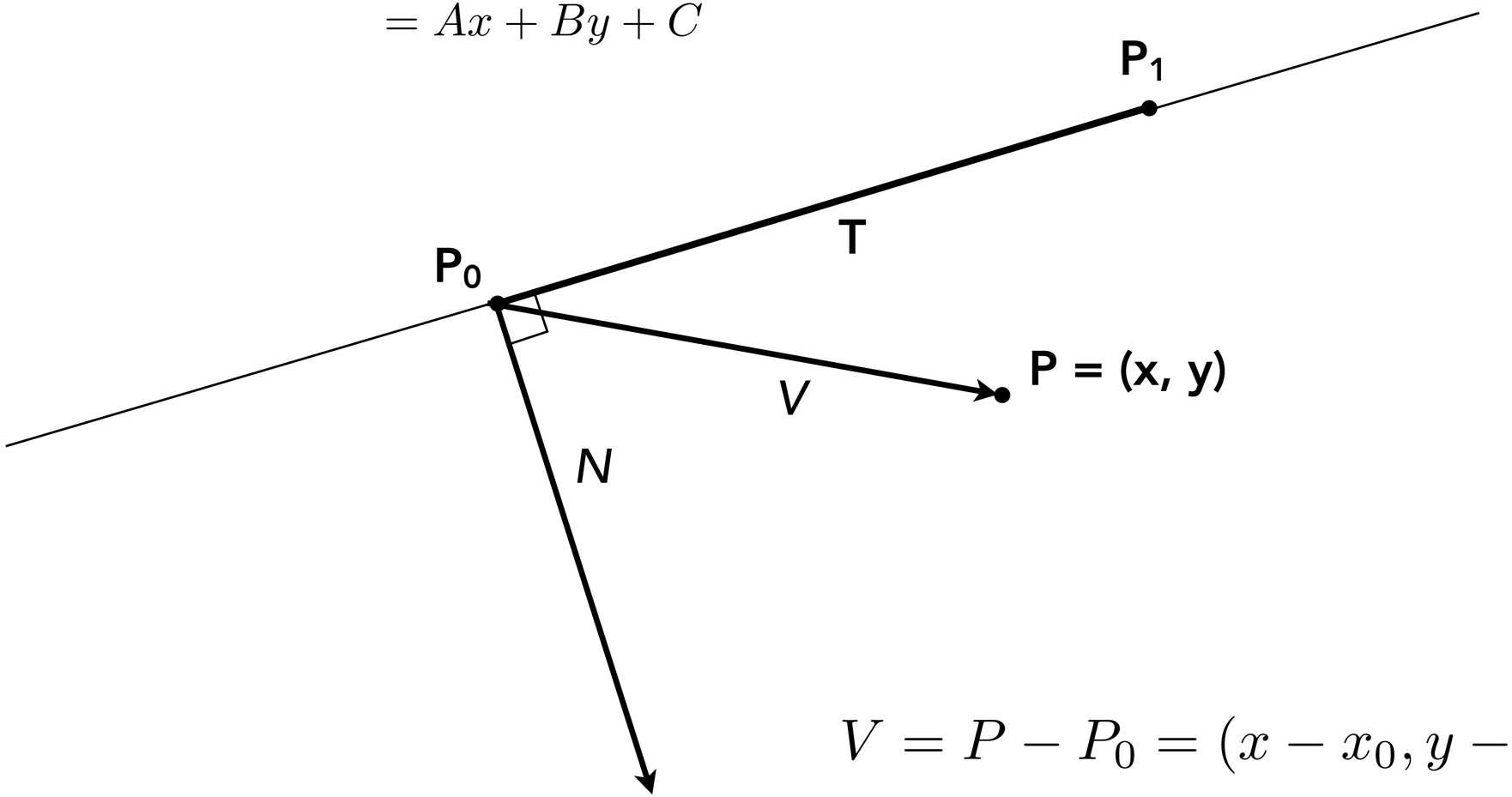


Line equation tests



Line equation derivation

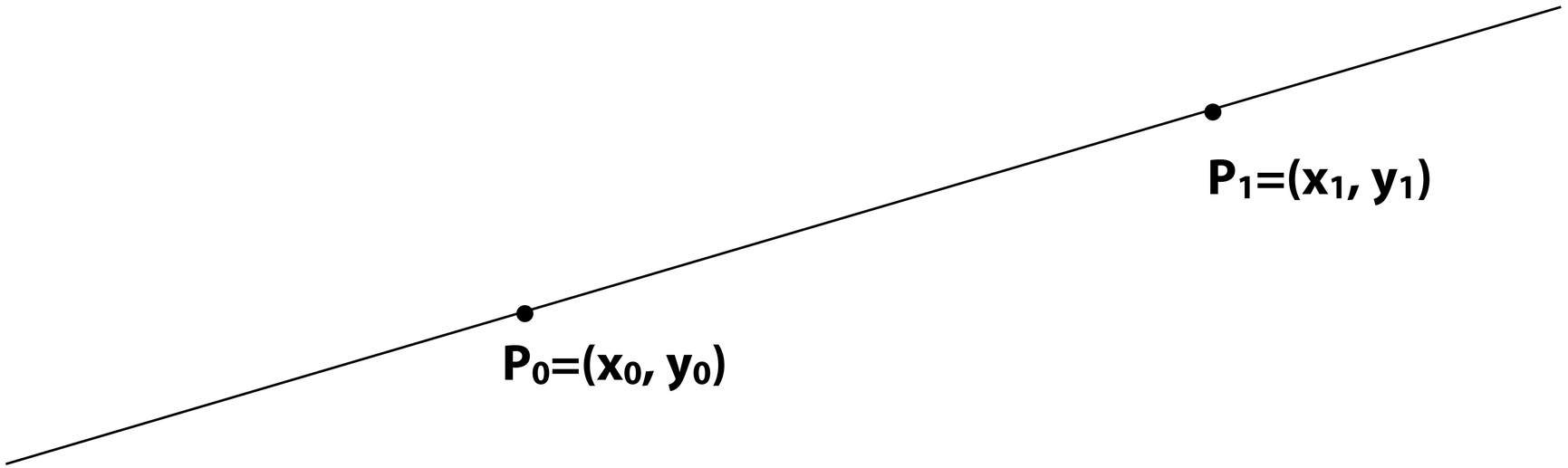
$$\begin{aligned} L(x,y) = V \cdot N &= -(y-y_0)(x_1-x_0) + (x-x_0)(y_1-y_0) \\ &= (y_1-y_0)x - (x_1-x_0)y + y_0(x_1-x_0) - x_0(y_1-y_0) \\ &= Ax + By + C \end{aligned}$$



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

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

New task:
Draw a 2D line onto an image



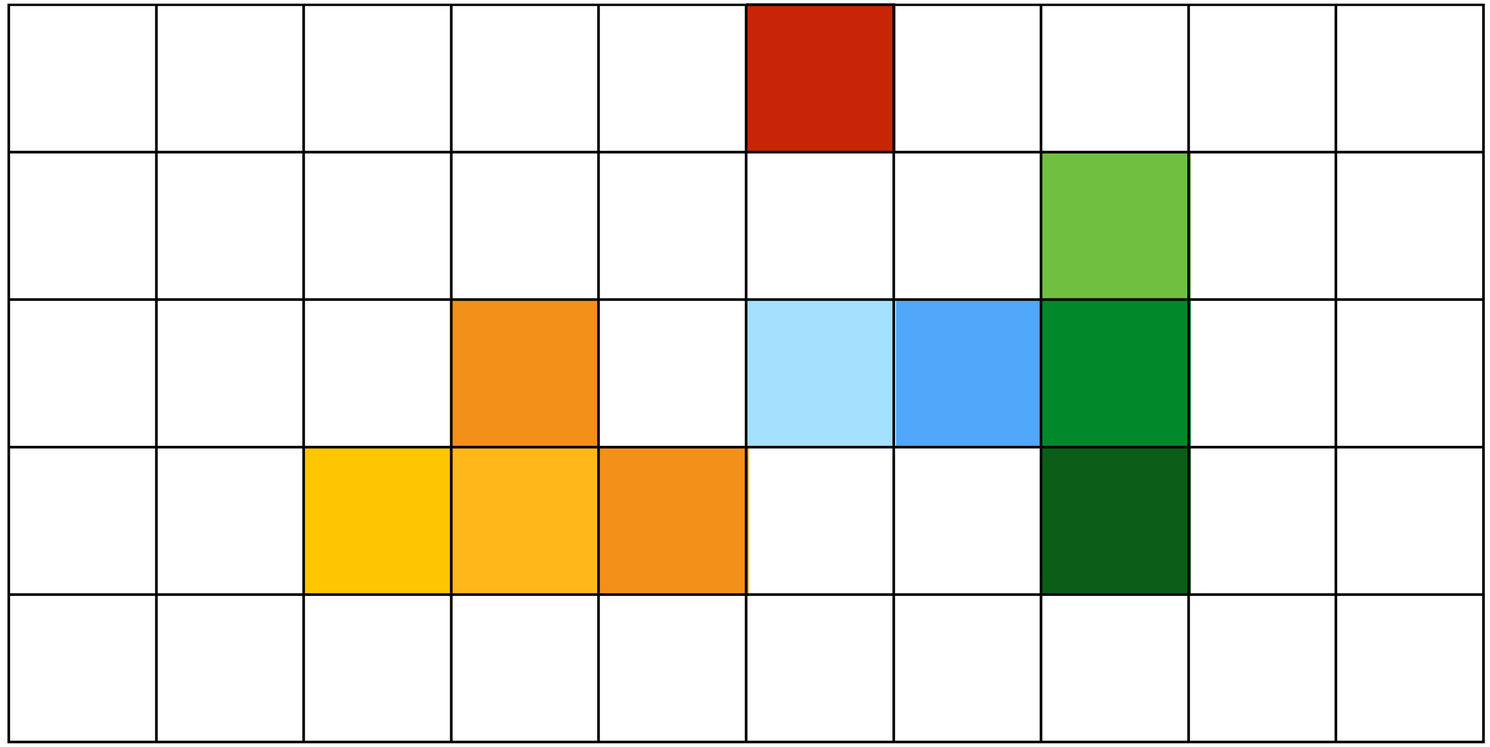
But wait…
How do we represent the image we want to draw on screen?

Frame buffer: memory for a raster display

image = “2D array of colors”

Output for a raster display

- Common abstraction of a raster display:
  - Image represented as a 2D grid of "pixels" (picture elements) **
  - Each pixel can can take on a unique color value



** We will strongly challenge this notion of a pixel "as a little square" next class. But let's go with it for now. ;-)

Josephine the graphics cat

Josephine the graphics cat

4K TV display

UHD TV resolution: (“4K”)
3840 x 2160 pixels (8.3 megapixels)

HDTV resolution:
1920 x 1080 (2.1 megapixels)

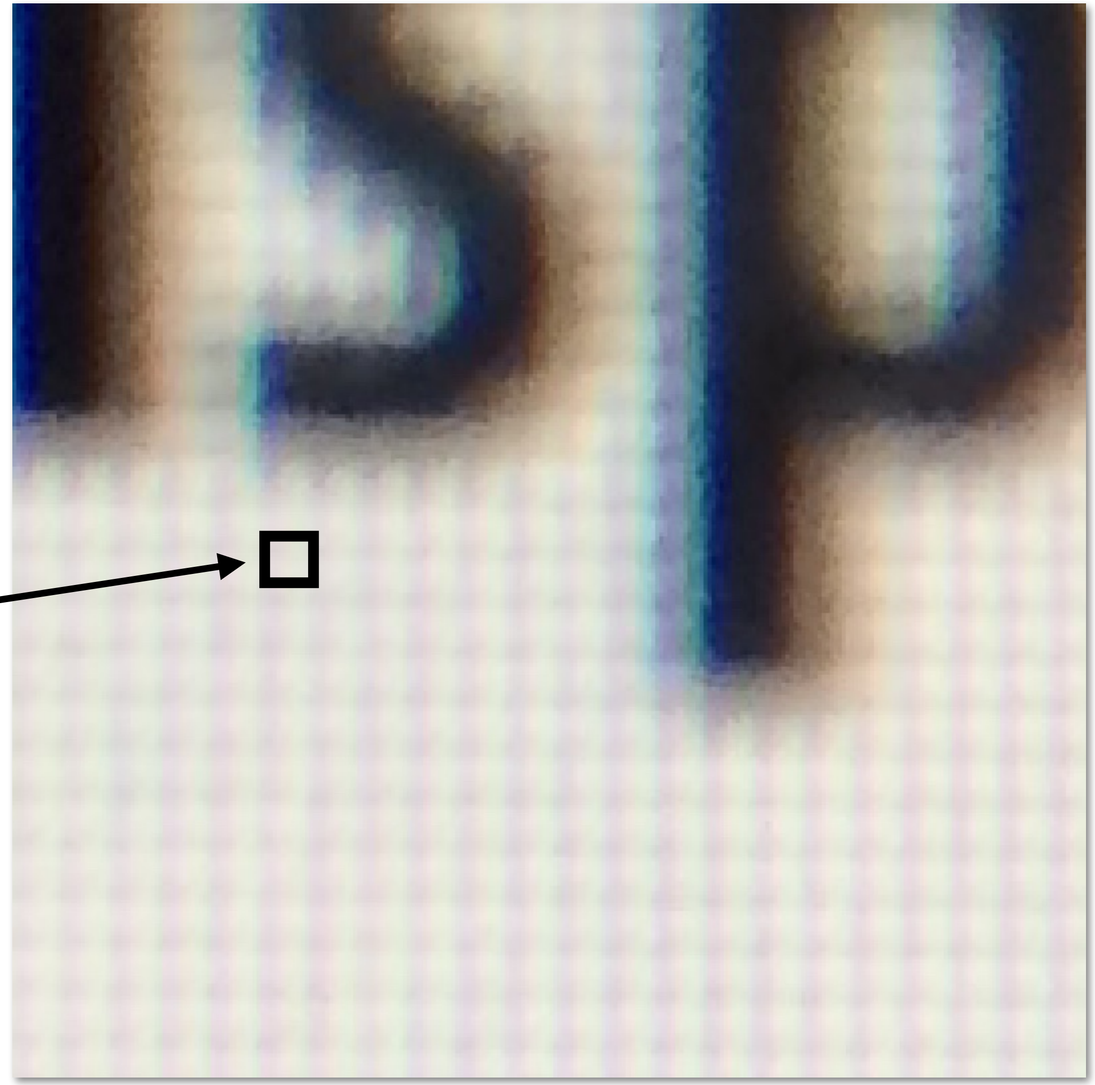
Photo credit: Mike Mozart (via Flickr)

iPhone 12 display

2532 x 1170 pixels
(2.9 megapixels)

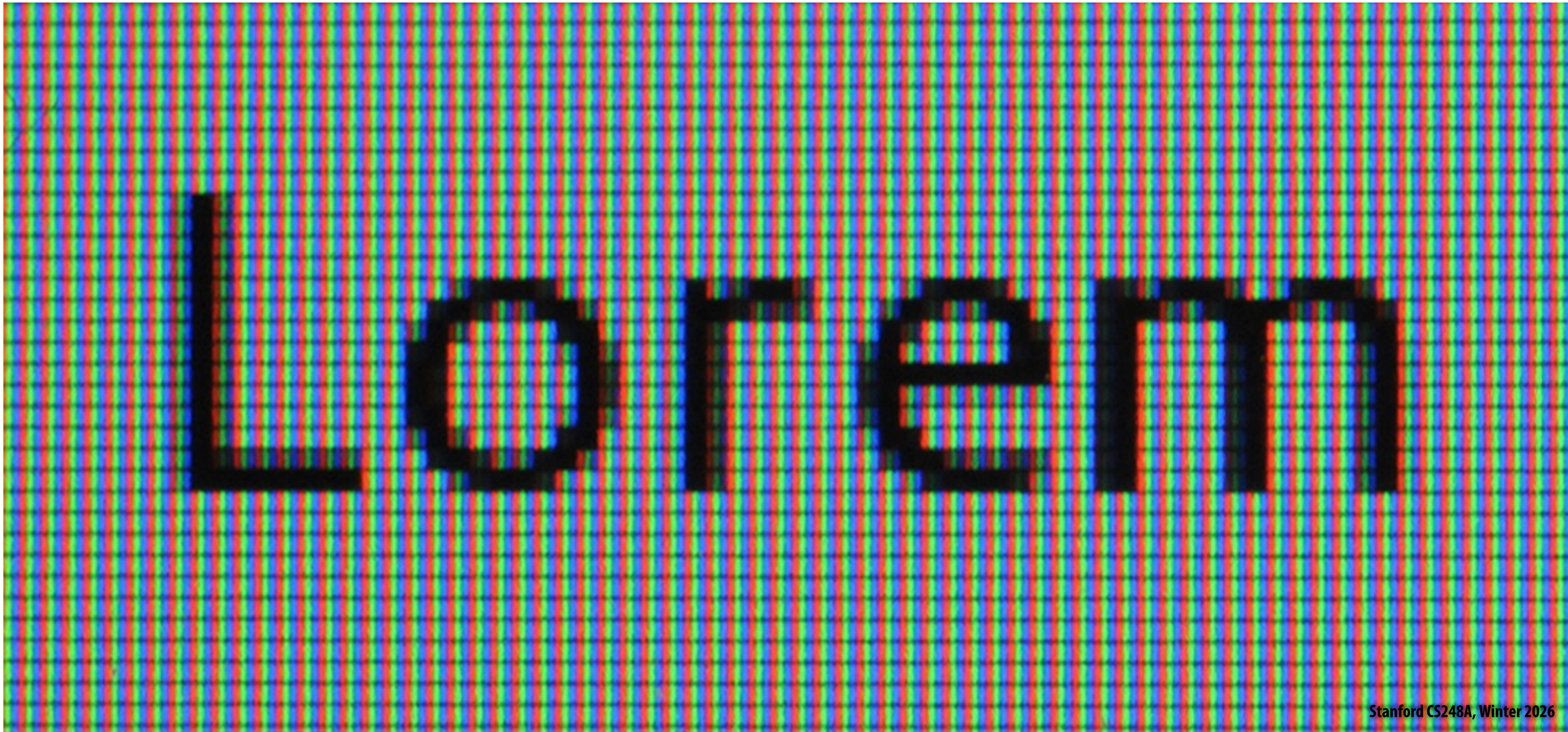
About 460 pixels per inch

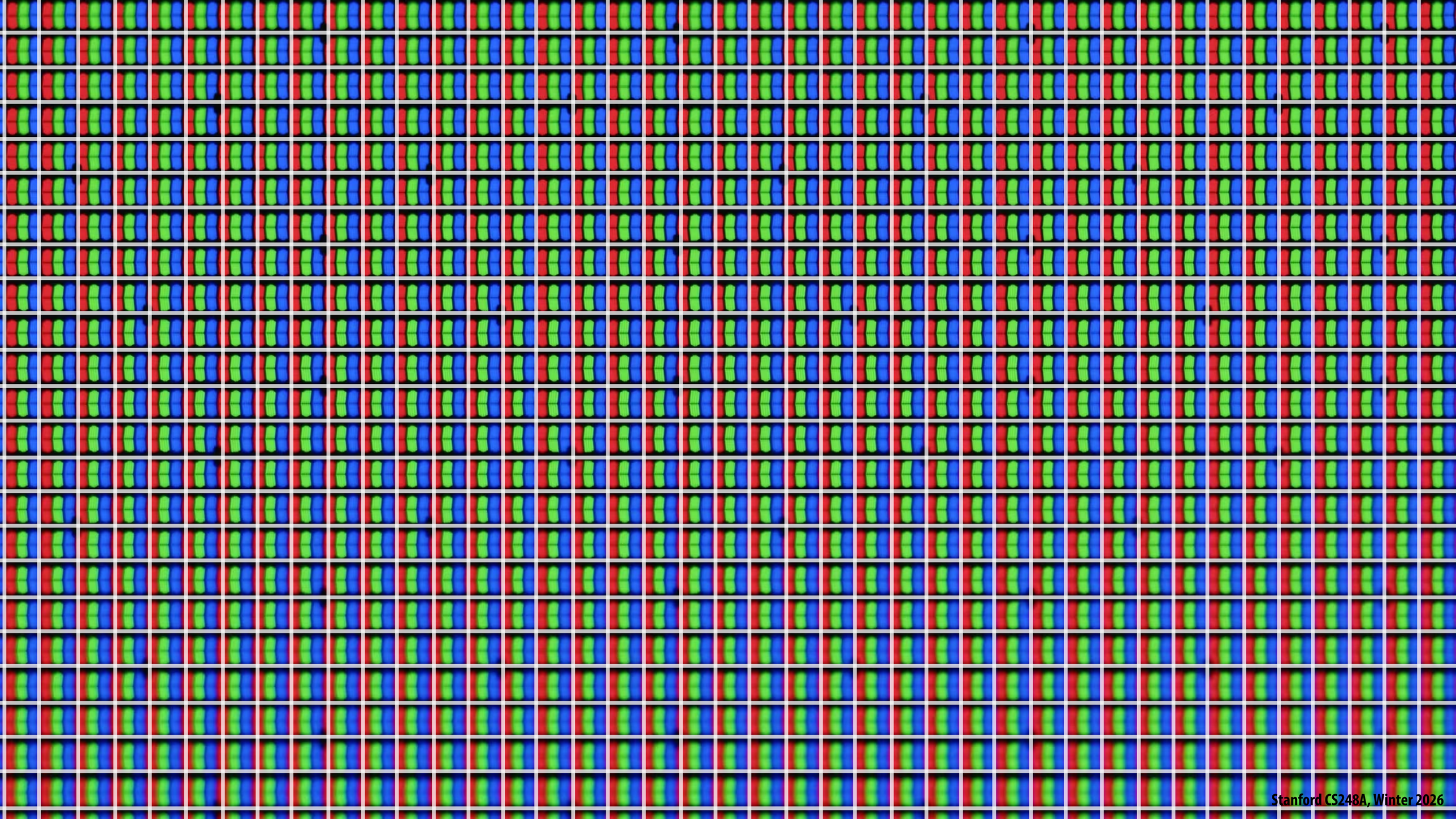
A raster display converts an image (a color value at each pixel) into emitted light



Display pixel on my laptop
(close up photo)

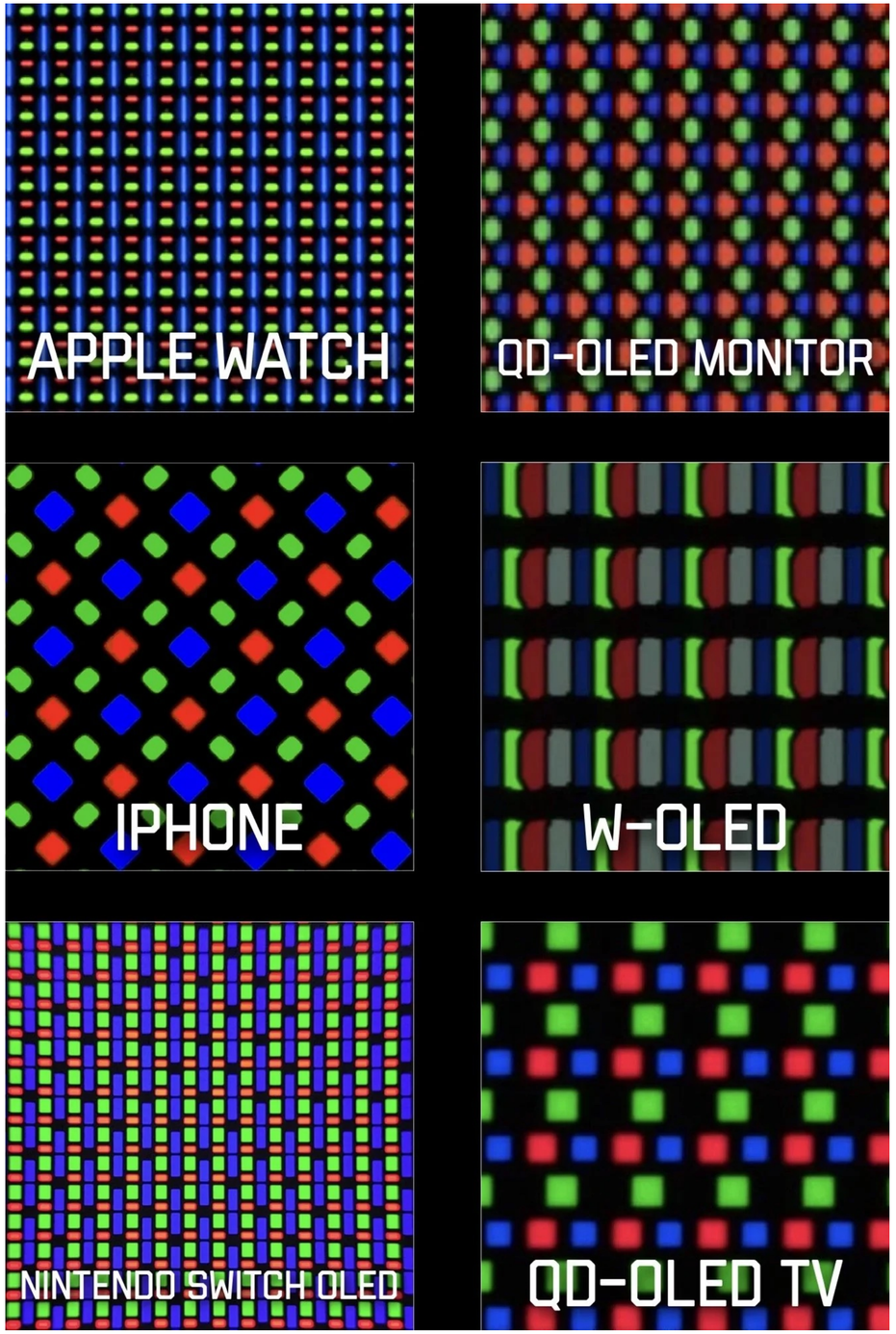
Close up photo of pixels on a modern display





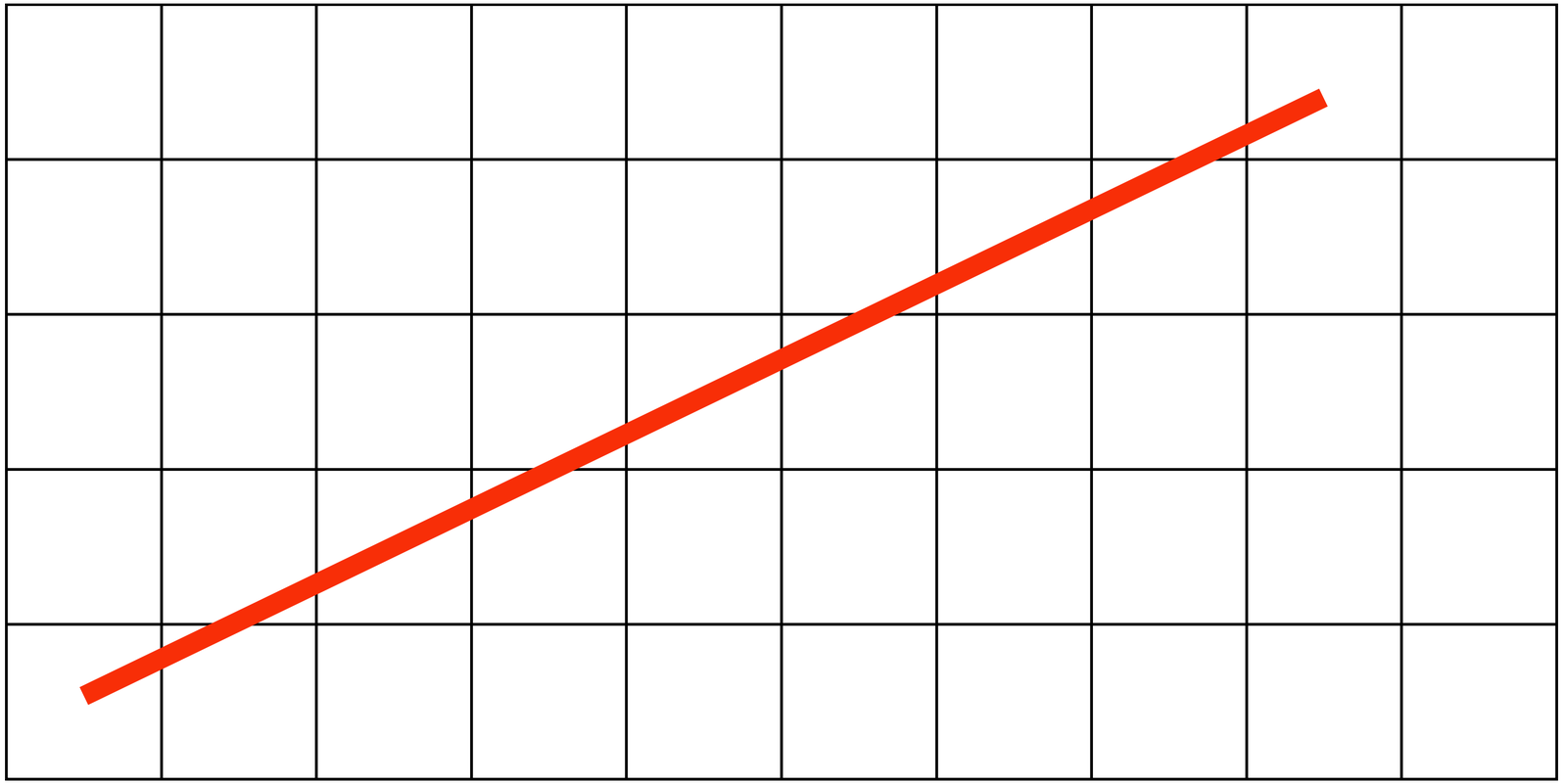
Aside: other sub pixel layouts

- So what is a pixel, anyway?
- (More on this in the next lecture)



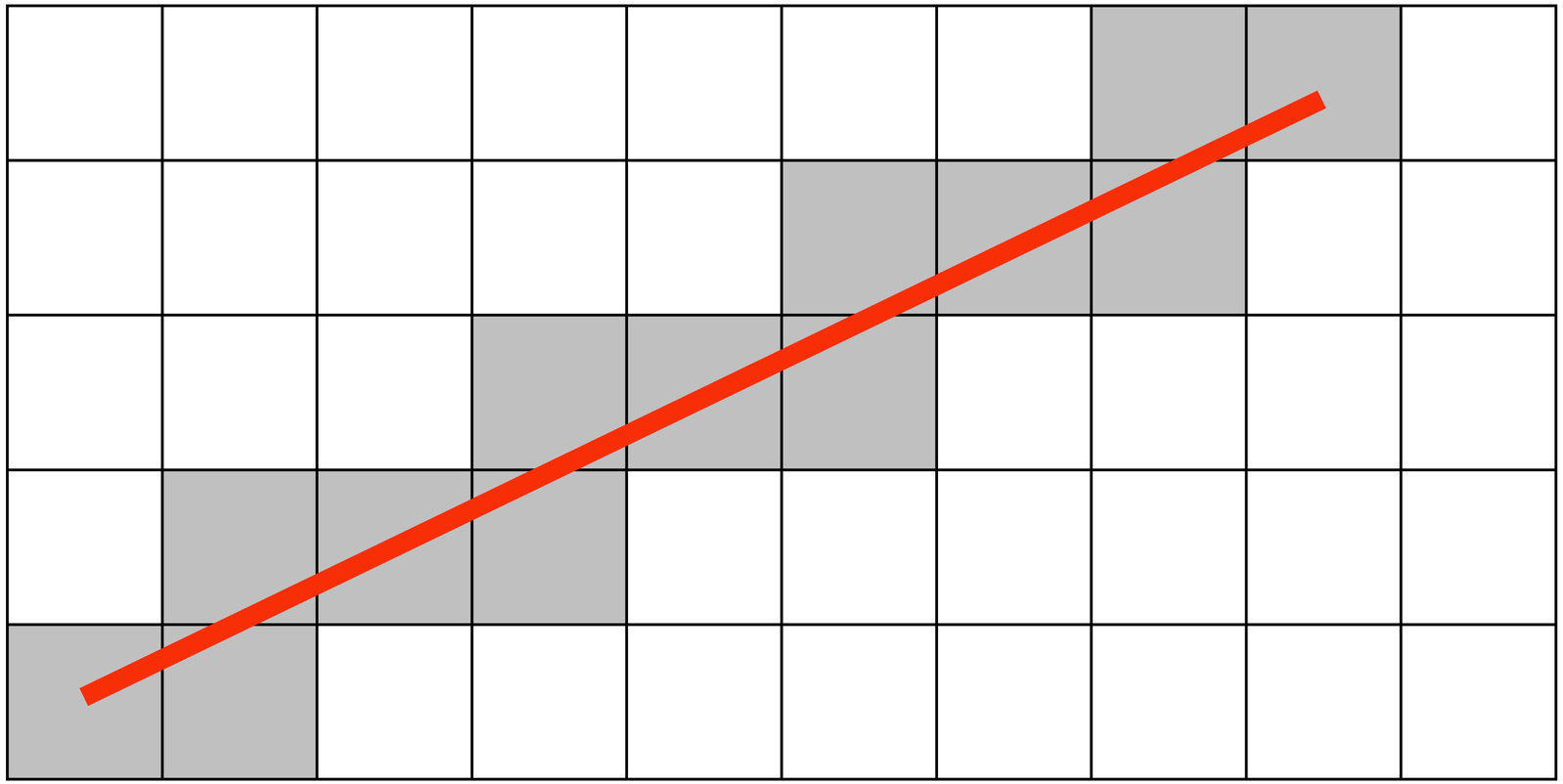
What pixels should we color in to depict a line?

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



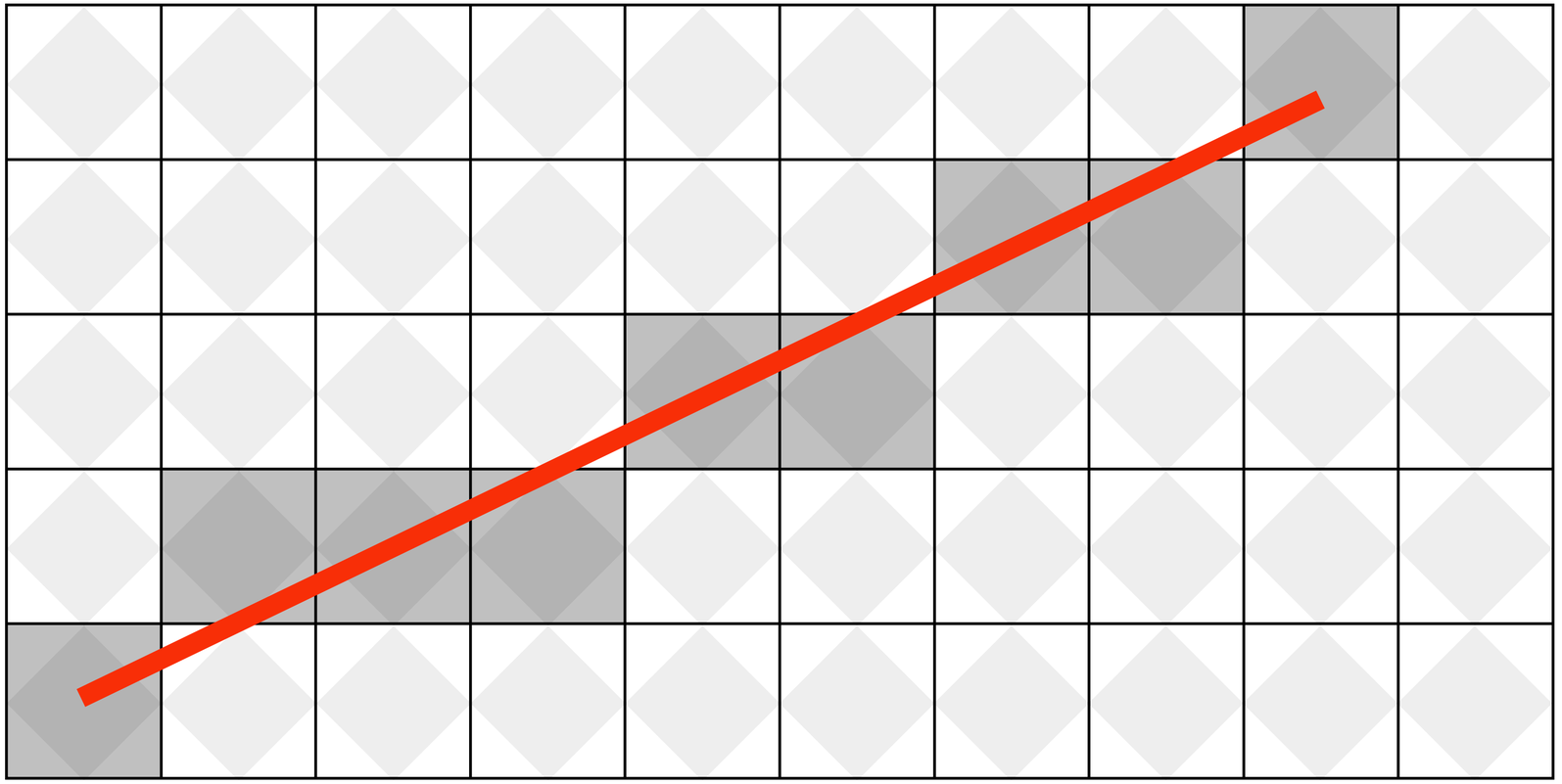
What pixels should we color in to depict a line?

Light up all pixels intersected by the line?



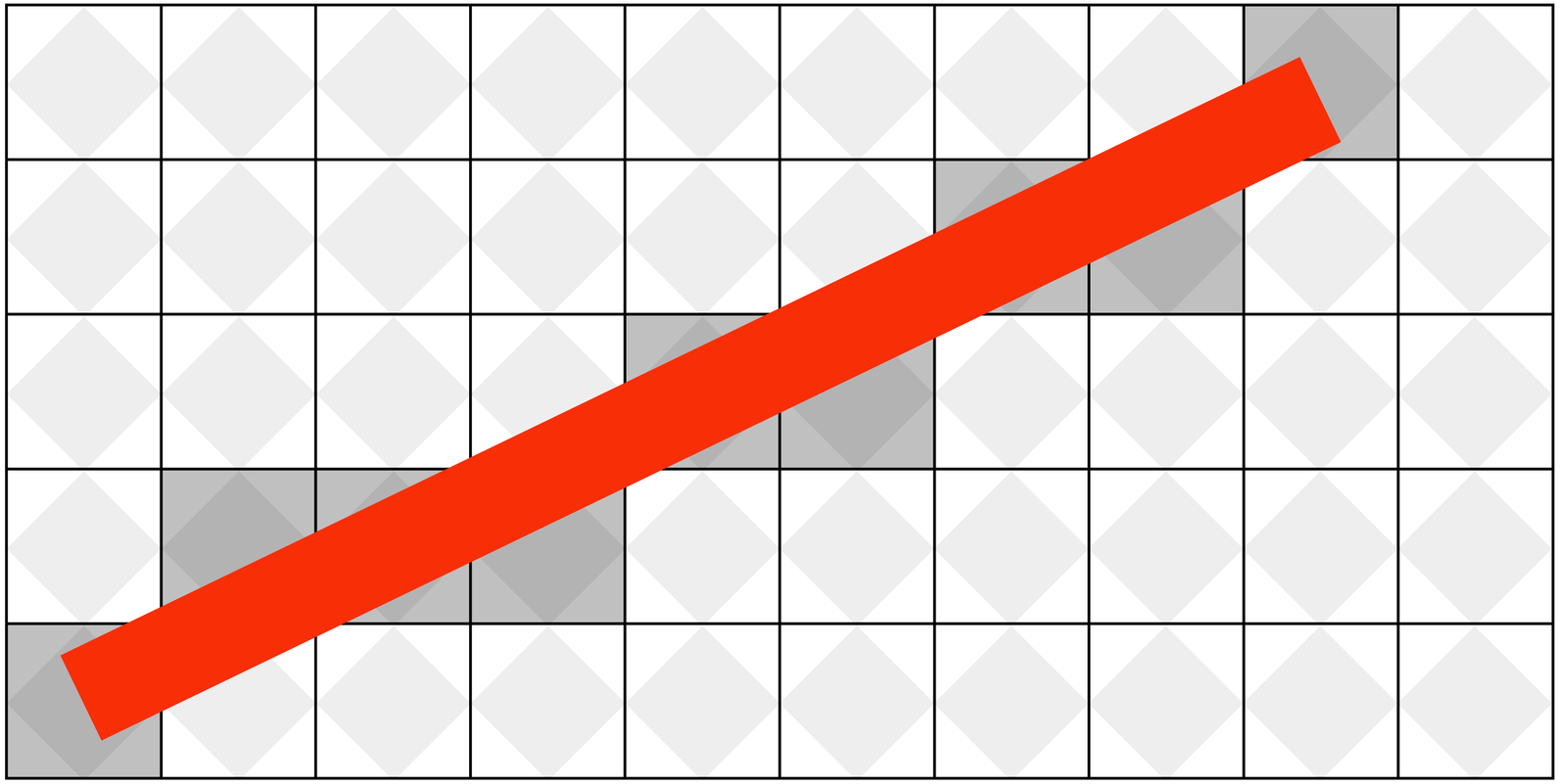
What pixels should we color in to depict a line?

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



What pixels should we color in to depict a line?

Is there a right answer?
(consider a drawing a "line" with thickness)



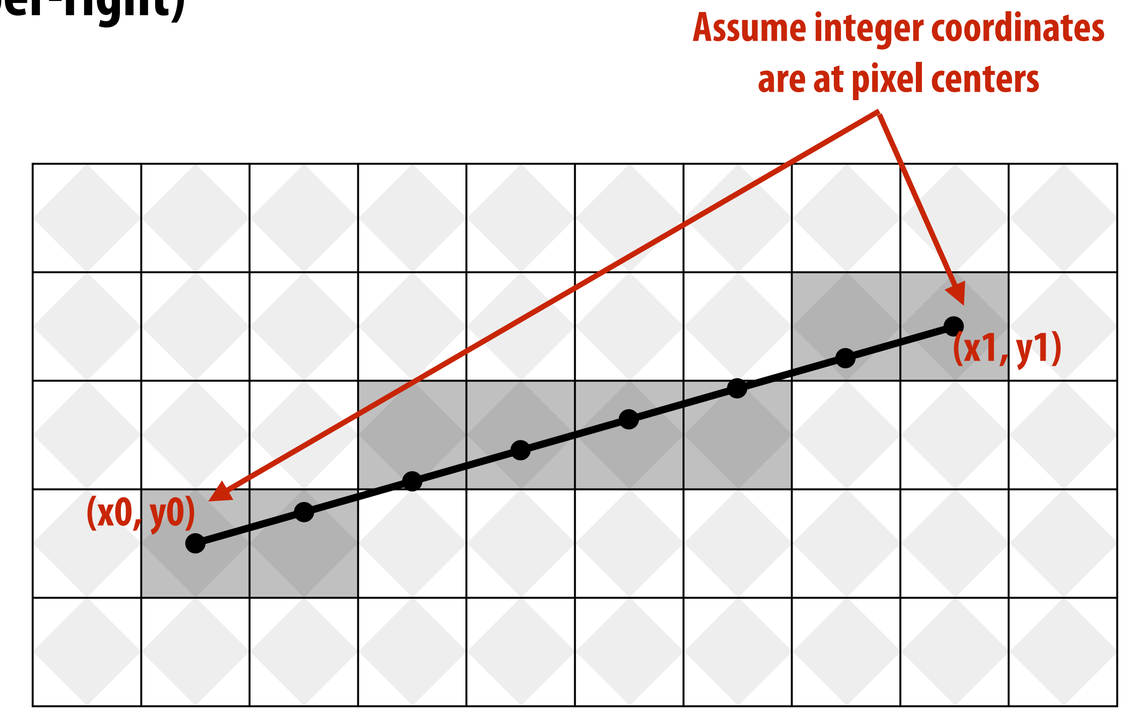
How do we find the pixels satisfying a chosen rasterization rule?

- Could check every single pixel in the image to see if it meets the condition...
  - $O(n^2)$ pixels in image vs. at most $O(n)$ "lit up" pixels
  - *Must* be able to do better! (e.g., algorithm that does work proportional to number of pixels painted when drawing the line)

Incremental line rasterization

- Let's say a line is represented with integer endpoints: (x0,y0), (x1,y1)
- Slope of line: m = (y1-y0) / (x1-x0)
- Consider an easy special case:
  - x0 < x1, y0 < y1 (line points toward upper-right)
  - 0 < m < 1 (more change in x than y)

```
y = y0;
for( x=x0; x<=x1; u++ )
{
    y += m;
    draw( x, round(y) )
}
```

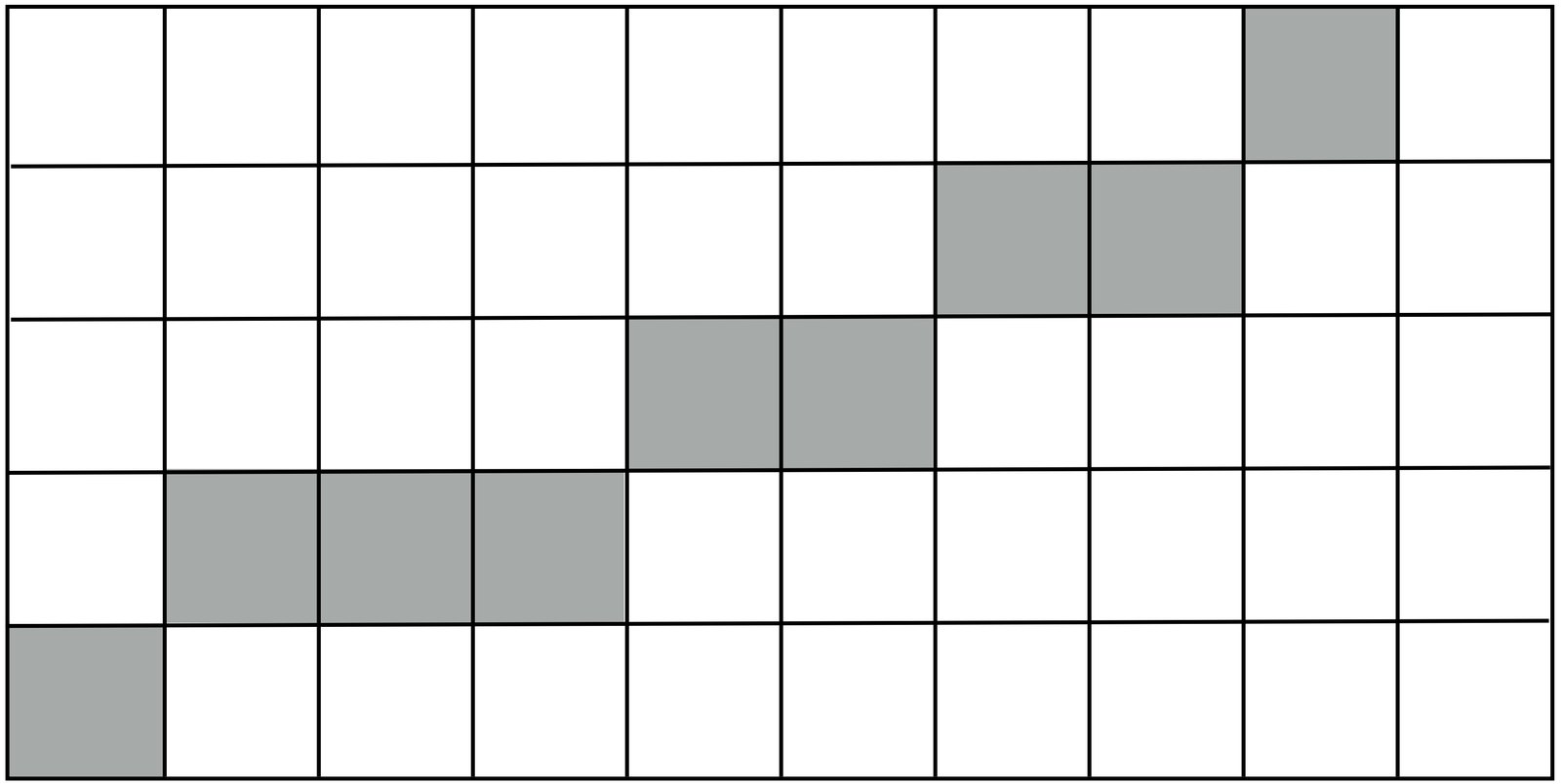


Common optimization: rewrite algorithm to use only integer arithmetic (Bresenham algorithm)

“Rasterized image” representation of a line

What is it a convenient representation for?

Is it a convenient representation for some of our prior tasks?



Recap: what have we done so far

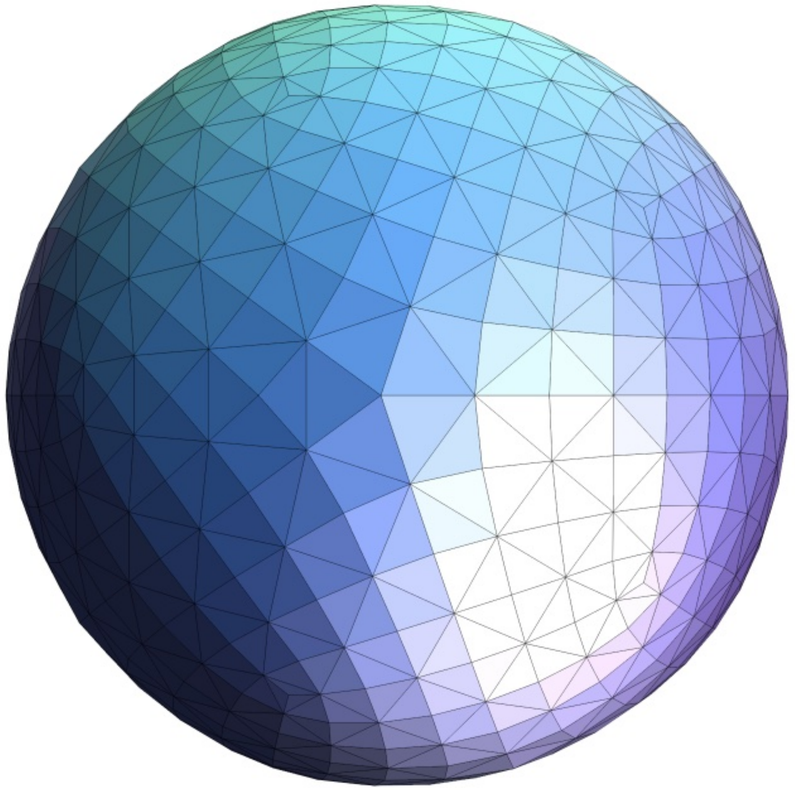
- We've explored a number of different spatial reason tasks involving lines
  - Enumeration of points on line, testing whether a point was on a line, drawing a line

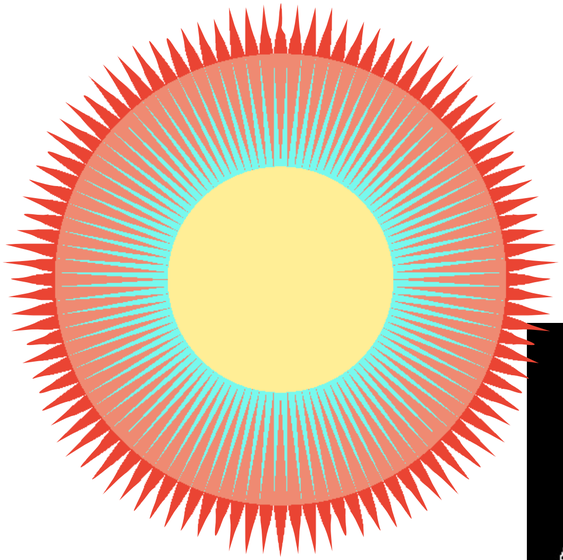
- We've seen how the task dictates which representations are "good"

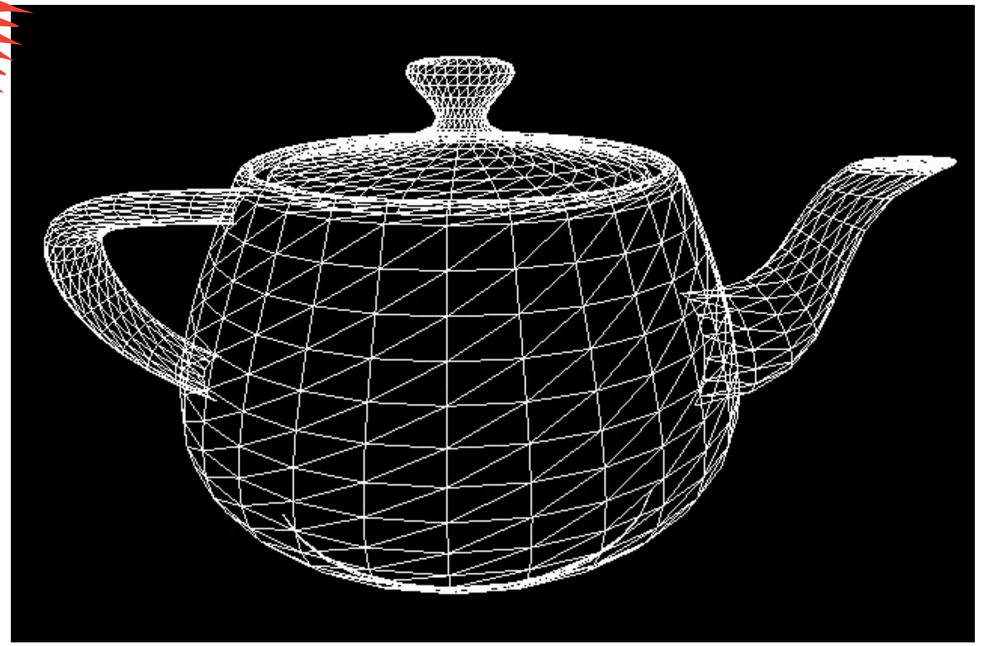
Let's move up in complexity:
What about drawing triangles?

Why triangles?

Triangles are a key primitive for creating more complex shapes and surfaces



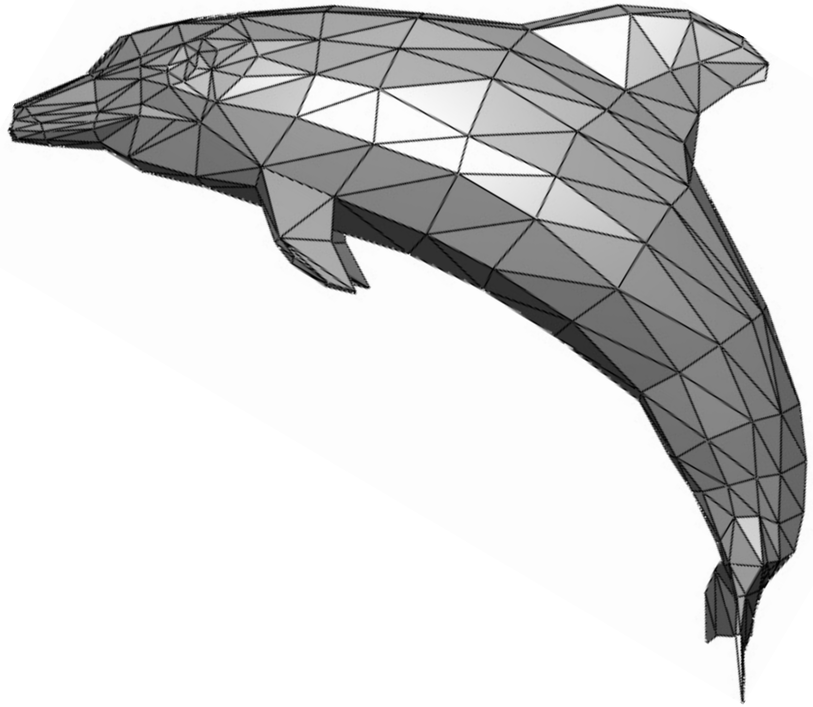




Why triangles?

- Most basic polygon
  - Can break up other polygons into triangles
  - Allows programs to optimize one implementation

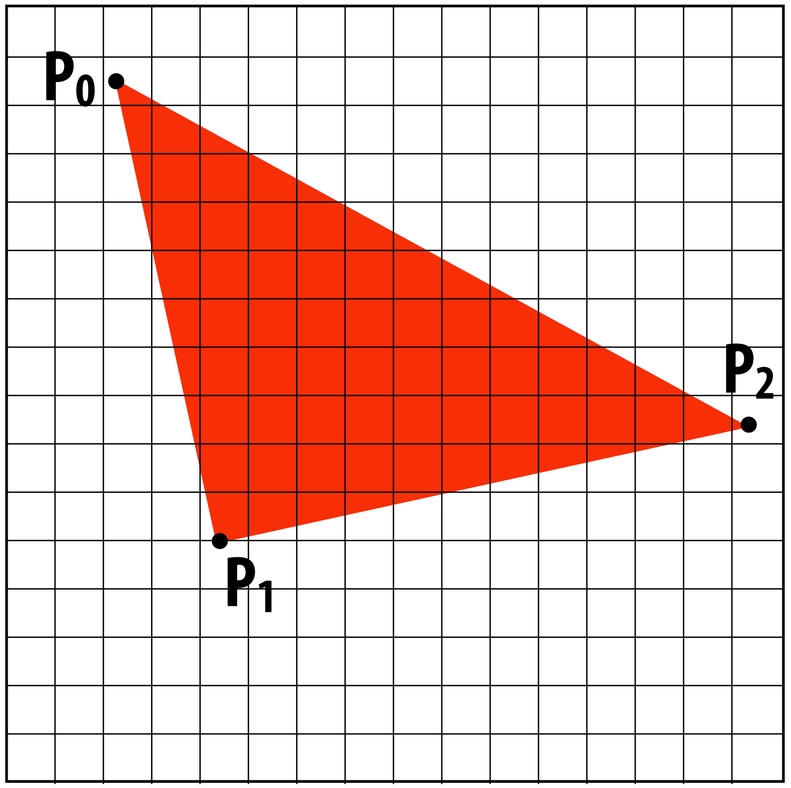
- Triangles have unique properties
  - Guaranteed to be planar
  - Well-defined interior
  - Well-defined method for interpolating values at vertices over triangle (a topic of a future lecture)



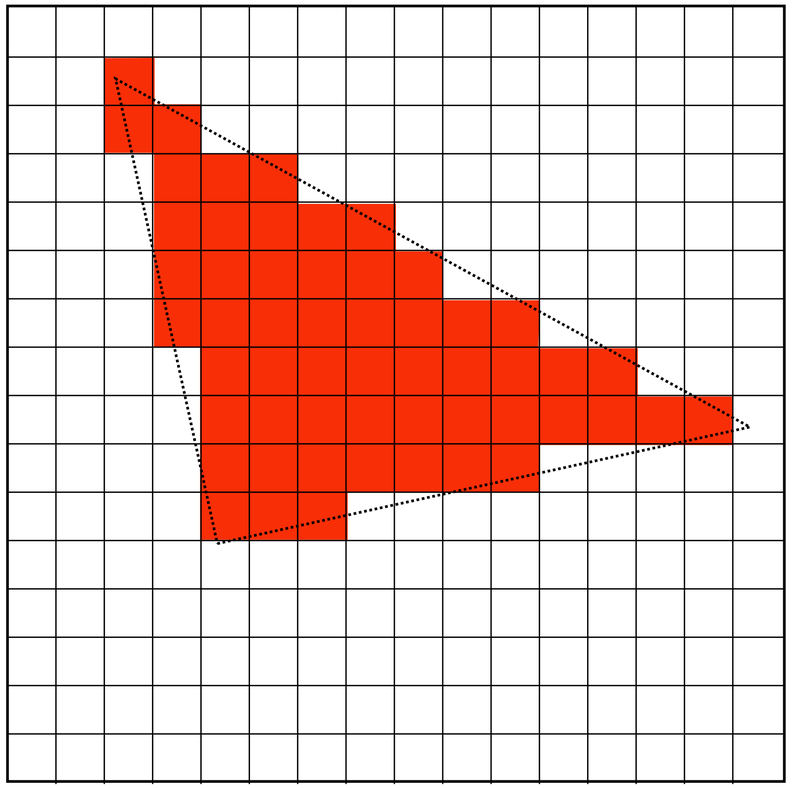
Drawing a 2D triangle ("triangle rasterization")

(Converting a 2D representation of a triangle into an image)

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

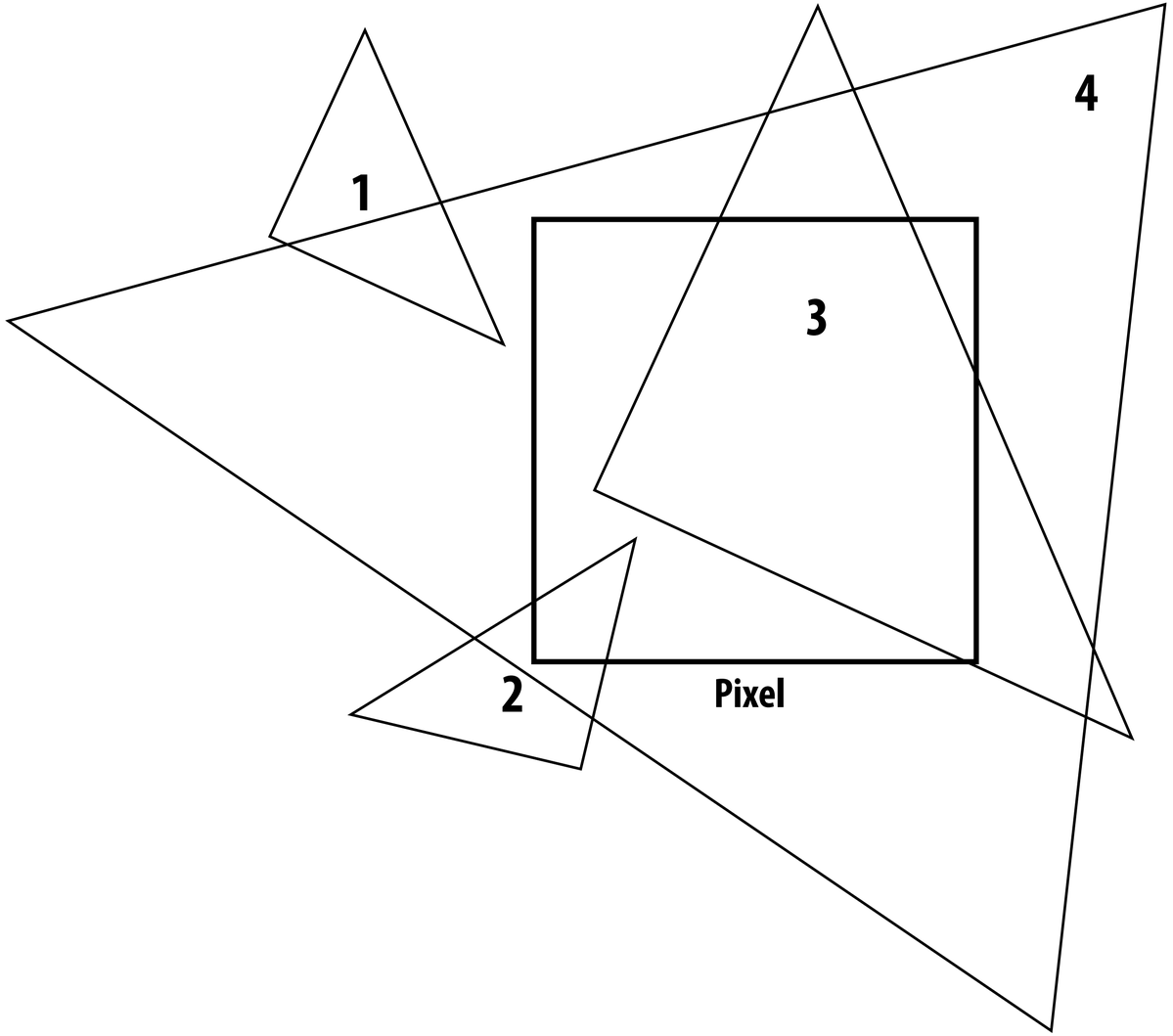


Output:
Set of pixels "covered" by the triangle

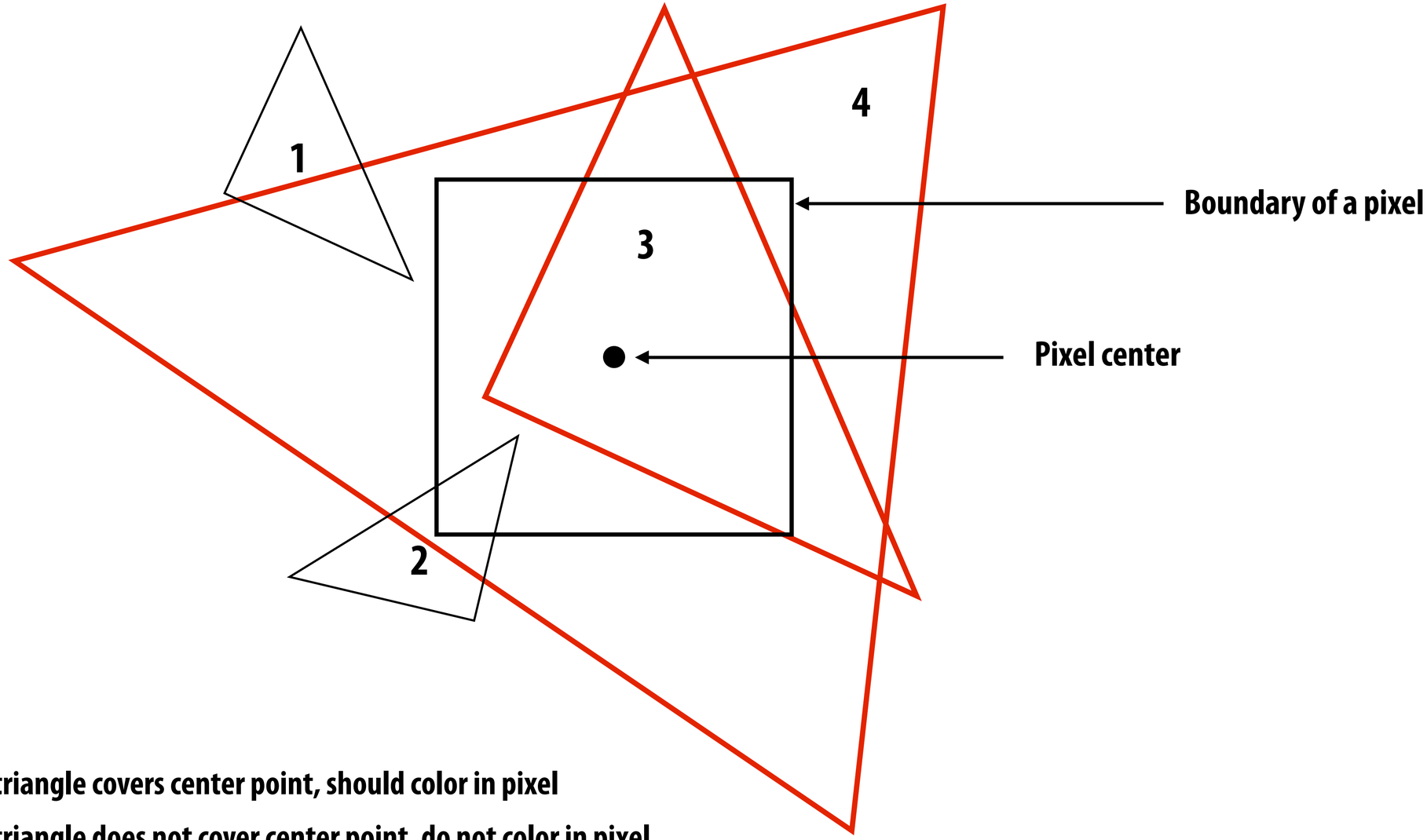


What does it mean for a pixel to be covered by a triangle?

Question: which triangles "cover" this pixel?



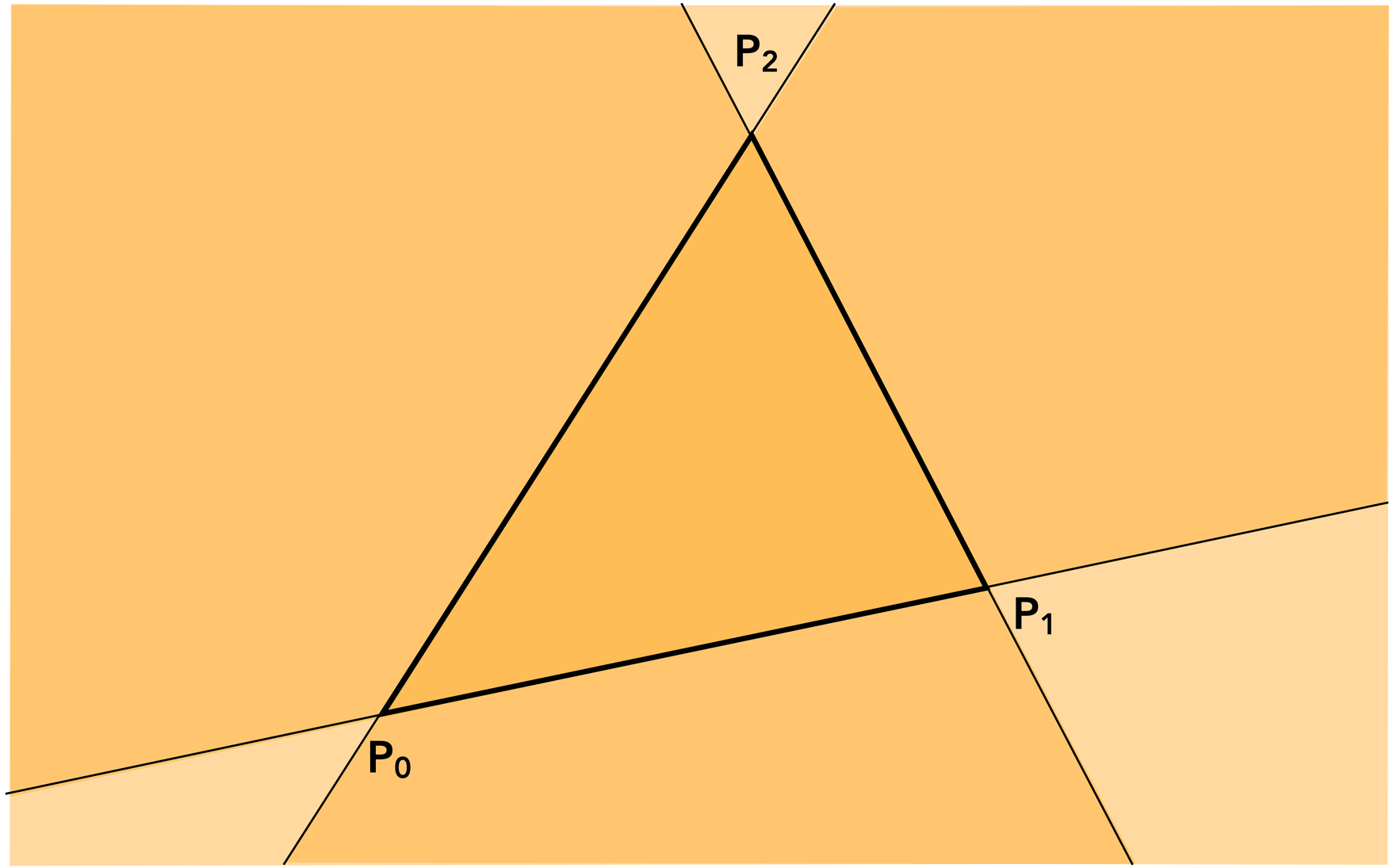
Idea: let's call a pixel "inside" the triangle if the pixel center is inside the triangle



= triangle covers center point, should color in pixel

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

Triangle = intersection of three half planes



Point-in-triangle test: via three point-edge tests

Set up implicit line equation for each edge, evaluate whether point *P* is "inside" all three edges

Triangle vertex *i*

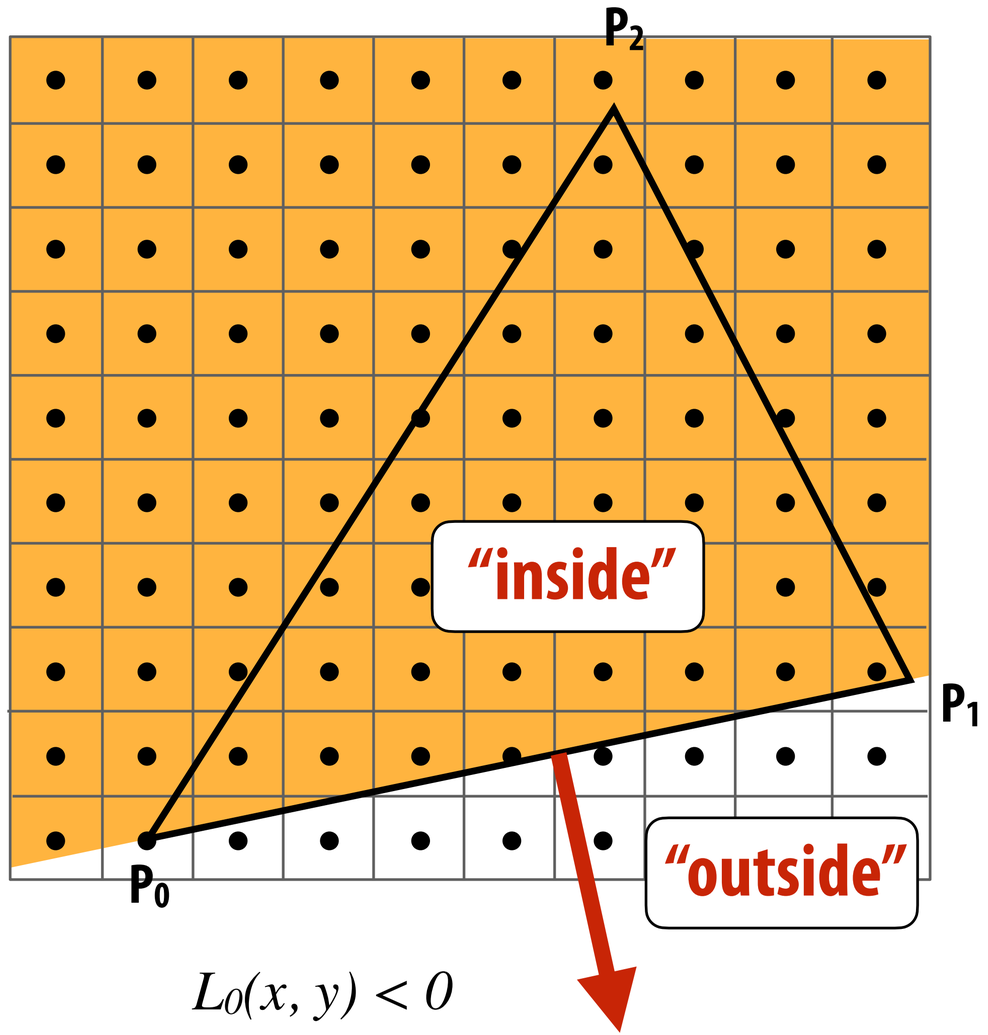
$$P_i = (X_i, Y_i)$$

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

Implicit form for edge *i*

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

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



Point-in-triangle test: via three point-edge tests

Set up implicit line equation for each edge, evaluate whether point *P* is "inside" all three edges

Triangle vertex *i*

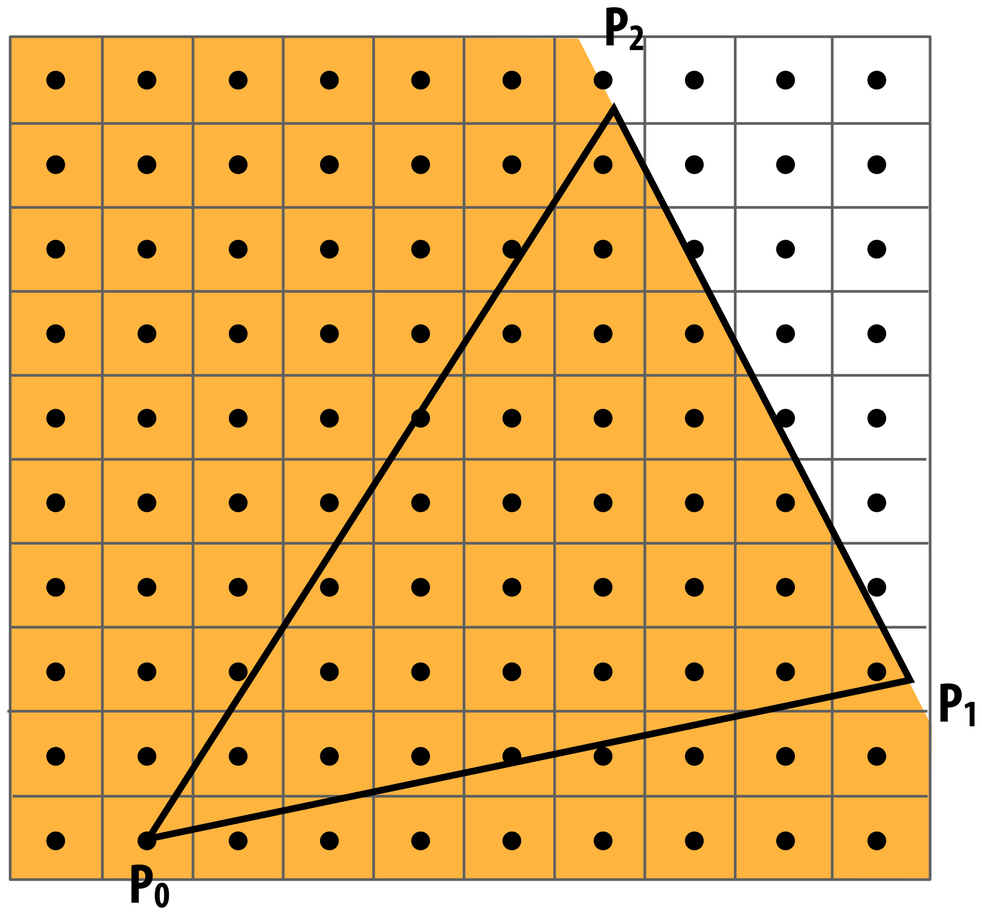
$$P_i = (X_i, Y_i)$$

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

Implicit form for edge *i*

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

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



$$L_1(x, y) < 0$$

Point-in-triangle test: via three point-edge tests

Set up implicit line equation for each edge, evaluate whether point *P* is "inside" all three edges

Triangle vertex *i*

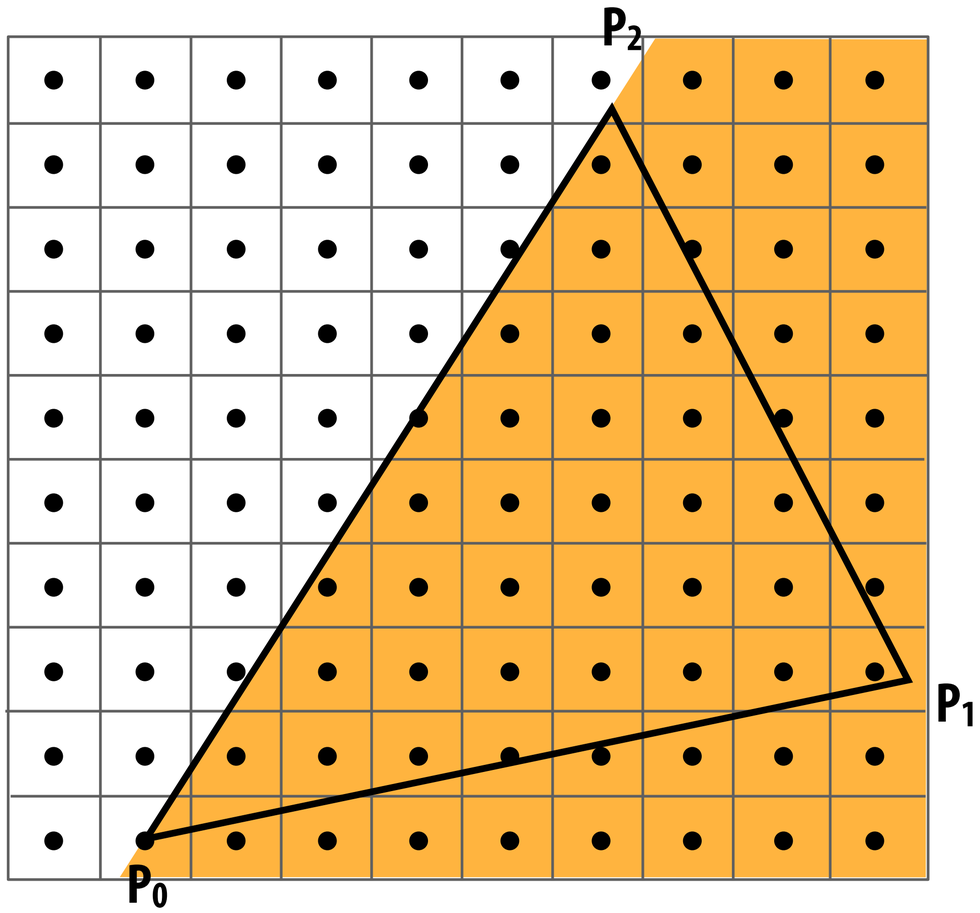
$$P_i = (X_i, Y_i)$$

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

Implicit form for edge *i*

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

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

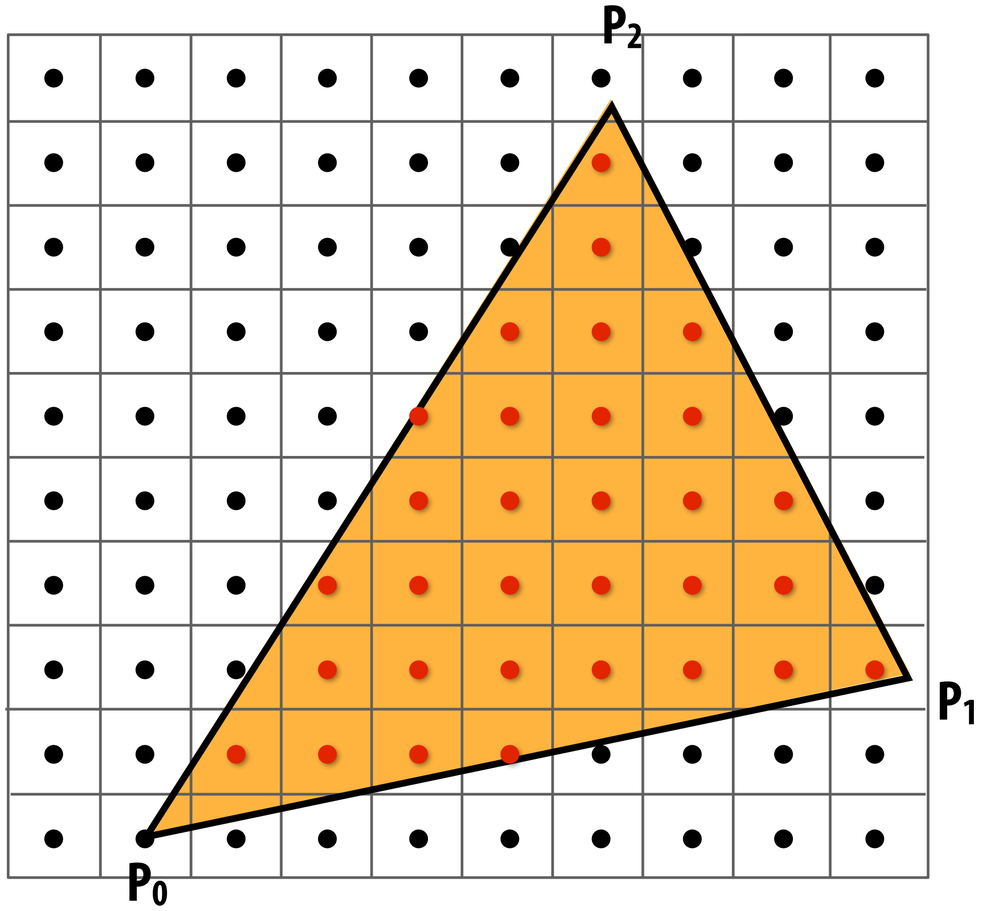


$$L_2(x, y) < 0$$

Point-in-triangle test

Sample point $s = (x, y)$ is inside the triangle if it is inside all three edges. **

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

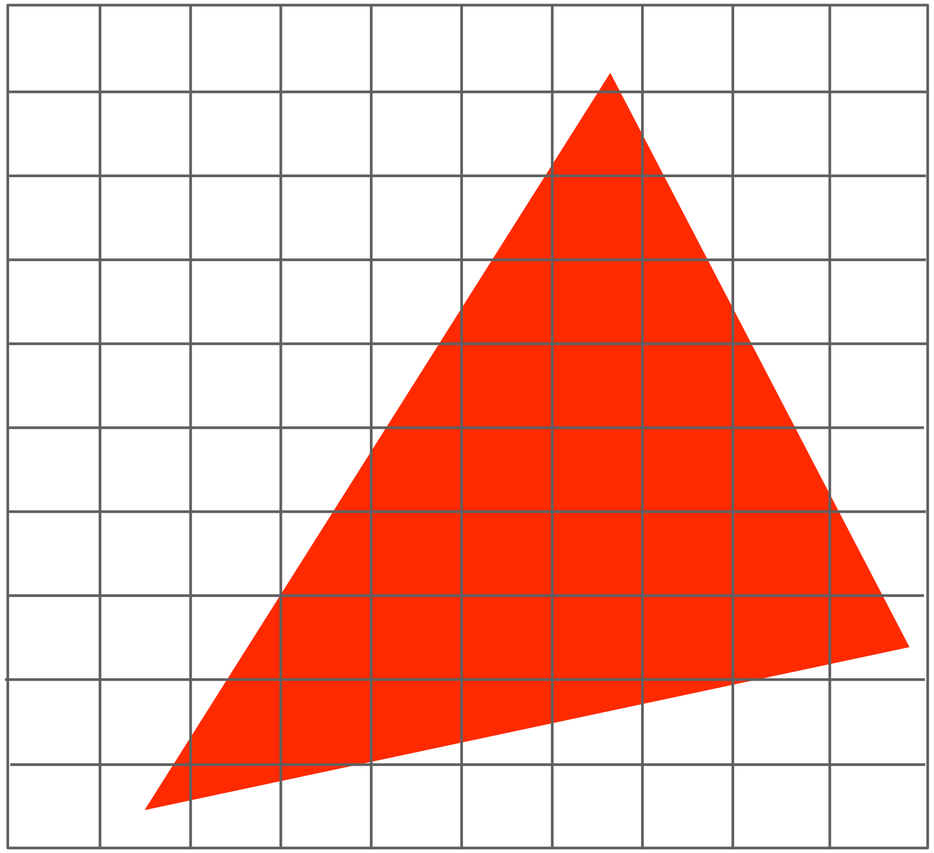


Sample points inside triangle are highlighted red.

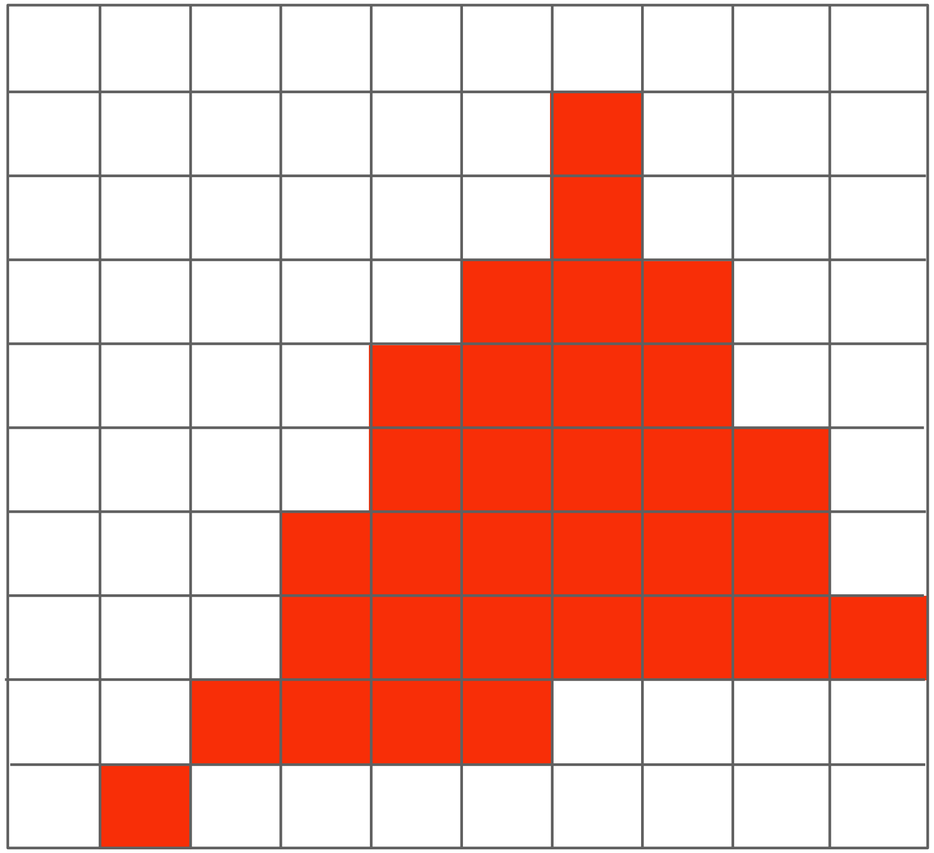
** **Note: actual implementation of $inside(x,y)$ involves ≤ checks based on the triangle coverage edge rules**

So here's our triangle…

(Overlaid over a pixel grid)



This is the result of rasterizing the triangle using our method

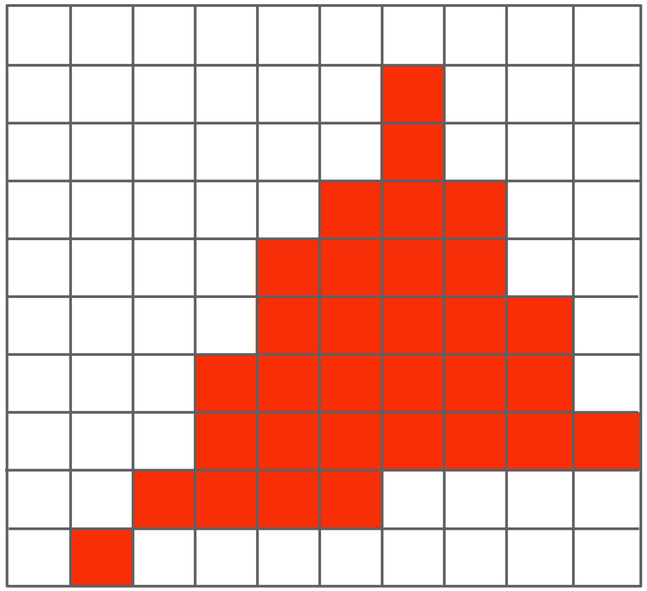


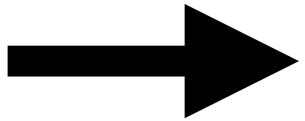
Anything that you are unsatisfied with?

One more task…

Given only an image of a triangle…
Estimate the positions of the three vertices

(a topic for later in the course)





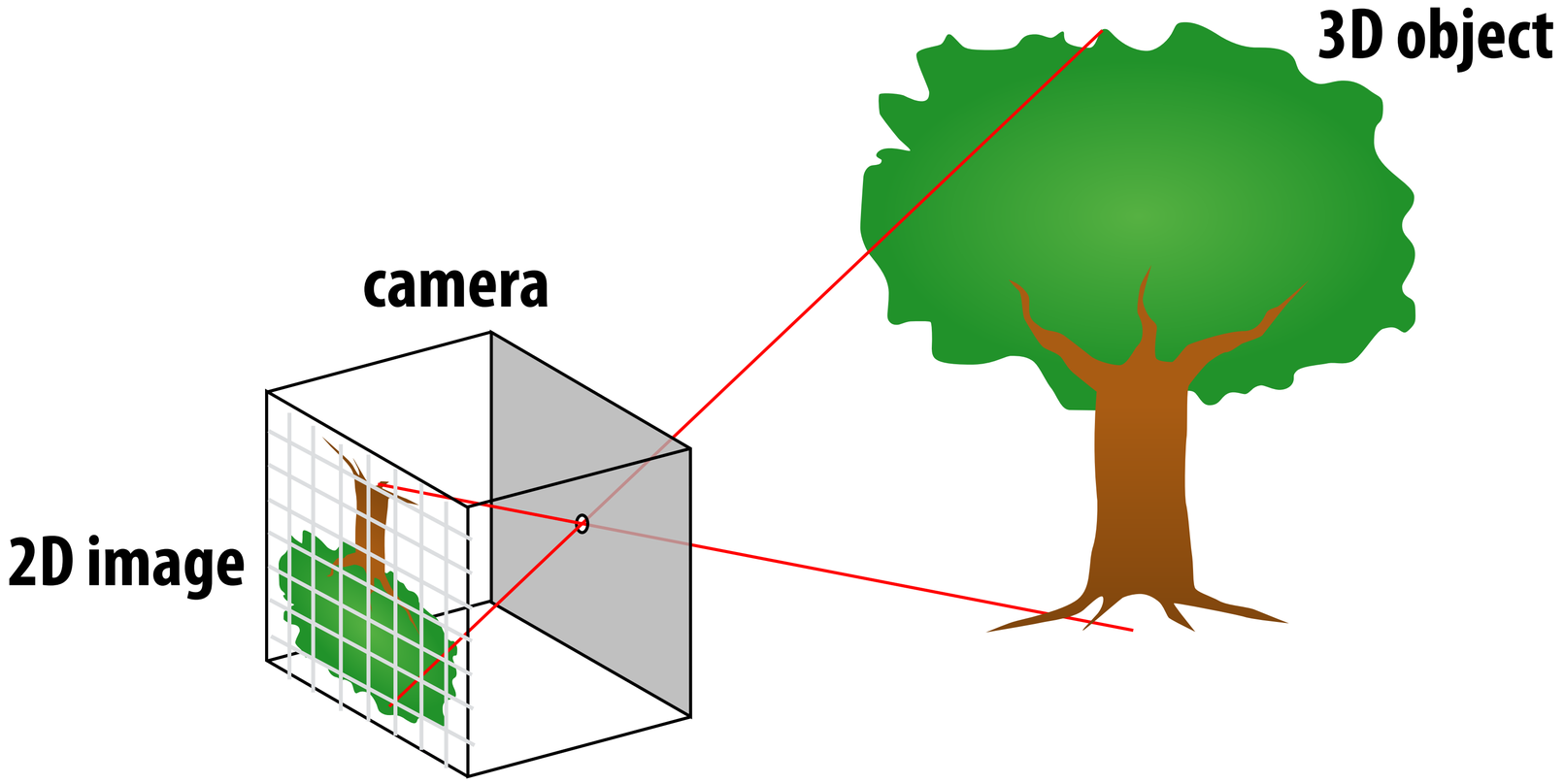
P0
P1
P2

Let's think about rendering a 3D object

We need to simulate what an object looks like, when viewed using a camera at a given position

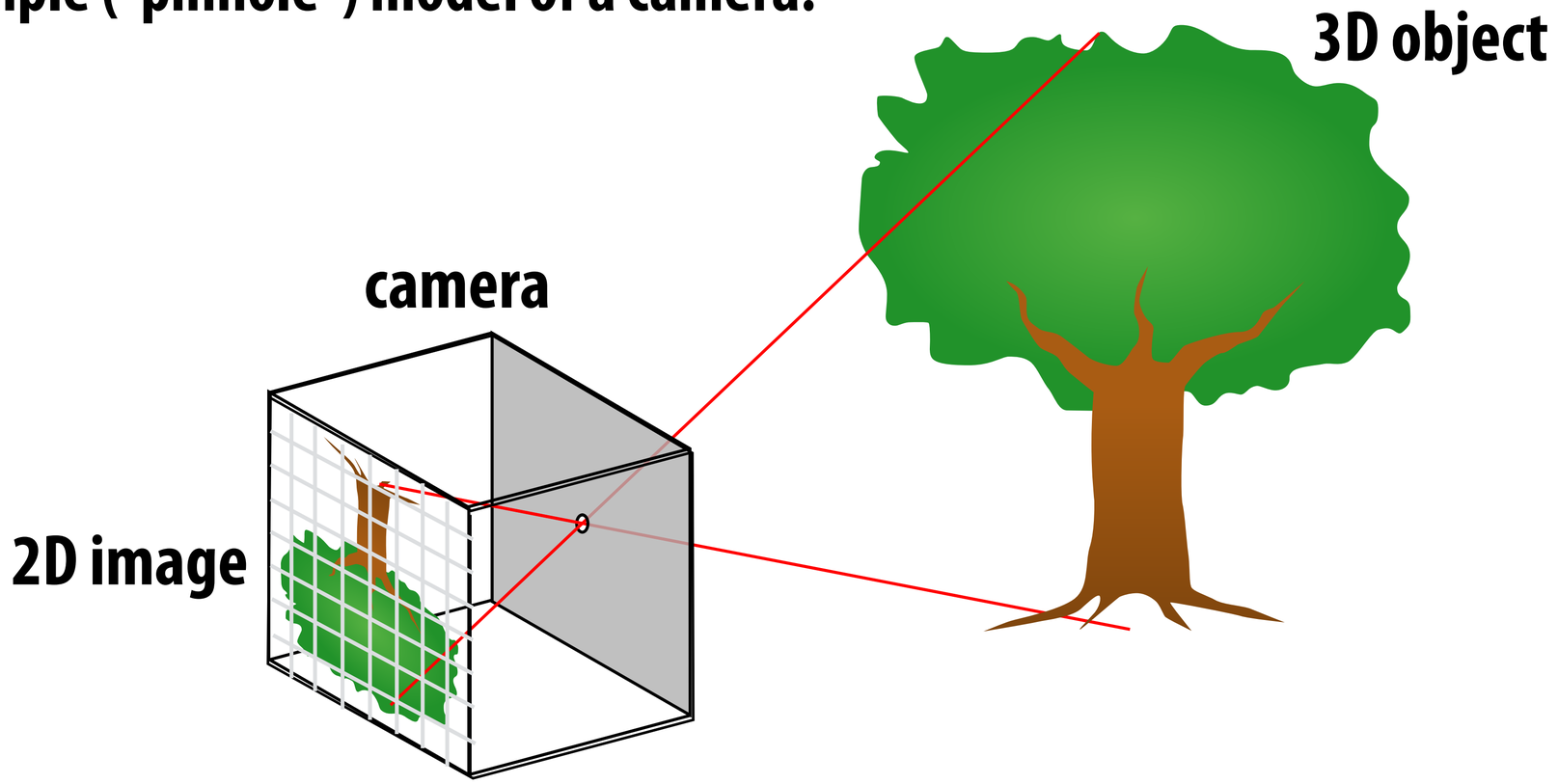
Perspective projection

- Objects look smaller as they get further away (“perspective”)
- Why does this happen?
- Consider simple (“pinhole”) model of a camera:

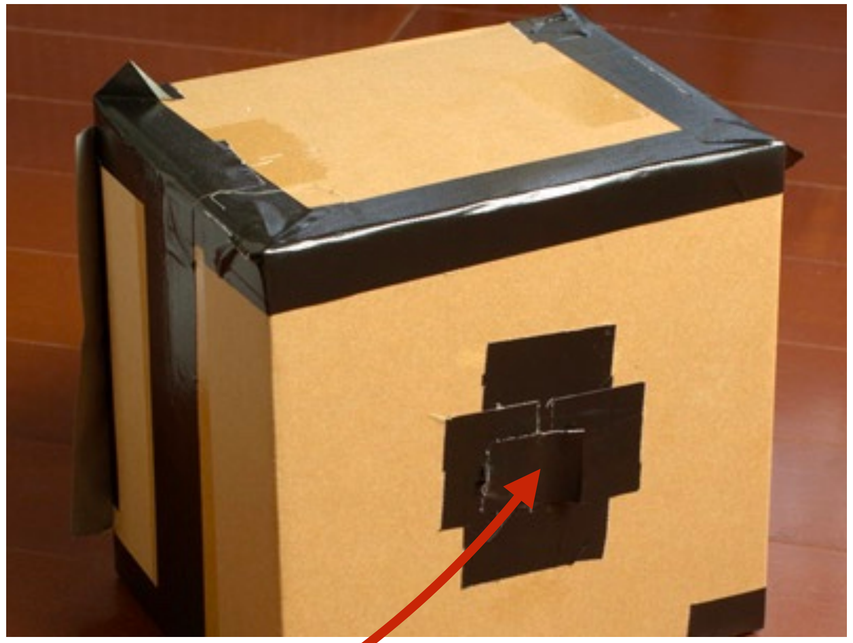


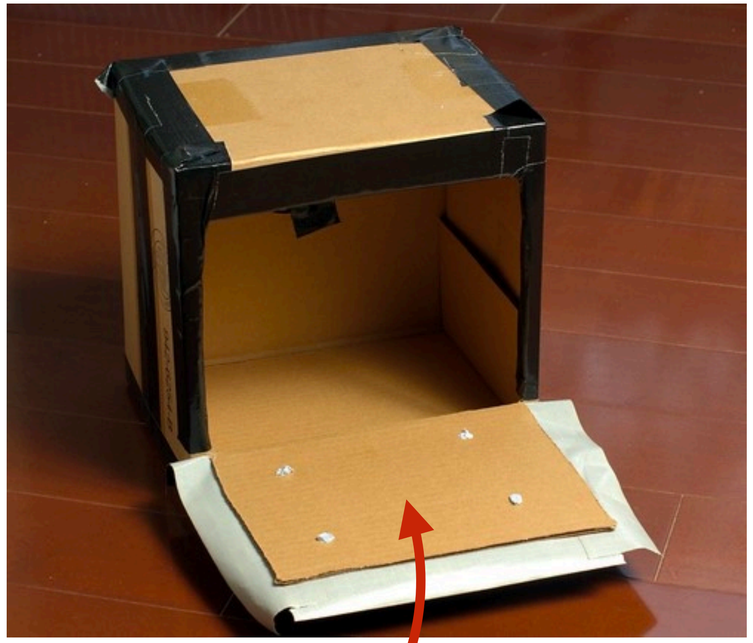
Projecting objects in a 3D scene onto a 2D image

- Where does a point in the scene appear in the image?
- Objects look smaller as they get further away (“perspective”)
- Why does this happen?
- Consider simple (“pinhole”) model of a camera:



For those that didn't do this in grade school





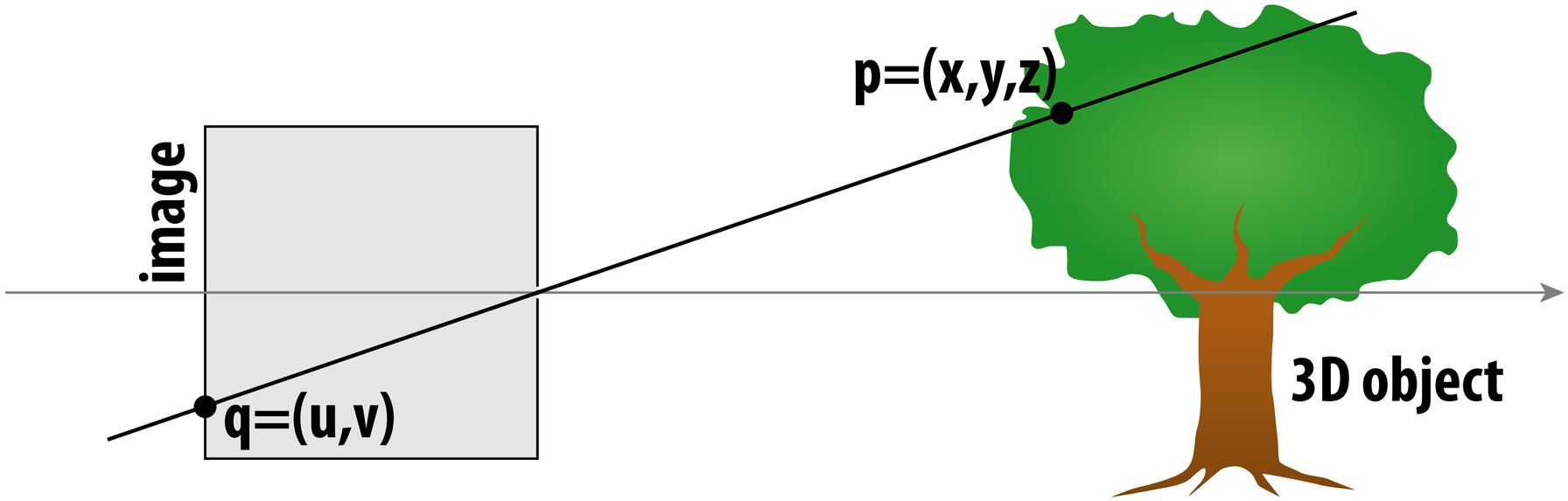
Pin hole

Place photosensitive paper here

http://janneinosaka.blogspot.com/2010/03/pinhole-time.html

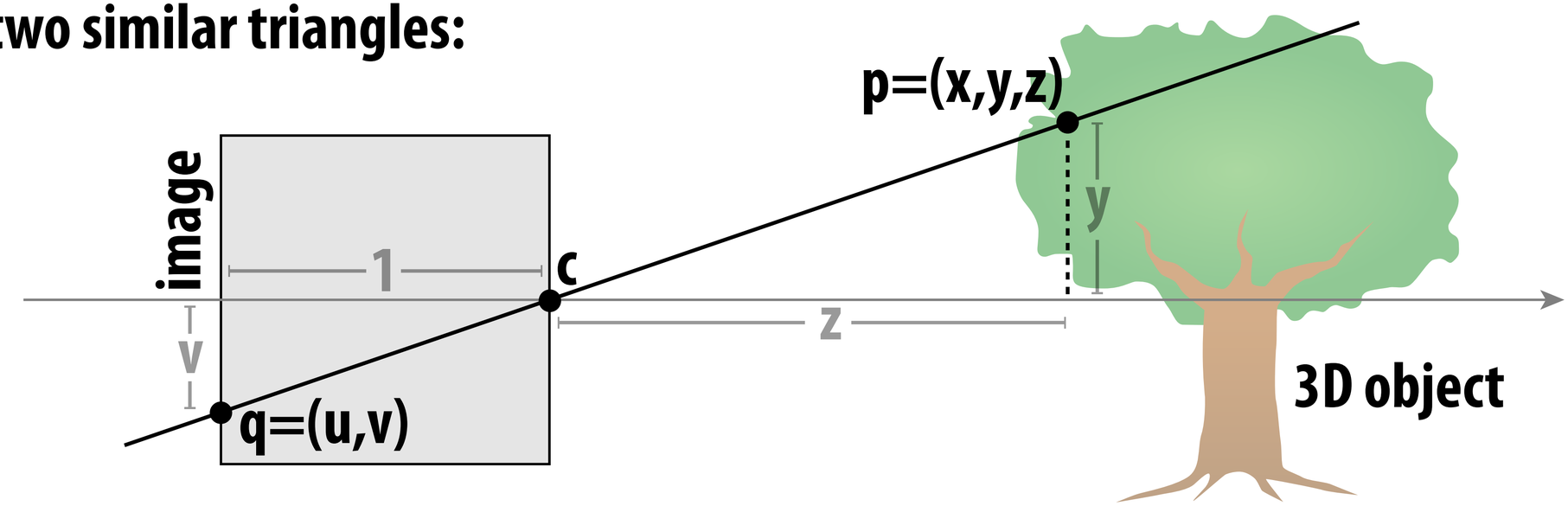
Perspective projection: side view

- Where exactly does a 3D point p = (x,y,z) on the tree end up on the image?
- Let's call the 2D image point q=(u,v)



Perspective projection: side view

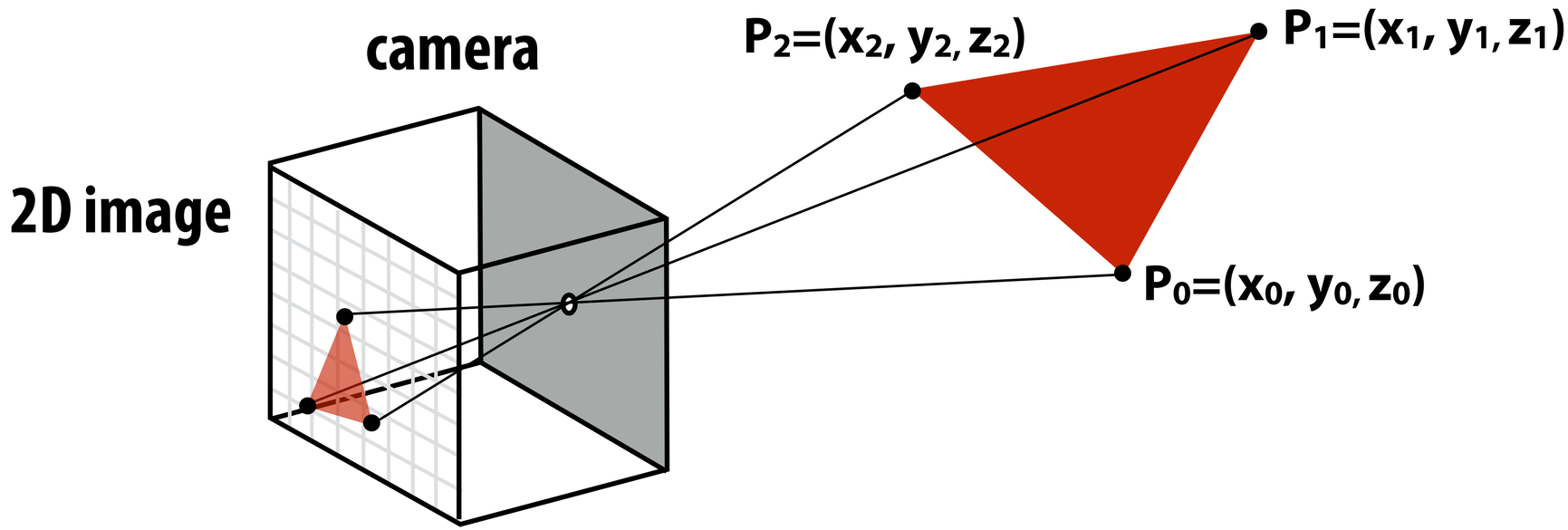
- Where exactly does a 3D point p = (x,y,z) on the tree end up on the image?
- Let's call the 2D image point q=(u,v)
- Notice two similar triangles:



- Assume camera has unit size, coordinates relative to pinhole c
- Then v/1 = y/z… v = y/z
- Likewise, horizontal offset u= x/z

Drawing a 3D triangle: rasterization perspective

Think: "What pixels does the projected triangle cover?"



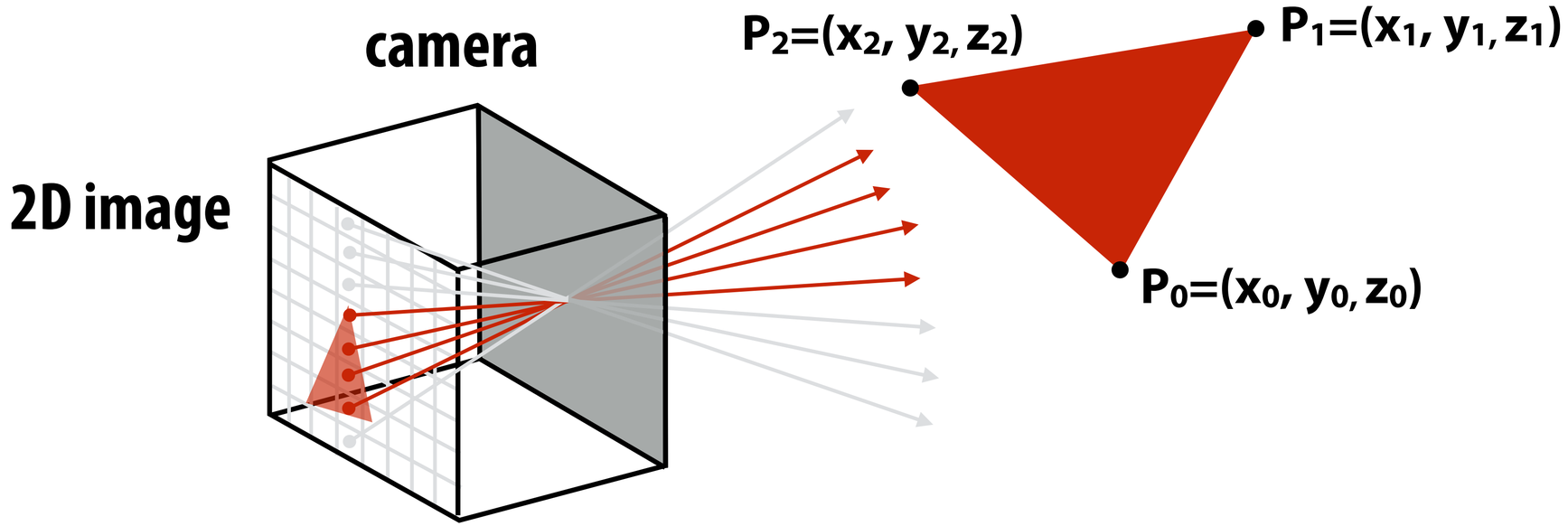
Simple pseudocode:

```
tri_proj = project_triangle(tri)
  for each image pixel p:
     if (p is inside tri_proj)
       color pixel p the color of tri_proj
```

Drawing a 3D triangle: ray casting perspective

Think: "Is the triangle visible along the ray from a pixel through the pinhole?"

Aka. Does a ray originating at the pixel center and leaving the camera "hit" the triangle?



Simple pseudocode:

```
for each image pixel p:
   let r = ray from p leaving camera
   if (r hits tri)
     color pixel p the color of tri
```

We know how to compute whether a point is inside a 2D triangle
But what about whether a ray hits a 3D triangle?

It turns out we already have everything we need!

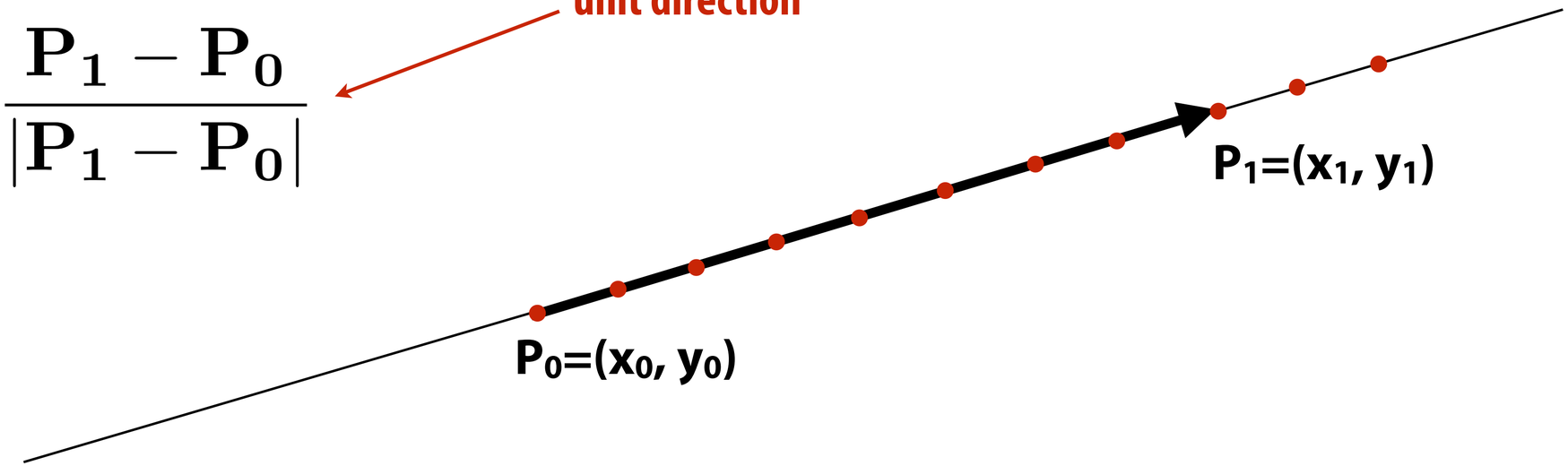
Recall parametric form of a 2D line

point that is distance *t* along line *L* from P_0.

$$\mathbf{L}(t) = \mathbf{P_0} + t\mathbf{d}$$

unit direction

$$\mathbf{d} = \frac{\mathbf{P_1} - \mathbf{P_0}}{|\mathbf{P_1} - \mathbf{P_0}|}$$



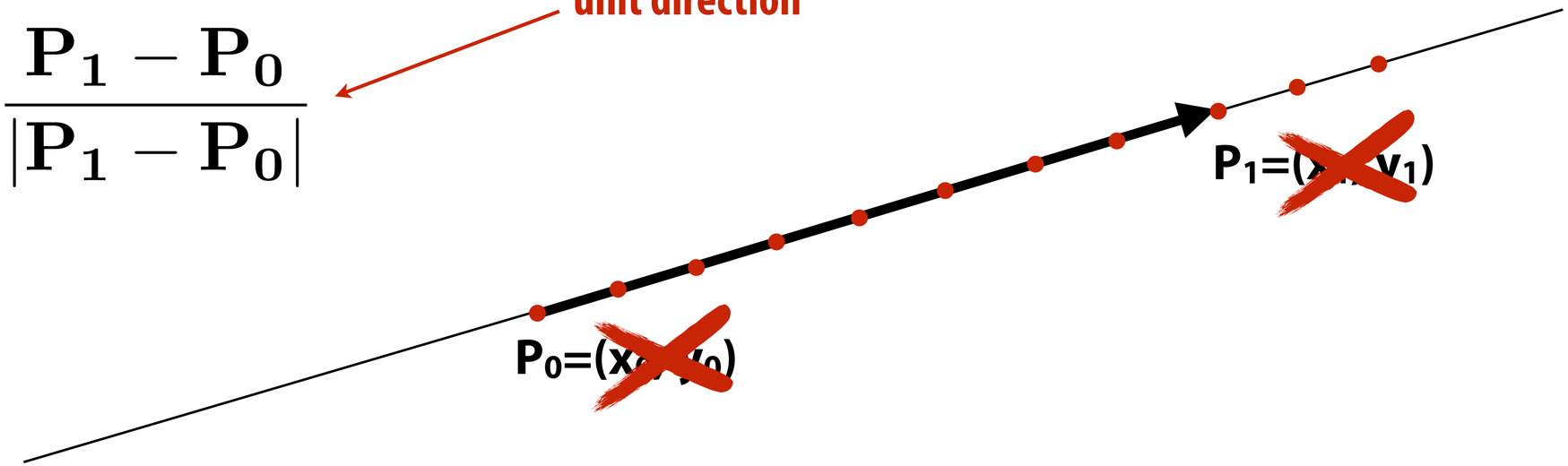
Recall parametric form of a 2D line in N-D space

point that is distance *t* along line *L* from P_0.

$$\mathbf{L}(t) = \mathbf{P_0} + t\mathbf{d}$$

unit direction

$$\mathbf{d} = \frac{\mathbf{P_1} - \mathbf{P_0}}{|\mathbf{P_1} - \mathbf{P_0}|}$$



3D ray equation (parametric)

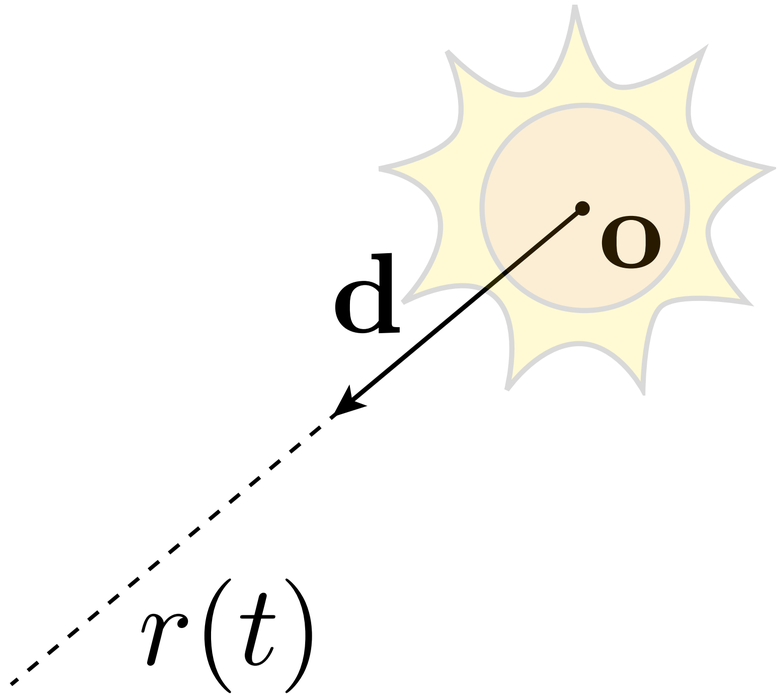
ray origin

unit direction

$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$$

point along ray

Distance along ray
(some students think "time")



How do we determine if a ray intersects a triangle?

What about where a ray intersects the plane containing the triangle?

Intersecting a ray with a 3D plane

- Parametric form of a ray: $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$
- So for the ray to intersect the plane, there must be some value of *t* for which *r(t)* is on the plane.

- We've seen how to determine if a point is on a line…
- What about point on plane?

Matrix form of a 2D line (and a 3D plane)

Line is defined by:

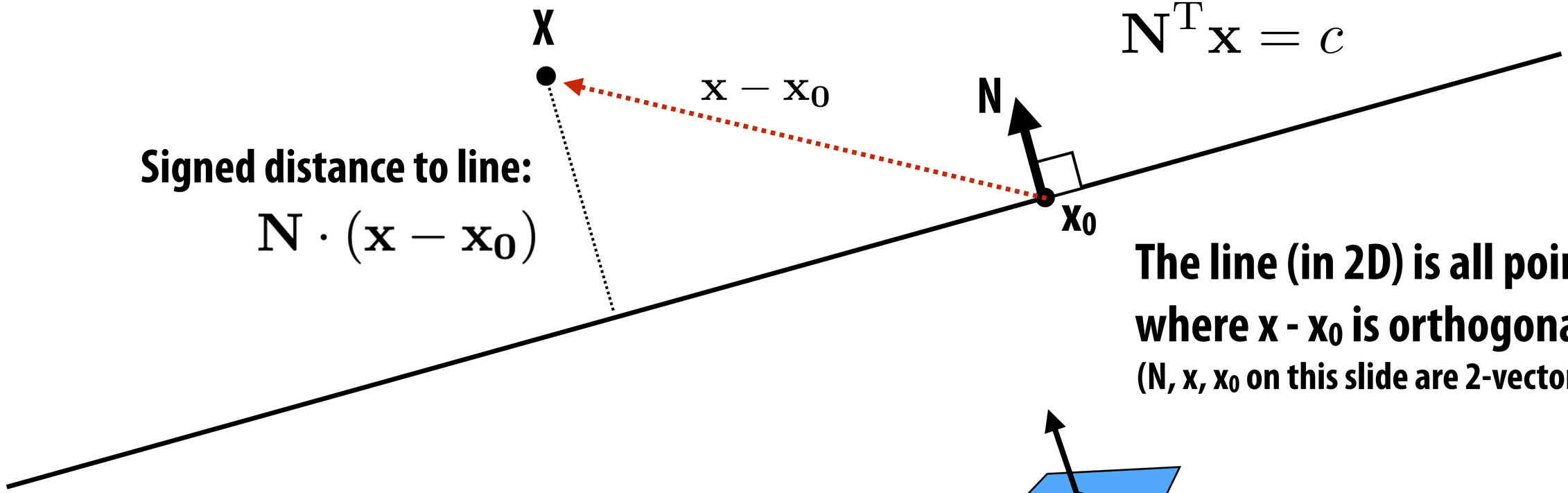
- **Its normal: N**
- **A point x_0 on the line**

$$\mathbf{N} \cdot (\mathbf{x} - \mathbf{x_0}) = 0$$

$$\mathbf{N}^{\mathrm{T}}(\mathbf{x} - \mathbf{x_0}) = 0$$

$$\mathbf{N}^{\mathrm{T}}\mathbf{x} = \mathbf{N}^{\mathbf{T}}\mathbf{x_0}$$

$$\mathbf{N}^{\mathrm{T}}\mathbf{x} = c$$



Signed distance to line:

$$\mathbf{N} \cdot (\mathbf{x} - \mathbf{x_0})$$

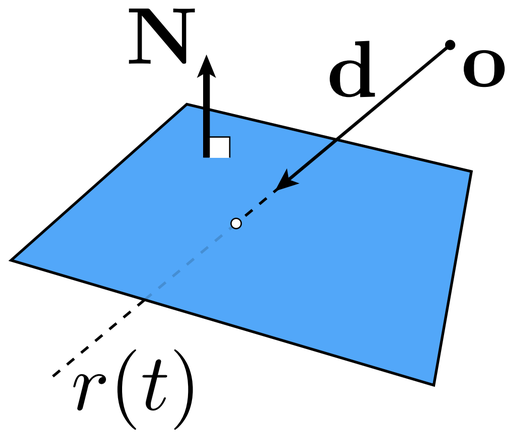
The line (in 2D) is all points x, where x - x_0 is orthogonal to N.

(N, x, x_0 on this slide are 2-vectors)

(And a plane (in 3D) is all points x where x - x_0 is orthogonal to N.)

In the plane case… N, x, x_0 are 3-vectors

Ray-plane intersection



- Suppose we have a plane $\mathbf{N}^T\mathbf{x} = c$
  - N - unit normal
  - c - offset
- How do we find intersection with ray r(t) = o + td?

- *Key idea:* again, replace the point x with the ray equation t:

$$\mathbf{N}^T\mathbf{r}(t) = c$$

- Now solve for t:

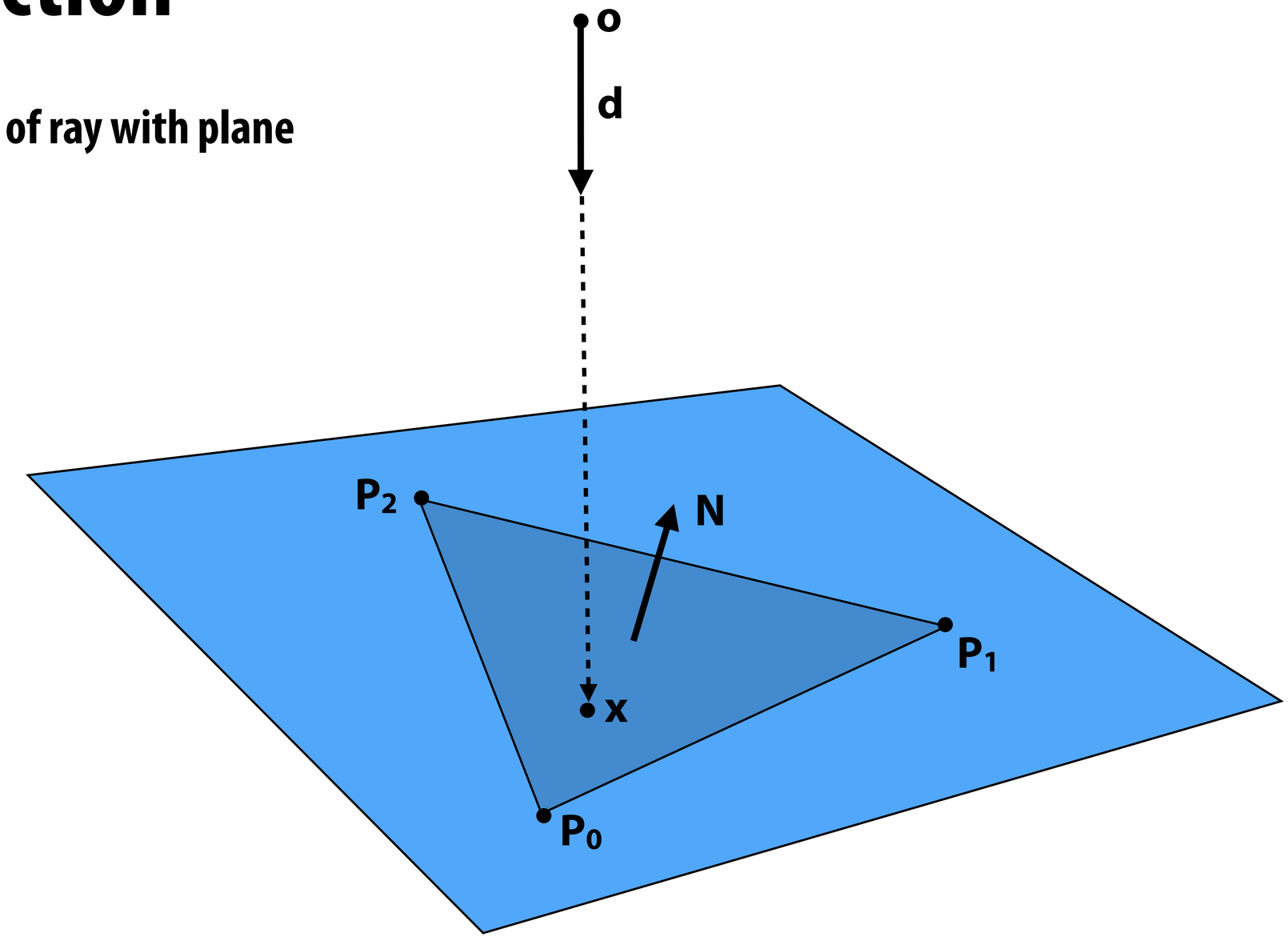
$$\mathbf{N}^T(\mathbf{o} + t\mathbf{d}) = c \qquad \Rightarrow t = \frac{c - \mathbf{N}^T\mathbf{o}}{\mathbf{N}^T\mathbf{d}}$$

- And plug t back into ray equation:

$$r(t) = \mathbf{o} + \frac{c - \mathbf{N}^T\mathbf{o}}{\mathbf{N}^T\mathbf{d}}\mathbf{d}$$

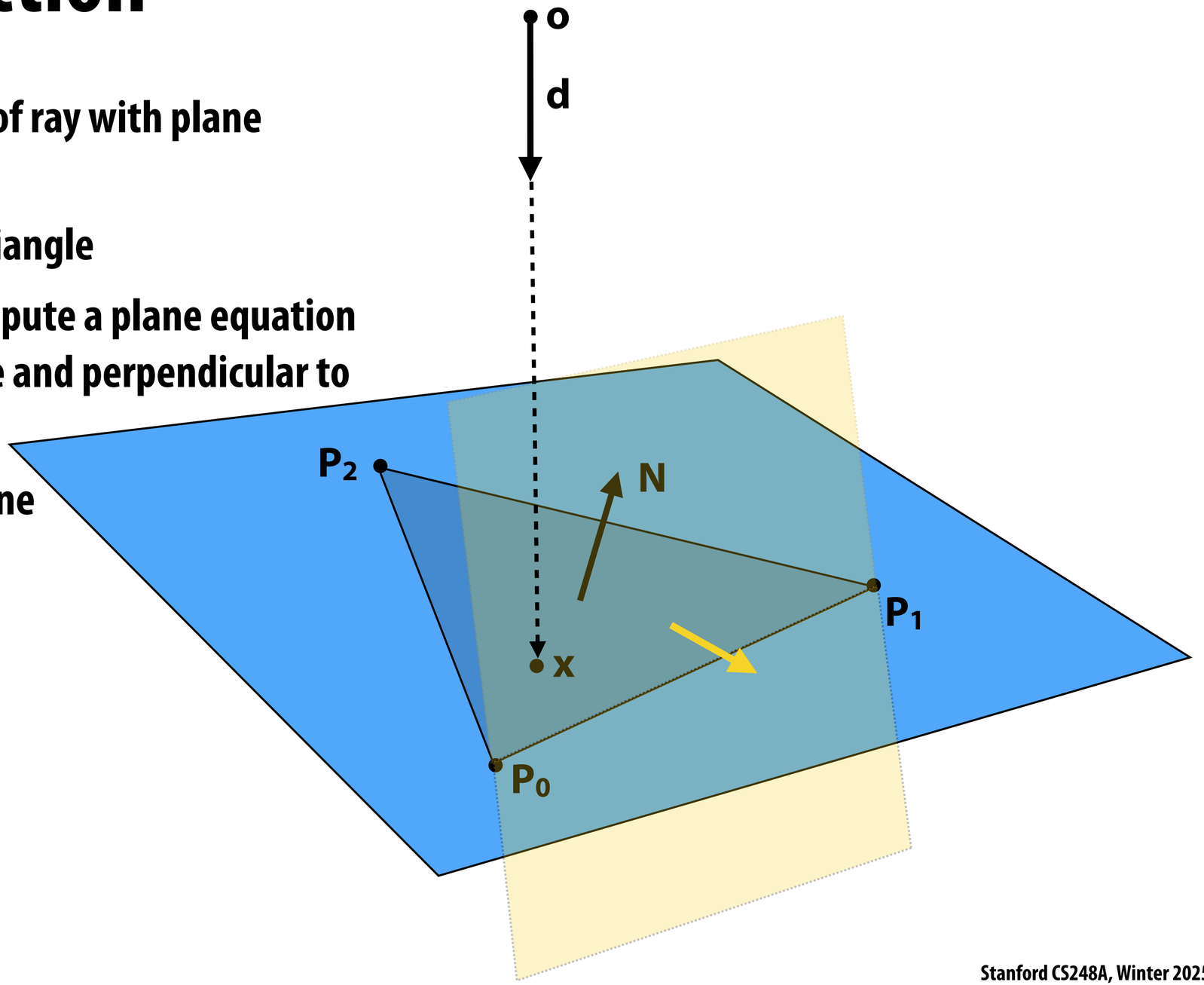
Ray-triangle intersection

- Step 1: find point of intersection (x) of ray with plane



Ray-triangle intersection

- Step 1: find point of intersection (x) of ray with plane containing triangle
- Step 2: determine if x is inside the triangle
  - For each edge of the triangle compute a plane equation for the plane containing the edge and perpendicular to the plane of the triangle
  - Determine if x is "inside" that plane



Recap: what have we done so far

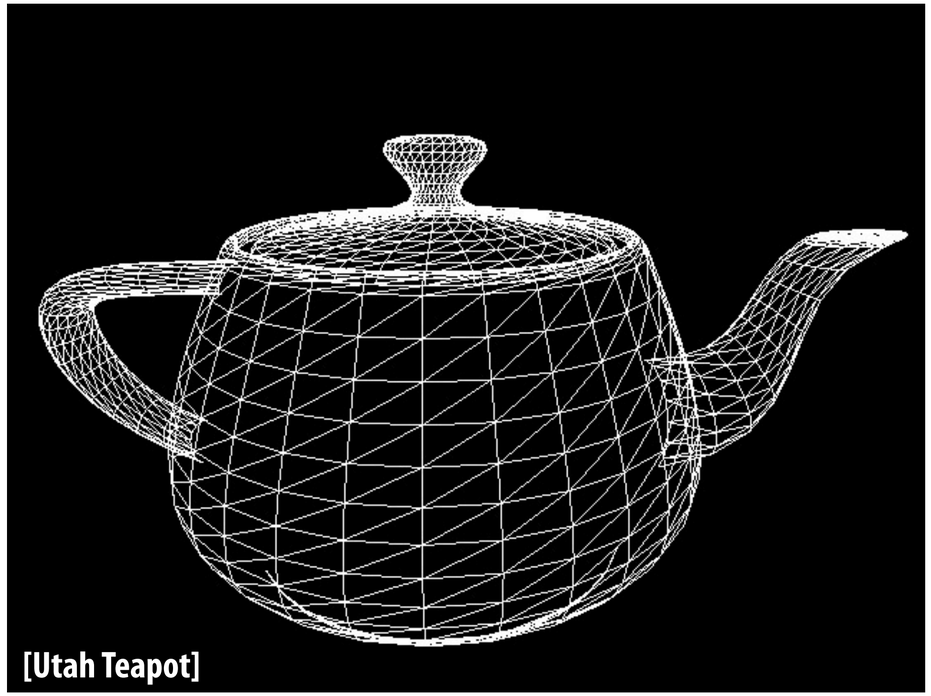
- We've thought about the task of rendering a picture from a 3D scene as:
  - Rasterization: perspective project THEN check 2D triangle's cover of pixel centers
  - Ray casting: see what 3D triangle is "hit" by a ray originating at a pixel center and leaving the camera

- In both cases we'll get the same picture…
  - Which suffers from "jaggies" if the image is low resolution

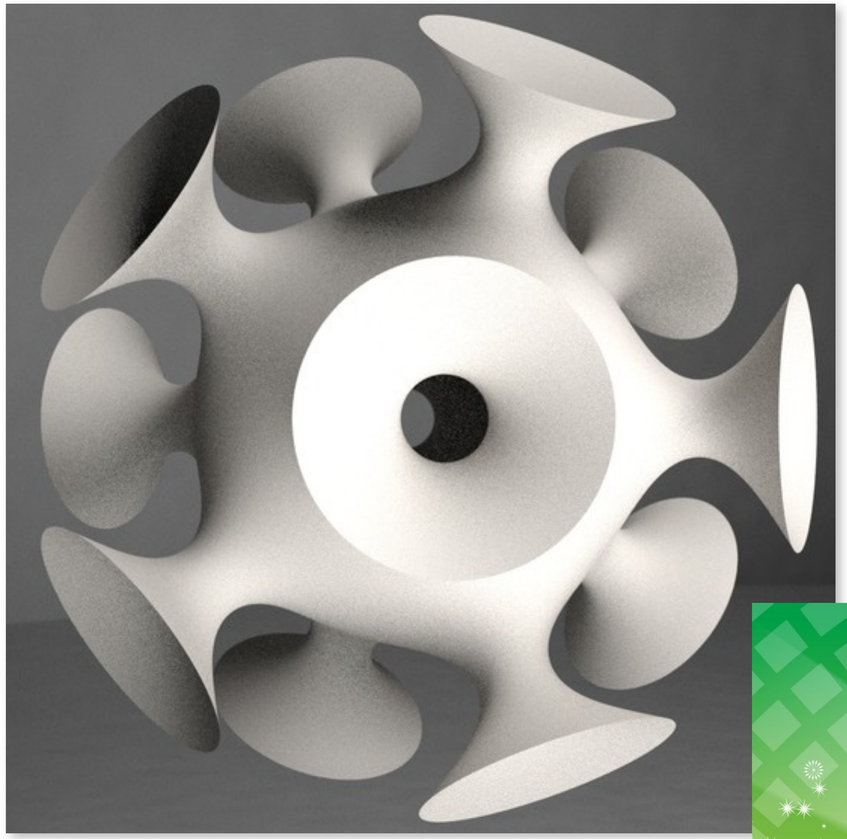
- We've shown that the matrix form of point-on-line queries generalized to point-on-plane in 3D, and we can build a 3D point in triangle test for this


But to render more realistic pictures
(or animations) we need a much richer model of the world.
e.g, complex surfaces, materials, lights

Complex 3D surfaces and 2D shapes



[Utah Teapot]



Platonic noid

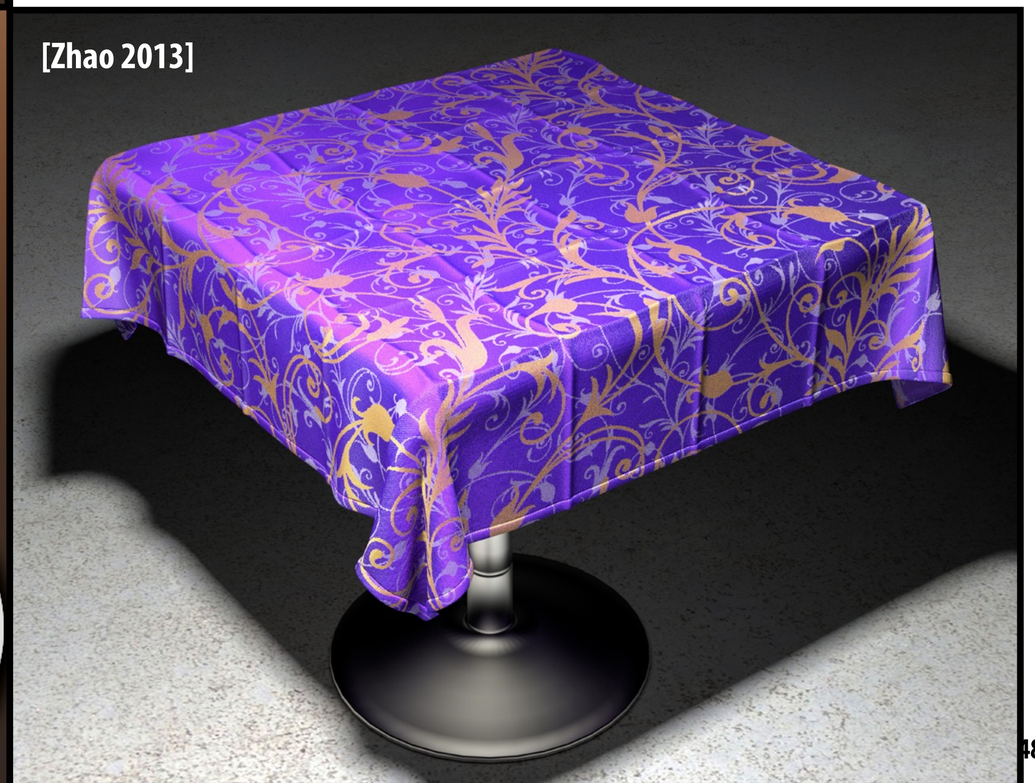
[Kaldor 2008]

[Source: Batra 2015]

Modeling material properties

[Wann Jensen 2001]

[Jakob 2014]



[Zhao 2013]

Realistic lighting environments

Wall-E, (Pixar 2008)

Animation: modeling motion

https://www.youtube.com/watch?v=6G3060o5U7w

Course Logistics and Expectations

About this course

- A broad overview of major topics and techniques fundamental to generating and editing 2D/3D scenes, images, and interactive worlds

- This year: major changes from previous versions of the course
  - Increased focus on representations suitable for visual AI and learning/optimization
  - Increased focus on ray tracing as an image generation algorithm
  - Discussions of relationship between human-engineered representations/algorithms and learned representations (e.g., NeRF's, neural rendering, and "world models")
  - All new programming assignments

- This is a learn-by-implementing course
  - Focus on implementing fundamental data structures and algorithms
  - We expect that you can understand/write/debug code in Python and in C-like languages (We'll be using a new GPU programming language called Slang)

Assignments / grading

- (56%) Programming assignments (including a self selected project)
  - Done in teams of up to two students (yes, you can work alone if you wish)
- (12%) 4 practice exercises (participation only)
  - Think of these as possible exam problems
  - Done in teams of three. We assign the teams randomly each assignment
- (4%) Post-lecture quizzes (participation only)
  - Done individually
  - Must be submitted by 11:59pm the day following the lecture
- (10%) Oral exam
  - With the course staff (during week 6)
- (18%) Written Exam
  - Evening exam (during week 8)

FAQ

- How are CS248A and CS248B related?
  - They are explicitly designed to be independent starter courses for the visual computing track. There is no assumption you've taken CS248A before CS248B or vice versa.
  - The biggest point of content overlap is the lecture on transforms

- Are lectures recorded?
  - Yes, since this is an GCOE class

- Can I audit the class?
  - Yes, email a CA and they'll add you to Canvas

FAQ

- Is there a final?
  - No… the final exam slot is used for our project showcase. Please plan to attend.

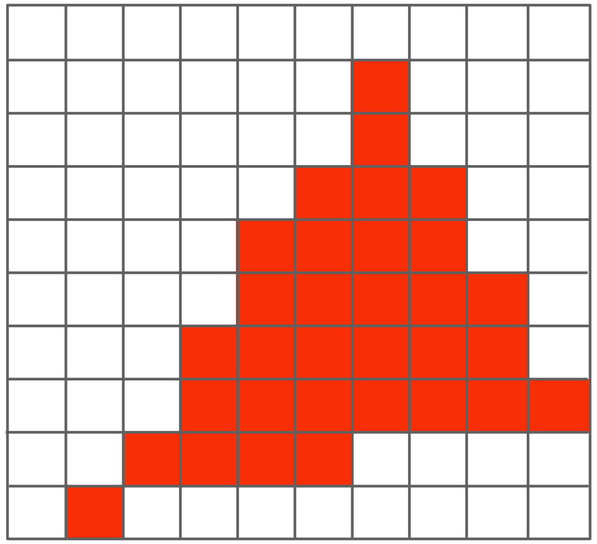
- Do I need a partner for programming assignments?
  - No, each year there are students that choose to do all the programming assignments alone
  - Need a partner: we will find one for you, via our partner search form

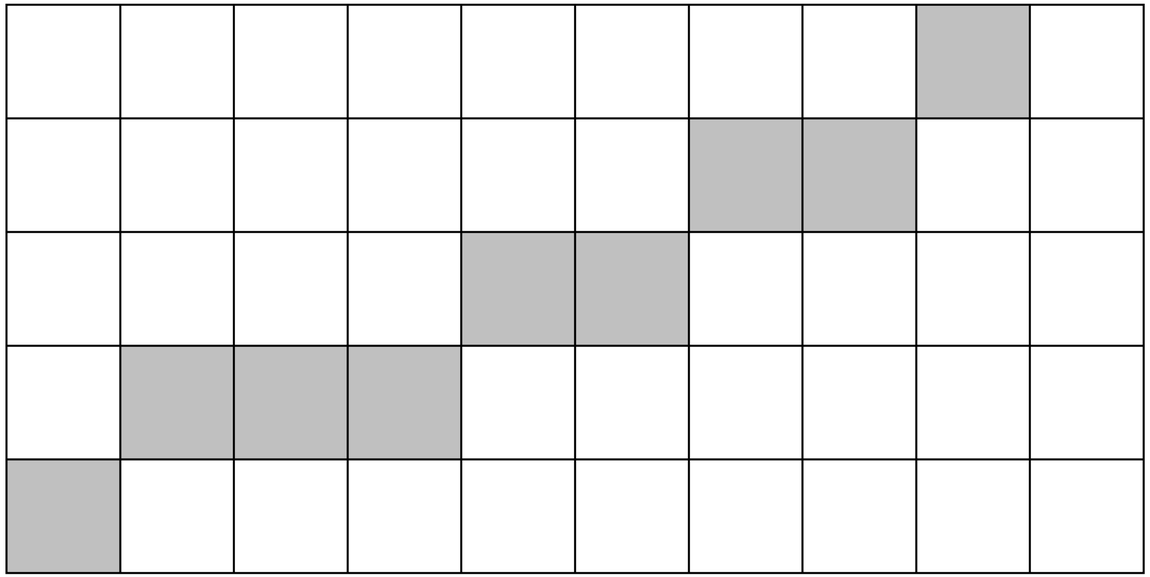
- What are the prereqs for CS248A?
  - You should have the math background: linear algebra (at least MATH 51) and 3D calculus
  - You should have the coding background: Python and C-like languages (probably at least CS107)
  - CS148 is not a pre-req for CS248A

See you next time!

Next time, we'll talk about drawing with more rigor

- What's up with these "jagged" lines and triangle edges?
- Why does increasing the resolution of the image improve quality?
- What can we do to improve image quality without more pixels?





Slide acknowledgements:
Thanks to Keenan Crane and Ren Ng