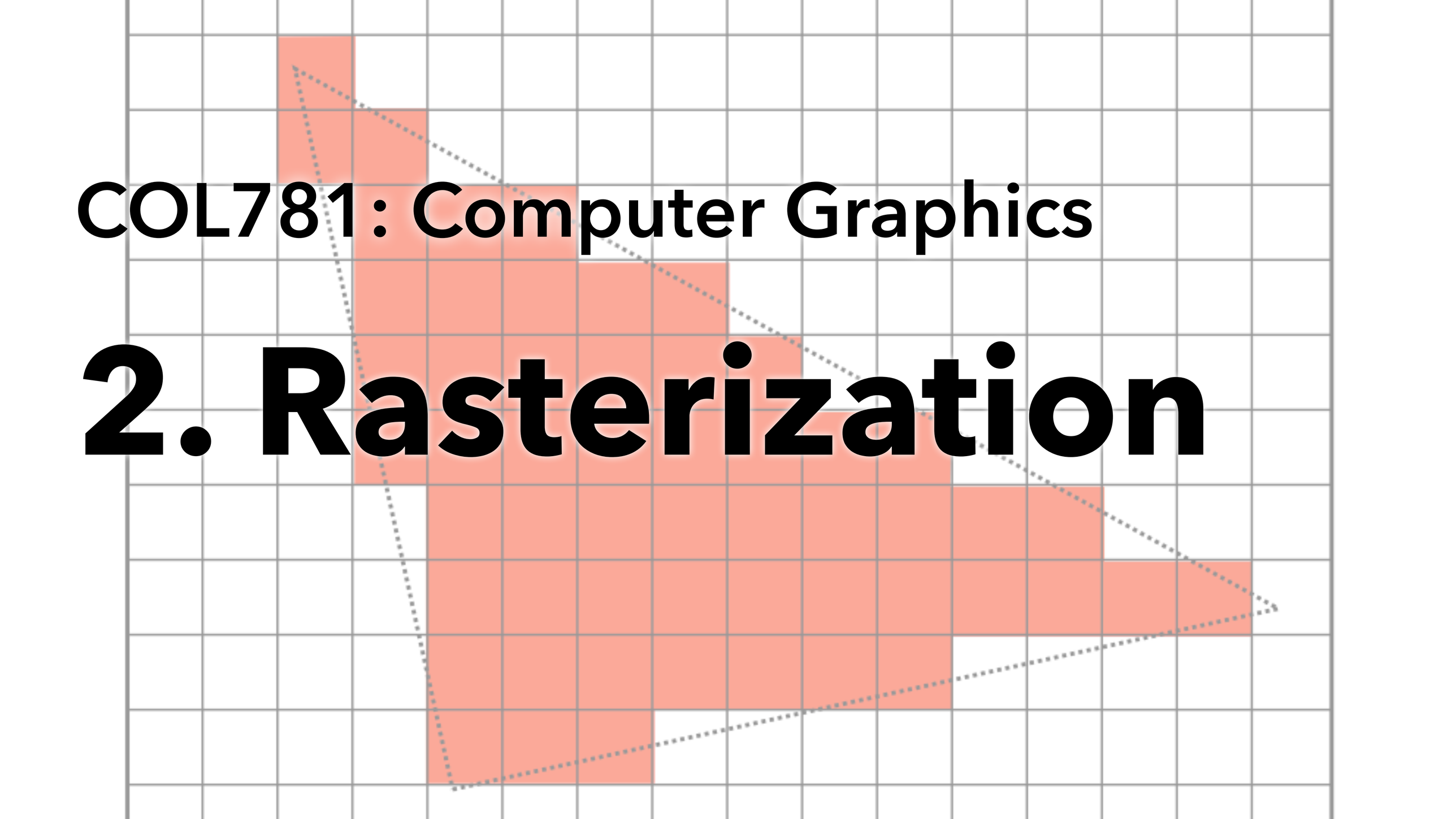


COL781: Computer Graphics

2. Rasterization

Course policies

Course webpage: https://www.cse.iitd.ac.in/~narain/, Ctrl+F 781 :)

Office hours: by appointment (please email me)

Announcements: on Moodle only

Questions: on Moodle Q&A forum only! Please do not ask by email

Textbooks: No required text, but the following are recommended:

- Hughes et al., *Computer Graphics: Principles and Practice*, 3rd Ed.
- Marschner and Shirley, *Fundamentals of Computer Graphics*, 4th or 5th Ed.

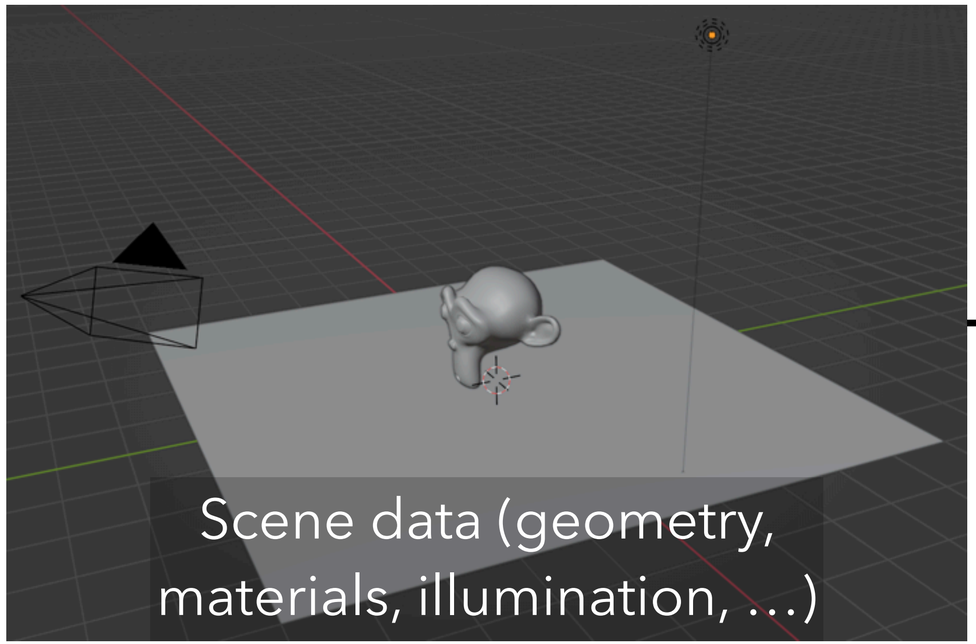
Course policies

Grading: Minimum 80% marks for A grade. Minimum 30% marks for D

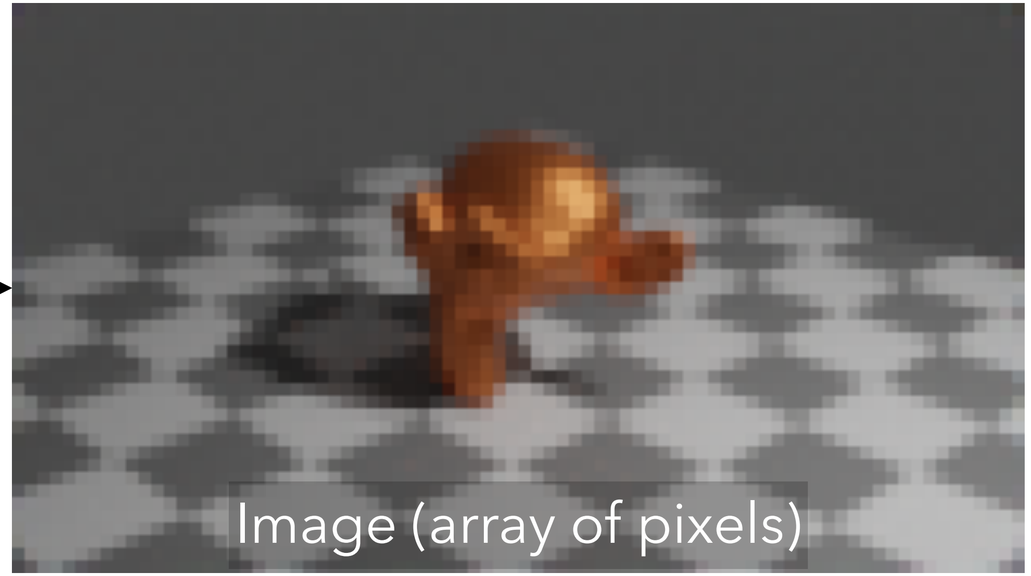
Audit policy: At least 40% in course total, and at least 20% in each assignment and each exam

Attendance policy: Attendance lower than 75% may result in a one-grade penalty (e.g. A to A–, or A– to B)

Graphics in a nutshell

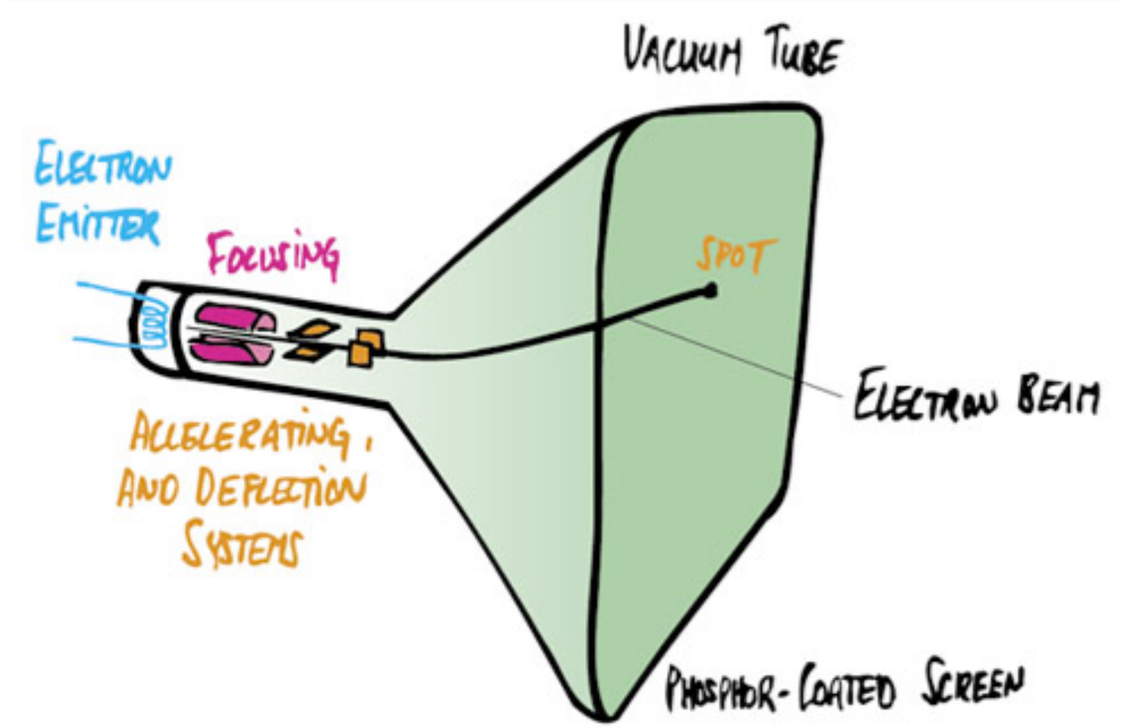


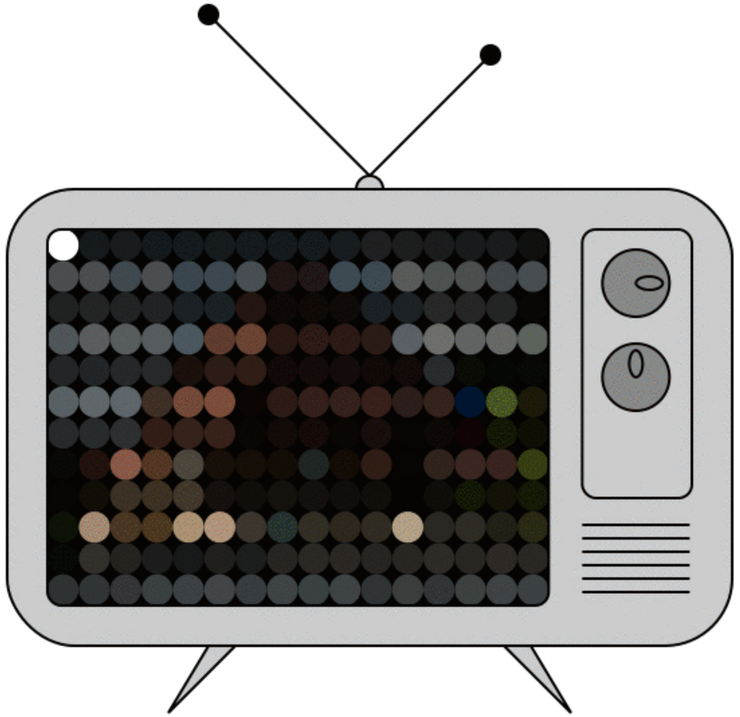
?



But why an array of pixels, specifically?

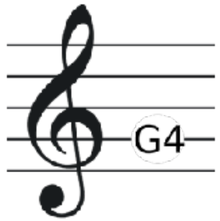
Raster displays





Cathode ray tubes (CRTs)

A rastrum

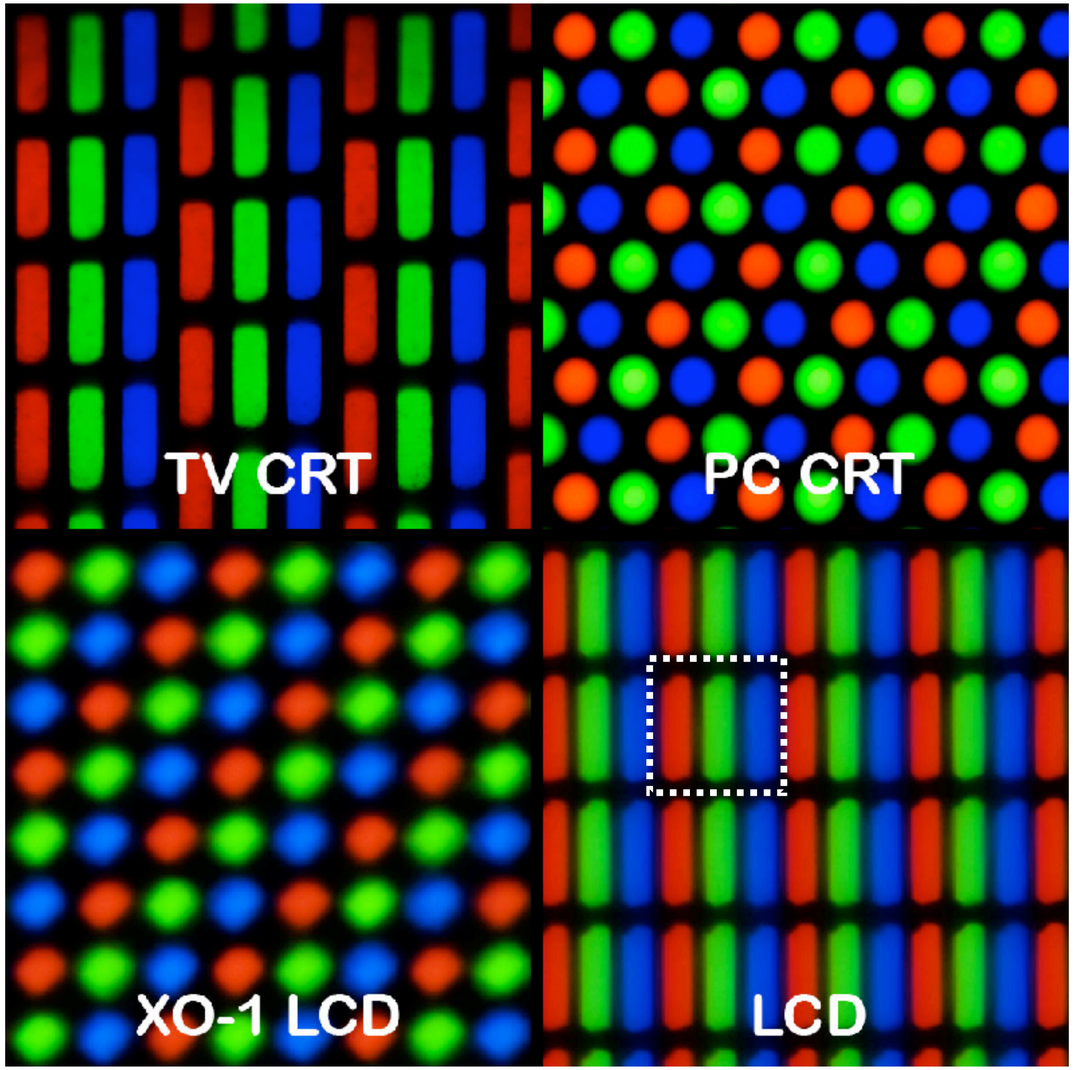


Raster displays

Liquid crystal displays (LCDs)

Raster displays

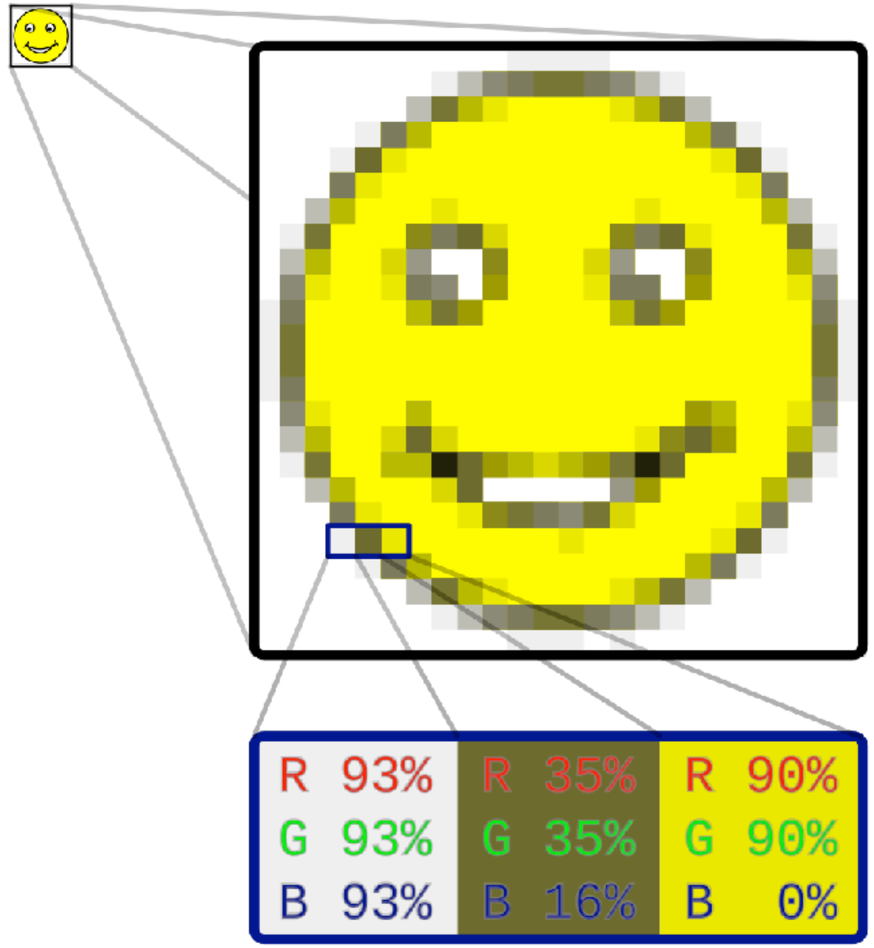
Light-emitting diodes (LEDs)


TV CRT
PC CRT
XO-1 LCD
LCD

Raster images

Raster displays provide a grid of **pixels** (picture elements) whose intensity / colour can be individually controlled

A **raster image** (or bitmap image, or just image) is a 2D array of pixel values, and can easily be displayed on a raster device.



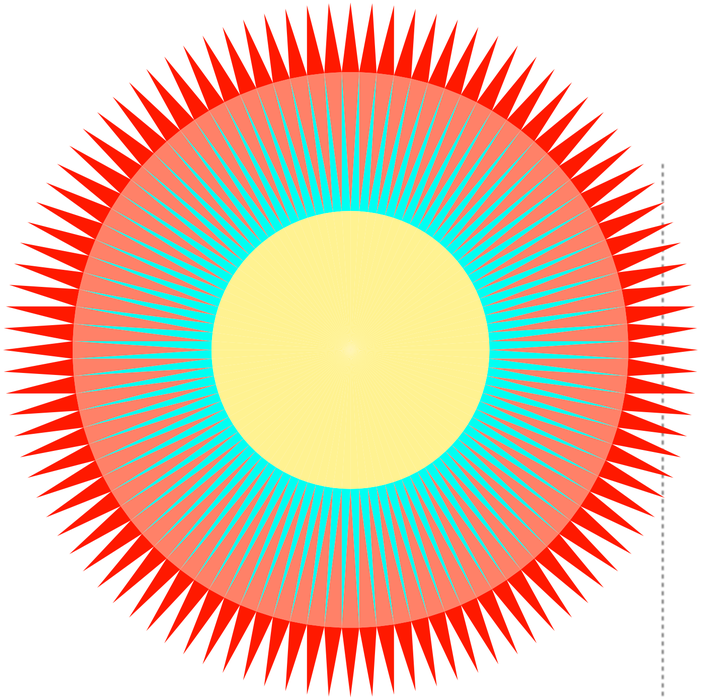
Vector graphics

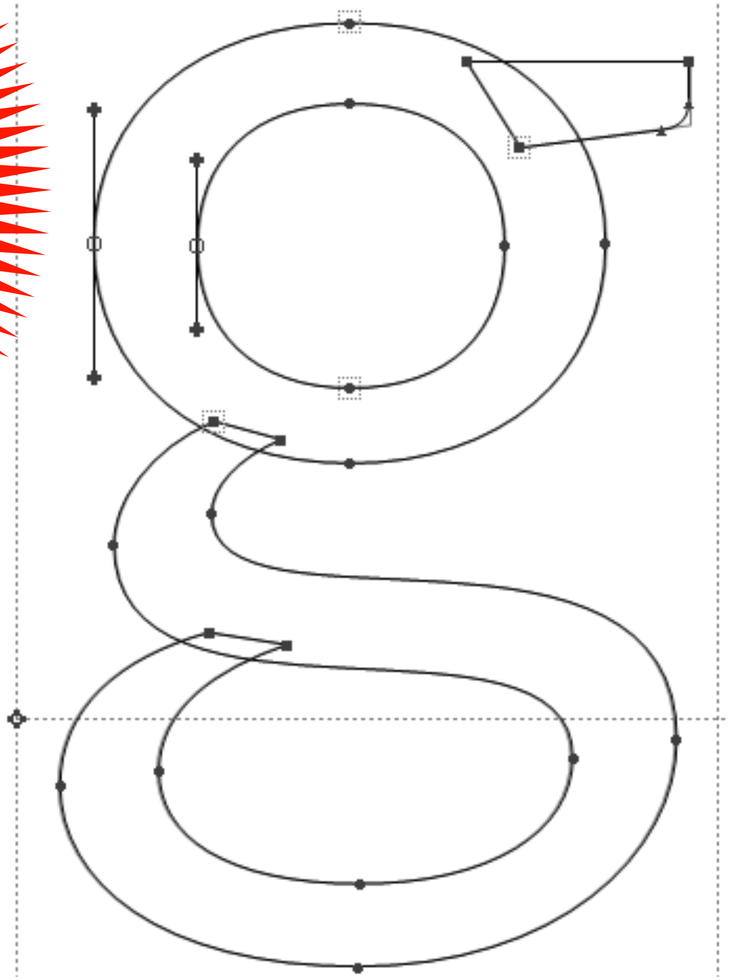
Users usually don’t want to draw things by setting individual pixels!

One wants to provide a high-level description:

```
<svg version="1.1"
     width="300" height="200"
     xmlns="http://www.w3.org/2000/svg">
  <rect width="100%" height="100%" fill="red" />
  <circle cx="150" cy="100" r="80" fill="green" />
  <text x="150" y="125" font-size="60"
        text-anchor="middle" fill="white">SVG</text>
</svg>
```

Vector graphics

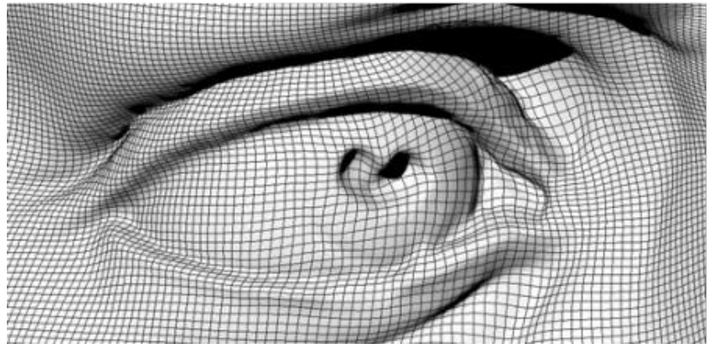


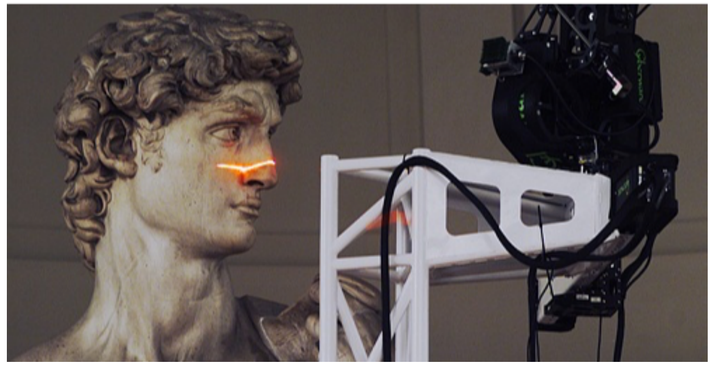


Raster image

Vector drawing

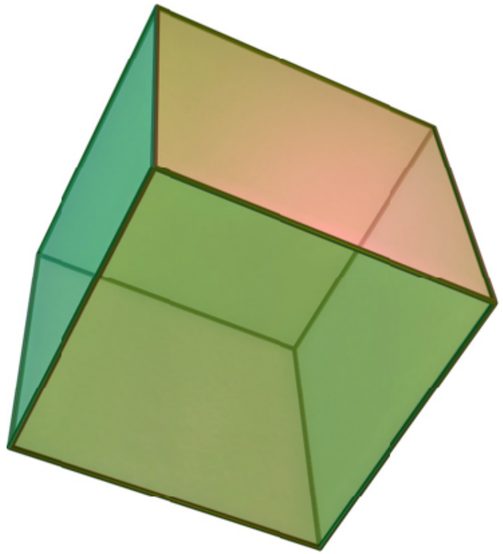
Similarly, 3D graphics models are almost always represented in a high-level form:





VERTICES

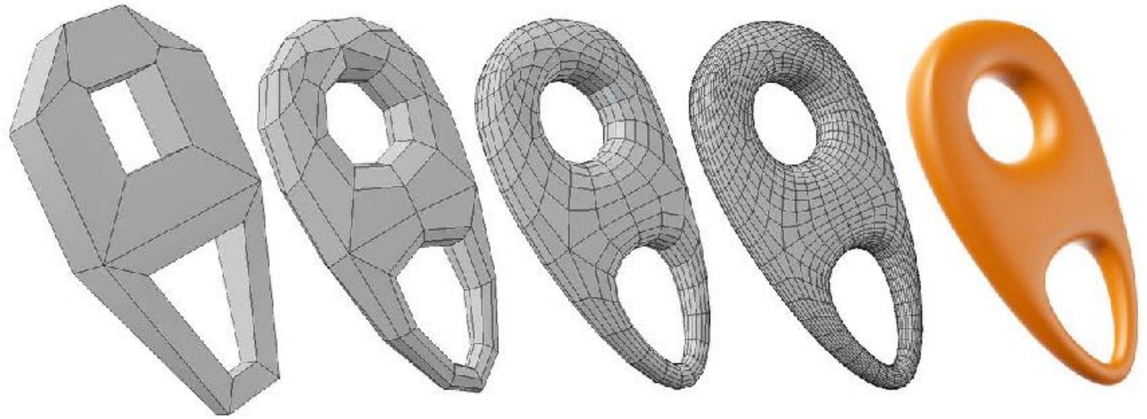
```
A:( 1, 1, 1) E:( 1, 1,-1)
B:(-1, 1, 1) F:(-1, 1,-1)
C:( 1,-1, 1) G:( 1,-1,-1)
D:(-1,-1, 1) H:(-1,-1,-1)
```



TRIANGLES

```
EHF, GFH, FGB, CBG,
GHC, DCH, ABD, CDB,
HED, ADE, EFA, BAF
```

Though in 3D we don’t refer to this as “vector graphics”!

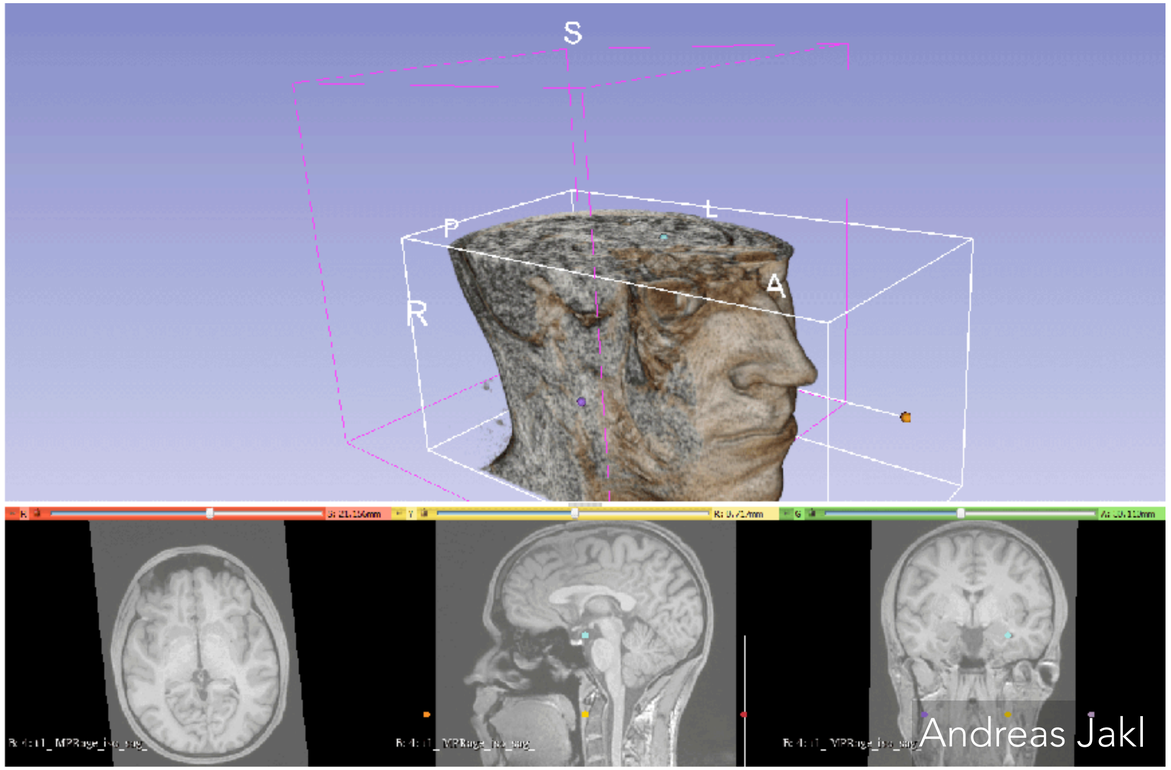


Puzzle:

What would be the 3D analogue of a raster image?

Can you think of a graphics application where it is / could be used?

Voxel data



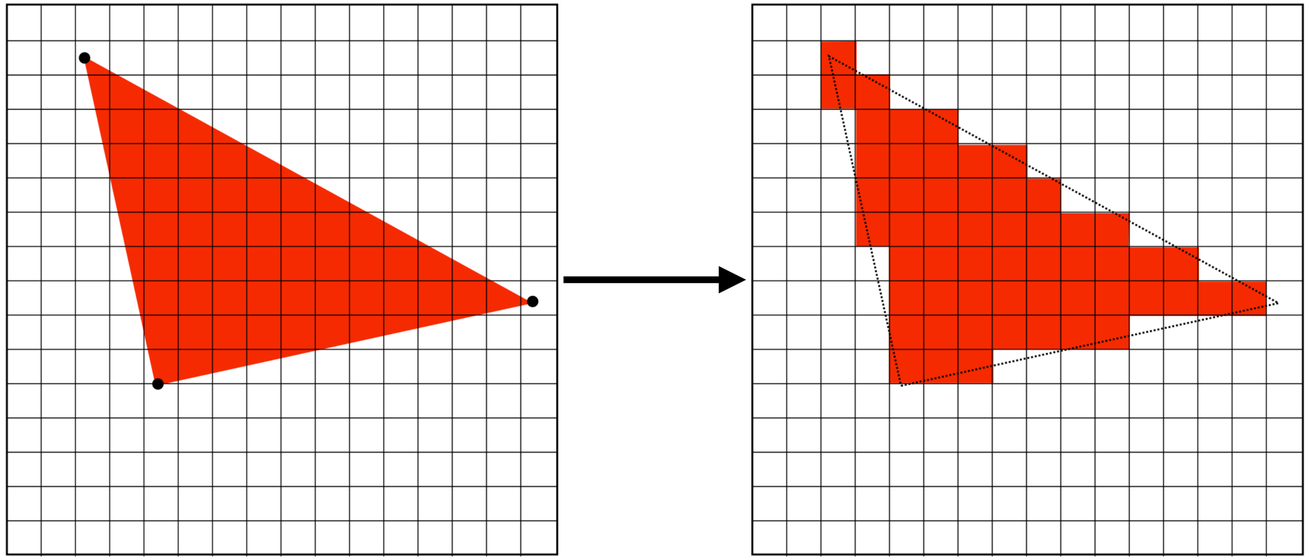
Rasterization

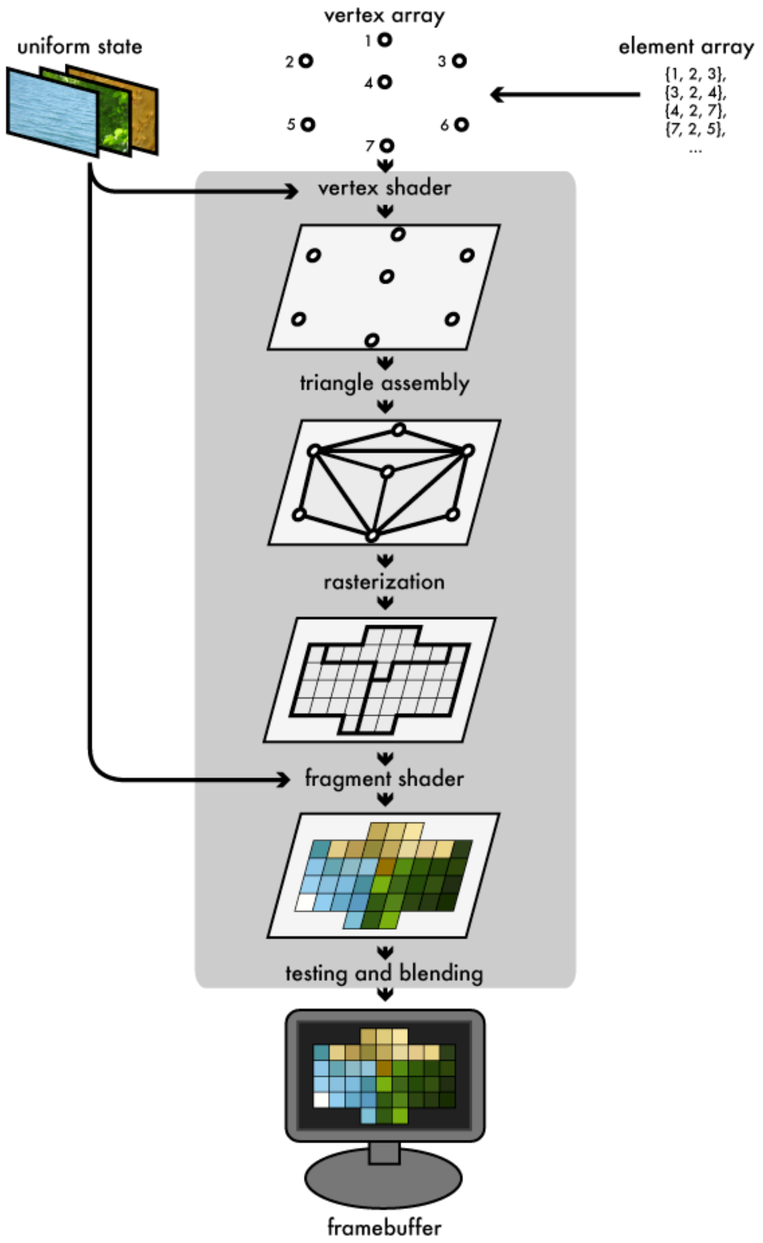
To display any 2D or 3D shape on a raster display, it needs to be rasterized!

Input: Geometrical “primitives” (usually triangles) with attributes (e.g. colour)

Output: Raster image approx-imating the given shape

Usually this is performed by the graphics processing unit (GPU)





Preview: The (real-time) graphics pipeline

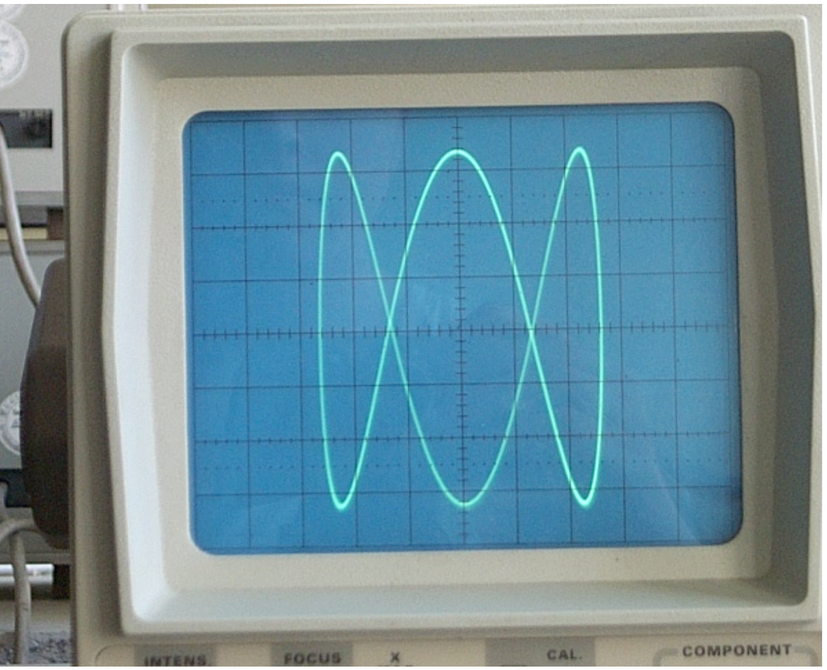
Even for 3D graphics,

1. first we project the vertices of each 3D triangle to their 2D locations on the screen

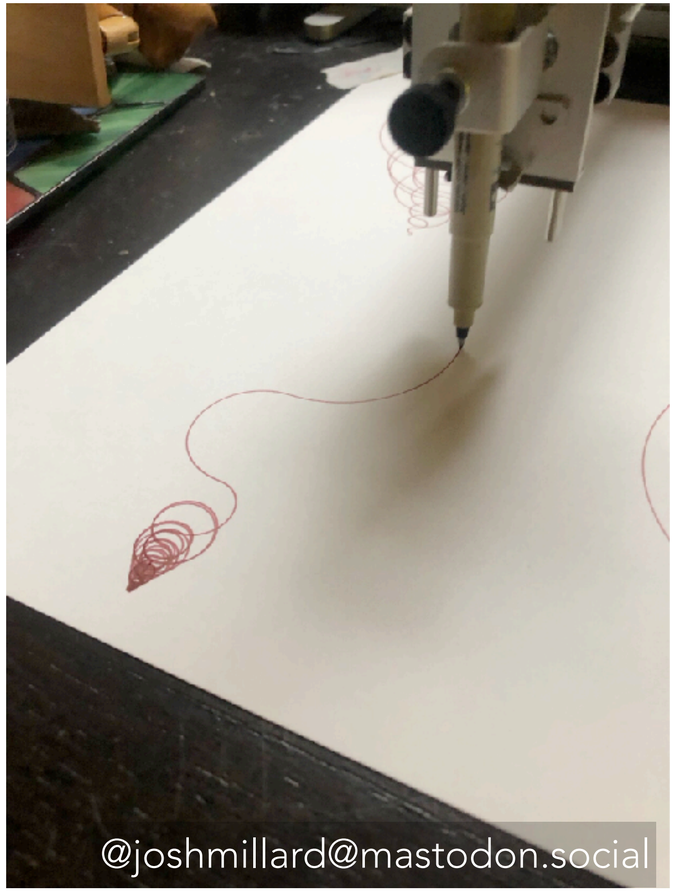
2. then we rasterize the 2D triangle!

So it makes sense to study rasterization of 2D graphics first.

Aside: Vector displays



Oscilloscopes

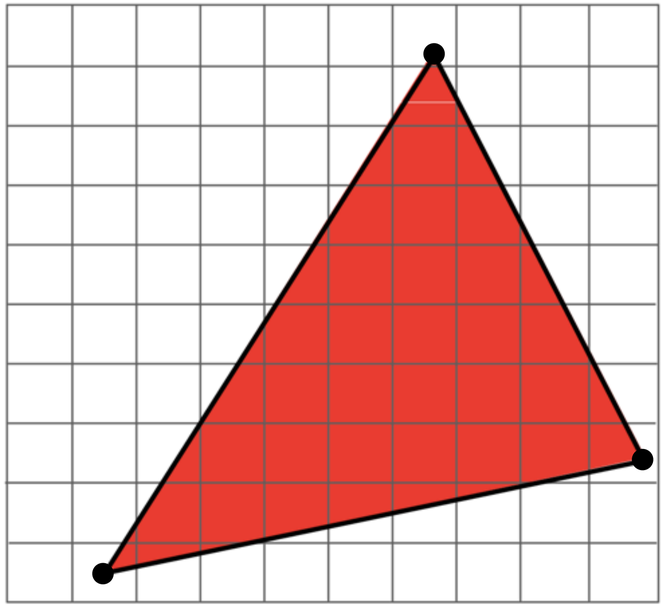


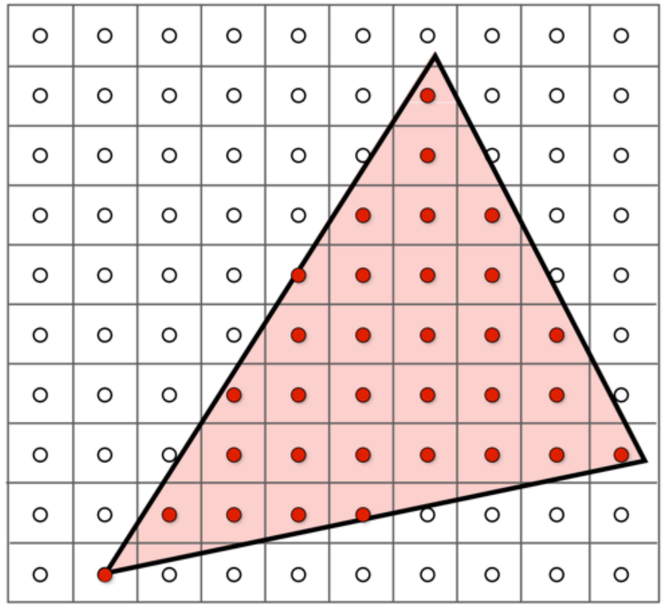
Plotters

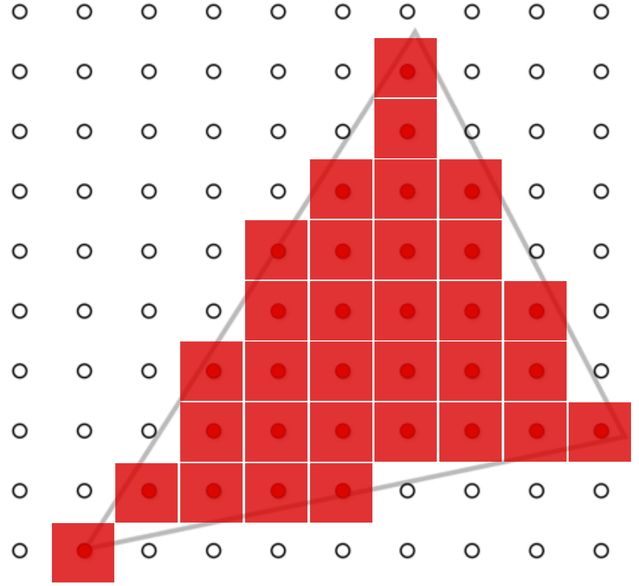
Laser light shows

How to draw an arbitrary triangle on a pixel grid?

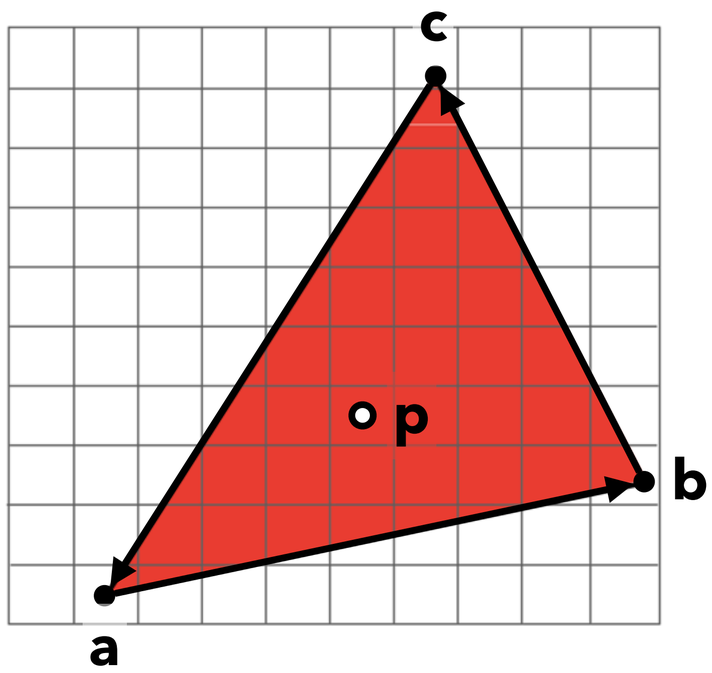
For now, let's pick a sample point at the center of each pixel, and colour the pixel if the sample point lies inside the triangle.





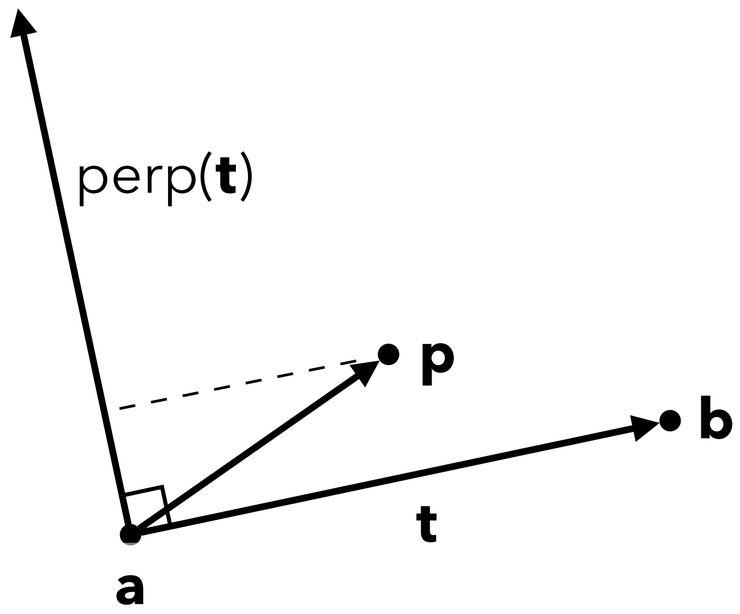


How to check whether a point is inside a triangle?



A point **p** is inside triangle **abc** if:

- **p** is to the left of edge **ab**, and
- **p** is to the left of edge **bc**, and
- **p** is to the left of edge **ca**.



Edge tangent vector:

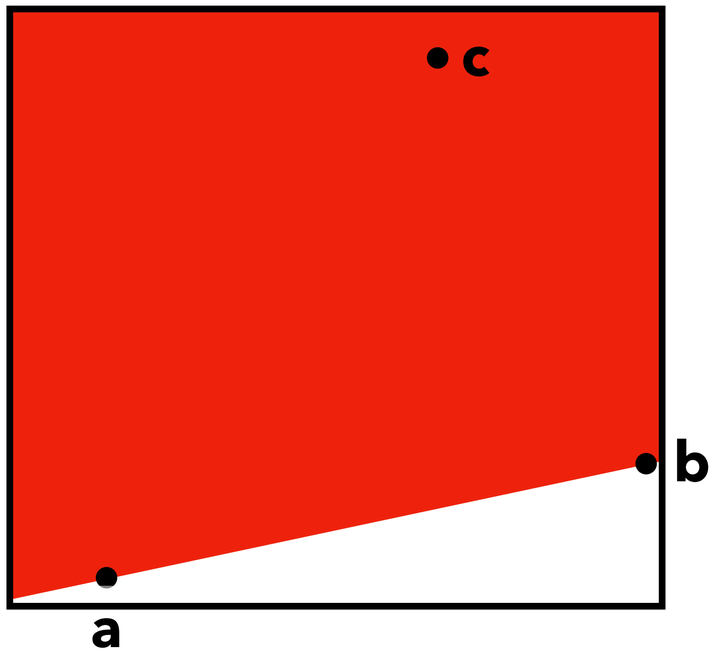
$$\mathbf{t} = \mathbf{b} - \mathbf{a} = (b_x - a_x, b_y - a_y)$$

Edge “normal” vector:

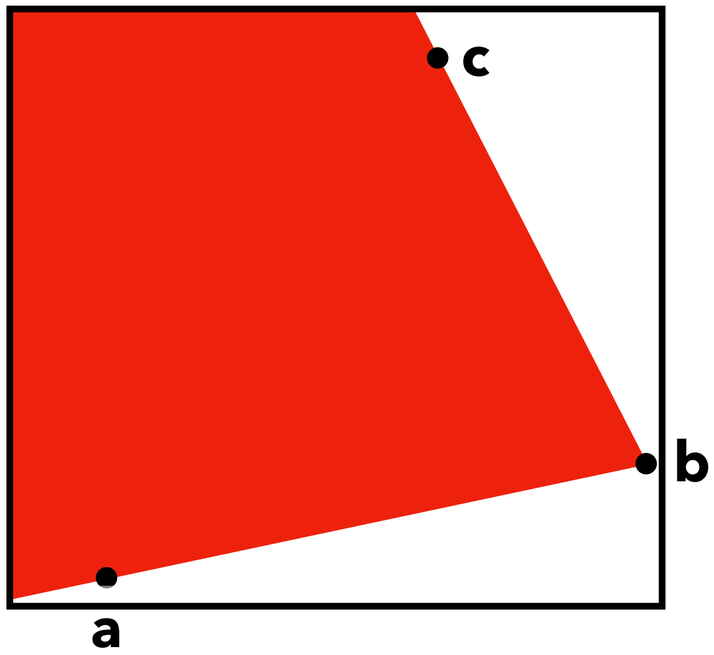
$$\mathbf{n} = \text{perp}(\mathbf{t}) = (-t_y, t_x)$$

p is to the left of **ab** if

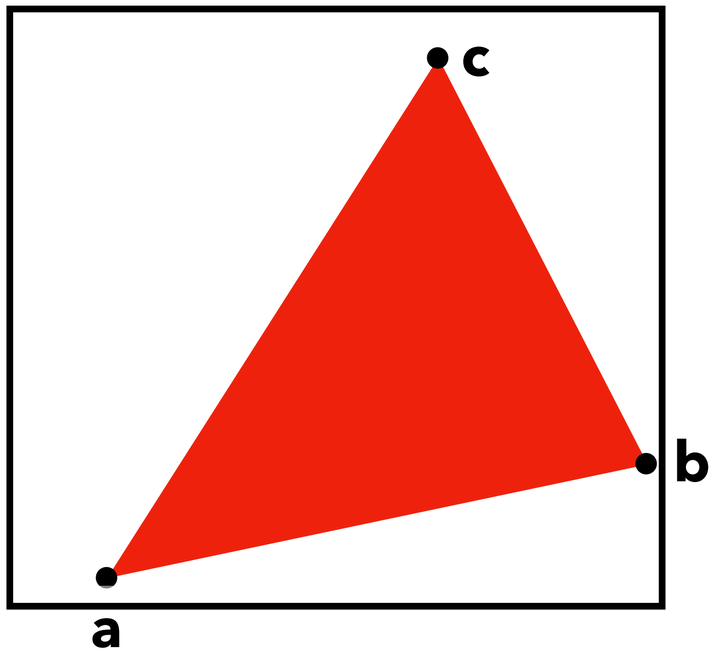
$$\mathbf{n} \cdot (\mathbf{p} - \mathbf{a}) \geq 0$$



Points to the left of **ab**

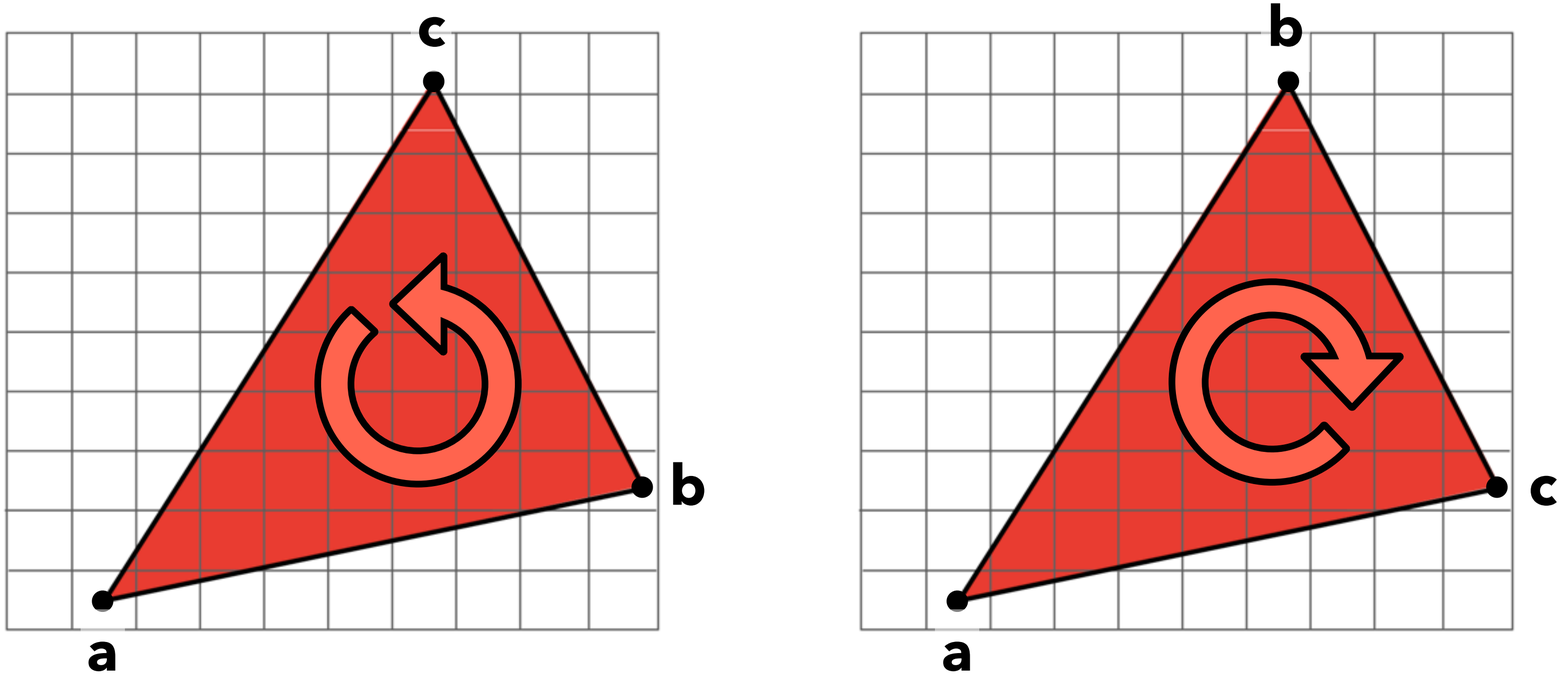


Points to the left of **ab**
and to the left of **bc**



Points to the left of **ab**
and to the left of **bc**
and to the left of **ca**

Would this still work if the vertices were given in clockwise order instead?

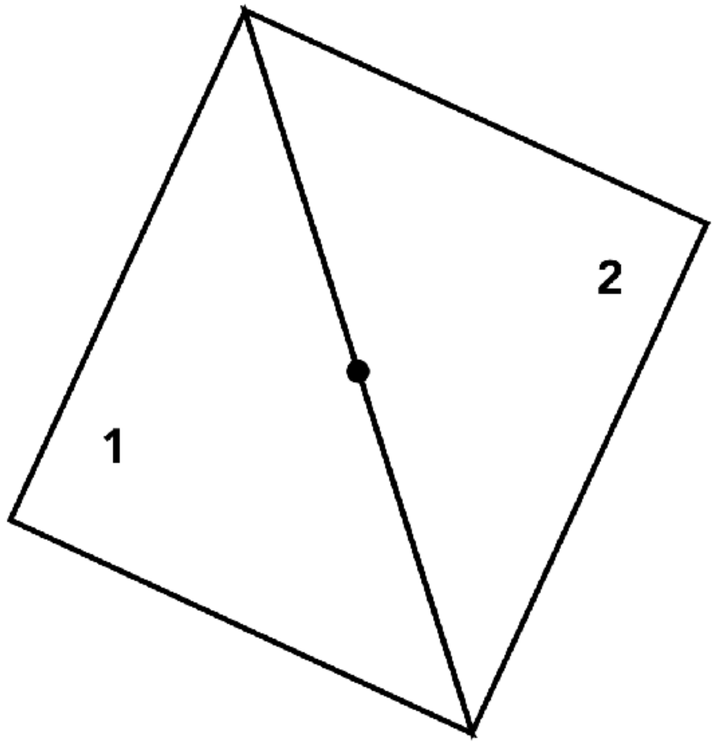


Easy to fix: First check if **c** is to the left or the right of **ab**.
But, better if you ensure all triangles are anticlockwise in the first place.

This is an issue of orientation! We will see more of these in this course…

Edge cases (literally)

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

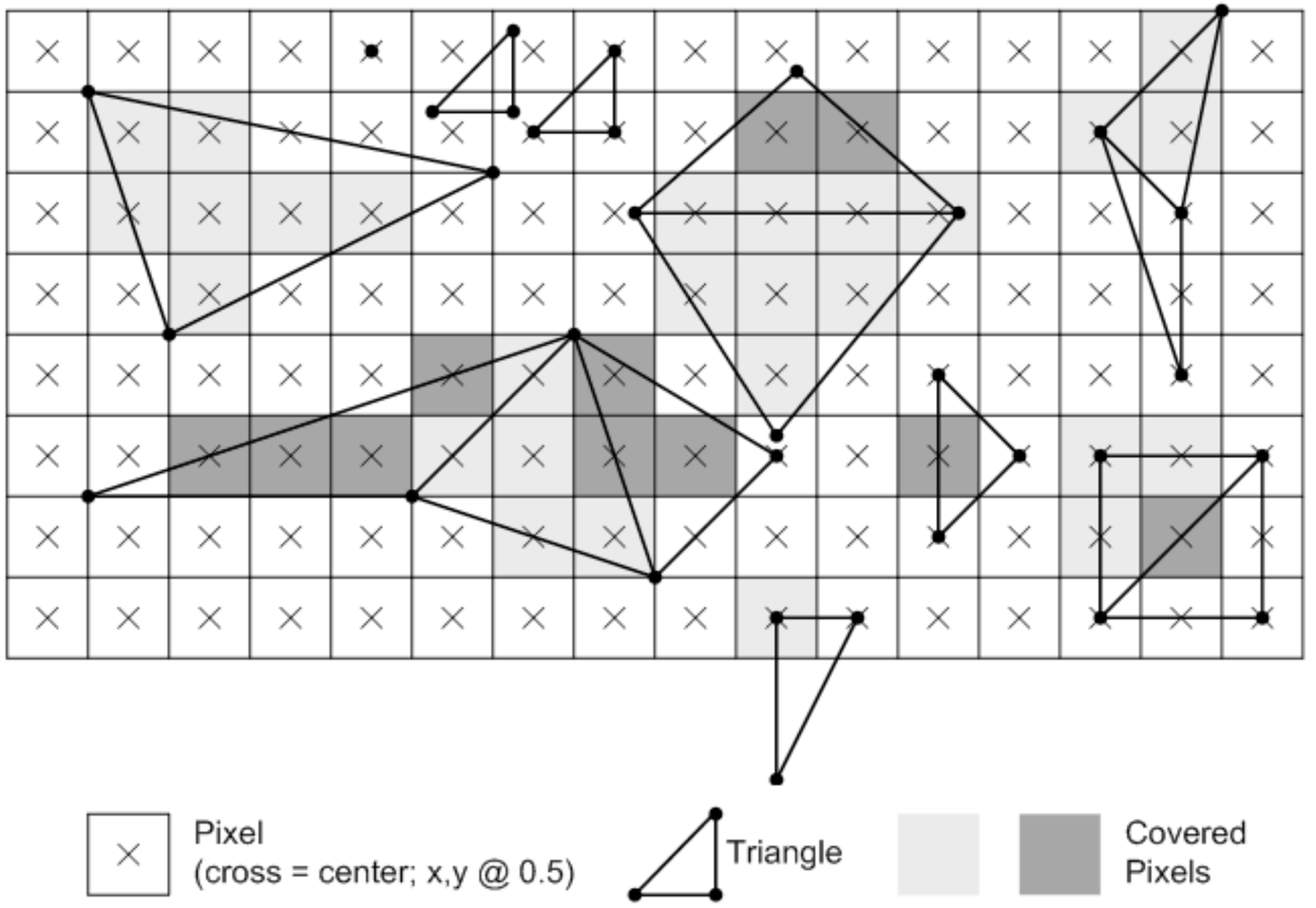


Most common edge rules in modern GPUs:

A sample point lying on a triangle boundary is classified as “covered” if each incident edge is a

- “top edge”: horizontal & above all other edges, or
- “left edge”: not exactly horizontal and on left side of triangle.

More importantly: no gaps, no over-draw between adjacent triangles



From Microsoft’s *Programming Guide for Direct3D 11*

So, here's what our rasterization algorithm looks like so far.

```
drawTriangle(triangle, colour):
  for x = 0 ... imageWidth:
    for y = 0 ... imageHeight:
      if isInside(x, y, triangle):
        image[x, y] = colour
```

Is this an efficient algorithm?

How can we make it faster?

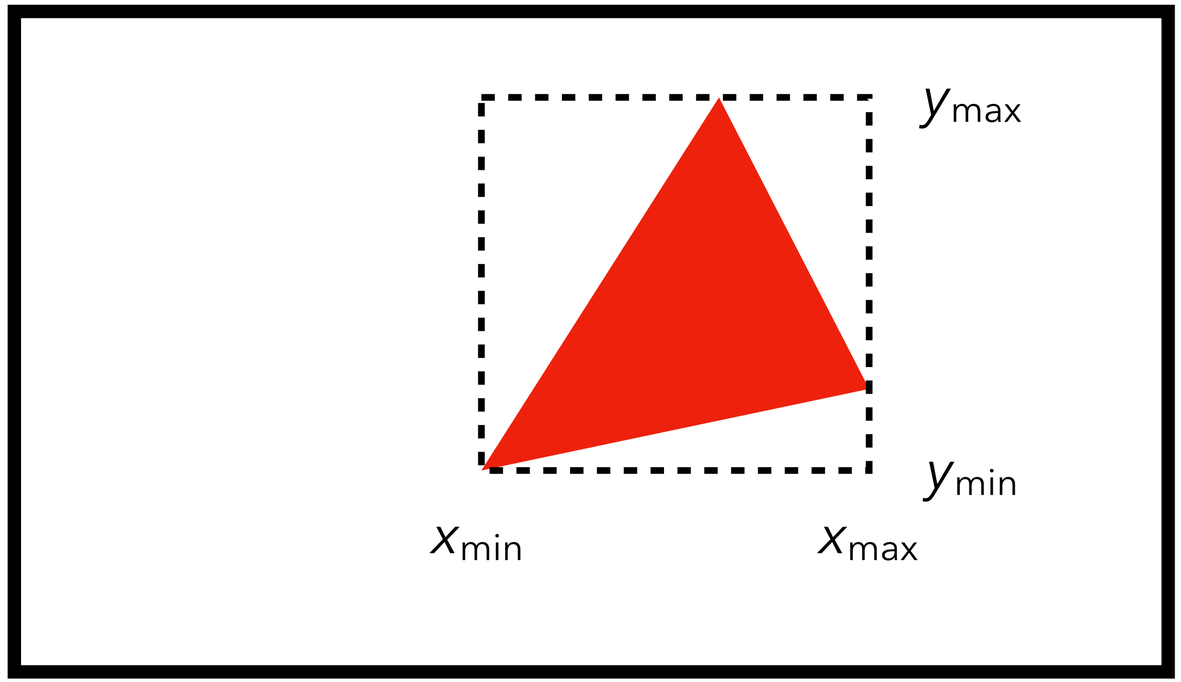
Better to only check the pixels in the bounding box of the triangle.

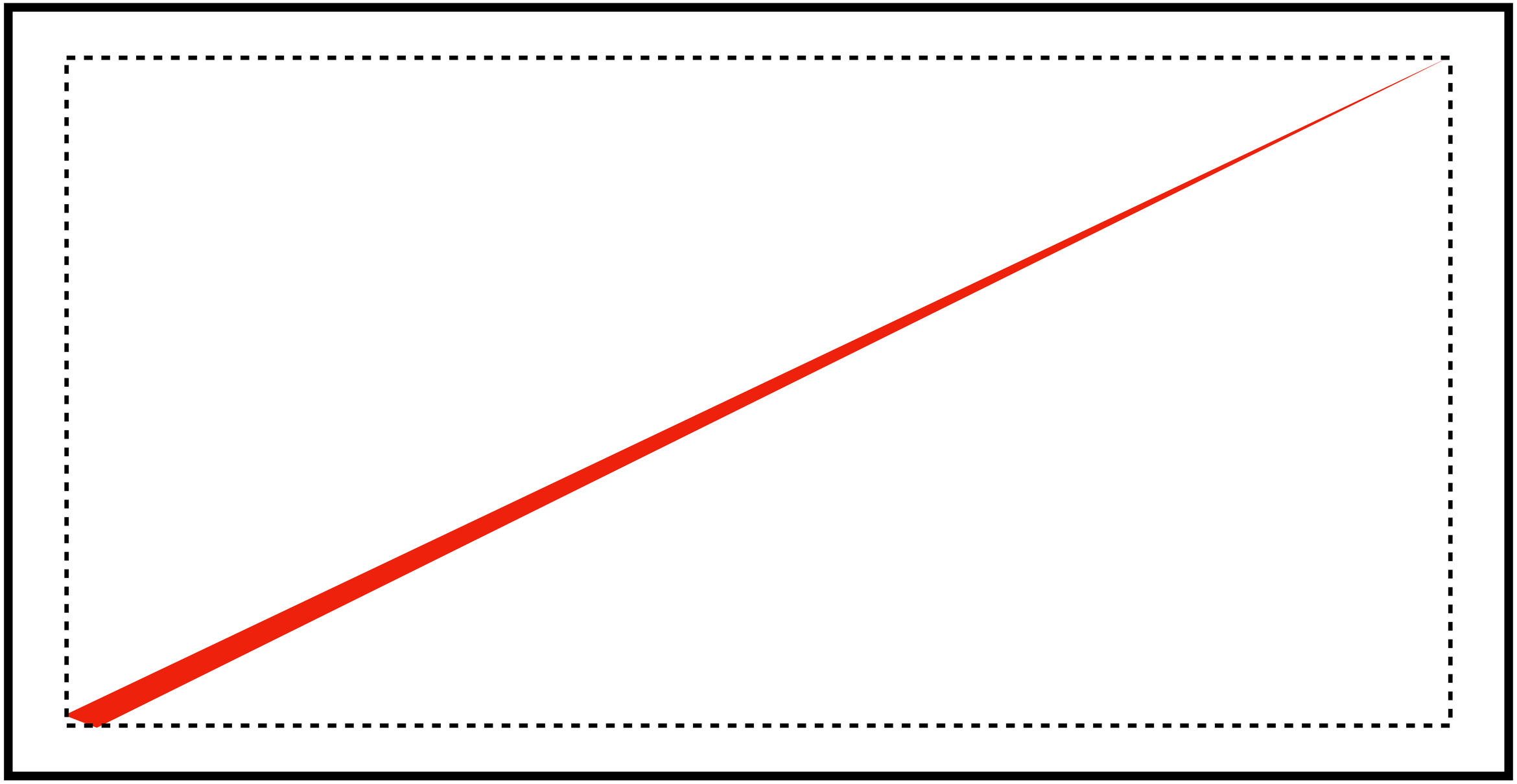
What are the coordinates of this box?

$x_{\min} = \min\{a_x, b_x, c_x\}$,

…

Are there any cases where this is also terribly inefficient?



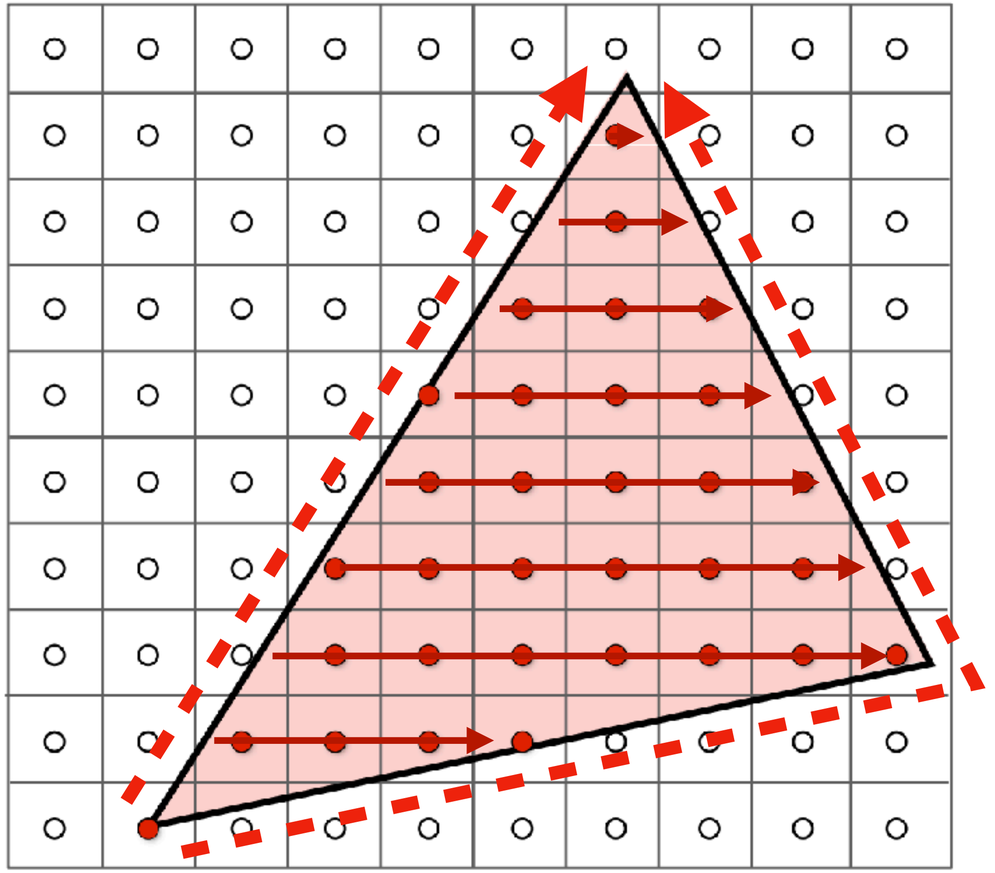


Incremental traversal

It’s possible to enumerate only the pixels actually covered by the triangle.

Roughly:

- Proceed row by row
- Keep track of where edges intersect row
- Fill in all pixels in between!



Which is better?

Incremental traversal does much less arithmetic…

- …but is also inherently serial
- Works well for “software rendering” i.e. on the CPU

GPUs are highly parallel SIMD (single instruction multiple data) hardware

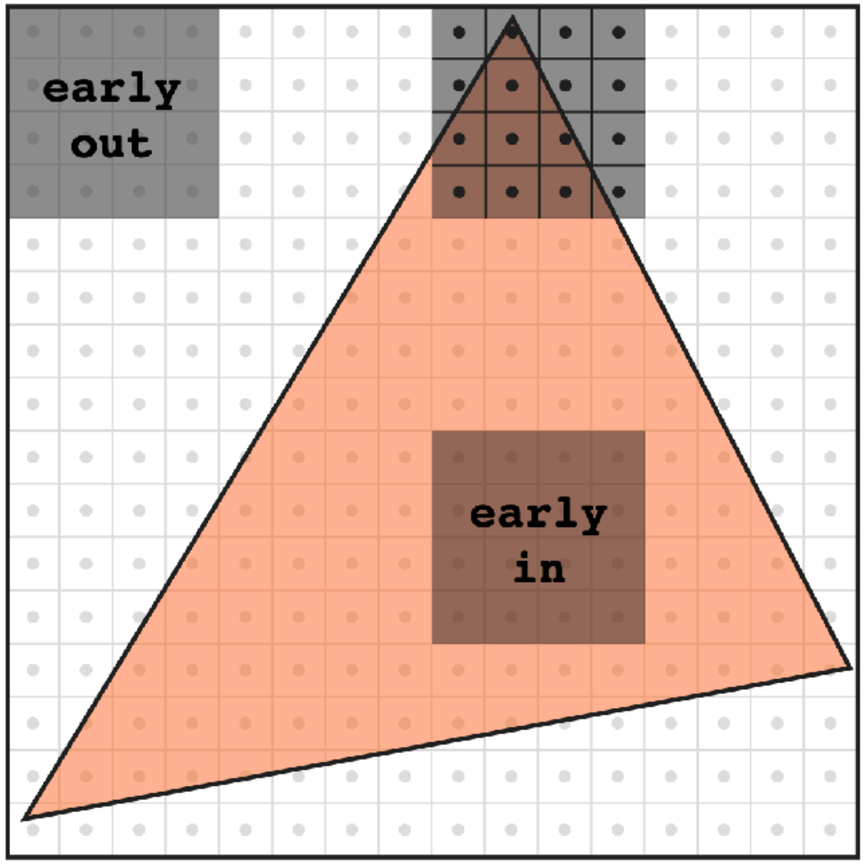
- Great for running the same tests in parallel on lots of pixels!

Tiled triangle traversal

Traverse the bounding box in blocks of pixels:

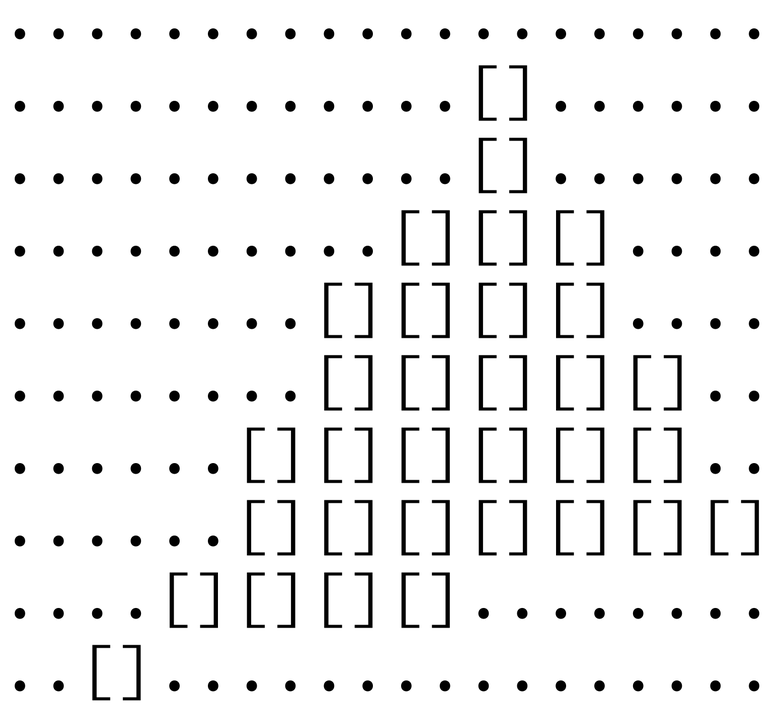
- If the block is entirely outside, just skip it (“early out”)
- If the block is entirely inside, process all the samples without testing them (“early in”)
- Otherwise, test each sample point in parallel

All modern GPUs have specialized hardware for doing this very very efficiently!



Homework to try

Write a simple program to rasterize a triangle with given vertex coordinates into an array of booleans. “Display” it by printing it out, maybe like this:



Acknowledgements

This lecture’s slides are heavily based on those of Ren Ng and Keenan Crane.