SCAN CONVERSION

2011 Introduction to Graphics

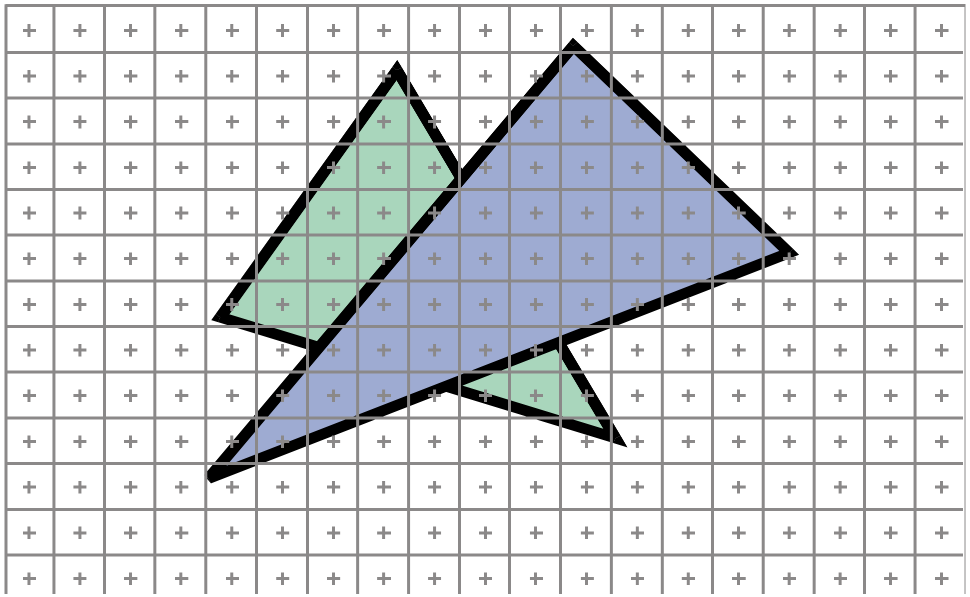
Lecture 9

Overview

- Recap
  - Inside/Outside a Polygon
- Naïve Filling Algorithm
- Active Edge Tables
  - Exploiting coherence
- Brute Force Rasterization
  - Half-Space Test

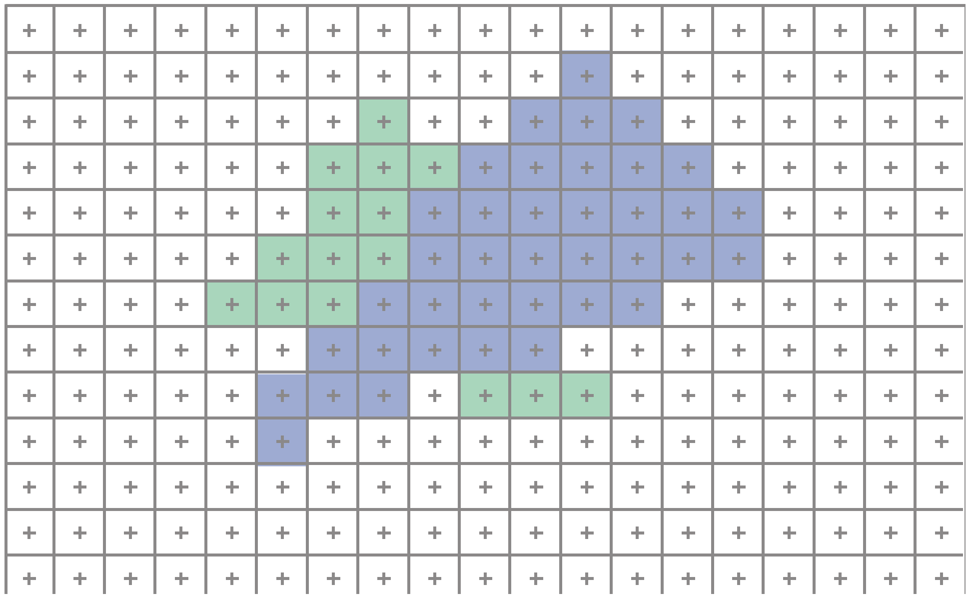
2D Scan Conversion

- Primitives are continuous; screen is discrete
  - Well, triangles are described by a discrete set of vertices
  - But they describe a continuous area on screen



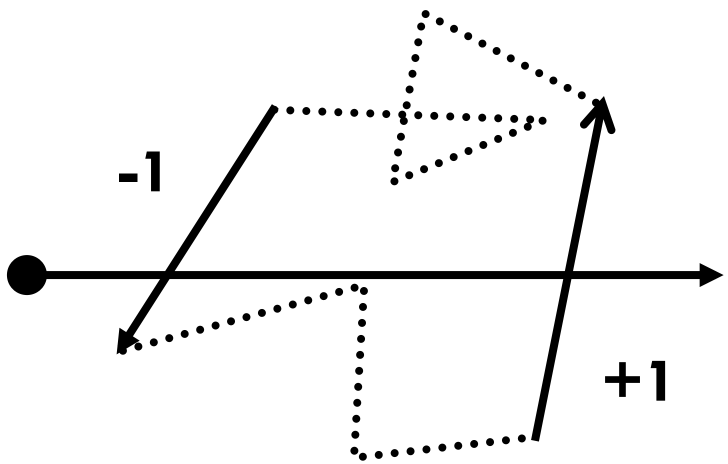
2D Scan Conversion

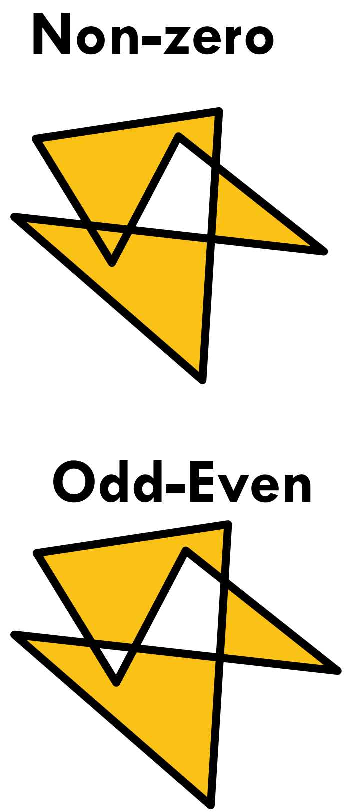
- Solution: compute discrete approximation
- Scan Conversion (Rasterization):
  algorithms for efficient generation of the samples comprising this approximation



Recap - Inside/Outside

- Draw a line from the test point to the outside
  - +1 if you cross anti-clockwise
  - -1 if you cross clockwise





Naïve Algorithm

- Find a point inside the polygon
- Do a **flood fill:**
  - Keep a stack of points to be tested
  - When the stack none empty
    - Pop the top point (Q)
    - Test if Q is inside or outside
      - If Inside, colour Q, push neighbours of Q if not already tested
      - It outside discard

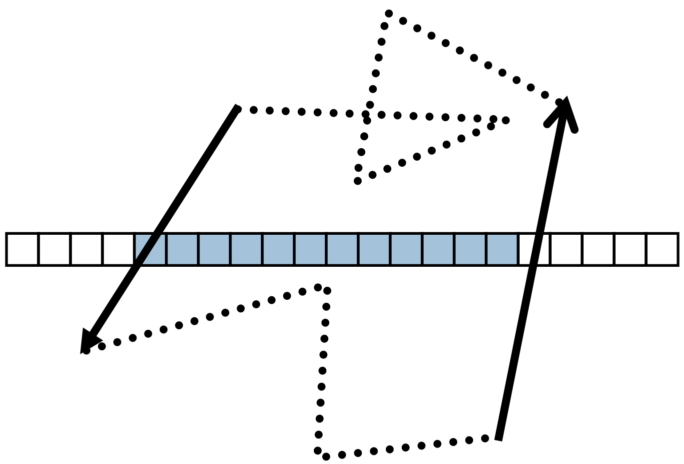
Critique

- Horribly slow
  - But still very common in paint packages!
- Stack might be very deep

- Need to exploit **TWO** types of coherency
  - Point coherency
  - Scan-line coherency

Point Coherency

- Ray shooting is fast, but note that for every point on one scan line the intersection points are the same
- Why not find the actual span for each line from the intersection points?



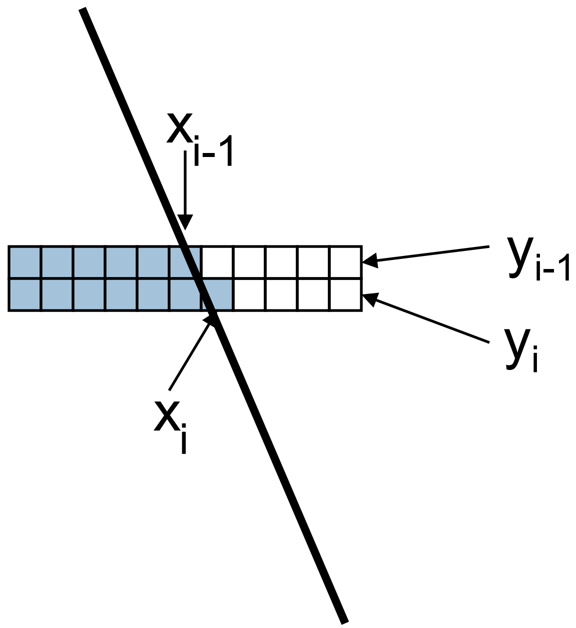
Scan-Line Coherency

- Intersection points of polygon edges with scan lines change little on a line by line basis

$$y_i = a + bx_i$$

$$y_{i-1} = a + bx_{i-1}$$

$$x_i = x_{i-1} + \frac{1}{b}$$



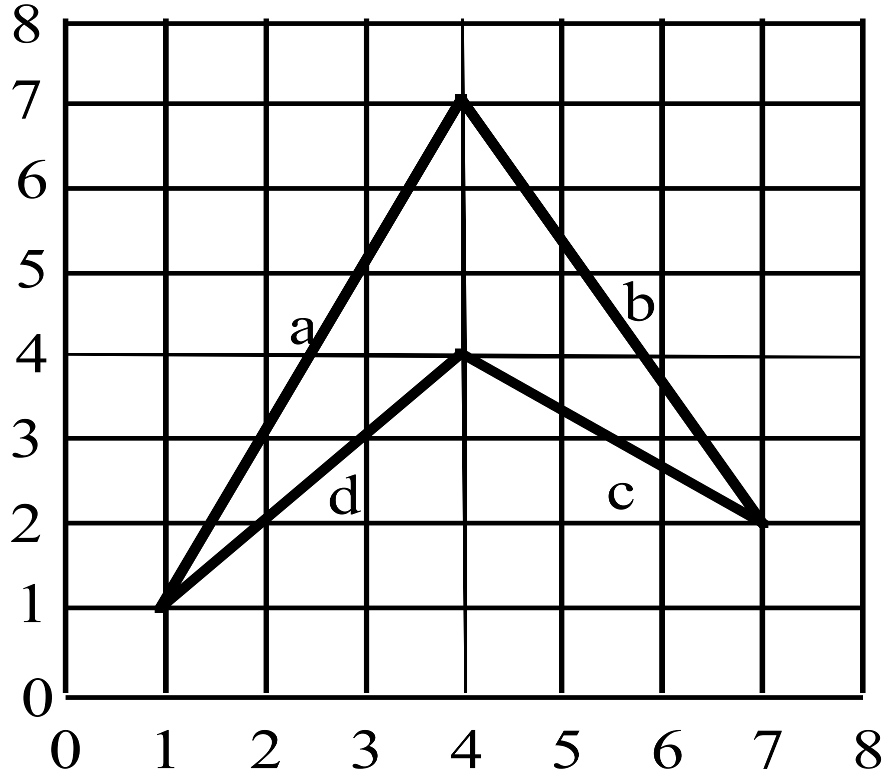
Overview of Active Edge Table

- For each scan-line in a polygon only certain edges need considering
- Keep an **active edge table (AET)**
  - Initialize the AET with details for first scan-line
  - Update this edge table based upon the vertical extent of the edges
  - From the AET extract the required spans

Setting Up

- “fix” edges
  - make sure $y_1 < y_2$ for each (x_1, y_1) (x_2, y_2)
- Form an ET
  - Bucket sort all edges on minimum y value
  - 1 bucket might contain several edges
  - Each edge element contains
    - (max Y, start X, X increment)
    - X increment = (x2-x1)/(y2-y1)

Example



Setup

- Edges are

Edge Label	Coordinates	y1	Structure
a	(1,1) to (4,7)	1	(7,1,0.5)
b	(7,2) to (4,7)	2	(7,7,-0.6)
c	(7,2) to (4,4)	2	(4,7,-1.5)
d	(1,1) to (4,4)	1	(4,1,1)

- Edge Table Contains

y1	Sequence of Edges
1	(7,1,0.5), (4, 1, 1)
2	(7,7,-0.6), (4, 7,-1.5)

Maintaining the AET

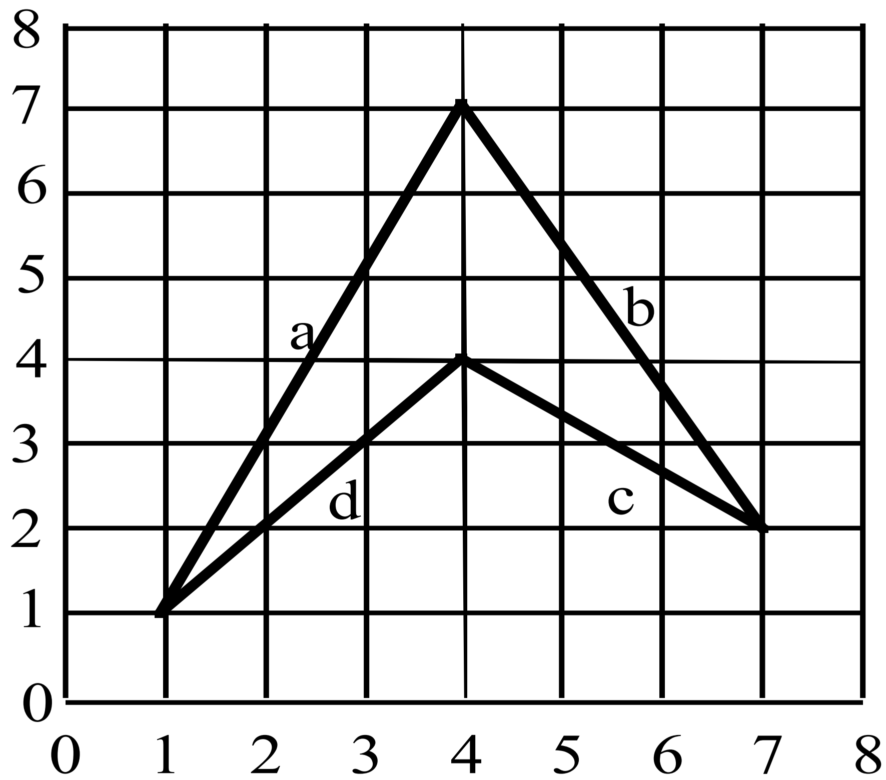
- For each scan line
  - Remove all edges whose y_2 is equal to current line
  - Update the x value for each remaining edge
  - Add all edges whose y_1 is equal to current line

Drawing the AET

- Sort the active edges on x intersection
- Pairs of edges are the spans we require

- Caveats (discussed in the notes)
  - Don’t consider horizontal lines
  - Maximum vertices are not drawn
  - Plenty of special cases when polygons share edges

Example



On Each Line

Line	Active Edge Table	Spans
0	empty	
1	(7,1,0.5), (4,1,1)	1 to 1
2	(7,1.5,0.5), (4,2,1), (4,7,-1.5), (7,7,-0.6)	1.5 to 2, 7 to 7
3	(7,2.0,0.5), (4,3,1), (4,5.5,-1.5), (7,6.4,-0.6)	2.0 to 3, 5.5 to 6.4
4	(7,2.5,0.5), (7,5.8,-0.6)	2.5 to 5.8
5	(7,3.0,0.5), (7,5.2,-0.6)	3.0 to 5.2
6	(7,3.5,0.5), (7,4.6,-0.6)	3.5 to 4.6
7	empty	
8	empty	

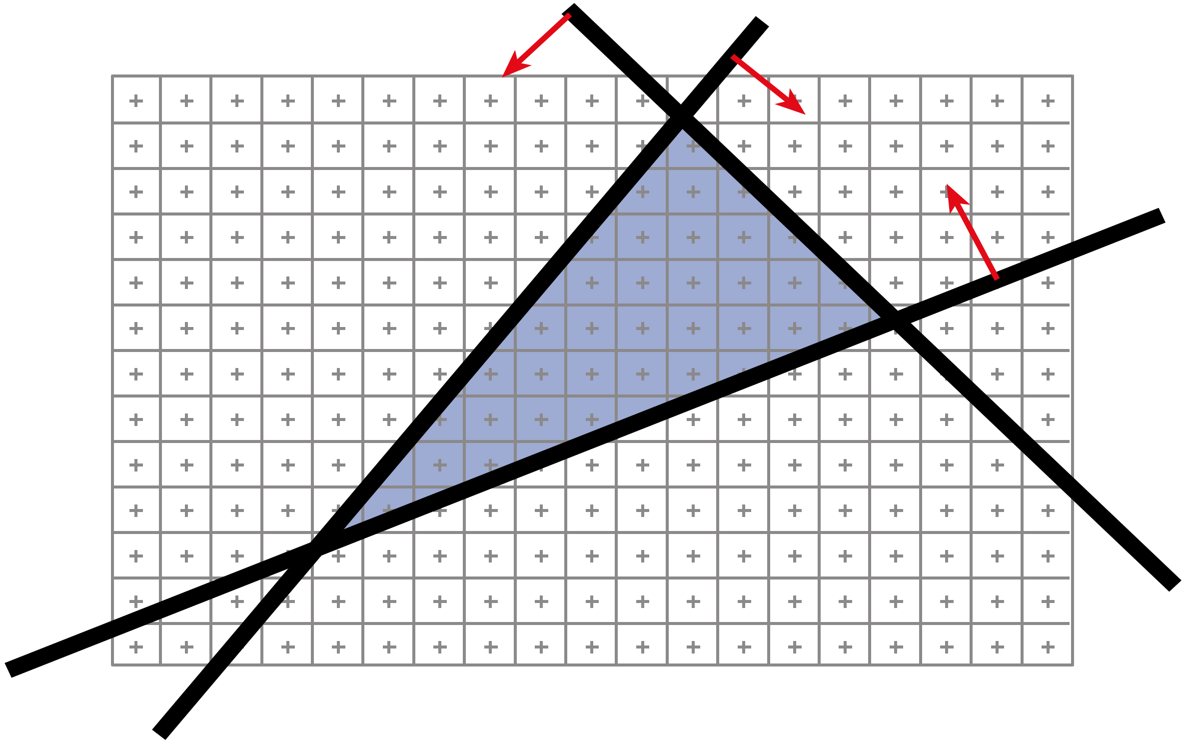
Is this really done in practise?

- Modern rasterisation works quite differently
- Reason:
  - GPU implementation of AET is very tricky
  - **Triangles** are a special case
    - Do not need generality of AET

- Start with a brute-force method and improve it…

Brute Force Solution for Triangles

- For *each* pixel
  - Compute line equations (half-space test) at pixel center
  - "clip" against the triangle



Half-Space Test

- For each edge compute line equation:

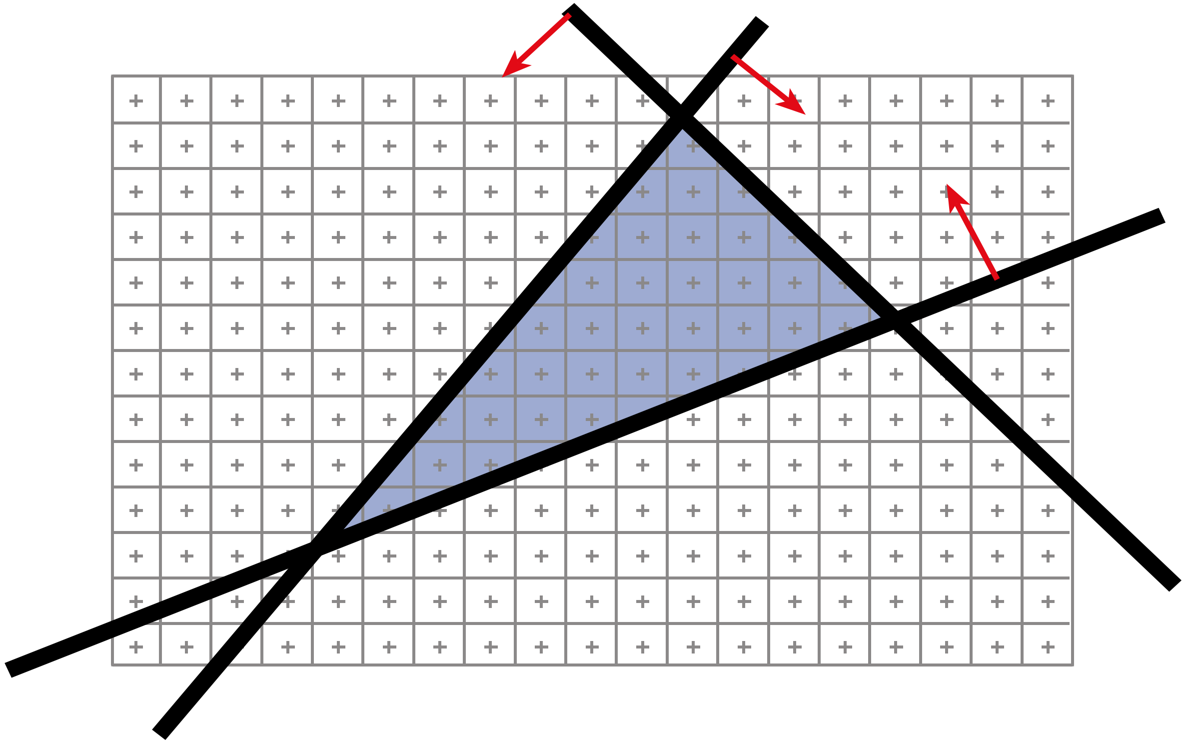
$$L_i(x, y) = a_i x + b_i y + c_i$$

- If $L_i(x,y) > 0$
  - point in **<u>positive</u>** half-space
- If $L_i(x,y) < 0$
  - point in **<u>negative</u>** half-space

- If all $L_{1,2,3}(x,y) >= 0$
  - Point (x,y) is inside triangle!

Brute Force Solution for Triangles

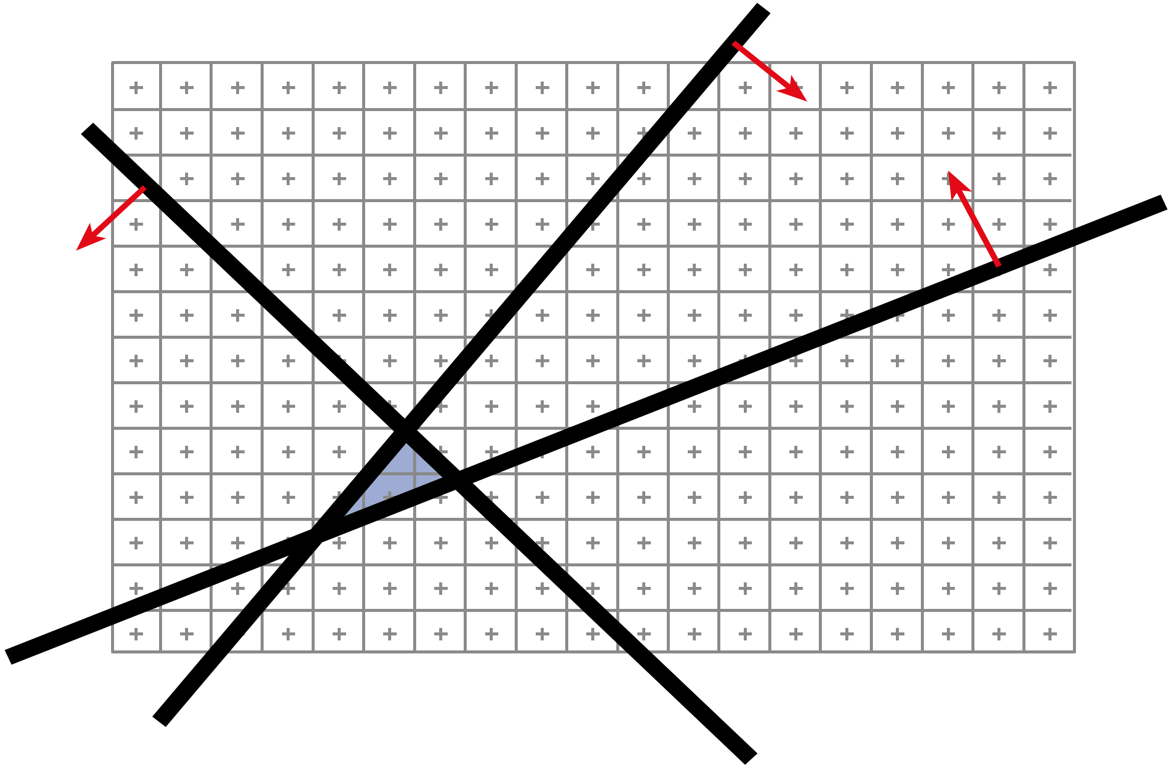
- For *each* pixel
  - Compute line equations at pixel center
  - "clip" against the triangle



Problem?

Brute Force Solution for Triangles

- For each pixel
  - Compute line equations at pixel center
  - “clip” against the triangle

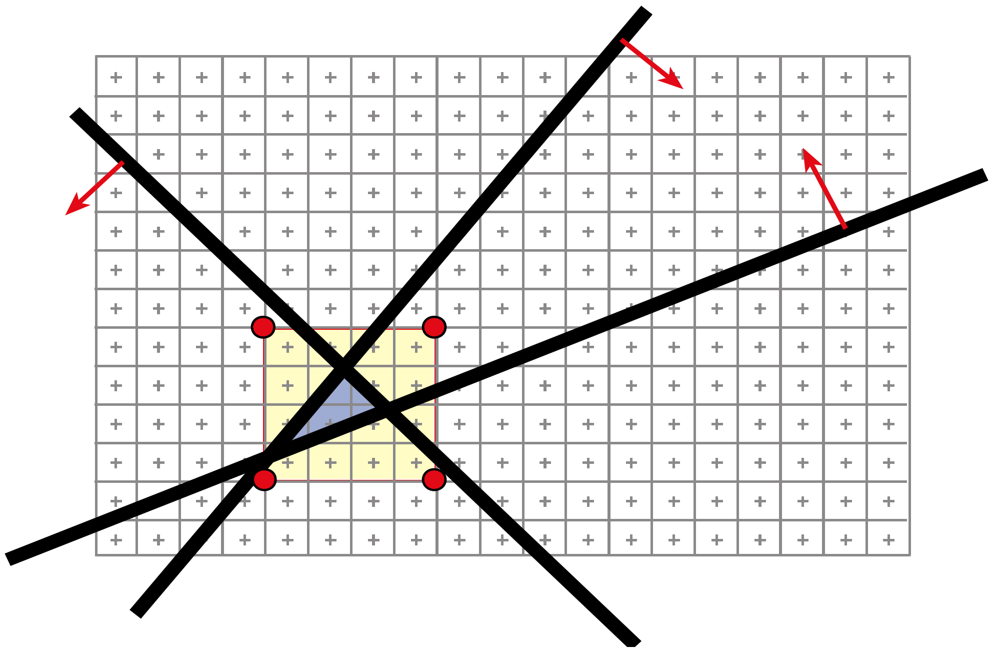


Problem?

If the triangle is small, a lot of useless computation

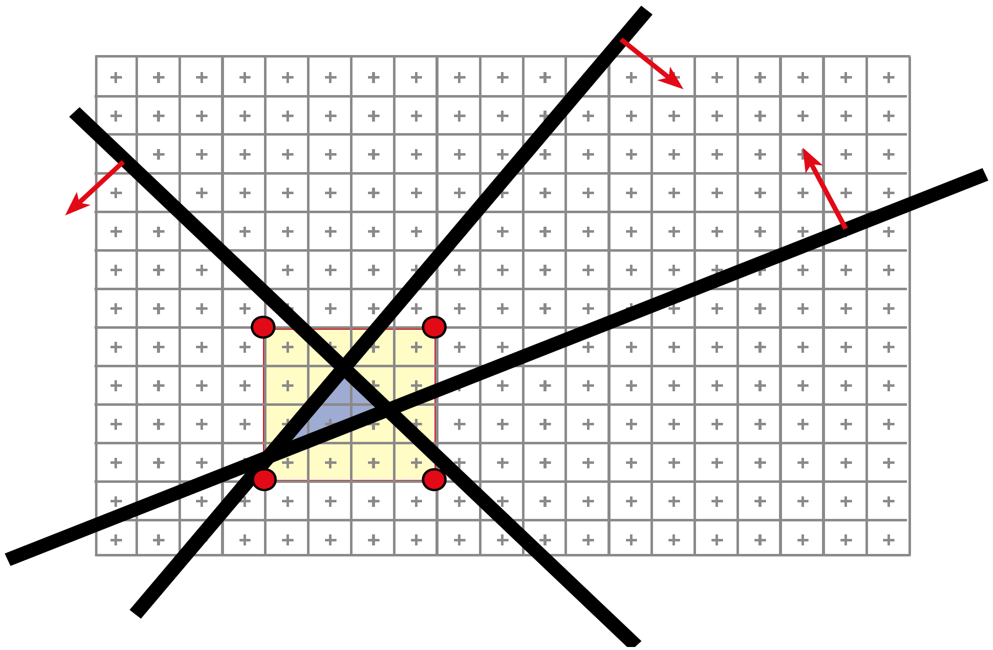
Brute Force Solution for Triangles

- Improvement: Compute only for the *screen bounding box* of the triangle
- How do we get such a bounding box?
  - Xmin, Xmax, Ymin, Ymax of the triangle vertices



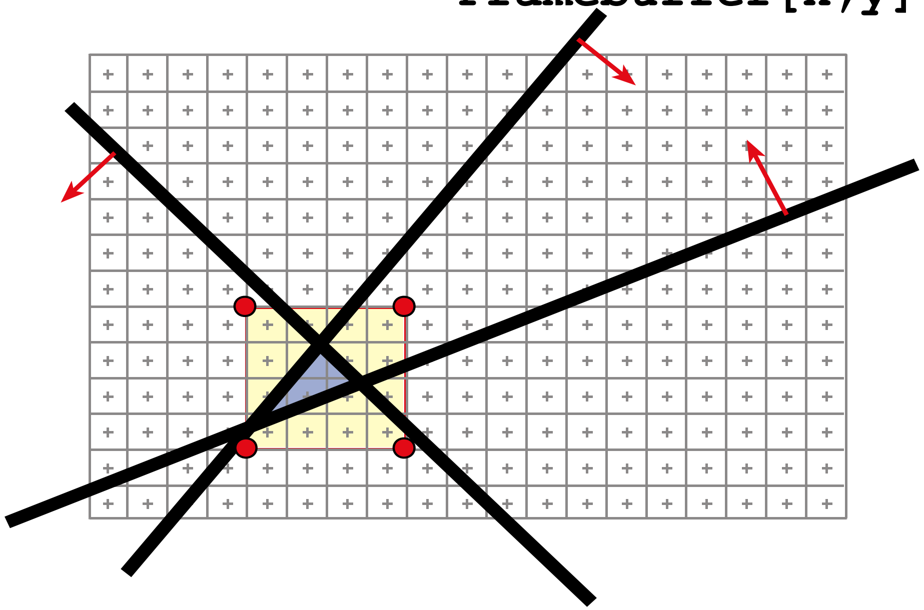
Rasterisation on Graphics Cards

- Triangles are usually very small
  - Setup cost are becoming more troublesome
- Clipping is annoying
- Brute force is tractable



Rasterisation on Graphics Cards

```
For every triangle
   ComputeProjection
   Compute bbox, clip bbox to screen limits
   For all pixels in bbox
      Compute line equations
      If all line equations>0  //pixel [x,y] in triangle
                Framebuffer[x,y]=triangleColor
```



Summary

- We have developed the Active Edge Table algorithm
  - Exploits coherency in two directions
- AET has many applications
- AET is an important algorithm in 2D and 3D graphics
- Brute force is viable alternative