Lecture #3b

Rasterization II

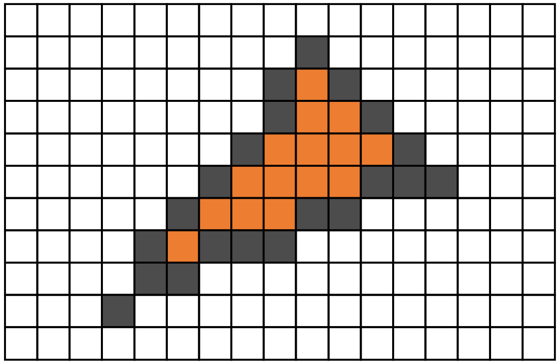
Computer Graphics

Winter Term 2016/17

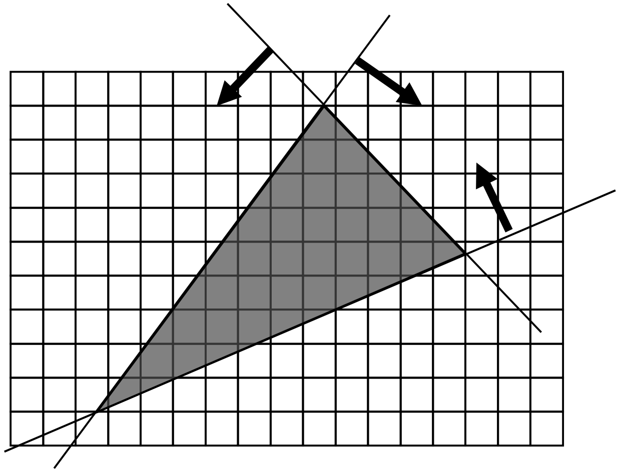
Marc Stamminger / Roberto Grosso

Previous Lecture

- Fill objects using **seed fill** → **recursion**

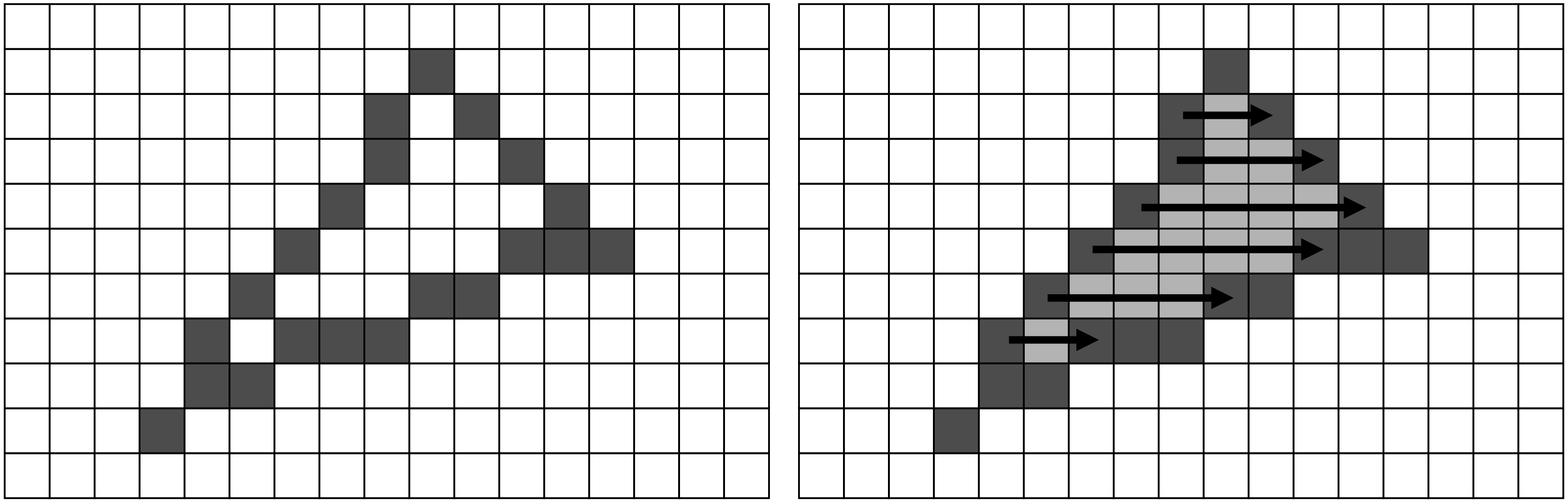


- or by testing pixels → 2D iteration → nested loop



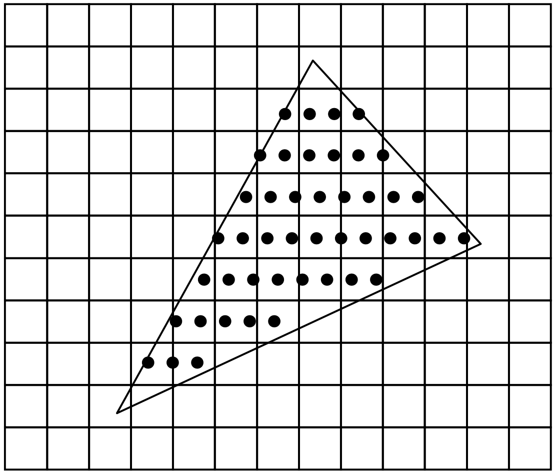
Polygon Rasterization

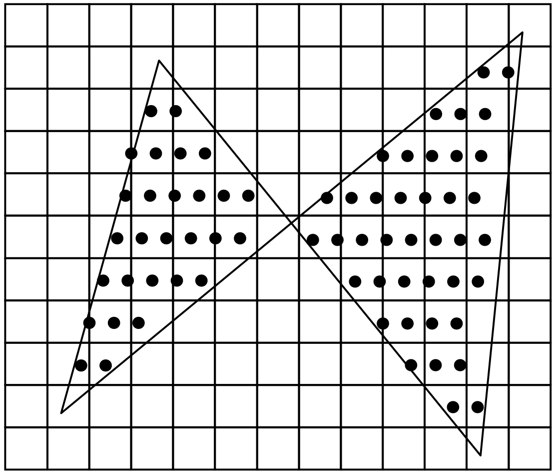
- Can we do better? – Yes!
  - Using line rasterization



Scanline Algorithm

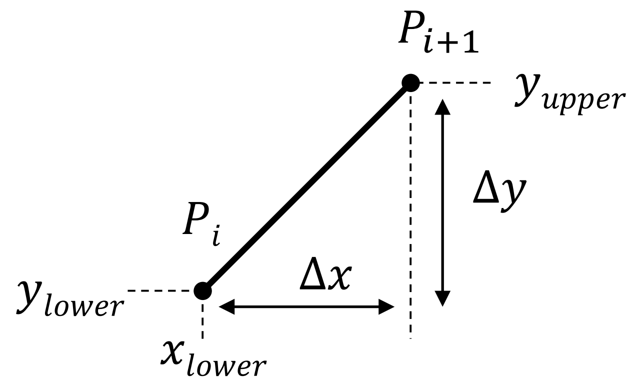
- Idea Scanline Algorithm
  - Proceed scanline by scanline from bottom to top
  - Find intersections of scanline with polygon
  - Fill this intersection





Scanline Algorithm

- Data Structures
  - Edge table (ET)
    - List of all polygon edges (upwards only!)
    - Content per edge
    - Linked list
    - Sorted by ylower
  - Note that 1/m is the x-increment when stepping to above scanline

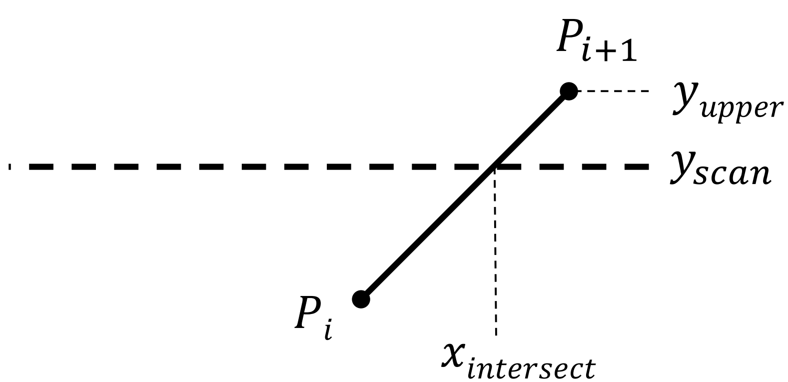


y_{lower}	x_{lower}	y_{upper}	$1/m = \Delta x/\Delta y$	→ next

Scanline Algorithm

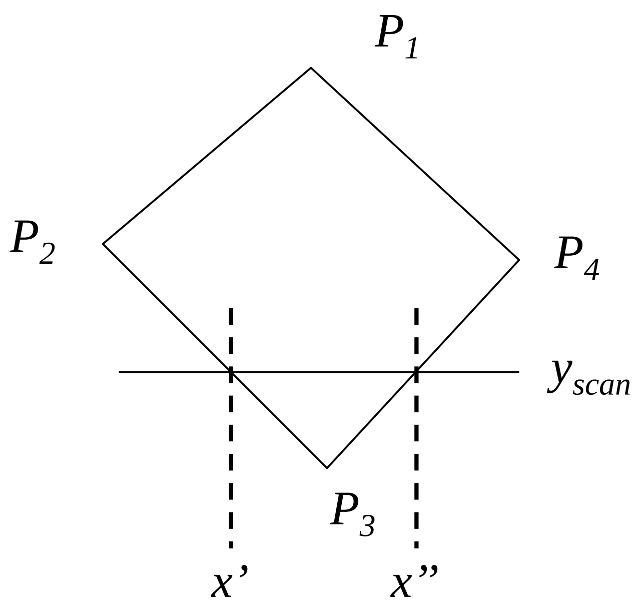
- Active Edge table (AET)
  - All edges from ET that intersect current scanline
  - Data per edge
  - Current scanline of y_{scan}
  - Current intersection of edge with scanline: $x_{intersect}, y_{scan}$
  - Sorted by $x_{intersect}$

$x_{intersect}$	y_{upper}	$1/m = \Delta x/\Delta y$	next



Scanline Algorithm

- Example
  - Edge table

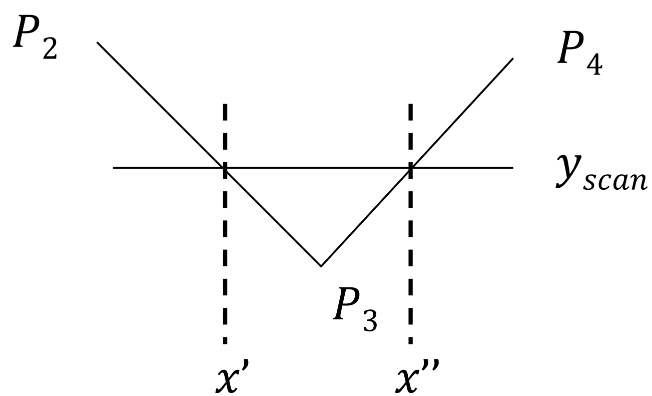


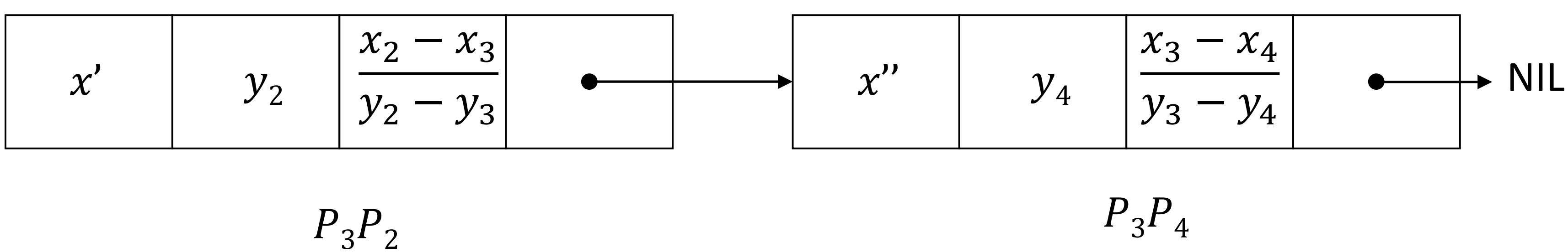
y_3	x_3	y_4	$\frac{x_4 - x_3}{y_4 - y_3}$	•	P_3P_4
y_3	x_3	y_2	$\frac{x_2 - x_3}{y_2 - y_3}$	•	P_3P_2
y_4	x_4	y_1	$\frac{x_1 - x_4}{y_1 - y_4}$	•	P_4P_1
y_2	x_2	y_1	$\frac{x_1 - x_2}{y_1 - y_2}$	•	P_2P_1

NIL

Scanline Algorithm

- Current scanline $y_{scan} \Rightarrow$ AET





Scanline Algorithm

- Remark on incrementing x
- $x_{old} = \frac{1}{m}(\mathrm{y_{scan}} - \mathrm{y_{lower}}) + \mathrm{x_{lower}}$
- $x_{new} = \frac{1}{m}(y_{scan} + 1 - y_{lower}) + x_{lower} = x_{old} + \frac{1}{m}$
- Where $m = \frac{y_{upper} - y_{lower}}{x_{upper} - x_{lower}}$
- So the update is $y \to y + 1, x \to x + \frac{1}{m}$

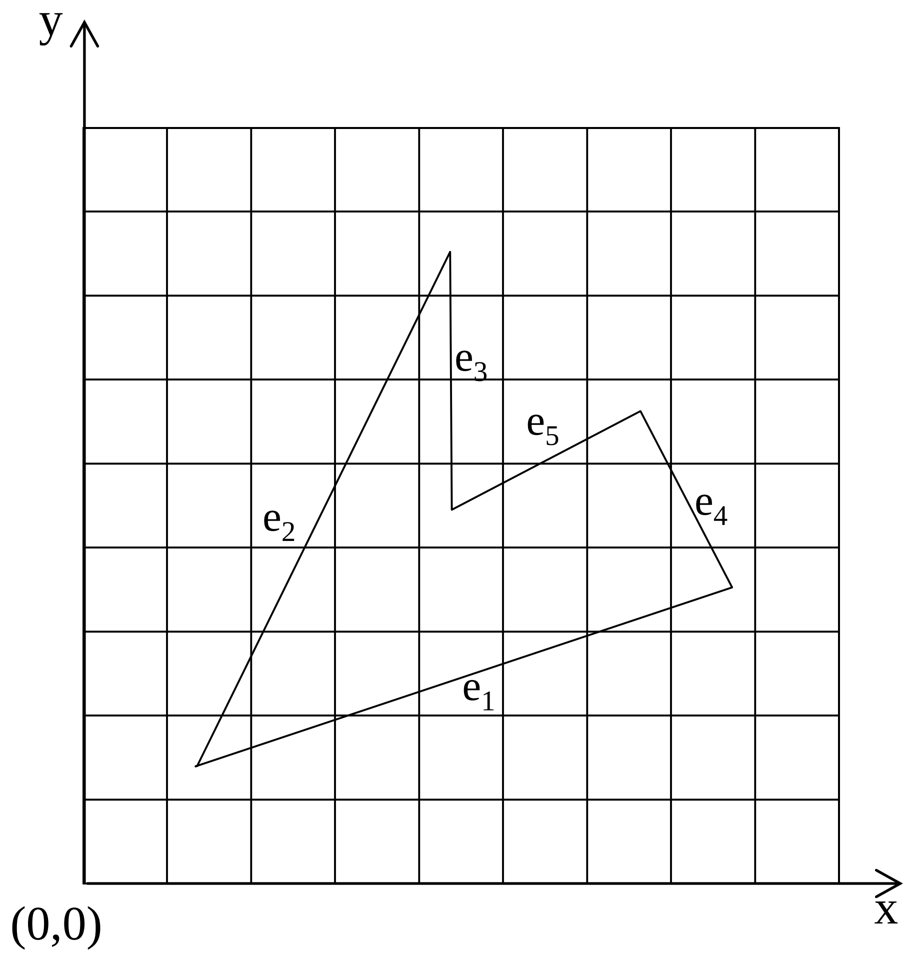
Scanline Algorithm

```
initialize ET
set AET to empty
set yscan to ylower of first entry in ET
    move all edges from ET with yscan == ylower to AET

while ET not empty or AET not empty
    sort AET for x
    draw lines from (AET[0].x,yscan) to (AET[1].x,yscan),
                   from (AET[2].x,yscan) to (AET[3].x,yscan), ……
    remove all edges from AET with yscan >= yupper
    for all edges in AET
        x:= x + 1/m
    move all edges from ET with yscan == ylower to AET
    yscan += 1
```

Scanline Algorithm

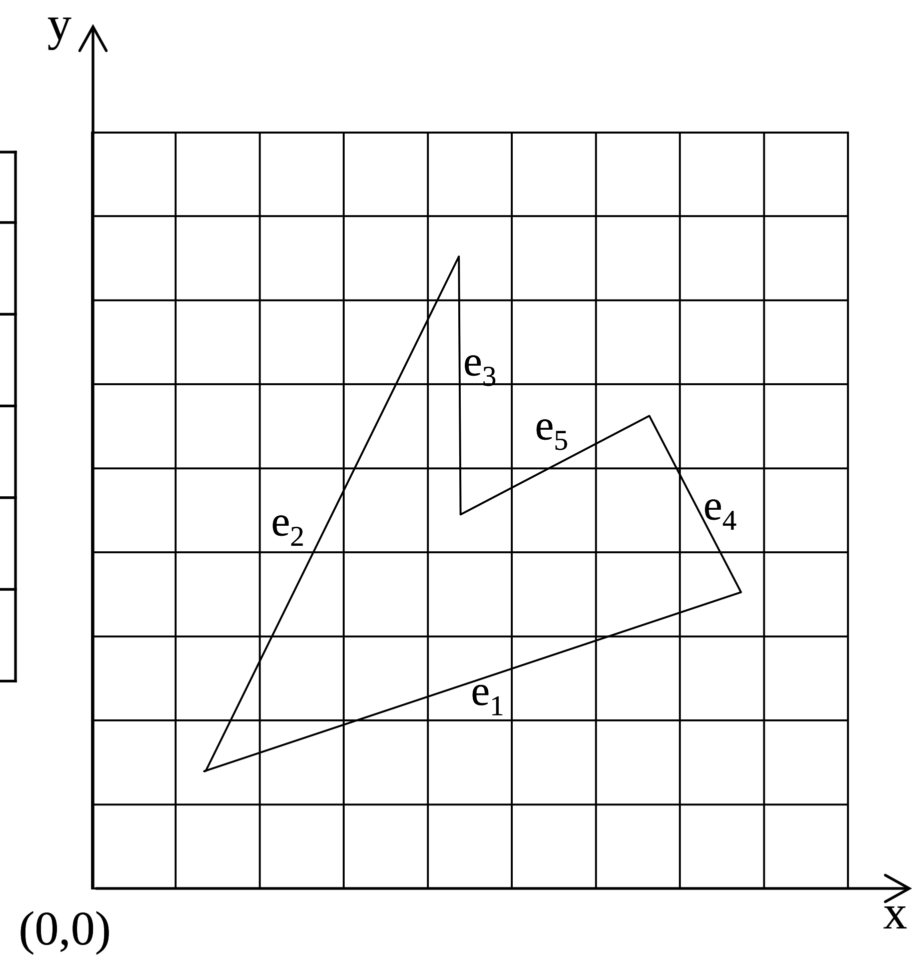
edge	y_{lower}	x_{lower}	y_{upper}	$1/m$
e_1	1	1	3	3
e_2	1	1	7	1 / 2
e_3	4	4	7	0
e_4	3	7	5	-3
e_5	4	4	5	2



Scanline Algorithm

ET: edge table, sorted on y_{lower}

edge	y_{lower}	x_{lower}	y_{upper}	$1/m$	Next
e_1	1	1	3	3	e_2
e_2	1	1	7	1 / 2	e_4
e_4	3	7	5	-3	e_3
e_3	4	4	7	0	e_5
e_5	4	4	5	2	NULL



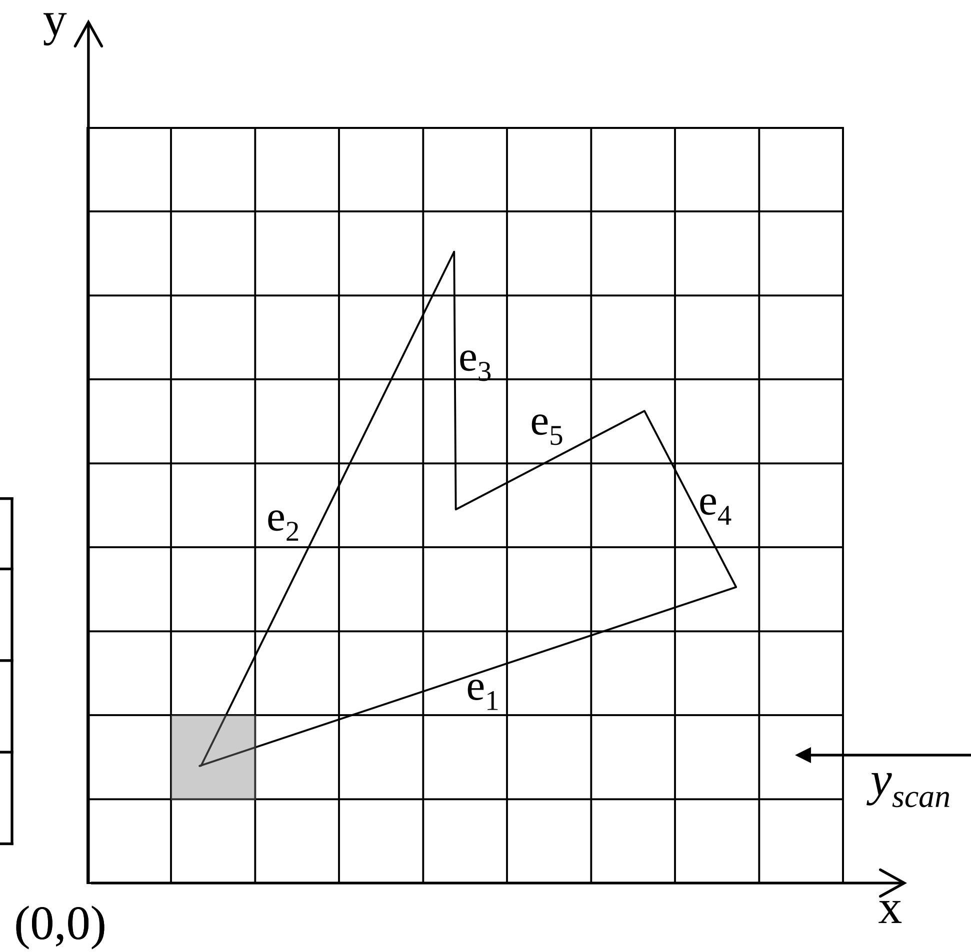
Scanline Algorithm

First scanline $y_{scan} = 1$
AET: edge table, sorted on $x_{intersect}$

edge	x_{inters}	y_{upper}	$1/m$	Next
e_1	1	3	3	e_2
e_2	1	7	1 / 2	NULL

ET: edge table, sorted on y_{lower}

edge	y_{lower}	x_{lower}	y_{upper}	$1/m$	Next
e_4	3	7	5	-3	e_3
e_3	4	4	7	0	e_5
e_5	4	4	5	2	NULL



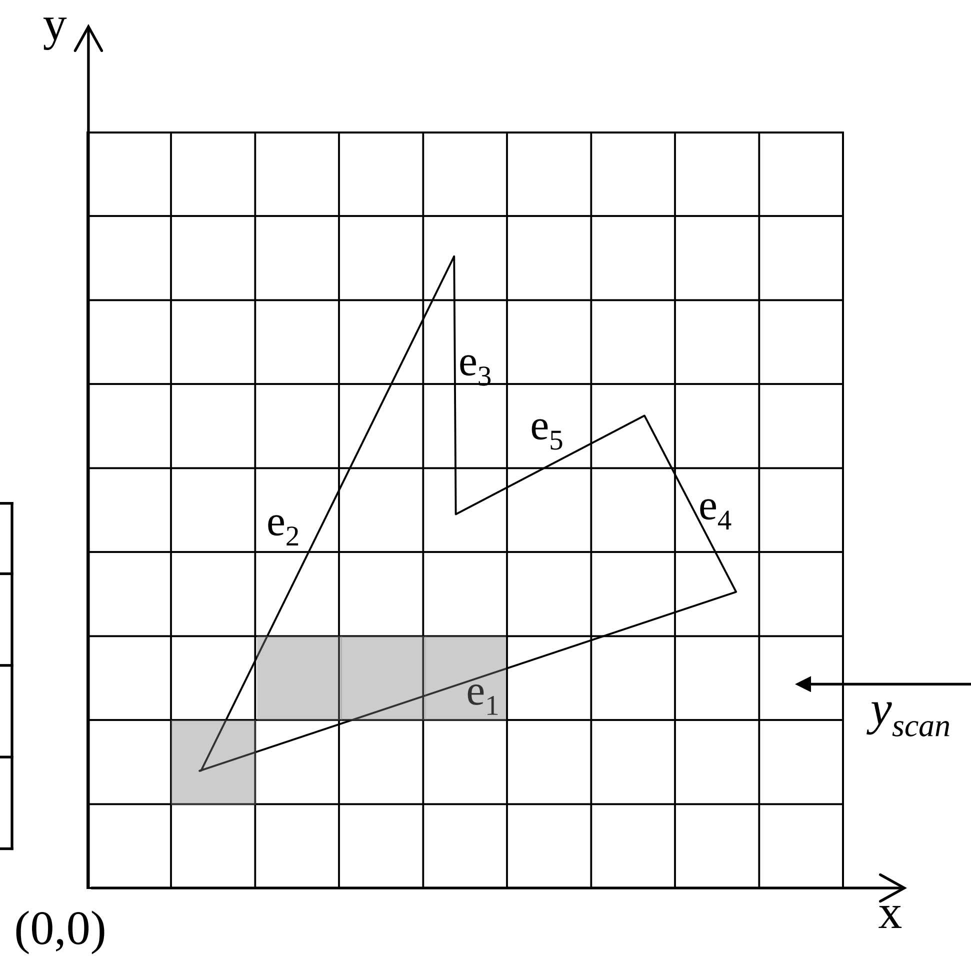
Scanline Algorithm

Scanline $y_{scan} = 2$
AET: edge table, sorted on $x_{intersect}$

edge	x_{inters}	y_{upper}	$1/m$	Next
e_2	3/2	7	1 / 2	e_1
e_1	4	3	3	NULL

ET: edge table, sorted on y_{lower}

edge	y_{lower}	x_{lower}	y_{upper}	$1/m$	Next
e_4	3	7	5	-3	e_3
e_3	4	4	7	0	e_5
e_5	4	4	5	2	NULL



Scanline Algorithm

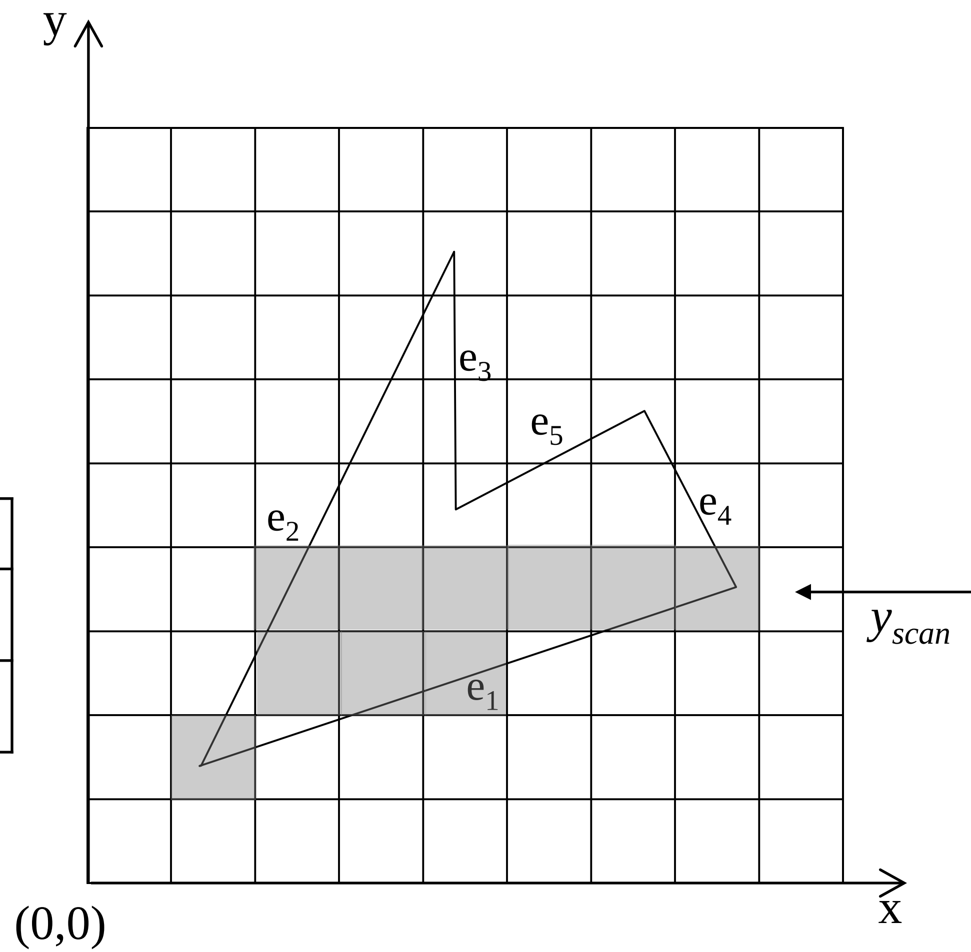
Scanline $y_{scan} = 3$

AET: edge table, sorted on $x_{intersect}$

edge	x_{inters}	y_{upper}	$1/m$	Next
e_2	2	7	1 / 2	e_1
e_4	7	5	-3	NULL

ET: edge table, sorted on y_{lower}

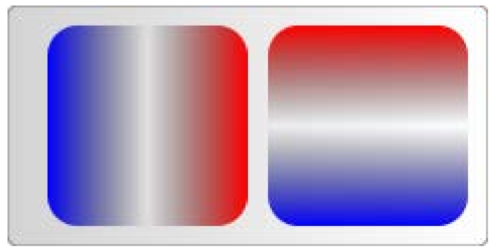
edge	y_{lower}	x_{lower}	y_{upper}	$1/m$	Next
e_3	4	4	7	0	e_5
e_5	4	4	5	2	NULL



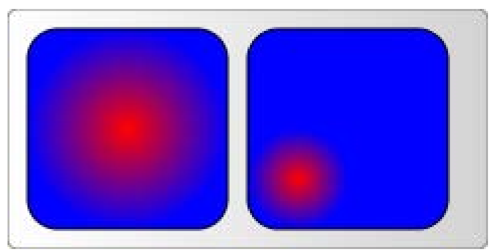
Scanline Algorithm

- Set pixels inside polygon to which color? → "**Shading**"

- We could define color gradients
  - e.g. SVG linear gradients



  - e.g. SVG radial gradients

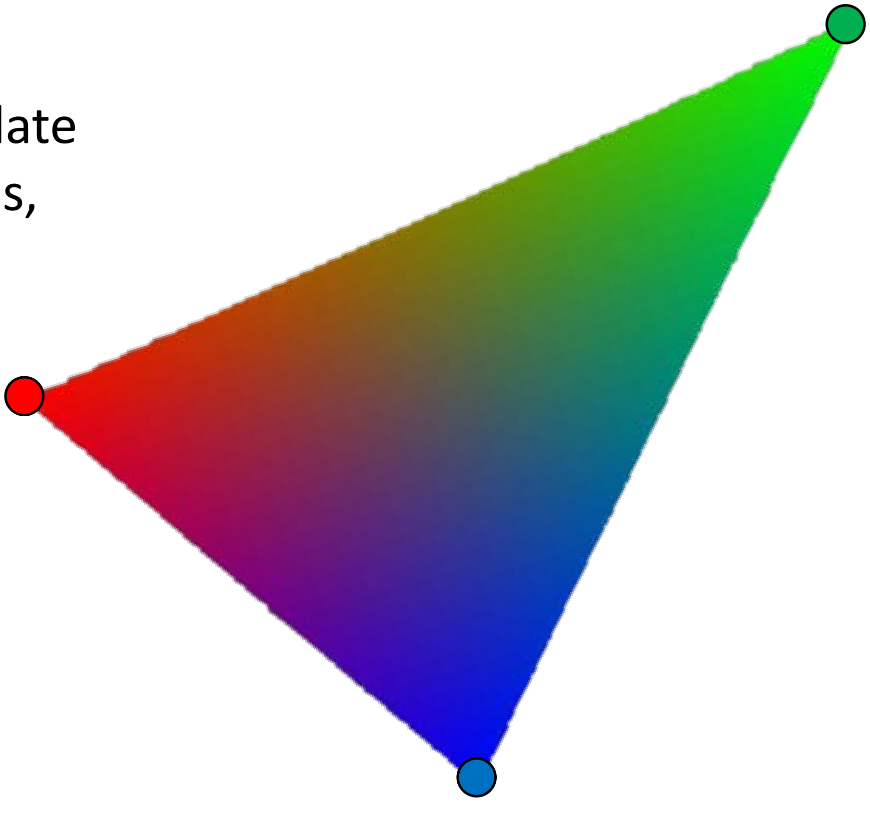


https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Gradients

Scanline Algorithm

- for our purpose, we want to define color values at the vertices of the polygon and interpolate these
  → **Gouraud Shading**

- Later on, we want to interpolate also other attributes (normals, texture coordinates, …)



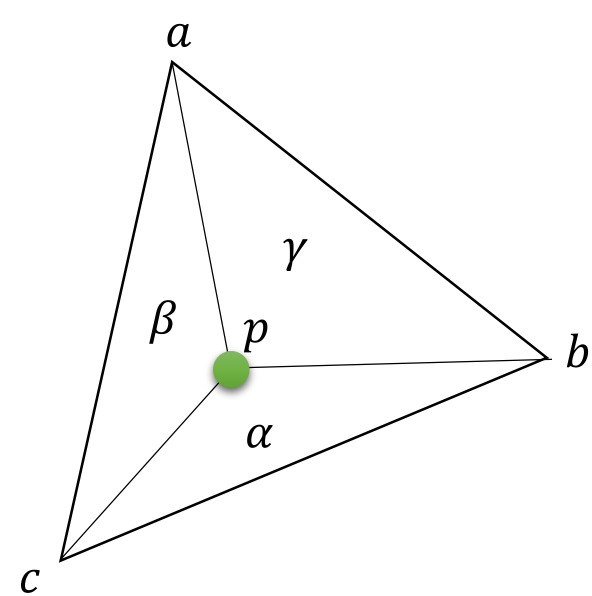
Gouraud Shading

- Interpolating intensities (or other attributes)

- Any point p inside the triangle abc is an affine combination of the vertices

$$p = \alpha a + \beta b + \gamma c$$

with $\alpha + \beta + \gamma = 1$
and $0 < \alpha, \beta, \gamma < 1$



- α, β, γ are the Barycentric Coordinates of p with respect to triangle abc

Gouraud Shading

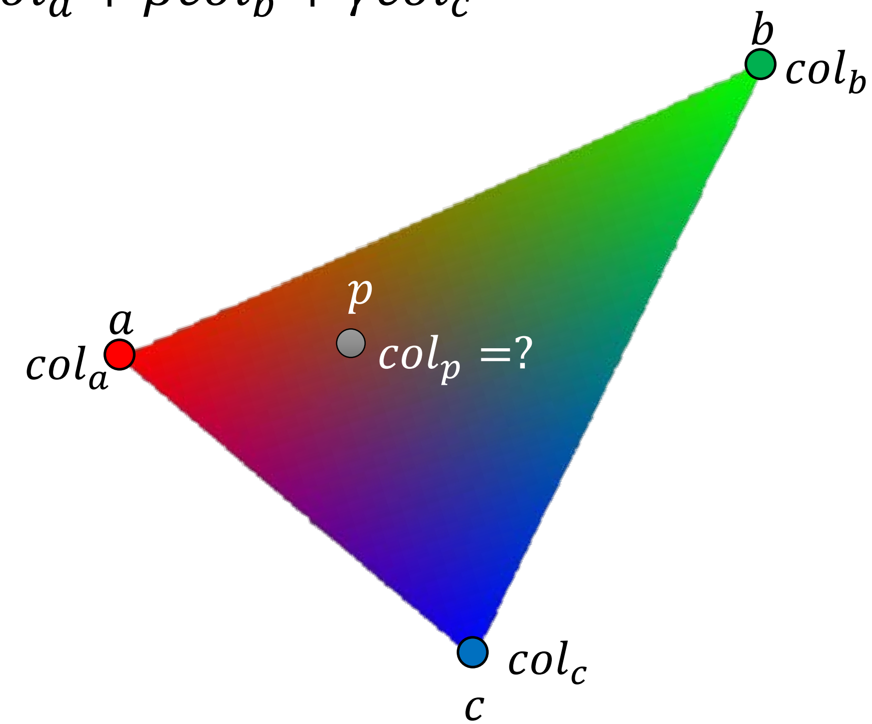
- We can shade a point p using its barycentric coordinates:

$$p = \alpha a + \beta b + \gamma c$$

- We interpolate colors with the same weights:

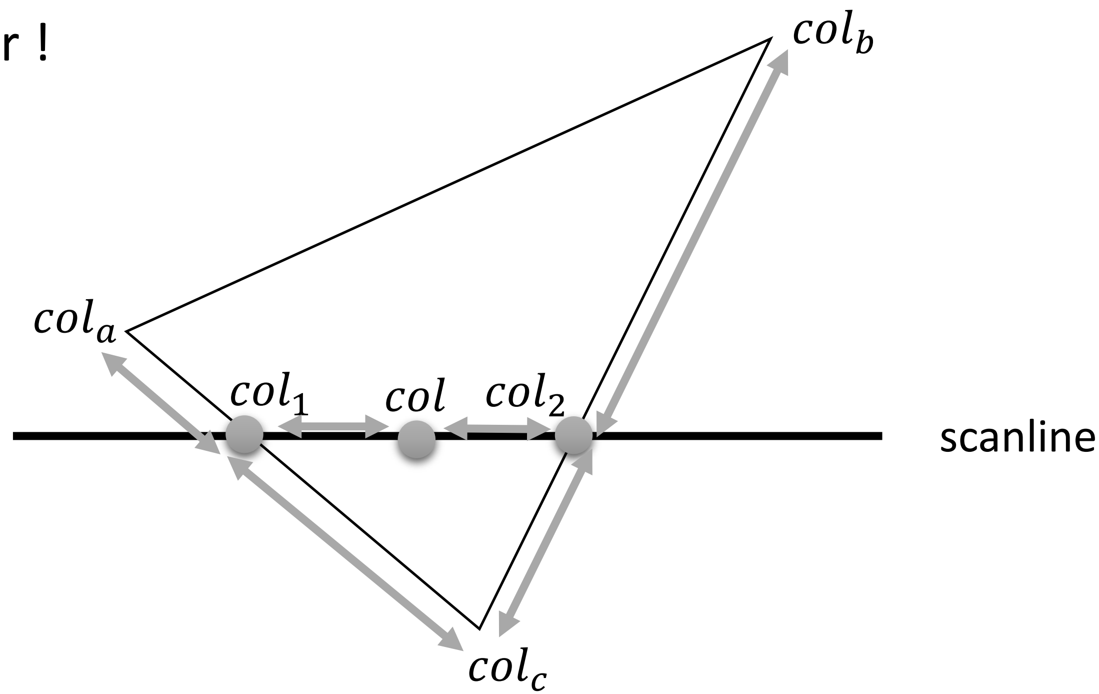
$$col_p = \alpha col_a + \beta col_b + \gamma col_c$$

→ linear interpolation



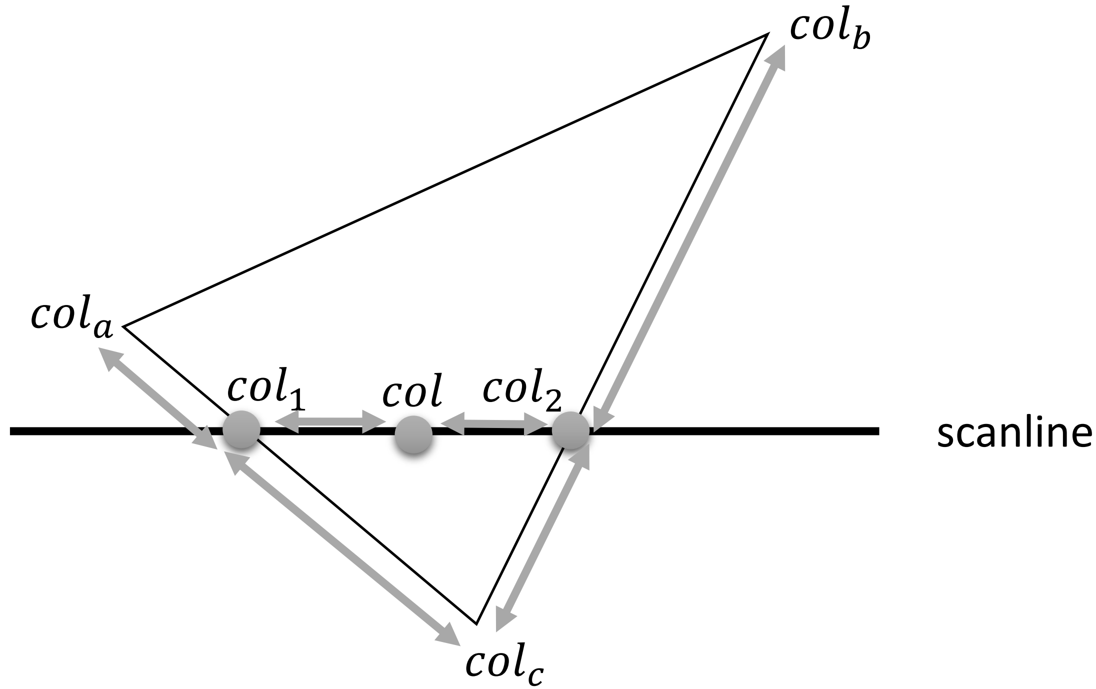
Gouraud Shading

- Algorithmically:
  - do linear interpolation of the attributes along the edges
  - within a span, interpolate linearily
- This is not bilinear, but linear !



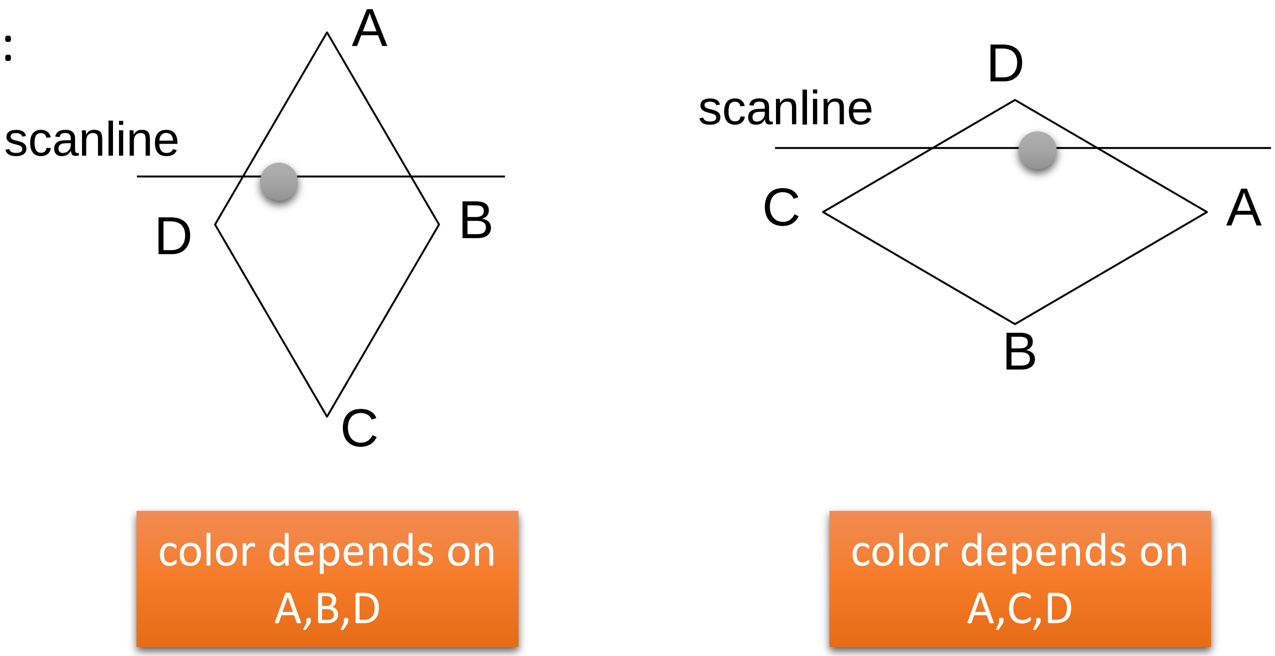
Gouraud Shading

- Can be well combined with scanline rasterization
  - with each edge, store increment of attribute when going one scanline up
  - do not only update x by $1/m$, but also attributes
  - when rasterizing a span, compute attribute updates for $x \to x + 1$



Polygon Shading

- Problems
  - Shading only rotation invariant for triangles
  - for more than 3 vertices: color inside polygon changes with rotation → BAD !
- Example:

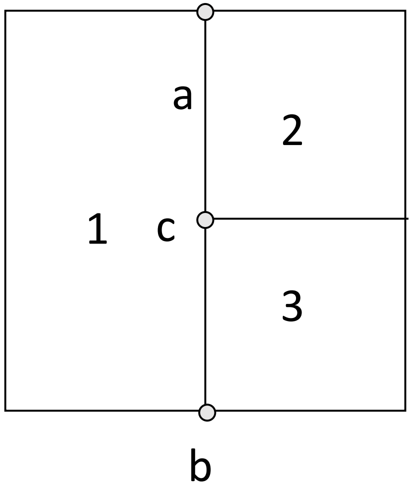


→ triangulate and rasterize triangles

→ but then the color depends on the triangulation…

Polygon Shading

- Problem: Vertex inconsistencies
  - Polygon 1
    - Interpolation between a and b → c
  - Polygons 2 and 3
    - c is separate vertex
- Solution: avoid hanging nodes



Next Lecture

- A short intro to rendering with GPUs…