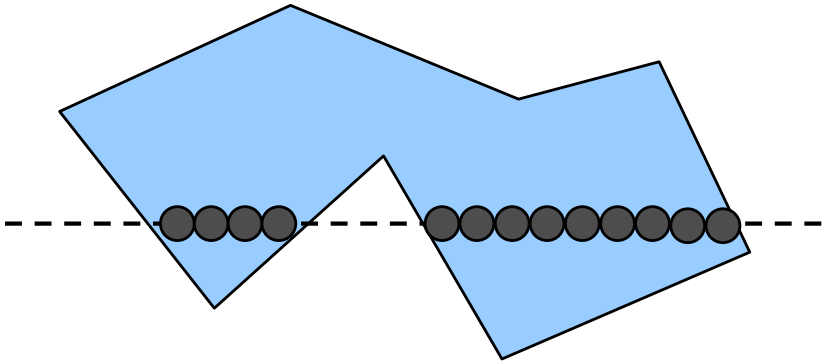
Scan Conversion Algorithms (3)

Contents

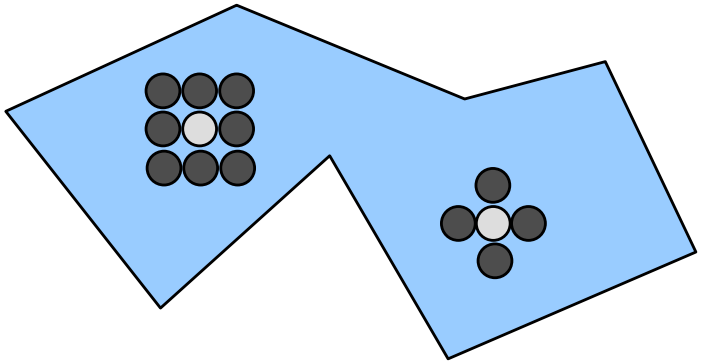
- Basic approach
- Fill area scan conversion
- Polygon scan conversion
- Scan Conversion
  - Ordered Edge List Algorithm
  - Edge Fill Algorithm
  - Fence Fill Algorithm
  - Edge Flag Algorithm
- Seed Filling

Basic approach

- Scan conversion

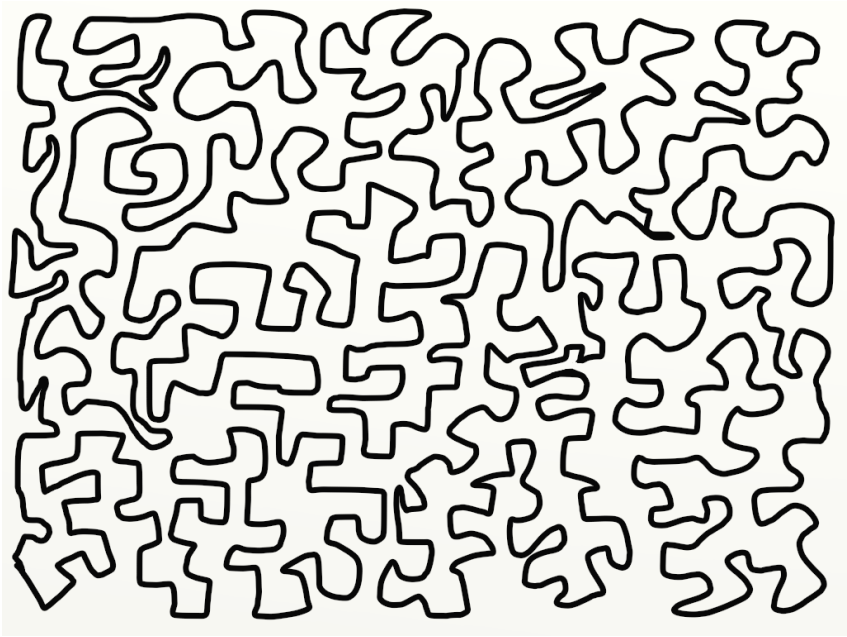


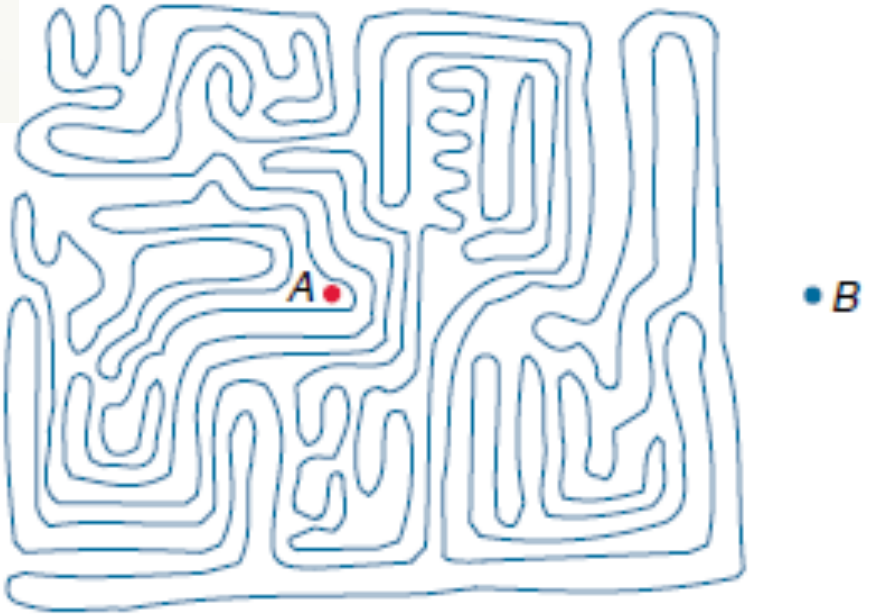
- Seed filling



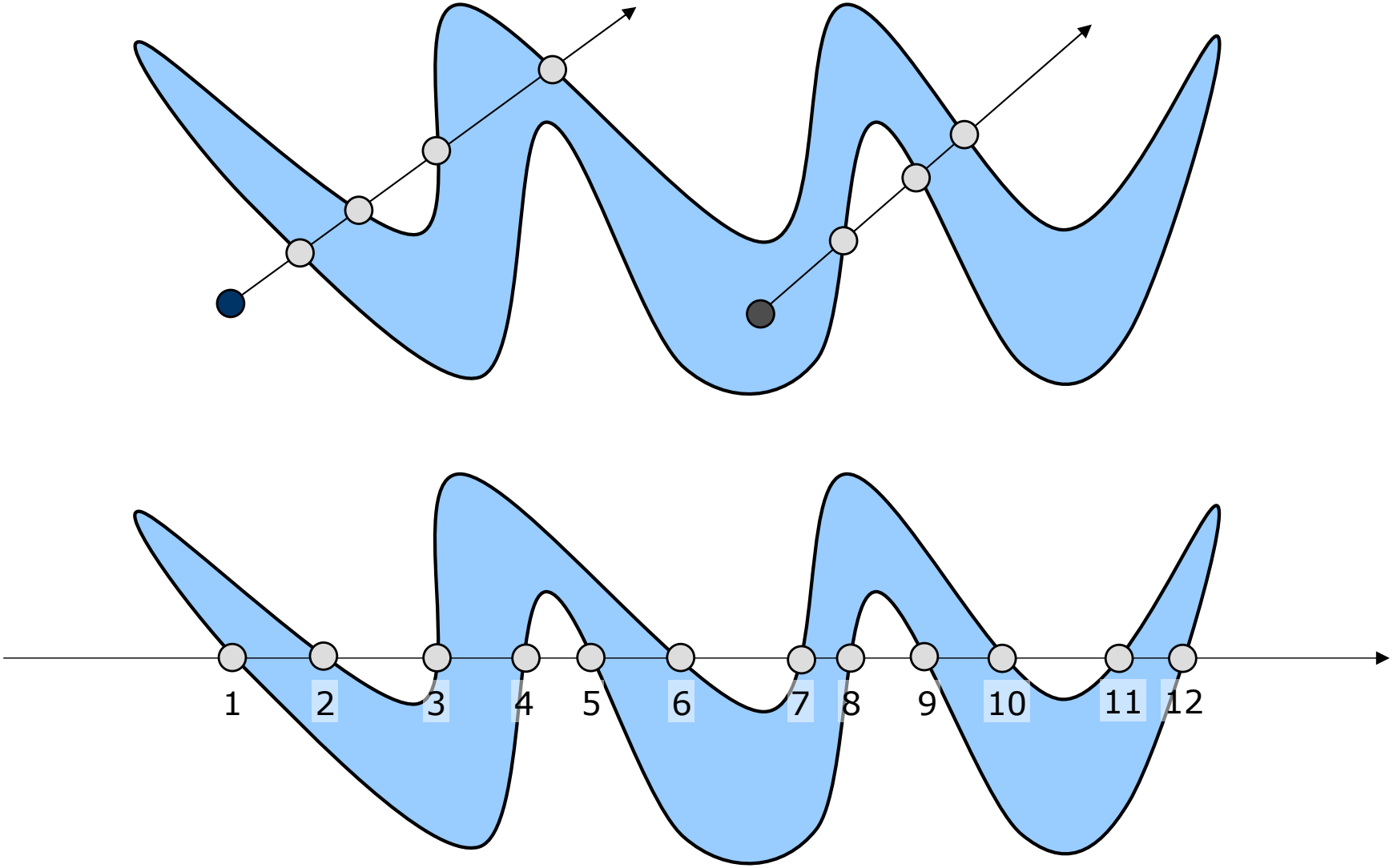
- Needs information about each pixel: position, color, intensity, alfa attribute, etc.

Jordan Theorem



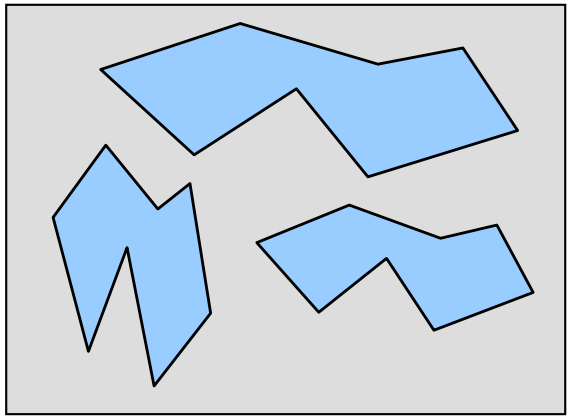


Jordan Theorem

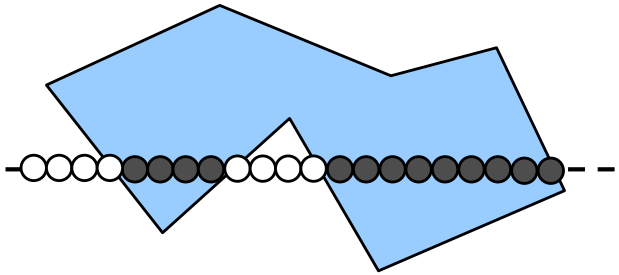


Data structures

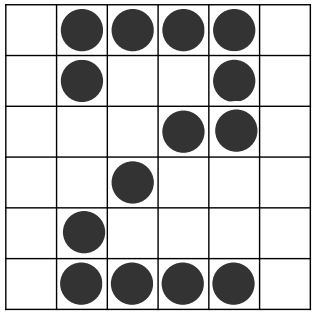
- Real-time scan conversion
  Using geometry and visual attributes



- Run-length encoding
  (intensity, run length)…○4●4○4●9……

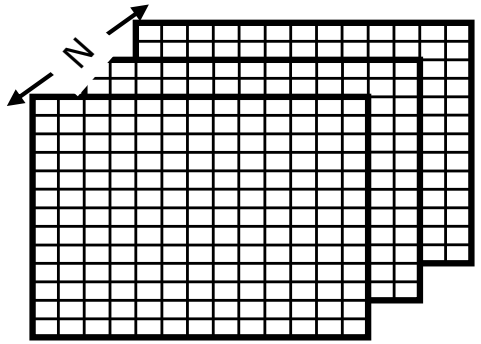


- Cell-organization



Pixel pattern

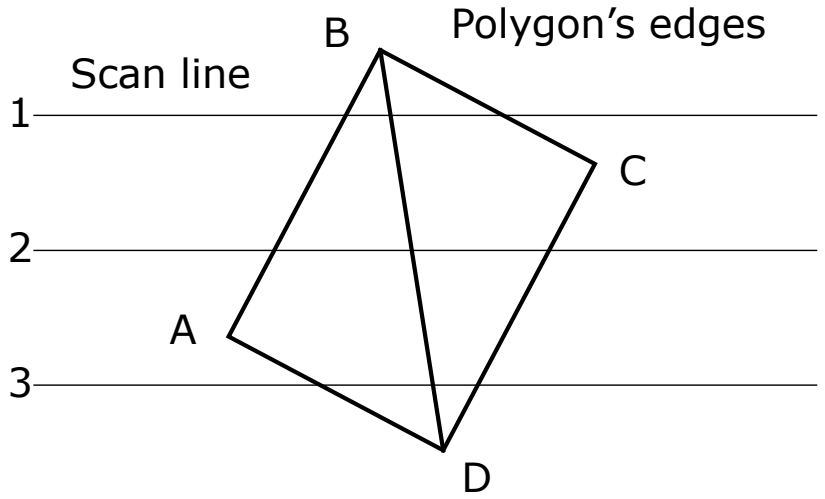
- Frame buffer



Frame Buffer

Real-time scan conversion

- Less memory
- High computation time
- Complex geometric processing
- Flexibility



Edge list 1:

1	2	3
BC ← b	BC	BC
BA	BA ← b	BA
BD ← e	BD	BD ← b
CD	CD ← e	CD
AD	AD	AD ← e

Edge list 2:

1	2	3
BA ← b	BA ← b	BA
BC	**BC**	BC
BD ← e	BD	BD ← b
CD	CD ← e	CD
AD	AD	AD ← e

Edge list 3:

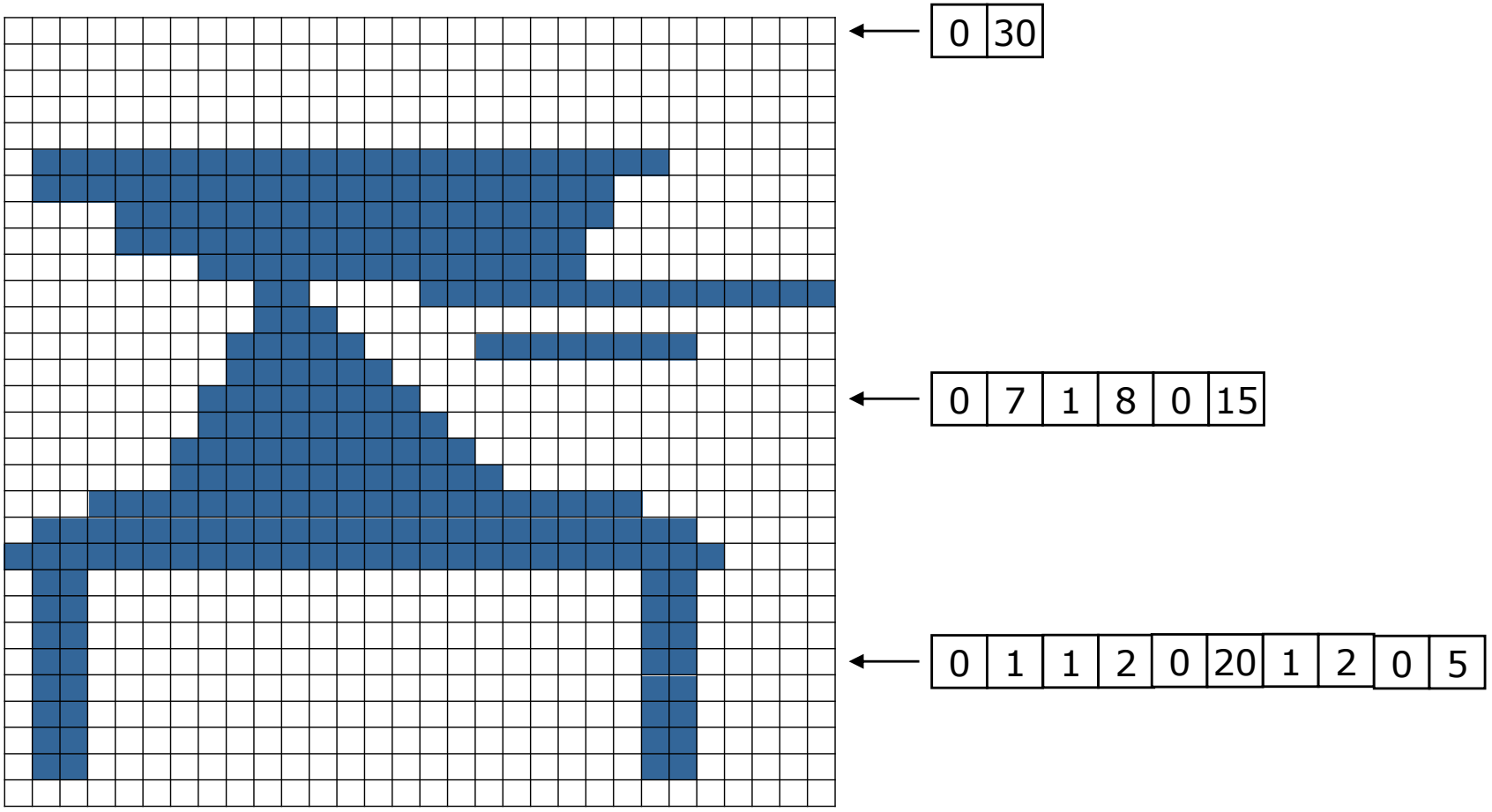
1	2	3
BD ← b	BD ← b	BD ← b
BA	BA	**BA**
BC ← e	**BC**	**BC**
CD	CD ← e	CD
AD	AD	AD ← e

The edge lists must be ordered to intersect consecutive edges.
Ordered by:

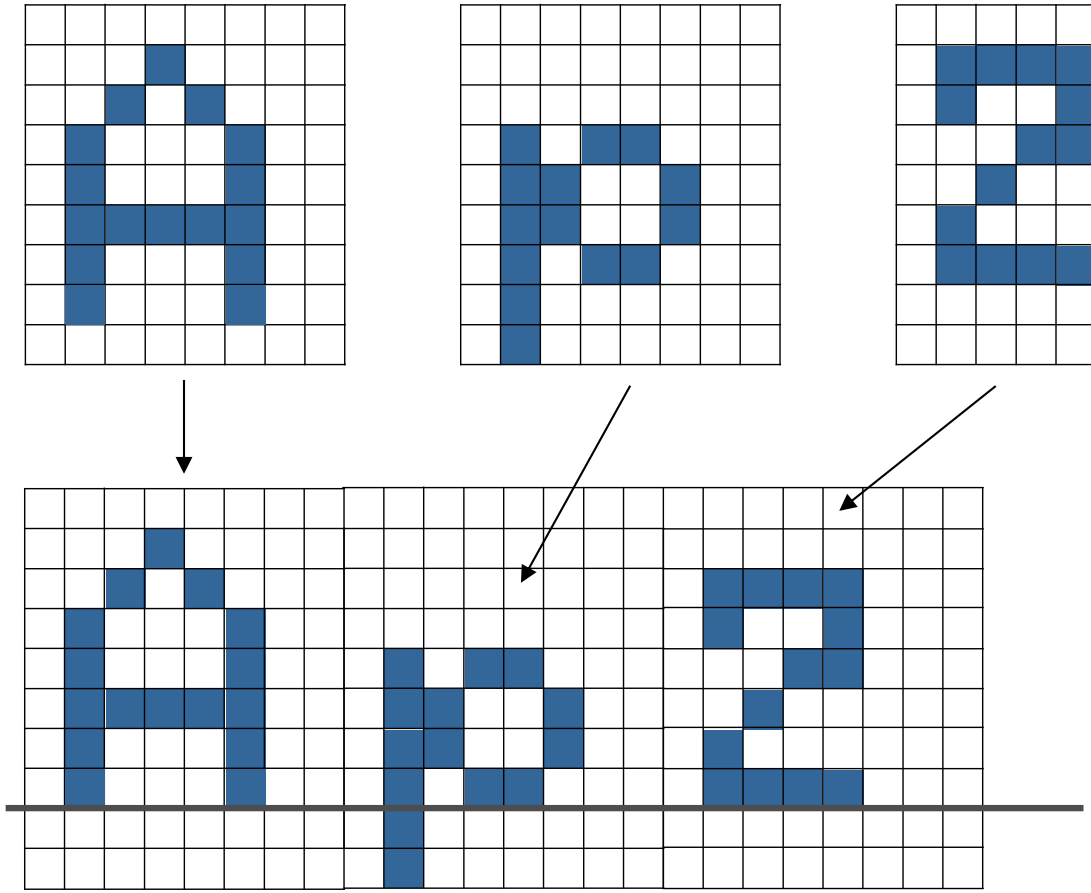
1. Max y of starting vertex (e.g. B, C, A)
2. Max y of ending vertex (e.g. C, A, D)

Run-length encoding

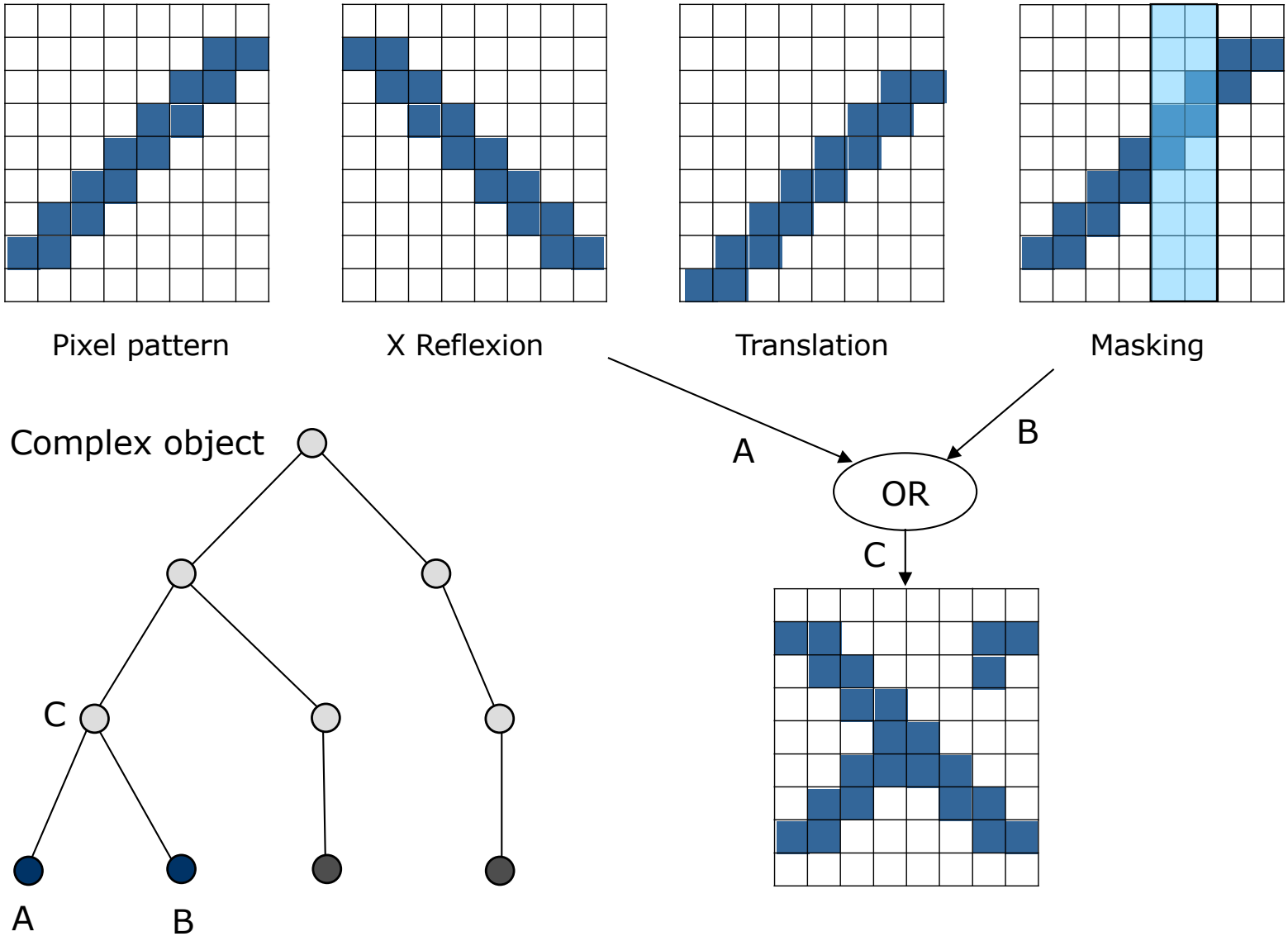
- BW encoding: (intensity, run length)
- Color encoding: (R,G,B, run length)
- Data compression
- High processing time



Cell encoding

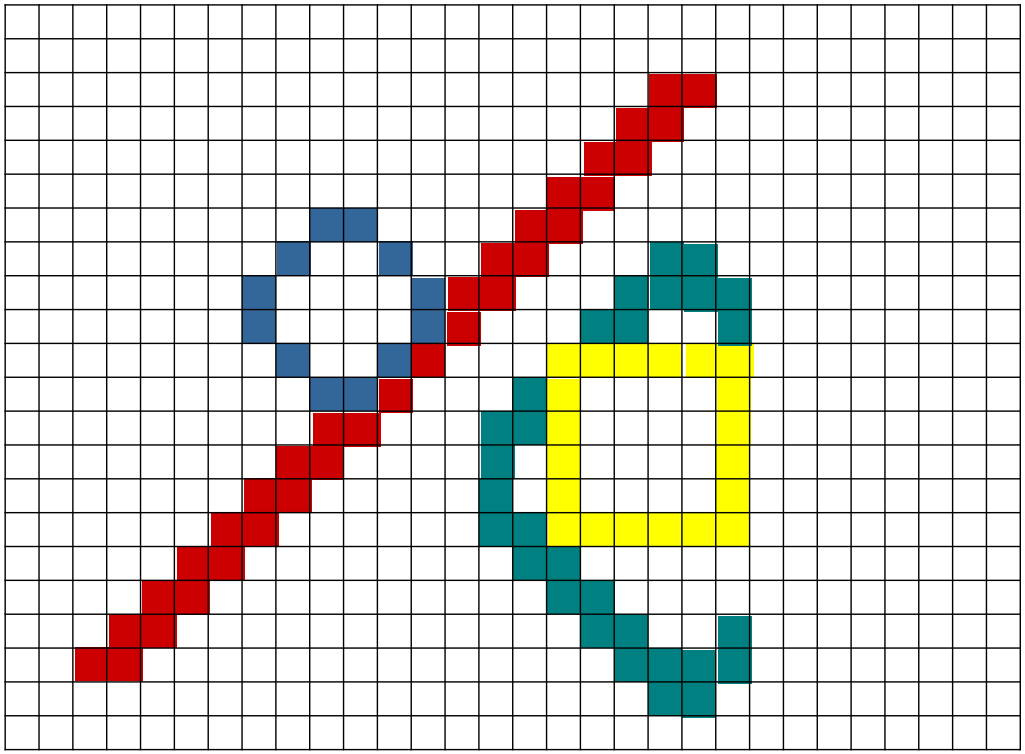


Cell encoding - pseudographics

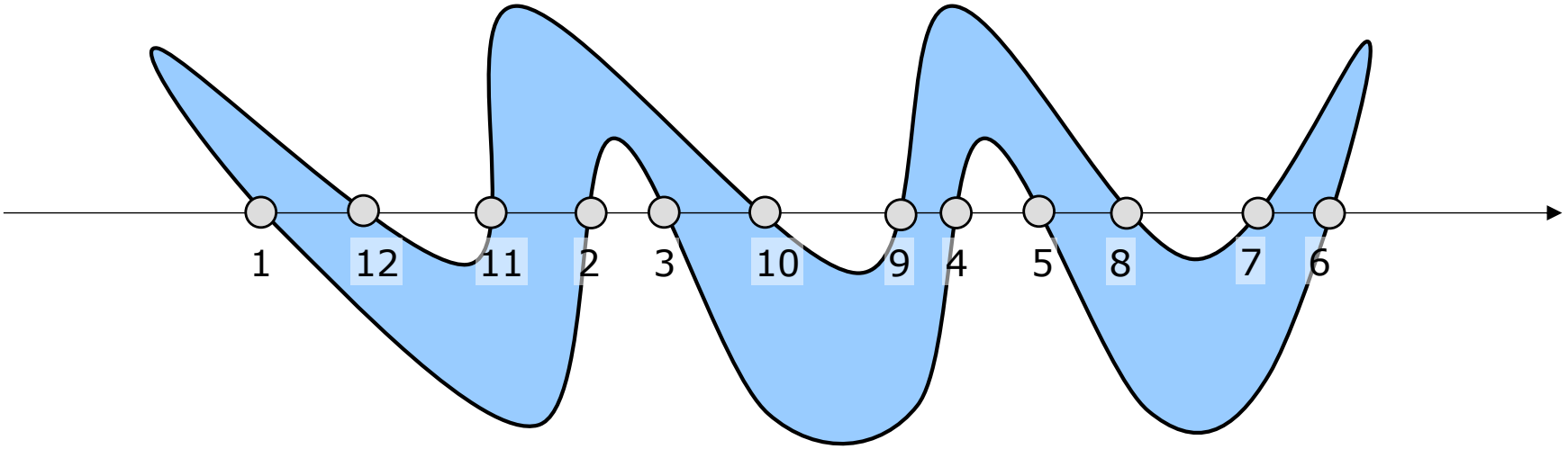


Frame buffer

- Large memory
- High speed
- Time consuming for graphical processing
- Image complexity does not depend on the performance of the display processor



Scan line approach



- Scan line coherence

  Adjacent pixels are likely to have the same characteristics

- Intersection points:

  1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

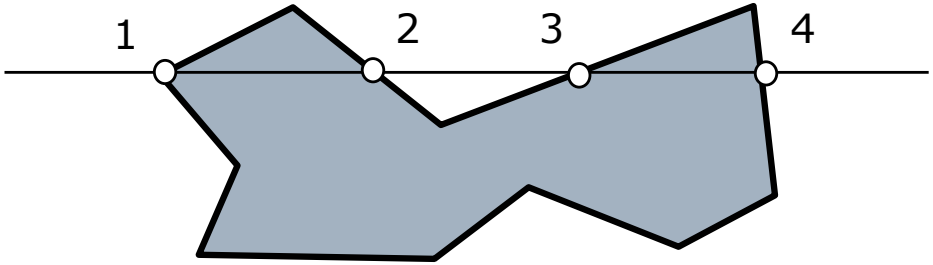
- Ordered sequence:

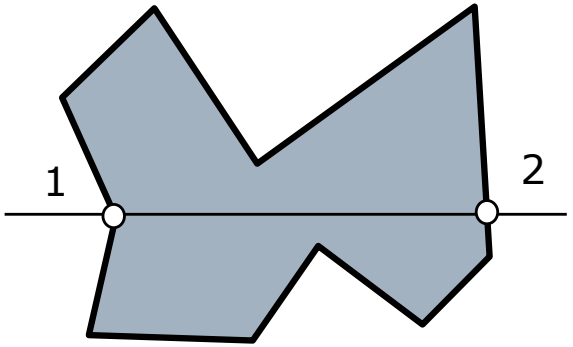
  1, 12, 11, 2, 3, 10, 9, 4, 5, 8, 7, 6

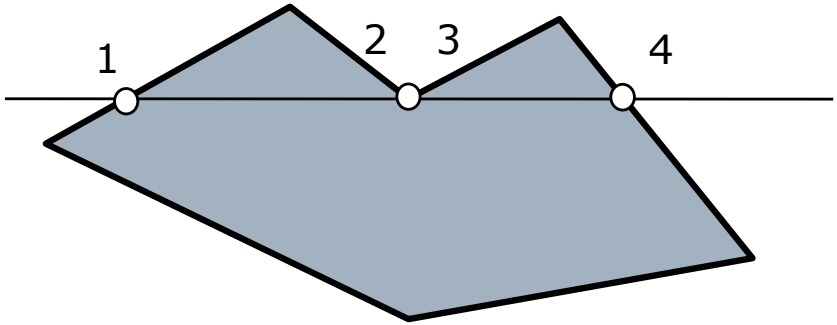
- Ordered pairs:

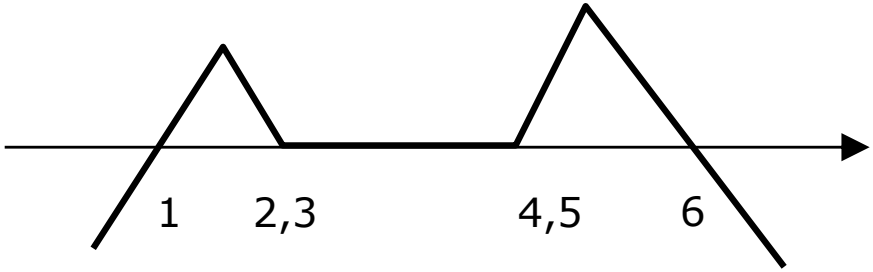
  (1,12), (11,2), (3,10), (9,4), (5,8), (7,6)

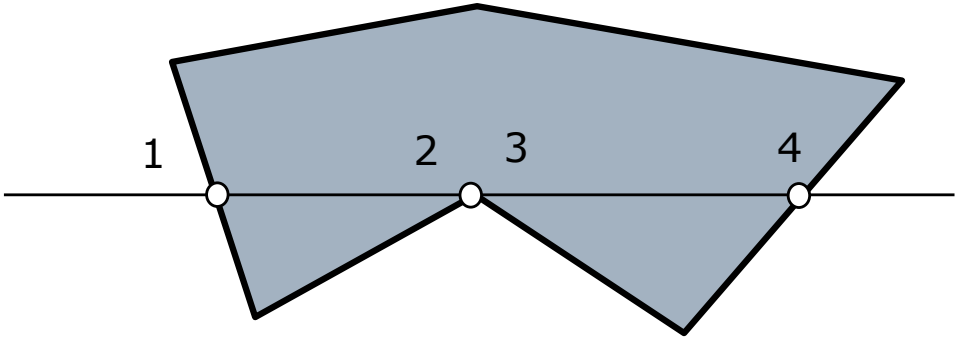
Scan line approach – particular cases

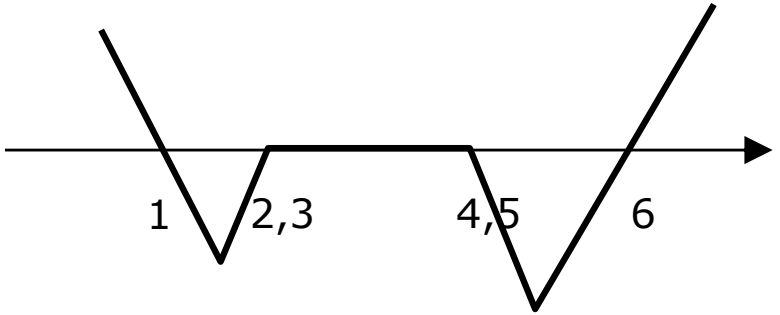




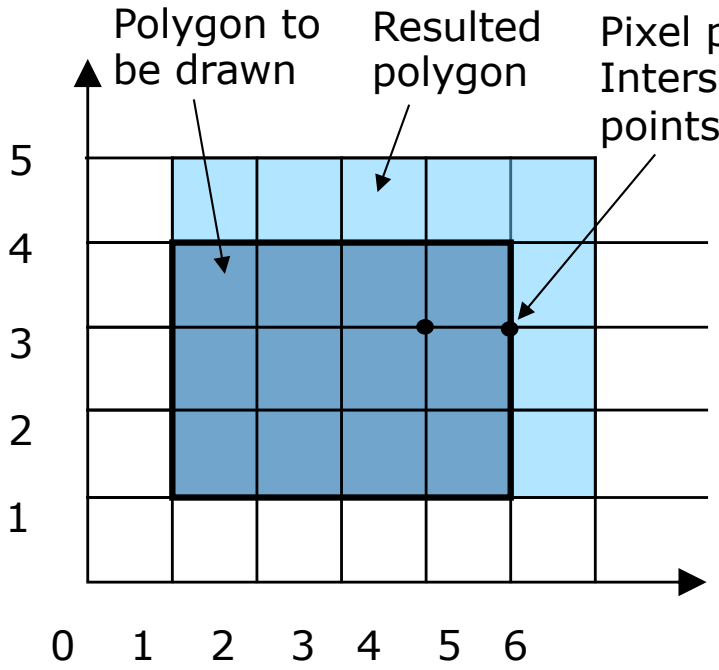




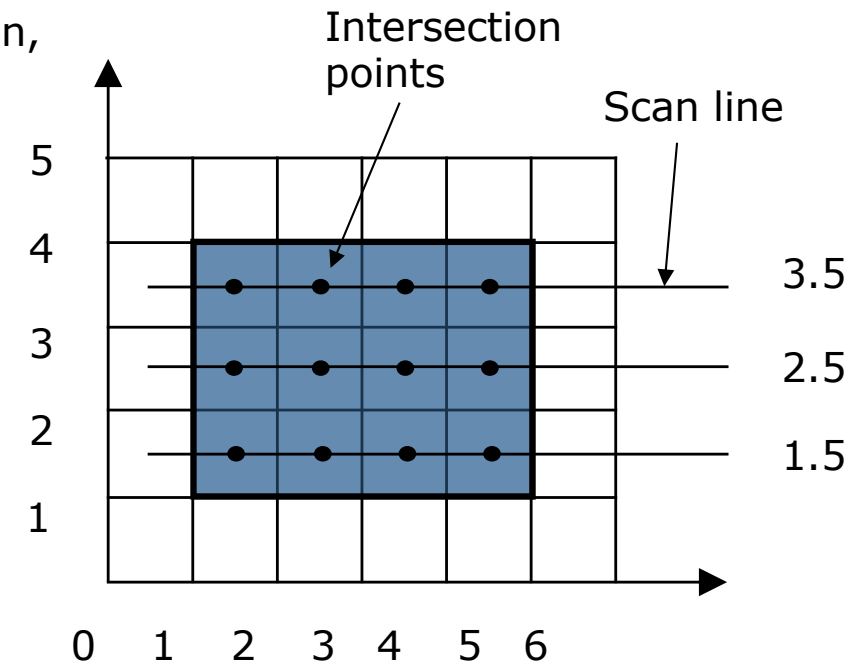




Computation issues



Integer domain:
1. Polygon vertices
2. Scan line y coordinate
3. Intersection coordinates
4. Pixel position

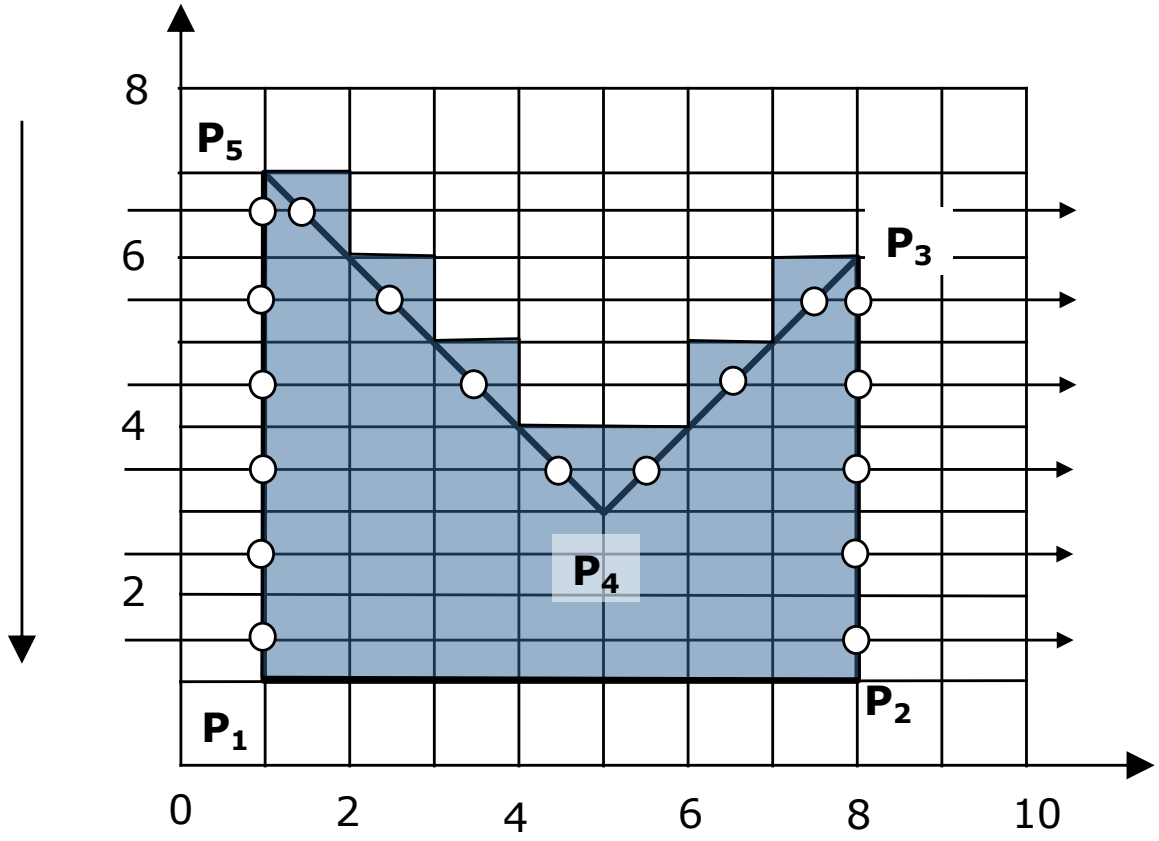


Integer domain:
1. Polygon vertices
2. Pixel position

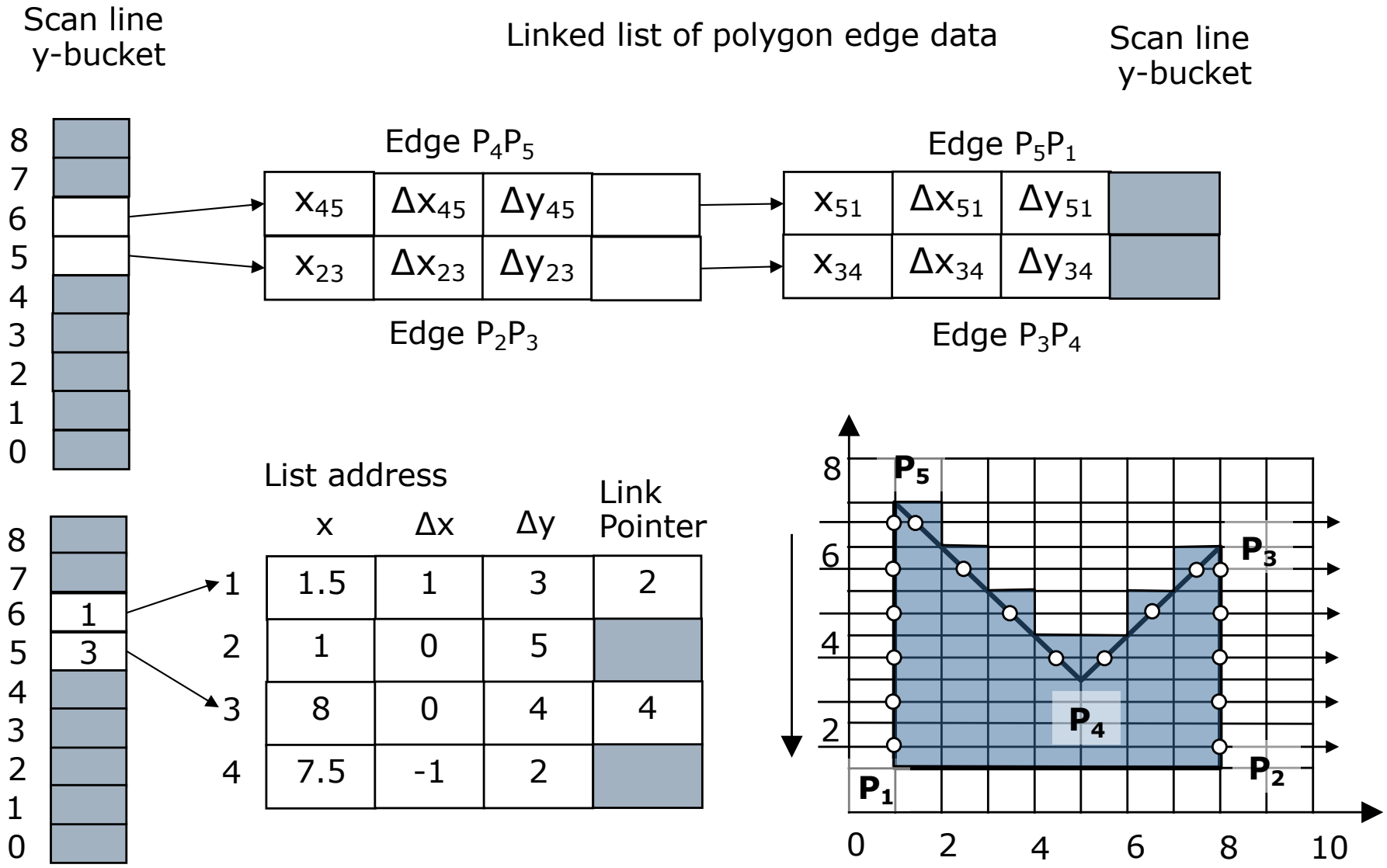
Real domain:
1. Scan line y coordinate
2. Intersection coordinates

Ordered Edge List Algorithm

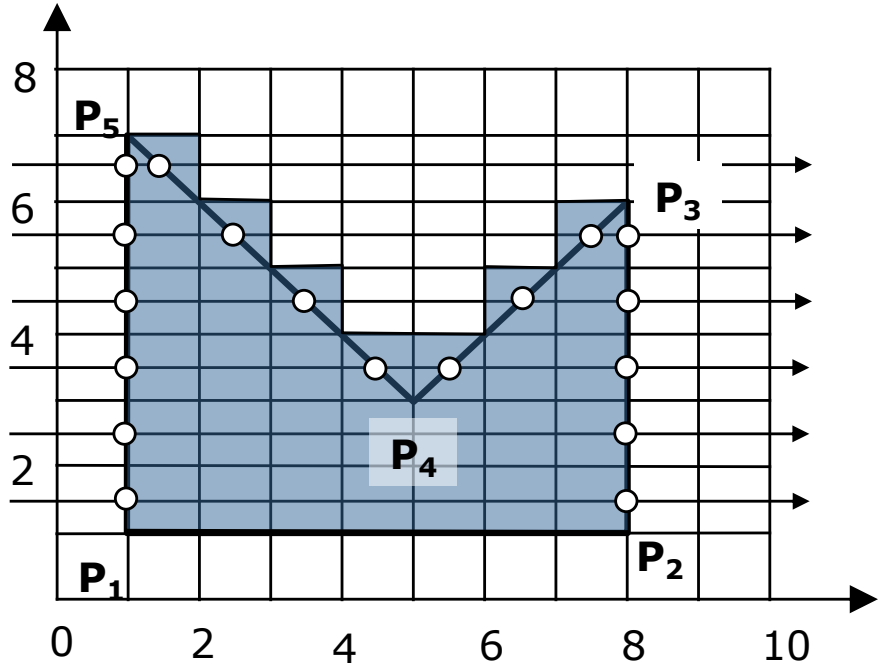
Edge sorting on
y coordinate

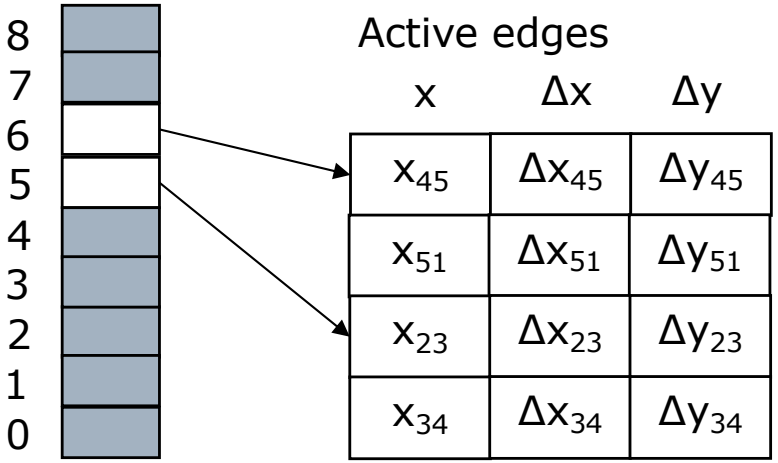


Ordered edge list - Data structures



Active edge list





Active edges

x	Δx	Δy
x_{45}	Δx_{45}	Δy_{45}
x_{51}	Δx_{51}	Δy_{51}
x_{23}	Δx_{23}	Δy_{23}
x_{34}	Δx_{34}	Δy_{34}

Active edge list

Scan line 5:

$x_{45}+\Delta x_{45}$, Δx_{45}, Δy_{45} -1
$x_{51}+\Delta x_{51}$, Δx_{51}, Δy_{51} -1
x_{23}, Δx_{23}, Δy_{23}
x_{34}, Δx_{34}, Δy_{34}

Scan line 4:

$x_{45}+2\Delta x_{45}$, Δx_{45}, Δy_{45} -2
$x_{51}+2\Delta x_{51}$, Δx_{51}, Δy_{51} -2
$x_{23}+\Delta x_{23}$, Δx_{23}, Δy_{23} -1
$x_{34}+\Delta x_{23}$, Δx_{34}, Δy_{34} -1

Scan line 3:

$x_{45}+3\Delta x_{45}$, Δx_{45}, Δy_{45} -3
$x_{51}+3\Delta x_{51}$, Δx_{51}, Δy_{51} -3
$x_{23}+2\Delta x_{23}$, Δx_{23}, Δy_{23} -2
$x_{34}+2\Delta x_{23}$, Δx_{34}, Δy_{34} -2

Scan line 2:

$x_{51}+4\Delta x_{51}$, Δx_{51}, Δy_{51} -4
$x_{23}+3\Delta x_{23}$, Δx_{23}, Δy_{23} -3

Active edge list

	x	Δx	Δy
8			
7			
6	1.5	1	3
	1	0	5
5	2.5	1	2
	1	0	4
	8	0	4
	7.5	-1	2
4	3.5	1	1
	1	0	3
	8	0	3
	6.5	-1	1
3	4.5	1	0
	1	0	2
	8	0	2
	5.5	-1	0
2	1	0	1
	8	0	1
1	1	0	0
	8	0	0
0			

Intersection points ordered by x coordinate

1	1.5		
1	2.5	7.5	8
1	3.5	6.5	8
1	4.5	5.5	8
1	8		
1	8		

Pixel list

- (1,6)
- (1,5), (2,5), (7,5)
- (1,4), (2,4), (3,4), (6,4), (7,4)
- (1,3),(2,3),(3,3),(4,3), (5,3),(6,3),(7,3)
- (1,2),(2,2),(3,2),(4,2), (5,2),(6,2),(7,2)
- (1,1),(2,1),(3,1),(4,1), (5,1),(6,1),(7,1)

Sample of computation

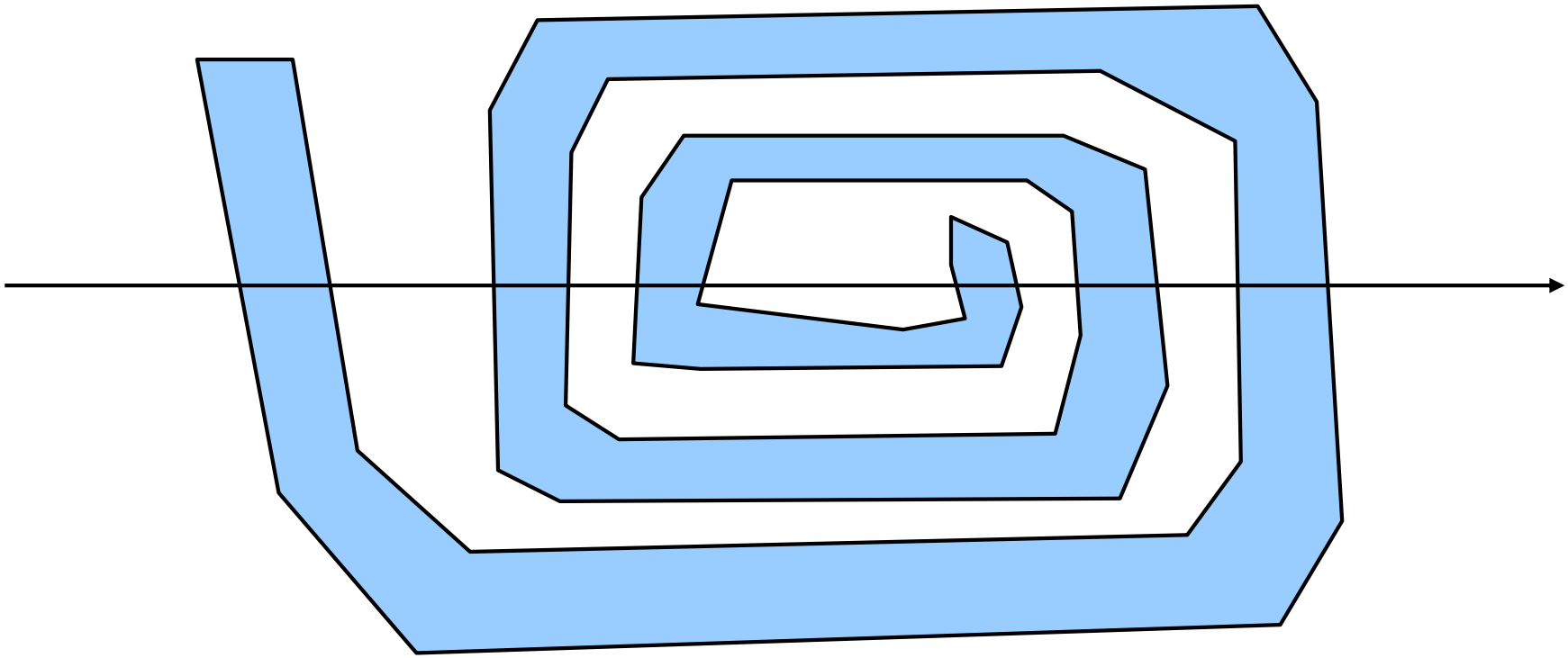
						$P_2(2,10)$		dy	1										Referinta
	Linie	Referinta Pixel stg					-0,33	dx	2,67										Pixel dr
	baleiaj	xp	yp	x+dx	y	5,50	8,50	9,81	9,44	9,06	8,69	8,31	7,94	7,56	7,19	x+dx	x	y	xp	yp
	9,5			1,83	9,50											3,33	2,50	9,50	2	9
	8,5	1	8	1,50	8,50											6,00	5,50	8,50	5	8
	7,5	1	7	1,17	7,50											8,67	8,50	7,50	8	7
						0,5	1,5	2,5	3,5	4,5	5,5	6,5	7,5	8,5	9,5					
						P_3										P_1				
						(1,7)										(9,7)				

Trasare algoritm:

lb		P2P3				P2P1			Pixel stg		Pixel dr	
9,5	→	x	dx	y	→	x	dx	y	x	y	x	y
		1,83	-0,33	2		3,33	2,67	2			2	9
8,5	→	x	dx	y	→	x	dx	y	x	y	x	y
		1,50	-0,33	1		6,00	2,67	1	1	8	5	8
7,5	→	x	dx	y	→	x	dx	y	x	y	x	y
		1,17	-0,33	0		8,67	2,67	0	1	7	8	7

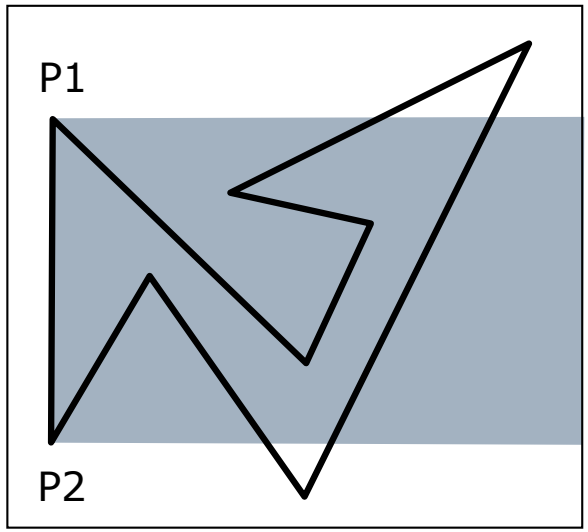
Edge fill algorithm

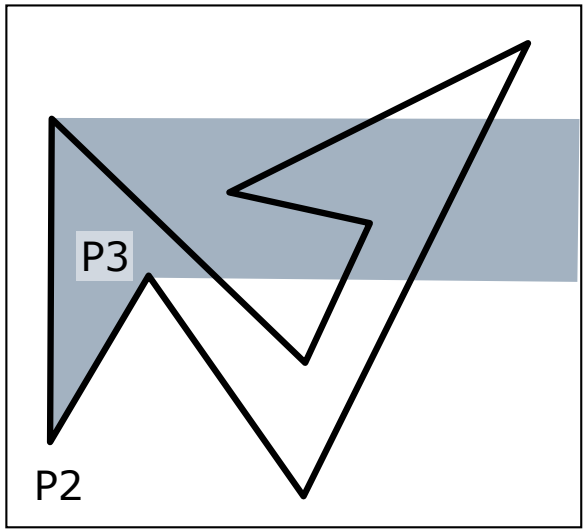
- Complement the pixel on the right of the edge

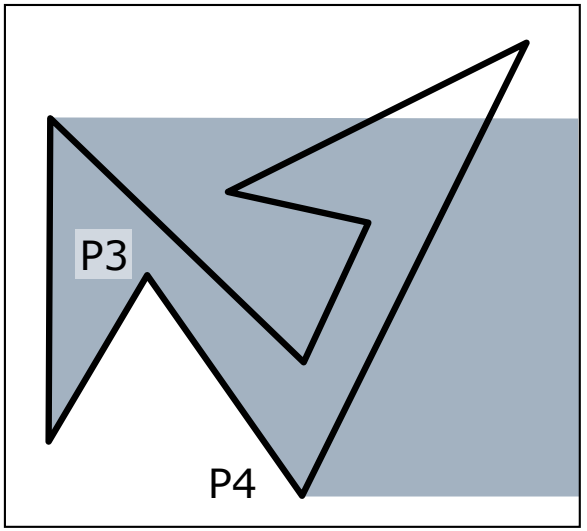


Edge fill algorithm

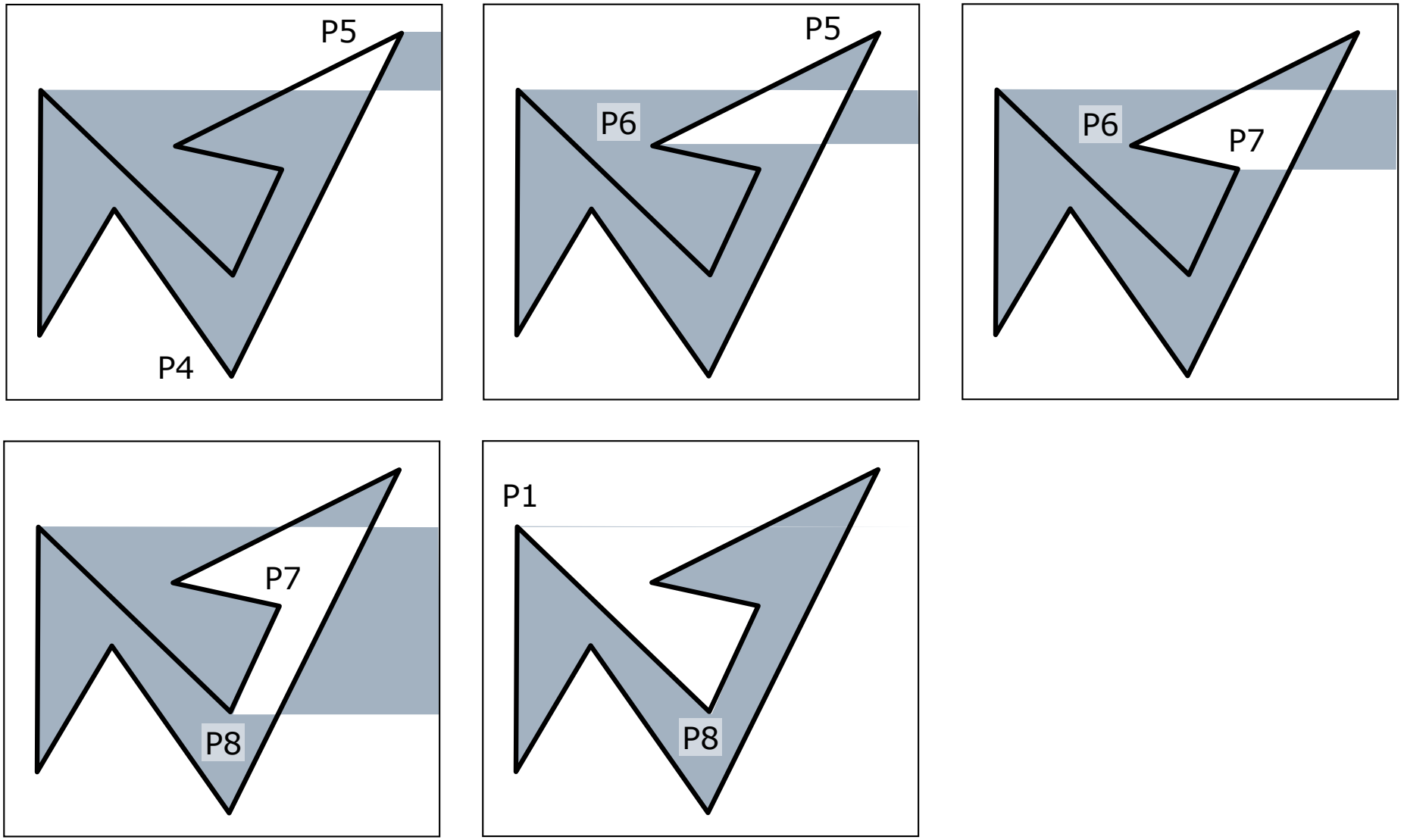
- No sorting
- Simple data structure
- Each pixel should be accessed many times





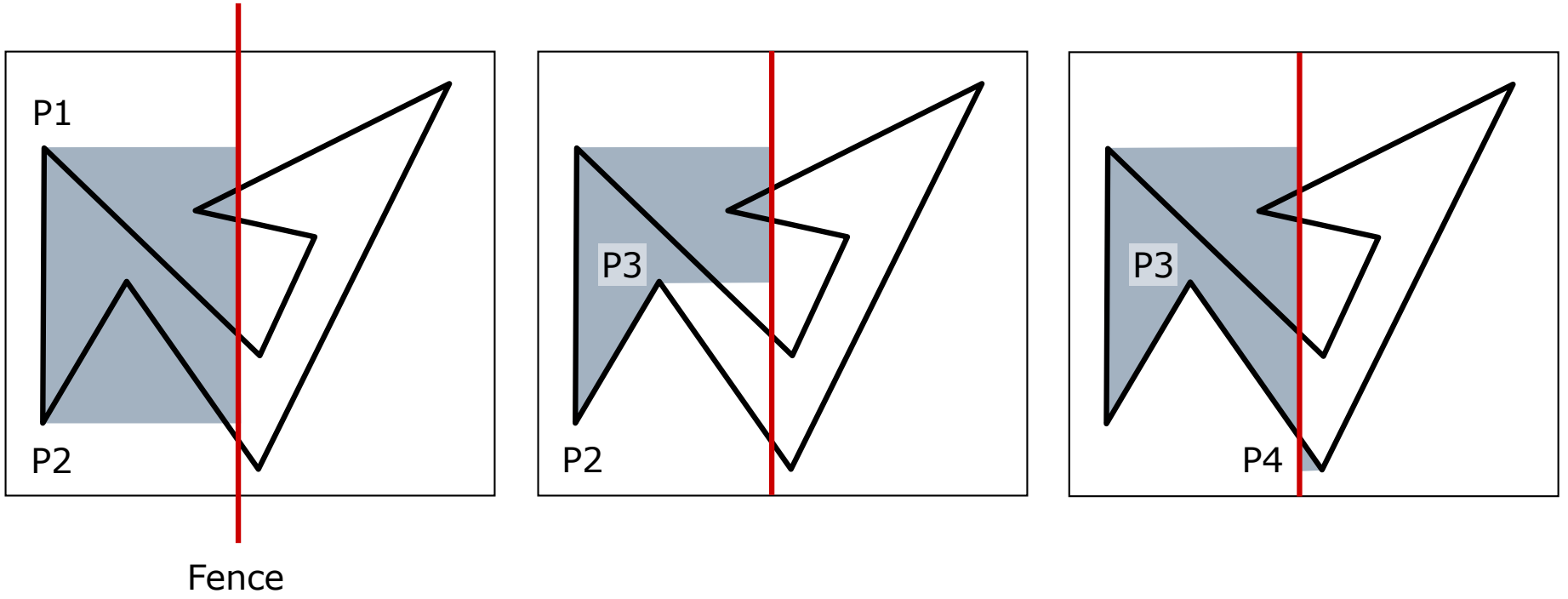


Edge fill algorithm

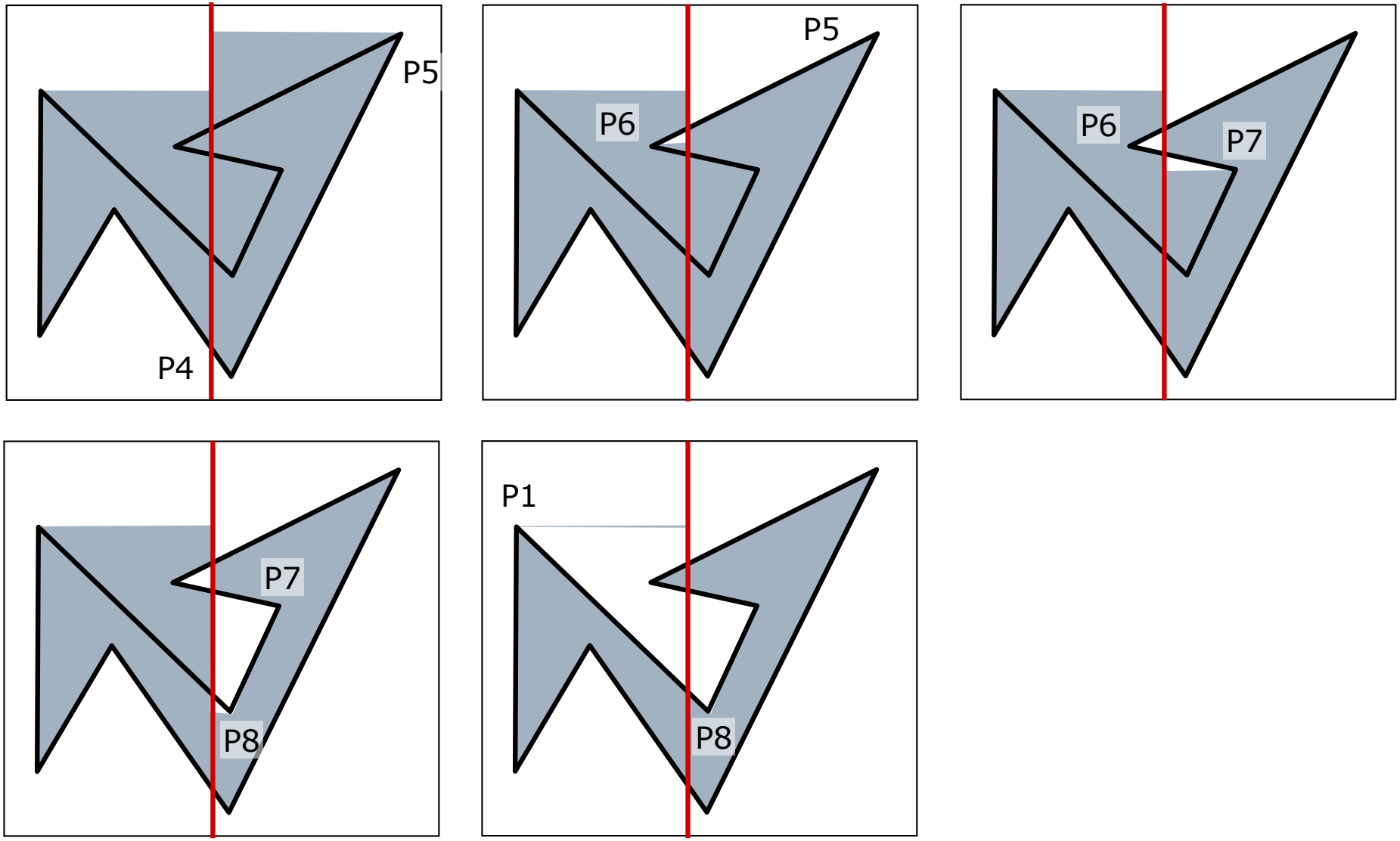


Fence fill algorithm

- Reduce the number of pixel accesses

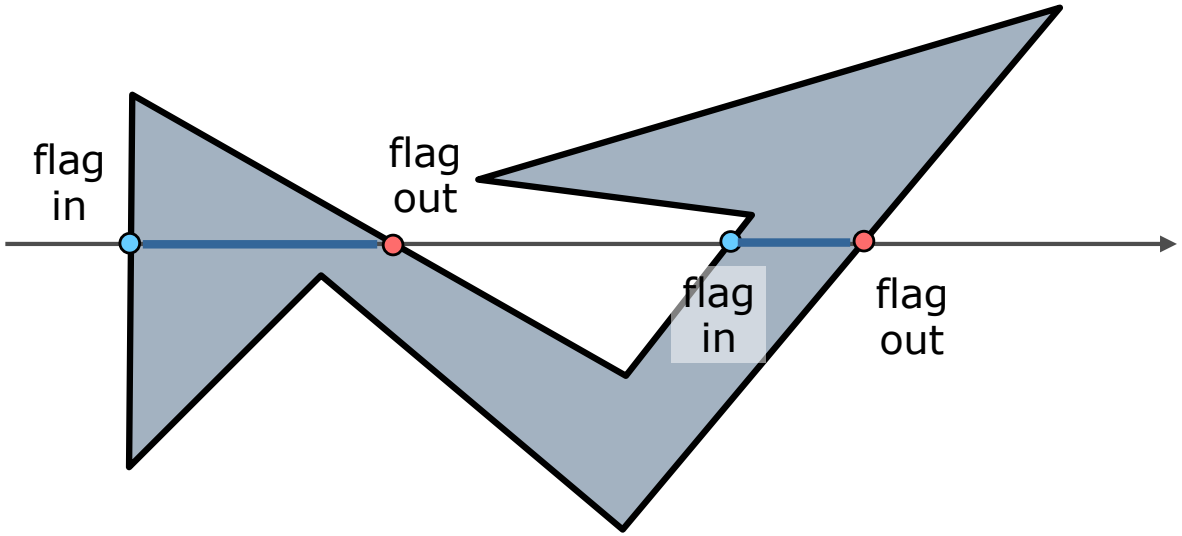


Fence fill algorithm



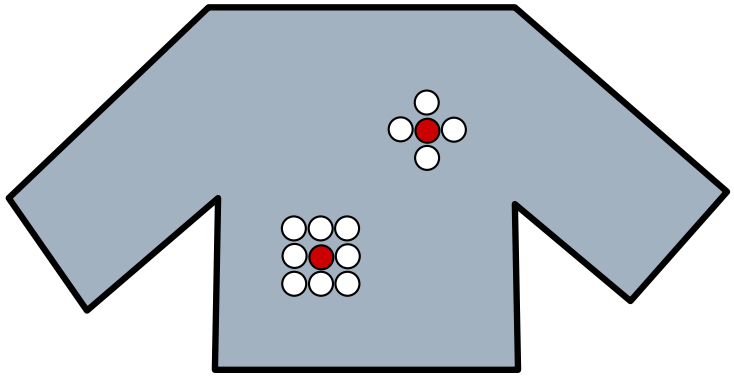
Edge flag algorithm

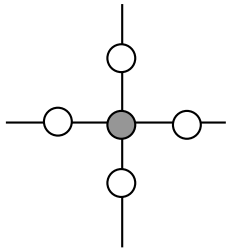
- Each pixel is accessed only once
- No sorting and maintaining edge lists
- Very fast and could be implemented by hardware

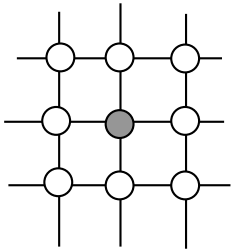


Seed fill algorithm

- Start at a pixel inside the polygon
- Seed: starting pixel
- Move to the neighbors using the connectivity by 4 and 8 neighbors
- Recursive algorithms

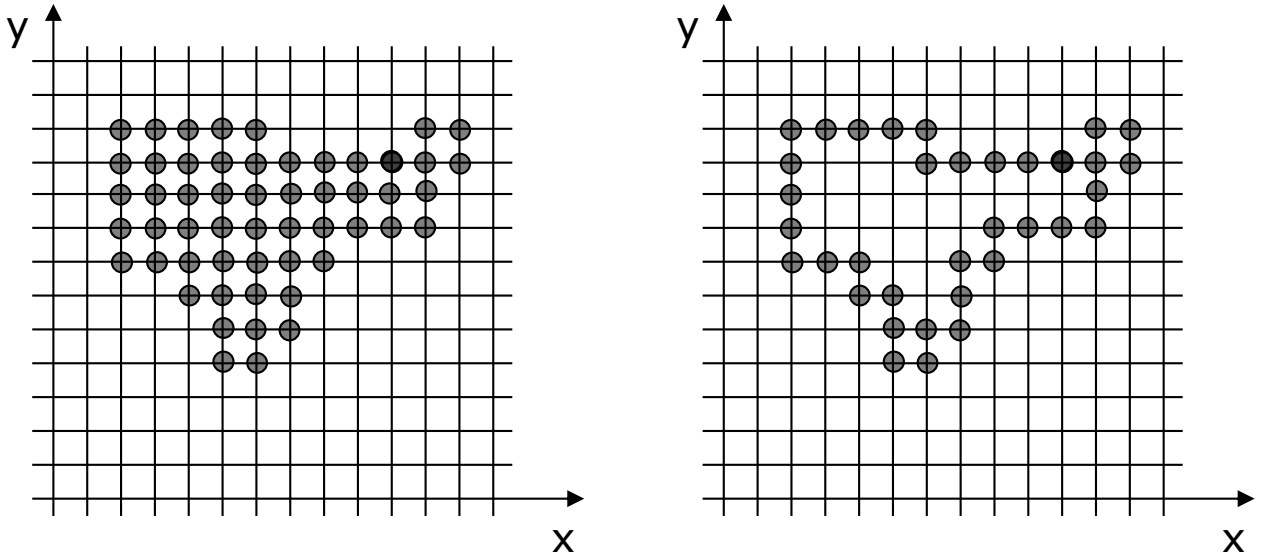






Type of region

- Defined by
  1. Content
  2. Contour



Recursive algorithms - Fill4Content

```
procedure Fill4Content(x, y, oldval, newval: integer);
{x, y: starting point, inside region;
oldval: the old inside pixel colour;
newval: the new colour of pixels}
begin
    if ReadPixel(x, y) = oldval then
    begin
        WritePixel(x, y, newval);
        Fill4Content(x, y-1, oldval, newval);
        Fill4Content(x, y+1, oldval, newval);
        Fill4Content(x-1, y, oldval, newval);
        Fill4Content(x+1, y, oldval, newval);
    end
end;
```

Recursive algorithms - Fill4Contour

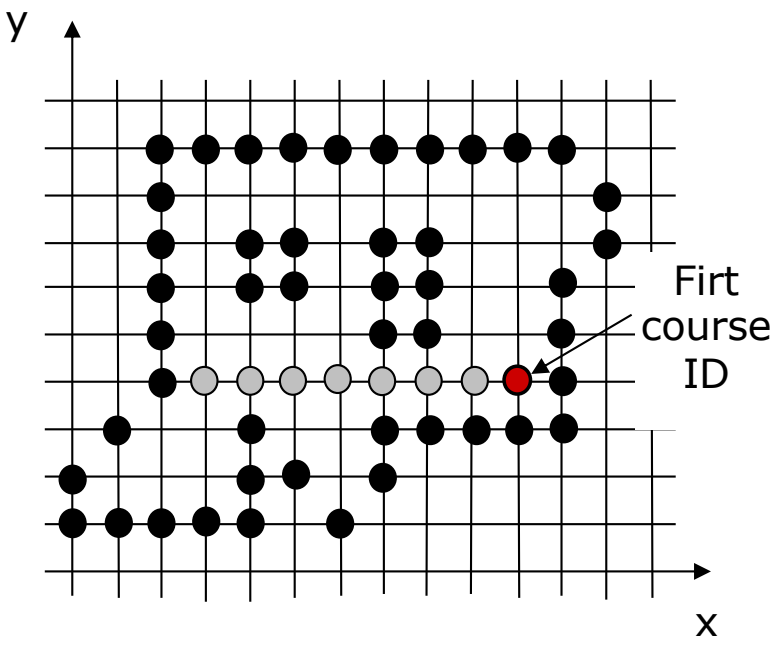
```
procedure Fill4Contour(x, y, contval, newval: integer);
{x, y: starting point, inside region;
contval: the old contourcolour;
newval: the new colour of the contour pixels}
begin
    if ReadPixel(x, y) <> contval and ReadPixel(x, y) <> newval then
    begin
        WritePixel(x, y, newval);
        Fill4Contour(x, y-1, contval, newval);
        Fill4Contour(x, y+1, contval, newval);
        Fill4Contour(x-1, y, contval, newval);
        Fill4Contour(x+1, y, contval, newval);
    end
end;
```

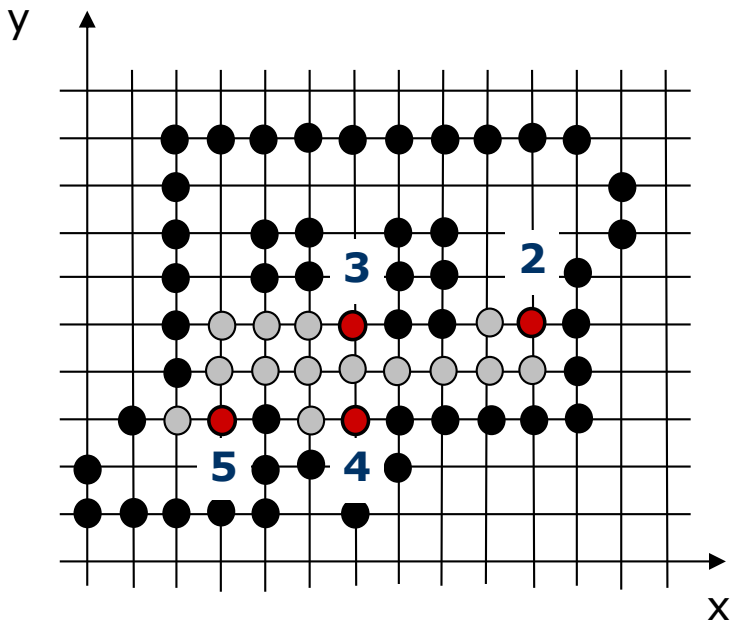
Course based algorithm

- Recursivity is very stack memory consuming
- Reduce the level of recursivity: *course*

  Course:

  1. Most right pixel positiony
  2. An horizontal sequence of consecutive pixels
- Only course indentifier stored into the stack





Course based algorithm

- Determinate the current course, related to the starting position
- Write into the stack the course ID
- While stack is not empty {

    Read and render a course from the stack

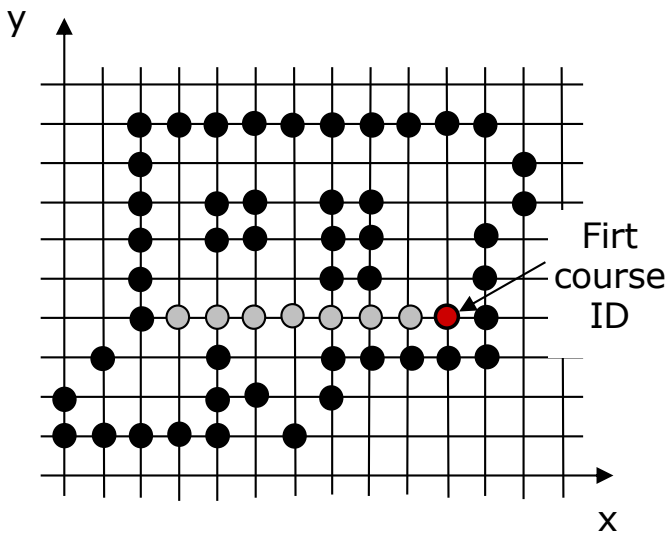
    For all above line courses

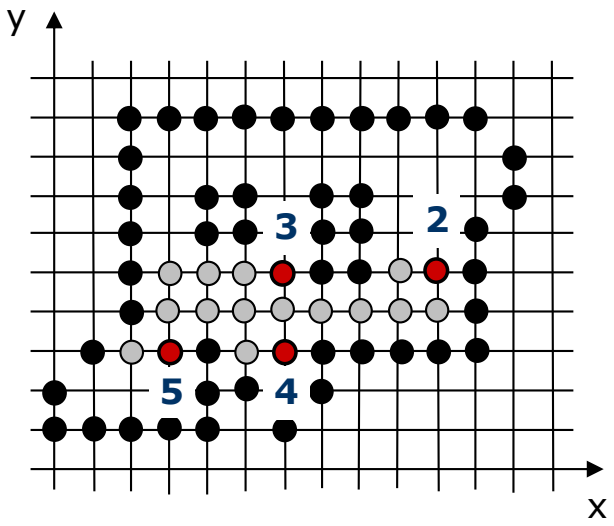
        Write into the stack the course ID

    For all below line courses

        Write into the stack the course ID

}





Questions and proposed problems

1. What is the main difference and similarity between the scan conversion and seed filling algorithms for polygon rendering?
2. Generate in the real-time scan conversion algorithm the edge list for rendering three adjacent triangles. Generate the edge list to keep intersected consecutive edges (Figure 1a, Figure 1b).
3. Is it possible to generate a single list that keep intersected consecutive edges? How many edge lists have to be considered?

Figure 1a

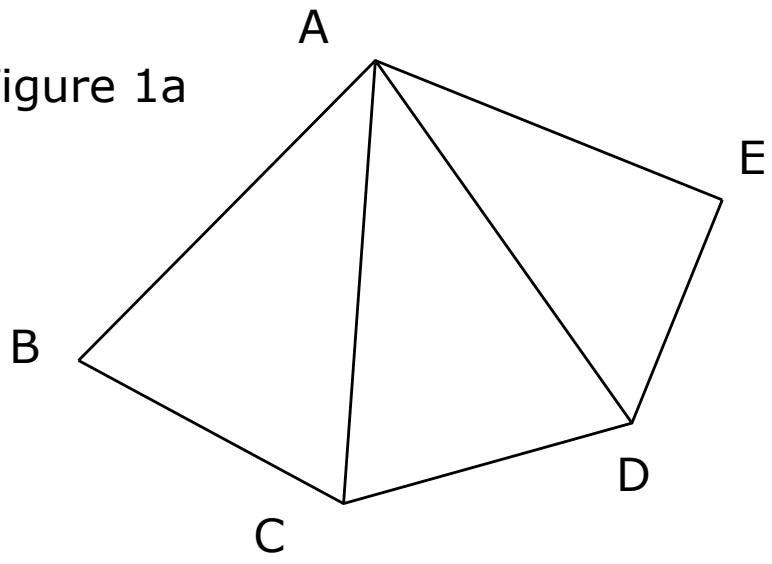
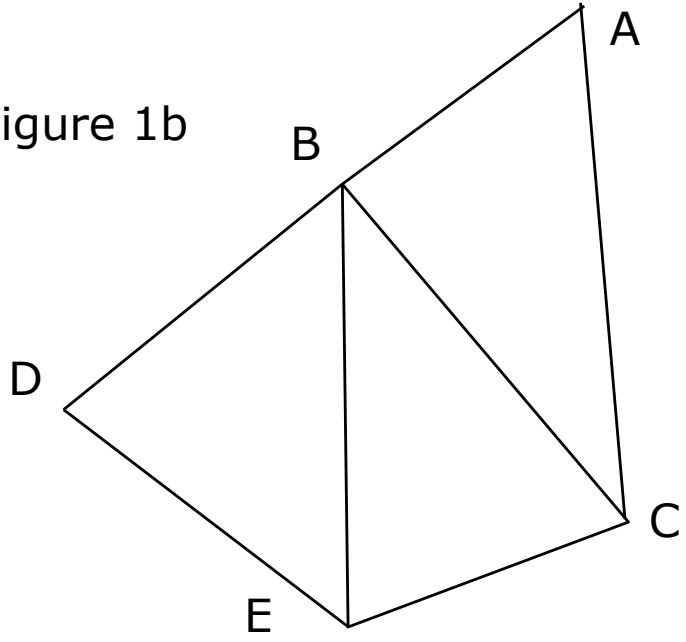


Figure 1b



Questions and proposed problems

4. Explain the generation of the edge list if the scan line is moving upward, instead downward.
5. Explain why the relationship between real polygon point, scan line and pixel position is important?
6. How big could be the polygon rendering error?
7. Explain the Ordered Edge List algorithm on Figure 1a.
8. Explain the Ordered Edge List algorithm on Figure 1b.
9. Explain the Ordered Edge List algorithm on Figure 1a and b, if the scan line is moving upward.
10. Explain how the Ordered Edge List algorithm depends on the upward and downward movement of the scan line?
11. Why would the seed filling algorithm be disadvantageous?
12. What is the worst case for the recursive seed filling algorithm?