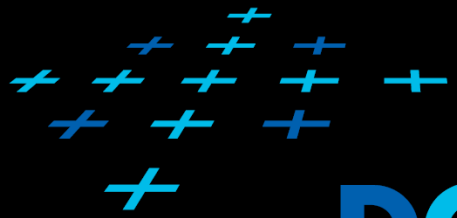


DCGI

KATEDRA POČÍTAČOVÉ GRAFIKY A INTERAKCE

APG – Filling Algorithms

JIŘÍ ŽÁRA

Filling areas

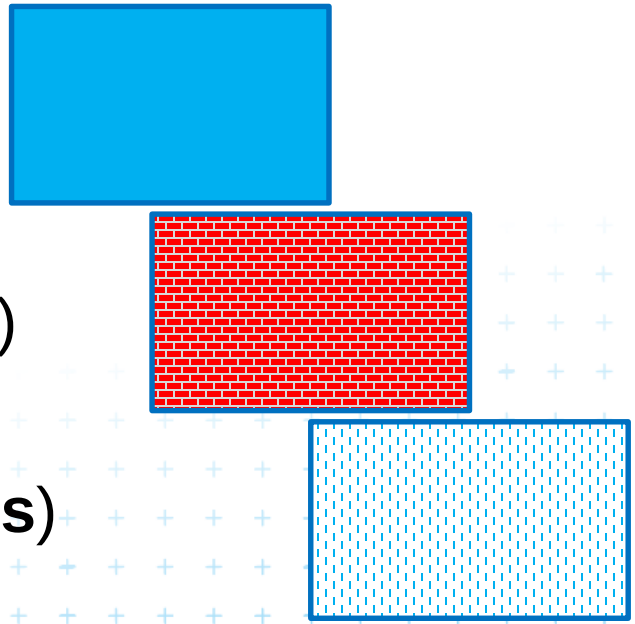
- Outline (border)

What to fill?

1. Defined by **geometry** (polyline)
2. Already stored in a **raster** (image memory)

- Filling

How to fill?



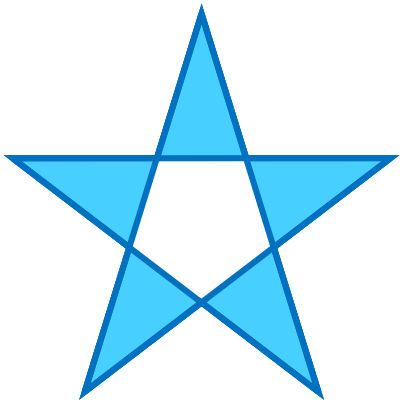
– Solid fill (color)

– Pattern fill (color array)

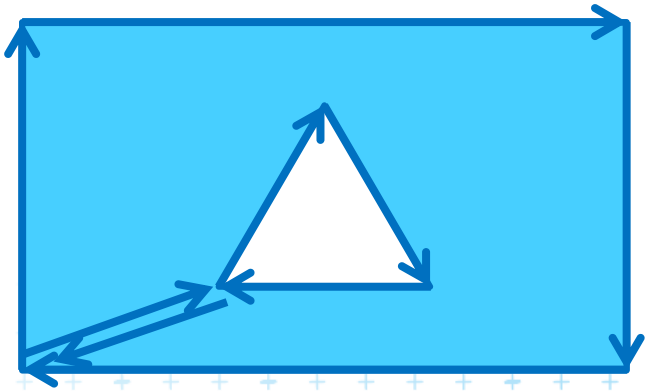
– Hatching (dashed **lines**)

Complex borders

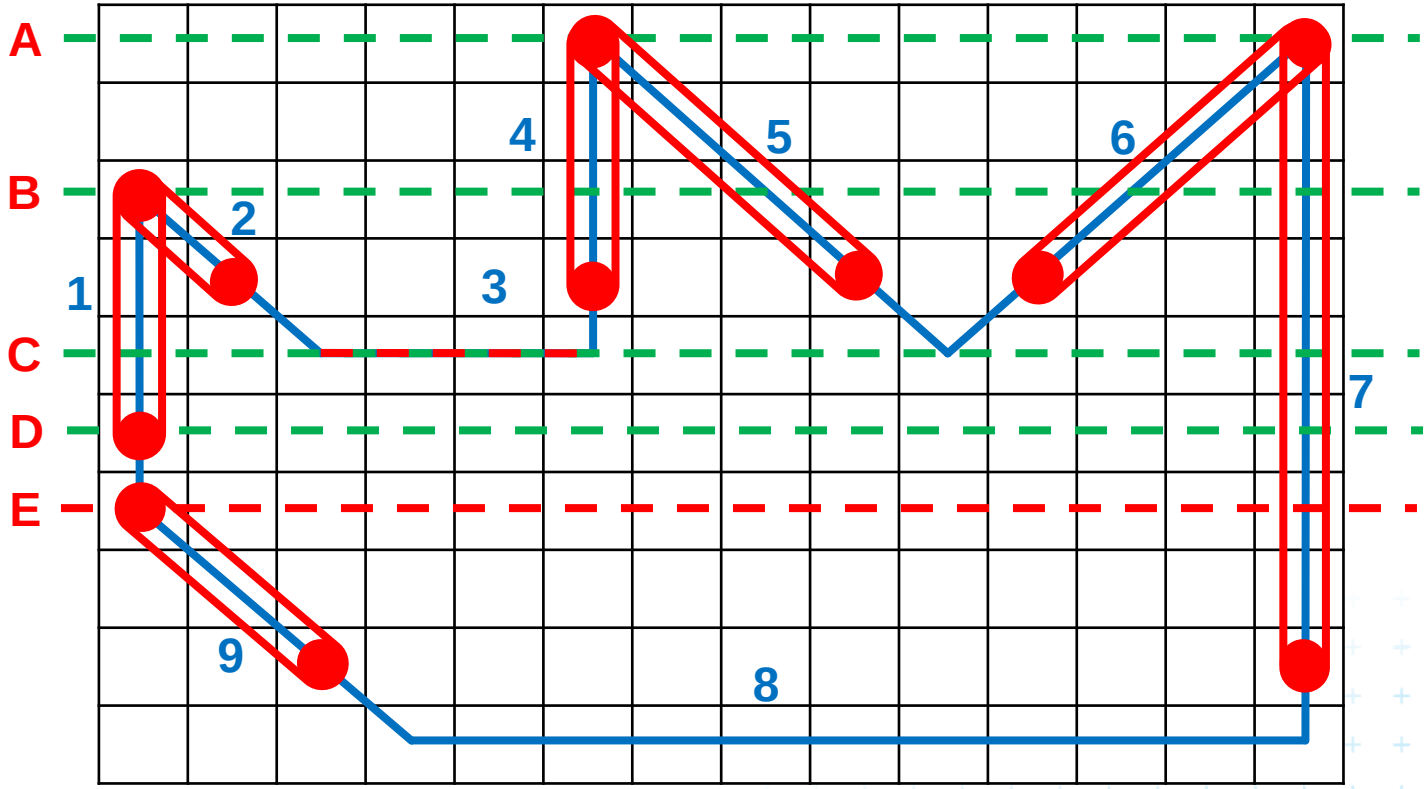
- Self crossing



- Multiple



1. Scan line filling algorithm



A: 4-5, 6-7

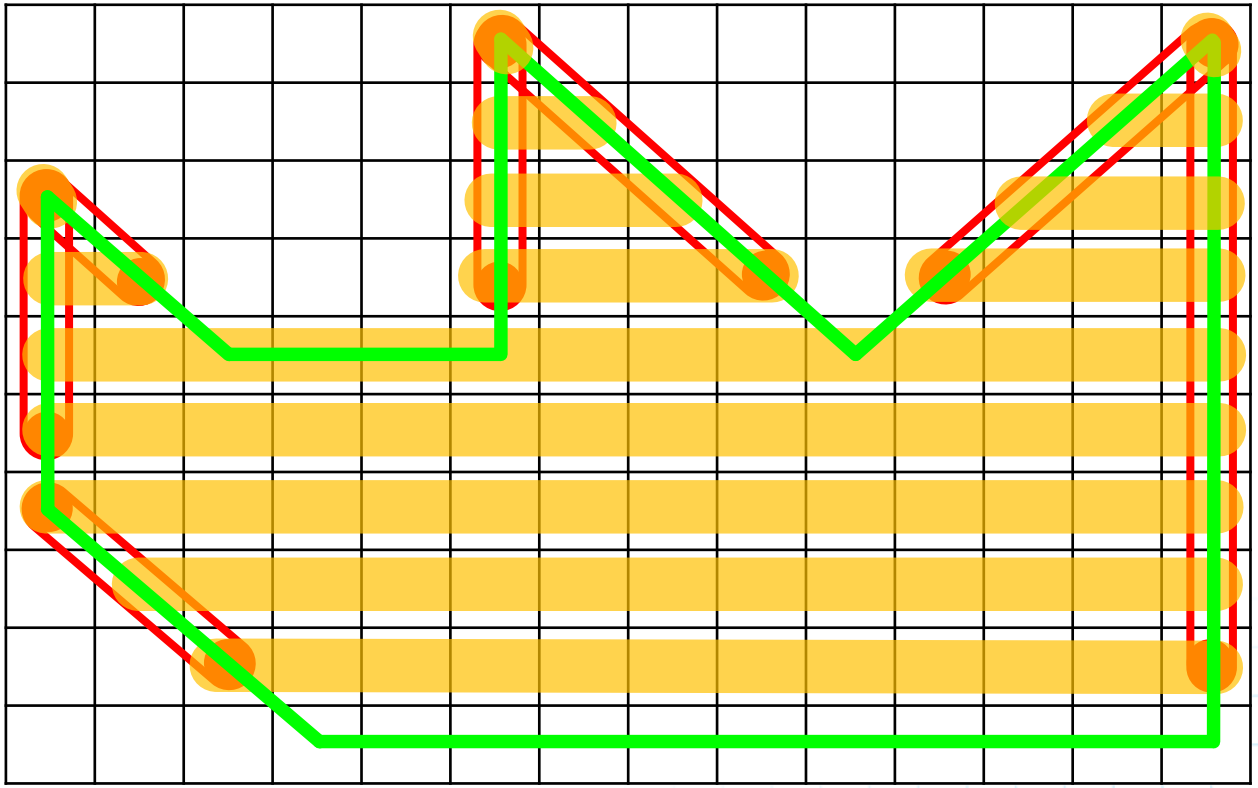
B: 1-2, 4-5, 6-7

C: 1-2, 3, 4-5, 6-7 x

D: 1-7

E: 1-9, 7 ?

Scan line filling algorithm (cont.)



Scan line filling algorithm (cont.)

- Top-down orientation
- Remove horiz. edges
- Edge shortening
- Finding Y_{MAX}, Y_{MIN}

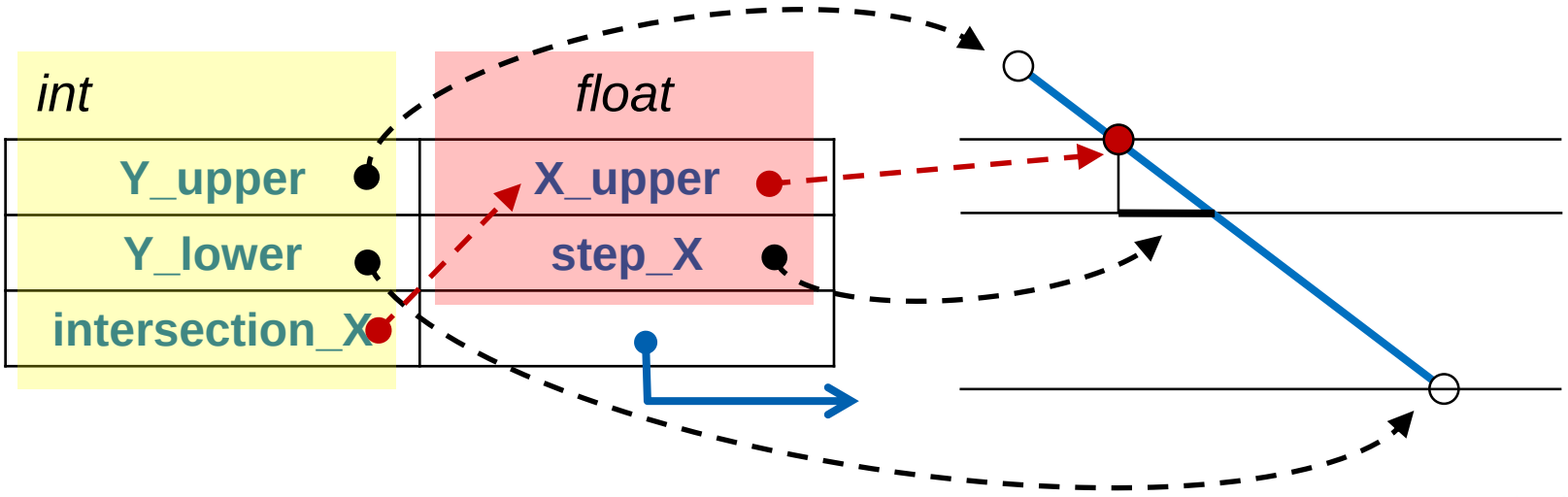
} Initial phase

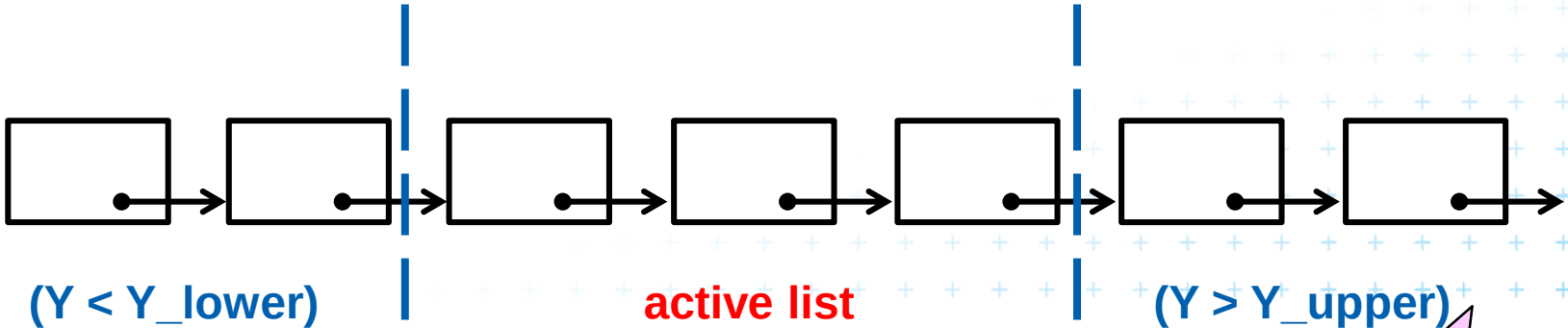
for (int i = Y_{MAX}; i >= Y_{MIN}; i--) {

- Find intersections: scan-line Y + edges // topology
- Line up intersections by X coordinate // sorting
- Draw/fill horizontal segments

}

Data structure: Edge





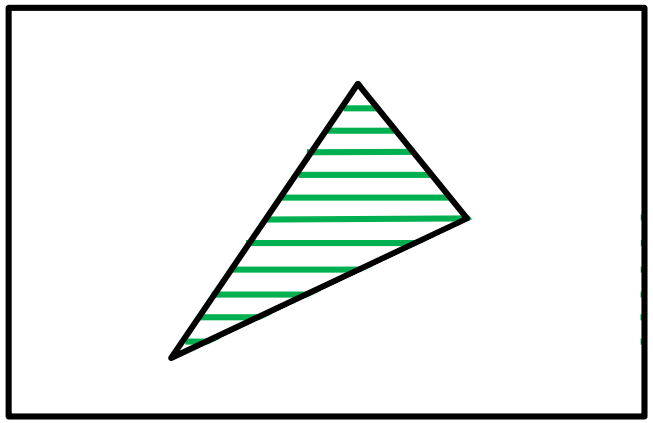
(shake-sort by **intersection_X**)

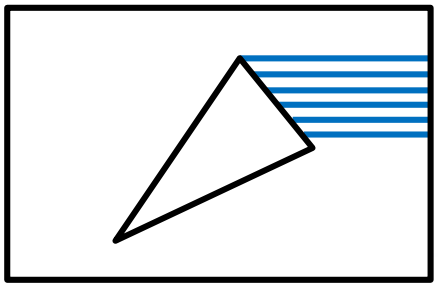
Where is a mistake?

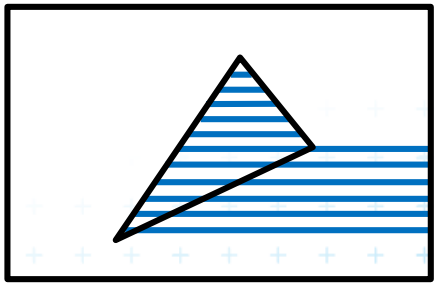
Inverse filling (using XOR logical function)

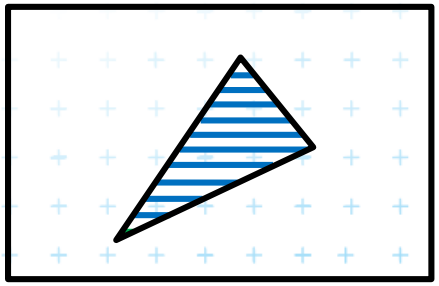
No sorting! ☺

1. Edge shortening (as before)
2. For each edge:
   - Digitizing/Sampling by Δy = 1
   - ***XOR_LINE*** to X_max







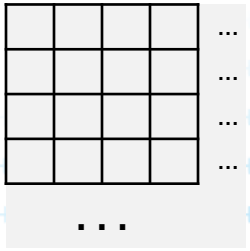


Filled area does not cover previously drawn objects…

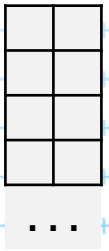
Using stencil buffer

- To improve previous method
- Put values to additional data structure (i.e. stencil buffer) first, then convert/interpret them into image memory.

a) Non convex areas
   - One-bit buffer (for XOR operation results)



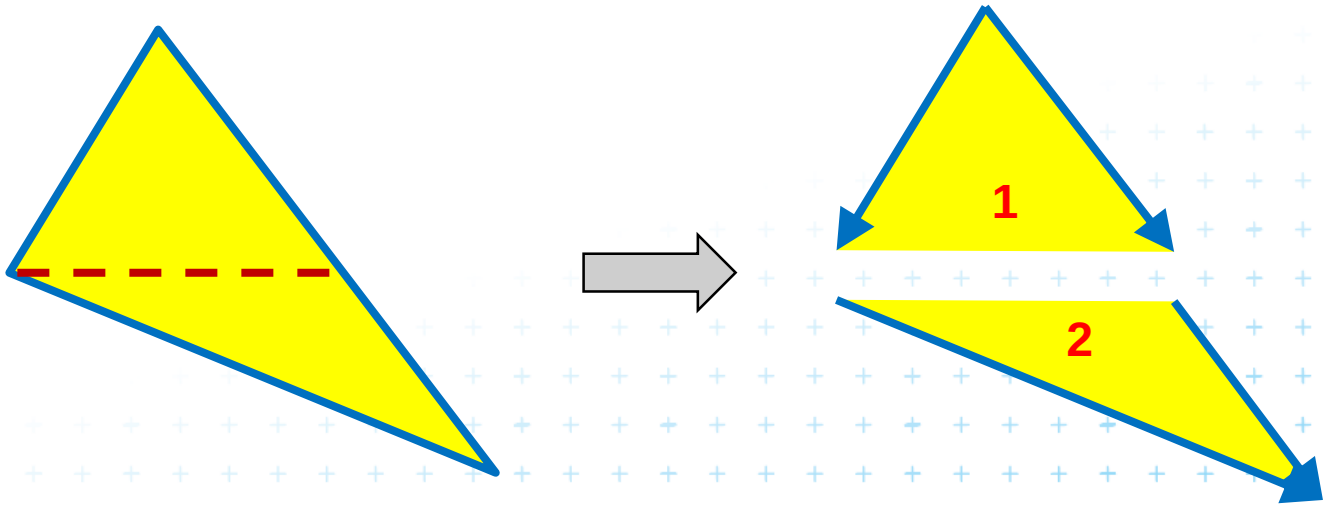
b) Convex areas
   - Array of integer pairs $[X_L,X_R]$ only (initialized by -1 values)



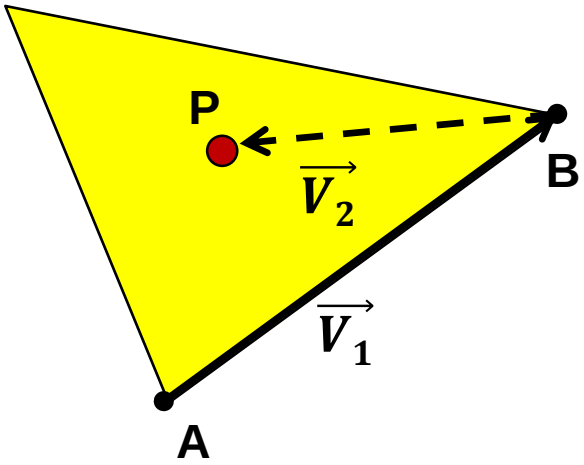
Filled triangle

The most often processed object in 3D graphics

- High speed processing required
- Always convex object ☺
- Decompose it to 2 areas limited by edge pairs
  ⇒ no sorting (preprocessing only)



Juan Pineda: Filling triangle



$\overrightarrow{(x_1, y_1, z_1)} \times \overrightarrow{(x_2, y_2, z_2)}$
= perpendicular vector
(in 3D)

$$z_3 = x_1 \cdot y_2 - x_2 \cdot y_1$$

Positive **z** ⟹ P is on the left from $\overrightarrow{V_1}$

Edge function **E (x_P, y_P)**:

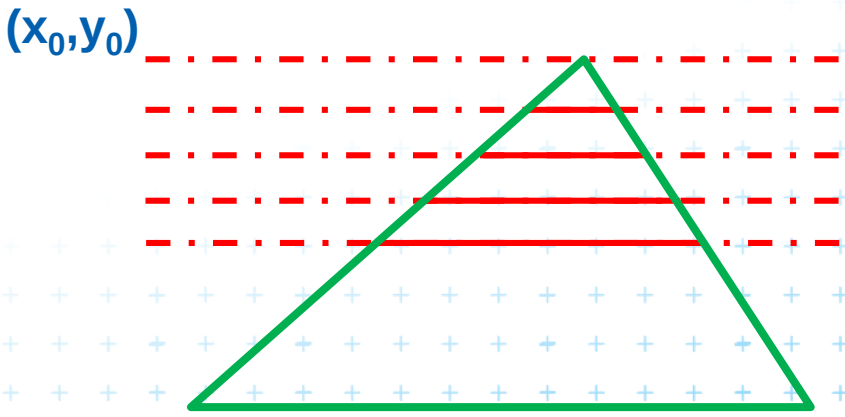
$$E\,(x_P, yP) = \Delta x \cdot (y_P - yB) - \Delta y \cdot (xP - xB)$$

$$\vdots$$

$$E\,(x_P + 1, y_P) = E\,(x_P, y_P) - \Delta y$$

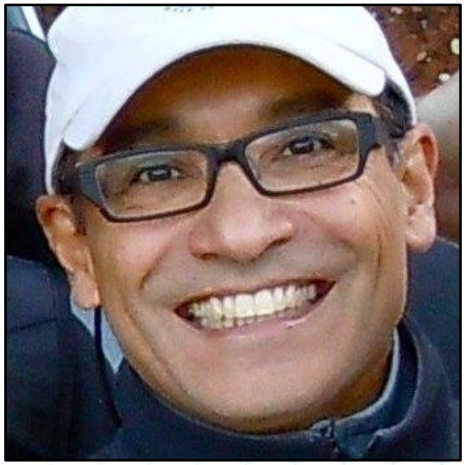
Juan Pineda: Algorithm

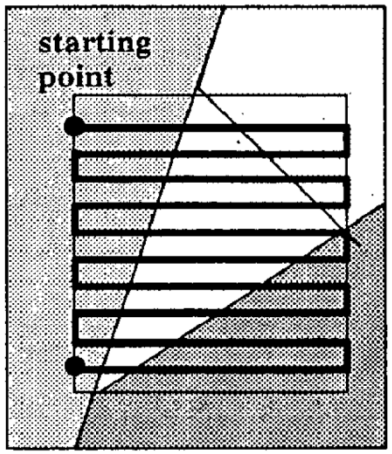
1) Initialization:
   - Compute $E_i(x_0, y_0)$ for each edge *i*
2) Systematically evaluate bounding rectangle:
   - For each pixel
     - Update all three E_i (incrementally)
     - Pixel is inner, when $\forall E_i > 0$



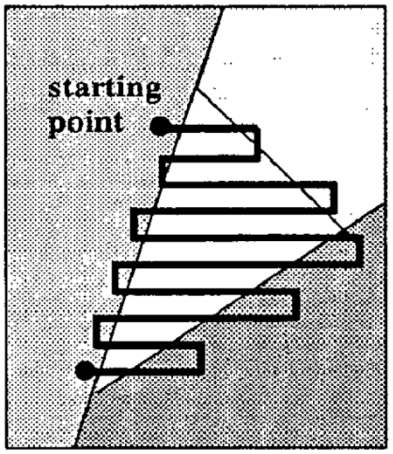
Juan Pineda

- Bc. at MIT 1978 (physics)
- Ms. at Brandeis Univ. 1982
- ***„A Parallel Algorithm for Polygon Rasterization“***
- SIGGRAPH 1988, 4 pages
- Apollo Computer





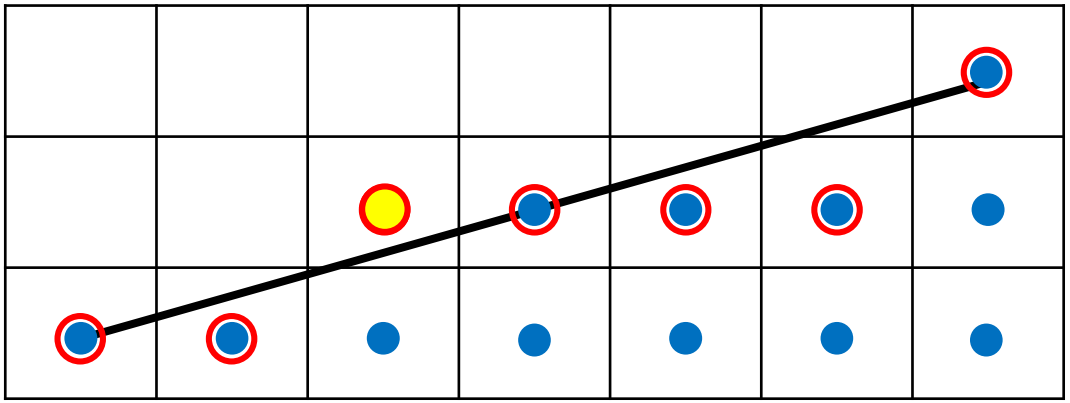
Traversing the Bounding Box



A More Efficient Traversal Algorithm

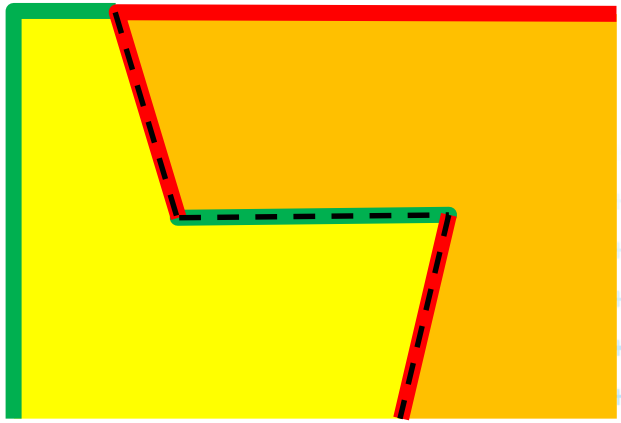
Border issues

- Are border edges covered by filling pixels?



  - **Why?**
  - **When?**

- Who is the owner of border?



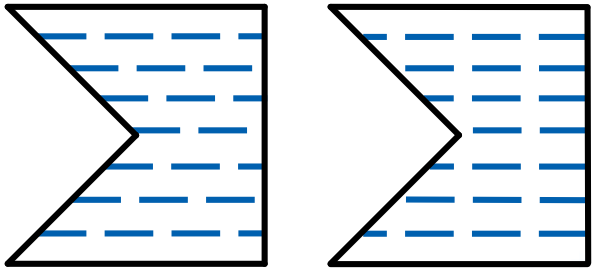
  - Left and upper border belongs to filled area (see slide Nr. 5)
  - Minimizes re-drawing
  - Order independent

Hatching - using lines

1) **Slashed lines**

   a) Rotate border edges (outline) by ***α***
   b) Find horizontal lines (scan-line algorithm)
   c) Rotate horizontal line segments back by ***-α***

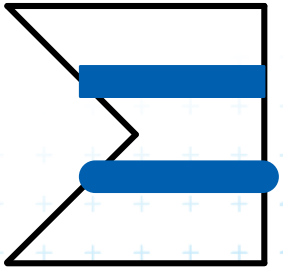
2) **Dashed lines**



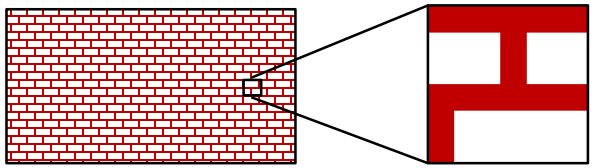
Beginning of dashed lines?

(carpet VERSUS window)

3) **Thick lines**



Hatching – using patterns



Pattern
6x6 pixels

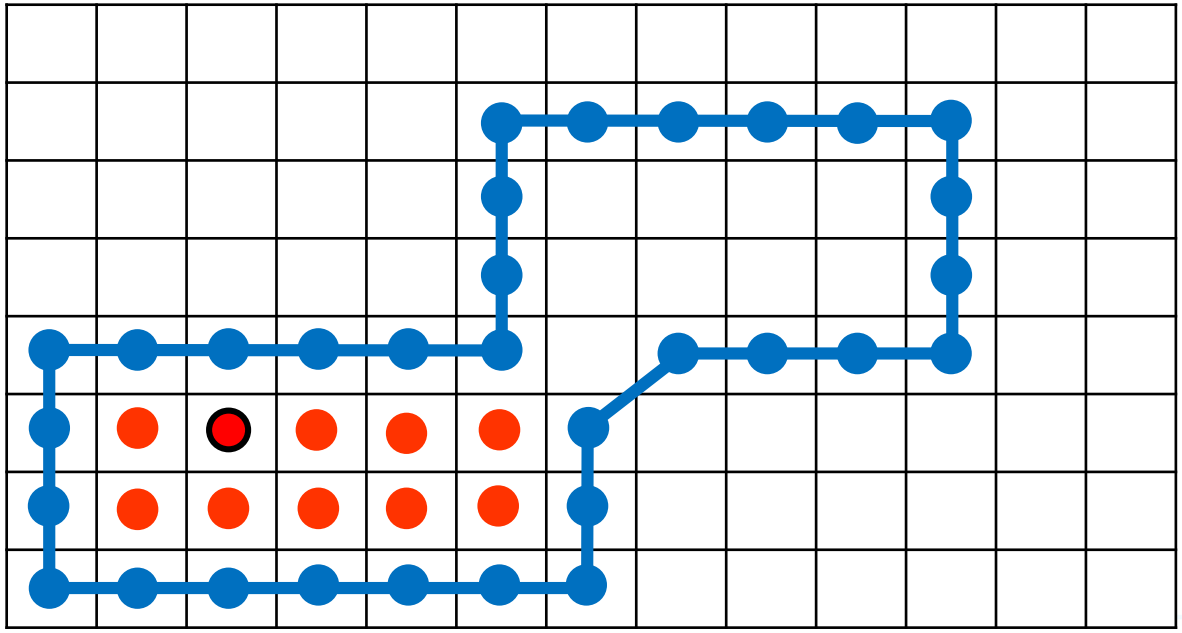
Color **pattern** [dimensionX] [dimesionY];

...

PutPixel (X, Y, **pattern** [X mod dimensionX]
[Y mod dimensionY]);

...

2. Filling raster area (in image memory)



4-connected neighborhood

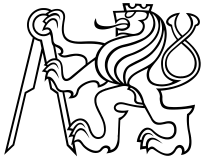
8-connected neighborhood

Seed-fill algorithm

```
void seed (int x, int y) {
    PutPixel (x, y, Fill_Color);
    if (Get_Pixel (x+1, y) != Border_Color &&
        Get_Pixel (x+1, y) != Fill_Color) {
            seed (x+1, y);
    }
    if … (x-1, y) …
    if … (x, y+1) …
    if … (x, y-1) …
}
```

Params: Border_Color, Fill_Color, Seed coordinates

# of recursion calls ~ # of inner pixels!



Scan-line seed-fill algorithm

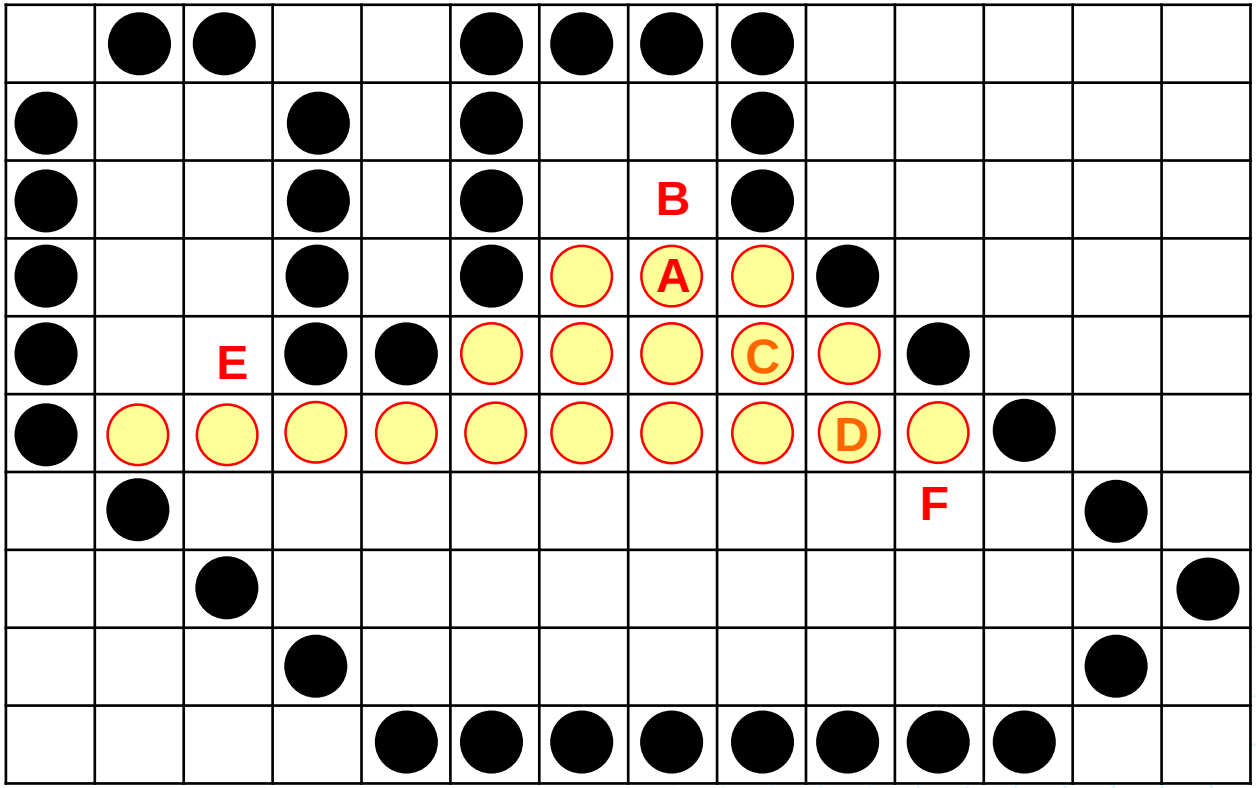
Push (X, Y) into stack;

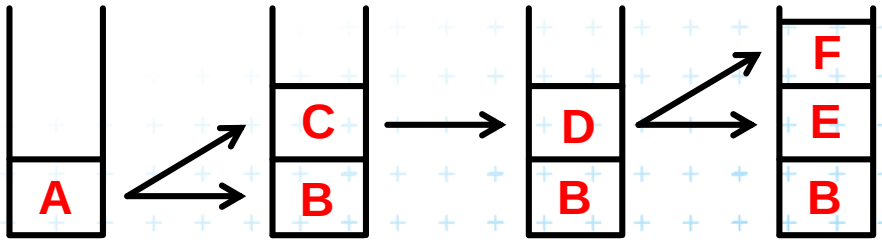
do {

- Pop a **seed** from stack;
- Find left and right border and fill this **horizontal line** segment;
- Analyze pixels **above** this line. For each free segment – push one seed into stack;
- Analyze pixels **below** this line. For each free segment – push one seed into stack;

} while (!stack.*empty*());

Scan-line seed-fill





Thank you for your attention

Jiří Žára, 28.01.2021