DCGI

KATEDRA POČÍTAČOVÉ GRAFIKY A INTERAKCE

Začneme brzy,
We will start soon,
at 16:15 today …

APG
Raster Graphics - Line

JIŘÍ ŽÁRA

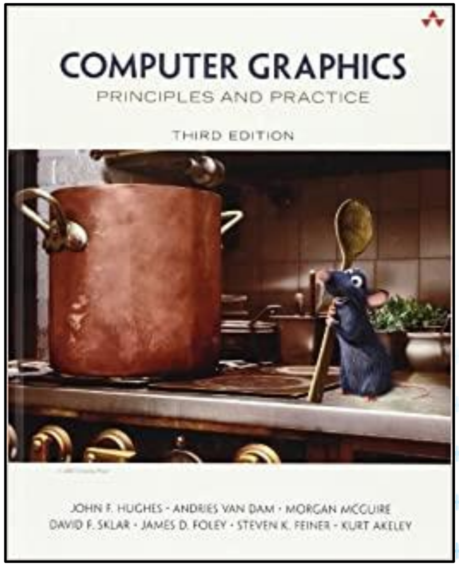
About this course

J. Žára a kol.:
Moderní počítačová grafika

← česky

J. Hughes et al.:
Computer Graphics: Principles and Practice

English →



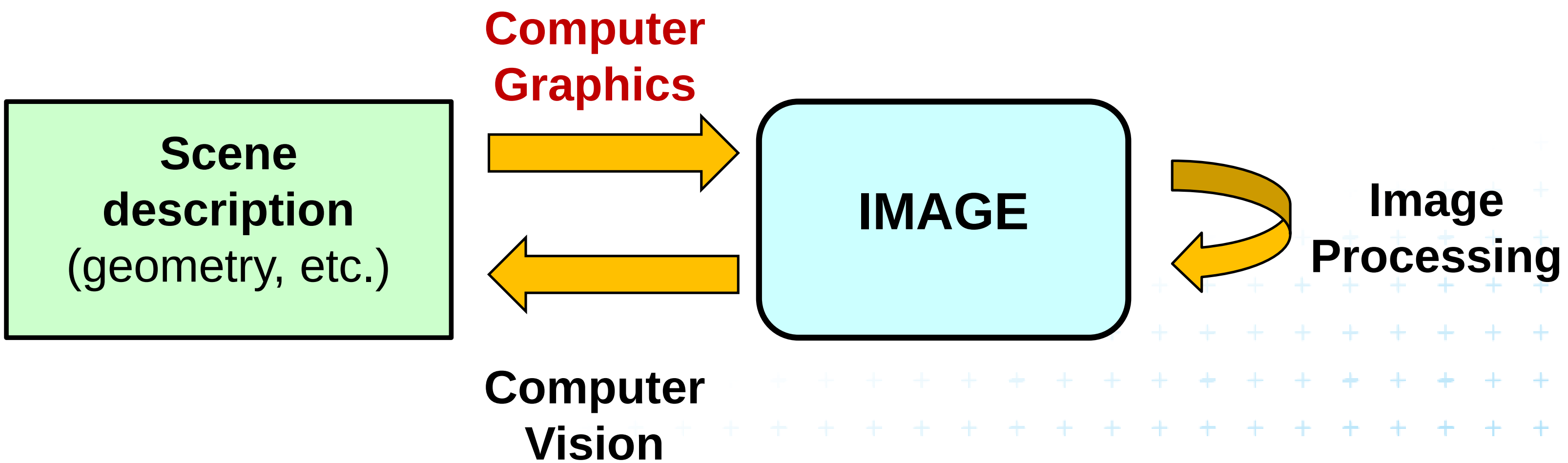
- About 90% of knowledge given in Lectures
- About 10% of knowledge from literature (above)

Contents

- About Computer Graphics
- Raster graphics
- **Line drawing**
  - **DDA algorithm**
  - **Bresenham algorithm**
- Dashed line
- Thick line
- Antialiasing

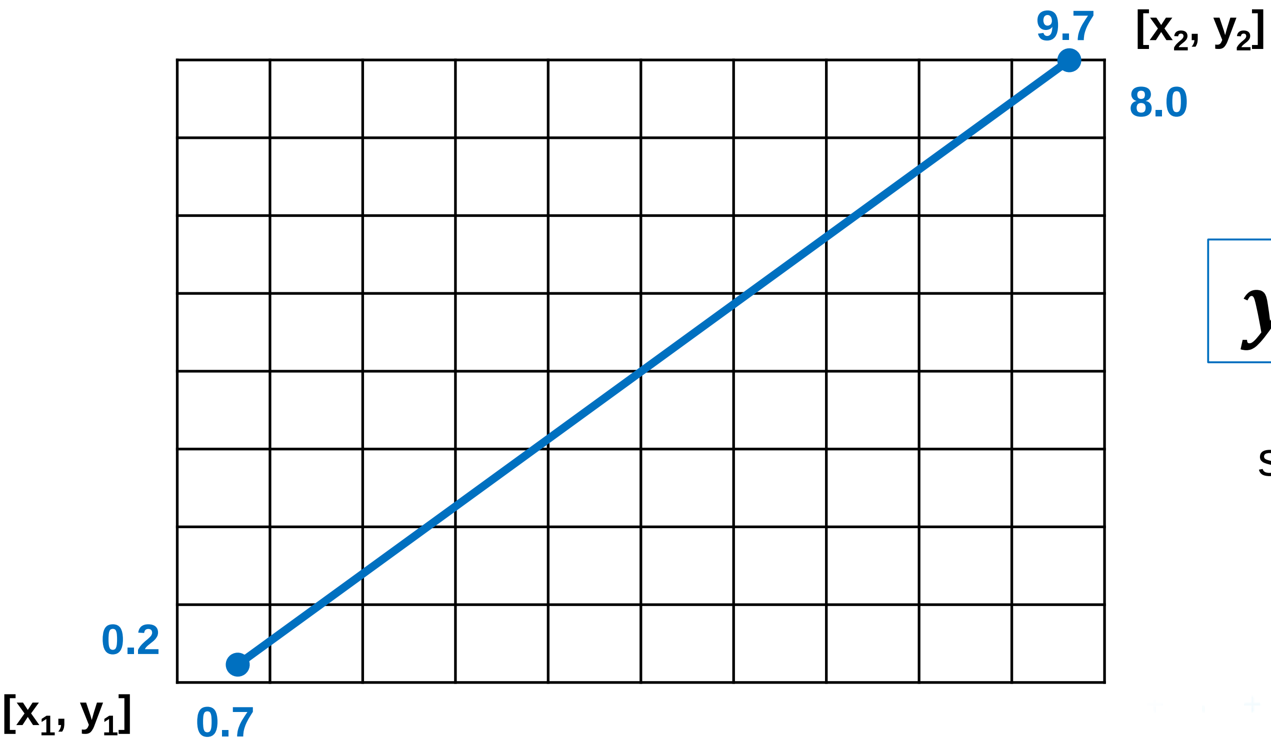
Computer Graphics

- Vector VERSUS **Raster**
- Conversion from abstract description into digital form (pixels) = **digitization** (2D), **rendering** (3D)



Computer Graphics
Image Processing
Computer Vision

Raster



$$y = k \cdot x + b$$

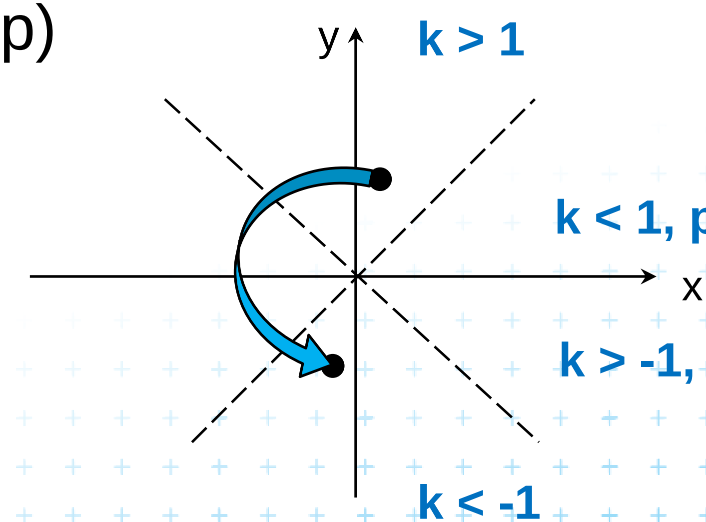
slope

shift, intercept

$$k = \frac{\Delta y}{\Delta x} = (y_2 - y_1)/(x_2 - x_1)$$

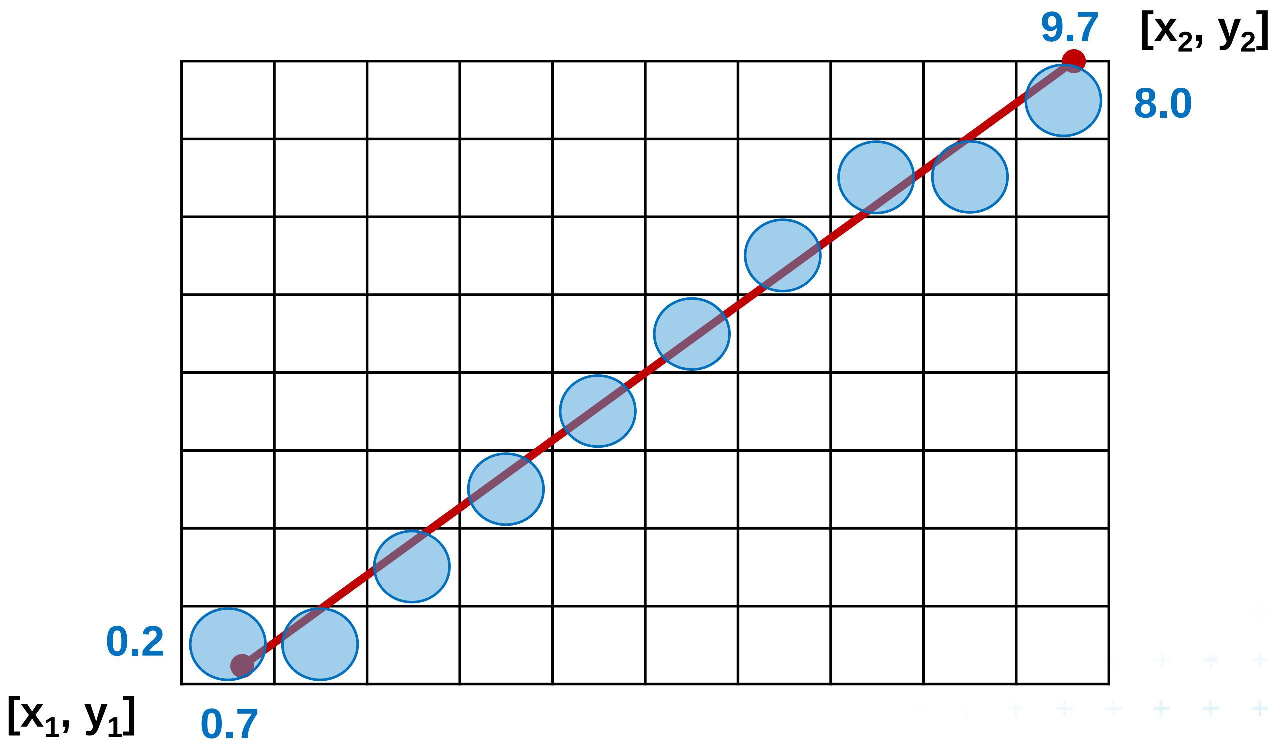
Line

- Mathematics – infinitely thin line
- Computer Graphics
  - – Sequence of neighboring pixels (*pixel, px, picture element*)
  - – Digitization = sampling of a continuous function
- **Slope** defines a driving/major axis for sampling (by 1 pixel step)



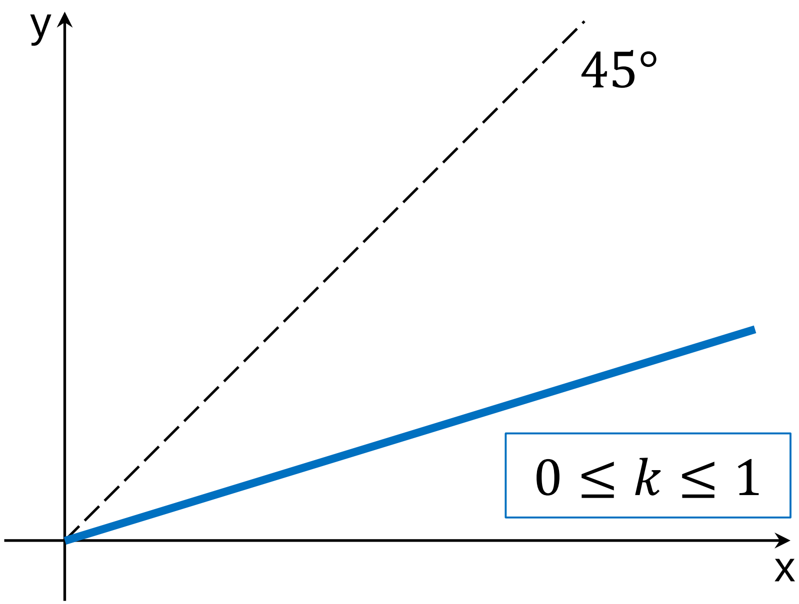
$$\boxed{k = \frac{\Delta y}{\Delta x}}$$

Line drawing methods



- DDA (Digital Differential Analyzer) [float]
- Bresenham algorithm [int]

DDA (Digital Differential Analyzer)



$$stepX = 1$$

$$stepY = \frac{\Delta y}{\Delta x} = k$$

$$x_{i+1} = x_i + 1$$
$$y_{i+1} = y_i + k$$

Note: k is a real (float) number

DDA algorithm

```
DDA (int x1, int y1, int x2, int y2) {
    double k, Y;
    k = (y2 – y1) / (x2 - x1);
    setPixel (x1, y1);
    Y = y1;
    for (int i = x1+1; i<=x2; i++) {
        Y += k;
        setPixel (i, round(Y));
    }
}
```

$k \in \langle 0, 1 \rangle$

$x1 < x2$

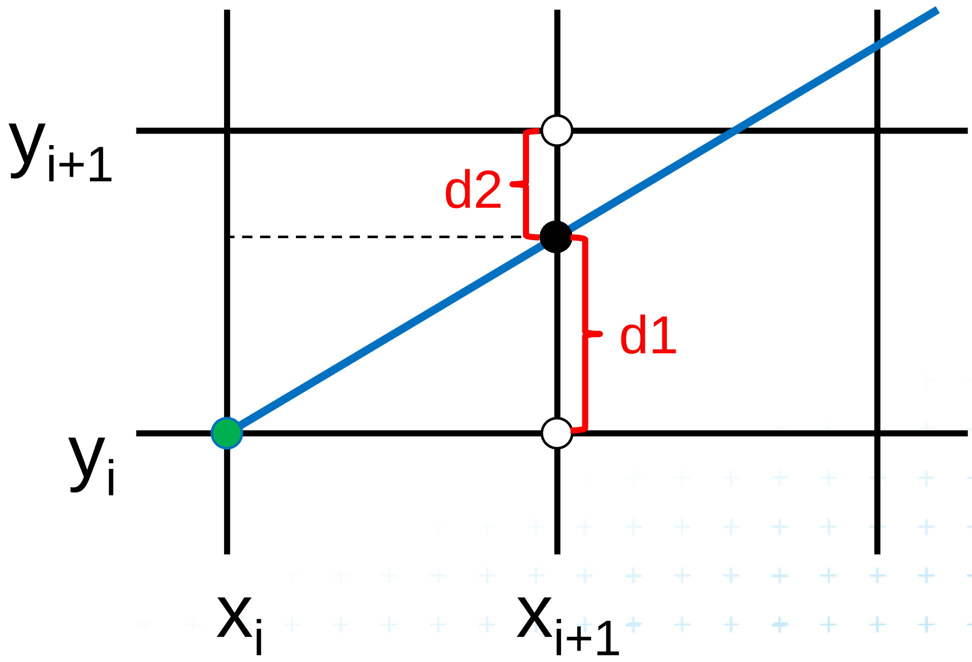
Endpoints preprocessing:

- Coordinate rounding (to int)
- Driving axis determination
- Orientation (from left/bottom)

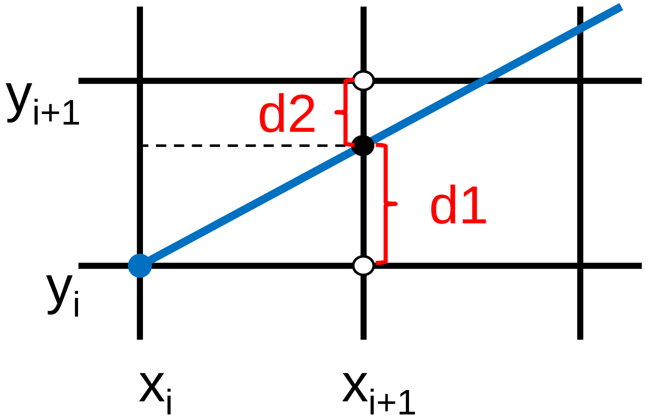
Jack Elton Bresenham, *1937

Integer only algorithm

- IBM, developed in 1962, published in 1965



Bresenham algorithm (1/2)



$$y_{real} = k\,(x_i + 1) + b$$

$$d_1 = y_{real} - y_i$$

$$d_2 = y_i + 1 - y_{real}$$

$$d = d_1 - d_2 = 2k \cdot (x_i + 1) - 2y_i + 2b - 1$$

$$k = \frac{\Delta y}{\Delta x}$$

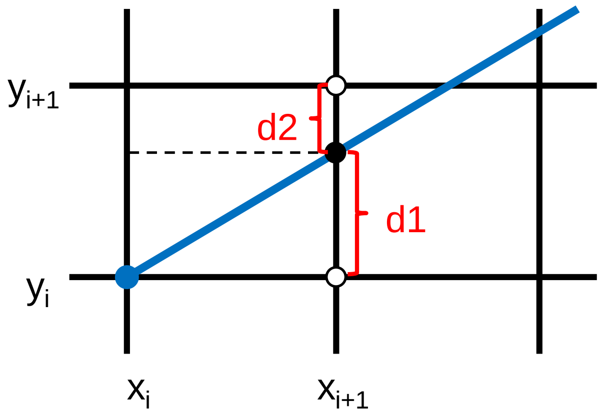
$$p_i = \Delta x \cdot (d_1 - d_2)$$

$$= 2\Delta y \cdot x_i - 2\Delta x \cdot y_i + \underbrace{2\Delta y + \Delta x \cdot (2b - 1)}_{\text{Const.}}$$

$$\boxed{p_{i+1} = p_i + 2\Delta y - 2\Delta x\,(y_{i+1} - y_i)}$$

prediction

Bresenham algorithm (2/2)



$[x_i, y_i, p_i] \Rightarrow [x_{i+1}, y_{i+1}, p_{i+1}]$

$x_{i+1} = x_i + 1$

$p_{i+1} = p_i + 2\Delta y - 2\Delta x\,(y_{i+1} - y_i)$

$\mathbf{p_i}$

≥ 0:

$y_{i+1} = y_i + 1$

$p_{i+1} = p_i + 2\Delta y - 2\Delta x$

< 0:

$y_{i+1} = y_i$

$p_{i+1} = p_i + 2\Delta y$

Note: $p_0 = 2\Delta y - \Delta x$

from $p_i = \ldots$, where $x_0 = y_0 = b = 0$

Bresenham algorithm – code

```
Bresenham (int x1, int y1, int x2, int y2) {
        int c0, c1, p;
        init (c0, c1, p);
        setPixel (x1, y1);
        for (int i = x1 + 1; i <= x2; i++) {
            if (p < 0) {
                    p += c0;
            } else {
                    p += c1;
                    y1++;
            }
            setPixel (i, y1);
        }
}
```

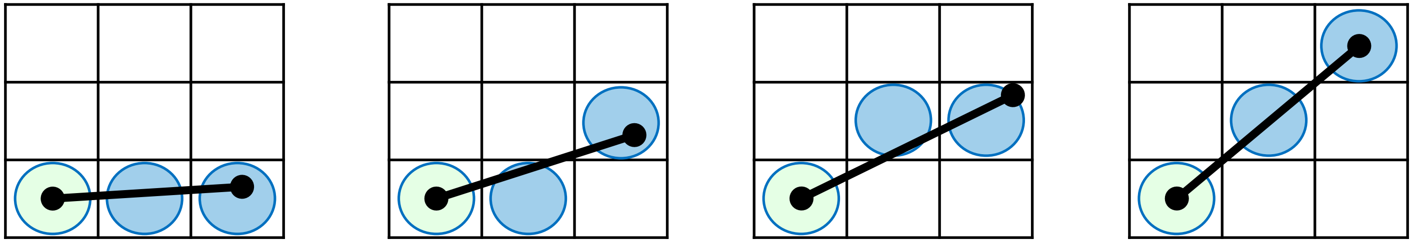
```
init (int c0, int c1, int p) {
        c0 = 2 · (y2 – y1);
        c1 = c0 – 2 · (x2 – x1);
        p = c0 – (x2 – x1);
}
```

$k \in \langle 0, 1 \rangle$

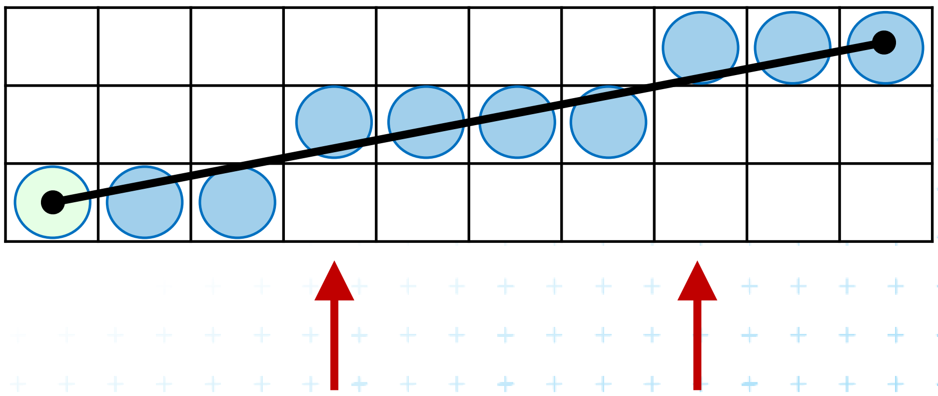
$x1 < x2$

Multi-step methods – interesting research

- Pairs/triplets of pixel



- Computing distance to minor axis change



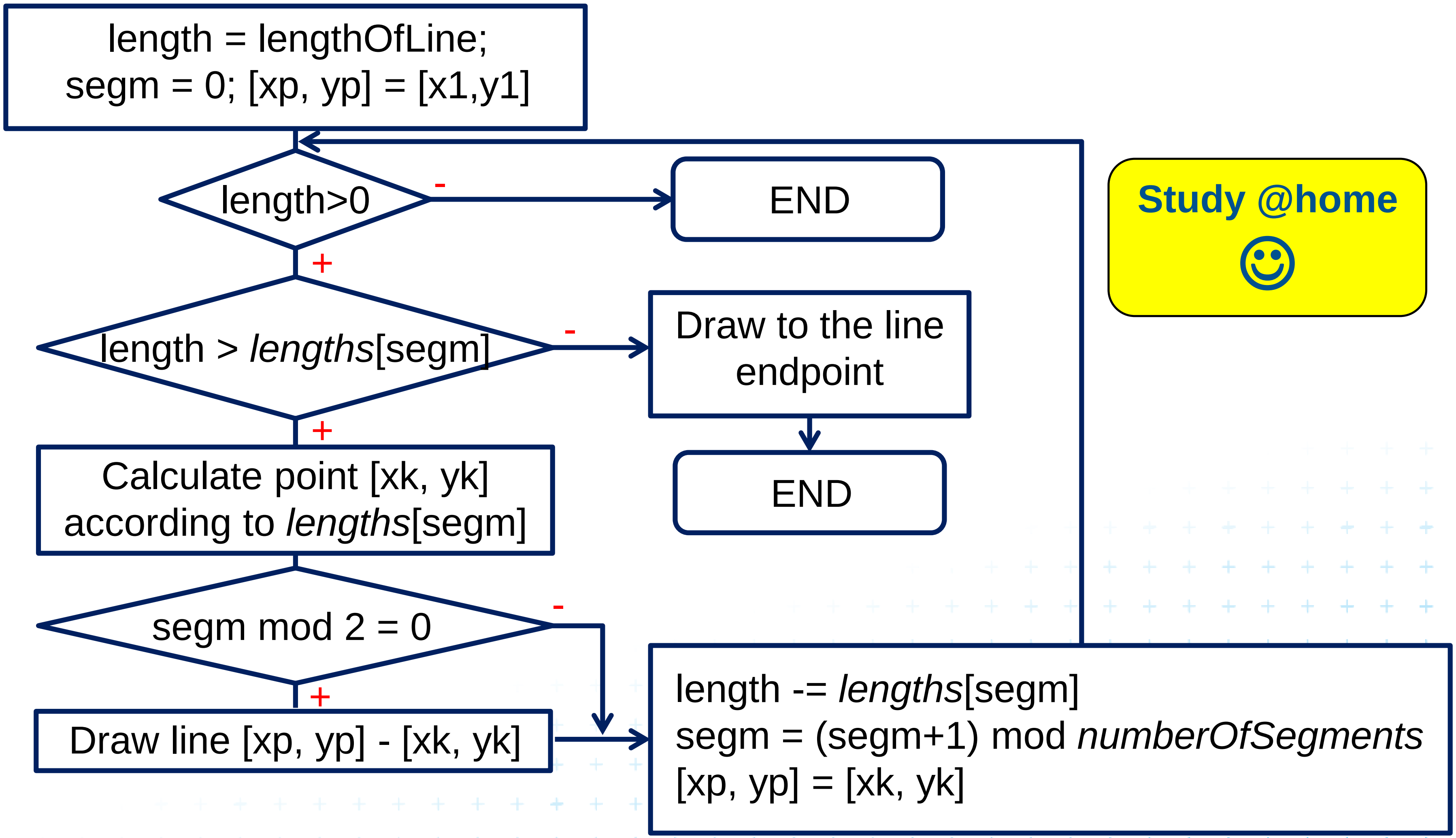
Dashed line

- Line appearance defined in a length segment array
- Odd segment drawn, even skipped

```
class DashedLine {
        int numberOfSegments;        // e.g. 6
        int [ ] lengths;
}
```

- Two possible approaches:
  1. By individual segments
  2. The whole line at once (modified Bresenham alg.)

1. Drawing by segments



Study @home ☺

2. Drawing the whole line at once

- Bresenham algorithm modification
- Blocking setPixel() according to odd/even segment:

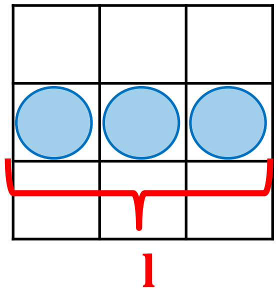
  a) int segm; int segmLength;

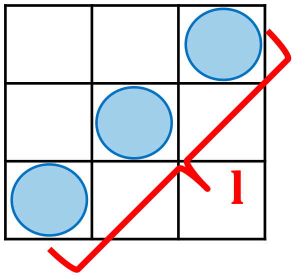
  b) segm mod 2 ⟹ enable/disable setPixel()

  c) decrement segmLength, when 0 ⟹ prepare for next segment

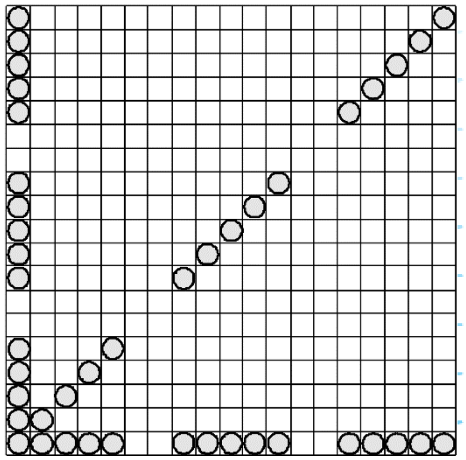
Problems with raster metric

- length VERSUS Nr. of pixels
- Oblique lines appear thinner





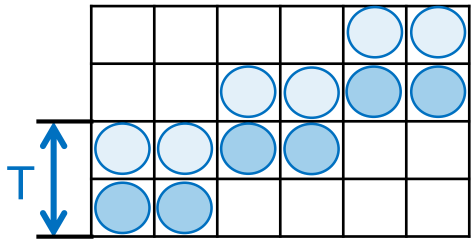
- Modified Bresenham = bad approach
- By segments = good technique



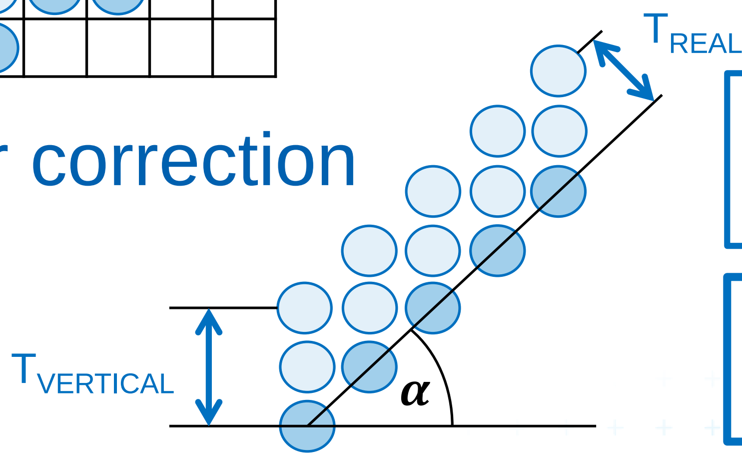
Thick line

a) Trivial solution – Bresenham modified

setPixel ⟹ T × setPixel (in a proper direction)



b) Angular correction



$$\boldsymbol{T = T_{REAL} = T_{VERTICAL} \cdot \cos \alpha}$$

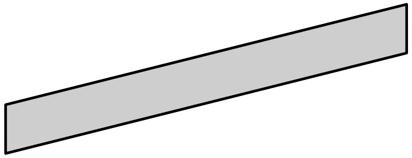
$$\boldsymbol{T_{VERTICAL} = \frac{T}{\cos \alpha}}$$

c) Filling boundary line (outline) = correct solution

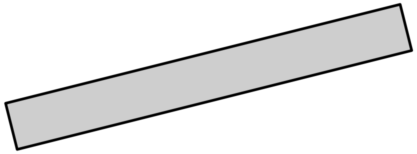
Outline computation + polygon filling (see next lectures)

Thick lines ending

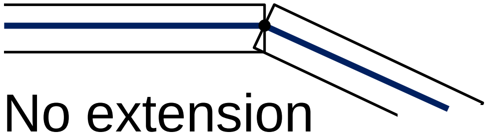
- a) Bresenham



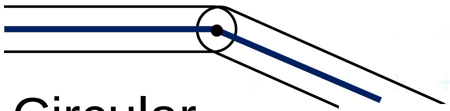
- b) Fill area



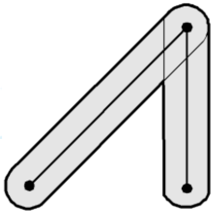
- Ending types:

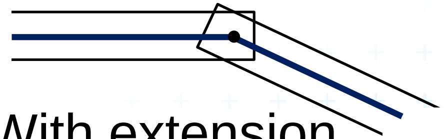


1) No extension



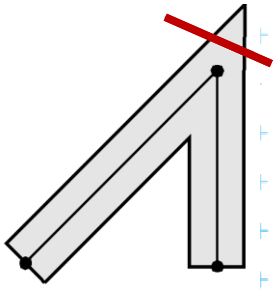
3) Circular extension





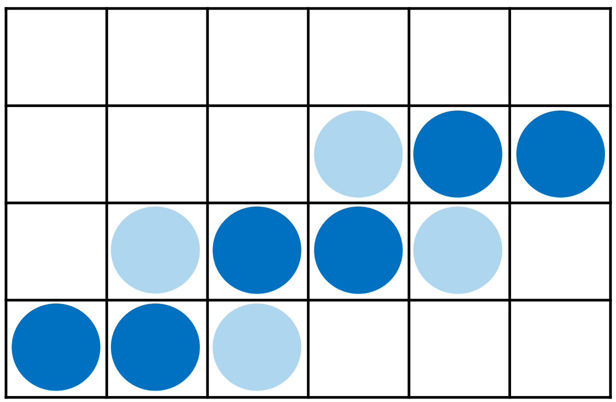
2) With extension

4) General outline



Aliasing & Antialiasing of lines

- Aliasing = causes jaggy lines (result of **subsampling**)
- Antialiasing = smoothing (via pixel **intensities**)



Aliasing

Antialiasing

Antialiasing

- **Local (Line) antialiasing**
  - when individual line is drawn
- **Global (Full screen) antialiasing**
  - after the whole image/screen is generated
  - image processing technique (filtering)

Global antialiasing

- Input = image memory
- Final pixel intensity influenced by neighboring intensities

Filter function H (kernel)

$$I'(x,y) = H(i,j) \cdot I(x+i, y+j)$$

$$H(i,j) = \begin{array}{c|c} {}^{i}/_{j} & \begin{array}{ccc} \mathbf{-1} & \mathbf{0} & \mathbf{1} \end{array} \\ \begin{array}{c} \mathbf{-1} \\ \mathbf{0} \\ \mathbf{1} \end{array} & \begin{bmatrix} \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\ \frac{2}{16} & \frac{\mathbf{4}}{16} & \frac{2}{16} \\ \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \end{bmatrix} \end{array}$$

Thank you for your attention

Jiří Žára, 23.09.2020

… and study “**Circle Digitizing**” methods @home