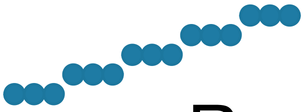
Einführung in Visual Computing

186.822

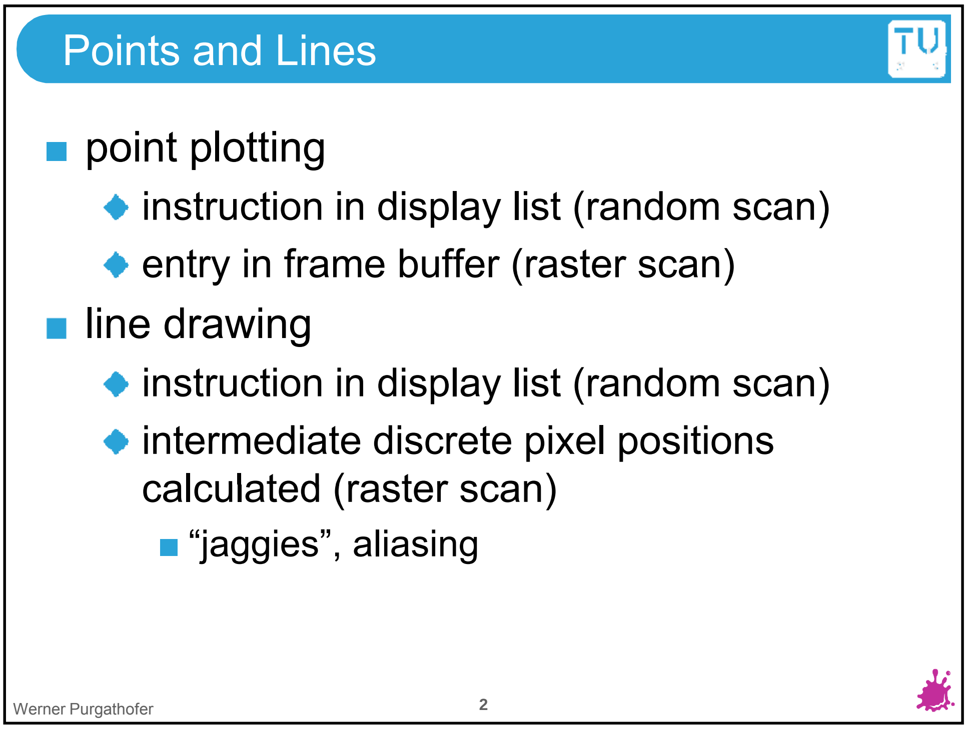


Rasterization

Werner Purgathofer

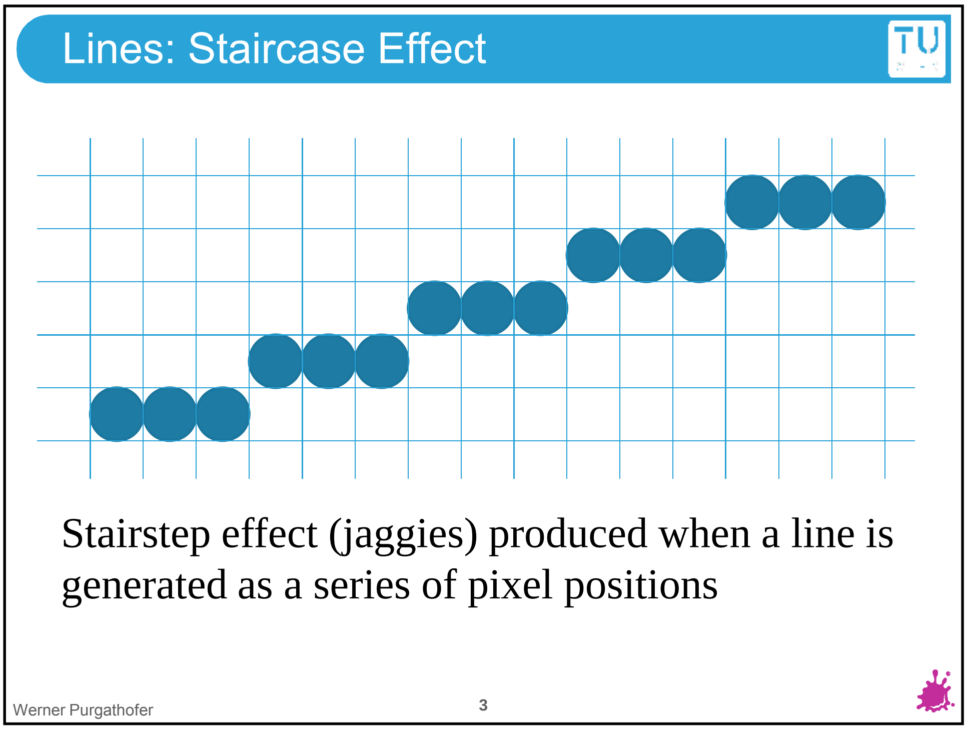
Important Graphics Output Primitives

- in 2D
  - points, lines
  - polygons, circles, ellipses & other curves (also filled)
  - pixel array operations
  - characters
- in 3D
  - triangles & other polygons
  - free form surfaces
- + commands for properties: color, texture, …



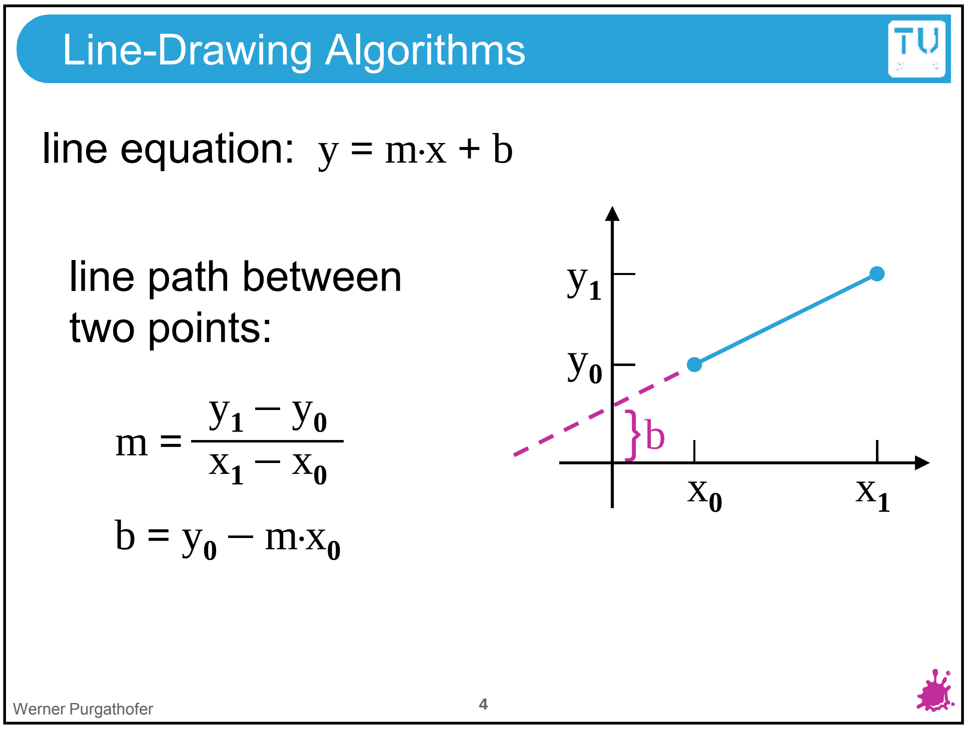
Points and Lines

- point plotting
  - instruction in display list (random scan)
  - entry in frame buffer (raster scan)
- line drawing
  - instruction in display list (random scan)
  - intermediate discrete pixel positions calculated (raster scan)
    - “jaggies”, aliasing



Lines: Staircase Effect

Stairstep effect (jaggies) produced when a line is generated as a series of pixel positions



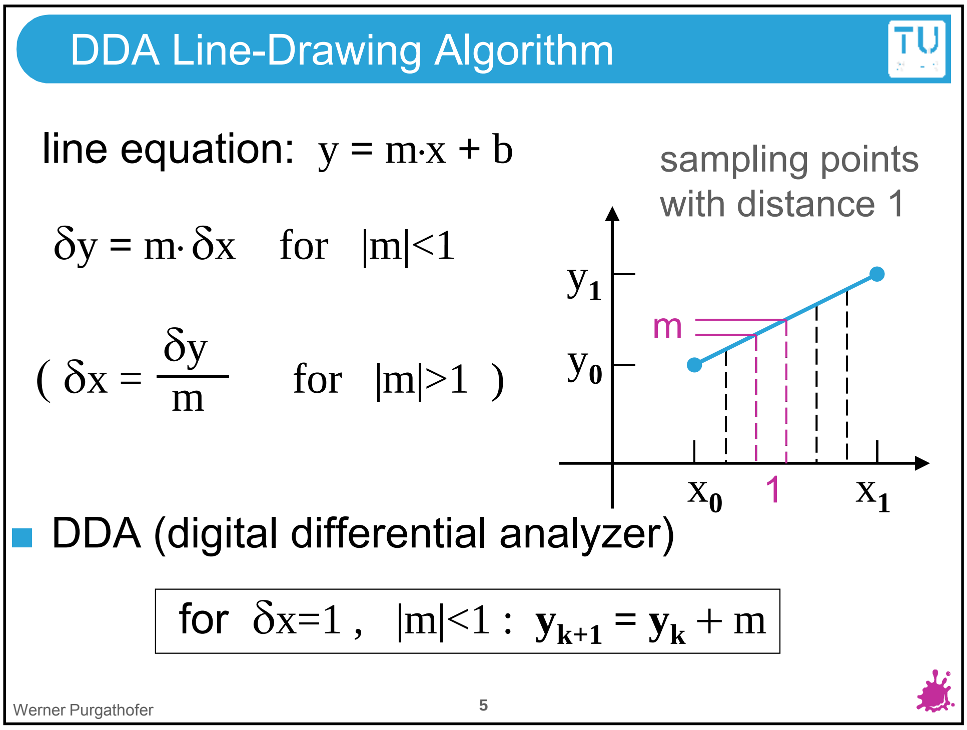
Line-Drawing Algorithms

line equation: $y = m{\cdot}x + b$

line path between two points:

$$m = \frac{y_1 - y_0}{x_1 - x_0}$$

$$b = y_0 - m{\cdot}x_0$$



DDA Line-Drawing Algorithm

line equation: $y = m{\cdot}x + b$

$\delta y = m{\cdot}\,\delta x \quad$ for $\quad |m|<1$

$(\ \delta x = \dfrac{\delta y}{m} \quad$ for $\quad |m|>1 \ )$

sampling points with distance 1

- DDA (digital differential analyzer)

$$\boxed{\text{for } \ \delta x=1\,, \quad |m|<1 : \ \mathbf{y_{k+1}} = \mathbf{y_k} + m}$$

DDA – Algorithm Principle

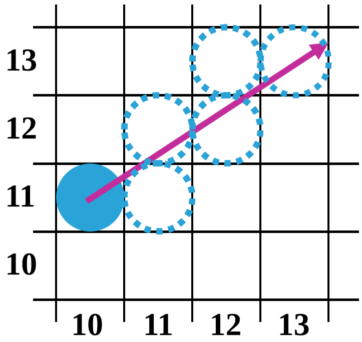
```
dx = x1 – x0; dy = y1 – y0;
m = dy / dx;

x = x0;  y = y0;
drawPixel (round(x), round(y));

for (k = 0; k < dx; k++)
  { x += 1; y += m;
  drawPixel (round(x), round(y)}
```

extension to other cases simple

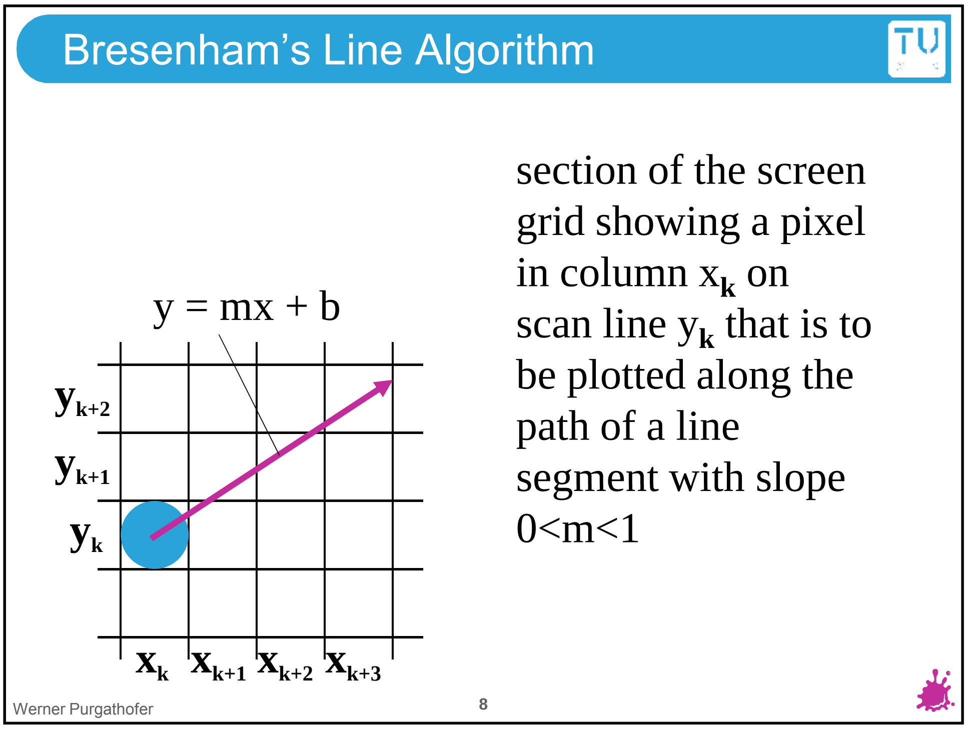
Bresenham's Line Algorithm

- faster than simple DDA
  - incremental integer calculations
  - adaptable to circles, other curves

$y = m\cdot(x_k + 1) + b$

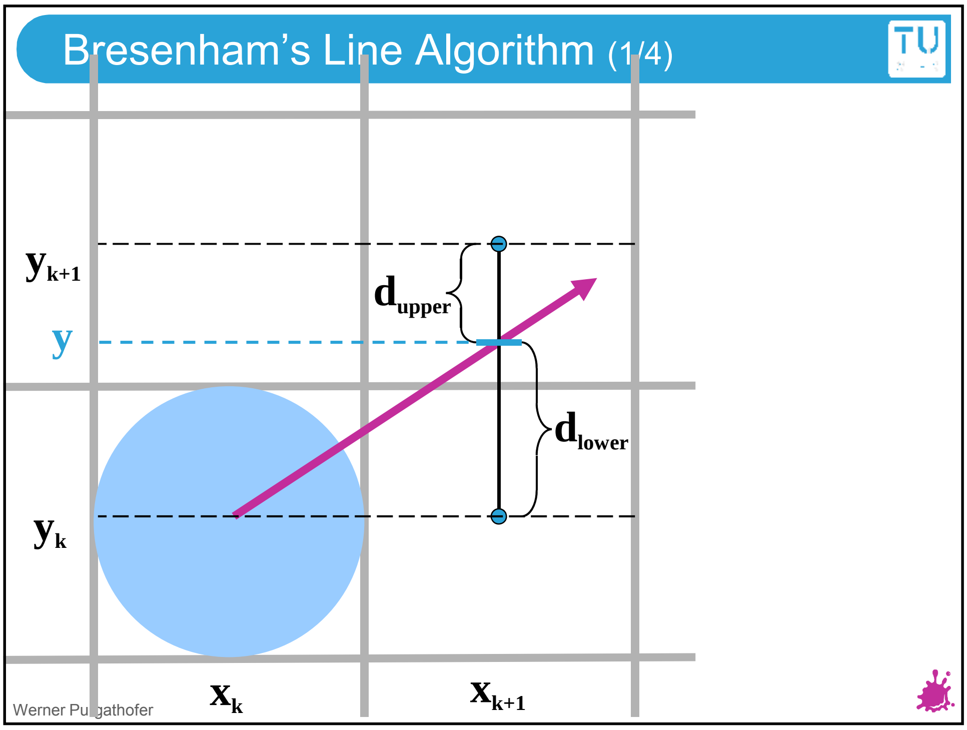


for every column it is decided which of the two candidate pixels is selected



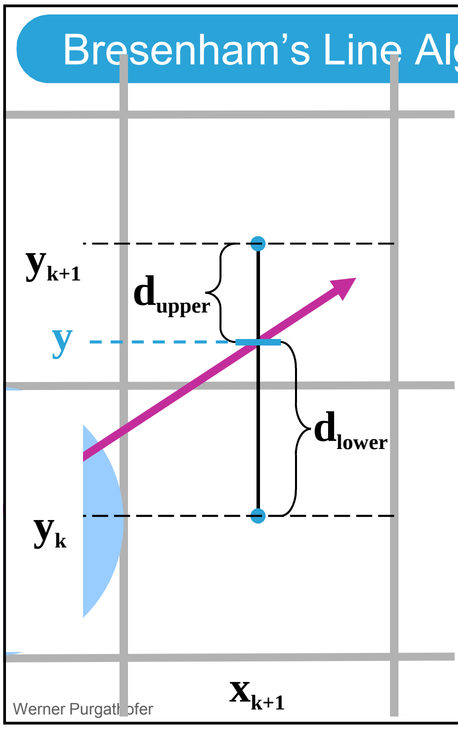
Bresenham’s Line Algorithm

section of the screen grid showing a pixel in column x_k on scan line y_k that is to be plotted along the path of a line segment with slope $0<m<1$



Bresenham’s Line Algorithm (1/4)

Bresenham’s Line Algorithm (1/4)



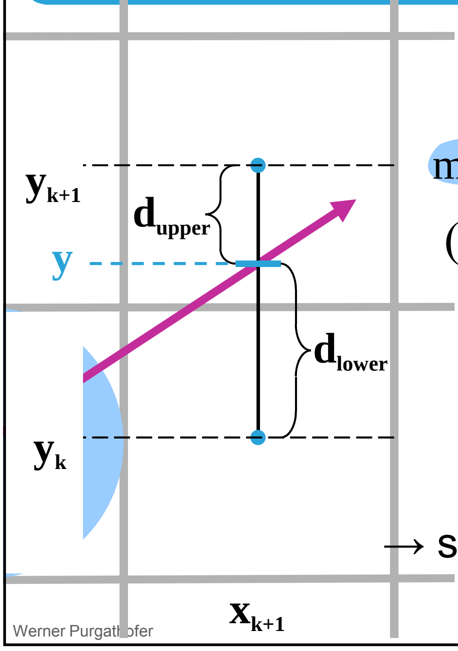
$$y = m\cdot(x_k + 1) + b$$

$$d_{lower} = y - y_k =$$
$$= m\cdot(x_k + 1) + b - y_k$$

$$d_{upper} = (y_k + 1) - y =$$
$$= y_k + 1 - m\cdot(x_k + 1) - b$$

$$d_{lower} - d_{upper} =$$
$$= 2m\cdot(x_k + 1) - 2y_k + 2b - 1$$

Bresenham’s Line Algorithm (2/4)



$$d_{lower} - d_{upper} =$$
$$= 2m\cdot(x_k + 1) - 2y_k + 2b - 1$$

$$m = \Delta y/\Delta x$$

$$(\Delta x = x_1 - x_0, \Delta y = y_1 - y_0)$$

decision parameter:

$$p_k = \Delta x\cdot(d_{lower} - d_{upper}) =$$
$$= 2\Delta y\cdot x_k - 2\Delta x\cdot y_k + c$$

→ same sign as $(d_{lower} - d_{upper})$

Bresenham’s Line Algorithm (3/4)

current decision value:

$$p_k = \Delta x \cdot (d_{lower} - d_{upper}) = 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c$$

next decision value:

$$p_{k+1} = 2\Delta y \cdot x_{k+1} - 2\Delta x \cdot y_{k+1} + c + 0$$
$$+ p_k - 2\Delta y \cdot x_k + 2\Delta x \cdot y_k - c =$$
$$= p_k + 2\Delta y - 2\Delta x \cdot (y_{k+1} - y_k)$$

Bresenham’s Line Algorithm (3/4)

current decision value:

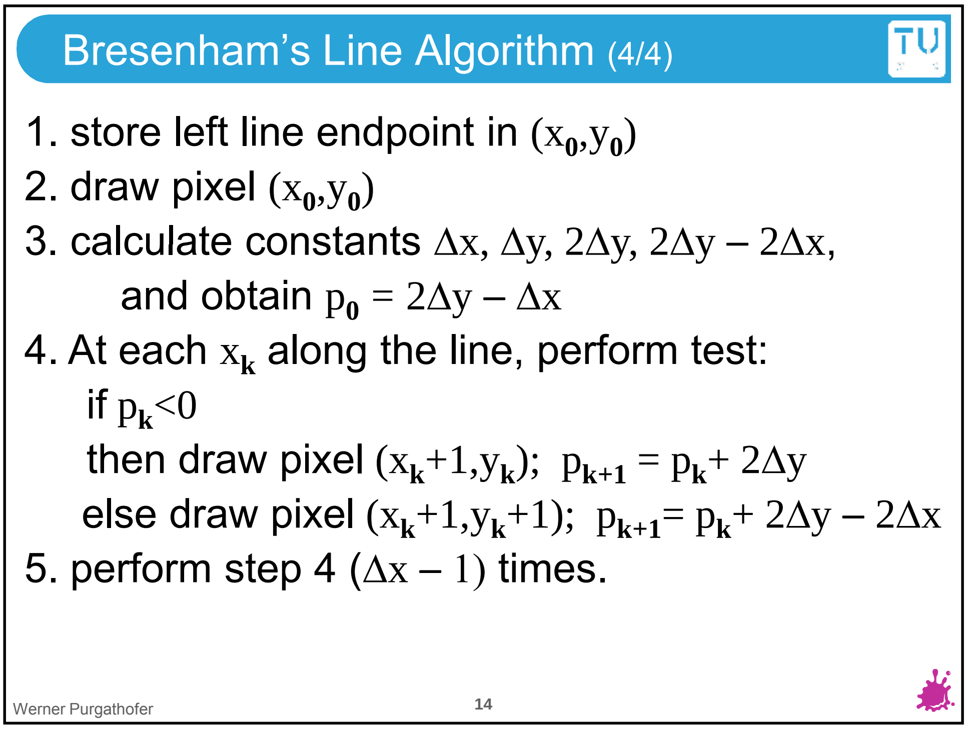
$$p_k = \Delta x \cdot (d_{lower} - d_{upper}) = 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c$$

next decision value:

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x \cdot (y_{k+1} - y_k)$$

starting decision value:

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

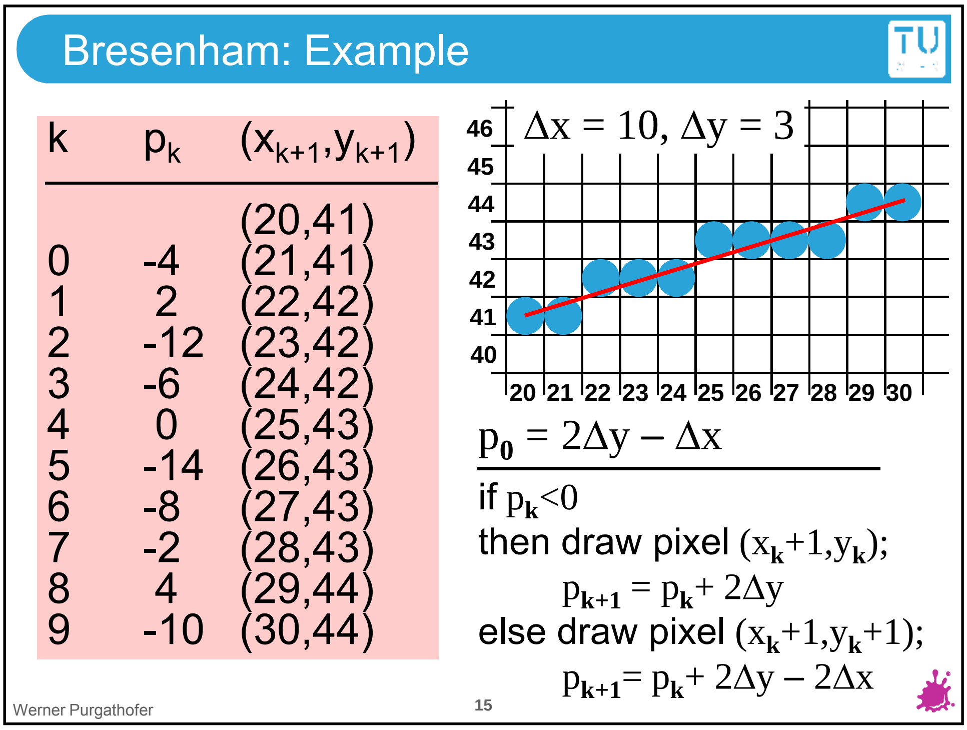


Bresenham's Line Algorithm (4/4)

1. store left line endpoint in (x_0,y_0)
2. draw pixel (x_0,y_0)
3. calculate constants Δx, Δy, $2\Delta y$, $2\Delta y - 2\Delta x$,
   and obtain $p_0 = 2\Delta y - \Delta x$
4. At each x_k along the line, perform test:

```
    if $p_k<0$
    then draw pixel $(x_k+1,y_k)$;  $p_{k+1} = p_k + 2\Delta y$
    else draw pixel $(x_k+1,y_k+1)$;  $p_{k+1} = p_k + 2\Delta y - 2\Delta x$
```

5. perform step 4 $(\Delta x - 1)$ times.



Bresenham: Example

k	p_k	(x_{k+1},y_{k+1})
		(20,41)
0	-4	(21,41)
1	2	(22,42)
2	-12	(23,42)
3	-6	(24,42)
4	0	(25,43)
5	-14	(26,43)
6	-8	(27,43)
7	-2	(28,43)
8	4	(29,44)
9	-10	(30,44)

$\Delta x = 10$, $\Delta y = 3$

46
45
44
43
42
41
40
20
21
22
23
24
25
26
27
28
29
30

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

```
if $p_k<0$
then draw pixel $(x_k+1,y_k)$;
        $p_{k+1} = p_k + 2\Delta y$
else draw pixel $(x_k+1,y_k+1)$;
        $p_{k+1} = p_k + 2\Delta y - 2\Delta x$
```

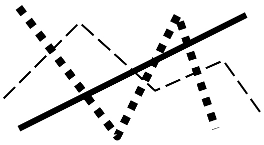
Attributes of Graphics Output Primitives

- in 2D
  - points, lines
  - characters
- in 2D and 3D
  - triangles
  - other polygons
  - ((filled) ellipses and other curves)

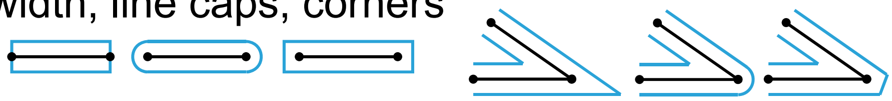
Points and Line Attributes

- color

- type: solid, dashed, dotted,…

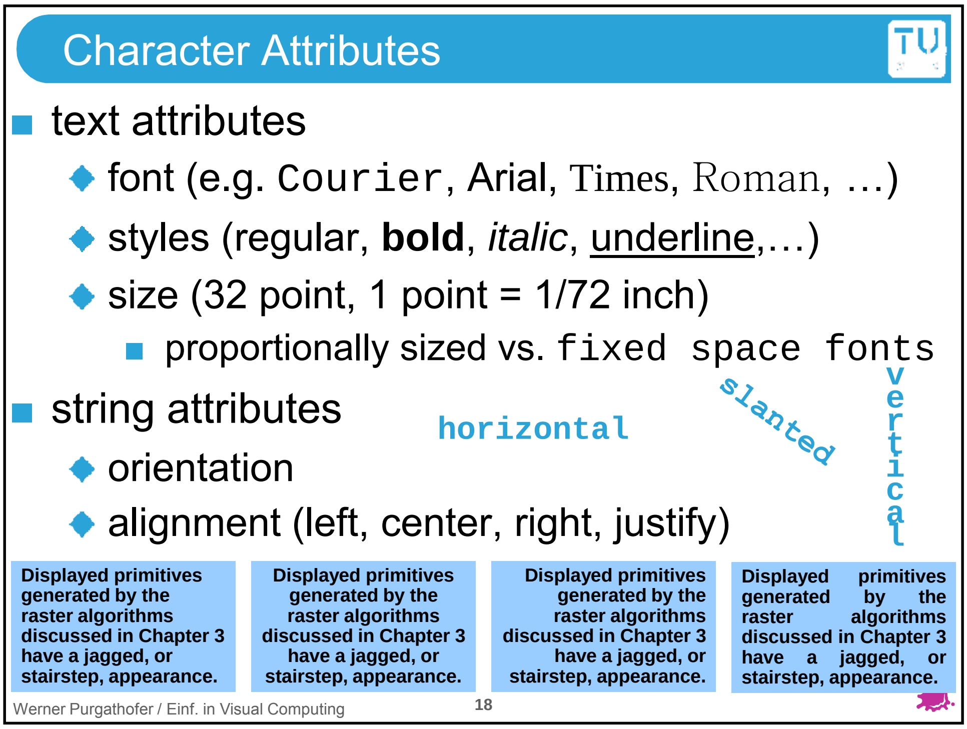


- width, line caps, corners



- pen and brush options

- …



Character Attributes

- text attributes
  - font (e.g. `Courier`, Arial, Times, Roman, …)
  - styles (regular, **bold**, *italic*, underline,…)
  - size (32 point, 1 point = 1/72 inch)
    - proportionally sized vs. `fixed space fonts`
- string attributes
  - orientation
  - alignment (left, center, right, justify)

horizontal

slanted

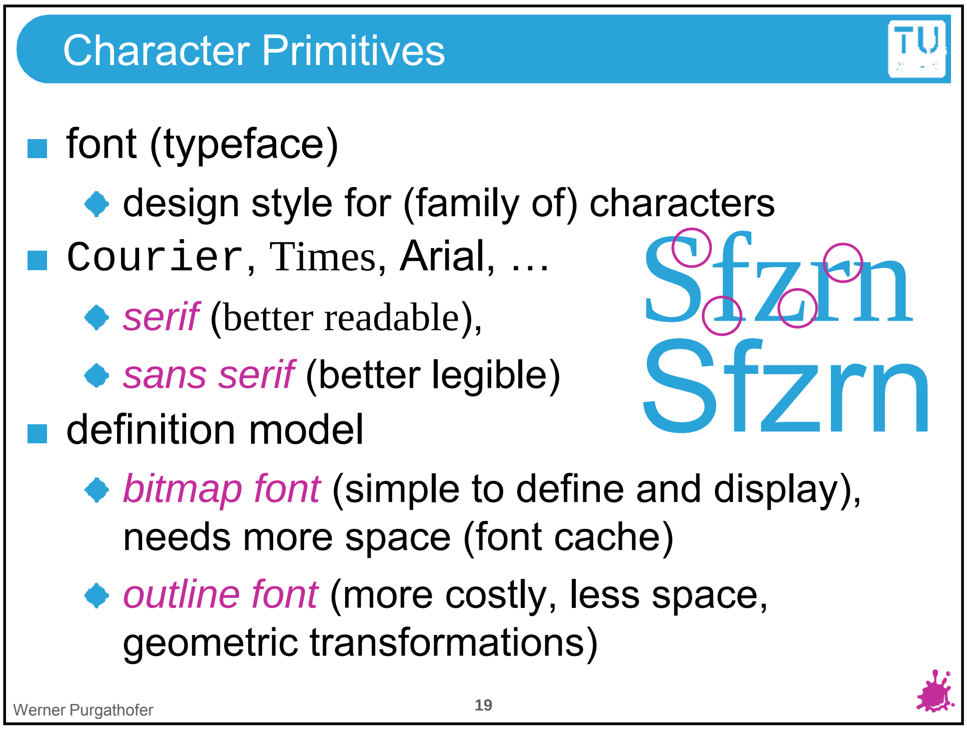
vertical

Displayed primitives generated by the raster algorithms discussed in Chapter 3 have a jagged, or stairstep, appearance.

Displayed primitives generated by the raster algorithms discussed in Chapter 3 have a jagged, or stairstep, appearance.

Displayed primitives generated by the raster algorithms discussed in Chapter 3 have a jagged, or stairstep, appearance.

Displayed primitives generated by the raster algorithms discussed in Chapter 3 have a jagged, or stairstep, appearance.

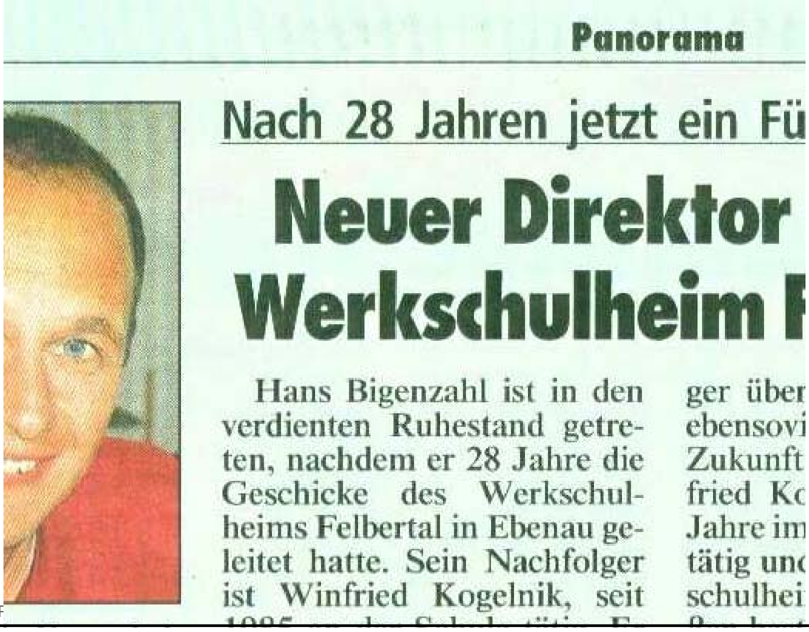


Character Primitives

- font (typeface)
  - design style for (family of) characters
- `Courier`, Times, Arial, …
  - *serif* (better readable),
  - *sans serif* (better legible)
- definition model
  - *bitmap font* (simple to define and display), needs more space (font cache)
  - *outline font* (more costly, less space, geometric transformations)

Sfzrn
Sfzrn

Example: Newspaper



Panorama

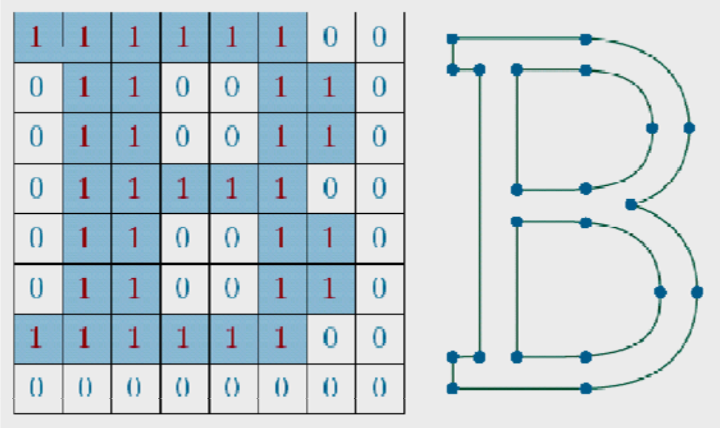
Nach 28 Jahren jetzt ein Fü

Neuer Direktor Werkschulheim F

Hans Bigenzahl ist in den verdienten Ruhestand getreten, nachdem er 28 Jahre die Geschicke des Werkschulheims Felbertal in Ebenau geleitet hatte. Sein Nachfolger ist Winfried Kogelnik, seit

ger über
ebensovi
Zukunft
fried Ko
Jahre im
tätig unc
schulhei

Character Generation Examples

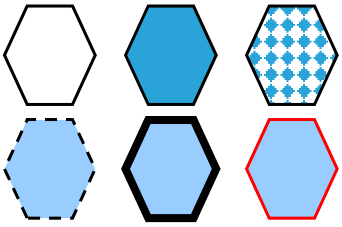


1	1	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0

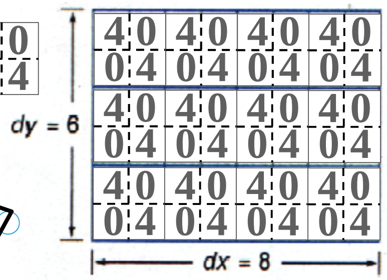
the letter **B** represented with an 8x8 bilevel bitmap pattern and with an outline shape defined with straight line and curve segments

Area-Fill Attributes (1)

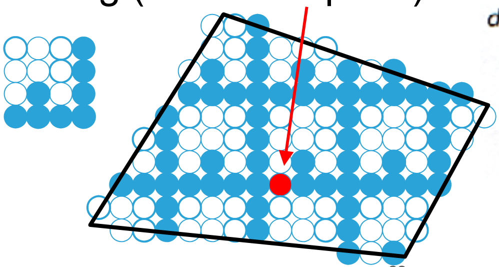
- fill styles
  - hollow, solid fill, pattern fill
- fill options
  - edge type, width, color
- pattern specification
  - through pattern tables
  - tiling (reference point)



4	0
0	4

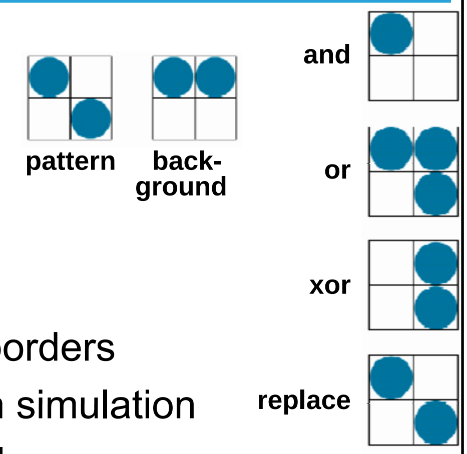


4	0	4	0	4	0	4	0
0	4	0	4	0	4	0	4
4	0	4	0	4	0	4	0
0	4	0	4	0	4	0	4
4	0	4	0	4	0	4	0
0	4	0	4	0	4	0	4



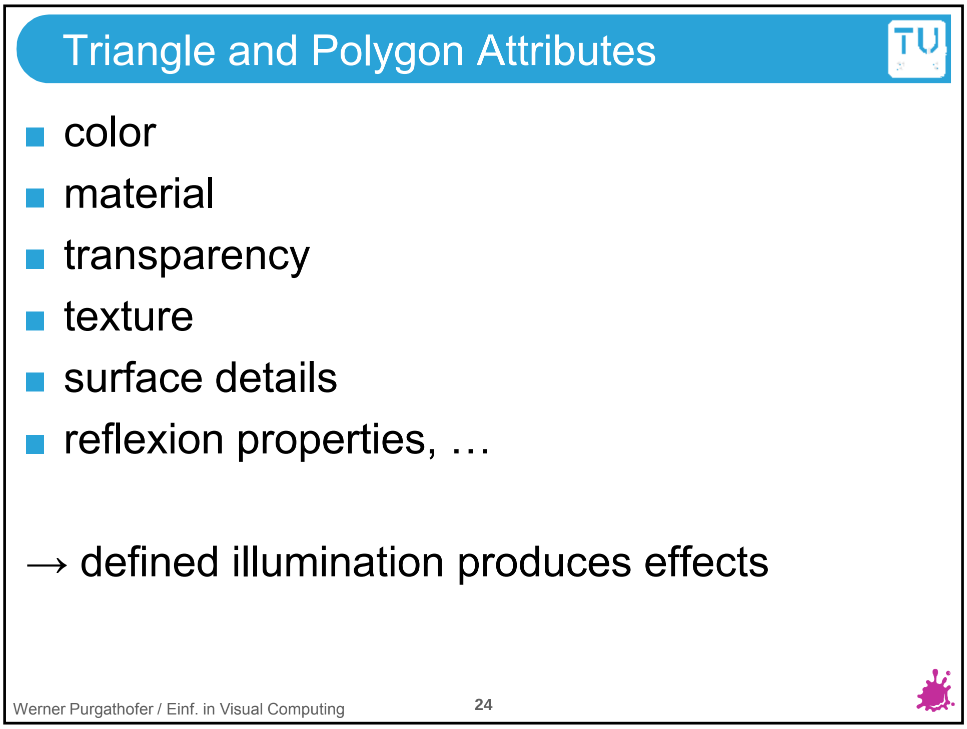
Area-Fill Attributes (2)

- combination of fill pattern with background colors
- soft fill
  - combination of colors
  - antialiasing at object borders
  - semitransparent brush simulation
  - example: linear soft-fill



F...foreground color
B...background color

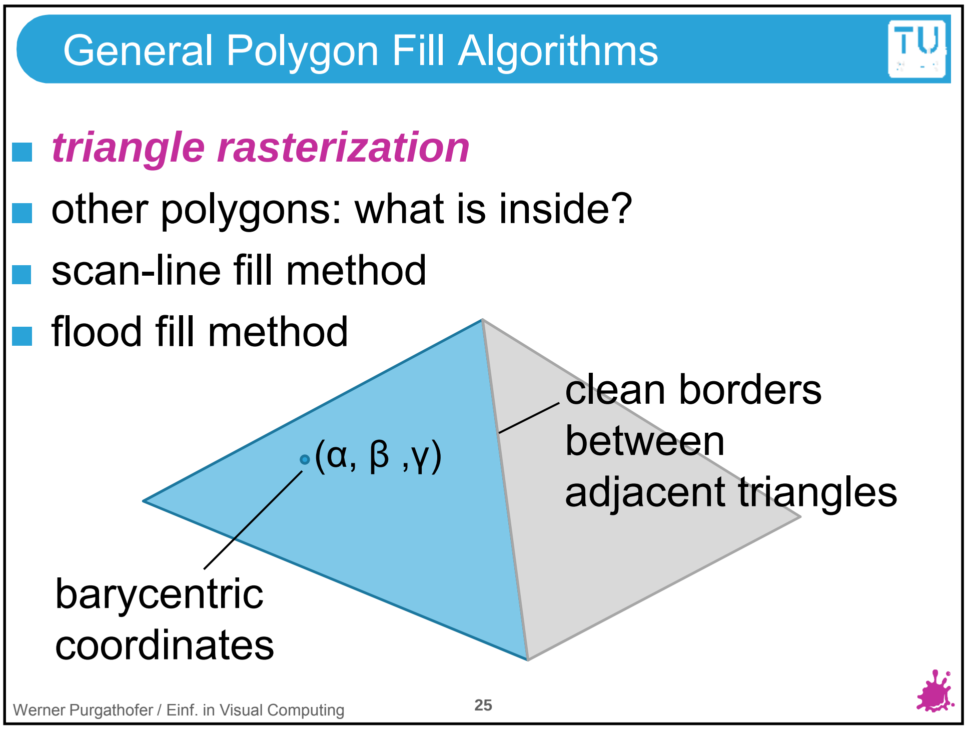
$$P = tF + (1 - t)B$$



Triangle and Polygon Attributes

- color
- material
- transparency
- texture
- surface details
- reflexion properties, …

→ defined illumination produces effects



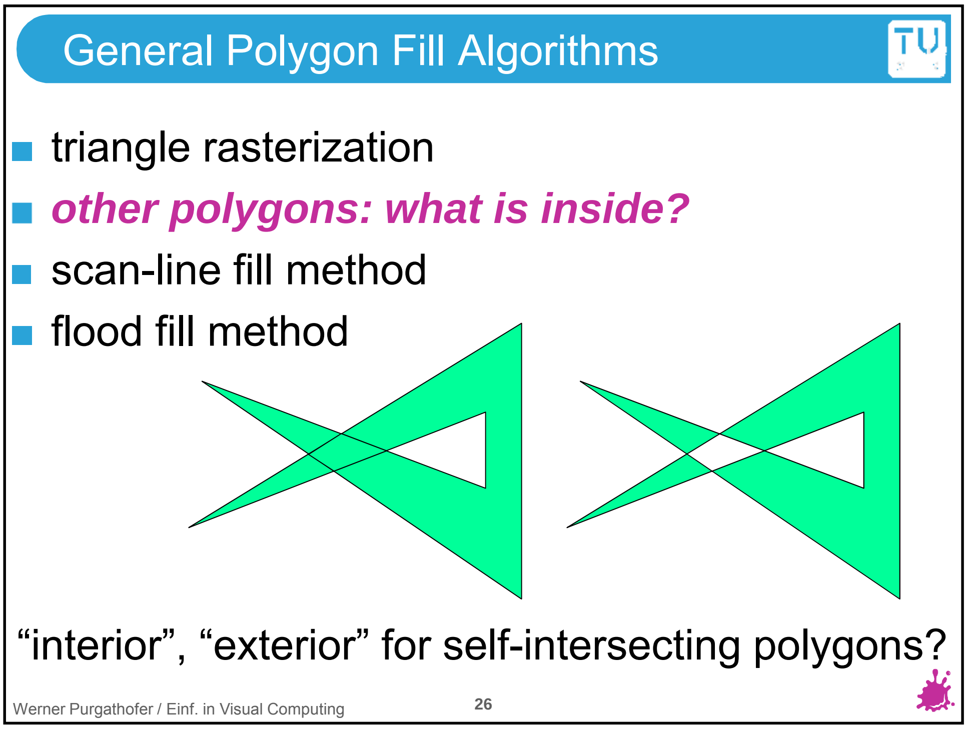
General Polygon Fill Algorithms

- ***triangle rasterization***
- other polygons: what is inside?
- scan-line fill method
- flood fill method

(α, β, γ)

clean borders between adjacent triangles

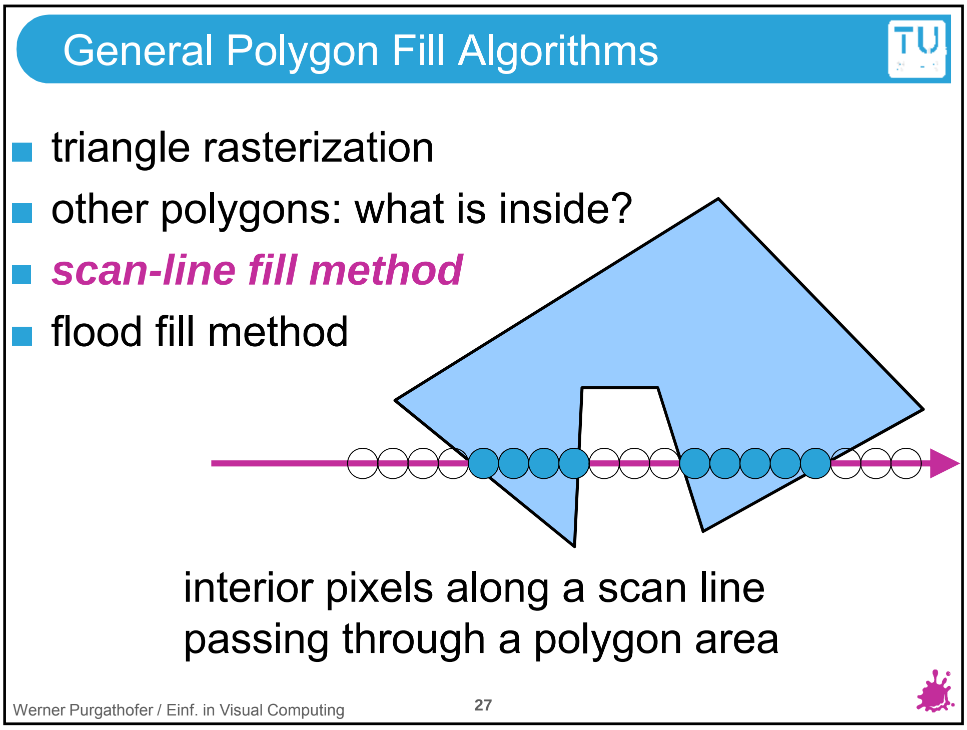
barycentric coordinates



General Polygon Fill Algorithms

- triangle rasterization
- ***other polygons: what is inside?***
- scan-line fill method
- flood fill method

“interior”, “exterior” for self-intersecting polygons?



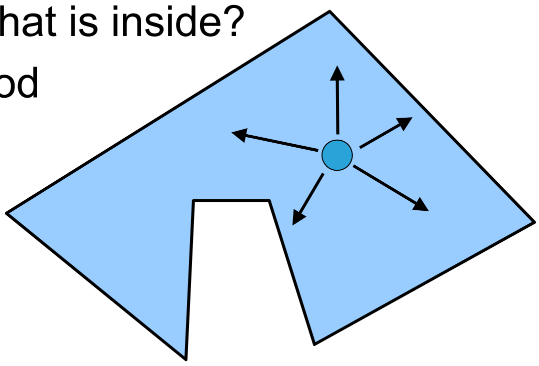
General Polygon Fill Algorithms

- triangle rasterization
- other polygons: what is inside?
- ***scan-line fill method***
- flood fill method

interior pixels along a scan line
passing through a polygon area

General Polygon Fill Algorithms

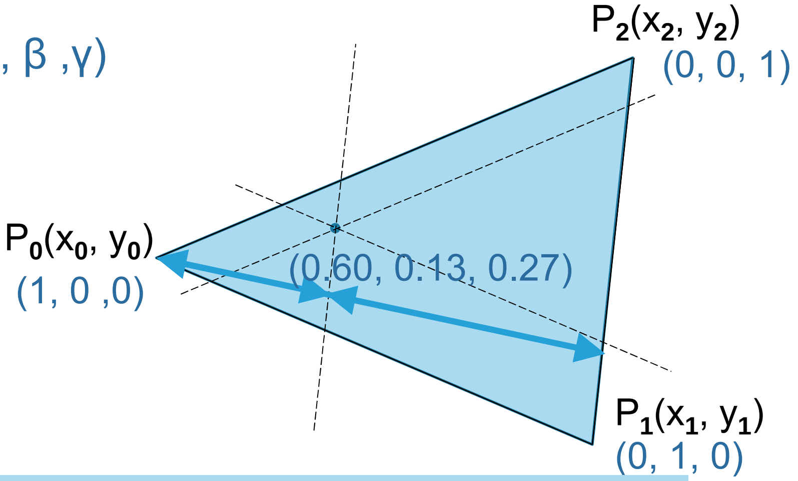
- triangle rasterization
- other polygons: what is inside?
- scan-line fill method
- ***flood fill method***



starting from a seed point fill until you reach a border

Triangles: Barycentric Coordinates

- (α, β ,γ)

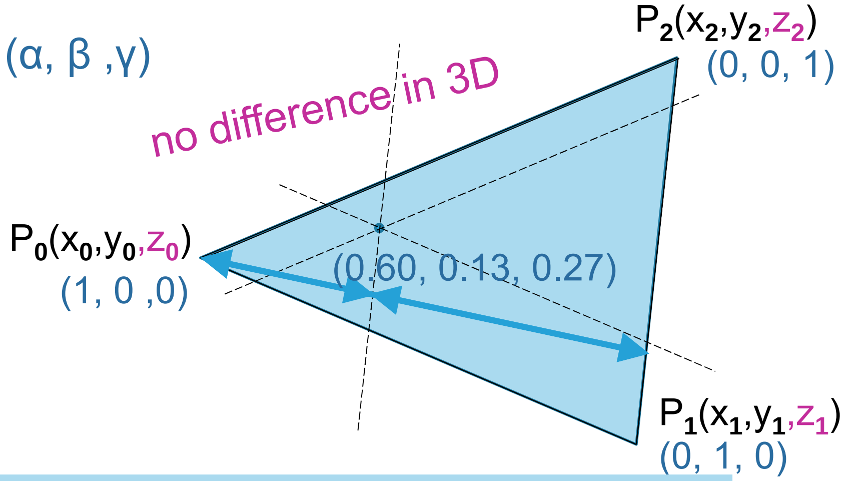


$P = \alpha P_0 + \beta P_1 + \gamma P_2$

triangle = $\{P | \alpha+\beta+\gamma=1, 0<\alpha<1, 0<\beta<1, 0<\gamma<1\}$

Barycentric Coordinates

- (α, β ,γ)



$P = \alpha P_0 + \beta P_1 + \gamma P_2$

triangle = $\{P \mid \alpha+\beta+\gamma=1,\ 0<\alpha<1,\ 0<\beta<1,\ 0<\gamma<1\}$

Triangle Rasterization Algorithm

```
for all x
  for all y          /* use a bounding box!*/
    {compute (α, β ,γ) for (x,y) ;
    if (0<α<1) and (0<β<1) and (0<γ<1)
    {c = αc₀ + βc₁ + γc₂ ;
      draw pixel (x,y)  with color c
    }}
```

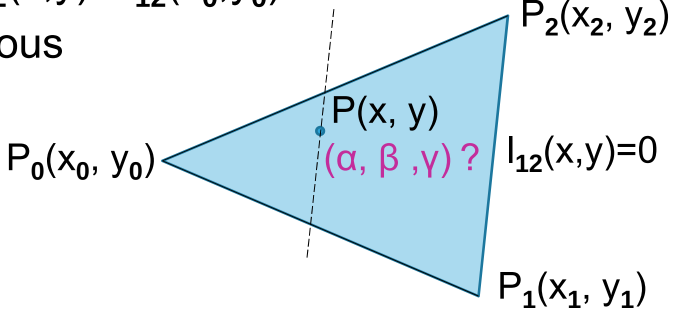
interpolates values at the corners (vertices) linearly inside the triangle (and along edges)

Computing (α, β ,γ) for P(x,y)

line through P_1, P_2: $l_{12}(x,y) = a_{12}x+b_{12}y+c_{12}=0$

then $\alpha = l_{12}(x,y) / l_{12}(x_0,y_0)$

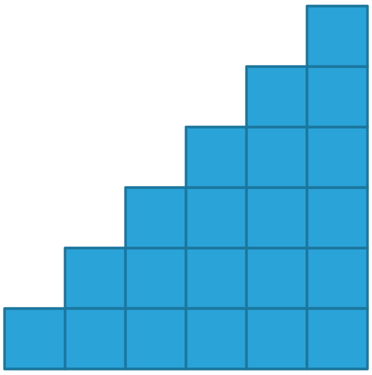
β,γ analogous



$P = \alpha P_0 + \beta P_1 + \gamma P_2$

triangle = {P| α+β+γ=1, 0<α<1, 0<β<1, 0<γ<1}

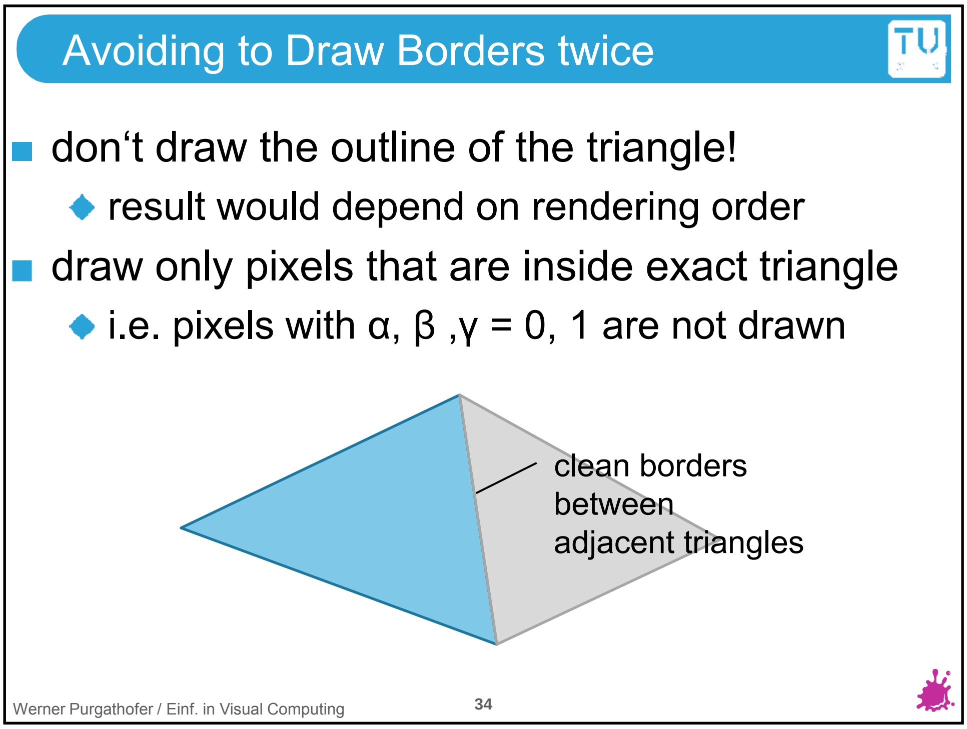
Barycentric Coordinates Example



1.2 -1.4 1.2	1.0 -1.2 1.2	0.8 -1.0 1.2	0.6 -0.8 1.2	0.4 -0.6 1.2	0.2 -0.4 1.2	0.0 -0.2 1.2	-0.2 0.0 1.2
1.2 -1.2 1.0	1.0 -1.0 1.0	0.8 -0.8 1.0	0.6 -0.6 1.0	0.4 -0.4 1.0	0.2 -0.2 1.0	0.0 0.0 1.0	-0.2 0.2 1.0
1.2 -1.0 0.8	1.0 -0.8 0.8	0.8 -0.6 0.8	0.6 -0.4 0.8	0.4 -0.2 0.8	0.2 0.0 0.8	0.0 0.2 0.8	-0.2 0.4 0.8
1.2 -0.8 0.6	1.0 -0.6 0.6	0.8 -0.4 0.6	0.6 -0.2 0.6	0.4 0.0 0.6	0.2 0.2 0.6	0.0 0.4 0.6	-0.2 0.6 0.6
1.2 -0.6 0.4	1.0 -0.4 0.4	0.8 -0.2 0.4	0.6 0.0 0.4	0.4 0.2 0.4	0.2 0.4 0.4	0.0 0.6 0.4	-0.2 0.8 0.4
1.2 -0.4 0.2	1.0 -0.2 0.2	0.8 0.0 0.2	0.6 0.2 0.2	0.4 0.4 0.2	0.2 0.6 0.2	0.0 0.8 0.2	-0.2 1.0 0.2
1.2 -0.2 0.0	1.0 0.0 0.0	0.8 0.2 0.0	0.6 0.4 0.0	0.4 0.6 0.0	0.2 0.8 0.0	0.0 1.0 0.0	-0.2 1.2 0.0
1.2 0.0 -0.2	1.0 0.2 -0.2	0.8 0.4 -0.2	0.6 0.6 -0.2	0.4 0.8 -0.2	0.2 1.0 -0.2	0.0 1.2 -0.2	-0.2 1.4 -0.2

$P = \alpha P_0 + \beta P_1 + \gamma P_2$

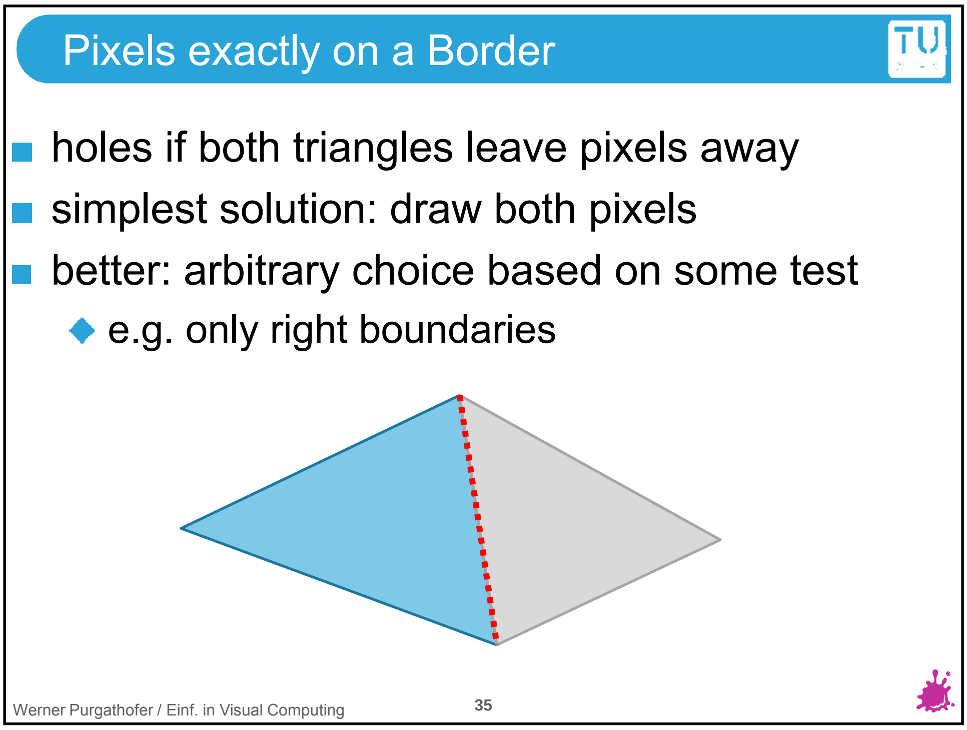
triangle = {P| α+β+γ=1, 0<α<1, 0<β<1, 0<γ<1}



Avoiding to Draw Borders twice

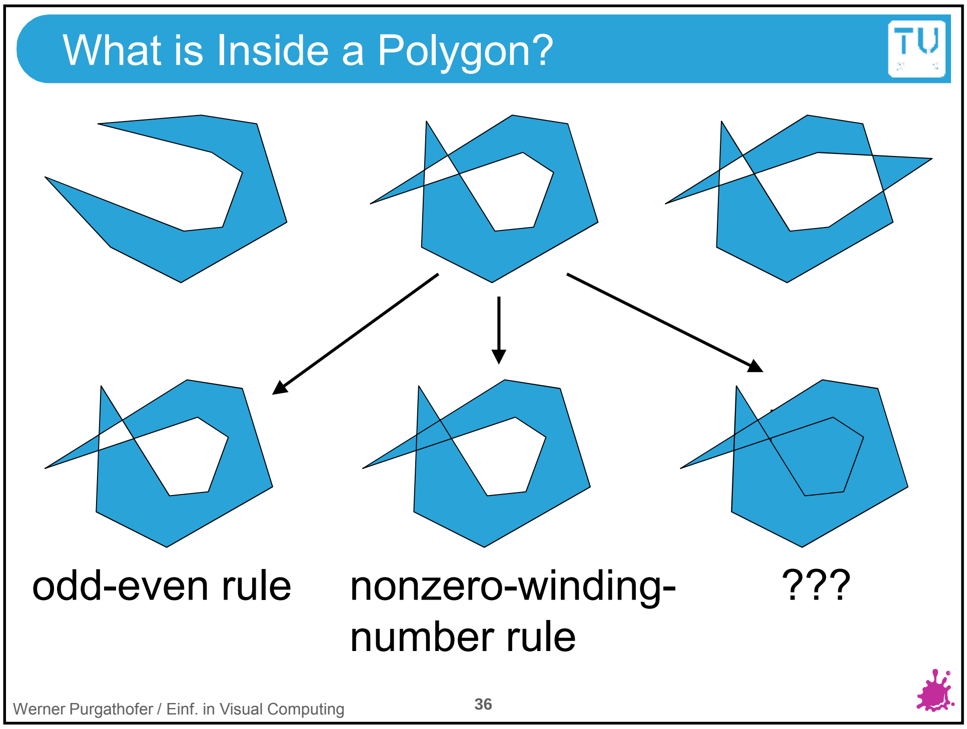
- don‘t draw the outline of the triangle!
  - result would depend on rendering order
- draw only pixels that are inside exact triangle
  - i.e. pixels with α, β ,γ = 0, 1 are not drawn

clean borders between adjacent triangles



Pixels exactly on a Border

- holes if both triangles leave pixels away
- simplest solution: draw both pixels
- better: arbitrary choice based on some test
  - e.g. only right boundaries

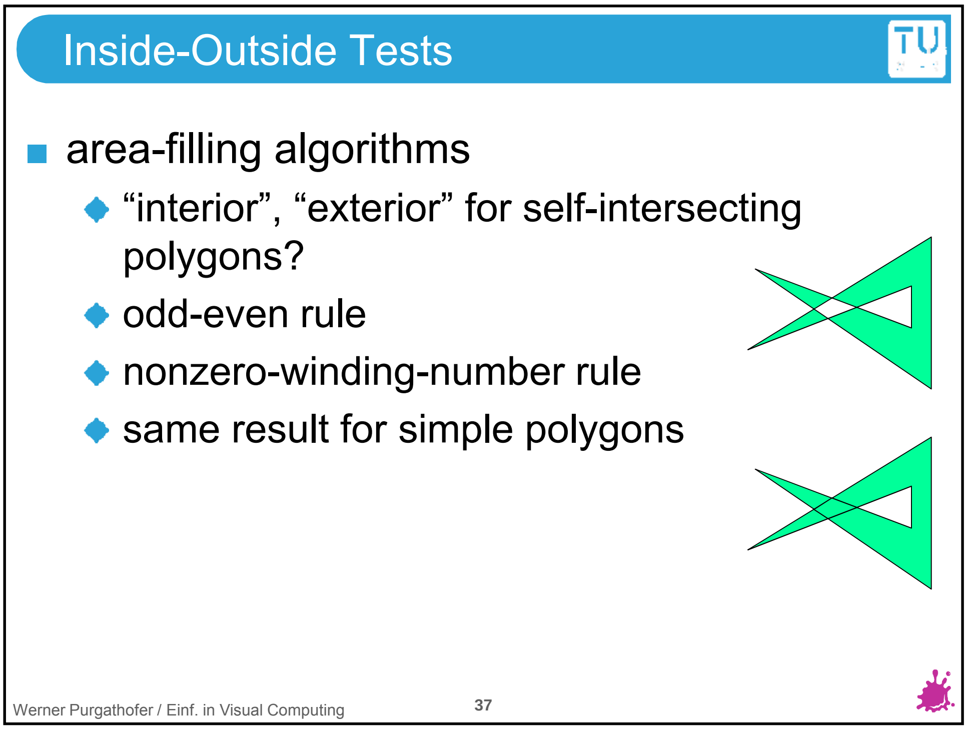


What is Inside a Polygon?

odd-even rule

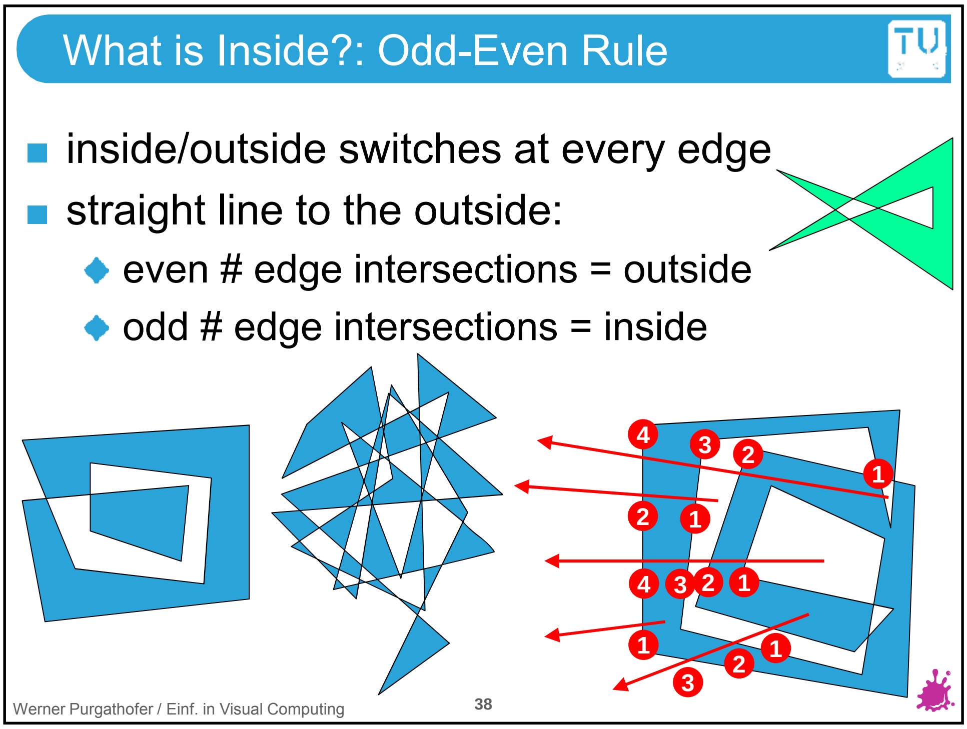
nonzero-winding-number rule

???



Inside-Outside Tests

- area-filling algorithms
  - “interior”, “exterior” for self-intersecting polygons?
  - odd-even rule
  - nonzero-winding-number rule
  - same result for simple polygons

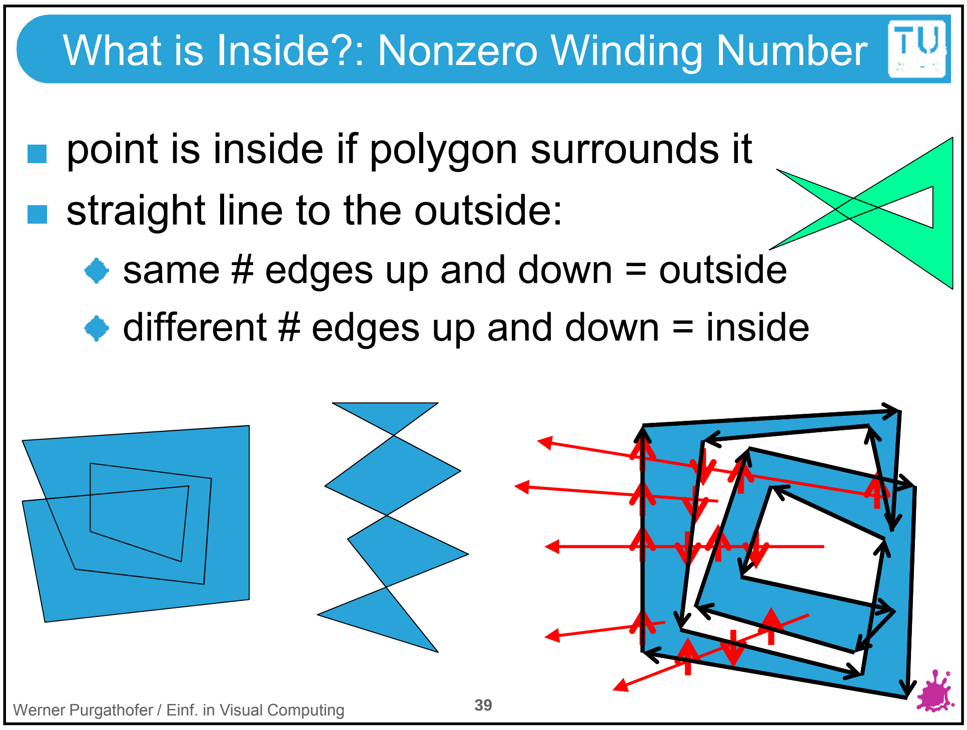


What is Inside?: Odd-Even Rule

- inside/outside switches at every edge
- straight line to the outside:
  - even # edge intersections = outside
  - odd # edge intersections = inside

Werner Purgathofer / Einf. in Visual Computing

38

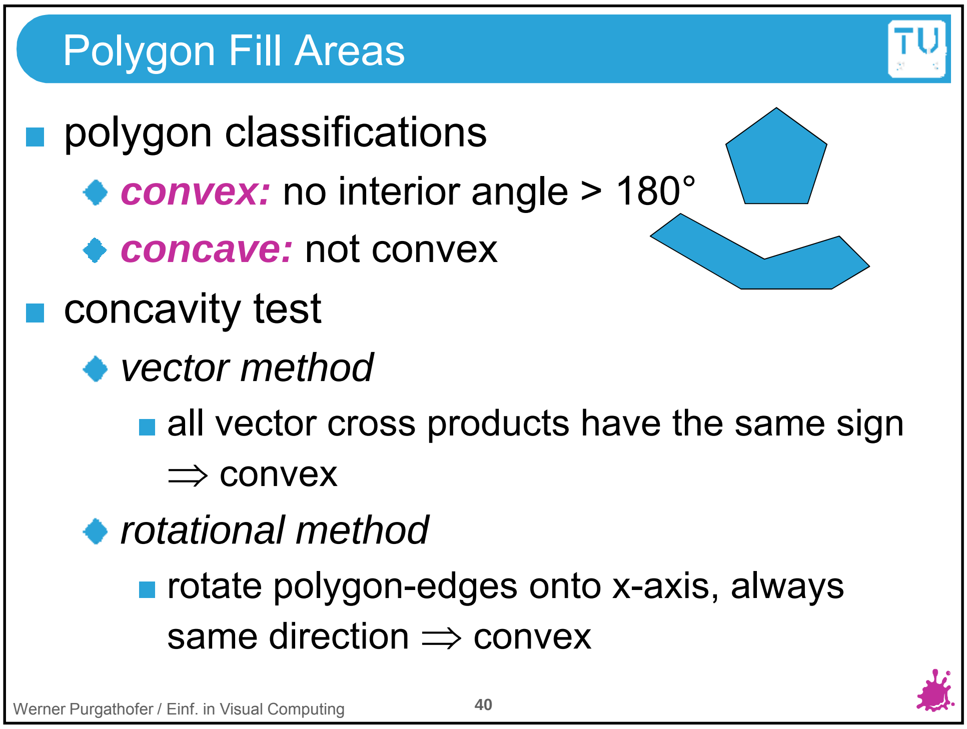


What is Inside?: Nonzero Winding Number

- point is inside if polygon surrounds it
- straight line to the outside:
  - same # edges up and down = outside
  - different # edges up and down = inside

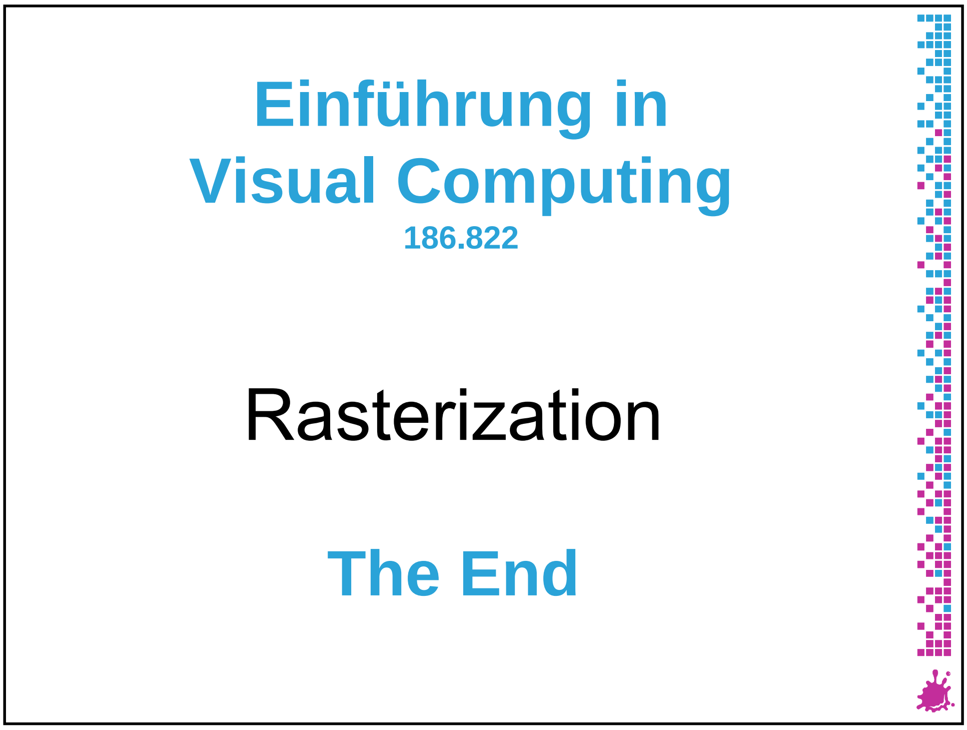
Werner Purgathofer / Einf. in Visual Computing

39



Polygon Fill Areas

- polygon classifications
  - ***convex:*** no interior angle > 180°
  - ***concave:*** not convex
- concavity test
  - *vector method*
    - all vector cross products have the same sign ⇒ convex
  - *rotational method*
    - rotate polygon-edges onto x-axis, always same direction ⇒ convex



Einführung in Visual Computing

186.822

Rasterization

The End