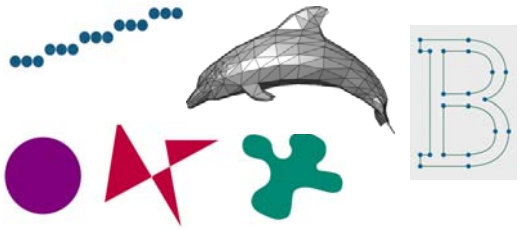
Graphics Output Primitives



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

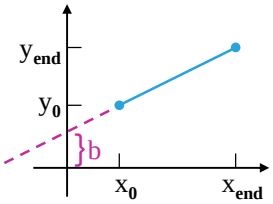
Line-Drawing Algorithms

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

line path between two points:

$$m = \frac{y_{end} - y_0}{x_{end} - x_0}$$

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



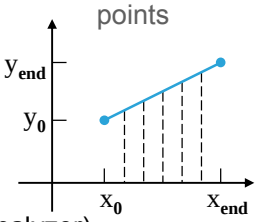
DDA Line-Drawing Algorithm

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

5 sampling points

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

$$\delta x = \frac{\delta y}{m} \quad \text{for} \quad |m|>1$$



- DDA (digital differential analyzer)

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

DDA – Algorithm Principle

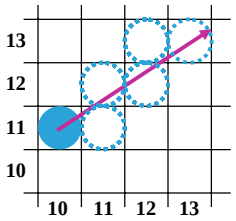
```
dx = xEnd - x0; dy = yEnd - y0;
m = dy / dx;

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

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

extension to other cases simple

Bresenham's Line Algorithm

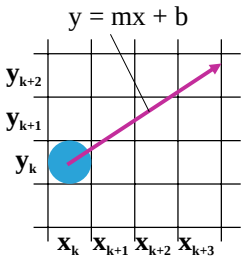
- faster than simple DDA
  - incremental integer calculations
  - adaptable to circles, other curves

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



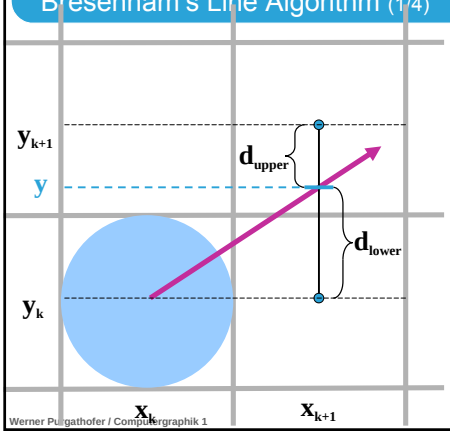
Section of a display screen where a straight line segment is to be plotted, starting from the pixel at column 10 on scan line 11

Bresenham's Line Algorithm

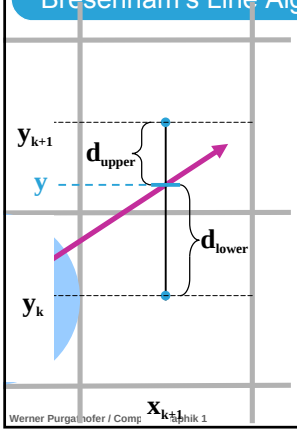


Section of the screen grid showing a pixel in column x_k on scan line y_k that is to be plotted along the path of a line segment with slope 0<m<1

Bresenham's Line Algorithm (1/4)



Bresenham's Line Algorithm (1/4)



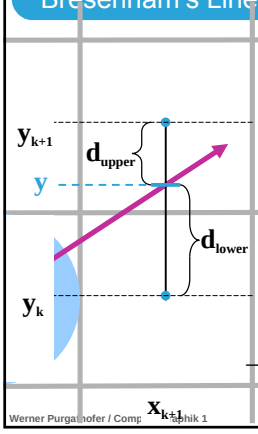
$y = m \cdot (x_k + 1) + b$

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

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

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

Bresenham's Line Algorithm (2/4)



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

$m = \Delta y / \Delta x$

$(\Delta x = x_{end} - x_0, \Delta y = y_{end} - y_0)$

decision parameter:

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

$\rightarrow$ 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:

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

starting decision value:

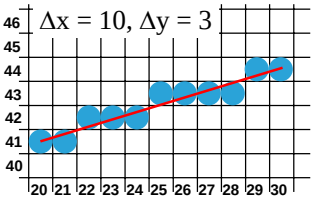
$$p_0 = 2\Delta y - \Delta x$$

Bresenham's Line Algorithm (4/4)

1. store left line endpoint in (x_0,y_0)
2. plot 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 plot pixel (x_k+1,y_k); $p_{k+1} = p_k + 2\Delta y$
   else plot 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)



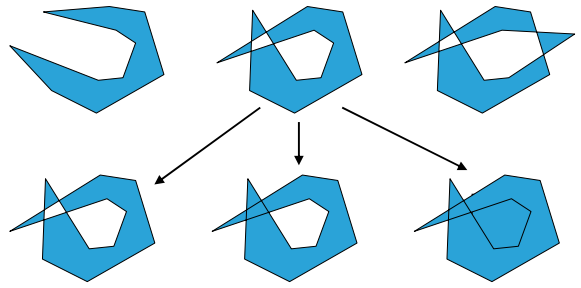
$p_0 = 2\Delta y - \Delta x$

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

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, …

What is Inside a Polygon?



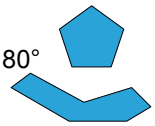
Odd Even rule | Nonzero Winding Number rule | ???

Fill-Area Primitives

- for polygon area (solid-color, patterned)
  - ***scan-line*** polygon fill algorithm
    - intersection points located and sorted
    - consecutive pairs define interior span
    - attention with vertex intersections
    - exploit coherence (incremental calculations)
  - ***flood fill*** algorithm

Polygon Fill Areas

- polygon classifications
  - ***convex:*** no interior angle > 180°
  - ***concave:*** not convex



- splitting concave polygons
  - *vector method*
    - all vector cross products have the same sign ⇒ convex
  - *rotational method*
    - rotate polygon-edges onto x-axis, always same direction ⇒ convex

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, …

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

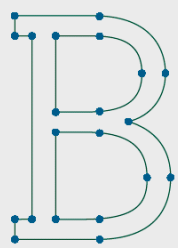
Neuer Direktor Werkschulheim F

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 und 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



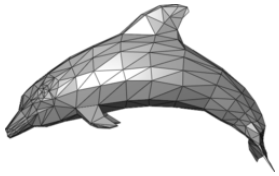
the letter **B** represented with an 8x8 bilevel bitmap pattern and with an outline shape defined with straight line and curve segments

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, …

Polygon Surfaces (1)

- set of surface polygons enclose object interior
  = ***Boundary Representation***
  ("B-rep")



Example of a triangle mesh representing a dolphin.

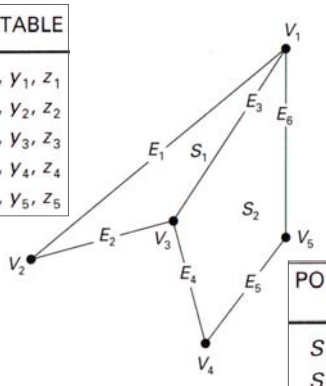
Polygon Surfaces (2)

- polygon tables (B-rep lists)
  - geometric and attribute tables
  - vertex, edge, polygon tables
  - consistency, completeness checks

Polygon Surfaces: Data Structure

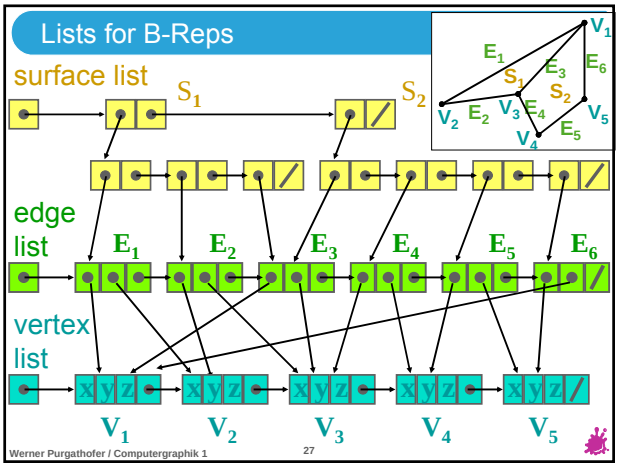
VERTEX TABLE	
V_1:	x_1, y_1, z_1
V_2:	x_2, y_2, z_2
V_3:	x_3, y_3, z_3
V_4:	x_4, y_4, z_4
V_5:	x_5, y_5, z_5

EDGE TABLE	
E_1:	V_1, V_2
E_2:	V_2, V_3
E_3:	V_3, V_1
E_4:	V_3, V_4
E_5:	V_4, V_5
E_6:	V_5, V_1



POLYGON-SURFACE TABLE	
S_1:	E_1, E_2, E_3
S_2:	E_3, E_4, E_5, E_6

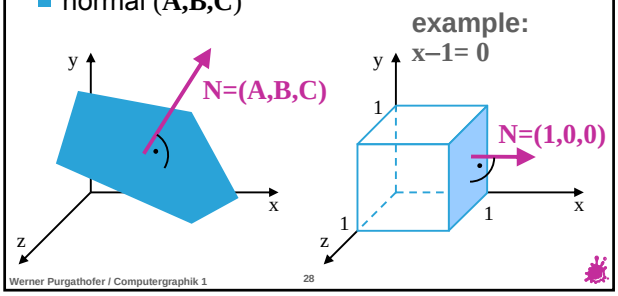
Lists for B-Reps



Polygon Surfaces: Plane Equation

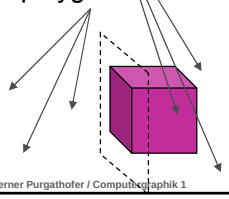
$$Ax + By + Cz + D = 0$$

- plane parameters **A,B,C,D**
- normal (**A,B,C**)



Front and Back Polygon Faces

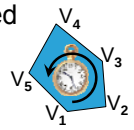
- ***back face*** = polygon side that faces into the object interior
- ***front face*** = polygon side that faces outward
- ***behind*** a polygon plane = visible to the polygon back face
- ***in front of*** a polygon plane = visible to the polygon front face



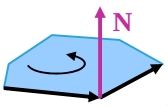
Front and Back Polygon Faces

$Ax + By + Cz + D = 0$ for points on the surface
< 0 for points behind
> 0 for points in front

if (1) right-handed coordinate system
(2) polygon points ordered counterclockwise



V_1, V_2, V_3 counterclockwise $\Rightarrow$ normal vector
$\mathbf{N} = (\mathbf{V_2} - \mathbf{V_1}) \times (\mathbf{V_3} - \mathbf{V_1})$



Reminder: Product of Vectors

$$V_1 = \begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix} \qquad V_2 = \begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix}$$

- ***scalar product:***

$$V_1 \cdot V_2 = \begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix} \cdot \begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix} = a_1 a_2 + b_1 b_2 + c_1 c_2$$

- ***cross product (vector product):***

$$V_1 \times V_2 = \begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix} \times \begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix} = \begin{pmatrix} b_1 c_2 - c_1 b_2 \\ c_1 a_2 - a_1 c_2 \\ a_1 b_2 - b_1 a_2 \end{pmatrix}$$