Computer Graphics: 6-Rasterization

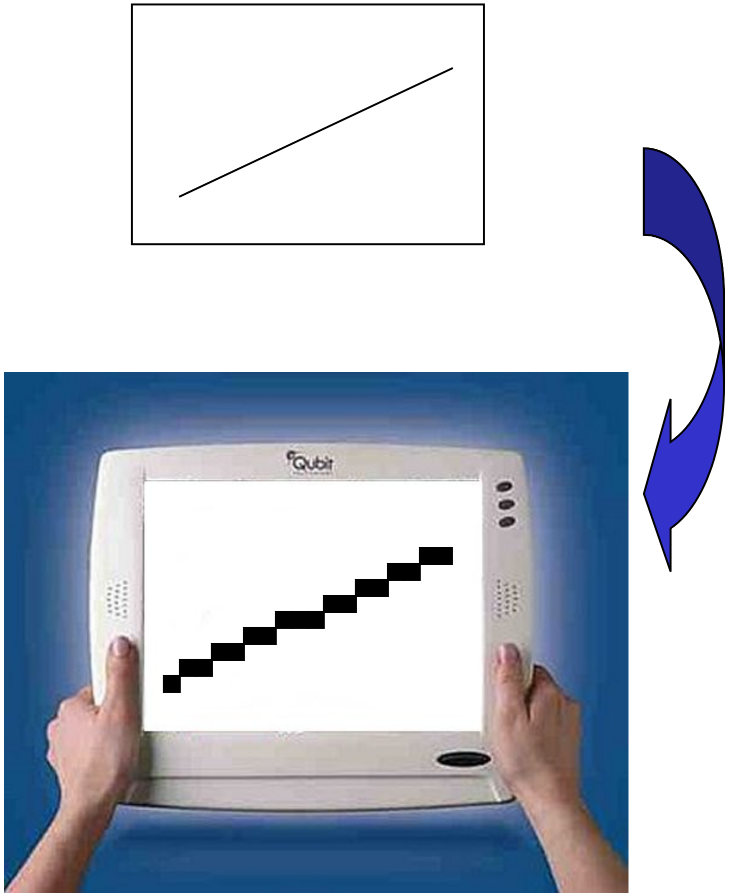
Prof. Dr. Charles A. Wüthrich,
Fakultät Medien, Medieninformatik
Bauhaus-Universität Weimar
caw AT medien.uni-weimar.de

Raster devices

- In modern devices the smallest addressable element is a point.
- Each of these dots is called a pixel, or „pel“, for picture element.
- Pixels can be represented as being the points of the plane having integer coordinates (~Z2)
- Note that this is just a mathematical representation: in reality, pixels are often broader than high, and have a shape which resemble
  - a 2D gaussian distribution (for CRT screens)
  - Small squares (for LCD displays)
  - A more or less circular dot (printers)

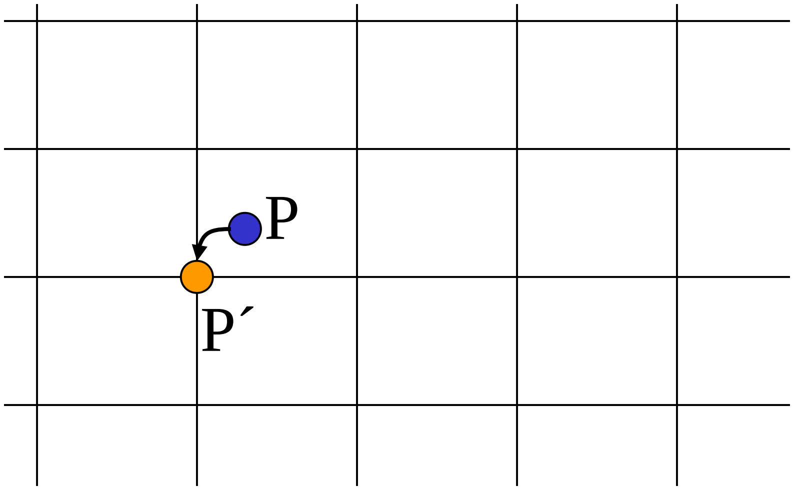
Rasterization

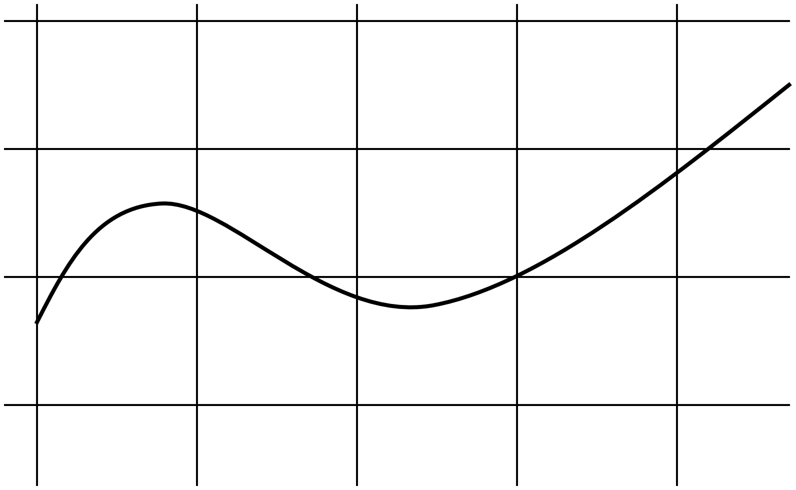
- Is the operation that allows to pass from our continuous representation of the world to the discrete world of computers
- It allows drawing lines, curves, polygons and patches on a 2D discrete output device
- How is this done?



Nearest neighbour rasterization

- One has to distinguish among two types of rasterization:
  - Point rasterization:
    Given a point P of R2, the nearest point P´ of Z2 is its rasterization.
  - Curve rasterization: nearest integer does not work any more, since curves are continuous
  - Rasterization must be based on intersection with some grid.
  - Is there a model that fits both methods?

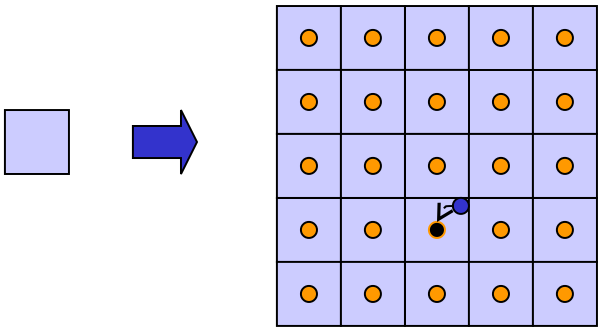




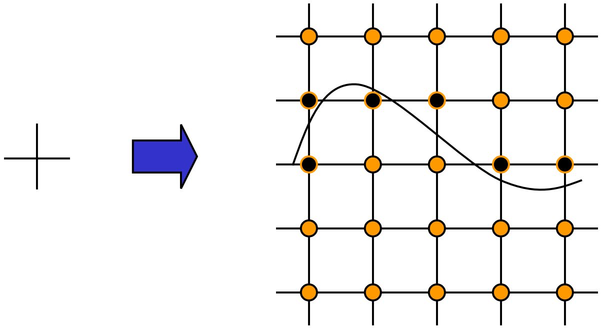
Nearest neighbour rasterization

- Sure there is!
- Let D be a compact set of R^2, ST it is included in the unit square (basic domain)
- Let D_z be the translated domain of D by the point of integer coordinates z=(i,j)
- Let A be a subset of R^2
- Def: The rasterization of A is the set of all points z such that $D_z \cap A \neq \emptyset$
- Basically one copies D around all points of integer coordinates and then takes as rasterization the corresponding points

- Different choices of D lead to different schemes



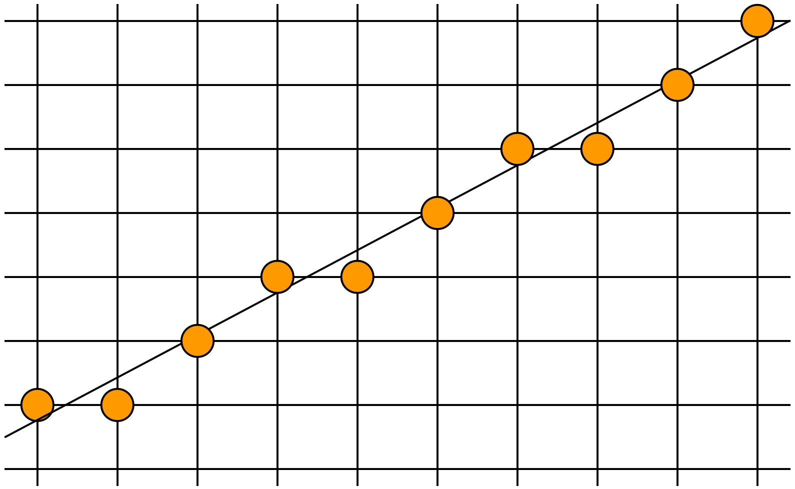
Cell rasterization



Grid intersection rasterization

Line rasterization

- Problem: Given the line passing through the points $P_I=(x_I,y_I)$, $P_F=(x_F,y_F)$, draw its rasterization
- Two basic methods for doing this:
  - Direct algorithms:
    - use global knowledge
    - generally slow
  - Incremental algorithms:
    - require only local knowledge
    - often highly optimized



Line rasterization

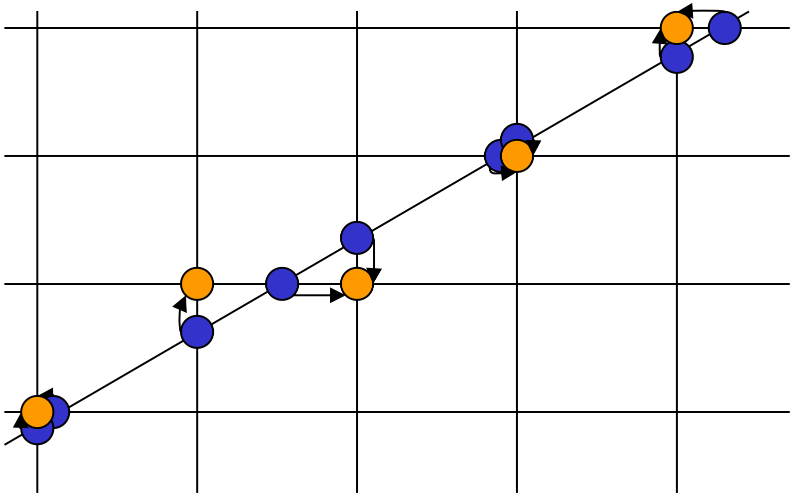
- Line through PI=(xI,yI), PF=(xF,yF):

$$y = \frac{y_F - y_I}{x_F - x_I} x + y_I - \frac{y_F - y_I}{x_F - x_I} x_I$$

- Simplest algorithm:

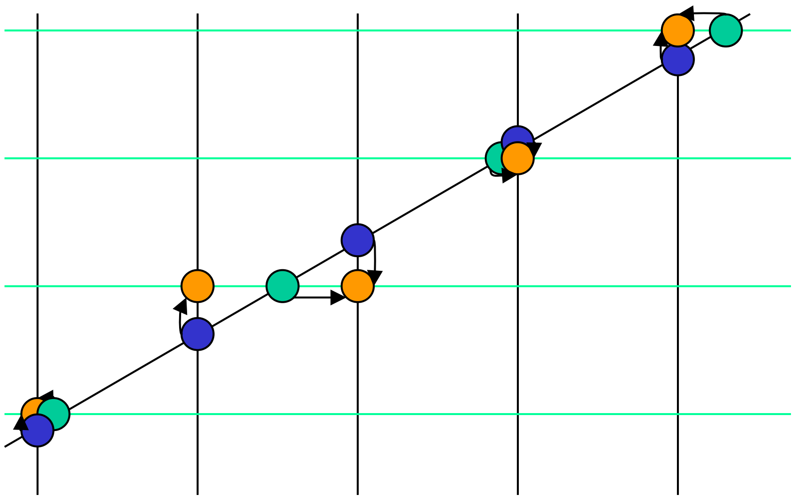
compute intersections with grid lines x=i, y=j (i,jϵZ)

  – Near intersection to next grid point



Rosenfeld‘s theorem

- There is a theorem that halves the number of intersection computations that have to be made
- Theorem: Let r be the straight line $y=mx+q$. Let $-1 \leq m \leq 1$ (slope between -45° and 45°). All the points of the rasterization of r can be found by computing the intersection with the straight lines of the form $x=i$
- Intersections with $y=j$ do not lead to additional points
- Note that a similar theorem can be stated for curves and their derivatives
- Note that intersections can lead to ambiguous rasterization points, in case they fall halfway between two integer points



DDA algorithm

- Without loss of generality, consider q=0
- Let us look at the table of the intersections with the straight lines x=i

x	y
0	q
1	m+q
2	2*m+q
3	3*m+q

DDA algorithm

- Without loss of generality, consider q=0
- Let us look at the table of the intersections with the straight lines x=i

x	y
0	q
1	m+q
2	2*m+q
3	3*m+q

At each step, $y_i+1=y_i+m$!!!!

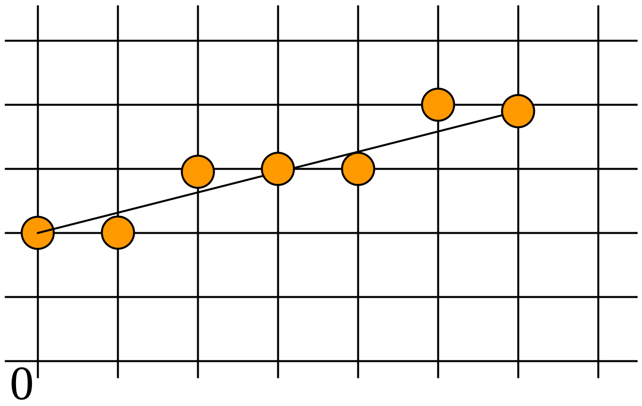
- This gives the idea for a new algorithm: we add at each step of the algorithm 1 to the x and m to the y
- The resulting algorithm is an incremental algorithm, because there is no need for the general equation

DDA algorithm

```
dy=yF-yI; dx=xF-xI;
m=dy/dx;
y=yI;
FOR x=xI TO xF
   WritePixel(x,⌈y+0.5⌉);
   y=y+m;
ENDFOR
```

x	y
0	2
1	2,33→2
2	2,66→3
3	3
4	3,33→3
5	3,66→4
6	4

- Example: $y=1/3x+2$ between (0,2) and (6,4)



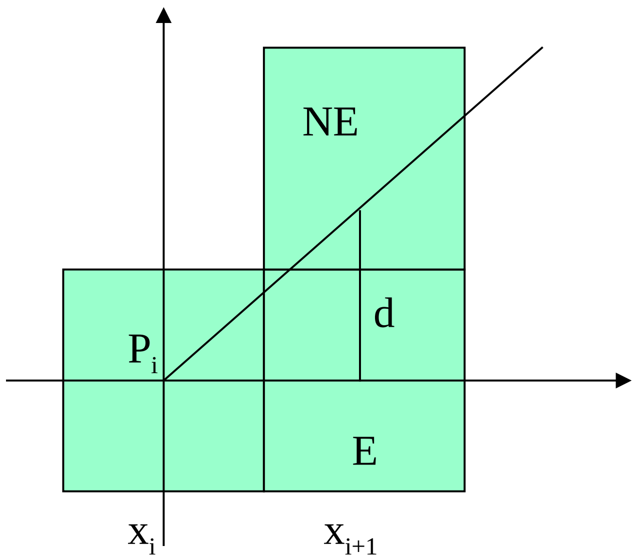
Integer DDA algorithm

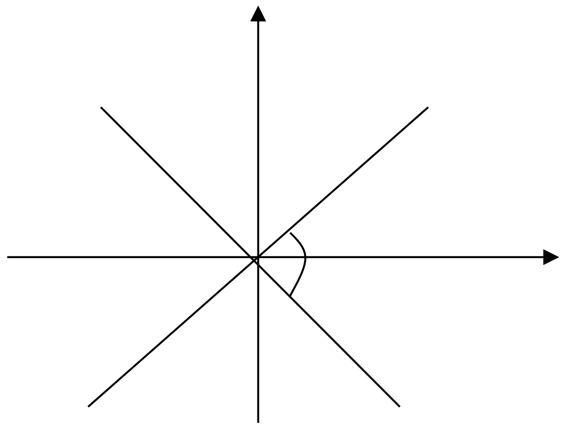
- One can improve the algorithm so as to make it use integer quantities only
- This by using the fact that only rational numbers are involved in a normal environment
- One can therefore multiply the equations by the maximum denominator to get rid of the denominators

```
dx=xF; dy=yF; x=0; y=0;
rest=0;
DrawPixel(x,y);
FOR (i=0; i<=xF; i++)
   x=x+1;
   rest=rest+dy;
   if(rest>dx)
   THEN y=y+1;
      rest=rest-dx;
   ENDIF
   DrawPixel(x,y)
ENDFOR
```

Bresenham‘s algorithm

- While tracing the line, at each step we use a control variable to check if we have to move to the right or to the upper right
- One can use thus a control variable to steer whether to step upwards or sideways
- Precompute increments and the game is done
- Also, mirroring has to be done to let the algorithm draw all cases, eventually through swapping main variable





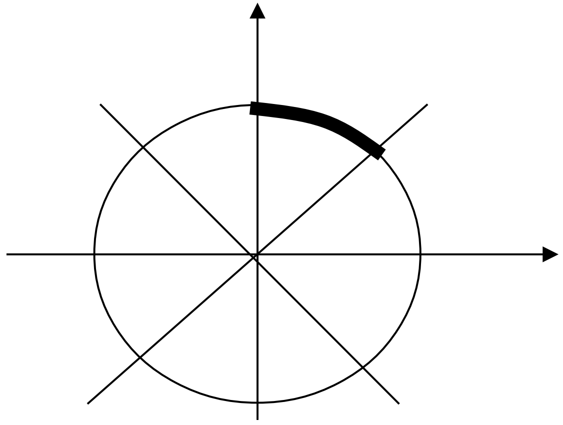
Bresenham's algorithm

```
PROCEDURE Bresenham(x1,y1,x2,y2,
   value: integer):
   var dx, dy, incr1, incr2, d, x,
   y, xend: INTEGER;
BEGIN
   dx:=ABS(x2-x1);
   dy:=ABS(y2-y1);
   d:=2*dy-dx;
   incr1:=2*dy;  /* increment E */
   incr2:=2*(dy-dx);  /* increment

                      NE */
   IF x1>x2
      THEN BEGIN   /* start at
                      point with
                      smaller x */
         x:=x2; y:=y2; xend:=x1;
         END
      ELSE BEGIN
         x:=x1; y:=y1; xend:=x2;
         END
```

```
WritePixel(x,y,value);

   /*first point in line */
   WHILE x<xend DO BEGIN
      x:=x+1;
      IF d<0
         THEN d:=d+incr1;
            /* increment East */
         ELSE BEGIN
           /* increment NE */
            y:=y+1;
            d:=d+incr2;
           END
         WritePixel(x,y,value);
      END      /*while*/
   END        /*Bresenham*/
```

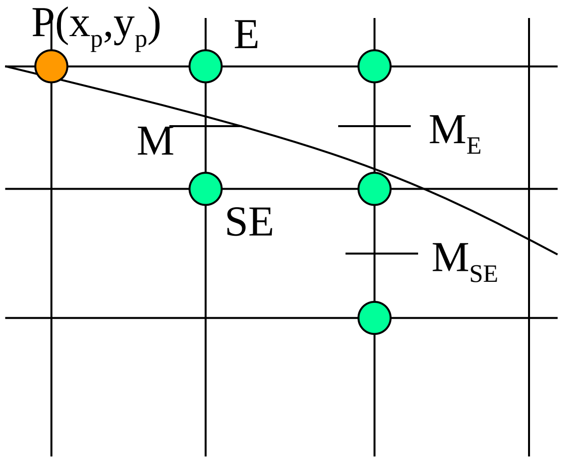
Circle rasterization

- Problem: Given the circle $x^2+y^2=r^2$ draw its rasterization
- The most common algorithm for drawing circles was developed by Bresenham
- Consider the second octant, from x=0 to x=y=r/sqrt(2)
- Let $F(x,y)=x^2+y^2-r^2$:
  $F>0$ outside the circle
  $F<0$ inside the circle
-



Circle rasterization

- Problem: Given the circle $x^2+y^2=r^2$ draw its rasterization
- The most common algorithm for drawing circles was developed by Bresenham
- Consider the second octant, from x=0 to x=y=r/sqrt(2)
- Let $F(x,y)=x^2+y^2-r^2$:
  F>0 outside the circle
  F<0 inside the circle
-

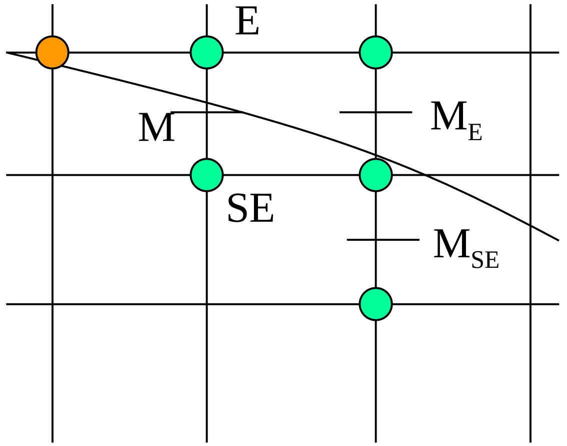


One can show that:

- if the midpoint between E and SE is outside the circle, then SE is closer to the circle
- Similarly, if the midpoint is inside the circle, then E is closer to the circle

Circle rasterization

- We choose a decision variable d, which is the value of the function at the midpoint
  $d_{old}=F(x_p+1, y_p-1/2)=$
  $(x_p+1)^2+(y_p-1/2)^2-r^2$
- If d<0, then E is chosen and the increment is
  $d_{new}=F(x_p+2, y_p-1/2)=$
  $(x_p+2)^2+(y_p-1/2)^2-r^2$
- Thus, $d_{new}=d_{old}+2x_p+3$, and the increment in case of E is
  $\Delta_E=2x_p+3$.
- If instead d>=0, then SE is chosen, and the next midpoint will be incremented by
  $d_{new}=F(x_p+2, y_p-3/2)=$
  $(x_p+2)^2+(y_p-3/2)^2-r^2$
- Thus, $d_{new}=d_{old}+2x_p-2y_p+5$, and
  $\Delta_{SE}=2x_p+ -2y_p+5$
- Here, the two Δ increments vary from step to step
- Otherwise it is similar to line drawing
- All it needs now is to compute a starting point and we are set



Circle rasterization

```
PROCEDURE
   MidpointCircle(radius,
         value: integer);
   var x, y: INTEGER; d:REAL
BEGIN
   x:=0; y:= radius;
   d:=5/4/radius;
   DrawPixel(x,y,value);
   WHILE y>x DO
   BEGIN
      IF(d<0) THEN
      BEGIN    /* select E */
         d:=d+2*x+3;
         x:=x+1;
      END
```

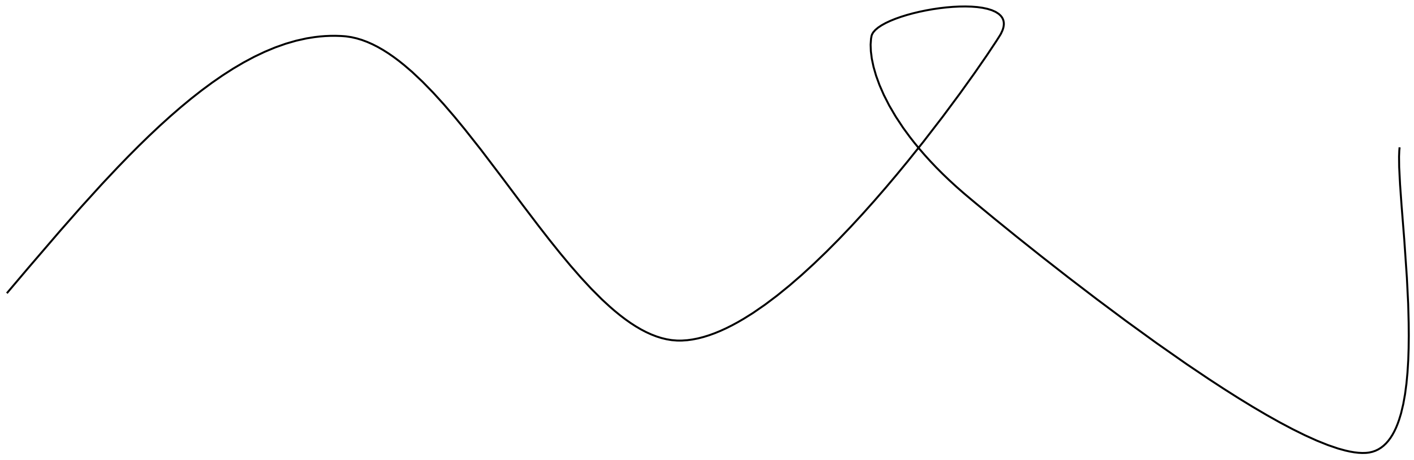
```
ELSE
      BEGIN   /* select SE */
         d:=d+2*(x-y)+5;
         x:=x+1; y:=y+1
      END
      WritePixel(x,y,value);
   END         /* While */
END
```

Circle drawing

- Also this algorithm can be integerized and perfectioned
- This by using second order differences
- Note that ellyppses can be drawn in a similar way

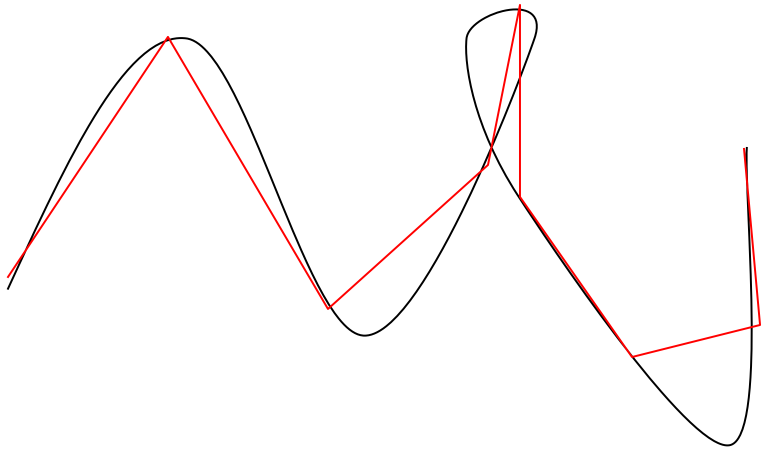
Higher order curves

- Suppose we want to rasterize a higher order curve: $x=f(t)$ $y=g(t)$ $(t \in [0,1])$



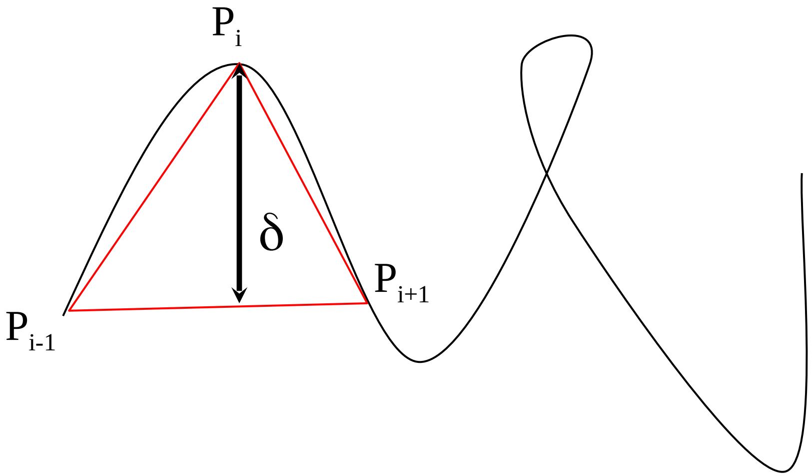
Higher order curves

- Usually, hardware companies would simply subdivide the interval parameter into equal parts (0, 1/n, 2/n …,1)
- Then evaluate the curve at these parameter values
- Finally plot the polyline of the points
- Prone to miss detail of the curve



Higher order curves

- A better method is to use adaptive steps
- Consider three consecutive samples $P_{i-1}P_iP_{i+1}$
- If the distance δ is bigger than a certain threshold, then I simply half the step
- If it is smaller, then I try doubling the step

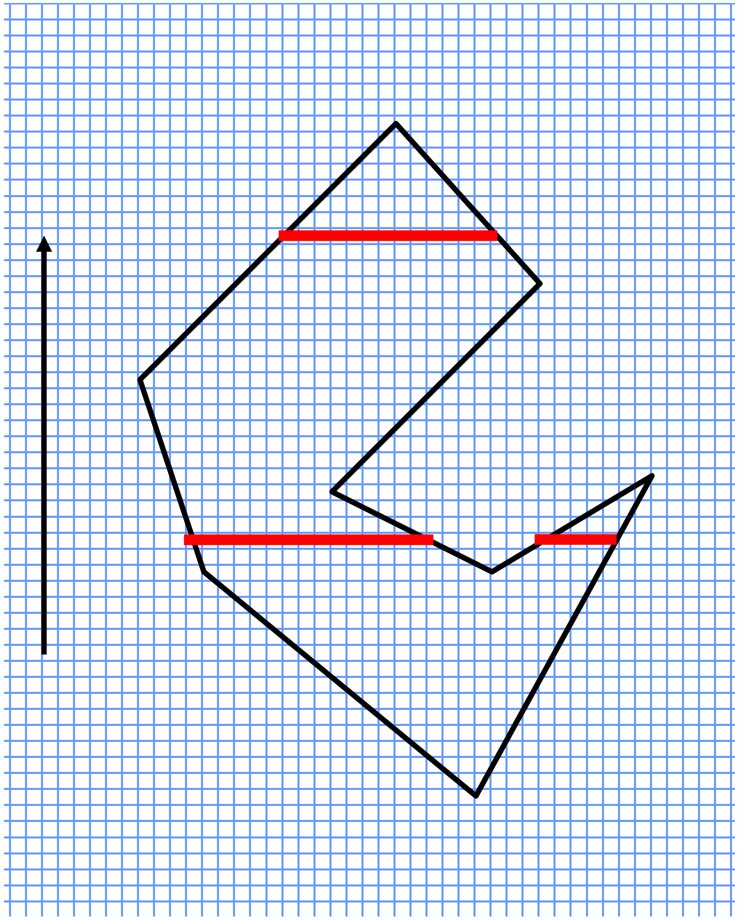


Polygon Rasterization

- In general, except if we are dealing with wireframes, we would want to draw a filled polygon on our screen.
- The advantage is clear: the polygon acquires thickness and can be use to render surfaces
- The simplest way one would do that is to draw the polygon border and then fill the region delimited by the polygon
- In fact, this is the start point for the real algorithm, the scanline algorithm
- The scanline algorithm combines the advantages of filling algorithms and of line tracing at the borders in a complex but very fast way
- As input one takes an ordered list of points representing the polygon

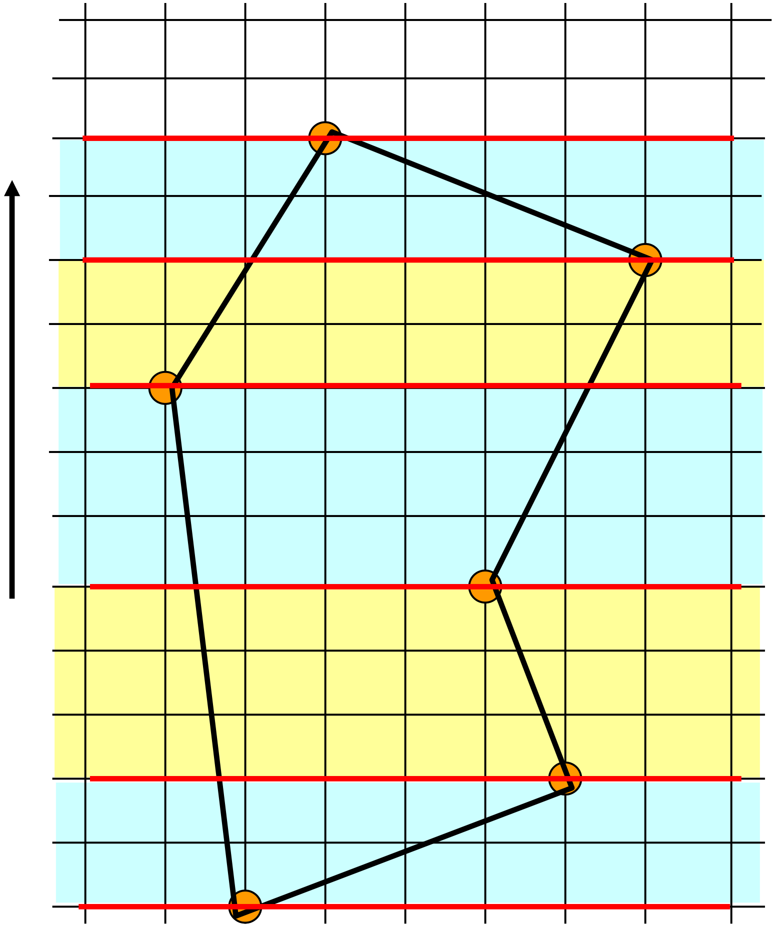
Scanline algorithm

- The basic idea is very simple:
  - A polygon can be filled one scanline at a time, from top to bottom
  - Order therefore polygon corners according to their highest y coordinate
  - Order each horizonal line according to the x coordinate of the edge intersections
  - Fill between pairs of edges, stop drawing until the next edge, and then restart filling again till the next one
  - once finished the edges at current line, restart at next y value
  - Of course, one can also draw upwards



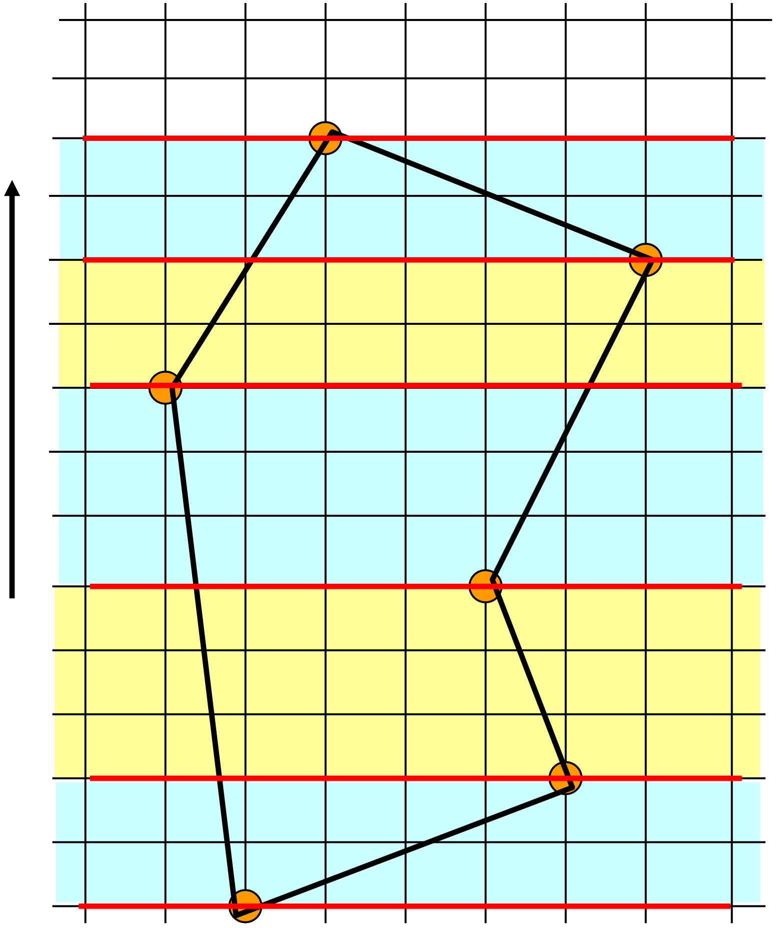
Scanline algorithm

- Notice that the number of edges remains constant between starting and ending points in the horizontal bands.
- Notice also that segments have only a limited contiguous range where they are active
- Notice that while proceeding downwards, borders can use a mirrored DDA to be drawn
- In this way, one can draw line borders and fill between them, after having ordered the border intersections with the current line WRT current coordinate



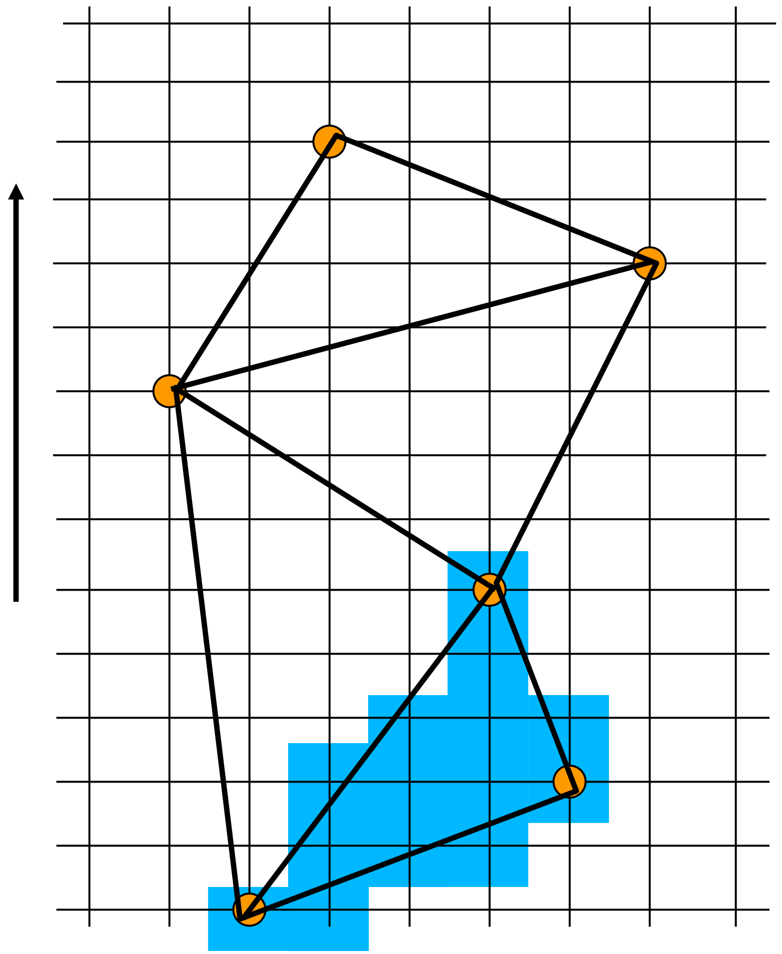
Scanline algorithm

- Polygon drawing starts at the bottom.
- Out of the edges list the ones with lowest starting point are chosen.
- These will remain part of the „active edge“ list until their end is met
- When they end, they are removed and replaced by new starting edges
- This until there is no edge left among the active edge
- At each value of the y variable, the edge rasterization is computed, and edges are ordered by growing x
- Colour is then filled between sorted pairs of edge rasterizations.



Triangle rasterization

- Modern graphics cards accept only triangles at the rasterization step
- Polygons with more edges are simply triangularized
- Obviously, the rasterization of a triangle is much easier
- This because a triangle is convex, and therefore a horizontal line has just the left and the right hand borders
- Filling is then done between the left side and the right side



End

+++ Ende - The end - Finis - Fin - Fine +++ Ende - The end - Finis - Fin - Fine +++