CS 4204 Computer Graphics

Scan Conversion

Yong Cao

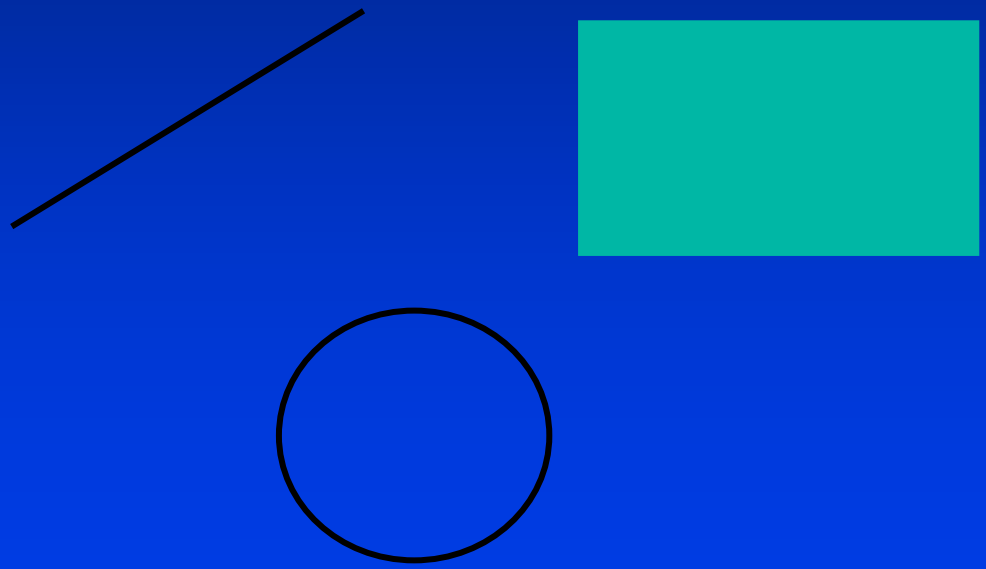
Virginia Tech

References:

"Introduction to Computer Graphics" course notes by Petros Faloutsos, UCLA

Primitives

Representations for Lines and Curves



Representations for lines and Curves

Line (in 2D)

- Explicit

$$y = \frac{dy}{dx}(x - x_0) + y_0$$

- Implicit

$$F(x,y) = (x - x_0)dy - (y - y_0)dx$$

if	$F(x,y) = 0$	then	(x,y) is on line
	$F(x,y) > 0$		(x,y) is below line
	$F(x,y) < 0$		(x,y) is above line

- Parametric

$$x(t) = x_0 + t(x_1 - x_0)$$
$$y(t) = y_0 + t(y_1 - y_0)$$
$$t \in [0,1]$$

$$P(t) = P_0 + t(P_1 - P_0), \text{ or}$$
$$P(t) = (1 - t)P_0 + tP_1$$

Circle

- Explicit

$$y = \pm\sqrt{r^2 - x^2}, \quad |x| \leq r$$

- Implicit

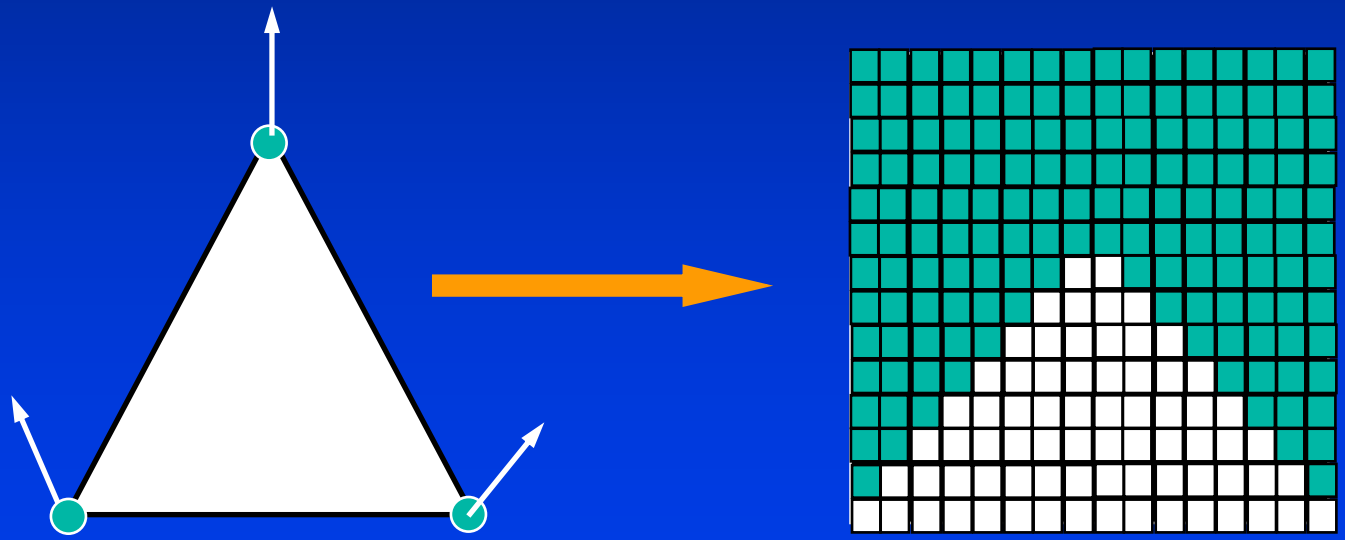
$$x^2 + y^2 = r^2$$
$$F(x, y) = x^2 + y^2 - r^2$$

if	$F(x, y) = 0$	then	(x, y) is on circle
	$F(x, y) > 0$		(x, y) is outside
	$F(x, y) < 0$		(x, y) is inside

- Parametric

$$x(\theta) = r\cos(\theta)$$
$$y(\theta) = r\sin(\theta)$$
$$\theta \in [0, 2\pi]$$

Rasterization



Line rasterization

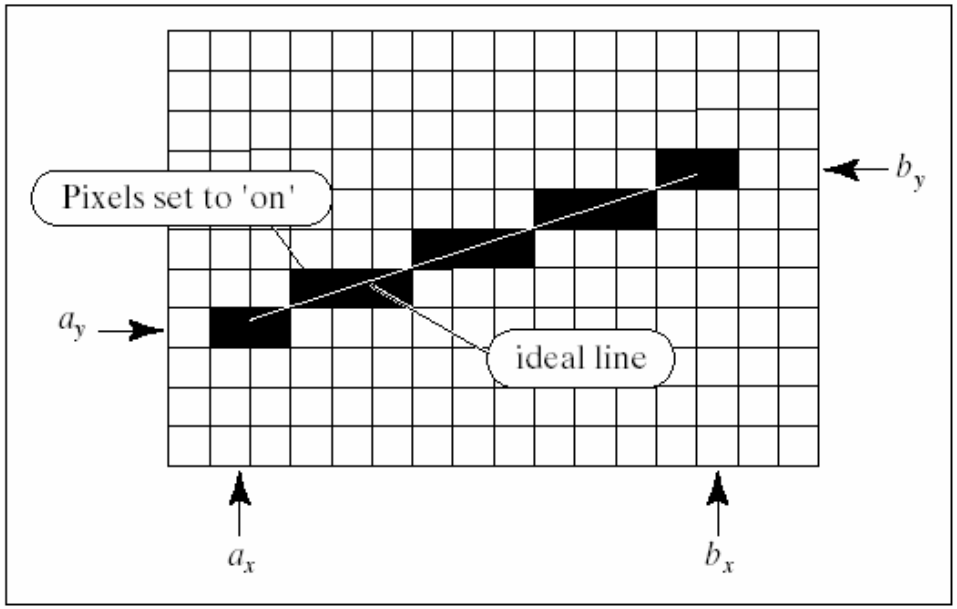


FIGURE 10.23 Drawing a straight-line-segment.

Line rasterization

Desired properties

- Straight
- Pass through end points
- Smooth
- Independent of end point order
- Uniform brightness
- Brightness independent of slope
- Efficient

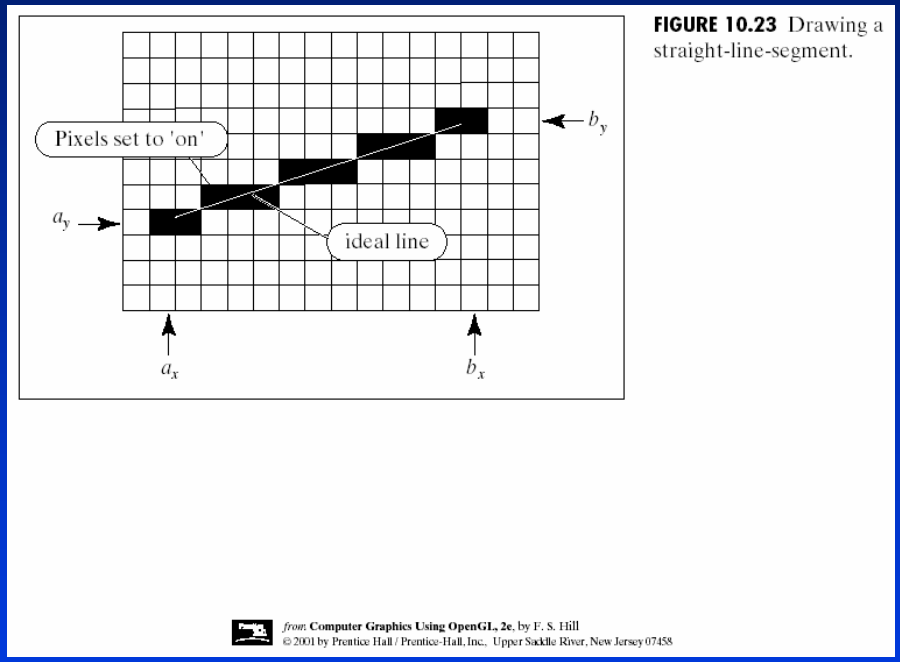


FIGURE 10.23 Drawing a straight-line-segment.

Straightforward Implementation

Line between two points

```
DrawLine(int x1,int y1,int x2,int y2)
    {
        float y;
        int x;

        for (x=x1; x<=x2; x++) {
                y = y1 +  (x-x1)*(y2-y1)/(x2-x1)
                SetPixel(x, Round(y) );

        }
    }
```

Better Implementation

How can we improve this algorithm?

```
DrawLine(int x1,int y1,int x2,int y2)
    {
        float y;
        int x;

        for (x=x1; x<=x2; x++) {
                y = y1 +  (x-x1)*(y2-y1)/(x2-x1)
                SetPixel(x, Round(y) );

        }
    }
```

Better Implementation

```
DrawLine(int x1,int y1,int x2,int y2)
    {
        float y,m;
        int x;
        dx = x2-x1 ;
        dy = y2-y1 ;
        m = dy/ (float) dx ;

        for (x=x1; x<=x2; x++) {

                y = y1 + m*(x-x1) ;
                SetPixel(x, Round(y) );
        }
    }
```

Even Better Implementation

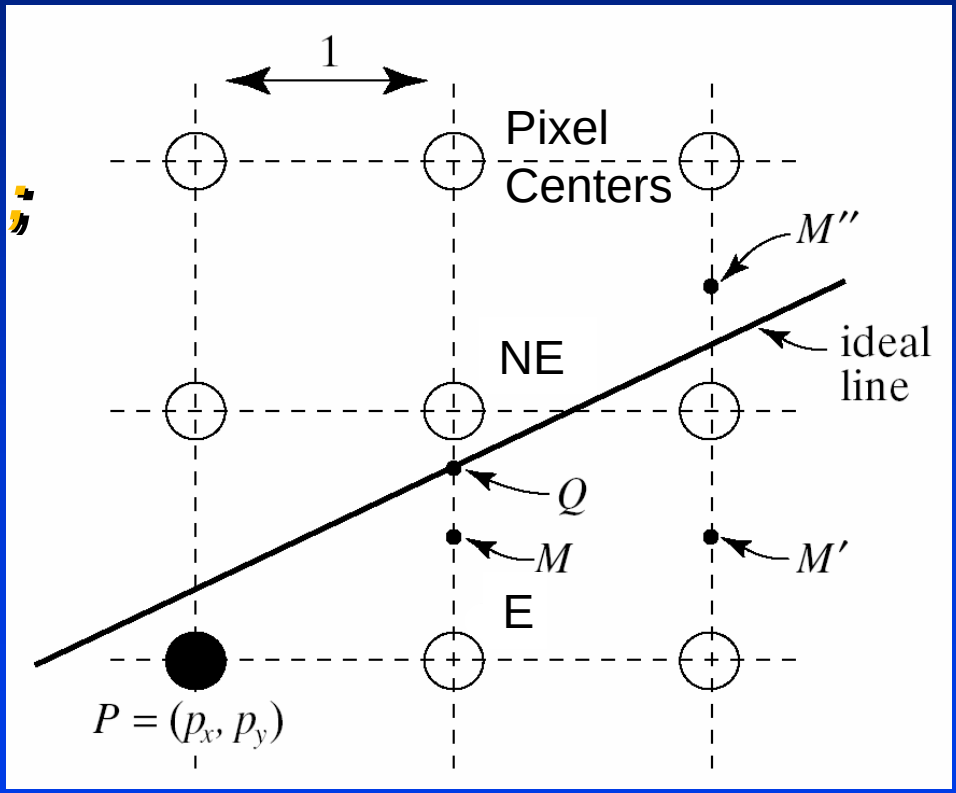
```
DrawLine(int x1,int y1,int x2,int y2)
    {
        float y,m;
        int x;
        dx = x2-x1 ;
        dy = y2-y1 ;
        m = dy/ (float) dx ;
        y = y1 + 0.5 ;

        for (x=x1; x<=x2; x++) {
                SetPixel(x, Floor(y) );
                y = y + m ;
        }
    }
```

Midpoint algorithm (Bresenham)

Line in the first quadrant (0<slope < 45 deg)

Implicit function:
F(x,y) = xdy - ydx + c,
dx,dy > 0 and dy/dx <= 1.0 ;

- Current choice P = (x,y).
- How do we chose next of P, P'= (x+1,y') ?

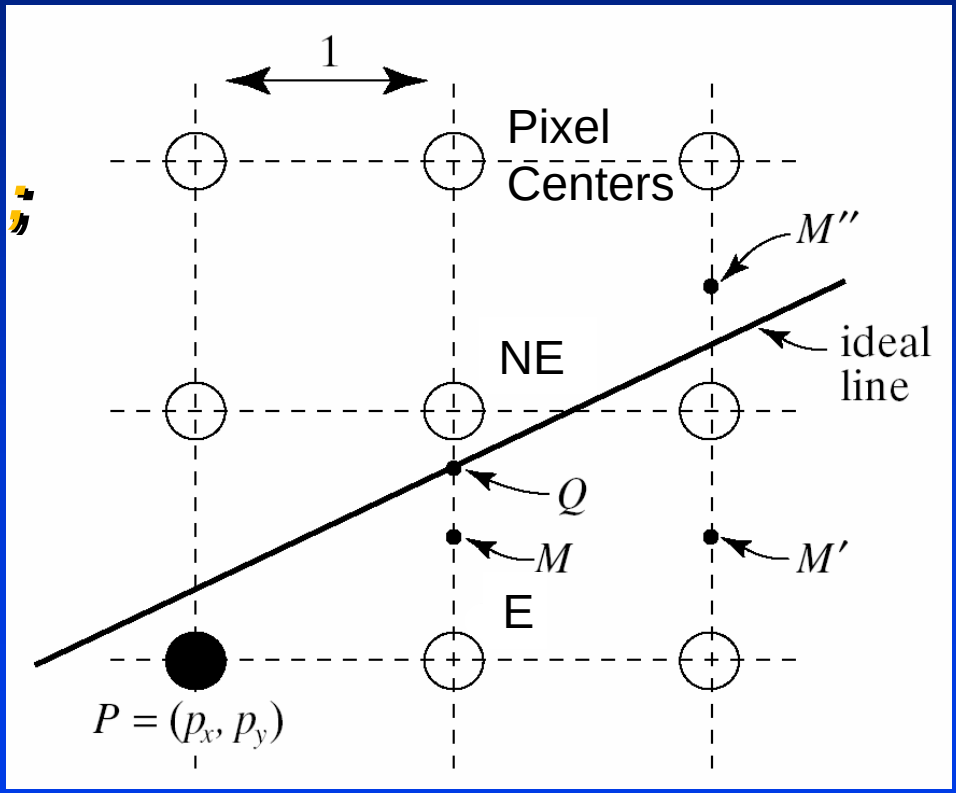


Midpoint algorithm (Bresenham)

Line in the first quadrant (0<slope < 45 deg)

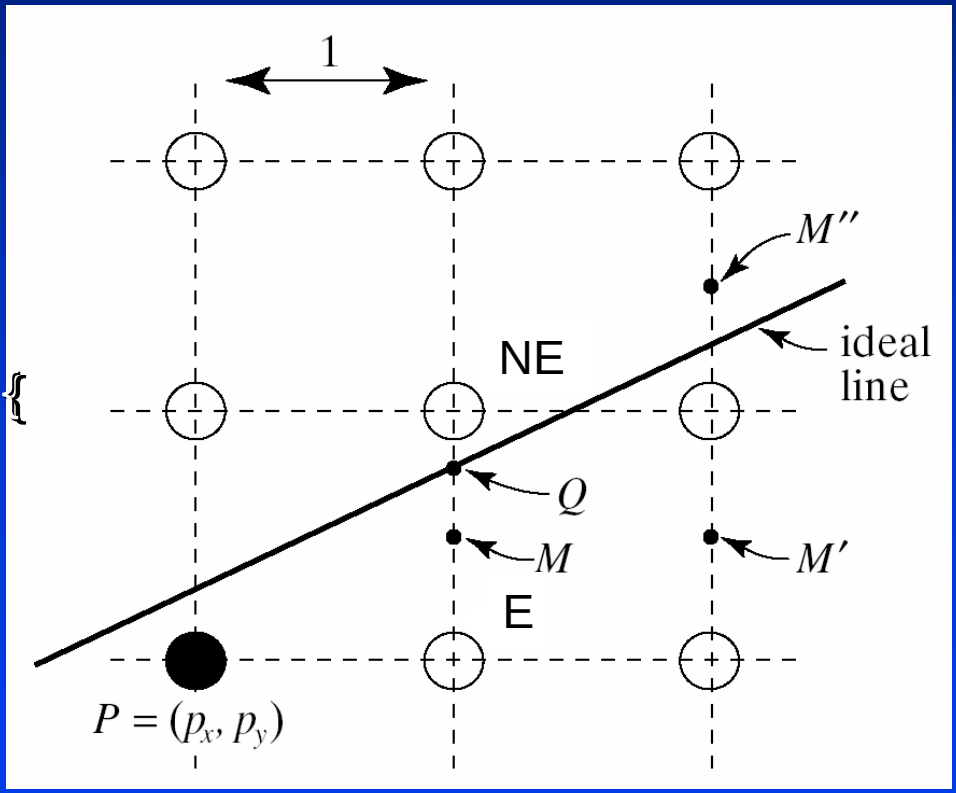
Implicit function:
F(x,y) = xdy - ydx + c,
dx,dy > 0 and dy/dx <= 1.0 ;

- Current choice P = (x,y).
- How do we chose next of P, P'= (x+1,y') ?
  If(F(M) = F(x+1,y+0.5) < 0)
  M above line so E
  else
  M below line so NE



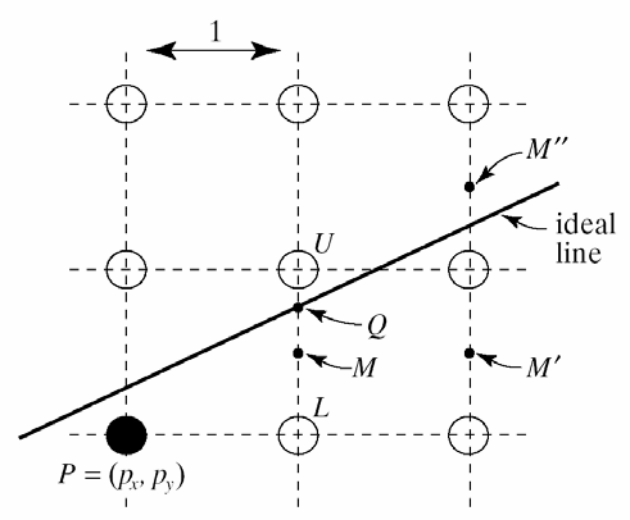
Midpoint algorithm (Bresenham)

```
DrawLine(int x1, int y1, int x2, int y2, int color)
    {
        int x,y,dx,dy;

        y = Round(y1) ;
        for (x=x1; x<=x2; x++) {
                SetPixel(x, y );
                if (F(x+1,y+0.5)>0) {
                        y = y + 1 ;
                }

        }
    }
```



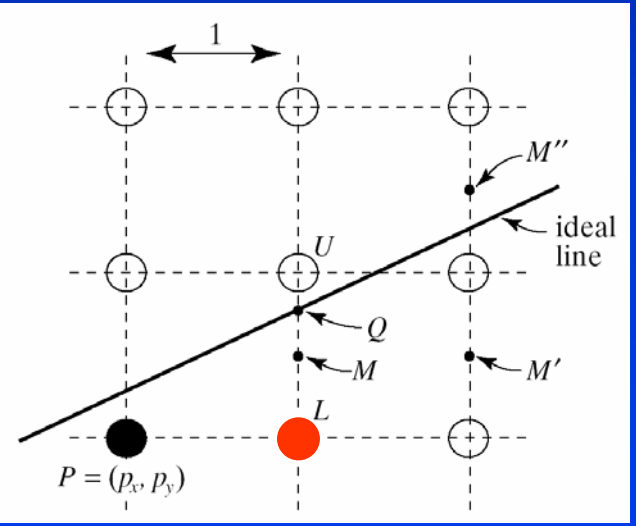
Can we compute F in a smart way?

- We are at pixel (x,y) we evaluate F at M = (x+1,y+0.5) and E=(x+1,y) or NE=(x+1,y+1) accordingly.
  (Reminder: F(x,y) = xdy - ydx + c)



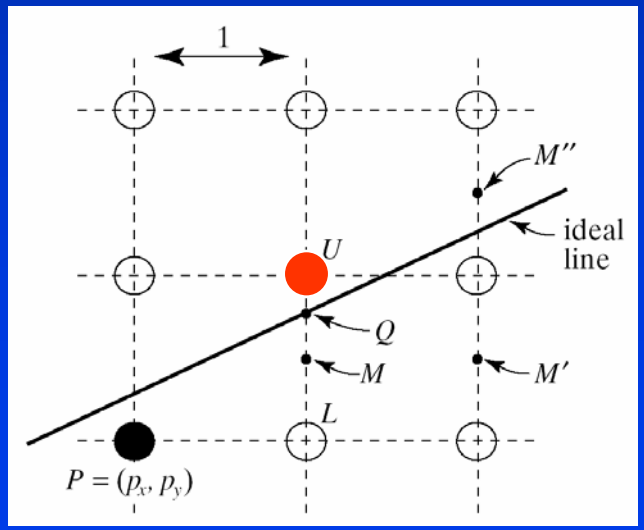
Can we compute F in a smart way?

- We are at pixel (x,y) we evaluate F at M = (x+1,y+0.5) and E=(x+1,y) or NE=(x+1,y+1) accordingly.
  (Reminder: F(x,y) = xdy - ydx + c)
- If we chose E for x+1 the next criteria will be at M':
  F(x+2,y+0.5) = [(x+1)dy +dy] - (y+0.5)*dx +c→
  F(x+2,y+0.5) = F(x+1,y+0.5) + dy →
  $F_E = F + dy = F + dF_E$



Can we compute F in a smart way?

- We are at pixel (x,y) we evaluate F at M = (x+1,y+0.5) and E=(x+1,y) or NE=(x+1,y+1) accordingly.
  (Reminder: F(x,y) = xdy - ydx + c)
- If we chose E for x+1 the next criteria will be at M':
  F(x+2,y+0.5) = (x+1)dy +dy - (y+0.5)*dx +c→
  F(x+2,y+0.5) = F(x+1,y+0.5) + dy →
  F_E = F + dy
- If we chose NE then the next criteria will be at M'':
  F(x+2,y+1+0.5) =
  F(x+1,y+0.5) + dy – dx →
  F_{NE} = F + dy - dx



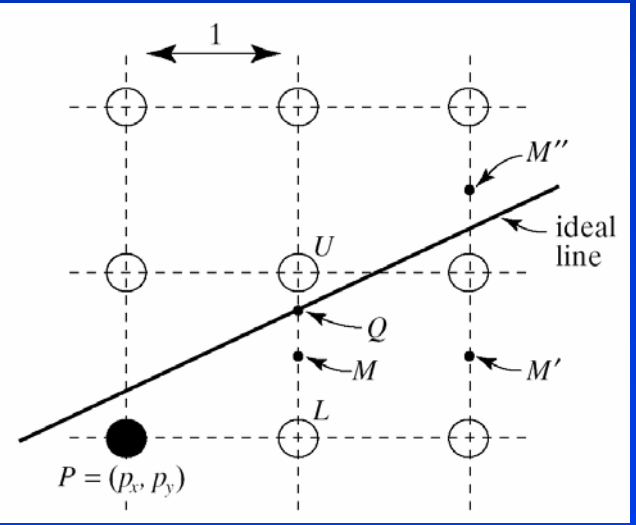
Can we compute F in a smart way?

- We are at pixel (x,y) we evaluate F at M = (x+1,y+0.5) and E=(x+1,y) or NE=(x+1,y+1) accordingly.
  (Reminder: F(x,y) = xdy - ydx + c)
- If we chose E for x+1 the next criteria will be at M':

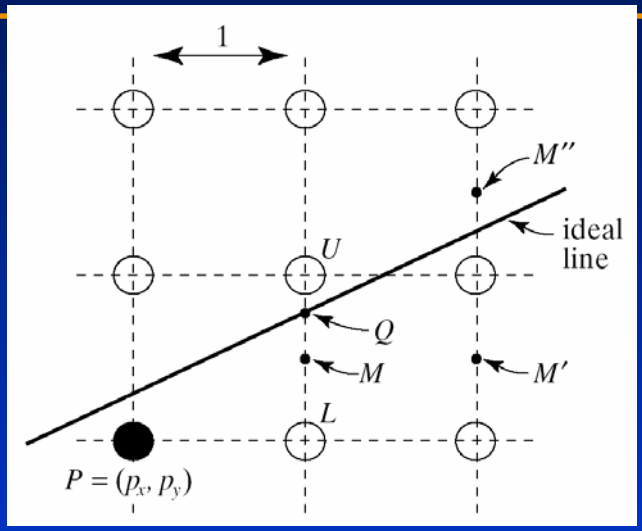
$F_E = F + dy$

- If we chose NE then the next criteria will be at M'':

$F_{NE} = F + dy - dx$



Criterion update



Update

$F_E = F + dy = F + dF_E$

$F_{NE} = F + dy - dx = F + dF_{NE}$

Starting value?

Line equation: $F(x,y) = xdy-ydx+c$

Assume line starts at pixel (x_0,y_0)

$F_{start} = F(x_0+1,y_0+0.5) = (x_0+1)dy -(y_0+0.5)dx + c =$

$= (x_0dy - y_0dx + c) + dy - 0.5dx = F(x_0,y_0) + dy - 0.5dx.$

(x_0,y_0) belongs on the line so: $F(x_0,y_0) = 0$

Therefore:

$F_{start} = dy - 0.5dx$

Criterion update (Integer version)

Update

$F_{start} = dy - 0.5dx$

$F_E = F + dy = F + dF_E$

$F_{NE} = F + dy - dx = F + dF_{NE}$

Everything is integer except F_{start}.

Multiply by 2 → $F_{start} = 2dy - dx$

$dF_E = 2dy$

$dF_{NE} = 2(dy-dx)$

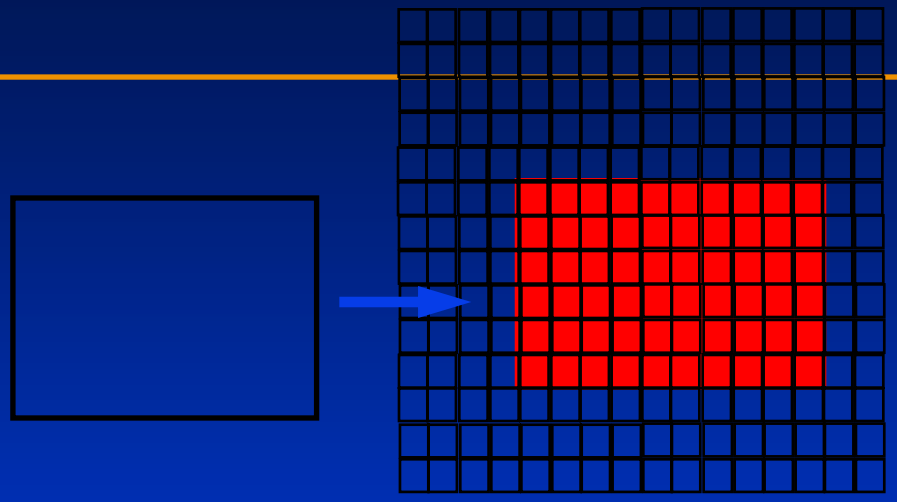
Midpoint algorithm

```
DrawLine(int x1, int y1, int x2, int y2, int color)
    {
        int x,y,dx,dy,dE, dNE;
        dx = x2-x1 ;
        dy = y2-y1 ;
        d = 2*dy-dx ; // initialize d
        dE = 2*dy ;
        dNE = 2*(dy-dx) ;
        y = y1 ;
        for (x=x1; x<=x2; x++) {
                SetPixel(x, y, color );
                if (d>0) {       // chose NE
                        d = d + dNE ;
                        y = y + 1 ;
                } else {         // chose E
                        d = d + dE ;
                }
        }
    }
```

Polygon Rasterization

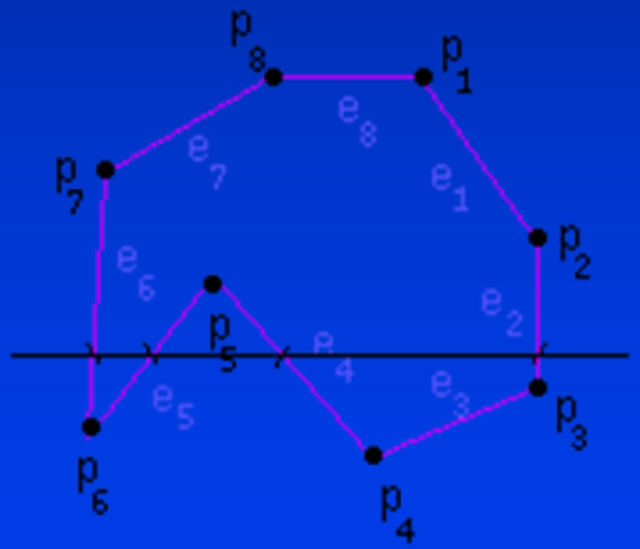
Scan conversion

shade pixels lying within a closed polygon efficiently.



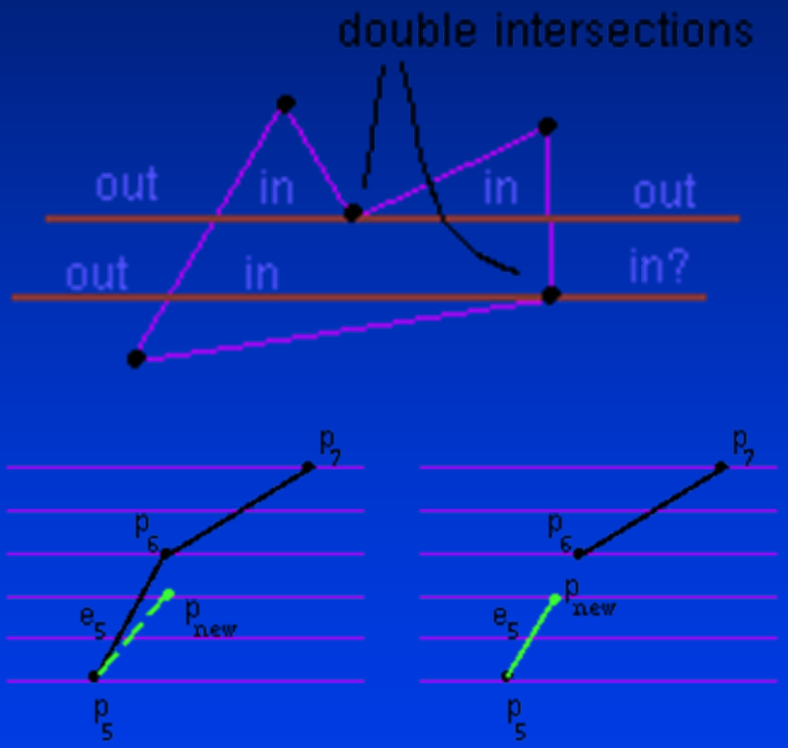
Algorithm

- For each row of pixels define a *scanline* through their centers
- intersect each scanline with all edges
- sort intersections in x
- calculate parity of intersections to determine in/out
- fill the 'in' pixels



Special cases

- Horizontal edges can be excluded
- Vertices lying on scanlines
  - *Change in sign of y_i-y_{i+1}: count twice*
  - *No change: shorten edge by one scanline*



Efficiency?

Many intersection tests can be eliminated by taking advantage of coherence between adjacent scanlines.

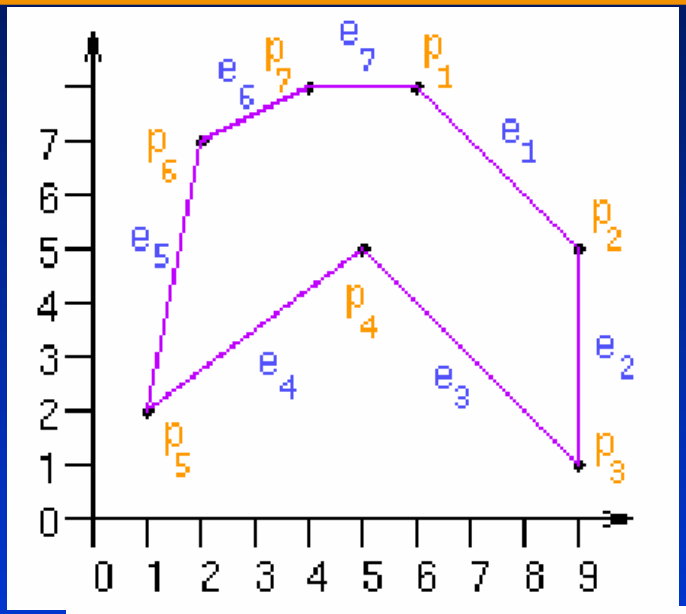
- Edges that intersect scanline y are likely to intersect y+1
- x changes predictably from scanline y to y+1

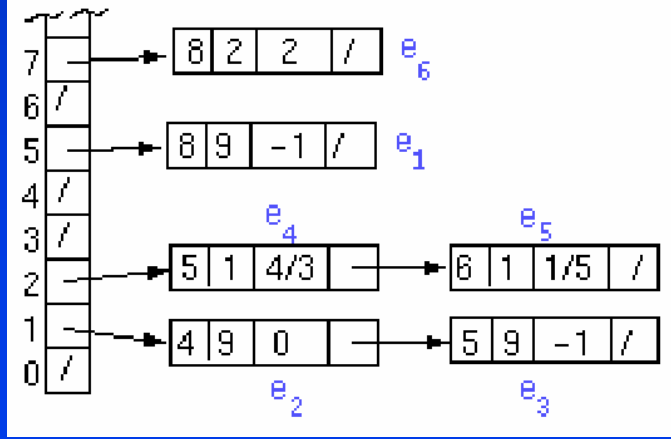
$$y = mx+a \rightarrow x = 1/m(y+a) \rightarrow x(y+1) = x(y) + 1/m$$

Data structure 1: Edge table

Building edge table

- Traverse edges
- Eliminate horizontal edges
- If not local extremum, shorten upper vertex
- Add edge to linked-list for the scanline corresponding to the lower vertex, storing:
  - *y_upper: last scanline to consider*
  - *x_lower: starting x coordinate for edge*
  - *1/m: for incrementing x; compute before shortening*





Data structure 2: Active Edge List (AEL)

- The AEL is a linked list of active edges on the current scanline, y.
- Each active edge has the following information:
  - *y_upper: last scanline to consider*
  - *x: edge's intersection with current y*
  - *1/m: for incrementing x*

The active edges are kept sorted by x.

Scan conversion algorithm

```
for each scanline
    add edges in edge table to AEL
    if AEL <> NIL
        sort AEL by x
        fill pixels between edge pairs
        delete finished edges
        update each edge's x in AEL
```

Example

for each scanline
    add edges in edge table to AEL
    if AEL <> NIL
        sort AEL by x
        fill pixels between edge pairs
        delete finished edges
        update each edge's x in AEL

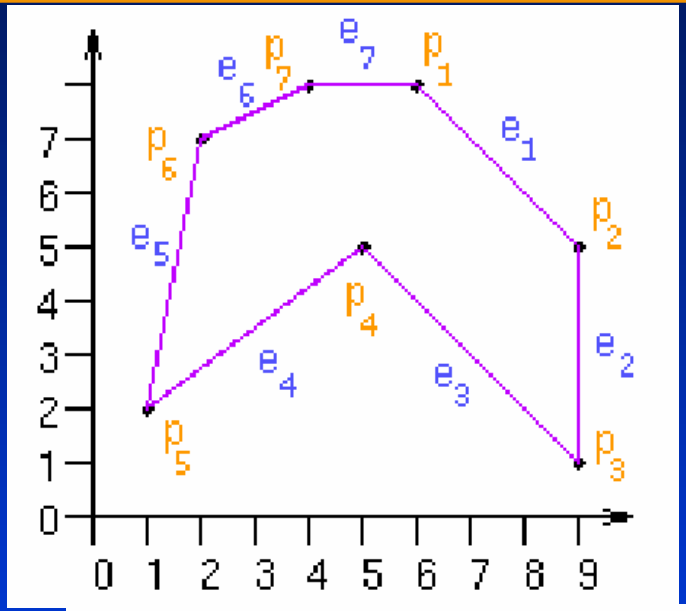
Reminder:

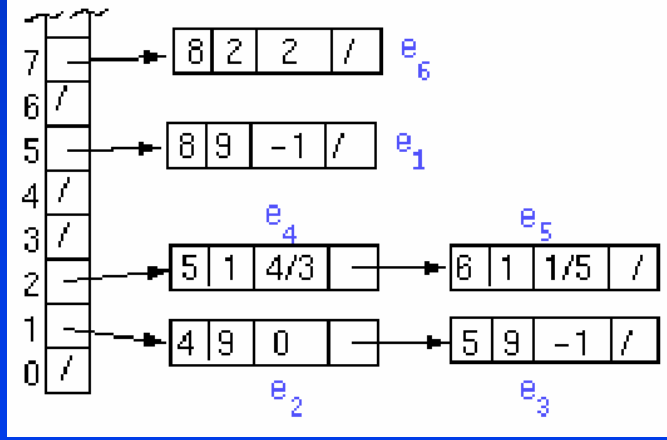
Edge table

y_upper	x_lower	1/m

AEL:

y_upper	x_current	1/m





Special cases

Triangles – Convex Polygons

- Maximum two edges per scanline

Overlaping polygons

- priorities

Color, patterns

Z for visibility

Interpolating information (incrementally)

Color, Normal, Texture coordinates

(x2,y2)

Right edge (1,2):

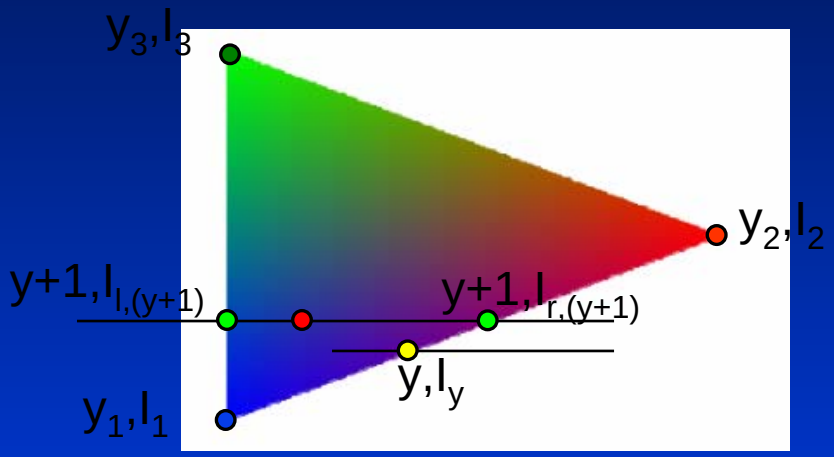
$$\frac{I_{r,(y+1)} - I_{r,y}}{(y+1) - y} = \frac{I_1 - I_2}{y_1 - y_2} \Rightarrow I_{r,(y+1)} = I_{r,y} + \frac{I_1 - I_2}{y_1 - y_2}$$

Left Edge (1,3):

$$\frac{I_{l,(y+1)} - I_{l,y}}{(y+1) - y} = \frac{I_1 - I_3}{y_1 - y_3} \Rightarrow I_{l,(y+1)} = I_{l,y} + \frac{I_1 - I_3}{y_1 - y_3}$$

Along scanline:

$$\frac{I_{(x+1)} - I_x}{(x+1) - x} = \frac{I_r - I_l}{x_r - x_l} \Rightarrow I_{r,(y+1)} = I_{r,y} + \frac{I_r - I_l}{x_r - x_l}$$



Interpolating information (incrementally)

Color, Normal, Texture coordinates

(x2,y2)

Right edge (1,2):

$$\frac{I_{r,(y+1)} - I_{r,y}}{(y+1) - y} = \frac{I_1 - I_2}{y_1 - y_2} \Rightarrow I_{r,(y+1)} = I_{r,y} + \frac{I_1 - I_2}{y_1 - y_2}$$

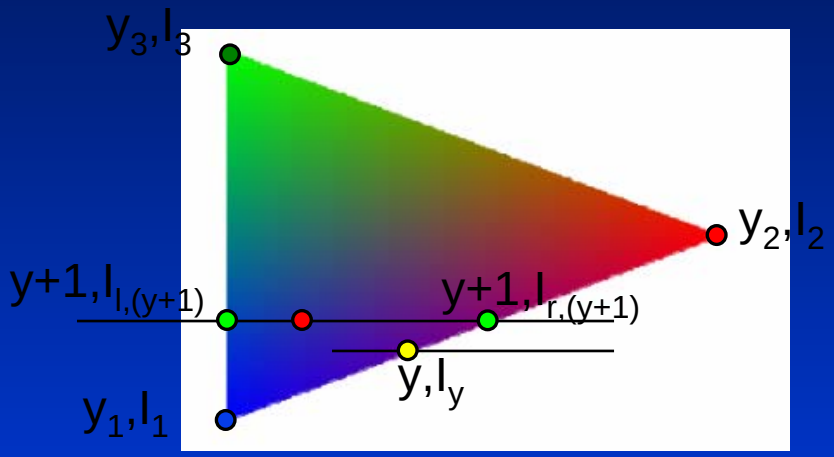
Left Edge (1,3):

$$\frac{I_{l,(y+1)} - I_{l,y}}{(y+1) - y} = \frac{I_1 - I_3}{y_1 - y_3} \Rightarrow I_{l,(y+1)} = I_{l,y} + \frac{I_1 - I_3}{y_1 - y_3}$$

Along scanline:

$$\frac{I_{(x+1)} - I_x}{(x+1) - x} = \frac{I_r - I_l}{x_r - x_l} \Rightarrow I_{r,(y+1)} = I_{r,y} + \frac{I_r - I_l}{x_r - x_l}$$

Constant along the line

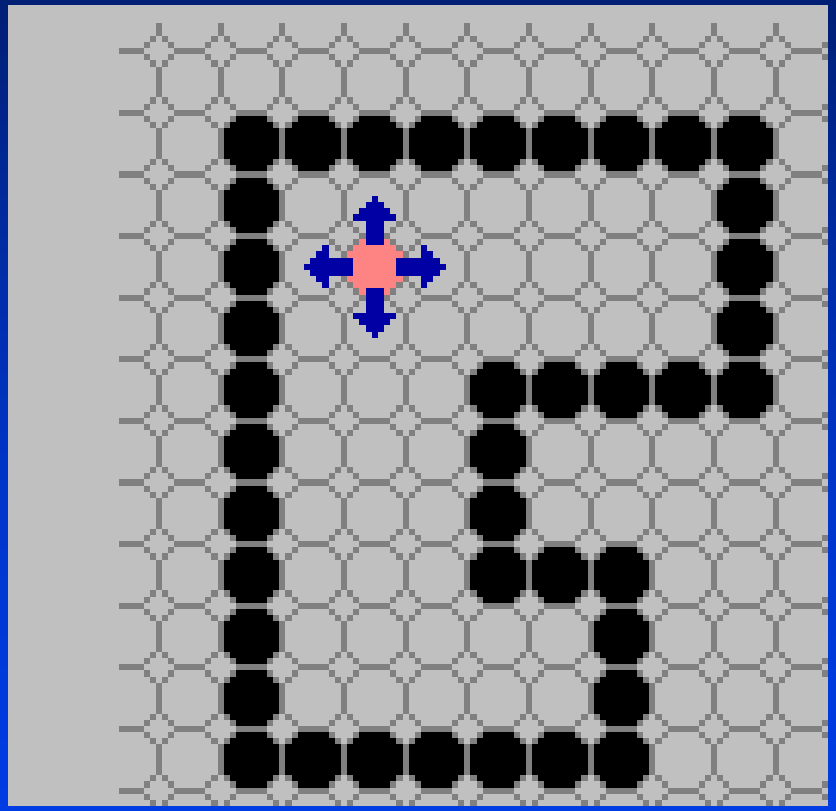


Pixel Region filling algorithms

Scan convert boundary

Fill in regions

2D paint programs



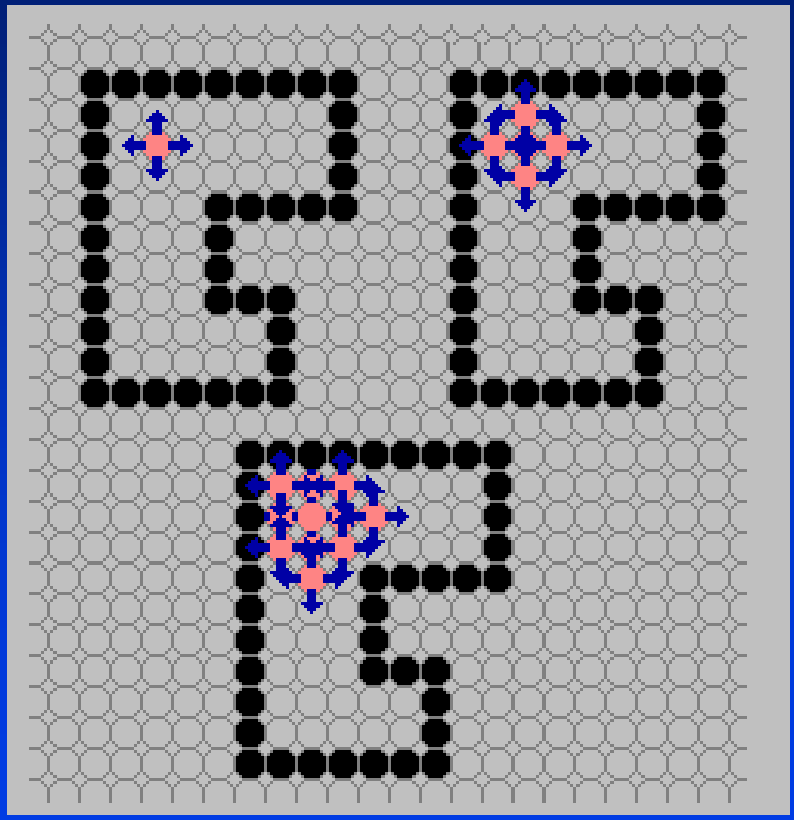
http://www.cs.unc.edu/~mcmillan/comp136/Lecture8/areaFills.html

BoundaryFill

```
boundaryFill(int x, int y, int fill, int boundary) {
        if ((x < 0) || (x >= raster.width)) return;
        if ((y < 0) || (y >= raster.height)) return;
        int current = raster.getPixel(x, y);
        if ((current != boundary) & (current != fill)) {
            raster.setPixel(fill, x, y);
            boundaryFill(x+1, y, fill,boundary);
            boundaryFill(x, y+1, fill, boundary);
            boundaryFill(x-1, y, fill, boundary);
            boundaryFill(x, y-1, fill, boundary);
        }

}
```

Flood Fill

```
public void floodFill(int x, int y, int fill, int old)
{
        if ((x < 0) || (x >= raster.width)) return;
        if ((y < 0) || (y >= raster.height))  return;

         if (raster.getPixel(x, y) == old) {
              raster.setPixel(fill, x, y);
              floodFill(x+1, y, fill, old);
              floodFill(x, y+1, fill, old);
              floodFill(x-1, y, fill, old);
              floodFill(x, y-1, fill, old);

         }
}
```



Adjacency

4-connected

8 connected