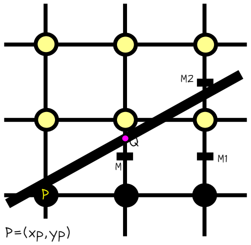
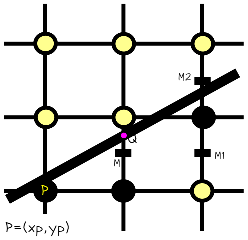
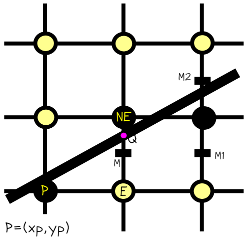
Computer Graphics



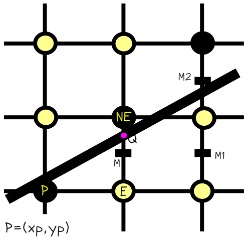
Pattern 1



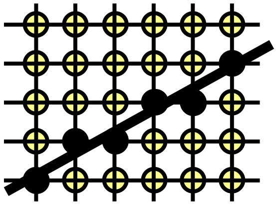
Pattern 2



Pattern 3

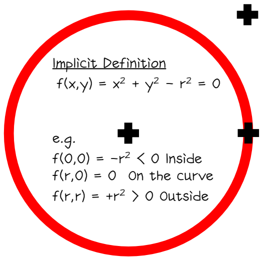


Pattern 4



Scan Conversion

Lines & circles



Implicit Definition

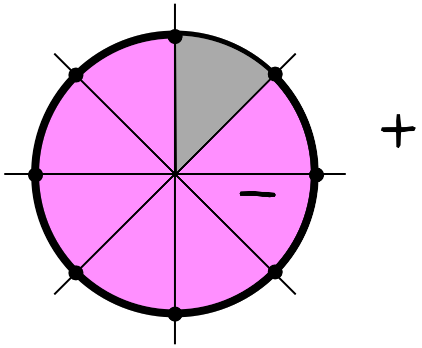
$f(x,y) = x^2 + y^2 - r^2 = 0$

e.g.

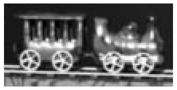
$f(0,0) = -r^2 < 0$ Inside

$f(r,0) = 0$ On the curve

$f(r,r) = +r^2 > 0$ Outside



by
Brian Wyvill

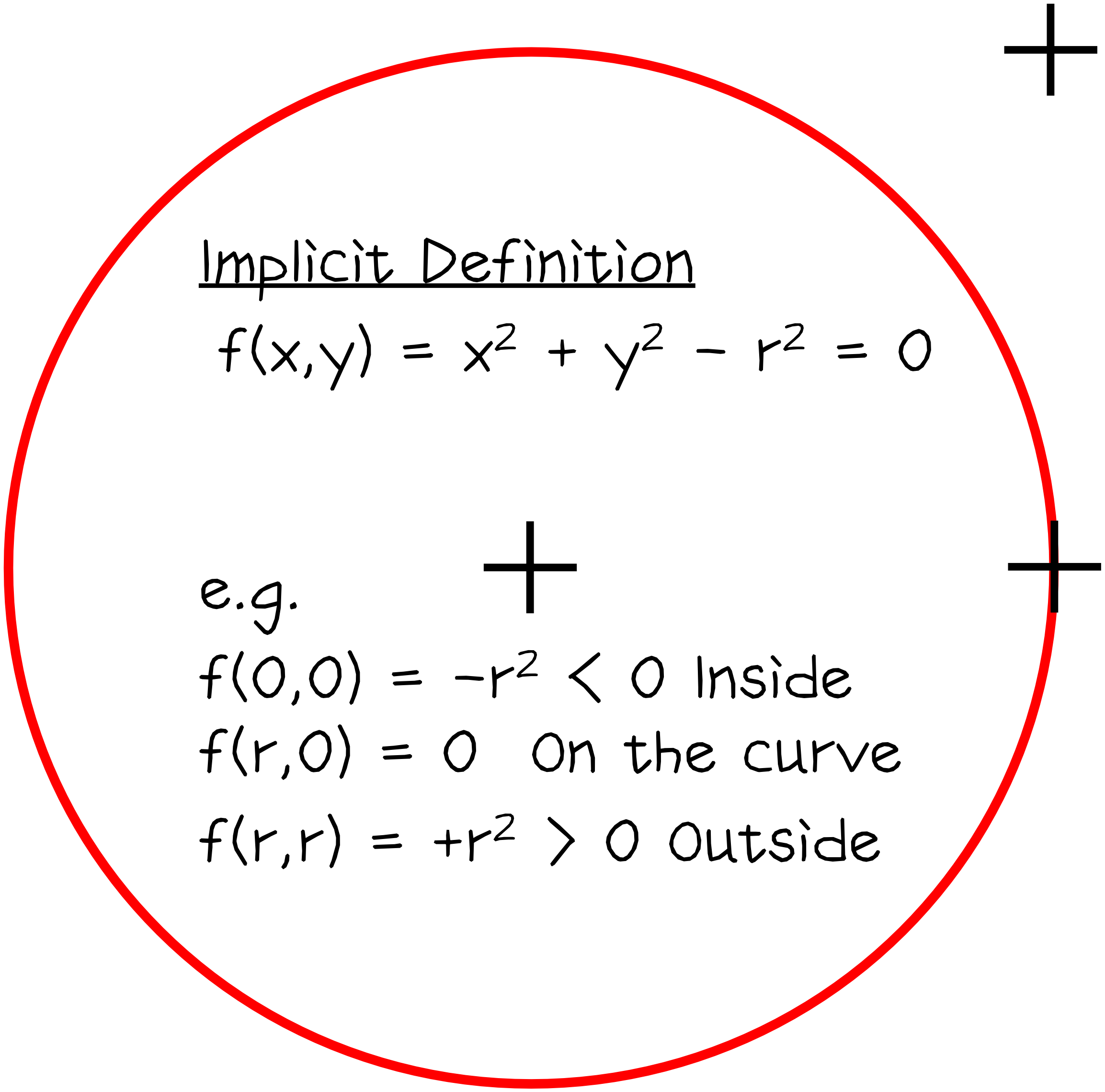


Implicit Equation

Parametric Definition

$x = r\cos(\alpha)$

$y = r\sin(\alpha)$



Implicit Definition

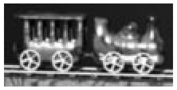
$f(x,y) = x^2 + y^2 - r^2 = 0$

e.g.

$f(0,0) = -r^2 < 0$ Inside

$f(r,0) = 0$ On the curve

$f(r,r) = +r^2 > 0$ Outside

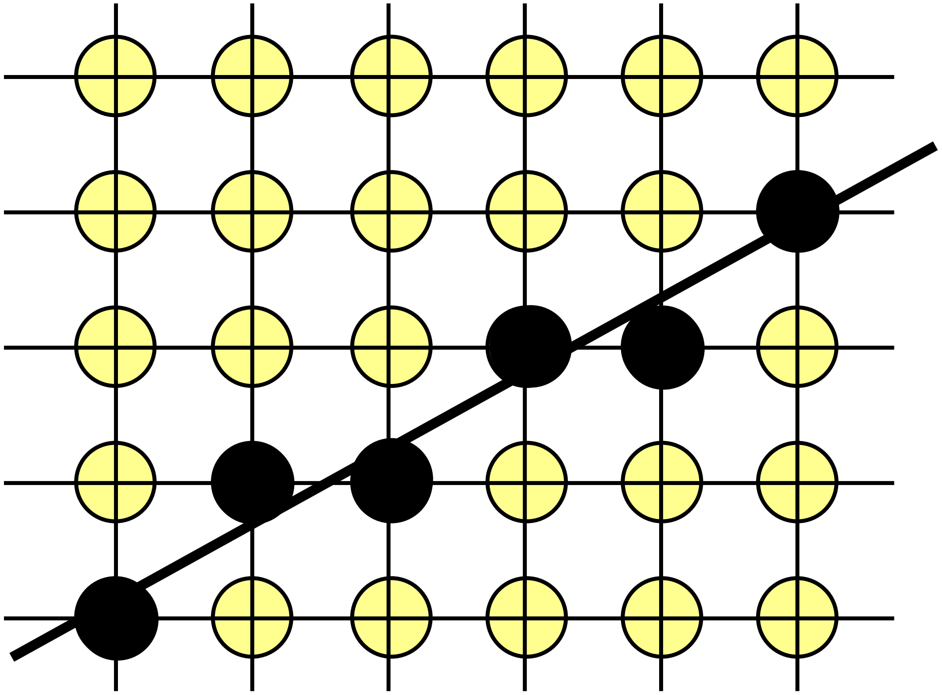


Graphics Primitives

Line Primitive

Problem is to show straight line on raster display

SCAN CONVERSION



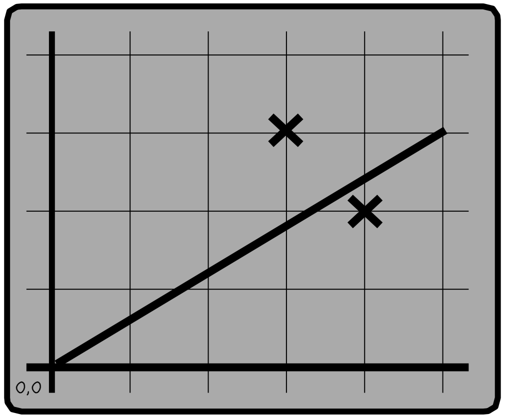
See F&VD.1 p73
and F&VD.2 p73

Implicit line equation

F(x,y) = ax + by + c = 0 ------(1)

Useful as the value of F(x,y) tells us which side of the line x,y lies.

E.g.: line (0,0) to (5,3)



Slope intercept form:

y = (dy/dx)x + B or

0 = dy.x − dx.y + dx.B

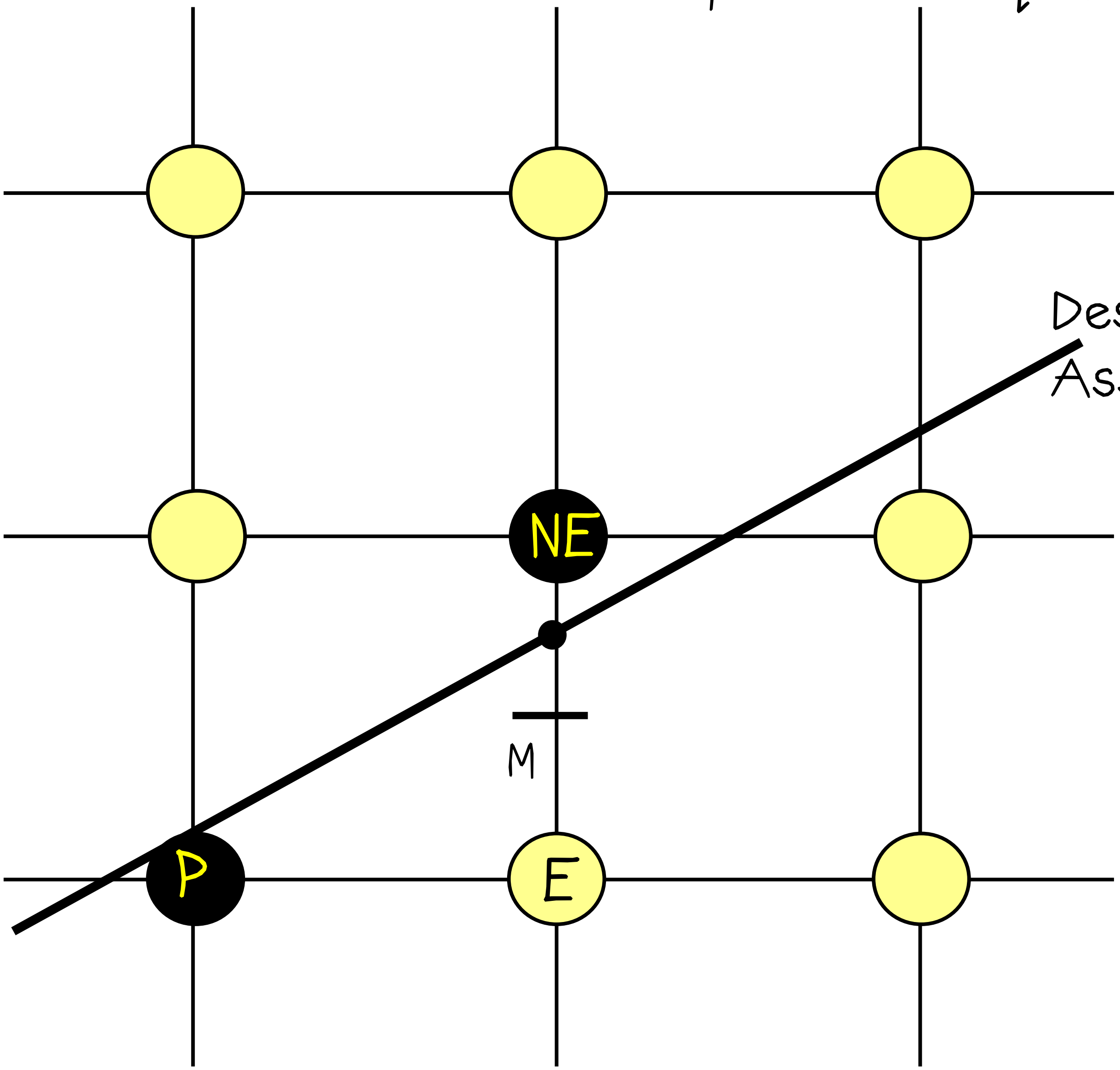
giving a = dy b = −dx c = B.dx in equation (1) in example dx=5 dy=3 B=0

F(3,3) = 3.3 − 5.3 = −6 < 0 above line

F(4,2) = 3.4 − 5.2 = 2 > 0 below line

Bresenham's Algorithm (Jack Bresenham 1965)

(midpoint technique)



Desired Line

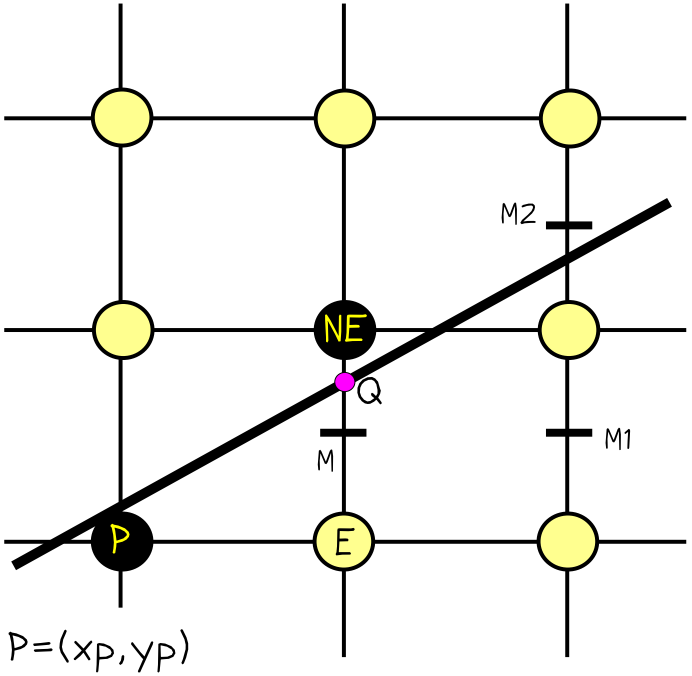
Assume 0 >= slope >= 1

Suppose we have chosen pixel P, which pixel should be next?

Since the slope is between 0 and 1 the next pixel will be either NE or E.

Bresenham's Algorithm

(Continued)



How to choose B or C, is Q closer to NE or E?

Slope = dy/dx

We need to compute $F(M) = F(xp+1,\ yp+\frac{1}{2})$

Define <u>discriminant</u> $d = F(xp+1,\ yp+\frac{1}{2})$

$d_M = a.(xp+1) + b.(yp+\frac{1}{2}) + c$

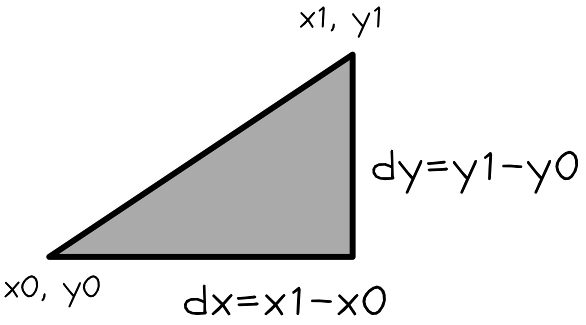
We wish to make an incremental algorithm:

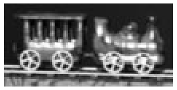
if d > 0 choose pixel NE, find M2

new $d_{M2} = F(xp+2,\ yp+\frac{3}{2}) = a.(xp+2) + b.(yp+\frac{3}{2}) + c$

if d < 0 choose pixel E, find M1

new $d_{M1} = F(xp+2,\ yp+\frac{1}{2}) = a.(xp+2) + b.(yp+\frac{1}{2}) + c$





Bresenham's Algorithm

Calculating the disciminant d

if d < 0 choose pixel E,

new $d_{M1} = a.(xp+2) + b.(yp+\frac{1}{2}) + c$

old $d_{M} = a.(xp+1) + b.(yp+\frac{1}{2}) + c$

Subtracting: $d_{M1} = d_{M} + a$
but a = dy known as ΔE

if d > 0 choose pixel NE, find M2

new $d_{M2} = F(xp+2, yp+\frac{3}{2})$
$= a.(xp+2) + b.(yp+\frac{3}{2}) + c$

Subtracting d_M gives:

$d_{M2} = d_M + a + b$
$\Delta NE = a + b = dy - dx$

Initial value of d

First pixel is endpoint xo, yo
First midpoint is $(xo+1, yo+\frac{1}{2})$
Find d by choosing E or NE:

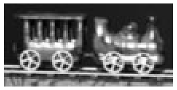
$$F(xo+1, yo+\frac{1}{2}) = a(xo+1) + b(yo+\frac{1}{2}) + c$$
$$= axo + byo + c + a + b/2$$
$$= F(xo, yo) + a + b/2$$

Since F(xo, yo) is on the line F(xo, yo) = 0
initial value of d = a + b/2 = dy − dx/2

Since we don't want the $\frac{1}{2}$ in the expression multiply by 2 which has no effect on the sign of d:

initial d = 2dy − dx

Increment of ΔE = 2dy
Increment of ΔNE = 2(dy − dx)



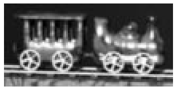
Bresenham's Algorithm

Midpoint Variation

```
void midpoint-line(int xo, int yo, int xe,
                   int ye, int value)
{
     int dx = xe - xo;
     int dy = ye - yo;
     int d = 2 * dy - dx;
     int incrE = 2 * dy;
     int icrNE = 2 * (dy - dx);
     int x = xo;
     int y = yo;
     writePixel(xo,yo, value);
```

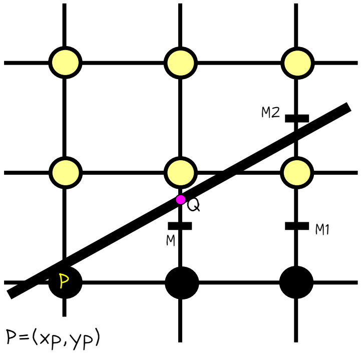
```
  while (x<xe) {
        if (d<=0) {   /* choose E */
            d+=incrE;
            x++;
        } else {        /* choose NE */
            d += incrNE;
            x++;
            y++;
        }
        writePixel(x,y, value);
  } /* end while */

  } /* end midPoint line */
```

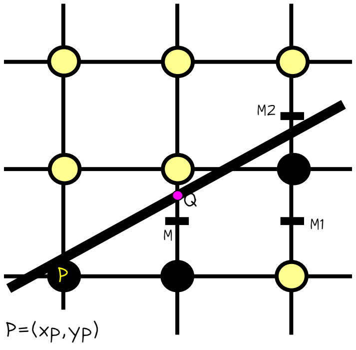


Double Step Line Algorithm X Wu (1987)

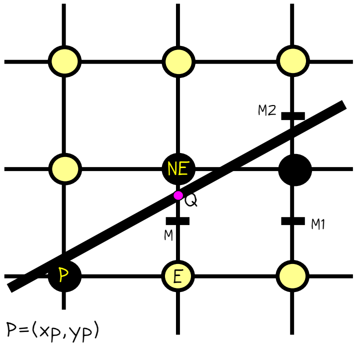
Symmetric Variation (blob 1987)



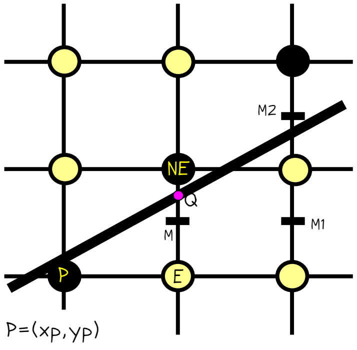
Pattern 1



Pattern 2



Pattern 3

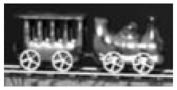


Pattern 4

Bresenham's chooses next pixel with one discriminator.
Wu chooses next pattern of two pixels with one discriminator.
By drawing from both ends of line at once 4 pixels can be chosen
with one discriminator.

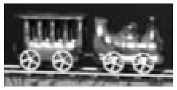
Observe that for 0 < slope <= 1/2 choose pattern 1 or (pattern 2 or 3)
for 1/2 < slope <= 1 choose pattern 4 or (pattern 2 or 3)

This can be done with only one discriminator calculated in the loop.
See code in course directory. For a full derivation see:
Fast Line Scan-Conversion, J.G. Rokne and B. Wyvill and X. Wu,
"ACM Transactions on Graphics", pp376-388, 9, 4, 1990

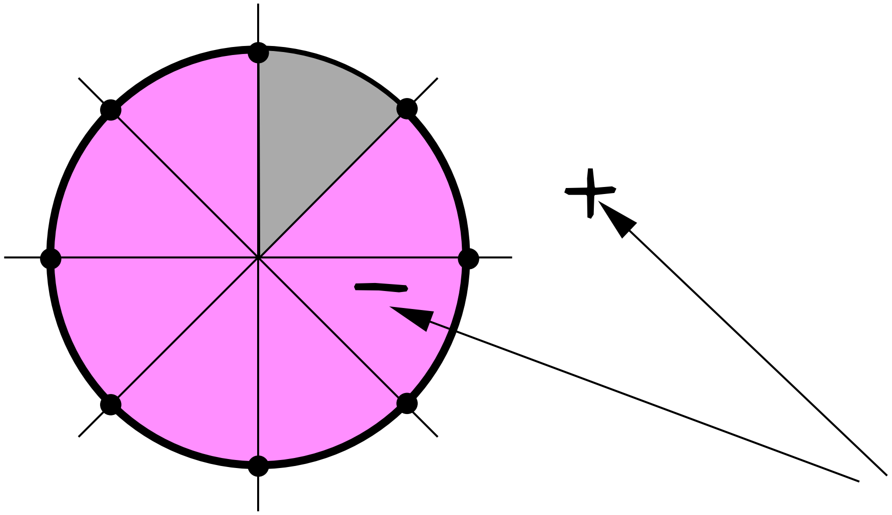


Double Step code for slope < 1/2

```
for (i = 0; i < xend; i++) {     /* plotting loop */
          ++x;
          --x1;
          if (D < 0) {  /* pattern 1 forwards */
                    plot(x, y, reverse);
                    plot(++x, y, reverse);
                      /* pattern 1 backwards */
                    plot(x1, y1, reverse);
                    plot(--x1, y1, reverse);
                    D += incr1;
          } else {
                    if (D < c) { /* pattern 2 forwards */
                              plot(x, y, reverse);
                              plot(++x, y += step, reverse);
                              /* pattern 2 backwards */
                              plot(x1, y1, reverse);
                              plot(--x1, y1 -= step, reverse);
                    }  else {
                              /* pattern 3 forwards */
                              plot(x, y += step, reverse);
                              plot(++x, y, reverse);
                              /* pattern 3 backwards */
                              plot(x1, y1 -= step, reverse);
                              plot(--x1, y1, reverse);
                    }
                    D + = incr2; /* only one discriminator incremented */
          }
}/* end for */
```



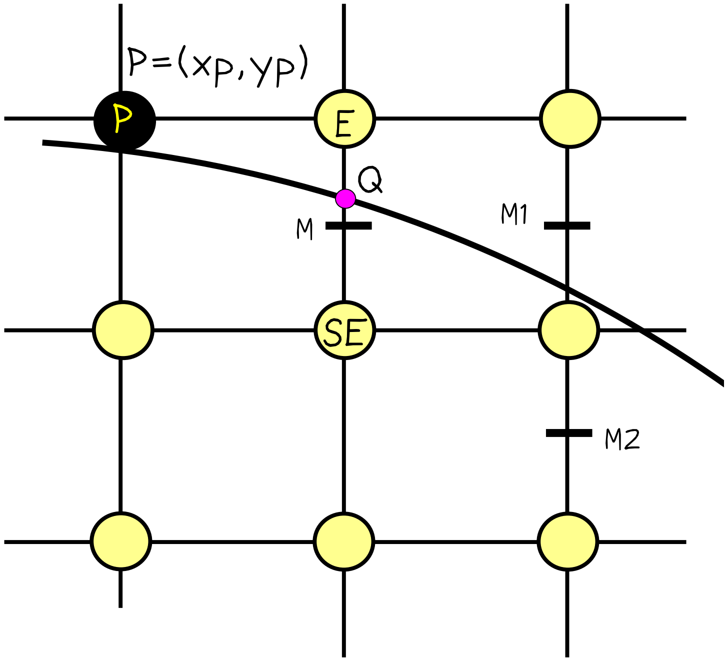
Scan Converting Circles



Only need consider 45° of the circle use symmetry to find other points.

E.g. consider octant from $x=0$ to $x=y=R/\sqrt{2}$
Choose which of two points closer to the midpoint:

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



d (d_M) is the value at the mid-point

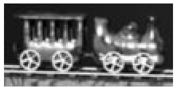
$$d = F(x_p+1, y_p-\frac{1}{2}) = (x_p+1)^2 + (y_p-\frac{1}{2})^2 - R^2$$

if $d < 0$ (M inside) choose E then

$$d_{M1}=F(x_p+2, y_p-\frac{1}{2}) = (x_p+2)^2 + (y_p-\frac{1}{2})^2 - R^2$$

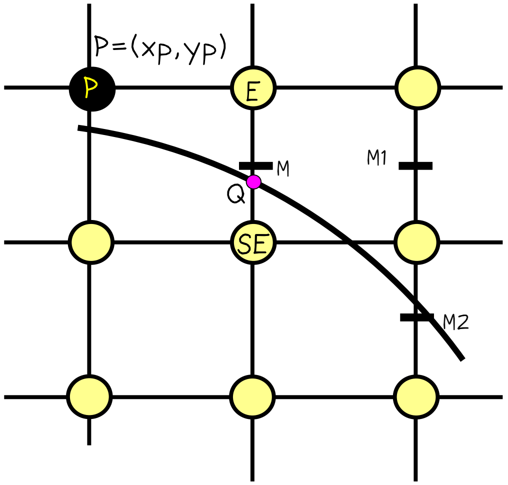
subtracting $d_{M1}-d = (x_p+2)^2 - (x_p+1)^2$

for move to E $\boxed{d_{M1} = \Delta_E = 2x_p + 3}$



Scan Converting Circles

Continued



if $d > 0$ (M outside) choose SE then

$d_{M2}=F(x_P+2,\ y_P-\frac{3}{2}) = (x_P+2)^2 + (y_P-\frac{3}{2})^2 - R^2$

subtracting $d_{M2}-d = (2x_P - 2y_P + 5)$

for move to SE $\boxed{d_{M2}-d = \Delta_{SE} = 2x_P - 2y_P + 5}$

Δ_E and Δ_{SE} vary at each step (constant for lines) but are linear functions depend on P.

Initial Condition

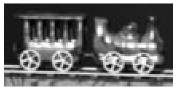
For integer radii in the second octant the circle starts at (0,R), the first midpoint will be at $(1,\ R-\frac{1}{2})$

$F(1,\ R-\frac{1}{2}) = 1 + (R^2 - R + \frac{1}{4}) - R^2 = 5/4 - R$

We can now make an algorithm similar to the Line algorithm.

```
void mid-pointCircle(int radius, int value)
/* assume centre of circle at origin */
{
    int x = 0;
    int y = radius;
    double d = 5.0 / 4.0 - radius;
    circlePoints(x,y,value); /* draws 8 points *//

    while (y>x) {
            if (d < 0)    /* select E */
            d += 2.0*x + 3.0;
            else {     /* select SE */
                    d += 2.0 * (x-y) + 5.0;
                    y--;
            }
            x++;
            circlePoints(x,y,value);
    } /* while*/
}
```



Eliminating Floating Point

Problem is that loop contains floating point operations. Initially d = 5/4 - radius

If we substitute h for d where h=d - 1/4

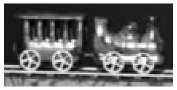
initially h = 1 - radius

comparison becomes if (h < -1/4)

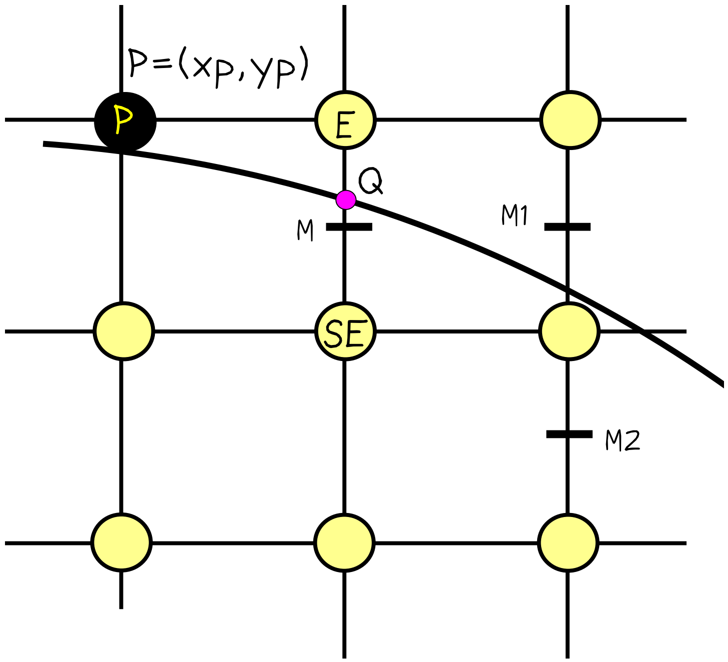
However since we are working in integer (comparison and incement in integer) we can still use if (h < 0)

```
void mid-pointCircle(int radius, int value)
/* assume centre of circle at origin */
{
    int x = 0;
    int y = radius;
    int d = 1 - radius;
    circlePoints(x,y,value); /* draws 8 points */

    while (y>x) {
            if (d < 0)   /* select E */
            d += 2.0*x + 3.0;
            else {    /* select SE */
                        d += 2.0 * (x-y) + 5.0;
                        y--;
            }
            x++;
            circlePoints(x,y,value);
    } /* while*/
}
```



Second order partial Differences



But any polynomial can be computed incrementally. Evaluate the function at adjacent points, calculate the difference (one degree lower) apply difference in each iteration.

For example suppose we choose E:
point of evaluation moves
from (x_p, y_p) to (x_p+1, y_p)

first order difference is:

$\Delta E_{old} = 2x_p + 3$ at (x_p, y_p)
$\Delta E_{new} = 2(x_p + 1) + 3$ at (x_p+1, y_p)

second order difference $\Delta E_{new} - \Delta E_{old} = 2$

Similarly for SE

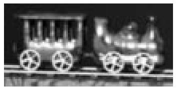
$\Delta SE_{old} = 2x_p - 2y_p + 5$
$\Delta SE_{new} = 2(x_p+1) - 2(y_p-1) + 5$

second order difference $\Delta SE_{new} - \Delta SE_{old} = 4$

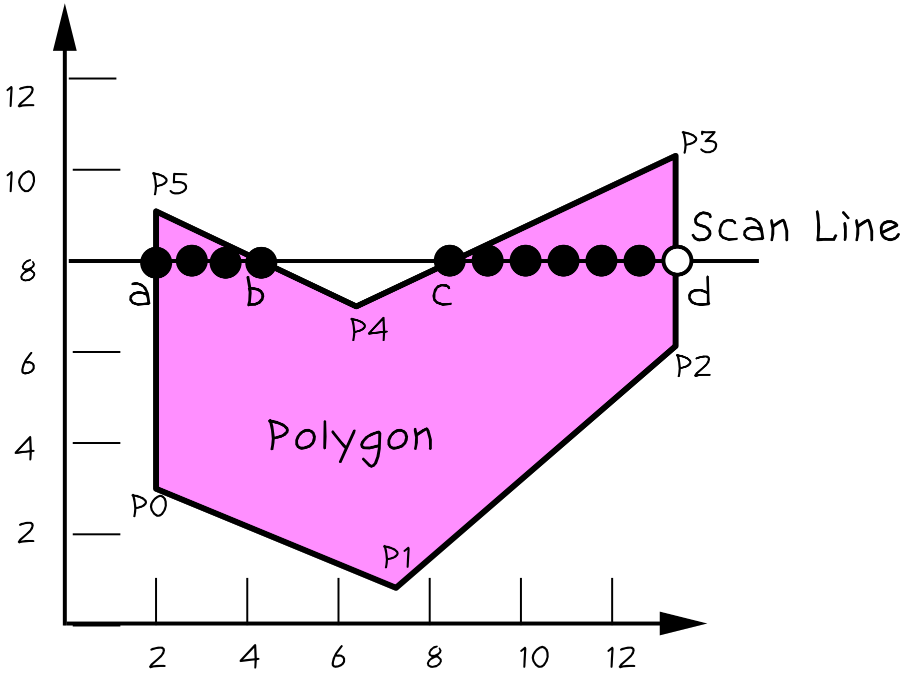
Circle Algorithm

Use second order differences to compute increments. assume centre of circle at origin

```
void mid-pointCircle(int radius, int value)
{  int x = 0;
   int y = radius;
   int d = 1 - radius;
   int deltaE = 3;
   int deltaSE = -2*radius + 5;
   circlePoints(x,y,value); /* draws 8 points */
   while (y>x) {
        if (d < 0) {  /* select E */
               d += deltaE;
               deltaE += 2;
               deltaSE+=2;
        } else {    /* select SE */
               d += deltaSE;
               deltaE += 2;
               deltaSE+= 4;
               y--;
        }
        x++;
        circlePoints(x,y,value);
   } /* while*/
}
```

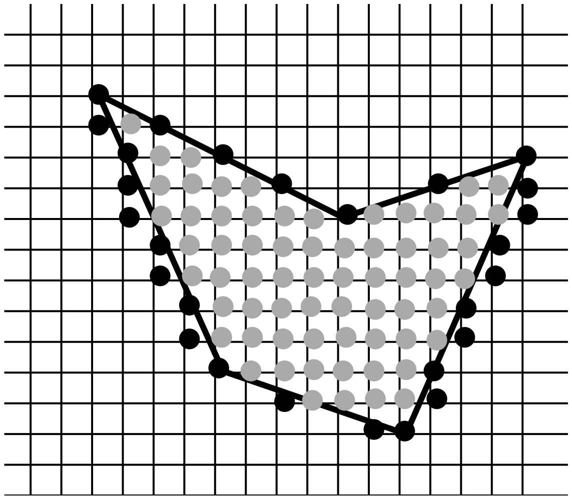


Scan Converting Polygons

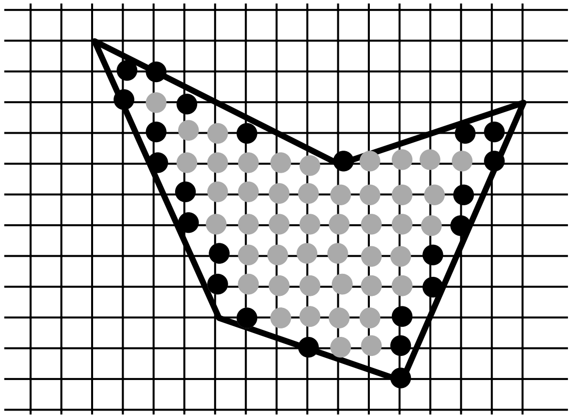


Polygons may be convex or concave self intersecting, have holes etc.

Can keep a table of spans. Find extrema from scan converting edges of polygon.



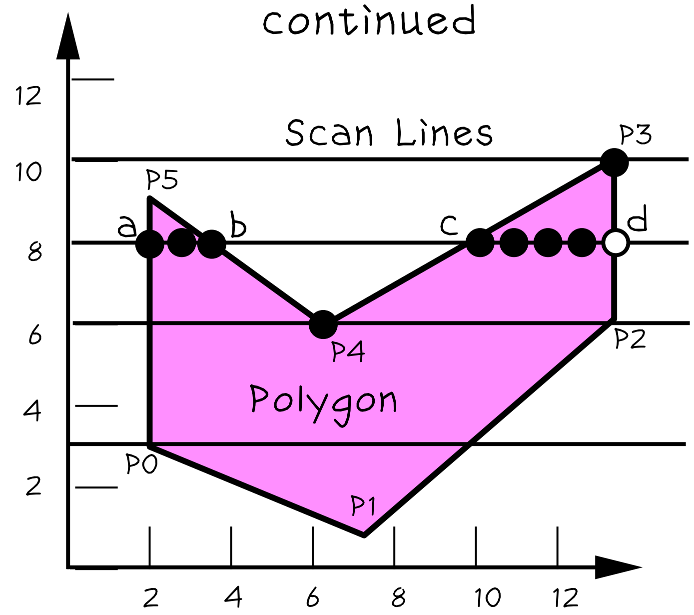
Some pixels lie outside the polygon.



Edge extrema only choosing inside edges. Care with abutting polygons.

Scan Converting Polygons

continued



Parity Rule

Parity=EVEN
At each edge invert parity.
Draw when parity is odd.

This scheme fails for scanline y=10 at p3

To calculate span extrema an incremental technique is used to avoid intersecting each edge with each scanline.

1. Find Intersection of scan line with all edges of polygon.

2. Sort intersections by increasing x-coordinate.

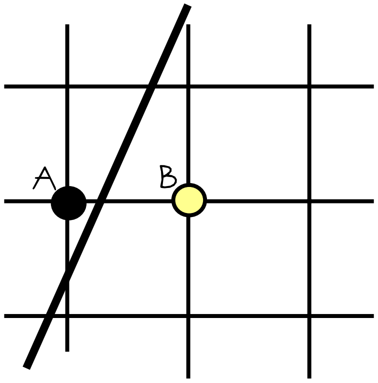
3. Fill spans using parity rule.

Shared vertices

e.g. p4 on scanline 6 is counted in parity calculation if it is ymin for that edge but not for ymax. parity is changed twice at p4 on scanline 6 but not changed at p3. P0 changes parity for edge p5p0 but not for edge p0 p1 so parity changes only once as scanline 3 enters the polygon.

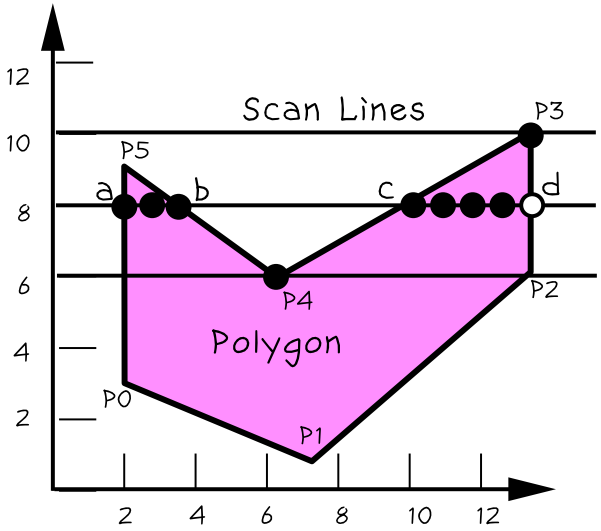
Scan Converting Polygons

continued



If intersection is fractional x value are pixels on either side interior?

If parity is odd (inside) round down (A will be inside B outside) if parity is even round up (B inside A outside).

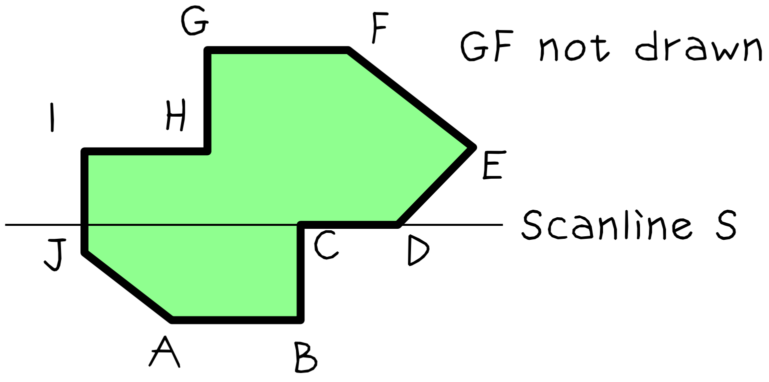


Intersection at integer pixel coordinates.

E.g. scanline 8.

Interior: if leftmost pixel in a span is integer

Exterior: if rightmost pixel in a span is integer



Horizontal Edges

The rule is that parity does not change for either vertex of horizontal edge. E.g. Vertex A is ymin wrt JA and AB does not change parity. B is ymin for BC but since AB does not contribute B changes parity to even for BC. Scanline S passes through C. CD does not affect it and C is ymax for BC and makes no change. GF not drawn. Aliasing problems - thin sliver polygons.

Edge Coherance

Problem: Find Intersection of scan lines with all edges of polygon.

If slope is m then successive scan line intersections can be found from:

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

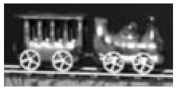
Where i is the scan line count. (Can avoid fp arithmetic by storing numerator and comparing to denominator, increment x when it overflows.)

Scan-line Algorithm

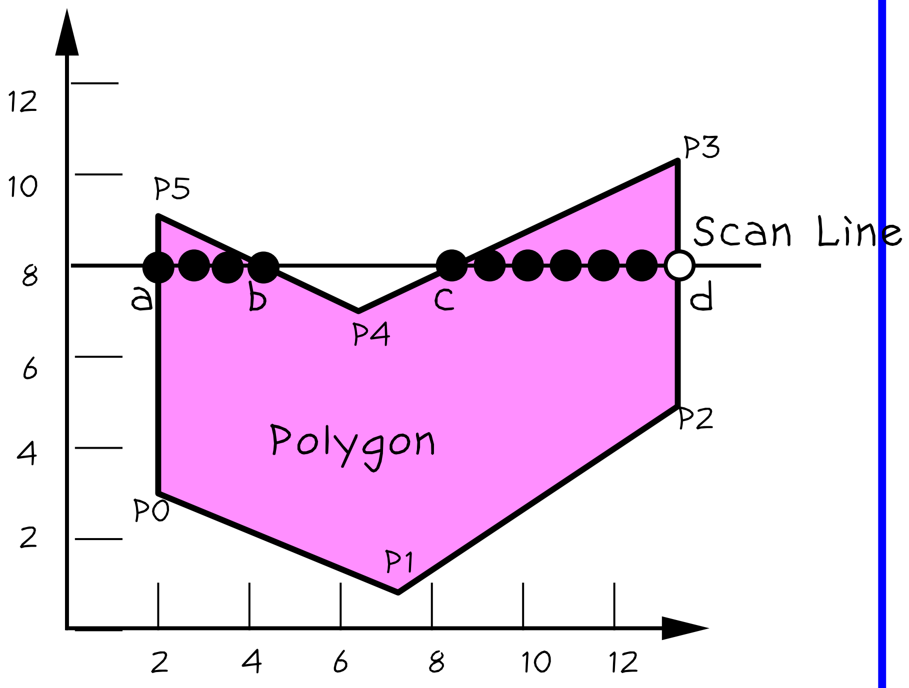
AET = Active Edge Table

store all edges intersected by a scan line sorted by x intersection. As each new scan line is encountered update AET. y y+1

1. Remove edges not intersected by y+1 (ymax=y)
2. Add edges intersected by new scan line ymin=y+1
3. Calculate new x intersections.



Scan Line Algorithm



Edge Table contains all scanlines

ymax	xmin	1/m	next