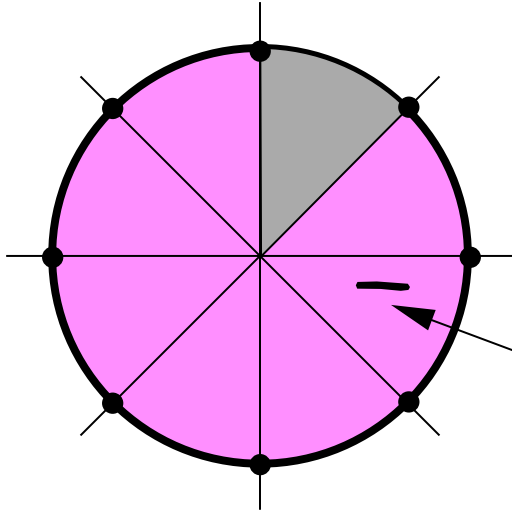


Scan Converting Circles



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:

$$\text{Let } F(x,y) = x^2 + y^2 - R^2$$

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

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

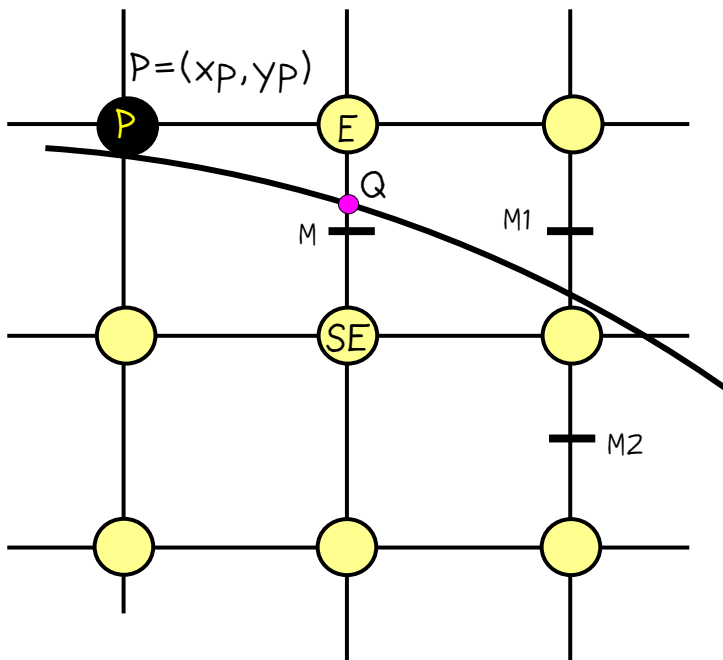
if $d < 0$ (M inside) choose E then

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

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

for move to E

$$d_{M1} = \Delta E = 2x_p + 3$$



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

$$d_{M2} - d = \Delta_{SE} = 2x_p - 2y_p + 5$$

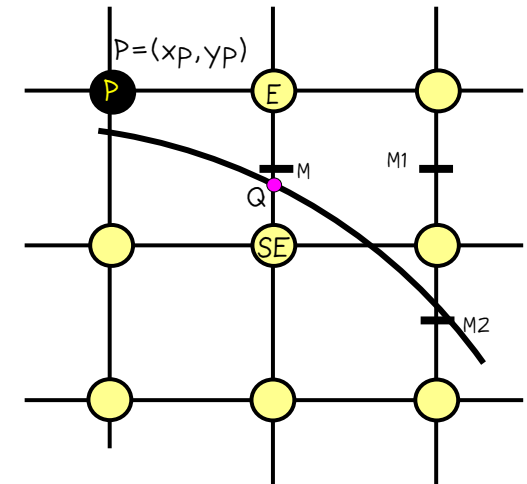
Δ_E and Δ_{SE} vary at each step (constant for lines) but are linear functions depend on P.

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 $d = 5/4 - \text{radius}$

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

initially $h = 1 - \text{radius}$

comparison becomes if ($h < -1/4$)

However since we are working in integer (comparison and increment 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:

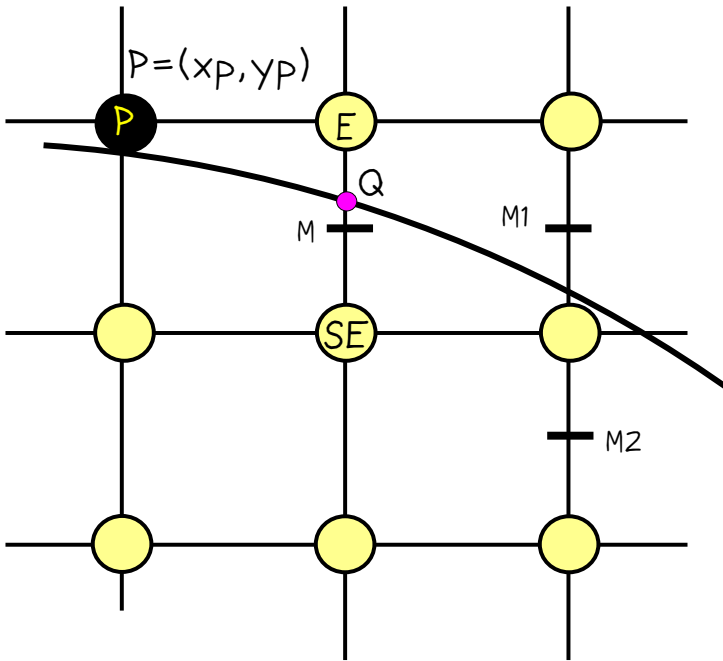
$$\begin{aligned}\Delta_{Eold} &= 2x_p + 3 && \text{at } (x_p, y_p) \\ \Delta_{Enew} &= 2(x_p + 1) + 3 && \text{at } (x_p+1, y_p)\end{aligned}$$

$$\text{second order difference } \Delta_{Enew} - \Delta_{Eold} = 2$$

Similarly for SE

$$\begin{aligned}\Delta_{SEold} &= 2x_p - 2y_p + 5 \\ \Delta_{SEnew} &= 2(x_p+1) - 2(y_p-1) + 5\end{aligned}$$

$$\text{second order difference } \Delta_{SEnew} - \Delta_{SEold} = 4$$



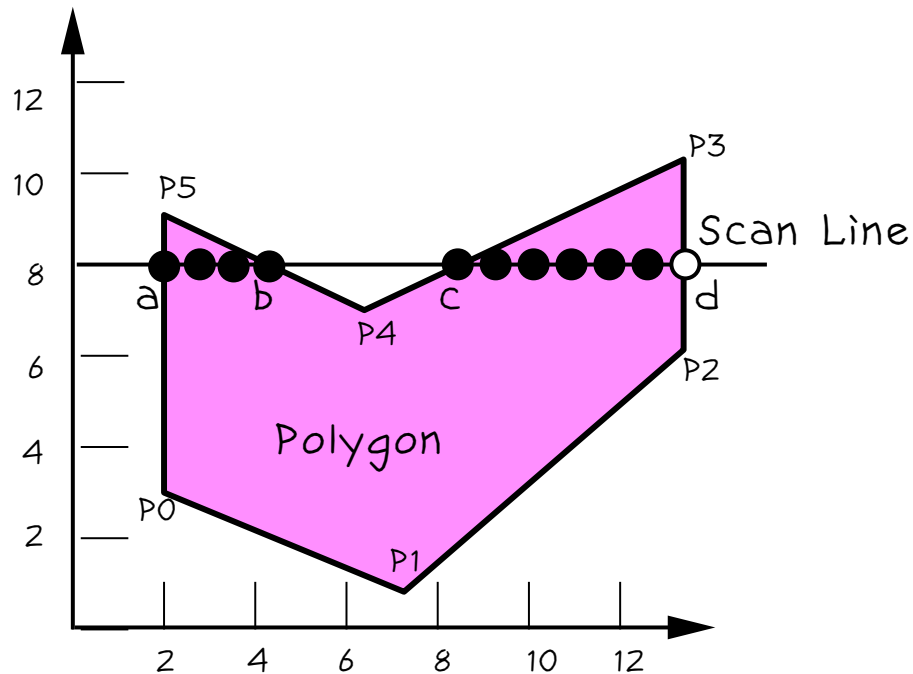
Circle Algorithm

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*/
}
```

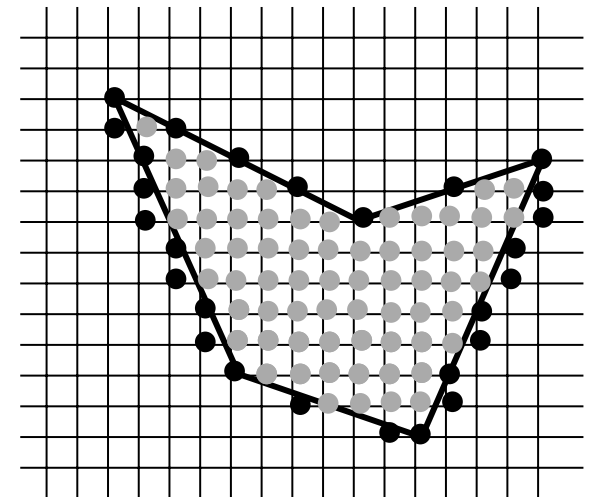


Scan Converting Polygons

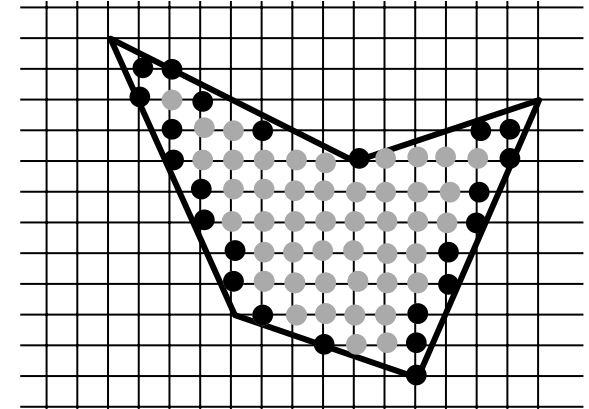


Polygons may be convex or concave self intersecting, have holes etc.

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



Some pixels lie outside the polygon.

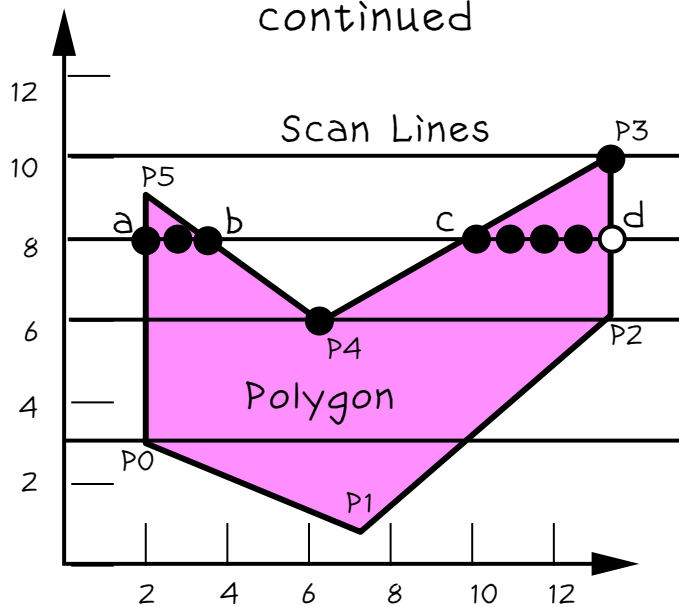


Edge extrema only choosing inside edges. Care with abutting polygons.



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 p_3

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 y_{min} for that edge but not for y_{max}. 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.

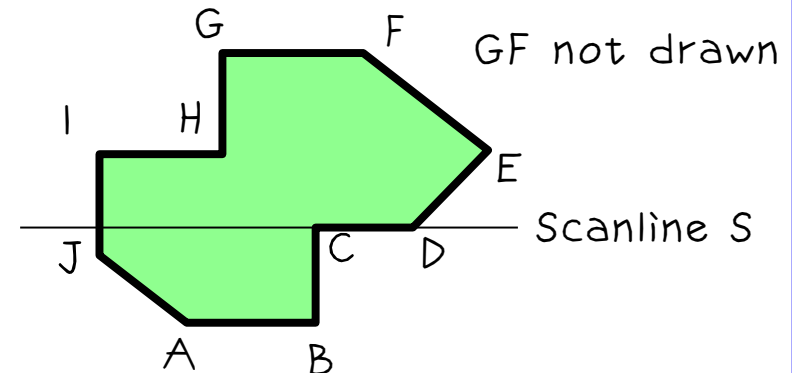
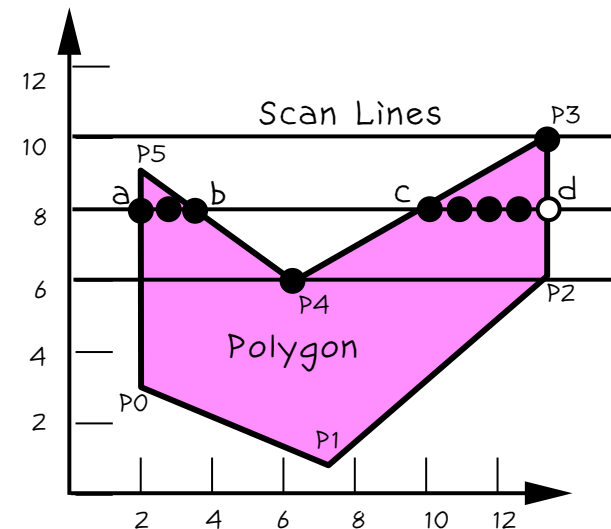
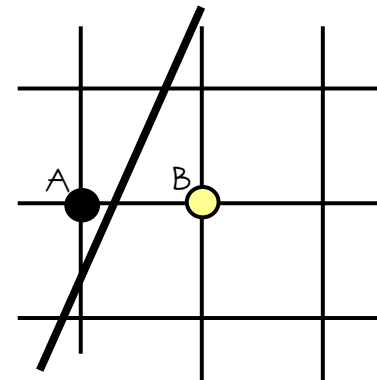
Interior: if leftmost pixel in a span is integer

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 y_{min} wrt JA and AB does not change parity. B is y_{min} 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 y_{max} for BC and makes no change. GF not drawn.

Aliasing problems - thin sliver polygons.



Edge Coherence

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. $y \rightarrow y+1$

1. Remove edges not intersected by $y+1$ ($y_{\max}=y$)
2. Add edges intersected by new scan line $y_{\min}=y+1$
3. Calculate new x intersections.



Scan Line Algorithm

Edge Table contains all scanlines

y _{max}	x _{min}	1/m	next
------------------	------------------	-----	------

