Chapter 3
Scan Conversion
Lecture-02

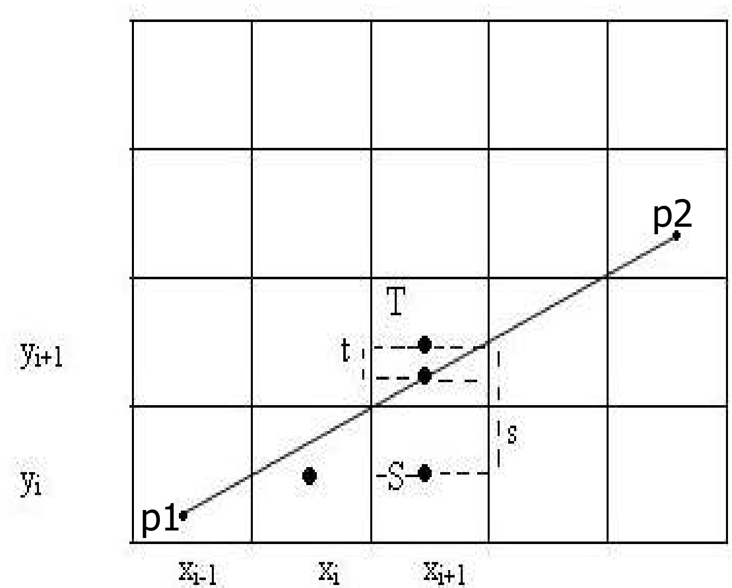
Prepared By-
Md Imtiaz Ahmed

Bresenham's Line Algorithm

- Bresenham's line algorithm
  - is a highly efficient incremental method for scan-converting lines.
  - It produces mathematically accurate results using only integer addition, subtraction and multiplication by 2, which can be accomplished by a simple arithmetic shift operation.

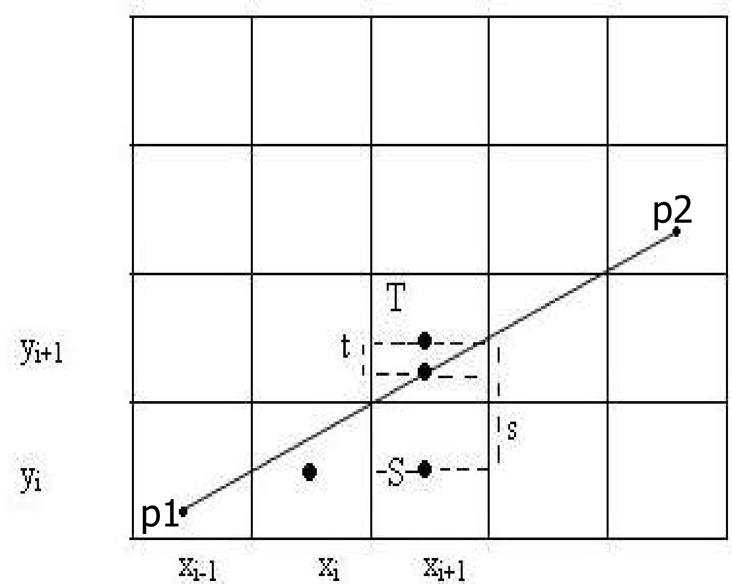
Bresenham’s Line Algorithm

- Scan convert the line in the Figure,
- $0<m<1$.
- Start with pixel $P^1(x^1,y^1)$.
- Choose either the pixel on the right or the pixel right and up.
- The coordinates of the last chosen pixel upon entering step i are (x^i,y^i).
- Choose the next between the bottom pixel S and the top pixel T.
- If the chosen pixel is the top pixel T ($d^i \geq 0$) then $x^{i+1} = x^i+1$ and $y^{i+1} = y^i + 1$ and so
- $d^{i+1} = d^i + 2 (\Delta y - \Delta x)$



Bresenham's Line Algorithm

- If the chosen pixel is pixel S ($d^i < 0$) then $x^{i+1} = x^i + 1$ and $y^{i+1} = y^i$ and so
- $d^{i+1} = d^i + 2\Delta y$
- where $d^i = 2\Delta y * x^i - 2\Delta x * y^i + C$
- and $C = 2\Delta y + \Delta x\,(2b - 1)$
- We use here a decision variable d^i. For the value of each d^i we calculate the corresponding value of d^{i+1}.



Bresenham's Line Algorithm

- Void Bresenham( )
- {
- Line 1: dx=x2-x1;
- Line 2: dy=y2-y1;
- Line 3: dT=2*(dy-dx);
- Line 4: dS=2*dy;
- Line 5: d=(2*dy)-dx;
- Line 6: putpixel(x1,y1);
- Line 7: while(x1<x2)
- Line 8: {
- Line 9: x1++;
- Line 10: if(d<0)
- Line 11: {
- Line 12: d=d+dS;
- Line 13: putpixel(x1,y1);
- Line 14: }
- Line 15: else
- Line 16: {
- Line 17: **y1++;**
- Line 18: d=d+dT;
- Line 19: putpixel(x1,y1);
- Line 20: }
- Line 21: }
- Line 22: putpixel(x2,y2);
- }

Scan-Converting a Circle

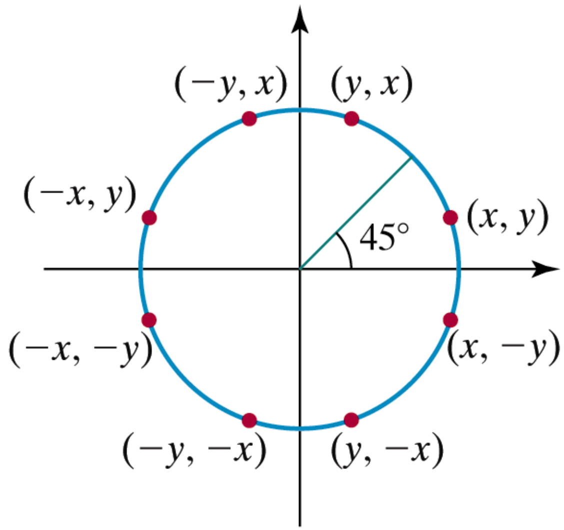


Figure 3-18

Symmetry of a circle. Calculation of a circle point (x, y)in one octant yields the circle points shown for the other seven octants.

Computer Graphics with Open GL, Third Edition, by Donald Hearn and M.Pauline Baker.
ISBN 0-13-0-15390-7

Midpoint Circle Algorithm

- We will first calculate pixel positions for a circle centered around the origin (0,0). Then, each calculated position (x,y) is moved to its proper screen position by adding xc to x and yc to y

- Note that along the circle section from x=0 to x=y in the first octant, the slope of the curve varies from 0 to -1

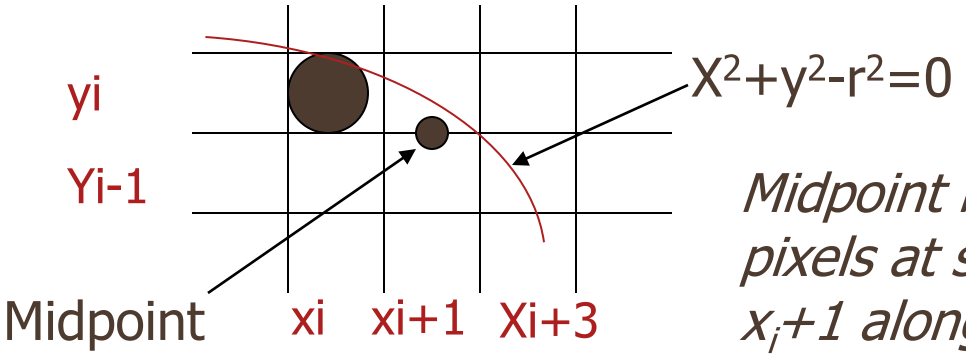
- Circle function around the origin is given by

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

- Any point (x,y) on the boundary of the circle satisfies the equation and circle function is zero

Midpoint Circle Algorithm

- For a point in the interior of the circle, the circle function is negative and for a point outside the circle, the function is positive
- Thus,
  - $f_{circle}(x,y) < 0$ if (x,y) is inside the circle boundary
  - $f_{circle}(x,y) = 0$ if (x,y) is on the circle boundary
  - $f_{circle}(x,y) > 0$ if (x,y) is outside the circle boundary



Midpoint between candidate pixels at sampling position x_i+1 along a circular path

Midpoint Circle Algorithm

- Assuming we have just plotted the pixel at (x_i, y_i) , we next need to determine whether the pixel at position $(x_i + 1, y_i - 1)$ is closer to the circle
- Our decision parameter is the circle function evaluated at the midpoint between these two pixels

$$p_i = f_{circle}(x_i + 1, y_i - 1/2) = (x_i + 1)^2 + (y_i - 1/2)^2 - r^2$$

If $p_i < 0$, this midpoint is inside the circle and the pixel on the scan line y_i is closer to the circle boundary. Otherwise, the

mid position is outside or on the circle boundary, and we select the pixel on the scan line $y_i - 1$

Midpoint Circle Algorithm

- Successive decision parameters are obtained using incremental calculations

$$P_{i+1} = f_{circle}(x_{i+1}+1,\ y_{i+1}-1/2)$$
$$= [(x_{i+1})+1]^2 + (y_{i+1} -1/2)^2 - r^2$$

OR

$$P_{i+1} = P_i+2(x_i+1) + (y_{i+1}{}^2 - y_i{}^2) - (y_i+1- y_i)+1$$

iWhere y_{i+1} is either y_i or y_{i-1} depending on the sign of p_i

- Increments for obtaining P_{i+1}:

$2x_{i+1}+1$ *if* p_i *is negative*

$2x_{i+1}+1-2y_{i+1}$ *otherwise*

Midpoint circle algorithm

- Note that following can also be done incrementally:

  $2x_{i+1} = 2x_i + 2$

  $2\ y_{i+1} = 2y_i - 2$

- At the start position $(0,r)$, these two terms have the values 2 and $2r-2$ respectively
- Initial decision parameter is obtained by evaluating the circle function at the start position $(x0,y0) = (0,r)$

  $p_0 = f_{circle}(1, r-1/2) = 1+ (r-1/2)^2-r^2$

OR

  $P_0 = 5/4 -r$

- If radius r is specified as an integer, we can round p_0 to

  $p_0 = 1-r$

Midpoint circle algorithm

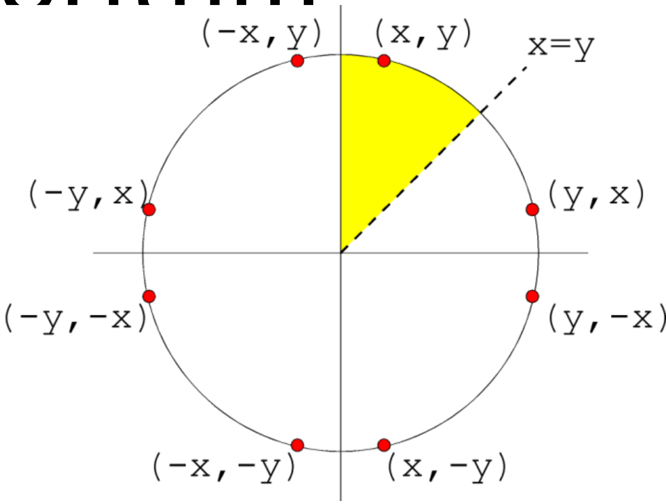
```
Int x=0,y=r,p=1-r;
While(x<=y)
{
setPixel(x,y);
If(p<0)
p=p+2x+3;
Else
{
P=p+2(x-y)+5;
y--;
}
x++;
}
```

Midpoint Circle Algorithm

- Implicit of equation of circle is:

  $x^2 + y^2 - R^2 = 0$

- Eight way symmetry $\Rightarrow$ require to calculate one octant
- Define decision variable ***d*** as:



$$d = F(M) = F(x_p + 1, y_p - \tfrac{1}{2})$$

$$= (x_p + 1)^2 + (y_p - \tfrac{1}{2})^2 - R^2$$

$d < 0$

$\Rightarrow M$ *is inside Circle*

$\Rightarrow$ *Choose E*

$d > 0$

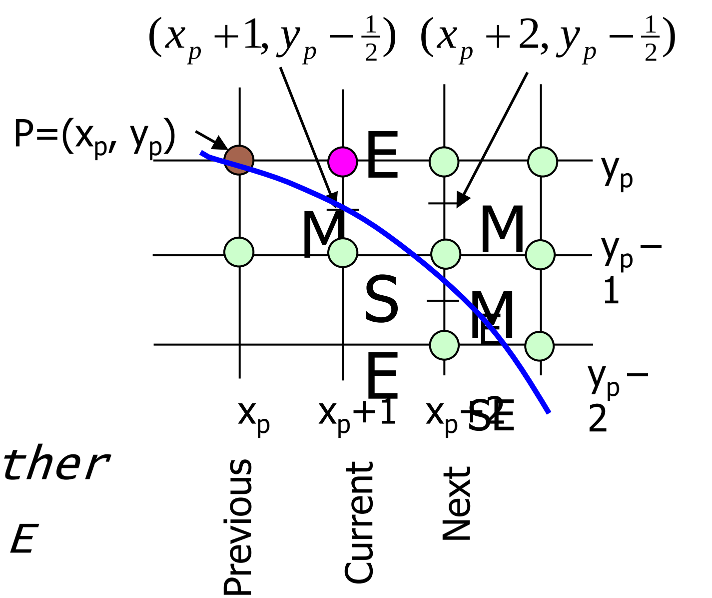
$\Rightarrow M$ *is outside Circle*

$\Rightarrow$ *Choose SE*

$d = 0$

$\Rightarrow$ *Choose either*

$\Rightarrow$ *we choose E*



Midpoint Circle Algorithm

- If d <= 0 then midpoint m is inside circle
  - we choose E
  - Increment x
  - y remains unchanged

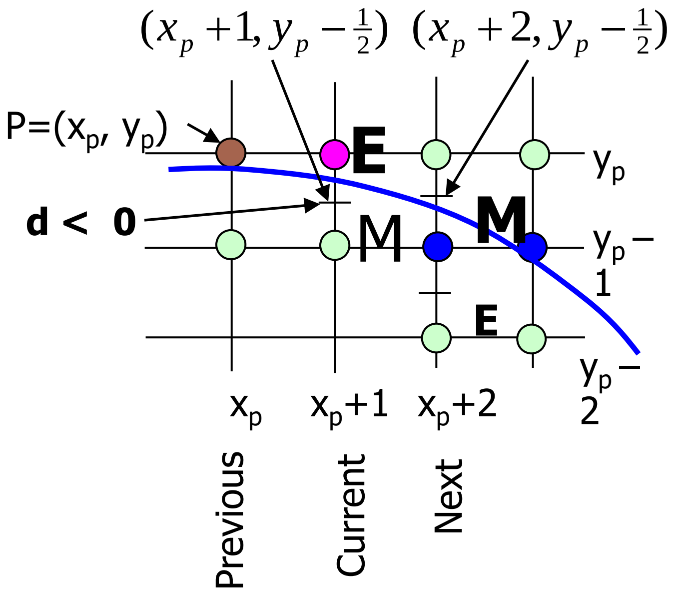
$$d = \left(x_p + 1\right)^2 + \left(y_p - \tfrac{1}{2}\right)^2 - R^2$$

$$d_{new} = F(x_p + 2, y_p - \tfrac{1}{2})$$

$$= \left(x_p + 2\right)^2 + \left(y_p - \tfrac{1}{2}\right)^2 - R^2$$

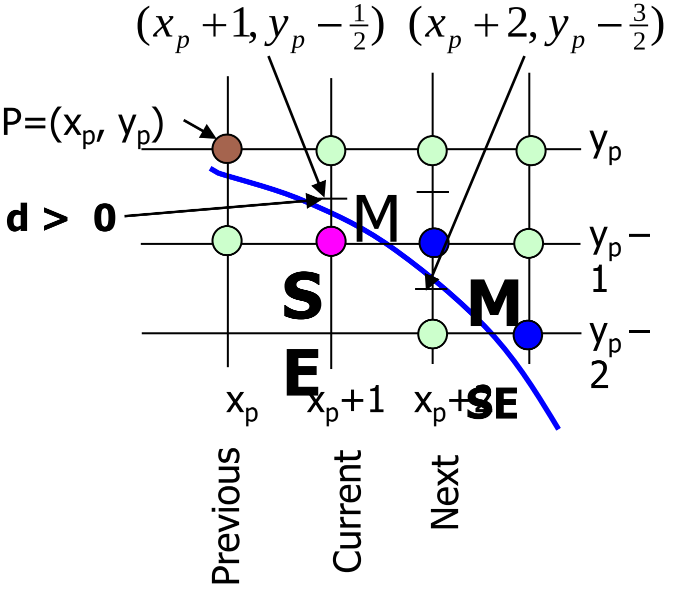
$$d_{new} - d = \underbrace{2x_p + 3}_{\Delta E}$$

$$d_{new} = d + \Delta E$$



Midpoint Circle Algorithm

- If d > 0 then midpoint m is outside circle
  - we choose E
  - Increment x
  - Decrement y



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

$$d_{new} = F(x_p+2, y_p - \tfrac{3}{2})$$

$$= (x_p+2)^2 + (y_p - \tfrac{3}{2})^2 - R^2$$

$$d_{new} - d = \underbrace{2x_p - 2y_p + 5}_{\Delta SE}$$

$$d_{new} = d + \Delta SE$$

Midpoint Circle Algorithm

Initial condition

- Starting pixel (0, R)
- Next Midpoint lies at (1, R – ½)
- $d_0 = F(1, R - ½) = 1 + (R^2 - R + ¼) - R^2 = {}^5/_4 - R$
- To remove the fractional value ${}^5/_4$:
  - Consider a new decision variable h as, **h = d – ¼**
  - Substituting **d** for **h + ¼**,
    - $\mathbf{d_0 = {}^5/_4 - R \Rightarrow h = 1 - R}$
    - $\mathbf{d < 0 \quad \Rightarrow h < -¼ \Rightarrow h < 0}$
    - Since h starts out with an integer value and is incremented by integer value (ΔE or ΔSE), e can change the comparison to just h < 0

Midpoint Circle Algorithm

```
void MidpointCircle(int radius, int value) {
        int x = 0;
        int y = radius ;
        int d = 1 – radius ;
        CirclePoints(x, y, value);
        while (y > x) {
                if (d < 0) {                /* Select E */
                        d += 2 * x + 3;
                } else {                                    /* Select SE */
                        d += 2 * ( x –  y ) + 5;
                        y – –;
                }
                x++;
                CirclePoints(x, y, value);
        }
}
```