Circle Drawing Algorithms

Basics

Properties of a circle:

- A circle is defined as a set of points that are all the given distance (x_c, y_c). This distance relationship is expressed by the pythagorean theorem in Cartesian coordinates as

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

- We could use this equation to calculate the points on the circle circumference by stepping along x-axis in unit steps from $x_c - r$ to $x_c + r$ and calculate the corresponding y values at each position as

$$y = y_c \ +(-)\ (r^2 - (xc - x)^2)^{1/2}$$

- This is not the best method:
  - *Considerable amount of computation*
  - *Spacing between plotted pixels is not uniform*

Polar co-ordinates for a circle

- We could use polar coordinates r and θ,

  $x = x_c + r\cos\theta \qquad y = y_c + r\sin\theta$
- A fixed angular step size can be used to plot equally spaced points along the circumference
- A step size of 1/r can be used to set pixel positions to approximately 1 unit apart for a continuous boundary
- But, note that circle sections in adjacent octants within one quadrant are symmetric with respect to the 45 deg line dividing the to octants
- Thus we can generate all pixel positions around a circle by calculating just the points within the sector from x=0 to x=y
- This method is still computationally expensive

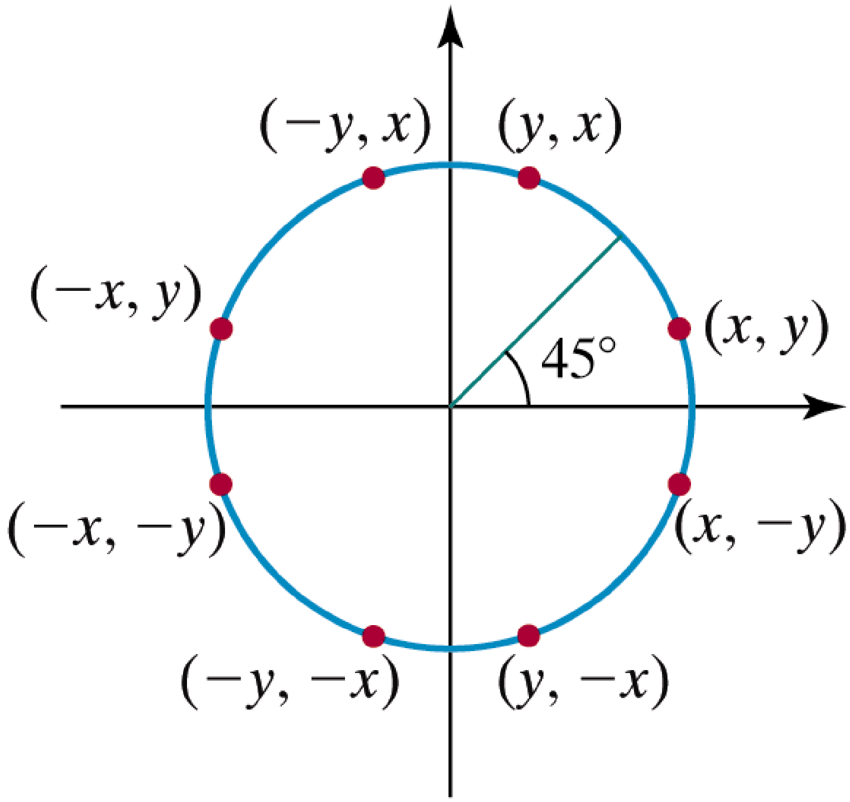


Figure 3-18

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.

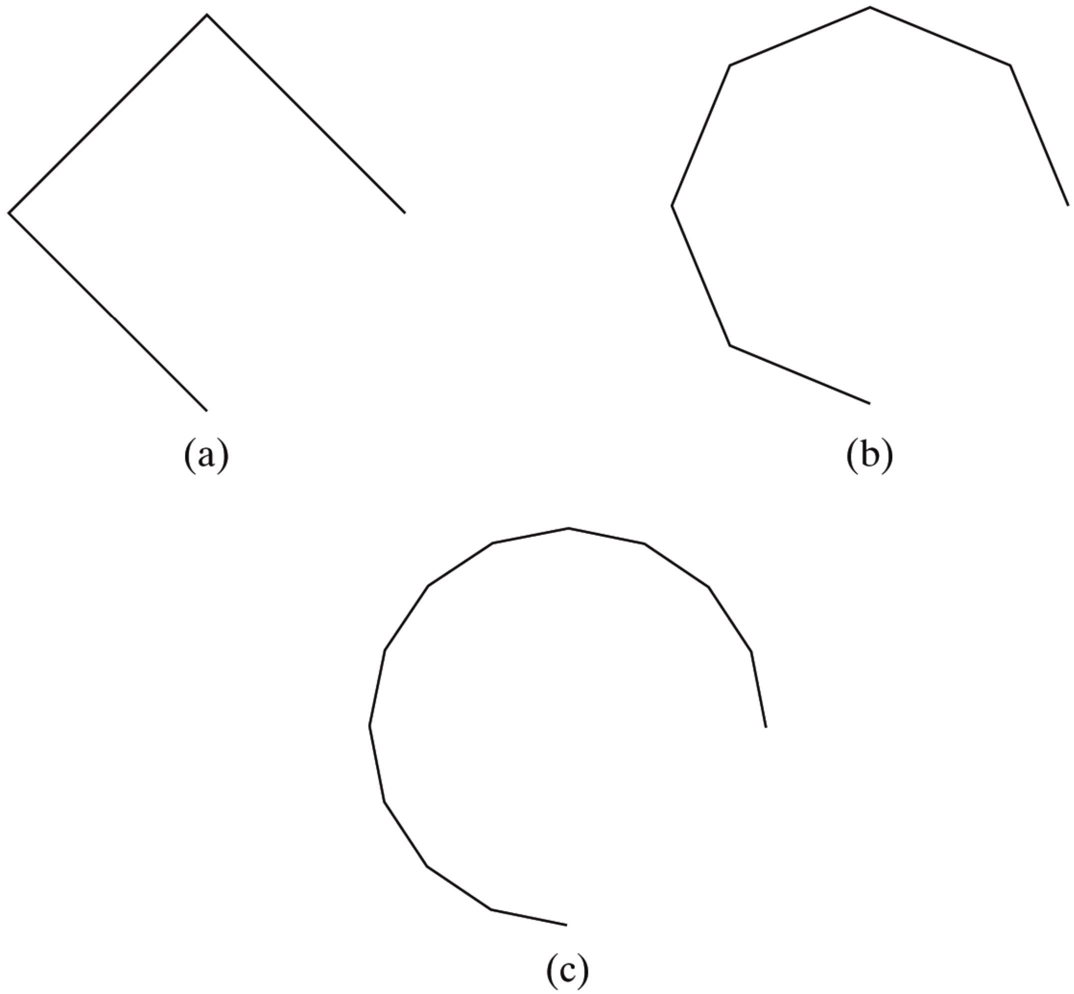


Figure 3-15

A circular arc approximated with (a) three straight-line segments, (b) six line segments, and (c) twelve line segments.

Bresenham to Midpoint

- Bresenham requires explicit equation
  - Not always convenient (many equations are implicit)
  - Based on implicit equations: Midpoint Algorithm (circle, ellipse, etc.)
  - Implicit equations have the form $F(x,y)=0$.

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

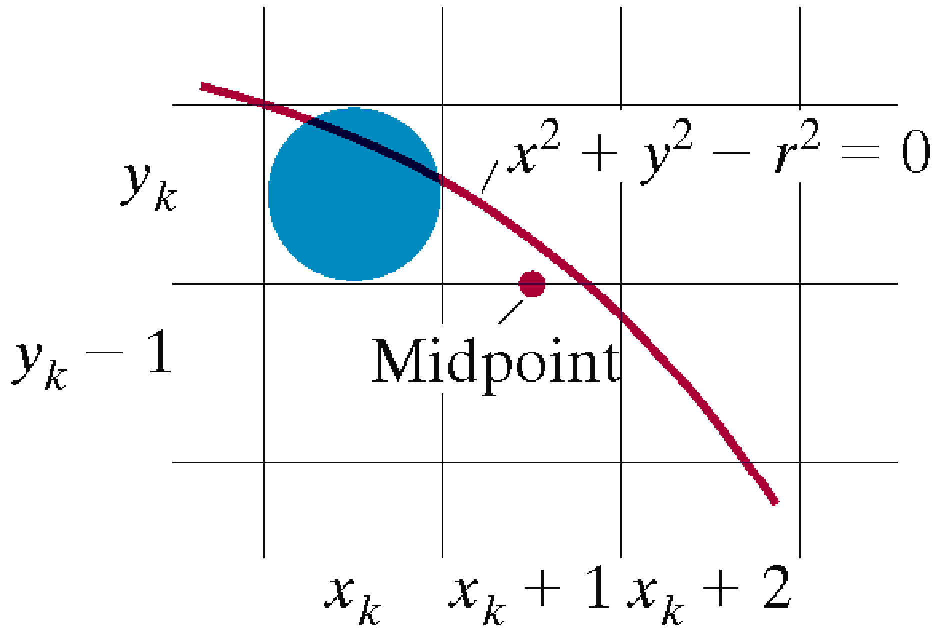


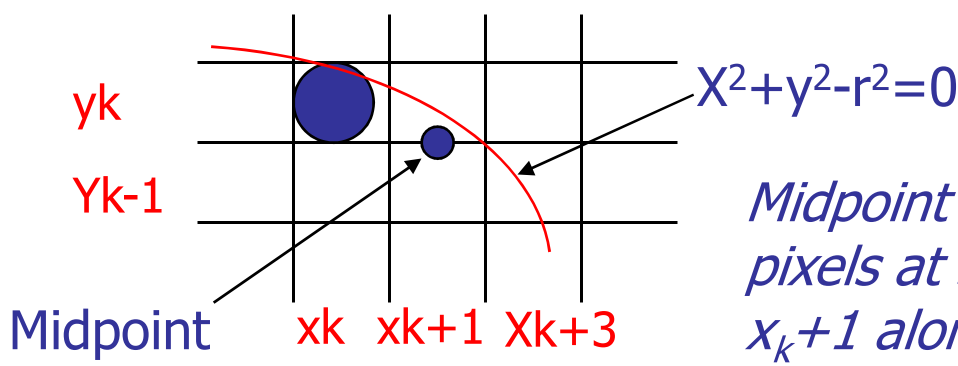
Figure 3-19

Midpoint between candidate pixels at sampling position $x_k + 1$ along a circular path.

Computer Graphics with Open GL, Third Edition, by Donald Hearn and M.Pauline Baker.

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_k+1 along a circular path

Midpoint Circle Algorithm

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

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

If $p_k < 0$, this midpoint is inside the circle and the pixel on the scan line y_k 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 $y_k - 1$

Midpoint Circle Algorithm

- Successive decision parameters are obtained using incremental calculations

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

OR

$$P_{k+1} = P_k+2(x_K+1) + (y_{K+1}^2 - y_k^2) - (y_k+1- y_k)+1$$

Where y_{k+1} is either y_k or y_{k-1}, depending on the sign of p_k

- Increments for obtaining P_{k+1}:

  $2x_{k+1}+1$ *if* p_k *is negative*

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

Midpoint circle algorithm

- Note that following can also be done incrementally:

  $2x_{k+1} = 2x_k + 2$

  $2\,y_{k+1} = 2y_k - 2$

- 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

  $$p_0 = 1-r$$

The actual algorithm

1: Input radius r and circle center (x_c,y_c) and obtain the first point on the circumference of the circle centered on the origin as

$$(x_0,y_0) = (0,r)$$

2: Calculate the initial value of the decision parameter as

$$P_0 = 5/4 - r$$

3: At each x_k position starting at $k = 0$, perform the following test:

If $p_k < 0$, the next point along the circle centered on (0,0) is (x_{k+1}, y_k) and

$$p_{k+1} = p_k + 2x_{k+1} + 1$$

The algorithm

Otherwise the next point along the circle is (x_{k+1}, y_{k-1}) and

$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$$

Where $2x_{k+1} = 2x_{k+2}$ and $2y_{k+1} = 2y_k - 2$

4: Determine symmetry points in the other seven octants

5: Move each calculated pixel position (x,y) onto the circular path centered on (x,yc) and plot the coordinate values

$$x = x + x_c \quad , \quad y = y + y_c$$

6: Repeat steps 3 through 5 until x >= y

Pseudocode

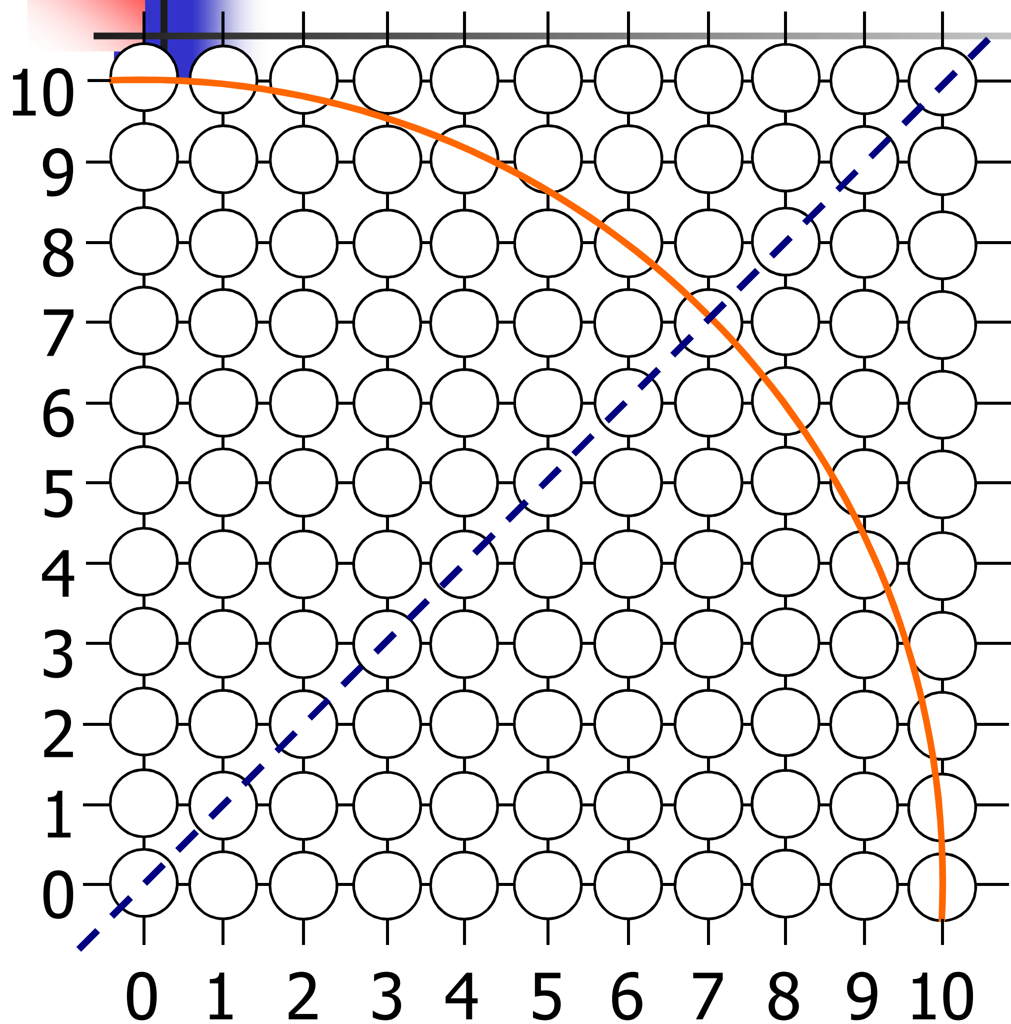
```
void circleMP(int xc,int yc,int r)
{
   int x = 0 , y = r , p = 1 – r;

   plotPoints(xc,yc,x,y);
   while (x < y){
         x = x +1;
          if (p < 0) then  p + = 2 * x + 1;
          else {
                    y - -;
                    p + = 2 *( x – y) + 1;
                   } // else complete
   plotPoints(xc,yc,x,y);
                  } // while complete

} // function def. complete
```

```
void plotPoints(int xc,int,yc,int x,int y)
{
  putpixel( xc + x, yc + y );
  putpixel( xc - x, yc + y );
  putpixel( xc + x, yc - y );
  putpixel( xc - x, yc - y );
  putpixel( xc + y, yc + x );
  putpixel( xc - y, yc + x );
  putpixel( xc + y, yc - x );
  putpixel( xc - y, yc - x );
}
```

Mid-Point Circle Algorithm Example

- To see the mid-point circle algorithm in action lets use it to draw a circle centred at (0,0) with radius 10

Mid-Point Circle Algorithm Example (cont…)



k	p_k	(x_{k+1},y_{k+1})	$2x_{k+1}$	$2y_{k+1}$
0				
1				
2				
3				
4				
5				
6				