Computer Graphics

Lecture 4

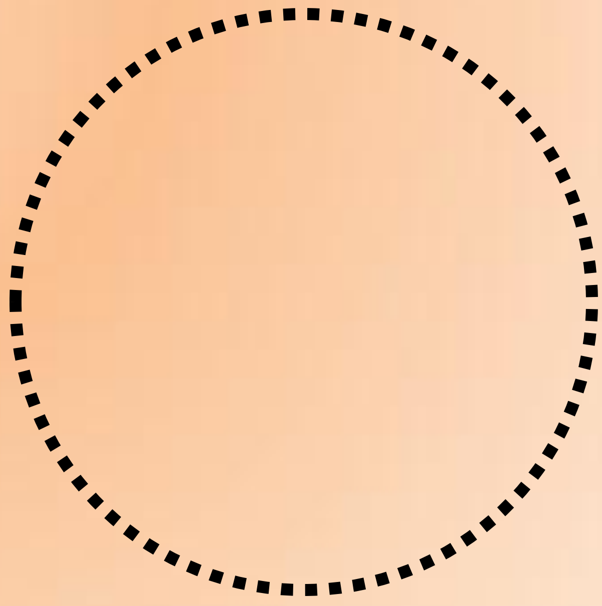
By

Kareem Ahmed

Circle Drawing Algorithms

- A circle is defined as a set of points that are all have the same distance from a given center $(X_c\,,\ Y_c)$.
- This distance relationship is expressed by the pythagorean theorem in Cartesian coordinates as

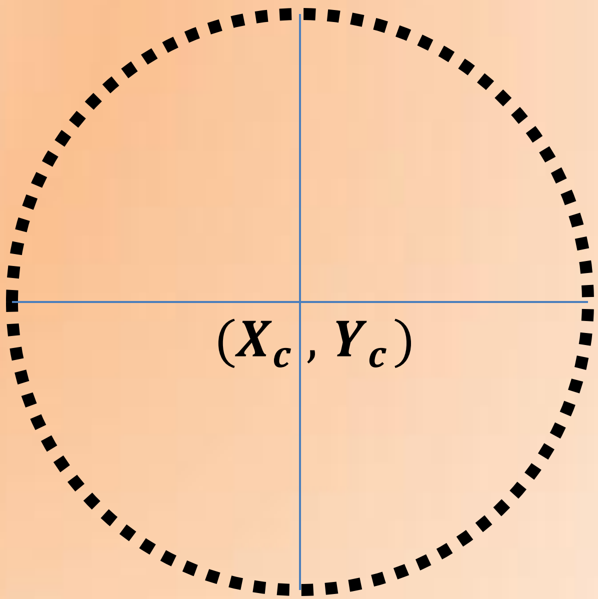
$$(x - x_c)^2 + (y - y_c)^2 = r^2$$



Circle Drawing Algorithms

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 from

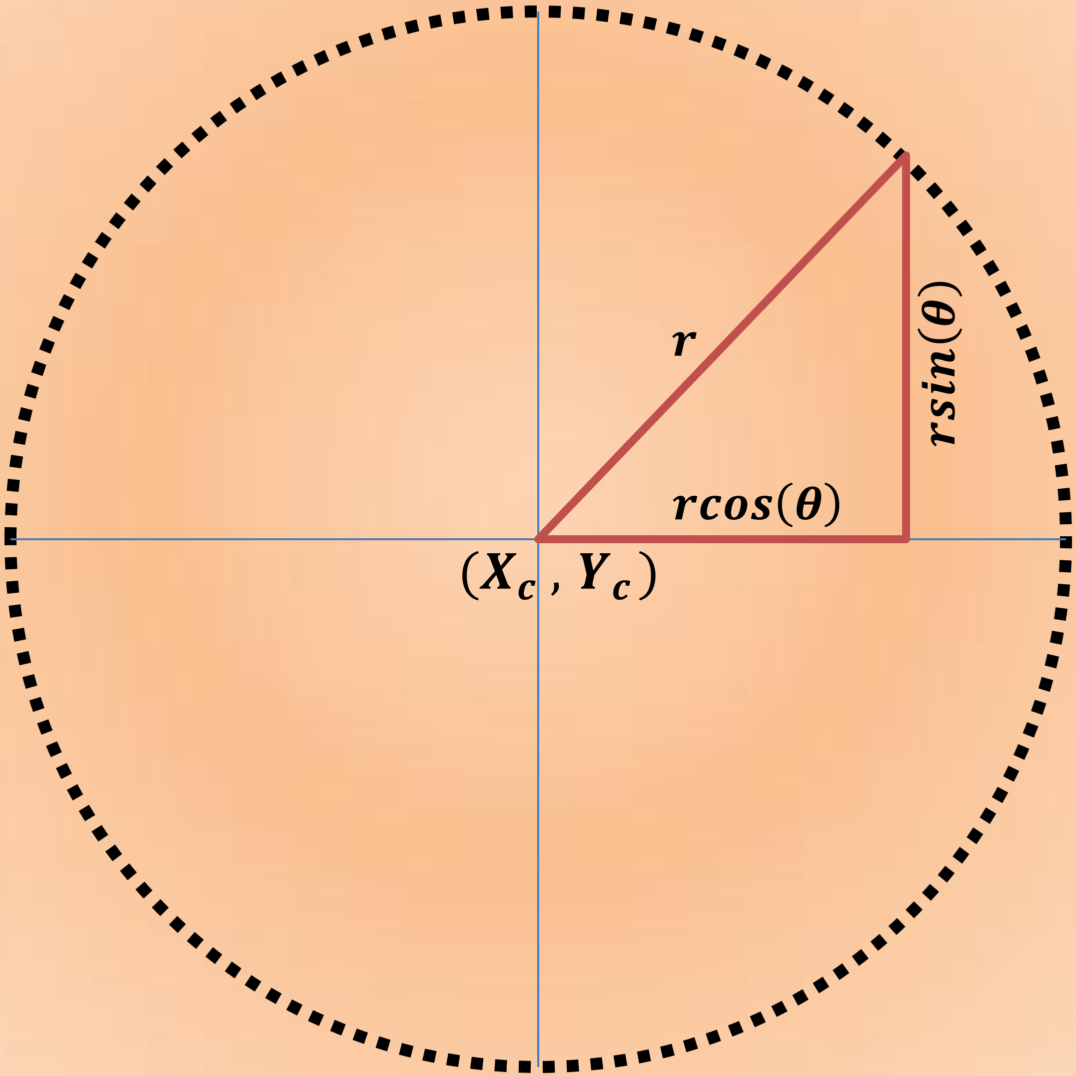
$$y = y_c \pm \sqrt{r^2 - (x_c - x)^2}$$



Circle Drawing Algorithms

- Drawbacks:
  - *Considerable amount of computation*
  - *Spacing between plotted pixels is not uniform*

Polar co-ordinates for a circle



Polar co-ordinates for a circle

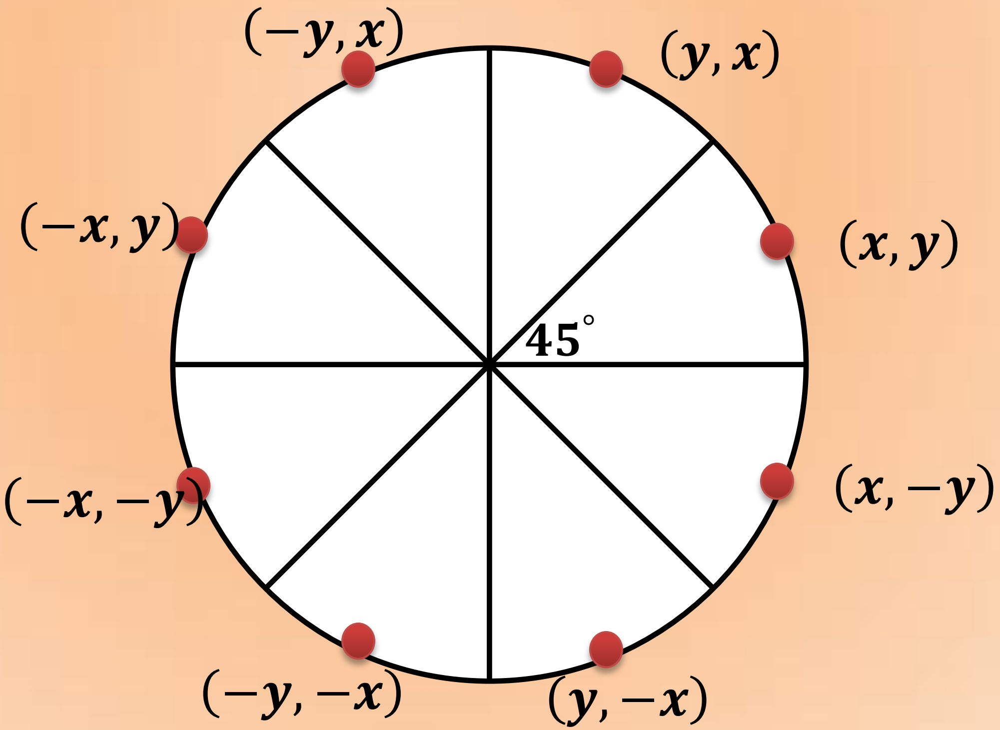
- We could use polar coordinates r and θ,

$$x = x_c + rcos(\theta)$$
$$y = y_c + sin(\theta)$$

- 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

Polar co-ordinates for a circle

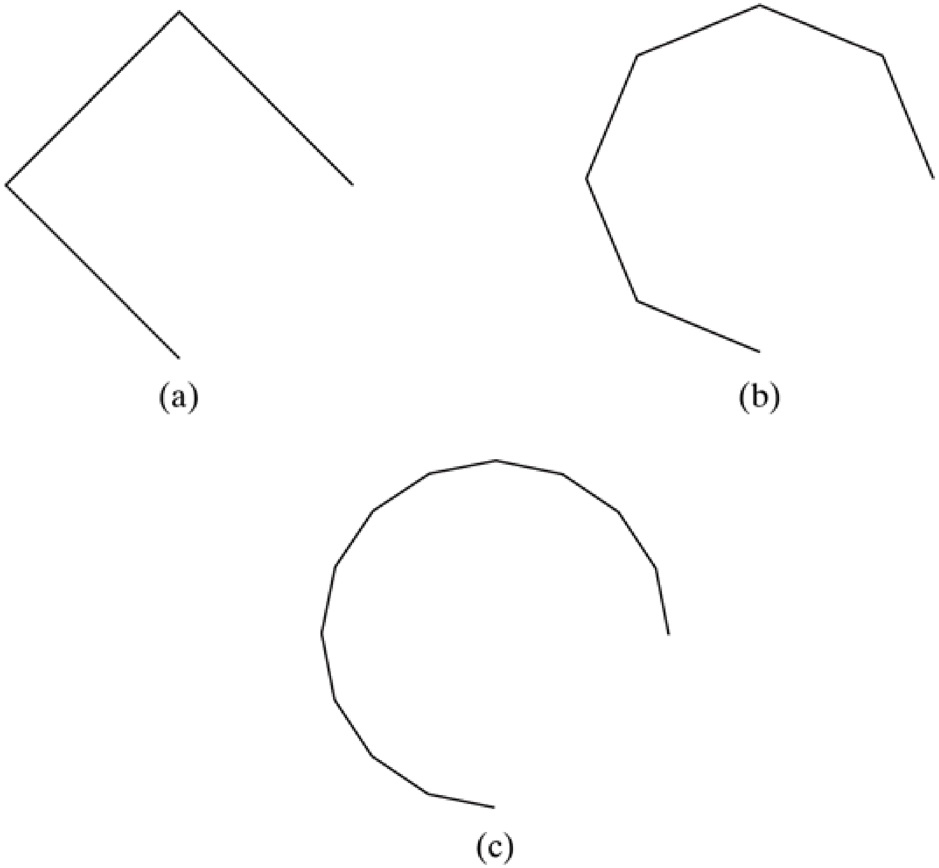
- But, note that circle sections in adjacent octants within one quadrant are symmetric with respect to the 45° line dividing the two octants.



Polar co-ordinates for a circle

- Symmetry of a circle. Calculation of a circle point (x, y) in one octant yields the circle points shown for the other seven 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
- But This method is still computationally expensive

Circle Drawing Algorithms



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

MidPoint Circle Algorithm

- Determining pixel positions along a circle circumference using Cartesian or polar coordinates equations still requires a good deal of computation time.
- The Cartesian equation involves multiplications and square-root calculations, while the parametric equations contain multiplications and trigonometric calculations.
- More efficient circle algorithms are based on incremental calculation of decision parameters, as in the Bresenham line algorithm, which involves only simple integer operations.

MidPoint Circle Algorithm

- Bresenham's line algorithm for raster displays is adapted to circle generation by setting up decision parameters for finding the closest pixel to the circumference at each sampling step.

MidPoint Circle Algorithm

- A method for direct distance comparison is to test the halfway position between two pixels to determine if this midpoint is inside or outside the circle boundary.
- This method is more easily applied to other conics; and for an integer circle radius, the midpoint approach generates the same pixel positions as the Bresenham circle algorithm.
- Also, the error involved in locating pixel positions along any conic section using the midpoint test is limited to one-half the pixel separation.

MidPoint Circle Algorithm

- 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

MidPoint Circle Algorithm

- 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

$$f_{\text{circle}}(x,y) = x^2 + y^2 - r^2$$

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

MidPoint Circle Algorithm

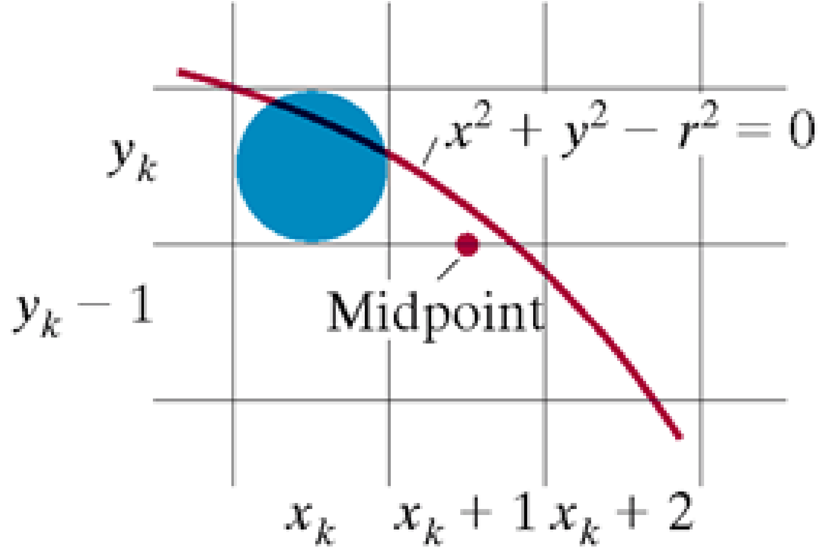


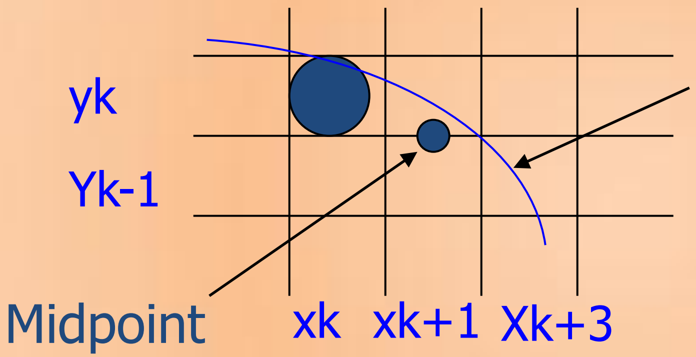
Figure 3-19

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

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,
  - $f_{\text{circle}}(x, y) < 0$ if (x, y) is inside the circle boundary
  - $f_{\text{circle}}(x, y) = 0$ if (x, y) is on the circle boundary
  - $f_{\text{circle}}(x, y) > 0$ if $(x, y)$is outside the circle boundary

MidPoint Circle Algorithm



$x^2 + y^2 - r^2$

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 decide which one of the following two pixels is closer to the circle:

$$(x_{k+1}, y_k) \quad \text{or} \quad (x_{k+1}, y_{k-1})$$

- Our decision parameter is the circle function evaluated at the midpoint between these two pixels

MidPoint Circle Algorithm

$$P_k = f_{\text{circle}}\left(x_k + 1, y_k - \frac{1}{2}\right)$$

$$P_k = (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2$$

- 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

$$P_{k+1} = f_{\text{circle}}\left(x_{k+1} + 1, y_{k+1} - \frac{1}{2}\right)$$

$$P_{k+1} = (x_{k+1} + 1)^2 + \left(y_{k+1} - \frac{1}{2}\right)^2 - r^2$$

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

MidPoint Circle Algorithm

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

If $P_k < 0$

$$Y_{k+1} = Y_k$$

$$P_{k+1} = P_k + 2(x_k + 1) + ({y_k}^2 - {y_k}^2) - (y_k - y_k) + 1$$

$$P_{k+1} = P_k + 2(x_k + 1) + 1$$

$$P_{k+1} = P_k + 2x_{k+1} + 1$$

MidPoint Circle Algorithm

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

If $P_k \geq 0$

$$Y_{k+1} = Y_k - 1$$

$$P_{k+1} = P_k + 2(x_k + 1) + \left((y_k - 1)^2 - {y_k}^2\right) - (y_k - 1 - y_k) + 1$$

$$P_{k+1} = P_k + 2(x_k + 1) + (-2y_k + 1) + 1 + 1$$

$$P_{k+1} = P_k + 2(x_k + 1) - 2y_k + 2 + 1$$

$$P_{k+1} = P_k + 2(x_k + 1) - 2(y_k - 1) + 1$$

$$P_{k+1} = P_k + 2x_{k+1} - 2y_{k+1} + 1$$

MidPoint Circle Algorithm

- 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:

$$2X_{k+1} = 2X_k + 2$$
$$2Y_{k+1} = 2Y_k - 2$$

MidPoint Circle Algorithm

Midpoint Circle Algorithm

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

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

2. Calculate the initial value of the decision parameter as

$$p_0 = \frac{5}{4} - r$$

MidPoint Circle Algorithm

$$p_0 = f_{\text{circle}}\left(1, r - \frac{1}{2}\right)$$

$$= 1 + \left(r - \frac{1}{2}\right)^2 - r^2$$

or

$$p_0 = \frac{5}{4} - r$$

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

$$p_0 = 1 - r \qquad \text{(for } r \text{ an integer)}$$

since all increments are integers.

MidPoint Circle Algorithm

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

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

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 $2y_{k+1} = 2y_k - 2$.

MidPoint Circle Algorithm

4. Determine symmetry points in the other seven octants.
5. Move each calculated pixel position (x, y) onto the circular path centered on (x_c, y_c) and plot the coordinate values:

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

6. Repeat steps 3 through 5 until $x \geq y$.

Example 3-2 Midpoint Circle-Drawing

Given a circle radius $r = 10$, we demonstrate the midpoint circle algorithm by determining positions along the circle octant in the first quadrant from $x = 0$ to $x = y$. The initial value of the decision parameter is

$$p_0 = 1 - r = -9$$

For the circle centered on the coordinate origin, the initial point is $(x_0, y_0) = (0, 10)$, and initial increment terms for calculating the decision parameters are

$$2x_0 = 0, \qquad 2y_0 = 20$$

Successive decision parameter values and positions along the circle path are calculated using the midpoint method as

k	p_k	(x_{k+1}, y_{k+1})	$2x_{k+1}$	$2y_{k+1}$
0	−9	(1, 10)	2	20
1	−6	(2, 10)	4	20
2	−1	(3, 10)	6	20
3	6	(4, 9)	8	18
4	−3	(5, 9)	10	18
5	8	(6, 8)	12	16
6	5	(7, 7)	14	14

MidPoint Circle Algorithm

```
#include "device.h"

void circleMidpoint (int xCenter, int yCenter, int radius)
{
  int x = 0;
  int y = radius;
  int p = 1 - radius;
  void circlePlotPoints (int, int, int, int);

  /* Plot first set of points */
  circlePlotPoints (xCenter, yCenter, x, y);

  while (x < y) {
    x++;
    if (p < 0)
      p += 2 * x + 1;
    else {
      y--;
      p += 2 * (x - y) + 1;
    }
    circlePlotPoints (xCenter, yCenter, x, y);
  }
}
```

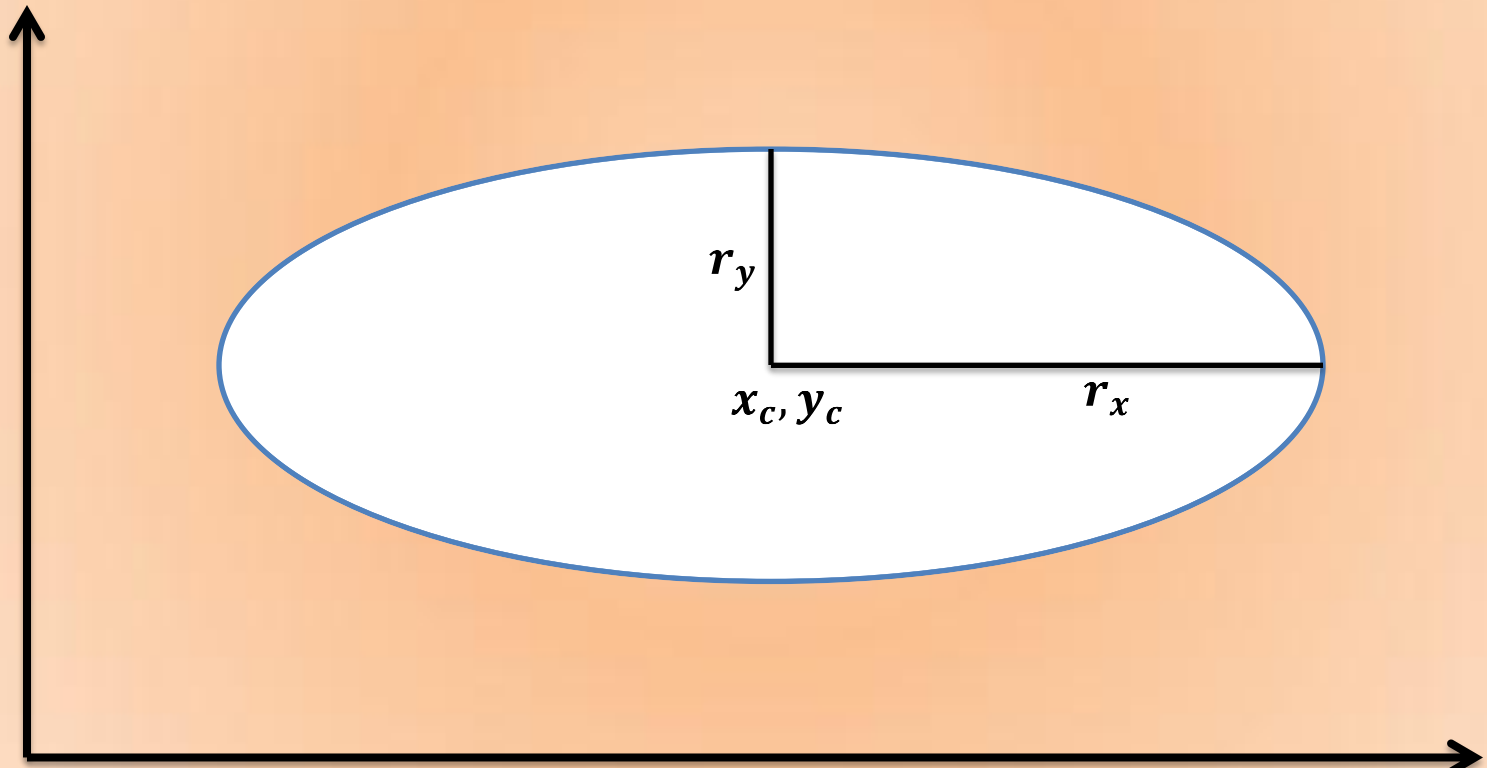
MidPoint Circle Algorithm

```
void circlePlotPoints (int xCenter, int yCenter, int x, int y)
{
  setPixel (xCenter + x, yCenter + y);
  setPixel (xCenter - x, yCenter + y);
  setPixel (xCenter + x, yCenter - y);
  setPixel (xCenter - x, yCenter - y);
  setPixel (xCenter + y, yCenter + x);
  setPixel (xCenter - y, yCenter + x);
  setPixel (xCenter + y, yCenter - x);
  setPixel (xCenter - y, yCenter - x);
}
```

MidPoint Circle Algorithm

- **Last update on 11-3-2014**

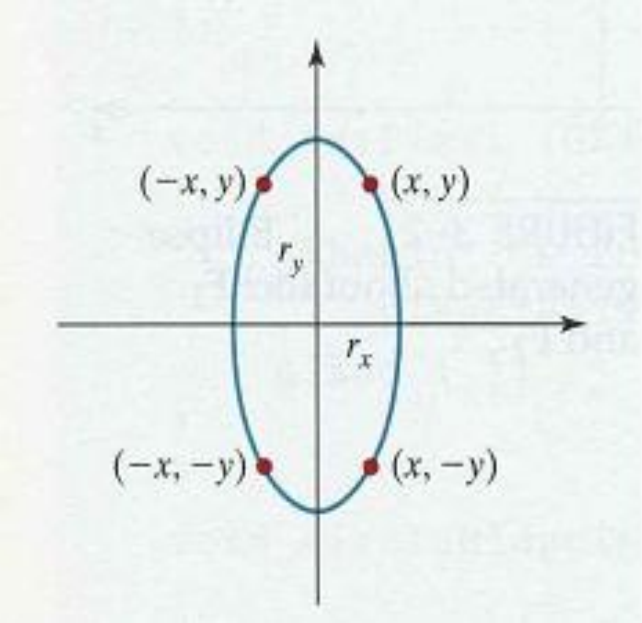
Ellipse Drawing Algorithm

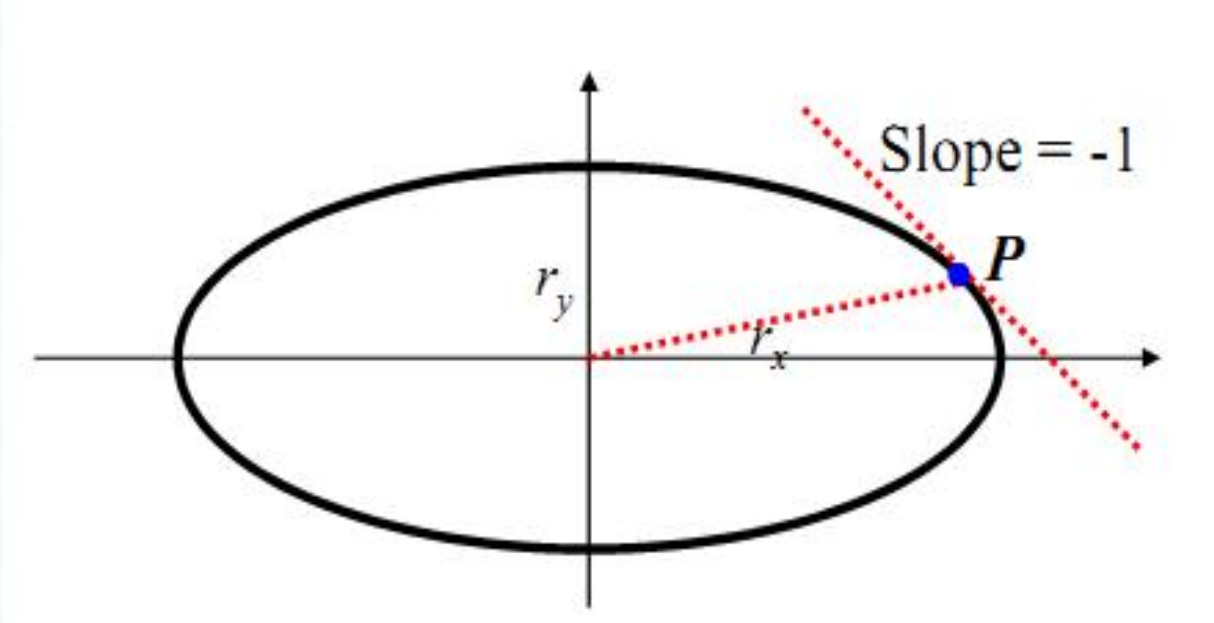


Midpoint Ellipse Algorithm

- Use symmetry of ellipse
- Divide the quadrant into two regions
  - the boundary of two regions is the point at which the curve has a slope of -1.
  - Process by taking unit steps in the x direction to the point P, then taking unit steps in the y direction
  - Apply midpoint algorithm.

Midpoint Ellipse Algorithm



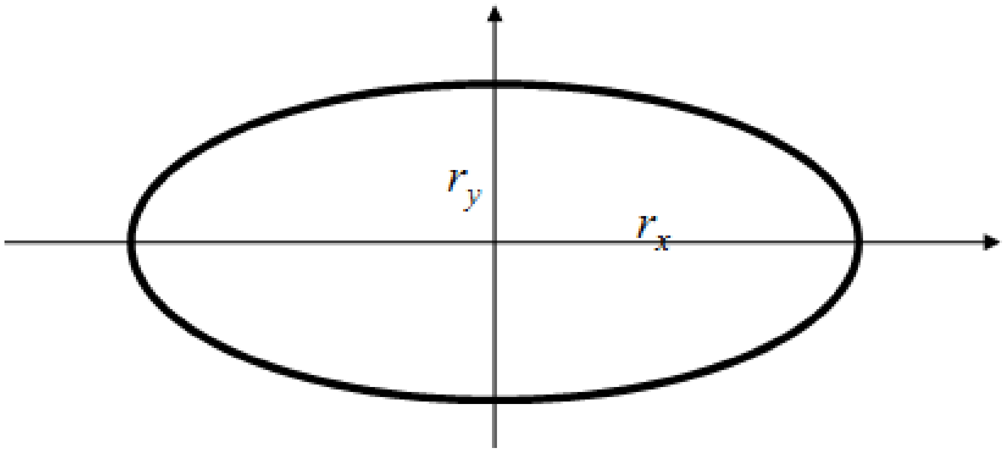


Midpoint Ellipse Algorithm

$$\left(\frac{x - x_c}{r_x}\right)^2 + \left(\frac{y - y_c}{r_y}\right)^2 = 1$$

$$f_{\text{ellipse}}(x, y) = r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2 \begin{cases} < 0 \text{ inside the ellipse boundary} \\ = 0 \text{ on the ellipse boundary} \\ > 0 \text{ outside the ellipse boundary} \end{cases}$$

Midpoint Ellipse Algorithm



$$p1_k = f_{\text{ellipse}}\,(x_k+1,\, y_k - ½)$$

$$p2_k = f_{\text{ellipse}}\,(x_k+½,\, y_k - 1)$$

Ellipse Drawing Algorithm

1. Input r_x , r_y and ellipse center (x_c, y_c) and obtain the first point on an ellipse centered on the origin as

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

Ellipse Drawing Algorithm

2. Calculate the initial value of the decision parameter in region 1 as

$$P1_0 = {r_y}^2 - {r_x}^2 r_y + \frac{1}{4}{r_x}^2$$

Ellipse Drawing Algorithm

3. At each x_k position in region 1, starting at k=0 perform the following test:

If $P1_k < 0$,

– The next point along the ellipse centered on (0,0) is $(x_k + 1,\ y_k)$ and

$$\boldsymbol{P1_{k+1} = P1_k + 2{r_y}^2 x_{k+1} + {r_y}^2}$$

Else

– The next point along the ellipse centered on (0,0) is $(x_k + 1,\ y_k - 1)$ and

$$\boldsymbol{P1_{k+1} = P1_k + 2{r_y}^2 x_{k+1} - 2{r_x}^2 y_{k+1} + {r_y}^2}$$

Ellipse Drawing Algorithm

where

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

$$2{r_x}^2y_{k+1} = 2{r_x}^2y_k - 2{r_x}^2$$

And continue until

$$2{r_y}^2x \ \geq 2{r_x}^2y$$

Ellipse Drawing Algorithm

4. Calculate the initial value of the decision parameter in region 2 using the last point (x_0, y_0) calculated in region 1 as

$$P2_0 = {r_y}^2\left(x_0 + \frac{1}{2}\right)^2 + {r_x}^2(y_0 - 1)^2 - {r_x}^2{r_y}^2$$

Ellipse Drawing Algorithm

5. At each y_k position in region 2, starting at k=0 perform the following test until $y = 0$:

If $P2_k > 0$,

– The next point along the ellipse centered on $(0,\ 0)$ is $(x_k,\ y_k - 1)$ and

$$\boldsymbol{P2_{k+1} = P2_k - 2{r_x}^2 y_{k+1} + {r_x}^2}$$

Else

– The next point along the ellipse centered on $(0, 0)$ is $(x_k + 1,\ y_k - 1)$ and

$$\boldsymbol{P2_{k+1} = P2_k + 2{r_y}^2 x_{k+1} - 2{r_x}^2 y_{k+1} + {r_x}^2}$$

Ellipse Drawing Algorithm

6. Determine symmetry points in the other three quadrants.
7. Move each calculated pixel position (x, y) onto the elliptical path centered on $(x_c,\ \ y_c)$ and plot the coordinate values:

$$\boldsymbol{x = x + x_c}$$
$$\boldsymbol{y = y + y_c}$$

Ellipse Drawing Algorithm

Ellipse Drawing Algorithm