Chapter 3

Scan Conversion Algorithms (Circle and Ellipse)

Scan Conversion Algorithms

1. Scan Conversion of Point
2. Scan Conversion of Line
3. **Scan Conversion of Circle**
4. Scan Conversion of Ellipse
5. Scan Conversion of Polygons

Scan Converting Circle

- A circle is defined as the locus of points at a distance r from a fixed point *(h,k)*. This distance is described In the Pythagoras theorem as :

$$(x-h)^2+(y-k)^2=r^2$$

  where *(h,k)* is the centre and *r* is the radius of the circle

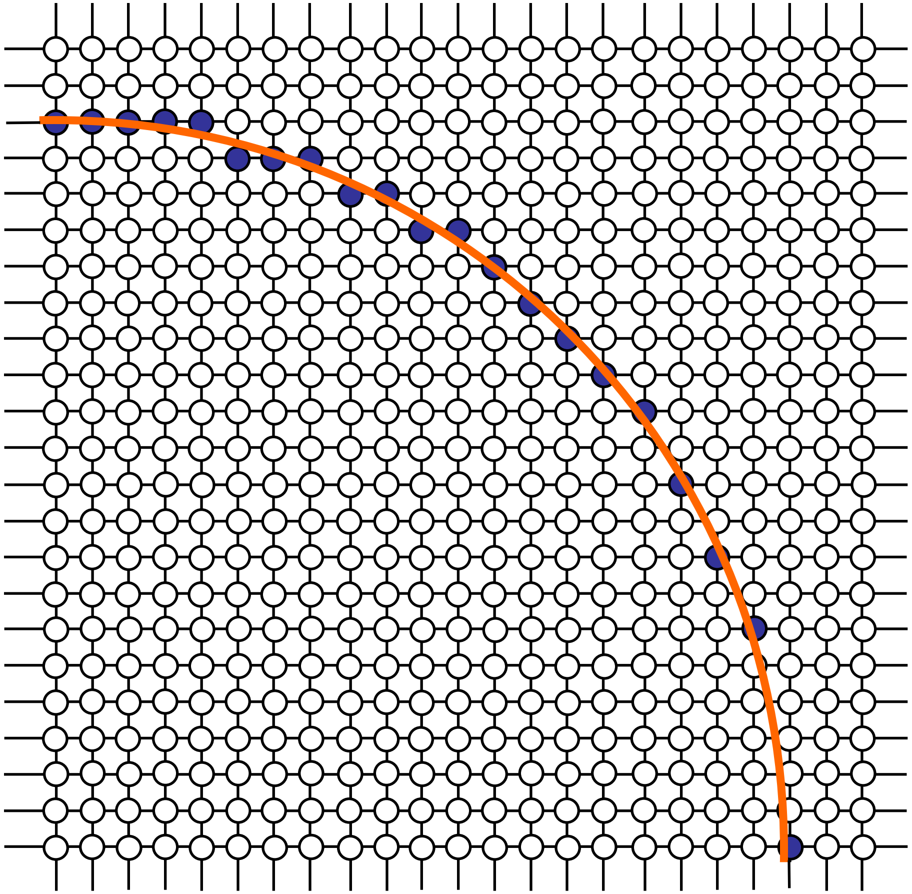
- The standard equation for the circle at centre (0,0) is:

$$x^2+y^2=r^2$$

- So, we can write a simple circle drawing algorithm by solving the equation for *y* at unit *x* intervals using:

$$y=\pm\sqrt{r^2-x^2}$$

Scan Converting Circle



$$y_0 = \sqrt{20^2 - 0^2} \approx 20$$

$$y_1 = \sqrt{20^2 - 1^2} \approx 20$$

$$y_2 = \sqrt{20^2 - 2^2} \approx 20$$

●
●
●

$$y_{19} = \sqrt{20^2 - 19^2} \approx 6$$

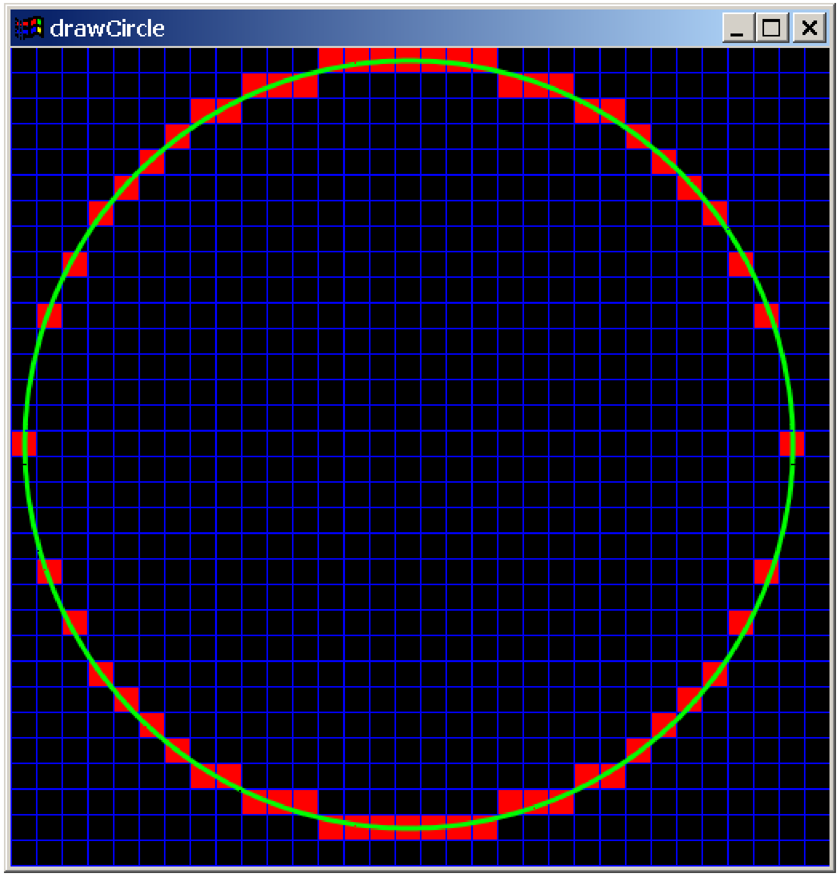
$$y_{20} = \sqrt{20^2 - 20^2} \approx 0$$

Scan Converting Circle

- However, unsurprisingly this is not a brilliant solution!
- Firstly, it involves all floating point calculations
- Secondly, the calculations are not very efficient
  - The square (multiply) operations
  - The square root operation – try really hard to avoid these!
- Thirdly, the resulting circle has large gaps where the slope approaches the vertical

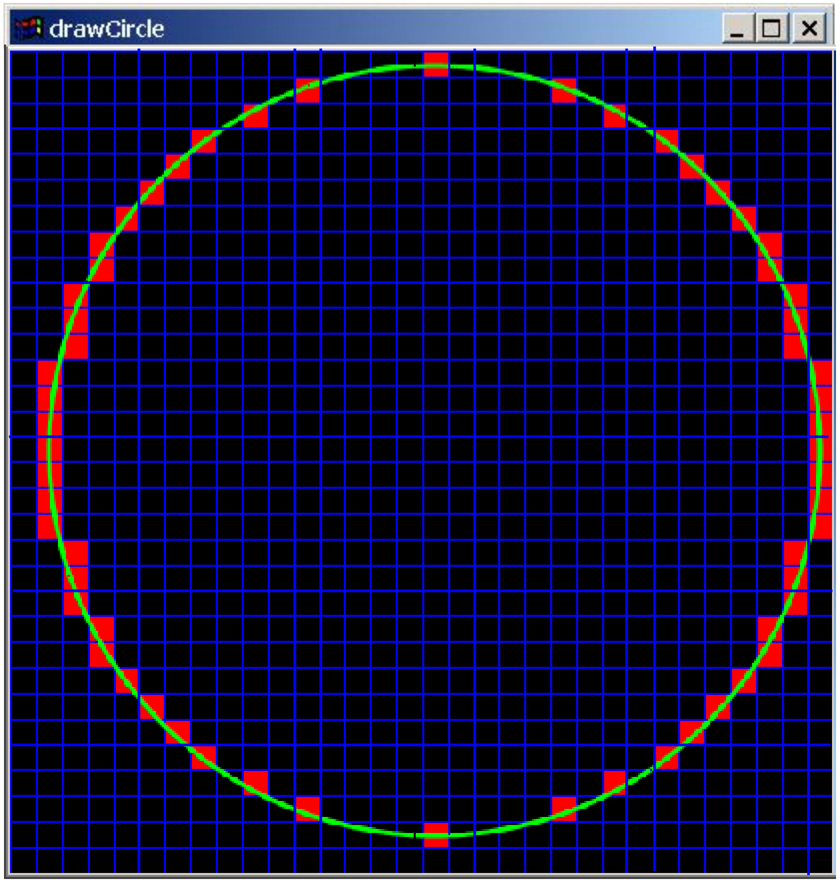
Scan Converting Circle

- We came across this problem with lines before…
- Sometimes the slope of the line tangent to a point the circle is greater than 1. Stepping in *x* won't work there.
- So we could look for this case and step in *y* …



Scan Converting Circle

- But that's also not a good solution!
- But on both cases, we were taking advantage of the circle's symmetry when we draw both positive and negative values for *y* (or x).
- This is called 2-way symmetry.
- Are there more symmetries to exploit?

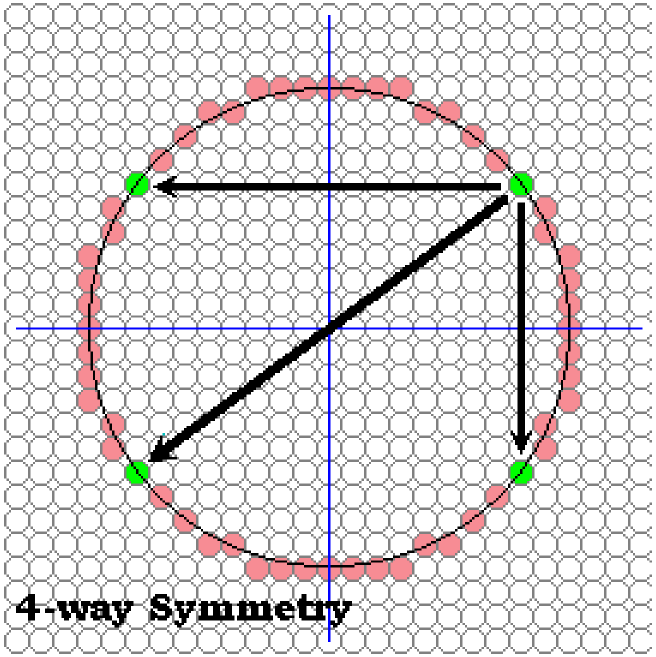


Scan Converting Circle

– Sure, let's try 4-way symmetry.

– With just a quick hack, we get:

```
put_circle_pixel(x,y,h,k)
  {
        Put_pixel(h+x, k+y)
        Put_pixel(h-x, k+y)
        Put_pixel(h-x, k-y)
        Put_pixel(h+x, k-y)
  }
```

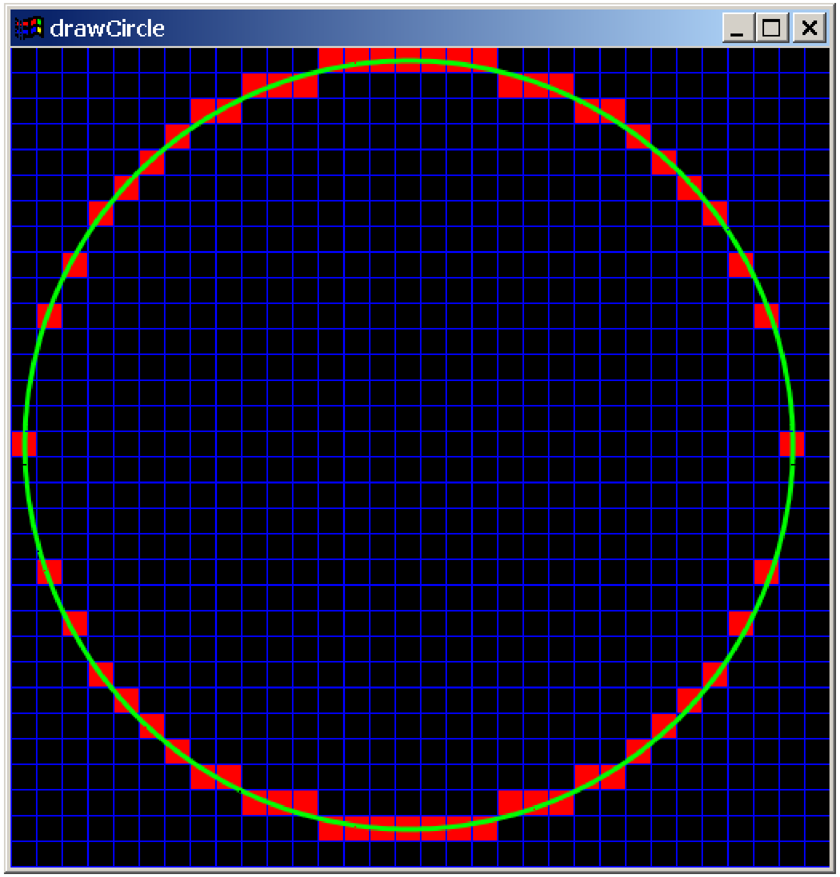


Scan Converting Circle

– That didn't fix a thing.

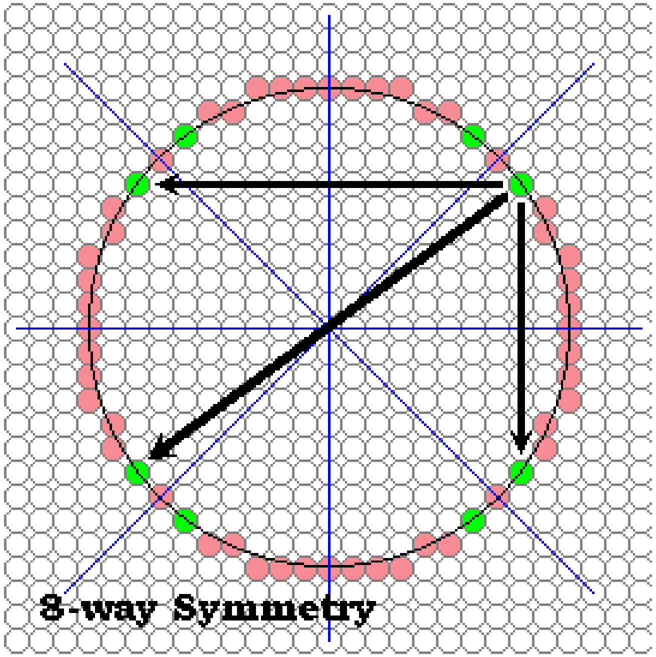
– Oh sure, it's *faster* – just half as many evaluations – but it's no more correct than our first try.

– Why didn't this work?



Scan Converting Circle

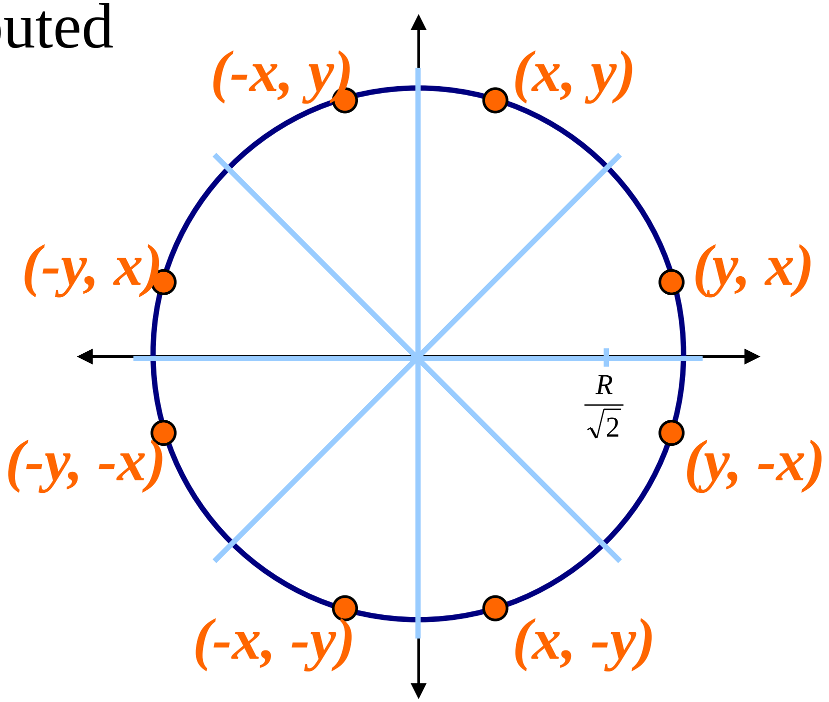
- What about 8-way symmetry?
- Now we're looking for symmetries across the diagonal lines, i.e. where *x*=*y*.
- Now when we step along *x*, we can permute the coordinates (swap *x* and *y*) and simultaneously be stepping along *y* in another part of the circle.
- That's back to how we fixed lines!
- Bonus, it's 4 times faster than the original!



Scan Converting Circle

Eight-Way Symmetry

- We can use *eight-way symmetry* to make our circle drawing algorithm more efficient.
- Now circle points are computed only in one octant, rest of the circle are found by symmetry.
- Centre can be shifted while plotting



Scan Converting Circle

– We define a routine that plots circle pixels with centre (h,k) in all the eight octants

```
put_circle_pixel(x,y,h,k)
    {
        Put_pixel(h+x, k+y)
        Put_pixel(h-x, k+y)
        Put_pixel(h-x, k-y)
        Put_pixel(h+x, k-y)
        Put_pixel(h+y, k+x)
        Put_pixel(h-y, k+x)
        Put_pixel(h-y, k-x)
        Put_pixel(h+y, k-x)
    }
```

Circle Drawing Algorithms

1. Midpoint Circle Algorithm

Midpoint Circle Algorithm

1. **Introduction**
   - Similarly to the case with lines, there is an incremental algorithm for drawing circles – the *mid-point circle algorithm.*
   - In the mid-point circle algorithm we use implicit equation of the circle.
   - Integer calculations are used to compute the circle points in one octant, rest of the seven points are plotted using eight way symmetry.
   - The equations derived will be similar to Bresenham’s circle algorithm

Midpoint Circle Algorithm

2. Basic Concept

- We will first calculate pixel positions for a circle centered around the origin (0,0). Centre is shifted later on.
- 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

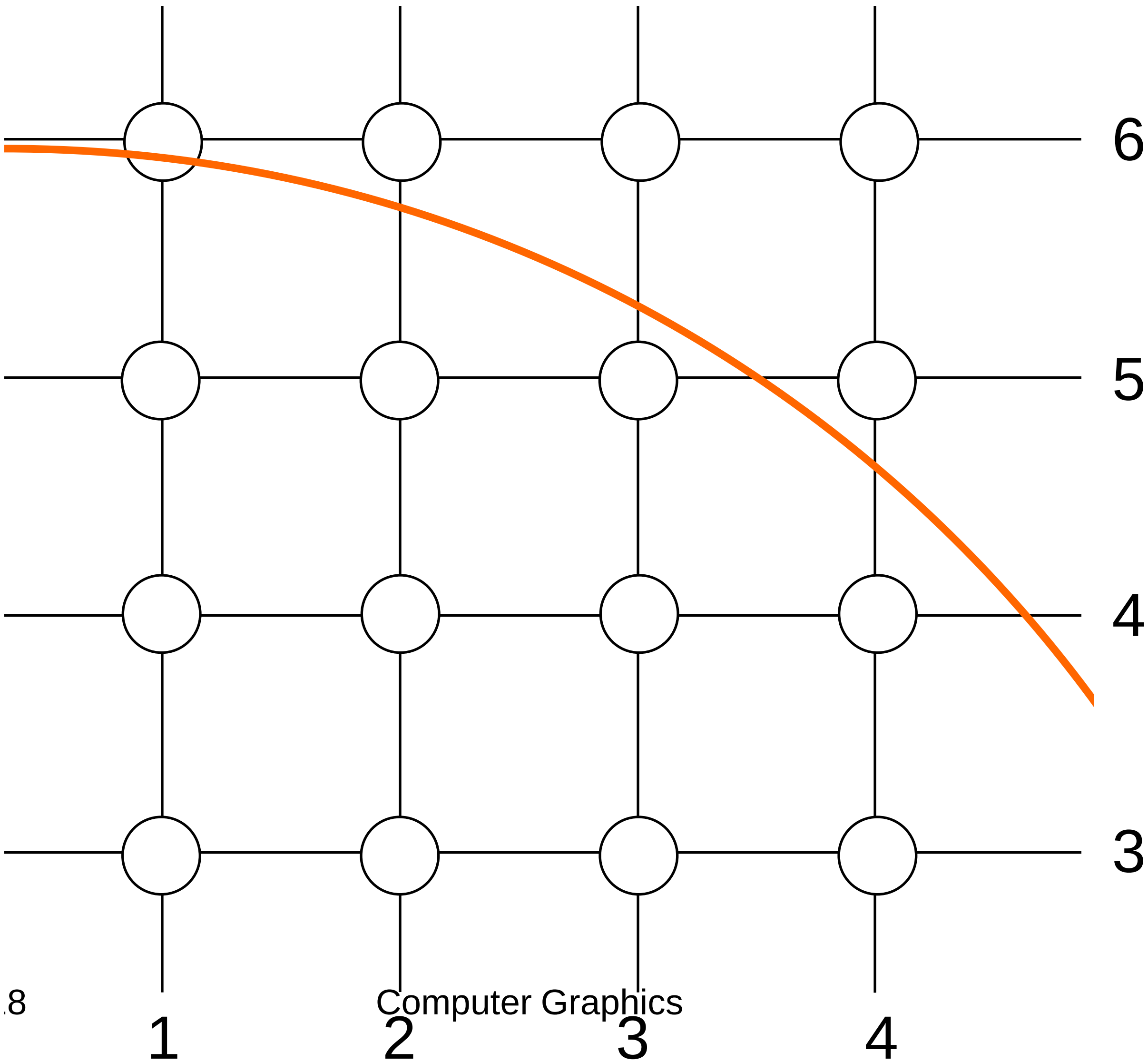
$$f_{circle}(x,y) = x^2 + y^2 - r^2$$

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

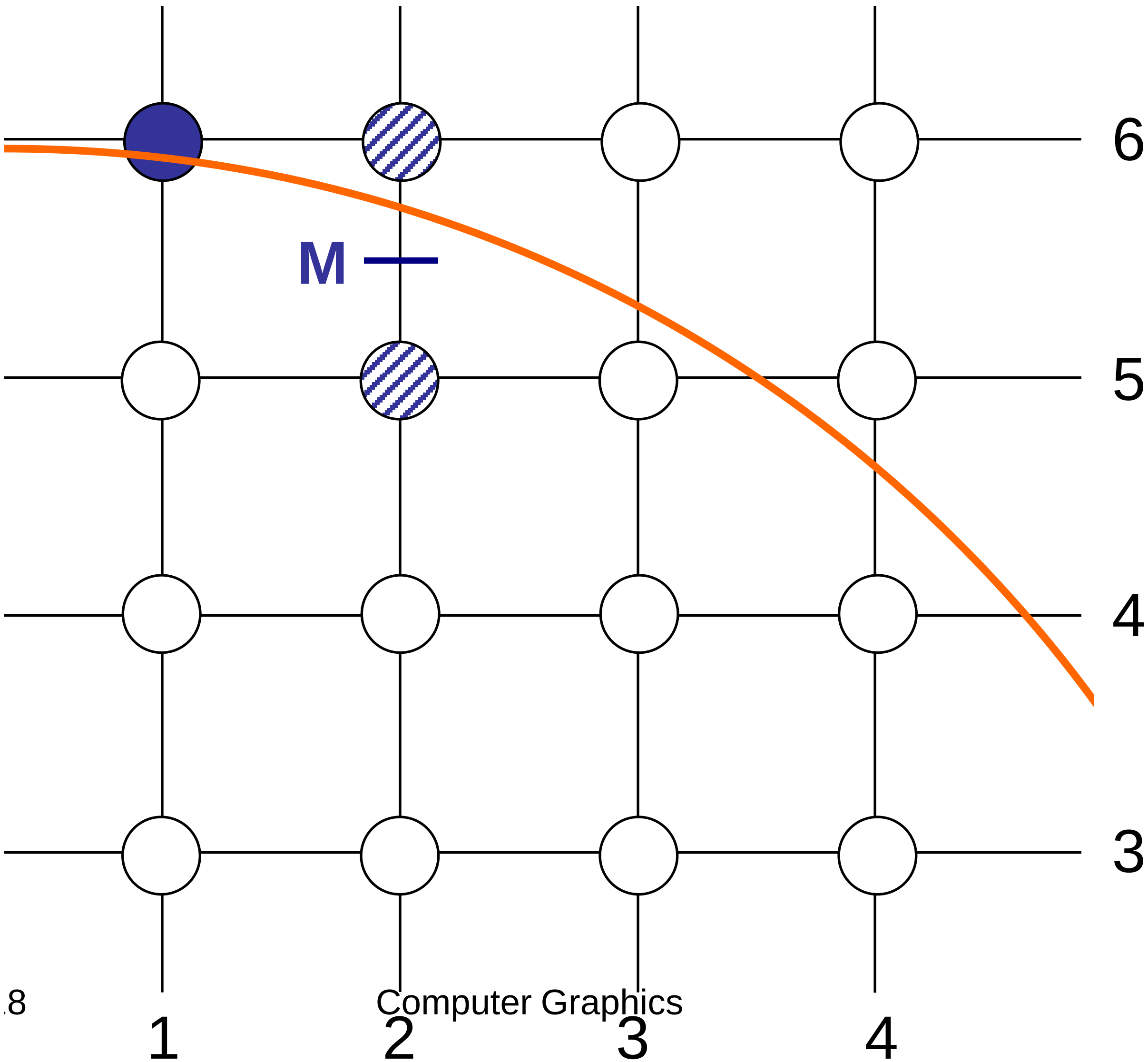
Midpoint Circle Algorithm

- 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, we have a discriminator function
  - $f_{circle}(x,y) < 0$ 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
- The algorithm does the above test at mid point $f_{circle}(x,y-½)$, if it lies inside outer point is plotted, else inner is considered to be the better choice

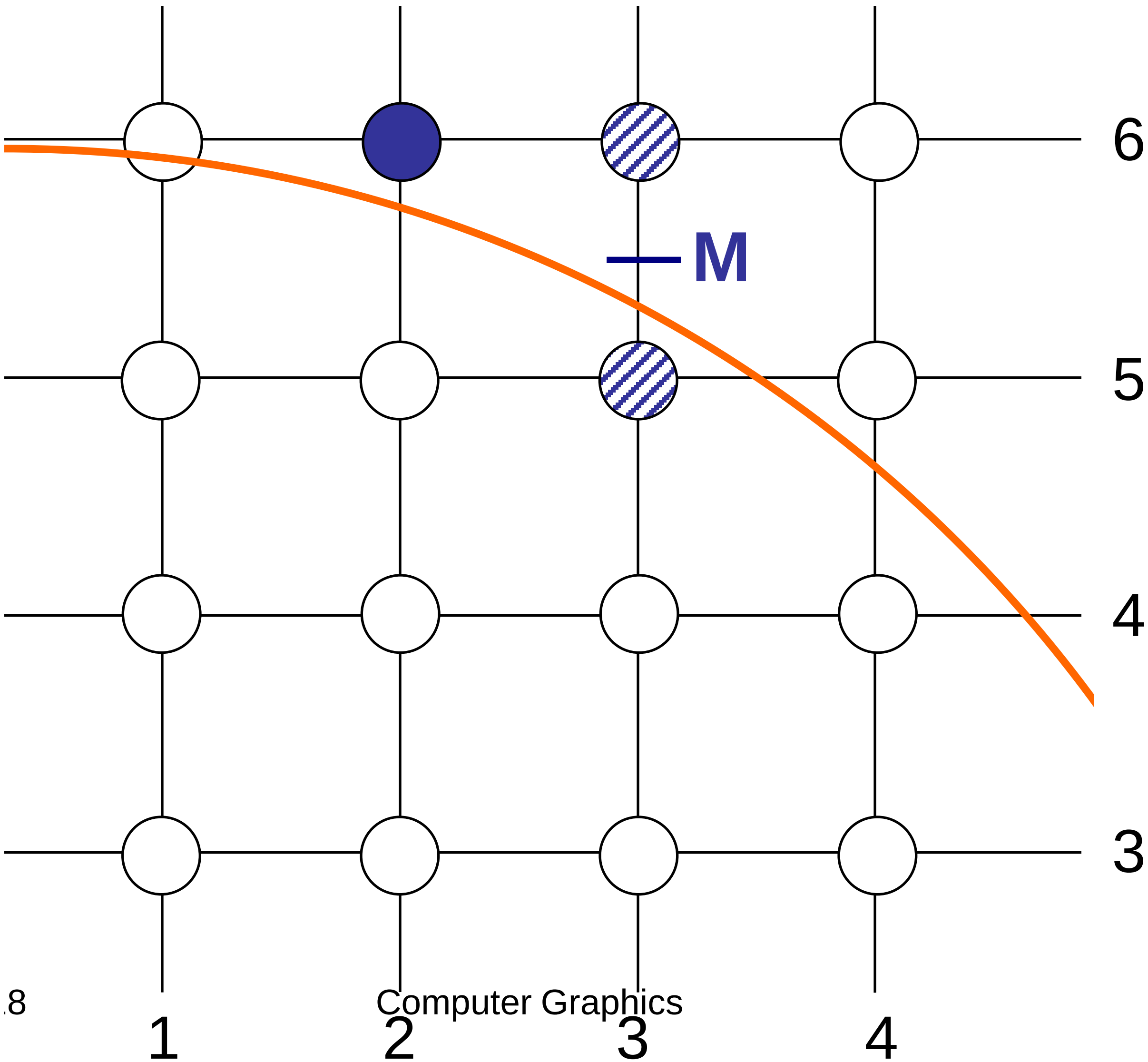
Midpoint Circle Algorithm



Midpoint Circle Algorithm



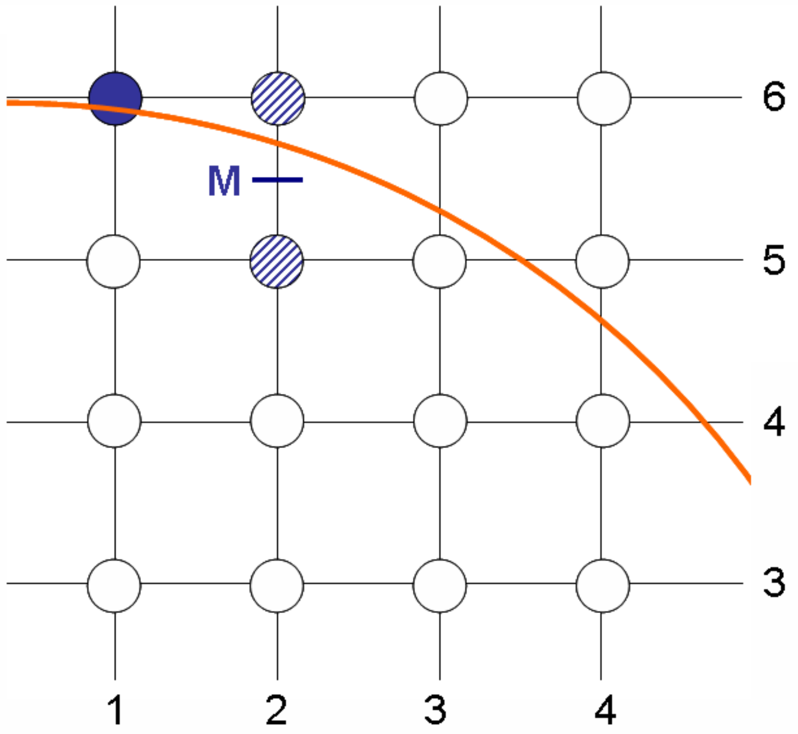
Midpoint Circle Algorithm



Midpoint Circle Algorithm

3. Derivation

– Let us assume that $P(x_k, y_k)$ is the currently plotted pixel. $Q(x_{k+1}, y_{k+1}) \leftrightarrow (x_{k+1}, y)$ is the next point along the actual circle path. We need to decide next pixel to be plotted from among candidate positions $Q1(x_k+1, y_k)$ or $Q2(x_k+1, y_k-1)$



Midpoint Circle Algorithm

- 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

$$p_k = (x_k + 1)^2 + (y_k - \tfrac{1}{2})^2 - r^2$$

Put k = k+1

$$p_{k+1} = [(x_{k+1}) + 1]^2 + (y_{k+1} - \tfrac{1}{2})^2 - r^2$$
$$= (x_k + 2)^2 + (y_{k+1} - \tfrac{1}{2})^2 - r^2$$

subtracting p_k from p_{k+1}

$$p_{k+1} - P_k = (x_k + 2)^2 + (y_{k+1} - \tfrac{1}{2})^2 - [(x_k + 1)^2 + (y_k - \tfrac{1}{2})^2]$$

or

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

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

Midpoint Circle Algorithm

If $p_k < 0$

$\Rightarrow$ Q1(x_k+1, y_k) was the next choice

$\Rightarrow y_{k+1} = y_k$

$\Rightarrow (y_{k+1} - ½)^2 - (y_k - ½)^2 = 0$

$\Rightarrow p_{k+1} = p_k + 2.x_k + 3$

else

Q2(x_k+1, y_k-1) was the next choice

$\Rightarrow y_{k+1} = y_k - 1$

$\Rightarrow (y_{k+1} - ½)^2 - (y_k - ½)^2 = - 2.y_k + 2$

$\Rightarrow p_{k+1} = p_k + 2(x_k - y_k) + 5$

Midpoint circle algorithm

Initial decision parameter is obtained by evaluating the circle function at the start position $(x0,y0) = (0,r)$

$$p_0 = f_{circle}(1, r - 1/2)$$
$$= 1 + (r - 1/2)^2 - r^2$$
$$= 5/4 - r$$

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

$$p_0 = 1 - r$$

Midpoint Circle Algorithm

5. Example

- To see the mid-point circle algorithm in action lets use it to draw a circle centred at (0,0) with radius 10
- To see the mid-point circle algorithm in action lets use it to draw a circle centred at (0,0) with radius 16.

$p_0 = 1 - r,$

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

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

Midpoint Circle Algorithm

1. Input radius r and circle center (x_c, y_c), then set the coordinates for 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

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

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

4. Determine symmetry points in the other seven octants.

5. Move each calculated pixel position (x, y) onto the circular path centered at (x_c, y_c) and plot the coordinate values:

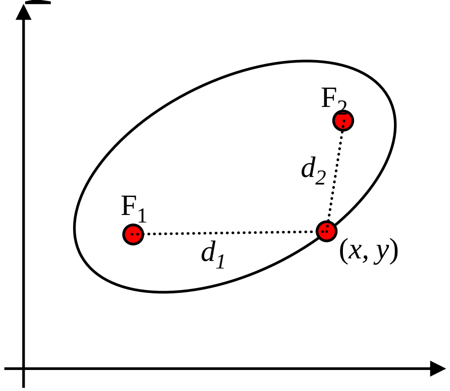
$$x = x + x_c, \qquad y = y + y_c$$

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

Scan Conversion Algorithms

1. Scan Conversion of Point
2. Scan Conversion of Line
3. Scan Conversion of Circle
4. **Scan Conversion of Ellipse**
5. Scan Conversion of Polygons

Scan Converting Ellipse



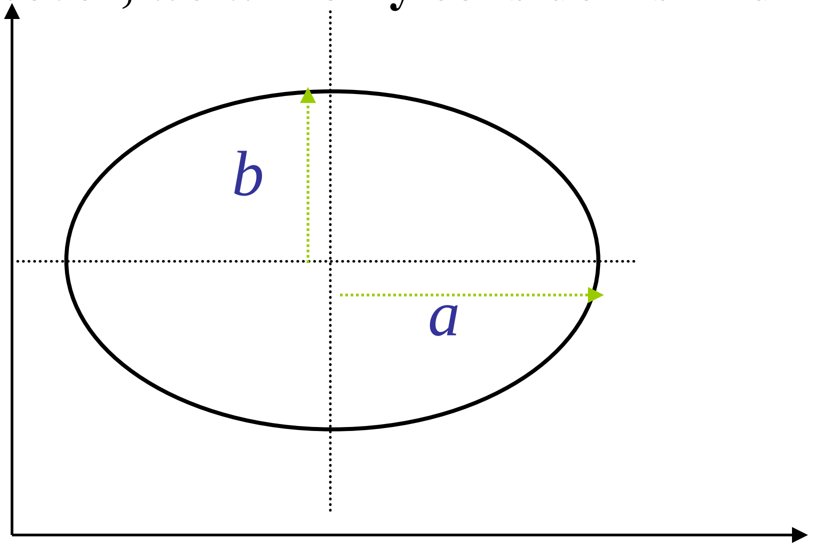
General equation of an ellipse:

$$d_1 + d_2 = constant$$

Or,

$$\sqrt{(x-x_1)^2+(y-y_1)^2}+\sqrt{(x-x_2)^2+(y-y_2)^2}=\text{constant}$$

However, we will only consider 'standard' ellipse:



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

Scan Converting Ellipse

- Formally An ellipse is defined as the locus of points satisfying following equation: r from a fixed point *(h,k)*. This distance is described In the Pythagoras theorem as :

$$\frac{(x-h)^2}{a^2}+\frac{(y-k)^2}{b^2}=1$$

where *(h,k)* is the centre, a and b are the length of major and minor axis respectively

- The standard equation for the ellipse at centre (0,0) is:

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$$

Scan Converting Ellipse

– So, we can write a simple circle drawing algorithm by solving the equation for *y* at unit *x* intervals using:

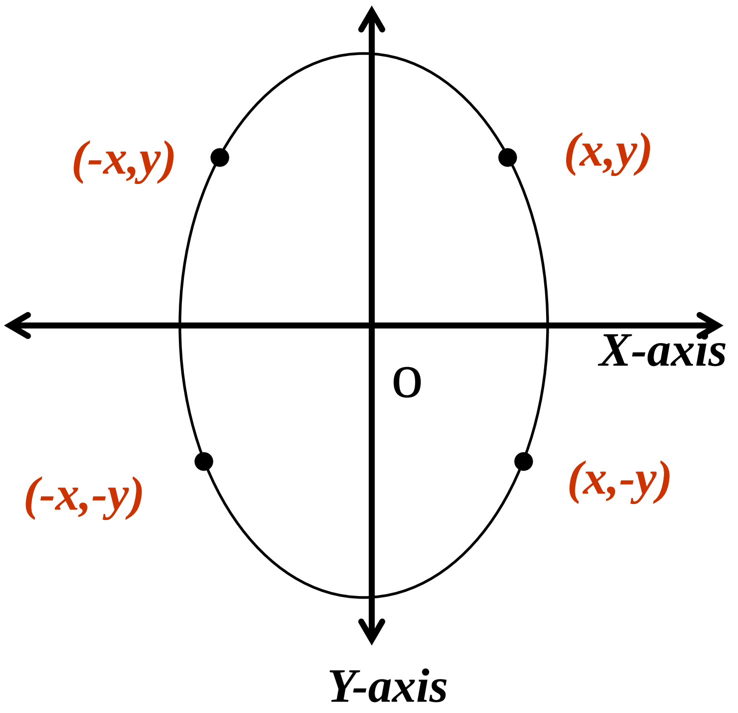
$$y = b.(\pm\sqrt{1^2 - (x/a)^2})$$

– Which is not a good solution. (you know that)

Scan Converting Ellipse

Four-Way Symmetry

- Like circle, ellipse centred at (*0*, *0*) follows symmetry. But now it is *four symmetry*.
- Thus ellipse points are computed in the first quadrant, rest three of the ellipse points are found by symmetry.
- Centre can be shifted while plotting



Scan Converting Ellipse

– So we define a routine that plots ellipse pixels with centre (h,k) in all the four quadrants

```
put_ellipse_pixel(x,y,h,k)
{
        Put_pixel(h+x, k+y)
        Put_pixel(h-x, k+y)
        Put_pixel(h+x, k-y)
        Put_pixel(h-x, k-y)
}
```

Ellipse Drawing Algorithms

1. Polar Domain Algorithms
2. **Midpoint Ellipse Algorithm**
3. Bresenham Ellipse Algorithm

Midpoint Ellipse Algorithm

Reconsider an ellipse centered at the origin:

$$\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 = 1$$

The discriminator function?

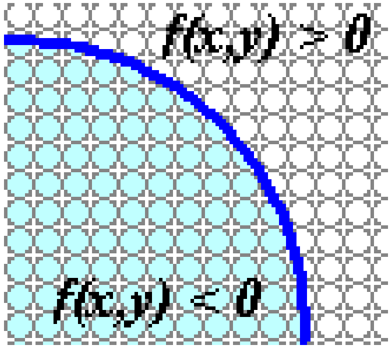
$$f_e(x, y) = b^2x^2 + a^2y^2 - a^2b^2$$

…and its properties:

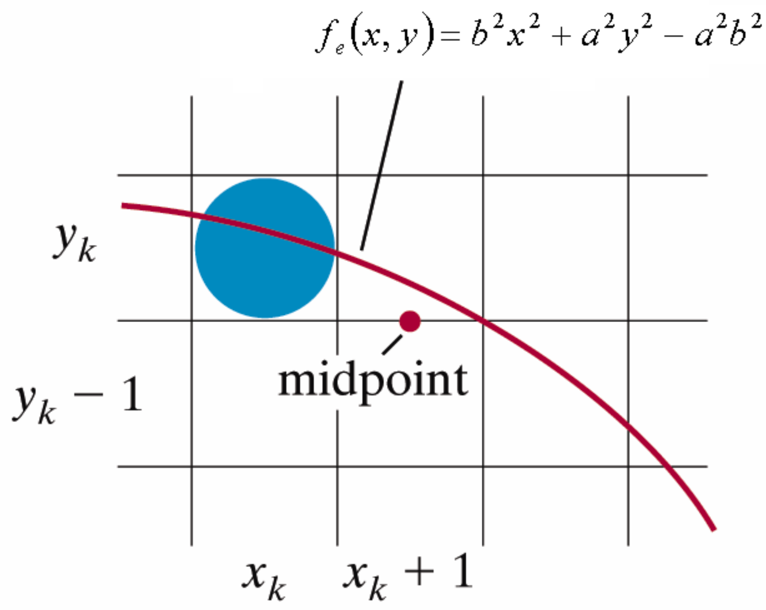
$f_e(x,y) < 0$ for a point inside the ellipse

$f_e(x,y) > 0$ for a point outside the ellipse

$f_e(x,y) = 0$ for a point on the ellipse



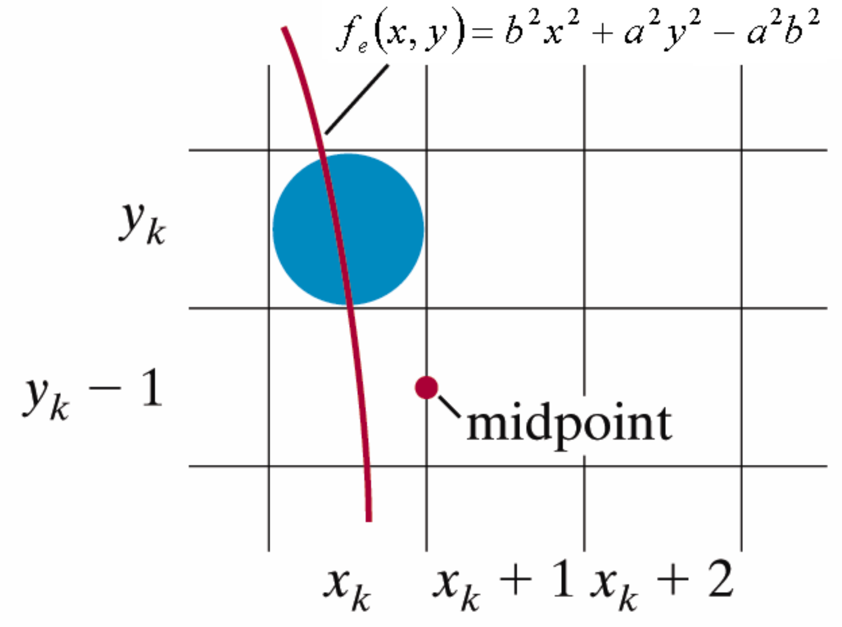
Midpoint Ellipse Algorithm



Midpoint between candidate pixels at sampling position $x_k + 1$ along an elliptical path.

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

Midpoint Ellipse Algorithm

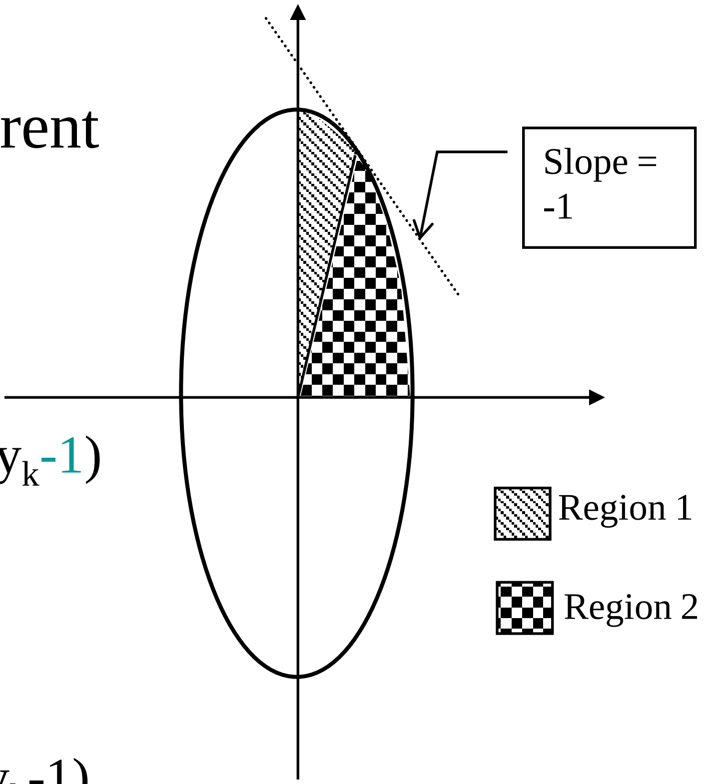


Midpoint between candidate pixels at sampling position $yk - 1$ along an elliptical path.

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

Midpoint Ellipse Algorithm

- Ellipse is different from circle.
- Similar approach with circle, different is sampling direction.
- Region 1:
  - Sampling is at *x* direction
  - Choose between (x_k+1, y_k), or (x_k+1, y_k-1)
  - Midpoint: $(x_k+1, y_k-0.5)$
- Region 2:
  - Sampling is at *y* direction
  - Choose between (x_k, y_k-1), or (x_k+1, y_k-1)
  - Midpoint: $(x_k+0.5, y_k-1)$



Midpoint Ellipse Algorithm

Decision Parameters

– Region 1:

$$p1_k = f_e\left(x_k + 1, y_k - \tfrac{1}{2}\right)$$

$p1_k$ –ve:

- midpoint is inside
- choose pixel $\boldsymbol{(x_{k+1}, y_k)}$

$p1_k$ +ve:

- midpoint is outside
- choose pixel $\boldsymbol{(x_{k+1}, y_k\text{-}1)}$

– Region 2:

$$p2_k = f_e\left(x_k + \tfrac{1}{2}, y_k - 1\right)$$

$p2_k$ –ve:

- midpoint is inside
- choose pixel $\boldsymbol{(x_k+1, y_k\text{-}1)}$

$p2_k$ +ve:

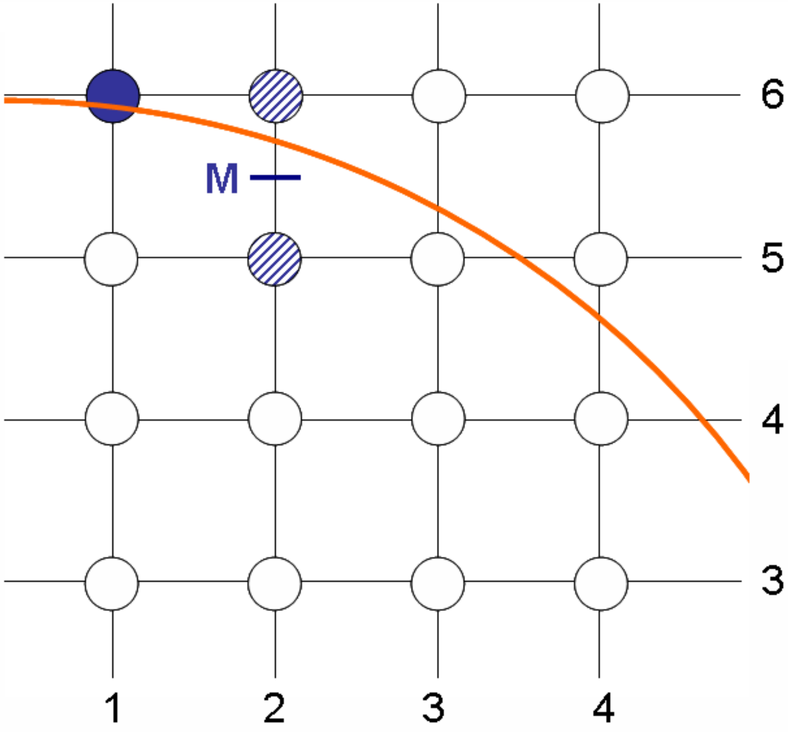
- midpoint is outside
- choose pixel $\boldsymbol{(x_k, y_k\text{-}1)}$

Midpoint Ellipse Algorithm

Derivation for Region 1:

– Let us assume that $P(x_k, y_k)$ is the currently plotted pixel. $Q(x_{k+1}, y_{k+1}) \leftrightarrow (x_{k+1}, y)$ is the next point along the actual circle path. We need to decide next pixel to be plotted from among candidate positions $Q1(x_k+1, y_k)$ or

$Q2(x_k+1, y_k-1)$



Midpoint Ellipse Algorithm

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

$$p1_k = f_e\,(x_k + 1, y_k - 1/2) = b^2(x_k + 1)^2 + a^2(y_k - 1/2)^2 - a^2\,b^2$$

– If $p1_k < 0$,
  - this midpoint is inside the ellipse and
  - the pixel on the scan line y_k is closer to the ellipse boundary.

  Otherwise,
  - the mid position is outside or on the ellipse boundary,
  - and we select the pixel on the scan line y_k-1

Midpoint Ellipse Algorithm

Successive decision parameters are obtained using incremental calculations

$$p1_k \quad = b^2 (x_k +1)^2 + a^2 (y_k - ½ \;)^2 - a^2b^2$$

Put k = k+1

$$p1_{k+1} = b^2 [(x_{k+1})+1]^2 + a^2(y_{k+1} - ½ \;)^2 - a^2b^2$$

$$= b^2 (x_k+2)^2 + a^2(y_{k+1} - ½ \;)^2 - a^2b^2$$

subtracting p_k from p_{k+1}

$$p1_{k+1} - p1_k = b^2(x_k+2)^2 + a^2(y_{k+1} - ½ \;)^2 - [b^2(x_k +1)^2 + a^2(y_k - ½ \;)^2]$$

or

$$p1_{k+1} = p1_k+2\, b^2(x_k+1) + a^2 [(y_{k+1} - ½)^2 - (y_k - ½)^2] + b^2$$

Where y_{k+1} is either y_k or y_{k-1}, depending on the sign of $p1_k$

Midpoint Ellipse Algorithm

If $p1_k < 0$

$\Rightarrow$ Q1(x_k+1, y_k) was the next choice

$\Rightarrow y_{k+1} = y_k$

$\Rightarrow (y_{k+1} - ½)^2 - (y_k - ½)^2 = 0$

$\Rightarrow p1_{k+1} = p1_k + b^2(2.x_k + 3)$

else

Q2(x_k+1, y_k-1) was the next choice

$\Rightarrow y_{k+1} = y_k - 1$

$\Rightarrow (y_{k+1} - ½)^2 - (y_k - ½)^2 = - 2.y_k + 2$

$\Rightarrow p1_{k+1} = p1_k + b^2(2.x_k + 3) + a^2(- 2.y_k + 2)$

Midpoint Ellipse algorithm

Initial decision parameter is obtained by evaluating the circle function at the start position *(x0,y0) = (0,b)*

$$p1_0 = f_e(1, b - 1/2)$$

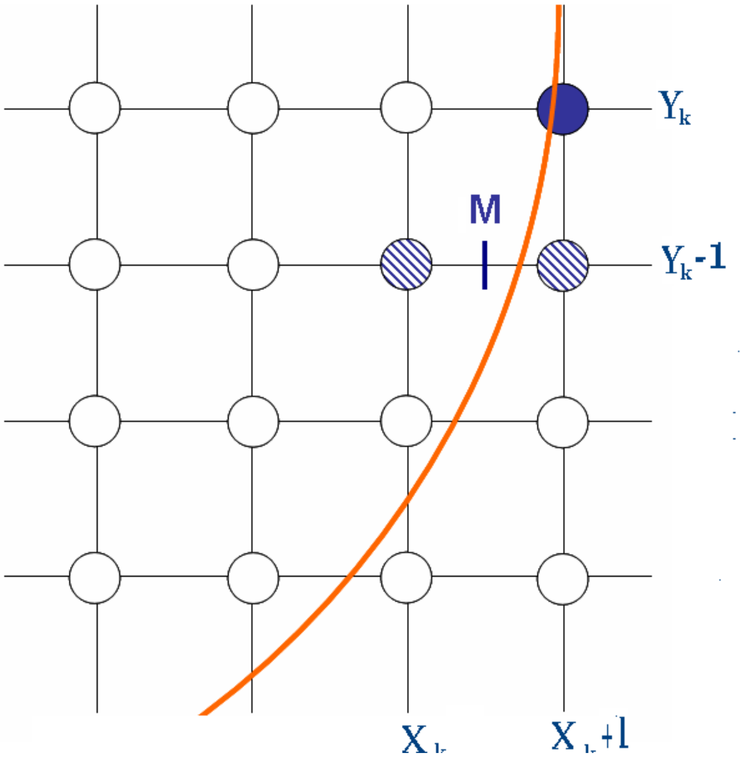
$$= b^2 - a^2 b + \tfrac{1}{4} a^2$$

Midpoint Ellipse Algorithm

Derivation for Region 2:

– Let us assume that $P(x_k, y_k)$ is the currently plotted pixel. $Q(x_{k+1}, y_{k+1}) \leftrightarrow (x_{k+1}, y)$ is the next point along the actual circle path. We need to decide next pixel to be plotted from among candidate positions $Q1(x_k, y_k -1)$ or

$Q2(x_k+1, y_k-1)$



Midpoint Ellipse Algorithm

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

$$p2_k = f_e\,(x_k + 1/2, y_k - 1) = b^2(x_k + 1/2)^2 + a^2(y_k - 1)^2 - a^2 b^2$$

– If $p2_k < 0$,
  - tthe mid position is inside or on the ellipse boundary,
  - and we select the pixel on the scan line x_k+1

  Otherwise,
  - his midpoint is outside the ellipse and
  - the pixel on the scan line x_k is closer to the ellipse boundary.

Midpoint Ellipse Algorithm

Successive decision parameters are obtained using incremental calculations

$$p2_k \quad = b^2 (x_k + ½)^2 + a^2 (y_k - 1 \ )^2 - a^2b^2$$

Put k = k+1

$$p2_{k+1} \ = b^2 [(x_{k+1})+½]^2 + a^2(y_{k+1} - 1 \ )^2 - a^2b^2$$

$$= b^2 [(x_{k+1})+½]^2 + a^2(y_k - 2 \ )^2 - a^2b^2$$

subtracting p_k from p_{k+1}

$$p2_{k+1} - p2_k = b^2(x_{k+1}+½)^2 + a^2(y_k - 2 \ )^2 - [b^2(x_k + ½)^2 + a^2(y_k - 1 \ )^2]$$

or

$$p2_{k+1} = p2_k + b^2[(x_{k+1}+½)^2 - (x_k + ½)^2] - a^2 (2.y_k - 3)$$

Where x_{k+1} is either x_k or x_{k+1}, depending on the sign of $p2_k$

Midpoint Ellipse Algorithm

If $p2_k < 0$

$Q2(x_k+1, y_k-1)$ was the next choice

$\Rightarrow x_{k+1} = x_k+1$

$\Rightarrow (x_{k+1}+½)^2 - (x_k+½)^2 = 2.x_k+2$

$\Rightarrow p2_{k+1} = p2_k + b^2(2.x_k+2) - a^2(2.y_k - 3)$

else

$\Rightarrow Q1(x_k, y_k - 1)$ was the next choice

$\Rightarrow x_{k+1} = x_k$

$\Rightarrow (x_{k+1}+½)^2 - (x_k+½)^2 = 0$

$\Rightarrow p2_{k+1} = p2_k - a^2(2.y_k - 3)$

Midpoint Ellipse algorithm

Initial decision parameter is obtained by evaluating the circle function at the start position (x_0,y_0) *the last point plotted after drawing region 1*

$$p2_0 = f_e(x_0+½, y_0 – 1)$$

$$= b^2(x_0+½)^2 + a^2(y_0 – 1)^2 –a^2b^2$$

Midpoint Ellipse algorithm

- **Condition to move from one region to another**

  Starting at point (0,b) in the region 1 where slope is <-1 we take unit step increment along x-direction until we reach region 2 where slope is >-1. At each step we need to check slope.
- The ellipse slope can be calculated as $\frac{dy}{dx} = -\frac{2b^2x}{2a^2y}$
- At the boundary of region 1 and 2

$$dy/dx = -1$$

$$\Rightarrow 2b^2x = 2a^2y$$

$$\Rightarrow \textit{We move out of region 1 if } 2b^2x \geq 2a^2y$$

Midpoint Ellipse 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

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

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

$$p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

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

$$p1_{k+1} = p1_k + 2r_y^2 x_{k+1} + r_y^2$$

Otherwise, the next point along the ellipse is $(x_k + 1, y_k - 1)$ and

$$p1_{k+1} = p1_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$$

with

$$2r_y^2 x_{k+1} = 2r_y^2 x_k + 2r_y^2, \qquad 2r_x^2 y_{k+1} = 2r_x^2 y_k - 2r_x^2$$

and continue until $2r_y^2 x \geq 2r_x^2 y$.

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

$$p2_0 = r_y^2 \left(x_0 + \frac{1}{2} \right)^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$

where (x_0, y_0) is the last position calculated in region 1.

5. At each y_k position in region 2, starting at $k = 0$, perform the following test. If $p2_k > 0$, the next point along the ellipse centered on $(0, 0)$ is $(x_k, y_k - 1)$ and

$$p2_{k+1} = p2_k - 2r_x^2 y_{k+1} + r_x^2$$

Otherwise, the next point along the ellipse is $(x_k + 1, y_k - 1)$ and

$$p2_{k+1} = p2_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$$

using the same incremental calculations for x and y as in region 1. Continue until $y = 0$.

6. For both regions, 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:

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