CSE- 4105

Lecturer-05

Mid-Point Circle Algorithm

Circle Generating Algorithms

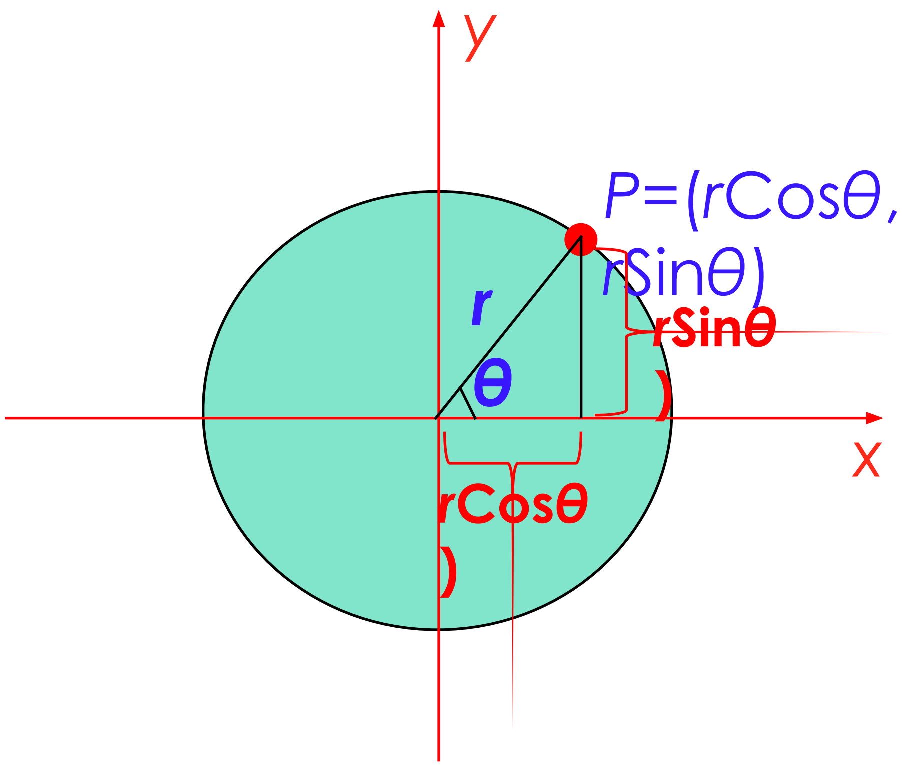
- Circles and ellipses are common components in many pictures.
- Circle generation routines are often included in packages.

Circle Equations

- Polar form

  $x = r\text{Cos}\theta$

  $y = r\text{Sin}\theta$ (*r* = radius of circle)



Drawing a circle

```
θ = 0°
while (θ < 360°)
    x = rCosθ
    y = rSinθ
    setPixel(x,y)
    θ = θ + 1°
end while
```

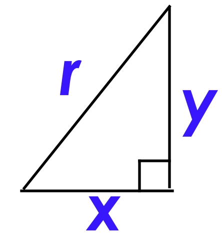
Disadvantages

- To find a complete circle θ varies from 0° to 360°
- The calculation of trigonometric functions is very slow.

Cartesian form

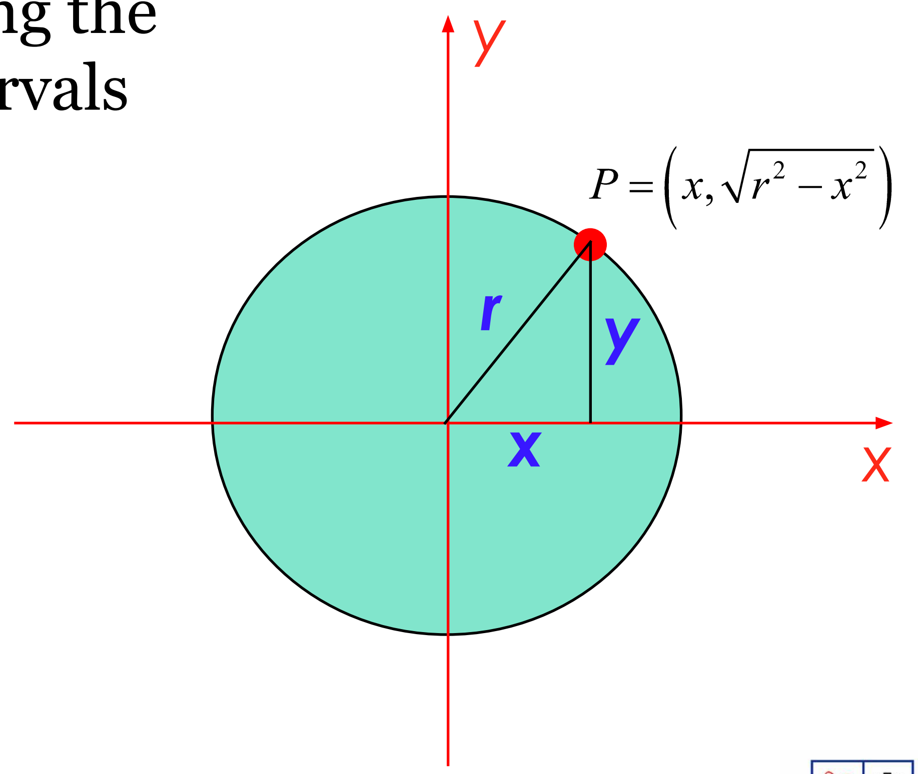
- Use Pythagoras theorem

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

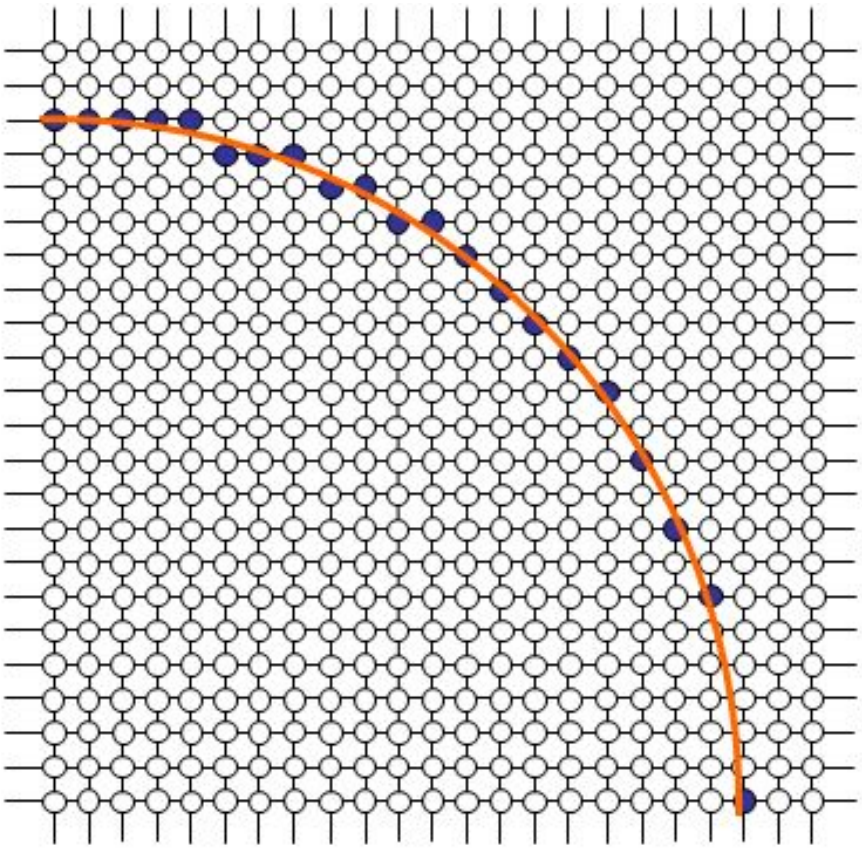


- So, we can write a simple circle drawing algorithm by solving the equation for y at unit x intervals using:

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



A Simple Circle Drawing Algorithm



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

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

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

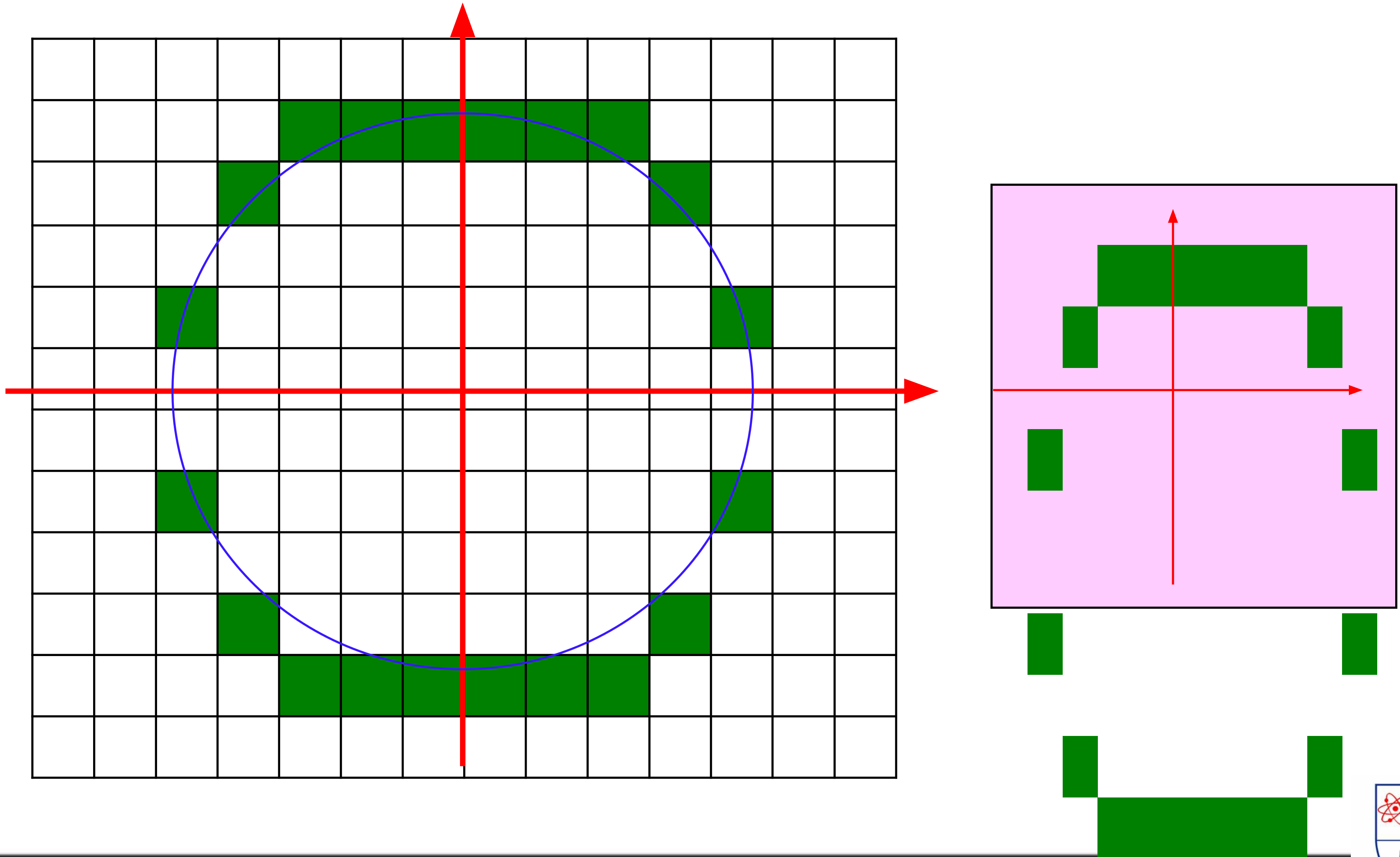
$$\vdots$$

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

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

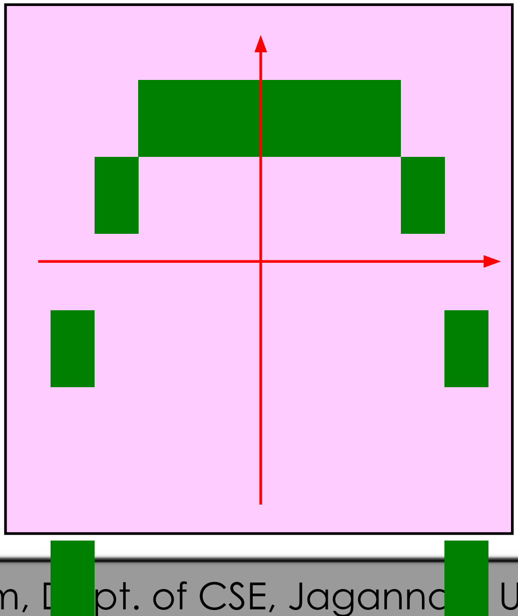
Circle algorithms

- Step through *x*-axis to determine *y*-values



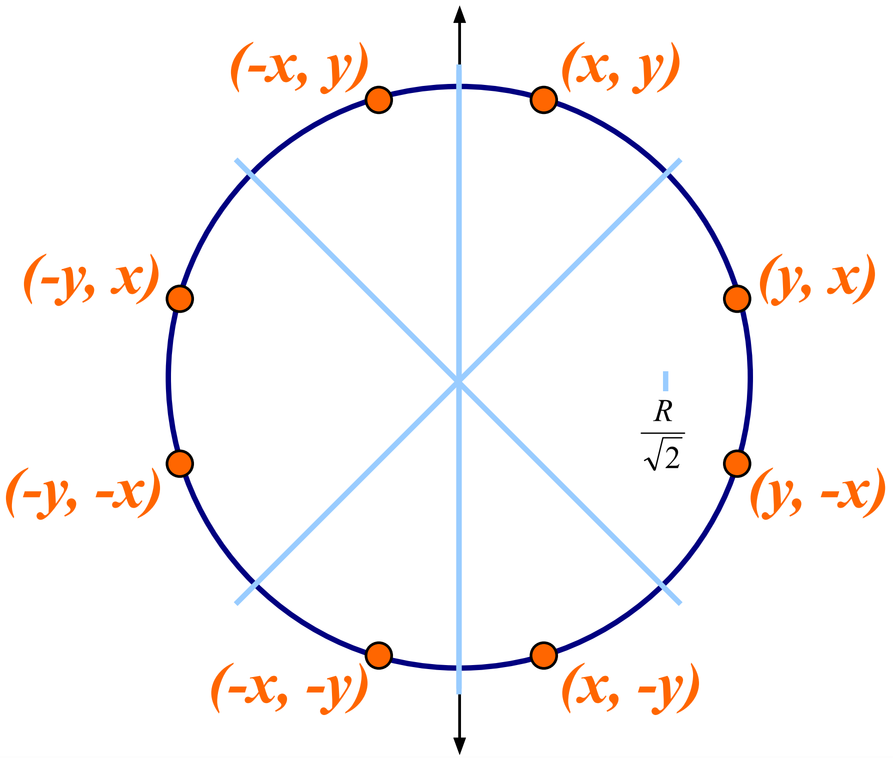
Problems

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



Circle Algorithms

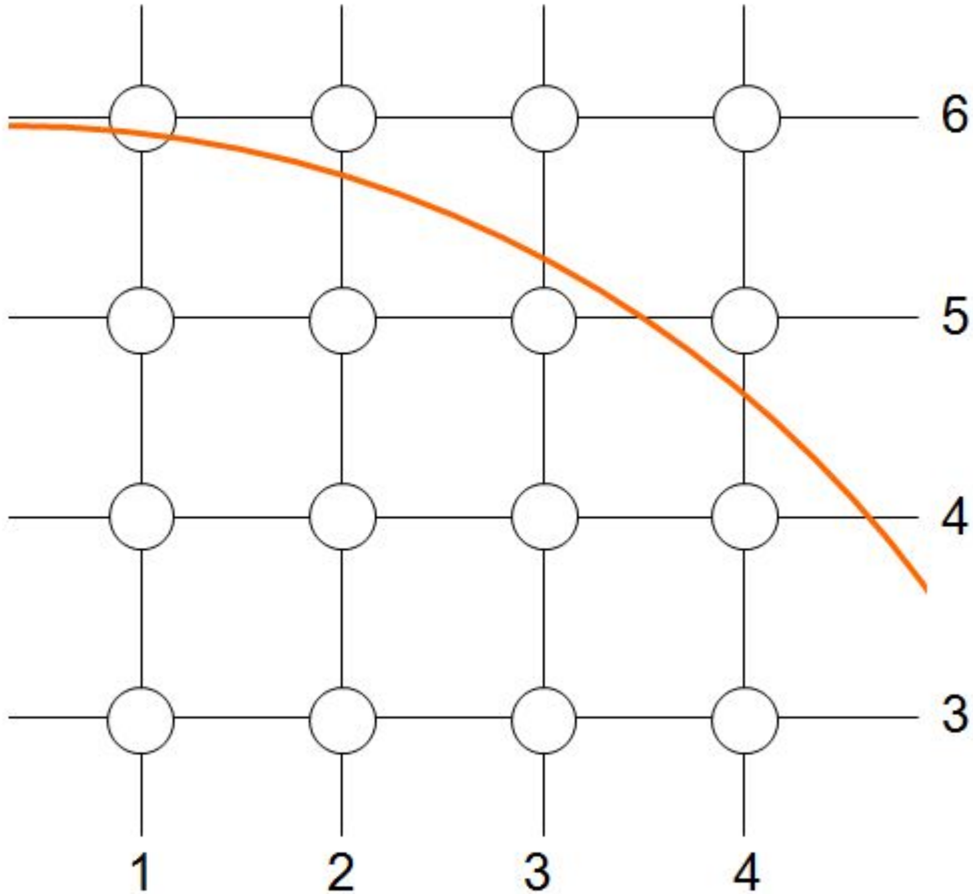
- Use 8-fold symmetry and only compute pixel positions for the 45° sector.
  - The first thing we can notice to make our circle drawing algorithm more efficient is that circles centred at (0, 0) have eight-way symmetry



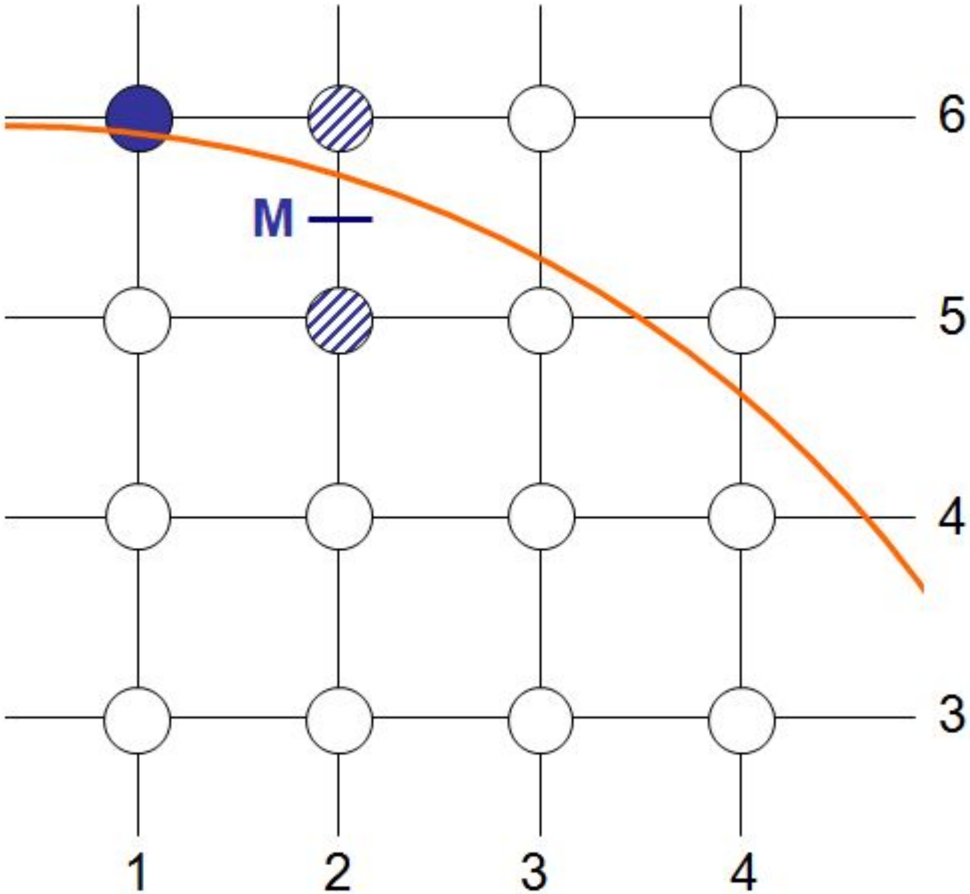
Mid-Point Circle Algorithm

- 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 eight-way symmetry
- so only ever calculate the points for the top right eighth of a circle, and then use symmetry to get the rest of the points

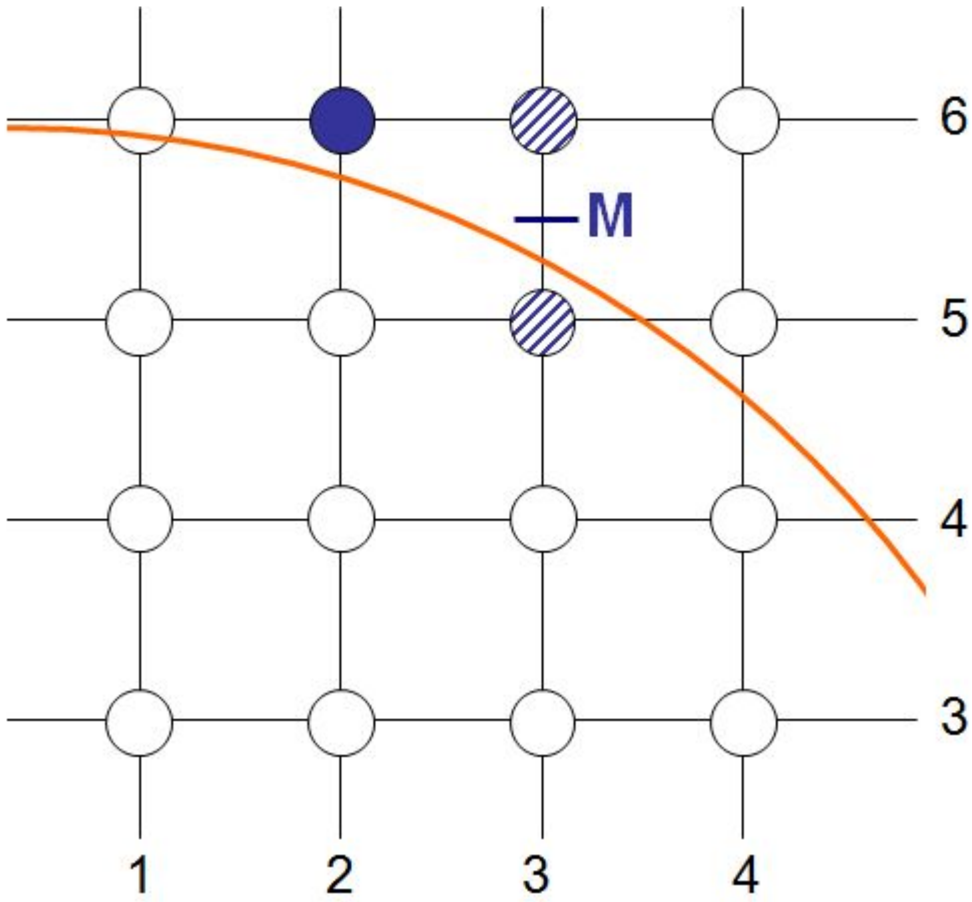
Mid-Point Circle Algorithm



Mid-Point Circle Algorithm



Mid-Point Circle Algorithm



Mid-Point Circle Algorithm

- Let's re-jig the equation of the circle slightly to give us:

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

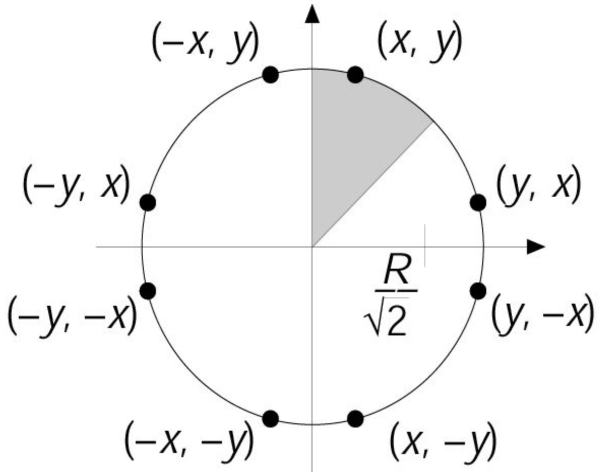
- The equation evaluates as follows:

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

- By evaluating this function at the midpoint between the candidate pixels we can make our decision

Mid-Point Circle Algorithm

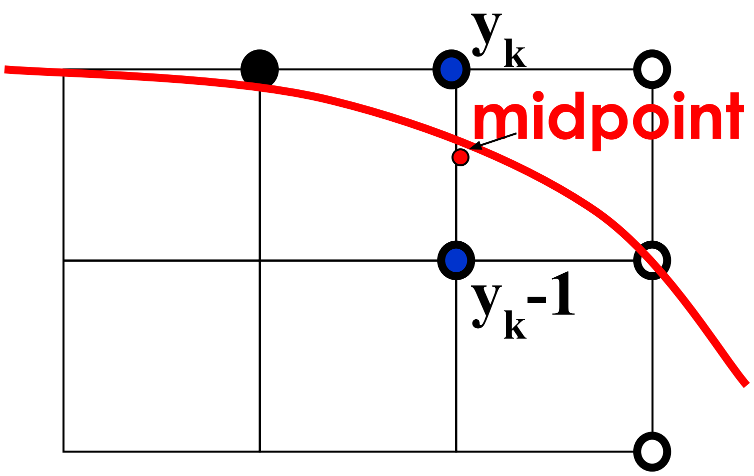
- We need a decision variable D:

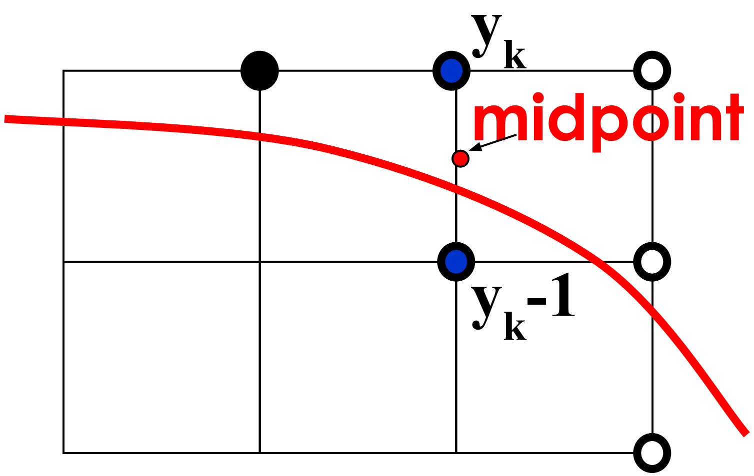


$$
\begin{aligned}
D &= F(M) = F(x_p + 1, y_p - \frac{1}{2}) \\
&= (x_p + 1)^2 + (y_p - \frac{1}{2})^2 - R^2.
\end{aligned}
$$

- If $D < 0$ then M is *below* the arc, hence the E pixel is closer to the line.
- If $D \geq 0$ then M is *above* the arc, hence the SE pixel is closer to the line.

Mid-Point Circle Algorithm





$F_k < 0$

$y_{k+1} = y_k$

Next pixel = (x_k+1, y_k)

$F_k >= 0$

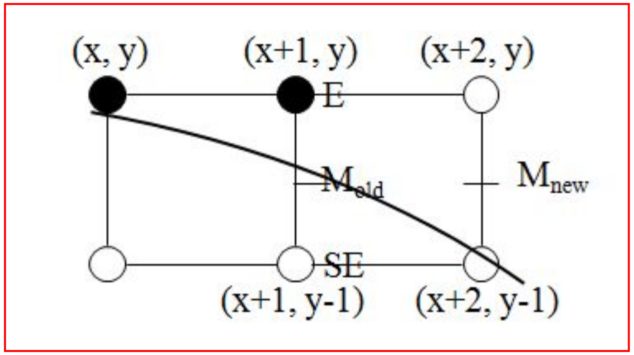
$y_{k+1} = y_k - 1$

Next pixel = (x_k+1, y_k-1)

Mid-Point Circle Algorithm

- What increment for computing a new *D*?
- Next midpoint is: $(x_p + 2, y_p - (1/2))$

$$
\begin{aligned}
D_{new} &= F(x_p + 2, y_p - \frac{1}{2}) \\
&= (x_p + 2)^2 + (y_p - \frac{1}{2})^2 - R^2 \\
&= (x_p^2 + 4x_p + 4) + (y_p - \frac{1}{2})^2 - R^2 \\
&= (x_p^2 + 2x_p + 1) + (2x_p + 3) + (y_p - \frac{1}{2})^2 - R^2 \\
&= (x_p + 1)^2 + (2x_p + 3) + (y_p - \frac{1}{2})^2 - R^2 \\
&= D + (2x_p + 3).
\end{aligned}
$$



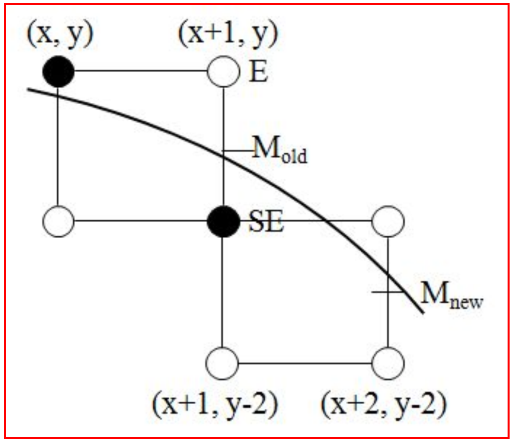
- Hence, increment by: $(2x_p + 3)$

Pics/Math courtesy of Dave Mount @ UMD-CP

Case II: When *SE* is next

- What increment for computing a new *D*?
- Next midpoint is:

$$(x_p + 2,\ \ y_p - 1 - (1/2))$$



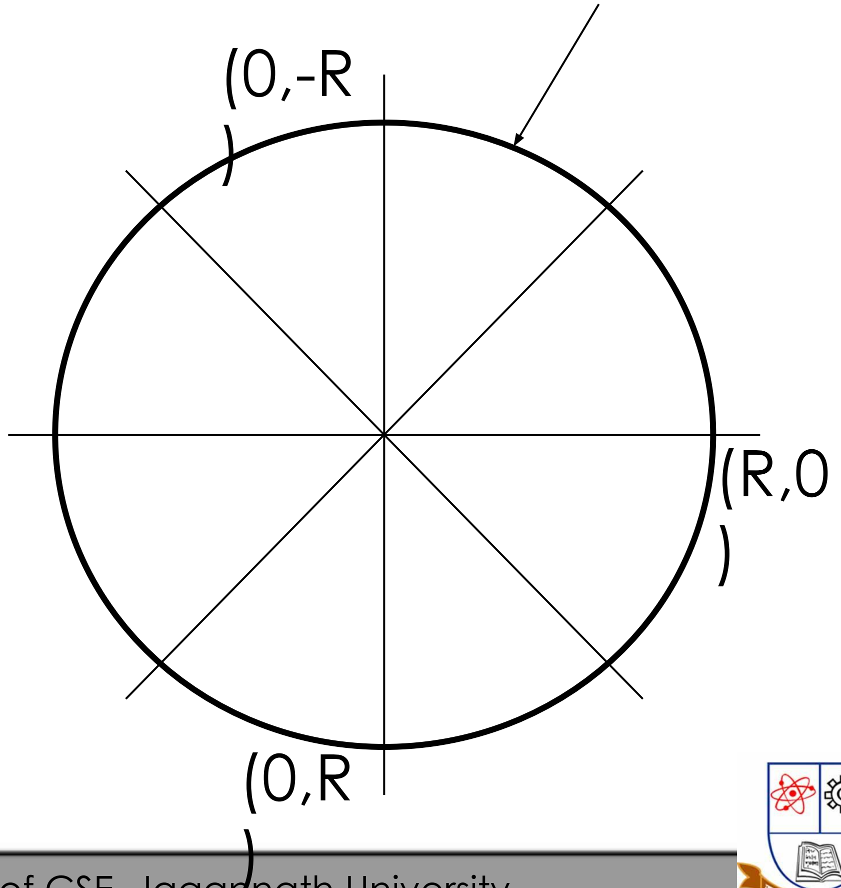
$$\begin{aligned}
D_{new} &= F(x_p + 2, y_p - \frac{3}{2}) \\
&= (x_p + 2)^2 + (y_p - \frac{3}{2})^2 - R^2 \\
&= (x_p^2 + 4x_p + 4) + (y_p^2 - 3y_p + \frac{9}{4}) - R^2 \\
&= (x_p^2 + 2x_p + 1) + (2x_p + 3) + (y_p^2 - y_p + \frac{1}{4}) + (-2y_p + \frac{8}{4}) - R^2 \\
&= (x_p + 1)^2 + (y_p - \frac{1}{2})^2 - R^2 + (2x_p + 3) + (-2y_p + 2) \\
&= D + (2x_p - 2y_p + 5)
\end{aligned}$$

- Hence, increment by: $(2x_p - 2y_p + 5)$

Scan Conversion of Circles

- How to compute the *initial* value of D:
- We start with $x = 0$ and $y = R$, so the first midpoint is at $x = 1$, $y = R\text{-}1/2$:

$$
\begin{aligned}
D_{init} &= F(1, R - \frac{1}{2}) \\
&= 1 + (R - \frac{1}{2})^2 - R^2 \\
&= 1 + R^2 - R + \frac{1}{4} - R^2 \\
&= \frac{5}{4} - R.
\end{aligned}
$$



Algorithm

```
x = 0;
y = -R;
d = 5/4 - R;  /* real */
setPixel(x,y);
while (y > x) {
      if (d > 0) {  /* E chosen */
            d += 2*x + 3;
            x++;
      } else {        /* SE chosen */
            d += 2*(x+y) + 5;
            x++; y++;
      }
      setPixel(x,y);
}
```

Thank You