Graph
Circle Drawing Algorithms

Software Department
Third class

م.م علاء الدين عباس عبد الحسن

Sunday 14/10/2012

Where do we draw a circle?

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

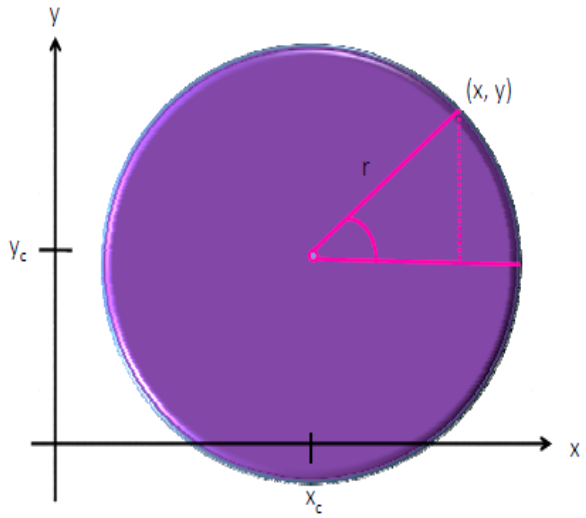
$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

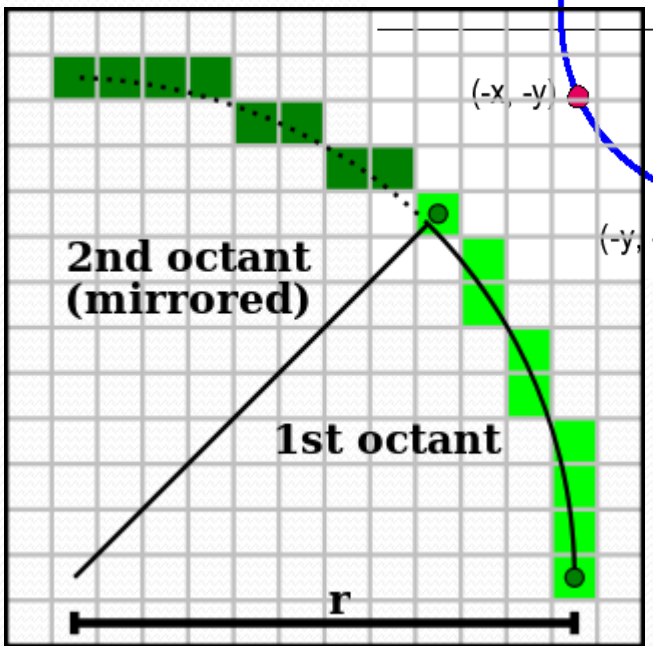
- We could use this equation to calculate the points on the circle circumference by stepping along x-axis in unit steps from $x_c - r$ to $x_c + r$ and calculate the corresponding y values at each position as

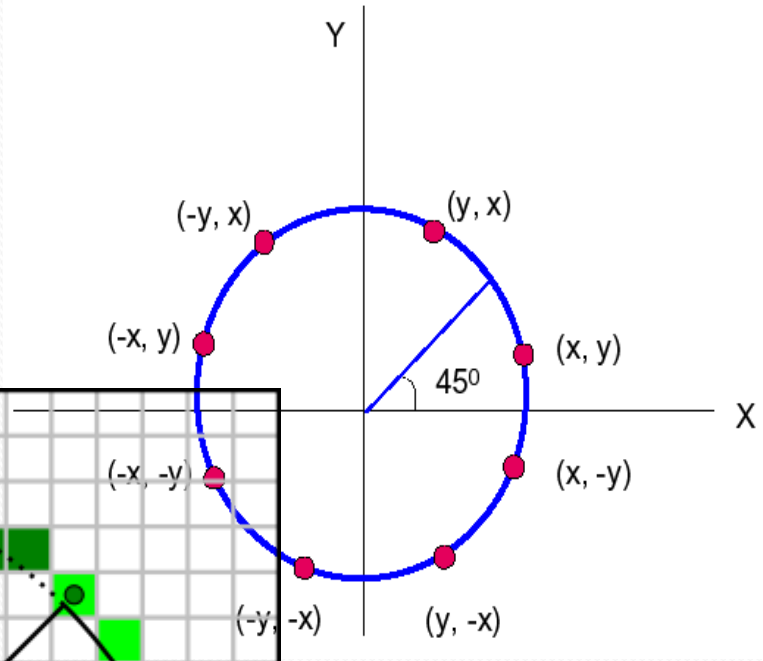
$$y = y_c + (-) (r^2 - (xc - x)^2)^{1/2}$$

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

Circle Drawing Methods







a)Cartesian Coordinates Method

Also known as square root method

- Distance relationship expressed by

the Pythagorean theorem in Cartesian coordinates as:

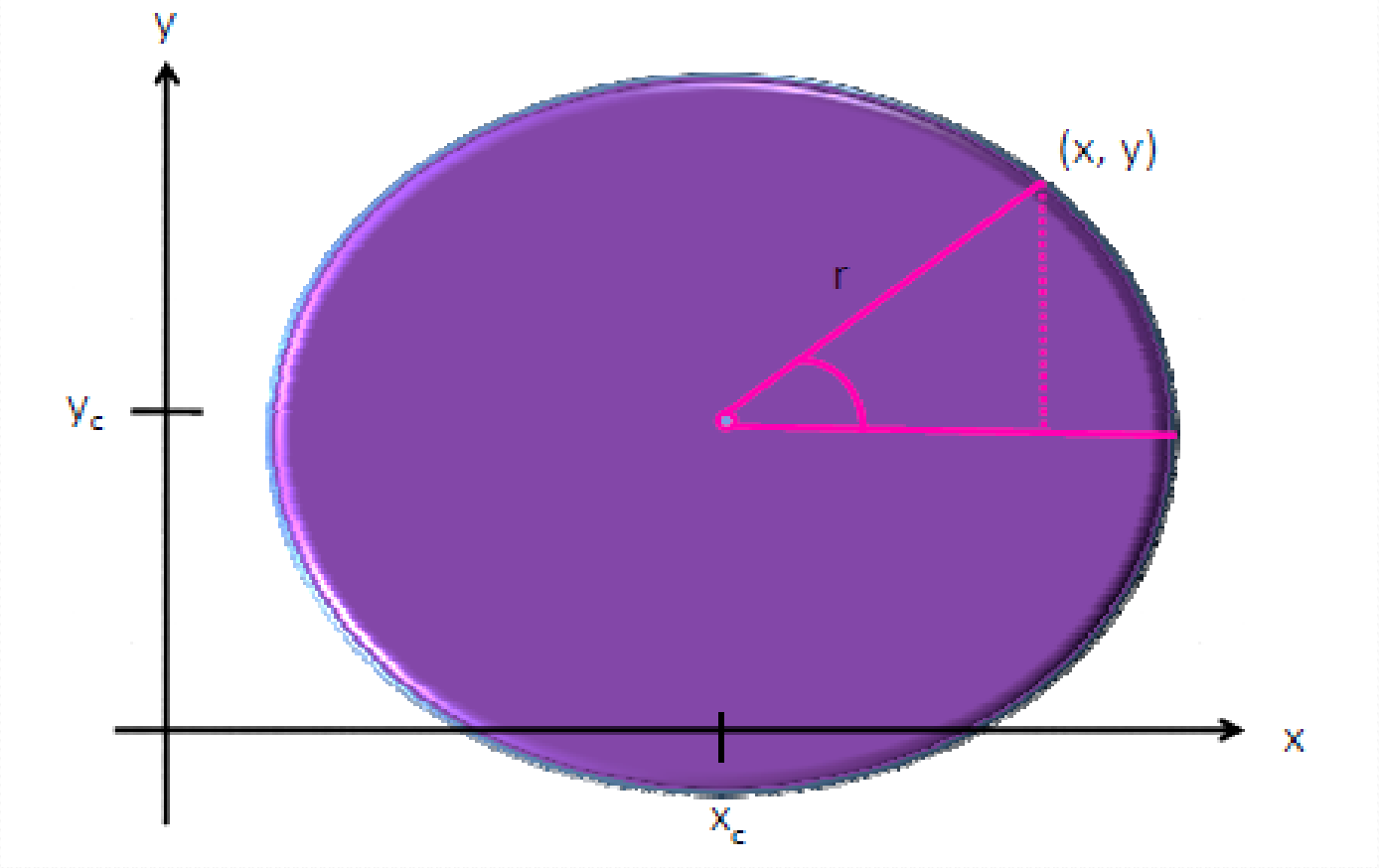
$(x - x_c)^2 + (y - y_c)^2 = r^2$

- So, when stepping along the x axis in

units intervals from xc to xc+ r and calculate the corresponding y coordinate

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

a)Cartesian Coordinates Method



a)Cartesian Coordinates Method

```
Begin
        xc , yc ---→ Circle Center
      r --→ Radius
      x , y ---→ point on circle
    for (x = xc-r to x = xc + r)
begin
          y = Square root((r2-(xc - x)2)
          point(x, yc + y);
          point(x, yc - y);
   end for
End
```

a)Cartesian Coordinates Method

Disadvantages:

- Considerable amount of calculation (squares and square roots)
- Spacing between pixel is not uniform around the circle
- Based on the upper octant (between y axis and 45o line), circles increases faster along x axis compared to y axis. Since we are stepping along x axis, pixels with successive x co-ordinate positions can have the same y coordinate – reduce or no gap
- In the lower octant , the circle increases faster along the y axis compared to x axis. Pixels with successive y co-ordinates need to have the same x co ordinates to avoid gaps. Since , we are using x axis in unit of intervals, so each pixel has a unique y-coordinates – bigger gap.

b) Polar method Enhancement:

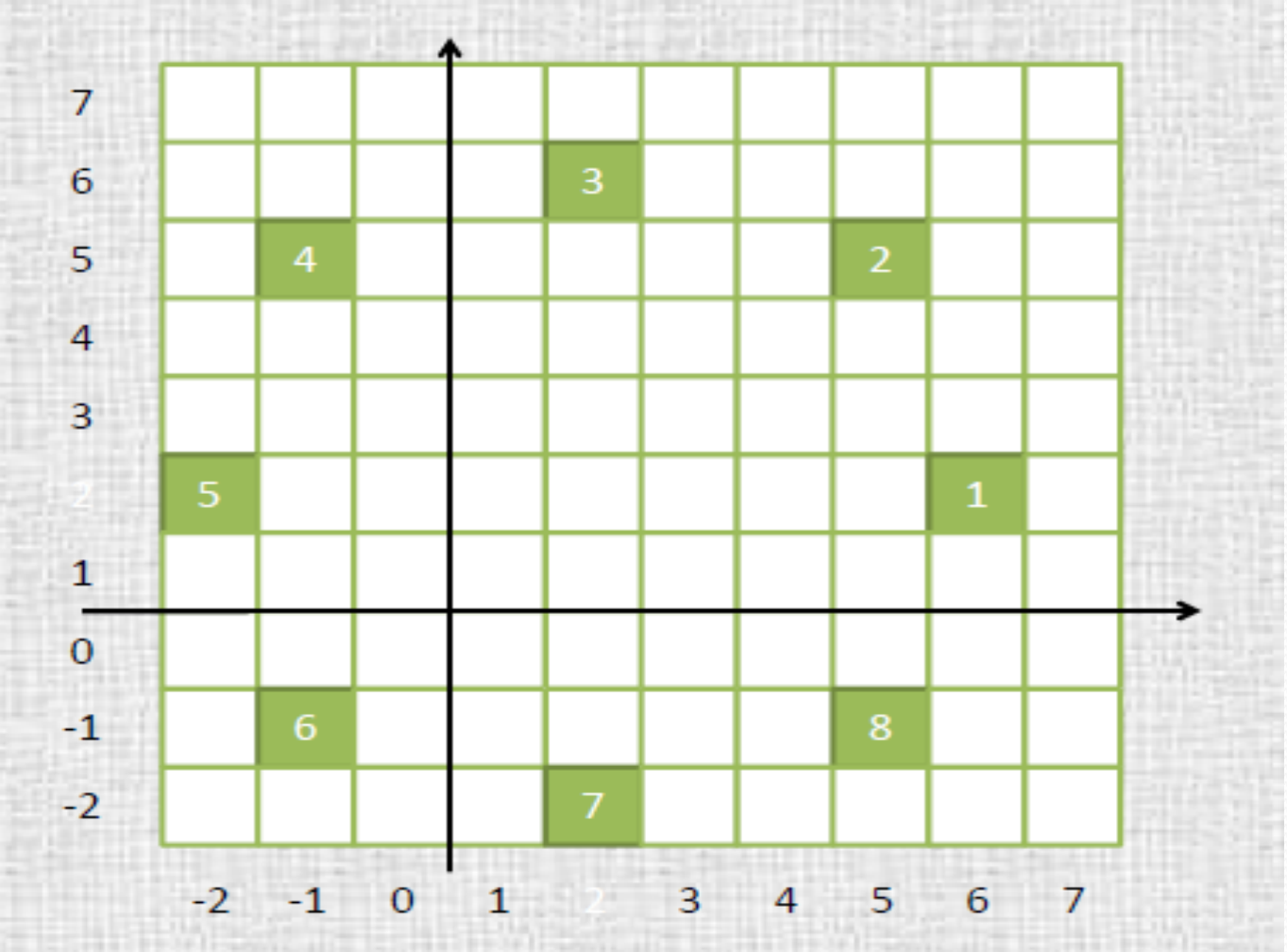
- Also known as Trigonometric Functions
- The equation of a circle is written in the polar coordinates r and θ as

$x = xc + r.\cos\theta$

$y = yc + r.\sin\theta$

- x and y are the coordinates of a point on the circumference, i.e. the end point of a line drawn from the center of the circle to the circumference – the radius.

b) Polar method Enhancement:

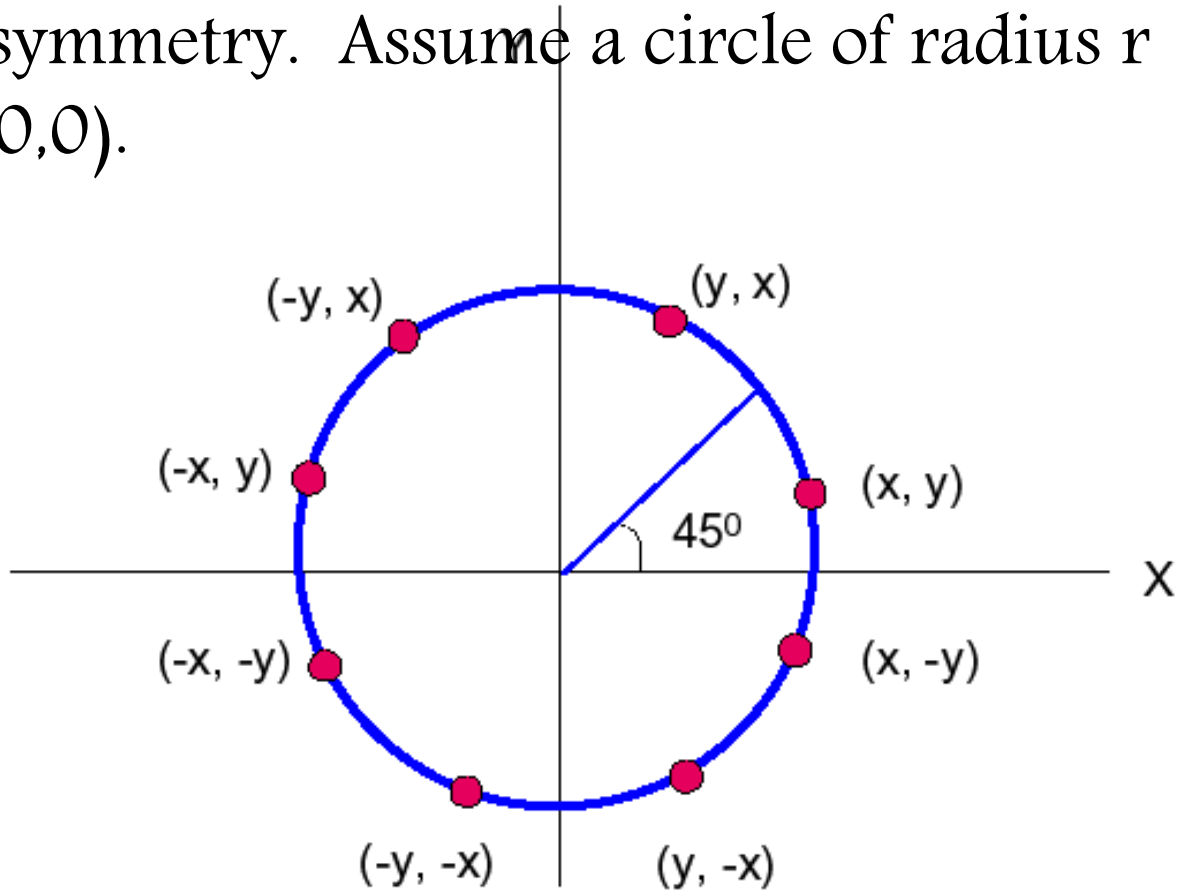


b) Polar method Enhancement:

```
Begin
      xc , yc ---→ Circle Center
     r --→ Radius
     x , y ---→ point on circle
            theta  --→ angle
           dTheta ---→ angle change
     dTheta = 1/r
    for (theta =0 to Pi*2  step = dTheta)
    begin
            x=xc + r * Cos(theta)
            y=yc + r * Sin(theta)
            point(x , y)
    endfor
end
```

c)Polar method speedup (Octanes)

- We only need to calculate the values on the border of the circle in the first octant. The other values may be determined by symmetry. Assume a circle of radius r with center at (0,0).



c) Polar method speedup (Octanes):

```
Begin
        xc , yc ---→ Circle Center
      r --→ Radius
      x , y ---→ point on circle
               theta  --→ angle
              dTheta ---→ angle change
      dTheta = 1/r
            while (x > y)
     begin
               x=xc + r * Cos(theta)
               y=yc + r * Sin(theta)
               point(x ,y) point(x,-y)
               point(-x ,y) point(-x,-y)
               point(y ,x) point(y,-x)
               point(-y ,x) point(-y,-x)
            endwhile
end
```

d)Midpoint circle algorithm

- In computer graphics , the **midpoint circle algorithm** is an algorithm used to determine the points needed for drawing a circle.

d)Midpoint circle algorithm

- **The algorithm**
- he algorithm starts with the circle equation . For simplicity, assume the center of the circle is at . We consider first only the first octant and draw a curve which starts at point and proceeds counterclockwise, reaching the angle of 45.
- The "fast" direction here (the basis vector with the greater increase in value) is the direction. The algorithm always takes a step in the positive direction (upwards), and occasionally takes a step in the "slow" direction (the negative direction).
- From the circle equation we obtain the transformed equation , where is computed only a single time during initialization.

d)Midpoint circle algorithm

- Let the points on the circle be a sequence of coordinates of the vector to the point (in the usual basis). Let denote the point index, with assigned to the point .
- For each point, the following holds:

$$X_n^2 + Y_n^2 = r^2$$

- This can be rearranged as follows:

$$X_n^2 = r^2 - Y_n^2$$

- And likewise for the next point:

$$X_{n+1}^2 = r^2 - Y_{n+1}^2$$

d)Midpoint circle algorithm

- In general, it is true that:

$$Y_{n+1}^2 = (Y_n^2 + 1)^2$$

$$= Y_n^2 + 2 Y_n^2 + 1$$

$$X_{n+1}^2 = r^2 - Y_n^2 + 2 Y_n^2 + 1$$

- So we refashion our next-point-equation into a recursive one by substituting

$$X_n^2 = r^2 - Y_n^2$$

$$X_{n+1}^2 = X_n^2 + 2 Y_n^2 + 1$$

d)Midpoint circle algorithm

- Because of the continuity of a circle and because the maxima along both axes is the same, we know we will not be skipping x points as we advance in the sequence. Usually we will stay on the same x coordinate, and sometimes advance by one.
- Additionally, we need to add the midpoint coordinates when setting a pixel. These frequent integer additions do not limit the performance much, as we can spare those square (root) computations in the inner loop in turn. Again the zero in the transformed circle equation is replaced by the error term.
- The initialization of the error term is derived from an offset of ½ pixel at the start. Until the intersection with the perpendicular line, this leads to an accumulated value of in the error term, so that this value is used for initialization.

d)Midpoint circle algorithm

- The frequent computations of squares in the circle equation, trigonometric expressions and square roots can again be avoided by dissolving everything into single steps and using recursive computation of the quadratic terms from the preceding iterations.
- Below is an implementation of the Bresenham Algorithm for a full circle in C. Here another variable for recursive computation of the quadratic terms is used, which corresponds with the term $2n+1$ above. It just has to be increased by 2 from one step to the next:

d)Midpoint circle algorithm

```
void rasterCircle(int x0, int y0, int radius)
{
  int f = 1 - radius;
  int ddF_x = 1;
  int ddF_y = -2 * radius;
  int x = 0;
  int y = radius;

  setPixel(x0, y0 + radius);
  setPixel(x0, y0 - radius);
  setPixel(x0 + radius, y0);
  setPixel(x0 - radius, y0);
```

d)Midpoint circle algorithm

```
while(x < y)
 {
   // ddF_x == 2 * x + 1;
   // ddF_y == -2 * y;
   // f == x*x + y*y - radius*radius + 2*x - y + 1;
   if(f >= 0)
   {
     y--;
     ddF_y += 2;
     f += ddF_y;
   }
```

d)Midpoint circle algorithm

```
 x++;
    ddF_x += 2;
    f += ddF_x;
    setPixel(x0 + x, y0 + y);
    setPixel(x0 - x, y0 + y);
    setPixel(x0 + x, y0 - y);
    setPixel(x0 - x, y0 - y);
    setPixel(x0 + y, y0 + x);
    setPixel(x0 - y, y0 + x);
    setPixel(x0 + y, y0 - x);
    setPixel(x0 - y, y0 - x);
  }
}
```


THANK YOU