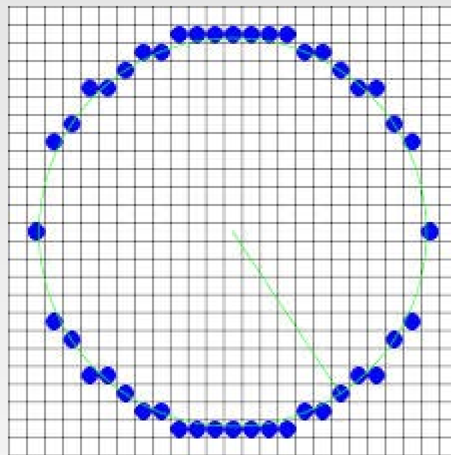
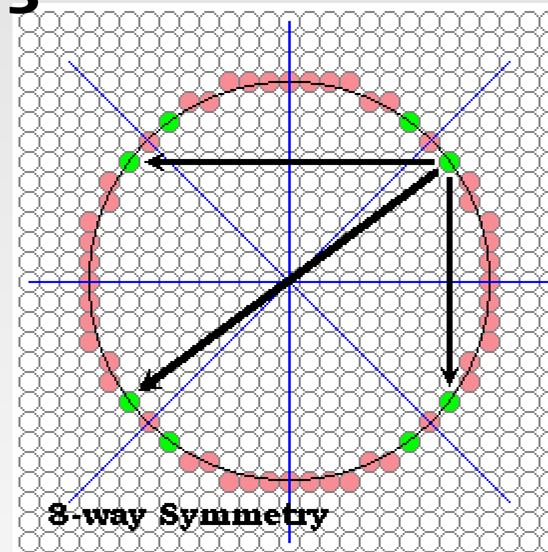
UNIP
UNIVERSIDADE PAULISTA

Computação Gráfica

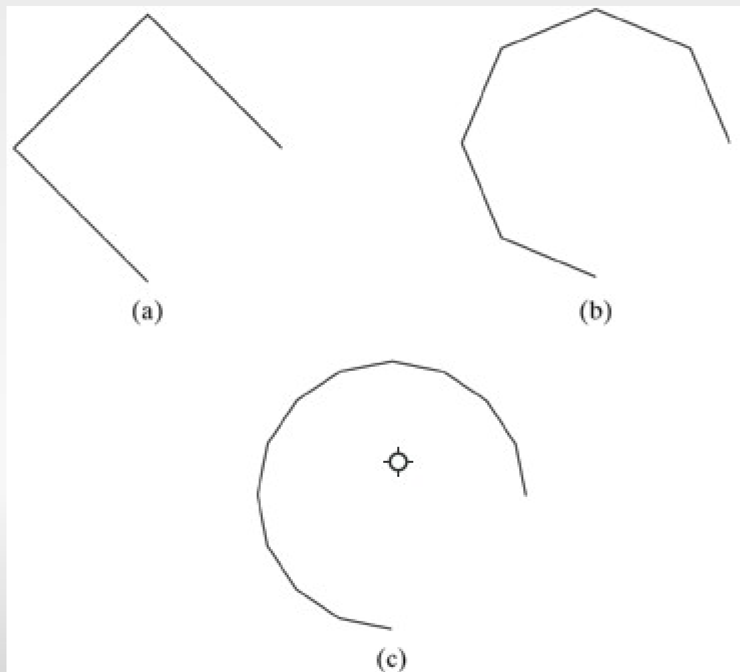




Rasterização de Curvas

Professora Sheila Cáceres

- Podemos representar uma curva por aproximação a uma polilinea.
- Para isso, precisamos localizar alguns pontos no caminho da curva e conectar os pontos com segmentos de reta.
- Para os segmentos de reta podemos usar qualquer algoritmo visto na aula anterior.



Um arco circular aproximado por:
a) 3 segmentos
b) seis segmentos
c) 20 segmentos

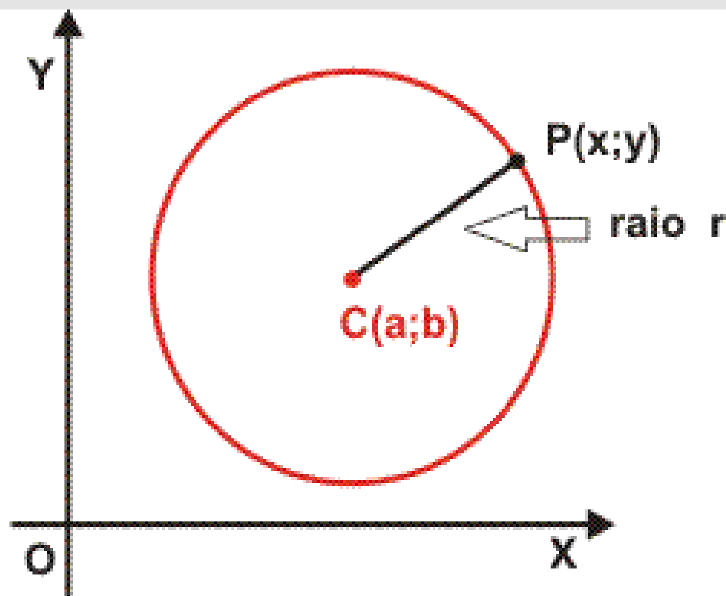
A mais segmentos, teremos uma curva mais suave

Circunferencia

- Uma circunferência é o lugar geométrico dos pontos de um plano que equidistam de um ponto fixo.
- O ponto fixo é o centro e a equidistância é o raio da circunferência.

Equação da Circunferência

- Num sistema de coordenadas cartesianas, uma circunferência pode ser descrita pela equação a seguir.
- Onde: ***a ou xc*** e ***b ou yc*** são as coordenadas do centro da circunferência e ***r*** é o raio.



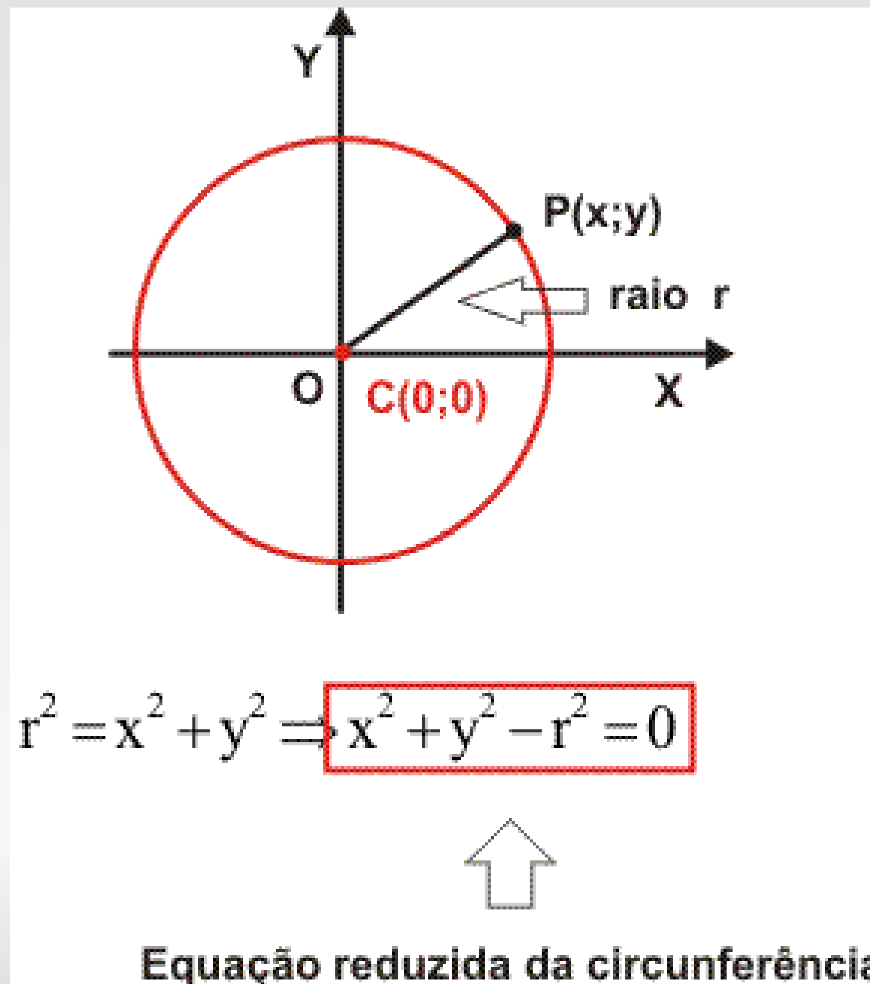
$$r^2 = (x-a)^2 + (y-b)^2 \Rightarrow (x-a)^2 + (y-b)^2 - r^2 = 0 \Rightarrow x^2 + y^2 - 2ax - 2by + a^2 + b^2 - r^2 = 0$$

Equação reduzida da circunferência

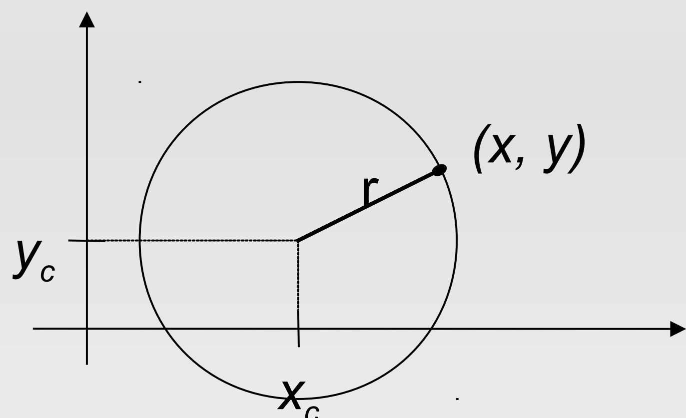
Equação geral da circunferência

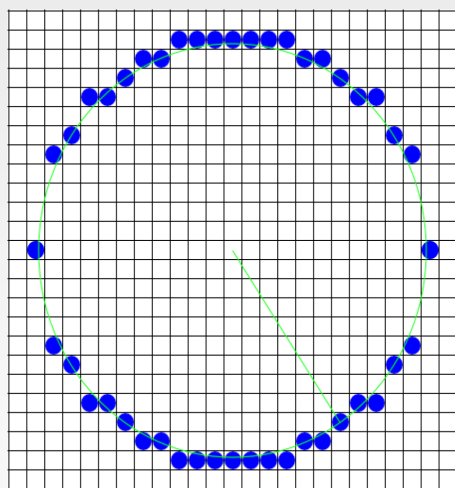
Equação da Circunferência

- Caso a circunferência tenha o centro sobre a origem do plano cartesiano, a equação é:



Equação da circunferência





Teorema de Pitagoras:

$$x^2 + y^2 = r^2 \quad \rightarrow \text{centro } (0,0)$$

Considerando outro centro: x_c, y_c

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

Para grafica-la:

A equação não é conveniente pois precisamos achar os valores de x e y para grafica-los, então:

Para cada x (de 1 em 1) podemos achar o y adequado

$$(x_c - r) \leq x \leq (x_c + r)$$

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

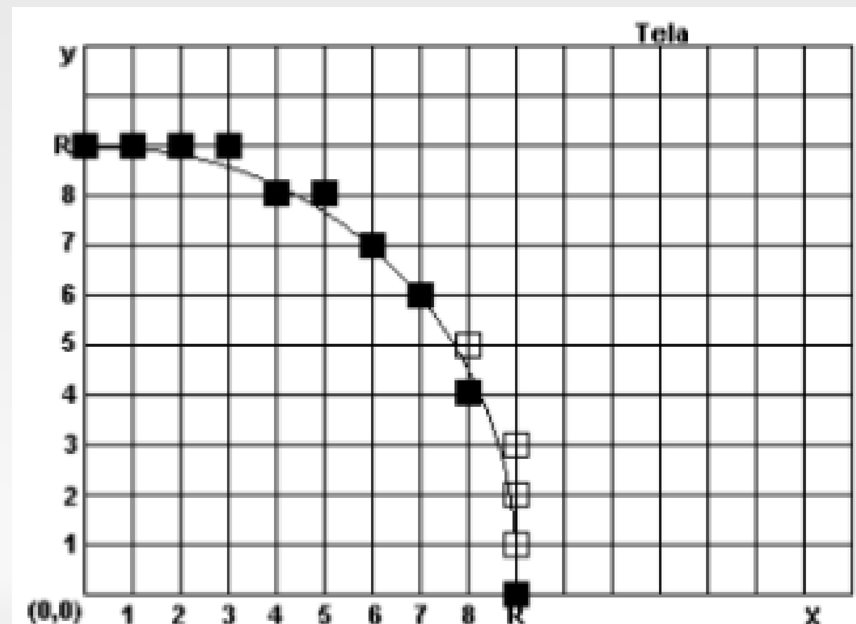
A operação ± é necessaria pois para cada valor de x são obtidos dois valores de y, um para a metade superior e outro para a metade inferior

Algoritmo baseado na Eq. da

```
public void circleSimple(int xCenter, int yCenter, int radius)
{
        int x, y, r2;
        r2 = radius * radius;
        for (x = -radius; x <= radius; x++) {
            y = (int) Math.round(Math.sqrt(r2 - x*x) );
            putPixell( xCenter + x, yCenter + y);
            putPixel( xCenter + x, yCenter - y);
        }
    }
```

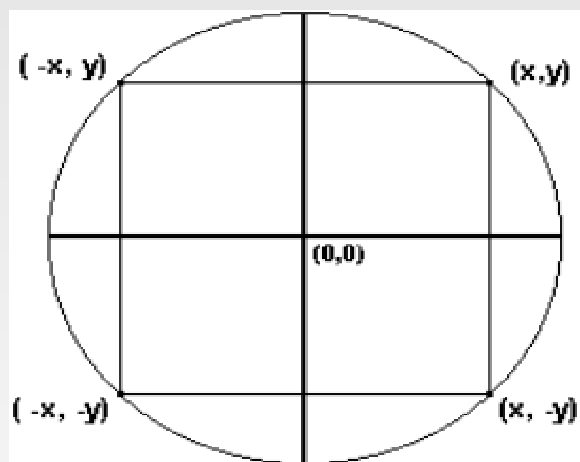
Deficiências

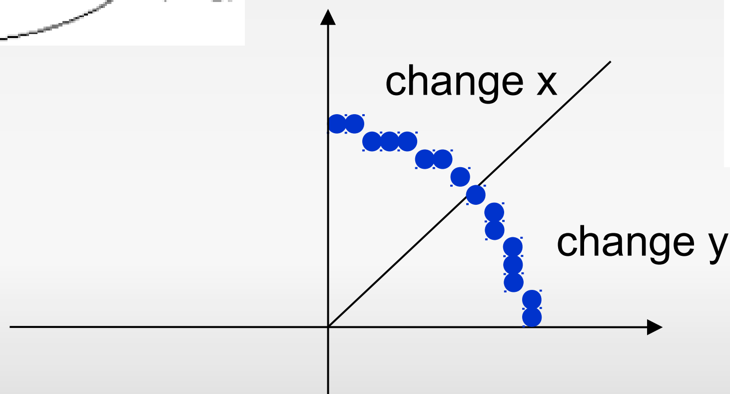
- Ineficiência computacional devido ao cálculo da raiz quadrada e operações de multiplicação.
- O desenho da circunferência fica descontínuo, com largos espaços a medida que a reta tangente a curva tem uma declividade muito alta.

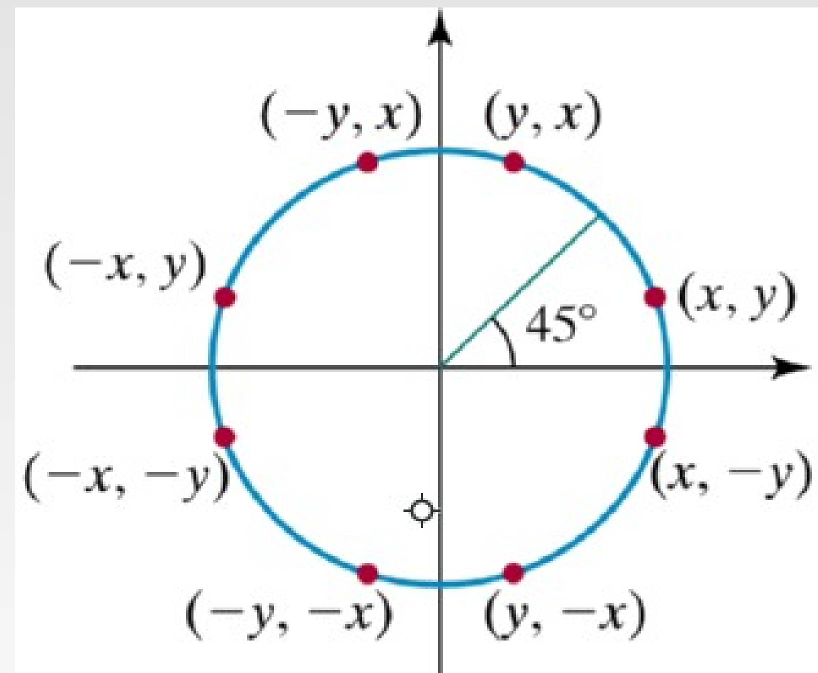


Considerações aplicáveis a vários algoritmos

- Poderiamos computar só os pontos da circunferência para os pontos de um quadrante ou octante e o restante poderia ser plotado por **simetria**.



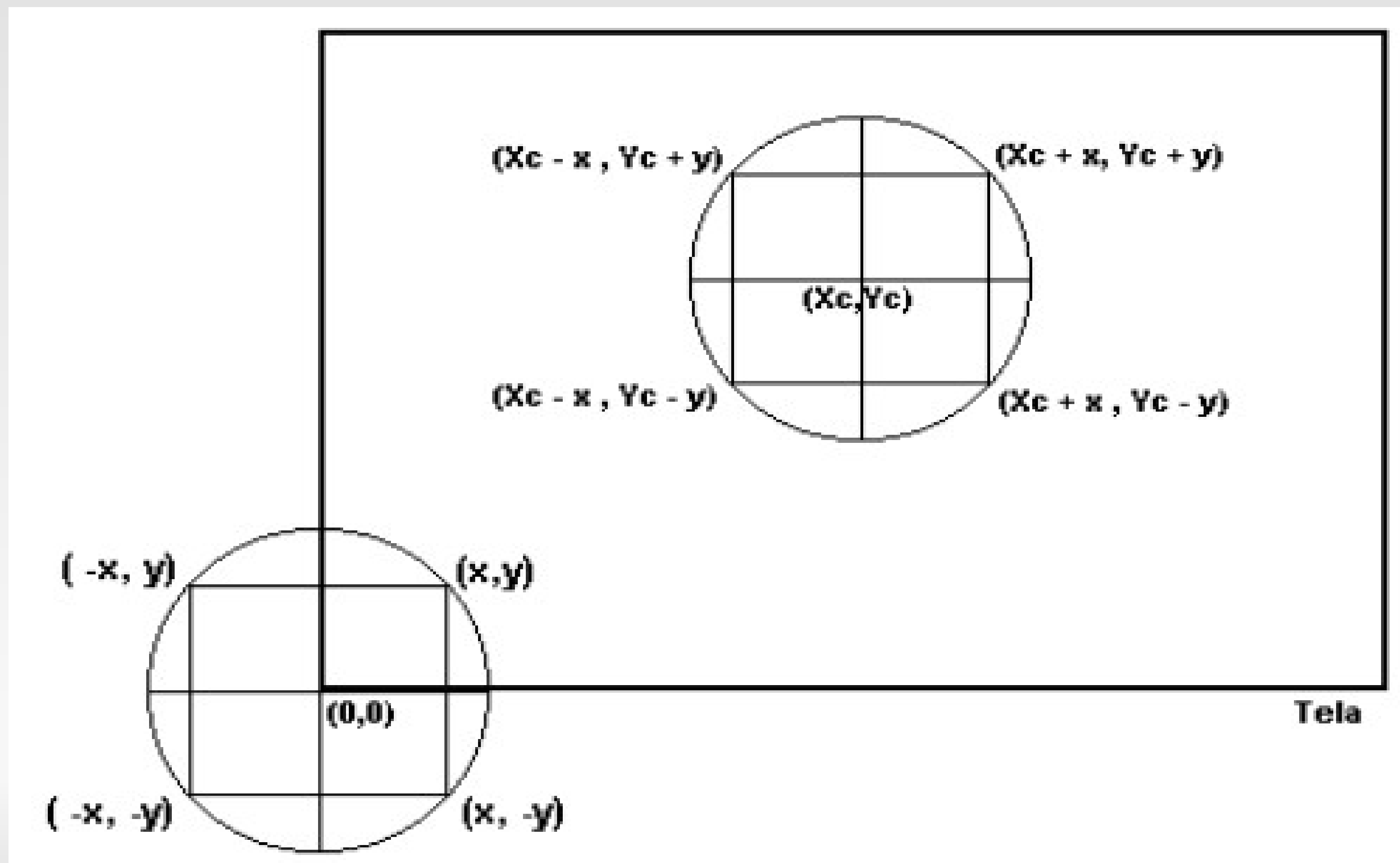




simetrias do círculo: cada ponto calculado define 8 *pixels*

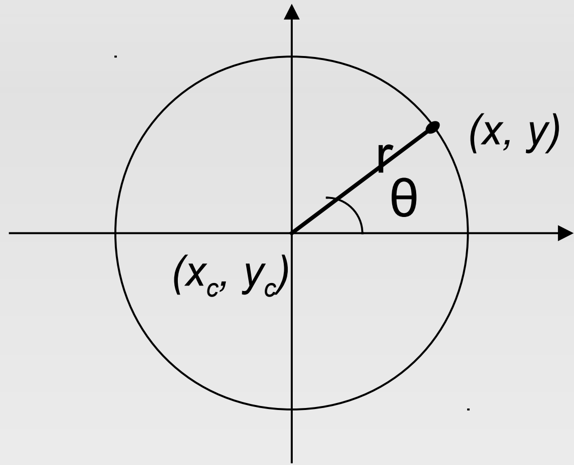
Considerações aplicáveis a vários algoritmos

- **Trasladamos** o ponto achado desde o origem de coordenadas para a posição do centro da circunferencia.



Usando coordenadas Polares

- Para diminuir os gaps gerados pelo algoritmo anterior, podemos graficar a circunferencia usando coordenadas polares.



<u>Equação usando coordenadas polares</u>:

$x = r \,.\, \cos\theta$

$y = r \,.\, \sin\theta$

ambos para centro (0,0)

Considerando outro centro: x_c, y_c

$x = x_c + r \,.\, \cos\theta$

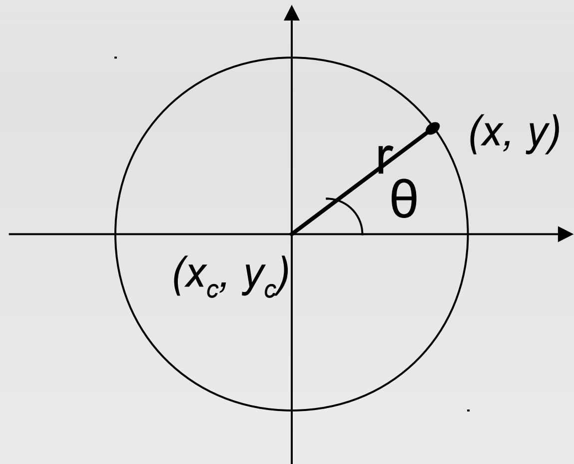
$y = y_c + r \,.\, \sin\theta$

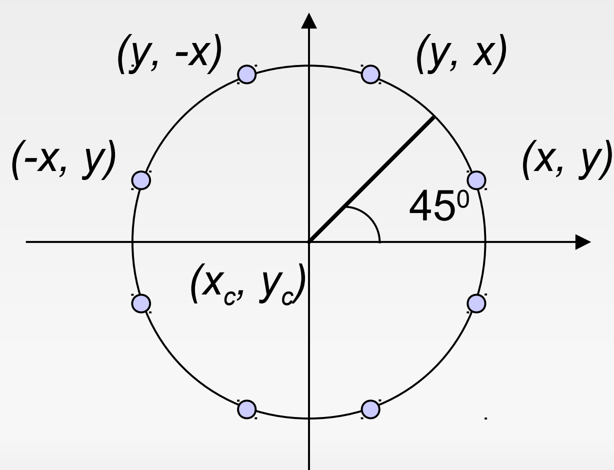
Para grafica-la:

Para cada valor do angulo

Computamos x e y --->

Usando coordenadas Polares





Varie θ com steps de Δθ:

$x = x_c + r \cdot \cos\theta$

$y = y_c + r \cdot \sin\theta$

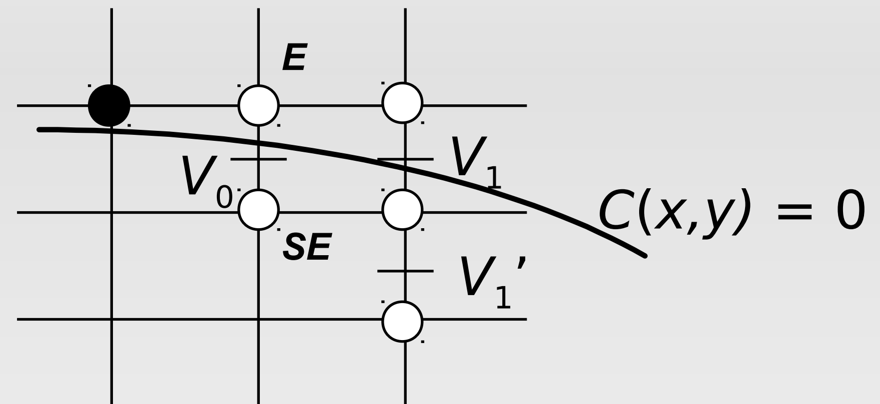
putPixel(x,y)

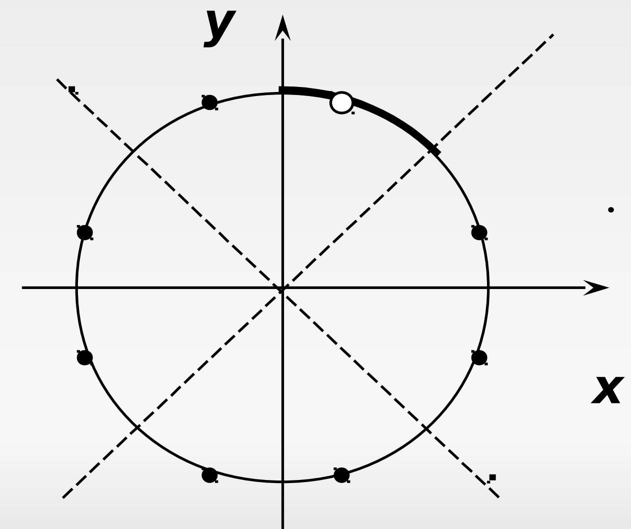
Quanto menor for Δθ menores são os GAPS gerados no desenho e mais perfeita será a circunferência gerada, por outro lado mais tempo se leva para gerá-la.

Para diminuir o tempo use **simetria** se $\theta > 45^0$

Algoritmo do Ponto Medio

- Avalia incrementalmente uma função que classifica o ponto médio entre um pixel e outro com relação a uma função implícita
- Apenas um octante precisa ser avaliado, os demais são simétricos
  - Para cada pixel computado, oito são pintados
- Outras cônicas podem também ser rasterizadas de forma semelhante



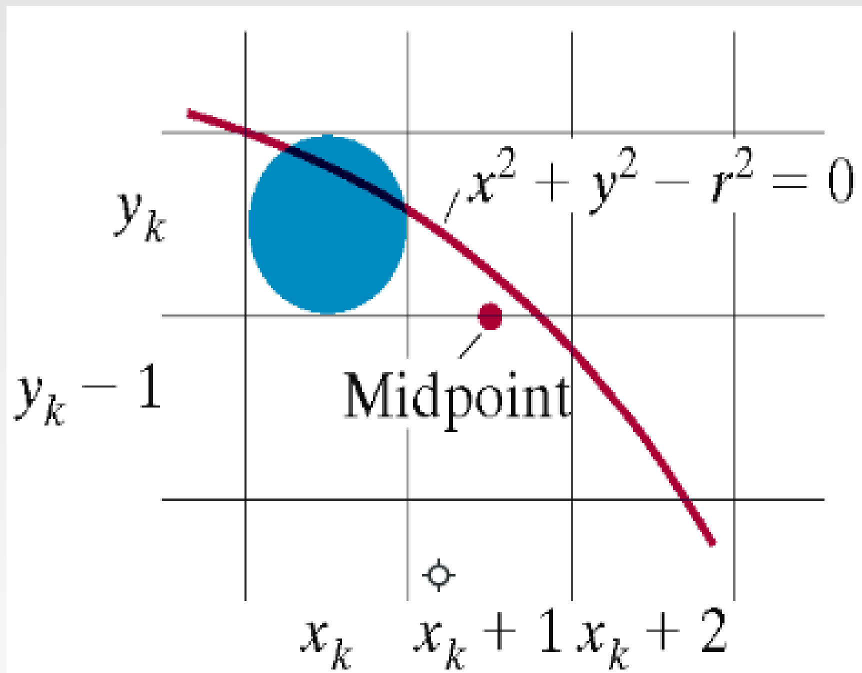


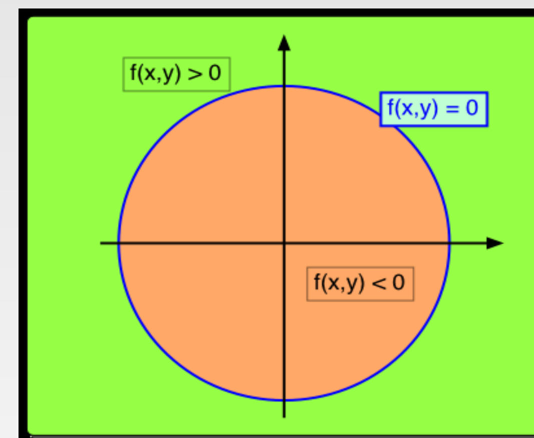
Algoritmo do Ponto Medio

- Variante do algoritmo de Bresenham para retas e portanto tmb é chamado: Alg. Bresenham para Circunferencia.

Definimos uma função do círculo

$f(x,y) = x^2 + y^2 - r^2$





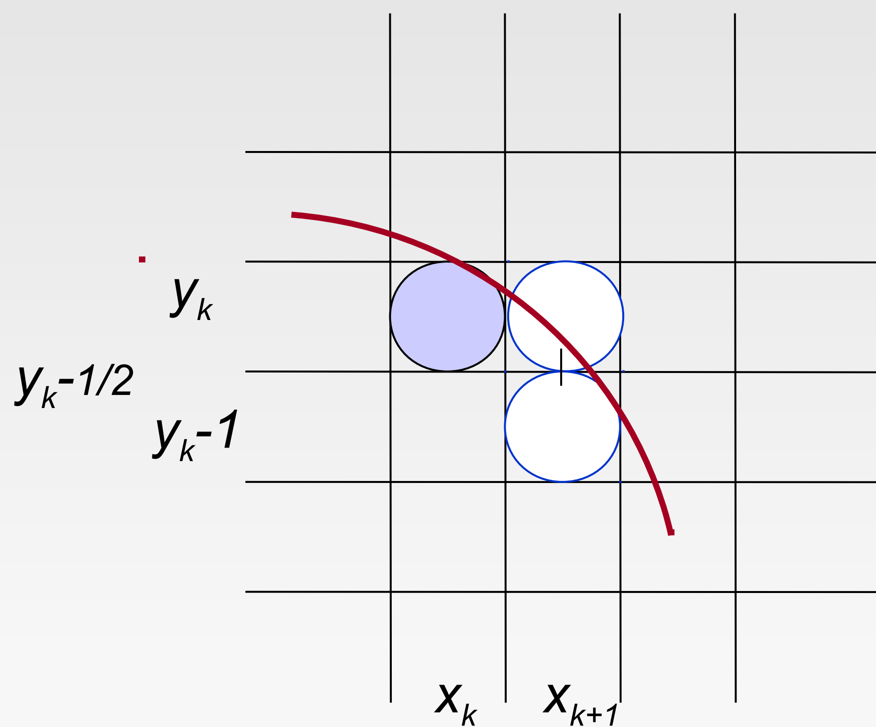
Onde:

$$f(x,y) \begin{cases} <0 \text{ se } (x,y) \text{ está dentro do circulo} \\ =0 \text{ se } (x,y) \text{ está no círculo} \\ >0 \text{ se } (x,y) \text{ está fora do circulo} \end{cases}$$

Achamos f só para um octante e o restante por simetria

Algoritmo do Ponto Medio

A cada passo determina o próximo melhor pixel baseado em quão próxima esta a verdadeira curva dos dois pixels possíveis.



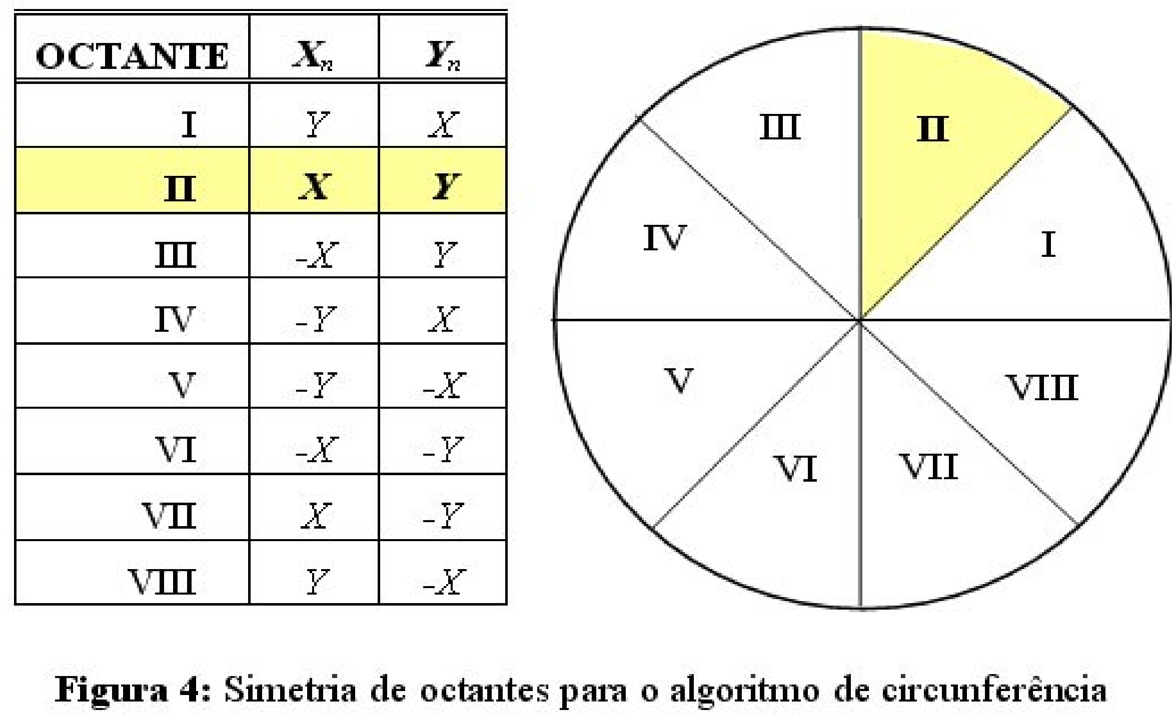
Similar ao Algoritmo para rectas, consideraremos um parametro de decisão **p** (baseado em f) evaluado no ponto medio entre os dois pixels possíveis a cada passo.

- $p_k = f(x_k + 1, y_k - ½) = (x_k + 1)^2 + (y_k - ½)^2 - r^2$

p_k {
<0 se (x,y) está dentro do circulo –> escolher y_k
>=0 se (x,y) está fora do circulo –> escolher y_k -1

- $p_0 = 1 - r$
- A cada passo:
  - $p_{k+1} = p_k + 2x_{k+1} + 1$ (when $p_k < 0$)
  - $p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$ (otherwise)

- Escolhemos o segundo octante → curva suave sem GAPS



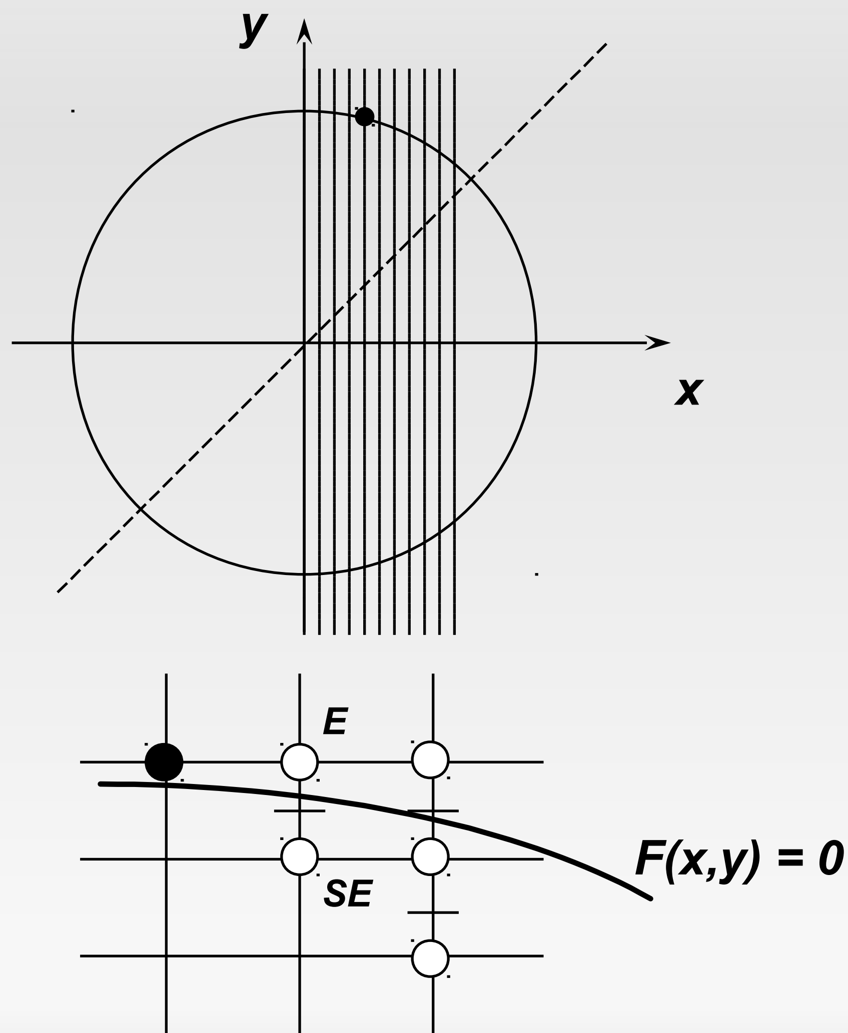
OCTANTE	X_n	Y_n
I	Y	X
II	X	Y
III	$-X$	Y
IV	$-Y$	X
V	$-Y$	$-X$
VI	$-X$	$-Y$
VII	X	$-Y$
VIII	Y	$-X$

Figura 4: Simetria de octantes para o algoritmo de circunferência

- Os outros octantes por simetria

Algoritmo do ponto Médio

- Pseudocódigo




```
y=raio;
for (x=0; x< y; x++) {
      if p_k <0
         y se mantem (Este)
          p += 2 * x + 1;
     else
         y – (SulEste)
         p += 2 * (x - y) + 1;
     putPixel(x,y)
     //pinte os simétricos
      …
}
```

Algoritmo do ponto Médio

```
void drawCircleMidpoint (Graphics g, int xc,int yc,int radius){
    int p = 1 - radius;
    int y = radius;

     // Calcula os proximos pontos e os plota em cada octante
    for (int x = 0;x <= y;x++) {
        drawCircleSimetricPoints(g, xc,yc,x,y);
        if (p < 0)
            p += 2 * x + 1;
        else {
            y--;
            p += 2 * (x - y) + 1;
        }
    }
}

void drawCircleSimetricPoints (Graphics g, int xc, int yc, int x, int y){
   putPixel (g, xc + x, yc + y);
   putPixel (g, xc - x, yc + y);
   putPixel (g,xc + x, yc - y);
   putPixel (g,xc - x, yc - y);
   putPixel (g,xc + y, yc + x);
   putPixel (g,xc - y, yc + x);
   putPixel (g,xc + y, yc - x);
   putPixel (g,xc - y, yc - x);
}
```

Referências

- Hearn Baker, Computer Graphics with OpenGL, 3 edition.
- Fundamentos da Computação Gráfica, J. Gomes e L. Velho, IMPA, 2003
- Material do Prof. Ismael H F Santos, UniverCidade
- Computação gráfica para programadores Java, Ammeraal e Zhang, segunda edição.
- Computação Gráfica, Eduardo Azevedo e Aura Conci.
- Prof. Humberto B. Pinheiro
- Material Unip.

- Nota: Algumas imagens e formulas da apresentação foram extraidas das fontes aqui apresentadas.