Aula 08

Traçado de Circunferências usando Bresenham

Algoritmo de Bresenham - Traçado de Retas

Este algoritmo, de 1965, permite o traçado de circunferências com vantagens importantes:

- Não usa operações em ponto flutuante, tornando mais rápida a sua execução

- Usa apenas operações de soma e subtração e multiplicações por dois, que podem ser substituídas por deslocamentos de bits

- Pode ser implementado em hardware simples ou em assembly de processadores mais básicos (anos 70)

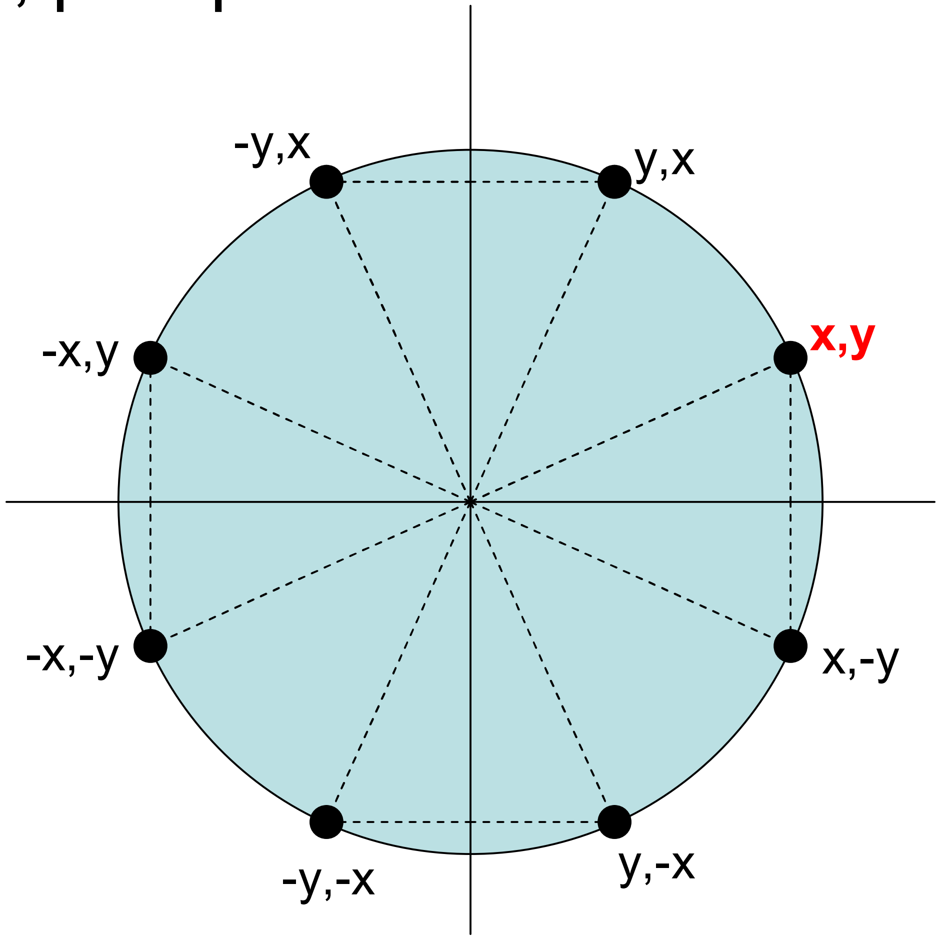
- **Não deixa buracos entre os pixels da circunferência**

- **Nenhum pixel é acessado mais de uma vez**

Algoritmo de Bresenham – Circunferências

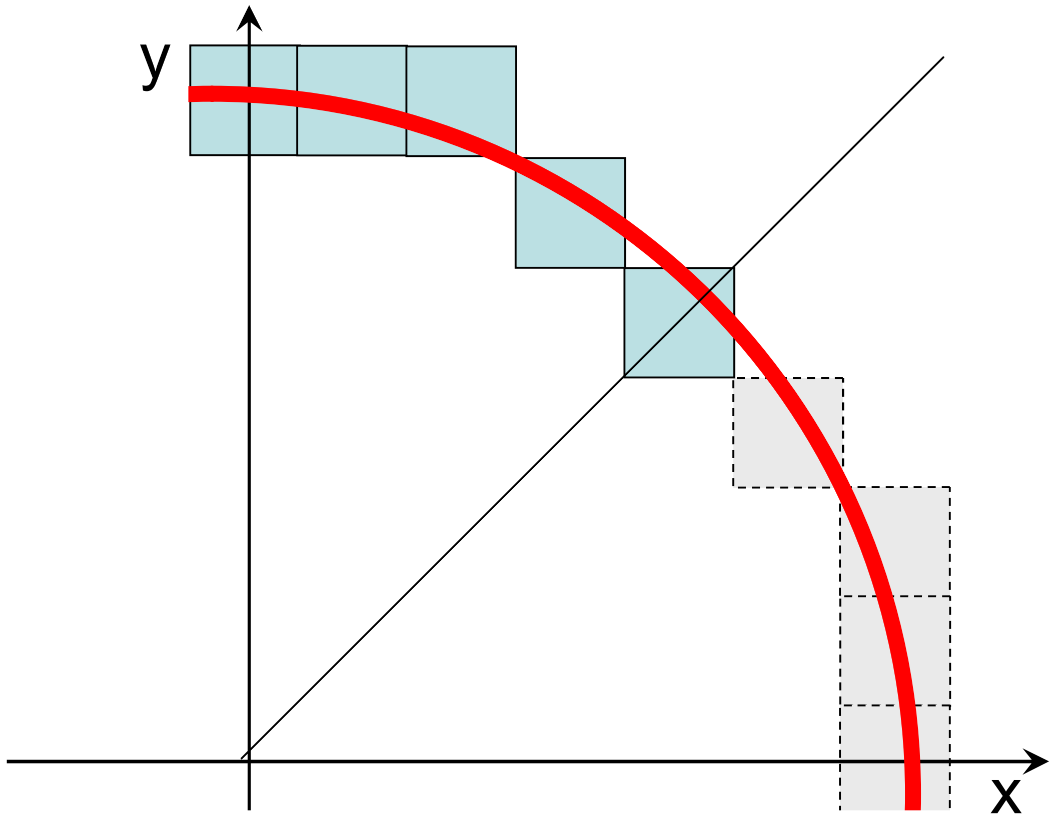
É importante lembrar que a circunferência é um objeto simétrico, assim, os cálculos em seu traçado podem ser bem reduzidos, pois pode se calcular uma parte dos pontos e, espelhar os demais

Por exemplo, basta calcular os pontos apenas entre 0 e 45º e espelhar eles em outros sete



Algoritmo de Bresenham – Circunferências

Assume que a circunferência está no segundo octante



Neste octante, x sempre deve ser incrementado de um em um e, o y será decrementado ou não

Algoritmo de Bresenham – Circunferências

A equação de uma circunferência com raio R,

centrada na origem é $\mathbf{x^2 + y^2 = R^2}$

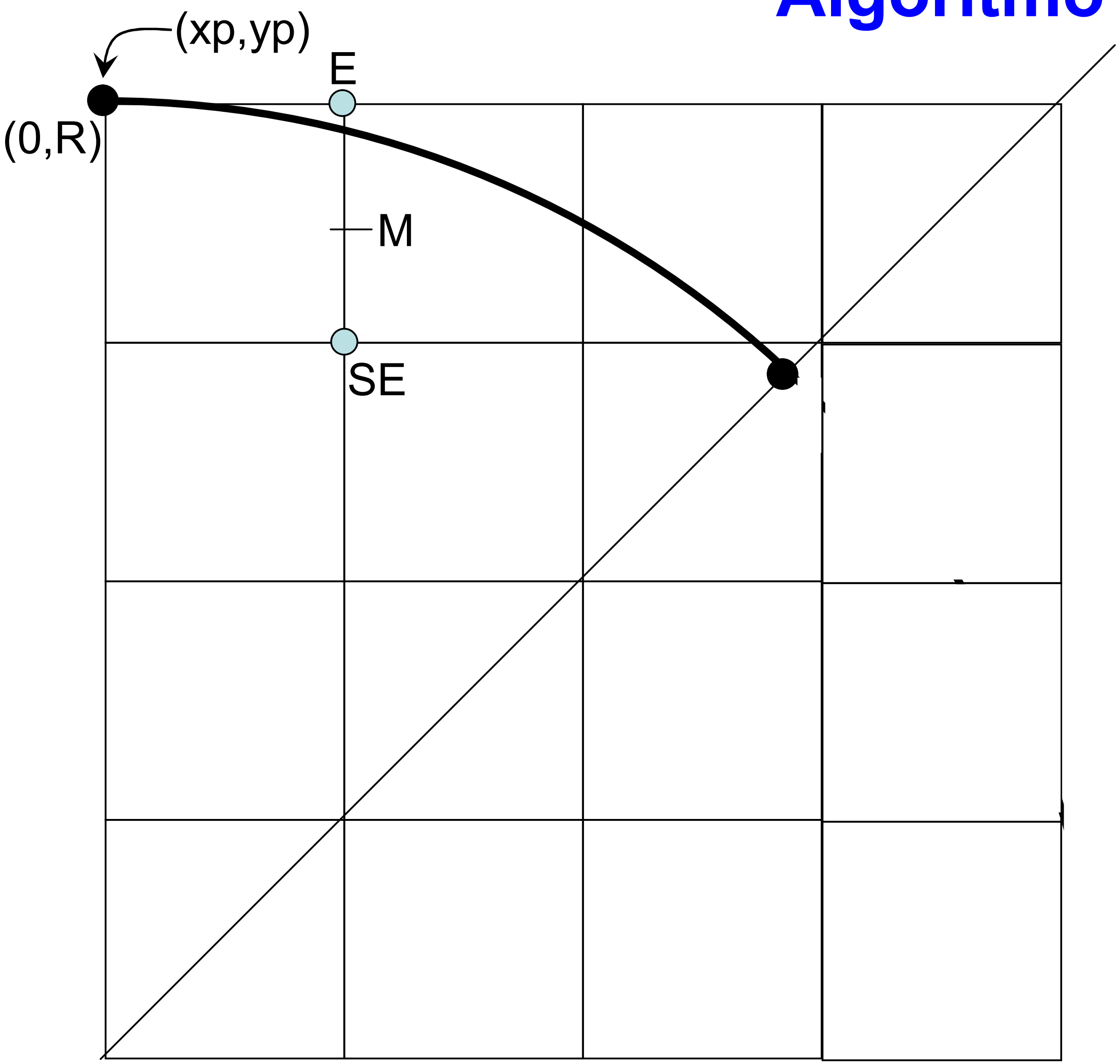
e a sua equação geral é $\mathbf{f(x,y) = x^2 + y^2 - R^2 = 0}$

Sabe-se que dado um ponto (x,y), se:

$$\begin{cases} f(x,y)=0\text{, então, o ponto está sobre a circunferência} \\ f(x,y)<0\text{, então, o ponto está dentro da circunferência} \\ f(x,y)>0\text{, então, o ponto está fora da circunferência} \end{cases}$$

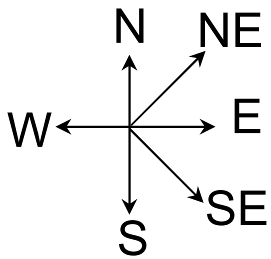
Algoritmo de Bresenham – Circunferências

Algoritmo do ponto Médio



As decisões são tomadas na grade

Quando M está dentro da circunferência, escolhe-se o ponto E, caso contrário, escolhe-se SE



Algoritmo de Bresenham – Circunferências

Considerando estar em xp,yp, o ponto M será:

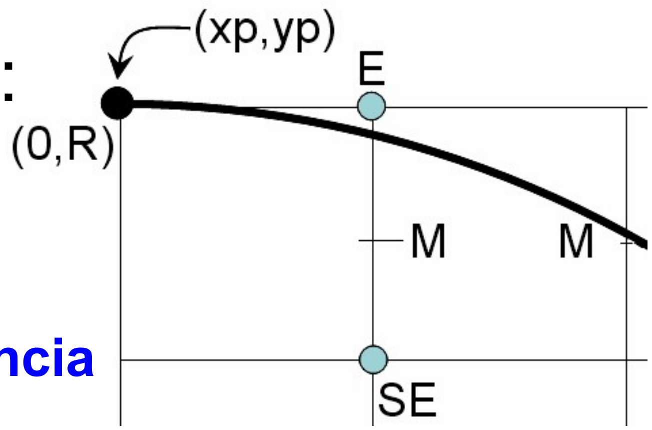
$M = (xp+1, yp-1/2)$

$$\boxed{f(x,y) = x^2 + y^2 - R^2 = 0}$$

Usa-se uma variável de decisão

$$d = f(M) = (xp+1)^2 + (yp-1/2)^2 – R^2$$

se **<u>d < 0</u>**, escolhe-se E e o próximo ponto M será incrementado apenas em x, ficando:



$$M = (xp+2, yp-1/2)$$

f(x,y)<0, então, o ponto está dentro da circunferência

Algoritmo de Bresenham – Circunferências

Assim, $d_{novo} = f(M) = (xp+2)^2 + (yp-1/2)^2 - R^2$

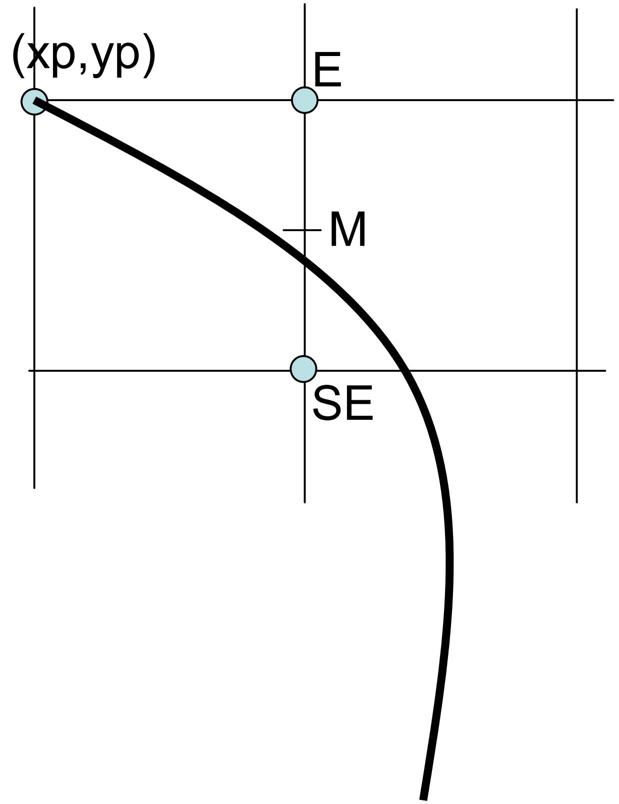
Usando o cálculo incremental, é possível obter d_{novo} a partir de d, pois:

$$
\begin{array}{rl}
d_{novo} - d = & (xp+2)^2 + (yp-1/2)^2 - R^2 \\
 & \underline{(xp+1)^2 + (yp-1/2)^2 - R^2} \quad \textbf{(-)} \\
 & 2xp+3
\end{array}
$$

Logo, $d_{novo} = d + 2xp + 3$

Este é o valor que deve ser somado à d, quando se escolhe **E**, ou seja, $\mathbf{\Delta E = 2xp + 3}$

Algoritmo de Bresenham – Circunferências



Se M está fora da circunferência

<u>**$d \geq 0$**</u>, escolhe-se SE e o

próximo ponto M será incrementado

em x e decrementado em y, ficando:

$$M = (xp+2, yp-3/2)$$

Neste caso, $d_{novo} = f(M) = (xp+2)^2 + (yp-3/2)^2 - R^2$

Algoritmo de Bresenham – Circunferências

Usando o cálculo incremental, é possível obter d_{novo} a partir de d, pois:

$$
\begin{array}{rl}
d_{novo} - d = & (xp+2)^2 + (yp-3/2)^2 - R^2 \\
 & \underline{(xp+1)^2 + (yp-1/2)^2 - R^2} \quad \color{red}{\mathbf{(-)}} \\
 & 2xp - 2yp + 5
\end{array}
$$

Logo, $d_{novo} = d + 2xp - 2yp + 5$

Este é o valor que deve ser somado à d, quando se escolhe **SE**, ou seja, **$\Delta SE = 2xp - 2yp + 5$**

ou ainda, **$\Delta SE = 2 * (xp - yp) + 5$**

Observar que ΔE e ΔSE são variáveis e não constantes, como no caso da reta

Algoritmo de Bresenham – Circunferências

Usando este cálculo incremental, os valores novos de d são calculados a partir dos anteriores

A questão é como obter o primeiro d ?

Para isto, deve-se considerar que o primeiro d é calculado com xp,yp em (0,R)

Logo o primeiro ponto M é (0+1,R – 1/2)

assim, $d = f(M) = 1^2 + (R - 1/2)^2 - R^2$

logo, **$d = 5/4 - R$**

Algoritmo de Bresenham – Circunferências

O problema aqui é que **$d = 5/4 - R$** usa cálculos em ponto flutuante

Para eliminar a fração, toma-se uma nova variável de decisão h, tal que **$h = d - 1/4$**, logo, **$d = h + 1/4$**

substituindo d por **$h + 1/4$** em **$d = 5/4 - R$**

tem-se $h + 1/4 = 5/4 - R$ → $h = 1 - R$

Mas, a decisão se **$d < 0$** (para escolher E) se transforma em **$h + 1/4 < 0$**

Algoritmo de Bresenham – Circunferências

ou seja, **$h < -1/4$**

Entretanto, como o valor inicial de **h** é inteiro e a variável é sempre incrementada de inteiros (ΔE e ΔSE), esta comparação pode ser alterada para

$h < 0$

Algoritmo de Bresenham – Circunferências

```
void circunferencia(int r, int cor)
{
  x = 0
  y = R
  h = 1 – r
  plotaPixel(x, y, cor)
  while (x<y)
   {
     if (h < 0)
       { // Seleciona E
         h = h + 2 * x + 3
         x = x + 1
       }
     else
       {  // Seleciona SE
         h = h + 2 * (x – y) + 5
         x = x + 1
         y = y - 1
       }
     plotaPixel(x, y, cor)
   }
}
```

Algoritmo de Bresenham – Circunferências

Diferenças de segunda ordem

É possível melhorar o desempenho do algoritmo ainda mais, substituindo as variáveis em ΔE e ΔSE

É segunda ordem, pois o cálculo de d é feito a partir das diferenças (ΔE e ΔSE) entre eles

Agora, as diferenças das diferenças serão calculadas

Algoritmo de Bresenham – Circunferências

Diferenças de segunda ordem

Se escolher **E**, o ponto de avaliação move-se de (xp,yp) para (xp+1,yp)

e, a diferença de 1ª ordem para E fica:

$\Delta E = 2xp + 3$

E_{velho} em (xp,yp) = 2xp + 3 e em

E_{novo} em (xp+1,yp) = 2(xp+1) + 3 = 2xp + 2 + 3

logo, $E_{novo} - E_{velho} =$ **2**

Algoritmo de Bresenham – Circunferências

Analogamente,1

$\Delta SE = 2xp - 2yp + 5$

SE_{velho} em $(xp,yp) = 2xp – 2yp + 5$ e em

SE_{novo} em $(xp+1,yp) = 2(xp+1) – 2yp + 5$

logo, $SE_{novo} – SE_{velho} = \mathbf{2}$

Algoritmo de Bresenham – Circunferências

Se escolher **SE**, o ponto de avaliação move-se de (xp,yp) para (xp+1,yp-1)

Assim,

E_{velho} em (xp,yp) = 2xp + 3 e em

E_{novo} em (xp+1,yp-1) = 2(xp+1) + 3 = 2xp + 2 + 3

logo, $E_{novo} – E_{velho}$ = **2**

Algoritmo de Bresenham – Circunferências

Analogamente,

SE_{velho} em (xp,yp) = 2xp – 2yp + 5 e em

SE_{novo} em (xp+1,yp-1) = 2(xp+1) – 2(yp-1) + 5

logo, $SE_{novo} - SE_{velho} =$ **4**

Algoritmo de Bresenham – Circunferências

No início, $x = 0$ e $y = R$ e $\Delta E = 2xp + 3$

Assim, com $xp = 0$ e $yp = R$, então **$\underline{\Delta E = 3}$**

para $\Delta SE = 2 * (xp – yp) + 5$, com $xp = 0$ e $yp = R$,

então $\Delta SE = 2*(0-R)+5$, ➔ **$\underline{\Delta SE = -2R+5}$**

Algoritmo de Bresenham – Circunferências

```
void circunferencia(int R, int cor)
{
  x = 0
  y = R
  h = 1 – R
  dE = 3
  dSE = -2*R+5
  plotaPixel(x, y, cor)
  while (x<y)
   {
    if (h < 0)
      { // Seleciona E
        h = h + dE
        dE = dE + 2
        dSE = dSE + 2
      }
    else
      { // Seleciona SE
        h = h + dSE
        dE = dE + 2
        dSE = dSE + 4
        y = y - 1
      }
    x = x + 1
    plotaPixel(x, y, cor)
   }
}
```

<u>Prática</u> – Implementar esta versão, usando a simetria-8 para ser entregue, junto com as demais práticas