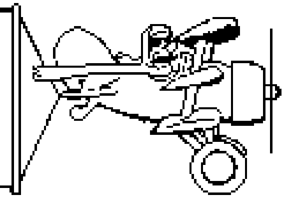
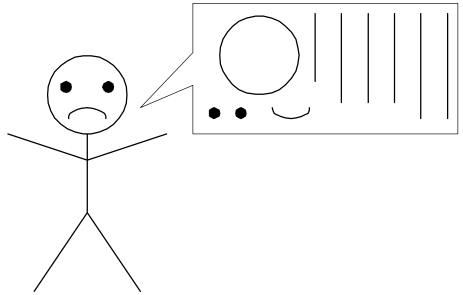
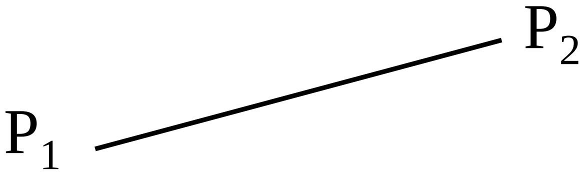
Primitivas de Saída

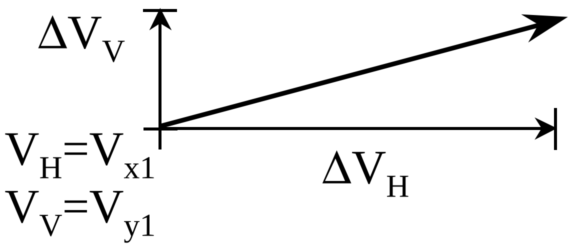


- **Uma primitiva de saída é uma estrutura geométrica básica, a partir da qual podem ser desenvolvidas estruturas mais complexas.**
  - ponto, linha, círculo, curva, caracter...

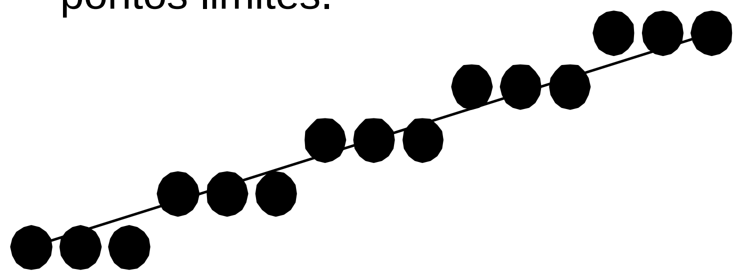




- **varrimento por vectores**
  - variação linear das tensões de deflexão horizontal e vertical proporcionalmente às alterações nas direcções X e Y.



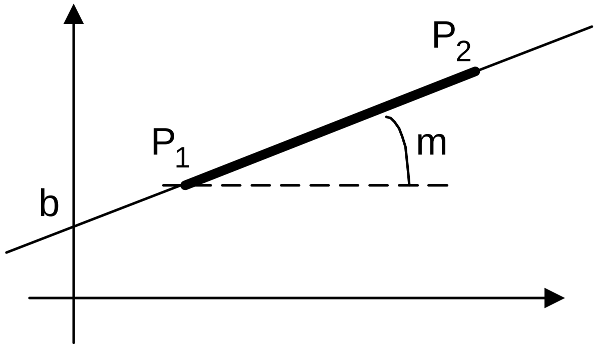
- **varrimento raster**
  - Preenchimento do conjunto de *pixels* que melhor aproxima a linha desejada entre os dois pontos limites.



Rasterização de linhas

Requisitos:

- Devem parecer “rectas”
- Principiar e terminar com precisão
- Brilho constante
- Rápidos (*software*) / Simples (*hardware*)

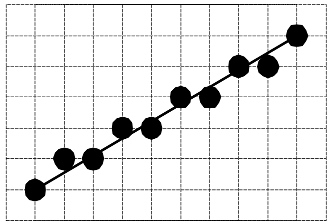


y = m.x + b

$$\begin{cases} m = \dfrac{y_2 - y_1}{x_2 - x_1} \\ b = y_1 - m.x_1 \end{cases}$$

Algoritmo básico

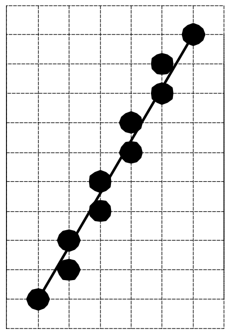
- considerando *-1 <= m <= 1*
  - 1 pixel **on** por coluna



$$\left(x_i, Round(m.x_i + b)\right)$$

$$\begin{cases} x_{i+1} = x_i + 1 \\ y_{i+1} = y_i + m \end{cases}$$

- considerando *|m| > 1*
  - 1 pixel **on** por linha



$$\left(Round\left(\frac{y_i - b}{m}\right), y_i\right)$$

$$\begin{cases} x_{i+1} = x_i + \dfrac{1}{m} \\ y_{i+1} = y_i + 1 \end{cases}$$

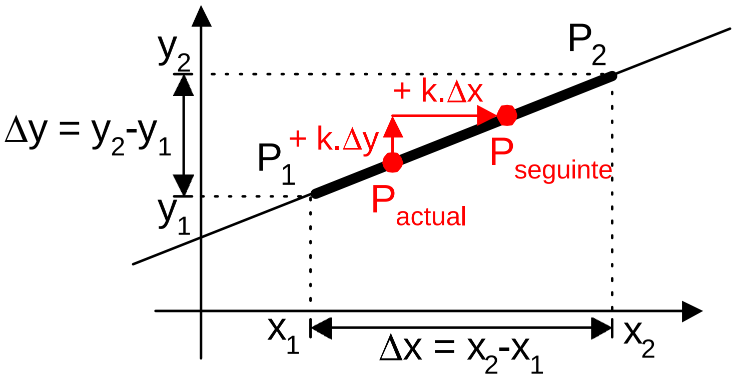
Algoritmos incrementais

- Calculam o ponto seguinte com base no actual.

Algoritmo DDA

Algoritmo DDA

- Analizador diferencial digital



- A partir do ponto actual, encontra-se o novo ponto somando o produto de uma constante K por Δy e Δx respectivamente às cooordenadas x e y desse ponto.
- Como os dispositivos de visualização têm resolução limitada, devem ser gerados apenas os pontos endereçáveis. (Arredondados)
- Como escolher k?

$$k = \frac{1}{\max(|\Delta x|, |\Delta y|)}$$

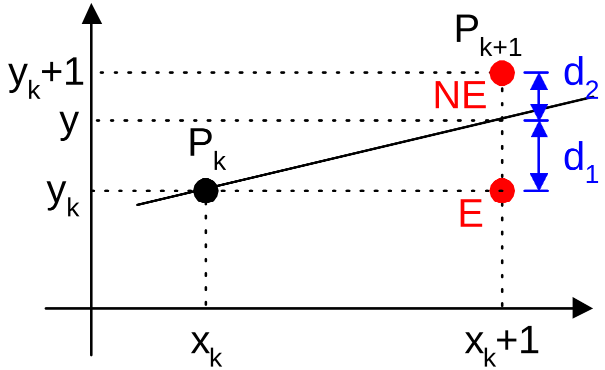
Algoritmo

```
procedimento DDA (x1,y1,x2,y2:inteiro);
   var
      dx, dy, passos, k : inteiro;
      x_incr, y_incr, x, y :  real;
   início
      dx = x2 - x1;
      dy = y2 - y1;
      se abs(dx) > abs(dy) então
         passos = abs(dx);
      senão
         passos = abs(dy);
      fim {se}
      x_incr = dx / steps;
      y_incr = dy / steps;
      x = x1; y = y1;
      set_pixel (round(x),round(y));
      para k = 1 até passos  faça
        início
           x = x + x_incr;
           y = y + y_incr;
           set_pixel (round(x),round(y));
        fim {para}
   fim {procedimento DDA}
```

Algoritmo de Bresenham

☞ Algoritmo de Bresenham (1965)

- Apenas realiza cálculos com inteiros.



- Considerando o ponto actual (P_k), e para o caso em que **0<=m<=1**, o ponto seguinte (P_{k+1}) apenas poderá ser o ponto à direita (E), ou o ponto acima e à direita (NE).
- A escolha do ponto seguinte depende das duas distâncias d_1 e d_2:
  - ➩ $d_1 - d_2 < 0$: ponto E
  - ➩ $d_1 - d_2 >= 0$: ponto NE

Considerando apenas o caso em que **0<=m<=1:**

$$y = m(x_k + 1) + b$$

Calcula-se as duas distâncias:

$$\begin{cases} d_1 = y - y_k = m(x_k + 1) + b - y_k \\ d_2 = (y_k + 1) - y = y_k + 1 - m(x_k + 1) - b \end{cases}$$

Determina-se a diferença entre ambas:

$$d_1 - d_2 = 2m(x_k + 1) - 2y_k + 2b - 1$$

Substitui-se o declive pelo quociente Δy/Δx de forma a utilizar apenas aritmética inteira.
Desta forma obtém-se uma variável de decisão p_k:

$$p_k = \Delta x.(d_1 - d_2) = 2.\Delta y.x_k - 2.\Delta x.y_k + c$$

Em que c é uma constante de valor:

$$c = 2\Delta y + \Delta x(2b - 1)$$

Pode-se agora calcular p_k de forma incremental:

$$p_{k+1} - p_k = 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k)$$

Mas como $x_{k+1} = x_k + 1$:

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k)$$

Em que:

$$y_{k+1} - y_k = \begin{cases} 0 \Leftarrow \text{ponto } E \\ 1 \Leftarrow \text{ponto } NE \end{cases}$$

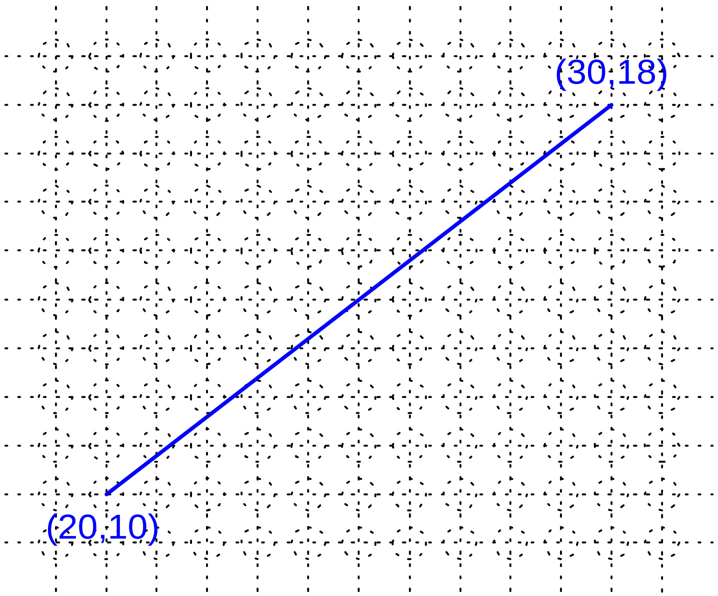
$$p_0 = 2\Delta y - \Delta x$$

Algoritmo de Bresenham

(cont.)

Exemplo

- Faz a rasterização de um segmento de recta do ponto P_1 (20,10) até ao ponto P_2 (30,18).



Algoritmo (considerar $\mathbf{x_1<x_2}$ e $\mathbf{0<=m<=1}$)

```
procedimento Bresenhams (x1,y1,x2,y2:inteiro);
    var
      dx, dy, x, y, p, const1, const2 : inteiro;
    início
      dx = abs( x2 - x1);
      dy = abs( y2 - y1);
      p = 2*dy - dx;
      const1 = 2*dy;
      const2 = 2*(dy-dx);
      x = x1; y = y1;
      set_pixel(x, y);
      enquanto x < x2 faça
         início
            x = x + 1;
            se p < 0 então
               início
                  p = p + const1;
               senão
                  p = p + const2;
                  y = y + 1;
               fim {se}
            set_pixel(x,y);
         fim (enquanto)
    fim {procedimento Bresenhams}
```

Rasterização de Circunferências

Circunferência

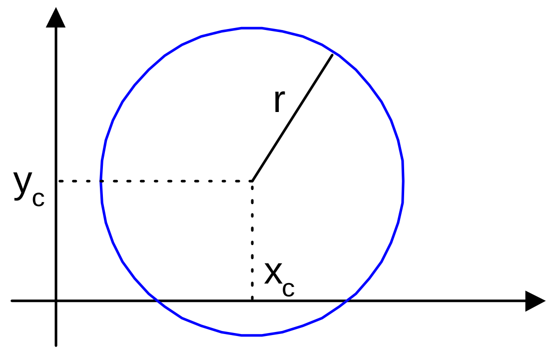
- Uma circunferência é definida como o conjunto de todos os pontos que se encontram a uma distância **r** do centro.
- Equação da circuferência:

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

  - variar x de $\mathbf{x_c}$**-r** a $\mathbf{x_c}$**+r**:

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

  - Elevada carga computacional.
  - Espaçamento não uniforme entre *pixels*.



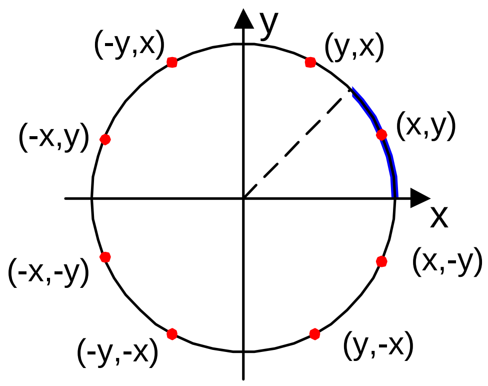
- Equações paramétricas (coordenadas polares):

$$\begin{cases} x = x_c + r.\cos(q) \\ y = y_c + r.sen(q) \end{cases}$$

  - Usando um passo angular constante obtém-se espaçamento uniforme entre *pixels*.

$$\Delta q = \frac{1}{r}$$

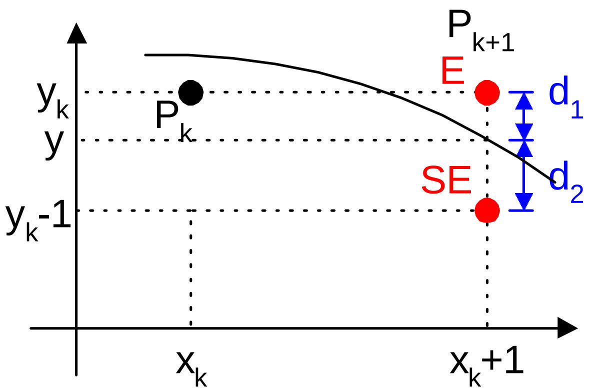
- aproveitar a simetria da circunferência
  - calcular apenas um octante



Rasterização de Circunferências
(cont.)

☞ Algoritmo de Bresenham

- Generalização do algoritmo para rectas.



- Considerando o ponto actual (P_k), e para o caso em que **45º<=θ<=90º**, o ponto seguinte (P_{k+1}) apenas poderá ser o ponto à direita (E), ou o ponto abaixo e à direita (SE).
- A escolha do ponto seguinte depende das duas distâncias d_1 e d_2:
  - ➾ $d_1 - d_2 < 0$: ponto E
  - ➾ $d_1 - d_2 >= 0$: ponto SE

Considerando apenas o caso em que **45º<=θ<=90º:**

$$y^2 = r^2 - (x_k + 1)^2$$

Calculam-se duas medidas relacionadas com as distâncias:

$$\begin{cases} d_1 = y_k^2 - y^2 = y_k^2 - r^2 + (x_k + 1)^2 \\ d_2 = y^2 - (y_k - 1)^2 = r^2 - (x_k + 1)^2 - (y_k - 1)^2 \end{cases}$$

Determina-se a diferença entre ambas:

$$p_k = d_1 - d_2 = y_k^2 - 2r^2 + 2(x_k + 1)^2 + (y_k - 1)^2$$

Pode-se agora calcular p_k de forma incremental:

$$p_{k+1} = y_{k+1}^2 - 2r^2 + 2\left((x_k + 1) + 1\right)^2 + (y_{k+1} - 1)^2$$

$$p_{k+1} - p_k = 4x_k + 6 + 2\left(y_{k+1}^2 - y_k^2\right) - 2(y_{k+1} - y_k)$$

Em que:

$$y_{k+1} - y_k = \begin{cases} 0 \Leftarrow \text{ponto } E \\ -1 \Leftarrow \text{ponto } SE \end{cases}$$

$$y_{k+1}^2 - y_k^2 = \begin{cases} 0 \Leftarrow \text{ponto } E \\ 1 - 2y_k \Leftarrow \text{ponto } SE \end{cases}$$

$$p_0 = 3 - 2r \Leftarrow \text{ponto } (0, r)$$

Donde:

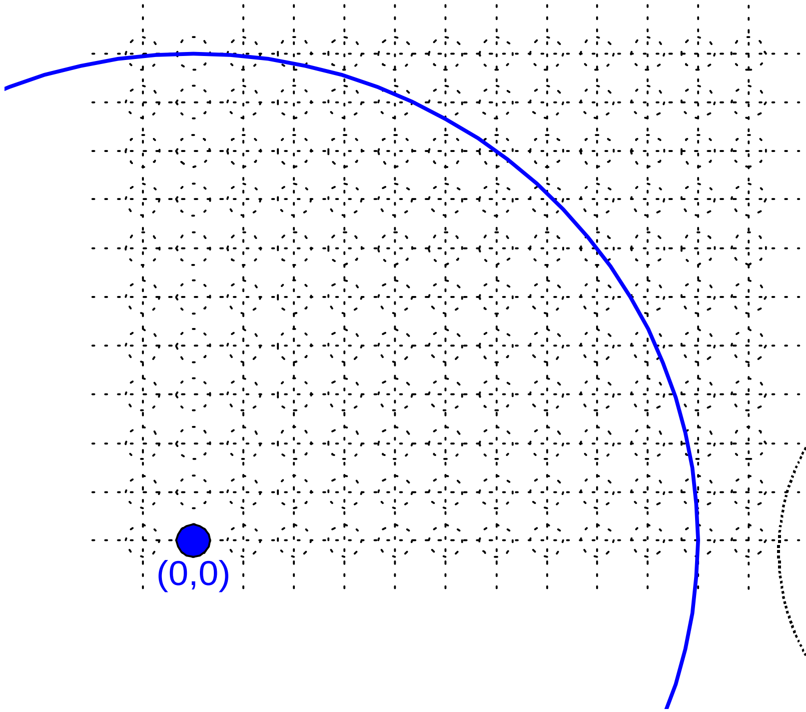
$$p_{k+1} - p_k = \begin{cases} 4x_k + 6 \Leftarrow \text{ponto } E \\ 4(x_k - y_k) + 10 \Leftarrow \text{ponto } SE \end{cases}$$

Rasterização de Circunferências

(cont.)

☞ Exemplo

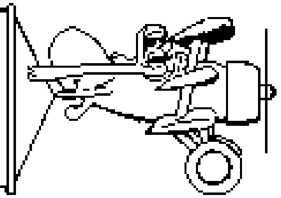
- Faz a rasterização da seguinte circunferência com centro na origem e raio 10.



☞ Algoritmo (considerar **45º<=θ<=90º**)

```
procedimento circBresenhams (xc,yc,r:inteiro);
  var
    x, y, p : inteiro;
  procedimento plot_circle_points;
    início
      set_pixel(xc+x,yc+y);
          ...
      set_pixel(xc-y,yc-x);
    fim{procedimento plot_circle_points}
  início
    x = 0;  y= r; p = 3 - 2*r
    plot_circle_points;
    enquanto x < y faça
      início
        se p < 0 então
          início
            p = p + 4*x + 6;
          senão
            p = p + 4*(x - y) + 10;
            y = y - 1;
          fim {se}
        x = x + 1;
        plot_circle_points;
      fim (enquanto)
  fim {procedimento circBresenhams}
```

```
set_pixel(xc+x,yc+y);
set_pixel(xc-x,yc+y);
set_pixel(xc+x,yc-y);
set_pixel(xc-x,yc-y);
set_pixel(xc+y,yc+x);
set_pixel(xc-y,yc+x);
set_pixel(xc+y,yc-x);
set_pixel(xc-y,yc-x);
```



Rasterização de Primitivas

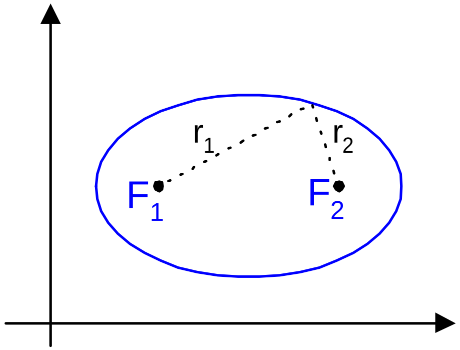
Elipses

- Uma elipse é definida como o conjunto de todos os pontos tais que a soma da distância a dois pontos denominados focos é igual a uma constante.
- Extensão do algoritmo de rasterização de circunferências.
- Equação da elipse:

$$\left(\frac{x - x_c}{r_1}\right)^2 + \left(\frac{y - y_c}{r_2}\right)^2 = 1$$

- Equações paramétricas:

$$\begin{cases} x = x_c + r_1 . \cos(q) \\ y = y_c + r_2 . sen(q) \end{cases}$$



Outras curvas

- Métodos semelhantes aos anteriores permitem gerar grande variedade de curvas:
  - cónicas, trignométricas, polinomiais, spline, …
    - Posicionamento através da equação ***y = f(x)***
    - Algoritmos incrementais a partir de ***f(x,y) = 0***

Caracteres

- O estilo de desenho de um conjunto (familia) de caracteres denomina-se **font** ou **typeface**.
- Duas formas de representação:
  - **Bitmapped font**: armazenado numa grelha rectangular.
    - Ocupam bastante memória, dado que todas as variações do tipo de letra devem ser armazenadas.
  - **Outline font**: definidas a partir de um conjunto de primitivas geométricas.
    - Ocupam menos memória, dado que os diferentes estilos podem ser obtidos manipulando a sua definição geométrica.
    - Maior carga computacional.