Computação Gráfica I

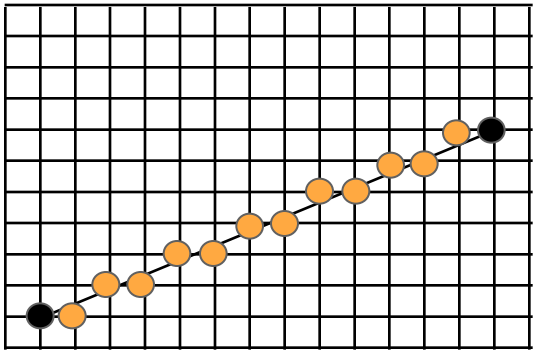
Professor:

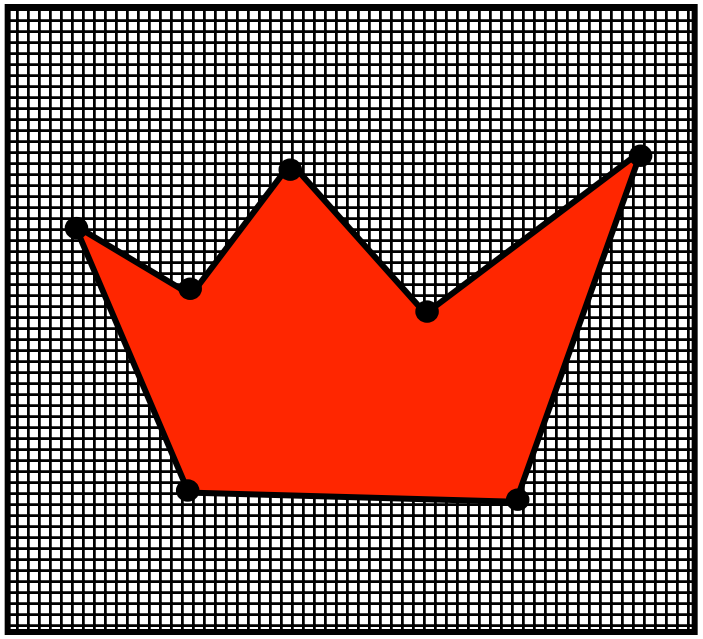
Anselmo Montenegro
www.ic.uff.br/~anselmo

Conteúdo:

- Algoritmos para rastreio (*scan-conversion*).

Algoritmos para rastreio: *introdução*



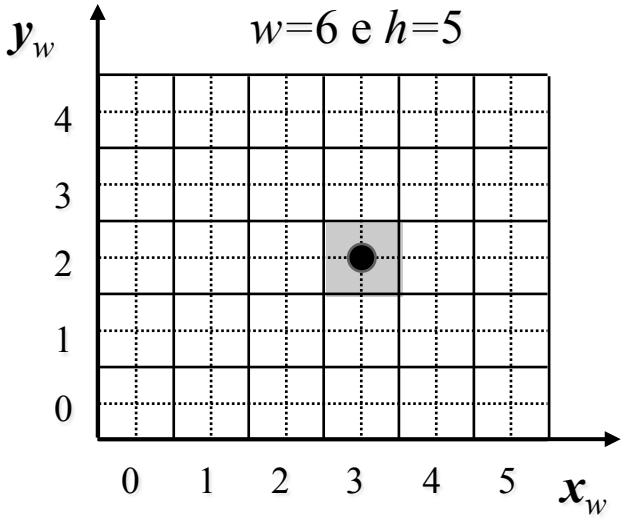


Algoritmos para rastreio: *Coordenadas de um ponto na janela (windows coordinate)*

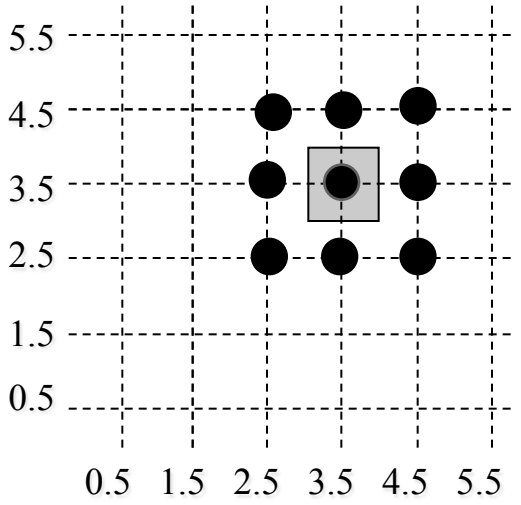
- As coordenadas do centro de um ponto de tamanho impar são:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \lfloor x_w \rfloor + \frac{1}{2} \\ \lfloor y_w \rfloor + \frac{1}{2} \end{pmatrix}$$

OpenGL Spec



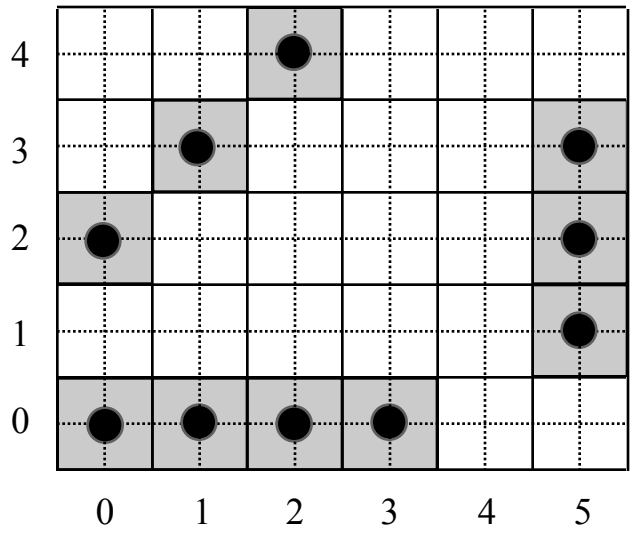
ponto: (3,2) tamanho 1 sem anti-alias



ponto: (3,3) tamanho 3 sem anti-alias

Algoritmos para rastreio: *critério geométrico para linhas horizontais, verticais e à 45°*

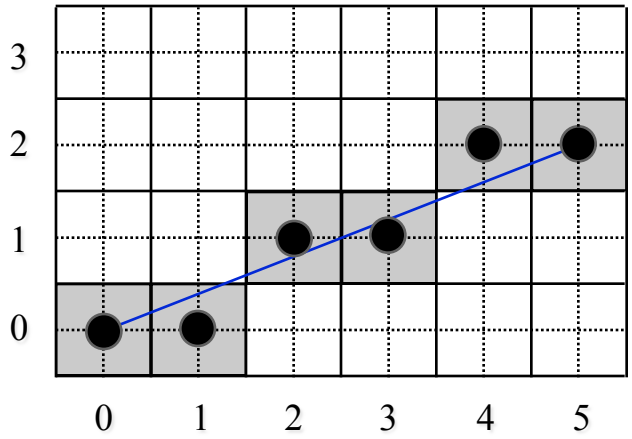
- Casos triviais em que as linhas passam pelo centro das células.

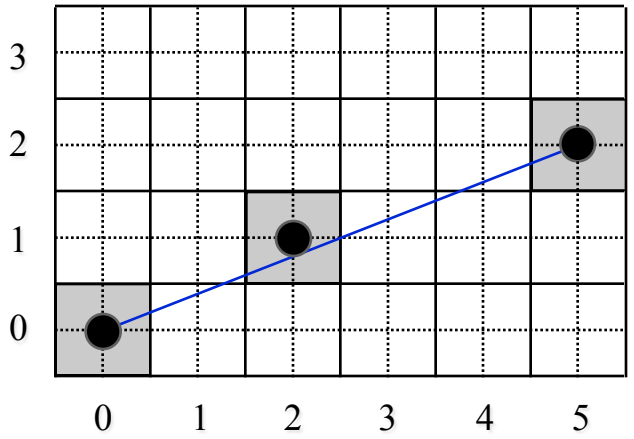



```
linha:  (0,0),(4,0)
linha:  (5,1),(5,4)
linha:  (0,2),(3,5)
```

Algoritmos para rastreio: *critério geométrico de Bresenham (1965) para linhas*

linha: (0,0),(5,2)





x dominante

☹

$|x_2 - x_1| \geq |y_2 - y_1| \Rightarrow$ *x dominante, um pixel por coluna*

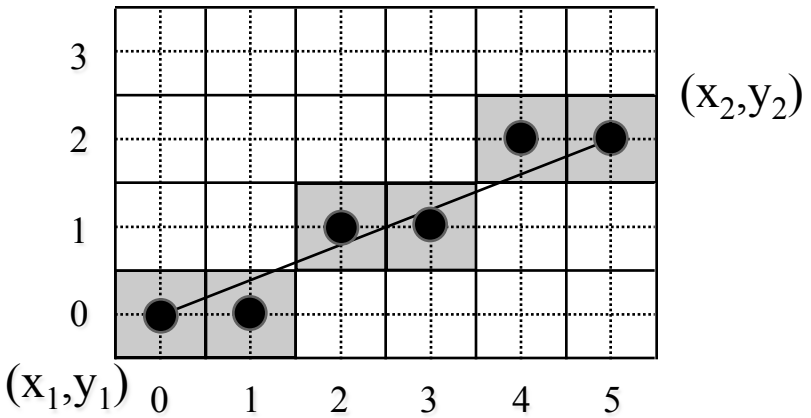
$|x_2 - x_1| < |y_2 - y_1| \Rightarrow$ *y dominante, um pixel por linha*

Algoritmos para rastreio: *algoritmo simples para o primeior octante*

$$y_i = mx_i + b$$

$$\begin{cases} m = \dfrac{y_2 - y_1}{x_2 - x_1} \\ b = y_1 - mx_1 \end{cases}$$

```
#define ROUND(x) (int)floor((x)+0.5)

void linha(int x1, int y1, int x2, int y2)
{
 float m = (y2-y1)/(x2-x1);
 float b = y1 - m*x1;
 float y;

 fragmento(x1,y1);
 while( x1 < x2 )
     {
      x1++;
      y = m*x1 + b;

      fragmento(x1,ROUND(y));
     }
}
```



Podemos evitar a multiplicação?

Algoritmos para rastreio: *algoritmo simples para o primeior octante – forma incremental*

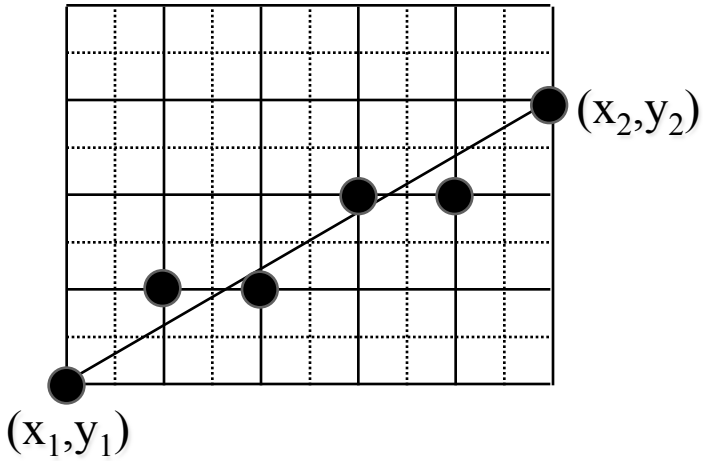
$$y_i = mx_i + b$$

$$y_{i+1} = m(x_i + 1) + b$$

$$\begin{cases} m = \dfrac{y_2 - y_1}{x_2 - x_1} \\ b = y_1 - mx_1 \end{cases}$$

$$y_i = mx_i + b$$

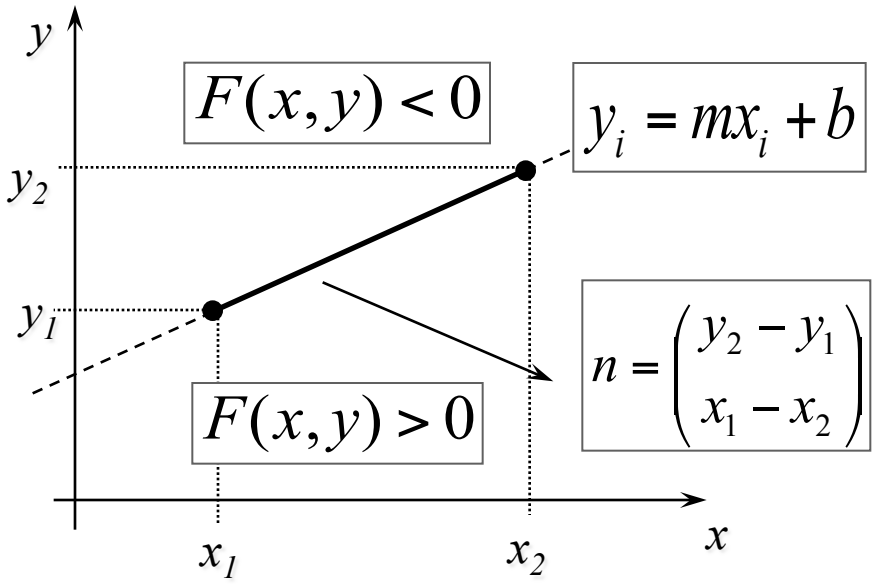
$$y_{i+1} - y_i = m$$




```
void linha(int x1, int y1, int x2, int y2)
{
 float m = (y2-y1)/(x2-x1);
 float b = y1 - m*x1;
 float y=y1;
 pixel(x1,y1);
 while( x1 < x2 )
     {
      x1++;
      y += m;
      fragmento(x1,ROUND(y));
     }
}
```

Podemos evitar o uso de ponto flutuante?

Algoritmos para rastreio: *equação implícita da reta*



$$y = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)x + \left(y_1 - \frac{y_2 - y_1}{x_2 - x_1}x_1\right) \qquad \begin{cases} m = \dfrac{y_2 - y_1}{x_2 - x_1} \\ b = y_1 - mx_1 \end{cases}$$

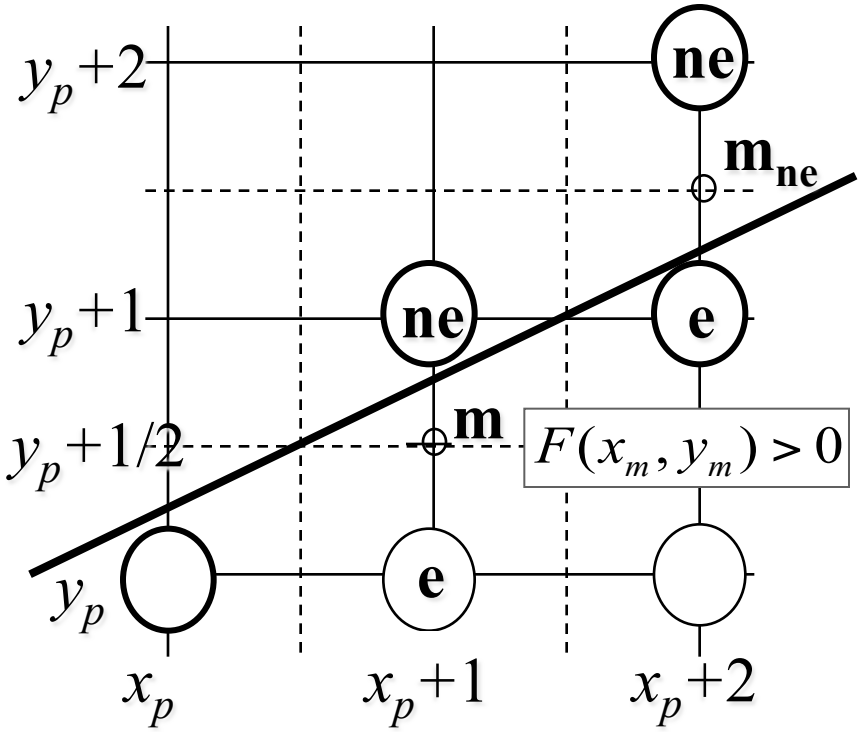
$$(x_2 - x_1)y = (y_2 - y_1)x + ((x_2 - x_1)y_1 - (y_2 - y_1)x_1)$$

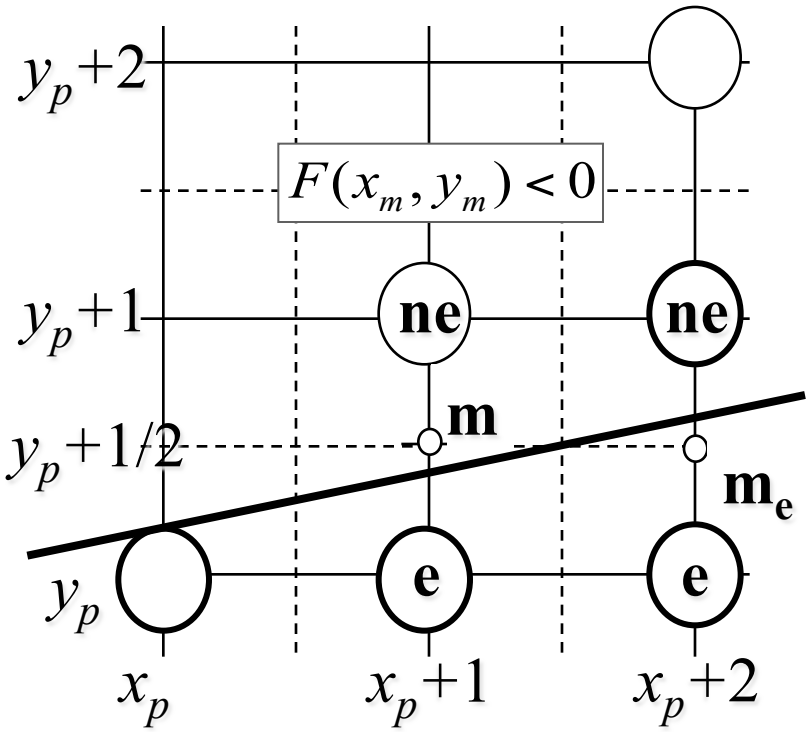
$$(y_2 - y_1)x + (x_1 - x_2)y + ((x_2 - x_1)y_1 - (y_2 - y_1)x_1) = 0$$

$$F(x,y) = a.x + b.y + c$$

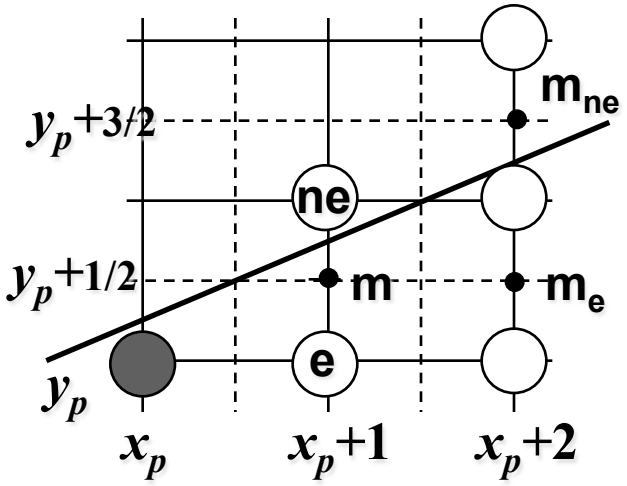
Algoritmos para rastreio: *equação básica do algoritmo do ponto médio para linhas*

$$F(x, y) = a.x + b.y + c$$





Algoritmos para rastreio: *algoritmo do ponto médio - versão para aritmética inteira*



$$d(x,y) = 2F(x,y)$$

$$d(\mathbf{m}) = 2F(\mathbf{m}) = \begin{cases} > 0 \rightarrow escolha\ \mathbf{ne} \\ \leq 0 \rightarrow escolha\ \mathbf{e} \end{cases}$$

$$d_{ini} = 2F(x_0 + 1, y_0 + \tfrac{1}{2}) = 2a(x_0 + 1) + 2b(y_0 + \tfrac{1}{2}) + 2c = 2a + b$$

e

$$d_{novo} = 2F(x_p + 2, y_p + \tfrac{1}{2}) = 2a(x_p + 2) + 2b(y_p + \tfrac{1}{2}) + 2c$$
$$d_{novo} = d_{ant} + 2a \qquad \Delta_e = 2a$$

ne

$$d_{novo} = 2F(x_p + 2, y_p + \tfrac{3}{2}) = 2a(x_p + 2) + 2b(y_p + \tfrac{3}{2}) + 2c$$
$$d_{novo} = d_{ant} + 2a + 2b \qquad \Delta_{ne} = 2a + 2b$$

$d_{ini} = 2.a + b$

$\Delta_e = 2a$

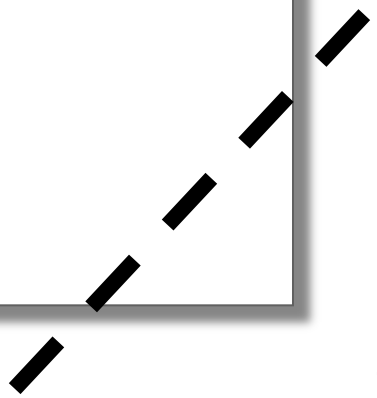
$\Delta_{ne} = 2(a+b)$

Algoritmos para rastreio: *algoritmo do ponto médio para linhas – código em C*

```
void linhaPM(int x1, int y1, int x2, int y2)
{
   int a = y2-y1;
   int b = x1-x2;
   int d=2*a+b;              /* valor inicial da var. decisao */
   int incrE = 2*a;          /* incremento p/ mover E */
   int incrNE = 2*(a+b);     /* incremento p/ mover NE */

   fragmento(x1,y1);

   while (x1<x2) {
       x1++;
       if (d<=0)       /* escolha E */
          d+=incrE;
       else {          /* escolha NE */
          d+=incrNE;
          y1++;
       }
       fragmento(x1,y1);
   }

}
```

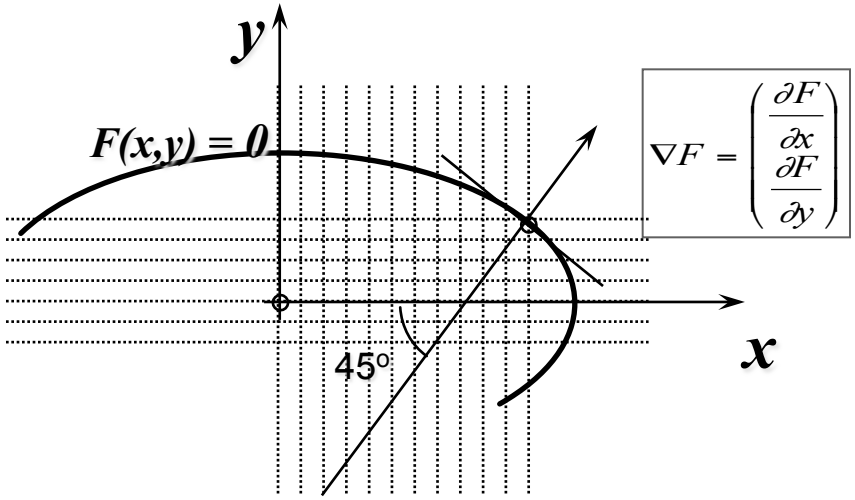
Algoritmos para rastreio: *algoritmo do ponto médio para linhas com estilo – código em C*

```
void linhaPM(int x1, int y1, int x2, int y2)
{
   int a = y2-y1;
   int b = x1-x2;
   int d=2*a+b;              /* valor inicial da var. decisao */
   int incrE = 2*a;          /* incremento p/ mover E */
   int incrNE = 2*(a+b);     /* incremento p/ mover NE */
   int style[8]={1,1,0,0,1,1,0,0}; int k=1;
   fragmento(x1,y1);  /* primeiro pixel */

   while (x1<x2) {
       x1++;
       if (d<=0)      /* escolha E */
          d+=incrE;
       else {         /* escolha NE */
          d+=incrNE;
          y1++;
       }
       if (style[(++k)%8]==1) fragmento(x1,y1);
   }

}
```



Algoritmos para rastreio: *rastreio para elipses – critério de Bresenham*

- Segundo o critério de Bresenham se a curva está mais próxima da horizontal tomamos um ponto por coluna caso contrário tomamos um ponto por linha.
- O ponto de transição é o que possui gradiente na direção dada pelo ângulo de 45 graus.

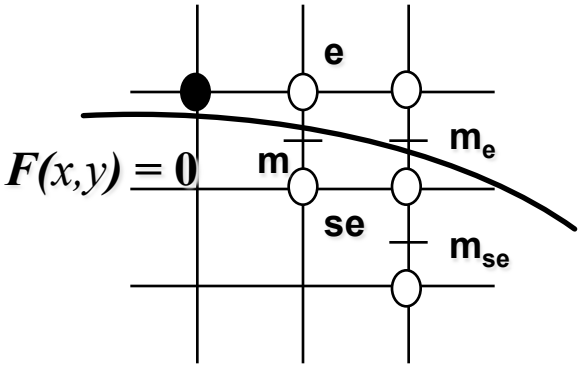


Algoritmos para rastreio: *rastreio para elipses*

- O critério do ponto médio é aplicado de forma semelhante ao aplicado para linhas.
- A diferença principal é que F(x,y) é quadrática.
- Para implementar o algoritmo eficazmente é necessário calcular incrementos dos incrementos de y quando x é acrescido de 1.

Algoritmos para rastreio: *rastreio para elipses*

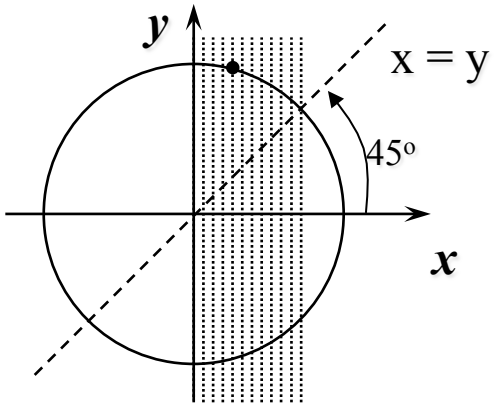
- A figura abaixo ilustra a porção do primeiro quadrante no qual a curva é x dominante. Neste caso as escolhas são e e se.
- A partir do ponto de transição a curva é y dominante e as escolhas são s e se.

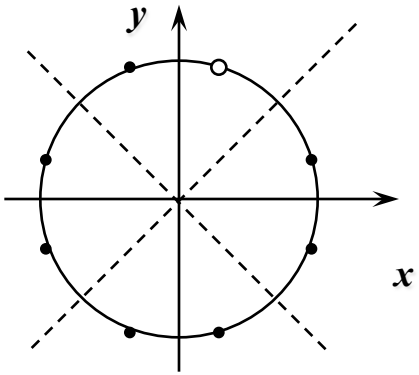


Algoritmos para rastreio: *rastreio de círculos*

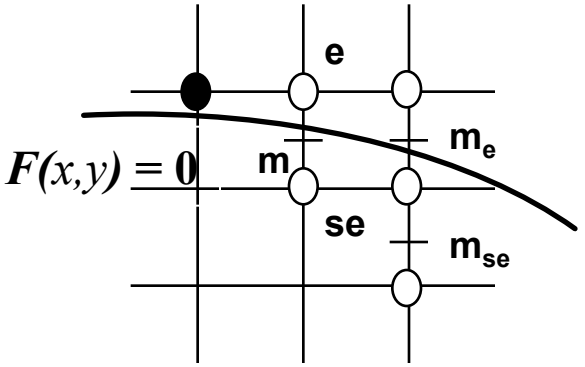
- O rastreio de um círculo é um caso particular do rastreio de uma elipse.
- Trabalhamos com o segundo octante, no qual a curva é x dominante.
- Utilizamos a simetria do círculo para definir os pontos nos demais octantes.

Algoritmos para rastreio: *rastreio de círculos*



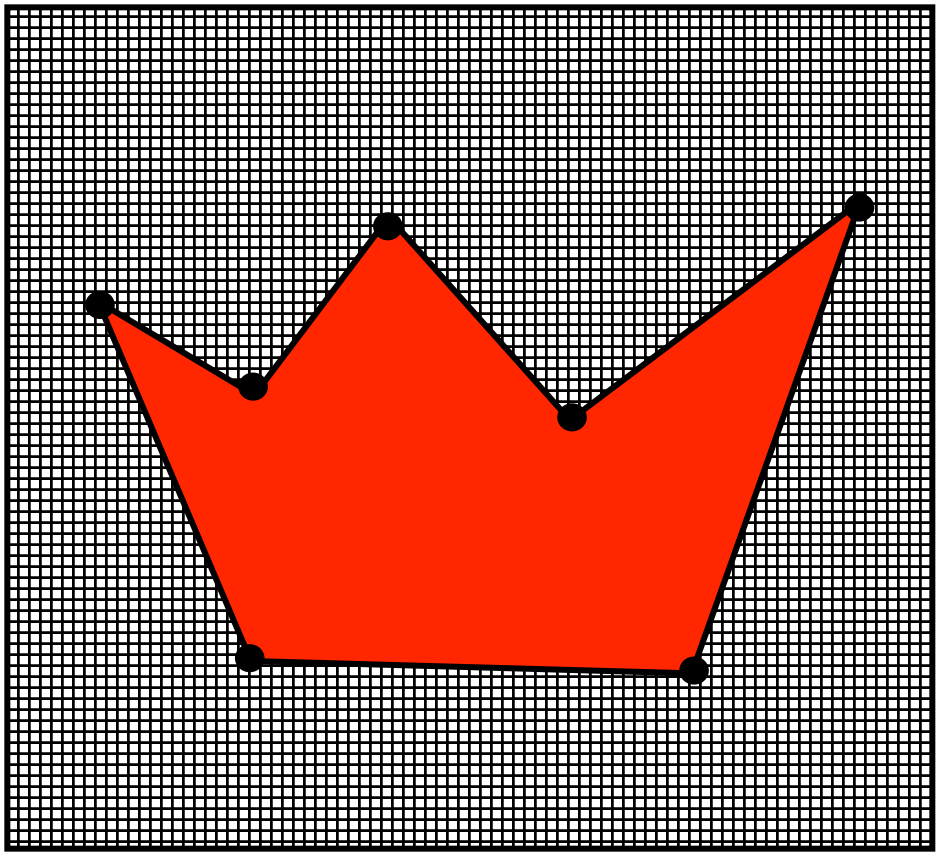


cada ponto calculado define 8 *pixels*

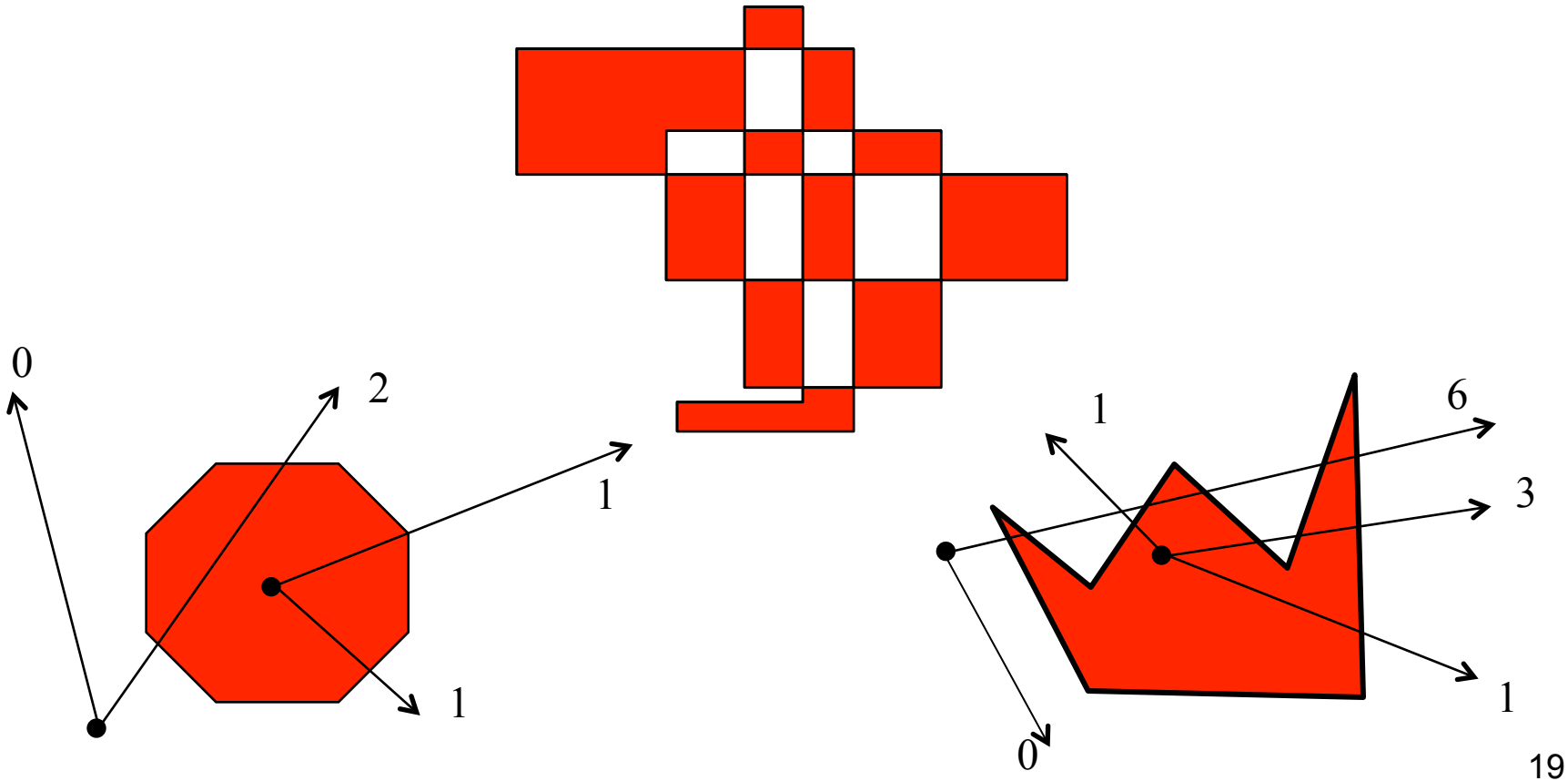



```
x=0,y=raio;
fragmento(x,y);
while (x<y) {
      x++;
      if (F(M)<0)
         escolha E;
     else
         escolha SE;
     fragmento(E ou SE);
}
```

Algoritmos para rastreio: *rastreio de polígonos*

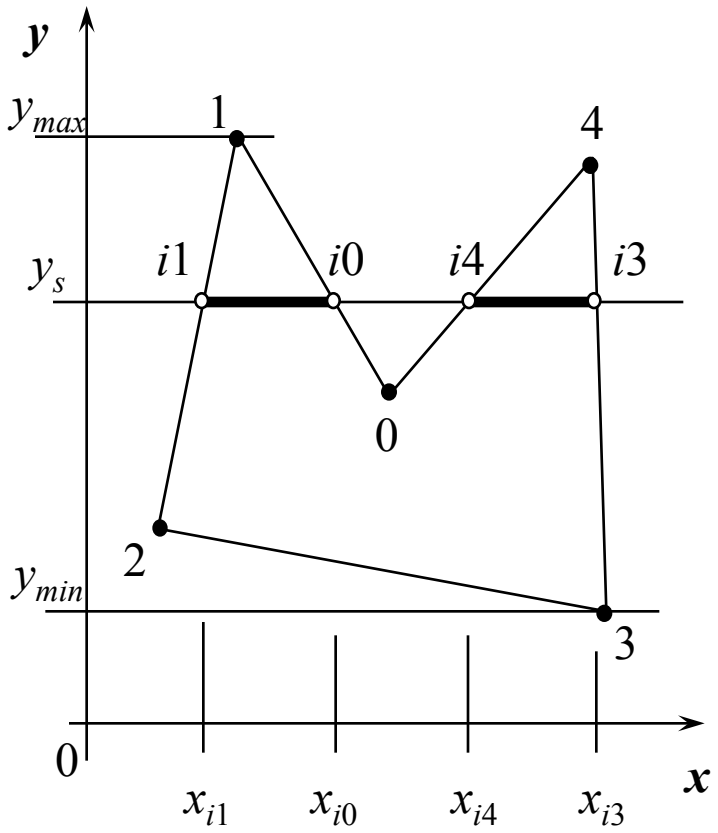


Algoritmos para rastreio: *rastreio de polígonos – noção de interior para um polígono qualquer*



Algoritmos para rastreio: *preenchimento de polígonos*

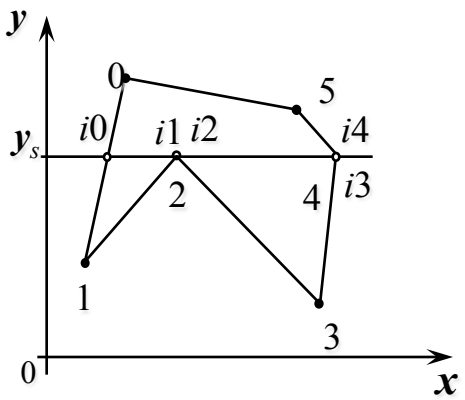
```
dados: {(x0,y0), (x1,y1), (x2,y2) (x3,y3), (x4,y4)}
```

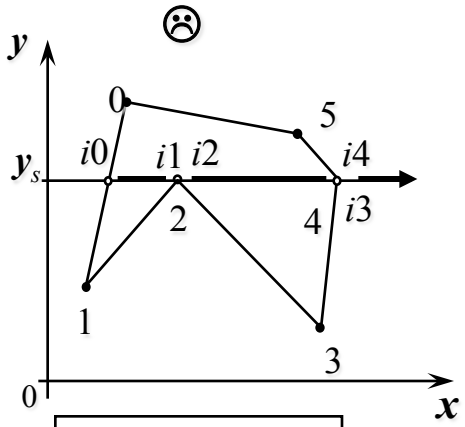



```
Achar ymax e ymin
Para cada y ∈[ymax,ymin]
   Para cada aresta
        calcular as interseções
   ordenar interseções
desenhar linhas horizontais
```

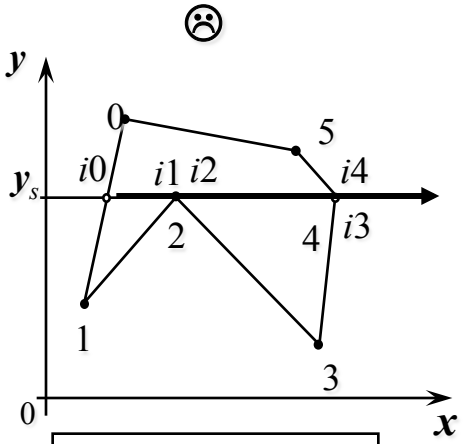
$$vx = \{x_{i1}, x_{i0}, x_{i4}, x_{i3}\}$$

Algoritmos para rastreio: *interseção nos vértices*



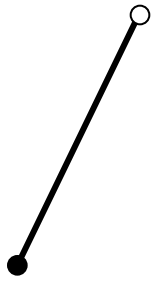


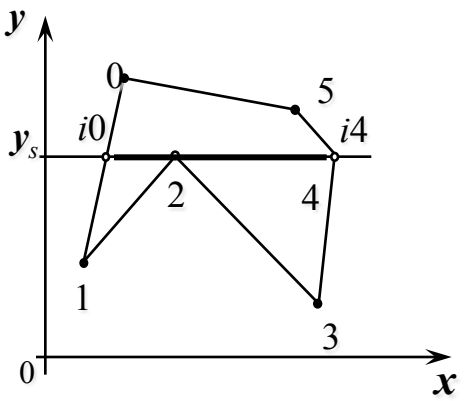
inclui vértices:
*i*0-*i*1, *i*2-i3, *i*4-?



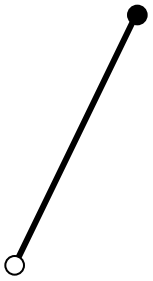
não inclui vértices:
*i*0-?

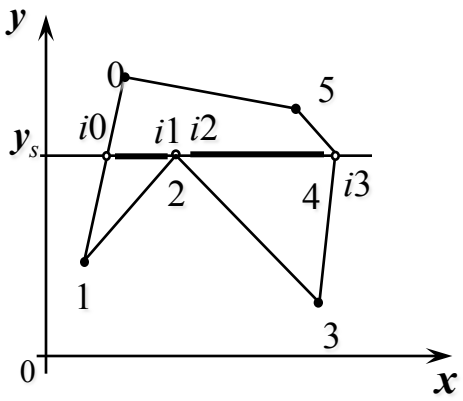
Solução:





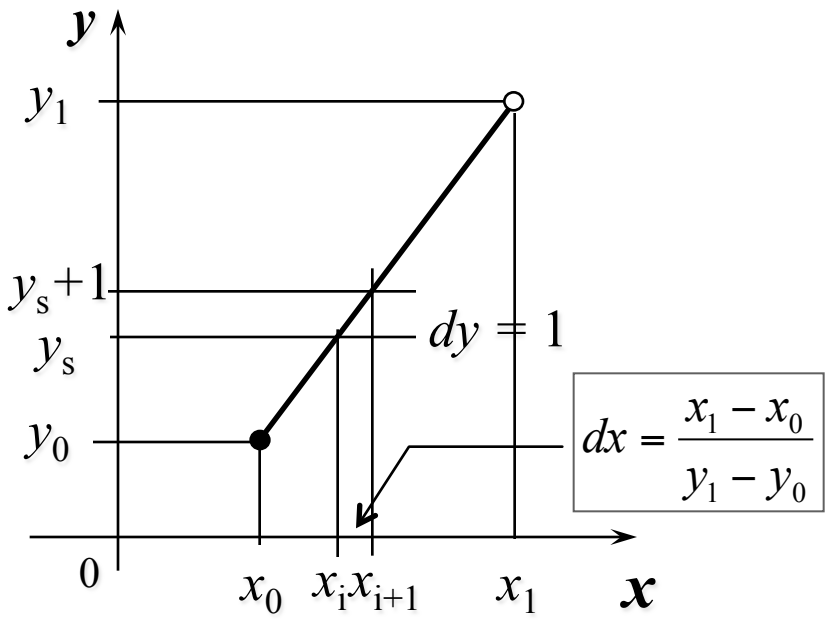
ou





Algoritmos para rastreio: *otimização do algoritmo de preenchimento – interpolação linear*

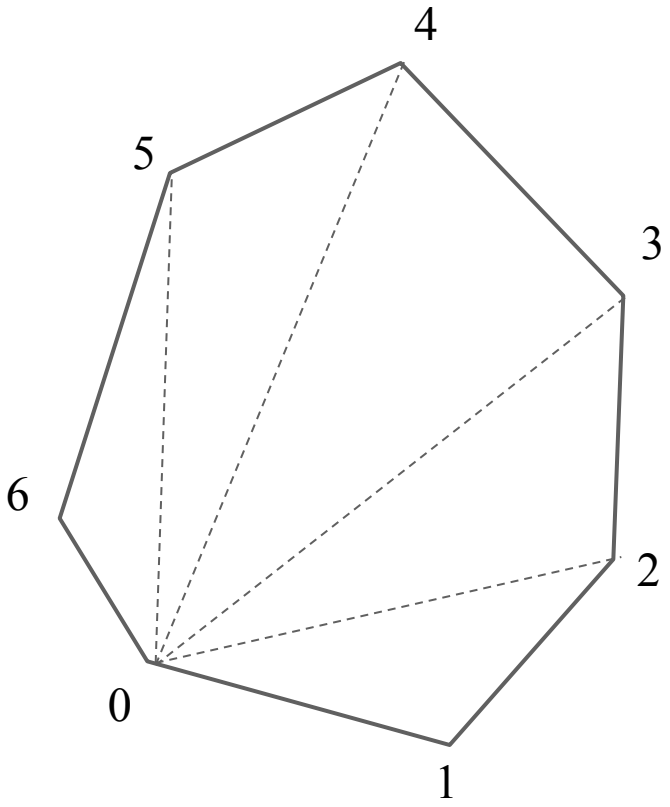
Interpolação linear na aresta

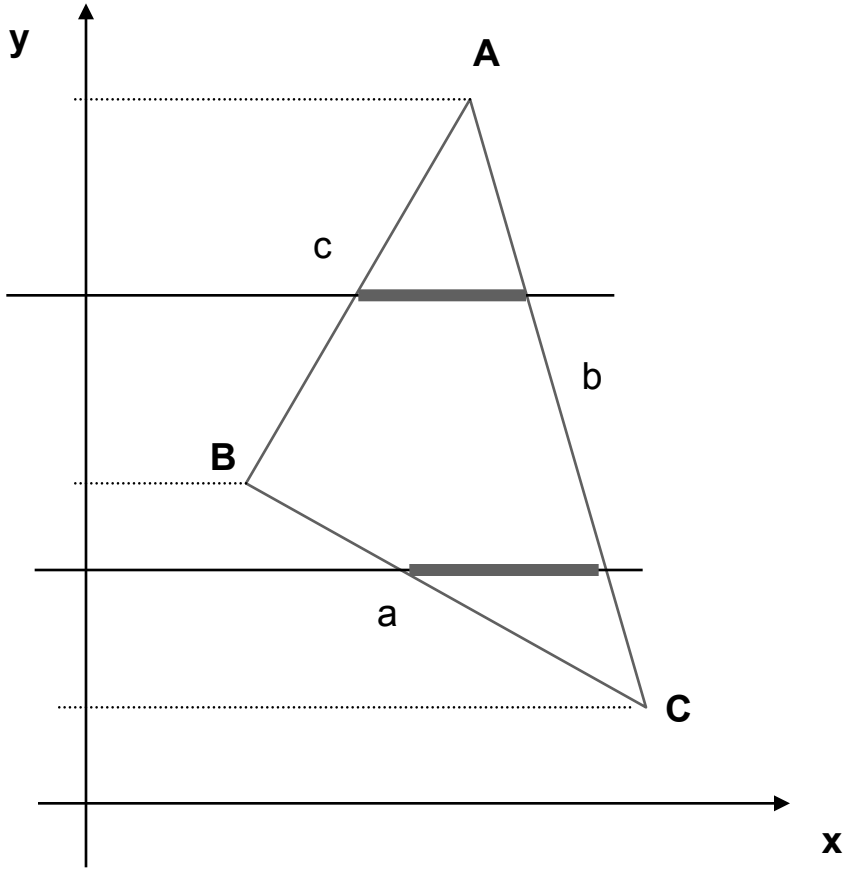


$$x_0, x_0 + dx,\ x_0 + 2dx, \ldots$$

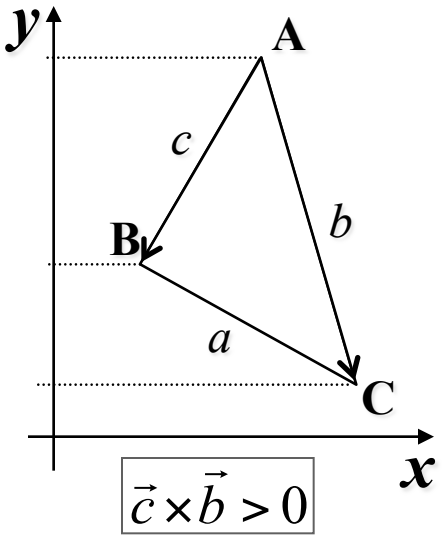
$$c_0, c_0 + dc,\ c_0 + 2dc, \ldots$$

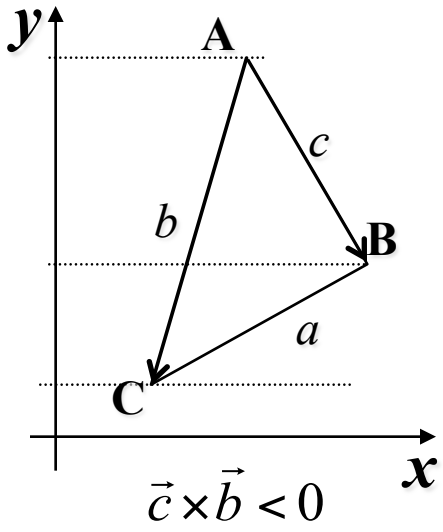
Algoritmos para rastreio: *otimização do algoritmo de preenchimento – triângulos*





Algoritmos para rastreio: *possíveis configurações para um triângulo* $y_A \geq y_B \geq y_C$





$$\vec{c} \times \vec{b} = (x_B - x_A)(y_C - y_A) - (x_C - x_A)(y_B - y_A)$$

Algoritmos para rastreio: *rastreio de um triângulo*