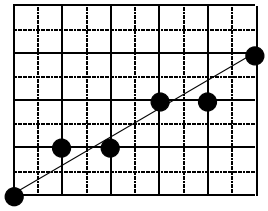
Rasterização de linhas e polígonos

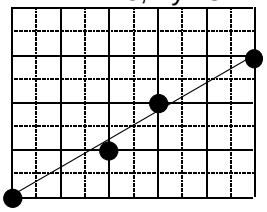
Algoritmos de *rasterização* de linhas

Suponha $\Delta x > \Delta y$



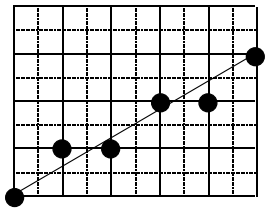
incrementa x e
vê o que acontece com y

$\Delta x = 5, \Delta y = 3$



incrementa y e
vê o que acontece com x

Algoritmo simples de linha (no primeiro octante)

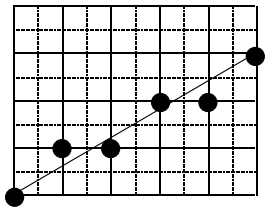


$y_i = m\,x_i + b$

onde:
$m = \Delta y/\Delta x$
$b = y_1 - m\,x_1$

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

 SetPixel(x1,y1, color);   y = y1;

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

     SetPixel(x1,ROUND(y), color);
    }
}
```

Algoritmo de linha incremental

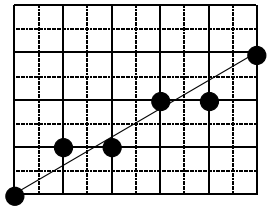


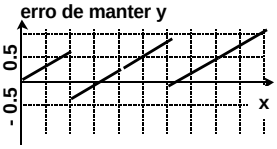
Se
$x_{i+1} = x_i + 1$

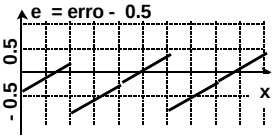
então
$y_{i+1} = y_i + \Delta y/\Delta x$

```
void LineDDA(int x1, int y1, int x2, int y2, int color)
{
 float y;
 float m = (y2-y1)/(x2-x1);

 SetPixel(x1,y1, c);   y = y1;

 while( x1 < x2 )
    {
     x1++;
     y += m;

     SetPixel(x1,ROUND(y), c);
    }
}
```

Algoritmo de linha baseado no erro








```
void BresLine0(int x1, int y1, int  x2, int y2, int c)
{
 int  Dx = x2 - x2;
 int  Dy = y2 - y1;
 float e= -0.5;

 SetPixel(x1, y1, c);

 while( x1 < x2 )
   {
    x1++;  e+=Dy/Dx;

    if (e>=0) {
       y1++ ;  e -= 1;
     }

    SetPixel(x1, y1, c);
   }
}
```

Algoritmo de Bresenham

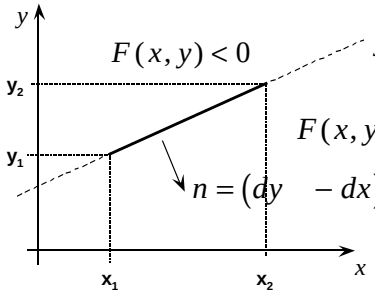
ei = 2*Dx*e

```
void BresLine0(int x1, int y1,
                    int  x2, int y2, int c)
{
 int  Dx = x2 - x1;
 int  Dy = y2 - y1;
 float e= -0.5;

 SetPixel(x1, y1, c);

 while( x1 < x2 )
   {
    x1++;  e+=Dy/Dx;

    if (e>=0) {
       y1++ ;  e -= 1;
     }

    SetPixel(x1, y1, c);
   }
}
```

```
void BresLine1(int x1, int y1,
                    int  x2, int y2, int c)
{
 int  Dx = x2 - x1;
 int  Dy = y2 - y1;
 int  ei = -Dx;

 SetPixel(x1, y1, c);

 while( x1 < x2 )
   {
    x1++;  ei += 2*Dy;

    if (ei>=0) {
       y1++ ;  ei -= 2*Dx;
     }

    SetPixel(x1, y1, c);
   }
}
```

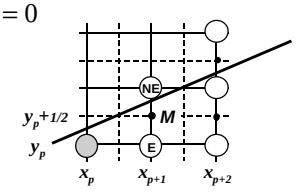
válidos somente quando Dx>Dy, x2 > x1 e y2 > y1

Equação implícita da reta

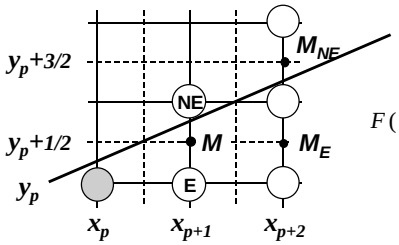


$$F(x,y) = dy.x - dx.y + B.dx = 0$$

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



Algoritmo do ponto médio - variável de decisão -



$$F(M) = \begin{cases} > 0 \rightarrow \text{escolha NE} \\ \leq 0 \rightarrow \text{escolha E} \end{cases}$$

$$d = F(x_p + 1, y_p + \tfrac{1}{2}) = a(x_p + 1) + b(y_p + \tfrac{1}{2}) + c$$

(E)
$$d_{new} = F(x_p + 2, y_p + \tfrac{1}{2}) = a(x_p + 2) + b(y_p + \tfrac{1}{2}) + c$$
$$d_{new} = d_{old} + a \qquad \Delta_E = a$$

(NE)
$$d_{new} = F(x_p + 2, y_p + \tfrac{3}{2}) = a(x_p + 2) + b(y_p + \tfrac{3}{2}) + c$$
$$d_{new} = d_{old} + a + b \qquad \Delta_{NE} = a + b$$

Algoritimo do ponto médio - redução para inteiros -

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

$$d_{start} = F(x_0, y_0) + a + b/2 = a + b/2$$
$$\Delta_E = a$$
$$\Delta_{NE} = a + b$$

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

$$d_{start} = 2.a + b$$
$$\Delta_E = 2a$$
$$\Delta_{NE} = 2(a + b)$$

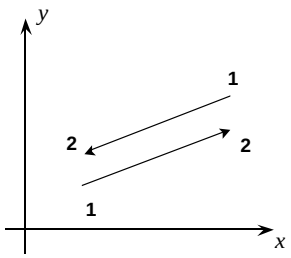
Algoritimo do ponto médio - código C -

```
void MidpointLine(int x0, int y0, int x1, int y1, int color)
{
   int dx = x1-x0;
   int dy = y1-y0;
   int d=2*dy-dx;         /* Valor inicial da var. decisao */
   int incrE = 2*dy;        /* incremento p/ mover E */
   int incrNE = 2*(dy-dx); /* incremento p/ mover NE */
   int x=x0;
   int y=y0;
   Pixel(x,y,fgcolor);  /* Primeiro pixel */

   while (x<xl) {
       if (d<=0) {  /* Escolha E */
          d+=incrE;
          x++;
       } else {     /* Escolha NE */
          d+=incrNE;
          x++;
          y++;
       }
       Pixel(x,y,color);
   } /* while */

} /* MidpointLine */
```

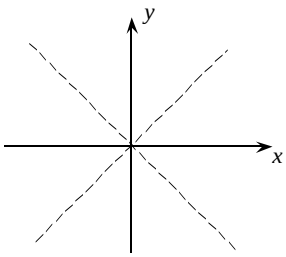
Estilos de linha

```
void MidpointLine(int x0, int y0, int x1, int y1, int color)
{
   int dx = x1-x0;
   int dy = y1-y0;
   int d=2*dy-dx;         /* Valor inicial da var. decisao */
   int incrE = 2*dy;       /* incremento p/ mover E */
   int incrNE = 2*(dy-dx); /* incremento p/ mover NE */
   int x=x0; int y=y0;
   int style[8]={1,1,0,0,1,1,0,0}; int k=1;

  Pixel(x,y,fgcolor)}
   while (x<xl) {
       if (d<=0) {  /* Escolha E */
          d+=incrE;
          x++;
       } else {     /* Escolha NE */
          d+=incrNE;
          x++;
          y++;
       }
        if (style[(++k)%8]) Pixel(x,y,color);
   } /* while */

} /* MidpointLine */
```

Rasterização de Retas -caso geral-

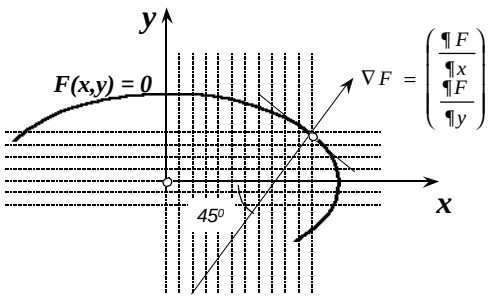


orientação

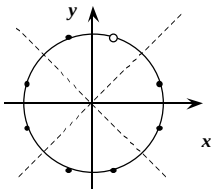


outros quadrantes

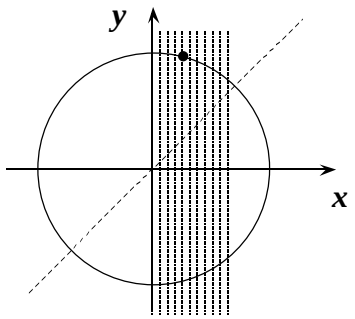
Rasterização de Cônicas



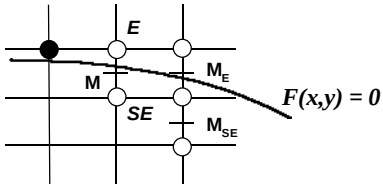
simetrias do círculo:
cada ponto calculado
define 8 *pixels*



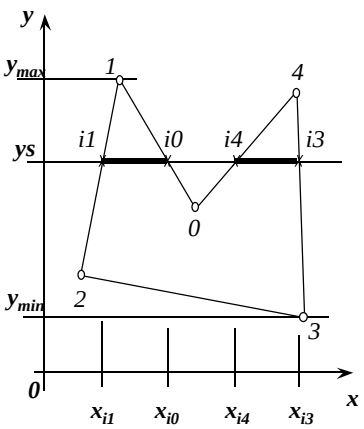
Rasterização de Cônicas




```
y=raio;
for (x=0; x< y; x++) {
    if F(M)<0
       escolha E
    else
       escolha SE
    Pixel (E ou SE)
    pinte os simétricos
}
```



Preenchimento de polígonos



dados:
$\{x_0, x_1, x_2, x_3, x_4\}$
$\{y_0, y_1, y_2, y_3, y_4\}$

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

acha y_{max} *e* y_{min}

Para cada $ys \in [y_{max}, y_{min}]$

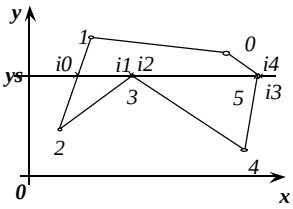
Para cada aresta

calcula as interseções

ordena interseções

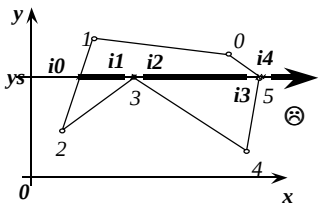
desenha linhas horizontais

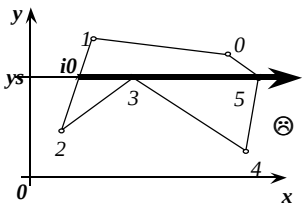
Preenchimento de polígonos (*scan* passando por vértices)



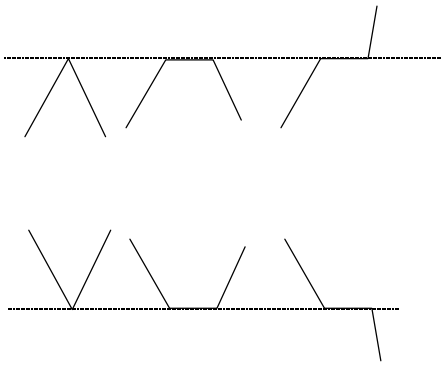
inclui vértices: i0-i1, i2-i3, i4-?

não inclui vértices: i0-?





Interseção nos vértices



só inclui vértices de menor y:
i0-i4 ☺

ou

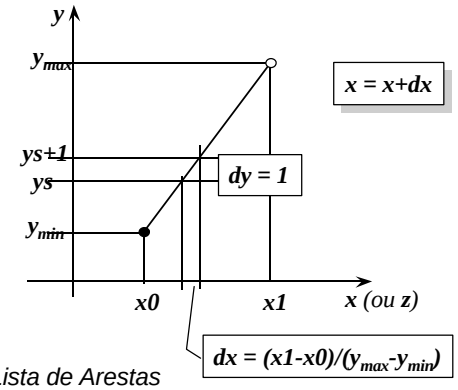
só inclui vértices de maior y:
i0-i1, i2-i3 ☺

reta horizontal não produz interseção

Algoritmo de *Fill*

```
void FillPolygon (int np, int *x, int *y)
{
 /* declarações */
   . . .

 /* calcula y max e min dos vértices*/
   . . .

 for(ys=ymim; ys<=ymax; ys--)    /* para cada linha de scan */
  {
   num_inters = 0;
   for(i=0; i<np; i++)                  /* para cada aresta */
    {
     yi = y[i];    yf =  y[(i+1)%np];
     if (yi!=yf && ys >= MIN(yi,yf) && ys < MAX(yi,yf) )
      {
       vxs[num_inters] = x[i] +
                         (ys-yi)*(x[(i+1)%np]-x[i])/(yf-yi);
       num_inters++;
       }
     }

   ordena(vxs,0,num_inters-1);          /* ordena as interseções */

   for (i=0;i<num_inters;i+=2)
          if (vxs[i]+1 <= vxs[i+1])   ScanLine(vxs[i],vxs[i+1],ys);
}
```

Otimizações do algoritmo de *fill*



Lista de Arestas

```
struct edge
{
  int     y_max;      /* maior y da aresta */
  int     y_min;      /* menor y da aresta */
  float   xs;         /* x correspondente a ys */
                      /* (no início é o correspondente a y_max) */
  float delta_xs; /* incremento de xs entre duas linhas de scan */
};
```

Algoritmo de Fill de Polígonos (Parte 1-Alocação de Memória)

```
#define Max(a,b)    (((a) > (b)) ? (a) : (b))
#define Min(a,b)    (((a) < (b)) ? (a) : (b))

void fill (int np, int *x, int *y)
{
  static struct edge *aresta=NULL;      /* vetor de arestas */
  static int *vxs=NULL;                 /* vetor de interseções */
  static int old_np=0;    /* número de pontos da última chamada */

  int ymax, ymin;   /* limites do polígono */
  int num_inters;   /* num. de interseções */
  int num_arestas;  /* num. de arestas */
  int ys;           /* ordenada da reta de scan */
  int i;

/* realoca os vetores de arestas e de interseções */
  if (np > old_np)
  {
   old_np=np;
   if (vxs) free (vxs);
   if (aresta) free (aresta);
   vxs=(int *) malloc ((np-1)*sizeof(int)); /* max num. De inters.*/
   aresta=(struct edge *) malloc (np*sizeof(struct edge));
  }

/* CONTINUA NA PARTE 2 */
```

Algoritmo de Fill de Polígonos (Parte 2-Lista de Arestas)

```
/* PARTE 1*/

/* calcula y max e monta o vetor de arestas */
  ymax = y[0];
  ymin = y[0];
  num_arestas = 0;
  for(i=0;i<np;i++)
  {
    int i1=(i+1)%np;
    if (y[i] != y[i1])
      {
          aresta[num_arestas].y_max = Max(y[i],y[i1]);
          aresta[num_arestas].y_min = Min(y[i],y[i1]);
          aresta[num_arestas].delta = ((float)(x[i1]-x[i])/
                                       (float)(y[i1]-y[i]));
          if (aresta[num_arestas].y_mim == y[i])
            aresta[num_arestas].xs = x[i];
          else
            aresta[num_arestas].xa = x[i1];

          if (aresta[num_arestas].y_max > ymax)
             ymax = aresta[num_arestas].y_max;
          if (aresta[num_arestas].y_min < ymin)
             ymin = aresta[num_arestas].y_min;
          num_arestas++;
      }
  }
/* CONTINUA NA PARTE 3 */
```

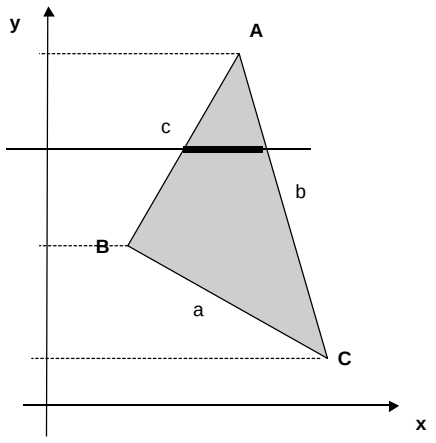
Algoritmo de *Fill* de Polígonos (Parte 3-Varredura)

```
/* PARTES 1 E 2 */

 for(ys=ymin; ys<ymax; ys++) /* para cada linha de scan */
  {
   num_inters = 0;
   for(i=0; i<num_arestas; i++)
    {
     if (aresta[i].y_max < ys){   /* retira da lista de arestas */
       aresta[i] = aresta[num_arestas-1];
       num_arestas--;
      }

     if((ys>=aresta[i].y_min)&&(ys<aresta[i].y_max)){ /* intersepta */
       vxs[num_inters] = aresta[i].xs;
       aresta[i].xs += aresta[i].delta; /* atualiza o xs */
       num_inters++;
      }
    } /* for */

   ordena(vxs,0,num_inters-1);          /* ordena as interseções */

   for(i=0;i<num_inters;i+=2)
     if (vxs[i]+1 <= vxs[i+1])   hline(vxs[i],vxs[i+1],ys,0xff);
    }
} /* fill */

/* FIM */
```

Ordenação no Algoritmo de *Fill*

```
static void ordena(int *vxs, int left, int right)
{
  int i,j;
  int a;

  i = left;
  j = right;
  a = vxs[(left + right)/2];
  do
    {
      while (vxs[i] < a && i < right) i++;
      while (a < vxs[j] && j > left)  j--;
      if (i<=j)
          {
            int b = vxs[i];
            vxs[i] = vxs[j];
            vxs[j] = b;
            i++;j--;
          }
    } while (i<=j);
  if (left < j)  ordena(vxs,left,j);
  if (i < right) ordena(vxs,i,right);
}
```

Caso Particular: Triângulo



Stipples, patterns e imagens

Stipple

```
void SetPixel(int x,int y)
{
  int i=(x-x0)%w;
  int j=(y-y0)%h;

  if (stipple[i][j]) {
      Pixel(x,y,foreground);
  } else {
      if (backopacity) Pixel(x,y,background);
  }
}
```

h = 5

w = 5

Pattern

```
void SetPixel(int x,int y)
{
  color = pattern[(x-x0)%w][(y-y0)%h]
  Pixel(x,y,color);
}
```