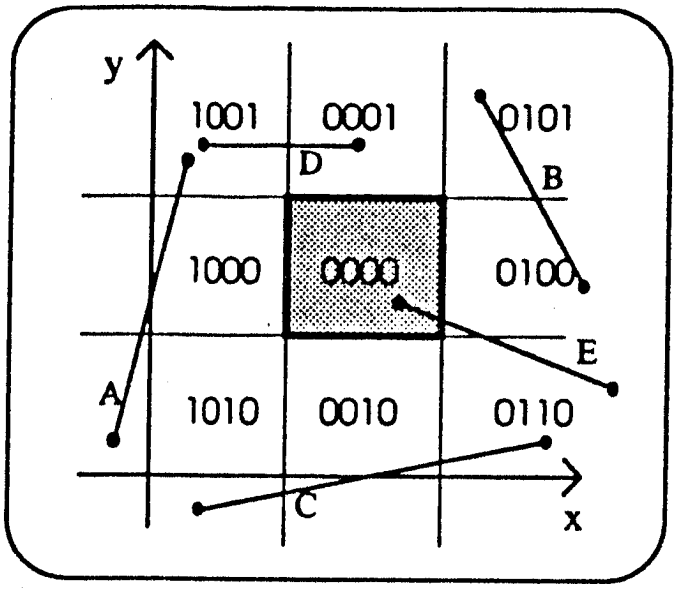
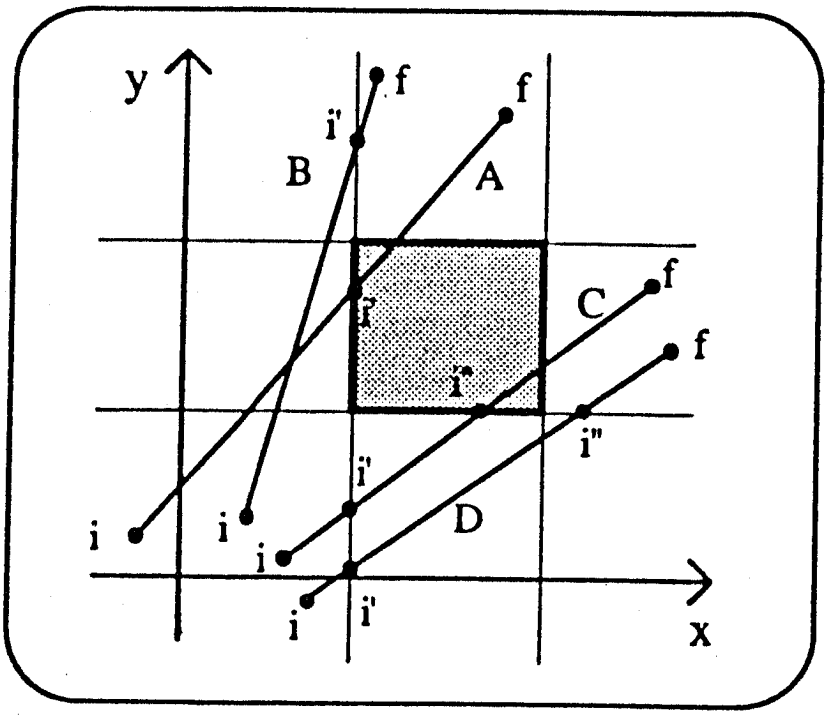
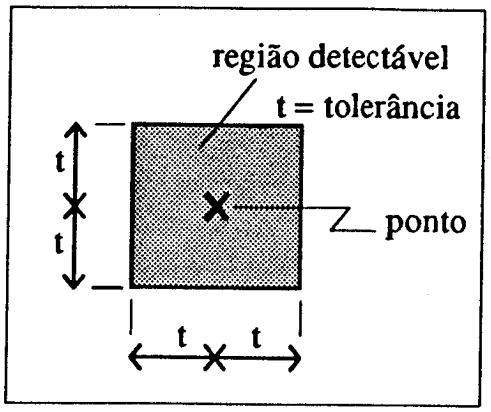


Regiões com o mesmo Código e Posições Triviais

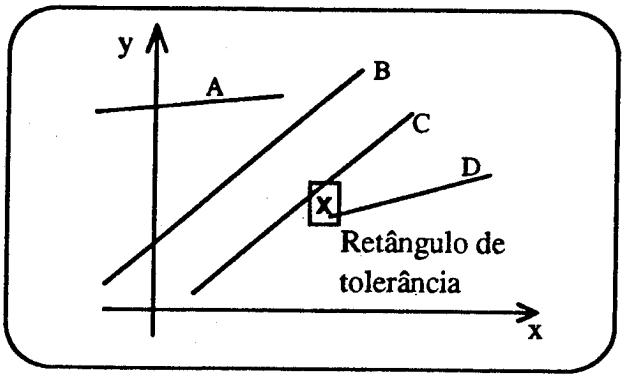


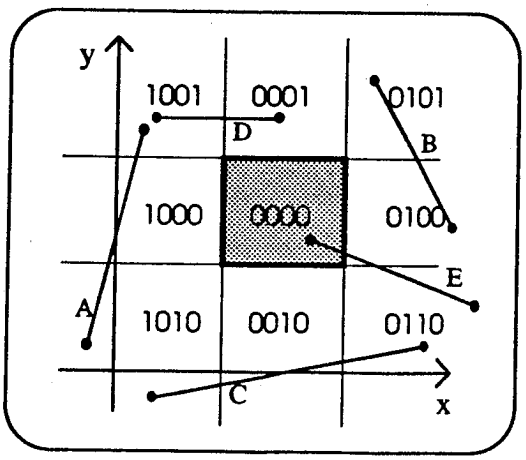
Algoritmo recursivo para recair nos casos triviais

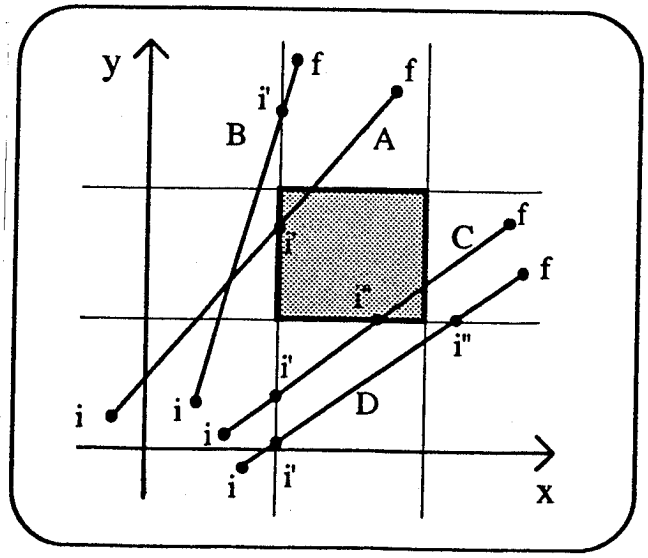
Pick de Ponto



Pick de Segmento de Reta

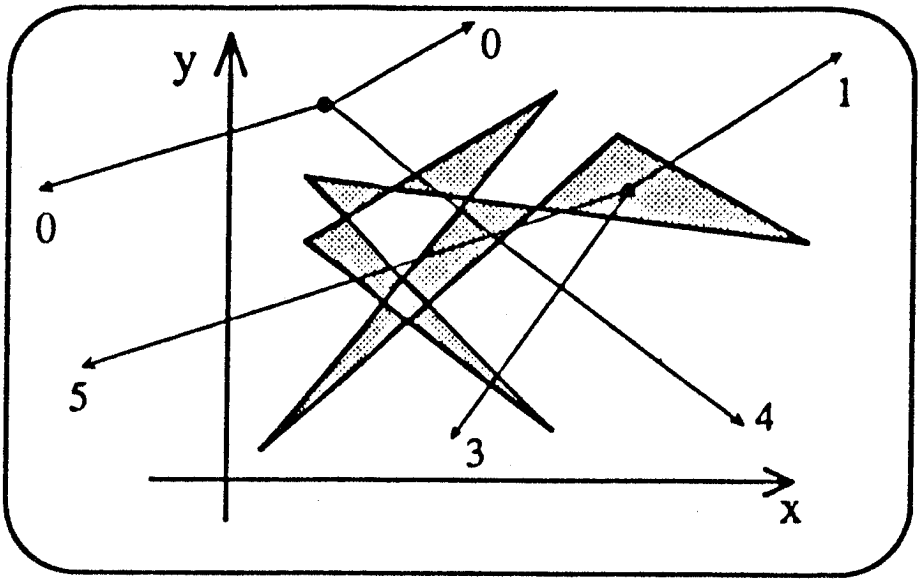






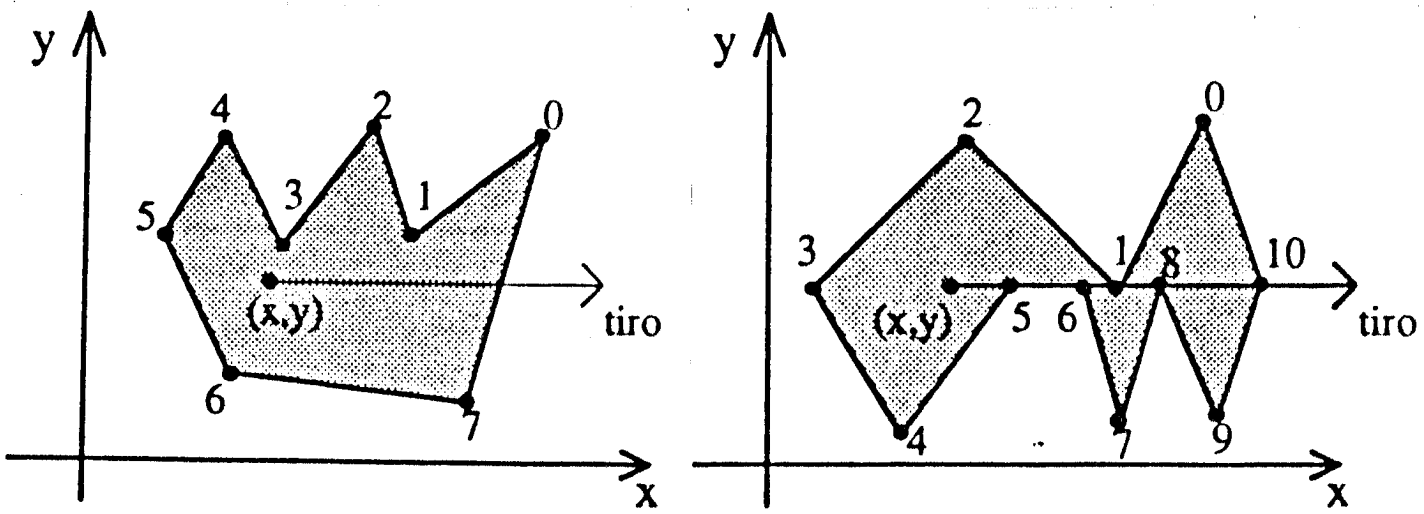
```
/*
** Calcula codigo para pick de linha
*/
void PickCode
(float x,float y,float xmin,float xmax,float ymin,float ymax,int cod[4])
{
 cod[0] = x < xmin;
 cod[1] = x > xmax;
 cod[2] = y < ymin;
 cod[3] = y > ymax;
}

/*
** Pick de linha poligonal
*/
int PkLine (int n, GraPnt line[], float x, float y, float tol)
{
 int    i, j;
 int    cod0[4], cod1[4];
 float  x0, y0, x1, y1;
 float  xmin, xmax, ymin, ymax;

 /* define janela de atracao */
 xmin = x - tol;
 xmax = x + tol;
 ymin = y - tol;
 ymax = y + tol;

 /* testa cada segmento */
 for (i = 1; i < n; i++)
 {
  x0 = line[i-1].x;
  x1 = line[i].x;
  y0 = line[i-1].y;
  y1 = line[i].y;

  PickCode (x1, y1, xmin, xmax, ymin, ymax, cod1);
  do
  {
   PickCode (x0, y0, xmin, xmax, ymin, ymax, cod0);
   for ( j = 0; j < 4; j++ )
    if (cod0[j] && cod1[j])     /* test no-trivial pick */
     break;
   if (j != 4)
    break;
   /* move point 0 to window limit */
   if (cod0[0])
    y0 += (xmin - x0) * (y1 - y0) / (x1 - x0), x0 = xmin;
   else if (cod0[1])
    y0 += (xmax - x0) * (y1 - y0) / (x1 - x0), x0 = xmax;
   else if (cod0[2])
    x0 += (ymin - y0) * (x1 - x0) / (y1 - y0), y0 = ymin;
   else if (cod0[3])
    x0 += (ymax - y0) * (x1 - x0) / (y1 - y0), y0 = ymax;
   else
    return 1;
  } while (1);
 }
 return 0;
}
```

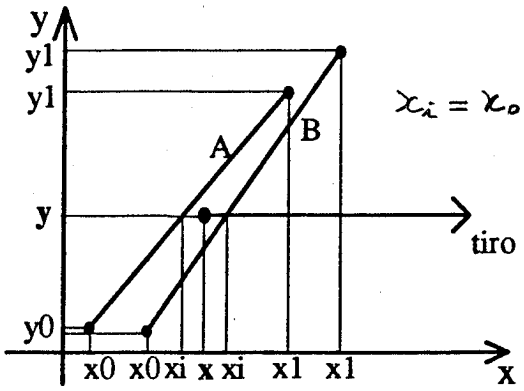
Preenchimento de Área e Localização de Ponto em Área



Critério para Definir o Interior de uma Área

Tiro para Determinação do Número de Interseções





$$x_i = x_0 + \frac{(y - y_0)(x_1 - x_0)}{(y_1 - y_0)}$$

```
/*
** Pick de area
*/
int PkArea (int n, GraPnt area[], float x, float y)
{
 int     i;
 int     ni = 0;                  /* numero de intersecoes */
 int     fst = n - 1;             /* comeca pelo ultimo no' */
 float   xc;
 GraPnt  p1, p2;                  /* pontos da aresta */

 for (i = 0; i < n; i++)
 {
  p1 = area[i];
  p2 = area[fst];
  if (!(p1.y == p2.y) &&                  /* descarta horizontais */
      !((p1.y > y) && (p2.y > y)) &&      /* descarta retas acima */
      !((p1.y < y) && (p2.y < y)) &&      /* descarta retas abaixo */
      !((p1.x < x) && (p2.x < x)))        /* descarta retas esquerda */
  {
   if (p1.y == y)                         /* primeiro ponto na mesma cota */
   {
    if ((p1.x > x) && (p2.y > y))
     ni++;                                /* a direita e acima do ponto */
   }
   else
   {
    if (p2.y == y)                 /* segundo ponto na mesma cota */
    {
     if ((p2.x > x) && (p1.y > y))
      ni++;                        /* a direita e acima do ponto */
    }
    else
    {
     if ((p1.x > x) && (p2.x > x))
      ni++;                        /* inteiramente a direita */
     else
     {                             /* verifica ponto de intersecao */
      float dx = p1.x - p2.x;
      xc = p1.x;
      if ( dx != 0.0 )
        xc += ( y - p1.y ) * dx / ( p1.y - p2.y );
      if (xc > x)
        ni++;
     }
    }
   }
  }
  fst = i;                         /* ultimo ponto para proxima aresta */
 }
 return ( ni % 2 );
}
```