Computação Gráfica I

Professor:

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

Conteúdo:

- Recorte 2D

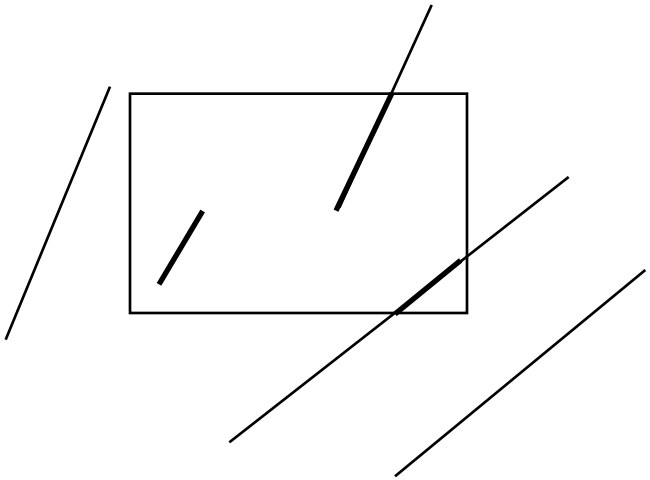
Recorte 2D: *introdução*

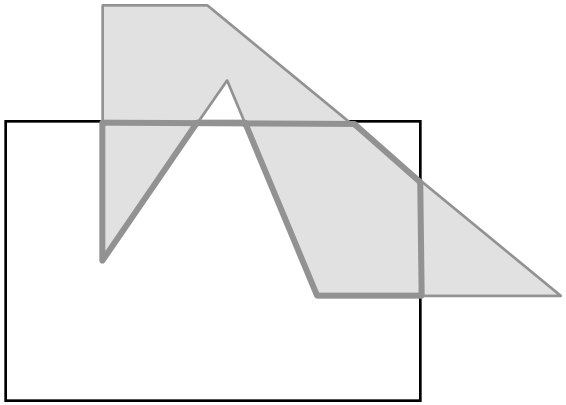
- Quando especificamos uma janela no espaço do objeto queremos exibir somente dos objetos que estejam no seu *interior.*

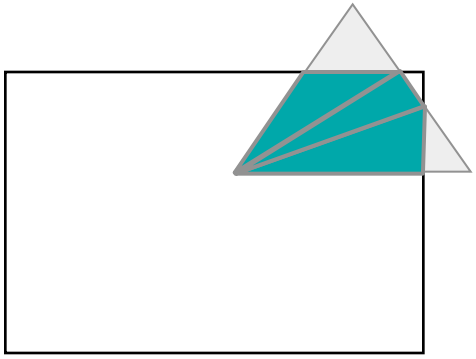
- Naturalmente alguns objetos estarão *parcialmente visíveis*.

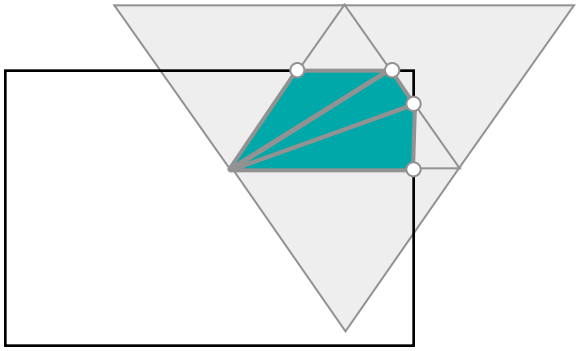
- A operação que determina quais partes dos objetos planares estão visíveis na janela é denominado *recorte 2D*.

Recorte 2D: *introdução*









Recorte 2D: *recorte segmento de reta x retângulo*

- Problema clássico 2D
- Entrada:
  - Segmento de reta P_1P_2
  - Janela alinhada com eixos $(xmin, ymin)$, $(xmax, ymax)$
- Saída: Segmento recortado (possivelmente nulo)
- Algoritmos:
  - Cohen-Sutherland
  - Cyrus-Beck

Recorte 2D: *algoritmo Cohen-Sutherland*

- Vértices do segmento são classificados com relação a cada semi-espaço plano que delimita a janela

$$x \geq xmin \text{ e } x \leq xmax \text{ e } y \geq ymin \text{ e } y \leq ymax$$

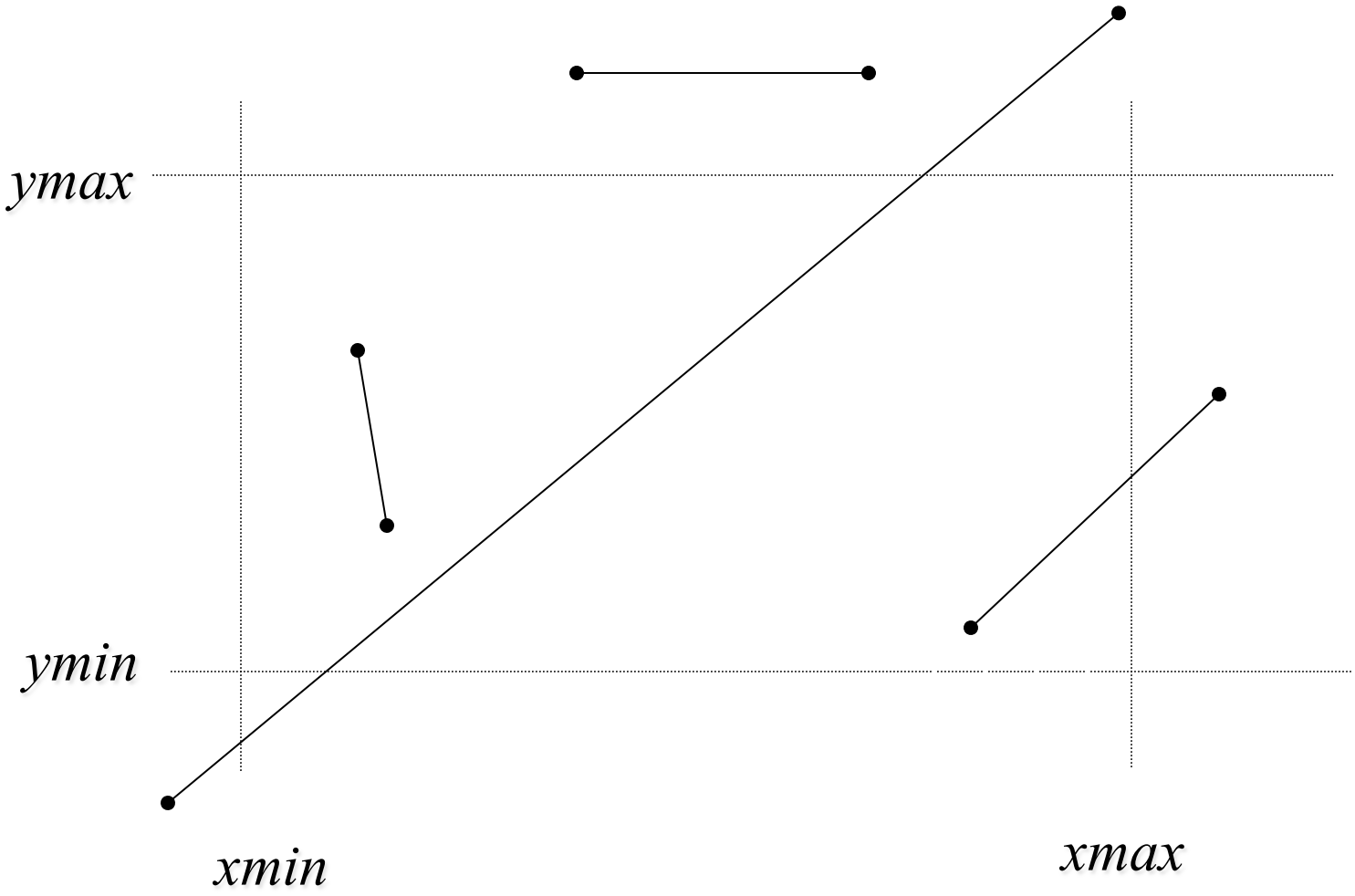
- Se ambos os vértices são classificados como "fora", descartar o segmento (totalmente invisível).

Recorte 2D: *algoritmo Cohen-Sutherland*

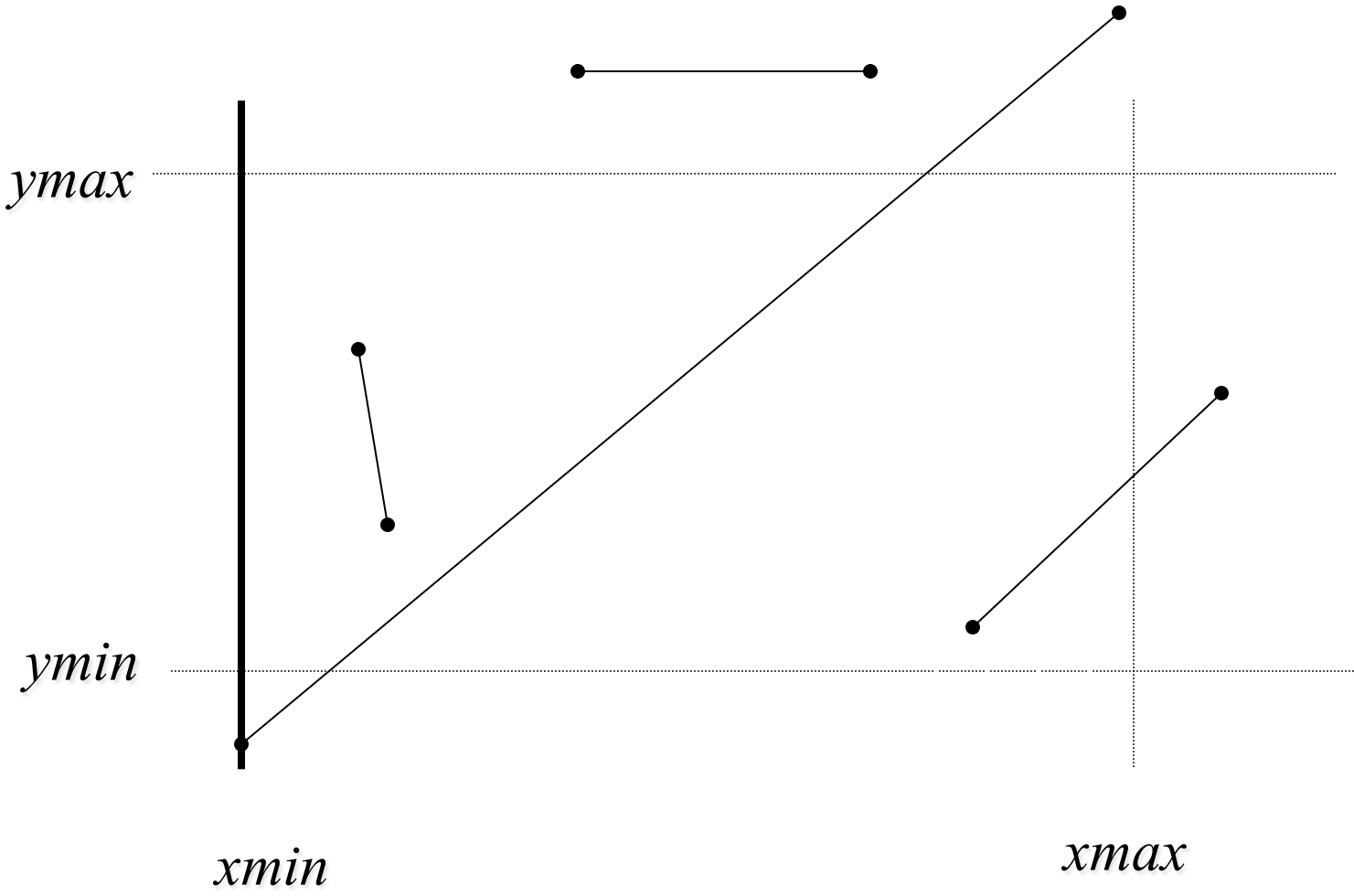
- Se ambos são classificados como "*dentro*", testar o próximo semi-espaço.

- Se um vértice está dentro e outro fora, *computar o ponto de interseção* Q e continuar o algoritmo com o segmento recortado.

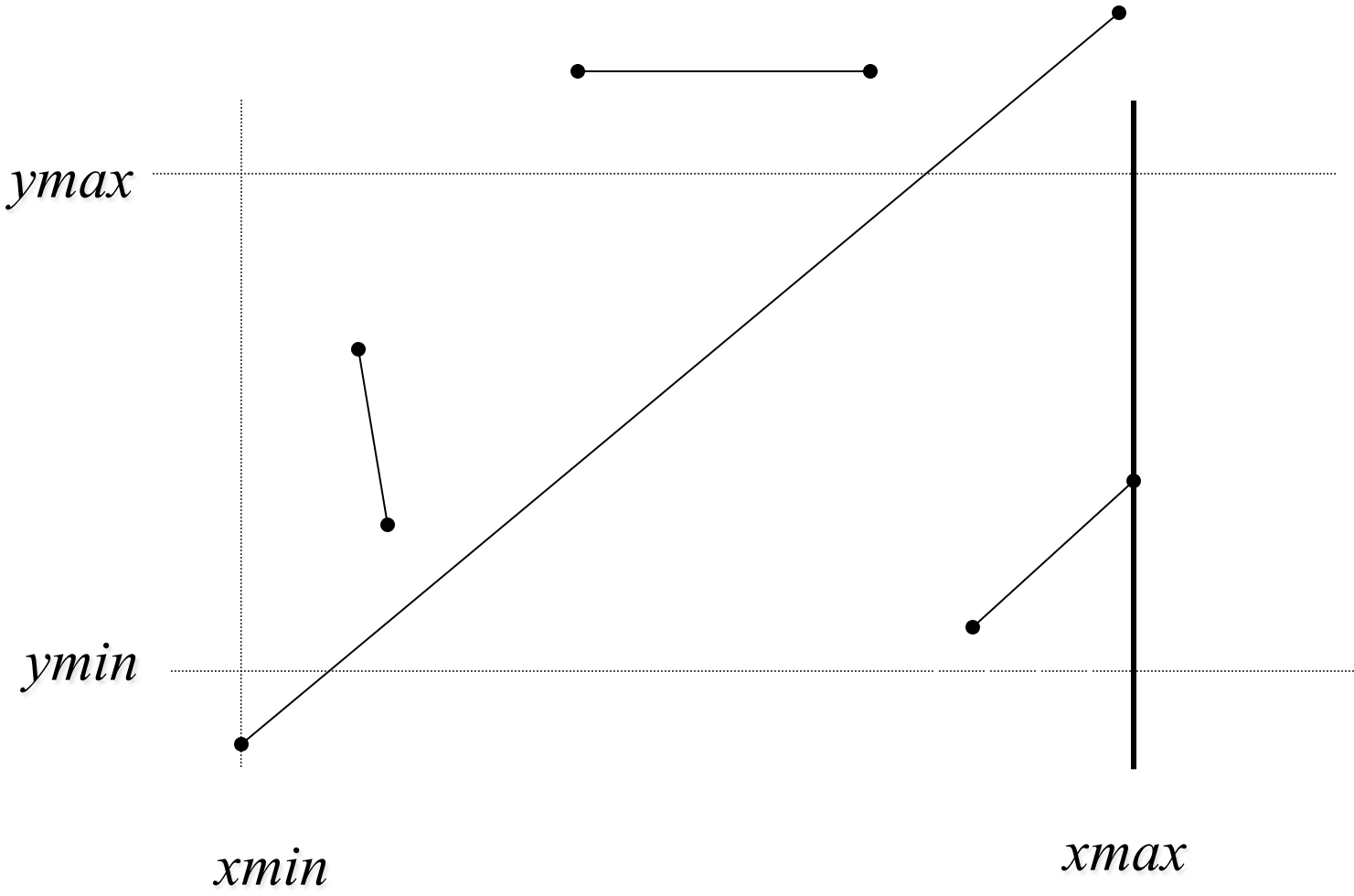
Recorte 2D: *algoritmo Cohen-Sutherland*



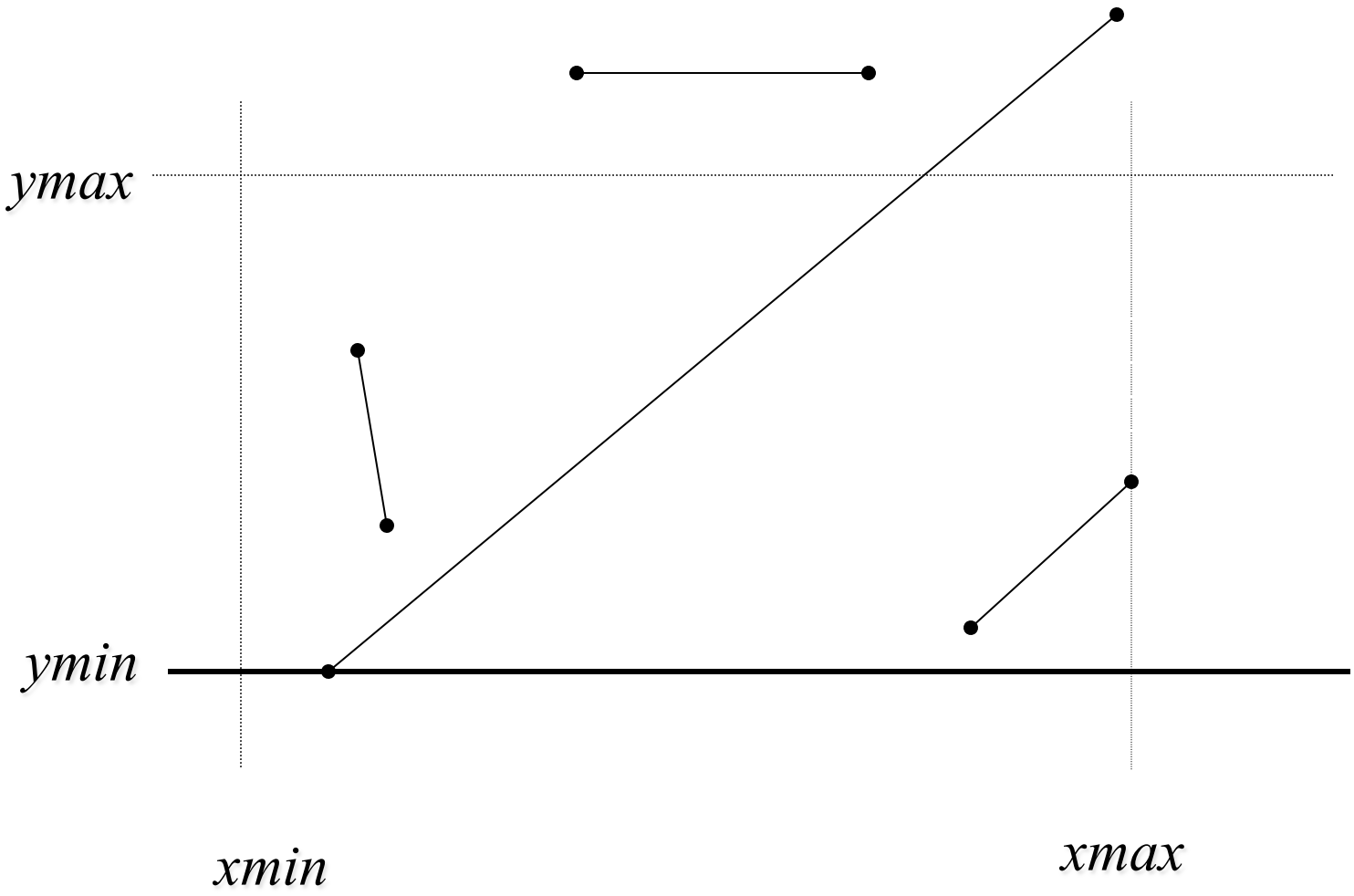
Recorte 2D: *algoritmo Cohen-Sutherland*



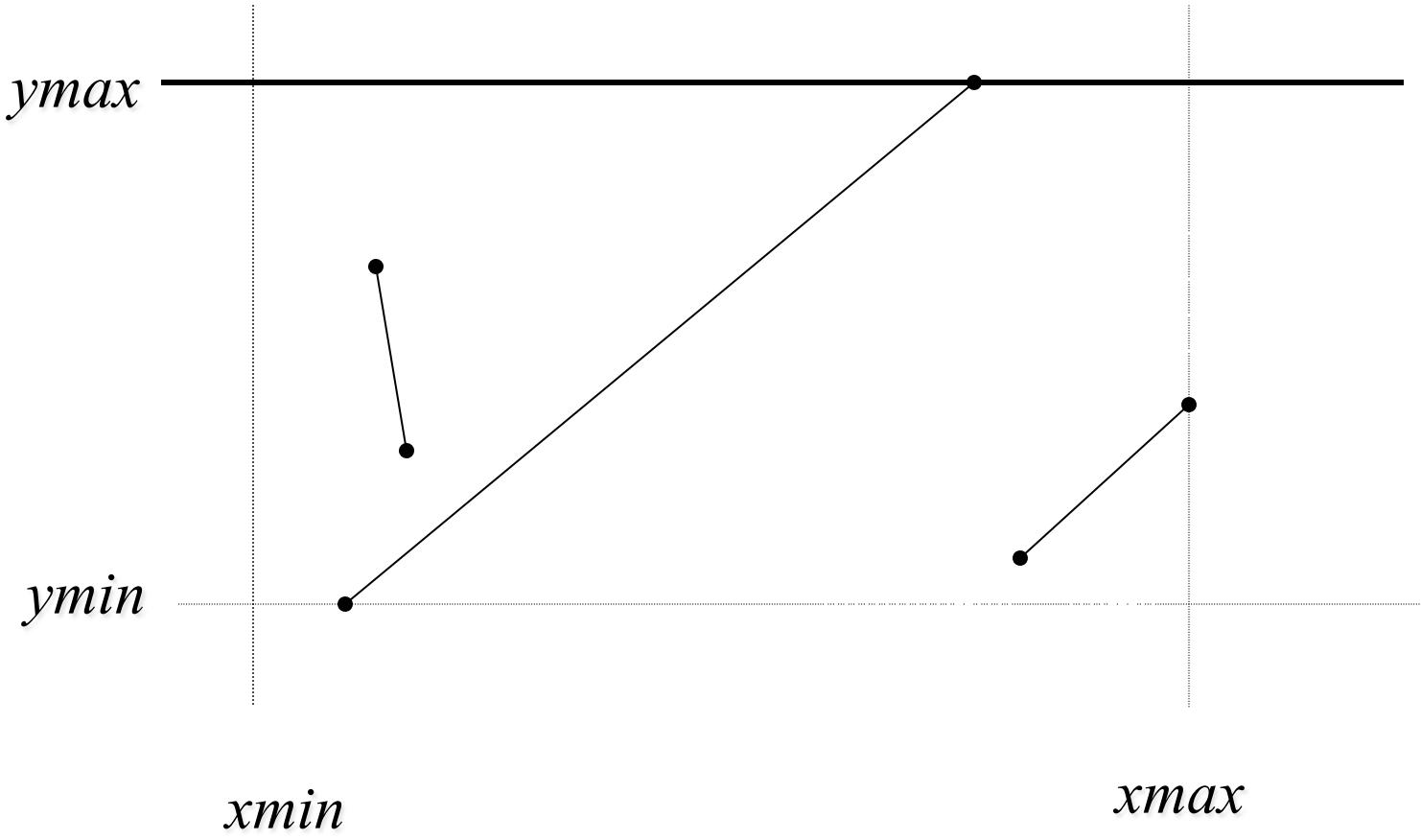
Recorte 2D: *algoritmo Cohen-Sutherland*



Recorte 2D: *algoritmo Cohen-Sutherland*

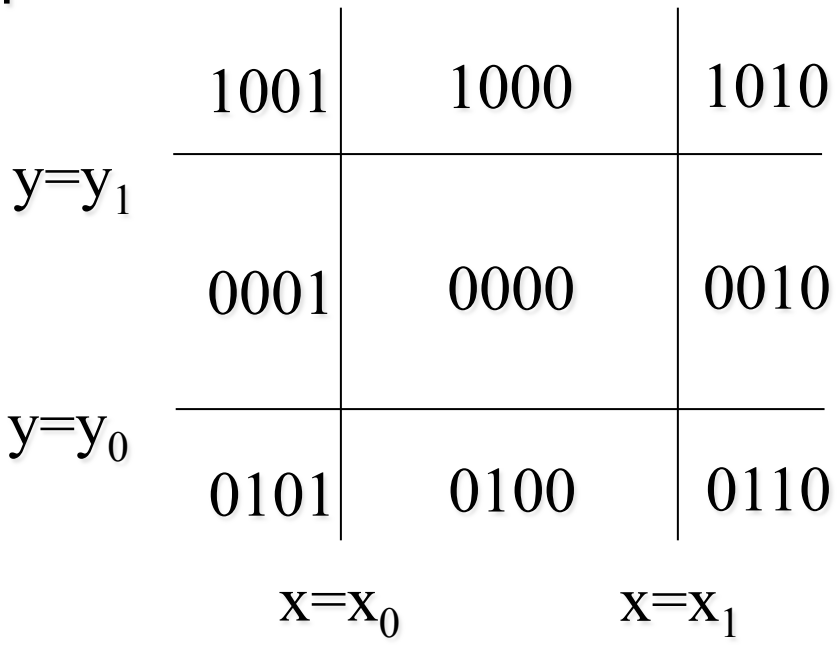


Recorte 2D: *algoritmo Cohen-Sutherland*



Recorte 2D: *algoritmo Cohen-Sutherland*

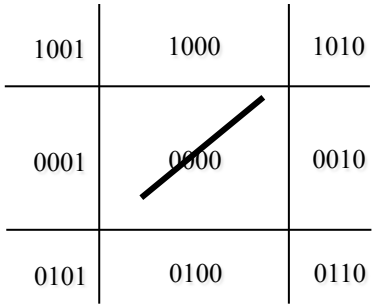
- Recorte só é necessário quando houver um *vértice dentro e outro fora*.
- Classificação de cada vértice *pode ser codificada em 4 bits*, um para cada semi-espaço: Dentro = 0 e Fora = 1



tbrl

Recorte 2D: *algoritmo Cohen-Sutherland*

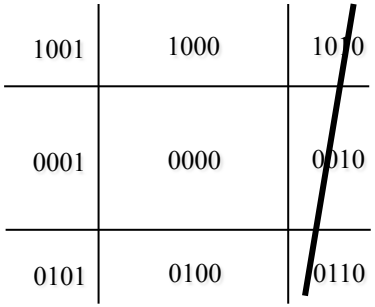
- Rejeição trivial: **Classif(*P*1) and Classif(*P*2) ≠ 0**
- Aceitação trivial: **Classif(*P*1) or Classif(*P*2) = 0**
- Interseção com quais semi-espaços? **Classif(*P*1) xor Classif(*P*2)**



0000 and 0000 = 0000

0000 or 0000 = 0000

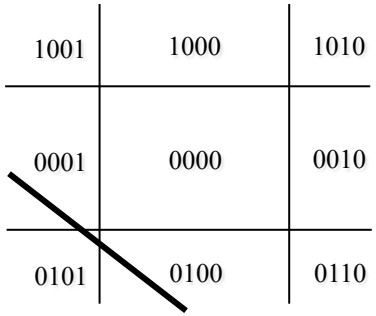
0000 xor 0000 = 0000



1010 and 0110 = 0010

1010 or 0110 = 1110

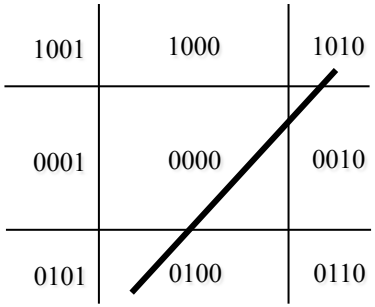
1010 xor 0110 = 1100



0001 and 0100 = 0000

0001 or 0100 = 0101

0001 xor 0100 = 0101



0100 and 1010 = 0000

0100 or 1010 = 1110

0100 xor 1010 = 1110

Recorte 2D: *algoritmo Cohen-Sutherland - cálculo do código de um vértice*

```
unsigned char code(double x, double y,
                   double xmin, double xmax, double ymin, double ymax)
{
   unsigned char code=0;

   if (y > ymax) code += 8;
   if (y < ymin) code += 4;
   if (x > xmax) code += 2;
   if (x < xmin) code += 1;

   return code;
}
```

Recorte 2D: *algoritmo Cohen-Sutherland*

```
void CohenSutherlandLineClip(double x0, double y0, double x1, double y1, double xmin, double xmax, double ymin, double ymax)
{
 unsigned char outcode0, outcode1, outcodeOut;
 double x, y;  boolean accept = FALSE, done = FALSE;

 outcode0 = code(x0, y0, xmin, xmax, ymin, ymax);
 outcode1 = code(x1, y1, xmin, xmax, ymin, ymax);

 do {
   if (outcode0 == 0 && outcode1 == 0) {
       accept = TRUE;  done = TRUE;               /* trivial draw and exit */
   } else if((outcode0 & outcode1) != 0) {
       done = TRUE;                               /* trivial reject and exit */
   } else {                                       /* discart an out part */
     outcodeOut = (outcode0 != 0) ?  outcode0 : outcode1;        /* pick an out vertice */
     if (outcodeOut & 8) {                                       /* discart top */
       x = x0 + (x1 - x0) * (ymax - y0) / (y1 - y0);  y = ymax;
     } else if(outcodeOut & 4) {                                 /* discart bottom */
       x = x0 + (x1 - x0) * (ymin - y0) / (y1 - y0);  y = ymin;
     } else if(outcodeOut & 2) {                                 /* discart right */
       y = y0 + (y1 - y0) * (xmax - x0) / (x1 - x0);  x = xmax;
     } else if(outcodeOut & 1) {                                 /* discart left */
       y = y0 + (y1 - y0) * (xmin - x0) / (x1 - x0);  x = xmin;
     }

     if (outcodeOut == outcode0) {
        x0 = x; y0 = y; outcode0 = code(x0, y0, xmin, xmax, ymin, ymax);
     } else {
        x1 = x; y1 = y;  outcode1 = code(x1, y1, xmin, xmax, ymin, ymax);
     }

   }
  } while (!done);

   if (accept) DrawLineReal(x0, y0, x1, y1);
}
```

Recorte 2D: *algoritmo Cyrus-Beck*

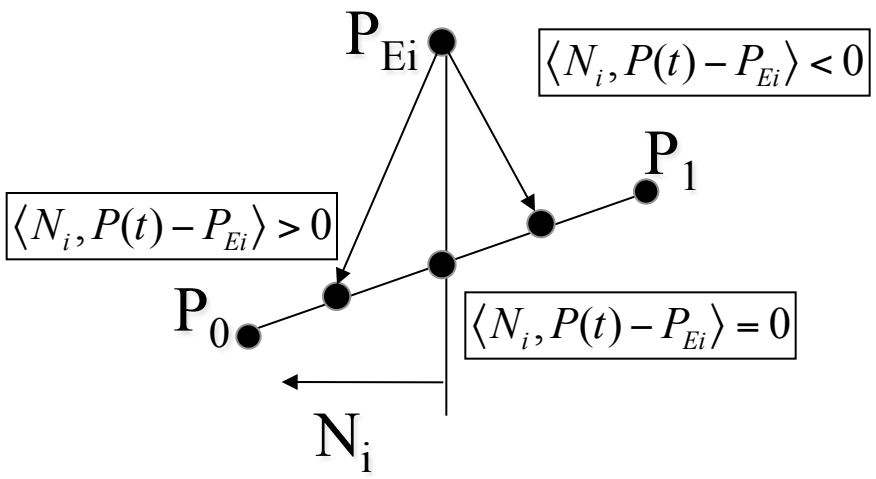
- O algoritmo de Cohen-Sutherland calcula *interseções desnecessárias*.

- O algoritmo de Cyrus-Beck utiliza uma estratégia que permite recortar um segmento com menor número de interseções.

- Pode ser utilizado quando a região de recorte é um *polígono convexo*.

Recorte 2D: *algoritmo Cyrus-Beck*

- Se baseia na *equação paramétrica* da reta e no cálculo de interseções baseado nesta representação.



$$P(t) = P_0 + t(P_1 - P_0)$$

$$\langle N_i, P(t) - P_{Ei} \rangle = 0$$

$$t = \frac{\langle N_i, P_0 - P_{Ei} \rangle}{-\langle N_i, P1 - P0 \rangle}$$

Recorte 2D: *algoritmo Cyrus-Beck*

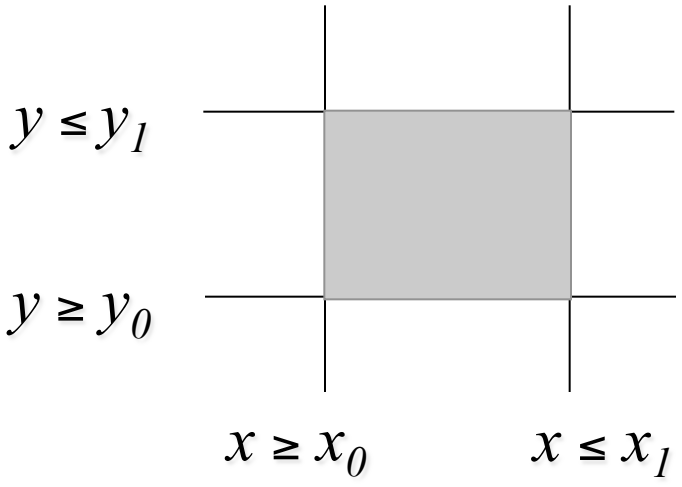
- Dado um segmento P_0P_1, sua reta suporte intersecta as quatro retas laterais do retângulo em *quatro pontos* (caso geral).

- Usando a equação paramétrica *calculamos os quatro valores do parâmetro* t no qual a reta intersecta as quatro retas laterais do retângulo de recorte.

- Após calcular o valor de t para as quatro retas *descartamos os valores de* t *fora do intervalo* $[0,1]$.

Recorte 2D: *algoritmo Cyrus-Beck*

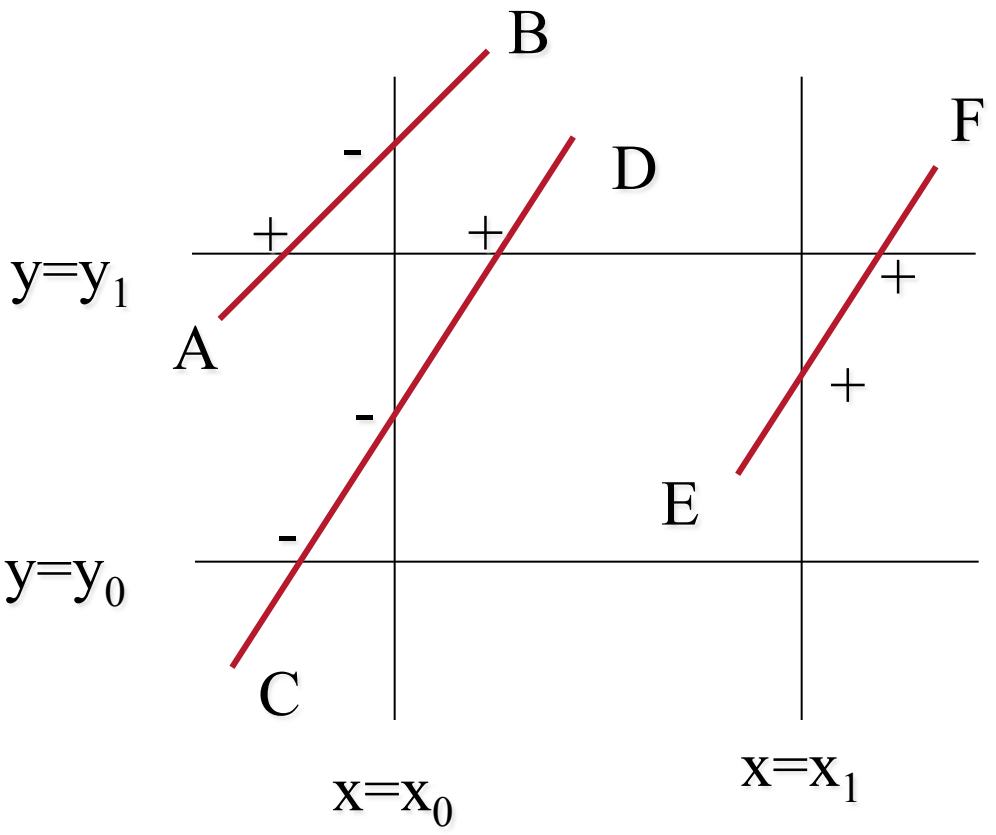
- Cada uma das quatro retas divide o espaço em dois semi-planos.
- Chamaremos de semi-planos positivos aqueles dados pelas equações $y \leq y_1$, $y \geq y_0$, $x \leq x_1$, $x \geq x_0$.



Recorte 2D: *algoritmo Cyrus-Beck*

- Quando a reta $P(t)$ entra em um dos semi-planos através do ponto $P(t_0)$, tal ponto é identificado com um sinal - (tE).
- Se o ponto for de saida indicamos com sinal $+(tL)$.
- O sinal do ponto é obtido usando o sinal do produto interno $<N_i,P_1-P_0>$.
- Se $<N_i,P_1-P_0><0$ então o ponto é negativo, caso contrário é positivo.

Recorte 2D: *algoritmo Cyrus-Beck*



Recorte 2D: *algoritmo Cyrus-Beck – pseudo-código*

```
{
   Calcule Ni e escolha um PEi para cada aresta

     tE = 0;
     tL = 1;
     for (cada aresta ){
        if (Ni.(P1-P0)!=0 ){ /* aresta não é paralela ao segmento */
           calcule t;
           use sign of Ni.(P1-P0) para categorizar como PE ou PL;
               if( PE ) tE = max(tE, t);
               if( PL ) tL = min(tL, t);
        } else {   /* aresta paralela ao segmento */
               if (Ni.(P0-PEi) > 0) /* está fora */
                   return nil;
        }
        if (tE > tL)
            return nil;
        else
            return P(tE) and P(tL) as true clip intersections;
      }
   }
}
```

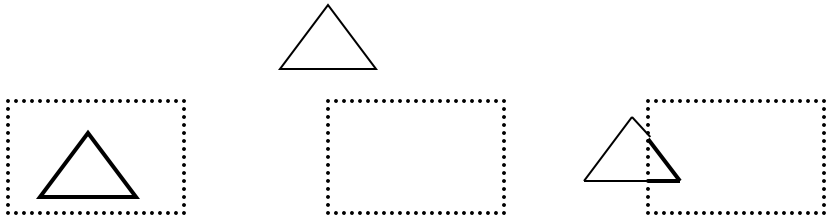
Recorte 2D: *recorte de polígono x retângulo*

- Inclui o problema de recorte de segmentos de reta.
  - Polígono resultante tem vértices que são:
    - Vértices da janela,
    - Vértices do polígono original, ou
    - Pontos de interseção aresta do polígono/aresta da janela

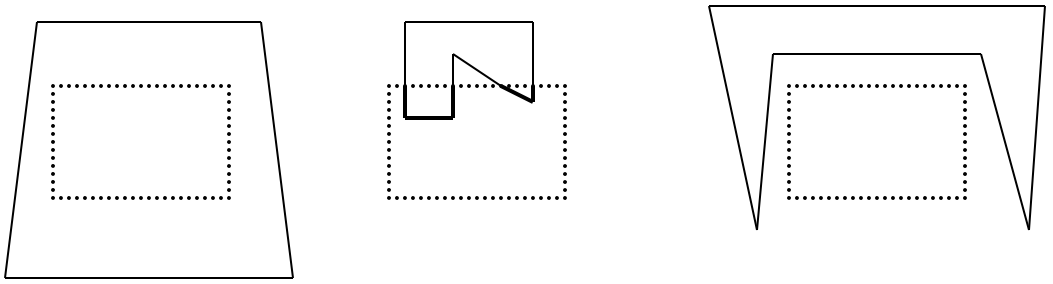
- Dois algoritmos clássicos
  - *Sutherland-Hodgman*
    - Figura de recorte pode ser qualquer polígono convexo.
  - *Weiler-Atherton*
    - Figura de recorte pode ser qualquer polígono.

Recorte 2D: *recorte de polígono x retângulo*

- Casos Simples

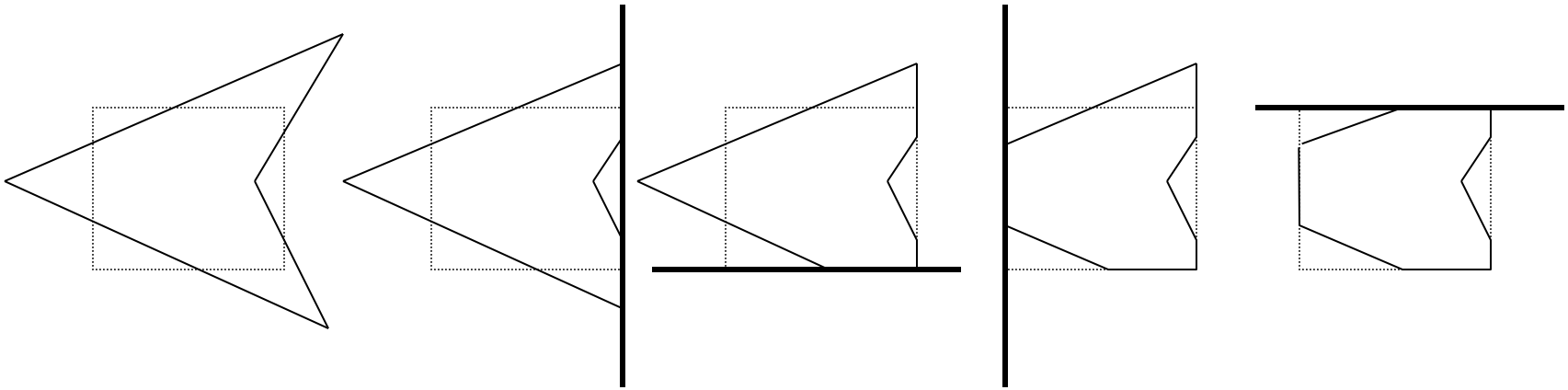


- Casos Complicados



Recorte 2D: *Algoritmo Sutherland-Hodgeman*

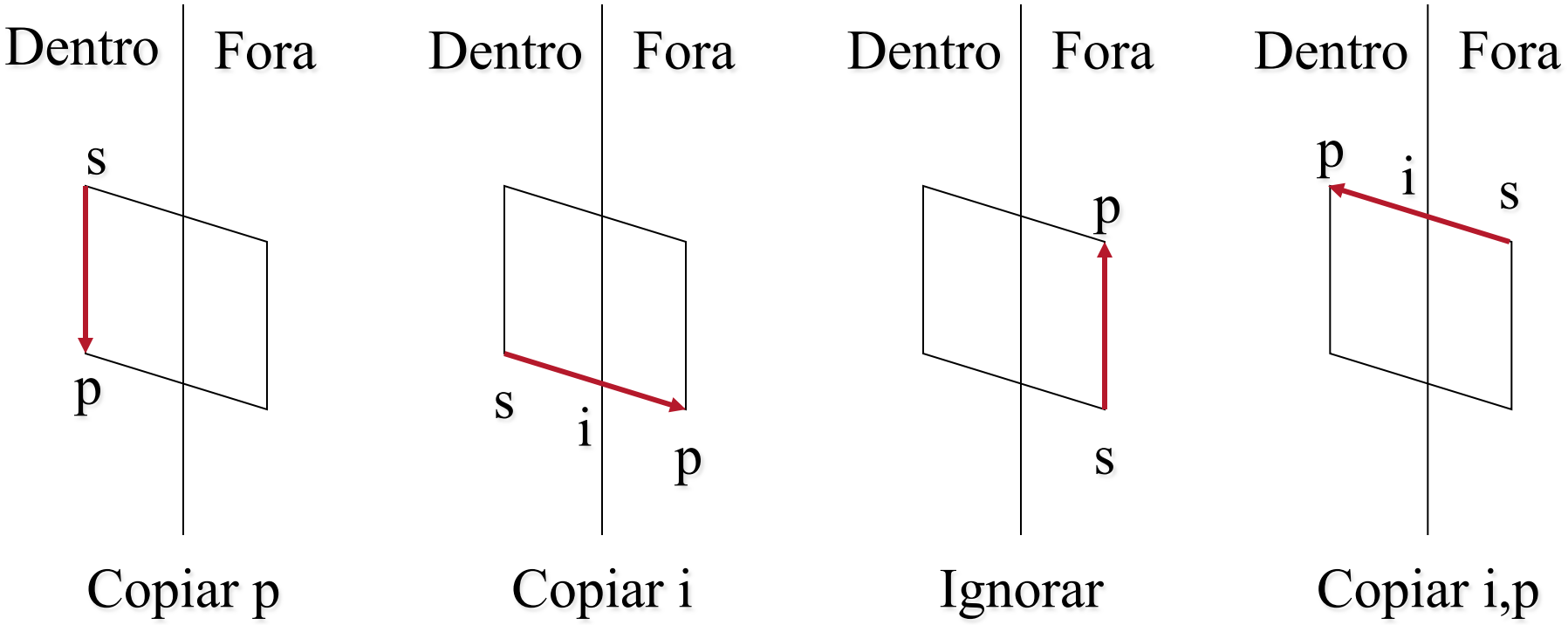
- Idéia é semelhante a do algoritmo de Cohen-Sutherland.
  - Recortar o polígono sucessivamente contra todos os semi-espaços planos da figura de recorte.



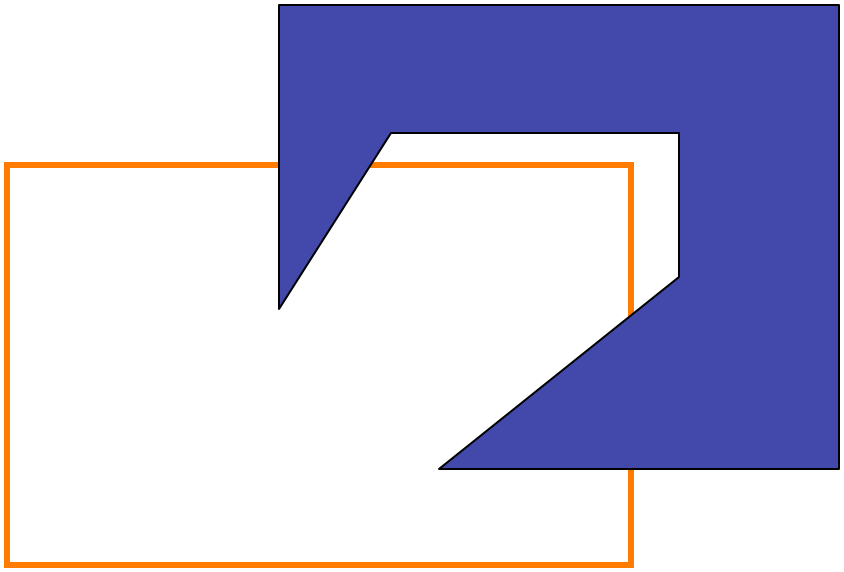
Recorte 2D: *Algoritmo Sutherland-Hodgeman*

- Polígono é dado como uma lista circular de vértices.
- Vértices e arestas são processados em seqüência e classificados contra o semi-espaço plano corrente.
  - Vértice:
    - Dentro: copiar para a saída.
    - Fora: ignorar.
  - Aresta
    - Intercepta semi-espaço plano (vértice anterior e posterior têm classificações diferentes): copiar ponto de interseção para a saída.
    - Não intercepta: ignorar.

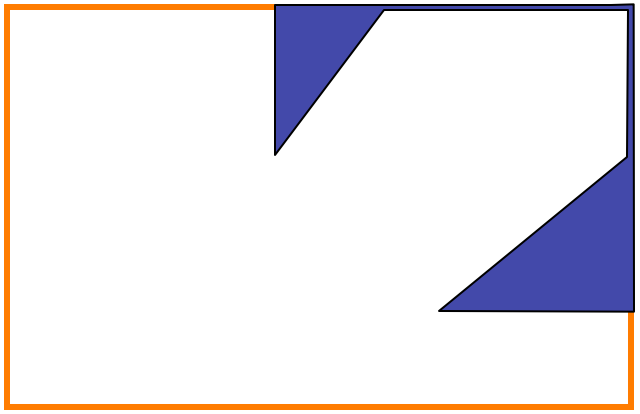
Recorte 2D: *Algoritmo Sutherland-Hodgeman*



Recorte 2D: *Algoritmo Sutherland-Hodgeman - exemplo*

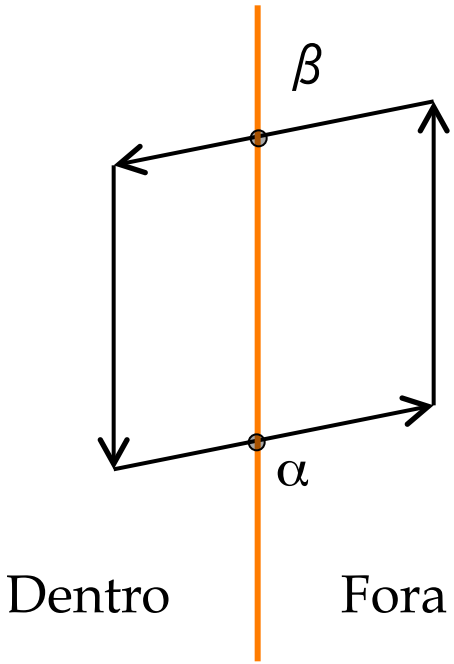


Recorte 2D: *Algoritmo Sutherland-Hodgeman - exemplo*

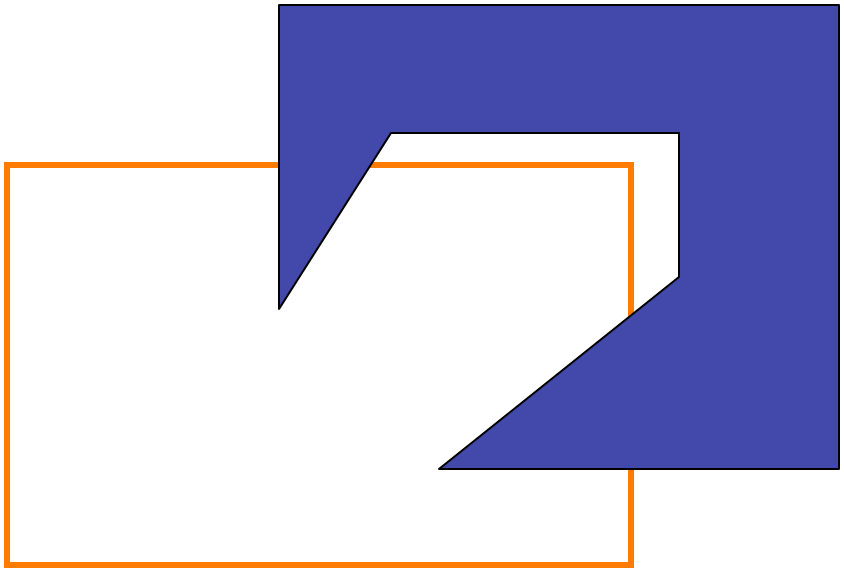


Recorte 2D: *eliminando arestas fantasmas*

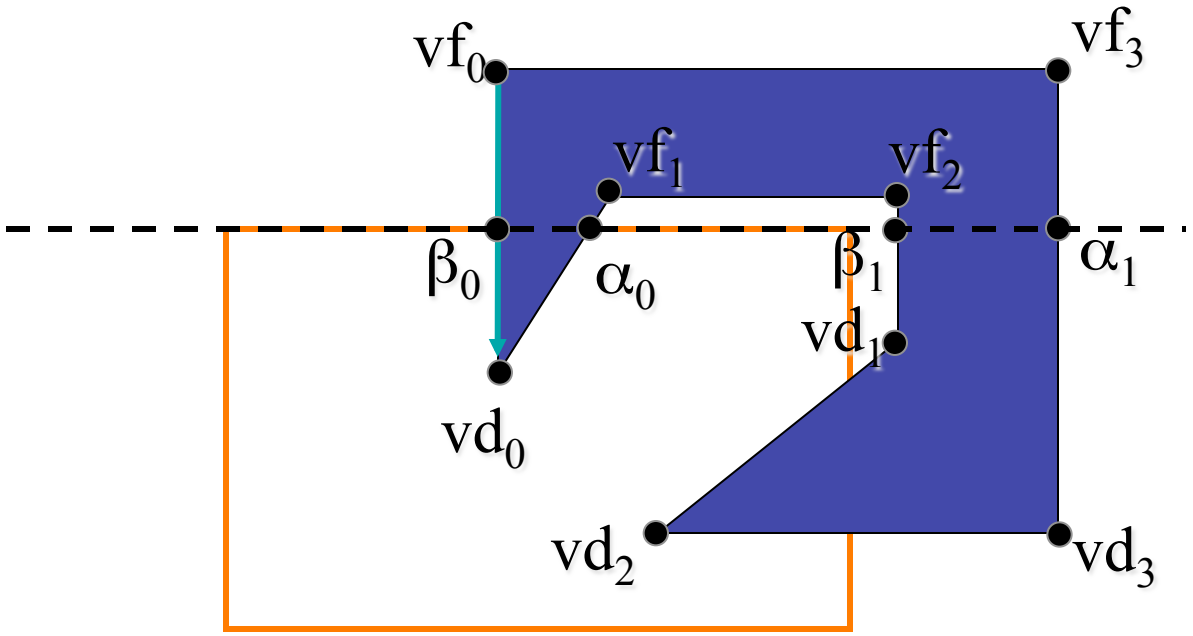
- Distinguir os pontos de interseção gerados:
  - De dentro para fora: rotular como do tipo α.
  - De fora para dentro: rotular como do tipo β.
- Iniciar o percurso de algum vértice "*fora*".
- Ao encontrar um ponto de interseção α, ligar com o último β visto.
- Resultado pode ter mais de uma componente conexa.



Recorte 2D: *eliminando arestas fantasmas - exemplo*



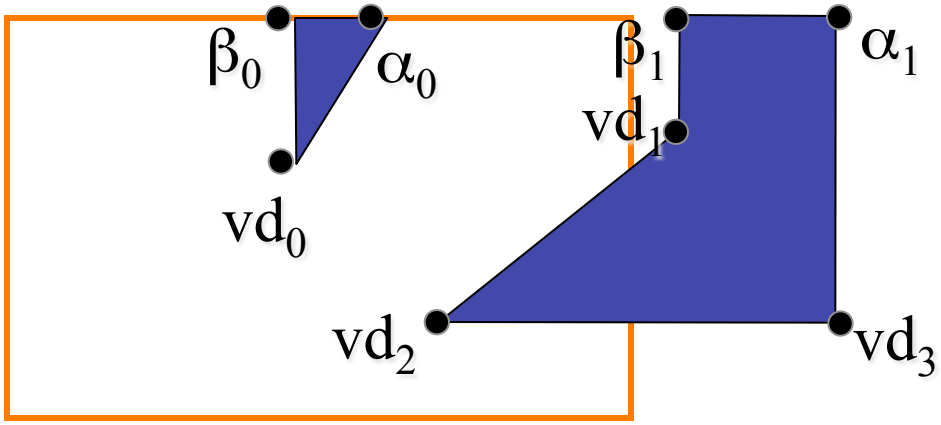
Recorte 2D: *eliminando arestas fantasmas - exemplo*



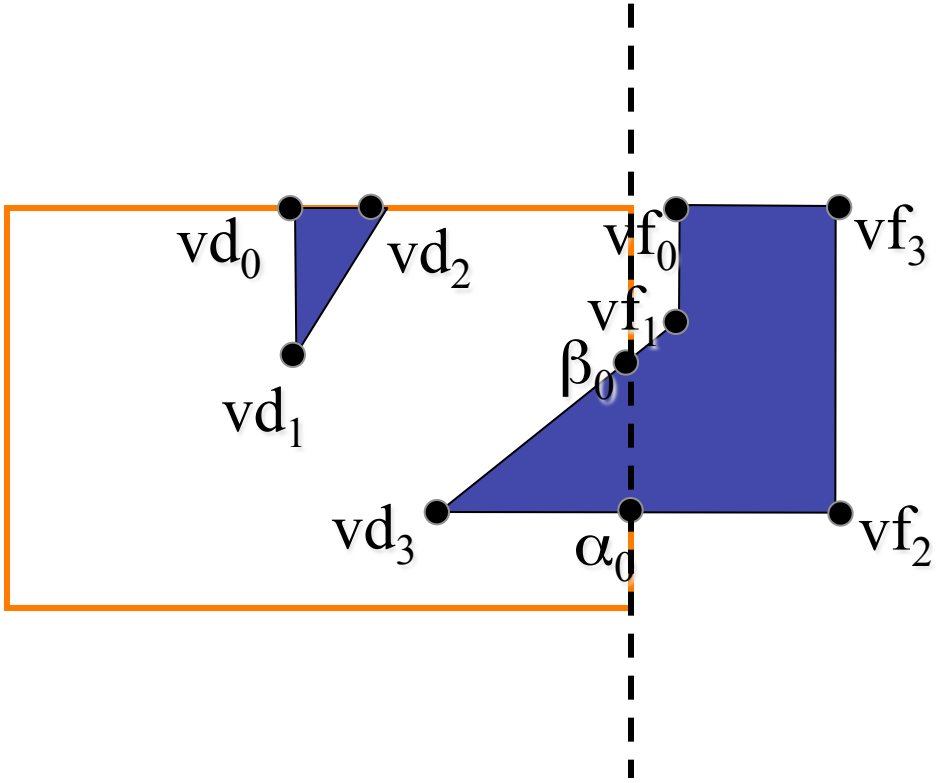
$vf_0\ \beta_0\ vd_0\ \alpha_0\ vf_1\ vf_2\ \beta_1 vd_1\ vd_2\ vd_3\ \alpha_1\ vf_3 vf_0$

$\beta_0\ vd_0\ \alpha_0\ \beta_0$ e $\beta_1\ vd_1\ vd_2\ vd_3\ \alpha_1\ \beta_1$

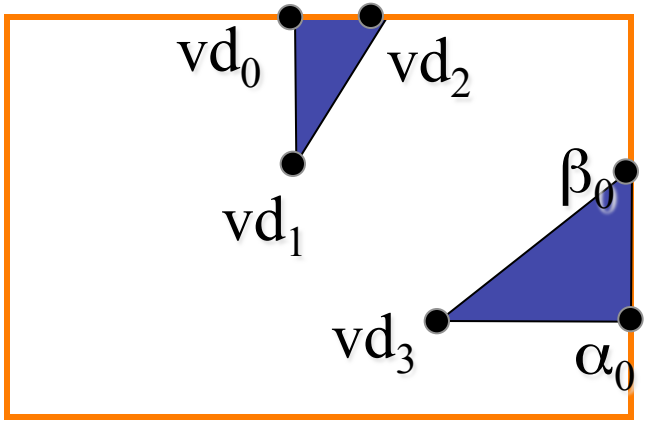
Recorte 2D: *eliminando arestas fantasmas - exemplo*



Recorte 2D: *eliminando arestas fantasmas - exemplo*



Recorte 2D: *eliminando arestas fantasmas - exemplo*



Recorte 2D: *algoritmo Sutherland-Hodgman - resumo*

- Facilmente generalizável para 3D.
- Pode ser adaptado para implementação em hardware.
  - Cada vértice gerado pode ser passado pelo *pipeline* para o recorte contra o próximo semi-espaço plano.
- Pode gerar arestas “fantasma”
  - Irrelevante para propósitos de desenho.
  - Podem ser eliminadas com um pouco mais de trabalho.