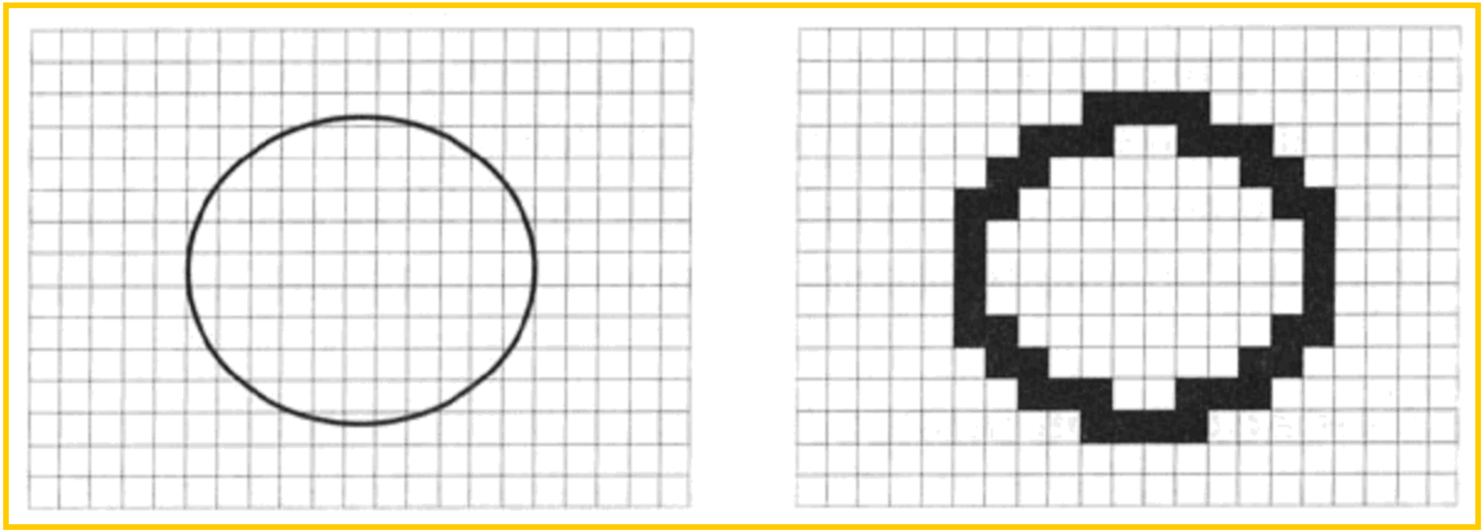
ALGORITMOS RASTER PARA DESENHO DE PRIMITIVAS EM 2D

Adair Santa Catarina
Curso de Ciência da Computação
Unioeste – Campus de Cascavel – PR

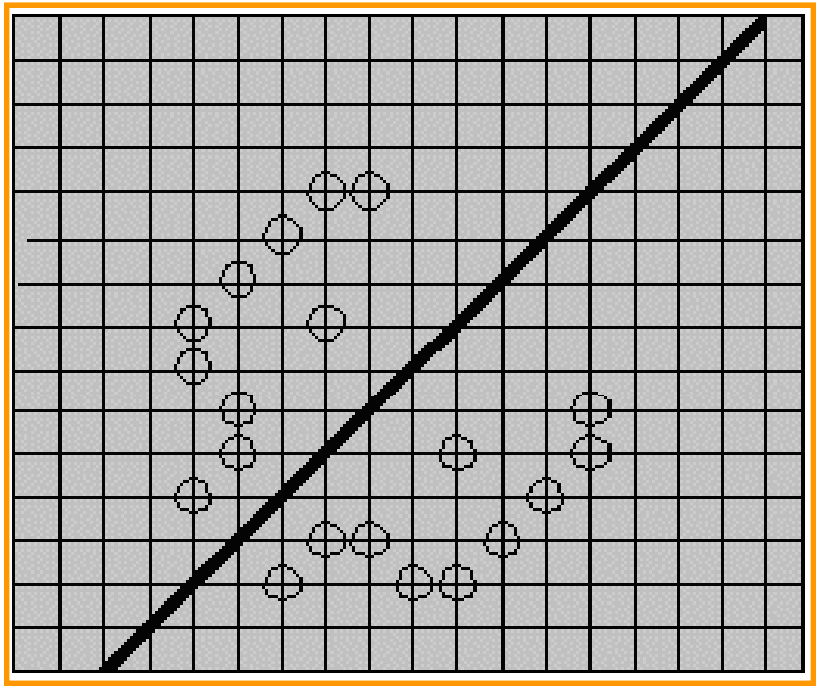
Jan/2021

Algoritmos de conversão matricial



- Convertem um elemento gráfico vetorial em uma representação matricial;
- Implementação em hardware (GPU);
- Implementação em software (assembly e otimizado).

Simetria e reflexão



- Retas horizontais, verticais e diagonais a 45º e 135º → eixos de simetria;
- Qualquer imagem pode ser refletida em relação a estes eixos.

Conversão matricial de segmentos de reta

- Características desejáveis:
  - Linearidade: *pixels* devem dar aparência de que estão sobre uma reta;
  - Precisão: segmentos devem iniciar e terminar nos pontos especificados, sem *gaps* entre segmentos contínuos;
  - Espessura uniforme: *pixels* igualmente espaçados, sem variar a intensidade ou a espessura do segmento ao longo de sua extensão;
  - Intensidade independente da inclinação: segmentos em diferentes inclinações deve manter a mesma intensidade;
  - Continuidade: ausência de *gaps* ao longo do segmento;
  - Rapidez no traçado dos segmentos.

Conversão matricial de segmentos de reta

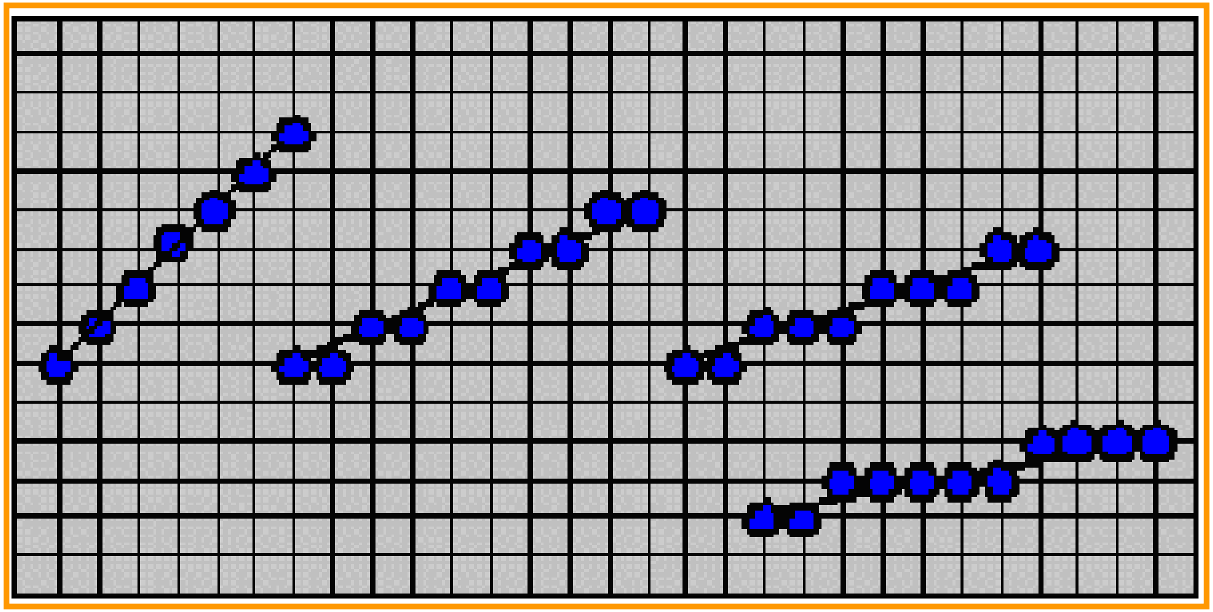
- Critérios adotados:
  - Um segmento de reta é definido por seus extremos (x_1, y_1) e (x_2, y_2);
  - O segmento está no primeiro octante, então os pontos respeitam as relações:

$$\boxed{\begin{gathered} 0 < x_1 < x_2 \\ 0 < y_1 < y_2 \\ y_2 - y_1 < x_2 - x_1 \end{gathered}}$$

  - O segmento de reta corta um número maior de linhas verticais do reticulado do que horizontais.

Conversão matricial de segmentos de reta

- Critério de conversão:
  - Em cada vertical do reticulado com abscissa entre x_1 e x_2 apenas o pixel mais próximo da interseção do segmento com a vertical faz parte de sua imagem.



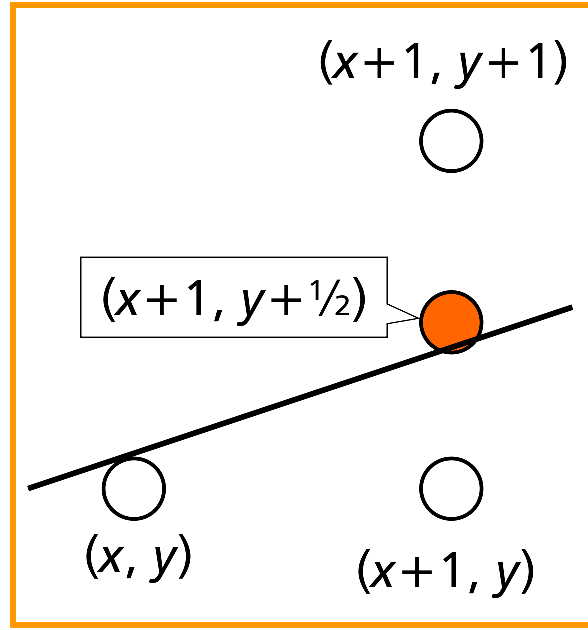
Algoritmo incremental para traçado de retas

$$y = y_1 + m(x - x_1),\ com\ m = \frac{(y_2 - y_1)}{(x_2 - x_1)}$$

```
void Line(int x1, int y1, int x2, int y2, int cor){
//Assume -1<=m <=1 e x1 < x2
   double dy = y2 - y1;
   double dx = x2 - x1;
   double m = dy/dx;
   double y = y1;
   for (x = x1; x <= x2; x++){
      writePixel (x, Round(y), cor);
      y += m;
   }
}
```

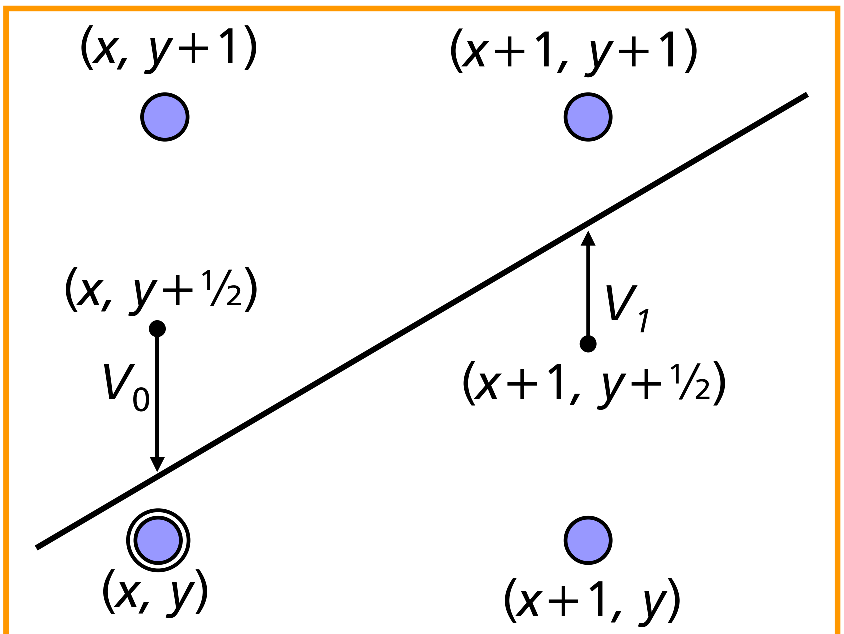
Algoritmo do ponto médio

- Proposto por Bresenham (1965);
- Incremental e utiliza apenas variáveis inteiras;
- Ideia básica:
  - Em vez de computar o valor do próximo y em ponto flutuante, decidir se o próximo pixel vai ter coordenadas (x+1, y) ou (x+1, y+1);
  - Decisão requer que se avalie se a linha passa acima ou abaixo do ponto médio (x+1, y+½).



Algoritmo do ponto médio

- Variável de decisão *V* é dada pela classificação do ponto médio com relação ao semi-espaço definido pela reta;
- Caso 1 – Linha passou abaixo do ponto médio:



$$ax + by + c = V$$

$$\text{onde} \begin{cases} V = 0 \rightarrow (x, y) \text{ sobre a reta} \\ V < 0 \rightarrow (x, y) \text{ abaixo da reta} \\ V > 0 \rightarrow (x, y) \text{ acima da reta} \end{cases}$$

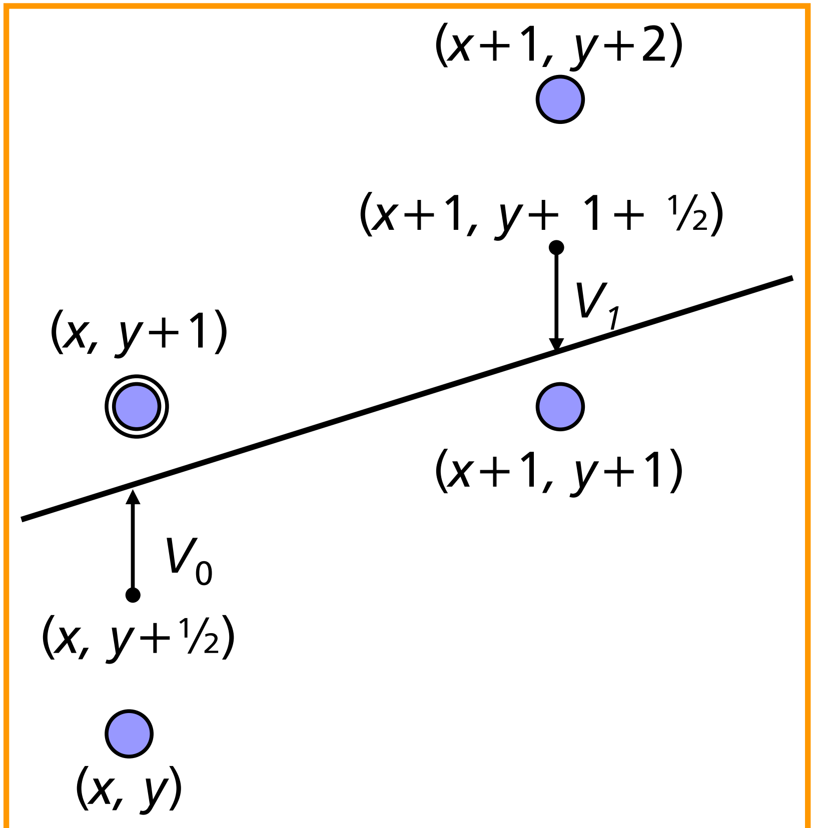
$$V_1 = a(x+1) + b(y + \tfrac{1}{2}) + c$$

$$V_0 = ax + b(y + \tfrac{1}{2}) + c$$

$$\therefore V_1 = V_0 + a$$

Algoritmo do ponto médio

- Caso 2 – Linha passou acima do ponto médio:



$$V_1 = a(x+1) + b(y+1+½) + c$$

$$V_0 = ax + b(y+½) + c$$

$$\therefore V_1 = V_0 + a + b$$

Algoritmo do ponto médio

- Coeficientes da reta:
  - a = y2 – y1
  - b = x1 – x2
  - c = x2.y1 – x1.y2
- Para iniciar o algoritmo, precisamos saber o valor inicial de V
  - V = (a (x1+1) + b(y1+½) + c) – (a(x1) + b(y1) + c)
  - V = a.x1 + b.y1 + c + a + b/2 – a.x1 – b.y1 – c
  - V = a + b/2 .
- Podemos evitar a divisão multiplicando V por 2.

Algoritmo do ponto médio (Bresenham)

```
void MidpointLine(int x1, int y1, int x2, int y2, int cor){
   int a = y2 - y1;
   int b = x1 - x2;
   int V = 2 * a + b; //valor inicial de V
   int incrE = 2 * a; //Mover para E
   int incrNE = 2*(a + b); //Mover para NE
   int x = x1;
   int y = y1;
   WritePixel (x, y, cor); //plota o ponto inicial
   while (x < x2){
      if (V <= 0) V += incrE; //escolhe E
      else{ //escolhe NE
         V += incrNE;
         ++y;
      }
      ++x;
      WritePixel(x, y, cor);  //Plota o ponto final
   }
}
```

Extensão para os demais octantes

- Se $x2 < x1$:
  - Trocar P1 com P2.

- Se $y2 < y1$:
  - $y1 = -y1;$
  - $y2 = -y2;$
  - Pintar pixel $(x, -y)$.

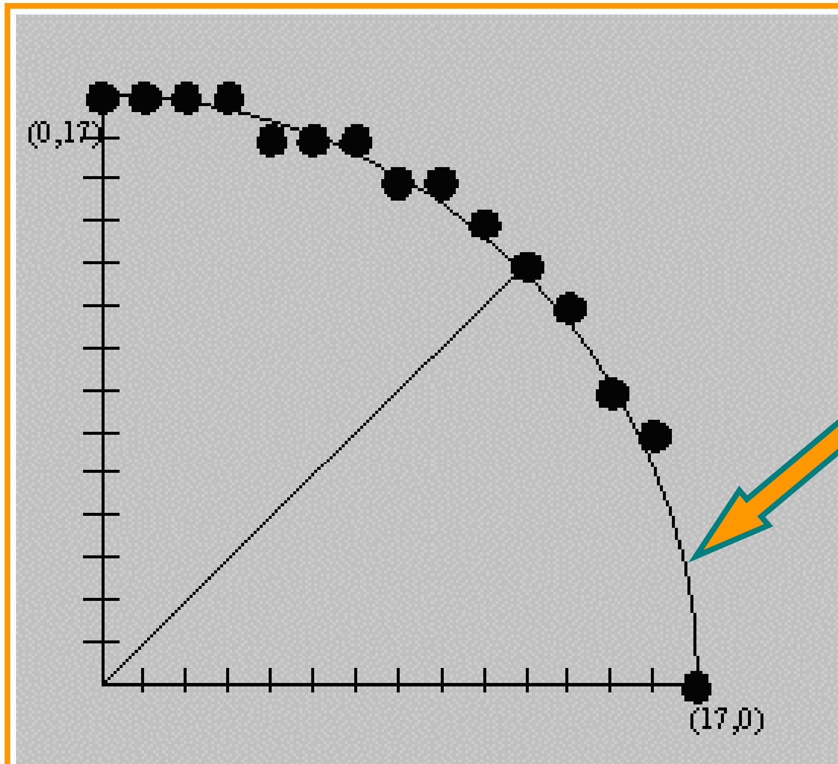
- Se $|y2 - y1| > |x2 - x1|$:
  - Repetir o algoritmo trocando "y" com "x".

Conversão matricial de circunferências

- A circunferência está centrada na origem (0, 0):
  - Quando isso não acontecer aplicar uma translação $(x+C_x, y+C_y)$.

$$x^2 + y^2 = R^2$$

$$y = \pm\sqrt{R^2 - x^2}$$

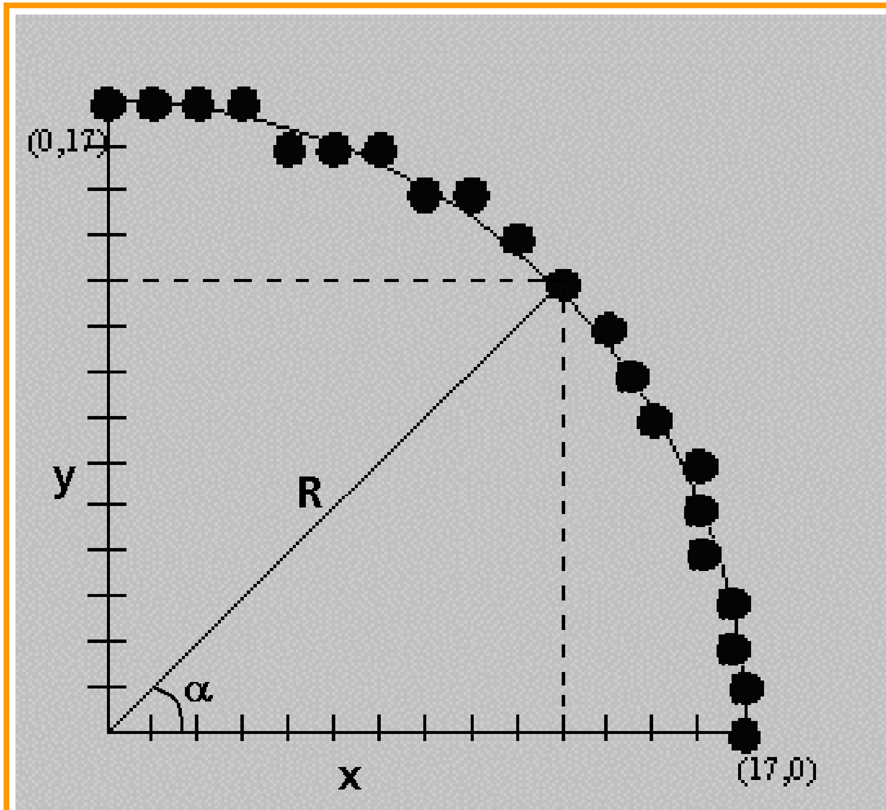


gap

Conversão matricial de circunferências

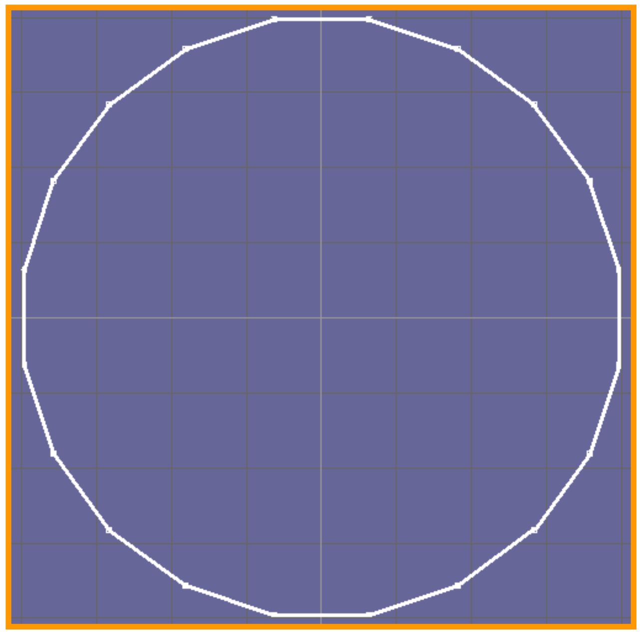
- Para evitar a presença de gaps pode-se utilizar incremento angular e funções trigonométricas.

$$x = R \cdot \cos(\alpha)$$
$$y = R \cdot \mathrm{sen}(\alpha)$$



Conversão matricial de circunferências

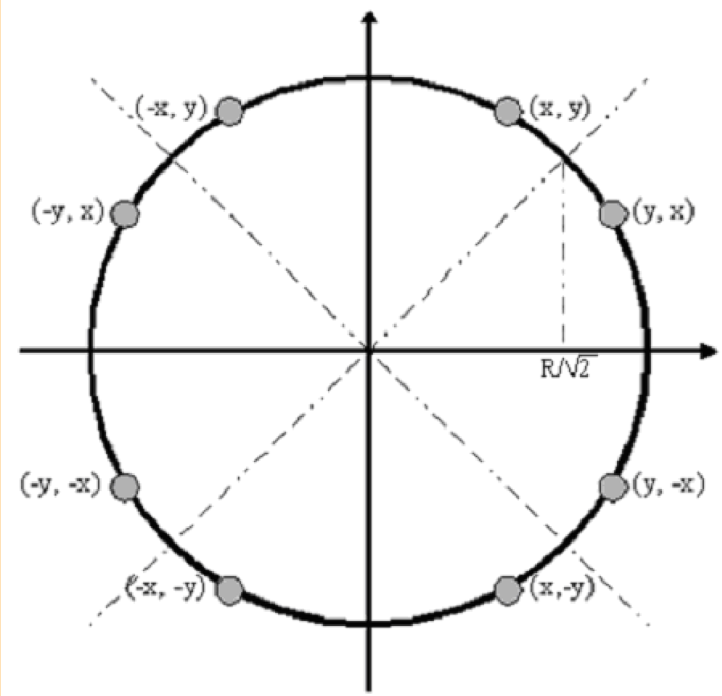
- Aproximação através de um polígono regular com n lados.



- Todas as alternativas são menos eficientes que o algoritmo do ponto médio para circunferências.

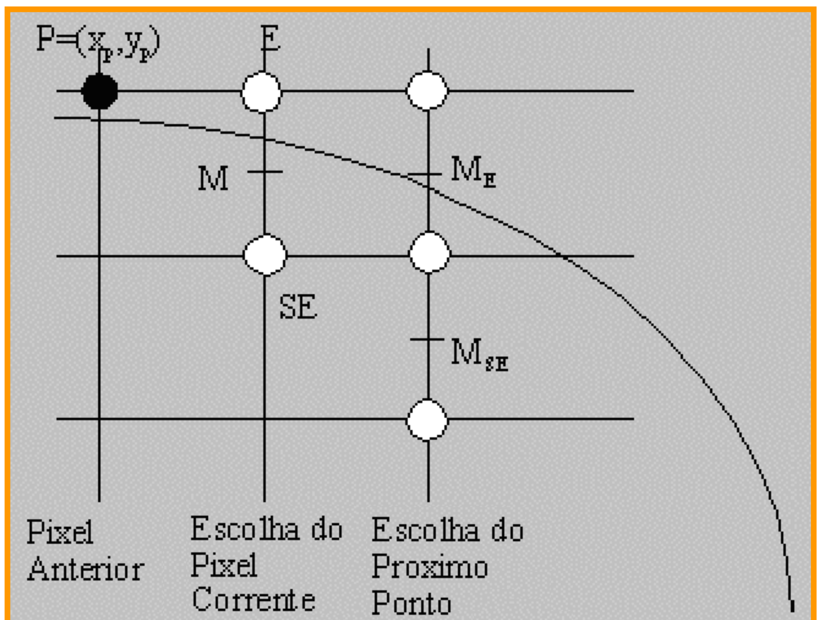
Simetria de ordem 8

```
void CirclePoints (int x, int y, int cor){
   WritePixel(x, y, cor);
   WritePixel(y, x, cor);
   WritePixel(y, -x, cor);
   WritePixel (x, -y, cor);
   WritePixel (-x, -y, cor);
   WritePixel (-y, -x, cor);
   WritePixel (-y, x, cor);
   WritePixel (-x, y, cor);
}
```



Algoritmo do ponto médio p/ circunferências

- Seja $F(x, y) = x^2 + y^2 - R^2$:
  - $F(x, y) > 0$ → fora da circunferência;
  - $F(x, y) < 0$ → dentro da circunferência.
- Se o ponto médio está entre os pixels E e SE:
  - Fora da circunferência → SE é escolhido;
  - Dentro da circunferência → E é escolhido.



Algoritmo do ponto médio p/ circunferências

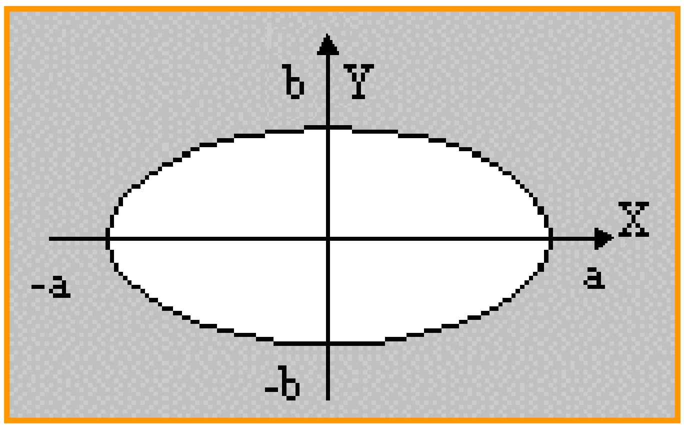
```
void MidpointCircle(int radius, int cor){
    //Assume que o centro do círculo está na origem
    int x = 0;
    int y = radius;
    int d = 1 - radius;
    CirclePoints(x, y, cor);
    while (y > x){
        if (d < 0 ) //escolhe E
            d += 2 * x + 3;
        else{ //escolhe SE
            d += 2 * (x - y) + 5;
            y--;
        }
        x++;
        CirclePoints(x, y, cor);
    }
}
```

Conversão matricial de elipses

- A elipse está centrada na origem (0, 0):
  - Quando isso não acontecer aplicar uma translação $(x+C_x, y+C_y)$;

$$\boxed{F(x,y) = b^2x^2 + a^2y^2 - a^2b^2 = 0}$$

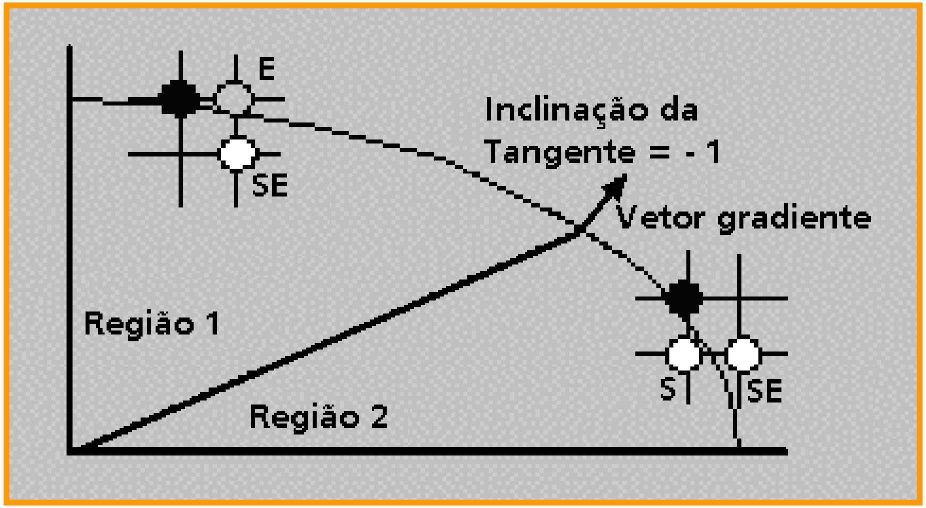
  - 2a = comprimento do eixo maior (eixo x);
  - 2b = comprimento do eixo menor (eixo y).



Conversão matricial de elipses

- A elipse possui simetria de ordem 4:
  - Traçar o primeiro quadrante;
  - Quadrante dividido em duas regiões;
  - Ponto de divisão → vetor gradiente.

$$\vec{\nabla} F(x,y) = \frac{\partial F}{\partial x} i + \frac{\partial F}{\partial y} j = 2b^2 x\, i + 2a^2 y\, j$$



Algoritmo do ponto médio para elipses

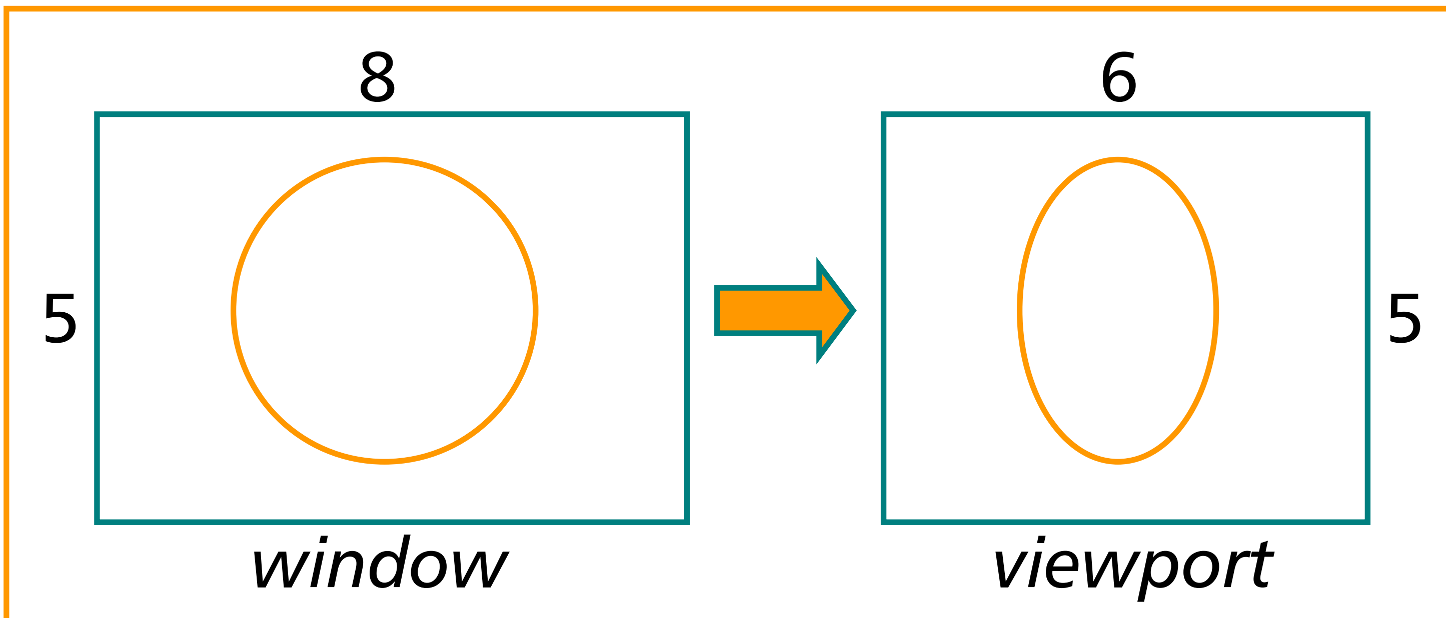
```
void MidpointElipse (int a, int b, int cor){
   //Assume que o centro da elipse é a origem
   int a2 = a * a;        int b2 = b * b;
   int twoa2 = 2 * a2;    int twob2 = 2 * b2;
   int x = 0;             int y = b;
   int px = 0;            int py = twoa2 * y;
   int p;

   EllipsePoints (x, y, cor);
   p = int(b2 - (a2 * b) + (0.25 * a2) + 0.5);
   while (px < py){
      x++;
      px += twob2;
      if (p < 0) p += b2 + px
      else{
         y--;
         py -= twoa2;
         p += b2 + px - py;
      }
      EllipsePoints (x, y, cor);
   }
```

Algoritmo do ponto médio para elipses

```
    p = int(b2 * (x+0.5)*(x+0.5) + a2 * (y-1)*(y-1) - a2 * b2 + 0.5);
    while (y > 0){
        y--;
        py -= twoa2;
        if (p > 0) p += a2 - py;
        else{
            x++;
            px += twob2;
            p += a2 - py + px;
        }
        EllipsePoints (x, y, cor);
    }
}
```

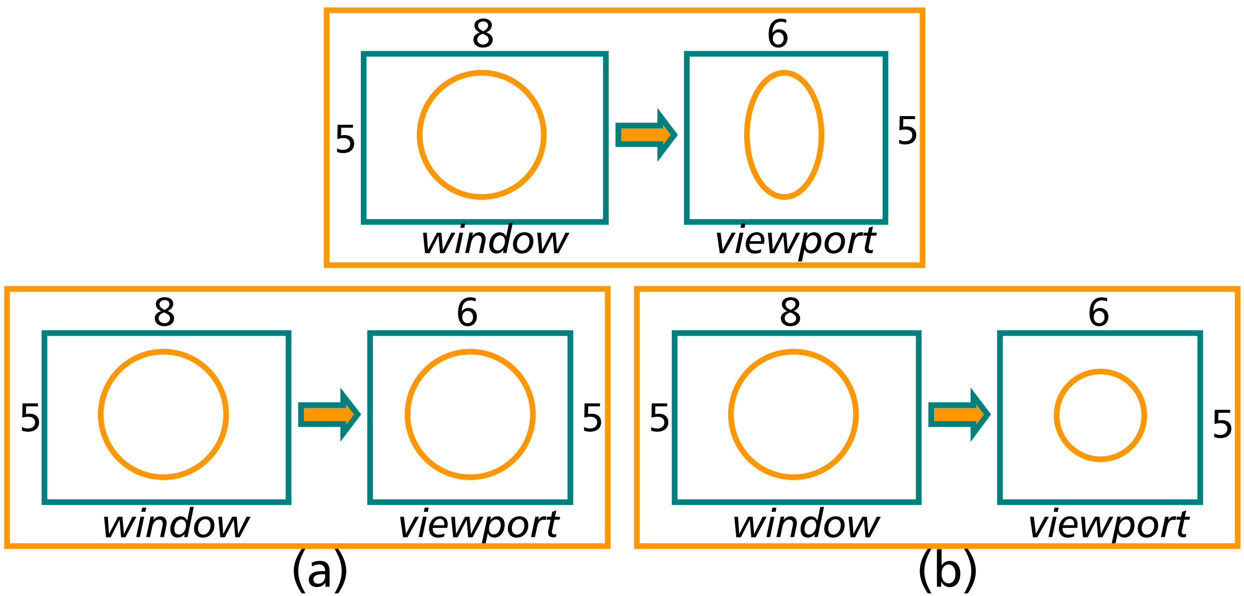
Correção no traçado

- Necessária quando a razão de aspecto física (*window*) difere da razão de aspecto do dispositivo (*viewport*);
- Solução → transformação de escala.



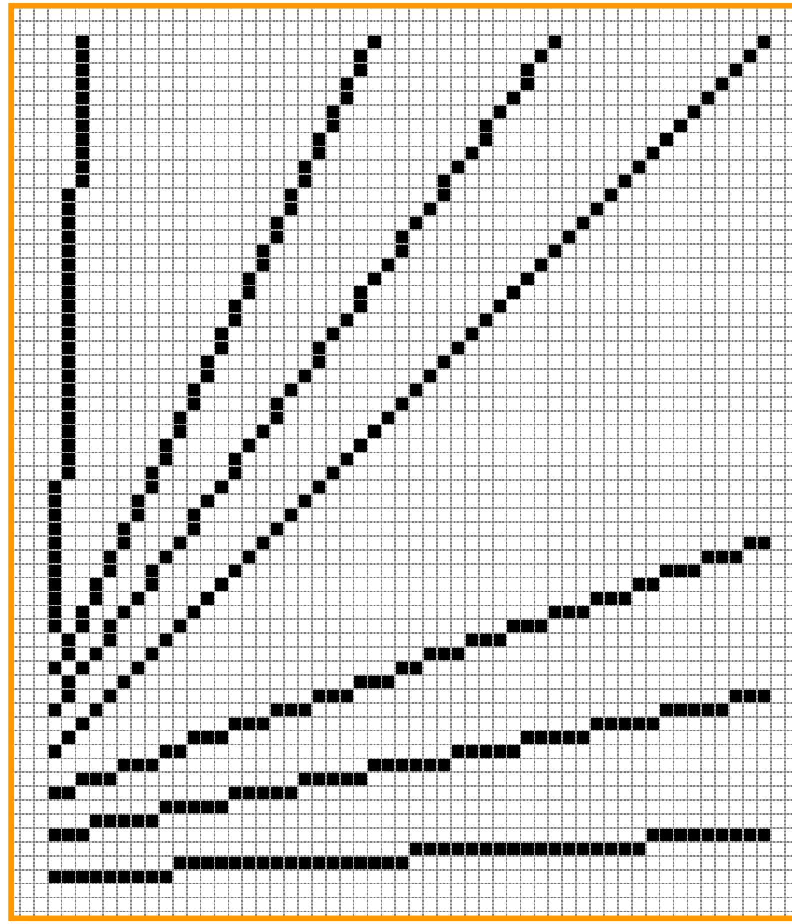
Correção no traçado

- Distorção no eixo x → duas alternativas na *viewport*:
  - a) Aumentar as dimensões horizontais do objeto:
    - $x_{ve} = x_v * ((width/height)/(ndh/ndv))$
  - b) Diminuir as dimensões verticais do objeto:
    - $y_{ve} = y_v * ((ndh/ndv)/(width/height))$



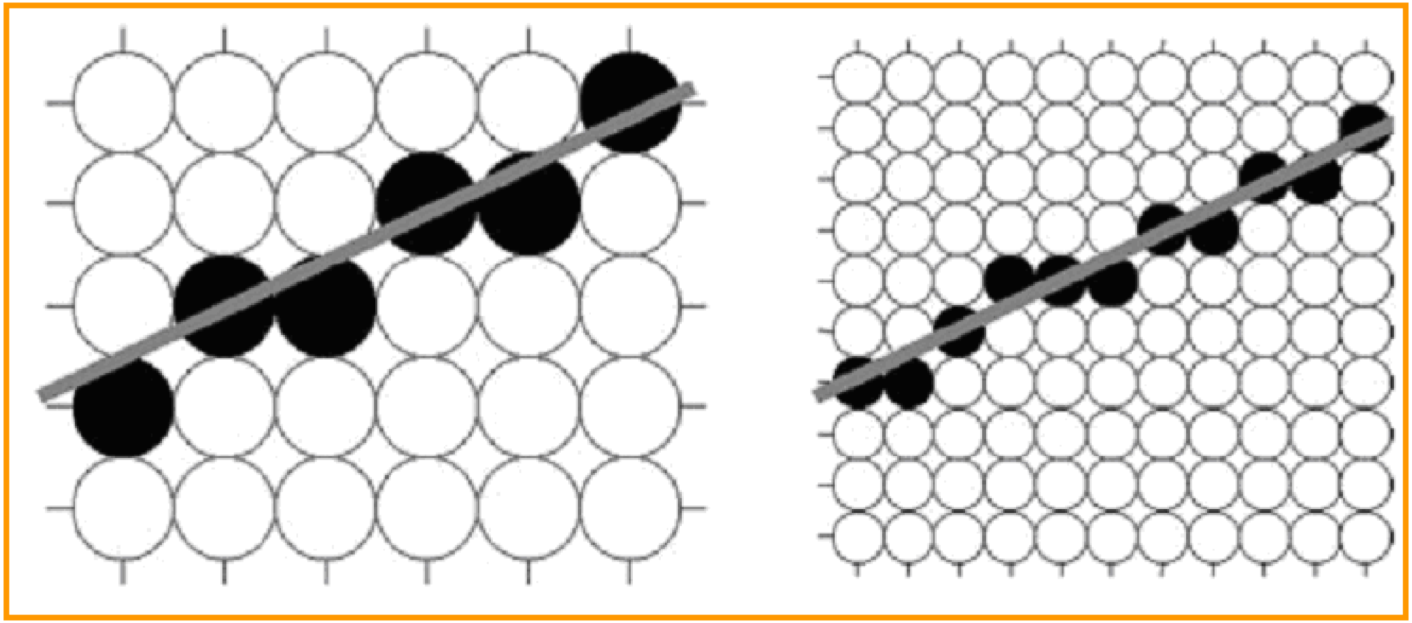
Correção no traçado

- Serrilhado → *Aliasing*;
- Natural no processo de conversão matricial;
- Mais pronunciado nos traços de arcos com inclinações próximas à horizontal e vertical;
- Correção → técnicas computacionalmente "caras";
- Controle da intensidade dos pixels vizinhos ao selecionado na conversão matricial.



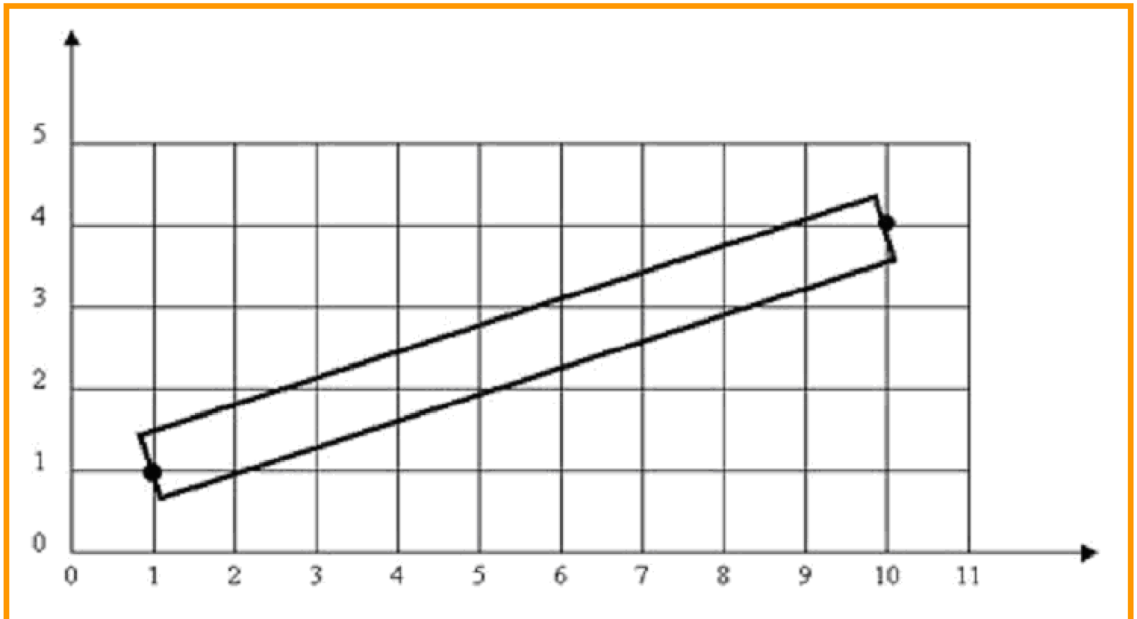
Antialiasing

- Aplicação de técnicas que reduzem o efeito de *aliasing*;
- Solução mais simples → aumentar a resolução do dispositivo de saída.



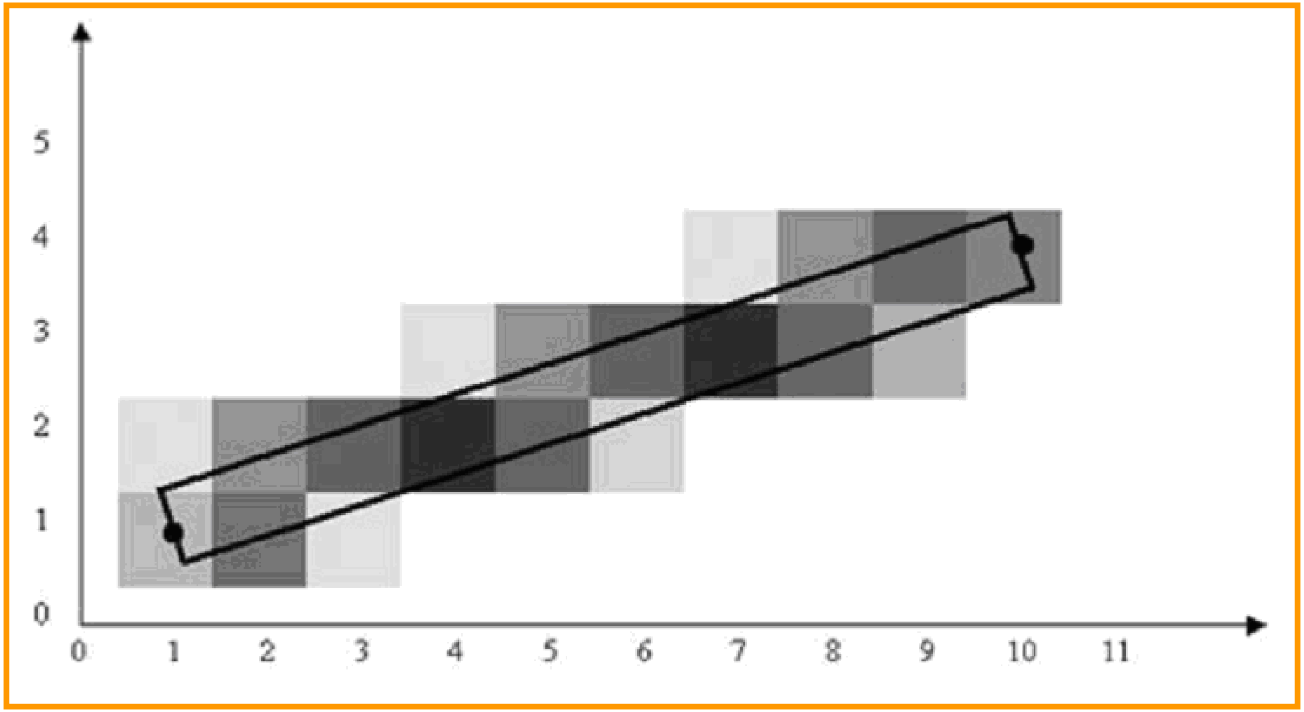
Amostragem de área não ponderada

- Um segmento de reta é um retângulo de espessura não nula que cobre uma região da malha de pixels;
- Linhas horizontais e verticais não apresentam problemas pois afetam só um pixel na coluna ou linha;
- As interseções da malha definem o centro do pixel.



Amostragem de área não ponderada

- Uma primitiva pode sobrepor toda ou parte da área ocupada por um pixel;
- Intensidade é proporcional à porcentagem da área do pixel coberta pela primitiva.

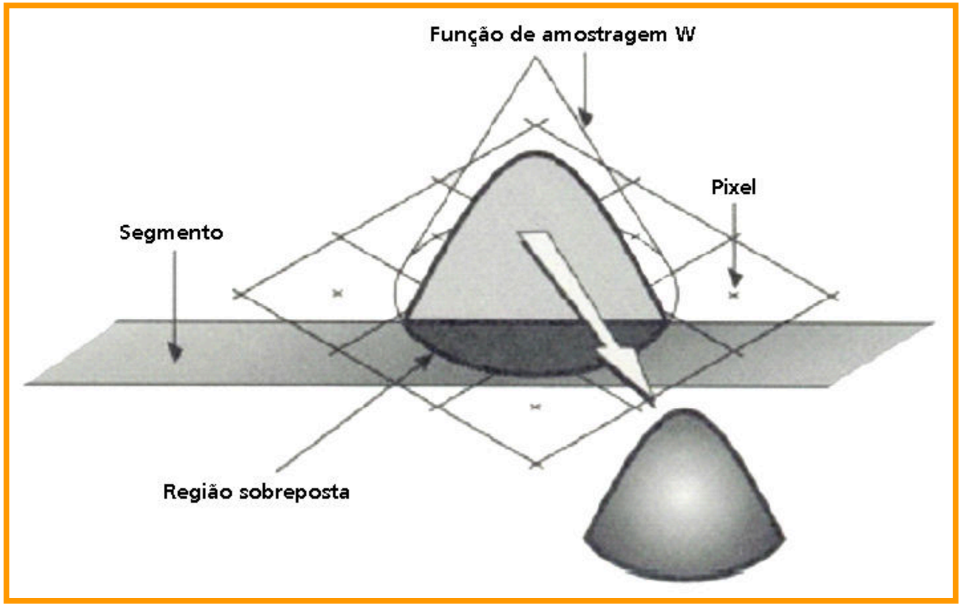


Amostragem de área ponderada

- Dois critérios:
  - A intensidade é proporcional à porcentagem da área do pixel coberta pela primitiva; e
  - se a primitiva não intercepta a área do pixel, então ela não contribui para a intensidade do pixel.
- Aumento da área do pixel:
  - O pixel é circular com área maior que o quadrado original.

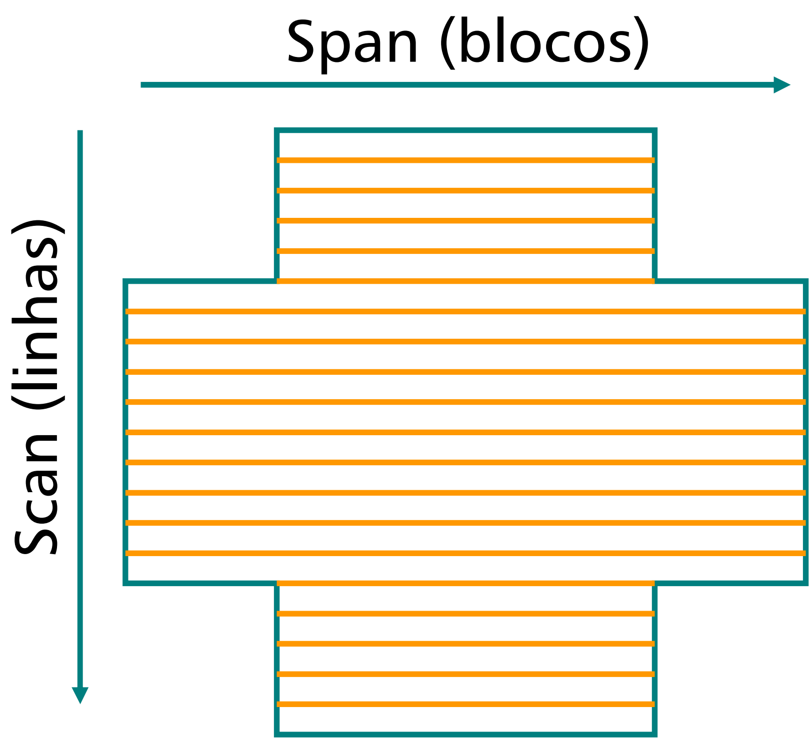
Amostragem de área ponderada

- Peso → Considera a proximidade da área sobreposta em relação ao centro do pixel.
  - Áreas iguais podem contribuir de forma desigual:
    - Uma área de sobreposição pequena próxima ao centro do pixel tem maior influência que uma área maior mais afastada.



Preenchimento de polígonos

- Tarefa dividida em duas etapas:
  - Decidir que pixels pintar para preencher o polígono;
  - Decidir com qual valor pintar o pixel.



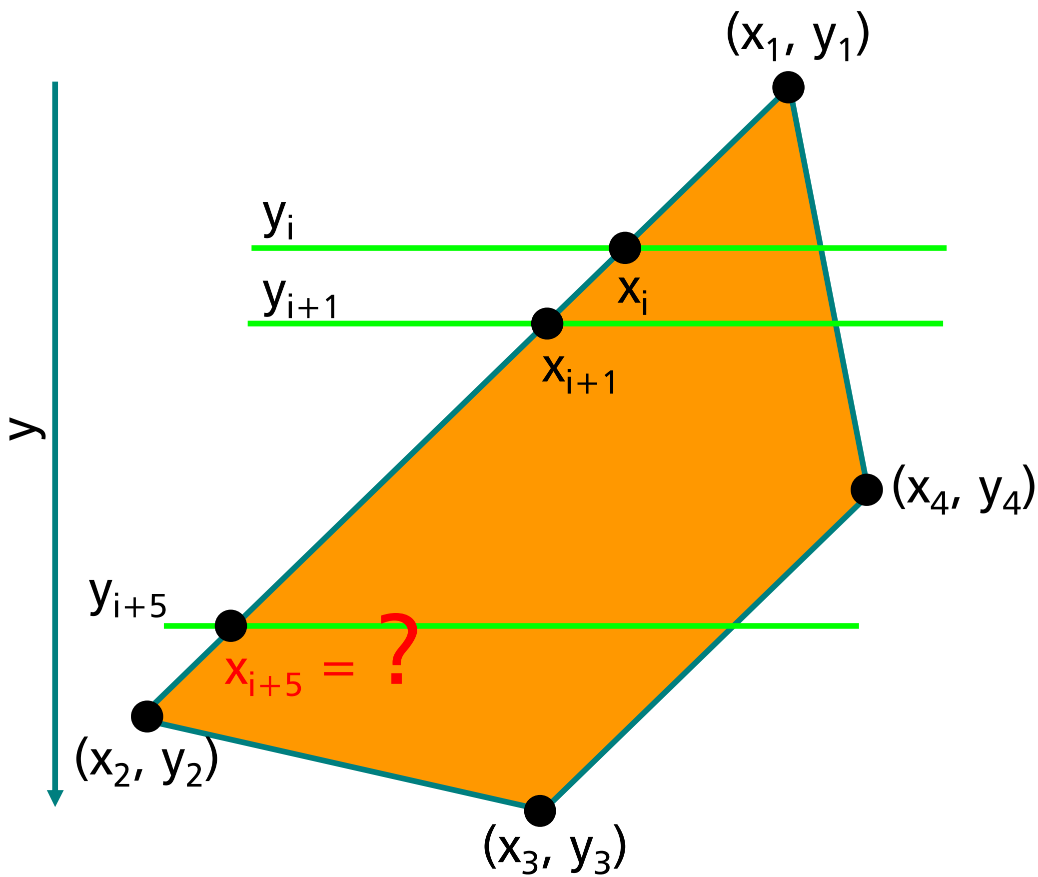
Preenchimento de retângulos


```
void FillRect (int x1, int y1, int x2, int y2, int cor){
   int x, int y;

   for (y = y1; y < y2; ++y)
      for (x = x1; x < x2; ++x)
         writepixel (x, y, value);
}
```

Explorando a coerência espacial

- A coerência espacial ocorre quando um bloco de pixels (span) é homogêneo ou um conjunto de linhas (scan) apresentam os mesmo limites.
- Há 3 coerências exploradas no preenchimento de polígonos:
  - **Coerência de bloco:** todos os pixels de um bloco (span) apresentam a mesma cor;
  - **Coerência de linha de varredura:** todas as linhas (scan) apresentam iguais limites mínimos e máximos;
  - **Coerência de arestas:** as arestas são formadas por linhas retas, possibilitando descobrir as interseções entre linhas e arestas através de cálculo incremental.

Coerência de arestas



$$m = \frac{(y_2 - y_1)}{x_2 - x_1}$$

$$x_i = \frac{(y_i - y_1)}{m} + x_1$$

$$x_{i+1} = \frac{(y_{i+1} - y_1)}{m} + x_1$$

$$como\ \Delta y = 1,\ então$$

$$x_{i+1} = x_i + \frac{1}{m}$$

Regra para o preenchimento

As arestas esquerda e inferior pertencem à primitiva e serão desenhadas; as arestas superior e à direita não pertencem e, portanto, não serão desenhadas.

- A aplicação desta regra evita que arestas compartilhadas entre polígonos sejam desenhadas duas vezes.
- Considerações:
  - Regra se aplica a polígonos regulares e irregulares;
  - Vértice do canto inferior esquerdo continua sendo desenhado duas vezes;
  - Em cada bloco falta o pixel mais à direita e em cada polígono faltam as arestas superiores.
- Não há solução perfeita para o problema.

Polígonos de forma arbitrária

- O algoritmo funciona para polígonos côncavos e convexos, mesmo aqueles que possuam autointerseção e buracos em seu interior.
- Este algoritmo pode explorar a coerência de arestas, utilizando cálculo incremental;
- 3 passos:

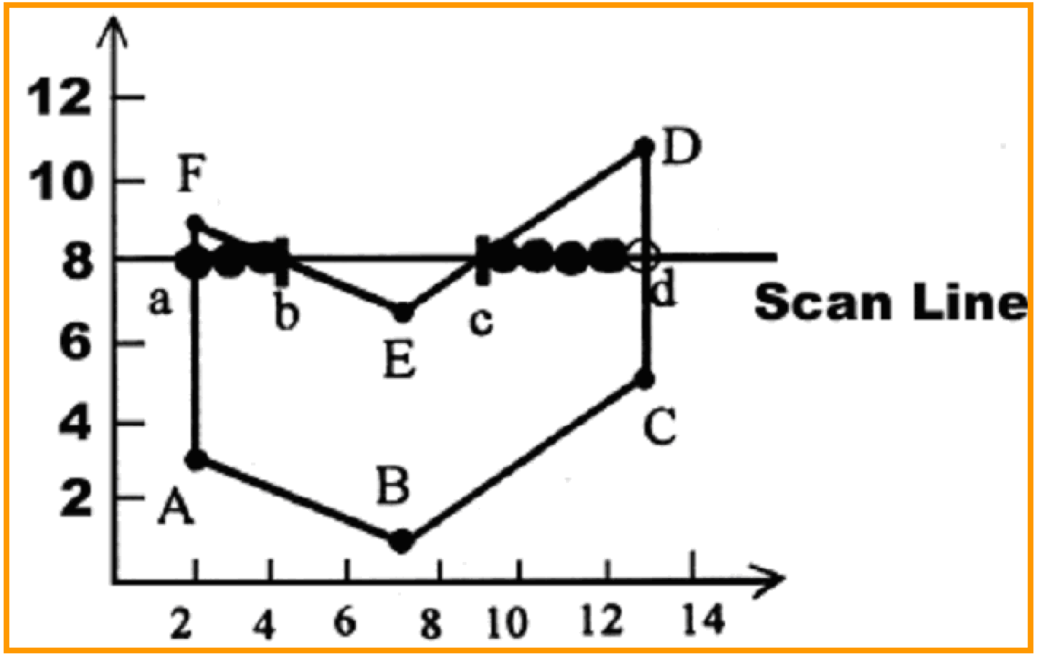
1 – Obter a interseção da linha de varredura (scan) com todos os lados do polígono;

2 – Ordenar os pontos de interseção (em *x* crescente);

3 – Preencher os pixels internos ao polígono. Usar a regra da paridade:

  - Par → Início → ponto fora do polígono;
  - Ímpar → ponto dentro do polígono → pintar.

Polígonos de forma arbitrária



- Para a linha de varredura com y = 8 há 4 interseções, com *x* crescente em:
  - (2; 4,5; 8,5; 13);
- Quais pixels pintar?

Polígonos de forma arbitrária

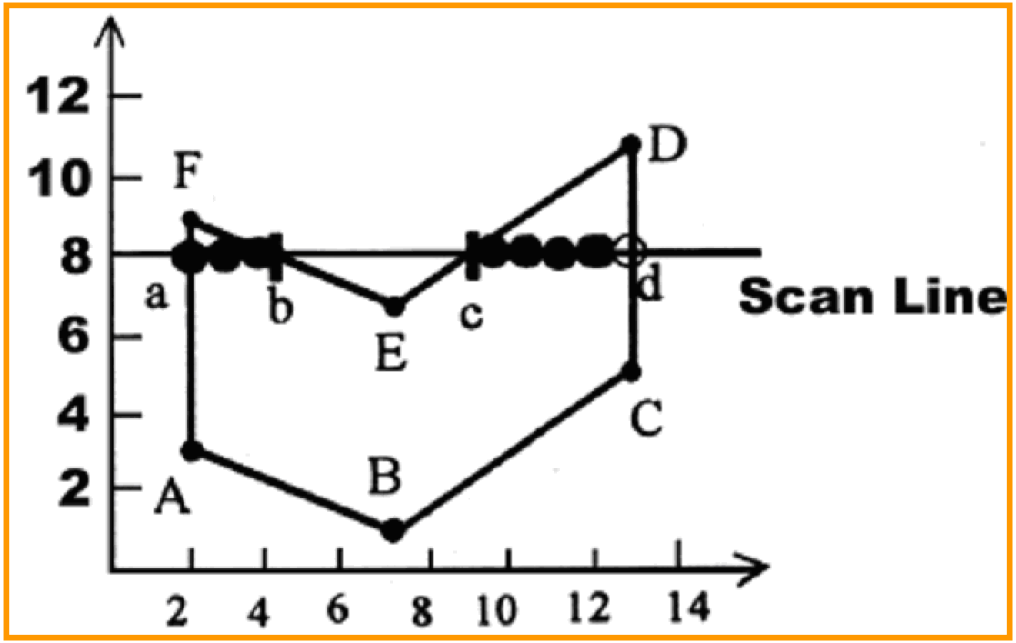
- Casos especiais:
  1 – Coordenada *x* da interseção é fracionária:
    - Se a paridade for par (fora do polígono) arredondamos o valor para cima;
    - Se a paridade for ímpar (dentro do polígono) arredondamos o valor para baixo.

  2 – Coordenada *x* da interseção é inteira:
    - Arestas à esquerda pertencem ao polígono e são traçadas; arestas à direita não pertencem ao polígono.

  3 – Um vértice é compartilhado por mais de uma aresta:
    - Só haverá mudança na paridade quando o vértice for y_{min} da aresta.

  4 – Os vértices definem uma aresta horizontal:
    - Arestas inferiores pertencem ao polígono e são traçadas; arestas superiores não pertencem ao polígono.

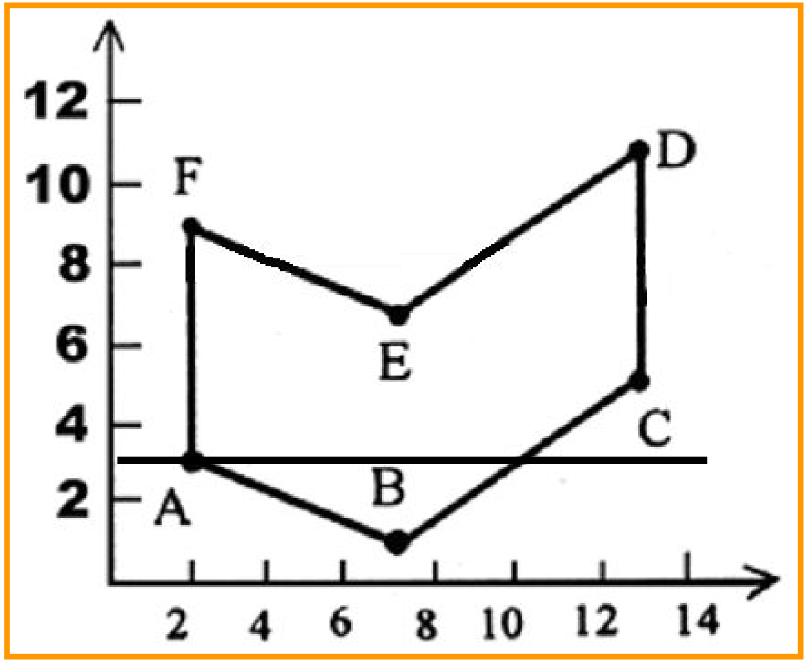
Polígonos de forma arbitrária – Exemplo



- Linha de varredura 8:

  Preenchimento do ponto ***a*** (2, 8) até o primeiro pixel à esquerda do ponto ***b*** (4, 8); preenchimento do primeiro pixel à direita do ponto ***c*** (9, 8) até um pixel à esquerda do ponto ***d*** (12, 8).

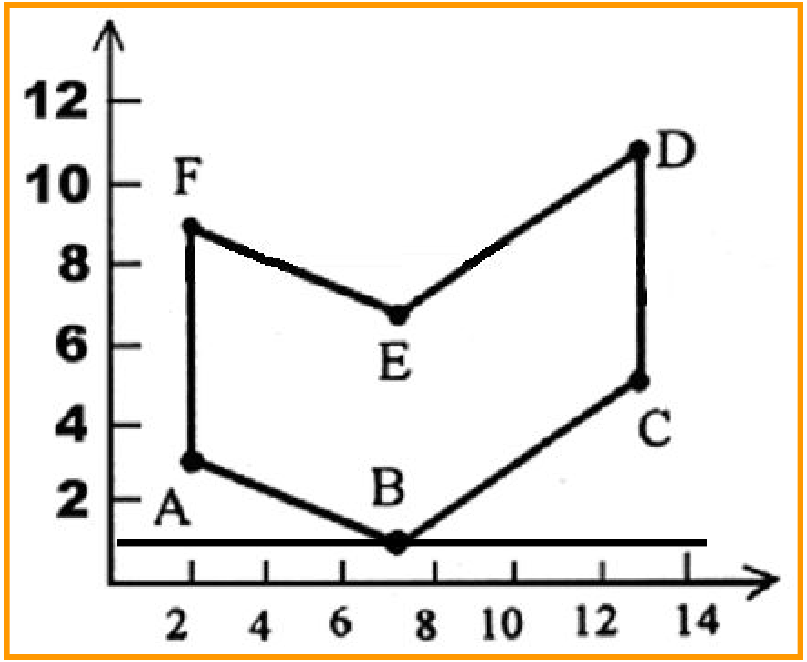
Polígonos de forma arbitrária – Exemplo



- Linha de varredura 3:

  O vértice ***A*** muda a paridade para ímpar pois é $\boldsymbol{y_{min}}$ da aresta ***FA***. O bloco é desenhado do ponto ***A*** até um pixel à esquerda da interseção com o lado ***BC***, onde a paridade muda para par.

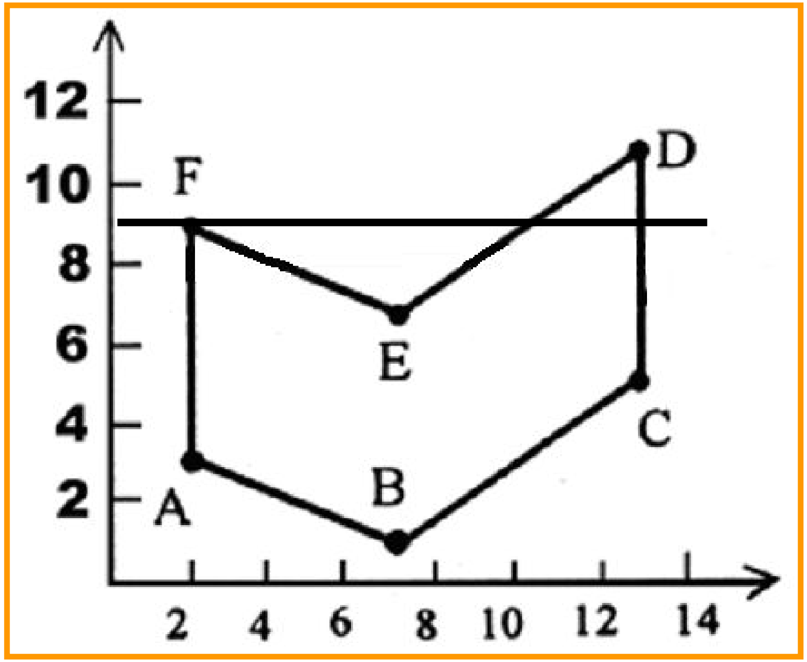
Polígonos de forma arbitrária – Exemplo



- Linha de varredura 1:

  Passa apenas pelo vértice ***B*** que é $\boldsymbol{y_{min}}$ das arestas ***AB*** e ***BC***. A paridade muda de par para ímpar e volta para par, formando um bloco com um pixel, que será desenhado porque é mínimo local.

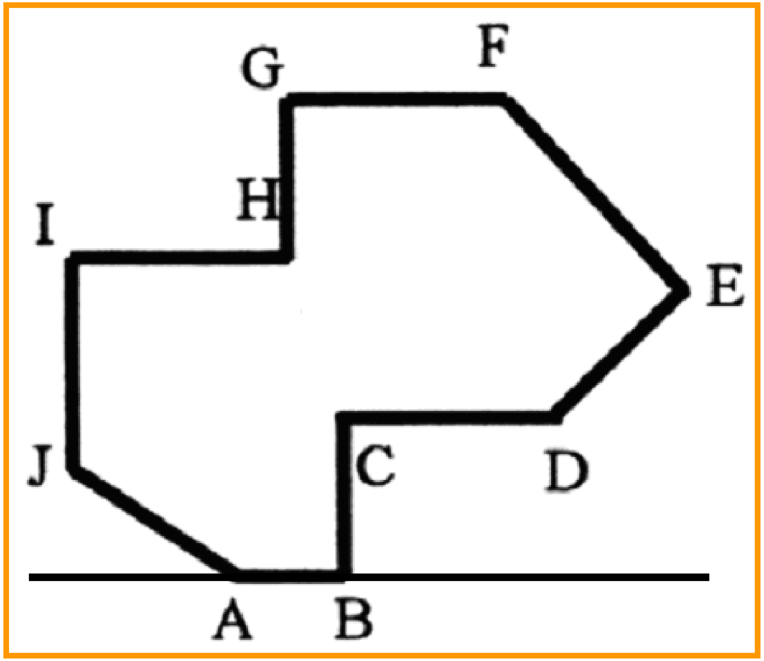
Polígonos de forma arbitrária – Exemplo



- Linha de varredura 9:

  O vértice ***F***, compartilhado pelas arestas ***EF*** e ***FA***, não afeta a paridade pois é máximo local. O bloco a ser desenhado vai da interseção com a aresta ***DE*** até a interseção com a aresta ***CD***.

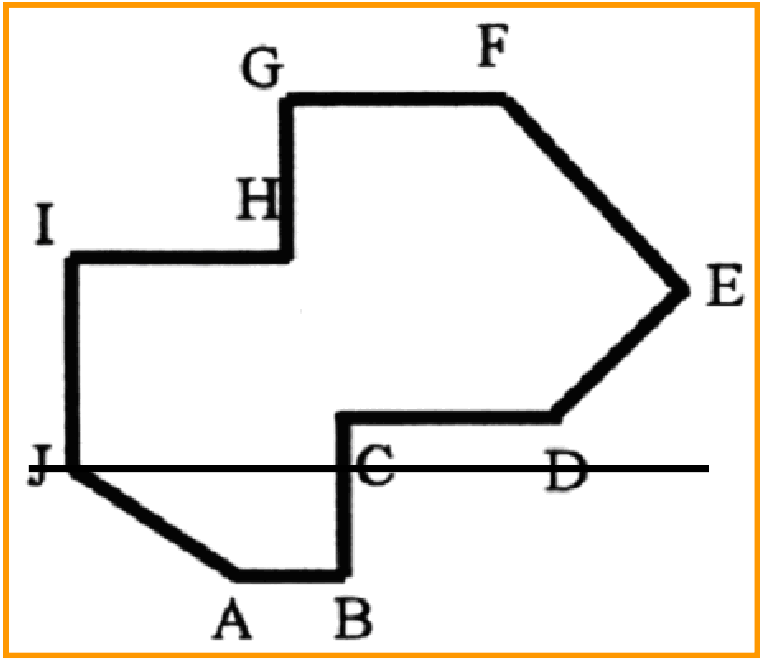
Arestas horizontais – Exemplo



- Linha de varredura AB:

  O vértice ***A*** é y_{min} da aresta ***JA***. A aresta ***AB***, por ser horizontal, não possui mínimo. Assim a paridade muda para ímpar retornando para par em ***B***, pois ***B*** é y_{min} da aresta ***BC***. Então o bloco ***AB*** é desenhado.

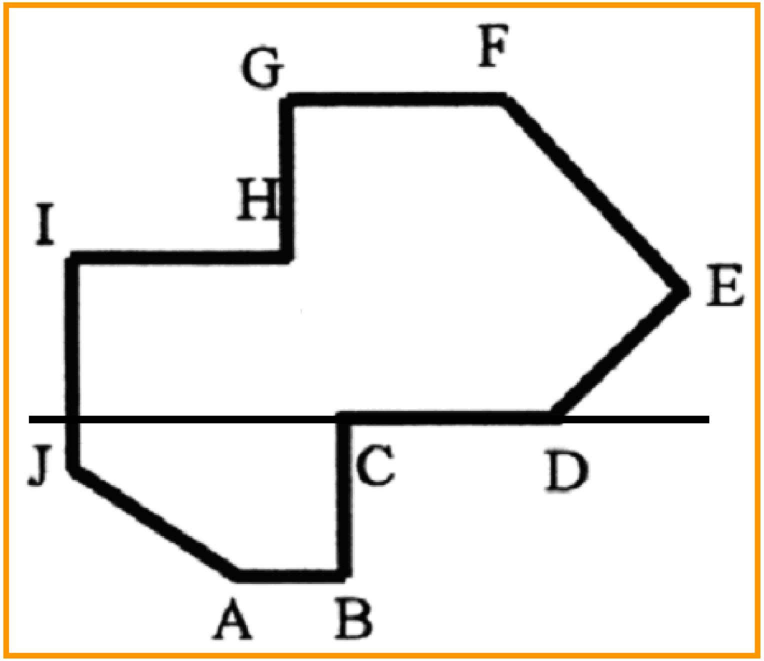
Arestas horizontais – Exemplo



- Linha de varredura J(BC):

  O vértice ***J*** é $\boldsymbol{y_{min}}$ da aresta ***IJ***. A paridade muda para ímpar e retorna para par na interseção com a aresta ***BC***. O bloco ***J***(***BC***) é desenhado.

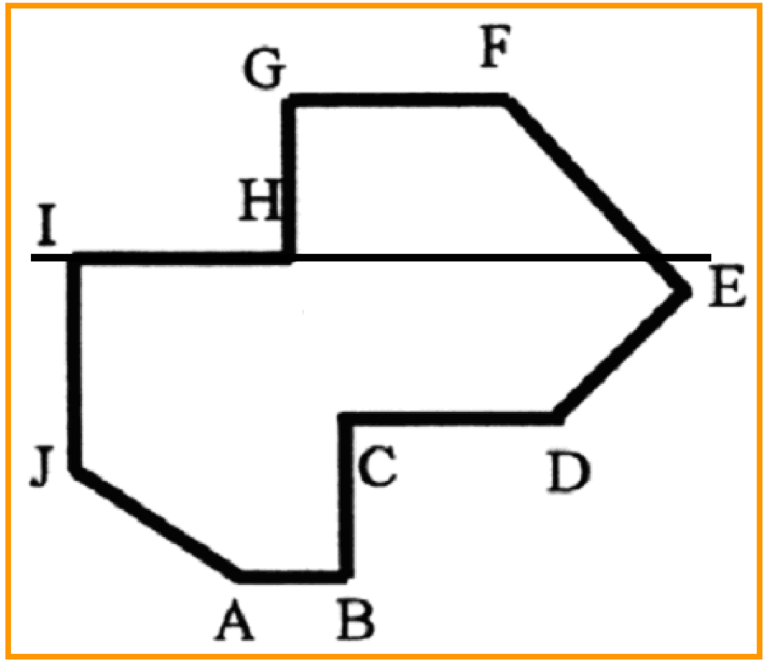
Arestas horizontais – Exemplo



- Linha de varredura (IJ)D:

  Na interseção com a aresta ***IJ*** a paridade muda para ímpar. No vértice ***C*** a paridade não muda pois ***C*** não é y_{min} de ***BC*** nem de ***CD***. A paridade volta para par em ***D*** pois e y_{min} de ***DE***. O bloco ***JD*** é desenhado.

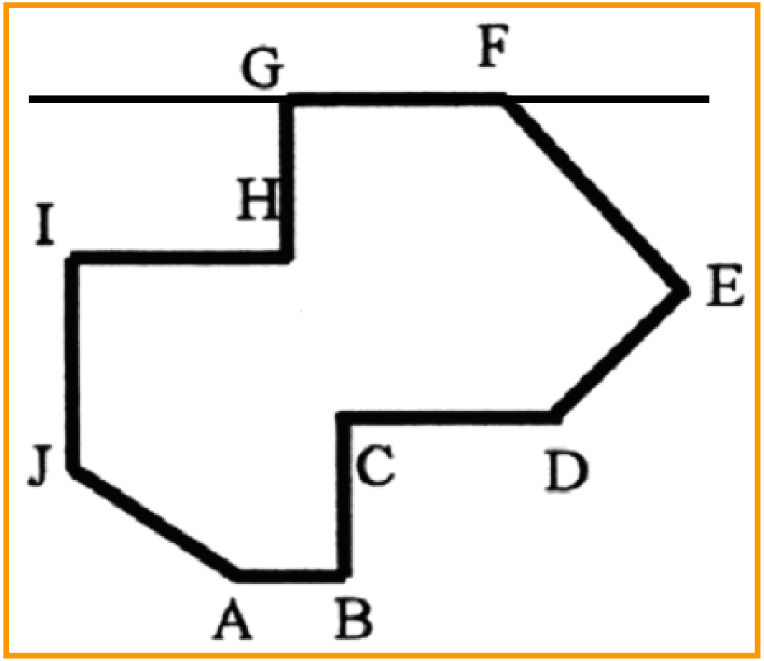
Arestas horizontais – Exemplo



- Linha de varredura I(EF):

  O vértice ***I*** não afeta a paridade pois não é $\boldsymbol{y_{min}}$ da aresta ***IJ*** e da aresta ***HI***. Mas o vértice ***H*** é $\boldsymbol{y_{min}}$ de ***GH***, mudando a paridade para ímpar. Esta volta a ser par na interseção com a aresta ***EF***. O bloco ***H***(***EF***) é desenhado.

Arestas horizontais – Exemplo

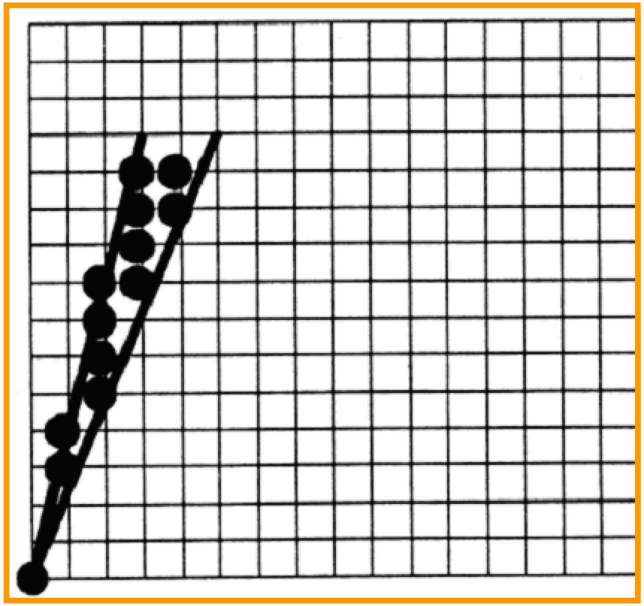


- Linha de varredura GF:

  O vértice ***G*** não afeta a paridade pois não é $\boldsymbol{y_{min}}$ da aresta ***GH***, nem da aresta ***FG***. O vértice ***F*** também não afeta a paridade pois não é $\boldsymbol{y_{min}}$ da aresta ***FG***, nem da aresta ***EF***.O bloco ***GF*** não é desenhado.

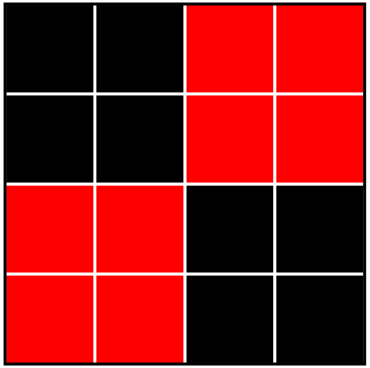
Slivers

- Polígonos com lados muito próximos geram um "*sliver*":
  - Área poligonal tão estreita que seu interior não contém um bloco de pixels para cada linha de varredura.
- Solução → *antialiasing:*
  - Permitir que pixels na fronteira, ou mesmo fora da área, sejam desenhados com intensidades variando em função da distância entre o centro do pixel e a primitiva.



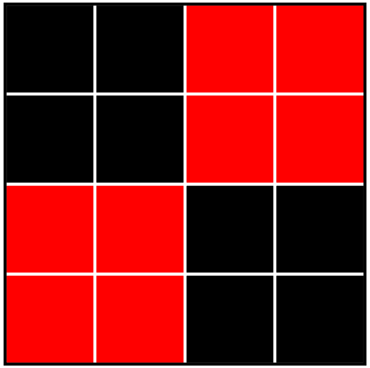
Preenchimento com padrões

- Dois estágios:
  - Determinar a matriz de pontos que compõe o padrão;
  - Determinar, para um pixel qualquer, qual cor da matriz devemos utilizar para o pixel.
- Matriz de pontos:
  - Mostra como o padrão é definido no mapa de pixels;
  - O tamanho dessa matriz é definido pelo programador.



Matriz de pontos

```
MATPIX[0, 0] := BLACK; {MATPIX[linha, coluna] = MATPIX[I, J]}
MATPIX[0, 1] := BLACK;
MATPIX[0, 2] := RED;
MATPIX[0, 3] := RED;
MATPIX[1, 0] := BLACK;
MATPIX[1, 1] := BLACK;
MATPIX[1, 2] := RED;
MATPIX[1, 3] := RED;
MATPIX[2, 0] := RED;
MATPIX[2, 1] := RED;
MATPIX[2, 2] := BLACK;
MATPIX[2, 3] := BLACK;
MATPIX[3, 0] := RED;
MATPIX[3, 1] := RED;
MATPIX[3, 2] := BLACK;
MATPIX[3, 3] := BLACK;
```



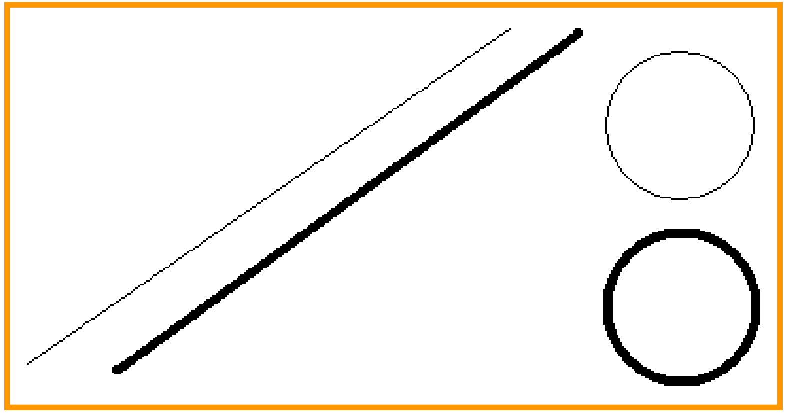
Determinação da cor para um pixel

- A determinação da cor do pixel na tela é feita da seguinte maneira:
  - Escolhe-se o padrão para preencher o objeto;
  - Para cada pixel (x, y) do objeto calcula-se a cor do pixel correspondente na matriz que define o padrão.
- Uso do operador "mod":
  - i = y mod (nº de linhas do padrão)
  - j = x mod (nº de colunas do padrão)
- Exemplo: pixel (30, 20):
  - i = 20 mod 4 = 0;
  - j = 30 mod 4 = 2;
  - Cor = MATPIX[0, 2] = RED.

Função GetColor(x, y, NC, NL)

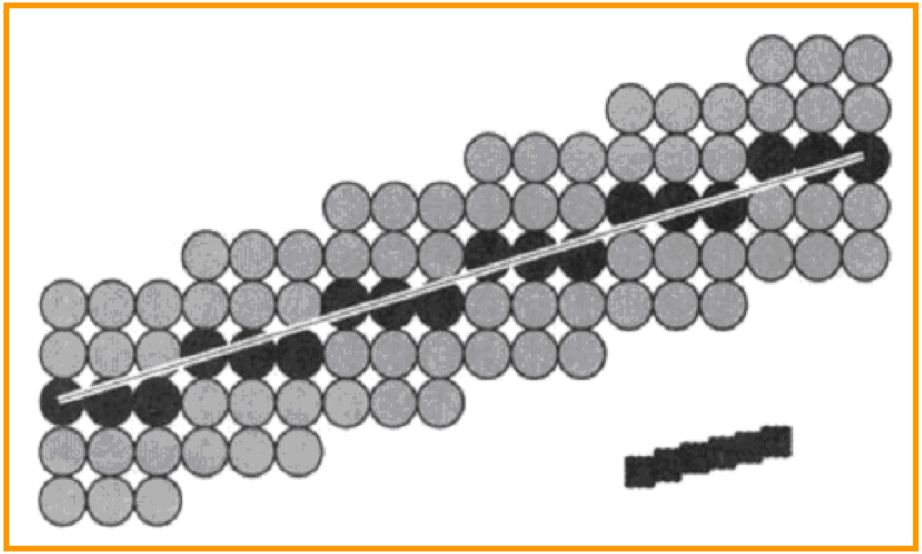
```
function GetColor (x, y, NC, NL : integer):byte;
{X e Y são as coordenadas do pixel do objeto}
{NC e NL são o número de colunas e
 o número de linhas da matriz padrão MATPIX}

var i, j : byte;

begin
   i := y mod NL;
   j := x mod NC;
   GetColor := MATPIX[i, j];
end;
```

Geração de primitivas espessas



- Geradas a partir de uma linha base obtida por conversão matricial;
- Três métodos:
  - Replicação de pixels;
  - Movimento da caneta;
  - Preenchimento de área entre dois limites.

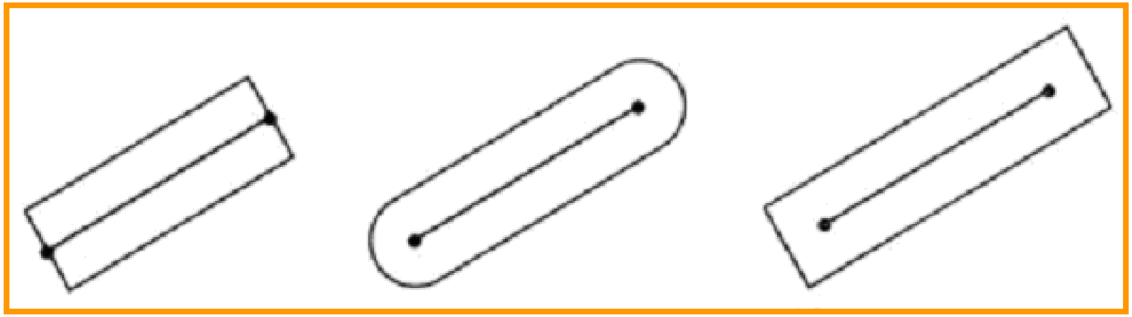
Replicação de pixels



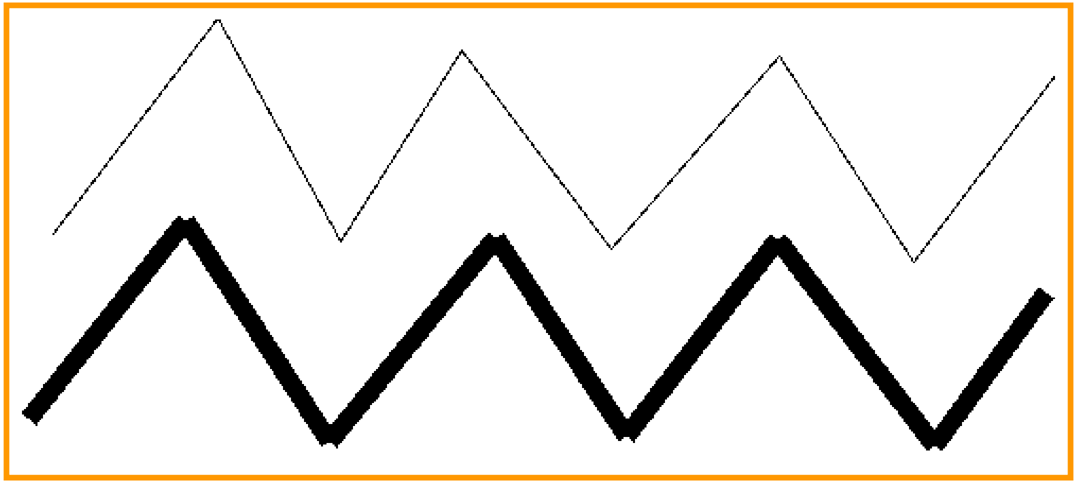
- Influência do coeficiente angular (m):
  - -1 < m < 1 → replicação em colunas;
  - Nos outros casos → replicação em linhas.
- Inconveniente:
  - extremos das linhas sempre em linhas retas.

Replicação de pixels

- Ajuste dos extremos finais pela adição de capas:
  - Butt cap:
    - Adição de quadrados ao extremo da linha, com inclinação igual –1/m.
  - Round cap:
    - Adição de um semicírculo preenchido centralizado nos pontos extremos da linha.
  - Projecting square cap:
    - Estende-se a linha adicionando “butt caps” posicionadas metade da largura da linha além dos extremos.

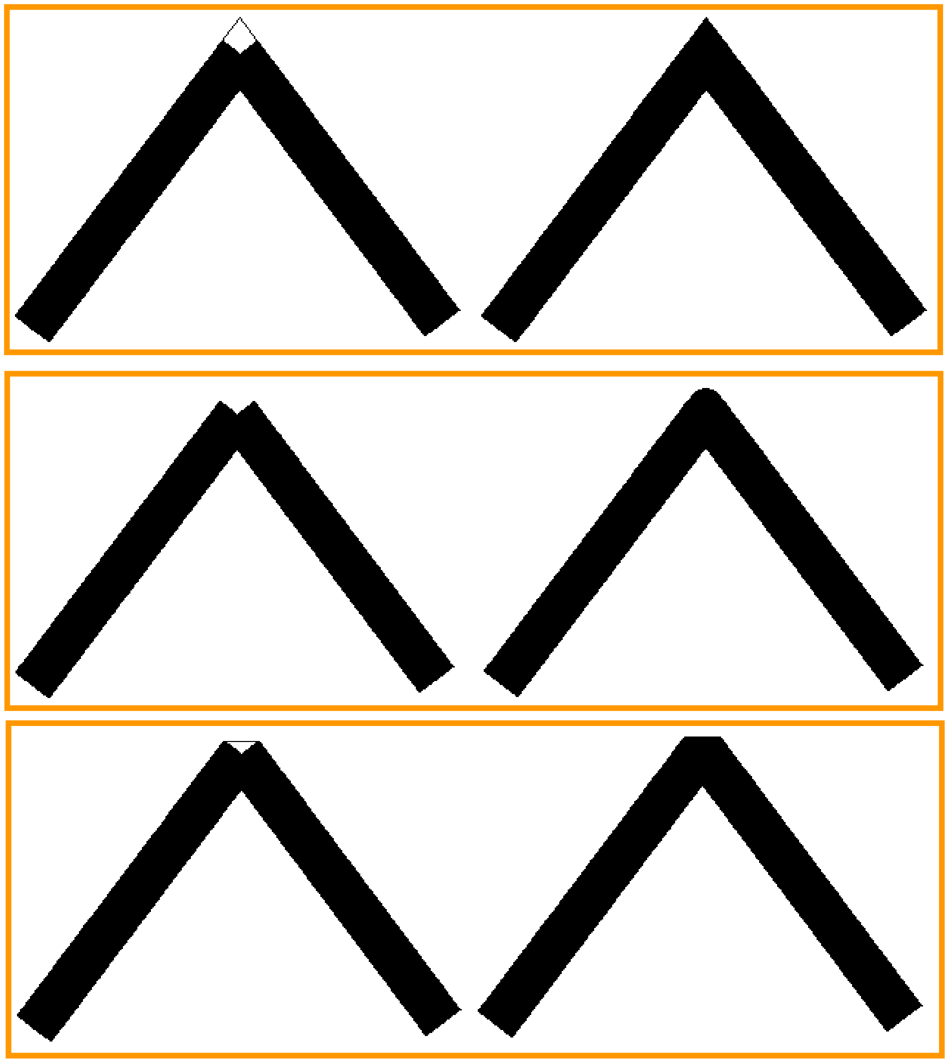


Geração de linhas múltiplas (*polylines*)



- Necessidade de gerar conexões suaves entre os segmentos da *polyline;*
- Requer o processamento dos extremos de cada segmento.

Junções

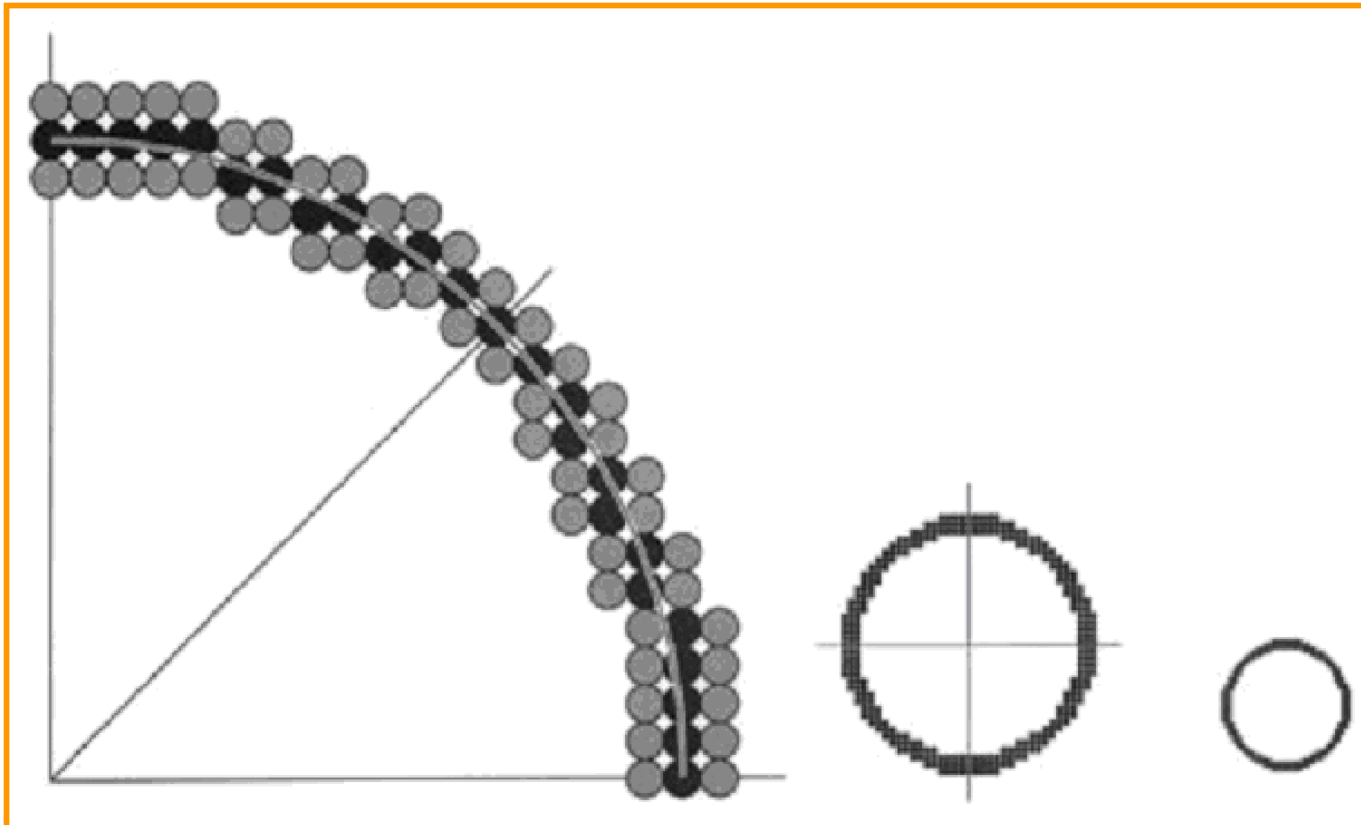


Turbante

Redonda

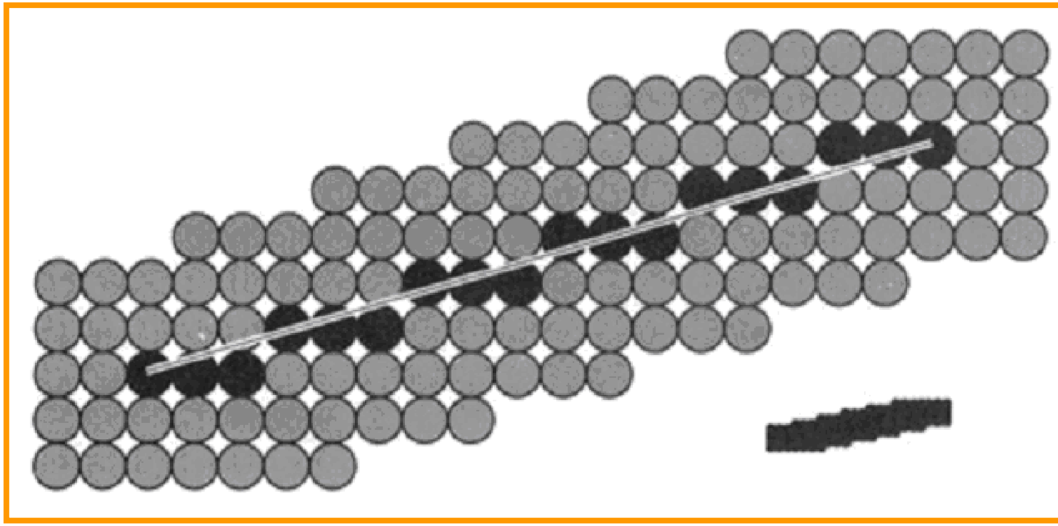
Bisel

Replicação de pixels



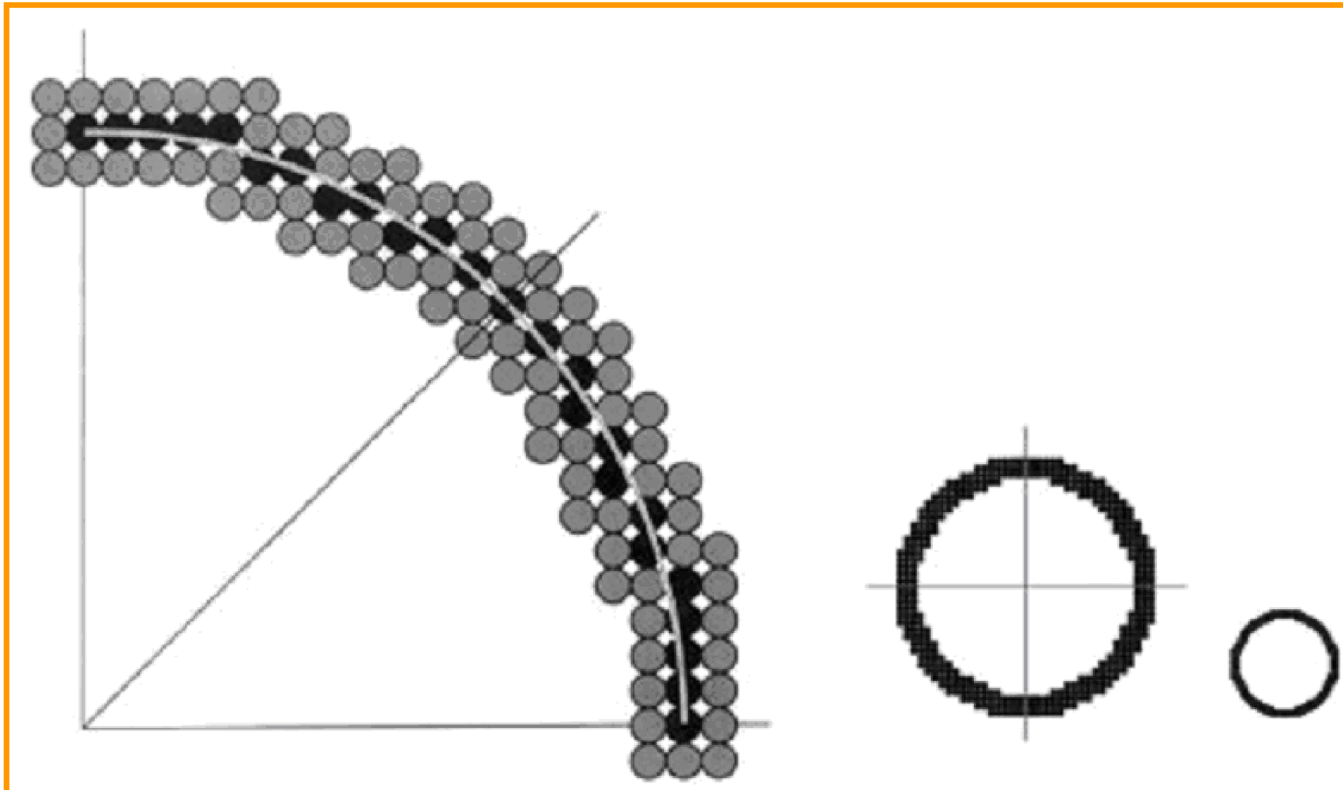
- Problemas:
  - Espessura varia conforme inclinação;
  - Linhas pares apresentam ligeiro deslocamento.

Movimento da caneta



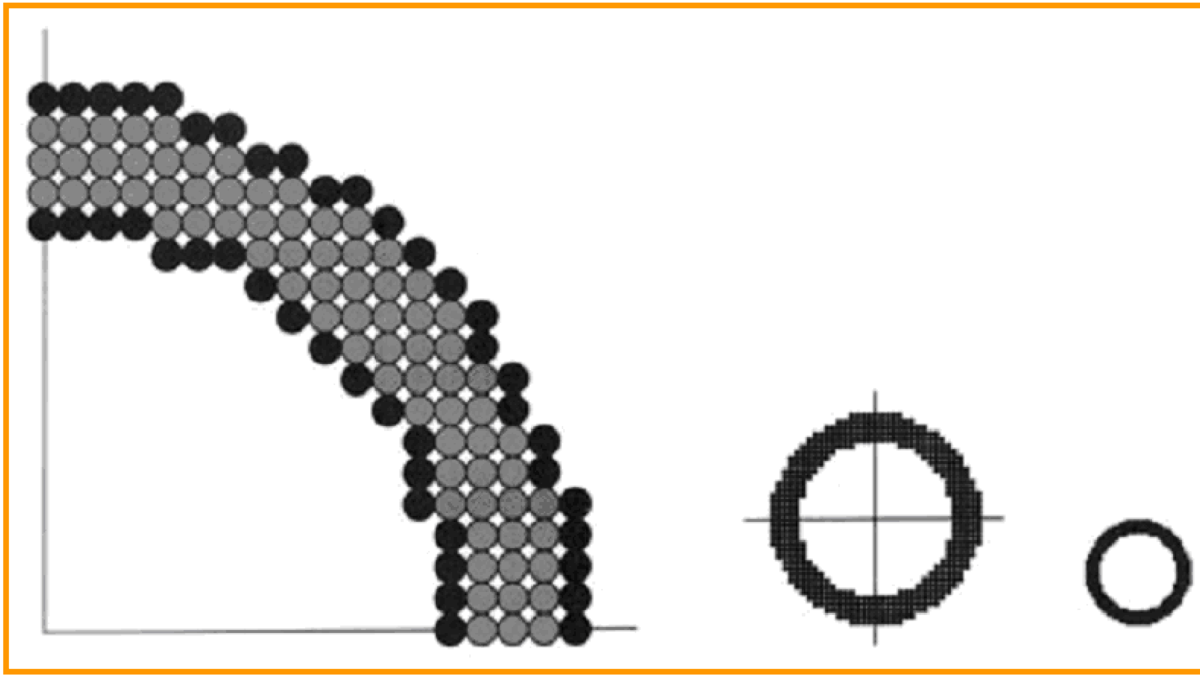
- Uso de uma caneta de seção transversal retangular;
- Como a caneta permanece alinhada na vertical, linhas horizontais e verticais apresentam largura menor que as inclinadas.

Movimento da caneta



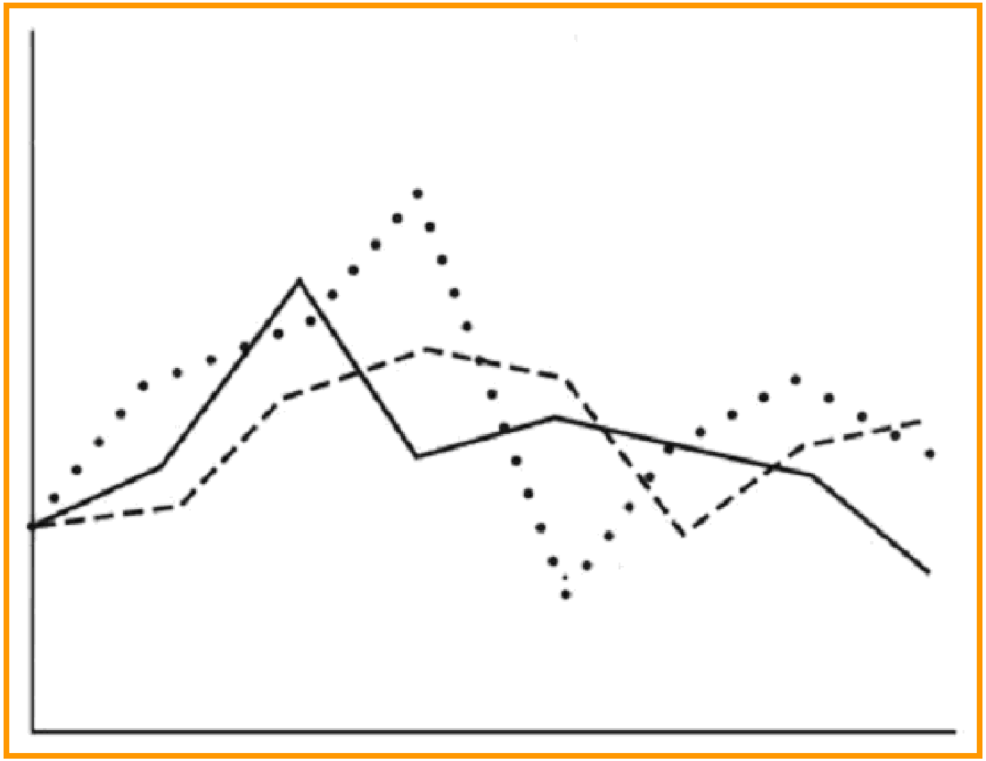
- Soluções:
  - Girar a caneta ao longo da trajetória;
  - Usar uma caneta de seção circular.

Preenchimento de área entre dois limites



- Traçar duas primitivas à distância t/2 em ambos os lados da primitiva básica.
- Problema com espessuras ímpares → Traçar a primitiva básica como externa e uma interna à distância t.

Tipos de linhas



- Modificação do algoritmo de conversão matricial de linhas (Como fazer?);
- Uso de máscara de bits para gerar padrões de linhas:
  - 11111000 = linha tracejada de 5 pixels espaçado por 3 pixels.

Algoritmos de recorte (*Clipping*)

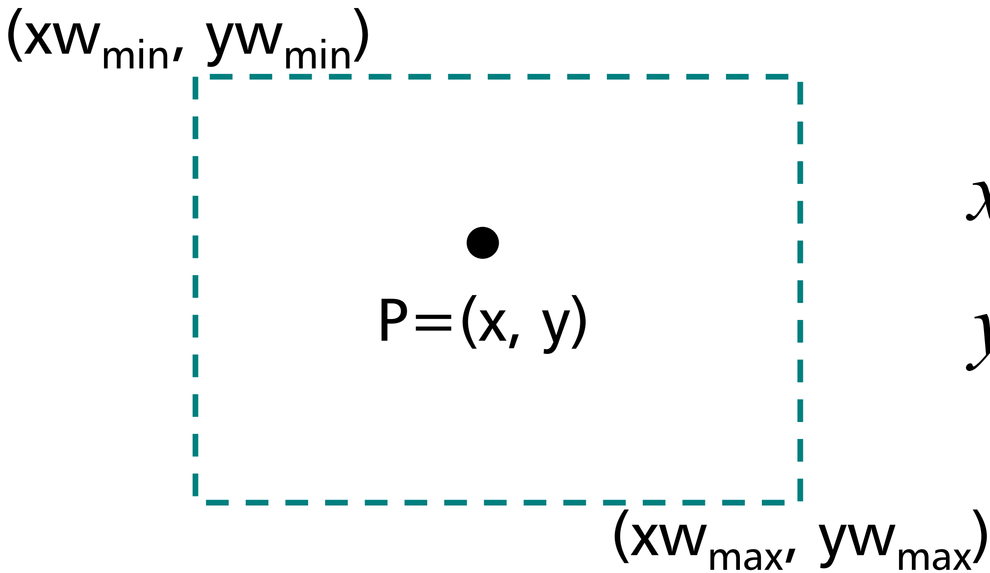
- Qualquer procedimento que identifica partes de uma figura que correspondam a regiões dentro ou fora de um espaço especificado;
- A região de recorte é chamada de *clip window*;
- Aplicações:
  - Extrair uma parte de uma cena para ser visualizada;
  - Identificar superfícies visíveis em 3D;
  - Mostrar múltiplas janelas.
- O processo de clipping pode ser feito em:
  - Coordenadas de mundo → contra a janela (*window*);
  - Coordenadas normalizadas;
  - Coordenadas de tela → contra a *viewport.*

Algoritmos de recorte (*Clipping*)

- Recorte contra a janela elimina objetos, ou partes deles, que estejam fora na janela.
  - Poupa processamento na conversão para coordenadas da *viewport*.
- A conversão para coordenadas da *viewport* pode estar concatenada nas matrizes de transformação e visualização, o que:
  - Reduz o número de cálculos;
  - Mas requer que todos os objetos sejam convertidos para coordenadas da *viewport*, inclusive aqueles que estão fora da janela de visualização.

Point Clipping – Recorte por pontos

- Comparamos qualquer pixel P = (x, y) contra os limites da janela ou da viewport:

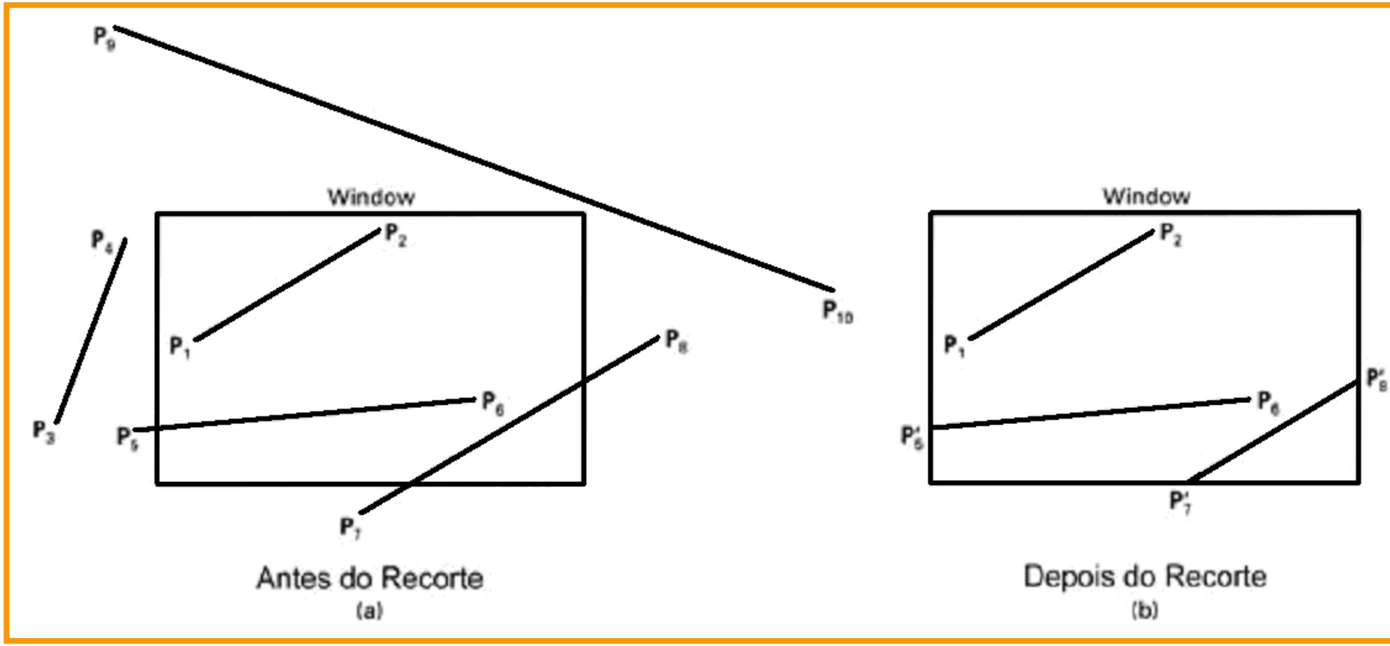


$$xw_{\min} \leq x \leq xw_{\max}$$

$$yw_{\min} \leq y \leq yw_{\max}$$

- Não é tão eficiente mas aplicável em alguns processos, como animação de partículas.

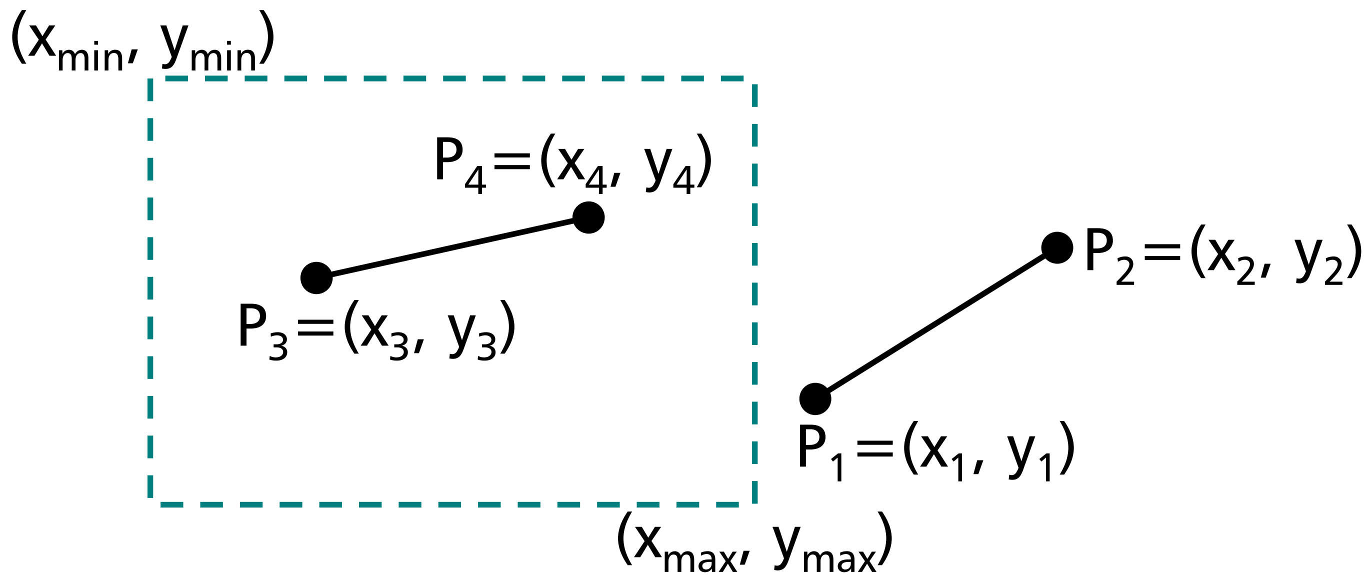
Line Clipping – Equações simultâneas



- Testar segmentos contra a janela de recorte:
  - Segmentos completamente dentro;
  - Segmentos completamente fora;
  - Segmentos que precisam ser recortados.

Line Clipping – Implementação

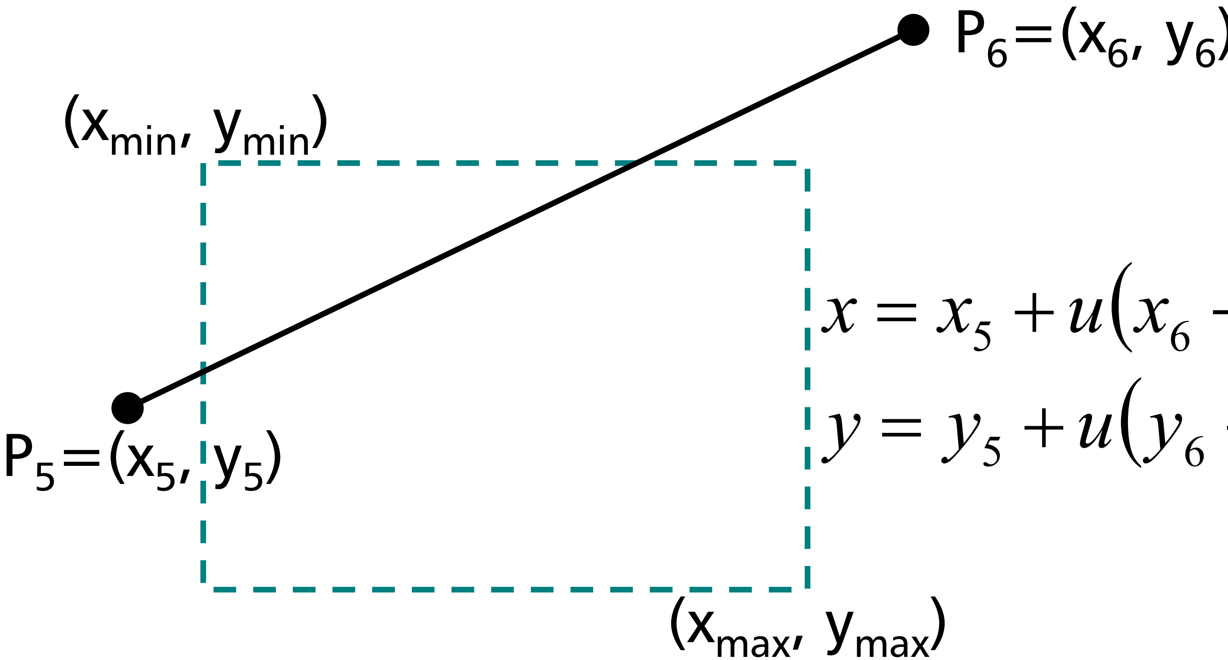
- Identificar, contra os limites da janela de recorte, segmentos com aceitação ou rejeição trivial.



- $x_1>x_{max}$ e $x_2>x_{max}$ $\rightarrow$ P_1P_2 fora;
- $x_{min}<x_3, x_4<x_{max}$ e $y_{min}<y_3, y_4<y_{max}$ $\rightarrow$ P_3P_4 dentro.

Line Clipping – Implementação

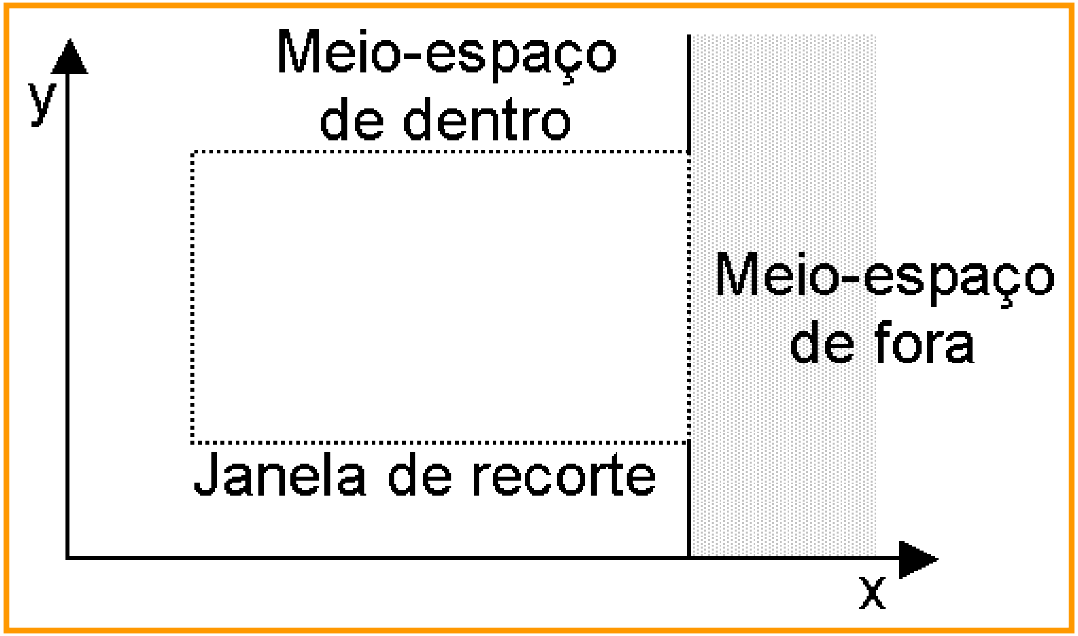
- Recortar os demais segmentos utilizando a equação paramétrica:



$$x = x_5 + u(x_6 - x_5)$$
$$y = y_5 + u(y_6 - y_5),\ 0 \leq u \leq 1$$

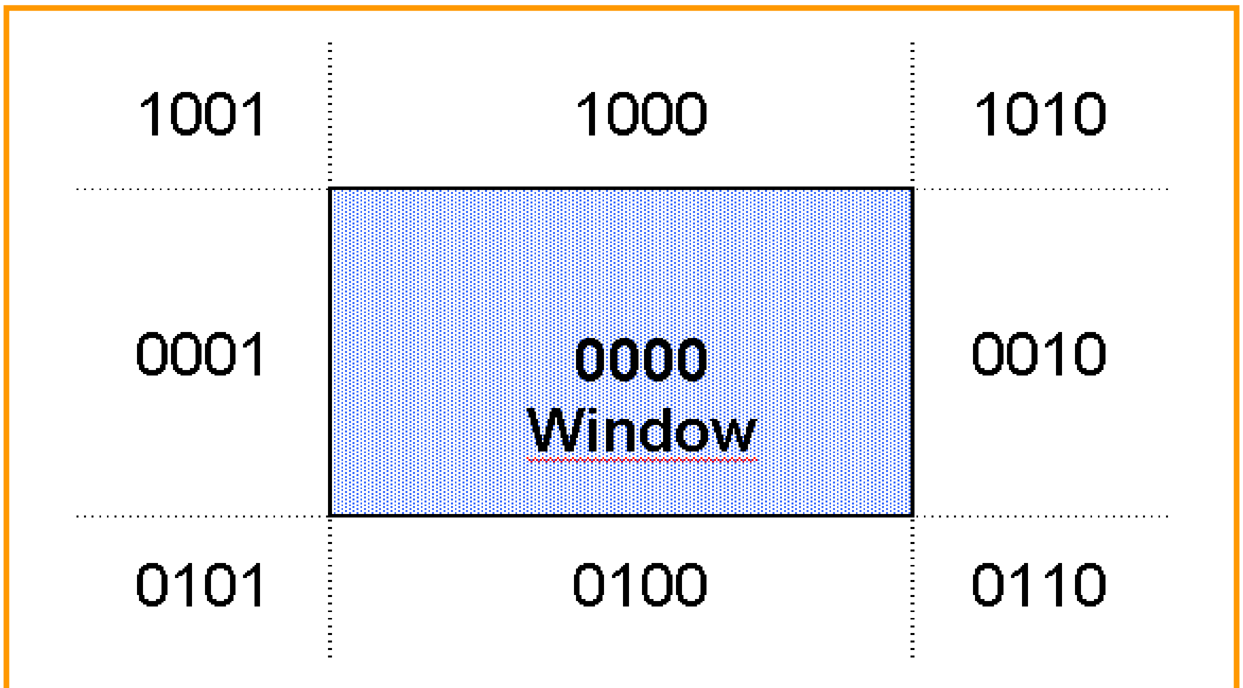
Algoritmo de Cohen-Sutherland

- Rapidamente detecta os casos triviais:
  - linhas inteiramente dentro ou inteiramente fora da área de recorte.
- Cada linha da janela define uma linha infinita que divide o espaço em dois meio-espaços.



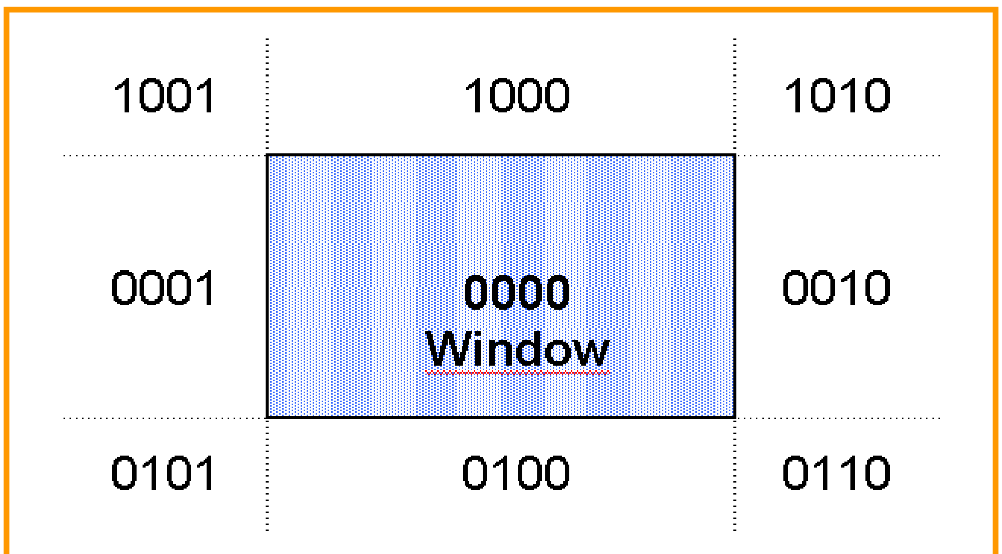
Algoritmo de Cohen-Sutherland

- Nove regiões são criadas:
  - 8 regiões externas;
  - 1 região interna.
- Associa-se um código de 4 bits para cada uma das regiões.



Algoritmo de Cohen-Sutherland

- Para qualquer ponto extremo de um segmento define-se seu código em relação à janela de recorte (TBRL);
  - L setado em 1: ponto à esquerda da janela → x < xmin;
  - R setado em 1: ponto à direita da janela→ x > xmax;
  - B setado em 1: ponto abaixo da janela → y > ymax;
  - T setado em 1: ponto acima da janela → y < ymin.



Algoritmo de Cohen-Sutherland

- Rejeição trivial:
  - AND lógico com os códigos correspondentes aos extremos do segmento;
  - Resultado diferente de zero → segmento fora.
- Aceitação trivial:
  - OR lógico com os códigos correspondentes aos extremos do segmento;
  - Resultado igual a zero → segmento dentro.
- Demais segmentos → recorte por equações simultâneas em função dos códigos dos extremos do segmento.

Algoritmo de Cohen-Sutherland

1 – Dado um segmento com extremos PQ;

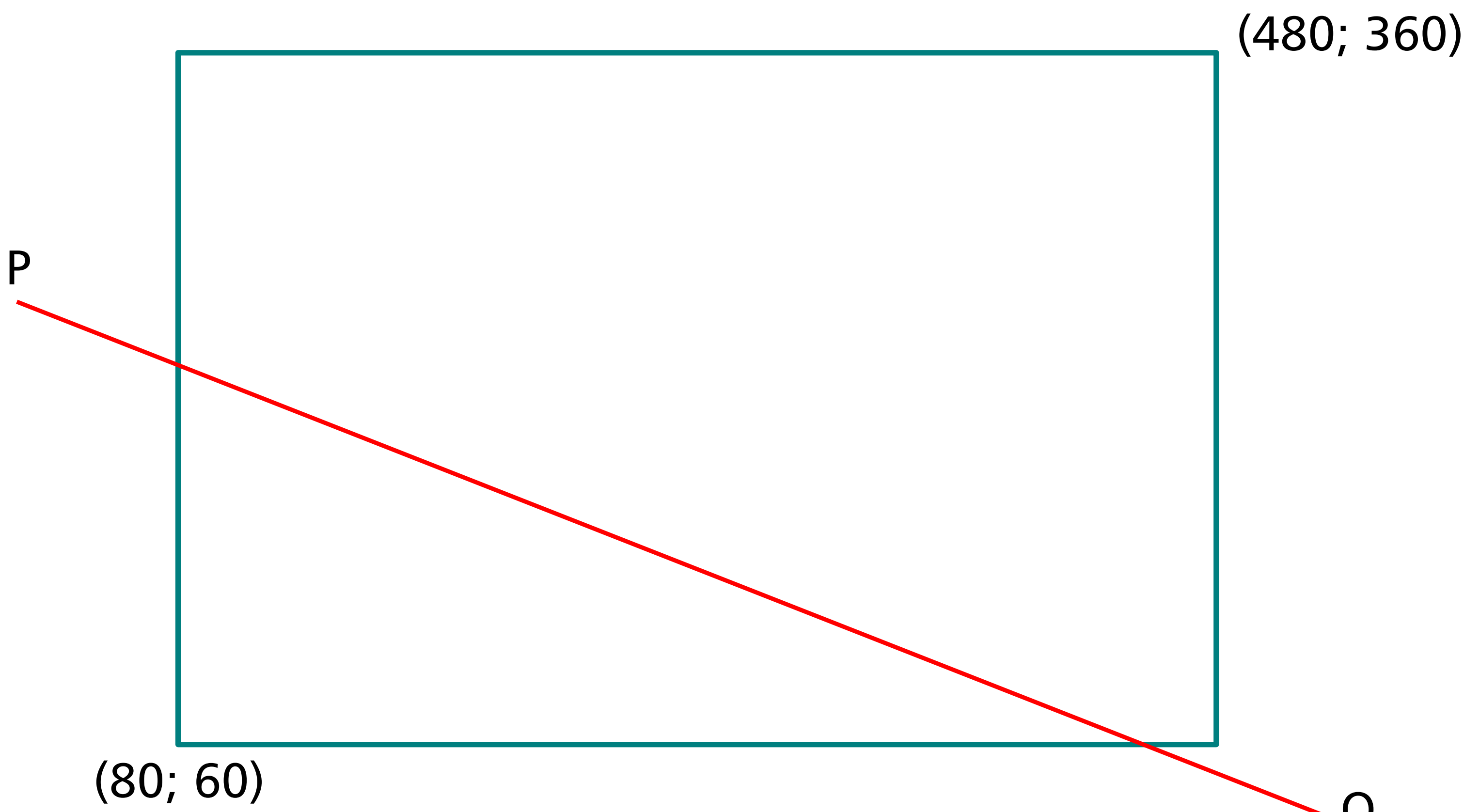
2 – Definir o código de 4 bits para cada extremo;

3 – Realizar os testes de rejeição/aceitação trivial (AND/OR);

4 – Se o segmento não sofrer rejeição/aceitação trivial:

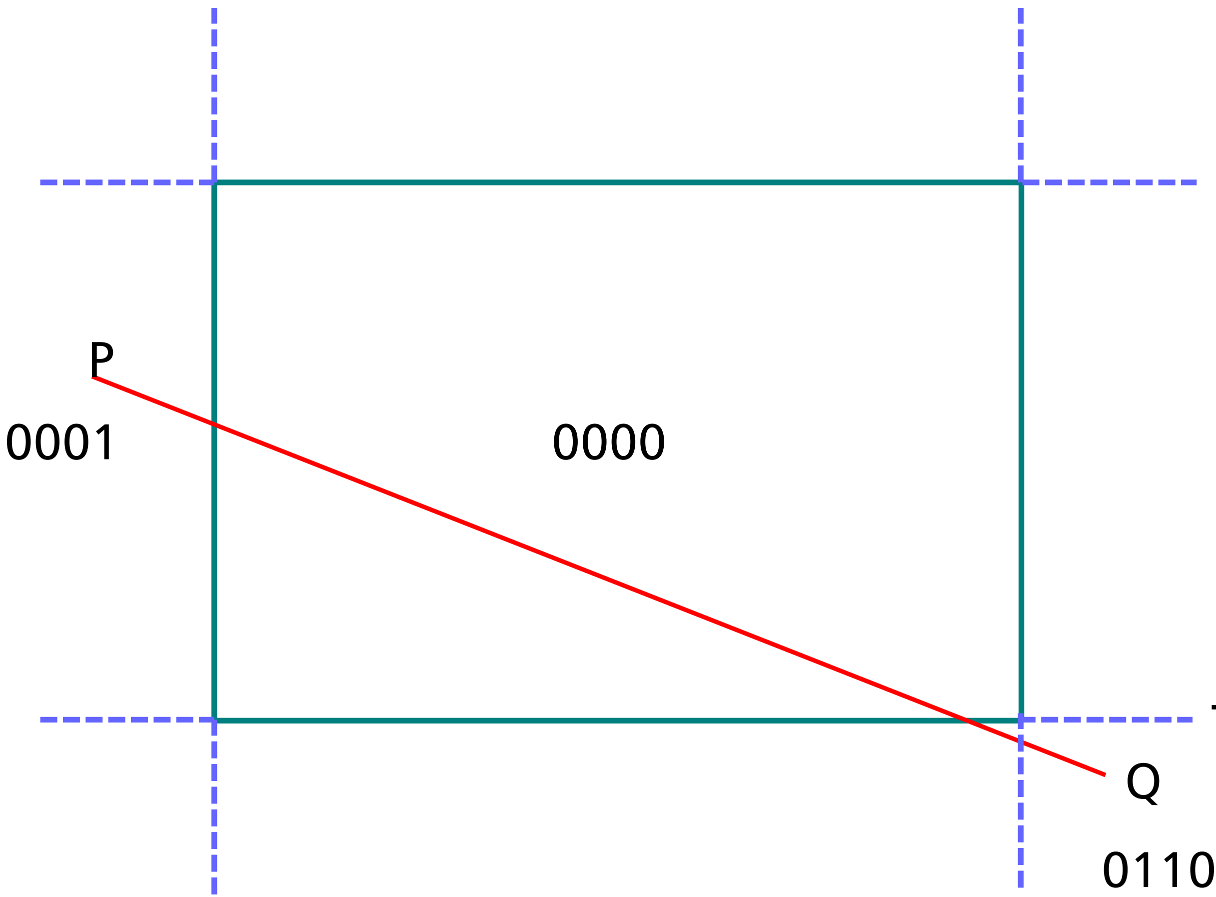
- 4.1 – Avaliar o código de cada extremo, da direita para a esquerda (TBRL), identificando contra qual extremo da janela o segmento deve ser recortado;
- 4.2 – Recortar o segmento utilizando as equações paramétricas;
- 4.3 – Definir o código de 4 bits para o novo extremo;
- 4.4 – Retornar ao passo 3.

Cohen-Sutherland – Exemplo 1



P=(10; 240)   Q=(560; 20)

Cohen-Sutherland – Exemplo 1



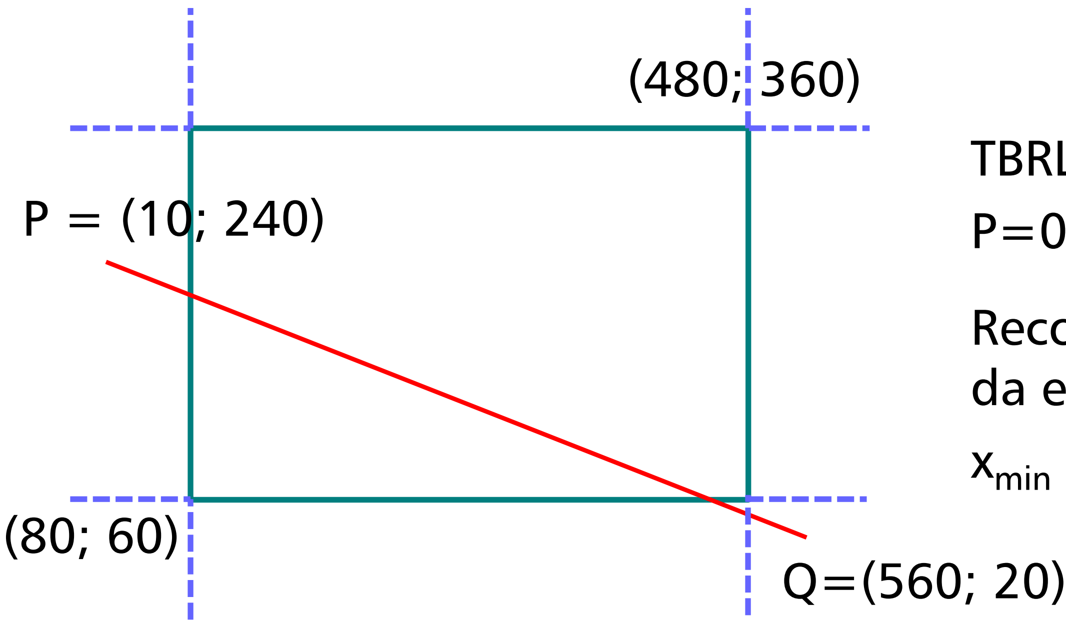
TBRL

P=0001

Q=0110

AND	OR
0001	0001
0110	0110
0000	0111

Cohen-Sutherland – Exemplo 1



TBRL

P=0001

Recortar contra a borda da esquerda.

$x_{min} = 80$

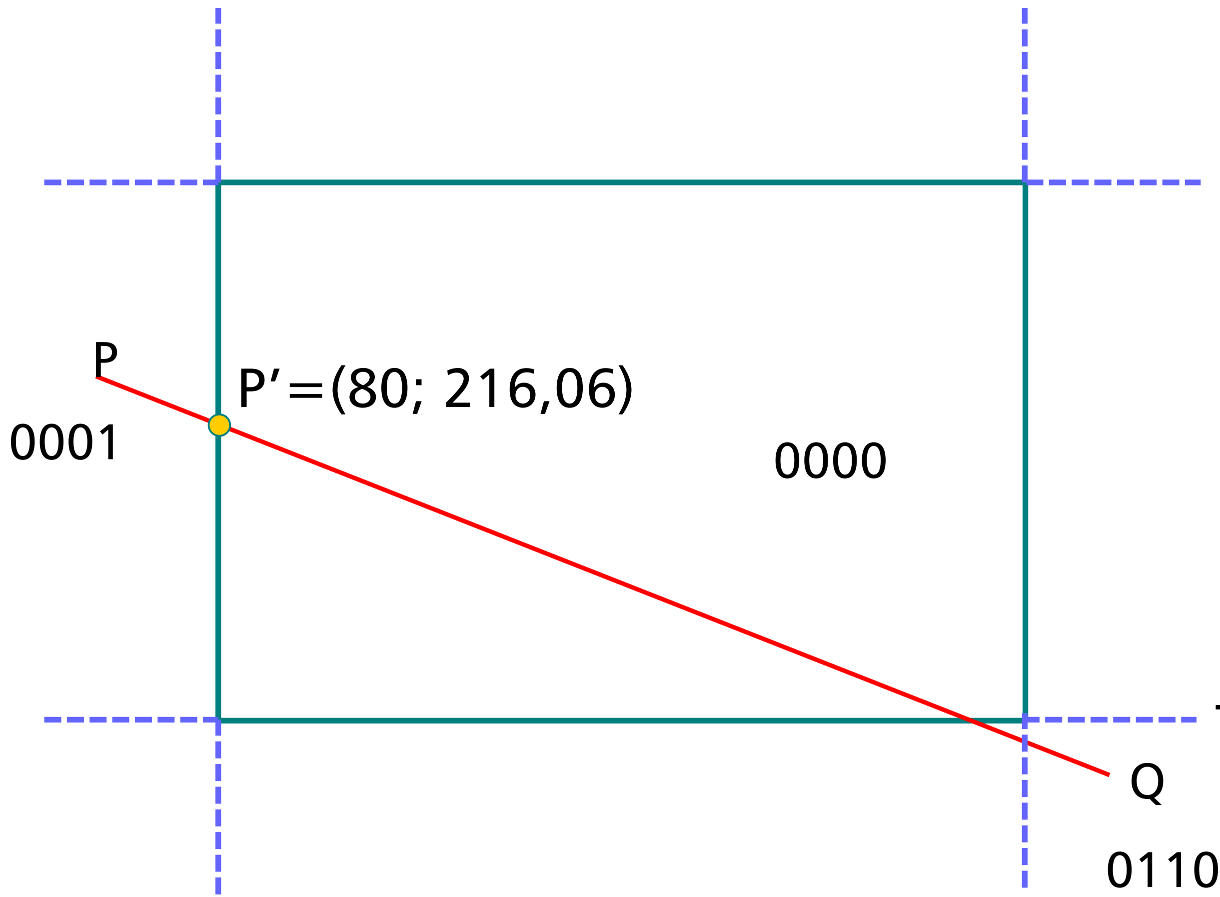
$$x = x_1 + u(x_2 - x_1)$$
$$80 = 10 + u(560 - 10)$$
$$u = \frac{70}{550} = 0{,}127$$

$$y = y_1 + u(y_2 - y_1)$$
$$y = 240 + 0{,}127(20 - 240)$$
$$y = 216{,}06$$

P'=(80; 216,06)

P'=0000

Cohen-Sutherland – Exemplo 1



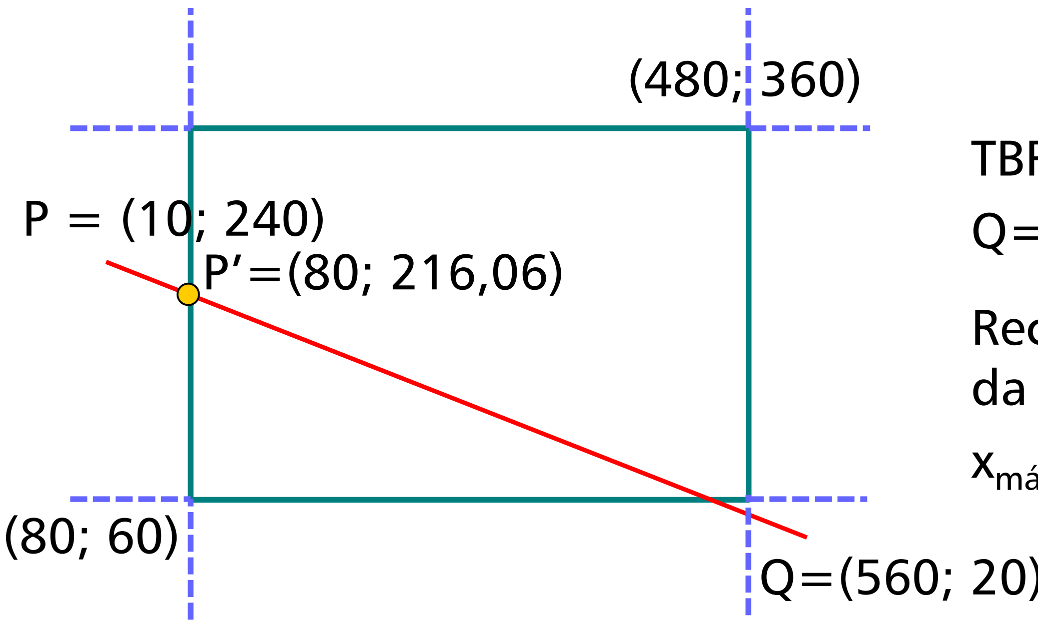
TBRL

P’=0000

Q=0110

AND	OR
0000	0000
0110	0110
0000	0110

Cohen-Sutherland – Exemplo 1



TBRL

Q=0110

Recortar contra a borda da direita.

$x_{máx} = 480$

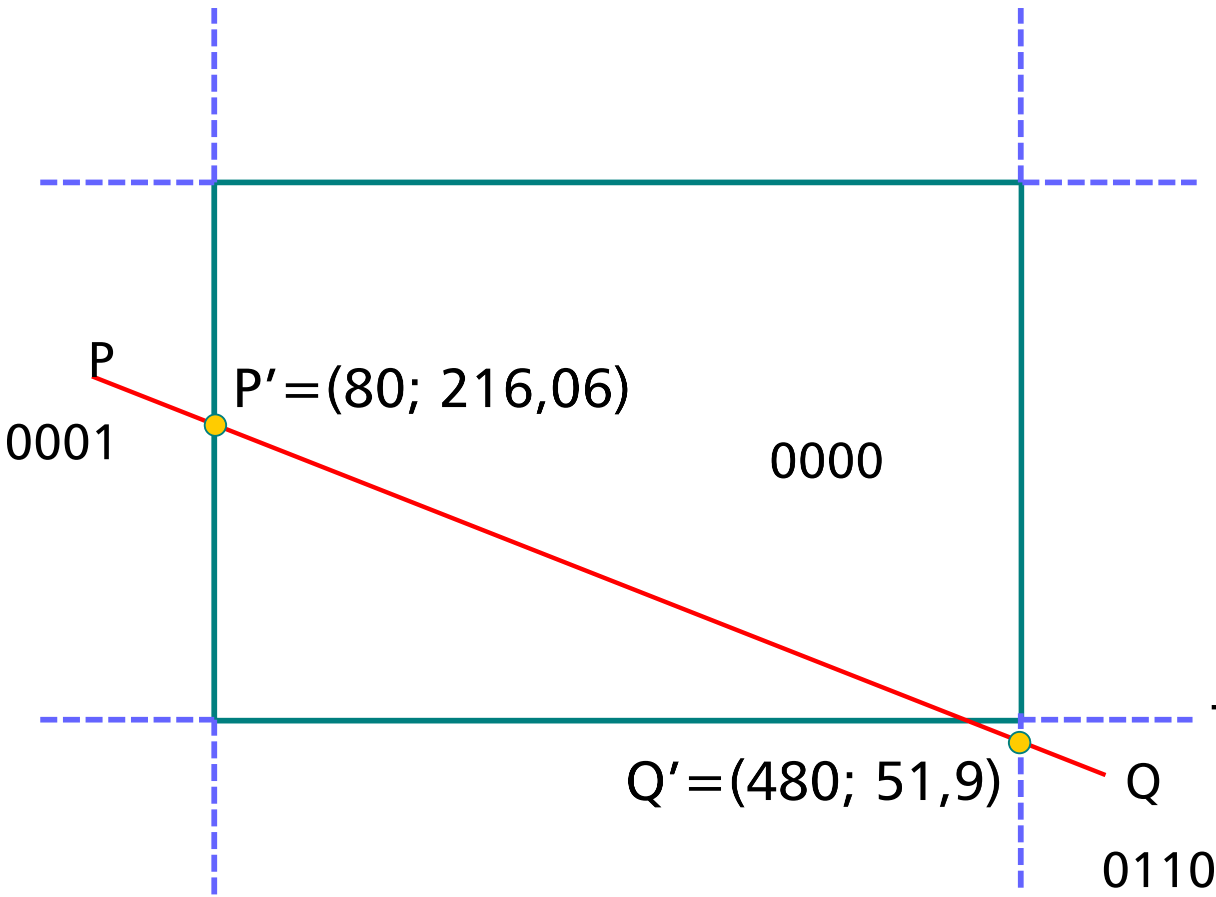
$$x = x_1 + u(x_2 - x_1)$$
$$480 = 10 + u(560 - 10)$$
$$u = \frac{470}{550} = 0{,}855$$

$$y = y_1 + u(y_2 - y_1)$$
$$y = 240 + 0{,}855(20 - 240)$$
$$y = 51{,}9$$

Q'=(480; 51,9)

Q'=0100

Cohen-Sutherland – Exemplo 1



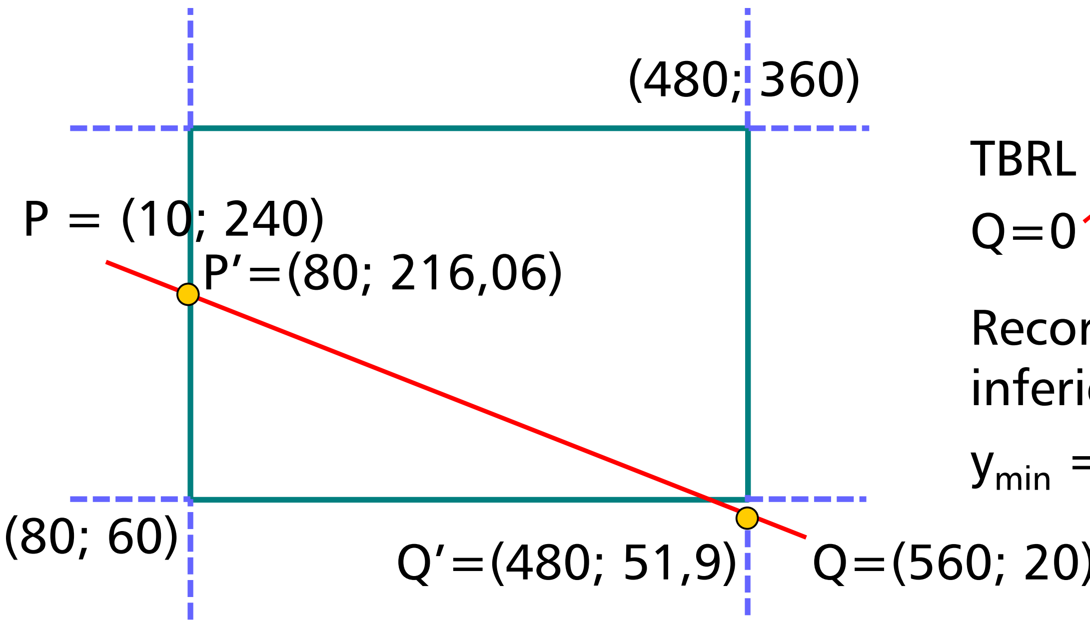
TBRL

P’=0000

Q’=0100

AND	OR
0000	0000
0100	0100
0000	0100

Cohen-Sutherland – Exemplo 1



TBRL

Q=0100

Recortar contra a borda inferior.

$y_{min} = 60$

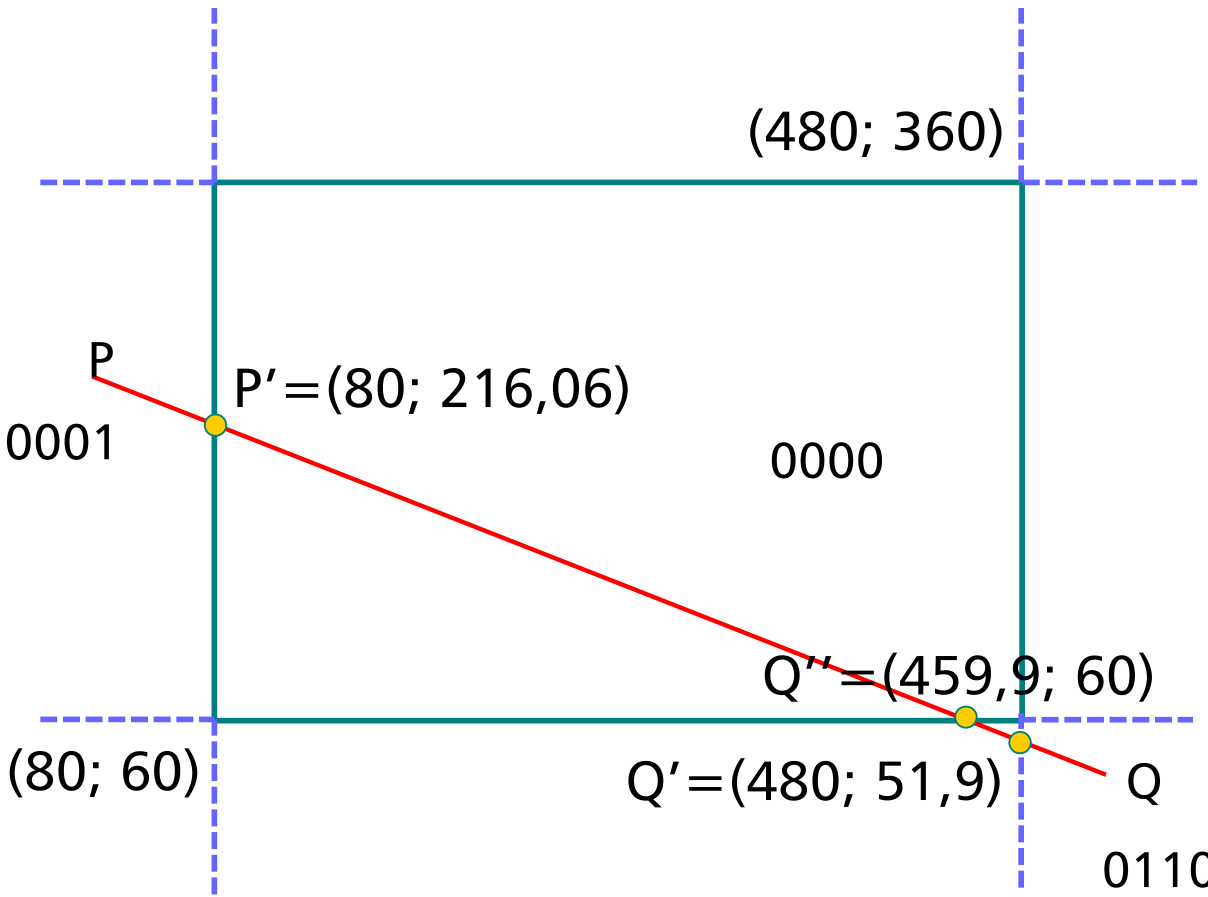
$$y = y_1 + u(y_2 - y_1)$$
$$60 = 240 + u(20 - 240)$$
$$u = \frac{-180}{-220} = 0{,}818$$

$$x = x_1 + u(x_2 - x_1)$$
$$x = 10 + 0{,}818(560 - 10)$$
$$x = 459{,}9$$

Q’’=(459,9; 60)

Q’’=0000

Cohen-Sutherland – Exemplo 1



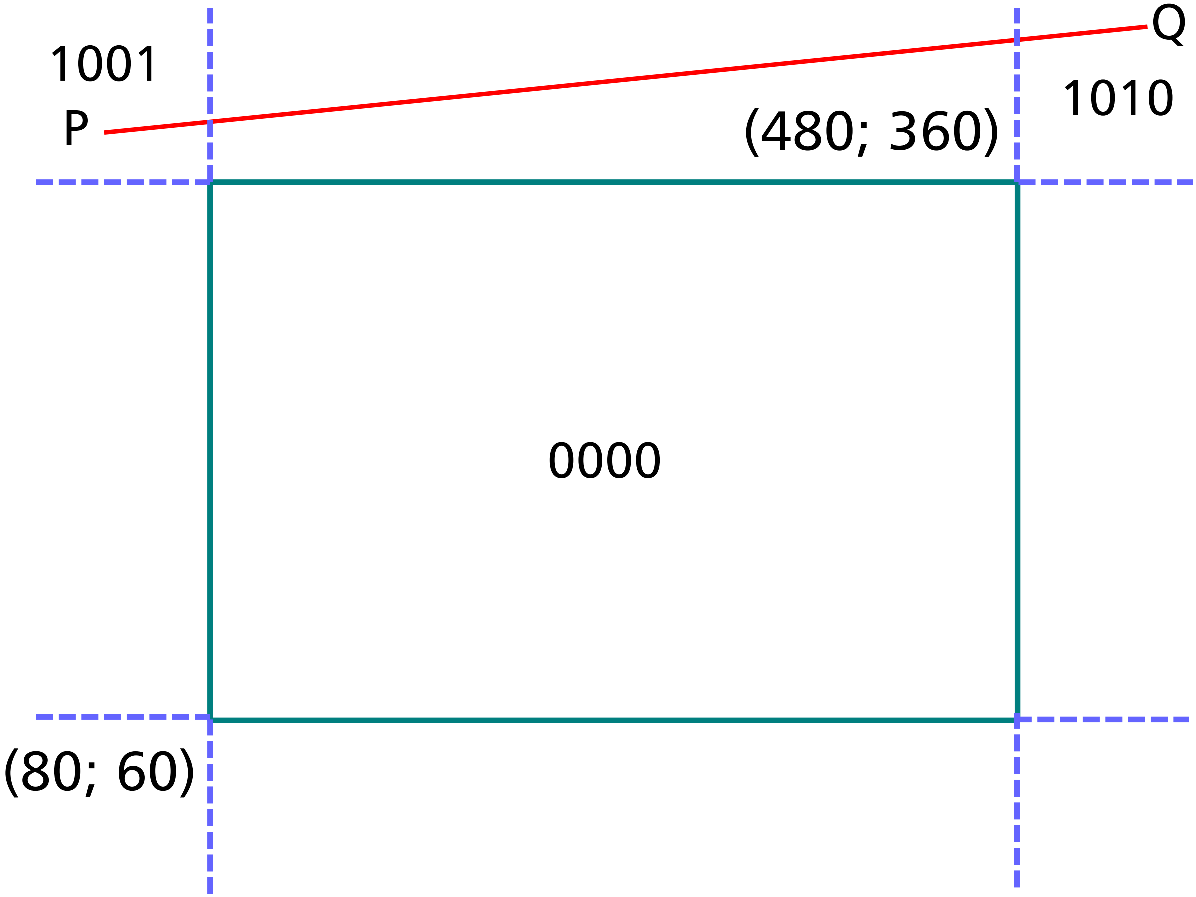
TBRL

P’=0000

Q’’=0000

AND	OR
0000	0000
0000	0000
0000	0000

Cohen-Sutherland – Exemplo 2



TBRL

P=1001

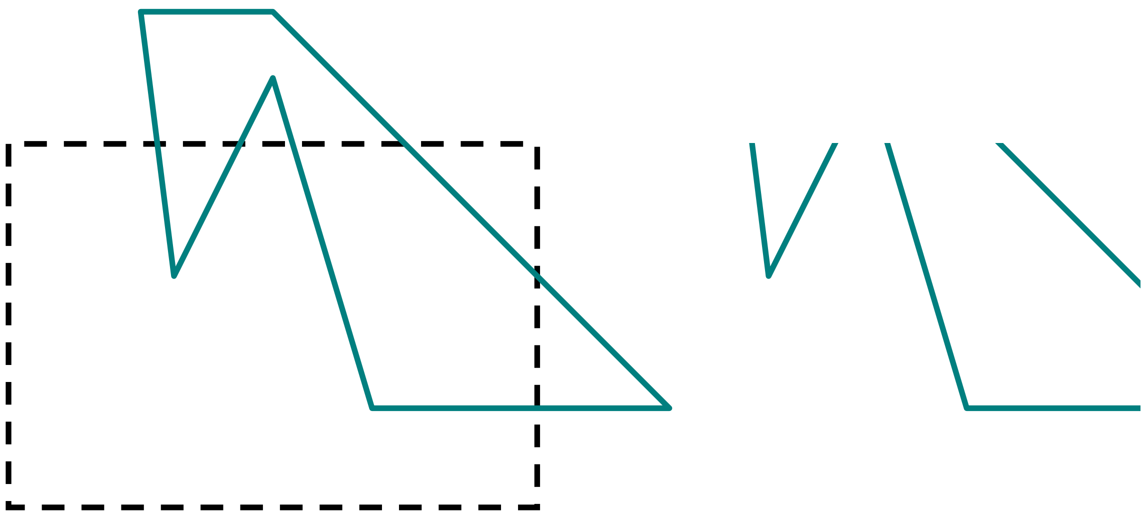
Q=1010

AND	OR
1001	1001
1010	1010
1000	1011

Rejeição Trivial

Recorde de Polígonos

- Os algoritmos de recortes de linhas podem gerar resultados incorretos.

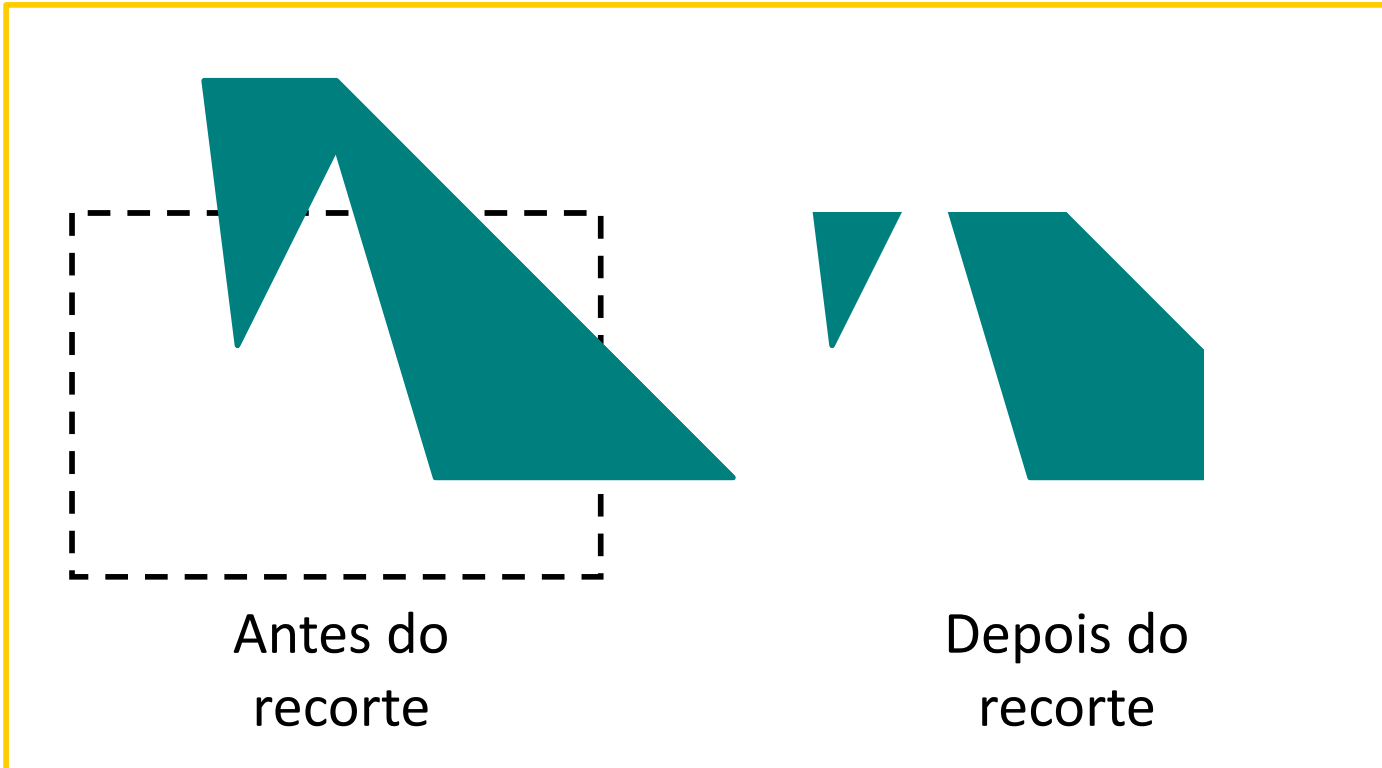


Antes do recorte

Depois do recorte

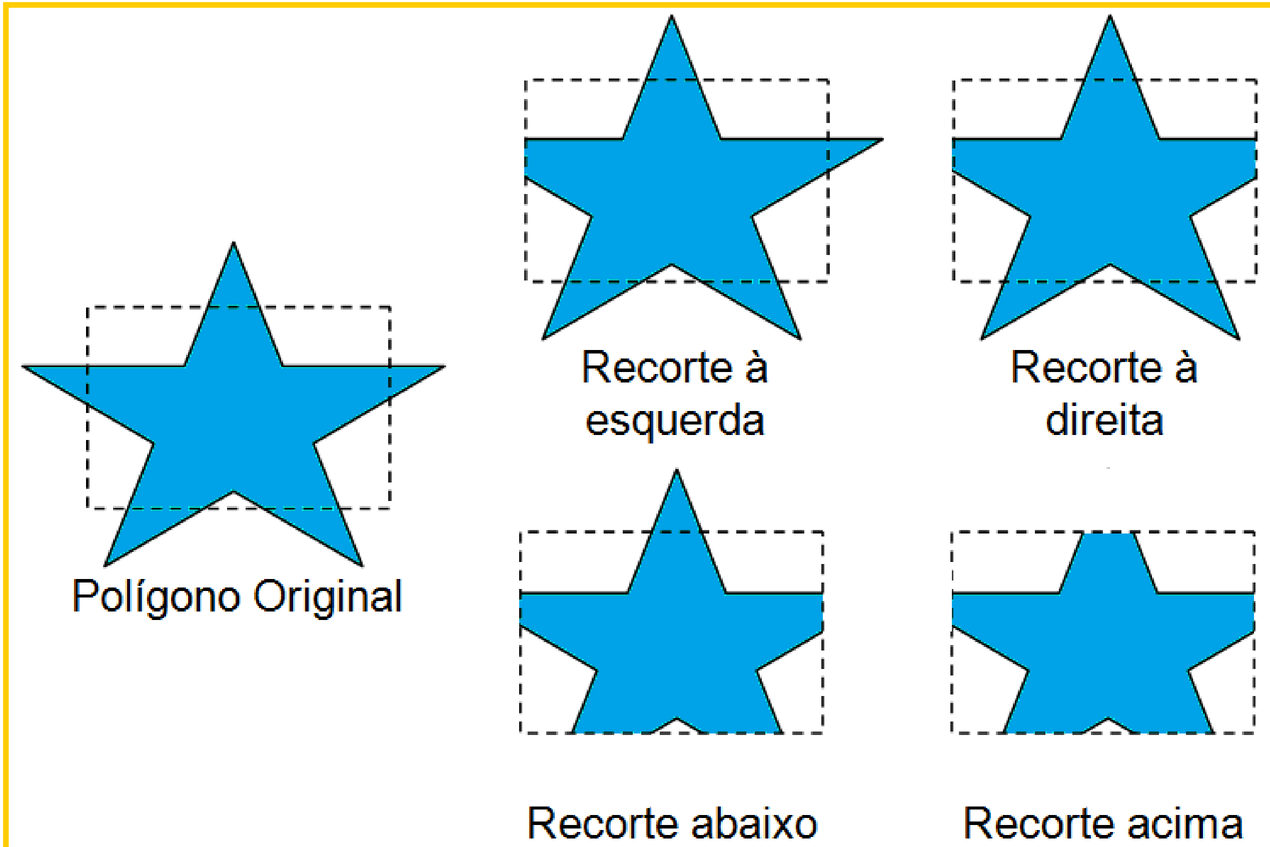
Recorde de Polígonos

- Para obter o resultado correto é necessário adaptar o algoritmo de recorte de linhas.



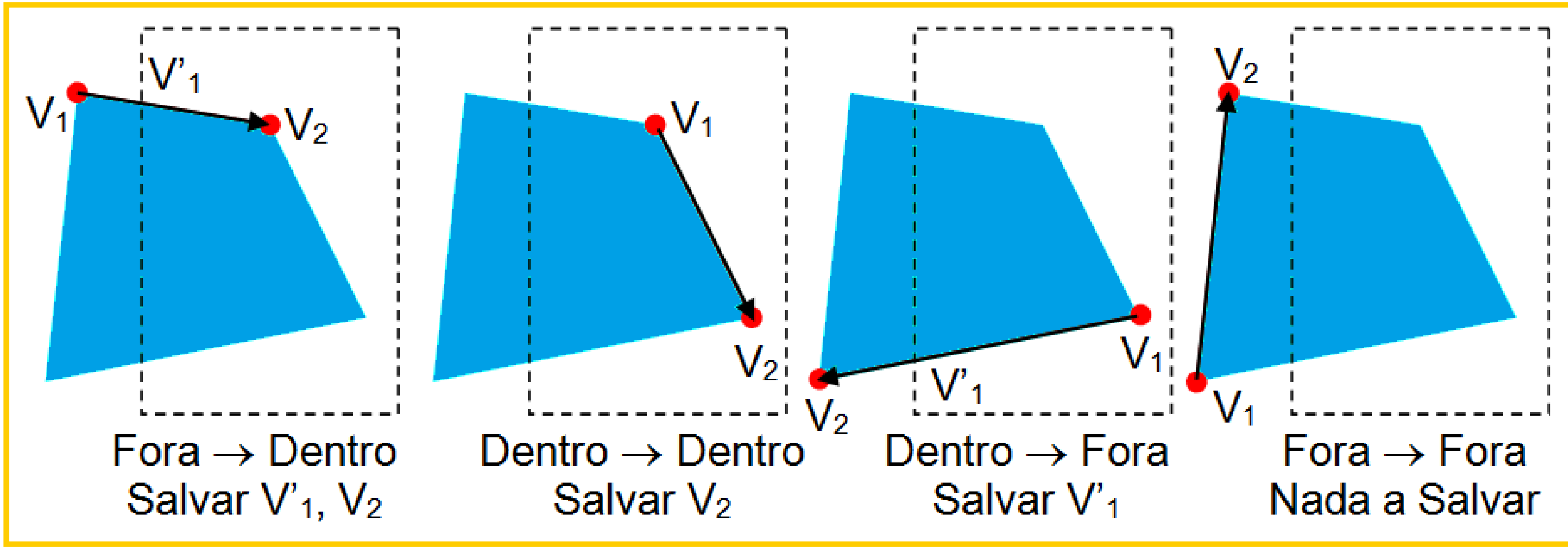
Algoritmo de Sutherland-Hodgeman

- Recorta todas as arestas do polígono contra cada borda da janela de recorte.



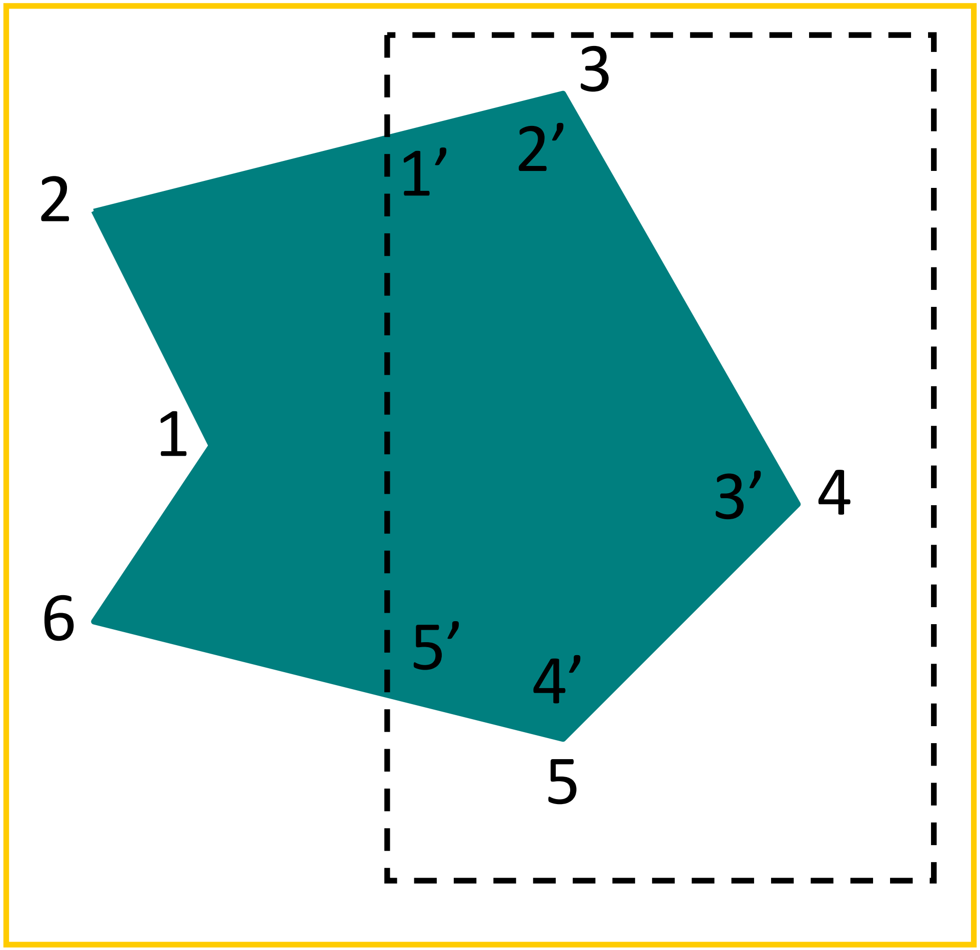
Algoritmo de Sutherland-Hodgeman

- Há 4 casos possíveis quando processamos os vértices ao redor do perímetro do polígono.



- Depois de recortar todas as arestas do polígono contra uma das bordas da janela de recorte, a lista de vértices de saída é recortada contra a próxima borda da janela.

Sutherland-Hodgeman – Exemplo



Lista inicial:

1, 2, 3, 4, 5, 6

Processo:

1 → 2 = nenhum

2 → 3 = 1’ e 2’

3 → 4 = 3’

4 → 5 = 4’

5 → 6 = 5’

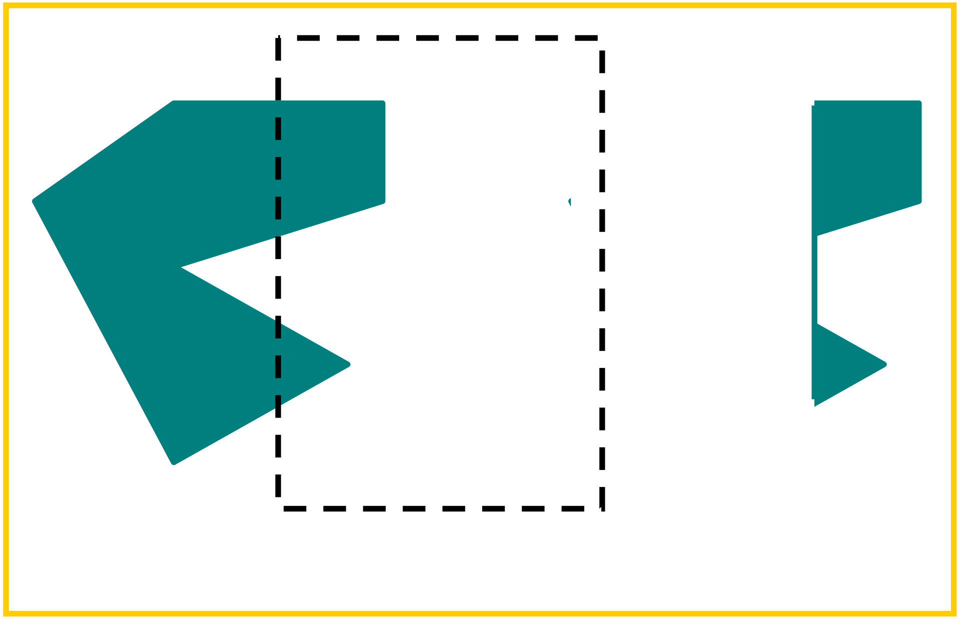
6 → 1 = nenhum

Lista de saída:

1’, 2’, 3’, 4’, 5’

Algoritmo de Sutherland-Hodgeman

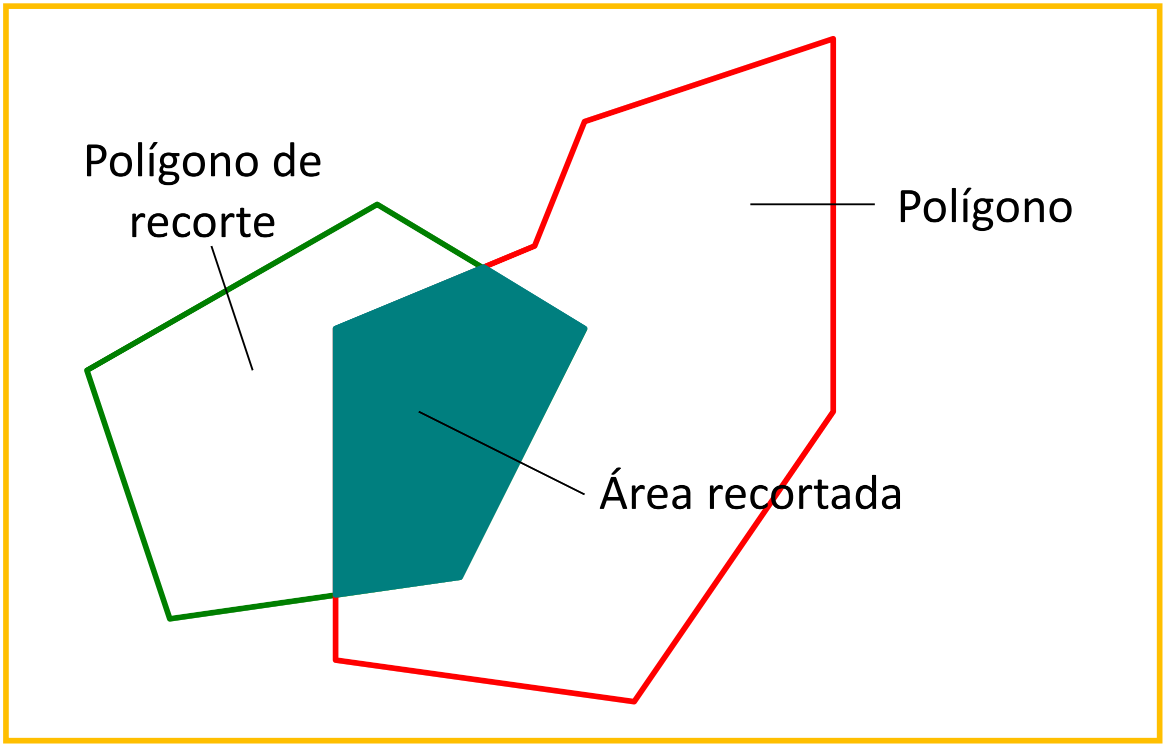
- Adequado para polígonos convexos;
- Para alguns polígonos côncavos podem surgir linhas fantasmas.



- Solução para polígonos côncavos → Algoritmo de Weiler-Atherton.

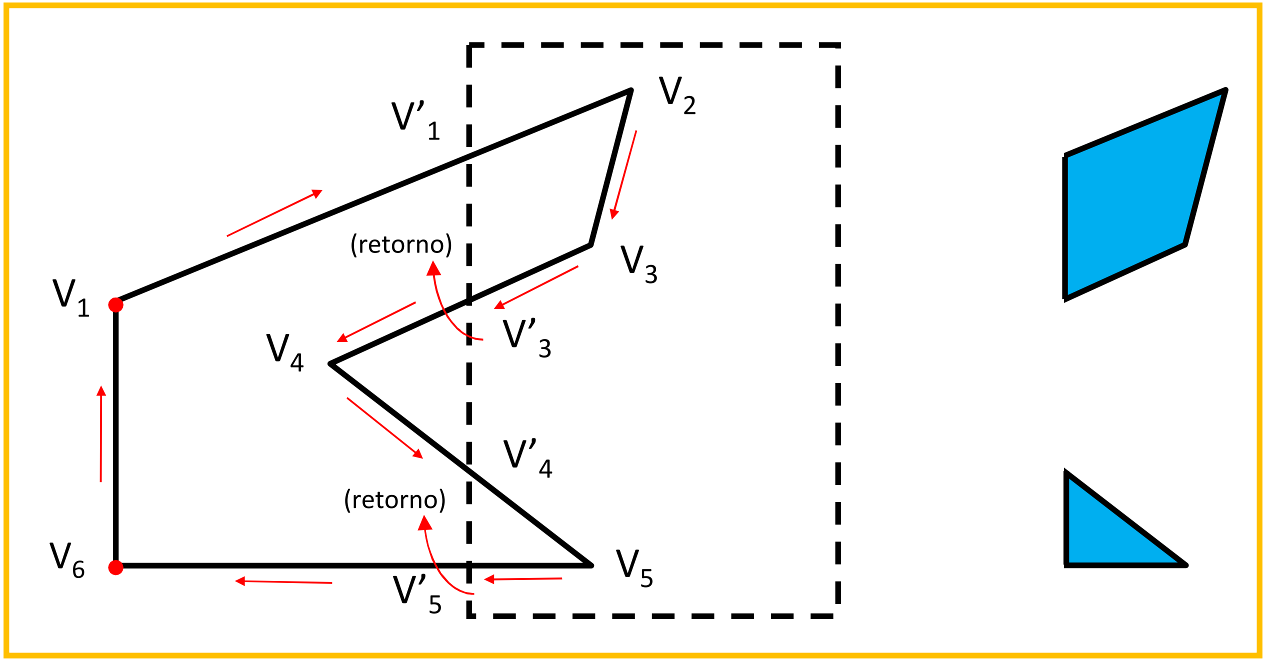
Algoritmo de Weiler-Atherton

- Permite recortar polígonos de formas arbitrárias;
- Alterna o caminhamento entre as arestas do polígono e as bordas da área de recorte.



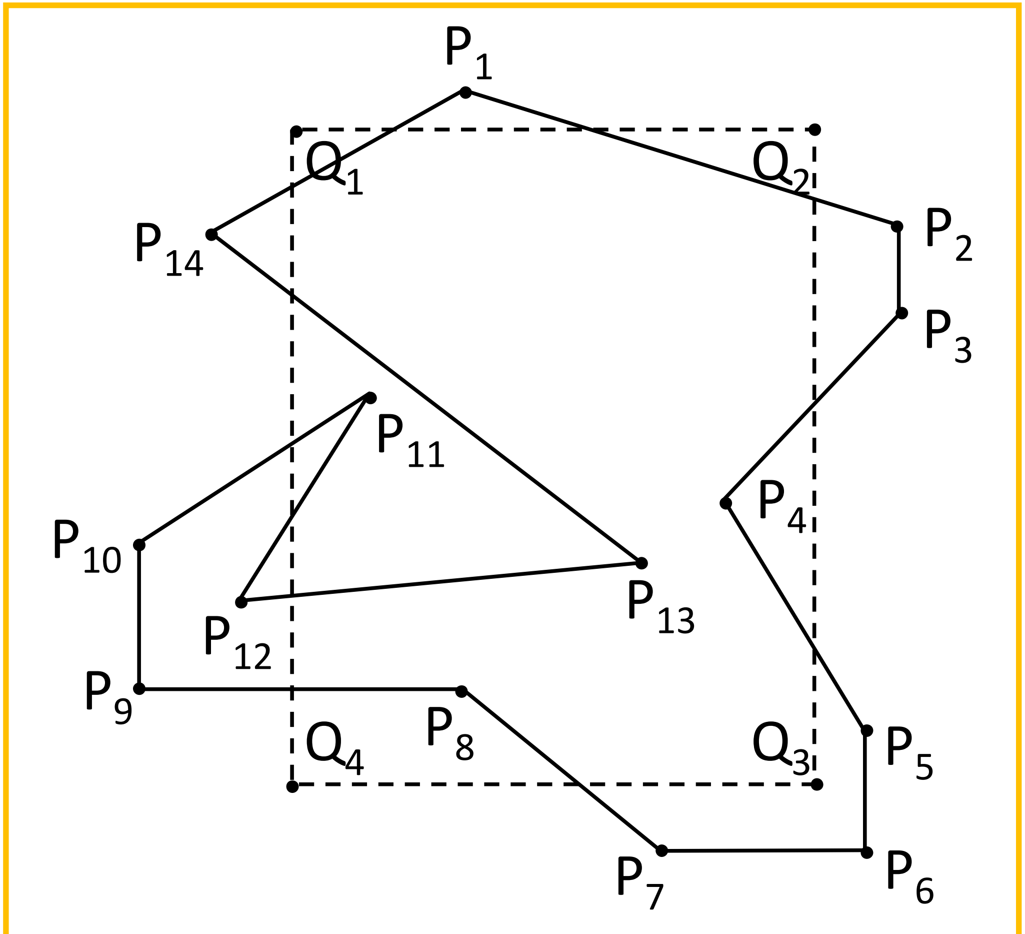
Algoritmo de Weiler-Atherton

- Para o processamento em sentido horário:
  - Par de vértices fora-dentro → siga as arestas do polígono;
  - Par de vértices dentro-fora → siga as arestas da área de recorte.



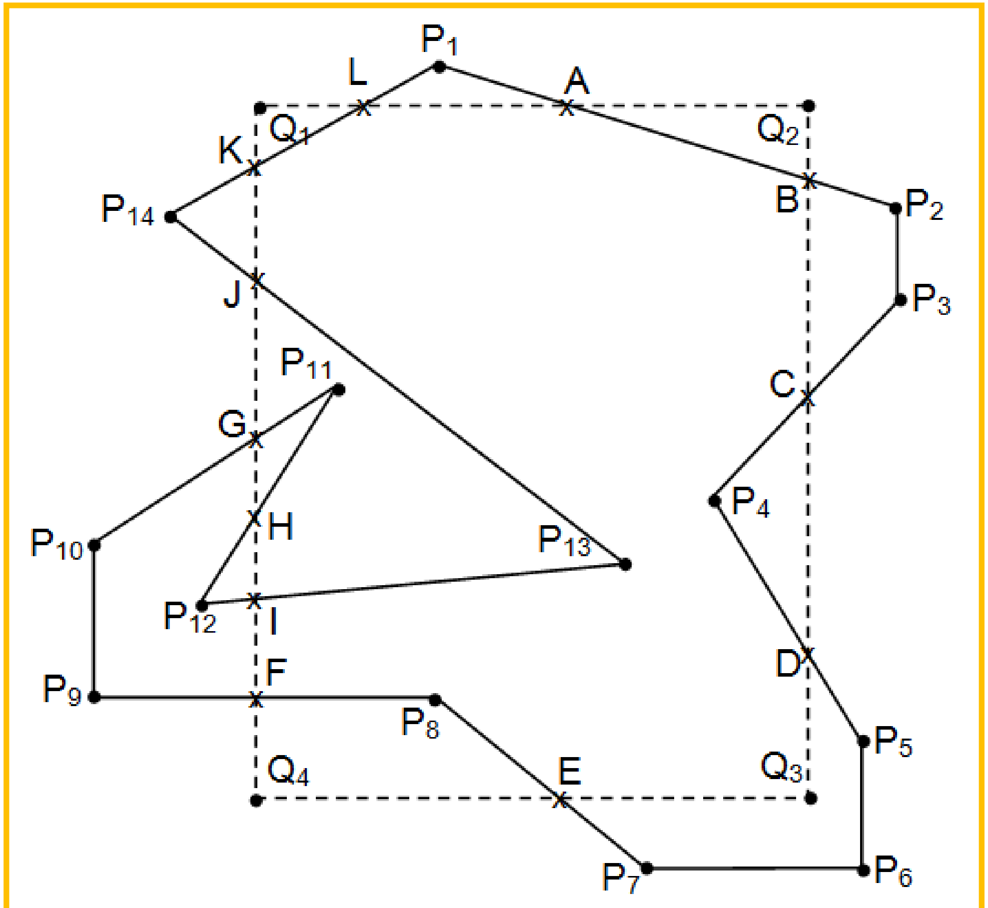
Algoritmo de Weiler-Atherton – Exemplo

- O polígono P será recortado contra a janela Q.



Algoritmo de Weiler-Atherton – Exemplo

- Calcular todas as interseções entre arestas do polígono P e janela Q, rotulando-as como mostrado na figura.



Algoritmo de Weiler-Atherton – Exemplo

- Montar as 3 listas auxiliares.

Lista 1: Caminhar sobre o polígono. Adicionar os vértices e interseções calculadas.

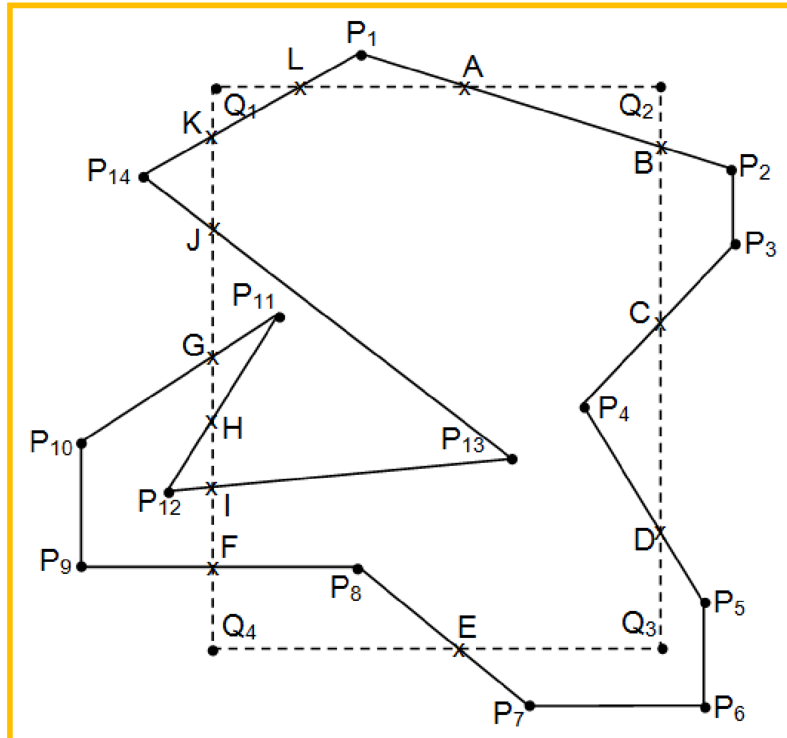
L1 (polígono):

$P_1 \rightarrow A \rightarrow B \rightarrow P_2 \rightarrow P_3 \rightarrow C \rightarrow P_4 \rightarrow D \rightarrow P_5 \rightarrow P_6 \rightarrow P_7 \rightarrow E \rightarrow P_8 \rightarrow F \rightarrow P_9 \rightarrow P_{10} \rightarrow G \rightarrow P_{11} \rightarrow H \rightarrow P_{12} \rightarrow I \rightarrow P_{13} \rightarrow J \rightarrow P_{14} \rightarrow K \rightarrow L$

Lista 2: Caminhar sobre a janela de recorte.

L2 (janela):

$Q_1 \rightarrow L \rightarrow A \rightarrow Q_2 \rightarrow B \rightarrow C \rightarrow D \rightarrow Q_3 \rightarrow E \rightarrow Q_4 \rightarrow F \rightarrow I \rightarrow H \rightarrow G \rightarrow J \rightarrow K$



Lista 3 (vértices): Vértices de interseção para arestas do polígono que adentram a janela de recorte:

L3: $A \rightarrow C \rightarrow E \rightarrow G \rightarrow I \rightarrow K$

Algoritmo de Weiler-Atherton – Exemplo

- Retirar um vértice da Lista 3;
- A partir deste vértice caminhar sobre a lista 1 (polígono) até encontrar um vértice de interseção;
- Alternar para lista 2 (janela) e caminhar sobre ela a partir do vértice da interseção;
- Ao encontrar outra interseção alternar e caminhar novamente sobre a lista 1 e assim sucessivamente;
- Parar quando encontrar um vértice de L3 já analisado.

Algoritmo de Weiler-Atherton – Exemplo

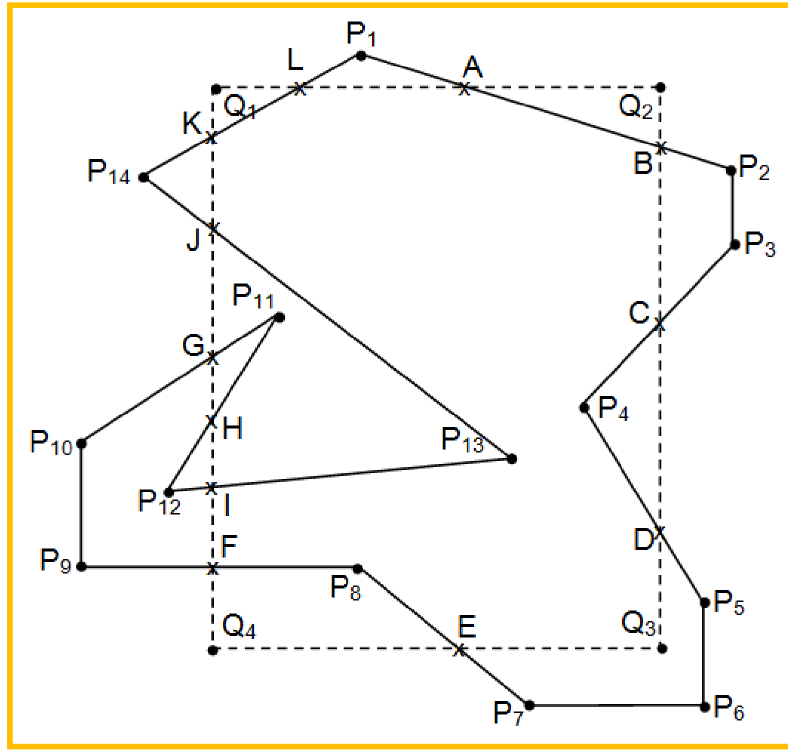
L1 (polígono):

$P_1 \rightarrow A \rightarrow B \rightarrow P_2 \rightarrow P_3 \rightarrow C \rightarrow P_4 \rightarrow D \rightarrow P_5 \rightarrow$
$P_6 \rightarrow P_7 \rightarrow E \rightarrow P_8 \rightarrow F \rightarrow P_9 \rightarrow P_{10} \rightarrow G \rightarrow$
$P_{11} \rightarrow H \rightarrow P_{12} \rightarrow I \rightarrow P_{13} \rightarrow J \rightarrow P_{14} \rightarrow K \rightarrow L$

L2 (janela):

$Q_1 \rightarrow L \rightarrow A \rightarrow Q_2 \rightarrow B \rightarrow C \rightarrow D \rightarrow Q_3 \rightarrow E \rightarrow$
$Q_4 \rightarrow F \rightarrow I \rightarrow H \rightarrow G \rightarrow J \rightarrow K$

L3: $A \rightarrow C \rightarrow E \rightarrow G \rightarrow I \rightarrow K$



- Iniciar em L3 → A;
- Polígono 1: A→L1:B→L2:C→L1:P_4→L1:D→L2:Q_3→L2:E→ L1:P_8→L1:F→L2:I→L1:P_{13}→L1:J→L2:K→L1:L→L2:A

Algoritmo de Weiler-Atherton – Exemplo

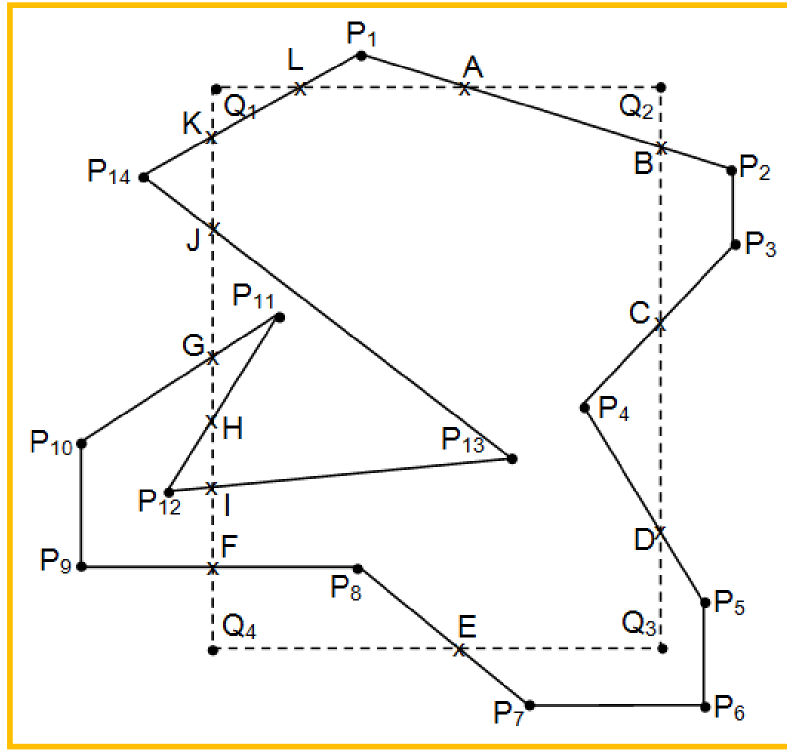
L1 (polígono):

$P_1 \rightarrow A \rightarrow B \rightarrow P_2 \rightarrow P_3 \rightarrow C \rightarrow P_4 \rightarrow D \rightarrow P_5 \rightarrow$
$P_6 \rightarrow P_7 \rightarrow E \rightarrow P_8 \rightarrow F \rightarrow P_9 \rightarrow P_{10} \rightarrow G \rightarrow$
$P_{11} \rightarrow H \rightarrow P_{12} \rightarrow I \rightarrow P_{13} \rightarrow J \rightarrow P_{14} \rightarrow K \rightarrow L$

L2 (janela):

$Q_1 \rightarrow L \rightarrow A \rightarrow Q_2 \rightarrow B \rightarrow C \rightarrow D \rightarrow Q_3 \rightarrow E \rightarrow$
$Q_4 \rightarrow F \rightarrow I \rightarrow H \rightarrow G \rightarrow J \rightarrow K$

L3: A→C→E→G→I→K

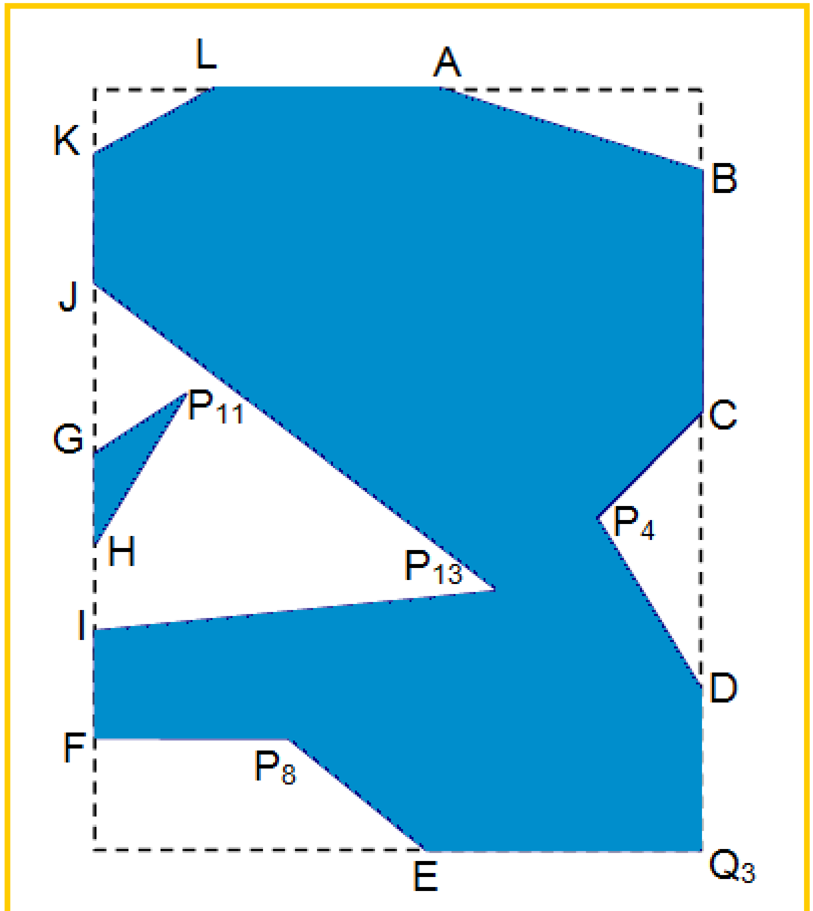


- Iniciar em L3 → G;
- Polígono 2: G→L1:P_{11}→L1:H→L2:G

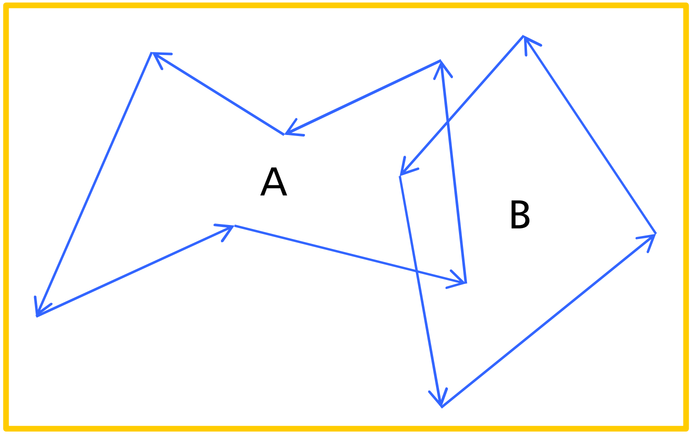
L3: A→C→E→G→I→K → Toda a lista L3 foi processada.

Algoritmo de Weiler-Atherton – Exemplo

- Dois polígonos são identificados.

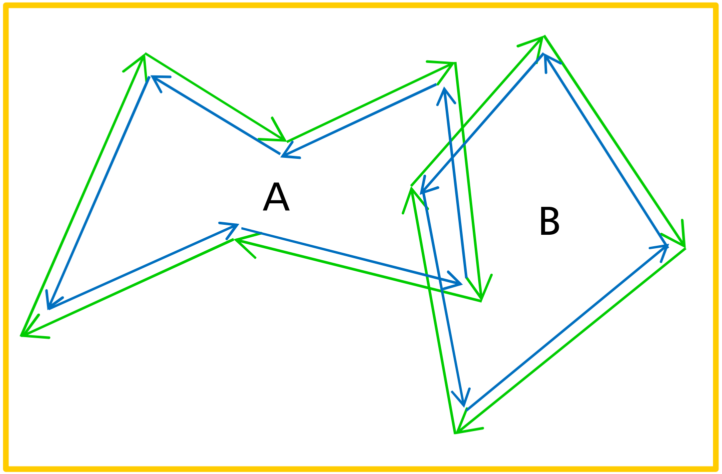


Mais – Algoritmo de Weiler-Atherton



Borda interna do polígono

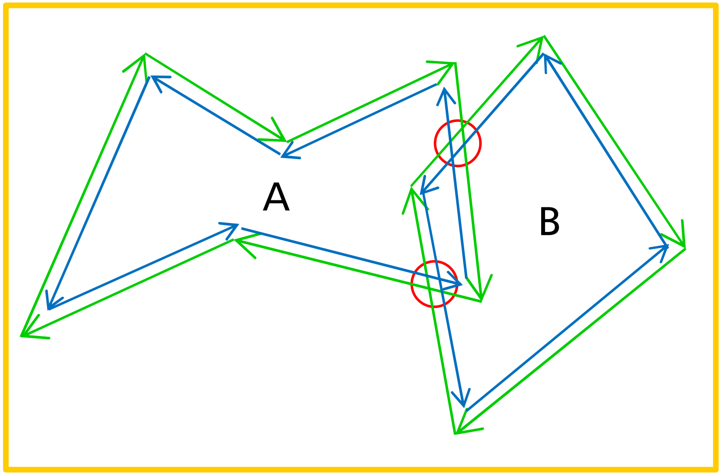
Sentido anti-horário



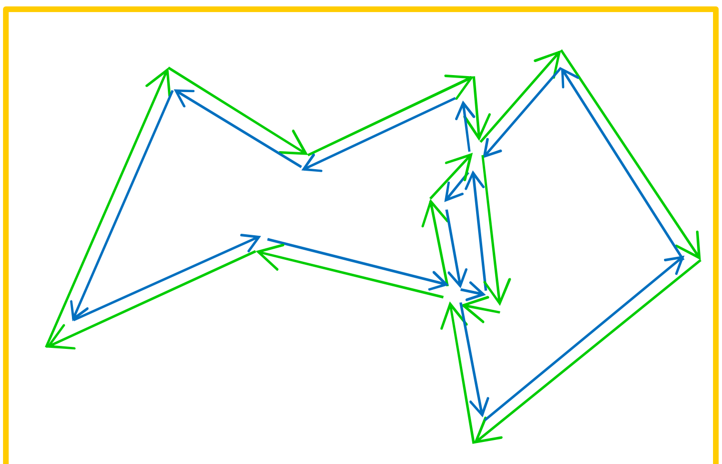
Borda externa do polígono

Sentido horário

Mais – Algoritmo de Weiler-Atherton

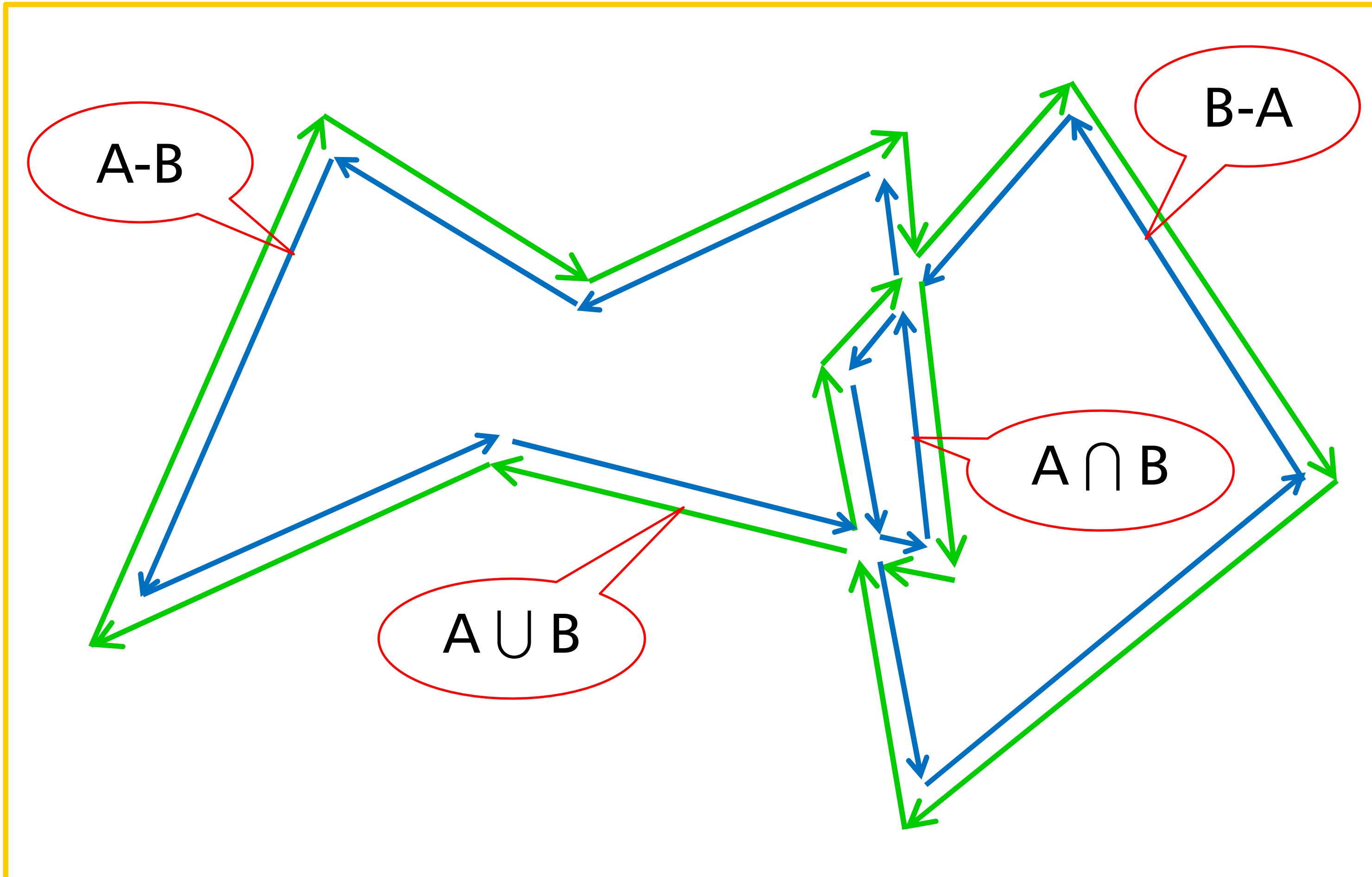


Calcular os pontos de interseção



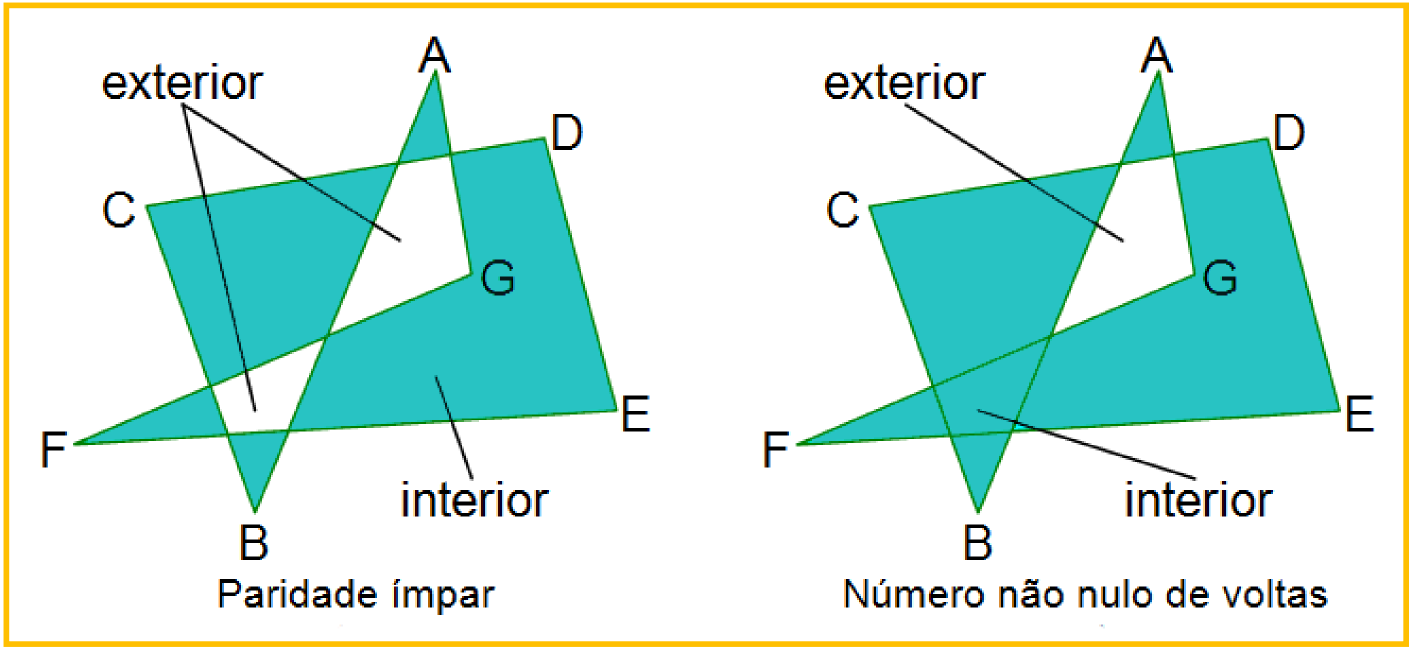
Costurar adequadamente as regiões

Mais – Algoritmo de Weiler-Atherton



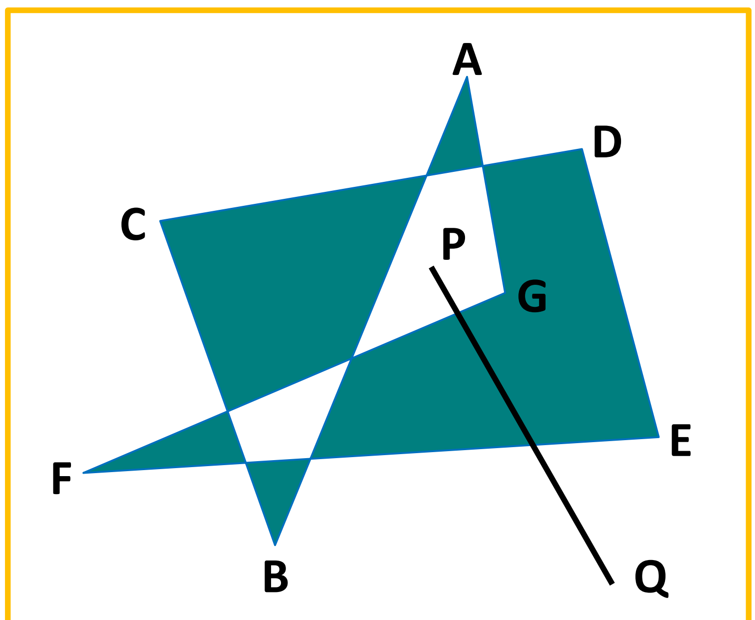
Testes Dentro-Fora

- Dois métodos:
  - Regra da paridade ímpar;
  - Regra do número não-nulo de voltas.
- Resultados distintos.



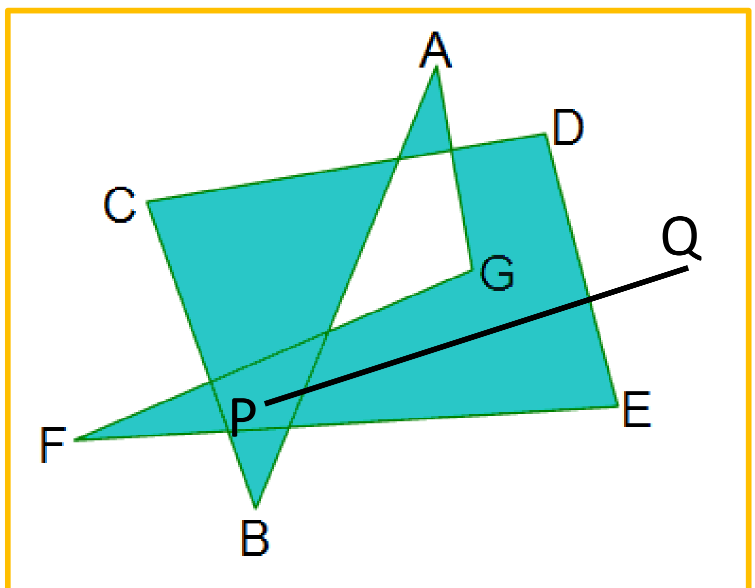
Regra da Paridade Ímpar

- Testar o ponto P:
  - Escolher um ponto Q, externo e distante do polígono; PQ não pode passar por nenhum vértice do polígono;
  - Contar quantas arestas são interceptadas pelo segmento PQ:
    - Se for ímpar → P é interno ao polígono;
    - Se for par → P é externo ao polígono.



Regra do Número Não-Nulo de Voltas

- Testar o ponto P:
  - Escolher um ponto Q, como no método anterior;
  - Para cada aresta interceptada pelo segmento PQ:
    - aresta cruza PQ da direita para a esquerda → ++contador;
    - aresta cruza PQ da esquerda para a direita → --contador;
    - Se, após processar todos os cruzamentos, o contador for não-nulo, então P é interno ao polígono, senão é externo.



Regra do Número Não-Nulo de Voltas

- Para determinar a direção do cruzamento fazemos:
  - Determinar o vetor $\boldsymbol{u} = Q - P$;
  - Determinar os vetores correspondentes às arestas, por exemplo $\boldsymbol{E}_{AB} = B - A$;
  - Calcular o produto vetorial $\boldsymbol{u} \times \boldsymbol{E}_{AB}$;
  - Se a componente z do produto vetorial for positiva a aresta cruza PQ da direita para a esquerda → +1;
  - Se a componente z for negativa a aresta cruza PQ da esquerda para a direita → -1;

  - Ou:

  - Com $\boldsymbol{u} = (ux, uy)$, fazer $\boldsymbol{u'} = (-uy, ux)$;
  - Calcular o produto escalar $\boldsymbol{u'}.\boldsymbol{E}_{AB}$;
  - Se o produto escalar for positivo → +1;
  - Senão → -1;