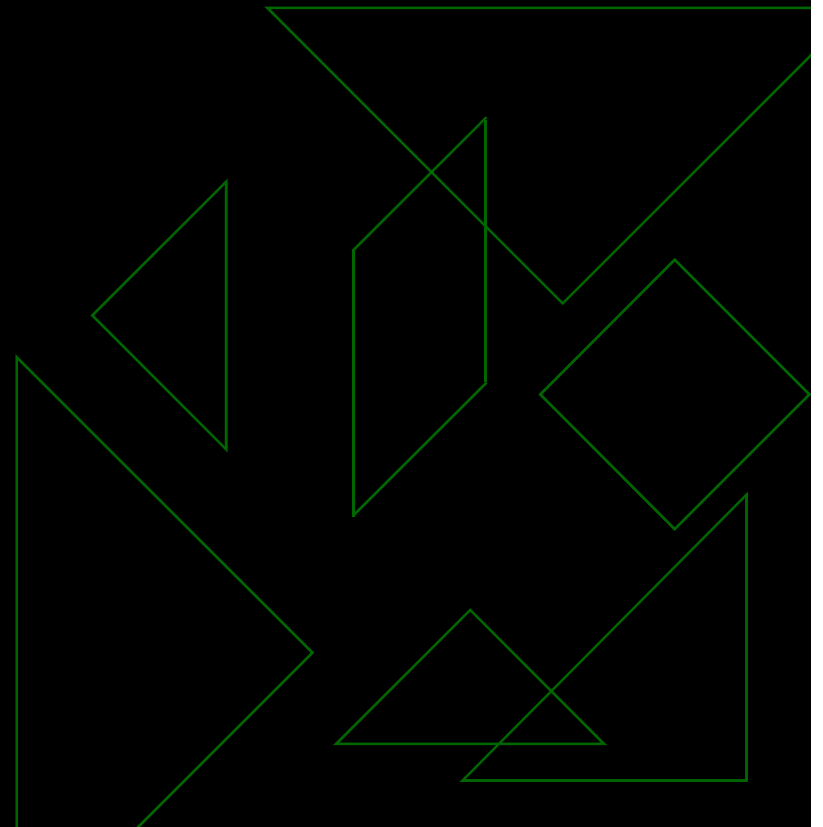


Rasterização de primitivas 2D e Pipeline 2D



Soraia Raupp Musse

Algoritmos de rasterização para primitivas 2D



Objetivo:

**Aproximar primitivas
matemáticas descritas
através de vértices por
meio de um conjunto de
pixels de determinada
cor**

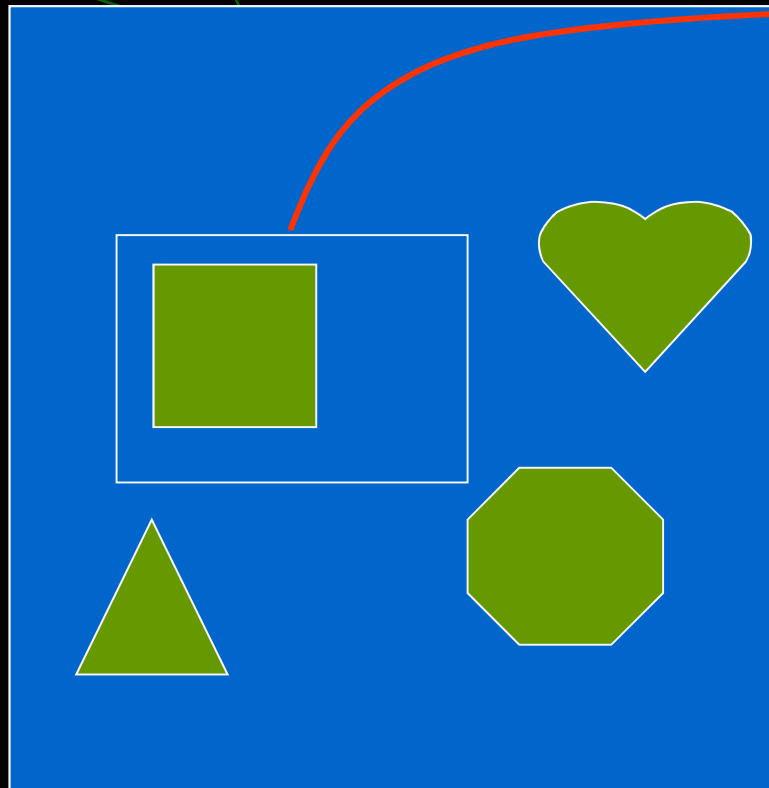
Modelo 2D

Display



Pipeline de visualização 2D

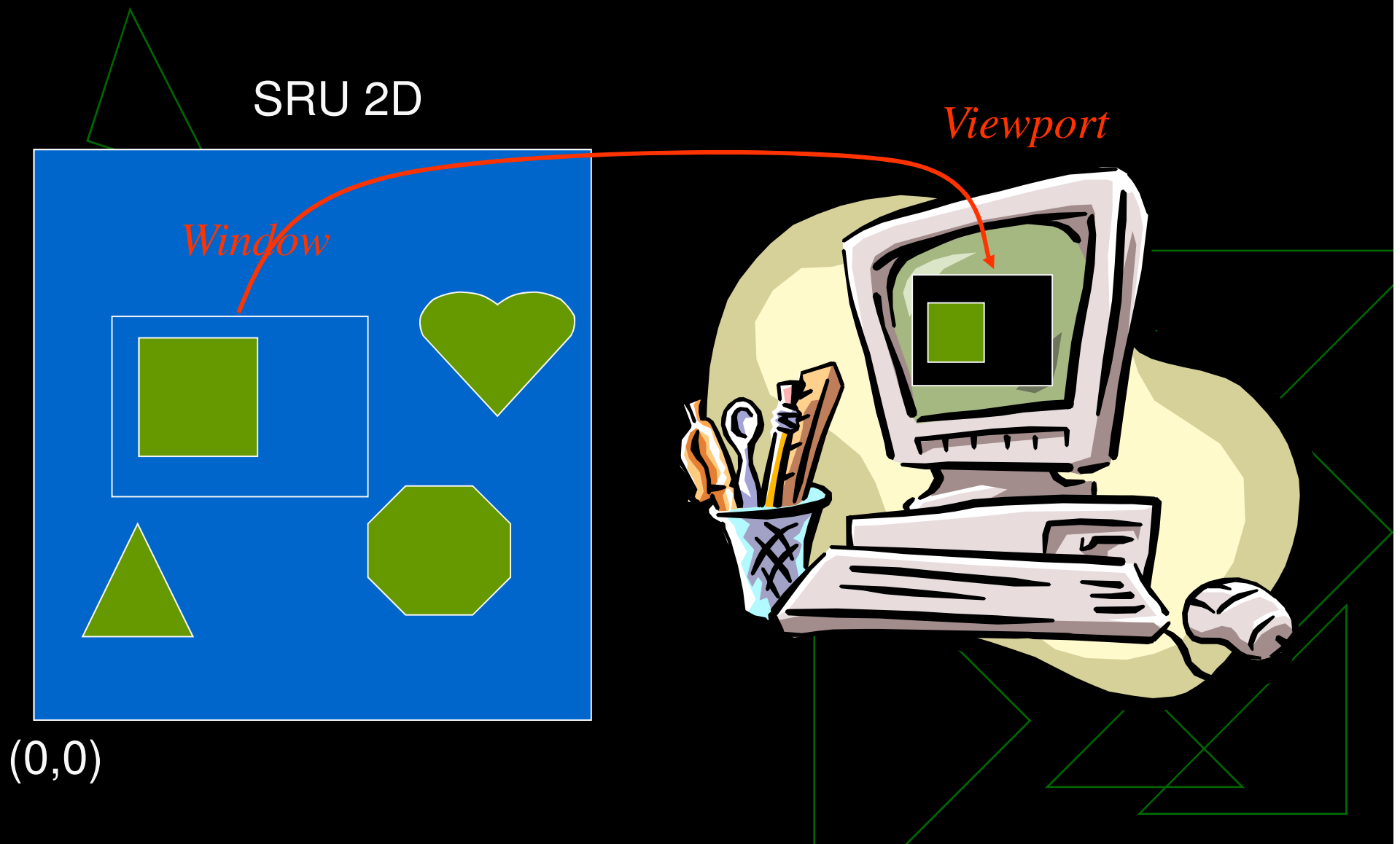
SRU 2D



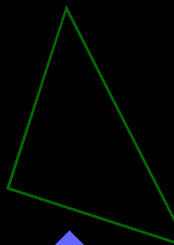
(0,0)

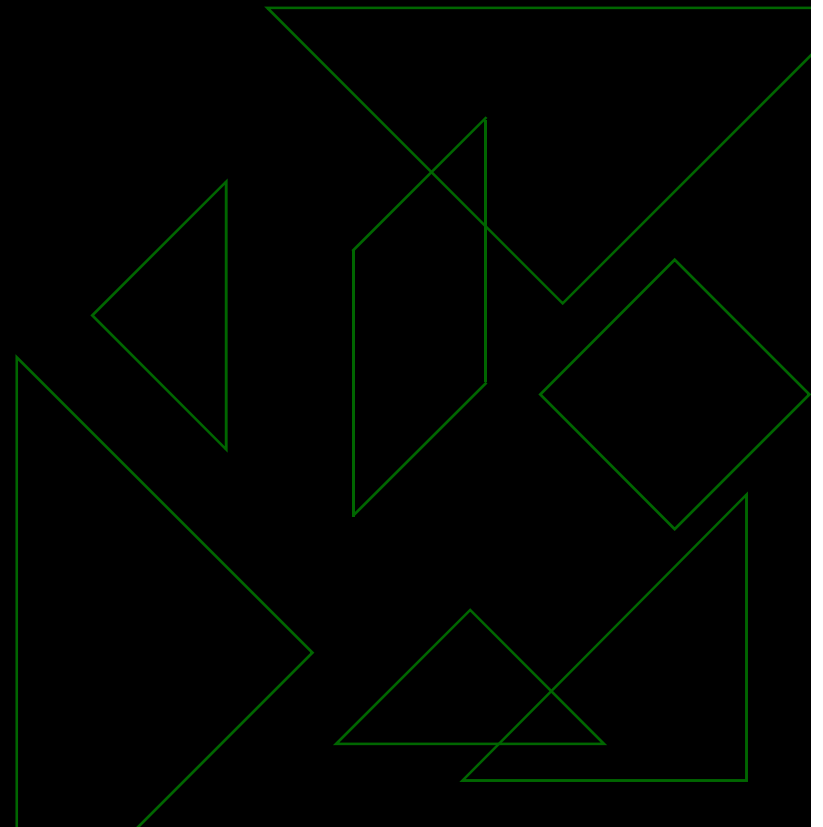


Pipeline de visualização 2D



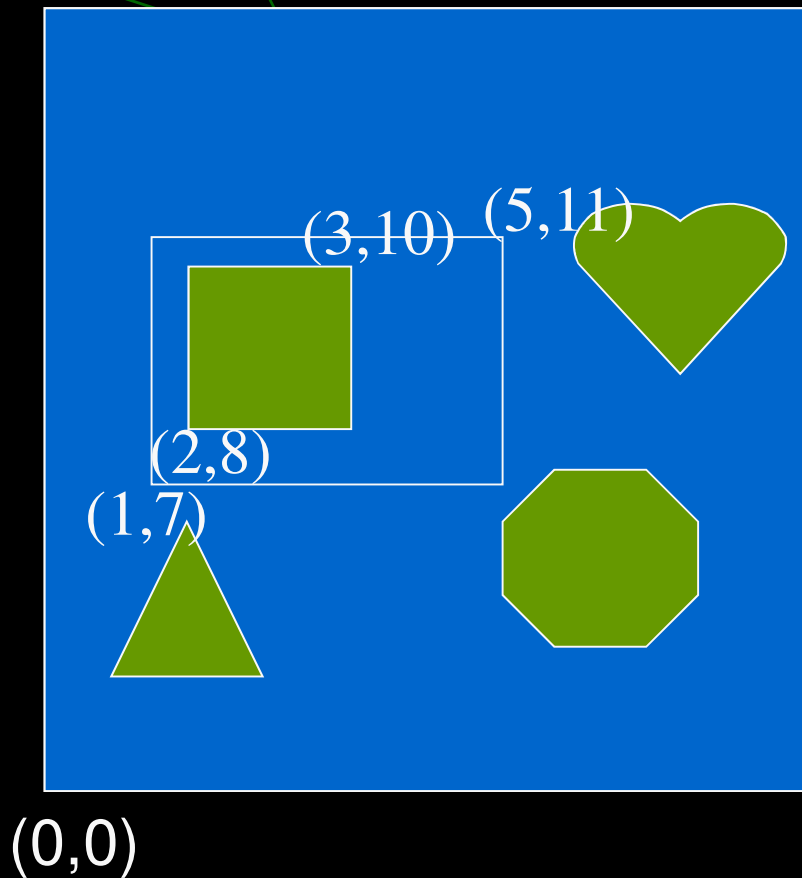
Pipeline 2D

- 
- ◆ SRO
 - ◆ SRU
 - ◆ SRW
(recorte 2D)
 - ◆ SRV
 - ◆ SRD



Pipeline de visualização 2D

SRU 2D



$\text{Obj1}_{vi} \text{SRU} = (2,8)$

$\text{Obj1}_{vf} \text{SRU} = (3,10)$

$\text{Window}_{vi} \text{SRU} = (1,7)$

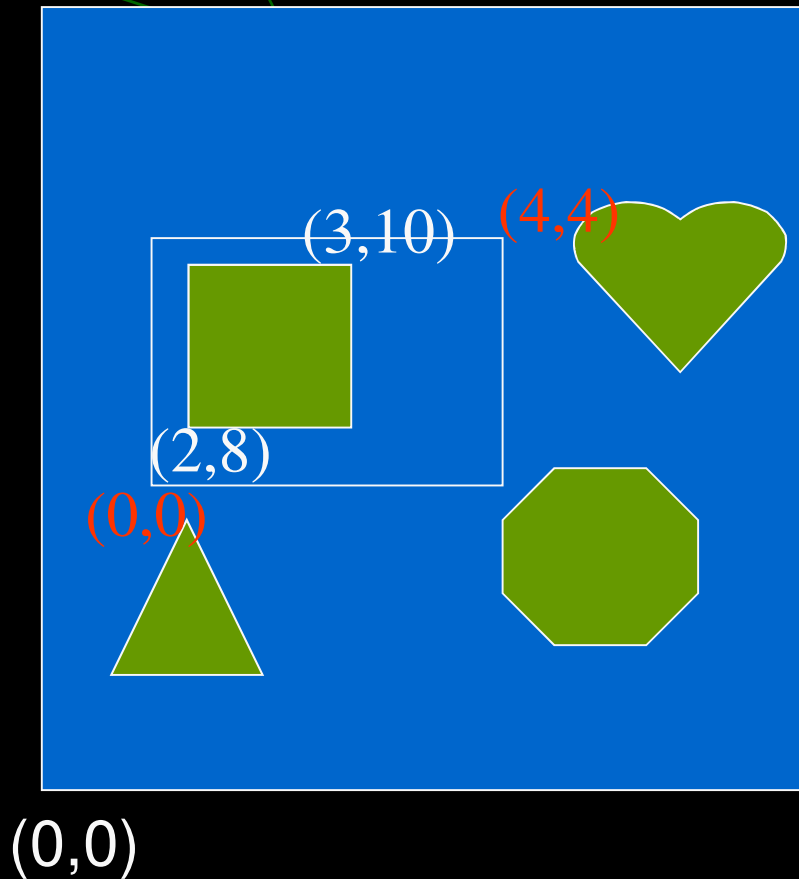
$\text{Window}_{vf} \text{SRU} = (5,11)$

$\text{Obj1}_{vi} \text{Window} = ?$

$\text{Obj1}_{vf} \text{Window} = ?$

Pipeline de visualização 2D

SRU 2D



$$\text{Window}_{vi} \text{ SRU} = (1,7)$$

$$\text{Window}_{vf} \text{ SRU} = (5,11)$$

$$T_{\text{SRU-SRW}} = 0 - \text{Window}_{vi} \text{ SRU}$$

$$T_{\text{SRU-SRW}} x = -1$$

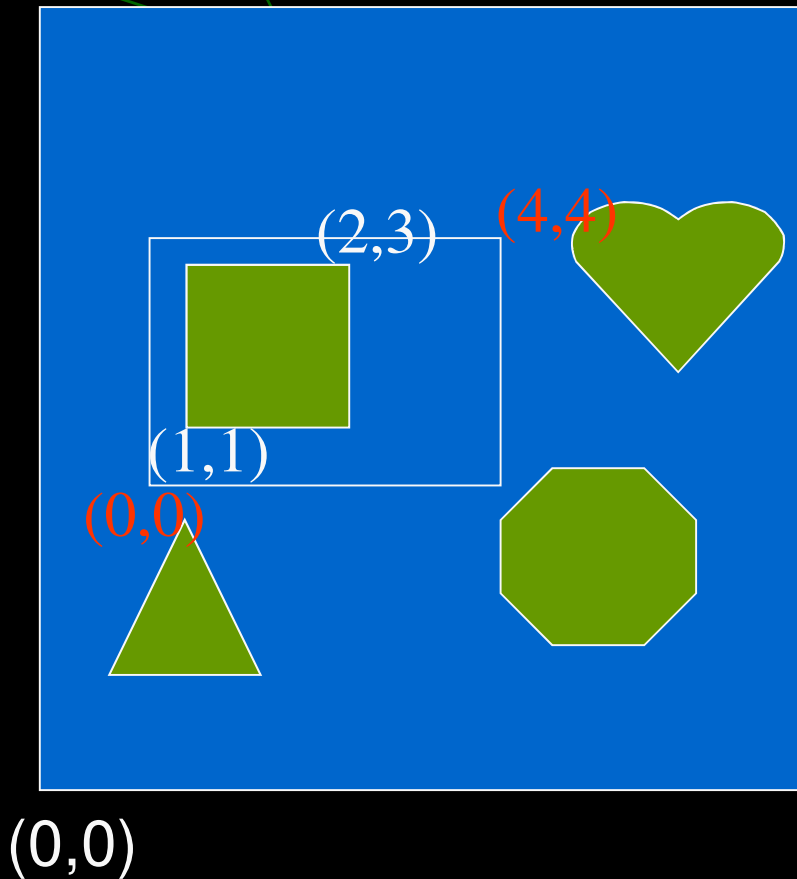
$$T_{\text{SRU-SRW}} y = -7$$

$$\text{Window}_{vi} \text{ SRW} = (0,0)$$

$$\text{Window}_{vf} \text{ SRW} = (4,4)$$

Pipeline de visualização 2D

SRU 2D



$$\text{Window}_{vi} \text{ SRU} = (1,7)$$

$$\text{Window}_{vf} \text{ SRU} = (5,11)$$

$$T_{\text{SRU-SRW}} = 0 - \text{Window}_{vi} \text{ SRU}$$

$$T_{\text{SRU-SRW}} x = -1$$

$$T_{\text{SRU-SRW}} y = -7$$

$$\text{Window}_{vi} \text{ SRW} = (0,0)$$

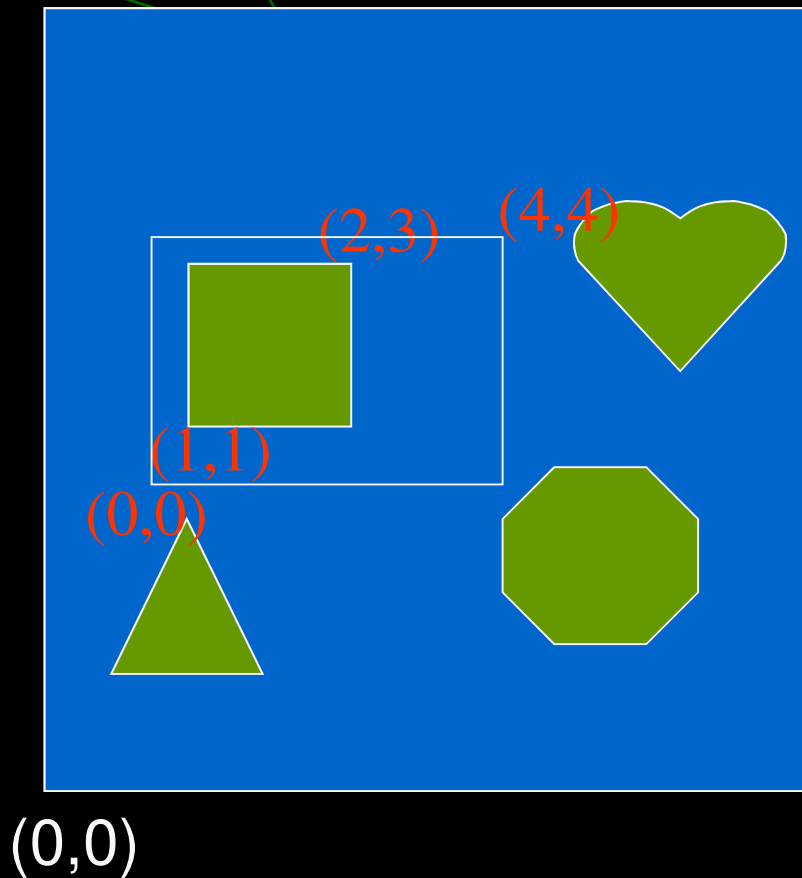
$$\text{Window}_{vf} \text{ SRW} = (4,4)$$

$$\text{Obj1}_{vi} \text{ Window} = (1,1)$$

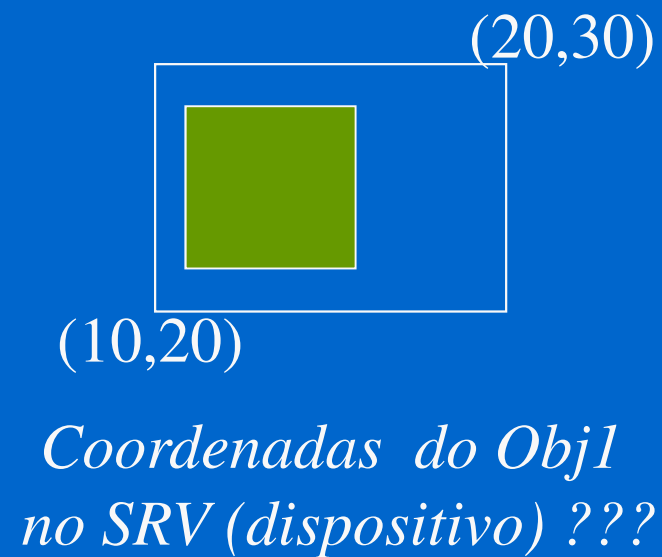
$$\text{Obj1}_{vf} \text{ Window} = (2,3)$$

Pipeline de visualização 2D

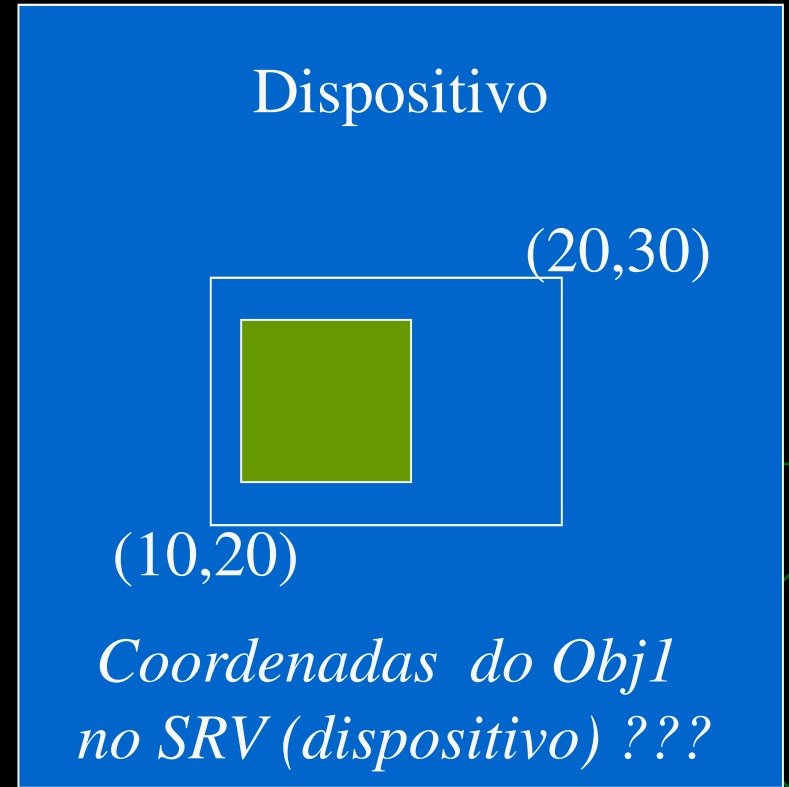
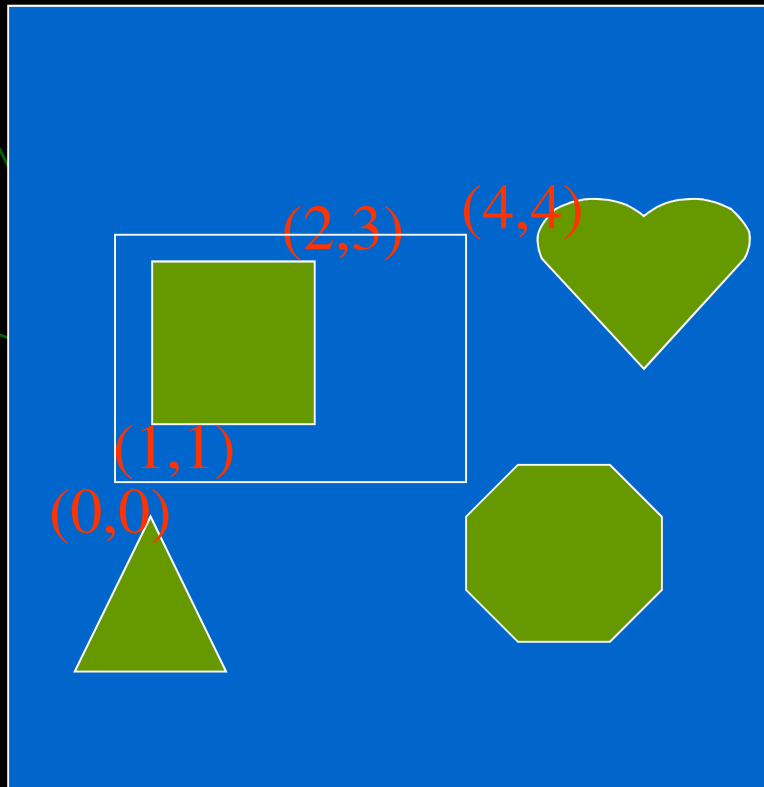
SRU 2D



Dispositivo



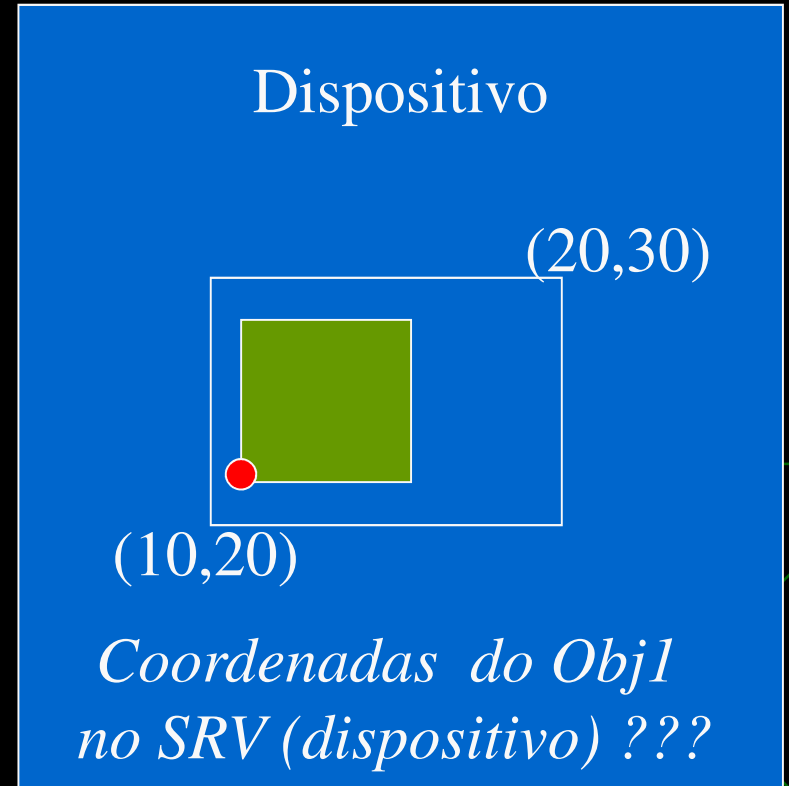
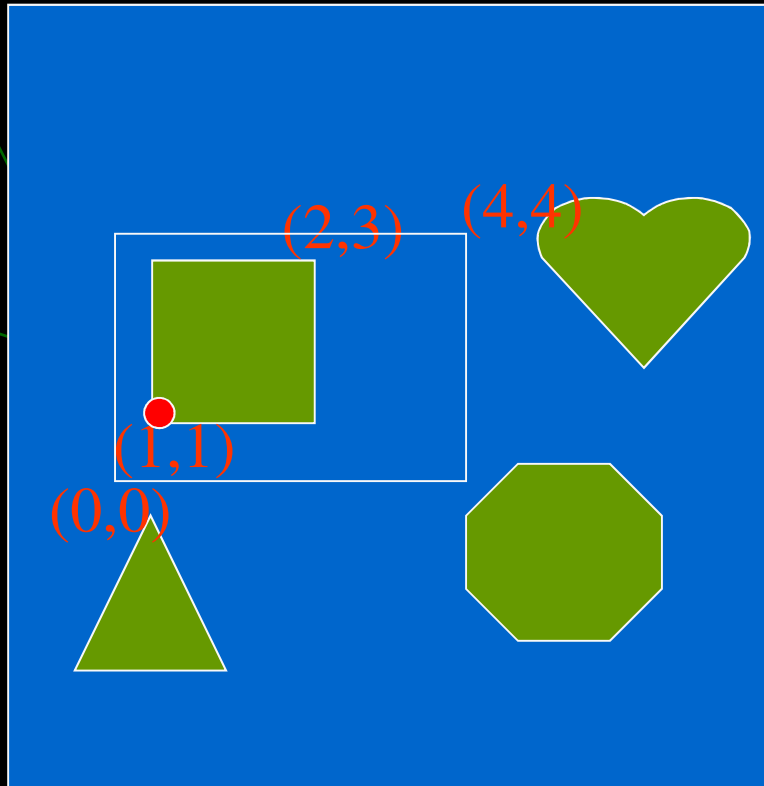
Transformação SRU->SRD



$$x_v = xi_v + \frac{(x_w - xi_w)}{(xf_w - xi_w)} (xf_v - xi_v)$$

>>>Proporcionalidade

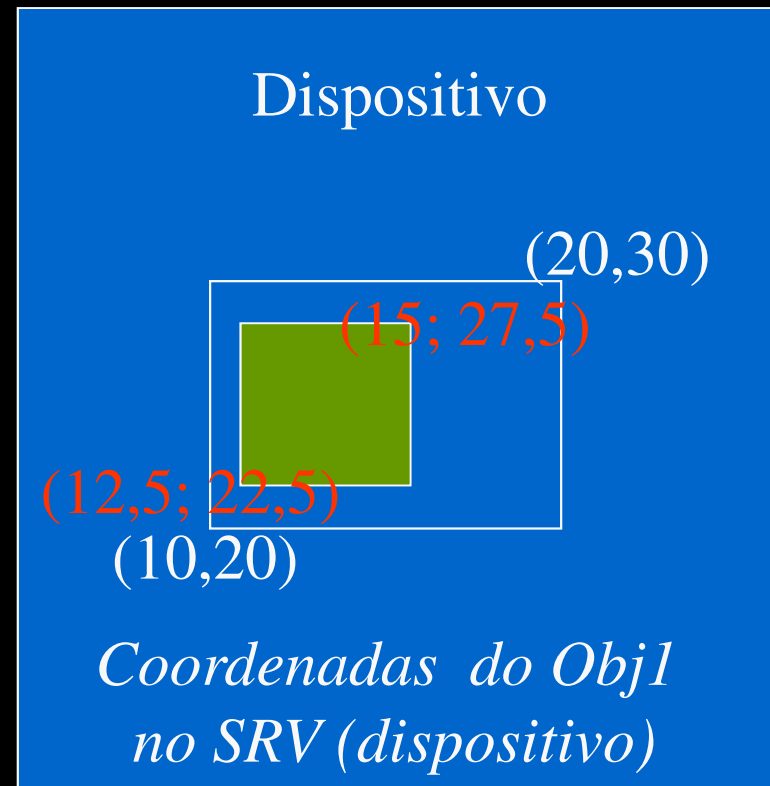
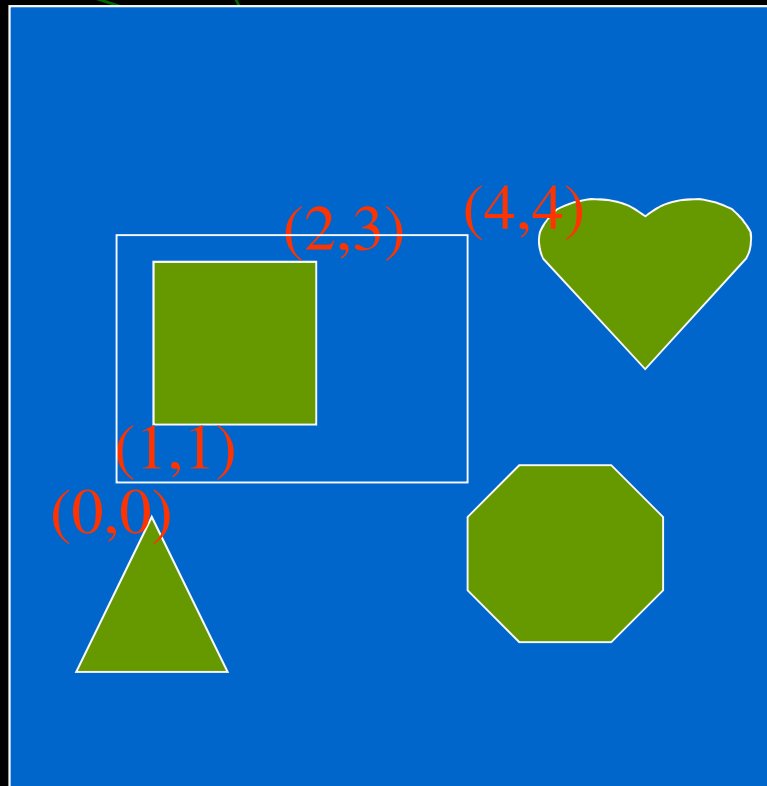
Você calcula....



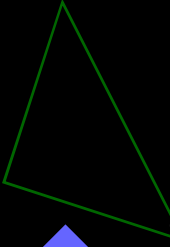
$$x_v = xi_v + \frac{(x_w - xi_w)}{(xf_w - xi_w)} (xf_v - xi_v)$$

>>>Proporcionalidade

Mapeamento:



SR's 2D



◆ SRO

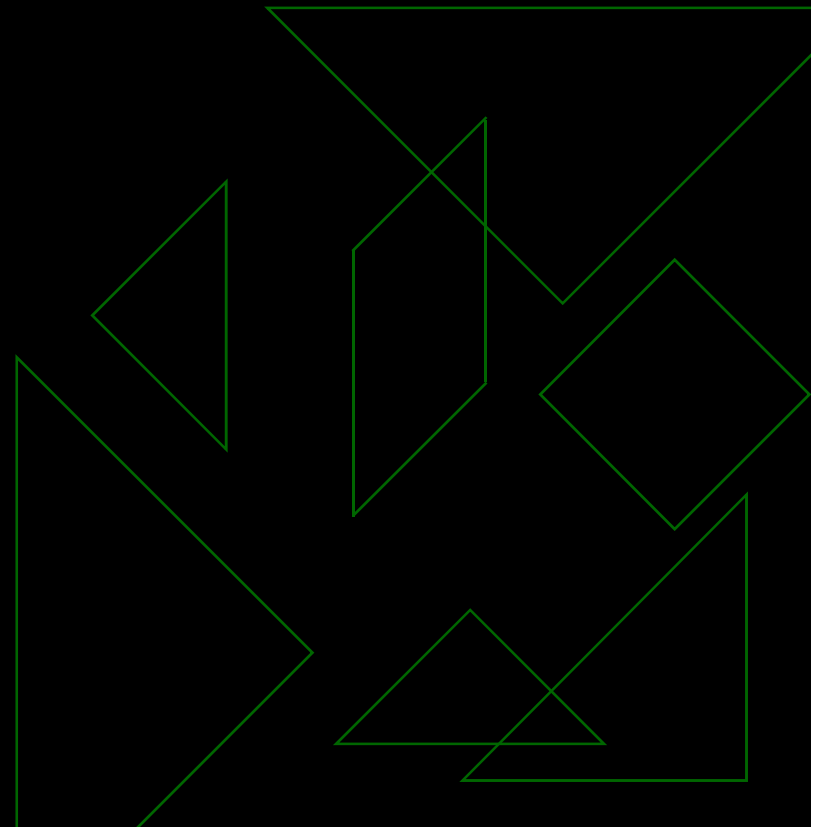
◆ SRU

◆ SRW

(recorte 2D)

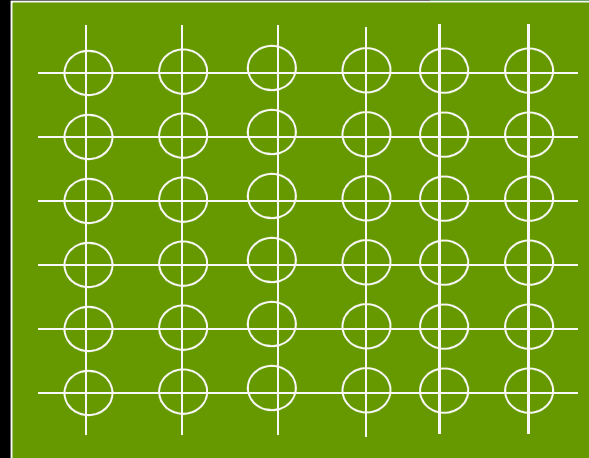
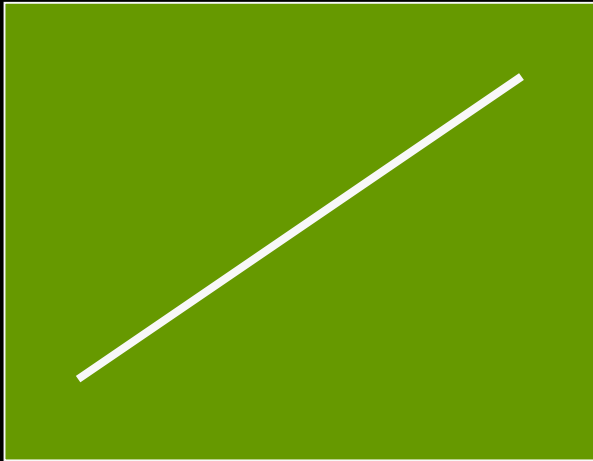
◆ SRV

◆ SRD



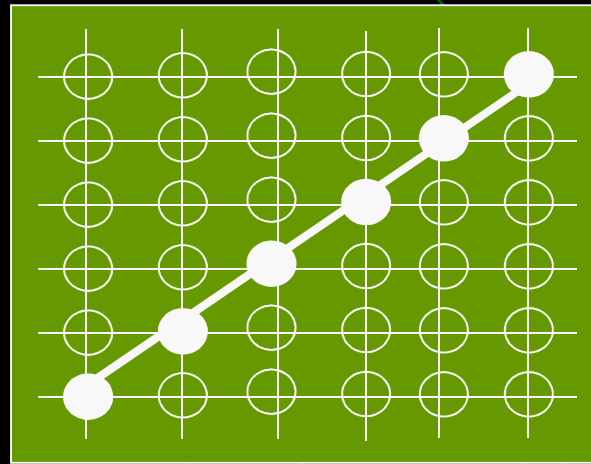
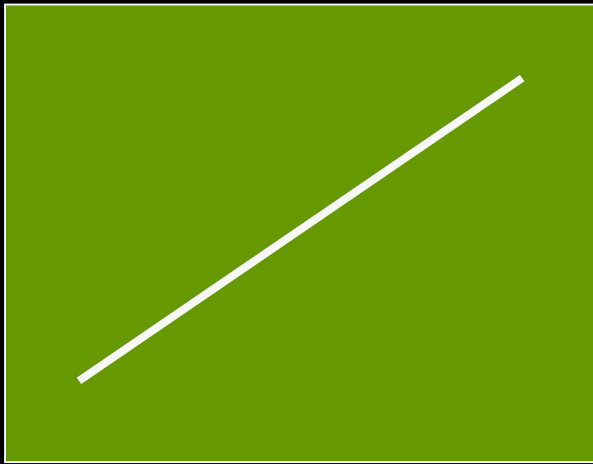
Algoritmos de rasterização

◆ Algoritmos de varredura



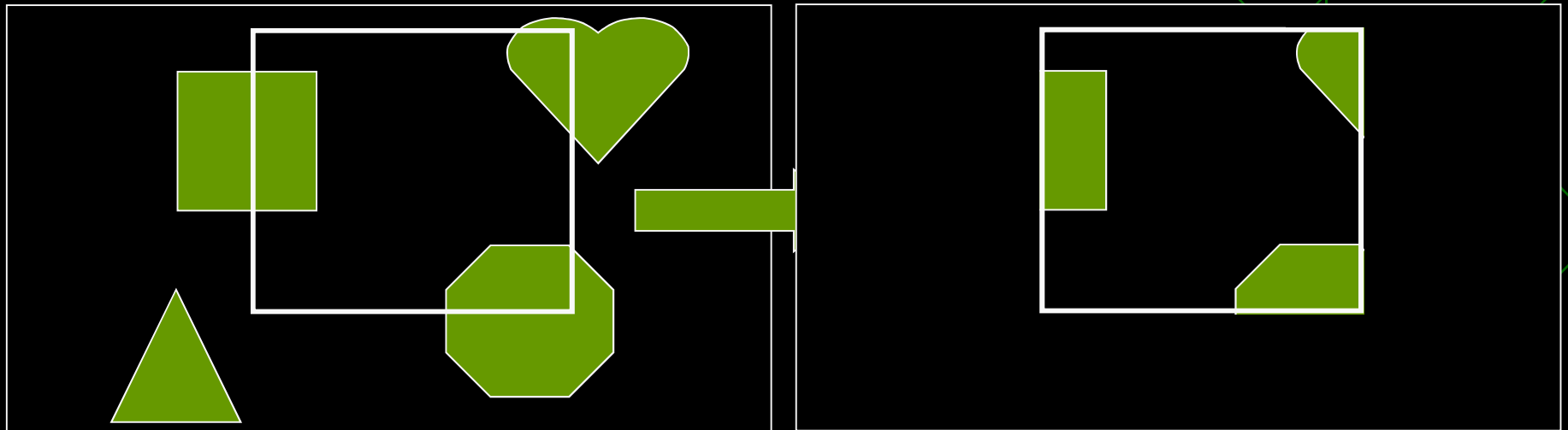
Algoritmos de rasterização

◆ Algoritmos de varredura



Algoritmos de rasterização

- ◆ Algoritmos de varredura
- ◆ Algoritmos de recorte

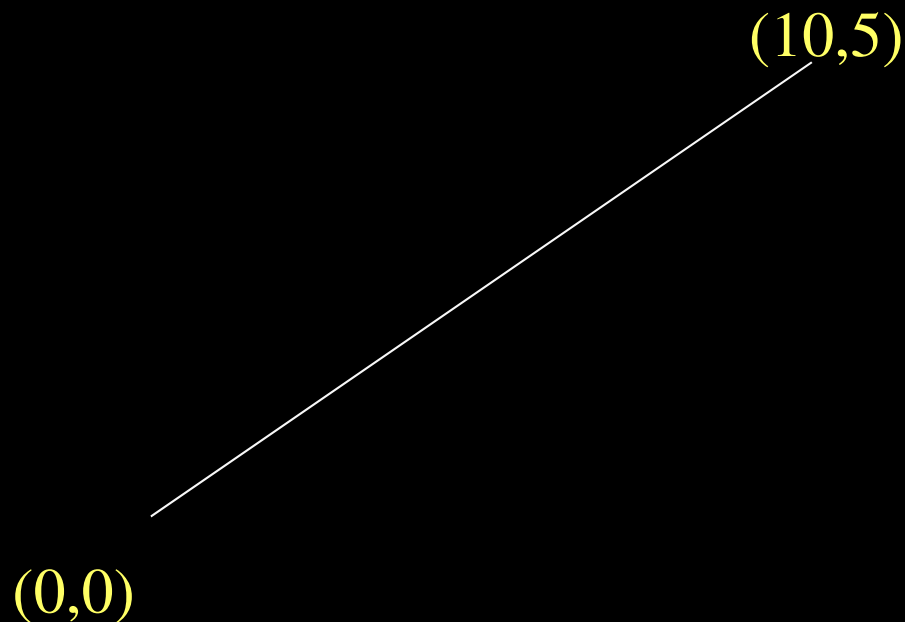


Algoritmos de varredura

- ◆ Resolução por Método incremental
 - Maneira para resolver equações diferenciais através de métodos numéricos
 - Sucessivas operações de incremento baseado no ponto atual

Algoritmos de varredura

◆ Método incremental

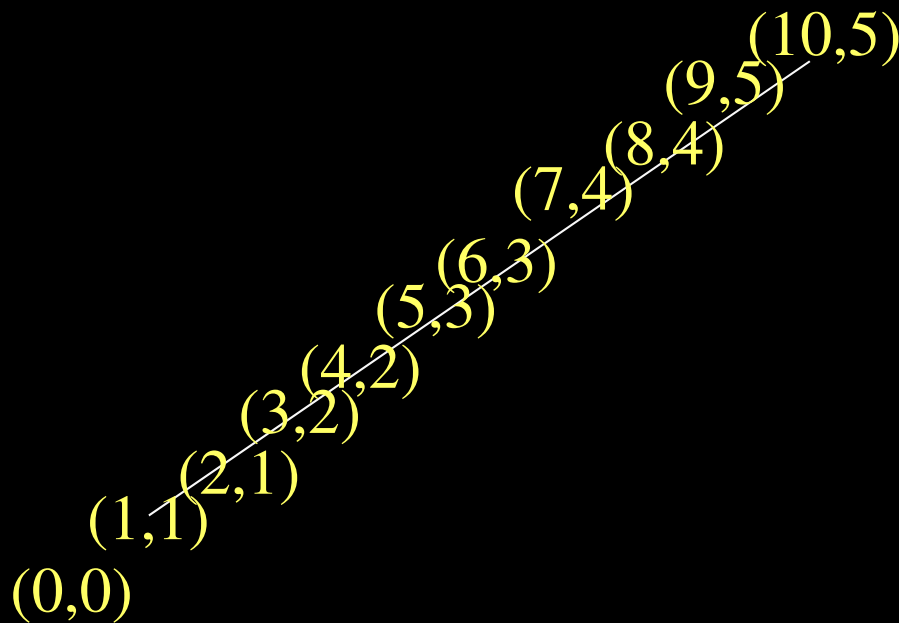


```
{ Dx=xf-xi;  
  Dy=yf-yi;  
  M=Dy/Dx;  
  y=yi;  
  For (x=xi;x<=xf;x++)  
  {  
    Write(x,int(floor(y+0,5), value);  
    y+=M;  
  }  
}
```

Coordenadas dos pixel
escritos?

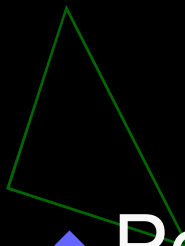
Algoritmos de varredura

◆ Método incremental

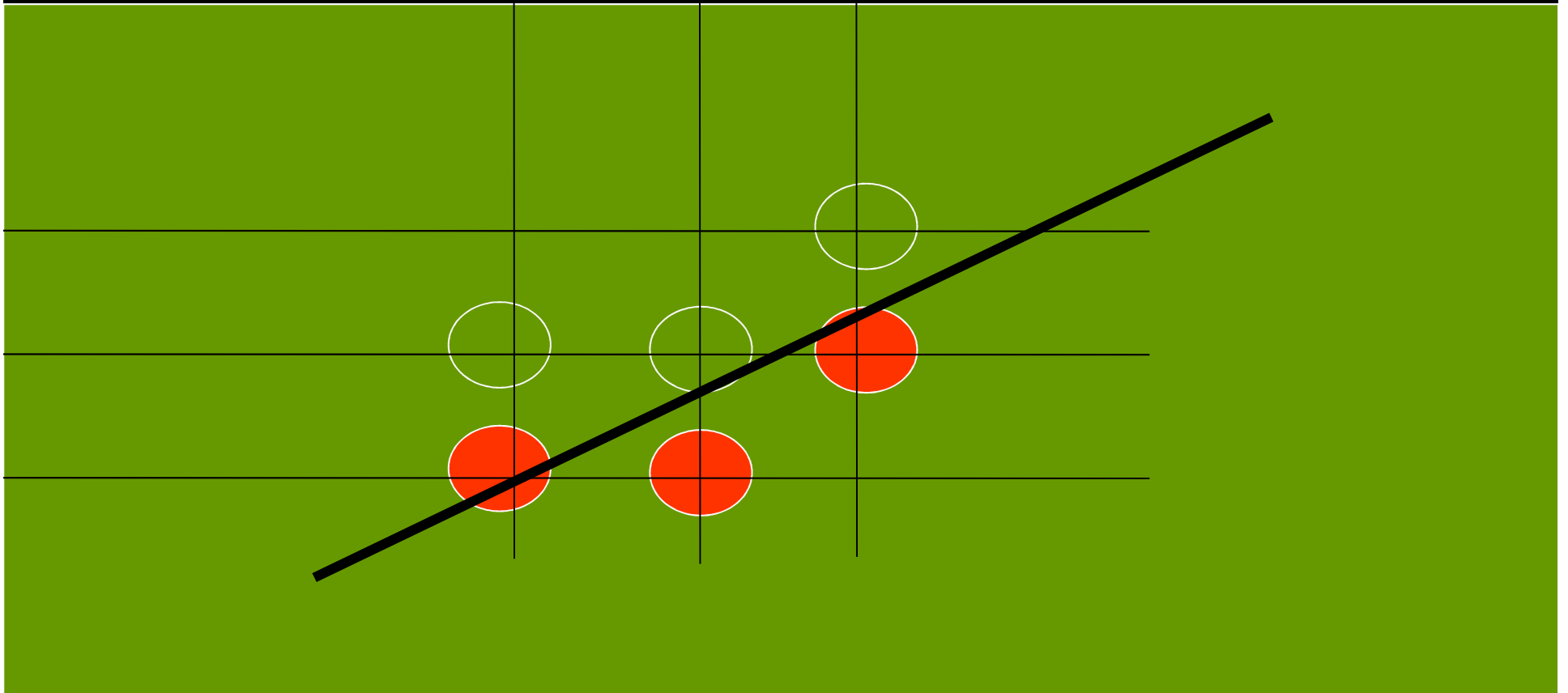


```
{ Dx=xf-xi;  
  Dy=yf-yi;  
  M=Dy/Dx;  
  y=yi;  
  For (x=xi;x<=xf;x++)  
  {  
    Write(x,int(floor(y+0,5), value);  
    y+=M;  
  }  
}
```

Algoritmos de varredura



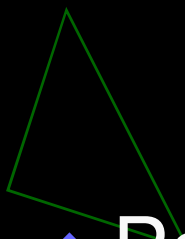
◆ Porque usar floor?



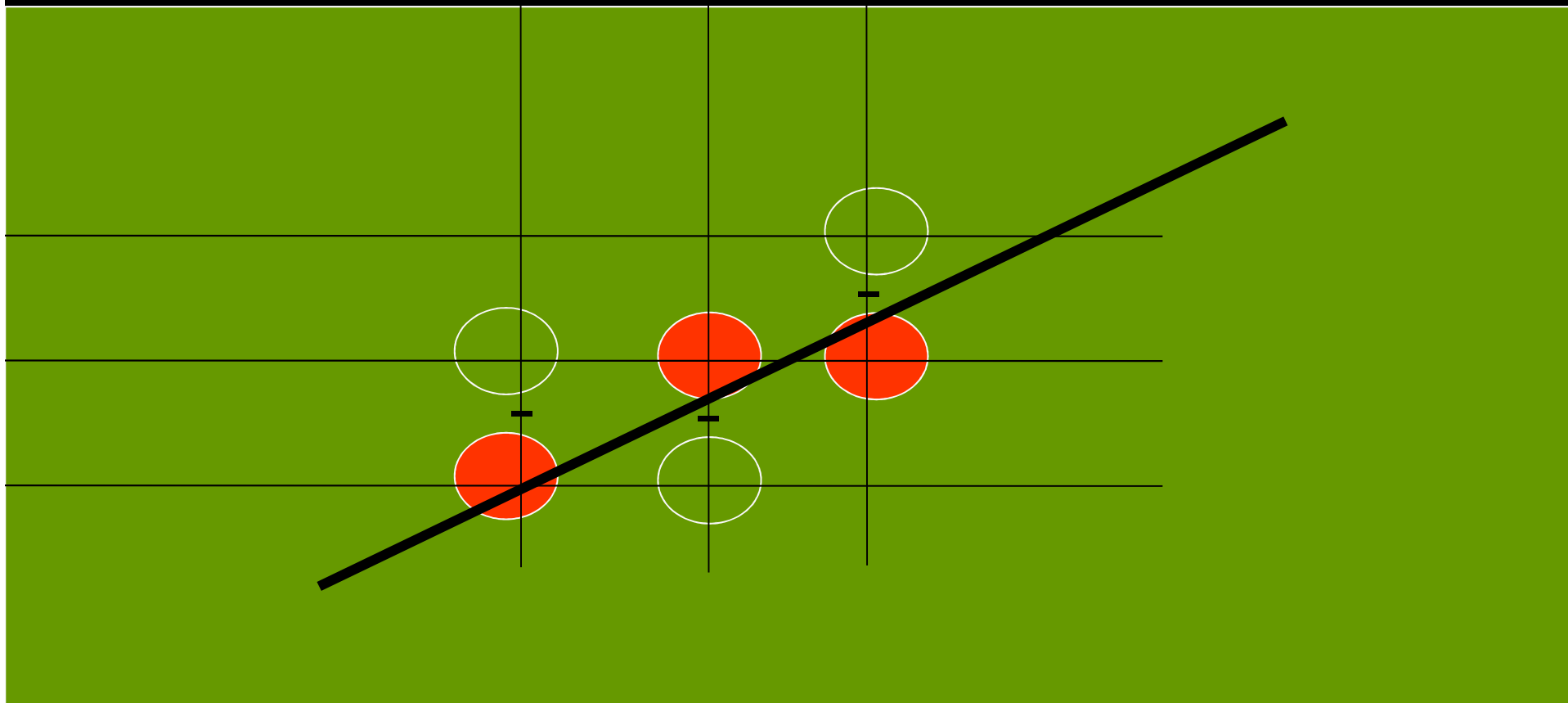
Algoritmos de varredura

- ◆ Bresenham – Algoritmo do ponto médio
 - Atrativo porque usa somente operações aritméticas (não usa round ou floor)
 - É incremental

Algoritmos de varredura



◆ Porque ponto médio?



Algoritmos de varredura

◆ Ponto médio

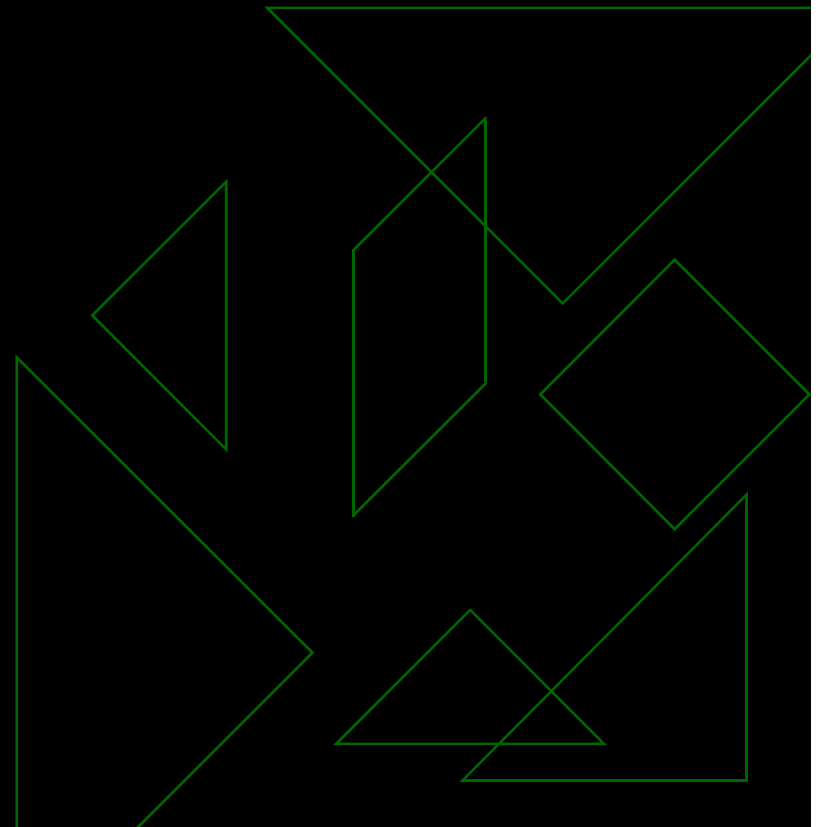
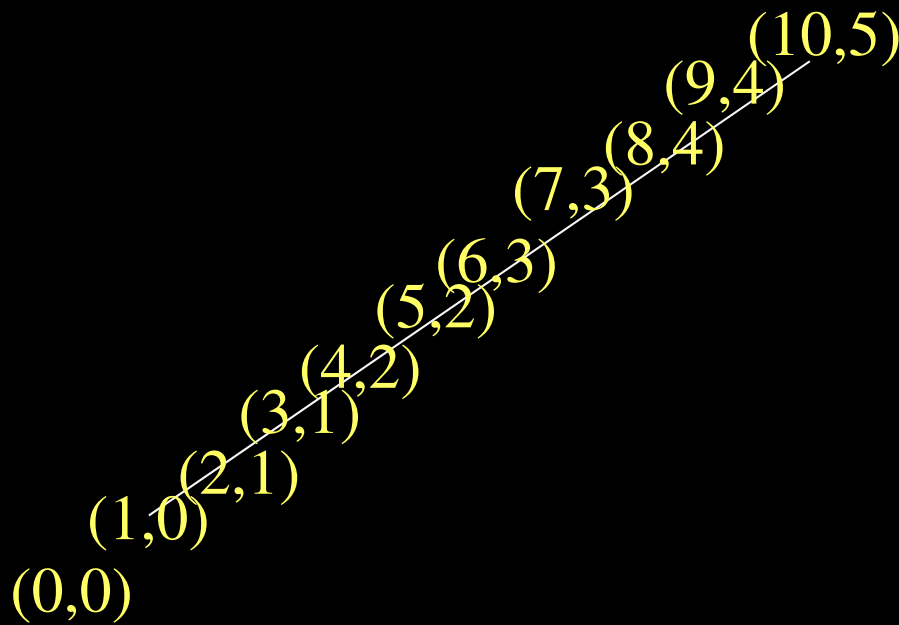
```
{ dx= x1 - x0;  
  dy = y1 - y0;  
  d = dy * 2 - dx;  
  incrE = dy * 2;  
  
  incrNE = (dy-dx) * 2;  
  x = x0;  
  y = y0;  
  writepixel(x,y,value);
```

```
while (x<x1) {  
  if (d<=0){  
    d+=incrE; x++;}  
  else {  
    d+=incrNE; x++;  
    y++;}  
  writepixel(x,y,value);  
}
```

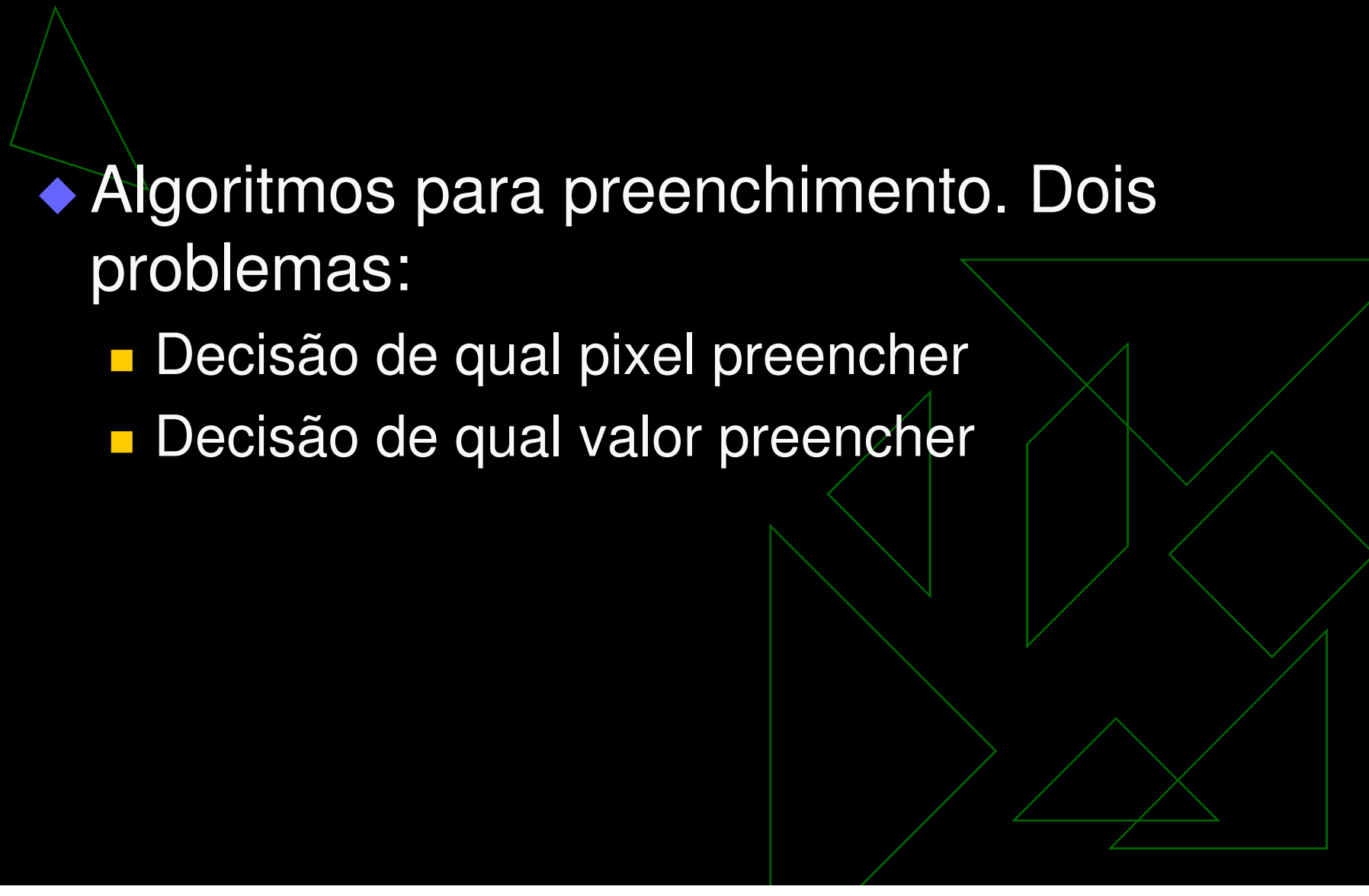
** Descubra os pixels “pintados” da reta descrita pelos vértices (0,0) e (10,5)

Algoritmos de varredura

◆ Algoritmo do ponto médio

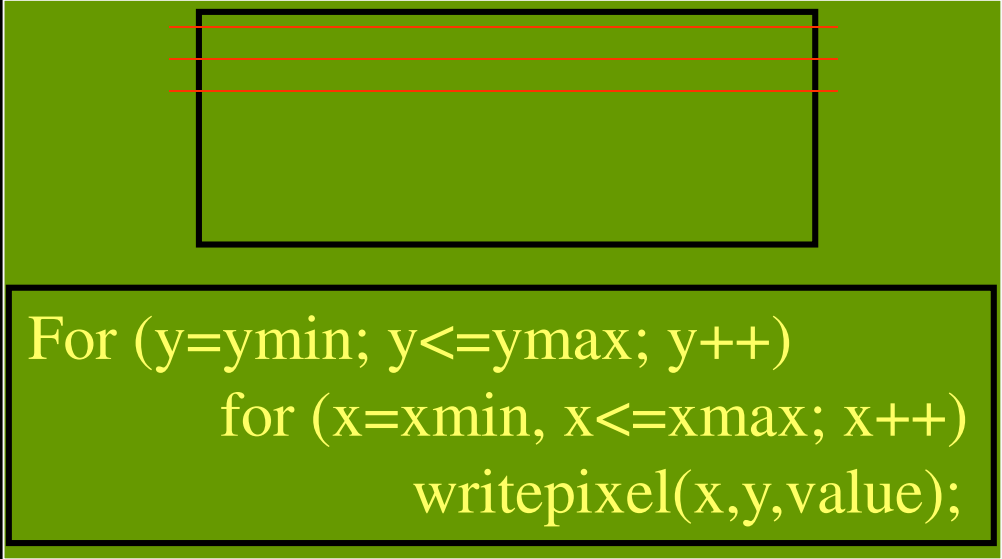


Algoritmos de varredura

- 
- The slide features several green-outlined geometric shapes as a background decoration. These include a triangle in the upper left, a large triangle in the lower right, and several overlapping rectangles and smaller triangles scattered across the right side of the slide.
- ◆ Algoritmos para preenchimento. Dois problemas:
 - Decisão de qual pixel preencher
 - Decisão de qual valor preencher

Algoritmos de varredura

- ◆ Algoritmos para preenchimento:
Retângulo

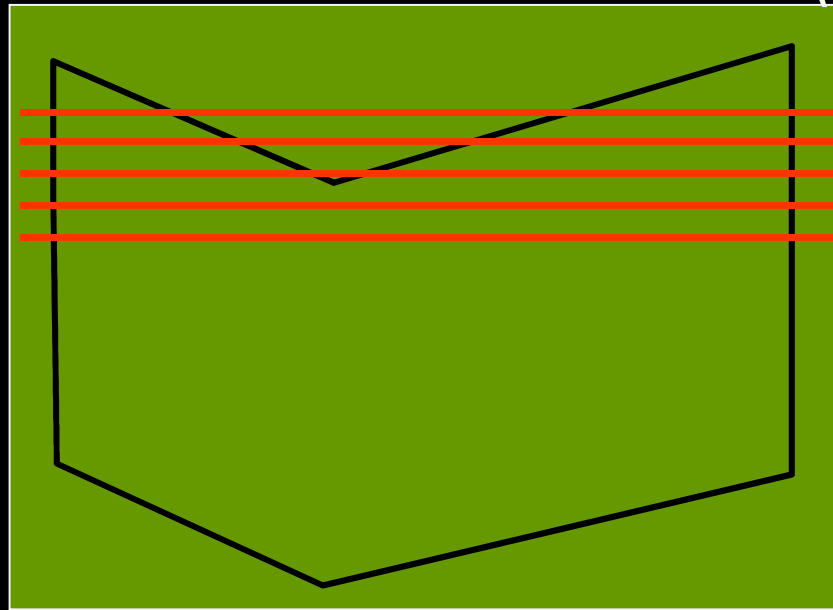


The diagram illustrates the scanline algorithm for filling a rectangle. It shows a large green rectangle with a smaller black-outlined rectangle inside it. Four horizontal red lines are drawn across the inner rectangle, representing the scanlines. Below the diagram, a code block contains the algorithm's logic.

```
For (y=ymin; y<=ymax; y++)  
    for (x=xmin, x<=xmax; x++)  
        writepixel(x,y,value);
```

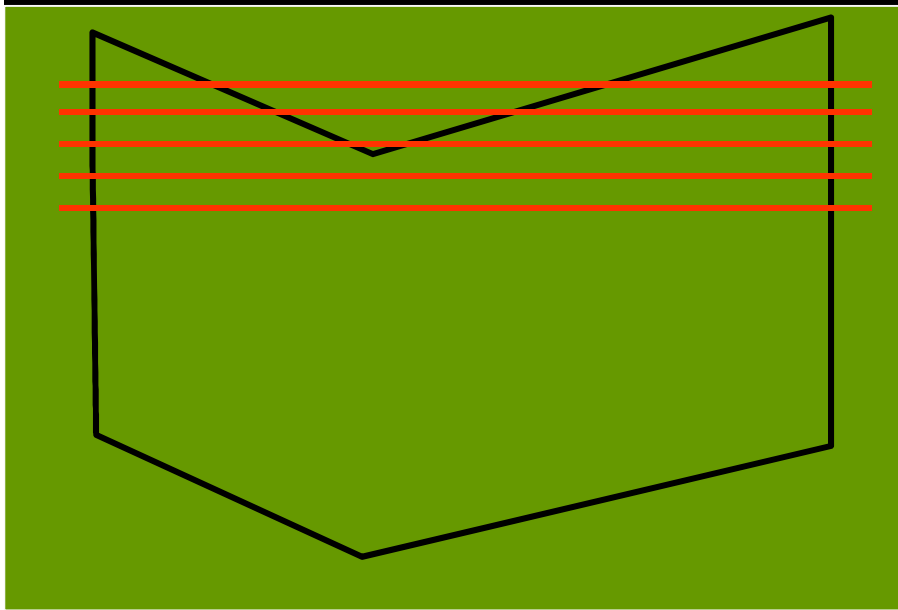
Algoritmos de varredura

- ◆ Algoritmos para preenchimento:
Polígonos convexos ou não (*scanline*)



Algoritmos de varredura

- ◆ Algoritmos para preenchimento:
Polígonos convexos ou não



Passos:

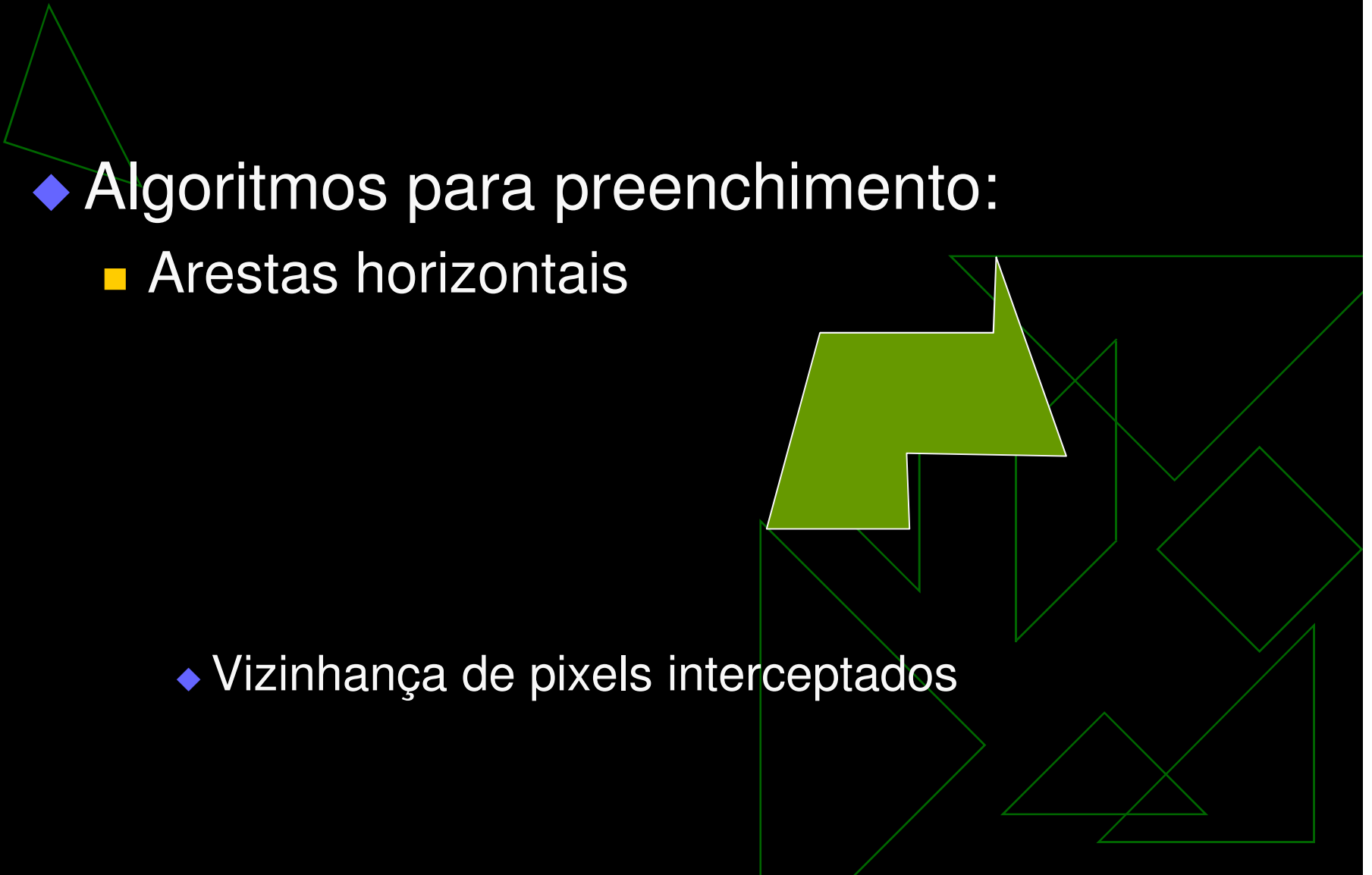
- 1) Encontrar as intersecções da scan line com as arestas do polígono
- 2) Ordenar as intersecções
- 3) Preencher os pixels entre 2 intersecções (regra de paridade que inicia em par, muda quando encontra uma intersecção e escreve quando é ímpar)

Algoritmos de varredura

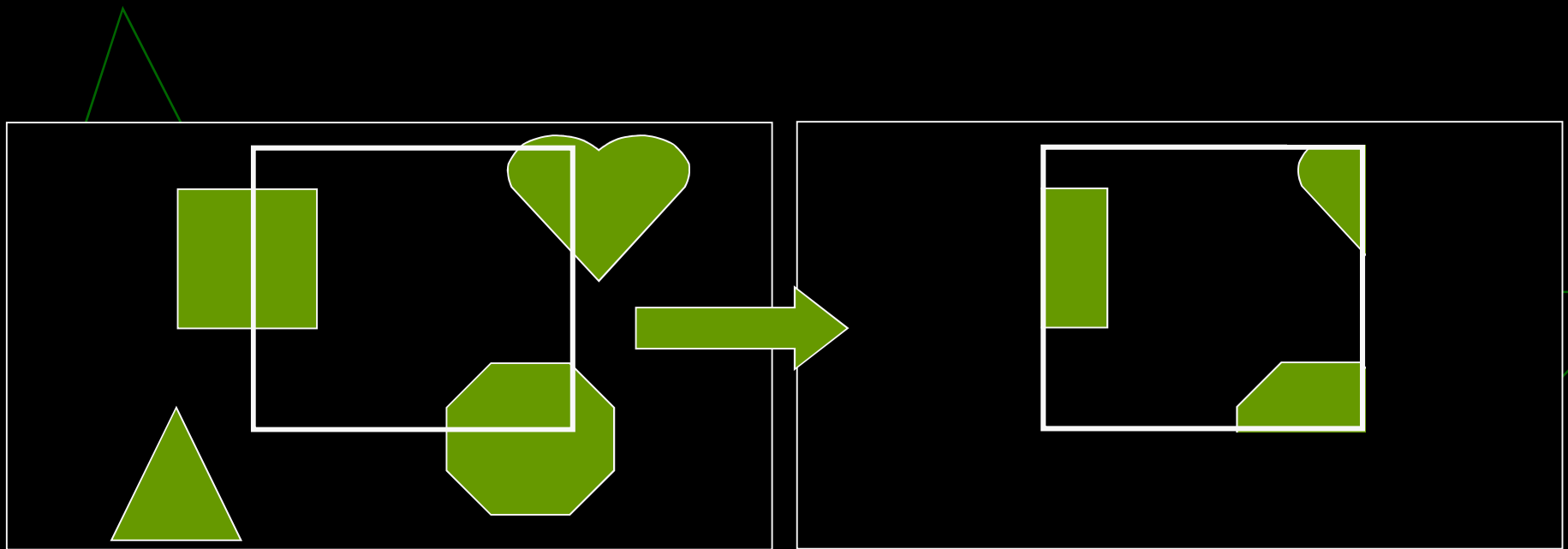
- ◆ Algoritmos para preenchimento:

- Arestas horizontais

- ◆ Vizinhança de pixels interceptados



Algoritmos de recorte

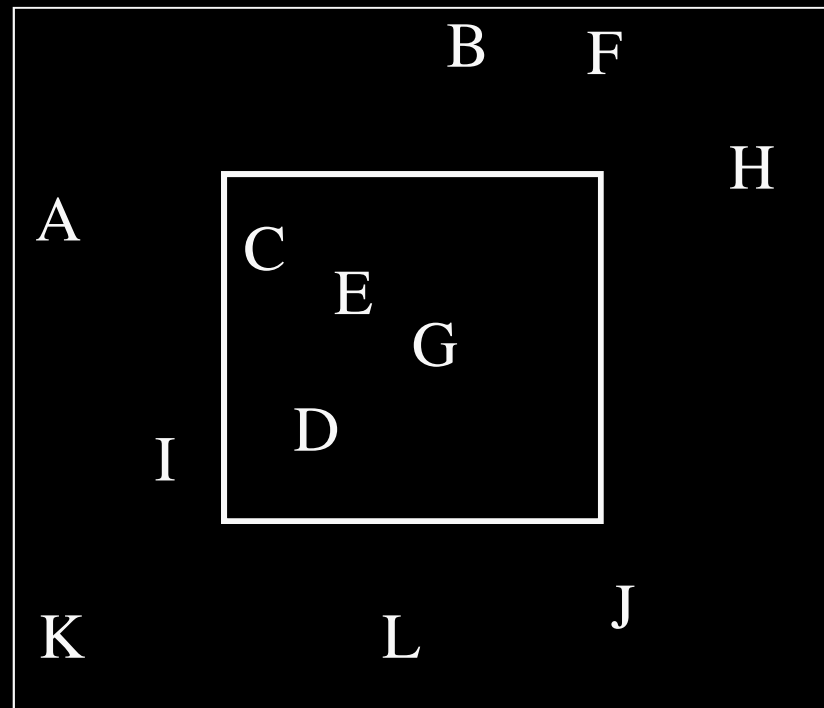


Objetivo: otimização das estruturas a serem desenhadas no dispositivo

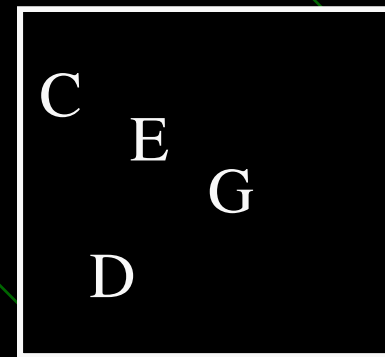
Trata o recorte contra as linhas da janela de seleção (retângulo)

Algoritmos de recorte

>> Recorte de pontos contra retângulo

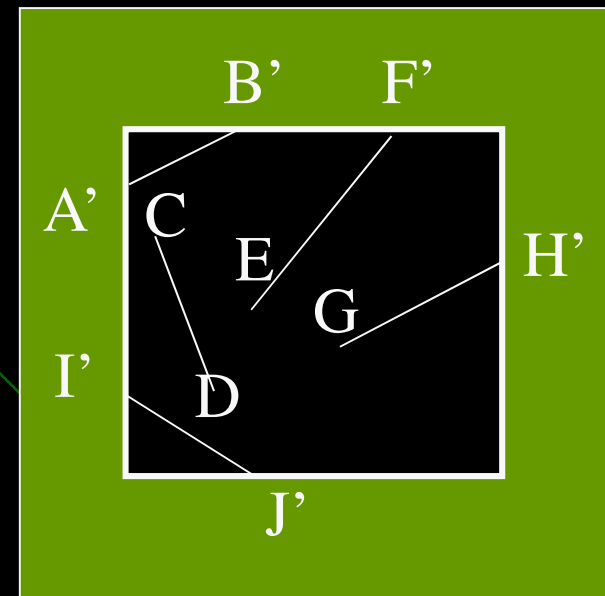
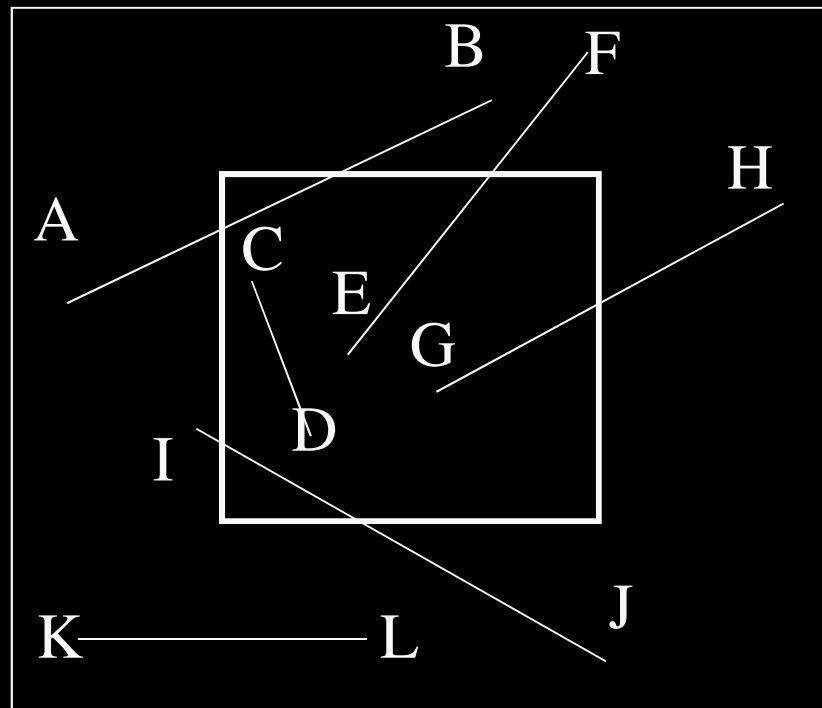


$$\begin{aligned} x_i &\leq x \leq x_f \\ y_i &\leq y \leq y_f \end{aligned}$$



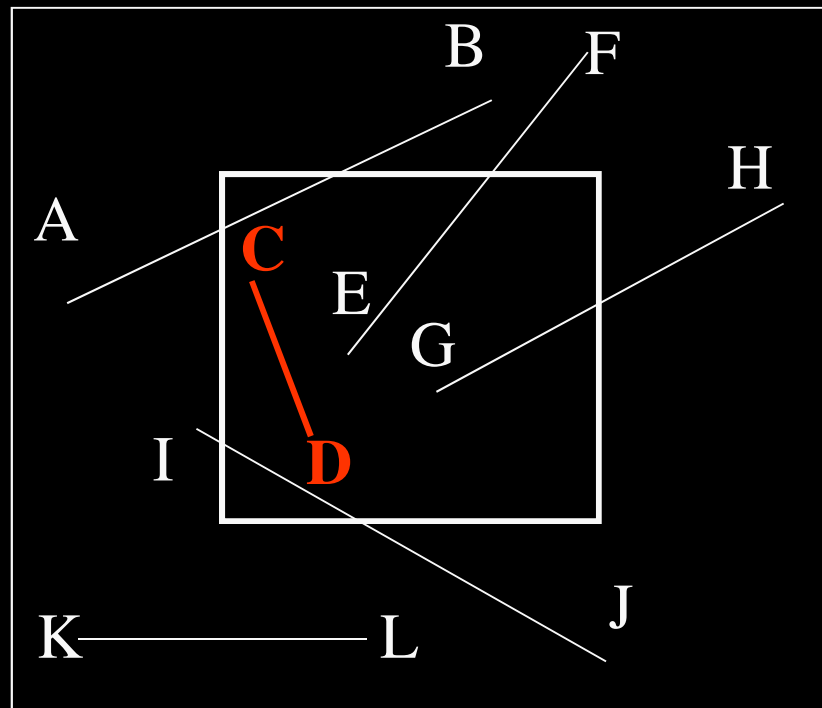
Algoritmos de recorte

>> Recorte de arestas (linhas) contra retângulo



Algoritmos de recorte

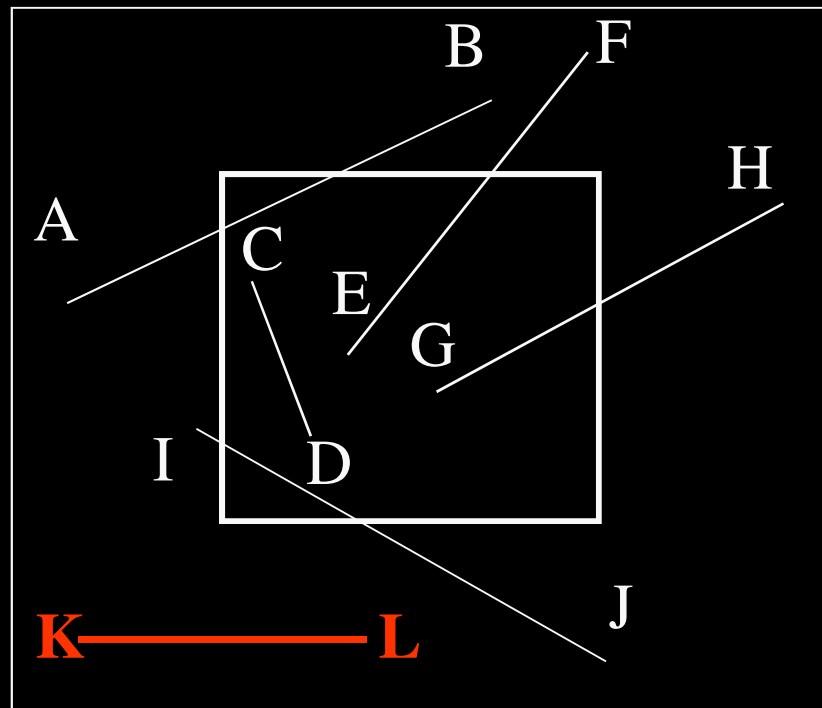
Recorte de linhas: Trivialmente aceito



- C e D estão dentro do retângulo de seleção

Algoritmos de recorte

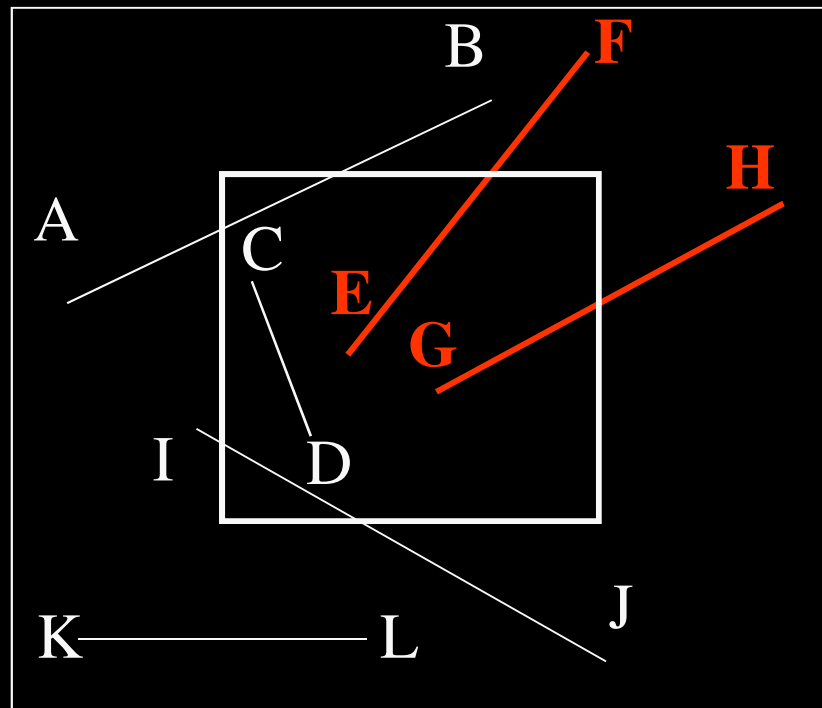
Recorte de linhas: Trivialmente recusado



- Os dois pontos estão fora do retângulo e tem $y < y_i$ ou $y > y_f$ e $x < x_i$ ou $x > x_f$

Algoritmos de recorte

Recorte de linhas: Cálculos de recorte

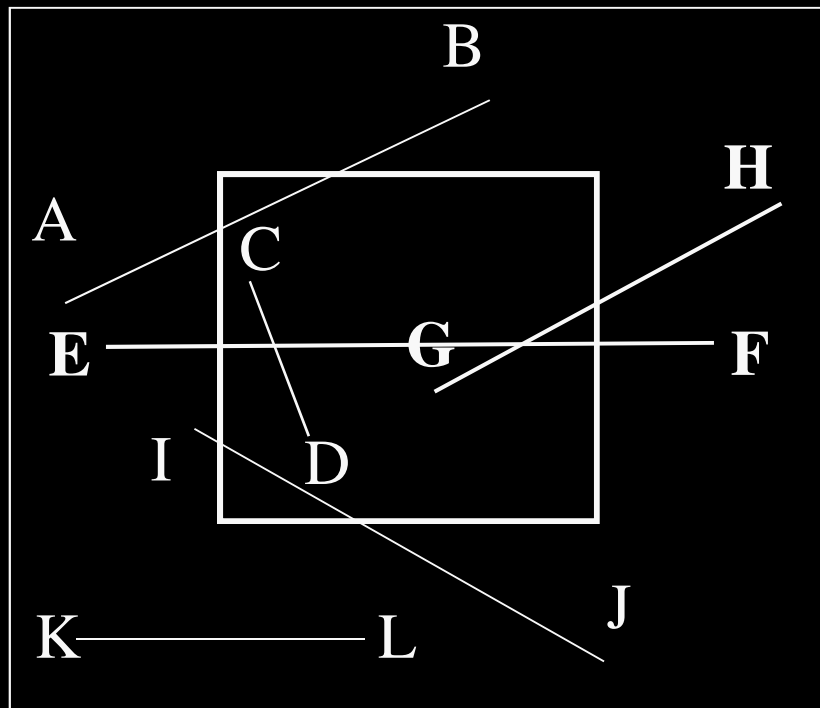


- Um dos pontos da linha está fora e outro está dentro do retângulo

- Linha deve ser recortada

Algoritmos de recorte

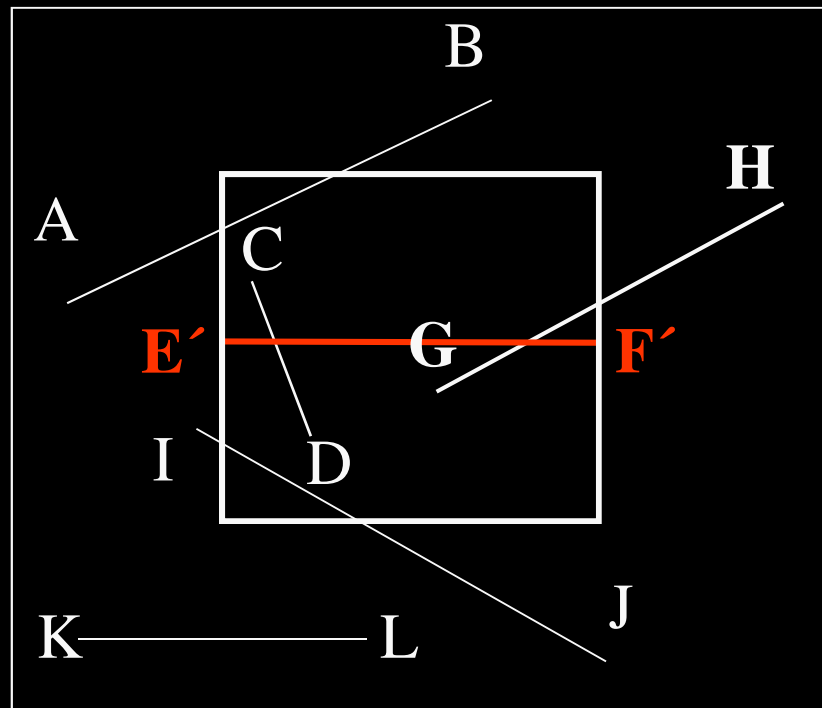
Recorte de linhas: Cálculos de recorte



- Os dois pontos estão fora
- Os dois pontos estão fora do retângulo e $!(y < y_i \text{ ou } y > y_f \text{ e } x < x_i \text{ ou } x > x_f)$
- Linha deve ser recortada

Algoritmos de recorte

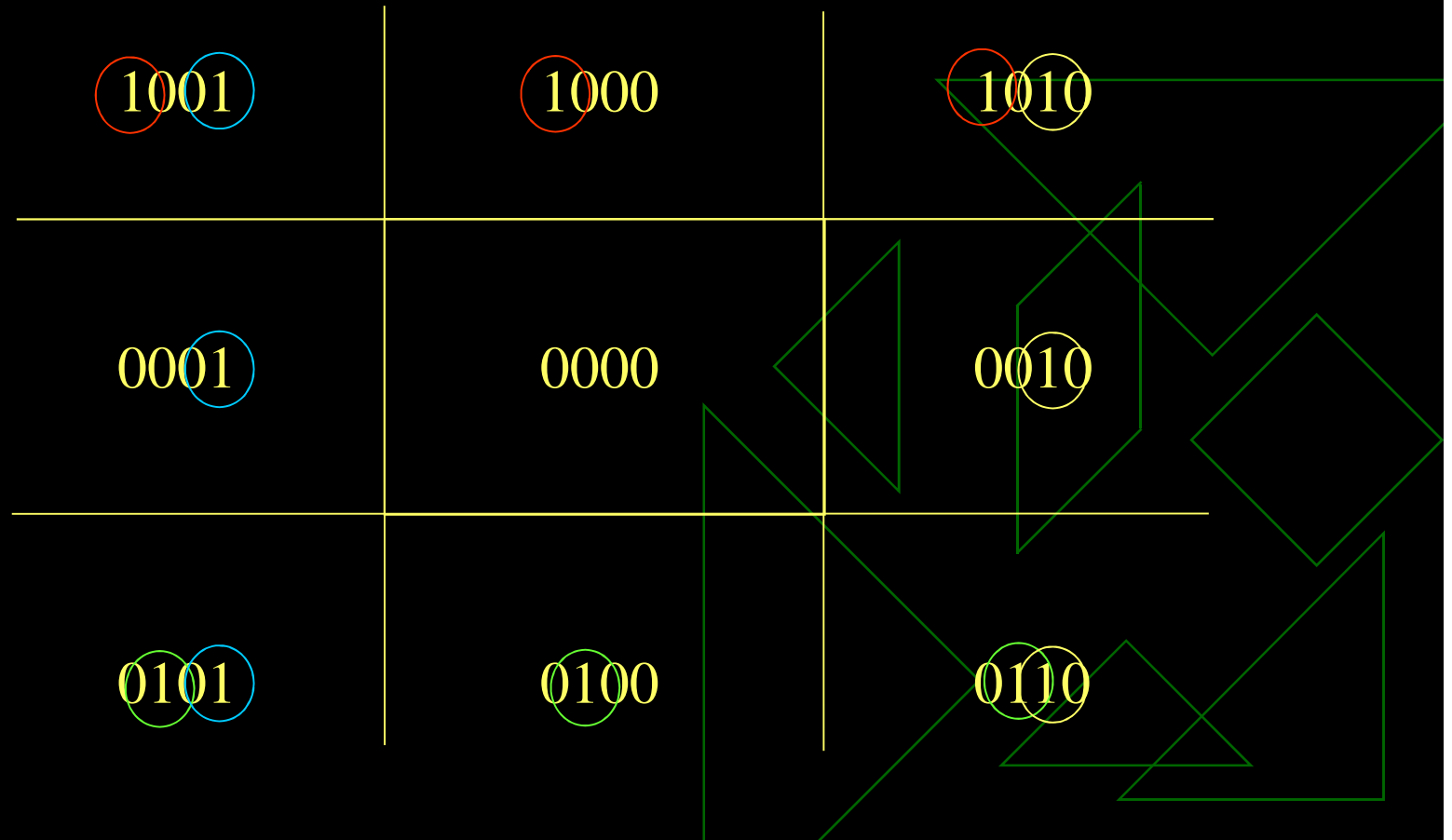
Recorte de linhas: Cálculos de recorte



- Os dois pontos estão fora
- Linha recortada
- Encontrar novos vértices

Algoritmos de recorte

Algoritmo de Cohen-Sutherland



Algoritmos de recorte

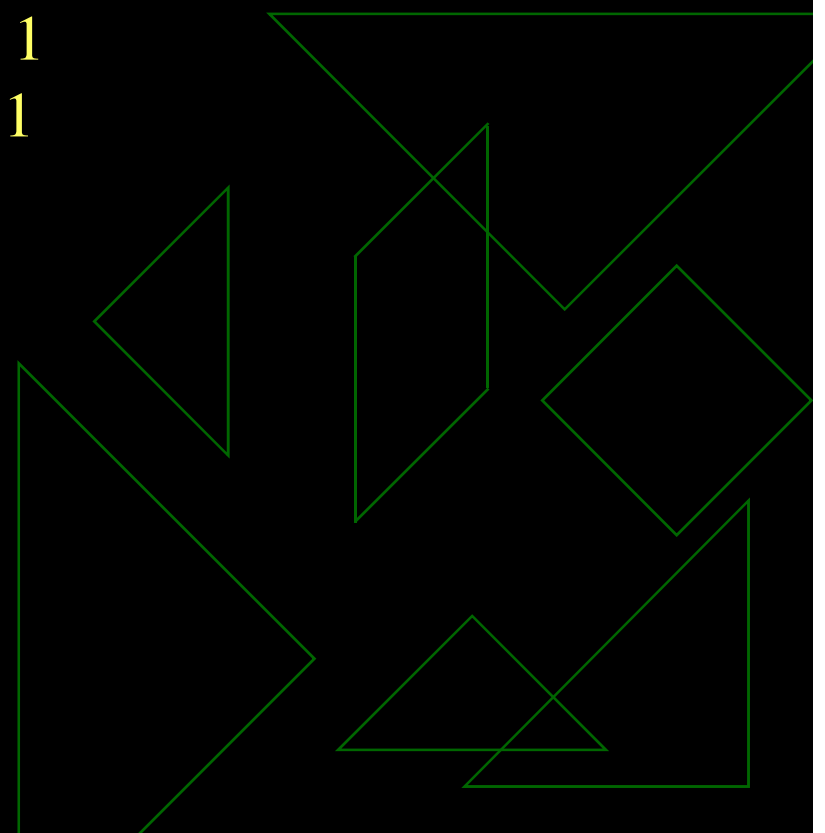
Algoritmo de Cohen-Sutherland

If $y > y_f$ → seta primeiro bit em 1

If $y < y_i$ → seta segundo bit em 1

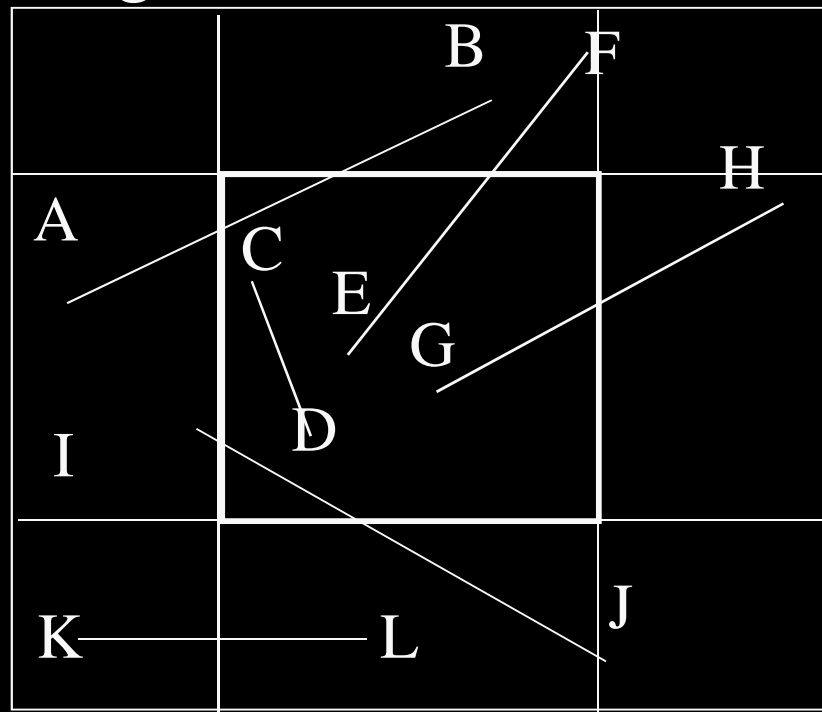
If $x > x_f$ → seta terceiro bit em 1

If $x < x_i$ → seta quarto bit em 1



Algoritmos de recorte

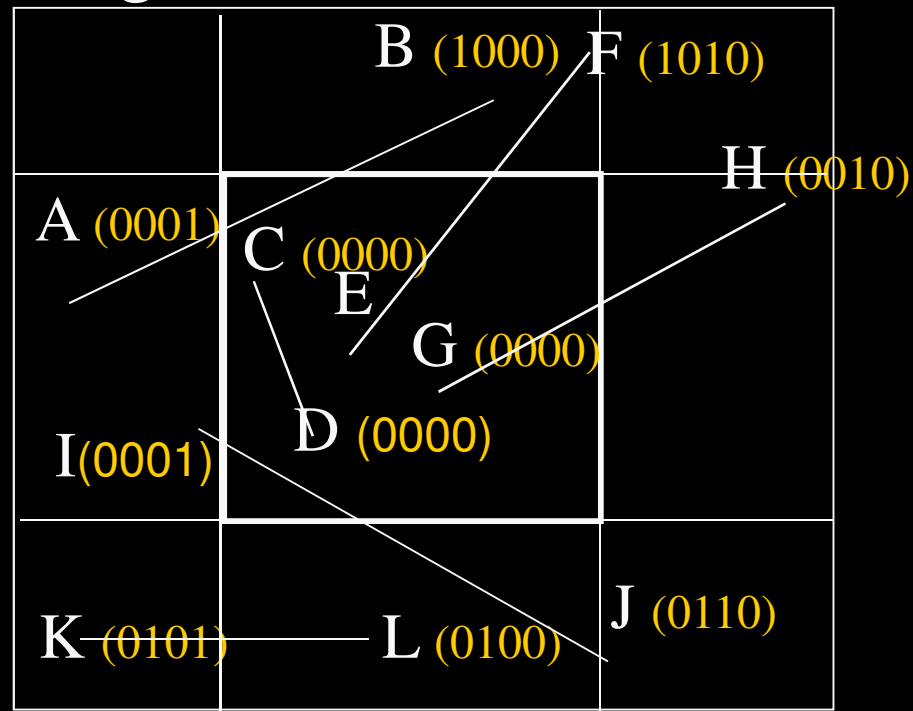
Algoritmo de Cohen-Sutherland



- Bit codes dos pontos?
- Como devem ser os bit codes dos pontos trivialmente aceitos?
- Como devem ser os bit codes das arestas trivialmente recusadas?
- O que fazer com os que não são nem aceitos nem recusados?

Algoritmos de recorte

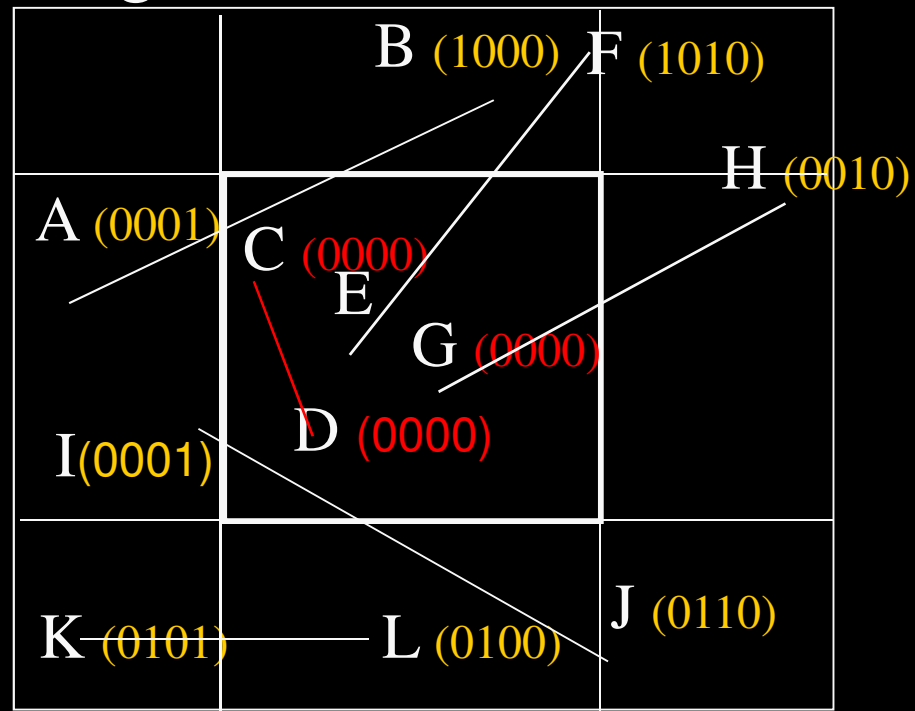
Algoritmo de Cohen-Sutherland



- Bit codes dos pontos?

Algoritmos de recorte

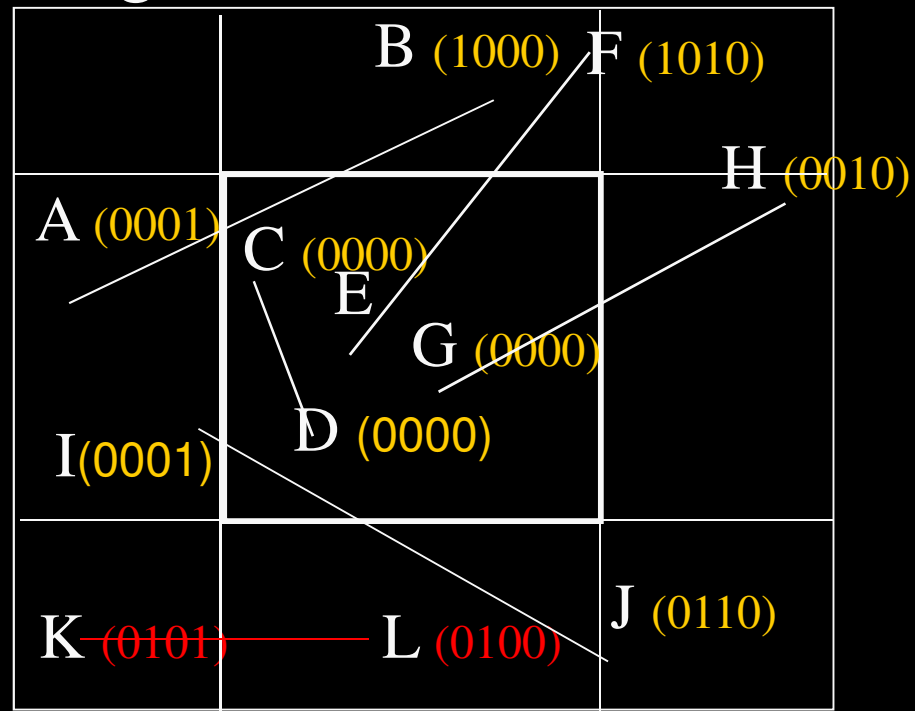
Algoritmo de Cohen-Sutherland



- Bit codes dos pontos?
- Como devem ser os bit codes dos pontos trivialmente aceitos?
0000
- Arestas formadas por pontos trivialmente aceitos são trivialmente aceitas

Algoritmos de recorte

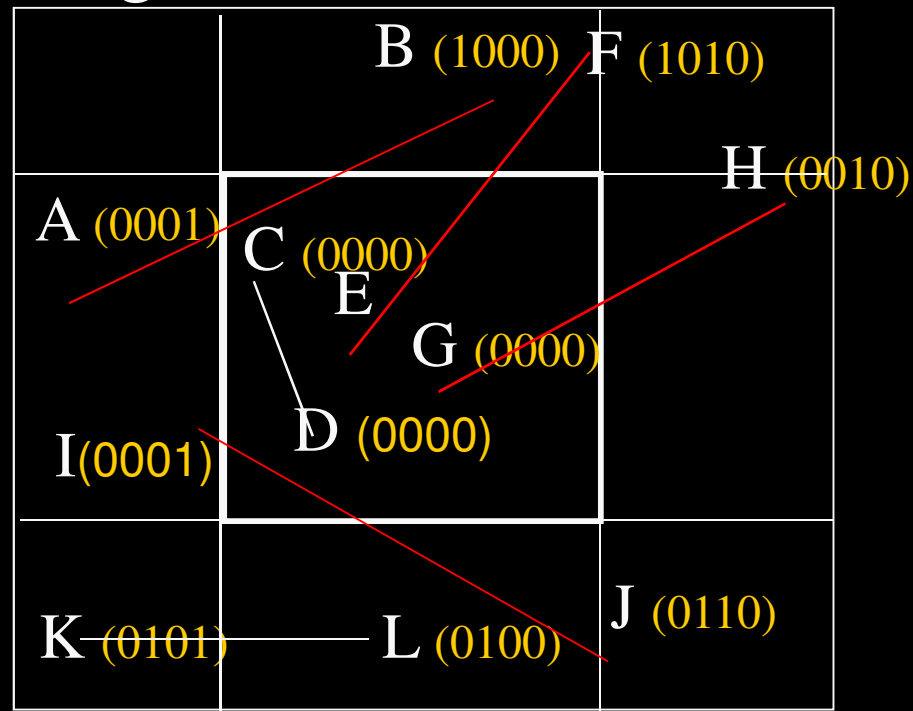
Algoritmo de Cohen-Sutherland



- Como devem ser os bit codes dos pontos trivialmente aceitos? **0000**
- Como devem ser os bit codes das arestas trivialmente recusadas? **AND** entre bit codes dos pontos deve ser **!= 0**

Algoritmos de recorte

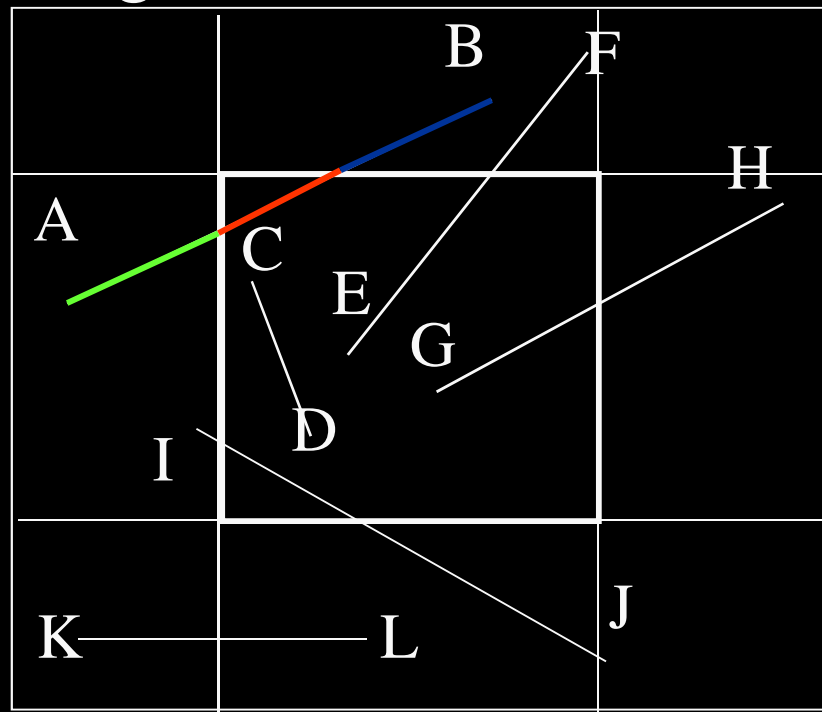
Algoritmo de Cohen-Sutherland



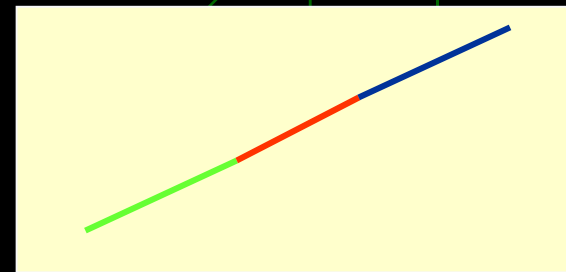
- Como devem ser os bit codes dos pontos trivialmente aceitos? **0000**
- Como devem ser os bit codes das arestas trivialmente recusadas? **AND entre end points deve ser != 0**
- O que fazer com os que não são nem aceitos nem recusados?

Algoritmos de recorte

Algoritmo de Cohen-Sutherland



- Calcular segmentos através das intersecções



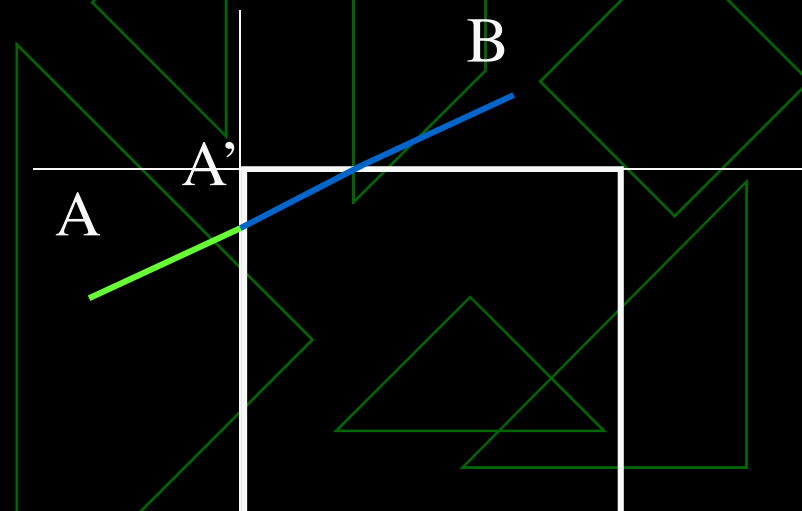
Algoritmos de recorte

Processo Iterativo:

- 1) Pares de pontos das arestas são testados para serem aceitos ou recusados trivialmente (slides anteriores)
- 2) Para as arestas A que sobram:
 - i) Calcular as intersecções da aresta A_i com as arestas do polígono de recorte;
 - ii) dividir a aresta em 2 segmentos: antes da primeira intersecção e depois
 - iii) Testar se recusado ou aceito para cada segmento da aresta

$A-B$

$A-A'$ e $A'-B$



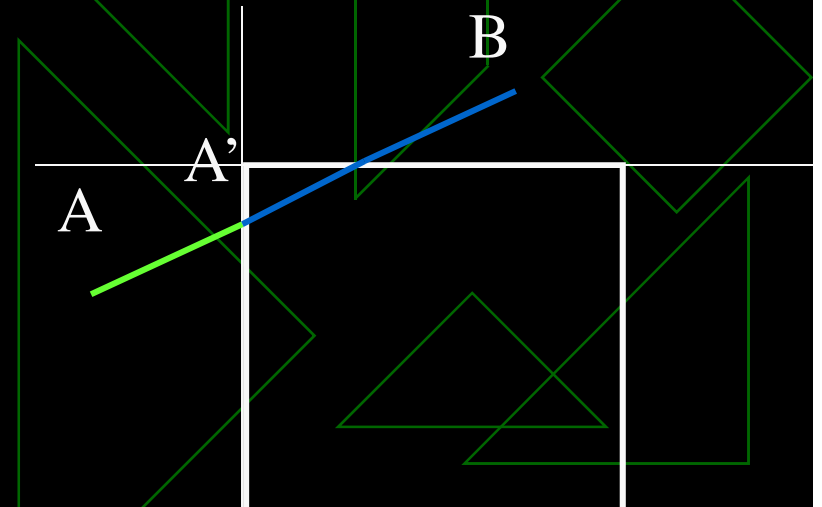
Algoritmos de recorte

Processo Iterativo:

- 1) Pares de pontos das arestas são testados para serem aceitados ou recusados trivialmente (slides anteriores)
- 2) Para as arestas A que sobram:
 - i) Calcular as interações da aresta A_i com as arestas do polígono de recorte;
 - ii) dividir a aresta em 2 segmentos: antes da primeira intersecção e depois
 - iii) Testar se recusado ou aceito para cada segmento da aresta

$A-B$

$A-A'$ (TR) e $A'-B$



Algoritmos de recorte

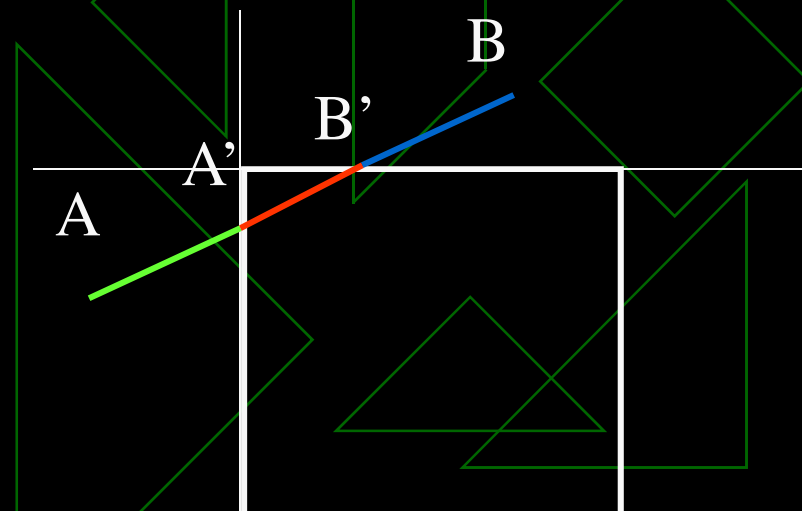
Processo Iterativo:

- 1) Pares de pontos das arestas são testados para serem aceitados ou recusados trivialmente (slides anteriores)
- 2) Para as arestas A que sobram:
 - i) Calcular as interações da aresta A_i com as arestas do polígono de recorte;
 - ii) dividir a aresta em 2 segmentos: antes da primeira intersecção e depois
 - iii) Testar se recusado ou aceito para cada segmento da aresta

$A-B$ = deve ser recortado

$A-A'$ (TR) e $A'-B$

$A'-B'$ e $B'-B$



Algoritmos de recorte

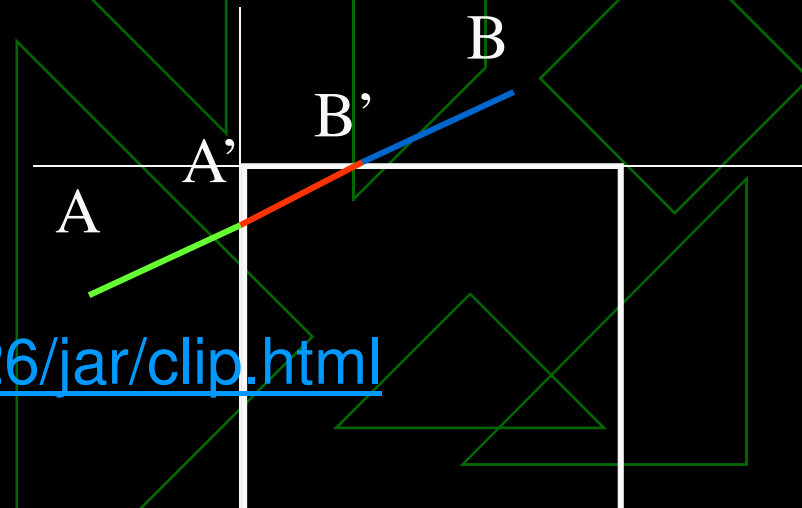
Processo Iterativo:

- 1) Pares de pontos das arestas são testados para serem aceitados ou recusados trivialmente (slides anteriores)
- 2) Para as arestas A que sobram:
 - i) Calcular as interações da aresta A_i com as arestas do polígono de recorte;
 - ii) dividir a aresta em 2 segmentos: antes da primeira intersecção e depois
 - iii) Testar se recusado ou aceito para cada segmento da aresta

$A-B$ = deve ser recortado

$A-A'$ (TR) e $A'-B$

$A'-B'$ (TA) e $B'-B$ (TR)



<http://www.cs.princeton.edu/~min/cs426/jar/clip.html>

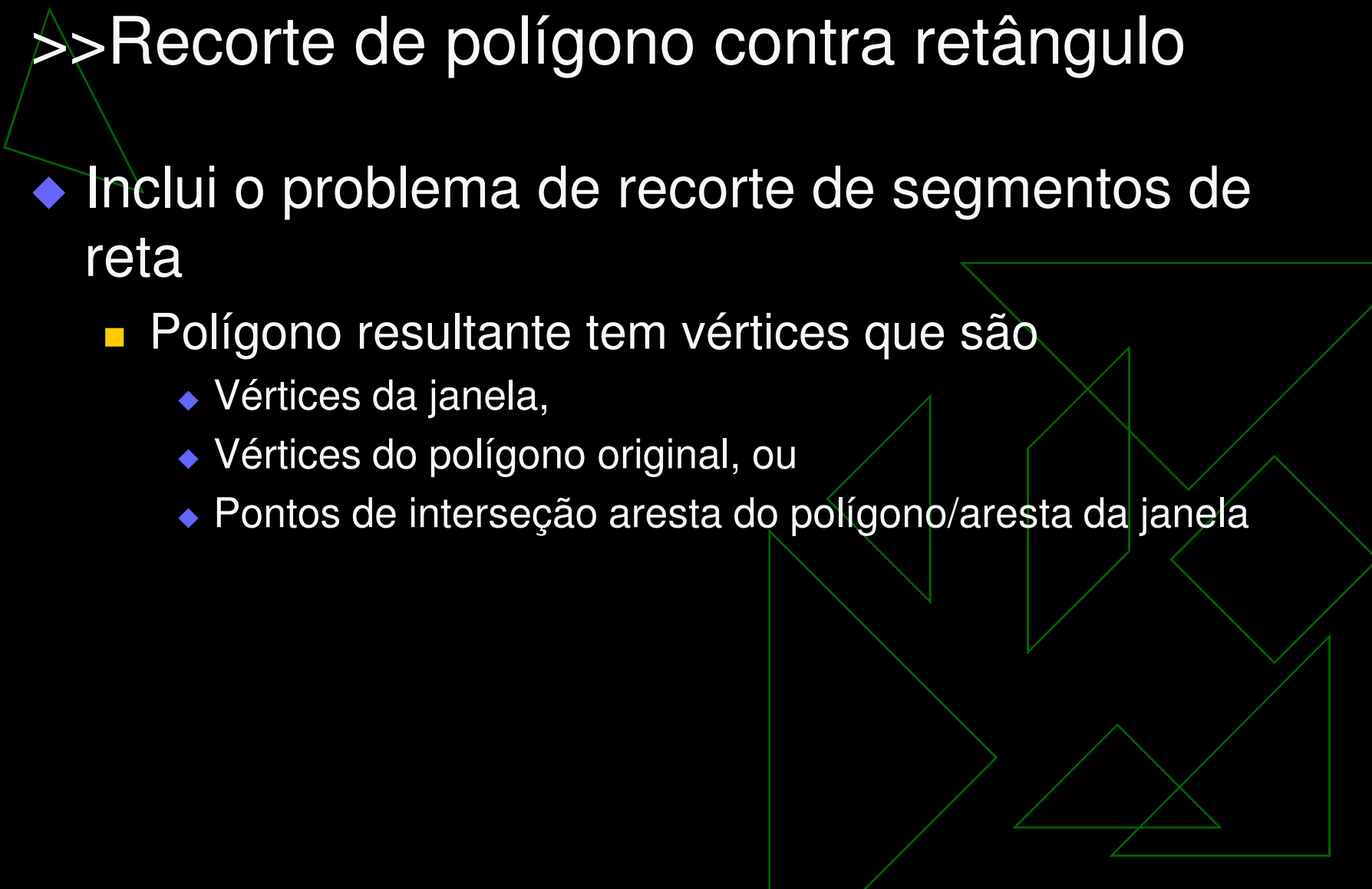
Algoritmos de recorte

>> Recorte de polígono contra retângulo

- ◆ Inclui o problema de recorte de segmentos de reta

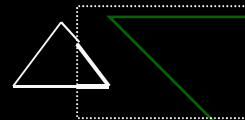
- Polígono resultante tem vértices que são

- ◆ Vértices da janela,
- ◆ Vértices do polígono original, ou
- ◆ Pontos de interseção aresta do polígono/aresta da janela

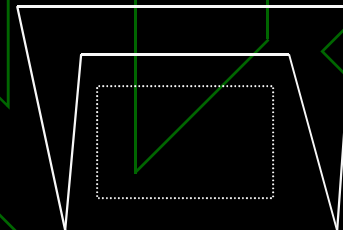
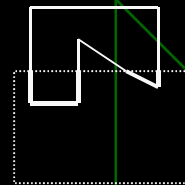
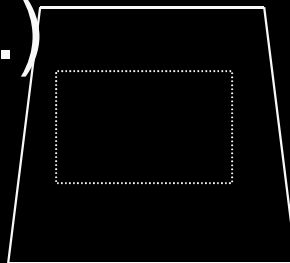


Recorte de Polígono contra Retângulo

◆ Casos Simples

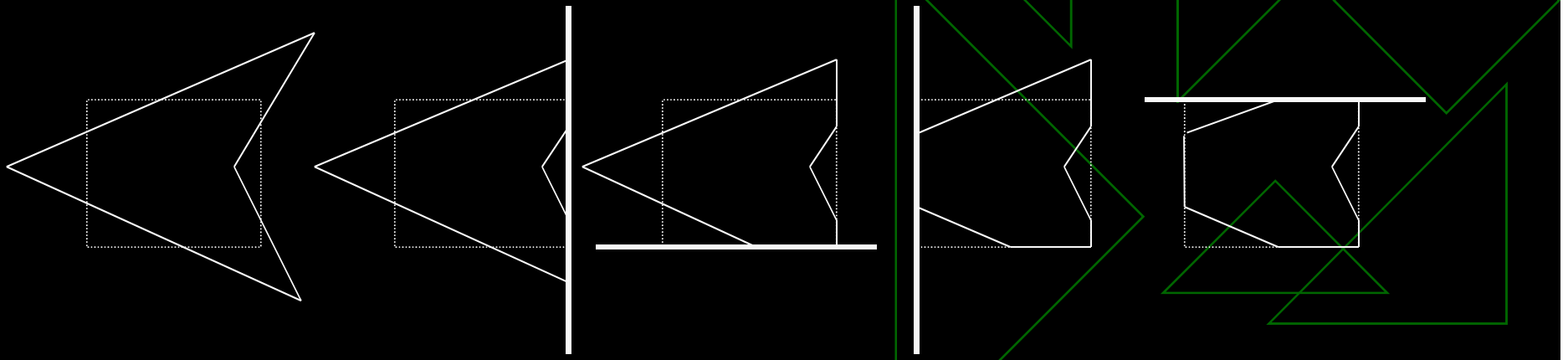


◆ Casos Complicados (pode mudar topologia...)



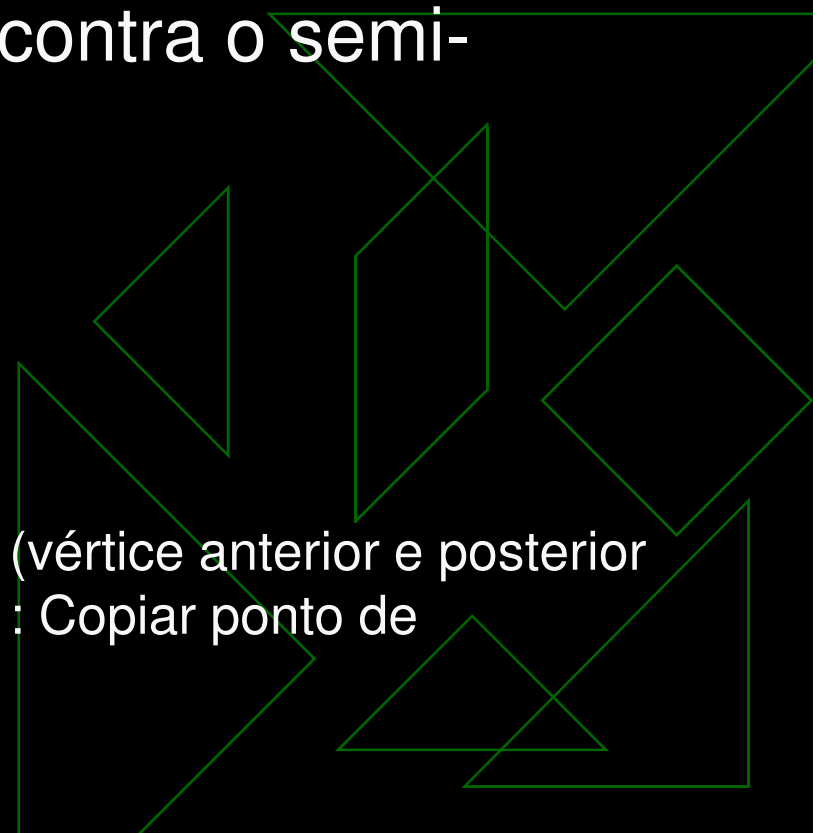
Algoritmo de Sutherland-Hodgman

- ◆ Idéia é semelhante à do algoritmo de Sutherland-Cohen
 - Recortar o polígono sucessivamente contra todos os semi-espacos planos da figura de recorte

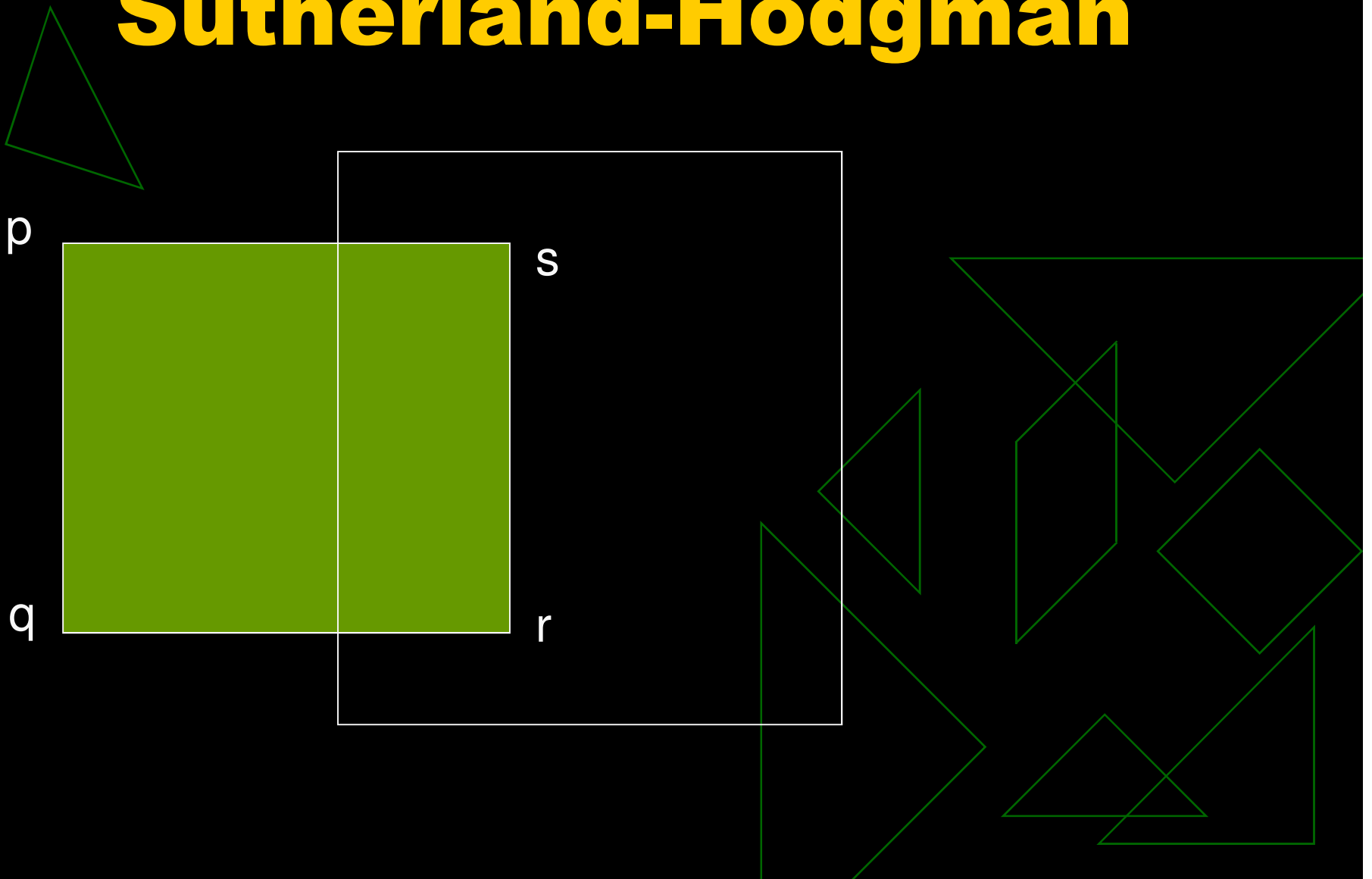


Algoritmo de Sutherland-Hodgman

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



Algoritmo de Sutherland-Hodgman

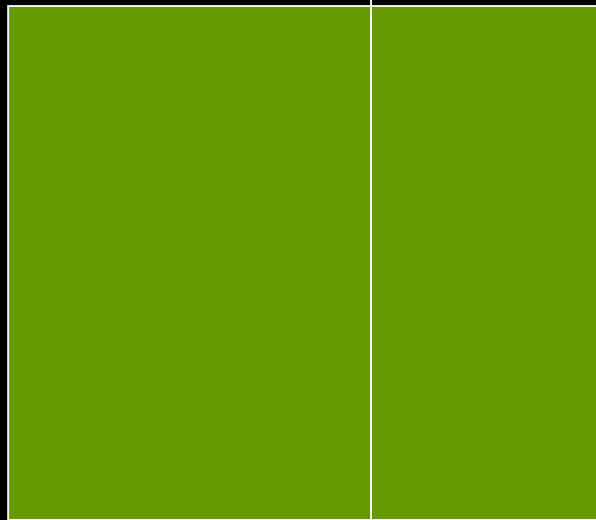


Algoritmo de Sutherland-Hodgman



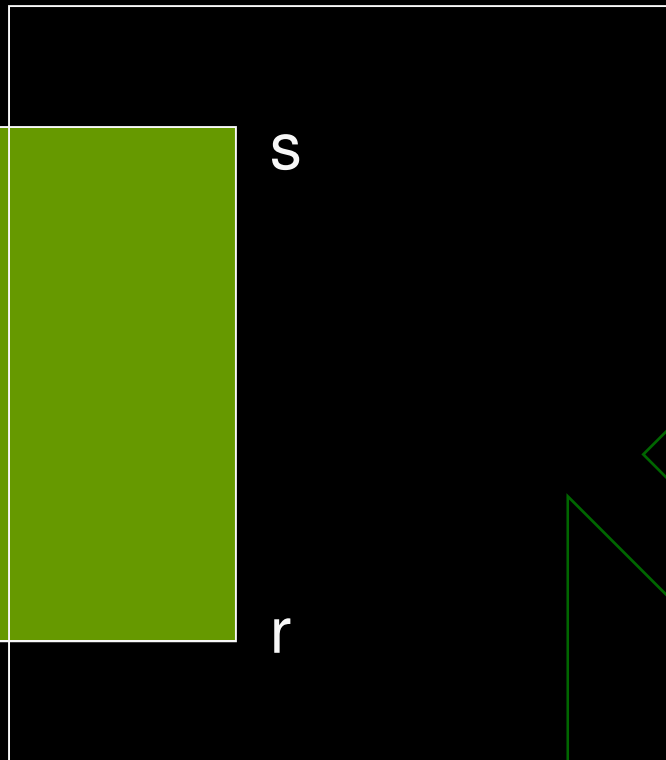
p

q

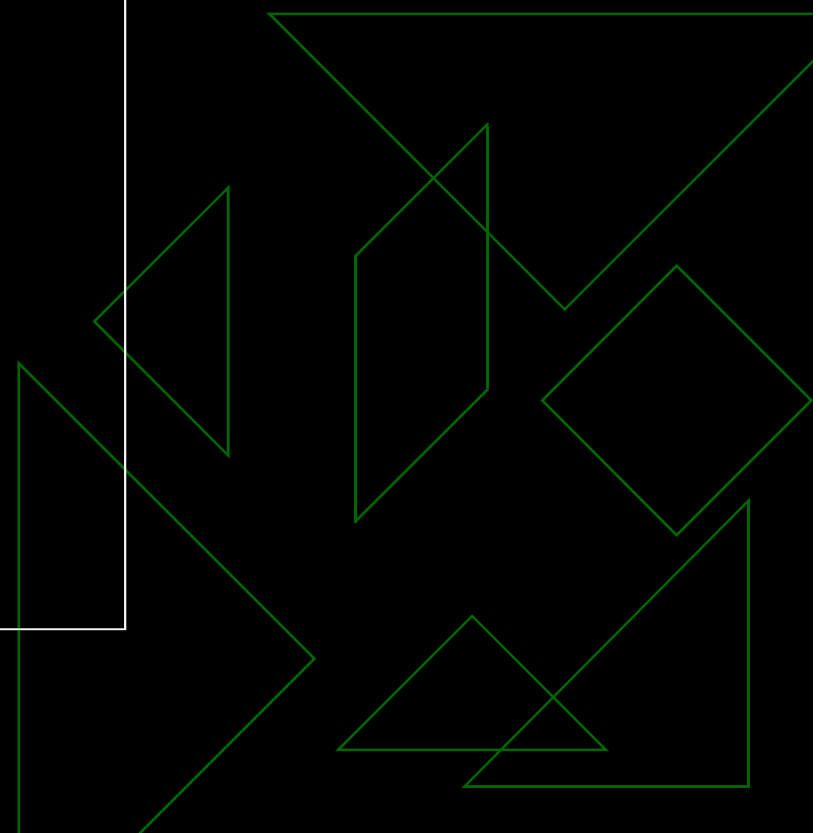


s

r



Vértice p -> fora

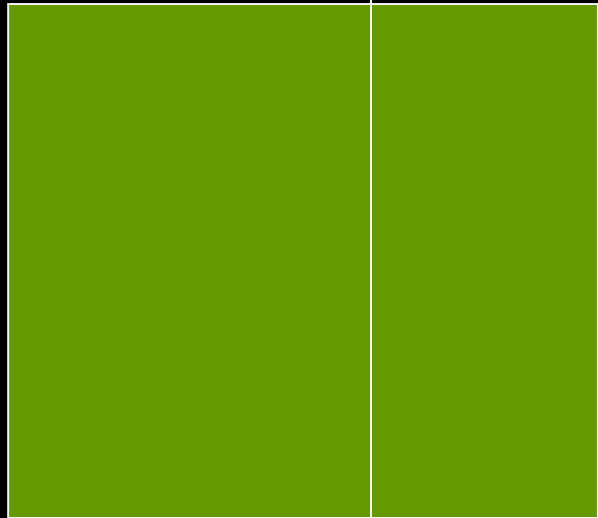


Algoritmo de Sutherland-Hodgman



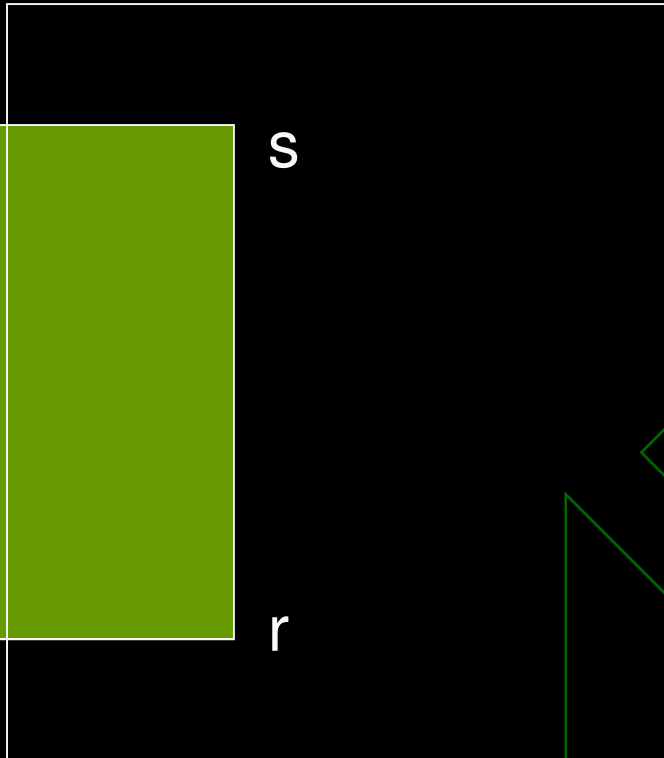
p

q

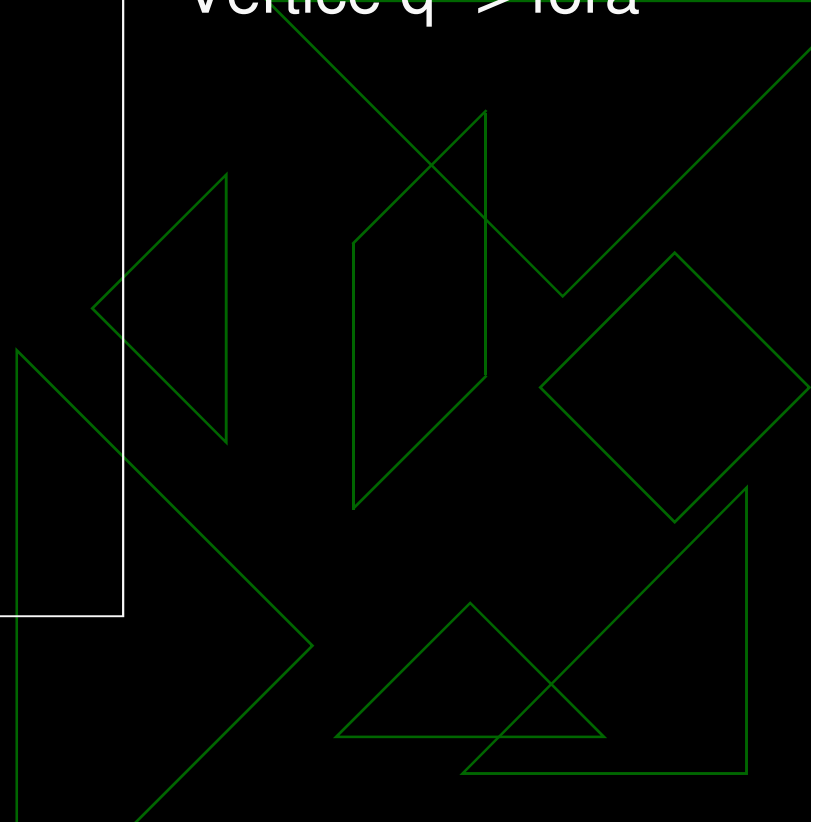


s

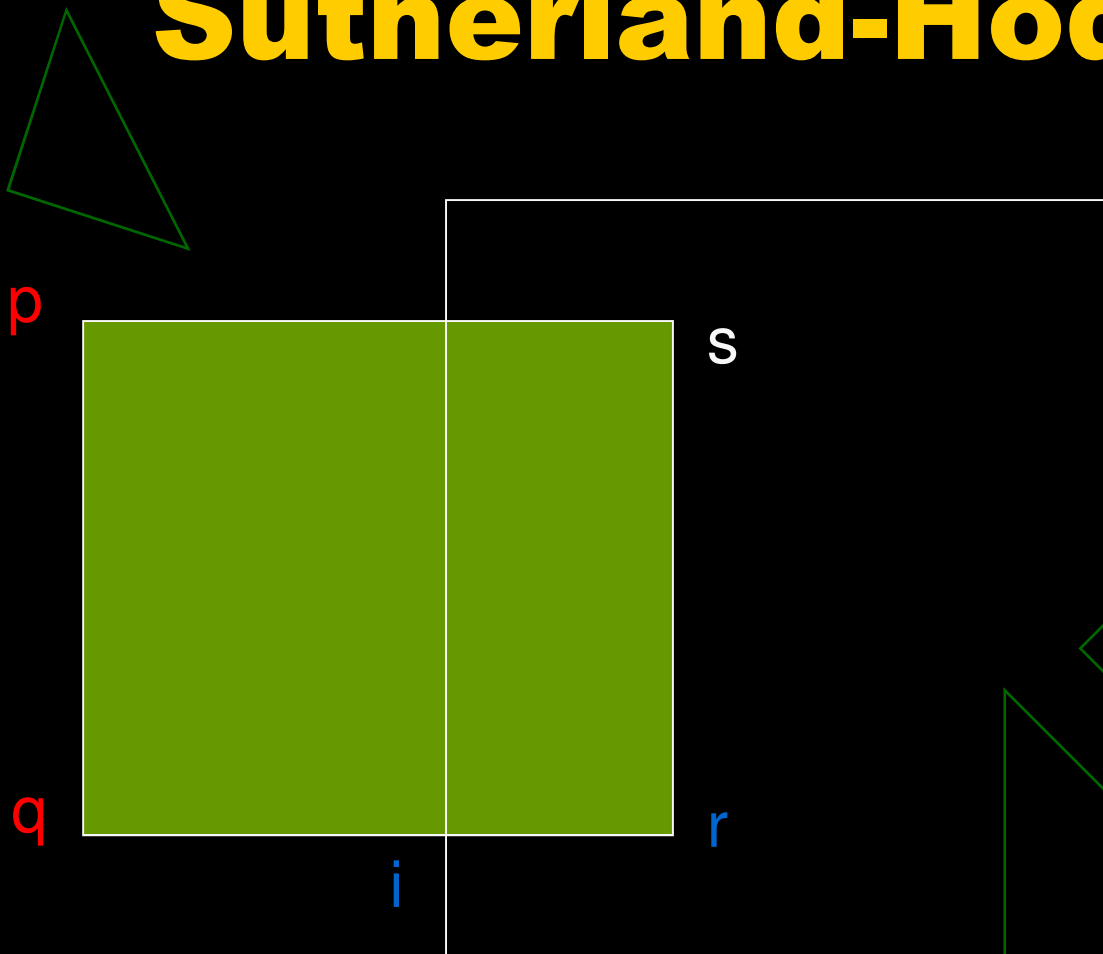
r



Vértice p -> fora
Vértice q -> fora

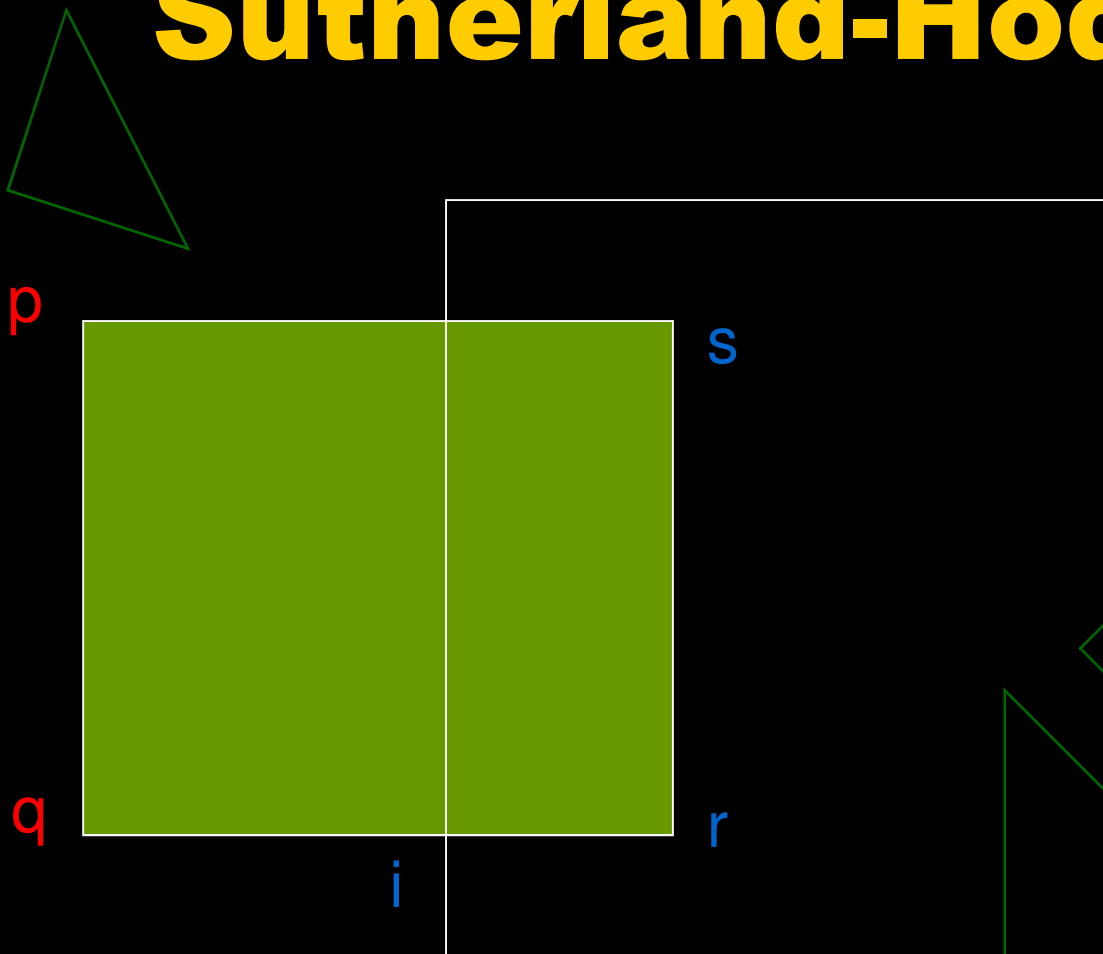


Algoritmo de Sutherland-Hodgman



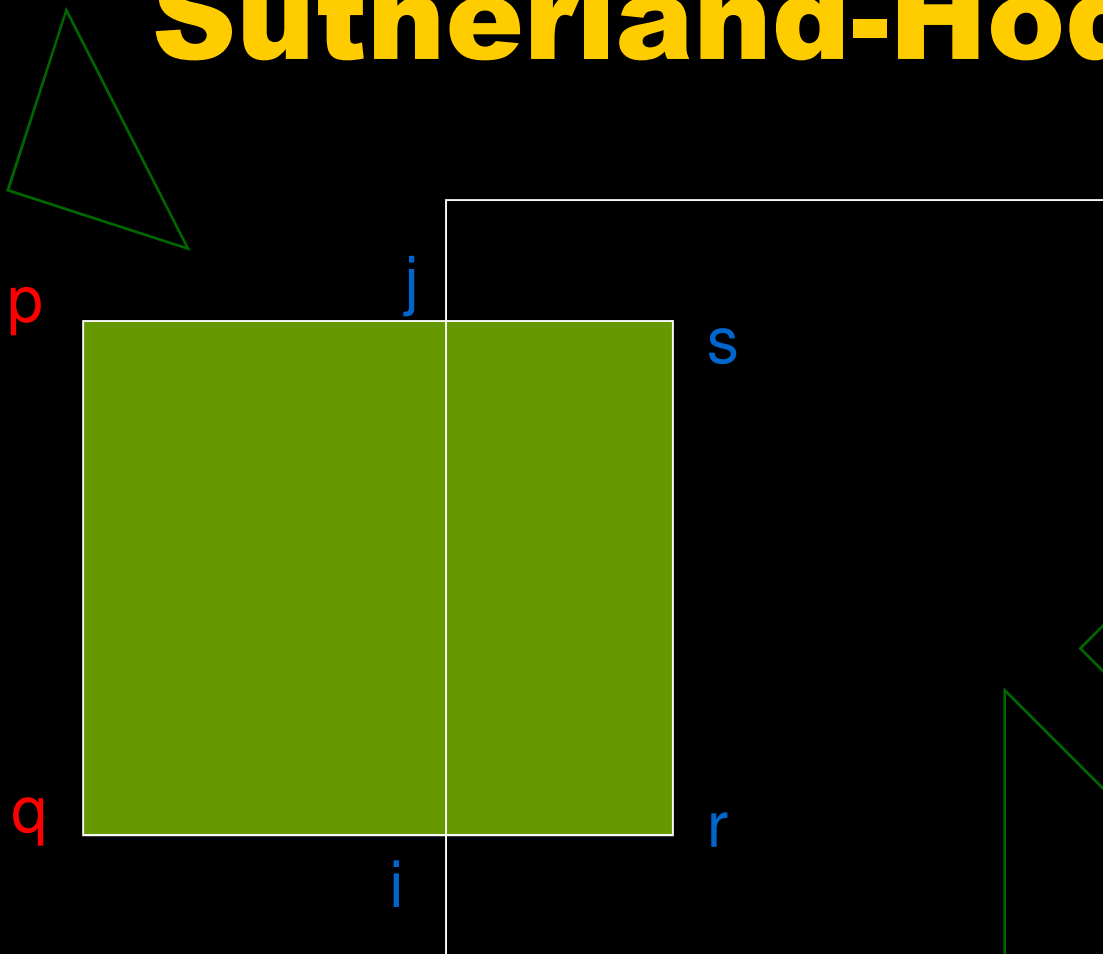
Vértice p -> fora
Vértice q -> fora
Vértice r -> dentro
Registra intersecção i
entre q e r (estados
diferentes)

Algoritmo de Sutherland-Hodgman



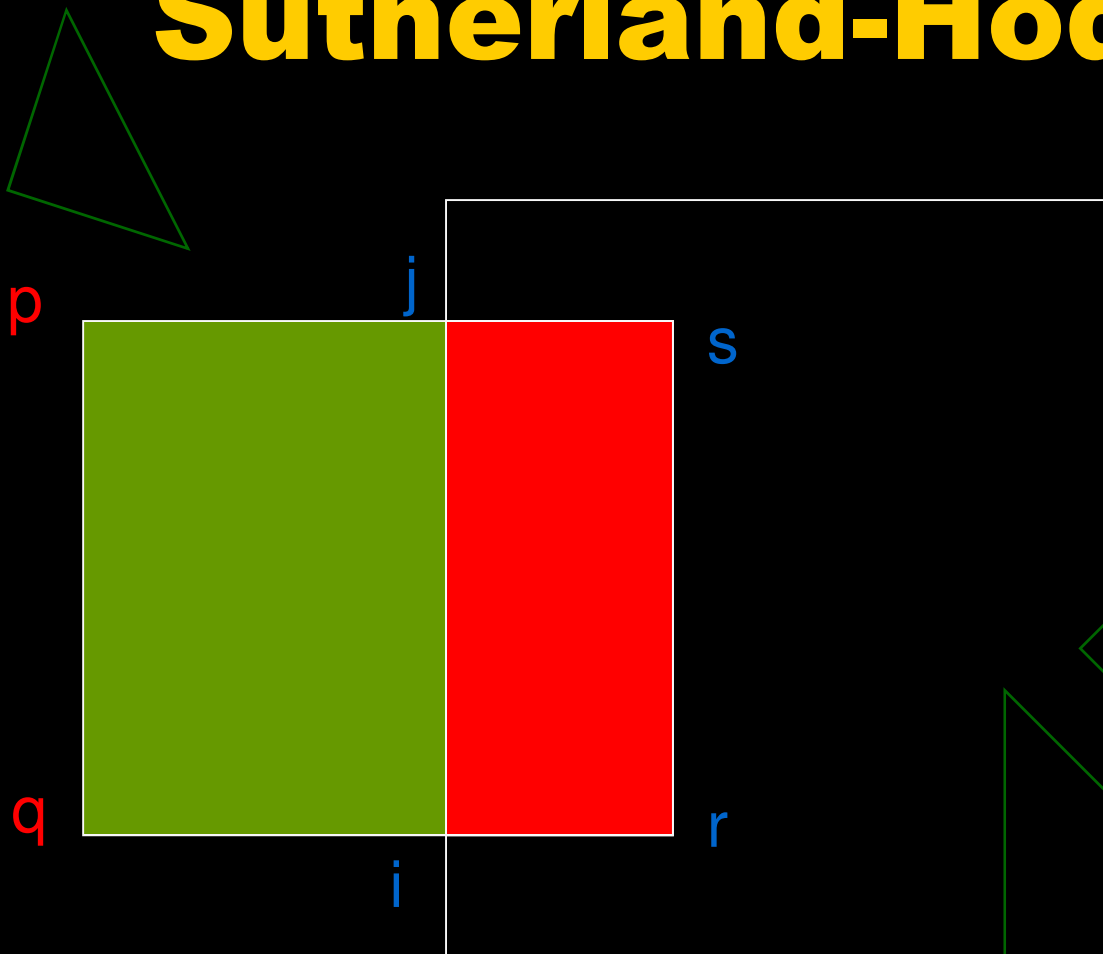
Vértice p -> fora
Vértice q -> for a
Vértice r -> dentro
Registra intersecção i
entre q e r (estados
diferentes)
Vértice s -> dentro

Algoritmo de Sutherland-Hodgman



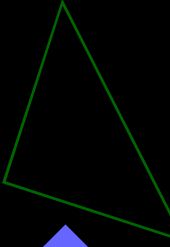
Vértice p -> fora
Vértice q -> for a
Vértice r -> dentro
Registra intersecção i
entre q e r (estados
diferentes)
Vértice s -> dentro
Registra j entre s e p

Algoritmo de Sutherland-Hodgman



Vértice p -> fora
Vértice q -> for a
Vértice r -> dentro
Registra intersecção i
entre q e r (estados
diferentes)
Vértice s -> dentro
Registra j entre s e p

SR's 2D



◆ SRO

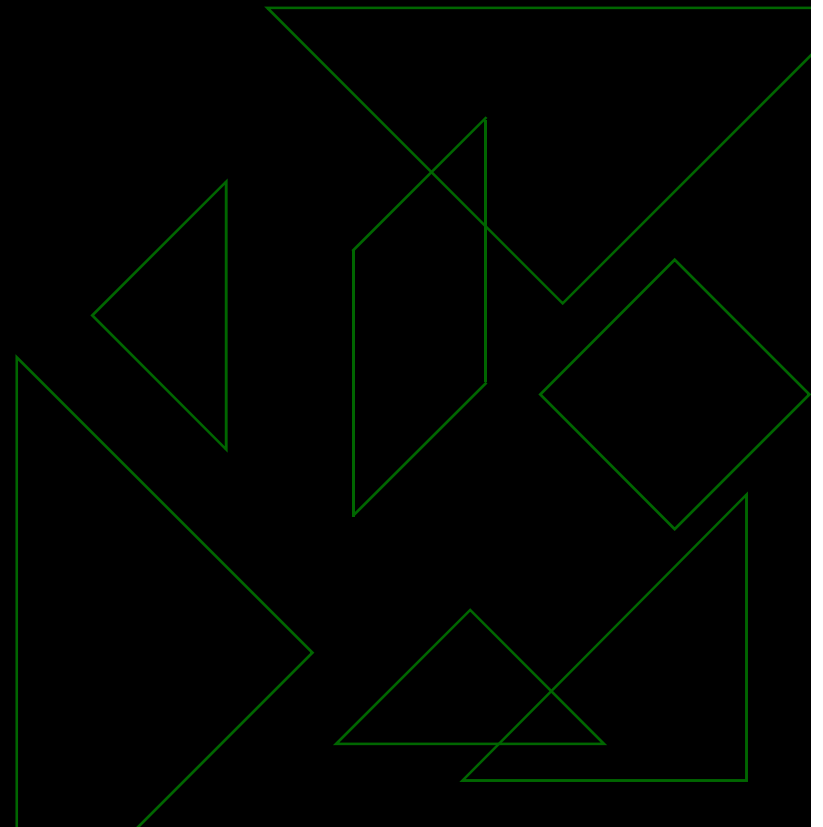
◆ SRU

◆ SRW

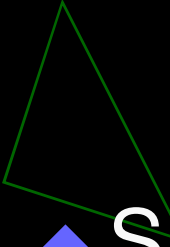
(recorte 2D)

◆ SRV

◆ SRD



SR's 3D



- ◆ SRO

- ◆ SRU

- ◆ SRC

(recorte 3D)

- ◆ SRP

- ◆ SRV-SRD

