

# Pipeline de Visualização 2D

André Tavares da Silva

[andre.silva@udesc.br](mailto:andre.silva@udesc.br)

Capítulo 2 do “Foley”

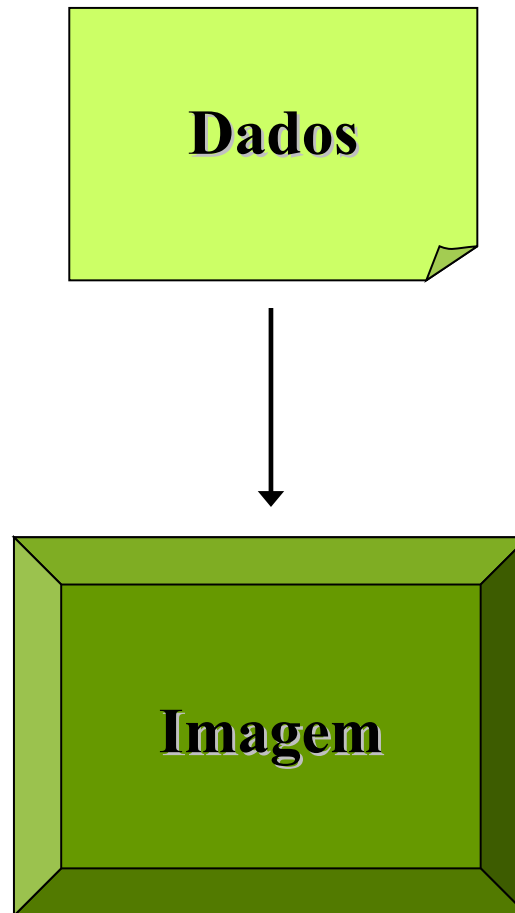
# Requisitos de matemática para CG

- Vetores e pontos
- Matrizes
- Transformações geométricas
- Pontos e espaços afim
- Representação de coordenadas
- Reta
- Plano

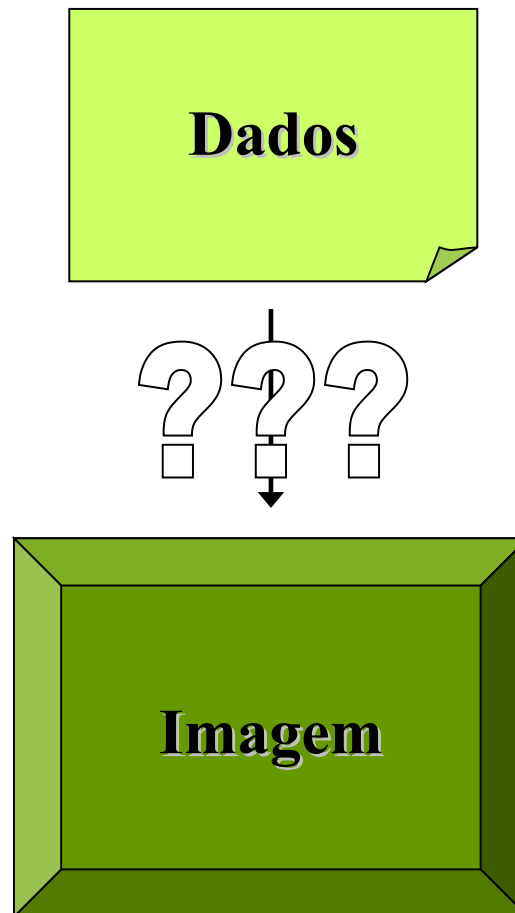
# Computação Gráfica

**Conjunto de técnicas utilizadas para converter dados, de forma a exibi-los em dispositivos gráficos.**

# Computação Gráfica



# Computação Gráfica



# Visualização Bidimensional

# Visualização 2D

**Modelo**

**Dados**

.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....

.....

Processo de  
Visualização 2D

**Imagem**



# Visualização 2D

## Modelo

```
typedef struct  
{  
    float xi, yi;  
    float xf, yf;  
} linha;  
...
```

Estrutura de  
dados

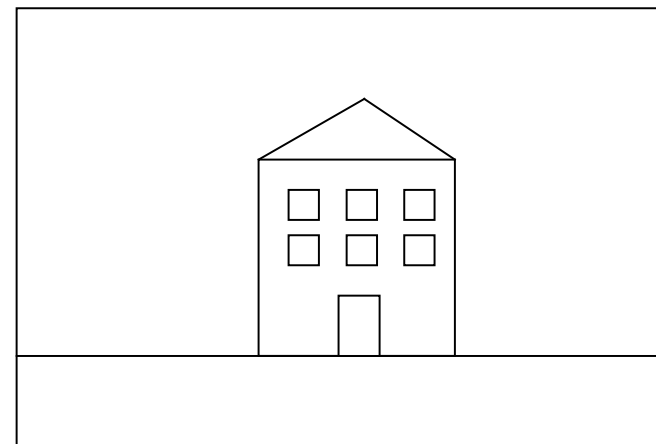
Vértices Arestas



**Pipeline 2D**



## Imagem



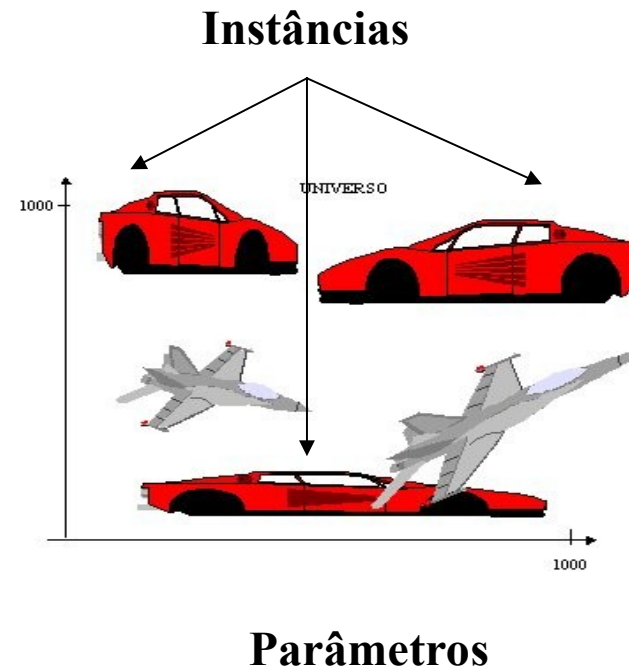
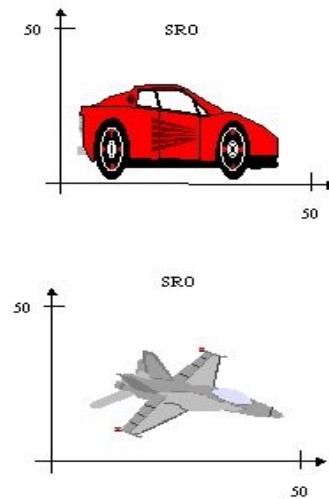
Mapa de bits



# Instanciamento

## Objetos

- Carro
- Avião



# Recorte

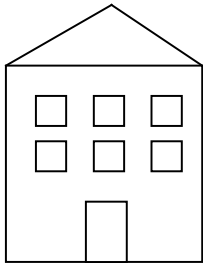
- Permite definir qual região da imagem será vista.

# Mapeamento

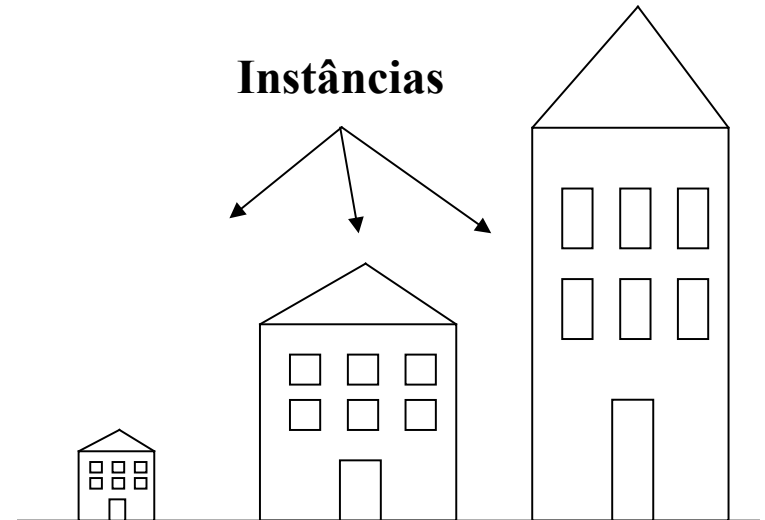
- Permite que se exiba em um dispositivo (tela) a visualização desejada do universo.

# Instanciamento

**Objeto Original**



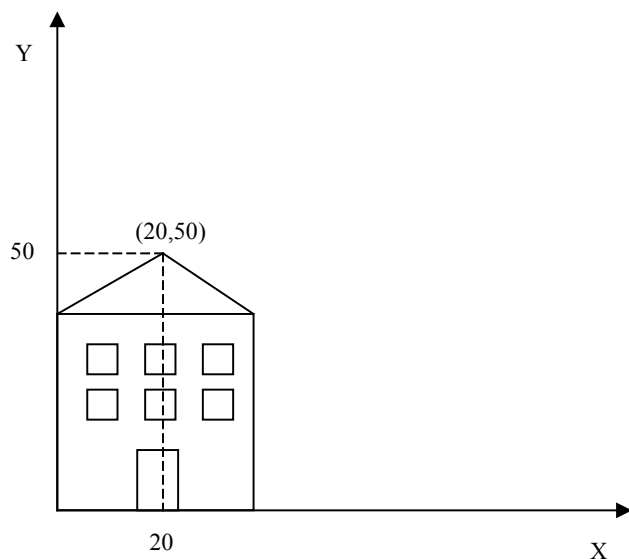
**Instâncias**



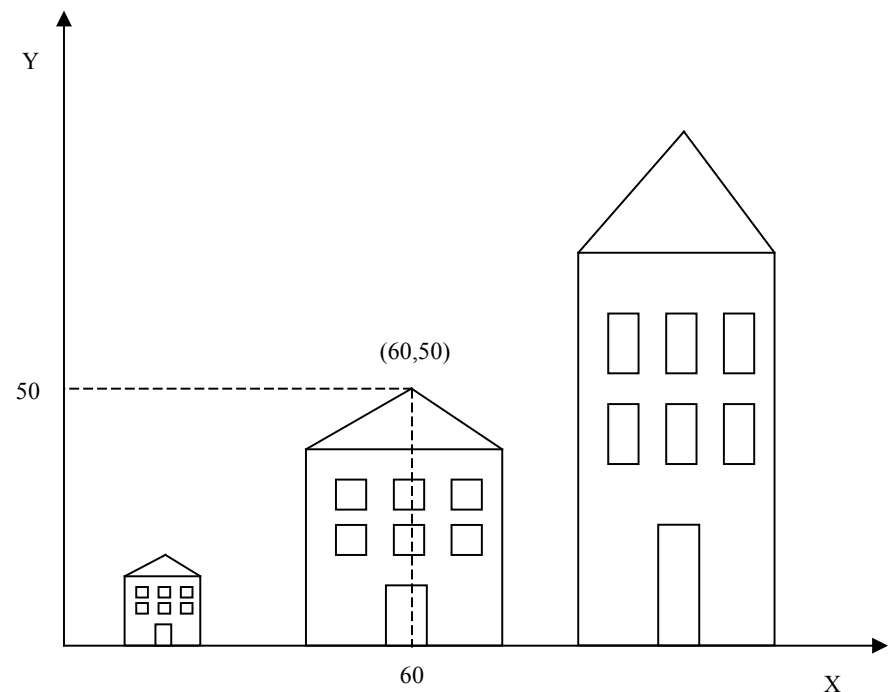
```
typedef struct  
{  
    float xi, yi;  
    float xf, yf;  
} linha;
```

**Lembrando!**  
Descrito por  
linhas.

# Sistemas de Referência

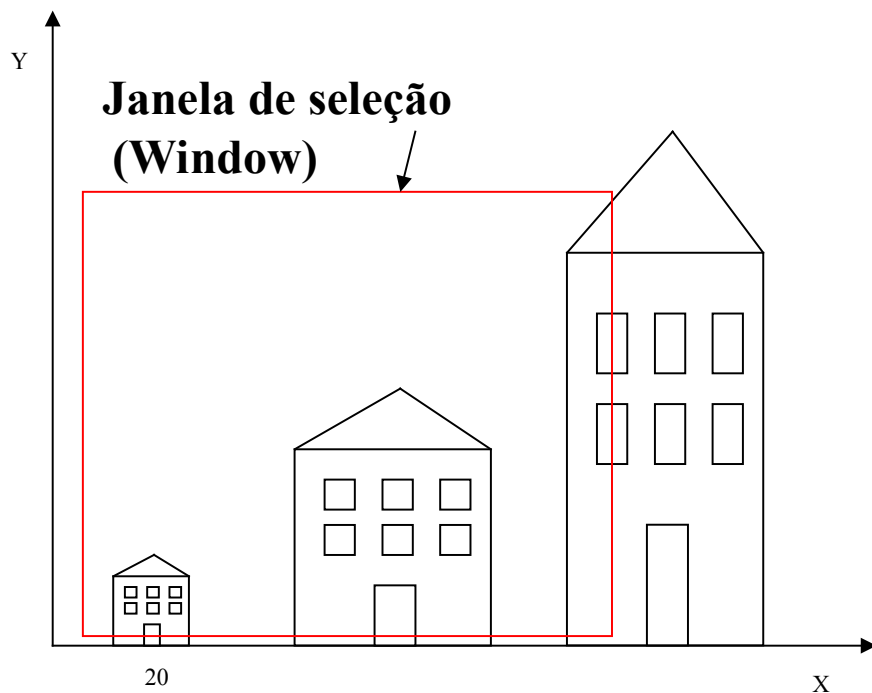


**Sistema de  
referência do objeto  
(SRO)**

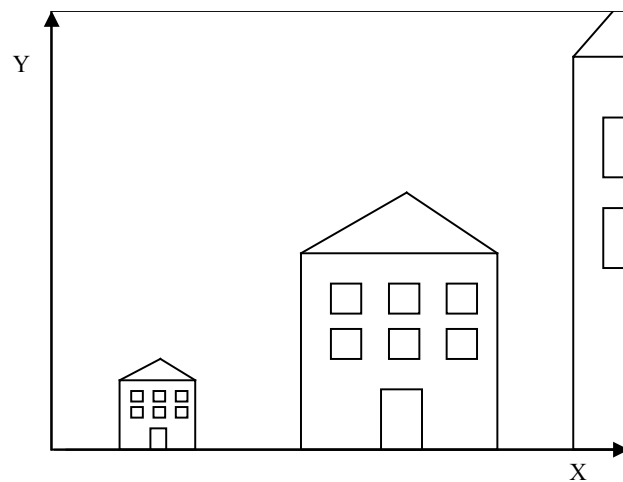


**Sistema de  
referência do universo  
(SRU)**

# Sistemas de Referência

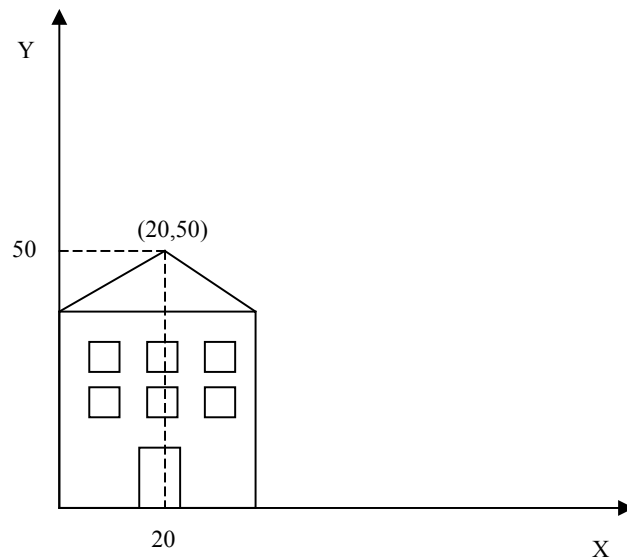


**Sistema de  
referência do universo  
(SRU)**



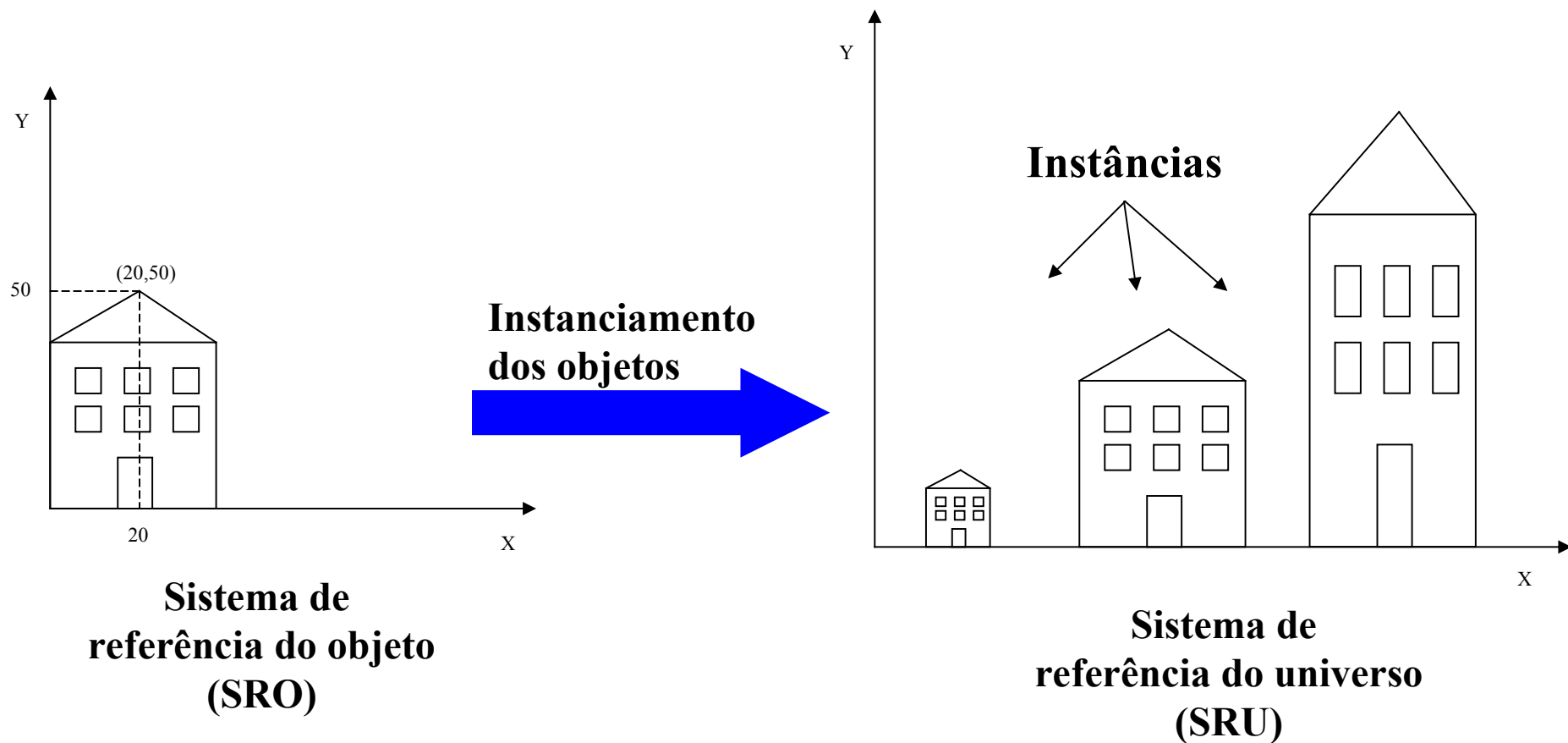
**Sistema de  
referência da seleção  
(SRS)**

# Processo de Visualização 2D (pipeline)



**Sistema de  
referência do objeto  
(SRO)**

# Processo de Visualização 2D (pipeline)



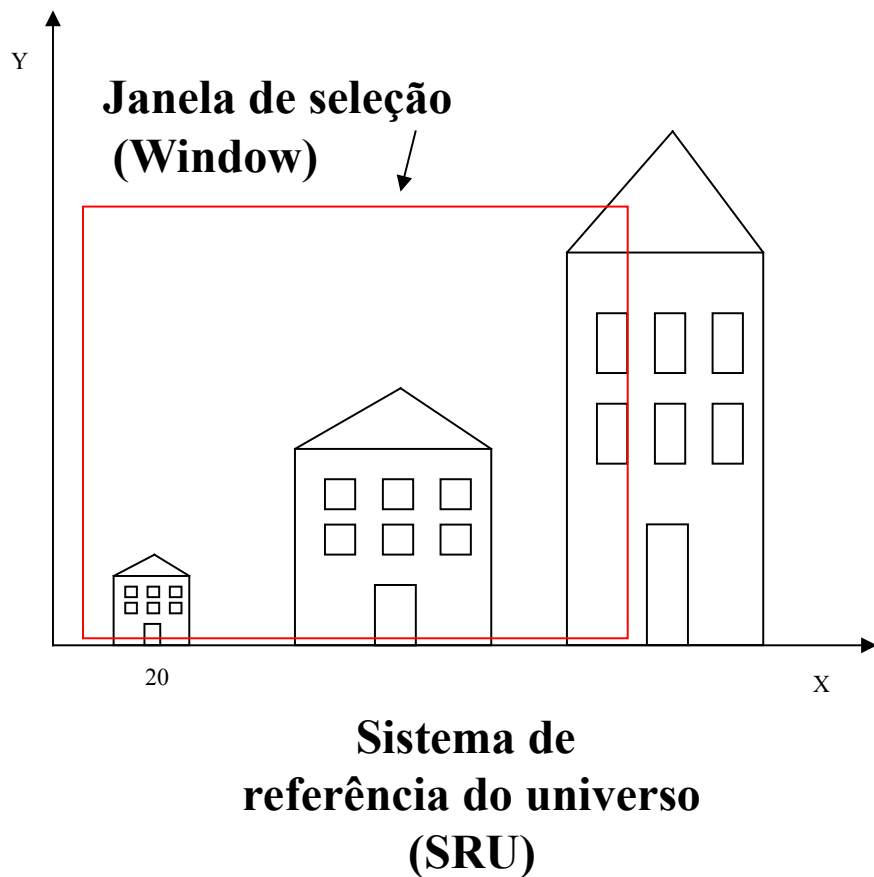


# Processo de Visualização 2D (pipeline)

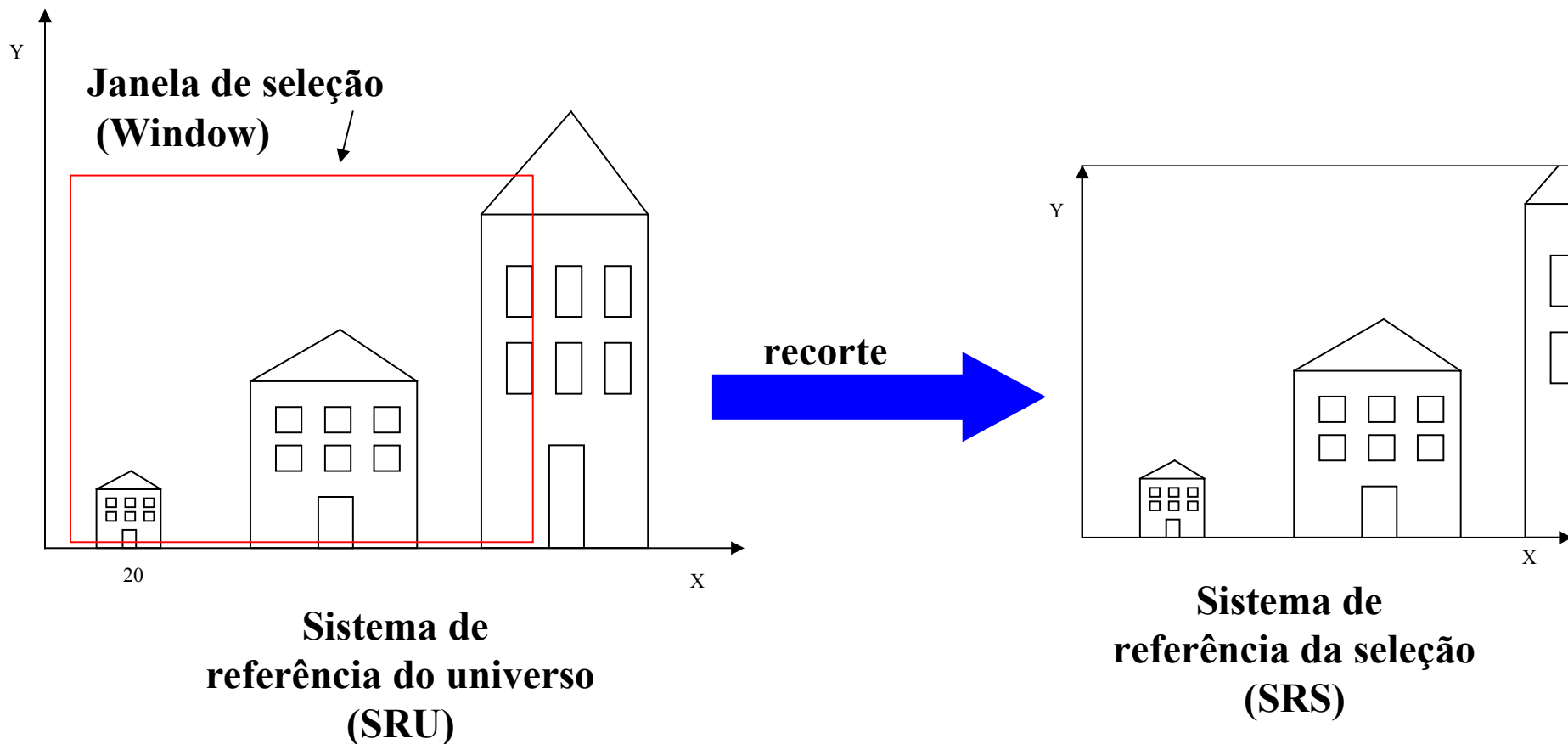


**Sistema de  
referência do universo  
(SRU)**

# Processo de Visualização 2D (pipeline)



# Processo de Visualização 2D (pipeline)

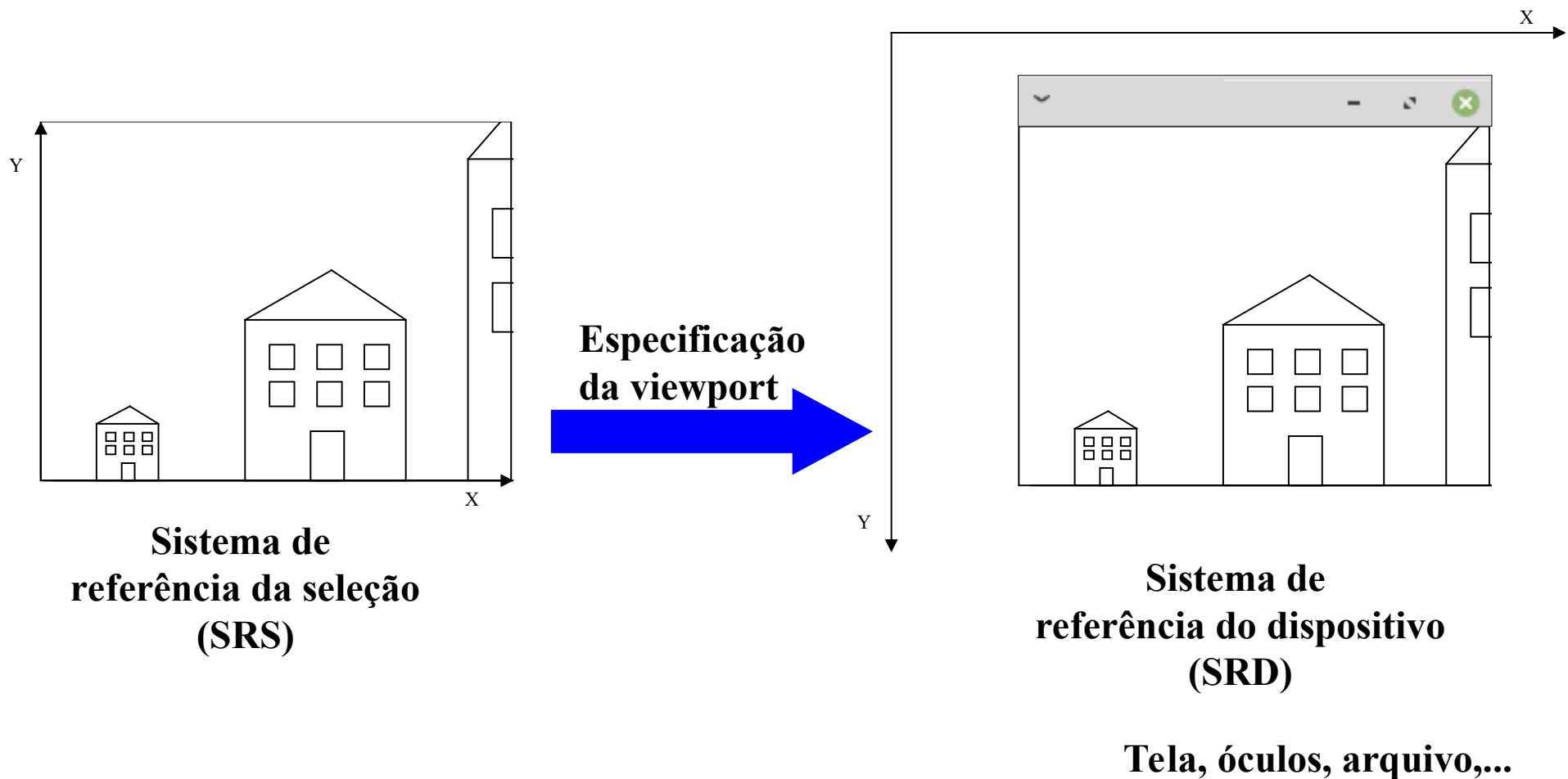


# Processo de Visualização 2D (pipeline)

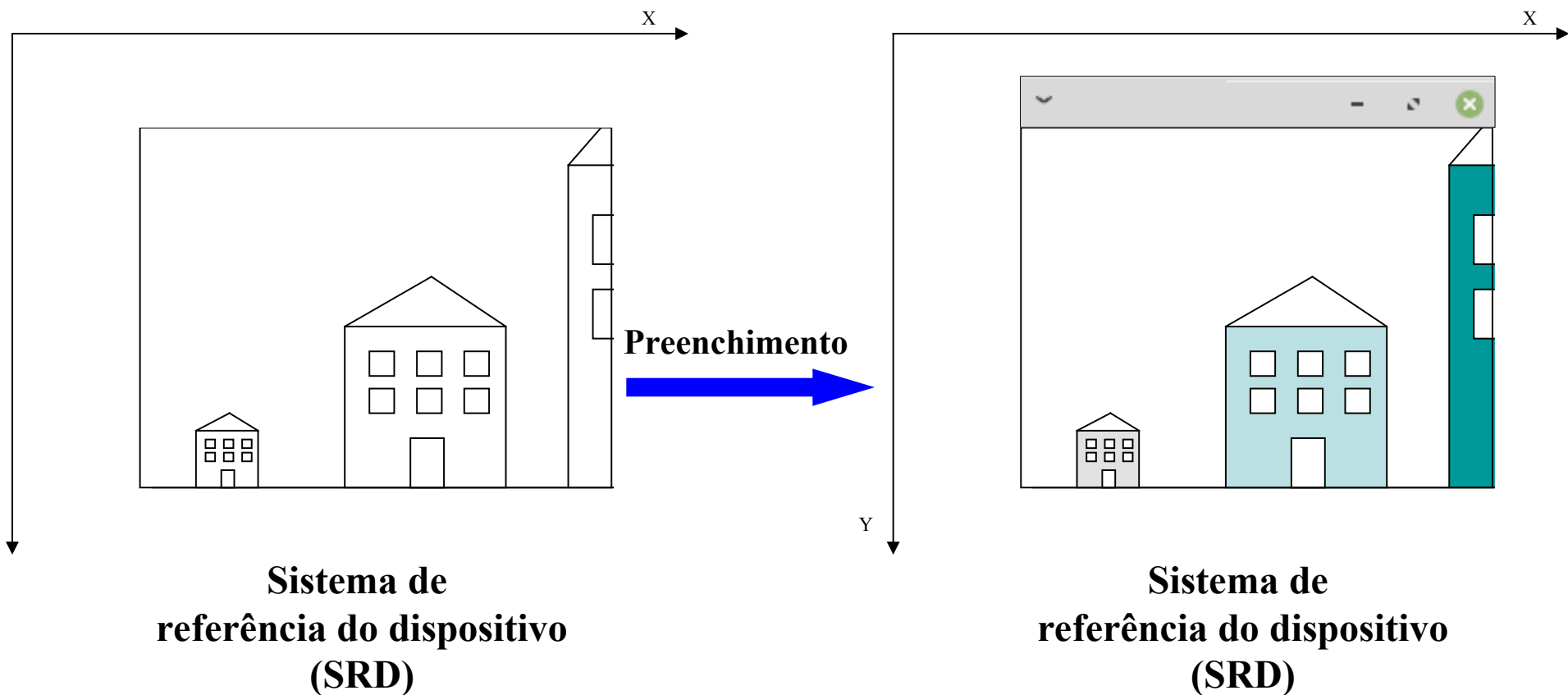


**Sistema de  
referência da seleção  
(SRS)**

# Processo de Visualização 2D (pipeline)

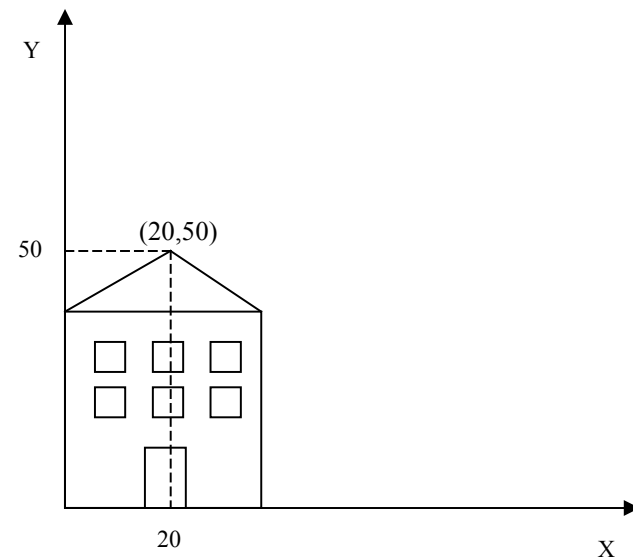


# Processo de Visualização 2D (pipeline)



# Etapas

**Sistema de referência  
do objeto (SRO)**



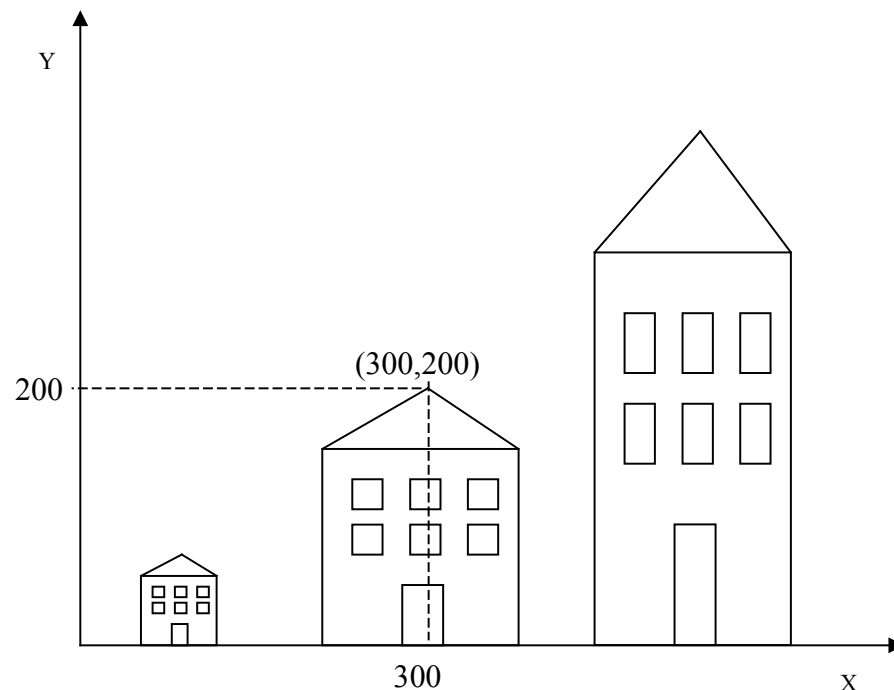
**Sistema de  
referência do objeto  
(SRO)**

# Etapas

**Sistema de referência  
do objeto (SRO)**

instanciamento

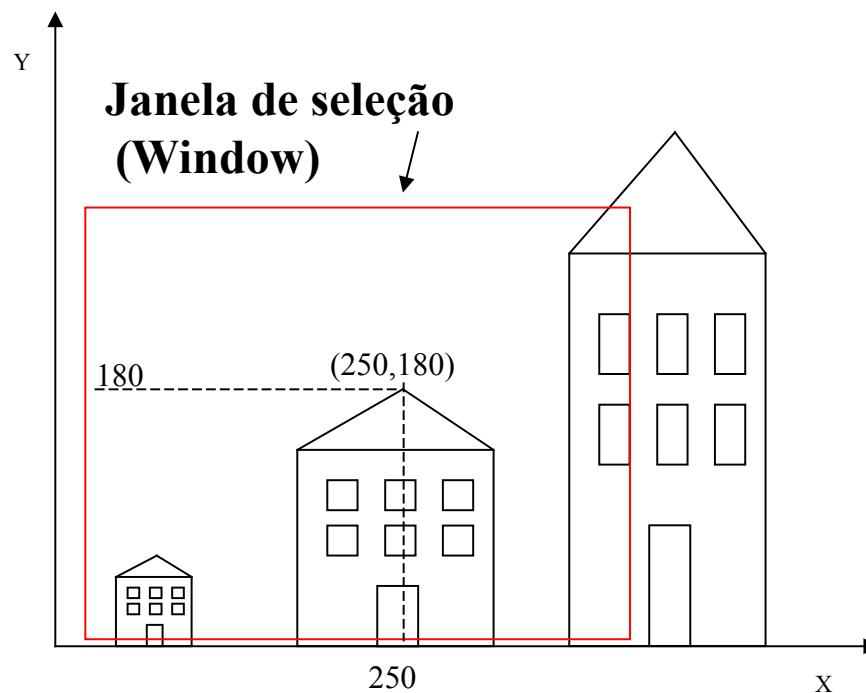
**Sistema de referência  
do universo (SRU)**



**Sistema de  
referência do universo  
(SRU)**

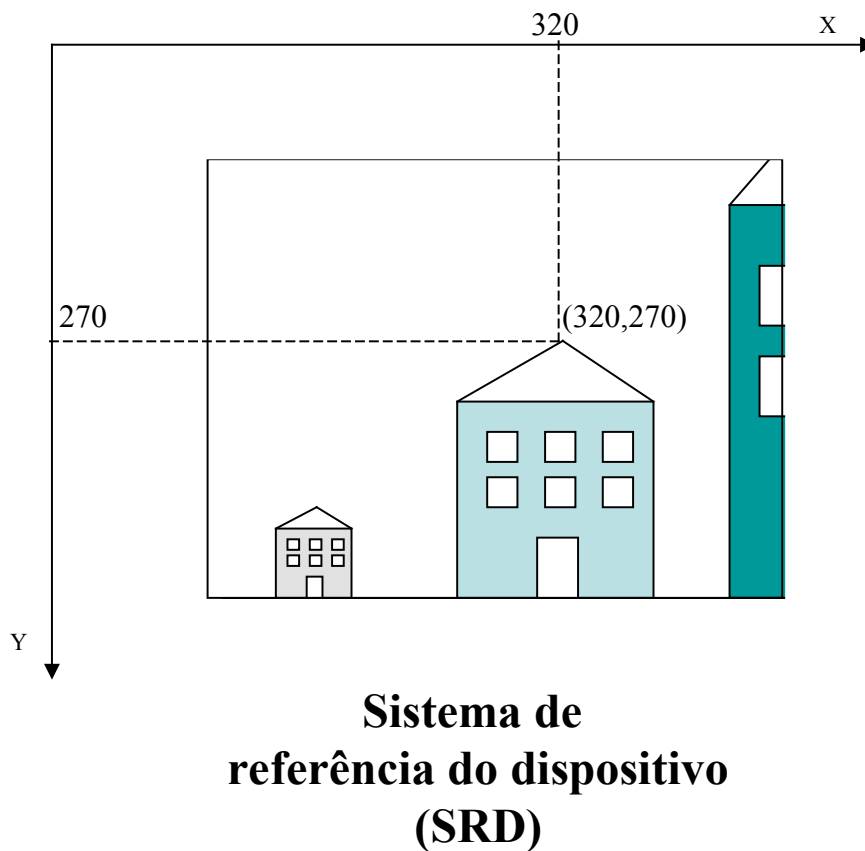
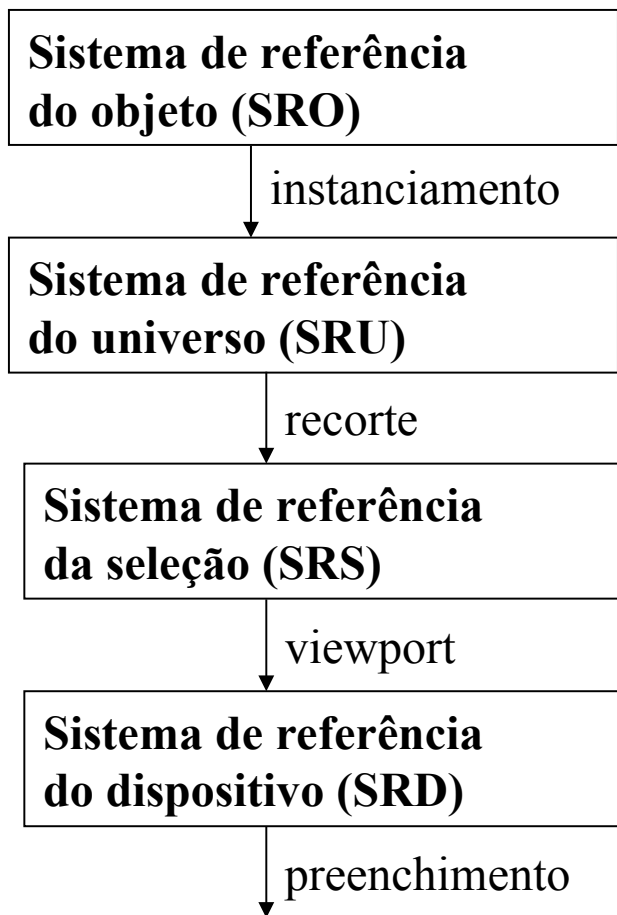


# Etapas



**Sistema de  
referência do universo  
(SRU)**

# Etapas

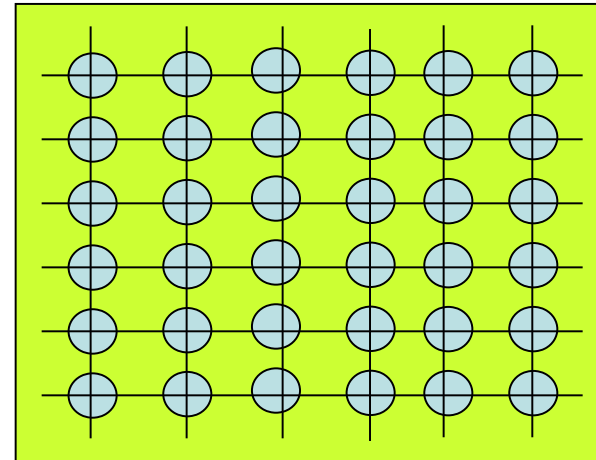
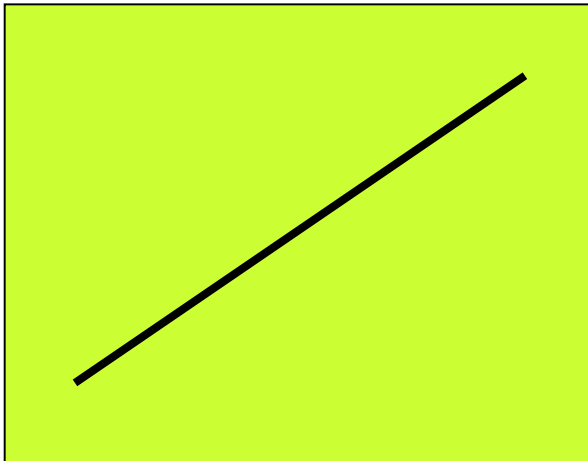


# Algoritmos

- Varredura
- Recorte

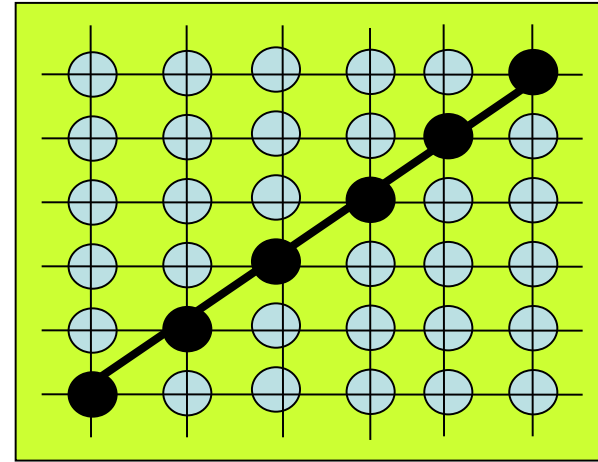
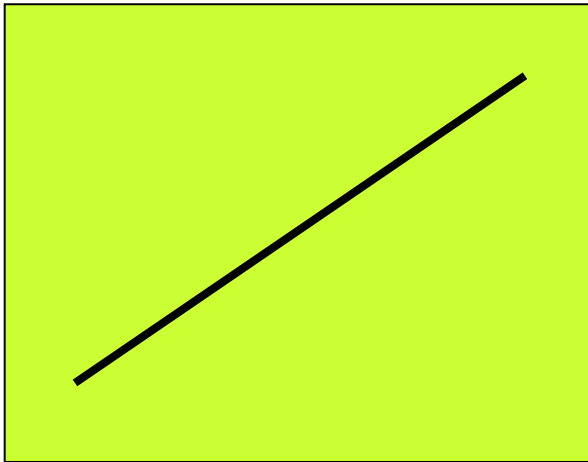
# Algoritmo de Varredura

## Rasterização



# Algoritmo de Varredura

## Rasterização



# Algoritmo de Varredura

## Desenho de linhas

- Algoritmo Incremental (DDA)
- Algoritmo Bresenham

# Algoritmo de Varredura

- Algoritmo incremental

(DDA - Digital Differential Analyzer)

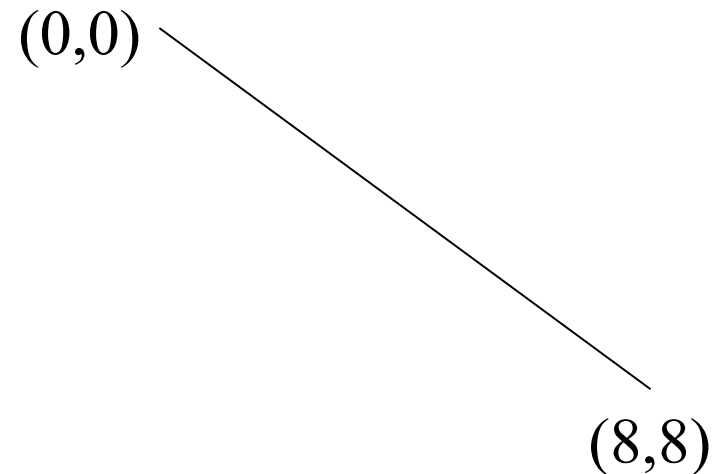
- Método para resolver equações diferenciais através de métodos numéricos.
- Sucessivas operações de incremento baseado no ponto atual.
- Lento pois utiliza **floor** (arredonda para inteiro inferior ou igual) ou **round** (arredonda para o o inteiro mais próximo) dependendo da implementação.

# Algoritmo de Varredura

- Método incremental - DDA

Calcular os pontos

```
Dx=xf-xi;  
Dy=yf-yi;  
m=Dy/Dx;  
y=yi;  
For (x=xi;x<=xf;x++) {  
    draw(x,int(floor(y+0.5), color);  
    y+=m;  
}
```



Resposta

(0,0);(1,1);(2,2);(3,3);(4,4);(5,5);(6,6);(7,7);(8,8)

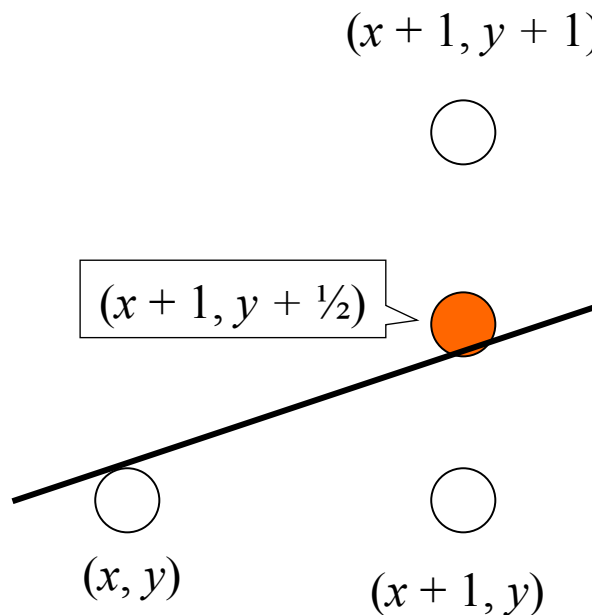


# Algoritmo de Varredura

- Algoritmo do Ponto Médio – Bresenham
  - Atrativo porque usa somente operações aritméticas (não usa round ou floor)
  - É incremental
- Idéia 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  $(x + 1, y + \frac{1}{2})$

# Algoritmo de Varredura

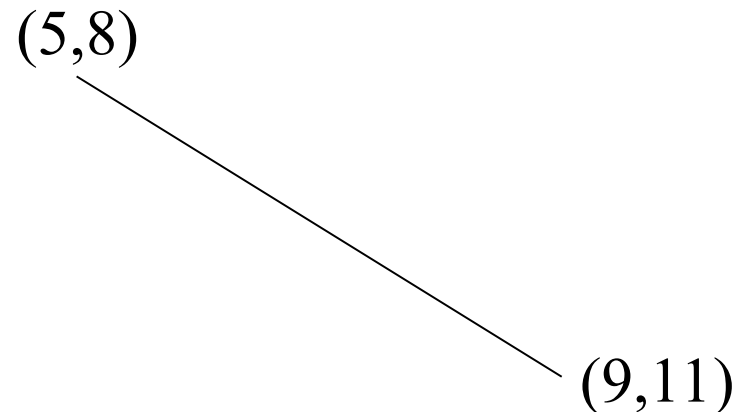
- Algoritmo do Ponto Médio – Bresenham



# Algoritmo do Ponto Médio - Resumo

```
dx = xf - xi;  
dy = yf - yi;  
d = dy * 2 - dx;  
x = xi;  
y = yi;  
while (x <= xf)  
{  
    draw(x, y, color);  
    x++;  
    if(d <= 0)  
        d += dy * 2;  
    else  
    {  
        d += (dy - dx) * 2;  
        y++;  
    }  
}
```

**Calcular os pontos**

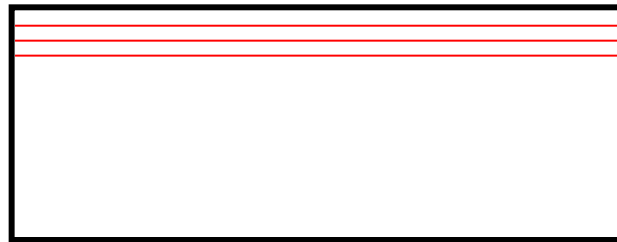


**Resposta**

**(5,8);(6,9);(7,9);(8,10);(9,11)**

# Algoritmos de Preenchimento

- Retângulo



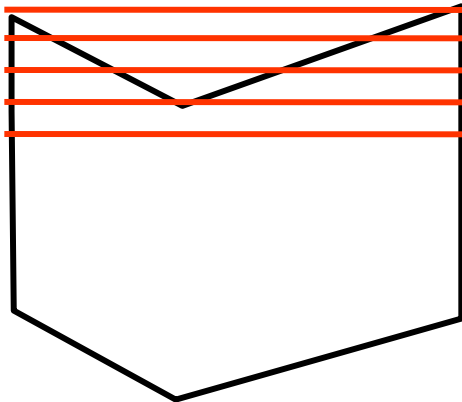
```
for(y=ymin; y<=ymax; y++)  
    for(x=xmin, x<=xmax; x++)  
        draw(x,y,color);
```

# Algoritmos de Preenchimento

- Polígonos

## Scanline

### Passos

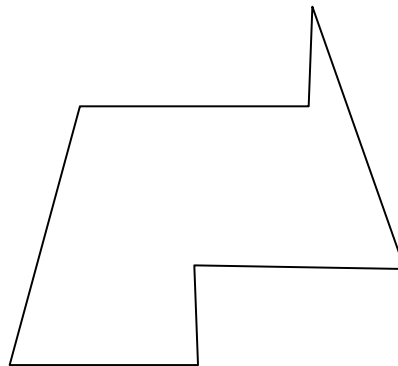


- 1) Encontrar as intersecções da scan line com as arestas do polígono
- 2) Ordenar as intersecções
- 3) Preencher os pixels entre 2 intersecções (regra de paridade que inicia em par, muda quando encontra uma intersecção e escreve quando é ímpar)

**Implemente !**

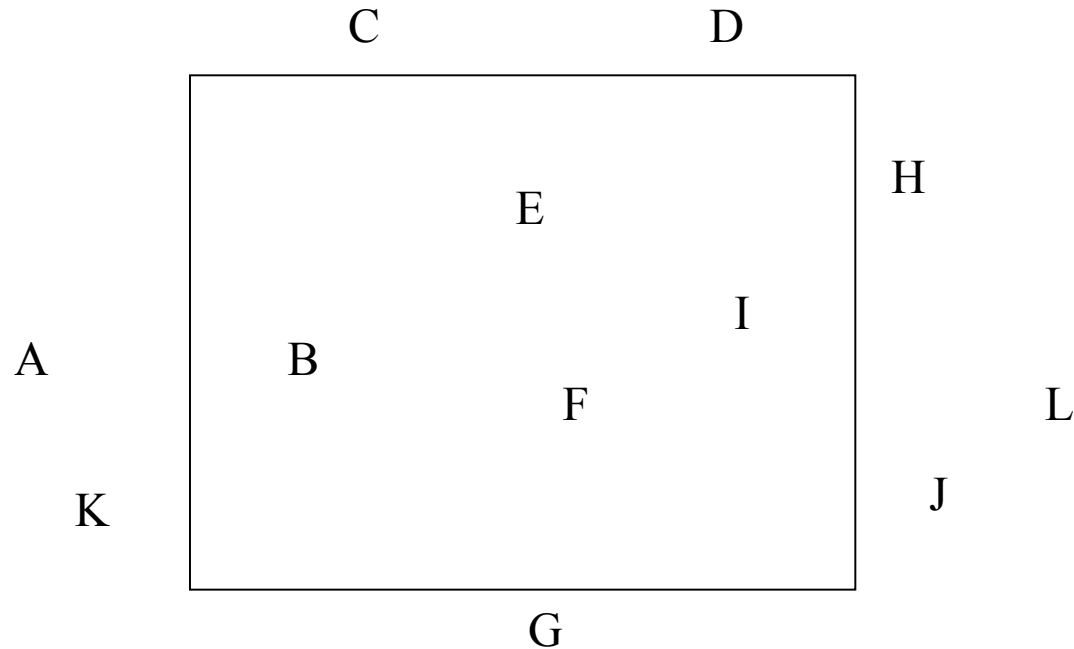
# Algoritmos de Preenchimento

- Problemas ?!?
  - Arestas horizontais



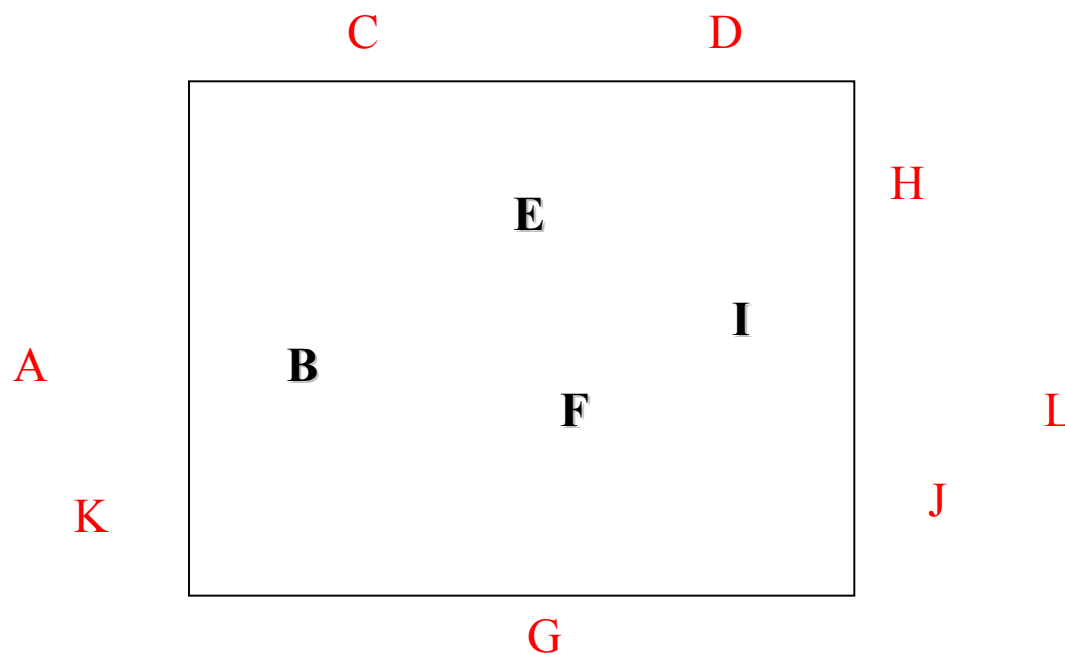
# Algoritmos de Recorte

Pontos



# Algoritmos de Recorte

Pontos

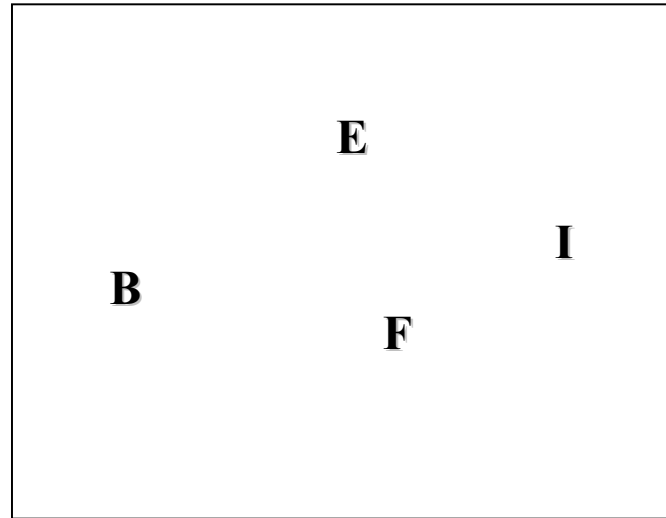


$$\begin{aligned} x_i &\leq x \leq x_f \\ &e \\ y_i &\leq y \leq y_f \end{aligned}$$



# Algoritmos de Recorte

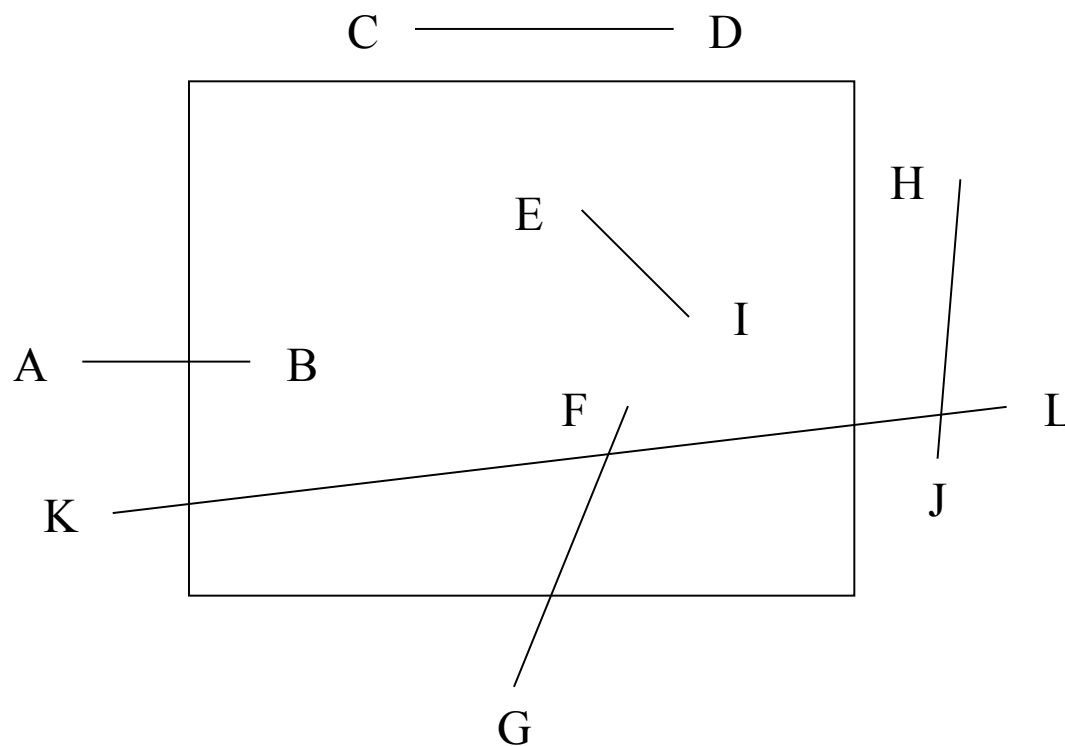
Pontos



Recorte pronto

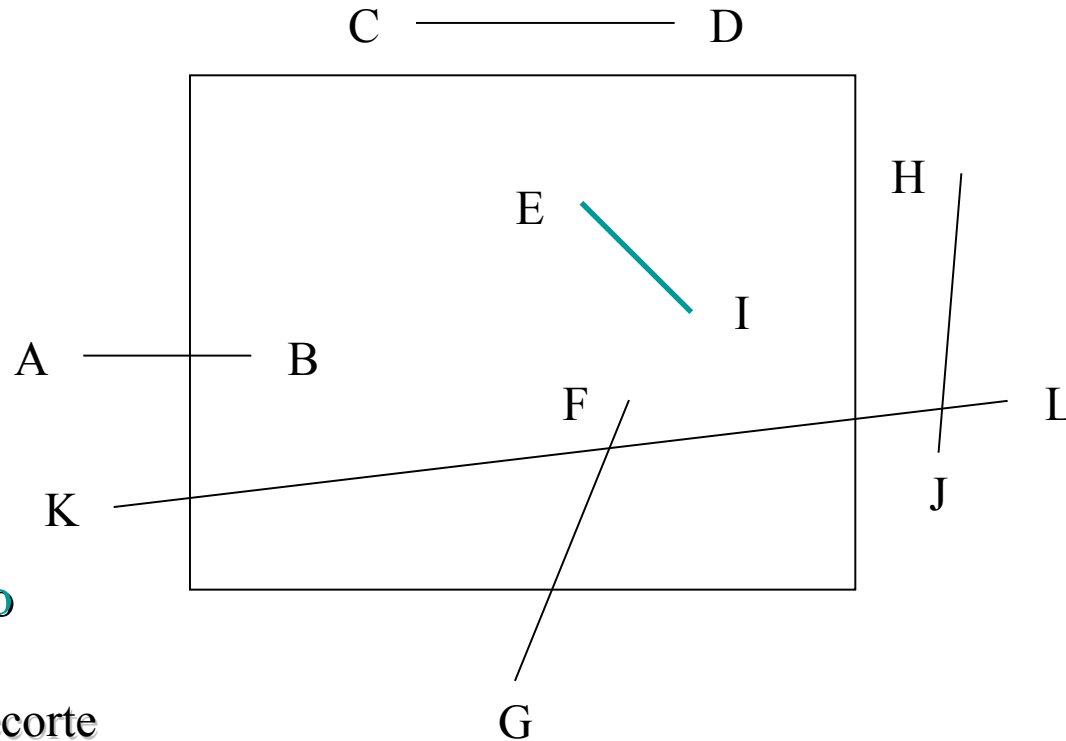
# Algoritmos de Recorte

Linhas



# Algoritmos de Recorte

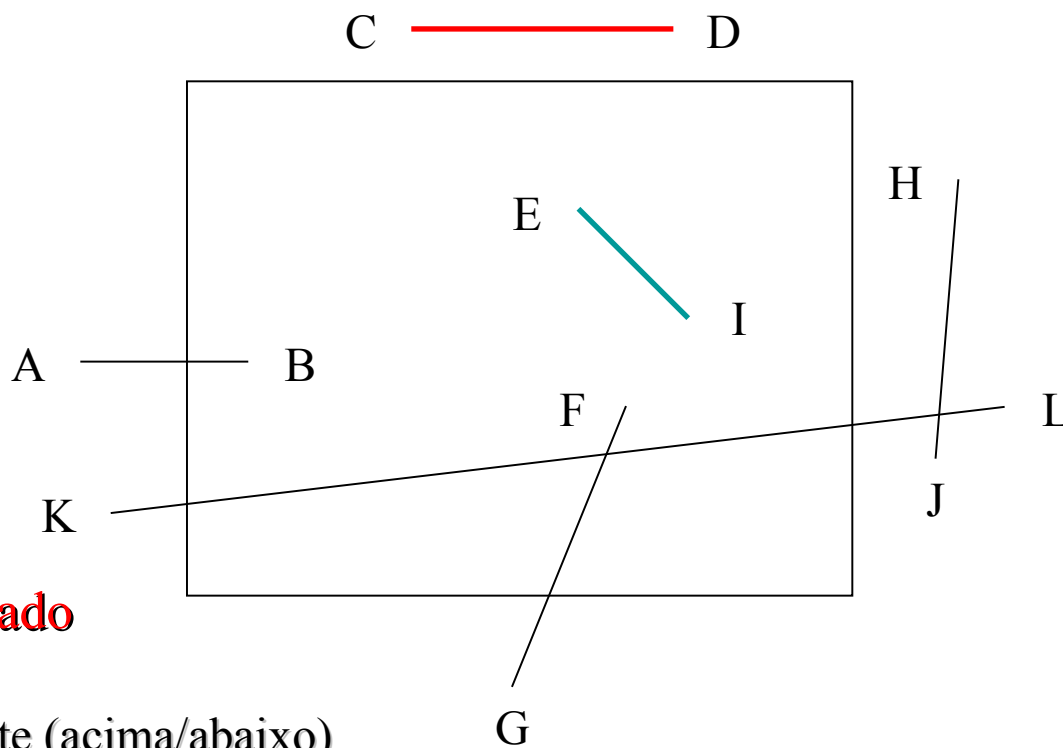
# Linhas



## Trivialmente aceito

- Pontos dentro do recorte

# Linhas



**Trivialmente recusado**

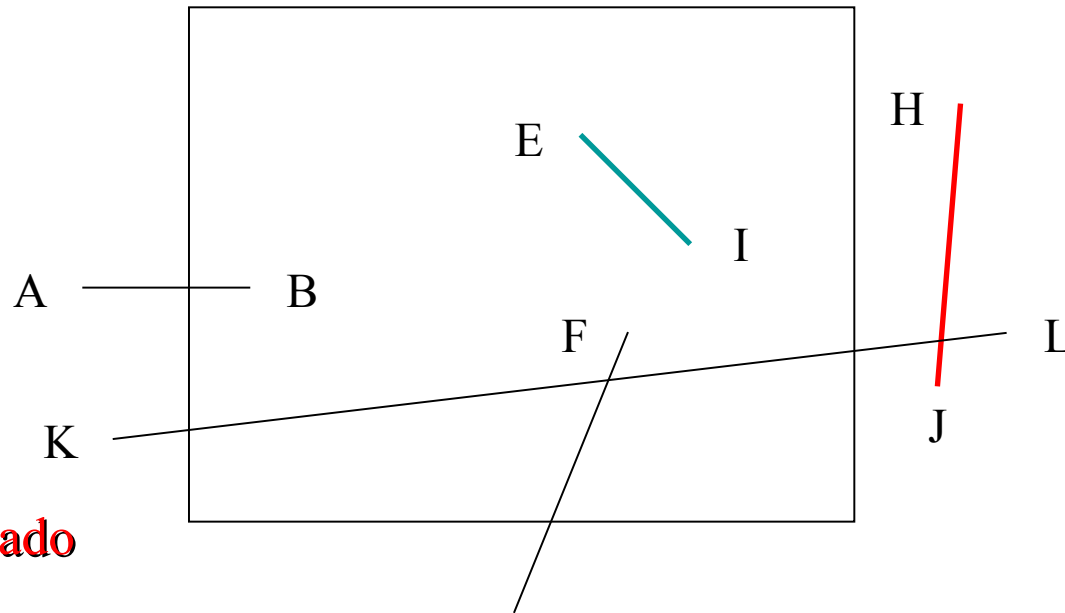
### Pontos fora do recorte (acima/abaixo)

$$C_y < y_i$$

$$D_y < yi$$

# Algoritmos de Recorte

# Linhas



## Trivialmente recusado

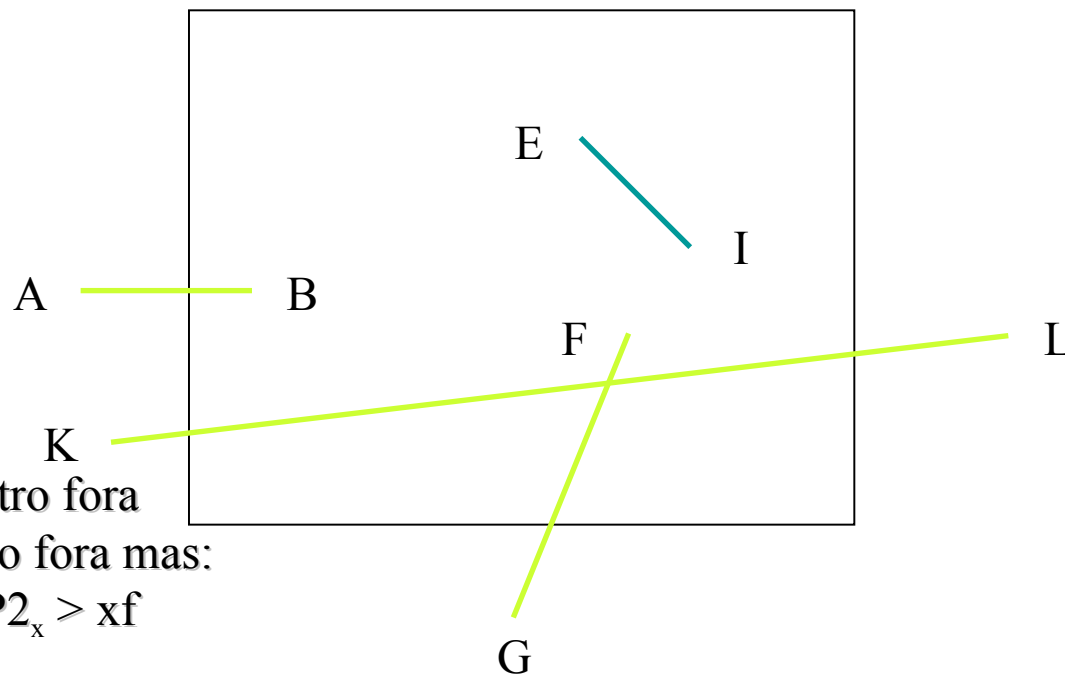
Pontos fora do recorte (esquerda/direita)

$$H_x > xf$$

$$\mathbf{J}_x > \mathbf{x}f$$

# Algoritmos de Recorte

## Linhas



### Recorte

Um ponto dentro outro fora

- Os dois pontos estão fora mas:

$$P1_x < x_i \text{ e } P2_x > x_f$$

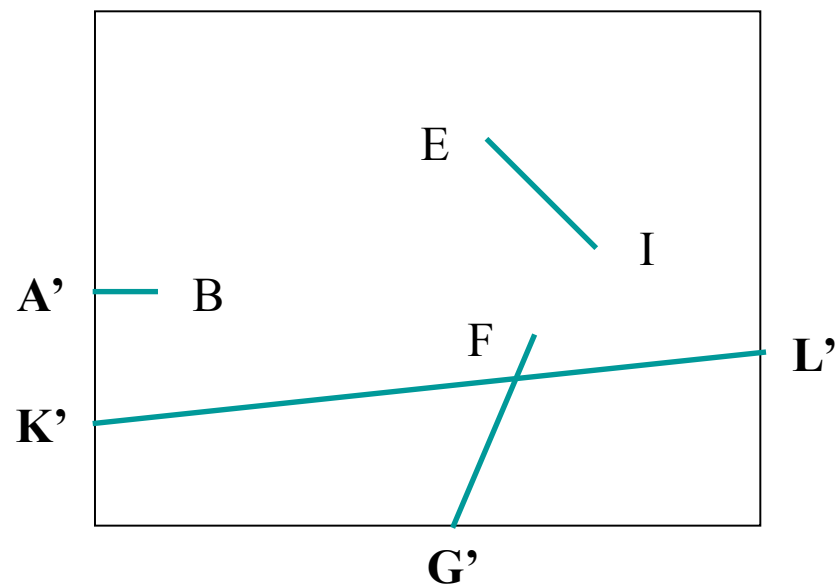
ou

$$P1_y < y_i \text{ e } P2_y > y_f$$

Ou ...

# Algoritmos de Recorte

Linhas



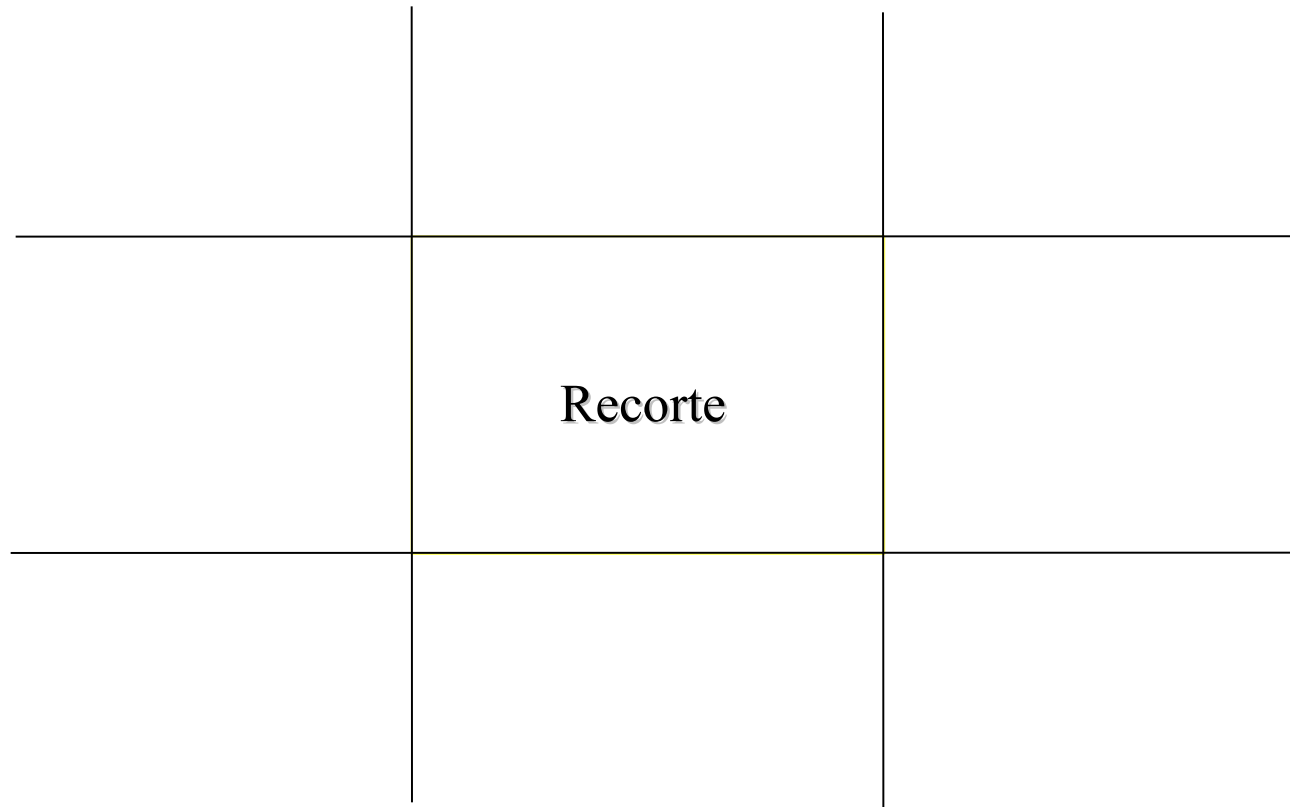
Recorte pronto

# Algoritmo de Cohen-Sutherland

- Dividir a área em nove partes.



# Algoritmo de Cohen-Sutherland



# Algoritmo de Cohen-Sutherland

- Atribuir códigos de 4 bits às regiões definidas pelas bordas da janela de visualização.
  - Bit 0: **esquerda**
  - Bit 1: **direita**
  - Bit 2: **abaixo**
  - Bit 3: **acima**

# Algoritmo de Cohen-Sutherland

1001	1000	1010
0001	0000	0010
0101	0100	0110

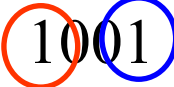





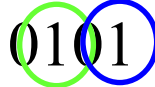


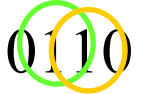

# Algoritmo de Cohen-Sutherland

Bit 0: **esquerda**

Bit 1: **direita**

Bit 2: **abaixo**

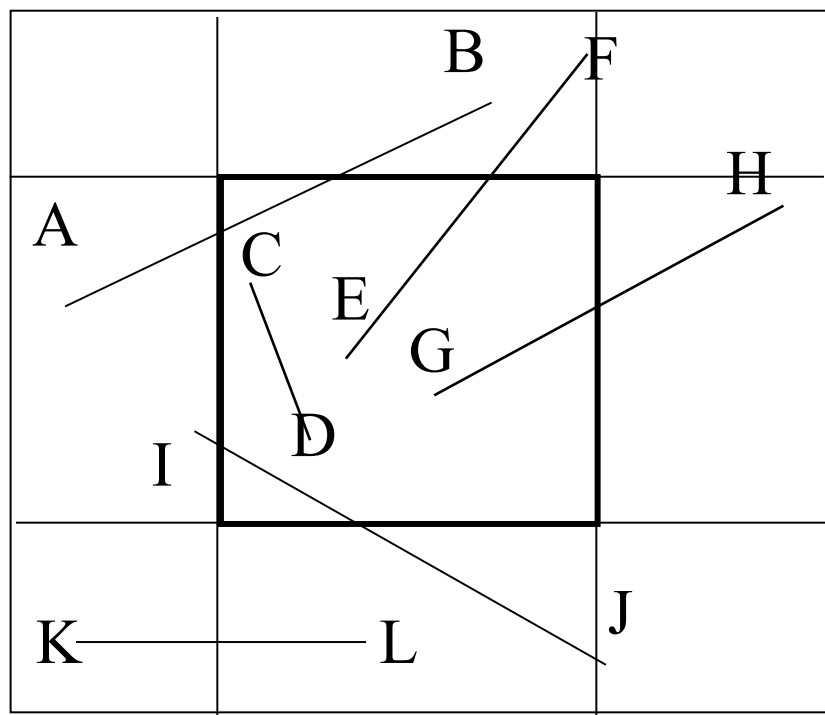
Bit 3: **acima**

 1001	 1000	  1010
000  1	0000	00  10
  101	 0100	  0110

# Algoritmo de Cohen-Sutherland

- $\text{if}(y > y_f) \rightarrow$  seta primeiro bit em 1
- $\text{if}(y < y_i) \rightarrow$  seta segundo bit em 1
- $\text{if}(x > x_f) \rightarrow$  seta terceiro bit em 1
- $\text{if}(x < x_i) \rightarrow$  seta quarto bit em 1

# Algoritmo de Cohen-Sutherland



Bit codes dos pontos:

A = 0001

B = 1000

C = 0000

D = 0000

E = 0000

F = 1000

G = 0000

H = 0010

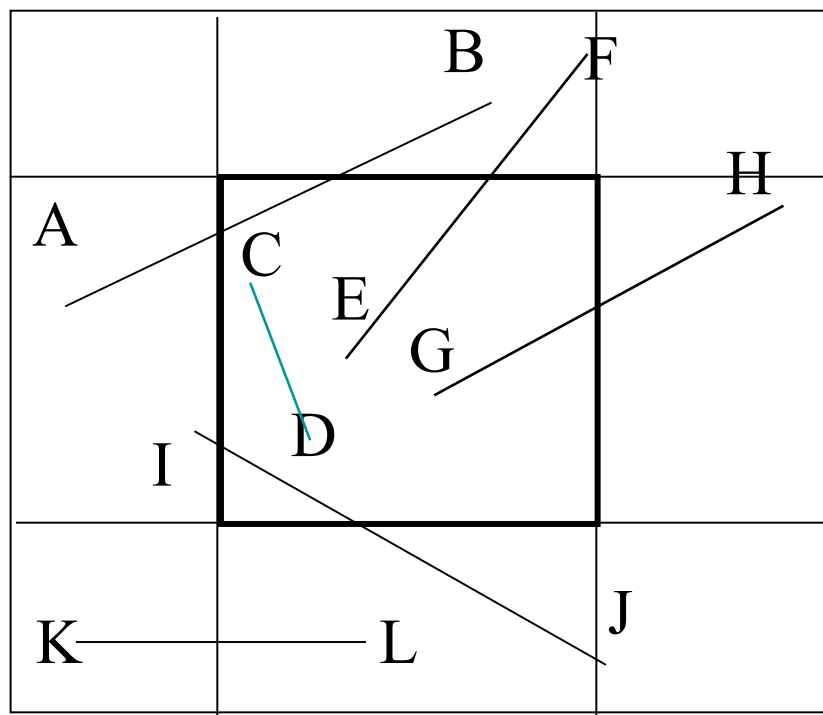
I = 0001

J = 0110

K = 0101

L = 0100

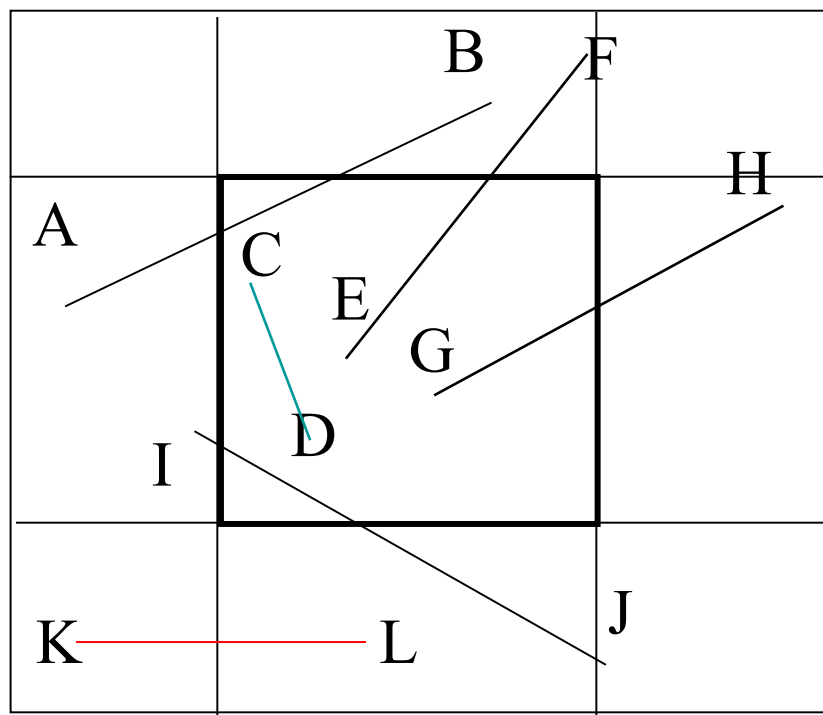
# Algoritmo de Cohen-Sutherland



Como devem ser os bit codes  
dos pontos trivialmente aceitos?

$P1 \mid P2 = 0000$

# Algoritmo de Cohen-Sutherland

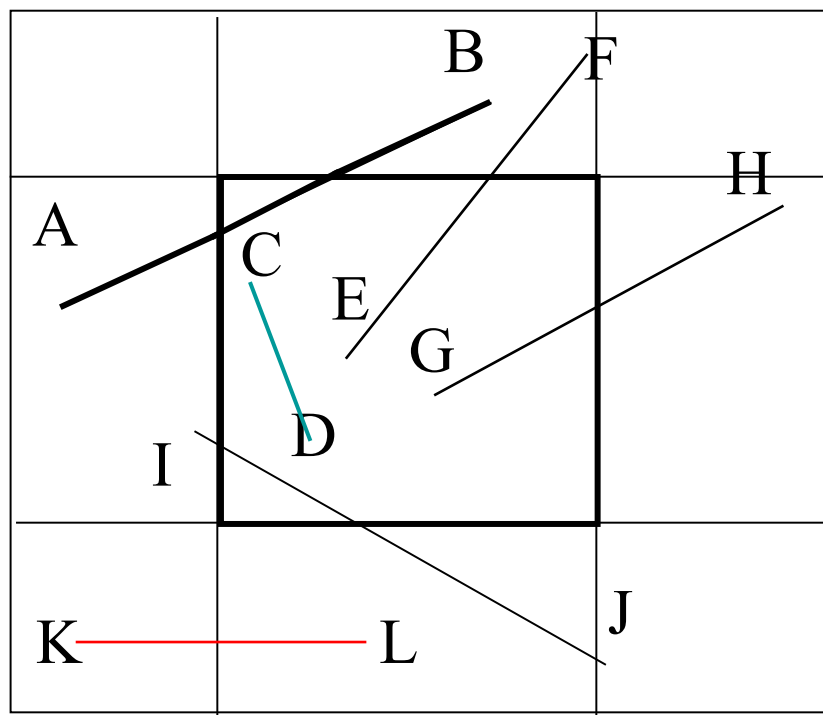


Como devem ser os bit codes dos pontos trivialmente recusados?

$(P1 \& P2) \neq 0000$

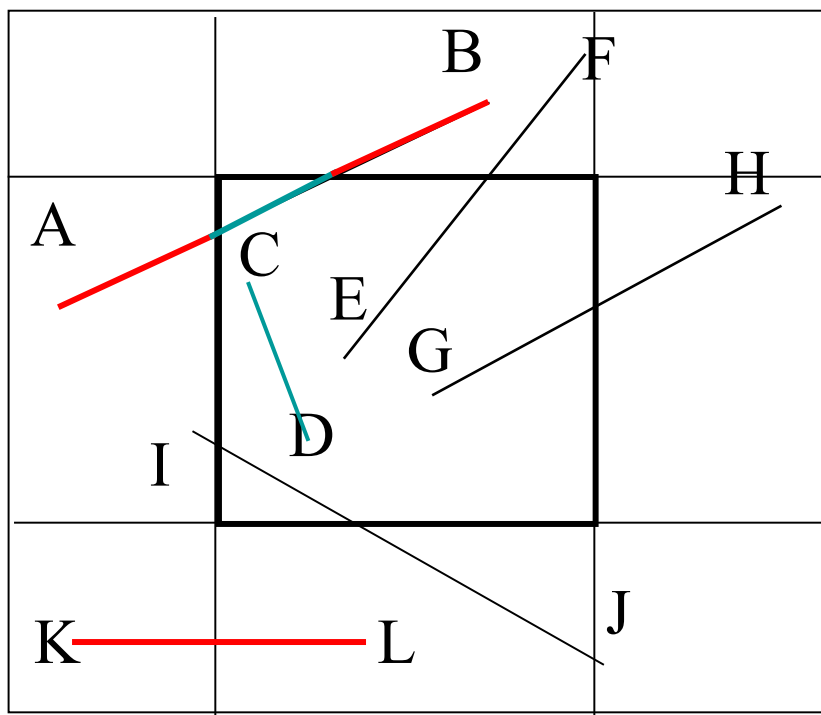


# Algoritmo de Cohen-Sutherland



O que fazer com os que não  
são nem aceitos nem recusados?

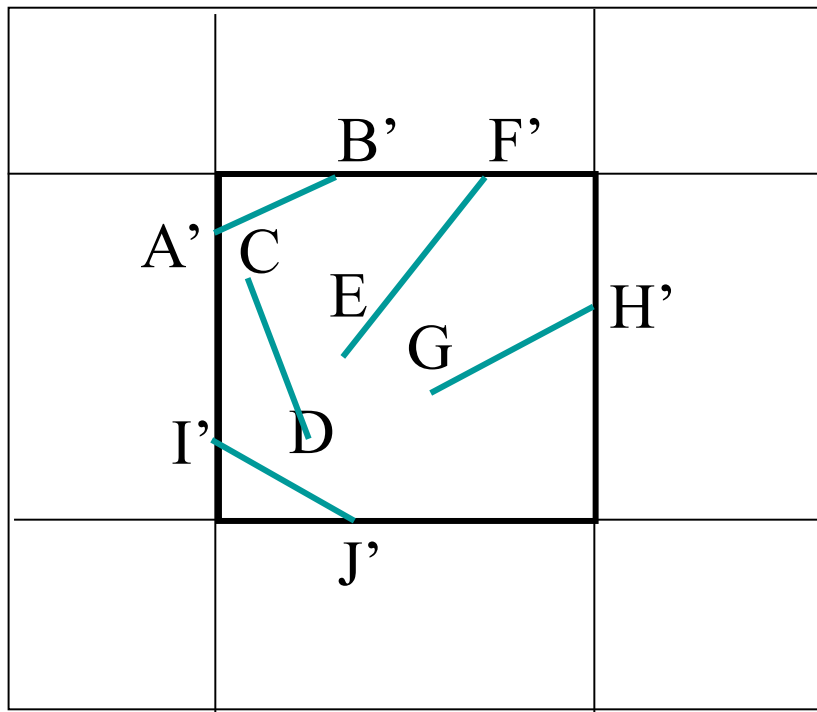
# Algoritmo de Cohen-Sutherland



O que fazer com os que não  
são nem aceitos nem recusados?

Calcular segmentos através das  
intersecções.

# Algoritmo de Cohen-Sutherland



# Algoritmo de Cohen-Sutherland

Exemplo