

Computação Gráfica

Aula 24: Rasterização

Vicente Helano Feitosa Batista Sobrinho
Faculdade Paraíso do Ceará
Sistemas de Informação
1o. semestre de 2011

Renderização

Dispositivos *raster* predominam no mercado



LCD



CRT

Renderização

Dispositivos *raster* predominam no mercado

Fonte: http://en.wikipedia.org/wiki/Cowboys_Stadium



Tela de LCD com 49 × 22 metros, resolução de 1.088 × 2.432 pixels.
Estádio dos Cowboys, Texas.

Renderização

O conteúdo de uma tela raster é dado por um arranjo bidimensional de *pixels* (picture elements)



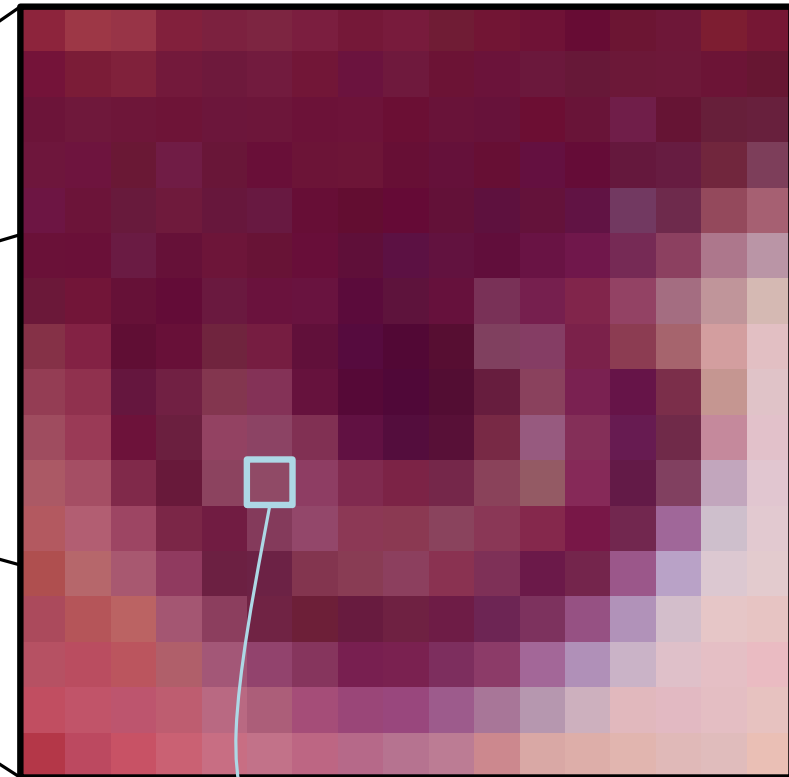
Fonte: <http://en.wikipedia.org/wiki/Lenna>

Renderização

O conteúdo de uma tela raster é dado por um arranjo bidimensional de *pixels* (picture elements)



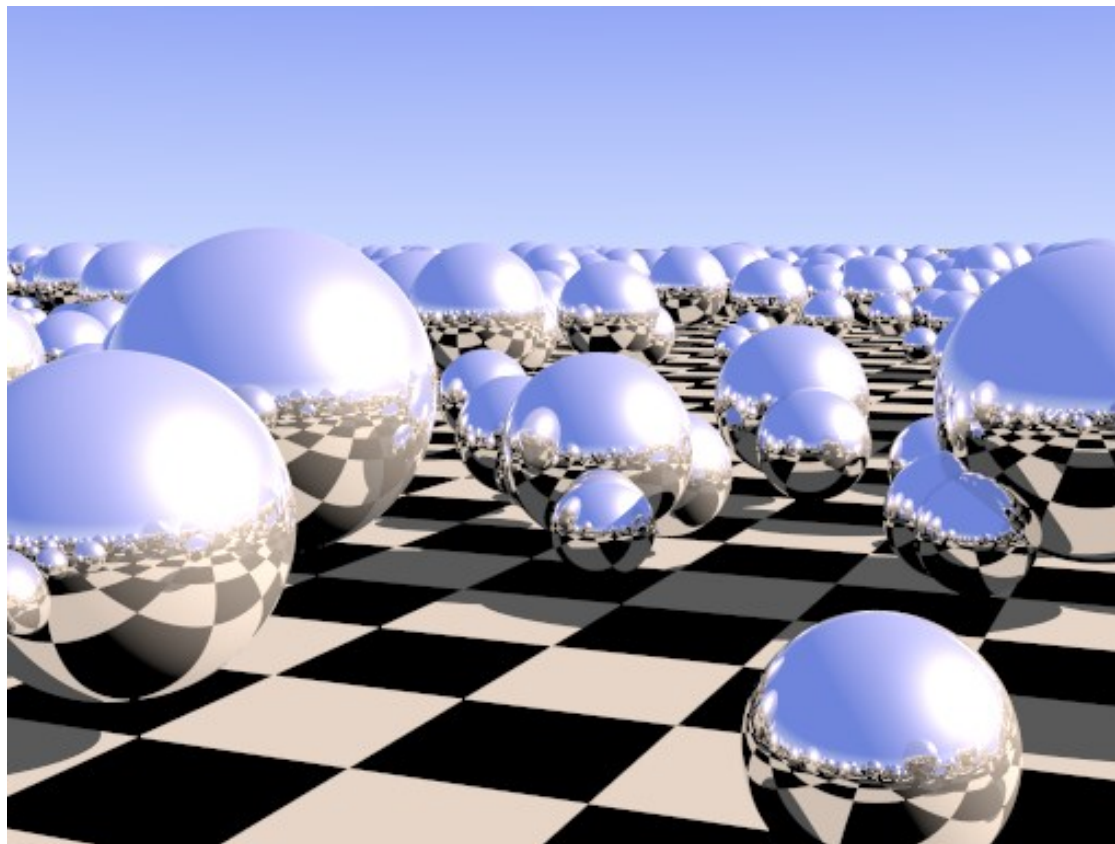
Fonte: <http://en.wikipedia.org/wiki/Lenna>



1 pixel

Renderização

É um processo de construção de uma imagem bidimensional a partir de uma cena definida por uma câmera fictícia, objetos tridimensionais, modelo de iluminação, texturas, etc.



Métodos clássicos de renderização

- Traçado de raios (*ray tracing*)
- Rasterização (*rasterization*)

Métodos clássicos de renderização

Traçado de raios	Rasterização
Elevado grau de realismo	Aproxima-se da realidade

Métodos clássicos de renderização

Traçado de raios	Rasterização
Elevado grau de realismo	Aproxima-se da realidade
Custo elevado	Rápido

Métodos clássicos de renderização

Traçado de raios	Rasterização
Elevado grau de realismo	Aproxima-se da realidade
Custo elevado	Rápido
Aplicações não-interativas	Ambientes interativos

Métodos clássicos de renderização

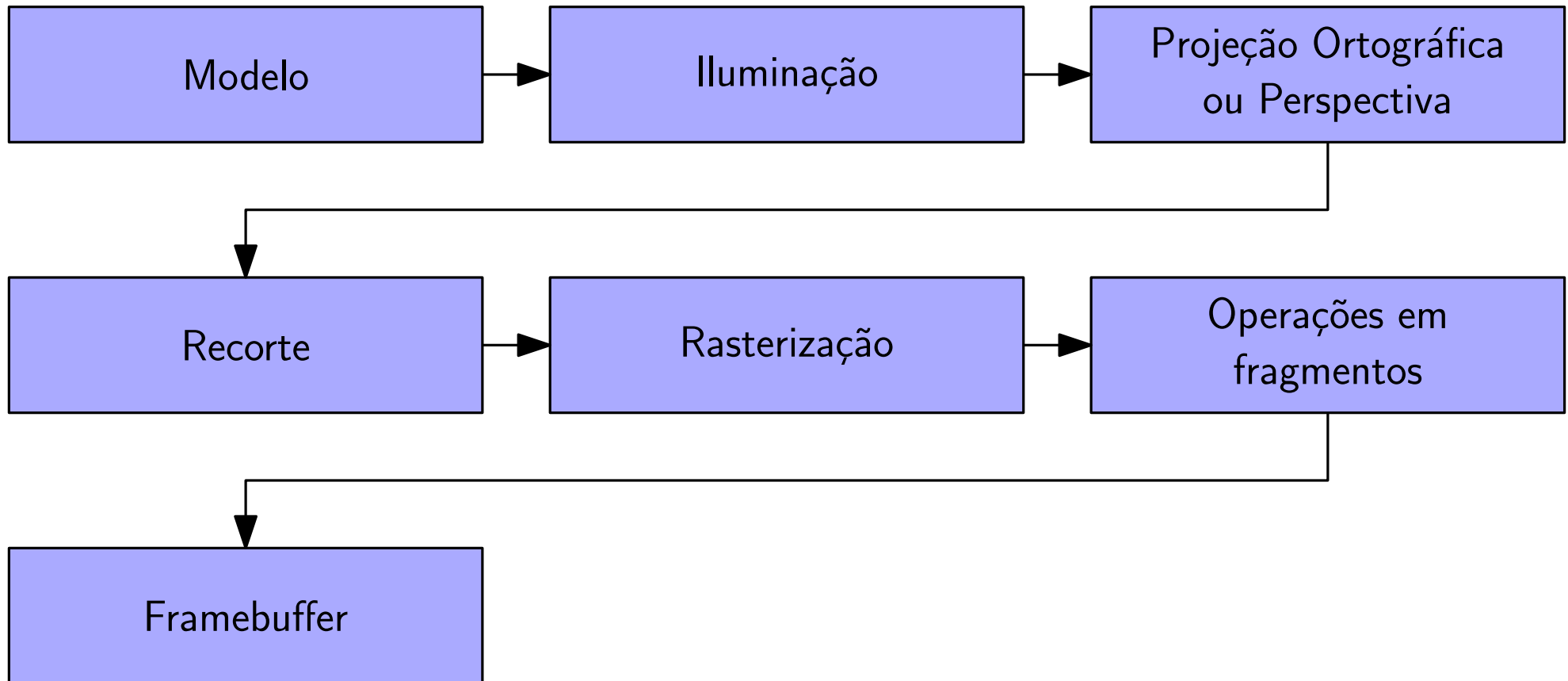
Traçado de raios	Rasterização
Elevado grau de realismo	Aproxima-se da realidade
Custo elevado	Rápido
Aplicações não-interativas	Ambientes interativos
Pixel-a-pixel	Objeto-a-objeto

Métodos clássicos de renderização

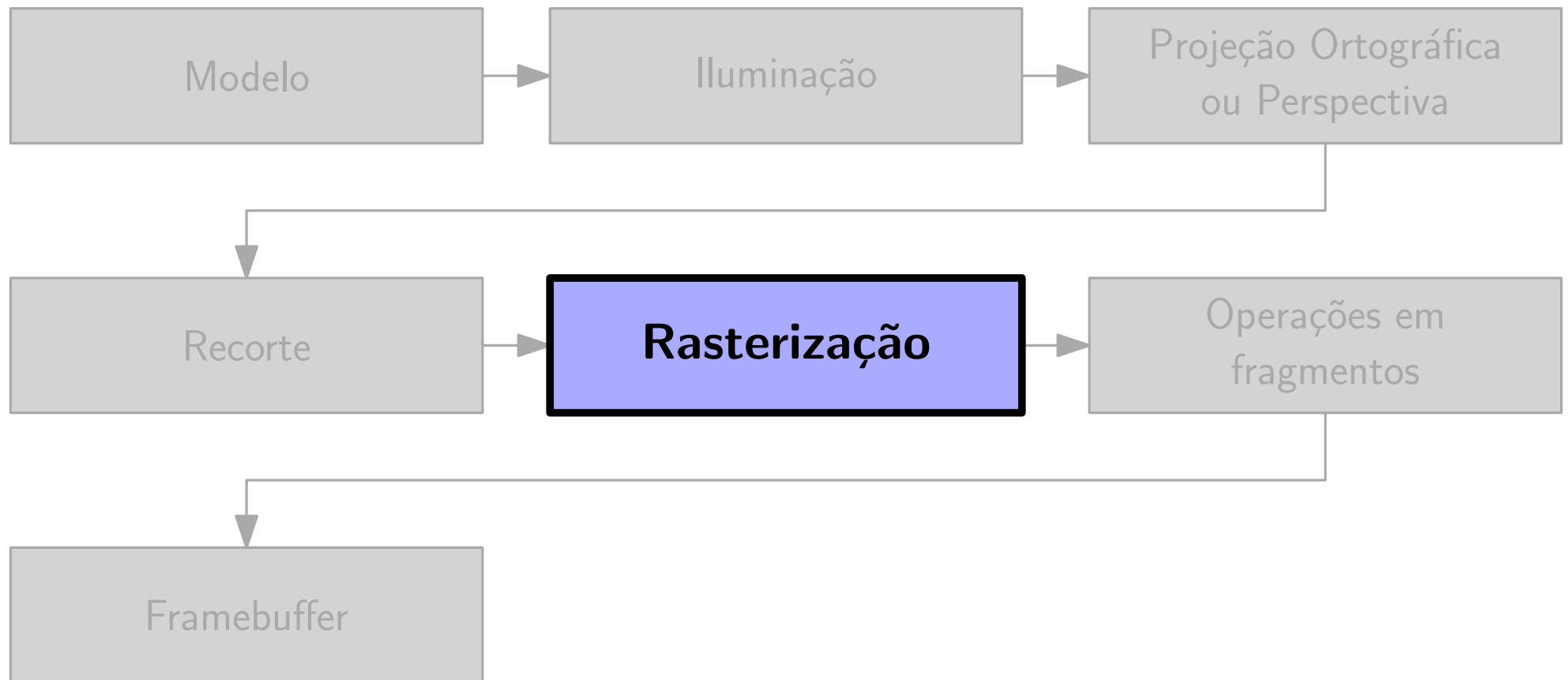
Abordado neste curso

Traçado de raios	Rasterização
Elevado grau de realismo	Aproxima-se da realidade
Custo elevado	Rápido
Aplicações não-interativas	Ambientes interativos
Pixel-a-pixel	Objeto-a-objeto

Pipeline gráfico (básico) de renderização por varredura



O que veremos hoje?



Entrada:

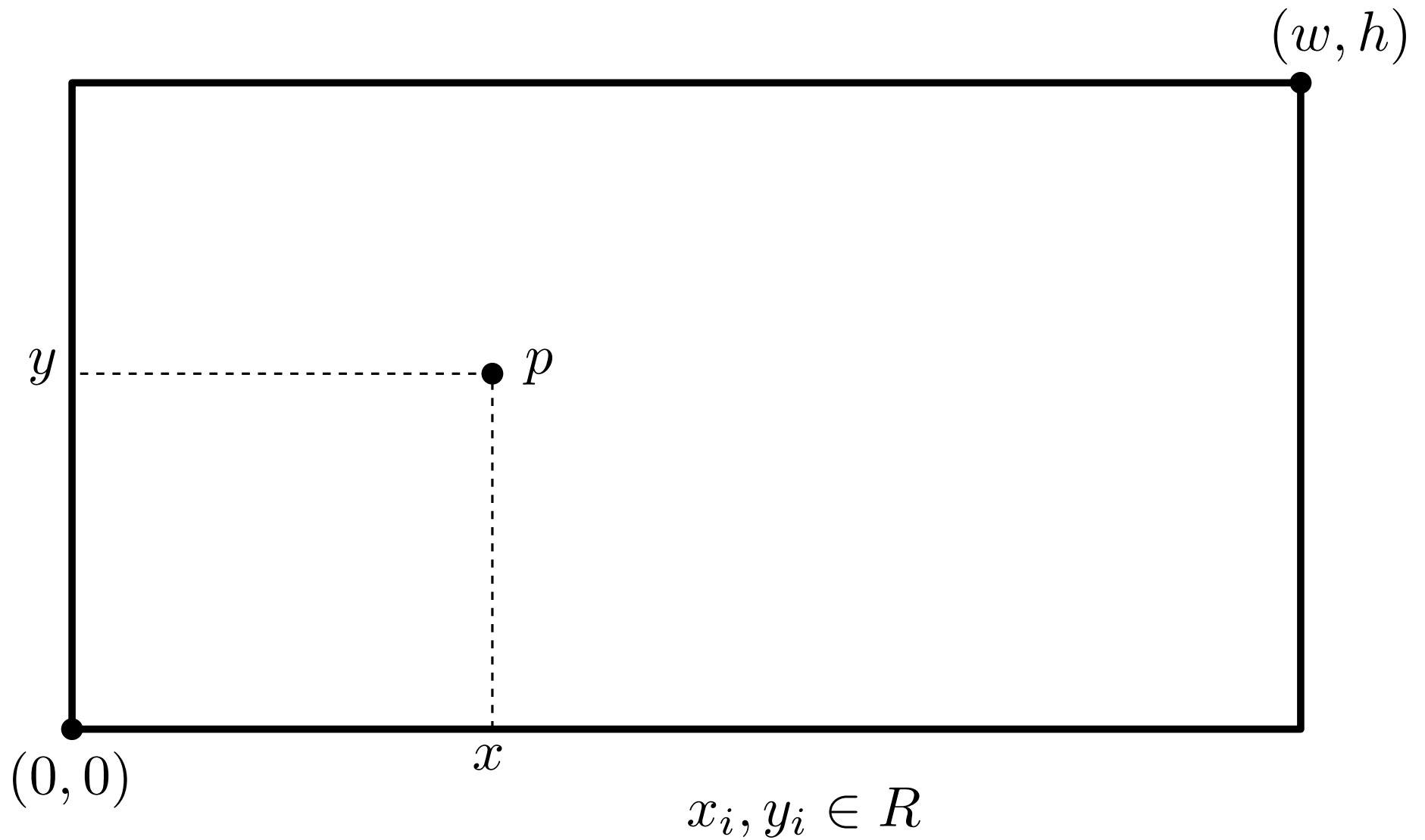
- Primitivas 2D no espaço de tela

Saída

- Coleção de fragmentos de pixels a serem pintados junto com os atributos (cor, profundidade, etc.) interpolados para cada pixel

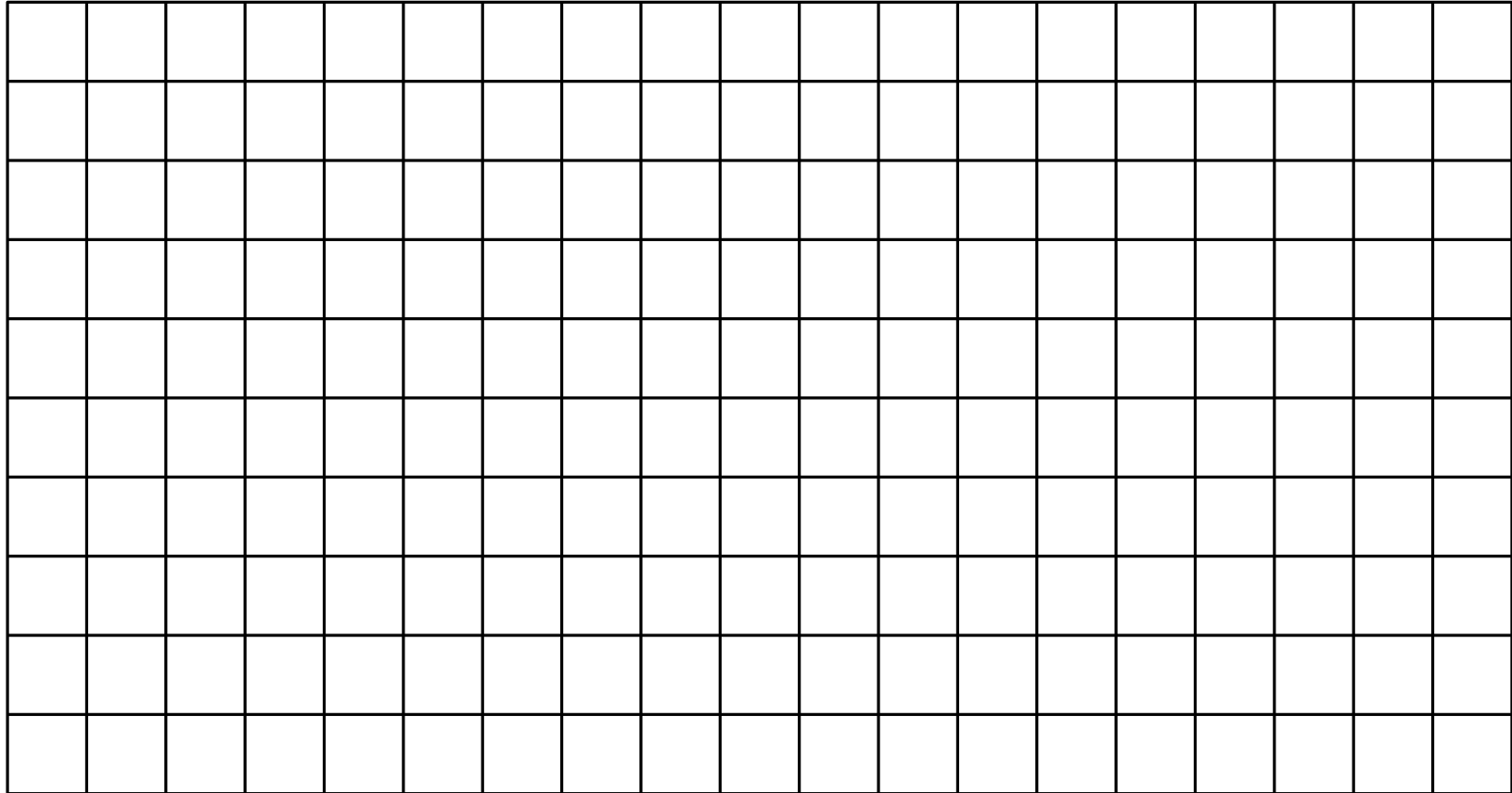
O problema de rasterização

Coordenadas da tela

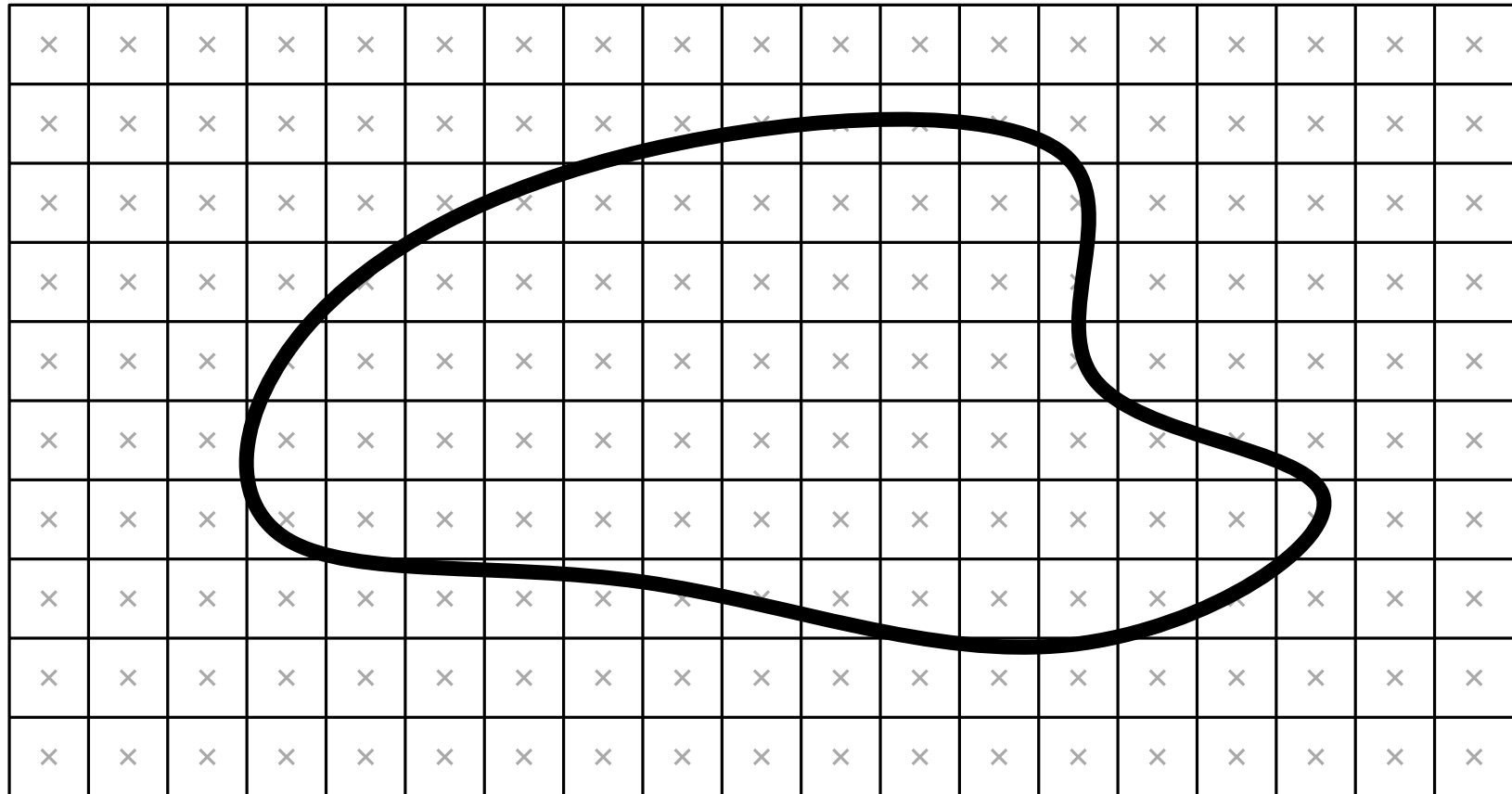


O problema de rasterização

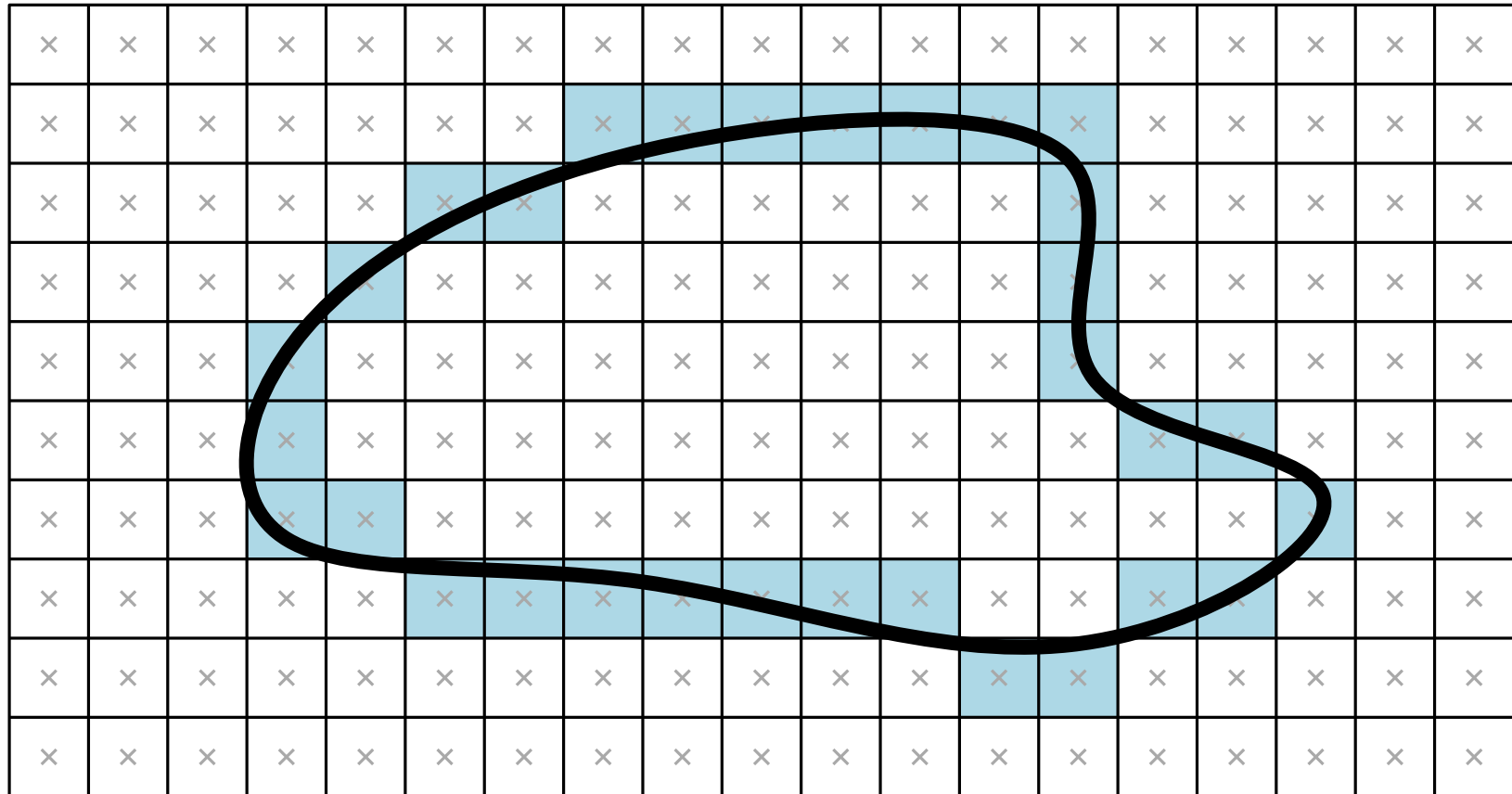
Reticulado de pixels subjacentes



Exemplo de rasterização



Exemplo de rasterização



Exemplo de rasterização

x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Rasterização de segmentos de reta

Dado um segmento de reta definido por suas extremidades (x_0, y_0) e (x_1, y_1) , espera-se que:

Rasterização de segmentos de reta

Dado um segmento de reta definido por suas extremidades (x_0, y_0) e (x_1, y_1) , espera-se que:

- Os pixels correspondentes às extremidades do segmento sejam sempre selecionados

Rasterização de segmentos de reta

Dado um segmento de reta definido por suas extremidades (x_0, y_0) e (x_1, y_1) , espera-se que:

- Os pixels correspondentes às extremidades do segmento sejam sempre selecionados
- Brilho e espessura uniformes, independentes da orientação e comprimento do segmento

Rasterização de segmentos de reta

Dado um segmento de reta definido por suas extremidades (x_0, y_0) e (x_1, y_1) , espera-se que:

- Os pixels correspondentes às extremidades do segmento sejam sempre selecionados
- Brilho e espessura uniformes, independentes da orientação e comprimento do segmento
- Aparência linear e contínua

Rasterização de segmentos de reta

Dado um segmento de reta definido por suas extremidades (x_0, y_0) e (x_1, y_1) , espera-se que:

- Os pixels correspondentes às extremidades do segmento sejam sempre selecionados
- Brilho e espessura uniformes, independentes da orientação e comprimento do segmento
- Aparência linear e contínua
- Rapidez

Rasterização de segmentos de reta

Dado um segmento de reta definido por suas extremidades (x_0, y_0) e (x_1, y_1) , espera-se que:

- Os pixels correspondentes às extremidades do segmento sejam sempre selecionados
- Brilho e espessura uniformes, independentes da orientação e comprimento do segmento
- Aparência linear e contínua
- Rapidez

Decisão de projeto:

Segmentos devem ser desenhados com espessura igual a 1 pixel

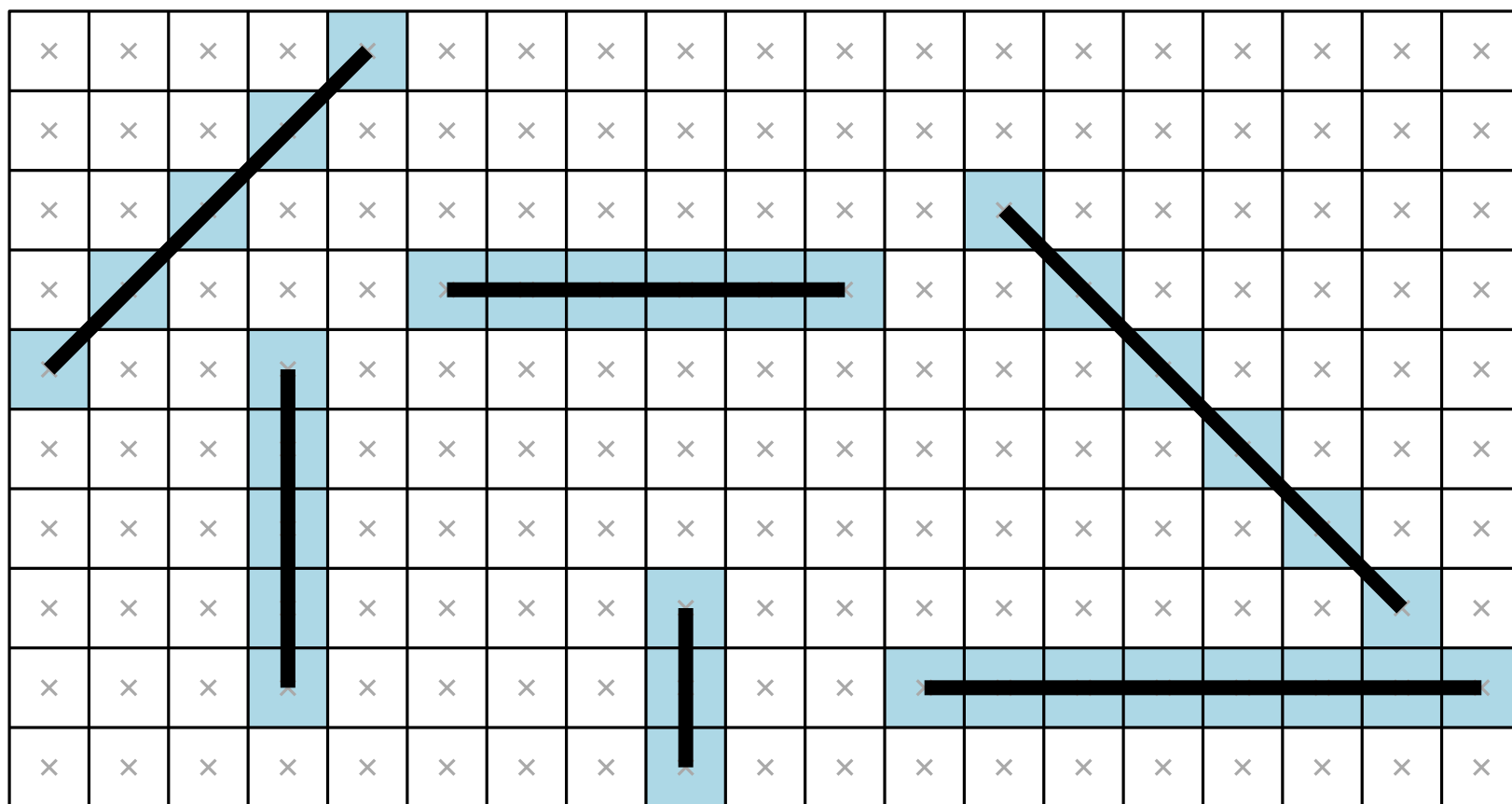
Observação.

Segmentos horizontais, verticais e a 45 graus são trivialmente rasterizados

Rasterização de segmentos de reta

Observação.

Segmentos horizontais, verticais e a 45 graus são trivialmente rasterizados



Observação.

Segmentos horizontais, verticais e a 45 graus são trivialmente rasterizados

Isto é, dada a equação da reta de suporte $y = mx + b$,
 $m = (y_1 - y_0)/(x_1 - x_0)$, temos:

Observação.

Segmentos horizontais, verticais e a 45 graus são trivialmente rasterizados

Isto é, dada a equação da reta de suporte $y = mx + b$, $m = (y_1 - y_0)/(x_1 - x_0)$, temos:

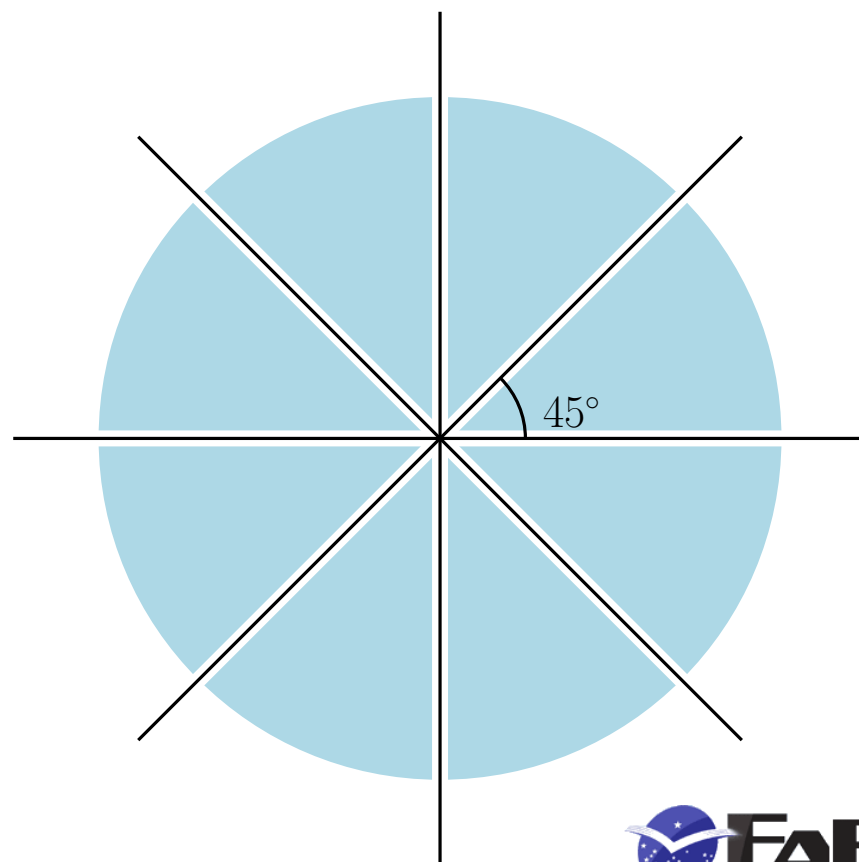
- $m = 0$ ou $m = 1$: trivial

Observação.

Segmentos horizontais, verticais e a 45 graus são trivialmente rasterizados

Isto é, dada a equação da reta de suporte $y = mx + b$, $m = (y_1 - y_0)/(x_1 - x_0)$, temos:

- $m = 0$ ou $m = 1$: trivial
- Caso contrário, temos 8 casos



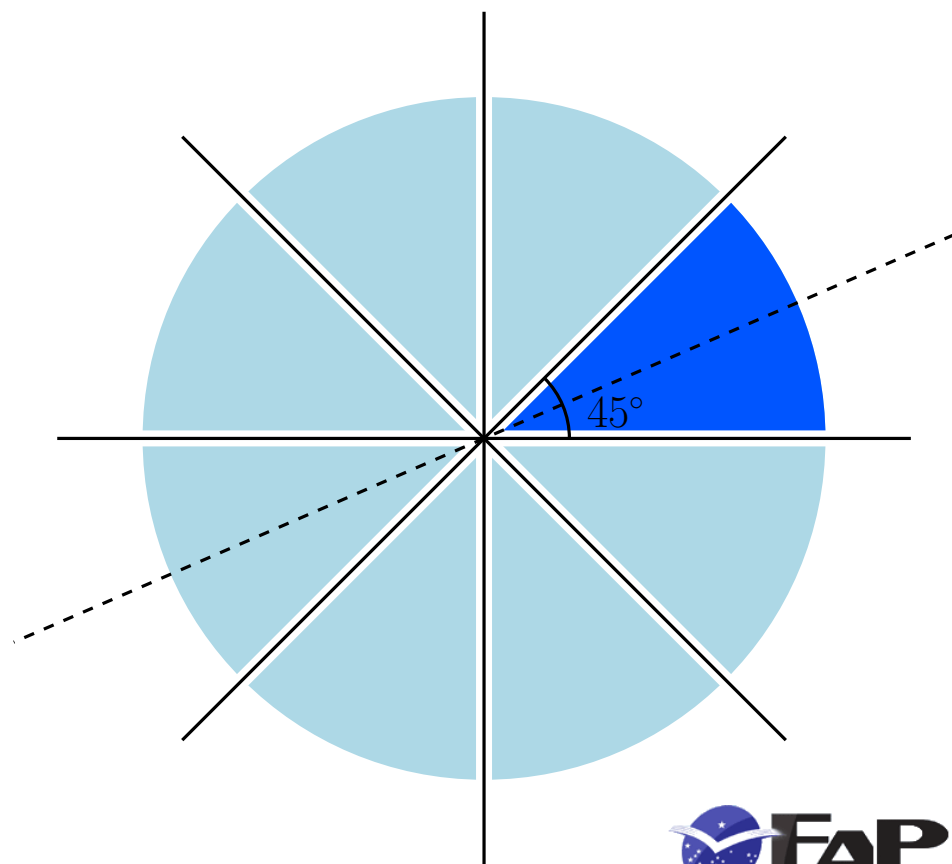
Observação.

Segmentos horizontais, verticais e a 45 graus são trivialmente rasterizados

Isto é, dada a equação da reta de suporte $y = mx + b$, $m = (y_1 - y_0)/(x_1 - x_0)$, temos:

- $m = 0$ ou $m = 1$: trivial
- Caso contrário, temos 8 casos

Basta considerar o 1º octante

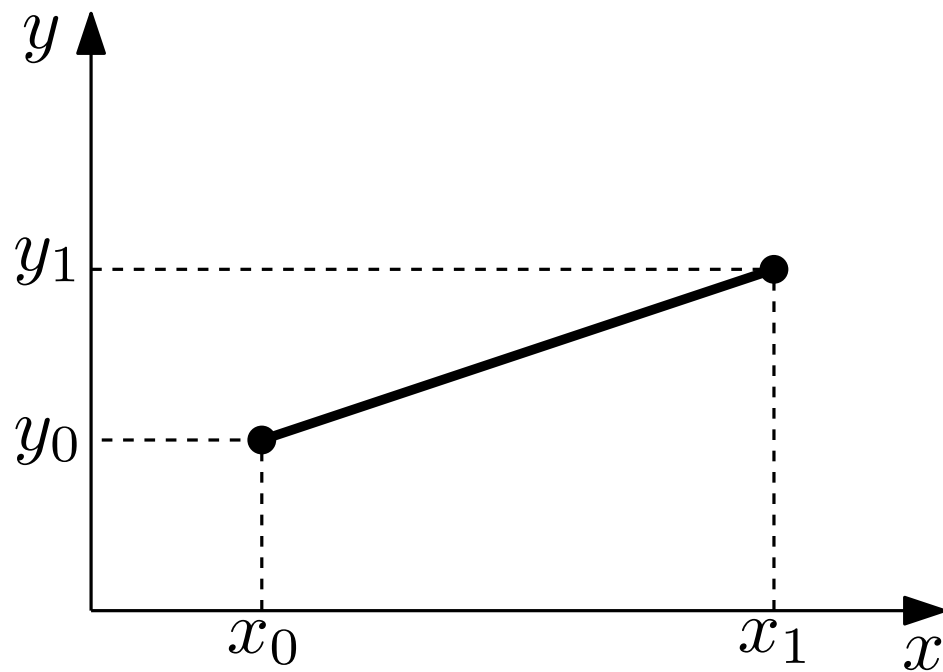


Ideia:

- Incrementar em x e utilizar a equação da reta de suporte para calcular o valor de y

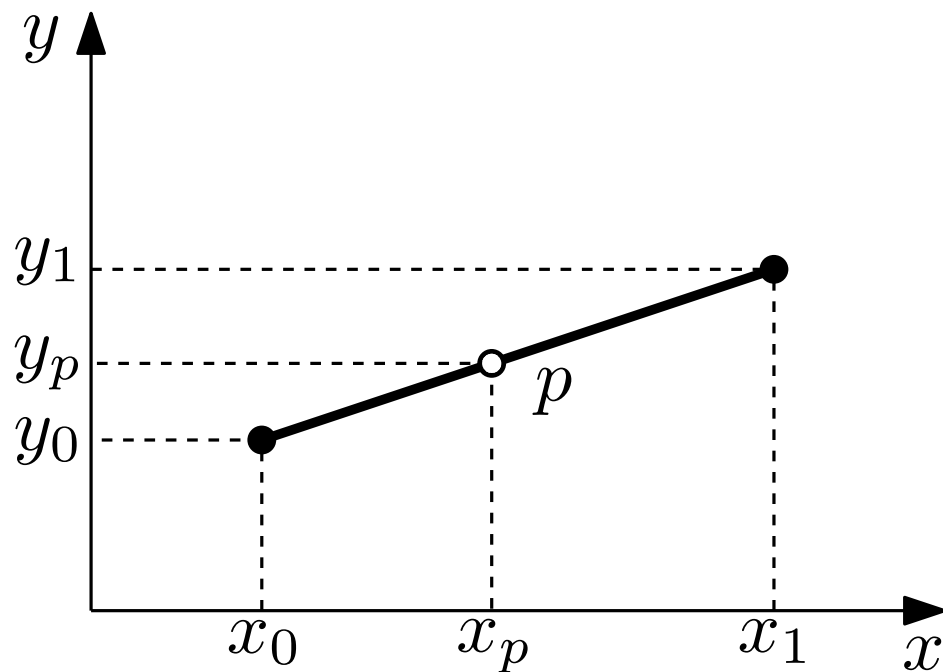
Ideia:

- Incrementar em x e utilizar a equação da reta de suporte para calcular o valor de y



Ideia:

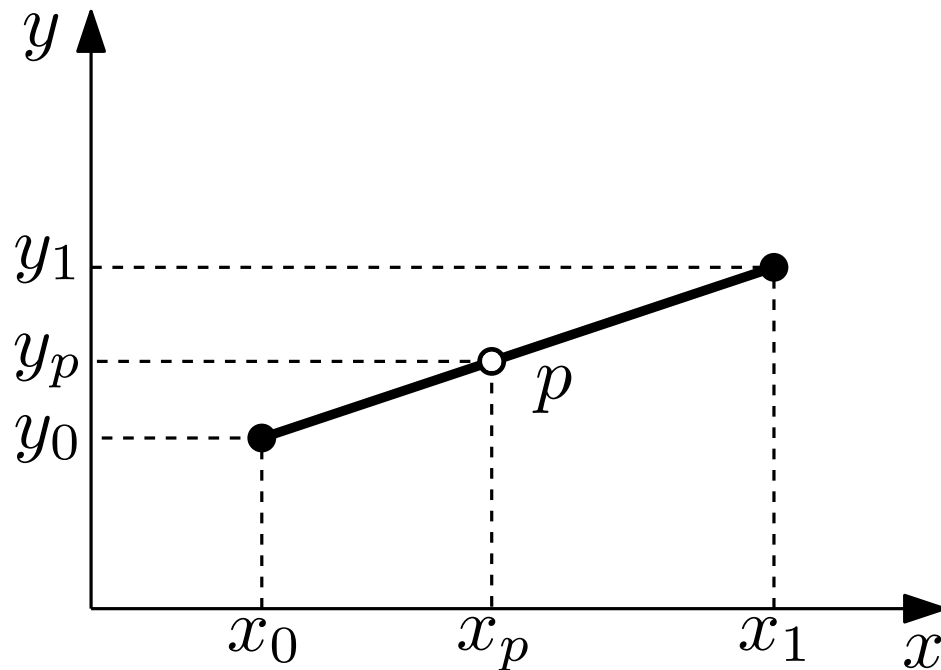
- Incrementar em x e utilizar a equação da reta de suporte para calcular o valor de y



Algoritmo ingênuo

Ideia:

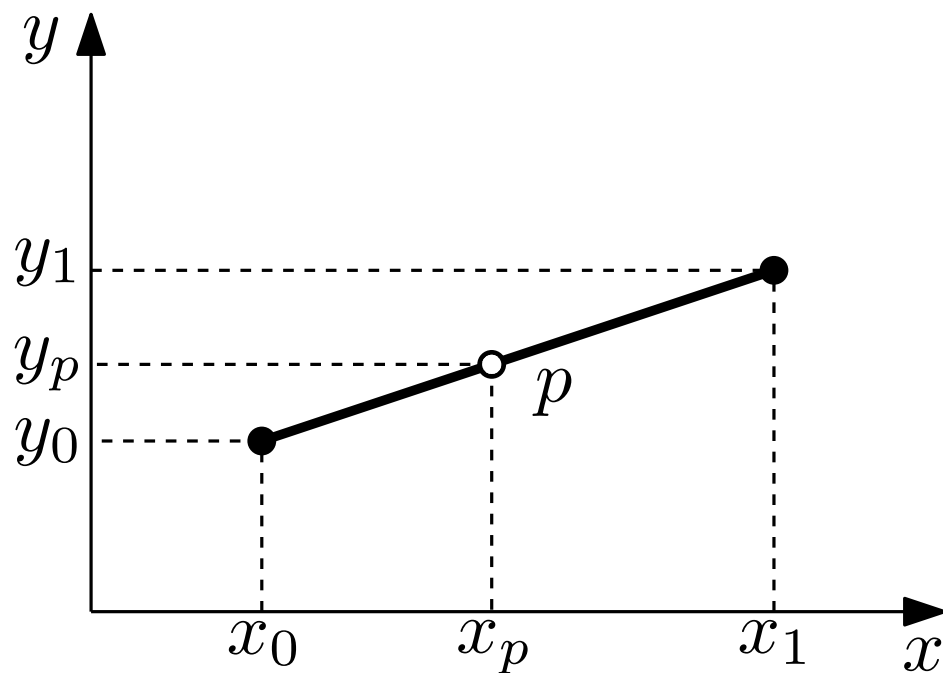
- Incrementar em x e utilizar a equação da reta de suporte para calcular o valor de y



$$\frac{y_p - y_0}{x_p - x_0} = \frac{y_1 - y_0}{x_1 - x_0}$$

Ideia:

- Incrementar em x e utilizar a equação da reta de suporte para calcular o valor de y



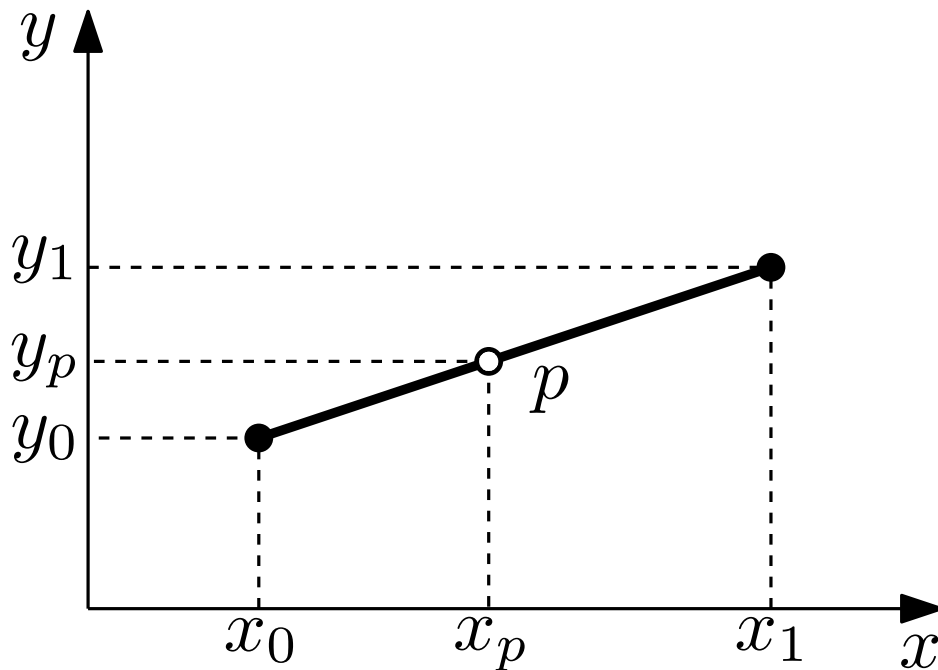
$$\frac{y_p - y_0}{x_p - x_0} = \frac{y_1 - y_0}{x_1 - x_0}$$

$$y_p = y_0 + \frac{y_1 - y_0}{x_1 - x_0} (x_p - x_0)$$

Algoritmo ingênuo

Ideia:

- Incrementar em x e utilizar a equação da reta de suporte para calcular o valor de y



$$\frac{y_p - y_0}{x_p - x_0} = \frac{y_1 - y_0}{x_1 - x_0}$$

$$\begin{aligned} y_p &= y_0 + \frac{y_1 - y_0}{x_1 - x_0} (x_p - x_0) \\ &= y_0 + m(x_p - x_0) \end{aligned}$$

Algoritmo ingênuo

Pseudocódigo

$m := (y_1 - y_0) / (x_1 - x_0)$

para $x := x_0$ **até** x_1 , **faça**:

$y := y_0 + m \times (x - x_0)$

 Pinte pixel $(x, \lfloor y + 0.5 \rfloor)$

fim para

Algoritmo ingênuo

Pseudocódigo

$m := (y_1 - y_0) / (x_1 - x_0)$

para $x := x_0$ **até** x_1 , **faça**:

$y := y_0 + m \times (x - x_0)$

Pinte pixel $(x, \lfloor y + 0.5 \rfloor)$

fim para

Algoritmo simples, porém:

- Utiliza aritmética de ponto flutuante (m precisa ser real)

Algoritmo ingênuo

Pseudocódigo

$m := (y_1 - y_0) / (x_1 - x_0)$

para $x := x_0$ **até** x_1 , **faça**:

$y := y_0 + m \times (x - x_0)$

Pinte pixel $(x, \lfloor y + 0.5 \rfloor)$

fim para

Algoritmo simples, porém:

- Utiliza aritmética de ponto flutuante (m precisa ser real)
- Susceptível a erros de arredondamento

Algoritmo ingênuo

Pseudocódigo

```
 $m := (y_1 - y_0) / (x_1 - x_0)$   
para  $x := x_0$  até  $x_1$ , faça:  
     $y := y_0 + m \times (x - x_0)$   
    Pinte pixel  $(x, \lfloor y + 0.5 \rfloor)$   
fim para
```

Algoritmo simples, porém:

- Utiliza aritmética de ponto flutuante (m precisa ser real)
- Susceptível a erros de arredondamento
- Implementação lenta

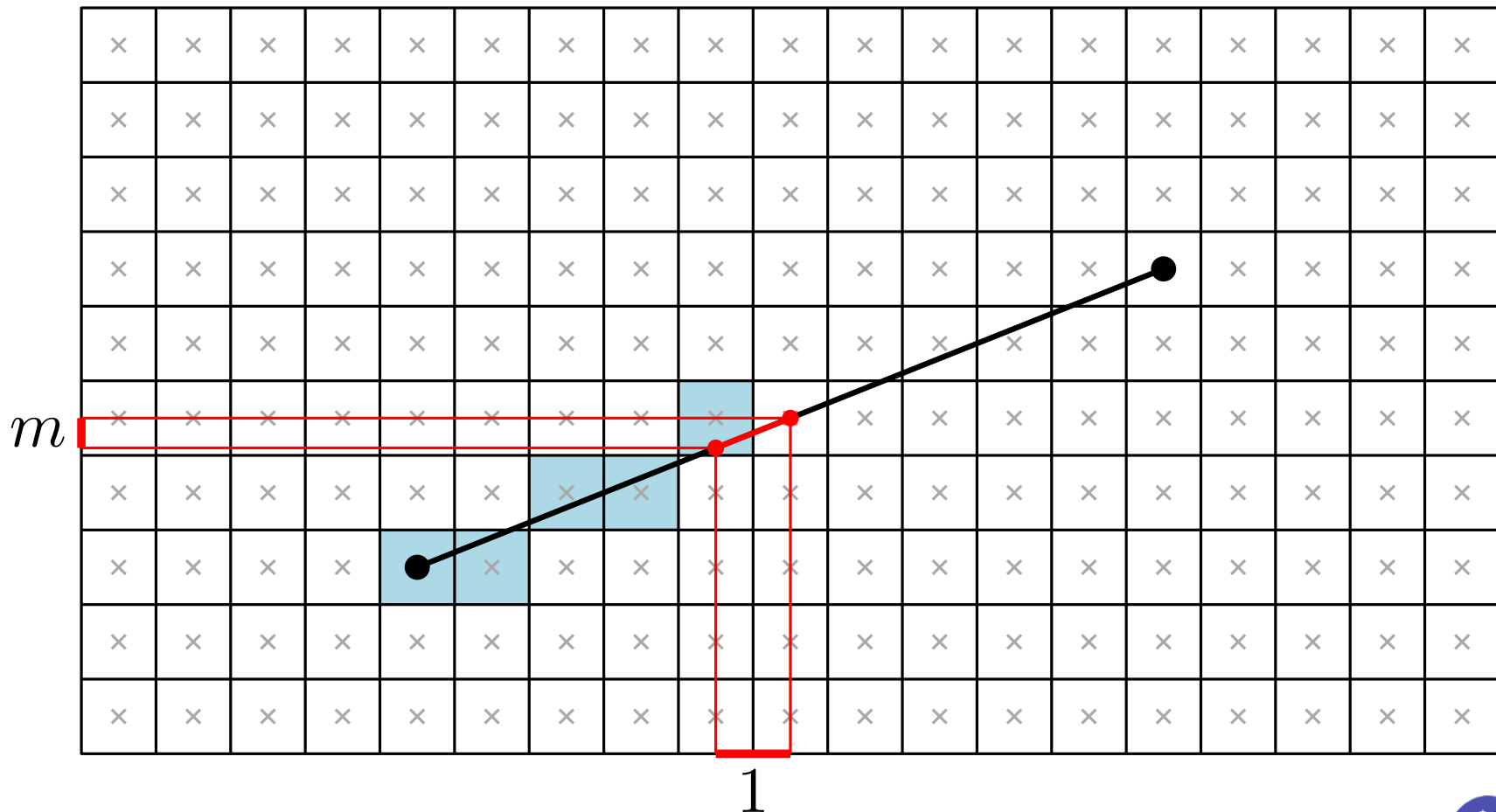
Algoritmo incremental (DDA)

Utiliza a ideia do *Digital Differential Analyzer* (DDA) para reduzir emprego de aritmética de ponto flutuante

Algoritmo incremental (DDA)

Observação.

Um incremento unitário em x implica em um incremento de m unidades em y



Algoritmo incremental (DDA)

Pseudocódigo

$x := x_0$

$y := y_0$

$m := (y_1 - y_0) / (x_1 - x_0)$

Pinte pixel $(x, \lfloor y + 0.5 \rfloor)$

enquanto $x \leq x_1$, **faça:**

$x := x + 1$

$y := y + m$

 Pinte pixel $(x, \lfloor y + 0.5 \rfloor)$

fim enquanto

Algoritmo incremental (DDA)

Pseudocódigo

$x := x_0$

$y := y_0$

$m := (y_1 - y_0) / (x_1 - x_0)$

Pinte pixel $(x, \lfloor y + 0.5 \rfloor)$

enquanto $x \leq x_1$, **faça:**

$x := x + 1$

$y := y + m$

 Pinte pixel $(x, \lfloor y + 0.5 \rfloor)$

fim enquanto

Evita a multiplicação com ponto flutuante, mas continua com operações de adição e arredondamento

Algoritmo de Bresenham

- Um dos algoritmos clássicos da computação gráfica
- Emprega apenas aritmética inteira
- Algoritmo padrão implementado em hardwares e softwares de rasterização

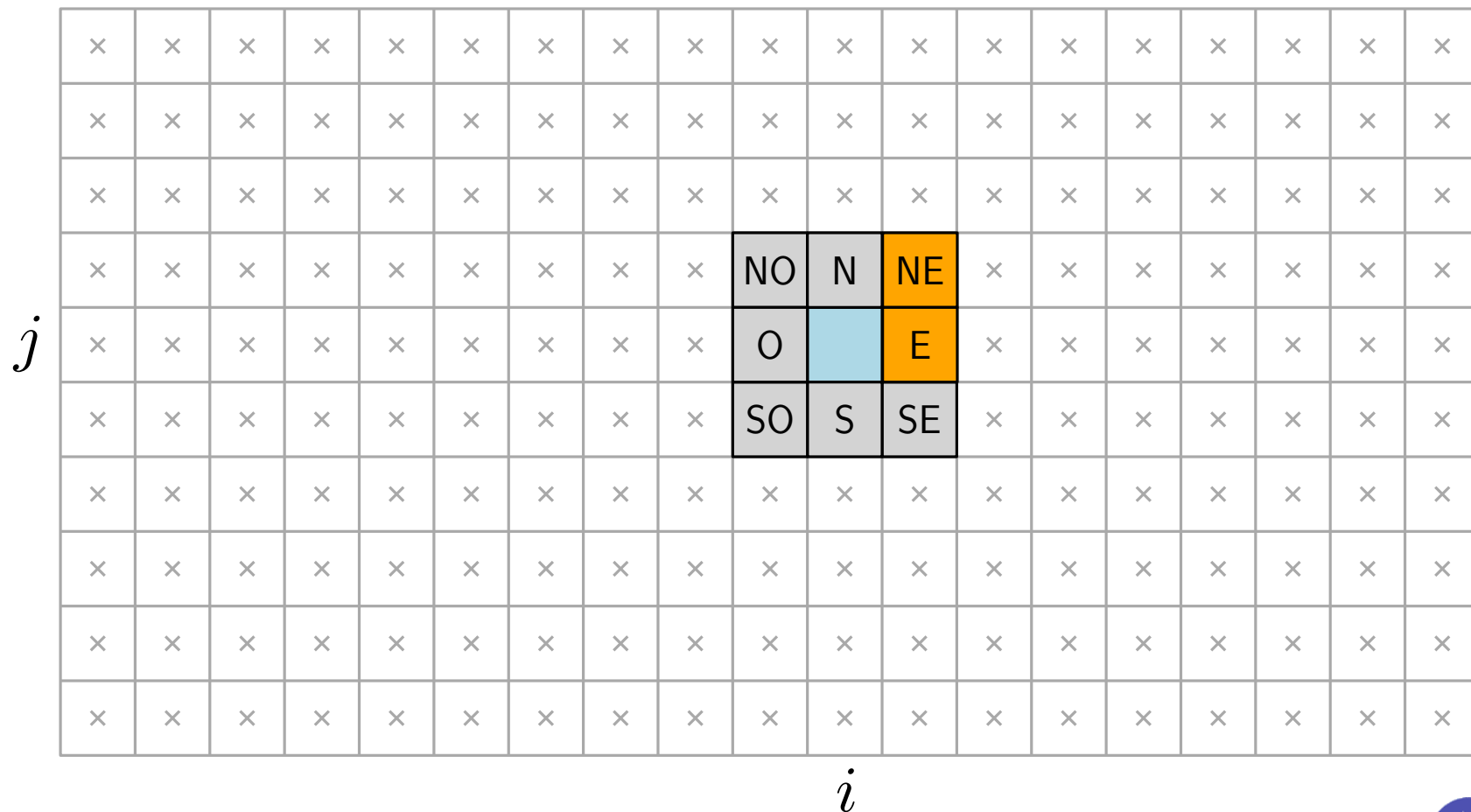
Idéia básica.

Se um pixel (i, j) é pintado, então o próximo pixel a ser pintado é o $E(i, j)$ ou o $NE(i, j)$

Algoritmo de Bresenham

Idéia básica.

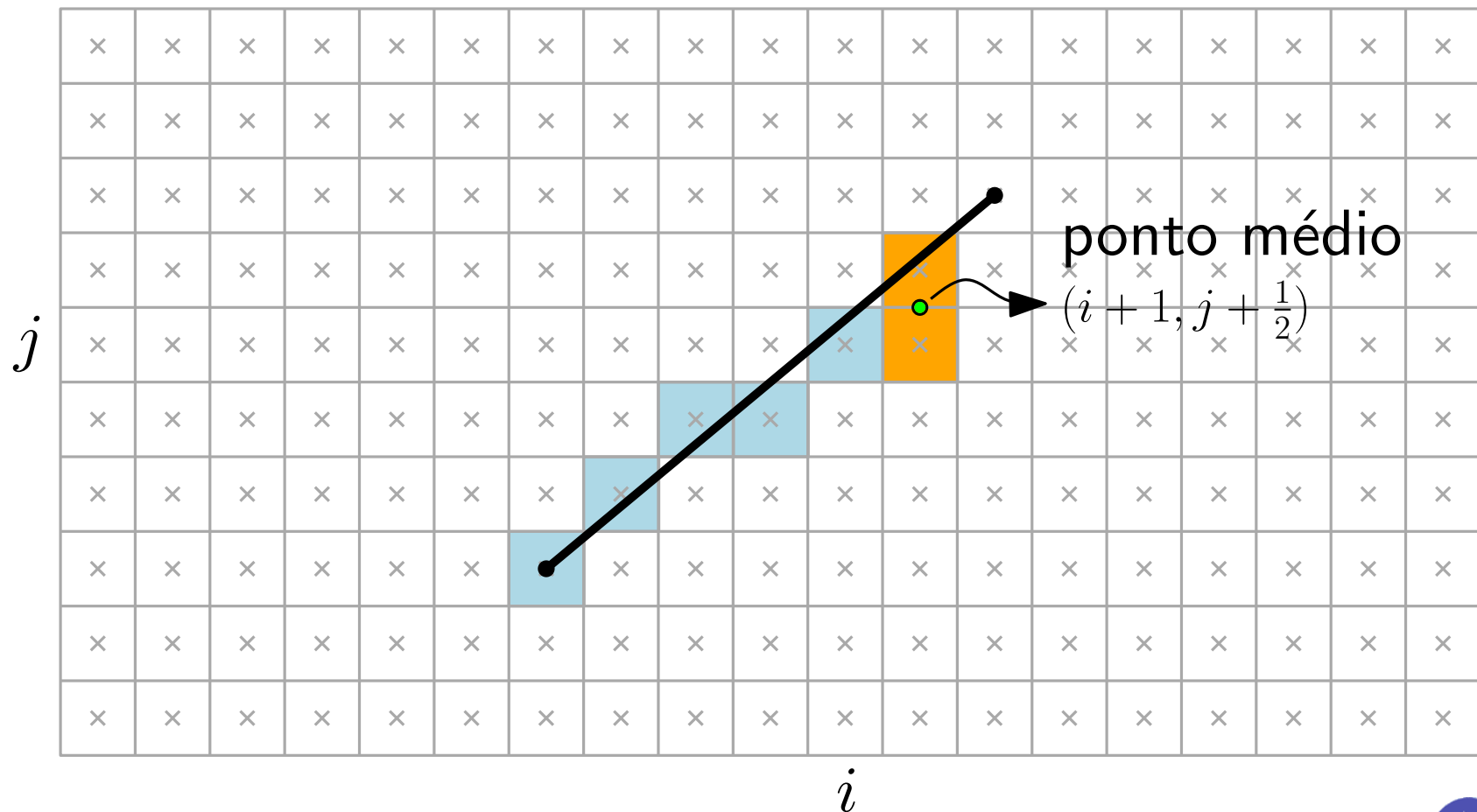
Se um pixel (i, j) é pintado, então o próximo pixel a ser pintado é o $E(i, j)$ ou o $NE(i, j)$



Algoritmo de Bresenham

Idéia básica.

Se um pixel (i, j) é pintado, então o próximo pixel a ser pintado é o $E(i, j)$ ou o $NE(i, j)$

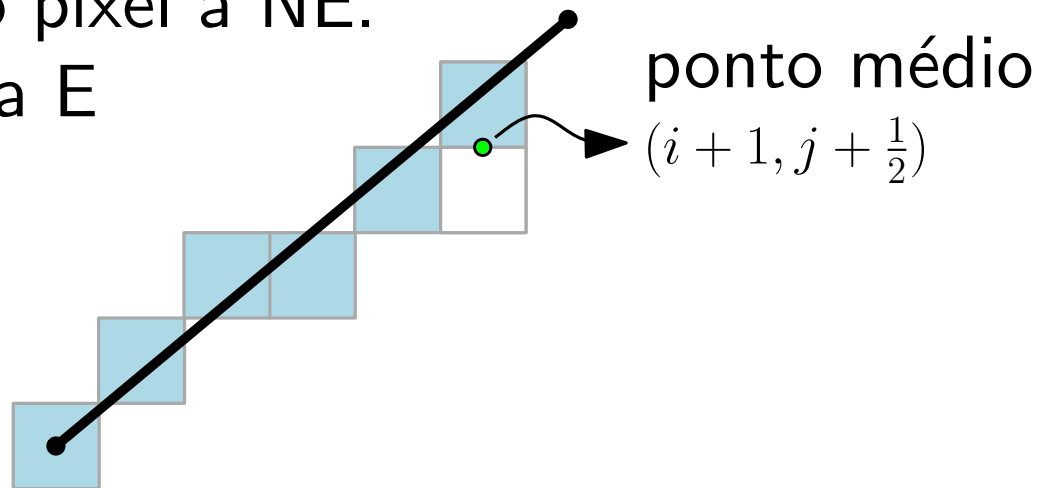


Idéia básica.

Se um pixel (i, j) é pintado, então o próximo pixel a ser pintado é o $E(i, j)$ ou o $NE(i, j)$

Se segmento passa acima do ponto médio, pinte o pixel a NE.

Senão, pinte o pixel a E



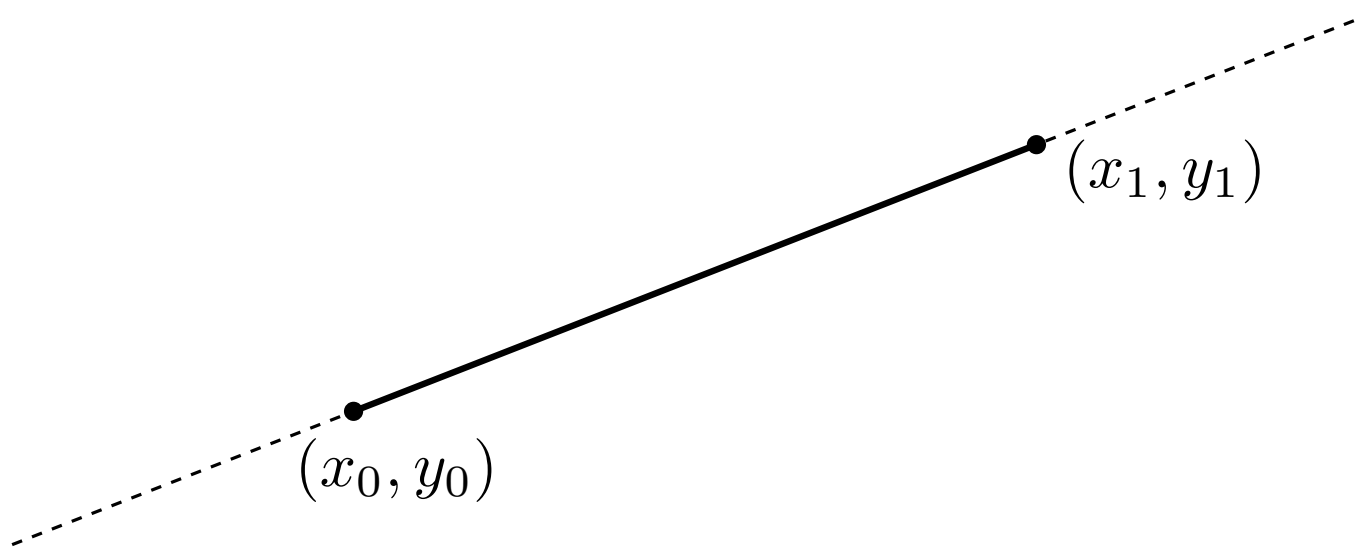
Predicado de decisão.

A equação do hiperplano afim gerado por um segmento de reta pode ser utilizada para decidir em qual região do plano um ponto p está localizado

Predicado de decisão.

A equação do hiperplano afim gerado por um segmento de reta pode ser utilizada para decidir em qual região do plano um ponto p está localizado

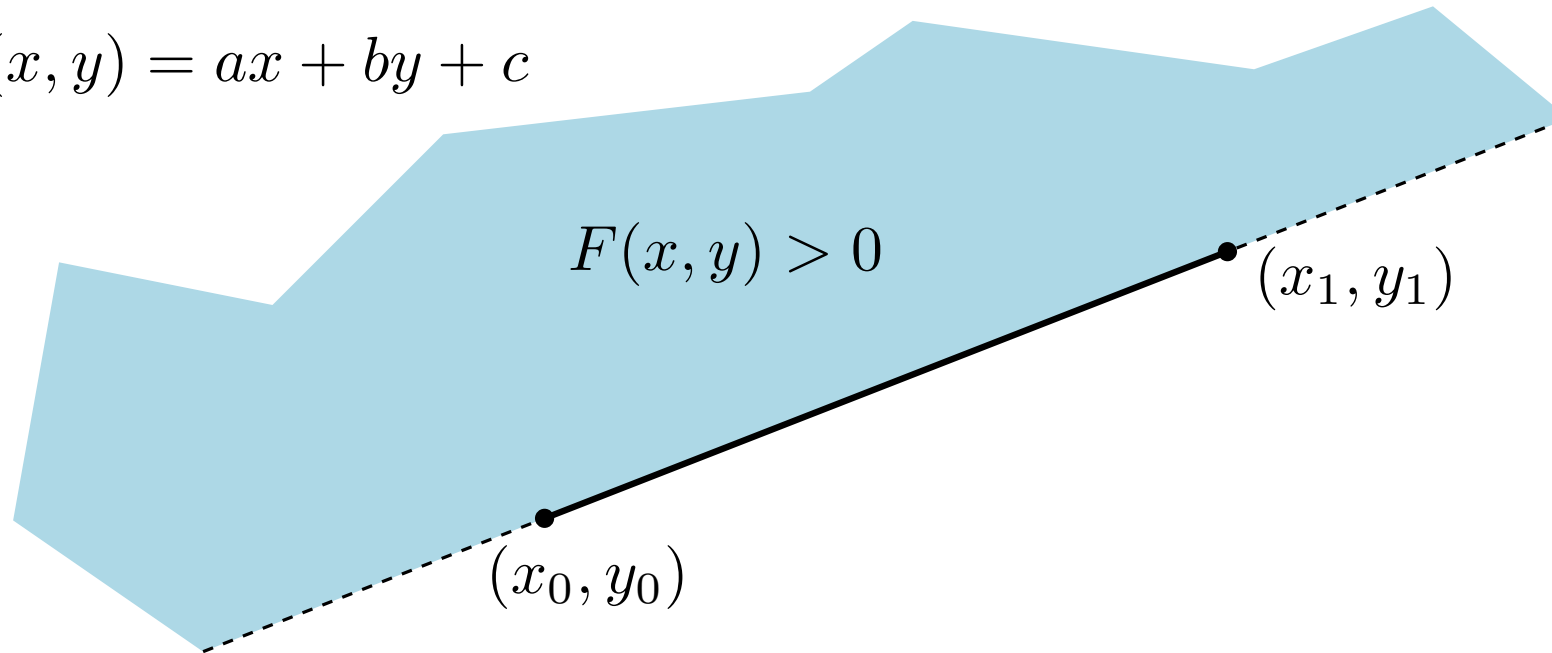
$$F(x, y) = ax + by + c$$



Predicado de decisão.

A equação do hiperplano afim gerado por um segmento de reta pode ser utilizada para decidir em qual região do plano um ponto p está localizado

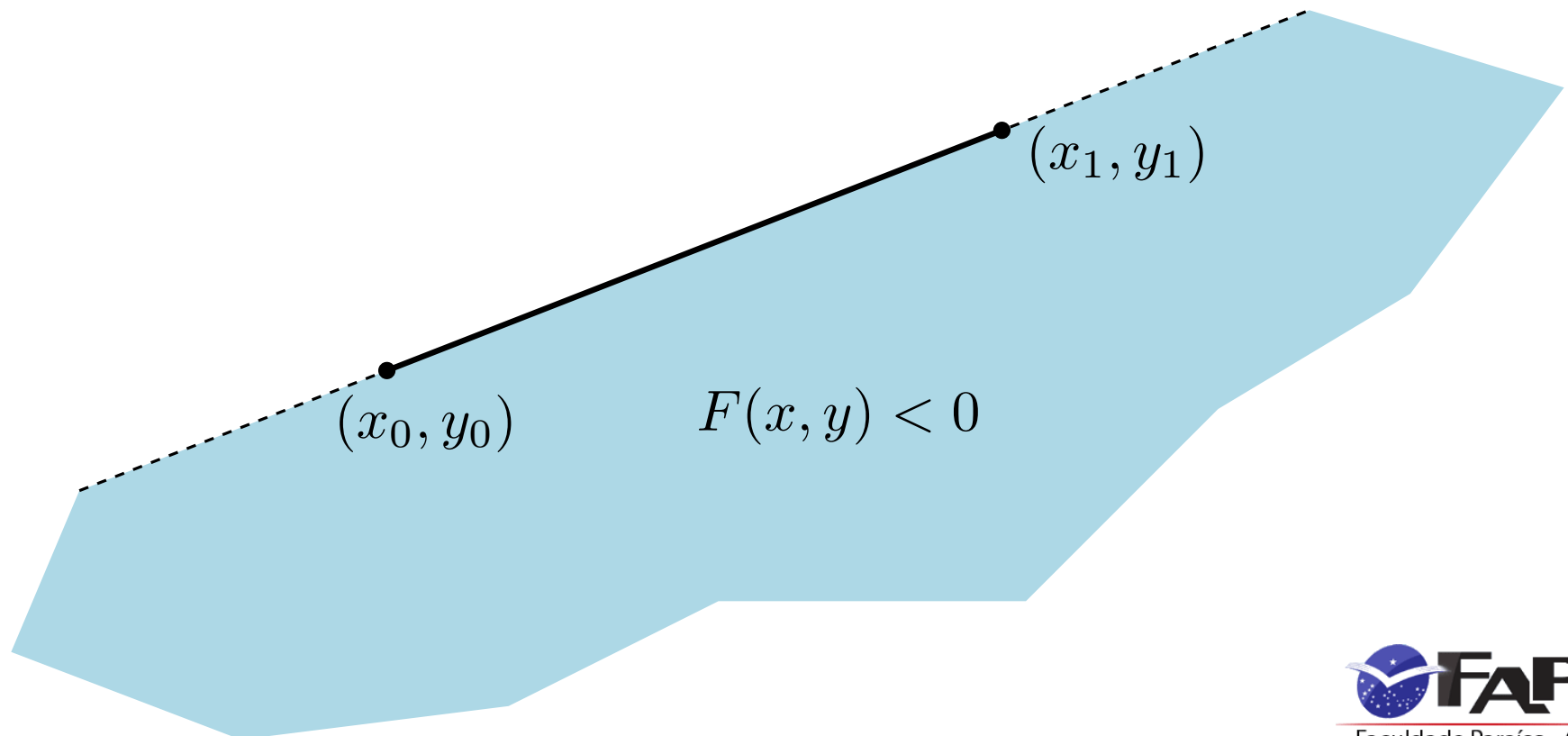
$$F(x, y) = ax + by + c$$



Predicado de decisão.

A equação do hiperplano afim gerado por um segmento de reta pode ser utilizada para decidir em qual região do plano um ponto p está localizado

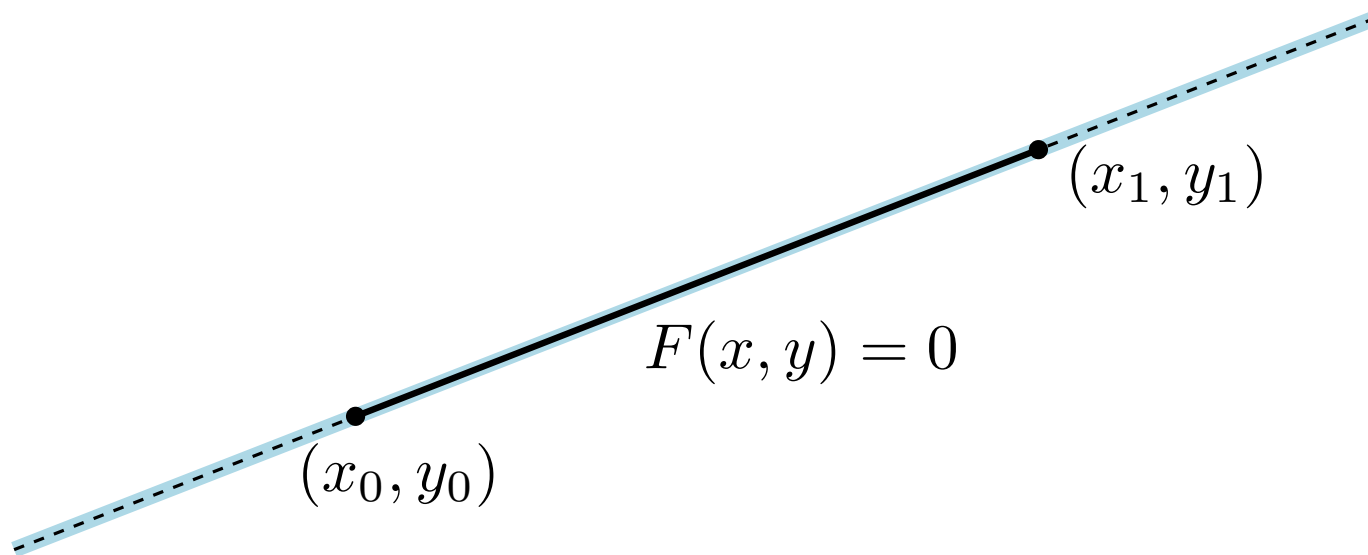
$$F(x, y) = ax + by + c$$



Predicado de decisão.

A equação do hiperplano afim gerado por um segmento de reta pode ser utilizada para decidir em qual região do plano um ponto p está localizado

$$F(x, y) = ax + by + c$$



Predicado de decisão.

A equação do hiperplano afim gerado por um segmento de reta pode ser utilizada para decidir em qual região do plano um ponto p está localizado

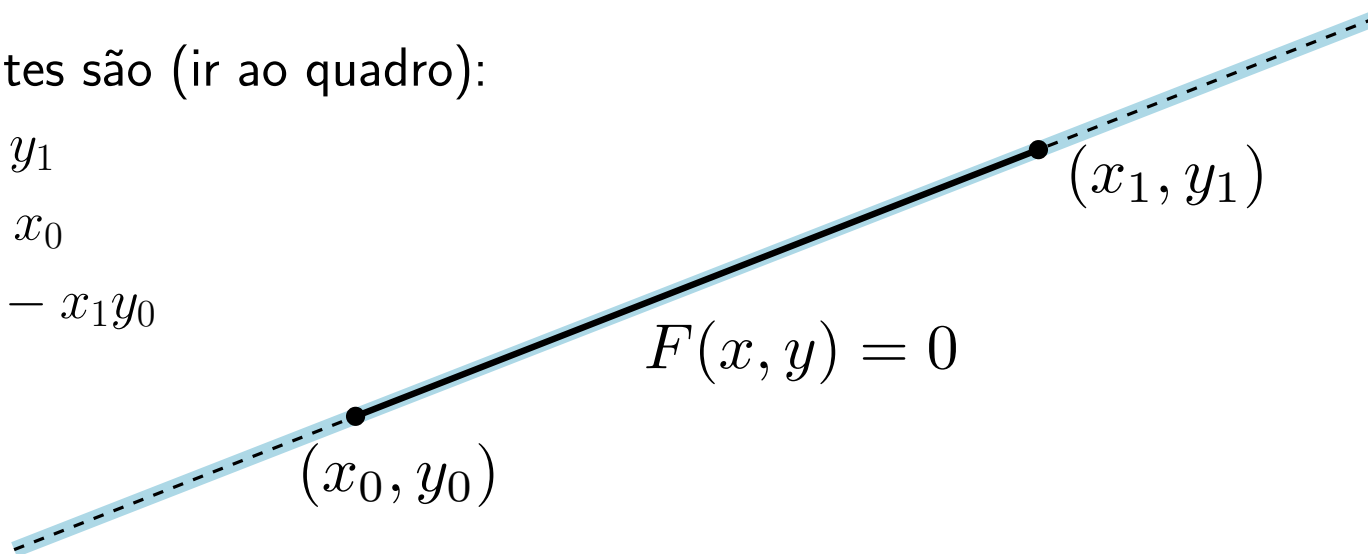
$$F(x, y) = ax + by + c$$

Os coeficientes são (ir ao quadro):

$$a = y_0 - y_1$$

$$b = x_1 - x_0$$

$$c = x_0y_1 - x_1y_0$$



A grande sacada.

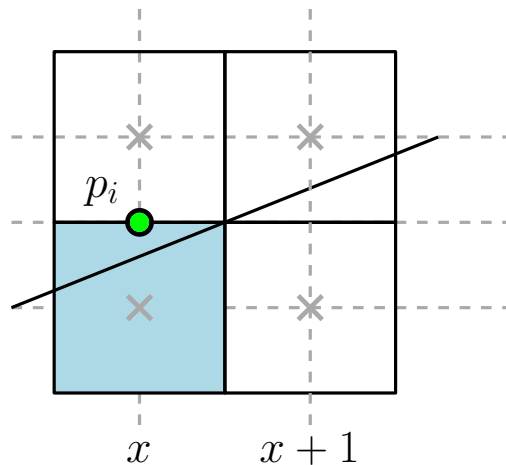
A função $F(x, y)$ pode ser calculada iterativamente

A grande sacada.

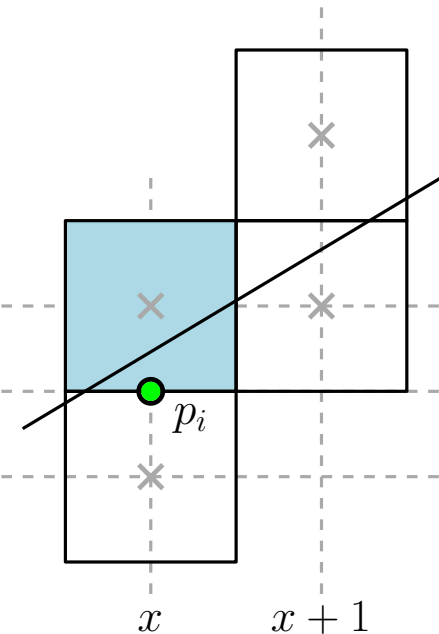
A função $F(x, y)$ pode ser calculada iterativamente

Existem duas situações:

Pixel selecionado anteriormente estava *abaixo* do ponto médio



$y + 1$
 $y + \frac{1}{2}$
 y



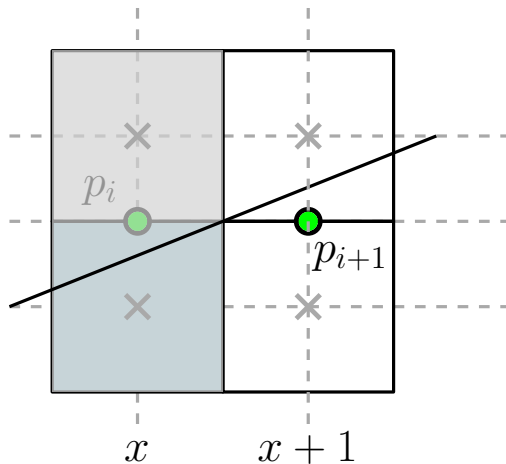
Pixel selecionado anteriormente estava *acima* do ponto médio

A grande sacada.

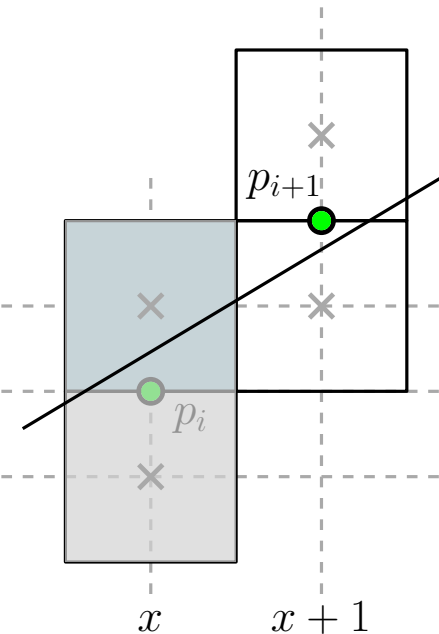
A função $F(x, y)$ pode ser calculada iterativamente

Existem duas situações:

Pixel selecionado anteriormente estava *abaixo* do ponto médio



$y + 1$
 $y + \frac{1}{2}$
 y



Pixel selecionado anteriormente estava *acima* do ponto médio

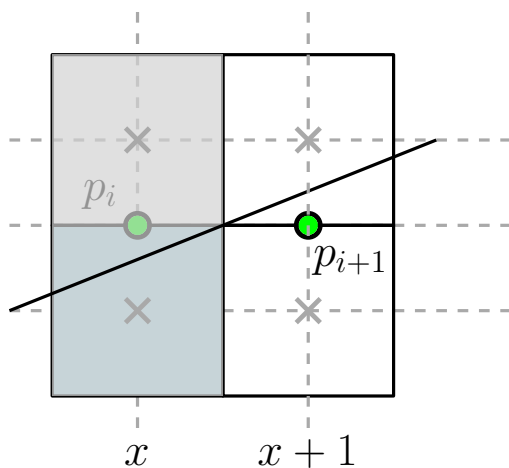
A grande sacada.

A função $F(x, y)$ pode ser calculada iterativamente

Existem duas situações:

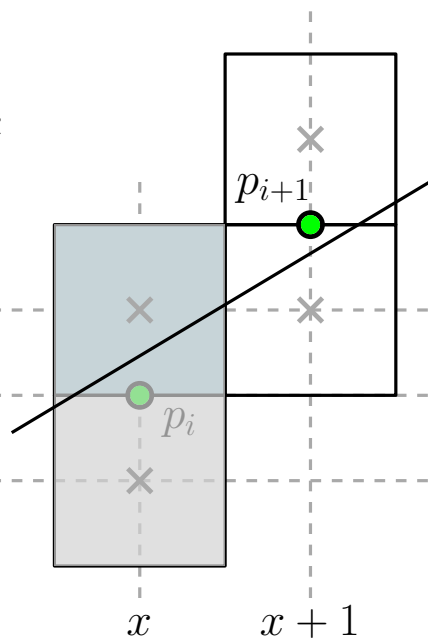
$$F(p_i) = F(x, y + \frac{1}{2}) = ax + b(y + \frac{1}{2}) + c$$

Pixel selecionado anteriormente estava *abaixo* do ponto médio



$y + 1$
 $y + \frac{1}{2}$
 y

Pixel selecionado anteriormente estava *acima* do ponto médio



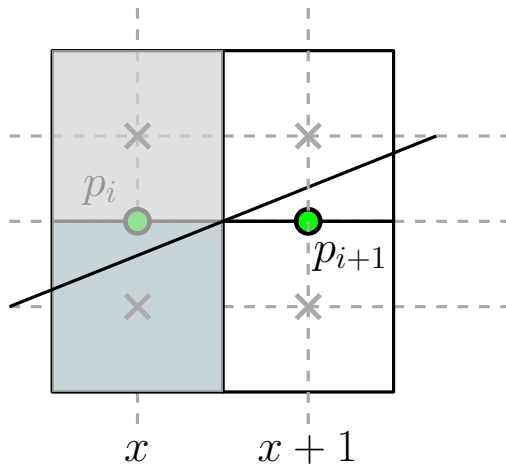
A grande sacada.

A função $F(x, y)$ pode ser calculada iterativamente

Existem duas situações:

$$F(p_i) = F(x, y + \frac{1}{2}) = ax + b(y + \frac{1}{2}) + c$$

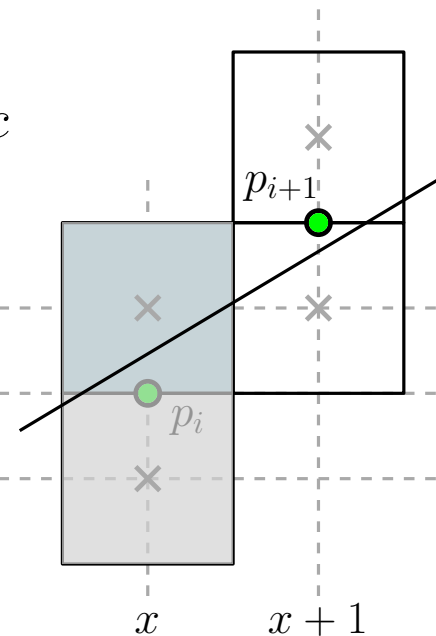
Pixel selecionado anteriormente estava *abaixo* do ponto médio



$$\begin{aligned} F(p_{i+1}) &= F(x + 1, y + \frac{1}{2}) \\ &= a(x + 1) + b\left(y + \frac{1}{2}\right) + c \\ &= F(p_i) + a \end{aligned}$$

$y + 1$
 $y + \frac{1}{2}$
 y

Pixel selecionado anteriormente estava *acima* do ponto médio



$$\begin{aligned} F(p_{i+1}) &= F(x + 1, y + 1 + \frac{1}{2}) \\ &= a(x + 1) + b\left(y + 1 + \frac{1}{2}\right) + c \\ &= F(p_i) + a + b \end{aligned}$$

Algoritmo de Bresenham

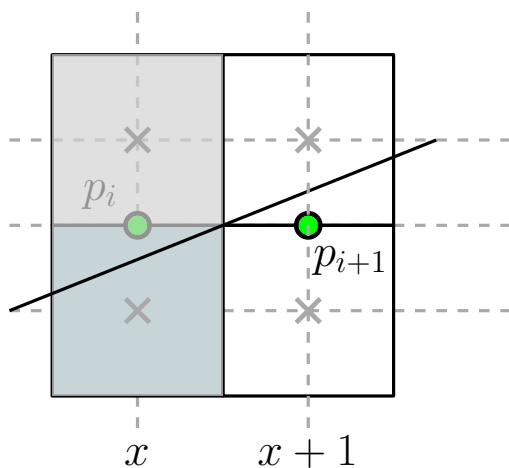
A grande sacada.

A função $F(x, y)$ pode ser calculada iterativamente

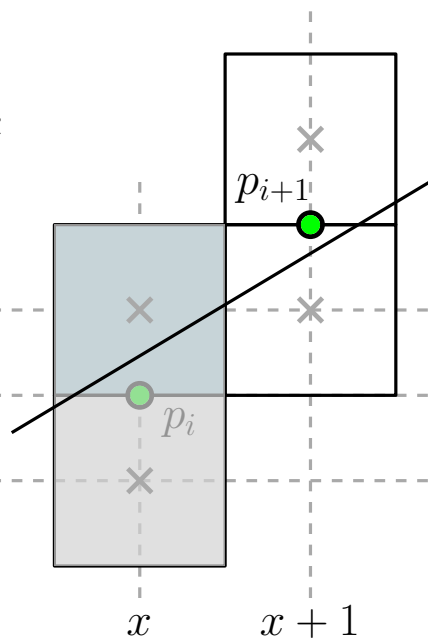
Existem duas situações:

$$F(p_i) = F(x, y + \frac{1}{2}) = ax + b(y + \frac{1}{2}) + c$$

Pixel selecionado anteriormente estava *abaixo* do ponto médio



$y + 1$
 $y + \frac{1}{2}$
 y



Pixel selecionado anteriormente estava *acima* do ponto médio

O algoritmo iniciará com:

$$\begin{aligned} F(x_0 + 1, y_0 + \frac{1}{2}) &= a(x_0 + 1) + b\left(y_0 + \frac{1}{2}\right) + c \\ &= a + b/2 \end{aligned}$$

Algoritmo de Bresenham

Pseudocódigo

$a := y_0 - y_1$

$b := x_1 - x_0$

$F := a + b/2$

$y := y_0$

para $x := x_0$ até x_1 , **faça**:

 Pinte pixel (x, y)

se $F < 0$, **então**:

$F := F + a + b$

$y := y + 1$

senão:

$F := F + a$

fim se

fim para

Algoritmo de Bresenham

Pseudocódigo

$a := y_0 - y_1$

$b := x_1 - x_0$

$F := a + b/2$

$y := y_0$

para $x := x_0$ até x_1 , **faça**:

 Pinte pixel (x, y)

se $F < 0$, **então**:

$F := F + a + b$

$y := y + 1$

senão:

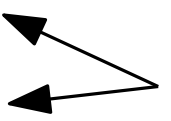
$F := F + a$

fim se

fim para

Algoritmo de Bresenham

Pseudocódigo evitando a divisão

$a := y_0 - y_1$
 $b := x_1 - x_0$  Números inteiros

$F := 2 \times a + b$

$y := y_0$

para $x := x_0$ até x_1 , **faça**:

 Pinte pixel (x, y)

se $F < 0$, **então**:

$F := F + 2 \times (a + b)$

$y := y + 1$

senão:

$F := F + 2 \times a$

fim se

fim para