Computação Gráfica

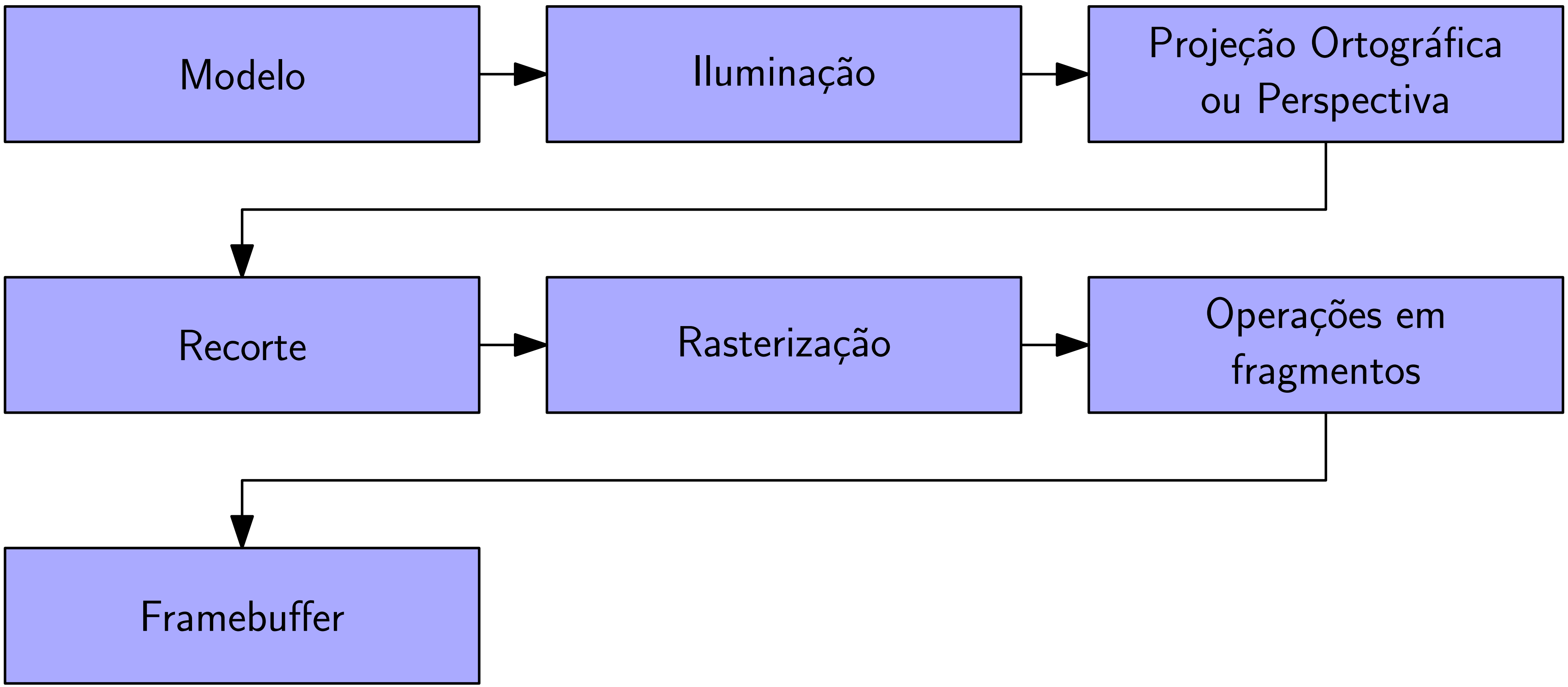
Aula 26: Rasterização (continuação)

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

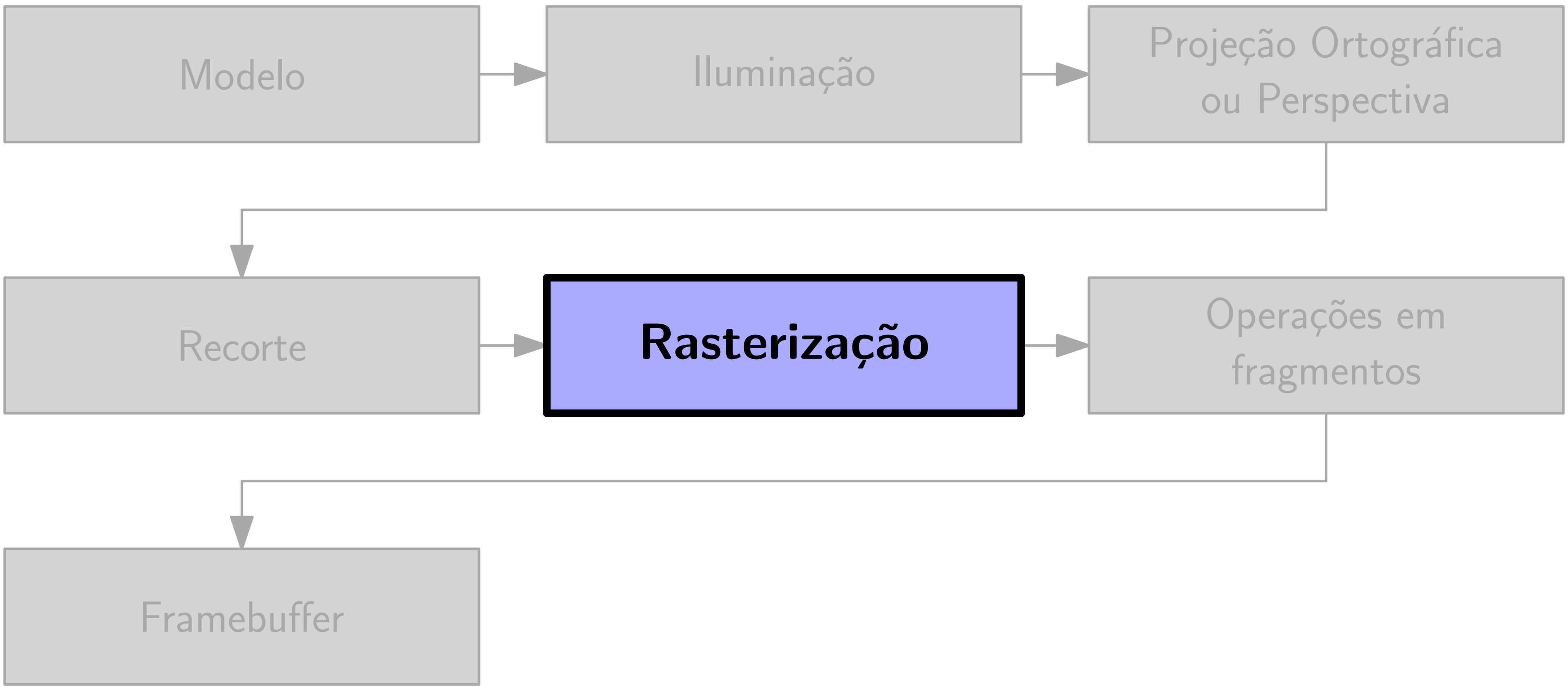
Faculdade Paraíso - CE

Relembrando

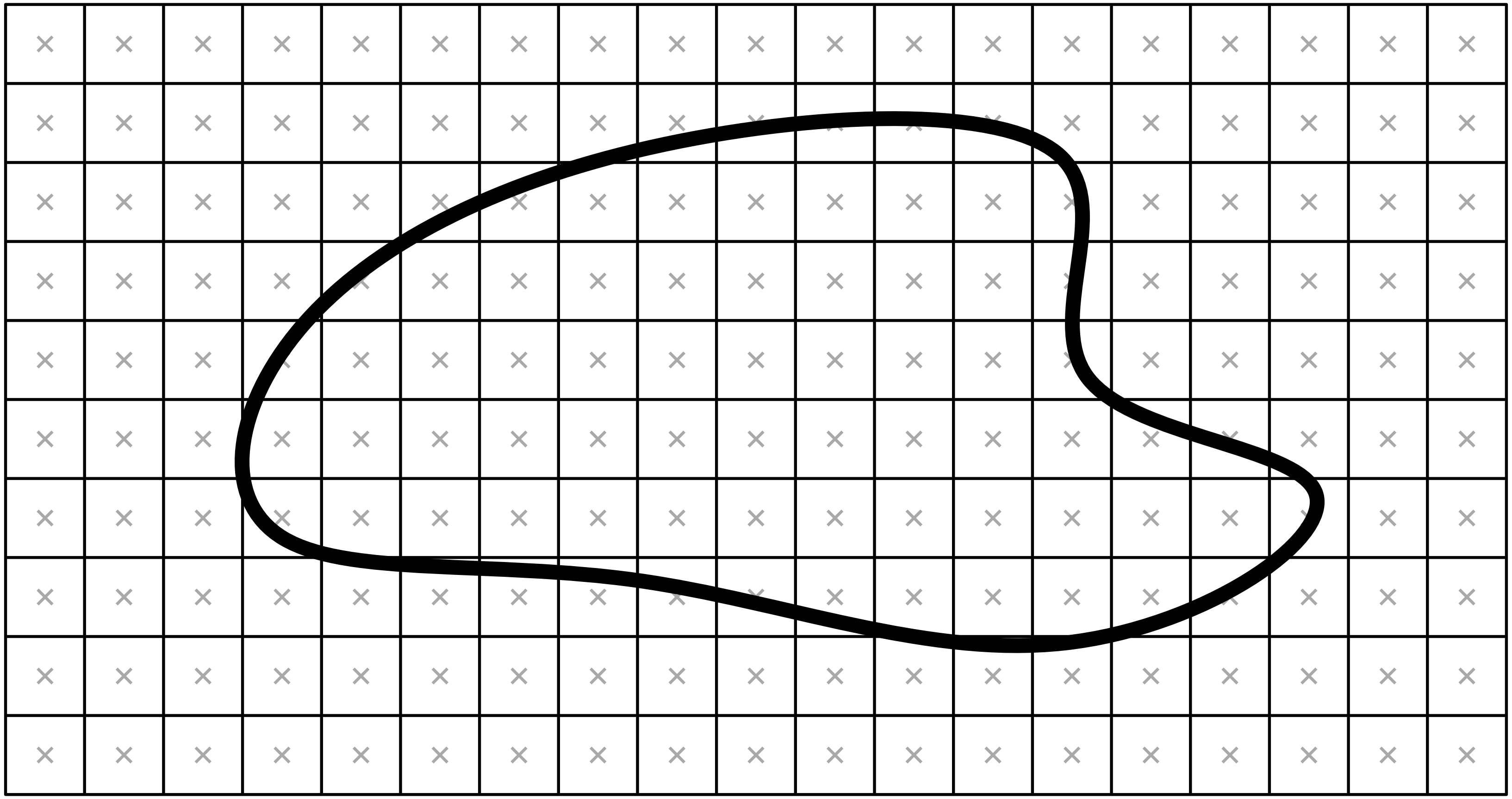
Pipeline gráfico (básico) de renderização por varredura



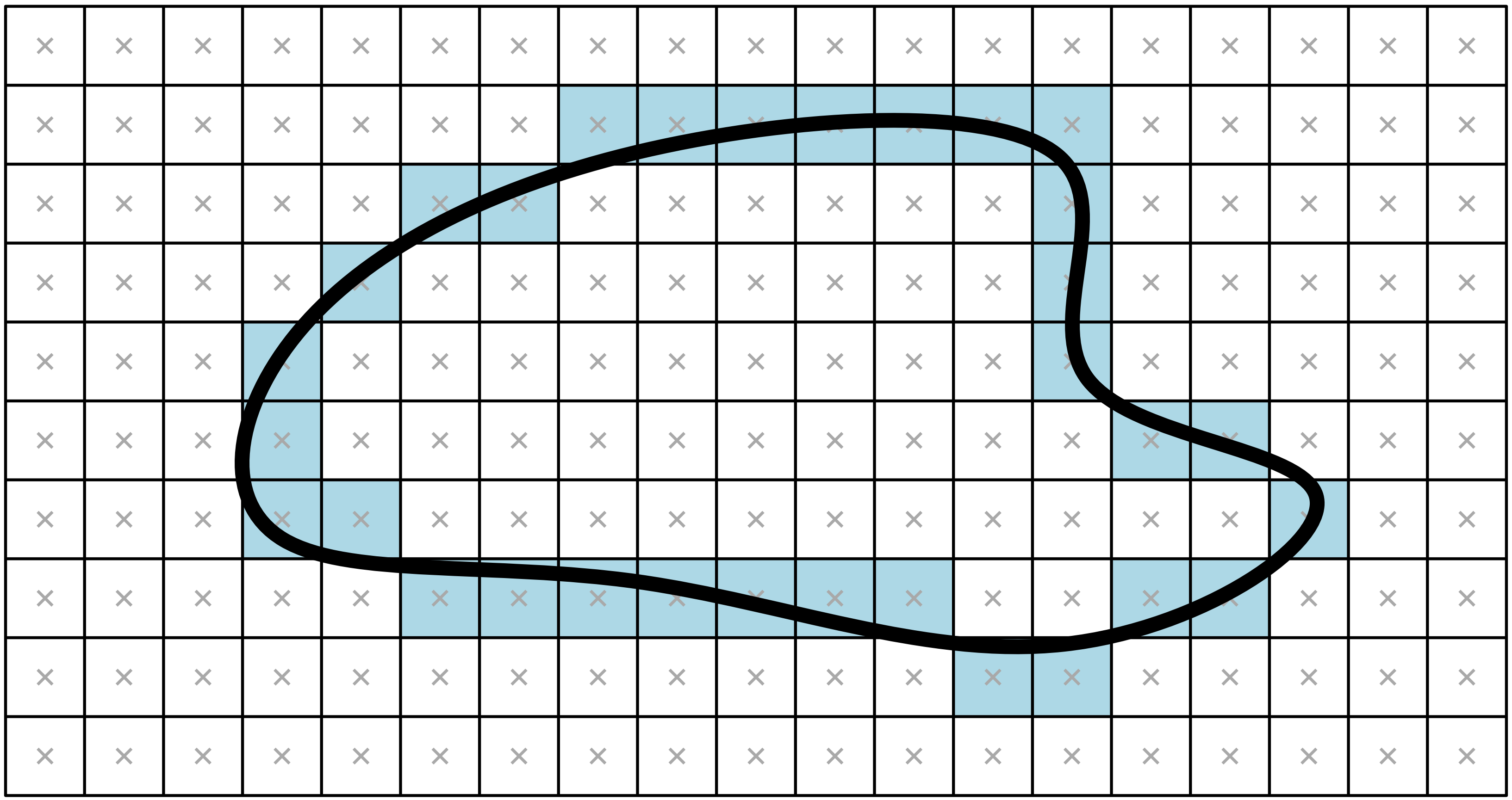
O que veremos hoje?



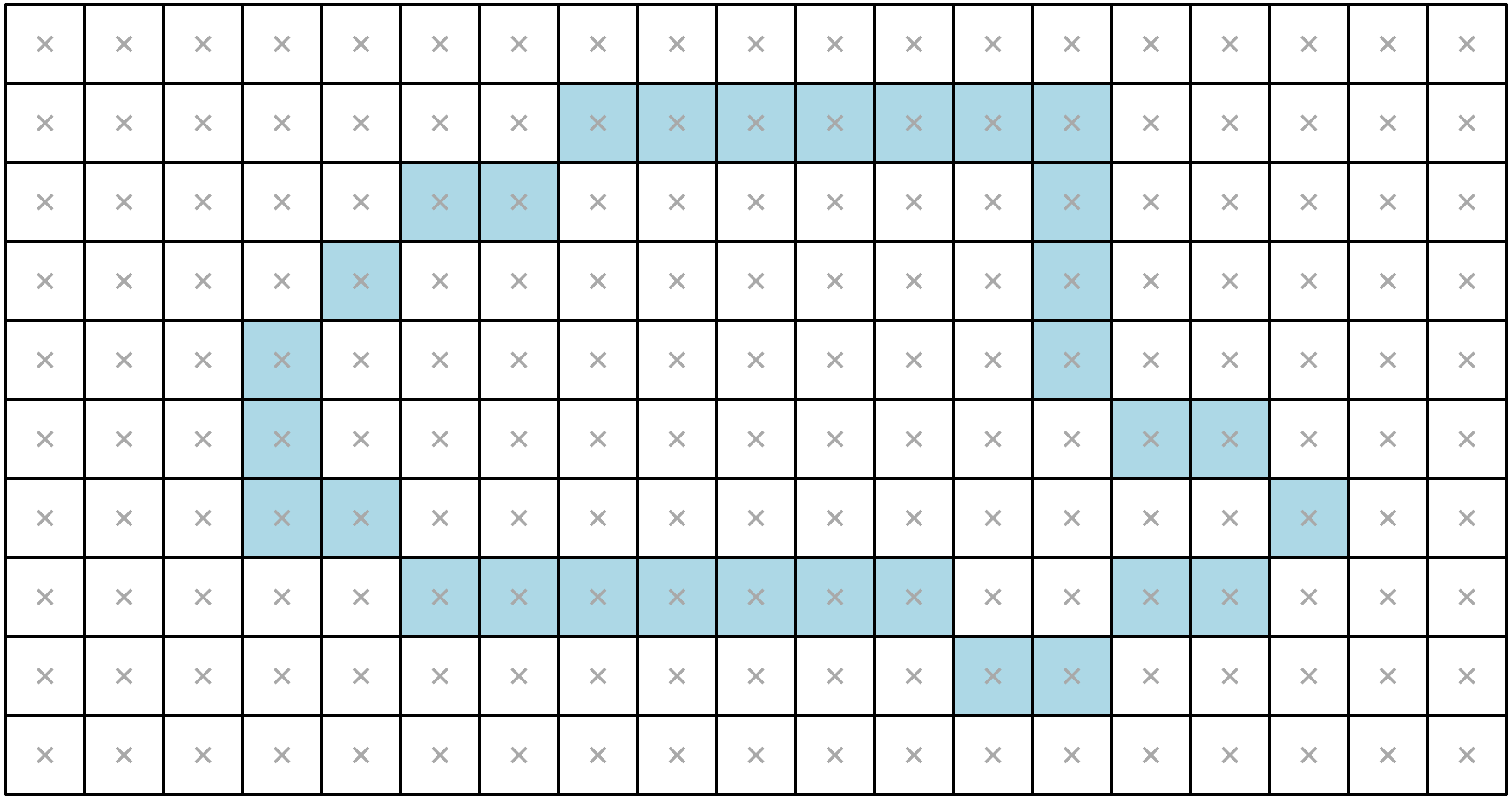
Exemplo de rasterização



Exemplo de rasterização



Exemplo de rasterização



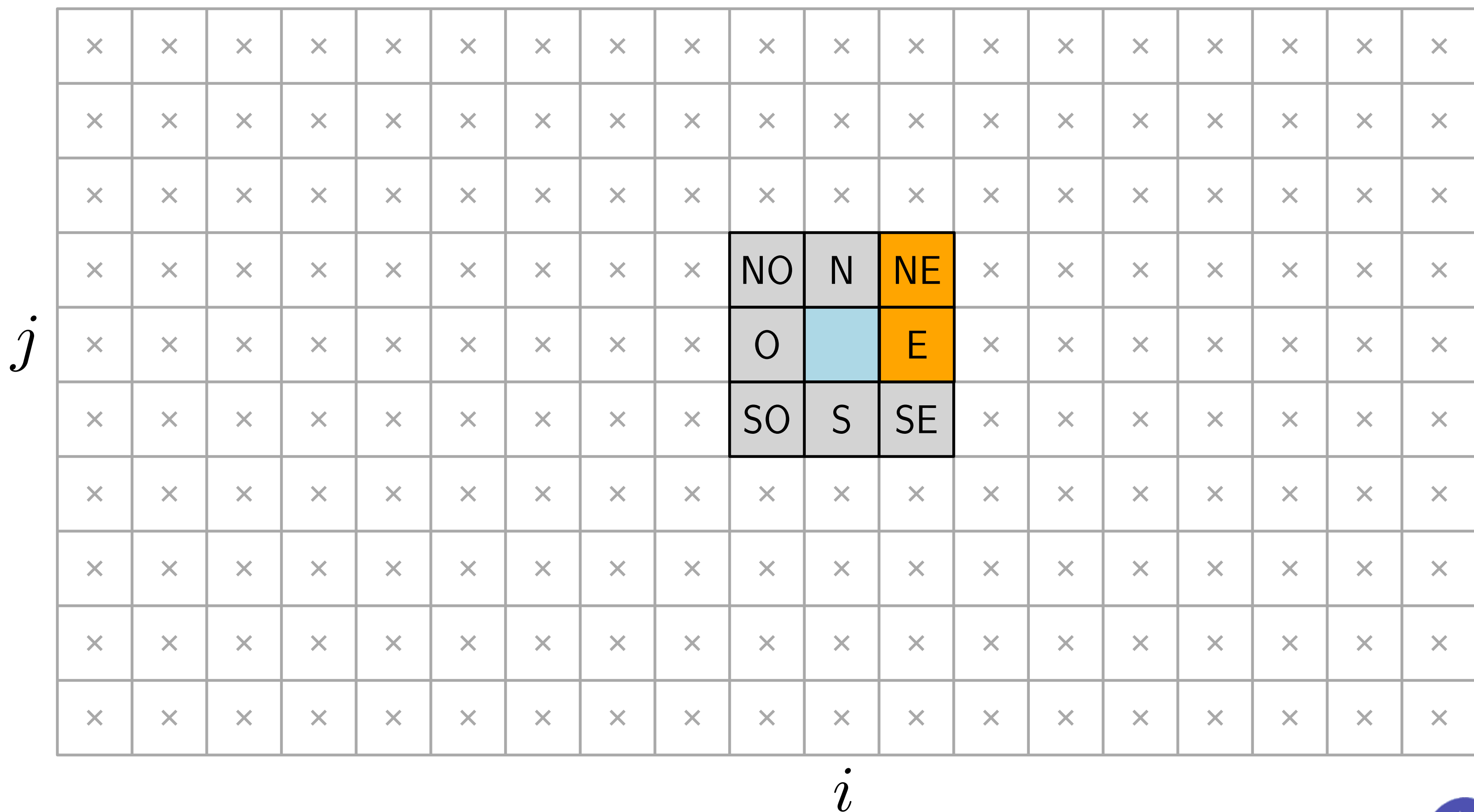
Idéia básica.

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

Relembrando o algoritmo de Bresenham

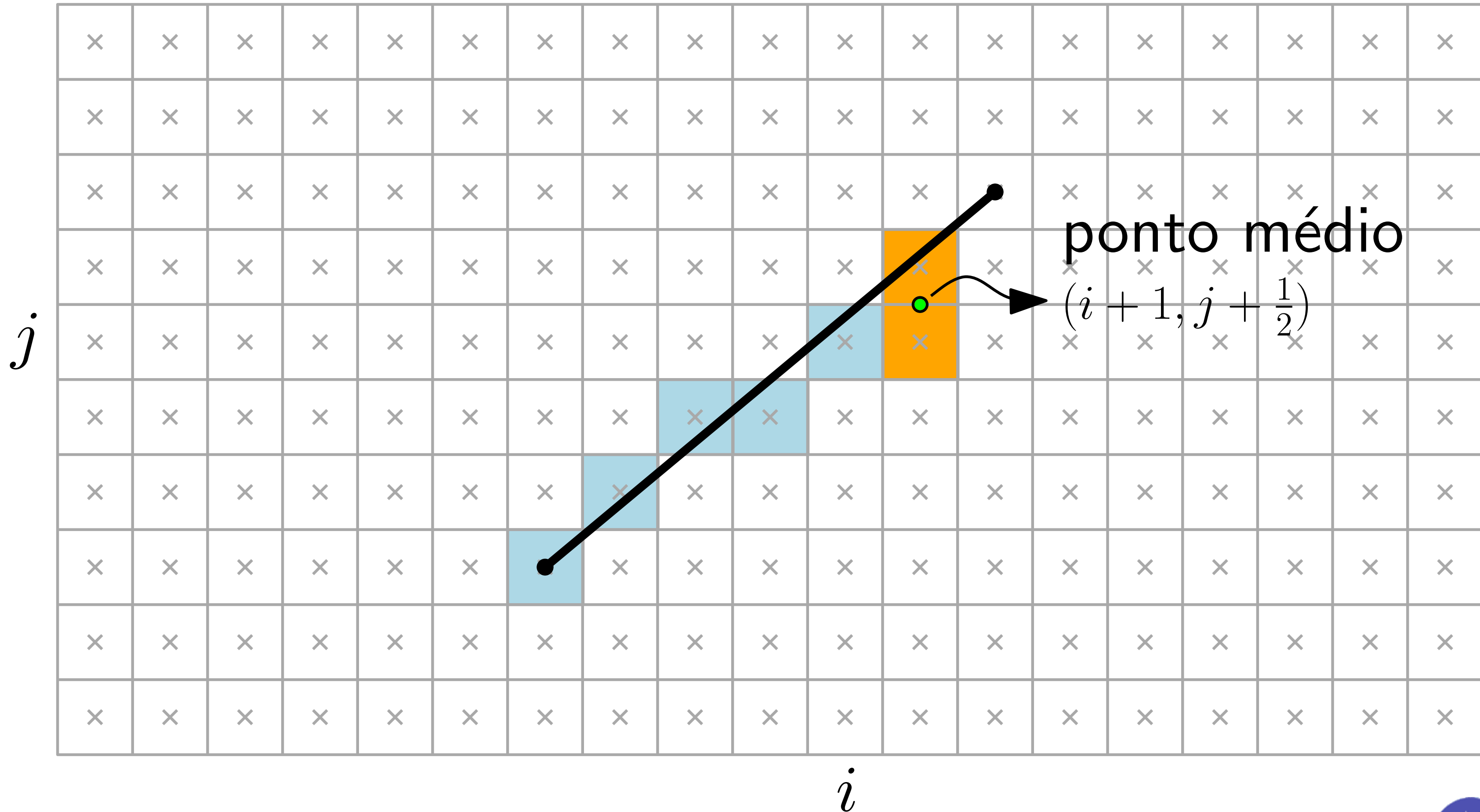
Idéia básica.

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)



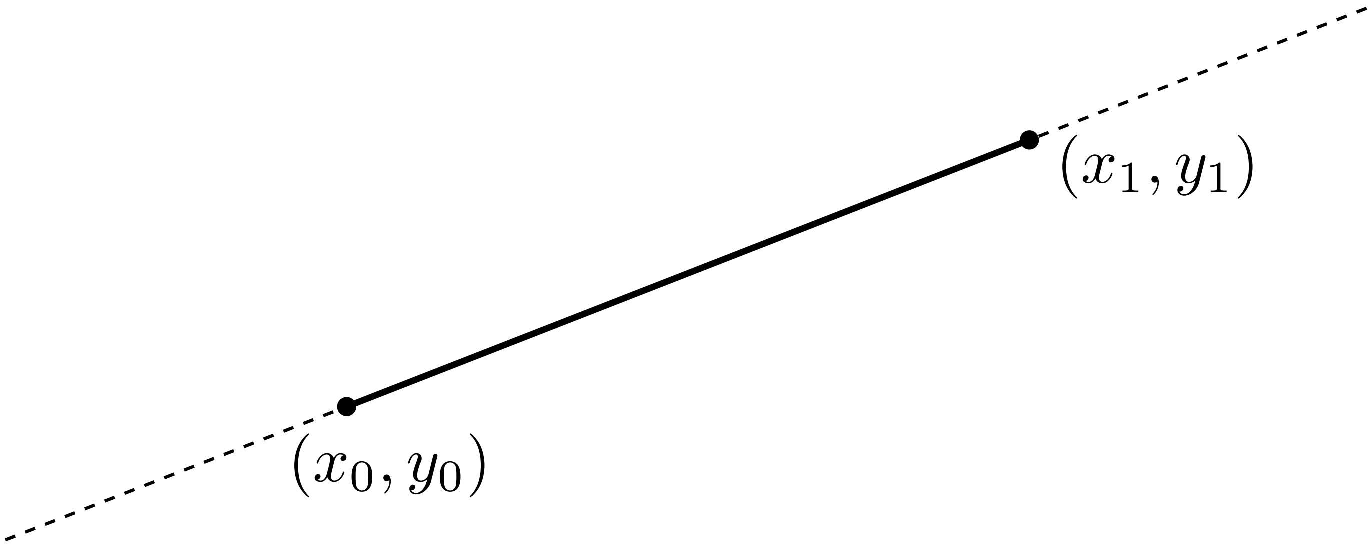
Predicado de decisão.

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.

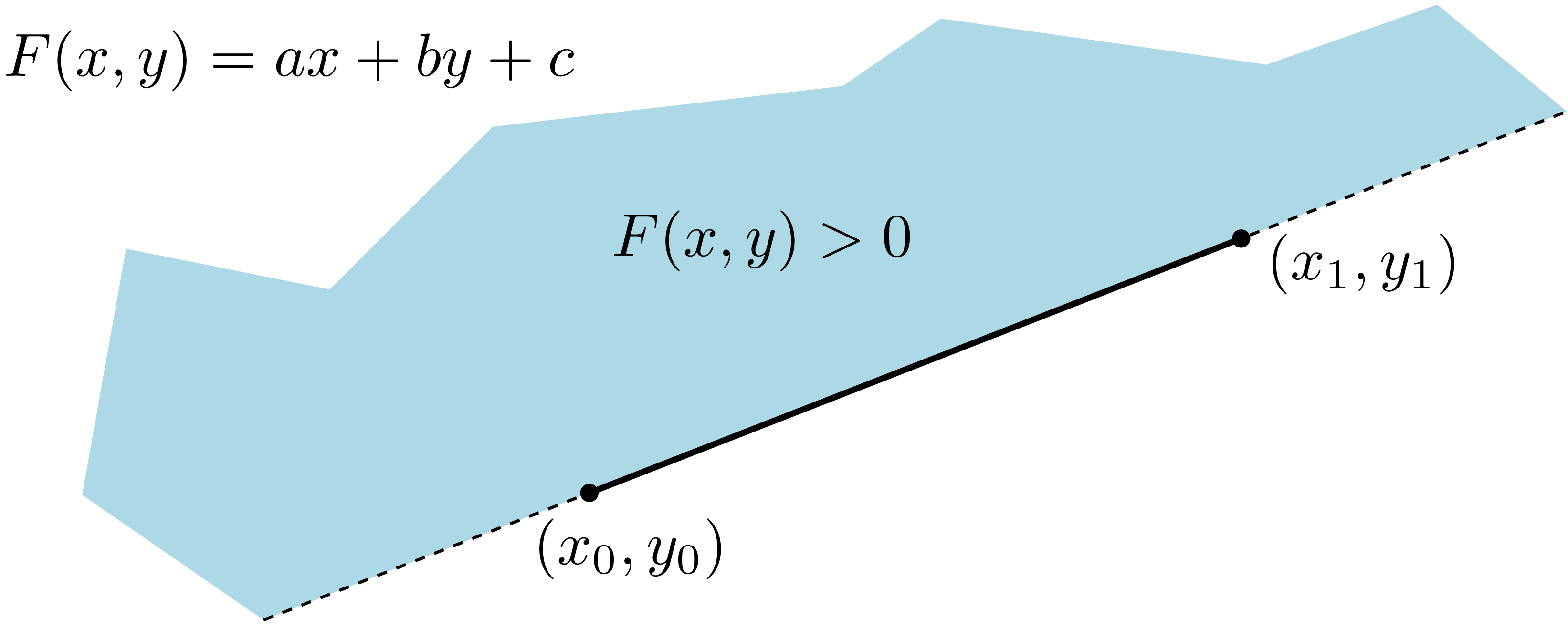
A equação do hiperplano afim gerado por um segmento de reta pode ser utilizada para decidir em qual região do plano um ponto p está localizado

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



Predicado de decisão.

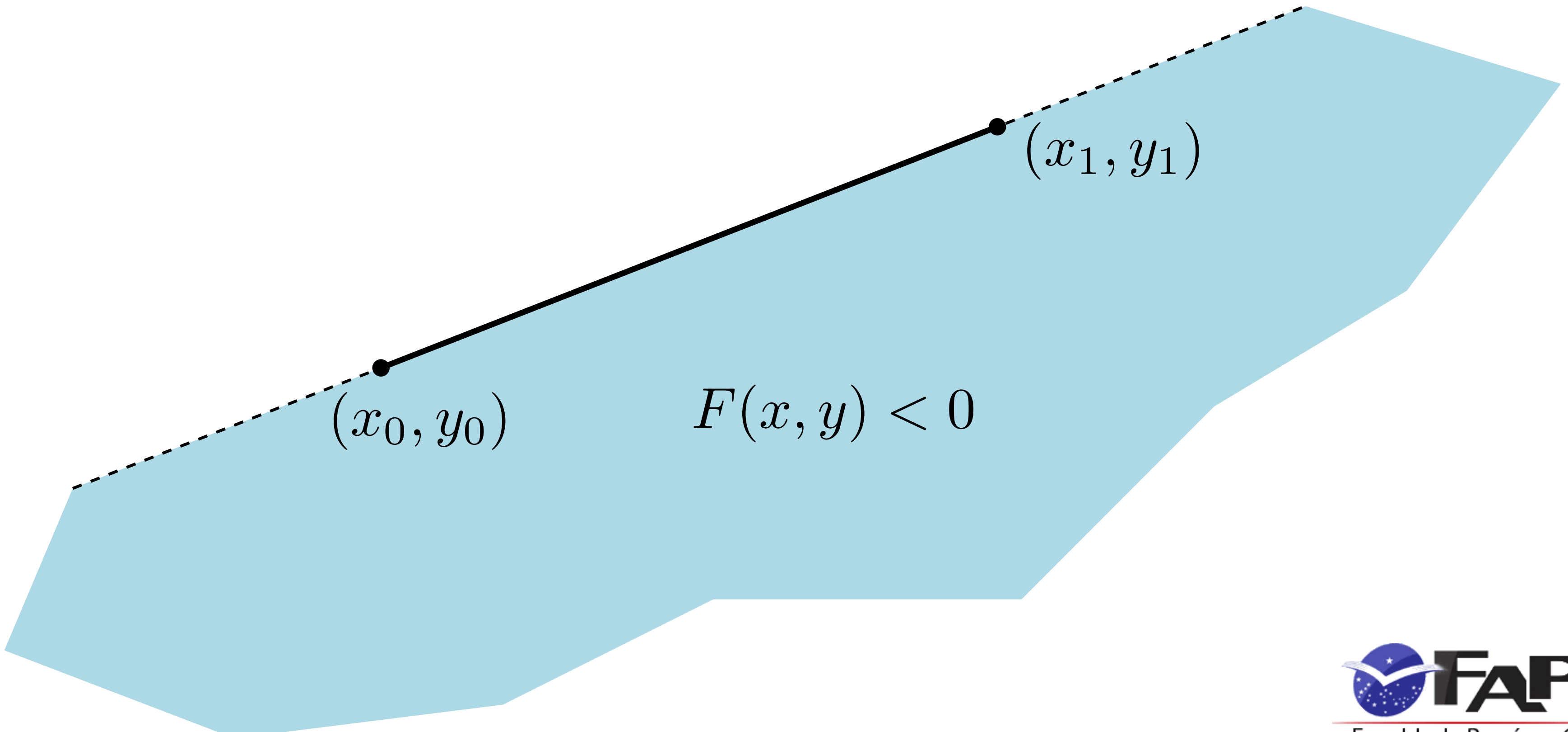
A equação do hiperplano afim gerado por um segmento de reta pode ser utilizada para decidir em qual região do plano um ponto p está localizado



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

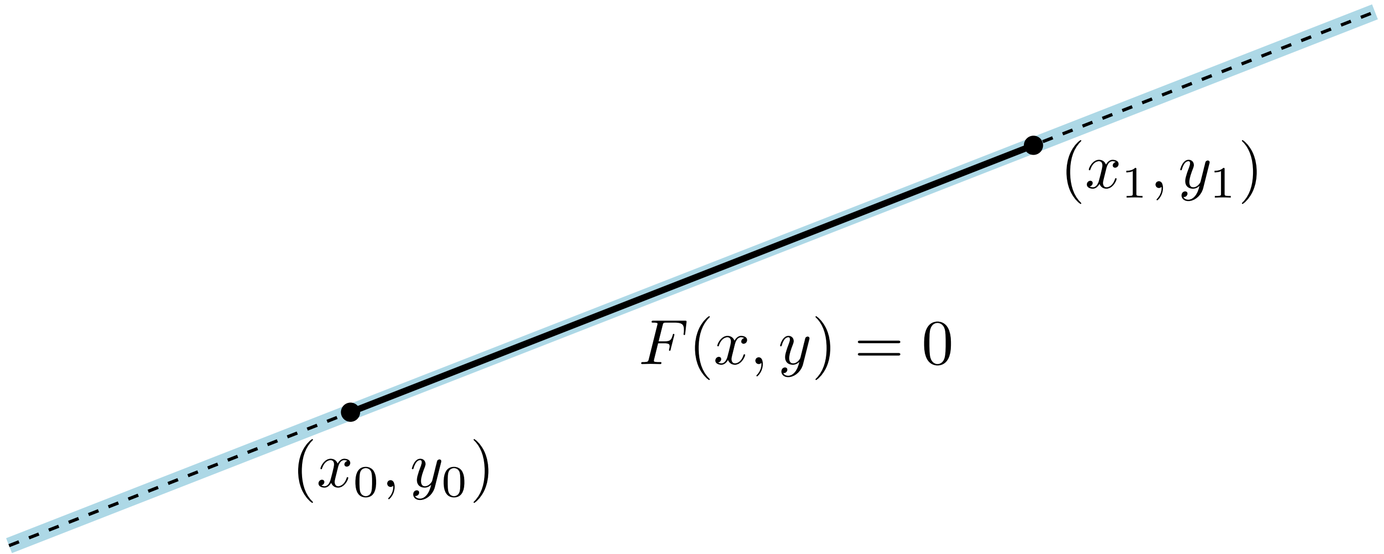
$$F(x, y) = ax + by + c$$



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

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



Relembrando o algoritmo de Bresenham

Predicado de decisão.

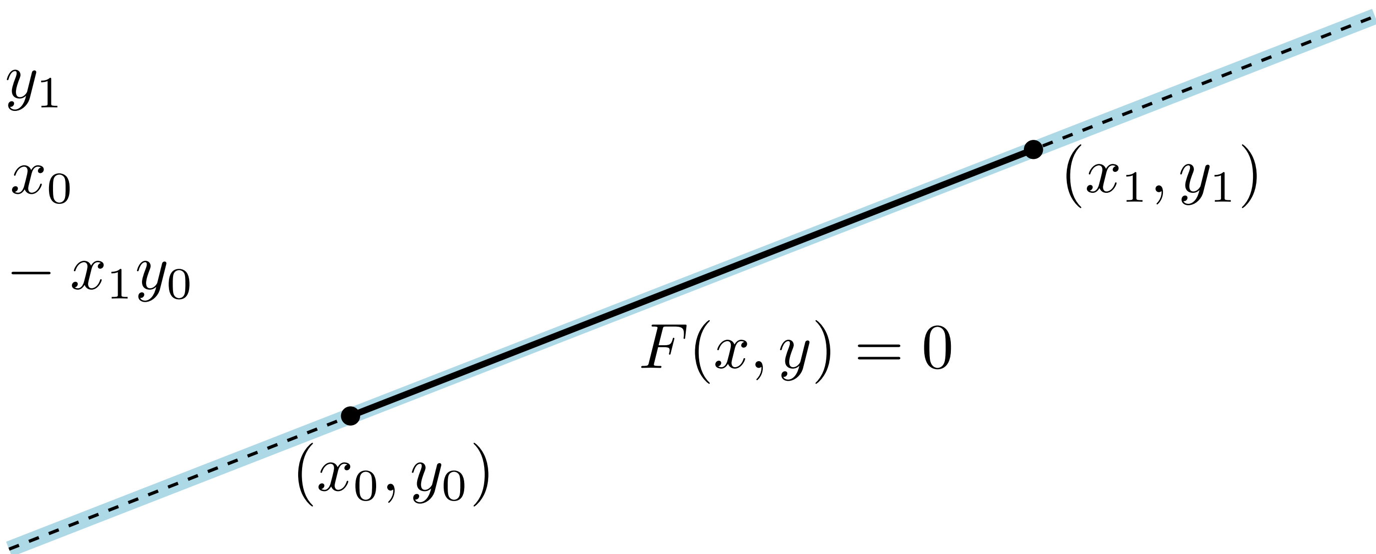
A equação do hiperplano afim gerado por um segmento de reta pode ser utilizada para decidir em qual região do plano um ponto p está localizado

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

$$a = y_0 - y_1$$

$$b = x_1 - x_0$$

$$c = x_0 y_1 - x_1 y_0$$

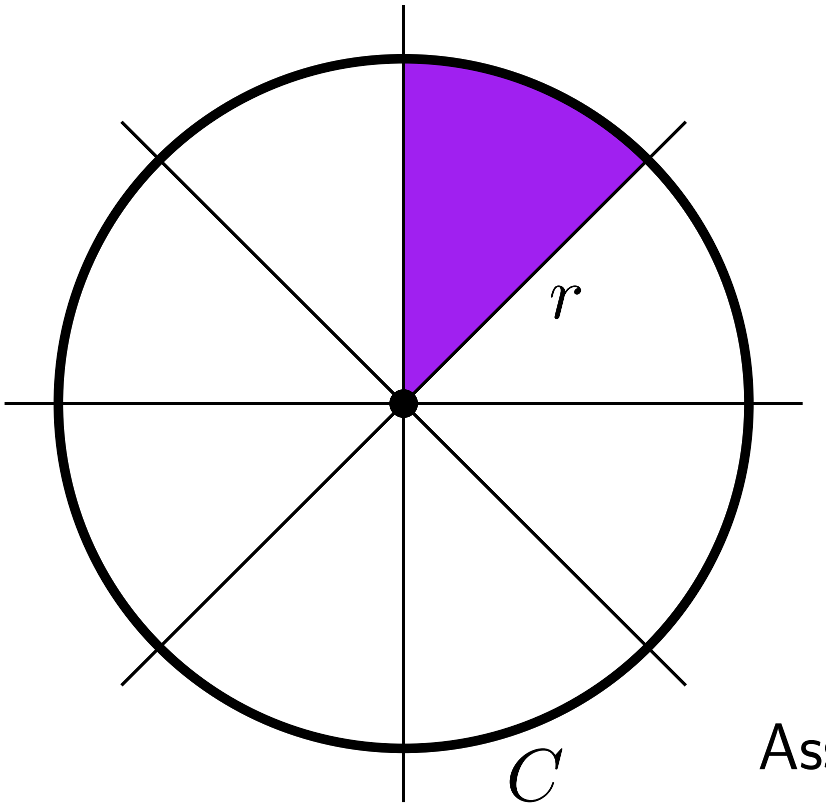


Podemos aplicar a ideia do algoritmo de Bresenham

Rasterização de círculos

Podemos aplicar a ideia do algoritmo de Bresenham

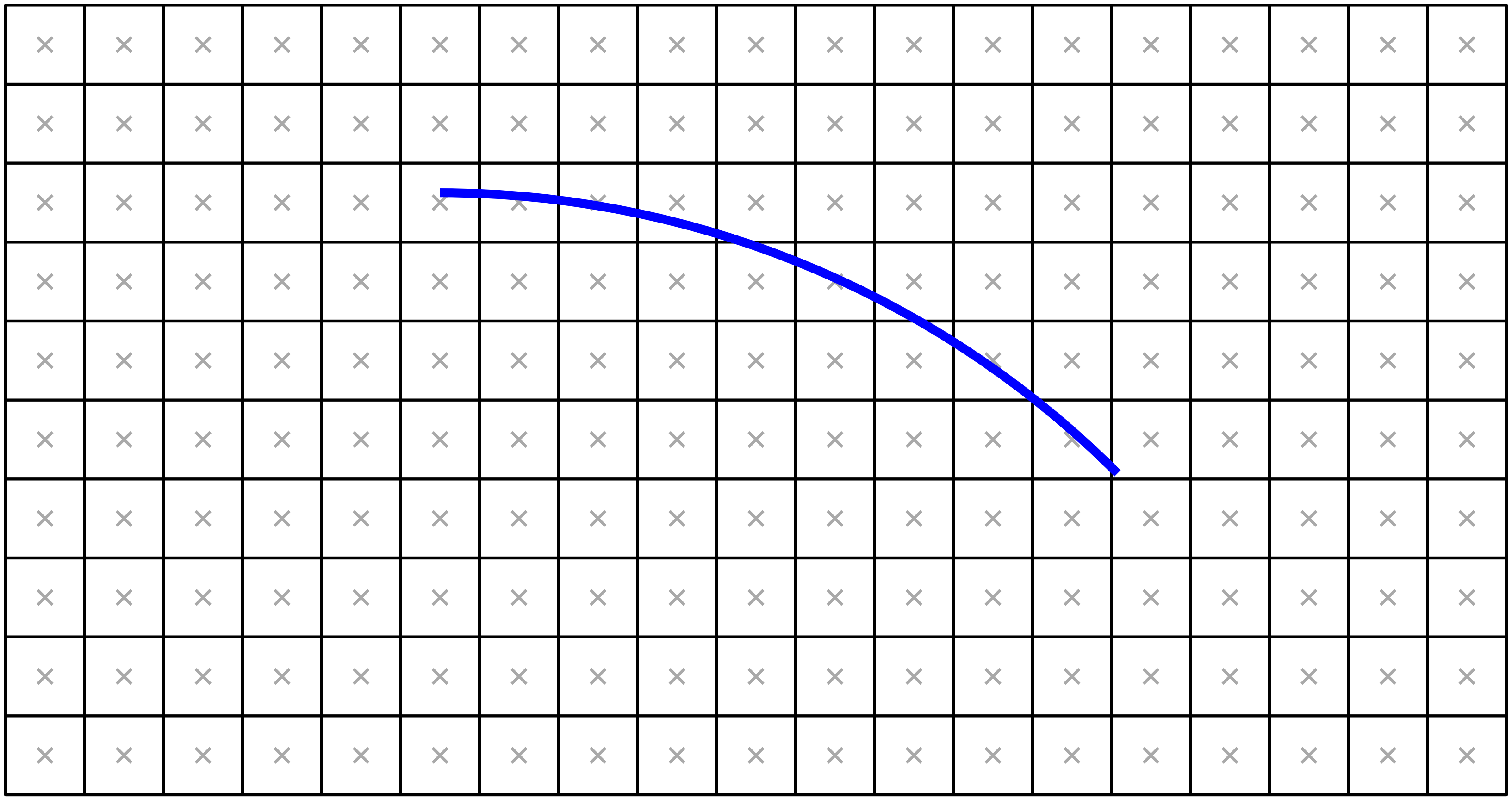
Basta considerar o $2^{\underline{o}}$ octante



Assumimos que o centro está em $(0, 0)$

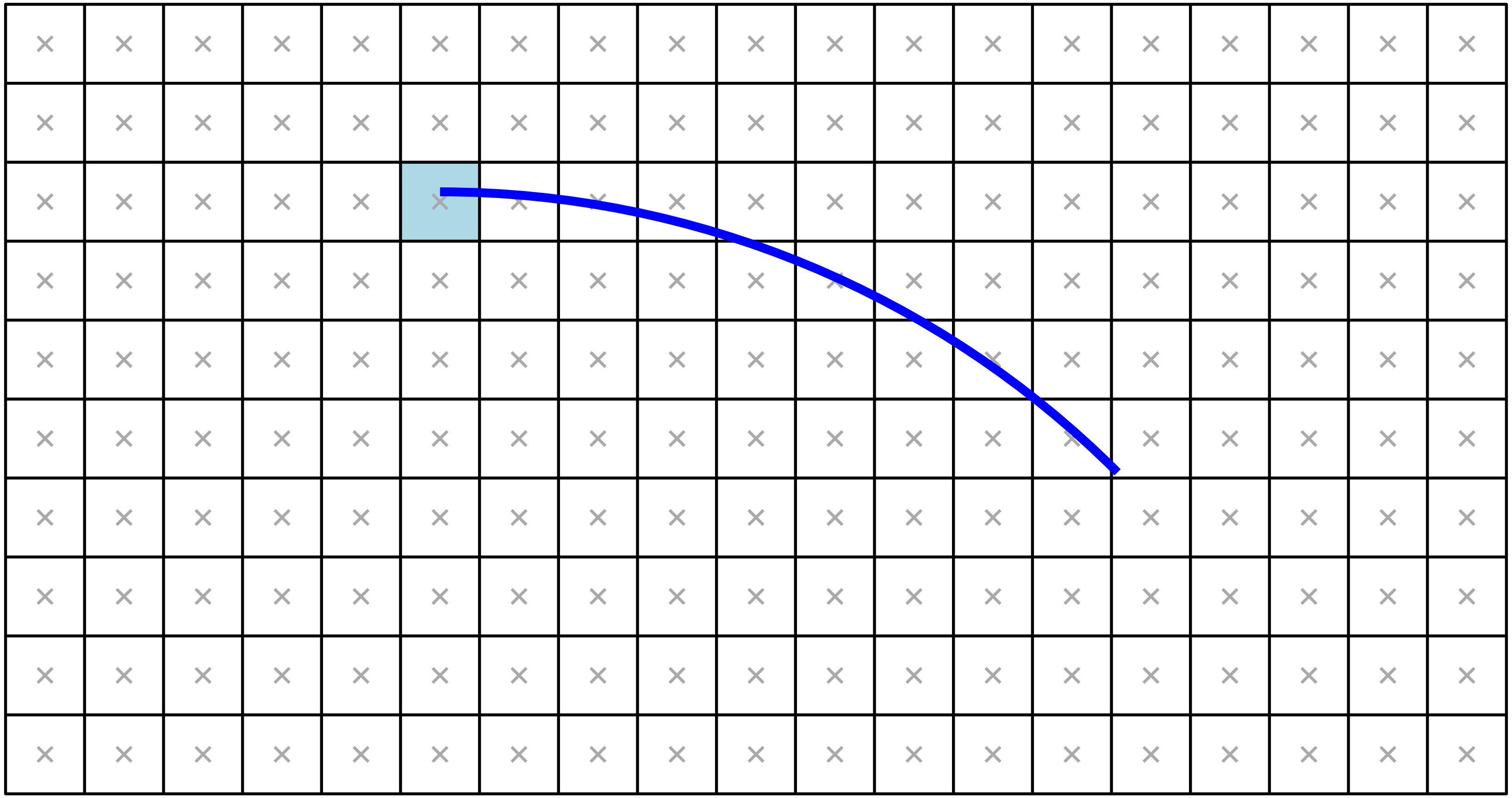
Podemos aplicar a ideia do algoritmo de Bresenham

Basta considerar o $2^{\underline{o}}$ octante



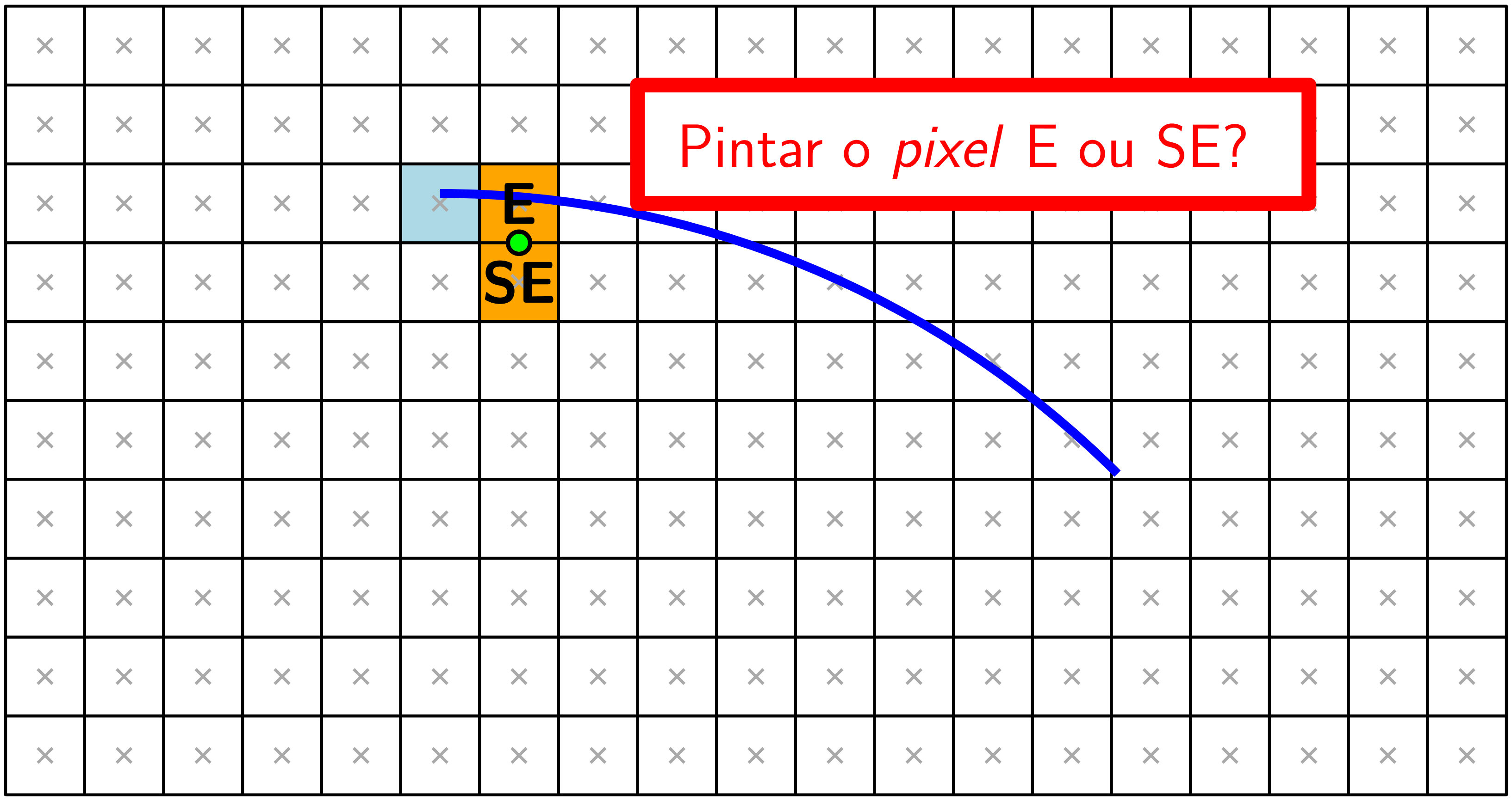
Podemos aplicar a ideia do algoritmo de Bresenham

Basta considerar o $2^{\underline{o}}$ octante



Podemos aplicar a ideia do algoritmo de Bresenham

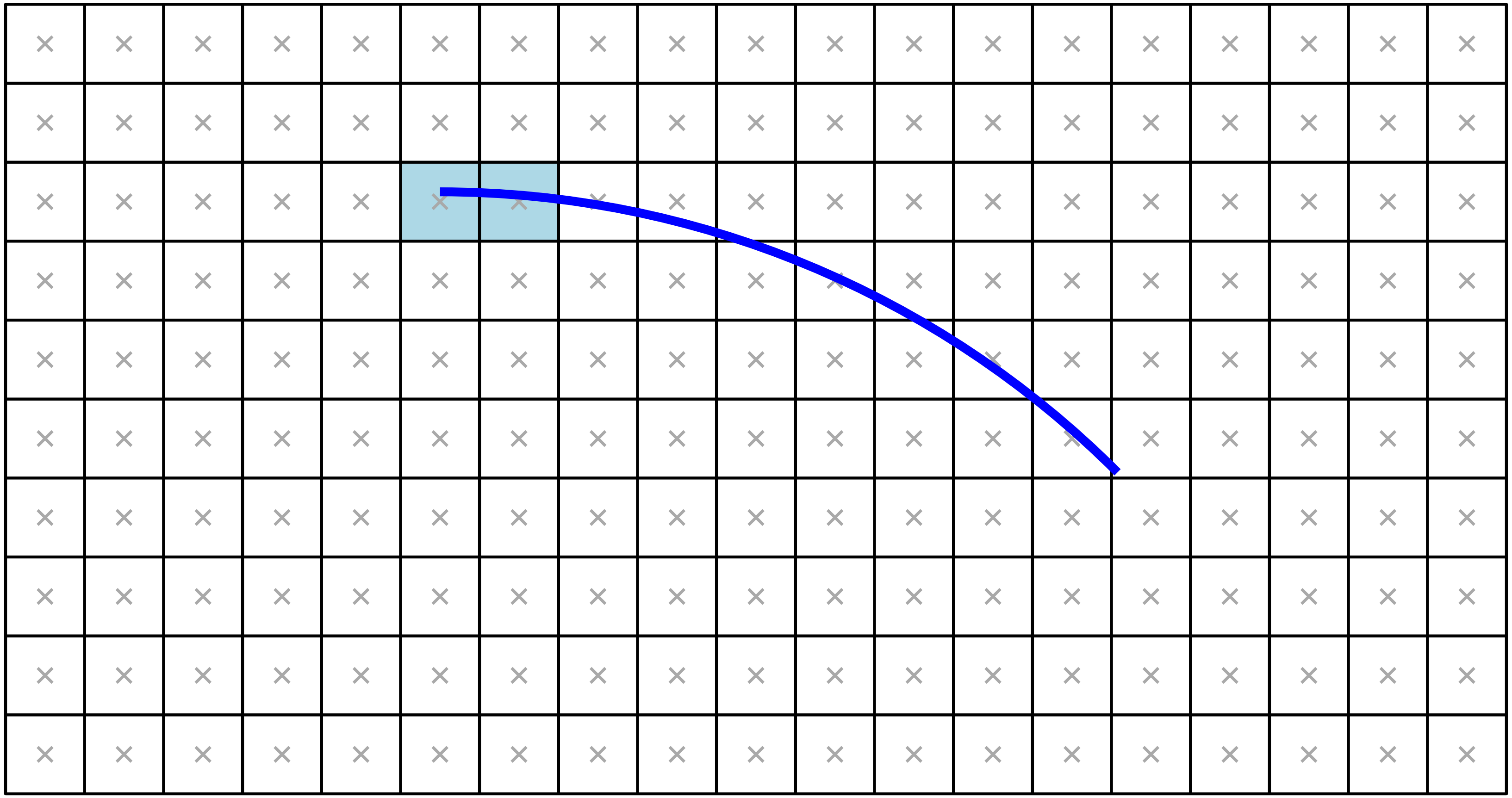
Basta considerar o $2^{\underline{o}}$ octante



Rasterização de círculos

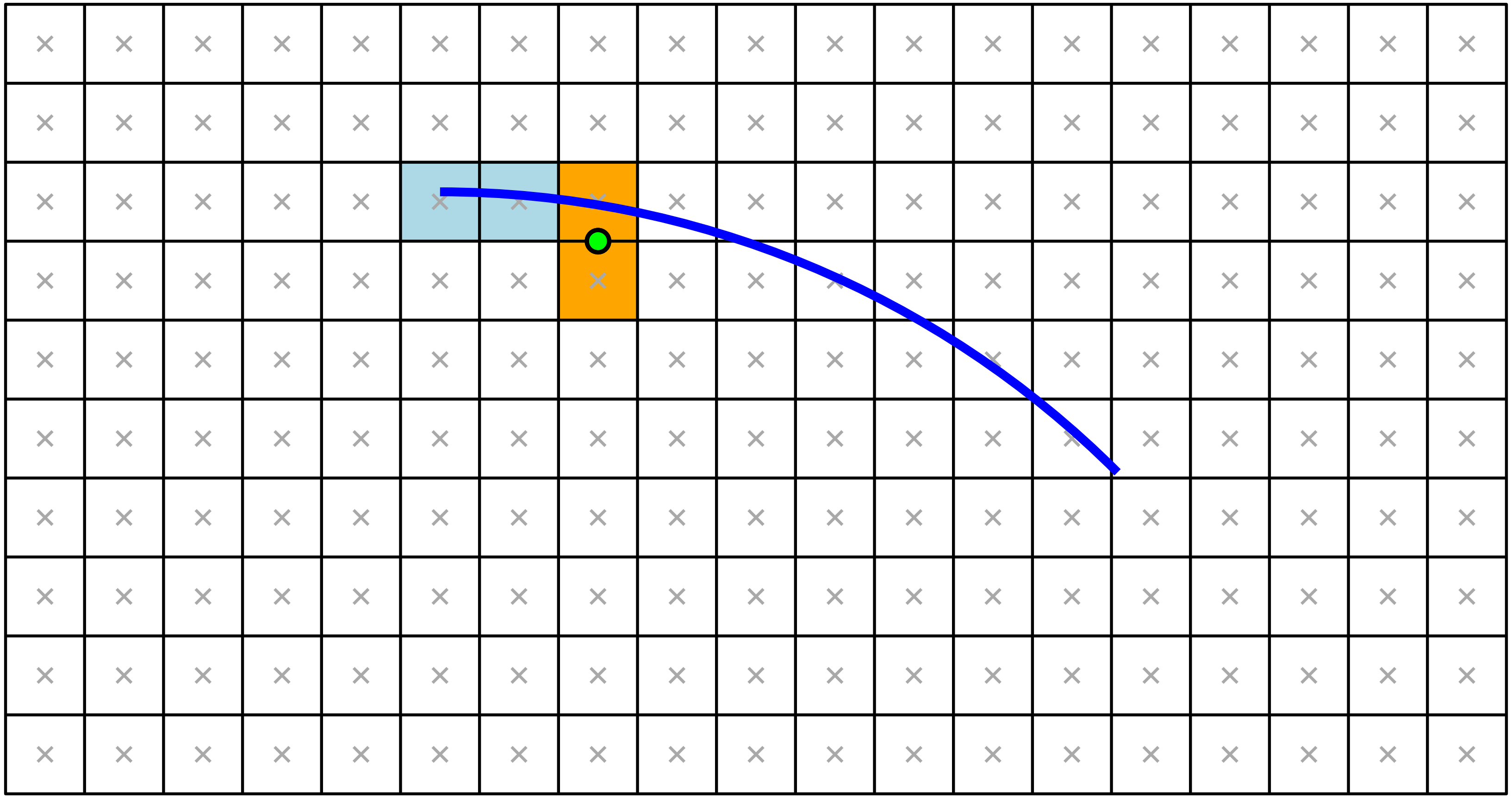
Podemos aplicar a ideia do algoritmo de Bresenham

Basta considerar o $2^{\underline{o}}$ octante



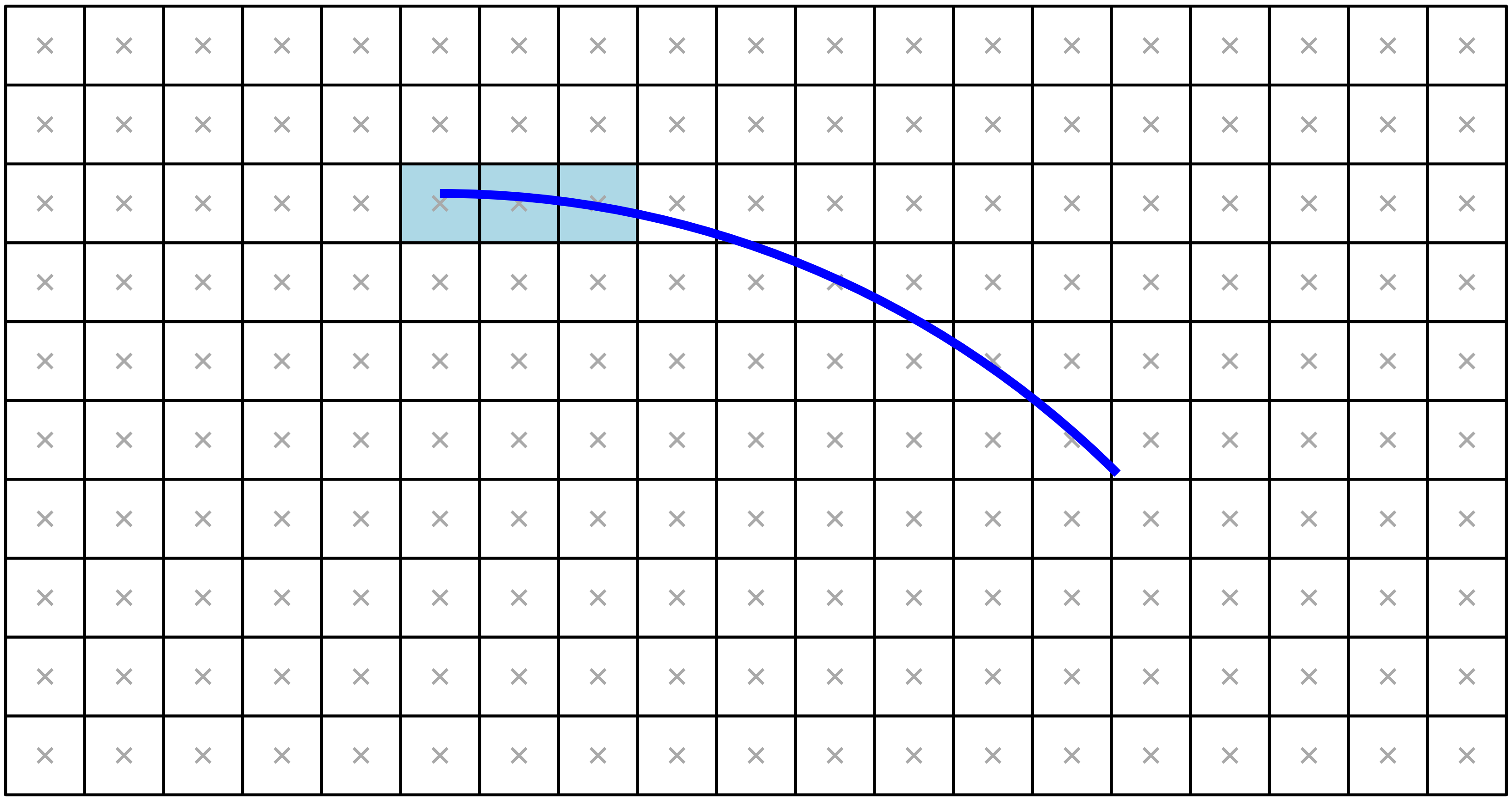
Podemos aplicar a ideia do algoritmo de Bresenham

Basta considerar o $2^{\underline{o}}$ octante



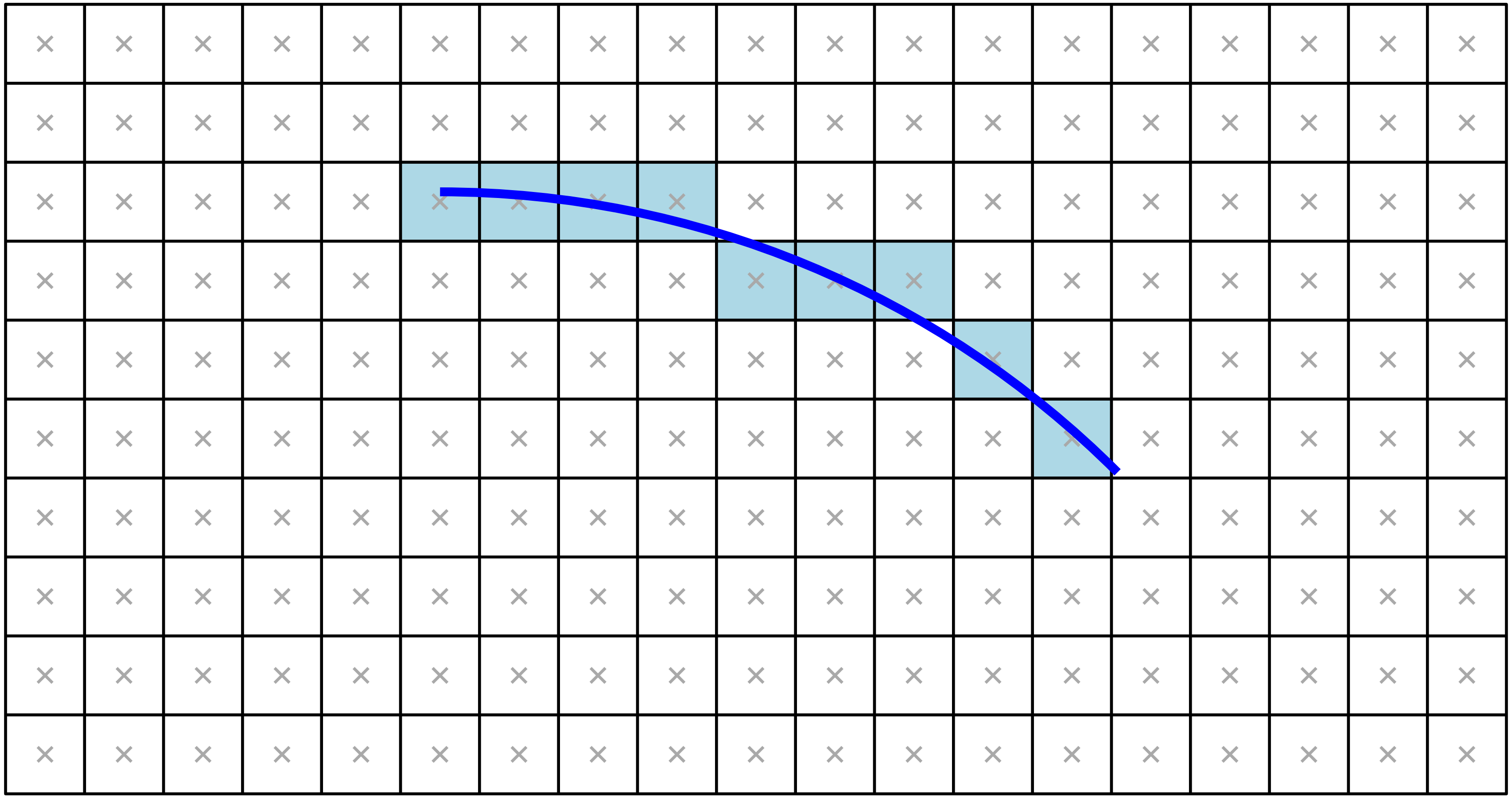
Podemos aplicar a ideia do algoritmo de Bresenham

Basta considerar o $2^{\underline{o}}$ octante



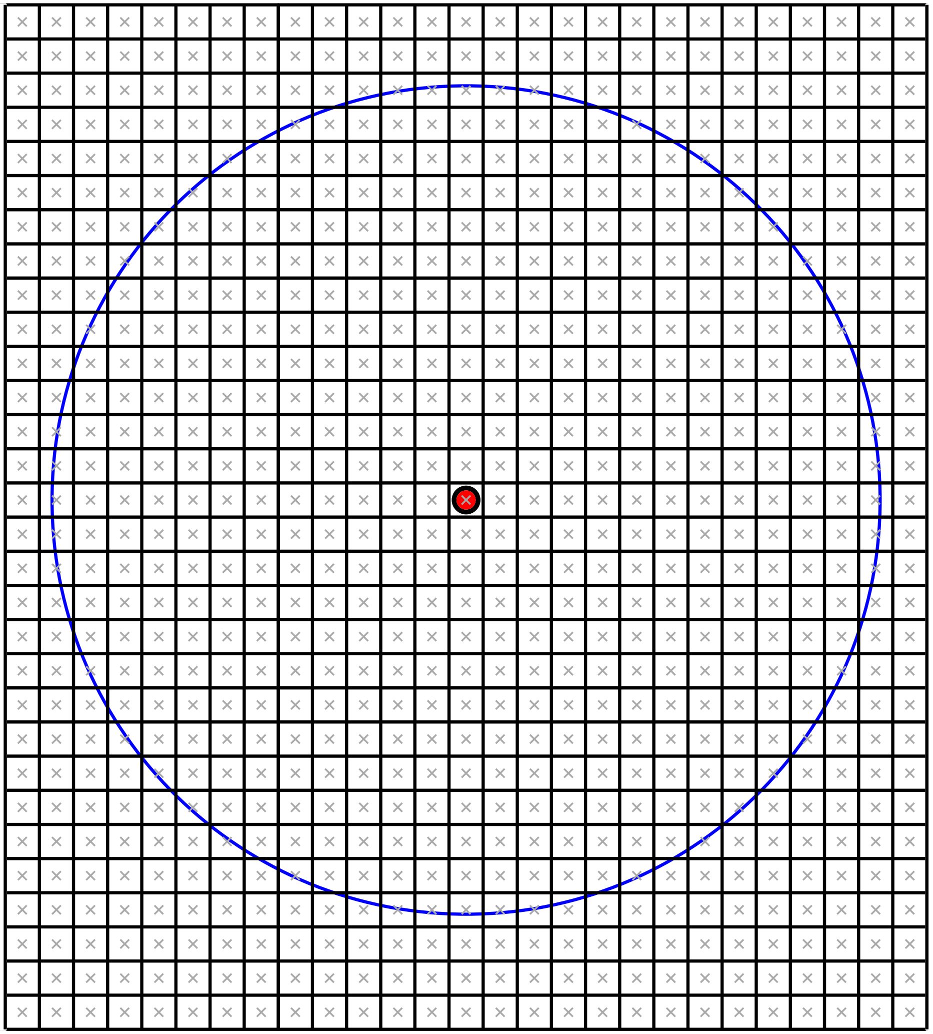
Podemos aplicar a ideia do algoritmo de Bresenham

Basta considerar o $2^{\underline{o}}$ octante



Podemos aplicar a ideia do algoritmo de Bresenham

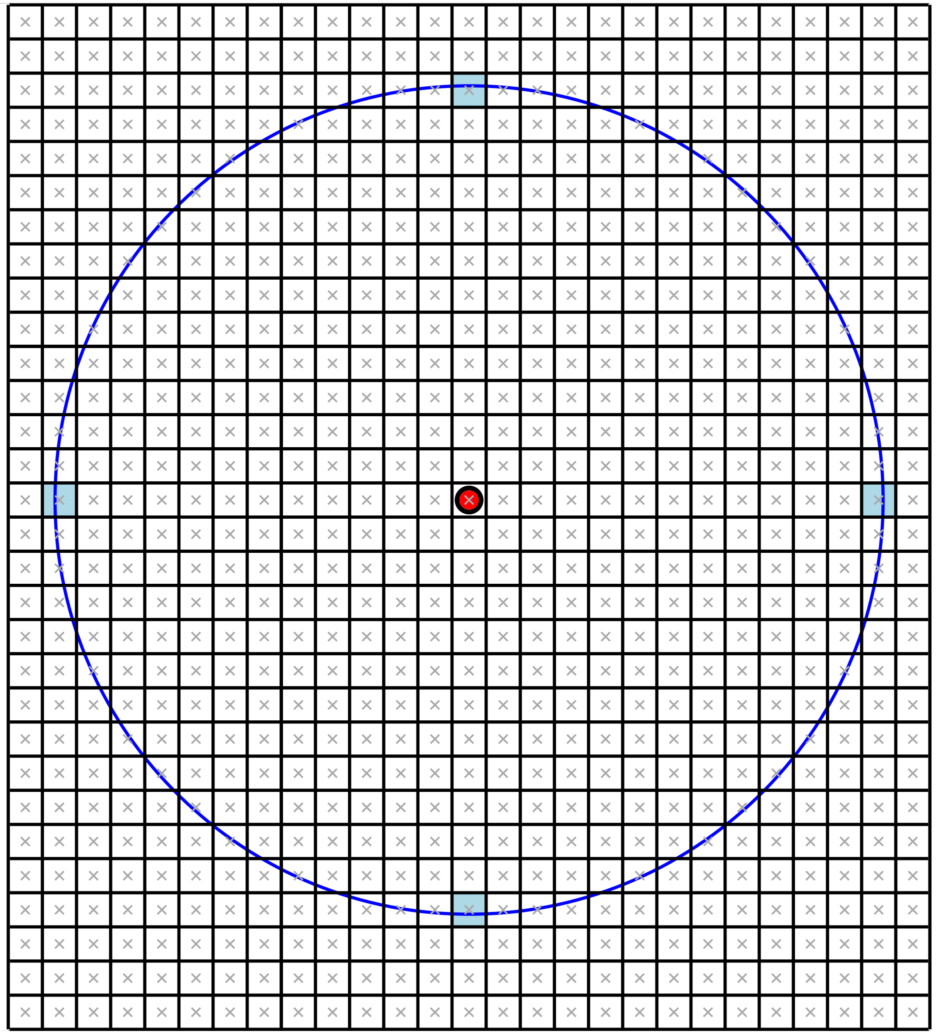
Ilustração do algoritmo com aplicação de simetria



Rasterização de círculos

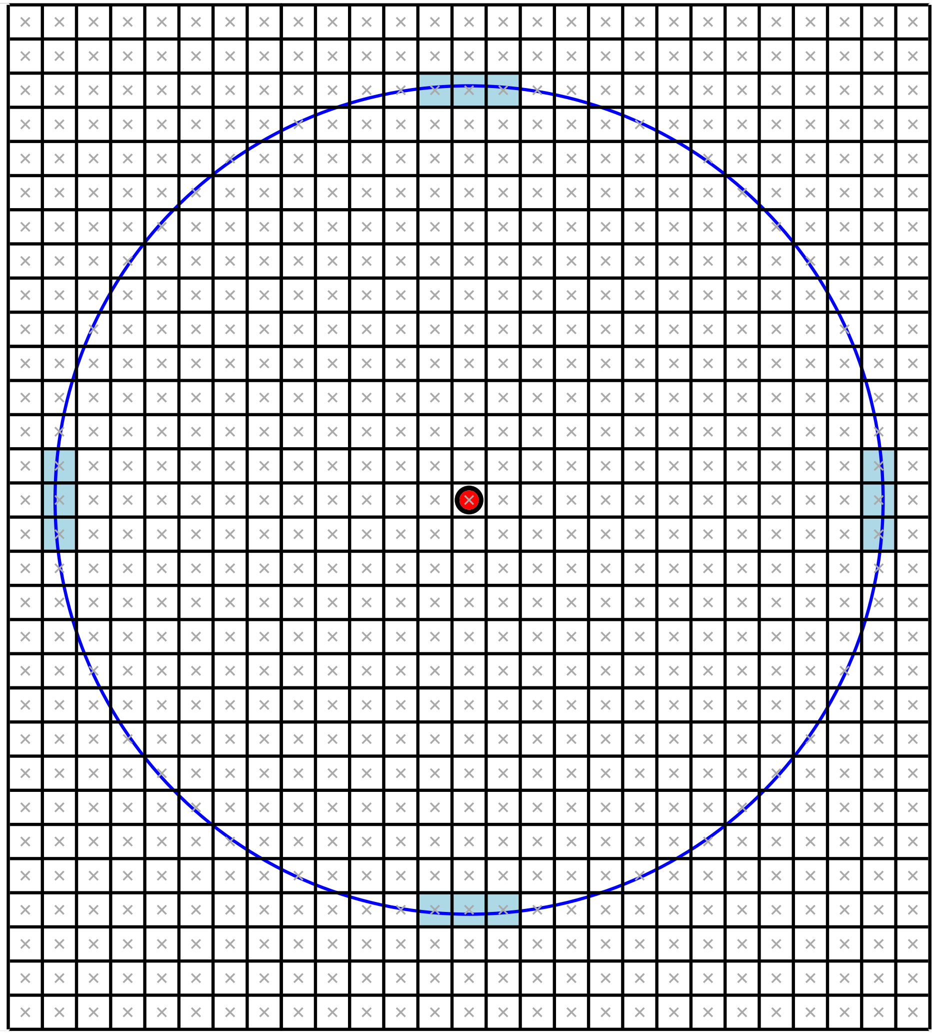
Podemos aplicar a ideia do algoritmo de Bresenham

Ilustração do algoritmo com aplicação de simetria



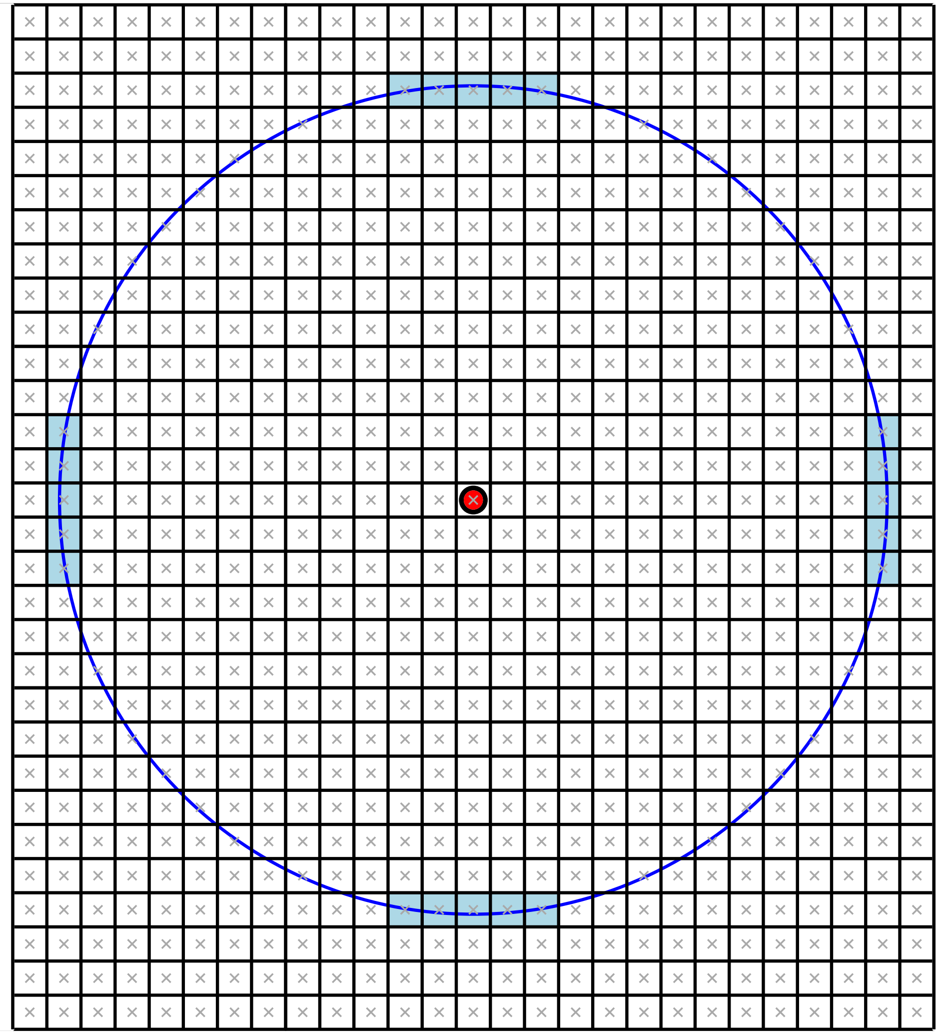
Podemos aplicar a ideia do algoritmo de Bresenham

Ilustração do algoritmo com aplicação de simetria



Podemos aplicar a ideia do algoritmo de Bresenham

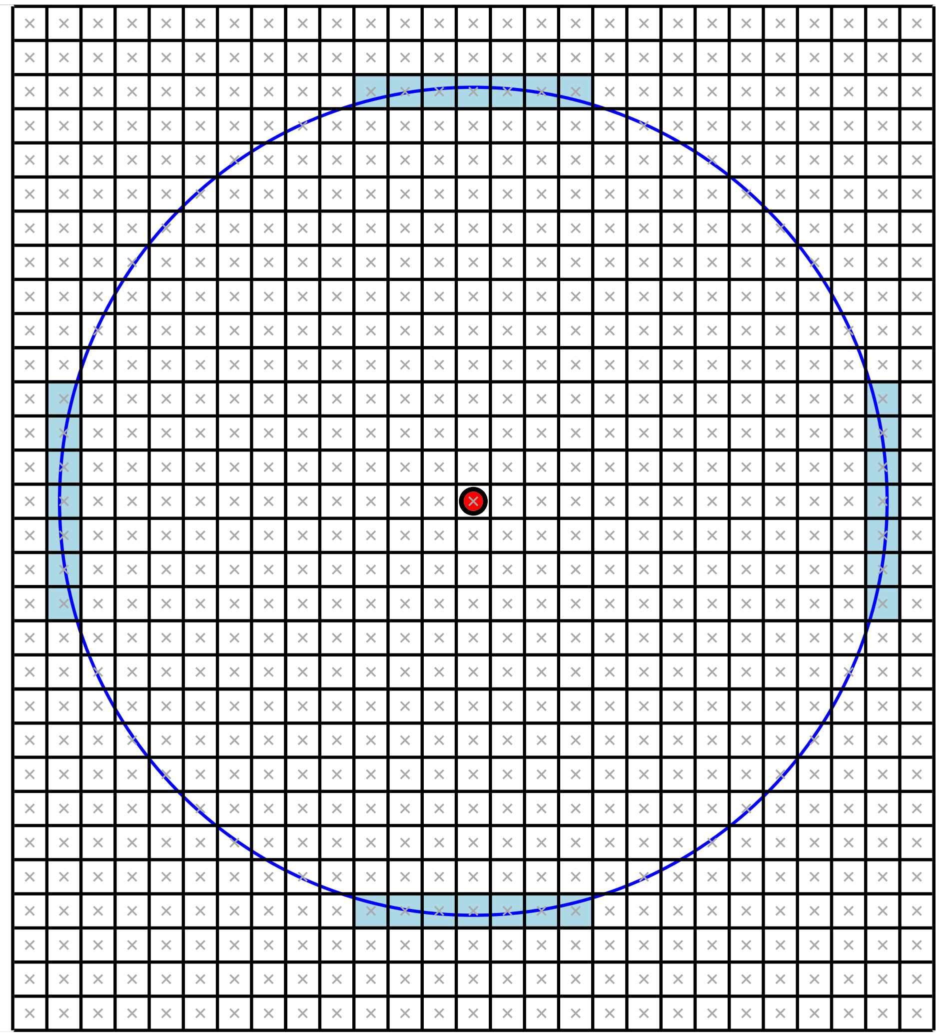
Ilustração do algoritmo com aplicação de simetria



Rasterização de círculos

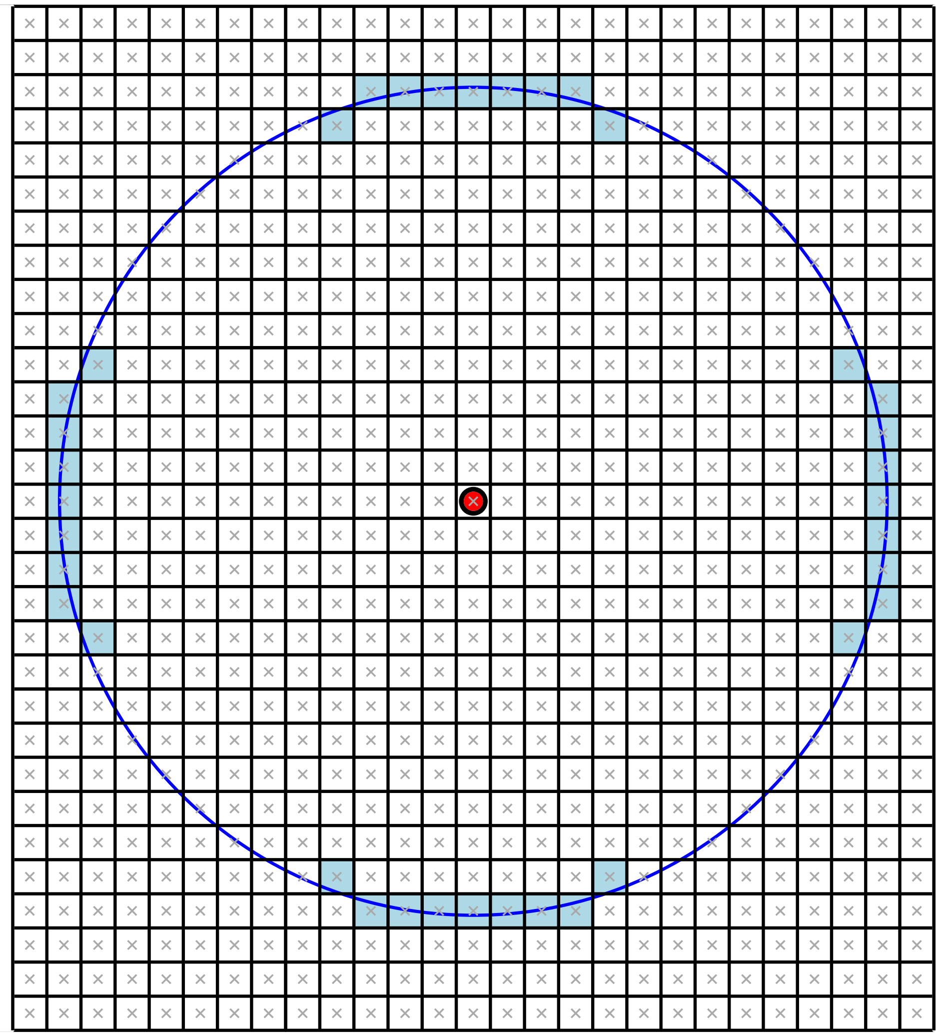
Podemos aplicar a ideia do algoritmo de Bresenham

Ilustração do algoritmo com aplicação de simetria



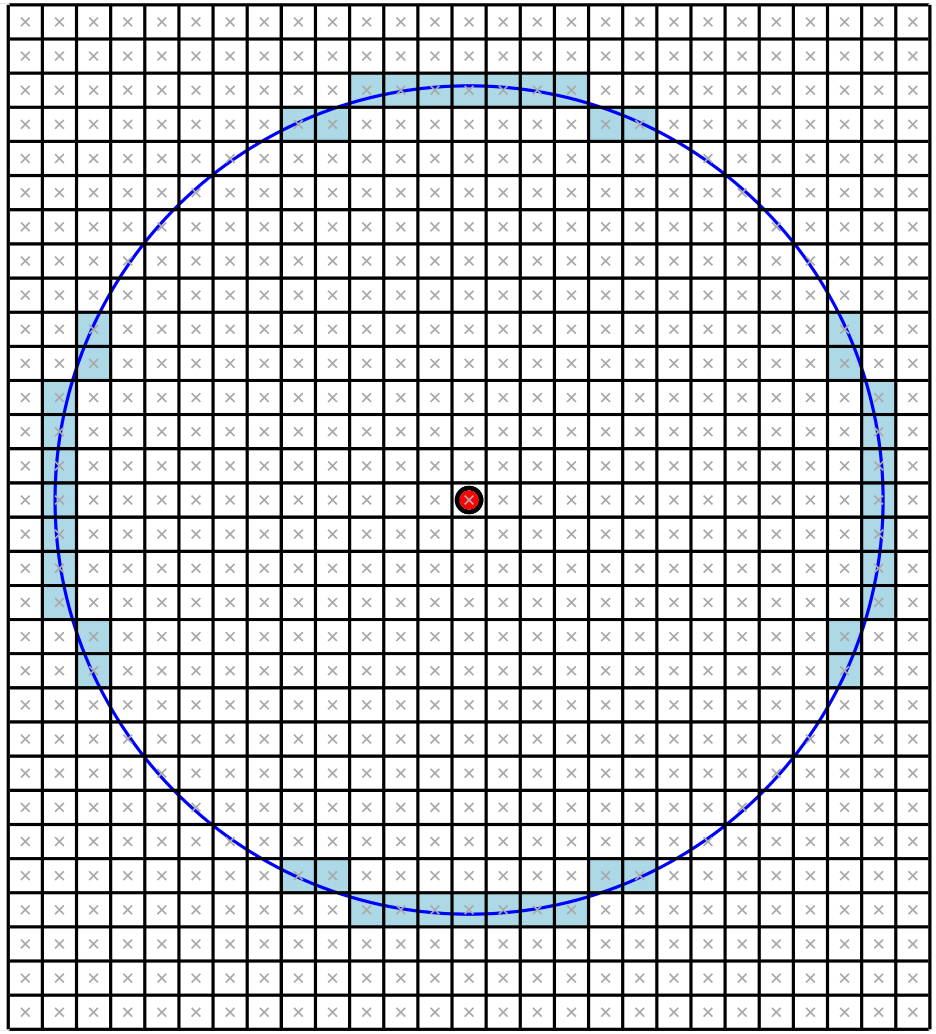
Podemos aplicar a ideia do algoritmo de Bresenham

Ilustração do algoritmo com aplicação de simetria



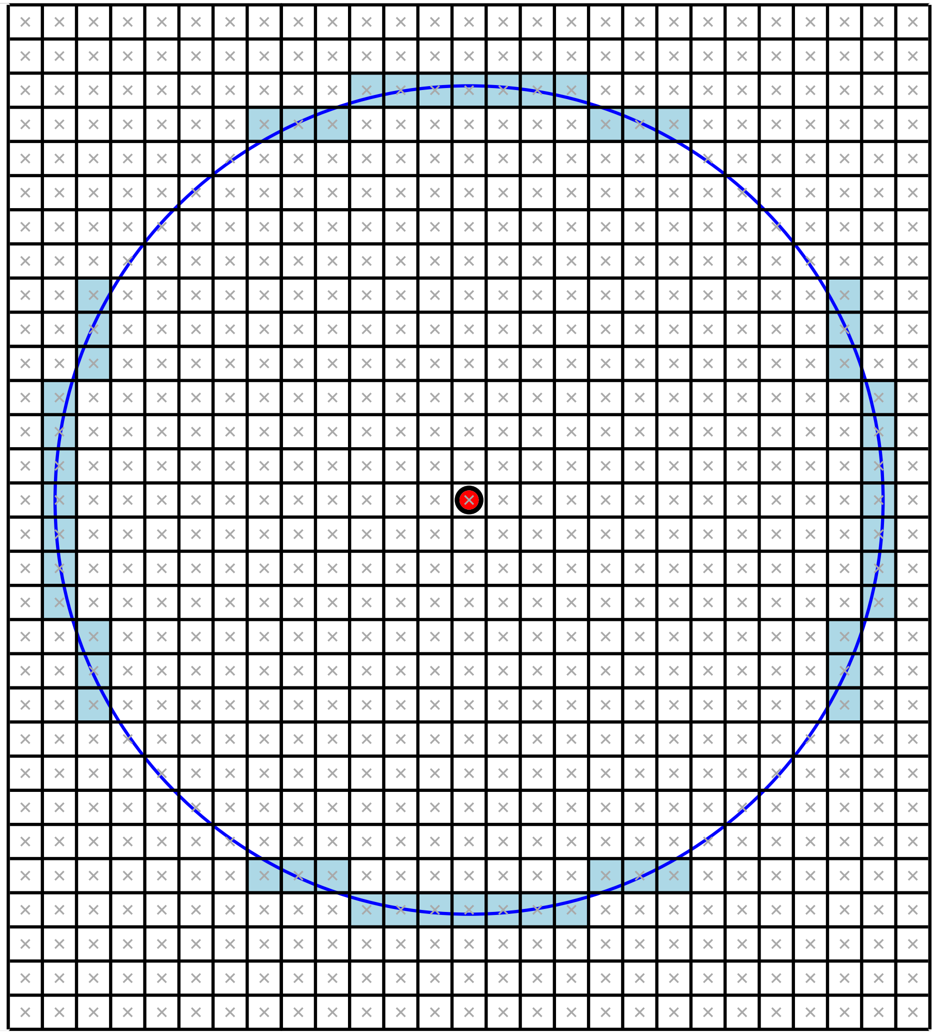
Podemos aplicar a ideia do algoritmo de Bresenham

Ilustração do algoritmo com aplicação de simetria



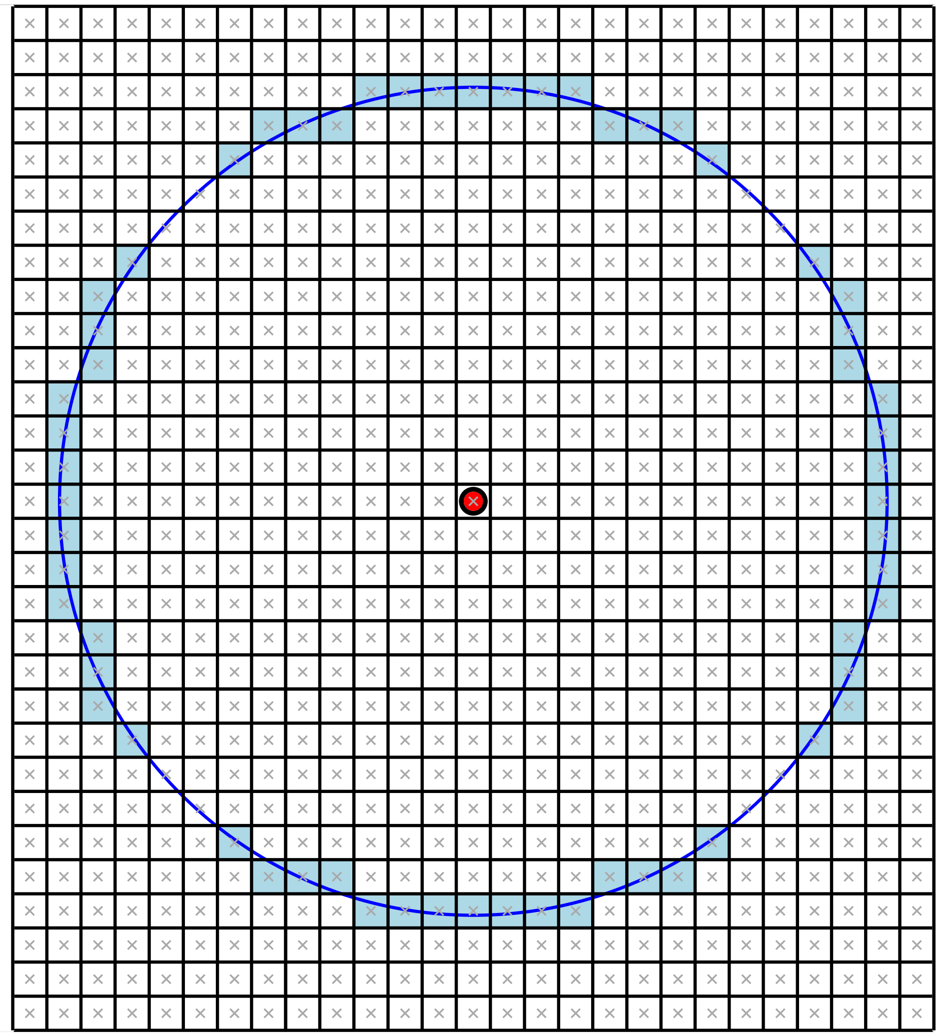
Podemos aplicar a ideia do algoritmo de Bresenham

Ilustração do algoritmo com aplicação de simetria



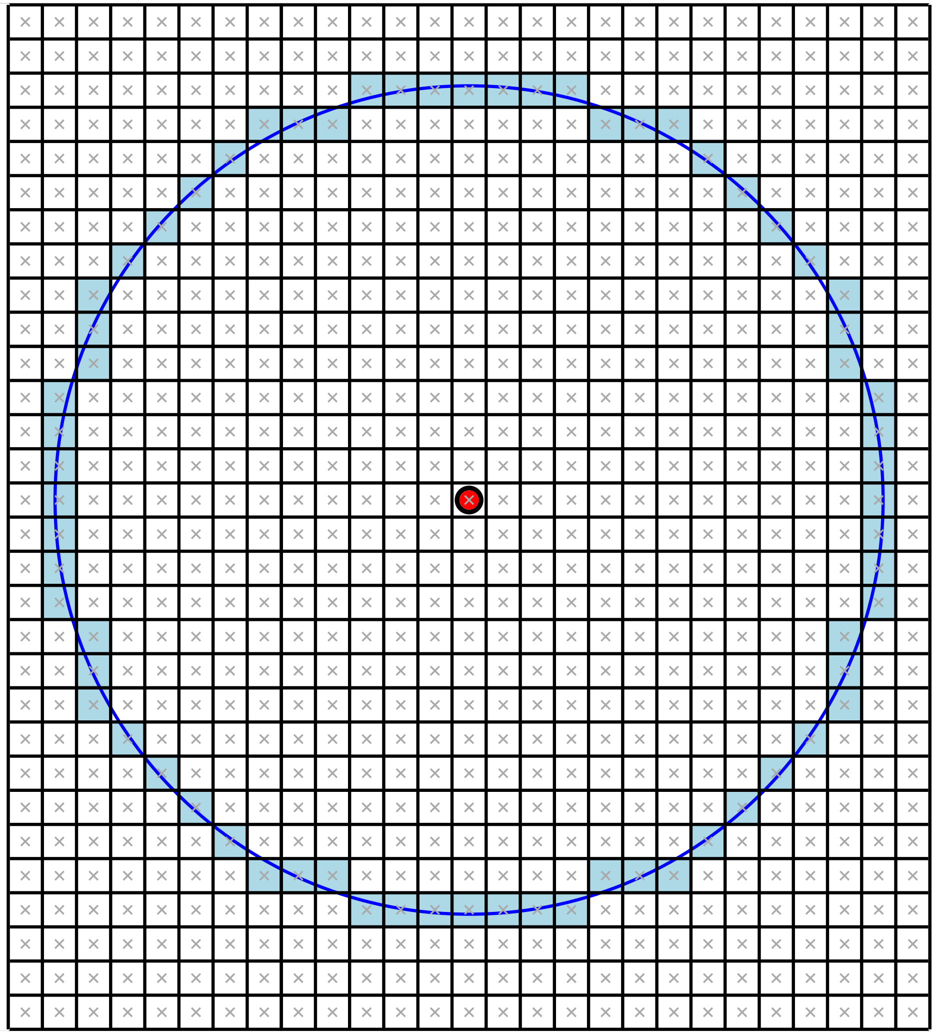
Podemos aplicar a ideia do algoritmo de Bresenham

Ilustração do algoritmo com aplicação de simetria



Podemos aplicar a ideia do algoritmo de Bresenham

Ilustração do algoritmo com aplicação de simetria

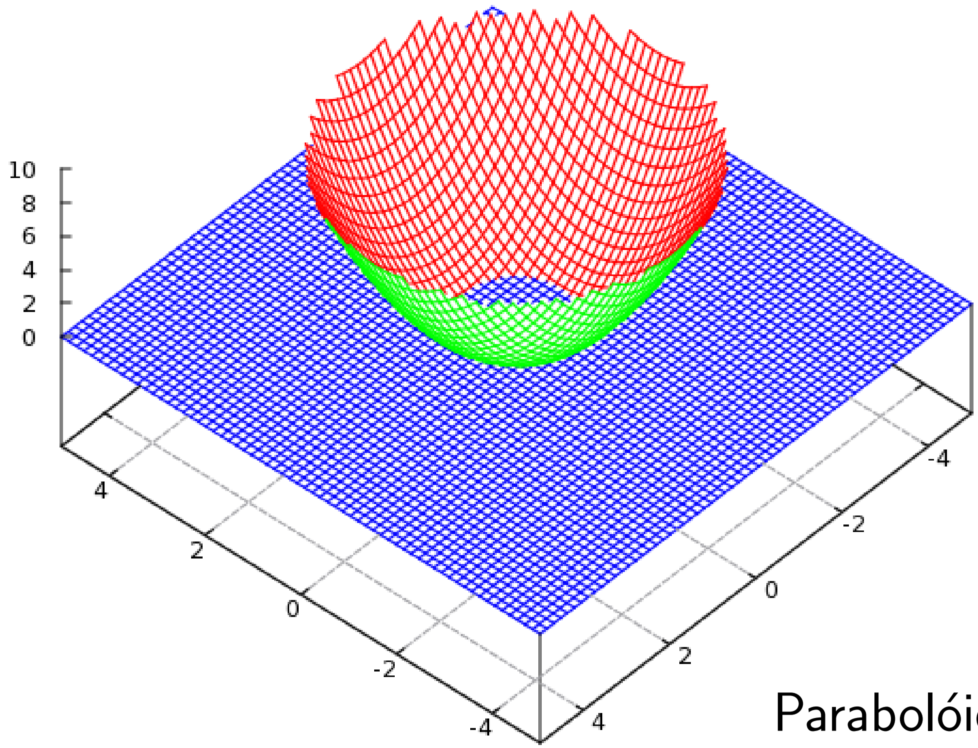


Podemos aplicar a ideia do algoritmo de Bresenham

Basta considerar o 2º octante

Com a função de decisão:

$$F(x, y) = x^2 + y^2 - r^2$$



Parabolóide

$F(p) = 0 \implies p$ está sobre F

$F(p) < 0 \implies p$ está dentro

$F(p) > 0 \implies p$ está fora

Assumimos que o centro está em $(0, 0)$

Podemos aplicar a ideia do algoritmo de Bresenham

Se formos para o pixel E

$$F(x_0 + 2, y_0 - 1/2) = F(x_0 + 1, y - 1/2) + 2x_0 + 3$$

Se formos para o pixel SE

$$F(x_0 + 2, y_0 - 3/2) = F(x_0 + 1, y - 1/2) + 2x_0 - 2y_0 + 5$$

onde o primeiro ponto médio será em $(1, r - 1/2)$**.**

$$F(x_0 + 1, y_0 - 1/2) = F(1, r - 1/2) = 5/4 - r$$

Preenchimento de regiões

Dada uma região *conexa* delimitada por um conjunto de *pixels* no espaço da imagem, como podemos preencher seu interior, dado um *pixel* "semente"?

Preenchimento de regiões

Dada uma região *conexa* delimitada por um conjunto de *pixels* no espaço da imagem, como podemos preencher seu interior, dado um *pixel* "semente"?

O que é uma região *conexa*?

Preenchimento de regiões

Dada uma região *conexa* delimitada por um conjunto de *pixels* no espaço da imagem, como podemos preencher seu interior, dado um *pixel* “semente”?

O que é uma região *conexa*?

É toda região na qual quaisquer dois *pixels* em seu interior podem ser conectados por um caminho através de seus *pixels* vizinhos internos

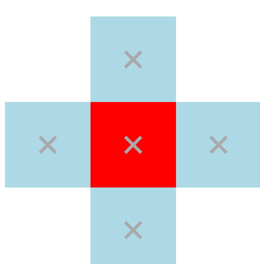
Preenchimento de regiões

Dada uma região *conexa* delimitada por um conjunto de *pixels* no espaço da imagem, como podemos preencher seu interior, dado um *pixel* “semente”?

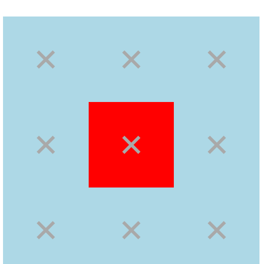
O que é uma região *conexa*?

> É toda região na qual quaisquer dois *pixels* em seu interior podem ser conectados por um caminho através de seus *pixels* vizinhos internos

Podemos considerar dois modelos de vizinhança:

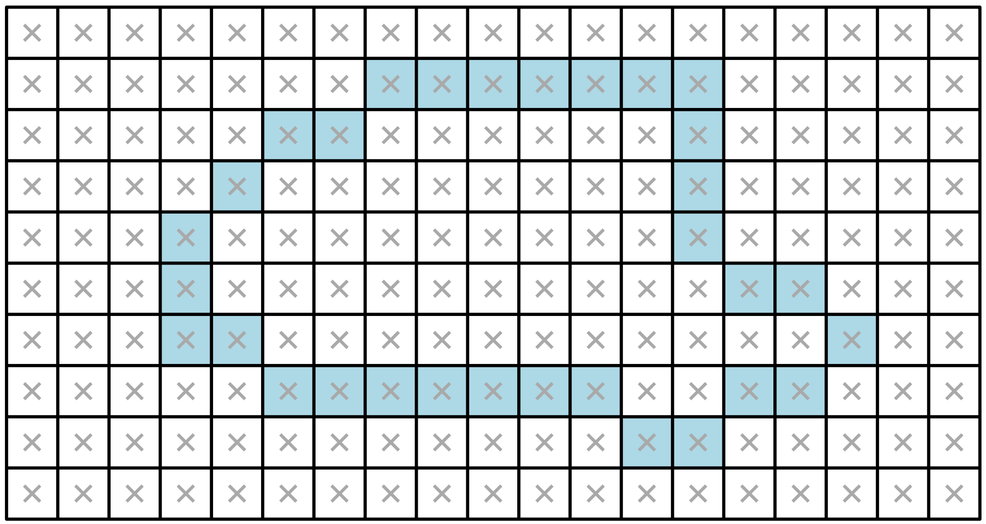


4-vizinhança

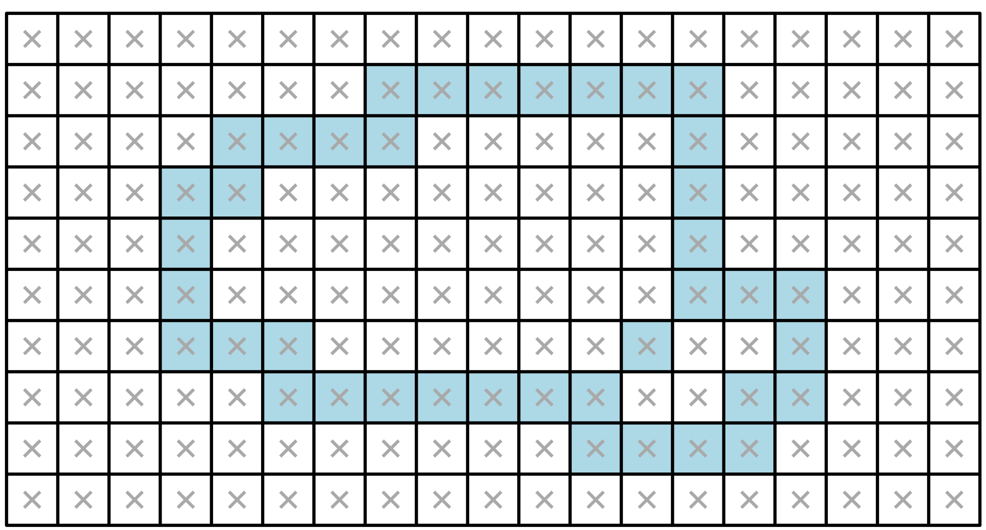


8-vizinhança

Dada uma região *conexa* delimitada por um conjunto de *pixels* no espaço da imagem, como podemos preencher seu interior, dado um *pixel* “semente”?



- Contorno 8-conexo
- Interior 4-conexo



- Contorno 4-conexo
- Interior 8-conexo

Preenchimento de regiões

Dada uma região *conexa* delimitada por um conjunto de *pixels* no espaço da imagem, como podemos preencher seu interior, dado um *pixel* "semente"?

```
flood_fill_4(x, y, cor, cor_desejada) {
  Se pixel(x, y).cor ≠ cor, então:
    Retorne;
  pixel(x, y).cor ← cor_desejada;
  flood_fill(x - 1, y, cor, cor_desejada);
  flood_fill(x + 1, y, cor, cor_desejada);
  flood_fill(x, y + 1, cor, cor_desejada);
  flood_fill(x, y - 1, cor, cor_desejada);
  Retorne;
}
```

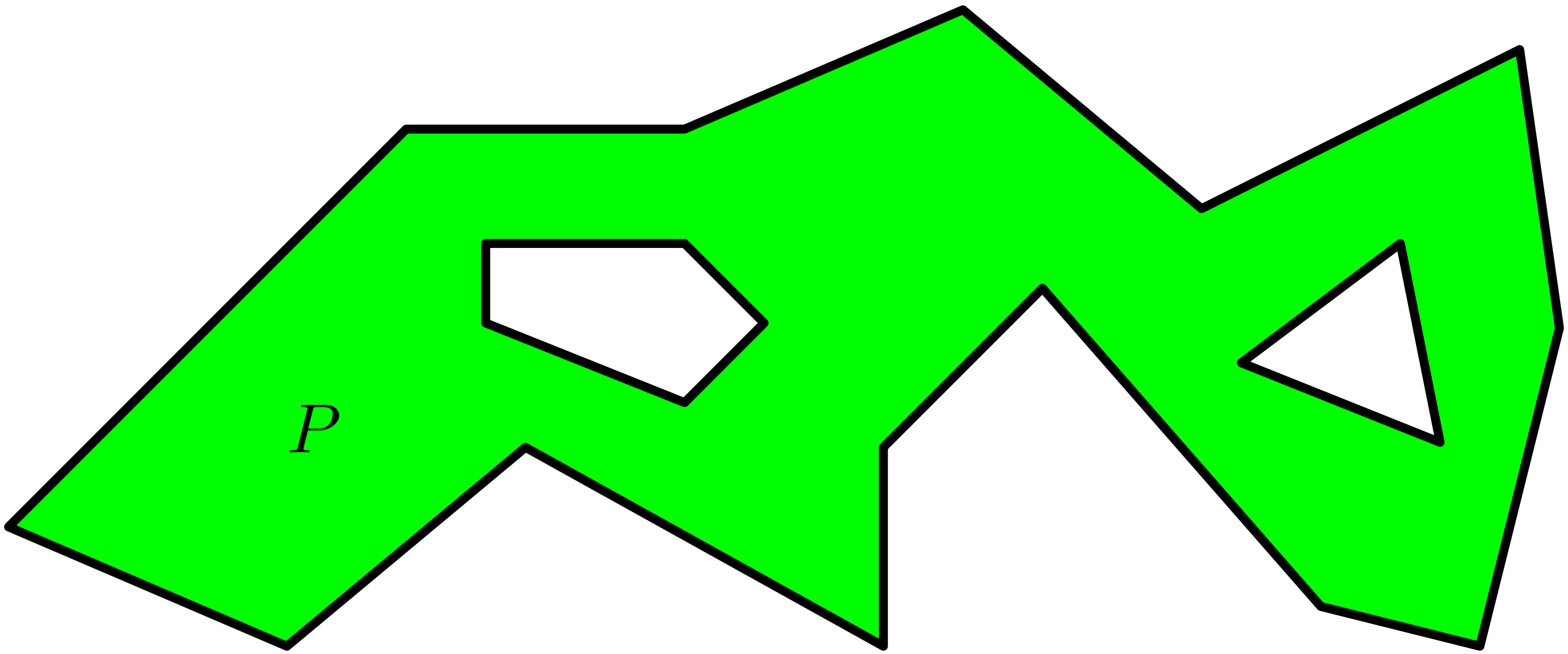
Preenchimento de regiões

Dada uma região *conexa* delimitada por um conjunto de *pixels* no espaço da imagem, como podemos preencher seu interior, dado um *pixel* "semente"?

```
flood_fill_4(x, y, cor, cor_desejada) {
  Se pixel(x, y).cor ≠ cor, então:
    Retorne;
  pixel(x, y).cor ← cor_desejada;
  flood_fill(x - 1, y, cor, cor_desejada);
  flood_fill(x + 1, y, cor, cor_desejada);
  flood_fill(x, y + 1, cor, cor_desejada);
  flood_fill(x, y - 1, cor, cor_desejada);
  Retorne;
}
```

É possível melhorar a performance com um *flag* para designar se um *pixel* já foi testado

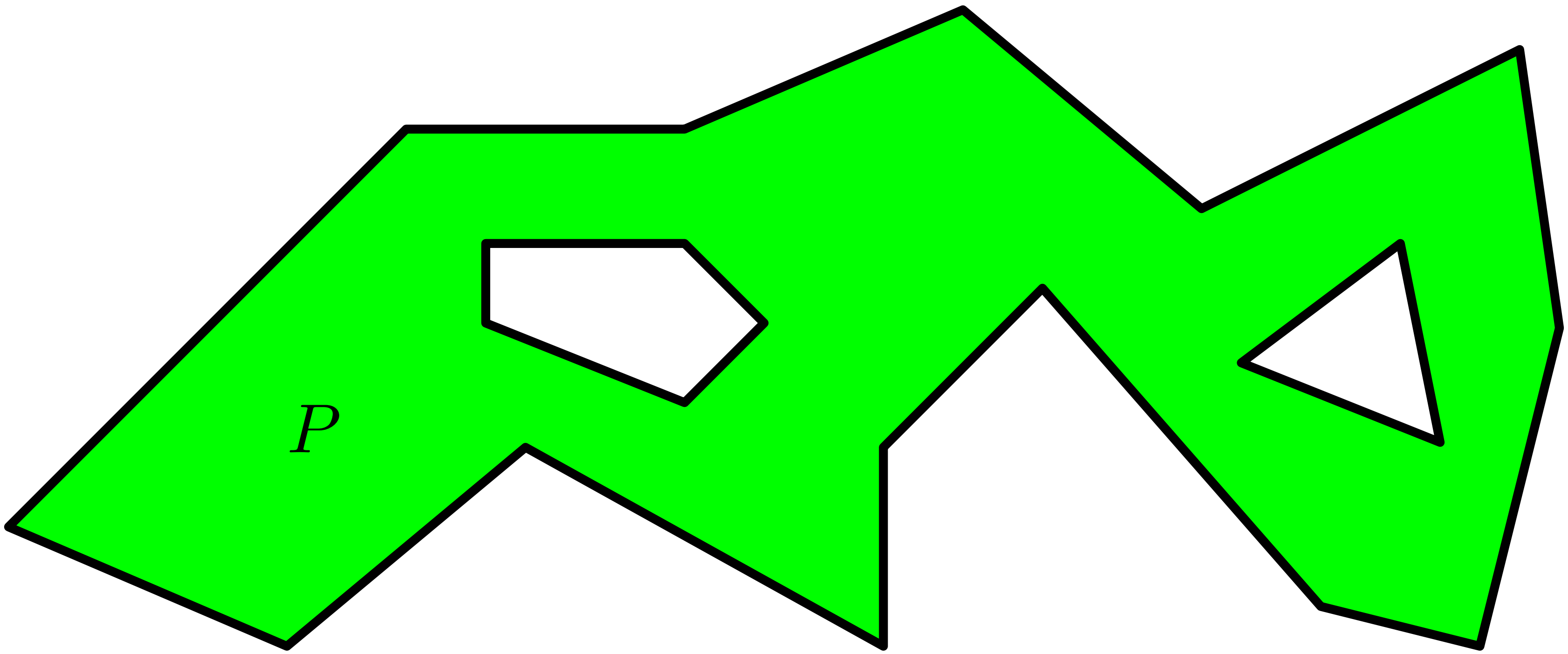
E se quisermos preencher o interior de um polígono P arbitrário?



Rasterização de polígonos

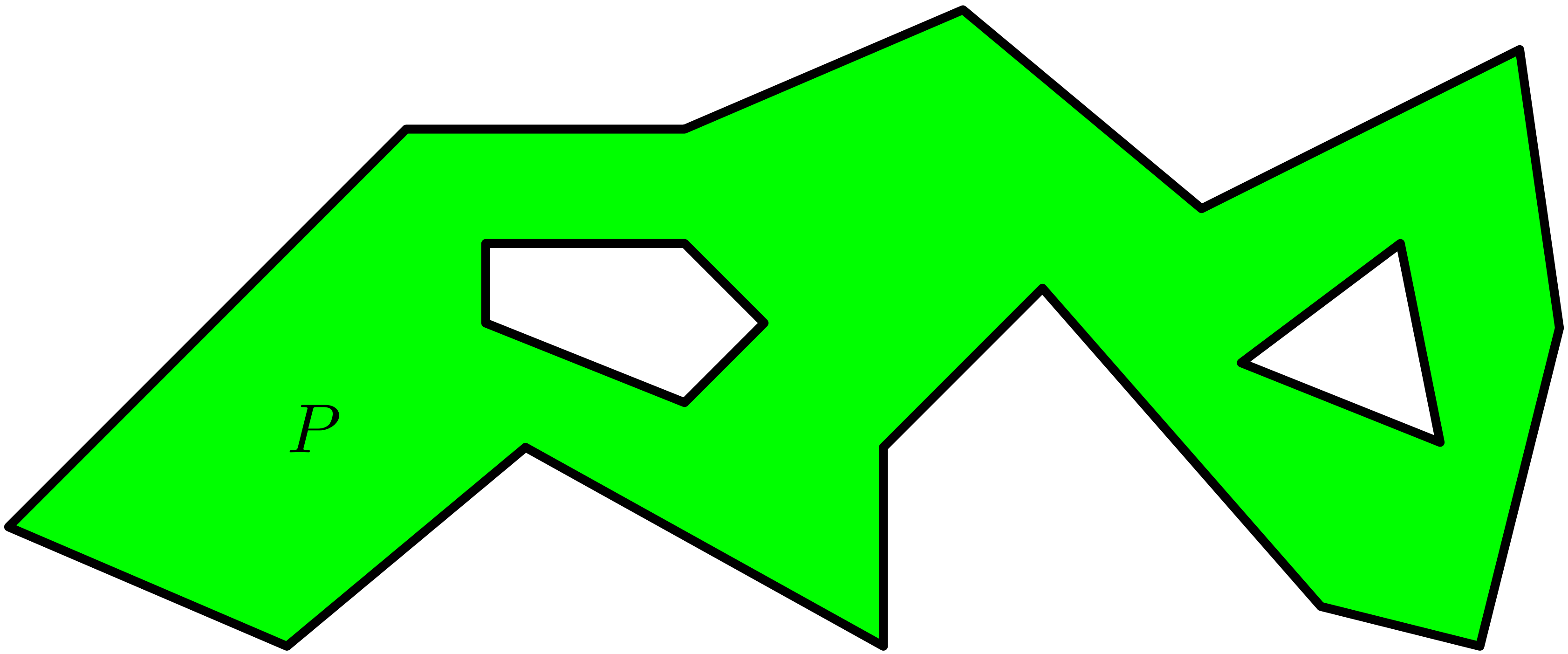
Estratégias clássicas para preencher P:

- Aplicar Bresenham nas arestas + `flood_fill`



Estratégias clássicas para preencher P:

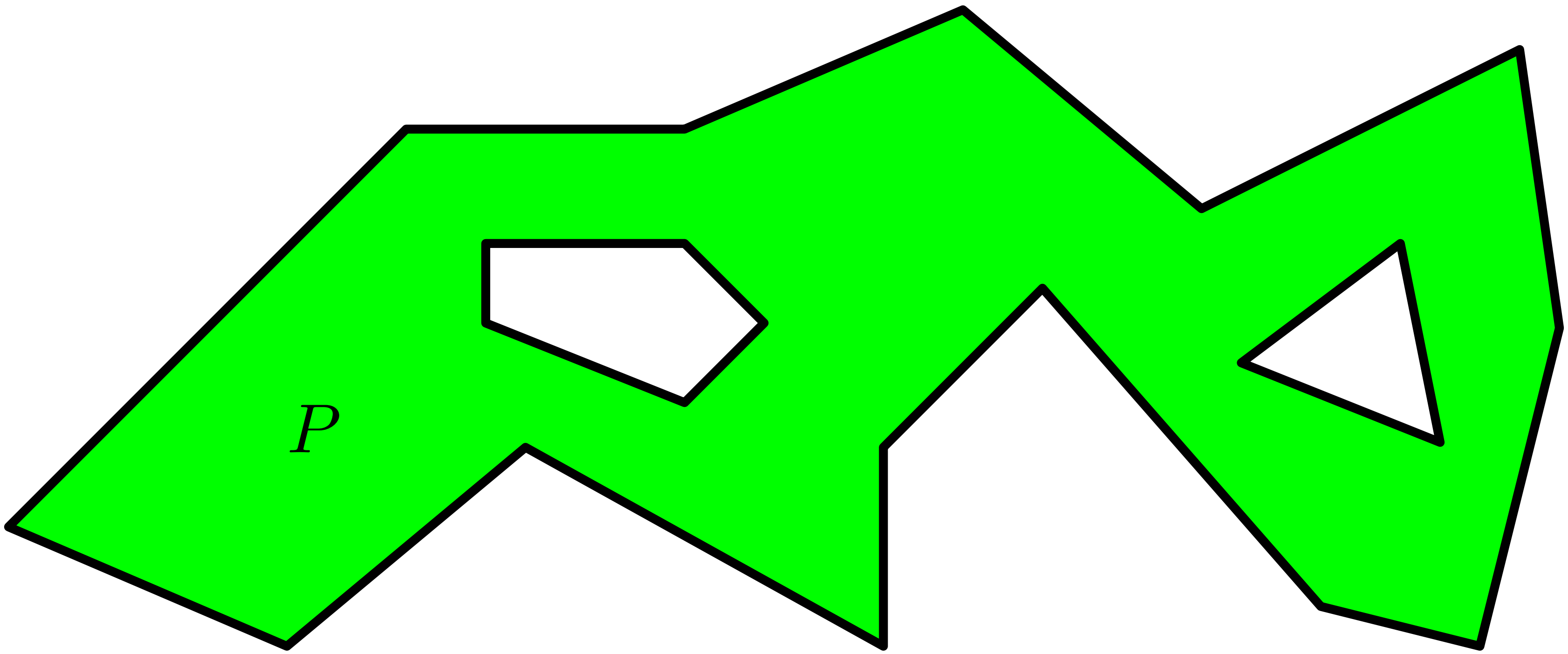
- Aplicar Bresenham nas arestas + `flood_fill`
- Algoritmo da linha de varredura



Rasterização de polígonos

Estratégias clássicas para preencher P:

- Aplicar Bresenham nas arestas + `flood_fill`
- Algoritmo da linha de varredura
- Triangular P, depois preencher cada triângulo



Faculdade Paraíso - CE

Ilustração do algoritmo da linha de varredura

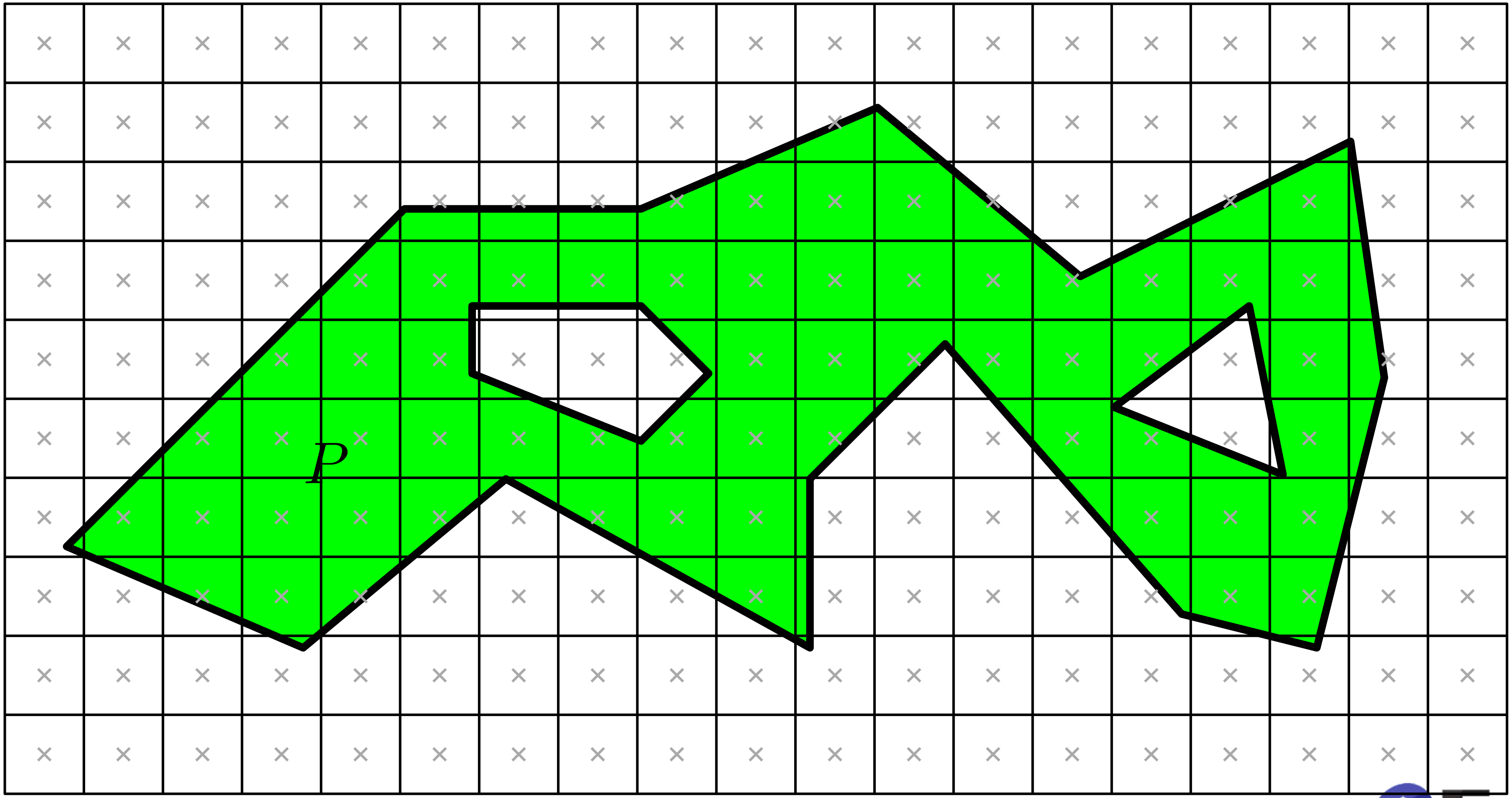
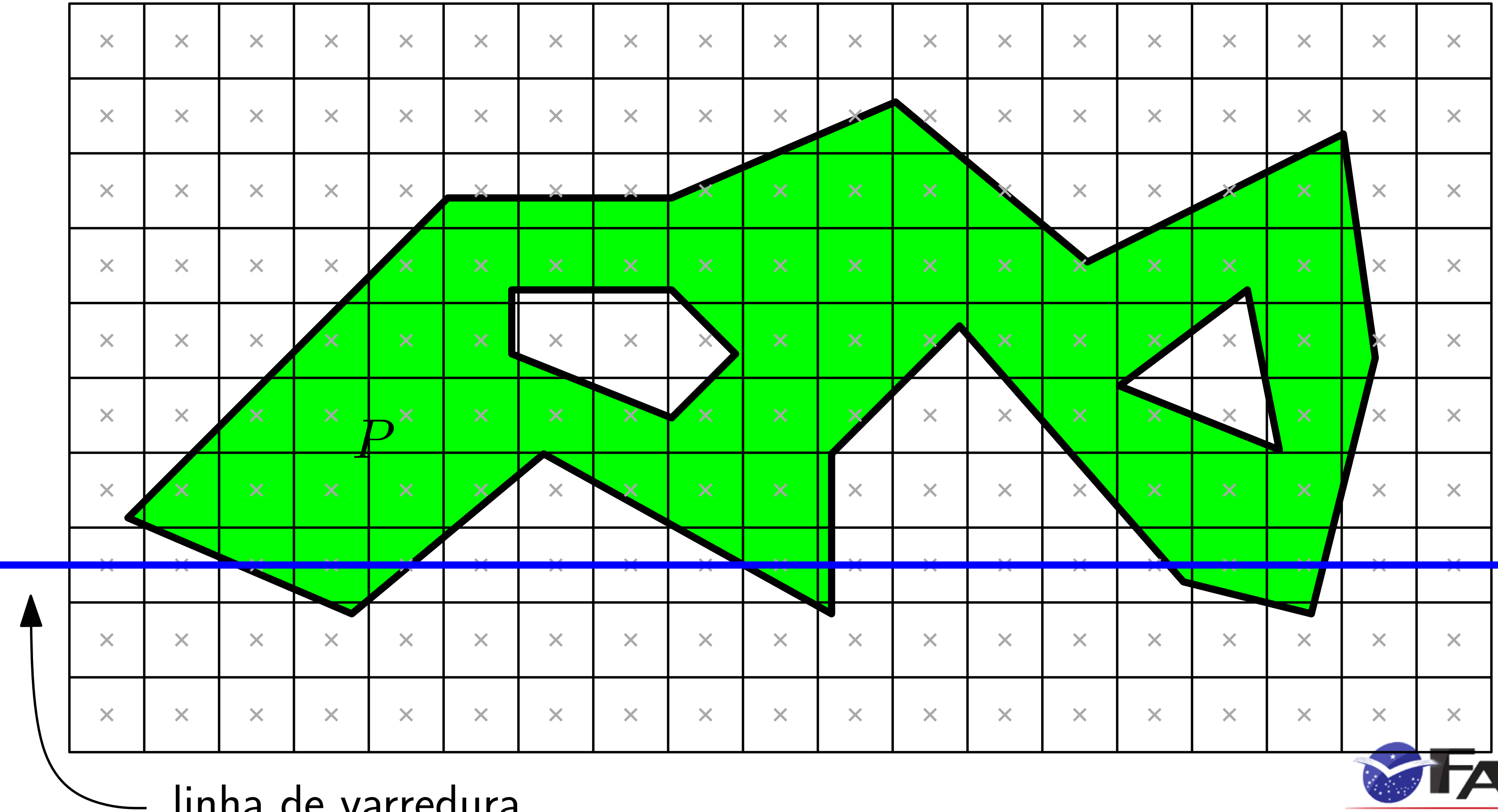


Ilustração do algoritmo da linha de varredura



linha de varredura

Ilustração do algoritmo da linha de varredura

1) Calcular os pontos de interseção entre L e P

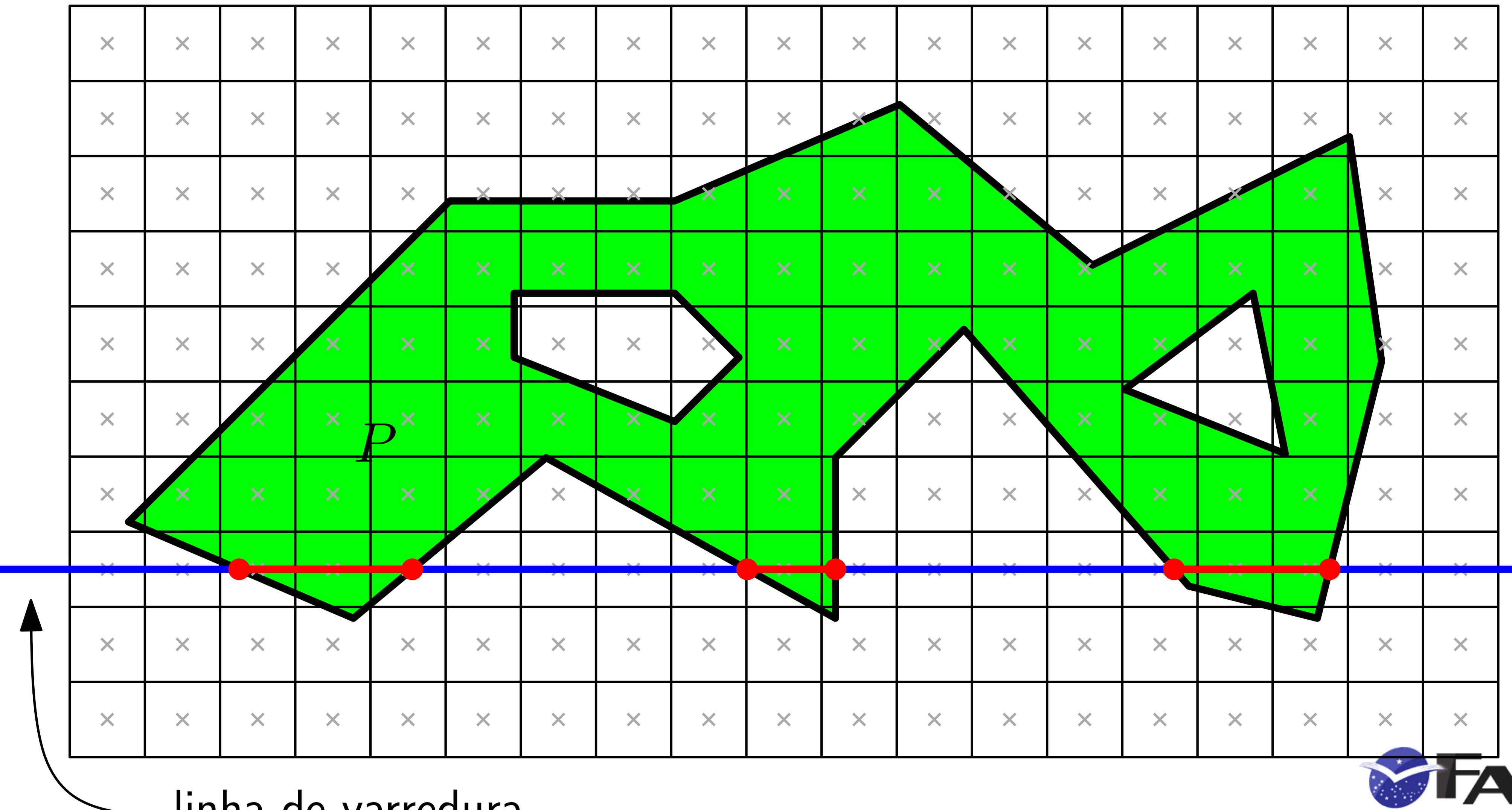
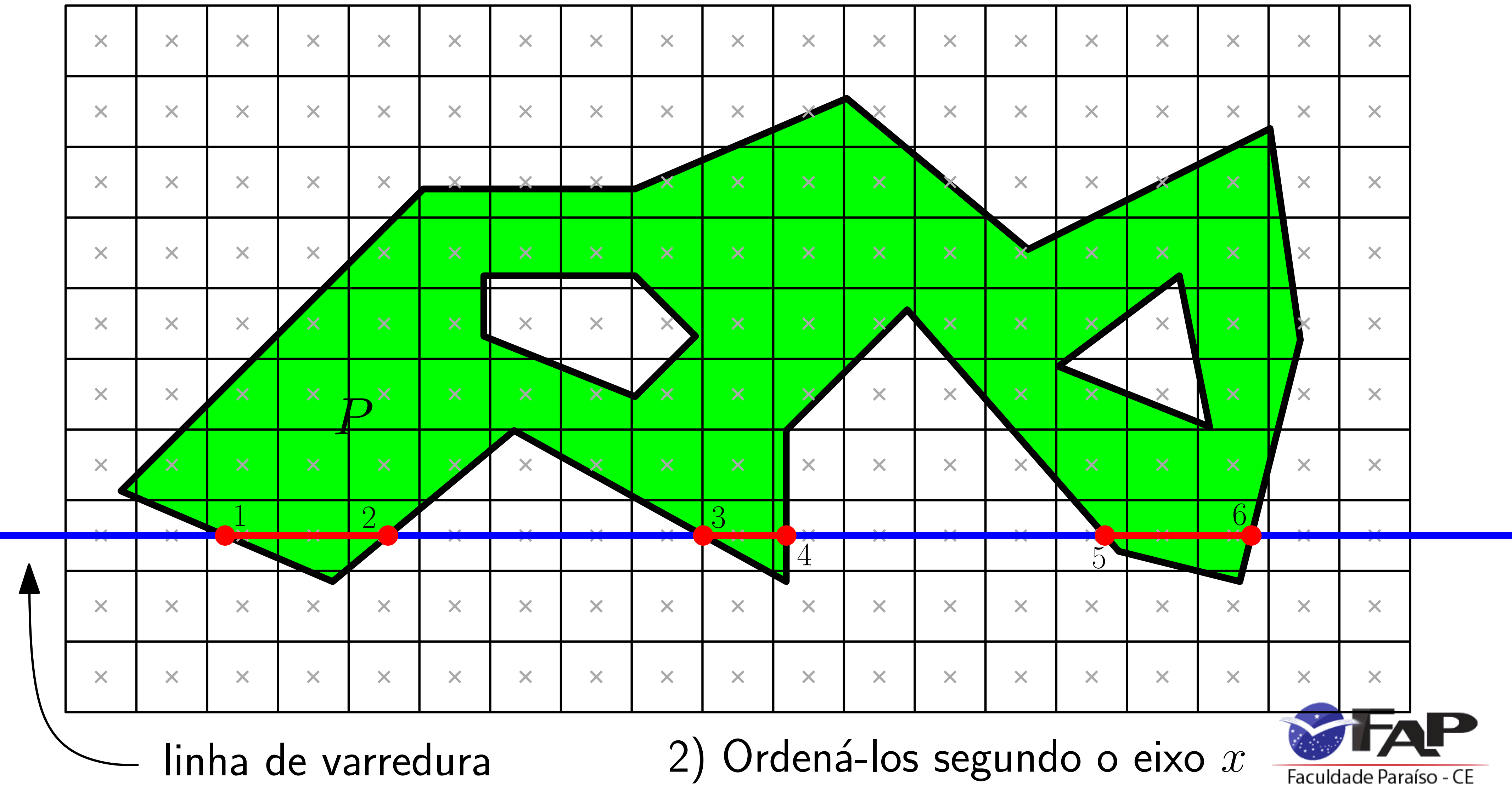


Ilustração do algoritmo da linha de varredura

1) Calcular os pontos de interseção entre L e P



2) Ordená-los segundo o eixo x

Ilustração do algoritmo da linha de varredura

3) Preencher os *pixels* entre pares consecutivos de interseções

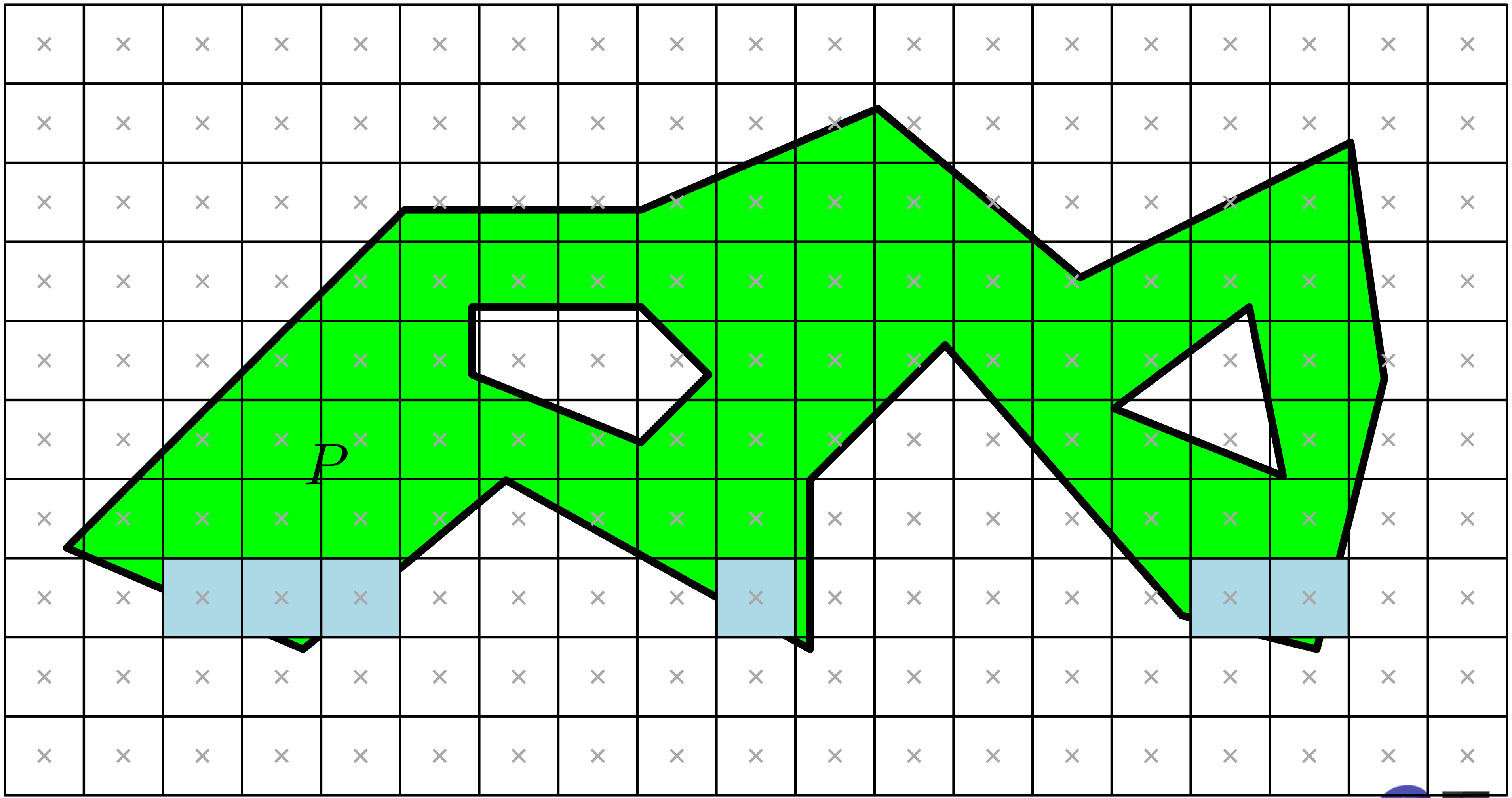


Ilustração do algoritmo da linha de varredura

3) Preencher os *pixels* entre pares consecutivos de interseções

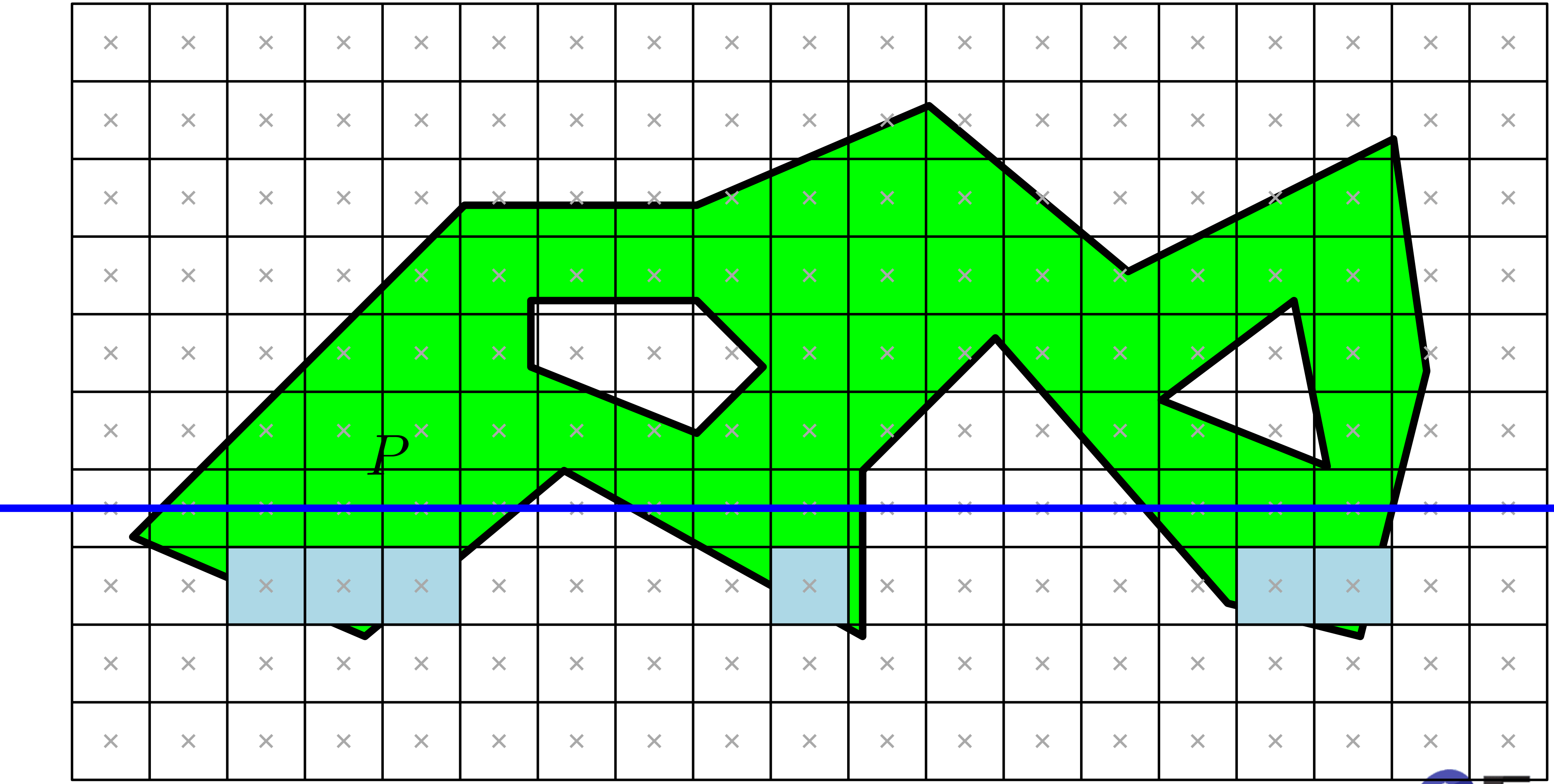


Ilustração do algoritmo da linha de varredura

3) Preencher os *pixels* entre pares consecutivos de interseções

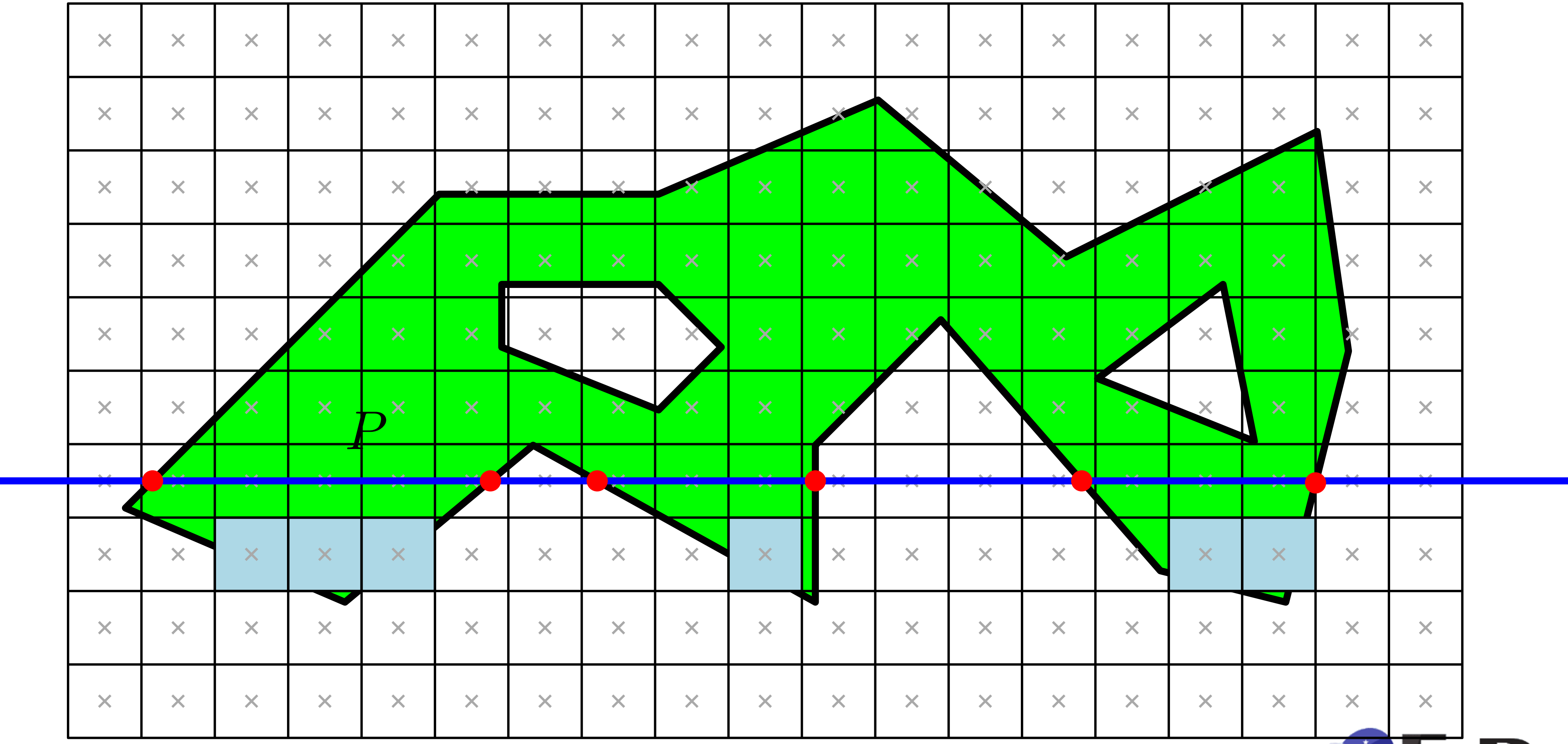


Ilustração do algoritmo da linha de varredura

3) Preencher os *pixels* entre pares consecutivos de interseções

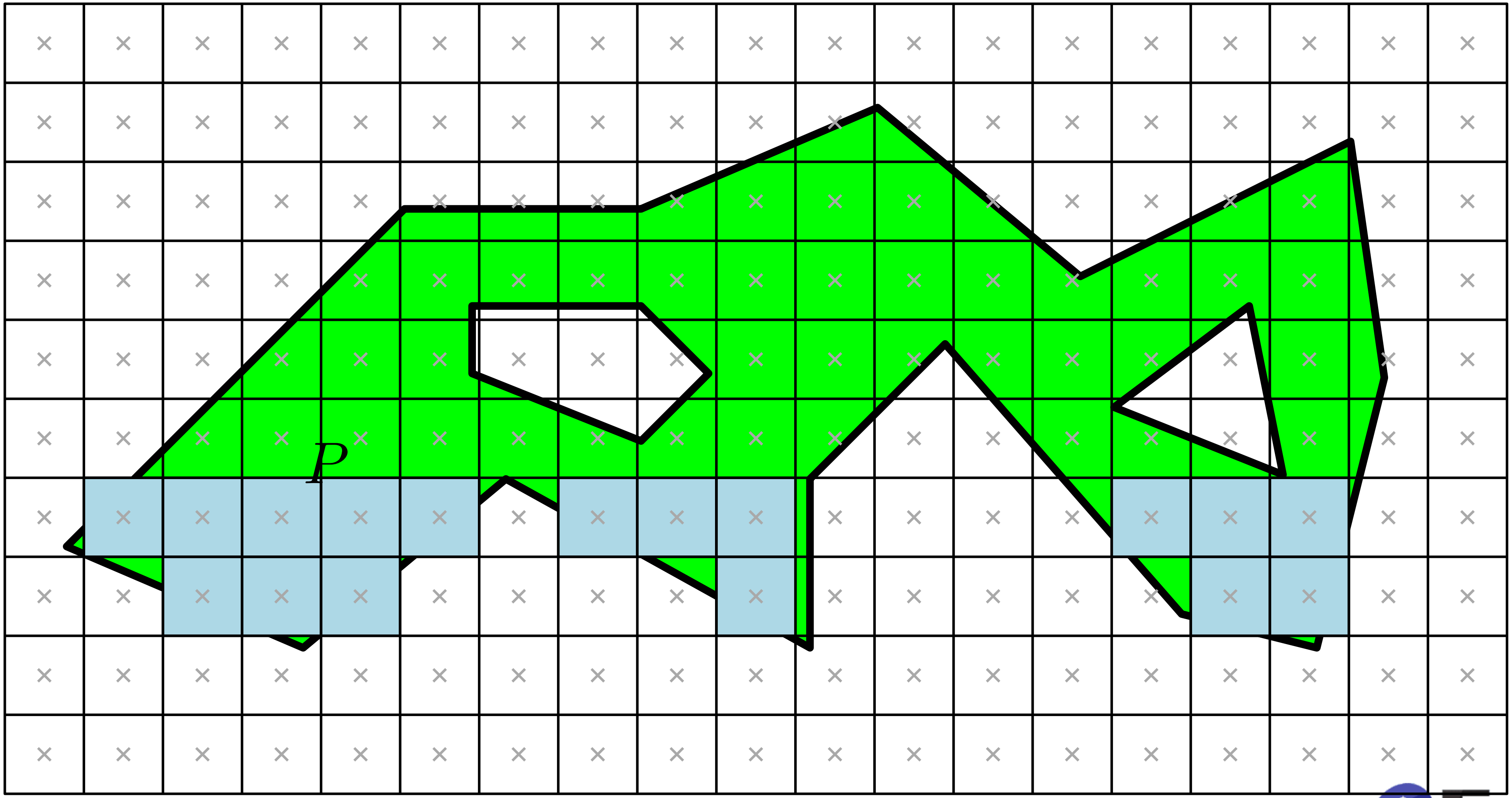
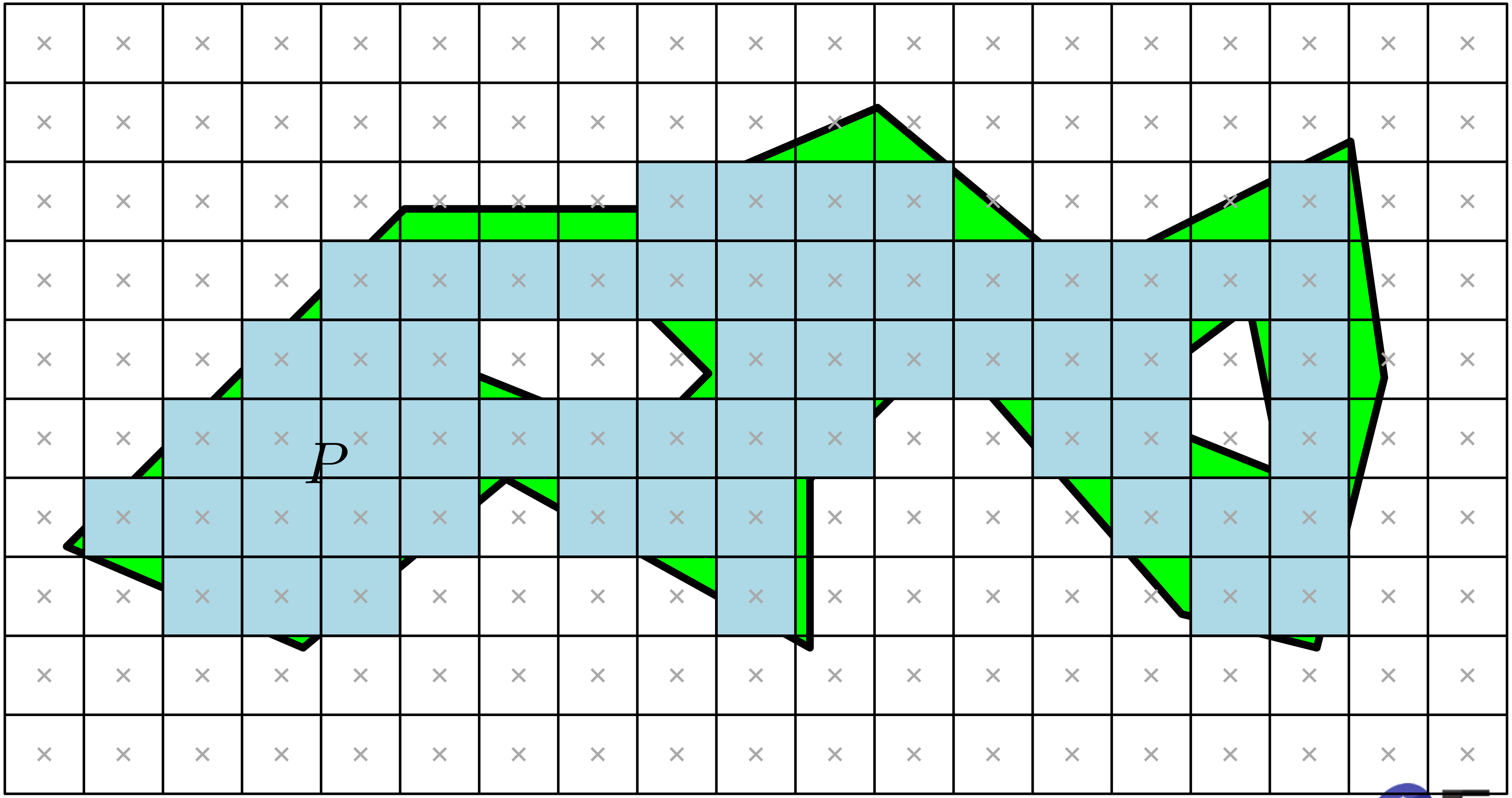


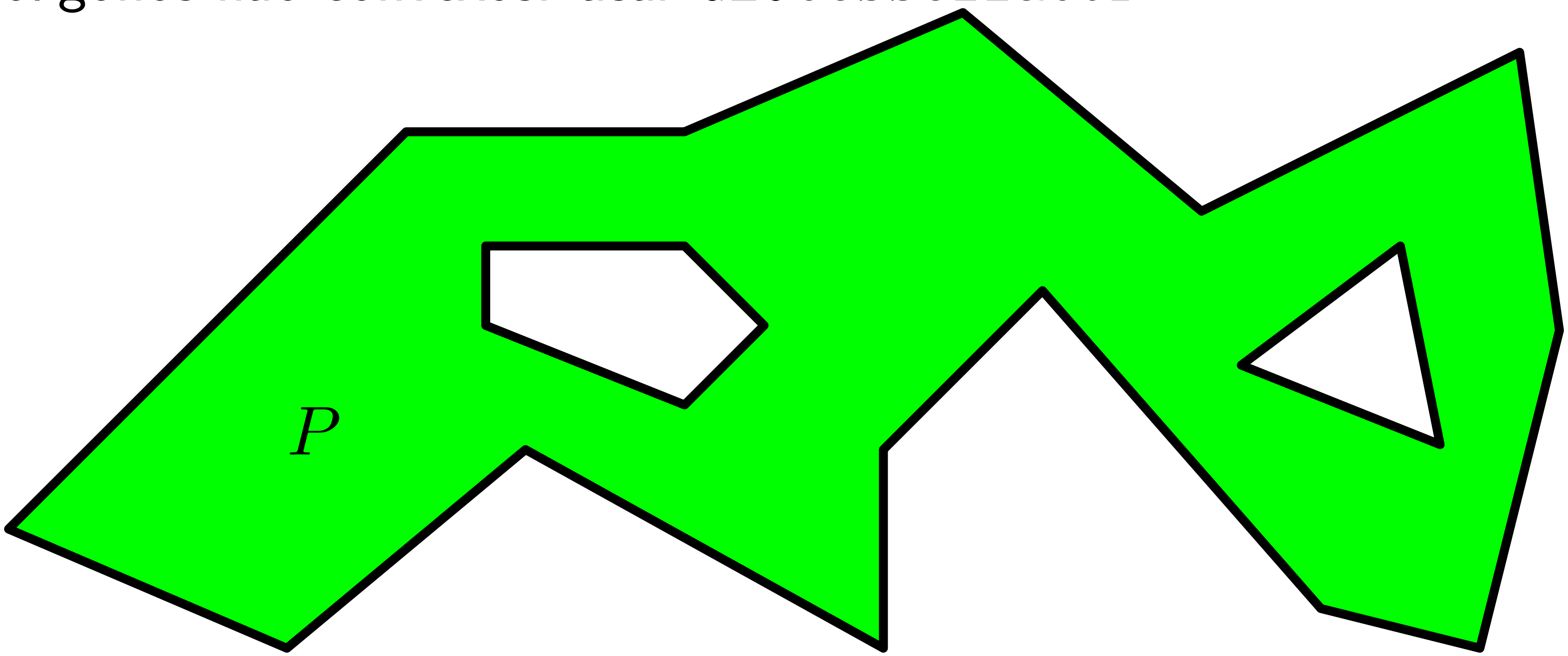
Ilustração do algoritmo da linha de varredura

3) Preencher os *pixels* entre pares consecutivos de interseções



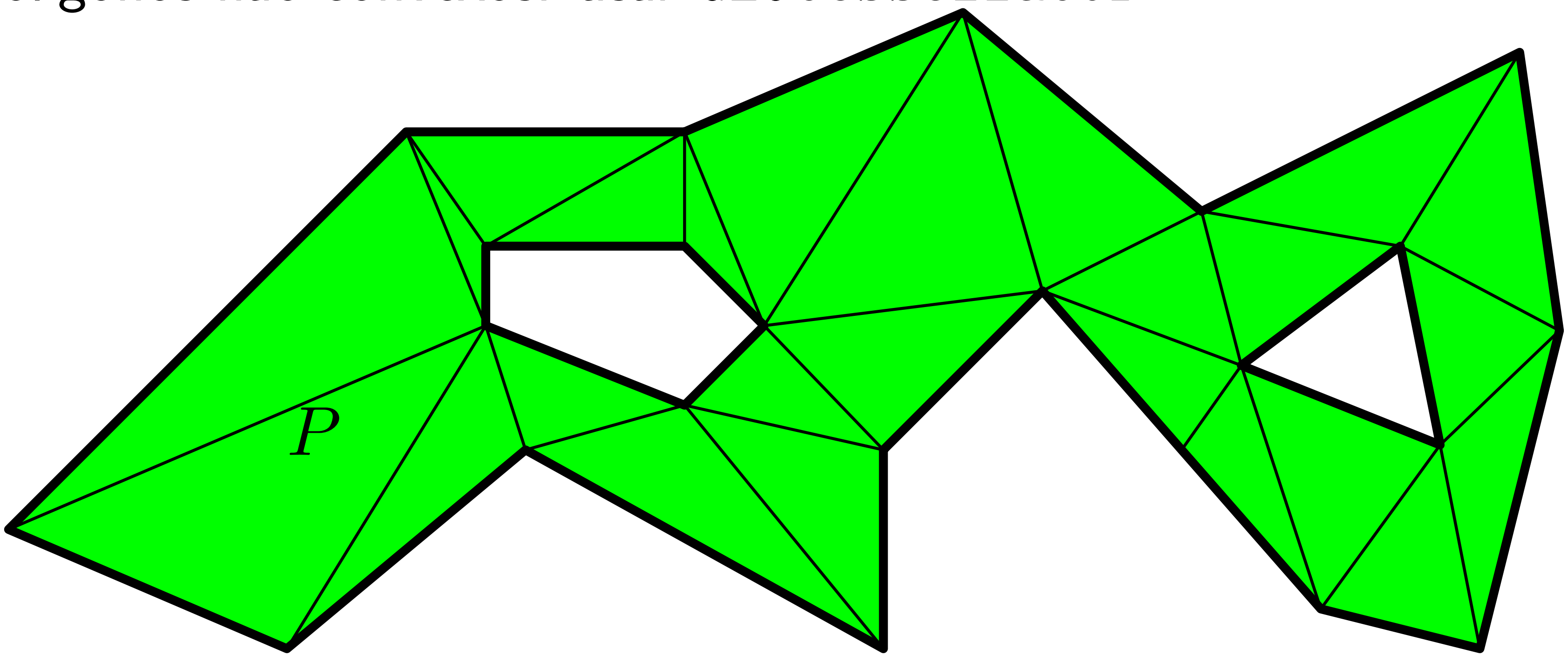
Algoritmo baseado em triangulação

- Estratégia adotada pela OpenGL: rasterizar apenas triângulos
- Polígonos convexos: triangulação trivial (orientação)
- Polígonos não-convexos: usar `GLUtesselator`



Algoritmo baseado em triangulação

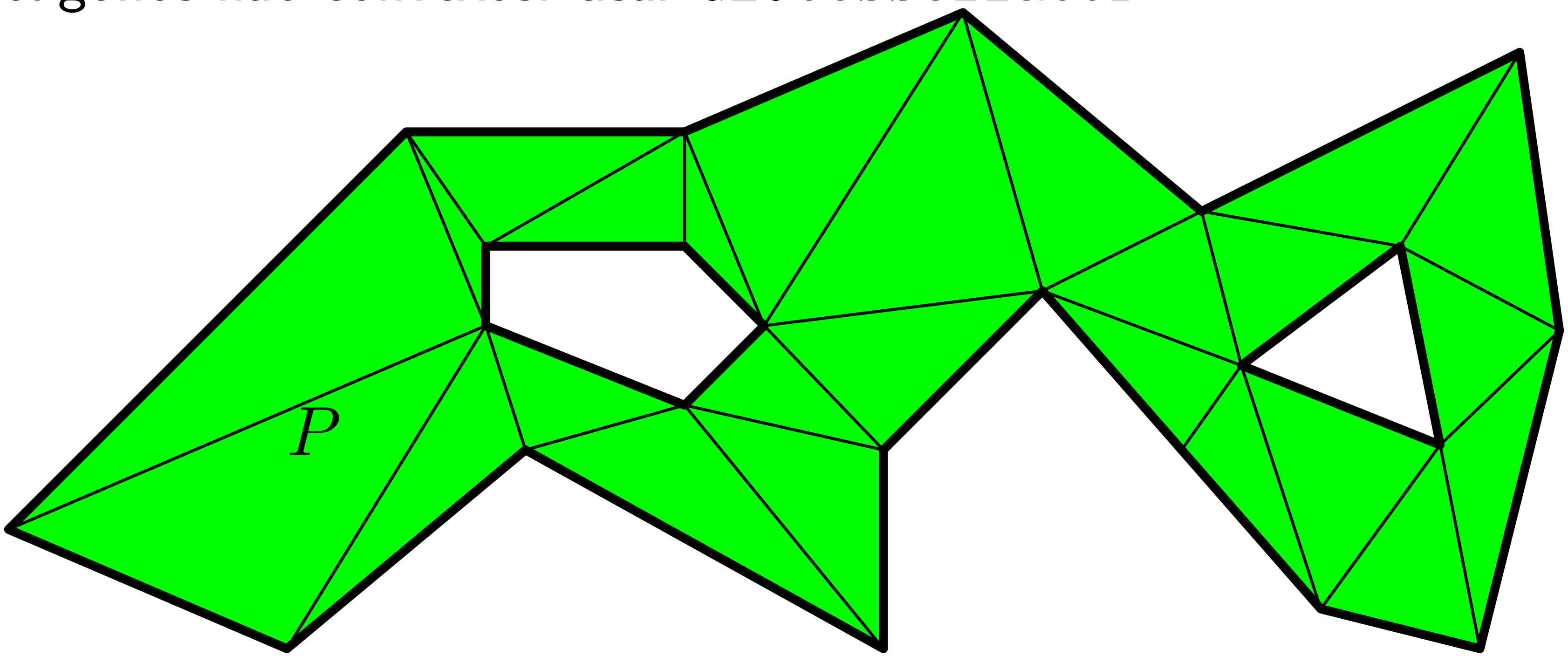
- Estratégia adotada pela OpenGL: rasterizar apenas triângulos
- Polígonos convexos: triangulação trivial (orientação)
- Polígonos não-convexos: usar `GLUtesselator`



Triangulação de P (sem pontos de Steiner)

Algoritmo baseado em triangulação

- Estratégia adotada pela OpenGL: rasterizar apenas triângulos
- Polígonos convexos: triangulação trivial (orientação)
- Polígonos não-convexos: usar `GLUtessellator`

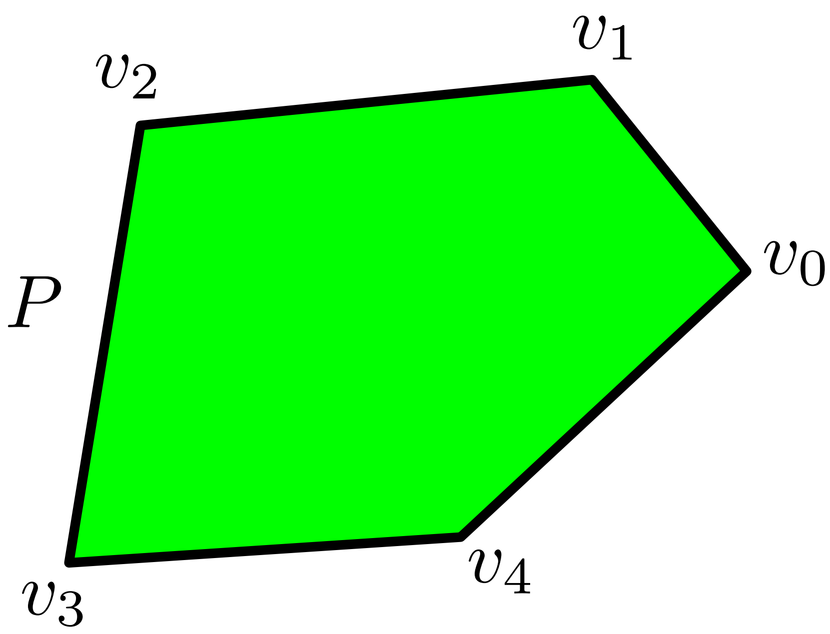


Triangulação de P (sem pontos de Steiner)

Então como rasterizar triângulos? A seguir...

Faculdade Paraíso - CE

Triângulo: primitiva básica para rasterização de polígonos da OpenGL

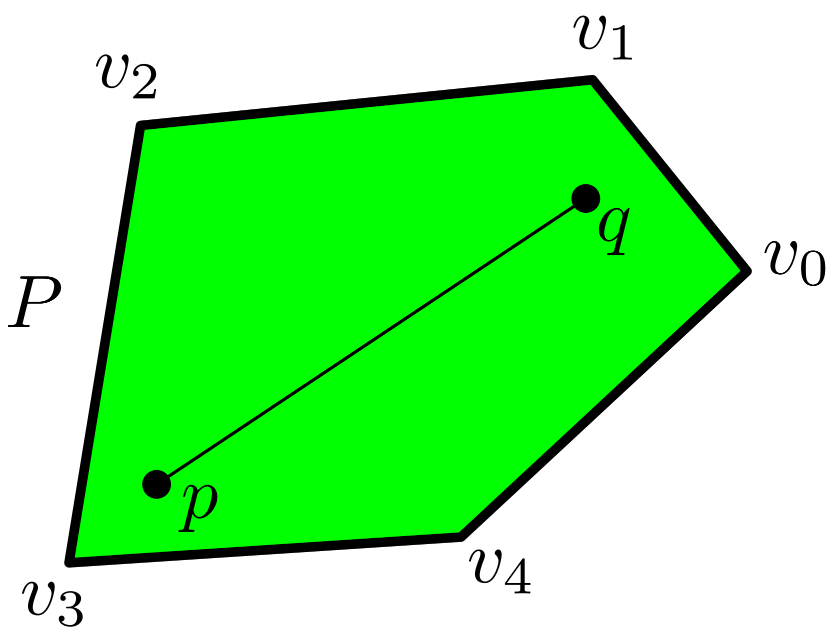


Triângulo: primitiva básica para rasterização de polígonos da OpenGL $\Longrightarrow$ Mas apenas polígonos *convexos*!

$$\lambda p + (1 - \lambda) q \in P$$

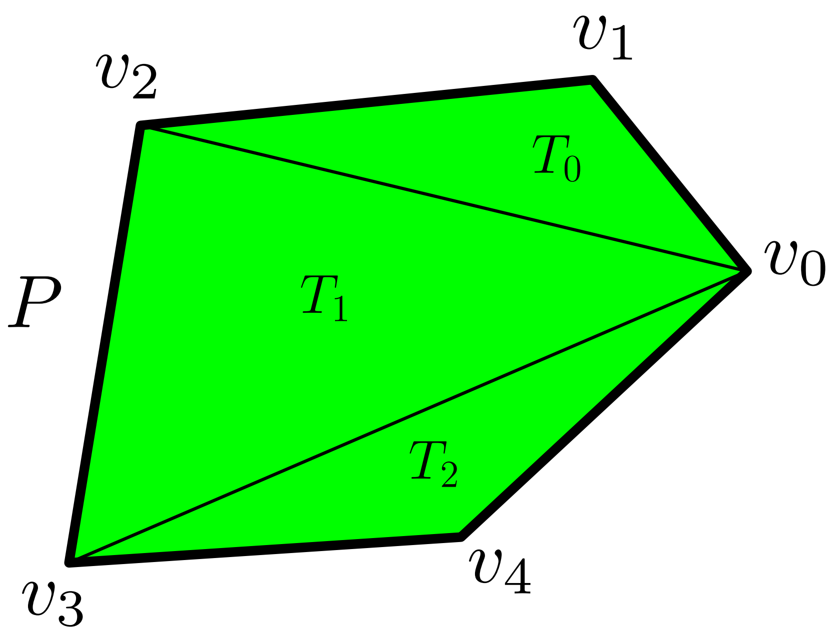
$$0 \leq \lambda \leq 1$$

Lembre-se: para polígonos não-convexos, usar `GLUTesselator`

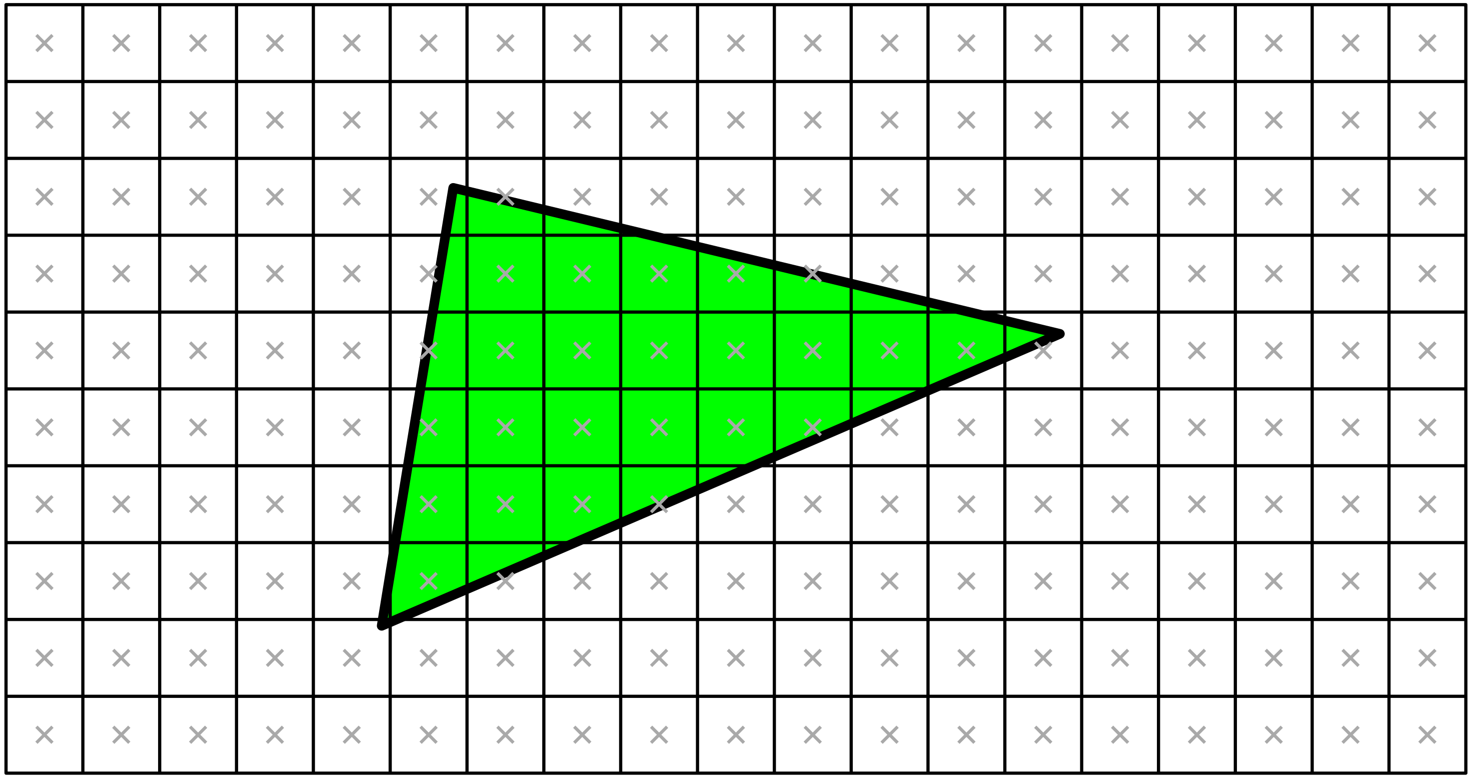


Triângulo: primitiva básica para rasterização de polígonos da OpenGL

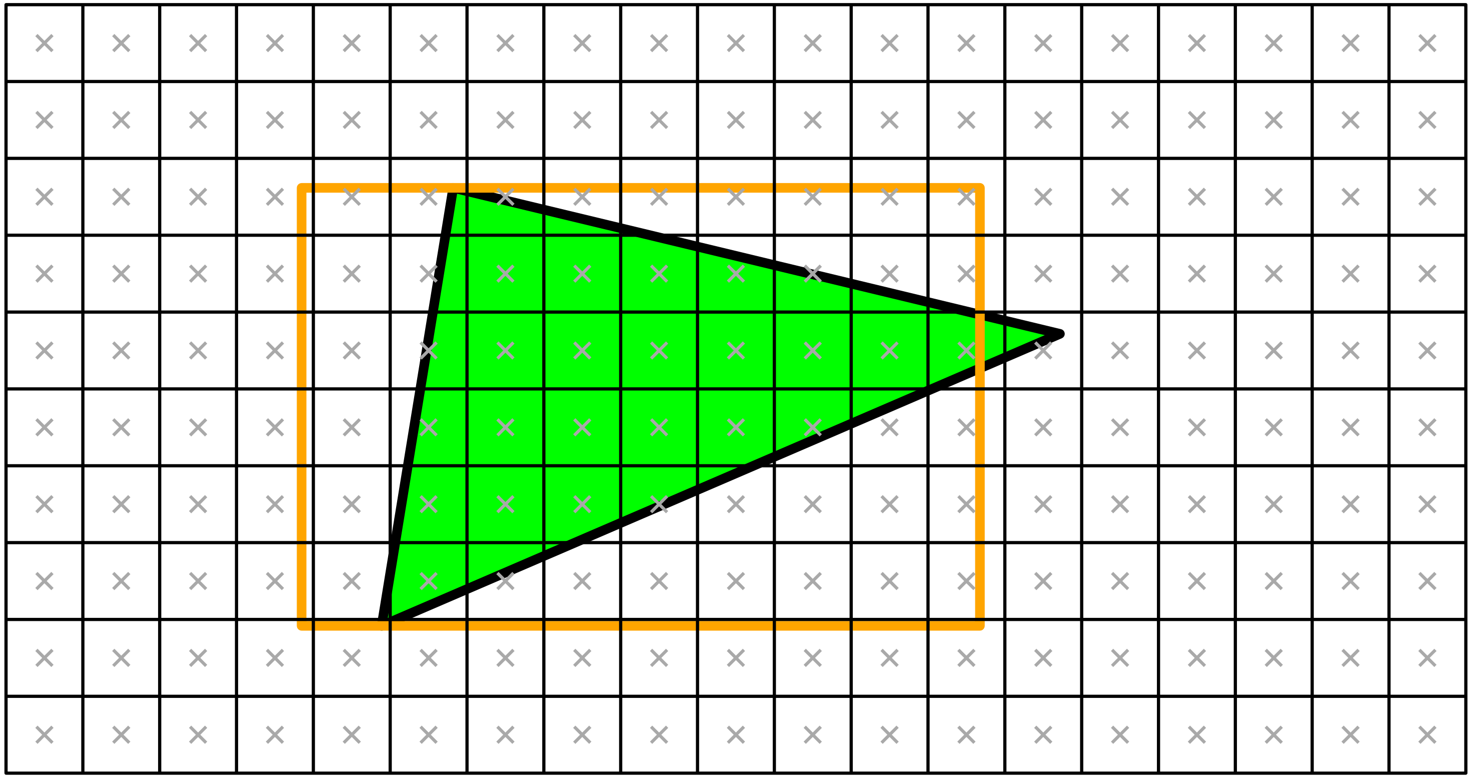
Triangulação de P



Rasterização de triângulos

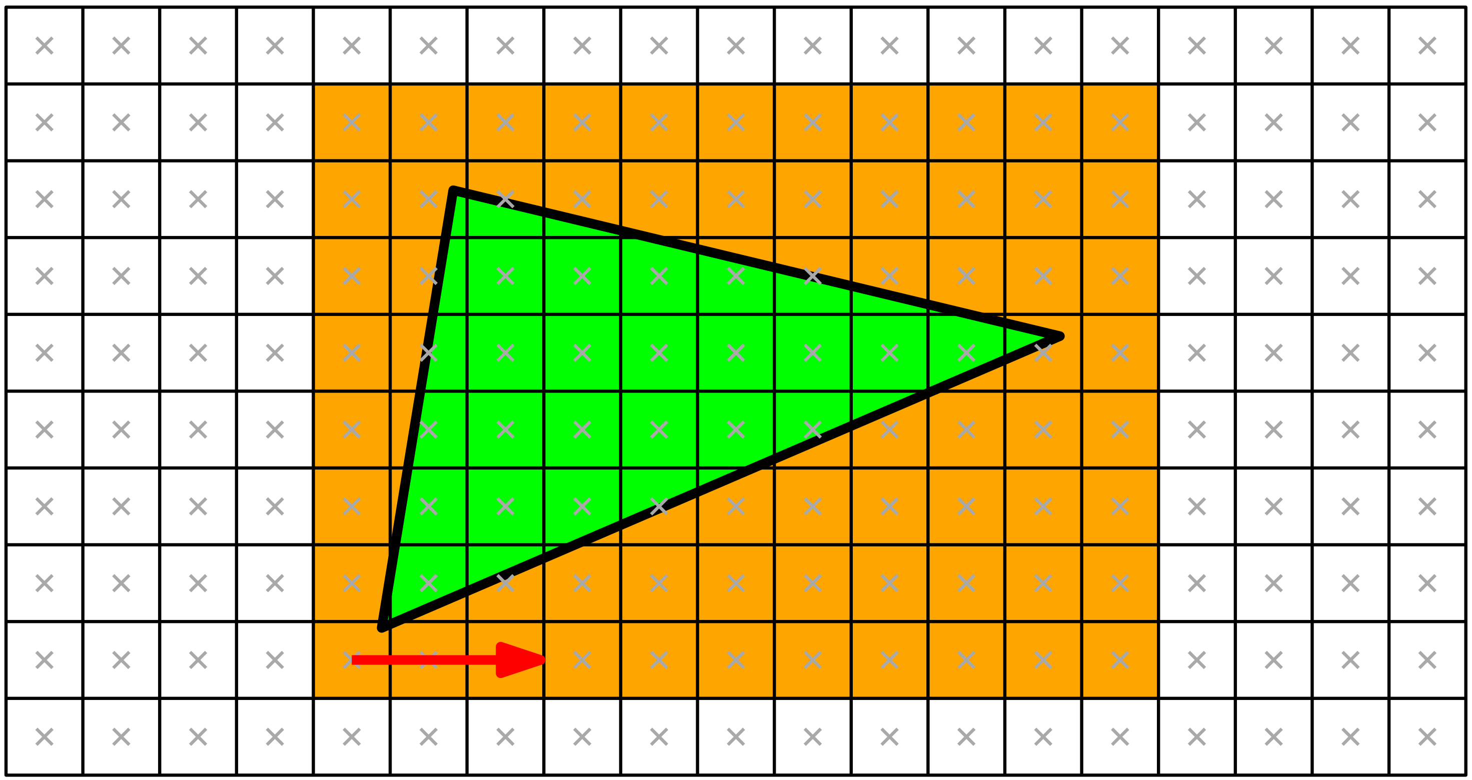


1) Determinar a caixa limitante alinhada com os eixos B de T_1

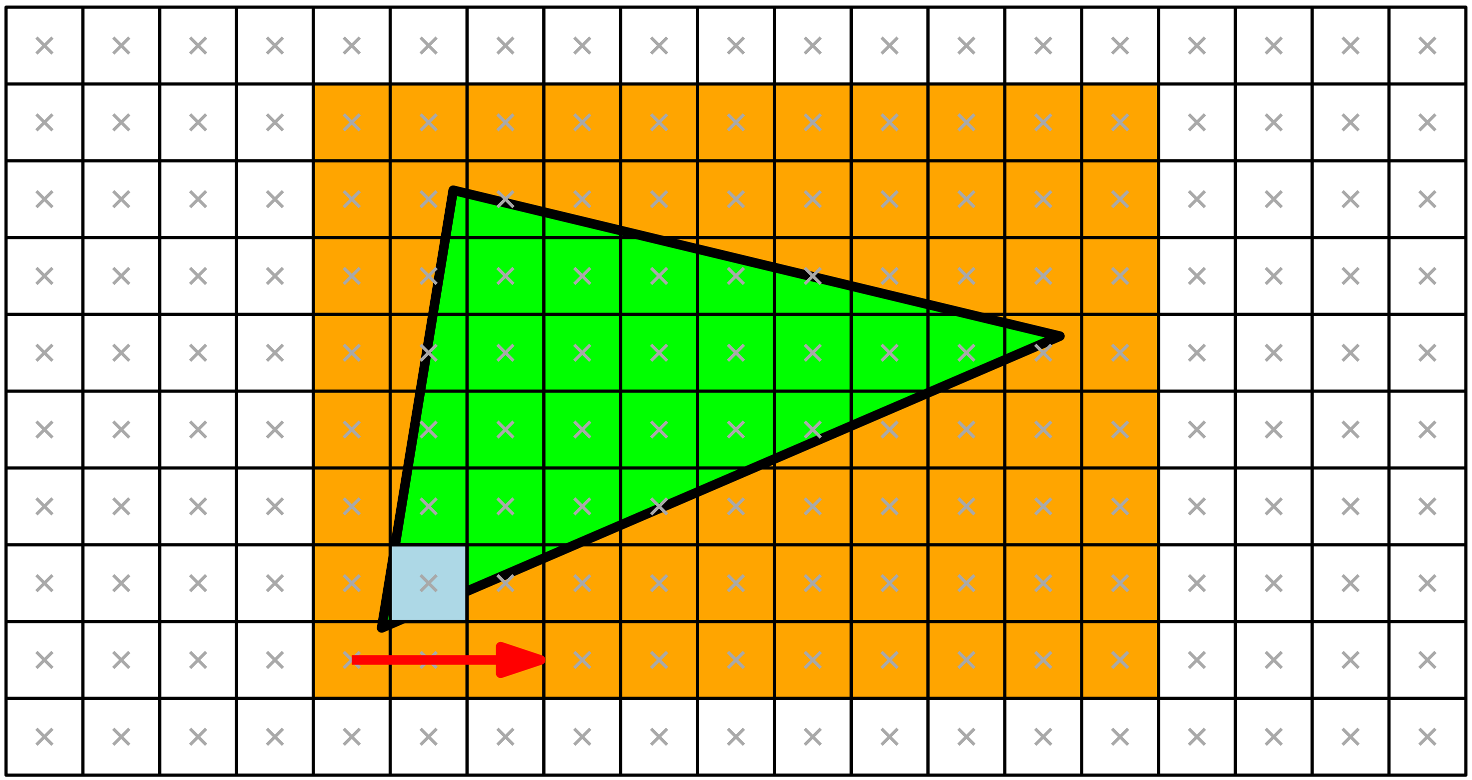


Rasterização de triângulos

1) Determinar a caixa limitante alinhada com os eixos B de T_1
2) Para cada *pixel* em B, faça:

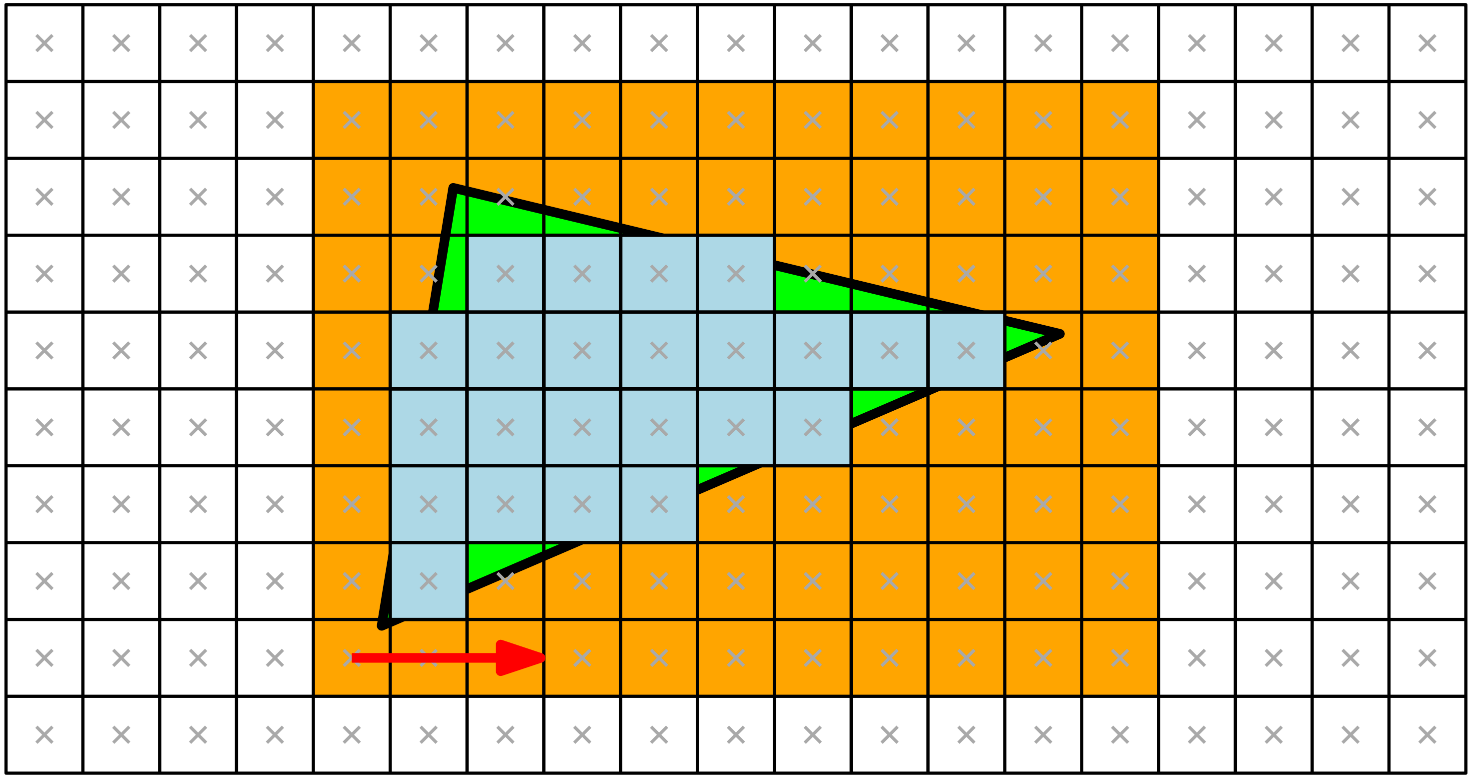


1) Determinar a caixa limitante alinhada com os eixos B de T_1
2) Para cada *pixel* em B, faça:
   Se *pixel* $\in T_1$, então pinte *pixel*



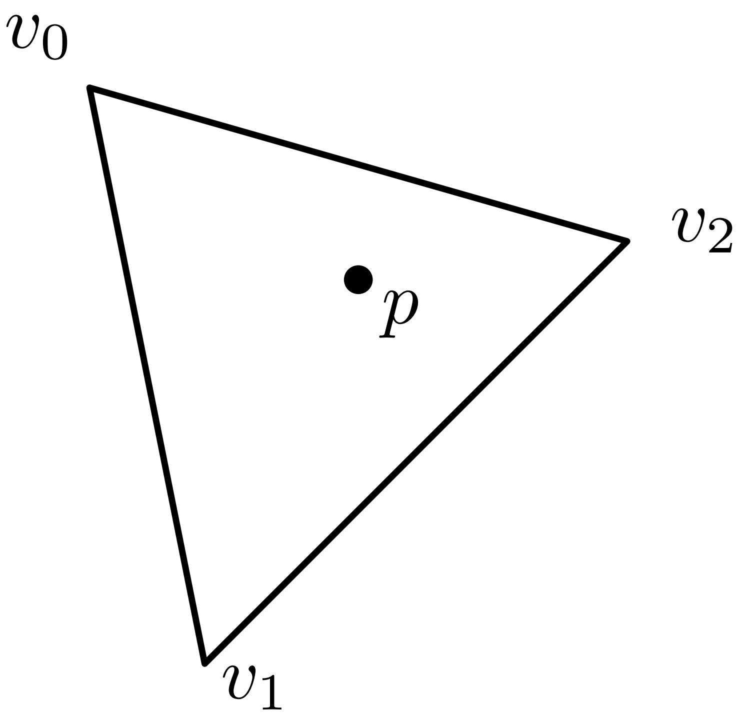
Rasterização de triângulos

1) Determinar a caixa limitante alinhada com os eixos B de T_1
2) Para cada *pixel* em B, faça:
   Se *pixel* $\in T_1$, então pinte *pixel*



Problema.

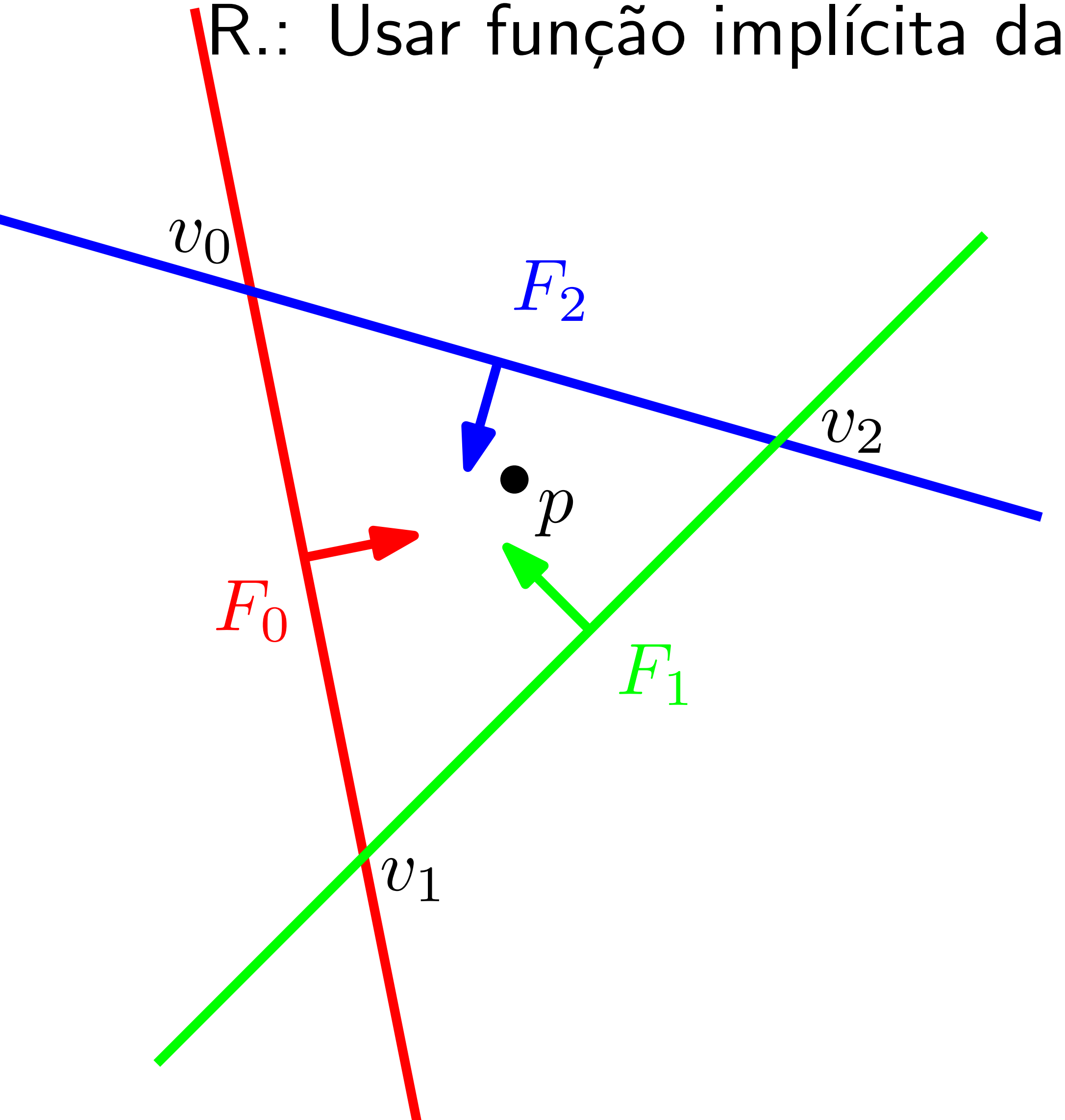
- Como determinar se um *pixel* está dentro do triângulo?



Problema.

- Como determinar se um *pixel* está dentro do triângulo?

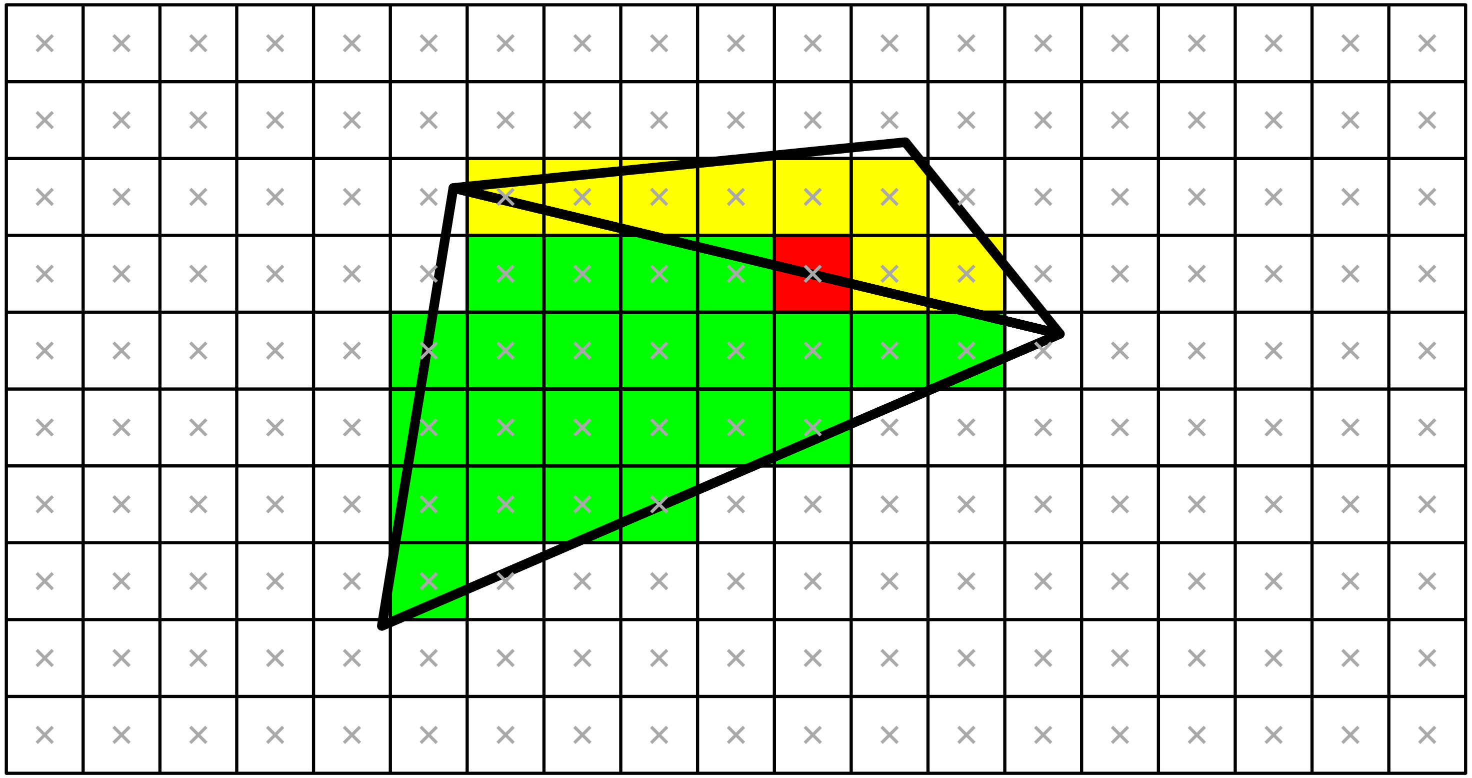
  R.: Usar função implícita da reta



Se $F_i(p) > 0$, $i = 0, 1, 2$, então p pertence ao triângulo

Problema.

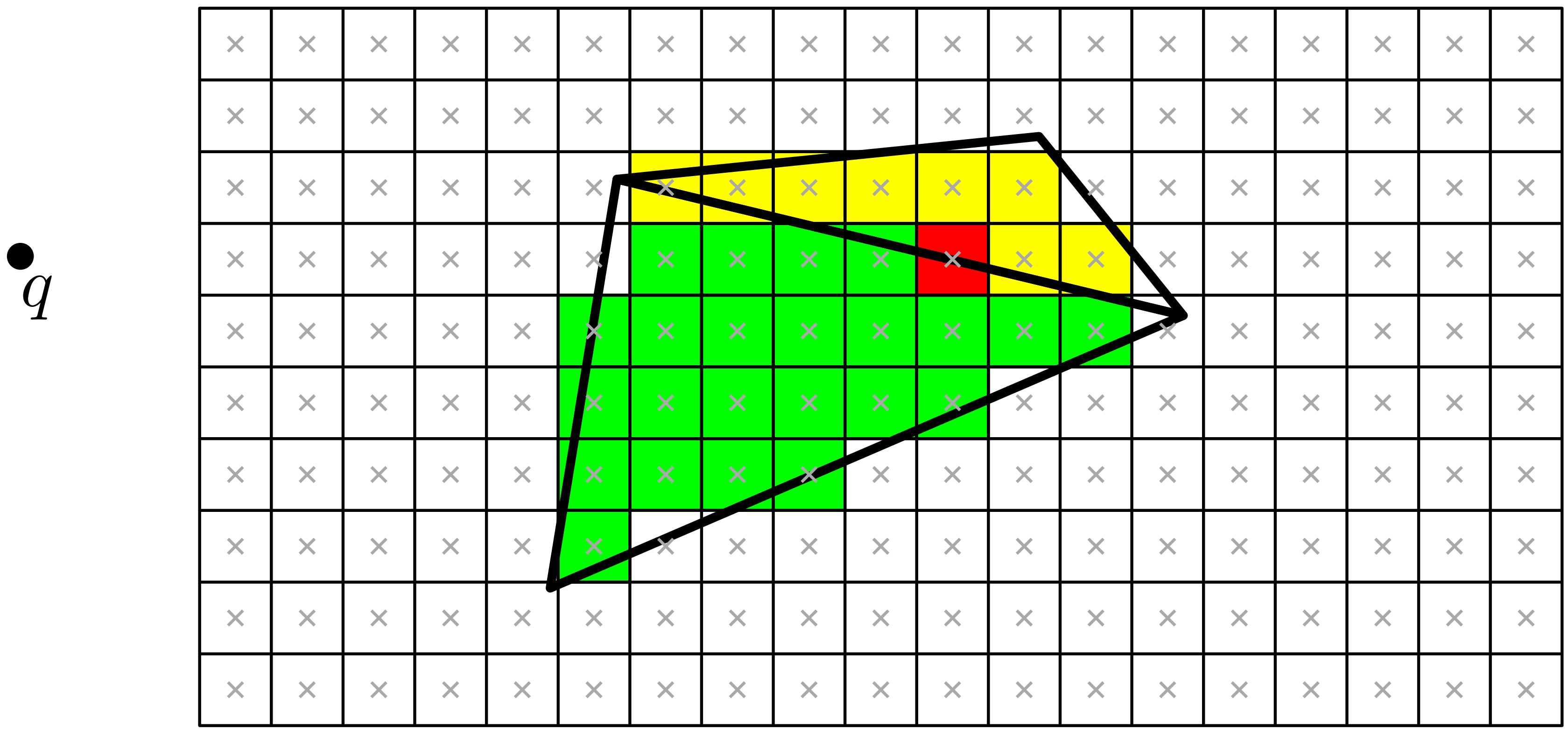
- O que fazer com um *pixel* na aresta do triângulo?



Problema.

- O que fazer com um *pixel* na aresta do triângulo?

  R.: Usar um ponto auxiliar



Problema.

- O que fazer com um *pixel* na aresta do triângulo?

  R.: Usar um ponto auxiliar