PRIMITIVAS EM GRÁFICOS RASTER

Computação Gráfica e Interfaces

FCT UNIVERSIDADE NOVA DE LISBOA

Sumário

Rasterização de primitivas

- Pontos e linhas
- Algoritmos DDA, de Bresenham e do Ponto Médio
- Activação de pixels

Preenchimento de áreas

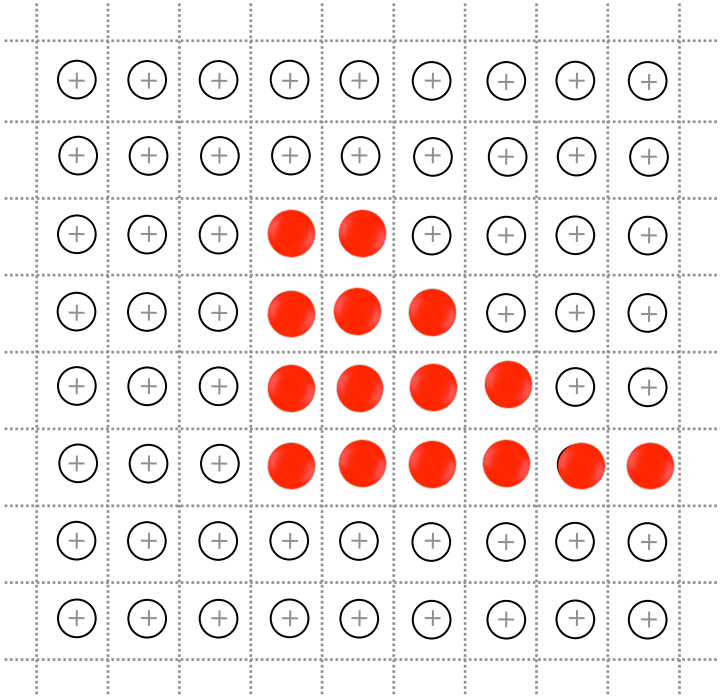
- Algoritmo Par-Ímpar para polígonos
- Preenchimento de área de pixels

Atributos gráficos

- Grossura e estilos de linhas
- Antialiasing

Tecnologia raster

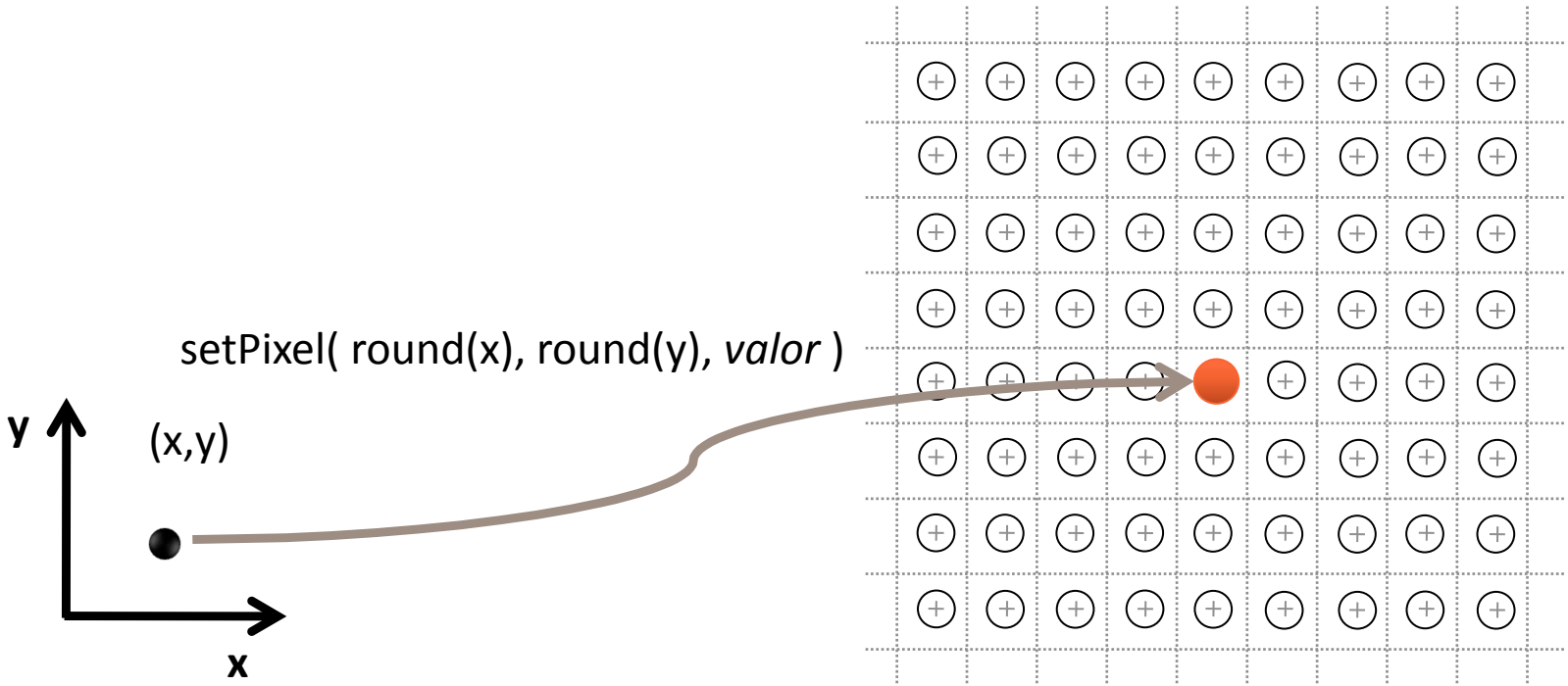
Considera-se uma imagem como um conjunto de pixels dispostos numa grelha e manipuláveis individualmente



Obs.: Nestes apontamentos, vamos utilizar o círculo para representar o pixel

Pontos

É necessário converter valores reais definidos num domínio contínuo, para valores inteiros no domínio discreto do dispositivo de saída

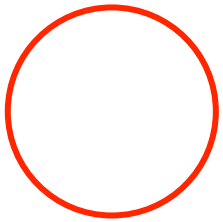


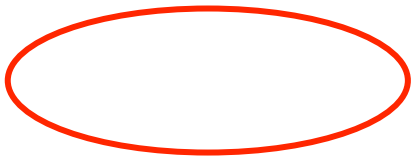
Linhas

Pretende-se que a conversão de linhas por varrimento em tecnologia raster permita obter resultados de qualidade e de uma forma rápida

Tipos de linhas

- Segmentos de recta
- Circunferências
- Etc.





Características desejáveis para a aproximação discreta da linha

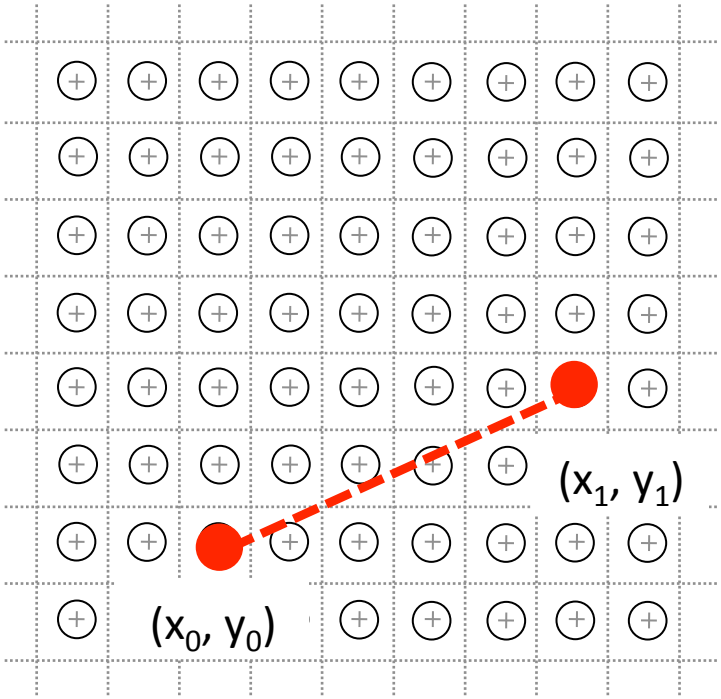
- Precisão, marcando os pixels mais próximos da linha real
- Aparência contínua
- Espessura e brilho constantes

Segmentos de recta

Em valores reais

$$y = m\,x + b\,, \quad \text{com} \quad m = \frac{y_1 - y_0}{x_1 - x_0} = \frac{\Delta y}{\Delta x}, \quad b = y_0 - m\,x_0\,,$$

- Definidos pelas coordenadas dos pontos extremos P_0, P_1
- A aproximação discreta será exacta no caso de segmentos de recta verticais, horizontais, ou diagonais de 45 graus relativamente a um dos eixos coordenados



Solução imediata :

```
for x from x0 to x1 step 1 do
        y ← m * x + b
        setPixel( x , round(y), colour )
endfor
```

Algoritmos incrementais

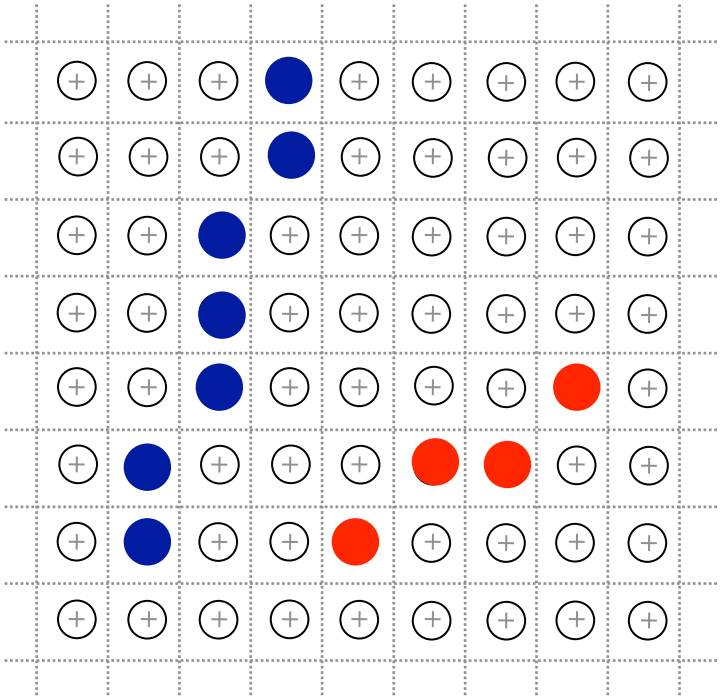
Os algoritmos incrementais são melhores – em cada iteração, uma das variáveis varia de uma unidade

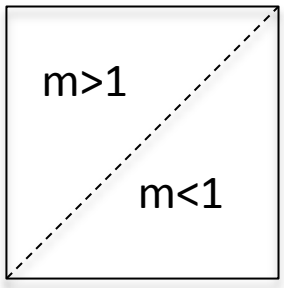
A escolha da variável independente é relevante, pois pretende-se obter uma linha com pixels adjacentes

No caso do 1º quadrante:

$$y_{i+1} = y_i + 1, \quad x_{i+1} = x_i + 1/m$$

$$x_{i+1} = x_i + 1, \quad y_{i+1} = y_i + m$$





Para os outros quadrantes, a análise é similar, podendo ser facilmente deduzida por simetria

Algoritmo DDA (**D**igital **D**ifferential **A**nalyser)

Exemplo para segmentos de recta entre (x_0, y_0) e (x_1, y_1), com $x_0 \leq x_1$, $0 < m < 1$

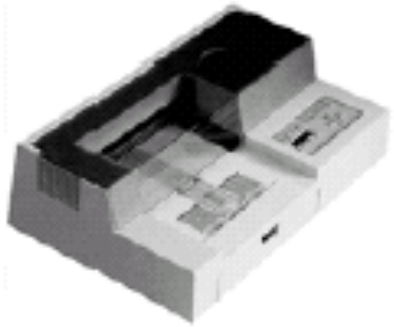
```
Δx ← x1 - x0 ,    Δy ← y1 - y0
m ← Δy  /  Δx
y ← y0
for x from x0 to x1 step 1 do
        setPixel( x , round(y), colour )
        y ← y + m
endfor
```

Note-se que:

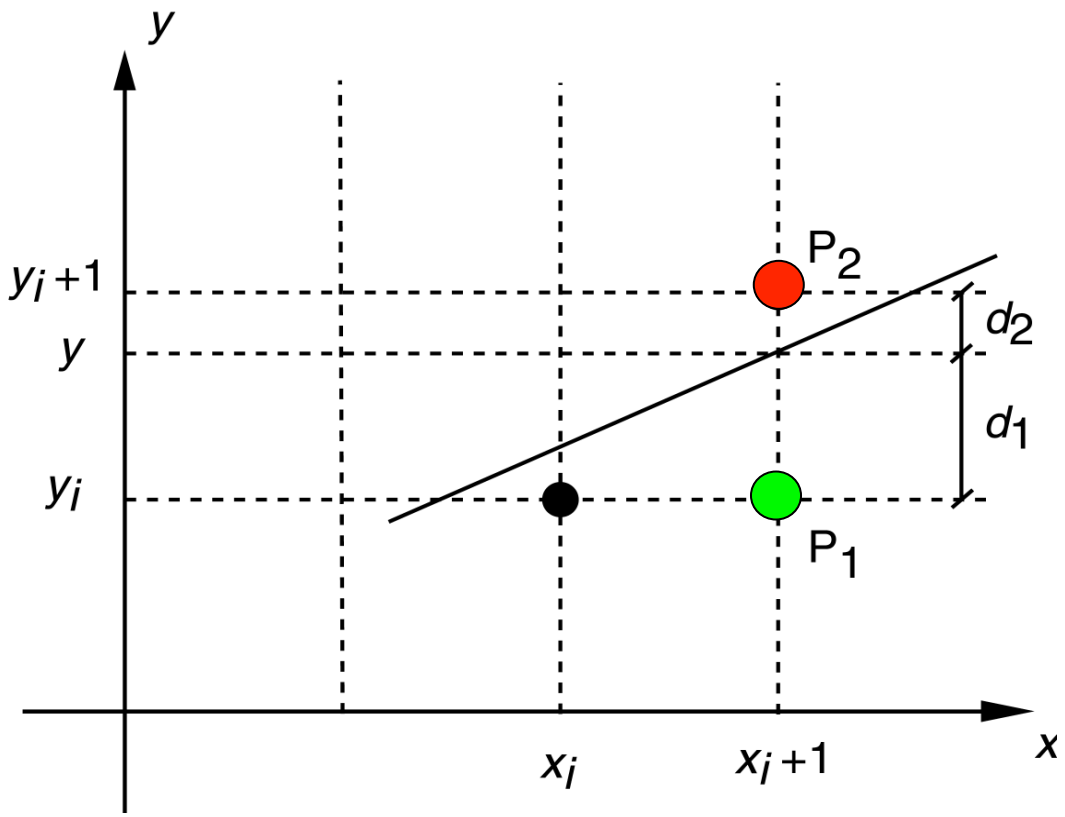
a) não são necessárias multiplicações
b) não existe acumulação de erros

Algoritmo de Bresenham

Um algoritmo clássico (1965), inicialmente concebido para utilização em traçador de canetas, apenas com aritmética de números inteiros



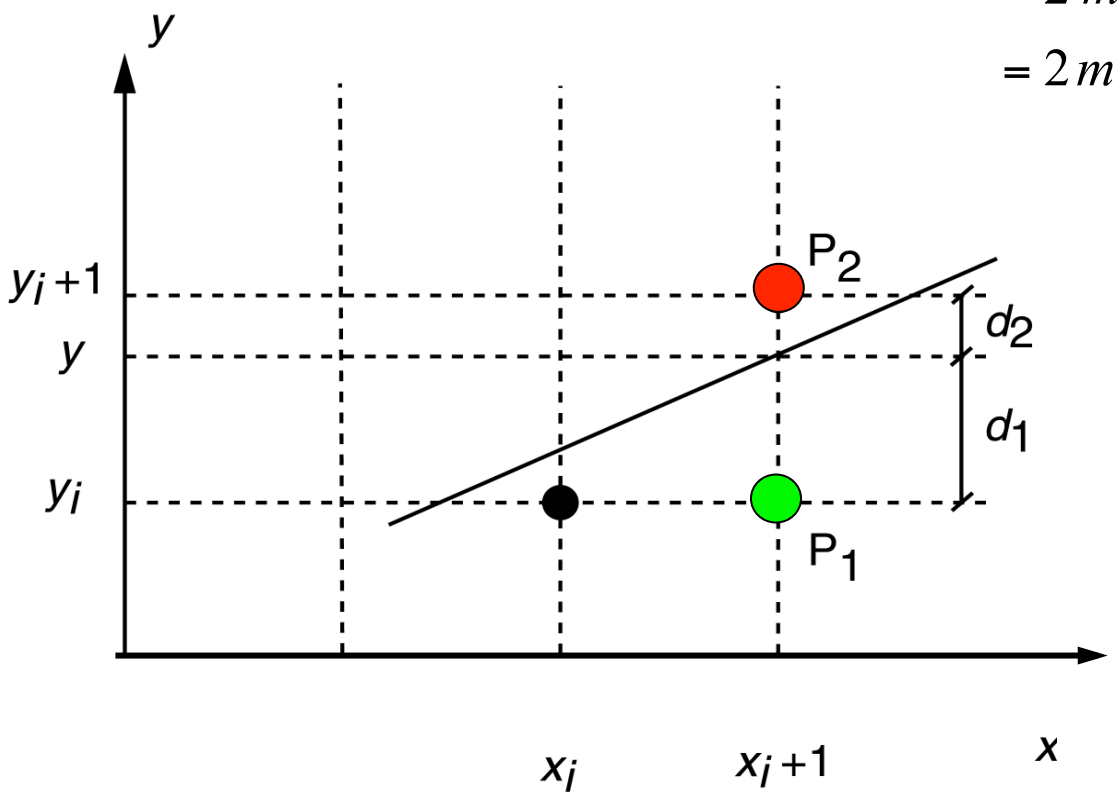
Num dado passo *i*, qual é o pixel seguinte ? P_1 ou P_2 ?



No caso do 1º octante e considerando $b = 0$

decisão baseada no valor $d_1 - d_2$

$$\begin{aligned} d_1 - d_2 &= (y - y_i) - (y_{i+1} - y) \\ &= 2y - 2y_i - 1 \\ &= 2\,m\,(x_i + 1) - 2y_i - 1 \\ &= 2\,m\,x_i + 2m - 2y_i - 1 \end{aligned}$$



Actualização da variável de decisão *p*

$$\begin{aligned} p_i &= \Delta x\,(d_1 - d_2) \\ &= \Delta x\,(2m\,x_i + 2m - 2y_i - 1) \\ &= 2\Delta y\,x_i - 2\Delta x\,y_i + k\,, \quad \text{com} \quad k = \Delta x\,(2m-1) \end{aligned}$$

logo $p_{i+1} = 2\Delta y\,x_{i+1} - 2\Delta x\,y_{i+1} + k$

$$\begin{aligned} p_{i+1} - p_i &= 2\Delta y\,x_{i+1} - 2\Delta x\,y_{i+1} - 2\Delta y\,x_i + 2\Delta x\,y_i \\ &= 2\Delta y\,(x_{i+1} - x_i) + 2\Delta x\,(y_{i+1} - y_i) \end{aligned}$$

Concluindo:

$$x_{i+1} = x_i + 1 \quad (\text{independentemente de ser } \mathrm{P_1} \text{ ou } \mathrm{P_2})\,,$$

$$p_{i+1} = \begin{cases} p_i + 2\Delta y\,, & \text{se} \quad y_{i+1} = y_i \quad (\mathrm{P_1})\,, \\ p_i + 2(\Delta y - \Delta x)\,, & \text{se} \quad y_{i+1} = y_i + 1 \quad (\mathrm{P_2}) \end{cases}$$

$$p_0 = 2\Delta y - \Delta x$$

Obs.: Dedução matemática fora do âmbito do estudo em CGI

Exemplo para segmentos de recta entre (x_0, y_0) e (x_1, y_1), com $x_0 \leq x_1$, $0 < m < 1$ (os outros casos podem ser deduzidos de forma análoga, ou até convertidos para este)

```
Δx ← x1 - x0 ,    Δy ← y1 - y0
p ← 2 Δy  -  Δx
p1  ← 2 Δy
p2  ← 2 ( Δy  -  Δx )
x ← x0 ,  y ← y0
for x from x0 to x1 step 1 do
        setPixel( x , y, colour )
        if  p < 0 then
                d ← p + p1
        else
                p ← p + p2
                y ← y + 1
        endif
endfor
```

Programa de demonstração

Algoritmo do Ponto Médio

O algoritmo do ponto médio (Pitteaway 1965, Van Aken 1985) é facilmente generalizável para o traçado de qualquer tipo de linha, o que não acontece com o algoritmo de Bresenham

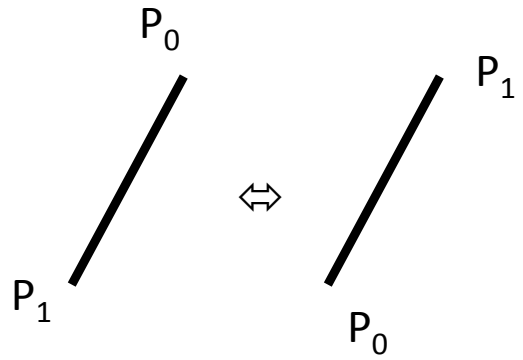
Equação implícita do segmento de recta $f(x,y) = a\,x + b\,y + c$

Equação explícita do segmento de recta $y = m\,x + b = \frac{\Delta y}{\Delta x}\,x + b$

logo $f(x,y) = x\,\Delta y - y\,\Delta x + b\,\Delta x = 0,$

com $a = \Delta y,\ b = -\Delta x,\ c = b\,\Delta x$

Considerando 0 < m < 1 e dy >=0

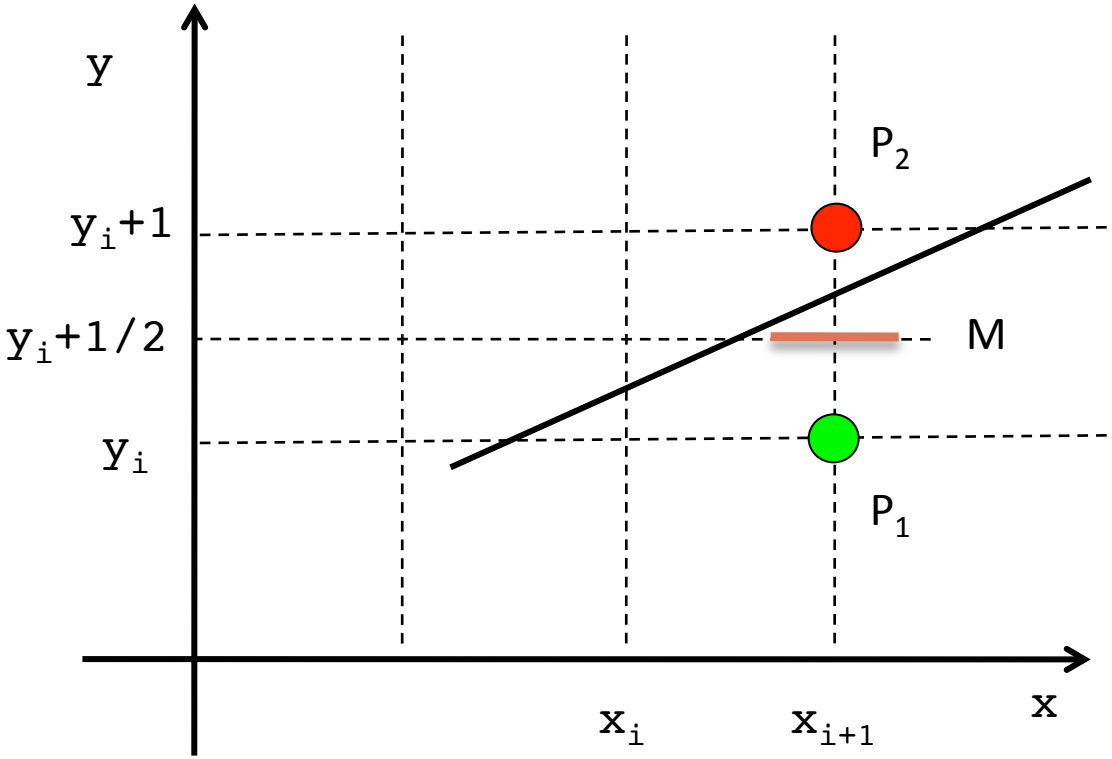


Num dado passo *i*, qual o pixel seguinte ? P_1 ou P_2 ?

Variável de decisão:

$$f(M) = (x_i + 1)\, a + (y_i + 1/2)\, b + c$$

```
if   f(M) < 0
     then P1
     else P2
endif
```



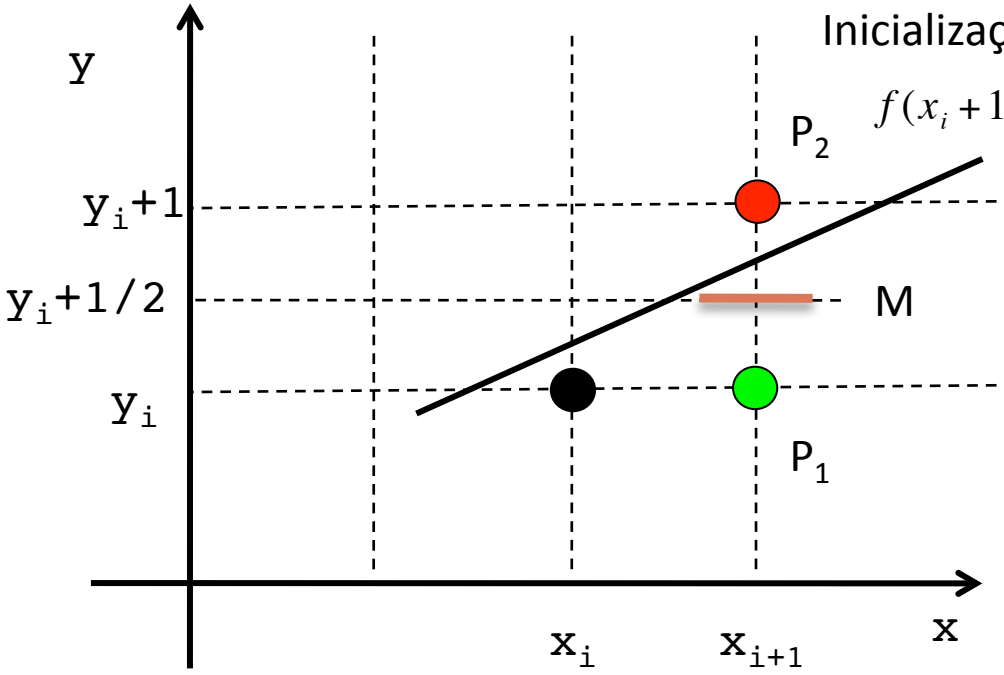
Actualização da variável de decisão $p = f(M)$

Se a escolha foi P_1, o novo M para x_i+2 manterá a ordenada → $p = p + a$

$$p = f(x_i + 2, y_i + 1/2) = (x_i + 2)\,a + (y_i + 1/2)\,b + c$$

Se a escolha foi P_2, o novo M, para x_i+2, terá a sua ordenada incrementada → $p = p + a + b$

$$p = f(x_i + 2, y_i + 3/2) = (x_i + 2)\,a + (y_i + 3/2)\,b + c$$



Inicialização de p no ponto inicial $P(x_0,y_0)$

$$\begin{aligned} f(x_i + 1, y_i + 1/2) &= (x_i + 1)\,a + (y_i + 1/2)\,b + c \\ &= x_i\,a + y_i + b + c + a + b/2 \\ &= f(x_i, y_i) + a + b/2 \\ &= a + b/2 \end{aligned}$$

Será que se obtém o mesmo código que o obtido pelo algoritmo de Bresenham?

Note-se que a e b são inteiros uma vez que as coordenadas dos pontos extremos do segmento de recta também o são. Para evitar aritmética real, utiliza-se como variável de decisão 2p em vez de p, o que não vai afectar o resultado final

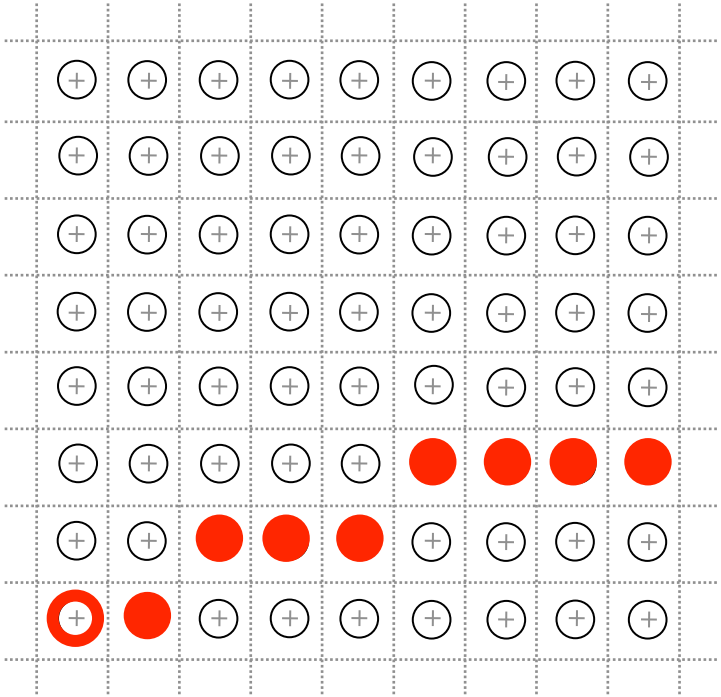
Outras abordagens

Constata-se que as variações na variável dependente obedecem a padrões. Por exemplo, considerando um movimento na horizontal como 0 e na diagonal como 1, será fácil conceber um algoritmo simplificado, desde que se conheça o padrão em causa

0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 ....

1 aparece isolado

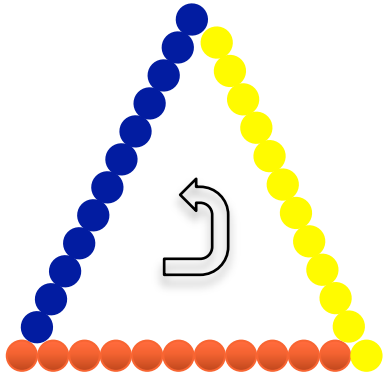
0 aparece em grupos de dois ou três



Activação de pixels

Nem sempre se marca o último pixel da linha

- Uma linha pode ser parte integrante de uma sequência de várias linhas, logo os pontos comuns não devem ser marcados duas vezes (ex.: o que aconteceria numa impressora a cores de jacto de tinta!)

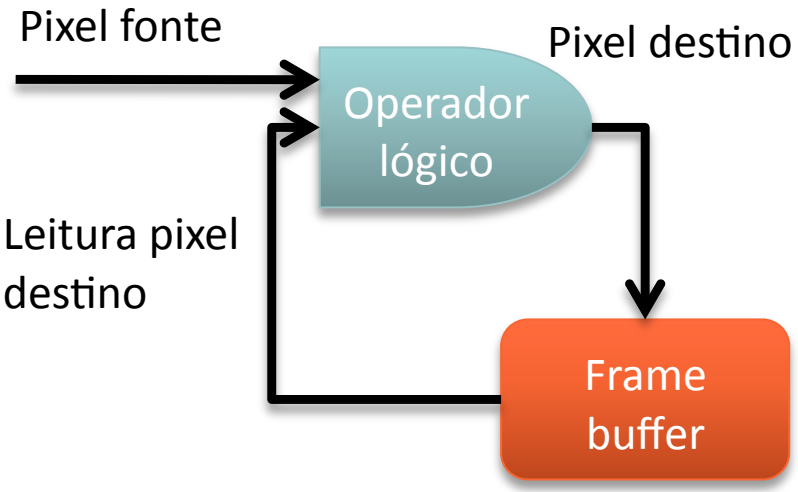


Existem vários modos de escrita/activação de um pixel no frame buffer, com consequências visuais distintas

Escrita de pixels no frame buffer

Modelo de escrita

- Pixel fonte: o que se pretende escrever
- Pixel destino: o que será afectado
- Podem ser utilizados 16 modos de operação lógica
  - Os modos mais relevantes são o modo COPY e o modo XOR



Aplicações do modo XOR

- Apagar linhas – note-se que a aplicação consecutiva do modo XOR permite voltar ao estado inicial (A xor B xor B = A)
- Desenhar cursores no ecrã
- Em consequência dos pontos anteriores, criação do efeito de rubber-banding (muito utilizado em sistemas de CAD)

Fonte	Leitura destino	COPY	XOR
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Curvas

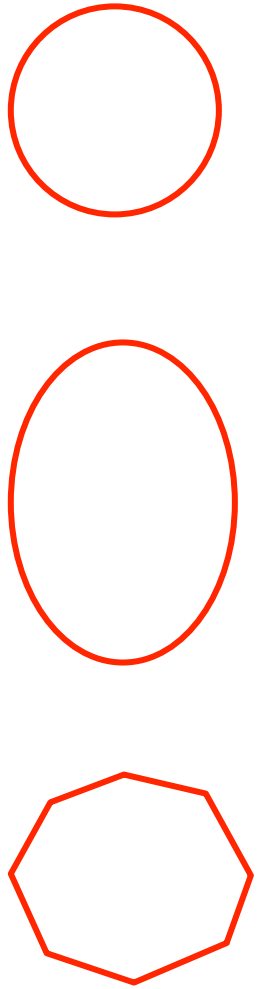
Os algoritmos para representação de curvas podem ser deduzidos seguindo os princípios utilizados para o caso do segmento de recta

A formulação matemática da curva é determinante (ex: equação de coordenadas polares ou equação implícita)

Sempre que possível, tira-se partido da simetria da curva

Como exemplo, refira-se que o algoritmo do Ponto Médio é facilmente ajustável para circunferências e elipses

Também é possível representar uma curva como uma sequência de vários segmentos de recta, i.e. uma poli-linha (tudo depende do binómio qualidade/rapidez desejado)



Desenho de primitivas em Java

The Java™ Tutorials

Download the JDK
Search the Tutorials

« Previous • Trail • Next »

Home Page

Trail: 2D Graphics

This trail introduces you to the Java 2D™ API and shows you how to display and print 2D graphics in your Java programs. The trial is intended for developers who want to enrich their knowledge of the Java 2D API, as well as for beginners in computer graphics. Almost every section contains relevant examples to illustrate specific capabilities. The Java 2D API enables you to easily perform the following tasks:

- Draw lines, rectangles and any other geometric shape.
- Fill those shapes with solid colors or gradients and textures.
- Draw text with options for fine control over the font and rendering process.
- Draw images, optionally applying filtering operations.
- Apply operations such as compositing and transforming during any of the above rendering operations.

This chapter also explains less familiar concepts such as compositing.

Using 2D Graphics API to display complex charts

Using image-filtering operations

This chapter describes the concept of drawing on-screen and off-screen images, as well as surfaces and printer devices. This trail covers the most common uses of the Java 2D APIs and briefly describes some of the more advanced features.

The Graphics2D class, which was released with JDK 1.2, extends the Graphics class to provide more sophisticated control over geometry, coordinate transformations, color management, and text layout. Beginning with JDK 1.2, this is the fundamental class for rendering two-dimensional shapes, text and images.

Programa de demonstração

Modos XOR e COPY em Java

setXORMode(Color c1)

Sets the paint mode of this graphics context to alternate between this graphics context's current color and the new specified color. This specifies that logical pixel operations are performed in the XOR mode, which alternates pixels between the current color and a specified XOR color.
When drawing operations are performed, pixels which are the current color are changed to the specified color, and vice versa.
Pixels that are of colors other than those two colors are changed in an unpredictable but reversible manner; if the same figure is drawn twice, then all pixels are restored to their original values.

setPaint()

Sets the paint mode of this graphics context to overwrite the destination with this graphics context's current color. This sets the logical pixel operation function to the paint or overwrite mode. All subsequent rendering operations will overwrite the destination with the current color.

Programa de demonstração

Sumário

Rasterização de primitivas

- Pontos e linhas
- Algoritmos DDA, de Bresenham e do Ponto Médio
- Activação de pixels

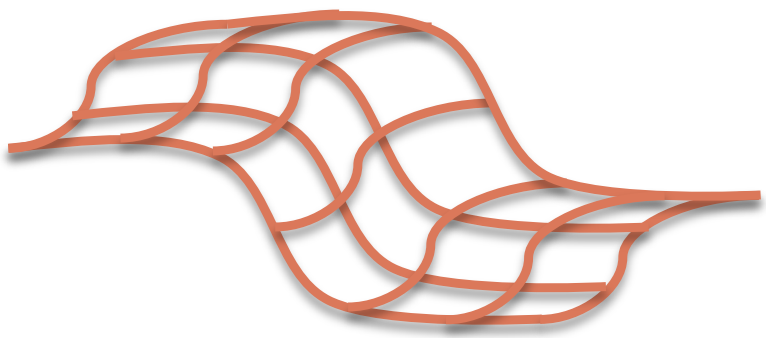
Preenchimento de áreas

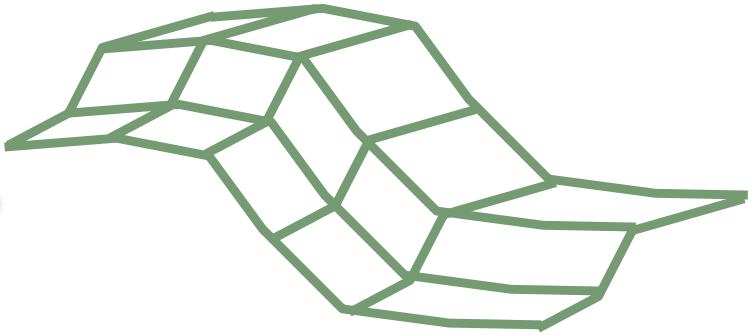
- Algoritmo Par-Ímpar para polígonos
- Preenchimento de área de pixels

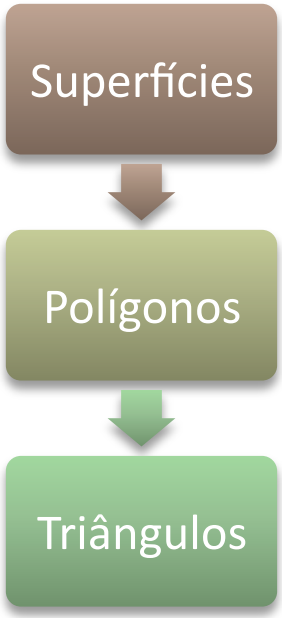
Atributos gráficos

- Grossura e estilos de linhas
- Antialiasing

Polígonos







As malhas poligonais são elementos fundamentais dos sistemas gráficos

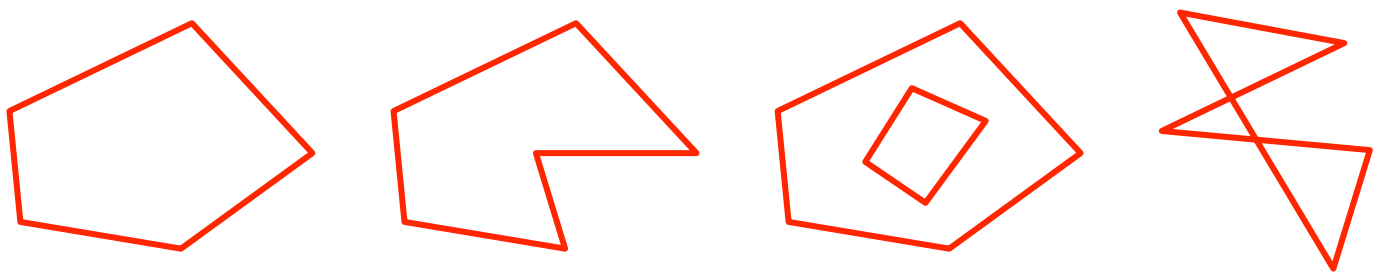
Os triângulos são polígonos planares, com apenas 3 vértices, convexos, em suma de tratamento matemático simples. Nomeadamente, podem ser processados ao nível de hardware

Preenchimento de polígonos

Um polígono num plano é definido pela sequência dos respectivos vértices, independentemente do conteúdo do frame buffer

Um polígono pode apresentar várias topologias: convexo, côncavo, com buracos, e com arestas que se auto-intersectam

Considera-se que a representação da fronteira de um polígono não é da responsabilidade de um algoritmo de preenchimento (fill area)



Algoritmo Par-Ímpar

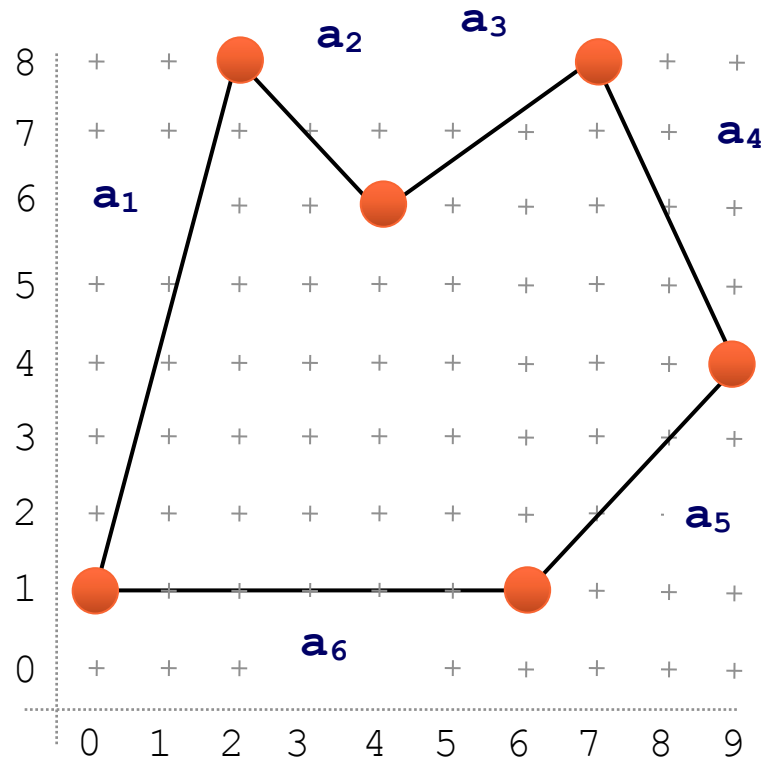
Preenchimento de um polígono definido pela sequência dos seus vértices

Ex: polígono com **6 vértices**

(0,1), (2,8), (4,6), (7,8), (9,4), (6,1)

e **6 arestas**

a_1, a_2, a_3, a_4, a_5, a_6



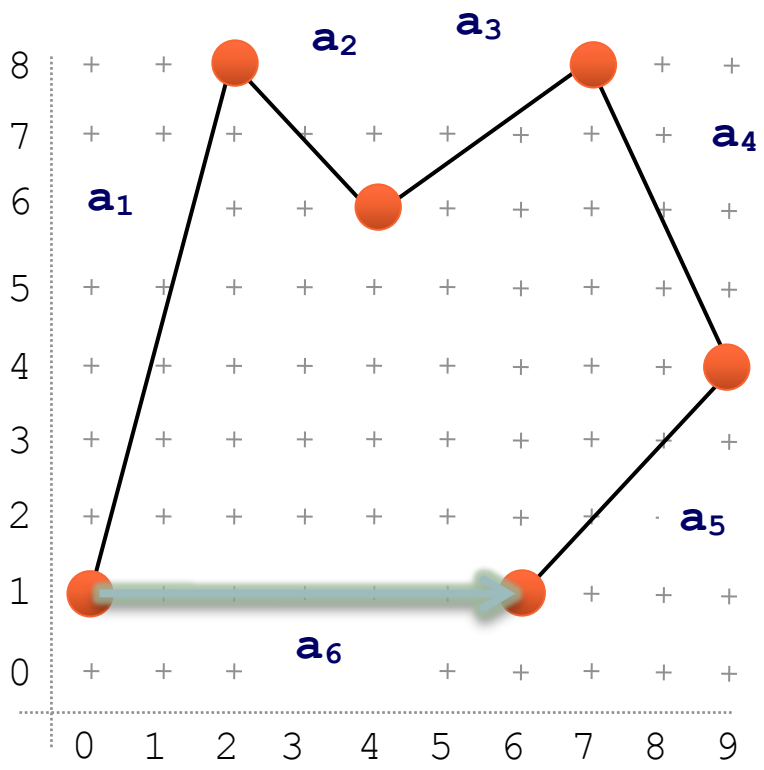
Preenchimento da área através de varrimento por linhas – *scan-lines*

1. Para um dada linha y = k, determinar as intersecções com arestas
2. Juntar esses pares de arestas consecutivas, activando conjuntos sucessivos de pixels

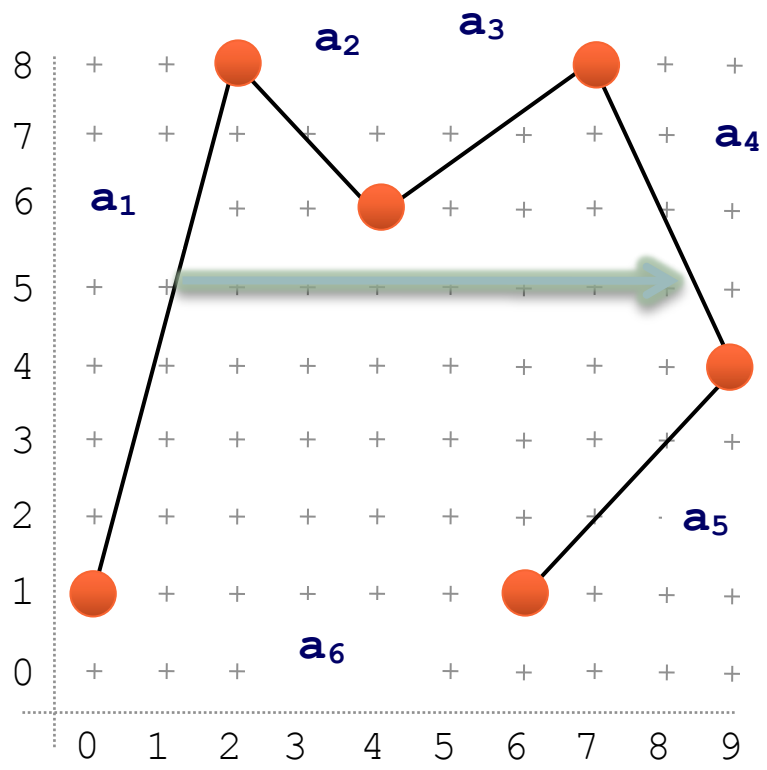
e tirando partido da coerência por

Linha de varrimento	Se um dado pixel duma linha pertence ao polígono, então é provável que os pixels vizinhos nessa linha também pertençam
Aresta do polígono	Se uma aresta intersecta uma dada linha de varrimento, então é provável que também intersecte as linhas de varrimento vizinhas

Arestas horizontais são ignoradas (ex: a_6 é desenhada implicitamente)



Linha de varrimento y = 5



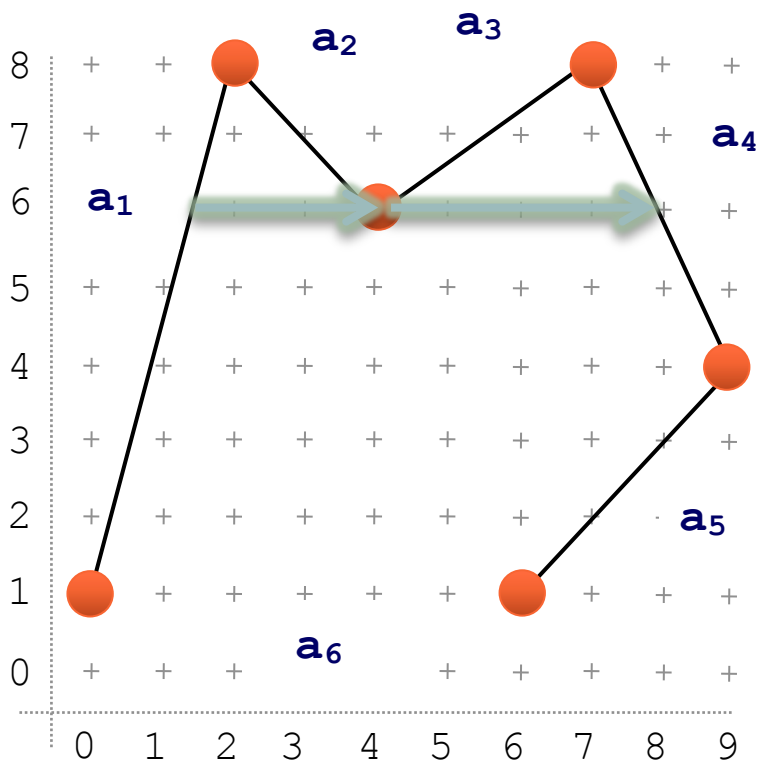
a_1 : intersecção x = 1.14

a_4 : intersecção x = 8.5

Os pixels 2 a 8 na linha de varrimento y = 5 são desenhados

. . . e os pixels 1 e 9 ?

Linha de varrimento y = 6



a_1 : intersecção x = 1. 4

a_2 : intersecção x = 4

a_3 : intersecção x = 4

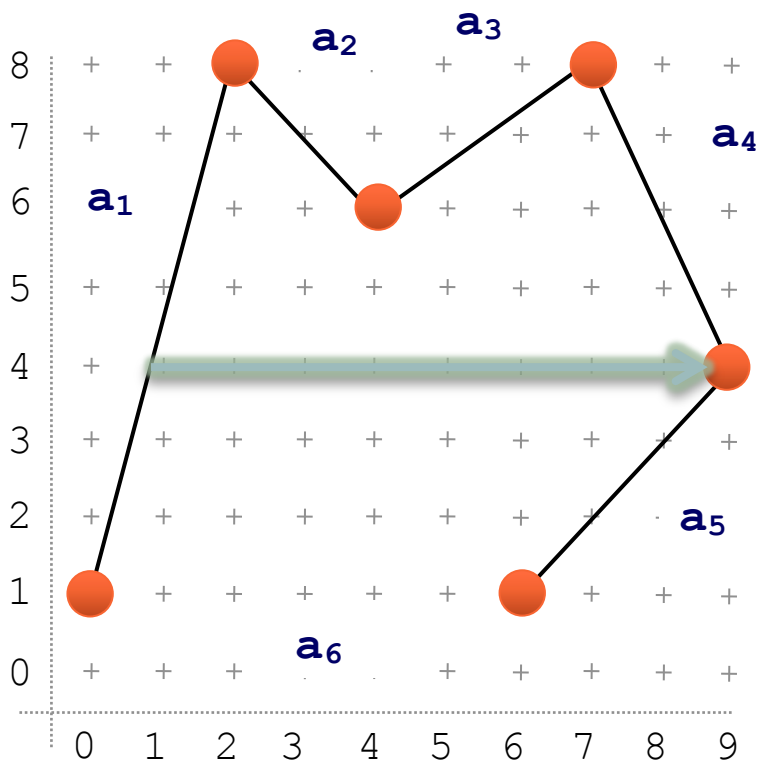
a_4 : intersecção x = 8

As arestas a_2 e a_3 encontram-se no mesmo lado relativamente à linha de varrimento. É um extremo local

São desenhadas as duas partes

Linha de varrimento y = 4

Número ímpar de intersecções, o que constituí um problema



a_1 : intersecção x = 0.9

a_4 : intersecção x = 9

a_5 : intersecção x = 9

As arestas a_4 e a_5 encontram-se em lados diferentes relativamente à linha de varrimento. Não é um extremo local

Só conta uma intersecção. A aresta a_5 é removida na linha anterior (y = 3)

O preenchimento faz-se entre cada par de intersecções com aresta, mas excluindo aquelas cuja ordenada máxima seja igual à linha de varrimento

Estruturas de dados

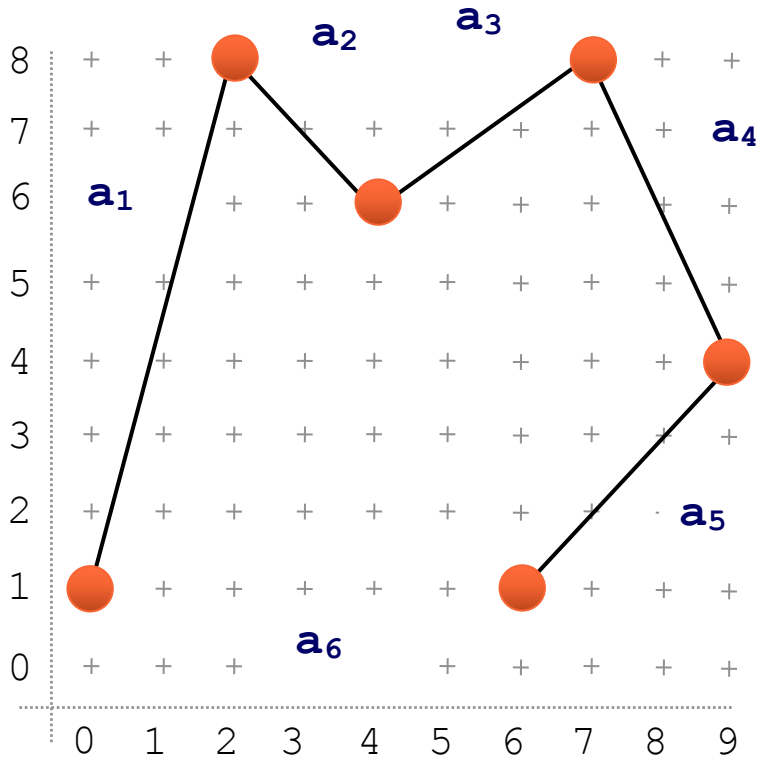
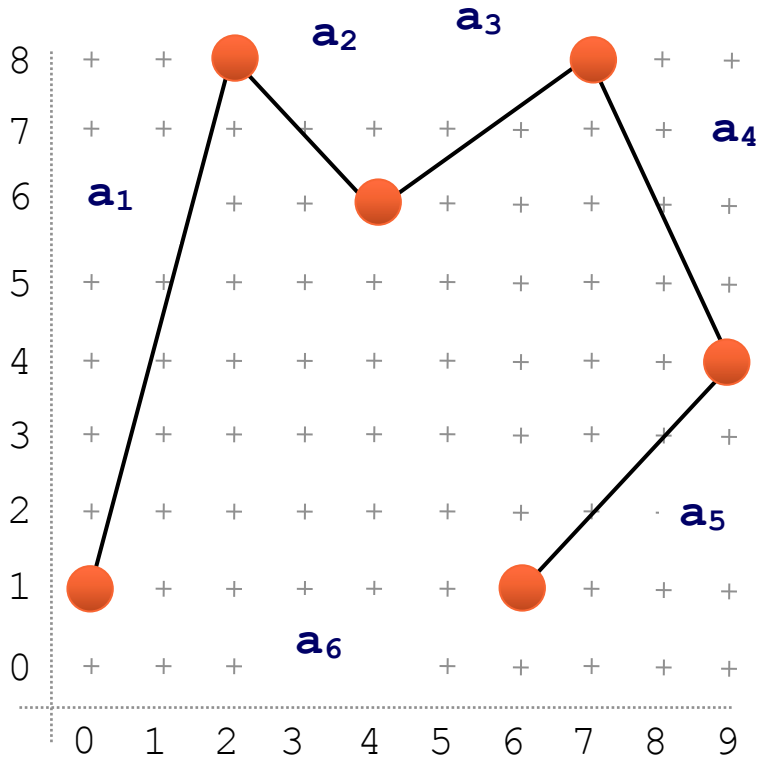


Tabela de **A**restas (TA)

Linha de varrimento (ordem em y)	Novas arestas intersectadas (ordem em x)
6	a_2 , a_3
4	a_4
1	a_1 , a_5

intersecção com a linha de varrimento (informação dinâmica, inicializada com o valor de x associado ao y mínimo da aresta



	ymax	x	1/m
a_1	8	0	2/7
a_2	8	4	-2/2
a_3			
a_4			
a_5	4	6	3/3

Estruturas de dados

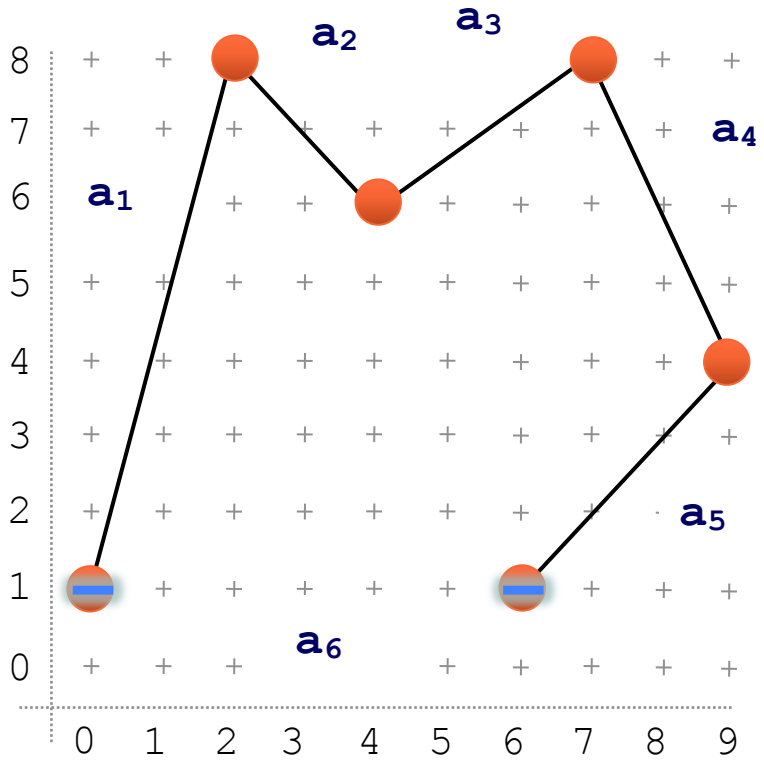


Tabela de **A**restas **A**ctivas (TAA)

coerência por arestas: na linha seguinte tem-se

$$y_{i+1} = y_i + 1$$
$$x_{i+1} = x_i + 1/m$$

y = 1

a1	8	0	2/7
a5	4	6	3/3

ymax x 1/m

TAA – estrutura dinâmica sempre ordenada por x

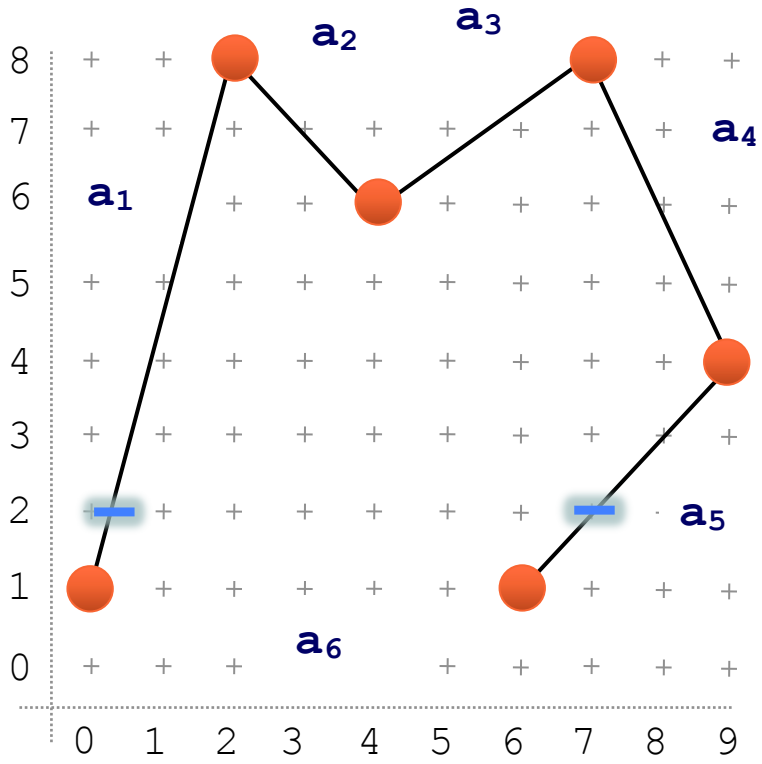


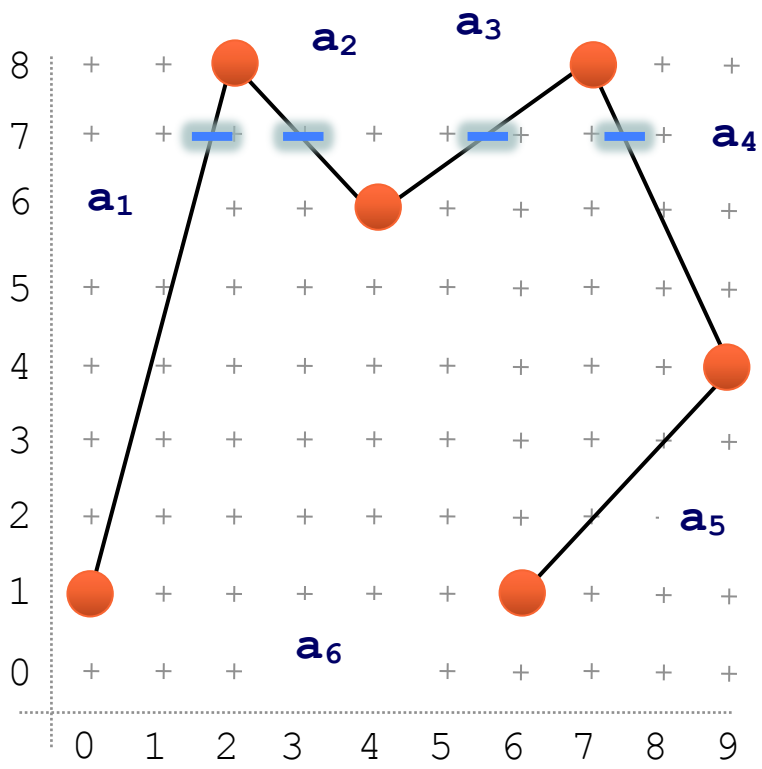
Tabela de **A**restas **A**ctivas

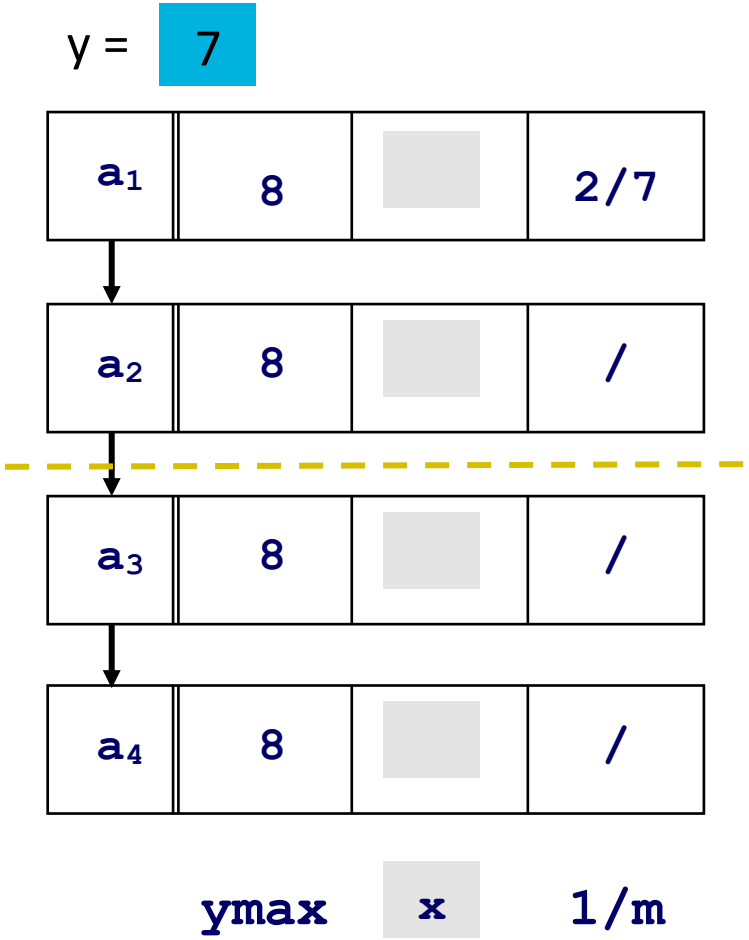
y = 2

a_1	8	2/7	2/7
a_5	4	7	3/3

ymax x 1/m

TAA – estrutura dinâmica sempre ordenada por x





TAA – estrutura dinâmica sempre ordenada por x

Considerações finais relativas à fronteira do polígono

O algoritmo apenas deve considerar pontos interiores ao polígono

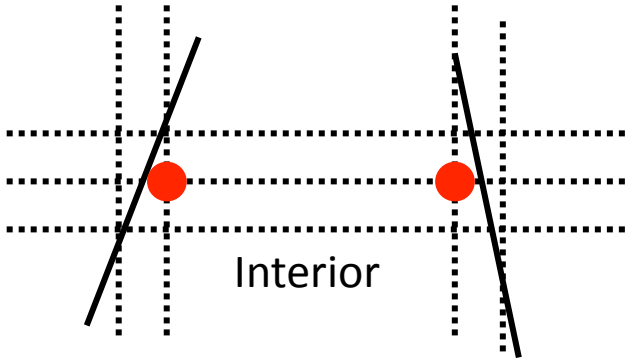
- Deve-se pois garantir que não são marcados pontos exteriores ao polígono
- Se se pretende visualizar as suas arestas, então estas devem ser desenhadas autonomamente e após o preenchimento do polígono

Ajustamentos

- Utilização de operações fraccionárias de modo a evitar erros por acumulação resultantes da operação x = x + 1/m
- Quando o valor de x não é um inteiro, deve-se arredondar para cima ou então truncar de modo a que o ponto a marcar não seja exterior ao polígono
- Para evitar sobreposição de polígonos com fronteira comum, segue-se uma convenção relativamente aos pixels de fronteira

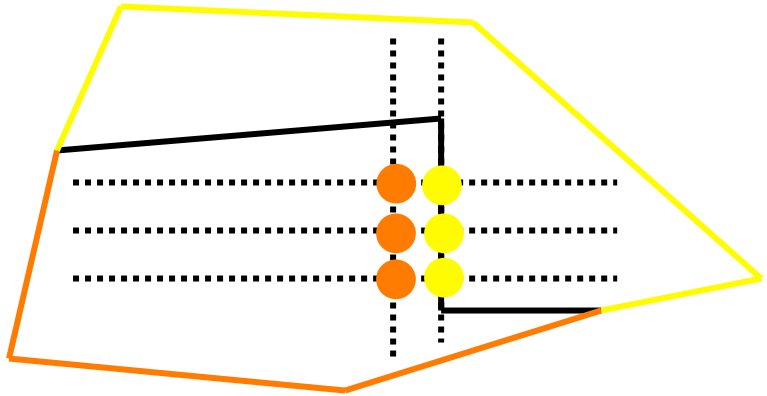
x não é inteiro

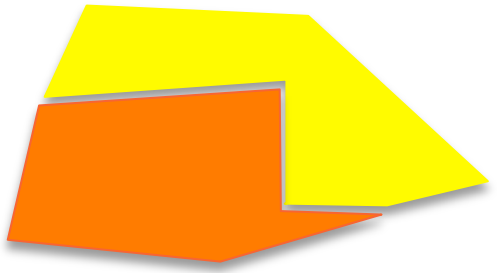
De fora para dentro	*Arredondar para cima*
De dentro para fora	*Truncar*



x é inteiro (Obs.: é uma convenção possível)

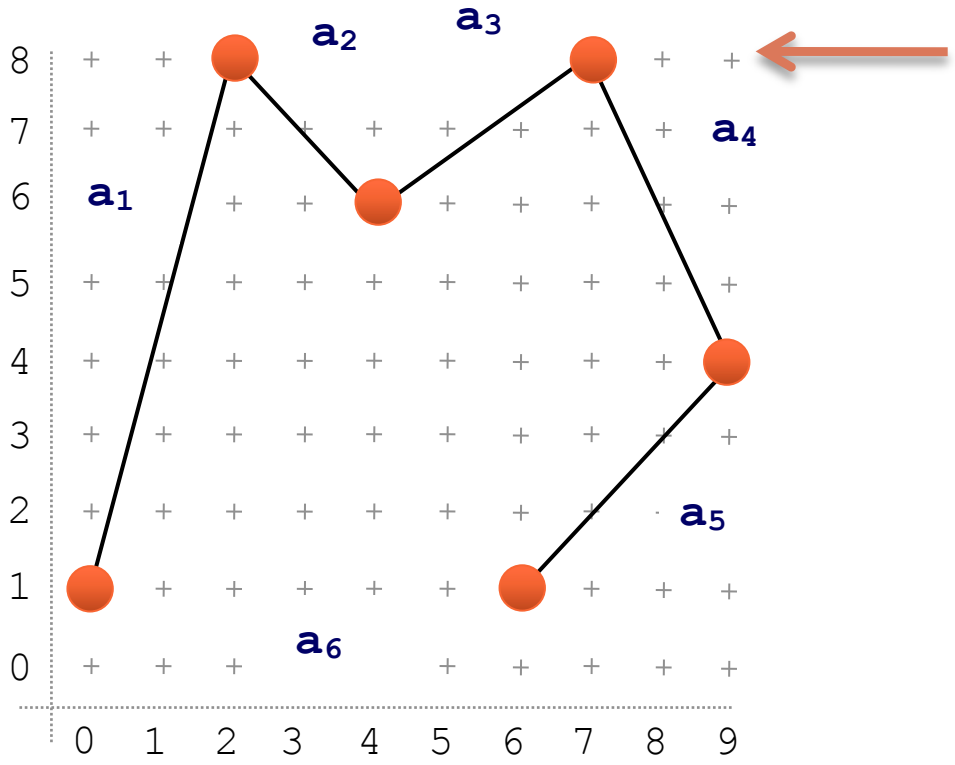
Pixel à esquerda	*É considerado interior*
Pixel à direita	*É considerado exterior*

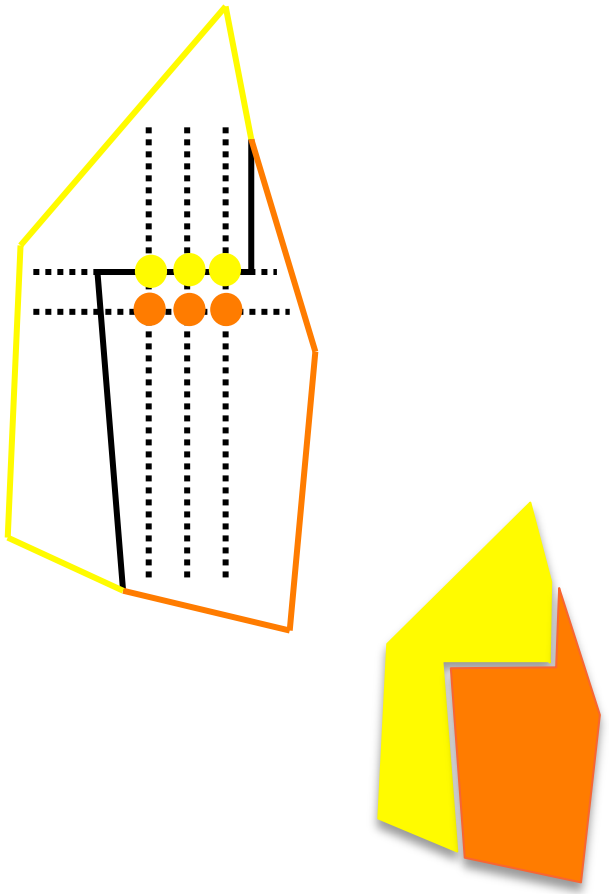




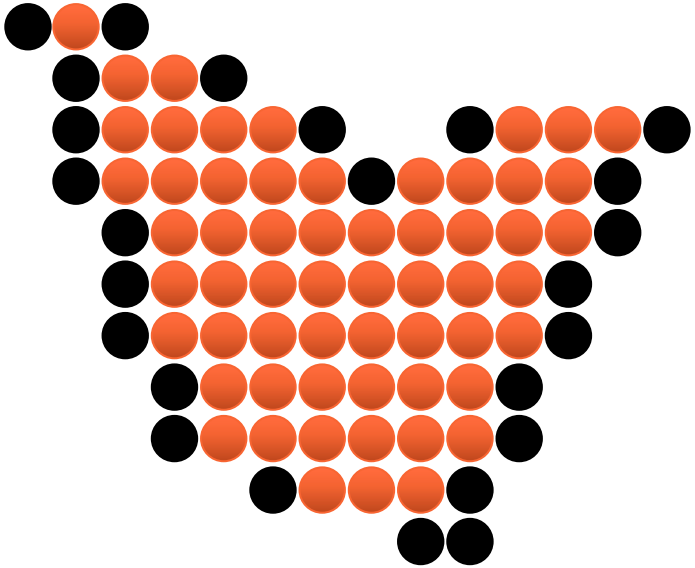
Linha de varrimento y = 8 ?

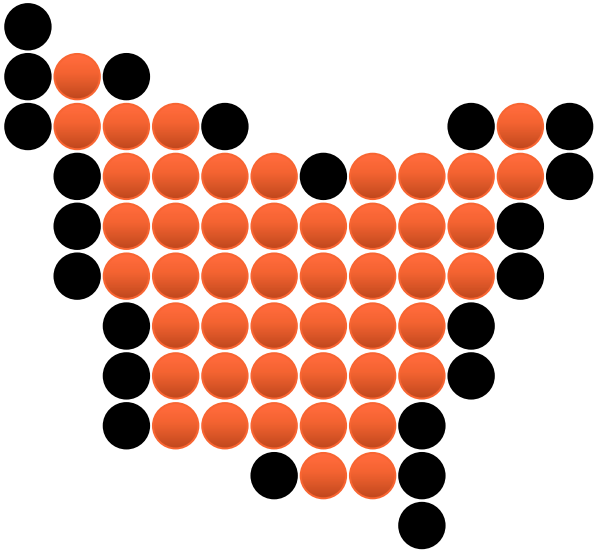
TAA vazia, ou seja, não existe linha de varrimento para o y máximo das arestas.
Garante-se assim a não sobreposição de polígonos adjacentes na vertical





Exemplos





Algoritmo

1. Construir TA (seja max o y da última entrada em TA)
2. TAA ← nil
3. y ← primeira entrada na TA não vazia, caso exista, senão max+1
4. Enquanto TAA <> nil ou y < max+1
    1. remover da TAA as arestas com ymax = y (*)
    2. inserir na TAA as arestas referidas em TA para a entrada y
    3. ordenar TAA segundo x
    4. preencher os pixels entre os valores de x de cada par de arestas
    5. y ← y + 1
    6. actualizar o x de cada aresta em TAA, segundo x ← x + 1/m

(*) deste modo não existe sobreposição de polígonos na vertical

---

Obs.: O algoritmo também é aplicável a arestas que se cruzem entre si. Assim, convém definir o conceito de ponto interior a um polígono

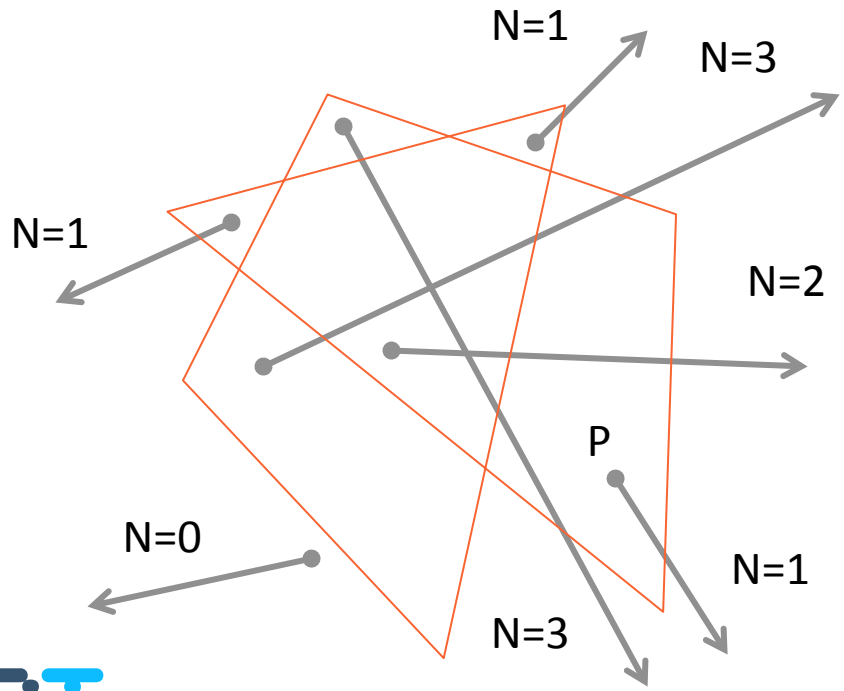
Ponto interior a um polígono

Um método, denominado teste ímpar-par:

1. Considere-se um segmento de recta com início em P e até ao infinito, não passando por vértices

2. Determina-se o número de intersecções N do segmento com as arestas do polígono

3. Se N for ímpar, o ponto P é interior. Caso contrário, é exterior

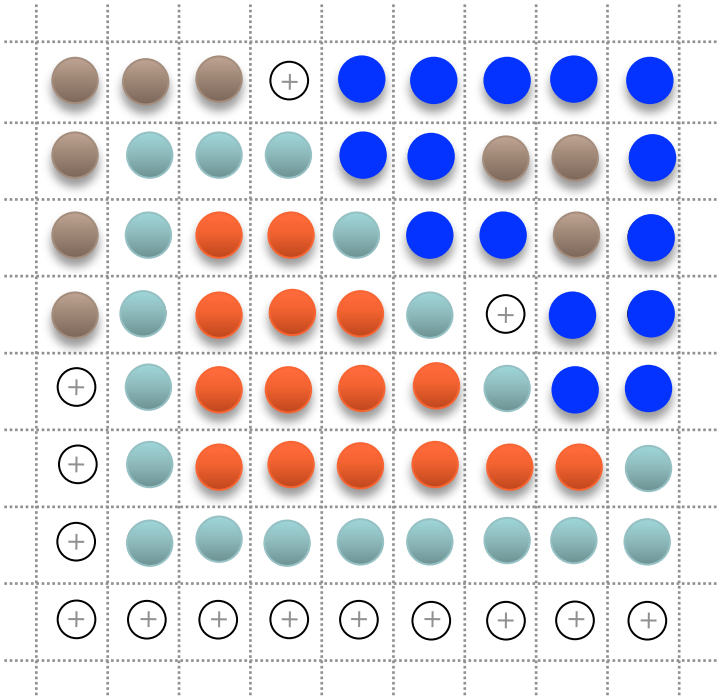


Preenchimento de área de pixels

A área será constituída por um conjunto de pixels adjacentes com igual valor/ cor

O preenchimento ocorre após a definição da fronteira, a qual pode ser irregular

Duas variantes algorítmicas: boundary-fill e flood-fill

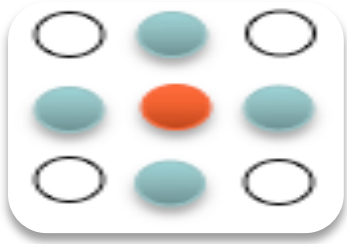


Boundary-fill

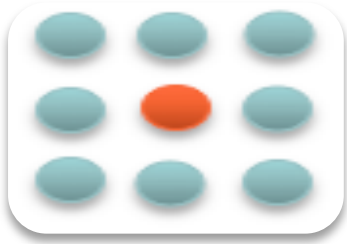
Começa num ponto do interior da área e preenche-a até atingir a sua fronteira

```
void boundaryFill( int x, int y, int fill, int boundary ) {
    int current = getPixel(x, y);
    if ((current != boundary) && (current != fill))  {
          setPixel(x, y, fill);
          boundaryFill(x+1, y, fill, boundary);
          boundaryFill(x, y+1, fill, boundary);
          boundaryFill(x-1, y, fill, boundary);
          boundaryFill(x, y-1, fill, boundary);

           boundaryFill(x+1, y+1, fill, boundary);
          boundaryFill(x+1, y-1, fill, boundary);
          boundaryFill(x-1, y-1, fill, boundary);
          boundaryFill(x-1, y+1, fill, boundary);
    }
}
```



Ligação a 4



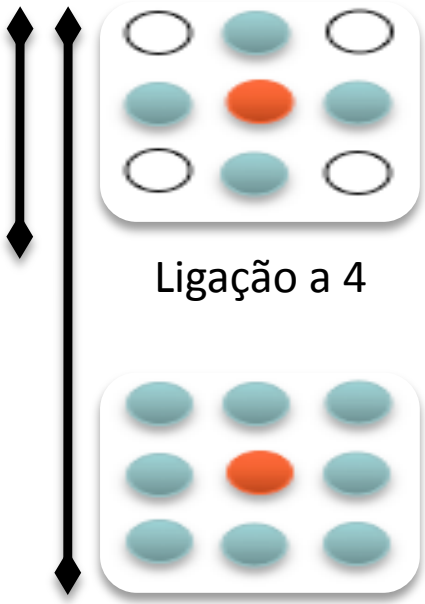
Ligação a 8

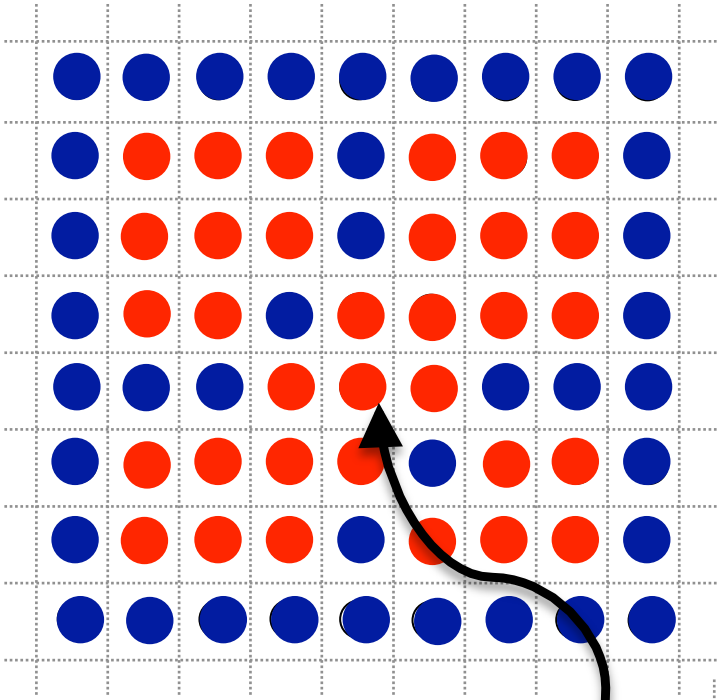
Flood-fill

Começa num ponto e liga todos os pontos seleccionados (old) com a mesma cor (fill). É uma variante do boundary-fill

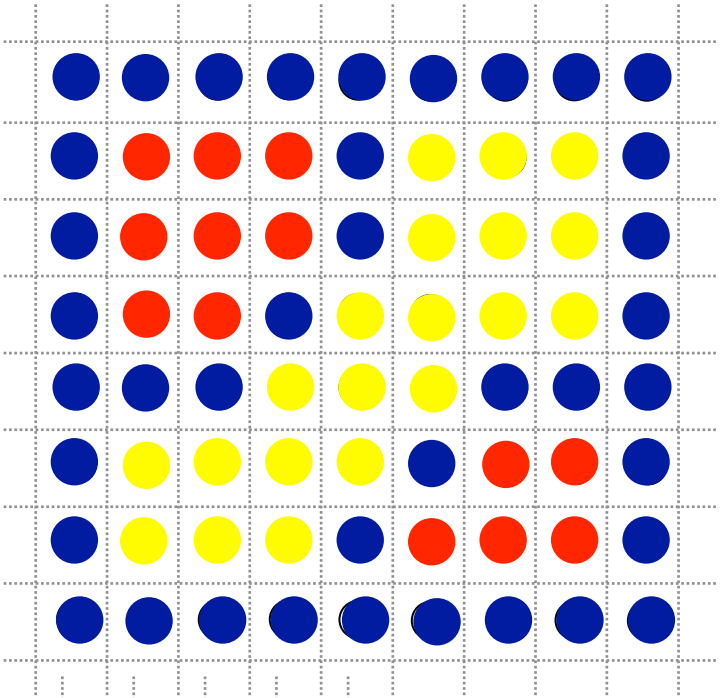
```
void floodFill( int x, int y, int fill, int old ) {
    int current = getPixel(x, y);
    if ((current == old))  {
       setPixel(x, y, fill);
       floodFill(x+1, y, fill, old);
       floodFill(x, y+1, fill, old);
       floodFill(x-1, y, fill, old);
       floodFill(x, y-1, fill, old);

       floodFill(x+1, y+1, fill, old);
       floodFill(x+1, y-1, fill, old);
       floodFill(x-1, y-1, fill, old);
       floodFill(x-1, y+1, fill, old);
    }
}
```





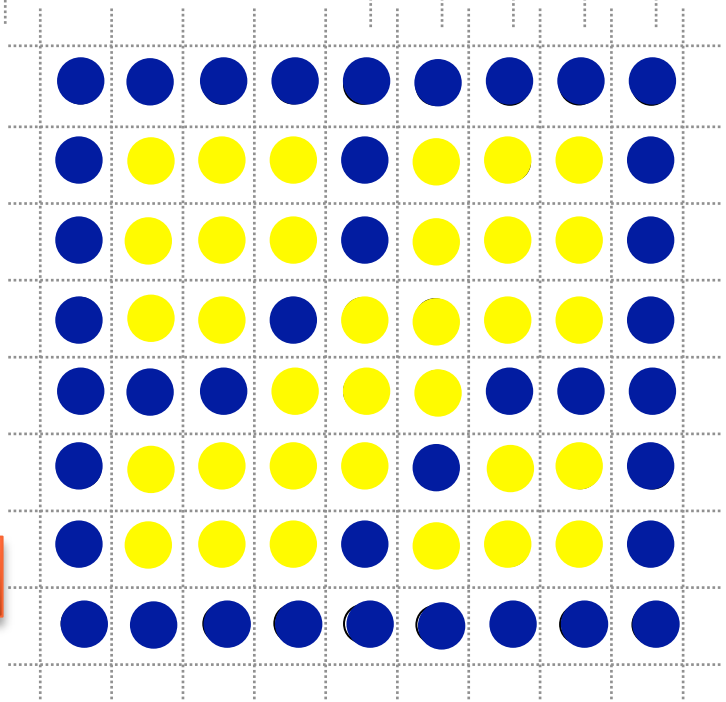
Ligação a 4



Flood fill com origem em

old

fill



Ligação a 8

Programa de demonstração

Sumário

Rasterização de primitivas

- Pontos e linhas
- Algoritmos DDA, de Bresenham e do Ponto Médio
- Activação de pixels

Preenchimento de áreas

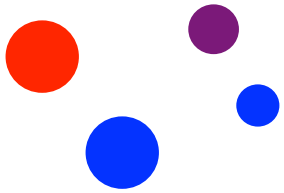
- Algoritmo Par-Ímpar para polígonos
- Preenchimento de área de pixels

Atributos gráficos

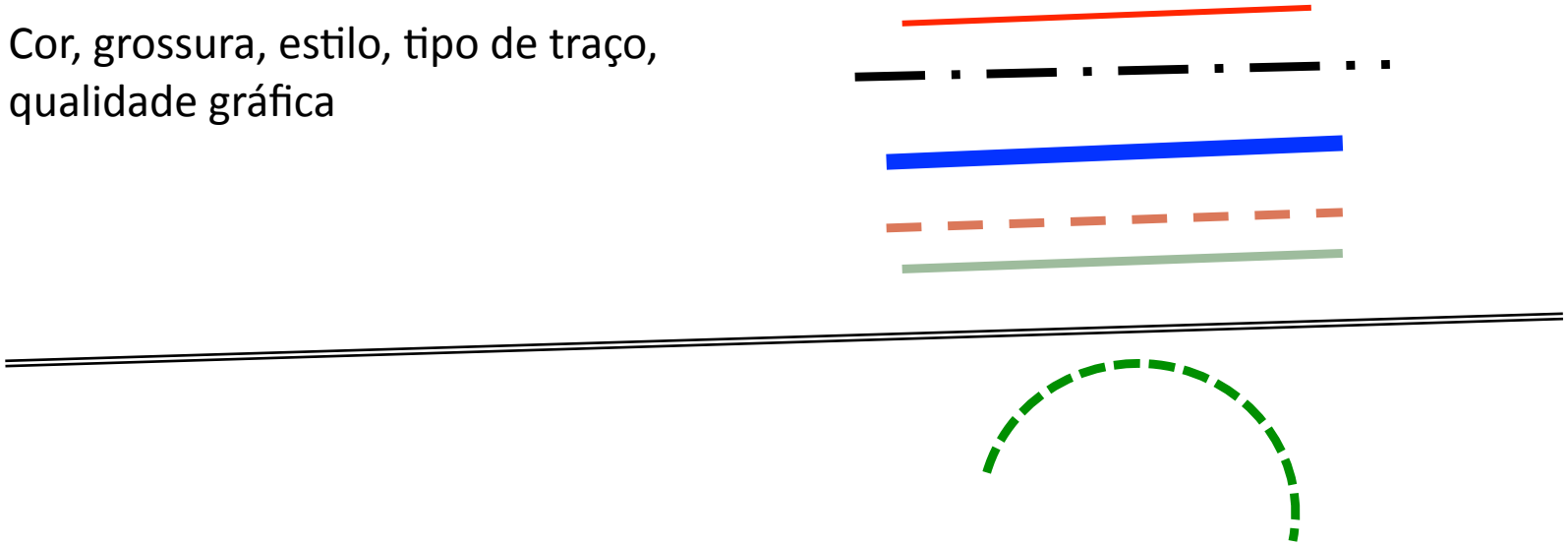
- Grossura e estilos de linhas
- Antialiasing

Atributos de pontos e linhas

Cor e tamanho



Cor, grossura, estilo, tipo de traço, qualidade gráfica



Voltaremos a este assunto mais tarde …