Algoritmo Scanline

SCC0250 - Computação Gráfica

Profa. Maria Cristina F. Oliveira

Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)

13 de novembro de 2023

Preenchimento de Áreas

- *Scanline*: algoritmo de conversão matricial (rasterização) por preenchimento de área
- APIs gráficas, em geral, admitem um conjunto limitado de primitivas geométricas:
  - **Polígonos**, que são de manipulação simples, pois são descritos por **equações lineares**
  - **Polígonos convexos**, simples de rasterizar utilizando o algoritmo *scanline*
  - scanline = linha de varredura
  - No caso de polígonos convexos, uma *scanline* intercepta no máximo **duas arestas** do polígono

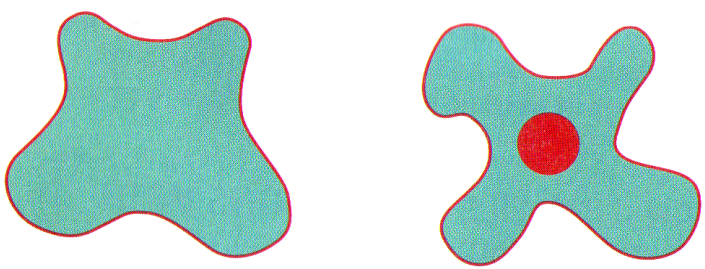
- A limitação a polígonos convexos se deve à implementação das APIs
- Em princípio, é possível preencher o interior de qualquer tipo de forma geomética (p.ex., diversas ferramentas de desenho oferecem tal recurso)

Preenchimento de Áreas

- Existem basicamente duas abordagens para preenchimento de áreas

- Polígonos
  - Considera linhas de varredura (*scanlines*), i.e., sequência de pixels em uma mesma linha horizontal
  - Determina quais blocos de pixels estão no interior do polígono (e devem ser coloridos)

- Objetos de formato arbitrário
  - A partir de um ponto inicial, expande a vizinhança colorindo os pixels, até encontrar as **bordas**



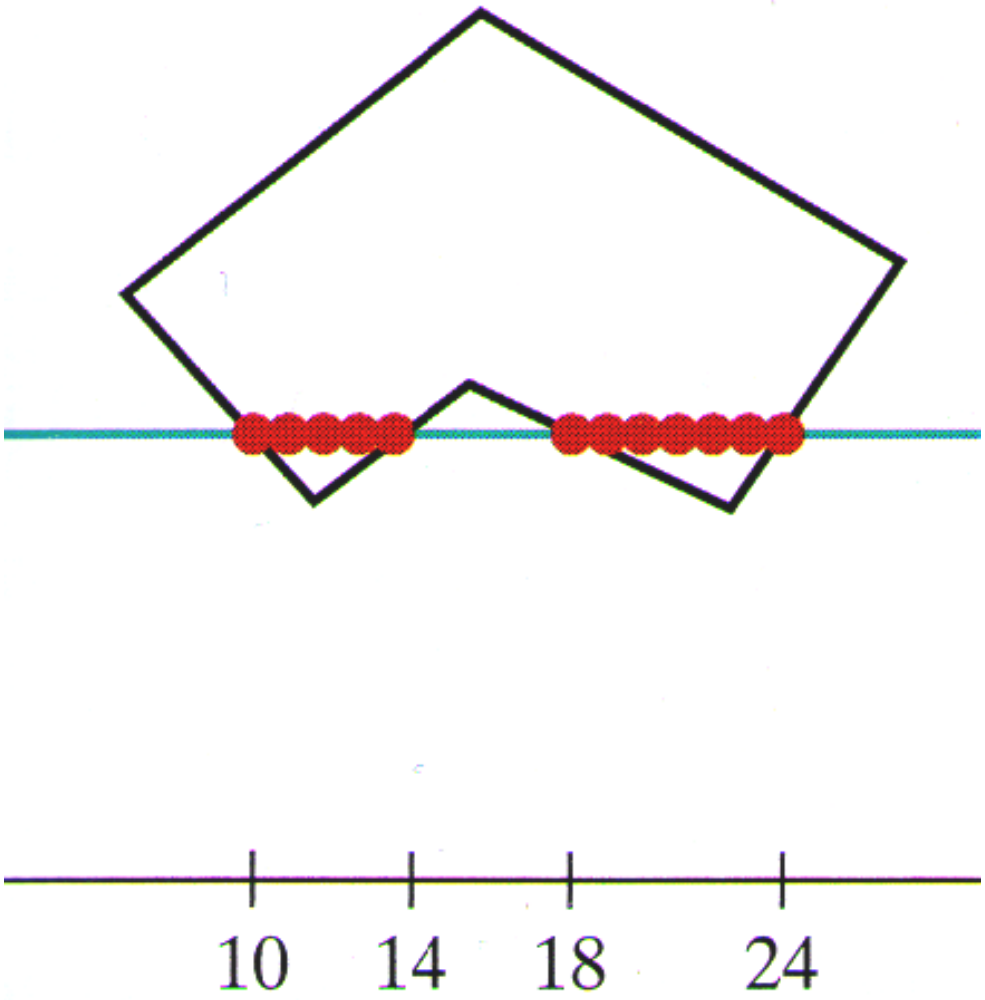
Algoritmo Scanline

- *Scanline*: linha de varredura no processo de exibição de imagens
- Algoritmo processa uma scanline por vez
- Determina as **intersecções** das *scanlines* com o **polígono**

- Os **segmentos** da *scanline* que ficam **dentro** do polígono são **coloridos** (ou tonalizados com o modelo de iluminação!)
  - Usa a **regra da paridade ímpar** como critério para determinar o interior/exterior

- Computa as intersecções a partir de duas equações lineares: a equação da linha de varredura e a equação de uma aresta do polígono
- Calcula as intersecções da esquerda para a direita

Algoritmo Scanline



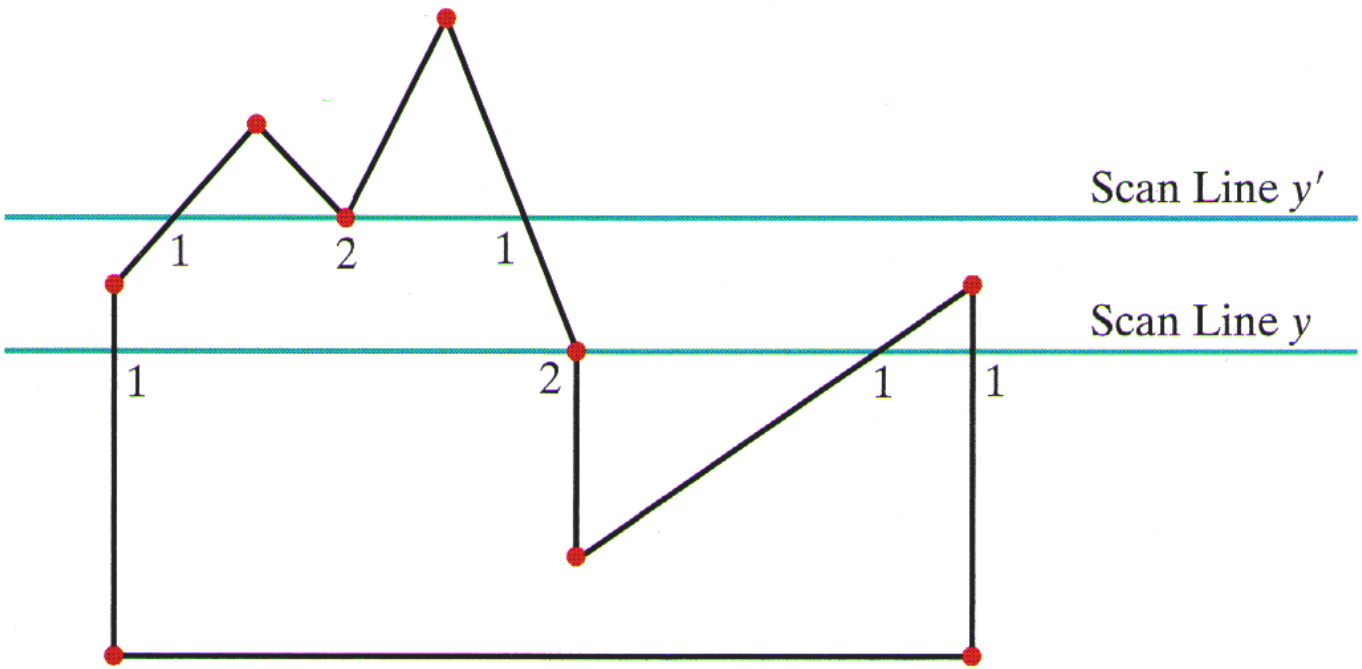
Regra Par-Impar

- O algoritmo *Scanline* considera as intersecções da linha de varredura com as arestas do polígono, iniciando com paridade par e invertendo a paridade a cada intersecção encontrada
- Regiões associadas à paridade ímpar estão no interior do polígono
- Pixels no interior devem ser 'pintados´

- Funciona?

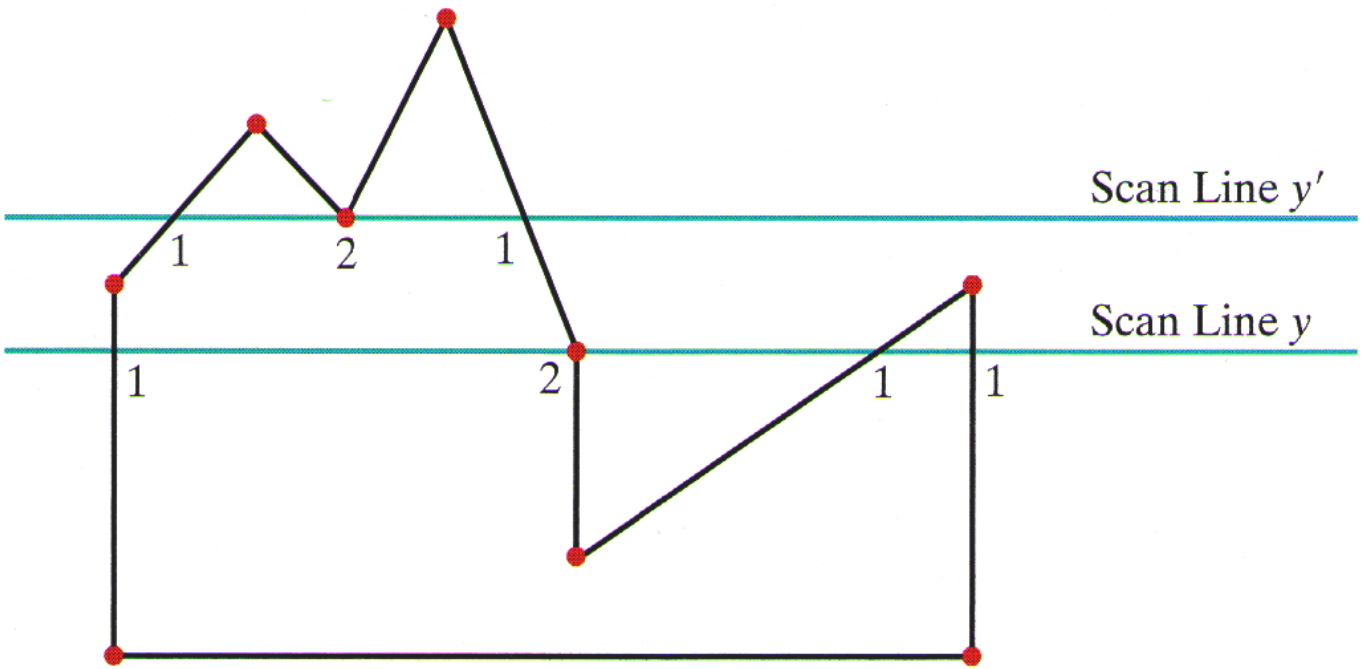
Problema das Intersecções

- **Problemas** quando a *scanline* intercepta um **vértice**
  - Intercepta duas arestas simultaneamente
  - Como contabilizar essas interseções para o teste da paridade? Conta uma ou duas intersecções?



Problema das Intersecções

- A **contagem** de intersecção deve considerar a **topologia** da aresta em relação à linha de varredura
  - Duas arestas em **lados opostos** da *scanline*: conta **uma intersecção**
  - Duas arestas do **mesmo lado** da *scanline*: conta **duas intersecções**

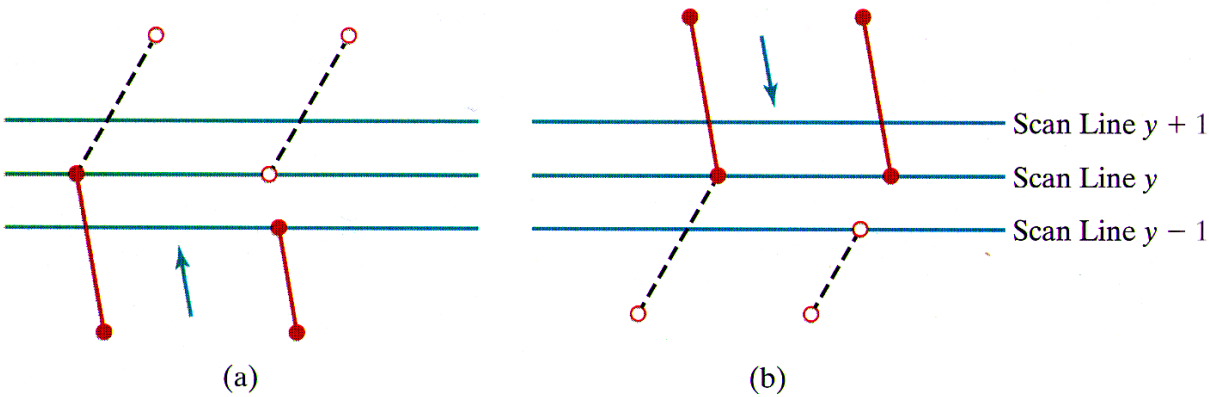


Solução para o Problema das Intersecções

- Para descobrir se as arestas estão em lados opostos ou não
  - Percorrer a **fronteira** do polígono no sentido **anti-horário** (ou horário) e observar as mudanças no valor da coordenada y dos vértices
  - Se o valor da coordenada y dos três vértices de duas **arestas consecutivas** são monotonicamente **crescentes** (ou decrescentes), conta somente **uma intersecção**
  - Caso **contrário**, conta **duas**

Solução para o Problema da Scanline

- Outra solução (pré-processamento)
  - Percorrer as arestas no sentido **anti-horário** (ou horário) e verificar se as coordenadas y dos 3 vértices consecutivos são monotonicamente **crescentes** (decrescentes)
  - Nesse caso, pode-se reduzir a **aresta inferior** para assegurar somente uma intersecção



Algoritmo Scanline (Intersecção)

- Para calcular a interseção da i-ésima *scanline* com uma aresta $\{(x_1, y_1), (x_2, y_2)\}$ tem-se as equações da *scanline* e da aresta

$$y = i$$

$$y = m * x + b$$
$$m = (y_2 - y_1)/(x_2 - x_1)$$

Algoritmo Scanline (Intersecção)

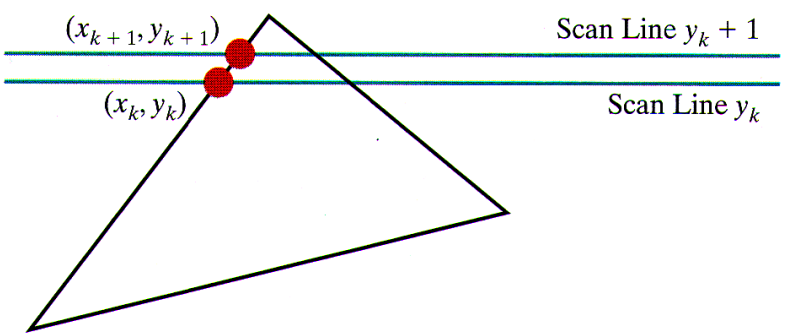
- É possível acelerar esse processo com uma abordagem incremental
- Observe que as *scanlines* sõ processadas em sequencia
- O que acontece com duas *scanlines* consecutivas?

$$m = (y_{k+1} - y_k)/(x_{k+1} - x_k)$$

- Considere as *scanlines* consecutivas y_{k+1} e y_k

$$y_{k+1} - y_k = 1$$

$$m = 1/(x_{k+1} - x_k)$$
$$x_{k+1} = x_k + 1/m$$



Algoritmo Scanline

- É possível usar somente aritmética inteira lembrando que

$$m = \frac{\Delta y}{\Delta x}$$

- Então

$$x_{k+1} = x_k + \frac{\Delta x}{\Delta y}$$

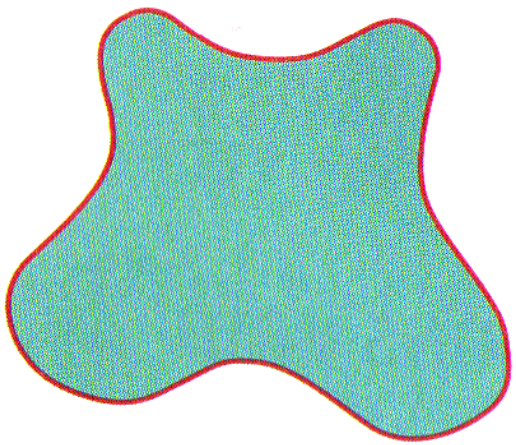
Algoritmo Scanline

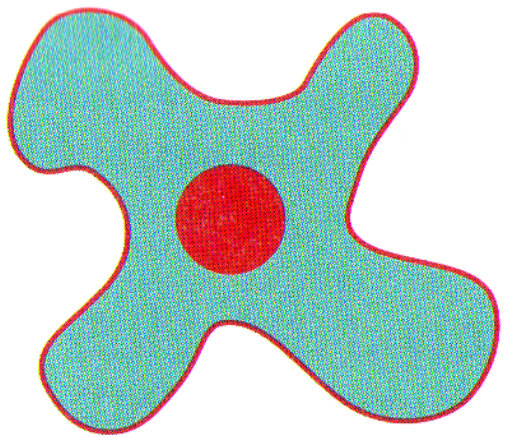
- Para **polígonos convexos**, existe **um único bloco de pixels** adjacentes em cada *scanline*
  - Só **processa** a *scanline* até encontrar **duas intersecções**

- Versão muito mais simples do algoritmo anterior
  - Intersecções nas arestas não apresentam dubiedade

Preenchimento de Regiões Irregulares

- É possível preencher uma região irregular selecionando um pixel e **pintando** os **pixels vizinhos** até alcançar as bordas

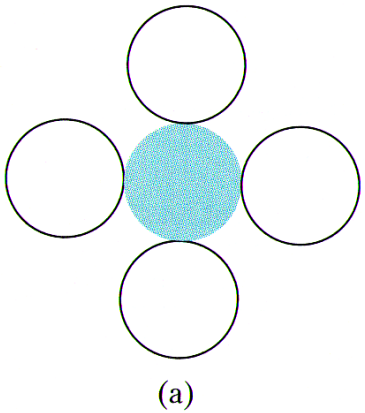




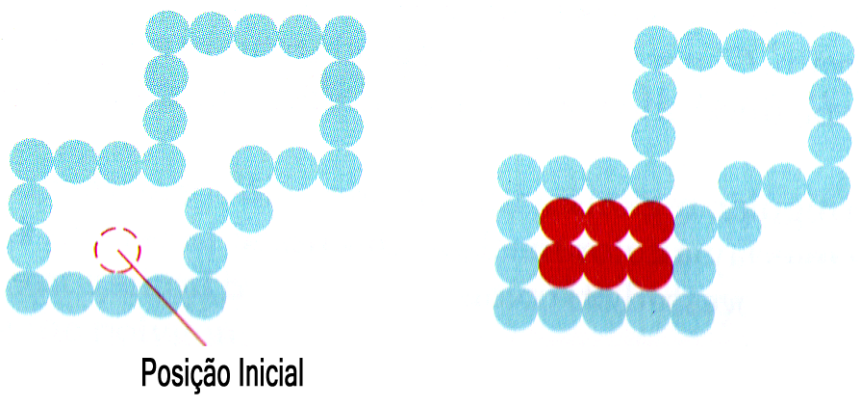
Algoritmo de Preenchimento de Borda

- Se a **borda** de uma região tem a **mesma cor**, é possível preencher essa região pixel por pixel até atingir a cor da borda
  - Normalmente usado em programas gráficos
  - Parte de um ponto inicial (x, y) e testa os vizinhos para ver a cor, se não for borda, preenche

Algoritmo de Preenchimento de Borda



(a) Diferentes testes de vizinhança



(b) Problemas com a máscara de quatro vizinhos

Algoritmo de Preenchimento de Borda

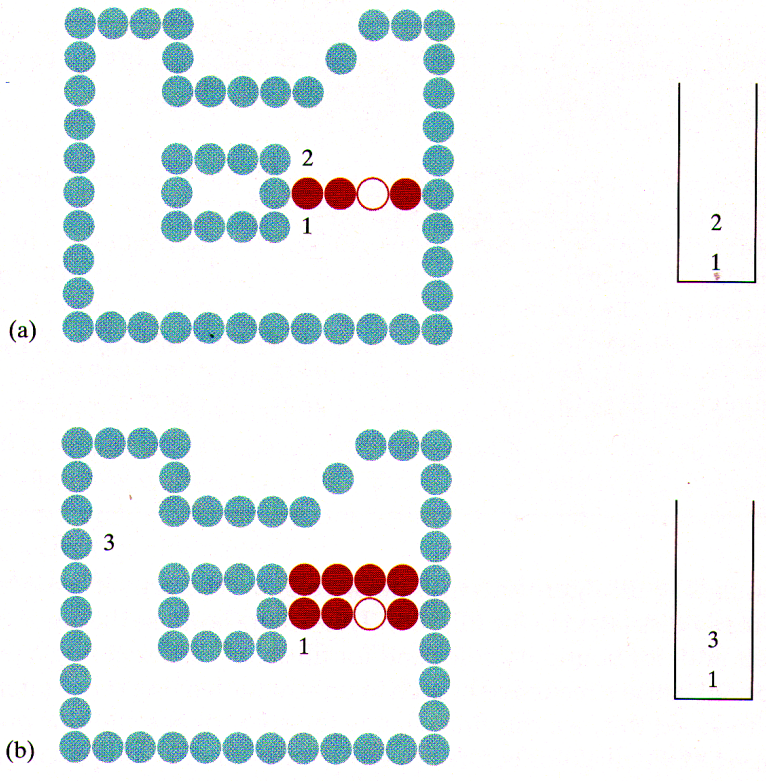
```
1  void fill(int x, int y, int fillcolor, int bordercolor) {
2     int intcolor;
3     getPixel(x,y,intcolor);
4     if((intcolor != bordercolor) && (intcolor != fillcolor)) {
5        setpixel(x,y, fillcolor);
6        fill(x+1,y,fillcolor,bordercolor);
7        fill(x-1,y,fillcolor,bordercolor);
8        fill(x,y+1,fillcolor,bordercolor);
9        fill(x,y-1,fillcolor,bordercolor);
10    }
11 }
```

Algoritmo de Preenchimento de Borda

- **Problemas** se algum **pixel interior** já for da **cor escolhida** para ser preenchida
  - Algum ramo da recursão pode ser descartado

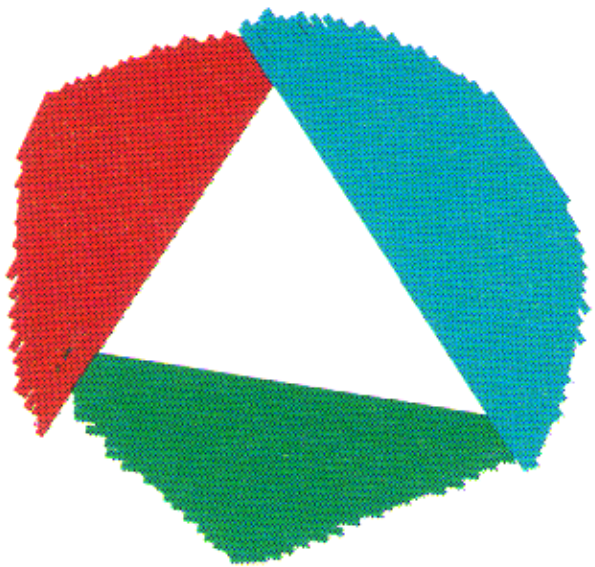
- A recursão pode levar a um consumo excessivo de memória
  - Solução é empilhar, ao invés de pixels vizinhos, blocos de pixels sucessivos (o pixel inicial desses blocos)

Algoritmo de Preenchimento de Borda



Algoritmo Flood-Fill

- As vezes é necessário colorir uma área que **não é definida** apenas por uma **cor de borda**
  - Ao invés de procurar uma cor de borda, procurar por uma **cor de interior**
  - Se o interior tem mais de uma cor, pode-se inicialmente substituir essa cor para que todos pixels do interior tenham a mesma cor



Algoritmo de Preenchimento de Borda

```
1  void fill(int x, int y, int fillcolor, int interiorcolor) {
2     int color;
3     getPixel(x,y,color);
4     if(color == interiorcolor) {
5        setpixel(x,y, fillcolor);
6        fill(x+1,y,fillcolor, interiorcolor);
7        fill(x-1,y,fillcolor, interiorcolor);
8        fill(x,y+1,fillcolor, interiorcolor);
9        fill(x,y-1,fillcolor, interiorcolor);
10    }
11 }
```

Observação

- No caso de preenchimento de áreas irregulares, cada pixel está sendo "pintado" com uma cor

- No caso de rendering de superfícies, cada pixel é pintado com a cor determinada pela aplicação do algoritmo de iluminação + tonalização (shading)