

## Computação Gráfica

Agostinho Brito

Departamento de Engenharia da Computação e Automação  
Universidade Federal do Rio Grande do Norte

22 de março de 2005

É aplicada em:

- Interfaces de usuário;
- Traçado de gráficos (interativos);
- Automação de escritório;
- CAD;
- Simulação de sistemas;
- Animação;
- Arte e comércio; etc.

O que será estudado

- OpenGL;
- Dispositivos de exibição;
- Algoritmos de rastreamento;
- Algoritmos de preenchimento;
- Recortes;
- Transformações geométricas 2D e 3D;
- Projeções em perspectiva;
- Modelagem geométrica;

- Representação de curvas no plano e no espaço;
- Tratamento de linhas e superfícies escondidas;
- Rendering;
- Modelos de iluminação;
- Modelos de cor;
- Tratamento de sombras;
- Ray Tracing/Radiância;
- Textura;

Foram os primeiros dispositivos gráficos de exibição. Tais dispositivos apresentavam as seguintes características:

- Uma tela de fósforo era sensibilizada por um feixe de luz;
- Linhas podiam ser traçadas de qualquer ponto para qualquer ponto na tela;
- O tempo de traçado dos desenhos dependia velocidade de comunicação entre o computador e o dispositivo gráfico e do número de objetos a serem desenhados;
- Ausência de cor;
- Traçado de objetos tridimensionais era muito custoso.

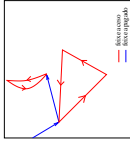


Figura: Dispositivo de exibição vetorial

## Implementação de um dispositivo raster

A implementação de um dispositivo raster em um tubo de raios catódicos - CRT - pode ser feito com o uso de *frame buffers*, obedecendo às seguintes etapas:

- Armazenar numa matriz os pontos a serem desenhados;
- Ler a informação digital em cada elemento da matriz e converter para uma voltagem elétrica com um DAC (conversor digital-analógico).
- Sensibilizar a tela gráfica nas coordenadas correspondentes às da matriz;

- Dispositivos raster são como matriz de células discretas que podem ser acesas ou apagadas. As linhas desenhadas aparecem serrilhadas, semelhantes a escadas. A este efeito é dado o nome de *aliasing*.
- O uso de dispositivos de rastreamento (*raster graphics*) permite que o tempo de desenho da imagem na tela seja independente do número de objetos desenhados.

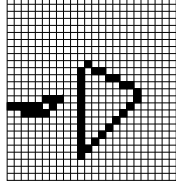


Figura: Dispositivo de exibição por rastreamento

## Funcionamento de um CRT

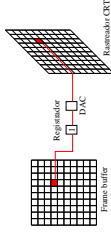
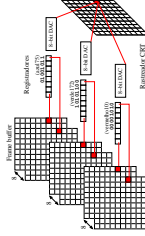


Figura: CRT monocrômico (preto e branco).



- Locator:** provê informações de coordenadas em 2 ou 3 dimensões.
- Valuator:** provê um valor simples, geralmente apresentado como um número real.
- Button:** utilizado para selecionar e ativar eventos ou procedimentos.
- Pick:** identifica ou seleciona objetos na tela.
- Keyboard:** coleção de botões.

- Tablet:** consiste em superfície plana e uma caneta, usada para apontar uma posição na superfície do *tablet*. Também chamado mesa digitalizadora.
- Touch panel:** semelhante ao *tablet*, atua como um localor, onde o dispositivo apontador pode ser, por exemplo, um dedo.
- Mouse:** é dotado de uma bola interna que atua sobre dois *valuators*, indicando posição. Botões adicionais servem para realizar *choice* ou *pick* de entidades na tela.
- Joystick:** semelhante ao mouse, mas com uma origem fixa.
- Trackball:** semelhante ao mouse. Utilizados quando o espaço físico é reduzido para a aplicação.
- Outros:** *Spaceball*, *data glove*, caneta ótica.

## Rasterização

- Sendo a tela gráfica uma matriz de pontos, é impossível traçar uma linha direta de um ponto a outro. Sendo assim, alguns pontos da tela deverão ser selecionados para representar o objeto que se deseja desenhar.
- O processo utilizado na determinação dos pixels que melhor aproximam um determinado objeto é denominado **rasterização** (*rastering*).

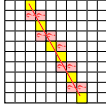
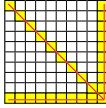


Figura: Rasterização de linhas retas.

## DDA - Analizador diferencial digital

- A equação da linha direta entre dois pontos  $(x_1, y_1)$  e  $(x_2, y_2)$  é dada pela equação (1)
 
$$y_{i+1} = y_i + \frac{y_2 - y_1}{x_2 - x_1} \Delta x,$$

onde:  $\Delta x = x_2 - x_1$  e  $\Delta y = y_2 - y_1$ .

- Para implementar um DDA simples, o maior dos valores de  $\Delta x$  ou  $\Delta y$  é escolhido como unidade de rasterização. O algoritmo DDA funciona nos quatro quadrantes.

```

if abs(x2 - x1) ≥ abs(y2 - y1) then
    Tamanho = abs(x2 - x1)
else
    Tamanho = abs(y2 - y1)
end if
(Seleciona o maior dos valores entre Δx e Δy como unidade rasterização)
Δx = (x2 - x1) / Tamanho
Δy = (y2 - y1) / Tamanho
i = 1
while i ≤ Tamanho do
    desenhaPonto(Floor(x), Floor(y)) (Floor: valor arredondado de um dado número real. Inteiro(-8.6) = -9; Inteiro(-8.4) = -8)
    x = x + Δx
    y = y + Δy
    i = i + 1
end while
    
```

**Limitações práticas**

- Utiliza aritmética de ponto flutuante;
- Se a função Floor for substituída por uma função inteira verdadeira, os resultados serão diferentes;

i	desenhaPonto	x	y
1	(0,0,0,0)	0.0	0.0
2	(-1,0,-0,0)	-1.0	-0.4
3	(-2,0,-1,0)	-2.0	-0.8
4	(-3,0,-1,0)	-3.0	-1.2
5	(-4,0,-2,0)	-4.0	-1.6

Tabela: Funcionamento do DDA

## Algoritmo de Bresenham para traçado de linhas

- Para cada ponto a ser traçado, o algoritmo verifica sua a distância entre a posição do ponto e a localização do grid. Apenas o sinal do erro é analisado. A base do algoritmo é mostrada na figura abaixo.

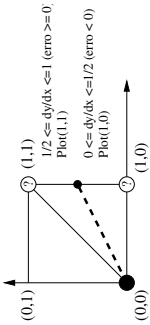


Figura: Base do algoritmo de Bresenham para linhas

## Algoritmo real de Bresenham para retas

- O erro é iniciado com valor igual a  $-1/2$ . A cada iteração,  $e = e + \Delta y / \Delta x$ . Quando o erro assume um valor positivo, é necessário reinicializá-lo, subtraindo  $1^{\text{a}}$  do seu valor.

```

x = x1
y = y1
Δx = x2 - x1
Δy = y2 - y1
m = Δy / Δx
e = m - 1/2
for i = 1 to Δx do
    desenhaPonto(x,y)
    while e ≥ 0 do
        y = y + 1
        e = e - 1
    end while
    x = x + 1
    e = e + m
end for
    
```

• O algoritmo de bresenham pode ser melhorado se a divisão por  $\Delta x$  for eliminada, passando a utilizar somente aritmética inteira. O novo erro será agora:

$$\bar{e} = 2e\Delta x$$

• As modificações são apresentadas no algoritmo inteiro de Bresenham para retas.

```

...
 $\bar{e} = 2\Delta y - \Delta x$ 
for  $i = 1$  to  $\Delta x$  do
  desenhaPonto( $x, y$ )
  while  $\bar{e} \geq 0$  do
     $y = y + 1$ 
     $\bar{e} = \bar{e} - 2\Delta x$ 
  end while
   $x = x + \Delta x$ 
   $\bar{e} = \bar{e} + 2\Delta y$ 
end for

```

```

 $x = x_1$ 
 $y = y_1$ 
 $\Delta x = \text{abs}(x_2 - x_1)$ 
 $\Delta y = \text{abs}(y_2 - y_1)$ 
 $s1 = \text{Sgn}(\Delta x)$ 
 $s2 = \text{Sgn}(\Delta y)$ 
if  $\Delta y > \Delta x$  then
  Temp =  $\Delta x$ 
   $\Delta x = \Delta y$ 
   $\Delta y = \text{Temp}$ 
  Troca = 1
else
  Troca = 0
end if
 $\bar{e} = 2\Delta y - \Delta x$ 
for  $i = 1$  to  $\Delta x$  do
  desenhaPonto( $x, y$ )
  while  $\bar{e} \geq 0$  do
    if Troca = 1 then
       $x = x + s1$ 
    end if
     $\bar{e} = \bar{e} - 2\Delta x$ 
  end while
  if Troca = 0 then
     $y = y + s2$ 
  end if
   $\bar{e} = \bar{e} + 2\Delta y$ 
end for

```

## Algoritmo de Bresenham para traçado de circunferências

- A geração dos pontos é feita apenas para o segundo octante da circunferência e replicados para os demais octantes.
- O pixel selecionado na figura foi previamente escolhido como o mais adequado. O próximo ponto a ser selecionado para o traçado será o que mais se aproximar da circunferência.

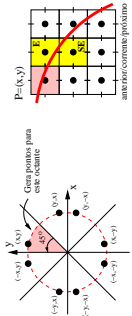


Figura: Base do algoritmo de Bresenham para circunferências

## Determinação do ponto seguinte no traçado da circunferência

- Seja  $F(x, y) = x^2 + y^2 - R^2$ ,  $F(x, y)$  vale zero, positivo ou negativo, caso o ponto  $(x, y)$  esteja sobre, fora ou dentro da circunferência.
- Seja  $d$  a variável de decisão, o valor da função  $F(x, y)$  no ponto central entre os dois pixels.

$$d_{\text{velho}} = F(x_p + 1, y_p - 1/2) = (x_p + 1)^2 + (y_p - 1/2)^2 - R^2$$

- Se  $d_{\text{velho}} < 0$ ,  $E$  é escolhido. Logo:

$$d_{\text{novo}} = F(x_p + 2, y_p - 1/2) = (x_p + 2)^2 + (y_p - 1/2)^2 - R^2$$

$$d_{\text{novo}} = d_{\text{velho}} + (2x_p + 3)$$

- Se  $d_{\text{velho}} \geq 0$ ,  $SE$  é escolhido e o novo valor de  $d$  será:

$$d_{\text{novo}} = F(x_p + 2, y_p - 3/2) = (x_p + 2)^2 + (y_p - 3/2)^2 - R^2$$

$$d_{\text{novo}} = d_{\text{velho}} + (2x_p - 2y_p + 5)$$

- O primeiro ponto da circunferência é  $(0, R)$ .
  - O próximo ponto central cai em  $(1, R - 1/2)$ , logo  $d = 5/4 - R$ .
  - Como  $d$  é incrementado com valores inteiros, a mudança  $d \rightarrow d + 1 - R$  não altera no processo de desenho.
  - **pontosDaCircunferência()**: replica os pontos no segundo octante para os octantes restantes.
- ```

x = 0
y = R
PontosDaCircunferencia(x,y)
while y > x do
  if d < 0 then
    d = d + 2 * x + 3
    x = x + 1
  else
    d = d + 2 * (x - y) + 5
    x = x + 1
    y = y - 1
  end if
  PontosDaCircunferencia(x,y)
end while
    
```

- Servem para definir o conjunto de pixels que será desenhando dentro de um determinado contorno fechado. Este contorno geralmente pode ser representado na forma poligonal.
- A triagem dos pixels normalmente é feita dentro de uma região limitante, denominada *bounding box*, como mostra a figura.

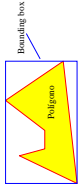


Figura: bounding box de um polígono

## Conversão de varredura

- Exceto nas bordas, pixels adjacentes em um polígono possuem as mesmas características. Esta propriedade é chamada coerência espacial. Assim, os pixels de uma dada linha (*scan line*) variam somente nas bordas do polígono.
- O processo de determinar quais pixels serão desenhados no preenchimento é chamado conversão de varredura (*scan conversion*), mostrado na figura 9.

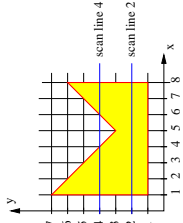


Figura: conversão de varredura para um polígono fechado

## Conversão de varredura

A *scan line* 4, por exemplo, pode ser dividida nas seguintes regiões:

| Intervalo         | Situação           |
|-------------------|--------------------|
| $x < 1$           | fora do polígono   |
| $1 \leq x \leq 4$ | dentro do polígono |
| $4 < x \leq 6$    | fora do polígono   |
| $6 \leq x \leq 8$ | dentro do polígono |
| $x > 8$           | fora do polígono   |

- A determinação dos pontos de interseção não é feita necessariamente da esquerda para a direita. Caso o polígono seja definido pela lista de vértices  $P_1, P_2, P_3, P_4$ , a sequência das interseções será 8, 6, 4, 1. É necessário então ordenar a lista obtida, ou seja, 1, 4, 6, 8.
- As interseções podem ser consideradas em pares. Pixels contidos no intervalo formado por estes pares são desenhados na cor do polígono.

Considere o traçado do retângulo definido pelas coordenadas (1,1), (5,1), (5,4), (1,4). O resultado do preenchimento utilizando este algoritmo é mostrado na figura.

### Ativação de pixels

**Problema:** a área do retângulo  $A = (5 - 1) \times (4 - 1) = 12$ , mas 20 pixels são ativados!  
**Solução:** realizar o teste na *scanline*  $y + 0,5$ . O resultado é mostrado na figura de baixo.

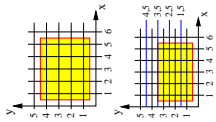


Figura: scanlines y e y + 1/2

### Exemplo de algoritmo

Para o polígono da última figura, de vértices  $P_1(1,1)$ ,  $P_2(8,1)$ ,  $P_3(8,6)$ ,  $P_4(5,3)$ ,  $P_5(1,7)$ , os dados obtidos para cada scanline são mostrados na tabela 2.

| scanline | interseções encontradas                |
|----------|----------------------------------------|
| 1,5      | (8,1,5), (1,1,5)                       |
| 2,5      | (8,2,5), (1,2,5)                       |
| 3,5      | (8,3,5), (5,5,3,5), (4,5,3,5), (1,3,5) |
| 4,5      | (8,4,5), (6,5,4,5), (3,5,4,5), (1,4,5) |
| 5,5      | (8,5,5), (7,5,5,5), (2,5,5,5), (1,5,5) |
| 6,5      | (1,5,6,5), (1,6,5)                     |
| 7,5      | nenhuma                                |

Tabela: determinação de interseções para o algoritmo *ordered edge list*

### Exemplo de algoritmo

- Quando ordenadas pelo algoritmo, as interseções com as  $y + 1/2$  scanlines formarão a seguinte lista:  
 $(1,6,5), (1,5,6,5), (1,5,5), (2,5,5,5), \dots, (1,1,5), (8,1,5)$
- Extraindo os pares de interseções desta lista e aplicando o processo de seleção de pontos descritos no algoritmo, será gerada a seguinte lista de pontos para ativação:  
 $(1,6), (1,5), (2,5), \dots, (1,1), (2,1), (3,1), (4,1), (5,1), (6,1), (7,1)$
- O resultado do processo de preenchimento é mostrado na figura.

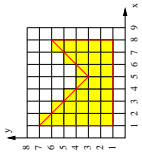


Figura: preenchimento de um polígono pelo algoritmo *ordered edge list*

O algoritmo anterior pode ser melhorado se o processo de ordenação for mais eficiente. Ao invés de ordenar toda a lista de uma só vez, para cada *scanline*, as coordenadas *x* da interseção são armazenadas em uma célula (*y bucket*) correspondente à *scanline*, como mostrado na figura. Assim, a ordenação é feita apenas dentro de cada *scanline*.

|   |     |     |     |   |   |   |   |
|---|-----|-----|-----|---|---|---|---|
| 8 | x   | x   | x   | x | x | x | x |
| 7 | x   | x   | x   | x | x | x | x |
| 6 | 1,5 | 1   | x   | x | x | x | x |
| 5 | 8   | 7,5 | 2,5 | 1 | x | x | x |
| 4 | 8   | 6,5 | 3,5 | 1 | x | x | x |
| 3 | 8   | 5,5 | 4,5 | 1 | x | x | x |
| 2 | 8   | 1   | x   | x | x | x | x |
| 1 | 8   | 1   | x   | x | x | x | x |
| 0 | x   | x   | x   | x | x | x | x |

(a)

|   |   |     |     |   |   |   |   |
|---|---|-----|-----|---|---|---|---|
| 8 | x | x   | x   | x | x | x | x |
| 7 | x | x   | x   | x | x | x | x |
| 6 | 1 | 1,5 | x   | x | x | x | x |
| 5 | 1 | 2,5 | 7,5 | 8 | x | x | x |
| 4 | 1 | 3,5 | 6,5 | 8 | x | x | x |
| 3 | 1 | 4,5 | 5,5 | 8 | x | x | x |
| 2 | 1 | 8   | x   | x | x | x | x |
| 1 | 1 | 8   | x   | x | x | x | x |
| 0 | x | x   | x   | x | x | x | x |

(b)

Figura: ybuckets para as scanlines do polígono da figura 9.

Exemplo do algoritmo

O resultado da aplicação deste algoritmo no preenchimento do polígono usado anteriormente é mostrado na figura abaixo.

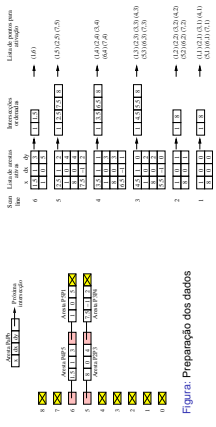


Figura: Preparação dos dados

Figura: Conversão dos dados

O novo algoritmo ainda necessita de muita memória alocada para armazenar as listas de interseções. Utilizar uma lista encadeada! Uma lista de fronteiras ativas indica para o algoritmo as arestas presentes (ativas) em cada *scanline*. (Preparação dos dados)

Determine para cada aresta as interseções com as  $(y + 1/2)$  *scanlines*, via Bresenham, as maiores *scanlines* interceptadas pela aresta.

Armazene a aresta do polígono no *y bucket* da *scanline* correspondente.

Armazene a interseção inicial *x*, o número de *scanlines* interceptadas pela aresta,  $\Delta y$ , e o incremento de *x*,  $\Delta x$ , de *scanline* para *scanline* em uma lista encadeada. (Conversão dos dados)

Para cada *scanline*, verifique o aparecimento de novas arestas nos *y buckets* correspondentes, e adicione a aresta à lista de arestas ativas.

Ordene as interseções da lista de arestas ativas na ordem crescente, ou seja,  $x_1$  precede  $x_2$  se  $x_1 \leq x_2$ .

Extraia os pares de elementos da lista,  $(x_1, y_1)$  e  $(x_2, y_2)$ .

Ative os pixels da *scanline* para valores inteiros de *x* tais que  $x_1 \leq x + 1/2 \leq x_2$ .

Para cada aresta na lista de arestas ativas, decemente  $\Delta y$  por 1. Se  $\Delta y < 0$ , remova aresta dessa lista.

Calcule a nova interseção *x* para cada elemento da lista de arestas ativas,  $x_{novo} = x_{velho} + \Delta x$ .

Preenchimento baseado em semente

- Algoritmos de preenchimento baseado em semente, ou *seed fill algorithms*, assumem que pelo menos um ponto no interior do polígono é conhecido. O algoritmo tenta encontrar o restante dos pontos no interior e preenchê-los com uma determinada cor.
- Neste caso, uma informação adicional é requerida: o tipo de conectividade da região. As regiões podem ser 4-conectadas ou 8-conectadas.
- Para uma região 4-conectada, todos os pixels no seu interior podem ser alcançados com combinações dos movimentos **leste**, **oeste**, **norte** e **sul**. Para uma região 8-conectada, os pontos no interior podem ser alcançados com combinações dos movimentos **leste**, **oeste**, **norte**, **sul**, **noroeste**, **sudeste** e **sudoeste**, como mostrado na figura.
- As regiões 4-conectadas são delimitadas por fronteiras 8-conectadas. As regiões 8-conectadas são delimitadas por fronteiras 4-conectadas.

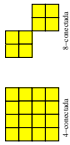


Figura: Tipos de conectividade em uma região



```
(Seed(x,y) é o pixel semente)
(Push/Pop: coloca/retira o pixel em uma pilha)
```

```
Pixel(x,y)= Seed(x,y)
```

```
Push Pixel(x,y)
```

```
while pilha não vazia do
```

```
  Pop Pixel(x,y)
```

```
  if Pixel(x+1,y) ≠ New value then
```

```
    Pixel(x,y) = New value
```

```
  end if
```

```
  if Pixel(x+1,y) ≠ New value and Pixel(x+1,y) ≠ Boundary value then
```

```
    Push Pixel(x+1,y)
```

```
  end if
```

```
  if Pixel(x,y+1) ≠ New value and Pixel(x,y+1) ≠ Boundary value then
```

```
    Push Pixel(x,y+1)
```

```
  end if
```

```
  if Pixel(x-1,y) ≠ New value and Pixel(x-1,y) ≠ Boundary value then
```

```
    Push Pixel(x-1,y)
```

```
  end if
```

```
  if Pixel(x,y-1) ≠ New value and Pixel(x,y-1) ≠ Boundary value then
```

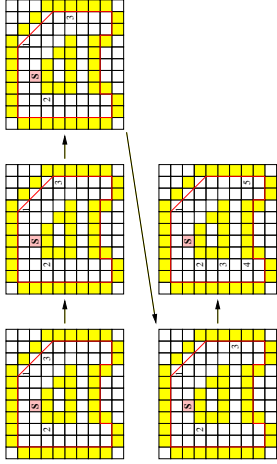
```
    Push Pixel(x,y-1)
```

```
  end if
```

```
end while
```

## Funcionamento do algoritmo scanline seed fill

Os números mostrados dentro dos pixels representam a posição da semente na pilha de sementes.



Embora simples, o algoritmo anterior consome muita memória com o uso de pilhas. Além disso, a pilha pode conter frequentemente informação duplicada. O algoritmo *scanline seed fill* contorna este problema semeando apenas um pixel nos trechos de uma *scanline* a ser preenchida. Algoritmo *seed fill* para regiões 4-conectadas:

```
while pilha não vazia do
```

```
  Retire um pixel semente de um trecho de uma pilha contendo a semente.
```

```
  Preencha os trechos à esquerda e à direita da semente, até que uma fronteira seja encontrada.
```

```
  Grave as coordenadas da extrema esquerda (Xleft) e da extrema direita (Xright) do trecho preenchido.
```

```
  Na taxa  $Xleft \leq x \leq Xright$ , para as scanlines imediatamente superior e imediatamente inferior, verifique se existem apenas pixels de fronteiras ou previamente preenchidos. Se estas scanlines não contêm apenas pixels de fronteiras ou previamente preenchidos, marque com uma semente o pixel da extrema direita de cada um dos trechos encontrados na faixa  $Xleft \leq x \leq Xright$ .
```

```
end while
```