Conversão Matricial

SCC0250 - Computação Gráfica

Prof. Fernando V. Paulovich
http://www.icmc.usp.br/~paulovic
paulovic@icmc.usp.br

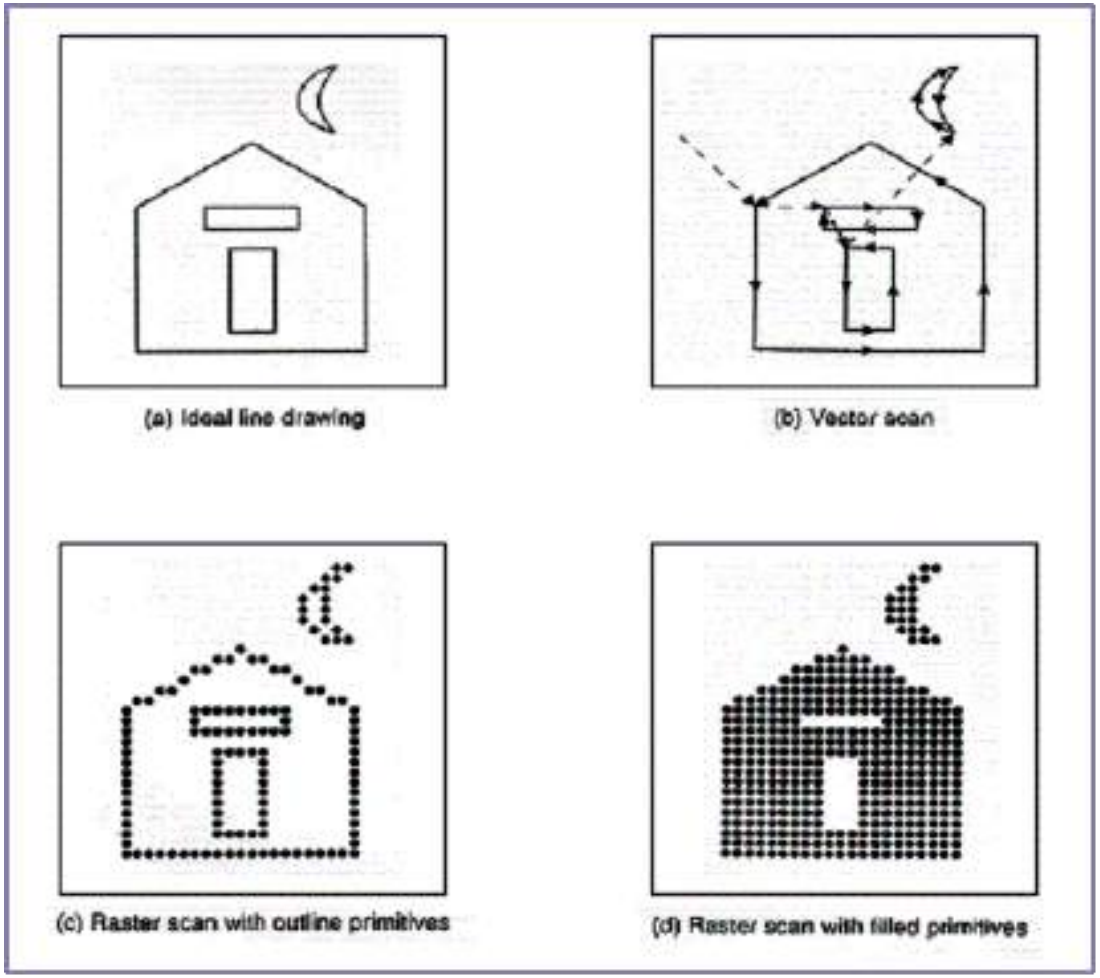
Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)

25 de abril de 2013

Sumário

Sumário

Imagem Vetorial x Imagem Matricial



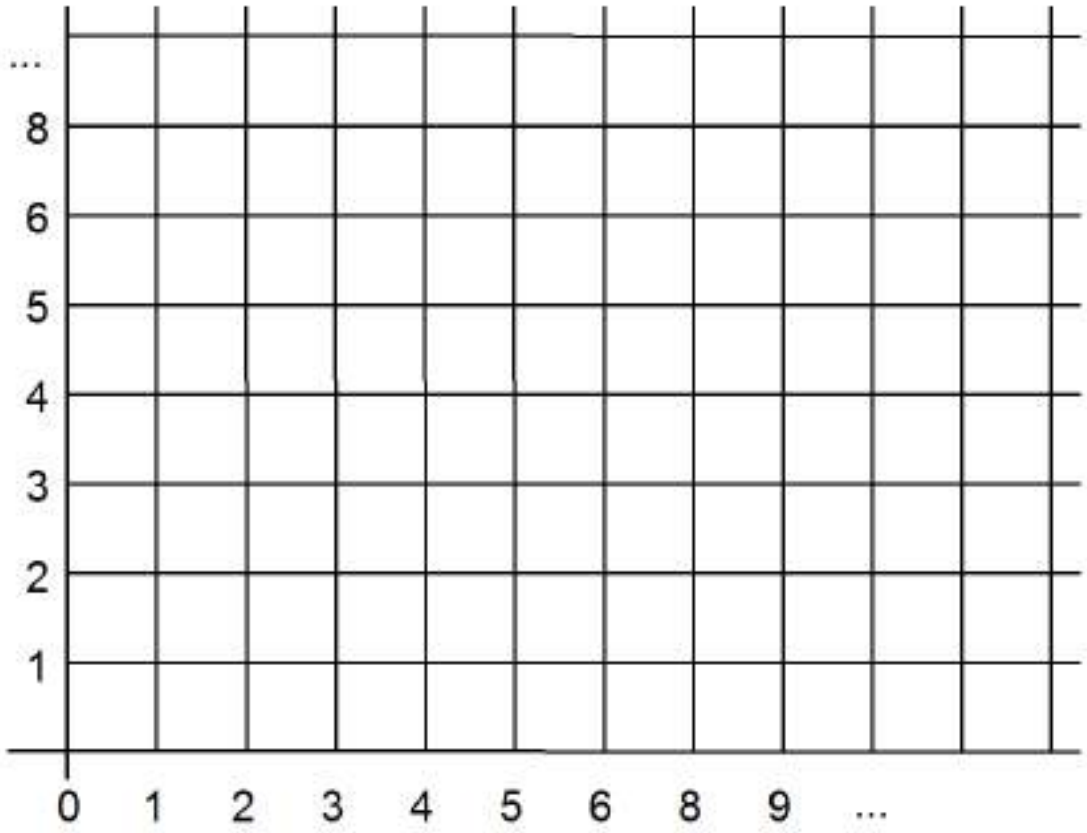
Problema

- Traçar primitivas geométricas (segmentos de reta, polígonos, circunferências, elipses, curvas, ...) no dispositivo matricial

- "rastering" = conversão vetorial $\rightarrow$ matricial

- Como ajustar uma curva, definida por coordenadas reais em um sistema de coordenadas inteiras cujos "pontos" tem área associada

Sistema de Coordenadas do Dispositivo



Sumário

Conversão de Segmentos de Reta

- Dados pontos extremos em coordenadas do dispositivo
  - $P_0(x_0, y_0)$ (inferior esquerdo)
  - $P_{end}(x_{end}, y_{end})$ (superior direito)

- Determina quais pixels devem ser "acesos" para gerar na tela uma boa aproximação do segmento de reta ideal

Conversão de Segmentos de Reta

- Características Desejáveis
  - Linearidade
  - Precisão
  - Espessura (Densidade Uniforme)
  - Intensidade independente de inclinação
  - Continuidade
  - Rapidez no traçado

Conversão de Segmentos de Reta

- Usar equação explícita da reta

$$y = m \cdot x + b$$

- Com m a inclinação da reta dada por

$$m = \frac{y_{end} - y_0}{x_{end} - x_0}$$

- Com b a intersecção do eixo y dada por

$$b = y_0 - m \cdot x_0$$

Conversão de Segmentos de Reta

Algoritmo Simples

- Variando-se x unitariamente de pixel em pixel, encontramos o valor de y

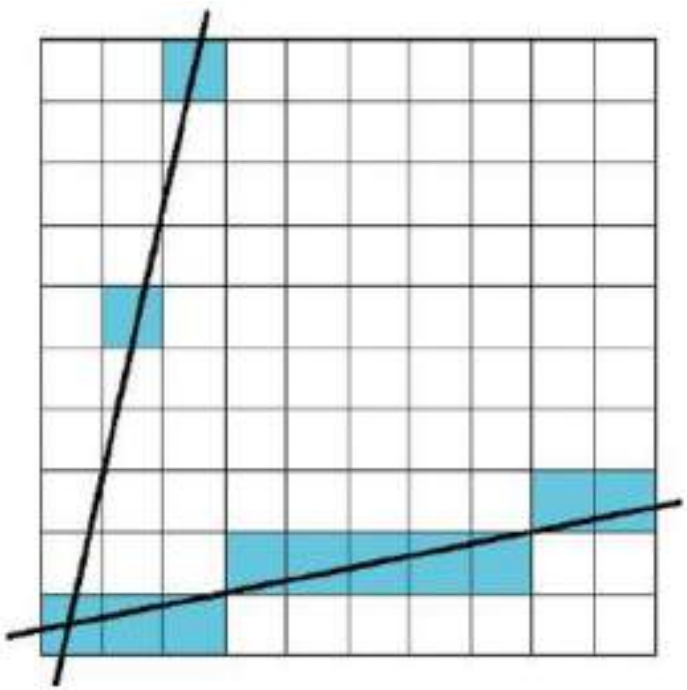
```
1  {
2    int x, x0, xend, y0, yend;
3    float y, m;
4    int valor; //cor do pixel
5
6    m = (yend - y0)/(xend - x0);
7    for (x = x0; x <= xend; x++) {
8      y = y0 + m * (x - x0);
9      write_pixel (x, round(y), valor); //arredonda y
10   }
11 }
```

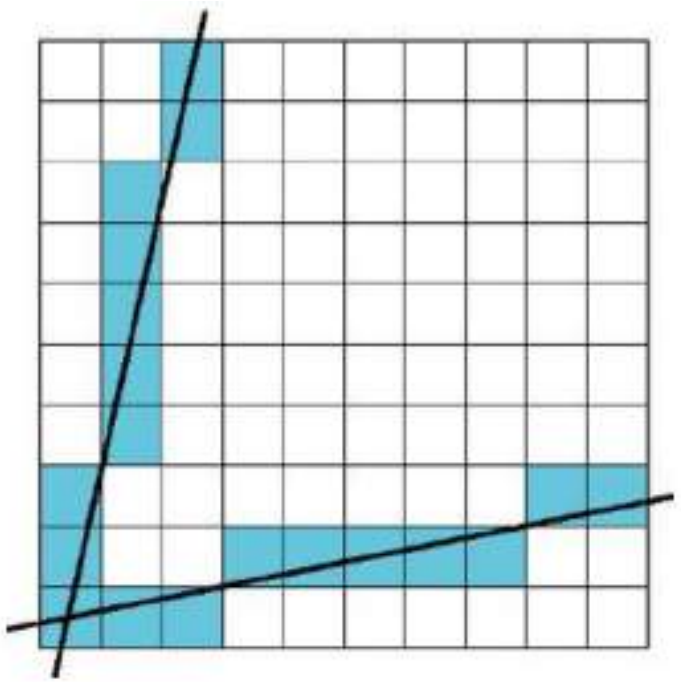
Problema

- Na forma dada, só funciona para segmentos em que $0 < m < 1$. Porque?

Problema

- Se $0 < m < 1$ a variação em x é superior à variação em y. Se esse não for o caso, vai traçar um segmento com buracos!!





Solução

- Se $m > 1$, basta inverter os papéis de x e y, i.e., amostra y a intervalos unitários, e calcula x

$$x = x_0 + \frac{y - y_0}{m}$$

Sumário

Algoritmo DDA

- Chamando de δx uma variação na direção de x, podemos encontrar a variação δy em y correspondente fazendo

$$\delta y = m \cdot \delta x$$

- ou similarmente

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

- O algoritmo *Digital Differential Analyzer (DDA)* se baseia no cálculo de δx ou δy

Algoritmo DDA

- Para $|m| \leq 1$, na iteração i temos

$$y_i = m \cdot x_i + b$$

- Sendo δ_x a variação na direção de x, na iteração $i+1$ temos

$$\begin{aligned} y_{i+1} &= m \cdot x_{i+1} + b \\ y_{i+1} &= m \cdot (x_i + \delta x) + b \\ y_{i+1} &= m \cdot x_i + m \cdot \delta x + b \\ y_{i+1} &= (m \cdot x_i + b) + m \cdot \delta x \\ y_{i+1} &= y_i + m \cdot \delta x \end{aligned}$$

- Se $\delta x = 1$, então

$$\begin{aligned} x_{i+1} &= x_i + 1, \text{ e} \\ y_{i+1} &= y_i + m \end{aligned}$$

Algoritmo DDA

- Se $|m| > 1$, inverte-se os papéis de x e y, isto é, $\delta y = 1$ e calcula-se x

$$
\begin{aligned}
x_i &= \frac{y_i - b}{m} \\
x_{i+1} &= \frac{y_{i+1} - b}{m} \\
x_{i+1} &= \frac{y_i + \delta y - b}{m} \\
x_{i+1} &= \frac{y_i - b}{m} + \frac{\delta y}{m} \\
x_{i+1} &= x_i + \frac{\delta y}{m}
\end{aligned}
$$

- Se $\delta y = 1$, então

$$
\begin{aligned}
y_{i+1} &= y_i + 1, \text{ e} \\
x_{i+1} &= x_i + \frac{1}{m}
\end{aligned}
$$

Algoritmo DDA

- Assume $x_0 < x_{end}$ e $y_0 < y_{end}$ (m positivo), processamento da esquerda para a direita

- Se não é o caso, então $\delta x = -1$ ou $\delta y = -1$, e a equação de traçado deve ser adaptada de acordo
  - Exercício: fazer a adaptação em cada caso

Exercício

- Aplique o algoritmo (e adaptações) para fazer a conversão dos seguintes segmentos de reta
  - P1: (0,1) P2: (5,3)
  - P1: (1,1) P2: (3,5)

Sumário

Sumário

Introdução

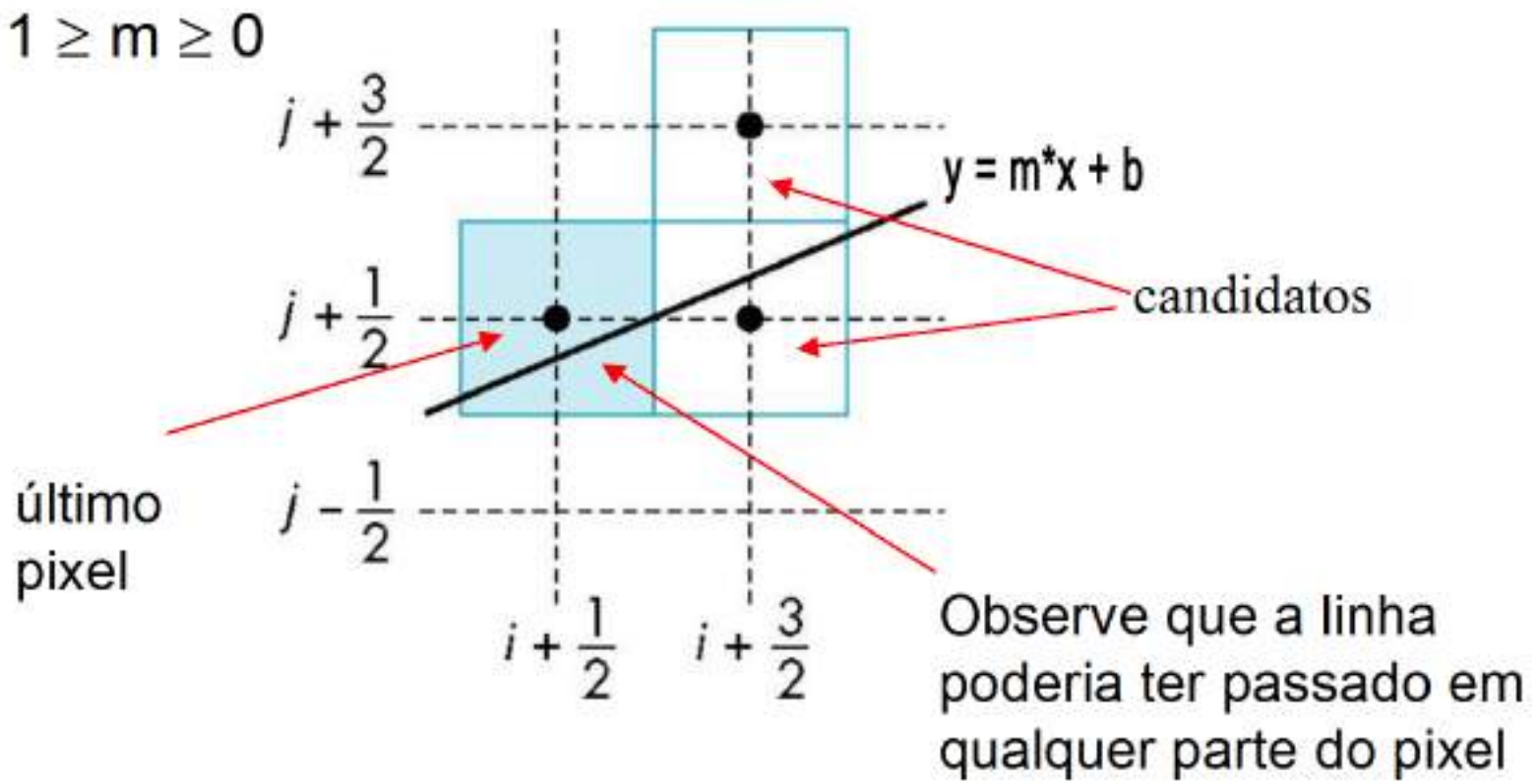
- **Algoritmo DDA** apesar de ser incremental, envolve cálculo com números flutuantes (cálculo de m): **ineficiente**

- O algoritmo de **Bresenham** trabalha somente com inteiros: **muito mais eficiente**

Algoritmo de Bresenham (Retas)

- Assume $0 < |m| < 1$
- Incrementa x em intervalos unitários, calcula o y correspondente
- Abordagem considera as duas possibilidades de escolha de y, decidindo qual a melhor
  - $(x_k, y_k) \rightarrow (x_k + 1, y_k)$
  - $(x_k, y_k) \rightarrow (x_k + 1, y_k + 1)$

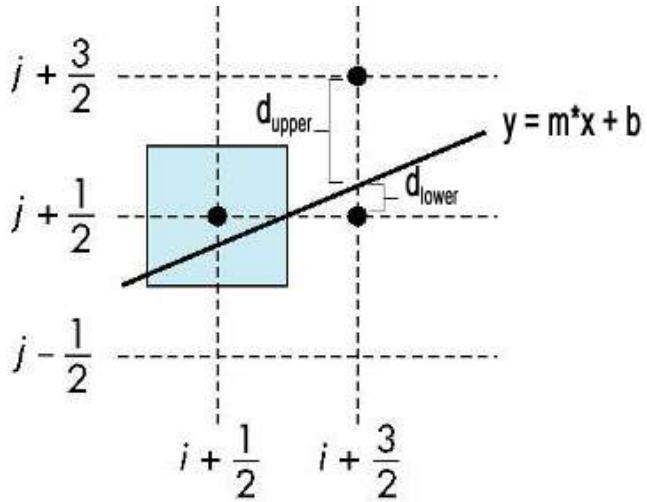
Algoritmo de Bresenham (Retas)



Algoritmo de Bresenham (Retas)

- $(d_{lower} - d_{upper}) \geq 0$
  usar o pixel superior

- $(d_{lower} - d_{upper}) < 0$
  usar o pixel inferior



Algoritmo de Bresenham (Retas)

- Com base na equação da reta $(y = m \cdot x + b)$, na posição $x_k + 1$, a coordenada y é calculada como

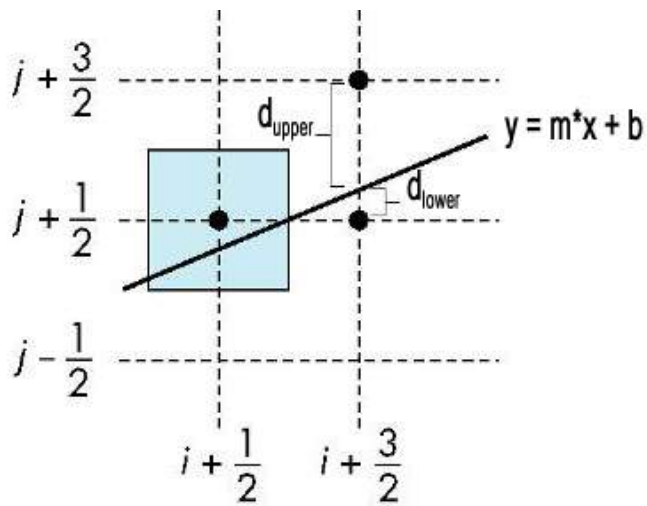
$$y = m \cdot (x_k + 1) + b$$

- Então

$$d_{lower} = y - y_k$$
$$d_{lower} = m \cdot (x_k + 1) + b - y_k$$

- e

$$d_{upper} = (y_k + 1) - y$$
$$d_{upper} = y_k + 1 - m \cdot (x_k + 1) - b$$



Algoritmo de Bresenham (Retas)

- Um teste rápido para saber a proximidade

$$p_k = d_{lower} - d_{upper}$$
$$p_k = 2m(x_k + 1) - 2y_k + 2b - 1$$

- Assim
  - $p_k > 0$: pixel superior
  - $p_k < 0$: pixel inferior

Algoritmo de Bresenham (Retas)

- Mas calcular m envolve operações de ponto flutuante

$$m = \frac{y_{end} - y_0}{x_{end} - x_0} = \frac{\Delta y}{\Delta x}$$

- então, substituindo m por $\frac{\Delta y}{\Delta x}$, e multiplicando tudo por Δx, temos

$$p_k = \Delta x(d_{lower} - d_{upper})$$

- como $\Delta x > 0$, o sinal de p_k não é alterado, então

$$p_k = 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c$$

- com $c = 2\Delta y + \Delta x(2b - 1)$, parâmetro constante independente da posição do pixel

Algoritmo de Bresenham (Retas)

- No passo $k+1$ temos

$$p_{k+1} = 2\Delta y \cdot x_{k+1} - 2\Delta x \cdot y_{k+1} + c$$

- subtraindo p_k dos dois lados temos

$$\begin{aligned} p_{k+1} - p_k = & \quad (2\Delta y \cdot x_{k+1} - 2\Delta x \cdot y_{k+1} + c) - p_k \\ p_{k+1} - p_k = & \quad 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k) \end{aligned}$$

- mas $x_{k+1} = x_k + 1$ (incremento unitário em x), então

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k)$$

Algoritmo de Bresenham (Retas)

- Nessa equação

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k)$$

- $y_{k+1} - y_k$ será 0 ou 1 dependendo do sinal de p_k

- Se $p_k < 0$, então o próximo ponto é $(x_k + 1, y_k)$ então

$$y_{k+1} - y_k = 0 \text{ e}$$
$$p_{k+1} = p_k + 2\Delta y$$

- Caso contrário o ponto será $(x_k + 1, y_k + 1)$ então

$$y_{k+1} - y_k = 1 \text{ e}$$
$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

Algoritmo de Bresenham (Retas)

- Esse cálculo iterativo é realizado para cada posição x começando da esquerda para a direita

- O ponto de partida é calculado como

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

Algoritmo de Bresenham (Retas)

```
1  void bresenham (int x1,int x2, int y1,int y2)
2  int dx,dy, incSup, incInf, p, x, y;
3  int valor;
4  {
5    dx = x2-x1; dy = y2-y1;
6    p = 2*dy-dx; /* fator de decisão: valor inicial */
7
8    incInf = 2*dy; /* Incremento Superior */
9    incSup = 2*(dy-dx); /* Incremento inferior */
10
11   x = x1; y = y1;
12   write_Pixel (x,y,valor); /* Pinta pixel inicial */
13
14   while (x < x2) {
15     if (p <= 0) { /* Escolhe Inferior */
16       p = p + incInf;}
17     else { /* Escolhe Superior */
18       p = p + incSup;
19       y++;} /* maior que 45o */
20     x++;
21     write_pixel (x, y, valor);
22   } /* fim do while */
23 } /* fim do algoritmo */
```

Algoritmo de Bresenham (Retas)

- Exercício: aplique o algoritmo para
  - P1: (5,8)
  - P2: (9,11)

Sumário

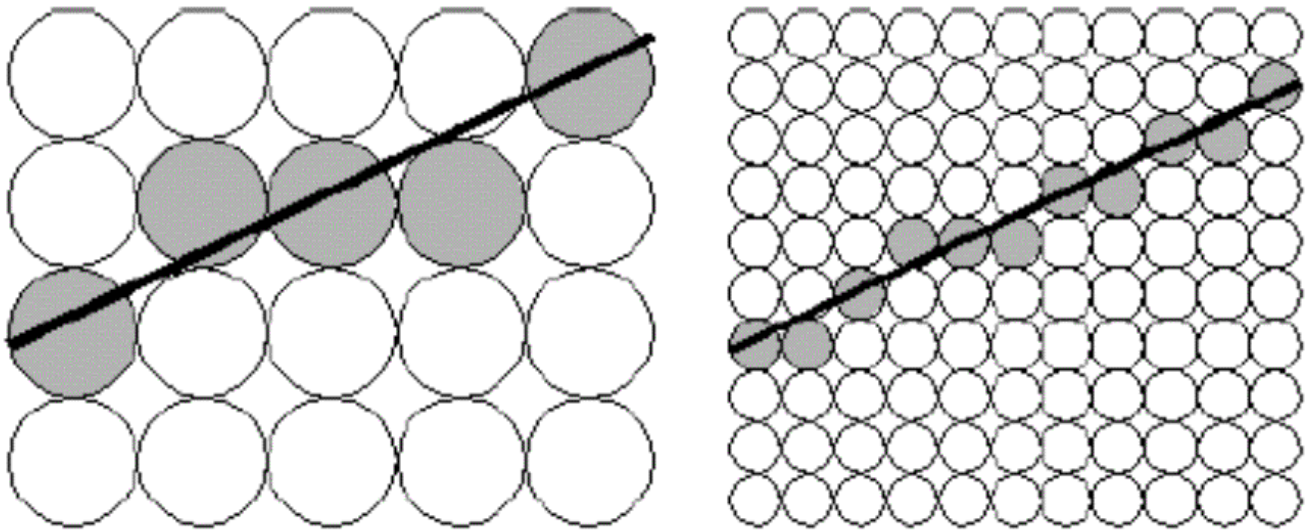
Correção de Traçado (Antialiasing)

- Segmentos de retas em sistemas *raster* tem espessura - ocupa uma área

- Devido ao processo de amostragem (discretização), segmentos de retas pode apresentar uma aparência serrilhada

- Uma forma de diminuir esse problema é usar monitores com pixels menores
  - Problemas para manter a taxa de restauro em 60Hz

Correção de Traçado (Antialiasing)



(a) (b)

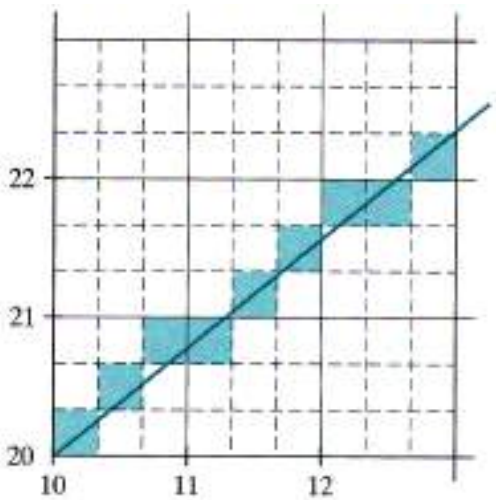
Correção de Traçado (Antialiasing)

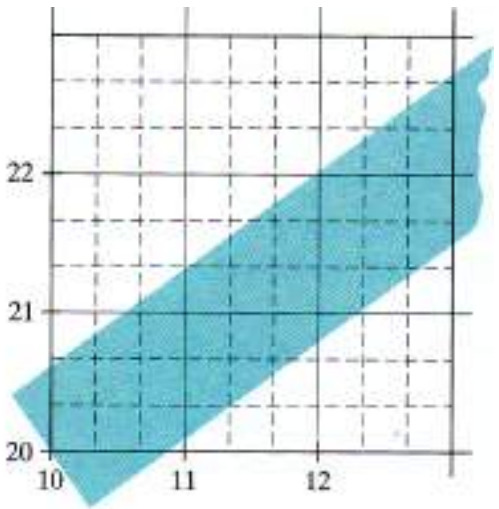
- Em sistemas que podem mostrar mais de dois níveis de intensidade, é possível usar uma solução de software

- Uma solução simples é conhecida como **superamostragem**
  - Aumenta a taxa de amostragem simulando um monitor com um (sub)pixel de menor tamanho
  - A intensidade do pixel real é definida com base na quantidade de subpixels cobertos

Superamostragem

- Dividir cada pixel em sub-pixels
  - A intensidade é dada pelo número de sub-pixel que estão sob o caminho da linha





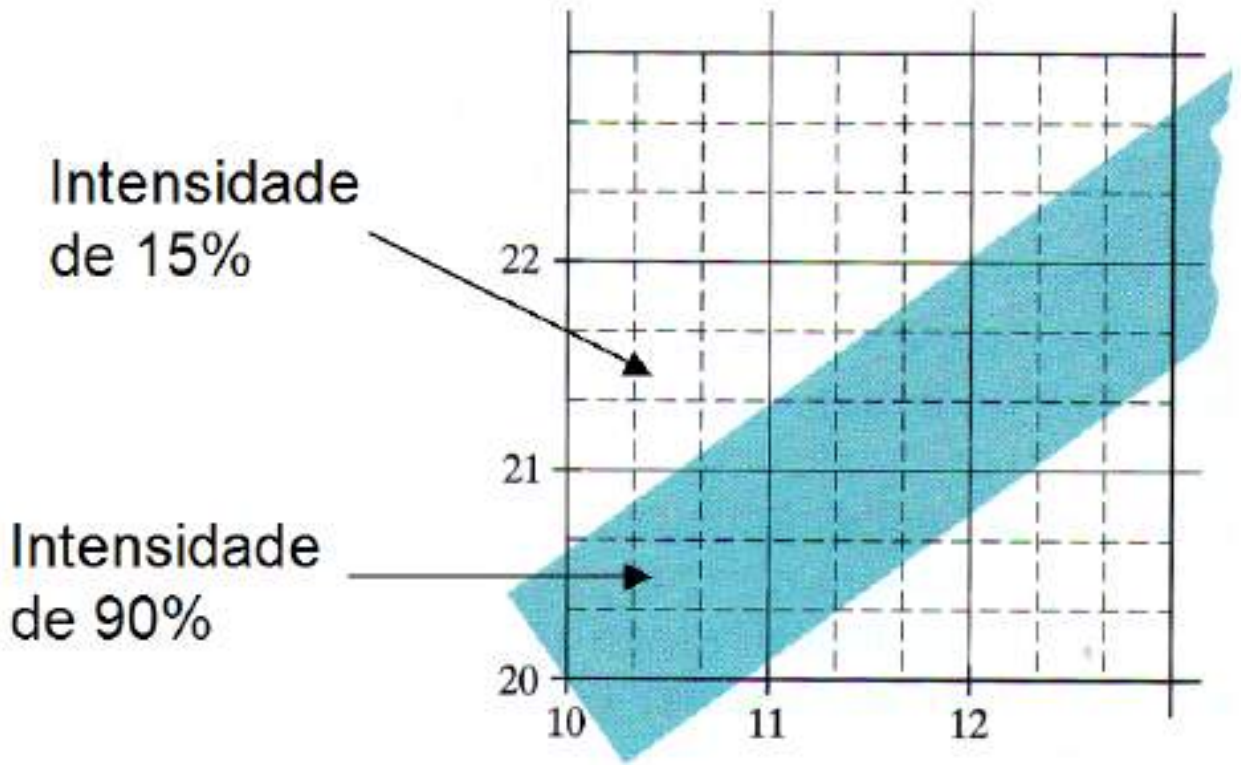
Máscara de Ponderação do Sub-Pixel

- Define uma máscara que assinala maior peso (intensidade) para sub-pixels no centro da área do pixel

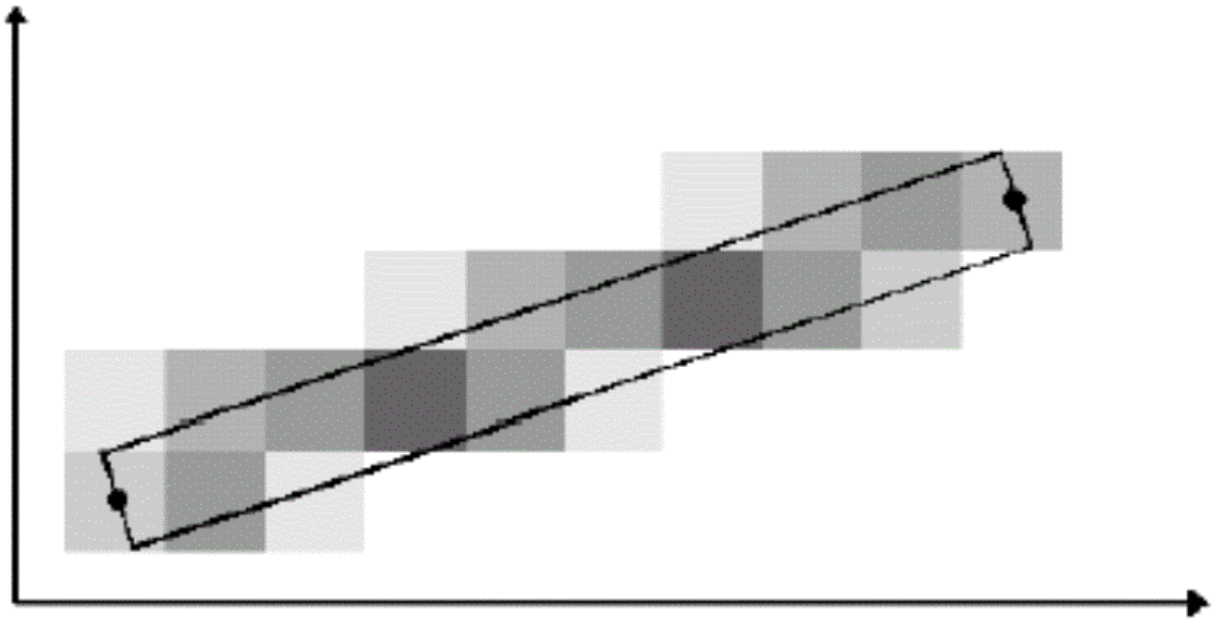
1	2	1
2	4	2
1	2	1

Antialiasing Baseado em Área

- Intensidade proporcional a área coberta do pixel considerando que a linha tem largura finita

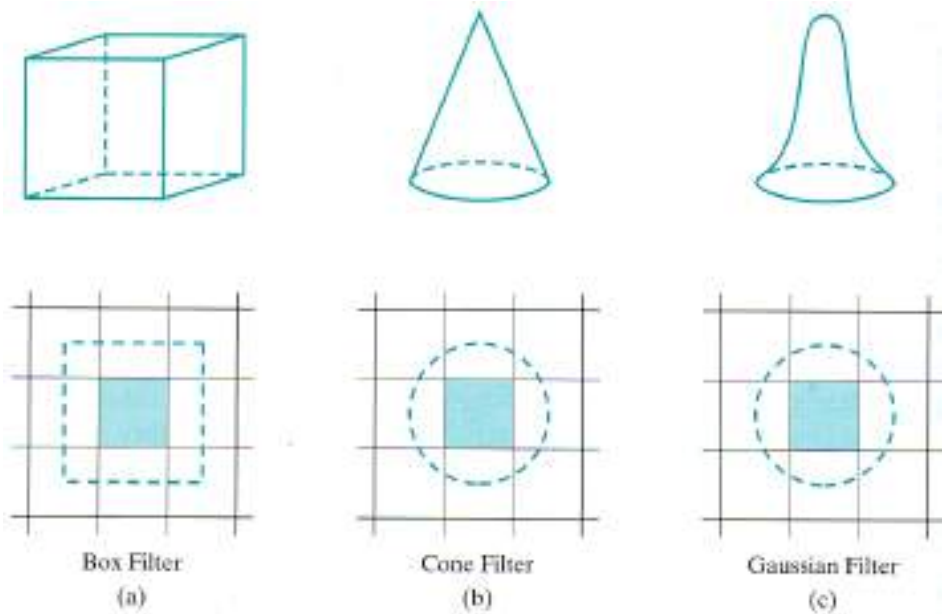


Antialiasing Baseado em Área



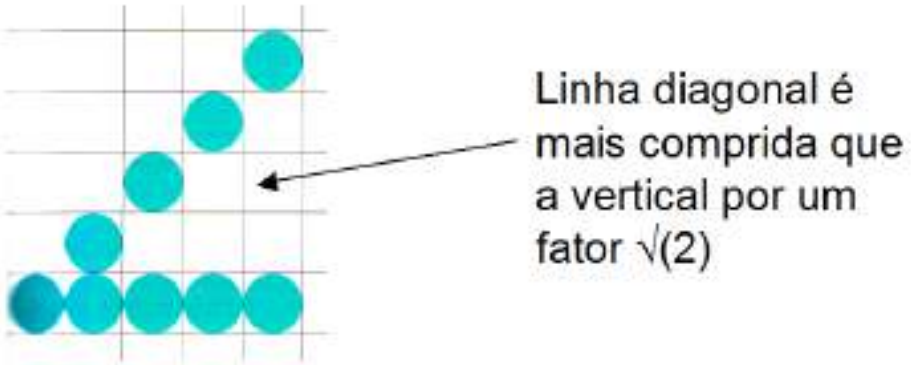
Técnicas de Filtragem

- Similar a técnica de máscara, porém mais precisa
  - Uma superfície contínua de ponderação é usada para determinar a cobertura do pixel
  - A ponderação é calculada por integração – Uso de tabelas para acelerar o processo



Compensando Diferenças de Intensidade

- Antialiasing pode compensar outro problema de sistemas raster
  - Linhas desenhadas com mesma quantidade de pixels apresentarem tamanhos diferentes – Linhas menores mais brilhantes



Sumário

Antialiasing e OpenGL

- A função de antialiasing em OpenGL é ativada usando

```
1 gl.glEnable(tipoprim)
```

- Tipos de primitivas
  - *GL.GL_POINT_SMOOTH*
  - *GL.GL_LINE_SMOOTH*
  - *GL.GL_POLYGON_SMOOTH*

- Também necessário ativar *blending* em RGBA (RGB)

```
1 gl.glEnable(GL.GL_BLEND);
2 gl.glBlendFunc(GL.GL_SRC_ALPHA, GL.GL_ONE_MINUS_SRC_ALPHA);
```

Antialiasing e OpenGL

```
1  public class Antialiasing implements GLEventListener {
2    ...
3
4    public static void main(String[] args) {
5      //cria o painel e adiciona um ouvinte GL
6      GLJPanel panel = new GLJPanel();
7      panel.addGLEventListener(new Antialiasing());
8
9      //cria uma janela e adiciona o painel
10     JFrame frame = new JFrame("Aplicao JOGL Simples");
11     frame.add(panel);
12     frame.setSize(400, 400);
13     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14     frame.setLocationRelativeTo(null);
15     frame.setVisible(true);
16   }
17   public void display(GLAutoDrawable drawable) {
18     GL gl = drawable.getGL();
19     gl.glClear(GL.GL_COLOR_BUFFER_BIT); //desenha o fundo (limpa a janela)
20     gl.glColor3f(1.0f, 0.0f, 0.0f); //altera o atributo de cor
21
22     gl.glBegin(GL.GL_LINES); //desenha uma linha
23     gl.glVertex2i(10, 10);
24     gl.glVertex2i(190, 170);
25     gl.glEnd();
26
27     gl.glFlush(); //processa as rotinas OpenGL o mais rápido possível
28   }
29 }
```

Antialiasing e OpenGL

```
1  public class Antialiasing implements GLEventListener {
2    ...
3
4    public void init(GLAutoDrawable drawable) {
5      GL gl = drawable.getGL();
6      gl.glClearColor(1.0f, 1.0f, 1.0f, 1.0f); //define cor de fundo
7      gl.glMatrixMode(GL.GL_PROJECTION); //carrega a matriz de projeção
8      gl.glLoadIdentity(); //carrega a matriz identidade
9
10     //ativa antialiasing para linha
11     gl.glEnable(GL.GL_LINE_SMOOTH);
12     gl.glEnable(GL.GL_BLEND);
13     gl.glBlendFunc(GL.GL_SRC_ALPHA, GL.GL_ONE_MINUS_SRC_ALPHA);
14
15     GLU glu = new GLU();
16     glu.gluOrtho2D(0.0, 200.0, 0.0, 200.0); //define projeção ortogonal 2D
17   }
18 }
```

Antialiasing e OpenGL