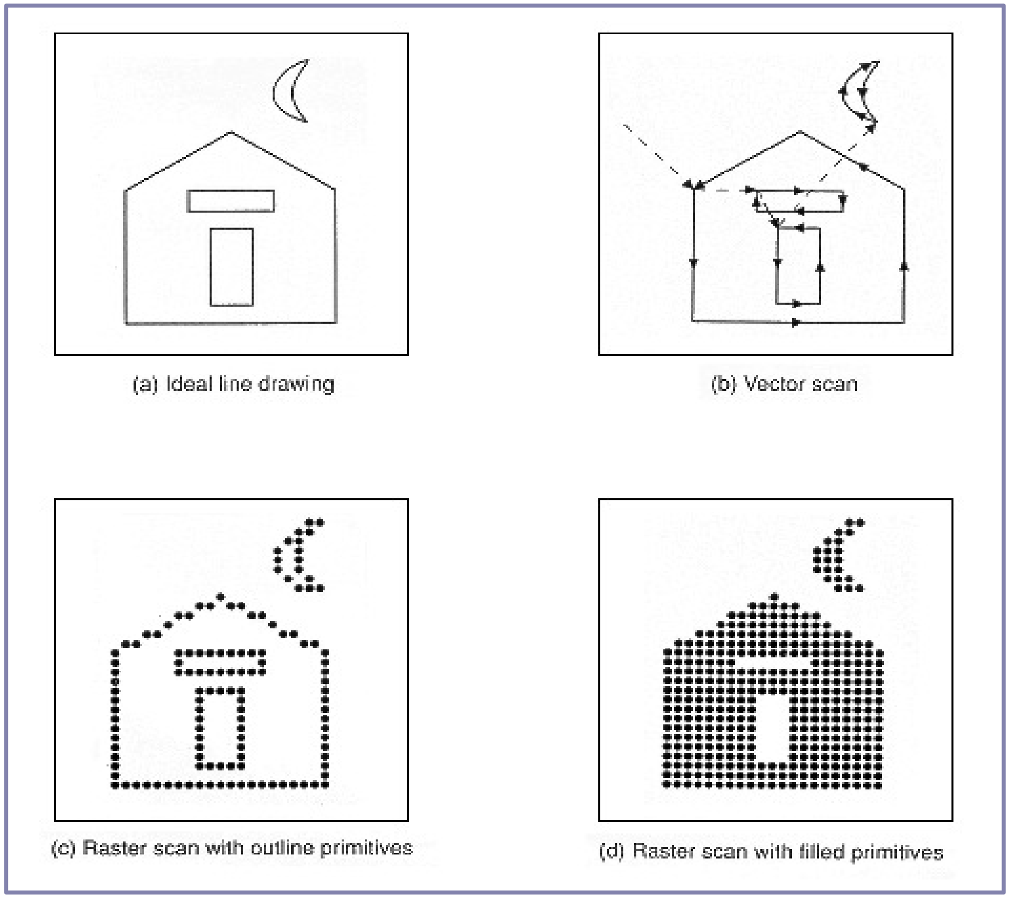
Conversão Matricial de Primitivas Gráficas

Maria Cristina F. de Oliveira
Fernando V. Paulovich

Imagem Vetorial x Imagem Matricial



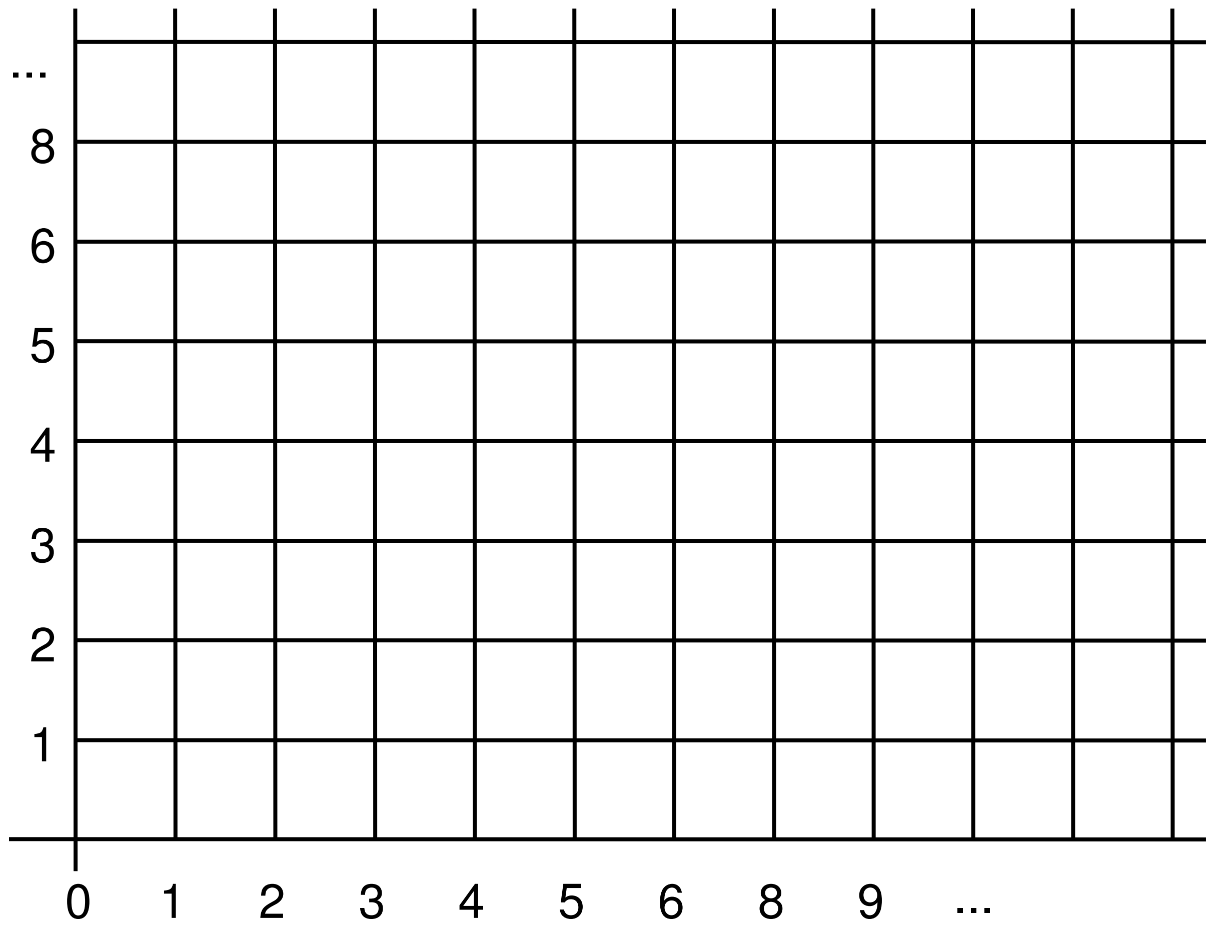
Problema

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

- ‘rastering’ = conversão vetorial -> 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



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

- Dados pontos extremos em coordenadas do dispositivo:
  - P1(x_1,y_1) (inferior esquerdo)
  - P2(x_2,y_2) (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 – Algoritmo DDA

- Usar equação explícita da reta

$y = mx + B$

$m = (y_2 – y_1)/(x_2 – x_1)$ //inclinação

$B = y_1 – m^*x_1$ //intersecção eixo y

$y = mx + (y_1 – m^*x_1) = y_1 + m^*(x-x_1)$

Algoritmo DDA

```
{ int x, x1, x2, y1, y2;
  float y, m;
  int valor;

  m = (y2-y1)/(x2-x1);                    /* 0 < |m| < 1
  for (x = x1; x <= x2; x++)  {
     /* arredonda y */
     y= y1+ m*(x-x1);
     write_pixel (x, round(y), valor);
}
```

- Cálculo em ponto flutuante: ineficiente

Otimização DDA

Na iteração i: $y_i = m * x_i + B$

Na iteração i+1:

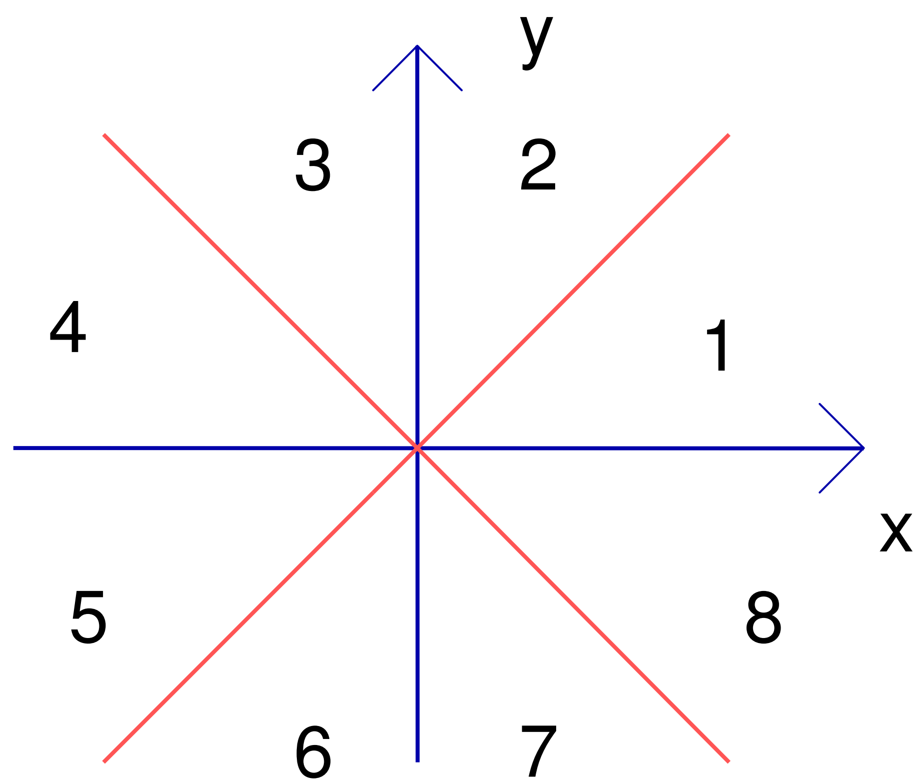
$$y_{i+1} = m * x_{i+1} + B = m * (x_i + \delta x) + B$$
$$= m * x_i + m * \delta x + B = y_i + m * \delta x$$

se $\delta x = 1$, então

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

Algoritmo incremental!!

Octantes do Sistema de Coordenadas Euclidiano



Algoritmo DDA

Na forma dada, funciona para segmentos em que $0 < m < 1$

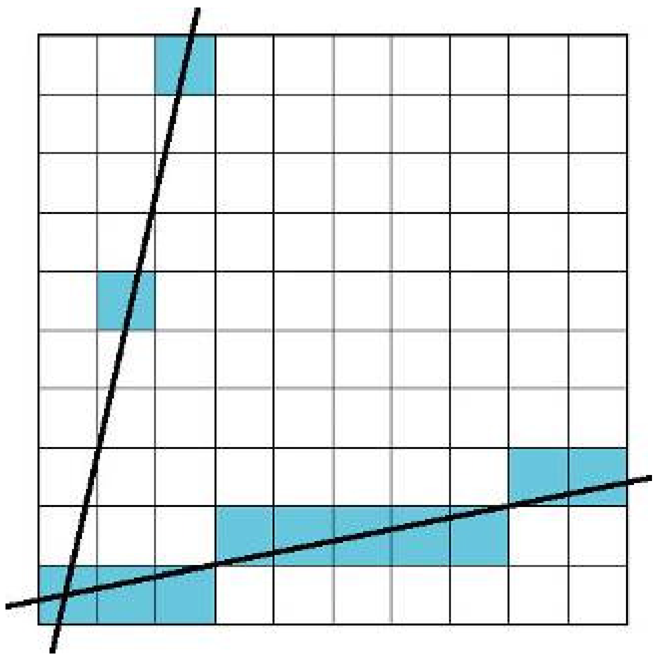
Porque?

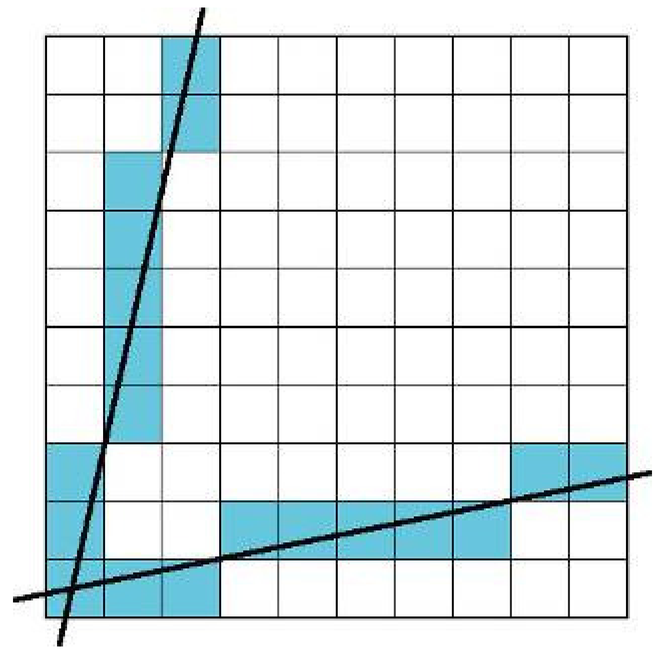
Algoritmo DDA

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

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

Algoritmo DDA





Algoritmo DDA

- Assume $x_1 < x_2$ e $y_1 < y_2$ (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

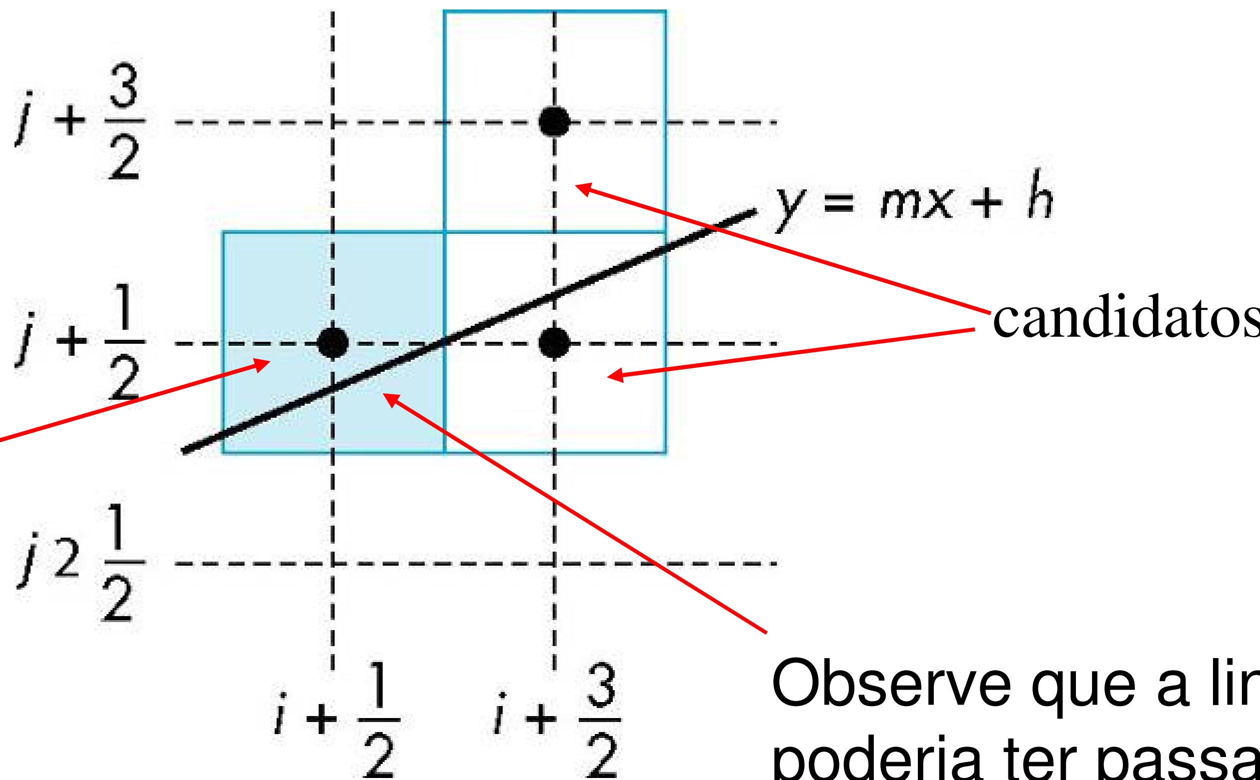
- Aplique o algoritmo (e adaptações) para fazer a conversão dos seguintes segmentos de reta:
  - P1: (0,1) P2: (5,3)
  - P1: (1,1) P2: (3,5)

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
  - $(x_k, y_k) \rightarrow (x_k+1, y_k)$ ou (x_k+1, y_k+1)

Algoritmo de Bresenham (Retas)

$1 \geq m \geq 0$

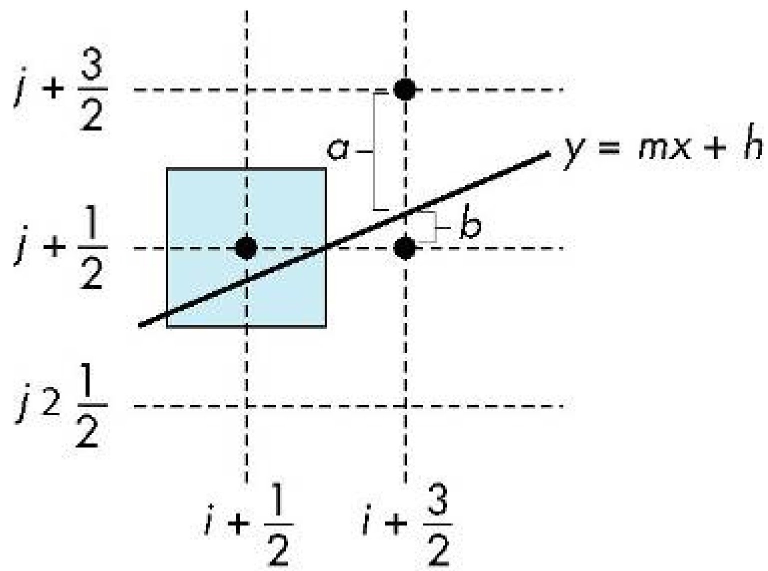


último pixel

Observe que a linha poderia ter passado em qualquer parte do pixel

Algoritmo de Bresenham (Retas)

(b-a) > 0 usar o pixel superior
(b-a) < 0 usar o pixel inferior



Algoritmo de Bresenham (Retas)

- Na posição $x_k + 1$, a coordenada y é calculada como
  - $y = m(x_k + 1) + B$
- Então
  - $b = y - y_k = m(x_k+1) + B - y_k$
- e
  - $a = (y_k + 1) - y = y_k+1 - m(x_k + 1) - B$

Algoritmo de Bresenham (Retas)

- Um teste rápido para saber a proximidade
  - $p_k = b - a = 2m(x_k+1) - 2y_k + 2B - 1$
- Assim
  - $p_k > 0$: pixel superior
  - $p_k < 0$: pixel inferior

Algoritmo de Bresenham (Retas)

- Mas calcular *m* envolve operações de ponto flutuante,
  - $m = (y_2 – y_1)/(x_2 – x_1) = \Delta y / \Delta x$
- então fazemos
  - $\Delta x p_k = \Delta x(b - a) \rightarrow p_k = \Delta x(b - a)$
  - $p_k = 2\Delta y * x_k - 2\Delta x * y_k + c$
  - constante $c = 2\Delta y + \Delta x(2B - 1)$

Algoritmo de Bresenham (Retas)

- Algoritmo incremental
  - $p_{k+1} = 2\Delta y * x_{k+1} - 2\Delta x * y_{k+1} + x$
- Subtraindo p_k de p_{k+1} temos
  - $p_{k+1} – p_k = 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k)$
- Como $x_{k+1} = x_k + 1$
  - $p_{k+1} = p_k + 2\Delta y - 2\Delta x\ (y_{k+1} - y_k)$
  - onde $y_{k+1} – y_k$ é 0 ou 1 dependendo do sinal de p_k

Algoritmo de Bresenham (Retas)

- Se $p_K < 0$, então o próximo ponto é (x_{k+1}, y_k) e
  - $p_{k+1} = p_k + 2\Delta y$
- Caso contrário o ponto será (x_{k+1}, y_{k+1}) e
  - $p_{k+1} = p_k + 2\Delta y - 2\Delta x$

Algoritmo de Bresenham (Retas)

```
void bresenham (int x1,int x2, int y1,int y2)
int dx,dy, incSup, incInf, p, x, y;
int valor;
{
     dx= x2-x1;  dy= y2-y1;
     p = 2*dy-dx;    /* fator de decisão: valor inicial */

     incInf = 2*dy;                  /* Incremento Superior */
     incSup =2*(dy-dx);        /* Incremento inferior */

     x= x1;  y= y1;
     write_Pixel (x,y,valor);   /* Pinta pixel inicial */
```

Algoritmo de Bresenham (Retas)

```
    while (x < x2) {
        if (p <= 0) {    /* Escolhe Inferior */
                p = p + incInf;        }
        else {           /* Escolhe Superior */
                p = p + incSup;
                y++;}    /* maior que 45° */
        x++;
        write_pixel (x, y, valor);
    }  /* fim do while */
} /* fim do algoritmo */
```

Algoritmo de Bresenham (Retas)

Exercício: aplique o algoritmo para

- P1: (5,8) P2: (9,11)

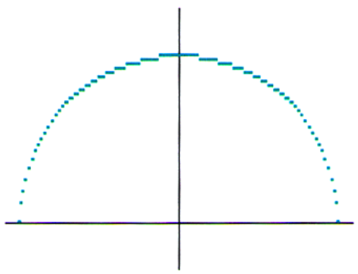
Traçado de Circunferências

- Circunf. com centro na origem e raio R:

  $x^2 + y^2 = R^2 \rightarrow y = \pm sqrt(R^2 - x^2)$ //forma explícita

  $x = R*\cos\theta, y = R*sen\theta$ //forma paramétrica

  - Partindo de P1: (0,R), porque não usar diretamente a equação explícita acima para traçar um arco de ¼ da circunf.?



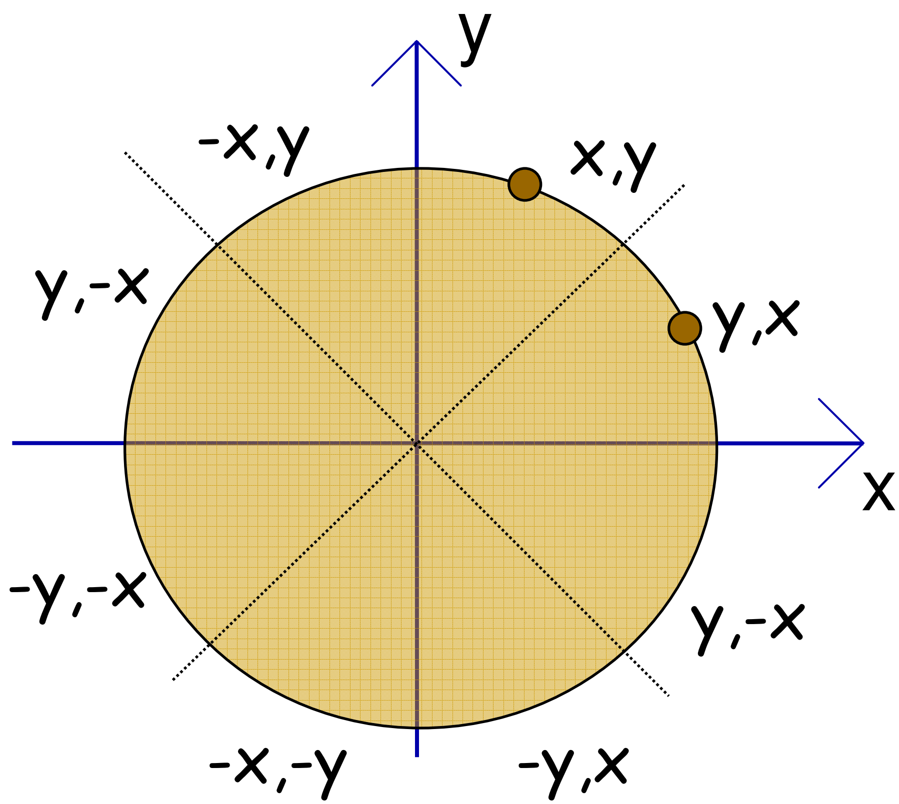
  - Porque não usar a forma paramétrica?

Algoritmo do Ponto Médio (Circunf.)

- Traçado de arco de 45º no segundo octante, de x = 0 a x = y = R/sqrt(2)

- O restante da curva pode ser obtido por simetria
  - Se o ponto (x,y) pertence à circunferência, outros 7 pontos sobre ela podem ser obtidos de maneira trivial...

Simetria de Ordem 8



Simetria de Ordem 8

```
void CirclePoints (int x, int y, int value)
{
    write_pixel( x,y,value);  write_pixel( x,-y,value);
    write_pixel(-x,y,value); write_pixel(-x,-y,value);
    write_pixel( y,x,value);  write_pixel( y,-x,value);
    write_pixel(-y,x,value); write_pixel(-y,-x,value);
}
```

Algoritmo de Bresenham (Circunf.)

- Define um parâmetro de decisão para definir o pixel mais próximo da circunferência

- Como a equação da circunf. é não linear, raízes quadradas serão necessárias para se calcular distâncias dos pixels
  - Bresenham evita isso comparando o quadrado das distâncias

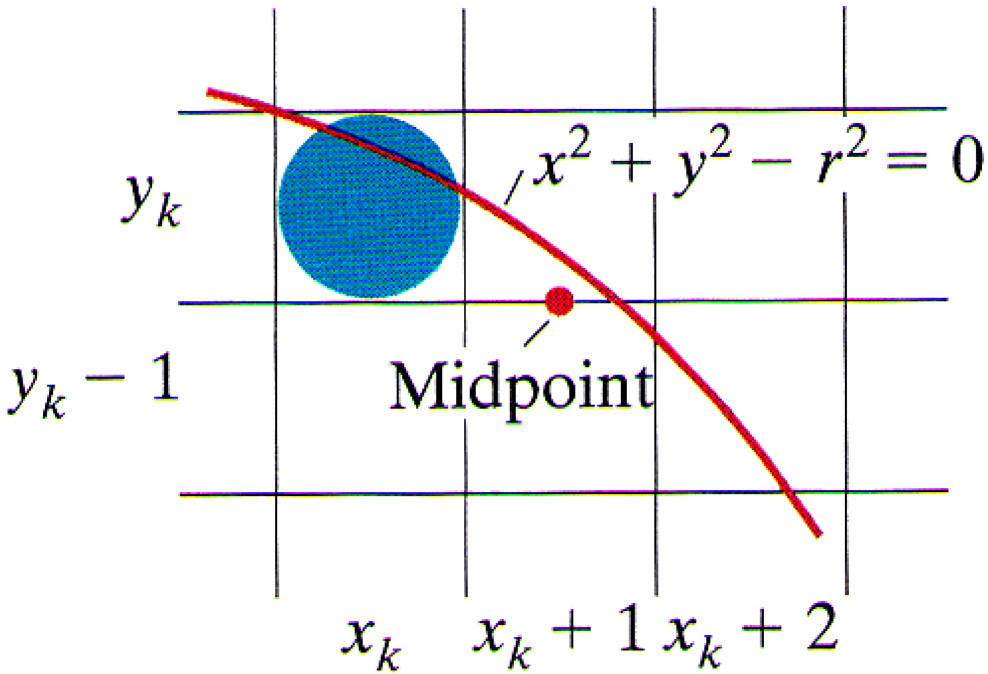
Algoritmo do Ponto Médio (Circunf.)

- Baseado na equação da circunferência define-se qual o pixel mais próxima da mesma
  - Isso é feito em um único octante, o resto é obtido por simetria

Algoritmo do Ponto Médio (Circunf.)

- $F_{circ}(x,y) = x^2 + y^2 - R^2$
  - $F_{circ}(x,y) < 0$ se (x,y) está dentro da circunf.
  - $F_{circ}(x,y) = 0$ se (x,y) está na circunf.
  - $F_{circ}(x,y) > 0$ se (x,y) está fora da circunf.
- Incrementa x e testa pixel está mais perto da circunf.
  - $F_{circ}(x,y)$ é o parâmetro de decisão e cálculos incrementais podem ser feitos

Algoritmo do Ponto Médio (Circunf.)



- Partindo de (x_k, y_k) as opções são
  - $(x_k + 1, y_k)$
  - $(x_k + 1, y_k - 1)$

Algoritmo do Ponto Médio (Circunf.)

- Então a função de decisão é
  - $P_k = Fcirc(x_k+1,\ y_k - ½)$
  - $P_k = (x_k+1)^2 + (y_k - ½)^2 - R^2$

- Se pk < 0 o ponto está dentro da circunf. e y_k está mais próxima da borda
  - Caso contrário, $y_k - 1$ está mais próxima

Algoritmo do Ponto Médio (Circunf.)

- A formulação incremental pode ser feita avaliando $x_{k+1} + 1$
  - $p_{k+1} = F_{circ}(x_{k+1} + 1, y_{k+1} - ½)$
  - $p_{k+1} = [(x_{k+1})+1]^2 + (y_{k+1} - ½)^2 - R^2$
  - $p_{k+1} = p_k + 2(x_k+1) + (y_{k+1}{}^2 - y_k{}^2) - (y_{k+1} - y_k) + 1$

- Se $p_k < 0$, então o próximo ponto é (x_{k+1}, y_k)
  - $p_{k+1} = p_k + 2x_{k+1} + 1$
- Caso contrário será $(x_k + 1, y_k - 1)$
  - $p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$
  - com $2x_{k+1} = 2x_k+2$ e $2y_{k+1} = 2y_k - 2$

Conversão matricial de elipses

- Algoritmo do Ponto Médio: mesmos princípios, com alguns complicadores...

- Tarefa: estudar algoritmo e sua derivação na apostila!!!

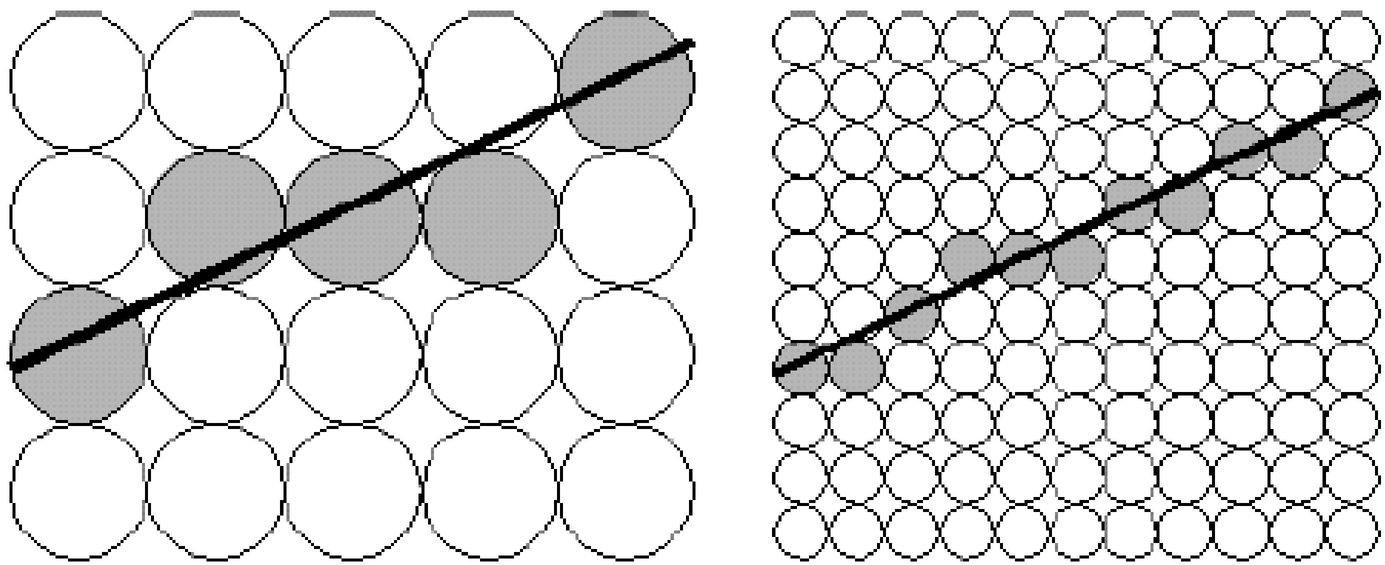
Correção de Traçado (Antialiasing)

- 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)

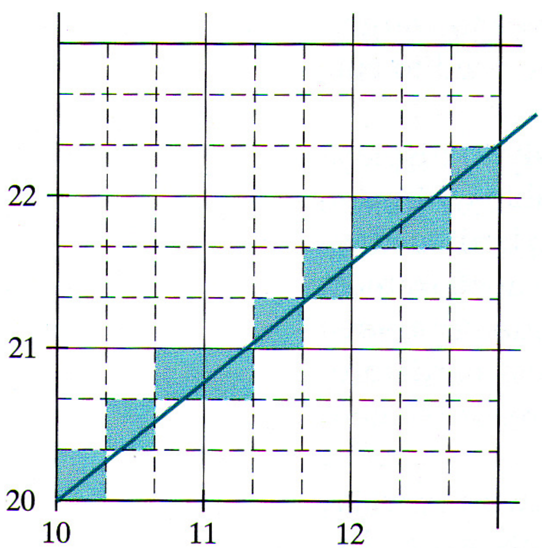
Correção de Traçado (Antialiasing)

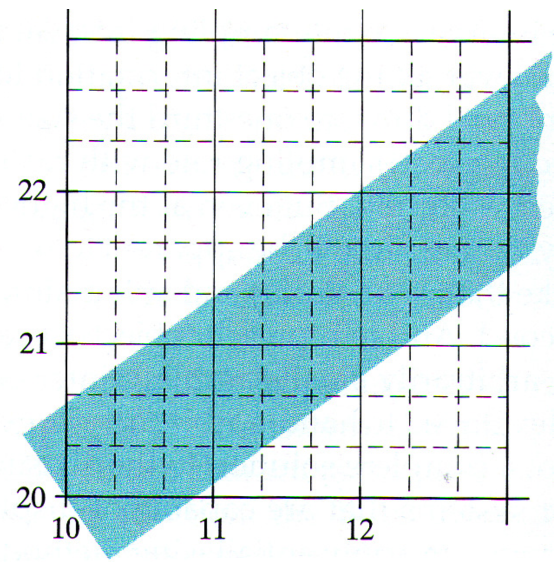
- Em sistemas que podem mostrar mais de dois níveis de intensidade, é possível usar uma solução de software

- 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





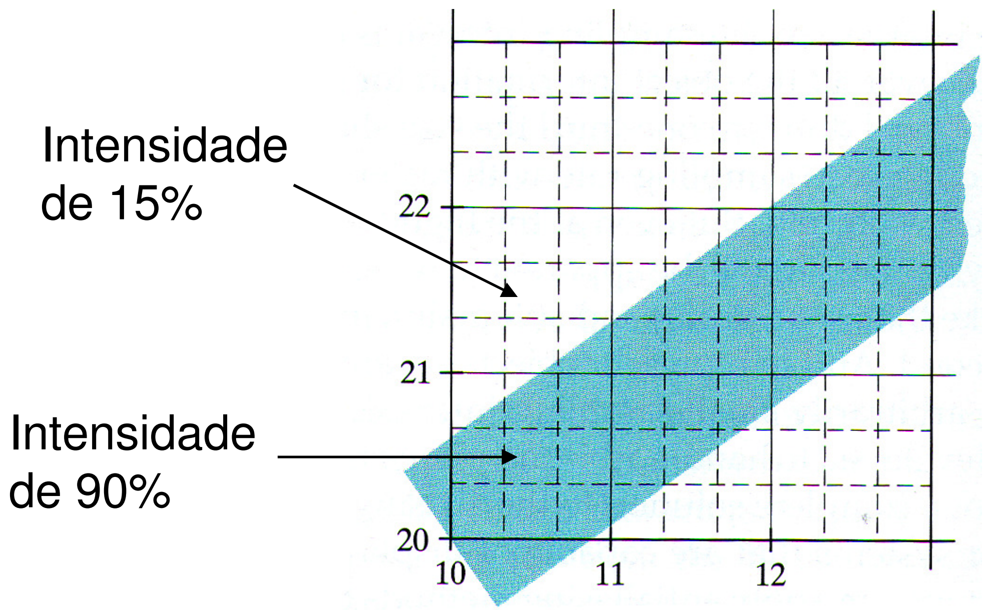
Máscara de Ponderação do Sub-Pixel

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

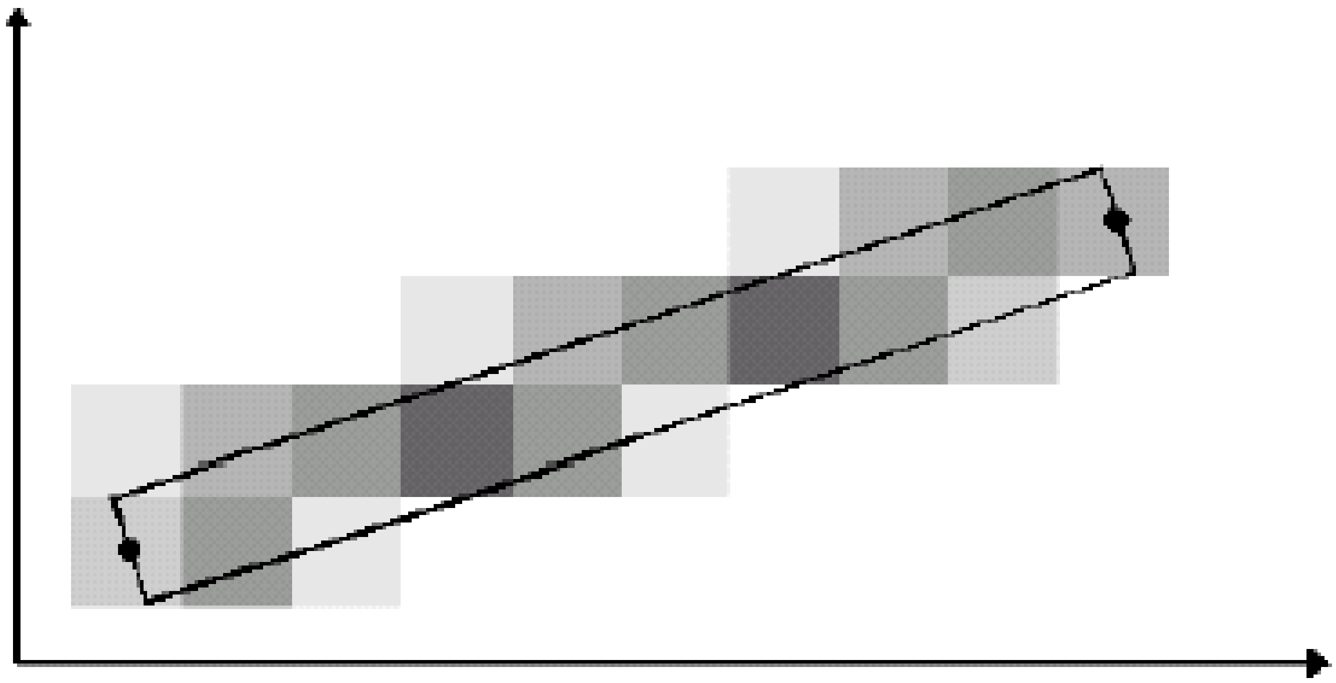
1	2	1
2	4	2
1	2	1

Antialiasing Baseado em Área

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

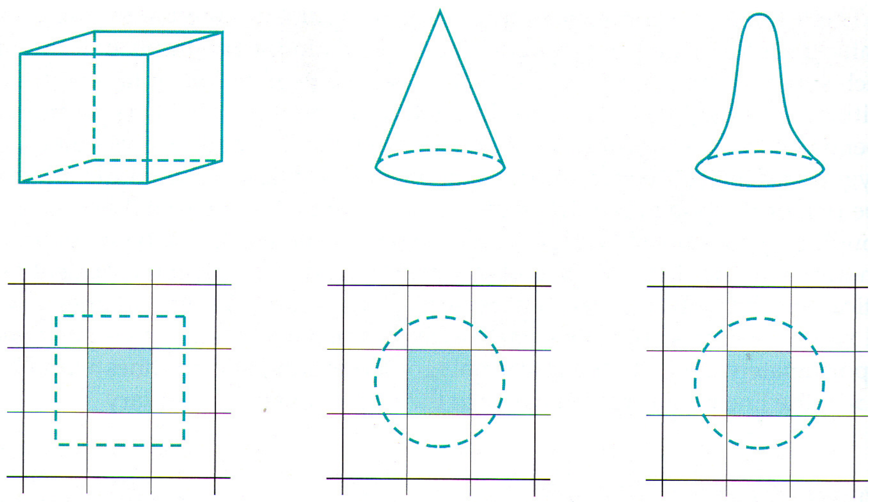


Antialiasing Baseado em Área



Técnicas de Filtragem

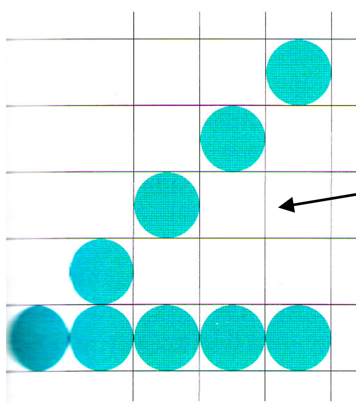
- Similar a técnica de máscara, porém mais precisa
  - Uma superfície contínua de ponderação é usada para determinar a cobertura do pixel
  - A ponderação é calculada por integração
    - Uso de tabelas para acelerar o processo



Box Filter (a) Cone Filter (b) Gaussian Filter (c)

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

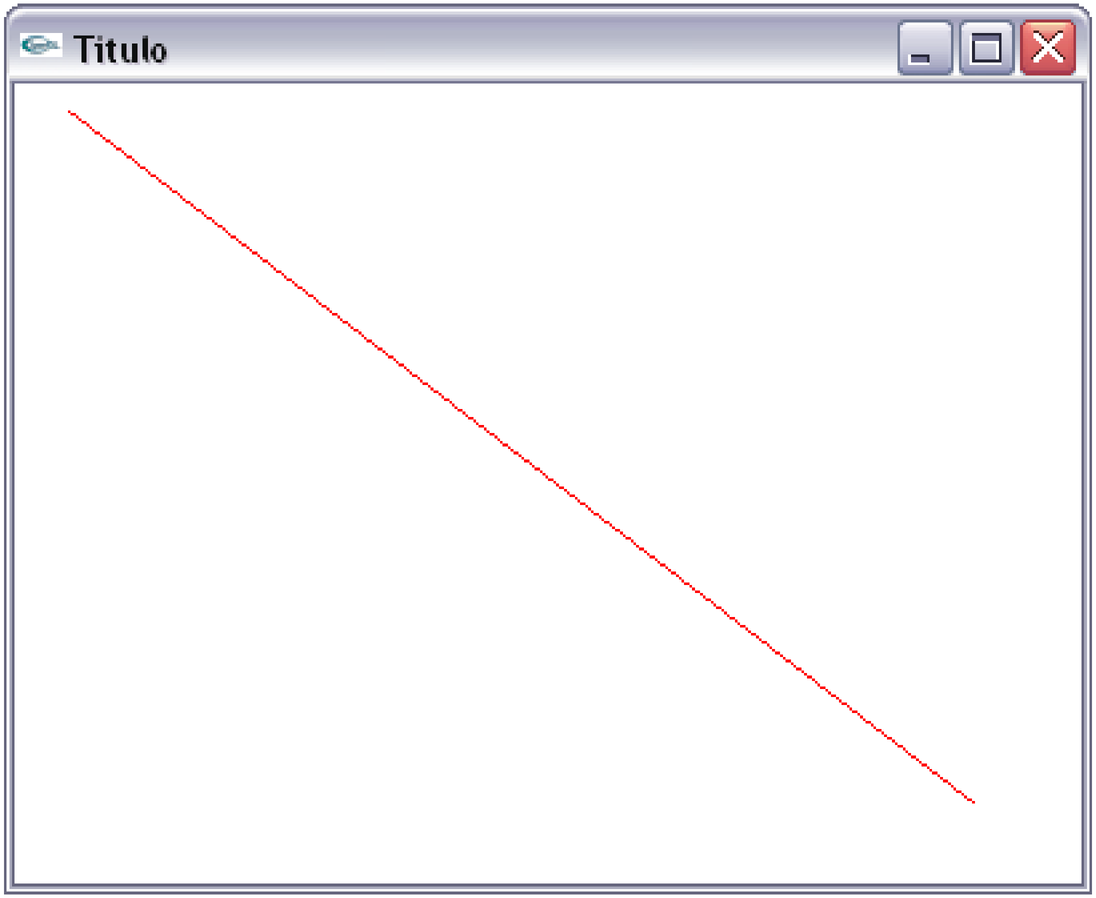


Linha diagonal é mais comprida que a vertical por um fator sqrt(2)

Antialiasing e OpenGL

- A função de antialiasing em OpenGL é ativada usando
  - glEnable(*tipoprim*)
- Tipos de primitivas
  - GL_POINT_SMOOTH
  - GL_LINE_SMOOTH
  - GL_POLYGON_SMOOTH
- Também necessário ativar *blending* em RGBA (RGB)
  - glEnable(GL_BLEND)
  - glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)

Antialiasing e OpenGL



Antialiasing e OpenGL