IA725 Computação Gráfica I
1° semestre de 2006

Profª. Wu, Shin-Ting
ting@dca.fee.unicamp.br
Bloco A – sala 317

Prof. José Mario De Martino
martino@dca.fee.unicamp.br
Bloco A – sala 317-A

Agenda

Aula	Dia	Tema	Projeto
17	09/05/06	Cor	
18	12/05/06	Cor	
19	16/05/06	Iluminação	
20	19/05/06	Iluminação	
21	23/05/06	Iluminação	
22	26/05/06	Textura	
23	30/05/06	Textura	Versão 0.2
24	02/06/06	Rasterização	
25	06/06/06	Rasterização	
26	09/06/06	Recorte	
27	13/06/06	Recorte	
-	16/06/06		
28	20/06/06	Visibilidade	
29	23/06/06	Visibilidae	
30	27/06/06	**Prova**	Versão 0.3
	30/junho		Versão Final

Rasterização

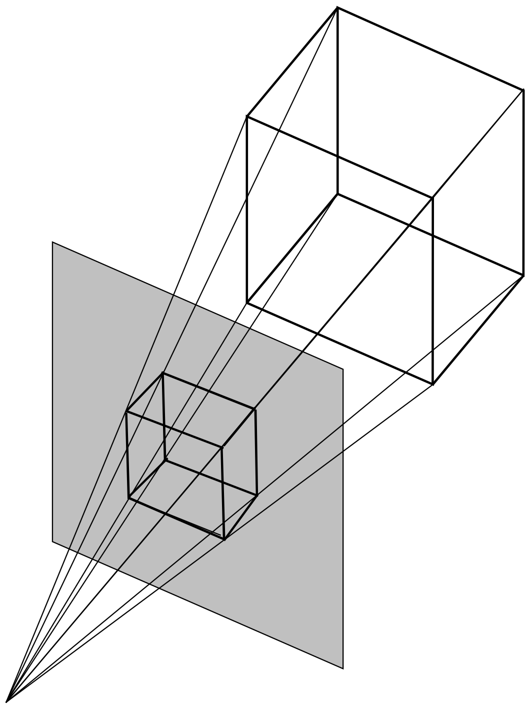
- Referências
  - Computer Graphics Principles and Practice (2nd Edition)
  - J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes
  - Addison-Wesley – 1990

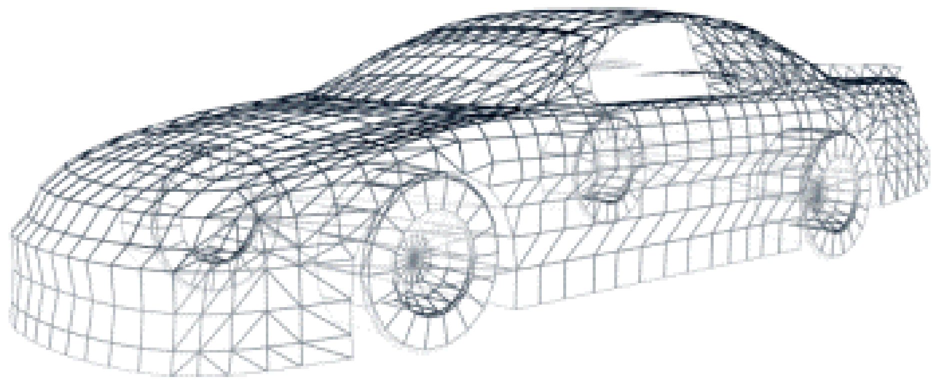
  - Procedural Elements for Computer Graphics
  - D. F. Rogers
  - McGraw-Hill – 1988

  - 3D Computer Graphcis (3rd Edition)
  - Alan Watt
  - Addison-Wesley – 2000

Algoritmos de Traçar Retas

- Motivação
  - Apresentação *wireframe (*em aramado*)*





Algoritmos de Traçar Retas

- Para a geração da imagem, após a projeção dos polígonos dos objetos da cena, faz-se necessária a definição da cor de cada *pixel*.

- A apresentação *wireframe* (em aramado) mostra apenas as arestas dos polígonos.

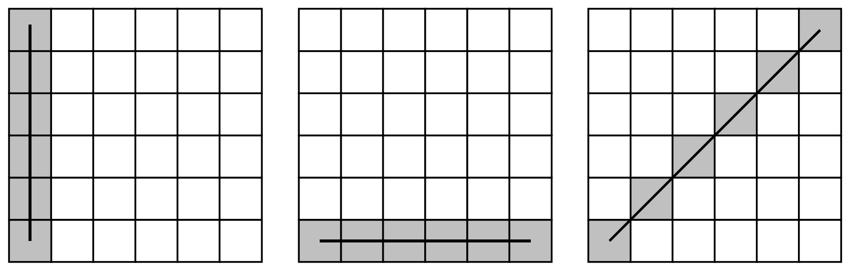
- Para a apresentação *wireframe* faz-se necessário algoritmos de traçar retas (*line drawing algorithms*).

- A apresentação *wireframe* permite a rápida geração da imagem, sendo interessante na fase de construção da geometria dos objetos ou mesmo cena onde a interatividade é importante.

Algoritmos de Traçar Retas

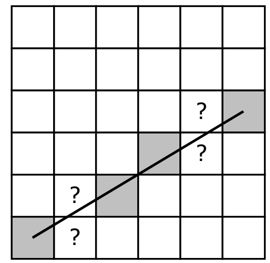
- Traçar uma reta significa acionar (acender) os *pixels* apropriados para que se tenha uma linha reta no dispositivos de apresentação tipo varredura (*raster*).

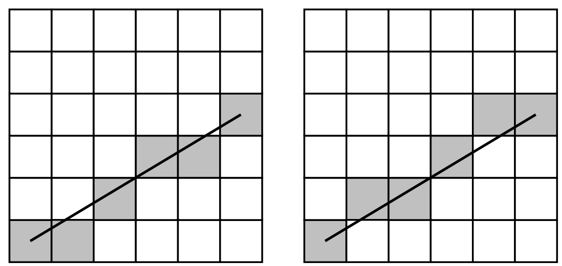
- Exemplos: reta vertical, horizontal e a 45°



Algoritmos de Traçar Retas

- Dependendo da inclinação da reta a escolha do *pixel* a ser acionado não é óbvia.





Algoritmos de Traçar Retas

- Algoritmo DDA (Digital Differential Analyzer)
  - Equação da reta

$$\frac{\Delta y}{\Delta x} = \frac{y - y_i}{x - x_i} \quad \text{com} \quad \frac{\Delta y}{\Delta x} = \frac{y_f - y_i}{x_f - x_i}$$

onde :

(x_i, y_i) – ponto inicial

(x_f, y_f) – ponto final

  - De forma incremental

$$y_{n+1} = y_n + \Delta y \quad \Rightarrow \quad y_{n+1} = y_n + \frac{y_f - y_i}{x_f - x_i}\Delta x$$

ou

$$x_{n+1} = x_n + \Delta x \quad \Rightarrow \quad x_{n+1} = x_n + \frac{x_f - x_i}{y_f - y_i}\Delta y$$

Algoritmo DDA

- Incrementar na direção de maior variação (maior Δ) de 1 (uma unidade).

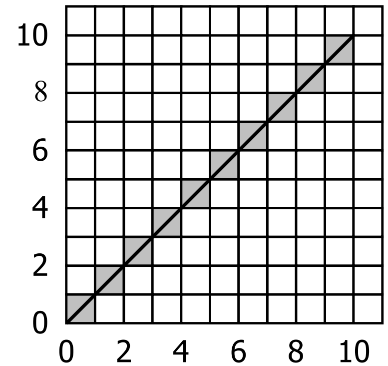
- Na outra direção incrementar de ($\Delta y/\Delta x$ ou $\Delta x/\Delta y$, dependendo se esta direção é y ou x)

Algoritmo DDA

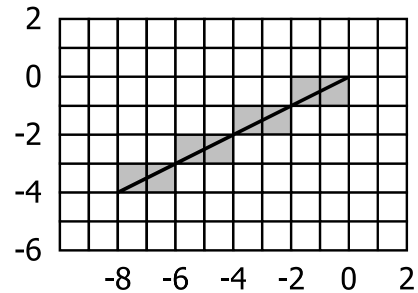
```
#define SIGN(x)  ((x) < 0 ? (-1): (1))
#define ABS(x)   ((x) < 0 ? (-x) : (x))
#define FLOOR(x) ((x) < 0 ? ( (x) - (int)(x) != 0 ? ((int)(x) - 1) : ((int)(x))) : (int)(x))

if( ABS( (x2 - x1) ) >= ABS( (y2 - y1) ) )
    length = ABS( (x2 - x1) );
else
    length = ABS( (y2 - y1) );

deltax = (float) (x2 - x1) / (float) length;
deltay = (float) (y2 - y1) / (float) length;
x = x1 + 0.5 * SIGN (deltax);
y = y1 + 0.5 * SIGN (deltay);

for (i = 0; i < length; i++) {
     setPixel( FLOOR(x), FLOOR(y) ) ;
     x += deltax;
     y += deltay;
}
```

Algoritmo DDA

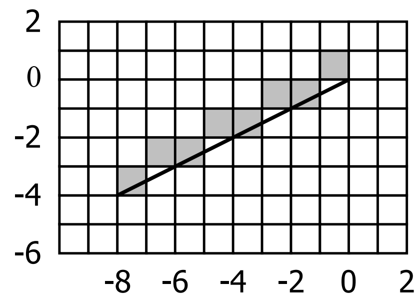
- Exemplo: (0,0) → (10, 10)



- Exemplo: (0,0) → (-8, -4)



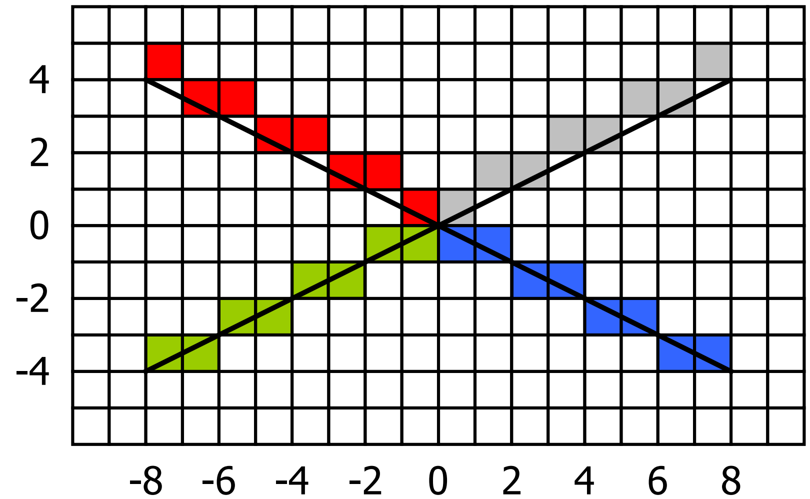
- Exemplo: (-8, -4) → (0,0)



Algoritmo DDA

- Exemplo

  - (0,0) → (8, 4)
  - (0,0) → (-8, 4)
  - (0,0) → (-8, -4)
  - (0,0) → (8, -4)



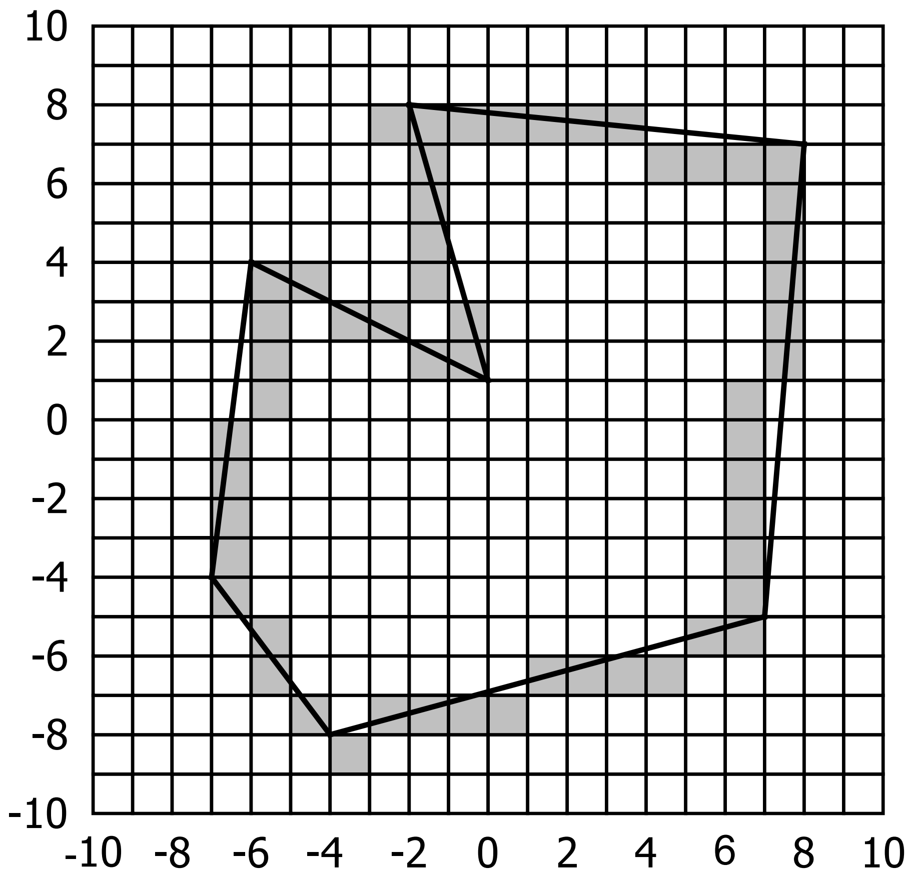
Algoritmo DDA

- Exercício
  - Traçar o contorno do polígono abaixo utilizando o algoritmo DDA.
    - Vértices (x, y) do polígono no plano imagem

(0, 1)
(-2, 8)
(8, 7)
(7, -5)
(-4, -8)
(-7, -4)
(-6, 4)

Algoritmo DDA

- Exercício



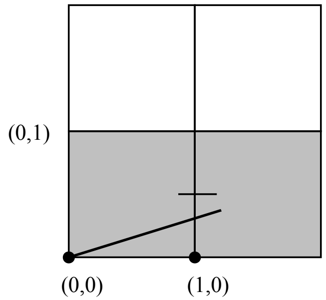
Algoritmos para Traçar Linhas

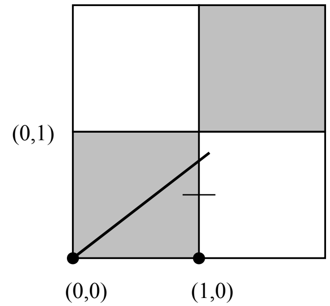
- Algoritmo de Bresenham

  - A cada iteração, uma das direções sempre é efetuado incremento unitário.

  - A cada interação, determina, em função da distância (erro) entre a reta e o *pixel* a ser ativado, se efetua ou não incremento (unitário) na outra direção.

Algoritmo de Bresenham





$x = 0, \quad y = 0$

$\text{erro} = -0{,}5 \quad \text{(inicialização)}$

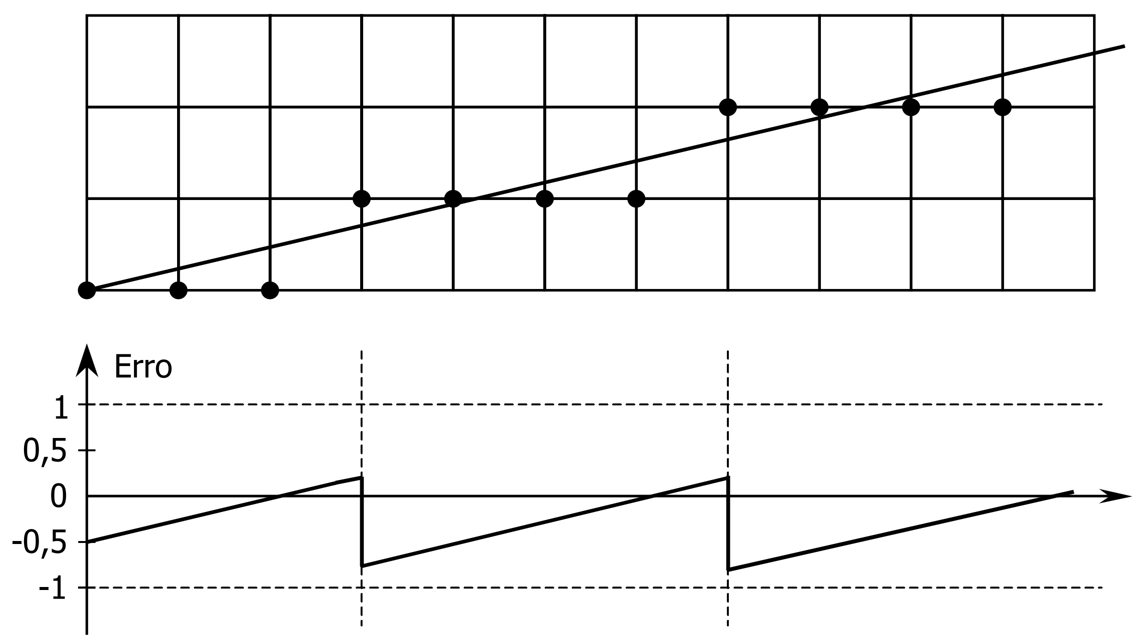
setPixel(x, y)

$$\text{erro} \mathrel{+}= \frac{\Delta y}{\Delta x}$$

$$\begin{cases} y = y & \text{se } \mathit{erro} < 0 \quad \left(0 \le \dfrac{\Delta y}{\Delta x} < \dfrac{1}{2}\right) \\ y \mathrel{+}= 1 & \text{se } \mathit{erro} \ge 0 \quad \left(\dfrac{1}{2} \le \dfrac{\Delta y}{\Delta x} \le 1\right) \end{cases}$$

setPixel(x, y)

Algoritmo de Bresenham



Algoritmo de Bresenham - 1°. Octante (float)

```
x = (int) x1;
y = (int) y1;

deltax = x2 - x1;
deltay = y2 - y1;

erro = (deltay / deltax) -  0.5;

for (i = 0; i < deltax; i++) {

   setPixel( x, y );

   while (erro >= 0.0) {
      y += 1;
      erro = erro - 1.0;
   }
   x += 1;
   erro = erro + (deltay / deltax);
}
```

Algoritmo de Bresenham

- Na versão apresentada, o erro é uma variável real (float ou double). Portanto, todo o cálculo que a envolve será efetuado em ponto-flutuante.

- Para agilizar o algoritmo, é possível derivar uma versão onde a variável erro é inteira. Para tanto, basta considerar:

$$\text{erro}' = 2 \cdot \Delta x \cdot \text{erro}$$

Algoritmo de Bresenham - 1°. Octante (inteiro)

```
x = x1;
y = y1;

deltax = x2 - x1;
deltay = y1 - y2;

erro = 2 * deltay - deltax;

for (i = 0; i < deltax; i++)  {

   setPixel(x,  y);

    while (erro >= 0) {
       y += 1;
      erro = erro - 2 * deltax;
    }

    x += 1;
    erro = erro + 2 * deltay;
}
```

Algoritmo de Bresenham - inteiro

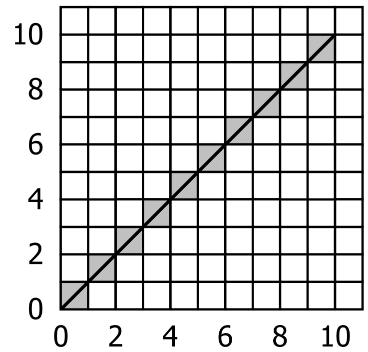
```
#define SIGN(x) ((x) < 0 ? (-1) : (1))
#define ABS(x) ((x) < 0 ? (-x) : (x))
#define FALSE 0
#define TRUE 1

deltax = ABS ( (x2 - x1) );
deltay = ABS ( (y2 - y1) );
signalx = SIGN ( (x2 - x1) );
signaly = SIGN ( (y2 - y1) );
x = x1;
y = y1;
if  (signalx < 0)
   x -=1;
if (signaly < 0 )
   y -= 1;

// trocar deltax com deltay dependendo da inclinacao da reta
interchange = FALSE;
if ( deltay > deltax) {
   tmp = deltax;
   deltax = deltay;
   deltay = tmp;
   interchange = TRUE;
}
erro = 2 * deltay - deltax;
```

Algoritmo de Bresenham - inteiro (cont.)

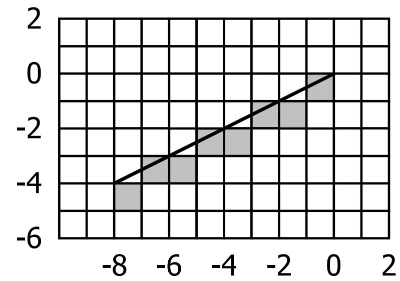
```
for (i = 0; i < deltax; i++) {

   setPixel( x, y);

   while (erro >= 0) {

      if (interchange)
         x = x + signalx;
       else
         y = y + signaly;

      erro = erro - 2 * deltax;

   } // while

   if (interchange)
      y = y + signaly;
   else
      x = x + signalx;

   erro = erro + 2 * deltay;

} // for
```

Algoritmo de Bresenham

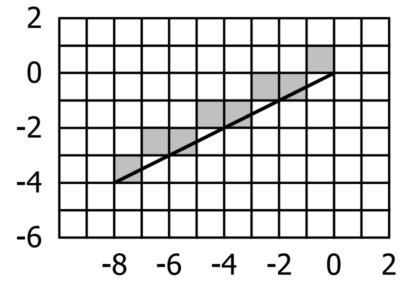
- Exemplo: (0,0) → (10, 10)



- Exemplo: (0,0) → (-8, -4)



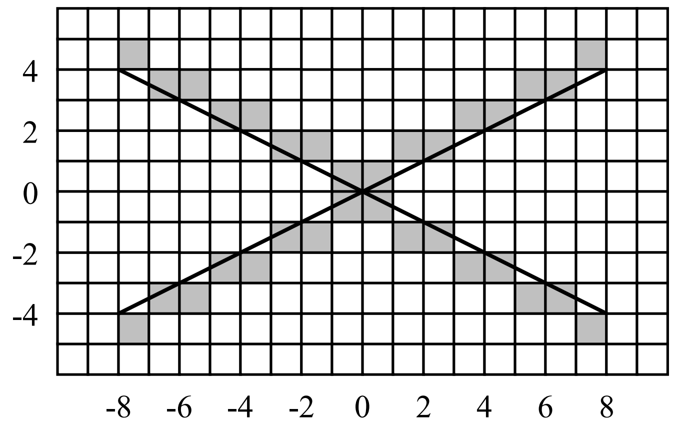
- Exemplo: (-8, -4) → (0,0)



Algoritmo de Bresenham

- Exemplo

(0,0) → (8, 4)
(0,0) → (-8, 4)
(0,0) → (-8, -4)
(0,0) → (8, -4)



Algoritmo de Bresenham

- Exercício

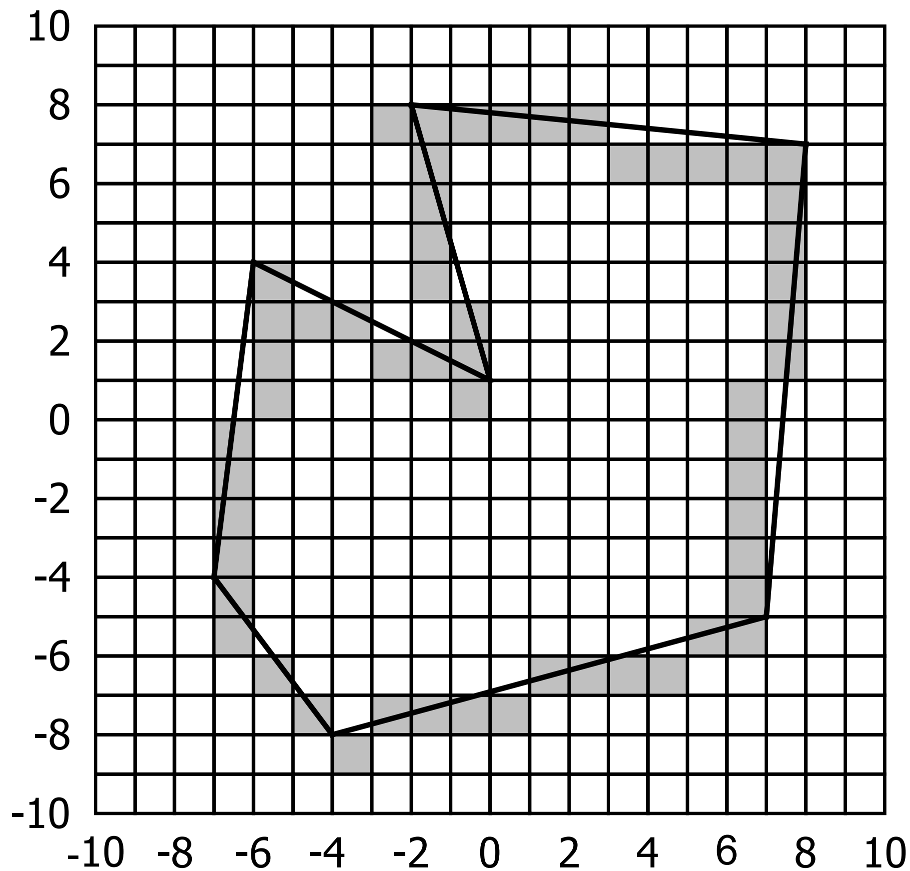
  - Traçar o contorno do polígono abaixo utilizando o algoritmo de Bresenham (inteiro).

    - Vértices (x, y) do polígono no plano imagem

      (0, 1)
      (-2, 8)
      (8, 7)
      (7, -5)
      (-4, -8)
      (-7, -4)
      (-6, 4)

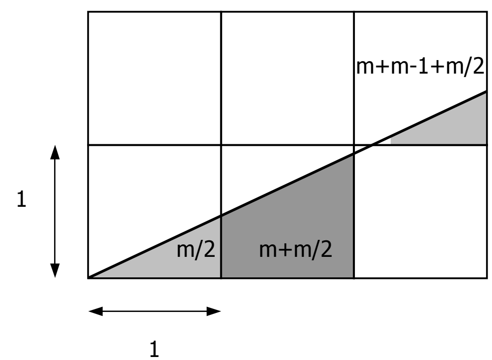
Algoritmo de Bresenham

- Exercício



Algoritmo de Bresenham – com anti-aliasing

- Filosofia
  - A intensidade reflete a área do *pixel* coberta pela reta.
  - Estimativa da área (y = mx).



Algoritmo de Bresenham com anti-aliasing - 1°. Octante

```
deltax = x2 - x1;
deltay = y2 - y1;
x = x1;
y = y1;

m = (float) deltay / (float) deltax;
w = 1 - m;
erro = m / 2.0f;

for(int i = 0; i < deltax; i++) {
          setPixel(x, y, erro);
          if( erro < w ) {
                    x++;
                    erro = erro + m;
          } else {
                    x++;
                    y++;
                    erro = erro  -w;
          }
}
```

Algoritmo de Bresenham – com anti-aliasing

- Exemplo
  - $(0,0) \rightarrow (8, 4)$