Rasterização de Linhas

Equação da reta

- Lei de formação:

$$y = f(x) = a\,x + b$$

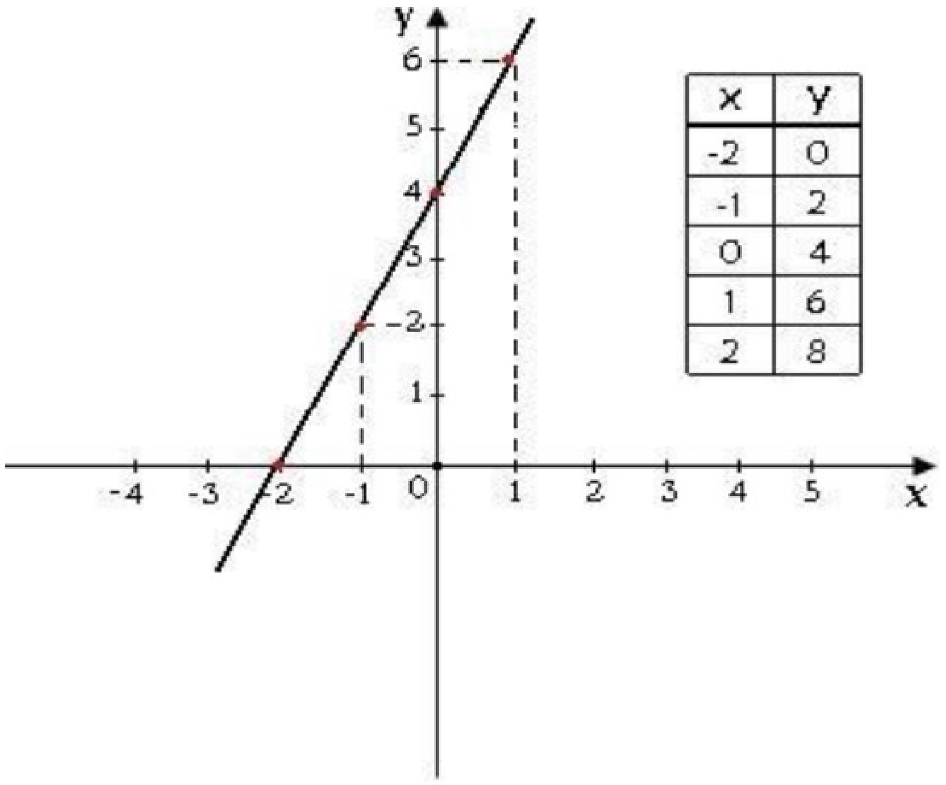
  - 'a' → coeficiente angular
  - 'b' → coeficiente linear
  - 'x' → variável independente

Equação da reta

- Exemplo:

  $f(x) = 2x + 4$

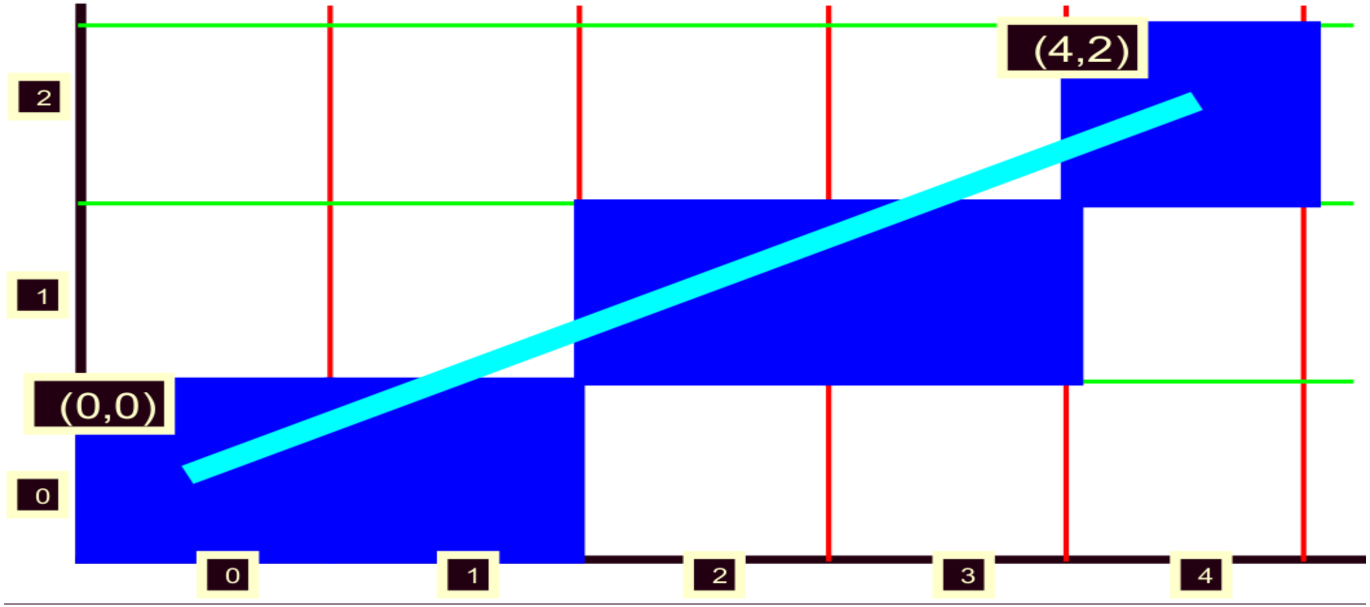
  - $f(x) = 2(-2) + 4 = 0$
  - $f(x) = 2(-1) + 4 = 2$
  - $f(x) = 2(0) + 4 = 4$
  - $f(x) = 2(1) + 4 = 6$
  - $f(x) = 2(2) + 4 = 8$



x	y
-2	0
-1	2
0	4
1	6
2	8

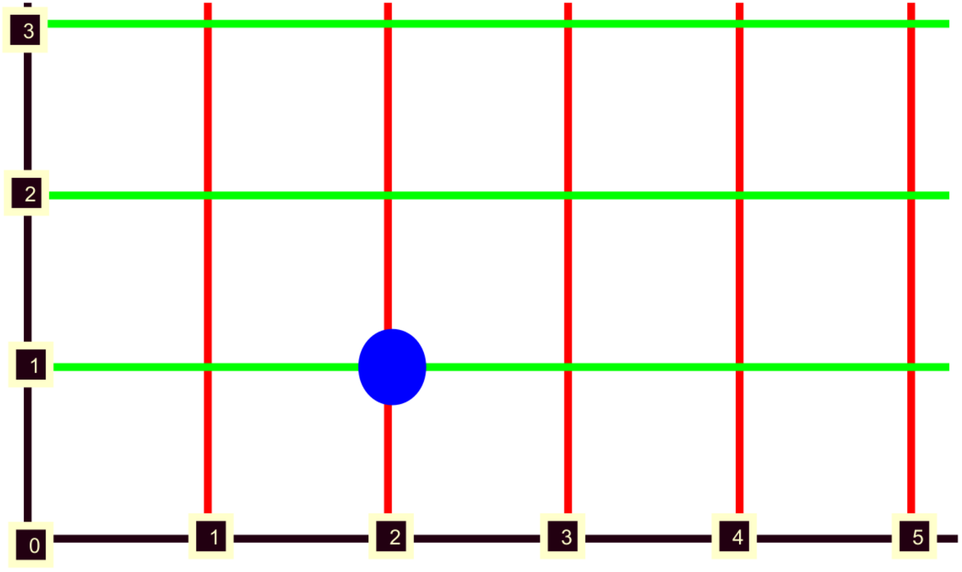
Rasterização

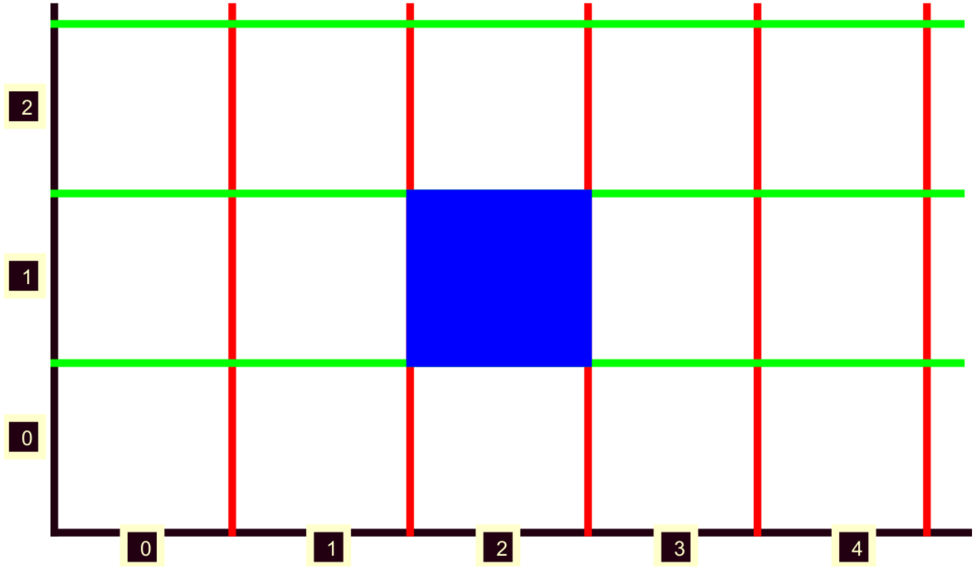
- Converte informação de vértices/arestas em pixels a serem mostrados na imagem



Rasterização

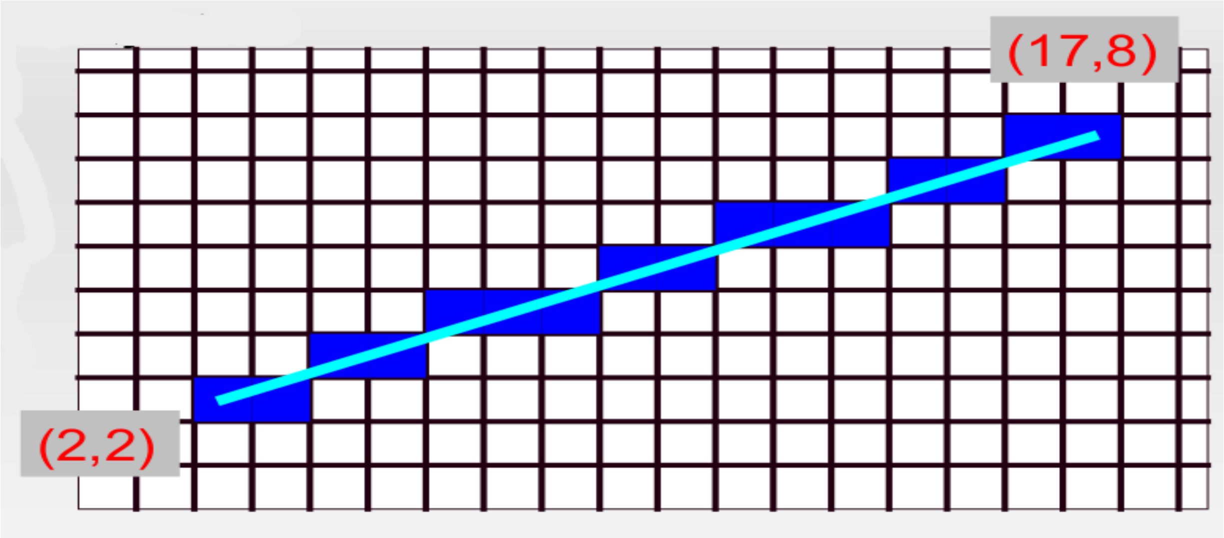
- Poderíamos obter a reta mapeando cada ponto do SRU para o SRD, mas:





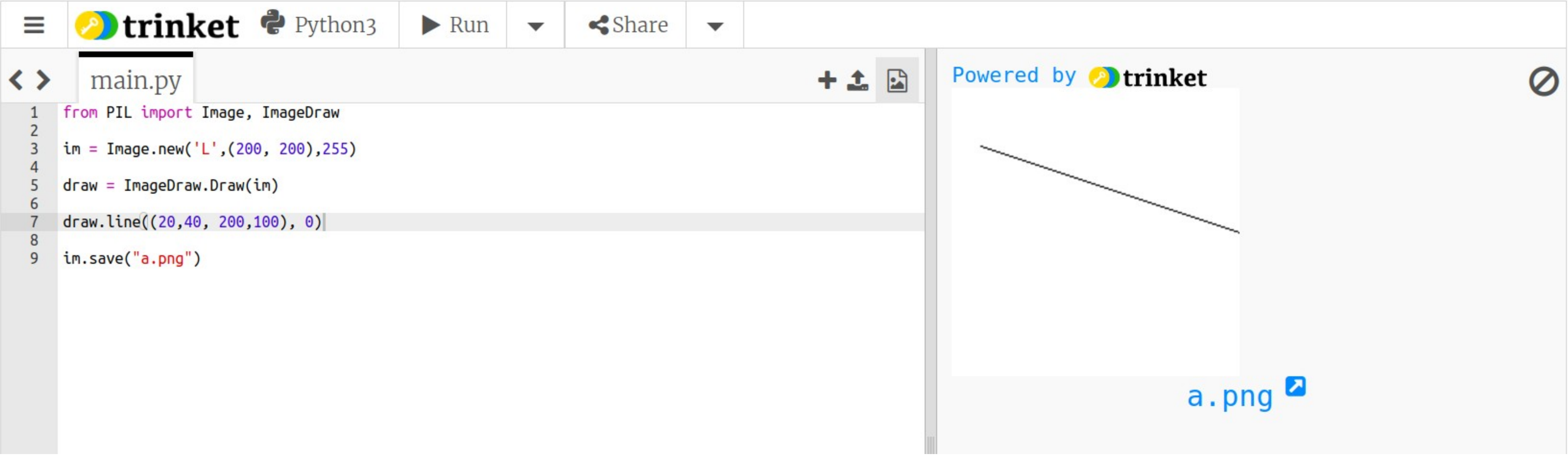
Rasterização

- Objetivo:
  - Aparência contínua
  - Uniformidade
  - Próximo a linha ideal
  - Rapidez de rasterização



Rasterização

- Método Slope-Intercept
- Método DDA
- Algoritmo de Bresenham



Método Slope-Intercept

- Da equação: $$y = f(x) = a\,x + b$$

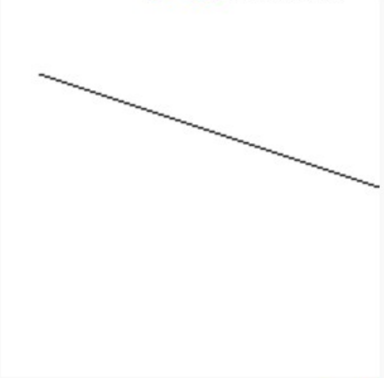
```
xi=20
xf=200
yi=40
yf=100
```

trinket Python3 Run Share

main.py

```
from PIL import Image, ImageDraw

im = Image.new('L',(200, 200),255)

draw = ImageDraw.Draw(im)

xi=20
xf=200
yi=40
yf=100

a=float(yf-yi)/(xf-xi)
b=yi - a*xi

for x in range(xi,xf):
  y = int(a*x+b)
  draw.point((x,y),0)

im.save("a.png")
```

Powered by trinket



a.png

Método Slope-Intercept

- Da equação: $$y = f(x) = a\,x + b$$

```
xi=20
xf=40
yi=40
yf=200
```

O que acontece?

Método Slope-Intercept

- Da equação: $$y = f(x) = ax + b$$

```
xi=20
xf=40
yi=40
yf=200
```

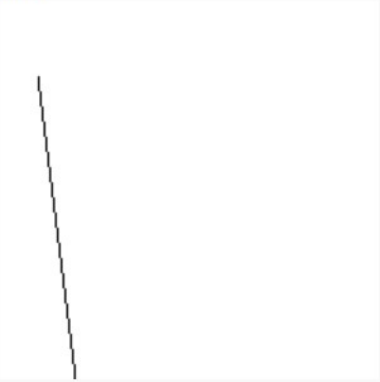
trinket Python3 Run Share

main.py

```python
from PIL import Image, ImageDraw

im = Image.new('L',(200, 200),255)

draw = ImageDraw.Draw(im)

xi=20
xf=40
yi=40
yf=200

a=float(yf-yi)/(xf-xi)
b=yi - a*xi

print(a)

if a<1:
  for x in range(xi,xf):
    y = int(a*x+b)
    draw.point((x,y),0)
else:
  for y in range(yi,yf):
    x = int((y-b)/a)
    draw.point((x,y),0)

im.save("a.png")
```

Powered by trinket

8.0



a.png

Método DDA

- Melhora o método anterior
- DDA → Digital Diferential Analyzer
  - Foca na equação: a = dy/dx
- Objetivo:
  - Fazer uma operação a menos dentro do ciclo

Método DDA

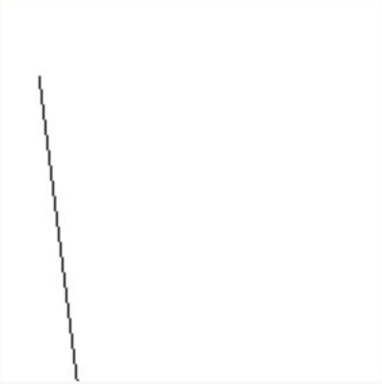
trinket Python3 Run Share

main.py

```python
from PIL import Image, ImageDraw

im = Image.new('L',(200, 200),255)

draw = ImageDraw.Draw(im)

xi=20
xf=40
yi=40
yf=200

a=float(yf-yi)/(xf-xi)
b=yi - a*xi

print(a)

if a<1:
  y=yi
  for x in range(xi,xf):
    y+=a
    draw.point((x,y),0)
else:
  x=xi
  for y in range(yi,yf):
    x+=1/a
    draw.point((x,y),0)

im.save("a.png")
```

Powered by trinket

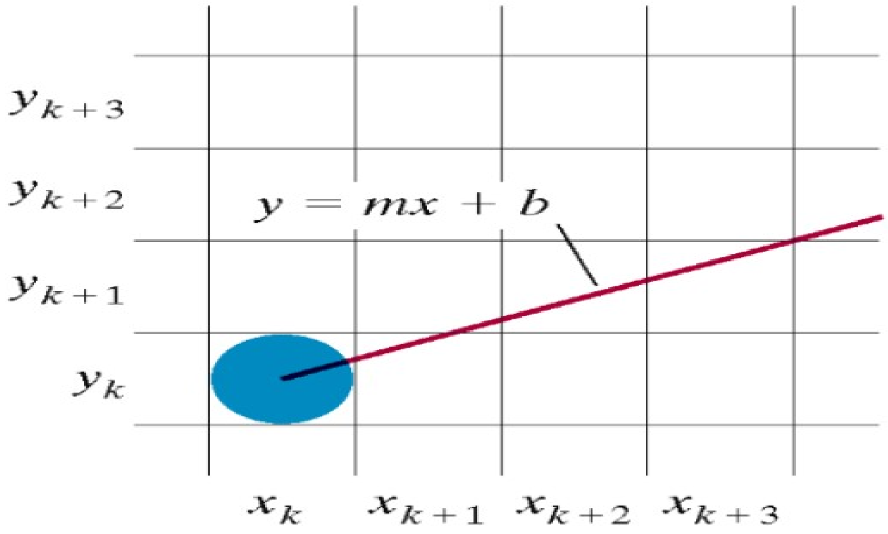
8.0



a.png

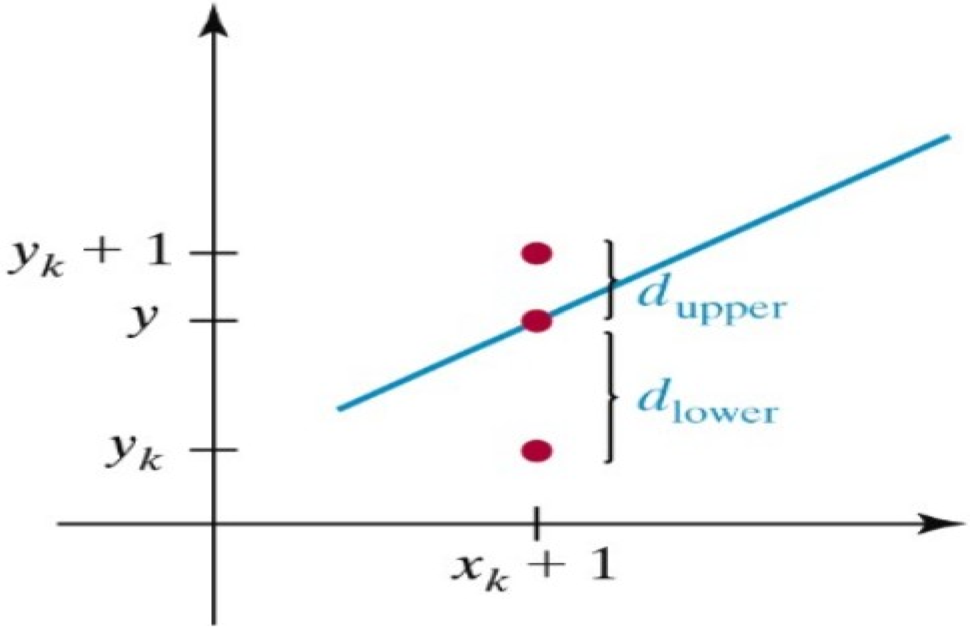
Algoritmo de Bresenham

- Algoritmo rápido eficiente e preciso
- Melhora o DDA para usar aritmética inteira



Algoritmo de Bresenham

- d (upper, lower) → distâncias entre o valor real e discretizado



Algoritmo de Bresenham

- Decisão de qual pixel preencher:
  - Sinal de $d_{lower} - d_{upper}$

$$d_{lower} - d_{upper} \begin{cases} + \rightarrow y_k + 1 \\ - \rightarrow y_k \end{cases}$$

Algoritmo de Bresenham

- Cálculo de: $d_{lower} - d_{upper}$

$$y = m(x_k + 1) + b \quad \text{(cálculo da coordenada y na linha } x_k+1\text{)}$$

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

$$\mathbf{d_{upper} = (y_k + 1) - y} \quad = \quad y_{k+1} - m\,(x_k + 1) - b$$

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

$$\mathbf{d_{lower} - d_{upper} = 2\,(\,dy\,/\,dx\,)(x_k + 1) - 2y_k + 2b - 1}$$

Algoritmo de Bresenham

- Parâmetro de decisão: p_k

  $p_k = dx\ (d_{lower} - d_{upper})$
- Sinal é o mesmo de: $d_{lower} - d_{upper}$
  - Cálculo mais simples

$$\mathbf{p_k} = dx\ (\ 2\ (dy / dx)(x_k + 1) - 2y_k + 2b - 1\ )$$

$$\mathbf{p_k = 2\ dy\ x_k - 2\ dx\ y_k + c}$$

Onde c é uma constante; $\mathbf{c = 2\ dy + 2\ dx\ b - dx}$

Algoritmo de Bresenham

- Achando p_{k+1}

$$
\begin{cases}
p_{k+1} = p_k + 2\,dy & p_k < 0 \rightarrow \text{negativo} \\
p_{k+1} = p_k + 2\,(dy - dx) & p_k >= 0 \rightarrow \text{positivo}
\end{cases}
$$

- $p_k = 2*dy + dx$ $\qquad k = 0$

Algoritmo de Bresenham

trinket Python3 Run Share

main.py

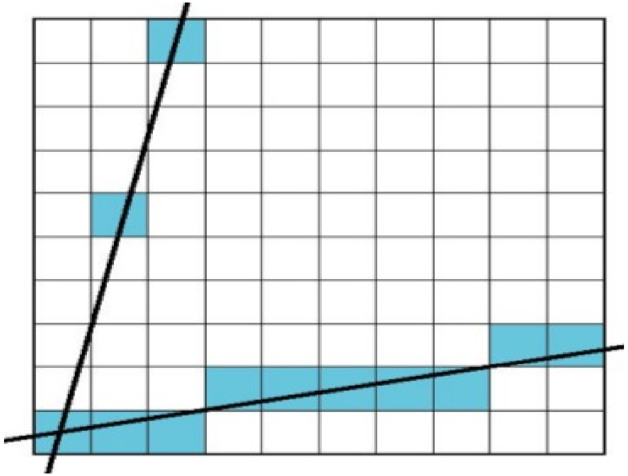
```
from PIL import Image, ImageDraw

im = Image.new('L',(200, 200),255)

draw = ImageDraw.Draw(im)

xi=20
xf=200
yi=40
yf=100

dx=abs(xf-xi)
dy=abs(yf-yi)
p=2*dy-dx

if xi>xf:
  x=xf
  y=yf
  xf=xi
else:
  x=xi
  y=yi

draw.point((x,y),0)
while x<xf:
  x+=1
  if p<0:
    p+=2*dy
  else:
    y+=1
    p+=2*(dy-dx)
  draw.point((x,y),0)

im.save("a.png")
# From Hearn & Baker's Computer Graphics with OpenGL, 3rd Edition
```

Powered by trinket

a.png

Algoritmo de Bresenham

- Problema
  - Para cada x, desenhar o pixel no melhor y



  - Solução: simetria → trocar x por y

Exercício

- Aplicar o algoritmo de Bresenham para:

```
xi=20
xf=40
yi=40
yf=200
```

Solução: simetria → trocar x por y