PODER EXECUTIVO
MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE RORAIMA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

DCC703 - COMPUTAÇÃO GRÁFICA

RELATÓRIO DO PROJETO DE RASTERIZAÇÃO DE LINHAS

ALUNOS:
Marcos Vinícius Tenacol Coêlho - 2021000759

Fevereiro de 2025
Boa Vista/Roraima

Resumo

Este trabalho explora a implementação e demonstração de diversos algoritmos fundamentais para a rasterização de formas geométricas em gráficos computacionais.

Para a **rasterização de linhas**, são apresentados três algoritmos distintos:

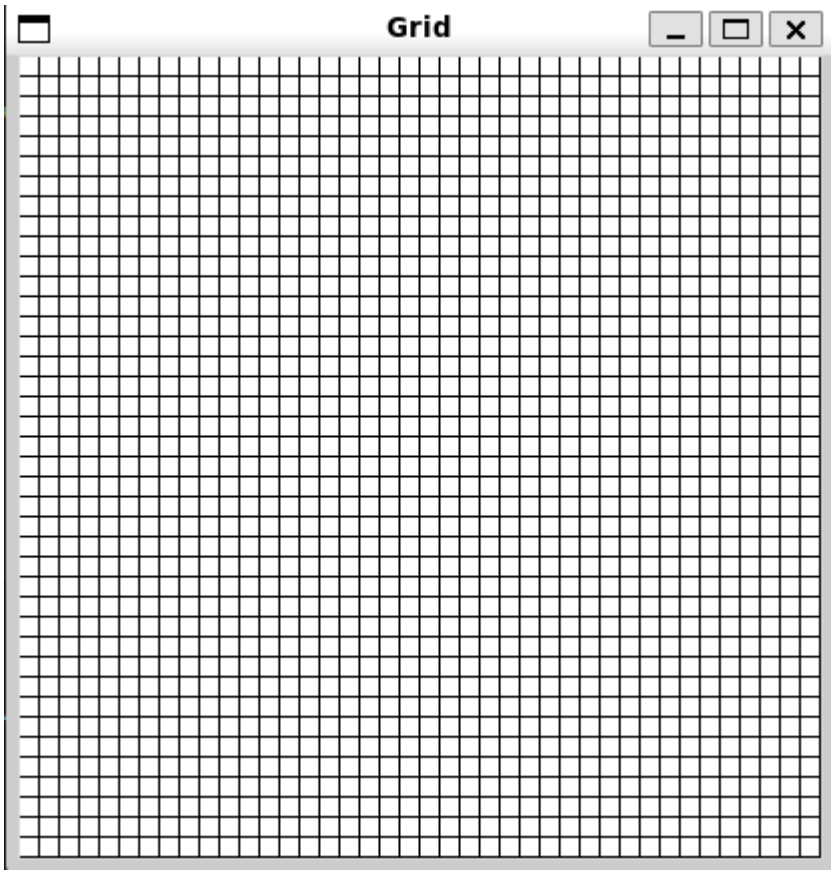
1. **Analítico**, que utiliza equações matemáticas diretas para determinar os pontos da linha;
2. **DDA**, que faz uso de incrementos fracionários para aproximar a linha de forma eficiente;
3. **Bresenham**, conhecido por sua precisão e desempenho, utilizando apenas operações inteiras para definir os pixels da linha.

Todos os algoritmos de rasterização implementados são capazes de traçar com precisão as formas especificadas, podendo ser adaptados para a criação de outras figuras geométricas.

Implementações

Para realizar a criação do grid de pixels, utilizei a biblioteca **Tkinter**, que oferece uma interface gráfica simples e eficiente para manipulação de elementos visuais. O **Tkinter** foi empregado tanto para criar o grid de pixels quanto para possibilitar a interação com ele, permitindo a "pintura" de pixels com base nos algoritmos de rasterização.

Demonstração do grid 40x40 gerado:



Método de Rasterização de linhas analítico

Esse é o método mais simples e intuitivo entre os 3 de rasterização de linhas, ele usa como base a equação de reta (y=mx+b), o código foi feito com base no pseudo-código:

% reta vertical X1 = X2	
Não	Sim

m = (Y2 - Y1)/(X2 - X1)
b = Y2 - m*X2
para X de X1 até X2
  Y = m*X + b
  liga_pixel(X, Y, Cor)
	para Y de Y1 até Y2
	  liga_pixel(X1, Y, Cor);

Código utilizado:

```
def analitico(x1, y1, x2, y2, cor="black"):
    if x1 == x2:
        for y in range(y1, y2 + 1):
            pintar(x1, y,cor)
        return

    if x2 < x1:
        x1, x2, y1, y2 = x2, x1, y2, y1

    m = (y2 - y1) / (x2 - x1)
    b = y2 - m * x2

    for x in range(x1, x2 + 1):
        y = round(m * x + b)
        pintar(x, y,cor)
```

1º Passo. Checa se é uma linha vertical.

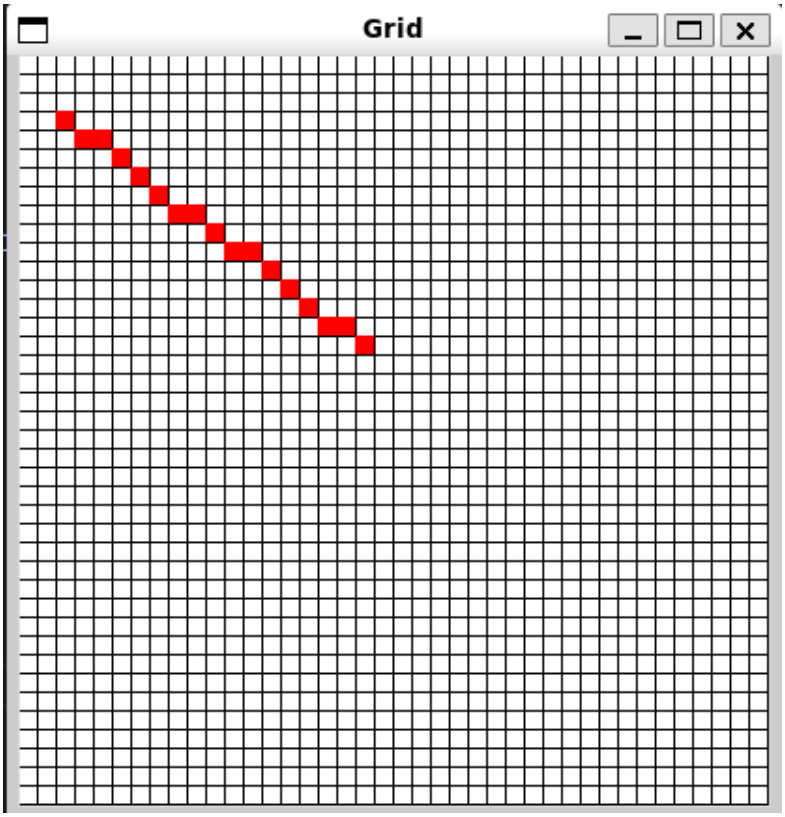
2º Passo. Organiza os valores, caso x2 seja menor que x1.

3º Passo. Realiza o cálculo da reta.

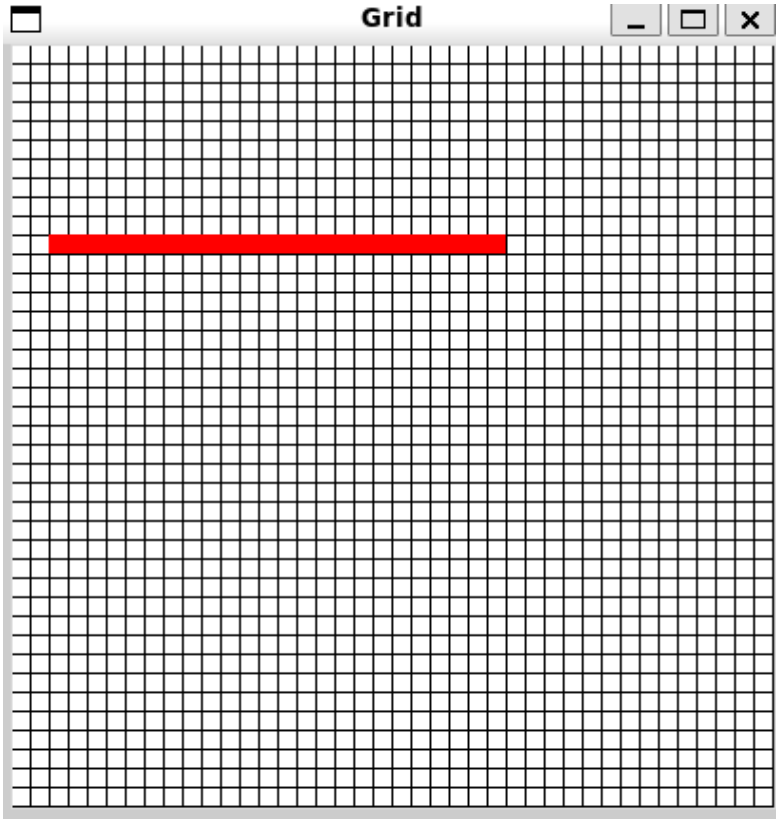
4º Passo. Gera o valor de y e pinta a coordenada.

Algumas demonstrações deste método:

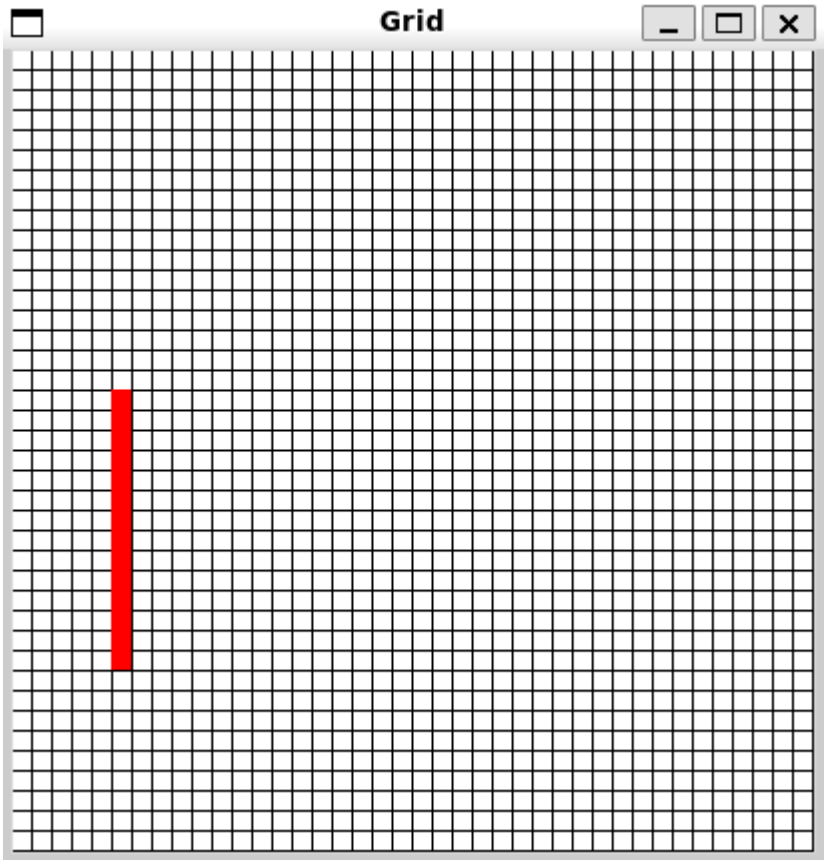
- analitico(2, 3, 18, 15, "red"):



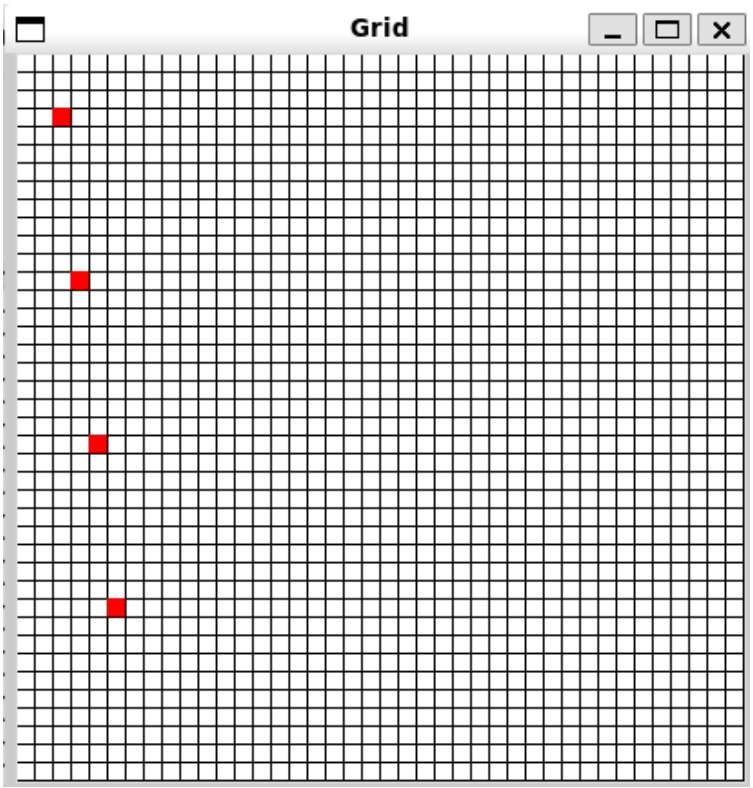
- analitico(2, 10, 25, 10, "red"):



- analitico(5, 17, 5, 30, "red"):



- analitico(2, 3, 5, 30, "red"):

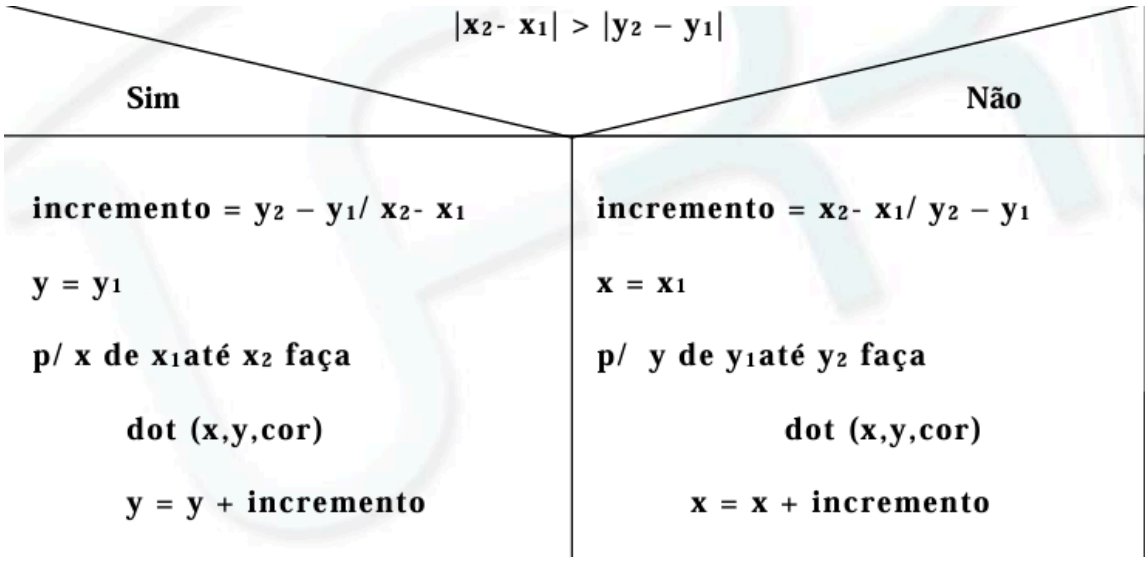


Método de Rasterização de linhas DDA:

O **método analítico** para rasterização de linhas utiliza uma abordagem matemática direta baseada na equação da reta para determinar quais pixels devem ser ativados no grid. A linha é descrita por uma equação linear no formato **y = mx + b**, onde **m** é o coeficiente angular (pendente) e **b** é o coeficiente linear (interceptação).

Para desenhar a linha, o algoritmo calcula os valores de **x** e **y** ao longo da reta e determina os pontos mais próximos dessa linha no grid de pixels. Esse método é simples e intuitivo, mas pode ser menos eficiente em comparação com algoritmos como o de **Bresenham**, pois envolve operações de ponto flutuante para cálculos de coordenadas, o que pode afetar o desempenho, especialmente em sistemas com recursos limitados.

o pseudo-código usado como base foi:



Código utilizado:

```python
def dda(x1, y1, x2, y2, cor="black"):
    if x2 < x1:
        x1, x2, y1, y2 = x2, x1, y2, y1

    dx = x2 - x1
    dy = y2 - y1
    if (abs(dx) > abs(dy)):
        incremento = dy / dx
        y = y1
        for x in range(x1, x2 + 1):
            pintar(x, round(y),cor)
            y += incremento

    else:
        incremento = dx / dy
        x = x1
        for y in range(y1, y2 + 1):
            pintar(round(x), y,cor)
            x += incremento
```

1º Passo. Organiza os x na ordem correta.

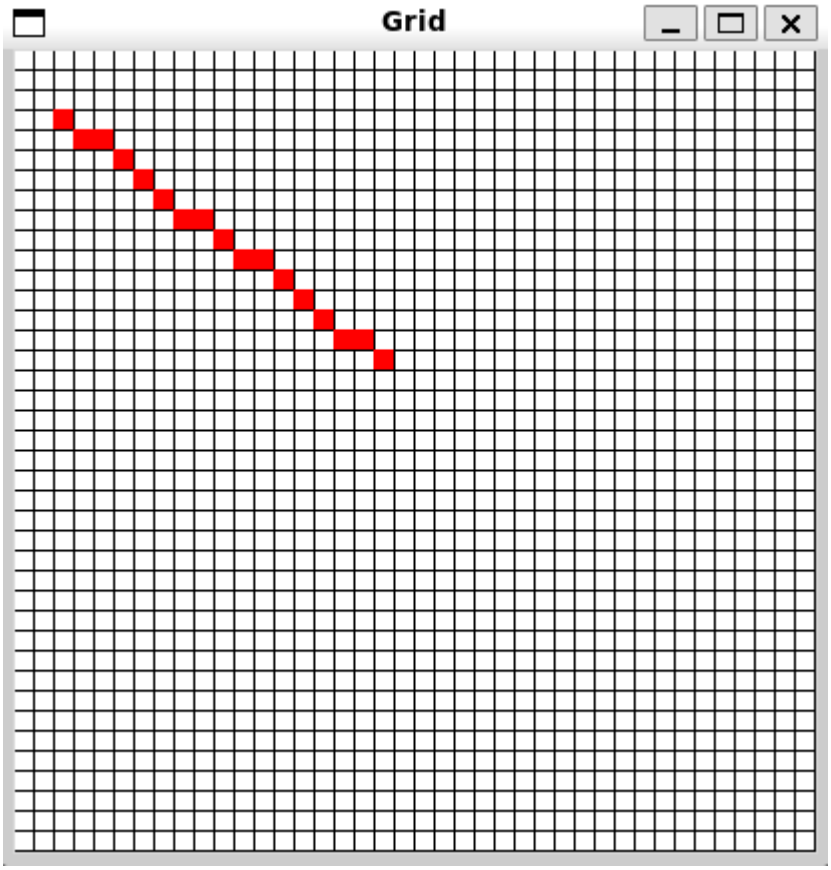
2º Passo. Checa delta x e y para saber se a linha é horizontal ou vertical.

3º Passo. Armazena o valor a incrementar e configura a coordenada inicial.

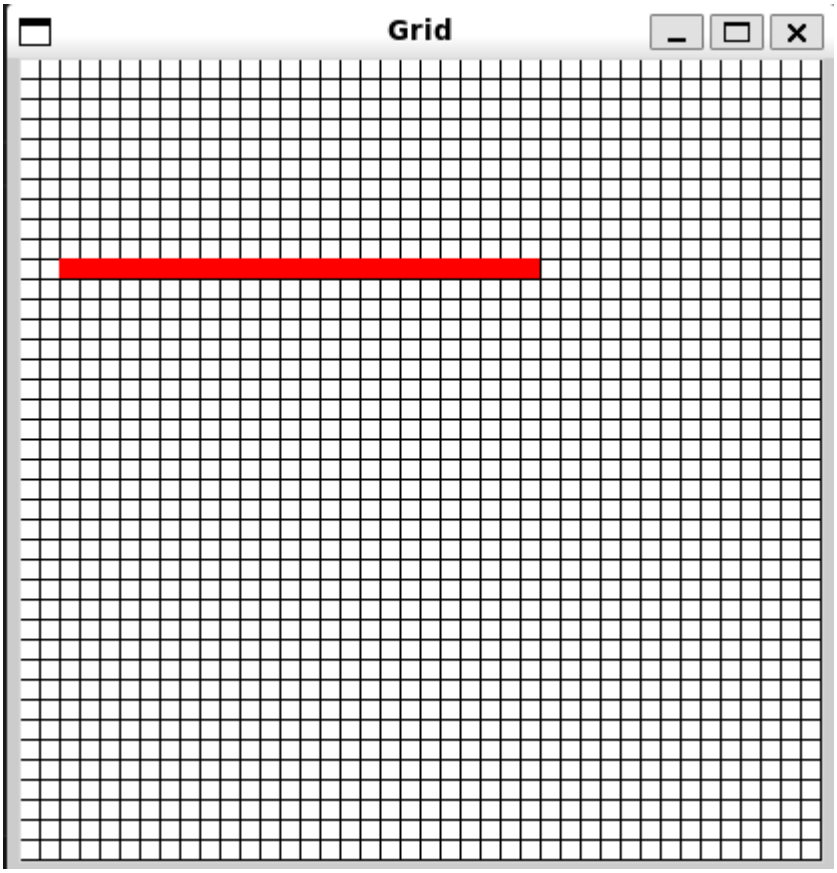
4º Passo. Pinta a coordenada e realiza o incremento.

Algumas demonstrações deste métodos são:

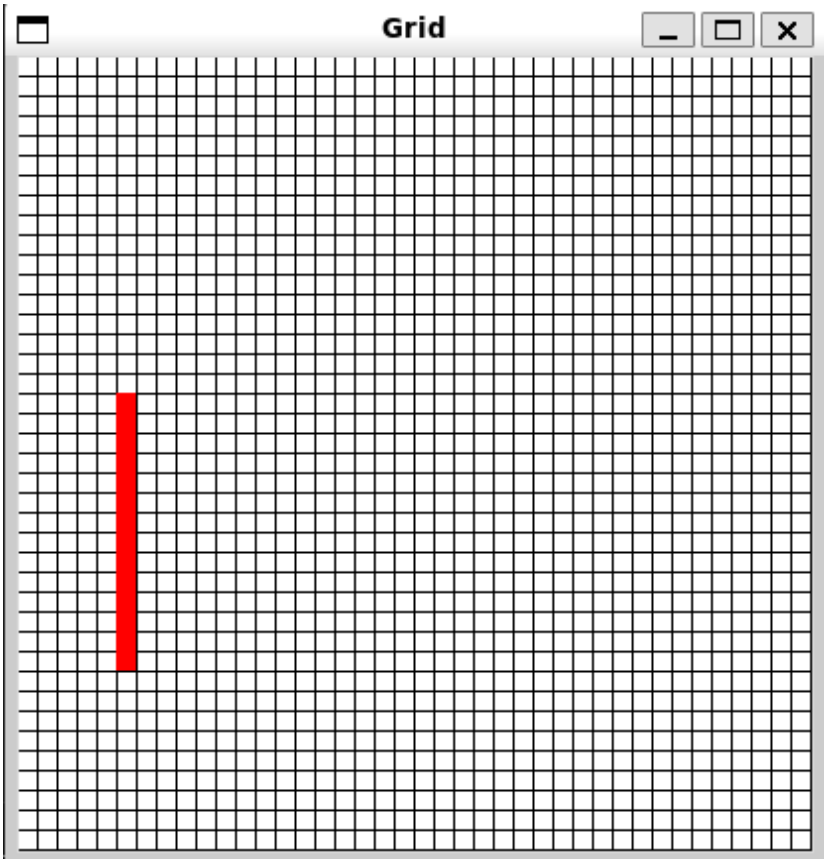
- dda(2, 3, 18, 15, "red"):



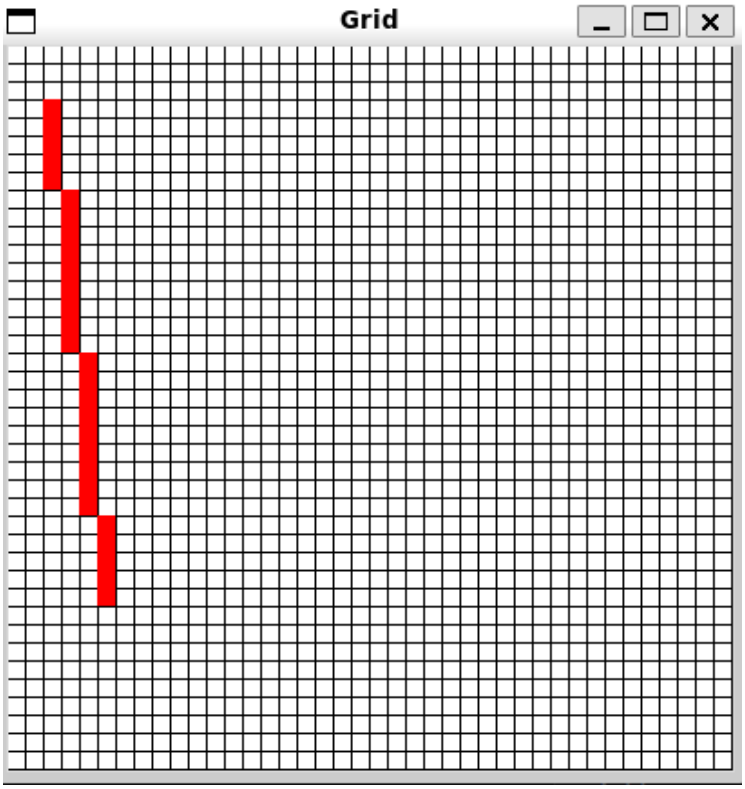
- dda(2, 10, 25, 10, "red"):



- dda(5, 17, 5, 30, "red"):



- dda(2, 3, 5, 30, "red"):

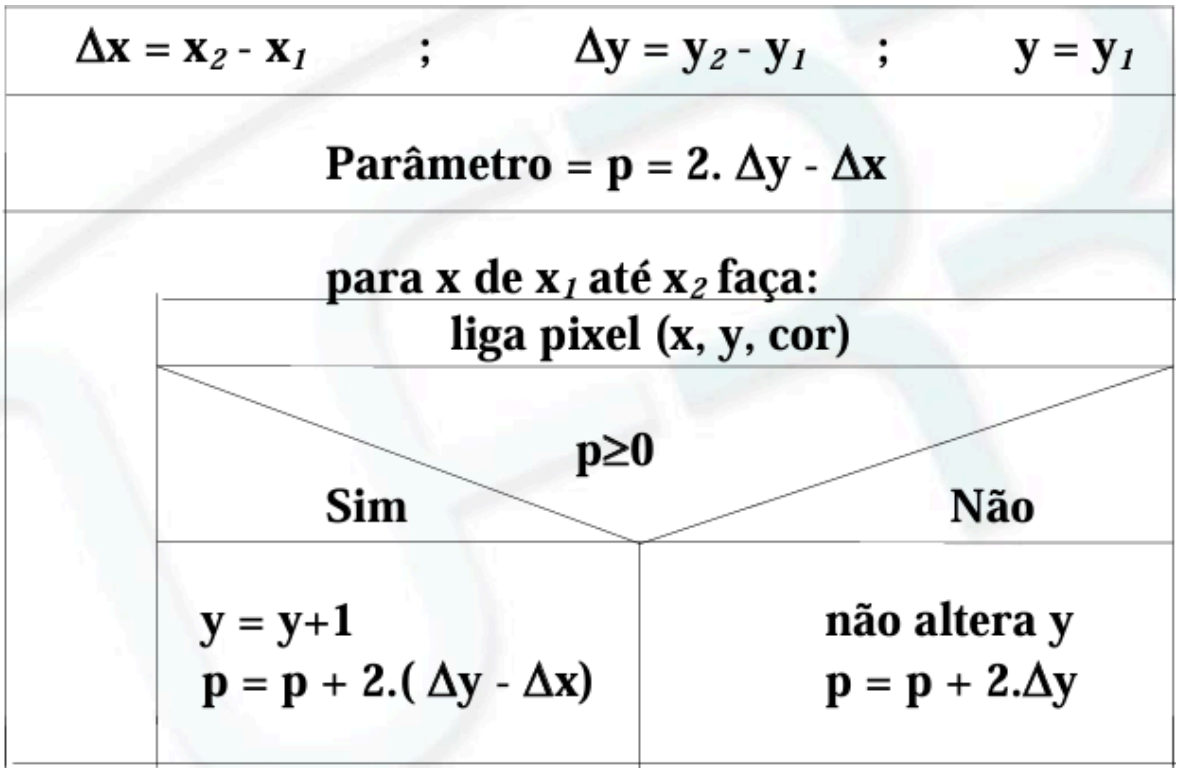


Método de Rasterização de linhas de Bresenham

O método de Bresenham para rasterização de linhas é um algoritmo eficiente utilizado para desenhar linhas retas em um grid de pixels. Ele foi desenvolvido para superar limitações dos métodos anteriores, como o DDA, ao evitar o uso de operações de ponto flutuante, utilizando apenas operações inteiras, o que melhora o desempenho, especialmente em sistemas com recursos limitados.

O algoritmo funciona determinando, a cada passo, o próximo pixel a ser ativado com base em uma decisão simples entre dois pixels possíveis, minimizando o erro de arredondamento. Ele decide qual dos dois pixels (acima ou ao lado do ponto atual) está mais próximo da linha ideal, garantindo que a linha desenhada seja o mais próxima possível da linha real, respeitando a precisão dos cálculos inteiros.

O pseudo-código usado como base foi:



Código utilizado:

```python
def bresenham(x1, y1, x2, y2, cor="black"):
    deltax = abs(x2 - x1)
    deltay = abs(y2 - y1)
    eixo = deltay > deltax

    if eixo:
        x1, y1 = y1, x1
        x2, y2 = y2, x2
        deltax, deltay = deltay, deltax

    if x1 > x2:
        x1, y1, x2, y2 = x2, y2, x1, y1

    p = 2 * deltay - deltax
    y = y1
    y_inc = 1 if y2 > y1 else -1

    for x in range(x1, x2 + 1):
        if eixo:
            pintar(y, x,cor)
        else:
            pintar(x, y,cor)
        if p >= 0:
            y += y_inc
            p -= 2 * deltax
        p += 2 * deltay
```

1º Passo. Calcula delta x e delta y.

2º Passo. Checa se a linha é horizontal ou vertical.

3º Passo. Reposiciona os valores menores e maiores.

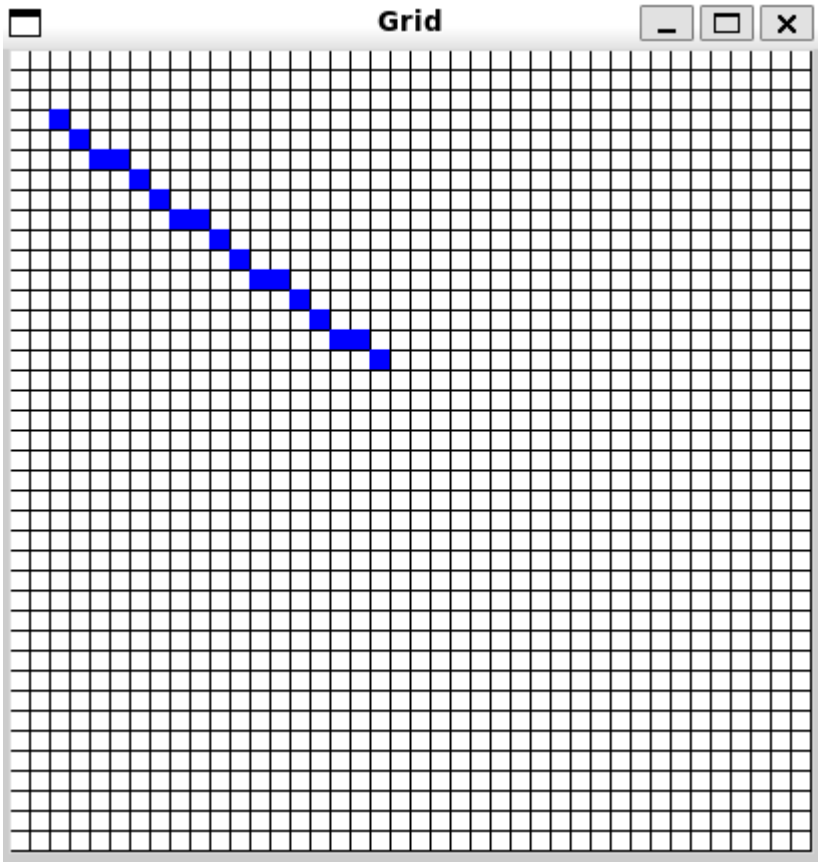
4º Passo. Checa se o incremento é positivo ou negativo.

5º Passo. Entra no loop e checa a condição de pintura.

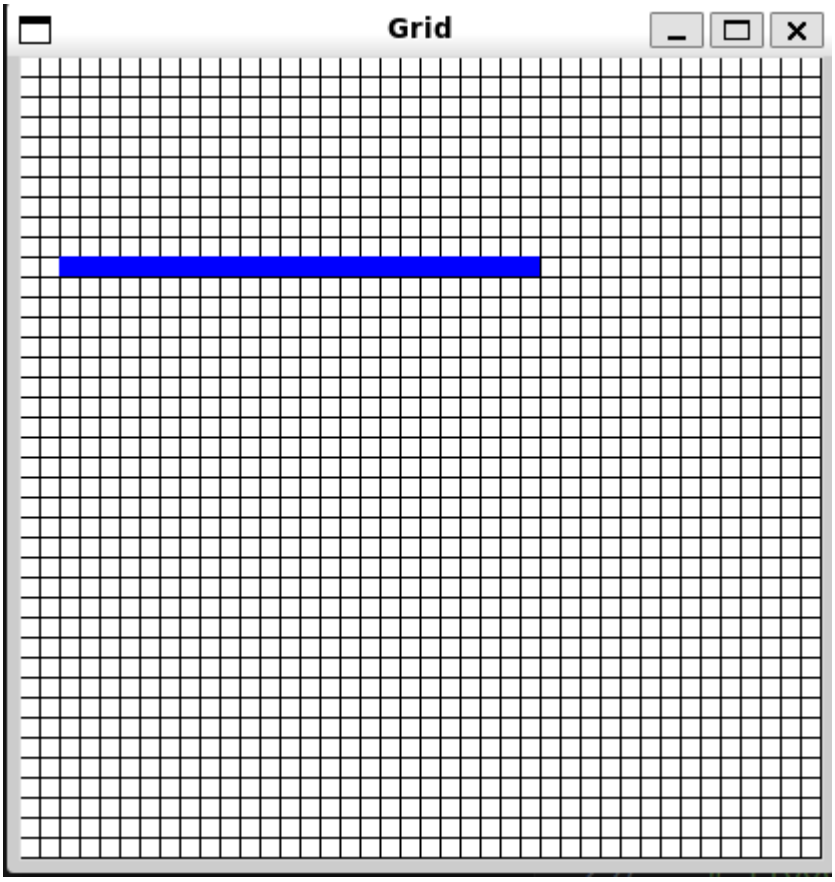
6º Passo. Faz o cálculo dos novos parâmetros.

Algumas demonstrações são:

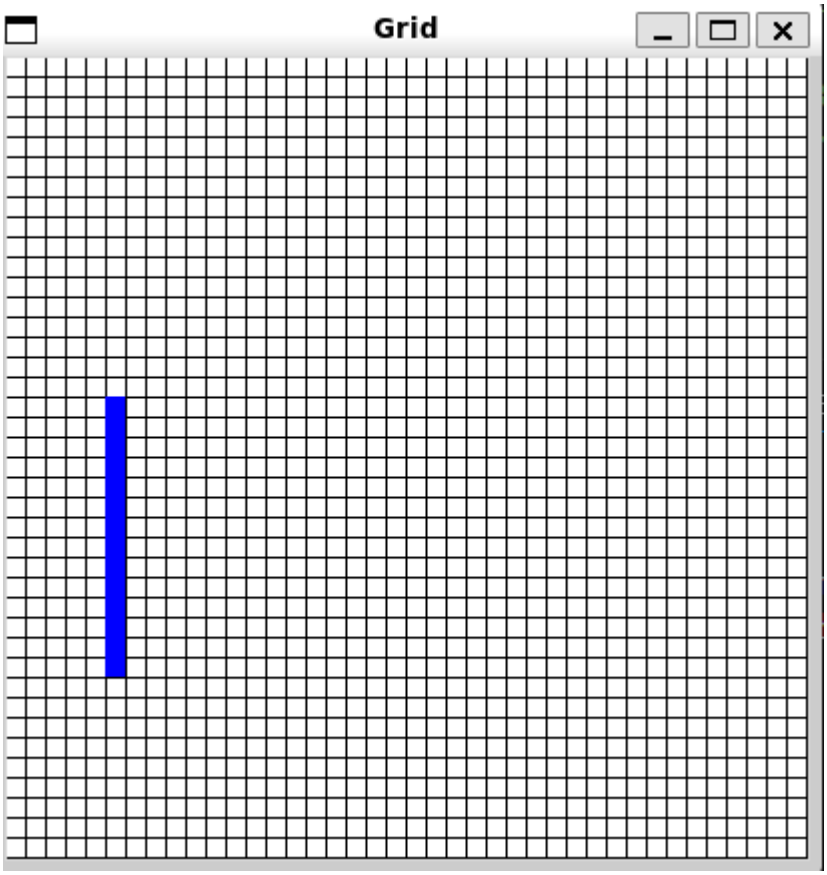
- bresenham(2, 3, 18, 15, "blue"):



- bresenham(2, 10, 25, 10, "blue"):



- bresenham(5, 17, 5, 30, "blue"):



- bresenham(2, 3, 5, 30, "blue"):