UNIVERSIDADE FEDERAL DE RORAIMA CENTRO DE CIÊNCIA E TECNOLOGIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
DCC703 - COMPUTAÇÃO GRÁFICA (2024.2)
Prof. LUCIANO FERREIRA SILVA

Nome: Marcia Gabrielle Bonifácio De Oliveira - 2020011319

TRABALHO 1

INTRODUÇÃO

Este trabalho apresenta diferentes algoritmos utilizados para a geração de retas em computação gráfica. Dentre os métodos abordados, estão o Método Analítico (também conhecido como Equação Paramétrica ou Equação da Reta) e o Algoritmo de Bresenham. Cada um desses métodos possui características específicas que os tornam mais ou menos eficientes, dependendo do contexto e dos requisitos de precisão e desempenho.

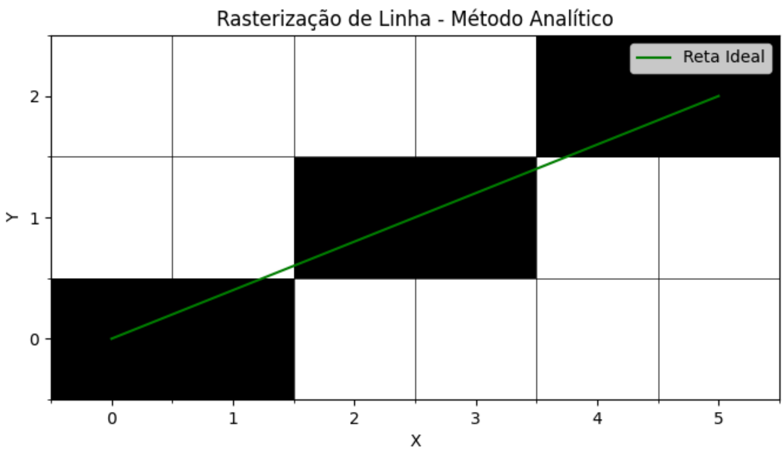
2. DESCRIÇÃO DOS ALGORITMOS

2.1 MÉTODO ANALÍTICO (OU EQUAÇÃO PARAMÉTRICA / EQUAÇÃO DA RETA)

O método analítico para rasterização de retas, também conhecido como método da equação da reta, baseia-se diretamente na fórmula $y=mx+b$. Aqui, m (inclinação) é calculado como $(y2-y1)/(x2-x1)$ e b (interseção em y) é obtido a partir de $b=y1-m \cdot x1$. Dessa forma, para cada valor de x que varia de x1 até x2, computamos y e, em seguida, arredondamos esse valor para o inteiro mais próximo, acendendo o pixel correspondente na tela.
Esse procedimento é muito simples e intuitivo, pois liga diretamente a matemática básica da reta à lógica de varredura dos pixels. No entanto, o método apresenta algumas desvantagens que devem ser consideradas. Uma delas é o tratamento especial necessário para o caso em que $x1=x2$, ou seja, a reta vertical, onde ocorre divisão por zero se não houver um desvio no código. Além disso, o uso de ponto flutuante (floats) em cada iteração gera arredondamentos constantes, o que pode introduzir leves imprecisões e aumentar o custo computacional, em comparação a algoritmos baseados exclusivamente em aritmética inteira (como Bresenham).
Abaixo apresentamos o código Python que implementa o método analítico de forma direta.

```python
def draw_line_analytic(x1, y1, x2, y2):
    # Calcula os parâmetros da equação da reta
    dx = x2 - x1
    dy = y2 - y1
    m = dy / dx  # Inclinação da linha
    b = y1 - m * x1  # Interseção com o eixo y

    # Lista de pixels resultantes
    pixels = []

    # Varre os valores de x e calcula y correspondente
    for x in range(x1, x2 + 1):
        y = m * x + b
        pixels.append((x, round(y)))

    return pixels
```

No gráfico resultante, é possível ver uma “escada” de quadrados pretos que representam os pixels efetivamente acesos pelo algoritmo, além de uma linha verde indicando a reta ideal em coordenadas contínuas. Essa diferença visual (entre a escada de pixels e a reta suave) reflete o fenômeno de rasterização, onde passamos de um mundo contínuo (fórmula matemática) para a discretização em pixels inteiros.



2.2 DDA (DIGITAL DIFFERENTIAL ANALYZER)

O método DDA (Digital Differential Analyzer) é uma forma de rasterizar retas usando incrementos proporcionais ao seu maior deslocamento. A ideia principal é determinar quantos passos (steps) serão necessários para percorrer o trecho de (x1,y1) até (x2,y2) e, a cada passo, atualizar as coordenadas x e y de maneira incremental.

Em primeiro lugar, calculam-se os deltas $\Delta x = x2 - x1$ e $\Delta y = y2 - y1$. Depois, define-se o número de passos (steps) como o maior valor entre $|\Delta x|$ e $|\Delta y|$. Isso significa que, se a reta for mais larga no sentido horizontal, haverá mais “subdivisões” em x, enquanto, se for mais inclinada no sentido vertical, os passos serão determinados por Δy.

Em seguida, determinam-se os incrementos em cada direção:

$x = \Delta x/steps$

$y = \Delta y/steps$

Esses incrementos são então somados às variáveis x e y a cada passo no loop, acumulando o deslocamento gradualmente até chegar ao fim da reta. Após cada atualização, arredonda-se o valor de x e y para o pixel mais próximo e acende-se esse pixel.

```python
def draw_line_dda(x1, y1, x2, y2):
    # Calcula os deltas
    dx = x2 - x1
    dy = y2 - y1

    # Determina o número de passos
    steps = int(max(abs(dx), abs(dy)))  # O maior delta determina os passos

    # Calcula os incrementos em cada direção
    x_inc = dx / steps
    y_inc = dy / steps

    # Lista para armazenar os pixels
    pixels = []

    # Valores iniciais
    x, y = x1, y1

    # Itera para determinar os pixels
    for _ in range(steps + 1):
        pixels.append((round(x), round(y)))
        x += x_inc
        y += y_inc

    return pixels
```

2.3 BRESENHAM

O algoritmo de Bresenham é amplamente utilizado na rasterização de retas, sendo famoso por trabalhar exclusivamente com aritmética de inteiros. Graças a essa característica, ele se mostra particularmente rápido e eficiente, tendo sido muito empregado em implementações de hardware gráfico mais antigas (quando a performance de operações em ponto flutuante era limitada). Mesmo nos dias de hoje, Bresenham permanece relevante por sua precisão e baixo custo computacional.

Em termos de funcionamento, o algoritmo evita a necessidade de dividir ou multiplicar por valores não inteiros. Em vez disso, ele mantém uma variável de erro acumulado (chamada aqui de d), que indica o quão "distante" estamos de precisar subir ou descer um pixel na direção vertical. A cada iteração, é feita uma verificação nesse erro para decidir se o ponto (x,y) deve ser incrementado apenas em x ou em x e y simultaneamente.

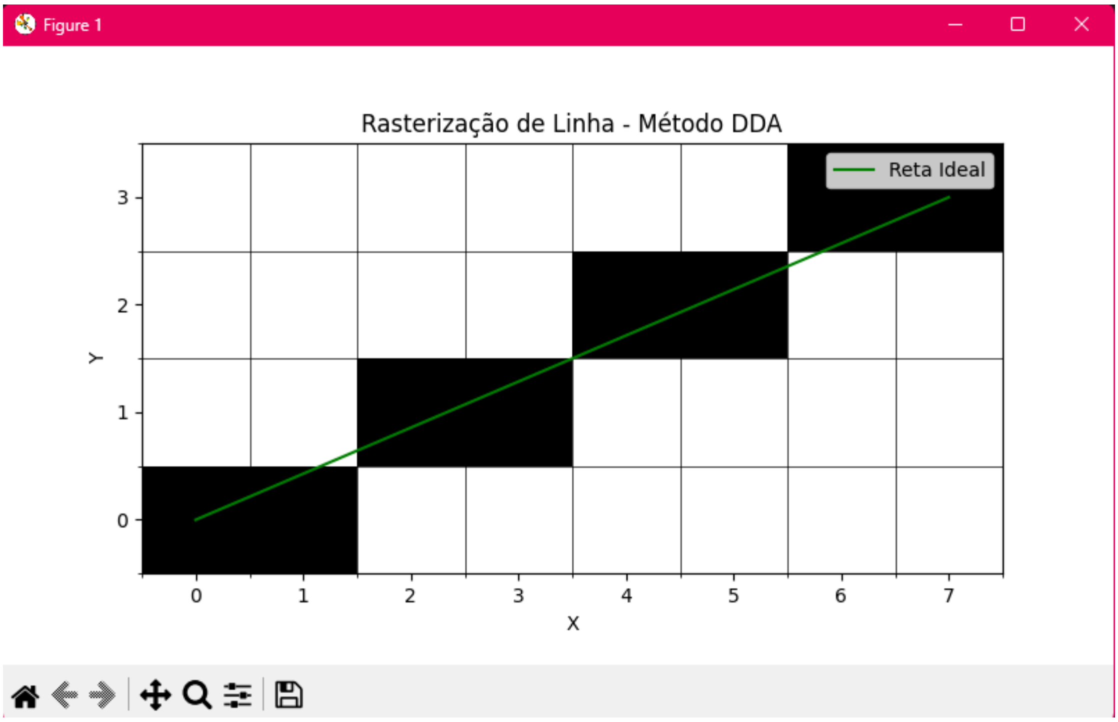
O "erro" se atualiza de forma incremental: quando o ponto fica abaixo da linha ideal, adicionamos incrE ($2 \cdot \Delta y$) a d; se a linha ideal já ultrapassou o "meio" entre os pixels, então somamos increNE ($2 \cdot (\Delta y - \Delta x)$) e também incrementamos o valor de y, pois precisamos "subir" no grid para acompanhar a reta. Esse processo permite que a escolha do próximo pixel seja feita sem operações de ponto flutuante, apenas usando somas e subtrações de valores inteiros.

```python
def bresenham_line(x0, y0, x1, y1):
    pixels = []

    dx = x1 - x0
    dy = y1 - y0
    d = 2 * dy - dx
    incrE = 2 * dy
    incrNE = 2 * (dy - dx)

    x, y = x0, y0
    pixels.append((x, y))

    while x < x1:
        if d <= 0:
            d += incrE
            x += 1
        else:
            d += incrNE
            x += 1
            y += 1
        pixels.append((x, y))

    return pixels
```

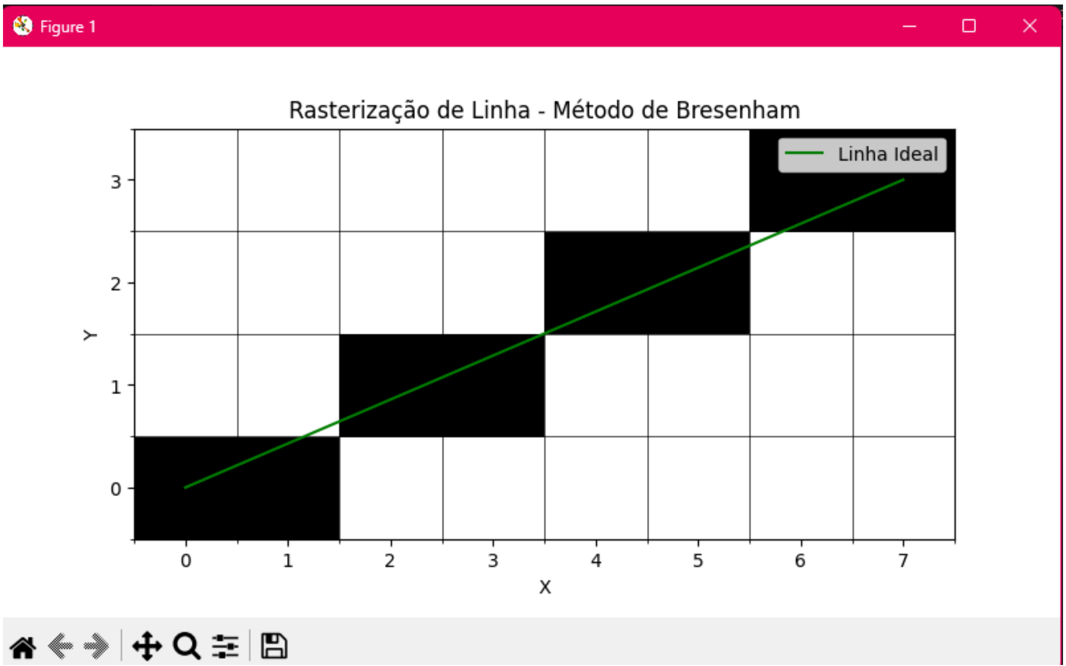
3. COMPARAÇÃO DOS ALGORITMOS

- **Exemplo 1: Reta com inclinação pequena (0,0)→(7,3).**
    - *Imagem do Algoritmo Analítico*

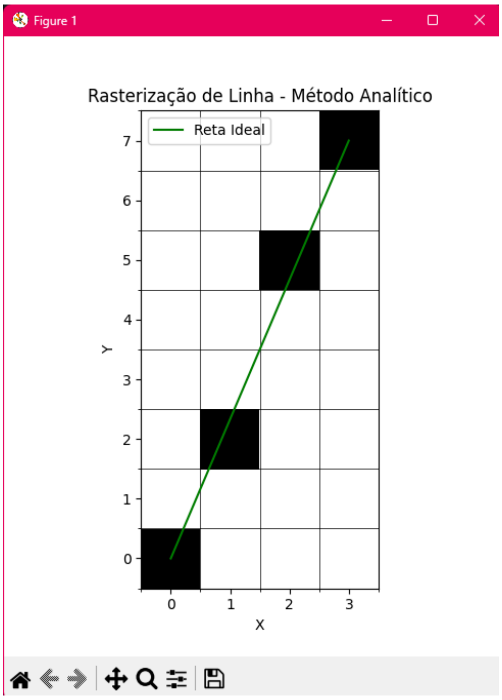
- ***Imagem do Algoritmo DDA***

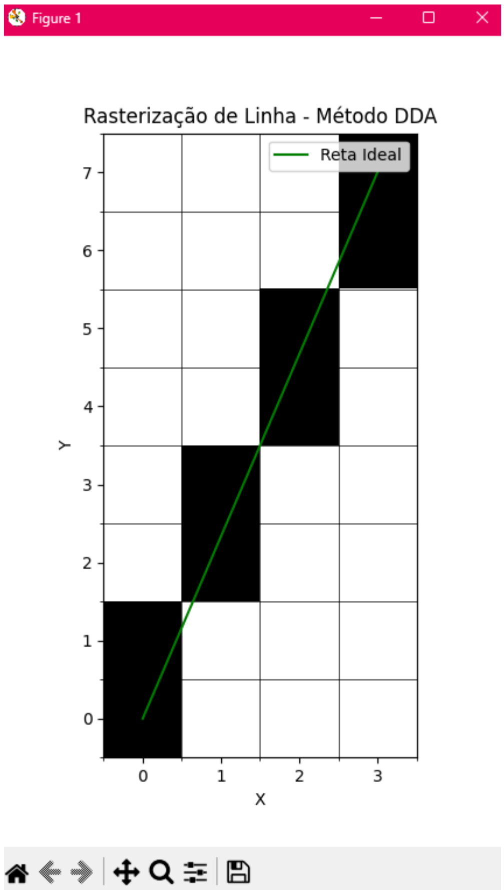


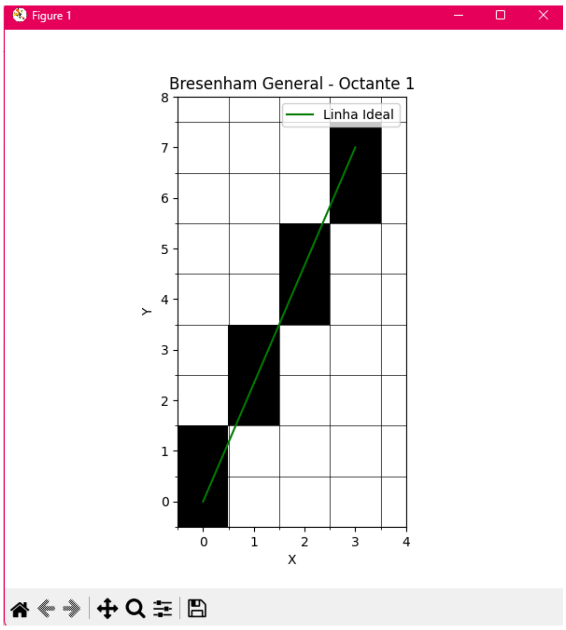
- ***Imagem do Algoritmo Bresenham***



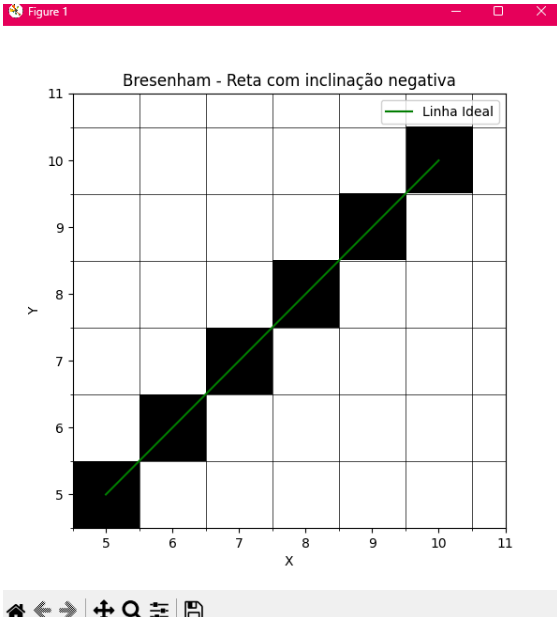
- **Exemplo 2: Reta com inclinação > 1 (0,0)→(3,7)).**



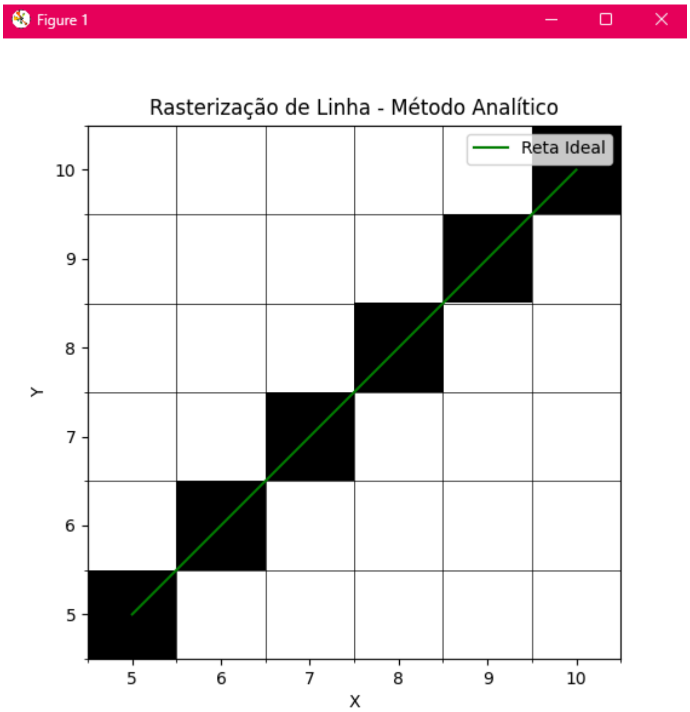




- **Exemplo 3: Reta com inclinação negativa (10,10)→(5,5).**



-



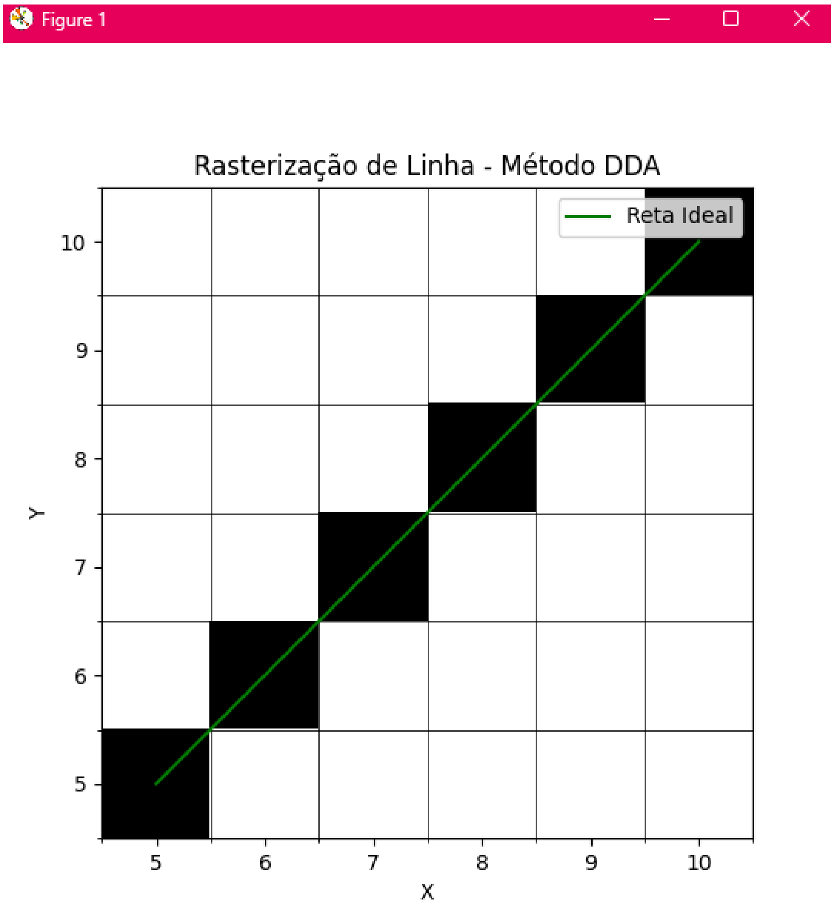


Figure 1
Rasterização de Linha - Método DDA
Reta Ideal
10
9
8
7
6
5
Y
5
6
7
8
9
10
X