Rasterização de Retas

DCC703 - Computação Gráfica (2024.2)
Prof. - Luciano Ferreira Silva
Aluno - Paulo Ferreira da Silva Júnior - 2019034400

Relatório de Rasterização de Retas

Introdução

Este relatório descreve a implementação e os resultados de três algoritmos de rasterização de retas: **Método Analítico**, **DDA (Digital Differential Analyzer)** e **Bresenham**. O objetivo é comparar as técnicas em termos de precisão, eficiência e comportamento visual das retas rasterizadas.

---

1. Método Analítico

Descrição do Algoritmo

O método analítico utiliza a equação da reta na forma:

$$y = m \cdot x + b$$

Onde:

- m é o coeficiente angular da reta, calculado como .

  $$m = \Delta y - 2\Delta y1 \setminus \Delta x - 2\Delta x1$$

- b é o termo independente, determinado como .

  $$b = y1 - m \cdot x1$$

O algoritmo percorre os valores de x e calcula os respectivos y, arredondando-os para valores inteiros para determinar os pixels ativados. Os pixels resultantes são armazenados em uma lista e posteriormente usados para rasterização.

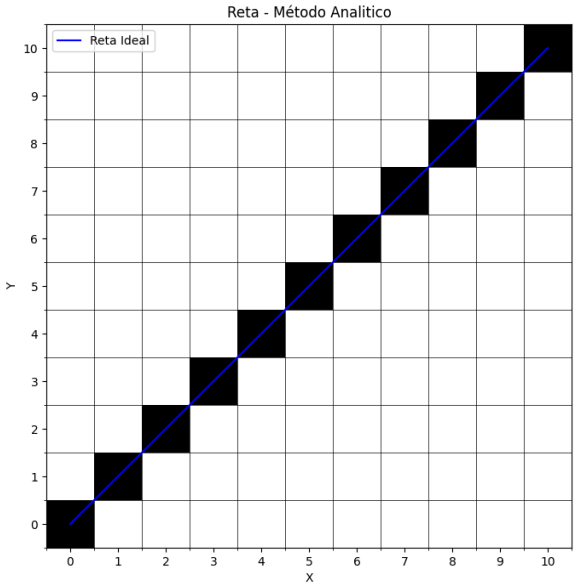
Código

```
import matplotlib.pyplot as plt
import numpy as np

def draw_line_analytical(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    m = dy / dx
    b = y1 - m * x1

    points = []
    x = x1
    while x <= x2:
        y = round(m * x + b)
        points.append((x, y))
        x += 1
    return points
```

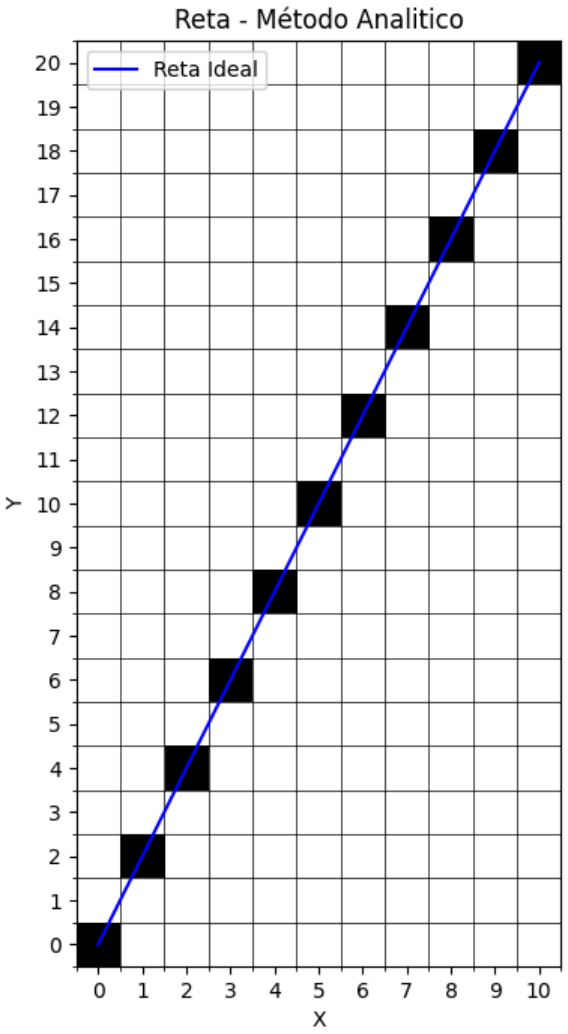
Observações

- **Precisão**: O método gera bons resultados para inclinações suaves, mas pode apresentar erros em inclinações acentuadas devido ao arredondamento.
- **Eficiência**: Calcula diretamente os valores de y , mas depende de operações de ponto flutuante.

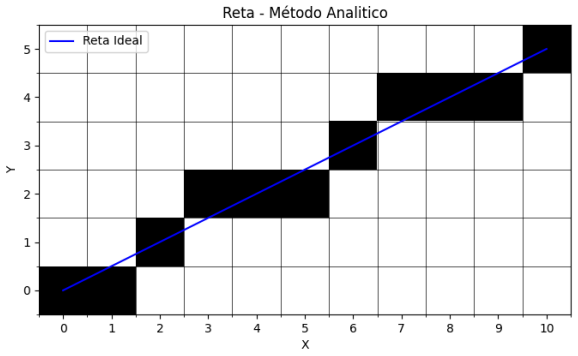
Imagem do Resultado



x2 = 10
y2 = 10
m = 1



x2 = 10
y2 = 20
m = 2



x2 = 10
y2 = 5
m = 0.5

2. DDA (Digital Differential Analyzer)

Descrição do Algoritmo

O método DDA utiliza incrementos constantes para rasterizar a reta. Calcula-se o número total de passos como:

steps=max(Δx,Δy) \ steps

Os incrementos em cada direção são calculados como:

- x_inc=Δx \ steps
- y_inc=Δy \ steps

Os valores de x e y são atualizados iterativamente e arredondados para determinar os pixels ativados.

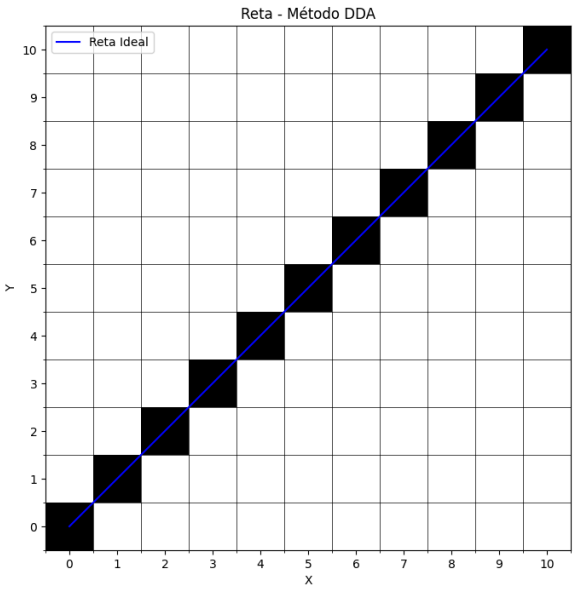
Código

```
import matplotlib.pyplot as plt
import numpy as np

def draw_line_dda(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1

    steps = max(abs(dx), abs(dy))
    x_inc = dx / steps
    y_inc = dy / steps

    x, y = x1, y1
    points = [(round(x), round(y))]

    for _ in range(int(steps)):
        x += x_inc
        y += y_inc
        points.append((round(x), round(y)))

    return points
```

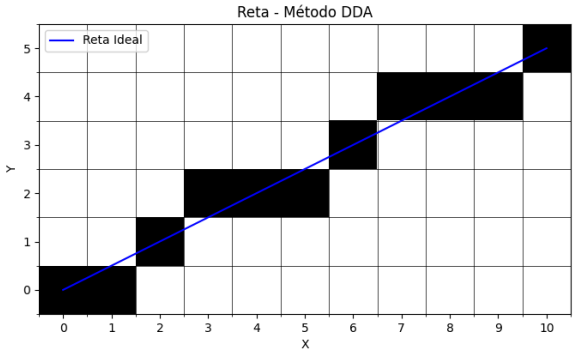
Observações

- **Precisão**: Melhor que o método analítico para inclinações acentuadas.
- **Eficiência**: Leve vantagem sobre o método analítico, pois utiliza incrementos fixos.

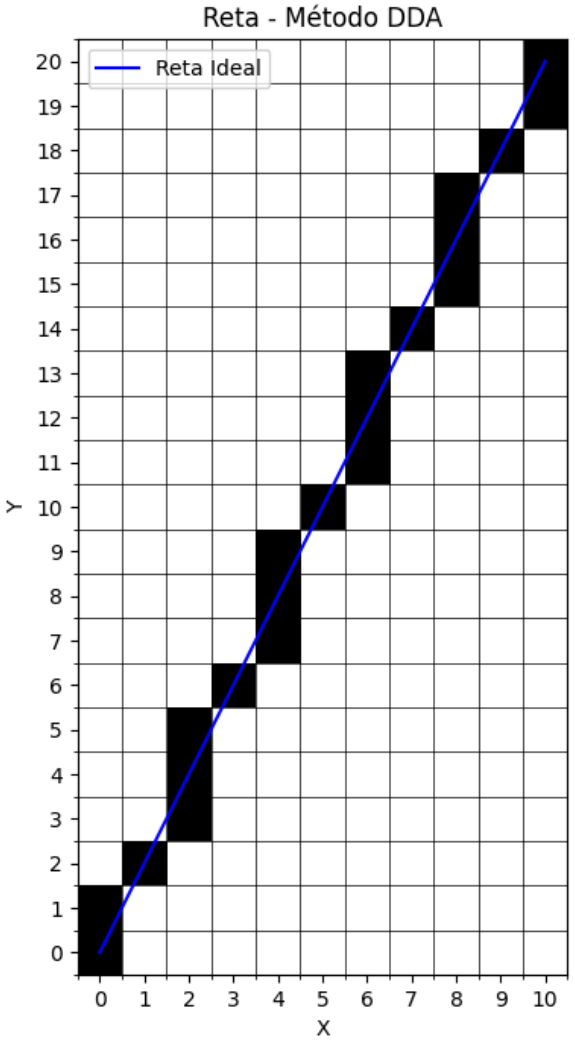
Imagem do Resultado



x2 = 10
y2 = 10
m = 1



x2 = 10
y2 = 5
m = 0.5



x2 = 10
y2 = 20
m = 2

3. Bresenham

Descrição do Algoritmo

O algoritmo de Bresenham é baseado em incrementos inteiros e evita o uso de ponto flutuante. Uma função de decisão determina o próximo pixel a ser ativado:

$d=2\cdot\Delta y-\Delta x$

Dependendo do valor de d, o algoritmo ajusta os valores de x e y.

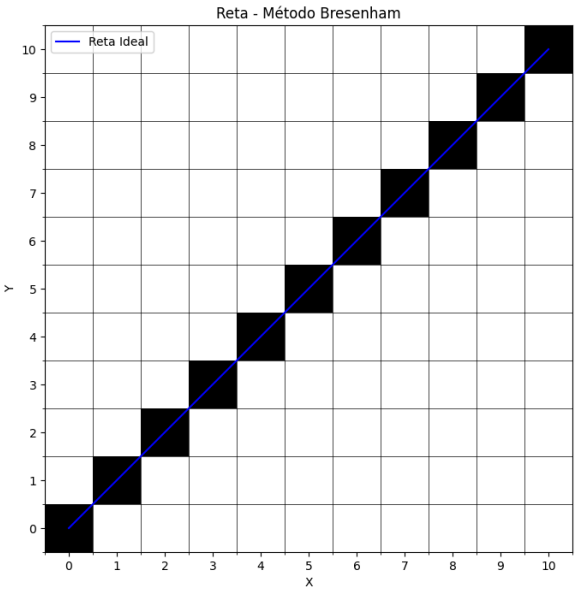
Código

```
import matplotlib.pyplot as plt
import numpy as np

def draw_line_bresenham(x1, y1, x2, y2):
    dx = abs(x2 - x1)
    dy = abs(y2 - y1)
    sx = 1 if x1 < x2 else -1
    sy = 1 if y1 < y2 else -1

    err = dx - dy
    points = []

    while True:
        points.append((x1, y1))
        if x1 == x2 and y1 == y2:
            break
        e2 = 2 * err
        if e2 > -dy:
            err -= dy
            x1 += sx
        if e2 < dx:
            err += dx
            y1 += sy

    return points
```

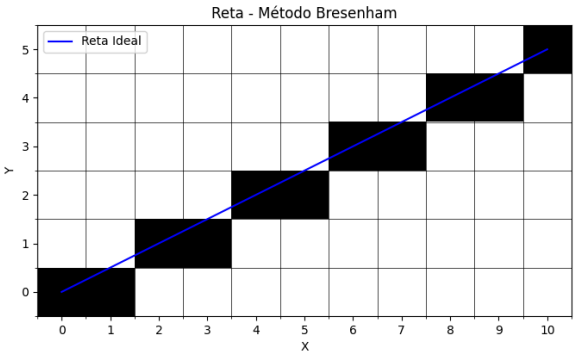
Observações

- **Precisão**: Excelente para todos os tipos de inclinação, com pixels mais consistentes.
- **Eficiência**: Altamente eficiente devido ao uso de incrementos inteiros e decisão simples.

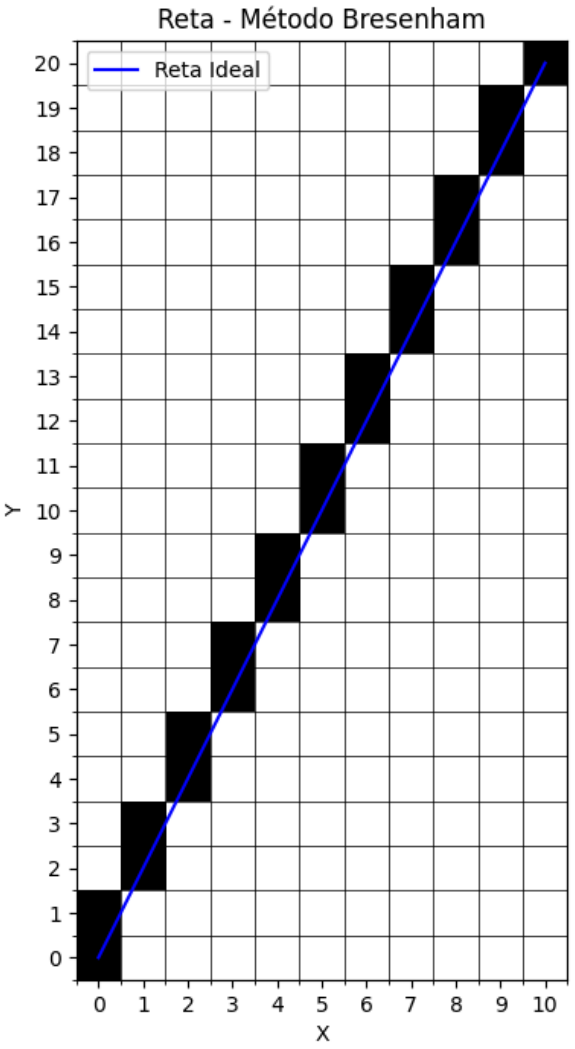
Imagem do Resultado



x2 = 10
y2 = 10
m = 1



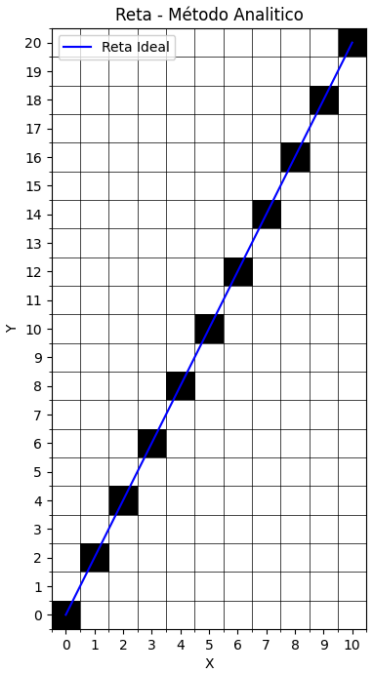
x2 = 10
y2 = 5
m = 0.5



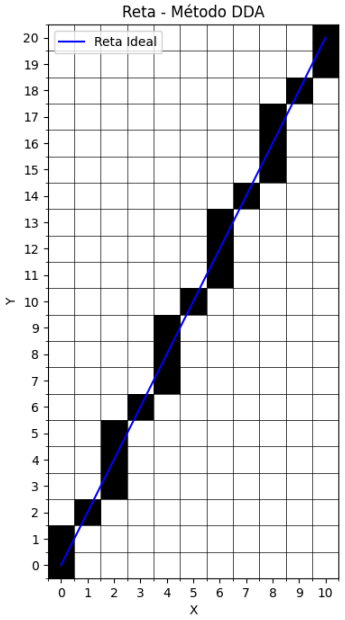
x2 = 10
y2 = 20
m = 2

Comparativo Geral

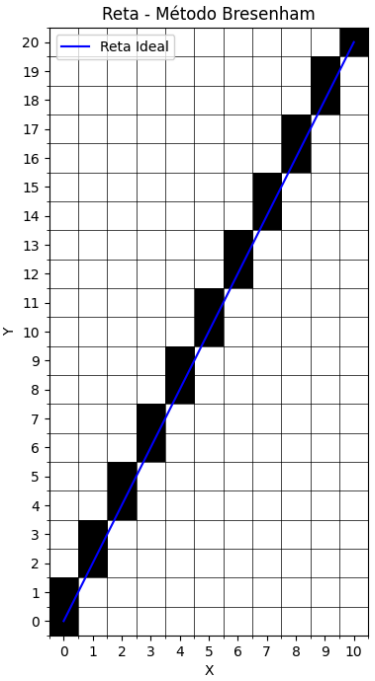
Método	Precisão	Eficiência	Complexidade
Analítico	Boa, mas com erros em inclinações acentuadas	Moderada	Simples
DDA	Melhor para inclinações variadas	Boa	Moderada
Bresenham	Excelente para todas as inclinações	Excelente	Alta



Analitico com inclinação acentuada



DDAcom inclinação acentuada



Bresenham com inclinação acentuada

Observações Finais

- O algoritmo de **Bresenham** se destaca em termos de precisão e eficiência, sendo ideal para aplicações práticas.
- O **DDA** oferece um bom compromisso entre simplicidade e precisão.
- O **método analítico** é mais simples, mas menos eficiente para inclinações extremas.