# Trabalho de Rasterização de Linhas

Aluno: Rosialdo Vicente

N° de Matricula: 2020018122

## Objetivo do Programa

### Desenvolver um programa que permita desenhar retas utilizando os algoritmos:

- Analítico
- DDA (Digital Differential Analyzer)
- Bresenham

## Tecnologias Utilizadas

Para a implementação deste trabalho, optei pela linguagem Python devido à minha familiaridade e à sua flexibilidade para desenvolvimento de aplicações gráficas.

### Linguagem de Programação

Para a implementação deste trabalho, optei pela linguagem `Python` devido à minha familiaridade e à sua flexibilidade para desenvolvimento de aplicações gráficas.

### Bibliotecas Utilizadas

As seguintes bibliotecas foram utilizadas no projeto:

- `sys`: Utilizada para encerrar o programa corretamente.
- `pygame`: Escolhida para visualização gráfica das retas devido à sua facilidade de uso e capacidade de manipulação de gráficos em tempo real.

## Implementação dos Algoritmos

Os algoritmos foram implementados em um único arquivo chamado `linhas.py` para facilitar o gerenciamento e compreensão do código.

## Algoritmo Analítico

Este método utiliza uma abordagem simples e intuitiva para rasterização de linhas. Ele calcula a equação da reta com base em dois vetores de pontos e, em seguida, varia x de unidade em unidade. Utilizando a equação da reta, ele determina o valor de Y por meio de um arredondamento.

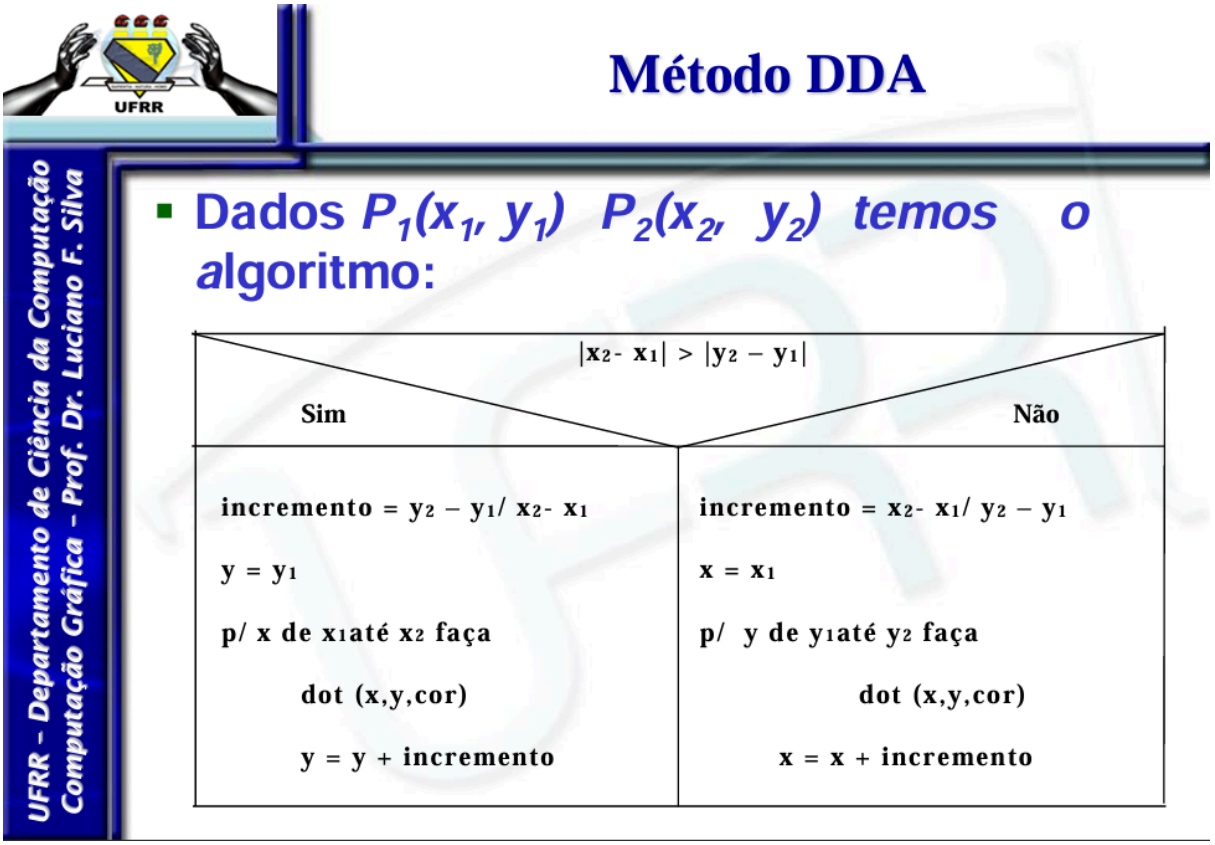
## Referência para o Algoritmo Analítico:

| % reta vertical X1 = X2 | |
|---|---|
| Não | Sim |
| m = (Y2 - Y1)/(X2 - X1)<br>b = Y2 - m*X2<br>para X de X1 até X2<br>  Y = m*X + b<br>  liga_pixel(X, Y, Cor) | para Y de Y1 até Y2<br>  liga_pixel(X1, Y, Cor); |

## Algoritmo DDA

O Algoritmo DDA (Digital Differential Analyzer) é similar ao Analítico a mudança que ele faz é no calculo, onde leva-se em consideração os ângulos para saber se deve-se fazer um incremento no valor de X ou de Y

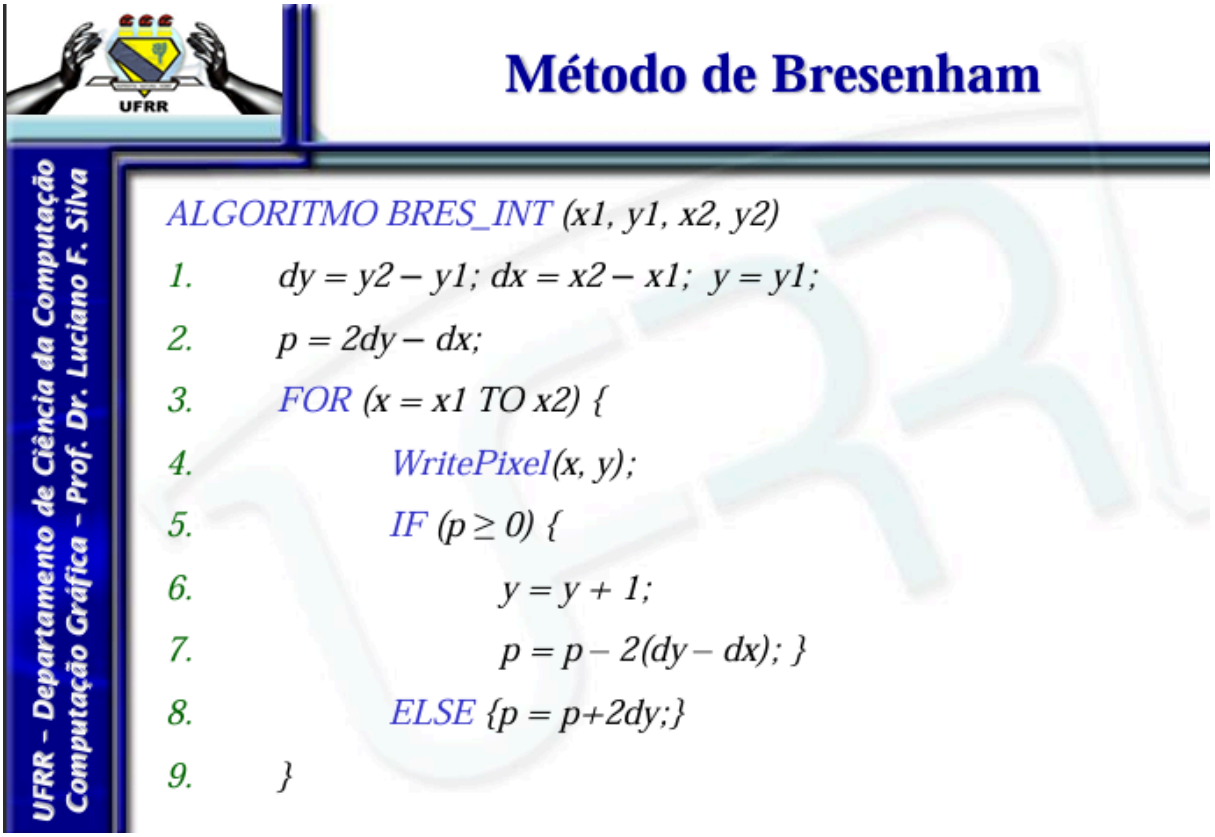
## Referência para o Algoritmo DDA:




## Algoritmo Bresenham

No Algoritmo de Bresenham evita operações com números de ponto flutuante. Ele decide, a cada passo, se o próximo pixel a ser desenhado estará na mesma linha horizontal ou se deve ser deslocado para cima, com base em um parâmetro de decisão incremental. A ideia central é avaliar a posição da linha em relação ao ponto médio entre dois pixels candidatos e escolher aquele mais próximo da reta ideal.

## Referência para o Algoritmo Bresenham :




## Desenvolvimento

Durante a implementação, encontrei dificuldades significativas ao desenvolver os algoritmos DDA e Bresenham. Após estudar os pseudocódigos fornecidos em aula, consegui superar esses obstáculos. Também enfrentei desafios na visualização das linhas, inicialmente tentando usar a biblioteca `matplotlib`. Optei então pelo `pygame`, após considerar sua adequação para demonstrações gráficas em tempo real, após conversas com colegas.

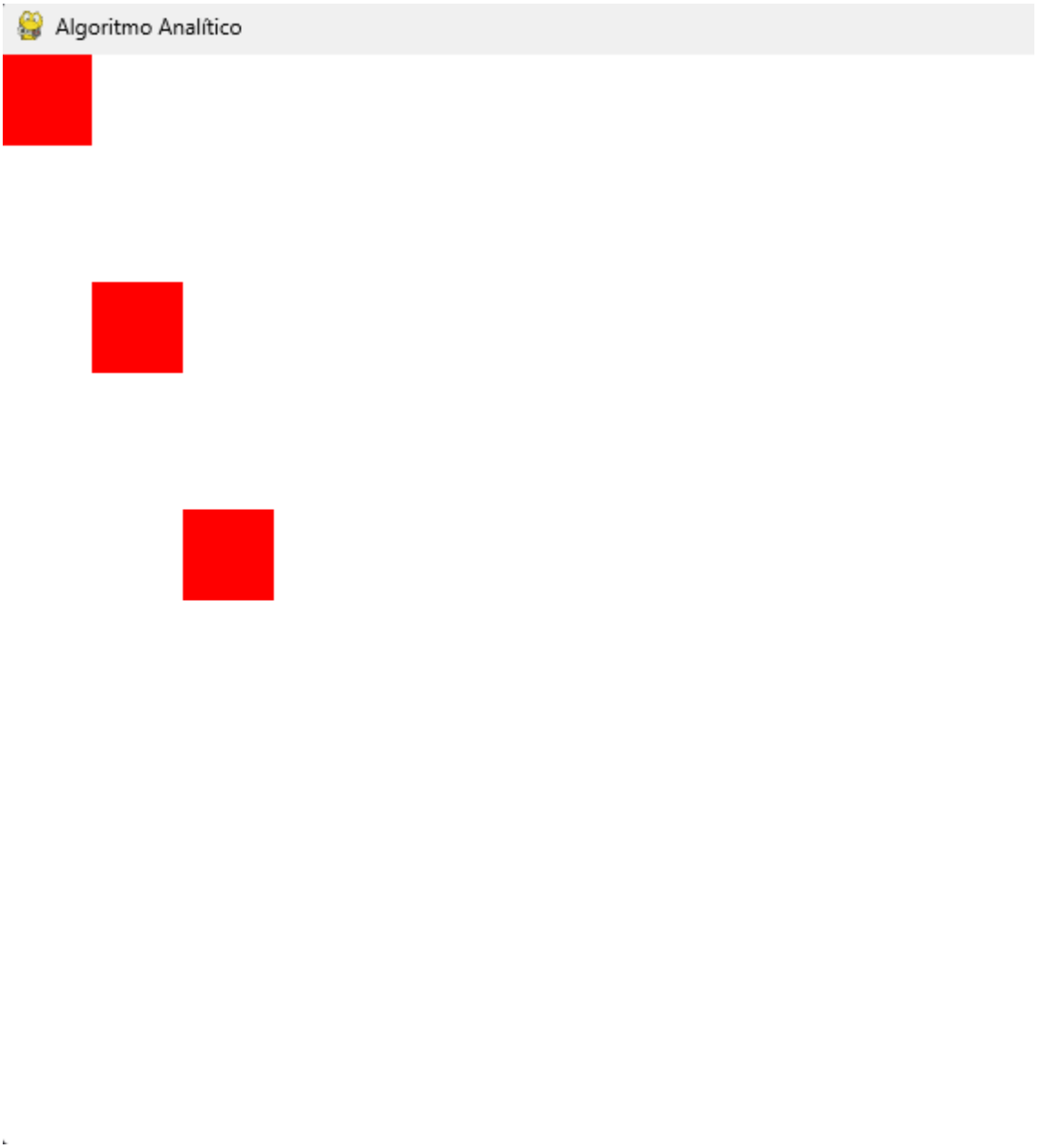
A implementação da visualização das linhas não correspondeu completamente às minhas expectativas iniciais, mas acredito que tenha sido eficaz para exemplificar os cálculos dos algoritmos.

## Testes

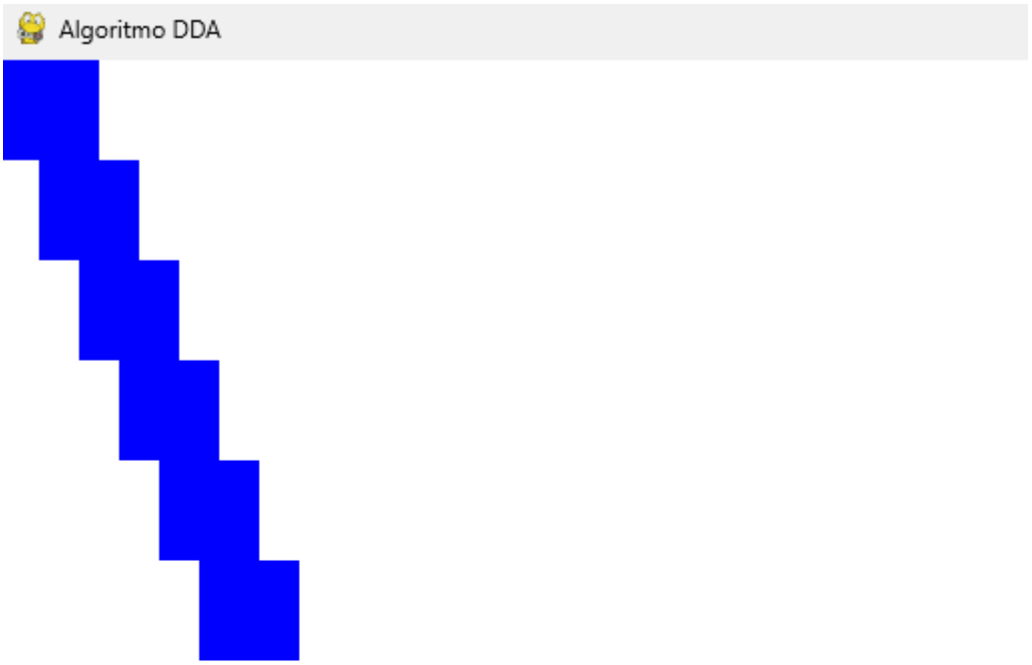
Todos os testes foram feitos com os seguintes valores:

`P1(X1,Y1) e P2(X2,Y2) = P1 (0,0) e P2 (2,5)`

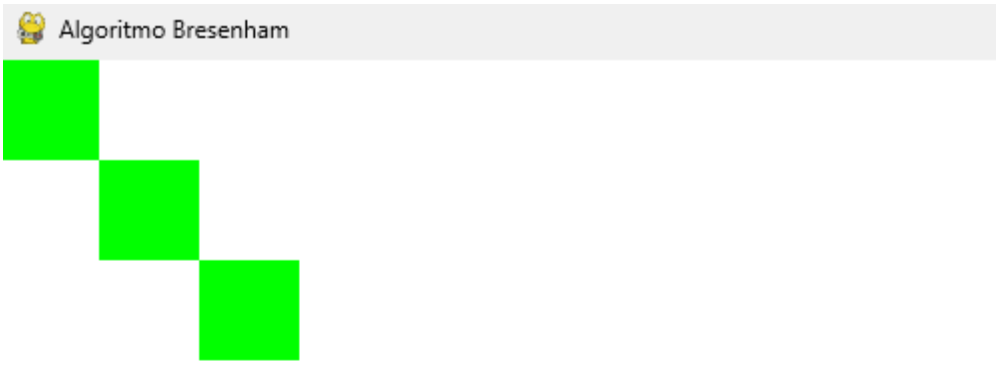
## Teste Algoritmo Analítico




## Teste Algoritmo DDA




## Teste Algoritmo Bresenham




## Comparando os Algoritmos

Durante os testes realizados, foi difícil perceber diferenças nos tempos de execução devido aos números pequenos utilizados. No entanto, mesmo com esses valores reduzidos, foi possível notar que o método analítico apresenta problemas quando X2 é menor que Y2. Isso ocorre porque a forma como ele calcula a ligação entre os pixels pode gerar lacunas.

Para solucionar esse problema, foi proposto o algoritmo DDA, que evita essas falhas ajustando a variação em X ou Y conforme a inclinação da reta. No entanto, ele ainda apresenta algumas limitações, como o uso de aritmética de ponto flutuante, possíveis erros de arredondamento e maior tempo de execução em escalas maiores.

Já o algoritmo de Bresenham se destaca pela eficiência no tempo de execução, pois evita operações com ponto flutuante e arredondamentos, utilizando apenas soma, subtração e deslocamento de bits, tornando-se uma solução mais rápida e precisa.