UNIVERSIDADE FEDERAL DE RORAIMA
CENTRO DE CIÊNCIA E TECNOLOGIA - CCT
CURSO DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO

COMPUTAÇÃO GRÁFICA

RASTERIZAÇÃO DE LINHAS

HENDRICK SILVA FERREIRA-2020026830

Janeiro de 2025

Boa Vista/Roraima

Resumo

Este relatório fala a construção de programas Python usando algoritmos de rasterização de três linhas: análise, DDA e Bresenham. Cada programa é desenvolvido através de etapas comuns, como configuração do ambiente, implementação de algoritmos específicos e controle de eventos. Os resultados dos algoritmos são comparados, destacando suas características únicas.

Conteúdo

Figuras

1 Introdução

Este relatório descreve a construção de três programas em Python para rasterização de linhas utilizando três algoritmos diferentes: Analítico, DDA e Bresenham. O objetivo é comparar e contrastar esses algoritmos quanto à sua eficiência e precisão na rasterização de linhas.

1.1 Algoritmo Analítico

O algoritmo analítico para rasterização de linhas é baseado na equação da reta, que é uma representação matemática da linha a ser desenhada. Ele calcula os incrementos para x e y com base nas coordenadas dos pontos de partida e destino da linha. O algoritmo segue a ideia de que, ao longo da linha, a variação de x e y é linear e constante. Ele é preciso na rasterização, mas pode ser menos eficiente em comparação com outros algoritmos, especialmente quando se lida com linhas com inclinações acentuadas.

1.2 Algoritmo DDA

O algoritmo DDA, ou Digital Differential Analyzer, é um algoritmo incremental usado para rasterizar linhas. Ele calcula os incrementos de x e y com base nas diferenças de coordenadas entre os pontos de partida e destino da linha. O DDA é eficiente e, em muitos casos, tão preciso quanto o algoritmo analítico. No entanto, ele evita cálculos de ponto flutuante, o que o torna mais eficiente em termos de desempenho.

1.3 Algoritmo Bresenham

O algoritmo de Bresenham é um dos algoritmos mais conhecidos e amplamente utilizados para rasterização de linhas. Ele é altamente eficiente e é especialmente adequado para sistemas com recursos limitados. O algoritmo de Bresenham mantém a precisão ao desenhar linhas, usando um cálculo incremental inteligente para determinar quais pixels devem ser incluídos na linha. Sua eficiência torna-o uma escolha preferida em muitos aplicativos gráficos e sistemas embarcados.

2 Construção dos Programas

A construção dos programas que utilizam os algoritmos de rasterização de linhas - Analítico, DDA e Bresenham - envolve um conjunto comum de etapas. Que são:

2.1 Configuração do Ambiente

A primeira coisa feita foi configurar o ambiente de desenvolvimento, no qual é a instalação de bibliotecas necessárias, como o pygame, que é usado para a exibição gráfica. Isso também envolve a configuração da janela de exibição, definição do título e dimensionamento da tela.

2.2 Entrada dos Pontos de Início e Destino

Optei em dar pontos de início e destino da linha a ser rasterizada já especificados, mas, essas coordenadas são variáveis que podem ser ajustadas conforme necessário.

2.3 Loop Principal

O loop principal foi usado para controlar a execução do programa. Ele normalmente inclui a manipulação de eventos, como o evento de fechamento da janela (pygame.QUIT), o preenchimento da tela com a cor de fundo desejada (que foi o preto) e a chamada da função de rasterização.

3 Descrição dos Programas

3.1 Algoritmo Analítico

O primeiro programa foi feito com o algoritmo analítico para rasterizar uma linha entre dois pontos. O algoritmo analítico é baseado na equação da reta, onde calculamos os incrementos para x e y e iteramos pelos pixels ao longo da linha. O programa ficou assim:

analitico.py

home > hendrick > Documentos > Rasterização de Linhas > analitico.py > ...

```python
import pygame
import sys

# Configuração da janela
screen = pygame.display.set_mode((400, 400))
pygame.display.set_caption("Rasterização de Linhas - Algoritmo Analítico")

# Função para rasterizar uma linha usando o algoritmo analítico
def draw_line_analytic(x1, y1, x2, y2):
    m = (y2 - y1) / (x2 - x1)
    b = y1 - m * x1

    dx = x2 - x1
    dy = y2 - y1

    if abs(dx) >= abs(dy):
        steps = abs(dx)
    else:
        steps = abs(dy)

    x_increment = dx / steps
    y_increment = dy / steps

    x, y = x1, y1
    for _ in range(int(steps) + 1):
        pygame.draw.circle(screen, (255, 255, 255), (int(x), int(y)), 1)
        x += x_increment
        y += y_increment

    pygame.display.flip()

# Ponto de partida e destino da linha
x1, y1 = 100, 100
x2, y2 = 300, 300

running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    screen.fill((0, 0, 0))
    draw_line_analytic(x1, y1, x2, y2)
    pygame.display.flip()

pygame.quit()
sys.exit()
```

Figura 1 – Algoritmo Analítico

3.2 Algoritmo DDA

O segundo programa utiliza o algoritmo DDA (Digital Differential Analyzer) para rasterizar uma linha. O DDA é um algoritmo incremental que calcula os incrementos para x e y com base nas diferenças de coordenadas entre os pontos de partida e destino. O programa ficou assim:

dda.py

home > hendrick > Documentos > Rasterização de Linhas > dda.py > ...

```python
import pygame
import sys

# Configuração da janela
screen = pygame.display.set_mode((400, 400))
pygame.display.set_caption("Rasterização de Linhas - Algoritmo DDA")

# Função para rasterizar uma linha usando o algoritmo DDA
def draw_line_dda(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    steps = max(abs(dx), abs(dy))

    x_increment = dx / steps
    y_increment = dy / steps

    x, y = x1, y1
    for _ in range(int(steps) + 1):
        pygame.draw.circle(screen, (255, 255, 255), (int(x), int(y)), 1)
        x += x_increment
        y += y_increment

    pygame.display.flip()

# Ponto de partida e destino da linha
x1, y1 = 100, 100
x2, y2 = 300, 300

running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    screen.fill((0, 0, 0))
    draw_line_dda(x1, y1, x2, y2)
    pygame.display.flip()

pygame.quit()
sys.exit()
```

Figura 2 – Algoritmo DDA

3.3 Algoritmo de Bresenham

O terceiro programa foi usado o algoritmo de Bresenham para rasterizar uma linha. Este algoritmo é amplamente utilizado devido à sua eficiência e é particularmente eficaz em sistemas com recursos limitados. O programa ficou assim:

bresenham.py

```python
import pygame
import sys

# Configuração da janela
screen = pygame.display.set_mode((400, 400))
pygame.display.set_caption("Rasterização de Linhas - Algoritmo de Bresenham")

# Função para rasterizar uma linha usando o algoritmo de Bresenham
def draw_line_bresenham(x1, y1, x2, y2):
    dx = abs(x2 - x1)
    dy = abs(y2 - y1)
    p = 2 * dy - dx
    two_dy = 2 * dy
    two_dy_minus_dx = 2 * (dy - dx)

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

    x, y = x1, y1

    pygame.draw.circle(screen, (255, 255, 255), (x, y), 1)

    while x < x2:
        x += 1
        if p < 0:
            p += two_dy
        else:
            y += 1 if y1 < y2 else -1
            p += two_dy_minus_dx

        pygame.draw.circle(screen, (255, 255, 255), (x, y), 1)

    pygame.display.flip()

```

```
# Ponto de partida e destino da linha
x1, y1 = 100, 100
x2, y2 = 300, 300

running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    screen.fill((0, 0, 0))
    draw_line_bresenham(x1, y1, x2, y2)
    pygame.display.flip()

pygame.quit()
sys.exit()
```

F**igura 3 –** Algoritmo Bresenham

4 Resultados e Comparação

Os três programas foram executados para rasterizar uma linha entre os mesmos pontos de partida e destino ((100, 100) e (300, 300)). Em questão de construção, os três foram trabalhosos, para mim. Procurei algumas referências, encontrei algumas que davam um direcionamento usando pygame. O que trouxe uma facilidade na hora da visualização. Pude perceber, durante a pesquisa para o trabalho, que o que mais distingue esses algoritmos são necessidades específicas do projeto. Por exemplo, o algoritmo de Bresenham é frequentemente preferido devido à sua eficiência, mas os outros algoritmos também podem ser usados com sucesso em diferentes situações.

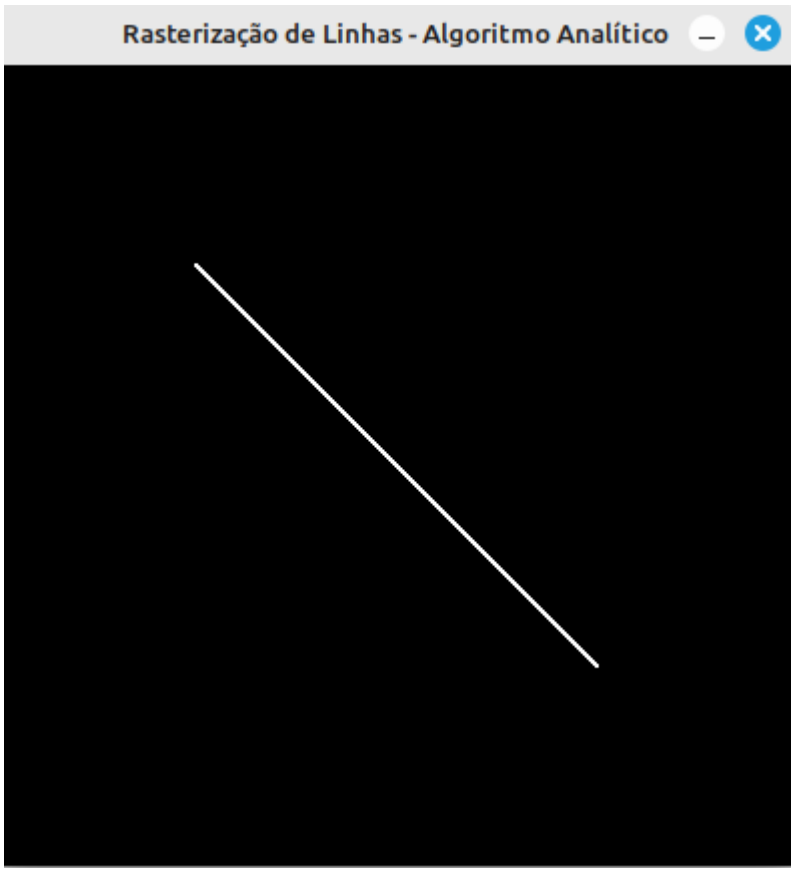

Figura 4 – Resultado do Algoritmo Analítico

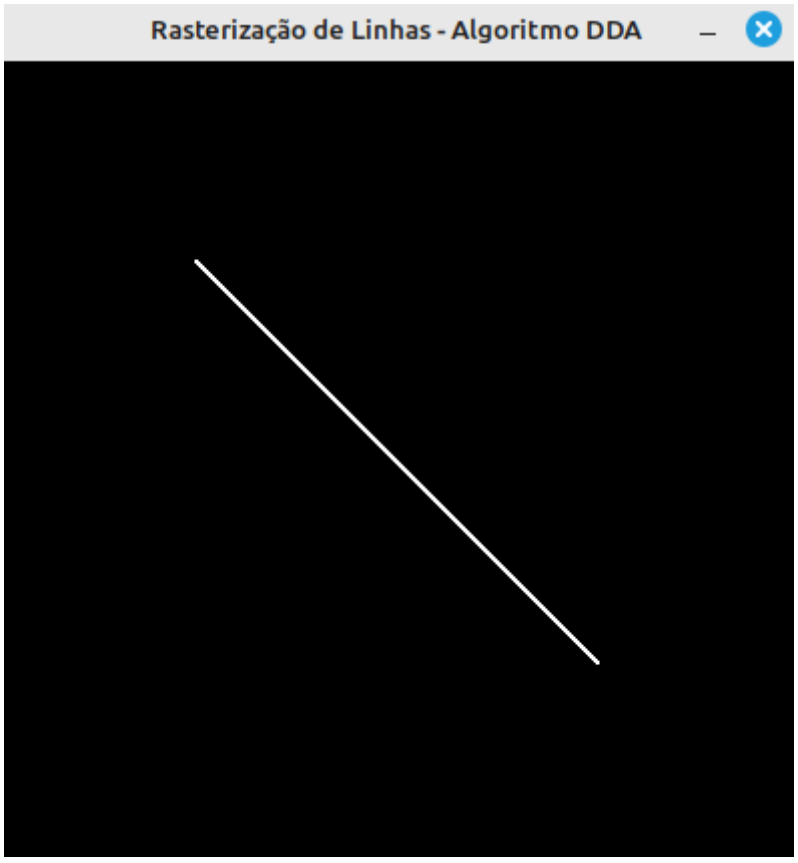

Figura 5 – Resultado do Algoritmo DDA

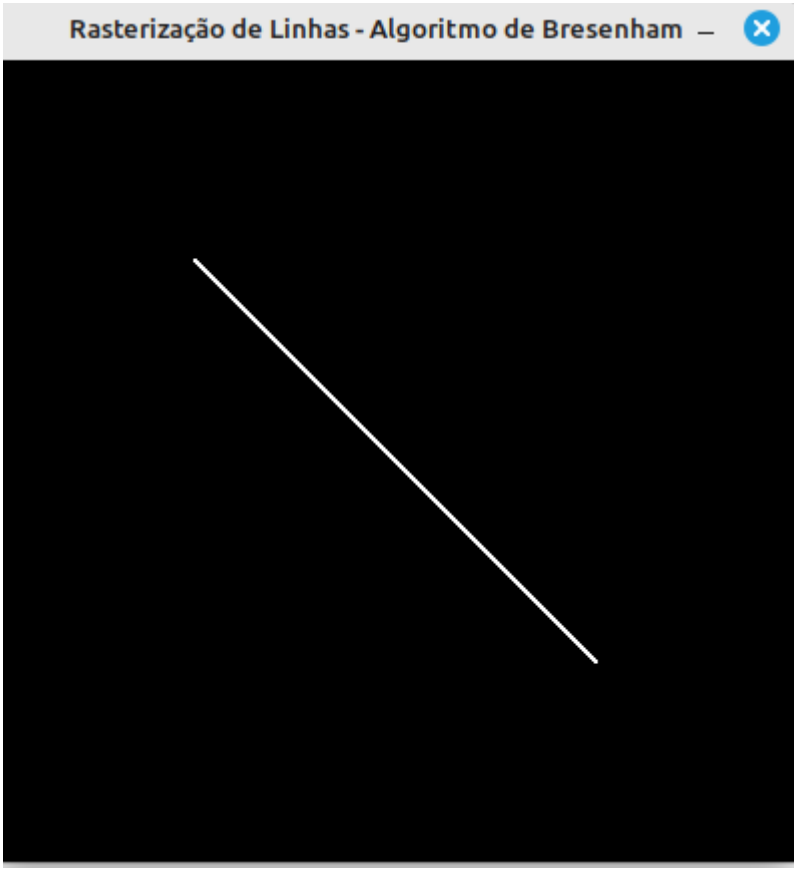

Figura 6 – Resultado do Algoritmo Bresenham

4.1 Comparativo

O algoritmo analítico é preciso, mas pode ser menos eficiente em sistemas sem suporte a ponto flutuante. O DDA é um pouco mais eficiente, especialmente em sistemas com recursos limitados, uma vez que evita cálculos de ponto flutuante. No entanto, o algoritmo de Bresenham é altamente eficiente e mantém a precisão, tornando-se a escolha preferida em muitos casos, especialmente em sistemas embarcados ou com recursos limitados.

5 Conclusão

A escolha do algoritmo de rasterização de linha depende das necessidades específicas do projeto. O algoritmo de Bresenham é geralmente preferido por sua eficiência, mas outros algoritmos podem ser usados com sucesso em diferentes situações. Ao escolher um algoritmo adequado, é importante considerar o equilíbrio entre precisão e eficiência, bem como os recursos disponíveis no ambiente de execução.

6 Referências

- https://gist.github.com/hallazzang/df3fde293e875892be02
- http://www.poshy.net/java/graphic/linedraw4.htm
- https://gist.github.com/0xKD/4714726
- https://stackoverflow.com/questions/57618029/how-to-slowly-draw-a-line-in-python