



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

1	Introdução.....	6
1.1	Algoritmo Analítico.....	6
1.2	Algoritmo DDA.....	6
1.3	Algoritmo Bresenham.....	6
2	Construção dos Programas.....	6
2.1	Configuração do Ambiente.....	7
2.2	Entrada dos Pontos de Início e Destino.....	7
2.3	Loop Principal.....	7
3	Descrição dos Programas.....	7
3.1	Algoritmo Analítico.....	7
3.2	Algoritmo DDA.....	9
3.3	Algoritmo de Bresenham.....	10
4	Resultados e Comparação.....	11
4.1	Comparativo.....	13
5	Conclusão.....	13
6	Referencias.....	14

Figuras

Figura 1 – Algoritmo Analítico	8
Figura 2 – Algoritmo DDA	9
Figura 3 – Algoritmo Bresenham	10
Figura 4 – Resultado do Algoritmo Analítico	12
Figura 5 – Resultado do Algoritmo DDA	12
Figura 6 – Resultado do Algoritmo Bresenham	13

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.

O método analítico utiliza a equação da reta na forma:

$$y = mx + b$$

Onde m (coeficiente angular) é dado por:

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

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 X
home > hendrick > Documentos > Rasterização de Linhas > analitico.py > ...
1  import pygame
2  import sys
3
4  # Configuração da janela
5  screen = pygame.display.set_mode((400, 400))
6  pygame.display.set_caption("Rasterização de Linhas - Algoritmo Analítico")
7
8  # Função para rasterizar uma linha usando o algoritmo analítico
9  def draw_line_analytic(x1, y1, x2, y2):
10     m = (y2 - y1) / (x2 - x1)
11     b = y1 - m * x1
12
13     dx = x2 - x1
14     dy = y2 - y1
15
16     if abs(dx) >= abs(dy):
17         steps = abs(dx)
18     else:
19         steps = abs(dy)
20
21     x_increment = dx / steps
22     y_increment = dy / steps
23
24     x, y = x1, y1
25     for _ in range(int(steps) + 1):
26         pygame.draw.circle(screen, (255, 255, 255), (int(x), int(y)), 1)
27         x += x_increment
28         y += y_increment
29
30     pygame.display.flip()
31
32 # Ponto de partida e destino da linha
33 x1, y1 = 100, 100
34 x2, y2 = 300, 300
35
36 running = True
37 while running:
38     for event in pygame.event.get():
39         if event.type == pygame.QUIT:
40             running = False
41
42     screen.fill((0, 0, 0))
43     draw_line_analytic(x1, y1, x2, y2)
44     pygame.display.flip()
45
46 pygame.quit()
47 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 x
home > hendrick > Documentos > Rasterização de Linhas > dda.py > ...
1  import pygame
2  import sys
3
4  # Configuração da janela
5  screen = pygame.display.set_mode((400, 400))
6  pygame.display.set_caption("Rasterização de Linhas - Algoritmo DDA")
7
8  # Função para rasterizar uma linha usando o algoritmo DDA
9  def draw_line_dda(x1, y1, x2, y2):
10     dx = x2 - x1
11     dy = y2 - y1
12     steps = max(abs(dx), abs(dy))
13
14     x_increment = dx / steps
15     y_increment = dy / steps
16
17     x, y = x1, y1
18     for _ in range(int(steps) + 1):
19         pygame.draw.circle(screen, (255, 255, 255), (int(x), int(y)), 1)
20         x += x_increment
21         y += y_increment
22
23     pygame.display.flip()
24
25 # Ponto de partida e destino da linha
26 x1, y1 = 100, 100
27 x2, y2 = 300, 300
28
29 running = True
30 while running:
31     for event in pygame.event.get():
32         if event.type == pygame.QUIT:
33             running = False
34
35     screen.fill((0, 0, 0))
36     draw_line_dda(x1, y1, x2, y2)
37     pygame.display.flip()
38
39 pygame.quit()
40 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 X
bresenham.py > ...
1  import pygame
2  import sys
3
4  # Configuração da janela
5  screen = pygame.display.set_mode((400, 400))
6  pygame.display.set_caption("Rasterização de Linhas - Algoritmo de Bresenham")
7
8  # Função para rasterizar uma linha usando o algoritmo de Bresenham
9  def draw_line_bresenham(x1, y1, x2, y2):
10     dx = abs(x2 - x1)
11     dy = abs(y2 - y1)
12     p = 2 * dy - dx
13     two_dy = 2 * dy
14     two_dy_minus_dx = 2 * (dy - dx)
15
16     if x1 > x2:
17         x1, x2 = x2, x1
18         y1, y2 = y2, y1
19
20     x, y = x1, y1
21
22     pygame.draw.circle(screen, (255, 255, 255), (x, y), 1)
23
24     while x < x2:
25         x += 1
26         if p < 0:
27             p += two_dy
28         else:
29             y += 1 if y1 < y2 else -1
30             p += two_dy_minus_dx
31
32         pygame.draw.circle(screen, (255, 255, 255), (x, y), 1)
33
34     pygame.display.flip()
35
```

```

36 # Ponto de partida e destino da linha
37 x1, y1 = 100, 100
38 x2, y2 = 300, 300
39
40 running = True
41 while running:
42     for event in pygame.event.get():
43         if event.type == pygame.QUIT:
44             running = False
45
46     screen.fill((0, 0, 0))
47     draw_line_bresenham(x1, y1, x2, y2)
48     pygame.display.flip()
49
50 pygame.quit()
51 sys.exit()

```

Figura 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.

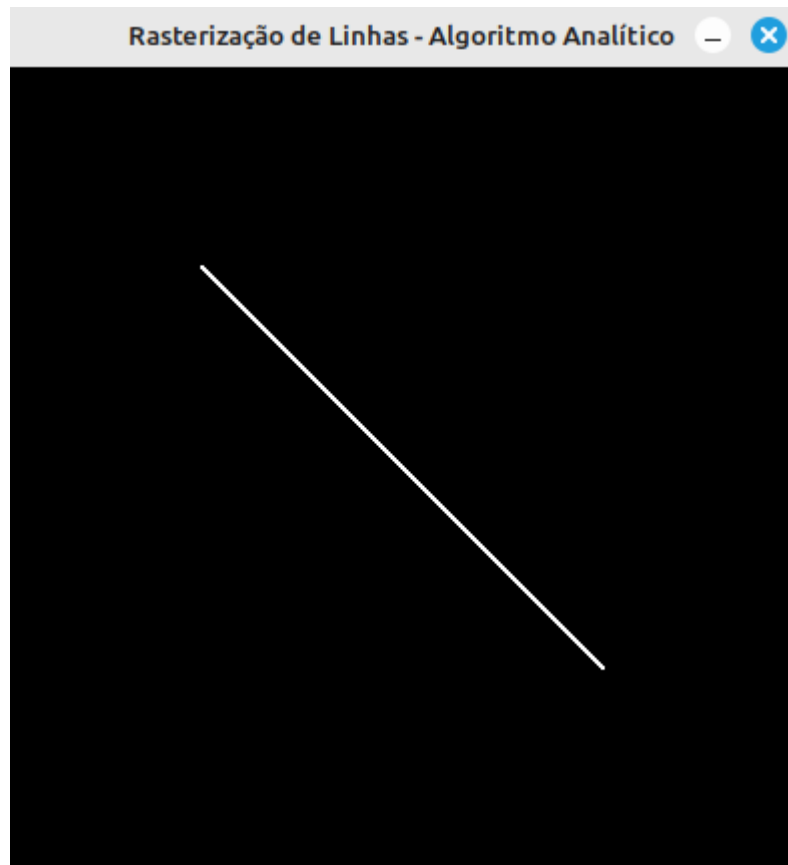


Figura 4 – Resultado do Algoritmo Analítico

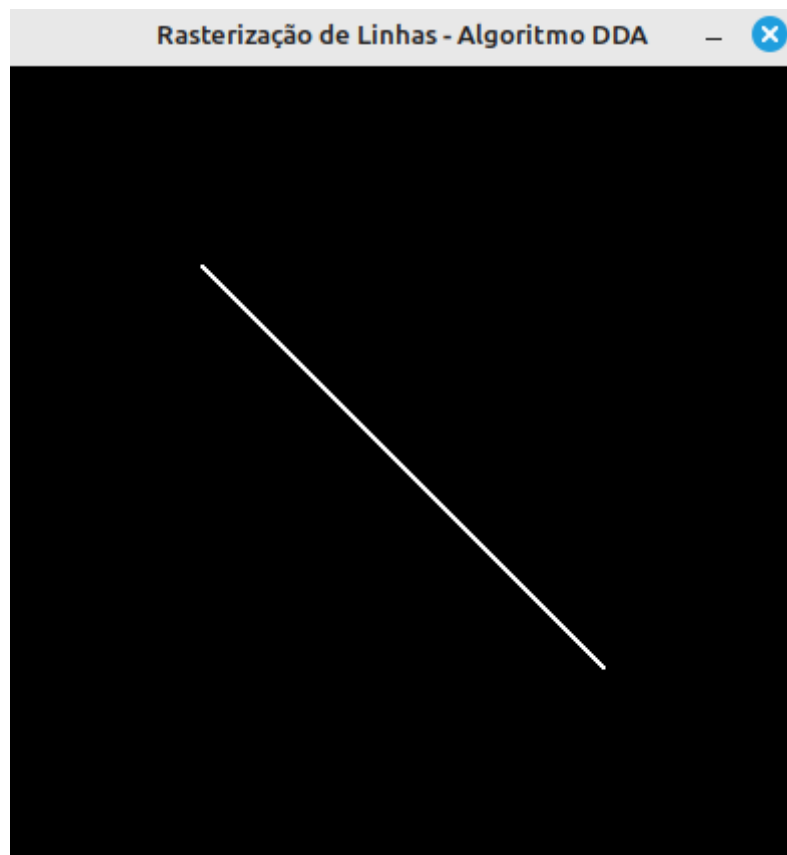


Figura 5 – Resultado do Algoritmo DDA

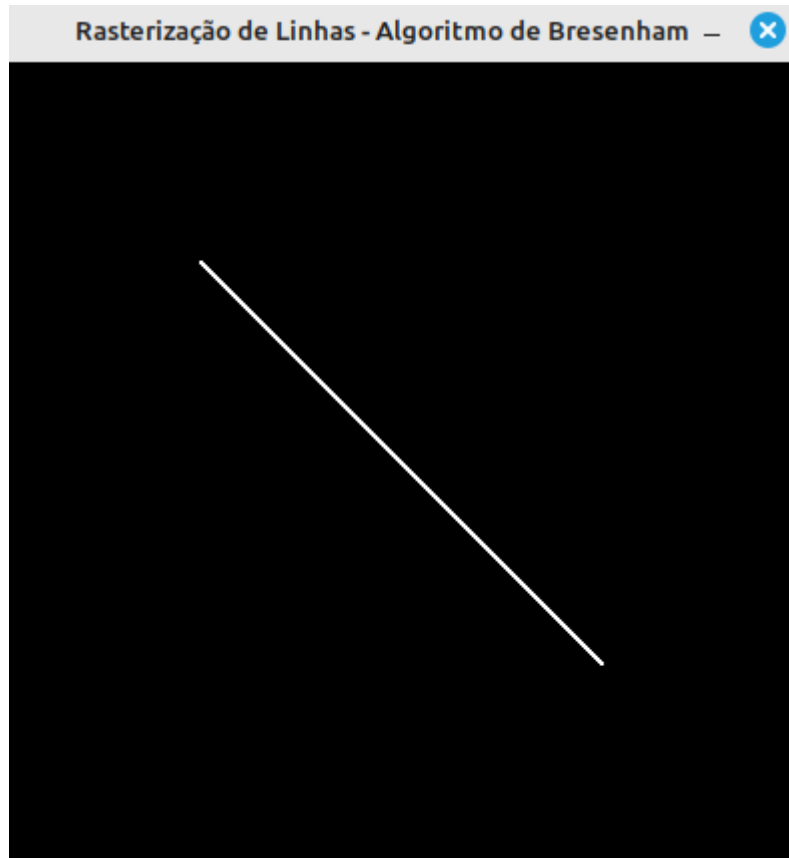


Figura 6 – Resultado do Algoritmo Bresenham

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>