



**UNIVERSIDADE FEDERAL DE RORAIMA
CURSO DE CIÊNCIA DA COMPUTAÇÃO**

VICENTE SAMPAIO

**Relatório Comparativo - Algoritmos de
rasterização e preenchimento**

**Boa Vista – RR
2025**

1 INTRODUÇÃO

Análise comparativa dos algoritmos de rasterização e preenchimento implementados, avaliando desempenho, qualidade visual e eficiência computacional.

2 METODOLOGIA

2.1 Ambiente de Teste

- **Hardware:** Processador i5-11400, Memória 16GB RAM, GPU 3060 Ti
- **Software:** Python 3.12.3, Pygame 2.6.1
- **Resolução:** 1000×1000 pixels, tamanho de pixel: 20×20

2.2 Métricas de Avaliação

- **Tempo:** Medido em milissegundos (média de 10 execuções)
- **Qualidade Visual:** Análise qualitativa através de imagens
- **Casos de Uso:** Cenários onde cada algoritmo se destaca

3 ALGORITMOS DE LINHA

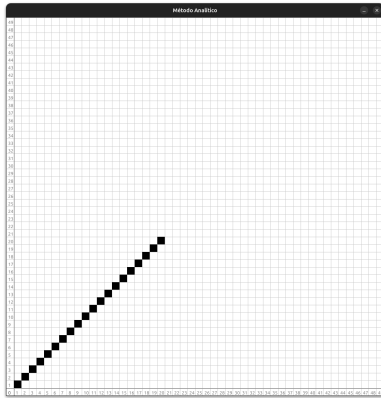
3.1 Algoritmos Testados

- Analítico (Equação da reta)
- DDA (Digital Differential Analyzer)
- Bresenham (Otimizado para inteiros)

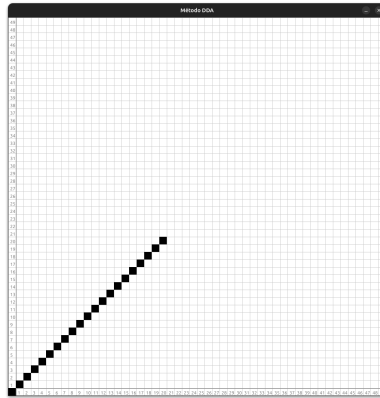
3.2 Métricas Comparativas - Tempo de Execução

Comprimento da Linha	Analítico	DDA	Bresenham
20 pixels	0.5453 ms	0.5384 ms	0.5421 ms
49 pixels	1.3838 ms	1.3839 ms	1.3838 ms

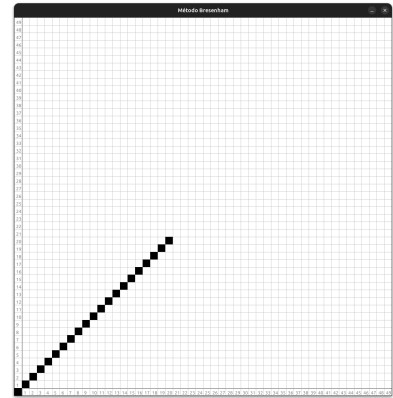
3.3 Imagens Comparativas - Linhas



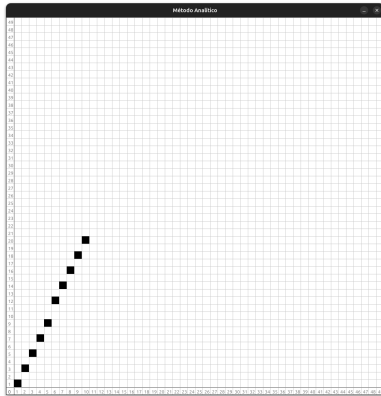
(a) Analítico - Diagonal 45°



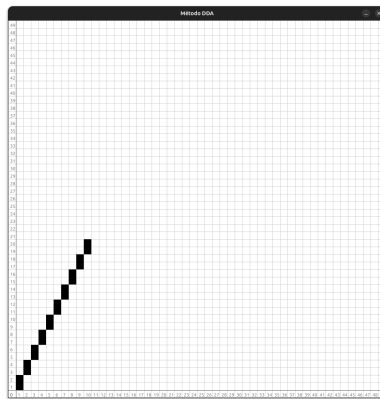
(b) DDA - Diagonal 45°



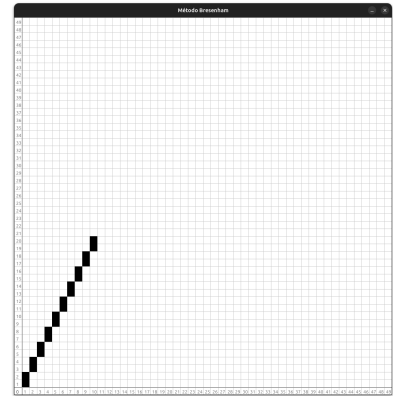
(c) Bresenham - Diagonal 45°



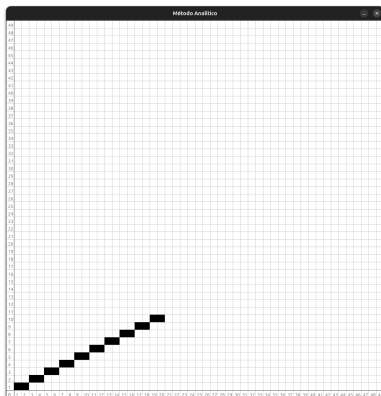
(d) Analítico - Diagonal acima 45°



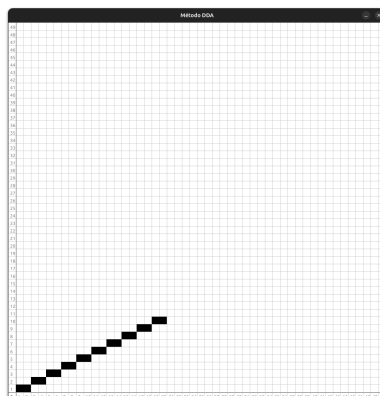
(e) DDA - Diagonal acima 45°



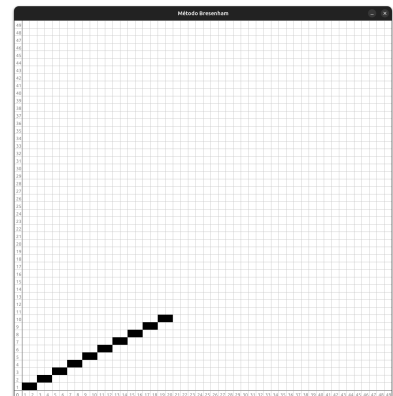
(f) Bresenham - Diagonal acima 45°



(g) Analítico - Diagonal abaixo 45°



(h) DDA - Diagonal abaixo 45°



(i) Bresenham - Diagonal abaixo 45°

Figura 1: Comparação dos algoritmos de linha em diferentes ângulos

3.4 Casos de Superioridade

3.4.1 Analítico

O algoritmo analítico, por mais que seja, dentre os três algoritmos, o pior, foi o primeiro a abordar a rasterização de linhas, servindo de base para os que o sucederam. Tem como pontos negativos o uso de variáveis ‘float’ e arredondamento de valores, o que o torna impreciso. Por conta destes dois problemas, ele sofre dificuldades para rasterizar linhas diagonais que estejam acima de 45 graus.

3.4.2 DDA

Este algoritmo surgiu para reparar o problema principal do algoritmo anterior. Ele é capaz de rasterizar linhas que estejam acima de 45 graus, porém, ainda trabalha com variáveis ‘float’ e arredondamento, o que o torna impreciso também.

3.4.3 Bresenham

O algoritmo de Bresenham se mostrou o melhor dentre os três. Além de tratar os erros do algoritmo analítico, também tratou as limitações do DDA. Este algoritmo é capaz de rasterizar linhas abaixo e acima dos 45 graus, além de conseguir trabalhar em todos os octantes. Por fim, ele excluiu as operações ‘float’ e também o arredondamento, trazendo consigo uma precisão maior.

4 ALGORITMOS DE CIRCUNFERÊNCIA

4.1 Algoritmos Testados

- Incremental Simétrico
- Paramétrico (Equações trigonométricas)
- Bresenham (Otimizado)

4.2 Métricas Comparativas

Métrica	Incremental	Paramétrico	Bresenham
Tempo (r=12)	0.3702 ms	5.9662 ms	0.3370 ms
Tempo (r=24)	0.6767 ms	5.9704 ms	0.6509 ms
Iterações/Pixels (r=24)	190/1520	360/3600	180/1440

4.3 Imagens Comparativas - Circunferências

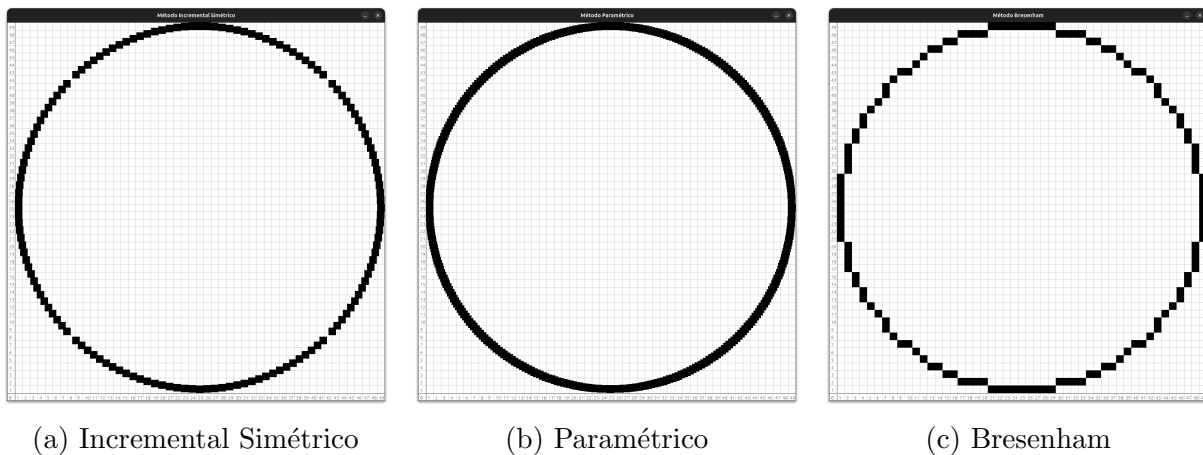


Figura 2: Comparação dos algoritmos de circunferência

4.4 Casos de Superioridade

4.4.1 Incremental

Este algoritmo resolve o problema do Paramétrico ao eliminar as diversas iterações para circunferências de raios diferentes. Ele é mais otimizado por usar valores fixo de seno e cosseno e pintar os pixels por octante, porém ainda trabalha com operações ‘float’.

4.4.2 Paramétrico

Traz consigo uma qualidade muito boa de desenho, porém sua abordagem é muito simples se comparada aos demais algoritmos. Este algoritmo sempre irá pintar 360 pixels por iteração, independente do raio da circunferência, ou seja, o desempenho sempre será inferior para qualquer tamanho do raio. Além disso, trabalha com operações ‘float’, o que o torna mais custoso.

4.4.3 Bresenham

Pinta pixels por octantes, evitando múltiplas iterações desnecessárias, é mais ‘barato’ que os demais algoritmos e também é mais eficaz, utilizando estratégias de cálculo de ativação do próximo pixel através do ponto médio entre eles.

5 ALGORITMOS DE PREENCHIMENTO

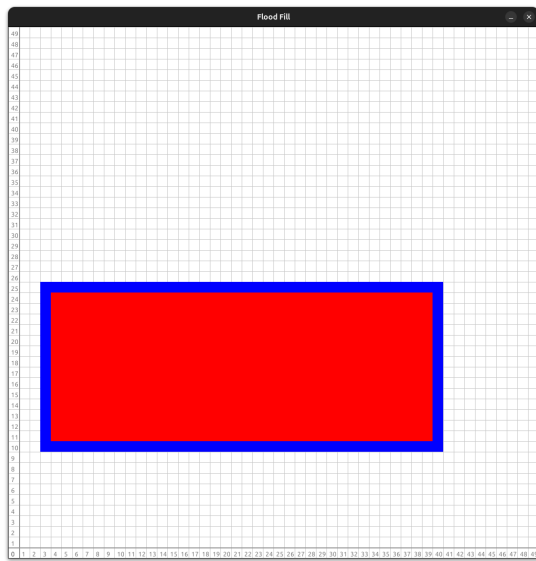
5.1 Algoritmos Testados

- Flood Fill
- Varredura

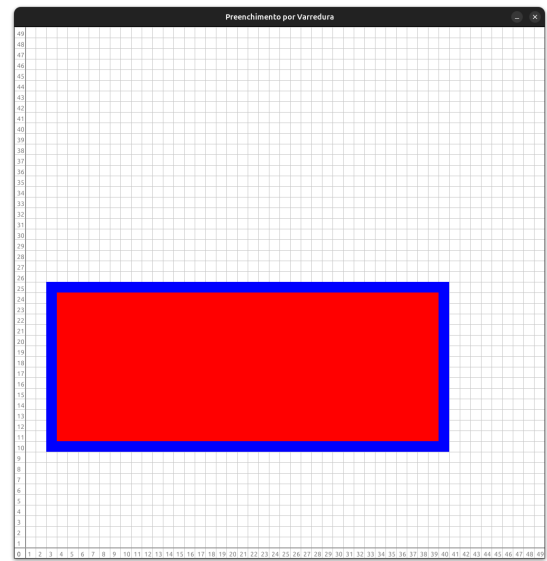
5.2 Métricas Comparativas

Cenário de Teste	Flood Fill	Varredura
Retângulo	8.3588 ms	1.8967 ms
Circunferência (r=12)	7.1401 ms	1.9940 ms
Polígono Complexo A	7.2371 ms	2.0425 ms
Polígono Complexo D	ERRO	1.1868 ms

5.3 Imagens Comparativas - Preenchimento

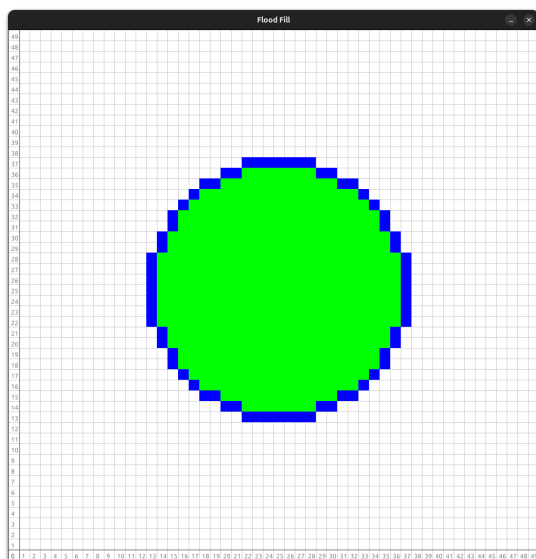


(a) Flood Fill - Retângulo

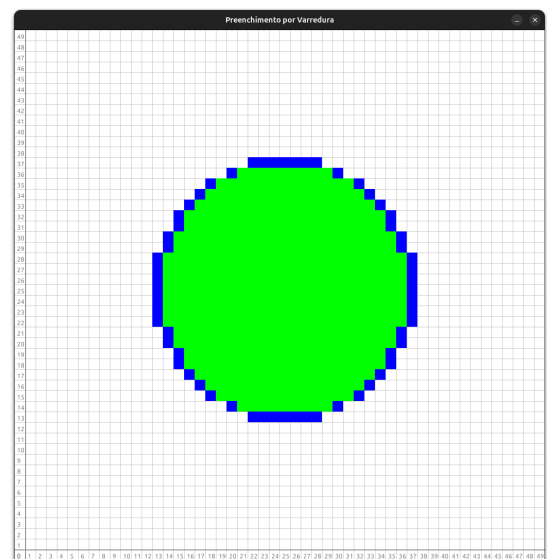


(b) Varredura - Retângulo

Figura 3: Comparação: Retângulo

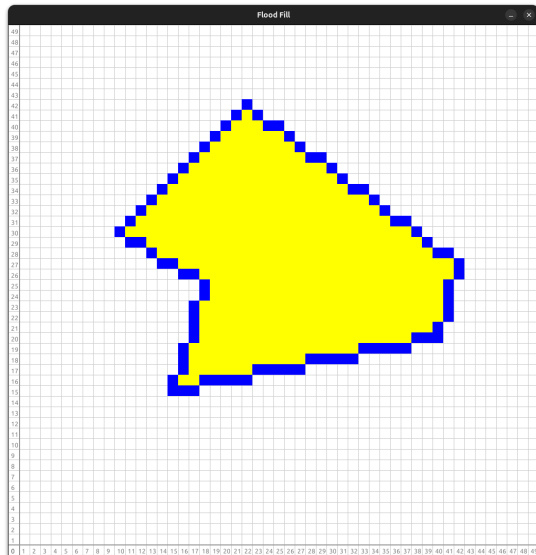


(a) Flood Fill - Circunferência

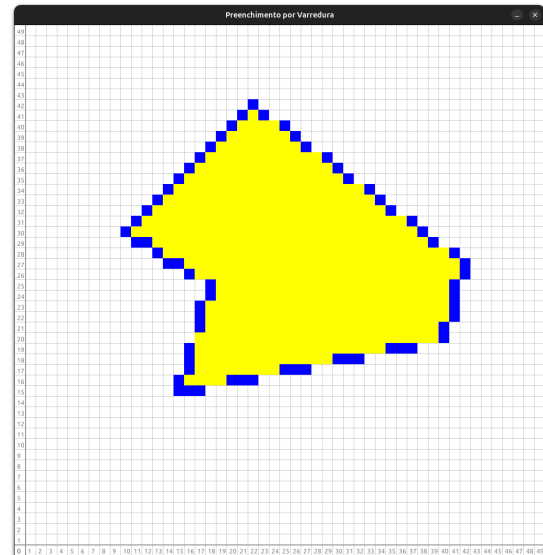


(b) Varredura - Circunferência

Figura 4: Comparação: Circunferência

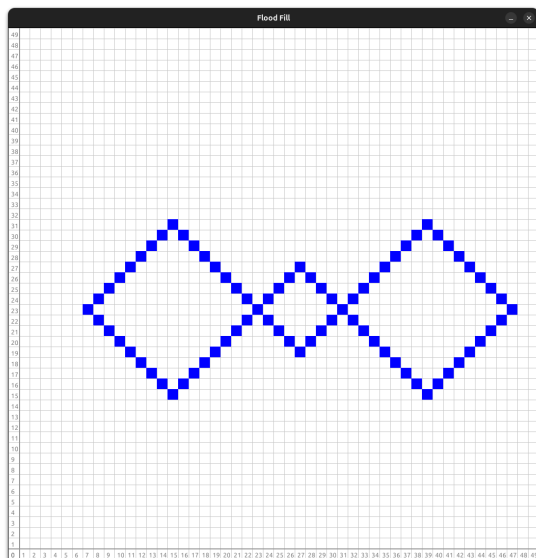


(a) Flood Fill - Polígono Complexo A

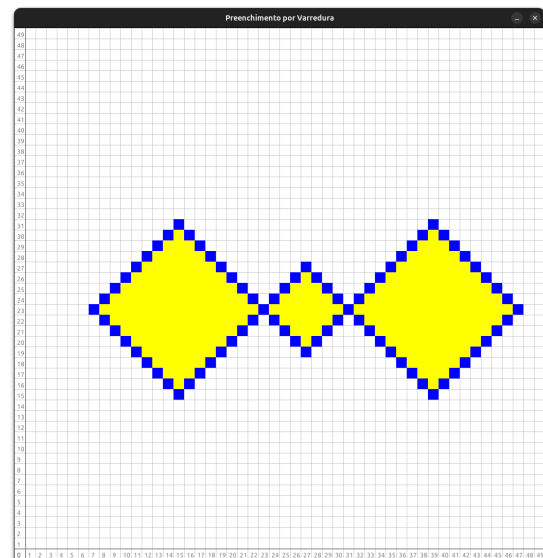


(b) Varredura - Polígono Complexo A

Figura 5: Comparação: Polígono Complexo A



(a) Flood Fill - Polígono Complexo D



(b) Varredura - Polígono Complexo D

Figura 6: Comparação: Polígono Complexo D

5.4 Casos de Superioridade

5.4.1 Flood Fill

Pinta perfeitamente objetos simples tais como, quadrado, retângulo, círculos, losangos, etc, porém, apresenta dificuldades ao trabalhar com forma geométricas que possuem camadas internas ou impasses, o que resulta na não pintura do pixel.

5.4.2 Varredura

Pinta perfeitamente qualquer tipo de objeto, com ou sem camadas internas.

6 CONCLUSÕES

6.1 Linhas e Circunferências

O algoritmo de **Bresenham** demonstrou ser o mais eficiente tanto para linhas quanto para circunferências, eliminando operações com ponto flutuante e oferecendo maior precisão.

6.2 Preenchimento

O algoritmo de **Varredura** mostrou-se superior ao Flood Fill em todos os cenários testados, sendo mais rápido e capaz de lidar com formas complexas sem erros.

6.3 Recomendações

- Utilizar Bresenham para rasterização de linhas e circunferências
- Utilizar Varredura para operações de preenchimento
- Evitar algoritmos baseados em ponto flutuante para aplicações que requerem alta performance