Preechimento de Poligonos

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

Relatório de Preechimento de Poligonos

Introdução

Este relatório descreve o desenvolvimento e os resultados da implementação dos algoritmos de preenchimento **Flood Fill** e **Varredura com Análise Geométrica**, aplicados a diferentes formas geométricas rasterizadas utilizando o algoritmo de **Bresenham** para desenho de contornos. O objetivo é comparar a eficiência e o comportamento desses algoritmos, evidenciando suas características e diferenças.

Algoritmos Implementados

Flood Fill

O **Flood Fill** é um algoritmo de preenchimento recursivo que expande a partir de um ponto inicial, substituindo os pixels de cor original pela cor de preenchimento. Para otimizar a execução, utilizamos uma abordagem iterativa baseada em filas (**queue-based flood fill**), que evita problemas de estouro de pilha.

Implementação

O algoritmo foi aplicado sobre quatro formas geométricas distintas:

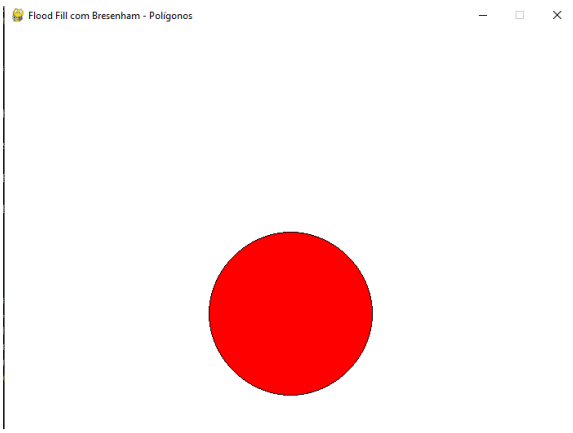
- **Círculo**: Desenhado pelo algoritmo de **Bresenham**.
- **Retângulo**: Construído utilizando o traçado de linhas de **Bresenham**.
- **Polígono C**: Baseado em coordenadas definidas e desenhado com **Bresenham**.
- **Polígono D**: Construído a partir da união de três losangos.

Para iniciar o preenchimento automaticamente, o ponto inicial foi calculado como o **ponto médio** da forma geométrica.

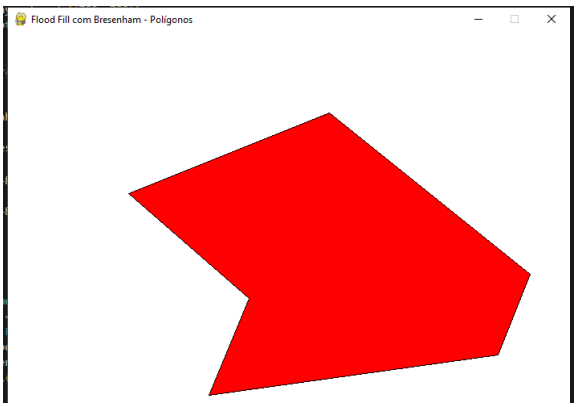
Trecho de Código

```
queue = deque([(x, y)])
target_color = surface.get_at((x, y))

if target_color == fill_color or target_color == border_color:
    return

while queue:
    x, y = queue.popleft()
    if surface.get_at((x, y)) != target_color:
        continue

    surface.set_at((x, y), fill_color)
    pygame.display.flip()
    time.sleep(0.000001)

    queue.append((x + 1, y))
    queue.append((x - 1, y))
    queue.append((x, y + 1))
    queue.append((x, y - 1))
```

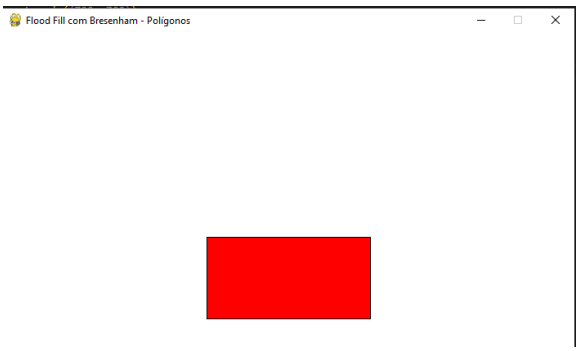
Resultado Visual



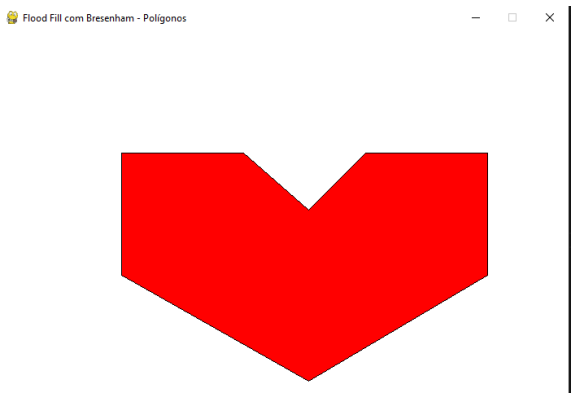
Poligono A - Flood Fill



Poligono C - Flood Fill



Poligono B - Flood Fill



Poligono D - Flood Fill

Algoritmo de Varredura com Análise Geométrica

O **Scanline Fill** é um método eficiente para preenchimento de regiões fechadas. Ele funciona percorrendo a imagem linha por linha (scanline) e preenchendo os segmentos internos da forma com base em interseções de arestas.

Implementação

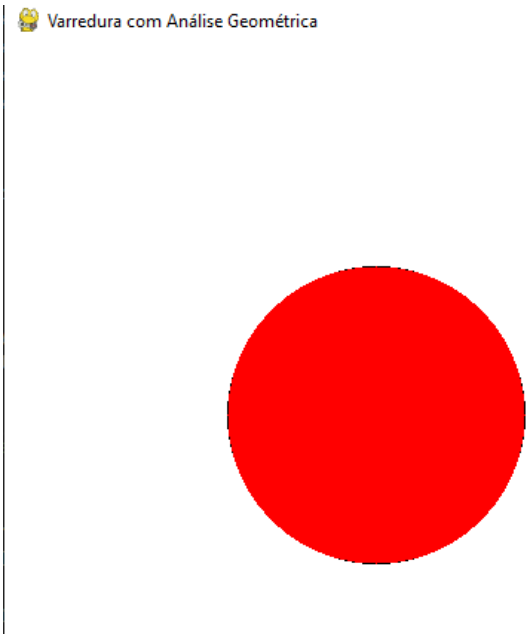
Cada forma geométrica foi desenhada com **Bresenham** e depois preenchida pelo algoritmo de varredura. A detecção de interseções foi feita a partir da ordenação das arestas pelo eixo **y**, garantindo um preenchimento correto e eficiente.

Trecho de Código

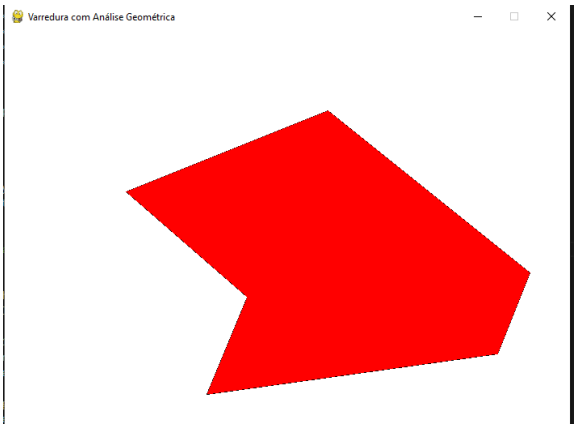
```
for y in range(min_y, max_y + 1):
    active_edges = []
    for edge in edges:
        if edge[0] <= y < edge[1]:
            active_edges.append(edge[2] + (y - edge[0]) * edge[3])

    active_edges.sort()

    for i in range(0, len(active_edges), 2):
        x_start = math.ceil(active_edges[i])
        x_end = math.floor(active_edges[i + 1])
        for x in range(x_start, x_end + 1):
            surface.set_at((x, y), fill_color)
```

```
    pygame.display.flip()
    time.sleep(0.002)
```

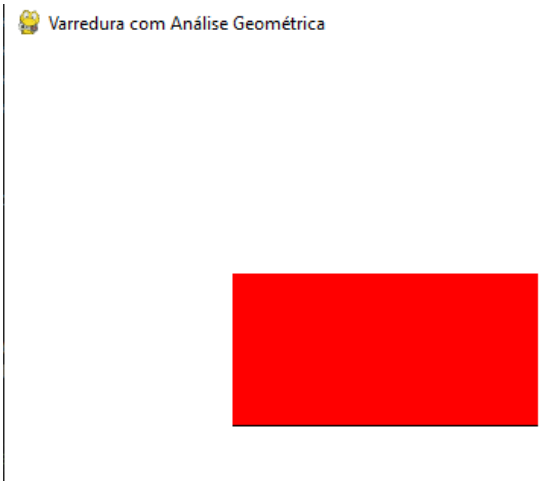
Resultado Visual



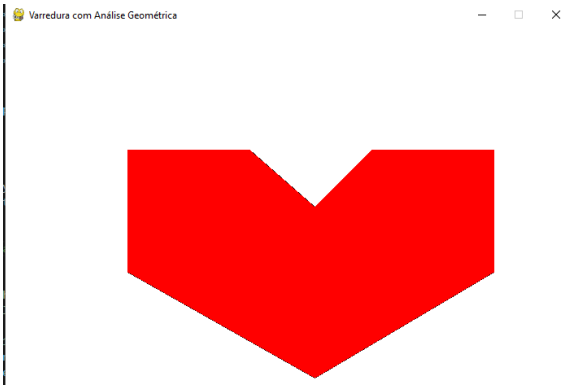
Poligono A - Análise Geométrica



Poligono C - Análise Geométrica



Poligono B - Análise Geométrica



Poligono D - Análise Geométrica

Comparativo entre os Algoritmos

Característica	Flood Fill	Varredura com Análise Geométrica
Princípio	Expansão recursiva/iterativa	Scanline baseado em interseções
Velocidade	Mais lento em formas complexas	Mais eficiente para regiões grandes
Memória	Pode consumir muita memória	Uso eficiente de estruturas
Precisão	Pode apresentar falhas em bordas	Preenchimento mais exato
Melhor aplicação	Pequenas regiões e preenchimento local	Polígonos e áreas extensas

Conclusão

Os dois algoritmos de preenchimento foram testados em diversas formas geométricas e apresentaram comportamentos distintos. O **Flood Fill** demonstrou ser mais simples de implementar, mas menos eficiente em formas complexas. Já o **Scanline Fill** proporcionou um preenchimento mais rápido e preciso, sendo mais indicado para aplicações que envolvem grandes áreas. Ambos os algoritmos foram adaptados para iniciar o preenchimento automaticamente a partir do **ponto médio** das formas, otimizando sua execução.