UNIVERSIDADE FEDERAL DE RORAIMA
CENTRO DE CIÊNCIA E TECNOLOGIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
DCC703 - COMPUTAÇÃO GRÁFICA

RAFAEL NÓBREGA DE LIMA

ALGORITMOS DE PREENCHIMENTO - RELATÓRIO

BOA VISTA - RR

2025

1. BASE DOS ALGORITMOS E ORGANIZAÇÃO

A linguagem escolhida para execução dos algoritmos de Preenchimento foi “Python”, pois é uma linguagem de certa forma simples comparada com outras de alto nível, além disso ela dispõe de muitas bibliotecas para a renderização de imagens. A biblioteca usada nesses algoritmos é a “Pygame”, uma biblioteca com foco na criação de jogos, mas com a capacidade para a renderização e visualização das formas criadas pelo o algoritmo.

A tela para o algoritmo de preenchimento foi definida com uma resolução de 800x800 pixels. Para aprimorar a visualização, foi implementado um grid com células de 10x10 pixels, resultando em uma grade de 80x80. Essa abordagem permite uma comparação mais clara entre os diferentes métodos de preenchimento, facilitando a observação das suas particularidades e diferenças.

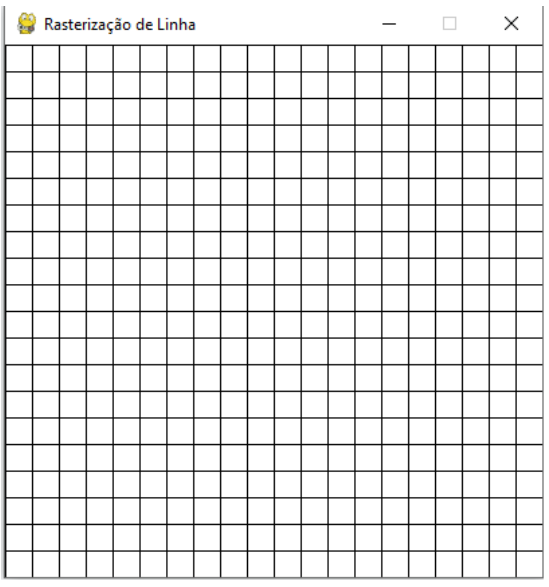
```python
# Configurações do grid
tamanho_celula = 10
linhas, colunas = altura // tamanho_celula, largura // tamanho_celula
```

1.1. DESENHO DO GRID

- A função desenhar_grid() configura e cria uma grade visual para representar um sistema de coordenadas discretas.

```python
# Desenha o grid na tela
def desenhar_grid():
    for x in range(0, largura, tamanho_celula):
        pygame.draw.line(tela, PRETO, (x, 0), (x, altura))
    for y in range(0, altura, tamanho_celula):
        pygame.draw.line(tela, PRETO, (0, y), (largura, y))
```

- Linhas verticais e horizontais são desenhadas para dividir a tela em células. Assim ajudando na melhor visualização das formas geradas.



1.2. PINTURA DE PIXELS

- A função **desenhar_pontos( )** percorre o grid verificando cada célula e pinta os pixels conforme a cor armazenada na matriz. Cada célula da grade (grid[y][x]) pode conter diferentes cores, como PRETO para as bordas ou VERMELHO para o preenchimento.

```
def desenhar_pontos():
    """Desenha os pontos do grid."""
    for y in range(linhas):
        for x in range(colunas):
            if grid[y][x] != BRANCO:
                pygame.draw.rect(tela, grid[y][x], (x * tamanho_celula, y * tamanho_celula, tamanho_celula, tamanho_celula))
```

- Os pontos são desenhados utilizando **pygame.draw.rect( )**, que cria pequenos quadrados preenchidos, representando cada pixel na tela. Esse método garante que todas as células do grid sejam atualizadas corretamente, refletindo a rasterização das bordas e o preenchimento interno das formas.

1.3. POLÍGONOS

- Para ser possível a criação das polígonos foi usado os algoritmos de Bresenham para rasterização de linhas e circunferências, com a implementação desses algoritmos foi possível a criação de polígonos mais complexos.

```
# 📌 **Escolha da Forma**
forma_selecionada = "zero"  # Altere para "retangulo", "triangulo", "hexagono", "circunferencia", "forma_a", "forma_c"

formas = {
    "zero": [(0,0)],
    "retangulo": [(-15, -10), (15, -10), (15, 10), (-15, 10)],
    "triangulo": [(0, -20), (20, 20), (-20, 20)],
    "hexagono": [(0, -20), (17, -10), (17, 10), (0, 20), (-17, 10), (-17, -10)],
    "forma_a": [(-20, -15), (5, -25), (30, -5), (26, 5), (-10, 10), (-5, -2)],
    "forma_c": [(-15, -15), (0, -15), (8, -8), (15, -15), (30, -15), (30, 0), (8, 13), (-15, 0)]
}
```

- As formas são compostas por coordenadas no grid, representando seus vértices. Cada forma pode ser selecionada e renderizada na tela, sendo sua borda construída por segmentos de reta rasterizados.
- Após a rasterização, a forma pode ser preenchida internamente utilizando os algoritmos de **Flood Fill** (propagação) ou **Scanline** (varredura linha por linha), garantindo um preenchimento visualmente coerente.

2. ALGORITMOS DE PREENCHIMENTO

Algoritmos de preenchimento servem para definir o conjunto de pixels que será desenhado dentro de um determinado contorno fechado.

2.1. Flood Fill

O Flood Fill é um algoritmo de preenchimento que expande a partir de um ponto inicial, colorindo todas as áreas conectadas da mesma cor. Ele funciona de forma semelhante à ferramenta balde de tinta em editores de imagem.

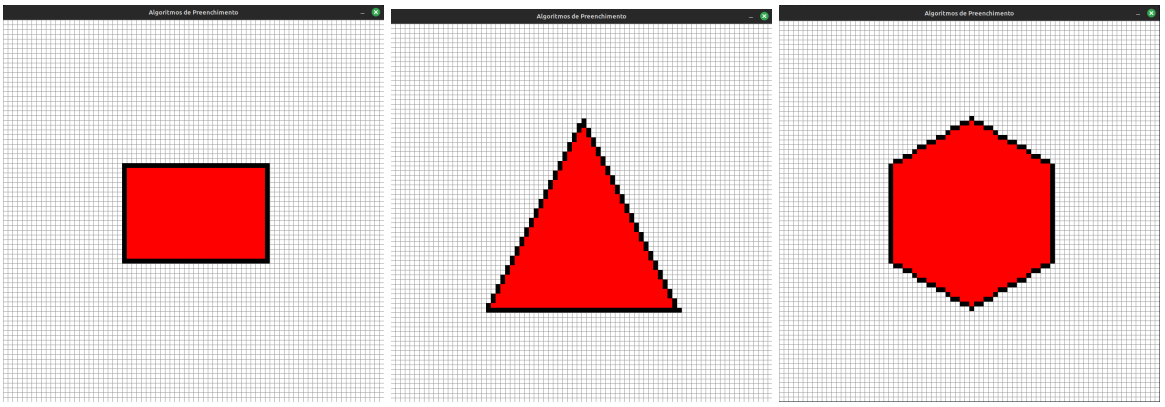
- **CÓDIGO IMPLEMENTADO**

Existem duas formas principais para se implementar o Flood Fill, a primeira seria usando recursão e a segunda é utilizando uma estrutura de fila. O problema de implementar usando recursão é que se a área a ser preenchida for muito grande pode ocorrer um estouro de pilha. Com isso o algoritmo implementado foi o que usa fila (BFS - Busca em largura), pois ele utiliza uma fila (deque) para armazenar os pixels a serem processados.

```python
def flood_fill(grid, x, y, target_color, new_color):
    """Preenchimento Flood Fill usando BFS."""
    if x < 0 or x >= colunas or y < 0 or y >= linhas or grid[y][x] != target_color or target_color == new_color:
        return
    queue = deque([(x, y)])
    while queue:
        cx, cy = queue.popleft()
        if 0 <= cx < colunas and 0 <= cy < linhas and grid[cy][cx] == target_color:
            grid[cy][cx] = new_color
            for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
                nx, ny = cx + dx, cy + dy
                queue.append((nx, ny))
```

- **PASSOS**
    - O algoritmo começa em um ponto específico dentro da região que deve ser preenchida.
    - Se a cor do pixel for **diferente** da cor de preenchimento e for igual à cor original, o preenchimento continua.
    - O pixel atual recebe a nova cor de preenchimento.
    - O algoritmo se move para os pixels adjacentes (normalmente quatro direções: **cima, baixo, esquerda, direita**).
    - O algoritmo continua expandindo até que **não existam mais pixels da cor original** para serem substituídos.

- **RESULTADOS**



- **LIMITAÇÕES**

  O Flood Fill apresenta problemas de desempenho em áreas grandes, pois quanto maior a região a ser preenchida, maior será o consumo de tempo e memória. Além disso, ele preenche apenas pixels conectados diretamente, o que significa que se houver um pequeno espaço de outra cor separando partes da mesma região, essas áreas não serão preenchidas. Na sua versão recursiva, pode ocorrer um estouro de pilha caso o número de chamadas ultrapasse o limite da linguagem. Outra limitação é sua ineficiência ao lidar com formas complexas que possuem buracos internos, pois o algoritmo interpreta as bordas desses buracos como limites da região, impedindo seu preenchimento.

2.2. SCANLINE FILL(PREENCHIMENTO POR VARREDURA)

O Scanline Fill é um algoritmo de preenchimento usado para colorir regiões delimitadas por polígonos. Ele funciona percorrendo a área linha por linha (ou varrendo horizontalmente), determinando os pontos internos da forma e preenchendo-os.

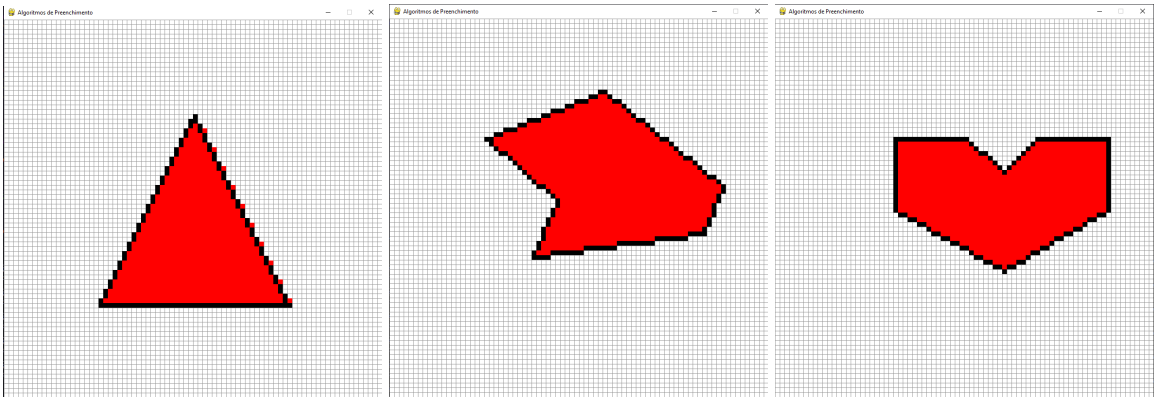
- **CÓDIGO IMPLEMENTADO**

```python
def scanline_fill(grid, pontos_poligono):
    """Preenchimento por varredura (Scanline)."""
    min_y, max_y = min(p[1] for p in pontos_poligono), max(p[1] for p in pontos_poligono)
    for y in range(min_y, max_y + 1):
        intersecoes = []
        for i in range(len(pontos_poligono)):
            x0, y0 = pontos_poligono[i]
            x1, y1 = pontos_poligono[(i + 1) % len(pontos_poligono)]
            if y0 == y1:
                continue
            if (y0 <= y < y1) or (y1 <= y < y0):
                x_intersecao = x0 + (y - y0) * (x1 - x0) / (y1 - y0)
                intersecoes.append(int(round(x_intersecao)))
        intersecoes.sort()
        for i in range(0, len(intersecoes), 2):
            if i + 1 < len(intersecoes):
                for x in range(intersecoes[i], intersecoes[i + 1] + 1):
                    if 0 <= x < colunas and 0 <= y < linhas and grid[y][x] != PRETO:
                        grid[y][x] = VERMELHO
```

- **PASSOS**
  - Para cada linha horizontal (scanline) da imagem, o algoritmo identifica os pontos onde essa linha cruza as bordas do polígono.
  - Os pontos de interseção encontrados são organizados em pares, indicando o início e o fim do preenchimento para aquela linha.
  - Após identificar os pares de interseção, o algoritmo preenche todos os pixels entre eles, garantindo que apenas a área interna do polígono seja preenchida.
  - Esse processo se repete para cada linha do grid, garantindo o preenchimento de toda a região interna do polígono.

- **RESULTADOS**

Este algoritmo tem um preenchimento satisfatório para polígonos mais complexos, mas se as bordas do polígono forem muito inclinadas , ocorre uma falha de preenchimento como foi o caso do triângulo que ficou com o preenchimento “vazado”.



- **LIMITAÇÕES**
  Se a borda do polígono for muito inclinada ou até mesmo se não estiverem bem definidas, pode ocorrer falhas no preenchimento, causando vazamentos ou pixels não preenchidos corretamente. Outro problema que pode vir a ocorrer já que os cálculos envolvem operações com coordenadas de pixels discretos, arredondamentos podem causar pequenos desalinhamentos nos preenchimentos.

- **CONCLUSÃO**
  O Scanline Fill é ideal para preenchimento rápido de polígonos fechados, enquanto o Flood Fill é melhor para preenchimento baseado em conectividade dentro de uma área.