UNIVERSIDADE FEDERAL DE RORAIMA CENTRO DE CIÊNCIA E TECNOLOGIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
DCC703 - COMPUTAÇÃO GRÁFICA (2024.2)
Prof. LUCIANO FERREIRA SILVA

Nome: Marcia Gabrielle Bonifácio De Oliveira - 2020011319

TRABALHO 3

1. INTRODUÇÃO

Nesta atividade, os algoritmos foram aplicados para preencher as seguintes formas:

- Círculo: Após desenhar sua borda (usando o algoritmo do ponto médio), o interior é preenchido.
- Retângulo: Desenhado com a função de desenho do PIL, preenchido internamente.
- Dois polígonos: Cujo conjunto de vértices é normalizado e um ponto "seed" é determinado para iniciar o preenchimento.

Diferentes abordagens foram exploradas:

Flood Fill Recursivo: A função chama a si mesma para preencher os pixels vizinhos a partir de um ponto seed. Para evitar problemas de *stack overflow* (devido à profundidade de recursão), as dimensões das formas foram reduzidas.

Algoritmo de Varredura (Scanline): Em vez de preencher pixel a pixel, este método percorre cada linha horizontal da imagem, determinando, com base na análise geométrica (por exemplo, através da equação do círculo ou da interpolação linear para polígonos), quais intervalos devem ser preenchidos.

2. CONCEITOS BÁSICOS

2.1. Flood Fill

O Flood Fill preenche uma região conexa a partir de um ponto inicial (seed). O procedimento básico envolve:

- Ponto Seed: Início do preenchimento a partir de um pixel que pertence à área desejada.
- Verificação dos Vizinhos: Os pixels imediatamente adjacentes (direita, esquerda, acima e abaixo) são verificados.
- Propagação Recursiva: Se o pixel vizinho possui a cor alvo (por exemplo, branco – 255), ele é preenchido com a nova cor (por exemplo, cinza – 150) e o processo é repetido para seus vizinhos.

2.2. Algoritmo de Varredura (Scanline)

O algoritmo de varredura é uma técnica que utiliza linhas horizontais (scanlines) para preencher uma área. Os passos principais são:

- Determinação dos Limites: Para cada linha (valor de *y*) dentro do intervalo de interesse, calcula-se os pontos de interseção da linha com a forma.
- Cálculo das Interseções:

Para uma circunferência: Utiliza-se a equação

$$\bullet\ x_1 = x_c - \sqrt{R^2 - (y-y_c)^2}$$

$$\bullet\ x_2 = x_c + \sqrt{R^2 - (y-yc)^2}$$

para definir os limites horizontais xmin=xc−dx e xmax=xc+dx

Para polígonos: São calculadas as interseções de cada aresta (por meio de interpolação linear), ignorando arestas horizontais, e os intervalos entre pares de interseções são preenchidos.

Preenchimento: Apenas os pixels internos à forma (não pertencentes à borda) são modificados.

3. Metodologia e Implementação em Python

3.1. Flood Fill Recursivo

A implementação recursiva foi realizada por meio da função flood_fill_recursive, que:

- Verifica os Limites e a Cor: Se o pixel atual estiver fora dos limites ou não tiver a cor alvo, a função retorna.
- Preenche o Pixel e Propaga: Caso contrário, o pixel é preenchido com a nova cor e a função é chamada recursivamente para os quatro vizinhos.
- Atenção à Recursão: Para evitar problemas de *stack overflow*, as dimensões das formas (círculo e retângulo) foram reduzidas.

```python
def flood_fill_recursive(img_array, x, y, target_color, fill_color):
    # Verifica se o ponto está fora dos limites
    if x < 0 or x >= img_array.shape[1] or y < 0 or y >= img_array.shape[0]:
        return
    # Se o pixel não tem a cor alvo, não faz nada
    if img_array[y, x] != target_color:
        return

    # Preenche o pixel
    img_array[y, x] = fill_color

    # Chama recursivamente para os vizinhos
    flood_fill_recursive(img_array, x + 1, y, target_color, fill_color)  # direita
    flood_fill_recursive(img_array, x - 1, y, target_color, fill_color)  # esquerda
    flood_fill_recursive(img_array, x, y + 1, target_color, fill_color)  # abaixo
    flood_fill_recursive(img_array, x, y - 1, target_color, fill_color)  # acima
```

```python
def find_seed_point(polygon, img_array):
    min_x = min(p[0] for p in polygon)
    max_x = max(p[0] for p in polygon)
    min_y = min(p[1] for p in polygon)
    max_y = max(p[1] for p in polygon)
    seed_x = (min_x + max_x) // 2
    seed_y = (min_y + max_y) // 2
    if img_array[seed_y, seed_x] == 255:
        return seed_x, seed_y
    else:
        for y in range(min_y, max_y):
            for x in range(min_x, max_x):
                if img_array[y, x] == 255:
                    return x, y
    return None, None
```

3.2. Algoritmo de Varredura (Scanline)

A técnica de varredura envolve funções específicas para cada forma:

- fill_rectangle_scanline: Varre horizontalmente, preenchendo intervalos fixos de x para cada y.

```python
def fill_rectangle_scanline(arr, x_min, y_min, x_max, y_max, fill_color):
    for y in range(y_min, y_max + 1):
        if y < 0 or y >= arr.shape[0]:
            continue  # ignora linhas fora da imagem
        # Garante que os valores de x estejam dentro dos limites da imagem
        x1 = max(x_min, 0)
        x2 = min(x_max, arr.shape[1] - 1)
        # Como já passamos coordenadas internas, podemos preencher diretamente
        arr[y, x1:x2 + 1] = fill_color
```

- fill_circle_scanline: Para cada linha dentro do intervalo do círculo, calcula *dx* e preenche os pixels entre xc−dx e xc+dx.

```python
def fill_circle_scanline(arr, xc, yc, R, fill_color):
    for y in range(int(yc - R), int(yc + R) + 1):
        if y < 0 or y >= arr.shape[0]:
            continue  # ignora linhas fora da imagem
        dy = y - yc
        dx = int(np.sqrt(R * R - dy * dy))
        x_min = int(xc - dx)
        x_max = int(xc + dx)
        # Ajusta os limites horizontais para a imagem
        x_min = max(x_min, 0)
        x_max = min(x_max, arr.shape[1] - 1)
        for x in range(x_min, x_max + 1):
            # Preenche apenas se o pixel não for parte da borda (valor 0)
            if arr[y, x] != 0:
                arr[y, x] = fill_color
```

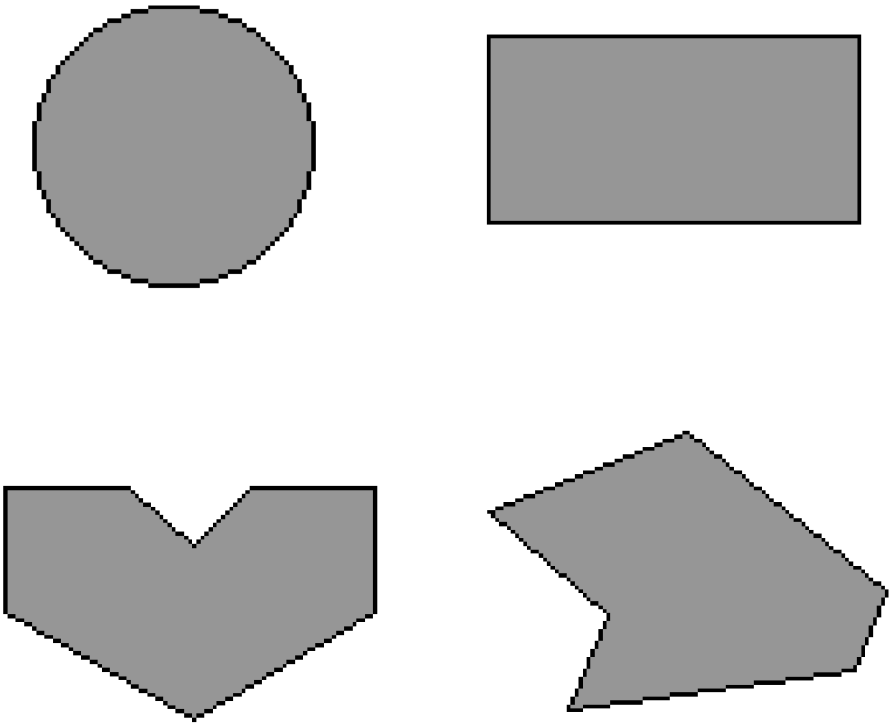
- fill_polygon_scanline: Determina, para cada linha, as interseções entre a scanline e as arestas do polígono, preenchendo os intervalos entre pares de interseções.

```python
def fill_polygon_scanline(arr, polygon, fill_color):
    n = len(polygon)
    y_min = min(y for (x, y) in polygon)
    y_max = max(y for (x, y) in polygon)
    for y in range(y_min, y_max + 1):
        intersections = []
        for i in range(n):
            x1, y1 = polygon[i]
            x2, y2 = polygon[(i + 1) % n]
            # Ignora arestas horizontais
            if y1 == y2:
                continue
            # Define o lado inferior e superior da aresta
            if y1 < y2:
                y_low, y_high = y1, y2
                x_low, x_high = x1, x2
            else:
                y_low, y_high = y2, y1
                x_low, x_high = x2, x1
            # A interseção ocorre se y estiver entre y_low (inclusive) e y_high (exclusiva)
            if y_low <= y < y_high:
                # Interpolação linear para determinar a coordenada x de interseção
                x_int = x_low + (y - y_low) * (x_high - x_low) / (y_high - y_low)
                intersections.append(x_int)
        intersections.sort()
        # Preenche os intervalos entre pares de interseções
        for i in range(0, len(intersections), 2):
            if i + 1 < len(intersections):
                x_start = int(np.ceil(intersections[i]))
                x_end = int(np.floor(intersections[i + 1]))
                for x in range(x_start, x_end + 1):
                    if arr[y, x] != 0:  # não sobrescreve a borda
                        arr[y, x] = fill_color
```

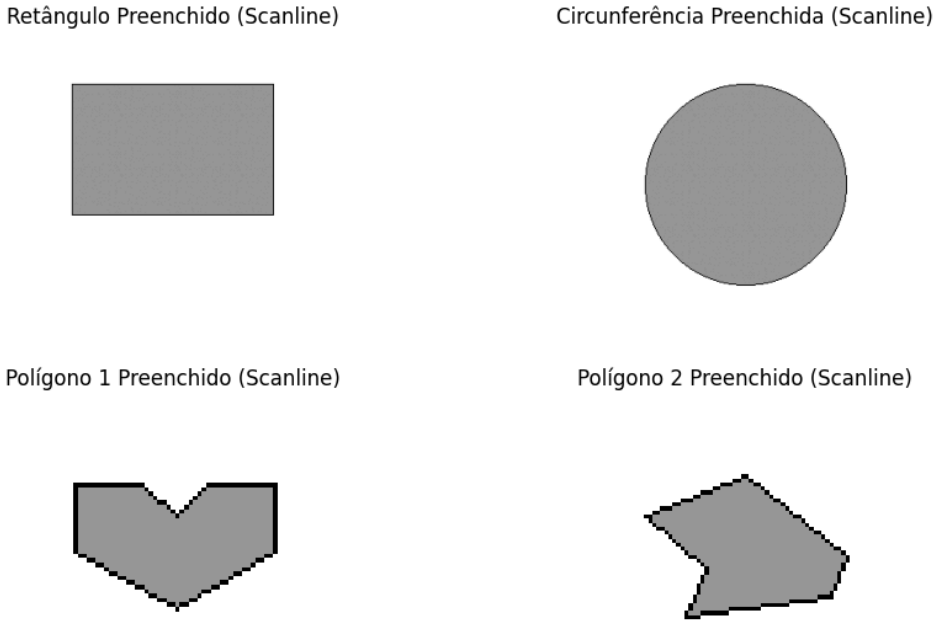
4. RESULTADOS COMPARATIVOS

Os dois métodos foram aplicados para preencher as mesmas formas e os resultados foram comparados:

- Flood Fill Recursivo:



- Algoritmo de Varredura (Scanline):



A abordagem recursiva é conceitualmente simples e intuitiva, porém, para áreas grandes pode demandar um gerenciamento cuidadoso da recursão (ou seja, ajustes no limite de recursão). Nesta implementação, as formas foram dimensionadas de forma a evitar *stack overflow*.
O algoritmo de varredura, por sua vez, oferece eficiência computacional e maior controle geométrico, mas pode ser mais complexo de implementar para formas irregulares.
Ambos os métodos foram capazes de preencher corretamente as áreas internas das formas, preservando as bordas desenhadas.