PODER EXECUTIVO
MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE RORAIMA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

DCC703 - COMPUTAÇÃO GRÁFICA

RELATÓRIO DO PROJETO DE PREENCHIMENTO DE FORMAS GEOMÉTRICAS

ALUNOS:
Marcos Vinícius Tenacol Coêlho - 2021000759

Fevereiro de 2025
Boa Vista/Roraima

Resumo

Este trabalho explora a implementação e demonstração de diversos algoritmos para preenchimento de formas geométricas em gráficos computacionais.

O trabalho inclui dois **algoritmos de preenchimento**:

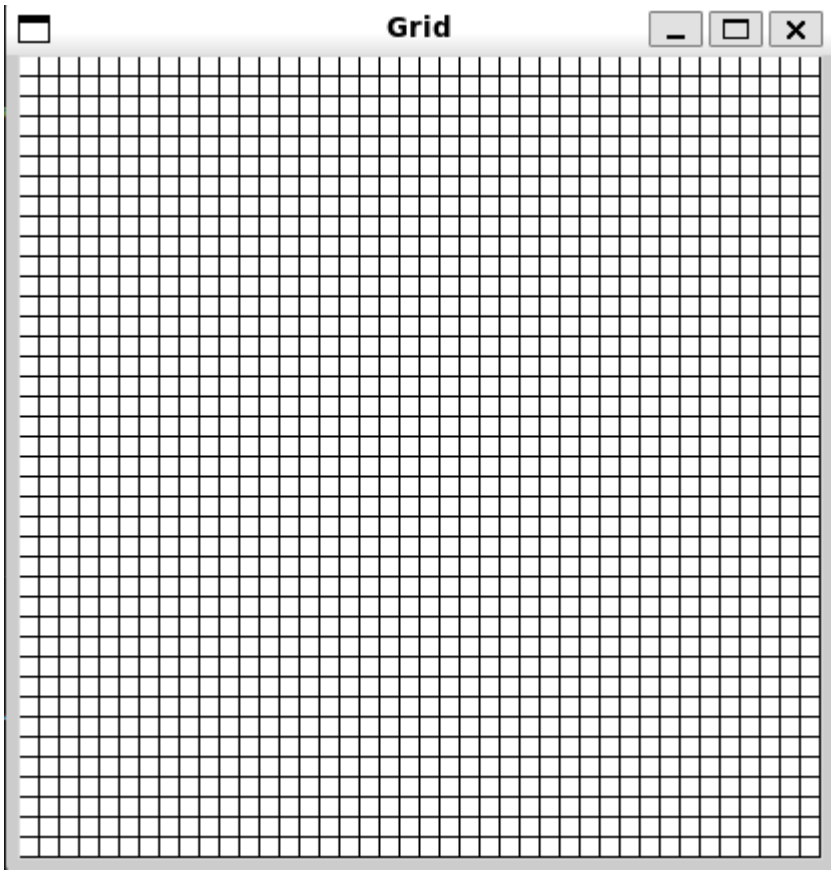
1. **Flood Fill**, utilizado para preencher áreas conectadas a partir de um ponto inicial, semelhante à ferramenta "balde de tinta" em softwares gráficos;
2. **Preenchimento por Varredura**, que preenche formas linha por linha, garantindo eficiência em polígonos mais complexos.

Os algoritmos de preenchimento são eficazes na tarefa de preencher áreas delimitadas pelas formas rasterizadas, completando o ciclo de desenho e renderização de gráficos básicos.

Implementações

Para realizar a criação do grid de pixels, utilizei a biblioteca **Tkinter**, que oferece uma interface gráfica simples e eficiente para manipulação de elementos visuais. O **Tkinter** foi empregado tanto para criar o grid de pixels quanto para possibilitar a interação com ele, permitindo a "pintura" de pixels com base nos algoritmos de rasterização.

Demonstração do grid 40x40 gerado:



Método de preenchimento flood fill

O método Flood Fill é um algoritmo usado para preencher uma área dentro de uma forma fechada, começando de um ponto específico. Ele funciona de forma parecida com a ferramenta de "balde de tinta" em programas de desenho: quando você clica em um ponto, o algoritmo preenche todos os pixels conectados a esse ponto que têm a mesma cor.

O funcionamento é simples: o algoritmo verifica os pixels ao redor do ponto inicial e preenche aqueles que ainda não foram preenchidos. Esse processo continua até que toda a área conectada esteja preenchida.

O código utilizado para flood_fill foi:

```
def getPixel(x,y):
    return canvas.find_overlapping(
        x * PIXEL_SIZE, y * PIXEL_SIZE, (x + 1) * PIXEL_SIZE, (y + 1) * PIXEL_SIZE
    )

def flood_fill( x, y, cor_alvo, nova_cor = "black"):
    objetos = getPixel(x,y)

    if not objetos:
        return
    cor_atual = canvas.itemcget(objetos[-1], "fill")

    if cor_atual != cor_alvo:
        return

    pintar(x, y, nova_cor)

    flood_fill( x + 1, y, cor_alvo, nova_cor)
    flood_fill(x - 1, y, cor_alvo, nova_cor)
    flood_fill( x, y + 1, cor_alvo, nova_cor)
    flood_fill(x, y - 1, cor_alvo, nova_cor)
```

1º Passo. Buscar o pixel atual;

2º Passo. Pegar a cor do pixel atual;

3º Passo. Checar se a cor do pixel atual é igual a cor_alvo(cor a ser modificada);

4º Passo. Pintar o pixel e chama recursivamente a função novamente para pintar os 4 pixels adjacentes.

Método de preenchimento por varredura com análise geométrica

O método de preenchimento por varredura com análise geométrica é uma técnica usada para preencher áreas dentro de formas fechadas, como polígonos. O algoritmo funciona desenhando linhas horizontais (varreduras) de um lado da forma até o outro. Cada vez que a linha cruza a borda da forma, o algoritmo alterna entre "preencher" e "não preencher", garantindo que apenas a área interna seja colorida.

Esse método é eficiente porque preenche a área linha por linha, usando cálculos simples para encontrar os pontos de interseção com as bordas da forma.

Código utilizado para varredura de retângulo:

```python
def varredura_retangulo(x1,y1,x2,y2,cor="black"):
    objetos = getPixel(x1, y1)

    if not objetos:
        return

    cor_borda = canvas.itemcget(objetos[-1], "fill")

    for y in range(y1 + 1, y2):
        for x in range(x1 + 1, x2):
            cor_atual = canvas.itemcget(getPixel(x, y)[-1], "fill")
            if cor_atual != cor_borda:
                pintar(x, y, cor)
```

1º Passo. Pegar a cor da borda do retângulo;

2º Passo. Começar o processo de checagem de cor a partir do primeiro pixel vazio;

3º Passo. Checar se a cor do pixel atual é diferente da cor da borda ou se a cor de preenchimento é igual a cor da borda;

4º Passo. Pintar a coordenada.

Código utilizado para varredura de circunferência:

```python
def varredura_circulo(xc,yc,r,cor="black"):
    y =  yc-r
    while(yc-r<=y<=yc+r):
        x1 = round(xc - math.sqrt(r**2 - (y - yc)**2))
        x2 = round(xc + math.sqrt(r**2 - (y - yc)**2))
        for x in range(x1+1,x2):
            pintar(x,y,cor)
        y+= 1
```

1º Passo. Passar o valor inicial de Y = centro - raio;

2º Passo. Checar se y está dentro da circunferência;

3º Passo. Fazer o cálculo do Xmin e Xmax;

4º Passo. Realizar Pintura e incrementar y.

Código utilizado para varredura geral:

```python
def calcular_intersecoes(poligono, y_scan):
    intersecoes = []
    for linha in poligono:
        x1, y1, x2, y2 = linha

        if y1 != y2 and min(y1, y2) <= y_scan < max(y1, y2):
            x_intersecao = x1 + (y_scan - y1) * (x2 - x1) / (y2 - y1)
            intersecoes.append(round(x_intersecao))

    return sorted(intersecoes)


def varredura(poligono,cor = "black"):
    ymin = min(y for _, y, _, _ in poligono)
    ymin2 = min(y for _, _,_,y in poligono)
    ymax = max(y for _, y, _, _ in poligono)
    ymax2 = max(y for _, _, _, y in poligono)

    if(ymax2>ymax):
        ymax = ymax2
    if(ymin2<ymin):
        ymin = ymin2


    for y_scan in range(ymin, ymax + 1):
        intersecoes = calcular_intersecoes(poligono, y_scan)

        if len(intersecoes) % 2 != 0:
            continue

        for i in range(0, len(intersecoes), 2):
            x_inicio = int(min(intersecoes[i], intersecoes[i + 1]))
            x_fim = int(max(intersecoes[i], intersecoes[i + 1]))

            for x in range(x_inicio, x_fim + 1):
                pintar(x, y_scan,cor)
```

1º Passo. Pegar o valor mínimo e máximo de y, dentro da lista de linhas;

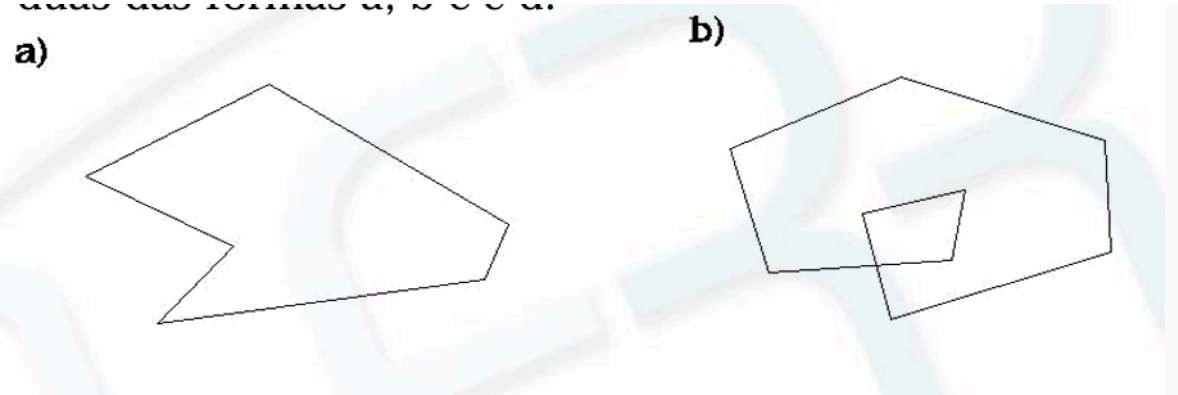
2º Passo. Checar as interseções com base no y atual(y_scan);

3º Passo. Checar se as interseções são par, já que não podem ser ímpar se não gerará erro;

4º Passo. Pintar as coordenadas variando entre os X adquiridos através da checagem de interseções.

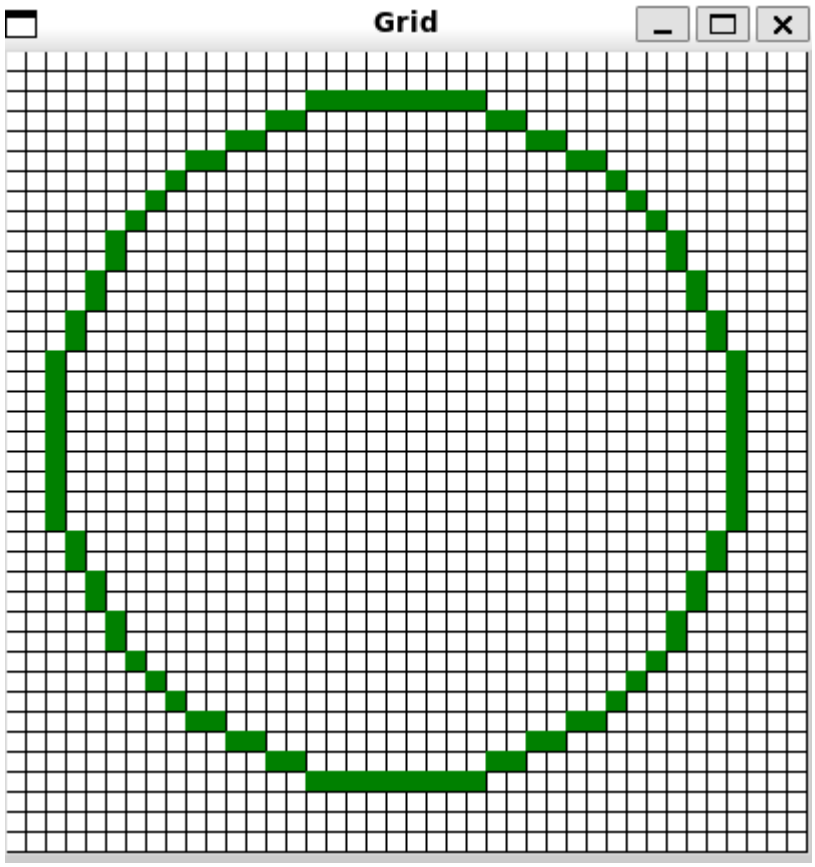
Preenchimento das formas

O objetivo do trabalho foi preencher quatro formas: um retângulo, uma circunferência e duas das seguintes formas:



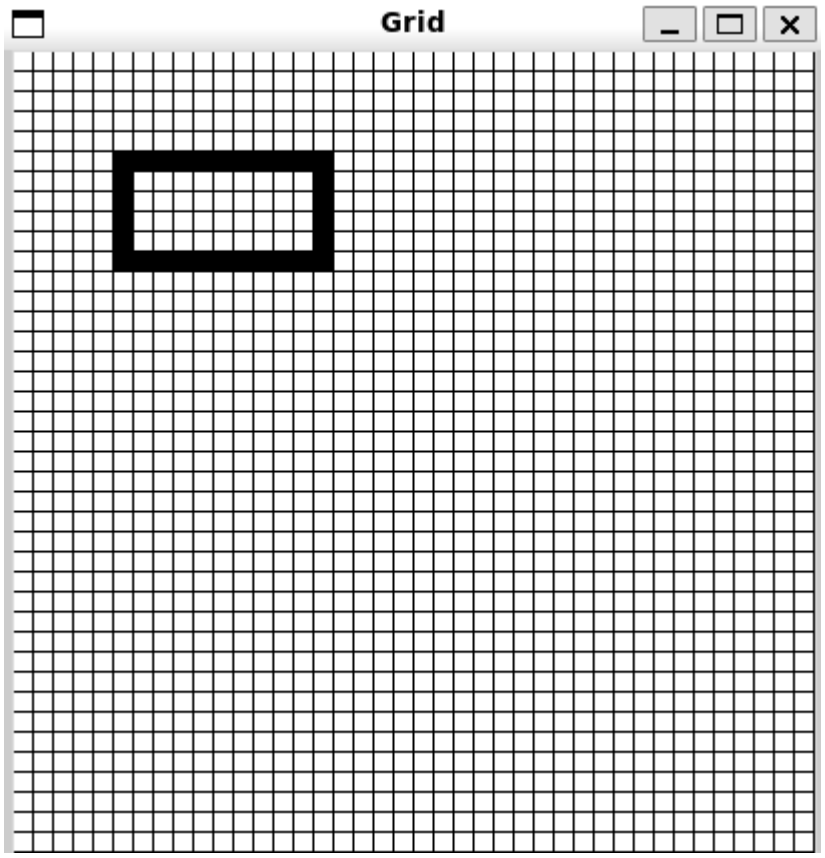
As formas foram rasterizadas usando algoritmo de bresenham de linha e de circunferência, chamados da seguinte forma, usando as seguintes coordenadas:

```
# Círculo

circ_bresenham(19, 19, 17, "green")
```



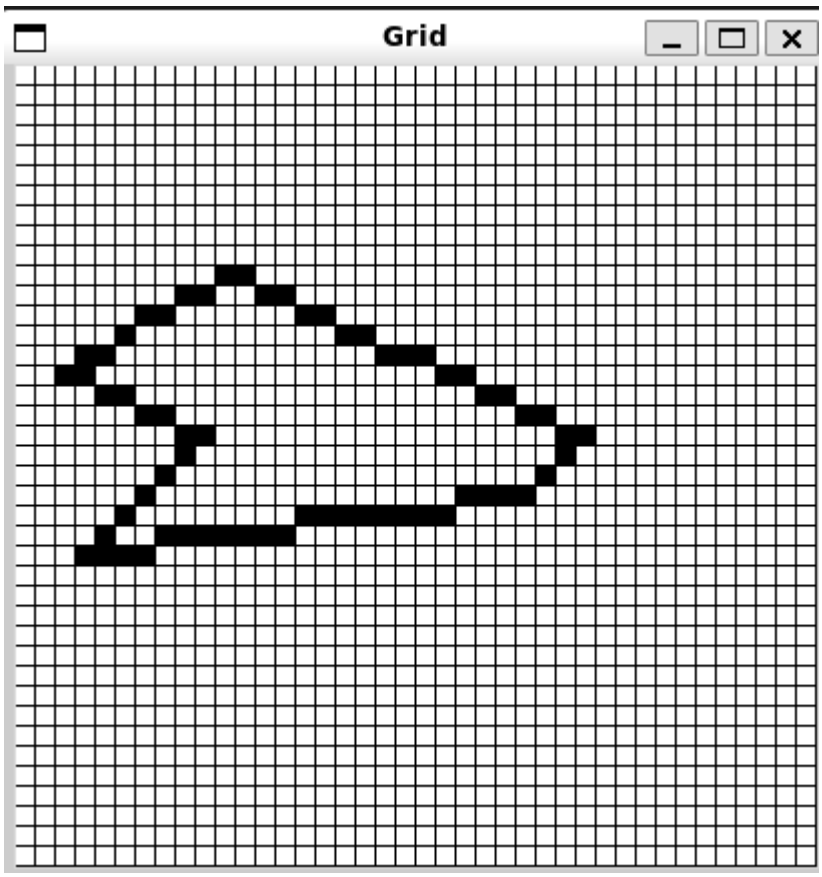

```
# Retângulo

bresenham(5, 5, 15, 5, "black")

bresenham( 5, 10, 15, 10, "black")
```

```
bresenham( 5, 5, 5, 10, "black")

bresenham( 15, 5, 15, 10, "black")
```

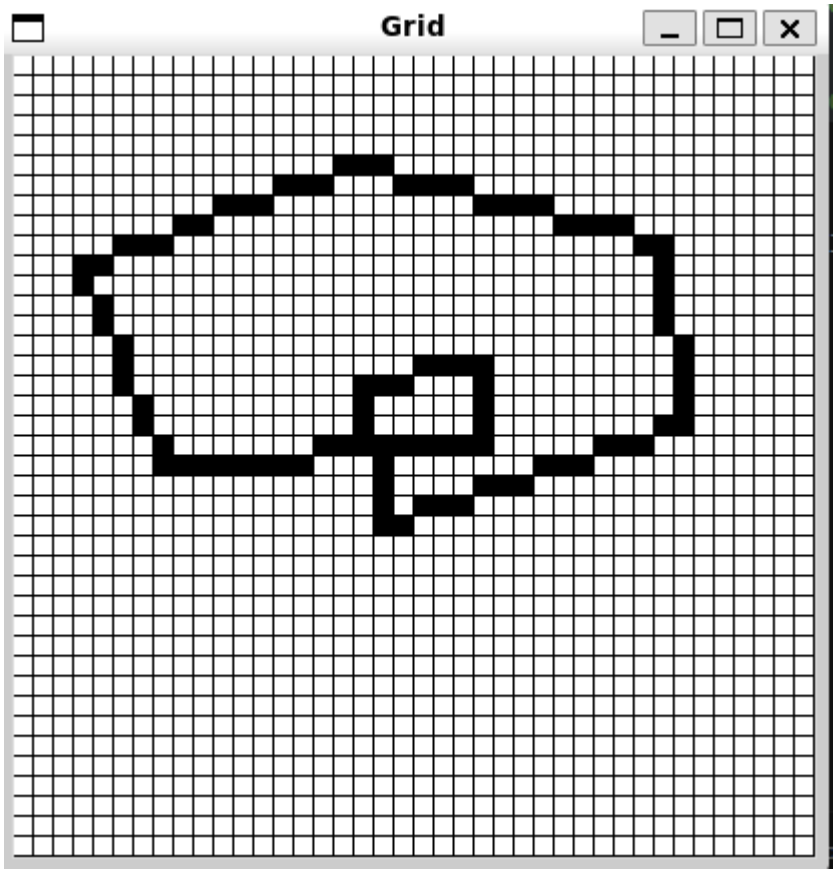


Nas formas A e B, foi necessário criar uma lista com as coordenadas das linhas para executar o algoritmo de varredura.

```
# Forma a)

bresenham( 10, 10, 28, 18)

bresenham( 10, 10, 2, 15)

bresenham( 9, 18, 2, 15)

bresenham(9, 18, 3, 24)

bresenham( 25, 21, 28, 18)

bresenham( 25, 21, 3, 24)

poligono1 = [(10, 10, 28, 18), (10, 10, 2, 15), (9, 18, 2, 15), (9,
18, 3, 24), (25, 21, 28, 18), (25, 21, 3, 24)]
```




```
# Forma b)

bresenham(3,10,7,20)

bresenham(23,19,7,20)

bresenham(23,19,23,15)

bresenham(17,16,23,15)

bresenham(17,16,18,23)

bresenham(33,18,18,23)

bresenham(33,18,32,9)

bresenham(17,5,32,9)

bresenham(17,5,3,10)

poligono2 = [ (3,10,7,20), (23,19,7,20), (23,19,23,15), (17,16,23,15),
(17,16,18,23), (33,18,18,23),(33,18,32,9), (17,5,32,9), (17,5,3,10)]
```



Os funcionamentos dos algoritmos resultaram em:

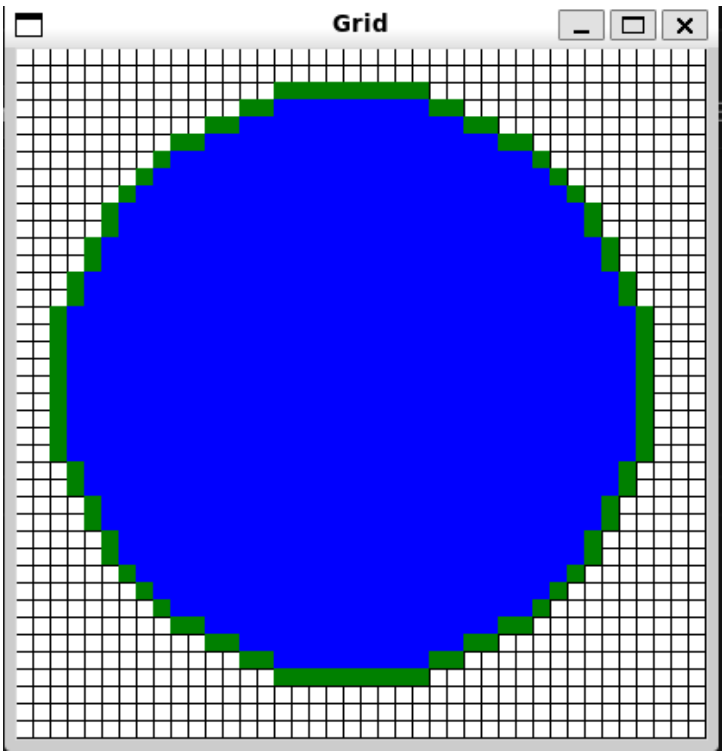
Circunferência:

Flood fill:

Chamada utilizada:

```
flood_fill(19, 19, "white", "blue")
```

Resultado:

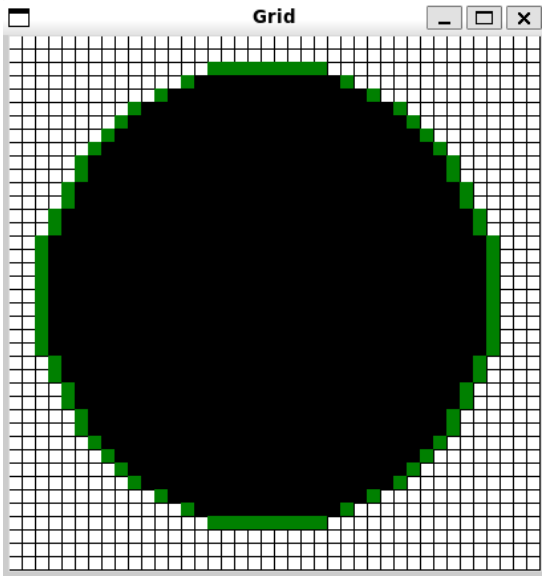


Varredura:

Chamada utilizada:

```
varredura_circulo(19, 19, 17,"black")
```

Resultado:



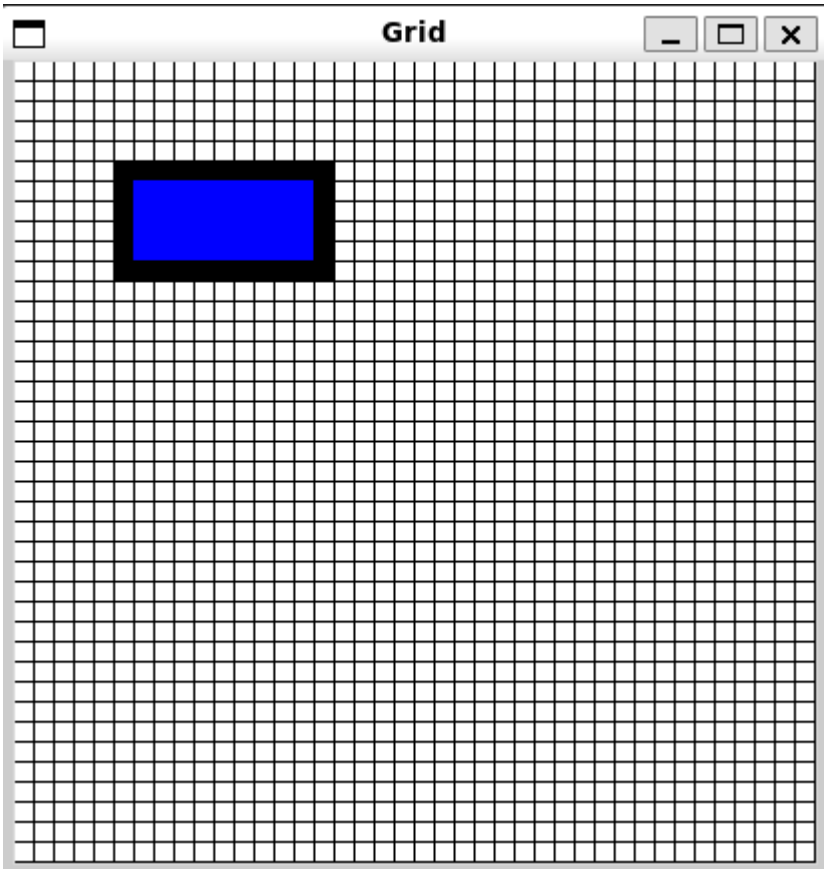
Retângulo:

Flood Fill:

Chamada Utilizada:

```
flood_fill(6,6,"white","blue")
```

Resultado:

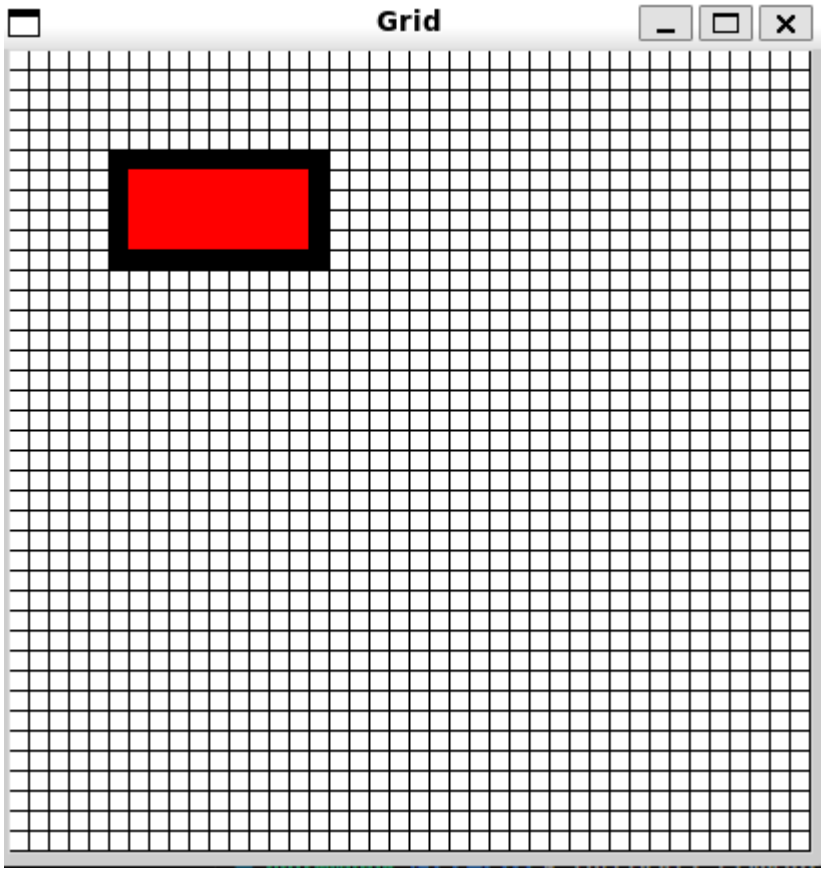


Varredura:

Chamada Utilizada:

```
varredura_retangulo(5,5,15,10,'red')
```

Resultado:



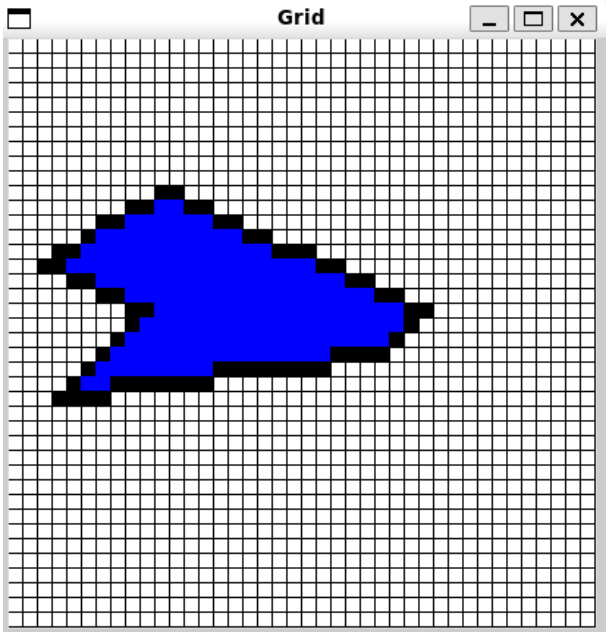
Forma a:

Flood Fill:

Chamada Utilizada:

```
flood_fill(4,15,"white","blue")
```

Resultado:

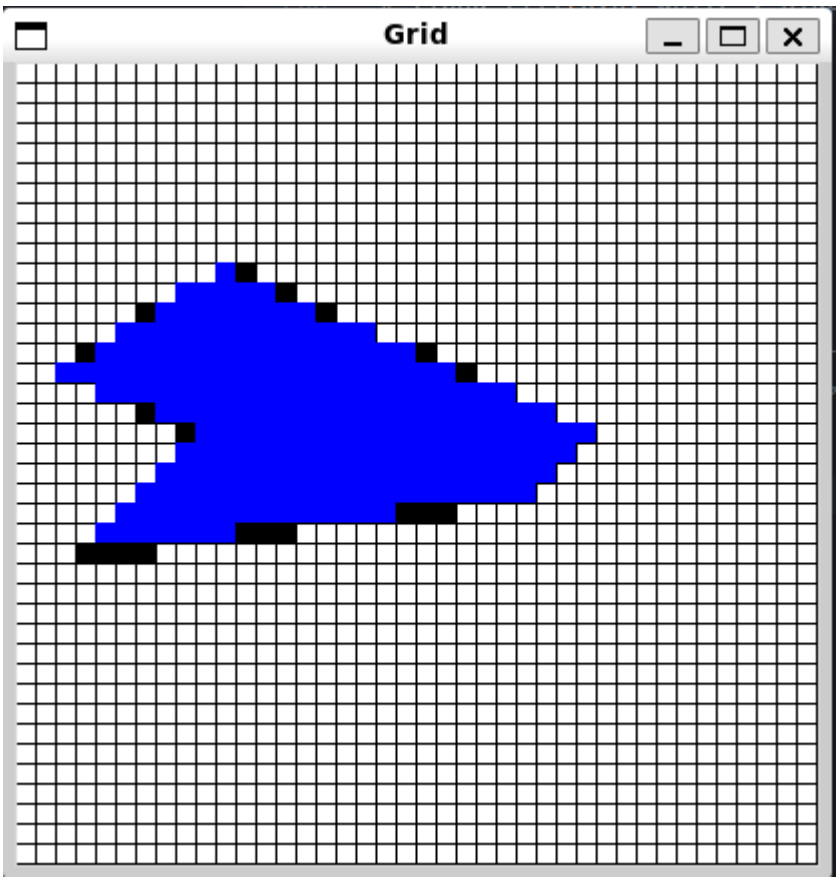


Varredura:

Chamada Utilizada:

```
varredura(poligono1, 'blue' )
```

Resultado:



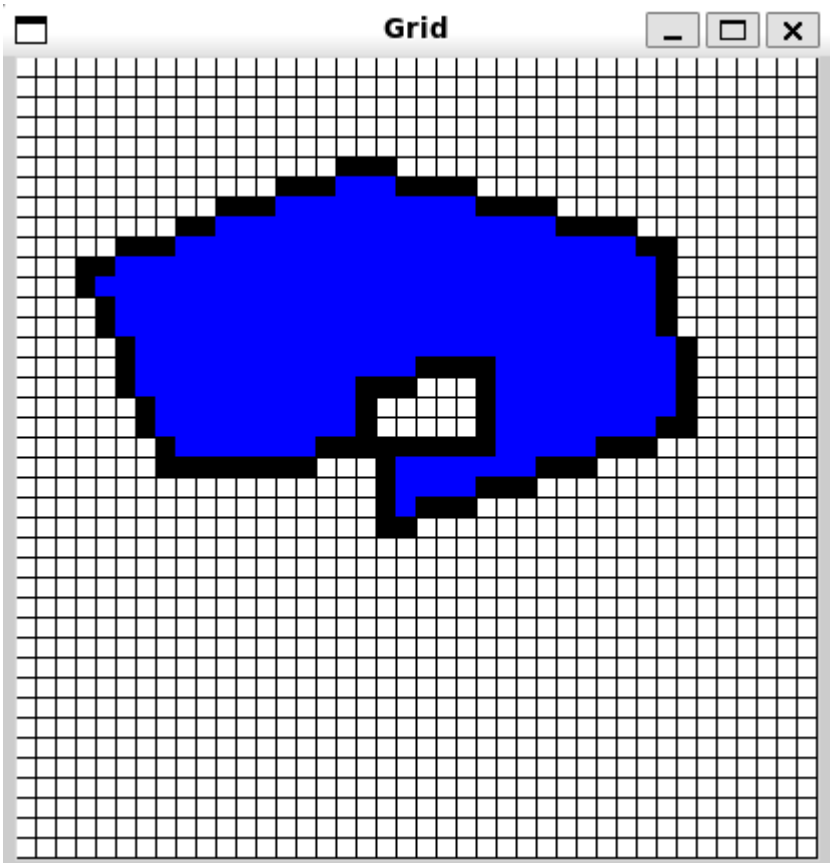
Forma b:

Flood Fill:

Chamada Utilizada:

```
flood_fill(4,15,"white","blue")
```

Resultado:

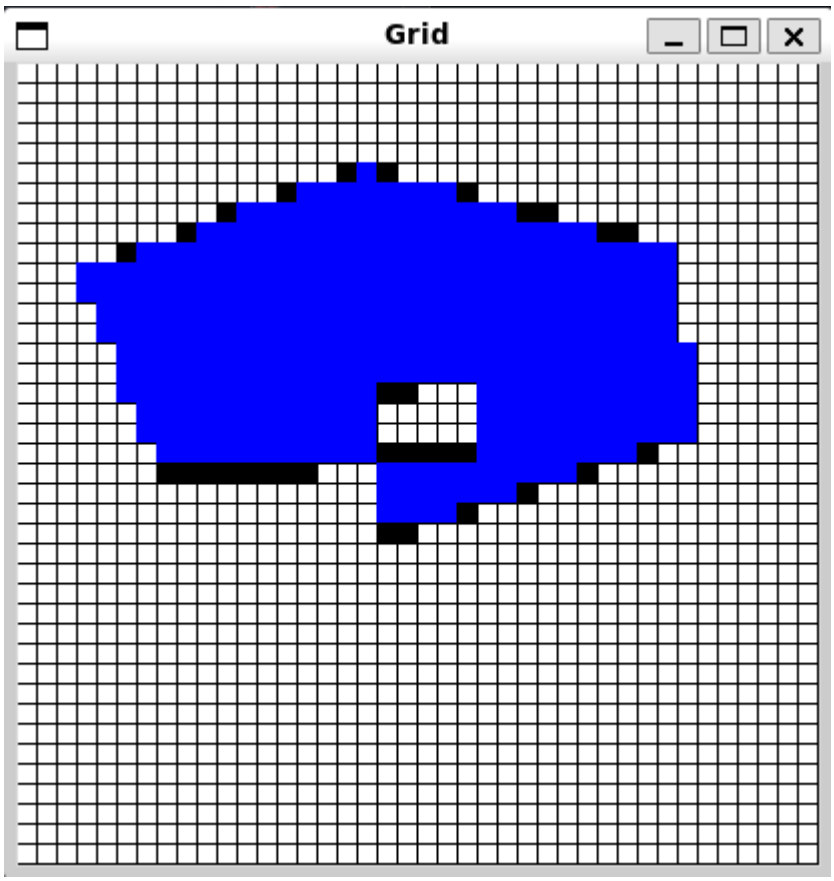


Varredura:

Chamada Utilizada:

```
varredura(poligono2, 'blue')
```

Resultado:


Grid