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

DCC703 - COMPUTAÇÃO GRÁFICA

RELATÓRIO DO PROJETO DE RASTERIZAÇÃO DE CIRCUNFERÊNCIA

ALUNOS:
Marcos Vinícius Tenacol Coêlho - 2021000759

Fevereiro de 2025
Boa Vista/Roraima

Resumo

Este trabalho explora a implementação e demonstração de diversos algoritmos fundamentais para a rasterização de circunferências.

Na **rasterização de circunferências**, foram implementados três métodos:

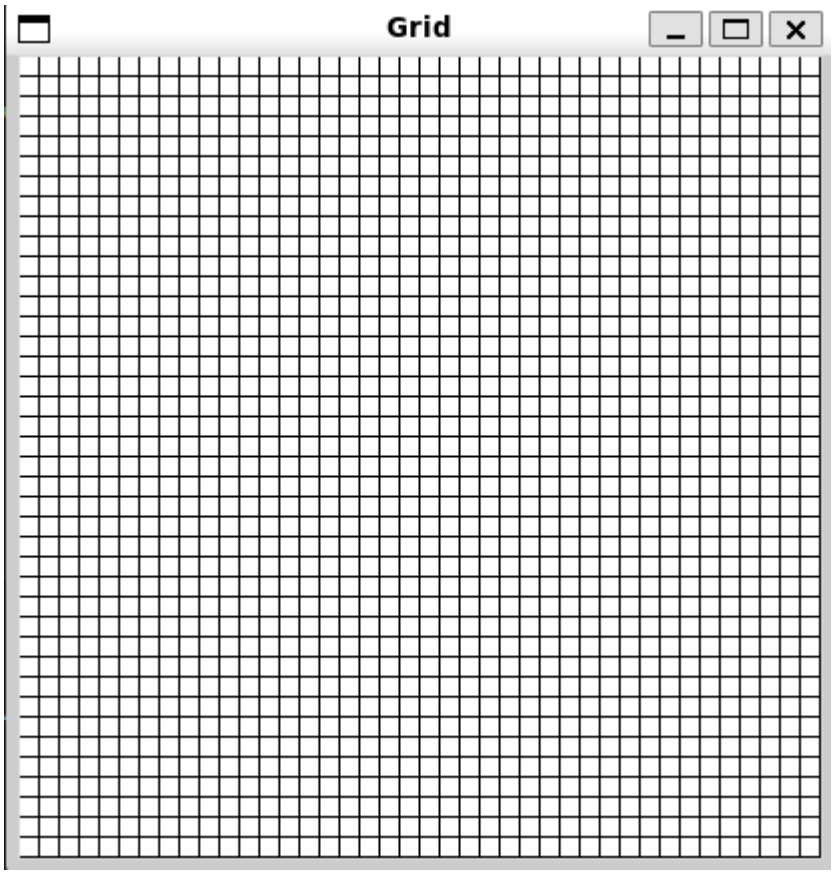
1. **Paramétrico**, que emprega funções trigonométricas para traçar a circunferência;
2. **Incremental com Simetria**, que explora a simetria da circunferência para otimizar o cálculo dos pontos;
3. **Bresenham**, adaptado para circunferências, oferecendo eficiência ao evitar cálculos de ponto flutuante.

Todos os algoritmos de rasterização implementados são capazes de traçar circunferências.

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 rasterização de circunferência paramétrico

O método paramétrico para a rasterização de circunferências utiliza uma abordagem baseada em parametrização trigonométrica para determinar os pontos ao longo da circunferência. A equação paramétrica de uma circunferência com centro em (xc, yc) e raio r é dada por:

$$x = x_c + r \cdot \cos(t)$$

$$y = y_c + r \cdot \text{sen}(t)$$

Onde θ varia de 0 a 360 graus. A cada valor de θ, são calculadas as coordenadas (x, y) que correspondem a

os pontos na circunferência. Para a rasterização, os valores de x e y são arredondados para os valores inteiros mais próximos, representando os pontos no grid de pixels.
Implementações de circunferências com raio muito grande pode gerar falhas.

O pseudo-código usado como base foi:

x = x_c + raio **y = y_c**

para t de 1 até 360 com passo "t"

pixel (x , y , cor)

$$x = x_c + r \cdot \cos\left(\frac{\pi \cdot t}{180}\right)$$

$$y = y_c + r \cdot sen\left(\frac{\pi \cdot t}{180}\right)$$

Código Utilizado:

```
def parametrica(xc, yc, r, cor="black"):
    x = xc + r
    y = yc
    for t in range(1, 360):
        pintar(round(x), round(y),cor)
        x = xc + r * math.cos((math.pi * t) / 180)
        y = yc + r * math.sin((math.pi * t) / 180)
```

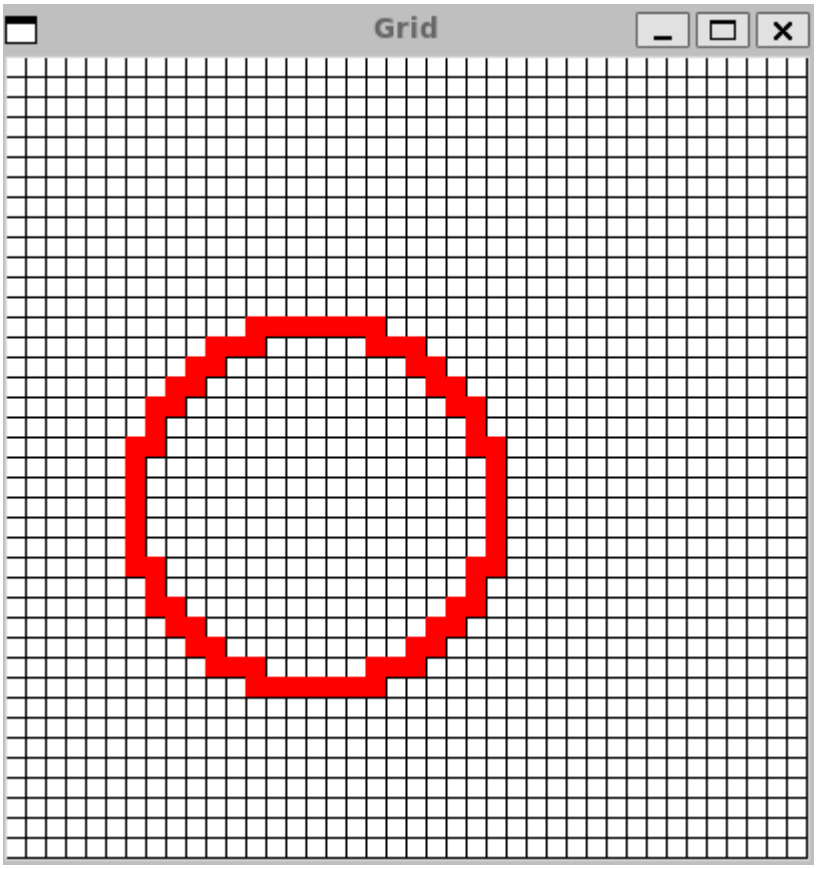
1º Passo. Configurar coordenadas inicias.

2º Passo. Iniciar loop de pintura.

3º Passo. Recalcular as novas coordenadas de x e y.

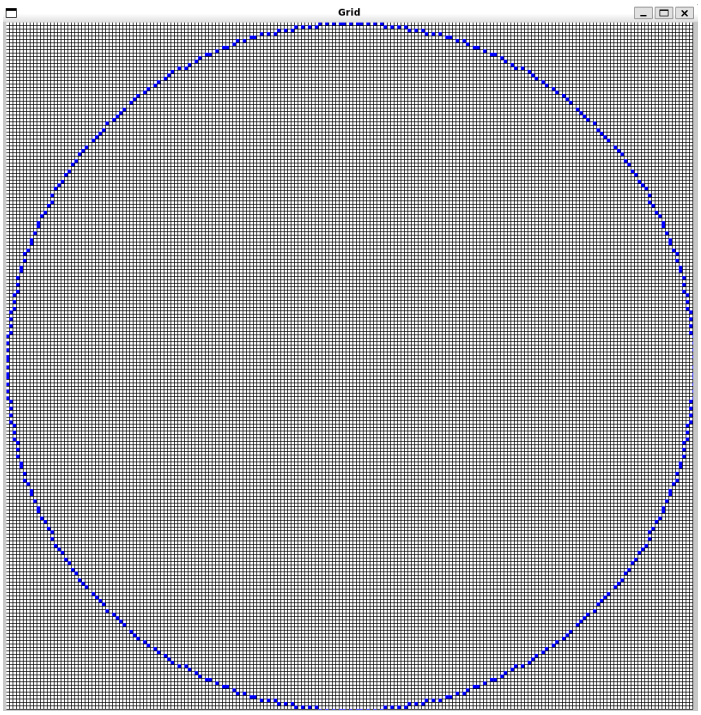
Demonstração do chamada para:

- parametrica(15, 22, 9, "red")



Demonstração de uma circunferência com raio muito grande:

- parametrica(100, 100, 100, "blue")



Método de rasterização de circunferência incremental com simetria

O método incremental com simetria para a rasterização de circunferências é uma otimização do método paramétrico que aproveita a simetria da circunferência para reduzir os cálculos necessários. Uma circunferência possui simetria em oito quadrantes, ou seja, se você conhece os pontos de um quadrante, pode calcular os pontos correspondentes nos outros sete quadrantes utilizando simples transformações.

A ideia central do algoritmo é calcular os pontos de um quadrante e, em seguida, usar a simetria para gerar os pontos nos outros quadrantes sem precisar calcular diretamente as coordenadas para todos os pontos da circunferência. Isso é feito através de incrementos sucessivos baseados na equação da circunferência, utilizando apenas operações inteiras para decidir a posição dos próximos pontos.

O código foi baseado em:

$$x_{k+1} = x_k \cdot \cos\theta - y_k \cdot sen\theta$$

$$y_{k+1} = y_k \cdot \cos\theta + x_k \cdot sen\theta$$

considerando que senθ e cosθ é são valores estáticos e θ = 1/r.

Código utilizado:

```
def incremental_simetria(xc, yc, r, cor="black"):
    theta = 1 / r
    ct = math.cos(theta)
    st = math.sin(theta)
    x = 0
    y = r

    while y >= x:
        pintar(round(xc + x), round(yc + y), cor)
        pintar(round(xc - x), round(yc + y), cor)
        pintar(round(xc + x), round(yc - y), cor)
        pintar(round(xc - x), round(yc - y), cor)
        pintar(round(yc + y), round(xc + x), cor)
        pintar(round(yc + y), round(xc - x), cor)
        pintar(round(yc - y), round(xc + x), cor)
        pintar(round(yc - y), round(xc - x), cor)
        x_novo = x * ct - y * st
        y_novo = x * st + y * ct
        x, y = x_novo, y_novo
```

1º Passo. Adquirir o valor de theta e calcular os valores estáticos.

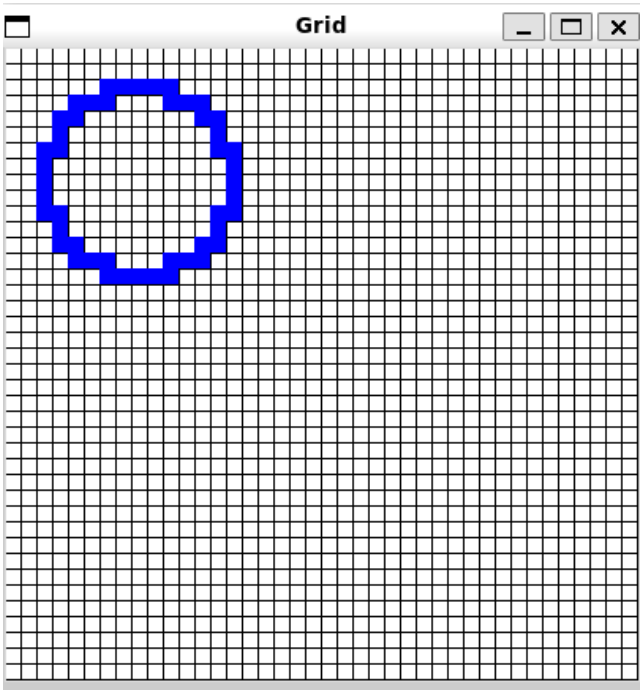
2º Passo. Configurar as coordenadas iniciais.

3º Passo. Pintar as coordenadas seguindo as condicionais.

4º Passo. Recalcular os novos valores de x e y.

Demonstração de chamada para:

- incremental_simetria(8, 8, 6, "blue")



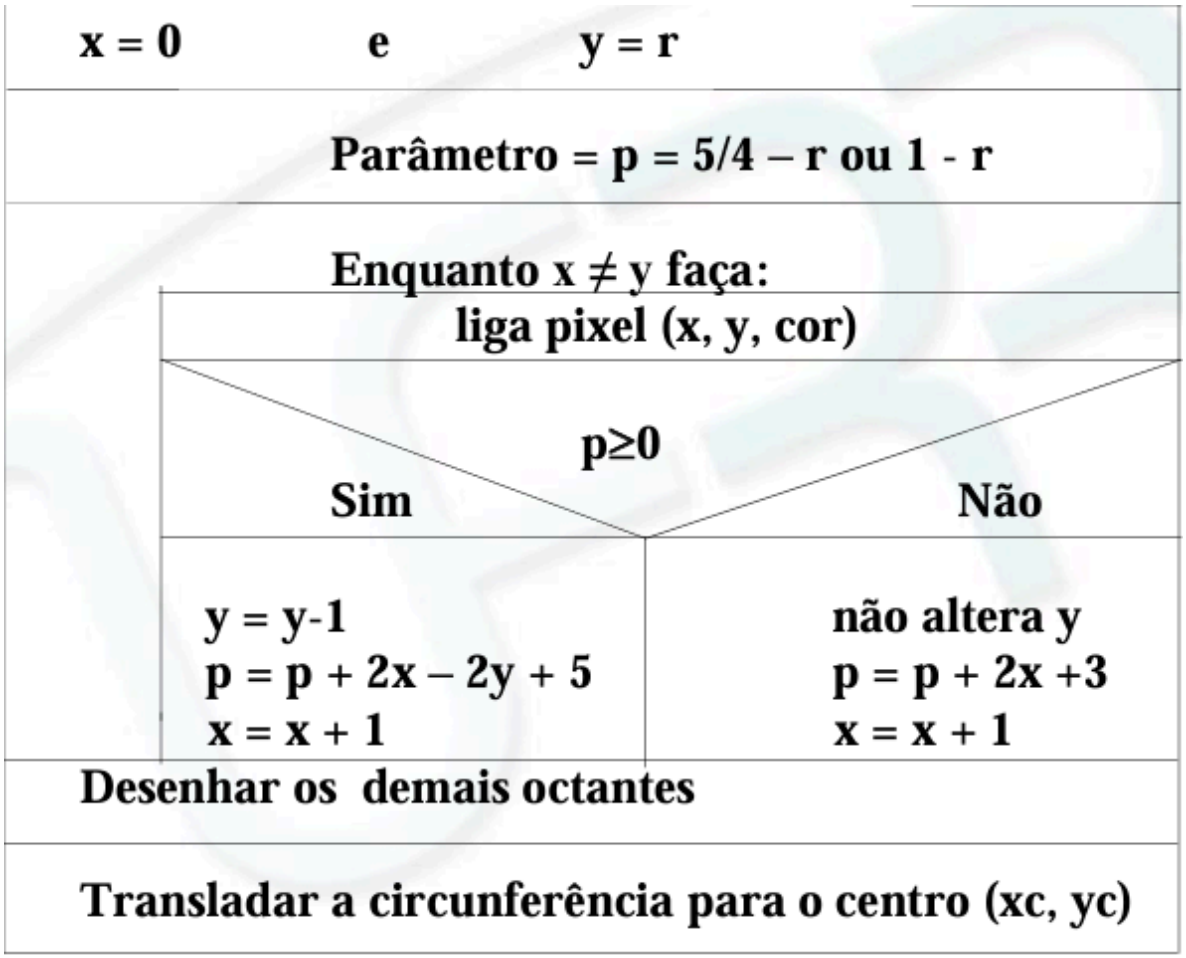
Método de rasterização de circunferência de Bresenham

O método de rasterização de circunferência de Bresenham é uma adaptação do algoritmo de Bresenham, originalmente desenvolvido para linhas, aplicada à rasterização de circunferências. Assim como no algoritmo de Bresenham para linhas, ele utiliza apenas operações inteiras para determinar os pontos da circunferência, o que torna o processo mais eficiente, especialmente em sistemas com limitações de desempenho.

O algoritmo começa a partir do ponto inicial e, a cada iteração, decide qual dos dois possíveis pontos mais próximos à circunferência deve ser ativado, com base em uma decisão de erro calculada entre os pontos possíveis. A principal vantagem desse método é que ele evita o uso de operações de ponto flutuante, utilizando apenas inteiros, o que reduz o custo computacional.

O funcionamento desse método tem como base o método de rasterização de simetria, espelhando os demais octantes.

O pseudo-código usado como base foi, tendo uma pequena alteração na condicional de loop while(x!=y) para while(x>=y):



Códigos da circunferência de Bresenham:

```python
def circ_bresenham(xc, yc, r, cor="black"):
    x = 0
    y = r
    p = 1 - r
    while y>=x:
        pintar(round(xc + x), round(yc + y), cor)
        pintar(round(xc - x), round(yc + y), cor)
        pintar(round(xc + x), round(yc - y), cor)
        pintar(round(xc - x), round(yc - y), cor)
        pintar(round(yc + y), round(xc + x), cor)
        pintar(round(yc + y), round(xc - x), cor)
        pintar(round(yc - y), round(xc + x), cor)
        pintar(round(yc - y), round(xc - x), cor)
        if (p >= 0):
            y = y - 1
            p = p + (2 * x) - (2 * y) + 5
            x = x + 1
        else:
            p = p + (2 * x) + 3
            x = x + 1
```

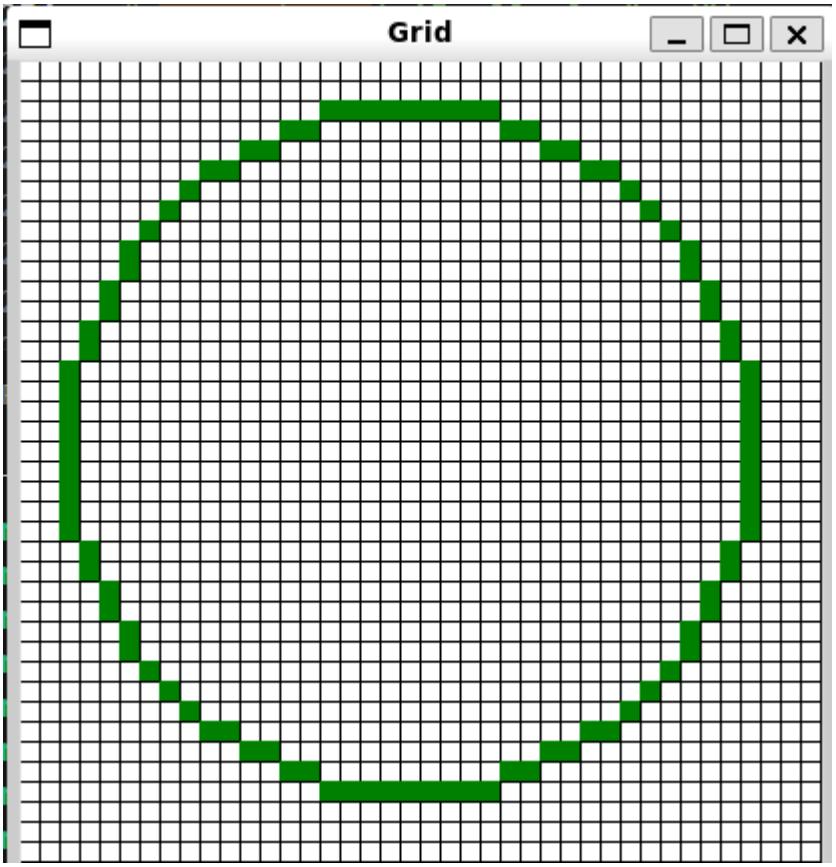
1º Passo. Pegar os valores iniciais e calcula o parâmetro.

2º Passo. Pintar as coordenadas.

3º Passo. Refazer o cálculo das coordenadas com base no parâmetro.

Demonstrações de chamada para:

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


Grid