UNIVERSIDADE FEDERAL DE RORAIMA CENTRO DE CIÊNCIA E TECNOLOGIA BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
DCC703 – COMPUTAÇÃO GRÁFICA (2024.2)

Data de entrega: 07/02/2025

DISCENTES:
FELIPE RUBENS DE SOUSA BORGES (2020020120)

COMPUTAÇÃO GRÁFICA

Relatório Rasterização de Circunferências

Relatório técnico de acordo com a proposta apresentada pelo docente Prof. Luciano Ferreira relacionado ao projeto de Rasterização de Circunferências da disciplina de Computação Gráfica, que deve desenvolver um programa que permita desenhar circunferências por meio dos algoritmos: Equação Paramétrica, Incremental com Simetria e Bresenham. Além de construir um relatório que descreva a construção e os resultados de maneira comparativa.

Boa Vista-RR
2024.2

Relatório de Rasterização de Circunferências

Relatório apresentado para o projeto de Rasterização de Circunferências da disciplina de Computação Gráfica, ofertada pelo curso de Ciência da Computação da Universidade Federal de Roraima.

Prof. Luciano Ferreira

Boa Vista-RR
2024.2

Resumo

Este relatório descreve a implementação de uma interface gráfica para desenho de circunferências utilizando a biblioteca Pygame, em Python. O programa tem como objetivo desenhar linhas na tela ao selecionar um dos três algoritmos disponíveis: Algoritmo de Bresenham, Incremental com Simetria e a Equação Paramétrica.

Algoritmos

- **Equação Paramétrica**

  A Equação Paramétrica é uma forma de representar curvas e figuras geométricas usando um parâmetro auxiliar, geralmente denotado por t. Em vez de descrever a relação direta entre x e y (como em funções convencionais y = f(x)), cada coordenada é expressa separadamente em função de t:

  x = f(t)
  y = g(t)

  Onde:

  - A equação paramétrica de uma circunferência de raio r, centrada em (h,k), é:

    x = xc + r cos(t)
    y = yc + r sin(t)

  - Onde t varia de 0 a 2π, percorrendo toda a circunferência.

  Como funciona:

  - Varia-se θ de 0 a 360∘, calculando x e y para cada valor e desenhando os pixels correspondentes.

- **Incremental com Simetria**

  O método incremental com simetria é uma otimização na rasterização de circunferências. A ideia principal desse algoritmo é explorar a simetria da circunferência para reduzir os cálculos necessários. Em vez de calcular todos os pontos de um círculo do zero, calculamos apenas 1/8 da circunferência e espelhamos esses pontos para os outros setores.

  - O círculo pode ser gerado incrementando pequenos ângulos.
  - Em vez de calcular diretamente a equação do círculo x2 + y2= r^2, o código usa transformações trigonométricas baseadas em:

    x′ = x.cos(θ) – y.sin(θ)
    y′ = y.cos(θ) + x.sin(θ)

  - Esse método evita o uso de raízes quadradas e cálculos complexos a cada iteração.
  - Como cos θ e sen θ são valores fixos, é possível otimizar os algoritmos.

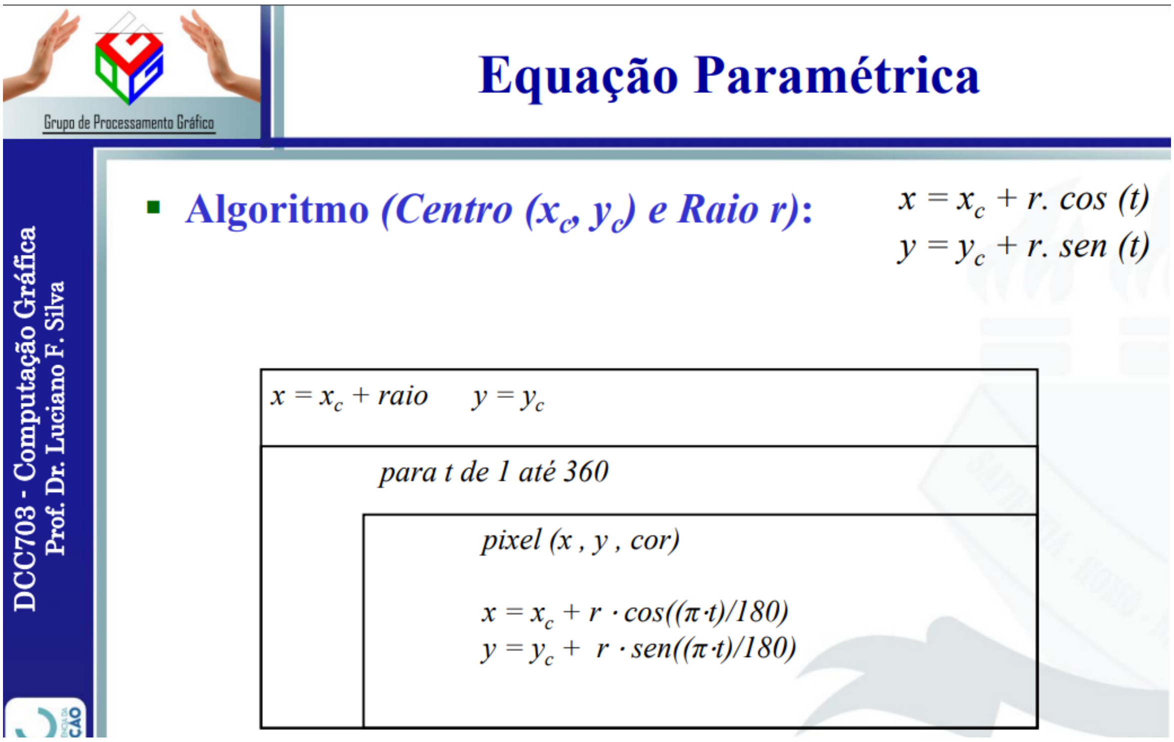
- **Algoritmo de Bresenham**
  O algoritmo de Bresenham para circunferências é uma versão otimizada para rasterização, baseada no método incremental. Ele evita cálculos de raiz quadrada e operações de ponto flutuante, usando apenas adições e subtrações para decidir o próximo ponto a ser desenhado.

  Como Funciona:
  - Inicia-se no topo da circunferência: O primeiro ponto é (0, r); sentido horário ou (r,0); sentido anti-horário.
  - Usa-se um critério de decisão d, que é atualizado em cada iteração:
    1. Se $d < 0$, o próximo ponto está na mesma linha $(x + 1, y)$.
    2. Se $d \geq 0$, o próximo ponto está uma linha abaixo $(x+1, y-1)$.
  - Explora a simetria para desenhar apenas 1/8 do círculo e espelhar os pontos nos outros 7 setores.
  - Repete o processo até $x \geq y$, garantindo a formação completa da circunferência.
  - Outro método também é utilizando a função de circunferência, como parâmetro de decisão, para recair a escolha em dois pixels:

  Fcircle(x,y) = x^2 + y^2 – R^2

Implementação

- **Equação Paramétrica**
  Para implementação da Equação Paramétrica, foi usada como base a fórmula dada nos slides da disciplina:

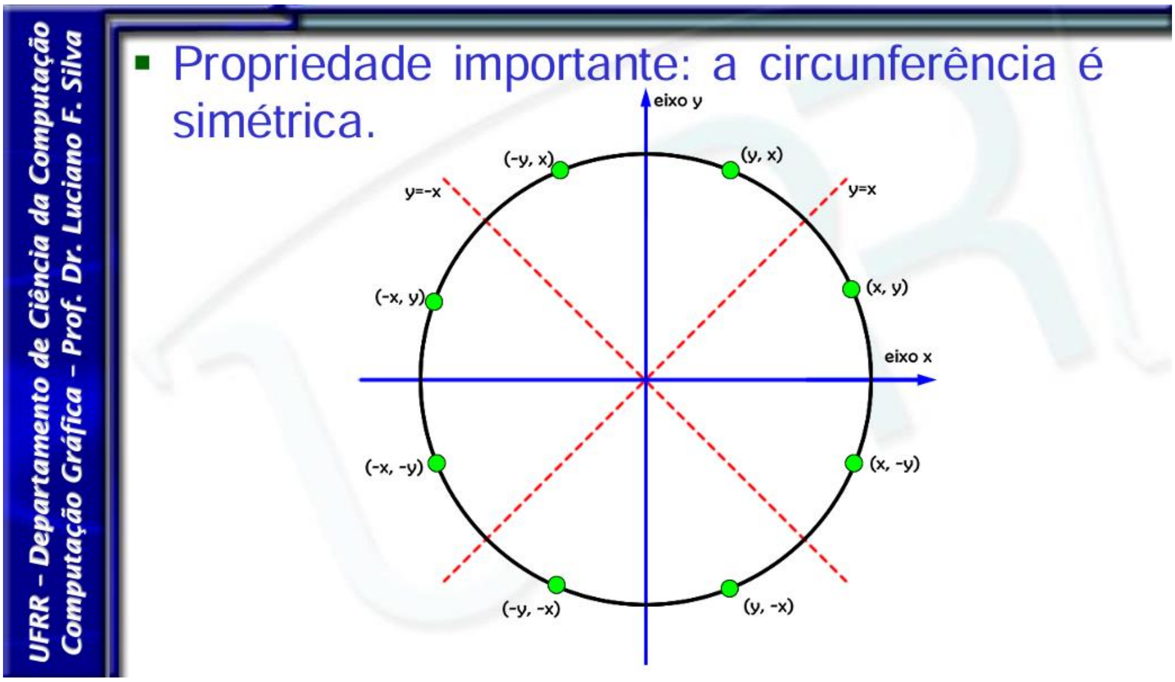


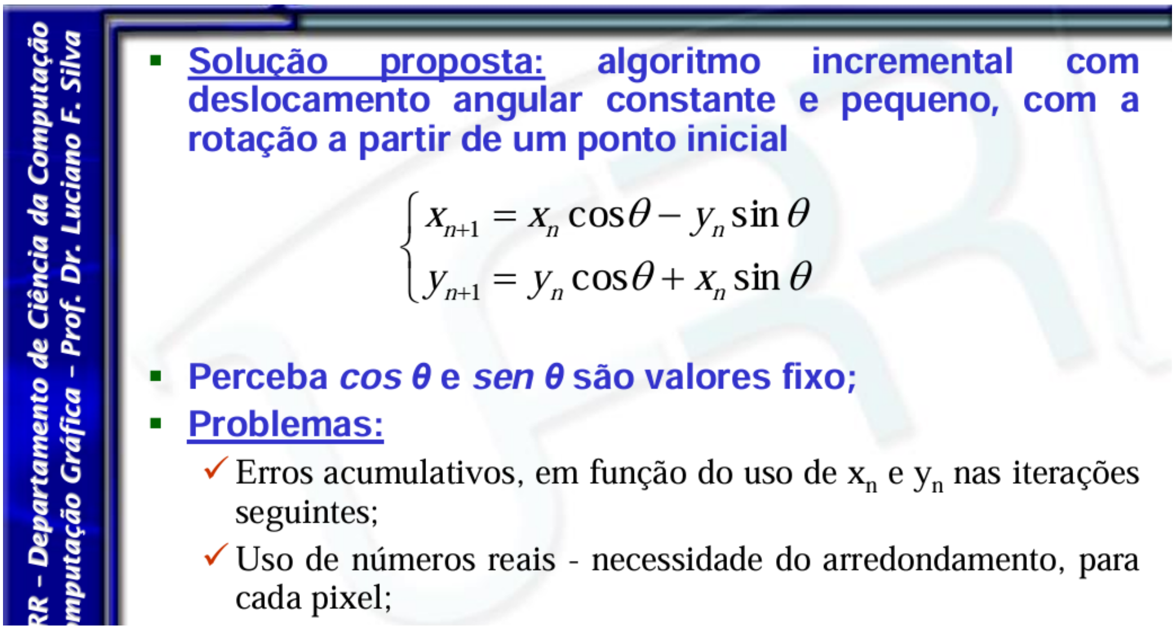
Função no código:

```python
# método Equação Paramétrica
def draw_circ_parametrica(surface, color, center, radius):
    cx, cy = center
    for angle in range(0, 360, 1):
        theta = math.radians(angle)
        x = int(cx + radius * math.cos(theta))
        y = int(cy + radius * math.sin(theta))
        surface.set_at((x, y), color)
        pygame.display.flip()
        pygame.time.delay(5)
```

- ## Incremental com Simetria

  Assim como no método anterior, também foi utilizada como base a fórmula





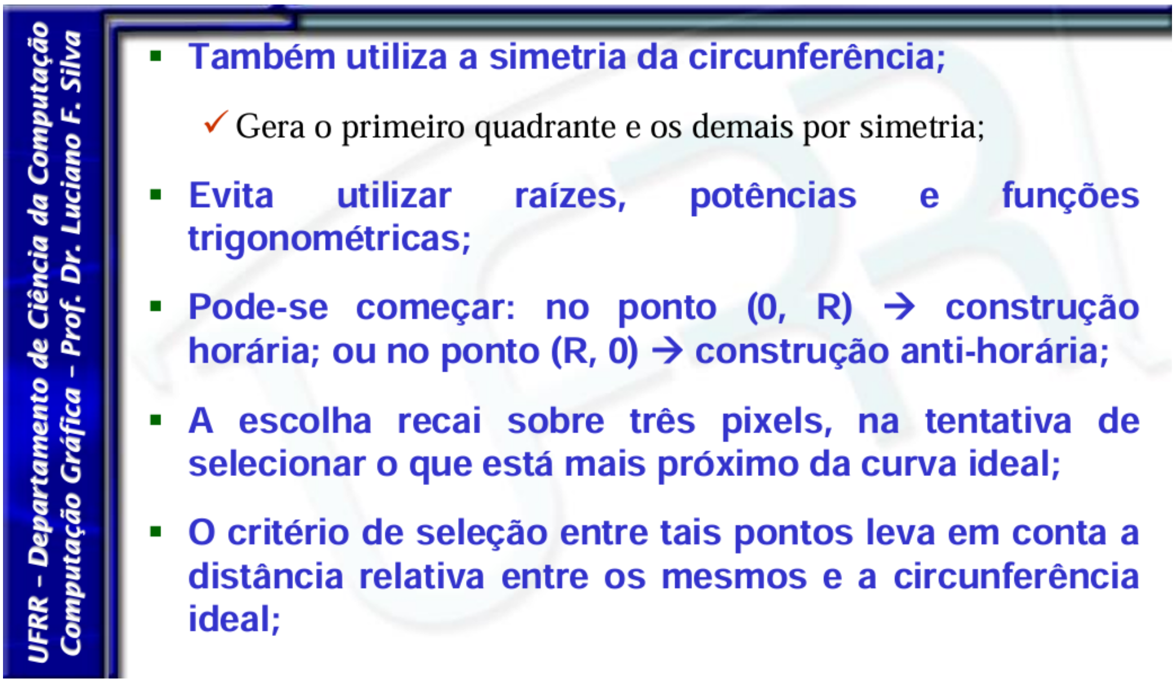
Função no código:

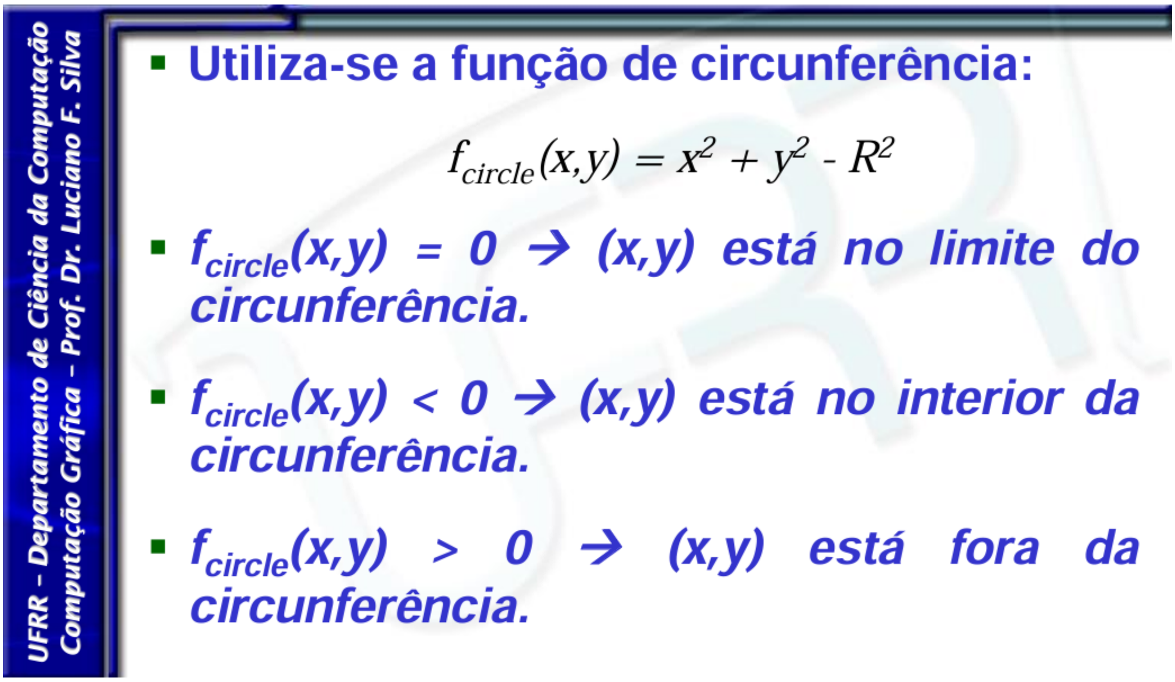
```python
# método Incremental com Simetria
def draw_circ_incremental(surface, color, center, radius):
    cx, cy = center
    x = radius
    y = 0
    theta = 1 / radius  # incremento angular
    C = math.cos(theta)
    S = math.sin(theta)

    while y <= x:
        points = [
            (int(cx + x), int(cy + y)), (int(cx - x), int(cy + y)),
            (int(cx + x), int(cy - y)), (int(cx - x), int(cy - y)),
            (int(cx + y), int(cy + x)), (int(cx - y), int(cy + x)),
            (int(cx + y), int(cy - x)), (int(cx - y), int(cy - x))
        ]
        for point in points:
            surface.set_at(point, color)
        pygame.display.flip()
        pygame.time.delay(5)

        xt = x
        x = x * C - y * S
        y = y * C + xt * S
```

- **Algoritmo de Bresenham**
  O método utilizando o algoritmo de Bresenham também se baseou na fórmula dada nos slides da disciplina:



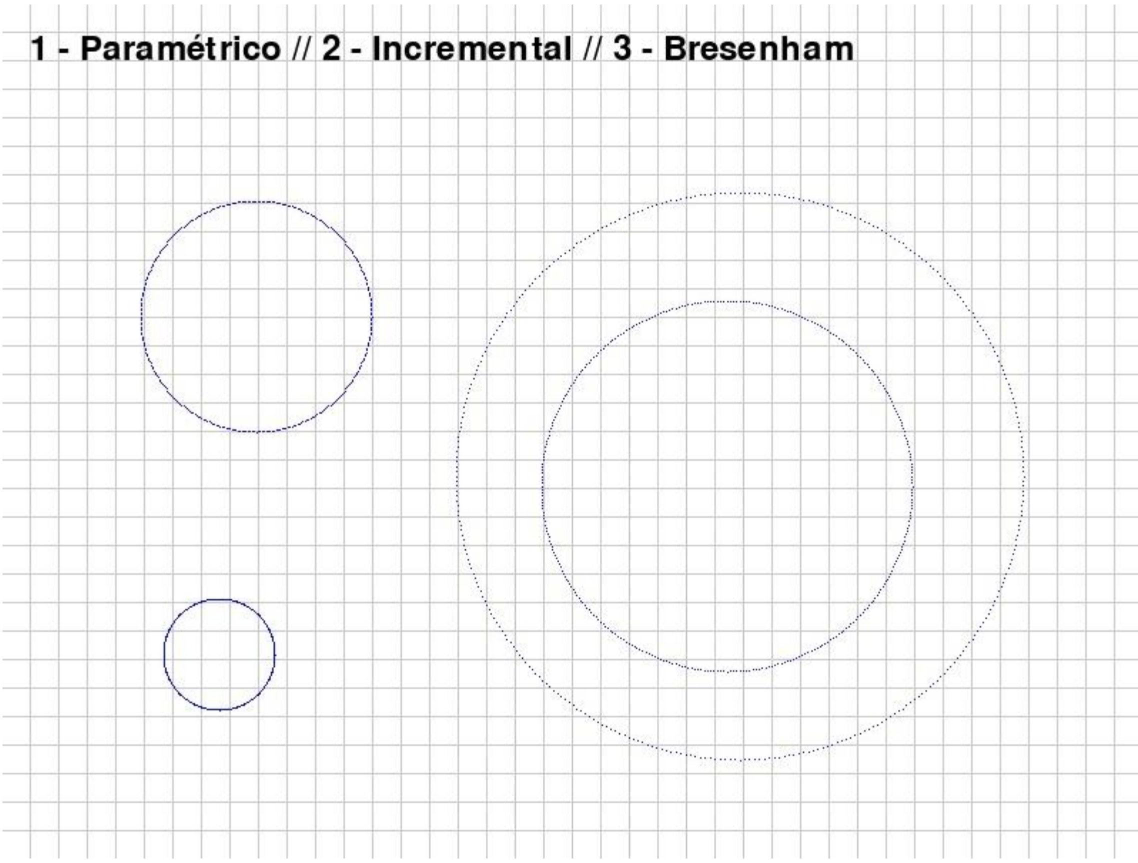


Função no código:

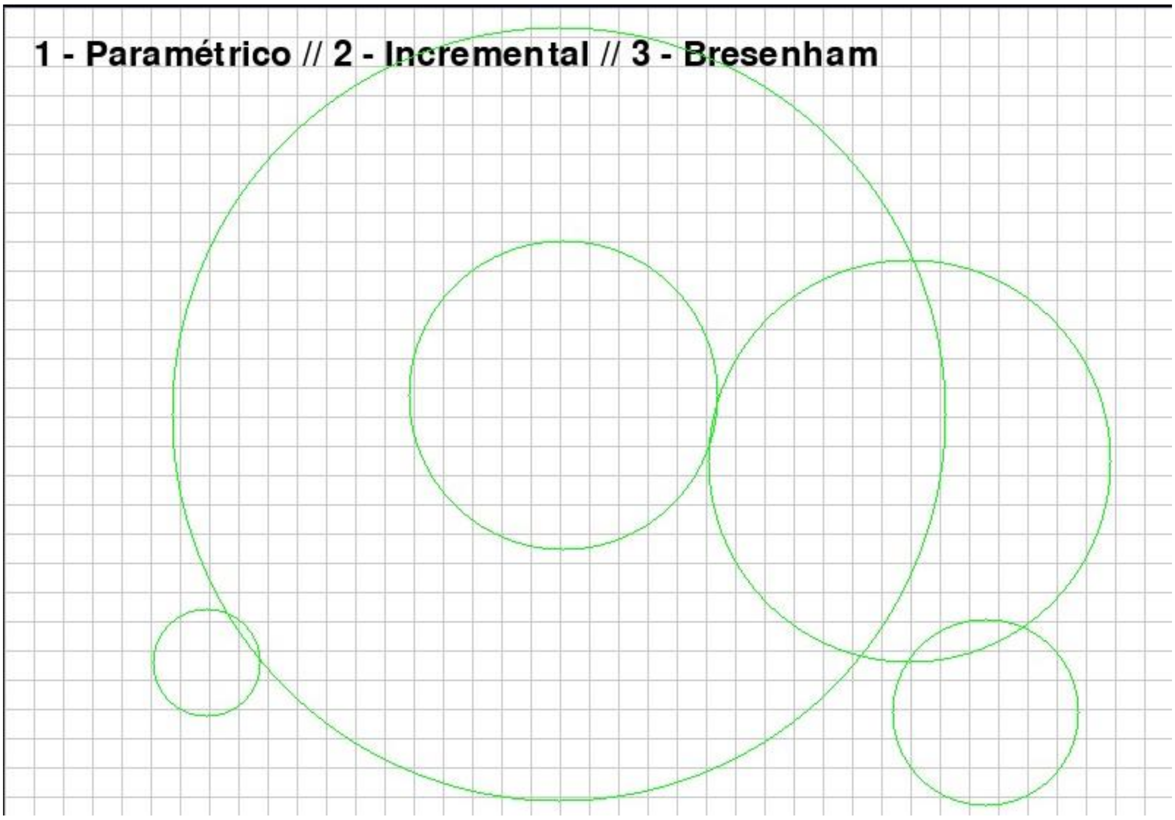
```python
# algoritmo de Bresenham
def draw_circ_bresenham(surface, color, center, radius):
    cx, cy = center
    x = 0
    y = radius
    d = 3 - 2 * radius
    while y >= x:
        points = [
            (cx + x, cy + y), (cx - x, cy + y), (cx + x, cy - y), (cx - x, cy - y),
            (cx + y, cy + x), (cx - y, cy + x), (cx + y, cy - x), (cx - y, cy - x)
        ]
        for point in points:
            surface.set_at(point, color)
        pygame.display.flip()
        pygame.time.delay(5)
        x += 1
        if d > 0:
            y -= 1
            d = d + 4 * (x - y) + 10
        else:
            d = d + 4 * x + 6
```

Resultados

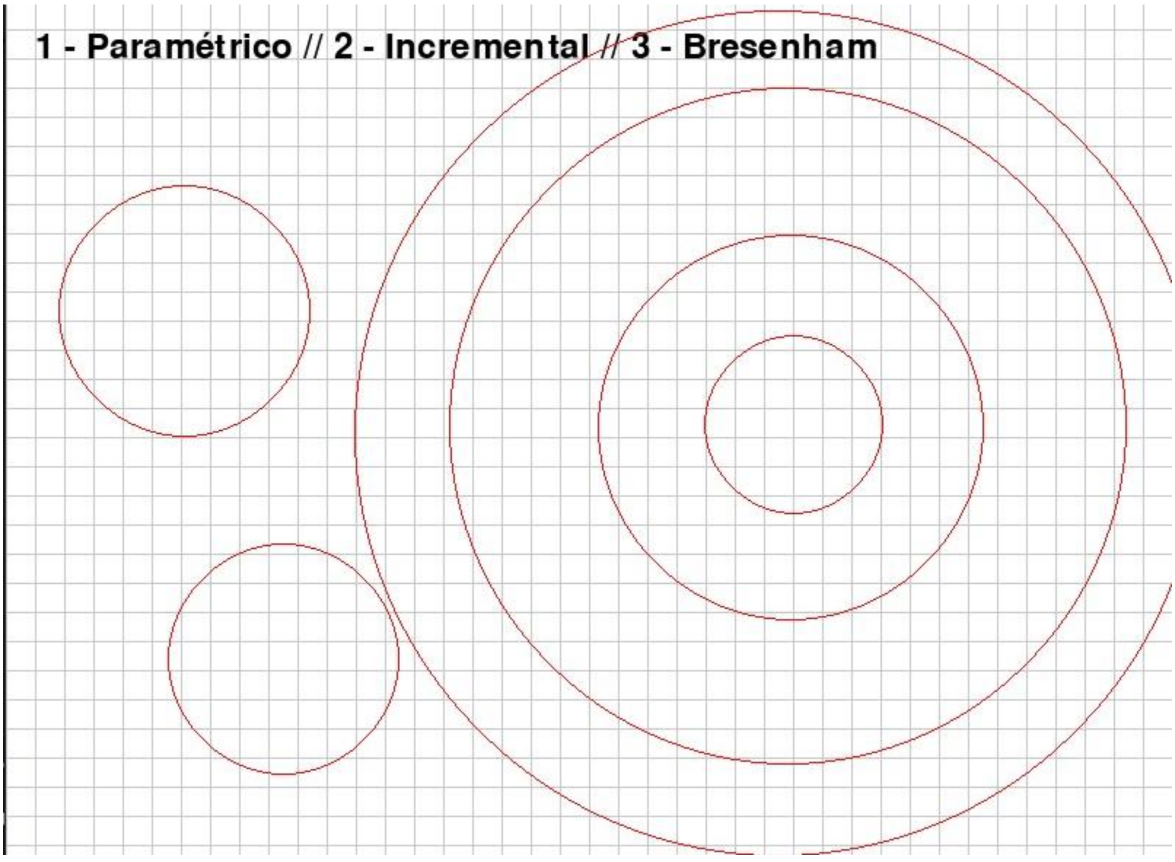
- **Equação Paramétrica (Aparência da circunferência)**



- **Método DDA (Aparência da circunferência)**



- **Método Algoritmo de Bresenham (Aparência da circunferência)**



Comparações

A partir dos resultados, pode-se fazer comparações entre as circunferências geradas por cada algoritmo:

- O **método da Equação Paramétrica** possui as seguintes vantagens:
  - Simples e intuitivo.
  - Gera uma boa aproximação da circunferência.
- Já as desvantagens:
  - Usa funções trigonométricas, que são computacionalmente caras.
  - Pode ter problemas de precisão devido ao arredondamento.
  - Não aproveita a simetria da circunferência.

- Já o **método Incremental com Simetria** leva vantagem em:
  - Não apresenta descontinuidades.
  - Reduz o número de cálculos ao aproveitar a simetria.
  - Como cos θ e sen θ são valores fixo, você consegue otimizar os algoritmos.
- Já os pontos negativos:
  - Erros acumulativos, em função do uso de xn e yn nas iterações seguintes.
  - Um pouco menos eficiente que o algoritmo de Bresenham.
  - Uso de números reais - necessidade do arredondamento, para cada pixel;

- Por fim, o **algoritmo de Bresenham** é o mais vantajoso:
  - Método mais eficiente, pois usa apenas operações inteiras (somas e subtrações).
  - Elimina a necessidade de funções trigonométricas e raízes quadradas.
  - Ideal para gráficos em tempo real devido à sua rapidez.
- Seu ponto negativo é ser:
  - Ligeiramente mais complexo de implementar do que os outros métodos.

Conclusão

Em resumo, a escolha do método de rasterização de circunferências depende do equilíbrio entre simplicidade, precisão e eficiência. O método **paramétrico** é fácil de entender, mas computacionalmente caro. O **incremental com simetria** melhora o desempenho ao não apresentar descontinuidades, enquanto o **algoritmo de Bresenham** é o mais eficiente, utilizando apenas operações inteiras. Para aplicações em tempo real e gráficos otimizados, Bresenham é a melhor opção, enquanto os outros métodos podem ser úteis para aprendizado e implementações mais simples.