Rasterização de Circunferências

DCC703 - Computação Gráfica (2024.2)
Prof. - Luciano Ferreira Silva
Aluno - Paulo Ferreira da Silva Júnior - 2019034400

Relatório: Rasterização de Circunferências

Introdução

A rasterização de circunferências é um processo fundamental na computação gráfica, sendo utilizada para desenhar curvas suaves em monitores digitais que são baseados em uma grade discreta de pixels. Neste relatório, exploramos três abordagens para a rasterização de circunferências:

1. **Equação Paramétrica**
2. **Incremental com Simetria**
3. **Bresenham**

Cada método foi implementado e testado, e os resultados foram comparados para analisar suas eficiências e limitações.

---

1. Equação Paramétrica

Descrição do Algoritmo

A equação paramétrica de uma circunferência de raio r e centro (xc,yc) é dada por:

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

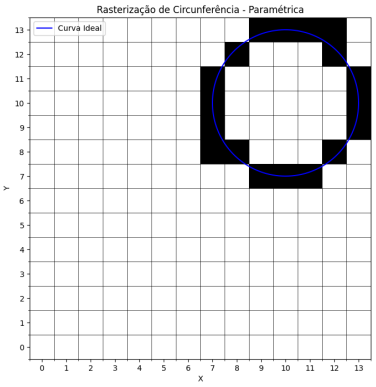
Onde t varia de 0 a 2π. Esse método calcula os pontos da circunferência diretamente e os arredonda para a grade discreta de pixels. Apesar de ser

intuitivo e de fácil implementação, pode introduzir erros de arredondamento e ser ineficiente devido ao uso de operações trigonométricas.

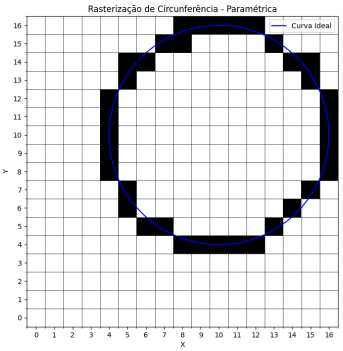
Código Implementado

```
import matplotlib.pyplot as plt
import numpy as np
import math

def draw_circle_parametric(xc, yc, r):
    points = []
    step = 1 / r  # Define o incremento para suavidade
    t = 0
    while t <= 2 * math.pi:
        x = round(xc + r * math.cos(t))
        y = round(yc + r * math.sin(t))
        points.append((x, y))
        t += step
    return points
```

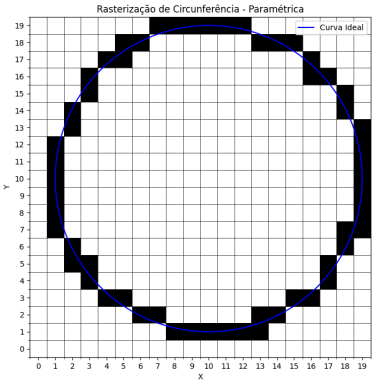
Resultado



Circunferência de raio 3



Circunferência de raio 6



Circunferência de raio 9

2. Incremental com Simetria

O método incremental com simetria evita o uso de funções trigonométricas e cálculos de raiz quadrada em cada iteração, tornando a rasterização mais

eficiente. O princípio desse método é baseado no **uso de incrementos pequenos e contínuos** nos valores de xxx e yyy, aplicando os conceitos de seno e cosseno para atualizar os pontos.

A cada iteração, os novos valores de xxx e yyy são calculados usando:

- $x_{novo} = x \cdot C - y \cdot S$
- $y_{novo} = y \cdot C + x \cdot S$

Onde:

- $\theta = 1/r$ é um pequeno ângulo incremental.
- $C = \cos(\theta)$ e $S = \sin(\theta)$ são os coeficientes de transformação.

Esses cálculos permitem que os pontos sejam gerados de forma incremental, reduzindo erros e tornando o algoritmo mais eficiente.

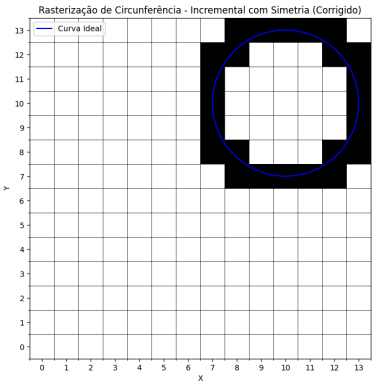
O método ainda faz uso da **simetria da circunferência**, desenhando apenas **um oitavo do círculo** e refletindo os pontos para os outros sete octantes.

Código Implementado

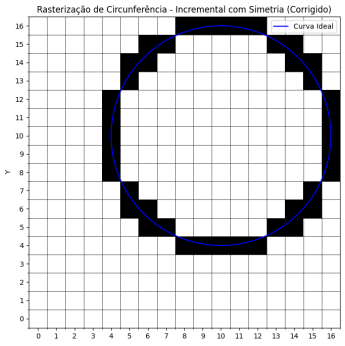
```
import matplotlib.pyplot as plt
import numpy as np
import math

def draw_circle_incremental_corrected(xc, yc, r):
    points = []

    # Inicializando variáveis conforme o método incremental correto
    x = r
    y = 0
    theta = 1 / r  # Incremento angular
    C = math.cos(theta)
    S = math.sin(theta)

    # Adiciona os primeiros pontos com simetria
    points.extend([
        (xc + x, yc + y), (xc - x, yc + y),
        (xc + x, yc - y), (xc - x, yc - y),
        (xc + y, yc + x), (xc - y, yc + x),
```

```
        (xc + y, yc - x), (xc - y, yc - x)
    ])

    while y < x:
        xt = x  # Armazena x temporariamente
        x = x * C - y * S  # Atualização incremental de x
        y = y * C + xt * S  # Atualização incremental de y

        # Arredonda para garantir a rasterização correta
        x_discreto = round(x)
        y_discreto = round(y)

        # Adiciona os pontos usando simetria
        points.extend([
            (xc + x_discreto, yc + y_discreto), (xc - x_discreto, yc + y_discreto),
            (xc + x_discreto, yc - y_discreto), (xc - x_discreto, yc - y_discreto),
            (xc + y_discreto, yc + x_discreto), (xc - y_discreto, yc + x_discreto),
            (xc + y_discreto, yc - x_discreto), (xc - y_discreto, yc - x_discreto)
        ])

    return points
```

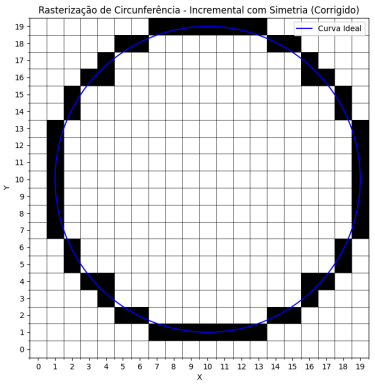
Resultado



Circunferência de raio 3



Circunferência de raio 6



Circunferência de raio 9

3. Bresenham

Descrição do Algoritmo

O algoritmo de Bresenham para circunferências é uma extensão do algoritmo de Bresenham para retas. Ele usa apenas operações inteiras para decidir quais pixels devem ser ativados, tornando-o o método mais eficiente entre os três apresentados. A decisão sobre o próximo pixel é baseada na função de decisão:

- $d=3-2\cdot r$

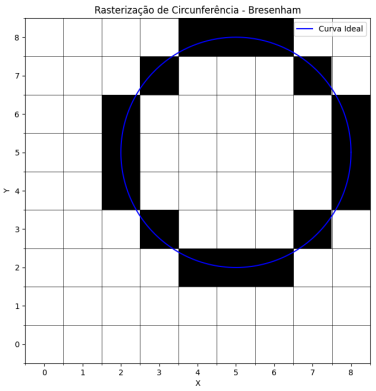
Os pixels são gerados apenas em um octante e depois refletidos para os demais, garantindo alta precisão e simetria.

Código Implementado

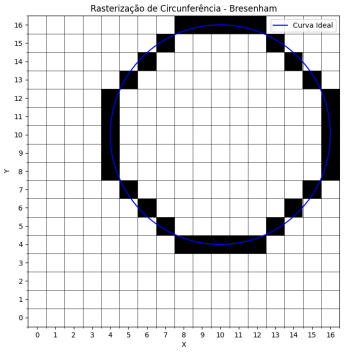
```
import matplotlib.pyplot as plt
import numpy as np

def draw_circle_bresenham(xc, yc, r):
    points = []
    x = 0
    y = r
    d = 3 - 2 * r
    while x <= y:
        points.extend([
```

```
            (xc + x, yc + y), (xc - x, yc + y),
            (xc + x, yc - y), (xc - x, yc - y),
            (xc + y, yc + x), (xc - y, yc + x),
            (xc + y, yc - x), (xc - y, yc - x)
        ])
        if d < 0:
            d = d + 4 * x + 6
        else:
            d = d + 4 * (x - y) + 10
            y -= 1
        x += 1
    return points
```

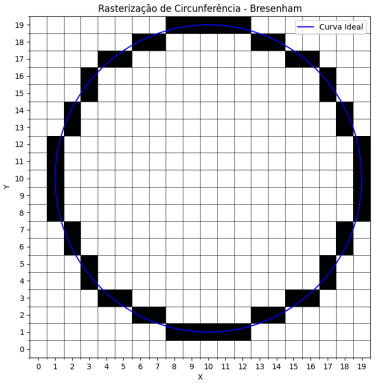
Resultado



Circunferência de raio 3



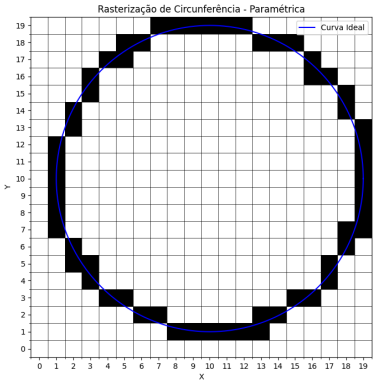
Circunferência de raio 6



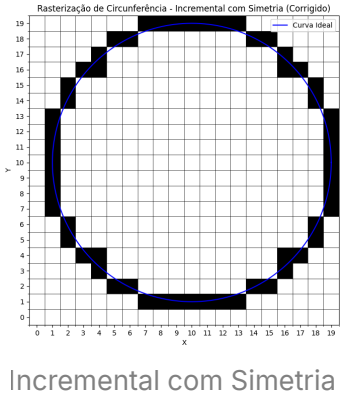
Circunferência de raio 9

Comparativo Geral

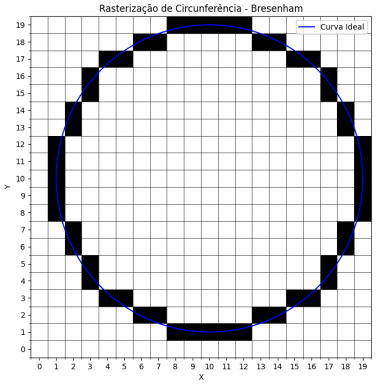
Método	Eficiência Computacional	Precisão	Observações
Equação Paramétrica	Baixa	Baixa	Utiliza operações trigonométricas e arredondamento
Incremental com Simetria	Média	Média	Introduz pequenos erros devido ao arredondamento
Bresenham	Alta	Alta	Usa apenas operações inteiras, garantindo alta eficiência e precisão



Equação Paramétrica de raio 9



Incremental com Simetria de raio 9



Bresenham de raio 9

Conclusão

Os experimentos mostraram que, embora todos os métodos sejam capazes de rasterizar uma circunferência, a escolha do algoritmo depende da aplicação desejada. O método de Bresenham é o mais eficiente, pois evita operações com números reais e minimiza erros de arredondamento. O método incremental com simetria é uma opção intermediária, apresentando algumas imprecisões. Já a equação paramétrica, apesar de intuitiva, é a menos eficiente e mais sujeita a erros.

Cada um dos métodos pode ser utilizado dependendo das restrições do sistema e da necessidade de precisão na geração das curvas. Para aplicações que exigem alto desempenho e qualidade, o algoritmo de Bresenham é a melhor escolha.