

Trabalho de Rasterização de Circunferências

Aluno: Rosialdo Vicente

Nº de Matricula: 2020018122

Objetivo do Programa

Desenvolver um programa que permita desenhar Circunferências utilizando os algoritmos:

- Paramétrico
- Incremental com Simetria
- Bresenham

Tecnologias Utilizadas

Linguagem de Programação

Para a implementação deste trabalho, optei pela linguagem `Python` devido à minha familiaridade e à sua flexibilidade para desenvolvimento de aplicações gráficas.

Bibliotecas Utilizadas

As seguintes bibliotecas foram utilizadas no projeto:

- `sys` : Utilizada para encerrar o programa de forma adequada.
- `pygame` : Escolhida para demonstrar graficamente os algoritmos de rasterização. Considerei inicialmente o uso da biblioteca matplotlib, mas optei pelo pygame por facilitar futuras implementações e pela sua abordagem mais intuitiva para manipulação de gráficos em tempo real.
- `math` : Utilizada para cálculos matemáticos necessários nos algoritmos, como funções trigonométricas, otimizando os cálculos relacionados à construção de circunferências.

Implementação dos Algoritmos

Os três algoritmos foram implementados em um único arquivo chamado `circunferencia.py`, facilitando a organização e apresentação do projeto.


Algoritmo Paramétrico

Este Algoritmo utiliza a equação paramétrica de uma circunferência para determinar os pontos. Suas entradas principais são:

- `xc` e `yc`: Coordenadas do centro da circunferência.
- `raio`: O raio da circunferência.

O algoritmo calcula os pontos da circunferência variando o ângulo de 0 a 360 graus. A cada iteração, utiliza funções trigonométricas para determinar as coordenadas X e Y.

Referência para o Algoritmo Paramétrico:



Circunferências: Eq. Paramétrica

UFRR – Departamento de Ciência da Computação
Computação Gráfica – Prof. Dr. Luciano F. Silva

■ **Algoritmo:**

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

$x = x_c + \text{raio} \quad y = y_c$
para t de 1 até 360 com passo “t”
<div>pixel (x , y , cor) $X = x_c + r \cdot \cos\left(\frac{\pi \cdot t}{180}\right)$$y = y_c + r \cdot \sin\left(\frac{\pi \cdot t}{180}\right)$</div>

Algoritmo Incremental com Simetria

O


algoritmo incremental gera a circunferência utilizando um **incremento angular fixo** para calcular novos pontos a partir dos anteriores. Ele evita chamadas repetitivas a **cos** e **sin**, substituindo-as por multiplicações e somas. A cada iteração, as coordenadas **(x, y)** são rotacionadas por um pequeno ângulo **theta = $1 / \text{raio}$** . Os pontos são espelhados nos outros octantes usando **simetria**.

Algoritmo Bresenham

O

algoritmo de Bresenham é um método eficiente para rasterização de circunferências, utilizando apenas **operações inteiras** (adição e subtração), evitando funções trigonométricas e cálculos de ponto flutuante. Ele se baseia em um **parâmetro de decisão** que ajuda a escolher entre dois possíveis pixels vizinhos, garantindo que o traçado fique o mais próximo possível da curva ideal.

Referência para o Algoritmo Bresenham :



UFRR

Circunferência Alg. de Bresenham

x = 0	e	y = r
Parâmetro = $p = 5/4 - r$ ou $1 - r$		
Enquanto $x \neq y$ faça: liga pixel (x, y, cor)		
p ≥ 0		
Sim		Não
y = y - 1 p = p + 2x - 2y + 5 x = x + 1		não altera y p = p + 2x + 3 x = x + 1
Desenhar os demais octantes		
Transladar a circunferência para o centro (xc, yc)		

UFRR - Departamento de Ciência da Computação
Computação Gráfica - Prof. Dr. Luciano F. Silva

Desenvolvimento

Durante a implementação, enfrentei desafios na visualização gráfica dos algoritmos. Inicialmente, tentei desenvolver os algoritmos de linha e circunferência simultaneamente, mas essa abordagem tornou o processo confuso. Para melhorar a organização, decidi priorizar a implementação das linhas, o que facilitou a adaptação posterior para o desenho das circunferências.

A função `simetria` apresentou maior complexidade, pois é responsável por distribuir corretamente os pontos espelhados nos diferentes octantes da circunferência, garantindo um desenho preciso. O **algoritmo incremental** exigiu mais tempo de implementação, pois demandou um estudo aprofundado de referências adicionais para sua compreensão completa.

Seria interessante ter implementado um modo de visualização progressiva do desenho dos círculos, permitindo observar as diferenças entre os algoritmos de forma mais detalhada. Além disso, optei por exibir todos os algoritmos simultaneamente em uma única tela para facilitar a análise comparativa.

No geral, considero que consegui implementar os algoritmos de forma satisfatória, e o programa final atende aos requisitos estabelecidos na atividade.

Testes

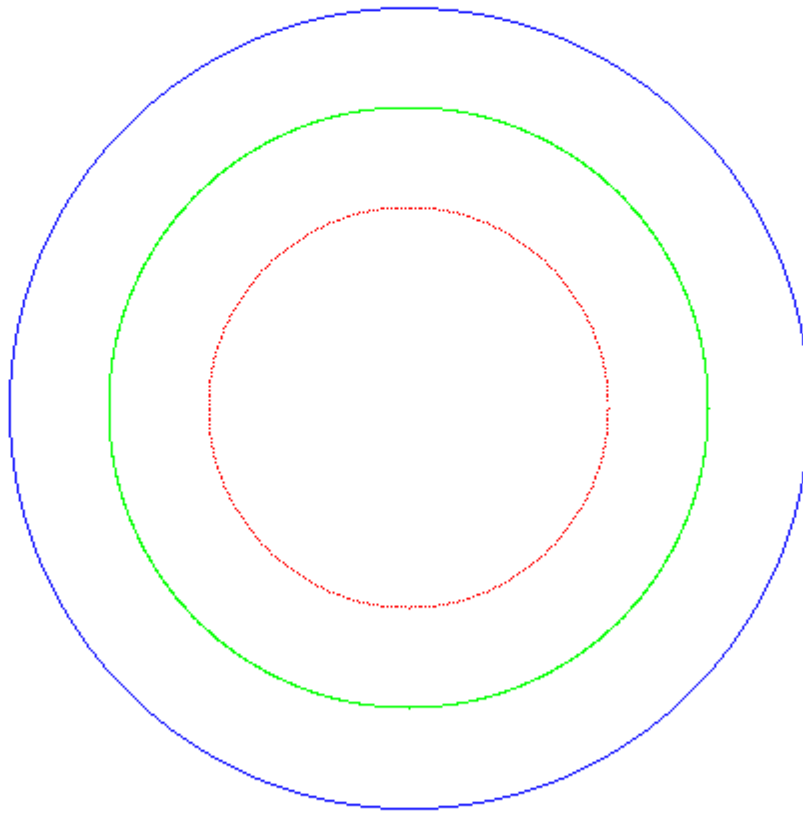
Todos os testes foram feitos com os seguintes valores:

Paramétrico: raio = 100°

Incremental com Simetria: raio = 150°

Bresenham: raio = 200°

Teste Circunferências



Comparando os Algoritmos

Os tr s algoritmos de rasteriza  o de circunfer ncias apresentam diferen as significativas em termos de efici ncia computacional e precis o. O **algoritmo param trico** utiliza fun  es trigonom tricas para calcular os pontos ao longo da circunfer ncia, garantindo uma representa  o precisa, por m com alto custo computacional devido ao uso de opera  es de ponto flutuante.

O **algoritmo incremental** reduz a complexidade ao substituir c lculos trigonom tricos por opera  es de soma e multiplica  o, tornando-se mais

eficiente. No entanto, ainda depende de aritmética de ponto flutuante, o que pode levar a erros acumulativos e imprecisões na renderização da curva.

O **algoritmo de Bresenham** é o mais eficiente entre os três, pois elimina a necessidade de operações com números reais, utilizando apenas soma e subtração inteiras. Isso resulta em um tempo de execução otimizado e uma rasterização precisa, tornando-o a melhor escolha para aplicações gráficas que exigem alto desempenho e baixo custo computacional.