UNIVERSIDADE FEDERAL DE RORAIMA
CENTRO DE CIÊNCIA E TECNOLOGIA - CCT
CURSO DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO

COMPUTAÇÃO GRÁFICA

RASTERIZAÇÃO DE CIRCUNFERÊNCIAS

HENDRICK SILVA FERREIRA-2020026830

Janeiro de 2025

Boa Vista/Roraima

Resumo

O relatório apresenta a implementação de três diferentes algoritmos para rasterização de circunferências em Python: a Equação Paramétrica, o Algoritmo Incremental com Simetria e o Algoritmo de Bresenham. Cada programa utiliza a biblioteca Pygame para exibir os resultados em uma janela gráfica. Ocom o objetivo de criar circunferências em um ambiente gráfico.s programas foram construídos com o objetivo de criar circunferências em um ambiente gráfico.

Conteúdo

Figuras

1 Introdução

Este relatório tem como objetivo apresentar a implementação de três diferentes algoritmos para a rasterização de circunferências em Python. Os algoritmos abordados são a Equação Paramétrica, o Algoritmo Incremental com Simetria e o Algoritmo de Bresenham.

1.1 Equação Paramétrica

O algoritmo da Equação Paramétrica é baseado em uma fórmula matemática que descreve uma circunferência. A fórmula paramétrica da circunferência é:

$$x = xc + r \cdot \cos(\varnothing)$$

$$y = yc + r \cdot \sin(\varnothing)$$

Onde x e y são as coordenadas dos pontos na circunferência, ((xc,yc) é o centro da circunferência, r é o raio e θ é o ângulo de 0 a 360 graus. O algoritmo calcula os pontos da circunferência iterando sobre os ângulos e aplicando essa fórmula. Embora seja fácil de entender, pode ser menos eficiente para circunferências grandes devido ao uso de funções trigonométricas.

1.2 Algoritmo Incremental com Simetria

O Algoritmo Incremental com Simetria é uma abordagem que evita cálculos trigonométricos, tornando-o eficiente para rasterização de circunferências. Ele calcula os pontos simétricos da circunferência em oito octantes, o que significa que apenas um oitavo da circunferência precisa ser calculado. Os outros sete oitavos podem ser obtidos refletindo os pontos calculados. O algoritmo utiliza decisões simples e operações de incremento para ajustar os pontos à medida que a circunferência é desenhada.

1.3 Algoritmo Bresenham

O Algoritmo de Bresenham é outro algoritmo eficiente para rasterização de circunferências que evita cálculos trigonométricos. Ele também calcula os pontos da circunferência de maneira incremental. O algoritmo ajusta os valores dos pontos em cada octante com base em um valor de decisão. Isso permite uma renderização precisa e simétrica da circunferência, com alta eficiência computacional.

2 Construção dos Programas

A construção dos programas que utilizam os algoritmos de rasterização de linhas - Analítico, DDA e Bresenham - envolve um conjunto comum de etapas. Que são:

2.1 Configuração do Ambiente

A primeira coisa feita foi configurar o ambiente de desenvolvimento, no qual é a instalação de bibliotecas necessárias, como o pygame, que é usado para a exibição gráfica. Isso também envolve a configuração da janela de exibição, definição do título e dimensionamento da tela.

2.2 Parâmetros da Circunferência

Para que os programas funcionem corretamente, defini os parâmetros da circunferência que será rasterizada. Que foi o mesmo para todos os programas.

2.3 Loop Principal

O loop principal foi usado para controlar a execução do programa. Ele normalmente inclui a manipulação de eventos, como o evento de fechamento da janela (pygame.QUIT), o preenchimento da tela com a cor de fundo desejada (que foi o preto) e a chamada da função de rasterização.

3 Descrição dos Programas

3.1 Equação Paramétrica

O programa da Equação Paramétrica utiliza a fórmula matemática de uma circunferência para calcular os pontos que compõem a circunferência. Ele itera por todos os ângulos entre 0 e 360 graus, calcula as coordenadas x e y correspondentes usando a fórmula paramétrica e pinta esses pontos na tela. O programa ficou assim:

equação_paramétrica.py

home > hendrick > Documentos > Rasterização de Circunferencias > equação_paramétrica.py > ...

```python
import pygame
import math

# Inicialize o Pygame
pygame.init()

# Defina as dimensões da janela
width, height = 800, 600
screen = pygame.display.set_mode((width, height))
pygame.display.set_caption("Rasterização de Circunferências")

# Função para rasterizar uma circunferência usando a equação paramétrica
def draw_circle_parametric(center_x, center_y, radius):
    for angle in range(0, 360):
        x = int(center_x + radius * math.cos(math.radians(angle)))
        y = int(center_y + radius * math.sin(math.radians(angle)))
        screen.set_at((x, y), (255, 255, 255))  # Define a cor do pixel

# Parâmetros da circunferência
center_x, center_y = width // 2, height // 2
radius = 100

# Loop principal
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    screen.fill((0, 0, 0))  # Limpa a tela
    draw_circle_parametric(center_x, center_y, radius)
    pygame.display.flip()

pygame.quit()
```

Figura 1 – Equação Paramétrica

3.2 Incremental com Simetria

O programa que utiliza o Algoritmo Incremental com Simetria segue uma abordagem diferente. Ele calcula os pontos simétricos em oito octantes da circunferência, usando apenas operações de incremento e decisões simples. Isso economiza tempo de cálculo e torna o algoritmo eficiente. O programa ficou assim:

incremental_com_simetria.py

home > hendrick > Documentos > Rasterização de Circunferencias > incremental_com_simetria.py > ...

```python
import pygame

# Inicialize o Pygame
pygame.init()

# Defina as dimensões da janela
width, height = 800, 600
screen = pygame.display.set_mode((width, height))
pygame.display.set_caption("Rasterização de Circunferências")

# Função para rasterizar uma circunferência usando o algoritmo incremental com simetria
def draw_circle_symmetric(center_x, center_y, radius):
    x, y = 0, radius
    d = 1 - radius
    while x <= y:
        screen.set_at((center_x + x, center_y + y), (255, 255, 255))  # Octante superior direito
        screen.set_at((center_x - x, center_y + y), (255, 255, 255))  # Octante superior esquerdo
        screen.set_at((center_x + x, center_y - y), (255, 255, 255))  # Octante inferior direito
        screen.set_at((center_x - x, center_y - y), (255, 255, 255))  # Octante inferior esquerdo
        screen.set_at((center_x + y, center_y + x), (255, 255, 255))  # Octante superior direito
        screen.set_at((center_x - y, center_y + x), (255, 255, 255))  # Octante superior esquerdo
        screen.set_at((center_x + y, center_y - x), (255, 255, 255))  # Octante inferior direito
        screen.set_at((center_x - y, center_y - x), (255, 255, 255))  # Octante inferior esquerdo
        if d < 0:
            d += 2 * x + 3
        else:
            d += 2 * (x - y) + 5
            y -= 1
        x += 1

# Parâmetros da circunferência
center_x, center_y = width // 2, height // 2
radius = 100

# Loop principal
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    screen.fill((0, 0, 0))  # Limpa a tela
    draw_circle_symmetric(center_x, center_y, radius)
    pygame.display.flip()

pygame.quit()
```

Figura 2 – Incremental com Simetria

3.3 Algoritmo de Bresenham

O programa que utiliza o Algoritmo de Bresenham também calcula os pontos da circunferência de maneira eficiente. Ele ajusta os valores dos pontos em cada octante com base em um valor de decisão, evitando a necessidade de cálculos trigonométricos, tornando-o um algoritmo eficiente para rasterização. O programa ficou assim:

Bresenham.py

home > hendrick > Documentos > Rasterização de Circunferencias > Bresenham.py > ...

```python
import pygame

# Inicialize o Pygame
pygame.init()

# Defina as dimensões da janela
width, height = 800, 600
screen = pygame.display.set_mode((width, height))
pygame.display.set_caption("Rasterização de Circunferências")

# Função para rasterizar uma circunferência usando o algoritmo de Bresenham
def draw_circle_bresenham(center_x, center_y, radius):
    x, y = 0, radius
    d = 3 - 2 * radius
    while x <= y:
        screen.set_at((center_x + x, center_y + y), (255, 255, 255))
        screen.set_at((center_x - x, center_y + y), (255, 255, 255))
        screen.set_at((center_x + x, center_y - y), (255, 255, 255))
        screen.set_at((center_x - x, center_y - y), (255, 255, 255))
        screen.set_at((center_x + y, center_y + x), (255, 255, 255))
        screen.set_at((center_x - y, center_y + x), (255, 255, 255))
        screen.set_at((center_x + y, center_y - x), (255, 255, 255))
        screen.set_at((center_x - y, center_y - x), (255, 255, 255))
        if d < 0:
            d += 4 * x + 6
        else:
            d += 4 * (x - y) + 10
            y -= 1
        x += 1

# Parâmetros da circunferência
center_x, center_y = width // 2, height // 2
radius = 100

# Loop principal
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    screen.fill((0, 0, 0))  # Limpa a tela
    draw_circle_bresenham(center_x, center_y, radius)
    pygame.display.flip()

pygame.quit()
```

Figura 3 - Bresenham

4 Resultados e Comparação

Os resultados obtidos ao executar cada um dos programas são circunferências exibidas na tela da janela gráfica. As circunferências são renderizadas com sucesso usando os três algoritmos. Pude perceber que as circunferências que foram exibidas têm uma leva diferença entre elas, como se pode ver pelas figuras. A circunferência exibida pelo programa usando a equação paramétrica, tem uma aparência menos suavizada em comparação as circunferências resultantes dos outros programas.

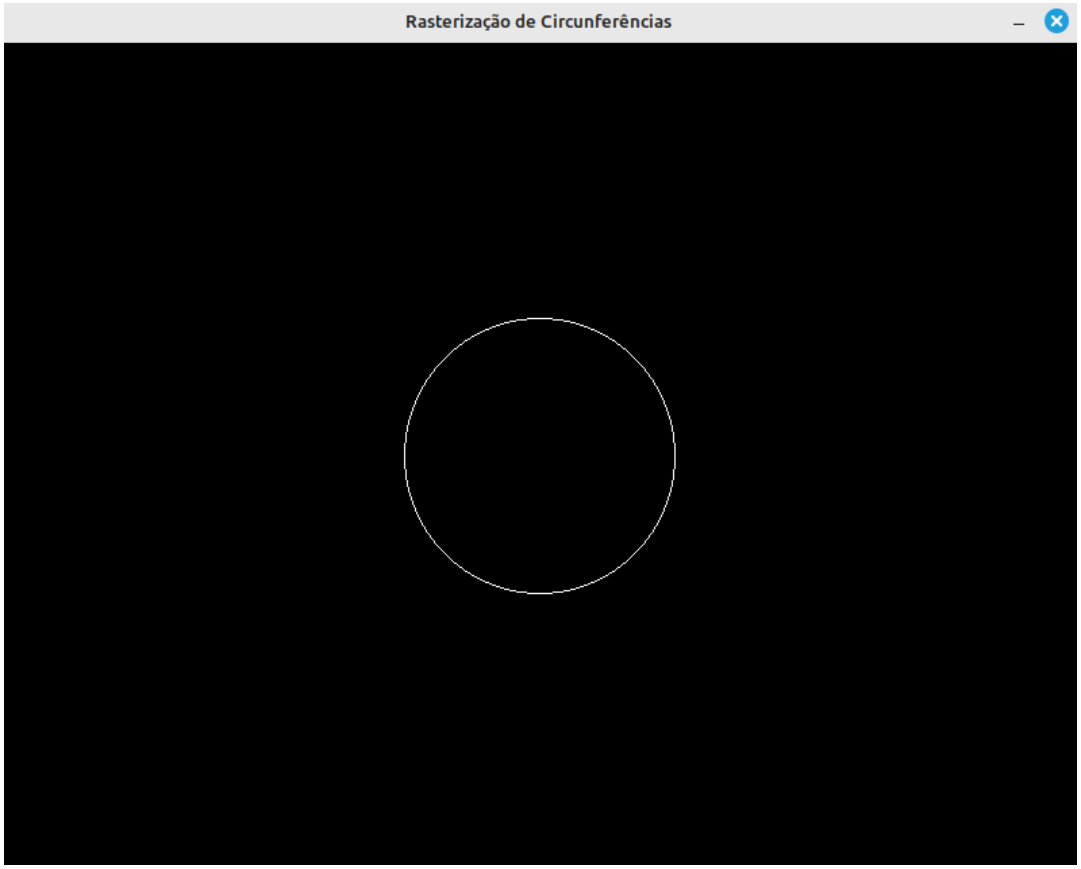

Figura 4 – Resultado da Incremental com Simetria

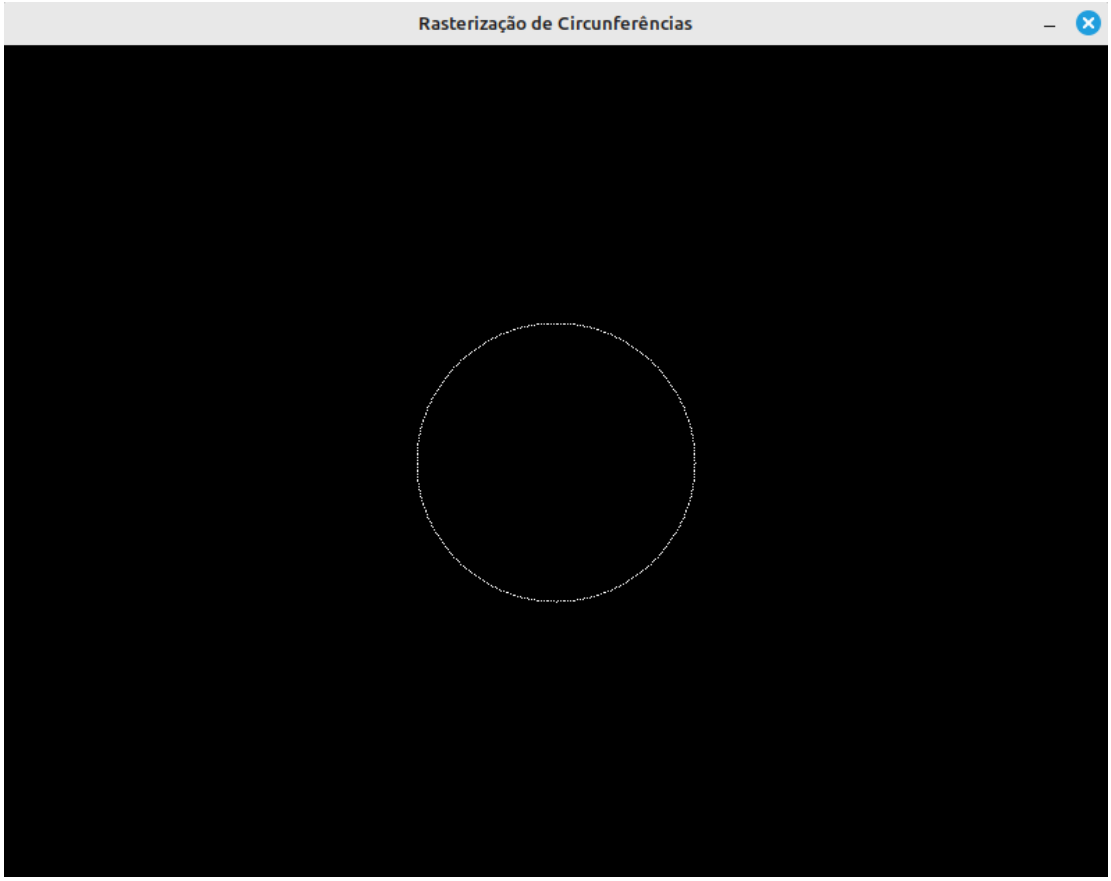

Figura 5 – Resultado da Equação Paramétrica

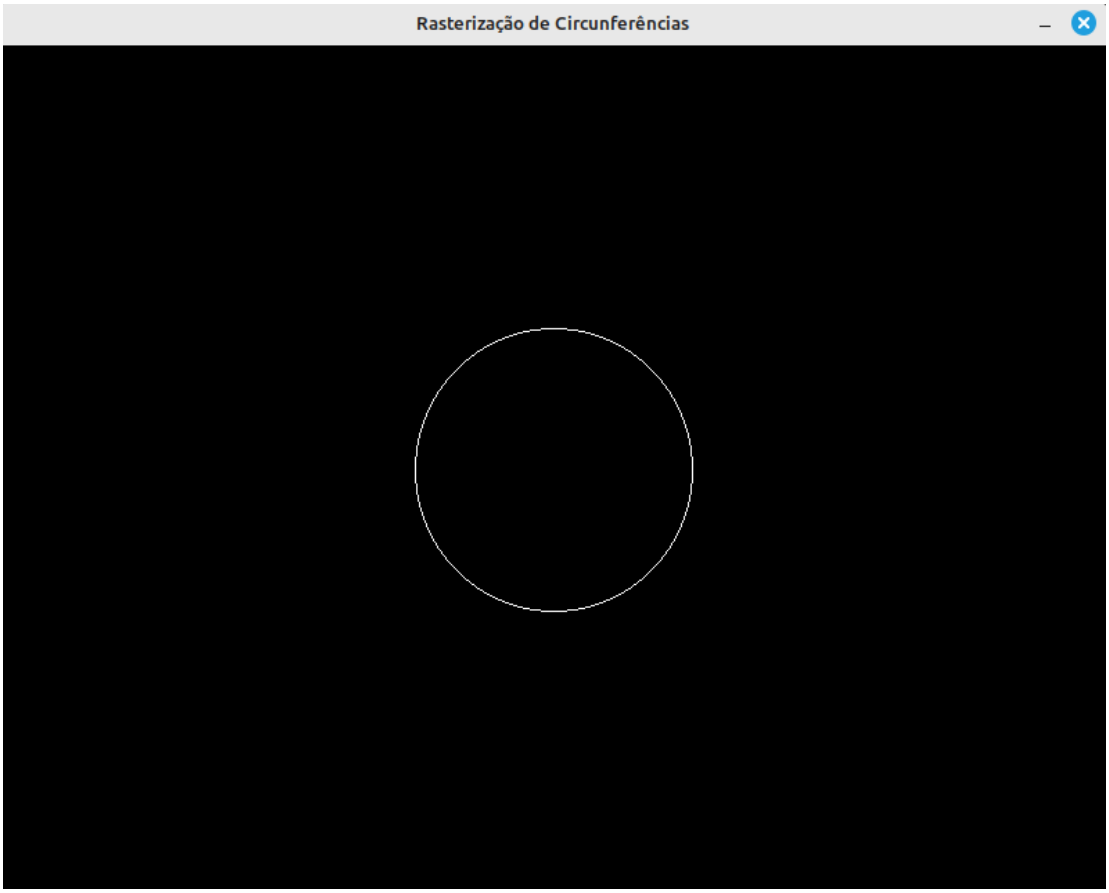

Figura 6 – Resultado da Bresenham

4.1 Comparativo

A Equação Paramétrica é direta e baseada em cálculos trigonométricos. É fácil de entender, mas pode ser menos eficiente para circunferências grandes devido ao uso de funções trigonométricas. O Algoritmo Incremental com Simetria é altamente eficiente e econômico em termos de recursos computacionais, pois evita cálculos trigonométricos. Ele é especialmente útil para aplicações em tempo real. O Algoritmo de Bresenham também é eficiente e evita cálculos trigonométricos, sendo uma boa escolha para sistemas com recursos limitados. Ele fornece resultados precisos e simétricos.

5 Conclusão

A escolha do algoritmo depende das necessidades específicas do projeto. Se a eficiência for crítica, o Algoritmo Incremental com Simetria e o Algoritmo de Bresenham são opções recomendadas. A escolha do algoritmo dependerá das necessidades específicas do projeto em termos de eficiência e simplicidade.

6 Referências

- https://gist.github.com/IgnacioCorto/eb8a931479ce23d1e307e3540146bd69
- https://stackoverflow.com/questions/76435638/finding-coordinates-of-all-
- pixels-when-drawing-a-circle-in-pygame
- https://gamedev.stackexchange.com/questions/55045/pygame-circular-
- motion-with-bresenhams-algorithm