UNIVERSIDADE FEDERAL DE RORAIMA
CENTRO DE CIÊNCIA E TECNOLOGIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
DCC703 - COMPUTAÇÃO GRÁFICA

RAFAEL NÓBREGA DE LIMA

ALGORITMOS DE RASTERIZAÇÃO DE LINHAS - RELATÓRIO

BOA VISTA - RR
2025

1. BASE DOS ALGORITMOS E ORGANIZAÇÃO

A linguagem escolhida para execução dos algoritmos de Rasterização de linhas e circunferências foi “Python”, pois é uma linguagem de certa forma simples comparada com outras de alto nível, além disso ela dispõe de muitas bibliotecas para a renderização de imagens. A biblioteca usada nesses algoritmos é a “Pygame”, uma biblioteca com foco na criação de jogos, mas com a capacidade para a renderização e visualização das formas criadas pelo o algoritmo.

A tela de cada algoritmo de rasterização foram pré definidas sendo 400x400 para a rasterização de linhas e 800x800 para a de circunferências. Para melhor visualização foi criado um grid com uma resolução menor para exibir os pixels no grid e assim mostrar melhor as diferenças de cada método de rasterização.

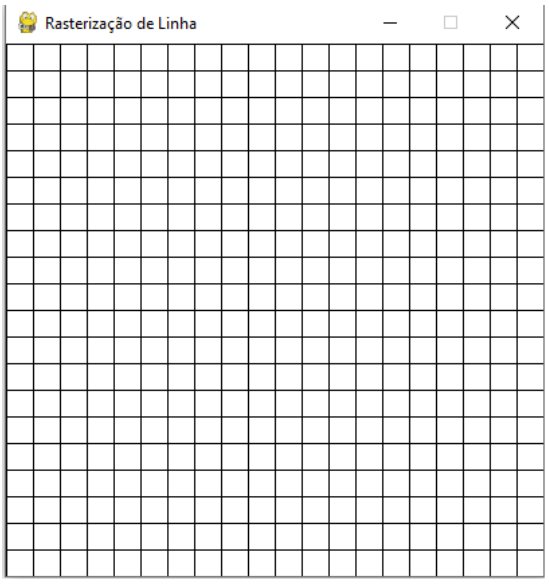
```
tamanho_celula = 20
linhas = altura // tamanho_celula
colunas = largura // tamanho_celula
```

1.1. DESENHO DO GRID

- A função desenhar_grid() configura e cria uma grade visual para representar um sistema de coordenadas discretas.

```
# Desenha o grid na tela
def desenhar_grid():
    for x in range(0, largura, tamanho_celula):
        pygame.draw.line(tela, PRETO, (x, 0), (x, altura))
    for y in range(0, altura, tamanho_celula):
        pygame.draw.line(tela, PRETO, (0, y), (largura, y))
```

- Linhas verticais e horizontais são desenhadas para dividir a tela em células. Assim ajudando na melhor visualização das formas geradas.



1.2. PINTURA DE PIXELS

- A função desenhar_pontos() precisa de dois argumentos sendo eles pontos (coordenadas de início e fim da forma) e cor, então a função percorre a lista de pontos calculados e pinta cada um usando a função pygame.draw.rect().

```
def desenhar_pontos(pontos, cor):
    for x, y in pontos:
        pygame.draw.rect(tela, cor, (x * tamanho_celula, y * tamanho_celula, tamanho_celula, tamanho_celula))
```

- Cada ponto é desenhado como um pequeno quadrado preenchido, representando um pixel da tela.

2. RASTERIZAÇÃO DE LINHAS

A rasterização é uma técnica utilizada para converter representações matemáticas de linhas em coordenadas de pixels discretos em uma tela. Assim representando uma linha de um ponto inicial a um ponto final.

2.1. MÉTODO ANALÍTICO

A rasterização usando o método analítico se baseia na equação da reta na forma explícita. A equação geral da reta é: ($y = m \cdot x + b$).

- Onde o $m = \Delta x/\Delta y$ é a inclinação da linha.
- E o $b = y0 - m \cdot x0$ é o ponto de interseção da linha no eixo y.

- **CÓDIGO IMPLEMENTADO**

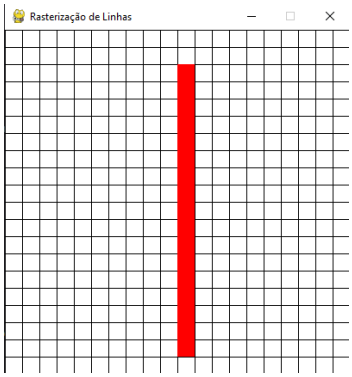
A função de rasterização de linhas analíticas recebe como parâmetros as coordenadas iniciais e finais(x0,y0 e x1,y1).

```
def rasterizar_linha_analitico(x0, y0, x1, y1):
    pontos = []
    dx = x1 - x0
    dy = y1 - y0
    if dx == 0:
        for y in range(min(y0, y1), max(y0, y1) + 1):
            pontos.append((x0, y))
    else:
        m = dy / dx
        b = y0 - m * x0
        for x in range(min(x0, x1), max(x0, x1) + 1):
            y = round(m * x + b)
            pontos.append((x, y))
    return pontos
```

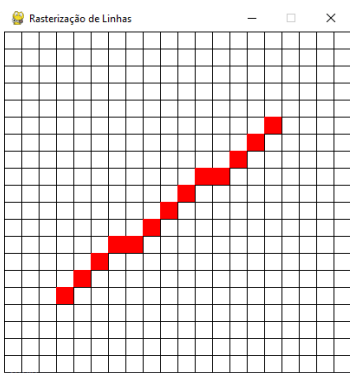
- **PASSOS**

  - Calcula-se e a partir dos pontos inicial e final da linha.
  - Itera-se em ou , dependendo da inclinação, e calcula-se a outra coordenada usando a equação da reta.
  - As coordenadas calculadas são arredondadas para o pixel mais próximo.
  - Ocorre a pintura do Pixel.

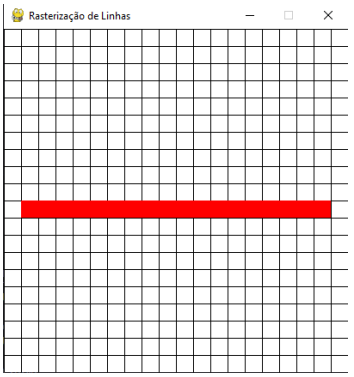
- **RESULTADOS**



Linha vertical



Inclinação 45º

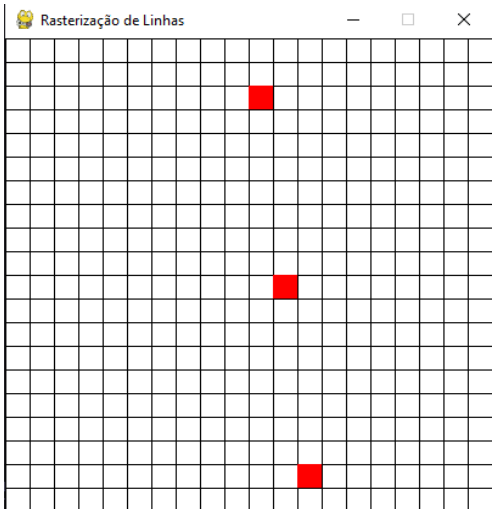


Linha Horizontal

- **LIMITAÇÕES**

Pelo método analítico, são realizados diversos cálculos, incluindo operações custosas com números em ponto flutuante, tornando necessário realizar arredondamentos em determinados momentos.

Por causa desses arredondamentos para se encaixarem na grade de pixels (que utiliza coordenadas inteiras), podem ocorrer desvios nas posições dos pixels fazendo assim que a linha fique “vazada”. Isso ocorre em linhas muito inclinadas.



Linha com as coordenadas
(x0, y0 = 10, 2 e x1, y1 = 12, 18)

2.2. DDA

O método DDA é baseado no uso de incrementos fixos para rastrear a linha usando o cálculo de Δx e Δy. Ele evita a necessidade de calcular a equação completa da reta para cada ponto.

- **CÓDIGO IMPLEMENTADO**

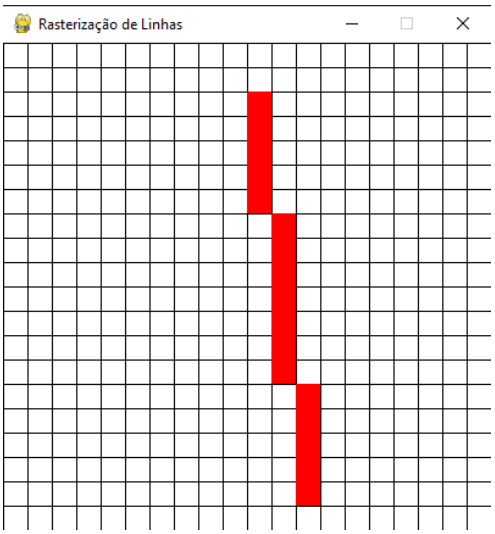
A função de rasterização pelo DDA também recebe como parâmetros as coordenadas iniciais e finais(x0,y0 e x1,y1).

```python
def rasterizar_linha_dda(x0, y0, x1, y1):
    pontos = []
    dx = x1 - x0
    dy = y1 - y0
    passos = max(abs(dx), abs(dy))
    inc_x = dx / passos
    inc_y = dy / passos
    x, y = x0, y0
    for _ in range(passos + 1):
        pontos.append((round(x), round(y)))
        x += inc_x
        y += inc_y
    return pontos
```

- **PASSOS**
  - Calcular as diferenças das coordenadas Δx e Δy;
  - Determinar o número de passos;
  - A cada passo, incrementar e pelos valores calculados e arredondar para os pixels mais próximos.
  - Iterar sobre os passos
  - Ocorre a pintura dos Pixels.

- **RESULTADOS**

  Com o DDA, os pontos da linha são calculados de forma incremental, o que distribui os erros de arredondamento de maneira uniforme. Isso evita que os erros se acumulem, como acontece no método analítico. Assim, as linhas geradas pelo DDA ficam mais contínuas e precisas.



Linha com as coordenadas
(x0, y0 = 10, 2 e x1, y1 = 12, 18)

- **LIMITAÇÕES**

Como no método analítico,o método DDA ainda faz operações custosas usando ponto flutuante e acaba tendo que arredondar os resultados assim fazendo com que aconteça a criação de linhas imprecisas.

2.3. BRESENHAM

O método de Bresenham é amplamente usado porque é rápido e eficiente comparados aos outros métodos já descritos nesse relatório, realizando apenas operações inteiras (adição, subtração e comparação). Esse algoritmo determina os pixels que mais se aproximam de uma linha reta ideal, utilizando um critério de decisão incremental para evitar cálculos complexos com números decimais ou ponto flutuante.

- **CÓDIGO IMPLEMENTADO**

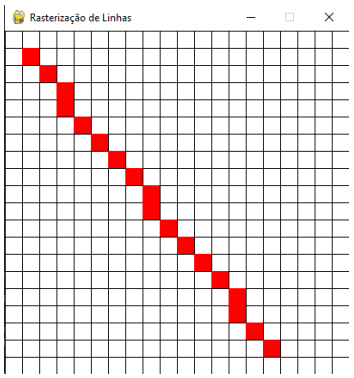
A função de rasterização por Bresenham também recebe como parâmetros as coordenadas iniciais e finais(x0,y0 e x1,y1).

```
def rasterizar_linha_bresenham(x0, y0, x1, y1):
    pontos = []
    dx = abs(x1 - x0)
    dy = abs(y1 - y0)
    sx = 1 if x0 < x1 else -1
    sy = 1 if y0 < y1 else -1
    p = dx - dy
    while True:
        pontos.append((x0, y0))
        if x0 == x1 and y0 == y1:
            break
        e2 = 2 * p
        if e2 > -dy:
            p -= dy
            x0 += sx
        if e2 < dx:
            p += dx
            y0 += sy
    return pontos
```

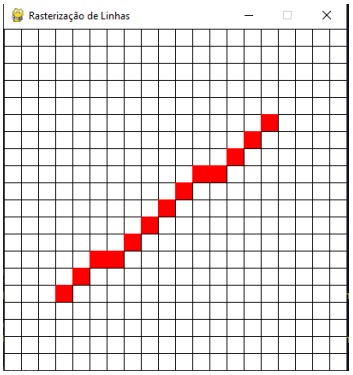
- **PASSOS**
  - Calcula as diferenças $\Delta x = x1 - x0$ *e* $\Delta y = y1 - y0$;
  - Determina o eixo principal de acordo
    - Se $|\Delta x| \geq |\Delta y|$, o eixo x será o principal (a linha cresce mais horizontalmente).
    - Caso contrário, o eixo y será o principal (a linha cresce mais verticalmente).
  - Inicializa o ponto inicial e a variável de decisão $p0 = 2\Delta y - \Delta x$.
  - A cada passo, use p para decidir se incrementa x ou y (dependendo da inclinação).
  - Ocorre a pintura do pixel mais próximo da linha ideal.

- **RESULTADOS**

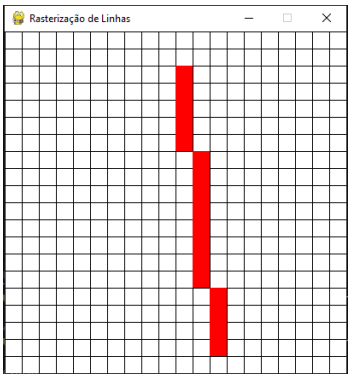
  O método Bresenham de rasterização de linhas se destaca entre os três métodos apresentados neste artigo por sua eficiência computacional e precisão. Pois utiliza apenas números inteiros em seus cálculos, eliminando a necessidade de uso de operações com números com ponto flutuante, tornando menos custoso computacionalmente. Outra vantagem dele é o uso de incrementos unitários fazendo com que ele seja mais preciso na definição dos pixels que compõem a linha, evitando os erros acumulados que podem surgir no DDA e no método Analítico.



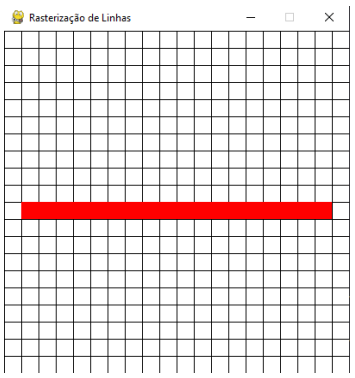
linha diagonal



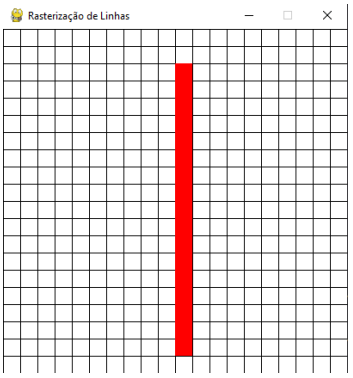
Inclinação 45º



linha quase vertical



linha horizontal



linha vertical

3. RASTERIZAÇÃO DE CIRCUNFERÊNCIAS

A rasterização de circunferência é um algoritmo que gera uma circunferência em pixels. Para isso, considera a simetria da circunferência para calcular pontos em um octante e depois refletir esses pontos nos outros octantes.

3.1. ALGORITMO PARAMETRICO

O algoritmo paramétrico para rasterização de circunferências usa a equação da circunferência no sistema paramétrico, baseada em funções trigonométricas. A equação paramétrica de uma circunferência com centro (xc,yc) e raio r é:

$$x = xc + r \cdot cos(\theta)$$
$$y = yc + r \cdot sin(\theta)$$

Onde θ é um ângulo variando de **0° a 360°**.

- **CÓDIGO IMPLEMENTADO**

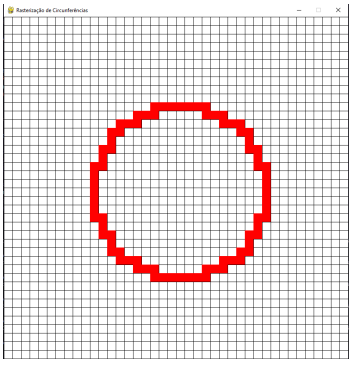
A função de rasterização paramétrico recebe como parâmetros as coordenadas do centro da circunferência e o raio(xc,xy e o r).

```python
def rasterizar_circulo_parametrico(xc, yc, r):
    pontos = []
    for angulo in range(0, 360):
        rad = math.radians(angulo)
        x = round(xc + r * math.cos(rad))
        y = round(yc + r * math.sin(rad))
        pontos.append((x, y))
    return pontos
```

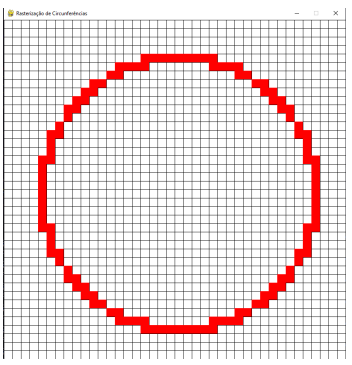
- **PASSOS**
  - Definir o centro e o raio da circunferência.
  - Iterar de 0° a 360° (em pequenos incrementos, geralmente de 1°).
  - Para cada valor do ângulo θ:
    - Converter para radianos (θ→radianos).
    - Calcular as coordenadas x,y usando as equações trigonométricas.
    - Arredondar os valores x,y para inteiros, pois estamos rasterizando em uma grade de pixels.
  - Adicionar os pontos calculados à lista de pontos que formam a circunferência.
  - Desenhar os pontos na tela.

- **RESULTADOS**

  Este algoritmo consegue gerar um conjunto de pontos de certa forma precisos para a circunferência, este algoritmo consegue funcionar bem para vários raios e centros sem a necessidade de ajustes complexos.



10 de raio



15 de raio

- **LIMITAÇÕES**

  Por precisar calcular seno e cosseno para cada ponto, isso o torna computacionalmente caro e assim tendo um baixo desempenho. Um outro problema é que pelo o ângulo ser incrementado de 1° por vez, os pixels podem ficar desigualmente espaçados em diferentes partes da circunferência

3.2. ALGORITMO INCREMENTAL COM SIMETRIA

algoritmo incremental com simetria é uma técnica que evita o uso de funções trigonométricas (seno/cosseno) ao rasterizar uma circunferência. Ele se baseia na equação da circunferência:

$$x^2 + y^2 = r^2$$

Ao invés de calcular cada ponto diretamente com seno e cosseno, o método usa um **passo incremental** para calcular os valores de y com base em x, garantindo eficiência.

- **IMPLEMENTAÇÃO NO CÓDIGO**

A função de rasterização incremental com simetria também recebe como parâmetros as coordenadas do centro da circunferência e o raio(xc,xy e o r).

```python
def rasterizar_circulo_incremental(xc, yc, r):
    pontos = []
    x = r
    y = 0
    theta = 1 / r  # Incremento angular
    C = math.cos(theta)
    S = math.sin(theta)
    while y < x:
        # Adiciona os 8 pontos simétricos
        pontos.extend([
            (round(xc + x), round(yc + y)), (round(xc - x), round(yc + y)),
            (round(xc + x), round(yc - y)), (round(xc - x), round(yc - y)),
            (round(xc + y), round(yc + x)), (round(xc - y), round(yc + x)),
            (round(xc + y), round(yc - x)), (round(xc - y), round(yc - x))
        ])
        # Atualiza os valores de x e y usando a fórmula incremental
        xt = x
        x = x * C - y * S
        y = y * C + xt * S
    return pontos
```

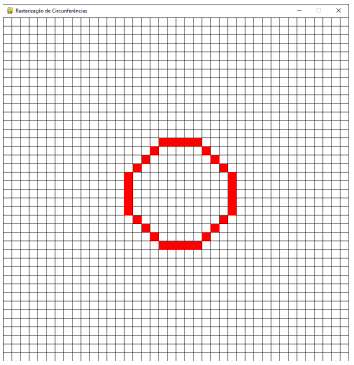
- **PASSOS**
  - Definir o centro (xc,yc) e o raio r da circunferência.
  - Começar com x = 0 e y = r (um ponto no topo da circunferência).
  - Para cada incremento de x (a partir de zero até y):
    - Calcular y reescrevendo a equação da circunferência:

$$y = \sqrt{r^2 - x^2}$$

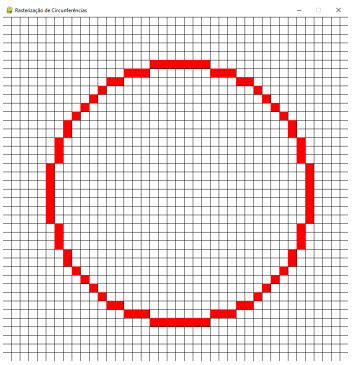
    - Arredondar o valor de y para um número inteiro (pois estamos rasterizando em pixels).
    - Usar a **simetria octante** para espelhar os pontos nos 8 quadrantes da circunferência.
    - Repetir até x > y (pois a simetria cobre toda a circunferência).

- **RESULTADOS**

Por não usar seno e cosseno este algoritmo consegue ser mais rápido que o método paramétrico e com isso é reduzido a quantidade de cálculos usando apenas a simetria ao seu favor.



10 de raio



15 de raio

- **LIMITAÇÕES**

  O cálculo da raiz quadrada envolve operações de ponto flutuante, podendo ser lento em processadores antigos, um outro problema é o arredondamento de y pode gerar pequenas falhas visuais na circunferência, pois os pontos podem não estar perfeitamente espaçados.

3.3. ALGORITMO DE BRESENHAM

O algoritmo de Bresenham é um método eficiente para rasterizar circunferências usando apenas operações inteiras (adições, subtrações e deslocamentos), sem precisar de raízes quadradas ou funções trigonométricas. Ele é baseado no algoritmo de Bresenham para linhas, mas adaptado para circunferências.

- **IMPLEMENTAÇÃO NO CÓDIGO**

A função de rasterização usando o bresenham também recebe como parâmetros as coordenadas do centro da circunferência e o raio(xc,xy e o r).

```
def rasterizar_circulo_bresenham(xc, yc, r):
    pontos = []
    x = 0
    y = r
    d = 3 - 2 * r
    while x <= y:
        pontos.extend([  # Adiciona os 8 simétricos
            (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 pontos
```

- **PASSOS**
  - Definir o centro (xc,yc) e o raio r.
  - Começar com x = 0 e y = r (um ponto no topo da circunferência).
  - Inicializar a variável de decisão:

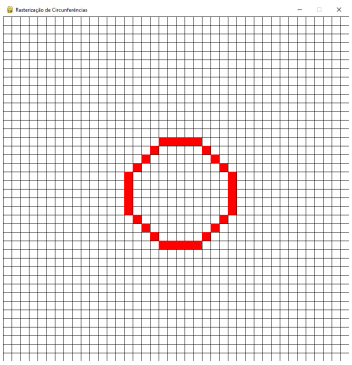
$$d = 3 - 2r$$

  - Para cada incremento de x, atualizar y **baseando-se no valor de d**:
    - Se $d < 0$, mantém y e atualiza d com:
      - $d = d + 4x + 6$

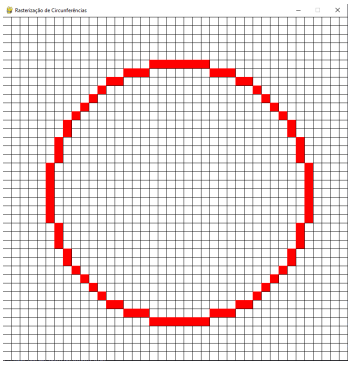
- Se $d \geq 0$, mantém y e atualiza d com:
  - $d = d + 4(x - y) + 10$
- Aplicar **simetria octante** para desenhar os 8 pontos correspondentes na circunferência.
- Parar quando $x > y$, pois a simetria cobre toda a circunferência.

- **RESULTADOS**

  Este algoritmo consegue ser mais rápido que os dois anteriores,pois não usa funções trigonométricas ou raízes quadradas. Por só usar operações inteiras, faz com que ele seja menos custoso computacionalmente e também garante que a circunferência seja bem formada e mantém a simetria exata.



10 de raio



15 de raio