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 Linhas

Relatório técnico de acordo com a proposta apresentada pelo docente Prof. Luciano Ferreira relacionado ao projeto de Rasterização de Linhas da disciplina de Computação Gráfica, que deve desenvolver um programa que permita desenhar retas por meio dos algoritmos: Analítico, DDA 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 Linhas

Relatório apresentado para o projeto de Rasterização de Linhas 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 linhas 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, DDA (Digital Differential Analyzer) e o método Analítico baseado na equação da reta.

Algoritmos

- **Método Analítico**
  O método analítico utiliza a equação da reta no formato da equação fundamental da reta:

  $$y = mx + b$$

  Onde:
  - **m** é o coeficiente angular da reta, definido como $m = \Delta x / \Delta y$.
  - **b** é o coeficiente linear (interseção com o eixo y).

  Como funciona?
  1. Dado um ponto inicial (x1, y1) e um ponto final (x2, y2) calcula-se o coeficiente angular m.
  2. Para cada valor de x, calcula-se y usando $y = mx + b$.
  3. Os pontos são arredondados para inteiros e desenhados na tela.

- **Método DDA**
  O método DDA (Analisador Diferencial Digital) é um algoritmo de rasterização de retas que evita o uso direto da equação da reta, reduzindo erros de arredondamento e melhorando a eficiência.

  Como funciona?
  1. Calcula-se o número de passos necessários com base na maior variação entre Δx e Δy:

     $$steps = max(|\Delta x|, |\Delta y|)$$

  2. Calculam-se os incrementos Xinc e Yinc para cada passo:

     $$Xinc = \Delta x / steps, \quad Yinc = \Delta y / steps$$

     Obs: *steps* representa o número de iterações necessárias para desenhar a reta.

     Onde:
     - $\Delta x = x2 - x1$ (diferença entre as coordenadas x)
     - $\Delta y = y2 - y1$ (diferença entre as coordenadas y)

Se for mais horizontal ($|\Delta x| > |\Delta y|$), os passos serão baseados em Δx.
Se for mais vertical ($|\Delta y| > |\Delta x|$), os passos serão baseados em Δy.

3. Começando do ponto inicial (x1, y1), incrementa-se os valores de x e y passo a passo, arredondando para obter coordenadas inteiras.

- **Método Algoritmo de Bresenham**

O algoritmo de Bresenham é um método eficiente para desenhar retas em gráficos rasterizados, pois utiliza apenas operações com números inteiros (adições e subtrações), evitando o uso de ponto flutuante como no DDA.

Como funciona?

1. Começa no ponto inicial (x1, y1).
2. Calcula Δx e Δy:

   $$\mathbf{\Delta x = x2 - x1, \Delta y = y2 - y1}$$

3. Define um parâmetro de decisão, como:

   $$\mathbf{p = 2.\Delta y - \Delta x}$$

O algoritmo decide quais pixels inteiros devem ser ativados para aproximar uma reta entre dois pontos (x1, y1) e (x2, y2). Ele percorre os pontos da reta pixel a pixel, escolhendo a posição mais próxima da trajetória ideal.

Calcula-se a variação horizontal e vertical da reta:

$\Delta x = x2 - x1$
$\Delta y = y2 - y1$

Define-se o parâmetro de decisão inicial: $p0 = 2\Delta y - \Delta x$
Define-se os incrementos em x e y:

1. Se $x1 < x2$, x cresce (+1), senão decresce (-1).
2. Se $y1 < y2$, y cresce (+1), senão decresce (-1).

Para cada pixel na direção de x, decide-se se o próximo ponto sobe para a linha superior ou mantém a linha atual:

3. Se $p < 0$: O próximo pixel está mais perto da linha atual → Incrementa x, mas mantém y.
4. Se $p \geq 0p$: O próximo pixel está mais perto da linha de cima → Incrementa x e y.

O parâmetro de decisão é atualizado:

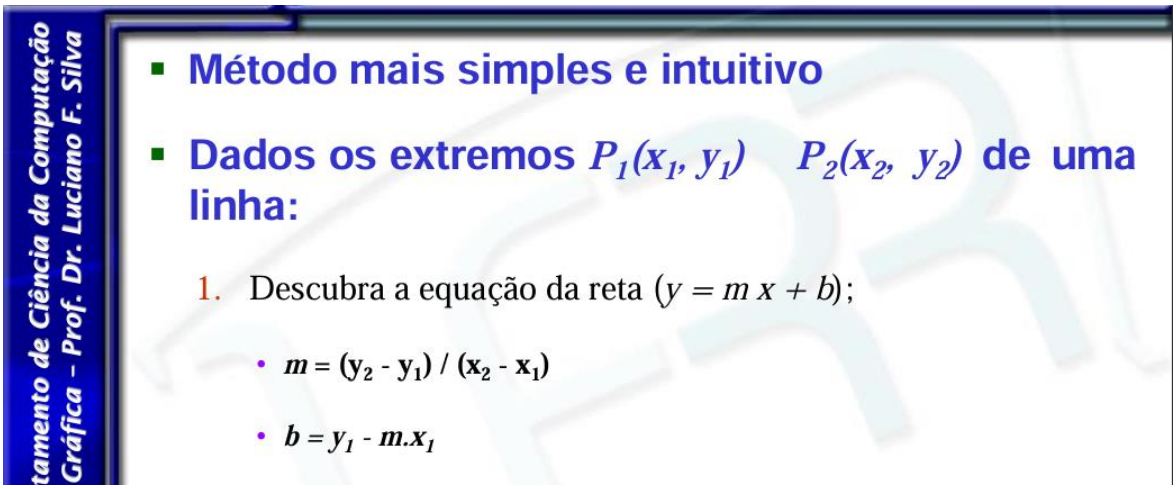
$p = p + 2\Delta y$ (se não mudar y)
$p = p + 2\Delta y - 2\Delta x$ (se mudar y)

Implementação

Inicialmente, foi feita uma tela, de resolução 800 x 600, que indica qual número o usuário deve selecionar para escolher o método para desenhar a reta:

```
95   # tela
96   WIDTH, HEIGHT = 800, 600
97   screen = pygame.display.set_mode((WIDTH, HEIGHT))
98   pygame.display.set_caption("Desenho de Retas")
99   screen.fill((255, 255, 255))
100  draw_grade(screen, WIDTH, HEIGHT)
101
102  font = pygame.font.Font(None, 36)
103  text = font.render("1 - Analítico // 2 - Método DDA // 3 - Bresenham", True, (0, 0, 0))
104  screen.blit(text, (20, 20))
105  pygame.display.flip()
106
```

- **Método Analítico**
  Para implementação do método Analítico, foi usada como base a fórmula dada nos slides da disciplina, com um tratamento especial para a verticalização da reta:

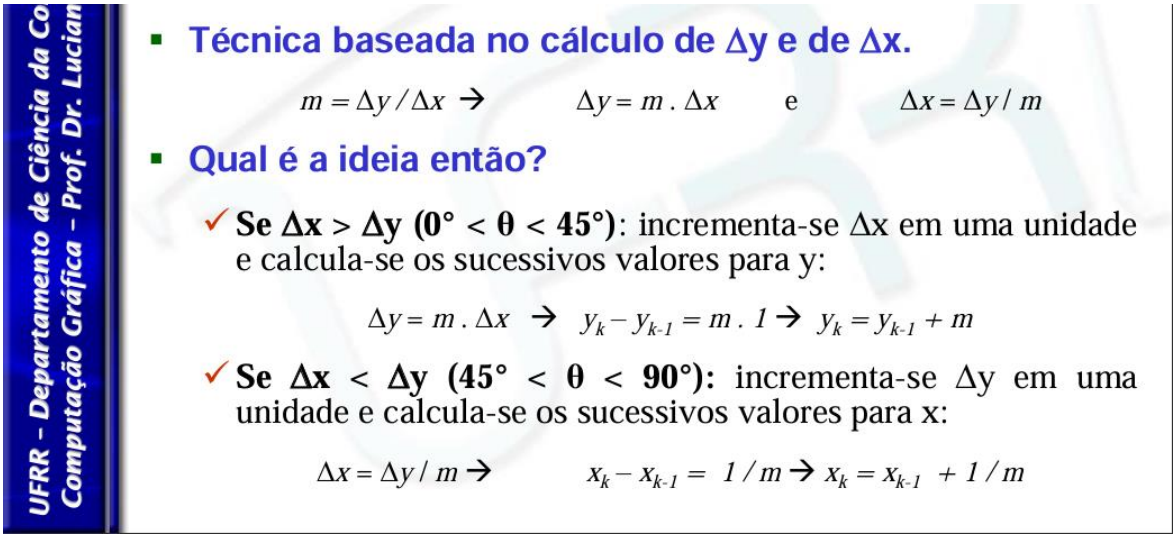


Função no código:

```python
# método Analítico
def draw_linha_analitica(surface, color, start, end):
    x1, y1 = start
    x2, y2 = end

    dx = x2 - x1
    dy = y2 - y1

    if dx == 0:  # tratamento da reta vertical
        for y in range(y1, y2 + 1):
            surface.set_at((x1, y), color)
            pygame.display.flip()
            pygame.time.delay(5)
    else:
        m = dy / dx
        b = y1 - m * x1
        for x in range(x1, x2 + 1):
            y = round(m * x + b)
            surface.set_at((x, y), color)
            pygame.display.flip()
            pygame.time.delay(5)

```

- **Método DDA**

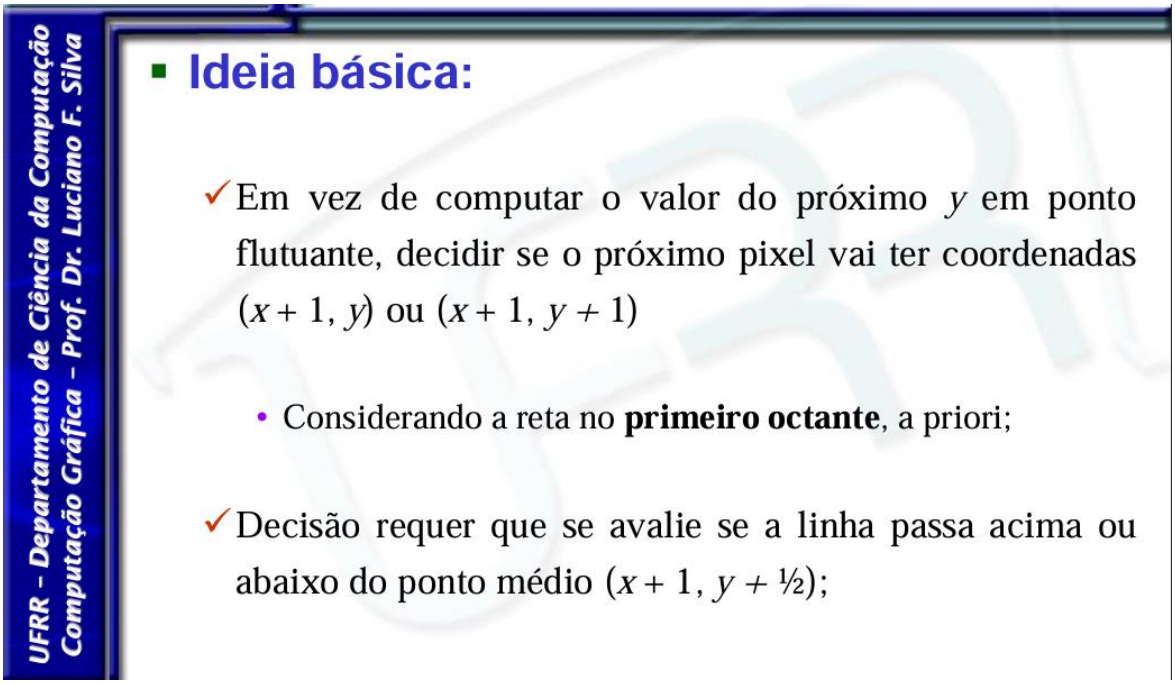
  Assim como no método anterior, também foi utilizada como base a fórmula presente nos slides da disciplina:



Função no código:

```python
# método DDA
def draw_linha_dda(surface, color, start, end, cell_size=20):
    x1, y1 = start
    x2, y2 = end

    x1, y1 = x1 // cell_size * cell_size, y1 // cell_size * cell_size
    x2, y2 = x2 // cell_size * cell_size, y2 // cell_size * cell_size

    dx = x2 - x1
    dy = y2 - y1
    steps = max(abs(dx), abs(dy))

    if steps == 0:
        return

    Xinc = dx / steps
    Yinc = dy / steps

    x, y = x1, y1
    for _ in range(steps):
        surface.set_at((round(x), round(y)), color)
        pygame.display.flip()
        pygame.time.delay(5)
        x += Xinc
        y += Yinc

```

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



Função no código:

```python
def bresenham(x1, y1, x2, y2):
    points = []

    # verifica se a inclinação é mais vertical
    steep = abs(y2 - y1) > abs(x2 - x1)

    if steep:
        x1, y1 = y1, x1
        x2, y2 = y2, x2

    # Se x1 > x2, inverte os pontos
    if x1 > x2:
        x1, x2 = x2, x1
        y1, y2 = y2, y1

    dx = x2 - x1
    dy = abs(y2 - y1)

    ystep = 1 if y1 < y2 else -1  # define a direção do incremento de y
    p = 2 * dy - dx  # parâmetro de decisão inicial como no slide
    y = y1

    for x in range(x1, x2 + 1):
        if steep:
            points.append((y, x))  # se trocou x e y, inverte de volta
        else:
            points.append((x, y))

        # decisão
        if p >= 0:
            y += ystep
            p += 2 * (dy - dx)
        else:
            p += 2 * dy

    return points
```

Resultados

- **Método Analítico (Aparência da reta)**
  - Se a inclinação for muito grande ou muito pequena, pode apresentar lacunas entre os pixels.
  - Para retas verticais (x1 = x2), precisa de um tratamento especial.

- **Método DDA (Aparência da reta)**
  - Retas são mais suaves e contínuas em comparação ao Algoritmo Analítico.
  - Pode apresentar artefatos de arredondamento em certos ângulos.

- **Método Algoritmo de Bresenham (Aparência da reta)**
  - As retas são nítidas e bem conectadas.
  - Não há lacunas ou artefatos como nos métodos anteriores.

Comparações

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

- O **método Analítico** possui as seguintes vantagens:
    1. Simples de entender e implementar.
    2. Funciona bem para coordenadas com valores reais.
- Também possui as seguintes desvantagens:
    1. Pode gerar erros de arredondamento ao converter coordenadas decimais para inteiros.
    2. Pode criar espaços entre os pixels, especialmente para inclinações pequenas ou grandes.

Além de também ter outros problemas como efetuar muitos cálculos no processo – eficiência computacional e a escolha do pixel não ser um fator considerado na elaboração da solução, podendo ser qualquer uma das redondezas do número obtido nas contas efetuadas.

- Já o **método DDA** leva vantagem em:
    1. Mais eficiente do que o Analítico, pois elimina multiplicações complexas.
    2. Gera retas mais suaves e com menos lacunas.
- Possuindo o seguinte ponto negativo:
    1. Ainda utiliza operações com números de ponto flutuante, o que pode ser mais lento do que operações inteiras.

Além de também estar sujeito a erros de arredondamento e pode ser lento em grande escala.

- Por fim, o **algoritmo de Bresenham** é o mais vantajoso:
    1. Extremamente eficiente por evitar operações com números decimais.
    2. Produz uma reta perfeita sem lacunas.
    3. Ideal para aplicações em hardware gráfico de baixo nível.
- Já seu ponto negativo:
    1. Pode ser um pouco mais difícil de entender e implementar em comparação com os outros algoritmos.

Conclusão

Por fim, quando se trata de aprender o conceito matemático, o Algoritmo Analítico é o mais recomendado. Enquanto quando o intuito é ter equilíbrio entre suavidade e desempenho, o método DDA se sobressai. E para oferecer eficiência e precisão máxima, o algoritmo de Bresenham é o mais indicado, além disso, o algoritmo de Bresenham é o mais utilizado para gráficos em jogos e sistemas embarcados devido à sua eficiência. Já a principal aplicação para o DDA pode ser útil para renderizações gráficas onde a suavidade é mais importante do que a eficiência. E o método Analítico é bom para estudos matemáticos, mas pode não ser a melhor escolha para aplicações de alto desempenho. Em resumo, foi possível obter os resultados esperados de cada algoritmo.