ENGENHARIA DE COMPUTAÇÃO

DISCIPLINA:
COMPUTAÇÃO GRÁFICA

RASTERIZAÇÃO DE RETAS E POLÍGONOS

PAULO RUBEM OLIVEIRA UCHÔA JÚNIOR

ANA KAROLINA COSTA DA SILVA

Fortaleza - CE

Introdução

O trabalho apresentado a seguir foi desenvolvido baseando-se nos algoritmos apresentados nas subseções 5.2.1 e 5.2.2 do livro Computação Gráfica Teoria e Prática - Eduardo Azevedo, os quais foram apresentados na aula do dia 10/09/2021. Tendo como critérios de implementação:

- Obter 3 imagens da rasterização de 4 semiretas diferentes para 3 resoluções diferentes e comparar as retas. Levando em conta uma implementação com as retas em posições distintas, onde uma estará na Horizontal e outra na Vertical. Também atende os casos onde delaX > deltaY e deltaY > detalX.

- Obter 6 imagens de rasterização de polígonos, para triángulo equilátero, quadrado e hexágono, sendo duas imagens para cada e cada imagem em 3 resoluções.

Implementação da Rasterização de Retas e Polígonos

O algoritmo consiste em realizar cálculos para determinar os pontos que compõem as retas, sendo assim temo como entradas os pontos extremos da reta e com a função de geração de pontos, é feito o cálculo para chegar a todos os pontos que ligam esses dois ponto iniciais, sendo esses os pontos que serão pintados, dessa forma foi criado um array, chamado **data** que armazena um conjunto de pontos que devem ser pintados.
Na construção visual do gráfico foram utilizados duas views, a **gridBody** que corresponde a grade xadrez, e a **lineBody** que corresponde aos pontos roxos. O **lineBody** faz uso do array data, de forma que para cada ponto dentro do array seja criado um quadrilátero colorido.

```
// É calculado o deltaX e o deltaY.
deltaX = CGFloat(x2 - x1)

deltaY = CGFloat(y2 - y1)

// Em seguida, é obtido o valor de m e b.
m = (deltaX == 0) ? 0 : deltaY/deltaX
b = CGFloat(CGFloat(y1) - m*CGFloat(x1))

// Por fim, é adicionado o ponto inicial, e começa a interação responsável por
 adicionar no array data, os pontos que devem ser pintados e formam as retas e
 o casco do poligono.
data.append(Point(x: Int(x1), y: Int(y1)))

if(abs(deltaX) > abs(deltaY)) {
    // Caso sim, o loop será realizado passando por cada ponto x do gráfico,
     começando no x1, até que o valor de x1 < x2-1.
    while x1 < x2-1 {
        x1 += 1
        y1 = Int(m*CGFloat(x1) + b)
        matrix[x1][y1] = 1
        data.append(Point(x: Int(x1), y: Int(y1)))

    }
    // A outra condição se dá quando o valor absoluto do deltaY é maior que o
     valor absoluto de deltaX.
} else if(abs(deltaX) < abs(deltaY)) {
    // Nesse caso, o loop será realizado passando por cada ponto y do gráfico,
     começando no y1, até que o valor de y1 < y2-1.
    while y1 < y2-1 {
        y1 += 1
        x1 = (m == 0) ? x2 : Int((Float(y1) - Float(b))/Float(m))
        matrix[x1][y1] = 1
        data.append(Point(x: Int(x1), y: Int(y1)))
```

Em casos onde os objetos são polígonos e não retas, também é utilizada a seguinte função, que recebe o array inicial **data**, que forma o casco do polígono e então essa função varre uma matriz que segue o tamanho da resolução escolhida, sendo inicialmente 50x50, através dessa varredura é feita a identificação e adição dos pontos internos do polígono ao array **fillData.**

```
//Função onde é feita a varredura da matriz *matrix* e são identificados os pontos que
 formaram o fillData
mutating func fillPoligon(lines: [Point]) {

    var x_start = 0
    var x_end = 0
    let width = Int(matrixValue)

    //Aqui se inicia a interação para geração dos pontos que serão preenchidos, aos
     quais ficarão no array dataFill. Nessa interação fazemos uma varredura na matriz
     da *matrix* onde analisamos quais linhas e colunas tem pelo menos dois pixels que
     compõem array *data* e assim são preenchidos os pontos que interligam esses pontos
     do casco do polígono.
    for line in 0...width-1  {
        x_start = -1
        x_end = -1
        for column in 0...width-1  {
            if self.matrix[line][column] as! Int == 1 {
                x_start = column
                break
            }
        }

        for column in (0...width-1).reversed()  {
            if self.matrix[line][column] as! Int == 1 {
                x_end = column
                break
            }
        }
        if x_start != -1 && x_end != -1 {
            for column in x_start..<x_end{
                dataFill.append(Point(x: line, y: column))
            }
        }
```

Para se implementar a rasterização de polígono é feito o uso do mesmo código da rasterização de retas, porém temos a adição de uma função, onde é implementado um processo para se identificar os pontos que compõem a parte interna do polígono.

Esse processo é feito através de uma interação que é iniciada após a função inicial **gerarPontos**, a nova função de preenchimento dos polígonos recebe o array com os pontos que compõem as retas e a matriz total. Dessa forma se passa por todas as linhas da matriz identificando quais linhas têm pelo menos pelo menos dois pontos pintados e assim são capturados esses pontos e preenchidos os pontos que ligam os dois. Assim é gerado um novo array de pontos a serem preenchidos que também será passado para a função que gera as imagens.

Resultados

Como pode ser visto nas imagens a seguir a uma grande diferença na nossa percepção de um objeto dependendo de sua resolução, cada objeto a seguir apresenta 3 resoluções diferentes para análise.

A primeira imagem começa com uma resolução de 50x50, onde é possível perceber claramente cada ponto pintado pelo algoritmo, e possui um serrilhado elevado.

Ao seguir para a segunda imagem com um aumento na resolução para 75x75, é evidente que o serrilhado começa a diminuir ocorrendo uma suavização na reta.

Por fim, na terceira imagem com uma resolução de 100x100, ocorre um nível de suavização ainda maior, levando a uma sensação de que cada vez mais parece uma reta quase sem falhas.

Assim, é inegável que o aumento da resolução causa o efeito de suavização dos objetos, e caso fosse aumentado ainda mais a resolução das imagens seria cada vez mais próximo de uma reta sem pixels tão evidentes.

Acerca das retas geradas podemos ver a relação de seus pontos com a construção dos deltas e a forma como as mesmas se deslocaram pelo gráfico.

- Reta 1:
  - Ponto Inicial = (0, 40)
  - Ponto Final = (40, 40)
  - Com esses pontos, é possível perceber que devido a posição Y inicial ser igual a posição Y final, a reta possui deslocamento horizontal.

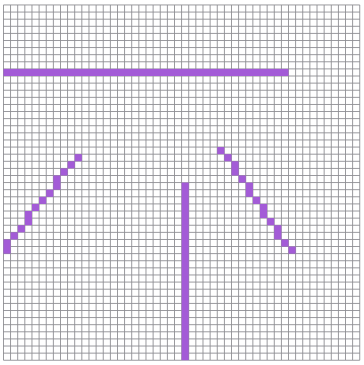
- Reta 2:
  - Ponto Inicial = (25,0)
  - Ponto Final = (25, 25)
  - Com esses pontos, é possível perceber que devido a posição Y inicial ser igual a posição Y final, a reta possui deslocamento horizontal.

- Reta 3:
  - Ponto Inicial = (0,15)
  - Ponto Final = (11, 29)
  - Com esses pontos, é possível perceber que o **deltaY** essa reta é maior que o **deltaX**, ou seja a reta possui um crescimento maior na vertical.

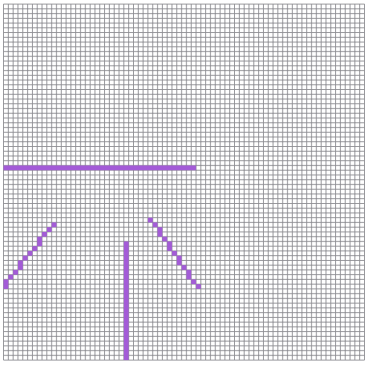
- Reta 4:
  - Ponto Inicial = (40,15)
  - Ponto Final = (30, 30)
  - Com esses pontos, é possível perceber que o **deltaY** essa reta é maior que o **deltaX**, ou seja a reta possui um crescimento maior na vertical.

Retas

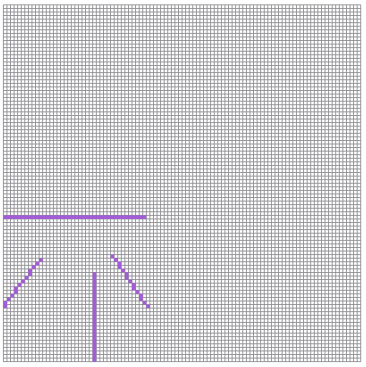
Resolution: 50 x 50



Resolution: 75 x 75

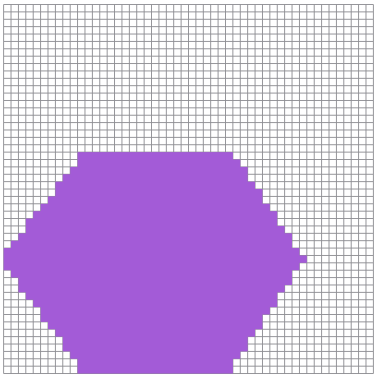


Resolution: 100 x 100

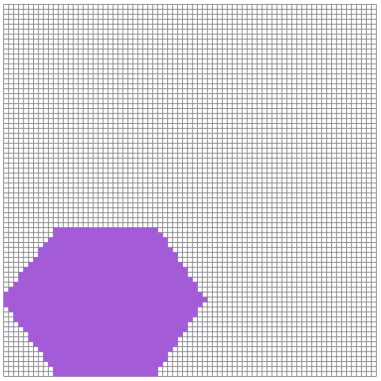


Hexágono 1

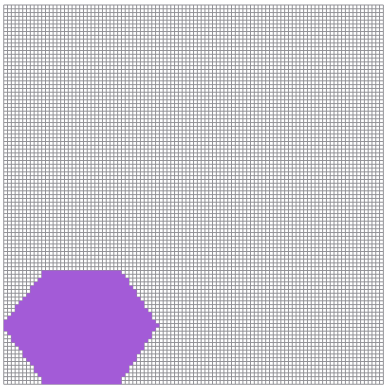
Resolution: 50 x 50



Resolution: 75 x 75

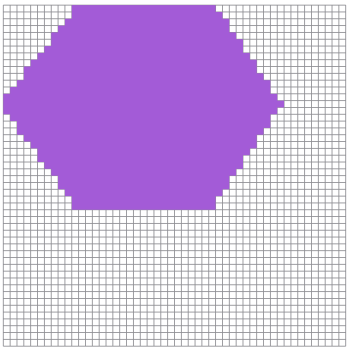


Resolution: 100 x 100

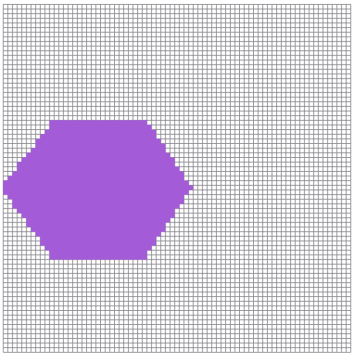


Hexágono 2

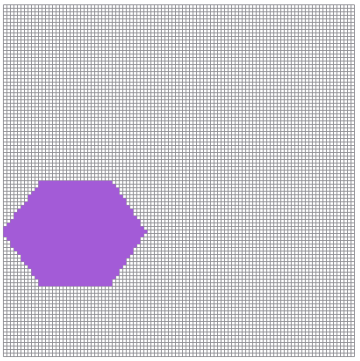
Resolution: 50 x 50



Resolution: 75 x 75

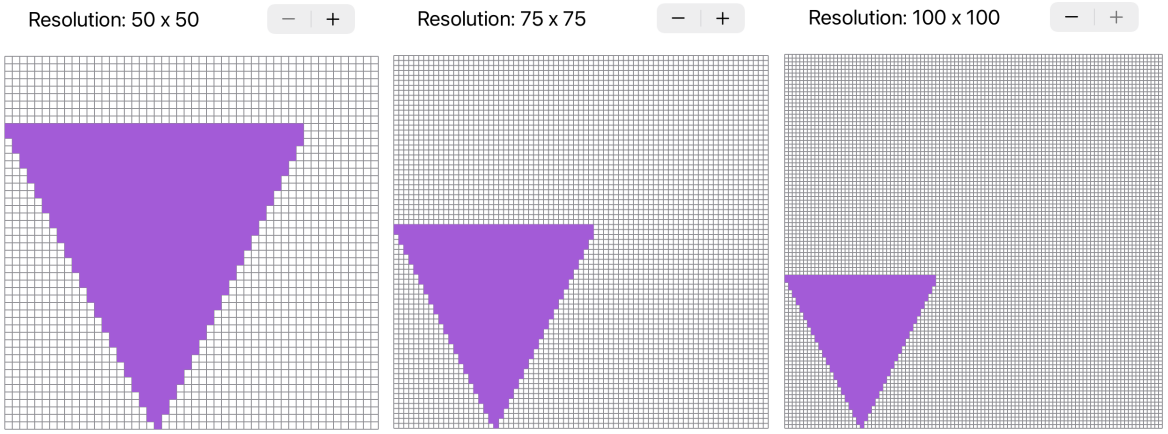


Resolution: 100 x 100



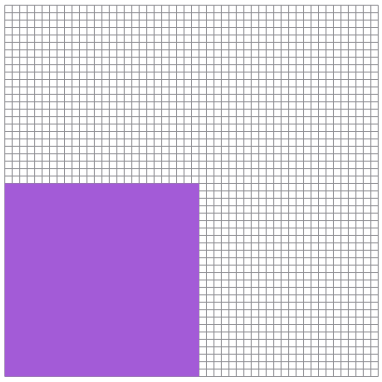
Triângulo 1

Triângulo 2

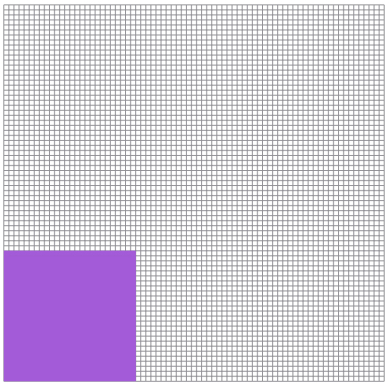


Quadrado 1

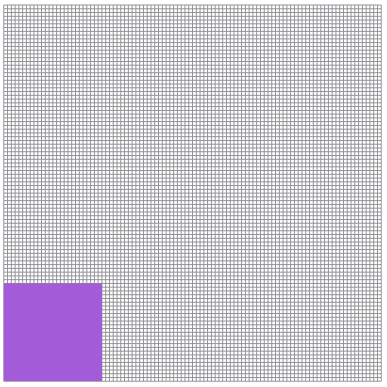
Resolution: 50 x 50 − +



Resolution: 75 x 75 − +

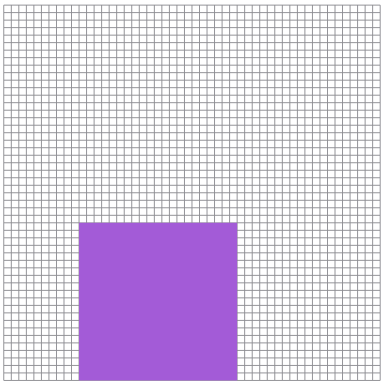


Resolution: 100 x 100 − +

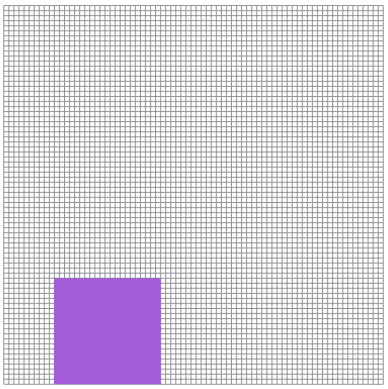


Quadrado 2

Resolution: 50 x 50 − +



Resolution: 75 x 75 − +



Resolution: 100 x 100 − +