

# CI1338 - Segundo Trabalho

Eduardo Mathias de Souza GRR20190428  
ems19@inf.ufpr.br

## Introdução

O trabalho consiste em fazer uma implementação do problema triangulação de polígonos. Neste problema é dado um polígono (em ordem horária ou anti horária) e como saída temos uma triangulação do polígono dado, através de uma estrutura de dados para representar a mesma.

Essa estrutura de dados pode ser descrita de maneira tal como:

- Cada vértice do polígono dado recebe um índice, de 1 a  $n$ , e suas coordenadas são armazenadas em um vetor indexado por estes índices.
- Cada triângulo recebe um índice, de 1 a  $m$ , e um vetor indexado por estes índices deve conter: três campos para os vértices do triângulo,  $V_1$ ,  $V_2$ , e  $V_3$ , com os índices dos três vértices e três campos com os índices dos triângulos vizinhos,  $T_1$ ,  $T_2$  e  $T_3$ , de forma que o triângulo  $T_i$  seja oposto ao vértice  $V_i$ . Os vértices de um triângulo devem estar em ordem horária e o primeiro índice ( $V_1$ ) deve ser o menor.

A implementação para resolução do problema veio do algoritmo Ear Clipping, feito por Hossam ElGindy, Hazel Everett e Godfried Toussaint, que como visto em aula, se baseia no teorema das duas orelhas (todo polígono simples com mais de três vértices tem pelo menos duas orelhas, ou seja, vértices que podem ser removidos do polígono sem introduzir nenhum cruzamento). O algoritmo consiste em encontrar essa orelha, removê-la do polígono (o que resulta em um novo polígono que ainda atende às condições) e repetir até que reste apenas um triângulo.

Esse algoritmo só funciona em polígonos sem buracos e sua execução tem complexidade de tempo de  $O(n^2)$ .

## Implementação

A resolução do trabalho foi feita por um algoritmo desenvolvido em C que possui duas estruturas de dados [adaptando das descritas pelo enunciado do trabalho 2] Point (definida por um int  $x$  e  $y$ , para representar respectivamente a coordenada  $x$  e  $y$  do ponto, um int index para representar o índice do ponto e um int opposite para guardar qual é o triângulo oposto àquele ponto) e Triangle (definida por três Point ( $a$ ,  $b$ ,  $c$ ) que representam os três vértices do triângulo e um inteiro index para representar o índice daquele triângulo).

Lê-se da linha de entrada com scanf e criamos um vetor de pontos para guardarmos todos os pontos passados, com seus respectivos índices e triângulos opostos como 0. Após essa primeira etapa, criamos também um vetor de triângulos para realizar a triangulação dos pontos.

Após definida a estrutura de dados, definimos os algoritmos da seguinte maneira:

- triangulate (Recebe como entrada vetor de pontos, a quantidade de pontos, o vetor de triângulos e o número de triângulos [iniciado em 0])
  1. Na primeira parte dessa função, criamos um vetor auxiliar chamado índices para realizarmos as operações em cima dos pontos e determinamos a

orientação (sentido horário ou anti-horário) do polígono dado pelos pontos recebidos, calculando o produto cruzado de três pontos consecutivos. Se o produto cruzado for diferente de zero, isso significa que os pontos não são colineares, e a orientação é definida de acordo.

- Se a orientação for horária, o vetor auxiliar recebe os pontos de forma reversa.
  - 2. Na parte principal da função, ela verifica repetidamente se há orelhas (triângulos) no polígono e as adiciona ao array de triângulos. Utilizando da função `is_ear` para determinar se um determinado trio de pontos é uma orelha ou não. Quando uma orelha é encontrada, o triângulo é adicionado ao array de triângulos, o array de índices é atualizado para remover o vértice da orelha, o número de pontos restantes é reduzido e o número de triângulos é atualizado. O loop continua até que não haja mais pontos para analisar restantes.
  - 3. No fim, damos free no vetor auxiliar índice.
- `is_ear` (Recebe um trio de pontos candidato para ser orelha, o vetor de pontos e o número de pontos)
    - 1. Verificamos através do produto vetorial se o trio de pontos candidatos é uma orelha de fato. Se não for, retorna falso
    - 2. Verifica se o triângulo formado pelos três pontos candidatos possui algum ponto do polígono dentro deles. Se existir, retorna falso
    - 3. Retorna verdadeiro

A função `triangulate` roda em uma complexidade de  $O(n^2)$  e a função `is_ear` roda em uma complexidade de  $O(n)$ . Logo,  $O(n^2)$

Agora, depois de termos as triangulações feitas presentes no vetor de triângulo, temos mais duas funções de pós-processamento:

- `find_opposite` (Recebe como entrada o vetor de triângulos)  
Para cada vértice dentro de um triângulo  $x$ , encontramos o triângulo vizinho e oposto a esse vértice. Para isso, verificamos se o ponto não é um vértice do triângulo vizinho e se os outros dois pontos são vértices do triângulo vizinho.  
A complexidade dessa função é  $O(n^2)$
- `order_points_triangle` (Recebe como entrada o vetor de triângulos)  
Essa função é responsável por ordenar os pontos do triângulo em ordem de menor vértice e horária.

## Exemplos utilizados para teste:

- 1) 4  
1 4  
1 20  
15 20  
15 4

Que resultou em:

Pontos

1: (1, 4)

2: (1, 20)

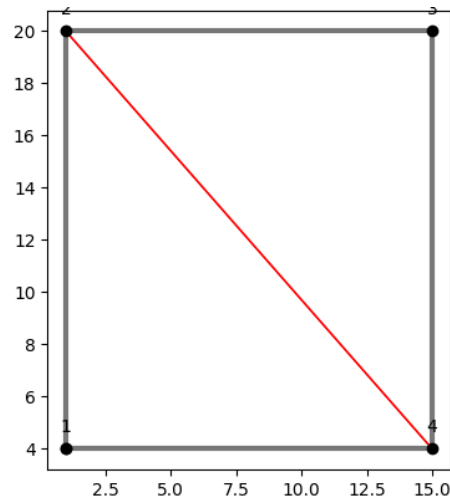
3: (15, 20)

4: (15, 4)

2 triângulos:

Triângulo 1: (2, 1, 20), (3, 15, 20), (4, 15, 4) (0 2 0)

Triângulo 2: (1, 1, 4), (2, 1, 20), (4, 15, 4) (1 0 0)



2) 6

0 0

5 0

7 2

5 4

3 4

1 2

Que resultou em:

Pontos

1: (0, 0)

2: (5, 0)

3: (7, 2)

4: (5, 4)

5: (3, 4)

6: (1, 2)

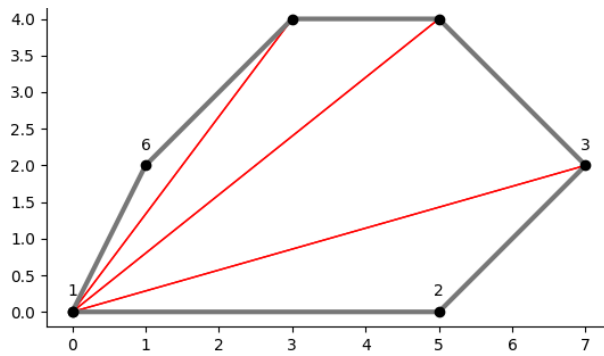
4 triângulos:

Triângulo 1: (1, 0, 0), (2, 5, 0), (3, 7, 2) (0 2 0)

Triângulo 2: (1, 0, 0), (3, 7, 2), (4, 5, 4) (0 0 1)

Triângulo 3: (1, 0, 0), (4, 5, 4), (5, 3, 4) (0 0 2)

Triângulo 4: (1, 0, 0), (5, 3, 4), (6, 1, 2) (0 0 3)



3) 5  
0 0  
2 0  
2 2  
1 3  
0 2

Que resultou em:

Pontos

1: (0, 0)

2: (2, 0)

3: (2, 2)

4: (1, 3)

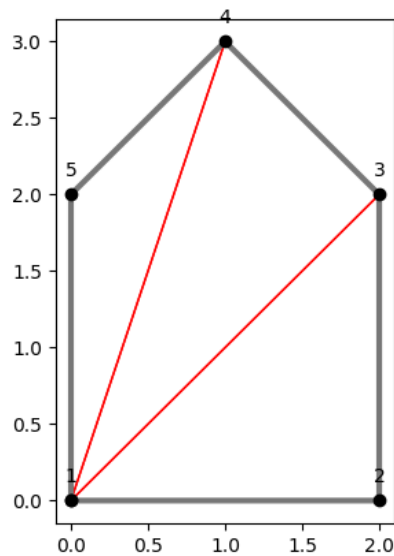
5: (0, 2)

3 triângulos:

Triângulo 1: (1, 0, 0), (2, 2, 0), (3, 2, 2) (0 2 0)

Triângulo 2: (1, 0, 0), (3, 2, 2), (4, 1, 3) (0 0 1)


Triângulo 3: (1, 0, 0), (4, 1, 3), (5, 0, 2) (0 0 2)



Referências:

[https://en.wikipedia.org/wiki/Polygon\\_triangulation](https://en.wikipedia.org/wiki/Polygon_triangulation)

<http://homepages.math.uic.edu/~jan/mcs481/triangulating.pdf>

 The Art Gallery Problem and Polygon Triangulation (4/4) | Computational Geometry - L...