# Geometria Computacional - Triangulação

Nico I. G. Ramos

*GRR20210574*

Julho 2023

## 1 Introdução

O problema da triangulação de polígonos consiste em achar uma divisão em triângulos de um polígono, essa divisão possuí diversas aplicações, como achar uma determinada região em um mapa e representar objetos em animações e jogos.

## 2 Problema

Para este trabalho, o objetivo era implementar uma estrutura de dados para representar a triangulação de um polígono. Nessa estrutura, cada triângulo deveria conter o índice de cada vértice e o índice do triângulo oposto a cada um de seus vértices.

## 3 Modelagem

Para realizar a triangulação, foi escolhido um algoritmo recursivo no qual se procura dividir o polígono em dois e, para cada parte, a triangulação é feita até que a parte dividida seja um triângulo.

Para achar onde dividir o polígono, é feito uma busca pelo seu vértice mais a esquerda, a partir do qual se traça uma aresta entre o vértice anterior e o próximo a ele. Com isso, é formado um triângulo com vértices nesses três pontos e, para cada outro vértice do polígono, é verificado se ele está dentro desse triângulo ou não.

Caso algum vértice esteja dentro do triângulo, é pego o mais a esquerda dentre os que são interiores a ele e a aresta é trocada para ser formada pelo vértice mais a esquerda do polígono e o vértice mais a esquerda interior ao triângulo.

Tendo achado a aresta, é feito duas chamadas recursivas, uma para cada parte do polígono, e, após o retorno delas, para cada um dos triângulos retornados, é feita uma busca para saber qual das arestas é a diagonal que chamou a recursão e o índice dela é atribuída ao vértice oposto no outro triângulo.

Por fim, é feita uma busca nos dois triângulos para saber qual deles ficou com a diagonal da recursão anterior a eles e é retornado o índice do triângulo que a contém.

## 4 Implementação

Para implementar esse algoritmo, foram necessárias três bibliotecas, uma de triangulação, uma para construir e operar com figuras geométricas e outra para realizar operações de geometria, como intersecção e verificar o sentido de um polígono.

Para garantir a padronização entre as triangulações e facilitar a implementação do algoritmo, antes de realizar a triangulação é verificado se o polígono é horário, caso não seja, seu sentido é alterado e o índice dos seus vértices também.

Para realizar a triangulação, duas variáveis globais foram utilizadas: um vetor de polígonos e um contador de índices. A cada chamada recursiva, dois parâmetros são passados: um polígono formado por pontos do polígono pai, o que preserva os índices entre as chamadas, e a di-

agonal que deu origem a ele. Além disso, a divisão do polígono é feita de maneira que ele continue horário.

Caso ao chamar a função de triangulação o polígono passado como parâmetro possuir tamanho três, isso significa que um triângulo foi achado e ele deve ser adicionado ao vetor global da triangulação. Para isso, é incrementado o índice global, é criado um triângulo de índice $i$ sem nenhum vizinho e ele é adicionado ao vetor da triangulação no índice $i$.

A estrutura escolhida para representar os triângulos consiste em um vetor de vértices e um vetor de vizinhos - no qual cada um consiste de uma aresta e o índice do triângulo com o qual faz divisa. Assim, o vértice $i$ é oposto ao vizinho $i$ e ao triângulo com índice no vizinho $i$.

O **Algoritmo 1** e o **Algoritmo 2**, apresentam os pseudocódigos para os dois algoritmos mais importantes para esse trabalho, respectivamente: a triangulação em si e a busca pela diagonal.

A **Triangulação(P, D)**, que é recursiva e recebe como parâmetro um polígono e a diagonal que deu origem a ele, e a **Diagonal(P)**, que recebe como parâmetro o polígono para o qual se deseja achar uma orelha formada pelo vértice mais a esquerda e seus vizinhos ou, caso não seja possível, uma aresta que o divida entre o vértice interior à orelha que seja o mais a esquerda da aresta formada pelos vizinhos do vértice mais a esquerda do polígono.

**Algoritmo 1:** Triangulação(P, D)

```
if size(P) == 3 then
    indice++;
    Crie o triângulo t;
    T[i] ← t;
    Retorne indice;
else
    diagonal ← Diagonal(P);
    Divida P em dois polígonos na diagonal;
    for Para cada divisão de P do
        Faça a Triangulação(P, diagonal)
    end
    for Para cada índice i retornado pelas
     triangulações do
        t ← T[i];
        Ache no triângulo t o índice do vizinho que
         ficou com a diagonal;
        Adicione o índice do vizinho no vértice
         oposto do outro triângulo retornado;
    end
    Ache a diagonal D entre os triângulos
     retornados;
    Retorne o índice do triângulo que ficou com
     a diagonal D;
end
```

**Algoritmo 2:** Diagonal(P)

```
l ← ponto_mais_esquerda(P);
dig ← Aresta(proximo(l), anterior(l));
esq ← oo;
for para cada vértice v em P do
    if v != l e v não está na diagonal then
        if v está no triângulo(l, dig) then
            if v está mais a esquerda do que esq
             then
                esq ← v;
            end
        end
    end
end
if esq != oo then
    dig ← Aresta(l, esq);
end
Retorne dig;
```

# 5 Resultados

Para verificar se os algoritmos escolhidos e a implementação realizada resolvem o problema corretamente, as seguintes triangulações foram feitas: polígono horário não convexo, quadrado horário e anti-horário, um "L"anti-horário, polígono com diagonais colineares, um único corte que não permite orelha, espiral anti-horária, estrela anti-horária, triagulóide anti-horário e "U"anti-horário.

Os testes tiveram como objetivo verificar se a triangulação é realizada de maneira correta tanto para polígonos horários como para anti-horários, se o corte na orelha é correto e se o corte quando não é possível dividir na orelha também é correto.

Além disso, os testes também buscarem verificar a corretude da divisão quando o polígono é particionado em dois polígonos com mais de três vértices, o que poderia causar problemas na atribuição dos vizinhos.

## 5.1 Divisão em polígono simples

Nos dois próximos polígonos, o objetivo foi verificar se a divisão seria correta em um polígono simples, seja ele horário ou anti-horário.

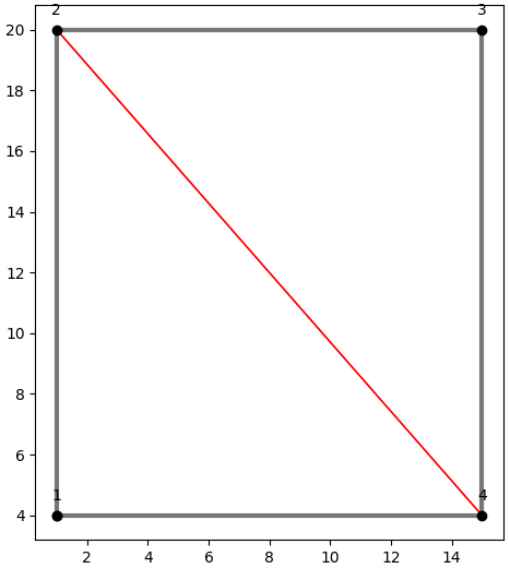


Figura 1: Quadrado horário

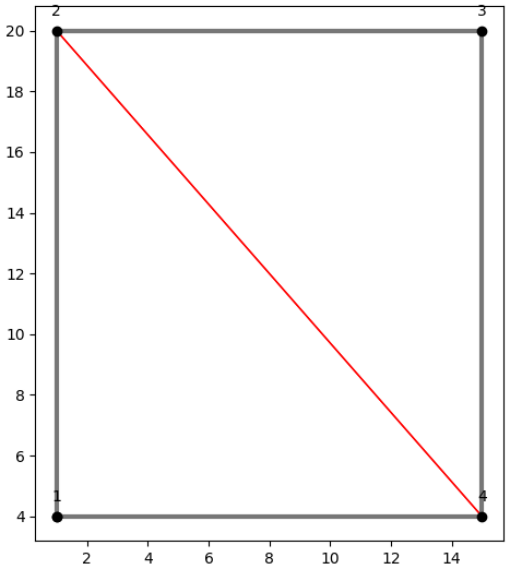


Figura 2: Quadrado anti-horário

Em ambos, tanto a divisão quanto a atribuição dos vizinhos foi correta.

## 5.2 Divisão sem orelha

Os testes tiveram como objetivo assegurar que a divisão quando a orelha não é possível foi implementada sem erros.

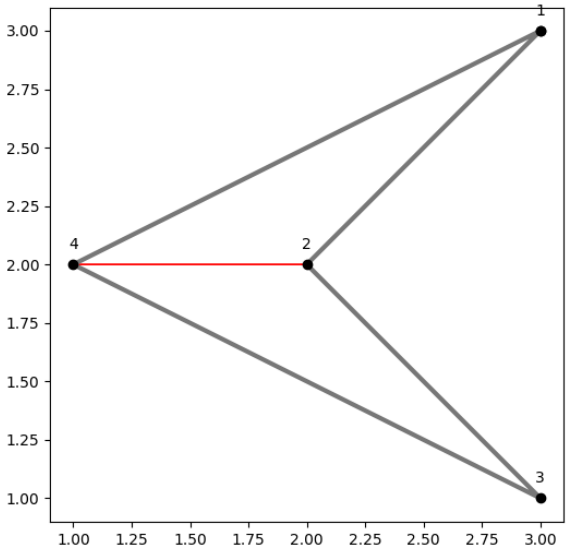


Figura 3: Polígono anti-horário sem orelha

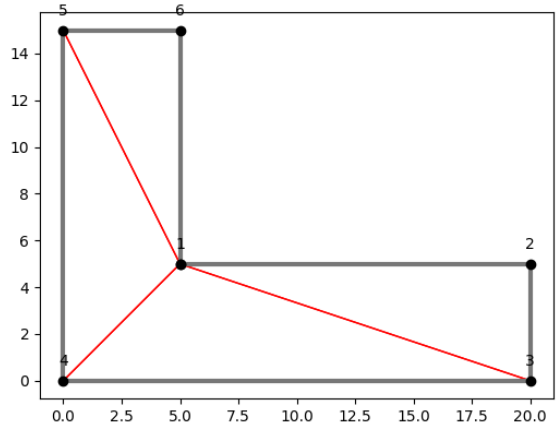


Figura 4: ”L”anti-horário

Os testes também foram satisfatórios e a implementação atribuiu corretamente os vizinhos e gerou uma triangulação válida.

## 5.3 Divisão de polígonos anti-horários

Aqui, objetivou-se verificar se a transformação dos polígonos anti-horários em horários geraria uma triangulação válida.

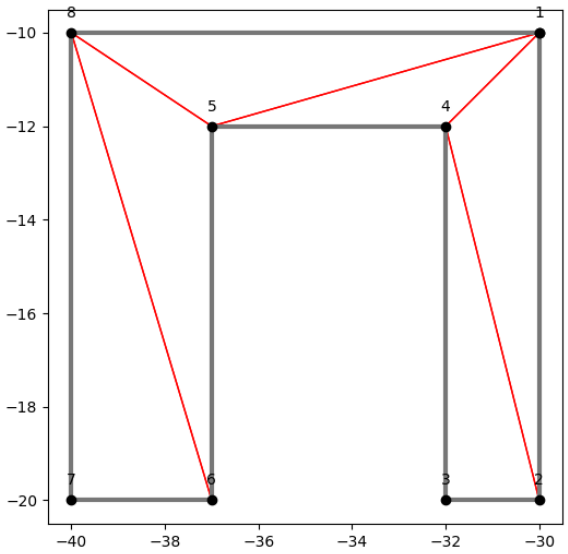


Figura 5: ”U”anti-horário

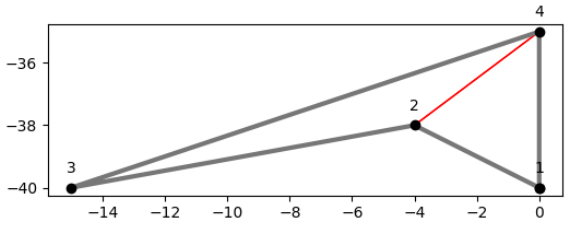


Figura 6: Triagulóide anti-horário

Em ambos os polígonos a triangulação e a atribuição dos vizinhos foi correta.

## 5.4 Divisão com vários pontos

O objetivo principal com esses testes foi verificar se o algoritmo não iria se perder com polígonos com vários pontos e que não fossem convexos, tendo que alternar entre dividir na orelha e fora dela.

Nesse polígono, o objetivo também foi testar se o algoritmo encontrava da maneira esperada o vértice a esquerda quando mais de um compartilha o mesmo $x$, se ele considera que o mais a esquerda é o de menor $y$.

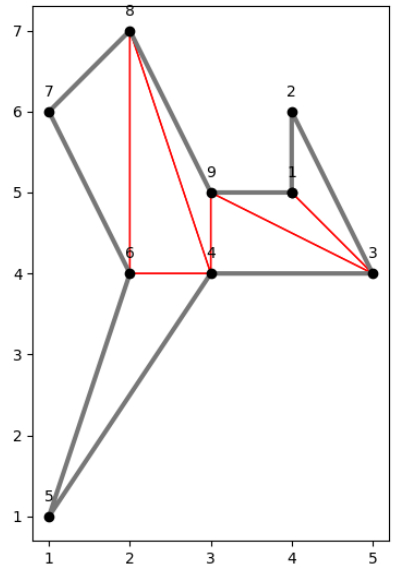


Figura 7: Polígono horário não convexo

O algoritmo selecionou corretamente o menor $x$ entre os vértices *1* e *7*, o que gerou a aresta entre os vértices *6* e *4*.

Na estrela, o objetivo foi verificar a divisão em um polígono com várias quinas, o que talvez pudesse confundir o algoritmo. É possível observar que foi gerada uma triangulação válida.

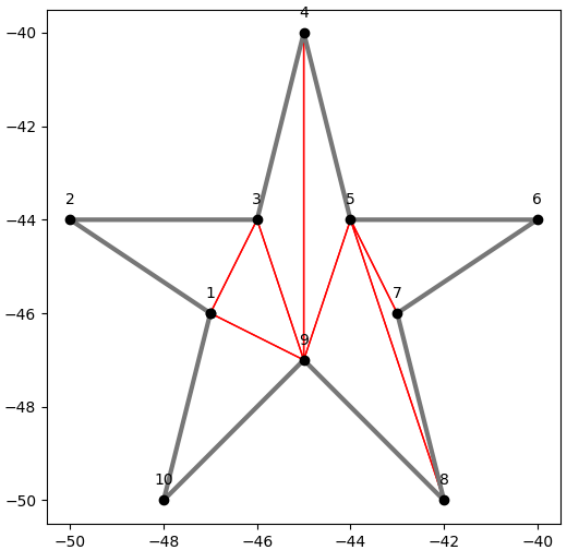


Figura 8: Estrela anti-horária

Na espiral, o objetivo foi verificar que o algoritmo não se perderia com um polígono que fizesse curvas e que não seria gerada arestas que passassem por fora do polígono.

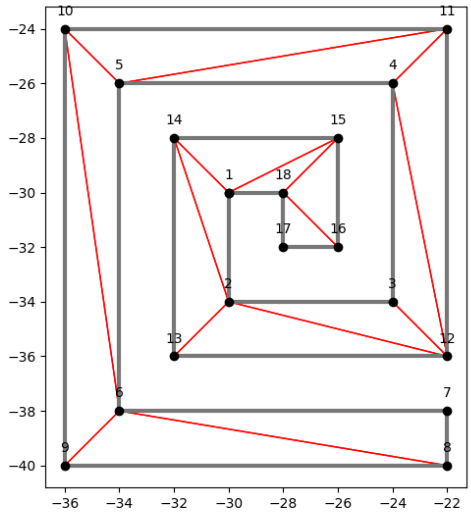

Figura 9: Espiral anti-horária

Nos três casos, além da triangulação ser válida, os vizinhos foram atribuídos corretamente.

### 5.5 Corte em pontos colineares

Com esse polígono, o objetivo foi verificar a atribuição dos vizinhos com diagonais com vértices colineares.

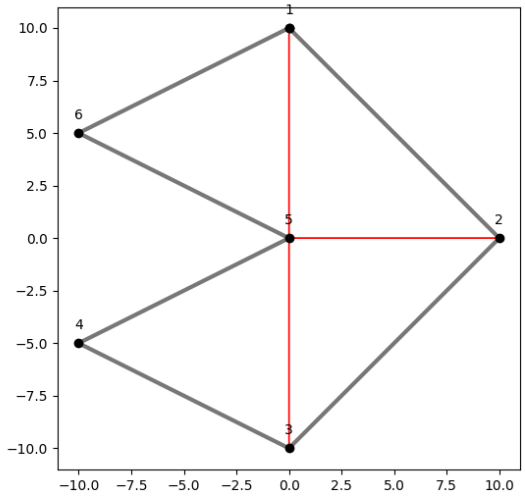

Figura 10: Polígono horário com cortes colineares

Para alguns triângulos, é difícil dizer qual deveria ser o triângulo oposto para cada vértice.

No triângulo formado pelos vértices *1, 5 e 6* o oposto ao vértice *1* pode ser tanto nenhum, quanto o triângulo formado pelos vértices *3, 4 e 5* ou o formado pelos vértices *2, 3 e 5*.

E, tanto no triângulo formado pelos vértices *1, 2 e 5* e no formado pelos vértices *2, 3 e 5* o oposto ao vértice *2* também pode ser nenhum, o triângulo formado pelos vértices *1, 5 e 6* ou o formado pelos vértices *3, 4 e 5*.

Entretanto, para o triângulo formado pelos vértices *3, 4 e 5* o oposto ao vértice *4* é o triângulo formado pelos vértices *2, 3 e 5* e, para o triângulo formado pelos vértices *1, 5 e 6* o oposto ao vértice *6* é o triângulo formado pelos vértices *1, 2 e 5*.

Com isso, embora a divisão gerada tenha sido correta, o algoritmo atribuiu de maneira errônea os vizinhos. No triângulo formado pelos vértices *3, 4 e 5*, atribuiu o oposto ao vértice *4* o triângulo formado pelos vértices *1, 2 e 5* ao invés de atribuir o formado pelos vértices *2, 3 e 5*.

## 6 Conclusão

Os algoritmos implementados resolvem o problema de gerar a triangulação. Contudo, em alguns casos, eles falham em atribuir os triângulos opostos corretamente.

O principal motivo disso ocorrer é que, na chamada recursiva, é buscado nos dois índices retornados para ela o triângulo que contém a diagonal que deu origem àquela chamada.

Entretanto, nem sempre um dos dois triângulos retornados irá conter a diagonal que criou aquela sub-divisão do polígono. O algoritmo foi implementado dessa maneira na tentativa de evitar ter de procurar a diagonal em todos os triângulos.

Uma alternativa pensada, mas não implementada, foi a de representar a diagonal por uma aresta e as duas faces que ela divide, passando essa informação na descida da recursão ao invés de pega-la no retorno. Com isso, seria preciso cuidar na divisão de um polígono que já possuí uma aresta marcada.