Relatório do Trabalho 1 de Geometria Computacional

Bruno Dal Pontte & Ruibin Mei

2 de julho de 2025

Resumo

Esse relatório tem como objetivo explicar as escolhas de implementação e o funcionamento do Trabalho 1 de Geometria Computacional. Este trabalho consiste em implementar um algoritmo que verifica, para cada um dos pontos fornecidos, dentro de quais polígonos simples dentre os fornecidos na entrada eles estão contidos. Para isso, os polígonos são classificados em simples e não simples, e também, dentre os simples, entre convexos e não convexos.

1 Introdução

1.1 Classificação de Polígonos e Pontos Interiores aos Polígonos

O programa implementado tem como objetivo verificar, para uma dada sequência de polígonos descritos por uma sequência de vértices e uma dada sequência de pontos no plano cartesiano, verificar para cada um dos pontos se eles estão contidos dentro de cada um dos polígonos simples dentre os fornecidos na entrada. Para que isso possa ser calculado, é necessário primeiro classificar os polígonos em simples e não simples, sendo que os polígonos simples podem ainda ser convexos ou não convexos.

Para tal finalidade, são implementados algoritmos de intersecção de segmentos de reta para verificar se os polígonos são simples, um algoritmo de verificar orientação das arestas para verificar se os polígonos são convexos, e por fim, após os polígonos terem sido classificados, um algoritmo de *Ray Tracing* para verificar dentro de quais polígonos simples cada ponto está contido. Para os propósitos deste trabalho, quando um ponto está exatamente sobre um vértice ou sobre uma aresta de um polígono, considera-se que esse ponto está dentro do polígono.

2 Estruturas de Dados

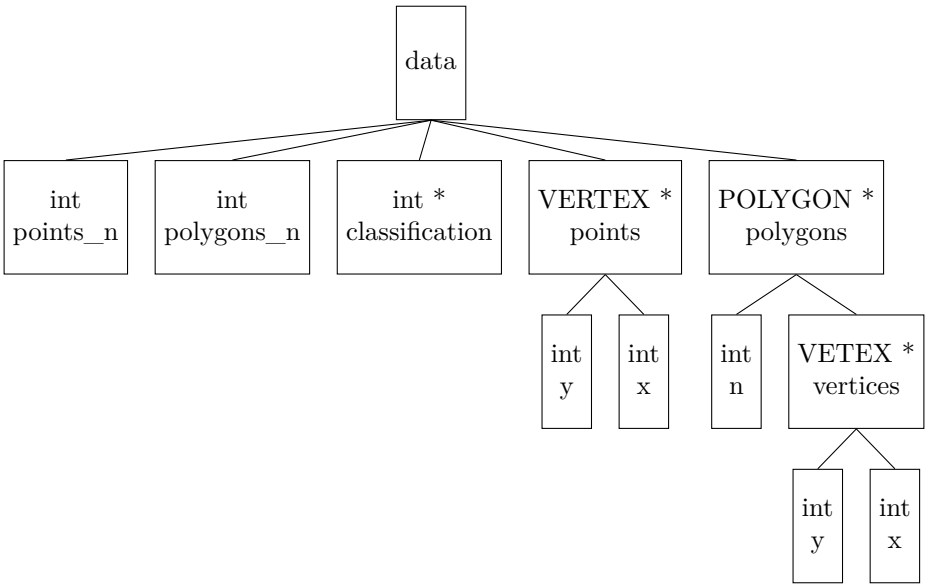
As estruturas de dados utilizadas neste trabalho são:

- Uma estrutura principal `PDATA data`, de tipo `*PDATA` (um ponteiro para `DATA`), que contém um vetor com estruturas representando os polígonos (`PPOLYGON polygons`), a fim de guardar todos os polígonos da entrada;
- Um vetor de estruturas de vértices `PVERTEX points` contido em `data`, de tipo `*PVERTEX` (um ponteiro para `VERTEX`), que armazena os pontos da entrada;
- Um vetor de inteiros (`int *classification`) contido em `data`, que representa a classificação dos polígonos da seguinte forma:
  - O valor `0` representa polígonos não simples;
  - O valor `1` representa polígonos simples e convexos;
  - O valor `2` representa polígonos simples mas não convexos.
- E por fim, dois índices inteiros `int polygons_n` e `int points_n` também contidos em `data`, que armazenam a quantidade de polígonos e de pontos, respectivamente.

A estrutura do polígono `POLYGON` (com ponteiro `*PPOLYGON`) é composta por um vetor composto por estruturas de vértices contidas no polígono, `PVERTEX vertices`, onde cada índice desse vetor contém uma estrutura `VERTEX` que armazena as coordenadas de cada vértice, e um índice `n` que armazena a quantidade de vértices contidas no polígono. Os vértices são armazenados na ordem dada pela entrada, que por convenção contém os vértices em sentido anti-horário. Porém, pelos nossos testes, o programa também funciona corretamente com entradas onde os polígonos dos vértices estão no sentido horário.

A estrutura de vértices `VERTEX` (com ponteiro `*PVERTEX`) é formada por dois valores inteiros `int x` e `int y`, os quais armazenam as coordenadas do ponto ou vértice.

Essas estruturas são apresentadas de forma gráfica no diagrama a seguir:



3 Algoritmo de Verificação de Polígonos

3.1 Intersecções entre Segmentos de Reta

Para a verificação de intersecção entre dois segmentos, foi feito o uso do produto vetorial para verificar se a orientação dos pontos é no sentido anti-horário, no sentido horário ou de maneira colinear. Ou seja, é verificado se os pares de segmentos de reta estão orientados são diferentes ou se são colineares e estão sobrepostos. Esse algoritmo foi baseado nos algoritmos explicados no *site Geeks for Geeks* [1] [2].

O produto vetorial se baseia em passar três pontos e fazer a verificação, exemplo: ponto A, B e C, formam vetores AB e BC, com isso, se o caminho A->B->C gira para esquerda (anti-horário), direita (horário) e se em linha reta (colineares), calculando assim uma direção relativa das arestas.

Caso x_2 e y_2 tenham orientação diferente em relação ao segmento $\overline{x_1 y_1}$ e vice-versa, isso significa que há intersecção entre os segmentos de reta. Porém, esse método não consegue verificar os casos colineares, nos quais um dos pontos de um dos segmentos está sobre a outra linha. Para checar esses casos, é feita uma verificação de se um dos pontos está no retângulo delimitado pelo outro segmento, ou seja, se ele não cruza a reta fora do intervalo delimitado pelo outro segmento de reta, para cada um dos quatro pontos em relação ao outro segmento. Caso isso ocorra, há intersecção, caso contrário não ocorre intersecção entre os segmentos de reta.

3.2 Polígonos Não Simples

Para verificação de polígonos não simples, foi utilizado o método mais simples, testando se algum par de aresta se intersecta. Assim, o algoritmo tem custo $O(n^2)$. Para amortizar um pouco o custo, é verificado apenas a intersecção entre os segmentos de reta i e j se $i < j$, pois a intersecção de segmentos é comutativa.

3.3 Polígonos Simples

3.3.1 Convexo e não convexo

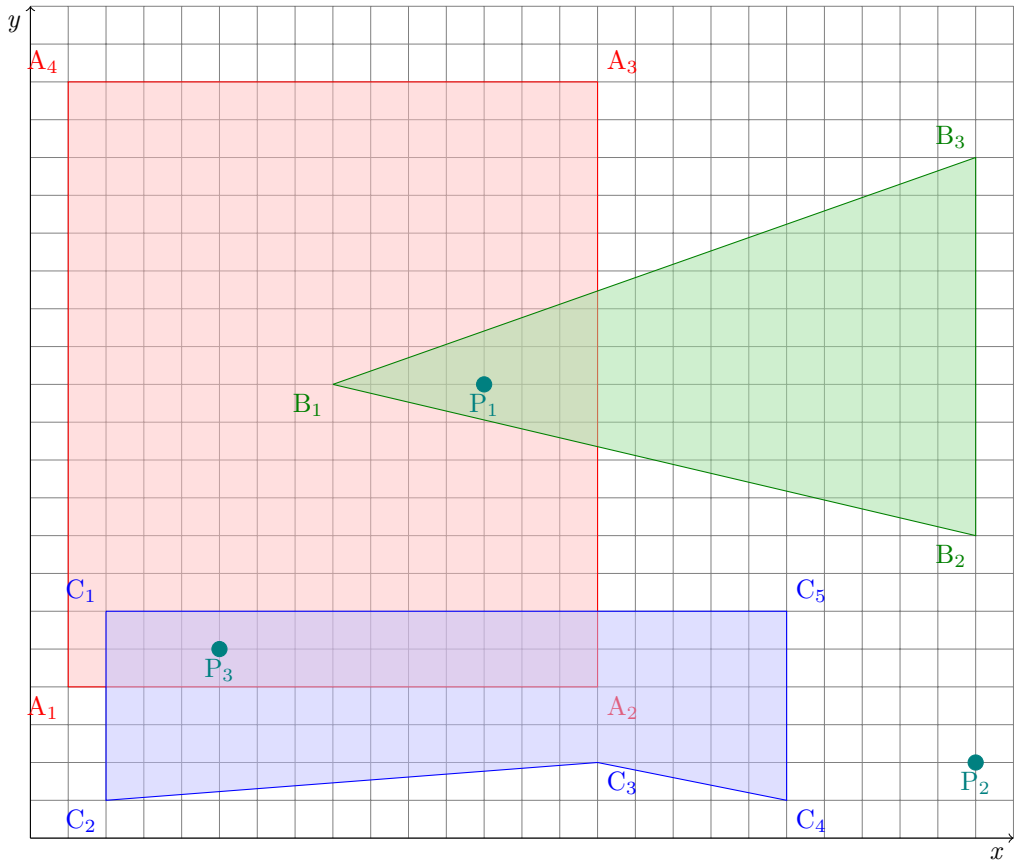
Após a verificação de polígono não simples, para determinar se o polígono simples é convexo, é verificado se todas as arestas tem o mesmo sentido de orientação (horário ou anti-horário). Dessa forma, se houver uma aresta com orientação diferente antes de terminar a varredura, o polígono já é classificado como não convexo. Essa verificação tem custo $O(n)$.

4 Algoritmo de Verificação de Pontos

Para verificar se os pontos pertencem ou não à parte interior de cada polígono simples, onde tais polígonos podem ou não ser convexos, foi utilizado o método simples de *Ray Casting*, na qual faz uma contagem em uma reta horizontal iniciando no ponto para a direita quantas vezes essa reta cruza as arestas dos polígonos. Se o número for *ímpar* então o ponto está dentro, já se esse número for *par* então o ponto está fora. Para a implementação desse algoritmo, foram utilizado como base os materiais do *site Geeks for Geeks* sobre o assunto [3] [4]. Além disso, esse algoritmo verifica casos especiais como, por exemplo, se o ponto está exatamente sobre a aresta ou vértice. Nesses casos, o algoritmo considera que o ponto está dentro do polígono.

5 Casos de Teste

Os casos de testes utilizados para verificar a corretude do algoritmo estão descritos a seguir. O primeiro desses casos de teste foi o passado pelo professor na especificação do trabalho. Ele está incluido no arquivo `entrada.txt`, e consiste dos seguintes polígonos e pontos:

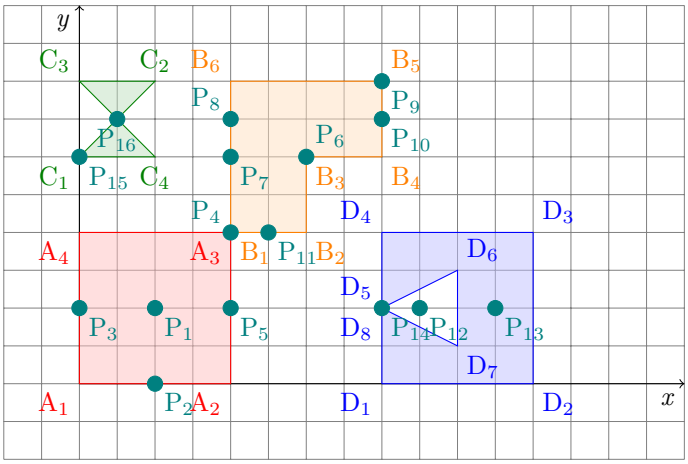


E esse arquivo de entrada leva à seguinte saída:

```
1 simples e convexo
2 simples e convexo
3 simples e nao convexo
1:1 2
2:
3:1 3
```

A outra entrada de testes utilizada está no arquivo "`entrada1.txt`". Ele consiste de quatro polígonos e alguns pontos, a fim de testar casos de borda do algoritmo. Alguns desses polígonos são não simples ou não convexos.

Abaixo está a figura correspondente à entrada do arquivo "`entrada1.txt`":



E a saída correspondente a esse arquivo de entrada:

```
1 simples e convexo
2 simples e nao convexo
3 nao simples
4 nao simples
1:1
2:1
3:1
4:1 2
5:1
6:2
7:2
8:2
9:2
10:2
11:2
12:
13:
14:
15:
16:
```

Referências

[1] Geeks for Geeks. *How to check if two given line segments intersect?* Disponível em https://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/. Acessado em abril de 2025.

[2] Geeks for Geeks. *CSES Solution - Line Segment Intersection.* Disponível em https://www.geeksforgeeks.org/cses-solution-line-segment-intersection/. Acessado em abril de 2025.

[3] Geeks for Geeks. *How to check if a given point lies inside or outside a polygon?* Disponível em https://www.geeksforgeeks.org/how-to-check-if-a-given-point-lies-inside-a-polygon/. Acessado em abril de 2025.

[4] Geeks for Geeks. *Point in Polygon in C.* Disponível em https://www.geeksforgeeks.org/point-in-polygon-in-c/. Acessado em abril de 2025.