Trabalho 1

Gabriela Fanaia De Almeida Dias Dorst (GRR20220070) [1]
João Pedro Vicente Ramalho (GRR20224169) [2]

[1]Departamento de Informática
Geometria Computacional
Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

{gfadd22 [1],jpvr22 [2]}@inf.ufpr.br

1. Introdução

Este trabalho tem como objetivo resolver um problema clássico da Geometria Computacional: a classificação de polígonos e a verificação de pontos interiores. Dado um conjunto de linhas poligonais fechadas e um conjunto de pontos no plano, o desafio consiste em classificar cada polígono como "não simples", "simples e não convexo" ou "simples e convexo". Além disso, para cada ponto dado, é necessário determinar quais dos polígonos simples (sejam convexos ou não) o contêm.

2. Fluxo do algoritmo

O algoritmo foi implementado da seguinte maneira:

1. Primeiro, o arquivo texto é lido e separado. É feita uma lista de polígonos de tamanho m e cada polígono é uma lista de vértices de tamanho n_i. Os pontos do arquivo também viram uma lista de tamanho n.
2. A partir de um laço que percorre cada polígono dentro da lista de polígonos é chamada as funções de classificação.
3. A primeira função de classificação chamada é is_convex para verificar se o polígono é convexo. Essa função tem análise assintótica $\theta(n)$. Se o polígono não for convexo o algoritmo segue. Caso seja convexo, sua classificação é adicionada ao vetor $classified$, no mesmo índice que ele ocupa no vetor de polígonos.
4. Em seguida é chamada a função is_simple para verificar se o poligono é simples. Essa função tem análise assintótica $\mathcal{O}(n^2)$. Se o retorno dessa função for False, ele é classificado no vetor de classificações como não simples.
5. Caso o polígono não seja convexo ou não seja complexo, sobra a opção de ser um polígono simples e não convexo. Então o polígono é classificado como tal.
6. Depois do vetor de classificações já pronto, ou seja, todos os polígonos foram devidamente classificados, é chamada a função $point\_inside\_polygon$, que verificará se o ponto está dentro dos polígonos simples (côncavos ou convexos). Essa função tem análise assintótica de $\mathcal{O}(p * q * v)$, sendo p o número de polígonos, q o número de pontos e v o número máximo de vértices de um polígono.

Vale destacar que a ordem das verificações foi planejada de forma a minimizar a frequência com que se atinge o pior caso de complexidade $\mathcal{O}(n^2)$. Isso ocorre porque, na configuração atual, a verificação de simplicidade do polígono só chega ao pior caso para polígonos côncavos. Polígonos convexos são filtrados anteriormente, e, no caso de polígonos complexos, é provável que uma interseção seja detectada precocemente, evitando a execução completa do algoritmo. Assim, assumindo uma distribuição uniforme entre as classes de polígonos (convexos, côncavos e complexos), apenas cerca de 1/3 das entradas resultam em pior caso quadrático.

3. Implementação das funções

3.1. Função para verificar se o polígono é convexo (`is_convex`)

1. Recebe um polígono representado por uma lista de vértices ordenados no sentido anti-horário.
2. Para cada sequência de três vértices consecutivos, calcula o produto vetorial entre os vetores formados por esses pontos.
3. O sinal do produto vetorial indica a direção da curvatura local no polígono (sentido horário ou anti-horário).
4. Se todos os produtos vetoriais tiverem o mesmo sinal, o polígono mantém uma orientação consistente e é considerado convexo.
5. Caso algum sinal seja diferente dos anteriores, a função identifica uma mudança de orientação e retorna False, indicando que o polígono é côncavo.
6. Se o laço termina sem encontrar conflitos de sinal, a função retorna True, confirmando que o polígono é convexo.

3.2. Função para verificar se o polígono é simples (`is_simple`)

1. Inicialmente, cria uma lista com todas as arestas do polígono, conectando cada par de vértices consecutivos. A última aresta conecta o último ponto ao primeiro, fechando o polígono.
2. Em seguida, verifica todas as combinações possíveis de pares de arestas, exceto aquelas que são adjacentes (ou seja, que compartilham um vértice), pois esse tipo de contato é esperado e não configura interseção.
3. Para cada par de arestas não adjacentes, testa se há interseção utilizando uma função baseada na orientação dos pontos (`do_intersect`). Essa verificação permite identificar se os segmentos se cruzam internamente.
4. Se alguma interseção for detectada entre arestas não adjacentes, a função retorna False, indicando que o polígono não é simples.
5. Caso nenhuma interseção seja encontrada após todas as verificações, a função retorna True, confirmando que o polígono é simples.

3.3. Função para verificar se um ponto está dentro de um polígono (`point_inside_polygon`)

1. A função processa todos os polígonos classificados como simples (convexos ou côncavos) e verifica, para cada ponto, sua relação com cada polígono válido.
2. Para cada ponto, verifica se ele coincide com algum vértice do polígono, usando tolerância numérica para lidar com erros de precisão.
3. Se o ponto for um vértice, ele é imediatamente associado ao polígono.
4. Caso o ponto não seja um vértice, verifica se está sobre alguma aresta do polígono usando a função `point_on_segment`.
5. Essa função utiliza o produto vetorial para verificar colinearidade e delimita as coordenadas mínimas/máximas do segmento para evitar falsos positivos.

6. Se o ponto não está no vértice nem na aresta, aplica-se o algoritmo de ray casting para determinar se está dentro do polígono.
7. Um raio horizontal é traçado a partir do ponto. Conta-se quantas vezes esse raio cruza as arestas do polígono.
8. Se o número de cruzamentos for ímpar, o ponto está dentro do polígono; se for par, está fora.
9. A função usa tolerância (`TOL`) para evitar erros em casos críticos (ex: pontos próximos a arestas).
10. Ao final, para cada ponto, armazena-se os índices (1-base) dos polígonos onde ele está contido, seja como vértice, na aresta, ou internamente.
11. Os resultados são ordenados e impressos.