Trabalho 1 - Triangulação de polígono

Gustavo Higuchi

October 20, 2016

Contents

1 Resumo

O problema escolhido para resolver foi a *triangulação de polígonos* e a solução implementada foi dividir o polígono em sub-polígonos monotônicos e, em seguida, triangular esses subpolígonos, resultando numa triangulação do polígono original. O algoritmo foi implementado em *Python*, utilizando uma abordagem que primeiro divide o polígono em polígonos *y-monotone* para então triangular cada sub-polígono em tempo linear, o algoritmo roda em tempo $O(n \log n)$.

2 Introdução

Em geometria, uma triangulação é uma subdivisão de um objeto geométrico em *simplexos* (1). Neste trabalho é feito uma triangulação de polígonos simples de duas dimensões. Para resolver o problema foi utilizada a solução apresentada no livro *Computational Geometry Algorithms and Applications* (2). O algoritmo apresentado neste trabalho utiliza uma ordenação e por isso é limitado inferiormente a rodar em tempo $O(n \log n)$ e é descrito na secção 4.

3 Descrição do problema

O problema da triangulação de polígonos consiste em dividir um polígono com diagonais que formem um triangulo, sem que seja adicionado um novo vértice. Para um polígono de n vértices, uma triangulação desse polígono teria exatamente $n - 2$ triangulos.

Para um polígono simples e não-convexo, basta ligar um vértices com todos os outros vértices exceto seus vizinhos, então teríamos uma triangulação em tempo linear com relação à entrada. Porém, em casos que o polígono é convexo, é preciso verificar se a diagonal inserida não está fora do polígono.

4 O algoritmo

O algoritmo neste trabalho utiliza uma abordagem para dividir o polígono simples em sub-polígonos *y-monotônicos*, i.e. uma linha horizontal intersecta a fronteira da esquerda e a fronteira da direita, ou não intersecta nenhuma aresta do polígono. Isto é feito utilizando uma *sweep line* que passa do vértice mais acima até o vértice mais abaixo, portanto é necessário organizar os vértices em uma fila de prioridades pela coordenada y, no caso de mesma coordenada y, o vértice mais a direita é usado como critério de prioridade.

A estrutura de dados utilizada é a mesma que do livro da disciplina, *Doubly Connneted Edge List* ou *DCEL* e é descrita no livro de Berg et al. (2), com algumas modificações específicas da implementação que fiz e é basicamente assim:

1. Cada vértice possui apontadores para o próximo vértice e o anterior, além de suas coordenadas e angulo inicial.
2. Cada face possui um apontador para uma aresta interna
3. Cada aresta possui um apontador para o vértice de origem, o vértice de destino, o vértice helper (utilizado no algoritmo de divisão em sub-polígonos y-monotônicos), a próxima aresta e a anterior, assim como um apontador para a face que ela está e sua aresta *gêmea* que é denominada de *twin*.
4. O polígono possui uma lista de vértices, uma lista de arestas e uma lista de faces.

O algoritmo implementado é tal como aquele apresentado por Berg et al (2), com a exceção de que a triangulação de polígonos y-monotônicos é feita diferente e será descrito em seguida.

Depois de dividido, a triangulação dos sub-polígonos *y-monotônico* é trivial e foi utilizado o algoritmo *ear clipping*, que insere uma diagonal entre o início de uma aresta para o destino de outra aresta, caso o angulo formado pelas duas arestas for menor que π. O algoritmo *triangulate* descrito abaixo descreve a implementação feita.

Algoritmo 1: triangulate

Entrada: Um polígono P subdividido em polígonos monotônicos
Saída: O polígono P subdividido em triângulos

início
    $Q \leftarrow$ lista de faces de P
    para *cada elemento de Q* **faça**
        $f \leftarrow$ primeiro elemento de Q
        remove o primeiro elemento de Q
        se *f não for um triangulo* **então**
            $v_i \leftarrow$ destino da aresta inicial de f
            $v_j \leftarrow$ origem da aresta anterior de v_i
            $ear_clipping(P, v_i, v_j)$
            insere em Q a face de
        fim
    fim
fim

A função `ear_clipping()` insere uma diagonal entre dois vértices v_i e v_j, apenas uma manipulação de ponteiros da aresta anterior e posterior dos dois vértices, executando em tempo constante. A inserção dessa diagonal divide o polígono em dois sub-polígonos, criando uma face nova. Essa face aponta para a aresta *gêmea* como sendo sua aresta incial.

4.1 Complexidade e corretude

A construção da fila de prioridades é feita com uma ordenação baseada em quick sort, então executa em tempo $O(n \log n)$, e a triangulação do poligono é feita em tempo linear, levando em conta que o laço da linha 3 executa no máximo o número de triangulos que o polígono pode possuir, ou seja $n - 2$ vezes. Assim o algoritmo inteiro ficaria limitado em tempo $O(n \log n)$.

A cada iteração do laço, uma face de Q é removida, e caso a face não seja um triangulo, transforma a face em um triangulo e insere uma nova face em Q, então o algoritmo para quando houver apenas triangulos em Q.

References

[1] "https://pt.wikipedia.org/wiki/triangulação$_($*geometria*$)$."

[2] M. de Berg, O. Cheong, M. v. Kreveld, and M. Overmars, *Computational Geometry Algorithms and Applications*. Springer, third ed., 2008.