Triangulação

Caio de Freitas Valente – 6552442

Problema:

Três soluções diferentes para triangulação de polígonos:

- Remoção de orelhas
- Algoritmo para polígonos Y-monótonos
- Algoritmo de Lee-Preparata

Para que os algoritmos funcionem corretamente há duas suposições:

- Polígono de entrada é simples
- Os vértices do polígono de entrada são dados no sentido anti-horário

Há ainda a suposição de que o polígono passado para o algoritmo Y-monótono é de fato Y-monótono, ou seja, não há verificação de erros.

Arquivos:

Há um arquivo do projeto original que foi modificado:

- geocomp/__init__.py → Para inserção das funções de triangulação.

Os arquivos novos estão contidos na pasta geocomp/triangulation. Os arquivos novos são:

- __init__.py
- avl.py
- dcel.py
- stack.py
- mergesort.py
- plotpolygon.py
- brute.py
- monotone.py
- leepreparata.py

Há ainda alguns arquivos de testes na pasta Dados/teste/

Todos polígonos simples com arestas no sentido anti-horário.

Estruturas de dados:

Para a implementação do projeto foram necessárias três estruturas de dados:

- Arvore binária balanceada – Foi implementada uma AVL – avl.py
- Doubly connected edge list – dcel.py
- Pilha – stack.py

Consumo de tempo:

ABB: Inserções, buscas e remoções consomem tempo O(logn)

Pilha: Inserção e remoção consomem tempo O(1)

Douby connected edge list: Inserção de uma aresta em uma face consome tempo O(1)

Mergesort: O(nlogn)

Algoritmo força bruta usando orelhas: $O(n^2)$

Algoritmo para polígonos y-monótonos: O(n)

Algoritmo Lee-Preparata: O(nlogn)

Note que antes de cada inserção de uma nova diagonal há verificações para determinar se a origem de cada aresta está no cone da outra aresta. Essa verificação consome tempo O(1), afinal sabemos que há inserção de arestas adicionais em um ponto no máximo duas vezes, totalizando no pior dos casos três arestas com origem em um dado vértice. No pior caso já teríamos 2 arestas com origem em cada vértice, o que implicaria em no máximo 4 verificações. Logo o custo de inserção de uma diagonal é de fato O(1).

Além disso utilizamos uma estrutura auxiliar baseada nas arestas, uma lista de arestas indexada por seu ponto de origem. Usada para facilitar o acesso as arestas com mesma origem e com isso facilitar a inserção de novas diagonais. – od → orderedDCEL

Observações:

No algoritmo de remoção de orelhas usamos um vetor de Python. Vetores em Python tem um comportamento similar ao de listas, logo aproveitamos essa característica para fazermos a remoção dos vértices que já geraram diagonais.

Cores usadas na animação:

- Branco → O polígono base.
- Vermelho → Diagonais.

Remoção de orelhas:

- Amarelo → Ponto que está sendo verificado para determinar se é orelha.
- Verde → Orelha.

- Ciano → A orelha que está gerando uma diagonal e causando a verificação de orelha nos dois vértices adjacentes.

Y-Monótono:

- Amarelo → Ponto que está sendo verificado – ponto evento.
- Verde → Topo da pilha – st.
- Ciano → Primeiro elemento da pilha – s1

Lee-Preparata:

- Amarelo → Ponto que está sendo verificado – ponto evento.
- Verde → Ponto superior e arestas do trapézio acima do ponto que está sendo verificado.
- Ciano → Ponto superior e arestas do segundo trapézio acima do ponto que está sendo verificado.

Desenvolvimento:

O projeto foi desenvolvido e testado usando Python 2.7.8 – 64bits para Windows 8.1

A parte gráfica foi testada usando tkgeocomp.py