Triangulação

Caio de Freitas Valente – 6552442

Problema:

Três soluções diferentes para triangulação de polígonos:

- Usando orelhas
- Algoritmo para polígonos y-monótonos
- Algoritmo Lee-Preparata

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
- 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(f) – sendo f o número de arestas na face do polígono

Mergesort: O(nlogn)

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

Algoritmo para polígonos y-monótonos: O(n)

Algoritmo Lee-Preparata:

Seja:

- P - Numero de pontas interiores
- f - Numero de elementos em uma face

A implementação consome tempo um pouco superior a O(nlogn), Isso se deve ao fato de inserções na estrutura de dados dcel consumir tempo O(f). Essa inserção leva tempo linear pois mantemos para cada aresta o índice da face a que pertence. Como temos p pontas interiores, entao o consumo de tempo para inserção de diagonais é de O(pf).

Além disso utilizamos uma estrutura auxiliar baseada nas arestas, uma lista de arestas indexada por seu ponto de origem*, isso é usado para que inserções de arestas com um sejam feitas sempre na mesma face. – od → orderedDCEL

Essa busca 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 no em um dado vértice. Obs: Face 0, que representa a parte externa do polígono é ignorada.

Logo o consumo de tempo total é: O(nlogn + pf)

*Exemplo da estrutura od:

Seja p[i] = (20, 30)

Suponha que há duas arestas, e1 e e2, que tem como origem p[i].

e1.origin = (20,30)

e1.destiny = (42, 50)

e2.origin = (20,30)

e2.destiny = (30, 10)

Então od[i] = {e1, e2}

Cores usadas na animação:

As animações ainda não foram feitas.

Não consigo executar ggeocomp.py nem tkgeocomp.py.

Qual a versão de Python usada na correção, 2.x ou 3.x?

Quais as versões das dependências?

Qual sistema operacional?

Desenvolvimento:

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