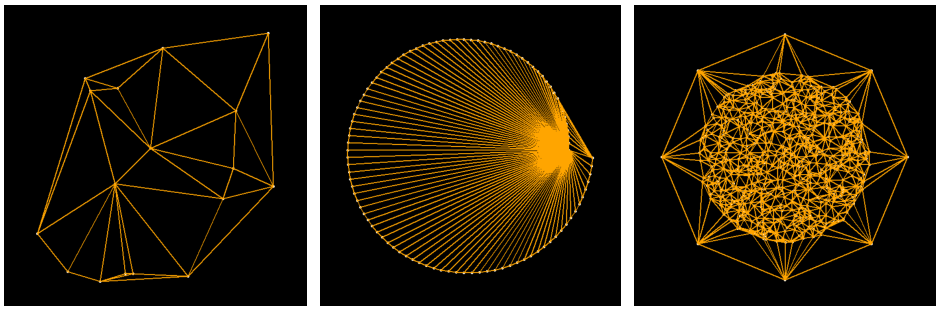
MAC0331 Geometria Computacional

Projeto 3 - Triangulação de Delauney

Pedro Gigeck Freire - 10737136



Estruturas de Dados

- **DCEL**

  A classe da DCEL foi implementada em `/geocomp/common/dcel.py` , a razão para ter implementado ela separadamente é que ela pode ser usada em outros algoritmos.

  Ela tem os atributos

  - v - Um dicionário que mapeia pontos à meia-arestas
  - f - Uma lista de meia-arestas, cada meia aresta de uma face diferente
  - extra_info - Uma lista de qualquer coisa, cada coisa associada a uma face

  E os métodos

  - init () - Inicia uma DCEL sem vértices e sem faces
  - add_vertex (v) - Insere um vértice sem arestas
  - add_edge (v1, v2) - Insere uma meia-aresta de v1-v2 e outra v2-v1
  - add_edge (v1, v2, f) - Insere uma meia-aresta de v1-v2 e outra v2-v1, conhecida a face f que eles estão
  - remove_edge (e) - Remove a meia-aresta 'e' e sua gêmea e.twin

  Os métodos init, add_vertex e remove_edge consomem tempo constante

  O método add_edge (v1, v2, f) consome tempo proporcional à quantidade de vértices da face f

  O método add_edge (v1, v2) consome tempo proporcional à (quantidade de vértices da face f + grau(v1) + grau(v2))

- **DAG**

  O DAG de triângulos foi implementado no mesmo arquivo do algoritmo `/geocomp/delauney/incremental.py` e é uma classe Node_Triang que tem os atributos:

  - p1, p2, p3 - Pontos em sentido anti-horário
  - a - Meia aresta da DCEL que faz parte da face p1-p2-p3
  - filhos - Uma lista de Node_Triangs que será o que ligará o DAG

  E os métodos

    - init (p1, p2, p3, a) - Cria um novo triângulo
    - busca (p) - Retorna o nó folha que contém o ponto p

- **OBS: A Salada de Ponteiros**

  Inserir e remover arestas da DCEL é um trabalho bastante chato. Tem que encontrar quem são arestas vizinhas, acertar os ponteiros de prox e prev de ambas, conferir se não criou ou extinguiu alguma face, criar novas faces quando a aresta forma um ciclo e repetir tudo para a meia-aresta gêmea.

  Além disso, como comentado no fórum, o DAG e a DCEL precisam estar amarrados. Então sempre que criamos um triângulo, nós criamos o Node_Triang `t` e adicionamos três arestas `e1`, `e2`, e `e3` na DCEL.

  Isso formará nova uma face que ficará no final do vetor de faces, então fazemos um `dcel.extra_info.append(t)`, depois nós fazemos `t.a = e1`. Dessa forma, as duas estruturas ficam em sincronia.

  Porém, os números das faces podem mudar. Quando removemos uma aresta ilegal, nós unimos duas faces, o que gera um pouco mais de confusão dos ponteiros.

  Encontrar ponteiro perdidos foi a minha diversão do fim de semana.

O Algoritmo

A função principal do algoritmo é a `Incremental (pontos)` que está em `/geocomp/delauney/incremental.py`.

O primeiro passo foi criar um primeiro triângulo fictício para ser a raiz do DAG.
Isso é feito pela função `pontos_infinitos(p)` que retorna três pontos bem distantes.

Para o processamento principal, para cada ponto `p` fazemos:

- Busca o triângulo `t` no DAG tal que `p` pertence a `t`
- Insere uma aresta de `p` para cada ponto de `t` na DCEL
- Insere os três novos triângulos no DAG
- Verifica por arestas ilegais nesses novos triângulos que vão sendo formados

No final, temos que lembrar de excluir os pontos e arestas envolvendo o triângulo fictício.

Devolvemos a DCEL.

Casos Degenerados

Conforme escrevi no fórum, praticamente não tratei os casos degenerados

Os pontos repetidos eu apenas ignoro (exemplo `/LOOSE_PTS/#a`).
Os pontos que caem sobre as arestas acabam funcionando normalmente, a aresta que ficou em cima das outras acaba ficando ilegal e é substituída. (exemplo `LOOSE_PTS/ptos5.pts` ou `LOOSE_PTS/incr`).

Complexidade do Algoritmo:

O primeiro e último passos analisamos em aula (slides aula 15). O número esperado de arestas legalizadas é constante e a soma esperada dos percursos no DAG é $\mathrm{O}(nlgn)$

No segundo passo, `t` guarda uma aresta `t.a` da DCEL e `t.a` conhece sua face, então inserir uma nova aresta consome tempo proporcional ao tamanho das faces, que são triângulos. Então consome tempo constante.

No terceiro passo, criar os novos nós é $O(1)$ e associar eles ao DAG também, pois a DCEL guarda quais triângulos que receberão os novos como filhos.

A Animação

Como comentado em aula, o algoritmo pode ficar um pouco confuso se mostrarmos as buscas no DAG, então segui a sugestão da professora eu fiz duas versões, uma que desenha a busca no DAG, que está em `/geocomp/delauney/incremental_dag.py` e a outra que não desenha a busca no DAG. A diferença entre elas é apenas uma flag `desenha_busca`.

Cor	Significado
Vermelho *firebrick*	Novo ponto e novas arestas que foram incluídos nessa iteração
Verde	Arestas que estão sendo testadas (para ver a ilegalidade)
Vermelho	Arestas ilegais
Laranja	Arestas da triangulação, adicionadas nas iterações anteriores
Cinza	Mostra os nós do DAG na busca pelo ponto (apenas na versão `incremental_dag`)

OBS:

Durante a quarentena eu implementei alguns outros algoritmos:

- Lee & Preparata para triangulação de polígonos (esse é legal porque usa a DCEL)
- Linha de varredura para par de pontos mais próximos
- Bentley & Ottmann para interseção de segmentos

O primeiro ainda não está com a complexidade correta e o último falha para casos degenerados, fica para uma próxima oportunidade consertá-los.