Algoritmo para gerar invólucros convexos

Artur Queiroz - PG38014
Luís Albuquerque - PG38015

December 4, 2019

1 Introdução

Neste trabalho abordamos um dos principais temas de Geometria Computacional, Invólucros Convexos. Estes são muito usados porque têm aplicações em bastantes àreas como reconhecimento de padrões, processamento de imagem, estatística, etc.

Existem várias formas de construir um, mas neste trabalho vamos nos cingir a implementar o "*merge-hull*" Que é um algoritmo que se usa, umas das tecnicas mais importantes na computação, que se chama "Dividir para conquistar". Que se baseia em dividir um problema complexo, em problemas mais pequenos e mais faceis.

2 Descrição

Para implementarmos o algoritmo apenas temos que seguir os passos a baixo.

- Ordenar os pontos por ordem lexicográfica.
- Remover os pontos repetidos.
- Criar o Invólucro convexo
  - Se o conjunto tem apenas 1 elemento, criar um "polígono" de 1 elemento e acaba-se a função.
  - Separar os pontos em dois conjuntos A e B, onde A contem os pontos da esquerda e B os da direita.
  - Calcular o invólucro convexo de A, $\mathcal{A} = \mathrm{I(A)}$ e o de B, $\mathcal{B} = \mathrm{I(B)}$ recursivamente

- Encontrar o pontos de junção.
- Juntar $\mathcal{A}$ e $\mathcal{B}$, calculando o invólucro convexo de A $\cup$ B.

Agora vamos explicar com mais detalhe todo o processo.

2.1 Ordenar os Pontos

Para ordenar os pontos, pode ser usado qualquer algoritmo de ordenação, tendo em atenção que a escolha do algoritmo de ordenação, pode alterar a complexidade do algoritmo como um todo. Nós optamos por escolher o algoritmo de ordenação *merge sort*, além de ter uma das melhores complexidades $\Theta(n \log n)$, achamos que se enquadra perfeitamente no espirito do algoritmo, "Dividir para conquistar".

2.1.1 Descrição de MergeSort

Input: array, indice esquerdo, indice direito

Começando com o indice esquerdo a 0, e o indice direito a (*tamanho do array*) - 1

- Primeiro encontra-se o indice médio do Array e divide-se em dois. (meio = (esquerda + direita)/2)
- Calcular o MergeSort(array, esquerda, meio) (a lista que fica à esquerda)
- Calcular o MergeSort(array, meio+1, direita) (a lista que fica à direita)
- No final junta os dois de forma ordenada.

2.2 Remover os Pontos Repetidos

Já que neste contexto sabemos que o array está ordenado, podemos fazer este algoritmo com 2 indices, um para percorrer o array, o outro para percorrer o novo array onde todos os elementos são diferentes. Se o valor dos dois indices for diferente então incrementa-se o segundo indice e atribui-se o valor do primeiro indice no valor do segundo indice. Se o valor dos dois indices for igual, apenas incrementa-se o primeiro indice.

2.3 Separar os Pontos em dois conjuntos

No trabalho decidimos usar um array, para que a sua divisão a meio tivesse uma complexidade constante é tão simples como encontrar o índice a meio do array.

2.4 Juntar os invólucros convexos

Liberta-se a memória dos vertices entre as duas ligações e de seguida conecta-se as duas ligações.

3 Correção

Depois de mostrarmos como é o algoritmo, aqui vamos provar, porque é que o algoritmo faz o que diz que faz. Tendo I como a nossa função descrita em cima, Queremos que:

$$< \forall S : S \in \mathcal{P}(Point) :< \forall i, j : i \in S \wedge j \in S : line(i, j) \subseteq I(S) >>$$

Caso base ($S = \emptyset$):
Como não há elementos é trivialmente verdade.
Caso base ($\#S = 1$):
Seja S = {e},
Temos de mostrar que $\{e\} \subseteq I(\{e\})$
$e \in I(\{e\})$
$\equiv$\{Descrição do algoritmo\}
$e \in Poligon([e])$
$\equiv$\{Caso base de Poligon\}
$e \in \{e\}$ (Trivial)
Caso indutivo:
Para $S = S_1 \cup S_2$
Por hipótese sabemos que:

1. $< \forall i, j : i \in S_1 \wedge j \in S_1 : line(i, j) \subseteq I(S_1) >$
2. $< \forall i, j : i \in S_2 \wedge j \in S_2 : line(i, j) \subseteq I(S_2) >$

Queremos que:
$< \forall i, j : i \in S \wedge j \in S : line(i, j) \subseteq I(S) >$

$\equiv\{...\}$
$< \forall i, j : i \in S \wedge j \in S : line(i, j) \subseteq join(I(S_1), I(S_2)) >$
$\equiv\{...\}$
$< \forall i, j : i \in S \wedge j \in S : line(i, j) \subseteq (I(S_1) \cup I(S_2)) \backslash freeJoin(S_1, S_2) >$
$\equiv\{...\}$
$< \forall i, j : i \in S \wedge j \in S : line(i, j) \subseteq I(S_1) \backslash freeJoin(S_1, S_2) \cup I(S_2) \backslash freeJoin(S_1, S_2) >$
$\equiv\{...\}$

■

3.1 Proposições

1. $< \forall S', S :: S' \subseteq S \Rightarrow I(S') \subseteq I(S) >$
2. As linhas de junção estão bem feitas
   (a) A de baixo $< \forall p \in S :: leftOn(l, p) >$
   (b) A de cima $< \forall p \in S :: rightOn(l, p) >$
3. Sejam as linhas de junção $l_1 = \overline{p_1, q_1}$ e $l_2 = \overline{p_2, q_2}$ Vamos definir $l'_1 = \overline{p_1, p_2}$ e $l'_2 = \overline{q_1, q_2}$ vamos querer que:
   $< \forall l : l \in Line : l \cap l'_1 \neq \emptyset \wedge l \cap l'_2 \neq \emptyset >$

4 Complexidade

A nossa implementação não foi exatamente igual ao algoritmo original, apesar de não alterar na conta da complexidade assintoticamente. Por isso vamos avaliar a correção da nossa implementação, e quando achamos pertinente, vamos fazer a ressalva, mensionando as diferenças em relação ao algoritmo original.

- Ordenar os pontos pela cordenada x, tem Complexidade $\Theta(n \log n)$
- Separar os pontos em dois conjuntos A e B, onde A contém os pontos da esquerda e B os da direita.
- Calcular o invólucro convexo de A, $\mathcal{A}$ = I(A) e o de B, $\mathcal{B}$ = I(B) recursivamente
- No final juntar $\mathcal{A}$ e $\mathcal{B}$, calculando o invólucro convexo de A $\cup$ B.

5 Conclusão