Geometria Computacional

Cristina G. Fernandes

Departamento de Ciência da Computação do IME-USP

`http://www.ime.usp.br/~cris/`

segundo semestre de 2018

Polígonos monótonos

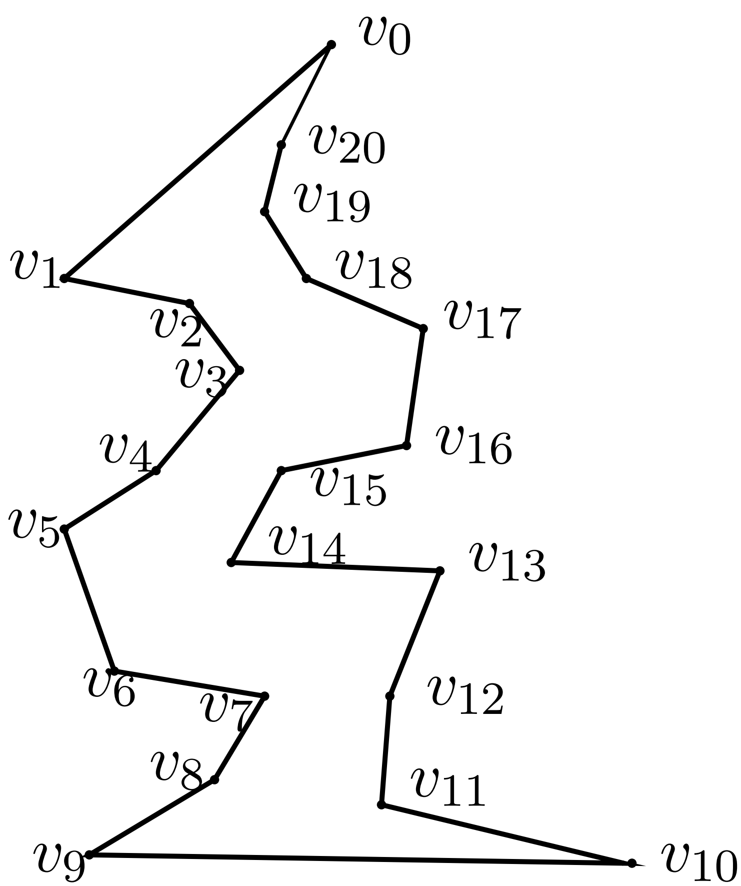
Um polígono P é monótono em relação a uma reta L se $P \cap L'$ é conexo para toda reta L' perpendicular a L.

Se L é o eixo y, dizemos que P é y-monótono.

Polígonos monótonos

Um polígono P é monótono em relação a uma reta L se $P \cap L'$ é conexo para toda reta L' perpendicular a L.

Se L é o eixo y, dizemos que P é y-monótono.



Polígonos monótonos

Seja P um polígono y-monótono com n vértices.

Podemos ordenar os vértices de P por y-coordenada em tempo $\mathrm{O}(n)$.

Polígonos monótonos

Seja P um polígono y-monótono com n vértices.

Podemos ordenar os vértices de P por y-coordenada em tempo $\mathrm{O}(n)$.

δP: fronteira de P

- determine a curva poligonal esquerda de δP
- determine a curva poligonal direita de δP
- intercale as duas curvas

Polígonos monótonos

Seja P um polígono y-monótono com n vértices.

Podemos ordenar os vértices de P por y-coordenada em tempo $\mathrm{O}(n)$.

δP: fronteira de P

- determine a curva poligonal esquerda de δP
- determine a curva poligonal direita de δP
- intercale as duas curvas

Cada um destes passos pode ser feito em tempo $\mathrm{O}(n)$.

Algoritmo

Entrada: polígono monótono P com n vértices
Saída: triangulação de P

Algoritmo

Entrada: polígono monótono P com n vértices
Saída: triangulação de P

Primeiro passo: ordenar os vértices de P por y-coordenada, obtendo $u_1, \ldots, u_n$

Restante: é iterativo e usa uma pilha

Algoritmo

Entrada: polígono monótono P com n vértices
Saída: triangulação de P

Primeiro passo: ordenar os vértices de P por y-coordenada, obtendo $u_1, \ldots, u_n$

Restante: é iterativo e usa uma pilha

O algoritmo produz uma seqüência de polígonos

$$P = P_0, P_1, \ldots, P_n = \emptyset$$

onde o polígono

P_i é obtido de P_{i-1} após o algoritmo processar u_i

Invariantes do algoritmo

Entrada: polígono monótono P com n vértices
Saída: triangulação de P

Primeiro passo: ordenar os vértices de P por y-coordenada, obtendo $u_1, \ldots, u_n$

Restante: é iterativo e usa uma pilha $S = (s_1, \ldots, s_t)$

Invariantes do algoritmo

Entrada: polígono monótono P com n vértices
Saída: triangulação de P

Primeiro passo: ordenar os vértices de P por y-coordenada, obtendo $u_1, \ldots, u_n$

Restante: é iterativo e usa uma pilha $S = (s_1, \ldots, s_t)$

No início de cada iteração, valem os seguintes invariantes:

- $s_1, \ldots, s_t$ em ordem crescente de y-coordenada e incluem todos os vértices abaixo de s_1 e acima de s_t
- $s_1, \ldots, s_t$ são vértices consecutivos na cadeia esquerda ou direita de P_{i-1}
- $s_2, \ldots, s_{t-1}$ são vértices reflexos de P_{i-1}
- P_i é o que falta triangular de P

Invariantes do algoritmo

Entrada: polígono monótono P com n vértices
Saída: triangulação de P

Primeiro passo: ordenar os vértices de P por y-coordenada, obtendo $u_1, \ldots, u_n$

Restante: é iterativo e usa uma pilha $S = (s_1, \ldots, s_t)$

No início de cada iteração, valem os seguintes invariantes:

- $s_1, \ldots, s_t$ em ordem crescente de y-coordenada e incluem todos os vértices abaixo de s_1 e acima de s_t
- $s_1, \ldots, s_t$ são vértices consecutivos na cadeia esquerda ou direita de P_{i-1}
- $s_2, \ldots, s_{t-1}$ são vértices reflexos de P_{i-1}
- P_i é o que falta triangular de P

Cadeia reflexa corrente: $s_1, \ldots, s_t$

Casos do algoritmo

Seja u_i o vértice processado nessa iteração.

Casos do algoritmo

Seja u_i o vértice processado nessa iteração.

Três casos:

(a) u_i é adjacente (em δP) a s_t mas não a s_1

(b) u_i é adjacente a s_1 mas não a s_t

(c) u_i é adjacente a s_1 e a s_t

Casos do algoritmo

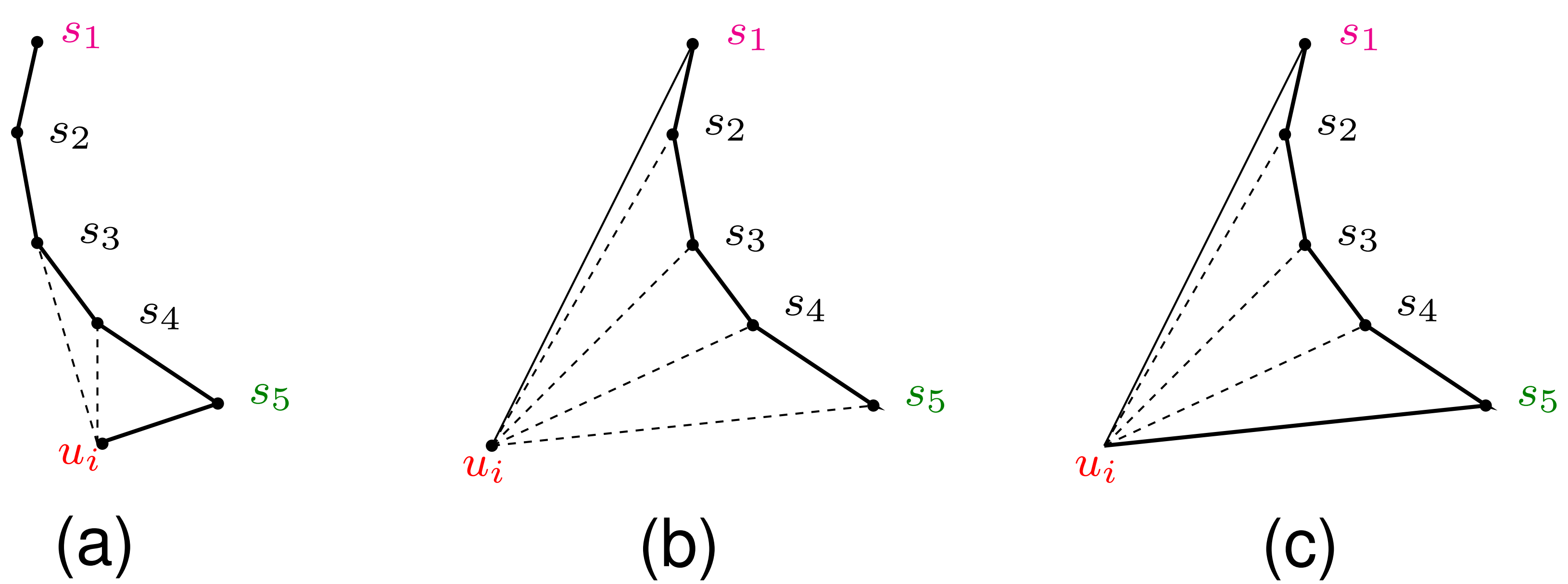
Seja u_i o vértice processado nessa iteração.

Três casos:

(a) u_i é adjacente (em δP) a s_t mas não a s_1

(b) u_i é adjacente a s_1 mas não a s_t

(c) u_i é adjacente a s_1 e a s_t



Triangula monótono

DIVIDEEMMONÓTONO-LP(n, P)

$u_1, \dots, u_n \leftarrow$ Ordena(n, P)
$S \leftarrow (u_1, u_2)$ $\quad D \leftarrow \emptyset$

para $i \leftarrow 3$ **até** n **faça**
  sejam $s_1, \dots, s_t$ os vértices de S

  Caso (a): u_i adjacente a s_t mas não a s_1

  Caso (b): u_i adjacente a s_1 mas não a s_t

  Caso (c): u_i adjacente a s_1 e a s_t $\quad \triangleright\ u_i = u_n$

devolva D

Triangula monótono

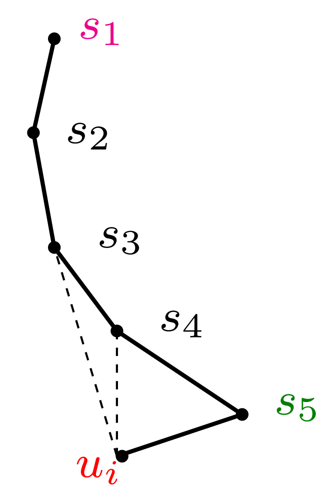
DIVIDEEMMONÓTONO-LP(n, P)

Caso (a): u_i adjacente a s_t mas não a s_1
  enquanto $t > 1$ **e** Ângulo$(u_i, s_t, s_{t-1}) < \pi$ **faça**
    Desempilha(S)
    $t \leftarrow t - 1$
    $D \leftarrow D \cup \{u_i s_t\}$
  Empilha(S, u_i)

Triangula monótono

DIVIDEEMMONÓTONO-LP(n, P)

Caso (a): u_i adjacente a s_t mas não a s_1
    enquanto $t > 1$ **e** Ângulo$(u_i, s_t, s_{t-1}) < \pi$ **faça**
        Desempilha(S)
        $t \leftarrow t - 1$
        $D \leftarrow D \cup \{u_i s_t\}$
    Empilha(S, u_i)



(a)

Triangula monótono

DIVIDEEMMONÓTONO-LP(n, P)

Caso (b): u_i adjacente a s_1 mas não a s_t

  $aux \leftarrow s_t$

  enquanto $t > 1$ **faça**

    $D \leftarrow D \cup \{u_i s_t\}$

    Desempilha(S)

    $t \leftarrow t - 1$

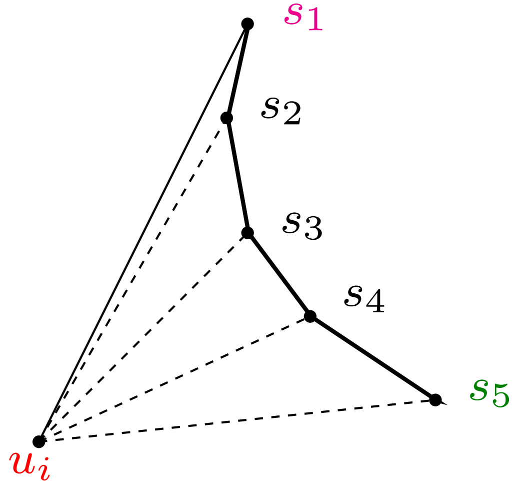
  Desempilha(S) ▷ desempilha s_1

  Empilha(S, aux) Empilha(S, u_i)

Triangula monótono

DIVIDEEMMONÓTONO-LP(n, P)

Caso (b): u_i adjacente a s_1 mas não a s_t
    $aux \leftarrow s_t$
    enquanto $t > 1$ **faça**
        $D \leftarrow D \cup \{u_i s_t\}$
        Desempilha(S)
        $t \leftarrow t - 1$
    Desempilha(S) ▷ desempilha s_1
    Empilha(S, aux) Empilha(S, u_i)



(b)

Triangula monótono

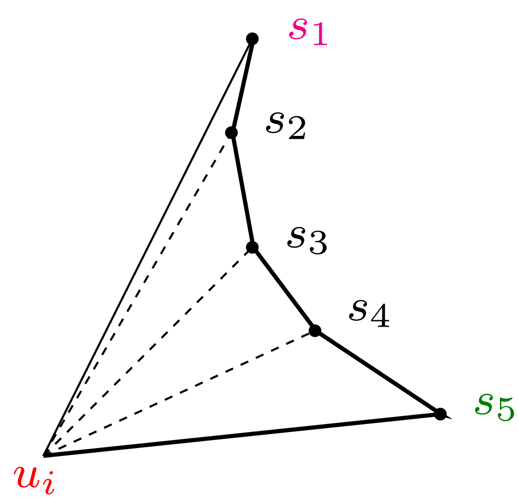
DIVIDEEMMONÓTONO-LP(n, P)

Caso (c): u_i adjacente a s_1 e a s_t $\triangleright$ $u_i = u_n$
  Desempilha(S) $\triangleright$ desempilha s_t
  enquanto $t > 2$ **faça**
    $t \leftarrow t - 1$
    $D \leftarrow D \cup \{u_i s_t\}$
    Desempilha(S)

Triangula monótono

DIVIDEEMMONÓTONO-LP(n, P)

Caso (c): u_i adjacente a s_1 e a s_t ▷ $u_i = u_n$
    Desempilha(S) ▷ desempilha s_t
    enquanto $t > 2$ **faça**
        $t \leftarrow t - 1$
        $D \leftarrow D \cup \{u_i s_t\}$
        Desempilha(S)



(c)

Triangula monótono em tempo linear

para $i \leftarrow 3$ **até** n **faça**
  sejam $s_1, \ldots, s_t$ os vértices de S
  Caso (a): u_i adjacente a s_t mas não a s_1
    enquanto $t > 1$ **e** Ângulo$(u_i, s_t, s_{t-1}) < \pi$ **faça**
      Desempilha(S); $t \leftarrow t - 1$; $D \leftarrow D \cup \{u_i s_{t-1}\}$
    Empilha(S, u_i)
  Caso (b): u_i adjacente a s_1 mas não a s_t
    $aux \leftarrow s_t$
    enquanto $t > 1$ **faça**
      $D \leftarrow D \cup \{u_i s_t\}$;   Desempilha$(S)$; $t \leftarrow t - 1$
    Desempilha(S) ▷ desempilha s_1
    Empilha(S, aux) Empilha(S, u_i)
  Caso (c): u_i adjacente a s_1 e a s_t ▷ $u_i = u_n$
    Desempilha(S) ▷ desempilha s_t
    enquanto $t > 2$ **faça**
      $t \leftarrow t - 1$; $D \leftarrow D \cup \{u_i s_t\}$;   Desempilha$(S)$

Triangula monótono em tempo linear

O número de chamadas de Empilha é não mais que $2n$.

Triangula monótono em tempo linear

O número de chamadas de Empilha é não mais que $2n$.

O número de chamadas de Desempilha portanto também é no máximo $2n$.

Triangula monótono em tempo linear

O número de chamadas de Empilha é não mais que $2n$.

O número de chamadas de Desempilha portanto também é no máximo $2n$.

O consumo de tempo do algoritmo é proporcional ao número de chamadas de Empilha mais o número de chamadas de Desempilha.

Triangula monótono em tempo linear

O número de chamadas de Empilha é não mais que $2n$.

O número de chamadas de Desempilha portanto também é no máximo $2n$.

O consumo de tempo do algoritmo é proporcional ao número de chamadas de Empilha mais o número de chamadas de Desempilha.

Portanto o consumo de tempo é $O(n)$.

Triangulação em $\mathrm{O}(n \lg n)$

P: polígono arbitrário com n vértices

Idéia do algoritmo:

Triangulação em $\mathrm{O}(n \lg n)$

P: polígono arbitrário com n vértices

Idéia do algoritmo:

- particionar P em polígonos monótonos
- triangular cada um deles em tempo linear

Triangulação em $\mathrm{O}(n \lg n)$

P: polígono arbitrário com n vértices

Idéia do algoritmo:

- particionar P em polígonos monótonos
- triangular cada um deles em tempo linear

Partição tem que consumir tempo $\mathrm{O}(n \lg n)$!

Triangulação em $\mathrm{O}(n \lg n)$

P: polígono arbitrário com n vértices

Idéia do algoritmo:

- particionar P em polígonos monótonos
- triangular cada um deles em tempo linear

Partição tem que consumir tempo $\mathrm{O}(n \lg n)$!

Como fazemos isso?

Triangulação em $\mathrm{O}(n \lg n)$

P: polígono arbitrário com n vértices

Idéia do algoritmo:

- particionar P em polígonos monótonos
- triangular cada um deles em tempo linear

Partição tem que consumir tempo $\mathrm{O}(n \lg n)$!

Como fazemos isso?

Usando uma trapezoidação especial de P.

Trapezoidação

Trapézio: quadrilátero com duas arestas paralelas

Trapezoidação

Trapézio: quadrilátero com duas arestas paralelas

Trapezoidação horizontal de um polígono P: resultado de traçar segmentos horizontais maximais contidos em P, passando por cada vértice de P.

Trapezoidação

Trapézio: quadrilátero com duas arestas paralelas

Trapezoidação horizontal de um polígono P: resultado de traçar segmentos horizontais maximais contidos em P, passando por cada vértice de P.