Geometria Computacional

Aula 6

Triangulação de polígonos monótonos

Sec 3.3 do livro de de Berg e outros

Polígonos monótonos

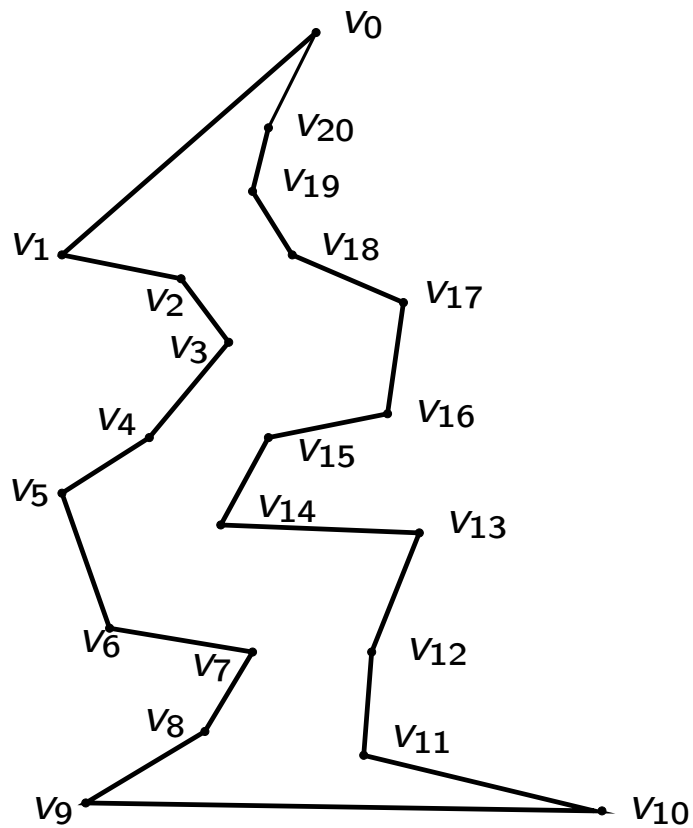
Um polígono P é monótono em relação a uma reta L se $P \cap L'$ é conexo para toda reta L' perpendicular a L.

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 sequência 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:

- P_i é o que falta triangular de P
- $s_1, \ldots, s_t$ está em ordem decrescente de y-coordenada e inclue todos os vértices abaixo de s_1 e acima de 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:

- P_i é o que falta triangular de P
- $s_1, \ldots, s_t$ está em ordem decrescente de y-coordenada e inclue todos os vértices abaixo de s_1 e acima de s_t
- $s_1, \ldots, s_t$ são vértices consecutivos na curva poligonal esquerda ou direita de P_{i-1}

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:

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

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:

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

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:

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

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:

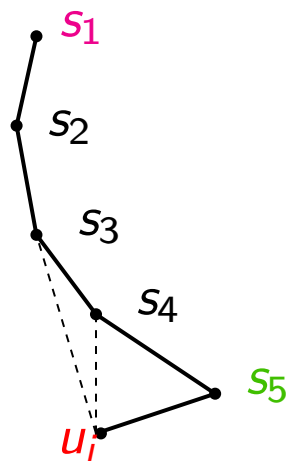
(a) u_i é adjacente (em δP) a s_t mas não a s_1

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



(a)

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

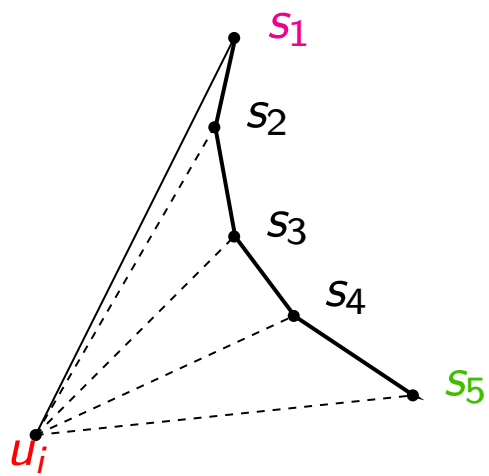
Casos do algoritmo

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



(b)

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

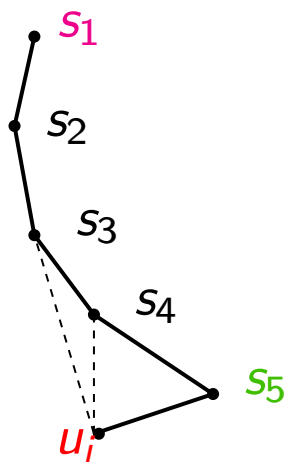
Seja u_i o vértice processado nessa iteração.

Três casos:

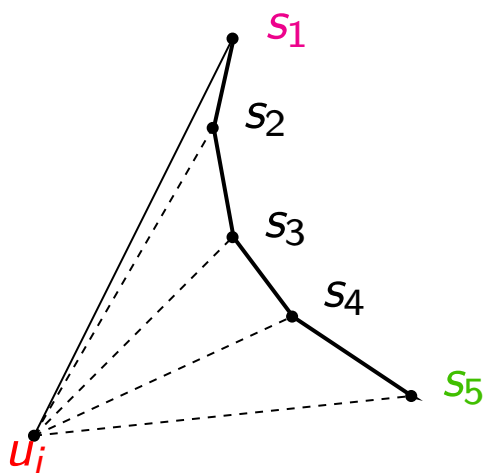
(a) u_i é adjacente (em δP) a s_t mas não a s_1

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

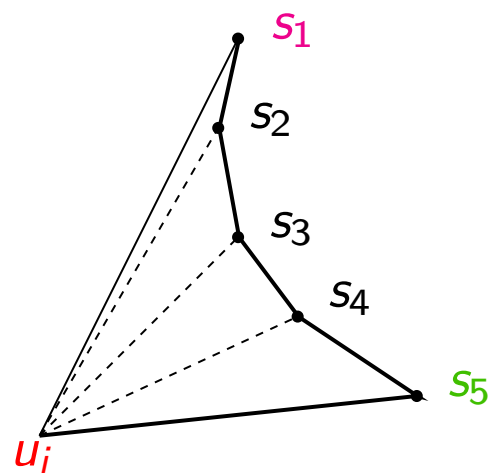
(c) u_i é adjacente a s_1 e a s_t



(a)



(b)



(c)

Triangula monótono

DivideEmMonótono-LP(n, P)

1 $u_1, \ldots, u_n \leftarrow$ Ordena(n, P)
2 $S \leftarrow (u_1, u_2)$ $t \leftarrow 2$ $D \leftarrow \emptyset$

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

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

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

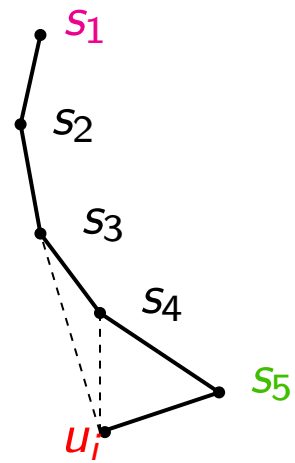
17 **Caso (c):** u_i adjacente a s_1 e a s_t $\triangleright$ $u_i = u_n$

21 **devolva** D

Triangula monótono

DivideEmMonótono-LP(n, P)

5 **Caso (a):** u_i adjacente a s_t mas não a s_1
6 **enquanto** $t > 1$ **e** Ângulo$(u_i, s_t, s_{t-1}) < \pi$ **faça**
7 $t \leftarrow t - 1$ ▷ Desempilha(S)
8 $D \leftarrow D \cup \{u_i s_t\}$
9 $t \leftarrow t + 1 \quad s_t \leftarrow u_i$ ▷ Empilha(S, u_i)

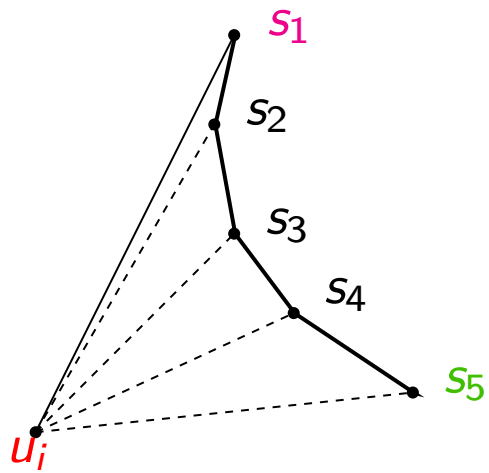


(a)

Triangula monótono

DivideEmMonótono-LP(n, P)

10 **Caso (b):** u_i adjacente a s_1 mas não a s_t
11 $aux \leftarrow s_t$
12 **enquanto** $t > 1$ **faça**
13 $D \leftarrow D \cup \{u_i s_t\}$
14 $t \leftarrow t - 1$ ▷ Desempilha(S)
15 $s_t \leftarrow aux$ ▷ desempilha s_1 e empilha aux
16 $t \leftarrow t + 1$ $s_t \leftarrow u_i$ Empilha(S, u_i)



(b)

Triangula monótono

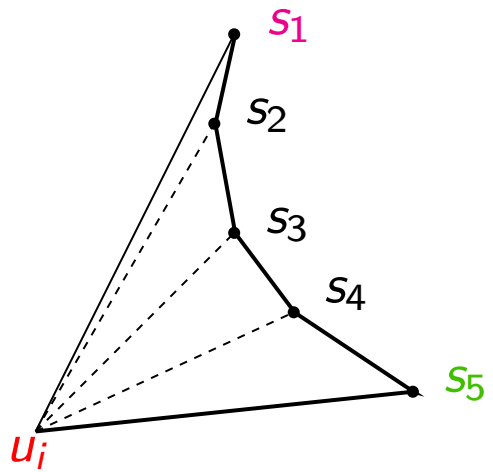
DivideEmMonótono-LP(n, P)

17 **Caso (c):** u_i adjacente a s_1 e a s_t ▷ $u_i = u_n$

18 **enquanto** $t > 2$ **faça**

19 $t \leftarrow t - 1$ ▷ Desempilha(S)

20 $D \leftarrow D \cup \{u_i s_t\}$



(c)

Triangula monótono em tempo linear

```
 3  para i ← 3 até n faça
 4     sejam s_1, ..., s_t os vértices de S
 5     Caso (a): u_i adjacente a s_t mas não a s_1
 6        enquanto t > 1 e Ângulo(u_i, s_t, s_{t-1}) < π faça
 7           t ← t − 1;    D ← D ∪ {u_i s_{t-1}}
 8        t ← t + 1    s_t ← u_i
 9     Caso (b): u_i adjacente a s_1 mas não a s_t
10        aux ← s_t
11        enquanto t > 1 faça
12           D ← D ∪ {u_i s_t};    t ← t − 1
13        s_1 ← aux;    s_2 ← u_i;    t ← 2
14     Caso (c): u_i adjacente a s_1 e a s_t                ▷ u_i = u_n
15        enquanto t > 2 faça
16           t ← t − 1;    D ← D ∪ {u_i s_t}
```

Triangula monótono em tempo linear

O número de chamadas de Empilha (incrementos de t)
é não mais que $2n$.

Triangula monótono em tempo linear

O número de chamadas de Empilha (incrementos de t)
é não mais que $2n$.

O número de chamadas de Desempilha (decrementos de t)
portanto também é no máximo $2n$.

Triangula monótono em tempo linear

O número de chamadas de Empilha (incrementos de t)
é não mais que $2n$.

O número de chamadas de Desempilha (decrementos de t)
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 (incrementos de t)
é não mais que $2n$.

O número de chamadas de Desempilha (decrementos de t)
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 $O(n \lg n)$

P: polígono arbitrário com n vértices

Idéia do algoritmo:

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

Triangulação em $O(n \lg n)$

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 $O(n \lg n)$!

Triangulação em $O(n \lg n)$

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 $O(n \lg n)$!

Como fazemos isso?

Triangulação em $O(n \lg n)$

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 $O(n \lg n)$!

Como fazemos isso?

Usando uma trapezoidação especial de P.

Trapezoidação

Trapézio: quadrilátero com duas arestas paralelas

Trapezoidação

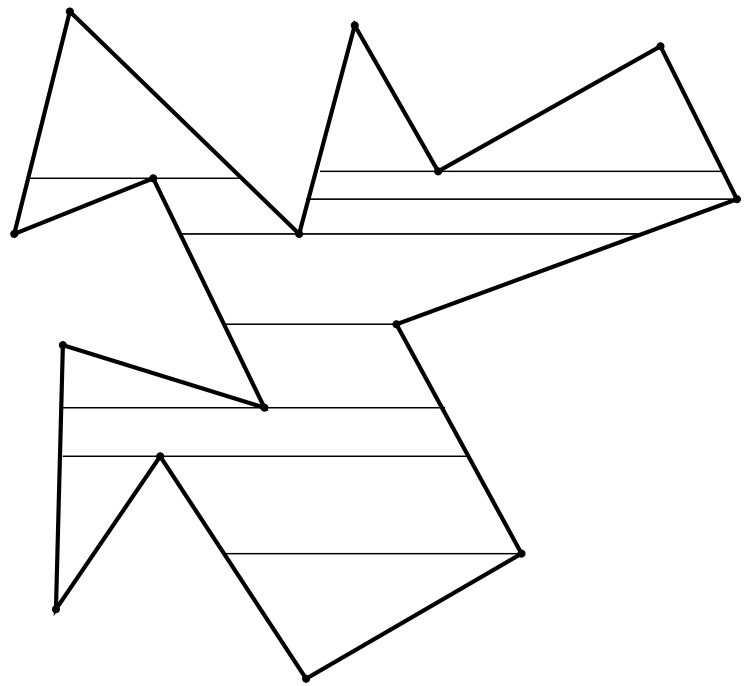
Trapézio: quadrilátero com duas arestas paralelas

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

Trapezoidação

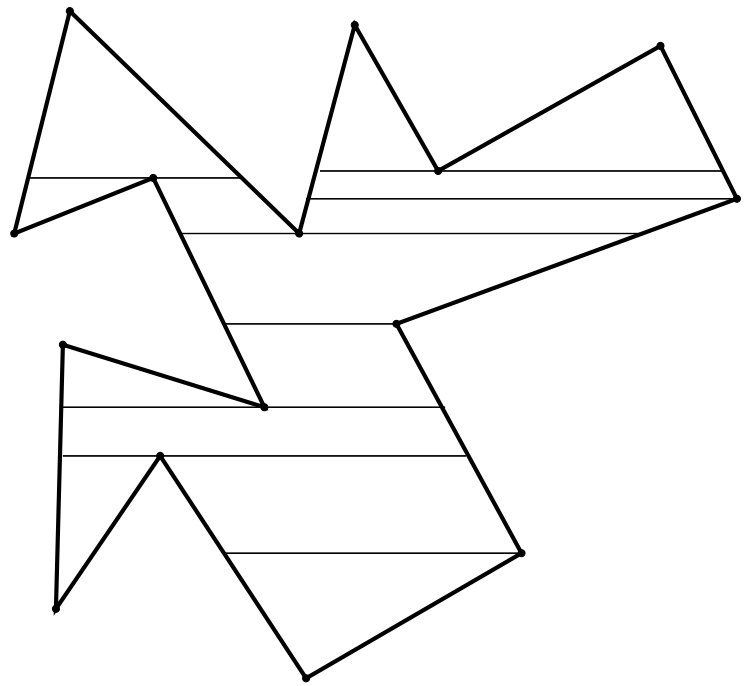
Trapézio: quadrilátero com duas arestas paralelas

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



Trapezoidação

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



Usaremos linha de varredura para calcular uma trapezoidação de P.