

Polígonos monótonos

Seja P um polígono y -monótono com n vértices.

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

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 $O(n)$.

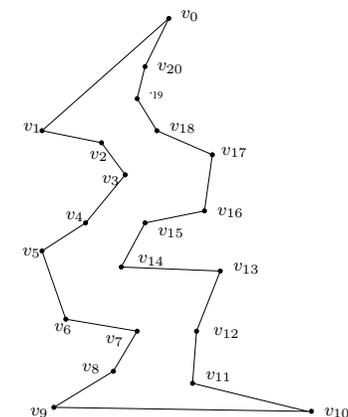
δP : fronteira de P

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

Polígonos monótonos

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**.



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, \dots, u_n

Restante: é iterativo e usa uma pilha

Polígonos monótonos

Seja P um polígono y -monótono com n vértices.

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

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 $O(n)$.

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, \dots, u_n

Restante: é iterativo e usa uma pilha

O algoritmo produz uma seqüência de polígonos

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

onde o polígono

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

Algoritmo

Entrada: polígono monótono P com n vértices

Saída: triangulação 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, \dots, u_n

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

No início de cada iteração, valem os seguintes invariantes:

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

Cadeia reflexa corrente: s_1, \dots, s_t

Geometria Computacional – p.4/11

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, \dots, u_n

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

No início de cada iteração, valem os seguintes invariantes:

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

Geometria Computacional – p.4/11

Casos do algoritmo

Seja u_i o vértice processado nessa iteração.

Geometria Computacional – p.5/11

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, \dots, u_n

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

No início de cada iteração, valem os seguintes invariantes:

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

Geometria Computacional – p.5/11

Triangula monótono

Monótono(n, P)

- 1 $u_1, \dots, u_n \leftarrow \text{Ordena}(n, P)$
- 2 $S \leftarrow (u_1, u_2) \quad D \leftarrow \emptyset$
- 3 **para** $i \leftarrow 3$ **até** n **faça**
- 4 sejam s_1, \dots, s_t os vértices de S
- 5 **Caso (a):** u_i adjacente a s_t mas não a s_1
- 11 **Caso (b):** u_i adjacente a s_1 mas não a s_t
- 19 **Caso (c):** u_i adjacente a s_1 e a $s_t \quad \triangleright u_i = u_n$
- 25 **devolva** D

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

Monótono(n, P)

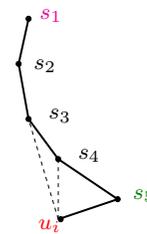
- 5 **Caso (a):** u_i adjacente a s_t mas não a s_1
- 6 **enquanto** $t > 1$ **e** $\hat{\text{Angulo}}(u_i, s_t, s_{t-1}) < \pi$ **faça**
- 7 **Desempilha**(S)
- 8 $t \leftarrow t - 1$
- 9 $D \leftarrow D \cup \{u_i s_t\}$
- 10 **Empilha**(S, u_i)

Casos do algoritmo

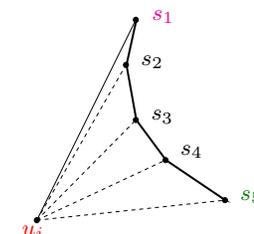
Seja u_i o vértice processado nessa iteração.

Três casos:

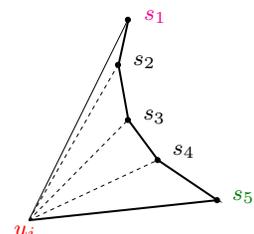
- (a) u_i é adjacente (em δP) a s_t mas não a s_1
- (b) u_i é adjacente a s_1 mas não a s_t
- (c) u_i é adjacente a s_1 e a s_t



(a)



(b)

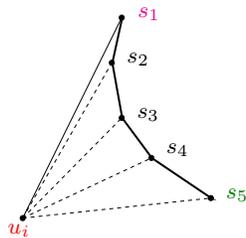


(c)

Triangula monótono

Monótono(n, P)

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

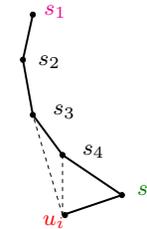


(b)

Triangula monótono

Monótono(n, P)

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



(a)

Triangula monótono

Monótono(n, P)

- 19 **Caso (c):** u_i adjacente a s_1 e a s_t ▷ $u_i = u_n$
- 20 **Desempilha**(S) ▷ desempilha s_t
- 21 **enquanto** $t > 2$ **faça**
- 22 $t \leftarrow t - 1$
- 23 $D \leftarrow D \cup \{u_i s_t\}$
- 24 **Desempilha**(S)

Triangula monótono

Monótono(n, P)

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

Triangula monótono em tempo linear

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

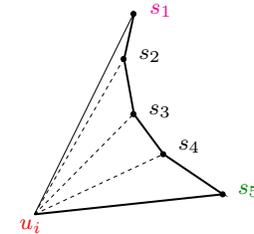
Triangula monótono

Monótono(n, P)

```

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

```



(c)

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

```

3  para  $i \leftarrow 3$  até  $n$  faça
4    sejam  $s_1, \dots, s_t$  os vértices de  $S$ 
5    Caso (a):  $u_i$  adjacente a  $s_t$  mas não a  $s_1$ 
6      enquanto  $t > 1$  e  $\text{Ângulo}(u_i, s_t, s_{t-1}) < \pi$  faça
7        Desempilha( $S$ );  $t \leftarrow t - 1$ ;  $D \leftarrow D \cup \{u_i s_{t-1}\}$ 
10     Empilha( $S, u_i$ )
11     Caso (b):  $u_i$  adjacente a  $s_1$  mas não a  $s_t$ 
12        $aux \leftarrow s_t$ 
13       enquanto  $t > 1$  faça
14          $D \leftarrow D \cup \{u_i s_t\}$ ; Desempilha( $S$ );  $t \leftarrow t - 1$ 
17       Desempilha( $S$ )  $\triangleright$  desempilha  $s_1$ 
18       Empilha( $S, aux$ ) Empilha( $S, u_i$ )
19     Caso (c):  $u_i$  adjacente a  $s_1$  e a  $s_t$   $\triangleright u_i = u_n$ 
20     Desempilha( $S$ )  $\triangleright$  desempilha  $s_t$ 
21     enquanto  $t > 2$  faça
22        $t \leftarrow t - 1$ ;  $D \leftarrow D \cup \{u_i s_t\}$ ; Desempilha( $S$ )

```

Triangula monótono em tempo linear

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

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 $O(n)$.