

Geometria Computacional

Cristina G. Fernandes

Departamento de Ciência da Computação do IME-USP

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

segundo semestre de 2014

Estrutura de dados

Problema: Dado um conjunto finito P de pontos no \mathbb{R}^3 , encontrar o fecho convexo $\text{conv}(P)$ dos pontos em P .

Estrutura de dados *winged-edge* (**arestas aladas**)

Estrutura de dados

Problema: Dado um conjunto finito P de pontos no \mathbb{R}^3 , encontrar o fecho convexo $\text{conv}(P)$ dos pontos em P .

Estrutura de dados *winged-edge* (**arestas aladas**)

A ED mantém uma lista de **vértices**, **arestas** e **faces** onde

Vértice. cada vértice v mantém as suas coordenadas (x, y, z) e um apontador $av(v)$ para uma aresta arbitrária incidente a v ;

Face. cada face f mantém um apontador para uma aresta arbitrária $af(f)$ da fronteira de f ;

Aresta. cada aresta e tem oito apontadores...

Arestas aladas

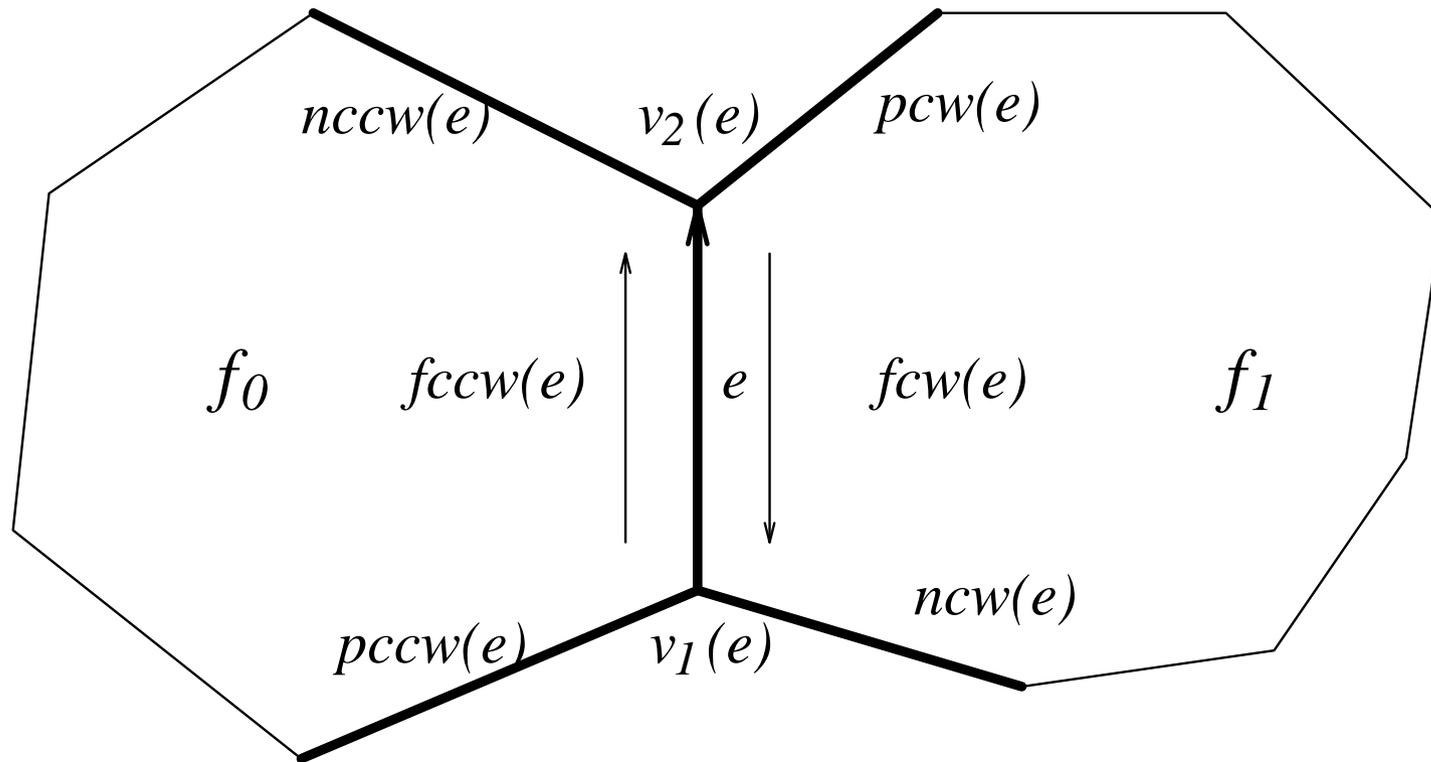
Cada aresta e tem oito apontadores:

- Dois apontadores para os extremos $v_1(e)$ e $v_2(e)$ de e . A ordem destes vértices fornece uma orientação para e .
- Apontadores $f_{ccw}(e)$ e $f_{cw}(e)$ para as duas faces incidentes a e . A face $f_{ccw}(e)$ é a esquerda de $e = v_1(e)v_2(e)$ e a face $f_{cw}(e)$ é a direita.
- Quatro apontadores para as **asas** (*wings*) de e : arestas que precedem e sucedem e em $f_{ccw}(e)$ e $f_{cw}(e)$.

Especificamente, $p_{ccw}(e)$ e $n_{ccw}(e)$ representam as arestas que precedem e sucedem e na face $f_{ccw}(e)$ (sentido anti-horário).

Analogamente, $p_{cw}(e)$ e $n_{cw}(e)$ representam as arestas que precedem e sucedem e na face $f_{cw}(e)$.

Arestas aladas



Note que cada registro dessa ED ocupa espaço constante.

Embrulho para presente

Embrulho para presente 2D: [Jarvis](#).

Para dimensões arbitrárias: [Chand & Kapur](#).

Embrulho para presente

Embrulho para presente 2D: **Jarvis**.

Para dimensões arbitrárias: **Chand & Kapur**.

Consumo de tempo para calcular $\text{conv}(P)$:

$O(nh)$, onde $n = |P|$ e h é o número de arestas de $\text{conv}(P)$.

Embrulho para presente

Embrulho para presente 2D: **Jarvis**.

Para dimensões arbitrárias: **Chand & Kapur**.

Consumo de tempo para calcular $\text{conv}(P)$:

$O(nh)$, onde $n = |P|$ e h é o número de arestas de $\text{conv}(P)$.

Suponha que $|P| \geq 4$.

Hipótese simplificadora: pontos de P estão em posição geral, ou seja, não existem quatro pontos em P que sejam coplanares.

Embrulho para presente

Embrulho para presente 2D: **Jarvis**.

Para dimensões arbitrárias: **Chand & Kapur**.

Consumo de tempo para calcular $\text{conv}(P)$:

$O(nh)$, onde $n = |P|$ e h é o número de arestas de $\text{conv}(P)$.

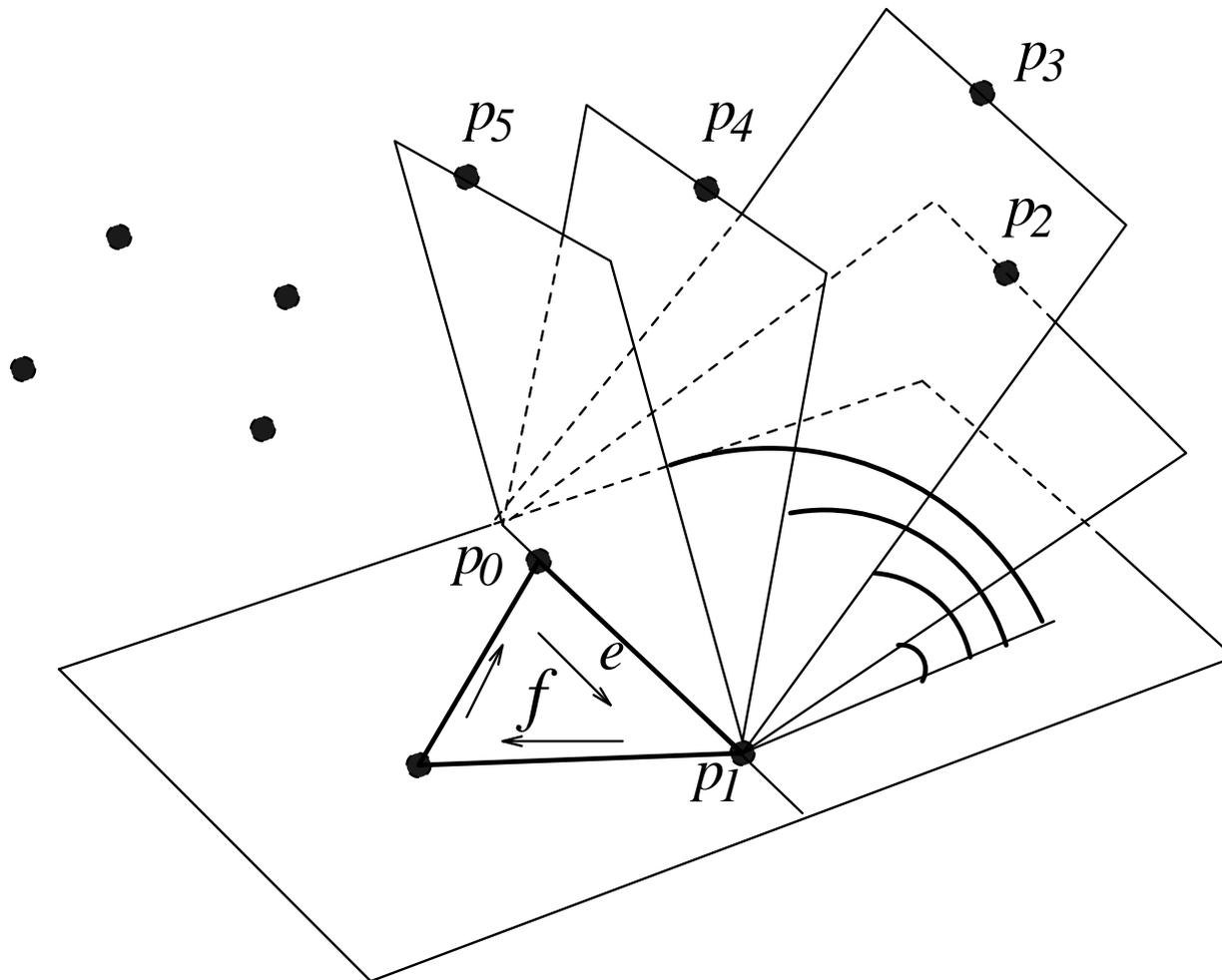
Suponha que $|P| \geq 4$.

Hipótese simplificadora: pontos de P estão em posição geral, ou seja, não existem quatro pontos em P que sejam coplanares.

Consequência: $\text{conv}(P)$ é simplicial (faces triangulares).

Embrulho para presente

Ilustração de um passo do algoritmo **EMBRULHO3D**.



Embrulho para presente

O algoritmo é iterativo.

Embrulho para presente

O algoritmo é iterativo.

Cada iteração começa com um conjunto de faces do fecho já determinadas, e um conjunto de arestas a serem processadas: **arestas que estão em exatamente uma das faces já determinadas.**

Embrulho para presente

O algoritmo é iterativo.

Cada iteração começa com um conjunto de faces do fecho já determinadas, e um conjunto de arestas a serem processadas: **arestas que estão em exatamente uma das faces já determinadas.**

A cada iteração, o algoritmo toma uma destas arestas, e determina a segunda face do fecho que a contém, incluindo no conjunto de arestas a serem processadas algumas novas arestas.

Inicialização do algoritmo Embrulho

Encontre um ponto extremo p_0 :

Tome um ponto com coordenada Z menor possível.

Inicialização do algoritmo Embrulho

Encontre um ponto extremo p_0 :

Tome um ponto com coordenada Z menor possível.

Encontre uma face contendo p_0 :

Tome a reta por p_0 paralela ao eixo das abscissas e o semi-plano horizontal π contendo p_0 e orientado positivamente na direção do eixo y .

Gire π no sentido de Y para Z

até encontrar outro ponto extremo p_1 da coleção.

Inicialização do algoritmo Embrulho

Encontre um ponto extremo p_0 :

Tome um ponto com coordenada Z menor possível.

Encontre uma face contendo p_0 :

Tome a reta por p_0 paralela ao eixo das abscissas e o semi-plano horizontal π contendo p_0 e orientado positivamente na direção do eixo y .

Gire π no sentido de Y para Z até encontrar outro ponto extremo p_1 da coleção.

Encontre uma face (triangular) contendo p_0p_1 :

Gire o novo plano π em torno da aresta p_0p_1 até encontrar um outro ponto extremo p_2 da coleção.

Embrulho para presente

EMBRULHO3D(P, n)

- 1 **CRIEFILA**(Q) \triangleright fila com as faces encontradas
- 2 **CRIEWE**(T) \triangleright ED winged edges para $\text{conv}(S)$
- 3 $f \leftarrow$ **FACEINICIAL**(P, n)
- 4 **INSIRAFILA**(Q, f)
- 5 **INSIRAWE**(T, f)
- 6 enquanto não **FILAVAZIA**(Q) faça
- 7 $f \leftarrow$ **REMOVAFILA**(Q)
- 8 para cada aresta livre e de f faça
- 9 $f' \leftarrow$ **FACEADJACENTE**(f, e)
- 10 **INSIRAFILA**(Q, f')
- 11 **INSIRAWE**(T, f)
- 12 devolva T

Embrulho para presente

EMBRULHO3D(P, n)

- 1 **CRIEFILA**(Q) \triangleright fila com as faces encontradas
- 2 **CRIEWE**(T) \triangleright ED winged edges para $\text{conv}(S)$
- 3 $f \leftarrow$ **FACEINICIAL**(P, n)
- 4 **INSIRAFILA**(Q, f)
- 5 **INSIRAWE**(T, f)
- 6 enquanto não **FILAVAZIA**(Q) faça
- 7 $f \leftarrow$ **REMOVAFILA**(Q)
- 8 para cada aresta livre e de f faça
- 9 $f' \leftarrow$ **FACEADJACENTE**(f, e)
- 10 **INSIRAFILA**(Q, f')
- 11 **INSIRAWE**(T, f')
- 12 devolva T

Consumo de tempo:

$O(hn)$, onde h é o número de arestas do fecho convexo.

Algoritmo incremental

INCREMENTAL(P, n)

- 1 $P_3 \leftarrow \text{conv}(\{p_0, p_1, p_2, p_3\})$
- 2 para $k \leftarrow 4$ até $n - 1$ faça
- 3 $P_k \leftarrow \text{conv}(P_{k-1} \cup \{p_k\})$
- 4 devolva P_{n-1}

Algoritmo incremental

INCREMENTAL(P, n)

- 1 $P_3 \leftarrow \text{conv}(\{p_0, p_1, p_2, p_3\})$
- 2 para $k \leftarrow 4$ até $n - 1$ faça
- 3 $P_k \leftarrow \text{conv}(P_{k-1} \cup \{p_k\})$
- 4 devolva P_{n-1}

Linha 3: dois casos a serem tratados.

- $p_k \in P_{k-1}$
Esta decisão pode ser feita em $O(n)$,
usando a rotina **VOLUME6**.
- $p_k \notin P_{k-1}$
Generalizaremos a ideia da versão 2D deste algoritmo.

Se $p_k \notin P_{k-1} \dots$

No caso 2D, encontrávamos as duas retas que passavam pelo ponto p_k e que são tangentes ao polígono P_{k-1} .

Se $p_k \notin P_{k-1} \dots$

No caso 2D, encontrávamos as duas retas que passavam pelo ponto p_k e que são tangentes ao polígono P_{k-1} .

No caso 3D, encontramos **planos tangentes** em vez de retas tangentes.

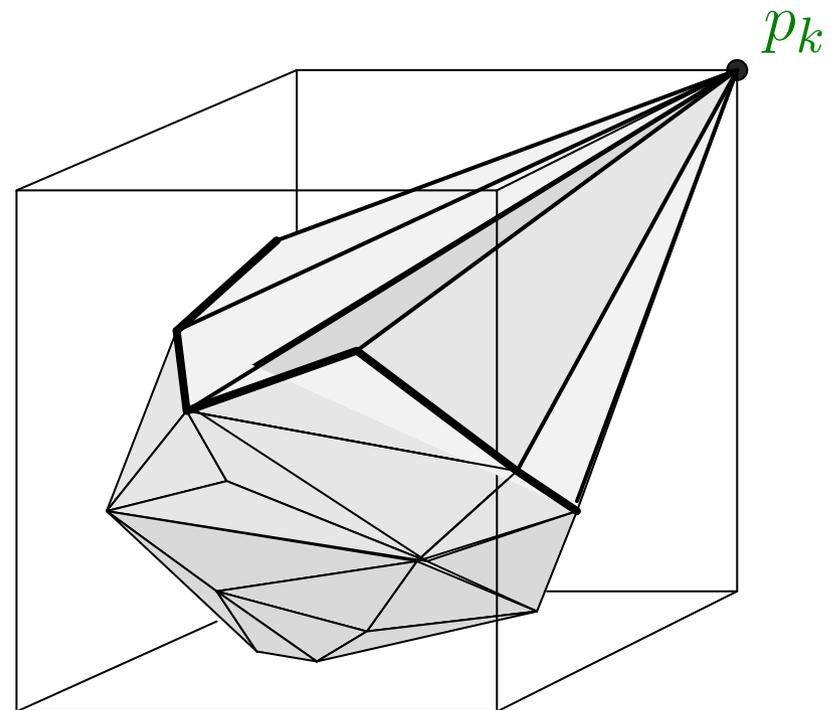
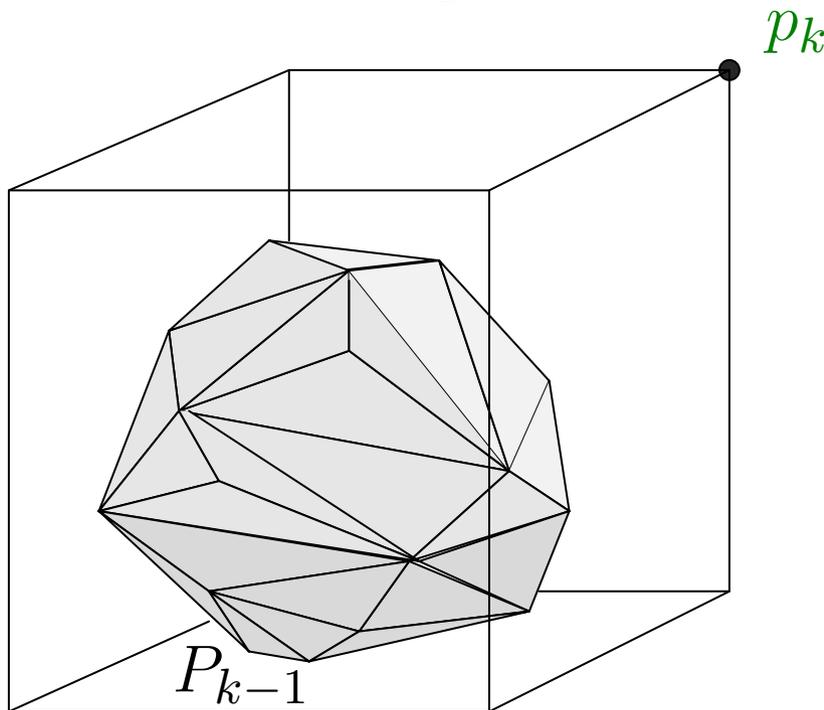
Estes **planos tangentes** determinam um cone que tem como faces triângulos e tem como bico o ponto p_k .

Se $p_k \notin P_{k-1} \dots$

No caso 2D, encontrávamos as duas retas que passavam pelo ponto p_k e que são tangentes ao polígono P_{k-1} .

No caso 3D, encontramos **planos tangentes** em vez de retas tangentes.

Estes **planos tangentes** determinam um cone que tem como faces triângulos e tem como bico o ponto p_k .



Como encontrar tais faces?

Faces de P_{k-1} a serem descartadas:
aquelas que são visíveis da posição que está o ponto p_k .

Como encontrar tais faces?

Faces de P_{k-1} a serem descartadas:

aquelas que são visíveis da posição que está o ponto p_k .

Faces visíveis: têm orientação positiva olhando de p_k .

Face $f = \triangle(a, b, c)$ é visível de p se o sinal do volume do tetraedro formado por a, b, c e p é positivo.

Como encontrar tais faces?

Faces de P_{k-1} a serem descartadas:

aquelas que são visíveis da posição que está o ponto p_k .

Faces visíveis: têm orientação positiva olhando de p_k .

Face $f = \triangle(a, b, c)$ é visível de p se o sinal do volume do tetraedro formado por a, b, c e p é positivo.

Arestas na fronteira das faces visíveis:

formarão as faces triangulares do cone com o ponto p_k

Como encontrar tais faces?

Faces de P_{k-1} a serem descartadas:

aquelas que são visíveis da posição que está o ponto p_k .

Faces visíveis: têm orientação positiva olhando de p_k .

Face $f = \triangle(a, b, c)$ é visível de p se o sinal do volume do tetraedro formado por a, b, c e p é positivo.

Arestas na fronteira das faces visíveis:

formarão as faces triangulares do cone com o ponto p_k

Suponha que e é uma aresta de P_{k-1} tal que o plano contendo e e o ponto p_k é tangente a P_{k-1} .

Como encontrar tais faces?

Faces de P_{k-1} a serem descartadas:

aquelas que são visíveis da posição que está o ponto p_k .

Faces visíveis: têm orientação positiva olhando de p_k .

Face $f = \triangle(a, b, c)$ é visível de p se o sinal do volume do tetraedro formado por a, b, c e p é positivo.

Arestas na fronteira das faces visíveis:

formarão as faces triangulares do cone com o ponto p_k

Suponha que e é uma aresta de P_{k-1} tal que o plano contendo e e o ponto p_k é tangente a P_{k-1} .

Cada aresta é compartilhada por exatamente duas faces. Uma das faces incidentes a e é visível a partir de p_k e a outra não. Logo, e está na fronteira da região visível de p_k .

Algoritmo incremental

INCREMENTAL3D(P, n)

```
1   $P_3 \leftarrow \text{TETRAEDRO}(p_0, p_1, p_2, p_3)$ 
2  para  $k \leftarrow 4$  até  $n - 1$  faça
3    para cada face  $f$  de  $P_{k-1}$  faça
4       $v \leftarrow \text{VOLUME6}(f, p_k)$ 
5      se  $v > 0$  então marque  $f$  como visível de  $p_k$ 
6  se nenhuma face é visível de  $p_k$ 
7    então  $P_k \leftarrow P_{k-1}$ 
8  senão para cada  $e$  na fronteira das faces visíveis
9    construa a face determinada por  $e$  e  $p_k$ 
10   para cada face visível  $f$ 
11     remova  $f$  de  $P_{k-1}$ 
12   faça os acertos finais obtendo  $P_k$ 
13 devolva  $P_{n-1}$ 
```

Algoritmo incremental

INCREMENTAL3D(P, n)

```
1   $P_3 \leftarrow \text{TETRAEDRO}(p_0, p_1, p_2, p_3)$ 
2  para  $k \leftarrow 4$  até  $n - 1$  faça
3    para cada face  $f$  de  $P_{k-1}$  faça
4       $v \leftarrow \text{VOLUME6}(f, p_k)$ 
5      se  $v > 0$  então marque  $f$  como visível de  $p_k$ 
6  se nenhuma face é visível de  $p_k$ 
7    então  $P_k \leftarrow P_{k-1}$ 
8  senão para cada  $e$  na fronteira das faces visíveis
9    construa a face determinada por  $e$  e  $p_k$ 
10   para cada face visível  $f$ 
11     remova  $f$  de  $P_{k-1}$ 
12   faça os acertos finais obtendo  $P_k$ 
13 devolva  $P_{n-1}$ 
```

Consumo de tempo: Pela fórmula de Euler, é $O(n^2)$.

Algoritmo incremental probabilístico

A versão probabilística do algoritmo escolhe uma permutação uniformemente dos n pontos dados, e os processa nesta ordem.

Algoritmo incremental probabilístico

A versão probabilística do algoritmo escolhe uma permutação uniformemente dos n pontos dados, e os processa nesta ordem.

Essa versão consome tempo $O(n \lg n)$.

Isso está provado na seção 11.3 do livro de de Berg et al.

Algoritmo incremental probabilístico

A versão probabilística do algoritmo escolhe uma permutação uniformemente dos n pontos dados, e os processa nesta ordem.

Essa versão consome tempo $O(n \lg n)$.

Isso está provado na seção 11.3 do livro de de Berg et al.

Lema: O número esperado de faces criadas por esta variante do **INCREMENTAL3D** é no máximo $6n - 20$.

Prova feita na aula.

Algoritmo incremental probabilístico

A versão probabilística do algoritmo escolhe uma permutação uniformemente dos n pontos dados, e os processa nesta ordem.

Essa versão consome tempo $O(n \lg n)$.

Isso está provado na seção 11.3 do livro de de Berg et al.

Lema: O número esperado de faces criadas por esta variante do **INCREMENTAL3D** é no máximo $6n - 20$.

Prova feita na aula.

Observe que o lema acima garante que o custo total para construir os fechos P_1, \dots, P_n poderia ser $O(n)$ em princípio.

No entanto, para isso, temos que ser capazes de encontrar as faces visíveis sem testar a visibilidade face a face.

Algoritmo incremental probabilístico

Para isso, o algoritmo carrega um grafo bipartido G onde um lado da bipartição são os vértices p_k, \dots, p_n e do outro lado da bipartição as faces de P_{k-1} .

Um vértice p_t é adjacente a uma face f se enxerga f .

Algoritmo incremental probabilístico

Para isso, o algoritmo carrega um grafo bipartido G onde um lado da bipartição são os vértices p_k, \dots, p_n e do outro lado da bipartição as faces de P_{k-1} .

Um vértice p_t é adjacente a uma face f se enxerga f .

$F_{conflito}(p)$: conjunto das faces vizinhas em G ao vértice p

$P_{conflito}(f)$: conjunto dos vértices vizinhos em G à face f

Algoritmo incremental probabilístico

Para isso, o algoritmo carrega um grafo bipartido G onde um lado da bipartição são os vértices p_k, \dots, p_n e do outro lado da bipartição as faces de P_{k-1} .

Um vértice p_t é adjacente a uma face f se enxerga f .

$F_{conflito}(p)$: conjunto das faces vizinhas em G ao vértice p

$P_{conflito}(f)$: conjunto dos vértices vizinhos em G à face f

Em cada iteração, $F_{conflito}(p_k)$ são as faces a serem removidas de P_{k-1} .

A partir destas faces, pode-se calcular o conjunto \mathcal{F} das arestas na borda da região visível.

Algoritmo incremental probabilístico

De posse dos conjuntos $F_{\text{conflicto}}(p_k)$ e \mathcal{F} , podemos atualizar a ED das arestas aladas, para que passe a representar P_k .

O tempo para isso é proporcional à soma do tamanho dos dois conjuntos, e isso, pelo lema anterior, é $O(n)$ no total.

Algoritmo incremental probabilístico

De posse dos conjuntos $F_{\text{conflito}}(p_k)$ e \mathcal{F} , podemos atualizar a ED das arestas aladas, para que passe a representar P_k .

O tempo para isso é proporcional à soma do tamanho dos dois conjuntos, e isso, pelo lema anterior, é $O(n)$ no total.

Mas... adicionalmente precisamos atualizar G .

Isso envolve:

- remover p_k ;
- remover as faces de $F_{\text{conflito}}(p_k)$;
- inserir as faces novas, uma para cada $e \in \mathcal{F}$;
- determinar o conjunto $P_{\text{conflito}}(f)$ para cada face nova f .

Algoritmo incremental probabilístico

Como determinar o conjunto $P_{\text{conflito}}(f)$ para cada f ?

Esta é a parte do algoritmo que consome $O(n \lg n)$ no total.

Algoritmo incremental probabilístico

Como determinar o conjunto $P_{\text{conflito}}(f)$ para cada f ?

Esta é a parte do algoritmo que consome $O(n \lg n)$ no total.

Para mostrar isso, primeiro observe que, se $e \in \mathcal{F}$ e f é a nova aresta formada por e e p_k , então

$$P_{\text{conflito}}(f) \subseteq P_{\text{conflito}}(f_1) \cup P_{\text{conflito}}(f_2)$$

onde f_1 e f_2 são as faces de P_{k-1} que compartilham e .

Algoritmo incremental probabilístico

Como determinar o conjunto $P_{\text{conflito}}(f)$ para cada f ?

Esta é a parte do algoritmo que consome $O(n \lg n)$ no total.

Para mostrar isso, primeiro observe que, se $e \in \mathcal{F}$ e f é a nova aresta formada por e e p_k , então

$$P_{\text{conflito}}(f) \subseteq P_{\text{conflito}}(f_1) \cup P_{\text{conflito}}(f_2)$$

onde f_1 e f_2 são as faces de P_{k-1} que compartilham e .

Seja $P(e) = P_{\text{conflito}}(f_1) \cup P_{\text{conflito}}(f_2)$.

Algoritmo incremental probabilístico

Seja $P(e) = P_{\text{conflito}}(f_1) \cup P_{\text{conflito}}(f_2)$.

Podemos calcular o conjunto $P_{\text{conflito}}(f)$ para cada face nova f em tempo $O(|P(e)|)$, usando a rotina **VOLUME6**.

Algoritmo incremental probabilístico

Seja $P(e) = P_{\text{conflito}}(f_1) \cup P_{\text{conflito}}(f_2)$.

Podemos calcular o conjunto $P_{\text{conflito}}(f)$ para cada face nova f em tempo $O(|P(e)|)$, usando a rotina **VOLUME6**.

No livro de de Berg et al., se encontra a prova de que

$$E[\sum_e |P(e)|] = O(n \lg n)$$

onde a soma é sobre todas as arestas que fizeram parte de \mathcal{F} em alguma iteração do algoritmo. Então...

Algoritmo incremental probabilístico

Seja $P(e) = P_{\text{conflito}}(f_1) \cup P_{\text{conflito}}(f_2)$.

Podemos calcular o conjunto $P_{\text{conflito}}(f)$ para cada face nova f em tempo $O(|P(e)|)$, usando a rotina **VOLUME6**.

No livro de de Berg et al., se encontra a prova de que

$$E[\sum_e |P(e)|] = O(n \lg n)$$

onde a soma é sobre todas as arestas que fizeram parte de \mathcal{F} em alguma iteração do algoritmo. Então...

Lema: O consumo de tempo esperado desta variante do **INCREMENTAL3D** é $O(n \lg n)$ para calcular o fecho convexo de n pontos do \mathbb{R}^3 .