

Geometria Computacional

Cristina G. Fernandes

Departamento de Ciência da Computação do IME-USP

<http://www.ime.usp.br/~cris/>

segundo semestre de 2011

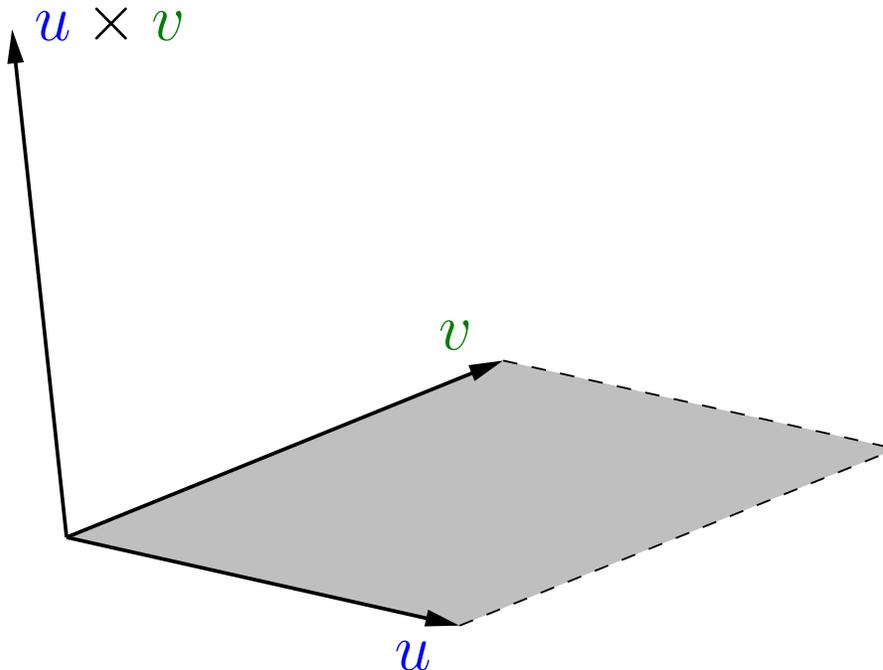
Recordação...

u e v : vetores no \mathbb{R}^3 com ponta inicial na origem.

Recordação...

u e v : vetores no \mathbb{R}^3 com ponta inicial na origem.

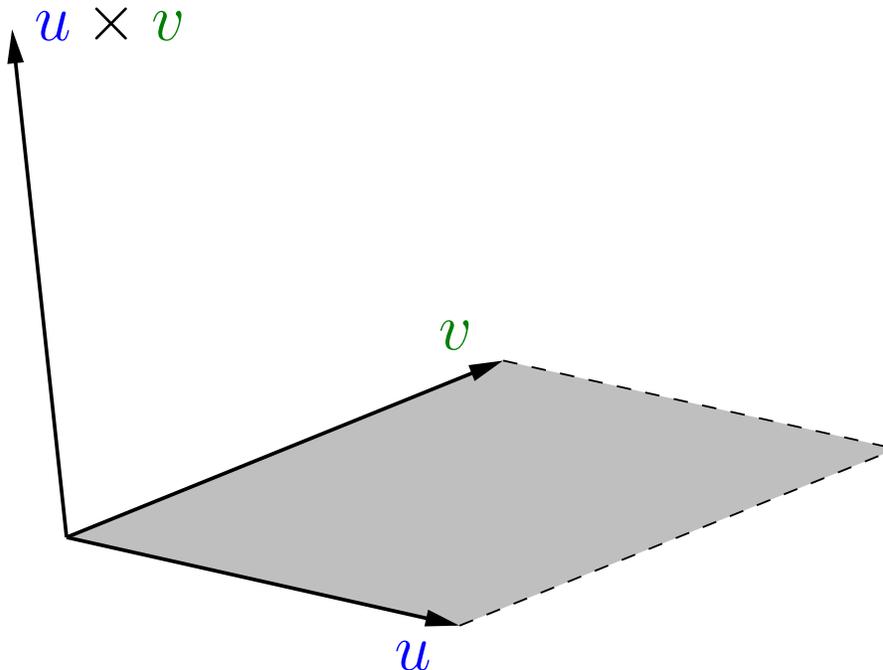
Produto vetorial $u \times v$: vetor perpendicular ao plano que contem u e v cujo comprimento é a **área do paralelepípedo** formado por u e v . Com que sentido?



Recordação...

u e v : vetores no \mathbb{R}^3 com ponta inicial na origem.

Produto vetorial $u \times v$: vetor perpendicular ao plano que contem u e v cujo comprimento é a **área do paralelepípedo** formado por u e v . Com que sentido?



Regra da mão direita: indicador na direção de u , dedo médio na direção de v ; o polegar indica o sentido de $u \times v$.

Produto vetorial

$$u = (u_1, u_2, u_3)$$

$$v = (v_1, v_2, v_3)$$

Produto vetorial

$$u = (u_1, u_2, u_3)$$

$$v = (v_1, v_2, v_3)$$

$$u \times v = (u_2v_3 - u_3v_2, u_3v_1 - u_1v_3, u_1v_2 - u_2v_1)$$

Produto vetorial

$$u = (u_1, u_2, u_3)$$

$$v = (v_1, v_2, v_3)$$

$$u \times v = (u_2v_3 - u_3v_2, u_3v_1 - u_1v_3, u_1v_2 - u_2v_1)$$

Alternativamente,

$$u \times v = \det \begin{pmatrix} \vec{i} & \vec{j} & \vec{k} \\ u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \end{pmatrix} .$$

Produto vetorial

$$u = (u_1, u_2, u_3)$$

$$v = (v_1, v_2, v_3)$$

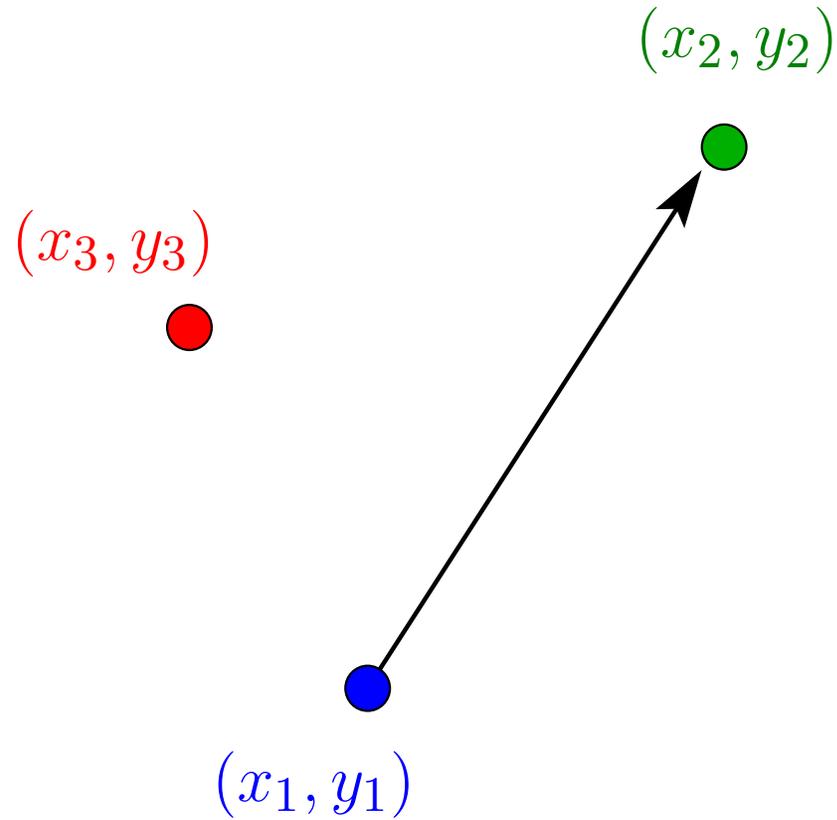
$$u \times v = (u_2v_3 - u_3v_2, u_3v_1 - u_1v_3, u_1v_2 - u_2v_1)$$

Alternativamente,

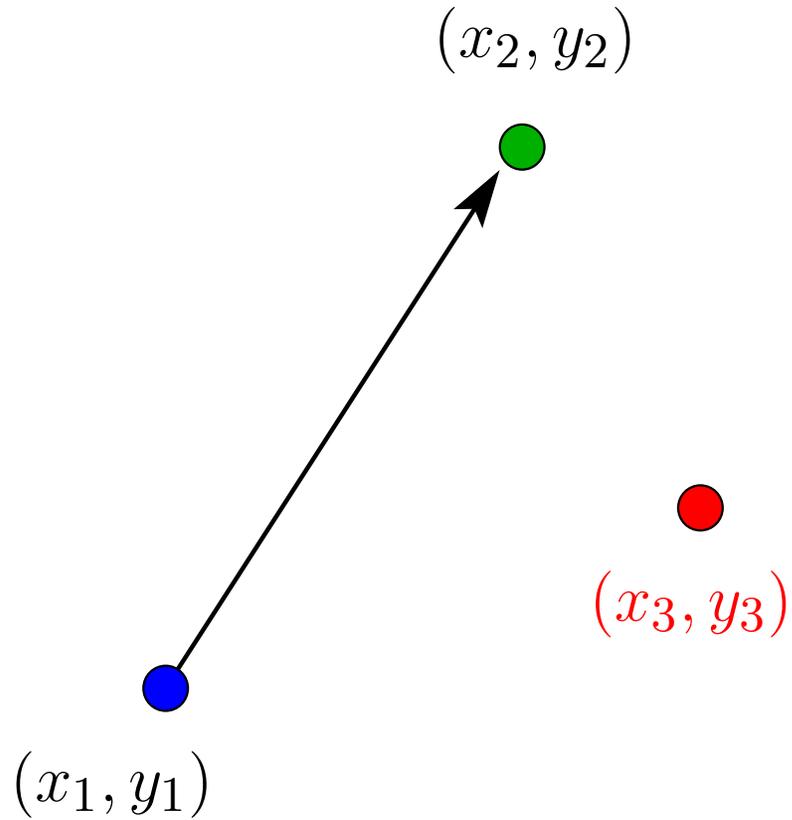
$$u \times v = \det \begin{pmatrix} \vec{i} & \vec{j} & \vec{k} \\ u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \end{pmatrix}.$$

Como usar isso para **decidir entre esquerda e direita?**

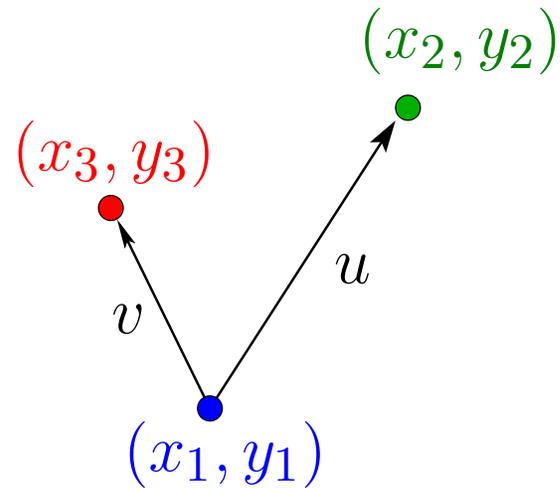
Esquerda e direita



Esquerda e direita

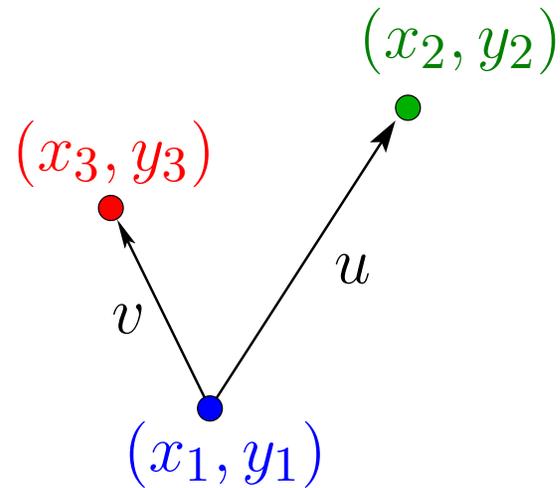


Esquerda e direita



Tomemos $u = (x_2 - x_1, y_2 - y_1, 0)$ e $v = (x_3 - x_1, y_3 - y_1, 0)$.

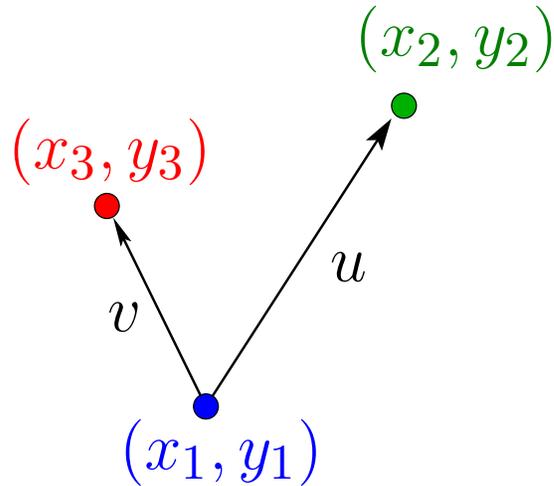
Esquerda e direita



Tomemos $u = (x_2 - x_1, y_2 - y_1, 0)$ e $v = (x_3 - x_1, y_3 - y_1, 0)$.

$$u \times v = (0, 0, (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1))$$

Esquerda e direita



Tomemos $u = (x_2 - x_1, y_2 - y_1, 0)$ e $v = (x_3 - x_1, y_3 - y_1, 0)$.

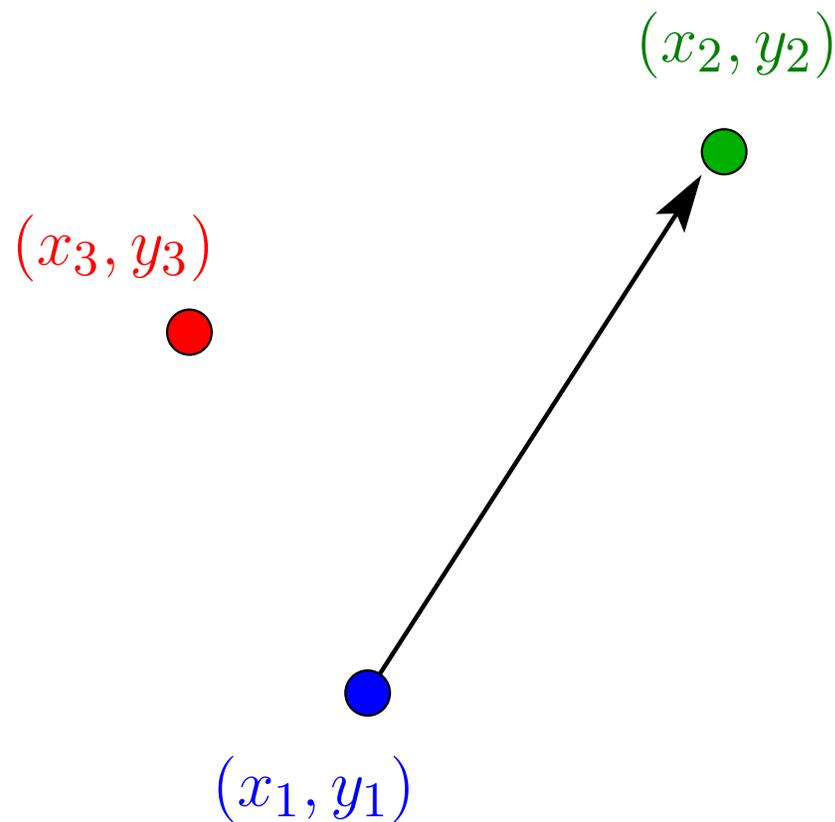
$$u \times v = (0, 0, (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1))$$

Se $(x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1) \geq 0$,

(x_3, y_3) está à esquerda, senão está à direita.

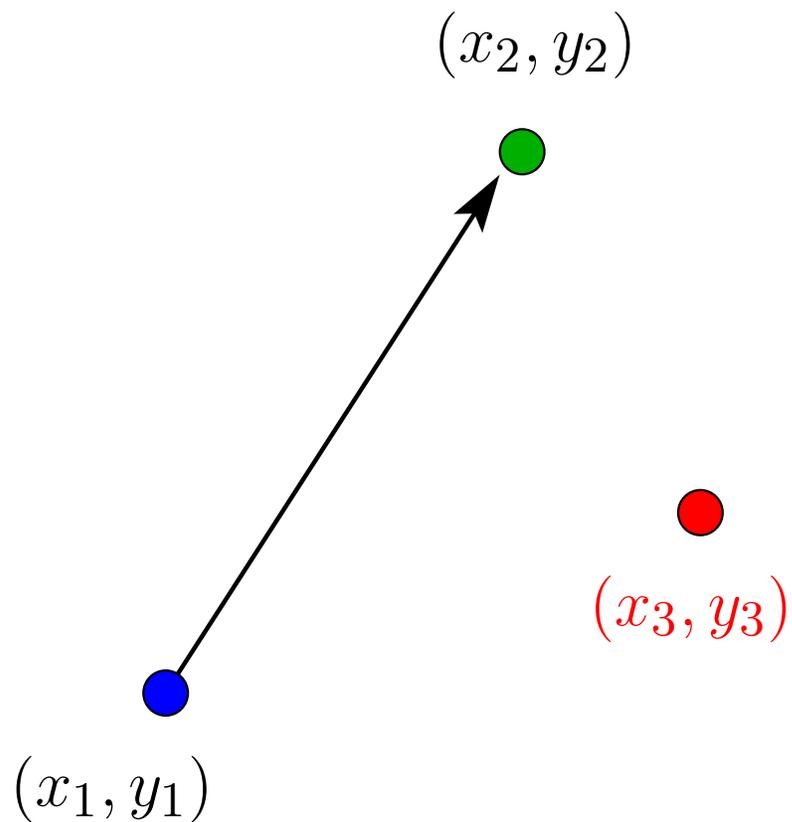
Predicados geométricos

ESQUERDA((x_1, y_1) , (x_2, y_2) , (x_3, y_3)) = VERDADE



Predicados geométricos

ESQUERDA($((x_1, y_1), (x_2, y_2), (x_3, y_3))$) = FALSO



Representação de ponto

Ponto: vetor de dimensão apropriada

Representação de ponto

Ponto: vetor de dimensão apropriada

Ficar nos inteiros enquanto for possível.

Representação de ponto

Ponto: vetor de dimensão apropriada

Ficar nos inteiros enquanto for possível.

```
#define X 0
#define Y 1
#define DIM 2 /* dimensão do espaço */

/* tipo ponto inteiro */
typedef int tPointi[DIM];

/* tipo ponto real */
typedef double tPointd[DIM];
```

Representação de polígono

Polígono: vetor ou lista ligada de pontos

Representação de polígono

Polígono: vetor ou lista ligada de pontos

Qual das duas opções escolher? **Depende...**

Representação de polígono

Polígono: vetor ou lista ligada de pontos

Qual das duas opções escolher? **Depende...**

Com vetor...

```
/* número máximo de pontos em um polígono */  
#define PMAX 1000  
  
/* tipo polígono de pontos inteiros */  
typedef tPointi tPolygoni[PMAX];  
  
/* tipo polígono de pontos reais */  
typedef tPointd tPolygond[PMAX];
```

Cálculos de área

O valor absoluto do determinante

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

Cálculos de área

O valor absoluto do determinante

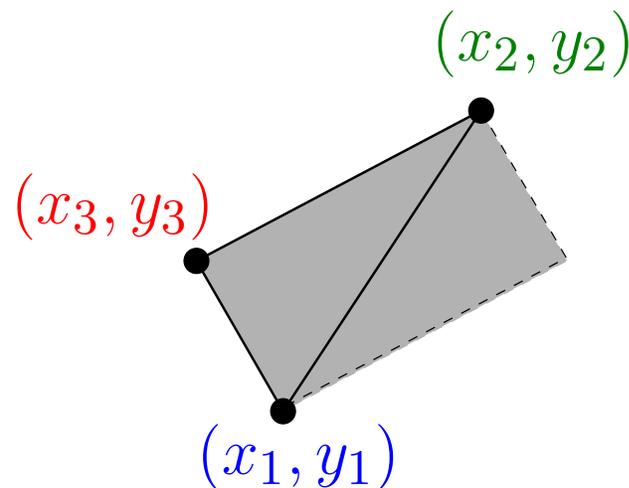
$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = (x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)$$

Cálculos de área

O valor absoluto do determinante

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = (x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)$$

é duas vezes a área do triângulo de extremos (x_1, y_1) , (x_2, y_2) e (x_3, y_3) .



Cálculos de área

Triângulo

```
int Area2 (tPointi a, b, c) {  
    return a[X]*b[Y]-a[Y]*b[X]+a[Y]*c[X]  
        -a[X]*c[Y]+b[X]*c[Y]-c[X]*b[Y];  
}
```

Cálculos de área

Triângulo

```
int Area2 (tPointi a, b, c) {  
    return a[X]*b[Y]-a[Y]*b[X]+a[Y]*c[X]  
        -a[X]*c[Y]+b[X]*c[Y]-c[X]*b[Y];  
}
```

Com menos multiplicações e em pseudocódigo:

Área2(a, b, c)

1 devolva $(a[X] - c[X]) * (b[Y] - c[Y]) -$
 $(a[Y] - c[Y]) * (b[X] - c[X])$

Cálculos de área

Triângulo

Área2(a, b, c)

1 **devolva** $(a[X] - c[X]) * (b[Y] - c[Y]) -$
 $(a[Y] - c[Y]) * (b[X] - c[X])$

Polígono

Cálculos de área

Triângulo

$\text{Área2}(a, b, c)$

1 **devolva** $(a[X] - c[X]) * (b[Y] - c[Y]) -$
 $(a[Y] - c[Y]) * (b[X] - c[X])$

Polígono

$\text{ÁreaPolígono2}(n, P)$

1 $s \leftarrow 0$

2 **para** $i \leftarrow 1$ **até** $n - 2$ **faça**

3 $s \leftarrow s + \text{Área2}(P[0], P[i], P[i + 1])$

4 **devolva** s

Segmentos e pontos

Ponto c à esquerda da reta dada por \vec{ab}

Esquerda⁺(a, b, c)

1 devolva $\text{Área2}(a, b, c) > 0$

Segmentos e pontos

Ponto c à esquerda da reta dada por \vec{ab}

Esquerda⁺(a, b, c)

1 devolva $\text{Área2}(a, b, c) > 0$

Ponto c à esquerda ou sobre a reta dada por \vec{ab}

Esquerda(a, b, c)

1 devolva $\text{Área2}(a, b, c) \geq 0$

Segmentos e pontos

Ponto c à esquerda da reta dada por \vec{ab}

Esquerda⁺(a, b, c)

1 devolva $\text{Área2}(a, b, c) > 0$

Ponto c à esquerda ou sobre a reta dada por \vec{ab}

Esquerda(a, b, c)

1 devolva $\text{Área2}(a, b, c) \geq 0$

Pontos a, b e c são colineares

Colinear(a, b, c)

1 devolva $\text{Área2}(a, b, c) = 0$

Interseção de segmentos

Interseção própria entre ab e cd :

a interseção é um único ponto no interior dos segmentos.

IntersectaProp(a, b, c, d)

- 1 **se** **Colinear**(a, b, c) **ou** **Colinear**(a, b, d)
ou **Colinear**(c, d, a) **ou** **Colinear**(c, d, b)
- 2 **então devolva** FALSO
- 3 **devolva** **Xor**(**Esquerda**⁺(a, b, c), **Esquerda**⁺(a, b, d))
e **Xor**(**Esquerda**⁺(c, d, a), **Esquerda**⁺(c, d, b))

A rotina **Xor** devolve o
ou exclusivo entre duas expressões booleanas.

Interseção de segmentos

Ponto c está no segmento ab

Entre(a, b, c)

1 **se não** **Colinear**(a, b, c)

2 **então devolva** FALSO

3 **se** $a[X] \neq b[X]$ $\triangleright ab$ não é vertical

4 **então devolva** $a[X] \leq c[X] \leq b[X]$ **ou** $b[X] \leq c[X] \leq a[X]$

5 **senão devolva** $a[Y] \leq c[Y] \leq b[Y]$ **ou** $b[Y] \leq c[Y] \leq a[Y]$

Interseção de segmentos

Ponto c está no segmento ab

Entre (a, b, c)

1 **se não** **Colinear** (a, b, c)

2 **então devolva** FALSO

3 **se** $a[X] \neq b[X]$ $\triangleright ab$ não é vertical

4 **então devolva** $a[X] \leq c[X] \leq b[X]$ **ou** $b[X] \leq c[X] \leq a[X]$

5 **senão devolva** $a[Y] \leq c[Y] \leq b[Y]$ **ou** $b[Y] \leq c[Y] \leq a[Y]$

Interseção entre ab e cd

Intersecta (a, b, c, d)

1 **se** **IntersectaProp** (a, b, c, d)

2 **então devolva** VERDADE

3 **devolva** **Entre** (a, b, c) **ou** **Entre** (a, b, d)

ou **Entre** (c, d, a) **ou** **Entre** (c, d, b)

Dentro ou fora?

Candidata a diagonal $P[i]P[j]$ está no interior do polígono?

Está no cone das arestas vizinhas do polígono?

NoCone (n, P, i, j)

1 $u \leftarrow i - 1 \pmod n$

2 $w \leftarrow i + 1 \pmod n$

3 **se** **Esquerda** $(P[u], P[i], P[w])$ $\triangleright P[i]$ é convexo

4 **então devolva** **Esquerda** $^+(P[i], P[j], P[u])$ **e**

Esquerda $^+(P[j], P[i], P[w])$

5 **senão devolva não** (**Esquerda** $(P[i], P[j], P[w])$ **e**

Esquerda $(P[j], P[i], P[u])$)

Teste de diagonal

Quase uma diagonal...

QuaseDiagonal(n, P, i, j)

1 **para** $k \leftarrow 0$ **até** $n - 1$

2 $\ell \leftarrow k + 1 \pmod{n}$

3 **se** $k \neq i$ **e** $k \neq j$ **e** $\ell \neq i$ **e** $\ell \neq j$

4 **então se** **Intersecta**($P[i], P[j], P[k], P[\ell]$)

5 **então devolva** FALSO

6 **devolva** VERDADE

Teste de diagonal

Quase uma diagonal...

QuaseDiagonal(n, P, i, j)

```
1  para  $k \leftarrow 0$  até  $n - 1$ 
2     $\ell \leftarrow k + 1 \pmod{n}$ 
3    se  $k \neq i$  e  $k \neq j$  e  $\ell \neq i$  e  $\ell \neq j$ 
4      então se Intersecta( $P[i], P[j], P[k], P[\ell]$ )
5        então devolva FALSO
6  devolva VERDADE
```

Diagonal de fato...

Diagonal(n, P, i, j)

```
1  devolva NoCone( $n, P, i, j$ ) e QuaseDiagonal( $n, P, i, j$ )
```

Teste de diagonal

Quase uma diagonal...

QuaseDiagonal(n, P, i, j)

1 **para** $k \leftarrow 0$ **até** $n - 1$

2 $\ell \leftarrow k + 1 \pmod{n}$

3 **se** $k \neq i$ **e** $k \neq j$ **e** $\ell \neq i$ **e** $\ell \neq j$

4 **então se** **Intersecta**($P[i], P[j], P[k], P[\ell]$)

5 **então devolva** FALSO

6 **devolva** VERDADE

Diagonal de fato...

Diagonal(n, P, i, j)

1 **devolva** **NoCone**(n, P, i, j) **e** **QuaseDiagonal**(n, P, i, j)

Tempo de execução: $\Theta(n)$

Um primeiro algoritmo de triangulação

Entrada: polígono $P[0 \dots n - 1]$ (vetor de n pontos)

Saída: coleção de diagonais da triangulação
(um par de índices de P descreve cada diagonal)

Um primeiro algoritmo de triangulação

Entrada: polígono $P[0 \dots n - 1]$ (vetor de n pontos)

Saída: coleção de diagonais da triangulação
(um par de índices de P descreve cada diagonal)

Descrição grosseira:

se $n = 3$, devolva o conjunto vazio

se $n > 3$, encontre uma diagonal por força bruta:

Um primeiro algoritmo de triangulação

Entrada: polígono $P[0 \dots n - 1]$ (vetor de n pontos)

Saída: coleção de diagonais da triangulação
(um par de índices de P descreve cada diagonal)

Descrição grosseira:

se $n = 3$, devolva o conjunto vazio

se $n > 3$, encontre uma diagonal por força bruta:

para $i \leftarrow 0$ até $n - 3$ faça

para $j \leftarrow i + 2$ até $n - 1$ faça

se **Diagonal**(n, P, i, j) i e j vértices não adjacentes

Um primeiro algoritmo de triangulação

Entrada: polígono $P[0 \dots n - 1]$ (vetor de n pontos)

Saída: coleção de diagonais da triangulação
(um par de índices de P descreve cada diagonal)

Descrição grosseira:

se $n = 3$, devolva o conjunto vazio

se $n > 3$, encontre uma diagonal por força bruta:

para $i \leftarrow 0$ até $n - 3$ faça

para $j \leftarrow i + 2$ até $n - 1$ faça

se **Diagonal**(n, P, i, j) i e j vértices não adjacentes

determine os polígonos P_1 e P_2 obtidos
da partição de P pela diagonal $P[i]P[j]$

Um primeiro algoritmo de triangulação

Entrada: polígono $P[0 \dots n - 1]$ (vetor de n pontos)

Saída: coleção de diagonais da triangulação
(um par de índices de P descreve cada diagonal)

Descrição grosseira:

se $n = 3$, devolva o conjunto vazio

se $n > 3$, encontre uma diagonal por força bruta:

para $i \leftarrow 0$ até $n - 3$ faça

para $j \leftarrow i + 2$ até $n - 1$ faça

se **Diagonal**(n, P, i, j) i e j vértices não adjacentes

determine os polígonos P_1 e P_2 obtidos

da partição de P pela diagonal $P[i]P[j]$

encontre recursivamente coleções D_1 e D_2

de diagonais que triangulem P_1 e P_2

Um primeiro algoritmo de triangulação

Entrada: polígono $P[0 \dots n - 1]$ (vetor de n pontos)

Saída: coleção de diagonais da triangulação
(um par de índices de P descreve cada diagonal)

Descrição grosseira:

se $n = 3$, devolva o conjunto vazio

se $n > 3$, encontre uma diagonal por força bruta:

para $i \leftarrow 0$ até $n - 3$ faça

para $j \leftarrow i + 2$ até $n - 1$ faça

se **Diagonal**(n, P, i, j) i e j vértices não adjacentes

determine os polígonos P_1 e P_2 obtidos

da partição de P pela diagonal $P[i]P[j]$

encontre recursivamente coleções D_1 e D_2

de diagonais que triangulem P_1 e P_2

devolva $D_1 \cup D_2 \cup \{P[i]P[j]\}$

Um primeiro algoritmo de triangulação

Descrição grosseira:

se $n = 3$, devolva o conjunto vazio

se $n > 3$, encontre uma diagonal por força bruta:

para $i \leftarrow 0$ até $n - 3$ faça

para $j \leftarrow i + 2$ até $n - 1$ faça

se **Diagonal**(n, P, i, j) i e j vértices não adjacentes

determine os polígonos P_1 e P_2 obtidos

da partição de P pela diagonal $P[i]P[j]$

encontre recursivamente coleções D_1 e D_2

de diagonais que triangulem P_1 e P_2

devolva $D_1 \cup D_2 \cup \{P[i]P[j]\}$

Um primeiro algoritmo de triangulação

Descrição grosseira:

se $n = 3$, devolva o conjunto vazio

se $n > 3$, encontre uma diagonal por força bruta:

para $i \leftarrow 0$ até $n - 3$ faça

para $j \leftarrow i + 2$ até $n - 1$ faça

se **Diagonal**(n, P, i, j) i e j vértices não adjacentes

determine os polígonos P_1 e P_2 obtidos

da partição de P pela diagonal $P[i]P[j]$

encontre recursivamente coleções D_1 e D_2

de diagonais que triangulem P_1 e P_2

devolva $D_1 \cup D_2 \cup \{P[i]P[j]\}$

Consumo de tempo no pior caso:

$$T(n) = T(n_1) + T(n_2) + O(n^3), \text{ onde } n_1 + n_2 = n + 2.$$

Um primeiro algoritmo de triangulação

Descrição grosseira:

se $n = 3$, devolva o conjunto vazio

se $n > 3$, encontre uma diagonal por força bruta:

para $i \leftarrow 0$ até $n - 3$ faça

para $j \leftarrow i + 2$ até $n - 1$ faça

se **Diagonal**(n, P, i, j) i e j vértices não adjacentes

determine os polígonos P_1 e P_2 obtidos

da partição de P pela diagonal $P[i]P[j]$

encontre recursivamente coleções D_1 e D_2

de diagonais que triangulem P_1 e P_2

devolva $D_1 \cup D_2 \cup \{P[i]P[j]\}$

Consumo de tempo no pior caso:

$$T(n) = T(n - 1) + O(n^3) = O(n^4)$$

Um primeiro algoritmo de triangulação

Triang-n4(n, P)

```
1  se  $n > 3$ 
2      então  $i \leftarrow 0$            $j \leftarrow 2$ 
3          enquanto não Diagonal( $n, P, i, j$ ) faça
4               $j \leftarrow j + 1$ 
5              se  $j = n$ 
6                  então  $i \leftarrow i + 1$ 
7                       $j \leftarrow i + 2$ 
8              imprima  $\{i, j\}$ 
9               $n_1 \leftarrow j - i + 1$ 
10              $n_2 \leftarrow n - n_1 + 2$ 
11              $P_1[0..n_1 - 1] \leftarrow P[i..j]$ 
12              $P_2[0..n_2 - 1] \leftarrow P[0..i] \cdot P[j..n - 1]$ 
13             Triang-n4( $n_1, P_1$ )
14             Triang-n4( $n_2, P_2$ )
```

Um primeiro algoritmo de triangulação

Triang-n4(n, P)

```
1  se  $n > 3$ 
2      então  $i \leftarrow 0$            $j \leftarrow 2$ 
3          enquanto não Diagonal( $n, P, i, j$ ) faça
4               $j \leftarrow j + 1$ 
5              se  $j = n$ 
6                  então  $i \leftarrow i + 1$ 
7                       $j \leftarrow i + 2$ 
8              imprima  $\{i, j\}$ 
9               $n_1 \leftarrow j - i + 1$ 
10              $n_2 \leftarrow n - n_1 + 2$ 
11              $P_1[0..n_1 - 1] \leftarrow P[i..j]$ 
12              $P_2[0..n_2 - 1] \leftarrow P[0..i] \cdot P[j..n - 1]$ 
13             Triang-n4( $n_1, P_1$ )
14             Triang-n4( $n_2, P_2$ )
```

Pior caso: $O(n^3)$ chamadas a Diagonal

Triangulação em $O(n^3)$: use orelhas!

PontaDeOrelha(n, P, i)

1 $j \leftarrow (i - 1) \bmod n$

2 $k \leftarrow (i + 1) \bmod n$

3 **devolva** Diagonal(n, P, j, k)

Triangulação em $O(n^3)$: use orelhas!

PontaDeOrelha(n, P, i)

- 1 $j \leftarrow (i - 1) \bmod n$
- 2 $k \leftarrow (i + 1) \bmod n$
- 3 **devolva** **Diagonal**(n, P, j, k)

Triang-n3(n, P)

- 1 **se** $n > 3$
- 2 **então** $i \leftarrow 0$
- 3 **enquanto não** **PontaDeOrelha**(n, P, i) **faça**
- 4 $i \leftarrow i + 1$
- 5 **imprima** $\{(i - 1) \bmod n, (i + 1) \bmod n\}$
- 6 $m, P' \leftarrow$ **Remove**(n, P, i)
- 7 **Triang-n3**(m, P')

Triangulação em $O(n^3)$: use orelhas!

PontaDeOrelha(n, P, i)

- 1 $j \leftarrow (i - 1) \bmod n$
- 2 $k \leftarrow (i + 1) \bmod n$
- 3 **devolva** **Diagonal**(n, P, j, k)

Triang-n3(n, P) \triangleright sem recursão de cauda

- 1 **enquanto** $n > 3$
- 2 $i \leftarrow 0$
- 3 **enquanto não** **PontaDeOrelha**(n, P, i) **faça**
- 4 $i \leftarrow i + 1$
- 5 **imprima** $\{(i - 1) \bmod n, (i + 1) \bmod n\}$
- 6 $P[i..n - 2] \leftarrow P[i + 1..n - 1]$ \triangleright remove o i
- 7 $n \leftarrow n - 1$

Triangulação em $O(n^3)$: use orelhas!

PontaDeOrelha(n, P, i)

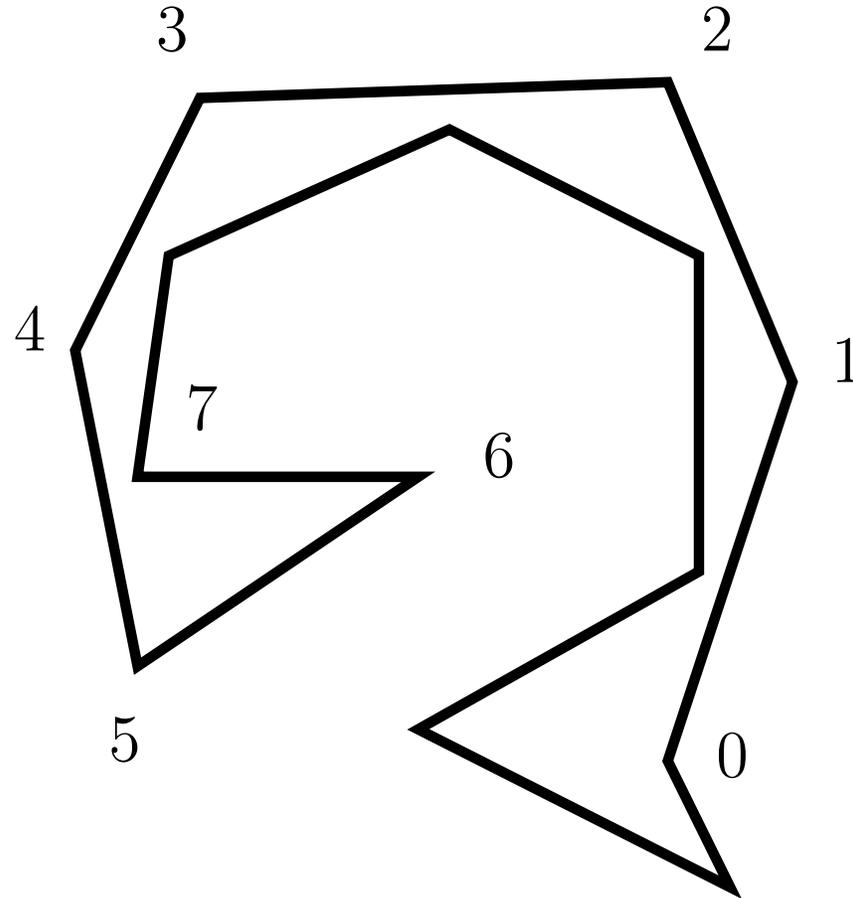
- 1 $j \leftarrow (i - 1) \bmod n$
- 2 $k \leftarrow (i + 1) \bmod n$
- 3 **devolva** **Diagonal**(n, P, j, k)

Triang-n3(n, P) \triangleright sem recursão de cauda

- 1 **enquanto** $n > 3$
- 2 $i \leftarrow 0$
- 3 **enquanto não** **PontaDeOrelha**(n, P, i) **faça**
- 4 $i \leftarrow i + 1$
- 5 **imprima** $\{(i - 1) \bmod n, (i + 1) \bmod n\}$
- 6 $P[i..n - 2] \leftarrow P[i + 1..n - 1]$ \triangleright remove o i
- 7 $n \leftarrow n - 1$

Pior caso: $\Theta(n^2)$ chamadas a **Diagonal**

Exemplo ruim



Triangulação em $O(n^2)$

Triang-n2(n, P)

```
1  MarcaOrelha( $P$ )
2  enquanto  $n > 3$  faça
3       $v_2 \leftarrow P$ 
4      enquanto não orelha[ $v_2$ ] faça
5           $v_2 \leftarrow prox[v_2]$ 
6           $v_1 \leftarrow prev[v_2]$ 
7           $v_3 \leftarrow prox[v_2]$ 
8          imprima {vert[ $v_1$ ], vert[ $v_3$ ]}
9           $prox[v_1] \leftarrow v_3$ 
10          $prev[v_3] \leftarrow v_1$ 
11          $P \leftarrow v_3$ 
12          $n \leftarrow n - 1$ 
13         orelha[ $v_1$ ]  $\leftarrow$  PontaDeOrelha( $P, v_1$ )
14         orelha[ $v_3$ ]  $\leftarrow$  PontaDeOrelha( $P, v_3$ )
```

Triangulação em $O(n^2)$

Triang-n2(n, P)

```
1  MarcaOrelha( $P$ )
2  enquanto  $n > 3$  faça
3       $v_2 \leftarrow P$ 
4      enquanto não orelha[ $v_2$ ] faça
5           $v_2 \leftarrow prox[v_2]$ 
6           $v_1 \leftarrow prev[v_2]$ 
7           $v_3 \leftarrow prox[v_2]$ 
8          imprima {vert[ $v_1$ ], vert[ $v_3$ ]}
9           $prox[v_1] \leftarrow v_3$ 
10          $prev[v_3] \leftarrow v_1$ 
11          $P \leftarrow v_3$ 
12          $n \leftarrow n - 1$ 
13         orelha[ $v_1$ ]  $\leftarrow$  PontaDeOrelha( $P, v_1$ )
14         orelha[ $v_3$ ]  $\leftarrow$  PontaDeOrelha( $P, v_3$ )
```

Pior caso: $\Theta(n)$ chamadas a Diagonal.

Orelhas com listas ligadas

PontaDeOrelha(P, v)

1 $u \leftarrow \text{prev}[v]$

2 $w \leftarrow \text{prox}[v]$

3 **devolva** Diagonal(P, u, w)

Orelhas com listas ligadas

PontaDeOrelha(P, v)

- 1 $u \leftarrow \text{prev}[v]$
- 2 $w \leftarrow \text{prox}[v]$
- 3 **devolva** **Diagonal**(P, u, w)

MarcaOrelha(P)

- 1 $v \leftarrow P$
- 2 **repita**
- 3 $u \leftarrow \text{prev}[v]$
- 4 $w \leftarrow \text{prox}[v]$
- 5 $\text{orelha}[v] \leftarrow \text{Diagonal}(P, u, w)$
- 6 $v \leftarrow w$
- 7 **até até** $v = P$

Orelhas com listas ligadas

PontaDeOrelha(P, v)

- 1 $u \leftarrow \text{prev}[v]$
- 2 $w \leftarrow \text{prox}[v]$
- 3 **devolva** **Diagonal**(P, u, w)

MarcaOrelha(P)

- 1 $v \leftarrow P$
- 2 **repita**
- 3 $u \leftarrow \text{prev}[v]$
- 4 $w \leftarrow \text{prox}[v]$
- 5 $\text{orelha}[v] \leftarrow \text{Diagonal}(P, u, w)$
- 6 $v \leftarrow w$
- 7 **até até** $v = P$

Diagonal também teria que ser reescrita para listas ligadas.

Exercícios

1. É verdade que um vértice de um polígono ou é ponta de orelha ou existe uma diagonal do polígono com extremo nele?
2. Qual é a relação da pergunta anterior com o consumo de tempo do algoritmo **Triang-n4**?
3. Escreva um algoritmo **PertenceConvexo**(P, n, q) que decide se um ponto q pertence ou não ao polígono convexo P , dado no vetor $P[0..n-1]$. **Dica:** Use a rotina **ÁREA2**.
4. A ideia do algoritmo do exercício anterior funciona também caso o polígono P não seja convexo? Elabore sobre isso.