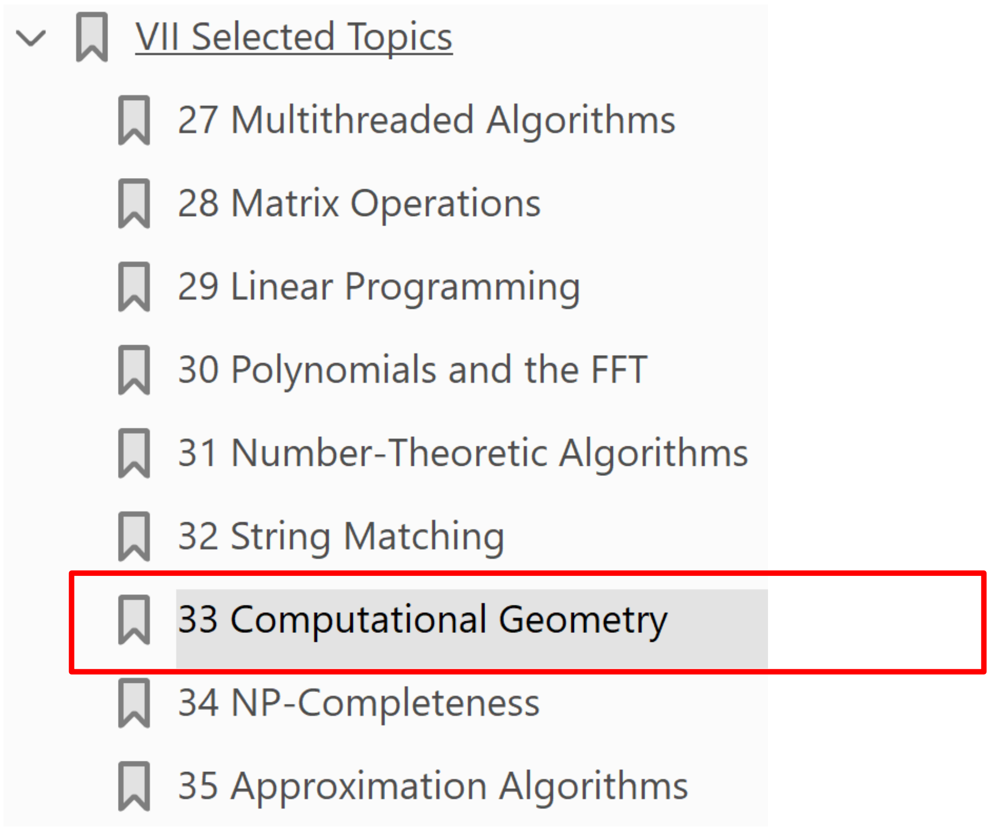
Chapter 33

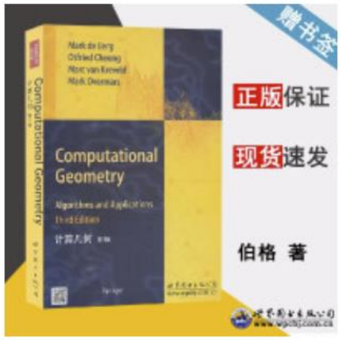
Computational Geometry

songyou@buaa.edu.cn

VII Selected Topics



33 Computational Geometry



¥63.20

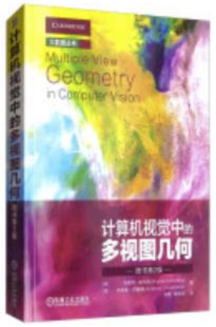
现货 计算几何 第三版 第3版 英文版 伯格

¥52.80

计算几何 第3版 新华书店，正版保证，关

¥58.40

科学计算及其软件教学丛书：计算几何教



¥130.40

计算机视觉中的多视图几何（原书第2

¥102.40

计算几何：空间数据处理算法 团购电话

¥146.50

计算共形几何 (理论篇) 100册以上团购优

¥30.80

包邮 计算几何 曲面表示论及其应用 罗钟

¥36.75

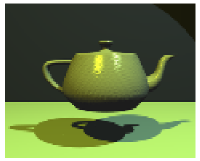
计算几何算法与实现（Visual C++版） 计

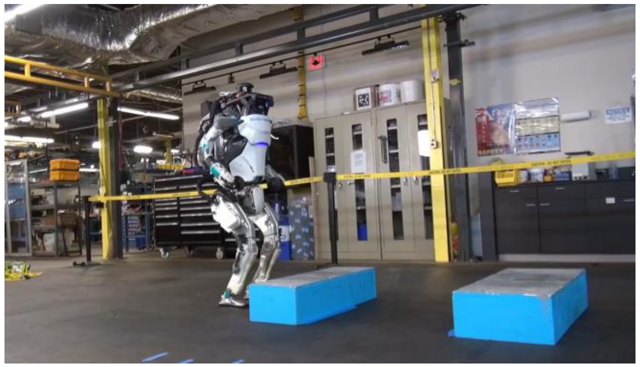
33 Computational Geometry

- A branch of CS (Computing Science) that studies algorithms for solving geometric problems

- Applications
  - computer graphics
  - Robotics
  - VLSI(very large-scale integration) design
  - computer- aided design

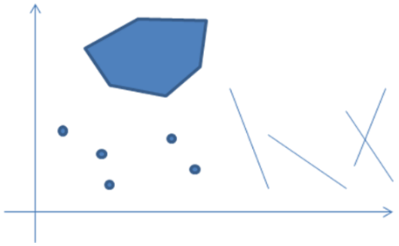
  …



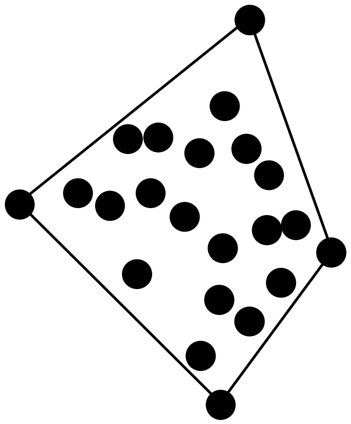


33 Computational Geometry

- Input: a description of a set of geometric objects
  - a set of points
  - a set of line segments
  - the vertices of a polygon in counterclockwise order
  - …



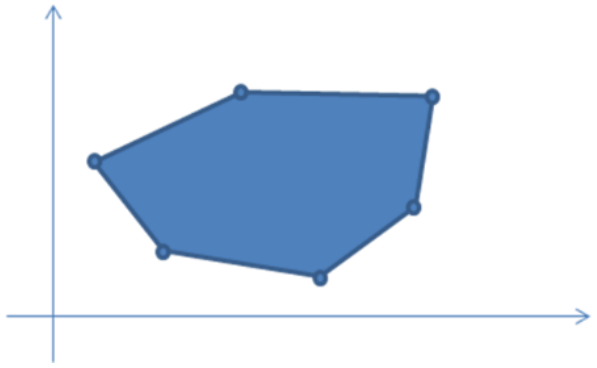
- Output: is often a response to a query about the objects
  - whether any of the lines intersect
  - some new geometric object, such as the convex hull (smallest enclosing convex polygon) …



33 Computational Geometry

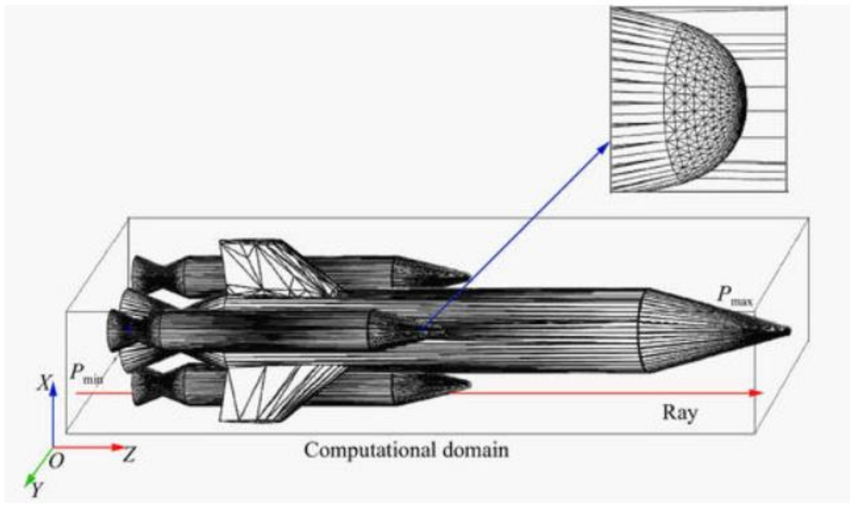
- In this chapter, we look at a few computational-geometry algorithms in **two dimensions**, that is, in the plane.

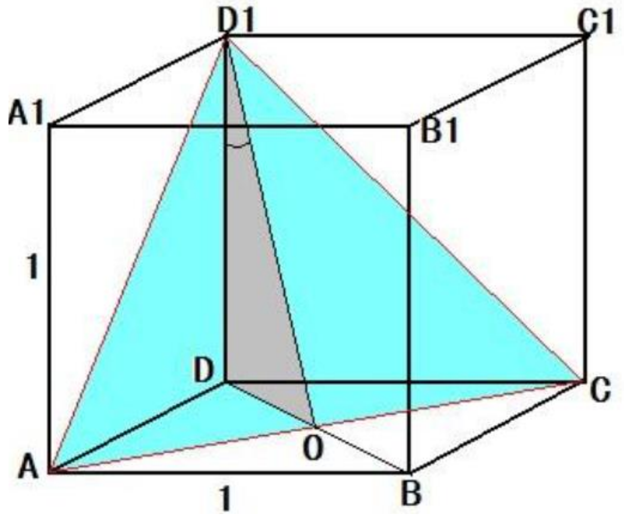
- Each input object is represented as a set of points $\{p_1, p_2, p_3, \ldots\}$, where each $p_i = (x_i, y_i)$ and $x_i, y_i \in$ R. For example, an n-vertex polygon P is represented by a sequence $p_0, p_1, p_2, p_3, \ldots, p_{n-1}$ of its vertices in order of their appearance on the boundary of P .



33 Computational Geometry

- Computational geometry can also be performed in three dimensions, and even in higher-dimensional spaces (an application to database), but such problems and their solutions can be very difficult to visualize.
- Even in two dimensions, however, we can see a good sample of computational-geometry techniques.

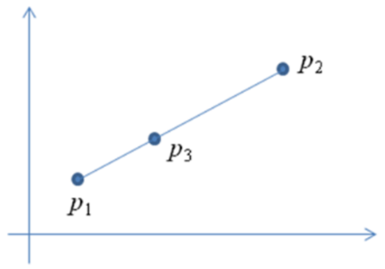




33.1 Line-segment properties

- A ***convex combination***(凸组合) of two distinct points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ is any point $p_3 = (x_3, y_3)$ such that for some α in the range $0 \leq \alpha \leq 1$, we have
  $\boldsymbol{x_3 = \alpha\, x_1 + (1 - \alpha)\, x_2}$ and $\boldsymbol{y_3 = \alpha\, y_1 + (1 - \alpha) y_2}$.
  We also write that $\boldsymbol{p_3 = \alpha\, p_1 + (1 - \alpha)\, p_2}$.

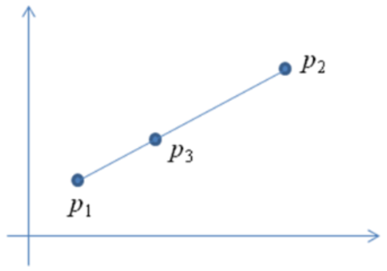
- p_3 is any point that is on the line passing through p_1 and p_2 and is on or between p_1 and p_2 on the line.

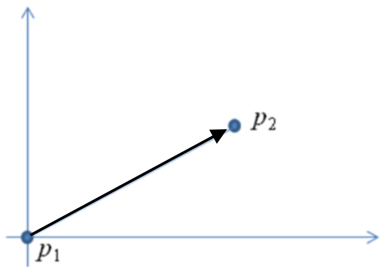


33.1 Line-segment properties

- A ***convex combination***(凸组合) of $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$:
  $\boldsymbol{p_3 = \alpha\, p_1 + (1 - \alpha)\, p_2}$, $(x_3 = \alpha\, x_1 + (1\text{-}\alpha)\, x_2$ and $y_3 = \alpha\, y_1 + (1 - \alpha)y_2$),
  $0 \le \alpha \le 1$.

- Given two distinct points p_1 and p_2, the ***line segment*** is the set of convex combinations of p_1 and p_2 . We call p_1 and p_2 the ***endpoints*** of segment.
- Sometimes the ordering of p_1 and p_2 matters, and we speak of the **directed segment** $\overrightarrow{p_1 p_2}$. If p_1 is the origin (0, 0), then we can treat the directed segment as the **vector** $\boldsymbol{p_2}$ **.**

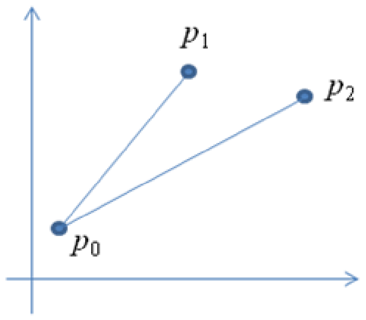


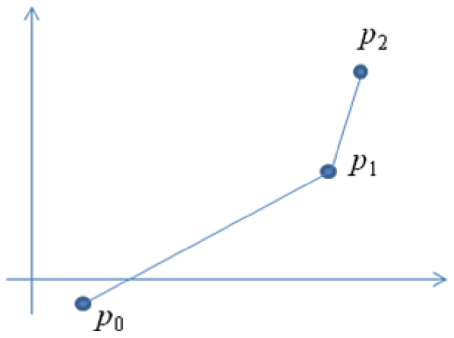


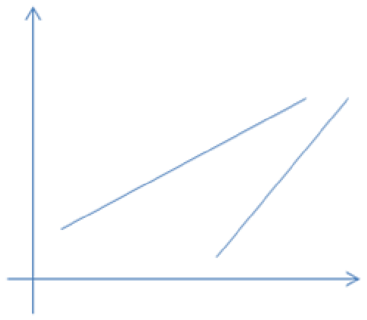
33.1 Line-segment properties

Questions:

1. Given two directed segments $\overrightarrow{p_0p_1}$ and $\overrightarrow{p_0p_2}$, is $\overrightarrow{p_0p_1}$ **clockwise** from $\overrightarrow{p_0p_2}$ with respect to their common endpoint p_0 ?
2. Given two line segments $\overrightarrow{p_0p_1}$ and $\overrightarrow{p_1p_2}$, if we traverse $\overrightarrow{p_0p_1}$ and then $\overrightarrow{p_1p_2}$, do we make a **left turn** at point p_1?
3. Do line segments $\overrightarrow{p_1p_2}$ and $\overrightarrow{p_3p_4}$ **intersect**?



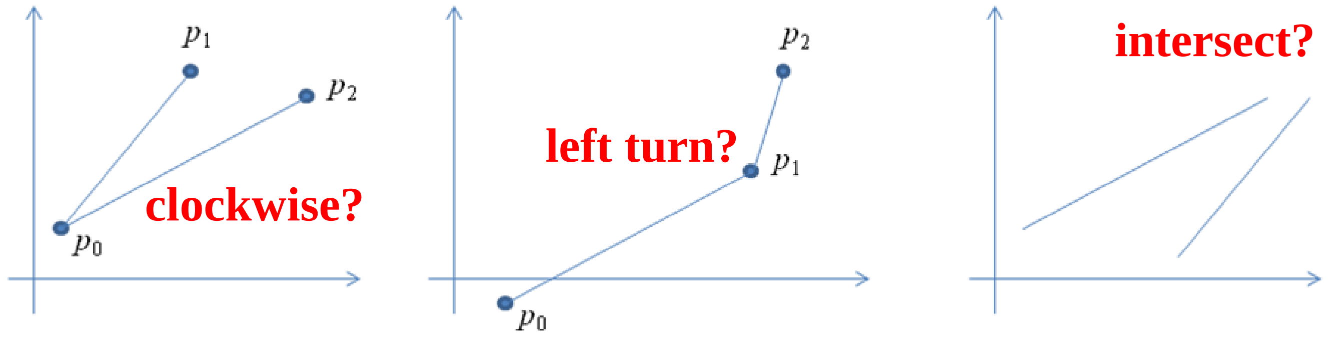




33.1 Line-segment properties

Questions:

1. is $\overrightarrow{p_0p_1}$ clockwise from $\overrightarrow{p_0p_2}$ with respect to their common endpoint p_0?
2. if we traverse $\overrightarrow{p_0p_1}$ and then $\overrightarrow{p_1p_2}$, do we make a left turn at point p_1?
3. Do line segments $\overrightarrow{p_1p_2}$ and $\overrightarrow{p_3p_4}$ intersect?

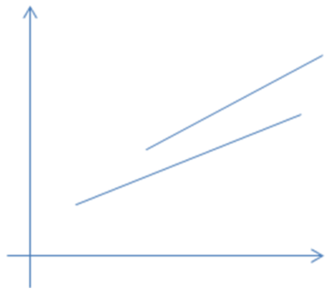


We can answer each question in $O(1)$ time.

Moreover, our methods will use only ***additions* (+)**, ***subtractions* (-)**, ***multiplications* (*)**, and ***comparisons* (==)**. We need **neither division(/) nor trigonometric functions (sin, cos, tan, …)**, both of which can be computationally expensive and prone to problems with round-off error. For example, …

33.1 Line-segment properties

Use only ***additions***, ***subtractions***, ***multiplications***, and ***comparisons***.
Neither division nor trigonometric functions.



For example,

- the "straightforward" method of determining whether two segments **intersect**
  - Compute the line equation of the form *y* = *mx* + *b* for each segment (*m* is the **slope**(斜率) and *b* is the *y*-intercept (*y* 轴截距)),
  - find the point of intersection of the lines (uses division to find the point of intersection),
  - and check whether this point is on both segments.
- When the segments are nearly parallel, this method is very sensitive to the precision of the division operation on real computers.

$$\begin{matrix} y = m_1 x + b_1 \\ y = m_2 x + b_2 \end{matrix} \quad \Rightarrow \quad x = \frac{b_2 - b_1}{m_1 - m_2},\ y = \frac{m_1 b_2 - m_2 b_1}{m_1 - m_2}$$

The method in this section, which avoids division, is much more accurate.

33.1 Line-segment properties- **Cross products**

- Computing cross products is at the **heart** of our line-segment methods.
- Consider vectors p_1 and p_2, define **cross product** $p_1 \times p_2$ as the determinant(行列式) of a matrix

$$\begin{aligned} p_1 \times p_2 &= \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} \\ &= x_1 y_2 - x_2 y_1 \\ &= -p_2 \times p_1 \end{aligned}$$

- Cross products（交叉乘积，叉积）：计算几何的核心运算
- 就像图算法中的几个核心运算：递归, BFS, DFS, Relax edge, …

33.1 Line-segment properties- **Cross products**

cross product

$$
\begin{aligned}
p_1 \times p_2 &= \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} \\
&= x_1 y_2 - x_2 y_1 \\
&= -p_2 \times p_1
\end{aligned}
$$

Equivalently, $p_1 \times p_2$ can be interpreted as the signed area of the parallelogram formed by the four points (0, 0), p_1, p_2, and $p_1 + p_2 = (x_1 + x_2, y_1 + y_2)$? See Figure 33.1(a). (Exercise: try to prove it. Hint: see Figure 33.1.a)

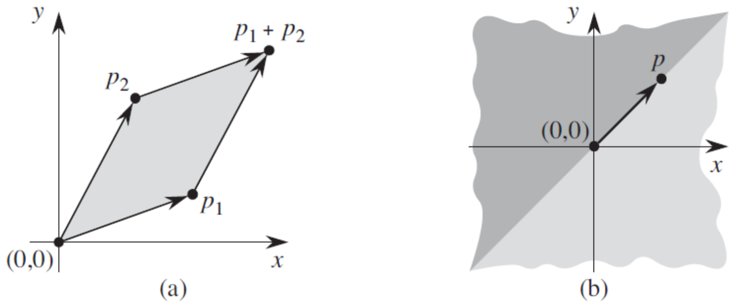
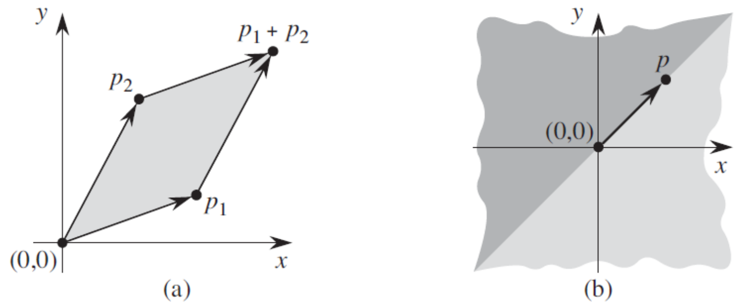


Figure 33.1:
(a) $p_1 \times p_2$ is the signed area of the parallelogram.
(b) The lightly shaded region contains vectors that are clockwise from *p*. The darkly shaded region contains vectors that are counterclockwise from *p*.

33.1 Line-segment properties- **Cross products**

$p_1 \times p_2$ can be interpreted as the signed area of the parallelogram formed by the four points (0, 0), p_1, p_2, and $p_1 + p_2 = (x_1 + x_2, y_1 + y_2)$.



$$
\begin{aligned}
p_1 \times p_2 &= \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} \\
&= x_1 y_2 - x_2 y_1 \\
&= -p_2 \times p_1
\end{aligned}
$$

Figure 33.1:
(a) $p_1 \times p_2$ is the signed area of the parallelogram.
(b) The lightly shaded region contains vectors that are clockwise from p. The darkly shaded region contains vectors that are counterclockwise from p.

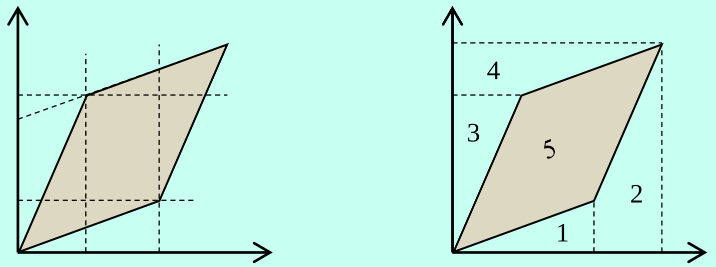


Figure 33.1.a: Hint for proof of cross product

5 = (1+2+3+4+5) - (1+2+3+4)

"1" + "2" + "3" + "4" + "5" = $(x_1+x_2)*(y_1+y_2)$

"1" = $x_1*y_1/2$

"3" = $x_2*y_2/2$

"2" = "3" +x_2*y_1

"4" = "1" +x_2*y_1

33.1 Line-segment properties- **Cross products**

Some characteristics of cross products

- if $p_1 \times p_2$ is positive, then p_1 is clockwise from p_2 with respect to the origin (0, 0).
- if negative, then p_1 is counterclockwise from p_2.
- Figure 33.1(b) shows the clockwise and counterclockwise regions relative to a vector p.
- A boundary condition arises if $p_1 \times p_2$ is 0, in this case, the vectors are ***collinear***(共线), pointing in either the same or opposite directions.

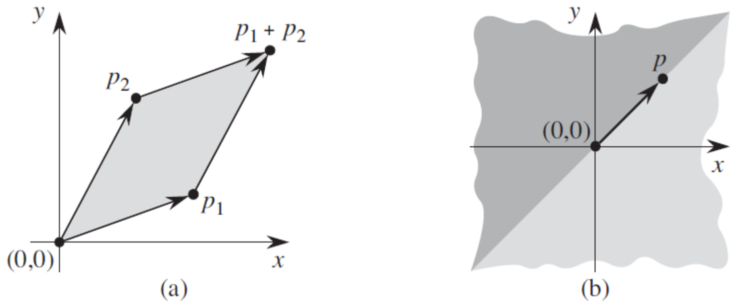
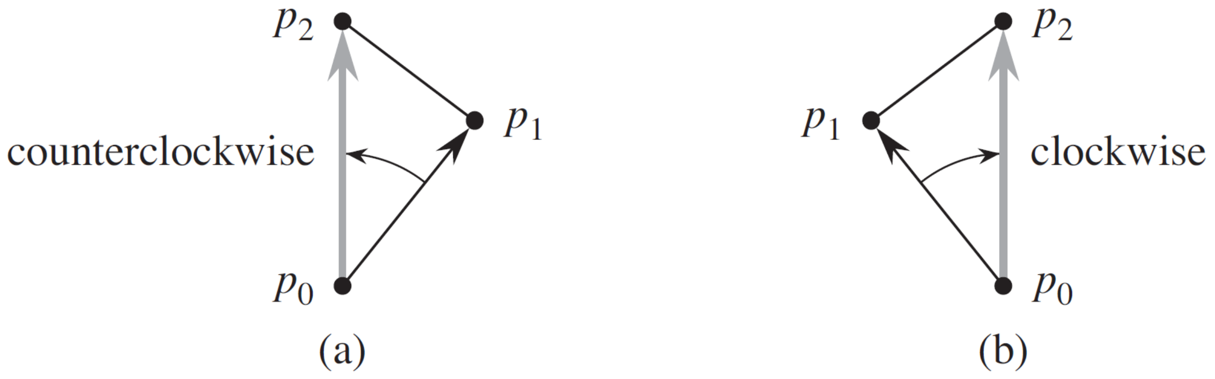


Figure 33.1: (a) $p_1 \times p_2$ is the signed area of the parallelogram.
(b) The lightly shaded region contains vectors that are clockwise from p. The darkly shaded region contains vectors that are counterclockwise from p.

33.1 Line-segment properties- **Cross products**

1. Determine whether $\overrightarrow{p_0 p_1}$ is clockwise from $\overrightarrow{p_0 p_2}$ with respect to their common endpoint p_0

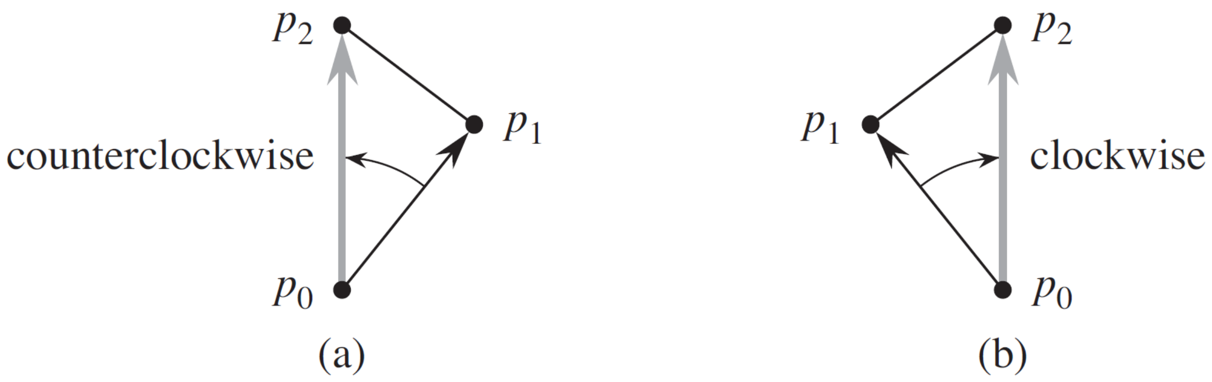
- Simply translate to use p_0 as the origin, compute the cross product $(p_1 - p_0) \times (p_2 - p_0) = (x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)$
- If positive, clockwise; else, counterclockwise.



33.1 Line-segment properties- **Cross products**

2. Determining whether consecutive segments turn left or right

- Whether two consecutive line segments $\overrightarrow{p_0 p_1}$ and $\overrightarrow{p_1 p_2}$ turn left or right at point p_1.
- Equivalently, determine which way a given angle $\angle p_0 p_1 p_2$ turns.
- Cross products allow us to answer this question without computing the angle. We simply check whether $\overrightarrow{p_0 p_2}$ is clockwise or counterclockwise relative to $\overrightarrow{p_0 p_1}$ **(**See Figure 33.2**)**.



33.1 Line-segment properties- **Cross products**

2. Determining whether consecutive segments turn left or right

To do this, we compute the cross product $(p_2 - p_0) \times (p_1 - p_0)$

- If negative, $\overrightarrow{p_0 p_2}$ is counterclockwise from $\overrightarrow{p_0 p_1}$, and thus we make a left turn at p_1.
- A positive cross product indicates a clockwise orientation and a right turn.
- A cross product of 0 means that points p_0 , p_1 , and p_2 are collinear.

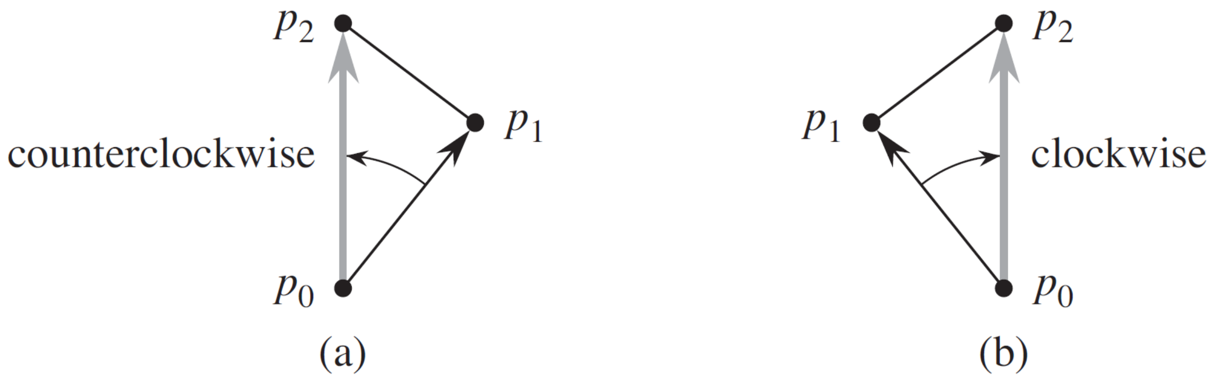


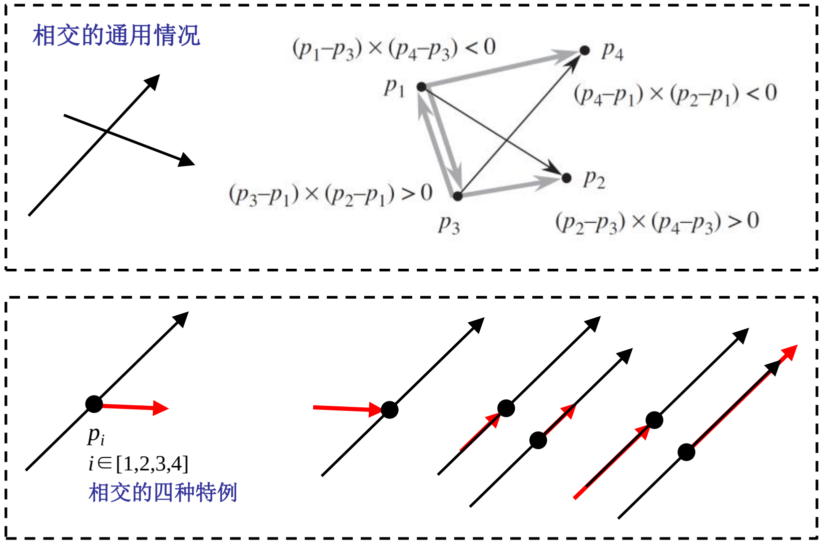
Fig 33.2: Using the cross product to determine how consecutive line segments $\overrightarrow{p_0 p_1}$ and $\overrightarrow{p_1 p_2}$ turn at point p_1. We check whether $\overrightarrow{p_0 p_2}$ is clockwise or counterclockwise relative to $\overrightarrow{p_0 p_1}$. (a) If counterclockwise, the point makes a left turn. (b) If clockwise, a right turn.

33.1 Line-segment properties- **Cross products**

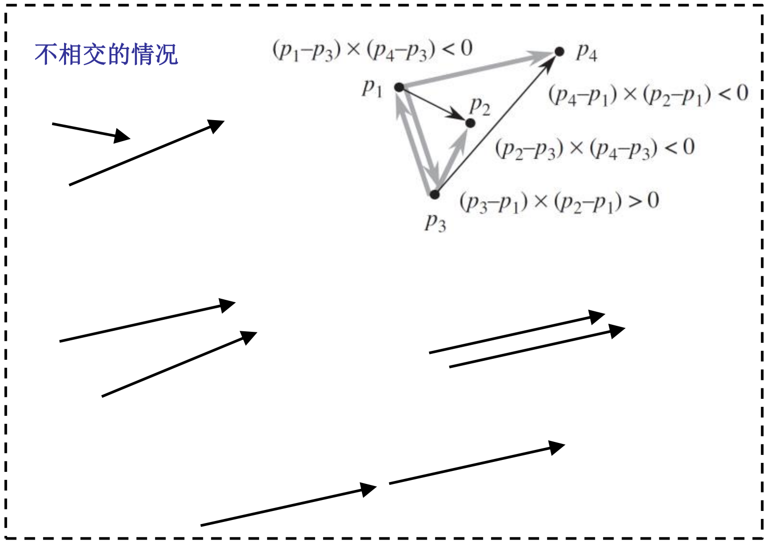
3. Determining whether two line segments intersect

By checking whether each segment **straddles**(横跨) the line containing the other. 相交的情况有5种：相互横跨；or某一个端点 p_i 在另一个线段上（共4个端点，4种情况，$i\in[1,2,3,4]$）

- **Straddles:** A segment $\overrightarrow{p_1p_2}$ straddles a line if point p_1 lies on one side of the line and point p_2 lies on the other side.
- *A boundary case* arises if p_1 or p_2 lies directly on the line.



相交的通用情况
$(p_1-p_3)\times(p_4-p_3)<0$
p_4
p_1
$(p_4-p_1)\times(p_2-p_1)<0$
p_2
$(p_3-p_1)\times(p_2-p_1)>0$
p_3
$(p_2-p_3)\times(p_4-p_3)>0$
p_i
$i\in[1,2,3,4]$
相交的四种特例



不相交的情况
$(p_1-p_3)\times(p_4-p_3)<0$
p_4
p_1
p_2
$(p_4-p_1)\times(p_2-p_1)<0$
$(p_2-p_3)\times(p_4-p_3)<0$
$(p_3-p_1)\times(p_2-p_1)>0$
p_3

Figure 33.3: Cases in the procedure SEGMENTS-INTERSECT

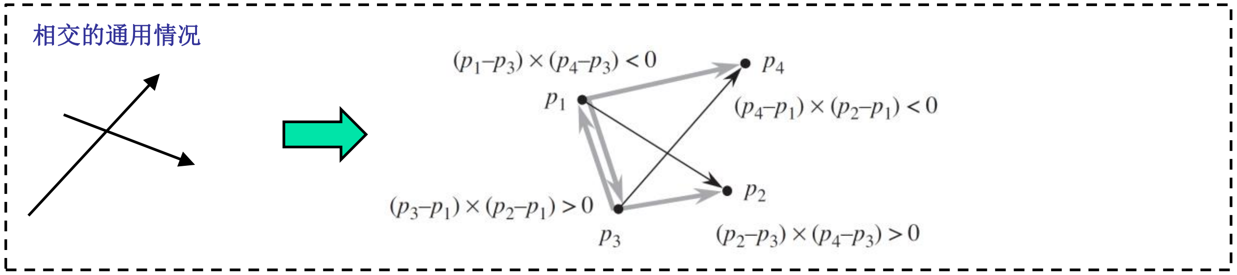
33.1 Line-segment properties- **Cross products**

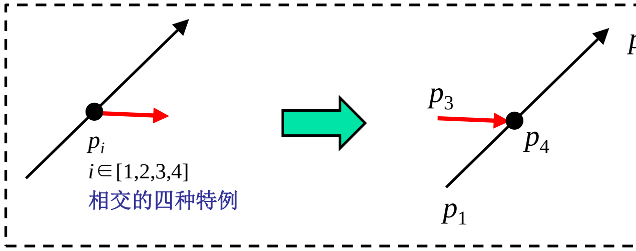
3. Determining whether two line segments intersect

Two line segments intersect if and only if either of the following conditions holds:

① Each segment straddles the line containing the other.

② An endpoint of one segment lies on the other segment.
(This condition comes from the boundary case.)


相交的通用情况
$(p_1-p_3)\times(p_4-p_3)<0$
p_4
p_1
$(p_4-p_1)\times(p_2-p_1)<0$
$(p_3-p_1)\times(p_2-p_1)>0$
p_2
p_3
$(p_2-p_3)\times(p_4-p_3)>0$



$(p_4 - p_1) \times (p_2 - p_1) == 0$

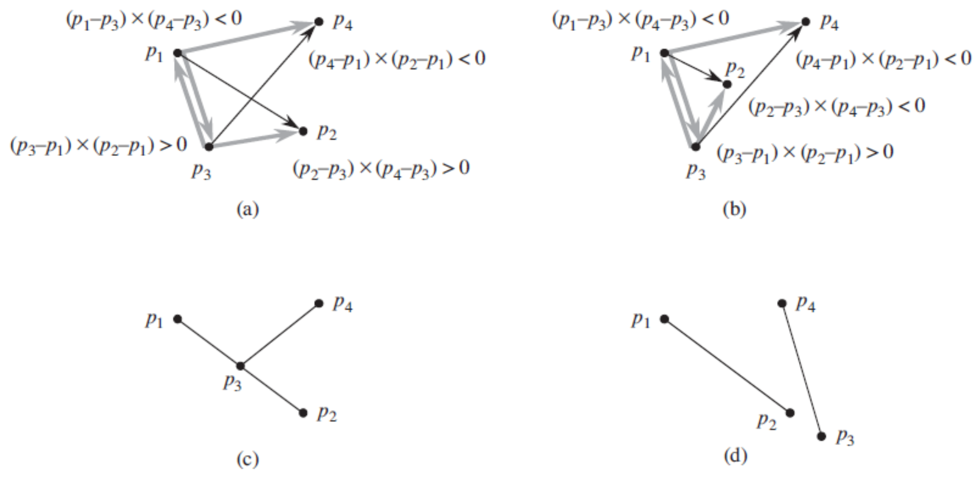
Then, if p_4 lies on $\overrightarrow{p_1p_2}$?

Figure 33.3: Cases in the procedure SEGMENTS-INTERSECT

33.1 Line-segment properties- **Cross products**

SEGMENTS-INTERSECT returns TRUE if segments $\overrightarrow{p_1p_2}$ and $\overrightarrow{p_3p_4}$ intersect. Return FALSE if not.

- Subroutine DIRECTION: computes relative orientations using the cross product.
- ON-SEGMENT: determines whether a point known to be collinear with a segment lies on that segment.



$(p_1-p_3)\times(p_4-p_3)<0$
p_4
p_1
$(p_4-p_1)\times(p_2-p_1)<0$
p_2
$(p_3-p_1)\times(p_2-p_1)>0$
p_3
$(p_2-p_3)\times(p_4-p_3)>0$
(a)
$(p_1-p_3)\times(p_4-p_3)<0$
p_4
p_1
p_2
$(p_4-p_1)\times(p_2-p_1)<0$
$(p_2-p_3)\times(p_4-p_3)<0$
$(p_3-p_1)\times(p_2-p_1)>0$
p_3
(b)
p_1
p_4
p_3
p_2
(c)
p_1
p_4
p_2
p_3
(d)

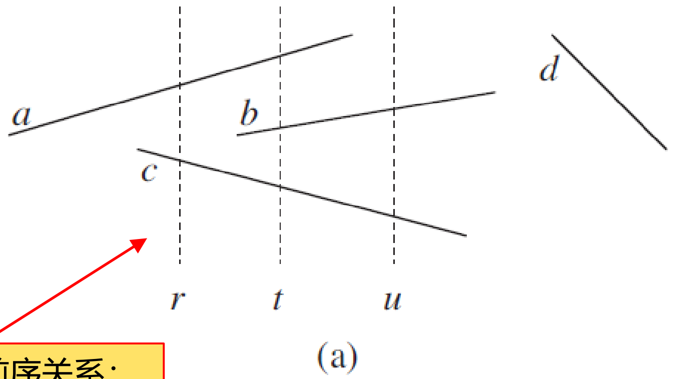
Figure 33.3: Cases in the procedure SEGMENTS-INTERSECT

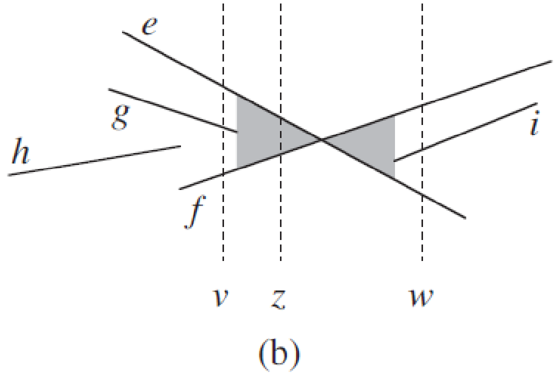
```
SEGMENTS-INTERSECT($p_1$, $p_2$, $p_3$, $p_4$)
$d_1$ ← DIRECTION($p_3$, $p_4$, $p_1$) // 若为0，$p_1$与线段 $p_3p_4$ 共线
$d_2$ ← DIRECTION($p_3$, $p_4$, $p_2$)
$d_3$ ← DIRECTION($p_1$, $p_2$, $p_3$)
$d_4$ ← DIRECTION($p_1$, $p_2$, $p_4$)
if (($d_1 > 0$ and $d_2 < 0$) or ($d_1 < 0$ and $d_2 > 0$)) and
(($d_3 > 0$ and $d_4 < 0$) or ($d_3 < 0$ and $d_4 > 0$)) // $d_1*d_2 < 0$ and $d_3*d_4 < 0$
    return TRUE
else if $d_1 == 0$ and ON-SEGMENT($p_3$, $p_4$, $p_1$)
    return TRUE
if $d_2 == 0$ and ON-SEGMENT($p_3$, $p_4$, $p_2$)
    return TRUE
else if $d_3 == 0$ and ON-SEGMENT($p_1$, $p_2$, $p_3$)
    return TRUE
else if $d_4 == 0$ and ON-SEGMENT($p_1$, $p_2$, $p_4$)
    return TRUE
else
    return FALSE
```

```
DIRECTION($p_i$, $p_j$, $p_k$)
return $(p_k - p_i) \times (p_j - p_i)$
```

```
ON-SEGMENT($p_i$, $p_j$, $p_k$)
if $\min(x_i, x_j) \le x_k \le \max(x_i, x_j)$ and
$\min(y_i, y_j) \le y_k \le \max(y_i, y_j)$
    return TRUE
else
    return FALSE
```

*33.2 Determining whether any pair of segments intersects

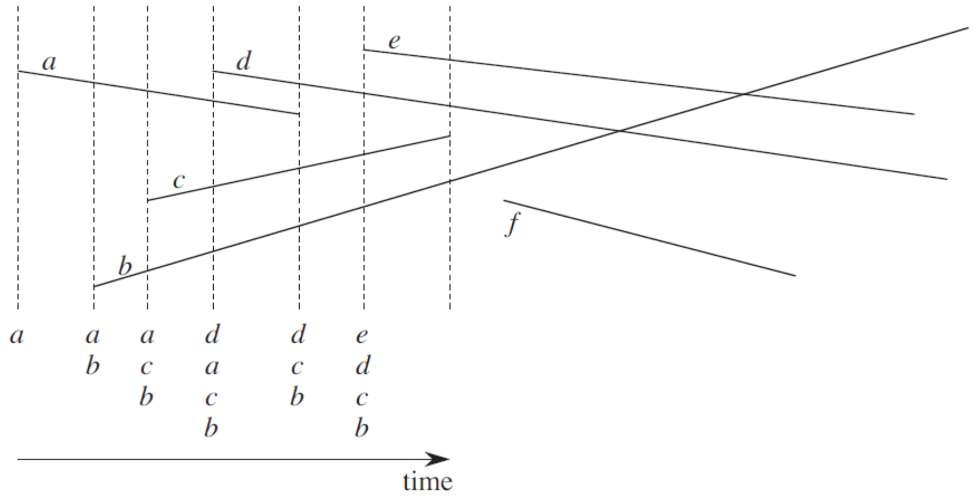




扫描线处的前序关系：
$a >_r c$

The algorithm uses a technique known as "sweeping"

- 先对线段的顶点进行排序
- 然后用sweeping技术，依序把"线段（顶点）"加入表 T 中或从表 T 中删除



*33.2 Determining whether any pair of segments intersects

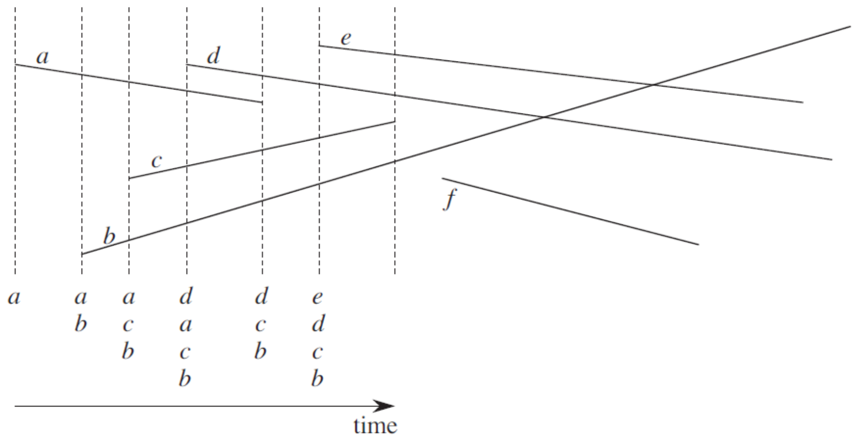
```
ANY-SEGMENTS-INTERSECT(S)
1   T = Ø   // 维护一个有顺序关系的顶点集合 T
2   sort the endpoints of the segments in S from left to right,
        breaking ties by putting left endpoints before right endpoints
        and breaking further ties by putting points with lower
        y-coordinates first
3   for each point p in the sorted list of endpoints
4       if p is the left endpoint of a segment s
5           INSERT(T, s)
6           if (ABOVE(T, s) exists and intersects s)
                or (BELOW(T, s) exists and intersects s)
7               return TRUE
8       if p is the right endpoint of a segment s
9           if both ABOVE(T, s) and BELOW(T, s) exist
                and ABOVE(T, s) intersects BELOW(T, s)
10              return TRUE
11          DELETE(T, s)
12  return FALSE
```

- 先对线段（顶点）进行排序，O($n$lg$n$)
- 然后sweeping，O(n)

Insert(T, s), Delete(T, s),
维护 T 里顶点的顺序关系：
【chap13, 红黑树，O(lg$n$)】*

O($n$lg$n$)

Running Time?



*33.2 Determining whether any pair of segments intersects

```
ANY-SEGMENTS-INTERSECT(S)
1   T = ∅    // 维护一个有前序关系的顶点集合 T
2   sort the endpoints of the segments in S from left to right,
        breaking ties by putting left endpoints before right endpoints
        and breaking further ties by putting points with lower
        y-coordinates first
3   for each point p in the sorted list of endpoints
4       if p is the left endpoint of a segment s
5           INSERT(T, s)
6           if (ABOVE(T, s) exists and intersects s)
                or (BELOW(T, s) exists and intersects s)
7               return TRUE
8       if p is the right endpoint of a segment s
9           if both ABOVE(T, s) and BELOW(T, s) exist
                and ABOVE(T, s) intersects BELOW(T, s)
10              return TRUE
11          DELETE(T, s)
12  return FALSE
```

定理：算法正确（找到交点时返回ture），当且仅当(if and only if) 有交点

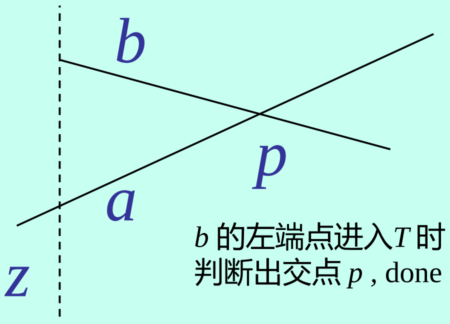
思路：

⇨ : 算法返回true，有交点。显然。

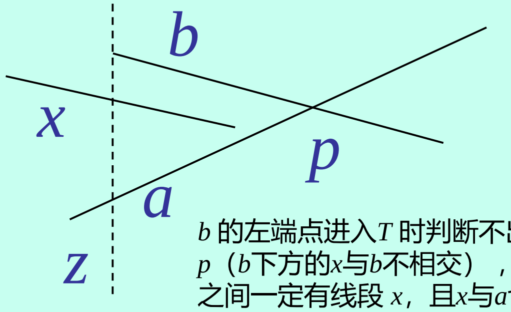
⇦ : **有交点，一定会被判断出来（会被发现）**，因此，算法返回ture。

Correctness?

设只有一个交点 *p*



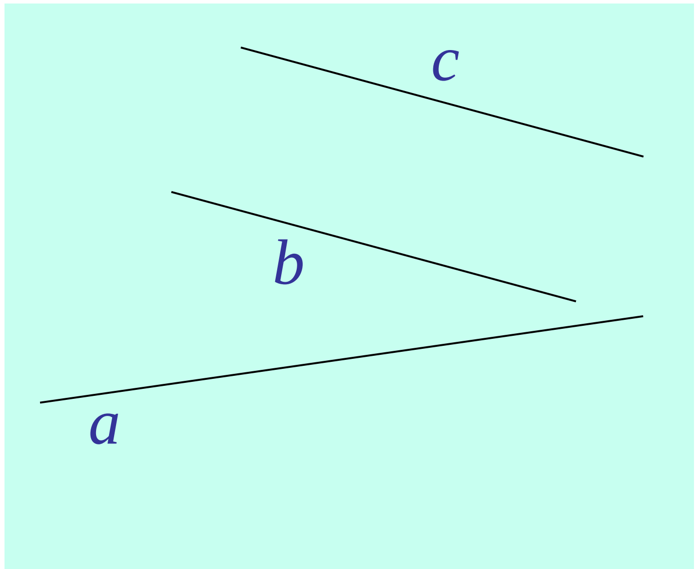
b 的左端点进入*T* 时判断出交点 *p* , done



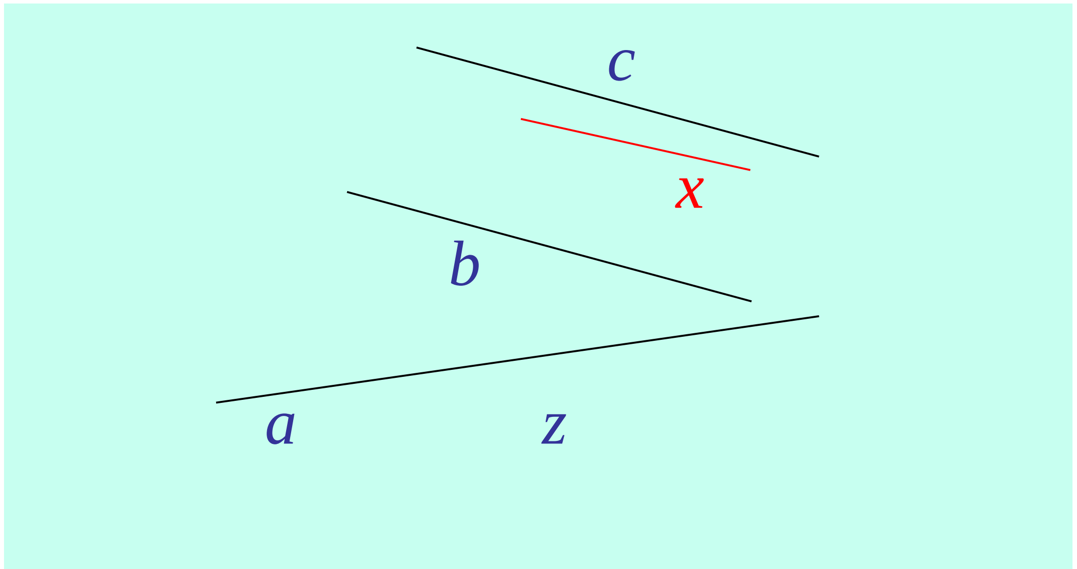
b 的左端点进入*T* 时判断不出交点 *p*（*b*下方的*x*与*b*不相交），则*b*和*a*之间一定有线段 *x*，且*x*与*a*也不相交（因为假设只有一个交点，或 *b* 加入前交点就被判断出来了）

*33.2 Determining whether any pair of segments intersects

思考：维护 T 里顶点的前序关系：如何判断一条新加入线段 x 在 T 中的哪个位置（x 的左端点 x_L 在哪条线段上方，在哪条线段下方）（x 加入前，没有线段相交）？



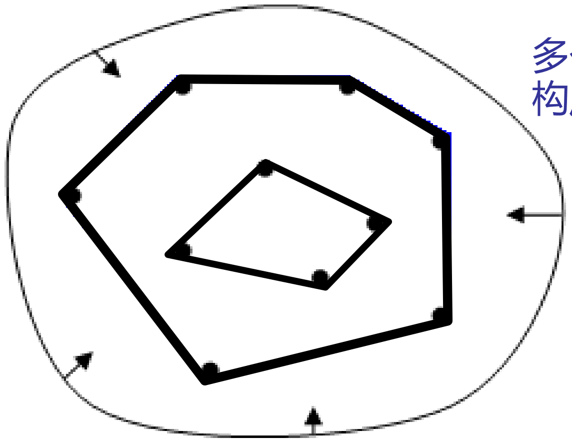
T 中：已知 $a < b < c$



33.3 Finding the convex hull

CH(Q), the ***convex hull*** (凸包) of a set Q of points, is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior.

Intuitively, we can think of each point in Q as being a nail sticking out from a board. The convex hull is then the shape formed by a tight rubber band that surrounds all the nails (See Figure 33.6).



多个凸包
构成凸层

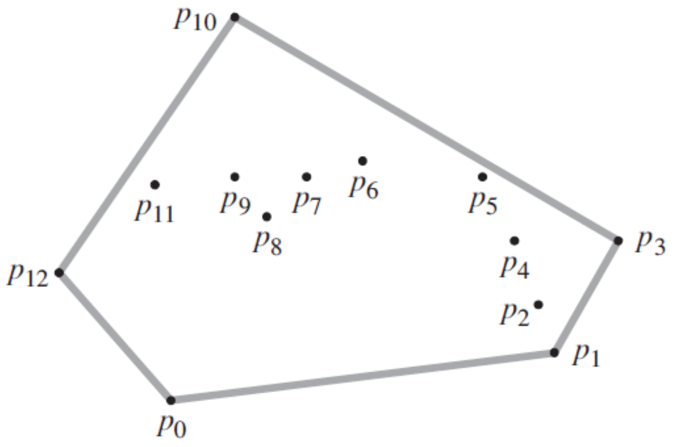
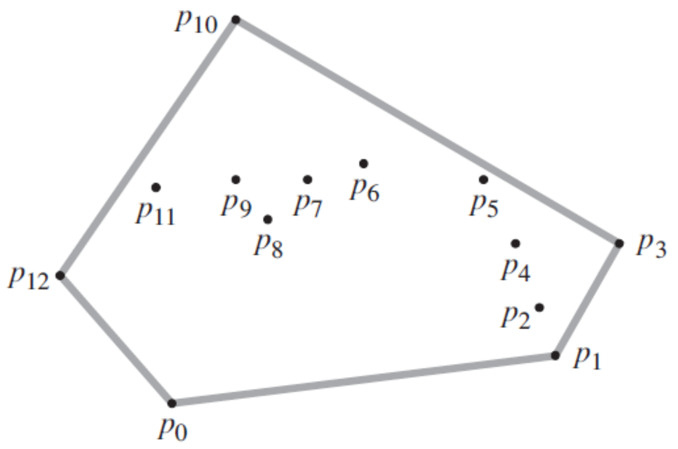


Figure 33.6: A set of points $Q = \{p_0, p_1, ..., p_{12}\}$ with its convex hull CH(Q) in gray

33.3 Finding the convex hull

Computing the convex hull of a set of points is an interesting problem in its own right.

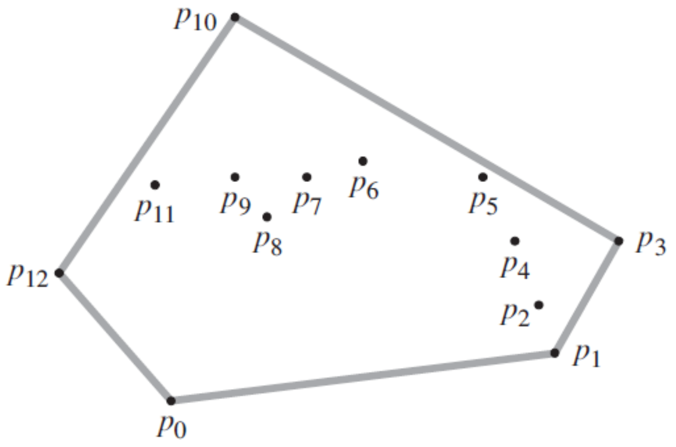


Moreover, algorithms for some other computational-geometry problems start by computing a convex hull. Consider, for example, the two-dimensional ***farthest-pair problem*** (Exer 33.3-3). (Exercises: 33.3-3 Given a set of points Q, prove that the pair of points farthest from each other must be vertices of CH(*Q*))

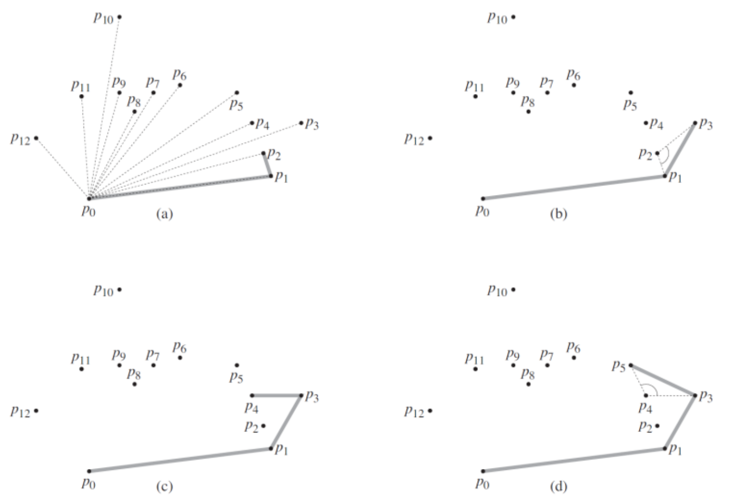
33.3 Finding the convex hull

Some algorithms that compute the convex hull of a set of *n* points:

- Graham's scan (格雷厄姆), runs in $O(n\lg n)$ time
- Jarvis's march (贾维斯), runs in $O(nh)$ time, where *h* is the number of vertices of the convex hull.
- Additional several methods
  - *incremental method*, $O(n\lg n)$
  - *divide-and-conquer method*, $O(n\lg n$
  - *prune-and-search method*, $O(n\lg h)$

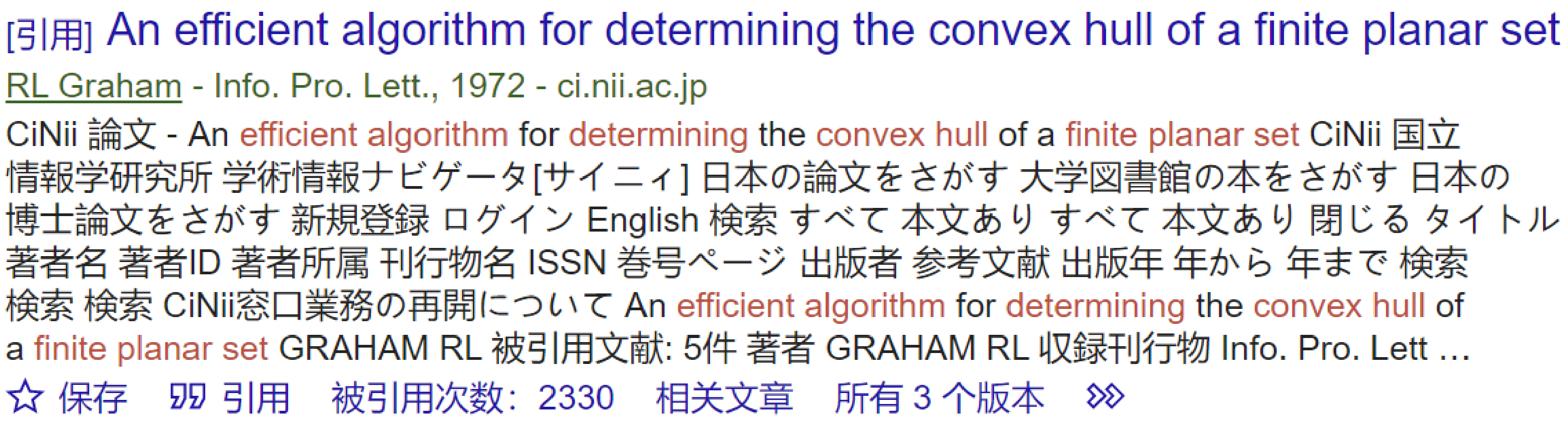


33.3 Finding the convex hull--**Graham's scan**



Information Processing Letters,

1(4): 132-133, 1972



[引用] An efficient algorithm for determining the convex hull of a finite planar set

RL Graham - Info. Pro. Lett., 1972 - ci.nii.ac.jp

CiNii 論文 - An efficient algorithm for determining the convex hull of a finite planar set CiNii 国立情報学研究所 学術情報ナビゲータ[サイニィ] 日本の論文をさがす 大学図書館の本をさがす 日本の博士論文をさがす 新規登録 ログイン English 検索 すべて 本文あり すべて 本文あり 閉じる タイトル 著者名 著者ID 著者所属 刊行物名 ISSN 巻号ページ 出版者 参考文献 出版年 年から 年まで 検索 検索 検索 CiNii窓口業務の再開について An efficient algorithm for determining the convex hull of a finite planar set GRAHAM RL 被引用文献: 5件 著者 GRAHAM RL 収録刊行物 Info. Pro. Lett …

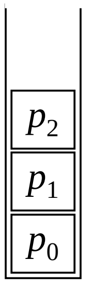
☆ 保存 99 引用 被引用次数：2330 相关文章 所有 3 个版本 »

33.3 Finding the convex hull--**Graham's scan**

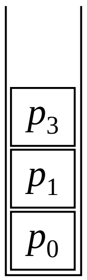
By maintaining a **stack *S*** of candidate points **consecutive segments turn left or right**

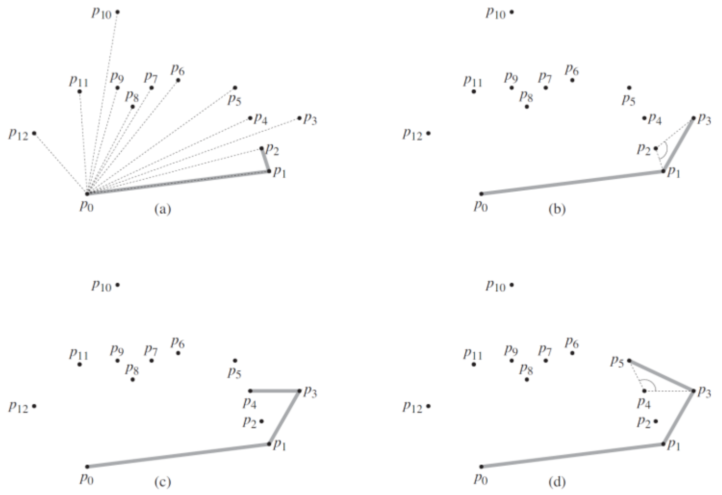
- Each point of the input set *Q* is pushed once onto the stack.
- The points that are not vertices of CH(*Q*) are eventually popped from the stack
- When the algorithm terminates, stack *S* contains exactly the vertices of CH(*Q*), in counterclockwise order of their appearance on the boundary.

```
GRAHAM-SCAN($Q$)
1  let $p_0$ be the point in $Q$ with the minimum $y$-coordinate,
       or the leftmost such point in case of a tie
2  let $<p_1, p_2, ..., p_m>$ be the remaining points in $Q$,
       sorted by polar angle in counterclockwise order around
       $p_0$ (if more than one point has the same angle, remove all
       but the one that is farthest from $p_0$ )
3  PUSH($p_0$, $S$),   PUSH($p_1$, $S$),   PUSH($p_2$, $S$)
4  for $i \leftarrow 3$ to $m$
5       while ( the consecutive segments formed by points
                NEXT-TO-TOP($S$), TOP($S$), and $p_i$  make a nonleft
                turn) // 直线，栈顶点在凸包边上；右转，点在凸包内
6            POP($S$)
7       PUSH($p_i$, $S$) // $p_i$  可能为凸包顶点，入栈
8  return $S$
```



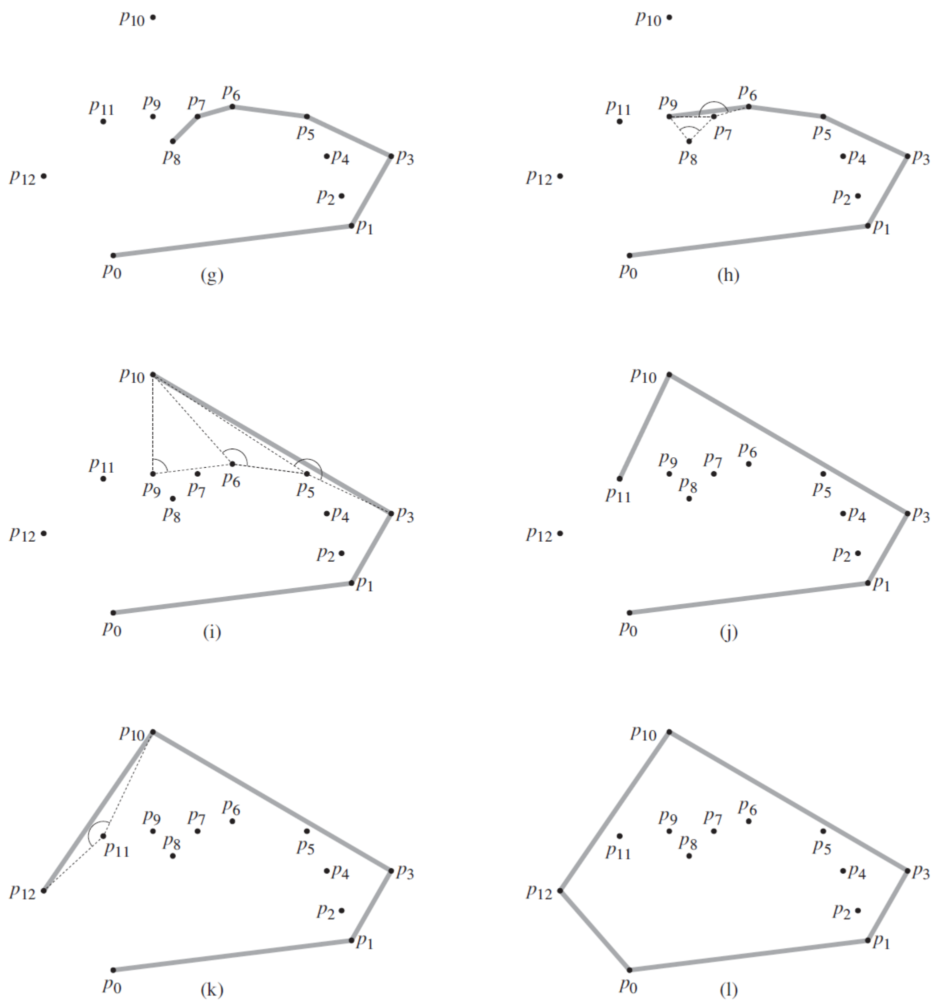
stack *S*





33.3 Finding the convex hull--**Graham's scan**

```
GRAHAM-SCAN($Q$)
1  let $p_0$ be the point in $Q$ with the minimum $y$-coordinate,
       or the leftmost such point in case of a tie
2  let $<p_1, p_2, ..., p_m>$ be the remaining points in $Q$,
       sorted by polar angle in counterclockwise order
       around $p_0$ (if more than one point has the same angle,
       remove all but the one that is farthest from $p_0$ )
3  PUSH($p_0$, $S$),   PUSH($p_1$, $S$),   PUSH($p_2$, $S$)
4  for $i \leftarrow 3$ to $m$
5       while ( the consecutive segments formed by points
              NEXT-TO-TOP($S$), TOP($S$), and $p_i$ make a
              nonleft turn)
              // 直线，栈顶点在凸包边上；右转，点在凸包内
6           POP($S$)
7       PUSH($p_i$, $S$) // $p_i$ 可能为凸包顶点，入栈
8  return $S$
```



33.3 Finding the convex hull--**Graham's scan**

```
GRAHAM-SCAN(Q)
1  let $p_0$ be the point in $Q$ with the minimum $y$-coordinate,
       or the leftmost such point in case of a tie
2  let $<p_1, p_2, ..., p_m>$ be the remaining points in $Q$,
       sorted by polar angle in counterclockwise order around
       $p_0$ (if more than one point has the same angle, remove all
       but the one that is farthest from $p_0$ )
3  PUSH($p_0$, $S$),  PUSH($p_1$, $S$),  PUSH($p_2$, $S$)
4  for $i \leftarrow 3$ to $m$
5       while ( the consecutive segments formed by points
               NEXT-TO-TOP($S$), TOP($S$), and $p_i$ make a nonleft
               turn) // 直线，栈顶点在凸包边上；右转，点在凸包内
6            POP($S$)
7       PUSH($p_i$, $S$) // $p_i$ 可能为凸包顶点，入栈
8  return $S$
```

T(n) = ?

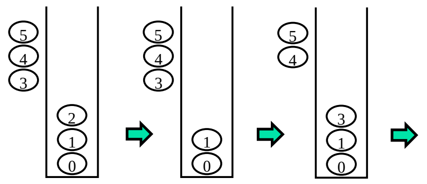
$\Theta(n)$

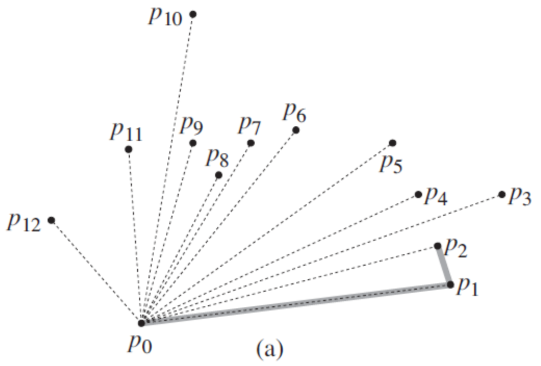
$O(n\lg n)$, using merge sort and the cross-product method to compare angles ?

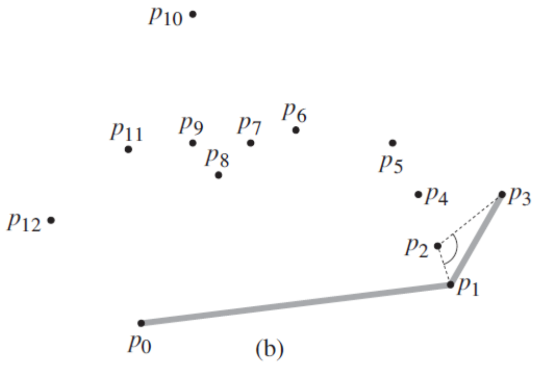
$O(1)$

$O(n-3)$

Aggregate analysis: **while** loop takes $O(n)$ time overall. For $i = 0, 1, ..., m$, each point p_i is pushed onto stack S exactly once, there is at most one POP operation for each PUSH operation. At least three points p_0, p_1, and p_m are never popped from the stack, so that in fact at most (m - 2) POP operations are performed in total?



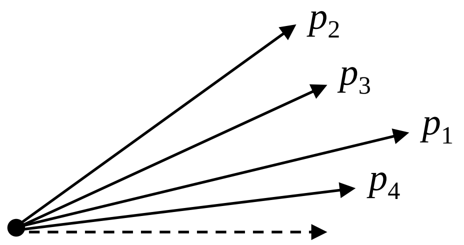




33.3 Finding the convex hull--**Graham's scan**

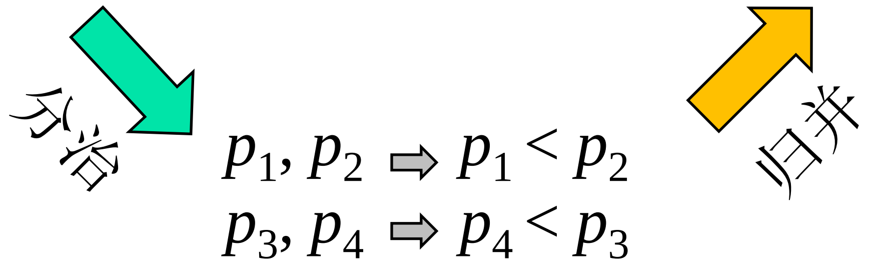
```
GRAHAM-SCAN($Q$)
1  let $p_0$ be the point in $Q$ with the minimum $y$-coordinate,
       or the leftmost such point in case of a tie
2  let <$p_1, p_2, \ldots, p_m$> be the remaining points in $Q$,
       sorted by polar angle in counterclockwise order around
       $p_0$ (if more than one point has the same angle, remove all
       but the one that is farthest from $p_0$)
```

$O(n\lg n)$, using merge sort and the cross-product method to compare angles ?



按极角序

Input: p_1, p_2, p_3, p_4 Output: $p_4 < p_1 < p_3 < p_2$



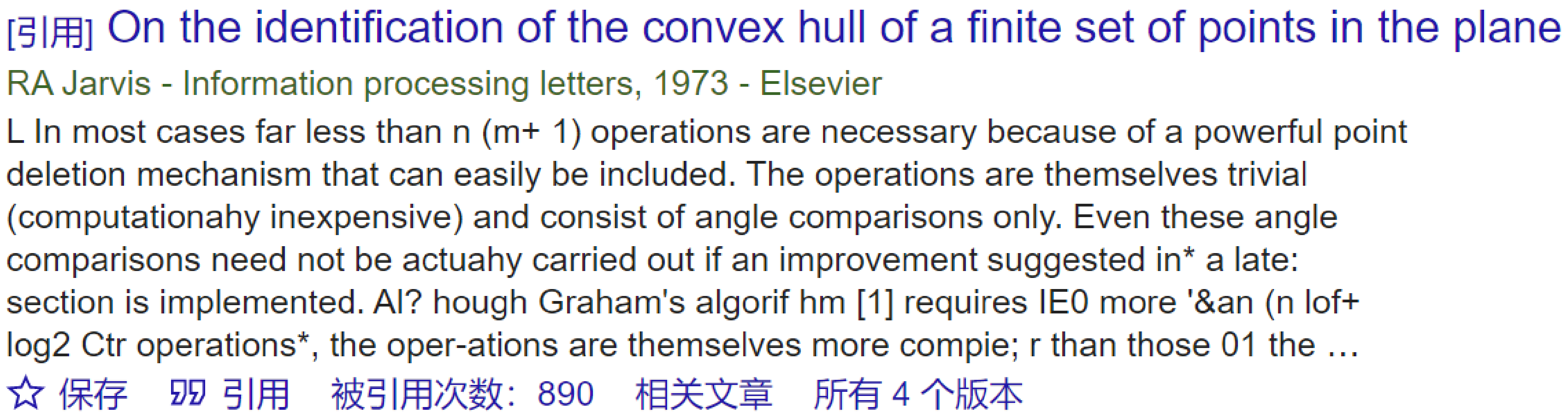
$p_1, p_2 \Rightarrow p_1 < p_2$

$p_3, p_4 \Rightarrow p_4 < p_3$

33.3 Finding the convex hull--**Jarvis's march**

Information Processing Letters,

2(1): 18-21, 1973



[引用] On the identification of the convex hull of a finite set of points in the plane

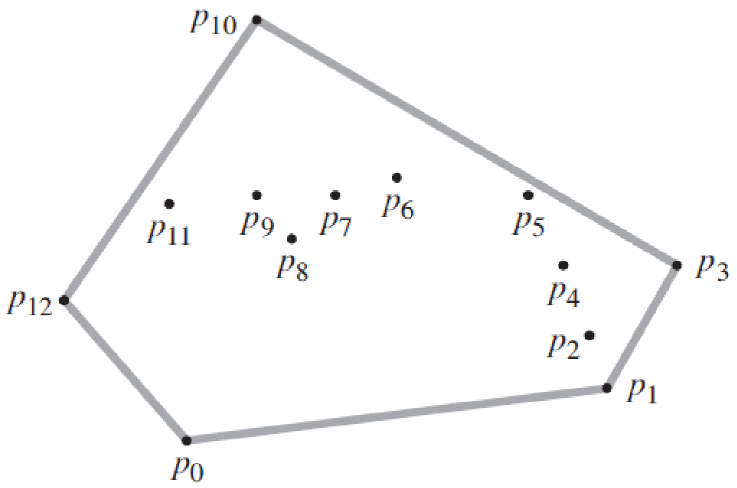
RA Jarvis - Information processing letters, 1973 - Elsevier

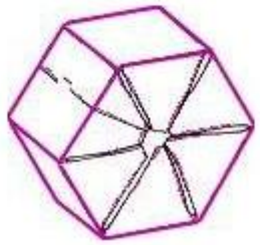
L In most cases far less than n (m+ 1) operations are necessary because of a powerful point deletion mechanism that can easily be included. The operations are themselves trivial (computationahy inexpensive) and consist of angle comparisons only. Even these angle comparisons need not be actuahy carried out if an improvement suggested in* a late: section is implemented. Al? hough Graham's algorif hm [1] requires IE0 more '&an (n lof+ log2 Ctr operations*, the oper-ations are themselves more compie; r than those 01 the …

☆ 保存　99 引用　被引用次数：890　相关文章　所有 4 个版本

33.3 Finding the convex hull--**Jarvis's march**

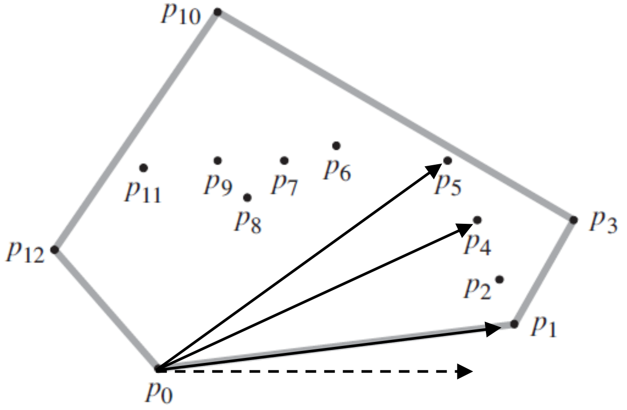
- ***Jarvis's march*** computes the convex hull of a set Q of points by a technique known as ***package wrapping*** (or ***gift wrapping***).
- The algorithm runs in time $O(nh)$, where h is the number of vertices of CH(Q).
- Is Jarvis's march asymptotically faster than Graham's scan, whose running time is $O(n\lg n)$?





33.3 Finding the convex hull--**Jarvis's march**

- Intuitively, Jarvis's march simulates wrapping a taut piece of paper around the set *Q*.
  - We start by taping the end of the paper to the lowest point in the set, that is, to the same point p_0 with which we start Graham's scan. This point is a vertex of the convex hull.
  - We pull the paper to the right to make it taut, and then we pull it higher until it touches a point. This point must also be a vertex of the convex hull.
  - Keeping the paper taut, we continue in this way around the set of vertices until we come back to our original point p_0.
- Jarvis's march has a running time of $O(nh)$?



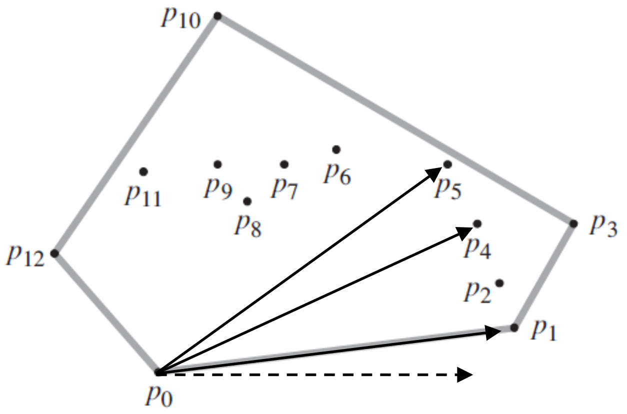
33.3 Finding the convex hull--**Jarvis's march**

- Intuitively, Jarvis's march simulates wrapping a taut piece of paper around the set Q.
- Jarvis's march has a running time of $O(nh)$?

p_0为端点： $p_0p'_1$ $p_0p'_2$ … $p_0p'_n$ ，找出极坐标角最小的点p_1 (CH point) $O(n)$
p_1为端点： $p_1p'_1$ $p_1p'_2$ … $p_1p'_n$ ，找出极坐标角最小的点p_i (CH point) $O(n)$
…

共 h 个CH point

如何求极坐标角最小?



33.3 Finding the convex hull--**Jarvis's march**

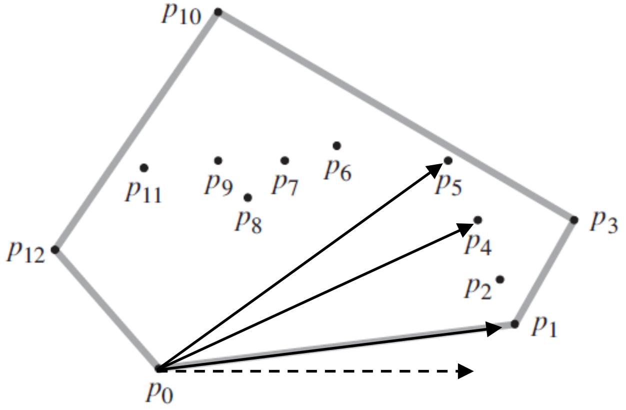
- Intuitively, Jarvis's march simulates wrapping a taut piece of paper around the set *Q*.
- Jarvis's march has a running time of $O(nh)$?

p_0为端点： $p_0p'_1$ $p_0p'_2$ … $p_0p'_n$ ，找出极坐标角最小的点p_1 (CH point) $O(n)$
p_1为端点： $p_1p'_1$ $p_1p'_2$ … $p_1p'_n$ ，找出极坐标角最小的点p_i (CH point) $O(n)$
…

共 h 个CH point

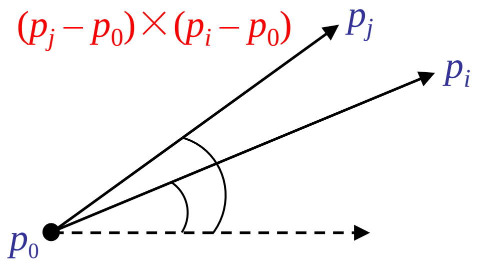
如何求极坐标角最小?

叉积

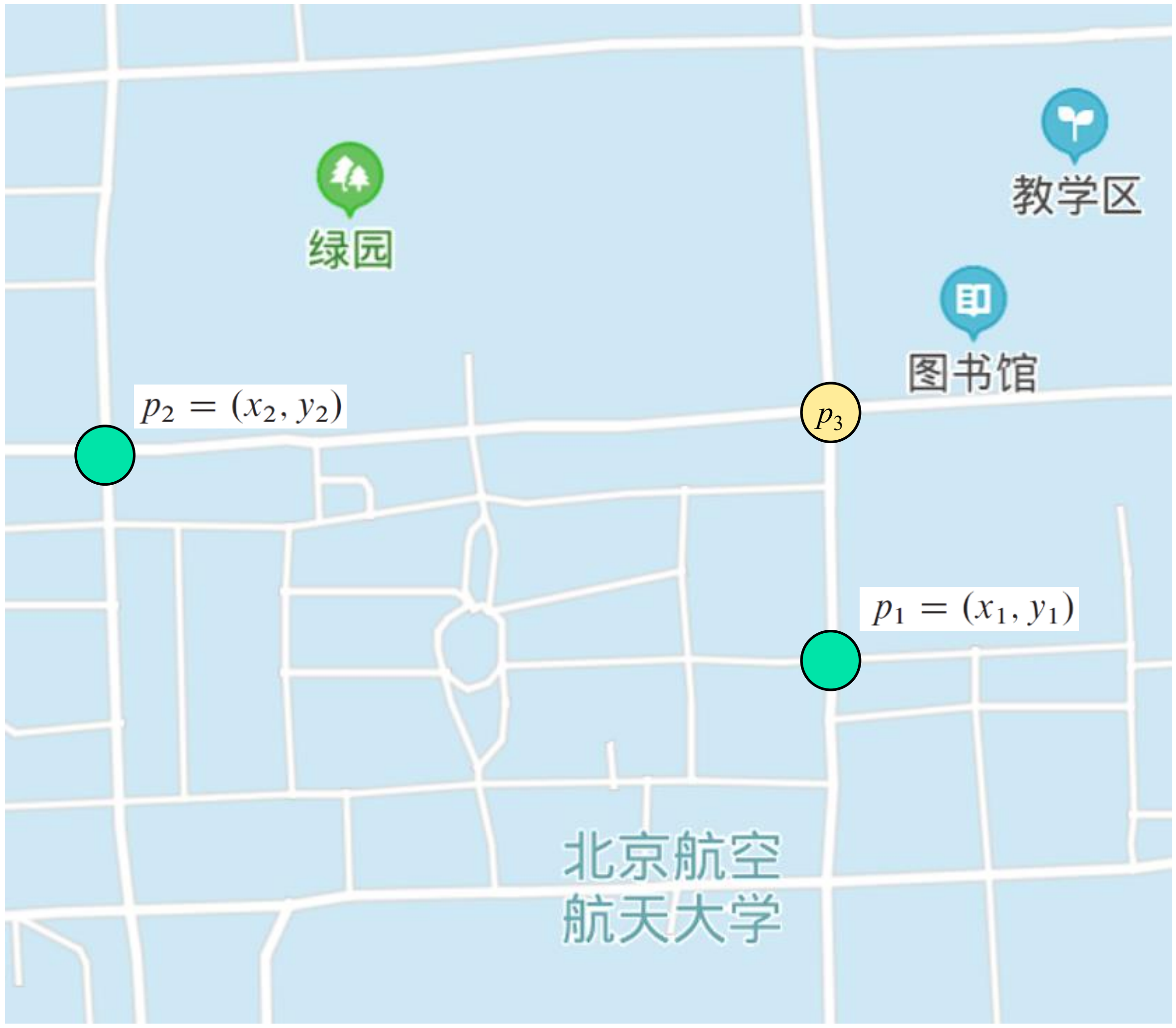


p_0p_j和 p_0p_i，哪个向量的极坐标角小（p_0为原点）？

$(p_j - p_0) \times (p_i - p_0)$



*33.4 Finding the closest pair of points



Euclidean distance

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

L_m-distance (Minkowski distance)

$(|x_1-x_2|^m + |y_1-y_2|^m)^{1/m}$

L_1 : Manhattan distance

$|x_1-x_2| + |y_1-y_2|$

L_2 : Euclidean distance

L_∞ : Visual distance

$\max(|x_1-x_2|, |y_1-y_2|)$

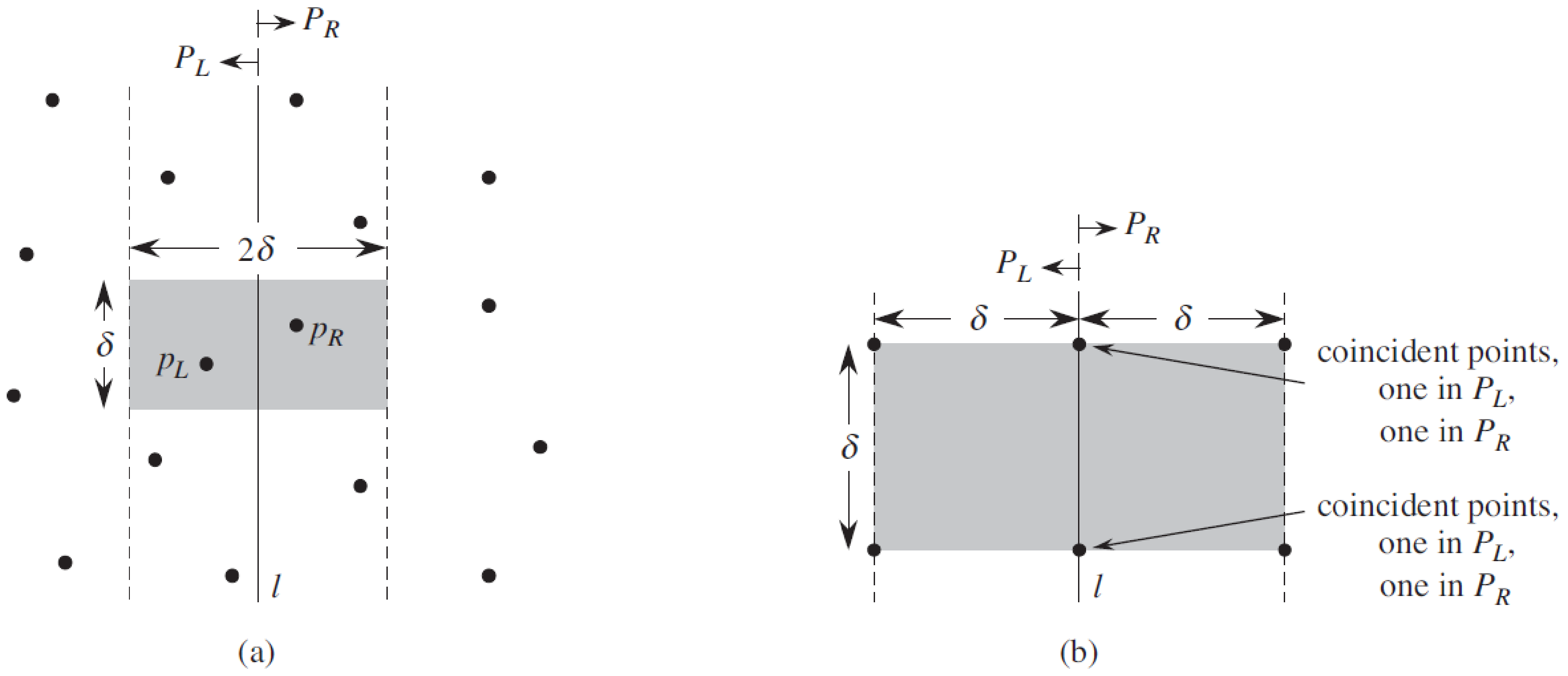
*33.4 Finding the closest pair of points

1. Brute method: $C(n, 2)$, $O(n^2)$

2. divide-and-conquer: $O(n \lg n)$

Euclidean distance

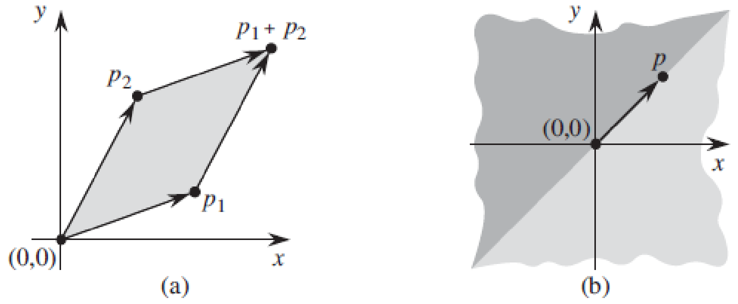
$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$



P_R
P_L
2δ
δ
p_R
p_L
l
(a)
P_R
P_L
δ
δ
δ
coincident points, one in P_L, one in P_R
coincident points, one in P_L, one in P_R
l
(b)

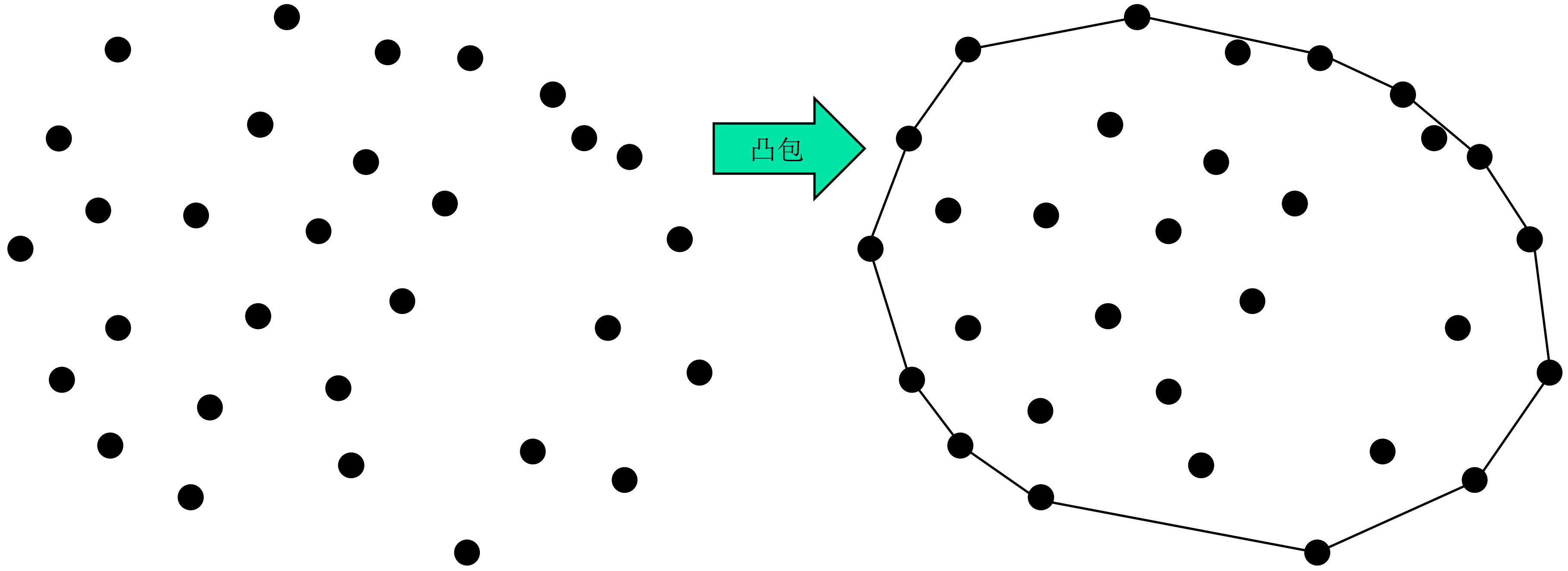
Exercise 1

$p_1 \times p_2$ can be interpreted as the signed area of the parallelogram formed by the four points (0, 0), p_1, p_2, and $p_1 + p_2 = (x_1 + x_2, y_1 + y_2)$?



Exercise 2

对类似下面这种特征的点集（左图）求凸包，从理论上讲，Jarvis's march 和 Graham's scan 哪个更快?

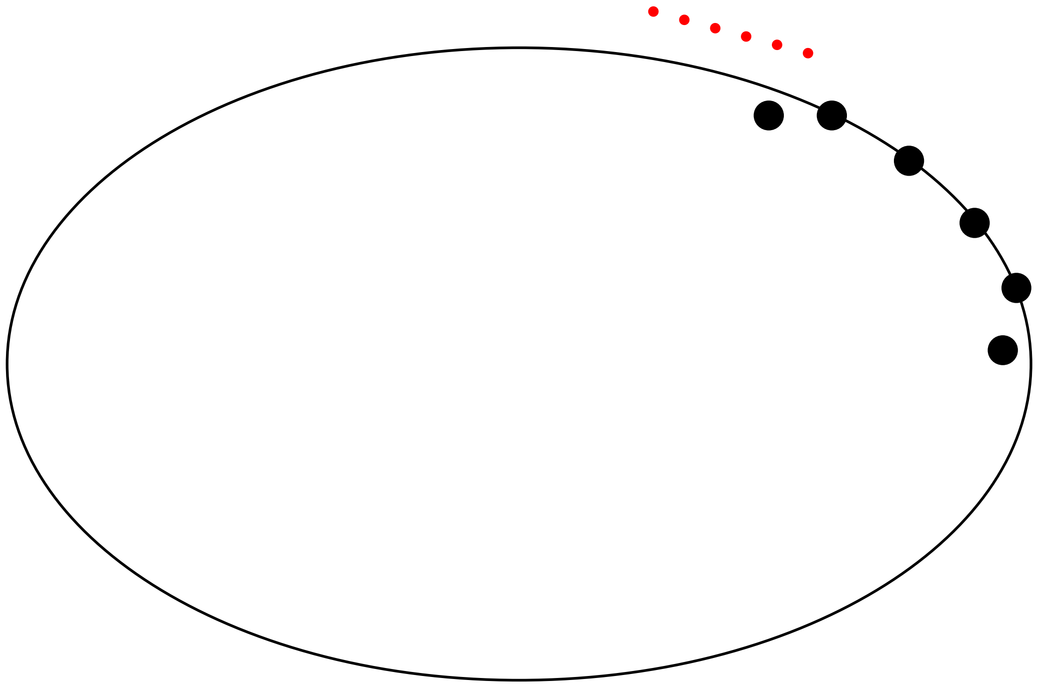


Q: 30个点

CQ: h = 14个顶点

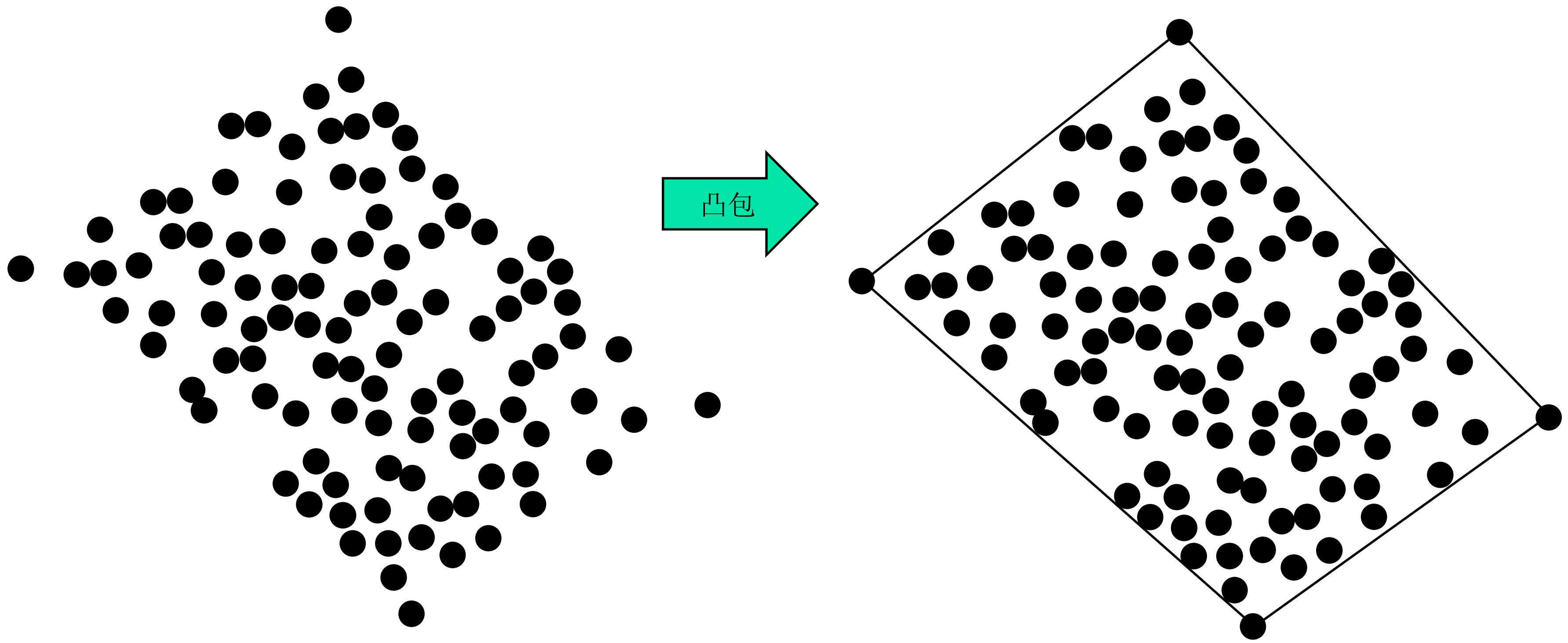
Exercise 3

这个呢？ （沿着椭圆周长上随机产生一列的点，内部有极少量零星的点）



Exercise 4

对类似下面这种特征的点集（左图）求凸包，从理论上讲，Jarvis’s march 和 Graham’s scan 哪个更快?



Q: #个点？

CQ: h = 4个顶点

讲座的建议

一、最后一次课（或两次），同学来讲。
二、主题（包括但不限于）：

1. 算法学习心得体会与心路历程
2. 本学期算法学习的知识串讲与知识点总结（如思维导图等）
3. 某个（些）关键算法的剖析（如图算法、计算几何等）
4. 算法验证（编程实现）的秘笈
5. 算法的研究与应用（一篇小论文）
6. 算法“实验室”（某些经典算法可视化展示）

n. ……

三、要求：把讲解内容提前发邮件给我，我筛选合适的内容来讲，讲之前我会单独指导。
四、大作业：主题的 2~n 均可，作品打包，发邮件给我。

五、奖励办法：

讲座的建议

一、最后一次课（或两次），同学来讲。
二、主题（包括但不限于）：

……

三、要求：把讲解内容提前发邮件给我，我筛选合适的内容来讲，讲之前我会单独指导。
四、大作业：主题的 2~n 均可，作品打包，发邮件给我。

五、奖励办法：

- 酌情加分
- 可能的其他奖励，包括但不限于：精神奖励和物质奖励
- 优先写各类推荐信
- 优先考虑为助教候选人
- 优先带毕业设计
- 优先考虑作为研究生候选人
- ……