第3讲
几何基础知识与应用

刘汝佳

目录

一、几何基础

向量空间

- 所有n维向量的集合, 加上两种基本操作
  - 向量加法: 满足交换律, 结合律, 单位元为0向量, 每个向量有逆元-v
  - 数量乘法: 满足交换律且对加法满足分配律
- 把数量乘法和向量加法合在一起构成了线性组合:

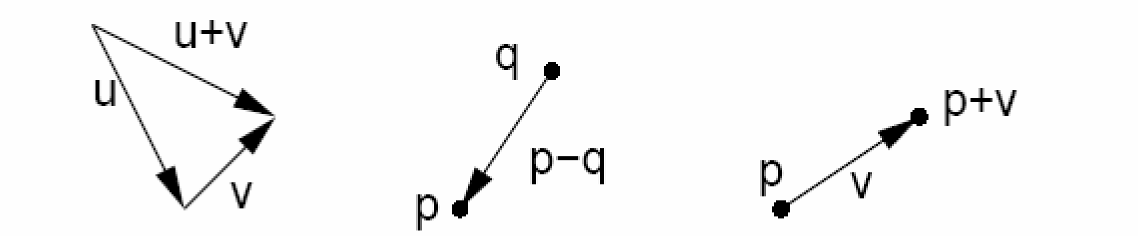
$$a_1\overrightarrow{v_1} + a_2\overrightarrow{v_2} + \cdots + a_n\overrightarrow{v_n}$$

- 如果规定向量的起点都在原点，则可以用几何图形表示向量加法和数量乘法

仿射几何

- 仿射几何(Affine Geometry)包括标量集合、点集合和自由向量集合, 标量一般是实数, 自由向量表示方向和大小, 而位置不固定

$S \cdot V$	$\rightarrow$	V	scalar-vector multiplication
$V + V$	$\rightarrow$	V	vector addition
$P - P$	$\rightarrow$	V	point subtraction
$P + V$	$\rightarrow$	P	point-vector addition



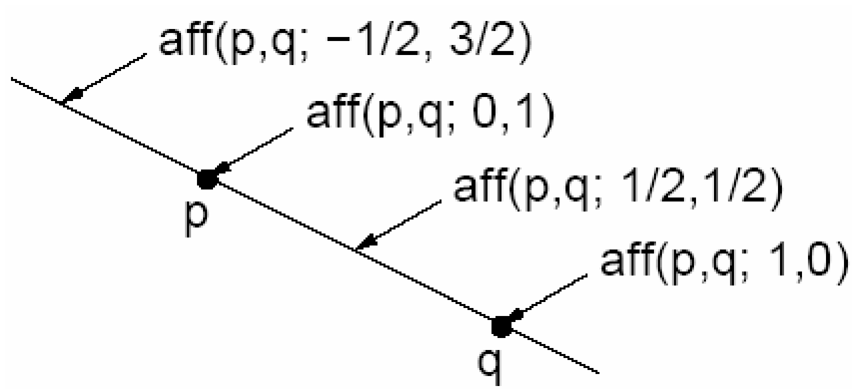
vector addition　point subtraction　point−vector addition

仿射组合

- 一般来说点和数不能相乘, 点和点也不能相加, 但 $\alpha_0 + \alpha_1 = 1$时以下情况是个例外

$$\mathrm{aff}(p_0, p_1; \alpha_0, \alpha_1) = \alpha_0 p_0 + \alpha_1 p_1 = p_0 + \alpha_1(p_1 - p_0).$$

- 中间项是不合法的, 但最后一项合法
- 系数在[0, 1]的情况称为凸组合, 三个非共线点的凸组合的轨迹为三角形



欧氏几何

- 在仿射几何中增加内积运算, 构成欧氏几何
- 最重要的内积运算是点积(dot product), 在坐标系下被定义为各分量对应乘积之和

Length: of a vector $\vec{v}$ is defined to be $|\vec{v}| = \sqrt{\vec{v} \cdot \vec{v}}$.

Normalization: Given any nonzero vector $\vec{v}$, define the *normalization* to be a vector of unit length that points in the same direction as $\vec{v}$. We will denote this by $\hat{v}$:

$$\hat{v} = \frac{\vec{v}}{|\vec{v}|}.$$

Distance between points: Denoted either $\mathrm{dist}(p, q)$ or $|pq|$ is the length of the vector between them, $|p - q|$.

Angle: between two nonzero vectors $\vec{u}$ and $\vec{v}$ (ranging from 0 to π) is

$$\mathrm{ang}(\vec{u}, \vec{v}) = \cos^{-1}\left(\frac{\vec{u} \cdot \vec{v}}{|\vec{u}||\vec{v}|}\right). = \cos^{-1}(\hat{u} \cdot \hat{v}).$$

低维点积与叉积

```
double dot(Vector2 u, Vector2 v)
{
    return u[1]*v[1]+u[2]*v[2];
}
double cross(Vector2 u, Vector2 v)
{
    return u[1]*v[2]-u[2]*v[1];
}

double dot(Vector3 u, Vector3 v)
{
    return u[1]*v[1]+u[2]*v[2]+u[3]*v[3];
}
Vector3 cross(Vector3 u, Vector3 v)
{
    return Vector3(u[2]*v[3]-u[3]*v[2], u[3]*v[1]-u[1]*v[3], u[1]*v[2]-u[2]*v[1];
}
```

二维仿射变换

- 平移**(Translations):** 坐标直接相加即可
- 旋转**(Rotations):** 仅考虑旋转中心为原点时. 如果不是, 则先平移, 再旋转, 最后平移回来, 以下是逆时针旋转a弧度的代码
- 反射**(Reflections)**: 仅考虑沿x=0反射的情形, 其他可以借助平移和旋转完成

```
Vector2 Trans2(Vector2 u, Vector2 v)
{ return Vector2(u[1]+v[1], u[2]+v[2]); }

Vector2 Rotate2(Vector2 u, double a)
{ return Vector2(cos(a)*u[1]+sin(a)*u[2], -sin(a)*u[1]+cos(a)*u[2]); }

Vector2 Reflections2(Vector2 u)
{ return Vector2(u[1], -u[2]); }
```

进一步学习的建议

- 其他变换: 缩放、剪切
- 齐次坐标与变换矩阵
- 三维仿射变换: 一般旋转矩阵、四元数…
- 投影
- …

二、基本二维计算

点到直线的距离

- 对于二维情形, 统一用两点表示直线, 直线方程写成参数形式, 直线上的点用参数表示
- 点到直线的距离. 设垂足为P, 则PP_3与P_1P_2垂直, 即$(P_3-P)\cdot(P_2-P_1)=0$, 把P代入直线方程$P=P_1+t(P_2-P_1)$解得

$$u=\frac{(x_3-x_1)(x_2-x_1)+(y_3-y_1)(y_2-y_1)}{\|P_2-P_1\|^2}$$

- 注意除0错误

线段、射线、直线交点

- 仍用参数方程, 设直线为P_1P_2和P_3P_4, 则交点在两直线上的参数分别为

$$
\begin{cases}
u_a = \dfrac{(x_4-x_3)(y_1-y_3)-(y_4-y_3)(x_1-x_3)}{(y_4-y_3)(x_2-x_1)-(x_4-x_3)(y_2-y_1)} \\
u_b = \dfrac{(x_2-x_1)(y_1-y_3)-(y_2-y_1)(x_1-x_3)}{(y_4-y_3)(x_2-x_1)-(x_4-x_3)(y_2-y_1)}
\end{cases}
$$

- 对于线段、射线仍然适用, 只需要检查参数的范围即可
- 注意二直线平行、重合的情形

直线和圆的交点

- 用参数式表示直线P_1P_2, 中心在P_3的圆用标准方程, 则联立可得一个关于参数u的二次方程, 其中三个系数分别为

$$\begin{cases} a = (x_2 - x_1)^2 + (y_2 - y_1)^2 \\ b = 2((x_2 - x_1)(x_1 - x_3) + (y_2 - y_1)(y_1 - y_3)) \\ c = x_3^2 + y_3^2 + x_1^2 + y_1^2 - 2(x_3x_1 + y_3y_1) - r^2 \end{cases}$$

过三点的圆

- 利用点积可以得到三个距离等式, 解出即可得到圆心坐标. 假设三个顶点按逆时针排列, 设$X_i=x_i-x_0$, $Y_i=y_i-y_0$, 则三角形有向面积

$$A = \frac{1}{2}\det\left[\begin{array}{cc} X_1 & Y_1 \\ X_2 & Y_2 \end{array}\right]$$

- 另L_{ij}为点i和点j的距离, 则圆心为

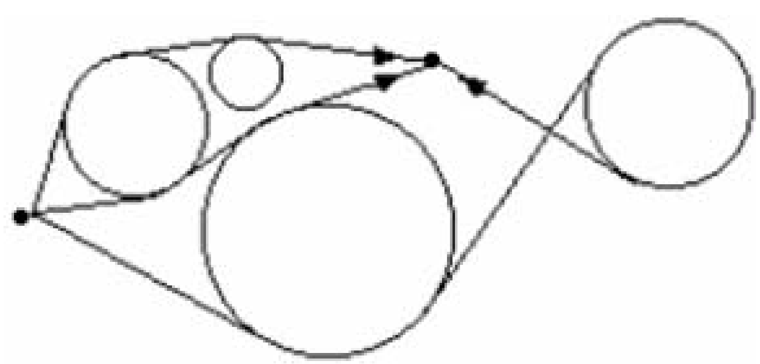
$$\left\{\begin{array}{l} x = x_0 + \frac{1}{4A}(Y_2 L_{10}^2 - Y_1 L_{20}^2) \\ y = y_0 + \frac{1}{4A}(X_1 L_{20}^2 - X_2 L_{10}^2) \end{array}\right.$$

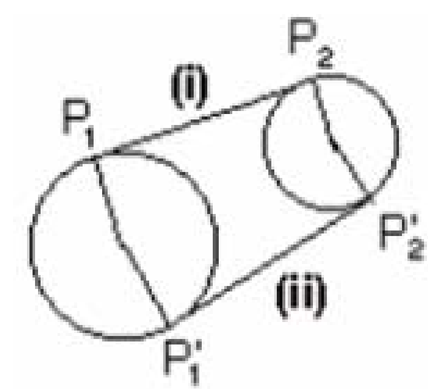
例1. 最短路

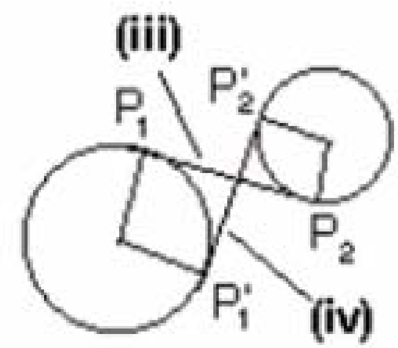
- 平面内有n个不相交的圆，求从出发地到目的地的不经过任何圆的最短路长

分析

- 基本性质: 绷紧
  - 最短路径由连续的线段和圆弧间隔组成
  - 线段连接两圆，并与两圆相切，且线段端点为切点。有四种情况
  - 圆弧总是在那些已知圆上

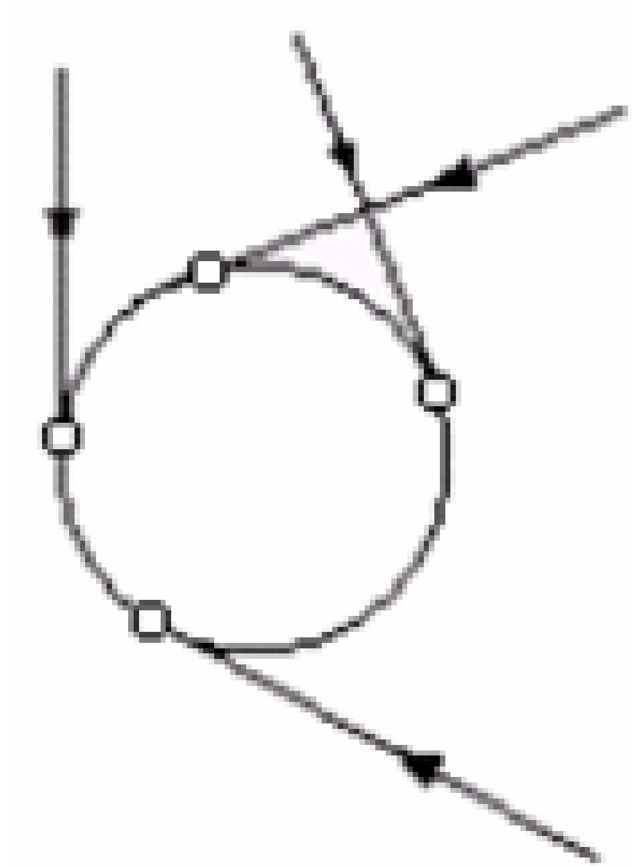






登陆点

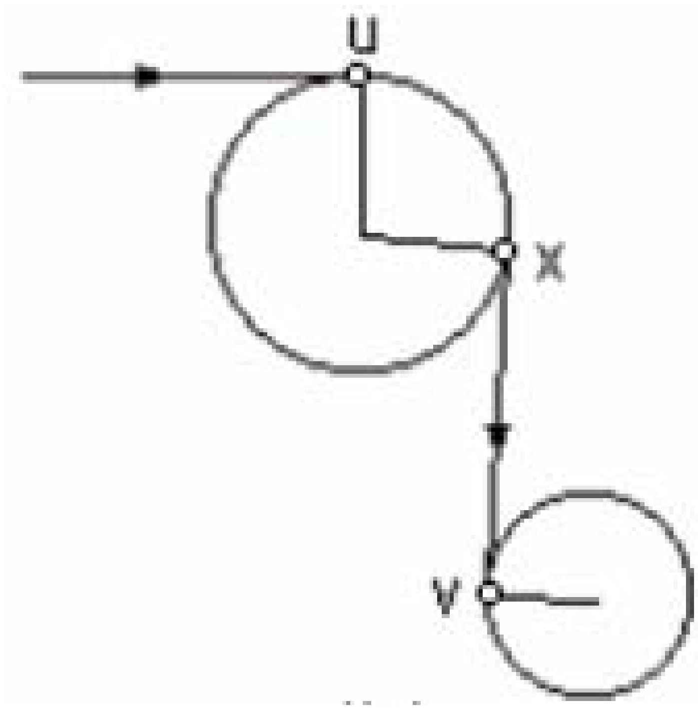
- 路径到达一个圆时，只可能在以入射路径为切线的切点登陆，所以若其余圆(出发地和目的地算是半径为0的圆)的个数为m，则至多有4m个登陆点。注意路径不能穿过任何圆



边权

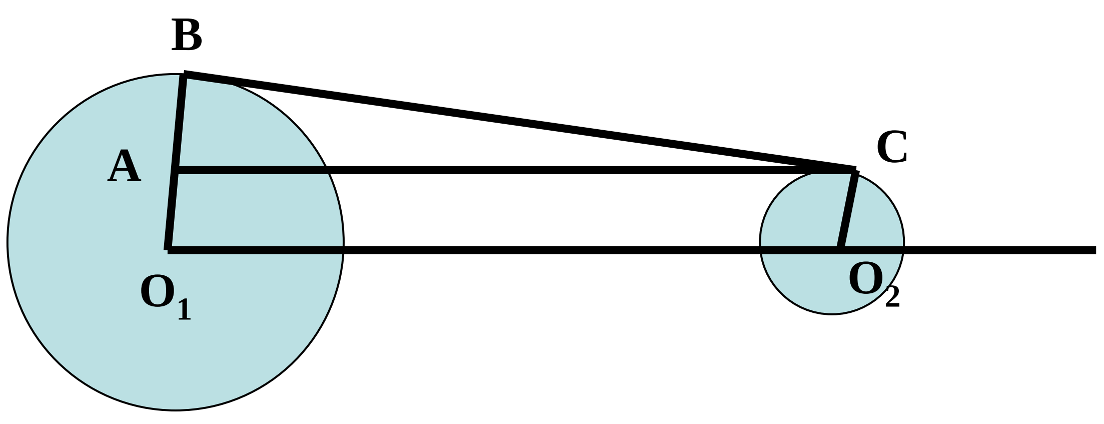
- 两个点uv之间的路段
  - 顺着圆滑过一短圆弧u→x
  - 沿直线离开到达v
- 因此

  $$w(u,v)=d_c(u,x)+d(x,v)$$
- 注意计算圆上距离d_c时应考虑顺时针和逆时针两种方案, 选短的一种



切点

- 本题的几何计算中最困难的是求出四种切线. 由于切点在圆上, 因此用极角表示点, 而不用坐标
- ACO_2O_1是平行四边形, 故$AB=r_1-r_2$, $AC=d$
- 直角△ABC中, $\angle BAC=\arccos(r_1-r_2)/d$



其他切点

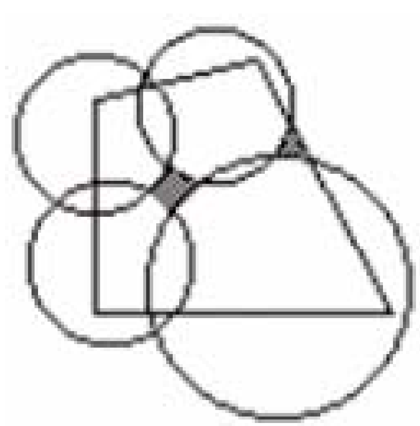
- 对于每两个圆i, j, 需要求出四种切点相对于二者的极角, 一共8个极角
- 算内公切线时$AB=r_1+r_2$
- 所求出的8个极角只是相对于O_1O_2的. 若O_1O_2不是水平方向, 则真正的极角还需要加上O_1O_2的极角
- 角的计算统一用atan2函数, 每次计算后归整化到[-pi, pi)范围之内

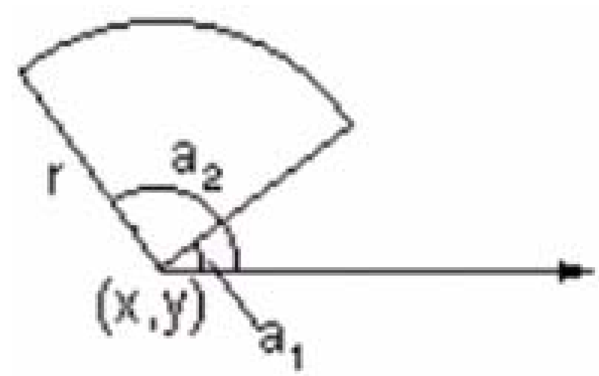
例2. 圆和凸多边形

- 给定一个凸多边形P和n个圆, 问这些圆的并是否覆盖了P

主算法

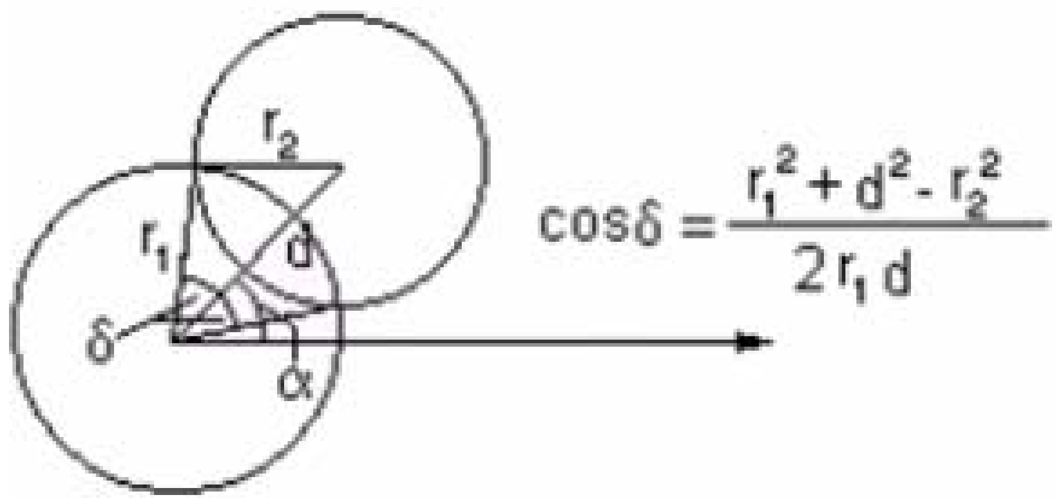
- 先求圆并的轮廓线C, 然后删除P外面的弧, 得到C’
  - C’非空，则C’和P一起围成了未盖部分
  - C’为空, 要么完全没被覆盖要么完全被覆盖
- 圆弧的表示: (x, y, r, a_1, a_2)





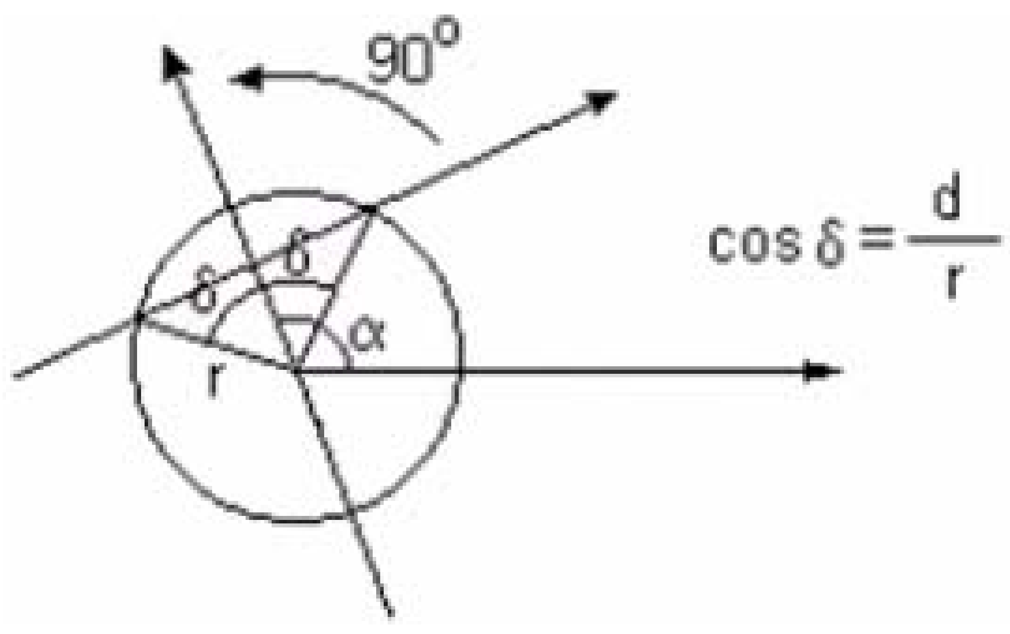
C的计算

- 所有圆两两求交, 删除被覆盖的弧, 则剩下的弧为所求
- 关键是计算出被删除圆弧的起止角度, 先求圆心连线角度, 则被删除范围为[α-δ, α+δ]



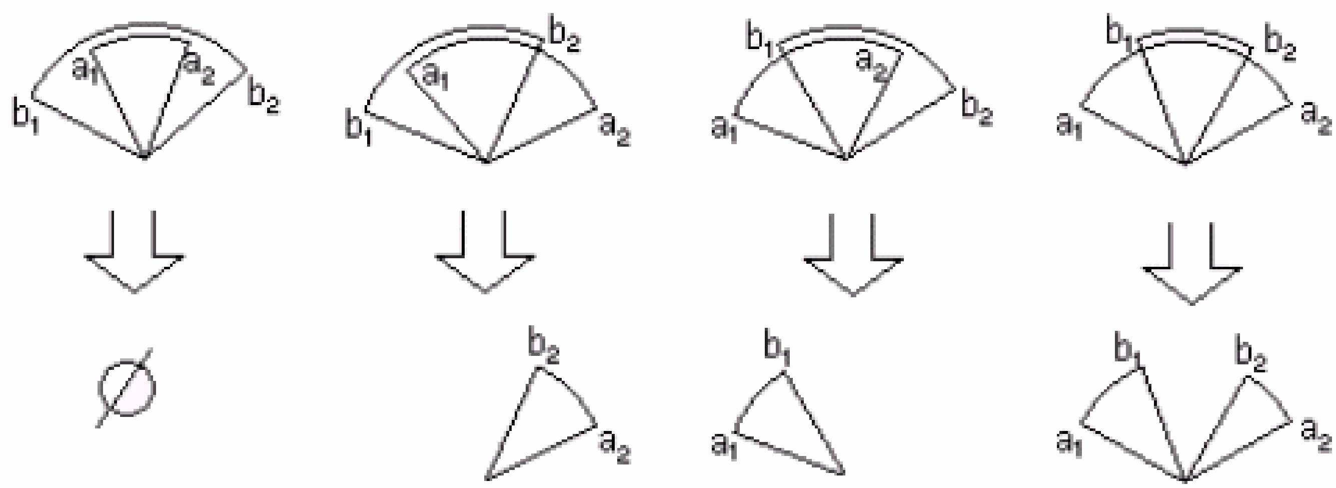
C’的计算

- 对于凸多边形每条边, 删除在它左侧的部分. 不求交点而直接计算被删圆弧的角度范围
- 为了方便, 把有向线段逆时针旋转90度



链表

- 为了处理各种情况, 现用链表储存每个圆上残留的圆弧, 以下是四种可能的情况 (原来为a_1-a_2, 删除部分为b_1-b_2)
- 总时间$O(n^3+mn^2)$, m为凸多边形顶点数

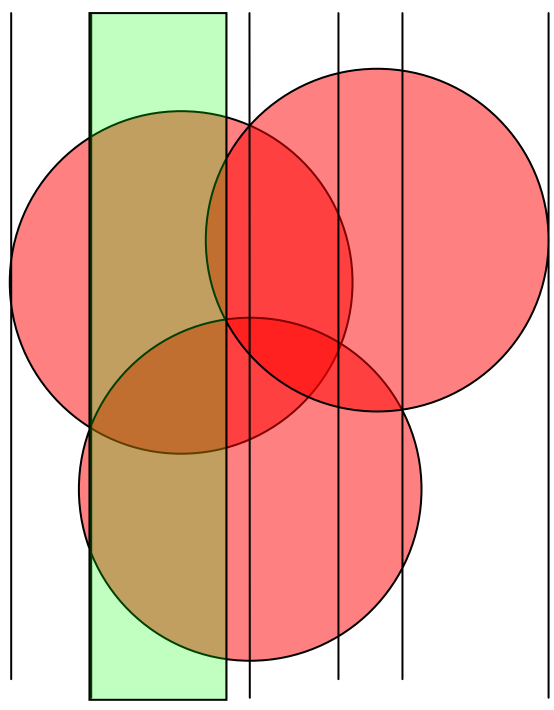


例3. 圆的并面积

- 求平面上n个圆的并的面积

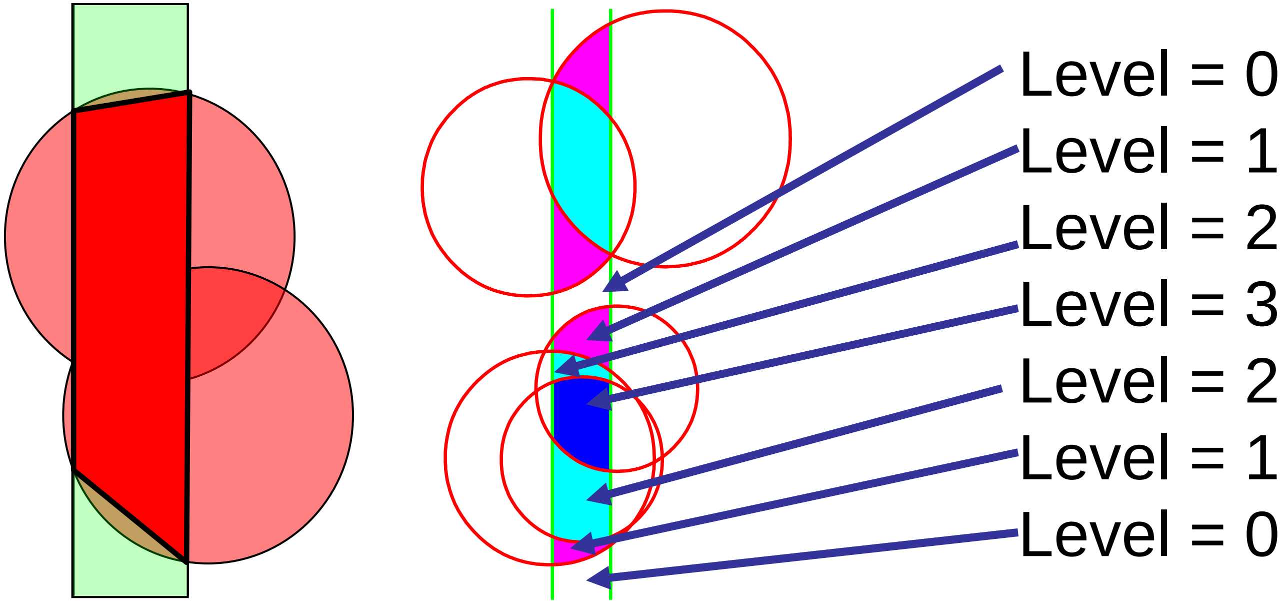
竖直离散化方法

- 考虑竖直离散化(交点和竖切线)则每个长条内为曲边梯形的并, 可以用扫描法求区间并



竖直条内的情况

- 每个圆和竖直条的相交部分由若干不相交的曲边梯形组成, 称为一个连通块. 注意每个连通块的上弧和下弧可能不是同一个圆贡献出来的



求各连通块

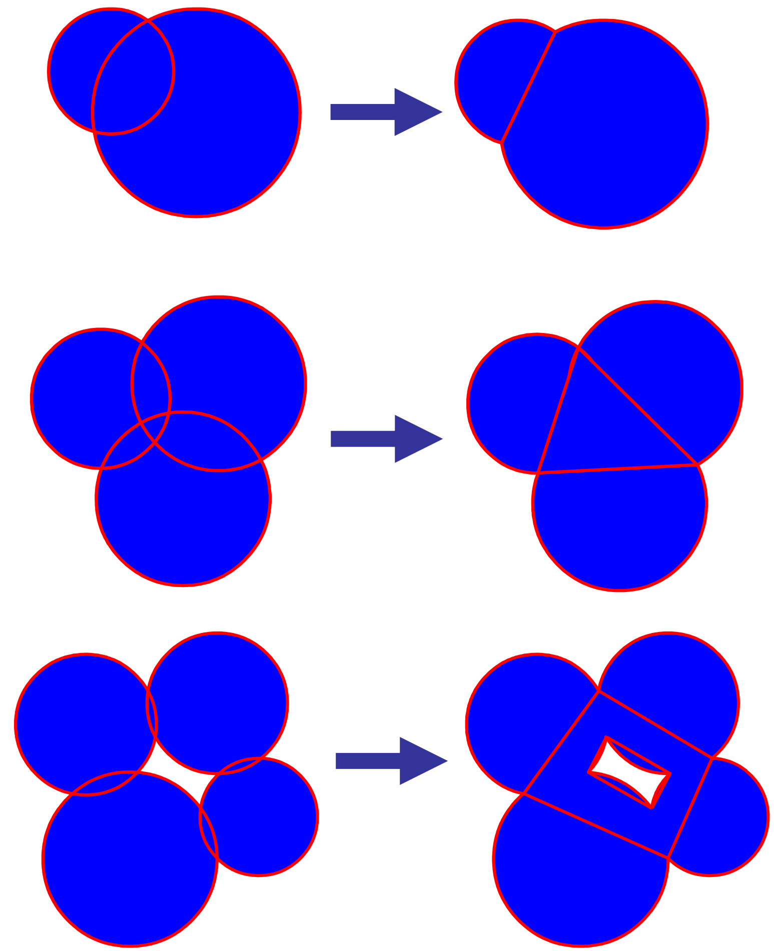
- 每个与竖直条切割的圆都被切出了一条上弧和一条下弧, 所有这些弧都是不相交的
- 把它们从上到下排序(按左右端点为主次关键字即可), 然后从上到下扫描
- 初始level为0, 见上弧就加1, 见下弧就减1, 好比处理括号一样
  - level第一次从0边1的时候意味着发现新连通快, 记录此上弧为上边界;
  - level第一次从1变为0时意味着连通块结束, 计算面积

时间复杂度

- 求交点: $O(n^2)$
- 离散化: $O(n^2 logn)$
- 每个竖直条内
  - 求所有圆被切割的上下弧: O(n)
  - 给O(n)对上下弧排序: O(nlogn)
  - 从上到下扫描/计算面积: O(n)
- 一共有$O(n^2)$个竖直条，因此时间复杂度为$O(n^3 logn)$, 实际上远没有这么大

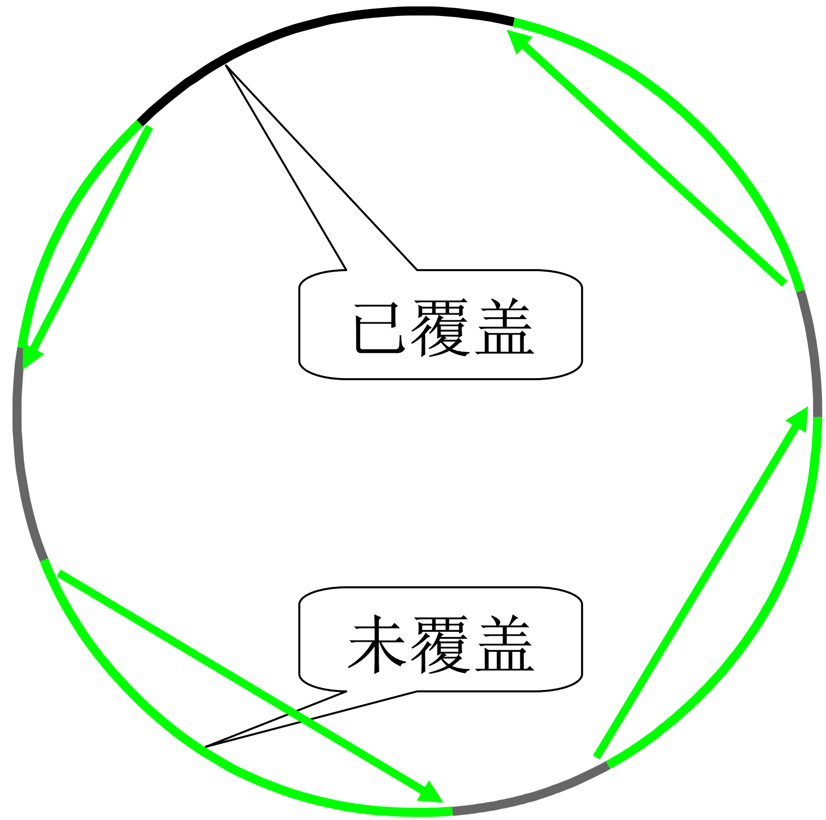
切割法

- 另一种直观的方法
- 忽略相交部分的轮廓
- 有的交点没用上
- 另一些交点相互连接
- 关键:
  - 求有用点
  - 求弓形
  - 求多边形



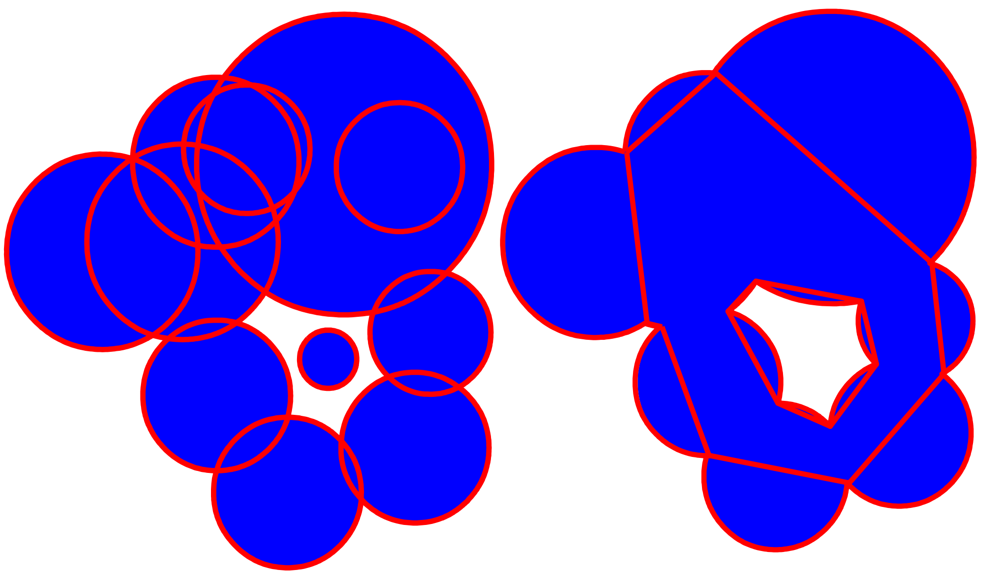
考虑单独每一个圆

- 每个圆被切割掉一些部分，用有向弦把未覆盖部分连接起来，构成弓形
- 弦的端点一定是圆的交点
- 每条有向弦恰好有一个后继弦



完整的算法

- 有向弦构成多个有向多边形, 逆时针为正, 顺时针为负
- 弓形面积和与有向多边形面积和即为所求



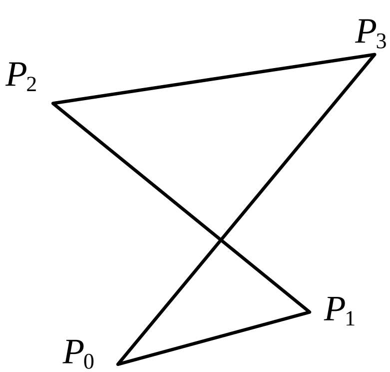
时间复杂度

- 处理每一个圆
  - 计算其他圆覆盖它的区间: O(n)
  - 排序并扫描得到各条有向弦: O(nlogn)
  - 计算各个弓形总面积: O(n)
- 这一步的总时间是$O(n^2 \log n)$
- 有向弦最多n^2条, 计算有向多边形总面积为$O(n^2)$, 总时间复杂度为$O(n^2 \log n)$

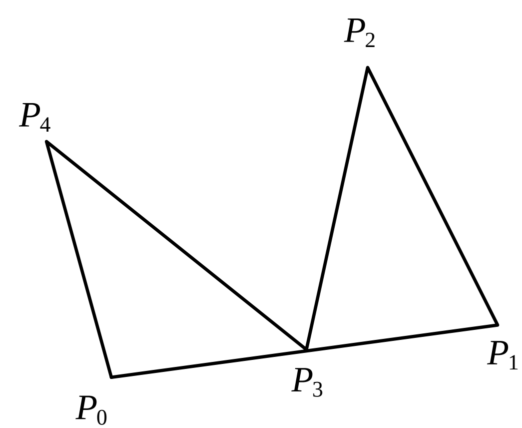
三、多边形

多边形的定义

- 二维平面上被一系列首尾相接、闭合的折线段围成的区域
- 复杂多边形、临界多边形、简单多边形
- 一般只考虑简单多边形



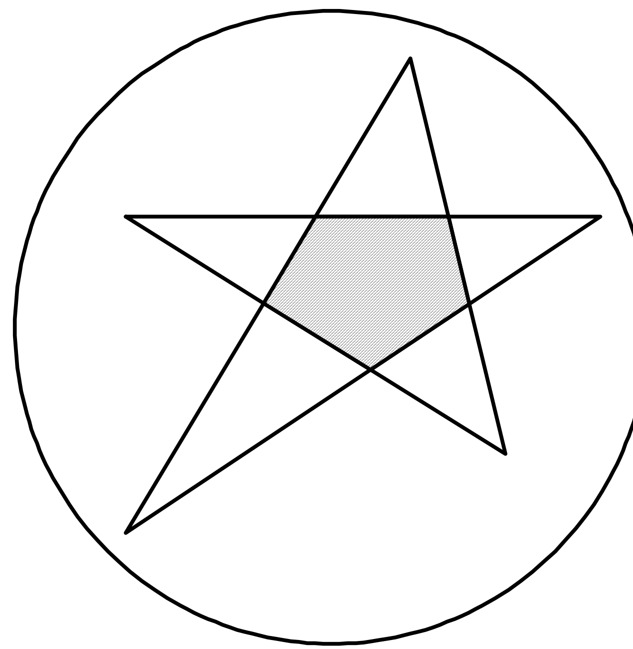
(a) 复杂多边形



(b) 邻界多边形

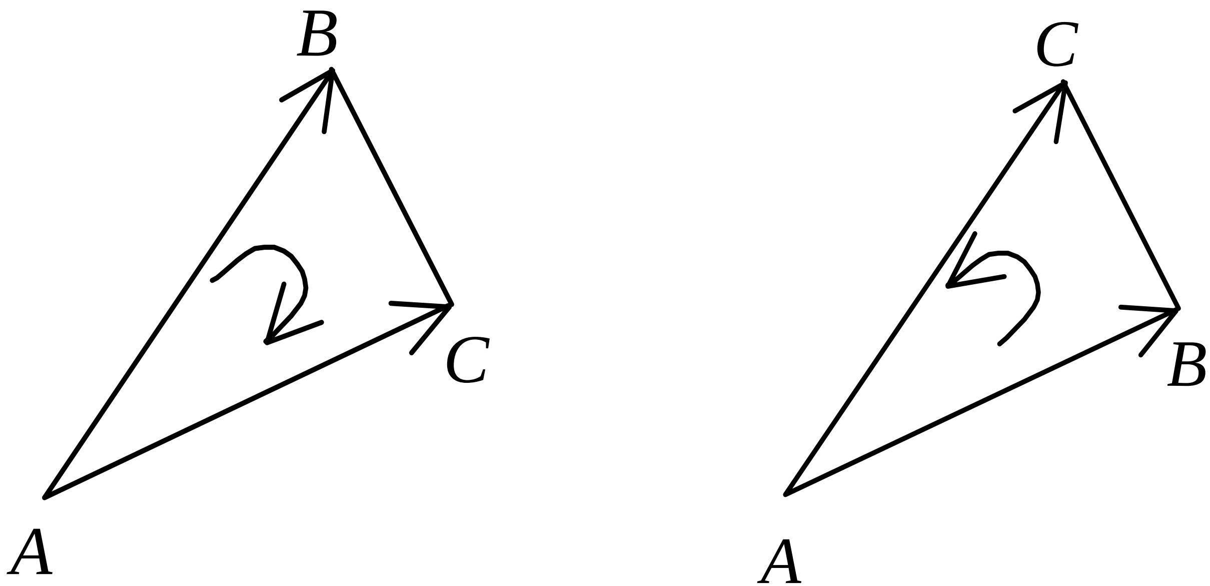
凸多边形和星形多边形

- 凸多边形
  - 对任何一条边e，整个多边形在e的一侧（如果是正方向，则必是左侧）
- 星形多边形
  - 存在多内部一点，能“看到”多边形内所有点
  - 所有满足定义要求的点组成的区域称为“核”
  - 由核和围绕着“核”的尖角构成，总与圆同态
- 注意“看到”和“清晰看到”的区别



有向面积

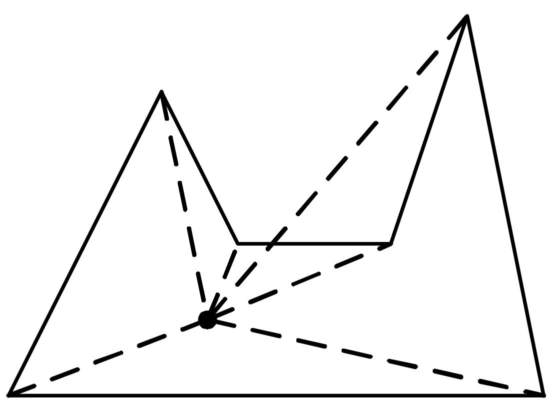
- 左图：ABC成左手系，负面积
- 右图：ABC成右手系，正面积

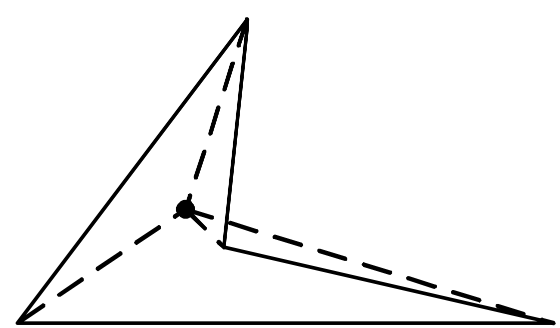


多边形面积

- 三角形的情形：叉积算有向面积
- 一般情形：扇形剖分法
- 公式：从(0,0)点出发所有三角形的有向面积和

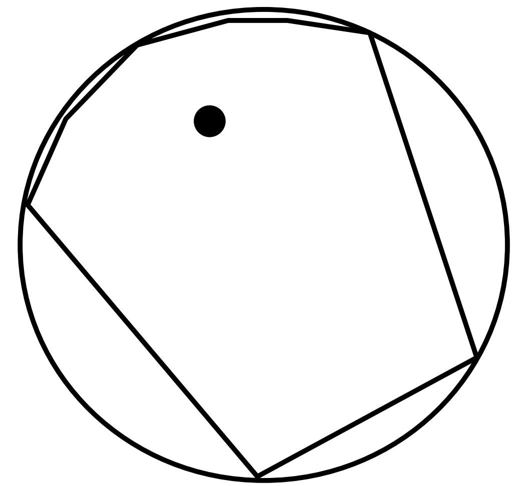
$$A=\sum_{i=1}^{N}A\left(\Delta OP_iP_{i+1}\right)=\frac{1}{2}\sum_{i=1}^{N}\overrightarrow{OP_i}\times\overrightarrow{OP_{i+1}}=\frac{1}{2}\sum_{i=1}^{N}\begin{vmatrix}x_i & y_i\\ x_{i+1} & y_{i+1}\end{vmatrix}$$

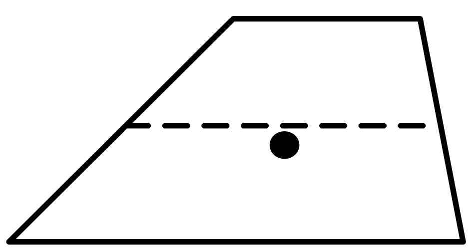


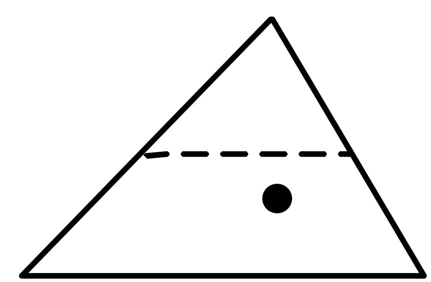


重心

- 质点系重心：加权平均
- 均匀物体重心
  - 坐标平均？
  - 扇形剖分

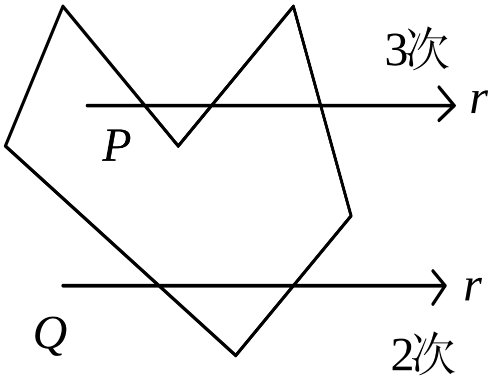


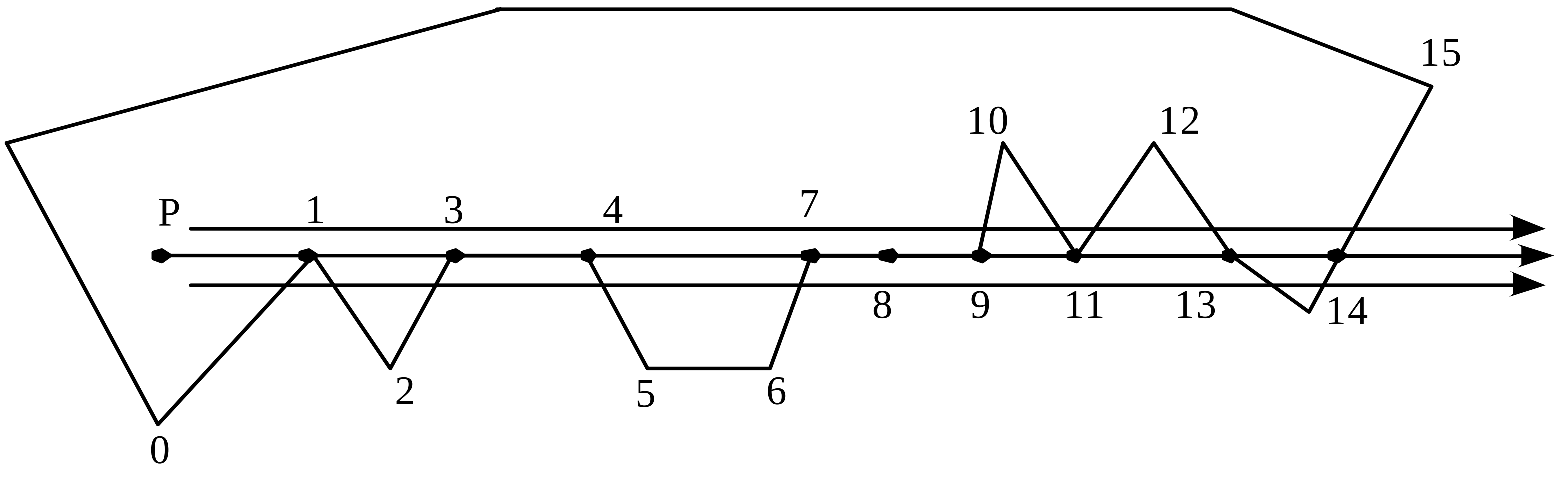




点在形内外的判定

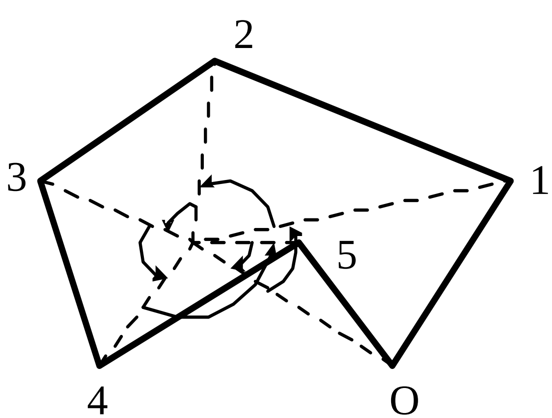
- 射线法
  - 交点奇数/偶数
  - 射线经过顶点: 左开右闭
  - 射线经过边：平移（模拟）
  - 点本身在边上呢？

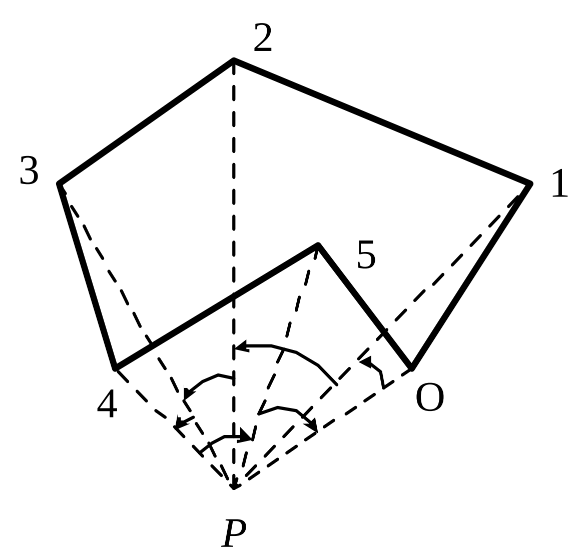




点在形内外的判定

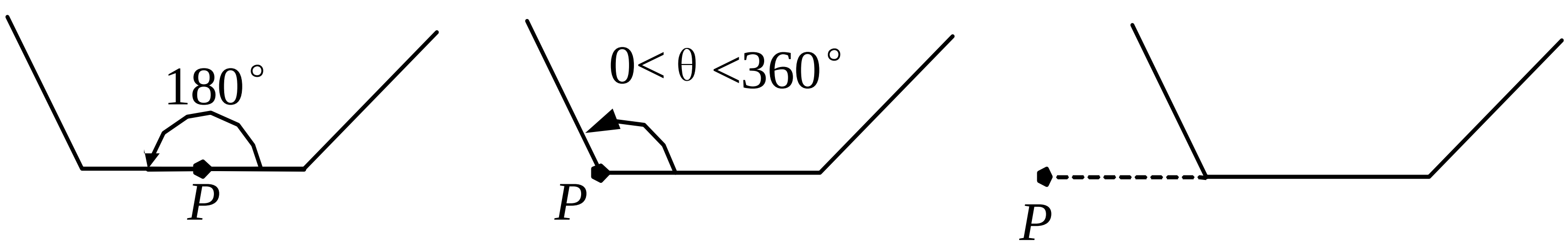
- 弧长法
  - 以被测点为圆心作单位圆，计算弧长代数和。
  - 代数和为0, 2π , π ， 点在多边形外部, 内部, 边上;
  - *优化: 累积法





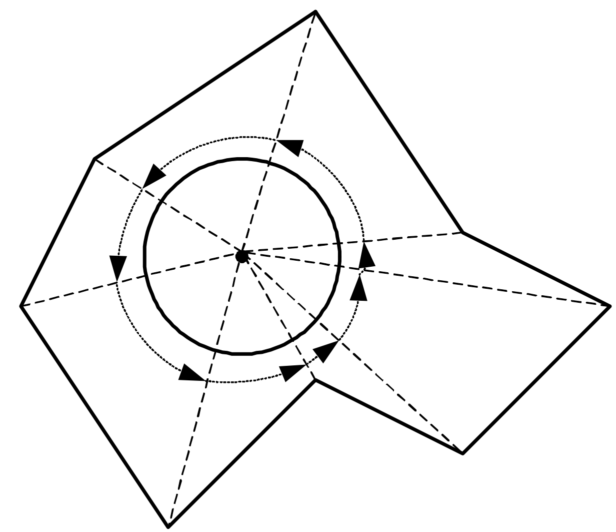
弧长法的讨论

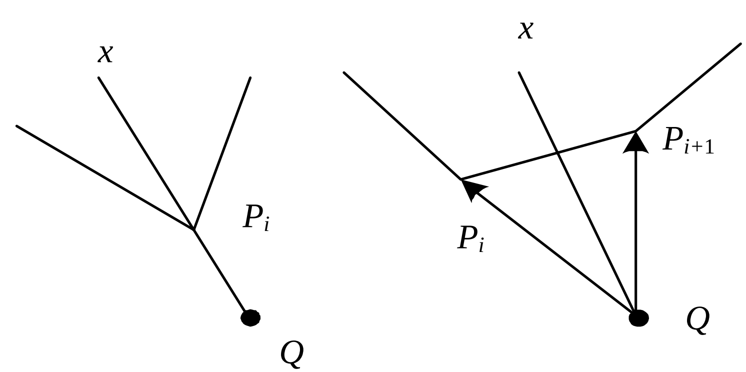
- 叉积=0
- 则总角度变化
  - (1) 180度
  - (2) 任意角度
  - (3) 无影响
- 由于(1)和(2)都只可能出现一次…



星形多边形的情形

- 二分：[0, 2π]
- 基本判断：左右（叉积）
- 结束
  - 共线：点积和坐标比较
  - 不共线：叉积判断
- 预处理
  - 凸：O(n)
  - 星形：O(nlogn)（先求核）
- 询问O(logn)





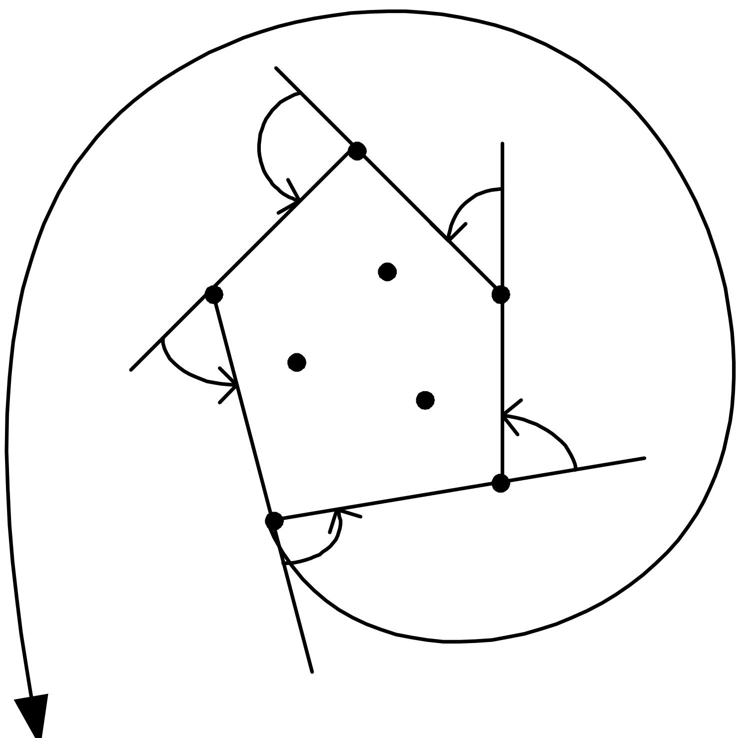
四、二维凸包

引例

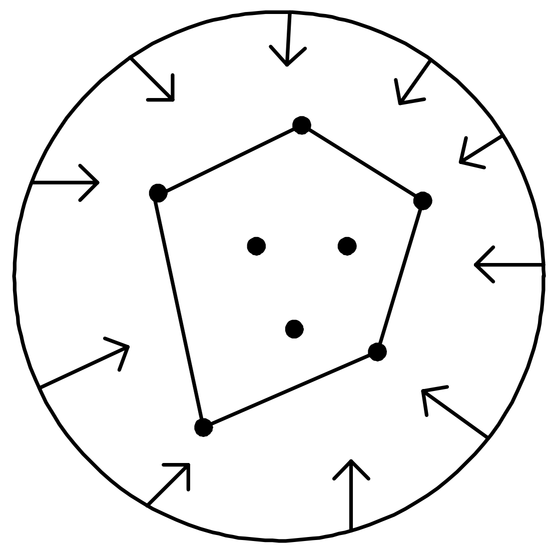
- 篱笆问题. 假设你种了很多树，想用一个篱笆把所有的树都包在里面。出于经济考虑，这个篱笆应该是越小越好。
- 合金制造问题. 假设你有一些金-银合金，它们的含金量和含银量各不相同，现在要求你通过按某种比例混合，制造出新的合金，那么哪些含金是可能被合成的？更精确地，通过这些现有的合金制造出的新合金，它们的含金量、含银量在什么范围内？

篱笆问题

- 凸包的形成方法



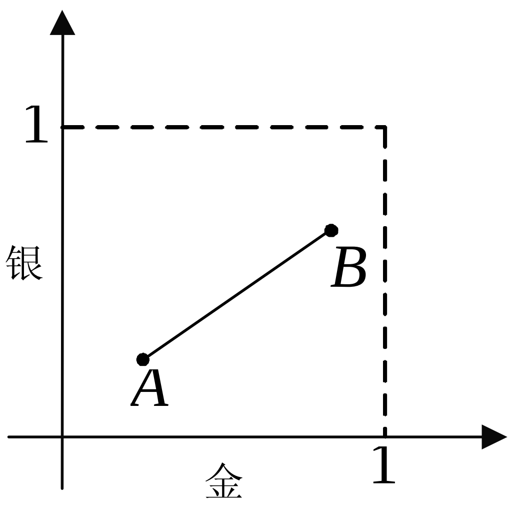
拉紧形成凸包



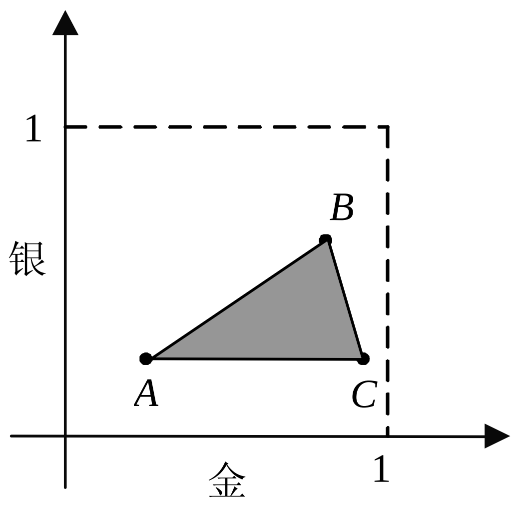
收缩形成凸包

合金制造问题

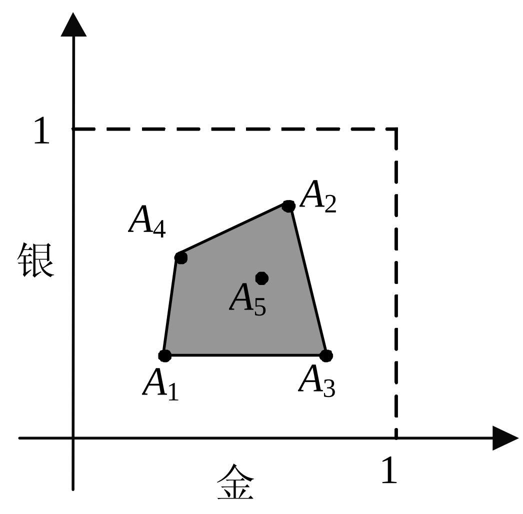
- 增量考虑
  - 两个合金的情形: 整条线段都能取到
  - 依次增加合金，即3、4、5…种合金
  - 区域内任一点和新增点组成的线段的轨迹



(b)



(c)



(d)

凸图形

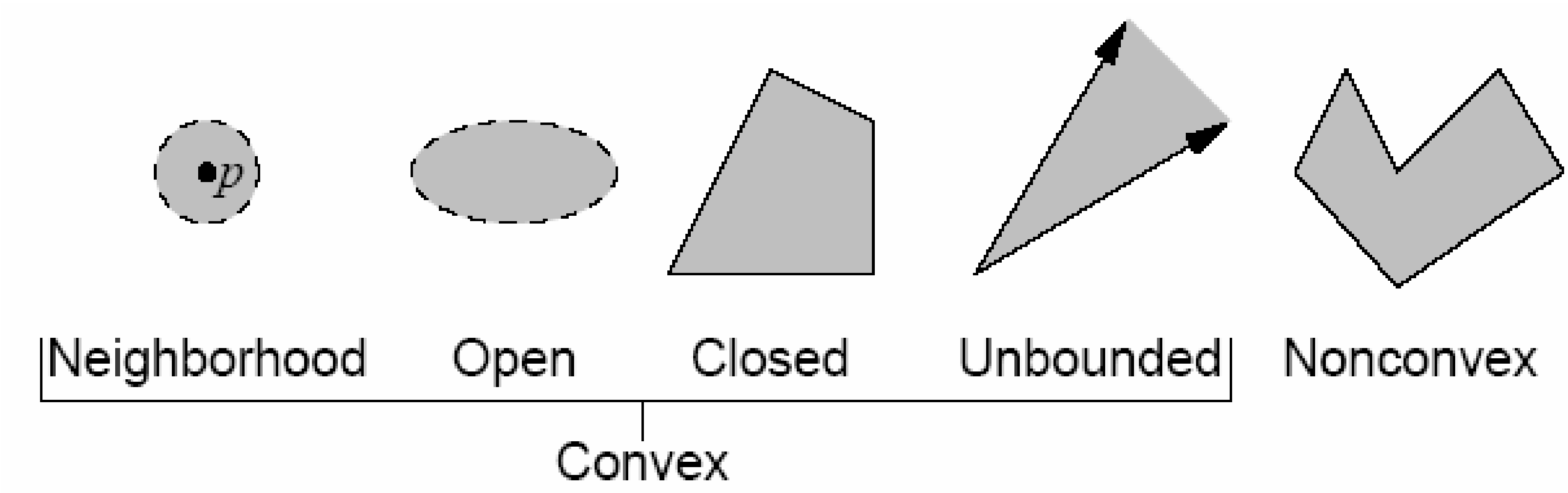
- 直观：整个图形在任一条边的一侧
- 代数：任意两点的中点都在此图形内
  - 推广1：中点→内分点
  - 推广2：两点→n点，权和为1
- 凸包：所有点的凸组合
- 极点：总是在原始点集中

凸包的优势

- 点的数量大大减少
  - k维球体中均匀独立分布n个点，凸包顶点数$O(n^{(k-1)/(k+1)})$
  - k=2（平面）时，$m(2)=O(n^{1/3})$
  - k=3（空间）时，$m(3)=O(n^{1/2})$
  - 凸包可以大大降低平均情况时空复杂度
- 凸包相对于原点集增加了一个“序”（多边形）
  - 相对于原多边形，凸多边形拥有许多特殊优美性质
  - 这些序和性质有时候可以从本质上带来新的算法，降低最坏情况时空复杂度。

什么是凸?

- 有时候, 需要注意特殊的凸

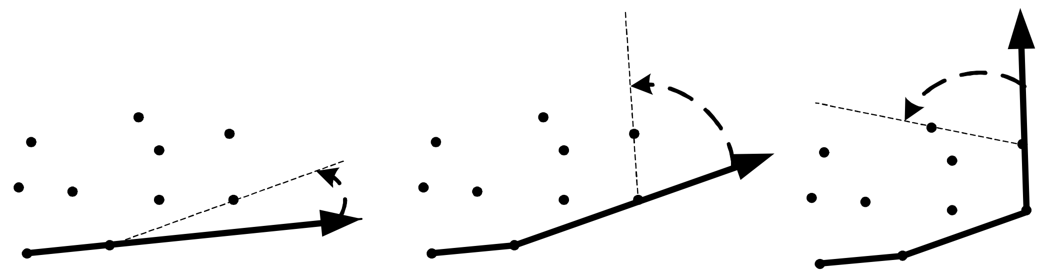


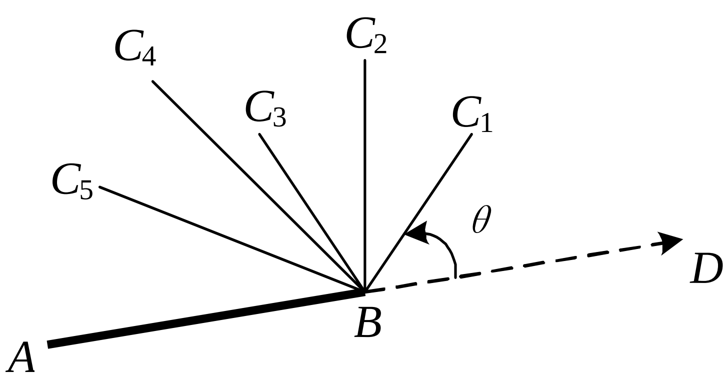
凸包的算法

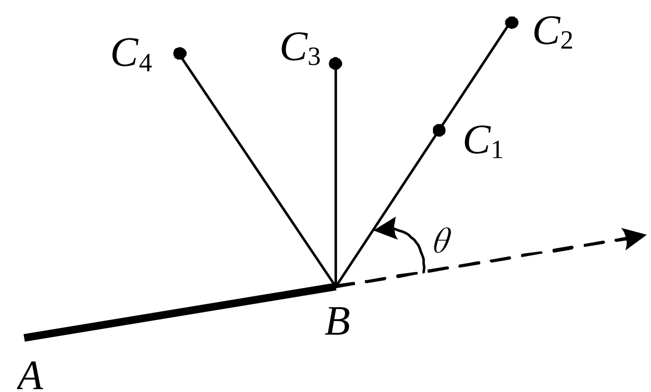
- 卷包裹法
- Graham-Scan算法
  - 试探性凸包法
  - 极角序下的Graham-Scan
  - 水平序下的Graham-Scan
  - 正确性证明
- 凸包的时间下界
- 多边形凸包的Melkman算法

卷包裹法

- 每次算出凸包的一条边
- 找下一条边：极角序的下一个（叉积）
- 时间：$O(n^2)$
- 共线点（右下）
  - 先判左右
  - 共线则判前后

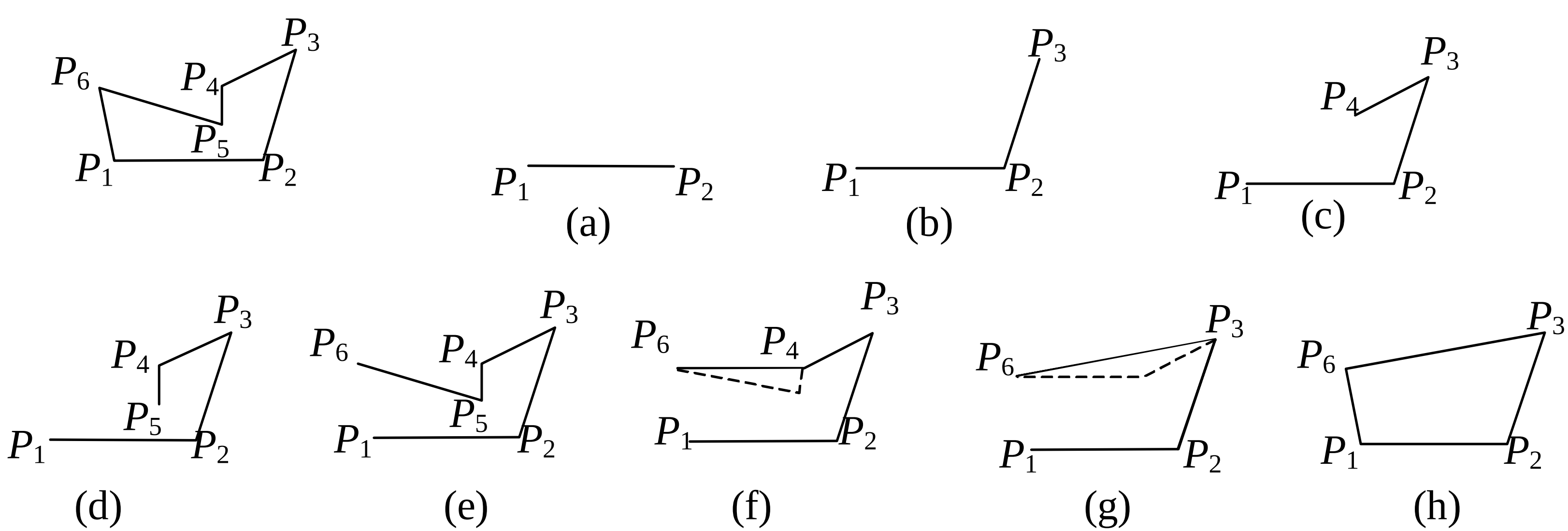






Granham试探性凸包

- 卷包裹：每次算出凸包的一条边
- 另一个想法：只需要维持临时凸包（局部凸包）
  - 试探性增长
  - 逐步修正错误，逼近正确解
- 输入是简单多边形：O(n) （不一定正确）



多边形→点集

- 多边形凸包: 简洁的算法（但不一定正确！）

  push(p_1); push(p_2);

  i=3;

  while i<=n do

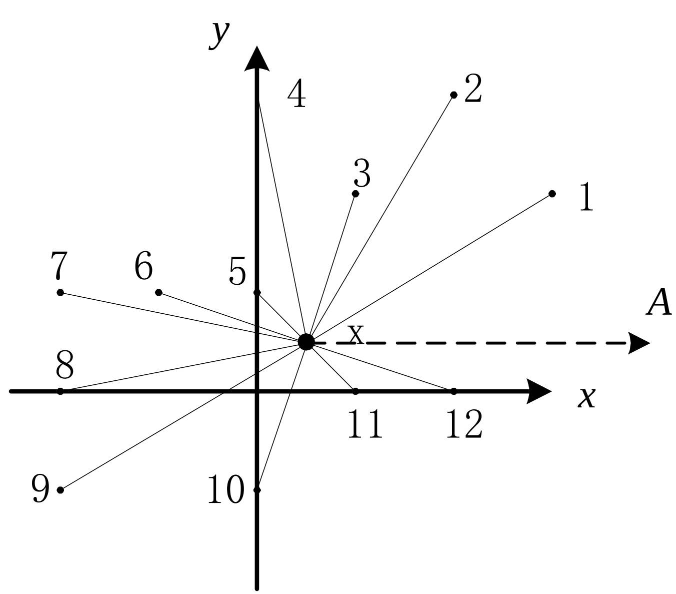
  if p_i 在栈顶边 $p_{t-1}p_t$ 左手方向

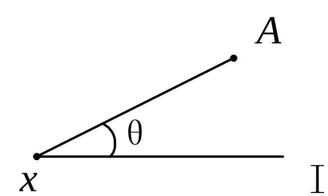
  then push(p_i) 并且 i++

  else pop();
- 试探凸包法的前提
  - 简单多边形
  - 预定的序!
- 无序的点集怎么给个序?

极角序

- 问题
  - 转了360度：叉积判断的问题
  - 起点：一定在凸包里吗？
  - 排序参考点x如何确定？

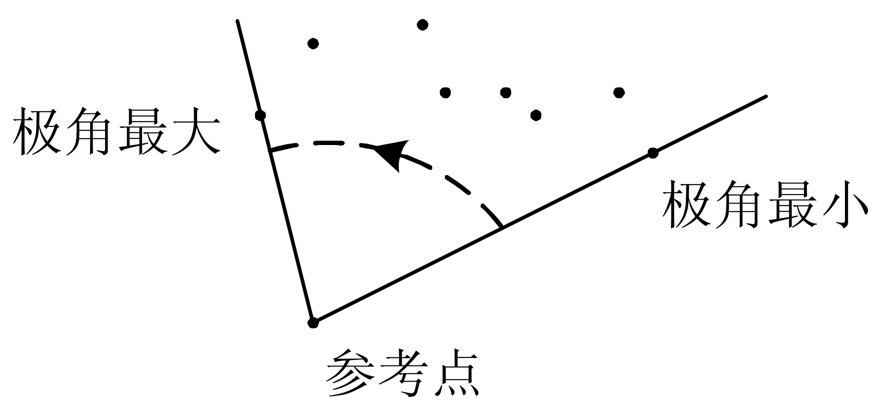




$$\theta = \arctan 2(\vec{A} - \vec{x})$$
$$= \arctan 2(A.x - x.x, A.y - x.y)$$

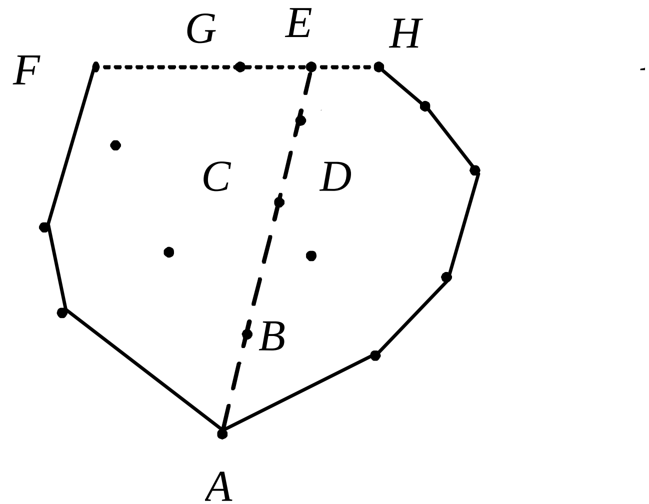
修改后的极角序

- 固定参考点的极角序
  – 左下角为参考点
  – 极角最小和最大为起点和终点，一定在CH上
  – 排序O(nlogn)，扫描O(n)
- 问题：重点（直接删除）和三点共线



三点共线难题

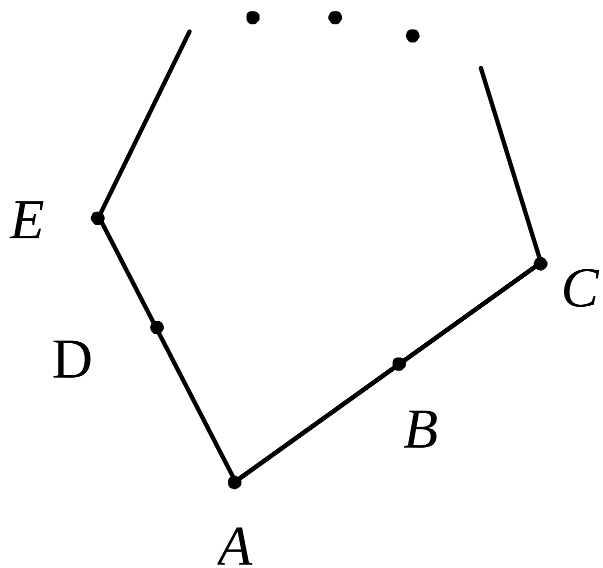
- 两种情形（可同时存在）
  - 排序时共线
  - 凸包边上的共线



(a) 两种共线的例子：
排序时的共线 *ABCDE*；凸包边上的共线 *FGEH*

三点共线的处理

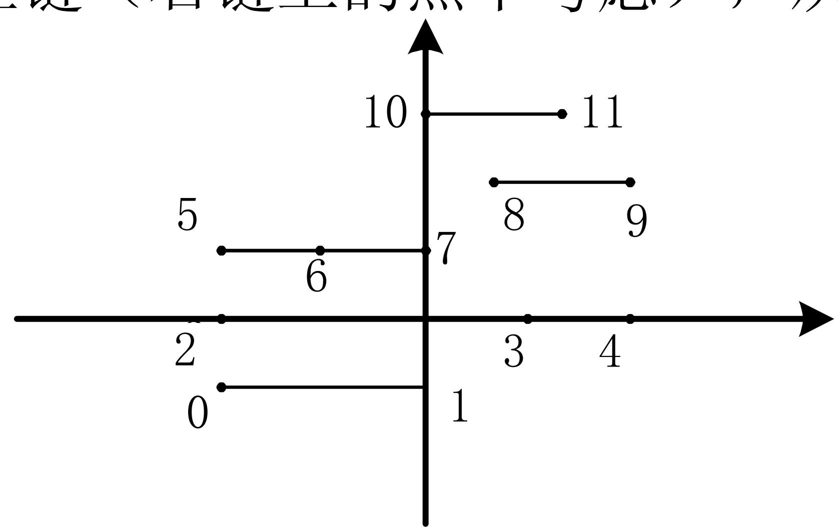
- 只求极点
  - 修改的极角排序法（先左右后前后）
- 求所有点？
  - 始边和终边矛盾！
- 可以寻找其他序！



(b) 两种共线重合：
始边 *ABC* 和终边 *EDA*

水平序

- 按*y*坐标排序，坐标相同的再按*x*坐标排序
  – 排序比较更简单了，只是简单的比较，没有运算
  – 起始点更好找了，就是排序后的0点
- 要把扫描过程分成两步，右链和左链
  – 先做右链，以0到排序最后点（即最高点11）
  – 再反向做左链（右链上的点不考虑），从最高点11到0

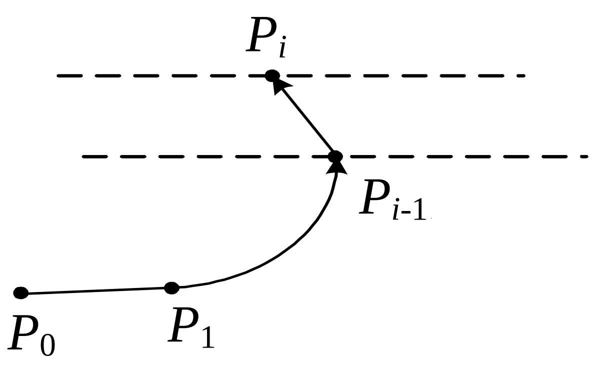


算法的正确性

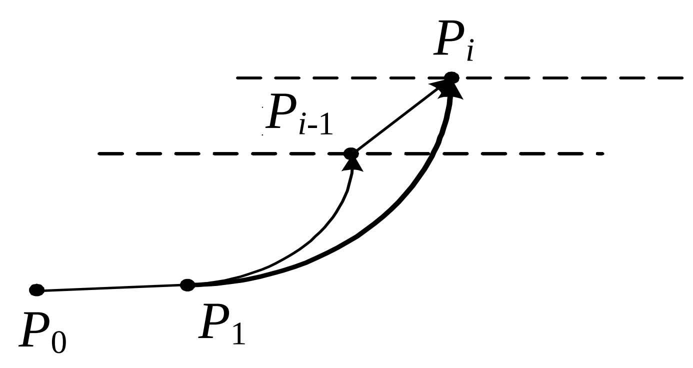
- 输出点一定在原点集中，只需证明
  - 结果是凸的
  - 包含所有点
- 结果的凸性：左转已经保证
- 水平序下右链关于y轴单调，有左右两侧
- 左右链等价，因此
  - 只需证：右链完成时，所有点都在其左边
  - 则完整凸包正确

右链的正确性

- 排序后点集为p_0, p_1, … p_{n-1}，CH(i)为p_0, … p_i的局部凸包
- 命题：点p_0, p_1, …p_{n-1}在CH(n-1)左侧
- 基础：p_0和p_1在右链上（左侧的特殊情况）
- 归纳假设：p_0到p_{i-1}的点都在CH(i-1)左侧
- 加入的新点p_i在CH(i)上，只需考虑以前的是否仍在左侧
  - 向左转(a)：正确，因为y是递增的，
  - 向右转(b)：正确，因为新链比老链更“右”



(a) 向左转的情形



(b) 向右转的情形

上-下Andrew算法

Graham's Scan

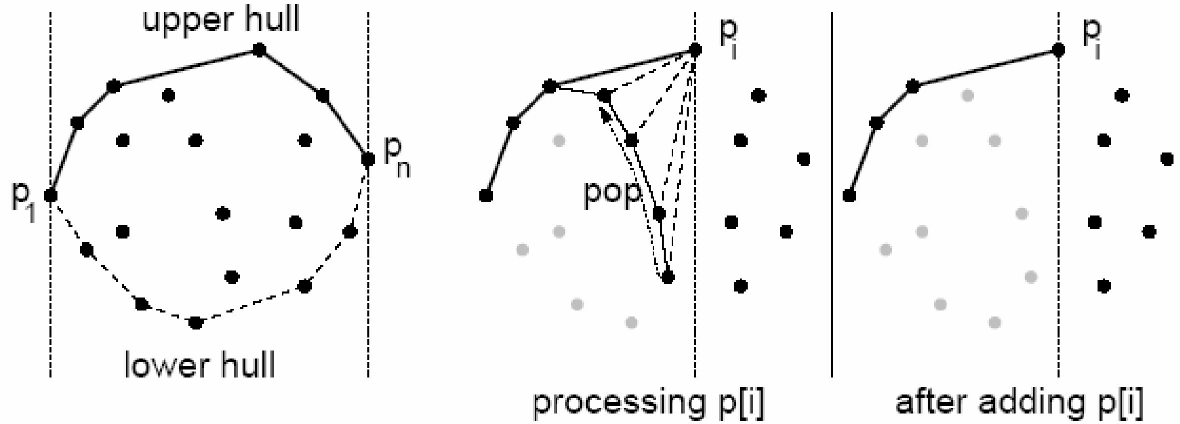
(1) Sort the points according to increasing order of their x-coordinates, denoted $\langle p_1, p_2, \ldots, p_n \rangle$.

(2) Push p_1 and then p_2 onto U.

(3) for $i = 3$ to n do:

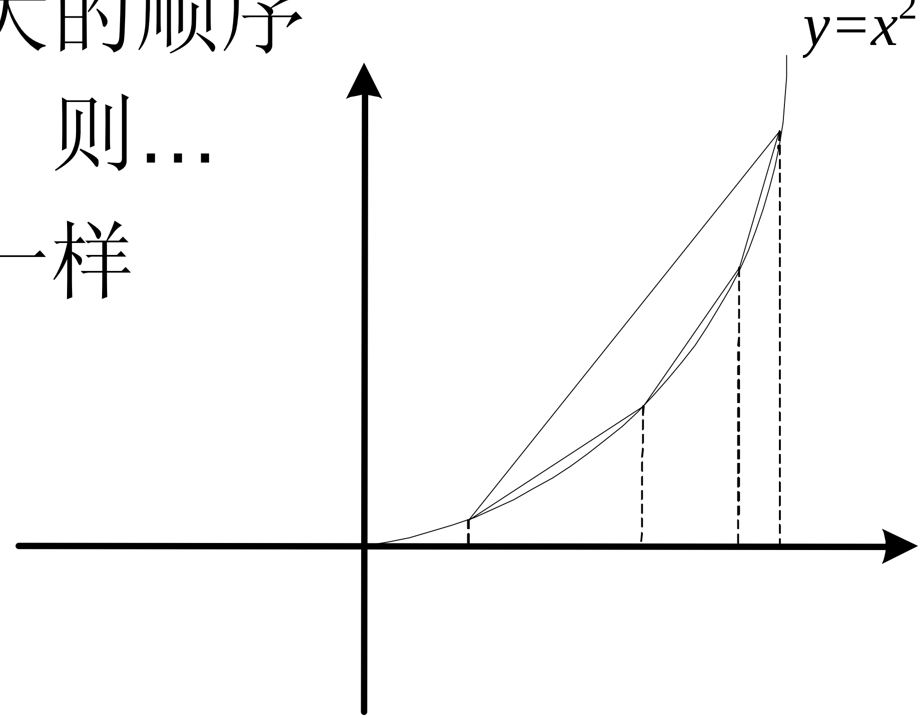
 (a) while size(U) ≥ 2 and Orient(p_i, first(U), second(U)) ≤ 0, pop U.

 (b) Push p_i onto U.



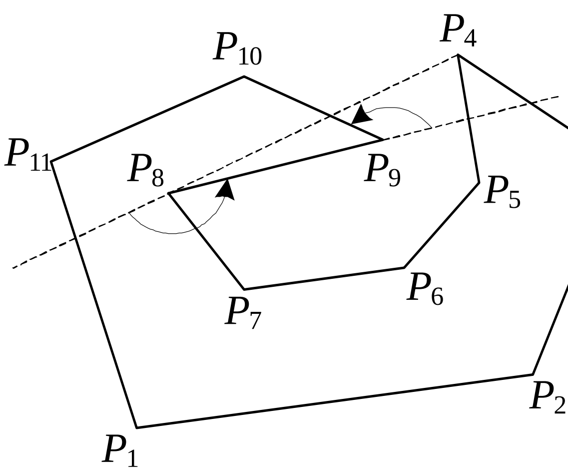
凸包的时间下界

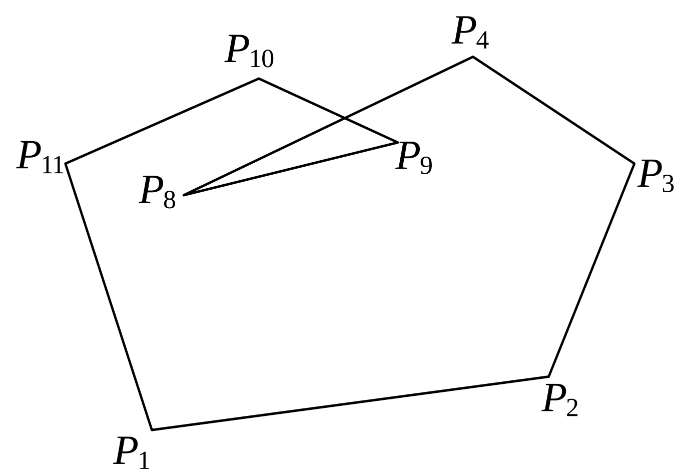
- 任给n个点，顺序随意
- 构造点(x_i, x_i^2)
- 凸包输出为从小到大的顺序
- 如果凸包比排序快，则…
- 即使不要求顺序，一样



多边形凸包

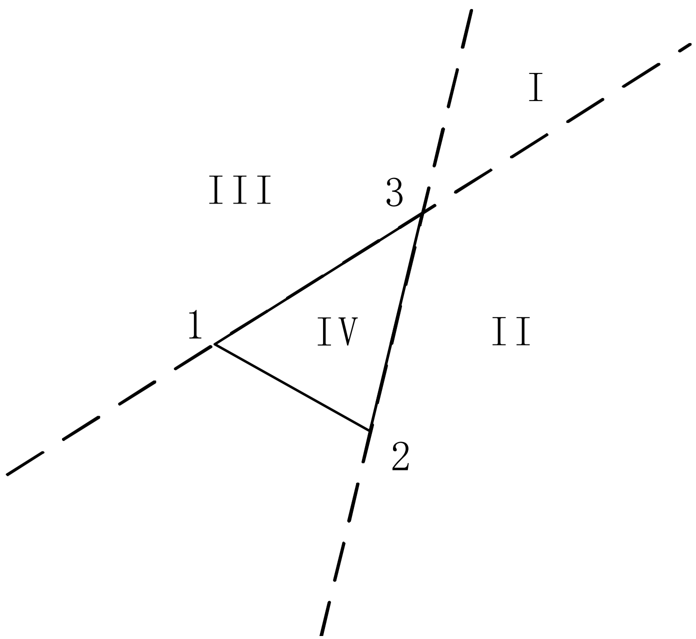
- 多边形有个序，可以直接用吗？
- Graham算法直接应用的反例



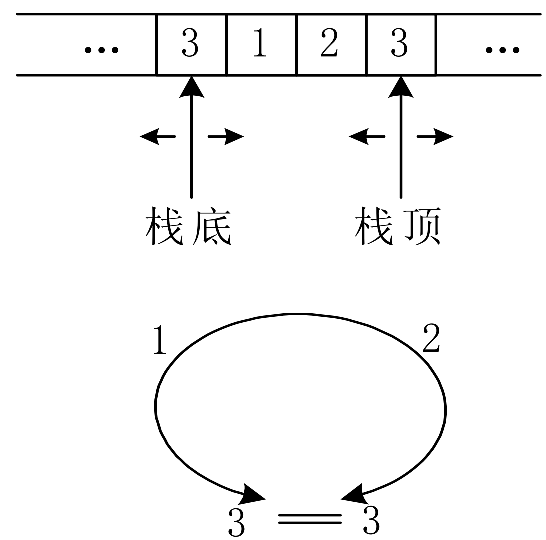


Melkman算法

- 使用deque（双头栈），栈里成右手系
- 两端都可以进行Graham维护



(a) 初始情况



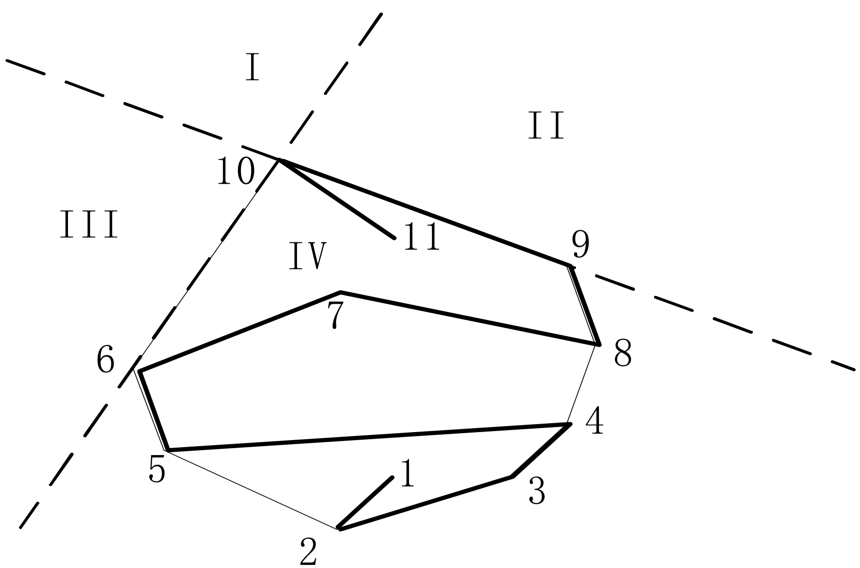
(b) 两头栈与项链

Melkman算法

- 新增一个点
  - 可能落在Ⅰ，Ⅱ，Ⅲ，Ⅳ四个区域
  - 若是Ⅳ，走入了凸包内部，“误入歧途”，应该忽略，考虑下一个点，直到走出这个区域。
  - 若是Ⅱ，则对栈顶一端进行Graham-Scan式维护（退栈直至左转），对栈底加上4（因为对栈底来说1—3—4是凸的）。
  - 若是Ⅲ，则与Ⅱ对称，对栈底进行维护（注意这里是退栈直至右转——或者统一地说，退栈直至“凸转”），同时对栈顶加上4（凸）。
  - 若是Ⅰ，则当作同时是Ⅱ，Ⅲ处理——即两头都是进行维护，这也是直观的。
- 于是如此循环，考虑所有点。

Melkman算法

- 简单的说，把当前最后一个点看为终点
  - 同时考虑逆时针和顺时针的最后一条边
  - 看是延伸还是退栈



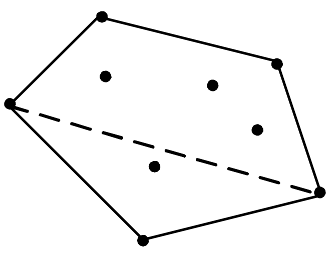
(a) 到 11 为止的图

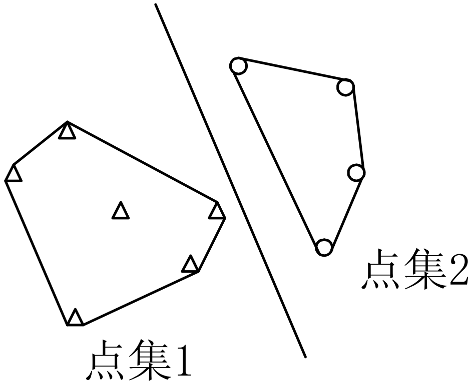
加入	1 2 3
4	4 1 2 3 4
5	5 2 3 4 5
6	6 5 2 3 4 6
7	7 6 5 2 3 4 7
8	8 7 6 5 2 3 4 8
9	9 7 6 5 2 3 4 8 9
10	10 6 5 2 3 4 8 9 10
11	10 6 5 2 3 4 8 9 10

(b) 每访问一个顶点 deque 的变化过程

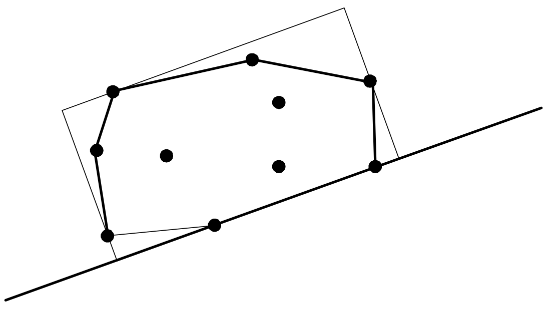
凸包的应用

- 应用1：点集直径（及对踵点介绍）
- 应用2：最小外接矩形（及旋转卡壳）
- 应用3：点集剖分（及凸多边形交）


(a)

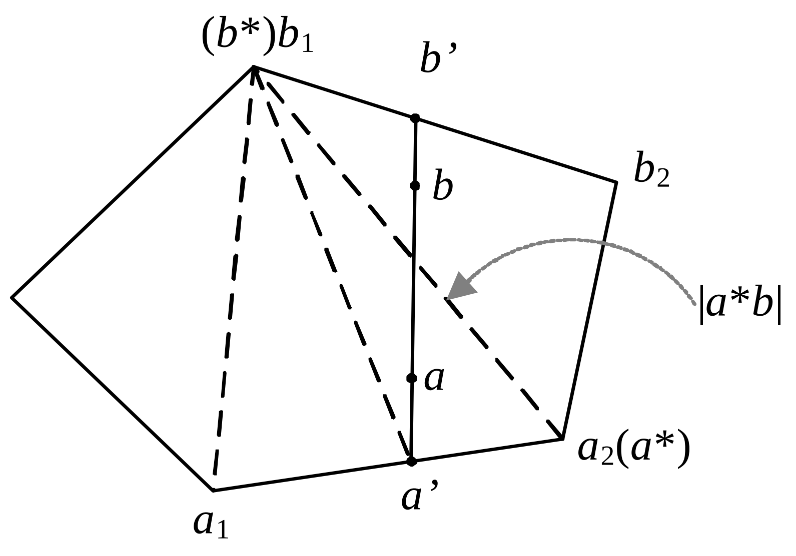


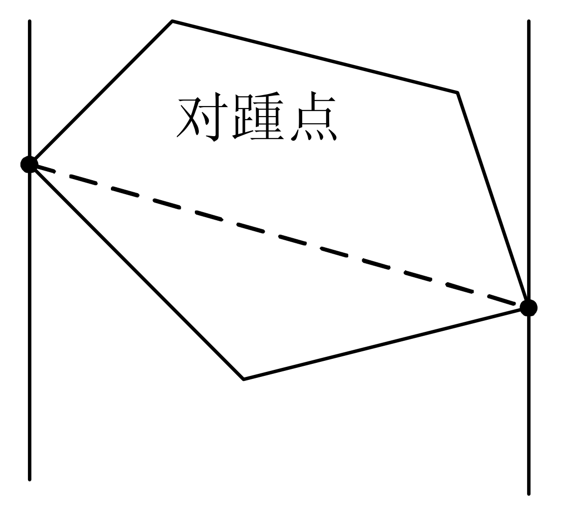
(b)


(c)

应用1——点集的直径

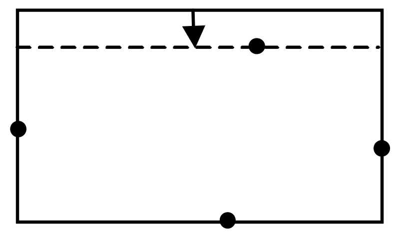
- 结论1: 直径端点必为凸包顶点
- 结论2: 对踵点(antipodal)才可能是直径端点
- 问题: 如何求对踵点?



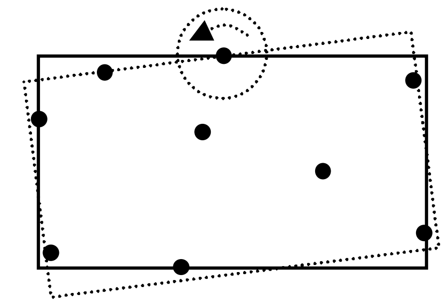


应用2——最小外接矩形

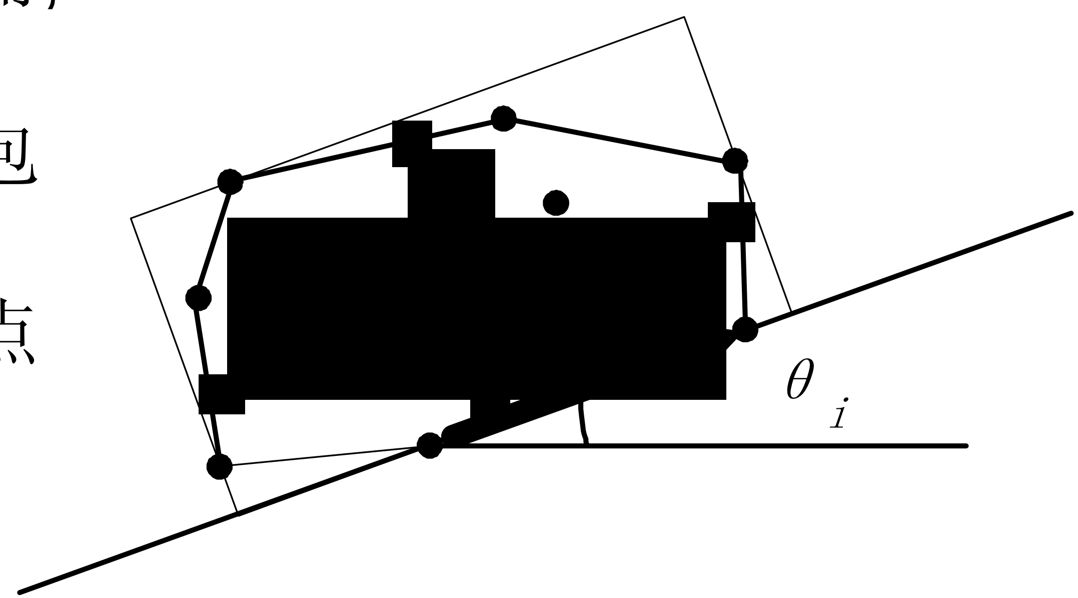
- 若边平行坐标轴…
- 模拟旋转法
- 问题: 模拟法不精确? 连续→离散!
- 初始解法: 枚举倾角, 求跨度, 总$O(n^3)$
- 先求凸包: 对于凸包每条边…$O(n^2)$
- 进一步讨论: 对踵点距离远, 因此…



(a) 每边上至少有一点

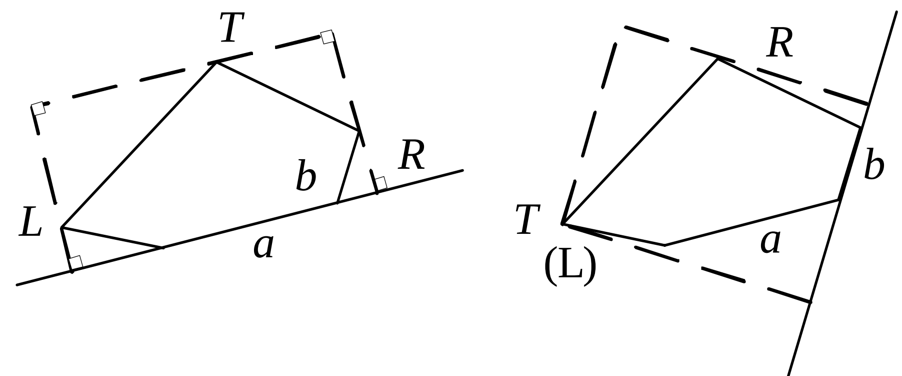


(b) 至少有一边上有两点的证明

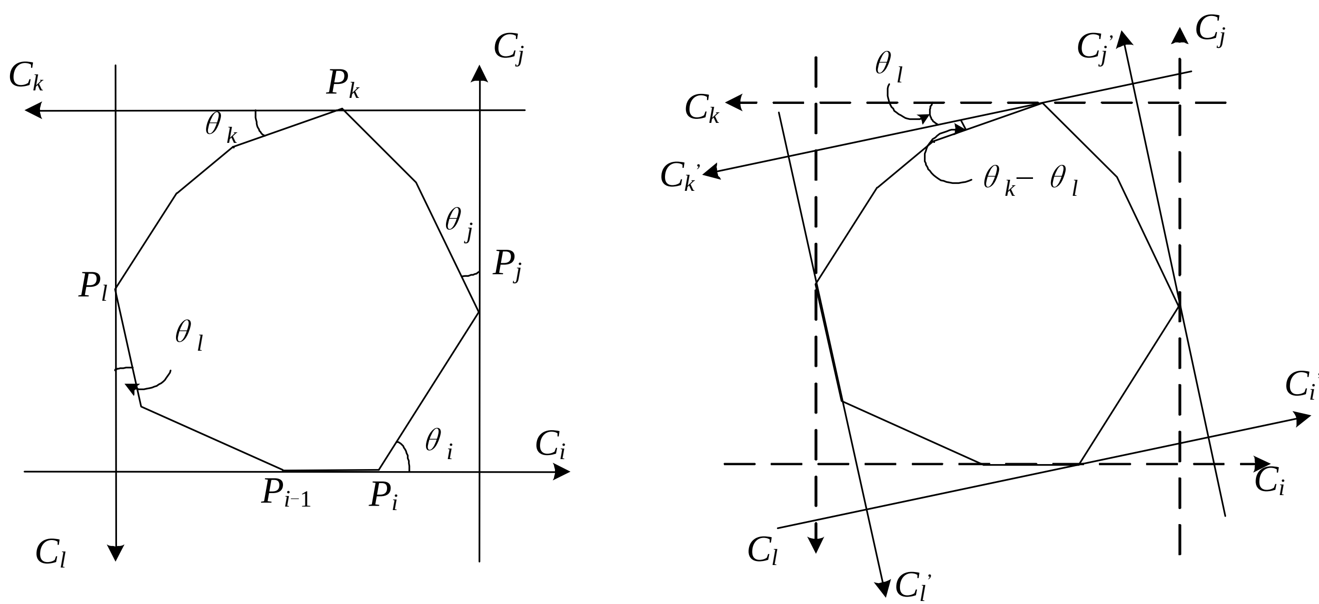


旋转卡壳

- 矩形运动方式: 旋转
- 算法
  - 最小角行进
  - 其他三个卡住的点不变
  - 累计超过90度, 结束
- 扫描O(n)



(a) (b)



应用3——点集分离问题

- 点集分离⇔凸包分离
- 注意包含的情形