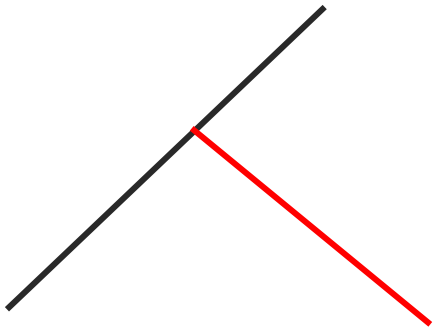
计算几何

谢迪

2007.11

二维平面内线段规范相交的判定

- 规范相交
  - 两条线段恰有唯一一个不是端点的公共点。



二维平面内线段规范相交的判定

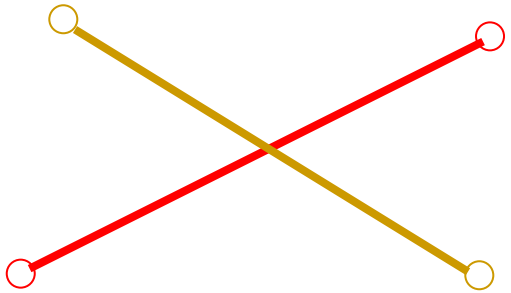
- 解析几何解法
  - 列直线方程：$Ax + By + C = 0$
  - 判断解的情况
    - 若无解则平行
    - 无穷多解——共线
    - 唯一解
      - 判断是否分别在两条线段的内部

二维平面内线段规范相交的判定

- 解析几何解法的问题
  - 求解方程需要浮点除法运算
    - 浮点误差
      - 特别是接近平行时
    - 浮点除法运算速度
  - 总体效率
    - 需要比较顶点
    - 交点没有用

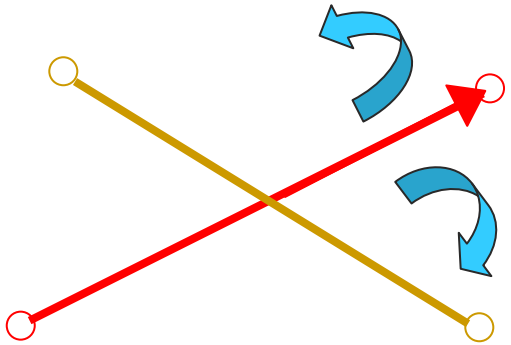
二维平面内线段规范相交的判定

- 计算几何的算法
  - 仅需要加、减、乘
  - 不求交点
- 如何判断？
  - 两条线段相交时，每条线段两个端点都在另一条线段的异侧



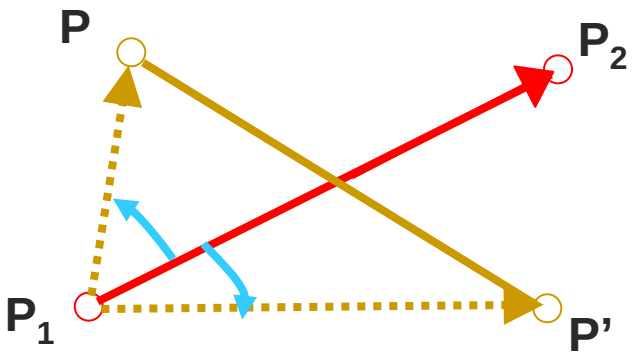
二维平面内线段规范相交的判定

- 两条线段相交时，每条线段两个端点都在另一条线段的异侧
  - 判断异侧
  - 有向线段
  - 判断一个点是在有向线段的左边还是右边



二维平面内线段规范相交的判定

- 判断一个点是在有向线段的左边还是右边
  - 添加一条辅助向量
  - 若要判断P的相对位置，只要判断向量P_1P在向量P_1P_2的顺时针还是逆时针

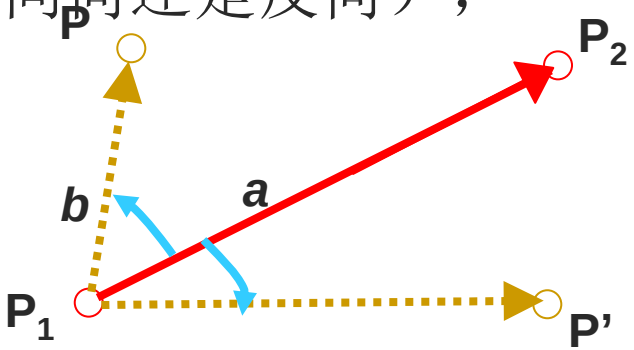


二维平面内线段规范相交的判定

- 向量的叉积

$$\vec{a}\times\vec{b}=x_a y_b - x_b y_a = \begin{vmatrix} x_a & y_a \\ x_b & y_b \end{vmatrix} = -(x_b y_a - x_a y_b) = -\vec{b}\times\vec{a}$$

  - 向量a到向量b成逆时针时，上述结果大于0；
  - 向量a到向量b成顺时针时，上述结果小于0；
  - 向量a和向量b共线时（不论同向还是反向），上述结果等于0。

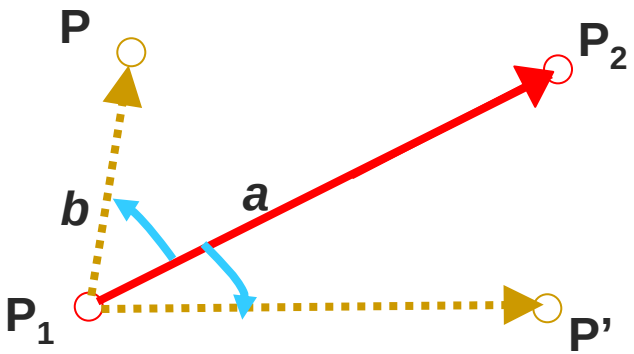


二维平面内线段规范相交的判定

- 向量的叉积

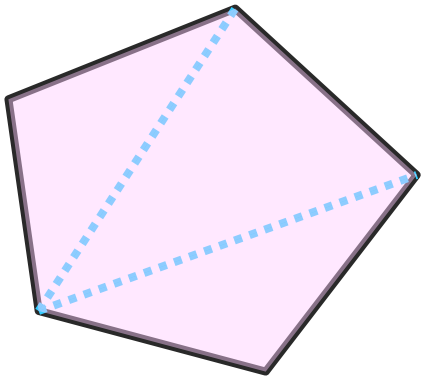
$$\vec{a}\times\vec{b}=x_a y_b - x_b y_a = \begin{vmatrix} x_a & y_a \\ x_b & y_b \end{vmatrix} = -(x_b y_a - x_a y_b) = -\vec{b}\times\vec{a}$$

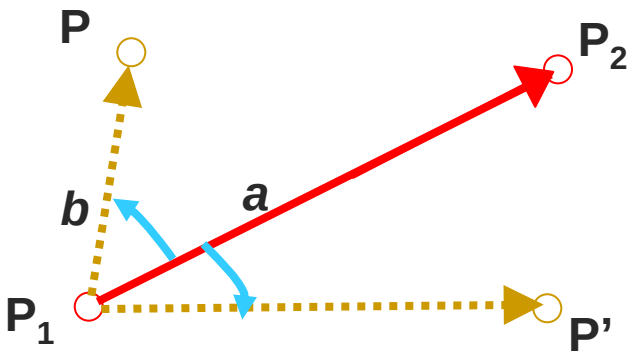
  - 本质上是有向面积
  - $a \times b = |a||b| \sin\theta$



二维平面内线段规范相交的判定

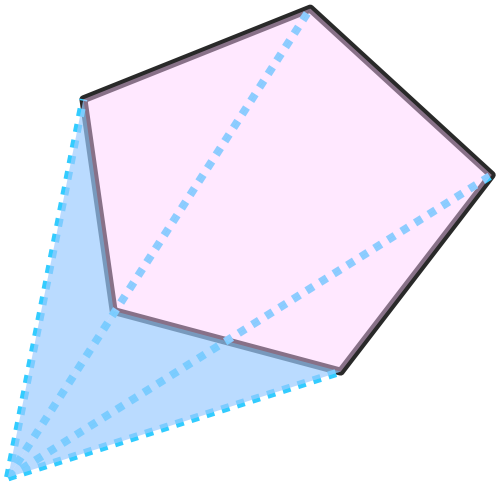
- 有向面积的应用
  - 求凸多边形的面积
  - 将凸多边形用三角剖分，然后求所有三角形面积之和。





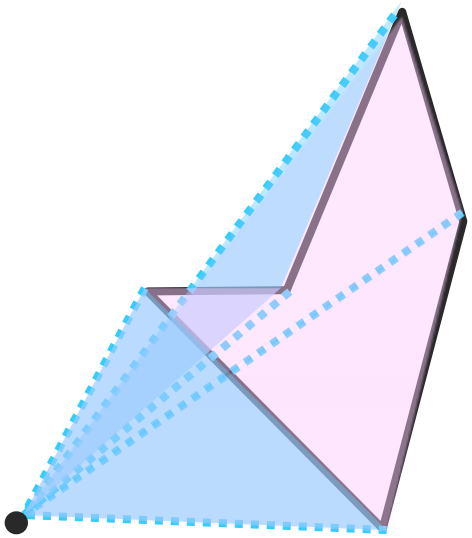
二维平面内线段规范相交的判定

- 有向面积的应用
  - 求凸多边形的面积
  - 任取一点（为计算方便可取原点），将它与所有的多边形上的顶点都连上辅助线
  - 依逆时针方向扫描顶点，依次求出当前顶点和下一个顶点以及原点构成的三角形的有向面积，即叉积/2。并将所有的面积相加即可。



二维平面内线段规范相交的判定

- 有向面积的应用
  - 任意多边形的面积
  - 与前面一种方法一样就可以求出
  - 可见，有向面积的引入极大的方便了计算。



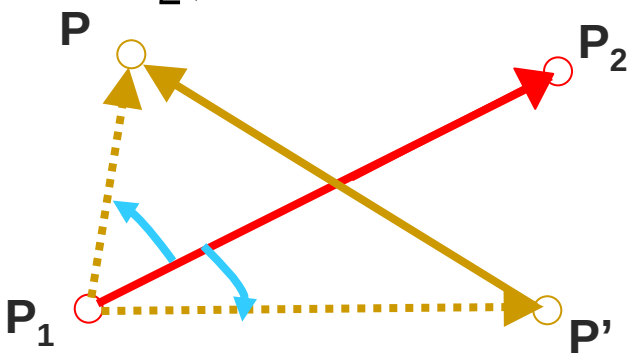
二维平面内线段规范相交的判定

- 规范相交的计算几何判定
  - P和P’在向量P_1P_2的异侧，即

$$(\overrightarrow{P_1P_2}\times\overrightarrow{P_1P})\bullet(\overrightarrow{P_1P_2}\times\overrightarrow{P_1P'})<0$$

  - 且P_1和P_2在向量PP’的异侧，即

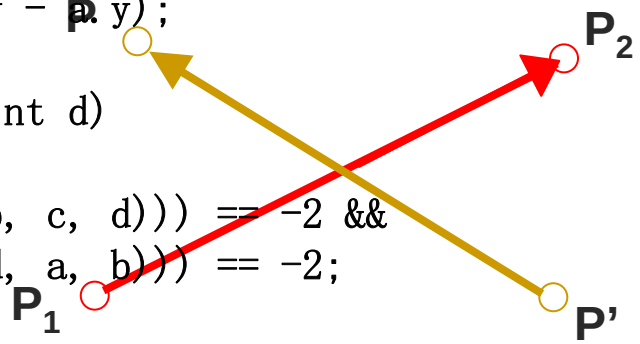
$$(\overrightarrow{P'P}\times\overrightarrow{P'P_1})\bullet(\overrightarrow{P'P}\times\overrightarrow{P'P_2})<0$$



二维平面内线段规范相交的判定

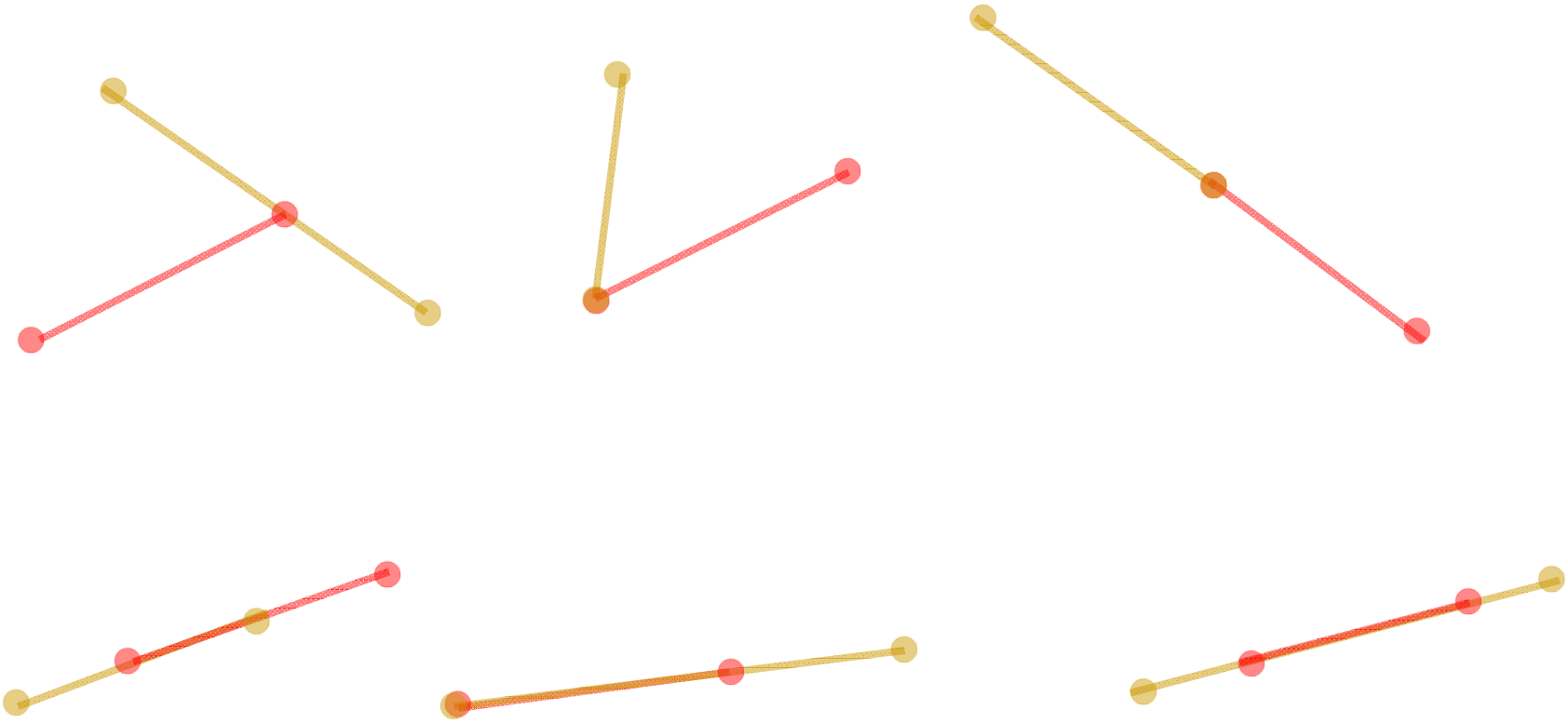
- 代码实现

```
typedef struct { double x, y; } Point;
int dblcmp(double d)
{
  if (fabs(d) < precision) return 0;
  return (d > 0) ? 1 : -1;
}
double det(double x1, double y1, double x2, double y2)
{
  return x1 * y2 - x2 * y1;
}
double cross(Point a, Point b, Point c)
{
  return det(b.x - a.x, b.y - a.y, c.x - a.x, c.y - a.y);
}
int segcrossSimple(Point a, Point b, Point c, Point d)
{
  return (dblcmp(cross(a, c, d)) ^ dblcmp(cross(b, c, d))) == -2 &&
         (dblcmp(cross(c, a, b)) ^ dblcmp(cross(d, a, b))) == -2;
}
```



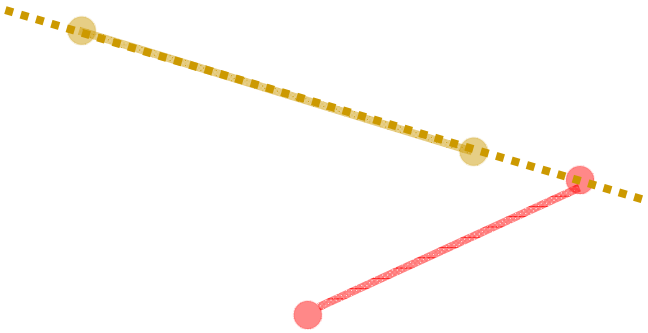
二维平面内线段非规范相交的判定

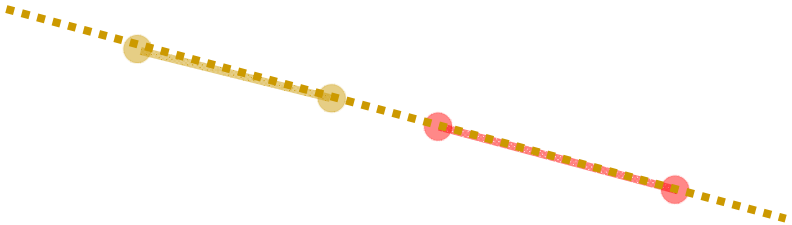
- 非规范相交的几种情况



二维平面内线段非规范相交的判定

- 非规范相交的几种情况
  - 它们没有被判断为规范相交是因为它们在叉乘过程中，都至少有一次叉乘结果是0。
  - 但是叉乘结果为0并不一定就是非规范相交。因为叉乘结果为0表示是某个点在某向量所在的直线上。如下情况也可能使叉乘结果是0，尽管它们并不是非规范相交或者规范相交。





二维平面内线段非规范相交的判定

- 非规范相交的判定的思路
  - 某个端点P在另外一条线段P_1P_2所在的直线上
    - 即$P_1P_2 \times P_1P=0$
  - 且该端点P在线段P_1P_2上
    - 即min(P1.x, P2.x) ≤ P.x ≤ max(P1.x, P2.x)且
    - min(P1.y, P2.y) ≤ P.y ≤ max(P1.y, P2.y)

二维平面内线段非规范相交的判定

- 代码实现

```
// ab与cd: 0 - 不相交; 1 - 规范相交; 2 - 不规范相交
int segcross(Point a, Point b, Point c, Point d)
{
    int d1, d2, d3, d4;
    d1 = dblcmp(cross(a, b, c));
    d2 = dblcmp(cross(a, b, d));
    d3 = dblcmp(cross(c, d, a));
    d4 = dblcmp(cross(c, d, b));
    if (d1 == 0 && betweenCmp(c, a, b) <= 0 ||
        d2 == 0 && betweenCmp(d, a, b) <= 0 ||
        d3 == 0 && betweenCmp(a, c, d) <= 0 ||
        d4 == 0 && betweenCmp(b, c, d) <= 0)
        return 2;
    return 0;
}
```

这里还需要插入规范相交的判定，即d1*d2<0而且d3*d4<0。

二维平面内线段非规范相交的判定

- 代码实现(续)

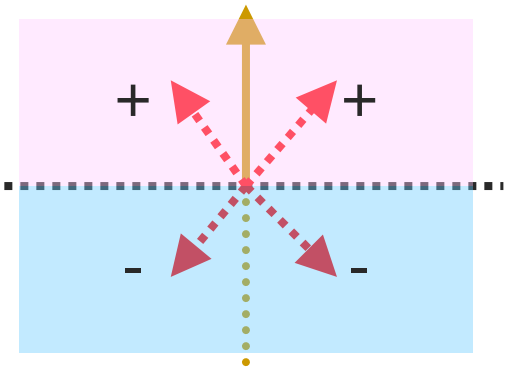
```
// 判断a是否在bc范围内
int betweenCmp(Point a, Point b, Point c)
{
    if (fabs(b.x - c.x) > fabs(b.y - c.y))
        return xyCmp(a.x, min(b.x, c.x), max(b.x, c.x));
    else
        return xyCmp(a.y, min(b.y, c.y), max(b.y, c.y));
}
int xyCmp(double p, double mini, double maxi)
{
    return dblcmp(p - mini) * dblcmp(p - maxi);
}
```

二维平面内线段非规范相交的判定

- 判断顶点在线段上的另外一种方法
  - 可以用点乘

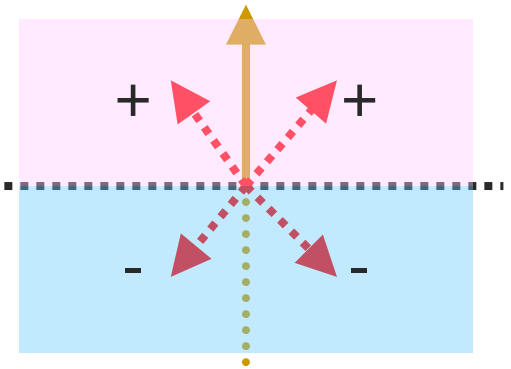
$$\vec{a}\cdot\vec{b}=x_a x_b+y_a y_b$$

$$\vec{a}\times\vec{b}=x_a y_b-x_b y_a=\begin{vmatrix} x_a & y_a \\ x_b & y_b \end{vmatrix}=-(x_b y_a-x_a y_b)=-\vec{b}\times\vec{a}$$

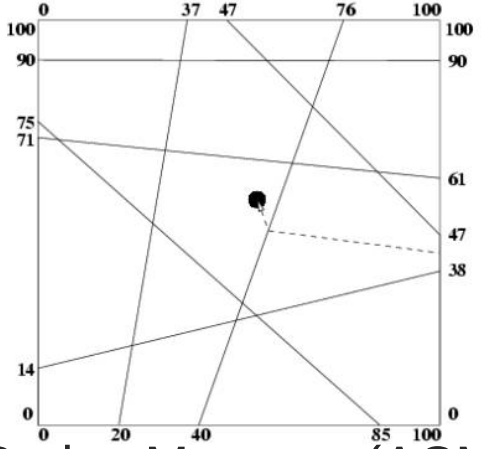


二维平面内线段非规范相交的判定

- 判断顶点在线段上的另外一种方法
  - 假设已经知道P在P_1P_2所在的直线上
    - 即$P_1P_2 \times P_1P=0$
  - 则判断PP_1与PP_2的点积结果的符号
    - 若$PP_1 \bullet PP_2<0$，则P在P_1P_2内部
    - 若$PP_1 \bullet PP_2>0$，则P在P_1P_2外部
    - 若$PP_1 \bullet PP_2=0$，则P与P_1或P_2重合



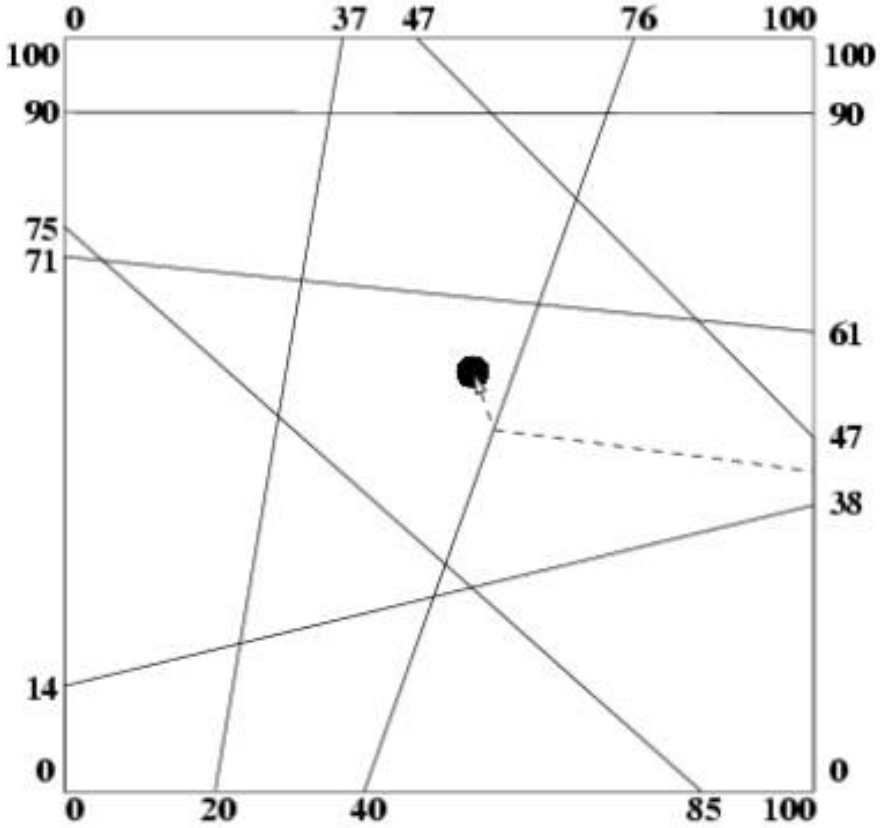
线段相交的应用



- POJ 1066 《Treasure Hunt》
- Archeologists from the Antiquities and Curios Museum (ACM) have flown to Egypt to examine the great pyramid of Key-Ops. Using state-of-the-art technology they are able to determine that the lower floor of the pyramid is constructed from a series of straightline walls, which intersect to form numerous enclosed chambers. Currently, no doors exist to allow access to any chamber. This state-of-the-art technology has also pinpointed the location of the treasure room. What these dedicated (and greedy) archeologists want to do is blast doors through the walls to get to the treasure room. However, to minimize the damage to the artwork in the intervening chambers (and stay under their government grant for dynamite) they want to blast through the minimum number of doors. For structural integrity purposes, doors should only be blasted at the midpoint of the wall of the room being entered. You are to write a program which determines this minimum number of doors.
- An example is shown below:

线段相交的应用

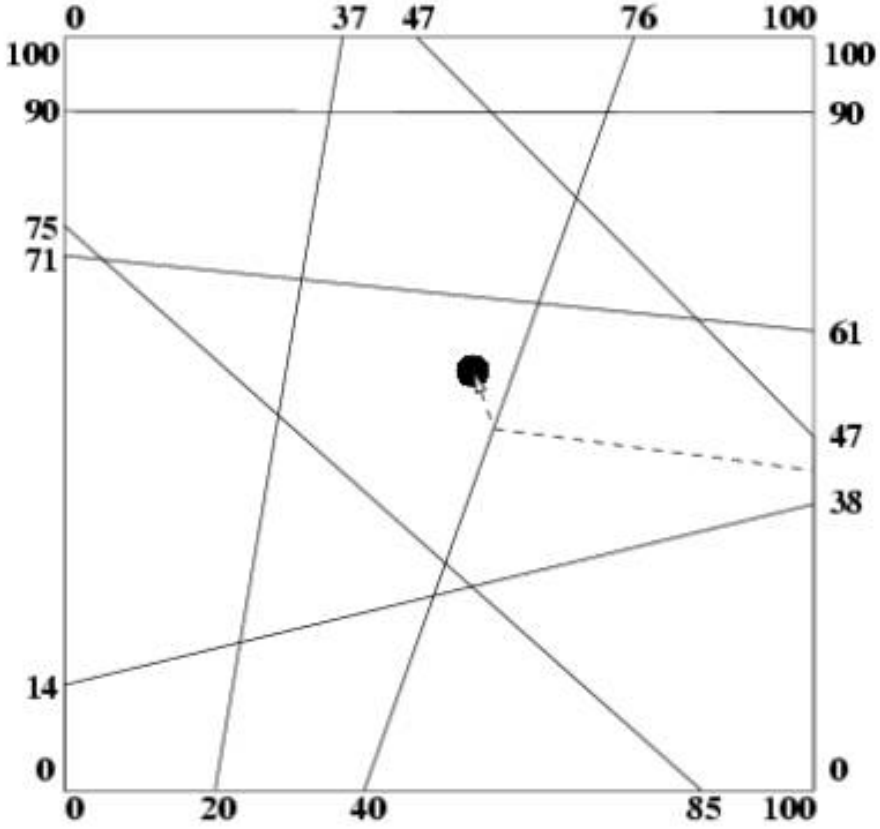
- The input will consist of one c
an integer n (0 <= n <= 30) sp
walls, followed by n lines cont
of each wall x1 y1 x2 y2 . The
pyramid have fixed endpoints
and (100,0) and are not inclu
interior walls always span fro
another exterior wall and are
more than two walls intersect
assume that no two given wa
listing of the interior walls the
containing the floating point c
in the treasure room (guarant



- Print a single line listing the minimum number of doors which need to be created, in the format shown below.

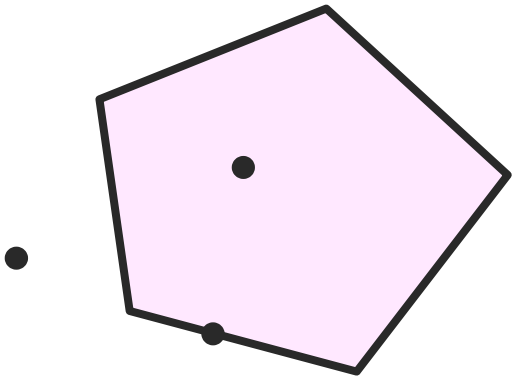
线段相交的应用

- 思路
  - 枚举入口
  - 计算从入口到最终目标需要穿越多少次门
  - 从中找一个最小的
- 计算穿越门的次数
  - 只要计算从入口到最终目标连线与墙相交的次数即可。
  - 为什么？



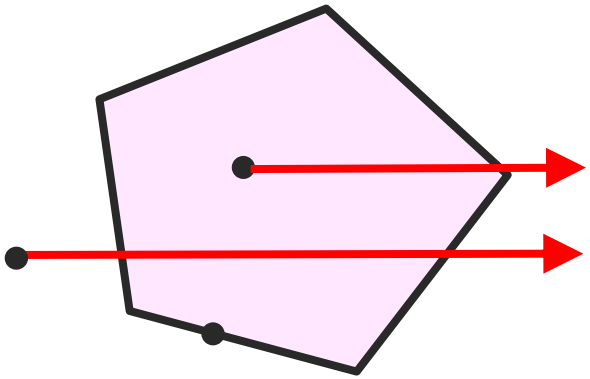
点在形内形外的判断

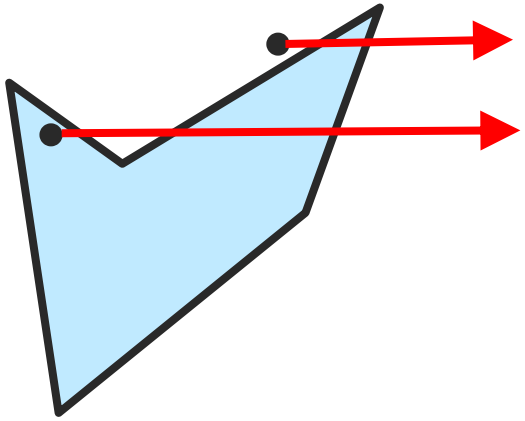
- 点与多边形的位置关系
  - 点在形内
  - 点在形外
  - 点在边界上
- 判断方法
  - 射线法
  - 转角法



点在形内形外的判断

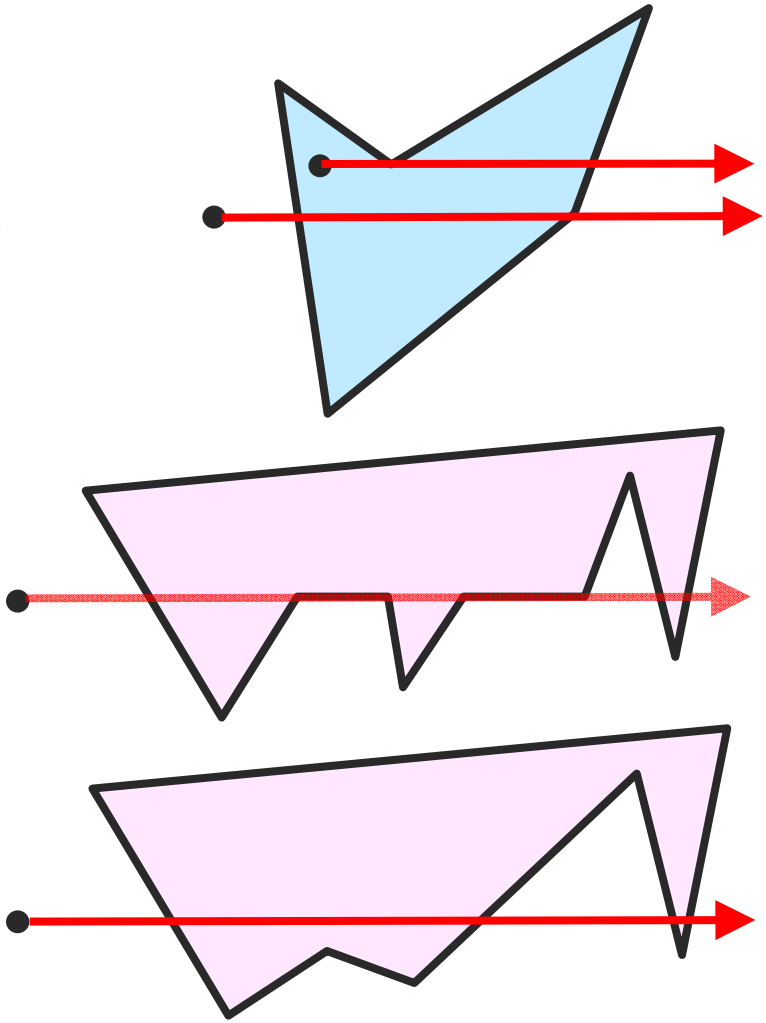
- 射线法
  - 通常取x轴正方向为射线方向
  - 奇数次相交，则在形内
  - 偶数次相交，则在形外
- 对于凹多边形也是可以的





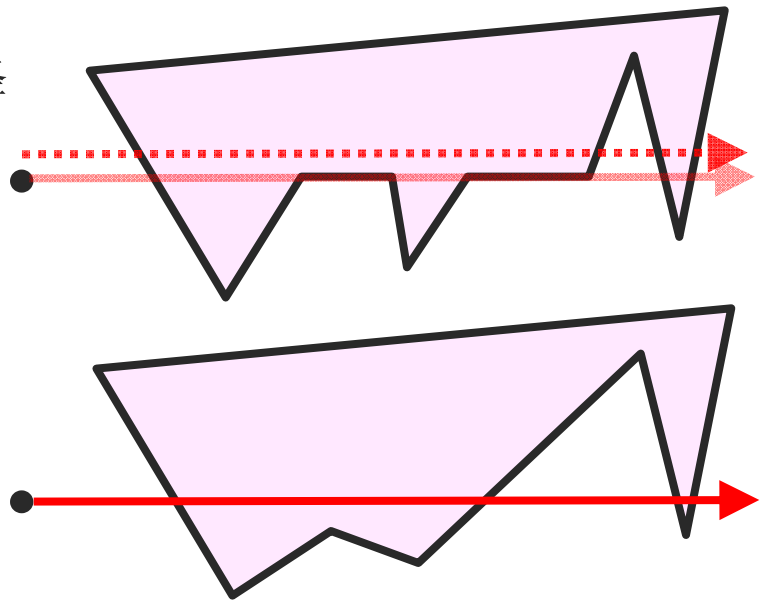
点在形内形外的判断

- 射线法的特殊情况
  - 与顶点相交
    - 与其相邻的端点或者线段在射线的异侧，则认为相交。
    - 否则不认为相交。
  - 与边部分重合
    - 缩点：遇到一个在射线上的点，向后连续跳过所有也在射线上的点，直到第一个不在射线上的点，再用上述条件判断。



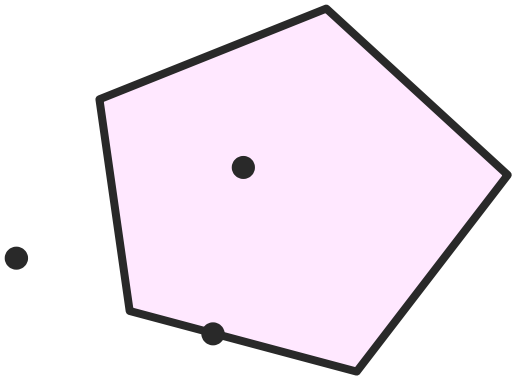
点在形内形外的判断

- 射线法的特殊情况
  - 与边部分重合
    - 缩点法：遇到一个在射线上的点，向后连续跳过所有也在射线上的点，直到第一个不在射线上的点，再用上述条件判断。
    - 平移法：把射线稍微上升或下降一个很小的量。
      - 实际操作时不用真的平移，只需要判断较高的端点高于射线，较低的端点低于射线或恰在射线上。



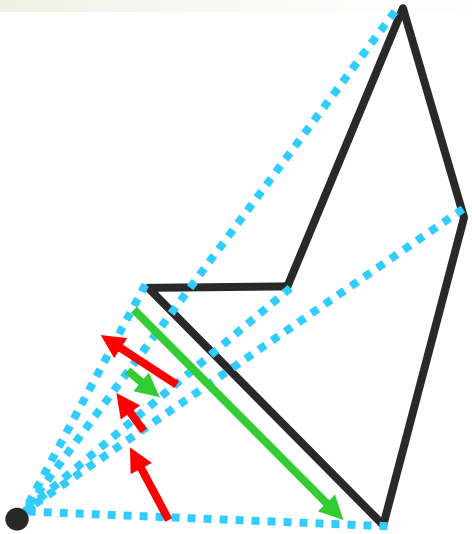
点在形内形外的判断

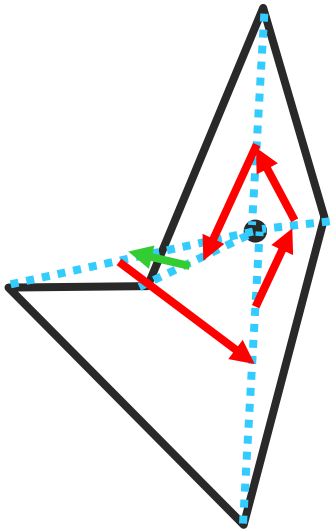
- 射线法的特殊情况
  - 边界情况
    - 可以附加判断
    - 判断是否在边界上或者与多边形顶点重合
      - 叉乘+坐标比较



点在形内形外的判断

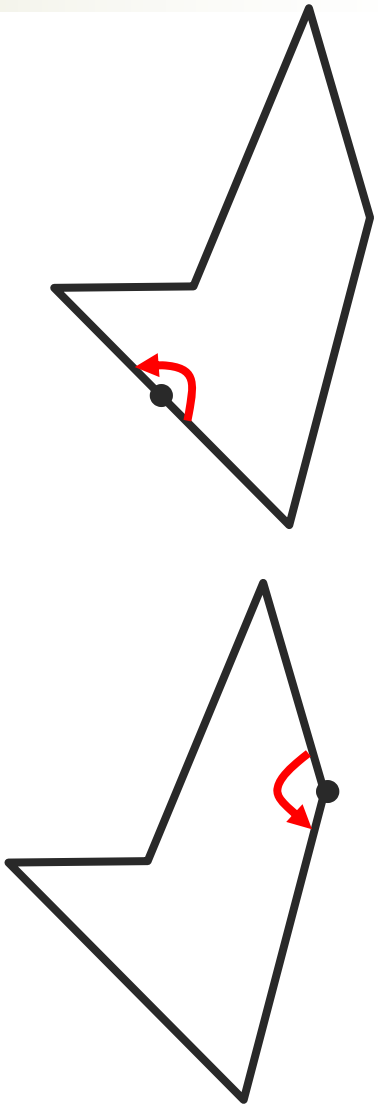
- 转角法
  - 通过沿多边形走一圈，累计绕了给定的点多少角度来判断：
    - 沿正方向转角的代数和为2π，则在内部
    - 沿正方向转角的代数和为0，则在外部





点在形内形外的判断

- 转角法
  - 特殊情况
    - 沿正方向转角的代数和为π，则在边上
    - 沿正方向转角的代数和为大于0小于2π，则在顶点上
    - 当相邻两条线段共线时，只根据角度有可能分辨不出两种情况，可以用点积辅以判断

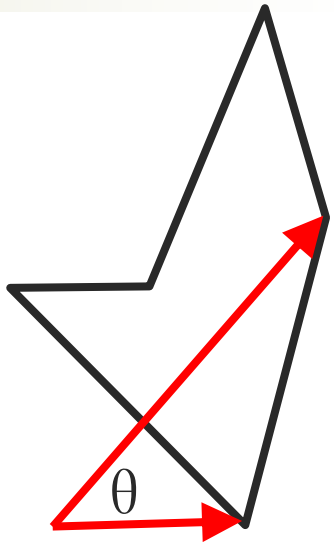


点在形内形外的判断

- 转角法
  - 角度的计算

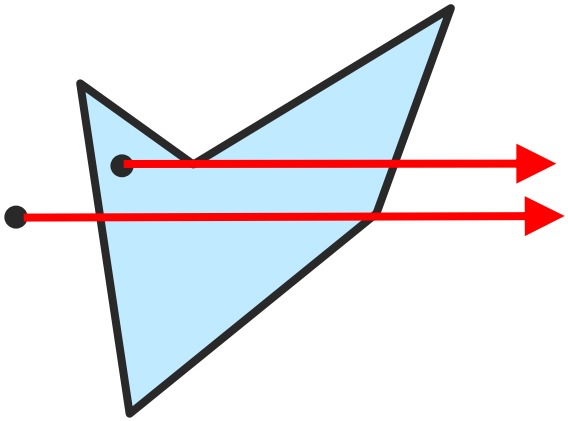
$$\vec{a}\cdot\vec{b}=x_a x_b+y_a y_b=\left|\vec{a}\right|\left|\vec{b}\right|\cos\theta$$

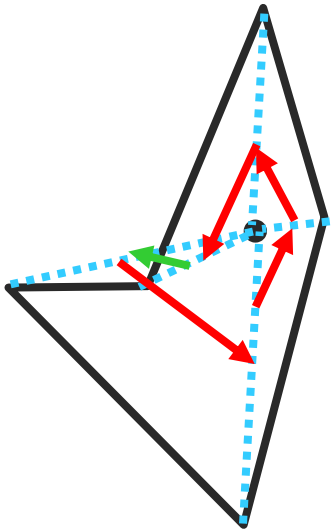
$$\theta=\cos^{-1}\frac{\vec{a}\cdot\vec{b}}{\left|\vec{a}\right|\left|\vec{b}\right|}$$



点在形内形外的判断

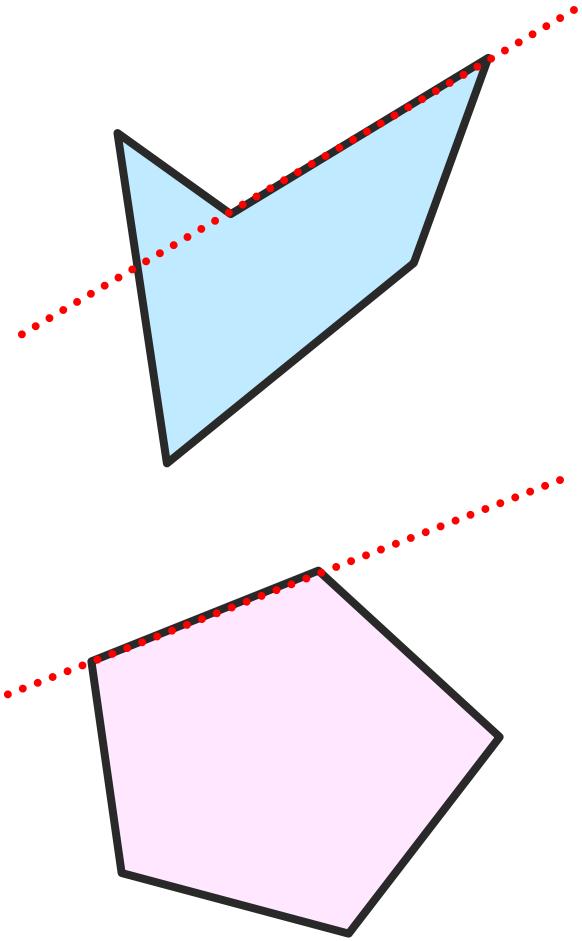
- 射线法还是转角法？
  - 射线法特殊情况的处理比较麻烦，不够优美。
  - 转角法很优美，而且时间复杂度与射线法一样。
  - 但是转角法需要反三角函数、开根、浮点除法的计算。因此实际运算的速度慢很多。





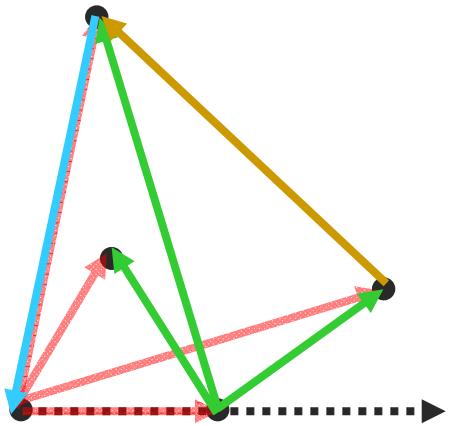
凸包

- 凸多边形
  - 整个图形在任一条边的一侧
- 凸图形
  - 任意两个内点的任一内分点也在内部
- 凸包
  - 对于一个平面点集或者一个多边形，它的凸包指的是包含它的最小凸图形或最小凸区域



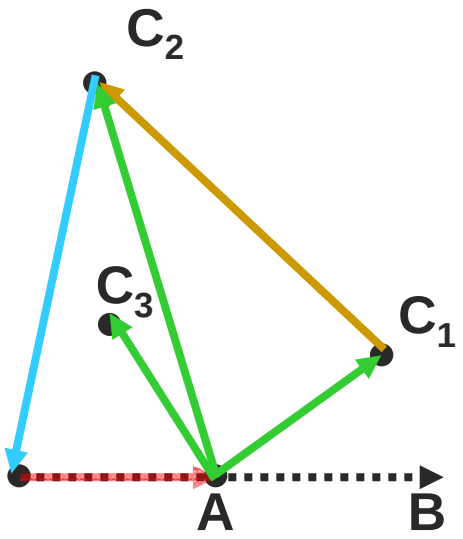
凸包的求法

- 卷包裹法
  - 从最左边的最低点P_0开始
  - 找一个点P_1，使得P_0为起点的水平方向的射线到P_0P_1的角度最小
  - 然后找下一个点P_2,使得P_2P_1到P_1P_0角度最小。
  - ……
  - 则$P_0P_1…P_m$是凸包上的顶点



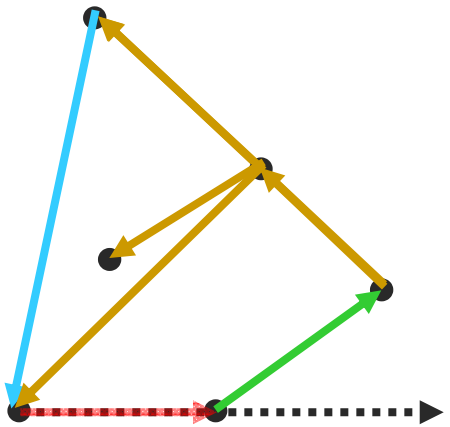
凸包的求法

- 卷包裹法
  - 实际比较的时候，不一定要用角度来衡量。
  - 可以采用叉乘来判断：只要知道相对的方向（顺时针还是逆时针）就可以
    - 比如判断AC_1比AC_2的夹角小，只要判断AC_1在AC_2的右边
  - 这样每次查找需要O(n)的时间复杂度。
  - 因此总的时间复杂度为$O(n^2)$



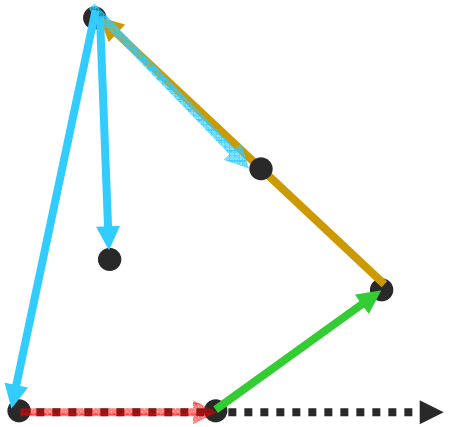
凸包的求法

- 卷包裹法的特殊情况
  - 多点共线的情况
    - 输出包括所有共线点
    - 输出不包括所有共线点
  - 解决：
    - 如果要共线点：则共线的时候，根据与上一个凸包中的点的距离从近到远选取
    - 不要共线点，则从远到近选取



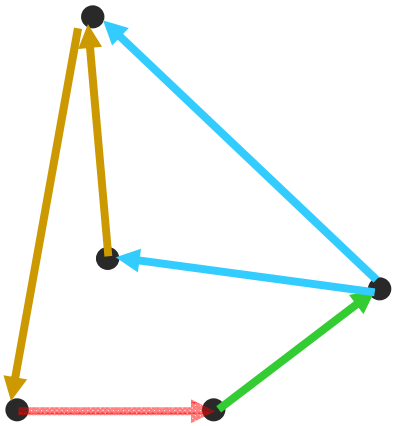
凸包的求法

- 卷包裹法的特殊情况
  - 多点共线的情况
    - 输出包括所有共线点
    - 输出不包括所有共线点
  - 解决：
    - 如果要共线点：则共线的时候，根据与上一个凸包中的点的距离从近到远选取
    - 不要共线点，则从远到近选取



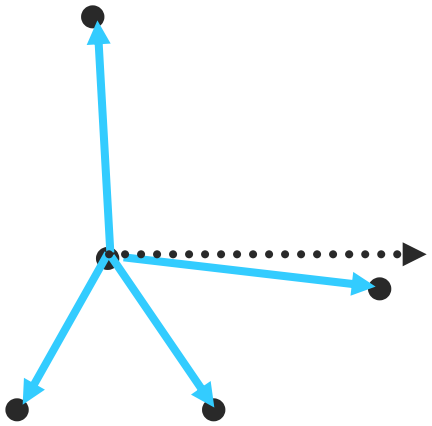
凸包的求法

- Graham-Scan法
  - 卷包裹法每一步都确定性的求出凸包上的一条边
  - Graham-Scan算法每一步是得到一个临时凸包
    - 只要当前点在上一条边的左手方向，就加入这个点
    - 否则回溯，直到新的点在左手方向为止



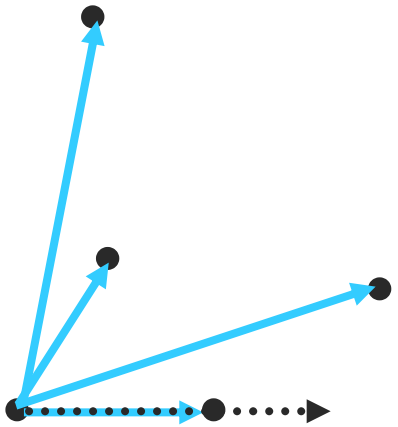
凸包的求法

- Graham-Scan法序的选取
  - 极角序：以一个内部点x为中心，点集中的所有点按关于x的极角逆时针排序，就以x点出发向右延伸，平行于x轴的一条射线为排序的起始位置
  - 但是不便于计算，不能用叉乘，而需要用到三角函数。
  - 还要保证起点一定在凸包上，否则会出错。



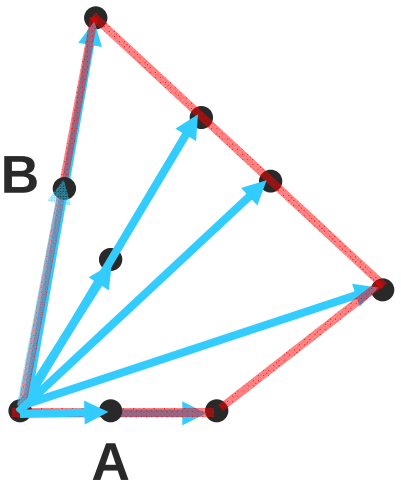
凸包的求法

- Graham-Scan法序的选取
  - 选取最低点中最左的一个作为参考点
  - 用叉乘来排序所有点的相对位置
  - 时间复杂度
    - 排序O(n log n)
    - 扫描O(n)
    - 总的是O(n log n)



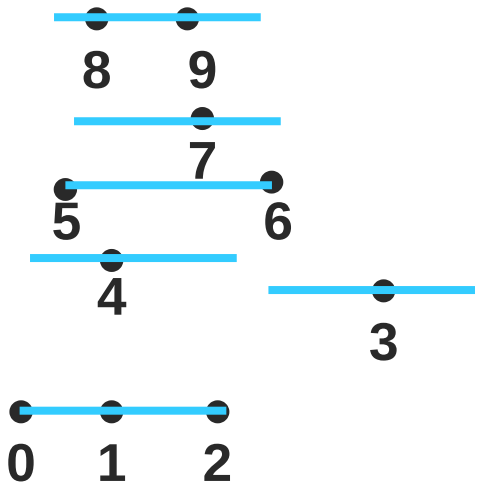
凸包的求法

- Graham-Scan法的特殊情况
  - 重复点
    - 删除
  - 共线点
    - 对于不要求求共线点的情况，可以对叉乘做严格的判定。
    - 对于要求求共线点的情况，没有办法简单而完美的处理：
      - A、B两点没有办法都加入凸包中



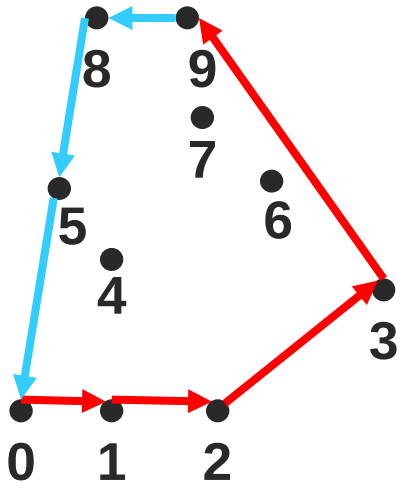
凸包的求法

- Graham-Scan法的另外一种序
  - 用水平序
    - 先按y坐标排
    - y相同的按x坐标排
  - 2次扫描
    - 先从第1个点即0开始
      到最后1个点即9得到右链
    - 再从最后1个点即9开始
      到第1个点即0，不包
      括已经在右链的点



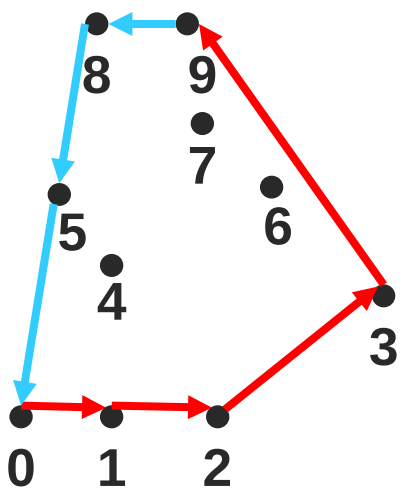
凸包的求法

- Graham-Scan法的另外一种序
  - 用水平序
    - 先按y坐标排
    - y相同的按x坐标排
  - 2次扫描
    - 先从第1个点即0开始
      到最后1个点即9得到右链
    - 再从最后1个点即9开始
      到第1个点即0，不包
      括已经在右链的点



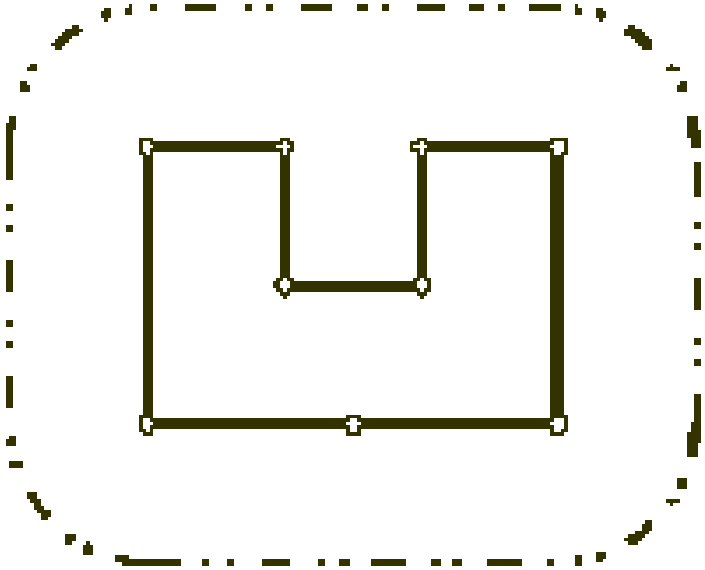
凸包的求法

- Graham-Scan法的另外一种序
  - 处理特殊情况
    - 如果不要共线的点，则严格判断叉乘（即只有在左边才可以）
    - 如果要共线的点，则叉乘等于0即共线也认为可以
  - 直观的理解
  - 很简洁，很完美~



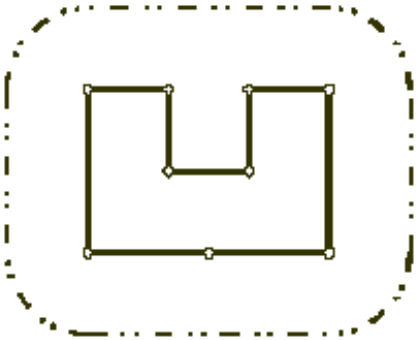
凸包的应用-POJ 1113《Wall》

- Once upon a time there was a greedy King who ordered his chief Architect to build a wall around the King's castle. The King was so greedy, that he would not listen to his Architect's proposals to build a beautiful brick wall with a perfect shape and nice tall towers. Instead, he ordered to build the wall around the whole castle using the least amount of stone and la
the wall should not come closer
certain distance. If the King finds
used more resources to build the
absolutely necessary to satisfy t
then the Architect will loose his h
demanded Architect to introduce
wall listing the exact amount of r
needed to build the wall.



凸包的应用-POJ 1113《Wall》

- Your task is to help poor Architect to save his head, by writing a program that will find the minimum possible length of the wall that he could build around the castle to satisfy King's requirements.
- The task is somewhat simplified by the fact, that the King's castle has a polygonal shape and is situated on a flat ground. The Architect has already established a Cartesian coordinate system and has precisely measured the coordinates of all castle's vertices in feet.



凸包的应用-POJ 1113《Wall》

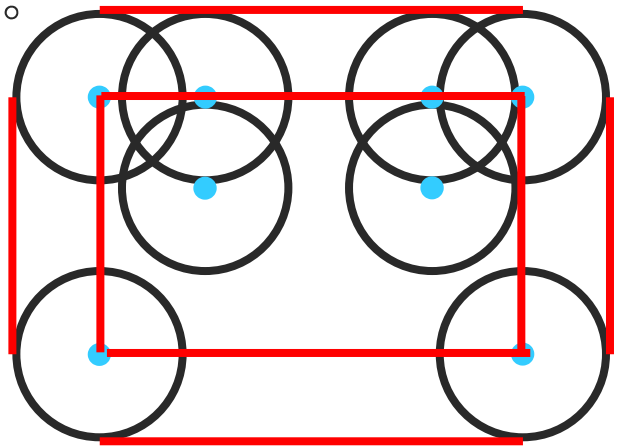
- **INPUT**
- The first line of the input file contains two integer numbers N and L separated by a space. N (3 <= N <= 1000) is the number of vertices in the King's castle, and L (1 <= L <= 1000) is the minimal number of feet that King allows for the wall to come close to the castle.
- Next N lines describe coordinates of castle's vertices in a clockwise order. Each line contains two integer numbers Xi and Yi separated by a space (-10000 <= Xi, Yi <= 10000) that represent the coordinates of ith vertex. All vertices are different and the sides of the castle do not intersect anywhere except for vertices.

凸包的应用-POJ 1113《Wall》

- **OUTPUT**
- Write to the output file the single number that represents the minimal possible length of the wall in feet that could be built around the castle to satisfy King's requirements. You must present the integer number of feet to the King, because the floating numbers are not invented yet. However, you must round the result in such a way, that it is accurate to 8 inches (1 foot is equal to 12 inches), since the King will not tolerate larger error in the estimates.

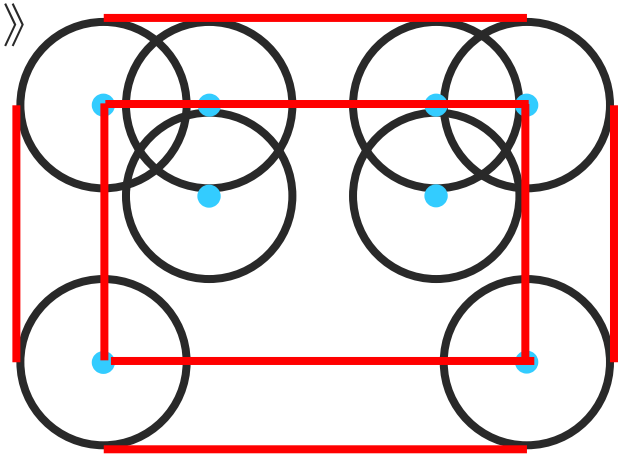
凸包的应用-POJ 1113《Wall》

- 想法
  - 先求凸包
  - 将凸包的点扩展成圆，再将相邻的圆的切线求出来。
  - 外侧切线的长度加上圆弧的长度就是最小值。
- 实际做法
  - 求得凸包长度，加上整圆周长。



凸包的应用-POJ 1113《Wall》

- 凸包的求法
  - 可以用排序+Graham扫描，复杂度O(n log n)，n为顶点个数
  - 因为是给出多边形求凸包，也可以用Melkman，其复杂度为O(n)。
    - 思想是栈顶和栈底都采用Graham Scan来维护凸性。
    - 可参考《算法艺术与信息学竞赛》



讨论题

- POJ 1873
- POJ 2932
- POJ 1556
- POJ 1031
- POJ 1271
- POJ 1418
- POJ 1654
- POJ 1687
- POJ 1688
- POJ 1696
- POJ 1758

参考书

- 《算法艺术与信息学竞赛》
- 《算法导论(Introduction to algorithm)》
- 《计算几何--算法分析与设计》
- 《实用算法的分析与程序设计》