2-D Convex Hull

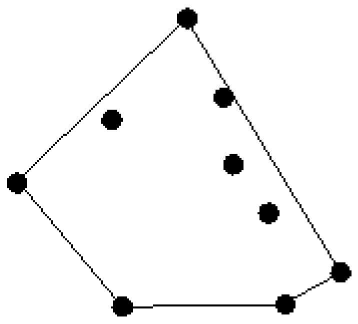
Prerequisite

- Geometry

The Abstraction

Given: A collection of points in the plane, find the convex polygon with smallest area such that each point is contained within (or on the boundary of) the polygon.

Observe that the vertices of such a polygon will be points from the given collection.



Sample Problem: Cow Herding

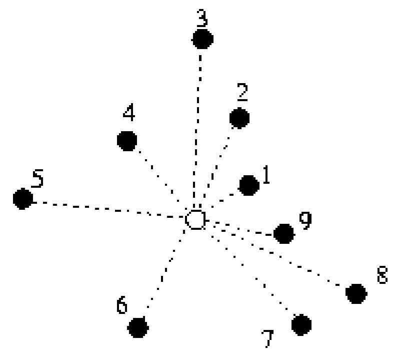
Farmer John wants a fence to keep the pesky local college students from tipping his cows over as they sleep. Each of his cows has a favorite spot for grazing grass, and Farmer John would like the fenced enclosure to include all of these favorite locations. Farmer John would like to enclose a convex figure, as it makes it a lot easier for the cows to make it to their grazing locations.

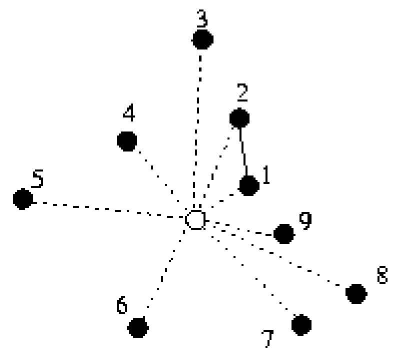
Help Farmer John by calculating the fence which encloses the minimum amount of area while still including all of the cows' favorite dining locations.

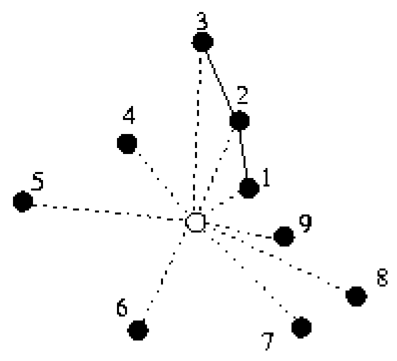
The Algorithm

Several algorithms solve the two dimensional convex hull algorithm. Here, we'll present only the ``gift-wrapping'' algorithm, which is probably the easiest to code and the easiest to remember.

The basic idea is to add the points in clockwise or counterclockwise order around some point within the final answer, checking to see if any angles greater than 180 degrees are created, which would make the final figure concave. Every time three points in a row appear such that the angle created by those three points is greater than 180 degrees, delete the middle point. Check the angle is done by calculating the cross product of vectors along two consecutive edges.







Find a point which will be within the convex hull:

- Calculate the angle that each point makes with the x axis (within the range 0 to 360 degrees)
- Sort the points based on this angle
- Add the first two points
- For every other point except the last point
- Make it the next point in the convex hull
- Check to see if the angle it forms with the previous two points is greater than 180 degrees
    - As long as the angle formed with the last two points is greater than 180 degrees, remove the previous point
- To add the last point
    - Perform the deletion above,
    - Check to see if the angle the last point forms with the previous point and the first point is greater than 180 degrees or if the angle formed with the last point and the first two points is greater than 180 degrees.
    - If the first case is true, remove the last point, and continue checking with the next-to-last point.
    - If the second case is true, remove the first point and continue checking.
    - Stop when neither case is true.
- The adding of the points is linear time, as is the calculation of the angles. Thus, the run-time is dominated by the time of the sort, so gift-wrapping runs in O(*n* log *n*) time, which is provably optimal.

Pseudocode

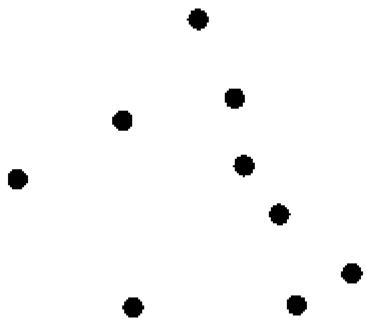
Here is the pseudocode for this convex hull algorithm:

```
 # x(i), y(i) is the x,y position
        #       of the i-th point
        # zcrossprod(v1,v2) -> z component
        #                         of the vectors v1, v2
        #   if zcrossprod(v1,v2) < 0,
        #         then v2 is "right" of v1
        #     since we add counter-clockwise
        #         <0 ->   angle > 180 deg
1      (midx, midy) = (0, 0)
2      For all points i
3          (midx, midy) = (midx, midy) +
                     (x(i)/npoints, y(i)/npoints)
4      For all points i
```

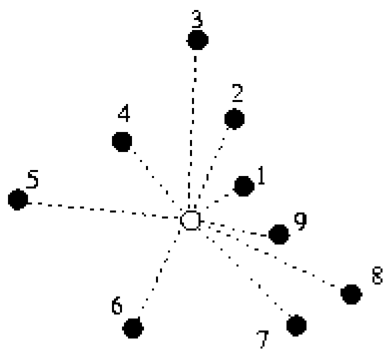
```
        angle(i) = atan2(y(i) - midy,
                                   x(i) - midx)
        perm(i) = i

    sort perm based on the angle() values
     # i.e., angle(perm(0)) <=
                  angle(perm(i)) for all i

     # start making hull
    hull(0) = perm(0)
    hull(1) = perm(1)
    hullpos = 2
    for all points p, perm() order,
                      except perm(npoints - 1)
        while (hullpos > 1 and
                zcrossprod(hull(hullpos-2) -
              hull(hullpos-1),
                hull(hullpos-1) - p) < 0)
                    hullpos = hullpos - 1
            hull(hullpos) = p
            hullpos = hullpos + 1

     # add last point
    p = perm(npoints - 1)
    while (hullpos > 1 and
                zcrossprod(hull(hullpos-2) -
              hull(hullpos-1),
                hull(hullpos-1) - p) < 0)
        hullpos = hullpos - 1

    hullstart = 0
    do
        flag = false
        if (hullpos - hullstart >= 2 and
                    zcrossprod(p -
                  hull(hullpos-1),
                    hull(hullstart) - p) < 0)
            p = hull(hullpos-1)
            hullpos = hullpos - 1
            flag = true
        if (hullpos - hullstart >= 2 and
                  zcrossprod(hull(hullstart) - p,
                  hull(hullstart+1) -
                  hull(hullstart)) < 0)
            hullstart = hullstart + 1
            flag = true
    while flag
    hull(hullpos) = p
    hullpos = hullpos + 1
```

Sample Run

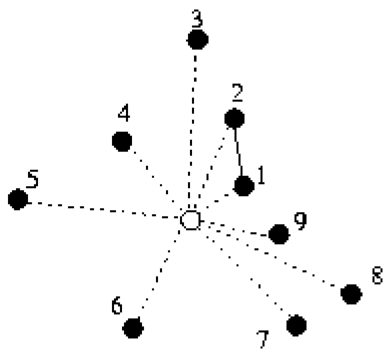
For the sample run, use these points:



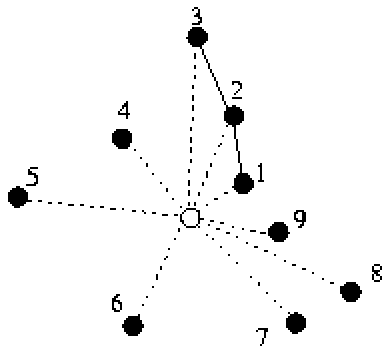
Select a center, calculate angles, and sort.



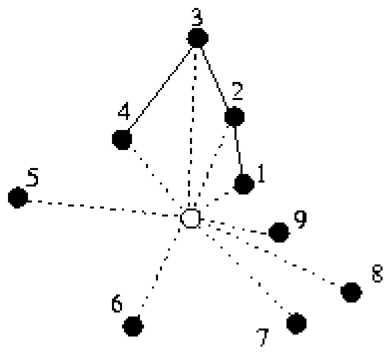
Now, start by adding the first two points.



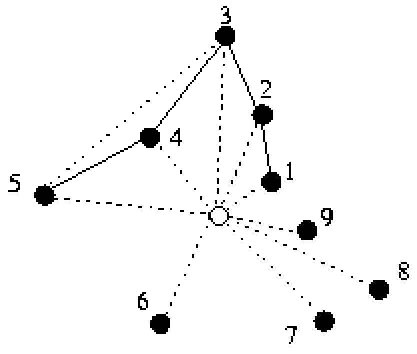
Now, add the third point. Since this does not create an angle of greater than 180 degrees with the first two points, we just have to add the point.



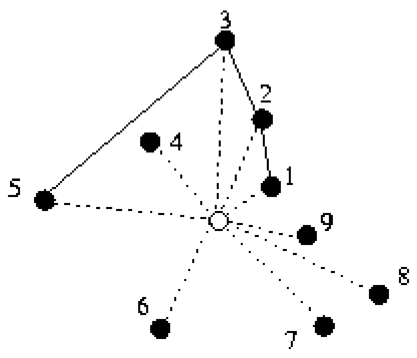
Add the fourth point. Again, no angle greater than 180 degrees was created, so no further work is necessary.



Add the fifth point.

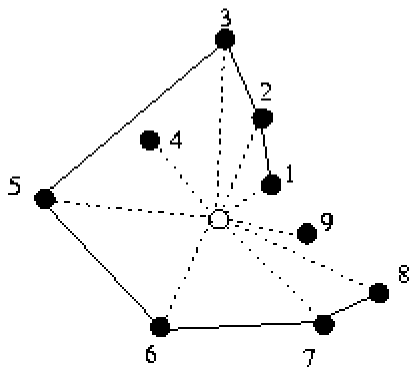


Since the third, fourth, and fifth points together create an angle of greater than 180 degrees (a "right" turn), we remove the fourth point.

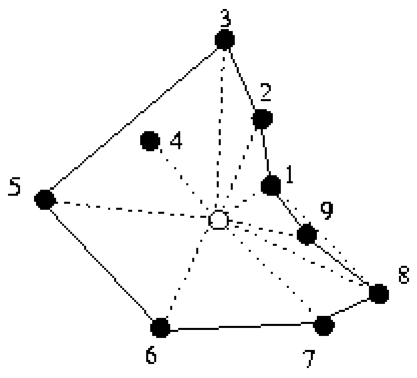


The second, third, and fifth points do not create an angle of greater than 180 degrees, so we are done with adding the fifth point.

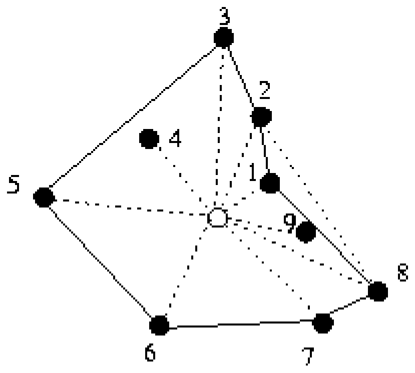
Add the sixth, seventh, and eighth points. None of these require additional work.



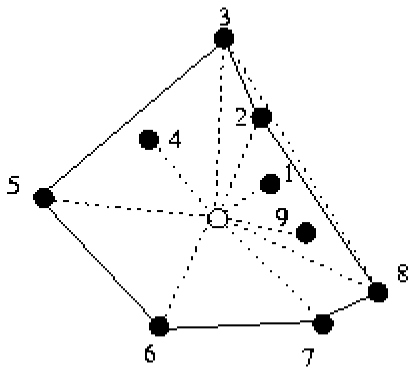
Next, add the last point.



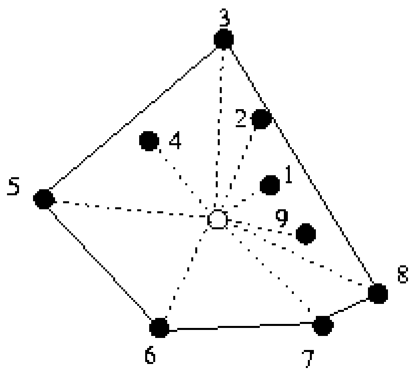
The eighth, ninth, and first point create a "right" turn; remove the ninth point.



The seventh, eighth, and first points are fine, but the eighth, first and second points have a "right" term, so we must remove the first point.



Now the eighth, second and thirst points have a "right" term, so we must remove the second point.



No more violations exist, so we are done, and we have the convex hull of the given points.

Problem Cues

Problems which ask for enclosing points within a polygon are usually convex hull problems. If the problem asks for a minimum area convex polygon or a polygon of minimum circumference, it's almost certainly asking for the convex hull.

Extensions

Unfortunately, this algorithm does not extend in an obvious manner to three dimensions. Fortunately, the three dimensional algorithms are all fairly complicated (four and higher+ dimensions are just plain ugly), so it's unlikely you'll get asked to do it there.

This algorithm no longer works if you limit the created polygon in any way (e.g., no more than n points or must be a rectangle).

Sample Problems

Trees Problem [IOI 1991, problem 2]

Given: a collection of trees, surround it with wire such that you use the minimal amount of wire. Calculate the trees that will be the vertices of the polygon, the length of wire required, and whether the farmer's house, which is at a given location, lies inside of, outside of, or across the polygon.

Analysis: The vertices of the polygon and the length of wire required follow fairly directly above. The farmer's house is specified as an axis-aligned rectangle, so it takes a bit of geometry to determine if all the points are within the convex hull, without the convex hull, or some are within and some without, which gives you the answer you wanted. See the Geometry pamphlet for clues on these kinds of intersections.

Cheapskate Moat Building

Given: A collection of polygon houses, calculate the minimum length moat that encloses all but at most one of them.

Analysis: To enclose a given polygon in convex polygon is equivalent to enclosing all of the vertices of the given polygon. This is very slightly a combination problem, which requires both a loop and a convex hull builder. For each house, delete the house and enclose the remaining vertices in a convex hull. Pick the house whose resulting convex hull is smallest. Note that only deleting houses which share a vertex with the convex hull of the entire set would help if some of the houses don’t share a vertex with that convex hull.

Two Dimensional Convex Hull

二维凸包

译 by Felicia Crazy

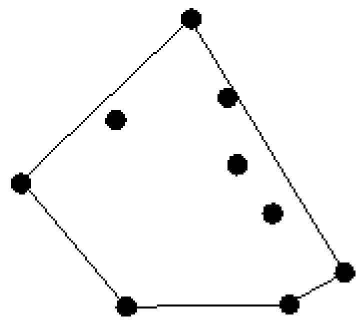
准备知识

- 几何学

数学模型

给出平面内的点集,找出面积最小的凸多边形,使得这些点在这个多边形之内(或者在它的边上）。

可以看出，多边形的顶点必须是给定点集中的点。



例题：放牧奶牛

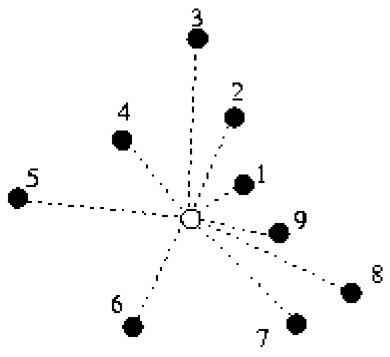
农夫约翰想要修建一个栅栏,来防止讨厌的当地大学生在他的奶牛们睡觉的时候把它们掀翻。他的每头奶牛都有一个最喜欢的吃草点，农夫约翰想要把这些点都围在栅栏内。农夫约翰要围出一个凸的形状，因为这样更容易把奶牛赶进牧场。

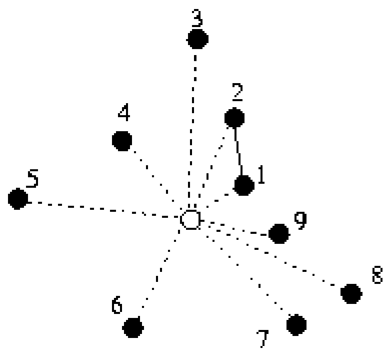
帮助农夫约翰确定面积最小的而且包括所有奶牛喜爱的吃草点的栅栏形状。

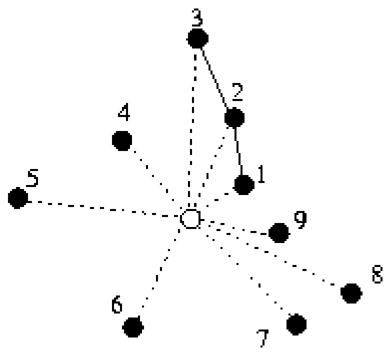
算法

解决二维凸包问题有好几种算法。这里，我们只介绍比较容易编码和记忆的“卷包裹”算法（其实就是我们所说的格拉汉扫描法——译者注）。

算法的基本思想是在一个肯定会在凸包内的点周围不断地由顺时针或逆时针方向增加顶点，并确保每个内角都小于 180 度（保证最终答案是凸的）。如果三个连续的顶点构成的角度大于 180 度，删掉中间的点。可以用两个沿着多边形边的连续向量的叉积来判断角度是否大于 180 度。







凸包算法流程：

- 找出一个必定会在凸包内的中点
- 计算每个点和中点的连线与 x 轴的夹角（在 0——360 度的范围内）
- 根据这些夹角对顶点排序
- 加入最初的两个顶点
- 对于除最后一个顶点以外的其余顶点
- 让其成为凸包上的下一个顶点
- 检查它和前面两个顶点组成的角是否大于 180 度

    - 如果它和前面两个顶点组成的角大于 180 度，那么把它前面那个顶点删掉
- 加入最后一个顶点
    - 完成上述的删除任务
    - 检查最后一个顶点和它的前一个顶点和第一个顶点所组成的角是否大于 180 度，或者最后一个顶点和第一、第二个顶点组成的角是否大于 180 度。
    - 如果第一种情况为真，删除最后一个顶点，并且检查倒数第二个顶点。
    - 如果第二种情况为真，删除第一个顶点，继续检查。
    - 当两种情况都不为真时，停止。
- 由于角度的计算方式，增加顶点的时间复杂度是线性的（就是我们所说的 O(n) ）。因此，运行的时间复杂度决定于排序的时间复杂度，所以“卷包裹法”的时间复杂度是 O(*n* log *n*)，这是最优的。

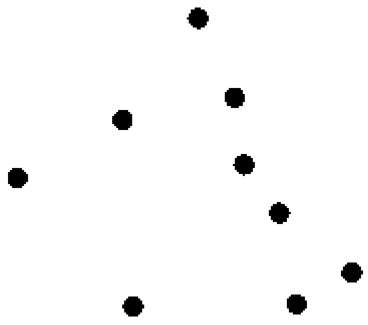
伪代码

这里是凸包算法的伪代码：

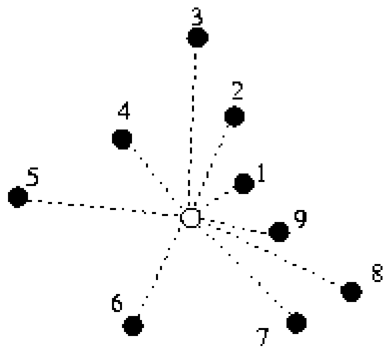
```
0    # x(i), y(i) 是第 i 个顶点的 x,y 坐标
     # atan2 在 Pascal 中是 arctan2，-pi<=结果<=pi
     # zcrossprod(v1,v2) 为 v1,v2 的叉积的 z 分量
     #  如果 zcrossprod(v1,v2) < 0,
     #      那么 v2 在 v1 的“右边”(好像指顺时针方向)
     #   因为我们逆时针加入点
     #      如果 zcrossprod(v1,v2) < 0
     #         则 angle > 180°
 1   (midx, midy) = (0, 0)
 2   For all points i
 3     (midx, midy) = (midx, midy) +
            (x(i)/npoints, y(i)/npoints)
 4   For all points i
 5     angle(i) = atan2(y(i) - midy,
                        x(i) - midx)
 6     perm(i) = i

 7   把 perm 基于 angle() 排序
     # 其中 angle(perm(0)) <= angle(perm(i)) for all i

     # 开始构造凸包
 8   hull(0) = perm(0)
 9   hull(1) = perm(1)
10   hullpos = 2
11   for 对所有点 p, 根据 perm() 的次序,
           除 perm(npoints - 1) 以外
  while (hullpos > 1 and
     zcrossprod(hull(hullpos-2) -
     hull(hullpos-1),
     hull(hullpos-1) - p) < 0)
        hullpos = hullpos - 1
    hull(hullpos) = p
    hullpos = hullpos + 1

# 加入最后一个点
p = perm(npoints - 1)
while (hullpos > 1 and
     zcrossprod(hull(hullpos-2) -
     hull(hullpos-1),
     hull(hullpos-1) - p) < 0)
  hullpos = hullpos - 1

hullstart = 0
do
  flag = false
  if (hullpos - hullstart >= 2 and
       zcrossprod(p -
       hull(hullpos-1),
       hull(hullstart) - p) < 0)
    p = hull(hullpos-1)
    hullpos = hullpos - 1
    flag = true
  if (hullpos - hullstart >= 2 and
      zcrossprod(hull(hullstart) - p,
      hull(hullstart+1) -
      hull(hullstart)) < 0)
     hullstart = hullstart + 1
     flag = true
while flag
hull(hullpos) = p
hullpos = hullpos + 1
```

跟踪

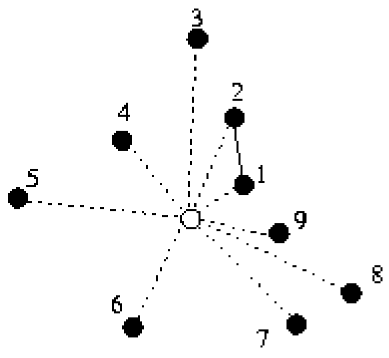
对于下面的跟踪，我们使用这些顶点：



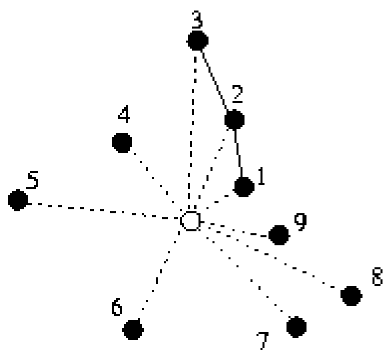
选择一个中心，计算夹角，并排序。



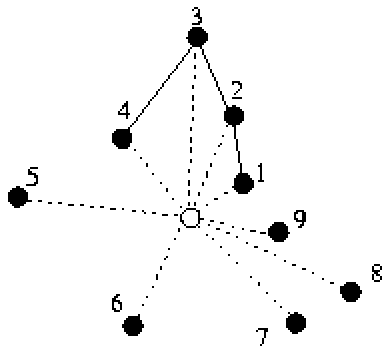
现在，从加入最初的两个顶点开始。



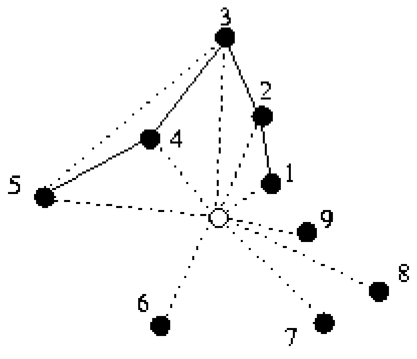
现在，加入第三个顶点。由于这步操作没有和前两个顶点构成一个大于 180 度的角，我们只需加入这个顶点。



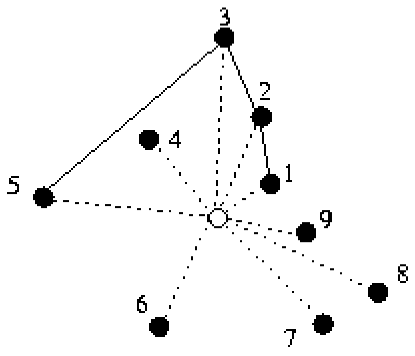
加入第四个顶点。这一次没有构成大于 180 度的角，所以不必做更多的工作。



加入第五个顶点。

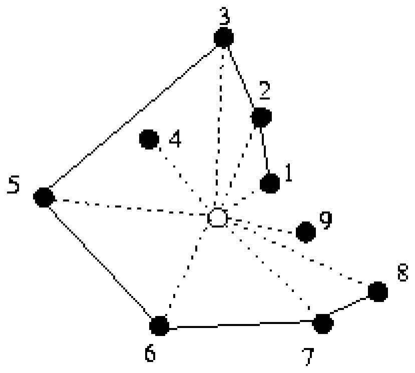


由于第三个，第四个，和第五个顶点构成了一个大于 180 度的角（一个“向右的”转弯），我们删去第四个顶点。

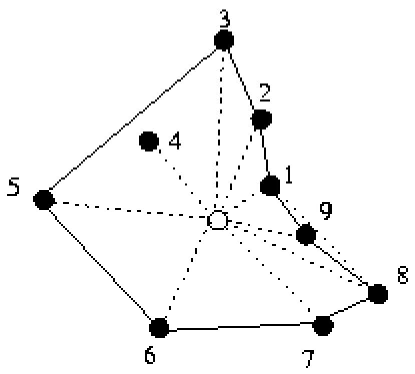


第二个，第三个，和第四个顶点没有构成一个大于 180 度的角，所以我们完成了加入第五个顶点的任务。

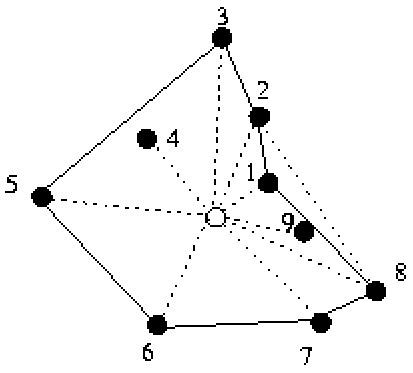
加入第六个，第七个，和第八 8 个顶点。这个过程中没有附加的工作。



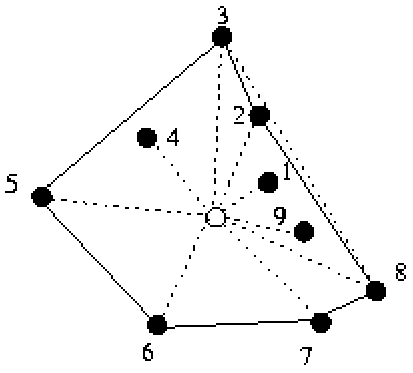
接着，加入最后一个顶点。



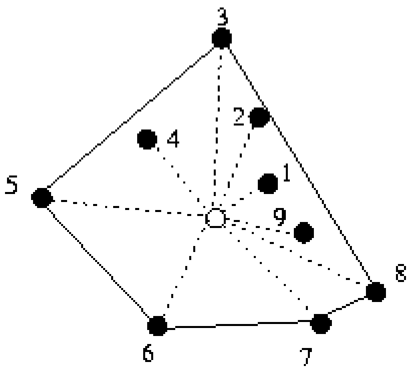
第八个，第九个，和第一个顶点构成“右转”；删除第九个顶点。



第七个，第八个，和第一个顶点已经完成了，可是第八个，第一个，和第二个顶点构成“右转”，所以我们必须删除第一个顶点。



现在，第八个，第二个，和第三个顶点构成“右转”，所以我们又要删除第二个顶点。



剩下的顶点并不违反“左转”原则，所以我们已经完成了任务，并且建立了给出

顶点的凸包。

问题提示

类似把顶点放入多边形的题目通常是求凸包。如果题目要求一个面积最小的凸多边形，或者周长最小的凸多边形，那么我们几乎可以确定是要求凸包了。

推广

不幸的是，这个算法不能简单地推广到三维的情形。幸运的是，三维凸包算法全都超级复杂（四维以上的更恶心），所以题目不太可能要你去求。

如果你给多边形加上任何限制条件时，这个算法就玩完了（例如，多边形的顶点不多于 n 个，或者必须是矩形）。

例题

树的难题 [IOI 1991, problem 2]

已知：有一些树，你必须用铁丝围绕这些树，使得你用的铁丝最短。计算哪些树会在多边形的顶点上和铁丝的长度，还有农夫的小屋是在多边形内，还是在它之外，还是跨过多边形的边？

分析：多边形的顶点和铁丝的长度可以由问题直接得到。农夫的小屋是坐标轴上的一个矩形，你需要一点几何学知识来确定矩形中的所有点都在凸包中，或者都在凸包外,或者一些在凸包内,一些在凸包外。这样你就可以得到你想要的答案。查查几何手册，找一下这类问题的解法。

吝啬的护城河建设

已知：一些多边形房屋，计算包含这些多边形除去一个多边形的护城河的最小长度。

分析：计算已知多边形的凸包相当于计算它的所有顶点的凸包。这是一个组合问题，需要一个循环和一个凸包生成程序。对于每座房屋，删去这座房屋并计算剩下顶点的凸包。选择使得凸包最小的那座房屋。注意，只要考虑那些顶点和整个凸包重合的房屋即可，考虑整个凸包内的房屋对于问题没有任何帮助。