A divide-and-conquer algorithm for computing voronoi diagram

---计算几何实验报告

尹辉　　刘难贵　　马红标

2007-1-11

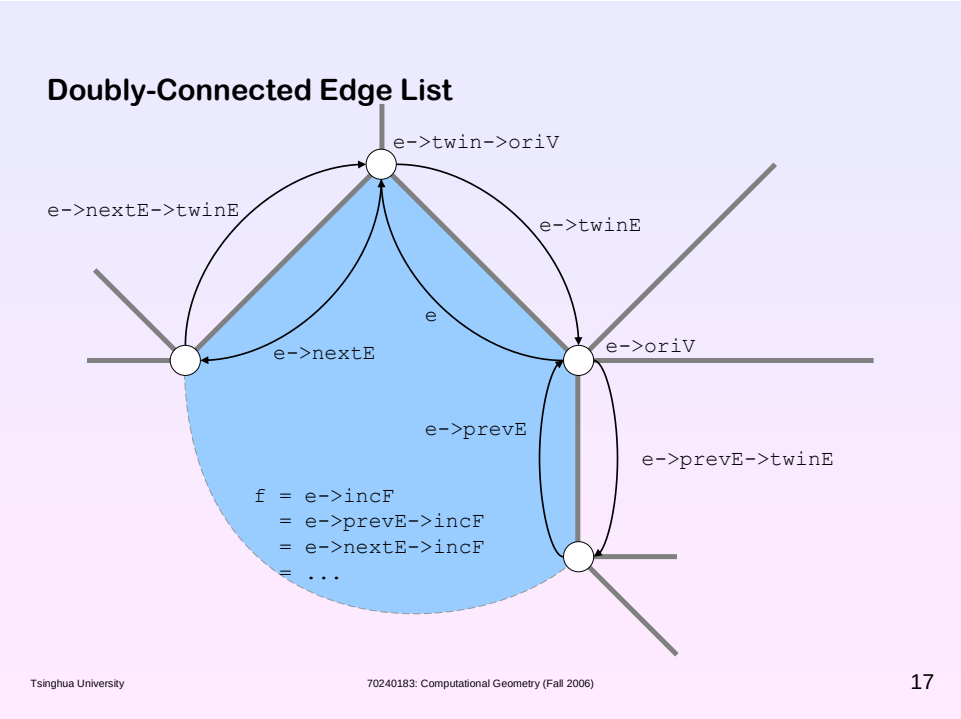
一　VD 问题的背景

计算几何范畴的 Voronoi 图是关于空间邻近关系的一种基础数据结构，自从 G. F. Voronoi (1868~1908) 提出 Voronoi 图以来，该图在实践中的广泛应用在三十年得以快速发展。其涉及和应用到的研究领域有：社会地理学。，几何形体重构、图形图像处理与模式识别、物理化学分子生物学、机器人运动规划以及加工中的路径规划等等。

VD 的基本模型为: 给定某区域内 n 个基点(site)，求出该区域的一种子区域划分(Subdivision)，使得该区域被划分成多个子区域(region)，每个子区域对应于一个基点，在此子区域内，任何位置离该基点都比距离其它基点更近。

二 相关概念及常用算法描述

在处理平面子区域划分问题时，使用 DCEL(Doubly-Connected Edge List)结构方便在点、边、面之间遍历，故 DCEL 结构在 VD 问题中得到广泛采用。 本算法也采用了 DCEL 结构，并稍加修改。下面先介绍 DCEL 的结构(如下图)。



VD 的常用算法有 3 个- 增量算法(Increnmental)、平面扫描算法（plane sweep）和分治算法(divide-and-conquer). 下面将简单介绍下前两种算法，并着重分析我们采用的分治算法。

1:增量算法

对于点集 S={p1,p2...pn}，令 Sk = {p1, …, pk}

增量算法依次计算序列：VD(S1), VD(S2), VD(S3), …, VD(Sn)

先初始化 DCEL 表示的空 VD 图，然后对每一个 p∈S，将 p 依次插入，修改 DCEL 结构，最后得到 VD(Sn)

增量算法复杂度为 O(n2logn)

2：平面扫描算法

Plane sweep 算法是由 Steve Fortune（1986）提出的一个优美

算法，Plane sweep 方法是计算几何最基本的技术之一，它把一个二维问题降为一维问题。一条垂直线(或水平线)，即 sweep line，从平面的上面移到下面(或左面移到右面)，Voronoi 图就沿着这条线构造。这个算法的时间复杂度为 O(nlogn) ，空间复杂度为 O(n)。

3: 分治算法：

DacVoronoi(S, n)　//divide-and-conquer algorithm for construct VD(S)

x-sort all sites into S = {p1, p2, ..., pn}　　//O(n**log**n)

return(dacVD(S, 1, n))

dacVD(S, i, j)　　//compute VD({pi, …, pj})

return (j-i < 3) ?

　　trivialVD(S, i, j) :

　　merge(dacVD(S, i, (i+j)/2), dacVD(S, (i+j)/2+1, j))

此算法先把点集按 x 坐标递增排序，然后把点分成 2 或 3 个点的子集。先计算 2-3 个点的 VD，再递归地把 VD（SL）和 VD(SR)merge 起来。 关键是 merge 的时候计算 contour 并演示 contour 从上到下的生成过程，这也是本实验的重点所在。

三: 主要的数据结构

1:半边:

```
struct half_edge
{
   vertex * pOriginVertex;
   half_edge * next;
   half_edge * previous;
   half_edge * twin;
   face * pLeftFace;
   bool bVisited;//标记有没有被访问过.

};
```

2:面

```
struct face
{
   half_edge * pStartHalfEdge;
   half_edge * pEndHalfEdge;
   site * pSite;
   int iType;//0:表示正常,1 表示退化.退化是指所有点共线

   face()//默认值
   {
```

```
        pEndHalfEdge   =  NULL;
        pStartHalfEdge =  NULL;
        iType          =  0;
    }
};
```

3:vertex

```
struct vertex
{
    double x;
    double y;
};
```

二:算法实现:

构建 voronoi 图

输入:site 集合

输出:voronoi 图.

0:初始工作,排序.按左右顺序排列各个 site.x 坐标小的在前面.如果 x 左边一样,那么 y 值大的在前面.

1:判断,当前 site 集合中的元素,是两个,则进入 2;是三个,则进入 3.大于三个进入 4;

2:构建两个 site 的 voronoi 图.返回

3:构建三个点的 voronoi 图.返回

4:设当前 site 的集合个数为 n,把集合分成$\left\lfloor \frac{n+1}{2} \right\rfloor$和$\left\lfloor \frac{n-1}{2} \right\rfloor$个元素的两个集合,分别进入 1;将$\left\lfloor \frac{n+1}{2} \right\rfloor$个元素的集合得到的 voronoi 图设为 LeftVoronoi,将$\left\lfloor \frac{n-1}{2} \right\rfloor$设为 RightVoronoi.进入 5.

5: 将 LeftVoronoi 和 RightVoronoi 进行 merge.

6:画图.

构建两个点的 voronoi 图.

输入:两个 site.

输出:两个 site 的 voronoi 图的 dcel 结构.

1:连接两个点.

2:两个点的中垂线.

3:用该中垂线建立两条半边.两条半边互为 twin.

4: 用 toleft 函数判断哪个 site 和哪个 halfedge 相关连.并且建立 face,设置 face 的各个域.

5:为各个 face 添加虚拟后继边

构建三个点的 voronoi 图.

输入:三个 site

输出:三个 site 的 voronoi 图的 dcel 结构.

1:从三个点连接出三条边,作出两条边的中垂线.

2:如果两条中垂线相交,计入 3;如果不相交,进入 5

3:如果该交点在三角形内部,那么以该点为起点,连接三个中点作三条射线.

如果该交点在三角形外面,那么钝角对应的射线需要改成从该交点出发,方向为从该钝角对应的边指向该交点.其他两条射线同 2.

4:用这三条射线建立六条半边.通过判断半边和各个 site 的位置关系,决定哪个 site

对应哪个 face,以及半边.进入 6

5:判断三个点的位置关系,也就是确定哪个点在中间.然后作出两条中垂线,以两条中

垂线为基础作出四条半边.以同一条中垂线为基础建立起来的半边互为 twin.

6:添加虚拟后继边

以上构建三个点和两个点的最后一步都是添加虚拟边,是由该 halfedge 的数据结构决定.对于任何一条 halfedge,它都没有保存自己的终点信息,而是必须到它的 next 中去活得该信息,该 halfedge 的终点,就是它的 next 的起点.但是对于无界的 face 来说,有一条边是没有 next 的,那么它的终点信息,就没有办法保存,所以我们添加一个虚拟后继边,该边进进有起点和 prevoius 信息,而没有其他的信息.

Merge 的过程

输入:左右两个 voronoi 图(右边的 voronoi 的图的所有 site 都在左边 voronoi 图的右边,或者正下面)

输出:归并后的 voronoi 图

1:通过左右两个 voronoi 图的两个凸包,得到两个凸包 merge 的时候新生成的两条边,这两条边就是 voronoi 图 merge 的时候重要的两条边,一个是 merge 的开始,另一个是结束,设两条边为 Line1 和 Line2. Line1 对应的两个 site 分别为 LSite1 和 RSite1,Line2 对应的两个 site 分别为 LSite2 和 RSite2.

2:做 site1 和 site2 的连线的中垂线,该中垂线从无穷远处,向其他的 site 相对于 Line1 的方向,以跟两个 voronoi 图相交.如果不相交,那么就是所有的点都共线,进入 3;否则进入 4;

3:以中垂线为基础建立两条半边,且互为 twin.判断两个半边和两个 site 的位置关系,将两条半边和两个 site,以及该 site 对应的 face 联系起来.返回.

4:如果相交,判断先跟左边的 face 相交,还是先跟有右边的 face 相交.如果是跟左边的 face 相交,设相交的半边为 Original_Halfedge,那么以射线从无穷远处到交点这

一部分建立两个半边,然后分别添加到左右 face.如果是先跟

右边相交,作类似的处理.注意左换成右.

5:设 Original_Halfedge 对应的 site 为 nextSite,用 nextSite 作 site1.如果 site1 不是 LSite2,site2 不是 RSite2,也就是还没有到 merge 的最后一步.进入 2;否则,进入 6.

6:以当前的交点作,LSite2 和 RSite2 的连线的中垂线,该中垂线为一条射线,起点为当前的交点,方向为其他 site 相对于当前 LSite2 和 RSite2 的连线的方向的异侧.以该射线为基础建立两个半边,添加到相应的 face.

7:返回.

实际上,在第一次和 voronoi 图相交以后,到最后做 LSite2 和 RSite2 的连线的中垂线之前的不走,可以不用做中垂线,而是仅仅连接当前的交点和下两个的 site 的中点做一条射线,该射线的起点是当前的交点,指向两个 site 的中点.然后和 voronoi 图相交,不过这也刚好是当前的两个 site 的中垂线,为了让算法看起来比较简单同一,就象上面一样描述了.

四： 复杂度分析及实验结果，分析

对输入的 n(n>=2)个点，有以下的分析结果：

1.分割阶段

① 对 n 个点进行快速排序，时间复杂度为 O(nlogn)。

②将 n 个排好序的点分为左右两部分，可以在常数时间内完成；将两个 Voronoi 图合并可以在 O(n)的时间内完成（见下一步分析的结果）。

2.合并阶段

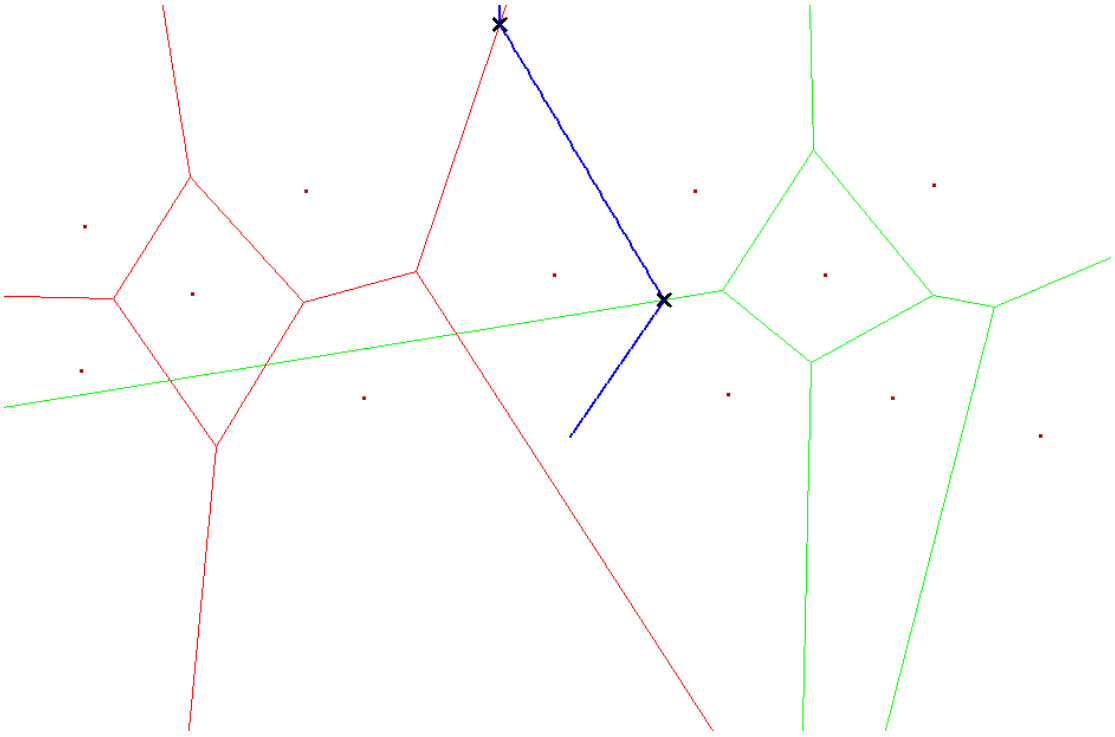
① 递归的构造凸包时间复杂度为 0(n)。

②寻找凸包支撑边的时间复杂度为 0(n)。

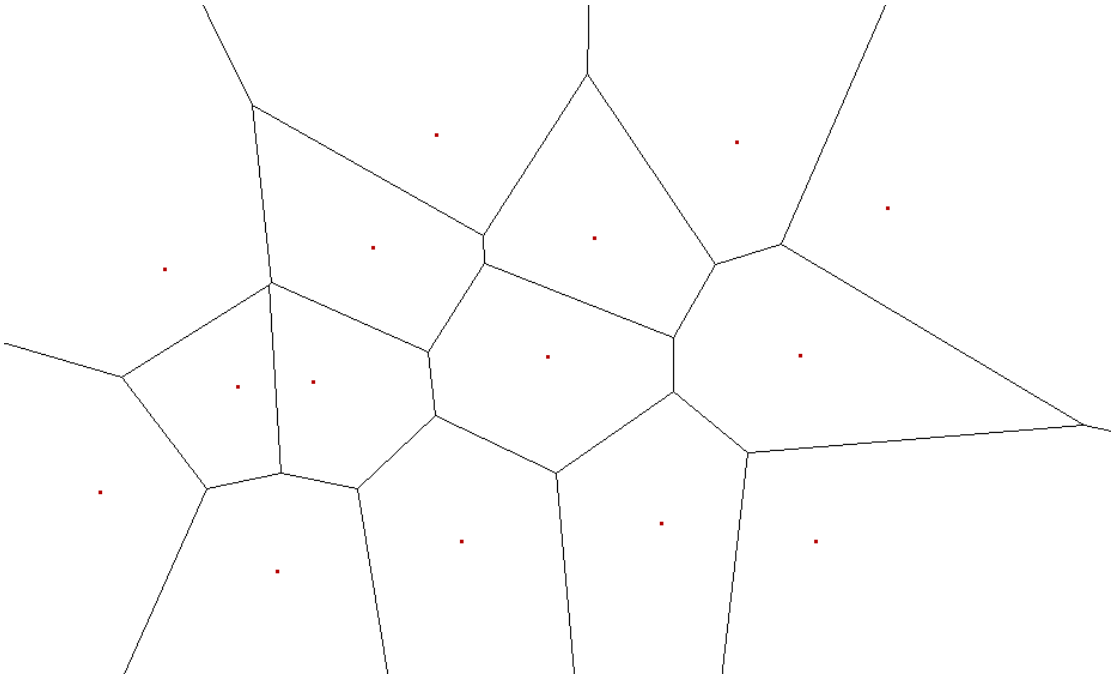
③寻找两个 Voronoi 图之间轮廓线的时间复杂度为 0(n)。因此合并阶段总的时间复杂度为 0(n)。

由递推关系 $T_n = 2T_{n/2} + O(n)$，知整个算法的时间复杂度为 O(nlogn)。

下图是程序合并阶段寻找两个 Voronoi 图轮廓线的示意图：



下图是由本程序生成的由 15 个点组成点集的 Voronoi 图：



由以上分析可知本程序成功地实现分治法求解 Voronoi 图。

五： 总结

在实现合并的时候，我们花了大量的时间，主要是这块的实现细节太多，逻辑关系复杂。虽然有过失落，沮丧，在这个过程中我们还是学到很多东西。

当然我们的程序还是有需要改进的地方：1.没有实现用户控制的单步运行；2.不能退回到上一步；3.用户界面还不够友好；4.没有与其他生成 Voronoi 图的算法进行对比，无法得到本程序的客观评价。这些都是进一步工作的方向。

参考文献

[1] M. I. Shamos and D. Hoey. Closest-point problems. In Proc. 16th Annu. IEEE Symbos. Found. Comput. Sci. ,pages 151-162, 1975.

[2] L. J. Guibas and J. Stolfi, Primitives for the manipulation of general subdivisions and the computions of Voronoi diagrams, ACM Trans. Graph. 4 (1985), 74-123.

[3] A. Okabe, B. Boots, and K. Sugihara, Spatial Tessellations, Concepts and Applications of Voronoi diagrams, John Wiley & Sons Ltd. England (1992)

[4] Computational Geometry Algorithms and Applications M.de Berg 著. 邓俊辉译