Computing Voronoi Diagram of line segments and points

——计算几何实验终期报告

来煜坤　杨旸　陈默

2003.12

目录

1. 问题背景

Voronoi 图（Voronoi Diagram，简称 VD）是计算几何领域的一个重要问题，它有着很强的现实意义。人们从实际问题中抽象出 Voronoi 分配模型（Voronoi Assignment Model）：在这一模型下，对于整个区域内的一个基点集合，每个点都会分配给距离它最近的那个基点。从 Voronoi 分配模型可以得到对整个区域的一个子区域划分，这个划分被称为这一基点集合的 Voronoi 图。

对于构造 Voronoi 图这一通用的几何结构，可以由不同的条件设置（包括维数，基点的性质，距离基于的几何空间的变化等等）产生各种问题。其中最基本的，是欧氏平面上针对点集的 Voronoi 图。在这一问题上，已有了复杂度达到最优（$O(n\log n)$时间）的算法，包括分治算法，Fortune 的平面扫描算法和由此衍生出来的海岸线（Beachline）算法。这一基本问题可以从很多方面得到推广：比如拓展到高维空间，变换几何空间（比如采用 Manhattan 几何空间），以及基点形状的变化。这些都是富有研究意义的。

我们的实验所针对的，是欧式平面上包括点和直线段的基点集合的 Voronoi 图。从二维平面上点的 Voronoi 图，我们可以扩展得到这样的定义：

不妨统称平面上的一点或者是一条直线段为一个 site，$S=\{p_1,.....,p_n\}$为平面上 n 个 site（点或直线段）所构成的集合。我们将 S 所对应的 Voronoi 图（记为$Vor(S)$）定义为对平面的一个子区域的划分——整个平面被划分成 n 个单元（Cell），它们具有这样的性质：任一点 q 位于 site p_i 对应的单元中，当且仅当对于对 $\forall p_j \in S, j \neq i$，都有 $dist(q,p_i) < dist(q,p_j)$。记点 p_i 对应的 Voronoi 单元为$V(p_i)$。

其中 $dist(q,p_k), \forall k \in [1,n]$ 定义如下：

<1>若 p_k 为点， 则 $dist(q,p_k)$ 即为两点间的几何距离；

<2>若 p_k 为直线段，则 $dist(q,p_k)$ 为点 q 到 p_k 的最短距离，即点 q 到 p_k 上各点距离中的最小值。

2. 算法及原理

我们在实验中实现了是 Fortune 在文献[1]中提出了基于 Sweepline 的点和线段的 Voronoi 图生成算法。这个算法的时间复杂度为$O(n\log n)$，空间复杂度为$O(n)$。

在描述算法之前，我们先简单说明算法针对直线段所作的处理：

1．在算法中，将直线段分为三个部分：两个端点和这条直线段本身。两个端点与基点集合中的那些点作同等处理，但是需要给基点集合中的所有点加上一条性质，这一性质表明这个点是单独的点，还是某线段的上端点，或者某线段的下端点（所谓线段的上下端点是针对扫描线移动的方向来说的）。这样原来的基点集合就被转化成不同的基点集合。

2．规定所有的直线段或者不相交，或者只在其端点处相交。实际上对于任意的相交情况只需要做简单的转化，就可以符合规定。

3．对于平面上基点集合的 Voronoi 图，某个基点所在的 Cell 实际上是这个基点与其他各个基点划分的各个半平面的交，即$V(p_i)=\bigcap_{1\le j\le n, j\neq i} h(p_i,p_j)$。两点的半平面分隔线为直线，因此平面上点的 Voronoi 图中 Cell 的边界是直线或者半直线或者直线段。而点和直线段的 Voronoi 图的情况就不是这么简单了。对于包括点和直线段的基点集合，两个基点 p,q 的半平面分隔线有如下四种情况（我们假定 p,q 不完全重合），参见图 1：

<1> p,q 都是点，则 B_{pq} 为两点的平分线，为直线；

<2> p 为直线段 q 的一端，则 B_{pq} 为过 p 且垂直于 q 的一条直线；

<3>直线段 q 不经过点 p，则 B_{pq} 为一条抛物线的一部分；

<4> p,q 为两条直线段，则 B_{pq} 可能为一点、或一条直线段、或者为空，取决于 $t_p \bigcap t_q$。

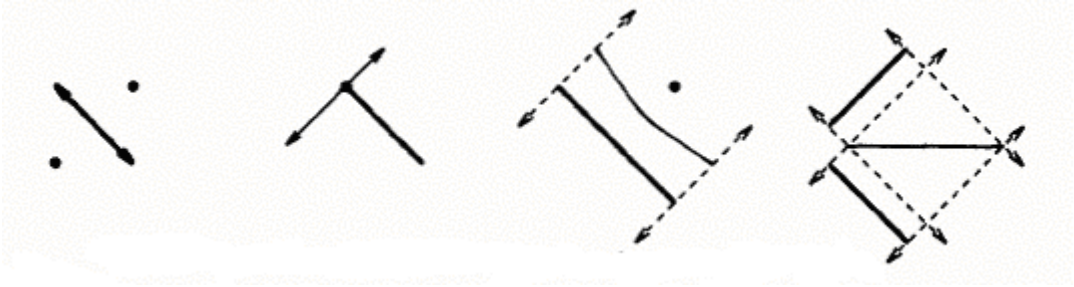


图 1 半平面分隔线的四种情况

因此点和直线段的 Voronoi 图中 Cell 的边界可以是由直线或者抛物线的整体或部分。

算法描述

基于 Sweepline 的点和线段的 Voronoi 图的算法(伪代码)：

1. 把 Q 初始化为包含所有的基本事件点(点和线段的端点)
2. $p \leftarrow \text{extract\_min}(Q)$
3. 处理第一个事件并相应更新 L，使之包含最初的值
4. while Q is not empty
   a) $p \leftarrow \text{extract\_min}(Q)$，注意：同一个平面上的点的多个事件，同时取出
   b) 分情况加以处理，可能修改 L 和 Q
5. 输出结果，算法结束

说明：

(1) 第 2.步及第 4a 步中处理时，同时考虑位置坐标的所有事件。这些事件包括：顶点，线段的上、下端点以及 bisector 变换后的像的交点。这是算法实现中所需要的。

(2) 第 3.步，初始化的情况下，也需要同时考虑坐标相同的多个事件。对于第一个事件点，它可能是这些情况的之一：顶点，一条或者多条线段的下端点。处理方法与下面讨论的类似。

(3) 4b 中包含相当多的情况，在下面给出。

分类讨论：需要考虑如下情况的组合。

a) 单独的顶点 p

找到该点 p 在 L 上所在的区域 R_q^*，

创建 bisector $B_{p,q}^*$，在 L 中把 R_q^* 替换为 $R_q^*, c_{p,q}^-, R_p^*, c_{p,q}^+, R_q^*$

把新加入的 $c_{p,q}^{+/-}$ 与周围的边界的交点添加到 Q 中

b) 两个边界的交点

主要工作：合并两个区域，产生新的 bisector，更新 L，删除无效的 Q 中的“事件”，检查新出现的相邻情况，使用 p 更新顶点和边界的端点

c) 该点是某一线段的上端点(图 2 中 Case a)

添加新顶点及与它相邻的边的 bisector(图中虚线)，并作必要的 Q、L 更新

d) p 是某一条线段的下端点(图 2 中 Case b)

需要考虑该顶点和所在的线段，以及该顶点和所在的区域的 bisector

e) d)的一种退化情形，与该顶点相邻的边是一条水平边，需要考虑 p 和相邻的线段；p 和所在区域的 site，以及相邻的线段和所在区域的 site(注意此时发生非 1-1 映射)(图 2 中 Case c)

f) 如果 p 是多条线段的下端点，且位于区域 q 内

需要添加的是任意两相邻线段的 bisector

g) 如果 p 还是某一条线段的上端点，那么需要根据角度情况考虑一些特例(图 2 中 Case d)

h) p 位于多条线段的上端点，此时，p 位于相应的边界上，情况与前一种类似，也是多条线段共点的情形(图 2 中 Case e)

i) p 位于两个 site q 和 r 的边界上，它们不全与 p 相邻。这种情况也会出现非 1-1 映射的情形) (图 2 中 Case f)

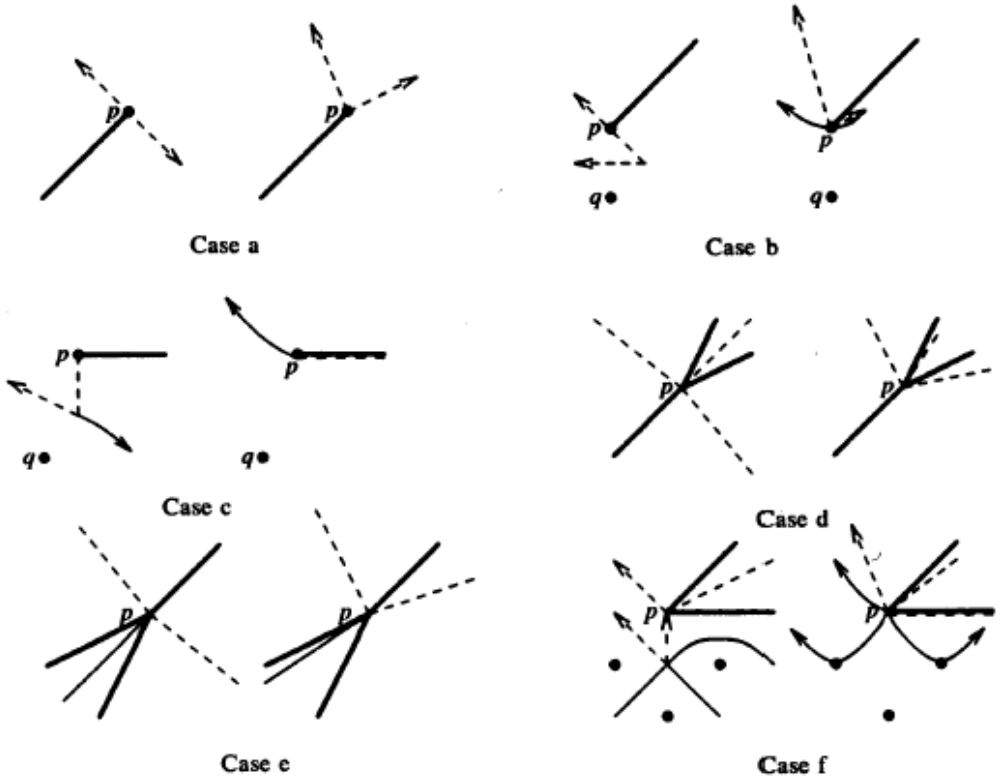


Figure 3.5: Dashed lines indicate bisectors new at p

图 2

上述只是比较典型的情况的一种比较容易讲清楚的顺序。从算法实现来讲，需要不重不漏地分类讨论。可以考虑如下 9 类情形：(a) 只是交点，(b) 只是顶点，(c) 只是(一条或多条)线段的上端点，(d) 只是(一条或多条)线段的下端点 (e) 继是(某条或某些)线段下端点也是(某条或某些)线段上端点 (f)-(i) 类似于(b)-(e)但是落在边界上(包括同时是交点的情形)。各种情形的处理类似于上面的讨论(虽然略有重复处理)，但是至少是不重不漏的。

3. 系统设计 & 数据结构

这里仅讨论计算点集与线段的 Voronoi 图的主要算法的设计与数据结构。

输入：

有限个点和线段构成的集合 S，为实现统一起见，约定线段视作三部分，即两个端点(视作与孤立的顶点相同)以及之间的(开)集。假定线段不在除端点以外的位置相交。注意：这种假定与不作这种假定得到的 Voronoi 图是有少量不同的，而正如 Fortune 的论文中所描述的，不作这种假定会使得结果非常复杂，因此，又是非常必要的。

输出：

集合 S 对应的 Voronoi 图。表示方式为一系列 bisector 的并。这里 bisector 可以是前文

所述的四种情形之一(点-点，线段-端点，线段-非端点，线段-线段)。如果两个端点不受约束，那么线段-非端点通常的结果是抛物线，而其它三种情况通常是直线。实际上，该曲线的两端或者两端之一往往是受限制的(由于与其它 bisector 相交)，在这种情况下，直线可以变成线段、射线，而抛物线可以是其中的一部分。

基本数据结构：

参考 Fortune 的论文[1]，从整体上对于变换域而言，使用的是一种扫描线算法。所以，它具有扫描线算法所具有的公共的、类似结构。主要包括：

- 事件的优先队列 Q。通常可以使用堆(Binary Heap)来完成。这个顺序就是事件点在(*)下的像的字典序(先以 y 坐标升序，相同情况下以 x 坐标为升序)。对于线段来说，两个端点都作为事件点，对于点来说，事件点是它自身，此外，两个 Voronoi 边作(*)映射后的交点也作为事件点。

- 扫描线的状态结构 L。理想情况下，它需要能够在 $O(\ln N)$ 的复杂性内完成基本的查询和更新操作。可以采用类似于平衡二叉树这样的结构(red-black tree or AVL tree 等)。扫描线 L 包含与当前扫描线(水平线)相交的区域和边界的一种交替。扫描线的相交情况以(*)映射后的像的基础上进行考虑。区域可以由任一 $p \in S$ 表示，边界则由两个区域 $p,q \in S$ 的 bisector 的形式表示。同时，通过把它们区分成 $c^{+}_{p,q}$ 和 $c^{-}_{p,q}$，可以使得每一个边界都是 y-单调的，这个性质可以在大大简化算法的实现。注意 L 上保存的信息是基于(*)映射后的，而保存和构造的 Voronoi 图的信息则是基于映射之前的。这是因为只有在(*)映射之后，才能满足扫描线所需要的序关系，而构造的过程在映射前更方便。

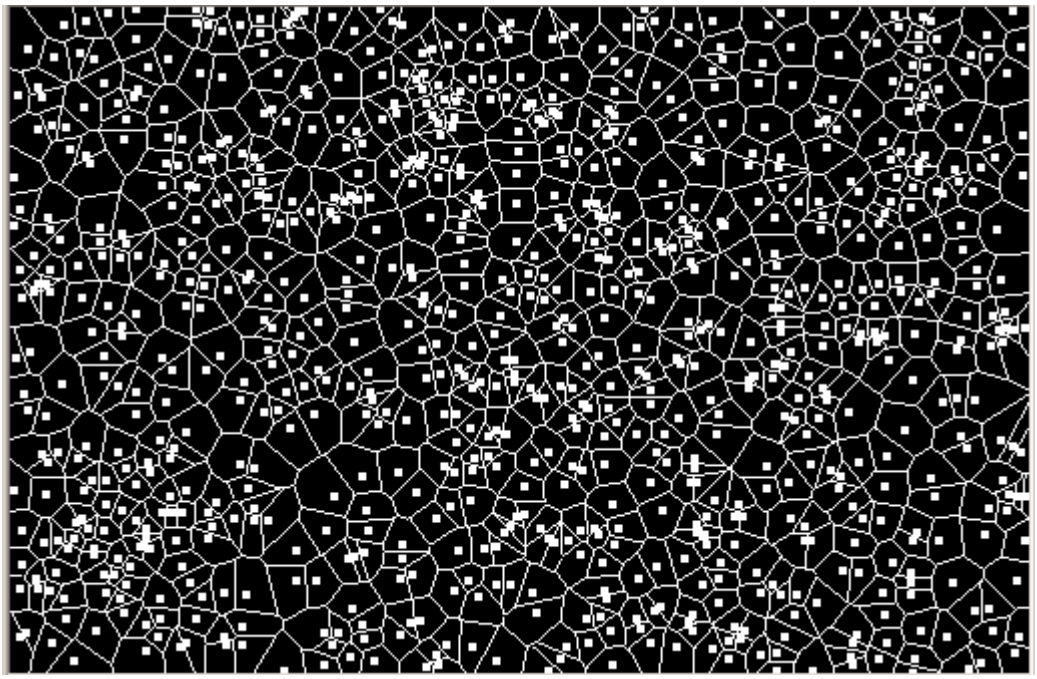
实现细节：

在实际实现时，有一些变通的方法可以使得算法更容易实现。例如：虽然从原理上讲，L 应该是区域和边界的交替形式，但是为了表达上的简洁，实际上，只需要存储边界，如果检索时落在两个边界之间，那么自然就属于相应的区域。

4. 测试、对比结果

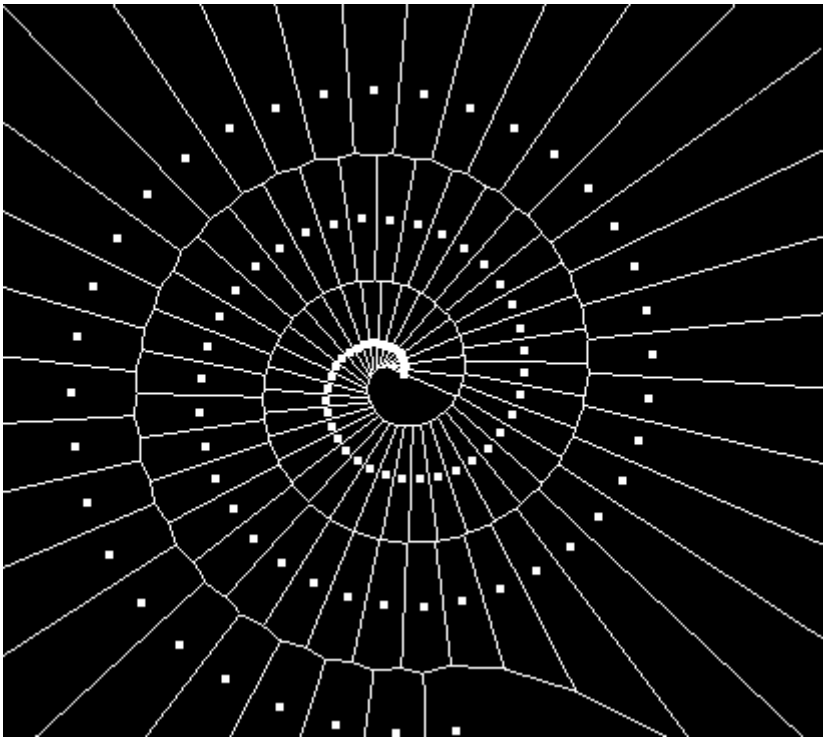
目前比较稳定的只是点集的情形，在实验提交时的补充文档中会包含线段的结果。

实验结果一：大量随机点集(1000 个)



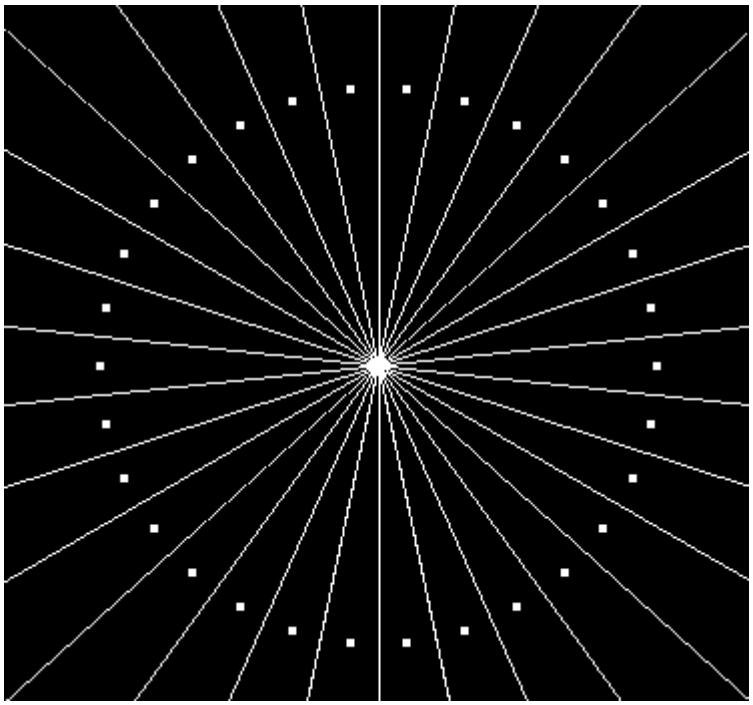
这说明算法比较稳定，而且这个结果能在较短的时间内得到。

实验结果二：阿基米德螺线



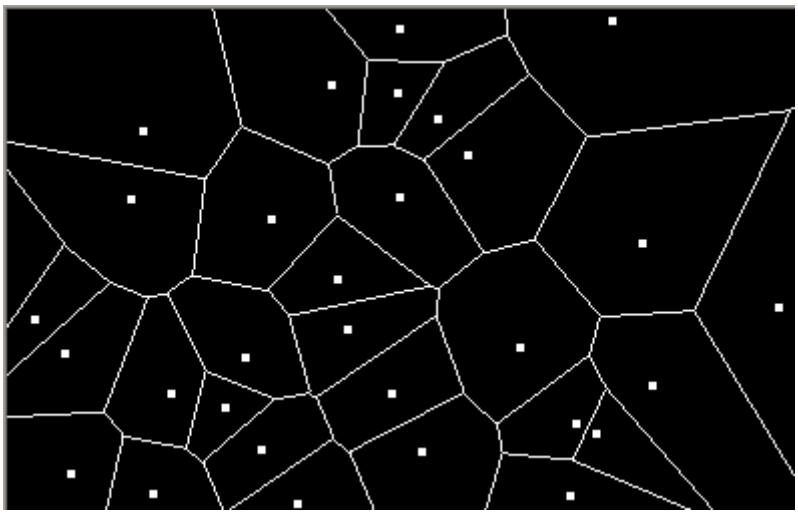
点的分布满足阿基米德螺线，得到的结果既说明了算法的稳定性，同时得到的结果也体现了 Voronoi 图的艺术性。

实验结果三：共圆的高度退化情形



这是一个特殊的高度退化的情形，在这种情况下会发生多点共圆的情形。在通常的情况下是不常见的，但是说明了算法在这种情况下的鲁棒性。

实验结果四：动态演示(snapshot)



动态演示说明：

由于屏幕快照只能显示某一帧的情形，下面简述动态演示的效果。每一个点(site)都可以看作具有单位正电荷的小球，它们以一定概率随机地从边界射入(为了防止跳变)，并具有一定的生命期。在运动过程中，任两小球之间的斥力与距离平方成反比。如果两球足够靠近，那么会看到接近于弹性碰撞的情形，如果与边界相遇，则会发生完全弹性碰撞。如果生命期已满，为了使变化平滑，这样的小球并不立即消失，而是在到达边界之后消失(即不弹回)。上述处理类似于图形学中的粒子系统。

在整个变化过程中，都能实时计算并绘制相应的 Voronoi 图，在一定程度上体现了算法

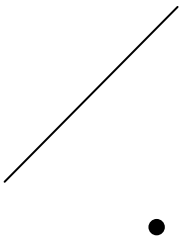
计算的性能。

5. 实现过程中遇到的问题及解决对策

- **bisector 及变换后的像的表示**

这是本实验中需要仔细处理的、而且也比较繁琐的工作之一。对于变换前的 bisector，最复杂的情形是点和线段形成的抛物线。与 Beachline 算法中的抛物线不同，这里的抛物线是一般位置的，而 Beachline 中的准线总是垂直的，换言之，后者的形式要简单得多。

另一个问题是变换后的像的形式表示。由于同时包含抛物线、双曲线和直线(及它们的部分)，因此实现起来复杂且有一定的难度。正如 Steven Fortune 的观察，总是可以把它们在最低点分成两部分，每一部分都是 y-单调的，从而大大简化处理过程。但是严格来讲，Fortune 的论文中的表述并不严格也不全面，虽然结果是可行的。其中称(p.316)取两个 site 各自的字典序最小点(对于一般位置的线段即最低点)，变换后的像以这个点为分界，x 坐标小于等于它的作为C^-，x 坐标大于等于它的作为C^+。这种表述并不严格。如顶点在线段下方，则变换后的抛物线最低点并不经过线段最低点(即使考虑整条抛物线)，分界线也并非线段最低点，而且抛物线的左单调支的 x 坐标也并不一定小于分界点。但是这种思想仍很有意义，且除了这种情况外，一般仍是正确的。更恰当的表述是在变换后曲线的最低点分成两支，两支都是 y 单调的。其实这种性质就足够了。用解析的方法处理时，最低点满足根式内为 0，而左、右支分别是根式前取负、正号。这样就很容易操作了。



- **变换与求交**

虽然求交的过程原则上是在变换域中进行，但是由于变换域后的曲线包括：线段，双曲线和抛物线，两两之间求交包含多达 9 种组合，而且双曲线、抛物线之间的求交涉及四次方程的求解，计算量大而实现复杂。在实际实现中，在变换前计算相应的交点，计算前仅包括抛物线和线段，交点相对容易计算，然后通过变换得到相应的变换后坐标，这一步也容易实现。之后，需要检查相应的分支是否一致，如果不一致，则是另一分支与之相交，而这种检查只是一个简单的求值过程。

- **退化情形**

退化情形在点集的算法中可以不予特殊处理，但是每次考虑的只是一个事件，如果同一坐标的多个事件则分别处理。在这种情形下，像四点共圆这样的情形，可以通过引入 0 长度的线段来解决。但是在点和线段的情形下，同一坐标的多个事件必须同时处理。因此，退化情形也需要特别处理。实际上上述的(f)-(i)都是某种程度上讲不常见的情形(其中多个线段共

点也可能产生上述情形，这种情况一般不视作退化)。处理时，需要对经过该点的已有边界进行限制，并引入相遇点的 site 与最边界的 site 之间的新的 bisector。如果只是变换域多点一起相交的退化情形，只要生成最边界的两个 site 创建的 bisector，并限制相遇点前的一系列 bisector。上述只是某些情况处理的一般原则，具体到实际实现，仍有多种情况需要具体分析。

6. 没有解决的问题

线段的情形的实验我们会在提交实验时的补充文档中讨论。

就这个算法基本而言，并没有过多未解决的问题。整体上而言这个算法还是很精巧的，虽然有各种情况需要考虑，但是采用这种方法可以把很多不同情况统一处理(如一般的处理对于所在 Region 对应 Site 是顶点还是线段可以一致地加以处理等)。如果有效实现，可以达到 O(nlogn)的复杂性，这也是最好的结果了。

主要的没有解决的问题实际上也是 Fortune 文中提到的一个重要缺陷是，这个过程并不是数值稳定的。虽然理论上讲这些算法确实是正确的，但是一般的实现中浮点数都是有误差的，特别是要快速地在目前的计算机上实现时更是如此。在这种情况下，由于算法会把距离远的点映射到距离很近，此外，算法还有如何确定两个点重合的问题(如顶点和交点等)，因此误差和浮点数的不精确性使得该算法(如果不采用特殊的精确计算函数库)将无法保证百分之百正确，这也是这个算法最大的缺陷。实际上，此后的很多文献特别是 Voronoi 工程方面的一些工作的目标就是通过有效的算法和谨慎的实现，得到快速、稳定的 Voronoi 算法，而这也是非常不容易的。

参考文献

[1] S. J. Fortune. “A sweepline algorithm for Voronoi diagrams”, Algorithmica, 2:153-174, 1987.