基于均匀网格的 Delaunay 三角网算法在随机聚合网屏中的应用

潘荣江 屠长河 孟祥旭 汪嘉业
(山东大学计算机科学系, 济南 250100)

摘 要 Delaunay 三角网一直是一个重要而有意义的研究课题, 并具有极其广泛的用途. 经过 20 多年来的研究, 它的生成算法已趋于成熟. 为了满足印刷、印染系统中随机聚合网屏生成的实时性需要, 将一种新的算法引入到 FM 网屏技术中, 并首先简要介绍了 Delaunay 三角网的特性及生成算法的分类; 然后主要介绍了一种基于均匀网格的 Delaunay 三角网生成算法在随机聚合网屏中的应用; 最后给出了算法的正确性证明. 经测试, 该算法的运算速度相当快, 具有接近于线性的时间复杂性, 能够满足排版印刷、印染系统中随机聚合网屏生成的需要.

关键词 Delaunay 三角网 生成算法 随机聚合网屏 均匀网格
中图法分类号: TP391.7 TS801.8 **文献标识码**: A **文章编号**: 1006-8961(2002)05-0495-06

An Algorithm for Delaunay Triangulation Using a Uniform Grid on Stochastic Clustered-Dot Screens

PAN Rong-jiang, TU Chang-he, MENG Xiang-xu, WANG Jia-ye
(*Department of Computer Science, Shandong University, Jinan* 250100)

Abstract Delaunay triangulation is widely applied in manifold fields and has a number of application-dependent approaches. This paper briefly introduces its significant properties and popular generation algorithms. Then an empirically efficient algorithm for Delaunay triangulation using a uniform grid in 2D is introduced. This method first preprocesses the data, divides the whole point distributed area into grids with around the same number of grids and points, puts all points into corresponding grids. It begins with forms an initial triangle. While looking for connecting triangles, it puts all new edges of found triangles into a queue and remove the edges which are used by two triangles or are known as boundary edges out from the queue. Repeat the process until the queue is empty, then the triangulation is finished. This paper also shows the validity of the algorithm. The algorithm is very easy for implementation and uses computer resources of time and space more reasonably. Through tests with random generated data, its running speed proves fast and exhibits linear time complexity. It can meet the requirement of stochastic clustered-dot screens in publishing, printing and dyeing systems.

Keywords Delaunay triangulation, Generation algorithm, Stochastic clustered-dot screens, Uniform grid

0 引 言

调频(FM)网屏技术(frequency modulated screening)是在 80 年代中期开始出现的一种较新的网屏技术, 由于它能产生比传统的调幅(AM)网屏(amplitude modulated screening)更细致的颗粒, 因此 20 世纪 90 年代初期开始流行, 并迅速普及起来. 为了避免 AM 网屏中产生固定间距的网点图案, FM 网屏采用某种数学算法, 将固定尺寸的网点或在某些随机网点处理过程中将不同尺寸的网点放置在随机的位置, 也就是使用的网点数目是变化的, 因此又称

基金项目: 国家自然科学基金(69873028)(69973028); 高等学校优秀青年教师教学科研奖励计划资助.
收稿日期: 2000-11-20; **改回日期**: 2001-06-25

为随机网屏. 这样含有较高网点密度的区域表现出来的色彩较浓, 而网点密度低的区域表现的色彩较淡, 从而可以有效地避免一些干扰图案, 如云纹等的生成[1]. FM 网屏使用的网点比 AM 网屏所使用的网点要小得多, 当图象为高位彩色, 且具有平滑的色调过渡、动态范围较宽、图象细节较为复杂时, FM 网屏具有强大的优势, 即使在低分辨率设备上, 也能输出高质量的图象, 但是调频网屏需要用复杂的算法来决定网点的位置(这在调频网屏中是最重要的) , 这既要有随机性, 又不能造成视觉上的颗粒化[2]. 虽然提出调频网屏已经有许多年了, 但是由于占用内存多、计算量大、算法复杂等原因, 致使长期未能得到应用, 直到最近, 随着计算机硬件和软件技术的发展才实用化, 实验证明, 这种技术有着很大的市场潜力[3].

目前国外有多家打印机厂家都采用了 FM 网屏技术, 但一般都没有公开其算法. 笔者在研究的过程中采用了一种新的算法[4], 其具体步骤如下:

(1) 在大抖动矩阵内随机放置圆盘. 圆盘的中心就是网点的中心, 其半径 r 决定了网点中心间的距离. 网点中心之间的最小距离是 r, 最大距离是 $2r$.

(2) 为了获得每一个网点的最大覆盖区域, 必须对上面产生的网点中心实施 Delaunay 三角化.

(3) 围绕每一个网点中心, 根据要求的灰度级产生等灰度区域, 按照灰度值从小到大的顺序, 以适当的比例填充三角形内的区域, 即生成随机聚合网屏.

1 Delaunay 三角网性质及生成算法

分布在平面上点集的三角网, 几十年来一直是重要而有意义的研究课题, 现已有很多教材[5]和论文讨论了其特点和构造算法.

Delaunay 三角网具有空外接圆和最大最小角两个重要性质. 这两个性质决定了 Delaunay 三角网具有极大的应用价值. 根据实现过程, 可以把生成 Delaunay 三角网的各种算法分为分而治之算法、逐点插入法、三角网生长法 3 类.

笔者在实际研究过程中发现, 有的算法虽然理论上具有较好的时间复杂性, 有的算法的时间复杂性甚至是 $O(N\log N)$[6,7], 但由于数据结构复杂、稳定性差, 因而实际执行速度较慢, 不能满足排版印刷中实时性的要求. 根据网点中心分布相对比较均匀的特点($r<$ 网点之间的距离$< 2r$), 本文采用了一种基于均匀网格的 Delaunay 三角网的算法[8]. 这个算法有如下几个新特点:

(1) 可直接处理数据点, 没有通常分而治之算法的开销(即堆栈管理、划分数据集、合并部分结果等) , 而且不要求数据有序.

(2) 能使用均匀网格结构来形成三角形、寻找边界边和最近点.

(3) 能通过分层循环产生三角形. 这就保证了三角网是完备的、正确的, 而且有助于算法在计算三角形时, 能直接输出数据的结构.

(4) 能使用链表控制三角化, 以便动态降低内部网格数据结构的复杂性.

(5) 具有稳定性, 能自动处理重合点和共线点.

(6) 不用任何额外的操作就可产生数据集的凸包.

(7) 对测试所用的随机产生的数据, 具有线性时间复杂性.

算法的步骤如下:

(1) 预处理数据

首先把数据点分布的平面区域划分成网格, 使网格单元数和点数大致相同, 然后根据点的坐标值, 把所有的点放进相应的网格单元中.

(2) 形成一个三角形

在靠近数据平面区域中间的网格单元中找到一个初始点, 然后找一个距离初始点最近的点, 形成一条有向边, 其方向为从初始点到最近点. 形成三角形时, 首先扫描这条有向边的右侧区域, 并寻找与这条有向边的两个端点角度最大的点; 然后扫描通过这条有向边的两个端点与该点所形成圆的区域, 如果圆内存在一个点, 则用那个点重新形成一个圆, 再扫描, 直到圆内没有点为止.

(3) 连接三角形

根据上面的算法找出所有的三角形. 在找三角形的过程中, 把找到的三角形的新数据点填加到一个链表中, 删除已经被三角形封闭起来的数据点. 重复这个步骤, 直到链表中的数据点全为边界点为止, 才结束三角化过程.

2 算法实现

2.1 数据结构

首先建立数据点的一个数组, 其中每一个数据点使用下面的数据结构:

```
struct cellnode
{
```

```
    int x, y; //点的标准化坐标
    bool used; //点是否已经在一个三角形中的标志
    bool bBoundary; //标记是否是边界点
}
```

使用一个二维指针数组作为网格的基本数据结构, 且每一个指针指向落在同一个网格单元中数据点的链表, 该链表中存储每个数据点在数据点数组中的下标.

采用与图的邻接表相类似的数据结构来保存生成的三角形结构, 这样做的优点是以后对三角形进行进一步处理比较简单. 邻接表中存储每个数据点在数据点数组中的下标.

同时在三角化的过程中, 动态维护一个临时邻接表.

2.2　数据预处理

首先把数据点分布的平面区域划分成若干均匀网格, 并使网格单元数和点数大致相同; 然后根据点的坐标值, 把所有的点放进相应的网格单元中.

设二维点集为$\{(x_i, y_i), i = 0, \cdots, n\}$. 为了计算均匀网格, 首先求数据集的包围盒.

利用下式计算网格的大小 s

$$s = \sqrt{\frac{(x_{\max} - x_{\min})(y_{\max} - y_{\min})}{n}}$$

二维平面 x、y 方向上网格单元数 R_x、R_y 为:

$$R_x = \text{int}\left|\frac{x_{\max} - x_{\min}}{s}\right| + 1$$

$$R_y = \text{int}\left|\frac{y_{\max} - y_{\min}}{s}\right| + 1$$

为了把每一个数据点放入一个, 且只一个网格单元中, 需对每一个点(x_i, y_i), $i = 0, \cdots, n$, 执行下列操作:

(1) 计算数据点所在网格单元号

$$c_i = \text{int}\left|\frac{x_{\max} - x_{\min}}{s}\right|, c_j = \text{int}\left|\frac{y_{\max} - y_{\min}}{s}\right|$$

(2) 若单元(c_i, c_j)是空的, 则把点(x_i, y_i)放入.

(3) 若单元(c_i, c_j)中已经有点, 则检查是否是重合点. 如果当前点与单元中已有的点重合, 则忽略该点; 否则, 把该点放入单元中.

经过上述处理, 每一个数据点属于一个, 且只一个网格单元, 并且自动消除了重合点.

2.3　找初始点和初始边

在生成三角网的算法中, 一个重要的问题就是三角网从哪一点开始生成, 也就是算法应该首先选择哪一个点和哪一条边. 从本质上讲, 算法可以从任何点开始, 但为了提高效率, 在本算法中最好选择位于数据集中心附近的点作为初始点(这一点是由于下面连接三角形过程造成的).

设中间单元为$(m, n) = (R_x/2, R_y/2)$, 如果单元(m, n)非空, 则选择链表的第 1 个结点; 否则搜索相邻的单元.

在得到初始点 P_1 后, 使用下面的方法寻找最近点, 并计算第 1 条边:

(1) 给 $d_{\min}$ 赋上一个比较大的初始值, 例如, 包围盒的对角线的长度;

(2) 如果包含初始点的单元中含有一个以上的点, 则寻找距离 P_1 最近的点, 并计算这两点之间的距离 d, 如果 $d < d_{\min}$, 则 $d_{\min} = d$;

(3) 按上面的行列顺序搜索相邻的单元, 对每一行、每一列执行下列操作:

从 P_1 到靠近 P_1 的网格边画一条垂线; 如果垂线的长度小于 $d_{\min}$, 则搜索这一行或列; 否则, 把这个方向(上、下、左、右)标记为无效方向; 如果发现了点, 则计算其到 P_1 的距离 d, 如果 $d < d_{\min}$, 则 $d_{\min} = d$; 当所有方向都被标记为无效方向时, 就停止搜索, 即在未搜索行列中所有点与 P_1 的距离都大于 $d_{\min}$.

通过执行上面的过程, 就可以找到初始点 P_1 和初始边$\langle P_1, P_2 \rangle$.

2.4　构造三角形

为了寻找初始边外第 3 点 P_3, 需使$\triangle P_1P_3P_2$满足 Delaunay 标准, 并执行下列操作:

(1) 如图 1 所示, 寻找单元(i_1, j_1)和(i_2, j_2), 这两个单元或是端点 P_1、P_2 所在的单元, 或是直接相邻的单元. 寻找单元时, 可以通过 P_1P_2 与 P_1 所在单元的底线相交来计算 i_1, 交点所在单元的列下标为 j_1. 类似地, 可用 P_2 所在单元的右边网格线来计算 j_2, 交点所在单元的行下标为 i_2.

(2) 形成 P_1P_2 的右侧由单元覆盖的三角形区域. 单元三角形的顶点是(i_1, j_1)、(i_1, j_2)、(i_2, j_2), 边界是水平方向、垂直方向和对角线方向排列的单元. 在对角线方向选择与直线 P_1P_2 相交的单元.

(3) 检查区域内的每一个单元, 可以有各种搜

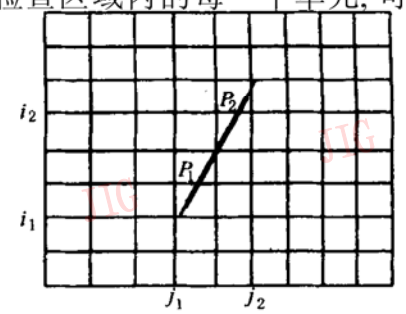


图 1　寻找第 3 点

索方法. 例如, 可以从(i_1, j_2)开始搜索每一列, 并往左移动, 直到(i_1, j_1); 或者先搜索(i_1, j_2), 然后沿对角线方向搜索. 这样, 首先搜索的单元最有可能包含所要求的点, 且能使生成的三角形在第 3 点的角度最大, 以尽可能接近于等角三角形.

(4) 如果发现了多个点, 则选择给出最大角的点(也就是最小余弦), 再通过这 3 个点作圆. 如果没有发现点, 则从$(i_1- k, j_1- k)$到$(i_1- k, j_2+ k)$来搜索行、从$(i_1- k, j_2+ k)$到$(i_2+ k, j_2+ k)$来搜索列(其中, $k= 1, 2, \cdots$), 直到发现点为止.

(5) 在获得第 1 个圆后, 继续在圆所覆盖的网格单元中搜索, 并做同样的处理, 即选择具有最小余弦的点进行画圆.

(6) 当对圆所覆盖的网格单元都搜索完以后, 若没有发现其他数据点, 则停止搜索.

动态修改圆是这个算法的关键部分, 因为搜索局限于靠近边P_1P_2的部分进行, 一般只用几步就可找到点.

如果边P_1P_2是水平的或垂直的, 则可以使用简单的包围盒代替三角形.

2.5 连接三角形

其连接步骤如下:

(1) 设初始点是P_1, 初始边是P_1P_2, 这时三角形的数据结构是:

$F: P_1^{\rightarrow} P_2$

$L: P_2^{\rightarrow} P_1$

并把P_1、P_2的 used 标志置为真.

(2) 从初始边P_1P_2开始, 找到第 3 点P_3, 这时做如下处理:

$F: P_1^{\rightarrow} P_2^{\rightarrow} P_3$(把$P_3$追加到$F$链表的表尾)

$P_2^{\rightarrow} P_3^{\rightarrow} P_1$(把$P_3$插入到原$L$链表的表头之后)

$L: P_3^{\rightarrow} P_1^{\rightarrow} P_2$(建立一个新的顶点链表, $P_3^{\rightarrow} F(1)^{\rightarrow} L(1)$)

并把P_3的 used 标志置为真, 以$L(1)F(1)$(即P_3P_1)为初始边, 继续寻找第 3 点.

(3) 继续按上述方法进行处理, 当到达图 2(a)所示的情况 1 时, 三角形的数据结构是:

$F: P_1^{\rightarrow} P_2^{\rightarrow} P_3^{\rightarrow} P_4^{\rightarrow} P_5^{\rightarrow} P_6$

$P_2^{\rightarrow} P_3^{\rightarrow} P_1$

$P_3^{\rightarrow} P_4^{\rightarrow} P_1^{\rightarrow} P_2$

$P_4^{\rightarrow} P_5^{\rightarrow} P_1^{\rightarrow} P_3$

$P_5^{\rightarrow} P_6^{\rightarrow} P_1^{\rightarrow} P_4$

$L: P_6^{\rightarrow} P_1^{\rightarrow} P_5$

这时以$F(1)L(1)$(即边P_1P_6)为初始边, 寻找第 3 点, 结果找到P_2, 若发现P_2已经使用过了, 并且$F(2)= P_2$, 则说明P_1已经使用完毕, 可以保存到最终的三角形结构中, 并从相应的网格单元链表中删除P_1, 以便减少搜索的次数. 相应的三角形结构变为:

$F: P_2^{\rightarrow} P_3^{\rightarrow} P_1^{\rightarrow} P_6$

$P_3^{\rightarrow} P_4^{\rightarrow} P_1^{\rightarrow} P_2$

$P_4^{\rightarrow} P_5^{\rightarrow} P_1^{\rightarrow} P_3$

$P_5^{\rightarrow} P_6^{\rightarrow} P_1^{\rightarrow} P_4$

$L: P_6^{\rightarrow} P_2^{\rightarrow} P_1^{\rightarrow} P_5$

(4) 继续进行处理, 当到达图 2(b)所示的情况 2 时, 三角形的数据结构是:

$F: P_3^{\rightarrow} P_4^{\rightarrow} P_1^{\rightarrow} P_2^{\rightarrow} P_8$

$P_4^{\rightarrow} P_5^{\rightarrow} P_1^{\rightarrow} P_3$

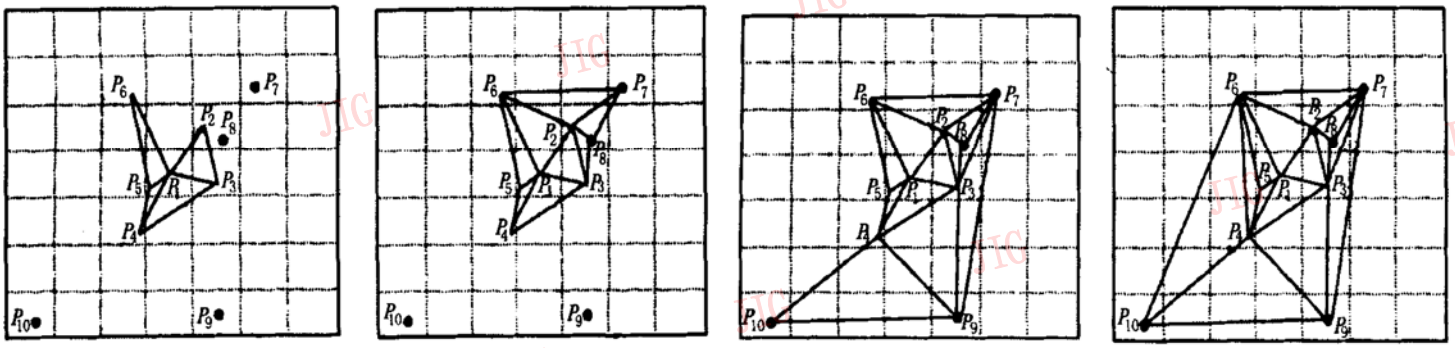
$P_5^{\rightarrow} P_6^{\rightarrow} P_1^{\rightarrow} P_4$

$P_6^{\rightarrow} P_7^{\rightarrow} P_2^{\rightarrow} P_1^{\rightarrow} P_5$

$P_7^{\rightarrow} P_8^{\rightarrow} P_2^{\rightarrow} P_6$

$L: P_8^{\rightarrow} P_3^{\rightarrow} P_2^{\rightarrow} P_7$

这时以$F(1)L(1)$(即边P_3P_8)为初始边, 寻找第 3 点, 结果找到P_7, 若发现P_7已经使用过了, 并且L链表的最后一个结点等于P_7, 则说明P_8已经使用完毕, 可以保存到最终的三角形结构中, 并从相应的网格单元链表中删除P_8, 以便减少搜索的次数. 相应的三角形结构变为:



(a) 情况 1 (b) 情况 2 (c) 情况 3 (d) 情况 4

图 2 三角形连接各种情况

$F: \overrightarrow{P_3}\ \overrightarrow{P_4}\ \overrightarrow{P_1}\ \overrightarrow{P_2}\ \overrightarrow{P_8}\ P_7$

$\overrightarrow{P_4}\ \overrightarrow{P_5}\ \overrightarrow{P_1}\ P_3$

$\overrightarrow{P_5}\ \overrightarrow{P_6}\ \overrightarrow{P_1}\ P_4$

$\overrightarrow{P_6}\ \overrightarrow{P_7}\ \overrightarrow{P_2}\ \overrightarrow{P_1}\ P_5$

$L: \overrightarrow{P_7}\ \overrightarrow{P_3}\ \overrightarrow{P_8}\ \overrightarrow{P_2}\ P_6$

(5) 继续进行处理, 当到达图 2(c) 所示的情况 3 时, 三角形的数据结构是:

$F: \overrightarrow{P_4}\ \overrightarrow{P_5}\ \overrightarrow{P_1}\ \overrightarrow{P_3}\ \overrightarrow{P_9}\ P_{10}$

$\overrightarrow{P_5}\ \overrightarrow{P_6}\ \overrightarrow{P_1}\ P_4$

$\overrightarrow{P_6}\ \overrightarrow{P_7}\ \overrightarrow{P_2}\ \overrightarrow{P_1}\ P_5$

$\overrightarrow{P_7}\ \overrightarrow{P_9}\ \overrightarrow{P_3}\ \overrightarrow{P_8}\ \overrightarrow{P_2}\ P_6$

$\overrightarrow{P_9}\ \overrightarrow{P_{10}}\ \overrightarrow{P_4}\ \overrightarrow{P_3}\ P_7$

$L: \overrightarrow{P_{10}}\ \overrightarrow{P_4}\ P_9$

这时以 $F(1)L(1)$ (即边 P_4P_{10}) 为初始边, 寻找第 3 点, 结果找到 P_6, 若发现 P_6 已经使用过了, 但不属于情况 1 或情况 2, 为了避免出现“洞”, 则可以把 F 链表移动到 L 链表之后. 相应的三角形结构变为:

$F: \overrightarrow{P_5}\ \overrightarrow{P_6}\ \overrightarrow{P_1}\ P_4$

$\overrightarrow{P_6}\ \overrightarrow{P_7}\ \overrightarrow{P_2}\ \overrightarrow{P_1}\ P_5$

$\overrightarrow{P_7}\ \overrightarrow{P_9}\ \overrightarrow{P_3}\ \overrightarrow{P_8}\ \overrightarrow{P_2}\ P_6$

$\overrightarrow{P_9}\ \overrightarrow{P_{10}}\ \overrightarrow{P_4}\ \overrightarrow{P_3}\ P_7$

$\overrightarrow{P_{10}}\ \overrightarrow{P_4}\ P_9$

$L: \overrightarrow{P_4}\ \overrightarrow{P_5}\ \overrightarrow{P_1}\ \overrightarrow{P_3}\ \overrightarrow{P_9}\ P_{10}$

以 $F(1)L(1)$ (即边 P_5P_4) 为初始边, 寻找第 3 点, 结果找到 P_6, 并形成三角形 $P_4P_5P_6$, 若这时出现情况 1, 则做修改, 再以 $F(1)L(1)$ (即边 P_6P_4) 为初始边, 寻找第 3 点, 结果找到 P_{10}, 形成三角形 $P_6P_4P_{10}$, 这样就可以避免出现“洞”.

(6) 如果遇到边界边的情况, 则标记该边的两个顶点为边界点, 并且可以把 F 链表移动到 L 链表之后.

(7) 重复上述步骤, 处理各种情况, 直到在三角形结构中, 边界点的个数等于链表的个数为止. 如图 2(d) 所示, 这时三角形结构中全部是边界点, 从中还可以顺便获得点集的凸包.

$F: \overrightarrow{P_6}\ \overrightarrow{P_7}\ \overrightarrow{P_2}\ \overrightarrow{P_1}\ \overrightarrow{P_5}\ \overrightarrow{P_4}\ P_{10}$

$\overrightarrow{P_7}\ \overrightarrow{P_9}\ \overrightarrow{P_3}\ \overrightarrow{P_8}\ \overrightarrow{P_2}\ P_6$

$\overrightarrow{P_9}\ \overrightarrow{P_{10}}\ \overrightarrow{P_4}\ \overrightarrow{P_3}\ P_7$

$L: \overrightarrow{P_{10}}\ \overrightarrow{P_6}\ \overrightarrow{P_4}\ P_9$

3　算法正确性证明

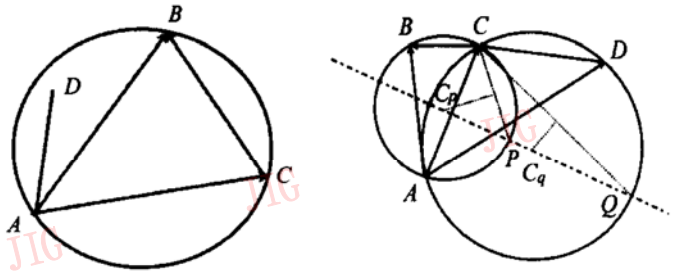
这个算法是否能产生有效的 Delaunay 三角形还需要进行如下证明: 对于每一个计算得到的三角形, 其外接圆内不含任何其他数据点. 由于算法只是搜索每条边的右侧, 因此可以保证该边的右侧, 在外接圆内没有其他数据点(既然算法使用具有最小余弦的点, 那么在圆内该边的右侧必定没有数据点) . 虽然已证明在外接圆内, 该边右侧没有其他数据点, 但是还需要证明在外接圆内, 该边的左侧也没有其他数据点.

图 3(a) 中, A 为起始点, 由于 B 是距离 A 最近的点, 因此对于任何与边 AB 构成三角形的其他点 C, 因 $|AB| \le |AC|$, 故 $\angle ACB \le \angle ABC$, $\angle ACB \le 90°$, 且不含 C 的弧 AB 小于半圆弧. 如果在三角形 ABC 的外接圆内有一点 D, 且位于 AB 的左侧, 则 $|AD| \le |AB|$, 由于这与 B 是距离 A 最近的点相矛盾. 因此在 ABC 的外接圆内不含任何其他数据点.

找到第 1 个 Delaunay 三角形 ABC 以后, 以 AC 为起始边继续在 AC 的右侧寻找第 3 点 D, 根据算法中使用的搜索原则, 可以保证在三角形 ACD 的外接圆内 AC 的右侧没有数据点, 但是需要证明在 ACD 的外接圆内 AC 的左侧也没有数据点.

由图 3(b) 可见, 因为 D 在圆 ABC 的外面, 所以圆 ACD 与 AC 的垂直平分线的交点 Q 一定位于圆 ABC 的外面, 且与点 P 均在 AC 的同一侧, 否则点 D 或者在圆 ABC 内或者在 AC 的左侧, 都会出现矛盾, 因此弧 AQC 的补弧全部包含在圆 ABC 内, 且在 AC 的左侧. 由于圆 ABC 内不含任何数据点, 因此在 ACD 的外接圆内, AC 的左侧也没有数据点.

依次类推, 可以确定该算法生成的三角形是 Delaunay 三角形.



(a) 初始三角形　　(b) 构造第 2 个三角形

图 3　Delanay 三角形的生成

4　结　论

这个算法非常简单, 不仅容易实现, 且速度较快, 能满足排版印刷中实时性的要求. 表 1 是在 PII 266、64M 内存的机器上采用不同算法对不同点

数进行三角化的运行时间, 由于随机放置数据点, 每次运行的时间略有不同, 因此需要对 3 次运行的结果进行平均, 加以比较. 图 4 是使用 IDL5.4 作出的不同点数 Delaunay 三角化运行时间的线性拟合曲线; 图 5 是对网点中心 Delaunay 三角剖分的结果. 而采用文献[7]所给出的算法, 当超过3 000个点时, 就需要几分钟, 根本不能满足实际需要.

表 1 不同算法进行不同点数三角化的运行时间

点数(个)	采用本算法运行时间(s)	采用文献[7]给出的算法运行时间(s)
530	0.110 00	1.633
1 032	0.176 67	5.117
1 614	0.346 67	12.253
1 993	0.421 33	21.802
2 520	0.573 33	37.715
2 903	0.627 67	51.244
3 433	0.749 67	75.639
3 958	0.910 00	110.008
4 667	1.113 34	155.944
4 984	1.316 67	211.623
5 761	1.423 33	251.431
7 315	1.903 33	384.103
9 513	2.522 00	712.705
10 385	3.089 67	
11 906	3.200 00	
12 373	3.493 67	
13 715	3.643 67	
17 046	5.216 00	
18 978	5.916 00	
22 468	7.413 37	

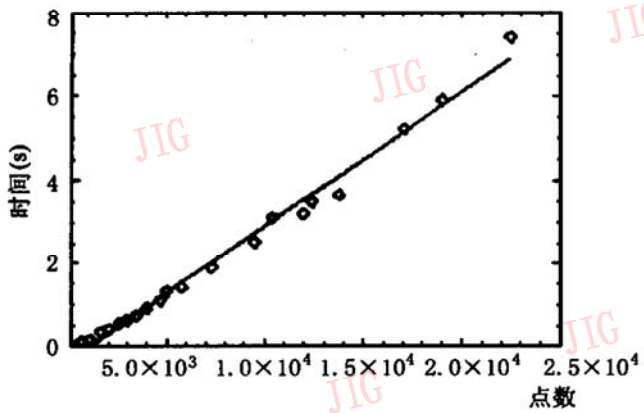


图 4 不同点数 Delaunay 三角化运行时间的线性拟合曲线

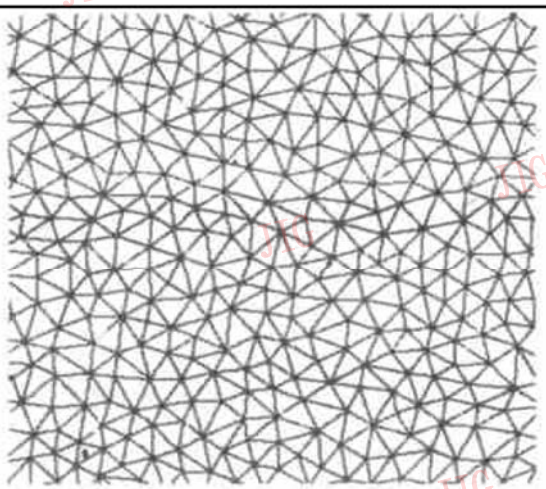


图 5 Delaunay 三角剖分的结果

参 考 文 献

1 PrePRESS Technology Report. Stochastic Screening [R], http://www.prepress.pps.com/mem/lib[EB/OL], 1999/10/20

2 Sybil E I. Preparing digital images for print(印前数字图象处理实用技术)[M]. USA, New York: McGraw-Hill, Inc. 1996. 李建军译. 北京: 电子工业出版社, 1998: 27~38.

3 Herzig Somerville. Stochastic screening, http://www.herzig.com/stochastic.html[EB/OL], 2000/01/15.

4 屠长河, 潘荣江, 孟祥旭. 一种随机聚合网屏的生成算法[J]. 计算机学报, 2000, 23(9): 938~942.

5 Joseph O'Rourke. Computational geometry in C[M]. U K, Cambridge University Press, 1994: 113~203.

6 Jurg Kraemer. Delaunay triangulation in two and three dimensions[D]. Deutsch, http://www.geom.umn.edu[EB/OL], 1999/03/23.

7 武晓波, 王世新, 肖春生. Delaunay 三角网的生成算法研究[J]. 测绘学报, 1999, 28(1): 28~35.

8 Tsung-Pao Fang, Les A Piegl. Delaunay triangulation using a uniform grid[J]. IEEE Computer Graphics and Applications 1993, 13(5): 36~47.

潘荣江 1968 年生, 山东大学计算机系 98 级硕士研究生, 助理研究员. 主要研究领域为计算机图形学、图象处理技术, 计算机软件.

屠长河 1968 年生, 山东大学计算机科学系博士研究生, 副教授. 主要研究领域为计算机图形学、图象处理技术、人机交互、参数化设计.

孟祥旭 1962 年生, 山东大学计算机科学系, 教授, 1998 年获中国科学院计算技术研究所博士学位. 主要研究领域为计算机图形学、人机交互与虚拟现实, 科学计算可视化、参数化设计.

汪嘉业 1937 年生, 山东大学计算机科学系教授. 主要研究领域为计算机图形学、计算几何等.