

文章编号: 1005—0523(2005)04—0106—04

双向分块快速 Delaunay 三角剖分算法

占自才

(华东交通大学 电气与电子工程学院, 江西 南昌 330013)

摘要:介绍一种双向分块快速 Delaunay 平面剖分算法, 该算法有别于其他的分治算法, 其特点是运算速度快, 时间度为 $O(N \log_2 N)$, 算法易于理解和实现. 该算法在二维平面中首先把被三角剖分的点集均匀分为多个只有 3 点(最多有一个块不是 3 个点)的点块. 首先对每一个点块进行 Delaunay 三角剖分, 再对相邻的点块中三角剖分进行合并. 并介绍了该算法的数据结构, 充分说明了该算法的可操作性.

关 键 词:双向分块; 快速; 合并; 三角剖分; 数据结构

中图分类号: O189.11

文献标识码: A

1 引 言

三角剖分是科学计算与分析中的一种重要数学工具, 在有限元分析、几何重构^[1]和数字地形构模、数字高程插值方面都得到了广泛的应用. 对这类方法的研究, 不论是在 2D 还是 3D 区域上都已有很多研究成果^{[1][2]}, 尤其是对 2D Delaunay 三角剖分的研究. 但对于散点数目比较大时, Delaunay 三角剖分计算量大, 算法执行速度较慢^[3]. 有些采用多台 Cpu 并行处理提高处理速度, 这中方法更适合于海量散点 Delaunay 三角剖分, 且算法复杂, 实现难度大.

在三角剖分算法中, 假设 N 是散点的数目, 则现有 Delaunay 三角剖分算法的计算时间为 $O(N \log_2 N)$ 或 $O(N^2)$ ^[4]. 而已有的一些算法虽然大部分基于分治思想, 有的不但计算量大, 而且数据结构复杂, 很难实现. 为此, 本文提出了一种数据结构较易实现, 运算时间度为 $O(N \log_2 N)$ 双向分块算法. 对于散点数目为几 $M \sim$ 几百 M Delaunay 三角

剖分, 其运算速度优势较为明显.

2 有关定义及符号

为了后面算法及数据结构的描述方便, 本文首先给出了相关定义及符号说明.

定义 1 给定顶点集合 $V = \{V_1, V_2, \dots, V_N\}$, 有 $DT(V)$, 如果 $DT(V)$ 满足下述条件, 则称 $DT(V)$ 是 Delaunay 三角剖分: 设边 $e_{i,j} = \{V_i, V_j\}$, 对于任意的 $\triangle V_i V_j V_k$, 如果 $e_{i,j}, e_{i,k}, e_{k,j} \in DT(V)$, 则 $\triangle V_i V_j V_k$ 的外接圆内不包含 V 中的任何其它顶点.

定义 2 设 V_i 为 V 中顶点, $V_{i,j} (0 < j \leq K)$ 表示与 V_i 相连的顶点(其中 K 与 V_i 为相连的顶点数目), $(V_i, V_{i,j}) \in DT(V)$, 对于 V_i 和所有 $V_{i,j}$, 它们必构成如图 1 所示的图. 我们定义如下两个函数:

从 $V_{i,j}$ 开始, 以逆时针方向遍历星状环, 遇到的第一个环上点 $(V_{i,j-1})$ 称为其后继点, 记为 $S(V_i, V_{i,j})$; 同样, 若以顺时针方向遍历, 则遇到第一个环上点 $(V_{i,j+1})$ 称为其前驱点, 记为 $P(V_i, V_{i,j})$. 有 $S(V_i, V_{i,j}) = (V_{i,j-1}), P(V_i, V_{i,j}) = (V_{i,j+1})$.

收稿日期: 2004—12—20

作者简介: 占自才(1969—), 男, 江西乐平人, 讲师.

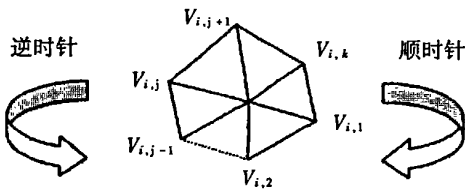


图 1

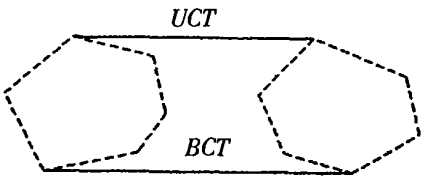


图 2

定义 3 对于点集 V , 若存在凸多边形, 它的定点均为点集 V 中的点 (如图 2 中以虚线相连的两个凸多边形), 且 V 中所有其它点都在该凸多边形的内部, 则该凸多边形称为点集 V 的凸壳, 记为 $CH(V)$.

定义 4 给定 V_L, V_R 两个左右点集 (右点集 X 坐标均不小于左点集 X 坐标), 设 P_1 是 $CH(V_L)$ 上的点, P_2 是 $CH(V_R)$ 上的点, 如果 $CH(V_L)$ 与 $CH(V_R)$ 上的点均不在有向直线 $P_1 P_2$ 的左边, 则称线段 $P_1 P_2$ 是点集 V_L 和 V_R 的顶部切线, 记为 UCT (如图 2 所示), 并记 P_1 为 $P_{top}(CH(V_L))$ 、 P_2 为 $P_{top}(CH(V_R))$; 同样, 若 $CH(V_L)$ 与 $CH(V_R)$ 上的点均不在有向直线 $P_1 P_2$ 的右边, 则称 $P_1 P_2$ 线段是点集 V_L 和 V_R 的底部切线, 记为 BCT , 并记 P_1 为 $P_{bottom}(CH(V_L))$ 、 P_2 为 $P_{bottom}(CH(V_R))$.

定义 5 给定定点 $V_i = (X_i, Y_i), V_j = (X_j, Y_j)$, 如果 $(X_i < X_j) \text{ or } ((X_i = X_j) \& (Y_i < Y_j))$, 则称按 X 坐标 $V_i < V_j$; 同理可定义称按 Y 坐标 $V_i < V_j$.

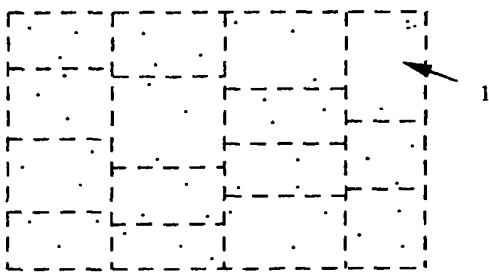


图 3

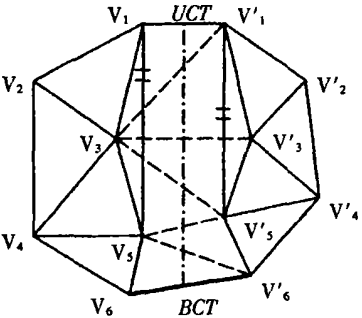


图 4

3.2 分块快速 Delaunay 三角剖分算法

设散点集合为 $W = (V_1, V_2, \Delta, V_N)$, 各算法分块分步叙述. 先介绍分区算法.

步 1: 如图 3 所示. 先对散点按 X 坐标进行排序, 依定义 5 使之按 X 坐标有 $V_1 < V_2 < \dots < V_N$; 按 X 坐标把点集分成 $\lfloor \sqrt{N/3} \rfloor + 1$ 份, 其中 $\lfloor \sqrt{N/3} \rfloor$ 表示不大于 $\sqrt{N/3}$ 的最大整数. 前面

3 分块三角剖分算法

在前面介绍了一些定义和符号的基础上, 下面给出 Delaunay 三角剖分双向分块快速算法.

3.1 散点分区

为了使算法能够分块进行, 必须把给定散点进行划分 (在划分之前必须剔除重复点). 传统的点二分法是把散点按 X 坐标分成 2^N 部分 (又称中点划分), 本文使用的一种新的划分方法, 就是先按 X 坐标进行大划分, 在按 Y 坐标进行小划分 (如图 3). 划分的结果使得只有 “1” 一个块有可能不是 3 个点 (N 除以 3 余数为 1 时有一个点, 余数为 2 时有两个点, 余数为零时有 3 个点), 其它块均有三个点. 把它门直接连成三角形或线段, 显然每一个块内点都各自组成 $DT(V)$ (单独一个点或单独一条线段也可作为一个 $DT(V)$), 然后可以进行递归合并, 但这样耗内存太大, 本文采用循环合并.

$\lfloor \sqrt{N/3} \rfloor$ 份中的每一份都有 $3 \times \lfloor \sqrt{N/3} \rfloor$ 个点, 最后一份有 $N - 3 \times \lfloor \sqrt{N/3} \rfloor \times \lfloor \sqrt{N/3} \rfloor$ 个点, 并做好标记, 使第 i 份点集记为: $W_i = (V_{(i-1) \times 3 \times \lfloor \sqrt{N/3} \rfloor + 1}, V_{(i-1) \times 3 \times \lfloor \sqrt{N/3} \rfloor + 2}, \Delta, V_k)$, 其中当 $1 \leq i \leq \lfloor \sqrt{N/3} \rfloor$ 时, $K = i \times 3 \times \lfloor \sqrt{N/3} \rfloor$, $i = \lfloor \sqrt{N/3} \rfloor + 1$ 时, $K = (i - 1) \times 3 \times \lfloor \sqrt{N/3} \rfloor + N - 3 \times \lfloor \sqrt{N/3} \rfloor \times \lfloor \sqrt{N/3} \rfloor$. 显然 $W = (W_1, W_2, \dots,$

$W_{(3 \times \lfloor \sqrt{N/3} \rfloor + 1)}$.

步 2:如图 3 按 Y 坐标进行小划分.对每一个点 W_i 集按 Y 坐标进行排序,依定义 5 使之按 Y 坐标有 $V_{(i-1) \times 3 \times \lfloor \sqrt{N/3} \rfloor + 1} < V_{(i-1) \times 3 \times \lfloor \sqrt{N/3} \rfloor + 2} < \Delta < V_K$, 其中 K 值意义与步 1 相同.

步 3:在每一个 W_i 点集内,依点的下标顺序每三个点直接连接好一个三角形,并做好三角形顶点、边的相互关联记录(数据结构后面再详细介绍).显然,连成的三角形就是该三个顶点的一个 Delaunay 三角剖分,每三个点集依次记为 $W_{i1}, W_{i2}, \Delta, W_{iK}$.其中当 $1 \leq i \leq \lfloor \sqrt{N/3} \rfloor$ 时, $K = \lfloor \sqrt{N/3} \rfloor$; 当 $i = \lfloor \sqrt{N/3} \rfloor + 1$ 时, $K = [(N - 3 \times \lfloor \sqrt{N/3} \rfloor \times \lfloor \sqrt{N/3} \rfloor) / 3]$,而且,当 N 除以 3 有余数时, K 还要加 1,且 W_{iK} 点集只有余数个点(1 或 2).

至此,点分区算法就完成了,下面接着分析合并算法,如图 4 所示, A 与 A' 点集都是已经进行了 Delaunay 三角剖分的,分别设为 $DT(A)$ 和 $DT(A')$,现在合并它们,我们先进行纵向,再进行横向合并,合并算法如下(我们这里以横向合并算法来说明,纵向合并类似,可仿照进行):

步 1:求 $CH(A)$ 与 $CH(A')$ 的上下切线(如图 4 所示).而 $CH(A)$ 与 $CH(A')$ 是前一轮合并时就求好了的.在求得了上下切线之后再求取 $CH(A - A')$,其中 $A - A'$ 点集等于 A 与 A' 的点集之和.

步 2:从上切线($V_1 - V'_1$)开始,向下进行搜索,使与 $V_1 - V'_1$ 两点相邻的所有点中的一个点(这里是 V_3 点)与 $V_1 - V'_1$ 两点一起构成一个三角形($\Delta V_1 V_3 V'_1$),并使得其它与 $V_1 - V'_1$ 两点相邻的点不在该三角形决定的外接圆以内. $\Delta V_1 V_3 V'_1$ 就是一个新的合格三角形.

步 3:在步 2 求得的 $\Delta V_1 V_3 V'_1$ 中,考虑新连成的边 $V_3 - V'_1$ 的对顶点 V_1 ,有没有点 V_1 所在的原来的边在 $\angle V_3 V_1 V'_1$ 里面.若没有,跳到步 4,从边 $V_3 - V'_1$ 继续向下搜索.若有,则除掉这条边,并改变其相关联的点、三角形的设置(显然, $V_1 - V_5$ 不相邻了, $\Delta V_1 V_3 V_5$ 没有了,如图 4 所示).

步 4:判断新添加的边(这里是 $V_3 - V'_1$ 边)是否等于下切线 BCT ,若等于,则 $DT(A)$ 和 $DT(A')$ 合并结束,得到 $DT(A - A')$,跳到步 1 进行新的合并,直到 $W = (A, A')$,则点集 $A - A'$ 的个数等于 N ,则所有合并才算完成,退出该合并循环算法;若不等于,以新添加的边(这里是 $V_3 - V'_1$ 边)开始,跳到步 2 继续新的搜索.

若是进行纵向合并,需要 $\lfloor \sqrt{N/3} \rfloor + 1$ 步大循环,每一个大循环内就是对点集进行 Delaunay 三角剖分,也就是每一个小循环算法与横向合并类似,只是步 4 中的退出小循环标志有所改变.在此,当合并后的点集 $A - A'$ 的数目等于点集 W_i 的数目则退出小循环.另外,上下切线改成左右切线,向下搜索改成从左到右搜索等.纵向合并完后,按上述步 1 ~ 步 4 再进行横向合并,则整个算法过程就算结束.

3.3 算法的数据结构

散点的数据结构:输入的散点可以用一般的数组存放,运算结果中的点因为要考虑一些

关联,可用链表存放,并可以搜索到相邻点,相邻点按双向链表存放(如表 1 所示),顺序按逆时针(如图 1).因为相邻点之间就表示有一条边,所以边不用单独存放.

如果只考虑三角网,而不用单独考虑三角形,则只需点之间的关联关系就可以,也即只需要点的数据结构就可以.但一般 Delaunay 三角剖分都需要以三角形的形式作为输出结果,所以下面给出三角形的数据结构,它也以链表的形式存放(如表 2 所示).

表 1 双向链表存放

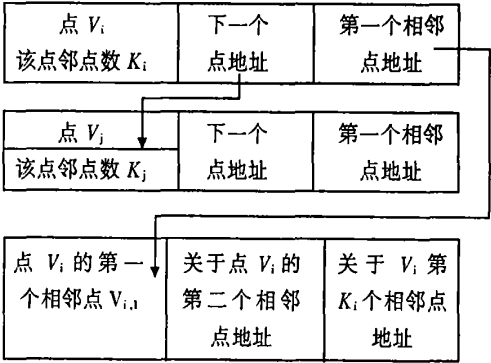
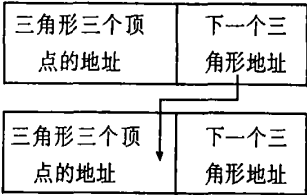


表 2 单向链表存放



4 算法的效率分析与结论

本算法采用了横向与纵向双向分块的方法,能有效地减少每一步合并的运算时间.可以想象,若只考虑横向分块,则大部分的一小块几乎在一条竖

直的直线上,合并前小块的三角剖分在合并后都几乎要改变,显然就增加了每一步的运算时间.采用双向分块,合并前小块中的三角形在合并后,可能大部分都是合并后的三角形,其状态改变少,每步运算时间自然少.另外,在合并时我们应用到了一个上下切线的概念,有很多文献上也用到这个概念^[4],只是本文在定义它时与别的有所不同,甚至认为用其它文献中提供的定义来设计算法时会出现错误,在算法程序设计时也得到了验证,但这些都不是本文的重点,不再详述.

在分小块时,本文都把它分成只有 3 个点的点集(除最后一个小块有可能不是 3 个点外),这样使得合并前每一小块的三角剖分非常简单快速.为了算法简单,合并的时候采用先纵向后横向合并.每次合并都是由下到上或由左到右合并,当每一轮合并小块数是单数时,最后一块可直接当着已合并的块加入合并后的块中.这样一来,本来是递归的算法很简单地变成了循环的算法,占用内存自然减

少,算法效率也会相应提高.显然,本文的算法时间度是 $N\log_2 N$,符合时间度最小原则.

本算法在实践应用中表明特别适合于几 $M \sim$ 几百 M 点数的 Delaunay 三角剖分,速度较其它算法优势明显.要想更进一步提供运算效率,合并时可以采用纵向与横向合并交错进行,但算法的复杂度就会大大提高.

参考文献:

[1]J G Gossonnat· Geometric structures for three dimensional shape representation· ACM Transactions on Graphics,1984,(4)
[2]L De Floriani· An on-line algorithm for constrained Delaunay triangulation· CVGIP,1992,(3).
[3]蔡志刚,金其杰· 二维保持边界的 Delaunay 三角剖分[C]·第 6 届全国图像图形学学术会议论文集,1992,(110~113).
[4]周培德· 计算几何——算法分析与设计[M]· 北京:清华大学出版社,2001.

A Speeding Double Direction Marking Block Algorithm for Delaunay Triangulation

ZHAN Zi-cai

(School of Electrical and Electronic Eng., East China Jiaotong Univ., Nanchang 330013, China)

Abstract: This paper introduces a speeding double direction marking block Algorithm for Delaunay Triangulation. Differing from other Algorithm, its characteristics is speediness of calculating, having time degree for $O(N\log^2N)$ and easy realization of the designs. The scattered data points are divided a lot of point-blocks and every one has only three points or two points. We process Delaunay Triangulation about every point-block and merge them which are conjoint in place. Then the paper introduces the data structures about the Algorithm and explains its maneuverability.

Key words: double direction marking block; speediness, merging; Delaunay triangulation; data structures