面向任意分布点云数据的二维 Delaunay 快速构网算法

王 雯[1)], 苏天赟[2)*], 王国宇[1)]

[1)] (中国海洋大学信息科学与工程学院 青岛 266100)
[2)] (国家海洋局第一海洋研究所海洋信息与计算中心 青岛 266061)
(sutiany@fio.org.cn)

摘 要: 为了更好地提高对二维点云数据的 Delaunay 构网效率, 并充分考虑点云数据规模庞大、分布多样的特点, 提出一种 Hilbert 曲线与多重网格划分相结合的算法. 首先通过多重网格划分解决规则网格对非均匀点集划分程度无法统一的问题; 其次通过添加控制点和采用 Hilbert 曲线顺序遍历网格的方式, 避免逐行遍历网格时产生大量需要重复创建和删除的狭长三角形的情况; 最后通过调整相邻网格间 Hilbert 曲线遍历顺序, 避免遍历过程的“跳跃”现象, 降低相邻网格插入点的点定位搜索步长. 实验结果表明, 与 CGAL、规则网格和多重网格划分算法相比, 该算法的构网效率对于分布均匀和非均匀的点云数据都有明显提升.

关键词: Delaunay 三角网; Hilbert 曲线; 网格划分; 多重网格; 点云数据
中图法分类号: TP391

Rapid 2D Delaunay Triangulation Algorithm for Random Distributed Point Cloud Data

Wang Wen[1)], Su Tianyun[2)*], and Wang Guoyu[1)]

[1)] (*College of Information Science and Engineering, Ocean University of China, Qingdao* 266100)
[2)] (*Marine Information and Computation Center, First Institute of Oceanography, State Oceanic Administration, Qingdao* 266061)

Abstract: Given the enormous scale and diverse distribution of 2D point cloud data, a multi-grid combined with Hilbert curve insertion algorithm is proposed for improving the efficiency of Delaunay Triangulation. First of all, the division problem caused by regular grid insertion scheme for non-uniform distributed point set can be resolved by the multi-grid one. Then, a large amount of conflicting elongated triangles, which have to be created and deleted many times, can be avoided by adding control points and adopting Hilbert curve traversing grids. Lastly, searching steps for positioning inserting point can be reduced by adjusting the Hilbert curve in adjacent grids for the avoided “jumping” phenomenon. The experimental results show that the efficiency of Delaunay triangulation by multi-grid combined with Hilbert curve insertion algorithm can be improved significantly for both uniform and non-uniform distributed point cloud data compared with CGAL, regular grid insertion and multi-grid insertion algorithm.

Key words: Delaunay triangulation; Hilbert curve; mesh division; multi-grid; point cloud data

在二维 Delaunay 三角剖分领域, 已经形成了 以逐点插入法[1]、分治算法[2]以及三角网生长法[3]

收稿日期：2014-08-18；修回日期：2015-02-11. 基金项目：国家科技重大专项(2011ZX05056-001-01); 海洋公益性行业科研专项(201205001). 王 雯(1989—), 女, 硕士研究生, 主要研究方向为科学计算可视化; 苏天赟(1977—), 男, 博士, 副研究员, 论文通讯作者, 主要研究方向为海洋信息技术; 王国宇(1962—), 男, 博士, 教授, 博士生导师, 主要研究方向为计算机图像处理与分析、模式识别.

为主的 3 种方法. 其中, 三角网生长法由于要频繁遍历数据点, 时间效率低且算法复杂, 目前已很少采用; 分治算法的优点是时间效率高, 但占用空间大, 算法复杂度高, 通常将并行计算与分治算法结合, 能够大幅提高算法的效率, 但并行计算对硬件要求较高, 不具有普遍性; 传统逐点插入法在定位插入点时要遍历平面上所有已生成的三角形, 当需要处理的点云数据较大时, 由于计算次数的增加和三角剖分程度的增加, 该算法的计算量大, 算法执行速度较慢[4]. 但逐点插入法算法简单、占用空间小、便于动态更新, 这些特有的优点使逐点插入法逐渐成为目前最流行的一种算法, 并且也出现了许多对逐点插入法的改进方法. 经过 Lee 等[5], Bowyer[6], Watson[7], Sloan[8], Macedonia 等[9], Floriani 等[10], Tsai[11]先后对逐点插入法不同程度的发展和完善[12], 形成了以创建初始包围盒、点定位、空腔扩展[6-7]、更新三角网为主要流程的逐点插入法. 在此基础上, 许多方法被提出用于进一步提高逐点插入法的效率. 网格索引法[13]利用网格索引提高点定位三角形的速度; 缺点是每形成一个新的三角形都要更新网格索引, 使整体效率降低, 若不使用网格索引, 算法效率会在很大程度上取决于点插入三角网的顺序. Amenta 等[14]提出了一种有偏随机插入顺序, 可以使外接圆判断次数降到很低, 但会在寻找点所在三角形时耗费大量时间. 文献[8]提出了划分均匀网格, 按网格将点排序后插入的方法. Liu 等[15], Zhou 等[16], Boissonnat 等[17], Buchin[18-19]在划分均匀网格排序的基础上提出了依照不同的空间填充曲线(如 Hilbert 曲线、Peano 曲线等)顺序遍历插入网格中点的方法, 进一步提高了划分均匀网格加点的效率; 但均匀网格划分只对于均匀分布点集具有较好的效果, 对于非均匀分布点集效果不佳. Devroye 等[20-21]提出了 k-D 树的网格划分方法, 相对均匀网格划分, 该方法能提高对非均匀分布点集的构网效率; 但 k-D 树本身算法复杂度高, 构建 k-D 树网格会耗费大量时间, 导致最终效率不高. 在此基础上, Lo[22]提出了多重网格划分算法, 该算法复杂度低, 对任意分布点集都有较高的构网效率; 但是由于对网格的处理采用逐列遍历的顺序和一次性插入网格中全部点的方式, 在构网过程中会产生很多不必要的狭长三角形, 构建和删除这些三角形会占用额外的时间, 从而使构网效率降低.

通过对二维 Delaunay 逐点插入算法效率的影响因素分析, 本文将多重网格划分与 Hilbert 曲线进行结合, 并采用添加控制点和调整相邻网格间 Hilbert 曲线遍历顺序的方式, 进一步提高构建 Delaunay 三角网的效率.

1 基于网格排序的逐点插入算法

基于网格排序的逐点插入法的构网时间主要分为三部分:

Part1. 对点插入顺序的预处理;

Part2. 找出插入点所在的三角形;

Part3. 用 Bowyer-Watson 算法[6-7]进行三角网的更新.

Part1 对于不是很复杂的点排序方法耗费的时间一般会比 Part2 和 Part3 的时间短很多, 在此不做重点讨论. 对于 Part2 和 Part3, 当插入点一个紧挨着一个插入时, 会使 Part2 的时间降到最短, 但是这样会形成很多狭长的三角形, 从而扩大受影响区域, 使插入点进行外接圆判断的次数过多, 造成 Part3 的时间成倍增加; 而当插入点顺序为随机排列时, Part3 的时间会降到最短, 但是会使插入点与最新更新的三角形相隔较远, 造成 Part2 的时间成几十甚至上百倍增加. 所以, 调整点的插入顺序, 使得插入点既相隔很近又能尽量减少生成的狭长三角形数量, 从而使 Part2 与 Part3 的时间之和降到最低, 是提高算法效率的重要因素.

根据上述时间规律, 规则网格划分对于分布较为均匀的点集能够较好地平衡 Part2 和 Part3 的时间关系[8]. 然而, 对于分布不均匀的点集, 由于在网格划分过程中会形成很多点集分布过密或者过疏的网格, 规则网格划分的构网效率相对较低. 文献[22]提出了一种多重网格划分算法, 用迭代划分的方式使网格划分更加合理, 对非均匀分布点集的构网效率有所提升; 但该算法网格遍历顺序仍然采用的是“左右右左”的逐行遍历形式, 如图 1 所示.

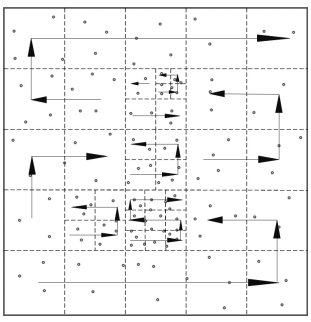


图 1 多重网格点的插入顺序

图 1 中的遍历网格顺序存在一定的弊端. 对于分布较均匀的点集, 当网格遍历到当前行时, 它会与前一行网格中的边界点连接形成很多狭长三角形(如图 2 中红色三角形), 这些三角形在以后点的插入过程中都需要删除, 会增加 Part3 的时间. 而且在搜索点所在三角形时, 由于这些狭长三角形比较密集, 也会增加 Part2 的时间. 除此之外, 多重网格构网过程中, 会出现插入点与包围盒的四个顶点形成很多狭长且需要删除的三角形(如图 3 所示)、以及相邻网格起始插入点 Part2 中搜索三角形数量较多(如图 4 中红色三角形)的情况.

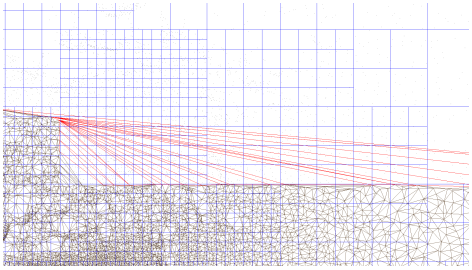


图 2　与前一行边界点形成大量狭长三角形

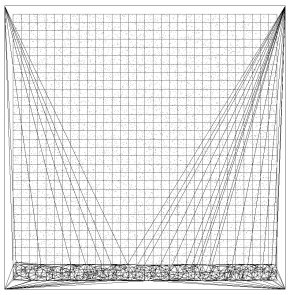


图 3　与包围盒四个顶点形成大量狭长三角形

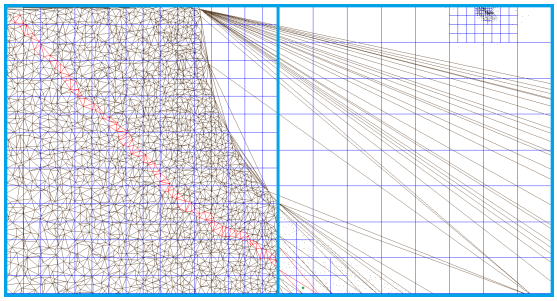


图 4　相邻网格插入点相距较远的现象

2　本文算法

根据基于网格的逐点插入算法存在的弊端和 Hilbert 曲线的空间局部性特征, 本文提出基于 Hilbert 曲线的多重网格构网算法: 先按照多重网格划分的规则来划分网格, 然后按照 Hilbert 曲线的路径来遍历网格, 结合控制点设置解决多重网格方法存在的问题, 提高 Delaunay 构网效率.

2.1　网格划分规则

多重网格划分的目的是在尽量减少空网格的前提下, 保证每个网格中点的个数均小于等于 $N_{\max}$ (经实验验证, 对于大规模点云数据, $N_{\max}$ 在 50~200 都可以取得较高的构网效率, 且差别不大. 本文 $N_{\max}$ 取 50), 使前后插入点之间相隔的三角形较少, 进而减少定位点所在三角形的时间. 因此, 当网格中点数大于 $N_{\max}$ 时, 就要进行下一级网格的划分. 设原始点集所在区域形成的零级网格为 G^0, 则对于网格 G^0 的每一个网格划分单元, 都会形成一个迭代划分过程, 每个迭代过程生成的网格单元有如下关系(m 为停止划分的当前网格级数): $G^0 \supset G^1 \supset \cdots G^{m-1} \supset G^m$.

为了满足 Hilbert 曲线遍历的要求, 必须保证每一级网格的网格数 $N_{\text{grid}} = 4^r$, 每行和每列的网格数 $N_{\text{side}} = 2^r$, 其中 r 为要划分 Hilbert 网格的阶数且 $r \in \mathbb{N}$. 对于含有 N_{p} 个点的满足网格划分条件($N_{\text{p}} > N_{\max}$)的网格 G^{m-1}, 设置每个网格的平均点数 N_{ave} (根据文献[22], 在 4~16 时构网效率最高), 由集合 P^{m-1} 中的点数 $N_{\text{p}} = N_{\text{ave}} \times 4^r$, 可求得: $r = \left\lfloor \log_4 \dfrac{N_{\text{p}}}{N_{\text{ave}}} \right\rfloor$. 为了避免 r 的取整操作使网格平均点数 $\dfrac{N_{\text{p}}}{4^r}$ 过大, 需要进行如下判断:

若 $\dfrac{N_{\text{p}}}{4^r} > 16$, 则 $r = r + 1$, 这样可以保证平均网格点数在 4~16.

用 $P^{m-1} = \{p_i^{m-1} = (x_i, y_i) \mid i = 1, 2, \cdots, N_{\text{p}}\}$ 表示落在网格 G^{m-1} 中的点, 令 $x_{\max}$, $x_{\min}$, $y_{\max}$, $y_{\min}$ 分别为 x_i 和 y_i 的最大最小值, 则集合 P^{m-1} 中点的横坐标跨度 $X_{\text{span}} = x_{\max} - x_{\min}$, 纵坐标跨度 $Y_{\text{span}} = y_{\max} - y_{\min}$, 网格的行宽 $W_{\text{r}} = \dfrac{Y_{\text{span}}}{N_{\text{side}}}$, 列宽 $W_{\text{c}} = \dfrac{X_{\text{span}}}{N_{\text{side}}}$.

最终, 对于分配点集合 P^{m-1} 中点到网格集合 $G^m = \{(row, col) \mid row, col = 1, 2, \cdots, N_{\text{side}}\}$ 中的分配规则为 $p_i^{m-1} \to \left(\dfrac{y_i - y_{\min}}{W_{\text{r}}} + 1, \dfrac{x_i - x_{\min}}{W_{\text{c}}} + 1 \right)$.

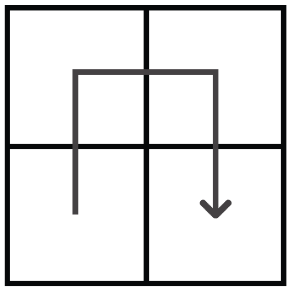
上述为在 G^{m-1} 区域继续划分网格 G^m 的规则，基于该规则对整个数据点集，从 $m=1$ 开始进行迭代网格划分. 每当在 G^{m-1} 区域划分网格 G^m 时，逐个判断每个 G^m 网格单元中点数是否满足网格划分条件，若满足，则按照上述规则在此 G^m 网格单元区域继续迭代划分网格 G^{m+1}；否则，进行下一个 G^m 网格单元的判断，直到所有的网格判断和划分完毕.

2.2 Hilbert 曲线路径

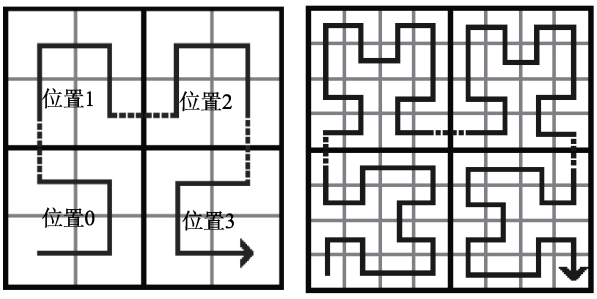
Hilbert 曲线是由德国数学家 David Hilbert 在 1891 年提出的一种空间分形曲线，其特点是充满整个空间并能维持数据的局部性特征. 与其他空间分形曲线相比，Hilbert 曲线拥有最优的聚集特性[23].

本文将二维 Hilbert 曲线应用到多重网格划分的 Delaunay 构网过程中. 根据李晨阳等[24]对 N 维 Hilbert 曲线的定义，n 阶二维 Hilbert 曲线 H_n ($n=1,2,3,\cdots$)是一条可以填充一个 $2^n\times 2^n$ 平面区域的曲线，它可以不自交地通过格形方块每一格点中心，有且仅有一次，并充满整个格形方块区域[25].

定义 **1.** 二维 Hilbert 细胞. 当 n=1 时，其对应的二维 Hilbert 曲线就是任意阶数二维 Hilbert 曲线的一个细胞，如图 5a 所示.



a. Hilbert细胞/1阶Hilbert曲线



b. 2阶Hilbert曲线　　c. 3阶Hilbert曲线

图 5　1~3 阶 Hilbert 曲线

定义 **2.** 二维 Hilbert 基因. 将任意一个二维 n 阶 Hilbert 曲线按照图 5b 划分为 4 等份，并把每一份的位置用变量 l 表示，l=0,1,2,3. 则一个二维 Hilbert 基因是一系列变换指令信息列表，其中 l 为 H_n 的位置变量，它控制如何由 H_{n-1} 生成 H_n. 基因列表如表 1 所示.

表 1　Hilbert 基因列表

l	变换	方向
0	顺时针旋转 90 °	求反
1	不变	不变
2	不变	不变
3	逆时针旋转 90 °	求反

表 1 中，“变换”是指 H_n 所对应的位置 l 部分以 H_{n-1} 怎样的旋转变换来填充，“方向”指对应的位置 l 的部分以变换好的 H_{n-1} 填充后在 H_n 中是否保持原来的方向. 按照基因列表填充 H_n 的 4 个区域后，如图 5b, 5c 所示，将位置 0~1, 1~2, 2~3 之间的曲线间断处连接，即完成了 H_{n-1} 生成 H_n 的过程，进而可以得到 n 为任意正整数的二维 Hilbert 曲线.

由图 5 可知，Hilbert 曲线的空间局部性特征可以使每一个网格与其前后数个通过的网格保持很近的距离. 所以以 Hilbert 曲线顺序遍历网格可以使连续 2 个插入点不会相隔太远，而且也不会形成太多狭长的三角形，从而可以提高 Delaunay 三角网的构网效率.

但是当 Hilbert 曲线顺序运用到多重网格中时，对于每一级网格，如果 Hilbert 曲线遍历顺序一直按照之前从左下角到右下角的形式，则会出现“跳跃”的情况. 如图 6 所示，不对 G^{m+1} 的网格遍历顺序作变换，2 个连续的 G^m 单元可能会出现前一个单元最后一个插入点 p 与后一个单元第一个插入点 q 相隔较远的情况，这样会在点 q 寻找所在三角形时增加判断的次数而降低效率. 所以，当遇到这种情况时，要在原 Hilbert 曲线的基础上作一定的

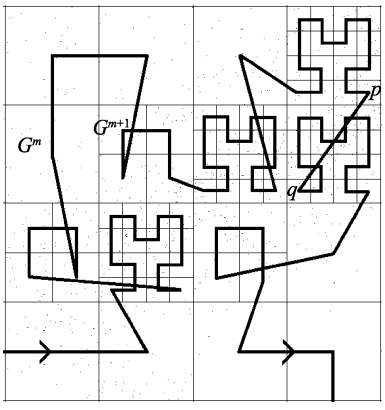


图 6　“跳跃”现象

变换, 使p和q能够位于相邻网格中, 从而提高效率.

定义 3. 二维多重 Hilbert 基因. 一个二维多重 Hilbert 基因是一系列变换指令信息列表, 它控制对G^m的一个网格单元所占区域划分形成G^{m+1}时G^{m+1}的网格遍历顺序, 设G_i^m为G^{m-1}区域按 Hilbert 曲线顺序遍历的第i个G^m网格, 如果要在G_i^m区域继续划分下一级网格G^{m+1}, 则基因列表如表 2 所示.

表 2　多重 Hilbert 基因列表

位置关系		遍历 G^{m+1} 的起止位置
G_{i-1}^m 与 G_i^m	G_i^m 与 G_{i+1}^m	
左→右	下→上	左下角→左上角
左→右	上→下	左下角→右下角
右→左	下→上	右上角→左上角
右→左	上→下	右上角→右下角
上→下	左→右	右上角→右下角
上→下	右→左	右上角→左上角
下→上	左→右	左下角→右下角
下→上	右→左	左下角→左上角
左→右	左→右	左下角→右下角
右→左	右→左	右上角→左上角
上→下	上→下	右上角→右下角
下→上	下→上	左下角→左上角
	左→右	左下角→右下角
左→右		左下角→右下角
	上→下	右上角→右下角
上→下		右上角→右下角
	右→左	右上角→左上角
右→左		右上角→左上角
	下→上	左下角→左上角
下→上		左下角→左上角

表 2 中, 第 1 列的空白代表i=0, 即G_i^m为第一个遍历的网格; 第 2 列的空白代表i取最大, 即G_i^m为最后一个遍历的网格.

按照表 2 对网格遍历顺序进行变换后, “跳跃”现象最终被避免. 图 7 所示为图 6 网格划分经过多重 Hilbert 基因列表变换后, 最终 Hilbert 曲线在多重网格中形成的路径顺序, 可以看出, 变换后的路径避免了“跳跃”现象.

2.3　控制点

为了减少插入点与包围盒 4 个顶点以及已完成 Delaunay 构网的区域边界形成的狭长三角形数量, 本文从全部数据点集中选取控制点集, 在按网格插入全部数据点之前先将控制点集插入三角网

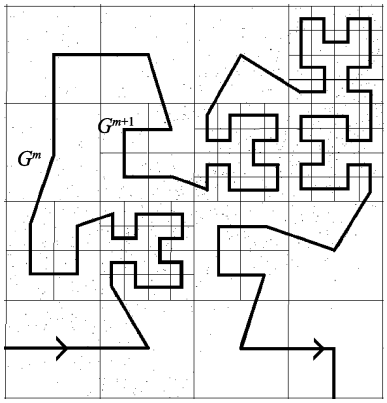
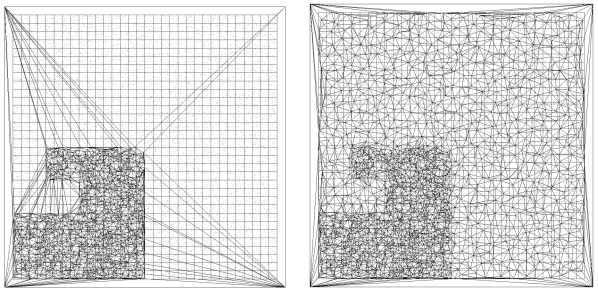


图 7　本文算法网格遍历顺序

中. 这样, 在插入非控制点时非控制点就会与控制点相连, 从而减少了与包围盒 4 个顶点和已完成 Delaunay 构网的区域边界形成的狭长三角形数量. 控制点集的选取规则如下：

对于每个有点的网格单元G^1, 只取该网格点集P^1的第一个点, 无点的网格单元直接跳过.

控制点选取的顺序是按照 Hilbert 曲线的逆序遍历网格G^1(正序为从左下角→右下角的顺序, 反序为从右下角→左下角的顺序), 因为逆序遍历可以使控制点集中最后一个插入点与第一个插入的非控制点在同一个G^1中, 使后者搜寻其所在三角形时能够快速搜索到. 图 8 所示为均匀分布的 10 000 个数据点的点集不添加控制点插入第 2 000 个点时形成的 Delaunay 三角网; 图 8b 所示为相同点集添加控制点后插入与图 8a 相同位置点时形成的 Delaunay 三角网. 从图 8 可以看出, 与不添加控制点相比, 添加控制点会使插入点与包围盒顶点以及已完成 Delaunay 构网的区域边界形成的狭长三角形数量大大减少.



a. 不添加控制点　　b. 添加控制点

图 8　插入点 2 000 点时 Delaunay 三角网情况

2.4　本文算法流程

本文提出的 Delaunay 构网整体排序过程伪代码如下:

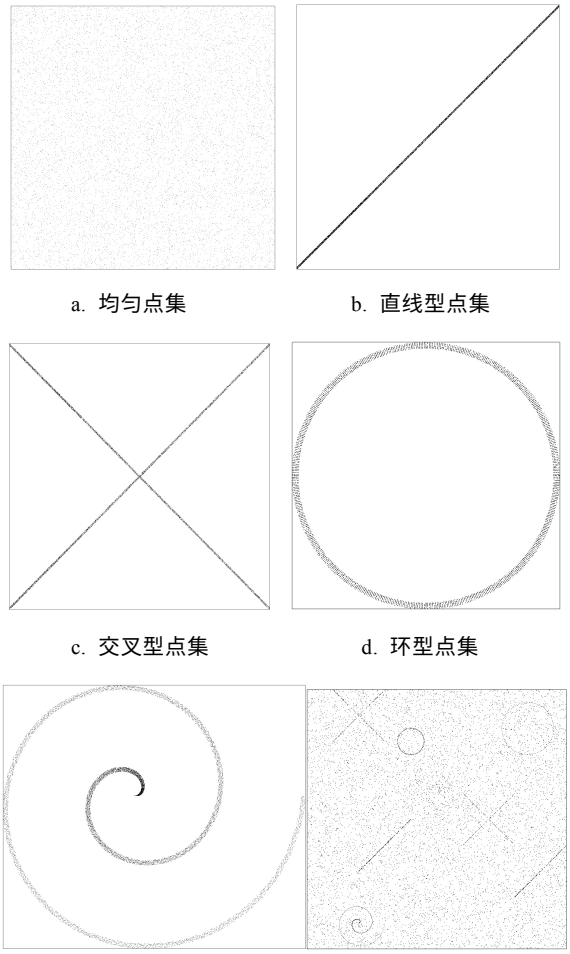
```
begin
  if 输入点集点数大于 Nmax, then
    m←1;
    griddivide(m);
  else
    将输入点集放入非控制点数组;
  end if
  按控制点数组中存放点顺序的逆序将其中的点
    依次插入三角网;
  按非控制点数组中存放点顺序将其中的点依次
    插入三角网;
end
```

函数 *griddivide*(*m*)的伪代码如下:

```
begin
  按网格划分规则划分 G^m, 并为 G^m 分配点;
  按照调整后的 Hilbert 曲线确定遍历 G^m 顺序;
  i←0;
  while G^m 未遍历结束 do
    if m==1, then
      将第 i 个 G^1 网格单元第一个点放入控制点
        数组;
      将该点从该网格中删除;
    end if
    if 第 i 个 G^m 网格中点个数大于 Nmax, then
      griddivide(m+1);
    else
      将该 G^m 网格单元中点放入非控制点数组;
    end if
    i++;
  end while
end
```

3 实验及结果分析

为了测试本文算法的效率, 对点数为 1 000 万的均匀点集, 非均匀点集(直线型、交叉型、环型、螺旋型)以及混合型点集(图 9a~9f, 图中为点集点数抽稀到 10 000 的情况)分别用 CGAL、规则网格划分算法(简称为 RG)、多重网格划分算法(简称为 MG)[22]、Hilbert 曲线与多重网格结合算法(简称为 HG)进行了实验, 实验环境为 Intel(R) Xeon(R) CPU E5-1620 0 @ 3.60 Hz, 8 GB 内存, win7, 64 位操作系统, 实验结果如表 3 所示. 根据第 1 节中所述, 逐点插入法的构网时间主要由 Part2 和 Part3 组成, 其中 Part2 的时间与每个点的首三角形与该点所在三角形之间要判断的三角形个数的平均值(the average number of passed triangles for point positioning, NP)成正比, Part3 的时间与每个插入点删除的三角形数量的平均值(the average number of removed conflicting triangles, NR)成正比. 在点排序算法不是很复杂的情况下, NR 和 NP 的值越小, 构网效率越高. 为使对比结果更加直观, 充分反映网格遍历顺序对构网效率的影响, 表 3 中的 MG 均采用与 HG 完全相同的网格划分, 只是网格遍历顺序不同; RG 则采用与 HG 中 G^1 完全相同的网格划分. 由于 CGAL 中的 Delaunay 建模方法没有提供 NR 和 NP 的统计信息, 因此表 3 只列出了 CGAL 的总时间, 来与其他 3 种方法进行对比.



a. 均匀点集 b. 直线型点集 c. 交叉型点集 d. 环型点集 e. 螺旋型点集 f. 混合型点集

图 9 不同分布的点集

从表 3 可以看出, 对于均匀分布点集, 由于 RG 和 MG 的网格划分结果是完全相同的, 它们的 NP, NR 值完全相同, 效率也基本相同; CGAL 的构网效率略高于 MG; HG 的构网效率是 3 种算法中最高的, NP, NR 值较 MG 有了显著的降低. 就构网时间来看, HG 能较 MG 减少约 20%的时间消耗. 对于直线型和交叉型分布点集, RG 的效率最低, 构网时间大约是 HG 的 1.5 倍; MG 的效率比 CGAL

表 3　不同点集 4 种算法效率对比①

算法	对比参数	点集类型					
		均匀	直线型	交叉型	环型	螺旋型	混合型
CGAL	总时间/ms	14 461	14 929	14 508	14 508	14 586	14 539
RG	NP	5.72	23.46	17.21	10.47	7.09	7.71
	NR	6.07	4.27	4.49	4.87	6.57	5.81
	点排序时间/ms	1 373	577	577	530	436	1 279
	点插入时间/ms	13 323	19 406	16 567	13 510	14 149	13 821
	总时间/ms	14 914	20 030	17 207	14 102	14 648	15 319
MG	NP	5.72	4.80	4.58	4.76	4.51	5.54
	NR	6.07	5.22	5.44	5.51	5.98	5.89
	点排序时间/ms	1 622	2 059	2 075	1 638	1 419	1 685
	点插入时间/ms	13 432	11 965	12 200	1 2028	12 168	13 057
	总时间/ms	15 070	14 040	14 290	1 3682	13 603	14 757
HG	NP	4.05	4.24	3.96	3.90	3.68	4.06
	NR	4.15	4.62	4.57	4.38	4.78	4.17
	点排序时间/ms	1 435	2 106	2 106	1 607	1 420	1 528
	点插入时间/ms	10 218	10 952	10 639	9 844	10 280	10 156
	总时间/ms	11 887	13 120	12 823	11 529	11 778	11 918

略高; HG 效率最高, 其 NP, NR 值都比 MG 降低了约 0.5, 构网时间减少了 10%. 对于环型和螺旋型分布点集, CGAL 和 RG 效率相当; MG 比 CGAL 和 RG 效率高 5%左右; HG 的 NP, NR 值相比 MG 降低了 1 左右, 所以效率提升也较直线型和交叉型分布明显, 构网时间较 MG 降低约 15%. 对于混合型点集, 与均匀分布点集相似, CGAL, RG, MG 的效率都差别不大, 而 HG 的 NP, NR 值都较 RG 和 MG 有较为明显的下降, 效率提高约 20%.

从不同算法的角度看, CGAL 随点集分布的不同, 构网时间几乎不产生变化, 其构网速度基本都稳定在 690 点/ms; RG 随点集分布的不同构网效率变化非常大, 在对直线型点集进行构网时, 速度约为 500 点/ms, 而对环型点集进行构网时, 速度约为 710 点/ms; MG 随点集分布的不同构网效率变化也不大, 其构网速度与 CGAL 相当, 对于均匀分布点集和混合型分布点集略慢, 约为 670 点/ms, 而对于非均匀分布点集略快, 约为 720 点/ms; HG 中直线型和交叉型分布相对于其他分布对构网效率的提升幅度略小, 构网速度约为 770 点/ms, 而其他分布的构网速度约为 850 点/ms. 从表 3 可以明显地看出, 与其他 3 种算法相比, 对于不同分布的点集, HG 的构网效率都是 4 种算法中最高的.

4　结　语

针对二维 Delaunay 构网过程中形成大量狭长三角形和相邻网格插入点相距较远等问题, 本文提出了一种 Hilbert 曲线与多重网格划分结合的算法. 该算法通过划分多重网格、添加控制点以及采用调整后的 Hilbert 路径遍历网格等方式, 使效率得到了进一步的提升. 实验结果表明, 不论点集如何分布, 本文算法的 NP, NR 值都比规则网格、多重网格要低, 说明该算法中要反复删除和创建的狭长三角形的数量以及相邻插入点间隔的三角形数量相对较少, 所以最终的构网时间也较这两种算法有所减少. 与目前普遍认为较快的 CGAL 构建二维 Delaunay 三角网的方法相比, 对于各种分布的点集, 本文算法的效率仍然能高出 10%~20%, 说明该算法对于任意分布的二维大规模点云数据的 Delaunay 建模具有较好的应用价值.

参考文献(References):

[1] Lawson C L. Software for C^1 surface interpolation[C] //Proceedings of the Symposium on Mathematical Software. New York: Academic Press, 1977: 161-194

[2] Lewis B A, Robinson J S. Triangulation of planar regions with

① 总时间=点排序时间+点插入时间+其他时间, 这里的其他时间包括创建初始包围盒等过程, 由于其占总时间比重太少, 所以没有列在表中进行对比

applications[J]. Computer Journal, 1978, 21(4): 324-332

[3] Green P J, Sibson R. Computing Dirichlet tessellations in the plane[J]. Computer Journal, 1978, 21(2): 168-173

[4] Sapidis N, Perucchio R. Delaunay triangulation of arbitrarily shaped planar domains[J]. Computer Aided Geometric Design, 1991, 8(6): 421-437

[5] Lee D T, Schachter B J. Two algorithms for constructing a Delaunay triangulation[J]. International Journal of Computer & Information Sciences, 1980, 9(3): 219-242

[6] Bowyer A. Computing Dirichlet tessellations[J]. Computer Journal, 1981, 24(2): 162-166

[7] Watson D F. Computing the *n*-dimension Delaunay tessellation with application to Voronoi polytopes[J]. Computer Journal, 1981, 24(2): 167-172

[8] Sloan S W. A fast algorithm for constructing Delaunay triangulations in the plane[J]. Advances in Engineering Software, 1987, 9(1): 34-55

[9] Macedonia G, Pareschi M T. An algorithm for the triangulation of arbitrarily distributed points: applications to volume estimate and terrain fitting[J]. Computers & Geosciences, 1991, 17(7): 859-874

[10] de Floriani L, Puppo E. An on-line algorithm for constrained Delaunay triangulation[J]. CVGIP: Graphical Models and Image Processing, 1992, 54(4): 290-300

[11] Tsai V J D. Delaunay triangulations in TIN creation: an overview and a linear-time algorithm[J]. International Journal of Geographical Information Systems, 1993, 7(6): 501-524

[12] Wu Xiaobo, Wang Shixin, Xiao Chunsheng. A new study for Delaunay triangulation creation[J]. Acta Geodaetica et Cartographica Sinica, 1999, 28(1): 28-35 (in Chinese)
(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)

[13] Wang Xing. Study on an algorithm for fast constructing Delaunay triangulation and 3D visualization in OpenGL environment[J]. Science Technology and Engineering, 2011, 11(9): 2070-2074 (in Chinese)
(王 星. 快速构建 Delaunay 三角网算法研究及 OpenGL 下三维可视化[J]. 科学技术与工程, 2011, 11(9): 2070-2074)

[14] Amenta N, Choi S, Rote G. Incremental constructions con BRIO[C] //Proceedings of the 19th Annual Symposium on Computational Geometry. New York: ACM Press, 2003: 211-219

[15] Liu Y, Snoeyink J. A comparison of five implementations of 3D Delaunay tessellation[J]. Combinatorial and Computational Geometry, 2005, 52: 439-458

[16] Zhou S, Jones C B. HCPO: an efficient insertion order for incremental Delaunay triangulation[J]. Information Processing Letters, 2005, 93(1): 37–42

[17] Boissonnat J D, Devillers O, Hornus S. Incremental construction of the Delaunay triangulation and the Delaunay graph in medium dimension[C] //Proceedings of the 25th Annual Symposium on Computational Geometry. New York: ACM Press, 2009: 208–216

[18] Buchin K. Organizing point sets: space-filling curves, Delaunay tessellations of random point sets, and flow complexes[D]. Berlin: Free University Berlin. Department of Mathematics and Computer Science, 2007

[19] Buchin K. Constructing Delaunay triangulations along space-filling curves[C] //Proceedings of the 17th Annual European Symposium on Algorithms. Heidelberg: Springer, 2009: 119-130

[20] Devroye L, Jabbour J, Zamora-Cura C. Squarish *k*-d trees[J]. SIAM Journal on Computing, 2000, 30(5): 1678-1700

[21] Devroye L, Lemaire C, Moreau J M. Expected time analysis for Delaunay point location[J]. Computational Geometry, 2004, 29(2): 61-89

[22] Lo S H. Delaunay triangulation of non-uniform point distributions by means of multi-grid insertion[J]. Finite Elements in Analysis and Design, 2013, 63: 8-22

[23] Frisken S F, Perry R N. Simple and efficient traversal methods for quadtrees and octrees[J]. Journal of Graphics Tools, 2002, 7(3): 1-11

[24] Li Chenyang, Duan Xiongwen, Feng Yucai. Algorithm for generating *n*-dimensional Hilbert curve[J]. Journal of Image and Graphics, 2006, 11(8): 1068-1075 (in Chinese)
(李晨阳, 段雄文, 冯玉才. *N* 维 Hilbert 曲线生成算法[J]. 中国图象图形学报, 2006, 11(8): 1068-1075)

[25] Chen Ningtao, Wang Nengchao, Chen Ying. Fast generation algorithm design and realization of Hilbert curve[J]. Mini-Micro Systems, 2005, 26(10): 1754-1757 (in Chinese)
(陈宁涛, 王能超, 陈 莹. Hilbert 曲线的快速生成算法设计与实现[J]. 小型微型计算机系统, 2005, 26(10): 1754-1757)