一种快速搜索散乱点云数据 *k* 邻近的算法

杜小雷　卓勇

（厦门大学物理与机电工程学院，福建 厦门 361005）

摘　要：针对逆向工程中散乱点云数据的 *k* 近邻搜索，提出一种快速搜索散乱点云 *k* 邻近点的算法。该算法根据点云数据的范围、点的总数及最近点数目 *k*，确定合适的立方体边长，采用空间划分策略，把数据划分成多个子立方体；然后用哈希表记录每个子立方体所包含的数据点及每个点所在的立方体索引号，并排除不包含数据的子立方体，以此确定邻近点的最佳搜索范围。实验结果表明：该算法有效的提高 *k* 近邻搜索的速度，同时保证了搜索结果的正确性。

关键词：*k* 邻近　逆向工程　点云

中图分类号：TP391　**文献标识码**：A　**文章编号**：1672-4801（2009）03-005-04

前言

由于散乱点云数据缺少明显的拓扑关系，因此需要通过 *k* 邻近搜索，建立散乱点间邻接关系的数据结构。利用每一点的 *k* 邻近结构来反映出待重建曲面在该点处的形状信息，如法矢和曲率特性等。随着测量设备的不断发展，高效率、高精度地采集样件的外形数据已可以实现，采集的数据点数目十分巨大，*k* 邻近搜索是基于散乱数据点集模型重建的关键步骤，*k* 邻近搜索的效率直接影响到逆向建模的效率，因而对此进行研究有重要的实际应用价值。

计算某点云数据的 *k* 邻近的最直观的方法是求出待选点与其余点的欧氏距离，并按升序排列，前 *k* 个点即为该点的 *k* 个最邻近点，然而这是一种耗时、低效的方法，用它来计算数据量巨大的点云的 *k* 邻近是不现实的。许多学者针对此问题进行了一些快速算法的研究，这些方法可分为两类：(1)利用点集 Voronoi 图来进行 *k* 个最近点的搜索[5]，但点集 Voronoi 图的计算量仍然非常大；(2)利用空间分块策略进行 *k* 个最近点搜索[3,4,6,7]，但文献[3]的方法既不能保证它的空间分块具有最佳或接近于最佳的搜索速度，也不能保证每个数据点都能找到 *k* 个最近邻域，而文献[6]只研究了平面点集的 *k* 个最近邻域搜索问题。

本文提出一种基于空间分块策略快速搜索散乱点云 *k* 邻近点的算法。该算法综合考虑了数据的大小、点的总数以及最近点个数 *k*，确定接近于最佳搜索速度的空间分块尺寸，并且在搜索策略上进行了优化改进，从而提高 *k* 邻近的搜索速度。

1 算法

本文算法首先根据点云数据的范围、点的总数及最近点数目 *k*，计算出子立方体边长 *L*，根据 *L* 将点云划分成多个子立方体，然后记录每个子立方体所包含的数据点及每个点所在的立方体索引号，并排除不包含数据的子立方体，最后利用子立方体内点的信息对每个数据点进行 *k* 近邻的搜索。

1.1 点云数据空间划分

首先读入点云数据，计算点云数据的 X，Y，Z 坐标的最小、最大值，从而形成一个与坐标轴平行的长方体包围盒，包围所有的数据点，然后计算子立方体边长 *L*，并根据 *L* 将长方体包围盒划分成 $m \times n \times l$ 个子立方体；判断每个数据点所在的子立方体，并将数据点的序号追加到该立方体对应的线性链表中，用哈希链表结构存放各个子立方体中包含的数据点。哈希表通过立方体栅格在 X，Y，Z，3 个方向的索引号直接定址。

设点云数据的最小坐标为 Min_x，Min_y，Min_z，最大坐标为 Max_x，Max_y，Max_z，则点云数据的子立方体边长 *L* 的计算公式为[1]：

$$L=\beta\cdot\sqrt[3]{\frac{k}{n}(\mathrm{Max\_x}-\mathrm{Min\_x})(\mathrm{Max\_y}-\mathrm{Min\_y})(\mathrm{Max\_z}-\mathrm{Min\_z})} \quad (1)$$

其中 β 为调节立方体边长 *L* 大小的比例因子，其最优值为 0.8～1.2[1]；*n* 为点云数据的总数，*k* 为邻近点的个数。

立方体边长 *L* 确定后，子立方体在三个坐标方向的个数为：

$$\begin{cases} m=\lceil \mathrm{Max\_x}-\mathrm{Min\_x})/\mathrm{L} \rceil \\ n=\lceil \mathrm{Max\_y}-\mathrm{Min\_y})/\mathrm{L} \rceil \\ l=\lceil \mathrm{Max\_z}-\mathrm{Min\_z})/\mathrm{L} \rceil \end{cases}$$

其中 $\lceil \bullet \rceil$ 为向上取整。

基金项目：福建省自然科学基金(2009J01266)

对于任意一点 P(x，y，z)，求该点所在的子立方体的哈希函数是：

$$\begin{cases} i = \lfloor (\mathrm{x} - \mathrm{Min\_x}) / L \rfloor \\ j = \lfloor (\mathrm{y} - \mathrm{Min\_y}) / L \rfloor \\ k = \lfloor (\mathrm{z} - \mathrm{Min\_z}) / L \rfloor \end{cases}$$

其中$\lfloor \bullet \rfloor$为向下取整；i，j，k 分别为该点所在立方体的 X,Y,Z 轴方向子立方体的索引号。

本文算法在空间划分之后，将不包含数据点的子立方体删除，从而减少了需要搜索的子立方体数量。以底盒部件为例，包围盒尺寸为 47mm×45mm×18mm，式(1)中取 β=0.8，则立方体边长 L 为 2.476mm，空间划分结果总共有 2888 个子立方体，去除不包含数据点的子立方体之后只剩 477 个子立方体，图 1 显示了只包含数据点的子立方体。

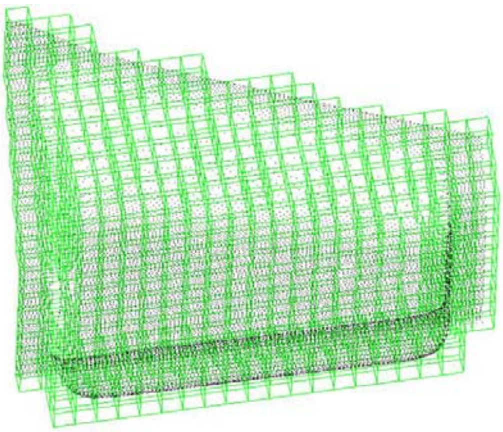

图 1 包含点云数据的子立方体

1.2 基本数据结构

如何快速对各数据点进行存储和查找，是提高 k 邻近搜索效率的关键，文献[7]所采用的八叉树结构，需要计算每个点的八叉树编码，在 k 邻近搜索之后的后续应用中，如计算曲率、法矢等过程中又需要重新解码，这不仅繁琐而且耗时。本文算法采用哈希链表结构来记录子方格中各数据点的信息，充分运用了哈希表快速查找及存储的优点[8]，从而提高了 k 邻近搜索的效率。

(1)数据点集链表。该表保存每个点的三坐标信息，表中每个元素结构如下：

```
typedef struct ListNode
{
    double  x, y, z;  //数据点的三坐标值
    struct ListNode *next; //数据点的下一个点
}ListNode,*LinkedList;
```

(2)立方体子空间表。该表保存每个子立方体内的所有点，并统计该子立方体内点的个数，其结构如下：

```
typedef struct CubeTable
{
    LinkedList  Link; //子立方体内数据点集表的头指针
    int   number;    //子立方体中所含数据点的个数
}CubeTable;
```

(3)数据点的 k 邻近表。该表保存每个点的 k 个邻近点以及它与邻近点的距离，其结构如下：

```
typedef struct KNNpoint
{
    double  x,, y, z;   // 邻近点的坐标信息
    double   dist;    // 数据点与邻近点之间的距离
    struct KNNpoint *next; //下一个邻近点
}KNNpoint, *KNNList;
```

1.3 k 邻近搜索

利用空间划分后的结果，搜索 k 个最邻近点的主要步骤有：根据每个候选点的坐标值，找出其所在的子立方体的索引号，以此立方体为初始搜索区域；计算该点到立方体六个面的最短距离 d_s；计算该搜索区域点的个数 N_1，如果 N_1 不小于 k，计算该点与搜索区域内其他个点的最大距离 d_{max}，如果 d_{max} 小于 d_s，取前 k 个点为该点的 k 邻近点，搜索完毕，否则以此区域为中心向外扩展一圈作为新的搜索区域重新搜索。k 邻近搜索的详细流程如图 2 所示。

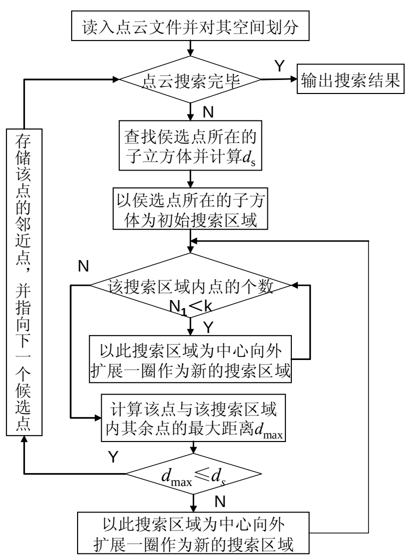

图 2 k 邻近搜索流程图

2 实验结果

为了验证本文算法的正确性和适用性，文中选取了几种典型的点云数据作为实验对象，实验数据均来自实测物体。图3～图5分别为底盒部件、高尔夫球杆头和鞋楦的点云数据，图 6 为高尔夫球杆头点云中任意10点的 k 邻近搜索结果及其中一点的 k 邻近局部放大图，其中尺寸最大的矩形点为选取点，较小的圆点为选取点的 k 邻近点。

实验中，计算子立方体边长式(1)中取 β=0.8。测试时间为搜索所有点的 k 个最邻近点的 CPU 运行时间，单位为 s。

图 3　人头模型

图 4　高尔夫球杆头

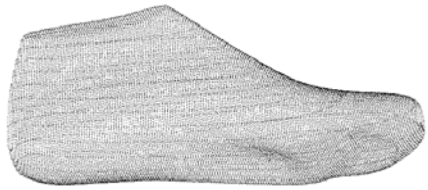

图 5　鞋楦

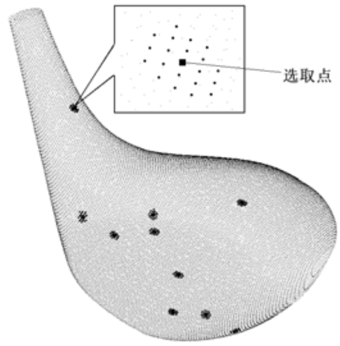

图 6　任意 10 点的 k 邻近搜索结果及局部放大图

本文算法在空间划分之后去除了不含数据点的子立方体，与文献[2]要在测点所在的立方体及其相邻的26子立方体中查找 k 个最邻近的点的搜索方法相比，大大减少了所要搜索的子立方体个数。表 1 记录了本文算法与文献[2]算法在不同 k 值时各点云的 k 邻近搜索时间，从表中可以看出本文算法比文献[2]算法大大节省了 k 邻近搜索时间，从而显著提高了搜索的效率。

表 1　两种搜索算法搜索时间对比

k 值	搜索时间/s					
	人头模型（12754）		高尔夫球杆头（38777）		鞋楦（63981）	
	文中算法	文献[2]算法	文中算法	文献[2]算法	文中算法	文献[2]算法
10	2.514	8.395	5.314	13.495	7.818	19.458
15	3.655	11.028	7.573	20.315	10.421	27.723
20	5.132	15.089	8.683	28.019	13.165	34.055
30	6.092	17.029	11.567	39.563	16.958	47.889
50	15.219	25.869	18.922	94.392	38.311	132.927

注：括号内的数字表示点云数据点的总数。

又由图 7 中所记录的不同 k 值下各点云的 k 邻近搜索时间，可以得出，k 邻近搜索时间随着 k 的增加近似于线性增长，这说明本文算法对 k 值有较强的适应性。

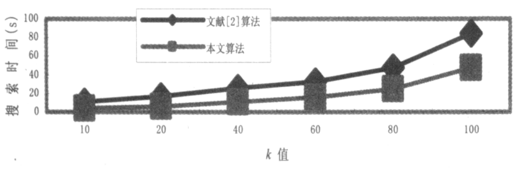

图 7　同一点云不同 k 值下的 k 邻近搜索时间

3 结论

k 邻近搜索是基于散乱数据点集模型重建的关键步骤，本文算法综合考虑了数据的大小、点的总数以及最近点个数 k，采用空间划分策略，排除不包含数据点的子立方体，从而减小了搜索范围，提高了搜索速度。通过对散乱点云数据进行的实验结果表明本文的算法既提高了搜索速度，又保证了准确性。鉴于本文算法是基于空间均匀划分的思想，对于采样密度（下转第 12 页）

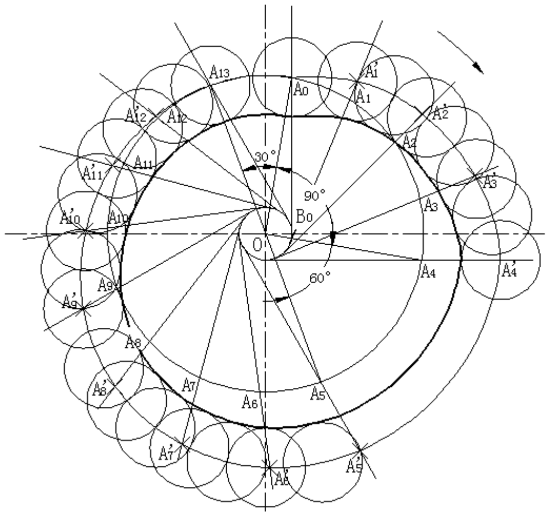

图 5 绘制凸轮实际轮廓曲线

(2)作滚子圆：将“对象捕捉”仅设置为“最近点”，以升程和回程理论轮廓上的点为圆心，以滚子半径为半径画一系列滚子圆。

(3)作滚子圆的包络线：将“对象捕捉”仅设置为“切点”，调用“样条曲线”命令，捕捉各滚子圆的切点即可画出升程和回程的实际轮廓。(如图 5 所示)

说明：看到这里，有的读者可能会说为什么不用偏移命令绘制出升程和回程的实际轮廓呢？不错，笔者也试过该方法，但这样操作的结果会出现一个问题，就是在两个行程的交接处轮廓会出现断开或相交的现象，如本例中升程的起点与近停程的终点处两段轮廓会断开，而在升程的终点与远停程的起点处两段轮廓又会相交，所以用偏移命令作图虽然很快，但结果同样不精确，故笔者不推荐用此方法。

3 小结

运用 AutoCAD 的强大功能能够精确、快速地绘制凸轮从动件位移线图，进而精确地绘制出凸轮轮廓曲线，解决了作图法手工绘制凸轮轮廓曲线误差较大、耗时较多的问题。因此运用 AutoCAD 设计凸轮轮廓曲线，可以大幅提高设计精度和效率。

参考文献

[1] 陈立德.机械设计基础[M].北京：高等教育出版社，2000.

[2] 张建中.机械设计基础[M].北京：高等教育出版社，2007.

[3] 曾令宜.AutoCAD2000 工程绘图教程[M].北京：高等教育出版社，2002.

[4] 应文豹.AutoCAD2006 实训教程[M].成都：电子科技大学出版社，2007.

作者简介：齐真(1963 年-),女,讲师；研究方向：从事机电类专业基础课的教学工作。

（上接第 7 页）

不均匀的点云数据，如果不同数据点密度的区域内的子立方体内的数据点个数差别较大，会对搜索的效率造成一定的影响。因此，作者在接下来的工作中将对此作进一步研究。

参考文献

[1] 熊邦书,何明一,俞华璟.三维散乱数据的 k 个最近邻域快速搜索算法[J].计算机辅助设计与图形学学报,2004,16(7):909－917.

[2] 张丽艳,周儒荣,蔡炜斌等.海量测量数据简化技术研究[J].计算机辅助设计与图形学报,2001，13(11):1019－1023.

[3] 周儒荣,张丽艳,苏旭等.海量散乱点的曲面重建算法研究[J]. 软件学报,2001,12(2):249－255.

[4] 平雪良,徐荣礼,孔俊等.基于空间划分的海量数据 K 邻近新算法[J].华南理工大学学报，2007，35(5)：65－69.

[5] Goodsell G. On finding p-th nearest neighbors of scattered points in two dimensions for small p[J].Computer Aided Geometric Design,2000,17(4):387－392.

[6] PieglL A,Tiller W.Algorithm for finding all k-nearest neighbors[J].Computer-Aided Design,2002,34(2):167－172.

[7] 刘晓东,刘国荣,王颖等.散乱数据点的 k 近邻搜索算法[J].微电子学与计算机,2006,23(4)：23－28.

[8] 严蔚敏,陈文博.数据结构及应用算法教程[M].北京：清华大学出版社，2000.

作者简介：杜小雷(1983 年-)，男，硕士研究生，主要研究方向：逆向工程，计算机辅助设计。