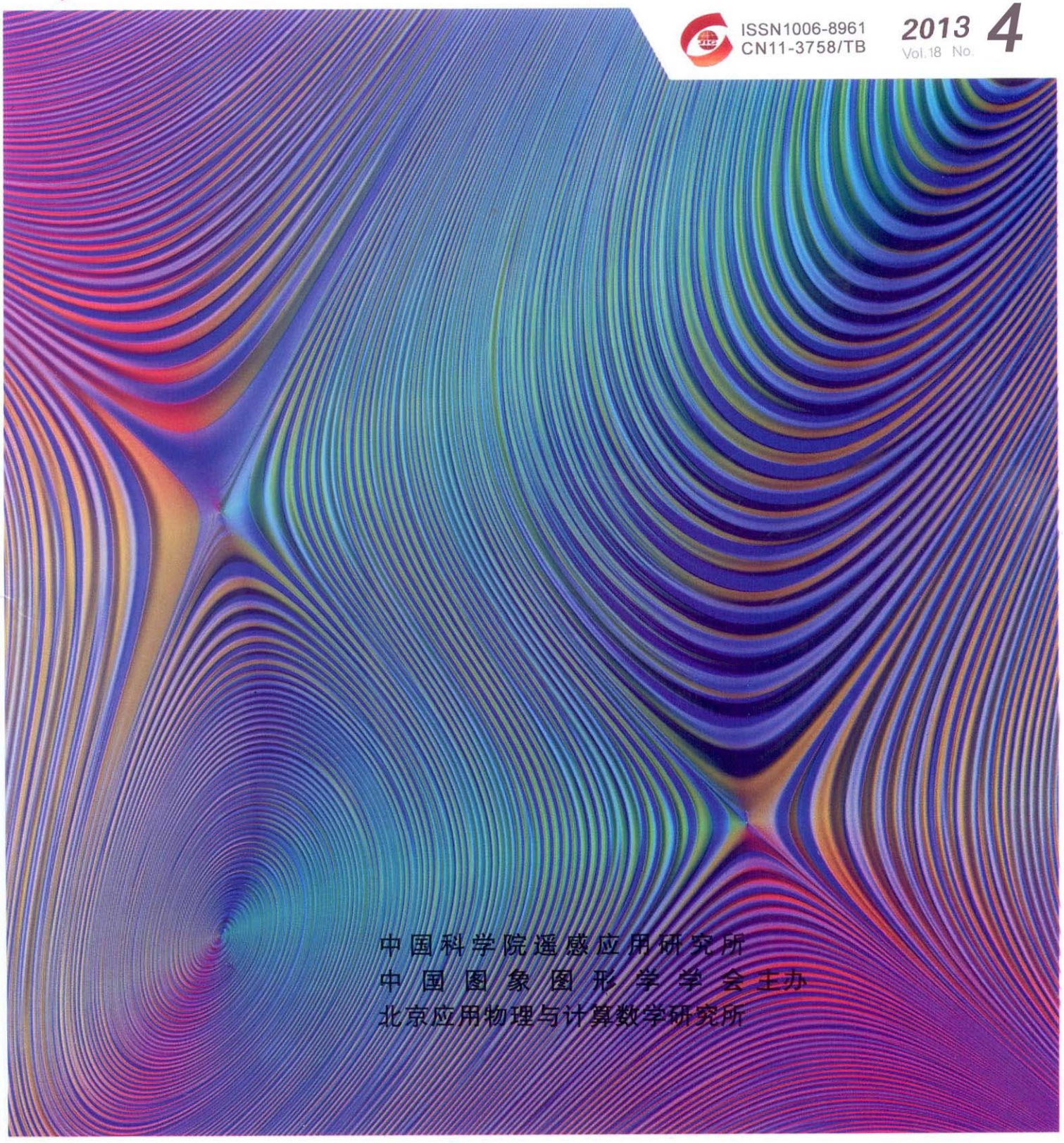
Journal of Image and Graphics

中国图象图形学报

ISSN1006-8961
CN11-3758/TB

2013 Vol.18 No. 4



中国科学院遥感应用研究所
中国图象图形学学会主办
北京应用物理与计算数学研究所

中国精品科技期刊

中国图象图形学报

Zhongguo Tuxiang Tuxing Xuebao

2013 年 4 月　第 18 卷　第 4 期(总第 204 期)

目　次

Journal of Image and Graphics

(**Monthly, Started in 1996**)

Vol. 18 No. 4 April 2013

Contents

第 18 卷 第 4 期
2013 年 4 月

中国图象图形学报
Journal of Image and Graphics

Vol. 18, No. 4
Apr., 2013

中图法分类号: TP391.4 **文献标识码**: A **文章编号**: 1006-8961(2013)04-0399-08

论文引用格式: 杨军,林岩龙,王阳萍,王小鹏. 大规模散乱点的 k 邻域快速搜索算法[J]. 中国图象图形学报,2013,18(4):399-406.

大规模散乱点的 k 邻域快速搜索算法

杨军, 林岩龙, 王阳萍, 王小鹏

兰州交通大学电子与信息工程学院, 兰州 730070

摘 要: 针对大规模散乱点数据 k 最近邻域搜索速度慢和稳定性差的问题,提出一种新的 k 邻域快速搜索算法。首先,引入空间分块策略将数据集中的点归入不同的子空间;其次,动态控制搜索步长的改变量,根据点到其自身小立方体边界的最小距离保证搜索结果的准确性;最后,通过改变预筛选点数量的右侧控制阈值来消除已有算法中由于初始数值不当引起的死循环。实验结果表明该算法对初始搜索步长、搜索步长增量、采样密度和不同的拓扑结构具有较强的稳定性,并且能更快地完成 k 邻域搜索。

关键词: k 最近邻域;曲面重建;点云;搜索步长

Fast algorithm for finding the k-nearest neighbors of a large-scale scattered point cloud

Yang Jun, Lin Yanlong, Wang Yangping, Wang Xiaopeng

School of Electronic & Information Engineering, Lanzhou Jiaotong University, Lanzhou 730070, China

Abstract: To solve the problem of low efficiency and weak stability in searching the k-nearest neighbors of a large-scale scattered point cloud, a fast algorithm for finding k-nearest neighbors is presented. First, the point cloud data is divided into different sub-spaces by using a space block strategy. Second, the variation of the search step length is controlled dynamically. The accuracy of the algorithm is ensured by the minimum distance from the point to the small cube boundary. Finally, the infinite loop problem due to improper initial values in existing algorithms is avoided by altering the right-side threshold, which controls the number of pre-screening points. The experiment results show that the proposed method obtains not only a good stability for the initial searching step, the step increment, and the sampling density at different topology structures, but also a better performance than the existing algorithms.

Key words: k-nearest neighbors; surface reconstruction; point cloud; search step

0 引 言

随着 3 维激光扫描技术的发展,基于散乱点(也称点云)的曲面重建技术成为国内外学者研究的热点。这项技术主要利用逼近理论,克服机械工程等领域传统造型过程比较繁琐,且过程不可逆转性的弊端,避免了资源的浪费,缩短了造型周期,而且还满足了一些复杂物体的高精度建模要求,在逆向工程、计算机图形学、医学等领域具有重要的意义和实用价值。

一般情况下,3 维激光扫描仪获得的 3 维点云数据量比较大,如果直接进行拓扑重建将会遇到难以克服的困难,在实际应用中,一般先搜索 k 邻近点。传统方法是先计算出其余各点到候选点的欧氏距离,然后按升序排序,前面的 k 个点即为该点的 k

收稿日期:2012-06-19;**修回日期**:2012-10-25

基金项目:国家自然科学基金项目(61162016);甘肃省自然科学基金项目(1208RJZA243);陇原青年创新人才扶持计划(201182)

第一作者简介:杨军(1973—),男,副教授,2007 年于西南交通大学获交通信息工程及控制专业博士学位,主要研究方向为计算机图形学、虚拟现实、数字图像处理。E-mail: tom.jun.yang@gmail.com

邻域点。当数据量比较小时,这种方法简单易于实现,然而真实的数据量一般比较大,这种方法的耗时是惊人的。国内外很多学者对此问题提出了一些比较快速的搜索算法。文献[1-2]利用 Voronoi 图进行 k 个最近点搜索,不过在计算 V 集时进行了较多的浮点计算,计算量仍然很大。文献[3-6]利用树的层级结构对点云空间进行 k 邻域搜索,但是这些方法思想比较复杂,不易实现,而且若邻近点和候选点处于不同层时会造成搜索范围扩大,降低搜索效率。周儒荣等人[7]将点云空间分块后进行搜索,但是在每个子空间边界点需要搜索与其最近的 26 个子空间才能确定其邻近点。文献[8]在文献[7]算法的基础上,推广了文献[9]中只能解决 2 维问题的搜索方法,提出了一种新的搜索算法,但是如果获得的点云数据分布不均匀,可能会造成某个空间内的点特别多,而且在其进行 k 邻域搜索时会进行大量的浮点运算,降低了搜索效率。马骊溟等人[10]通过对点云数据坐标点值进行 3 维排序,然后根据坐标点的 3 维序号计算 3 个方向的交集,最后再根据传统方法搜索邻近点,避免了大量的浮点运算,但是其算法本身对搜索步长增量的敏感度比较大,而且如果初始数值选取不当还可能会造成死循环等问题,此外,该算法没有考虑点模型数据分布的局部性,每次搜索整个空间造成时间的浪费。文献[11]中以候选点到所对应立方体小栅格的环六壁的距离为球体半径的取值范围,通过逐步增加球体半径扩大搜索范围来进行 k 邻域搜索,但是该算法不适合 k 值较大的情况,而且在边界处的立方体栅格内进行 k 邻域搜索时,可能会对搜索范围进行不必要的扩大,造成搜索时间的浪费。赵俭辉等人[12]采用二次划分策略,利用多个子空间结构逐步减小搜索范围以提高搜索速度,不过该算法的稳定性不是很理想。

针对大规模点云数据提出一种新的 k 邻域快速搜索算法。首先,利用分块思想将整个点云空间分块,并将点归入各个相应的子空间;其次,在每个子空间中进行 3 维排序并选取符合条件的坐标点,对这些点利用传统方法进行 k 邻域搜索;最后,利用候选点到小立方体边界的距离作为搜索终止条件,保证搜索结果的准确性,并且采用子空间扩张机制和避免死循环机制,进一步提高 k 邻域搜索速度,提高了算法的稳定性。

1 算 法

算法首先将点云空间分块,将数据集中点归入不同的子空间中;然后对每个子空间的坐标进行 3 维排序;最后通过求取 3 个方向的交集进行预筛选;利用传统方法对子空间的点进行 k 邻域搜索,并根据候选点到其他点之间的距离是否小于候选点到自身小立方体边界的最短距离作为候选点 k 邻域是否搜索成功的判断标准。当筛选过程中 3 个坐标方向的边界不在子空间时,利用候选点到子空间外壁(非整个点云空间外壁)的最短距离判断。

1.1 算法描述

1.1.1 空间的初始化

首先计算整个点云空间的最外边界 $[x_{\min}, x_{\max}]$, $[y_{\min}, y_{\max}]$, $[z_{\min}, z_{\max}]$,然后计算每个子空间的边长[8]

$$L = b \times \sqrt[3]{k/n(x_{\max}-x_{\min})(y_{\max}-y_{\min})(z_{\max}-z_{\min})} \tag{1}$$

式中,b 为常数,用来调节子空间边长,称为子空间边长调节因子,n 是点云数据的采样点数目,$x_{\max}$、$x_{\min}$分别为点云空间中在 x 方向的最大值和最小值,$y_{\max}$、$y_{\min}$分别为点云空间中在 y 方向的最大值和最小值,$z_{\max}$、$z_{\min}$分别为点云空间中在 z 方向的最大值和最小值。

边长确定以后,将各采样点归入相应的子空间中,即

$$\begin{aligned} s_x &= \lfloor (p_{ix} - x_{\min})/L \rfloor \\ s_y &= \lfloor (p_{iy} - y_{\min})/L \rfloor \\ s_z &= \lfloor (p_{iz} - z_{\min})/L \rfloor \end{aligned} \tag{2}$$

式中,s_x、s_y、s_z 为子空间的编号,p_{ix},p_{iy},p_{iz}分别为 p_i 在 x,y,z 方向的坐标值。在分块成功以后,利用常用的排序算法对空间内采样点的 3 维坐标值进行排序。由于合并排序算法具有良好的排序效率和稳定性,使用此排序算法对点的坐标序号进行初始化。

1.1.2 小立方体边界的确定

对子空间中任一点 p,其坐标的 3 维排序序号为(u,v,w),在 x,y,z 3 个坐标轴的正反两个方向分别搜索 l 个点,它们的交集满足

$$S(p_i) = \{u-l \leqslant u_i \leqslant u+l, v-l \leqslant v_i \leqslant v+l, w-l \leqslant w_i \leqslant w+l\}$$

在 x、y、z 方向,以对应的坐标序号为 $u-l$、$u+l$、$v-l$、$v+l$、$w-l$、$w+l$ 点的坐标值为边界形成一个小立方体,此立方体称为候选点 p 的小立方体,$u-l$、$u+l$、$v-l$、$v+l$、$w-l$、$w+l$ 称为该立方体的边界序号。其中 l 为坐标序号搜索步长。

1.2　k 邻域快速搜索算法

将子空间内点云数据在 x、y、z 方向排序,对任意候选点求得交集 $\boldsymbol{S}$ 和其小立方体,如果小立方体的边界序号小于子空间内 3 维坐标序号的最大值且大于最小值,则计算候选点到自身小立方体边界的最小距离 d_short。如果候选点到其本身 k 个邻近点的欧氏距离小于 d_short,则认为候选点 k 邻域搜索成功,$flag$ = true,否则 $flag$ = false。$flag$ 为点的标识符,标记点的 k 邻域是否搜索成功。如果某一候选点搜索失败且其小立方体的边界序号位于子空间内 3 维坐标序号的最小值和最大值之间,则扩大坐标序号搜索步长继续搜索。如果候选点的小立方体边界序号小于等于子空间 3 维坐标排序序号的最小值且大于等于最大值,则计算 d_short 等于其候选点到此子空间外壁(非整个点云空间外壁)的最小距离。如果候选点到自身 k 个邻近点的欧氏距离小于 d_short,则认为搜索成功,$flag$ = true,否则$flag$ = false。如果子空间内所有点 k 邻域搜索都成功,则认为这个子空间查找完成,否则子空间向外扩充,按上述方法继续对原子空间中 k 邻域查找失败的点进行 k 邻域搜索,直至子空间全部点查找成功,再进入下一个子空间搜索,最终至整个空间内点的 k 邻域查找成功。

点云空间进行 k 邻域快速搜索算法的具体步骤如下:

1)按式(1)(2)对整个点云空间分块,并将采样点归入相应的子空间中。

2)若子空间候选点个数不为 0,则对某个子空间内任意候选点计算其交集 $\boldsymbol{S}$ 和交集内的点数 K。

3)如果 $K \geqslant \alpha k$,且 $K \leqslant \beta k$,α、β 为预筛选点数目的左、右控制阈值,则转入步骤 5),否则继续。

4)如果 $K \leqslant \alpha k$,则以 $l = l + m_1\Delta l$ 重新搜索交集 $\boldsymbol{S}$ 和计算点数 K;如果 $K \geqslant \beta k$,则以 $l = l - m_2\Delta l$ 重新搜索交集 $\boldsymbol{S}$ 和计算点数 K。如果 $K \geqslant \beta k$ 的循环次数超过其设定的上限,则认为程序进入死循环状态,β 值加 1。若 K 符合条件,则转入步骤 5),否则转到步骤 2)继续执行。其中,m_1、m_2 分别表示各自的循环次数,Δl 为搜索步长增量,l 为初始搜索步长。

5)根据此时候选点小立方体的边界序号大小计算相对应的 d_short。

6)计算集合 $\boldsymbol{S}$ 中 K 个点与候选点的欧氏距离并按升序排序,如果前 k 个值小于 d_short,则认为候选点的 k 邻域搜索成功,$flag$ = true,否则 $flag$ = false;若候选点搜索失败且其小立方体的边界序号位于子空间 3 维排序序号的最大值和最小值之间,则 $l = rl$,重新在此子空间内搜索,直至其小立方体的边界序号超出子空间的范围。r 为一实数,这里取 $r = 1.5$。

7)判断这个子空间内是否有查找失败的点。如果有,则子空间向外扩充,按步骤 2)—6)对查找失败的点重新查找;若子空间内所有点的 k 邻域搜索成功,则进入下一个子空间,否则子空间继续扩大,直至这个子空间内查找失败点的 k 邻域查找成功。

8)对整个点云空间内所有子空间进行 k 邻域搜索。

9)结束。

算法的流程图如图 1 所示。

2　实　验

在实验中,点云数据均为真实物体的扫描数据,如图 2 所示。所有实验均在酷睿双核 CPU,主频 2.20 GHz,内存 2 G 的个人笔记本电脑上完成,测量时间为 1 个点的 k 邻域搜索的平均 CPU 时间,单位为 ms。

实验 1　不同初始搜索步长 l 和步长增量 Δl 的对比研究,实验结果如表 1 所示。实验模型为图 2(e)(f),其中在计算时间开销时本文算法选取 b = 0.6(可能不是最佳的)。由表 1 中数据可以看出,本文算法对同一模型在初始步长 l 为 10 和 100 时,对于相同的 k 值,步长增量 Δl 分别为 10、100、1 000 时,其算法的搜索时间效率差距不是很大,表明本文算法对初始搜索步长 l 和步长增量 Δl 的敏感度不高,这主要是由于本文算法将点云按坐标值分布进行分块,减小了搜索范围,对于实验中出现 Δl 增加而算法效率降低的现象,原因可能是由于搜索步长太大,造成子空间内频繁地搜索有效点或者子空间进行不必要的扩张。反之,文献[10]中算法对同一模型在不同的 l、相同的 Δl 和 k 时其搜索时间效率相差不大,但仍然高于本文算法;而在相同的 l 和 k、

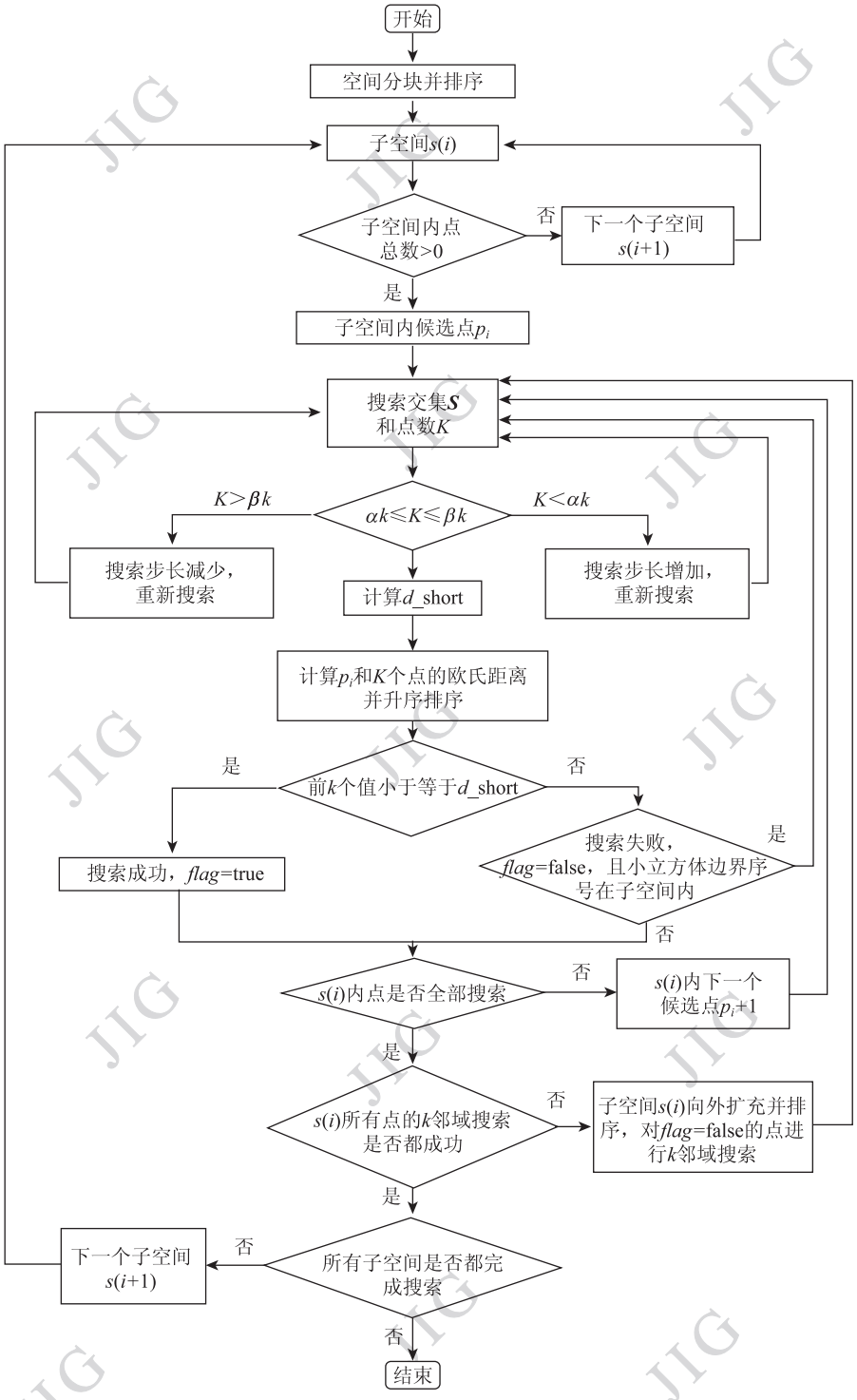
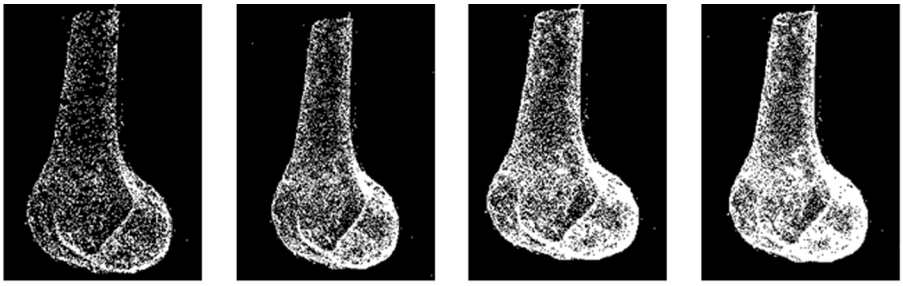


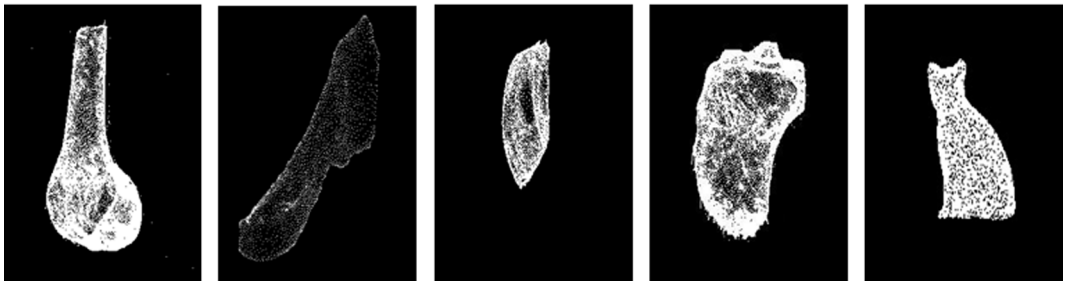
图 1 算法流程图

Fig. 1 Flow diagram of the algorithm

不同的 Δl 下其搜索效率相差较大，并且随着数据规模和 k 的增大，其时间效率差距更加明显。另外，在实验过程中，文献[10]的算法如果 Δl 选取不恰当，则其搜索耗时可能很大，不能满足需求，更糟糕的情况是如果预筛选点数目的左、右阈值设置不当会使程序进入死循环，无法得到结果。实验结果表明本文算法对 l 和 Δl 的稳定性较强，并且不论初始值如何设置都能得到结果；而文献[10]中算法对初始搜索步长 l 敏感度略低，而对步长增量 Δl 的敏感度较高，而且有可能进入死循环状态。



(a) 股骨20%重采样模型　(b) 股骨40%重采样模型　(c) 股骨60%重采样模型　(d) 股骨80%重采样模型



(e) 股骨原始模型　(f) 膝关节韧带模型　(g) 膝关节髌骨模型　(h) 膝关节胫骨模型　(i) cat模型

图 2　点云模型

Fig. 2　Point cloud model

表 1　不同初始搜索步长 l 和步长增量 Δl 的搜索时间对比

Table 1　Comparison of search performance under different initial searching steps l and step increments Δl

/ms

模型	算法	k	$l=10$			$l=100$		
			$\Delta l=10$	$\Delta l=100$	$\Delta l=1\,000$	$\Delta l=10$	$\Delta l=100$	$\Delta l=1\,000$
图 2(f) ($n=3\,233$)	本文	10	0.159	0.183	0.207	0.150	0.164	0.309
		50	0.309	0.323	0.357	0.284	0.304	0.333
	文献[10]	10	1.648	0.358	0.739	1.020	0.295	0.826
		50	2.996	0.554	0.725	2.373	0.483	0.812
图 2(e) ($n=32\,438$)	本文	10	0.308	0.237	0.339	0.282	0.206	0.219
		50	0.537	0.462	0.549	0.481	0.431	0.443
	文献[10]	10	61.492	8.387	1.667	62.972	8.027	1.650
		50	145.735	14.354	2.385	138.129	14.152	2.368

通过上述实验,选取初始搜索步长 $l=100$,步长增量 $\Delta l=100$ 进行实验 2,讨论左右控制阈值 α 和 β 对算法效率的影响。

实验 2　不同控制阈值的搜索效率对比,实验结果如表 2 所示,实验模型为图 2(e)(f)。由表 2 数据可知,对于同一模型,k 和 β 相同、α 不同时,搜索时间相差比较大,而且随着 α 的增大,搜索时间增加,且其差距随着 k 的增大愈加明显,这是由于随着 α 和 k 的增加,左侧的控制值 αk 会翻倍增加,致使在 k 邻域搜索过程中为了达到左侧控制值的要求而频繁地增加步长来搜索,还可能造成原子空间不能满足要求而对子空间进行扩充,这样在一定程度上造成了搜索效率的降低;当 k 和 α 相同、β 不同时,时间效率相差较小。随着模型规模的增加,搜索效率的变化不是太大。这说明本文算法对左侧控制阈值 α 比较敏感,对右侧控制阈值 β 敏感度相对比较低,同时对模型具有一定的稳定性。

通过实验 1、实验 2,本文算法均取 $l=100$,$\Delta l=100$,$\alpha=2$,$\beta=20$ 进行以下实验:

表 2 不同控制阈值的搜索效率对比

Table 2 Comparison of search performance under different control threshold values

/ms

模型	k	$\beta=10$		$\beta=20$		$\beta=30$	
		$\alpha=2$	$\alpha=7$	$\alpha=2$	$\alpha=7$	$\alpha=2$	$\alpha=7$
图 2(f)	10	0.169	0.178	0.164	0.183	0.165	0.198
($n=3\,233$)	50	0.294	0.483	0.294	0.468	0.304	0.488
图 2(e)	10	0.208	0.267	0.204	0.263	0.202	0.263
($n=32\,438$)	50	0.429	0.541	0.430	0.540	0.430	0.540

实验 3 同一模型在不同采样密度下对本文算法的影响,实验结果如表 3 所示。实验模型为图 2(a)—(d)。从表 3 中可以看出,对同一模型在不同采样密度下的 k 邻域搜索都能通过设置合适的 b 值达到或者接近最佳搜索速度,而且 b 的分布比较集中,一般 k 值较小时(如 $k=10$),b 值集中在 2.4 ~ 2.8,而 k 值较大时(如 $k=50$ 或 100),b 值集中在 0.2 ~ 0.6。另外,随着密度的增大,采用相同的 k、不同的 b 时,搜索时间的变化也会随之增加,但是在 k 较大时变化相对较小。因此,对于分布密集,点规模较大的模型选取合适的 b 值很重要,但是由于本文算法中 b 的分布比较集中,根据实验很容易得到适用于模型的 b 值。

表 3 同一模型的不同采样密度对本文算法的影响

Table 3 Impact of different sampling densities of the same model on the proposed algorithm

/ms

模型	k	b											
		0.2	0.4	0.6	0.8	1.0	2	2.2	2.4	2.6	2.8	3.0	4.0
图 2(a) ($n=6\,487$)	10	0.301	0.209	0.185	0.199	0.207	0.199	0.209	0.190	0.185	0.175	0.178	0.219
	50	0.390	0.332	0.373	0.440	0.501	0.602	0.683	0.633	0.647	0.664	0.691	0.948
	100	0.534	0.481	0.623	0.799	0.975	1.245	1.262	1.276	1.336	1.413	1.411	1.278
图 2(b) ($n=12\,975$)	10	0.545	0.286	0.235	0.222	0.216	0.193	0.189	0.186	0.191	0.208	0.208	0.273
	50	0.497	0.353	0.420	0.487	0.499	0.580	0.588	0.687	0.671	0.621	0.632	0.977
	100	0.599	0.574	0.718	0.899	1.058	1.252	1.052	1.062	1.147	1.286	1.460	2.797
图 2(c) ($n=19\,462$)	10	0.828	0.567	0.286	0.246	0.228	0.191	0.194	0.212	0.183	0.195	0.219	0.268
	50	0.636	0.415	0.413	0.468	0.477	0.599	0.618	0.639	0.665	0.654	0.845	0.944
	100	0.675	0.613	0.737	0.906	0.972	1.105	1.320	1.171	1.140	1.174	1.271	2.428
图 2(d) ($n=25\,950$)	10	1.058	0.451	0.313	0.270	0.237	0.207	0.210	0.208	0.229	0.314	0.225	0.95
	50	0.729	0.430	0.427	0.457	0.495	0.592	0.641	0.647	0.672	0.730	0.855	0.912
	100	0.761	0.669	0.774	0.931	0.966	1.104	1.161	1.437	1.411	1.327	1.334	2.383

注:表中下划线表示 b 值对于不同 k 的最佳搜索效率。

实验 4 本文算法对不同拓扑结构点云模型的对比研究,实验结果如表 4 所示。实验采用的点云模型为图 2(e)—(g)。在实验中,对不同拓扑结构,不同的 k 值最佳或接近最佳搜索效率的 b 值分布范围和实验 2 的分析结果接近,并且从实验结果可以看出,对不同模型的搜索时间在 b 的取值范围内相差不大,说明算法对不同的点云模型具有较强的稳定性。

实验 5 不同算法对同一点云模型的搜索速率的对比研究,实验结果如表 5 所示。实验采用的点云数据为图 2(e)—(i)。通过表 5 搜索时间的比较,对同一规模较小的模型(如人体 ACL 模型),当 k 值较小时本文算法快于文献[10]算法,但是随着 k 的增加本文算法时间消耗的增加速度会快于文献[10]算法,造成这种结果的原因是本文算法不仅要对各个子空间初始化,还会在搜索过程中改变子空

间大小并重新搜索,而文献[10]算法只是对点云坐标进行一次性排序。当点云规模较小时,排序的时间消耗不是太大,使其效率高于本文算法。但是,随着数据规模的增大,本文算法明显快于文献[10]算法,这是因为本文算法将分块思想引入,对 3 维坐标的排序和搜索只须在局部子空间内完成,另外,对搜索步长的改变量随着其循环次数动态改变,且对步长的增加或者减少采取分别控制,采用增加控制阈值机制避免死循环,而且利用候选点到自身小立方体边界的最小距离来有效地终止搜索。同时,本文算法在搜索效率上也快于文献[8]和文献[12]算法,这是因为本文算法虽然在前期处理中增加了排序操作过程,但对数据点进行了预筛选,减少了计算欧氏距离时的浮点运算,而且本文算法降低了由于分块不均匀对算法效率的影响。随着点云规模和 k 值的增大,本文算法在搜索速率上的优势相对更加明显。

表 4　对不同拓扑结构点云模型的搜索效率对比

Table 4　Comparison of search performance of different models

/ms

模型	k	b											
		0.2	0.4	0.6	0.8	1.0	2	2.2	2.4	2.6	2.8	3.0	4.0
图 2(f)(n=3 233)	10	0.231	0.231	0.202	0.202	0.202	0.178	0.164	0.183	0.197	0.207	0.236	0.260
	50	0.294	0.304	0.352	0.396	0.434	0.444	0.507	0.507	0.488	0.492	0.502	0.521
	100	0.531	0.589	0.710	0.754	0.811	0.733	0.782	0.787	0.802	0.787	0.725	0.870
图 2(g)(n=7 486)	10	0.312	0.227	0.198	0.185	0.191	0.187	0.198	0.179	0.175	0.179	0.176	0.191
	50	0.346	0.296	0.271	0.294	0.323	0.404	0.444	0.465	0.475	0.494	0.530	0.540
	100	0.598	0.480	0.530	0.569	0.644	0.878	0.901	0.914	0.949	0.859	0.870	1.008
图 2(e)(n=32 438)	10	1.252	0.519	0.345	0.288	0.258	0.215	0.204	0.214	0.225	0.251	0.238	0.342
	50	0.797	0.438	0.430	0.455	0.468	0.565	0.684	0.705	0.683	0.727	0.807	0.996
	100	0.828	0.671	0.742	0.894	0.935	1.162	1.168	1.275	1.633	1.555	1.413	2.119

表 5　同一模型在不同算法之间的搜索时间对比

Table 5　Comparison of search performance of different algorithms on the same model

/ms

模型	k	本文算法	文献[10]算法	文献[8]算法	文献[12]算法 α=0.25
图 2(f)(n=3 233)	10	0.164	0.265	0.280	1.667
	50	0.294	0.348	0.318	1.996
	100	0.531	0.341	0.613	2.696
图 2(g)(n=7 468)	10	0.175	0.467	0.244	3.602
	50	0.271	0.549	0.384	3.839
	100	0.480	0.547	0.632	4.387
图 2(h)(n=10 000)	10	0.324	0.835	0.675	4.815
	50	0.587	1.214	0.667	5.039
	100	0.636	1.418	0.679	5.77
图 2(i)(n=29 284)	10	0.220	1.623	0.504	14,443
	50	0.448	2.413	0.907	15.663
	100	0.802	2.612	1.254	17.724
图 2(e)(n=32 438)	10	0.204	1.552	0.496	15.486
	50	0.430	2.394	0.881	16.244
	100	0.671	3.097	1.457	17.438

3　结　论

本文算法引入空间分块思想,根据循环次数对搜索步长的增加量或减少量采取分别控制的策略使其动态改变,并根据循环次数判断是否进入死循环和通过增加右侧控制阈值来解除死循环状态,采用到相应小立方体边界的最小距离作为判断 k 邻域搜索终止的条件,保证结果的准确性。实验结果表明本文算法不仅对初始搜索步长、步长增量、采样密度、不同拓扑结构有较强的稳定性,而且和已有算法相比在搜索速度上有明显的优势,有较高理论参考价值和实用性。此外,算法中子空间边长调节因子 b 的分布具有一定的规律性。本文算法也存在一定的不足,如算法对左侧控制阈值比较敏感 ,存在分块不均匀的现象等,这也是本研究在未来要进一步解决的两个问题。

参考文献(References)

[1]　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.

[2] Dickerson M T, Drysdale R L S, Sack J R. Simple algorithms for enumerating interpoint distance and finding k nearest neighbors [J]. International Journal of Computational Geometry and Applications, 1992, 2(3): 221-239.

[3] Sarkar M, Leong T Y. Application of k-nearest neighbors algorithm on breast cancer diagnosis problem[C]//The 2000 AMIA Annual Symposium. Los Angeles: IEEE Computer Society Press, 2000: 759-763.

[4] Connor M, Kumar P. Fast construction of k-nearest neighbor graphs for point clouds[J]. IEEE Transactions on Visualization & Computer Graphics, 2010, 16(4): 599-608.

[5] Sankaranarayanan J, Samet H, Varshney A. A fast all nearest neighbor algorithm for applications involving large point-clouds [J]. Computers & Graph, 2007, 31(2): 157-174.

[6] Procopiuc O, Agarwal P K, Arge L, et al. Bkd-tree: a dynamic scalable kd-tree[C]//International Symposium on Spatial and Temporal Databases. Berlin: Springer Press, 2003: 46-65.

[7] Zhou R R, Zhang L Y, Su X, et al. Algorithmic research on surface reconstruction from dense scattered points[J]. Journal of Software, 2001, 12(2): 249-255. [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.]

[8] Xiong B S, He M Y, Yu H J. Algorithm for finding k-nearest neighbors of scattered points in three dimensions[J]. Journal of Computer-Aided Design & Computer Graphics, 2004, 16(7): 909-912. [熊邦书,何明一,余华璟. 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): 909-912.]

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

[10] Ma L M, Xu Y, Li Z X. Fast k-nearest neighbors searching algorithm for scattered points based on dynamic grid decomposition [J]. Computer Engineering, 2008, 34(8): 10-11. [马骊溟,徐 毅,李泽湘. 基于动态网格划分的散乱点 k 邻域快速搜索算法[J]. 计算机工程, 2008, 34(8): 10-11.]

[11] Liu Y H, Liao W H, Liu H. Research of k-nearest neighbors search algorithm in reverse engineering[J]. Machinery Design & Manufacture, 2012, 1(3): 256-258. [刘越华,廖文和,刘浩. 逆向工程中散乱点云的 k 邻域搜索算法研究[J]. 机械设计与制造,2012, 1(3): 256-258.]

[12] Zhao J H, Long C J, Ding Y H, et al. A fast algorithm for finding k-nearest neighbors based on small cube grids[J]. Geomatics and Information Science of Wuhan University, 2009, 34(5): 615-618. [赵俭辉,龙成江,丁乙华,等. 一种基于立方体小栅格的 k 邻域快速搜索算法[J]. 武汉大学学报:信息科学版,2009, 34(5): 615-618.]