Pattern Recognition Letters 49 (2014) 162–170

Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

A new extracting algorithm of *k* nearest neighbors searching for point clouds ☆

Zisheng Li [a,b], Guofu Ding [a,*], Rong Li [a], Shengfeng Qin [c]

[a] Institute of Advanced Design & Manufacturing, School of Mechanical Engineering, Southwest Jiaotong University, Chengdu 610031, China
[b] School of Manufacturing Science and Engineering, Southwest University of Science and Technology, Mianyang 621010, China
[c] Department of Design, Northumbria University, City Campus East Building 2, Newcastle upon Tyne NE1 2 SW, UK

ARTICLE INFO

Article history:
Received 5 October 2013
Available online 22 July 2014

Keywords:
kNN searching algorithm
Extracting algorithm
Distance comparison using vector inner product
Point clouds

ABSTRACT

k Nearest neighbors (*kNN*) searching algorithm is widely used for finding *k* nearest neighbors for each point in a point cloud model for noise removal and surface curvature computation. When the number of points and their density in a point cloud model increase significantly, the efficiency of a *kNN* searching algorithm becomes critical to various applications, thus, a better *kNN* approach is needed. In order to improve the efficiency of a *kNN* searching algorithm, in this paper, a new strategy and the corresponding algorithm are developed for reducing the amount of target points in a given data set by extracting nearest neighbors before the search begins. The nearest neighbors of a reverse nearest neighborhood are proposed to use in extracting nearest points of a query point, avoiding repetitive Euclidean distance calculation in an extracting process for saving time and memories. For any point in the model, its initial nearest neighbors can be extracted from its reverse neighborhood using an inner product of two related vectors other than direct Euclidean distance calculations and comparisons. The initial neighbors can be its full or partial set of the all nearest neighbors. If it is a partial set, the rest can be obtained by using other fast searching algorithms, which can be integrated with the proposed approach. Experimental results show that integrating extracting algorithm proposed in this paper with other excellent algorithms provides a better performance by comparing to their performances alone.

1. Introduction

A variety of *kNN* searching algorithms are widely used in point cloud modeling [34], spatial database retrieval [10,12] and data mining [27], etc. A *kNN* searching problem can be described as: given an existing data set S with n points, for a query point $p_0 \in S$, finding a subset S' with k points (p_0 is not included) where $S' \subset S$ and $k < n$ such that for any point $p_1 \in S'$ and $p_2 \in S - S'$, $dist(p_0, p_1) \leqslant dist(p_0, p_2)$, here $dist(p_i, p_j)$ representing the distance between p_i and p_j.

Distance metric can have different forms depending on its applications. In point cloud modeling, Euclidean distance is usually used in *kNN* search for estimating the geometric properties such as the normal and curvature of a point. In order to search the nearest neighbors of a query point, a *kNN* searching algorithm needs to (1) calculate distances between the query point and any others in the data set, (2) sort these points by the distance in ascending order, (3) choose the most closest *k* points. When the query point is changed, the above procedure will be repeated, thus all distances computed before will be used only once and need to be updated every time. This is also called brute force approach [26].

Many scholars have studied various *kNN* algorithms in R^d ($d \geqslant 2$) space for extensive applications and proposed a few efficient searching algorithms. Those algorithms fall into four categories broadly: multi-step progressive algorithm, parallel algorithm, data reorganization algorithm (*DRA*) and spatial partition algorithm (*SPA*). The prominent searching algorithms are *DRA* and *SPA*.

DRA involves a tree-like data structure, divides a whole data set into multi-level subspaces which are applied to build tree nodes depending on splitting rules recursively. The data structure used in these algorithms is a binary or multi children tree whose nodes differ from each other due to splitting rules; therefore splitting rules determine searching efficiency among these algorithms. For example, *KD – tree* [6] uses tree nodes to store space range to partition data space into clipped super planes for reducing searching

☆ This paper has been recommended for acceptance by D. Coeurjolly.

* Corresponding author. Tel./fax: +86 028 8760 1643.
E-mail address: dingguofu@163.com (G. Ding).

http://dx.doi.org/10.1016/j.patrec.2014.07.003
0167-8655/

scope. *Cell − tree* [9] refines the distance bound between cells, and searches nearest neighbors by partitioning the data space into equal-size cubic cells on which cell-tree is built. *VP − tree* [32] builds a binary tree by dividing the data space using large spherical with distance from the selected vantage point instead of cubic cells employed in *Cell − tree*. *BBD − tree* [4] builds a tree differing from *KD − tree* and *Cell − tree* in its tree nodes involving not only points but also the set theoretic difference. *PAT* [11, 21] builds an efficient search tree by using principal component analysis (*PCA*) and conducts its search by using partial distance search (*PDS*) while *OST* [19] builds a tree using orthogonal base vectors and elimination inequality rules. *Quad − tree* for disk accessing and *R − tree* for searching [26] with locality are efficient as well as *C − tree* [33]. Algorithms in this category partition space into small regions to build a tree, each node having nearly the same number (N_c) of points, and use bounding comparison approaches to remove child nodes that cannot be included in candidate set in the searching process. We have tested that N_c is a key parameter for searching nearest neighbors. If N_c is small, this algorithm can degenerate into a brute force approach, while if it is large, its answer set can be disordered seriously, However, there is no method to calculate the proper N_c yet.

SPA [20,23,24,30,31,35] divides the bounding box of a data set into cells, its splitting procedures are similar to that of *DRA*, but it does not reorganize points into a tree and just records which cell contains which points. The algorithm proposed in [23] has been applied to two-dimensional data sets while those in [20,24,30,31,35] are for various three-dimensional point clouds. When a search begins, *SPA* first locates the cell with a query point in and then calculates distances of every point in that cell to the query point and sorts them in increasing order. If there are enough points in this cell, and if the *k*th shortest distance is smaller than the distance between the query point and the closet wall of the cell, the search stops. Otherwise, it continues with one or more cells depending on the expanding rules to repeat the search procedure. *SPA* is an excellent searching algorithm with satisfactory accuracy and acceptable speed because it utilizes neighborhoods of a point to speed up a search process by splitting a whole data space into cells and reducing the searching scope in turn. However, distances between a query point and any other points still need to be calculated again every time, thus, it can still be regarded as a brute force algorithm in this sense.

Neighbor finding technology performs wasteful repeated work [22] as points in proximity share neighbors [26]. To avoid using a brute force method in neighborhood, calls for novel methods to extract nearest neighbors directly. Lazy search algorithm proposed by Song [28] can extract partial nearest neighbors from the latest query point for a current query point. But it is used for a moving point. There is a key difference between the *kNN* searching problem for a point cloud and that of moving point searching problem. For the latter, the query point is moving and it is not an element of the data set. We have tested that the criterion proposed in [28] leads to serious inaccuracies when it is applied to extract nearest neighbors for point cloud models.

The motivation of this paper is to reduce the number of target searching points differing from the prominent algorithms which aim to reduce searching scope. In our proposed approach, in order to find *kNN* for a query point *P*, if having extracted k_1 points from a reverse nearest neighborhood [8] of *P*, we only need to search $k - k_1$ nearest neighbors for *P* further in the subsequent searching process. This new algorithm extracts nearest neighbors (*EkNN*) directly rather than applying a brute force method in neighborhood. In addition, inner products of related vectors are used to sort out the nearest neighbors avoiding the use of direct distance comparison and saving both time and memories. The technical contribution of our work can be summarized as follows:

- An accurate criterion for extracting nearest neighbors from the reverse neighborhood of a point in query is proposed, although all nearest neighbors of a query point can be extracted from its reverse neighborhood recursively, we only do it once through each neighbor for better performance.
- An alternative method for comparing Euclidean distances is presented. This approach utilizes the inner product of two vectors among the query point and two points in checking to sort out their orders.
- Finally, the proposed method can be integrated with any other searching algorithms. We tested our method with *SPA* and *DRA*, and used a linked list to manage and save memories.

The rest of the paper is organized as follows. Section 2 defines the related concepts of *kNN* searching and our new approach. Section 3 gives the details of our novel algorithm for extracting nearest neighbors. Section 4 presents the results of experiments and the conclusions are finally drawn in Section 5.

2. A new approach for *kNN* searching

Surface reconstruction has been a central problem in reverse engineering [3,5,15]. Technological advances in laser scanning enable the creation of large 3D point cloud data sets with high density and accuracy and also present a real application challenge in generating product models through reverse engineering approaches accurately and rapidly. There is a wide diversity of reverse engineering methods for surface modeling from point clouds [29].The analytical functions of point clouds are unknown, so all geometric properties such as normals can only be estimated with a variety of methods such as regression [15], delaunay [2], statistic [16], one-ring [13] and hough transformation [7]. For instance, estimating normals needs to construct the best local tangent plane for each point and *kNN* of each point must have been searched before constructing the tangent plane. Thus, *kNN* searching plays an important role in point clouds application, and reverse engineering in turn.

Although there are a lot of *kNN* searching algorithms as stated in Section 1, they are not all for point clouds applications such as reverse engineering from a big data set with high density, while classic research in reverse engineering is much focused on surface reconstruction, smoothing and etc., paying less attention to *kNN* searching problems. Nevertheless, *kNN* searching algorithm is vital to the performance of reverse engineering on a large scale point cloud modeling with high density and accuracy, it needs further studies.

In this paper, *n* denotes the number of points in a point cloud. *P* is called a query point if we are going to search *kNN* points for *P*, and $k_{NN}(P)$ is the data set that consists of *k* nearest points to *P*. If $P' \in k_{NN}(P)$, then *P* is called a reverse nearest neighbor of P'. Reverse nearest neighbors is abbreviated to *rkNN*. All *rkNN* points of P' forms a set $rk_{NN}(P')$. Euclidean distance between P_i and P_j denoted by $dist(P_i, P_j)$.

If $P' \in k_{NN}(P)$ is the next query point, for any $Q \in k_{NN}(P)$ $(Q \neq P)$, there are two cases that $Q \in k_{NN}(P')$ or $Q \notin k_{NN}(P')$. If we know that $Q \in k_{NN}(P')$ before a search begins, then only $k - 1$ points are needed to be found in the subsequent search process(the computation time in the searching process reduces when *k* decreases): (1) In order to judge whether $Q \in k_{NN}(P')$ or not, we proposed a fast extracting algorithm. By using this extracting algorithm, if $Q \in k_{NN}(P')$, we can determine the truth before the search begins. (2) Direct distance computation and comparison will be replaced with the inner product of two vectors formed from a query point and other two points in indirectly distance comparison for improving its efficiency further (the proof is given in A). The latter method