Effective Nearest Neighbor Search for Aligning and Merging Range Images

Ryusuke Sagawa† Tomohito Masuda‡ Katsushi Ikeuchi‡
† Institute of Scientific and Industrial Research, Osaka University
8-1 Mihogaoka, Ibaraki-shi, Osaka, 567-0047, JAPAN
sagawa@am.sanken.osaka-u.ac.jp
‡ Institute of Industrial Science, University of Tokyo
4-6-1 Komaba Meguro-ku, Tokyo, 153-8505, JAPAN
{tom,ki}@cvl.iis.u-tokyo.ac.jp

Abstract

This paper describes a novel method which extends the search algorithm of a k-d tree for aligning and merging range images. If the nearest neighbor point is far from a query, many of the leaf nodes must be examined during the search, which actually will not finish in logarithmic time. However, such a distant point is not as important as the nearest neighbor in many applications, such as aligning and merging range images; the reason for this is either because it is not consequently used or because its weight becomes very small. Thus, in this paper, we propose a new algorithm that does not search strictly by pruning branches if the nearest neighbor point lies beyond a certain threshold. We call the technique the Bounds-Overlap-Threshold (BOT) test. The BOT test can be applied without re-creating the k-d tree if the threshold value changes. Then, we describe how we applied our new method to three applications in order to analyze its performance. Finally, we discuss the method's effectiveness.

1 Introduction

The nearest neighbor problem in multidimensional space is a major issue in many applications. Many methods have been developed to search for the nearest neighbor of a query. A simple exhaustive search computes the distance from a query to every point. Its computational cost is $O(n)$. This approach is clearly inefficient. Hashing and indexing[21, 3] search in constant time, these methods require a large space in which to store the index table. Some hierarchical structures have been proposed to access multidimensional data, such as k-d tree[1, 6], quadtree[18], k-d-B tree[17], hB-tree[11] and R-tree[9]. These trees differ in structure, but their searching algorithms are similar. For details, refer to [7].

The k-d tree[1, 6] is one of the most widely used structures for searching for nearest neighbors. It is a kind of binary tree that partitions space by using hyperplanes that are perpendicular to the coordinate axes. If a k-d tree consists of n records, that k-d tree requires $O(n \log_2 n)$ operations to construct and $O(\log_2 n)$ to search. In this paper, we analyze the search algorithm by using the k-d tree and propose a novel method of pruning branches to reduce the computational cost for aligning and merging range images.

A case in which a search finishes in $O(\log_2 n)$ is an ideal case, in that only a leaf node is examined, and the nearest neighbor belongs to it. However, a search using a k-d tree does not actually finish in logarithmic time in many cases because a search often needs to examine several leaf nodes before finding the nearest neighbor point. In the worst case scenario, all leaf nodes must be examined. When the nearest neighbor is far from a query, the number of leaf nodes is apt to increase (see Section 2). Therefore, we introduced a novel test that takes place during the traversing of the k-d tree. This test compares the distance from a query to the nearest neighbor with a threshold defined by the user (See Section 3).

Since the threshold depends on the application, we tested our new method in aligning range images and merging range images. These applications were three-dimensional cases, in which 3-D models were compared. For aligning range images, several methods have previously been proposed to speed up the search of a k-d tree by caching the closest points [19, 8]; These methods are applied to aligning range images and use the correspondences in the previous iteration as the initial estimate. They assume that the correspondence does not change drastically; however, we do not make such an assumption. Thus, our method can be applied not only to aligning range images but also to merging range images. In higher dimensional case, Nene and Nayar[15] proposed a method to find the nearest neighbor point within a distance ϵ in high-dimensional space; however, this method cannot find any point outside the distance

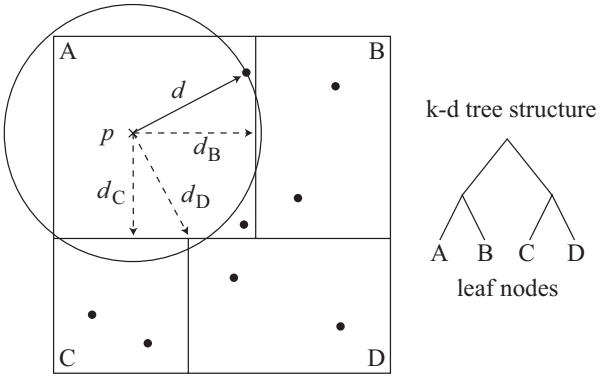

Figure 1. A 2-D example of a k-d tree

ϵ, and it has to re-create the data structure if ϵ changes.

Finally in this paper, we discuss the performance of our method and present our summary in the Section 5 and Section 6.

2 Basic Search Algorithm using k-d Tree

First, we describe the basic algorithm by which the k-d tree searches for the nearest neighbor. Figure 1 shows a 2-D example of a k-d tree that consists of four leaf nodes labeled A, B, C and D. We do not describe how to construct a k-d tree in this paper. For a detailed description. please refer to [1, 6]

2.1 Finding the Nearest Neighbor Point

.

Now we will describe how we find the nearest neighbor point from a query point $\boldsymbol{p}$. In the searching algorithm, we will start at the root node and traverse down to the leaf node that contains the query point. In Figure 1, the leaf node A contains $\boldsymbol{p}$, and we compute the distances from $\boldsymbol{p}$ to the records of A.

To avoid examining all leaf nodes, the algorithm prunes branches by the Bounds-Overlap-Ball (BOB) test[6]. After node A is examined, the distance from $\boldsymbol{p}$ to the nearest neighbor is d. We determine B whether d satisfies the following BOB test:

$$d > d_B, \tag{1}$$

where d_B is the distance from the query point $\boldsymbol{p}$ to the boundary of A and B. Similarly, we compare d with d_C and d_D to decide whether we will examine C and D. In this case, d satisfies (1) for B, C and D. Thus, we have to examine all nodes. If the hypersphere of radius d is completely inside of a node after examining the node, the algorithm finishes the search. (It is called Ball-Within-Bounds (BWB) test.)

Algorithm 1 SearchNearestNeighbor(N)

```
input: Node N
  if N is leaf node then
    Examine records of N
  else
    if p is inside leftson(N) then
      SearchNearestNeighbor(leftson(N))
      if d > d_rightson(N) then
        SearchNearestNeighbor(rightson(N))
      end if
    else
      SearchNearestNeighbor(rightson(N))
      if d > d_leftson(N) then
        SearchNearestNeighbor(leftson(N))
      end if
    end if
  end if
```

The basic search algorithm is represented by a recursive function (Algorithm 1). N is the node of interest. $\boldsymbol{p}$ is the query point. d is the distance of the current nearest neighbor. rightson(N) and leftson(N) mean the sons of node N. $d_{\text{rightson}(N)}$ and $d_{\text{leftson}(N)}$ are the distances from the query to the boundary of the right/left son of N.

2.2 Estimating Computational Cost

If a k-d tree contains n records, the depth of the tree is $O(\log_2 n)$. Because of this, Friedman et al.[6] say that the computational cost of searching for the nearest neighbor using the k-d tree is $O(\log_2 n)$. However, this is only for a case in which the leaf node that contains the query is examined, and all other branches are pruned by the BOB test.

Actually, in the worst case, such as Figure 1, no branch can be pruned, and the computational cost becomes $O(n)$[1]. In particular, when the distance d from the query to the nearest neighbor is large in comparison with the distribution of records in the k-d tree, almost all boundaries can be inside the ball.

3 Bounds-Overlap-Threshold Test

In this section, we propose a new method to reduce the computational cost of searching for the nearest neighbor points using the k-d tree. Many applications need nearest neighbor points that are actually near queries. Thus, when the records are far from the queries, they are either not used or contribute minimally. However, as we estimated the computational cost in the previous section, if all records

[1]This is a rough estimation. A more accurate estimation is $O(\sum_{k=0}^{\log_2 n} 2^k)$

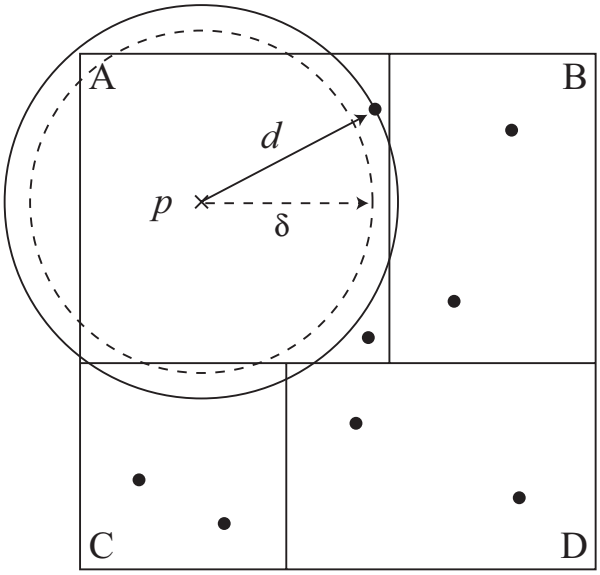

Figure 2. The Bounds-Overlap-Threshold (BOT) test

in a k-d tree are far from a query, the computational cost is larger than it is when the nearest neighbor is close.

If we can assume that it is not important if the nearest neighbor is far from a query, it is sufficient that we find out that there are no records near the query. Then, we propose a new method to reduce the computational cost when the nearest neighbor is far from a query. We introduce the Bounds-Overlap-Threshold (BOT) test to the searching algorithm.

Now, we assume that it is not important if the nearest points are farther than δ. Figure 2 shows the same situation with Figure 1. When we decide whether to examine node B, we define the BOT test as follows:

$$\delta > d_B. \tag{2}$$

We prune a branch if the BOB test or the BOT test fails. Since the BOT test failed in the cases of B and D, we did not examine them. However, we examined C, since $d > d_C$ and $\delta > d_C$.

If $d \leq \delta$, the algorithm is completely the same as that before the BOT test is introduced. If $d > \delta$, the new algorithm may not find the true nearest neighbor point. The distance d_{true} from the query to the true nearest neighbor is

$$\delta < d_N < d_{true} \leq d, \tag{3}$$

where d_N is the smallest distance from the query to the boundary of node N, which is larger than δ.

If the maximum distance of two points in a k-d tree is $2D$, the volume of the hypersphere, which contains all

Algorithm 2 SearchNearestNeighborBOT(N)

input: Node N
  if N is leaf node **then**
    Examine records of N
  else
    if $\boldsymbol{p}$ is inside leftson(N) **then**
      SearchNearestNeighborBOT(leftson(N))
      if $d > d_{\text{rightson}(N)}$ $\wedge$ $\delta > d_{\text{rightson}(N)}$ **then**
        SearchNearestNeighborBOT(rightson(N))
      end if
    else
      SearchNearestNeighborBOT(rightson(N))
      if $d > d_{\text{leftson}(N)}$ $\wedge$ $\delta > d_{\text{leftson}(N)}$ **then**
        SearchNearestNeighborBOT(leftson(N))
      end if
    end if
  end if

points of a k-d tree, is proportional to D^k. The volume of the hypersphere of radius δ is also proportional to δ^k. Therefore, if the points in a k-d tree have uniform distribution, our new method reduces the computational cost of the worst case from $O(n)$ to $O((\frac{\delta}{D})^k n)$.

The search algorithm with the BOT test is represented by a recursive function (Algorithm 2). N is the node of interest. $\boldsymbol{p}$ is the query point. d is the distance of the current nearest neighbor. rightson(N) and leftson(N) mean the sons of node N. $d_{\text{rightson}(N)}$ and $d_{\text{leftson}(N)}$ are the distance from the query to the boundary of the right/left son of N. The difference from the basic algorithm is illustrated by gray boxes.

In our implementation, the data to be loaded to memory is managed by the memory mapping function provided by the operating systems. Since we load only the data to be traversed during the search, we can reduce the required memory for searching for the nearest neighbor.

4 Applying Our New Method to Aligning and Merging Range Images

To test our new method, we considered two applications that use nearest neighbors in a multidimensional space: aligning range images and merging range images. Figure 3 shows the steps in modeling an object's shape.

4.1 Aligning Range Images

We can acquire the shape of an object from various viewpoints using range finders[5, 14]. A range image acquired by the range finders contains part of the shape of an object. Thus, we capture range images from various viewpoints to

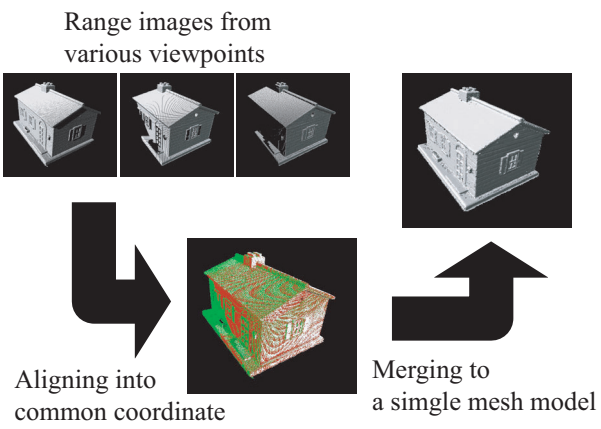

Figure 3. Steps of geometric modeling of an object

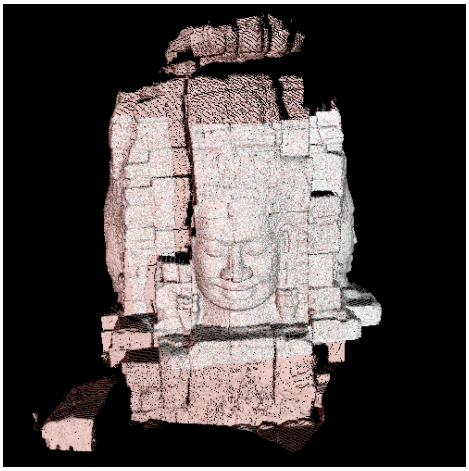

Figure 4. An example of aligned two range images (red and white)

acquire the whole shape of the object. We do not know the mutual relationship of the range images a priori. Therefore, we align them with a common coordinate system by using iterative registration techniques [2, 20, 16, 13]. To find corresponding points between two range images, we use the nearest neighbor points by searching k-d trees that we construct from each range image.

In aligning range images based on the iterative registration techniques, range images are assumed to be roughly aligned. Thus, if the distances of corresponding points are large, some methods regard that the correspondence is wrong and omit it from computing the posture of range images by thresholding or M-estimation [16, 20, 13]. Since our aligning method uses M-estimator [20], the weights of the corresponding points which are far from each other are quite small. Therefore, we set the threshold distance δ to the distance at which the weight of the M-estimator becomes sufficiently small. Even though the corresponding points are not the nearest points because of the BOT test, the result is not affected by them since the weight is quite small or zero.

Figure 4 shows the two range images (red and white) which we aligned to estimate our method. Since they are partially overlapped, about 30% of the points of the range images have no corresponding points. When we align them using the basic search algorithm, Figure 5 shows the distribution of the number of records examined during the search for a nearest neighbor point in the aligning application. The number of records examined grows according to the distance from the queries.

On the other hand, when we search for nearest neighbor points using the BOT test, we can drastically reduce the number of records examined in the area where the distance from the query is larger than d. In this experiment, we tested two thresholds, $\delta = m$ and $\delta = m + \sigma$, where m and σ are the average and the standard deviation of distances computed in the previous iteration. Figure 6 shows the distribution of the number of records examined with $\delta = m$, and Figure 7 is the result with $\delta = m + \sigma$.

The total number of records examined is 1,937,616 without the BOT test, 1,129,552 with $\delta = m (= 0.10)$, and 1,421,459 with $\delta = m + \sigma (= 0.24)$. while the number of search is 137,009. The computational cost of searching for the nearest neighbor points With the BOT test is regarded as 58.3%($\delta = m$) and 73.4%($\delta = m + \sigma$) of that of the basic search algorithm (see Table 1). The time of a search is also reduced as shown in Table 2, which shows the average time of a search computed by Intel PentiumIII 1.0GHz processor. Thus, we have demonstrated that our method efficiently finds the nearest neighbor points in the overlapping area with pruning unnecessary branches in the isolated area.

4.2 Merging Range Images

To merge range images, we convert the images to volumetric representation by computing the signed distance field (SDF) from the multiple range images [20, 4, 10]. Figure 8 shows that the merging algorithm is applied to a situation in which there are three range images which are intersecting between two neighboring voxels. First, the algorithm finds the nearest point of each range image from the the center of the voxel $\boldsymbol{x}$. Next, if $\boldsymbol{x}$ is outside by considering the normal vector of the nearest point, the signed distance $f(\boldsymbol{x})$ is positive; otherwise $f(\boldsymbol{x})$ becomes negative. The magnitude of $f(\boldsymbol{x})$ is the distance from $\boldsymbol{x}$ to the

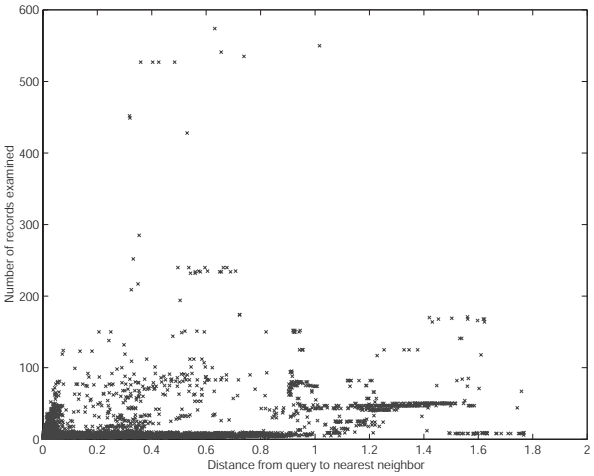

Figure 5. An example of the relationship between the distance from a query to the nearest neighbor and the number of records examined, using the basic search algorithm for aligning range images.

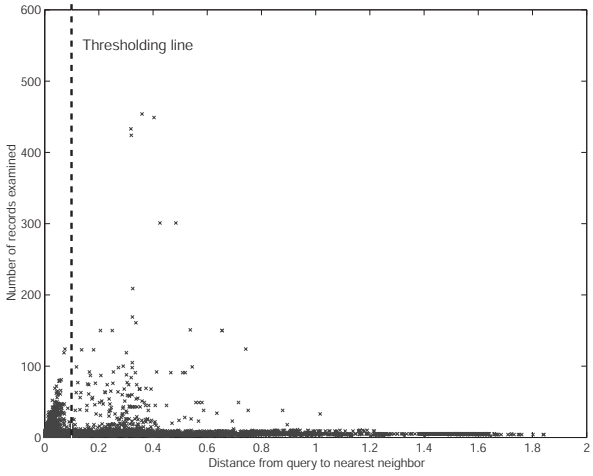

Figure 6. Relationship between distance from a query to the nearest neighbor and the number of records examined, using the BOT test for aligning range images ($\delta = m$).

nearest point. In this case, $f(\boldsymbol{x})$ is positive and $f(\boldsymbol{x}')$ is negative. We compute the signed distance for every voxel into which the whole 3D space is partitioned. Consequently, the shape of the object's surface is represented by the isosurface $f(\boldsymbol{x}) = 0$.

We create a mesh model of the whole object by converting from the SDF by using the marching cubes algorithm[12]. If the distance from a voxel to the range images is larger than $\frac{\sqrt{3}}{2}w$, where w is the interval of voxels, there is no surface around the voxel. Thus, it is enough for us to find that no point in the k-d tree is closer than $\frac{\sqrt{3}}{2}w$, and we set $\delta = \frac{\sqrt{3}}{2}w$. Our merging method, which is based on Wheeler's method[20], reduces the computation of the SDF in an octree manner. Therefore, the voxel width w varies according to the depth of octree subdivision to which the current voxel belongs. We change the threshold δ as well as the voxel width w.

We merged 32 range images, some of which are shown in Figure 3. Since they are partially overlapped, some range images are near a voxel and others are far from it. Thus, our method efficiently finds the nearest neighbor with pruning unnecessary branches of far range images. Figure 9 and Figure 10 shows an example of the distribution of the number of records examined during the search for a nearest neighbor point in the merging application. When we search for the nearest neighbor points using the BOT test, the number of records examined gets closer to 1 at any distance from the query. This is because we adjust threshold δ according to the voxel width. In this example, the total numbers of records examined are 12,510,697,618 without the BOT test and 902,199,301 with the BOT test, while the number of search is 57,470,464. Specifically, the computational cost of searching the nearest neighbor points is reduced to 7.2% of that of the basic search algorithm (see Table 1), and Table 2 shows the computational time computed by Intel Xeon 2.4GHz processor.

Table 1. The number of records examined of the basic algorithm and our new method

	Basic algorithm	with BOT test	Ratio
Aligning$_{(\delta = m)}$	1,937,616	1,129,552	58.3%
Aligning$_{(\delta = m + \sigma)}$	1,937,616	1,421,459	73.4%
Merging	12,510,697,618	902,199,301	7.2%

5 Discussion

The performance of the BOT test depends on the application. In this section, we consider the application conditions in which our method works best. First, the most important requirement in applying our method is that the threshold δ can be determined in the application. After determining δ, the performance of the BOT test depends on the distribution of distances from queries to nearest neighbor points. Our method works best when the portion of the number of nearest neighbor points that are farther than δ becomes larger. In many cases in aligning and merging range images, since

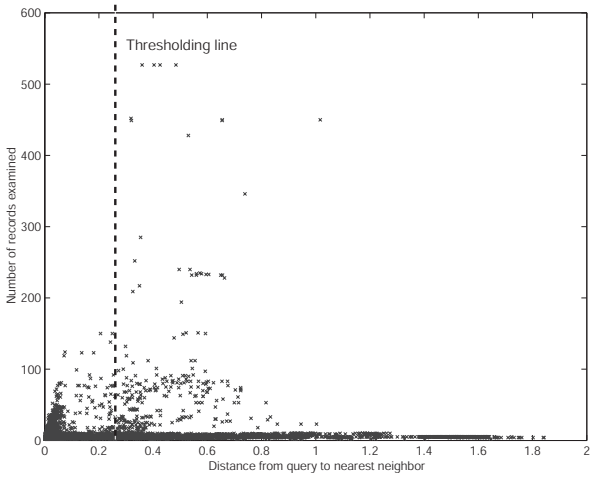

Figure 7. Relationship between distance from a query to the nearest neighbor and the number of records examined, using the BOT test for aligning range images ($\delta = m + \sigma$).

Table 2. Average time of the basic algorithm and our new method

	Basic algorithm	with BOT test	Ratio
Aligning$_{(\delta = m)}$	12.42usec	9.26usec	74.6%
Aligning$_{(\delta = m + \sigma)}$	12.42usec	10.58usec	85.2%
Merging	52.12usec	8.92usec	17.1%

range images are partially overlapped as shown in Section 4, some range images are far from a query point. It is not important to find the nearest neighbor point for far range images in aligning and merging. Thus, our method efficiently reduces the computational cost by pruning far range image points. The reason why the computational cost in merging range images was drastically reduced is considered to be that only a few range images are near a voxel and most of all range images are far from it. Therefore, it is expected that the efficiency of our method increases if we align many range images simultaneously because most of all range images are far from a query point.

A simple solution to prune branches of far range image points is to omit them in creating a k-d tree if they are outside of a bounding box; however, it is necessary to re-create the k-d tree if the position and size of a bounding box changes. In particular, as shown in merging range images, our merging method uses the variable threshold δ. Our method can be applied without re-creating the structure of a k-d tree. Therefore, our method is efficient in cases in which the appropriate threshold varies according to the

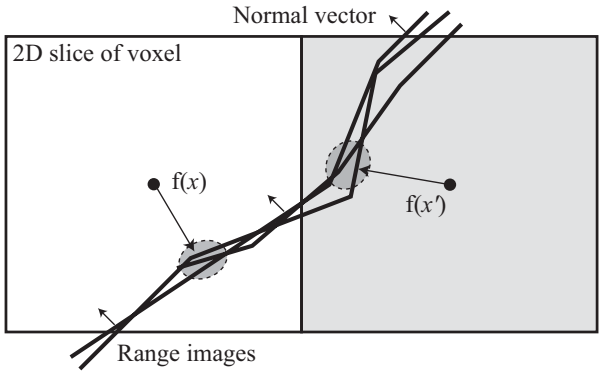

Figure 8. Compute signed distance by finding the nearest points of range images.

situation. On the other hand, methods which uses a parameter to define neighborhood, such as a simple bounding box and Nene's method [15], have to re-create its structure for searching in those cases.

From the viewpoint of the correctness of the nearest neighbor point, if the distance is smaller than δ, the found result is correct. If the distance is larger than δ, the distance to the correct nearest neighbor is in the range given by (3). Thus, (3) gives us the estimation for the distance of the true nearest neighbor. However, we cannot obtain a good estimation for the vector of the true nearest neighbor. Therefore, we have to set δ larger than the minimum distance of the vector of the nearest neighbor that we need to obtain.

6 Summary

In this paper, we have proposed a new algorithm for searching for the nearest neighbor by using the k-d tree. If the nearest neighbor point is far from a query, it is not an important nearest neighbor in many applications. Thus, we have proposed the Bounds-Overlap-Threshold test, which does not search strictly by pruning branches if the nearest neighbor point is beyond a threshold. This technique drastically reduces the computational cost if the nearest neighbor is far from a query. Since the threshold of the BOT test can be changed without re-creating a k-d tree, the technique is suitable for applications in which variable thresholds are considered. Finally, we have discussed the performance, which depend on the distribution of the distance from a query to the nearest neighbor.

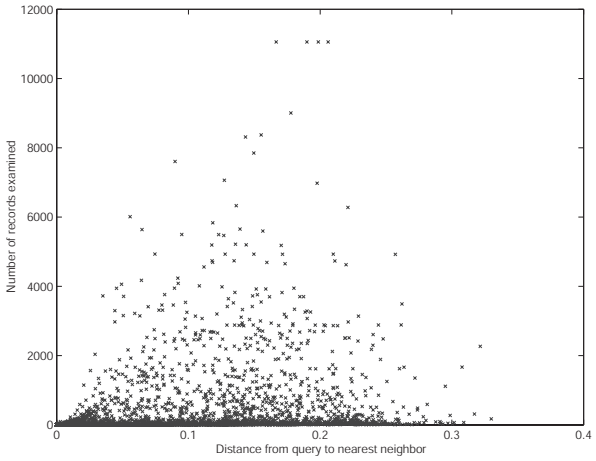

Figure 9. Relationship between distance from a query to the nearest neighbor and the number of records examined using the basic search algorithm for merging range images.

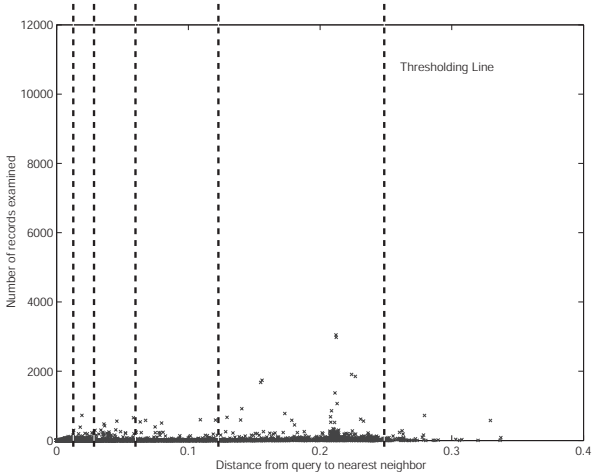

Figure 10. Relationship between distance from a query to the nearest neighbor and the number of records examined with the BOT test for merging range images.

Acknowledgments

This research was, in part, supported by JST under CREST Ikeuchi Project.

References

[1] J. L Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[2] P.J. Besl and N.D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Patt. Anal. Machine Intell.*, 14(2):239–256, Feb 1992.

[3] A. Califano and R. Mohan. Multidimensional indexing for recognizing visual shapes. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 16:373–392, 1994.

[4] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Proc. SIGGRAPH'96*, pages 303–312. ACM, 1996.

[5] Cyra Technologies, Inc. http://www.cyra.com.

[6] J.H. Friedman, J. Bentley, and R. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, 1977.

[7] Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[8] M. Greenspan and G. Godin. A nearest neighbor method for efficient ICP. In *Proc. 3DIM*, pages 161–168, 2001.

[9] A. Guttman. R-trees: A dynamic index struc- ture for spatial searching. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 47–54, 1984.

[10] A. Hilton, A.J. Stoddart, J. Illingworth, and T. Windeatt. Reliable surface reconstruction from multiple range images. In *Proceedings of European Conference on Computer Vision*, pages 117–126, Springer-Verlag, 1996.

[11] D. B. Lomet and B. Salzberg. The hb-tree: A multiattribute indexing method with good guaranteed performance. *ACM Trans. Database Systems*, 15(4):625–658, 1990.

[12] W. Lorensen and H. Cline. Marching cubes: a high resolution 3d surface construction algorithm. In *Proc. SIGGRAPH'87*, pages 163–170. ACM, 1987.

[13] T. Masuda, K. Sakaue, and N. Yokoya. Registration and integration of multiple range images for 3-d model constuction. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 879–883, June 1996.

[14] MINOLTA Co. Ltd. Vivid 900 non-contact digitizer. http://www.minoltausa.com/vivid/.

[15] S.A. Nene and S.K. Nayar. A simple algorithm for nearest neighbor search in high dimensions. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 19(9):989–1003, 1997.

[16] K. Pulli. Multiview registration for large data sets. In *Second Int. Conf. on 3D Digital Imaging and Modeling*, pages 160–168, Oct 1999.

[17] J. T. Robinson. The k-d-b tree: A search structure for large multidimensional dynamic indexes. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 10–18, 1984.

[18] H Samet. The quadtree and related hierarchical data structure. *ACM Computing Surveys*, 16(2):187–260, 1984.

[19] D. A. Simon, M. Hebert, and T. Kanade. Realtime 3-d pose estimation using a high speed range sensor. In *Proc. ICRA*, pages 2235–2241, 1994.

[20] Mark D. Wheeler. *Automatic Modeling and Localization for Object Recognition*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1996.

[21] Haim J. Wolfson and Isidore Rigoutsos. Geometric hashing: An overview. *IEEE Computational Science & Engineering*, 4(4):10–21, 1997.