Cached *k*-d tree search for ICP algorithms

Andreas Nüchter, Kai Lingemann, and Joachim Hertzberg
University of Osnabrück, Institute of Computer Science
Knowledge-Based Systems Research Group
Albrechtstraße 28, D-49069 Osnabrück, Germany
nuechter@informatik.uni-osnabrueck.de

Abstract

The ICP (Iterative Closest Point) algorithm is the de facto standard for geometric alignment of three-dimensional models when an initial relative pose estimate is available. The basis of ICP is the search for closest points. Since the development of ICP, k-d trees have been used to accelerate the search. This paper presents a novel search procedure, namely cached k-d trees, exploiting iterative behavior of the ICP algorithm. It results in a significant speed-up of about 50% as we show in an evaluation using different data sets.

1 Background

Registering 3D models is a crucial step in 3D model construction. Many application benefit from efficient ICP algorithms: Nowadays precise 3D scanners are available that are used in architecture, industrial automation, agriculture, cultural heritage conservation, and facility management. These 3D scanners deliver tons of 3D data as point clouds. Other applications of point cloud registration algorithms include medical data processing, art history, archaeology, and rescue and inspection robotics. The advent of 3D cameras is likely to generate another burst of ICP applications in the near future [22, 23].

ICP algorithms are widely used for registering geometric 3D models in a common coordinate system. Given two point clouds and a starting guess for the relative poses, the algorithm computes the rotation and translation such that the 3D models fit together. ICP is an iterative algorithm based on searching for closest points. The key of efficient implementation is the fast computation of closest points. This paper presents a novel search method, namely, cached *k*-d trees, for computing of closest points iteratively. The method is based on *k*-d trees and a buffer for leaf nodes. The buffer is used to start backtracking in subsequent searches. Therefore, the iterative structure of ICP is exploited for speed-ups. The proposed search algorithm computes exact nearest neighbors as closest point, thus side effects due to approximation cannot occur.

The paper is organized as follows: The next part presents briefly the ICP algorithm, followed by a discussion of related work. The following section describes the data structures used for closest point searches within the ICP algorithm and introduces our novel approach. The obtained results are then evaluated. Section 4 concludes.

1.1 The ICP Algorithm

The ICP Algorithm was developed by Besl and McKay [5] and is usually used to register two given point sets in a common coordinate system. The algorithm calculates iteratively the registration. In each iteration step, the algorithm selects the closest points as correspondences and calculates the transformation, i.e., rotation and translation $(\mathbf{R}, \mathbf{t})$, for minimizing the equation

$$E(\mathbf{R},\mathbf{t}) = \sum_{i=1}^{N_m} \sum_{j=1}^{N_d} w_{i,j} \left|\left|\mathbf{m}_i - (\mathbf{R}\mathbf{d}_j + \mathbf{t})\right|\right|^2, \qquad (1)$$

where N_m and N_d, are the number of points in the model set M and data set D, respectively, and w_{ji} are the weights for a point match. The weights are assigned as follows: $w_{ji} = 1$, if $\mathbf{m}_i$ is the closest point to $\mathbf{d}_j$, $w_{ji} = 0$ otherwise. Eq. (1) can be reduced to

$$E(\mathbf{R},\mathbf{t}) \quad \propto \quad \frac{1}{N} \sum_{i=1}^{N} \left|\left|\mathbf{m}_i - (\mathbf{R}\mathbf{d}_i + \mathbf{t})\right|\right|^2, \qquad (2)$$

with $N = \sum_{i=1}^{N_m} \sum_{j=1}^{N_d} w_{i,j}$, since the correspondence matrix can be represented by a vector $\mathbf{v}$ containing the point pairs, i.e., $\mathbf{v} = ((\mathbf{d}_1, \mathbf{m}_{f(\mathbf{d}_1)}), (\mathbf{d}_2, \mathbf{m}_{f(\mathbf{d}_2)}), \ldots, (\mathbf{d}_{N_d}, \mathbf{m}_{f(\mathbf{d}_{N_d})}))$, with $f(\mathbf{x})$ the search function returning the closest point. The assumption is that in the last iteration step the point correspondences, thus the vector of point pairs, are correct.

In each ICP iteration, the transformation can be calculated by any of these four methods: A SVD based method of Arun et al. [1], a quaternion method of Horn [12], an algorithm using orthonormal matrices of Horn et al. [13] and a calculation based on dual quaternions of Walker et al. [24]. These algorithms show similar performance and stability concerning noisy data [15].

Besl and McKay show that the iteration terminates in a minimum [5]. Note: Normally, implementations of ICP would use a maximal distance for closest points to handle partially overlapping point sets. In this case the proof in [5] does no longer hold, since the number of points as well as the value of $E(\mathbf{R}, \mathbf{t})$ might increase after applying a transformation.

1.2 State of the Art

Many variants of the ICP algorithm have been proposed in recent years. Different strategies for point reduction, i.e., point selection, matching and weighting have been proposed and evaluated [20]. Rusinkiewicz and Levoy propose a high speed ICP variant using a point-to-plane error metric [16] and a projection-based method to generate point correspondences [6]. Furthermore they conclude that the other stages of the ICP process appear to have little effect of convergence rate, so that they choose the simplest ones, namely random sampling, constant weighting, and a distance threshold for rejecting point pairs [20].

ICP algorithms tend to have problems if too many points are chosen from featureless regions of the data. In this case the algorithm converges slowly, finds the wrong pose, or even diverges. Normal space sampling, as proposed by Rusinkiewicz and Levoy, aims at constraining translational sliding of input meshes, generated from the point cloud [20]. Their algorithm tries to ensure that the normals of the selected points uniformly populate the sphere of directions. Covariance sampling as proposed by Levoy et al. extends the normal space approach. They identify whether a pair of meshes will be unstable in the ICP algorithms by estimating a covariance matrix from a sparse uniform sampling of the input [8].

However, these state of the art ICP variants all assume that the input data is given as a mesh. In many application scenarios a mesh is not available, e.g., 3D data in robotics. Here, measurements contain in addition to Gaussian noise so called salt-and-pepper noise. Furthermore, in robotics the scenes are often sparsely sampled by the sensor. For these two reasons, simple meshing methods based on the topology of the acquired points cannot be applied and roboticists stick to using the raw point clouds. In this case the point-to-point metric, cf. Eq. (1), and closest point search have to be used. For computing closest points, *k*-d trees [7] are the standard search structure (see section 2.1 for a detailed description of *k*-d trees). Simon et al. obtained much speedup from a closest point cache that reduced the number of necessary *k*-d tree lookups [21].

Improving the speed of ICP algorithms received much attention recently. The developed wide variety of methods aim to increase the performance of computing corresponding points. Currently available methods include for example exploiting the triangle equation [11, 9] and heuristics based on multi resolution [14].

Recently, Greenspan and Yurick used approximate *k*-d trees for ICP algorithms [10]. The idea behind this is to return as an approximate nearest neighbor the closest point in the bucket region where the query point lies. This value is determined from the depth-first search only, thus expensive ball-within-bounds tests and backtracking are not used [10]. Inspired by these ideas, the ICP algorithms has been evaluated in [17] using the approximate nearest neighbor search introduced by Arya et al. [2, 3]: *k*-d trees empirically outperform bd-trees (box decomposition trees) with and without approximation. Approximation does not significantly deteriorate the quality of scan registration, but significantly increases the speed of scan matching [17].

The following section presents a novel method for computing exact closest points for ICP. It combines *k*-d trees with caching.

2 Closest Point Search

2.1 *k*-d trees

k-d trees are a generalization of binary search trees. Every node represents a partition of a point set to the two successor nodes. The root represents the whole point cloud and the leaves provide a complete disjunct partition of the points. These leaves are called buckets (cf. Fig. 2). Furthermore, every node contains the limits of the represented point set.

2.1.1 Searching *k*-d trees

k-d trees are searched recursively. A given 3D point needs to be compared with the separating plane in order to decide on which side the search must continue. This procedure is executed until the leaves are reached. There, the algorithm has to evaluate all bucket points. However, the closest point may be in a different bucket, iff the distance to the limits is smaller than the one to the closest point in the bucket. In this case backtracking has to be performed. Fig. 2 shows a backtracking case, where the algorithms has to go back to the root. The test is known as ball-within-bounds test [4, 7, 10].

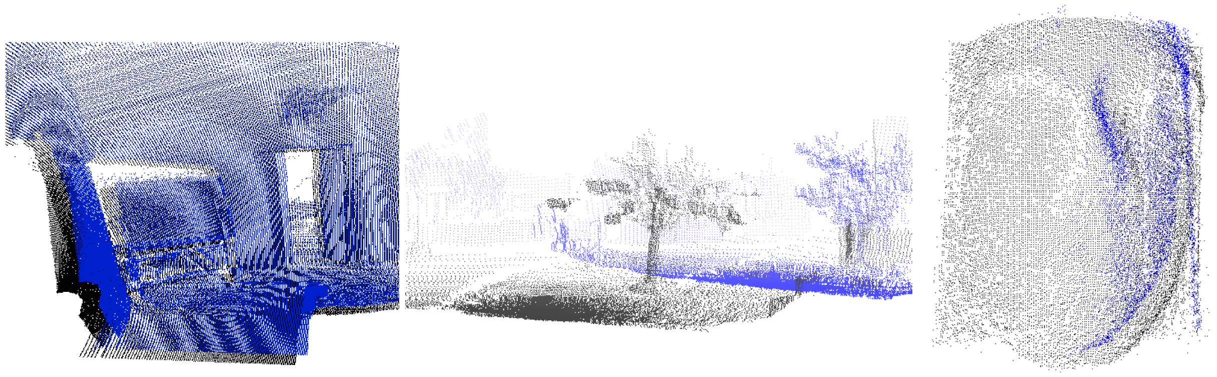

Figure 1. Some test scenes used in this paper. Left: Typical cluttered indoor environment. Middle: Outdoor environment with trees. Right: 3D face scans.

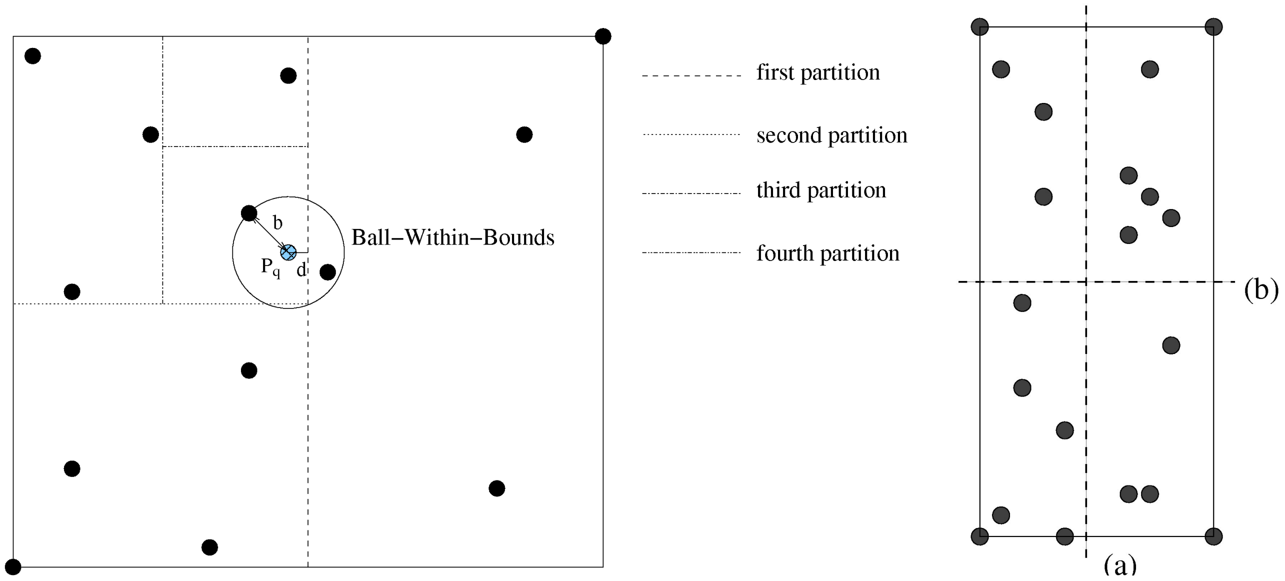

Figure 2. Left: Recursive construction of a k-d tree. If the query consists of point p_q, k-d tree search has to backtrack to the tree root to find the closest point. Right: Partitioning of a point cloud. Using the cut (b) rather than (a) results in a more compact partition and a smaller probability of backtracking [7].

2.1.2 The optimized *k*-d tree

The objective of optimizing *k*-d trees is to reduce the expected number of visited leaves. Three parameters are adjustable, namely, the direction and position of the split axis as well as the maximal number of points in the buckets. Splitting the point set at the *median* ensures that every *k*-d tree entry has the same probability [7]. The median can be found in linear time, thus the time complexity for constructing the tree is not affected. Furthermore, the split axis should be oriented *perpendicular* to the longest axis to minimize the amount of backtracking (see Fig. 2). Friedman and colleagues prove that a bucket size of 1 is optimal [7]. Nevertheless, in practice it turned out that a slightly larger bucket size is faster [11].

2.2 Approximate *k*-d tree search

S. Arya and D. Mount introduce the following notion for approximating the nearest neighbor in *k*-d trees [2]: Given an $\varepsilon > 0$, then the point $\mathbf{p} \in D$ is the $(1+\varepsilon)$-approximate nearest neighbor of the point $\mathbf{p}_q$, iff

$$||\mathbf{p} - \mathbf{p}_q|| \leq (1+\varepsilon)\,||\mathbf{p}^* - \mathbf{p}_q||\,,$$

where $\mathbf{p}^*$ denotes the true nearest neighbor, i.e., $\mathbf{p}$ has a maximal distance of ε to the true nearest neighbor. Using this notation, in every step the algorithm records the closest point $\mathbf{p}$. The search terminates if the distance to the unanalyzed leaves is larger than

$$||\mathbf{p}_q - \mathbf{p}||\,/(1+\varepsilon).$$

Fig. 3 (left) shows an example where the gray cell needs not to be analyzed, since the point $\mathbf{p}$ satisfies the approximation criterion.

2.3 Approximate box decomposition trees

Arya et al. have presented an algorithm for approximate nearest neighbor search and proved its optimality [3]. They use a balanced box decomposition tree (bd-tree) as their primary data structure. This tree combines two important properties of geometric data structures: First, as in the *k*-d tree case, the set of points is exponentially reduced. Second, the aspect ratio of the tree edges is bounded by a constant. Not even the optimized *k*-d tree is able to make this assurance, but quadtrees show this characteristic [3]. The actual box decomposition search tree is composed of splits and shrinks. Fig. 3 (c) shows the general structure.

The search procedure of bd-trees is similar to the one of approximate *k*-d trees. The approximate search is discontinued (cf. Fig. 3) if the distance to the unanalyzed leaves is larger than

$$||\mathbf{p}_q - \mathbf{p}||\,/(1+\varepsilon).$$

2.4 Cached *k*-d tree search

2.4.1 The cached *k*-d tree search

k-d trees with caching contain, in addition to the limits of the represented point set and to the two child node pointers, one pointer to the predecessor node. The root node contains a null pointer. During the recursive construction of the tree, this information is available and no extra computations are required.

For the ICP algorithm, we distinguish between the first and the following iterations: In the first iteration, a normal *k*-d tree search is used to compute the closest points. However, the return function of the tree is altered, such that in addition to the closest point, the pointer to the leaf containing the closest point is returned and stored in the vector of point pairs. This supplementary information forms the cache for future look-ups.

In the following iterations, these stored pointers are used to start the search. If the query point is located in the bucket, the bucket is searched and the ball-within-bounds test applied. Backtracking is started, iff the ball lies not completely within the bucket. If the query point is not located within the bucket, then backtracking is started, too. Since the search is started in the leaf node, explicit backtracking through the tree has to be implemented using the pointers to the predecessor nodes (see Fig. 4). Algorithm 1 summarizes the ICP with cached *k*-d tree search.

Algorithm 1 ICP with cached *k*-d tree search

```
1:  for $i = 0$ to maxIterations do
2:    if $i == 0$ then
3:      for all $\mathbf{d}_j \in D$ do
4:        search k-d tree of set $M$ top down for point $\mathbf{d}_j$
5:        $\mathbf{v}_j = (\mathbf{d}_j, \mathbf{m}_{f(\mathbf{d}_j)}, \text{ptr\_to\_bucket}(\mathbf{m}_{f(\mathbf{d}_j)}))$
6:      end for
7:    else
8:      for all $\mathbf{d}_j \in D$ do
9:        search k-d tree of set $M$ bottom up for point $\mathbf{d}_j$
          using ptr_to_bucket($\mathbf{m}_{f(\mathbf{d}_j)}$)
10:       $\mathbf{v}_j = (\mathbf{d}_j, \mathbf{m}_{f(\mathbf{d}_j)}, \text{ptr\_to\_bucket } \mathbf{m}_{f(\mathbf{d}_j)})$
11:     end for
12:   end if
13:   calculate transformation $(\mathbf{R}, \mathbf{t})$ that minimizes the error function Eq. (2)
14:   apply transformation on data set $D$
15: end for
```

2.4.2 Performance of cached *k*-d tree search

The proposed ICP variant uses exact closest point search. In contrast to the previously discussed approximate *k*-d tree

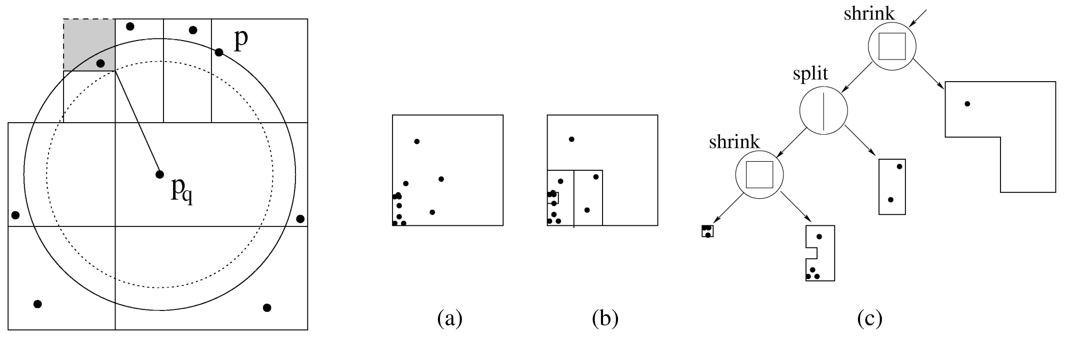

Figure 3. Left: The $(1+\varepsilon)$-approximate nearest neighbor. The solid circle denotes the ε environment of $\mathbf{p}_q$. The search algorithm need not analyze the gray cell, since $\mathbf{p}$ satisfies the approximation criterion. Middle and right: (a) Given point set. (b) decomposition into buckets. (c) Tree layout. Fig. adapted from [2, 3].

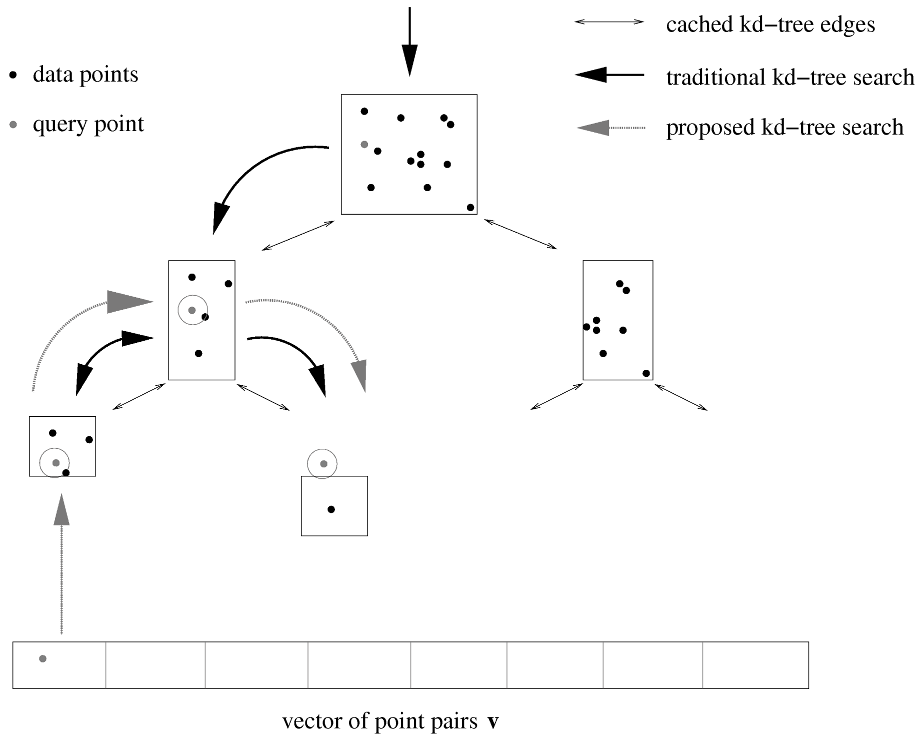

Figure 4. Schematic description of the proposed search method: Instead of closest point searching from the root of the tree to the leaves that contain the data points, a pointer to the leaves is cached. In the second and following ICP iteration, the tree is searched backwards.

search for ICP algorithms [10, 17], registration inaccuracies or errors due to approximation cannot occur.

Friedman et al. prove that searching for closest points using k-d trees needs logarithmic time [7], i.e., the amount of backtracking is independent of the number of stored points in the tree. Since the ICP algorithm iterates the closest point search, the performance derives to $\mathcal{O}(IN_d \log N_m)$, with I the number of iterations. Note: Brute-force ICP algorithms have a performance of $\mathcal{O}(IN_d N_m)$.

The proposed cached k-d tree search needs $\mathcal{O}((I + \log N_m)N_d)$ time in the best case. This performance is reached if constant time is needed for backtracking, resulting in $N_d \log N_m$ time for constructing the tree, and $I \cdot N_d$ for searching in case no backtracking is necessary. Obviously the backtracking time depends on the computed ICP transformation $(\mathbf{R}, \mathbf{t})$. For small transformations the time is nearly constant.

Cached k-d tree search needs $\mathcal{O}(N_d)$ extra memory for the vector $\mathbf{v}$, i.e., for storing the pointers to the tree leaves. Furthermore, additional $\mathcal{O}(N_m)$ memory is needed for storing the backwards pointers in the k-d tree.

3 Evaluation and Results

The proposed method has been evaluated with 5 data sets from different domains. The computation was done on a Pentium-IV with 2.8 GHz running Linux OS, with the same compiler options, i.e., with `gcc -O2`. Since k-d tree search and cached k-d tree search are very similar, most parts of the code were identical in the comparison experiments. In all tests, ICP with cached k-d tree search outperformed ICP with conventional k-d tree search. Next, the detailed results on one particular data set are described and explained. Then we summarize the the results from the data sets.

The data set "cluttered indoor environment" was recorded during the Rescue Robotics Camp 2004 with a 3D laser range finder that is built on basis of a SICK scanner. A servo motor is used to achieve a controlled pitch motion of the 2D scanner. Such 3D scanners are commonly used in robotics. Four detailed analyses are provided:

1. The performance of the cached k-d tree search depending on a change of the bucket size was tested: For small bucket sizes, the speed-up is larger (Fig. 5, top left). This behavior originates from the increasing time needed to search larger buckets.

2. The search time per iteration was recorded during the experiments (Fig. 5, top right). For the first iteration the search times are equal, since cached k-d tree search uses conventional k-d tree search to create the cache. In the following iterations, the search time drops significantly and remains nearly constant. The conventional k-d tree search increases in speed, too. Here, the amount of backtracking is reduced due to the fact that the calculated transformations $(\mathbf{R}, \mathbf{t})$ are getting smaller.

3. The number of points to register influences the search time. With increasing number of points, the positive effect of caching algorithms becomes more and more significant (Fig. 5, bottom left).

4. The overall performance of the ICP algorithm depends both on the search time and on the construction time of the tree. However, the construction time of the trees seems to be negligible. In addition, a comparison with a reference implementation shows the effective implementation. As reference implementation the software from the papers [2, 3] was used (Fig. 5, bottom right).

In addition to the detailed analysis, a number of experiments with different data sets have been made. Table 1 summarizes the results on different data sets originating from various ICP applications. Overall, a speedup in the order of 50% percent is achieved. The speedup on a point set with random points is lower, since more cache misses occur. The reason for this effect is that ICP aligns scans using their local structures or clusters. The cache exploits the data conglomeration, too, but it is not present in the random point set.

4 Conclusion

In this paper we present a simple search method that improves k-d tree search for ICP algorithms for point clouds. The algorithm exploits the iterative fashion of the ICP by caching pointers to tree buckets containing the closest points. Since the transformations in each step become smaller from iteration to iteration, the number of tree operations is drastically reduced. The resulting ICP variant is usually about 50% faster than the conventional k-d tree ICP algorithm.

Needless to say, a lot of work remains to be done. In future work we plan to develop an algorithm suite that contain implementations of Elias' algorithm [19], search methods that include the exploitation of the triangle equation [9] and other caching methods, e.g. [21]. Such a framework is necessary to allow an unbiased evaluation.

Furthermore, we are currently working on global registration methods for several 3D point clouds. Here, the ICP error function is replaced by a global error function, that moves all scans in the minimization step. New search methods are needed, since after applying the transformation, the (cached or approximate) k-d trees have to be time-consumingly altered or reconstructed.

Table 1. Computation times for ICP based scan registration using standard k-d tree search vs. cached k-d tree search. The test set differ in size and registration of scans is applied repeatedly to construct a complete model. Illustrations of parts of the first two and last data sets are given in Fig. 1.

Name of the data set	standard k-d tree search	cached k-d tree search	speedup
Cluttered indoor environment	29345 ms	12959 ms	56%
Outdoor environment	521 sec	311 sec	41%
Abandoned mine	1112 ms	532 ms	52%
Rescue arena	4378 ms	2159 ms	51%
Facial scans	32 sec	13 sec	59%
Random point set	3556 ms	2521 ms	29%

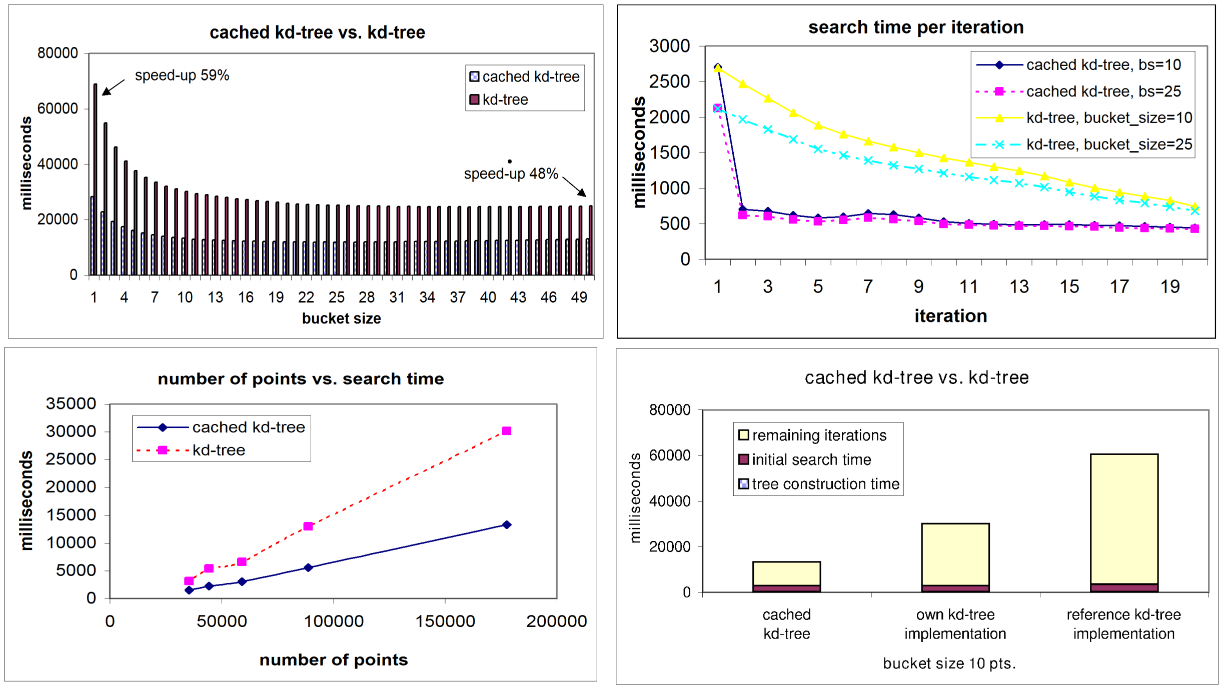

Figure 5. Detailed results for the data set "cluttered indoor environment". Top Left: search time vs. bucket size. Top right: search time per iteration for bucket sizes 10 and 25. Bottom left: Search time depending on the number of points. Bottom right: Overall comparison of the algorithms and a reference k-d tree implementation [2, 3].

References

[1] K. S. Arun, T. S. Huang, and S. D. Blostein. Least square fitting of two 3-d point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(5):698 – 700, 1987.

[2] S. Arya and D. M. Mount. Approximate nearest neigbor queries in fixed dimensions. In *Proceedings of the 4th ACM-SIAM Symposium on Discrete Algorithms*, pages 271 – 280, 1993.

[3] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An Optimal Algorithms for Approximate Nearest Neighbor Searching in Fixed Dimensions. *Journal of the ACM*, 45:891 – 923, 1998.

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

[5] P. Besl and N. McKay. A method for Registration of 3–D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 14(2):239 – 256, February 1992.

[6] G. Blais and D. Levine. Registration Multiview Range Data to Create 3D Computer Objects. *IEEE Transaction on Pattern Analysis and Machine Intelligence (PAMI)*, 17(8):820 – 824, August 1995.

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

[8] N. Gelfand, S. Rusinkiewicz, and M. Levoy. Geometrically Stable Sampling for the ICP Algorithm. In *Proceedings of the 4th IEEE International Conference on Recent Advances in 3D Digital Imaging and Modeling (3DIM '03)*, Banff, Canada, October 2003.

[9] M. Greenspan and G. Godin. A Nearest Neighbor Method for Efficient ICP. In *Proceedings of the third International Conference on 3D Digital Imaging and Modeling (3DIM '01*, pages 161 – 168, Quebec City, Canada, May 2001.

[10] M. Greenspan and M. Yurick. Approximate K-D Tree Search for Efficient ICP. In *Proceedings of the 4th IEEE International Conference on Recent Advances in 3D Digital Imaging and Modeling (3DIM '03)*, pages 442 – 448, Banff, Canada, October 2003.

[11] M. A. Greenspan, G. Godin, and J. Talbot. Acceleration of Binning Nearest Neighbor Methods. In *Proceedings of the Conference Vision Interface*, 2000.

[12] B. K. P. Horn. Closed–form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629 – 642, April 1987.

[13] B. K. P. Horn, H. M. Hilden, and Sh. Negahdaripour. Closed–form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society of America A*, 5(7):1127 – 1135, July 1988.

[14] T. Jost and H. Hügli. A Multi-Resolution ICP with Heuristic Closest Point Search for Fast and Robust 3D Registration of Range Images. In *Proceedings of the 4th IEEE International Conference on Recent Advances in 3D Digital Imaging and Modeling (3DIM '03)*, Banff, Canada, October 2003.

[15] A. Lorusso, D. Eggert, and R. Fisher. A Comparison of Four Algorithms for Estimating 3-D Rigid Transformations. In *Proceedings of the 4th British Machine Vision Conference (BMVC '95)*, pages 237 – 246, Birmingham, England, September 1995.

[16] P. Neugebauer. Geometrical Cloning of 3D Objects via Simulataneous Registration of Multiple Range Images. In *Proceedings of the Fourth Symposium on Solid Modeling and Applications (SMA '97)*, pages 130 – 139, Atlanta, Georgia, U.S.A., May 1997.

[17] A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann. 6D SLAM with Approximate Data Associoation. In *Proceedings of the 12th IEEE International Conference on Advanced Robotics (ICAR '05)*, pages 242 – 249, Seattle, U.S.A., July 2005.

[18] A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann. Accurate Object Localization in 3D Laser Range Scans. In *Proceedings of the 12th IEEE International Conference on Advanced Robotics (ICAR '05)*, pages 665 – 672, Seattle, U.S.A., July 2005.

[19] R. L. Rivest. On the optimality of elias's algorithm for performing best match searches. *Information Processing*, 74:678 – 681, 1974.

[20] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Proceedings of the Third International Conference on 3D Digital Imaging and Modellling (3DIM '01)*, pages 145 – 152, Quebec City, Canada, May 2001.

[21] D. Simon, M. Hebert, and T. Kanade. Real–time 3–D pose estimation using a high–speed range sensor. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA '94)*, volume 3, pages 2235 – 2241, San Diego, CA, USA, May 1994.

[22] PMDTechnologies GmbH; `http://www.pmdtec.com/inhalt/produkte/kamera.htm`, 2006.

[23] Swissranger; `http://www.swissranger.ch/`, 2006.

[24] M. W. Walker, L. Shao, and R. A. Volz. Estimating 3-d location parameters using dual number quaternions. *CVGIP: Image Understanding*, 54:358 – 367, November 1991.