Are You Using the Right Approximate Nearest Neighbor Algorithm?

Stephen O'Hara and Bruce A. Draper
Colorado State University
Fort Collins, CO 80523
svohara@cs.colostate.edu

Abstract

Many computer vision tasks such as large-scale image retrieval and nearest-neighbor classification perform similarity searches using Approximate Nearest Neighbor (ANN) indexes. These applications rely on the quality of ANN retrieval for success. Popular indexing methods for ANN queries include forests of kd-trees (KDT) and hierarchical k-means (HKM). The dominance of these two methods has led to implementations in many popular computer vision tool kits, perhaps leading some to assume that ANN indexing is a solved problem. This paper introduces a new method, the Proximity Forest, which is based on a forest of randomized metric trees. We present evidence that forests of metric trees significantly outperform KDT and HKM when evaluated on real-world data, controlling for dimensionality, data set size, distance function, and the number of neighbors returned by the query. Randomized metric trees are simple to implement, are generic in regards to the chosen distance function, and can be applied to a wider class of data representations.

1. Introduction

Efficient and accurate methods for Approximate Nearest Neighbor (ANN) queries are important for large-scale computer vision applications such as similarity search and nearest-neighbor classification. Popular ANN methods include forests of randomized kd-trees (KDT), hierarchical k-means (HKM), and locality-sensitive hashing (LSH). There may be an assumption by many in the computer vision community that ANN indexing is a solved problem. But is it? Are these algorithms so good that any new methods would only provide marginal benefit?

The popularity of ANN indexing methods in computer vision has grown with the interest in large-scale image retrieval using bag-of-words representations. Nister and Stewenius were among the first to demonstrate fast image retrieval from a data set consisting of a million images and billions of descriptors [11]. Underlying their method's scalability is a Hierarchical K-Means (HKM) structure for indexing the visual vocabulary. Along similar lines, Mooseman et al. presented randomized clustering forests for building fast visual vocabularies [9]. Philbin et al. [13] use a forest of kd-trees (KDT) to speed up k-means clustering of visual vocabularies. Silpa-Anan and Hartley explored optimizations for KDT to speed up image descriptor matching [14]. Another popular approach to ANN indexing is through Locality Sensitive Hashing (LSH) [6], which uses a hashing mechanism such that neighboring data samples have similar hash codes.

Today, there are popular implementations of KDT and HKM that make these methods de facto standards in the vision community. FLANN (Fast Library for Approximate Nearest Neighbors) is an excellent software package developed by Muja and Lowe that includes implementations of KDT and HKM and a mechanism for automatically selecting and tuning these algorithms for a given data set [10]. FLANN is also packaged as part of the OpenCV library. Another popular library containing implementations of HKM and KDT is the VLFeat package from Vedaldi and Fulkerson [17].

Although KDT, HKM and LSH are well known and widely used ANN algorithms, they assume the data samples are points in a vector space. When this assumption is violated, a different ANN indexing mechanism is required. There exist structures for nearest-neighbor indexing of general metric spaces (see §2). They involve creating tree structures to partition metric spaces, and thus we refer to them generically as *metric trees*.

However, there is little precedent in the literature for employing metric trees in contemporary computer vision applications. Since data is often represented by feature vectors with similarity defined by metrics such as the Euclidean or $L1$ norm, in principle, metric trees could be used for image retrieval or nearest-neighbor classification. Instead, the use of KDT, HKM, and hashing approaches has dominated the discussion. Is the reason for this because metric trees perform poorly on vector data, or have they simply have been overlooked by those in the field?

Our attempt to answer this question is embodied by the evaluation presented in this paper. We introduce an ANN indexing method, called a Proximity Forest, which is a forest of randomized metric trees. Our contribution is two-fold. First, the Proximity Forest is the first description of a forest of randomized metric trees that we are aware of. Second, upon comparing Proximity Forests with KDT and HKM, we found that randomized forests of metric trees yield substantially more accurate results on traditional computer vision data representations.

2. ANN Indexing Methods

A kd-tree ([5]) is an index for exact nearest neighbor query that partitions a vector space by recursively generating hyperplanes to cut along coordinates where there is maximal variance in the data. A modification to support ANN indexing with kd-trees is the addition of a priority queue to eliminate backtracking in queries [2, 1]. Forests of kd-trees consist of kd-trees that randomize the splitting criteria by stochastically choosing the partitioning coordinate. The effectiveness and speed of KDT (e.g., [14]) has made this approach very popular.

ANN queries can also be supported via hierarchical k-means clustering. Nister and Stewenius describe HKM in [11]. There is evidence showing that HKM performs competitively with KDT, and that both outperform locality sensitive hashing (LSH) on real-world data [10].

Uhlmann introduced metric trees for the partitioning of general metric spaces [16]. Uhlmann presents two methods for constructing metric trees. The first way is to randomly select one of the points in the data set to be a pivot. The distances from each of the remaining points to the pivot are computed. The median distance is chosen as the partition boundary, which divides the data into two balanced sets: those closer to the pivot than the median distance, and those further away. Repeating this process recursively yields a tree structure which covers the data set. It can be viewed as recursively partitioning the metric space with median-radii hyperspheres centered on the pivot points. The second metric tree described by Uhlmann is the Generalized Hyperplane tree (GH-tree). In a GH-tree, two pivots are selected at each tree node in such a way that they are relatively far apart. The remaining points are split according to which of the two pivots they are closest to.

Yianilos developed the Vantage Point tree (vp-tree), which is essentially the same as Uhlmann's median-splitting metric tree, but the pivots are denoted as "vantage points" [18]. Yianilos provides analysis on the performance guarantees of vp-trees, and additional pivot selection strategies. M-trees, introduced by Ciaccia et al. [4], extend metric trees with optimizations for dynamic data sets. Chavez et al. [3] provides a survey of these and other related metric space searching techniques.

Liu et al. [7] compares GH-trees with Locality Sensitive Hashing (LSH) [6], noting the following advantages of GH-trees: they automatically adapt their splitting resolution according to the density of the local data and the hyperplanes used to partition the data can be along any direction, whereas in LSH the hyperplanes are constrained to align with coordinate directions. Importantly, Liu considers how to adapt metric trees for use in approximate similarity search applications, which trades exact results for speed and scalability. Liu introduced the spill tree, which is a modification of the GH-tree that incorporates an overlap region to allow for efficient approximate nearest neighbor search by eliminating back tracking.

More recently, O'Hara et al. introduced a structure called a Subspace Forest [12] for supporting ANN queries of points on Grassmann manifolds. The Subspace Forest was used for performing ANN classification of actions in video clips, demonstrating state-of-the-art performance on benchmark action recognition data sets. O'Hara's construction of a Subspace Tree is similar to that of a median-splitting metric tree, with the specific requirement that the data points are fixed-dimensional subspaces and the distance function, called the chordal distance, is derived from the principal angles between the subspaces and forms a metric over the Grassmann manifold.

For the remainder of this paper, we use the term "metric tree" (uncapitalized) to indicate any of several variations of tree structures for partitioning metric spaces. "Metric Tree" (capitalized), will specifically indicate Uhlmann's algorithm for median-distance based hypersphere partitioning of a metric space.

3. Method

There are two questions being addressed by this paper. The first is whether there is an ANN mechanism that will work on general metric data with performance comparable to vector-space methods when applied to vector data. The second is whether the most popular methods used by computer vision experts for large scale similarity search, namely KDT and HKM, have the best performance on real-world data sets compared with other options. Our results indicate that the answer to the second question is also answered by the first. A simple structure based on metric trees not only provides accurate proximity queries of non-Euclidean metric data (as in [12]), but is also notably more accurate than KDT and HKM when applied to commonly used image feature vectors.

We describe a method for ANN queries based on generating a forest of randomized metric trees. Our structure is called a *Proximity Forest*, and is described in detail below. We compare the Proximity Forest to KDT and HKM on data sets consisting of 128-dimensional integer-valued SIFT features, 12-dimensional real-valued MSER features

and 3-dimensional real-valued point cloud data. We evaluate the performance impact of the following variables: dimensionality, data set size, distance function, and the number of neighbors to return in the query. Each is described in more detail below, as well as a further description of the data sets and the implementations used in the evaluation.

We do not directly compare Proximity Forests to LSH, as it has been shown in other work [10] that KDT and HKM outperform LSH for indexing SIFT features and similar computer vision data. We realize there are many other ANN indexing algorithms, but a comprehensive competitive analysis is outside the scope of this evaluation.

3.1. Proximity Forest

A *Proximity Forest* consists of a set of *Proximity Trees*, described below. Nearest neighbor computations are performed on each tree in the forest, and the best results from each tree are compared to return the nearest neighbors from the forest. Proximity Trees are constructed in a manner similar to Metric Trees. However, unlike Metric Trees, the Proximity Tree approximates the median by computing the distance between the pivot and a random subset of the input points. Doing so allows the Proximity Forest to be constructed incrementally as data arrives in a streaming source. A Proximity Tree can be characterized as a randomized metric tree.

A batch algorithm for constructing a Proximity Tree is shown in Algorithm 1. We present the batch construction for clarity even though our actual implementation builds the tree in an incremental fashion. To determine the neighbors of a sample, it is submitted to the tree and sorted to a leaf node. Sorting at each node occurs based on whether or not the distance between the sample and the pivot element is less than the current node's distance threshold. As shown in the algorithm, the distance threshold (which varies from node to node) is based on the median distance between a subset of elements to the randomly selected pivot element. The elements of the leaf node are considered the sample's neighborhood, from which the K nearest can be selected.

Due to the near/far nature of the splitting criterion, nodes in a left subtree may often be more self-similar than nodes of a right subtree. With enough samples, the right subtrees will be broken down, recursively, into smaller subsets of greater self-similarity. Yet it remains true that the far right leaf nodes of a Proximity Tree are often less compact neighborhoods than the rest. This is mitigated by using a forest of trees, and selecting the nearest from the set returned by each tree. See Figure 1.

3.2. Software Implementation

We implemented the Proximity Forest using the Python programming language. The Proximity Forest implementation is an unoptimized serial implementation which is available to the public under an open source license. [1] We employed Muja's FLANN library [10] for both the KDT and HKM implementations. The FLANN library is mature, reasonably optimized, and freely available. More specifically, we use the pyFlann Python bindings to FLANN.

Algorithm 1: ProximityTree(S,τ,δ)

/* Recursive construction of a Proximity Tree */ ;
Input: S, a set of data elements from which to construct the tree
Input: τ, the number of data elements to sample before splitting
Input: δ, a distance function
if $|S| < \tau$ **then Return** /*Leaf node. Base case for recursion */ ;
$\hat{S} \leftarrow$ random selection of τ elements from S ;
$P \leftarrow$ random selection of a pivot element from $\hat{S}$;
$D \leftarrow \{\delta(x, P),\ \forall x \in \hat{S}\}$;
$dt \leftarrow median(D)$;
/* Partition S into left and right subsets */ ;
$S_{<} \leftarrow \{x \in S \mid \delta(x, P) \leq dt\}$;
$S_{>} \leftarrow \{x \in S \mid \delta(x, P) > dt\}$;
$LeftChild \leftarrow$ ProximityTree($S_{<}$, τ, δ) ;
$RightChild \leftarrow$ ProximityTree($S_{>}$, τ, δ) ;

3.3. Data Sets

Our evaluation uses three data sets: SIFT descriptors, MSER detections, and 3D point cloud data. We use SIFT and MSER because they have been used extensively in image retrieval applications [15, 11, 13]. We use 3D point cloud data because it is low-dimensional spatial data, allowing us to contrast performance of ANN methods between low-dimensional and high-dimensional data.

The first data set consists of 10,000 SIFT features, available from the FLANN library repository.[2] SIFT features are 128-dimensional integer-valued histogram vectors. The second data set consists of over seven million MSER points, which were sampled to produce testing sets of size 10K, 100K, and 1M. MSER points were proposed by Matas et al. for wide-baseline stereo correspondence [8], but have since been used as feature detectors for image retrieval and other tasks (e.g., [11]). The MSER data is available for download from the University of Kentucky benchmark data site.[3] The final data set consists of 250,000 points from a 3D point cloud generated by scanning the handles of a pair of scissors.[4]

4. Empirical Evaluation

This section describes five tests that were conducted to compare the accuracy of the three methods while controlling for: 1) data dimensionality, 2) the number of K nearest neighbors to return, 3) data set size, 4) distance measure, and 5) forest size, in the case of Proximity Forests and KDT.

[1] http://sourceforge.net/projects/proximityforest
[2] http://people.cs.ubc.ca/ mariusm/uploads/FLANN/datasets/dataset.hdf5
[3] http://vis.uky.edu/ stewe/ukbench/
[4] http://www.qcgroup.com/engineering/resources/

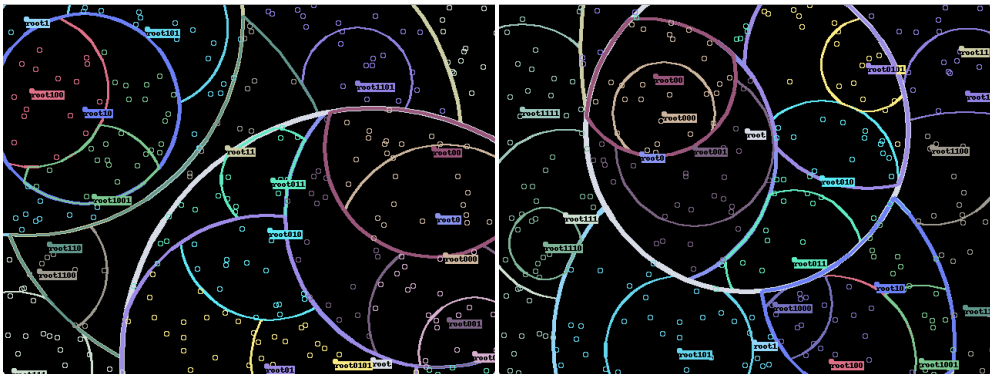


Figure 1: Partitioning planar data with two trees of a Proximity Forest. Poorly localized neighborhoods in a single tree are mitigated by using a forest.

All experiments we run are using real-world data. We do not evaluate on synthesized data. Our focus is on data of broad interest to the computer vision community, and synthetic data can introduce regularities and other artifacts that can skew results. See the discussion in §4 of Gionis et al. [6] on the importance of using real data sets for evaluating approximate nearest neighbor algorithms.

There is only one parameter to select when using the Proximity Forest, which is τ, the number of data elements to sample per node when determining the estimated median distance to the pivot. We set $\tau = 15$ for all experiments reported upon in this paper.

4.1. Experiment 1: Data Dimensionality

In this experiment, we fix the data set size at 10K points, randomly divided into 9K target points used to build the ANN index and 1K query points for testing. We compare performance on 3D, MSER, and SIFT features, which represent dimensions of 3, 12, and 128, respectively. For each query, we return the three nearest neighbors (3NN), using the Euclidean distance.

We compare the accuracy of the three methods, Proximity Forest (PF), KDT, and HKM, against the exact results. For each query, we score a point for each of the results that are one of the true 3NN. For a single trial, there are 1,000 queries, so the maximum score is 3,000. Accuracy is the percentage of the max score. All results reported in this paper represent the average over five trials of 1,000 queries each.

For PF and KDT, 15 trees in the forest were employed. For HKM, we initially used a branching factor of 128 with 15 iterations, in agreement with recommended settings from [10]. However, we found that HKM accuracy was improved in this experiment by using the default settings in the FLANN implementation, which is a branching factor of 32 with 5 iterations.

Results are shown in Figure 2. The variation in the results of all three methods is small – the standard deviation is shown in the figure as the red section of each bar. All three methods perform well on the 3D point cloud data, with PF correctly finding nearly all exact neighbors on all five trials. The performance difference between PF and the other methods is more pronounced for higher dimensional data, with a 15 to 20% improvement over the next best method. For the 12-dimensional MSER data, PF finds 98.5% of the exact 3NN, while the next best method is HKM at 80.5%. With the 128-dimensional SIFT data, PF gets nearly 75% of the 3NN, while KDT and HKM score less than 57%.

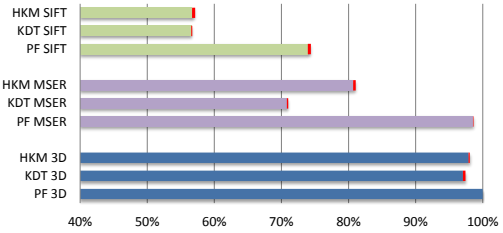


Figure 2: ANN Accuracy vs. Data Dimension. Red caps at end of bars indicate the width of one standard deviation in the results.

4.2. Experiment 2: Varying K

In this experiment, we use the same settings and data sets as before, but we vary K to find the 1, 3, 5, or 10 nearest neighbors of every query point. Results are shown in Figure 3. PF outperforms the other two methods for every value of K on both MSER and SIFT data.[5] As K increases, all methods suffer a penalty to accuracy, which is more pronounced on the higher-dimensional SIFT data.

[5] This is also true for the 3D data, which was omitted for space.

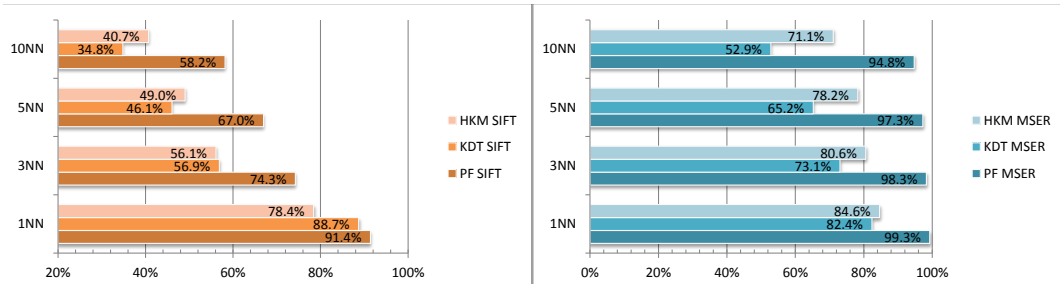
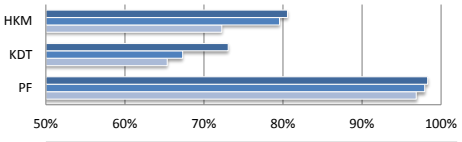


Figure 3: ANN Accuracy vs. K. Performance of the three methods is shown varying the number, K, of nearest neighbors to return. On SIFT data (left side), the accuracy advantage of PF over the other methods is more pronounced for $K \geq 3$. On MSER (right side), the performance advantage is consistent for all K.

4.3. Experiment 3: Data Set Size

Experiment 3 tests whether the size of the data set used to build the index affects the relative accuracy of the three ANN methods. We used the MSER data to create subsets of size 10K, 100K, and 1M, from which 1K were selected as query points and the remaining as the target set. All other settings were as previously described, with K=3. Results are shown in Figure 4. The Proximity Forest outperforms the other methods over all three data sizes. Compared to PF, HKM and KDT both experience a larger drop in accuracy between the smallest and largest data sets. These results suggest that PF accuracy may scale better to very large data sets than the other two methods.



	PF	KDT	HKM
10K MSER	98.3%	73.1%	80.6%
100K MSER	97.9%	67.3%	79.6%
1M MSER	96.9%	65.4%	72.3%

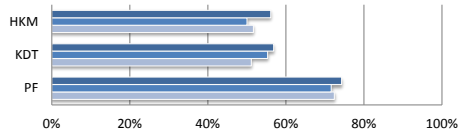
Figure 4: ANN Accuracy vs. Data Set Size.

4.4. Experiment 4: Distance Measure

In this experiment, we test three distance functions for the nearest neighbor index to determine if distance function choice affects the relative performance of the methods. We test on 10K SIFT points, K=3, using Euclidean, Manhattan, and χ^2 distance functions. All other parameters are the same as in the previous experiments. Figure 5 shows that PF outperforms the other two methods regardless of the choice of distance function used in the test.

4.5. Experiment 5: Forest Size

PF and KDT are both methods that employ a forest of randomized trees. This experiment compares the accuracy



	PF	KDT	HKM
Euclidean	74.27%	56.87%	56.10%
Manhattan	71.63%	55.33%	50.03%
Chi-Squared	72.43%	51.17%	51.67%

Figure 5: ANN Accuracy vs. Distance Measure.

of these two methods against the size of the forest. We apply both methods to the three data sets, using 10K points each, and compute the 3NN accuracy with forest sizes: $\{1, 3, 5, 10, 15, 20, 25\}$. Other settings are the same as previously described. The results are shown in Figure 6. PF performance suffers with forest sizes of 3 or fewer trees. A single randomized metric tree is not a good ANN indexing method, but with a modest number of trees, performance significantly outperforms KDT, especially on higher-dimensional data.

5. Conclusion

In this paper, we draw attention to a method of performing approximate nearest neighbor queries that may have been overlooked by many in the computer vision community. Metric trees are structures that allow for the partitioning of general metric spaces. While a single metric tree may not yield a strong ANN index, a forest of randomized metric trees, which we call a Proximity Forest, appears to return substantially more accurate ANN queries on a variety of real-word vector data.

Notwithstanding the provocative title of this paper, we make no claim that a Proximity Forest is the best possible ANN indexing method. Instead, this paper serves to highlight a practical alternative indexing strategy, and to encourage those building vision systems to think beyond current black-box ANN implementations.

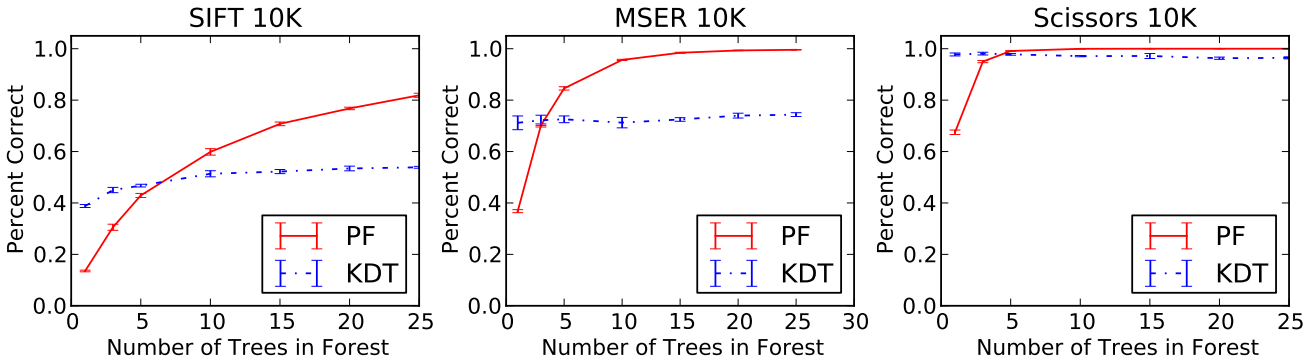


Figure 6: ANN Accuracy vs. Forest Size. Error bars, which are small, show the upper and lower 95% confidence.

In the data sets explored in this paper, the accuracy improvement over randomized kd-trees and hierarchical k-means is nearly 20% for MSER and SIFT data. This performance gain is consistent across tests controlling for various factors that could influence indexing performance, such as data set size, distance function, and desired number of nearest neighbors (K).

By using metric trees, the Proximity Forest also has the significant advantage of being able to index any metric data. This allows the computer vision practitioner interested in large-scale similarity search to explore a wider variety of potential image/video representations.

References

[1] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998. 2

[2] J. S. Beis and D. G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Proc. Computer Vision and Pattern Recognition (CVPR)*. IEEE, 1997. 2

[3] E. Chavez, G. Navarro, R. Baeza-Yates, and J. L. Marroquin. Searching in metric spaces. *ACM Computing Surveys (CSUR)*, 33(3):273–321, 2001. 2

[4] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proc. Int'l Conf. on Very Large Data Bases (VLDB)*, 1997. 2

[5] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226, 1977. 2

[6] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proc. Int'l Conf. on Very Large Data Bases (VLDB)*, 1999. 1, 2, 4

[7] T. Liu, A. W. Moore, A. Gray, and K. Yang. An investigation of practical approximate nearest neighbor algorithms. In *Proc. Neural Information Processing Systems (NIPS)*, 2004. 2

[8] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, 22(10):761–767, 2004. 3

[9] F. Moosmann, W. Triggs, and F. Jurie. Randomized clustering forests for building fast and discriminative visual vocabularies. In *Proc. Neural Information Processing Systems (NIPS)*, 2006. 1

[10] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *Proc. Int'l Conf. on Computer Vision Theory and Application (VISSAPP)*, 2009. 1, 2, 3, 4

[11] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *Proc. Computer Vision and Pattern Recognition (CVPR)*, 2006. 1, 2, 3

[12] S. O'Hara and B. A. Draper. Scalable action recognition using a subspace forest. In *Proc. Computer Vision and Pattern Recognition (CVPR)*, 2012. 2

[13] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *Proc. Computer Vision and Pattern Recognition (CVPR)*, 2007. 1, 3

[14] C. Silpa-Anan and R. Hartley. Optimised KD-trees for fast image descriptor matching. In *Proc. Computer Vision and Pattern Recognition (CVPR)*, 2008. 1, 2

[15] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *Proc. Int'l Conf. on Computer Vision (ICCV)*, 2003. 3

[16] J. K. Uhlmann. Satisfying general proximity / similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179, Nov. 1991. 2

[17] A. Vedaldi and B. Fulkerson. VLFeat: an open and portable library of computer vision algorithms. In *Proc. Int'l Conf. on Multimedia (MM)*, 2010. 1

[18] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 311–321, 1993. 2