Shape Indexing Using Approximate Nearest-Neighbour Search in High-Dimensional Spaces

Jeffrey S. Beis and David G. Lowe
Department of Computer Science
University of British Columbia
Vancouver, B.C., Canada V6T 1Z4
beis/lowe@cs.ubc.ca

Abstract

Shape indexing is a way of making rapid associations between features detected in an image and object models that could have produced them. When model databases are large, the use of high-dimensional features is critical, due to the improved level of discrimination they can provide. Unfortunately, finding the nearest neighbour to a query point rapidly becomes inefficient as the dimensionality of the feature space increases. Past indexing methods have used hash tables for hypothesis recovery, but only in low-dimensional situations. In this paper, we show that a new variant of the k-d tree search algorithm makes indexing in higher-dimensional spaces practical. This **Best Bin First**, *or BBF, search is an approximate algorithm which finds the nearest neighbour for a large fraction of the queries, and a very close neighbour in the remaining cases. The technique has been integrated into a fully developed recognition system, which is able to detect complex objects in real, cluttered scenes in just a few seconds.*

1. Introduction

Shape-based indexing uses feature vectors from an image to access an index structure, rapidly recovering possible matches to a database of object models. This procedure is much more efficient than the alternative of exhaustive comparison, at a cost of some offline preprocessing to create the index, and the extra memory required to store it.

The goal of indexing is to recover from the index the most similar model shapes to a given image shape. In terms of feature vectors, or points in a feature space, this corresponds to finding a set of *nearest neighbours* (NN) to a query point. Virtually all previous indexing approaches in model-based vision [4, 5, 9, 10, 16, 18, 19] have used hash tables for this task. This is somewhat surprising since it is well-known in other communities (e.g. pattern recognition, algorithms) that tree structures do the job much more efficiently.

In large part this oversight can be explained by the fact that indexing techniques are generally applied in low-dimensional spaces, where hash table search can be quite efficient. Such spaces are adequate when the number of objects is small, but higher-dimensional feature vectors are essential when the model database becomes large, because they provide a much greater degree of discrimination. Unfortunately, nearest-neighbour search times depend exponentially on the dimension of the space.

One data structure that has been used extensively for NN lookup is the k-d tree [6]. While the "curse of dimensionality" is also a problem for k-d trees, the effects are not as severe. Hash table inefficiency is mainly due to the fact that bin sizes are fixed, whereas those in a k-d tree are adaptive to the local density of stored points. Thus, in some cases (high-density region), a hashing approach will have to do a long linear search through many points contained in the bin in which the query point lands; in other cases (low-density), an extensive search through adjacent bins may be required before the best neighbour can be determined. In addition, there is the difficulty of choosing an appropriate bin size.

In a previous paper [2], we presented an indexing method which uses k-d trees to recover a small set of nearest neighbours to an image feature vector. The neighbours are used to compute probabilities for each model-image match hypothesis, which allows the hypotheses to be ranked so that the most likely of them can be verified first. This process was shown to be effective in a $4D$ feature space, with a small model database of 4 objects. In this paper we will demonstrate that the technique is feasible in spaces with up to 10 or 20 dimensions. The combination of highly specific feature vectors and probabilistic hypothesis generation provides extremely efficient indexing.

Our method relies on the rapid recovery of nearest neighbours from the index. In high-dimensional spaces, standard

k-d tree search often performs poorly, having to examine a large fraction of the points in the space to find the exact nearest neighbour. However, a variant of this search which efficiently finds *approximate* neighbours will be used to limit the search time. The algorithm, which we have called *Best Bin First* (BBF) search, finds the nearest neighbour for a large fraction of queries, and finds a very good neighbour the remaining times. This type of search has wider application than for shape indexing alone: another vision-related use would be for closest point matching in appearance-based recognition (see e.g. [15]).

2. Previous work

We first discuss several prominent indexing methods which have used hash tables for indexing. Forsythe, *et al.*[5], outlined several types of projective invariant features for indexing planar objects viewed in arbitrary $3D$ orientations. However, their experiments were carried out using a $2D$ index space generated by pairs of conics. Rothwell, *et al.*[16], used $4D$ indices defined by area moments of planar curves, that were first transformed into canonical reference frames. Clemens and Jacobs [4] generated $4D$ and $6D$ index spaces from hand-grouped point sets. For each of the methods above, the dimensionality of the spaces and the number of models were too low for the inefficiency of hashing to be critical.

The *geometric hashing* indexing technique (e.g. [10]) is also generally applied in low- (2- or 3-) dimensional index spaces. While this technique differs substantially from other indexing methods in that voting overcomes some of the difficulty with bin boundaries, Grimson [7] notes that performance is poor even with small amounts of noise and clutter. This is due to the hash table becoming overloaded with entries, and indicates that the use of higher-dimensional spaces is important. In [9], geometric hashing is applied with larger ($8D$) indices generated from planar curves ("footprints"). However, the experiments did not truly test indexing performance because only a few models were used, with 3 or 4 features each.

Stein and Medioni presented a method for $2D$-from-$2D$ [18] and $3D$-from-$3D$ [19] indexing that also used hash table lookup. While the former method used index spaces of between 3 and 6 dimensions, the latter work considered higher- (up to about 14-) dimensional spaces. They avoided the exponential (with dimension) search for NN by using extremely coarse quantization of the bins (e.g., $60°$ for angle features), and looking only in the single bin containing the query point. This leads to the recovery of a large number of hypotheses, with a low signal-to-noise ratio, since highly dissimilar shapes are allowed to match. Nor does it preclude missing a significant percentage of good hypotheses which lie just across one of the many bin boundaries.

In [3], Califano and Mohan argue for the use of higher-dimensional spaces. Their analysis indicates a dramatic speedup from increasing the size of the feature vectors used for indexing. However, they again use hash tables for the lookup and do no search. Thus, in their method a pose clustering stage is required to accumulate results, presumably because of the high likelihood that a query will land in a different bin than that of the best neighbour.

In the pattern classification literature, various types of tree have been used to find nearest neighbours. Just as in hashing, these methods divide the space into bins; unlike hashing, the methods are *adaptive* in that the partition of the data space depends on the data points to be stored.

The best-known and most widely used of these is the *k-d tree* [6], a complete binary tree having smaller bins in the higher-density regions of the space. The analysis in the original paper shows expected logarithmic time lookup, but experiments in higher dimensions [17] have shown that the method often suffers from inefficient search, in many cases having to examine a large fraction of the stored points to determine the exact nearest neighbour. Other trees specialized to fast NN lookup exist (e.g. [8, 14]), but are sub-optimal because more constraints are set on where bin boundaries can be placed.

Instead of finding the exact nearest neighbour, if we are willing to accept an *approximate* neighbour in some fraction of the cases, then processing time can be greatly reduced. One early method [13] uses a hierarchical tree data structure in which internal nodes are the centers of mass of the nodes at the next lower level. The point recovered from the first leaf node encountered provides an approximate NN in a very short search time, but the quality of this neighbour is relatively low.

In [1], Arya presents an approach based on neighborhood graphs. While his main results concern the asymptotic behavior of one particular algorithm, which contains impractically large constant factors, he does give a practical variant (RNG*) that demonstrates good results for moderate dimensionalities (8-16). Independent of our own work, he also develops the equivalent of BBF search which he terms "priority k-d tree search." Comparisons between RNG* and the priority algorithm show comparable performance for moderate dimensionalities.

Most recently, Nene and Nayar [15] have used a different type of approximation to NN lookup. They only guarantee recovery of the best neighbour if it is within ϵ of the query point, by using a set of sorted lists of the coordinates of the stored points, and successively eliminating points further away than ϵ, one dimension at a time. This method is effective if ϵ can be kept small (≤ 0.25 when each dimension has unit extent), but in higher-dimensional spaces that is only possible when the number of points is extremely large, on the order of $(\frac{1}{\epsilon})^k$.

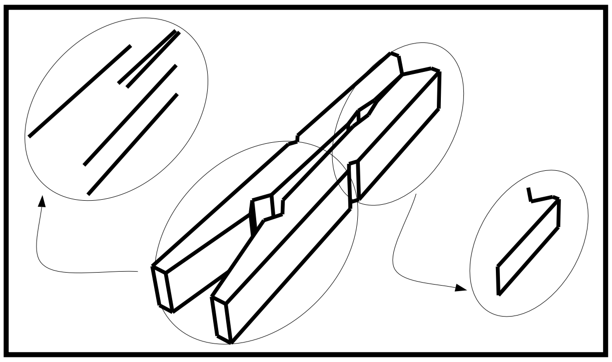

Figure 1. Examples of feature groupings, including a parallel segment grouping (upper left) and a segment chain grouping (lower right). Each model segment may be redundantly included in several groupings, which may be of various cardinalities and extend to different areas of the object.

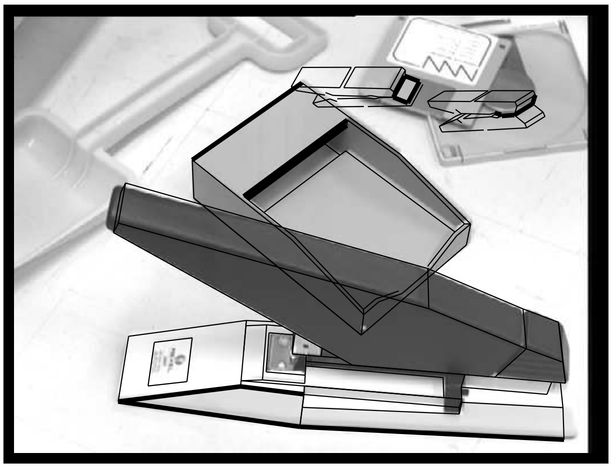

Figure 2. Example of recognition system showing correct and incorrect indexing hypotheses, and properly determined pose for the correct hypotheses. The initial shape match hypotheses are highlighted (bold lines).

3. Shape-based indexing

We now outline our method of shape-based indexing [2], within which the BBF algorithm provides an important component. To create a shape index, a preprocessing stage is required in which a set of feature vectors is extracted from the database of object models. These are stored in the index, each with a pointer to the model feature set that generated it. At runtime, feature vectors are extracted from an image and used to access the index, recovering a set of neighbours to each image vector that provide model-image match hypotheses.

Our approach is somewhat unusual in that most other methods have used feature vectors that are *invariant* to the full set of model transformations (rotation, translation, scaling, and projection). Presumably this has been due to fears of excessive memory usage or lookup time. We have found that neither fear is justified, and that requiring objects to include invariant feature sets is too restrictive.

Our method instead uses features that are partially invariant (i.e. to translation, scaling, and image plane rotation), but vary with out-of-plane rotations. Simple perceptual grouping methods [11] that use the properties of parallelism and cotermination in various combinations have been used to build complex feature sets, with straight edge features serving as the primitives (Figure 1). Angles and edge length ratios, both of which satisfy the partial invariance property, are extracted from these groupings and used to form high-dimensional feature vectors.

Each view of an object in general contains several, possibly overlapping feature groupings. As well, all groupings satisfy the additional perceptual grouping property of proximity. This combination of locality and redundancy of the feature sets means that indexing can succeed even with significant occlusion and noise. A second form of redundancy, the use of several different cardinalities of grouping simultaneously, provides robustness for those cases where none of the larger segment groupings are faithfully detected.

To deal with the variation of shape with viewpoint, it is necessary to store feature vectors for a set of views that provides a good coverage of the viewing sphere. This introduces two problems: (a) the need to interpolate between nearby views; and (b) the possibility that so many views will be required that memory or NN lookup times will be excessive.

We solve these problems as follows. At runtime, a set of nearest neighbours is found using a new form of k-d tree search (BBF) which sets an absolute time limit on the search. Then, a kernel-based density estimation method (weighted k-NN) uses the recovered neighbours to interpolate between views, at the same time generating *a posteriori* probability estimates for the likelihood that a given match hypothesis is correct. Hypotheses can thus be prioritized so that the least ambiguous are verified first, an important aspect of any efficient recognition system [20].

From each match hypothesis, it is possible to estimate object pose [12]. False positives are then rejected using an iterative back-projection procedure. At each stage, the image is examined for further matches near the predicted object feature locations, and a least-squares fit of predicted versus actual image locations is computed. This is over-constrained, so it is highly unlikely that an incorrect initial match will lead to many additional feature correspondences. Therefore, the final recognition can be robust and accurate despite the fact that the initial indexing is probabilistic.

All of the above components (grouping, indexing, hypothesis ranking, and verification) have been integrated into a fully functioning recognition system. Rapid recovery of nearest neighbours is a key to making the entire process ef-

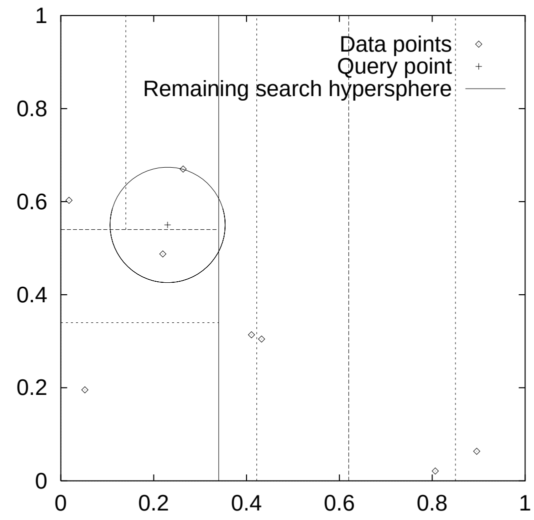

Figure 3. k-d tree with 8 points, k=2. In BBF search, we examine the bins closest to the query point q first. More than the standard search, this is likely to maximize the overlap of (A) the hypersphere centered on q with radius D_{cur}, and (B) the hyperrectangle of the bin to be searched. In the case above, BBF would reduce the number of leaf nodes examined since the NN is in the closest adjacent bin in space (directly below q), rather than the closest bin in the tree structure (to the left of q).

ficient. The NN search described below keeps lookup time manageable, with the proviso that a neighbour will occasionally be missed. The large degree of redundancy built into our method means that this is rarely a problem. Our current system is capable of recognizing $3D$ objects from a small (10) model database, in real images of complex scenes with clutter and occlusion, in a few seconds (e.g. Figure 2). Experiments with synthetic data show that performance should scale well to a larger number of models.

4. k-d tree lookup of nearest neighbours

Before introducing the BBF search algorithm, we first review the standard version of the k-d tree, which is built as follows. Beginning with a complete set of N points in $\Re^k$, the data space is split on the dimension i in which the data exhibits the greatest variance. A cut is made at the median value m of the data in that dimension, so that an equal number of points fall to one side or the other. An internal node is created to store i and m, and the process iterates with both halves of the data. This creates a balanced binary tree with depth $d = \lceil \log_2 N \rceil$.

The leaves of a k-d tree form a complete partition of the data space, with the interesting property that bins are smaller in higher-density regions and larger in lower density areas. This means that there is never an undue accumulation of points in any single bin, and that the NN to any query should lie, with high probability, in the bin where the query falls, or

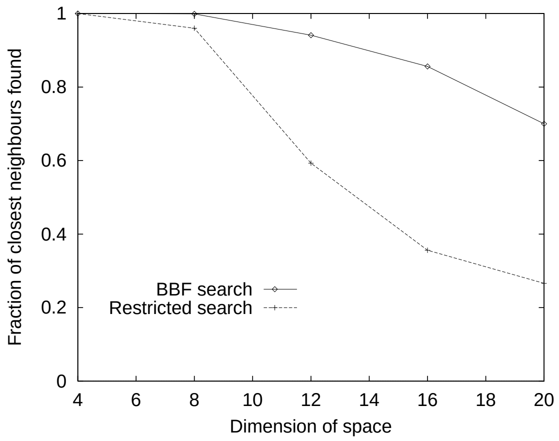

Figure 4. Approximate NN lookup vs dimension of space: Fraction of neighbours found. (Uniform distribution; $100,000$ stored points; $E_{max}(BBF)$=200; $E_{max}(restricted)$=480; averaged over 1000 queries.)

in an adjacent bin.

To look up the NN to a query point q, a backtracking, branch-and-bound search is used. First the tree is traversed to find the bin containing the query point. This requires only d scalar comparisons, and in general the point recovered from the bin is a good approximation to the nearest neighbour. In the backtracking stage, whole branches of the tree can be pruned if the region of space they represent is further from the query point than D_{cur} (the distance from q to the closest neighbour yet seen). Search terminates when all unexplored branches have been pruned.

This process can be very effective in low-dimensional spaces, but in higher dimensions there are many more bins adjacent to the central one that must be examined, and performance degrades rapidly. Interestingly, a great deal of this search is spent examining bins in which only a small fraction of their volume could possibly supply the nearest neighbour (see Figure 3). If we are willing to settle for an *approximate* NN, then we can avoid prolonged search by limiting the number of leaf nodes we are willing to examine (to E_{max}), and settling for the best neighbour found up to that point.

This modification extends the domain of k-d trees for fast NN recovery a small amount. However, the backtracking search described above is still inefficient because the order of examining leaf nodes is according to the tree structure, which depends only on the stored points, and does not take into account the position of the query point. A simple idea for a more optimal search is to look in bins in order of increasing distance from the query point. The distance to a bin is defined to be the minimum distance between q and any point on the bin boundary.

Such a search can be easily implemented with a small amount of overhead using a priority queue. During NN lookup, when a decision is made at an internal node to

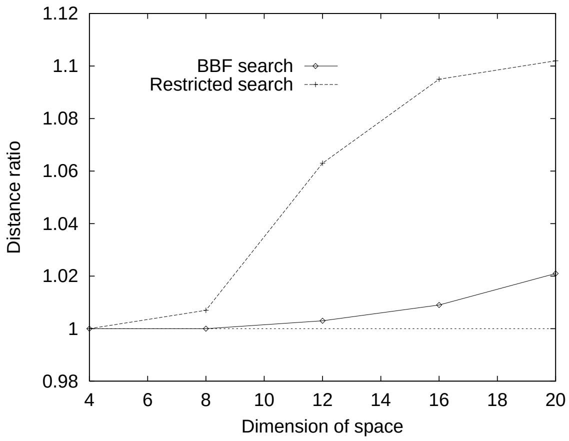

Figure 5. Approximate NN lookup vs dimension of space: Average of ratios of approximate to actual NN distance. (Uniform distribution; $100,000$ stored points; $E_{max}(BBF)$=200; $E_{max}(restricted)$=480; averaged over 1000 queries.)

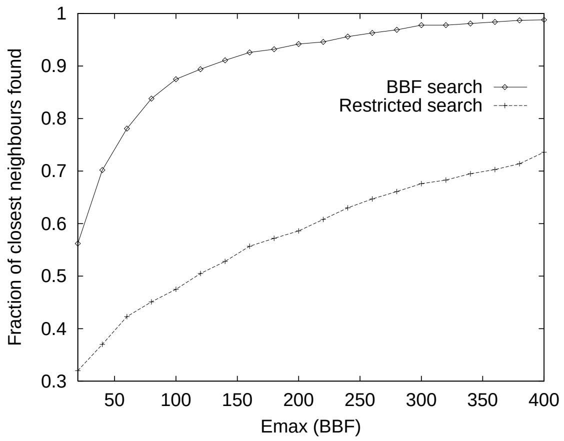

Figure 6. Approximate NN lookup vs E_{max}. (Uniform distribution; $100,000$ stored points; dimension of space$=$ 12; averaged over 1000 queries. $E_{max}(restricted)$ was 2.4 times E_{max}(BBF).)

branch in one direction, an entry is added to the priority queue to hold information about the option not taken. This includes the current tree position and the distance of the query point from the node. After a leaf node has been examined, the top entry in the priority queue is removed and used to continue the search at the branch containing the next closest bin.

This *Best Bin First* (BBF) search strategy provides a dramatic improvement in the NN search for moderate dimensionality (e.g. 8-15), making indexing in these regimes practical. For fairly small values of E_{max}, the method discovers the exact NN a large percentage of the time, and a very close neighbour in the remaining cases. The next section provides experimental evidence to support this claim.

5. Experimental results

5.1. Uniform distribution

We perform experiments with points randomly drawn from a uniform distribution in the unit hypercube, and present results for the 1-NN problem, which are indicative of the performance of the methods for k-NN lookup. Rather than providing complete coverage of the three variables of interest (dimension of index space D, the number of stored points N, and E_{max}), in each experiment two of the three are fixed and the other allowed to vary. In each case, changing the fixed values only changes the absolute numbers on the curves, while the trends remain the same as those shown. The values of the parameters when fixed are $D = 12$, $N = 10^5$, and $E_{max} = 200$. This point is on all curves, and we believe it provides a challenging scenario for current indexing systems.

We refer to the standard k-d tree search as "exact" search, and define "restricted" search to be standard k-d tree search terminated after examining at most E_{max} leaves. Because of the overhead involved in keeping the priority queue for BBF search, it would be unfair to make a straight comparison with the restricted search. Based on timing experiments which covered the entire range of parameters used below, the number of leaves that standard k-d tree search is able to visit in the same time that BBF completes its search is a factor of between 1.8 and 2.4 times the number that BBF sees. Therefore, we adopt the conservative approach of allowing restricted search to visit 2.4 times the number of leaves that BBF does in the experiments that follow.

Figure 4 sketches the performance of approximate search with respect to the dimension of the space. Restricted search becomes poor quickly after D=8 whereas BBF degrades smoothly. In some sense this curve is the most important to indicate that indexing in moderate dimensionalities is practical. In 12 dimensions, for example, BBF recovers the closest neighbour 94% of the time (versus 59% for restricted search) while on average examining only 200 of the $100,000$ leaf nodes. For this same case, the "exact" search has to examine over 2400 leaves.

Figure 5 shows that even when the method does not discover the exact nearest neighbour, it does not fail by much. Even up to dimension 20, the average distance of recovered neighbours is *only* 2% *greater* than the true NN distances. This is important when considering our method of indexing (Section 3), which uses the distance-weighted support of several neighbours, recovered with k-NN search. Even if the exact NN is not among this set, it is likely that some other close neighbours, corresponding to other nearby training views of the same object, will contribute to the correct classification (i.e. match hypothesis).

Figure 6 gives performance with respect to E_{max}. For this graph, the x-axis values are for BBF only, and the val-

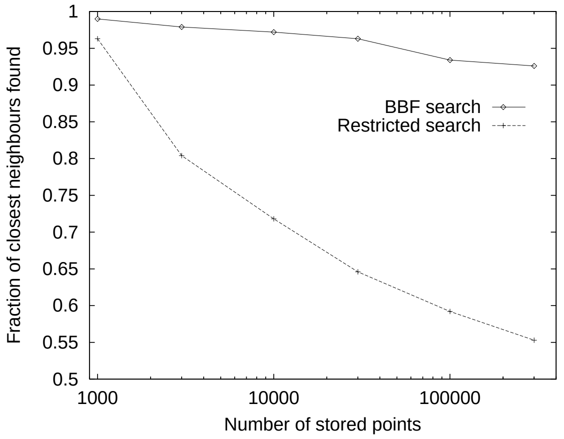

Figure 7. Approximate NN lookup vs Number of Stored Points. (Uniform distribution; $E_{max}(BBF)$=200; $E_{max}(restricted)$=480; dimension of space= 12; averaged over 1000 queries.)

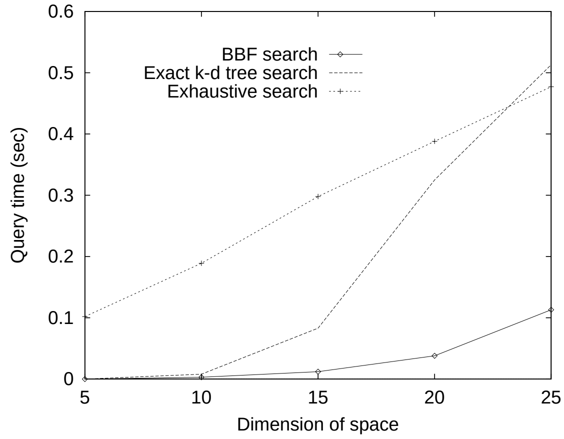

Figure 8. Timing comparison of NN lookup methods. (Uniform distribution; 30,000 stored points; $E_{max}(BBF)$ set to recover 95% of exact neighbours; Averaged over 1000 queries.)

ues for restricted search are those for BBF multiplied by 2.4. For reasonable values of $E_{max}(BBF)$ (150-400) the closest neighbour is found by BBF more than 90% of the time.

Figure 7 shows that performance of BBF varies remarkably slowly with the number of stored points. In this $12D$ space, the algorithm uncovers better than 92% of the exact neighbours for up to $300,000$ points. In the latter case, BBF examines only one out of every 1500 points, as compared to one out of every 100 points for exact k-d tree search.

In order to demonstrate that the overhead involved in BBF search is not large, we performed some timing experiments in which we compared the algorithm both to exhaustive search and to the exact k-d tree search algorithm (Figure 8). E_{max} was set to recover 95% of the exact neighbours, determined by running the experiment for several values of E_{max} at each dimensionality. In dimension 10, for

$\frac{\#bins}{dim}$	3	4	5
Fraction NN found	0.318	0.177	0.072

Table 1. Hash table lookup: fraction of nearest neighbours found in "single bin" access. (Uniform distribution; 62,536 stored points; dimension of space = 8; averaged over 1000 queries.)

example, BBF provides speedup by a factor of 60 over exhaustive search, and in dimension 20 it is still an order of magnitude faster.

5.2. Hash table lookup

While the above experiments demonstrate the advantages of BBF search over restricted k-d tree search, it is important to note that k-d trees are already a significant improvement over hash table lookup. In this section we compare with the method used in [3, 18, 19], in which only a single hash bin is accessed. This can be considered another form of approximate NN search, but one which performs very poorly in high-dimensional spaces. More extensive hash searches that would perform better are infeasible: searching to either side of the query bin in each dimension would mean accessing 3^k bins; and the computations required to do a more intelligent search, similar to BBF (i.e. to determine the next closest hash bin to check during a search of adjacent bins) are too expensive.

For this "single bin" hash lookup we have chosen a situation quite favorable to the method: the dimension of the space is not extreme ($8D$); the distribution of points is uniform (best case for hashing), in the unit hypercube; and the bin sizes are relatively large (as suggested by [3, 19]). The number of points ($4^8 = 62,536$) is chosen to give $1\frac{point}{bin}$ in the moderate-density regime, when $\frac{\#bins}{dim} = 4$. This leads to $10\frac{points}{bin}$ when $\frac{\#bins}{dim} = 3$ ("high-density"), and to $0.17\frac{points}{bin}$ when $\frac{\#bins}{dim} = 5$ ("low-density"). Table 1 shows that, no matter how coarsely the bins are quantized, the method performs poorly. As a comparison, for the same problem BBF requires $E_{max} = 57$ to recover an average of 95% of the exact neighbours.

As the dimensionality increases, the numbers for hashing become even worse very quickly, for two reasons. First, there is the inevitable decline due to the fact that a larger fraction of the points become potential nearest neighbours. Secondly, unless the number of points is increased exponentially with dimension, even with the coarsest possible quantization ($2\frac{bins}{dim}$) the average bin will be very unlikely to contain any points. Then the performance will look more like the low-density entry of Table 1 than either of the others.

5.3. Synthetic model database

We also performed an experiment using non-uniform data, provided by our database of object models (Figure 9).

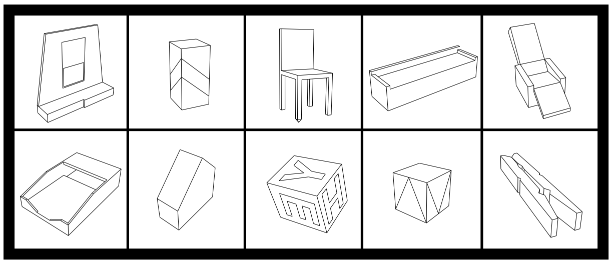

Figure 9. Database of 10 models.

From each of the 10 models we extracted $10,000$ feature vectors by taking synthetic images from random viewpoints, which required an average of 103 images per object. The feature groupings we used were 6-segment chains, which provided a $10D$ feature space consisting of 5 angles and 5 edge-length ratios.

Recall from Section 3 that in our indexing method, rather than using a single NN, hypotheses are formed using a distance-weighted combination of a small number of recovered neighbours, in this case 10. Hypotheses are then sorted into a list, based on the weightings. In this experiment we compare the "approximate" lists generated by BBF with very small E_{max} (50), to the "true" lists as determined by the exact algorithm. We averaged results over 1000 test images, each one comprised of a single database model chosen at random, and viewed from a random viewpoint. The average length of a "true list" generated from an image was 93 hypotheses.

The speedup from using BBF over exact k-d tree search was a factor of 14. Even though only 50 of the $100,000$ stored points were being examined, BBF missed less than 4% of the hypotheses, and even more significantly, the average ranking of the first missing hypothesis was 80^{th} out of 93. As well, the average difference in the rank of a hypothesis between the lists was less than 2 positions. As mentioned earlier, these results are partially due to the fact that most of the good hypotheses have the support of a few neighbours (which correspond to the same grouping as seen from nearby viewpoints in the training set), so if only one of these is missing, indexing is still able to succeed.

6. Summary and conclusions

We have argued for the use of k-d trees in shape-based indexing. To this end, we presented an efficient, approximate nearest neighbour algorithm which makes indexing in high-dimensional spaces practical. High-dimensional feature spaces provide a degree of discrimination that is essential for large model databases, which is the natural domain of application for indexing techniques. Our experiments demonstrated that BBF search is capable of finding close neighbours over a wide range of dimensions and for very large numbers of stored points, by examining only a small fraction of the points. We also presented some results indicating that the hash table method most commonly used for indexing breaks down in higher dimensions.

The BBF method may also be applicable to other areas such as appearance-based recognition. There, objects are represented as low-dimensional manifolds embedded in high-dimensional spaces, and stored as point sets sampled from the manifolds. Object classification is achieved using NN lookup in the large-dimensional spaces, and BBF appears to be more efficient than the methods currently used for this task (e.g. [15]).

Acknowledgements

Three of the database models were from http://www.eecs.wsu.edu /IRL/3DDB/Models: thanks to Pat Flynn for the repository and USF vision research group for the models.

References

[1] S. Arya. Nearest neighbor searching and applications. Technical Report CAR-TR-777, Center for Automation Research, University of Maryland, June 1995.

[2] J. Beis and D. Lowe. Learning indexing functions for 3-d model-based object recognition. In *Proceedings CVPR '94*, pages 275–280, Seattle, Washington, June 1994.

[3] A. Califano and R. Mohan. Multidimensional indexing for recognizing visual shapes. *IEEE Trans. PAMI*, 16(4):373–392, 1994.

[4] D. Clemens and D. Jacobs. Space and time bounds on indexing 3-d models from 2-d images. *IEEE Trans. PAMI*, 13(10):1007–1017, 1991.

[5] D. Forsyth, J. Mundy, A. Zisserman, and C. Brown. Invariance - a new framework for vision. In *Proceedings ICCV '90*, pages 598–605, 1990.

[6] J. Friedman, J. Bentley, and R. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Software*, 3:209–226, 1977.

[7] W. Grimson and D. Huttenlocher. On the sensitivity of geometric hashing. In *Proceedings 3rd ICCV*, pages 334–338, 1990.

[8] B. Kim and S. Park. A fast k nearest neighbor finding algorithm based on the ordered partition. *IEEE Trans. PAMI*, PAMI-8(6):761–766, 1986.

[9] Y. Lamdan, J. Schwartz, and H. Wolfson. Affine invariant model-based object recognition. *IEEE Trans. Rob. Aut.*, 6(5):578–589, 1990.

[10] Y. Lamdan and H. Wolfson. Geometric hashing: a general and efficient model-based recognition scheme. In *Proceedings ICCV '88*, pages 238–249, 1988.

[11] D. Lowe. *Perceptual Organization and Visual Recognition*. Kluwer Academic, Hingham, MA, 1985.

[12] D. Lowe. Fitting parametrized three-dimensional models to images. *IEEE Trans. PAMI*, 13(5):441–450, 1991.

[13] L. Miclet and M. Dabouz. Approximative fast nearest neighbor recognition. *Pattern Recognition Letters*, 1:277–285, 1983.

[14] H. Neimann and G. Goppert. An efficient branch-and-bound nearest neighbour classifier. *Pattern Recognition Letters*, 7:67–72, 1988.

[15] S. Nene and S. Nayar. Closest point search in high dimensions. In *Proceedings CVPR '96*, pages 859–865, 1996.

[16] C. Rothwell, A. Zisserman, J. Mundy, and D. Forsyth. Efficient model library access by projectively invariant indexing functions. In *Proceedings CVPR '92*, pages 109–114, 1992.

[17] R. Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6:579–589, 1991.

[18] F. Stein and G. Medioni. Structural indexing: efficient 2-d object recognition. *IEEE Trans. PAMI*, 14(12):1198–1204, 1992.

[19] F. Stein and G. Medioni. Structural indexing: efficient 3-d object recognition. *IEEE Trans. PAMI*, 14(2):125–145, 1992.

[20] M. Wheeler and K. Ikeuchi. Sensor modelling, probabilistic hypothesis generation, and robust localization for object recognition. *IEEE Trans. PAMI*, 17(3):252–265, 1995.