Interval hash tree: An efficient index structure for searching object queries in large image databases

T. F. Syeda-Mahmood, P. Raghavan, N. Megiddo
IBM Almaden Research Center
650 Harry Road, San Jose, CA 95120
stf,raghavan,megiddo@almaden.ibm.com

Abstract

As image databases grow large in size, index structures for fast navigation become important. In particular, when the goal is to locate object queries in image databases under changes in pose, occlusions and spurious data, traditional index structures used in databases become unsuitable. This paper presents a novel index structure called the interval hash tree, for locating multi-region object queries in image databases. The utility of the index structure is demonstrated for query localization in a large image database.

1. Introduction

As image databases grow large in size, index structures for fast navigation become important. In particular, when the goal is to robustly locate object queries under changes in imaging conditions, efficient index structures are needed that can consolidate information within and across images in a compact and efficiently searchable form to avoid a linear search of the database. This problem is often recognized but has not been explicitly addressed. Current approaches rely on conventional spatial access structures such as R-trees[4], R*-trees[1], R+-trees[8], SR-trees[5], SS-tree[11],K-D-B trees[7], etc. For example, R-trees and their variants, assume the object-containing region in the image can be enclosed by bounding rectangles and focus on efficient grouping of the enclosing rectangles at successive levels to yield a balanced search tree that is optimized for disk accesses. The search for object queries in such index structures involves finding all database rectangles that overlap a query rectangle. Under changes in pose, the bounding rectangles undergo considerable change including translation, so that the search using these index structures no longer yields a match to the query. Moreover, when the object query consists of multiple regions, the spatial layout of regions is not adequately taken into account through bounding rectangles. Hence traditional index structures are not directly applicable for the general object localization problem unless the spatial layout of regions can be modeled.

In this paper, we present a novel geometric index structure called the interval hash tree for locating multi-region object queries in image databases. Specifically, the interval hash tree (IHT) is a two-way interval tree in which the layout of regions in images of a database is represented in an affine-invariant fashion through affine intervals computed with respect to chosen basis features. Localizing object queries involves representing the query affine intervals also as an interval hash tree, and finding database intervals that overlap with the affine intervals represented at each node of the query IHT through careful tree exploration. The overall goal of such indexing is to accumulate evidence for the most common occurrence of basis features.

The interval hash tree has several desirable properties. First, by representing object layout information using affine intervals rather than bounding rectangles, the localization is made affine-invariant. Secondly, by building sufficient redundancy in the representation of affine intervals, tolerance to occlusions and background clutter is achieved. Third, using a balanced binary search tree based on median-based partitioning of affine space, the search for overlapping intervals for a single query interval is made efficient. Finally, by consolidating the search for query affine intervals through query IHT, repeated exploration of database IHT can be avoided.

2. Localizing objects through region hashing

To motivate the need for IHT, we briefly review the technique of representing and recognizing objects.

It is well-known that the shape of a 2d object can be described in an affine-invariant fashion by recording the affine coordinates of features within object computed with respect to a triple of basis features chosen as an object-based reference frame[6]. A simpler yet effective way of describing their relative location is by recording the affine interval, i.e., the 2d-interval in which affine coordinates of features lie w.r.t to the chosen basis triple. Since such intervals bound the affine coordinates, they are also affine-invariant. The uniqueness of the affine intervals is not guaranteed though, since two different distributions of affine coordinates could be bound by the same interval. However, the chance of this can be minimized if we accumulate evidence from multiple object region pairs. Thus one way of recognizing or localizing an object is to hash for, i.e., find evidence for as many common affine intervals between the query region pairs and region pairs in candidate images, in a manner similar to geometric hashing[6] and is termed *region hashing*. In practice, to

0-7695-0695-X/00 $10.00 (c) 2000 IEEE

account for occlusions and region segmentation errors that shrink the image affine intervals to become a subset of the corresponding object affine intervals, region hashing actually looks for overlap rather than exact registration of affine intervals.

In summary, to localize a multi-region query object using region hashing, we represent the spatial layout of region pairs on query object as well as images of the database through affine intervals, and look for as many database affine intervals that overlap with query affine intervals. For each such common interval found, the count of the associated image basis feature set can be updated. At the end of this process, the top hit image basis features and their associated enclosing image regions can be declared as corresponding to query basis feature set and the associated query region pairs, thus localizing the query object.

3. Interval Hash Trees

From the above formulation, it is clear that the most computationally intensive part of object localization is the process of determining database intervals that overlap with query affine intervals. We now present an index structure called the interval hash tree to address the above problem. An interval hash tree(IHT) is a two-dimensional interval tree. It extends the concept of 1d interval trees known in computational geometry [2, 3] to organizing a set of 2d intervals as balanced binary search trees. Specifically, it uses the first coordinate of intervals, namely, the x coordinate and organizes the x-end points of intervals as a regular interval tree (called x-interval tree)[1]. The central branch of the this tree at each node is modified to be another interval tree, now on the second coordinate, i.e., the y-coordinate (called a y-interval tree).

To construct an interval hash tree, affine intervals are sorted based on the first coordinate (i.e. the α coordinate). Let the sorted set of intervals be denoted by I. Let Xmid be the median end point in set I. We split the interval I into three subsets, namely, I_{xleft}, I_{xright}, and I_{xmid}, to correspond to the set of intervals whose left and right x-end points are to left of $Xmid$, to the right of $Xmid$, and whose left x-end point is to the left and right x-end points are to the right of $Xmid$ respectively. A bounding rectangle for the affine coordinates spanned by the set I can additionally be included for ease of searching at the node. The interval I_{xmid} then forms the central branch of a node of the x-interval tree and the I_{xleft} and I_{xright} form left and right subtrees. Each of the left and right subtrees are recursively constructed in a similar manner by repeatedly picking their median x-end point and splitting into the three subsets explained above. The central branch of each node of the x-interval tree is similarly organized as a y-interval tree using the y-end points of the set I_{xmid}. The corresponding set Ly_{below} to I_{xleft} is sorted in increasing order while Ly_{above} is sorted in decreasing order to enable range searching. Note that unlike in the case of non-leaf nodes of the x-interval tree, the data associated with affine intervals is listed under the central branch of a node in the y-interval tree. In the case of region hashing, this data constitutes the image index, the region index, and the basis triple index, necessary for query localization.

Figure 1c depicts an interval hash tree for a set of affine intervals. These intervals are obtained by pairing region marked 1 on the object depicted in Figure 1a w.r.t. to all other marked regions on the object. The affine intervals themselves are indicated in Figure 1b. Here xlij,xrij stand for the left and right end points of the affine interval of region j computed w.r.t. a basis triple in region i (in this case, the middle basis triple on region 1's contour). The empty branches of the tree are denoted by circles and single interval branches by the interval itself. Although the lists Ixmid and Iymid are indicated at the nodes for purpose of illustration, only the median point Xmid (Ymid) and the bounding rectangle of the intervals under a tree node are stored at each node For a leaf node, this reduces to the affine interval itself.

3.1. Searching Interval hash trees

We now address the problem of searching for all database intervals that overlap with a given set of query intervals. The basic idea is to organize the affine intervals of the query also as an interval hash tree, and perform a careful simultaneous search of database and query IHT in a way that maintains the property that a database interval that overlaps more than one query interval is discovered only once. To perform this search, we use a version of the IHT construction that retains the bounding box of all affine intervals under a tree node (x or y-tree node) as part of the node information. Using this additional piece of information and a pre-order traversal at both the x and y query IHT levels, nodes of the query IHT are successively searched against the nodes of the database IHT. If we denote the x and y-interval trees at the current query node by V_{qx} and V_{qy}), then the order of exploration is $V_{qx} -> \{V_{qy}, Lcy(V_{qy}), Rcy(V_{qy})\}$ followed by $Lcx(V_{qx})$ and $Rcx(V_{qx})$. For each such exploration against a database IHT node V_d, we use the bounding rectangle range (Qlx, Qrx, Qby, Qay) to determine which of the subtrees to explore. For example, if $Qlx < X_{mid}(V_{xd}) < Qly$ then matching intervals can be found in either the y-interval tree under V_{xd} i.e. V_{yd}, or the left and right subtrees $Lcx(V_{xd})$ and $Rcx(V_d)$ of V_{xd}. Specifically, the algorithm uses the range (Qby, Qay), and the relative placement of the median point of query y-interval tree $Ymid(V_{qy})$ w.r.t the median point $Ymid(V_{yd})$ of the database node to decide the order of exploration. For example if $Qby \leq Ymid(V_{yd}) \leq Qay$, and $Iymid(Vqy) > Iymid(Vyd)$, then by walking along $Ly_below(Vqy)$ and $ly_above(V_{yd})$ from their top-most end point, we can report all database intervals that contain the current query end point.

[1] For ease of notation we refer to the coordinates as x and y coordinates. In the case of affine intervals for region hashing, the x-coordinate corresponds to the α coordinate, and the y-coordinate corresponds to the β coordinate. For other applications there may be other coordinate interpretations.

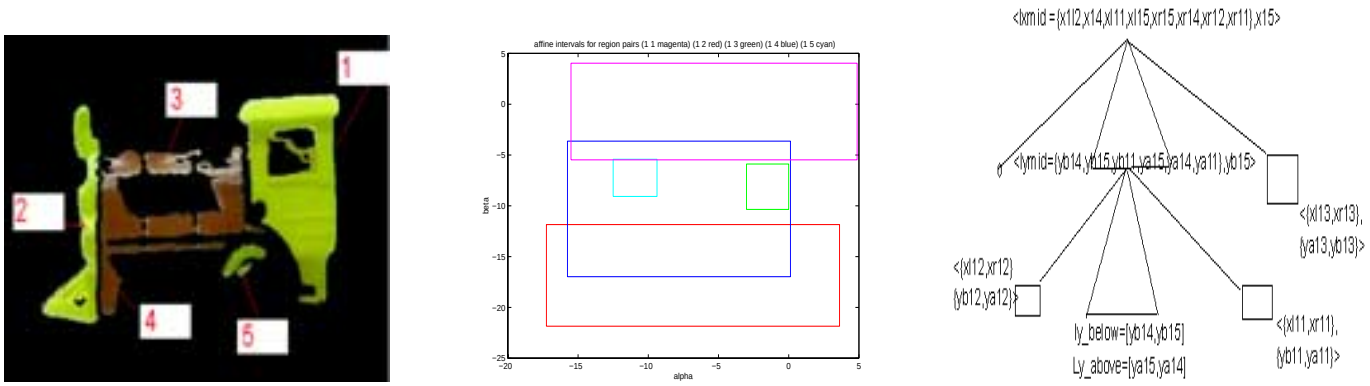

Figure 1: Illustration of affine intervalsand their corresponding interval hash tree. (a) regions on an example object (b) The affine intervals of the spatial layout of all regions w.r.t. region 1 using the middle basis triple from those on the boundary curve of region 1. (c) Interval hash tree for the affine intervals in (b).

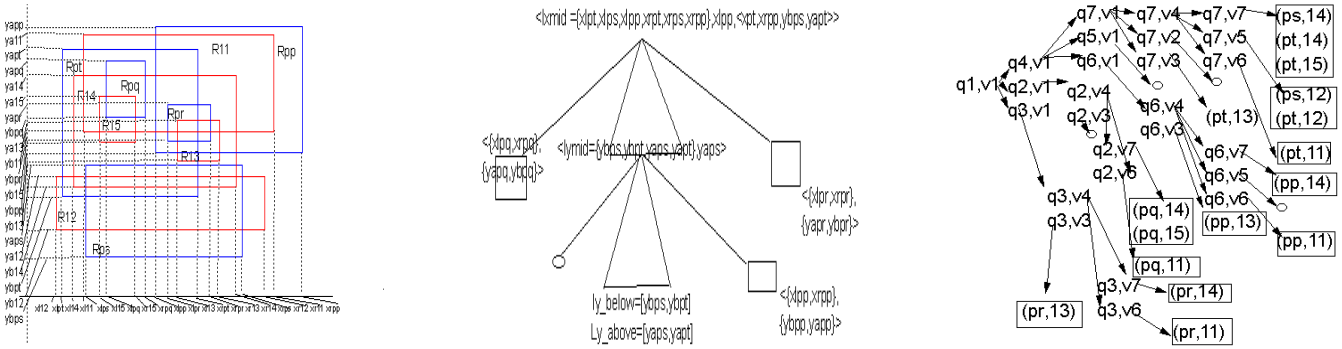

Figure 2: Illustration of IHT search using a set of query affine intervals. (a) The affine intervals of database shown in red. Affine intervals of a sample query object are shown in blue. The orthogonal projections used to form the query interval hash tree are labeled. (b) The interval hash tree of the query in (a). The database IHT in this example is the one shown in figure 1c.

S.No.	Query regions	Query intervals	Percentage database touched		
			affine intervals	IHT nodes	Images
1.	5	200	14.5	12.3	2.3
2.	6	144	13.9	23.4	4.3
3.	4	64	13.3	15.2	1.2
4.	9	324	25.4	34.5	5.7
5.	7	196	17.9	16.3	9.8
6.	10	400	27.4	44.2	15.4

Table 1: Illustration of effectiveness of indexing using IHT. The complexity of the query is indicated by the number of query regions. Here four basis features per query region were used to generate the query affine intervals. The fraction of the database touched is shown in terms of affine intervals touched, the IHT nodes actually visited and the images indexed.

In addition, we can report, without further search, this overlap to hold for all query intervals in V_{xq} whose bottom-end points are below the current query end point. *This critical step avoids the repetitious search for multiple query intervals.*

We illustrate IHT search through an example in Figure 2. The database IHT is shown in Figure 1c. The corresponding affine intervals are redrawn in red in Figure 2a. For searching in this tree, we select a query whose affine intervals are as shown in Figure 2a colored in blue. The corresponding query IHT is shown in Figure 2b. A node number is assigned to each node in the two trees to illustrate the order of navigation. The order in which nodes will be examined and the overlap discovered is shown in Figure 2c. We encourage the reader to walk through this example using the algorithm to confirm the result. Note that the left or the right subtree branches are left out during some recursive traversals (eg. (q2,v7) (q2,v6) but not (q2,v5)). Also note that the overlap of region-pair (pt,14) is automatically derived from the overlap of region pair (ps,14) without explicit search. Note also that all overlapping intervals are found by this process.

3.2 Region hashing using IHT

Since the purpose of designing the IHT was to demonstrate its use for query localization, we evaluated the indexing performance of IHT for use in region hashing. The experiments were conducted on a database of images assembled from multiple sources. These include: the MPEG-7 test data set (4000 images), the HIPS dataset from University of Michigan (200 images), the COREL database (500 images), the Kodak PhotoCD database (500 images), and a home-brewed dataset (400 images) assembled for 2d and 3d objects depicted in the presence of occlusion and clutter under indoor florescent and tungsten light sources. Some of the database images were obtained by placing objects from one collection (eg. HIPS collection) in different settings to generate images. Each of the images of the database, and the query objects underwent similar image processing and feature extraction to generate the affine intervals needed for IHT. Specifically, regions in images of the database and queries were extracted using color information. We adopted an approach based on surface color classes to detect and recognize color regions as described in[10, 9]. Next, images are processed using an edge detector to form curves, which are segmented at zero crossings to generate corner features. The enclosing region information is associated with the corners. Triples of adjacent corner features are used to form basis sets (using both directions of curve ordering). For each distinct pair of regions on the object, affine coordinates of corner features on curves on the boundaries of one region are computed w.r.t each basis triple on the other region. The affine intervals are used to construct a database IHT. Similar processing is done on the query features to generate a query IHT. Each node of the query IHT was used to search for overlapping intervals in the database IHT as indicated in Section . A histogram of the indexed basis triples is taken and the maxima noted. The basis triples, and their associated images corresponding to the peaks of the basis histogram are taken as potential locations where the query object is present.

4. Results

We now report on the indexing performance of IHT in comparison with region hashing without an index structure. First, we show that the use of index structure results in a small fraction of the database being examined as compared to linear search. Next, we show that the indexing time performance is not significantly impacted by paging issues because the size of overall structure for most database sizes, is small enough to fit in main memory.

Effect of indexing using IHT

To evaluate the effect of indexing, we searched for a total of 200 object queries against the 5600 image database and noted the fraction of the database touched by indexing. The fraction of database touched was noted using three parameters, namely, the percentage of affine intervals that were found to overlap, the percentage of IHT nodes touched, and the percentage of images indexed. The result for a few sample queries is listed in Table 1. Here the query complexity is indicated by its number of regions. Also, four basis features were used to generate the query affine intervals. As can be seen from Column 6 of this table, only a small fraction of the images are touched as against linear search that would examine all the images for possible matches. Also, from Column 4, we notice that the fraction of affine intervals touched though larger than the number of images touched, is still small indicating that only a small number of image locations are examined as potential candidates for containing the query object. Finally, the number of IHT nodes visited remains less than 50% indicating the worst-case search scenario in IHT is rarely reached.

Effect of indexing on precision and recall

Since fewer database images are touched one can expect that the precision using IHT indexing would be higher. False positives may still be present among the indexed images since the discovery of overlapping affine intervals does not guarantee the evidence for an object. As for recall, we note that since all overlapping database intervals are found during search, there should be no false negatives in theory. In practice, however, depending on the manner in which affine intervals are computed (using one or more basis triples per region pair), the database affine intervals may not come up as a match to query affine intervals if there are no corresponding basis features. False negatives are also possible since the region hashing technique ranks the indexed images w.r.t their relevance to the query, and can prune the matches returned by indexing. To evaluate the effect of IHT indexing on the precision and recall of

0-7695-0695-X/00 $10.00 (c) 2000 IEEE

Query	Query Occurrences	Occurences in top 20 pairs
1.	17	14
2.	8	5
3.	11	7
4.	16	10
5.	13	9
6.	2	2

Table 2: Illustration of precision and recall using IHT in region hashing.

Database size	Avg. # regions	affine intervals	IHT nodes
100	13.6	21496	1849
500	17.8	88420	6752
1000	19.56	362454	9122
2000	24.3	980980	12239
5600	26.5	3932600	48315

Table 3: Illustration of storage size of IHT.

region hashing, we noted the number of query occurrences in the database for each query tested. We also recorded the number of indexed occurrences using IHT in the top 20 matches declared. The result for a few sample queries is indicated in Table 2. As can be seen, the use of IHT does not adversely affect the precision and recall performance of region hashing.

Storage performance

To estimate the amount of storage used by the index structure, we gradually increased the size of our database from 100 images onwards until the entire 5600 image database was populated in the IHT. The result is shown in Table 3. As can be seen, the number of affine intervals grows quadratically with the number of regions in images, and linearly with the number of images. The number of IHT nodes remains a few Mbytes, a size small enough to avoid excessive paging problems relating to I/O. As database size becomes increasingly larger, however, we expect it to face paging problems.

Indexing time performance

Even though fewer images are indexed using IHT, this would not be an advantageunless the actual time performance of IHT indexing was good as well. As mentioned above, the size of the interval hash tree for even the large collection of images we tested was a few megabytes, making it possible to index images without any paging-related I/O problems. The average indexing time using a C/C++ implementation on a Windows NT platform (200 Mhz CPU and 250Mbytes paging size) involved processing of the query features which was about 0.5 second, followed by the retrieval which was about 1.2 seconds.

5. Conclusions

In this paper we have presented a novel index structure called the interval hash tree for localizing 2d object queries in image databases. We have shown by using this index structure, objects can be localized under changes in pose, occlusions and in the presence of scene clutter. We have shown that the number of images that are searched is dramatically reduced by the use of interval hash trees.

References

[1] N. Beckmann et al. The r*-tree: An efficient and robust access method for points and rectangles. In *Proc. ACM SIGMOD*, pages 322–331, 1990.

[2] T.H. Cormen, C.E. Lieserson, and R.L. Rivest. *Introduction to Algorithms* New York: McGraw Hill, Cambridge: MIT Press, 1990.

[3] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry.* Berlin: Springer Verlag, 1997.

[4] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD*, pages 47–57, 1984.

[5] N. Katayama and S. Satoh. The sr tree: An index structure for high-dimensional nearest neighbor queries. In *Proc. ACM SIGMOD*, pages 369–380, 1997.

[6] Y. Lamdan and H.J. Wolfson. Geometric hashing: A general and efficient model-based recognition scheme. In *Proceedings of the International Conference on Computer Vision*, pages 218–249, 1988.

[7] J.T. Robinson. The k-d-b-tree: A search structure for large multidimensional dynamic indexes. In *Proc. ACM SIGMOD*, pages 10–18, 1981.

[8] T. Sellis, N. Roussopoulos, and C. Faloutsos. The r^+ -tree: A dynamic index for multi-dimensional objects. In *Proceedings of Conf. on Very Large Databases*, pages 507–518, 1987.

[9] T. Syeda-Mahmood. Indexing of topics using foils. In *IEEE Conf. on Computer Vision and Pattern Recognition*, 2000.

[10] T.F. Syeda-Mahmood and Y-Q. Cheng. Indexing colored surfaces in images. In *Proceedings Int. Conf. on Pattern Recognition*, 1996.

[11] D.A. White and R. Jain. Similarity indexing with the ss-tree. In *Proc. 12th Intl. Conf. on Data Engineering*, pages 516–523, 1996.