



ELSEVIER

Data & Knowledge Engineering 32 (2000) 271–289

**DATA &
KNOWLEDGE
ENGINEERING**

www.elsevier.com/locate/datak

Indexing shapes in image databases using the centroid–radii model

Kian-Lee Tan *, Beng Chin Ooi, Lay Foo Thiang

Department of Computer Science, School of Computing, National University of Singapore, Lower Kent Ridge Road, Singapore 119260, Singapore

Received 14 April 1999

Abstract

In content-based image retrieval systems, the content of an image such as color, shapes and textures are used to retrieve images that are similar to a query image. Most of the existing work focus on the retrieval effectiveness of using content for retrieval, i.e., study the accuracy (in terms of recall and precision) of using different representations of content. In this paper, we address the issue of retrieval efficiency, i.e., study the speed of retrieval, since a slow system is not useful for large image databases. In particular, we look at using the shape feature as the content of an image, and employ the centroid–radii model to represent the shape feature of objects in an image. This facilitates multi-resolution and similarity retrievals. Furthermore, using the model, the shape of an object can be transformed into a point in a high-dimensional data space. We can thus employ any existing high-dimensional point index as an index to speed up the retrieval of images. We propose a multi-level R-tree index, called the Nested R-trees (NR-trees) and compare its performance with that of the R-tree. Our experimental study shows that NR-trees can reduce the retrieval time significantly compared to R-tree, and facilitate similarity retrieval. We note that our NR-trees can also be used to index high-dimensional point data commonly found in many other applications. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Content-based retrievals; Shape feature; Image database; Retrieval effectiveness; Retrieval efficiency; NR-trees

1. Introduction

With the advances in multi-media technologies (such as compression, display, database and visualization technologies), and the increasing emphasis on multi-media applications, the production of image and video information has resulted in large volumes of images and video clips. This trend is likely to continue, and to cope effectively with the explosion of multi-media information, image-based systems have been proposed to properly organize and manage these information for rapid retrieval [3,6,8,9,13–16,23,26,31,33,42,43,45,46].

* Tel.: +65-874-2862; fax: +65-779-4580.

E-mail addresses: tankl@comp.nus.edu.sg (K.-L. Tan), ooibc@comp.nus.edu.sg (B.C. Ooi), thiangla@comp.nus.edu.sg (L.F. Thiang).

At the same time, users of image retrieval systems are demanding for more user friendly querying facilities. There is a clear preference to access images based on their content through queries such as “Retrieve all images that looks like *this*”, where *this* refers to a sample image provided by the user or one that the user has painted on some sketching tool. Providing such capabilities calls for content-based image retrieval system that accesses images based on the content of the image such as color, texture and shape. In a content-based image retrieval system, the content of the database images are (semi)automatically generated and stored. A query image’s content is extracted during runtime and used to match against those in the database. The result of the query is a set of images that are *similar* to the query image, rather than an exact match. Examples of content-based retrieval systems include QBIC [31], Photobook [35], Virage [1], Candid [21], Chabot [33], VisualSEEK [41], and VIPER [34].

More importantly, content-based retrieval systems are needed in many applications. We have witnessed its use in applications such as medical imaging [21,24], TV production [40], multi-media [41], art history [20], geology [39] and satellite image databases [22]. Other potential applications include a trademark system, an ultra-high resolution image supplier system, and an image classifier system. For the trademark system, a trademark can only be assigned if it does not resemble those in the database too closely. Likewise, a user may want to find out if his/her low resolution picture is available in the database of the image supplier system. Finally, such a system can also facilitate the classification of images [3].

While existing retrieval systems can handle queries about content, the technology for content-based retrieval remains an important challenge. In this paper, we focus on the shape feature. The shape feature is extremely useful for image database systems like an X-ray system or a criminal picture identification system. In an X-ray system, queries like “Retrieve all kidney X-rays with a kidney stone of this shape” are very common. For a criminal picture system, we expect queries like “Retrieve all criminals with a round face shape”. The example shape, the shape of a kidney stone in the first case, and *round* in the second, can be supplied using an example image.

Our emphasis, is however, on the retrieval efficiency of image retrieval, rather than on the retrieval effectiveness of image retrieval. In other words, we study how to quickly retrieve similar images given a query shape. This issue has been largely ignored by existing work and we believe it is very important since a slow system will not be useful or practical for large image databases. This is achieved by building indexes on the shape information to facilitate pruning of the search space. For simplicity, shape feature is captured by the centroid–radii model [7]. This model can easily support multi-resolution and similarity retrieval. Furthermore, it allows us to compactly represent a shape as a point in a multi-dimensional data space, and hence facilitates exploitation of existing point access methods to speedily retrieve images containing objects with similar shapes. We propose a novel multi-level indexing structure, called the *Nested R-trees* (NR-trees), to index shape features. We implemented the technique and compare its retrieval efficiency against the R-tree [12]. Our results demonstrate that the NR-trees structure is superior over the R-tree in query performance.

While we have applied the NR-trees structure for indexing shapes in image databases, it is clearly an index structure that can be used to index high-dimensional data that are common in many advanced database applications.

The remainder of this paper is organized as follows. In the next section, we review existing work on shape modeling and shape indexing structures. Section 3 presents the centroid–radii model that

is used in our work. In Section 4, we present the proposed indexing structure – the Nested R-trees. Section 5 presents the experimental study that evaluates the proposed scheme and reports our findings, and finally, we conclude in Section 6 with directions for future work.

2. Related work

Shape features can be represented using their boundary information by any of 16 primitive shape features (i.e., a line, an arc with a starting point, an ending point, and so on). Since each primitive feature can be encoded by a distinct character, we can compactly store a shape feature as a one-dimensional string [18]. The shape features of a shape boundary can then be represented by substrings of the one-dimensional string. This simple representation allows the exploitation of existing efficient string matching algorithms. Since objects with the same shape will be encoded in the same manner, exact string matching can be performed to identify objects with similar shapes. To speed up retrieval, the string representation can be indexed by an inverted file.

Shape boundaries can also be modeled using a set of structural components [30], which are an ordered set of interest points such as locally maximal curvature points or vertices of the polygonal approximation. A shape feature can be obtained by fixing the number of points to be used to represent the shape feature. The feature is then mapped into a point in a multi-dimensional space, where the dimension is given by the number of points used to represent the shape. The similarity measure can then be given by the Euclidean distance between pair of points in the multi-dimensional space. Any point multi-dimensional access method can be used to index the shape feature (i.e., points).

In [17], a collection of rectangles that forms a *rectangular cover* of the shape is used. Since shapes vary widely from objects to objects, the number of rectangles can be very large. To reduce the storage requirement, at most k rectangles in the cover are used to represent the shape. The k rectangles picked must capture the most important features of the shape “sequentially”, that is the k rectangles form a sequence. As each rectangle is represented by two pairs of coordinates, and there are at most k rectangles, the shape feature can be easily mapped into a point in a $4k$ -dimensional space. Thus, a multi-dimensional point access method can be readily used for indexing the shape feature. Similarity retrieval based on Euclidean distance is performed using a region search query.

Shape can also be represented based on the concept of mathematical morphology [24,29,47], which employs a primitive shape to interact with an image to extract useful information about its geometrical and topological structure. A $(2M + 1)$ vector, called the size distribution of a shape [38], can be used to store the measurements of the area of an image at different $(2M + 1)$ of them scales. The pattern spectrum [28] turns out to be a compact representation that captures the same information. The advantage of the scheme is that it is essentially invariant to rotation and translation, and can highlight differences at several scales. In [24], the pattern spectrum is first employed to capture the shape information of an image (in the domain of a tumor database). The information is then mapped into the $(2M + 1)$ vector of the size distribution so that a multi-dimensional point index can be employed to index the shape information. While similarity retrieval is essentially a nearest neighbor search, the paper also presented a distance function, *max-granulometric* distance, that guarantees no false dismissals.

Numerical vectors have also been employed to model shape. These include using the coefficients of the 2-D Discrete Fourier Transform or Discrete Wavelet Transform [27], as well as first few moments of inertia [9,10]. These techniques usually map the shape feature to multi-dimensional point access method and use the Euclidean distance for similarity retrieval. Alternatively, the shape features can be represented by the geometric properties of the image such as shape factors (for example, ratio of height to width), mesh features, the moment features and curved line features. In this case, the inverted file has been used for indexing.

The design of indexes to support high-dimensional data search is an area of active research among the database community. Work on high-dimensional data search have focused on designing efficient high-dimensional indexing structures [5]. These structures include the K-D-B tree [36], grid file [32], SR-tree [19], R-tree [12], R*-tree [2], R⁺-tree [37], TV-tree [25] and X-tree [4]. These structures, however, are not suited for multi-resolution querying. Furthermore, their performance degenerate as the number of dimensions increases.

3. Modeling shape using the centroid–radii model

In this paper, we adopt the centroid–radii model to represent the shape feature [7]. In the centroid–radii model, lengths of a shape’s radii from its centroid at regular intervals are captured as the shape’s descriptor (see Fig. 1). More formally, let θ be the regular interval (measured in degrees) between radii. Then, the number of intervals is given by $k = \lceil 360/\theta \rceil$. Furthermore, without loss of generality, suppose that the intervals are taken clockwise starting from the y-axis direction. Thus, the shape descriptor can be represented as a vector $(l_0, l_\theta, l_{2\theta}, \dots, l_{(k-1)\theta})$ where $l_{i\theta}$, $0 \leq i \leq (k-1)$, is the $(i+1)$ th radius from the centroid to the boundary of the shape. With sufficient number of radii, dissimilar shapes can be differentiated from each other.

A shape is considered similar to another if and only if the lengths of their radii at the respective angles differ by a small value, ϵ . In other words, for two shapes with descriptors $(l_{10}, l_{1\theta}, l_{12\theta}, \dots, l_{1(k-1)\theta})$ and $(l_{20}, l_{2\theta}, l_{22\theta}, \dots, l_{2(k-1)\theta})$ the shapes are considered similar if

$$|l_{1i\theta} - l_{2i\theta}| < \epsilon \quad \forall i \in [0, k-1].$$

Hence, for similarity retrieval, we can extract two *filters* from the query shape: $(l_{10} - \epsilon_1, l_{1\theta} - \epsilon_1, l_{12\theta} - \epsilon_1, \dots, l_{1(k-1)\theta} - \epsilon_1)$ and $(l_{10} + \epsilon_2, l_{1\theta} + \epsilon_2, l_{12\theta} + \epsilon_2, \dots, l_{1(k-1)\theta} + \epsilon_2)$ for some values ϵ_1 and ϵ_2 . The two filters have the same shapes as the query shape except that one is slightly larger and one is slightly smaller. If the shape boundary of a database object lies between the two

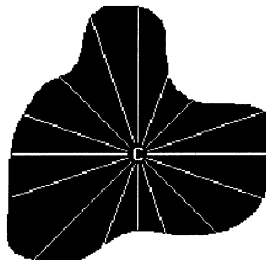


Fig. 1. The centroid–radii modeling of shape.

filters, it is considered similar. However, if a shape is dissimilar, part of its boundary falls outside the filters. Fig. 2 illustrates an example. We note that the degree of similarity can be determined by adjusting the size of the two filters.

From the above discussion, we note that the descriptor can be regarded as a multi-dimensional key. Hence, each shape can be represented as a point in a multi-dimensional space. Similarly, the 2 filters are 2 multi-dimensional keys. The region bounded by the filters has an equivalent region in the multi-dimensional space. The 2 filter keys define this equivalent region. Therefore, similarity retrieval is simply a range retrieval using the 2 filter keys on the database of multi-dimensional shape keys.

Using this model, we can also facilitate multi-resolution querying, i.e., we can conduct similarity retrieval by issuing a query shape with fewer number of radii (see Fig. 3). For example, for a database whose shape descriptors are represented using k dimensions, a $k/2$ -dimension query descriptor can be of the form $(l_0, l_{2\theta}, l_{4\theta}, \dots, l_{(k-2)\theta})$. Similarly, a $k/4$ dimension query descriptor can be of the form $(l_0, l_{4\theta}, l_{8\theta}, \dots, l_{(k-4)\theta})$. This is a less precise description of the query shape. To facilitate query retrieval with such query descriptor, the query descriptor can be mapped to a k -dimension vector where values of those unspecified dimensions are filled with “don’t-care” values. Using the same example, the $k/2$ -dimension query descriptor would be mapped into a k -dimension vector $(l_0, *, l_{2\theta}, *, l_{4\theta}, \dots, l_{(k-2)\theta}, *)$. where $*$ denotes don’t-care value. Likewise, the $k/4$ -dimension query descriptor would be mapped into a k -dimension vector $(l_0, *, *, *, l_{2\theta}, *, *, *, l_{4\theta}, \dots, l_{(k-2)\theta}, *, *, *, *)$. The query can then be evaluated as if all dimensions’ values are known. Clearly, more shapes match such query shape descriptor than when all the dimensions are specified. However, by varying the number of radii, the user can control the degree of similarity required. As we shall see, the proposed NR-trees index structure is designed to exploit this approach in an efficient manner.

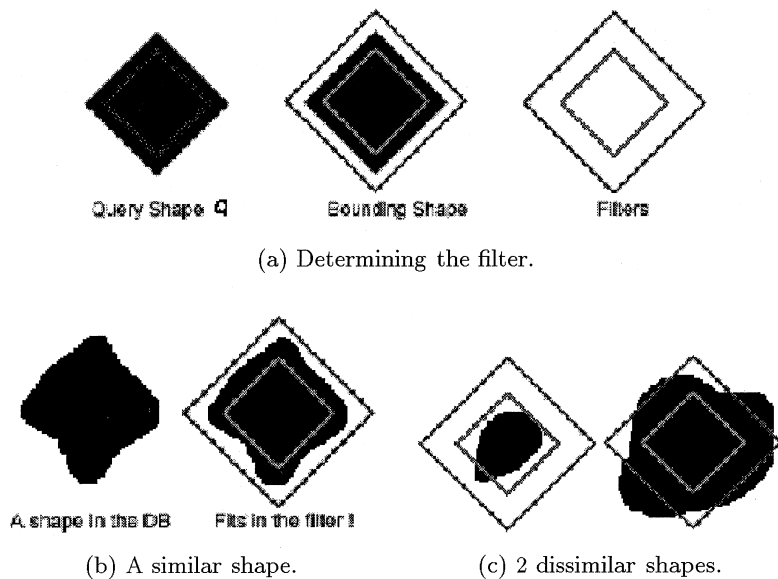


Fig. 2. Similarity retrieval in the centroid–radii model.

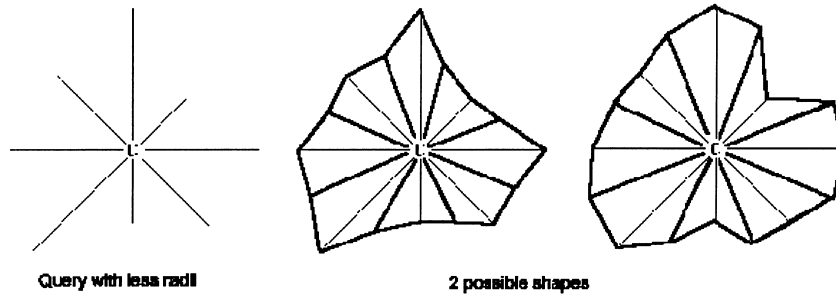


Fig. 3. Similarity retrieval method using fewer radii lengths.

4. NR-trees

In this section, we present a novel index structure for high-dimensional data. In particular, we discuss how such a structure can be exploited to index shape features.

4.1. An overview

R-tree [12] is a hierarchical index structure that recursively partitions a data space into smaller subspaces to the level where objects in an unpartitioned subspace can be stored in a physical page. An internal node has a data space that bounds the data spaces of all the subtrees. Logically, the organization is a series of nested rectangles arranged in such a way that each node corresponds to a spatial entity and contains descriptions of smaller spatial entities. The description of a spatial entity consists of a rectangle which is the *minimum bounding rectangle* (MBR) of the entity, and a pointer pointing to the spatial entity. Except for the root entity, each entity is completely contained in the entity corresponding to its parent node.

The NR-trees is a multi-layer tree structure designed to index high-dimensional data. Each layer of the NR-trees is a collection of R-trees that indexes on a subset of the dimensions. This is achieved by splitting a k -dimension search key into a set of L subkeys, each with $\lceil k/L \rceil$ dimensions. For the first $\lceil k/L \rceil$ -dimension subkey, a single R-tree (called the root R-tree or Level 1 R-tree) of $\lceil k/L \rceil$ dimensions will index it. The leaf nodes of the root R-tree contain pointers to a group of R-trees or level 2 R-trees. Level 2 R-trees will use the second subkey as the indexing attributes and the leaf nodes at this level point to Level 3 R-trees. This idea is recursively applied until there are L levels of R-trees. Therefore, multiple levels of lower dimensional R-trees can be used to service high-dimensional keys. The resulting structure is a nesting of R-trees within R-trees, thus the name Nested R-trees. Fig. 4 shows the structure of the NR-trees.

Each level of the NR-trees is a simple filter that reduces the search space quickly. By dealing with subkeys (and smaller number of dimensions), any path that differs on any subkey can be pruned away immediately. More importantly, using a smaller number of dimensions can avoid performance degeneration that arise for high-dimensional data. In addition, the NR-trees facilitates similarity retrieval. By stopping the search process at a certain level, say i , corresponds to a search on shapes that are similar on the subkey formed by concatenation of the subkeys from levels 1 to i , i.e., all records under the umbrella of influence of the leaf (reached) at level i are retrieved. In fact, the lower level the search process terminates, the more precise is the description of the shapes.

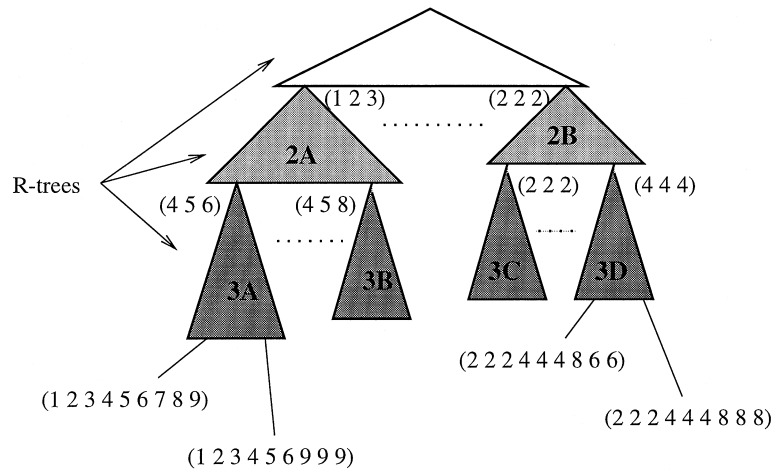


Fig. 4. An NR-tree.

We shall illustrate the operation of the NR-trees with an example as shown in Fig. 4. Suppose a shape feature is captured using a 9-dimensional point, and we would like to generate an NR-tree with 3 levels. In other words, each R-tree is a 3-dimension R-trees. To insert a record with key (1 2 3 4 5 6 7 8 9), we can traverse the Level 1 R-Tree using subkey (1 2 3) first. The leaf nodes of the root R-Tree reached point to the Level 2 R-Tree, say 2A in the figure. In a similar manner, R-Tree 2A will be traversed with subkey (4 5 6) and R-Tree 3A is hit. Hence, the record is inserted under R-Tree 3A. In fact, all records with keys of the form (1 2 3 4 5 6 * * *) are inserted under R-Tree 3A. Similarly, all keys of the form (2 2 2 * * * * *) go under the R-Tree 2B and keys (2 2 2 4 4 4 * * *) go under R-Tree 3D.

We note that multi-tier indexes have been proposed before [11,34]. However, none of these structures exploit the idea of splitting the search key to reduce the dimensionality of the search key.

4.2. Design of the NR-trees

To ensure that the NR-trees can be designed as an effective and efficient structure, we have identified several issues to be resolved. This section looks at these issues and our proposed solutions. In our discussion, we shall assume a k -dimensional search key and a L -level NR-trees.

4.2.1. Breaking down of a k -d key

Let the k -dimensional search key to be indexed be $(n_0, n_\theta, \dots, n_{(k-1)\theta})$. The search key is divided into L subkeys of length $l = \lceil k/L \rceil$ (except possibly the last subkey which may be shorter). Without loss of generality, let $(n_0, n_{L\theta}, n_{2L\theta}, \dots, n_{(k/L-1)L\theta})$ form the first subkey, $(n_\theta, n_{(L+1)\theta}, \dots, n_{(k/L-1)L+1\theta})$ be the second subkey, and so on.¹ Picking the optimal value of L is an important issue. A small L implies fewer levels of R-trees, but each level of R-trees must be built on a higher

¹ We note that different sequence of dimensions can also be picked by allowing users to rearrange the original k -dimension key sequence. However, for shapes, the above sequence allows each subkey to be seen as an approximate description of a shape.

number of dimensions. This may be bad because of the dimensionality curse [4] that can result in the degeneration of searching the index structure to a sequential scan. On the other hand, a large L value can lead to many levels of R-trees. This may also contribute to poor performance as more number of nodes have to be traversed before the data can be retrieved. The optimal value of L is likely to be domain specific and requires tuning for different applications.

4.2.2. Handling skewed data distribution

In the basic NR-trees, only the leaf nodes of the last level of R-trees contain pointers to data buckets, while the leaf nodes at internal levels contain pointers to next level R-trees.

One issue that can affect the performance of the NR-trees is the distribution of data. Consider the case that most of the data records are clustered within a region in the data space and very few records fall outside this region. Each branch of the NR-trees consists of L R-trees (one at each level) and a sub-tree defines a region in the data space. If there are only a small number of data records under the branch, the cost of creating L R-trees is expensive in terms of both traversing the sparse R-trees and storage cost. To deal with this limitation, we propose that buckets be “hanged” at the leaf nodes of R-trees at internal levels to collect the small number of data. This will save the cost of creating a sub-tree. When the size of the bucket reaches a limit, the branch grows a new, next level R-tree. The bucket of records will be inserted into the new R-tree. Fig. 5 illustrates this enhancement. We note that the NR-trees is no longer a globally height-balanced tree structure, though each R-tree is locally height-balanced.

4.2.3. Supporting similarity retrieval

The NR-trees is designed to facilitate similarity retrieval. As mentioned, for query shapes where all dimensions are specified, results correspond to range queries in a high-dimensional space (using the two filters).

In addition, the NR-trees can be effectively used to support queries with fewer number of dimensions. This is easily facilitated as each level of the NR-trees essentially provides an approximate description of shapes. The top level provides an approximate description with l dimensions (recall that l is the number of dimensions used in each level R-tree). The second level provides an approximation description with $2l$ dimensions, and so on, until the last level which will provide the description of the shape in the precise form that is represented in the system. In fact, for an approximate query that matches the dimensions up to a certain internal level, the results of the query are actually those shapes that fall under the leaf nodes of that level. Retrieving all such objects will satisfy the query. Thus, to support such approximate queries, we propose that each R-Tree of the NR-trees contains an additional piece of information – the list of all its children R-trees and buckets. With this list, the children R-trees and buckets can be accessed directly and

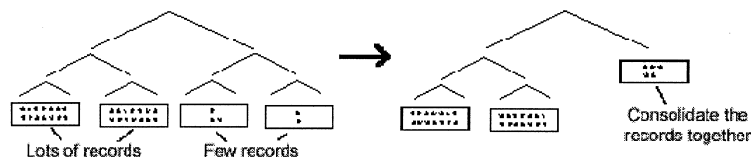


Fig. 5. Buckets at non-leaf levels.

quickly without traversing the R-trees. In this way, the retrieval process can stop at any internal level and retrieve all records under the sub-tree (at that level).

4.2.4. Implementation of the NR-tree

From the above discussion, the implemented NR-tree comprises five main objects: the R-tree object, the bucket object, the bucket manager, the ID-bucket object, and the ID-bucket manager object. Fig. 6 illustrates the implemented structure.

The R-tree object is a point R-tree implementation of Guttman's R-tree [12]. All R-trees, except those at the last level, index on subkeys of the same length. Every R-tree is tagged with an R-tree ID for identification. The bucket object is implemented as a circular linked list of pages. Records are kept in random order. Data (page) requests are made to a bucket manager object. Every bucket is tagged with a bucket ID for identification. The bucket manager organizes disk space for allocation to buckets using a free space manager. The bucket manager also maintains a B⁺-tree index to map bucket IDs to their bucket locations. Bucket IDs are used to invoke the bucket manager to access the required bucket.

Each R-tree has an accompanied ID bucket. An R-tree's ID-bucket contains the R-tree IDs of the children R-trees and the bucket IDs of the buckets hanged at its leaves. During a similarity retrieval, searching may stop at an internal level R-tree's leaf. The next step is to traverse to all the R-trees under the leaf's area of influence. At each R-tree, its children R-trees' IDs and buckets' IDs are retrieved. ID-bucket provides all the IDs directly without using a wide *bounding box* to retrieve from individual R-tree. Thus, after traversal stops at a leaf node, instead of accessing the R-trees at the next level, the ID-buckets are accessed to retrieve all the data records. An ID-bucket manager manages ID-bucket exactly like the bucket manager manages buckets.

Fig. 7 presents the algorithmic description of the insertion algorithm for the NR-tree. Let each subkey be l -dimension. The algorithm is highly abstracted. Routine *extractSubKey*(k, i) extracts from key k the l -dimensional subkey to be used at level i for indexing. The routine *traverse*(k, v) searches an R-tree rooted at v for subkey k . Routine *createRtree*(bk) creates an R-tree given a set of records in bucket bk . The routine *splitRtree*(v) splits a node v into two, v_1 and v_2 . It also updates the entry in the parent node of v to reflect the splitting. Finally, routine *parent*(v) returns the

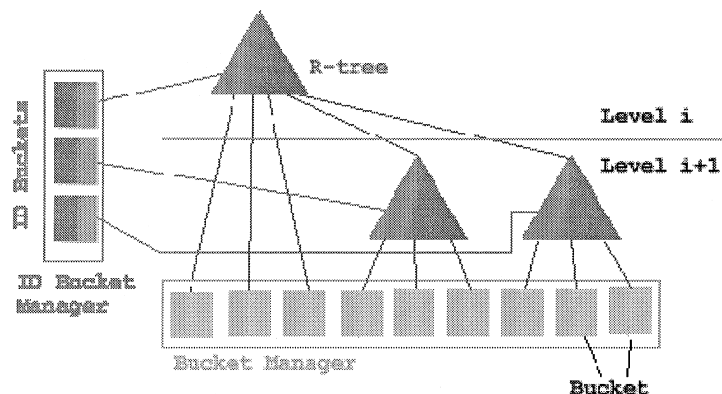


Fig. 6. Implementation of NR-trees.

Algorithm: NRInsert

Input: data D to insert has key k , a rootnode v of an R-tree, level i

```

1. subK ← extractSubKey( $k$ ,  $i$ )
2. while  $v$  is not a leaf node
3.    $v$  ← traverse( $subK$ ,  $v$ )
4. if  $v$  contains the entry  $subK$ 
5.   let  $u$  be the entry in  $v$  corresponding to  $subK$ 
6.   if  $u$  points to a bucket
7.     if this is the last level or bucket has not reached limit
8.       insert  $D$  into  $bucket$ 
9.   else
10.    insert  $D$  into  $bucket$ 
11.    newtree ← createRtree( $bucket$ )
12.     $u.ptr$  ←  $newtree$ 
13.  else // points to next level R-tree
14.    NRInsert( $D$ ,  $k$ ,  $u.ptr$ ,  $i + 1$ )
15. else //  $D$ 's key  $k$  not found
16.   create  $newbucket$ 
17.   insert  $subK$  into  $v$  with pointer to  $newbucket$ 
18.   while  $v$  overflows
19.     splitRtree( $v$ )
20.      $v$  ←  $parent(v)$ 

```

Fig. 7. Algorithmic description of NRInsert.

parent node of v . The algorithm recursively traverses an NR-tree level, to identify the next level R-tree until a bucket is hit.

The algorithm extracts the appropriate subkey for the current level (line 1) and traverses the current R-tree using the subkey (lines 2,3). First, it determines whether the leaf node at the level contains the subkey. If so, the pointer either points to a bucket or a next level R-Tree (lines 4–6, 13). For the former case (lines 6–12), there are two scenarios. In the first scenario, the bucket capacity has not been reached, and the datum is inserted there. In the second scenario, the bucket is full, and hence a new next level R-Tree is created. All records from the bucket plus the new data record are inserted into the new R-Tree. For the latter case (lines 14,15), the insertion algorithm is recursively invoked on the next level R-Tree. If the subkey does not exist in the current R-Tree (lines 15–20), a new bucket is created and the datum is inserted. The insertion may trigger the parent level R-trees to be recursively split.

The retrieval algorithm is shown in Fig. 8. It takes in the NR-tree instance, 2 keys that define the range retrieval and a StopLevel value. Instead of traversing all the levels of the NR-tree, retrieval may stop at an intermediate level indicated by StopLevel. This routine will return a list of the data records retrieved. As before, the algorithm is highly abstracted. The function *extractKey*(k, i) extracts a subkey at level i given key k . The routine *search*(v, k_1, k_2) returns the set of matching keys rooted at node v that fall in the range $[k_1, k_2]$. Routine *NR_GetAllRecords*($k.ptr, k_1, k_2$) examines all data records that can be found in the tree rooted at $k.ptr$, and returns the set whose key values are in the range $[k_1, k_2]$.

Algorithm: NRRetrieve

Input: search range $[k_1, k_2]$, an R-tree root node v , current level i , stopLevel to terminate search

Output: *result*

1. $sk_1 \leftarrow \text{extractKey}(k_1, i)$
2. $sk_2 \leftarrow \text{extractKey}(k_2, i)$
- // traverse R-tree at level i
3. $matchKeys \leftarrow \text{search}(v, sk_1, sk_2)$
4. for each matching key k in $matchKeys$
5. if $k.ptr$ points to a bucket
6. scan the bucket, and include into *result* those data whose keys
 fall between sk_1 and sk_2
7. else // $k.ptr$ points to next-level R-tree
8. if ($i < \text{StopLevel}$)
9. $result \leftarrow result \cup \text{NRRetrieve}([k_1, k_2], i+1, k.ptr, \text{StopLevel})$
10. else // $i == \text{StopLevel}$
11. $result \leftarrow result \cup \text{NR_GetAllRecords}(k.ptr, k_1, k_2)$
12. return *result*

Fig. 8. Algorithmic description of NRRetrieve.

First, the routine extracts the appropriate subkeys from the 2 keys defining the range retrieval (lines 1, 2). Using the 2 subkeys, range retrieval is applied on the current R-Tree to obtain a list of matching keys (line 3). For each of matching keys, if it points to a bucket, the data records are retrieved (lines 5, 6). For the other case where the pointer points to a child R-Tree, if the current level is the StopLevel, the routine NR_GetAllRecords is executed for a faster retrieval of all records under this R-tree (lines 10, 11). Otherwise, NRRetrieve() is recursively called on the child R-Tree.

5. A performance study

To evaluate the effectiveness of the proposed Nested R-trees, we implemented the structure and run experiments on synthetic test datasets. We compare the NR-trees against the R-trees and sequential scan approach (SCAN). This section reports on the experimental setup and our findings.

5.1. Experimental setup

The NR-trees, R-trees and SCAN are implemented in C on a Sun UltraSparc Workstation. Table 1 summarizes the default settings. In our study, due to storage constraint, we model the shape feature with 16 dimensions. Each node/page is also restricted to 1024 bytes. For the NR-trees, each R-tree will index on 4 dimensions. This is based on an earlier work that showed that breaking up a 16-dimension key to 4 4-dimension subkeys provides the best performance [44]. This gives rise to a fanout of 28 for each node. However, for a single R-tree that indexes on 16 dimensions, the fanout is reduced to 7. The size of a bucket is also fixed at 1 page. Several synthetic datasets containing random shapes are generated. These differ in the cardinalities of the

Table 1
Parameters and their default settings

Description	Default value
Size of a node/page	1024 bytes
Dimension of data	16
Dimension of each R-tree in the NR-tree structure	4
Fanout of each R-tree node in the NR-tree structure	28
Fanout of each R-tree node in a 16 dimension R-tree structure	7
Size of a bucket	1 page

datasets, and the distributions used (we shall defer this discussion to the experiments). Range queries are generated as follows. Suppose the data space has 16 dimensions and each axis takes value from 0 to 99, i.e., 100 possible values. Let us define the range width of an axis as the number of possible values for the axis. In the above example, the range width of each axis is 100. 2 keys are issued to perform range retrieval. A range retrieval of width 0.05 means that the difference in the i th values of both keys is 0.05 of the i th axis' range width. For the example data space, a pair of sample query keys will be like (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0) and (4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4). In our study, we use the number of page accesses as the metrics for comparison.

We have also conducted a visual inspection of the retrieval results which showed the accuracy of the retrieval system under the centroid–radii model. However, we would like to emphasize that *the goal of this paper is not to study the effectiveness of the centroid–radii model, but rather to see how the relevant images can be efficiently retrieved.*

5.2. Experiment 1: Uniform data distribution

The test data are generated in a 16-dimensional data space. Each axis of the data space ranges from 0 to 99. A total of 300000 shape features are generated randomly, i.e., the data are distributed uniformly in the data space. The process of generating random shape features for insertion is as follows. Each shape descriptor has 16 values. To obtain each value, we generate a random number between 0 and 99 with equal probability. The resultant NR-trees has only one level R-tree, and the height of the R-tree is 5. The total number of nodes is 190002. On the other hand, the single 16-dimension R-tree has a height of 9 and 76859 nodes.

The average insertion cost for R-trees is 14 pages per insertion while the insertion cost of NR-trees is 89 pages per insertion. The cost for NR-trees is about 4 times higher than the R-trees. This is because of the random distribution of data which results in more buckets being created. A possible explanation is that the level one R-trees of NR-trees handle 300000 shape descriptors in a 100^4 space while the 16-dimensional single R-tree has a 100^{16} data space. The 4-dimensional data space is more crowded than the 16-dimensional space. This leads to more overlapping MBRs for the NR-trees. Hence, the number of splitting is higher. Similar results have also been presented in [44].

Fig. 9 shows the retrieval cost for varying range width. We have not included the cost for sequential scanning as the cost is too large – more than 18000 pages. As expected, for both schemes, as the retrieval range increases, the cost increases. The graph also shows that the NR-trees is more efficient than the R-tree. For point retrieval, the NR-trees has an average of 139

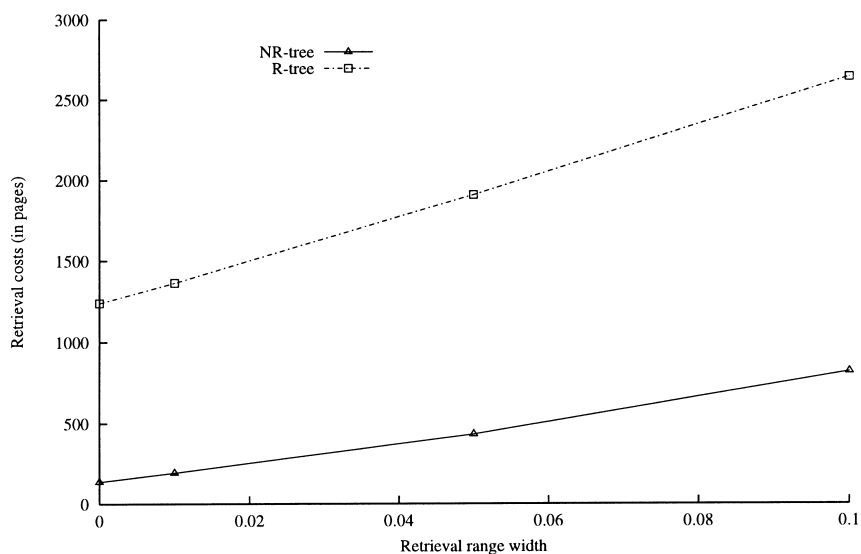


Fig. 9. Retrieval performance for uniform data distribution.

pages per retrieval while the cost for R-tree is 1241. This shows that the NR-trees performs better than the R-trees. This is expected as the NR-trees has only a single 4-dimensional R-trees and 4-dimensional R-trees has been proven to be more efficient than 16-dimensional R-tree. For range retrieval of width 0.01, the cost for the NR-trees is only 193 pages per retrieval while the cost for R-tree is 1365 pages per retrieval. The cost for the R-tree is 7 times more than that for the NR-trees. For widths of 0.05 and 0.10, the R-tree costs 4 and 3 times more than the NR-trees, respectively.

The results conform to conventional findings that retrieval cost for R-tree increases quickly as the retrieval range increases. The graph shows an increase in the cost for NR-trees too. Since the NR-trees comprises only one R-tree of 4-dimensional R-trees, its performance is basically dependent on the 4-dimensional R-trees.

5.3. Experiment 2: Skewed data distribution

In this experiment, we study the effect of skewed data distribution. The test data are generated in a 16-dimensional data space. Each axis of the data space ranges from 0 to 99. A total of 300 000 shape descriptors are generated. The distribution of the descriptors grows denser as one moves closer to the origin. A random number generator is used to generate a random value P between 0 and 1. Based on the value of P in Table 2, a value n between the range indicated by the corresponding entry is randomly generated. This process is repeated 16 times to obtain the 16-dimensional values. The 16 values form a record key for insertion.

The resultant NR-trees has 2 levels of 4-dimensional R-trees, giving a total of 144 R-trees each with an average height of 2 levels and an average number of 129 nodes. The resultant 16-dimensional R-trees has 9 levels with 99 179 nodes. As in the case for uniform data distribution, the insertion cost of the NR-trees remains high at 1265 pages, while that of the R-trees is 16 pages.

Table 2
Lookup table

Probability, p	Range
$0.0000 \leq p < 0.5000$	00–19
$0.5000 \leq p < 0.7500$	20–39
$0.7500 \leq p < 0.8750$	40–59
$0.8750 \leq p < 0.9373$	60–79
$0.9373 \leq p < 1.0000$	80–99

Fig. 10 shows the results for the retrieval performance. Once again, we have not shown the cost for sequential scan which exceeds 18000 pages. As shown, the cost of retrieval for the NR-trees has shot up greatly compared to that for the uniform data distribution. Point retrieval for the NR-trees is now 2181 pages per retrieval. This clearly demonstrated the effect of the skewed data distribution on the NR-trees. The NR-trees attempts to filter by subdividing the data space quickly. Each level of R-trees reduces the data space by 4 dimensions. However, if records fall within certain clusters, the effectiveness of the filter mechanism diminishes quickly. Consider a 16-dimensional key $(a_1, a_2, a_3, \dots, a_{16})$. If the range of values for a_1, a_2, a_3, a_4 is small, the first level R-tree will have lots of overlapping MBRs. The NR-trees will have an ineffective first level R-tree, thus, decreasing its efficiency. It is known that the performance of R-tree is affected by the distribution of records. As the main bulk of operation in the NR-trees includes the R-tree operations, it is only natural that the NR-trees is also affected. For range retrieval, the gap between the NR-trees and the R-tree is also narrowing due to the increase in the retrieval cost for the NR-trees. From the figure, it is clear that as the range retrieval width increases from 0 to 0.10, the gap is getting smaller too. Generally, the NR-trees outperforms the R-tree.

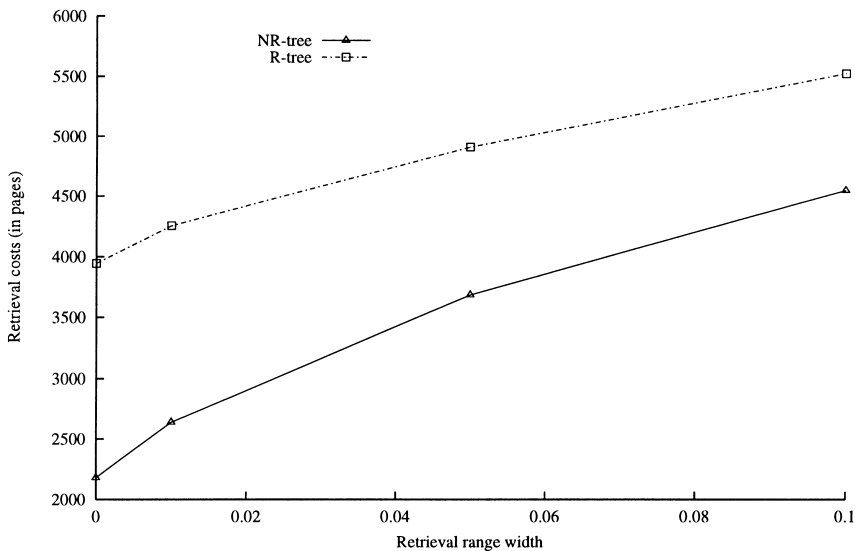


Fig. 10. Retrieval performance for skewed data distribution.

5.4. Experiment 3: Densely populated data space

We would also like to study the effect of densely populated data space, i.e., a large part of the data space contains data. The test data are generated in a 16-dimensional data space. Each axis of the data space ranges from 0 to 1. A total of 50000 shape descriptors are generated randomly. Note that the total data space is $2^{16} = 65535$, thus, the data space is almost full of records. Generating 16 random values between 0 and 1 with equal probability will create a record to be inserted. The process is repeated to create the required 50000 descriptors.

The resultant NR-trees has 4 levels of R-trees. There are 1, 16, 256 and 32 R-trees at levels 1, 2, 3 and 4, respectively. The 16-dimensional R-tree has 8 levels. The results of this experiment is interesting. First, the NR-trees has better insertion cost performance of an average of 4 pages per insertion compared the R-trees' 14 pages per insertion.

Fig. 11 shows the retrieval result for the NR-trees. The result for R-trees exceeds 9000 pages for range width of 0, which is inferior to that of the sequential search of 3200 pages. The result for R-tree is not surprising. As the data space is almost filled up, the number of overlaps in MBRs is large. Hence, the page access for R-tree is very large. On the other hand, the NR-trees remains effective in pruning the data space. Each R-tree of the NR-trees handles a smaller data space with fewer records. The result shows that the NR-trees is very effective for densely populated data space.

5.5. Experiment 4: Effect of database size

In the above studies, we have shown that the NR-trees is an effective indexing mechanism for high-dimensional data. In this experiment, we study the effect of database size on the NR-trees. Two different synthetic data sets are conducted, uniform data distribution and skewed data distributions. For each test, the number of the records is incremented in steps of 50000.

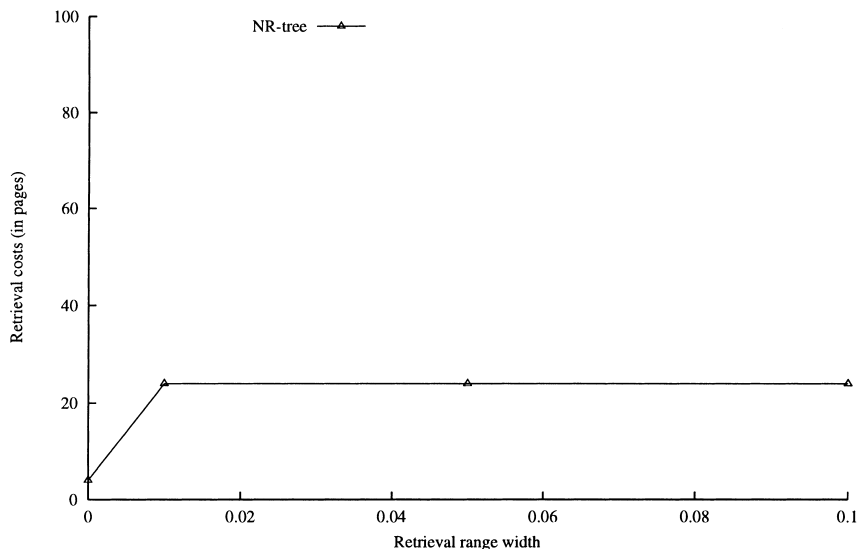


Fig. 11. Retrieval performance for dense data distribution.

Figs. 12 and 13 summarize the effect of database size on the performance of NR-trees. As expected, as more number of records are inserted, the insertion cost and the retrieval cost increase. Moreover, we note that for small database sizes, the average insertion cost is higher for uniform data distribution (Fig. 12). On the contrary, for larger database sizes, the cost is higher for skewed data distribution. For the retrieval cost (Fig. 13), as expected, it is more costly to retrieve from a skewed data distribution than a uniform data distribution. The results lend further support to our

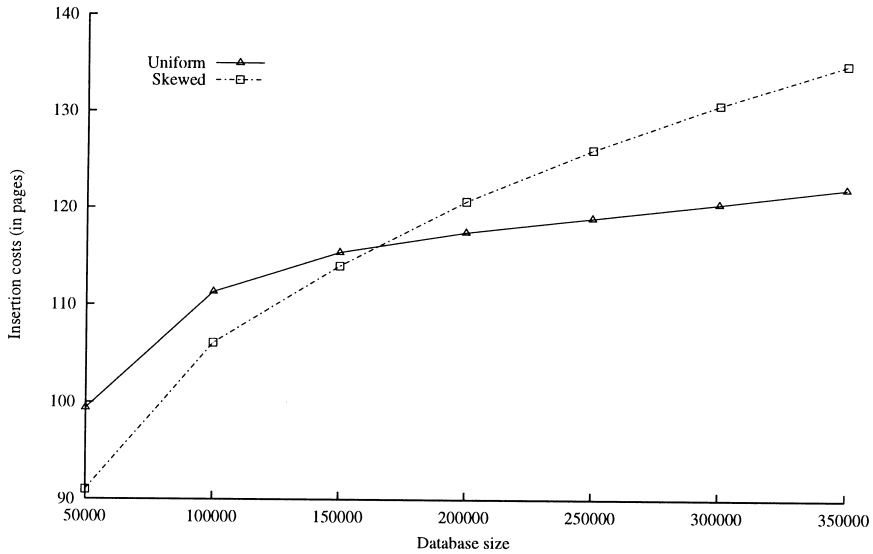


Fig. 12. Insertion performance (range width = 0.05).

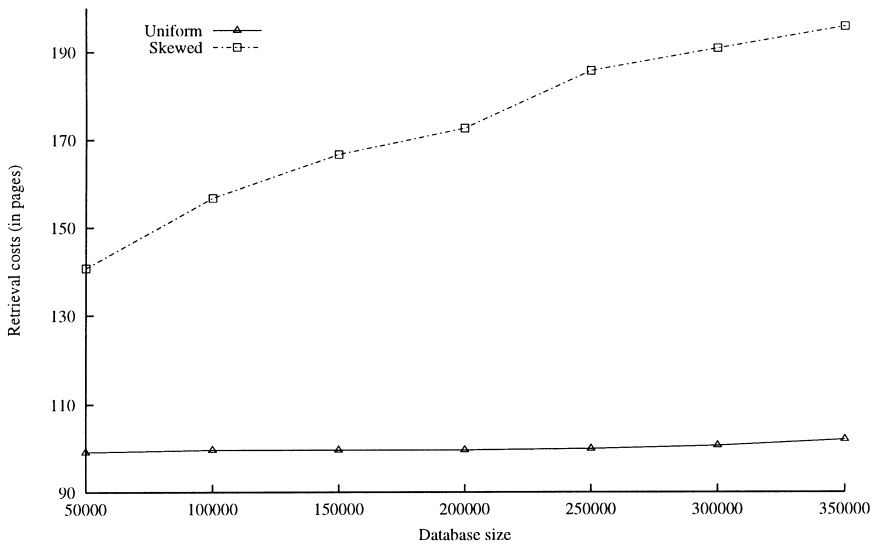


Fig. 13. Retrieval performance (range width = 0.05).

earlier observation that the distribution of records in the data space affects the operation of the NR-trees.

6. Conclusion

In this paper, we have addressed the issue of retrieval efficiency in image databases. We have employed the centroid–radii model to represent the shape of objects in an image. Using the model, the shape of an object can be mapped into points in a high-dimensional data space. This allows us to exploit existing high-dimensional point indexes to speed up the retrieval of images. We have also proposed a multi-level R-tree index, called the NR-trees and evaluated its performance against that of the R-tree. Our experimental results showed that the NR-tree is an effective index structure, and can facilitate similarity retrieval. We plan to extend this work in the following directions. First, we note that the NR-tree is not a height-balance structure. As such, we plan to study how we can achieve global height-balance in the structure. Second, we note that our NR-trees can also be used to index high-dimensional point data commonly found in many other applications. We are planning to study how effective the structure is in these applications.

Acknowledgements

This work is partially funded by NUS research grant RP950658. Lay Foo Thiang is also supported by grant RP3981674.

References

- [1] J. R. Bach, C. Fuller, A. Gupta, A. Hampapur, B. Horowitz, R. Humphrey, R.C. Jain, C. Shu, The virage image search engine: An open framework for image management. in: SPIE Proceedings of the Storage and Retrieval for Still Images and Video Databases IV, February 1996, pp. 76–86.
- [2] N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger, The R*-tree: An efficient and robust access method for points and rectangles, in: Proceedings of the 1990 ACM-SIGMOD Conference, Atlantic City, NJ, June 1990, pp. 322–331.
- [3] S. Belongie, C. Carson, H. Greenspan, J. Malik, Recognition of images in large databases using color and texture, 1997 (<http://elib.cs.berkeley.edu/papers.html>).
- [4] S. Berchtold, D.A. Keim, H-P Kriegel, The X-tree: An index structure for high-dimensional data, in: Proceedings of the 22nd VLDB Conference, Mumbai, India, September 1996, pp. 28–39.
- [5] E. Bertino, B.C. Ooi, R. Sacks-Davis, K.L. Tan, J. Zobel, B. Shilovsky, B. Catania, Indexing Techniques for Advanced Database Systems, Kluwer Academic Publishers, Dordrecht, 1997.
- [6] Y. Chahir, L. Chen, Piano key rediscovery for content-based retrieval of images, in: Proceedings of the SPIE Multimedia Storage and Archiving Systems II, Dallas, Texas, November 1997, pp. 172–181.
- [7] C.C. Chang, S.M. Hwang, D.J. Buehrer, A shape recognition scheme based on relative distances of feature points from the centroid, Pattern Recognition 24 (11) (1991) 1053–1063.
- [8] S.K. Chang, A. Hsu, Image information systems: Where do we go from here?, IEEE Transactions on Knowledge and Data Engineering 4 (5) (1992) 431–442.
- [9] C. Faloutsos, W. Equitz, M. Flickner, W. Niblack, D. Petkovic, R. Barber, Efficient and effective querying by image content, Journal of Intelligent Information Systems 3 (3) (1994) 231–262.
- [10] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, P. Yanker, Query by image and video content: The QBIC system, IEEE Computer 28 (9) (1995) 23–32.
- [11] H. Gunadhi, A. Segev, Efficient indexing methods for temporal relation, IEEE Transactions on Knowledge and Data Engineering 5 (3) (1993) 496–509.

- [12] A. Guttman, R-trees: A dynamic index structure for spatial searching, in: *Proceedings of the 1984 ACM-SIGMOD Conference*, May 1984, pp. 47–57.
- [13] W. Hsu, T.S. Chua, H.K. Pung, An Integrated color-spatial approach to content-based image retrieval, in: *Proceedings of the 1995 ACM Multimedia Conference*, San Francisco, CA, November 1995, pp. 305–313.
- [14] J. Huang, S.R. Kumar, M. Mitra, Combining supervised learning with color correlograms for content-based image retrieval, in: *Proceedings of the ACM Multimedia'97*, Seattle, Washington, November 1997, pp. 325–334.
- [15] J. Huang, S.R. Kumar, M. Mitra, W.J. Zhu, R. Zabih, Image indexing using color correlograms. in: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, San Jose, Puerto Rico, June 1997, pp. 762–768.
- [16] C.E. Jacobs, A. Finkelstein, D.H. Salesin, Fast multi-resolution image querying, in: *Proceedings of the Computer Graphics Conference*, Los Angeles, CA, August 1995, pp. 277–286.
- [17] H.V. Jagadish, A retrieval technique for similar shape, in: *Proceedings of the ACM-SIGMOD Conference*, May 1991, pp. 208–217.
- [18] K.F. Jea, Y.C. Lee, Building efficient and flexible feature-based indexes, *Information Systems* 16 (6) (1990) 653–662.
- [19] N. Katayama, S. Satoh, The SR-tree: An index structure for high-dimensional nearest neighbor queries, in: *Proceedings of the 1997 ACM-SIGMOD Conference*, Tucson, Arizona, May 1997, pp. 369–380.
- [20] T. Kato, Database architecture for content-based image retrieval, in: *SPIE Proceedings of the International Society for Optical Engineering*, San Jose, CA, 1992, pp. 112–123.
- [21] P.M. Kelly, M. Cannon, D.R. Hush, Query by image example: The comparison algorithm for navigating digital image databases (CANDID) approach, in: *SPIE Proceedings of the Storage and Retrieval for Still Images and Video Databases III*, February 1995, pp. 238–249.
- [22] A. Kitamoto, C. Zhou, M. Takagi, Similarity retrieval of NOAA satellite imagery by graph matching, in: *SPIE Proceedings of the Storage and Retrieval for Still Images and Video Databases I*, February 1993, pp. 60–73.
- [23] D.E. Knuth, L.M. Wegner (Eds.), *Visual Database Systems, II*, North-Holland, Amsterdam, 1992.
- [24] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, Z. Protopapas, Fast nearest neighbor search in medical image databases, in: *Proceedings of the 22nd VLDB Conference*, Mumbai, India, September 1996, pp. 215–226.
- [25] K. Lin, H.V. Jagadish, C. Faloutsos, The TV-tree: An index structure for high-dimensional data, *The VLDB Journal* 3 (4) (1994) 517–542.
- [26] H. Lu, B.C. Ooi, K.L. Tan, Efficient image retrieval by color contents, in: *Proceedings of the 1994 International Conference on Applications of Databases*, Vadstena, Sweden, June 1994, pp. 95–108.
- [27] S. Mallat, A theory for multiresolution signal decomposition: The wavelet representation, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 11 (7) (1989) 2091–2110.
- [28] P. Maragos, Pattern spectrum and multiscale shape representation, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 11 (7) (1989) 701–716.
- [29] P. Maragos, R.W. Schafer, Morphological skeleton representation and coding of binary images, *IEEE Transactions on Acoustics Speech and Signal Processing* 34 (1986) 1228–1244.
- [30] R. Mehrotra, J.E. Gary, Feature-based retrieval of similar shapes, in: *Proceedings of the Ninth Data Engineering Conference*, April 1993, pp. 108–115.
- [31] W. Niblack, R. Barber, W. Equitz, M. Flicker, E. Glasman, D. Petkovic, P. Yanker, C. Faloutsos, The QBIC project: Query images by content using color texture and shape, in: *SPIE V1908*, 1993.
- [32] J. Nievergelt, H. Hinterberger, K.C. Sevcik, The grid file: An adaptable symmetric multikey file structure, *ACM TODS* 9 (1) (1984) 38–71.
- [33] V.E. Ogle, M. Stonebraker, Chabot: Retrieval from a relational database of images, *IEEE Computer* 28 (9) (1995) 40–48.
- [34] B.C. Ooi, K.L. Tan, T.S. Chua, W. Hsu, Fast image retrieval using color-spatial information, *The VLDB Journal* 7 (2) (1998) 115–128.
- [35] A. Pentland, R.W. Picard, S. Sclaroff, *Photobook: Content-based Manipulation of Image Databases* 255, MIT Media Lab Perceptual Computing, 1993.
- [36] J.T. Robinson, The K-D-B-tree: A search structure for large multidimensional dynamic indexes, in: *Proceedings of the 1981 ACM-SIGMOD Conference*, June 1981, pp. 10–18.
- [37] T. Sellis, N. Roussopoulos, C. Faloutsos, R⁺-trees: A dynamic index for multi-dimensional objects, in: *Proceedings of the 16th VLDB Conference*, Brighton, England, August 1987, pp. 507–518.
- [38] J. Serra, *Image Analysis and Mathematical Morphology*, vol. 2, Theoretical Advances, Academic Press, San Diego, CA, 1988.
- [39] R. Shann, D. Davis, J. Oakley, F. White, Detection and characterization of Carboniferous Foraminifera for content-based retrieval from an image database, in: *SPIE Proceedings of the Storage and Retrieval for Still Images and Video Databases I*, February 1993, pp. 507–518.
- [40] M. Shibata, S. Inoue, Associative retrieval method for image database, *Transactions of the Institute of Electronics Information and Communication Engineers (D-II, J73D-II)* (1990) 526–534.

- [41] J.R. Smith, S-F. Chang, Visual SEEK: A fully automated content-based image query system, in: Proceedings of the 1996 ACM Multimedia Conference, Boston, MA, November 1996, pp. 87–98.
- [42] M.J. Swain, Interactive indexing into image database, in: SPIE V1908, 1993.
- [43] M.J. Swain, D.H. Ballard, Color indexing, *International Journal of Computer Vision* 7 (1) (1991) 11–32.
- [44] Y.T. Tan, Indexing high dimensional data, Technical Report Project Report 350, Department of Information Systems and Computer Science, NUS, Singapore, April 1996.
- [45] X. Wan, C.C.J. Kuo, Pruned octree feature for interactive retrieval, in: Proceedings of the SPIE Multimedia Storage and Archiving Systems II, Dallas, Texas, November 1997, pp. 182–193.
- [46] C.Y. Yee, K.L. Tan, T.S. Chua, B.C. Ooi, An empirical study of color-spatial retrieval techniques for large image databases, in: Proceedings of the International Conference on Multimedia Computing and Systems'98, June 1998, pp. 218–221.
- [47] Z. Zhou, A.N. Venetsanopoulos, Morphological skeleton representation and shape recognition, in: Proceedings of the IEEE 2nd International Conference on ASSP, New York, 1988, pp. 948–951.

Kian-Lee Tan received his B.Sc. (Hons) and Ph.D. in computer science from the National University of Singapore (NUS), in 1989 and 1994, respectively. He is currently an Assistant Professor in the Department of Computer Science, NUS. His major research interests include multi-media information retrieval, wireless information retrieval, query processing and optimization in multi-processor and distributed systems, and database performance. Kian-Lee is a member of ACM and IEEE Computer Society.

Beng Chin Ooi received his B.Sc. and Ph.D. in computer science from Monash University, Australia, in 1985 and 1989, respectively. He was with the Institute of Systems Science from 1989 to 1991 before joining the Department of Computer Science, National University of Singapore. His research interests include database performance issues, multi-media databases and applications, high-dimensional databases and GIS. He is the author of a monograph entitled “Efficient Query Processing in Geographic Information Systems” (LNCS #471, Springer, Berlin, 1990), and a co-author of a monograph entitled “Indexing Techniques for Advanced Database Applications” (Kluwer Academic Publishers, Dordrecht, 1997). He has published over 50 conference/journal papers and served as a PC member for a number of international conferences (including ACM-SIGMOD, VLDB, EDBT) and serves as an editor for *Geoinformatics*, *International Journal of GIS*, and *Journal on Universal Computer Science*. He is a member of ACM and IEEE Computer Society.

Lay Foo Thiang received his B.Sc. (Hons) in computer science from the National University of Singapore (NUS), in 1998. He is currently pursuing his M.Sc. in the School of Computing at the National University of Singapore. His research interests include high-dimensional indexing and content-based image retrievals.