RJ 9453 (83074) August 3, 1993
Computer Science

Research Report

EFFICIENT AND EFFECTIVE QUERYING BY IMAGE CONTENT

C. Faloutsos
M. Flickner
W. Niblack
D. Petkovic
W. Equitz
R. Barber

IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

RJ 9453 (83074) August 3, 1993
Computer Science

Efficient and Effective Querying by Image Content

C. Faloutsos *
M. Flickner
W. Niblack
D. Petkovic
W. Equitz
R. Barber

IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

Abstract

In the QBIC (Query By Image Content) project we are studying methods to query large on-line image databases using the images' content as the basis of the queries. Examples of the content we use include color, texture, sketch, and shape of image objects and regions. Potential applications include medical ("Give me other images that contain a tumor with a texture like this one"), photo-journalism ("Give me images that have blue at the top and red at the bottom"), and many others in art, fashion, cataloging, retailing, and industry.

We describe a set of novel features and similarity measures allowing query by image content, together with the QBIC system we implemented. We demonstrate the effectiveness of our system with normalized precision and recall experiments on test databases containing over 1000 images and 1000 objects populated from commercially available photo clip art images, and of images of airplane silhouettes. We also present novel methods for efficient processing of QBIC types of queries, that consist of filtering and indexing steps. We are specifically addressing two problems: (a) non Euclidean distance measures; and (b) high dimensionality of feature vectors. For the first problem, we introduce a new theorem that makes efficient filtering possible by bounding the non-Euclidean, full cross-term quadratic distance expression with a simple Euclidean distance. For the second, we illustrate how orthogonal transforms, such as the Karhunen Loeve transform, can help reduce the dimensionality of the search space. Our methods are general and allow some "false alarms" ("false hits", or "false positives") but *no* false dismissals.

The resulting QBIC system offers effective retrieval using image content, and for large image databases significant speedup over straightforward indexing alternatives. The system is implemented in X/Motif and C running on an RS/6000.

*On sabbatical from Univ. of Maryland, College Park. His work was partially supported by SRC, by the National Science Foundation under the grant IRI-8958546 (PYI).

1 Introduction

Current technology allows us to generate, scan, transmit, store, and manipulate large numbers of digital images. However, current practice in accessing and retrieving images is still primitive. Today we access image databases based on captions, e.g. [10]. Although useful, there are several problems with this approach, such as the fact that often the original keywords do not allow for unanticipated search in subsequent applications, and more important, inadequacy of uniform textual descriptions of such categories as color, texture, sketch, shape, layout etc. With the future availability of large image and multimedia databases, there will be a need to access images more naturally, using their content, in addition to traditional SQL and text queries. In our QBIC project we are investigating the means to allow users to search through databases consisting of very large numbers of images using sketches, layout or structural descriptions, texture, color, sample images, and other iconic and graphical information to specify the images desired. An example query might be: *Find all images with a pattern similar to one to which the user is pointing.* Possible applications of QBIC technology include "edutainment", journalism, museum and library cataloging, document processing, medical, intelligence and military, etc. We believe the QBIC technology should be a part of future multimedia databases that will contain text, sound, image and video. Crucial characteristics of image, and in general, multimedia databases (e.g., see [36] [14]), when compared to traditional databases include:

1. The size of data items can be very large (color images are of the order of a few MBytes; one hour of MPEG compressed VCR quality video is on the order of 1 GByte). This imposes severe requirements not only on storage, but also on data delivery (final query results, browsing etc.).

2. Storage and delivery of video data requires guaranteed continuous and synchronized delivery of data (i.e. video and audio), which in turn significantly influences the design of the whole system

3. Various query mechanisms need to be combined in an easy to use system containing traditional SQL and text search techniques together with novel content based search techniques. User and query interface should be visual as much as possible (i.e. allow the user to select color, shape and texture by graphical means) and also enable visual relevance feedback and user-guided navigation.

4. In many cases (like CD-ROM image databases or large image "archives") updates are rarely done or not done at all. In these cases we can optimize the indexing and precompute many image features. In other cases large volumes of data might be analyzed on-line as they are received by information filtering system (i.e. incoming news video) in which case similar search techniques could be applied, but with different optimization criteria.

Content based searches like QBIC have important distinctions compared to traditional searches. They are approximate and therefore there is no exact match. Images are typically sorted by similarity to the posed query and usually only a fraction of top ranked images is displayed. QBIC techniques serve as "database filters" and reduce the search for the user who will ultimately discard false retrievals or visually browse the returned images and select the ones he wants. We limit "content" to parameters that are feasible to compute given the state of the art in computer vision today, such as color distribution, shape, texture, layout etc. We do not, for the moment, attempt to automatically derive more complex semantic descriptions such as "dog", "cat", "house" etc. which are currently beyond the reach of pattern and image understanding technology. These descriptors should be entered by the user, if necessary. We also contrast QBIC technology with typical machine vision applications. In QBIC applications, through the interaction with the system, the user is offered the possibility of a virtually unlimited set of unanticipated queries rather than having a system performing only a limited set of predefined functions such as defect detection, fingerprint recognition etc. It is also important to note that in QBIC applications the main output is a set of images with desired properties that the user will use for subsequent application (inclusion in multimedia story etc.), rather than a symbolic decision as in typical pattern recognition (part is good/bad etc.).

A major challenge in QBIC queries is to determine a set of attributes or features that (a) describe the contents of an image; (b) admit appropriate similarity measures; and (c) form the basis of an index into the image collection. Having a natural visual user query interface that allows query refinement and navigation through the image database is also very important.

Many attributes are available from the fields of machine vision and image analysis describing image content – for example, a color histogram describing the set of colors in an image. These attributes are typically k-element feature vectors. However, there is a mismatch between the characteristics of these vectors and widely used multi-key indexing tools: the dimensionality k is often large, leading most indexing methods to "exponential explosions"; even worse, the distance function is not always Euclidean, in which case the multi-key indexing methods (R-trees, grid files etc) can not be efficiently applied.

In this paper, we:

1. Give an overview of our QBIC system that allows the user to query large image databases using image content such as color, texture, sketch and shape;

2. Present experimental results showing the effectiveness of our system in retrieving images for a range of content based image queries; and,

3. Propose and experimentally evaluate novel techniques to transform the feature vectors, so that they are amenable to efficient filtering and indexing.

In terms of efficient query processing we propose several methods that consist of applying efficient filtering techniques followed by more complex post processing, and also methods to reduce the dimensionality of search space by appropriately transforming the original feature vectors. These techniques are applicable to large classes of feature vectors and offer considerable speed-ups for large databases, compared to straightforward methods.

Section 2 presents a survey of relevant efforts in query by image content. Section 3 presents an overview of our QBIC system, its environment and our feature and similarity functions. In this section we also present implementation issues and some sample query results. Section 4 presents experimental results in effectiveness of QBIC retrieval, measured with normalized recall and precision, showing that the derived features indeed capture the similarity perceived by humans. Section 5 presents novel methods that allow efficient search for QBIC type of systems. Section 6 presents response time experiments that illustrate the speed-up of our novel indexing methods. Section 7 presents the conclusions.

2 Previous Work

This section reviews the past efforts in the areas of image query by content and multi-dimensional indexing.

2.1 Query by image content

Querying image databases by their image content is an active area of research. In terms of features to use, it benefits from the large body of work in machine vision on feature extraction and similarity measures, e.g., see [11], [3], [6].

In terms of methods and systems for image retrieval, examples of recent work include [48] and [50], which consider methods to retrieve images of line drawings and engineering diagrams; [7, 8, 29, 30], which assume known objects have been identified in images, and define and use 2D- and 2D-C strings to perform image retrieval based on the relative position of combinations of these known objects; [18], which presents a method for "query by sketch" in which a rough user-sketch of overall scene layout is used as the basis of a query; [21, 23, 26, 9, 32, 16, 28], which give methods for retrieving images based on the shape of objects they contain (or related methods to index into a database of shape models); and [5, 46, 20], which present retrieval methods based on the colors in a scene.

In many cases, the article emphasizes the vision aspects of the problem (e.g. [18, 21, 5, 20]), or the indexing issues (e.g., [23]). However, in [25, 1, 37], it was observed that there needs to be increased communication between the vision and the database communities for such problems, which is one of the contributions of this paper.

2.2 Multi-dimensional indexing

Within the database community, approximate matching has been attracting increasing interest. In [45] authors propose an indexing method that uses the triangular inequality and some precomputed distances to prune the search. [2] surveys recent research on Voronoi diagrams, along with their use for nearest neighbor queries. In [23] it is suggested to use a few minimum bounding rectangles to extract features from shapes and subsequently managing the resulting vectors using a spatial access method, like k-d-B-trees, grid files, which allows us to tap the vast literature on multi-key/spatial access methods. To do this, it makes two assumptions: (a) the "distance" between two objects corresponds to the Euclidean distance of the points in feature space, and (b) the dimensionality of the feature space is reasonably low. As described below, we consider cases where either or both assumptions do not hold.

Apart from these problems, which we show how to solve, our application needs a multidimensional indexing method that works for large, disk-based databases. The prevailing methods form three classes: (a) R^*-trees [4] and the rest of the R-tree family [17, 22]; (b) linear quadtrees [43]; and (c) grid-files [38].

Most multidimensional indexing methods explode exponentially with the dimensionality, eventually reducing to sequential scanning. For linear quadtrees, the effort is proportional to the hypersurface of the query region [19]; the hypersurface grows exponentially with the dimensionality. Grid files face similar problems, since they require a directory that grows exponentially with the dimensionality. The R-tree based methods seem to be most robust for higher dimensions, provided that the fanout of the R-tree nodes remains > 2. Experiments [39] indicate that R-trees work well for at least 20 dimensions. The most successful variant is the R^*-tree [4], typically being faster than the other R-tree variants. Thus, in QBIC, when we explore the impact of multidimensional indexing, we use the R^*-tree as an underlying indexing method.

3 Design and Overview of QBIC System

In this section we give an overview of QBIC algorithms and implementation. More details can be found in [37, 12]. In our current QBIC system we restricted ourselves to databases of still images, with two main datatypes: "images" (≡ "scenes") and "objects". A scene is a (color) image, and

Property	Query on:	
	image	object
color	√ □	√
shape		√ □
texture	√	√
sketch	√	

Table 1: Classification of Queries in QBIC."√":"supported"; □: "experiments presented"

an object is a part of a scene, e.g., a person, piece of outlined texture, apple etc. Each scene has 0 or more objects. Objects are determined by an outline, i.e. closed contour over image area, that is generated manually or semi-automatically. In QBIC, the user can draw an initial coarse outline around an object, which the system uses to "shrink-wrap" a final outline [37].

In QBIC, the three main steps are:

1. Database population, where images are imported into the system, and text keywords and objects outlined (if desired).

2. Image representation computation, where image features and attributes are computed and stored in the database, and subsequently used for content based searches.

3. Query process, where the user creates, navigates and refines the queries until he/she is satisfied with the output

3.1 Types of Queries

Given that semantic features are outside the capability of current machine vision technology, we selected the properties of color, texture, sketch and shape, because they have broad, intuitive applicability. For either a scene or an object, the user may ask a query on any of the above properties, or a Boolean combination of the above. All QBIC queries are "approximate" or "similarity"queries, and the system basically ranks the images based on the selected similarity function.

Table 1 shows the classification of queries. A √ indicates that QBIC can presently handle them; a □ indicates that we present experiments with these types of queries in this paper.

As an example, if a user is interested in retrieving a beach scene, he/she needs to map the query into available parameters offered by our system, e. g. the ones based on the color distribution (e.g. 35% white, 65% blue area coverage) and textures (presence of sand texture). QBIC will

retrieve and display images with these properties ranked by the selected similarity measure. The results will include beach scenes, as well as false retrievals (images that happened to have similar color and texture distribution). It is our experience that this does not represent a problem for the user, since the human visual system is excellent in quickly focusing on items of interest and discarding unwanted patterns, as long as there are not too many of them.

QBIC supports two ways of specifying a query:

- a "direct query", where the user specifies the desired color/shape/texture/sketch directly, e.g., by picking colors from a palette on the screen (e.g. "*multi-color picker*", see Figure 3, or drawing a sketch with the mouse).
- a "query by example", closely related to the concept of relevance feedback [42]: that is, the user can choose one of the displayed images/objects, and ask for images/objects similar to the selected one.

Notice that the two modes of operation may be used *interchangeably* within the same query session.

3.2 Image Features and Distance Functions

We describe the feature sets that comprise image/object representations and the associated distance functions that try to capture the similarity that a human perceives.

Color: We compute the average $(R, G, B), (Y, i, q), (L, a, b)$, and MTM (Mathematical Transform to Munsell [33]) coordinates of each object and each image. We also compute a k element color histogram, where k is user-settable. (Primarily we use $k = 64$.) Because the original color images may include any of $(2^8)^3 = 16$M colors, we quantize color space to k levels. Following [20], we initially quantize each axis in R, G, B to 16 levels, giving a color space of 4096 cells. We then compute the MTM coordinates of the center of each cell, and perform a standard, greedy, minimum sum of squares clustering [11] (pp. 235), to obtain the best k colors. Each color is actually the center of a "super-cell" in color space, where these supercells form a partition of the space. The image or object histogram is the normalized count of the number of pixels that fall in each of the supercells. Once these histograms are computed, there are a variety of ways in which to compute similarity between a pair of color histograms. In one method, the distance between two histograms ($K \times 1$ vectors) $\vec{x}$ and $\vec{y}$ is given by

$$d^2_{hist}(\vec{x}, \vec{y}) = (\vec{x} - \vec{y})^t A (\vec{x} - \vec{y}) = \sum_i^K \sum_j^K a_{ij}(x_i - y_i)(x_j - y_j) \quad (1)$$

where the superscript t indicates matrix transposition, and the matrix A has entries a_{ij} which describe the similarity between color i and color j. This formulation was proposed in [20], and

results in desirable retrieval performance. For example, with this measure we can correctly compute that orange images are similar to red images, and that a half-red/half-blue image is different from an all-purple one. We have also developed several new and convenient methods for specifying and calculating the similarity between color histograms. In the best of these, the user has the flexibility to not only compare two complete histograms, but can also specify partial histograms in queries such as "show me all images with approximately 20% red and 30% blue, where I don't care at all about the rest of the picture" (see "multicolor picker" in Sample Query Results section).

Notice that the Euclidean distance is the special case of a distance of Equation 1 if the matrix A is the identity matrix (I). The main difference between a Euclidean distance and the distance of Equation 1 is that the latter takes into account the "cross-talk" between two colors (such as orange and red). All the multi-key indexing methods make the implicit assumption that such cross-talk does not exist thus preventing their efficient implementation in our case. We address these problems in 5 and 6.

Texture: There is a wide variety of texture features described in the machine vision literature, but many were inappropriate for our application due to their computational complexity, and lack of correlation with human perception. Our texture features are based on modified versions of the coarseness, contrast, and directionality features proposed in [47]. The color images are first converted to gray scale before the texture features are computed. The *coarseness* feature helps measure the scale of the texture (such as pebbles vs. boulders), and is efficiently calculated using moving windows of different sizes. The *contrast* feature describes the vividness of the pattern, and is a function of the variance of the gray-level histogram. The *directionality* feature describes whether or not the image has a favored direction (like grass), or whether it is isotropic (like a smooth object), and is a measure of the "peakedness" of the distribution of gradient directions in the image. We modified the above features by making them more robust with respect to different sized and non-homogeneous images. We also modified the methods to make them more computationally feasible for the size of application we were designing for. See [12] for more details on our exact implementation.

Texture distance is computed as weighted Euclidean distance in the three dimensional texture space. The most common weighting (i.e. normalization) factors are the inverse variances for each component, computed over the samples in the databases. The assumption here is that the three texture measures are practically uncorelated, which we verified experimentally. For example, when only querying on texture, the distance between object i and object j is computed as

$$d_{ij} = \frac{(O_i - O_j)^2}{\sigma_O^2} + \frac{(C_i - C_j)^2}{\sigma_C^2} + \frac{(D_i - D_j)^2}{\sigma_D^2} \qquad (2)$$

where O, C, and D represent the texture features coarseness, contrast, and directionality respectively.

Shape: One of the most challenging aspects to content based image retrieval is retrieval by shape. Shape similarity has proven to be a difficult problem [35, 34] in model based vision applications. The problem remains difficult in content based image retrieval applications where similarity depends on the particular application, i.e. in some cases user might desire orientation invariance, in same cases not. Therefore, in general, we offer the user considerable control over these requirements.

Currently, our shape features are based on a combination of heuristic shape features such as area, circularity, eccentricity, major axis orientation, and a set of algebraic moment invariants. All shapes are assumed to be non-occluded planar shapes allowing for each shape to be represented as a binary image. The area is computed as the number of pixels set in the binary image. Circularity is computed as $perimeter^2/Area$. The second order covariance matrix is computed using just the boundary pixels. From this covariance matrix the major axis orientation is the direction of the largest eigenvector. Similarly, eccentricity is computed as the ratio of the smallest eigenvalue to the largest eigenvalue [24]. The algebraic moment invariants are computed from the first m central moments and are given as the eigenvalues of predefined matrices, $M_{[j,k]}$, whose elements are scaled factors of the central moments [49]. Using the notation of [49] we use moments up to degree 8 and the eigenvalues of the matrices $M_{[2,2]}, M_{[2,3]} \times M_{[3,2]}, M_{[3,3]}, M_{[3,4]} \times M_{[4,3]}, M_{[4,4]}, M_{[4,5]} \times M_{[5,4]}$ for a total of 18 features invariant to affine transformations.

The matching on the shape features is done similarly to that for texture, as weighted Euclidean distance where the weights are the inverse variances for each features. Any subset of the features can be selected by the user, enabling queries that are sensitive/insensitive to selected shape properties, in particular, to object size and object orientation. Since similar moments do not guarantee similar shapes we sometimes see perceptually different matches. This fact is leading us to examine more perceptual shape measures such as curvature and turning angle.

Sketch: We implemented the image retrieval method described in [18, 27] that allows images to be retrieved based on a rough user sketch. The feature needed to support this retrieval consists of a reduced resolution edge map of each image. To compute these edge maps, we (1) convert each color image to a single band luminance; (2) compute the binary edge image using a Canny edge operator; and (3) reduce the edge image to size 64 x 64. To do the reduction, for w the width of the image in pixels and h its height, we partition the image into blocks of size $w/64 \times h/64$, and if any pixel in a partition of the full size edge image is an edge pixel, the corresponding pixel in the reduced edge map is set to an edge pixel. Finally, we thin this reduced image. This gives the reduced edge map or "image abstraction" ([18]) on which the retrieval by sketch is performed.

The query, based on [18, 27], works by matching the user drawn edges to automatically extracted edges from the images in the database. The main steps of the algorithm are: (1) reduce the user sketch, which is a binary image, to size 64 by 64; (2) partition this into an 8 by 8 set of blocks, each block being 8 by 8 pixels; (3) for each image in the database, correlate each block of the sketch with a corresponding search area of size 16 x 16 in the database image; (4) compute a score for each database image as the sum of the correlation scores of each local block. The correlation in step (3) is done as a "logical binary" correlation, with specific values given on a pixel by pixel basis to edge-edge match between the user sketch and database image, edge-no edge match, no edge-no edge match, etc. Because each 8 by 8 block is spatially correlated separately, the method allows for some spatial warping between the sketch and database images. We are currently working on improving this method by making it more robust and efficient.

3.3 Implementation

The prototype QBIC system is written in C, with the user interface written in X/Motif. It consists of three primary parts – object identification, a set of batch feature calculation programs, and the query routine – plus a set of supporting utilities.

Typical use of the system involves running these three programs. Initially a set of images is read (for example, from a CD) and thumbnails of a common size (100x100) and color palette are generated. The user can optionally identify objects in each scene using the object identification program. The main output of this program is a set of binary masks, one per object.

In the second step, the batch feature calculation programs are run, generating color, texture, shape, location, and sketch features. This is a compute-intensive one-time operation. As a rough estimate of the computation time involved, on one of our systems (RS 6000 model 350), computing all features over approximately 1000 images and 2000 identified objects requires about 4 hours.

The third step is performing queries. All the above mentioned methods are available, as well as a control window to let the user specify which properties are to be used in a given query (color and/or texture and/or ...) and their relative weighs. The speed of the queries varies depending on the attribute queried on. Two examples: a color query retrieving the best n objects (for any n) from 2000 database objects requires about 2 seconds, plus thumbnail display time which is approximately 1-2 seconds; and a comparable sketch query requires about 40 seconds plus the same display time.

Our query user interface allows a user to specify color, texture, shape, and/or location properties visually by example, and to initiate queries based on these properties. Each property has one or more "pickers". The color picker to select a single color consists of a standard set of (R, G, B) sliders and a color patch showing the currently selected color. We have also used various color

wheels and IHS color pickers. The color picker that allows multiple colors and their amounts to be selected, such as for selecting a scene containing both blue and yellow, is shown in the left part of Figure 3. The palette on the left displays the set of selectable colors. Once a color is selected, it appears in a box on the right, and the amount of the color can be specified by the sliders. The texture picker presents patches of textures to be selected for query. Different sets of texture patches are available in our system, both synthetic and natural. Other texture sets can be easily created, set, and modified by the user according to desired application. For shape, two forms of pickers are available. One is a blackboard freehand drawing area in which the user draws a shape. An example is shown in Figure 2. A second form is a set of predefined shapes from which one is selected.

All the above measures can be used individually or in any weighted combination (as selected by the user), allowing, for example, queries based on color only, on color and shape, and so on. In this case, particular similarity measures from each individual query are combined into a final similarity measure. We got the best results by normalizing all individual similarity measures by their variance, and combining them in the form of a weighted sum.

When a query is run, results are displayed in order, from best match to nth best match (n is user-settable and we usually display top 20). Each image returned is displayed as a reduced "thumbnail". This thumbnail is an active menu button that can be clicked on to give a list of options. The options include queries of the form "Find images like this one" ("this one" selected by clicking on a thumbnail image), display the similarity value of this image to the query image, display the (larger) full scale image, place the image in a holding area for later processing, perform a user defined image operation or comparison, and so on. In order to improve the relevance feedback to the user we plot a similarity curve – a plot of similarity value versus order of retrieval for the images in a query. Often this curve will have a "knee", indicating that after some number of similar hits, the similarity falls off rapidly. By observing this knee, a user can decide how many images to retrieve, if it is likely that additional similar images are available, etc. In this way, we help the user navigate and refine queries during the search for the desired images.

All above queries are based on image content. However, available text information also provides valuable query information, and the QBIC system includes basic text search capabilities that can be combined with the above QBIC queries.

3.4 Sample Query Results

Sample query results are shown in the following figures that show a color and texture query, a color and shape query, a color histogram query and a sketch query. Note that in case two rows of images are displayed, the best match is in row 1, position 1, the next match is row 2, position 1, the next

is row 1, position 2 etc. This permits easy browsing by horizontal scrolling. The queries were done on different databases, ranging in size from about 700 to 2000 elements. The databases contain a wide diversity of photo clip art subject matter (buildings, people, landscapes, animals, etc.) read from commercial CD-ROM image libraries obtained from Applied Media Corp, West Chester PA.

Figure 1: Combined color and texture query, with the results.

Figure 2: Result of query by combining shape and color

Figure 3: Result of query on color histogram with multicolor picker.

4 Effectiveness of QBIC

In this section we present the results of experiments we did in order to asses the retrieval accuracy of QBIC system. Experiments on assessing efficiency of our indexing methods are presented in one of the subsequent sections.

Any system that deals with large number of complex patters that are formed by real signals, with their noise and inherent variability, needs to be extensively tested using carefully designed large scale experiments. It is important to note that QBIC falls into the category of information retrieval systems where there is no "exact" output, but a list of data items, sorted by similarity to the posed query is produced.

Information retrieval systems based on exact match may be evaluated using precision and recall statistics [42]. For a given query, let T the total number of relevant items available, R_r the number of relevant items retrieved, and T_r the total number of retrieved items. Then precision is defined as R_r/T_r, and recall as R_r/T. These two parameters are interdependent and usually one can not be improved without sacrificing the other.

In a system such as QBIC that performs similarity retrieval as opposed to exact match *normalized* precision and recall have been suggested [42]. These reflect the positions in which the set of relevant items appear in the retrieval sequence (ordered by some similarity measure). For example, *normalized recall* measures how close to the top of the list of retrieved items the set of relevant items appears, compared to an ideal retrieval in which the T relevant items appear in the first T positions. We used a variation of these measures (described below) to assess the performance of QBIC.

Figure 4: Result of query by sketch.

For the experiments, we selected q test images/shapes; the experimenters decided beforehand which items are relevant for each test case; and issued the query, displaying the best 20 matches. For each query, we calculated the following measures:

- $AVRR$ is the average rank of all relevant, displayed items (the first position is the 0-th rank).
- $IAVRR$ is the ideal average rank, when all T relevant images are ranked at the top: $IAVRR = (0 + 1 + \ldots + (T-1))/T$.
- RT is the rank of the retrieved test image.
- MT is the number of relevant images that were 'missed', ie, not among the displayed 20.

For each experiment, we report the average of the above measures over the q test queries. Notice that the ratio of AVRR to IAVRR gives one measure of the effectiveness of our retrievals. We report two experiments: color histograms using multicolor picker; and shape queries. We also performed many other successful experiments like simple match on average color, on texture etc., but do not report them here due to the space limitations.

4.1 Color Queries using Histograms

The test database consisted of over 1000 full color scenes. The multicolor color picker contained 64 colors. A query is formed by selecting up to five colors, together with their relative percentage. If (and only if) their sum was greater than 100, they were normalized to add to 100%. Two experimenters selected q=10 test images with distinct color properties. The same experimenters determined beforehand a set of relevant images for each of the test image. Relevant images were defined as those that the experimenters felt should be retrieved with the query that was being tested. By observing the test image from the printed album (with relatively poor color fidelity) the experimenter used multicolor picker and tried to retrieve the desired test image.

The left column of Table 1. shows the measures, averaged over the q=10 tests. The resulting AVRR=5.4 is close to the ideal: IAVRR=2.8. This implies that the average relevant image appears in the 5-6th position; given that we displayed 20 images, most of the relevant images were displayed, and were also in good ranks. In fact, only 3 relevant images were missed (i.e. not displayed in the top 20) out of the total 72 relevant images for the 10 test cases. No test images were ever missed; the average rank RT of the test images was 1.1, which means that the test image would typically appear in the 2nd place.

Measure	Colors	Shapes
size of db	1000+	295×3
# of queries T	10	6
IAVRR	2.8	4.2
AVRR	5.4	8.6
RT	1.1	n.a.
MT	3/72	11/64

Table 2: Results of Experiments on effectiveness

The results indicate that the color features and distance function were very effective. In addition, the multicolor color picker interface seemed quite intuitive and easy to teach to the novice. We observed that it was quite easy for the user to quickly discard the non-relevant images among the top 20. This points out that image information retrieval systems should be optimized for the human perceptual system and that they should benefit from its power to quickly scan the displayed set and easily reject non-relevant images. We also observed that the user would get better with time, probably by getting a better "feel" on how the query works.

4.2 Shape Queries

We tested our moment-based shape retrieval methods. For this test, we assumed size and orientation invariance (i.e. objects are to be retrieved if they are of similar shape even though rotated or of a different size). The test database consisted of 259 airplane silhouettes that contained various airplane types in three main views: front, top and side. Each view was considered a separate shape. In this experiment, as in the previous experiments, we did not want to test whether the shape measures can be used to accurately classify airplane types, but rather to see if they can be used to perform coarse shape classification, such as side views vs. top views, angled wings versus perpendicular wings, etc.

In each of six experiments, the experimenter sketched a silhouette of an airplane, without many fine details (fuel reservoirs etc), concentrating on the main features such as the angle of the wings and the general shape. The sketching was done using a polygon drawing routine. We displayed the 20 top ranked retrievals and recorded the same measures as for the color experiment. Again, relevant images were determined subjectively by the experimenter. Results are given in the right column in Table 1. Again, the average AVRR was well below the number of displayed images

(8.6 vs. 20), indicating that we typically retrieve the majority of the relevant images within the displayed set. Over all experiments, there were 64 images judged relevant, and 11 total misses.

The shape retrieval performed well. It was not capable of exact distinction among finer details (i.e. presence of payload on the wings etc.) but acted as a filter to reduce the set of images returned to the user.

Ultimately, the power of QBIC is in a user's ability to interactively form queries based on combination of color, shape, texture, sketch and text, adapted to the type of the images to be retrieved. We noticed that the user gets better with continued use of the system, indicating that the paradigm of human-machine intelligent information retrieval is indeed powerful.

5 Facilitating indexing

The methods we are describing for image and object retrieval require that every item (object or image) be mapped to a feature vector. For efficiency, these feature vectors are precomputed and stored. For a small size database, sequential scanning will be fast. However, as the database grows, sequential scanning's linear scale-up becomes prohibitively slow. The solution, as in [23], is to treat each feature vector as a point in n-d space, and employ a multi-dimensional indexing method.

In our application, we encountered two obstacles in applying the above method:

1. The *quadratic nature of the distance function*: The distance function in the feature space is not necessarily (weighted) Euclidean; there is "cross-talk" among the features, resulting in a distance function, as the one for color histograms (Equation 1), that is a full quadratic form involving all cross terms. Not only is such a function much more expensive to compute than a Euclidean (or any L_p) distance, but it also precludes efficient implementation of commonly used multi-key indexing methods.

2. The *"dimensionality curse"*: n may be large (e.g.. 64, or 256 for color features). Most multi-dimensional indexing methods require space and/or time exponential on n.

To address the above problems, we use a "signature" approach, that is, we create an efficient filter that is used in a preprocessing step, allowing some false hits, but *no* false dismissals. The candidate feature vectors are then subsequently processed by more expensive similarity function. This philosophy has been successfully used in several environments (text retrieval [13], differential files [44], hash-join algorithms, spelling checking [31], etc.).

Thus, in both the above problems, our goal is to find a mapping of the feature vector $\vec{X}$ into a vector $\vec{X}' = f(\vec{X})$, where $\vec{X}'$ is a vector in a more suitable space, with a distance function $D'()$

which will underestimate the actual distance:

$$D'(\vec{X}', \vec{Y}') \leq D(\vec{X}, \vec{Y}) \tag{3}$$

An operation that has such a property when using Euclidean distance is, for example, the projection of 3-d points on the x-y plane (i.e., truncation of the z co-ordinate/feature). The reason that we want Equation 3 to hold is to guarantee that a range query will not miss any actual hit. For example, the query *"retrieve all feature vectors $\vec{X}$ within tolerance ϵ from the query vector $\vec{Q}$"* will first operate on the lower-dimensional space, retrieving all $\vec{X}'$ such that $D'(\vec{X}', \vec{Q}') \leq \epsilon$. Thus, if a vector $\vec{X}$ qualifies ($D(\vec{X}, \vec{Q}) \leq \epsilon$), then, by Equation 3, it is guaranteed to qualify for filtering step $D'(\vec{X}', \vec{Q}') \leq (D(\vec{X}, \vec{Q}) \leq \epsilon$. We present two different forms of the function f. The first, which is a stronger result, is applicable to cases of high dimensional data that represent distributions over a lower dimensional space. The second is a more common dimensionality reduction technique.

We illustrate the first, using the color features and a new theorem (*"Quadratic Distance Bounding"*) described in the next subsection. It allows us to compute the parameters of the filter in terms of an efficient range query that generates candidates for more expensive similarity sorting using the actual similarity function. For the second problem, we propose the use of distance-preserving transformations, such as the Karhunen Loeve (KL), Discrete Fourier (DFT), or Discrete Cosine (DCT) transforms [40] which map n-d feature vectors into n-d vectors. The gain is that the transformed vectors will have most of the "information" (or "energy") in the first few coefficients. Thus, we use the first few coefficients for indexing resulting in a lower dimensional index, sacrificing a small number of false hits.

In the next two subsections we provide the mathematical foundation for the solutions. Following that, we show experimental results that demonstrate the efficiency gains.

5.1 Efficient Bounding of Color Distance Function

The key to efficient retrieval of images based on their full color histogram is an efficient approximation to d_{hist}, the histogram color distance defined in Equation 1. We do this by lower bounding the histogram color distance with a multiple of $d_{avg}(\bar{x}, \bar{y})$, the distance between the average color of two items. For example, the average color vector for a outdoor scene with 50% blue sky and 50% green grass would be, as an RGB vector, (0,50,50). That is, we establish that $d_{hist} \geq k d_{avg}$. As will be clear, the d_{avg} distance function is much cheaper to compute, both in terms of CPU cycles and in terms of disk accesses. This allows us to use d_{avg} as a means for filtering the database. For example, if we want to select all vectors for which $d_{hist} \leq \epsilon$, all we need to do is select all the vectors for which $k d_{avg} \leq \epsilon$ using range query methods which are very efficient. If we do this we are guaranteed to have selected all the vectors we wanted (for which $d_{hist} \leq \epsilon$), plus some additional

false hits. A second pass on this selected set using the actual similarity function can separate out the false hits.

The average color distance d_{avg} is defined on a different (and significantly smaller) color feature than the color histogram. This feature, the average color $\bar{x}$ of an image, should be pre-computed and stored in the database, even though it is computable from the color histogram (and the definitions of the color bins). Without loss of generality, say that colors are described by the triplet (R,G,B) (for 'R'ed, 'G'reen, 'B'lue), although different color spaces (such as Lab or YUV) result in a closer modeling of human judgment of color similarity. The average color of an image $\bar{x} = (R_{avg}, G_{avg}, B_{avg})^t$, is defined in the obvious way, with

$$R_{avg} = (1/N)\sum_{p=1}^{N} R(p), \tag{4}$$

$$G_{avg} = (1/N)\sum_{p=1}^{N} G(p), \tag{5}$$

$$B_{avg} = (1/N)\sum_{p=1}^{N} B(p). \tag{6}$$

Here we say that there are N pixels in the image, and that $R(p)$, $G(p)$, and $B(p)$ are the red, green and blue components (intensities, typically in the range 0-255) respectively of the p^{th} pixel. Now, given the average colors $\bar{x}$ and $\bar{y}$ of two images, we define d_{avg} as the simple Euclidean distance between the 3-dimensional average color vectors,

$$d^2_{avg}(\bar{x}, \bar{y}) = (\bar{x} - \bar{y})^t(\bar{x} - \bar{y}) \tag{7}$$

Before stating the main theorem we need to prove our bound, we establish a few definitions. We define a $K \times K$ matrix W in terms of the representative colors of each of the K color histogram bins. For example, if we say that bin i of the histogram is defined by the color (R_i, G_i, B_i), then we define W so that

$$w_{ij} = (R_i, G_i, B_i)^t(R_j, G_j, B_j) \tag{8}$$

We define an $(K-1) \times (K-1)$ matrix $\tilde{W}$ as a function of the matrix W, with

$$\tilde{w}_{ij} = w_{ij} - w_{iK} - w_{Kj} + w_{KK}. \tag{9}$$

A $(K-1) \times (K-1)$ matrix $\tilde{A}$ is defined similarly in terms of the color similarity matrix A.

Given these definitions, we state our theorem, the proof of which we leave to an appendix.

Theorem 5.1 (*QDB-* **"Quadratic Distance Bounding"**) *With d_{hist} and d_{avg} defined as above, and with $\tilde{A}$ positive semi-definite, we know that*

$$d^2_{hist} \geq \lambda_1 d^2_{avg},$$

where λ_1 is the minimum eigenvalue of the generalized eigenvalue problem

$$\tilde{A}\tilde{z} = \lambda\tilde{W}\tilde{z}.$$

In our application, $\tilde{A}$ is in fact positive semi-definite, so the theorem can be applied to prove the bound we need. Notice that the constant λ_1 depends only on the way we have created the K colors. The net result is that, given a color query, our retrieval proceeds by first filtering the set of images based on their average (R, G, B) color, then doing a final, more accurate matching using their full k-element histogram. The resulting speedup is discussed in Section 6.

5.2 Distance Preserving Transforms

Our shape similarity measure is the weighted Euclidean distance between the corresponding features. Using matrix notation this can be represented as $(\vec{x} - \vec{q})^t(\vec{x} - \vec{q})$, where $\vec{x}$ and $\vec{q}$ are $n \times 1$ feature vectors corresponding to two objects, a data object and a query object, respectively. If the data and the query features are transformed via an orthonormal transformation A, their distance will be preserved. This is easily seen since A is orthonormal ($AA^t = I$), and the distance can be computed as in Equation 10.

$$(A\vec{x} - A\vec{q})^t(A\vec{x} - A\vec{q}) = (A(\vec{x} - \vec{q}))^t A(\vec{x} - A\vec{q}) = (\vec{x} - \vec{q})^t A^t A(\vec{x} - A\vec{q}) = \sum_i (x_i - q_i)^2. \quad (10)$$

We seek a method that will under-estimate this distance using fewer dimensions, and our method is to select a subset of the transformed feature set. Let $\vec{a}_i\vec{x}$ be the i^{th} transformed feature, where $\vec{a}_i$ is the i^{th} row of A. Here we will be setting $\vec{a}_i$ to 0 for the $n - m$ least important entries. The sum in Equation 10 can be broken into two parts, so that

$$\sum_{i=1}^{n}(x_i - q_i)^2 - \left(\sum_{i=1}^{m}(\vec{a}_i(\vec{x} - \vec{q}))^2 + \sum_{i=m+1}^{n}(\vec{a}_i(\vec{x} - \vec{q}))^2\right) = 0. \quad (11)$$

By simple rearrangement of Equation 11, we can see that the error introduced by computing the distance on a subset of the transformed features (setting the last $\vec{a}_i$ to zero) is given by

$$\delta = \sum_{i=1}^{n}(x_i - q_i)^2 - \sum_{i=1}^{m}(\vec{a}_i(\vec{x} - \vec{q}))^2 = \sum_{i=m+1}^{n}(\vec{a}_i(\vec{x} - \vec{q}))^2 > 0. \quad (12)$$

Note that by Equation 12, we know that the truncated distance will always under-estimate the distance, i.e., $\delta > 0$. It is exactly this property that guarantees that the method will never miss a valid match.

The suitable transformations form two large families:

- Data dependent transforms, like the Karhunen Loeve transform, which need a sample of the data to perform statistical analysis for the determination of the transformation matrix A.
- Data independent transforms, such as feature sub-selection, Discrete Cosine (DCT), Harr, Fourier, or wavelet transform, where the transformation matrix A is determined a-priori.

The trade-offs between the two alternatives are as follows: The data-dependent transforms can be fine-tuned to the specific data set, and therefore they can achieve better performance, concentrating the energy into fewer features in the feature vector. Their drawback is that, if the data set evolves over time, its statistical characteristics may change, degrading the effectiveness of the result. In this case, a recomputation of a better A may be required. Given that a reorganization will be expensive, the data-dependent methods are recommended in the following two cases: (a) if the database is static (eg., a database published on CD-ROM, or a database with archival data), (b) if the sample that is used to determine the A matrix is representative of the full database.

Using least squares error minimization on δ in Equation 11 results in the statistically optimal data dependent transformation, the Karhunen Loeve transform. The results of this minimization is that A consists of the eigenvectors of the covariance matrix of the data. The rows of A are chosen so that the non-zero entries correspond to the eigenvectors with the largest eigenvalues of the covariance matrix. For a detailed explanation see [15]. To achieve dimensionality reduction, a common heuristic is to select the eigenvalues/vectors that contain say more than 80% of the information. If the eigenvalues λ_i are sorted in decreasing order $\lambda_i \geq \lambda_{i+1}$, this can be determined by finding a m such that

$$0.80 < \sum_{i=0}^{m} \lambda_i / tr(\Sigma) \tag{13}$$

where Σ is the covariance matrix and tr is the trace (sum of the diagonal elements).

For data-independent transforms, like the Discrete Fourier Transform (DFT), the problem of updating the transformation matrix A disappears. There is another problem, though: we want to ensure that the first few coefficients of the transform will carry most of the information. Fortunately, many transforms like the Discrete Cosine Transform (DCT) will perform as well as the Karhunen Loeve if the data follows specific statistical models. For example, the DCT widely used in image compression is very good at decorrelating data and would be an excellent choice if the features are highly correlated. If the statistical properties of the data are well understood, a data-independent transform in many common situations will obtain near optimal results.

6 Efficiency experiments

We ran simulations to test the effectiveness of the described methods. Specifically, we ran experiments on color, using the bounding theorem, and on shapes, using the Karhunen Loeve (KL) transform for dimensionality reduction.

6.1 Experiments on Fast Color Matching

We compared the relative performance (in terms of CPU time and disk accesses) between the two methods: (a) simple sequential evaluation of d_{hist}, for all database vectors, (referred to by *"naive"*), and (b) filtering using d_{avg} followed by evaluation of d_{hist} only on those records that pass through the filter (referred to by *"filtered"*). In our evaluations we did not implement indexing methods (such as R-trees) so that we may focus on the gains from the filtering step.

In evaluating the performance of the methods, we determined the CPU times required by the methods by performing simulations on an actual database of color image histograms. Our database consisted of 924 assorted natural images, and the sample queries involved matching each record in the database against the remaining records. Experiments were performed on a standard workstation (IBM RS/6000 model 530 — approximately 30 MIPs). In these experiments, we used K=256 element histograms.

First, we compare the selectivity of the filtering step. Results shown are the average performance over all experiments. In the ideal case, the filtering step would filter out only exactly those records for which d_{hist} was in the desired range. We are guaranteed that we will include all records for which this is true, but we also will retrieve some "false hits". Figure 5 shows the extent of the false hits for our test database. We can see that if we set our tolerance so that we retrieve the best 5% (ie., 50 best matches) of the database (in terms of d_{hist}), the filtering step will in fact retrieve approximately 30% of the database as candidates. This results in considerable savings, since the cost of doing even a few complete d_{hist} calculations is large compared with the cost of filtering applied to the entire database.

We now evaluate the relative cost of direct sequential evaluation of d_{hist} on all vectors as compared with filtering followed by evaluation of d_{hist} only on the selected subset of the original database. To avoid problems with buffering and pre-fetching, we estimate the number of disk accesses and subsequent time by assuming:

- Sequential disk accessing of a page of 1K bytes of data is 2ms
- Random disk accessing of a page of 1K bytes of data is 8ms

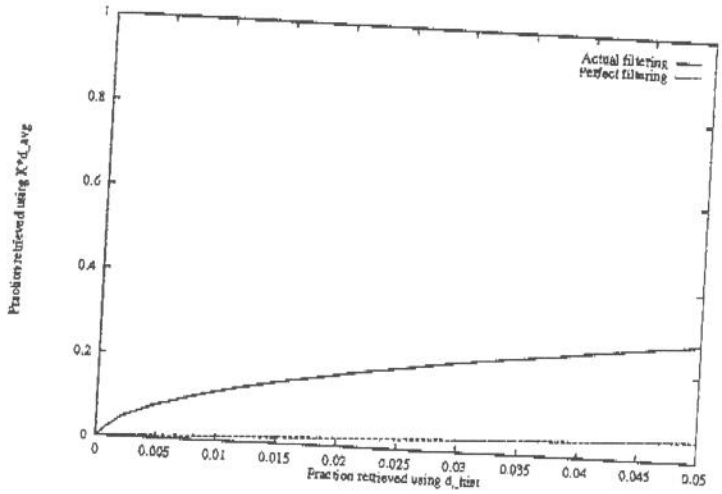

Figure 5: Amount of overshoot from color distance filtering

- The CPU time required to compute d_{avg} is negligible compared to the time required to compute d_{hist}.

- All disk accesses required for calculating d_{hist} in the filtering method are equivalent to a random access of an entire 1K byte page, regardless of the actual size of the histogram feature vectors. It was assumed that one would only return a single histogram vector per random disk access, although one would likely do better than this.

Results are shown in Figure 6. Given these worst case assumptions, we note that when we set our tolerances to retrieve a small fraction of the database such as 5%, the proposed filtering method, although it includes no special indexing methods, clearly outperforms the naive sequential application of the d_{hist} distance to all vectors in the database. Notice further that the naive method is CPU bound, which is almost 80% of the total time.

6.2 Experiments on Fast Shape Indexing

As we saw, the proposed solution to high-dimensional feature vectors with a Euclidean distance comparison function is to use an orthogonal transform; the optimal one for the data-dependent case being the KL transform. The result of the KL transform is that it produces new features, sorted in "strength" order. Thus, we can index on the first few of the new features. The questions that naturally arise are:

- How many of the new features we should keep

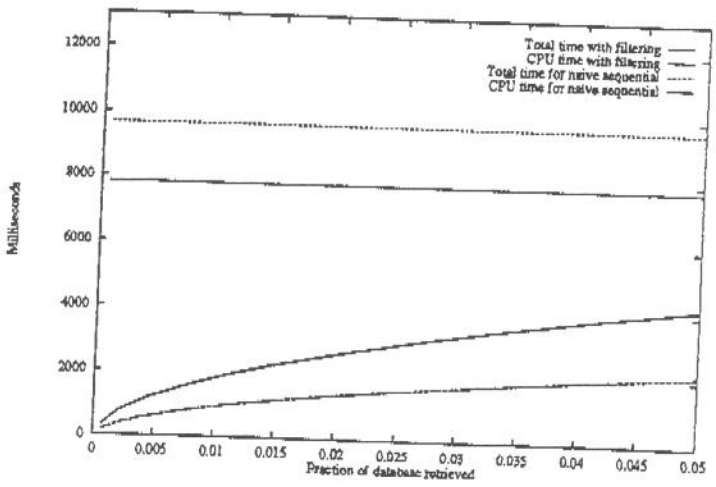

Figure 6: Time spent with sequential retrieval vs. filtered retrieval

- What is the performance gain of the method
- How does the method scale up for larger databases.

Simulations were run using 20 affine invariant shape features to answer the above questions. In both experiments, the page size for the R^*-tree was P=1024 bytes. Two databases were utilized. The first is the current QBIC object database with 1785 objects and is referred to as "Small" in the plots. The second database has 10634 shapes and was generated by computing the shape features on randomly thresholded luminance images of color photographs obtained from a commercially available CD-ROM. We refer to this database as "Large" in the plots.

In order to determine the gain from applying a orthogonal transformation, we took both databases, truncated the features to m dimensions, and inserted them into an m-dimensional R^*-tree. Thus we are randomly throwing away all but m features. Similarly, we KL transformed the features, ordered the transformed features in eigenvalue order, truncated the features to m dimensions and inserted them into an m-dimensional R^*-tree. We then performed range queries against the databases. For the small database the queries consisted of the database itself. For the large database the queries were 1041 random records in the database. Since the CPU time was negligible, compared to the I/O time we present number of disk accesses in our graphs. Figure 7 shows the number of disk accesses as a function of m, the number of retained features. This number includes both the accesses for the R^*-tree pages plus the random accesses to do the post processing to eliminate false hits. The tolerance ϵ was set to 200, resulting in an average of 22 hits/query for the large database and 2 hits/query for the small database.

The conclusions are:

- The optimal (in terms of disk access) m is $\approx$2. This is in good agreement with the heuristic presented (Eq. 13). In both databases, 75% of the energy is in the first two components. For larger tolerances the optimal is not as well defined, but generally there is a flat region for m between 2 and 8.
- Transforming the data achieves significant savings over non-transformed truncation. In fact, performance is better even if we miss the optimal m, i.e. we are better off with KL index of m=12 than with any non-transformed index. Note also that the absolute performance is better on an optimally created database 6 times smaller.

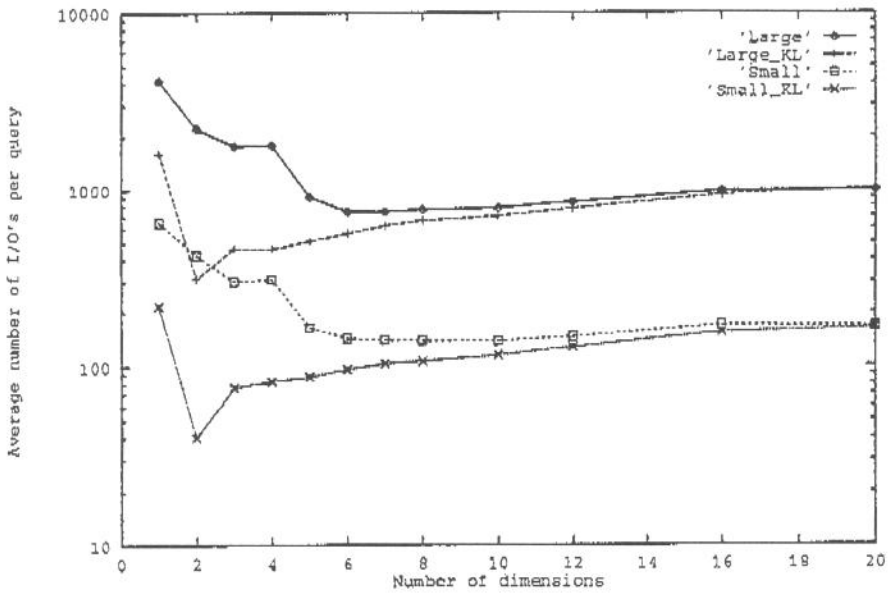

Figure 7: Average Disk I/O's per query, vs. dimensions kept m; ϵ=200

The second experiment determines how our method scales with database size. In this experiment we took a random subset of the "Large" database. Again we inserted the truncated features into an m-dimensional R^*-tree and the KL transformed version of the features into another m-dimensional R^*-tree. 1041 range queries were made with tolerance ϵ=200. Figure 8 plots the I/O operations for the naive approach and the KL approach for several values of m.

As shown in the graph, by KL transforming the data we gain a linear advantage in the number of I/O's per query. Also, as m increases the slope of our linear advantage decreases. This can be

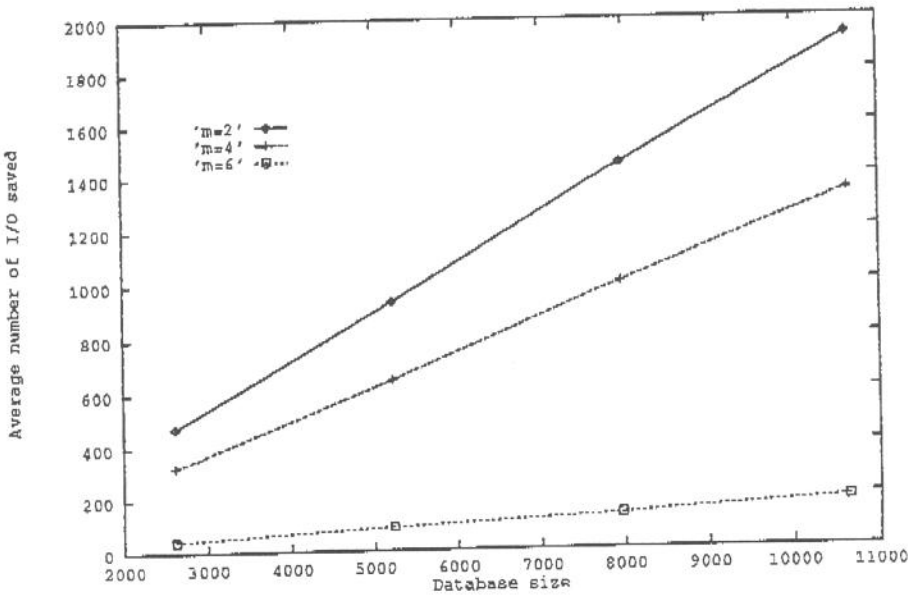

Figure 8: Average Saved Disk I/O's per query, vs. database size N

explained by the fact that as more features are used the index becomes larger and more expensive to traverse while there is a diminishing amount of information in the remaining features.

From a practical point of view, our experiments on real databases showed the following:

- for the Karhunen Loeve (KL) transform, a good cut-off value m for practical cases may be very low – in our case it was 2. That is, the first 2 features of the KL transform are enough for indexing purposes.
- by using the "Quadratic Distance Bounding" theorem, we replaced a distance function which was quadratic in the feature dimension with a constant time filtering function, followed by selective application of the original distance function. For large color histograms, this technique drops the CPU time, making the process I/O bound again.

7 Conclusions

Large online image collections are becoming more and more common, and novel methods to manage, organize, and retrieve images from these collections need to be developed. The goal of our QBIC project is to provide effective AND efficient query by image content on very large image database.

The focus of this paper is the design and implementation of such a system, coupling a set

of features and similarity functions from machine vision, with fast indexing methods from the database area. One of the our contributions is the introduction of techniques to make machine vision and database indexing tools work together. The result of this joint effort is an operational prototype that is both effective and efficient: the features and the similarity functions exhibit very good normalized recall, and the response time is much better than straightforward alternatives. Our indexing methods are expected to scale-up well for even larger databases. Great effort was undertaken to experimentally verify both the retrieval accuracy and the efficiency of the system.

Future work includes combining traditional text search methods with a variety of QBIC queries, and working on query design, optimization and experimental evaluation of such a system. We believe that the powerful multimedia database and information systems can be built by offering users a variety of intuitive query types (SQL, text, QBIC) that can be combined in an easy user interface, and efficiently executed. Our QBIC project is one step toward this dream.

Acknowledgments: We are thankful to E. Glasman for his help in development of sketch query method. We would also like to thank Jim Hafner and Harpreet Sawhney for their help with the proof of the *QDB* theorem.

Appendix — Proof of Color Distance Theorem

Define image histogram vectors:

$$\vec{x} = (x_1, \ldots, x_N),\ 0 \le x_i \le 1,\ \sum_i x_i = 1 \tag{14}$$

$$\vec{y} = (y_1, \ldots, y_N),\ 0 \le y_i \le 1,\ \sum_i y_i = 1 \tag{15}$$

$$\vec{z} = \vec{x} - \vec{y},\ -1 \le z_i \le 1,\ \sum_i z_i = 0, \tag{16}$$

color values (for example, A=Red, B=Green, C=Blue):

$$\vec{c}_i = \begin{bmatrix} A_i \\ B_i \\ C_i \end{bmatrix} \tag{17}$$

$$V^t = \begin{bmatrix} A_1 & A_2 & \cdots & A_N \\ B_1 & B_2 & \cdots & B_N \\ C_1 & C_2 & \cdots & C_N \end{bmatrix} \tag{18}$$

$$= (\vec{c}_1, \vec{c}_2, \ldots, \vec{c}_N), \tag{19}$$

average color distance:

$$\vec{x}_{avg} = V^t \vec{x} \tag{20}$$

$$\vec{y}_{avg} = V^t \vec{y} \tag{21}$$

$$d^2_{avg} = (\vec{x}_{avg} - \vec{y}_{avg})^t (\vec{x}_{avg} - \vec{y}_{avg}) \tag{22}$$

$$= (V^t \vec{z})^t (V^t \vec{z}) \tag{23}$$

$$= \vec{z}^t V V^t \vec{z} \tag{24}$$

and histogram distance:

$$a_{ij} = \text{similarity between color } i \text{ and color } j \tag{25}$$

$$d^2_{hist} = \vec{z}^t A \vec{z}. \tag{26}$$

To prove the bound, first change the $N \times N$ problem to an $(N-1) \times (N-1)$ problem, removing the constraint that $\sum z_i = 0$. Define $\tilde{z}$ as the truncated version of $\vec{z}$,

$$\tilde{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_{N-1} \end{bmatrix}, \tag{27}$$

and define $A_{N-1}, \vec{a}_{*N}, \vec{a}^t_{N*}$, and $\vec{1}$ appropriately so that you can expand $\vec{z}^t A \vec{z}$ as:

$$\vec{z}^t A \vec{z} = \begin{bmatrix} \tilde{z}^t & z_N \end{bmatrix} \begin{bmatrix} A_{N-1} & \vec{a}_{*N} \\ \vec{a}^t_{N*} & a_{NN} \end{bmatrix} \begin{bmatrix} \tilde{z} \\ z_N \end{bmatrix} \tag{28}$$

$$= \begin{bmatrix} \tilde{z}^t & -\tilde{z}^t \vec{1} \end{bmatrix} \begin{bmatrix} A_{N-1} & \vec{a}_{*N} \\ \vec{a}^t_{N*} & a_{NN} \end{bmatrix} \begin{bmatrix} \tilde{z} \\ -\vec{1}^t \tilde{z} \end{bmatrix} \tag{29}$$

$$= \tilde{z}^t A_{N-1} \tilde{z} - \tilde{z}^t \vec{a}_{*N} \vec{1}^t \tilde{z} - \tilde{z}^t \vec{1} \vec{a}^t_{N*} \tilde{z} + \tilde{z}^t \vec{1} a_{NN} \vec{1}^t \tilde{z} \tag{30}$$

$$= \tilde{z}^t \left[A_{N-1} - \vec{a}_{*N} \vec{1}^t - \vec{1} \vec{a}^t_{N*} + a_{NN} \vec{1} \vec{1}^t \right] \tilde{z}. \tag{31}$$

Now we can define the $(N-1) \times (N-1)$ matrix $\tilde{A}$ such that $\vec{z}^t A \vec{z} = \tilde{z}^t \tilde{A} \tilde{z}$, with

$$\tilde{A} = \left[A_{N-1} - \vec{a}_{*N} \vec{1}^t - \vec{1} \vec{a}^t_{N*} + a_{NN} \vec{1} \vec{1}^t \right], \tag{32}$$

and

$$\tilde{a}_{ij} = a_{ij} - a_{iN} - a_{Nj} + a_{NN}. \tag{33}$$

Now, if we let $W = VV^t$, and define $\tilde{W}$ analogously to $\tilde{A}$, we have

$$d^2_{hist} = \vec{z}^t A \vec{z} \tag{34}$$

$$= \tilde{z}^t \tilde{A} \tilde{z} \tag{35}$$

$$d^2_{avg} = \vec{z} V V^t \vec{z} \tag{36}$$

$$= \vec{z}^t W \vec{z} \tag{37}$$

$$= \tilde{z}^t \tilde{W} \tilde{z}, \tag{38}$$

and we are ready for the following theorem:

Theorem 7.2 *With d_{hist} and d_{avg} defined as above, and with $\tilde{A}$ positive semi-definite, we know that*

$$d_{hist}^2 \geq \lambda_1 d_{avg}^2,$$

where λ_1 is the minimum eigenvalue of the generalized eigenvalue problem

$$\tilde{A}\tilde{z} = \lambda \tilde{W}\tilde{z}.$$

Proof: We will show that $\tilde{z}^t\tilde{A}\tilde{z} \geq \lambda_1 \tilde{z}^t\tilde{W}\tilde{z}$, (a similar relation is posed as exercise 22.1 in [41]). For the sake of simplicity, we will assume A is symmetric (with real eigenvalues), since a non-symmetric A can be decomposed to a symmetric part and an asymmetric part, with the asymmetric part contributing nothing to the quadratic form.

For any $C > 0$, we set up the constrained minimization problem

$$\min_{\tilde{z}:\tilde{z}^t\tilde{W}\tilde{z}=C} \tilde{z}^t\tilde{A}\tilde{z}.$$

Use Langrange multipliers to convert this to the unconstrained minimization problem

$$\min_{\tilde{z}} \tilde{z}^t\tilde{A}\tilde{z} - \lambda\left(\tilde{z}^t\tilde{W}\tilde{z} - C\right).$$

Setting the derivatives equal to zero yields

$$\tilde{A}\tilde{z} = \lambda\tilde{W}\tilde{z},$$

so a necessary condition for minimizing the function is that λ and $\tilde{z}$ be generalized eigenvalues and eigenvectors for the problem $\tilde{A}\tilde{z} = \lambda\tilde{W}\tilde{z}$. Now from this necessary condition we can see that when the function is minimized

$$\begin{aligned}\tilde{z}^t\tilde{A}\tilde{z} &= \lambda\tilde{z}^t\tilde{W}\tilde{z} \\ &= \lambda C,\end{aligned}$$

so it is clear that to minimize $\tilde{z}^t\tilde{A}\tilde{z}$ we must choose the smallest of the candidate λ's. In other words, we must choose $\lambda = \lambda_1$, the smallest eigenvalue, which tells us that

$$\min_{\tilde{z}:\tilde{z}^t\tilde{W}\tilde{z}=C} \tilde{z}^t\tilde{A}\tilde{z} = \lambda_1 C.$$

Now, for any $\tilde{z}$, $\tilde{z}^t\tilde{W}\tilde{z}$ takes on some value (≥ 0), and when that value is strictly greater than 0 we use this value as our C, and by the above we know that

$$\tilde{z}^t\tilde{A}\tilde{z} \geq \lambda_1\tilde{z}^t\tilde{W}\tilde{z}.$$

When $\tilde{z}^t \tilde{W} \tilde{z} = 0$, we rely on $\tilde{A}$ being positive semi-definite and the inequality still holds. Consequently,

$$d^2_{hist} \geq \lambda_1 d^2_{avg},$$

where λ_1 is the smallest eigenvalue of the generalized eigenvalue problem $\tilde{A}\tilde{z} = \lambda \tilde{W} \tilde{z}$.

References

[1] ACM SIGIR. *Proceedings of International Conference on Multimedia Information Systems*, Singapore, 1991.

[2] Franz Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, September 1991.

[3] D. Ballard and C. Brown. *Computer Vision*. Prentice Hall, 1982.

[4] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: an efficient and robust access method for points and rectangles. *ACM SIGMOD*, pages 322–331, May 1990.

[5] Elizabeth Binaghi, Isabella Gagliardi, and Raimondo Schettini. Indexing and fuzzy logic-based retrieval of color images. In *Visual Database Systems, II, IFIP Transactions A-7*, pages 79–92. Elsevier Science Publishers, 1992.

[6] W. E. Blanz, D. Petkovic, and J. L. Sanz. *Algorithms and Architectures for Machine Vision*. ed. C.H. Chen, Marcel Decker Inc., 1989.

[7] C. C. Chang and S. Y. Lee. Retrieval of similar pictures on pictorial databases. *Pattern Recognition*, 24(7):675–680, 1991.

[8] Chin-Chen Chang and Tzong-Chen Wu. Retrieving the most similar symbolic pictures from pictorial databases. *Information Processing and Management*, 28(5):581–588, 1992.

[9] Zen Chen and Shinn-Ying Ho. Computer vision for robust 3d aircraft recognition with fast library search. *Pattern Recognition*, 24(5):375–390, 1991.

[10] S. Christodoulakis, M. Theodoridou, F. Ho, M. Papa, and A. Pathria. Multimedia document presentation, information extraction and document formation in minos: a model and a system. *ACM TOOIS*, 4(4), October 1986.

[11] R. Duda and P Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

[12] W. Equitz. Retrieving images from a database using texture — algorithms from the QBIC system. Research report, IBM Almaden Research Center, San Jose, CA, 1993.

[13] C. Faloutsos. Signature-based text retrieval methods: a survey. *IEEE Data Engineering*, 13(1):25–32, March 1990.

[14] Edward A. Fox. Advances in interactive digital multimedia systems. *IEEE Computer*, 24(10):9–21, October 1991.

[15] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, second edition, 1990.

[16] William I. Grosky, Peter Neo, and Rajiv Mehrotra. A pictorial index mechanism for model-based matching. *Data and Knowledge Engineering*, 8:309–327, 1992.

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

[18] Kyoji Hirata and Toshikazu Kato. Query by visual example. In *Advances in Database Techonology EDBT '92, Third International Conference on Extending Database Technology*, Vienna, Austria, March 1992. Springer-Verlag.

[19] G.M. Hunter and K. Steiglitz. Operations on images using quad trees. *IEEE Trans. on PAMI*, PAMI-1(2):145–153, April 1979.

[20] Mikihiro Ioka. A method of defining the similarity of images on the basis of color information. Technical report RT-0030, IBM Tokyo Research Lab, 1989.

[21] M. A. Ireton and C. S. Xydeas. Classification of shape for content retrieval of images in a multimedia database. In *Sixth International Conference on Digital Processing of Signals in Communications*, pages 111 – 116, Loughborough, UK, 2-6 Sept., 1990. IEE.

[22] H. V. Jagadish. Spatial search with polyhedra. *Proc. Sixth IEEE Int'l Conf. on Data Engineering*, February 1990.

[23] H. V. Jagadish. A retrieval technique for similar shapes. In *International Conference on Management of Data, SIGMOD 91*, pages 208–217, Denver, CO, May 1991. ACM.

[24] Anil K. Jain. *Fundamentals of Digital Image Procssing.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

[25] R. Jain and W. Niblack. Nsf workshop on visual information management, February 1992.

[26] T. Kato, T. Kurita, H. Shimogaki, T. Mizutori, and K. Fujimura. A cognitive approach to visual interaction. In *International Conference of Multimedia Information Systems, MIS'91*, pages 109–120. ACM and National University of Singapore, January 1991.

[27] Toshikazu Kato, Takio Kurita, Nobuyuki Otsu, and Kyoji Hirata. A sketch retrieval method for full color image database. In *International Conference on Pattern Recognition (ICPR)*, pages 530–533, The Hague, The Netherlands, September 1992. IAPR.

[28] Yehezkel Lamdan and Haim J. Wolfson. Geometric hashing: A general and efficient model-based recognition scheme. In *2nd International Conference on Computer Vision (ICCV)*, pages 238–249, Tampa, Florida, 1988. IEEE.

[29] Suh-Yin Lee and Fang-Jung Hsu. 2d c-string: A new spatial knowledge representation for image database systems. *Pattern Recognition*, 23(10):1077–1087, 1990.

[30] Suh-Yin Lee and Fang-Jung Hsu. Spatial reasoning and similarity retrieval of images using 2d c-string knowledge representation. *Pattern Recognition*, 25(3):305–318, 1992.

[31] M.D. McIlroy. Development of a spelling list. *IEEE Trans. on Communications*, COM-30(1):91–99, January 1982.

[32] Rajiv Mehrotra and William I. Grosky. Shape matching utilizing indexed hypotheses generation and testing. *IEEE Transactions on Robotics and Automation*, 5(1):70–77, 1989.

[33] Makoto Miyahara and Yasuhiro Yoshida. Mathematical transform of (r,g,b) color data to munsell (h,v,c) color data. In *Visual Communication and Image Processing*, volume 1001, pages 650–657. SPIE, 1988.

[34] David Mumford. The problem with robust shape descriptions. In *First International Conference on Computer Vision*, pages 602–606, London, England, June 1987. IEEE.

[35] David Mumford. Mathematical theories of shape: Do they model perception ? In *Geometric Methods in Computer Vision*, volume 1570, pages 2–10. SPIE, 1991.

[36] A. Desai Narasimhalu and Stavros Christodoulakis. Multimedia information systems: the unfolding of a reality. *IEEE Computer*, 24(10):6–8, October 1991.

[37] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, and P. Yanker. The QBIC project: Querying images by content using color, texture, and shape. In *IS&T/SPIE 1993 International Symposium on Electronic Imaging: Science & Technology, Conference 1908, Storage and Retrieval for Image and Video Databases*, February 1993.

[38] J. Nievergelt, H. Hinterberger, and K.C. Sevcik. The grid file: an adaptable, symmetric multikey file structure. *ACM TODS*, 9(1):38–71, March 1984.

[39] Michael Otterman. Approximate matching with high dimensionality r-trees. M.Sc. scholarly paper, Dept. of Computer Science, Univ. of Maryland, College Park, MD, 1992. supervised by C. Faloutsos.

[40] William K. Pratt. *Digital Image Processing*. John Wiley and Sons, Inc, New York, NY, second edition, 1991.

[41] C. R. Rao. *Linear Statistical Inference and Its Applications*. Wiley Series In Probability and Mathematical Statistics. John Wiley & Sons, New York, second edition, 1973.

[42] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[43] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1989.

[44] D.G. Severance and G.M. Lohman. Differential files: Their application to the maintenance of large databases. *ACM TODS*, 1(3):256–267, September 1976.

[45] Dennis Shasha and Tsong-Li Wang. New techniques for best-match retrieval. *ACM TOIS*, 8(2):140–158, April 1990.

[46] Michael J. Swain and Dana H. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

[47] Hideyuki Tamura, Shunji Mori, and Takashi Yamawaki. Texture features corresponding to visual perception. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-8(6):460–473, 1978.

[48] Satoshi Tanaka, Mitsuhide Shima, Jun'ichi Shibayama, and Akira Maeda. Retrieval method for an image database based on topological structure. In *Applications of Digital Image Processing*, volume 1153, pages 318–327. SPIE, 1989.

[49] Gabriel Taubin and David B. Cooper. Recognition and positioning of rigid objects using algebraic moment invariants. In *Geometric Methods in Computer Vision*, volume 1570, pages 175–186. SPIE, 1991.

[50] Koji Wakimoto, Mitsuhide Shima, Satoshi Tanaka, and Akira Maeda. An intelligent user interface to an image database using a figure interpretation method. In *9th Int. Conference on Pattern Recognition*, volume 2, pages 516–991, 1990.

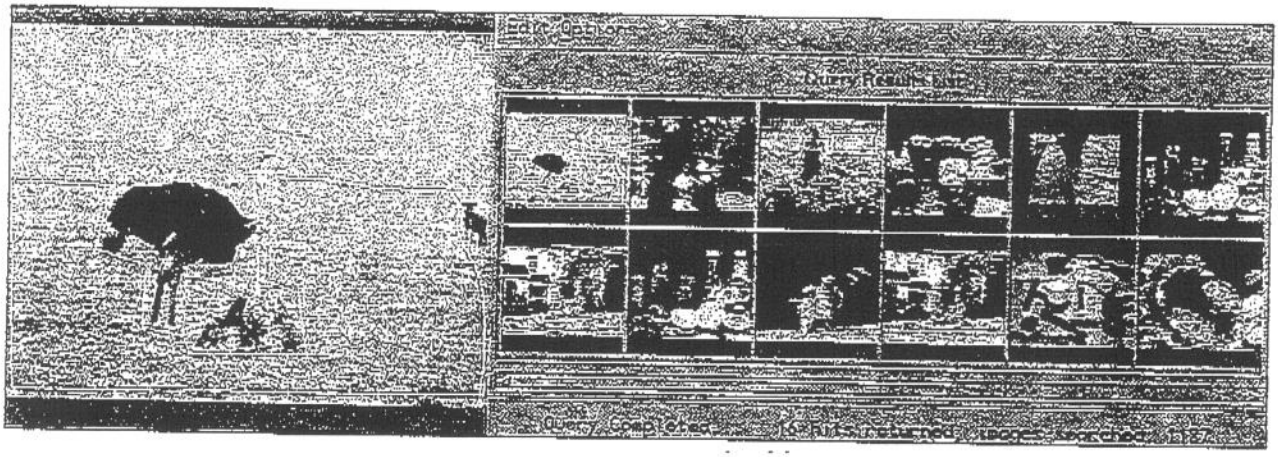

Figure 1: Combined color and texture query, with results.

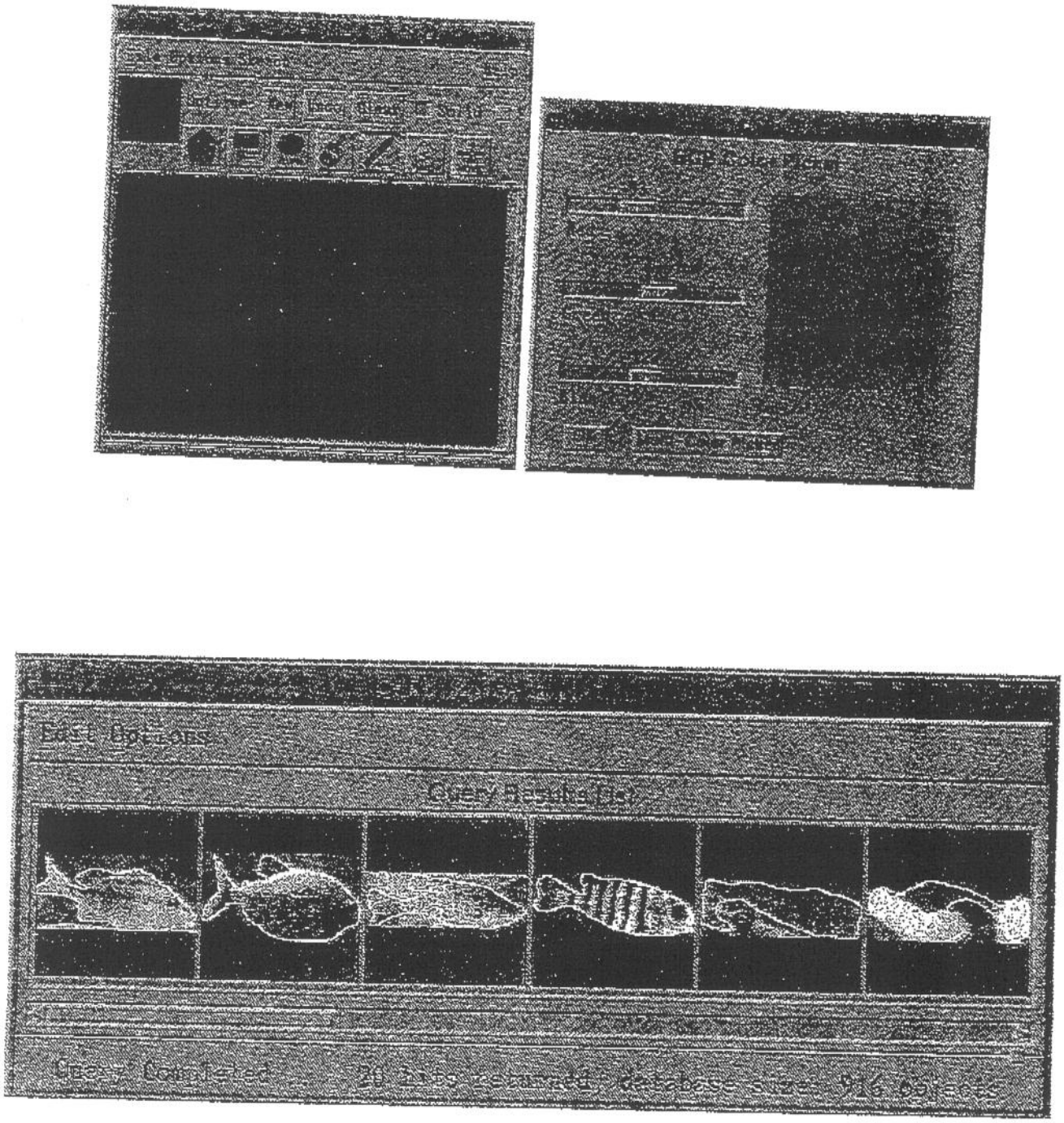

Figure 2: Result of query by combining shape and color.

Figure 3: Result of query on color histogram with multicolor picker.

Figure 4: Result of query by sketch.