A Flexible Image Database System for Content-Based Retrieval *

Andrew P. Berman
aberman@cs.washington.edu

Linda G. Shapiro
shapiro@cs.washington.edu

Department of Computer Science and Engineering
University of Washington

Abbreviated title: A Flexible Image Database System

Contact information:

Linda G. Shapiro
Dept. of Computer Science and Engineering
University of Washington
Box 352350
Seattle, WA 98195-2350

phone: (206) 543-2196
fax: (206) 543-2969
email: shapiro@cs.washington.edu

*This research was partially supported by the National Science Foundation under Grant No. IRI-9711771.

Abstract

There is a growing need for the ability to query image databases based on similarity of image content rather than strict keyword search. As distance computations can be expensive, there is a need for indexing systems and algorithms that can eliminate candidate images without performing distance calculations. As user needs may change from session to session, there is also a need for run-time creation of distance measures. In this paper, we present FIDS, "Flexible Image Database System". FIDS allows the user to query the database based on complex combinations of dozens of pre-defined distance measures. Using an indexing scheme and algorithms based on the triangle inequality, FIDS can often return matches to the query image without directly comparing the query image to more than a small percentage of the database. This paper describes the technical contributions of the FIDS approach to content-based image retrieval.

1 Introduction

There is a growing interest in image databases that can be queried based on image content rather than just with keywords. Such queries are based on the use of distance measures–scoring functions that rate the similarity of two images based on pre-defined criteria. There are several challenges that arise from this approach. The first challenge is that of flexibility. The user's definition of similarity may change from session to session, creating a need for runtime creation of distance measures. Another challenge is that of speed. Distance measure computation requires accessing either the images being compared, or pre-computed associated data. Furthermore, distance computation can be somewhat expensive. Thus, a system that must compute the distance from the query image to each image in a large database may exhibit prohibitively unsatisfactory performance. For certain distance measures and data sets, indexing or clustering schemes can be used to reduce the number of direct comparisons. Yet standard clustering or indexing schemes may not be efficient for some combinations of data sets and distance measures.

We have designed and implemented a prototype database system that allows the user a great deal of flexibility in runtime distance measure creation. The system can also reduce the number of direct distance measure calculations between a given query object and the database elements. FIDS, or "Flexible Image Database System," has been tested on a database of thirty seven thousand images, and is being upgraded to handle one hundred thousand images. FIDS allows the user to find approximate matches to query images using complex combinations of dozens of pre-defined distance measures. FIDS can often return results without directly comparing the query image to more than a small percentage of the original database.

There are algorithms in the literature based on the triangle inequality that can reduce the number of distance measure calculations[4, 6, 3, 10, 2] in object retrieval. These methods have the advantage of being applicable to any distance measure that satisfies the triangle inequality. FIDS uses algorithms based on the triangle inequality that are extensions to the methods in the literature.

In certain application domains, object *features* can be pre-computed and stored. A system can take advantage of this precomputation to speed up object searches at runtime. However, the cost of feature comparison by itself can still be prohibitive. In this paper, we do not distinguish between "image comparisons" and "feature comparisons." Although FIDS does pre-compute features of images in its database, it does not currently store those features and does not use them directly. Thus, when we state that a direct comparison is eliminated, the implication is that the feature comparison is also eliminated.

This paper describes the technical contributions of the FIDS approach to content-based image retrieval. Section 2 discusses related work in image retrieval. Section 3 discusses the use of multiple distance measures in user queries and the operations that can be used to combine them. Section 4 describes the FIDS indexing

scheme, which employs the triangle inequality to rule out large portions of the image database from direct comparison with the query, and Section 5 presents the FIDS interface. Section 6 gives experimental results of the basic retrieval algorithm, while Section 7 discusses and presents results for several different methods for selecting keys to be used in indexing. Section 8 describes an additional data structure, the triangle-trie, and presents experimental results on using it in conjunction with the basic FIDS retrieval methods to speed up the search even more.

2 Related Literature

The related work falls into two categories: 1) systems that attempt to provide a general image retrieval capability and 2) systems that provide a single retrieval technique, often for a particular application. In general image retrieval systems, one of the first efforts was the work of S. K. Chang [13] in which retrieval of images was achieved through attribute matching, spatial relation matching, structural (contour) matching, and similarity matching using various application-specific similarity measures. Kato [23] [27] developed an experimental database system called ARTMUSEUM that was intended to be an electronic art gallery. The system included a visual interface by which a user could enter a hand-drawn sketch, a monochrome photo or xerox copy, or a full color image of a painting in order to retrieve matching images from the database. The QBIC (Query by Image Content) system developed at IBM Almaden [33] became the first commercial product. The original QBIC allowed retrieval of images by color, texture, and the shape of image objects or regions. The system is constantly updated to add new retreival methods. Another commercial product, VIR, developed by Virage, Inc. [20], allows retrieval based on color, composition, texture, and structure measures. Virage has collaborated with Compaq to produce the *AV Photo Finder*, a website that allows retrieval of images based on color, composition, structure, texture, and keywords. They catalog over ten million images.

Pentland's group at MIT developed the Photobook system [34], a set of interactive tools for browsing and searching images and image sequences that allows queries by appearance, shape, and texture. Appearance refers to the technique of matching with eigenimages developed by Turk and Pentland [43], shape is based on the work of Pentland and Sclaroff [37], and texture matching comes from the work of Picard and Liu [36].

Most single-purpose systems have also concentrated on one of the four main classes of matching: shape, color, texture, and composition matching. In shape matching, most of the 2D contour matching techniques developed for computer vision apply, although several researchers have particularly targeted their work toward image databases. Grosky and Mehrotra [19] [31] used index trees to access a database of 2D object contour models of industrial parts. Califano and Mohan [11] developed a related indexing method that uses multi-dimensional global invariants of tuples of local interest features as indexes that vote for object models

in the database. Del Bimbo [8] retrieved images containing specified 2D shapes using an elastic matching technique.

The spatial and/or frequency distributions of colors or gray tones in an image are often used for retrieval. All of the general purpose systems contain one or more distance measures for color or "appearance-based" retrieval. Jacobs, Finkelstein, and Salesin [24] used a multiresolution wavelet decomposition to represent images for rapid matching and retrieval. Color indexing has been thoroughly analyzed by Swain and Ballard[41].

Texture classification and segmentation algorithms have been heavily studied in the computer vision/pattern recognition community [42]. In the image database community, Picard and Minka [35] developed a texture-based vision annotation system that learns from user feedback. Kelly and Cannon [28] [29] at Los Alamos National Laboratory used a global signature to characterize images and a signature distance function to compare them. Jain, Murthy, and Chen [26] have developed a methodology for comparing various texture similarity measures.

Most of the above techniques are global in that they compare some global feature or features of an image, such as signatures, histograms, eigenimages, and wavelet representations. Structural techniques break the image into a set of extracted entities and produce a description in terms of these entities, their properties, and their interrelationships [40]. Descriptions are encoded as string, tree, or graph structures, and relational matching is performed to determine the distance between a query and a stored description. S. K. Chang, Shi, and Yan [14] encoded the spatial relations among picture objects as 2D strings and provided an iconic indexing technique using these strings to retrieve images. In 3D object recognition work performed at the University of Washington, Costa and Shapiro developed an accumulator-based relational indexing technique that uses subgraphs of a structural description of the image to vote for 3D view-class object models [39].

Del Bimbo [7] [9] used both spatial and temporal relations to select image sequences that match a query. Bach, Paul, and Jain developed a feature-based approach to retrieval of face images [1]. The distance between a query image and an image from the database was a weighted sum of the differences between pairs of corresponding query image features and database image features.

Much of the older work in content-based retrieval only gave illustrations of system performance without any real evaluation. In newer work, Jain and Vailaya [25] describe a system for content-based retrieval of trademark images using a combination of color and shape features. A thorough set of experiments was run to judge the accuracy, stability, and speed of the approach. Accuracy was defined to mean that the most similar image to the query image should be in the top set of returned images. Stability or robustness meant that the procedures should not break down under various conditions; the conditions tested were arbitrary

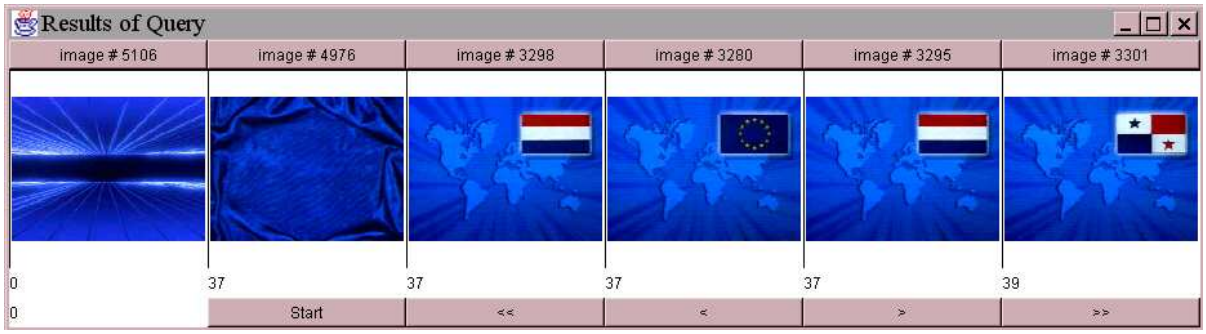


Figure 1: Result of query with a color-based distance measure.

rotations, size variations, and addition of random noise. Another interesting and thorough study was the recent work of Minka and Picard [32]. The Four Eyes system, which extends their texture work, represents a general supervised learning approach to image retrieval.

More recently, Sclaroff, *et. al.*[38] used a k-d tree algorithm to index images collected on the Web. Using a relevance feedback mechanism, the user attempts to progressively home in on the desired images. They also combined their results with keyword searches. Netra[30] allows searches based on simple combinations of color, texture, shape and position of shapes extracted from images offline.

Because of the shortcomings of computer vision segmentation algorithms, very little of the older work has involved actual object identification. Forsyth *et al* [18] recently proposed a new approach that involves a sequence of grouping activities that satisfy increasingly strict constraints. As an example, Fleck, Forsyth, and Bregler [16] developed a procedure for finding naked people in color images, for possible use in products that restrict access to web sites with pornographic material. The process has two levels of filtering. The first filter looks for large areas (30% of the image) of skin color. Regions that pass this test go on to a geometric analyzer looking for elongated regions that can be grouped into certain spatial relationships that are typical of limbs connected to torsos. This work is a first step in the recognition of abstract concepts in difficult, real images. Another promising approach is that of Carson, et. al.[12] who segment images into *blob* regions with annotated color and texture properties, enabling retrieval of images that contain similar regions.

3 The Use of Multiple Distance Measures

A *distance measure* is a function that computes and returns a value corresponding to the similarity between two objects according to some predefined criteria. For example, Figure 1 illustrates the result of a query that used a color-histogram distance measure to retrieve images similar to the leftmost image. Figure 2 illustrates the result of a query that used a texture-based distance measure to retrieve images for the same

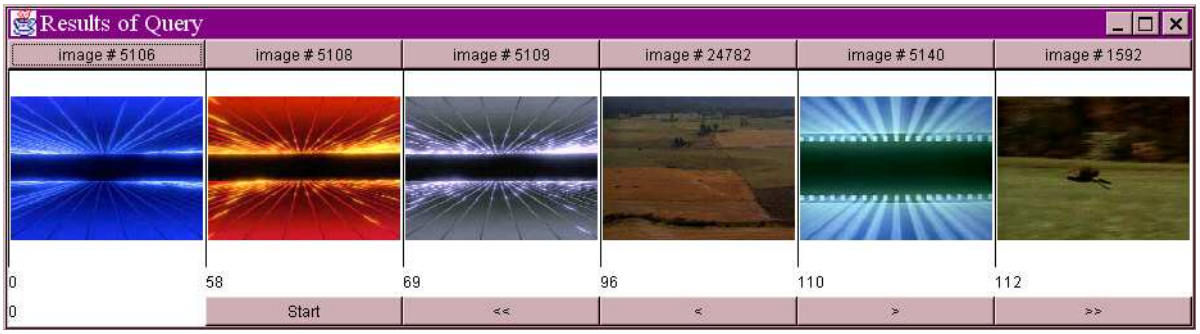


Figure 2: Result of query with a texture-based distance measure.

query image as in Figure 1.

Two fundamental classes of distance measures are those measures based on color properties of images and those measures based on image texture properties. There are many different specific color measures and texture measures in use, and an infinite number of distinct possible measures. Other types of measures may be based on features such as the existance of particular objects within the image, the relative position of various image features or literally any quality that one can ascribe to an image.

3.1 The Need for Multiple Measures

One can create an innumerable number of distance measures for images based on any set of features and an arbitrary scoring mechanism. We cannot program all these distance measures in advance. This difficulty motivates the idea of giving the user a pre-defined set of base distance measures that he or she can combine to create more complex measures. The potential space of creatable distance measures increases as we increase the number of pre-defined measures. For example, Haering, Myles, and da Vetoria Lobo [22] demonstrated a neural network with 43 distance measures as input that was trained to detect deciduous trees.

Apart from having many different distance measures, we believe that there is a need for multiple variants of the basic distance measures. A piece of software representing a distance measure also represents many decisions made by the programmer. The user of the distance measure may disagree with, not understand, or not even know about these decisions. For example, there are innumerable ways to define bin size and placement for the underlying color histogram of a color-histogram-based distance measure. A user who does not wish to distinguish pink from red will be unhappy with a color measure that does distinguish the two colors. Giving the user multiple color histograms from which to choose increases the potential utility of the system for such a user.

The problems of color measurement only increase when dealing with texture distance measures. As in

color-histogram measures, there may be many decisions made by the programmer that control the behavior of the texture measure. These decisions are opaque to the user, yet they affect the performance of the measure for the user. The difficulties are compounded in that the semantics of texture are more obtuse than the semantics of color. Thus, just giving the user multiple textures from which to choose is not a complete strategy, since he or she will have no method (other than trial and error) by which to decide what measures to use.

A more subtle problem with distance measures is that they force the user to make concrete decisions about similarity when the user may not desire to do so. The user may wish to find images that are "similar" without actually specifying any particular measure. For example, consider the user who wants the set of all closest matches to a query image over all possible color-histogram-based distance measures. This set may actually be quite small, yet reliance on any single distance measure will leave out some matches. In this case, such a fuzzy similarity can be approximated by querying the database with several different color distance measures.

3.2 Methods of Combining Distance Measures

Given a database system with several distance measures, one has to decide in what ways the user shall be allowed to combine the distance measures. There are three separate, possibly conflicting objectives. The first objective is to provide many choices to the user. The second objective is to provide an interface that allows the user to understand his choices. The third objective is to support database search of the resultant queries in an efficient manner.

Systems such as QBIC[17] and Virage[21] offer the user the ability to take weighted combinations of color, texture, shape, and position measures, combined with keyword search. But what of the user who wishes to formulate a queries such as "Match on colors, unless the texture and shape are both very close" or "two out of three of color, texture, and shape must match"? These queries cannot be expressed as a weighted sum of individual distance measures. In order to expand users' searching vocabulary, more complex combinations of distance measures must be offered.

To deal with these problems to some extent, we proposed[5] the following set of operations to enable more expressive queries: ($d_1 ... d_n$ represent distance measures)

- Addition: $d = d_1 + d_2$
- Weighting: $d = cd_1$
- Max: $d = \text{Max}(d_1, d_2, \ldots, d_n)$
- Min: $d = \text{Min}(d_1, d_2, \ldots, d_n)$

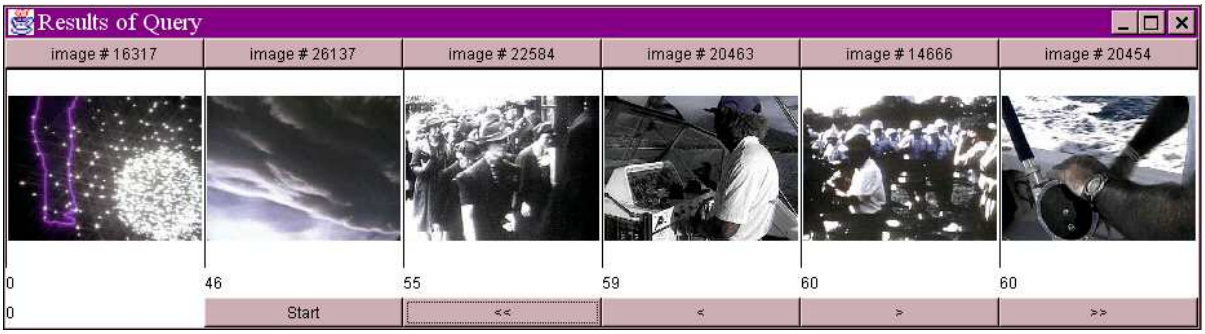


Figure 3: Poor result of a query with a color-based distance measure

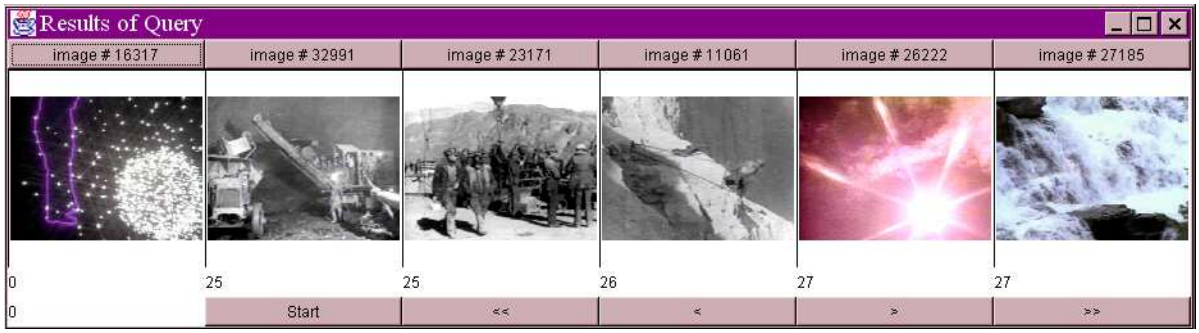


Figure 4: Poor result of a query with a texture-based distance measure.

These operations are all invariant under inequality. That is,

$$x_1 \leq y_1, x_2 \leq y_2 \Rightarrow x_1 + x_2 \leq y_1 + y2$$

$$x \leq y, c \geq 0 \Rightarrow cx \leq cy$$

$$x_1 \leq y_1, x_2 \leq y_2 \Rightarrow Min(x_1, x_2) \leq Min(y_1, y_2)$$

$$x_1 \leq y_1, x_2 \leq y_2 \Rightarrow Max(x_1, x_2) \leq Max(y_1, y_2)$$

We later show how to take advantage of this invariance to apply triangle-inequality-based pruning algorithms to distance measures that are combined together using the above operations.

Weighting and addition are already commonly used in both commercial and research systems [21, 17]. The reason for Max and Min is that they enable queries that we expect to be useful. Max, for example, enables tight searches. "I want a match on color and texture and position and shape." Min enables more speculative searches similar to those required by the data mining community: "I want a match on color or texture or position or shape." Figures 3, 4, and 5 demonstrate the utility of combining distance measures. The query image is a particularly difficult one for the distance measures in the system. In Figure 3, a color distance measure returns unsatisfactory results. A texture measure proves no better as shown in Figure 4.

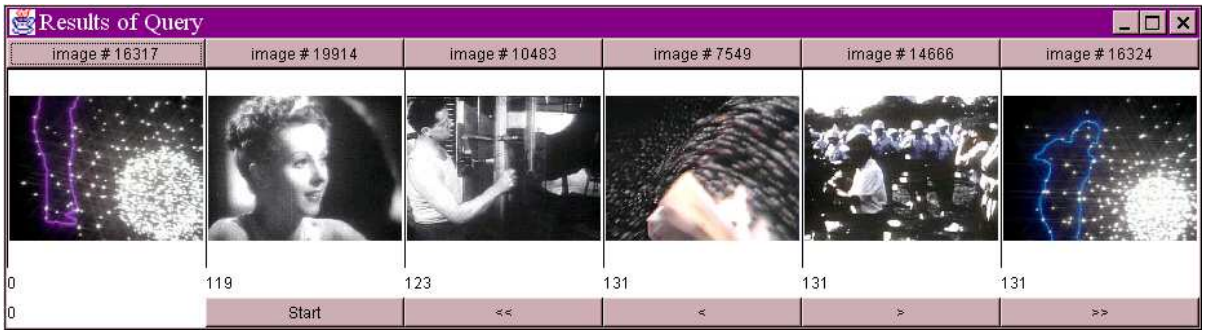


Figure 5: Improved result of query with a combination distance measure.

Yet a distance measure based on the sum of the color and texture measures returns a viable match to the query in Figure 5.

Fagin[15] has also proposed extending multimedia queries to boolean combinations, using Min and Max to implement them. We experimented with taking powers of distance measures[6], but decided that the functionality was too unintuitive with little apparent gain in utility.

3.3 Query Strategies: Threshold and Best-Match

Given an image database $\mathcal{S}$, a query image Q, and a distance measure d, there are two main types of searches. One can request all images $I \in \mathcal{S}$ such that the distance $d(Q, I)$ is not greater than some given threshold value t. We label this task to be a *threshold* style query. The second method is to find the image or images in $\mathcal{S}$ which minimize $d(Q, I)$. We label this task to be a *best-match* style query. Note that we can naturally extend best-match queries to find the n best matches.

3.4 How to Query the Database

User understanding of the distance measures is a problem with any content-based retrieval system. Our proposed set of operations adds a great deal of complexity to the system. There is a need for an additional layer to bridge the gap between user understanding and system capabilities. Possibilities include example-based learning and natural language translation. It may be that different image database domains will require different interfaces.

Standard comparison techniques usually involve comparing features which have been computed over entire images. However, one can consider the utility of comparing corresponding sections of images. For example, the user may wish to find images whose centers match the center area of a query image. Or the user may care about color in the top half of an image, but care about texture in the lower half. We can give

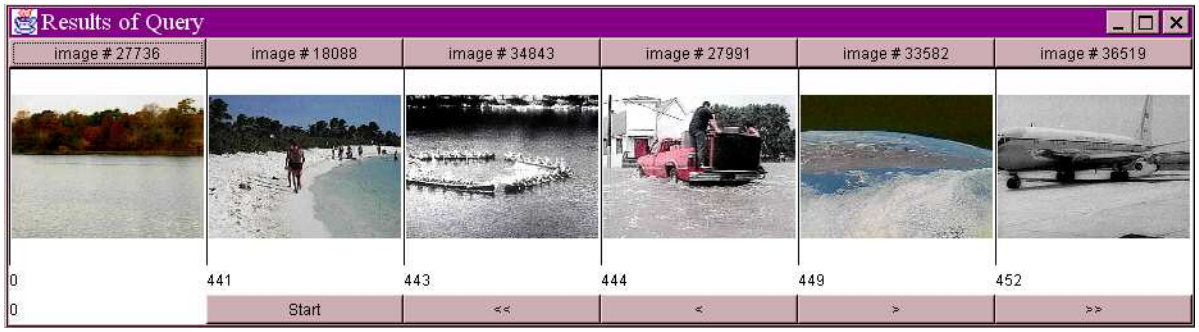


Figure 6: Looking for water and trees, but not necessarily fall tree colors

the user some control over the locations which the user cares about by using *gridded* distance measures. The idea is to break up each image into a grid of rectangles of the same proportions as the original image. Given image I, let $I_{r,c,x,y}$ be the (x, y) rectangle of I which has been broken up into a (r, c) grid of rectangles. Given a distance measure d, let $d_{r,c,x,y}(I, Q)$ be defined as $d(I_{r,c,x,y}, Q_{r,c,x,y})$. That is, $d_{r,c,x,y}$ "pretends" that the chosen rectangles are the entire images to be compared.

Using gridded distance measures gives the user the capability to create queries with some positional semantics. For example, a user could express the following queries with combinations of gridded distance measures:

- Match the lower left corner by color.
- Match the top half by color and texture and the bottom by texture.
- Match the center by color and all of the image by texture.

Assuming a $3x3$ grid, color measure C and texture measure T, this last query would be expressed in our formulation by $Max(C_{3,3,1,1}, T)$. The distance measure $C_{3,3,1,1}$ compares the (1,1) or central rectangle of the query Q to the central rectangle of the database image I as illustrated in Figure 7. The texture measure T compares all of Q to all of I. The Max operation implements the equivalent of a logical AND operator.

Our system employs such gridded measures. We have found it useful in retrieving matches to images that have different qualities in different areas of the image. In Figure 6, we are looking for matches to a scene with water in the foreground and trees in the background. The picture was taken in fall as can be discerned by the autumn coloring of the leaves. However, we don't wish to restrict the returned set to autumn images. Therefore, we choose to match on texture in all four quadrants of the image, but only match on color in the bottom half of the image. Not including the query image itself, two of the returned images have water in the lower half and trees in the upper half. However, the trees in the returned images do not share the fall coloring of the query image. Given that T was our texture measure and C was our color measure, the formula for the compos-

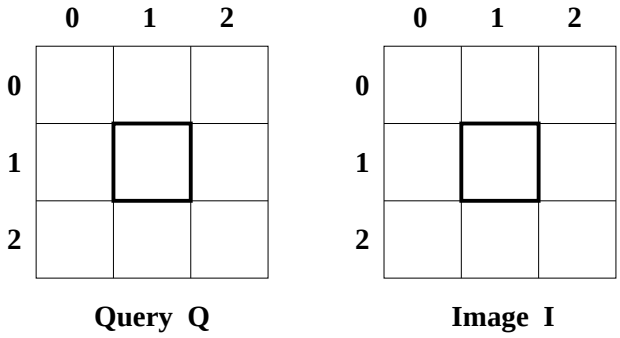


Figure 7: Gridded distance measures are defined on a particular grid rectangle of the query and the corresponding grid rectangle of the image being compared. The highlighted grid rectangle is the (1,1) rectangle of the (3,3) grid. Distance measures with subscript (3,3,1,1) reference this grid rectangle.

ite distance measure used in this search was $2(Max(T_{2,2,0,0}, T_{2,2,0,1}, T_{2,2,1,0}, T_{2,2,1,1})) + Max(C_{2,2,0,1}, C_{2,2,1,1})$. This formula is ample illustration of the need for good user interfaces. It is not something a user would wish to type.

One could imagine extending gridded distance measures to arbitrary shapes. One can further imaging extending this to non-constant shapes such as those calculated by segmentation. Here are some other examples using these extensions:

- Match the largest segments by color and the next largest segments by texture.
- Match the center circular area by color and the largest segments by color and texture.

4 Indexing with the Triangle Inequality

There are several schemes in the literature [2, 3, 4, 6, 10] that take advantage of the triangle inequality to reduce the number of direct comparisons in a threshold style database search. The intuition behind all the schemes is that the distance between two objects cannot be less than the difference in their distances to any other object.

The indexing scheme and algorithm described here, defined informally as the *bare-bones triangle inequality algorithm*, outputs a value for each database image corresponding to a lower bound on the distance between that image and the query image. This set of values can be used in several different ways. One can use the lower bounds to discard images that are shown to be too far from the query image to be a potential match. Alternatively, one can sequence the images in increasing order of their calculated lower bounds. Experimental evidence suggests that the first images in such a sequence are the ones most likely to be the best matches to the query. In our examples below, we assume that a threshold is applied to the lower

bounds on the images, rather than an ordering.

The bare-bones triangle inequality algorithm is probabilistic in nature. The lower bounds are guaranteed to be correct, but the ordering of the images based on their lower bounds is not guaranteed to be the same as the ordering of the images based on their true distances to the query image. To improve accuracy, one can add an additional step in which the subset of database images with the smallest lower bounds are compared directly to the query image to obtain their true distances. Our experiments have shown that good results can be obtained with direct comparison of only a tiny fraction of the original database to the query.

4.1 Indexing with a Single Distance Measure

Let I represent a database object, Q represent a query object, K represent an arbitrary fixed object known as a *key*, and d represent some distance measure that is a metric. As d is a pseudo-metric, the two triangle inequalities, $d(I,Q) + d(Q,K) \geq d(I,K)$ and $d(I,Q) + d(I,K) \geq d(Q,K)$, must be true. We can combine them to form the following inequality, which places a lower bound on $d(I,Q)$:

$$d(I,Q) \geq |d(I,K) - d(Q,K)| \tag{1}$$

Thus, by comparing the database and query objects to a third *key* object, a lower bound on the distance between the two objects can be obtained. We define $l(d,K,I,Q) = |d(I,K) - d(Q,K)|$ to be equal to this lower bound on $d(I,Q)$. We further shorten $l(d,K,I,Q)$ to $l(d,K)$ when there is no confusion as to the identity of I and Q.

Burke and Keller[10] first proposed the idea of using the triangle inequality to reduce comparisons. This idea was used by Uhlmann[44] to create vantage-point trees, where each node in a tree corresponds to a carefully chosen key. The subtree rooted at that node was partitioned according to the distance of the leaf elements to the key. Berman[4] and Baeza-Yates, *et. al.*[2] separately refined Burke and Keller's algorithm by creating a single set of keys to use for all the objects in the database. The single key method differs from vantage-point trees in that it reduces the total number of key comparisons at the expense of increasing the number of fast operations. Barros, *et. al.*[3], successfully used a single set of keys and the triangle inequality in a real image database. What follows is a description of the general algorithm used with a single set of keys:

Equation (1) can be extended naturally by substituting a set of keys $\mathcal{K} = (K_1, \ldots, K_M)$ for K as follows:

$$d(I,Q) \geq max_{1 \leq s \leq M} |d(I,K_s) - d(Q,K_s)| \tag{2}$$

We can see that this inequality is valid by noting that $d(I,Q) \geq |d(I,K_s) - d(Q,K_s)|$ for all values of s. We define $l'(d,\mathcal{K},I,Q)$ to be equal to the lower bound on $d(I,Q)$ found by using equation (2). As before, we shorten $l'(d,\mathcal{K},I,Q)$ to $l'(d,\mathcal{K})$ where possible.

Table 1: Sample Database and Stored Distances

Image	$d(I, K_1)$	$d(I, K_2)$
I_1	2	8
I_2	4	4
I_3	1	5
I_4	6	9
I_5	4	1
I_6	7	3

Consider a large set of database objects, $\mathcal{S} = \{I_1, \ldots, I_n\}$ and a much smaller set of key objects, $\mathcal{K} = \{K_1, \ldots, K_m\}$. Pre-calculate $d(I_s, K_t)$ for all $\{1 \leq s \leq n\} \times \{1 \leq t \leq m\}$. Now consider a request to find all database objects I_s such that $d(I_s, Q) \leq t$ for some query image Q and threshold value t. We can calculate lower bounds on $\{d(I_1, Q), \ldots, d(I_n, Q)\}$ by calculating $\{d(Q, K_1), \ldots, d(Q, K_m)\}$ and repeatedly using equation (2). If we prove that t is less than $d(I_s, Q)$, then we eliminate I_s from our list of possible matches to Q. After the elimination phase, we may search linearly through the uneliminated objects, comparing each to Q in the standard fashion. This algorithm involves $m + u$ distance measure calculations, and $O(mn)$ simple (constant cost) operations, where u is the number of uneliminated objects. The hope is that $m + u$ is sufficiently smaller than n to result in an overall time savings. The pseudocode for setting up the index structure, calculating a lower bound for a query and an image, and retrieving images for a query with a single distance measure is given in Figure 8.

Example of indexing with a single distance measure

Let our sample database be an image database composed of the images $\mathcal{S} = (I_1, \ldots, I_6)$. Our keys are images $\mathcal{K} = (K_1, K_2)$. To initialize the database for distance measure d, we calculate $d(I_s, K_j)$ for all s, j as shown in Table 1. Now suppose we wish to find all images I_s in our database such that $d(I_s, Q) \leq 2$ for some query object Q. We calculate $d(K_1, Q) = 3$ and $d(K_2, Q) = 5$. We subtract 3 from each element in the first column in Table 1 and subtract 5 from each element of the second column. We then place the absolute values of the results in Table 2.

By examining the values of $l'(d, \mathcal{K}, I_s, Q)$ for $1 \leq s \leq 6$, we see that only I_2 and I_3 can possibly be within a distance of 2 to query Q. Thus, only $d(I_2, Q)$ and $d(I_3, Q)$ need to be calculated to determine all close matches to Q. The efficiency of the algorithm is highly dependent on the selection of keys, the relative expense of distance measure calculation, and the statistical behavior of the distance measure over the set of database objects.

```
procedure Index_Single(ImageSet[],KeySet[])
{
for X := 0 to SizeOf(ImageSet)-1
   for Y := 0 to SizeOf(KeySet)-1
      DistanceMatrix[X][Y] := d(ImageSet[X],KeySet[Y]);
}


procedure Calc_Lower_Bound(Query,i,DistanceMatrix)
'Query' is the Query image
'i' is the database Image's index
{
for X := 0 to SizeOf(KeySet)-1
   QKDist[X]=d(Query,KeySet[X]);
Distance := 0;
for Y := 0 to SizeOf(KeySet)-1
   if (Abs(QKDist[Y]-DistanceMatrix[i][Y]) > Distance)
   then Distance = Abs(QKDist[Y]-DistanceMatrix[i][Y]);
}


procedure Retrieve_Single_With_Threshold(Query,t)
{
R = null;
for Y := 0 to SizeOf(ImageSet)-1
   if Calc_Lower_Bound(Query,Y,DistanceMatrix) ≤ t
   then R = R ∪ ImageSet[Y] ;
return R;
}
```

Figure 8: Pseudo-code for the indexing and retrieval procedures using the triangle-inequality algorithm with a single distance measure.

Figure 9 shows a real example from our system. In this case, we used a simple color measure to find close matches to the leftmost image. The possible threshold range for this color measure is 0 for an exact match to 1000 for no color match at all. Using a threshold of 100, we eliminated all but 19 out of 37,748 images as

Table 2: Calculating minimum distances of each image in a database to query image q by use of the triangle inequality

Image	$l(d, K_1)$	$l(d, K_2)$	$l'(d, \mathcal{K})$
I_1	1	3	3
I_2	1	1	1
I_3	2	0	2
I_4	3	4	4
I_5	1	4	4
I_6	4	2	4

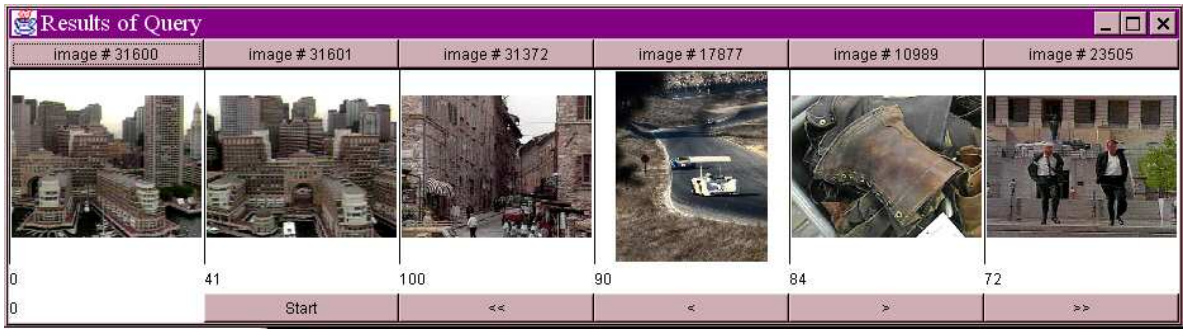


Figure 9: An example from our system using a simple color measure. The query image is on the far left.

potential matches to the query.

The Berman[4] and Baeza-Yates[2] papers also introduced the idea of combining all the pre-calculated distances into a *trie*. Use of this data structure can, in some circumstances, reduce the number of simple calculations to below $O(nm)$. Berman showed an $O(u \log n + n^{\epsilon})$ expected upper bound on simple calculations for the string matching problem using the Hamming distance on random binary strings, where ϵ is a complicated function always less than one. We discuss this data structure, the *triangle trie*, in Section 9.

4.2 Indexing with Multiple Distance Measures

We extended the above scheme to work with combinations of distance measures[6]. The intuition is that lower bounds on the distance between two objects for distance measures d_1 and d_2 can often be used to calculate a lower bound between the objects for distance measure d when d can be calculated as a combination of d_1 and d_2.

Let $D = \{d_1, \ldots, d_p\}$ be a set of distance measures. These distance measures will be known as the *base* distance measures. Let $\mathcal{K}' = \{\mathcal{K}_1, \ldots, \mathcal{K}_p\}$ be a sequence of sets of keys, one set of keys for each distance

measure. Note that each set may have a different number of keys and that the sets may or may not intersect. Let $L(D, \mathcal{K}', I, Q)$ be the set of lower bounds $l'(d_s, \mathcal{K}_s, I, Q)$ calculated from equation (2) for each pair $(d_s \in D, \mathcal{K}_s \in \mathcal{K}'), 1 \leq s \leq p$.

Now consider a new distance measure d' that is of the form

$$d'(I, Q) = f(d_1(I, Q), \ldots, d_p(I, Q))$$

where f is monotonically non-decreasing in its parameters. For example, f might describe a weighted sum of the base measures, or even combinations of minimums and maximiums of sets of the base measures. Since $l'(d_s, \mathcal{K}_s, I, Q) \leq d_s(I, Q)$ for all s, substituting $l'(d_s, \mathcal{K}_s, I, Q)$ for each instance of $d_s(I, Q)$ gives us

$$d'(I, Q) \geq f(l'(d_1, \mathcal{K}_1, I, Q), \ldots, l'(d_p, \mathcal{K}_p, I, Q)).$$

Thus we can calculate a lower bound on $d'(I, Q)$ given lower bounds on the base distance measures. We can then either order the database images based on these lower bounds or threshold out database images as candidates for matches to the query image. Here we note that the operations on distance measures described earlier–Addition, Weighting, Max, and Min–can be combined to form monotonically non-decreasing functions. We also note that the power function $f(x) = cx^e, c \geq 0, e \geq 0$ is monotonically non-decreasing.

The pseudocode for setting up the index structure, calculating a lower bound for a query and an image, and retrieving images for a query with multiple distance measures is given in Figure 10.

Example of Indexing with Multiple Distance Measures

Let our database be a set of images $(I_1, \ldots, I_6)$, with two base distance measures (d_1, d_2) and two sets of keys, $\mathcal{K}_1 = (K_{11}, K_{12})$ and $\mathcal{K}_2 = (K_{21}, K_{22})$. We pre-calculate $d_s(I_t, K_{su})$ over all s, t, u to obtain Table 3. Now suppose we have query Q and distance measure $d'(X, Y) = d_1(X, Y) + 3d_2(X, Y)$. We wish to find all objects I in the database such that $d'(I, Q) \leq 10$. We calculate $d_1(K_{11}, Q) = 3$, $d_1(K_{12}, Q) = 5$, $d_2(K_{21}, Q) = 2$, and $d_2(K_{22}, Q) = 8$. Taking the absolute differences between these values and the values in Table 3, we produce the $l(d_s, K_{su})$ values over s, u and combine them to calculate $l'(d_1, \mathcal{K}_1)$ and $l'(d_2, \mathcal{K}_2)$. We then combine these results to produce the $l'(d', \mathcal{K}')$ values. The $l(d_s, K_{su})$ and $l'(d', \mathcal{K}')$ values are shown in Table 4. In this case, $l'(d', I_5, Q, \mathcal{K}') \leq 10$ and $l'(d', I_2, Q, \mathcal{K}') \leq 10$. Thus, I_2 and I_5 are returned as possible matches, with the remaining images eliminated.

We can modify the algorithm to return the best match. In this case, the images are returned in increasing order of their $l'(d', \mathcal{K})$ values as $(I_5, I_2, I_1, I_4, I_3, I_6)$. Direct comparisons could then be made from the query image to some prefix of this set to validate the best image.

```
procedure Index_Multiple(ImageSet[],KeySets[][],d[]())
{
for X := 0 to SizeOf(d)-1
   for Y := 0 to SizeOf(ImageSet)-1
      for Z := 0 to SizeOf(KeySet[X])
         DMatrix_X[Y][Z] = d_X(ImageSet[Y],KeySet[X][Z]);
}


procedure Calc_Lower_Bound_Multiple(Query,i,f())
'Query' is the Query image
'i' is the database Image's index
'f()' is the composite distance measure, as in d(Q, I) = f(d_1(Q, I), d_2(Q, I), ...)
{
for X := 0 to NumDistanceMeasures
   L[X] = Calc_Lower_Bound(Query,I,d_X());
return f(L[1],...,L[NumDistanceMeasures]);
}


procedure Retrieve_Multiple_With_Threshold(Query,t)
{
R = null;
for Y := 0 to SizeOf(ImageSet)-1
   if Calc_Lower_Bound_Multiple(Query,Y,DistanceMatrix) ≤ t
   then R = R ∪ ImageSet[Y] ;
return R;
}
```

Figure 10: Pseudo-code for the indexing and retrieval procedures using the triangle-inequality algorithm with multiple distance measures.

Figure 11 shows a real example from our system with multiple distance measures. The query is the leftmost image. We used a measure representing a weighted sum of a color measure and a texture measure, with an appropriate threshold. We eliminated all but 290 out of $37,748$ images as potential matches to the query. This example also illustrates a best-match example, as the returned images are ordered by their

Table 3: Sample Database and Stored Distances With Multiple Distance Measures

Image	$d_1(K_{11}, I)$	$d_1(K_{12}, I)$	$d_2(K_{21}, I)$	$d_2(K_{22}, I)$
I_1	2	8	5	15
I_2	4	4	3	6
I_3	1	5	12	9
I_4	6	9	10	8
I_5	4	1	2	8
I_6	7	3	15	15

Figure 11: Results of a query using a combination of a color measure and a texture measure. The query is on the far left.

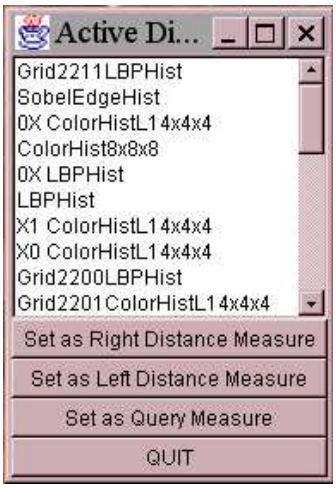
calculated distances to the query.

5 Fast Image Database System: A Prototype for Testing

FIDS, the Fast Image Database System, is a prototype content-based image retrieval system, which currently has over thirty seven thousand images. It contains a number of distance measures based on color,

Table 4: Calculating lower bounds on $d' = d_1 + 3d_2$ by use of triangle inequality

Image	$l(d_1, K_{11})$	$l(d_1, K_{12})$	$l(d_2, K_{21})$	$l(d_2, K_{22})$	$l'(d', \mathcal{K}')$
I_1	1	3	3	7	$3 + 3 * 7 = 24$
I_2	1	1	1	2	$1 + 3 * 2 = 10$
I_3	2	0	10	1	$2 + 3 * 10 = 32$
I_4	3	4	8	0	$4 + 3 * 8 = 28$
I_5	1	4	0	0	$4 + 3 * 0 = 4$
I_6	4	2	13	7	$4 + 3 * 13 = 43$



(a) List of available distance measures

(b) New distance measure creation window

Figure 12: The LIST and BUILD windows for the Fast Image Database System

texture, and feature detection. Position matching is implemented by "gridding" the distance measures and performing distance calculations on corresponding sub-rectangles.

The FIDS interface is shown in Figures 12 and 13. There are 4 windows. The LIST window is a list of available distance measures. When the user clicks on a distance measure, a brief description is offered. The user can select distance measures from this window and copy them to either the BUILD window or the QUERY window. The BUILD window contains a smaller set of distance measures. One creates a new distance measure by performing the following four activities:

1. Selecting two or more measures in the BUILD window,
2. Selecting one of the three operations "SUM", "MIN", or "MAX",
3. Entering the appropriate weights
4. Entering a name for the new distance measure,
5. Pressing the 'build' button.

This new distance measure is then added to the LIST window.

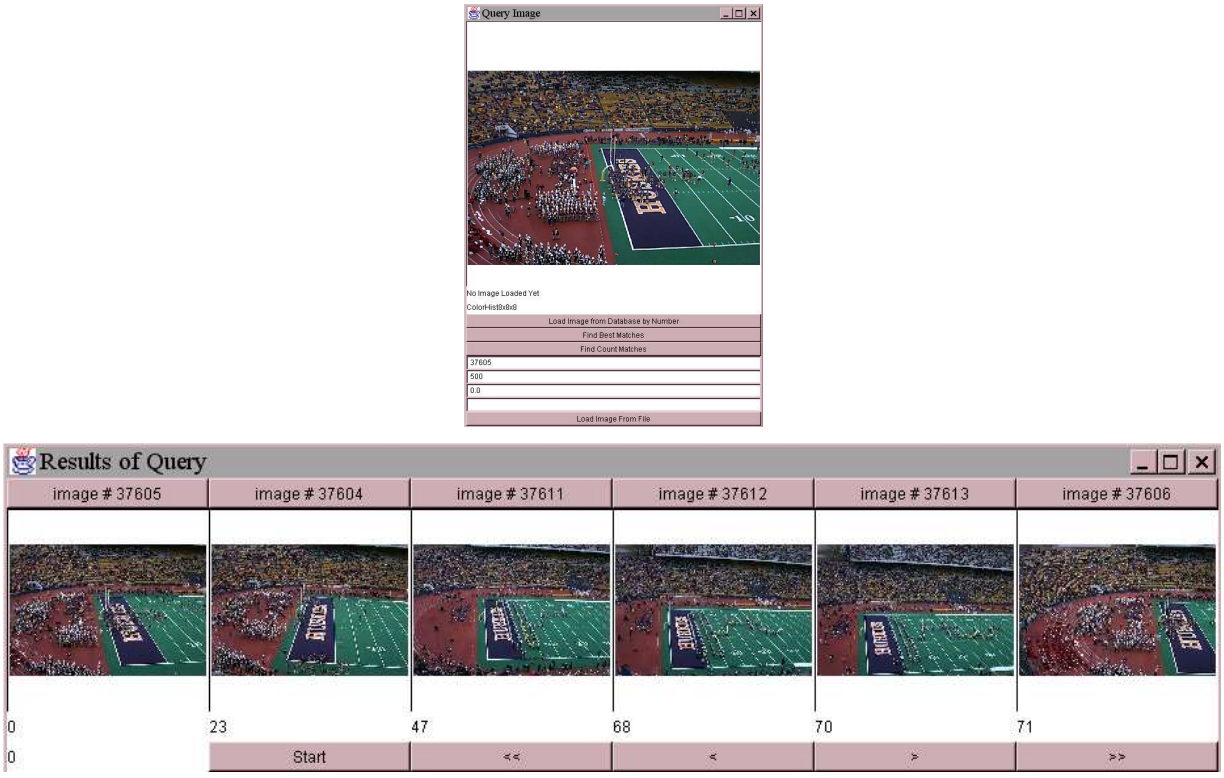


Figure 13: The QUERY and RESULTS windows for the Fast Image Database System

The QUERY window contains the query image and the chosen distance measure. The user runs the query by pressing the run-query button. The RESULTS window contains the sixteen images judged to be the closest by the system. In the full FIDS system, there are three ways to get results. The first way is to order the images by the lower bounds calculated from the triangle inequality. This method is the fastest as no direct comparisons of the query to the database are made. In the second method, the user selects how many images to verify. An image is verified if its true distance to the query is less than the triangle-inequality-derived lower bounds of all the remaining images. The algorithm calculates the true distances to the returned images in lower-bound-derived order until the required number of images have been verified. Optionally, one can set an upper limit on the number of images that need to be verified. The third method is to perform direct comparisons on all the images. In Berman and Shapiro[6], verification with an upper limit of a few percent of the database almost always returned the correct images.

The user can click on any of the images in the RESULTS window to move that image to the QUERY window. This provides an efficient browsing mechanism for groups of similar images.

6 Distance Measures in FIDS

Experiments conducted using the FIDS system were performed with various subsets of the following distance measures. They are labeled in the experiments using the labels in boldface below:

- **Color:** The color histogram distance measure was first published by Swain and Ballard[41]. Consider RGB space as a three-dimensional cube with the axes labeled Red, Green, and Blue. We quantize this cube by breaking it up into a set of sub-cubes. For our experiments, we created a $4 \times 4 \times 4$ set of sub-cubes of equal size. Given an image I, we label each sub-cube with the fraction of I that has any of the colors contained in that sub-cube. The distance between two images is the sum of the absolute differences between the values in their corresponding sub-cubes, known more formally as the L_1 distance. For some experiments, we divided the cube into an $8 \times 8 \times 8$ set of sub-cubes. To prevent confusion, we note the dimensions of the sub-cubes in some cases. If no dimensions are given, the $4 \times 4 \times 4$ set was used.

- **LBP:** The Local Binary Partition texture measure is a standard texture measure with very good performance[45]. For each pixel P, the 8 neighbors are examined to see if their intensity is greater than P's intensity. The result is encoded as an 8 bit binary number. A histogram of these numbers is created for each image. Two images are compared by taking the L_1 distance between their corresponding histograms.

- **Sobel:** The Sobel edge detector was run over the greyscale version of each image. A histogram of the values of the resultant matrix was calculated. The distance between two images was the L_1 distance of the histograms. Note that in the standard Sobel edge detector, a threshold value is used to create a binary edge image. We did not use a threshold value, but simply used the results of the Sobel convolution.

- **Wavelet:** This measure is based on the wavelet decomposition distance measure developed by Jacobs, *et. al.*[24] A Haar wavelet decomposition of the images is calculated, resulting in a set of coefficients. These coefficients are then weighted and a distance is calculated between two images based on the difference between corresponding coefficients. As the wavelet decomposition is only defined for images of size $2^n \times 2^n$, we resized all the images to 32×32 before calculating the coefficients.

- **Flesh:** We implemented a flesh detector based on the work of Fleck and Forsyth[16]. For each image, we calculated the percentage of pixels that contained flesh according to the detector. The distance between two images is simply the difference in this percentage. Technically, this is equivalent to a 2-bin histogram with an L_1 distance calculation.

- **Grid Color, Grid LBP, Grid Flesh, Grid Wavelet, Grid Sobel** We implemented a positionally dependent version of each of the distance measures described above. Rather than comparing whole

images, gridded versions of distance measures only compare chosen pieces of the images. For these experiments, the gridded versions of the distance measures compare the lower left quarter of each image.

- **Horizontal Color and Horizontal LBP** For these two distance measures, each image was split into three equal-sized horizontal pieces. Two images were compared by averaging the Color or LBP distance between the corresponding pieces.

- **Vertical Color and Vertical LBP** These distance measures are similar to the horizontal measures above, except that the images were split vertically.

- **SUM**$(d_1, d_2, \ldots)$ A distance measure with the suffix SUM represents a composite measure that is the sum of the enclosed distance measures. That is, the distance between the images in question is computed for each enclosed measure, and the sum of these distances is returned.

- **MIN**$(d_1, d_2, \ldots)$ A distance measure with the suffix MIN represents a composite measure that is the minimum of the enclosed distance measures. That is, the distance between the images in question is computed for each enclosed measure and the minimum value is returned.

- **MAX**$(d_1, d_2, \ldots)$ A distance measure with the suffix MAX represents a composite measure that is the maximum of the enclosed distance measures. That is, the distance between the images in question is computed for each enclosed measure and the maximum value is returned.

In some experiments, one or more base distance measures were combined. An equal weighting of distance measures was always used in those cases.

7 Performance of the Triangle Inequality Algorithm

We first measured the speed of the bare-bones triangle inequality algorithm. The process by which a system using this algorithm returns a set of matches to a query can be broken into four steps:

- **Step 1:** The system extracts relevant features from the query image. In our system, this step takes from a fiftieth to a quarter of a second, depending on the distance measure. Table 5 shows this range in the second column.

- **Step 2:** The system calculates the distance from the query image to each of the key images. For the basic distance measures on our system, this step takes from about a microsecond per image to more than four-fifths of a millisecond per image. The third column of table 5 shows the values.

- **Step 3:** The system calculates the lower bound distances from the query image to each of the database images.

Distance Measure	Feature Extraction Time in Seconds	Distance Calculation Time in ms per 1000	Number Of Distance Calculations per Second
Color (4x4x4 RGB)	.12	46	2174
Color (8x8x8 RGB)	.13	37	2669
LBP	.04	28	3623
Flesh	.25	.12	833333
Sobel	.20	4.1	24937
Wavelet	.02	863	115

Table 5: Feature extraction and distance calculation time for representative distance measures used in FIDS on a Pentium Pro 200 mhz PC.

- **Step 4:** The system returns the images with the smallest lower bound distances calculated in the previous step.

The third step above is the deciding factor on throughput. The timing of the other steps is relatively stable across database sizes, although the number of keys in step two which are necessary for adequate performance will tend to increase as the number of images in the database increases.

We measured a time of approximately 4 milliseconds to perform the third step on 1,000 images if 35 keys are used. This represents a throughput of well over two hundred and fifty thousand images per second, once the feature extraction and query-key comparisons are finished. By comparison, the final column of Table 5 gives the throughput per second for each distance measure given a system that stores the features for its images.

7.1 The Accuracy of the Triangle-Inequality Algorithm

To evaluate the efficiency of using the triangle inequality to return close matches to queries, we performed experiments using both single and composite distance measures. Given a query and distance measure, the system returns all the images in the database ordered by calculated lowest bounds on their distances to the query. We use terms such as *lower-bound sequence* or *returned ordering* to refer to the system's output. On the other hand, there is the *true sequence* or *true ordering*, which consists of all the images in the database ordered by their true distance to the query. The hope is that the images which are at the front of the true ordering are also at or near the front of the lower-bound sequence. Measuring the placement of close matches was the main goal of these experiments.

In order to test the quality of the returned ordering, some ground truth was required. To achieve this ground truth, we examined the database by hand and selected 51 pairs of similar images. For each pair,

Distance Measure Class	100% in first 400	90-99% in first 400	80-89% in first 400
Single Measures	3	2	3
AND'ed Measures	7	12	1
OR'ed Measures	16	1	3
SUM'ed Measures	10	10	0
Totals	36	25	7

Table 6: Summary performance of distance measures using full set of validated images. There were no distance measures in which less than 80% of the target images were returned in the first 400 images out of 37,748.

we queried the system with one of the images and measured the position of the other *target* image in the returned sequence. We then calculated the true position of the target image and compared the returned positions to the true positions.

We measured the fraction of matches that were returned as part of the first 25 images. This corresponds to a scenario where a user might ask for several images to be returned for closer examination. We further measured the fraction of matches that were returned within the first 400 matches, corresponding to a scenario in which the user or system is willing to do more work to find a closer match. We also measured the fraction of images that were returned in their precise position. As we selected matching pairs of test images by hand, we had to deal with the possibility that the test image pairs didn't truly match with respect to the distance measures on the system. Our experiments were designed to measure the ability of the system to find the closest images to queries, so we excluded an image pair from tests on a distance measure if it was discovered that the target image was not one of the five actual closest images to the query image for that distance measure. Our database contained 37,748 images, and we used 35 keys for each base measure.

7.2 Results for the Bare-Bones Triangle Inequality Algorithm

Tables 6 and 7 show the results of experiments using the full set of valid matches. Depending on the distance measure, from 50% to 100% of all the matches were within the first 25 images returned by the system. In almost every case, over 90% of the matches were within the first 400 returned images. In a second set of similar experiments, we simulated a user with a more restricted definition of closeness. For each distance measure, the query-target pairs were sorted in increasing order of query-target distance. Only the first half of these lists were used to compute these results. The results were markedly better for the experiments using the closer half of the valid matches. Tables 8 and 9 show the results of experiments with the restricted set of valid matches. In this case, only 9 distance measures out of the 68 tested measures had less than 80% of

Distance Measure Class	100% in first 25	90-99% in first 25	80-89% in first 25	70-79% in first 25	50-69% in first 25
Single Measures	1	0	1	4	2
AND'ed Measures	2	2	10	6	0
OR'ed Measures	6	0	0	12	2
SUM'ed Measures	4	6	10	0	0
Totals	13	8	21	22	4

Table 7: Summary performance of distance measures using full set of validated images, showing how many distance measures of each type returned the target images in the first 25 images.

Distance Measure Class	100% in first 400	90-99% in first 400	80-89% in first 400	70-79% in first 400
Single Measures	6	0	2	0
AND'ed Measures	20	0	0	0
OR'ed Measures	18	1	0	1
SUM'ed Measures	20	0	0	0
Totals	64	1	2	1

Table 8: Summary performance of distance measures using closer pairs of validated images. Almost every distance measure returned all targets within the first 400 images.

the target images returned within the first 25 images. Furthermore, in 64 out of 68 distance measures, every single one of the matches were returned within the first 400 images.

Distance Measure Class	100% in first 25	90-99% in first 25	80-89% in first 25	70-79% in first 25	50-69% in first 25
Single Measures	2	1	2	1	2
AND'ed Measures	13	5	0	2	0
OR'ed Measures	6	0	10	2	2
SUM'ed Measures	16	3	1	0	0
Totals	37	9	13	5	4

Table 9: Summary performance of distance measures using closer pairs of validated images, showing how many distance measures of each type returned the target images in the first 25 images.

8 Key Selection

We begin this section with a discussion of what makes a good single key. Later, we discuss the choice of keys in combination. We make the simplifying assumption that all distances are within the range of 0 to 1, inclusive.

8.1 Good Keys for *Threshold* Style Queries

Consider database image I, query image Q, key image K, distance function d. We say that key K *separates* Q from I for value v if $|d(I,K) - d(Q,K)| > v$. Suppose that $d(I,Q) > t$ for some threshold t. The triangle inequality implies that the value $|d(I,K) - d(Q,K)|$ can range from 0 to $d(I,Q)$. Key K will eliminate image I as a candidate match to Q only if it separates I from Q for value t. The purpose of the algorithm is to eliminate as many non-matching candidate images as possible through key comparison. Thus, a good key will eliminate more candidate images than a poor key. The concept of separation described above motivates the following discussion.

Given a set of database images $\mathcal{S}$, distance measure d, and key K, we can compute a density function f on $d(I,K), I \in \mathcal{S}$. Since we do not know the queries in advance, we make the simplifying assumption that the queries are taken from the database images and ignore exact matches in our searches. Given threshold t, we can calculate the fraction of images that K will separate from a random query by looking at this density function. For example, if all of the area of the density function lies in a narrow range $(x, x+e), e < t$, as shown in Figure 14a, then K will never separate any query from any image in the database. If the density function has a uniform distribution, as shown in Figure 14b, then for $0 < t < 1/2$, $P(K$ separates I from $Q) = (1-t)^2$. If the density function is multipolar, with n equally sized narrow spikes separated by distance greater than t, as shown in Figure 14c, then $P(K$ separates I from $Q) = (n-1)/n$. If the density function has a Gaussian shape, as shown in Figure 14d, then, roughly speaking, greater standard deviations will indicate greater average separation of images by the key.

The issue gets more complicated when choosing several keys. Using keys K_1 and K_2 will be no better than just using K_1 if they both separate the same images from queries. The question of whether or not two keys separate the same images is computationally expensive to answer in the general case, but can be approximately answered by sampling. One can also use the fact that very similar keys will separate many of the same images and thus try to avoid keys that are too close together. For example, in a clusterable database, keys should come from different clusters. Indeed, the key selection algorithms with the best results make use of clustering and ensuring that different keys separate different images.

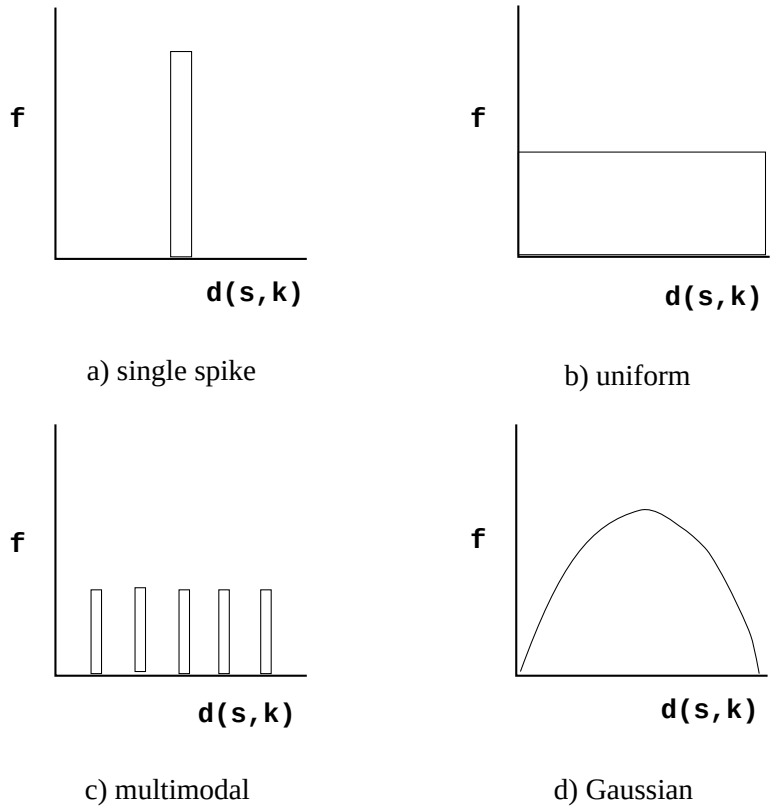


Figure 14: The shape of the density functions determines the performance of the keys.

8.2 Good Keys for *Best-Match* Queries

Given images I_1 and I_2, query Q and key K, assuming that $d(I_1, Q) < d(I_2, Q)$, key K *orders* I_1 and I_2 correctly if $l(d, K, I_1, Q) < l(d, K, I_2, Q)$. We can extend this definition naturally to sets of keys and multiple distance measures. Although our analyses were for *threshold* queries, the results were very good for *best-match* queries as well. Further analysis of keys optimized for *best-match* queries is an open area of research.

8.3 Algorithms for Key Selection

We examined five different algorithms for key selection: random keys, choosing keys by examining the variance of the density function, ranking by testing thresholding efficiency, a greedy thresholding algorithm, and a clustering algorithm. The algorithms assume a database $\mathcal{S}$ and a set of candidate keys.

RANDOM Our prototype image database system currently uses a set of (up to 35) keys chosen randomly and uniformly from the database itself. The triangle inequality algorithms give excellent performance compared to linear search even with random keys, so this is a natural benchmark against which to test the other algorithms.

VARIANCE Taking a subset $\mathcal{S}'$ of our database $\mathcal{S}$, we calculated the density function of $d(I, K), I \in \mathcal{S}'$ for each candidate key K. We selected those candidate keys which had the density functions with the greatest variance.

SEPARATION We examined our database by hand to find pairs of images that we judged to be approximate matches. The average distance between these pairs was calculated. This value t represented a potential threshold value that one might use in a query to find approximate matches. We then selected those candidate keys K which maximized $P(|d(I_1, K) - d(I_2, K)| > t)$ over all pairs $I_1, I_2 \in \mathcal{S}'$, where $\mathcal{S}'$ was a subset of our database $\mathcal{S}$.

GREEDY VARIANCE and SEPARATION may choose several keys which separate the same pairs of images. We thus modified SEPARATION to keep track of which pairs of images were separated by each key. The first key selected was the same as that selected by SEPARATION. The performance of the remaining keys were then recalculated to discount pairs of images already separated by the first key. This process was continued for subsequent keys until a preset number of keys was selected.

CLUSTER We used a simple clustering algorithm on the database. We selected the two database images K_1 and K_2 that were furthest apart, and used them as initial seeds for clustering. These two images were placed into our set of keys, and the remaining images were assigned to clusters based on their distances to the key images. We then found the image that was furthest from the current set of keys, added it to the set, and re-clustered the database on the updated set of keys. We continued this process until the correct number of keys were selected. Note that this algorithm differs from the others in that the selected keys came from the database itself.

8.4 Experiments

For our experiments, we collected two sets of images, one with 600 members, and one with 800 members. From each set, 100 images were chosen arbitrarily to be candidate key images. The remaining 500 and 700 images became the test database. The five algorithms were run on the candidate images to choose sets of 1 to 9 keys.

We queried the database against itself testing the system's performance using the keys chosen by the key selection algorithms. To eliminate exact matches, we temporarily removed each query image from the database. To test the performance of the keys on a *best-match* search, we determined the best match to the query and calculated its position in the ordering of the lower bounds. To test the performance of the keys on a *threshold* search, we counted the number of images separated from the query by a given threshold value. This threshold value was determind off-line by calculating the average distance between pairs of images known to be similar. For the RANDOM key selection algorithm, we ran the tests 10 times and averaged the

A	B	C	D	E	F	G
Color Histogram	GREEDY	0.9%	CLUSTER	1.1%	1.6%	1.8
LBP Texture	CLUSTER	1.2%	GREEDY	1.6%	2.3%	1.9
Horizontal Color	GREEDY	3.6%	CLUSTER	3.7%	4.6%	1.3
Vertical Color	GREEDY	2.5%	RANDOM	3.5%	3.5%	1.4
Horizontal Texture	CLUSTER	2.4%	GREEDY	3.7%	4.8%	2.0
Vertical Texture	CLUSTER	2.1%	GREEDY	2.2%	3.9%	1.9
Min(Color,Texture)	GREEDY	1.0%	CLUSTER	1.7%	2.2%	2.2
Color + Texture	GREEDY	1.0%	CLUSTER	1.1%	1.7%	1.7

Table 10: Best Algorithms for *best-match* with 9 keys on a database of 500 images

Column Headings

A: Distance measure

B: Best key selection algorithm

C: Average rank of best match using best algorithm

D: Second best key selection algorithm

E: Average rank of best match using second best algorithm

F: Average rank of best match using randomized key selection

G: Ratio of performance of randomized algorithm to best algorithm (F/C)

results.

8.5 Results of Key Selection Tests

We discuss the performance of the various key selection algorithms, first for *best-match* and then for *threshold*. As the rankings of the algorithms did not change much as a function of number of keys, we only show the results for 9 keys, the maximum number tested. As the performance of the algorithms on the two databases was very similar, we only show tables for the larger database.

8.5.1 Performance of Key Selection Algorithms for *Best-Match*

As is shown in table 10, CLUSTER provided the best keys for the texture measures, while GREEDY provided the best keys for the color measures and the combination color/texture measures. The second best algorithm was also always GREEDY or CLUSTER except for the *vertical texture* measure in the larger database, which had RANDOM as the second best.

Columns C, E, and F of the table show the average rank of the best match using the appropriate key

A	B	C	D	E	F	G
Color Histogram	GREEDY	98.5%	RANDOM	97.4%	97.4%	1.7
LBP Texture	GREEDY	83.2%	CLUSTER	81.2%	77.8%	1.3
Horizontal Color	GREEDY	94.4%	RANDOM	90.8%	90.8%	1.6
Vertical Color	GREEDY	93.2%	VARIANCE	90.0%	89.8%	1.5
Horizontal Texture	GREEDY	55.3%	CLUSTER	49.9%	47.5%	1.2
Vertical Texture	GREEDY	54.6%	VARIANCE	52.7%	47.2%	1.2
Min(Color,Texture)	GREEDY	97.8%	CLUSTER	96.4%	96.1%	1.8
Color + Texture	GREEDY	94.5%	CLUSTER	91.5%	91.2%	1.6

Table 11: Best Algorithms for *threshold* with 9 keys on Database of 700 images

Column Headings

A: Distance measure

B: Best key selection algorithm

C: Average percent of database eliminated using best algorithm

D: Second best key selection algorithm

E: Average percent of database eliminated using second best algorithm

F: Average percent of database eliminated using randomized key selection

G: Ratio of performance of randomized algorithm to best algorithm $(100\% - F)/(100\% - C)$

selection algorithm. For example, a 2% would mean that the true best match was ranked in the top 2% of the returned images. Column G represents the ratio of the number of images that would be examined using the RANDOM keys to the number of images that would be examined using the best discovered keys. Thus, in the first row, the best keys returned the closest match in the top 0.9% of the images. For the 700 image database, this translates to the top 6 or 7 images. The random keys returned the closest match in the top 11 images. If this database was a representative sample of a $700,000$ image database, then the number of images needed to be directly compared would be approximate 630 and 1120 respectively. On average, there was a 42% reduction in the number of images examined using the best discovered keys compared to using the random keys.

The overall performance of the algorithms was excellent. In the worst case for the database of 700 images, *Horizontal Color*, the closest match was ranked in the top 3.6%–that meant that only approximately 24 images had to be compared directly after pruning to find the best match. The database of 500 images had slightly worse performance with the average ranking of the best image ranging from 1.2% to 5.1%, again with the worst performance found in *Horizontal Color*.

8.5.2 Performance of Key Selection Algorithms for *threshold*

As table 11 shows, the GREEDY key selection method was the clear winner for *Threshold*, yielding the best performance for every distance measure. There was no clear second place algorithm–RANDOM, VARIANCE, and CLUSTER all appeared in second place for several measures. The GREEDY keys reduced the number of images that had to be directly compared by the RANDOM keys by 16% to 44%.

The second thing to note in Table 11 is the wide range of performance between distance measures. The triangle inequality algorithm thresholded 98.5% of the images for the *Color Histogram* distance measure, yet it only thresholded 54.6% of the images for the *Vertical Texture* distance measure. It is difficult to compare across distance measures since the distribution of distance values across pairs of images vary greatly from measure to measure. Especially interesting was the fact that using 5 keys for *Vertical Texture* resulted in a 53% thresholding. Thus, the additional four keys only eliminated an additional two percent of the database. In [2], Baeza-Yates, et. al.., demonstrated how, given a random model for database objects and keys, a logarithmic number of keys should threshold almost all of the database. For our experiments to have supported this, the addition of four more keys would have had to increase the thresholding from 53% to about 70%. That this didn't occur demonstrates that traditional models of randomness do not really apply to sets of real images.

8.6 Analysis

The performance of keys in image retrieval is intimately tied to the statistical behavior of the distance measures over the image set. At present, we have a limited understanding of this behavior; this limits the sophistication of our key selection algorithms. Thus, more research into the behavior of the distance measures is called for. A more complete set of distance measures will also be used in our future tests. Distance measures have been proposed for color, texture, shape[8], object presence[18], and object spatial relationships[7]. We would like to include representatives of each type of measure in our tests.

In our work, we selected keys from the database itself. The space of possible keys is huge– it is the space of possible images. We would like to take advantage of this freedom in some tractable manner. For example, it may be possible to construct artificial images that are excellent keys for either a specific database, or even for large image domains. Furthermore, our analysis contained the assumption that the query domain was similar to the database. This is not necessarily the case.

Even if we restrict our candidate keys to some random subset of n images, the number of possible subsets of m keys is exponential in m. There is no guarantee that there isn't some difficult to find set of keys that will prune the database far more than any other set. It may be that heuristics like those traditionally used

for NP-complete algorithms may be applicable for key selection.

Finally, there has been no published work on the proper number of keys to use for a database of a given size. There is a tradeoff between the elimination power of a set of keys and the execution time required to compare the query to the key set. Some queries may require more keys than other queries for good performance.

Of the algorithms tested, CLUSTER and GREEDY clearly gave the best results. The improvement over random key selection was up to a factor of two. As random key selection reduces *best-match* searches to just a few percent of the database, the use of random keys may be perfectly acceptable for smaller databases.

One thing to note is that, given n sample database images and m candidate keys, CLUSTER and GREEDY take $O(n^2)$ and $O(mn^2)$ time respectively. Thus, for very large databases, one might consider sampling the database. We used sampling in GREEDY and not in CLUSTER, yet GREEDY was essentially as good as CLUSTER in some cases and better than CLUSTER in the rest. It is promising that relatively simple algorithms were able to increase performance to the extent shown in this paper.

9 The Triangle Trie

Although much faster than direct comparisons, the basic triangle-inequality algorithm described above has a running time of $O(nk)$, where n is the number of images and k is the number of keys. Running time may become unacceptable for very large databases with a large number of keys. Therefore, we take advantage of a data structure called the *triangle trie*[4] to reduce the number of operations.

A *triangle trie*, also called a *Really Fixed Query Tree*[2], is a data structure developed for approximate-match searching both by Baeza-Yates *et al.* and by Berman, in independent research efforts. A single triangle trie is associated with a distance measure, a set of key images, and a set of database elements. It is a form of *trie*, which is a tree in which the edges leading from the root to a leaf "spell out" the index of the leaf. The leaves of the tree contain the database elements. Each internal edge in the tree is associated with a non-negative number. Each level of the tree is associated with a single key. The path from the root of the tree to a database element in a leaf represents the distances from that database element to each of the keys.

Figure 15 illustrates a triangle trie with four elements (W, X, Y, Z), and two keys (J, K). The distance from W to J is 3 and the distance from W to K is 1. This is expressed in the trie by the path from the root to the leaf containing W.

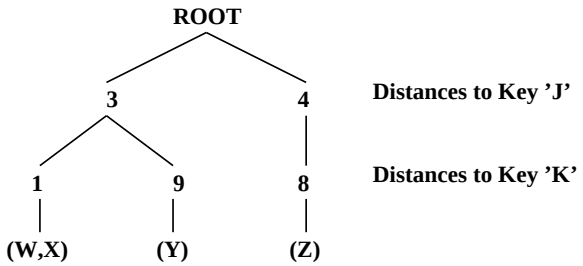


Figure 15: Triangle Trie with two levels.

Construction of the trie is straightforward. Compute the distances from the keys to the database elements. Starting with an empty trie, insert the database elements one at a time using the vector of its key distances. Create nodes as necessary until every element is in the trie. Formally, let $S = (x_1, ...x_n)$ be our set of objects in the database. Let $key_1, ...key_j$ be another set of objects, known as "key objects". For each x_i in S compute the vector $v_i = (d(x_i, key_1), d(x_i, key_2), \ldots, d(x_i, key_j))$. Then combine the vectors $v_1, \ldots, v_n$ into a *trie*, with x_i being placed on the leaf reached by following the path represented by v_i.

9.1 Pruning the Search with a Triangle Trie

Suppose we are given query q and threshold integer t and wish to find all objects in our database with a distance from q of not more than t. Now, consider a node p at level l with a value of C. Every object at leaves descendant from p has a distance of C from the key object key_l. Thus, if $|C - d(q, key_l)|$ is greater than t, then we know from the triangle inequality that $d(q, s')$ is greater than t for all object s' which are descendants of p. Thus, we can safely prune the search at node p.

The algorithm for searching the database using the triangle trie is straightforward. Compute the distances from q to each key: $d(q, key_1), \ldots, d(q, key_j)$. Perform a depth-first search of the trie. If there is a node p at level l with value C such that $|c - d(q, key_l)| > t$, then prune the search at node p. When a leaf is reached, measure the distance from q to every object in the leaf and return those objects i for which $d(q, i)$ is less than or equal to t.

Searching the Trie: An Example

We continue with the example. Suppose we wish to search our database for a close match to object V where our maximum allowed distance to V is 1. We compute v_V by calculating $d(V, key_1)$ and $d(V, key_2)$. We discover that $v_V = (3, 8)$. We then perform our depth-first search. At the top level, we only search nodes with a value within 3 ± 1, so both nodes at this level are searched. At the second level, we only search children of those nodes with a value within 8 ± 1. At this level Y and Z are returned as potential matches,

while X and W are eliminated. The final step is to compute $d(V, Y)$ and $d(V, Z)$. The algorithm does not need to compute $d(V, X)$ and $d(V, W)$.

There may be cases where the distance measure is not integer valued, or where the distance measure has such a wide variance that any resultant triangle trie would have a very quick fan-out. In these cases, it may be necessary to map the calculated distances to a smaller set of values. We call this process *binning* the distances. We have conducted experiments with triangle tries of various bin sizes and depths to better understand the effect of binning on performance.

9.2 Two-Stage Pruning with a Triangle Trie

The breadth of a triangle trie expands with its depth, up to a maximum breadth equal to the number of database elements. The value of a pruning step is directly related to the number of leaves of the pruned subtree. Thus, as the breadth increases, the performance of the tree-pruning algorithm decreases, until it is unfavorable when compared to directly calculating lower bounds for each database object. On the other hand, the efficiency of the triangle inequality algorithm increases with the number of keys used. Our work suggests that by using a relatively short trie, and by storing additional key distances in the leaves, we can obtain the best of both worlds. Our two-stage algorithm works as follows: Given database images $I_1, \ldots, I_n$, keys $K_1, \ldots, K_m$, and distance measure d, we create a triangle trie T of depth T_{depth} where $T_{depth} < m$. For each stored image I_i, we reference I_i in the trie along with $d(I_i, K_j)$ for all $K_1, \ldots, K_m$. Given query Q, we perform our search of the trie as described above. Once completed, we calculate lower bounds on the returned images using all the keys. This further reduces the size of the returned set.

9.3 Using Triangle Tries with Composite Measures

A triangle trie is designed to enable thresholded database searches for a single distance measure. However, it is possible to use multiple triangle tries to enable thresholded database searches over a composite measure. The intuition is to figure out for each distance measure, what threshold should be used for the corresponding triangle trie. Searches are done on the individual tries and the returned images are either merged or intersected, depending on the particular operation. This operation substitutes for the first stage of the two-stage pruning algorithm. The second stage proceeds as before.

To see how this works, consider the following example: Define $R(T, Q, t)$ as the set of images returned from a search on trie T with threshold t. Now consider the composite distance measure $d(I, Q) = Min(d_1(I, Q), d_2(I, Q))$. Assume the threshold used is t. Let T_1 and T_2 represent the tries associated with d_1 and d_2 respectively. Since $d(I, Q) \leq t$ whenever $d_1(I, Q) \leq t$ or $d_2(I, Q) \leq t$, we must find all images

where $d_1(I,Q) \leq t$ or $d_2(I,Q) \leq t$. Thus, we calculate $R(T_1,Q,t)$ and $R(T_2,Q,t)$ and merge the results. Call this resultant set S'. This set consists of all images that have a possibility of being within distance t to Q by distance measure d. We then prune S' with the triangle inequality on the composite function d.

The objective in using the triangle trie is to reduce the number of images for which we have to compute the triangle inequality with the full set of keys. Therefore, when using multiple triangle tries, our objective should be to return as small as possible set of images that need to be further pruned. We have developed algorithms for each of the operations– Min, Max, Sum, and Weight– that reduce the size of the returned set. We describe the various algorithms for binary operations and then show how to combine them for a more complex composite distance function.

9.3.1 The Max function

Given distance functions d_1 and d_2, associated triangle tries T_1 and T_2, query Q, and threshold t, suppose we wish to find all images I such that $d(I,Q) \leq t$ where $d(I,Q) = Max(d_1(I,Q), d_2(I,Q))$. For $d(I,Q) \leq t$ to be true, both $d_1(I,Q)$ and $d_2(I,Q)$ must also be true. Thus, the algorithm for the Max function is to calculate $R(T_1,Q,t) \cap R(T_2,Q,t)$ by searching on T_1 and T_2 and taking the intersection of the results. Second stage pruning is then applied to the intersection set.

Note that we could perform second stage pruning on $R(T_1,Q,t)$ and $R(T_2,Q,t)$ separately and take the intersection of the results. Assume the sizes of the two sets are n and m with k keys and an intersection of $w \leq Min(n,m)$. Intersecting before key comparisons results in an intersection step time of $O(n+m)$, followed by a key comparison step of time $O(kw)$. Performing key comparison first results in a key comparison time of $O(k(n+m))$ followed by an intersection step of unknown time (all elements could have been eliminated). Which method is faster in any particular case would probably rely heavily on details of the system.

9.3.2 The Min function

Suppose $d = Min(d_1, d_2)$. If image I has the property that $d(I,Q) \leq t$, either $d_1(I,Q) \leq t$ or $d_2(I,Q) \leq t$ must be true. Thus, I must be in $R(T_1,Q,t) \cup R(T_2,Q,t)$. To find potential approximate matches to Q in this case, we compute the union of the two R functions.

9.3.3 The Addition function

Suppose $d = d_1 + d_2$. Suppose image I has the property that $d(I,Q) \leq t$. Certainly $d(I,Q) \leq t \Rightarrow d_1(I,Q) \leq t, d_2(I,Q) \leq t$. Thus, if image I has the property that $d(I,Q) \leq t$, then as in the Min function above, I must be in $R(T_1,Q,t) \cap R(T_2,Q,t)$. This leads to an algorithm where we calculate $R(T_1,Q,t)$ and $R(T_2,Q,t)$ and take their intersection.

There are other algorithmic possibilities. For example, $s \in R(T_1, Q, t) \cap R(T_2, Q, t) \Rightarrow s \in R(T_1, Q, t)$ implies that we could just calculate $R(T_1, Q, t)$ and not bother to calculate $R(T_2, Q, t)$. This may be faster in some cases. Similarly we could just compute $R(T_2, Q, t)$. We could compute both and return the smaller set. We also can reduce the thresholds for the individual sets, as follows: Suppose that $d_1(I, Q) > v$ for a given image I and some arbitrary value v. Then $d(I, Q) \leq t$ implies that $d_2(I, Q) \leq t - v$. Thus, $d(I, Q) \leq t \Rightarrow d_1(I, Q) \leq v, d_2(I, Q) \leq t - v$, for any v. We therefore can claim that I must be in $R(T_1, Q, v) \cup R(T_2, Q, t - v)$ for any legitimate $0 \leq v \leq t$. Thus, to find potential approximate matches in this case, we can pick some value for v and compute the union of the two R functions with the modified thresholds.

It is an open question as to how to efficiently decide the best value for v. Choosing $v = 0$ or $v = t$ has the advantage of eliminating the search of one trie entirely as well as the consequent merging of results. Yet there is evidence that halving a threshold more than halves the returned results. In our system we simply choose $v = t/2$.

9.3.4 The Weight function

Suppose $d = Cd_1$ for some positive constant C. Then $d(I, Q) \leq t$ implies $d_1(I, Q) \leq t/C$. In this case we find candidates for approximate matches to Q by calculating $R(T_1, Q, t/C)$.

9.4 Experiments on Triangle Tries

We tested the performance of the triangle trie in image searches. We used the basic distance measures described earlier and gridded versions of the measures. Currently, our database contains $37,748$ images. We used only $20,000$ of these images in our triangle-trie experiments to reduce disk accesses, which might otherwise have disrupted the time measurements.

Trie searches are performed given both a query and a threshold value. No images that are provably more distant to the query than the threshold value are returned from a trie search. The efficiency of the trie search is dependent on the chosen threshold value. Thus, we had to determine appropriate threshold values for each distance measure. We examined the distances between pairs of closely matching images in our database to determine appropriate threshold values. These values are listed in Table 12. The threshold values for the gridded distance measures were chosen to be the same as those for the non-gridded distance measures. This consistency afforded an opportunity to see if there were marked differences in performance between the gridded measures and regular measures.

We created a set of triangle tries for each distance measure, varying the tries by depth and bin size. The depths ranged from 1 to 11, and the bin size ranged from 10 to 130 stepping by 10. All the distance

Table 12: Threshold values chosen for the triangle-trie experiments.

Distance Measure	Threshold Value
Color (4x4x4 RGB)	200
LBP	90
Wavelet	15
Sobel	10
Flesh	10

measures except the Haar Wavelets had a range of 0 to 1000. The Haar Wavelets had a theoretical range greater than 1000, but no distances greater than 1000 were calculated during the experiment. Experiments were not conducted for a given depth and bin size if the created trie would contain more than $20,000$ nodes. This limit was chosen due to memory constraints.

There are 35 key images in our database. Any subset of them can be used as the keys for the tries. We created five random orderings of the keys. For each triple of 10 distance measures, 11 depths, and 13 bin sizes, we created five tries using the appropriate five prefixes of the random key orderings. For each such trie, we tested five images, resulting in a total of twenty-five tests for each triple of distance measure, depth, and bin size. This resulted in a total of $33,295$ experiments, not including experiments that were abandoned due to too large a trie being created. Each experiment was repeated 250 times to improve the accuracy of the measured time. We recorded the size of the created tries, the number of matches returned for a search, the number of internal trie nodes accessed, the number of leaf trie nodes accessed, and the time it took to walk the trie and return the matches.

9.5 Triangle-Trie Test Results with Single Distance Measures

In section 6, we estimated a time of approximately 4 milliseconds to calculate the lower bounds on 1000 images using the bare-bones triangle inequality algorithm, not including key calculations and feature extraction. We used this value to calculate the speed-improvement factor gained by using the two-stage pruning algorithm instead of the bare-bones algorithm.

Table 13 shows the results of the tries that had the highest speed improvement factors. The gridded version of the local-binary-partition texture measure had the minimum improvement, being about twice as fast with a trie as without one. At the other end of the scale, the Sobel-edge-histogram texture measure was almost fifty times as fast with a trie as without one. This represents a potential processing rate of almost twelve million images per second.

Note that there are many factors which affect trie performance. Even within a given trie depth, bin size, and distance measure, the speed of a trie search varied greatly with the keys chosen and the query image. Trie search performance varies with the chosen threshold value–as the algorithm uses the threshold value to prune subtrees, reducing the threshold results in less of the tree being walked, yielding a time savings. In particular, the algorithm walks only one path with a threshold value of zero, while it walks the full trie with a threshold value of infinity.

9.6 Triange-Trie Results with Composite Distance Measures

The *two-stage pruning* algorithm, detailed in Section 5.2, is a method for using triangle-tries for composites of basic distance measures. Given a distance measure $d' = g(d_1, \ldots, d_k)$ where the system has triangle tries for the distance measures $d_1, \ldots, d_k$, the system computes appropriate threshold values for the basic distance measures and composites the returned potential matches dependent on the character of $g()$. The bare-bones triangle-inequality algorithm is then run on the returned potential matches.

The two major methods for compositing returned potential matches are intersection and union. As expected, intersection reduces the number of potential matches, while merging increases them. Suppose the returns from several triangle tries are merged, and the resultant set of images is the same size as the full database. In that case, no benefit is accrued from the use of the triangle trie. On the other hand, suppose the results from several tries are to be intersected, and a search of one of the tries returned only a small handful of potential matches. In that case, since the intersection of X and Y is contained within X, it may be worth directly proceeding to lower-bound calculation with that small returned handful and avoiding the cost of processing the remaining tries or of intersecting the results. Tables 16 and 17 show the average results of merging and intersecting various returned matches from the triangle tries for the basic distance measures.

9.7 Discussion of Results for Triangle Trie-Experiments

The Triangle Trie offered improved performance on all of the tested distance measures. Some of the improvements were quite significant and might enable otherwise difficult tasks such as nearest neighbor searches for large sets of images. The performance of triangle-inequality-based searches is dependent on a number of factors, including the nature of the image sets, the distance measures, the number and nature of the chosen keys, the chosen threshold, and the properties of the query images. Thus, it is impossible to categorically state that the performance boosts demonstrated here will translate to somebody else's image database. We have demonstrated, however, that the triangle trie is worth considering in the case that a performance boost is needed.

Table 13: Characteristics of the best tries found for each distance measure, rated by speed factor. Time is in milliseconds for a search of 20,000 images and is just for the trie search itself. Speed Factor is the ratio of time taken for a search of 20,000 images using the triangle inequality algorithm without the triangle trie to a search using the triangle trie as a precursor to the triangle inequality algorithm.

Measure	Speed Factor	Bin Size	Trie Depth	Time	Returned Matches
Sobel	47.8	10	6	0.4	318.6
Gridded Sobel	38.6	30	15	0.7	343.0
Gridded Wavelets	22.3	20	8	1.2	598.6
Gridded Color	15.01	130	11	1.8	882.4
Color	8.8	80	8	4.0	1273.0
Wavelets	3.8	40	4	11.8	2343.6
Gridded Flesh	3.2	30	4	7.0	4588.2
LBPHist	2.8	30	7	20.1	5740.7
Flesh	2.6	80	11	7.4	5980.2
Gridded LBPHist	2.4	80	8	12.1	5465.0

Table 14: Characteristics of tries found for each distance measure with the median speed factor. Time is in milliseconds for a search of 20,000 images and is just for the Trie Search itself. Speed Factor is the ratio of time taken for a search of 20,000 images using the triangle inequality algorithm without the triangle trie to a search using the triangle trie as a precursor to the triangle inequality algorithm.

Measure	Speed Factor	Bin Size	Trie Depth	Time	Returned Matches
Sobel	14.6	90	8	1.2	1014.0
Gridded Sobel	14.6	40	6	1.3	992.2
Gridded Color	6.8	90	4	3.1	1962.0
Gridded Wavelets	5.2	20	3	4.2	2564.4
Color	5.0	20	4	6.8	1826.5
Gridded Flesh	2.2	60	6	10.5	5703.2
Flesh	1.8	130	8	10.3	7668.6
Gridded LBPHist	1.8	100	8	13.3	6250.8
Wavelets	1.6	130	5	18.7	6542.4
LBPHist	1.4	130	9	16.5	8466.4

10 Summary

The overall goal of our research has been to create technology useful in a generalized system for content-based image retrieval. We believe that there are three main requirements for a useful content-based retrieval system. The system must search and retrieve images quickly. The system should allow as wide a selection of queries as possible. Finally, the system should be easy to use.

To enable fast searches, we have developed algorithms based on the triangle inequality and triangle tries. The experiments in this paper provide evidence that these algorithms can be used for fast image retrieval in large databases. Our current system searches a database of $37,000$ images in an average time of less than a

Table 15: Averages of Trie statistics (each column individually averaged). The average is over all the tries in the experiments with depths ranging from 1 to 13, bin sizes ranging from 10 to 140, and no tries with more than 20,000 nodes included.

Measure	Speed Factor	Time	Returned Matches	Internal Nodes	Leaf Nodes
Sobel	15.4	20.8	1679.5	17.3	4.9
Gridded Sobel	15.0	22.8	1514.8	28.1	6.4
Gridded Color	6.6	61.6	3024.0	123.7	103.0
Gridded Wavelets	6.5	63.4	3997.5	47.7	14.6
Color	4.8	77.3	3167.2	152.2	132.3
Gridded Flesh	2.2	98.2	5954.8	9.4	1.8
Wavelets	1.7	231.9	5439.1	35.0	15.2
Flesh	1.7	115.8	7616.3	11.9	2.4
LBPHist	1.4	167.4	8060.9	350.7	257.2
Gridded LBPHist	1.6	163.8	7050.9	408.5	231.4

Table 16: Average size of image sets after merging or intersecting image sequences returned from triangle-trie searches, using one or two distance measures. The rows labeled with single distance measures show the average sizes of the original sets.

Distance Measures	Merged Set Average Size	Intersected Set Average Size
Color	3568	
LBP	6102	
Wavelet	3023	
Sobel	427	
Flesh	8090	
Color Flesh	10001	1656
Color LBP	8342	1327
Color Sobel	3889	106
Color Wavelet	5570	1021
LBP Flesh	11727	2464
LBP Sobel	6355	173
LBP Wavelet	7949	1177
Wavelet Sobel	3382	68
Wavelet Flesh	9771	1342
Sobel Flesh	8366	151

second, depending on the query. We have provided a set of operations for combining distance measures in ways we believe to be flexible enough to allow users to fashion useful queries. Importantly, we have demonstrated that the triangle-inequality algorithms can be used with these combinations of distance measures.

Table 17: Average size of image sets after merging or intersecting image sequences returned from triangle-trie searches, using three or four distance measures.

Distance Measures	Merged Set Average Size	Intersected Set Average Size
Color LBP Wavelet	9606	438
Color LBP Sobel	8537	46
Color LBP Flesh	12964	653
Color Wavelet Sobel	5854	31
Color Wavelet Flesh	11199	538
LBP Wavelet Sobel	8169	35
LBP Wavelet Flesh	12812	581
LBP Sobel Flesh	11894	64
Wavelet Sobel Flesh	10010	31
Color Sobel Flesh	10224	52
Color LBP Wavelet Sobel	9781	16
Color LBP Wavelet Flesh	13753	251
Color LBP Sobel Flesh	13102	22
Color Wavelet Sobel Flesh	11398	18
LBP Wavelet Sobel Flesh	12960	17

Thus the flexibility of our system was not obtained by sacrificing performance.

There are a number of open problems in the various data structures and algorithms we described. We have already mentioned some of them, like key selection, number of keys, trie depth and bin size. More generally, the statistical behavior of distance measures over different sets of images influences the behavior of all the algorithms and thus needs to be explored.

Facilitating user understanding of a system such as ours is an open problem. We believe that user understanding will be an issue for any system that offers a reasonable degree of flexibility. Fundamentally, images are hard to describe. This is the flip-side of the phrase, "an image is worth a thousand words." And to whatever extent images are hard to describe, the relationship between two images can be even harder to describe. Yet it is precisely this relationship that the user must define when querying an image database based on content. There is a lot of room for fruitful research in the area of interfaces between users and content-based retrieval systems.

References

[1] J.R. Bach, S. Paul, and R. Jain. A visual information management system for the interactive retrieval of faces. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):619–628, 1993.

[2] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In *Combinatorial Pattern Matching*, pages 198–212. Springer-Verlag, June 1994.

[3] J. Barros, J. French, W. Martin, P. Kelley, and M. Cannon. Using the triangle inequality to reduce the number of comparisons required for similarity-based retrieval. In *IS&T/SPIE - Storage and Retrieval for Still Image and Video Databases*, volume IV, Jan 1996.

[4] A. Berman. A new data structure for fast approximate matching. Technical Report 1994-03-02, Dept. of Computer Science, University of Washington, 1994.

[5] A. Berman and L. G. Shapiro. A flexible image database system for content-based retrieval. In *17th International Conference on Pattern Recognition*, 1998.

[6] A. P. Berman and L. G. Shapiro. Efficient image retrieval with multiple distance measures. In *Proceedings of the SPIE Conference on Storage and Retrieval for Image and Video Databases*, February 1997.

[7] A. Del Bimbo, M. Campanai, and P. Nesi. 3d visual query language for image databases. *Journal of Visual Languages and Computing*, 3, 1992.

[8] A. Del Bimbo, P. Pala, and S. Santini. Visual image retrieval by elastic deformation of object sketches. In *IEEE Symposium on Visual Languages*, pages 216–223, 1994.

[9] A. Del Bimbo, E. Vicario, and D. Zingoni. Sequence retrieval by contents through spatio temporal indexing. In *IEEE Symposium on Visual Languges*, pages 88–92, 1993.

[10] W. A. Burkhard and R. M. Keller. Some approaches to best-match file searching. *Communications of the ACM*, 16(4):230–236, Apr 1973.

[11] A. Califano and R. Mohan. Multidimensional indexing for recognizing visual shapes. *IEEE Trans. on Pattern. Analysis. Machine. Intell.*, 4:373–392, 1994.

[12] C. Carson, S. Belongie, H. Greenspan, and J. Malik. Region-based image querying. In *IEEE Workshop on Content-Based Access of Image and Video Libraries*, pages 42–51, 1997.

[13] S.K. Chang, J. Reuss, and B.H. McCormick. Design considerations of a pictorial database system. *International Journal on Policy Analysis and Information Systems*, 1(2):49–70, 1978.

[14] S.K. Chang, Q.Y. Shi, and C.W. Yan. Iconic indexing by 2d strings. *IEEE Trans. on Pattern. Analysis. Machine. Intell.*, 9:413–427, 1987.

[15] R. Fagin. Fuzzy queries in multimedia database systems. In *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 1–10, June 1998.

[16] M. M. Fleck, D. A. Forsyth, and C. Pregler. Finding naked people. In *Proceedings of the European Conference on Computer Vision*, pages 593–602. Springer-Verlag, 1996.

[17] M. Flickner, H. Sawhnew, W. Niblack, J. Ashley, Qian-Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: the qbic system. *Computer*, 28(9):23–32, Sep 1995.

[18] D. A. Forsyth, J. Malik, M. M. Fleck, H. Greenspan, T. Leung, S. Belongie, C. Carson, and C. Bregler. Finding pictures of objects in large collections of images. In *Proceedings of the 2nd International Workshop on Object Representation in Computer Vision*, April 1996.

[19] W.I. Grosky and R. Mehrotra. Index-based object recognition in pictorial data management. *Comp. Vision Graphics and Image Processing.*, 52:416–436, 1990.

[20] A. Gupta. Visual information retrieval: A virage perspective. Technical report, http://www.virage.com/literature/wpaper.html, 1995.

[21] A. Gupta. Visual information retrieval: A virage perspective. Technical report, Virage, Inc. http://www.virage.com/wpaper, 1995-1997.

[22] Niels Haering, Zrina Myles, and Niels de Vitoria Lobo. Locating deciduous trees. In *Content-Based Access of Image and Video Libraries*, June 1997.

[23] K. Hirata and T. Kato. Query by visual example. *Advances in Database Technology*, pages 56–71, 1992.

[24] C.E. Jacobs, A. Finkelstein, and D.H. Salesin. Fast multiresolution image querying. In *Computer Graphics Proceedings, Annual Conference Series*, 1995.

[25] A. K. Jain and A. Vailaya. Image retrieval using color and shape. *Pattern Recognition*, 29:1233–1244, 1996.

[26] R. Jain, S.N.J. Murthy, and P.L.J. Chen. Similarity measures for image databases. In *Proceedings of SPIE Storage and Retrieval for Image Databases III*, pages 58–65, 1995.

[27] T. Kato, T. Kurita, N. Otsu, and K. Hirata. A sketch retrieval method for full color image database. In *11th International Conference on Pattern Recognition*, pages 530–533, 1992.

[28] P.M. Kelly and T. M. Cannon. Candid: Comparison algorithm for navigating digital image databases. In *Proceedings of the Seventh International Working Conference on Scientific and Statistical Database Management*, pages 252–258, 1994.

[29] P.M. Kelly and T.M. Cannon. Query by image example: the candid approach. In *SPIE Storage and Retrieval for Image and Video Databases III*, volume 2420, pages 238–248, 1995.

[30] W. Y. Ma and B. S. Manjunath. Netra: A toolbox for navigating large image databases. In *IEEE International Conference on Image Processing*, 1997.

[31] R. Mehrotra, F.K. Kung, and W. Grosky. Industrial part recognition using a component-index. *Image and Vision Computing*, 3:225–231, 1990.

[32] T. P. Minka and R. W. Picard. Interactive learning using a "society of models". Technical Report 349, M.I.T. Media Laboratory Perceptual Computing Section, 1995.

[33] W. Niblack. The qbic project: querying images by content using color, texture, and shape. Technical Report RJ 9203, IBM Research, 1993.

[34] A. Pentland, R. W. Picard, and S. Sclaroff. Photobook: Tools for content-based manipulation of image databases. Technical report, MIT Media Lab, 1993.

[35] R. W. Picard and T. P. Minka. Vision texture for annotation. *Journal of Multimedia Systems*, 3:3–14, 1995.

[36] R.W. Picard and F. Liu. A new wold ordering for image similarity. In *Proc. Conf. on Acoust., Speech and Sig. Proc.*, 1994.

[37] S. Sclaroff and A. Pentland. Object recognition and categorization using modal matching. In *Proc. of 2nd CAD-Based Vision Workshop*, pages 258–265, 1994.

[38] S. Sclaroff, L. Taycher, and M. La Cascia. Imagerover: A content-based image browser for the world wide web. In *Proc. IEEE Workshop on Content-based Access of Image and Video Libr aries*, June 1997.

[39] L. G. Shapiro and M. S. Costa. Scene analysis using appearance-based models and relational indexing. In *IEEE Symposium on Computer Vision*, pages 103–108, 1995.

[40] L. G. Shapiro and R. Haralick. Structural descriptions and inexact matching. *IEEE Trans. on Pattern. Analysis. Machine. Intell.*, 3:504–519, 1981.

[41] M. Swain and D. Ballard. Color indexing. *International Journal of Computer Vision*, 7:11–32, 1991.

[42] M. Tuceryan. and A.K. Jain. Texture analysis. *Handbook of Pattern Recognition and Computer Vision*, pages 235–276, 1994.

[43] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 1991.

[44] J. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40:175–179, 1991.

[45] L. Wang and D. C. He. Texture classification using texture spectrum. *Pattern Recognition Letters*, 13:905–910, 1990.