Reducing Noise Interference in Image Query Processing

Khanh Vu *Kien A. Hua*

School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL 32816-2362
U. S. A.

E-mail: {khanh,kienhua}@cs.ucf.edu

Abstract

Query-by-example is the most popular query model for today's image retrieval systems. A typical query image contains not only relevant objects (e.g., Eiffel Tower), but also irrelevant image areas (e.g., the background). The latter, referred to as noise in this paper, has limited the effectiveness of existing image retrieval systems. To address this drawback, the user must be allowed to draw contours to precisely outline the relevant image areas as the query. We refer this class of queries as *noise-free queries* (NFQs), and propose a technique for processing them in a sampling-based matching framework. We introduce a new similarity model, and investigate indexing techniques for this new environment. We also implement a prototype to demonstrate this approach. Our query model is more expressive than the standard query-by-example. The user can draw a contour around a number of objects to specify spatial (relative distance) and scaling (relative size) constraints among them, or use separate contours to dis-associate these objects. Our experimental results confirm that traditional approaches, such as Local Color Histogram and Correlogram, suffer from noisy queries. In contrast, our method can leverage NFQs to offer significantly better performance. This is achieved using only 1/16 of the storage overhead required by the other two techniques.

Keywords: Image processing and retrieval, noise reduction, arbitrary-shaped queries, image indexing, clustering, core-area detection.

1 Introduction

The popularity of digital images have spurred a demand for robust content-based image retrieval systems. These systems are needed in various application domains including medical imaging, digital libraries, geographic mapping, and electronic commerce, to name but a few. The growing demand for this technology has led to a significant body of recent research addressing image retrieval problems ([1],[2],[3],[4],[5],[6],[7],[8], etc.).

Image databases can be queried in several ways. Query-By-Example (QBE), however, is by far the most widely supported model in research prototypes and commercial products. In this environment, a user formulates a query by means of giving an example image selected from a pool of general image categories. Since this sample set is generally small, the expectation of finding a perfect example (i.e. the entire content is relevant) is low. As a result, a query image typically contains not only the objects of the user's interest, but also irrelevant image areas. We refer to the latter as *noise* in this paper. Many content-based image retrieval techniques have been developed in recent years to enhance robustness with respect to translation, scaling, and changes in the viewing angle. However, problems with noise have been largely neglected. We address this outstanding issue in this paper by proposing a similarity model for noise-free queries. This new model enables the user to precisely include only relevant objects in formulating a query. Queries so defined are called *noise-free queries* (NFQs). In the following, we review some of the existing image retrieval techniques and examine their ability to support NFQs.

In a typical content-based image retrieval system, essential properties of the database images are extracted and stored as *feature vectors*. During a retrieval process, the feature vector of the query image is computed and compared against those in the database. Typically, these feature vectors capture the color distribution (e.g., color histograms). Recent techniques also include other features such as texture [9], structure [8], etc. to increase the overall effectiveness. A scheme, based on local matching, was proposed in [2]. This method utilizes a large number of overlapping square subimages of each image. A signature is computed for each such subimage. The signatures

of each image are mapped into points in the feature space, and clustered into *subimage groups*. For each subimage group, the centroid of the signature points is used as the *group signature*. Matching between two images is done by pair-wise comparison of their matching groups (i.e., having similar group signatures). This scheme minimizes the effect of global features on local matches. However, it cannot tell the difference between several scattered red balloons and a red car since the balloons and the car would have similar group signatures. To address this problem, a scheme, called *Correlogram* [7], can be used to capture finer details of the color distribution by incorporating both color and its spatial layout, so called *spatial color indexing*. Their experimental results [7] indicate that Correlogram offers excellent performance for a wide range of images. It can handle both translation and scaling in the matching objects. It can also tolerate differences in the background areas, or in the object appearance due to small changes in the viewing angle. In this paper, we refer to all the aforementioned techniques as the *whole-matching* approach since they extract features based on the entire image area, and tightly integrate them into feature vectors. Any matching must be done on the entire image area; and noise exclusion obviously is not possible.

Image comparison can also be done by matching homogeneous image regions. This is referred to as the *region-based* approach (e.g., QBIC [1], VisualSeek [10], Netra [11], Blobworld [12].) This scheme segments each image into several regions; and image matching is done by comparing the visual features, such as average color, dominant color, texture, shape, size, etc., of these regions. While this approach has many advantages, its effectiveness relies on the accuracy of image segmentation techniques which are far from reliable today. This is due to the fact that image segmentation and content recognition are a pair of ”chicken and egg” problem [13]. While reliable image segmentation cannot be achieved without recognition of the image content, image recognition relies on good image segmentation. When image regions are incorrectly detected, their visual features are wrong and would affect the precision of the image retrieval result. Today’s region-based systems also do not support NFQs. The user still submits the entire image area as the query. We note that matching arbitrary-shaped regions is not the same as searching for images similar to an arbitrary-shaped query. The former may contain noise.

From the above discussion, we can make the following observation. To support noise-free queries, we must get outside the conventional whole-matching paradigm. The user must be allowed to use an irregular-shape image as a query. That is, one must be able to draw contours around the relevant objects in order to specify the query precisely. We can support such NFQs in a region-based framework by matching only image regions falling within the query contours. While this solution is straightforward, its performance is limited by the reliability of the image segmentation techniques as we have discussed. In this paper, we address this class of queries using a sampling-based approach called *SamMatch* [14]. The contribution of this paper is the design of a new similarity model and its implementation techniques including indexing methods and a query model to support NFQs. By insulating the similarity computation from the interference of noise, our experimental results indicate that retrieval effectiveness can be improved significantly for a wide range of queries.

The remainder of this paper is organized as follows. We review our SamMatch environment, and present a similarity model for NFQ's based on this framework in Section 2. We describe our indexing method in Section 3. Our query processing technique is introduced in Section 4. In Section 5, we discuss an efficient algorithm for determining the core area of a query. Our experimental study is discussed in Section 6. Finally, we present our concluding remarks in Section 7.

2 A Similarity Model for NFQs

In this section, we first review the SamMatch environment. We then present the similarity measure for processing NFQs in this framework, and discuss the technique for handling scaling in the matching objects.

2.1 SamMatch Environment

In SamMatch, we take samples of 16×16-pixel blocks at various locations of each image. The rationale for this block size is that the correlation between pixels tends to decrease after 15 to 20 pixels [15]. In other words, a run of similar pixels usually ends after 15 to 20 pixels. This characteristic allows us to significantly reduce storage overhead by representing each sampled block using its average color. To support general applications, we collect these blocks at uniform locations throughout the image (Figure 1). This strategy is reminiscent of the audio digitizing process, in which the magnitudes of a sound wave are sampled at some fixed time intervals and stored as numbers. This collection of numbers captures the characteristics of the sound wave to the details allowed by the sampling rate. Similarly, our technique samples an image with respect to space intervals as illustrated in Figure 1. It shows 113 samples evenly spread out in a 256×256 image frame.

Without loss of generality, we assume that the images are stored using the *Munsell* color system[1]. In this uniform color system, the dissimilarity of two colors is simply the distance between them. We quantize the (H,V,C) data into 256 possible values. The most dominant Haar wavelet coefficient of each sampled block is used as its average color [17].

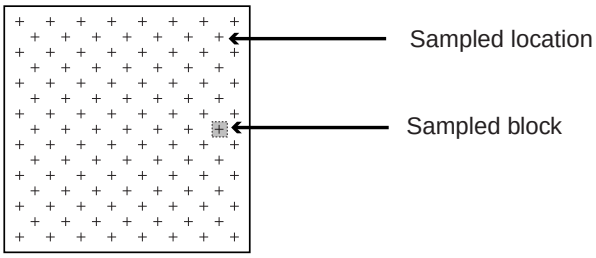


Figure 1: Ckeckerboard-pattern sampling of image blocks

[1] If the images are in RGB, one can convert them into Munsell using the mathematical transformation presented in [16].

2.2 Similarity Measure for Processing NFQ's

Unlike the rectangular shapes of traditional queries, each NFQ is an arbitrary-shaped image. A distinct advantage of the SamMatch environment is that we can compare arbitrary-shaped subimages by comparing only the sampled blocks falling within the subimage area. This feature of SamMatch lends itself naturally to supporting NFQ's. Consider two arbitrary-shape subimages Q and I, each represented by n sampled blocks. Their matching score can be computed as follows:

$$S(Q, I) = \sum_{i=1}^{n} \frac{w_i}{1 + \mathcal{D}(c_i^I, c_i^Q)} \tag{1}$$

where $\mathcal{D}(c_i^I, c_i^Q)$ is the distance between the average color of block i of subimage I, c_i^I, and the average color of block i of subimage Q, c_i^Q. In other words, we compare the corresponding sampled blocks of the two subimages. A '1' is added to the denominator to prevent division by zero. In the numerator, w_i is a weighing factor which can be set to indicate the significance of the match at block i. This parameter can be formulated in different ways to enhance the matching effectiveness.

Since SamMatch compares the corresponding sampled blocks of two images, this strategy can be viewed as a structured-based matching technique. We can also call it an object-based matching method because the comparisons are based on matching the sampled-block structure (or skeleton) of the objects within the query areas. This matching scheme implicitly takes into account the shape, size, and texture features, etc. of the image objects. SamMatch, therefore, has the benefits of the region-based techniques without having to rely on the unpredictable accuracy of segmentation methods. This is achieved by letting the user point out the object areas at query time.

We consider the following two factors in our system:

- In general, not all colors are equally likely in real-world images. As a result, matches on rare colors are more discriminating. w_i can be designed to exploit this fact by taking into account the frequency of the matching color at block i. That is, w_i is proportional to the inverse frequency of the matching color c_i^Q. To facilitate this enhancement, we studied 16,000 images to determine the frequencies of the 256 possible average colors.

- We consider a cluster of well-matched blocks as more significant than a collection of scattered matched blocks. The rationale is that a cluster of adjacent blocks often identifies an object in the image whereas scattered blocks are less meaningful. Thus, w_i should be raised or lowered proportional to the matching scores of the neighboring blocks within a preset vicinity. We consider this spatial correlation in our similarity computation.

2.3 Handling Scaling in the Matching Objects

In SamMatch, the same sampling rate is used for all database images. However, we can apply various sampling rates on the query image to find matches at various scales. This is illustrated in Figure 2. Figure 2(a) shows one database image with its sampled blocks. In this example, the query image is assumed to be smaller in size, and is sampled at three different rates as illustrated in Figures 2(b), 2(c), and 2(d). We note that a lower sampling rate corresponds to a longer distance between the sampled blocks. By comparing the blocks in Figure 2(b) with the corresponding blocks in the database images, we can find images of similar but bigger objects. Similarly, the blocks in Figure 2(c) are sampled at the same rate as that used for the database images to find images of matching objects of similar size. Finally, the sampling rate shown in Figure 2(d) is used to find images of similar, but smaller objects.

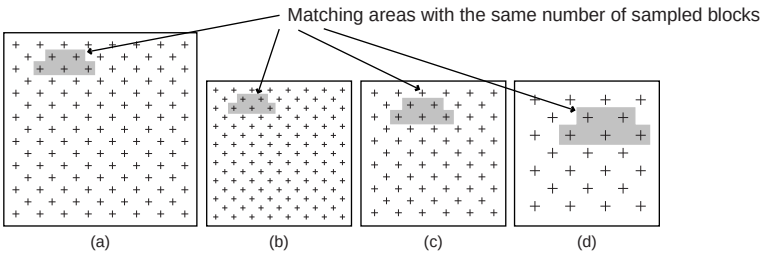


Figure 2: One fixed sampling rate for database images (a) and three sampling rates for the query (b, c, d) in order to match larger objects, same-size objects and smaller objects, respectively

To support matching at various practical scales, our system currently employs five different sampling rates for the query images. This results in five different sampling configurations with 25,

41, 61, 85, and 113 sampled blocks, respectively. Although we can take advantage of additional sampling rates, our experiments indicate that these five are sufficient to achieve excellent performance. We will address the translation in the matching objects in Section 3. We decide not to consider rotation invariance in this paper since this property is not always desirable, e.g., resulting in false retrievals.

3 Indexing

In this section, we present implementation techniques for the similarity model presented in Section 2. To facilitate subimage matching at different scales, we slide five square-shape windows of various sizes over database images, and index them on the subimages captured by these windows at sliding positions. We will refer to these subimages as *indexing subimages*. In the following two subsections, we first describe signatures, and then discuss the index structure in more detail.

3.1 Image Signatures

We use sliding window of sizes 25, 41, 61, 85, and 113 sampled blocks, respectively, as illustrated in Figure 3(a). Thus, the smallest indexing subimage covers only 25 sampled blocks at any sliding position. On the other hand, the largest indexing subimage can contain all 113 sampled blocks of an image. In our design, a *sliding step* is the distance between two nearest sampling blocks in the sliding direction. An example of sliding a small window over five positions is shown in Figure 3. The rationale for our window sizes and the sliding step size is as follows:

1. Subimages containing fewer than 25 sampled blocks are not interesting for most image-retrieval purposes.

2. The contents of heavily overlapping subimages are essentially similar. It is not necessary to consider square-shape subimages less than one sliding step from the indexing subimages.

In other words, the subimages used in our design practically represent all ”possible” subimages of the database image.

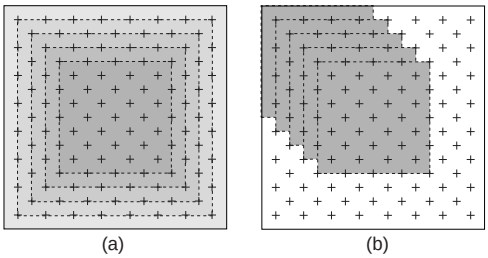


Figure 3: Various subimage sizes (a) and sliding examples (b)

There are 41, 25, 13, 5 and 1 indexing subimages of sizes 25, 41, 61, 85, and 113 sampled blocks, respectively. In total, each database image contains 85 indexing subimages. For each indexing subimage, we compute its *signature* as seven statistical average-variance pairs as follows:

- the first pair is computed from the average colors of all the sampled blocks in the indexing subimage (Figure 4(a));
- each of the next four pairs is computed from the average colors of the sampled blocks in a distinct quadrant of the indexing subimage (Figure 4(b));
- finally, each of the last two pairs is computed from the average colors of the sampled blocks along one of the two diagonals of the indexing subimage (Figure 4(c)).

We use as many as 14 features because R*-tree [18] has been shown to perform well up to 16 dimensions [19]. These average-variance pairs are designed to capture the various local characteristics of the indexing subimage. For each pair, the *average* is the mean of the average colors of the relevant sampled blocks; and the *variance* is the statistical variance of these average colors. Using average-variance pairs has the following advantage. Those computed from heavily overlapping indexing subimages are similar although their Haar wavelet coefficients may be quite different.

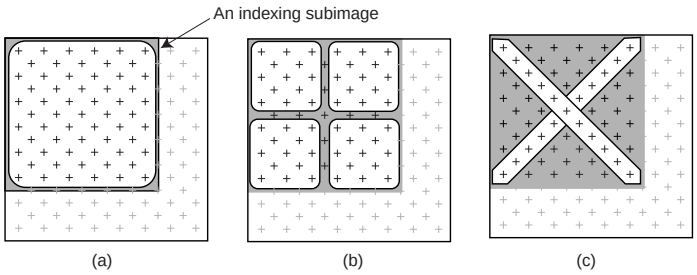


Figure 4: The seven areas used to compute the signature of an indexing subimage.

3.2 Clustering and Indexing

The introduction of the signatures is to support our initial search procedure as follows. Given an arbitrary-shaped image as the query, we can identify its *core area* as the largest square subimage that contains a noise level (i.e., irrelevant area) no greater than a predetermined limit. We will discuss an efficient algorithm for finding this core area in Section 5. Once this core area has been determined, we can compute its signature, and use it to find similar indexing subimages in the database. This initial search step can leverage an index structure such as an R*-tree to quickly filter out those database images that have no chance of matching the query image. Only those database images with at least one matching indexing subimage need to participate in the more thorough test of comparing the corresponding sampled blocks. We will discuss the image retrieval procedure in more detail in the next section.

Although we can index the database images on their 85 signatures, such an access structure would be a monolith. To substantially reduce the size of the index, we map the 85 signatures of each image into *signature points* in the 14-dimensional feature space, and cluster them into m *minimal bounding rectangles* (MBRs). In other words, each database image is represented by its m MBRs; and we index the database on these MBRs. This strategy significantly reduces the height of the R*-tree.

There are many high-performance data clustering algorithms proposed recently such as BIRCH [20], WaveCluster [21]. Without loss of generality, we adapt BIRCH for our system. Since a straightforward application of this algorithm could result in too many MBRs, we modified it as

follows to control the number of generated MBRs:

1. Apply the BIRCH algorithm [20] to obtain clusters using the default values.
2. Split the clusters (also described in [20]) until none contains more than N signature points.
3. Order the clusters by forming a Hamilton chain between the two farthest clusters using a greedy approach, i.e., find the closest cluster and connect it to the chain at each step.
4. Let the first cluster be called *recipient*
5. If there is a cluster following *recipient* then let the *recipient*'s next cluster be called *donor*. Exit otherwise, i.e., there are no more clusters.
6. While the number of signature points in *recipient* is smaller than N, repeat:
   (a) If *donor* is non empty then
      i. Select one signature point from *donor* that is nearest to *recipient*.
      ii. Move it to *recipient*.
      iii. If *donor* is now empty then free *donor* and let the cluster following *donor* be *donor*.
   (b) Exit otherwise, i.e., there are no more signature points.
7. Let *donor* be *recipient* and go to step 5.

The above procedure forces a fixed number of clusters for each image, each (except the last) containing N signature points. Although these clusters might not be as compact as those generated by BIRCH, the significant reduction in the height of the R^*-tree helps reduce the search time. We will examine this issue in more detail when we discuss the experimental results in Section 6. A clustering example is given in Figure 5. The signature points, computed from an actual image, are grouped into five clusters using the above procedure. A straightforward application of the BIRCH algorithm would create as many as 15 clusters.

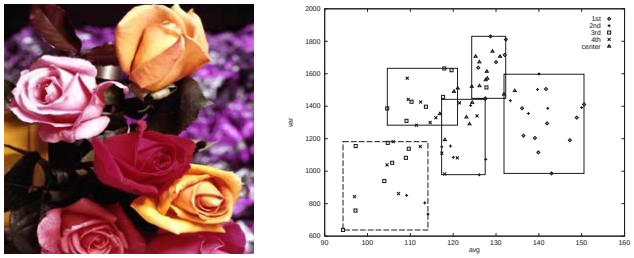


Figure 5: Five clusters of 2-D signature points (right) computed from the image (left).

We note that our sliding windows are not used in the same way as in the Walrus technique [2]. The latter slides a window over the image area to discover regions of similar characteristics, whereas we use sliding windows of different sizes to capture the various possible subimages for the purpose of indexing.

4 Image Retrieval Procedure

In this section, we first present our query model, and then discuss the implementation technique using the index structure presented in the last section.

4.1 Query Formulation

We recall that SamMatch is capable of matching irregular-shaped subimages. This feature enables us to support noise-free queries as follows. To express a query, the user first selects a suitable image from the collection of image categories. A contour is then drawn as precisely as desired to specify the relevant objects in the selected image. An example is illustrated in Figure 7(a). It shows a query for a person about to enter a car. This query is represented internally by the sampled blocks falling within the contour. We observe that this query is essentially noise free since irrelevant parts of the image, such as the trees and clouds, have no effect on the outcome of the retrieval.

Other unique benefits of this query model are as follows. A query contour imposes spatial (relative distances) and scaling (relative sizes) constraints on the enclosed objects. For instance,

the NFQ in Figure 7(a) implies the following semantic constraints:

- Spatial Constraint: The person must be close to the driver's side of the car.
- Scaling Constraint: The car must be about five times the size of the person.

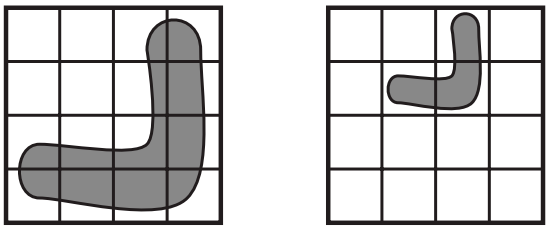
Enforcing scaling constraints is not the same as handling scaling in existing methods. The intent of the query in our example is to find images of a person about to enter a car. Such a query should not match an image having at the bottom a person standing on the street, and a billboard of a car in the upper right corner. Existing retrieval techniques capable of tolerating translation and scaling of objects (e.g., [2],[7]) would incorrectly return this image as a good match. In fact, they would retrieve any images having a person and a car, disregarding the intended constraints on the two objects. If disassociating the query objects is desired, SamMatch allows the user to use a separate contour for each object. Thus, our query model is more expressive than conventional whole-image matching model.

Some degree of flexibility in formulating queries can also be achieved by treating each database image as a collection of non-overlapping subimages, each having its own color histogram [5],[22]. For a given query, any combination of the local histograms can be selected for matching the database images with the query image. We refer to these techniques as the *Local Color Histogram* (LCH) approach. Although they provide a means to exclude some degree of noise, several drawbacks remain:

- The storage overhead is very high compared to SamMatch. For instance, the storage cost for only one local color histogram is about the same as the storage overhead for the SamMatch approach. We will discuss this issue in more detail in Section 6.
- This approach is not noise free since each rectangular subimage is rather large and can contain substantial noise.
- It is not clear how to handle matching at different scales when the objects of interest span

multiple subimages. For instance, the query image in Figure 6(a) will not match the database image in Figure 6(b) no matter which subimages are selected for matching.

We will show experimental results to compare SamMatch and LCH in Section 6.



(a) Query image (b) A database image

Figure 6: LCH does not support matching at different scales.

4.2 Query Processing

As we have discussed previously, the query image is sampled at five different rates to support matching at five different scales. The execution of a query involves the following steps:

- Search Preparation:

  1. We apply five different sampling rates to the query image to create five different versions of the NFQ, each designed to compare against subimages of a predetermined size.

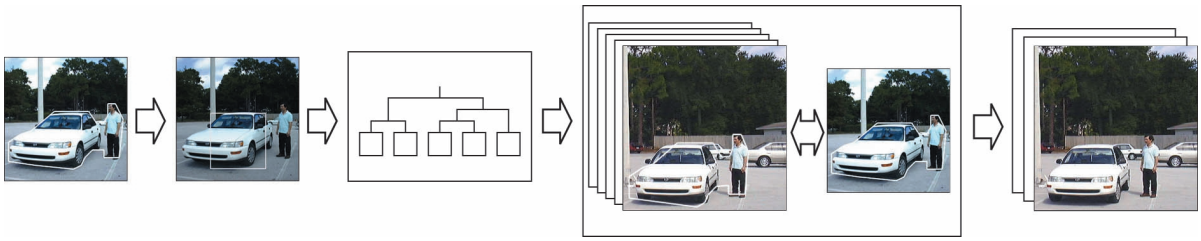
  2. We determine the core area of the NFQ using the algorithm presented in the next section. We recall that a core area is the largest square subimage that contains a noise level no greater than a predetermined limit. We compute the signature of this core area using the same technique used for the indexing subimages (discussed in Section 3). Any one of the query versions can be used in this computation to give essentially similar signatures due to the property of the Haar wavelet coefficients. This step is illustrated in Figure 7(b).

- Initial Search: We use a tiny hyperrectangle centered at the signature point of the core area as the search key to find all relevant MBRs using the R^*-tree. This step is illustrated in Figure 7(c). The result is a collection of candidate subimages enclosed in the qualified MBRs.

- Detailed Matching: For each candidate subimage, we compare it against the version of NFQ having the same number of sampled blocks. This is done by comparing the average colors of the corresponding sampled blocks falling within the query contour. Equation (1) is used to compute the matching score. This step is illustrated in Figure 7(d). If the matching score is higher than a predefined threshold, the database image containing the matching indexing subimage is returned as a query result. This step is illustrated in Figure 7(e).

- Ranking: We rank the returned images according to their matching score.

We note that a tiny hyperrectangle, centered at the signature point of the core area, is used as the search key in the Initial Search step since the R^*-tree supports range queries. For convenience, we refer to this hyperrectangle as the *query rectangle*, and the signature point of the core area as the *query point*. In practice, the query rectangle is adjusted to control the size of the returned set in the initial search.



(a) (b) (c) (d) (e)

Figure 7: Query Processing

A requirement of the proposed query processing technique is that the core area must not be smaller than the smallest indexing subimage. This should not be deemed as a limitation of the SamMatch approach since we can always employ higher sampling rates to support core areas as

small as desired. This can be accomplished without incurring more indexing overhead since we index the database on the MBRs, not individual subimages. Alternatively, we can also use non-square sliding windows for specialized applications. For instance, long rectangular windows can better support applications with many slim query objects.

5 Core Area Detection

As mentioned in the last section, we need to determine the core area in order to use the R^*-tree in the initial search procedure. We treat this issue in this section by presenting an efficient algorithm for the detection of the optimal core area given the maximum tolerable noise level (i.e., number of irrelevant pixels). Such a core area is optimal in the sense that it is maximized to capture the characteristics of the NFQ to the greatest extent. Although we could ask the user to draw this core area as part of the query, such a core area is likely not optimal. This approach also makes the system more tedious to use.

5.1 Problem Definition

Given an image frame, it defines a discrete $Z \times Z$ space (digitized equivalent of Euclidean $R \times R$ space). Let us consider an NFQ $\mathtt{S}$, and a square subset $\mathtt{L}$ of the image, such that $\mathtt{S} \subseteq Z \times Z$ and $\mathtt{L} \subseteq Z \times Z$. The number of relevant pixels in $\mathtt{L}$ can be represented as $\Re(\mathtt{L}) = |\mathtt{L} \cap \mathtt{S}|$ and the number of irrelevant pixels, or *noise*, as $\overline{\Re}(\mathtt{L}) = |\mathtt{L} - \mathtt{S}|$. Using these concepts, we can describe the problem of finding the core area as follows.

Core-Area Detection Problem : Given an NFQ $\mathtt{S} \subseteq Z \times Z$ and the maximum tolerable noise level ξ, a square $\mathtt{L} \subseteq Z \times Z$ is the *core area* of the NFQ if it satisfies the following conditions:

1. $\overline{\Re}(\mathtt{L}) \leq \xi$, and
2. for any square $\mathtt{L'} \subseteq Z \times Z$

(a) $\overline{\Re}(\mathtt{L'}) > \xi$, or

(b) $\overline{\Re}(\mathtt{L'}) \leq \xi$ and $\Re(\mathtt{L'}) < \Re(\mathtt{L})$, or

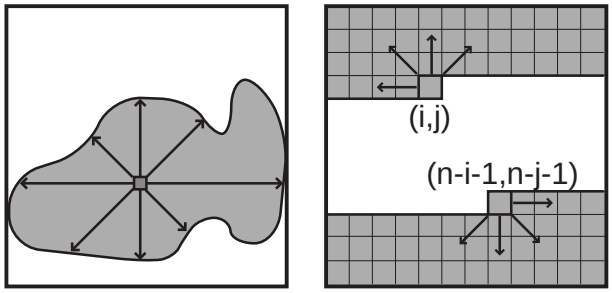
(c) $\Re(\mathtt{L'}) = \Re(\mathtt{L})$ and $\overline{\Re}(\mathtt{L}) \leq \overline{\Re}(\mathtt{L'}) \leq \xi$.

Intuitively, the above definition requires the core area be the largest square subimage with noise no more than the maximum tolerable level. Furthermore, it contains the least noise among those square subimages with the same dimensions.

Although there has been research on related problems (e.g., binary shape decomposition [23], [24]), we are not aware of any solution to the above problem. In [23], for example, 2-D binary shapes are morphologically decomposed into conditionally maximal convex polygons. Each convex polygon component is a subset of the given image, and the union of all such polygons is the original image. Obviously, this is a different problem, in which noise and the sizes of those polygons are not of concern.

5.2 An Efficient Algorithm

The core area can be found by exhaustively scanning over the NFQ. This approach however, is very expensive. We consider a much more efficient strategy in this subsection. In our technique, we first determine a set of good *seed pixels*. For each such pixel, we gradually expand an enclosing square subimage, up to the noise limit, to search for the optimal core area. A pixel is a good seed if it has good potential to expand into the optimal core area. This potential is computed based on the number of reachable pixels in each of the eight predefined directions as illustrated in Figure 8(a). For each direction, we count the number of connected relevant pixels until a noise pixel (in $\overline{\Re}(\mathtt{L})$) is encountered. These eight counts form an array called the *extend* of the pixel. We note that we can take advantage of the counts already computed for the adjacent pixels when doing the counting for a given pixel as illustrated in Figure 8(b). Thus, the extends of all the pixels in the NFQ can be calculated in one pass over the image area using dynamic programming.



(a) (b)

Figure 8: Optimal core area detection - (a) compute the extend by counting the connected pixels in eight predefined directions, (b) using dynamic programming to compute all extends in one pass

Within each `extend` array, the eight counts are recorded in ascending order (i.e., $\texttt{extend}[0] < \texttt{extend}[1] < \cdots < \texttt{extend}[7]$) to prepare for the following computation:

$$\texttt{potential}_{(x,y)} = \sum_{i=0}^{d} \frac{\texttt{extend}[i] - \texttt{extend}[i-1]}{2^i} + \frac{x + 2 \cdot y}{10^4}, \quad where\ \texttt{extend}[-1] = 0 \tag{2}$$

Using the above equation, we can determine the potential of a given pixel, at location (x, y), as a seed for expanding into the optimal core area. We explain this equation as follows. The first term increases the potential measure by an increment for each round i. The denominator, however, reduces this increment amount by a factor of 2^i due to the following reasons. The eight components of the `extend` array can be seen as the "radii" of eight concentric squares as illustrated in Figure 9. These squares form eight square bands of pixels. The innermost band is the smallest square with only relevant pixels; and it has full potential to be included in the core area. However, as we move to a band further from the center, the percentage of relevant pixels in this band typically drops. The potential of this band being a part of the core area, therefore, decreases accordingly. This characteristic is captured in our equation by using $1/2^i$ as a weighing factor for round i. In essence, Equation (2) roughly estimates the size of the core area seeded at (x, y). We use this estimate as the potential measure of this location as a seed for expanding into the optimal core area. The second term in Equation (2) is very small. It is included for the convenience of resolving ties when comparing the potential measures of neighboring pixels. Since this term is $\frac{x+2 \cdot y}{10^4}$, we favor the y value over the x value. This choice is arbitrary and has no effect on the outcome.

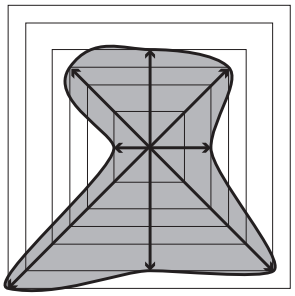


Figure 9: The eight concentric squares with respect to a pixel in the NFQ. Their radii are the `extend`'s of the pixel.

Once all the potential measures have been computed, we scan the image area to find candidate pixels with a better potential than all their adjacent pixels. Among these candidates, we select only those pixels with a potential measure greater than a certain threshold to participate in the subsequent search operation. This is done by applying a `SmartExpand` procedure to each of the final candidates. At each step, this procedure expands the square, enclosing the candidate pixel, toward the image area where the least noise is absorbed. This is illustrated in Figure 10. Although there are four directions for expanding the white square, it is done in the lower-right direction because the least noise is absorbed. This expansion process repeats until the maximum allowable noise is encountered. We note that considering only pixels with a better potential than all their immediate neighbors in the final search procedure allows us to substantially reduce the cost of our technique.

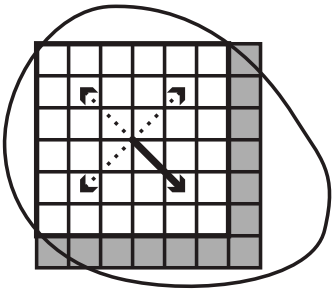


Figure 10: Applying *SmartExpand* to each good seed pixel to search for the core area.

A high-level description of the proposed core-area detection technique is given in Algorithm 1.

To illustrate the effectiveness of this technique, a core area detected by this algorithm is given in Figure 11.

Algorithm 1 *`Detecting the core area L of an NFQ`*

1. *Find the minimum bounding square, MBS, that encloses the NFQ.*
2. *For all $(x, y) \in MBS$, compute their `extend` in one pass using dynamic programming.*
3. *For all $(x, y) \in MBS$, compute their `potential` using Equation (2).*
4. *Scan the pixels in MBS to determine the set of good candidates, `candidates`, whose potential is greater than all their adjacent pixels and a predefined threshold.*
5. *Let `L` be a minimum square of size 0×0 .*
6. *Process each (x, y), in `candidates`, in turn as follows:*
   (a) *Let `L'` be a `extend[0]` $\times$ `extend[0]` square centered at (x, y).*
   (b) *Apply the `SmartExpand` procedure to expand `L'`. If the size of `L'` is now greater than `L`, then let `L` be `L'`.*

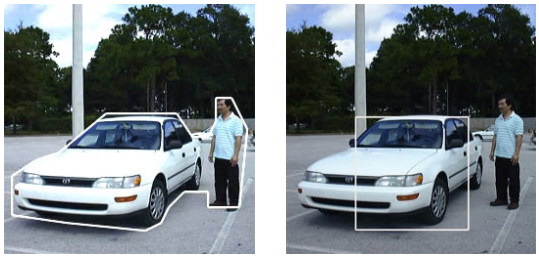


Figure 11: An NFQ (left) and its core area (right) detected by Algorithm 1 using $\xi = 10\%$

Time and Space Complexity

Assuming the size of MBS is $n \times n$, the time complexity of Algorithm 1 can be broken down as follows:

- The time complexity of Step (2) is $O(n^2)$ using dynamic programming.
- In Step (3), all the potential measures can be computed in $O(n^2)$.
- Step (4) takes $O(n^2)$ since only immediate neighbors are considered.
- Step (5) requires a constant time to execute.
- Step (6) can be completed in $O(n^2 \cdot m)$, where m is the number of candidates, each requires no more than $O(n^2)$ time to count the noise pixels.

Thus, the time complexity of Algorithm 1 is $O(n^2 \cdot m)$. Since m is generally less than 6 as observed in our experiments, the time complexity of Algorithm 1 is $O(n^2)$.

In terms of memory space, it is easy to see that Algorithm 1 requires $O(n^2)$ space, which is very reasonable. We will provide experimental results in the next section to better illustrate the efficiency of our technique.

6 Experimental Study

To assess the benefits of the SamMatch (SM) approach, we did experiments to compare it with *Local Color Histogram* (LCH) [22] and *Correlogram* (Corr.) [7]. The rationales for using these two schemes in our study are as follows:

- Local color histograms were used in [22] to allow the user to compose query images using subimages from multiple image sources. We adapt this scheme for our study as follows. Since subimages are treated independently with their own color histogram, one can easily exclude irrelevant subimages in a query. Since there is no other practical technique for NFQs, we use LCH as a reasonable approach to assess the effectiveness and efficiency of SamMatch.

- Correlogram was also selected for this study because it has been shown to provide excellent results [7] when the entire image area is relevant. This scheme is used to verify that SamMatch support NFQs not at the expense of whole-image matching.

In addition to comparing SamMatch with the above methods, we performed additional experiments to study the alternative implementation techniques for SamMatch.

We designed more than 100 NFQs of various characteristics to compare the robustness of the three image retrieval techniques. These NFQs contain diverse objects, such as roses, birds, airplanes, balloons, buildings, skies, sand, statues, food, etc. The test queries can be classified into three groups as follows:

- Type 1 (30 queries): The query image, from which we define the NFQ, has the same size as those images in the database. Furthermore, the NFQ is small covering only a small region of the query image.
- Type 2 (20 queries): The query image has the same size as those images in the database. However, the NFQ is relatively large covering almost the entire query image.
- Type 3 (more than 50 queries): The query image is smaller or larger than the size of the database images. The corresponding NFQ can be small or large.

We note that Type 3 was included in our study to capture the fact that images are typically not uniform in size. It is unrealistic and restrictive to assume that the sizes of the query images and those of the database images are always the same. We present and discuss the experimental results in the following subsections.

6.1 Comparative Studies

We describe our performance metric as follows. Let $A_1, A_2, \ldots, A_q$ denote the q relevant images in response to a query Q. The *recall R* is defined for a *scope S*, $S > 0$, as:

$$R/S = \frac{|\{A_i | rank(A_i) \leq S\}|}{q}$$

This measure indicates the percentage of the returned images ranked within the specified scope. For instance, if a query returns 30 out of a total of 40 relevant images in some scope *S*, then its R/S is computed as 30/40 or 0.75. The rationale for this metric is that the retrieved images that fall far behind in the rankings often do not make it to the user. This metric was first used in [7]. An important benefit of this approach is that we do not need to use a threshold in the image retrieval procedure. Instead, for each query we request the system to retrieve the S highest-ranked images. To compute R/S, we need to know the relevant images in the database. This was done manually in our study as follows. We used a commercial image database, called *Art Explosion*, available from *Nova Development*. This database is organized as a hierarchy of subfolders according to the image categories. We added a few images of our own in order to test specific features. In total, the test database has 15,808 images which is significantly larger than many databases used in recent studies [2],[25],[7],[12],[26],[22],[9],[11],[27],[28]. For each query, we determine the relevant images by inspecting the 500 highest-ranked images returned by each of the three image retrieval techniques, and those images in the relevant categories according to the organization of the database. In total, we inspect about 1,200 images for each test query.

6.1.1 Performance under Type-1 Queries

In this study, we compare the three techniques in processing NFQs that cover some small area of the query image. An example of these NFQs is shown in Figure 12. The R/S averages observed in this experiment are presented in Figure 14(a). It shows that SamMatch offers 40% improvement over Correlogram. Although LCH also shows some improvement, it is not as good as SamMatch which allowed us to more precisely exclude noise from the query.

(a) *query* (b) highest-ranked returned images

Figure 12: Some of the highest ranked (left-to-right) images returned by SamMatch (first row), by Corr. (second row) and LCH (third row) in response to the query (left): ”Retrieve images containing a yellow flower”.

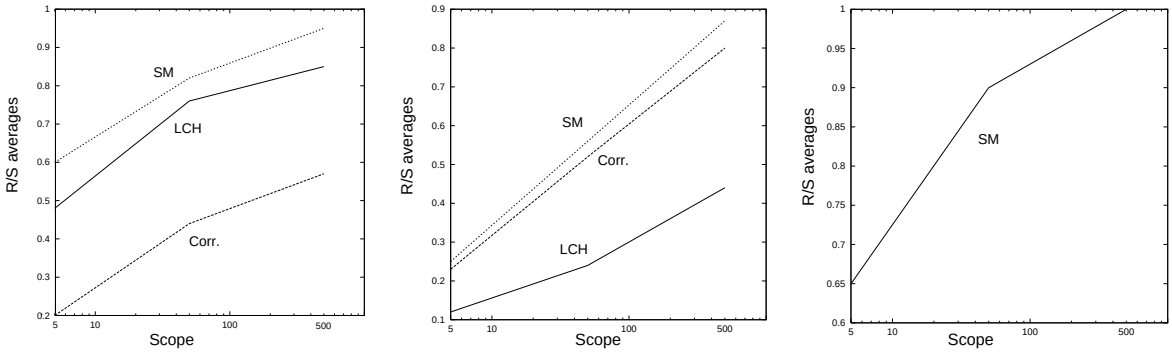
6.1.2 Performance under Type-2 Queries

query	SM: 2	SM: 21	*query*	SM: 2	SM: 16	SM: 41
	Corr.: 2	Corr.: 89		Corr.: 2	Corr.: 9	Corr.: 223
	LCH: 2	LCH: 201		LCH: 2	LCH: 93	LCH: 1258
	(a)			(b)		

Figure 13: Some relevant answers in Type-2 queries. Lower is better.

In this experiment, the objects of interest cover almost the entire area of the query image. Two such NFQ examples and their similar database images are shown in Figures 13(a) and 13(b). The results of this study are plotted in Figure 14(b). We see that LCH actually performs much worse than Correlogram under these queries. This indicates that we cannot handle NFQs by simply

partitioning images into their subimages as in LCH. In contrast, SamMatch continues to offer the best performance under this category. We also observe in Figure 13(b) that both SamMatch and Correlogram can detect objects viewed from different angles. This can be attributed to the fact that these schemes are much better than LCH in capturing the spatial layout of the colors.

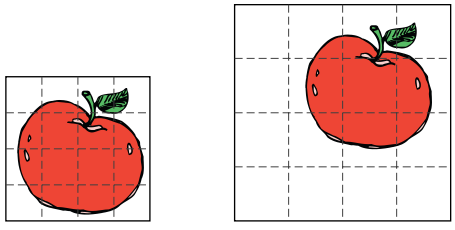


(a) under type-1 NFQs (b) under type-2 NFQs (c) under type-3 NFQs

Figure 14: R/S averages under specific types of NFQs

6.1.3 Performance under Type-3 Queries

As we have discussed, this class of queries represents many practical applications, in which database and query images do not have the same size. We observe that LCH cannot support such queries. An example is illustrated in Figure 15 showing that there is no easy way to match the two identical apples using LCH. Similarly, we cannot perform such comparisons under Correlogram. Due to these limitations, we investigated only SamMatch in this experiment. The experimental results are



(a) query (b) database image

Figure 15: There is no easy way to match the apple in the small query image with the apple of the same size in the larger database image.

presented in Figure 14(c). They indicate that SamMatch is highly effective. We show two examples in Figures 16 and 17 to illustrate the quality of this technique. They demonstrate that SamMatch is robust to scaling and translation of the objects. The objects in the retrieved images have different sizes and are found at different locations. Furthermore, the large amount of irrelevant content (occupying as much as 75%) of the database images does not affect their ranking. Interestingly, the two rightmost images in Figure 17 are ranked very low. This is because the person and the car in this image do not satisfy the spatial and scaling constraints imposed in the query. Since we are looking for images of a person about to enter a car, these two images should not be considered as very relevant. On the other hand, the 11^{th} and 17^{th} ranking of the rightmost images in Figure 16 appear to be low. This is due to the misalignment of the apple and the sliding windows in the region. Such problems can be easily addressed by using more window sizes and smaller sliding steps. We feel that the current configuration is sufficient for most applications.

(a) *query* (b) high-ranked retrieved images

Figure 16: Some images (ranked, left-to-right, 1, 2, 3, 4, 11, and 17) returned in response to query (left): Retrieve images containing an apple of any size at any location.

(a) *query* (b) some retrieved images

Figure 17: Some images (ranked, left-to-right, 1, 2, 215 and 395) returned in response to query (left): Retrieve images containing a man about to enter his car.

6.1.4 Space Requirement

In this study, we compare the three image retrieval techniques in terms of storage overhead. Since quick-and-dirty filtering and indexing were not used for Correlogram in [7] and LCH in [22], we assume that index is not used in all three techniques in this study. Thus, the storage overhead reported in this subsection is due to the feature vectors alone.

In our performance studies, the size of each image is 256×256 pixels, and there are 256 possible colors. We can compute the space requirement for the feature vector of each technique as follows:

- There are three levels of image partitioning in LCH [22]. Each database image is partitioned into 1, 4, and 16 equal-size non-overlapping subimages at levels 1, 2, and 3, respectively. Therefore, the required storage space is $(1 + 4 + 16) \times 256 \times 2$ = 10,752 bytes - "256" represents the 256 possible colors; and "2" is due to the fact that two bytes are used for counting each color since up to 256×256 = 65,536 pixels can have the same color.
- We use the distance set $D = \{1, 3, 5, 7\}$ (i.e., four color histograms) for Correlogram as in [7]. The space requirement for its feature vector is $4 \times 256 \times 2$ or 2048 bytes.
- For SamMatch, the space requirement is 113 bytes, one for each sampling block.

In summary, LCH requires more than five times the storage overhead of Correlogram in order to achieve very limited support for NFQs. On the other hand, SamMatch can take full advantage of NFQs while using only 1/16 the space required by Correlogram. These results indicate that our approach is more suitable for very large image databases.

6.2 Performance Issues Specific to SamMatch

In this subsection, we investigate the effect of the number of clusters per image on the overall performance of the system.

Building an index on individual subimages in the database would result in a very deep R^*-tree. To reduce the time required to search through this access structure, we index the database on the MBRs, each represents a cluster of subimages in a given image. The tradeoff is that each qualified MBR must be searched sequentially to find the similar subimages. The experimental results for different clustering configurations, up to ten clusters, are shown in Figure 18. The light bar indicates the average time spent on searching the R^*-tree; and the dark bar represents the average time spent on searching within the qualified MBRs to find the similar indexing subimages. The sum of the two gives the average response time of the queries for the corresponding clustering configuration. The plot shows that the performance is significantly better when the number of clusters is from 4 to 7. This is a nice property since it allows us a lot of leeway to select a good number of clusters for a given application. In practice, the database administrator can tune this parameter, as in any database system, to ensure the best performance.

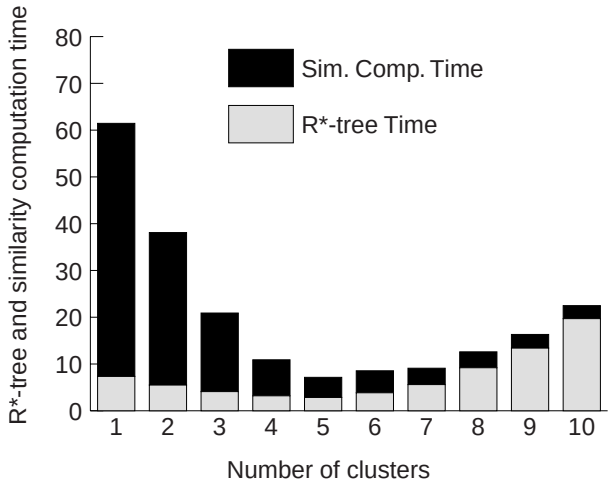


Figure 18: Average search times for different clustering configurations

To get more insight, we provide more information on three of the clustering configurations in Table 1. We explain the parameters as follows:

#MBRs: The average number of MBRs retrieved from the R^* tree. They intersect with the core area of the query.

#signatures: The average number of subimages retrieved from the R^* tree. These subimages are the signatures points fall within the qualified MBRs.

R^* tree time: The average time spent on searching the R^* tree for qualified MBRs.

Similarity Computation: The average time spent on searching within the qualified MBRs to find the similar subimages.

Speedup: Speedup with respect to a sequential algorithm that scans the entire database to compute the similarity between each image and the query image.

We observe that using a single cluster for each image is a bad idea; the benefit is limited. As we increase the number of clusters per image from 5 to 10, the number of subimages retrieved from the R^*-tree decreases from 16,371 to 5,219. Although this helps to reduce the similarity computation cost from 4.30 seconds to 2.78 seconds, the time spent in searching the R^* tree increases substantially from 2.85 seconds to 19.7 seconds.

No. of Cluster per image (c)	**One**	**Five**	**Ten**
#MBRs	2,988	963	614
#signatures	253,980	16,371	5,219
R^* tree time	7.35	2.85	19.7
Similarity Computation	54.1	4.30	2.78
Speedup	6.83	58.7	18.7

Table 1: Summary of results on different clustering configurations

7 Concluding Remarks

Query-by-example is by far the most popular query model for image database management systems. Typically, a query image, selected from a collection of general image categories, contains not only the objects of interest, but also irrelevant image areas. Excluding these noise areas from

similarity computation is essential to achieve better precision and recall. This calls for the ability to define the relevant areas precisely at query time. This flexibility, however, would render the precomputation of the feature vectors impossible since we do not know the matching areas at the database build time. To the best of our knowledge, this issue has not been studied in the literature. To address this problem, we defined in this paper a new class of queries, called *noise-free queries* (NFQs), and introduced a new similarity model, based on our *SamMatch* framework, to support them.

SamMatch enables the user to draw a contour on a query image to precisely outline the relevant areas for image matching. To support such NFQs, we take samples of each image much in the same way audio is digitized. Each sample is a small block of pixels with their average color (i.e., the most dominant Haar wavelet coefficient) pre-computed. To process a NFQ, the system only needs to compare the samples falling within the query contour. This environment also allows the system to match objects at different scales by applying different sampling rates to the query image. Furthermore, ”sliding” one set of samples over another can uncover translations of the matching areas. Our query model is also more expressive than standard QBE. The user can draw a single contour to specify spatial and scaling constraints among the query objects, or use multiple contours to disassociate them. The net effect of all these new features is a highly effective technique for image retrieval systems.

To assess the feasibility of the proposed similarity model, we investigated indexing and query processing techniques for this new environment. We implemented three systems based on SamMatch, Local Color Histogram (LCH), and Correlogram, respectively, and compared their performance. Although LCH can be adapted for NFQs, our experimental results indicate that it is not very effective. Correlogram only performs well when the objects of interest occupy almost the entire image area. Under this condition, Correlogram outperforms LCH by a significant margin. In all cases, SamMatch still exhibits the best performance. In practice, the query image may not have the same size as the database images. Only SamMatch can support this more realistic environment, and continue to offer excellent performance. We also analyzed the storage overhead incurred by

the feature vectors. It shows that SamMatch uses less than 1/16 the storage space required by the other two techniques.

References

[1] W. Niblack, R. Barber, W. Equitz, M. Flicker, E. Glasman, D. Petkovic, P. Yanker, and C. Faloutsos. The qbic project: Query images by content using color, texture and shape. In *SPIE V1908*, 1993.

[2] A. Natsev, R. Rastogi, and K. Shim. Walrus: A similarity retrieval algorithm for image databases. In *Proc. of the 1999 ACM SIGMOD Int. Conf. on Management of Data*, pages 395–406, 1999.

[3] M.E.J. Wood, N.W. Campbell, and B.T. Thomas. Iterative refinement by relevance feedback in content-based digital image retrieval. In *The Sixth ACM International Multimedia Conference*, pages 13–20, September 1998.

[4] C. Faloutsos, W. Equitz, M. Flickner, W. Niblack, D. Petkovic, and R. Barber. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3(3):231-262, 1994.

[5] Y. Gong, H.Chua, and X. Guo. Image indexing and retrieval based on color histogram. In *Proc. of the 2nd Int. Conf. on Multimedia Modeling*, pages 115–126, November 1995.

[6] M. Swain. Interactive indexing into image database. In *SPIE V1908*, 1993.

[7] J. Huang, S. R. Kumar, M. Mitra, W. Zhu, and R. Zabih. Image indexing using color correlograms. In *Proc. Computer Vision and Pattern Recognition*, pages 762–768, 1997.

[8] D. Chetverikov. Detecting regular structures for invariant retrieval. In *Third International Conference on Visual Information Systems*, pages 459–466, 1999.

[9] N. Nes and M.C. d'Ornellas. Color image texture indexing. In *Third International Conference on Visual Information Systems*, pages 467–474, 1999.

[10] J. R. Smith and S. F. Chang. Visualseek: A fully automated content-based image query system. In *Proc. of the 1996 ACM International Multimedia Conference*, pages 87–98, 1996.

[11] W.Y. Ma and B.S. Manjunath. Netra: A toolbox for navigating large image databases. *Multimedia System*, 7:184–198, May 1999.

[12] C. Carson, M. Thomas, S. Belongie, J.M. Hellerstein, and J. Malik. Blobworld: A system for region-based image indexing and retrieval. In *Third International Conference on Visual Information Systems*, pages 509–516, 1999.

[13] Y. Gong. *Intelligent Image Databases- Towards Advanced Image Retrieval*. Kluwer Academic Publishers, 1998.

[14] K. A. Hua, K. Vu, and J. H. Oh. Sammatch: A flexible and efficient sampling-based image retrieval technique for large image databases. In *Proc. of the 1999 ACM International Multimedia Conference*, pages 225–234, Oct 1999.

[15] P.K. Andleigh and K. Thakrar. *Multimedia Systems Design*. Prentice Hall, New York, 1996.

[16] M. Hiyahara and Y. Yoshida. Mathematical transform of (r, g, b) color data to munsell (h, v, c) color data. In *SPIE Visual Communications and Image Processing*, pages 650–657, 1988.

[17] E. J. Stollnitz, T. D. DeRose, and D. H. Salesin. *Wavelets for Computer Graphics: Theory and Applications*. Morgan Kaufmann Publishers, 1996.

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

[19] N. Katayama and S. Satoh. The sr-tree: An index structure for high-dimensional nearest neighbor queries. In *Proc. of the 1997 ACM SIGMOD Int. Conf. on Management of Data*, pages 369–380, May 1997.

[20] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *Proc. of the 1996 ACM SIGMOD Int. Conf. on Management of Data*, pages 103–114, June 1996.

[21] G. Sheikholeslami, S. Chatterjee, and A. Zhang. *WaveCluster*: A wavelet-based clustering approach for multidimensional data in very large databases. *The VLDB Journal*, 8(4):289-304, Feb. 2000.

[22] J. Malki, N. Boujemaa, C. Nastar, and A. Winter. Region queries without segmentation for image retrieval by content. In *Third International Conference on Visual Information and Information Systems*, pages 115–122, 1999.

[23] J. Xu. Morphological decomposition of 2-d binary shapes into conditionally maximal convex polygons. In *Pattern Recognition*, volume 29, pages 1075–1104, 1996.

[24] Y. Zhao and R.M. Haralick. Binary shape recognition based on an automatic morphological shape decomposition. In *Proc. of Int. Conf. Acoust., Sign. Proceesing*, pages 1691–1694, 1989.

[25] T. Seidl and H.P Kriegel. Efficient user-adaptable similarity search in large multimedia databases. In *Proc. of the 23rd VLDB Conference*, pages 506–515, 1997.

[26] G. Ciocca and R. Schetteni. Using a relevance feedback mechanism to improve content-based image retrieval. In *Third International Conference on Visual Information and Information Systems*, pages 107–114, 1999.

[27] Y. Gong. Advancing content-based image retrieval by exploiting image color and region features. *Multimedia System*, 7:449–457, November 1999.

[28] A. Dimai. Assessment of effectiveness of content based image retrieval systems. In *Third International Conference on Visual Information Systems*, pages 525–532, 1999.