Performing Color-Based Similarity Searches in Multimedia Database Management Systems Augmented with Derived Images

Leonard Brown[1] and Le Gruenwald[2]

[1] The University of Texas at Tyler, Department of Computer Science, Tyler, TX, 75799
Leonard_Brown@UTTyler.edu
[2] The University of Oklahoma, School of Computer Science, Norman, OK, 73069
ggruenwald@ou.edu

Abstract. In order to improve the robustness of systems that perform similarity searching, it may be necessary to augment a collection of images in a Multimedia DataBase Management System (MMDBMS) with an additional set of edited images. Previous research has demonstrated that space can be saved in such a system by storing the edited images as sequences of editing operations instead of as large binary objects. The existing approaches for performing similarity searching in an MMDBMS, however, typically assume that the data objects are stored as binary objects. This paper proposes algorithms for performing color-based similarity searches of images stored as editing operations and provides a performance evaluation illustrating their respective strengths and weaknesses.

1. Motivation

Because images and other types of multimedia data objects are different than traditional alphanumeric data, a MultiMedia Database Management System (MMDBMS) has different requirements from a traditional database management system. For example, multimedia data is typically larger than traditional data, so an MMDBMS should utilize efficient storage techniques. In addition, users interpret the content of images and other multimedia data objects when they view or hear them, so an MMDBMS should facilitate searching in those systems utilizing that content. Also, the binary representation of two images of the same objects may be very different, so an MMDBMS must perform similarity searching of images as opposed to searching for exact matches.

To facilitate similarity searching, systems typically extract features and generate a signature from each image in the database to represent its content in order to search those features in response to a user's query. For searching a heterogeneous collection of images, systems will often compare images utilizing a combination of low-level features including color, texture, and shape as in [1]. When searching images under a specific domain, such as searching a set of facial images, systems will often extract more complex features for searching such as the shape and location of noses, eyes, mouths, and hair outline [2, 3]. Irrespective of whether the features are low-level or

high-level, the underlying system typically searches for images that are similar to a query image Q by first extracting the desired features from Q and then comparing them to the previously extracted features of the images in the database.

One of the limitations of the above approach for performing similarity searching is that it is dependent upon having the binary representations of all of the images that are stored in the database so that features can be extracted or a signature can be generated. To illustrate why this can be a limitation, consider the problem of facial recognition in an example application of a crime database containing a collection of suspects' faces. It is difficult enough to match images of the same faces under varying lighting conditions [2], but the current research does not appear to sufficiently address the problem of matching a facial image from a person intentionally trying to disguise themselves utilizing either simple techniques such as adding or removing facial hair or extensive technique such as undergoing plastic surgery.

One technique for addressing the above problem is to expand a given query image Q into several query images. Each query image is created by image by editing Q using common disguises and then submitted to the database as a separate query. The results from all of the query images are combined together to form one result. This technique is analogous to text retrieval systems that augment terms in a user's query utilizing a manually produced thesaurus before searching a collection of documents. The problem with this approach for searching images is that the features must be extracted from each query image in order to search the underlying MMDBMS. Because this is a very expensive process, the time needed to respond to the query will dramatically increase.

Instead of utilizing the above technique, the system may simply augment the collection of images stored in its underlying database with edited versions of the images. So, for each image object O in the database, the system will store O along with a set of images created by transforming O with editing operation sequences representing common disguises. Thus, the system augments its database with a set of edited images derived from its original collection.

As an alternative to storing all images in such a database in a binary format, previous research [4] proposed to save space by storing the images derived from an original base image object O in a virtual format, meaning that they are each stored as a reference to O along with the sequence of editing operations that were used to create it. Displaying an image stored virtually can be accomplished by accessing the referenced image and sequentially executing the associated editing operations on it, which is a process called instantiation [4].

The use of virtual images can be beneficial in numerous application areas in addition to law enforcement. Specifically, they are useful in any applications in which users create and store new objects by editing existing ones. For example, this approach will be useful in multimedia authoring environments where several versions of various images created by a user can be archived [5] such as publishing and web design. Virtual images will also be useful in the area of medicine where plastic surgeons can illustrate the changes they intend to make to their patients by editing their photo. Finally, virtual images can be useful for business applications where users order customized objects by selecting from a list of external features and enhancements, such as ordering clothes, class rings, or other types of apparel.

As a first step in investigating approaches for searching databases augmented with derived images, this paper focuses on searching a heterogeneous collection of images by color. Systems that search by color typically extract features from each image to represent its content and then search those features in response to a user's query. When extracting color features, one common method is to generate a histogram for each image stored in the database where each bin contains the percentage of pixels in that image that are of a particular color. These colors are usually obtained by quantizing the space of a color model such as RGB or Luv into an equal number of divisions. To search for images that are similar to a query image Q, the MMDBMS can compare the histograms stored in the database to the histogram of Q. Common functions used to evaluate the similarity between two n-dimensional histograms $<x_1, \ldots, x_n>$ and $<y_1, \ldots, y_n>$ include (1) the Histogram Intersection [6] and (2) the L_p Distances [7].

$$\sum_{i=1}^{n} \min(x_i, y_i) \tag{1}$$

$$\sqrt[p]{\sum_{i=1}^{n} (x_i - y_i)^p} \tag{2}$$

The above distance functions allow the system to process nearest-neighbor queries of the type "*Retrieve the k images that are the most similar to Q*". Systems that perform similarity searches using methods involving histogram-based techniques include [1, 8, 9, 10].

The above histogram techniques are used for color-based searching in systems that assume all of the images, including any derived ones, are stored in a binary format. New color-based searching algorithms and techniques are needed for systems that save space by storing derived images virtually. The purpose of this paper is to present three such techniques and compare their respective performances to the above approach of storing all images in a binary format. The remainder of this paper has the following format. Each of Sections 2, 3, and 4 presents a different approach for searching virtual images by color. Section 5 presents the results of a performance evaluation comparing these three approaches to the conventional approach. Finally, Section 6 summarizes the paper and describes the future work.

2. Virtual Storage that Instantiates during Searching (VSIS)

The previous section described the conventional approach to searching images using color-based features which stores all images in a binary format. This paper refers to this approach as Binary Storage with Histograms (BSH), and algorithms for it are presented in Figures 2 and 3. The insertion process for BSH is displayed in Figure 1, and in it, histograms are extracted from all images as they are inserted into the system. The histogram of an image is stored in the database along with the object ID of the image. The BSH algorithm for processing similarity searches of the type "*Retrieve the k images that are the most similar to Q*" is provided in Figure 2. In the algorithm,

a histogram is extracted from the query image Q and compared to the histograms stored in the database to identify the k most similar images.

1. Extract histogram from the image
2. Store the image in database with an image ID
3. Store histogram and image ID together in the database

Fig. 1. BSH Algorithm for Inserting Images

1. Identify the query image Q and the number of desired images k from the given query
2. Extract the histogram from Q and call it HQ
3. For each histogram, H, stored in the database
    3.1 Compute the distance between H and HQ
    3.2 Remember the k images corresponding to the histograms that are the smallest distances from HQ
4. Return the identifiers of the images stored in Step 3.2

Fig. 2. BSH Algorithm for Sequentially Processing Similarity Searches

Elements of BSH are used as part of the first strategy for processing similarity searches in MMDBMSs that store edited images virtually. The strategy is based upon instantiating the virtual images into a binary format during the searching and then using conventional histogram techniques to search them. Since instantiation is performed during searching, this paper refers to these algorithms as the Virtual Storage that Instantiates while Searching (VSIS) approach. In this approach, the images that are stored virtually are directly inserted into the database without any feature extraction processing, while histograms are extracted from the images stored in a binary format. This process for inserting images using the VSIS strategy is listed in Figure 3.

1. If image is binary
    1.1 Extract the histogram from the image
    1.2 Store the image in the database using its ID
    1.3 Store histogram and image ID together in the database
2. Else
    2.1 Store the virtual image in the database using its ID

Fig. 3. VSIS Algorithm for Inserting Images

When performing a similarity search, the VSIS algorithm begins in the same manner as the BSH algorithm described earlier. Specifically, a histogram is extracted from the query image and then compared to the histograms previously extracted from the binary images. The difference is that the VSIS algorithm must determine the distances from the query image to the images stored virtually. This is accomplished by accessing the virtual images, instantiating them, and then extracting histograms from them. When this process is complete, these newly extracted histograms can be compared to the histogram of the query image in order to determine the images that correspond to the smallest distances. The algorithm is presented in Figure 4.

1. Identify the query image Q and the number of desired images k from the given query
2. Extract the histogram from Q and call it HQ
3. For each histogram, H, stored in the database
    3.1 Compute the distance between H and HQ
    3.2 Remember the k images corresponding to the histograms that are the smallest distances from HQ
4. For each virtual image, V, stored in the database
    4.1 Instantiate V
    4.2 Extract Histogram H from the Instantiated Image
    4.3 Compute the distance between H and HQ
    4.4 Update the list of k images with histograms that are the smallest distances from HQ

Fig. 4. VSIS Algorithm for Sequentially Processing Similarity Searches

3. Virtual Storage that Instantiates during Inserting (VSII)

The second strategy to performing similarity searches with virtual images avoids image instantiation during searching. Instead, the virtual images are instantiated at the time they are inserted into the database. Thus, this paper refers to this strategy as the Virtual Storage that Instantiates during Inserting (VSII) approach.

1. If image is binary
    1.1 Extract the histogram from the image
    1.2 Store the image in the database using its ID
    1.3 Store histogram and image ID together in the database
2. Else
    2.1 Instantiate the virtual image
    2.2 Extract the histogram from the instantiated image
    2.3 Store histogram and image ID together in the database
    2.4 Discard the instantiated image
    2.5 Store the virtual image in the database using its ID

Fig. 5. VSII Algorithm for Inserting Images

Figure 5 displays the details of the VSII insertion algorithm. After a virtual image is instantiated, the system extracts and stores its histogram. Once the histogram is saved, the instantiated image is then discarded leaving only the virtual image to be added to the database. Also, since the VSII algorithm generates a histogram from both binary and virtual images when they are inserted, VSII can use the same searching algorithm as BSH. Thus, the algorithm in Figure 2 can be used to perform similarity searches for VSII.

A disadvantage of both VSIS and VSII is that instantiation is a slow process. Consequently, VSIS will quickly insert images, but perform poorly when performing similarity searches. Alternatively, VSII will perform similarity searches as quickly as BSH, but it will perform poorly when inserting virtual images. The next section will

present a third strategy that is able to process similarity searches without instantiating images, thus saving time over both VSIS and VSII.

4. Virtual Storage with Rules (VSR)

The VSR algorithm processes similarity searches in systems that store edited images virtually without instantiating the virtual images. This means that VSR uses the same insertion algorithm as VSIS, which inserts virtual images directly into the database as shown in Figure 3.

The VSR algorithm for processing similarity searches is shown in Figure 6, and it consists of two major tasks. The first task is to identify the distances from the binary images in the database to the query image Q, and the second task is to identify the distances of the virtual images to Q. In the first three steps, the binary images' distances are computed using the same technique as in BSH, which is to compare histograms. The remaining steps compute the distance from Q to each virtual image based upon the Histogram Intersection [6]. Thus, the distance from Q to a virtual image V is computed the sum of the minimum values of their corresponding bins. Since the minimum values are desired, Step 4 of the algorithm determines the bins that are nonzero in the histogram of Q. The next step is a loop that executes once for each bin identified in the previous step, where the purpose of this loop is to determine the minimum values of the corresponding bins.

1. Identify the query image Q and the number of desired images k from the given query
2. Extract the histogram from Q and call it HQ
3. For each histogram, H, stored in the database
    - 3.1 Compute the distance between H and HQ.
    - 3.2 Remember the k images corresponding to the histograms that are the smallest distances from HQ
4. Enumerate the colors in the query image using the histogram HQ
5. Loop until there are no more colors in Q or all virtual images are eliminated
    - 5.1 Identify the histogram bin HB corresponding to the next color in HQ
    - 5.2 Let match equal the value in bin HB of HQ
    - 5.3 Using rules, eliminate the virtual images that do not have a value in bin HB within δ of match, where δ is the distance from Q to the k^{th} most similar image.
    - 5.4 For each remaining virtual image V
        - 5.4.1 Increase V's distance from Q by the minimum of match and its value in bin HB.
    - 5.5 Update the list of k images that are closest to Q using the distances of the remaining virtual images

Fig. 6. Rule-Based Algorithm for Processing Similarity Searches

The following steps are performed during each iteration of the above loop. First, the current bin is denoted as bin HB. Next, the algorithm identifies the value of the histogram of Q in bin HB and stores it in *match*. Given match, the algorithm must identify the value in bin HB for each virtual image. Once this value is identified for a

virtual image V, the distance from Q to V is increased by that value to comply with the histogram intersection.

The key step in the above algorithm is the identification of the value in bin HB for a virtual image. Since the proposed approach never instantiates the virtual images, no histograms are stored for them. Thus, this algorithm must estimate the value that would be in bin HB for a virtual image if it were instantiated. This estimation is accomplished using rules establishing the effects of editing operations on the percentages of colors that are contained in an image. The rules used are dependent upon the editing operations that may appear in the virtual images, which are restricted to the set of operations listed in [4], Define, Combine, Modify, Mutate, and Merge. The reason why this set is used is that it can be used to perform any image transformation by manipulating a single pixel at a time [11].

The Define operation selects the group of pixels that will be edited by the subsequent operations in the list, and the parameters to the operation specify the coordinates of the desired group of pixels, called the Defined Region (DR). The Combine operation is used to blur images by changing the colors of the pixels in the DR to the weighted average of the colors of the pixels' neighbors, and the parameters to the operation are the weights (C_{11}, …, C_{33}) applied to each of the neighbors. The Modify operation is used to explicitly change the colors of the pixels in the DR that are of a certain color, RGB_{old}, and the parameters of the operation specify that old color as well as the new color, RGB_{new}. The Mutate operation is used to move pixels within an image, and the parameters specify the matrix (M_{11}, …, M_{33}) used to shift the locations of the pixels. Finally, the Merge operation is used to copy the current DR into a target image, and the parameters specify the target image, *target*, and the coordinates specifying where to copy the DR (x_p, y_p).

The purpose of each rule is to determine how its corresponding editing operation can change a given histogram bin HB. Thus, each rule is expressed as an adjustment to the minimum and maximum bounds on the percentage of pixels that may be in bin HB when the virtual image is instantiated. The percentages are adjusted by repeatedly updating the total number of pixels that are in the image as well as the minimum and maximum number of pixels that are in bin HB for each editing operation listed in the virtual image.

Both the Combine and Modify operations only change the colors of the pixels in the current DR. Because of this, one rule for both operations is that the total number of pixels in the image will neither increase nor decrease after their application. In addition, the number of pixels that may change color is bounded by the number of pixels in the DR, denoted |DR|. So, depending on whether RGB_{old} or RGB_{new} maps to bin HB, the number of pixels in bin HB may increase or decrease by at most |DR| as a result of the Modify operation. While |DR| also serves as a bound for the number of pixels that may change as a result of the Combine operation, note that pixels within homogeneously colored regions will not change because the operation determines a new color for a pixel based on the average color of its neighbors. As a result, the rule for the Combine operation is that the adjustment to the number of pixels that are in bin HB will be so small that it can be ignored.

The Mutate operation can scale the pixels in the DR to overwrite other locations in the image, which means that it can dramatically alter the distribution of colors within an image. Consequently, the rules for the Mutate operation are based on specific

instances of its parameters. One such rule is that if the current DR contains the whole image, then the distribution of colors in the image should remain the same. Another rule is that if the parameters imply a rigid body transformation [12], then the DR will simply be moved without any scaling. Thus, the total number of pixels that may change color is bounded by |DR| as in the Modify operation.

The rules for the Merge operation adjust the percentage of pixels in bin HB based on the combination of the pixels in the DR and the colors in the target image. Table 1 provides the formulae for computing these adjustments based on the parameters of the operation, along with a summary of the rules for the Combine, Modify, and Mutate operations presented previously. In the table, |V| represents the number of pixels in the virtual image, |T| represents the number of pixels in the target image of the Merge operation, $|T_{HB}|$ represents the number of pixels in the target that are in bin HB, $|HB|_{min}$ represents the minimum number of pixels in bin HB, and $|HB|_{max}$ represents the maximum number of pixels in bin HB.

Table 1. Rules for Adjusting Bounds on Numbers of Pixels in Bin HB

Editing Operation	**Conditions**	**Minimum Number in bin HB**	**Maximum Number in bin HB**	**Total Number of Pixels in Image**
Combine *(C_{11}, …, C_{33})*	All	No change	No change	No change
Modify *(RGB_{old}, RGB_{new})*	RGB_{new} maps to HB	No Change	Increase by |DR|	No Change
	RGB_{old} maps to HB	Decrease by |DR|	No Change	No Change
Mutate *(M_{11}, M_{12}, M_{13}, M_{21}, M_{22}, M_{23}, M_{31}, M_{32}, M_{33})*	DR contains image	Multiply by $\|M_{11}\times M_{22}\|$	Multiply by $\|M_{11}\times M_{22}\|$	Multiply by $\|M_{11}\times M_{22}\|$
	Rigid Body	Decrease by |DR|	Increase by |DR|	No Change
Merge *(Target, x_p, y_p)*	Target is NULL	$\|DR\| - (\|V\| - \|HB\|_{min})$	$MIN[\|HB\|_{max}, \|DR\|]$	|DR|
	Target is Not NULL	$\|DR\| - (\|V\| - \|HB\|_{min}) + \|T_{HB}\| - \|DR\|$	$MIN(\|HB\|_{max}, \|DR\|) + MIN(\|T_{HB}\|, \|T\| - \|DR\|)$	$[MAX((x_p+x_2-x_1), \text{height of Target}) - MIN(x_p,0)+1] \times [MAX((y_p+y_2-y_1), \text{width of Target}) - MIN(y_p,0)+1]$

5. Performance Evaluation

The proposed algorithms presented in this paper have been implemented as part of a web-enabled prototype virtual image retrieval system [13] accessible from the URL http://www.cs.ou.edu/~lbrown. Modules in the above prototype were used to conduct tests to evaluate the performance of the algorithms proposed in this paper in terms of permanent storage space required, insertion time, searching time, and searching accuracy. The data set consists of 5 groups of images, where each group contains one binary image and 99 edited images stored virtually yielding a total of 500 images. The parameters of the evaluations are listed in Table 2. The binary images in the random data set were taken from a combination of various collections of images representing different application areas including law enforcement, space exploration, business, and weather [14]. Each virtual image in the data set was created by randomly selecting a binary image as the base image and randomly generating editing operations from [4] for it. Table 3 summarizes the results of the evaluation, and it illustrates the strengths and weaknesses of each of the proposed approaches.

Table 2. Parameters Used in Evaluation

Description	Default Value
Number of Images in the Database	500
Number of Binary Images in the Database	5
Number of Virtual Images in the Database	495
Average Number of Operations within a Virtual Image	2.11
Average Size of Binary Images in the Database (in bytes)	11396.60
Average Size of Virtual Images in the Database (in bytes)	70.27
Average Size of Histogram Extracted From Binary Images (in bytes)	190.40

Table 3. Summarized Results of Performance Evaluation

Approach	Storage Space (MB)	Insertion Time (sec)	Searching Time (sec)	Searching Accuracy
BSH	14.36	876.90	2.29	83%
VSR	0.09	6.54	2.33	73%
VSIS	0.09	6.54	12174.96	83%
VSII	0.15	13049.23	2.29	83%

BSH has the advantage of being simple. All images can be stored in the same format so the underlying database management system only needs to manipulate one type of image. Also, one of the main reasons of using features or signatures to search images is that it will be much faster than analyzing the images themselves during searching, so the BSH approach is as fast as the other approaches in terms of searching time. The disadvantage of BSH is that it consumes the most space of the four approaches since no images are stored virtually. Because the virtual approaches stored 99% of the images virtually, they used 99.35% less space than BSH. In addition, BSH extracts histograms from all images during insertion, so it was also 99.95% slower than VSR and VSIS when inserting the images.

One of the advantages of VSIS is that is saves space by storing images virtually. In addition, the approach does not store or extract any information from virtual images when they are first inserted, so it is as fast as VSR when inserting images. Another advantage of VSIS is that it produces the same results as BSH since it is histogram-based. The main disadvantage of VSIS is that it is extremely slow when processing retrieval queries since it instantiates all of the virtual images and extracts their histograms. Both VSR and BSH were 99.98% faster than VSIS when searching the images.

VSII extracts histograms from all images like BSH, so it shares some of the same advantages. Specifically, VSII searches as quickly as BSH, and the searches produce the same results. In addition, VSII used 98.91% less space than BSH in the tests due to it storing images virtually. The main disadvantage of VSII is that it is slow when inserting virtual images into the underlying database management system since it instantiates the images before extracting their features. In the tests, VSR and VSIS were 99.95% faster when inserting images.

VSR has the advantage that it required 99.35% less space than BSH. It also required 40.79% less space than the VSII approach since it does not store histograms for all images. The main contribution of VSR, however, is that it does not instantiate while inserting virtual images or searching them. The result is that in the tests, it was 99.98% faster than the VSIS approach when searching and 99.95% faster than the VSII approach when inserting. A disadvantage of VSR is that it does not produce the same results as the other approaches. During the tests, it was 12.3% less accurate.

Considering the above discussion, BSH is best suited for applications that are most dependent on retrieval time and not space or insertion time. Thus, the approach is the most appropriate for applications with static data sets that are searched frequently, such as medical diagnostic applications where an image taken from a patient may be compared with images of known diseases. BSH is also appropriate for data sets that do not contain sets of images that are similar; otherwise, they will waste space. VSIS is not suitable for applications that allow users to frequently perform content-based searches. Thus, VSIS is best suited for applications in which users may constantly create and add new images to the database but rarely retrieve them, such as applications that archive data. VSII is not appropriate for applications in which users will frequently update the database by adding new images. Instead, VSII is most appropriate for many of the same applications as BSH, which include applications with static data sets that are frequently searched. The difference is that VSII should be used with those applications that store edited images, such as one that displays standard automobile models that differ only in color. Finally, VSR is best suited for applications where users frequently edit existing images and add them to the underlying database with the intention that those images will also be searched frequently. Applications with these characteristics that also need to reduce the amount of space by storing the edited images virtually include web-based stores where users want to browse similar images of products that are updated quickly.

6. Summary and Future Work

MultiMedia DataBase Management Systems (MMDBMSs) focus on the storage and retrieval of images and other types of multimedia data. In data sets that contain derived images, storing them virtually allows the images to be stored using a smaller amount of space when compared to storing them in conventional binary formats. This paper presented three different strategies for searching derived images stored virtually.

This paper focused on searching a heterogeneous collection of images utilizing color features. As a result, the rules used in the VSR approach focused on identifying the effects of editing operations on color features. In order to continue with the VSR approach, it will be necessary to identify rules for retrieving images using other features besides color, such as texture and shape. Ultimately, rules must be identified for identifying the effects of editing operations on more complex features within a set of images from a specific domain, such as the effects of common disguises on the hair color, eye color, eye shape, and nose length on an image of a face.

It is also worth noting that the rules presented in this paper attempt to estimate the effect of editing operations on color histograms. So, if the rules were perfect, a system using VSR would retrieve images with the same accuracy as the histogram or feature-based approaches. An alternative approach would be to develop rules that ignore the features extracted from images altogether. Such an approach would be based on the transformation approach [15] which computes the similarity between two images as the cost required to transform one into the other. This cost could be computed directly from the editing operations listed within the semantics of a virtual image. Such an approach would allow the VSR approach to potentially be more accurate than the feature-based approaches.

Another open research area is to evaluate the effect of using algorithms that are hybrids of the VSR and VSII algorithms presented in this paper. For example, one such hybrid approach could store histograms corresponding to some of the virtual images in order to optimize both the permanent storage space and retrieval time. An open issue, then, in creating such a hybrid approach is how to determine which histograms to store that optimize those metrics.

Finally, an area of future work is to construct and compare indexes for each of the three proposed strategies for searching virtual images. While many multidimensional indexes exist for arranging histograms extracted from images, new methods are needed to efficiently arrange the data for VSIS and VSR, which do not store extracted histograms.

References

1. Ortega, Michael et al., "*Supporting Ranked Boolean Similarity Queries in MARS*", IEEE Transactions on Knowledge and Data Engineering, Volume 10, Number 6, November/December 1998, pp. 905-925.
2. Zhao, W. et al., "*Face Recognition: A Literature Survey*", ACM Computing Surveys, Volume 35, Number 4, December 2003, pp. 399-458.

3. Hsu Rein-Lien and Anil K. Jain, "*Semantic Face Matching*", Proceedings of the 2002 IEEE International Conference on Multimedia and Expo, 2002, pp. 145-148.
4. Speegle, Greg et al., "*Extending Databases to Support Image Editing*", Proceedings of the IEEE International Conference on Multimedia and Expo, August 2000.
5. Gibbs, Simon, Christian Breiteneder, and Dennis Tsichritzis, "*Modeling of Time-Based Media*", The Handbook of Multimedia Information Retrieval, Chapter 1, Grosky, Jain, and Mehrotra (Eds.), Prentice-Hall, 1997, pp. 13-38.
6. Swain, Michael J. and Dana H. Ballard, "*Color Indexing*", International Journal of Computer Vision, Volume 7, Number 1, 1991, pp. 11-32.
7. Jagadish, H. V., "*Content-Based Indexing and Retrieval*", The Handbook of Multimedia Information Management, Chapter 3, Grosky, Jain, and Mehrotra (Eds.), Prentice Hall, 1997.
8. Stehling, Renato O., Mario A. Nascimento, and Alexandre X. Falcão, "*A Compact and Efficient Image Retrieval Approach Based on Border/Interior Pixel Classification*", Proceedings of the 11th International Conference on Information and Knowledge Management, November 2002, pp. 102-109.
9. Oria, Vincent, et al., "*Similarity Queries in the DISIMA Image DBMS*", Proceedings of the 9th ACM International Conference on Multimedia, October 2001, pp. 475-478.
10. Park, Du-Sik et al., "*Image Indexing using Weighted Color Histogram*", Proceedings of the 10th International Conference on Image Analysis and Processing, 1999.
11. Brown, Leonard, Le Gruenwald, and Greg Speegle, "*Testing a Set of Image Processing Operations for Completeness*", Proceedings of the 2nd International Conference on Multimedia Information Systems, April 1997, pp. 127-134.
12. Gonzales, Rafael C. and Richard E. Woods, Digital Image Processing, Addison-Wesley Publishing Company, Reading, MA, 1993.
13. Brown, Leonard and Le Gruenwald, "*A Prototype Content-Based Retrieval System that Uses Virtual Images to Save Space*", Proceedings of the 27th International Conference on Very Large DataBases (VLDB), 2001.
14. Binary Images obtained from the Internet at http://nix.nasa.gov/browser.html, http://www.cs.ou.edu/~database/members.htm, http://www.weathergallery.com/tornado-gallery.shtml, http://c2.com/~ward/plates/, http://www.toyota.com, last accessed January 7, 2002.
15. Subrahmanian, V. S., Principles of Multimedia Database Systems, Morgan Kaufmann Publishers Inc., San Francisco, CA, 1998.