EXACT MATCHING IN IMAGE DATABASES

Peter Bosch, Alex van Ballegooij, Arjen P. de Vries, Martin Kersten

Center for Mathematics and Computer Science (CWI), Amsterdam, The Netherlands

ABSTRACT

We believe there is a niche for exact sub-image searching in image databases. When only a sub-image is searched for, traditional image searching approaches fail because the sub-image features usually represent only a small fraction of the global image features. Said differently: it is difficult to find the sub-image in the global features. We present a new technique to find sub-images in a (possibly large) image database. The users articulate their interest in a sub-image, *i.e.* an object within an image, in a so-called precise multi-spot query. We search our spatially oriented color-based image database image-by-image for matches with this multi-spot query.

1. INTRODUCTION

We argue that by exactly formulating sub-image queries, in contrast to whole image queries, users are able to retrieve qualitatively better results from an image database compared to traditional approaches. In our system, users formulate image queries based on colors, shapes and spatial locations. They select ranges from sample images they value highly in the form of *image spots* and within these spots they select what they find interesting in terms of colors and shapes. The spatial inter-relation of a number of these spots in the same image is used to tie together various parts of an object and it constitutes an extra search constraint.

Once a query is formulated, we use our search engine to match the query in an exact manner in our image database. Our image database stores for all images it keeps, coloring information and spatial location of colors. Our image search engine then iteratively examines each image sub-area in our database against the *multi-spot* query and selects those that satisfy the query fully. Hence, contrary to existing approaches, our search engine is an exact matching search algorithm.

Many approaches and systems today make use of global image features. At best, in those systems, users articulate a query based on their interest (*i.e.*, local features), and the image search engine matches the local features onto global image features. Such global image features are often pre-calculated and are stored in a database as a feature vector. Searching becomes a simple matter of comparing the local features with a global image feature vector. Although such an approach deems simple and straightforward, it is, in our view, not a solution to finding sub-image areas. If, for example, a user is trying to find a face in an image, this face often represents a small fraction of the entire image. It is quite likely that other areas of the image dominate the feature vector thereby reducing the chance of finding the sub-image.

Our approach is an attempt in making a general image search engine, which can be used to search for sub-image features through colors. We have tried to stay away from image domain specific features, and instead build a search engine with which all kinds of domain specific image search engines can be built. All internal search policies except those related to image index storage are available for and alterable by users: every part of the search can be directed and controlled through user-settable parameters. The reason for exposing these policies to the surface is our experience showing that hidden search policies inside the engine often clutter the search results: we have had more than a single occasion in which we traced search results back to policy decisions inside the search engine rather than image features.

Our system stores the local spatial-based image features (*i.e.* spatial color information). For each image, we store a quad-tree representation of the dominant colors of the image. Hence, an image can be rebuilt by traversing the quad tree. The storage mechanisms are described separately [3, 2]. Our system also implements our search engine that drives the spot-based image search operations through the stored coloring information.

To find images in our database, users need to formulate precise image-spot queries themselves. Although it can be quite cumbersome to formulate such queries – users have to remember the interesting features for a particular domain themselves – it is also enlightening: by analyzing the results on queries, users learn why their query led to the found results.

2. RELATED WORK

Many image retrieval systems use the (color) histogram approach for matching (parts of) images. In the histogram approach a histogram is constructed from the image and images are compared using a distance function. Examples of this distance function are the L_1 distance metric as used by Swain *et al.* [14] or a more elaborate distance function such as the weighted quadratic distance function [10].

QBIC is a system with which users can query, for example, an image database and search for colors, textures and shapes [6]. Users can query the database through 'query by example,' or can actually look for particular color or texture component in the images through a color or texture histogram search. (Weighted) Euclidean distance functions are used to combine the multi-dimensional vectors.

VisualSEEk [11] implements a segmentation approach where all similarly colored regions are extracted from the image. All regions found can be used as queries. The images are extracted by iterating over all possible combinations in a strongly reduced color set. VisualSEEk's approach allows searching for multiple regions in images and allows for spatial constraints of the regions.

In BlobWorld [4] a high-dimensional space is constructed from the image's L*a*b color space, its textures, and, regions that have the same color/texture. These regions are called blobs. Feature vectors are matched with a weighted Euclidean distance function. The improvement of BlobWorld with respect to QBIC is that the index holds 'things' (*i.e.* objects) rather than low-level 'stuff.' It

is shown that BlobWorld works well for distinctive scene images (*e.g.* tigers, cheetahs or zebras). However, there are classes for which the system does not work well (in particular planes). The problem with this particular class is that the object that is looked for has a common color and texture.

Noise Free Queries (NFQs) [15] are queries where the object a user is interested in is described in a precise manner. NFQs are matched in multi-scale manner onto the image set by using a similarity measure. In some respect, a NFQ is similar to a single-spot match. The difference between Vu *et al.*'s approach and our approach is that we have taken the queries a step further: we describe actual shapes and describe the interrelationship between single spots through a multi-spot vector.

There exist many other systems and solutions that implement content-based image retrieval [8, 9, 13, 7, 12]. Unfortunenately, space considerations prohibit us to describe all of them seperately.

3. SPOTS

We define an image spot as a region from an image that, according to the user, is an important part of the image. Basically, users select spots from images, when they want to find examples of such sub-images in other images. To select a spot, users first select an image area before informing the system which spot colors are interesting.

Through selecting colors, shapes of objects can become apparent. Object-based image retrieval systems such as BlobWorld [4] pre-calculate objects through colors and/or textures. Often, however, objects are colored in multiple colors. For example, a color of skin as is shown in Figure 1 has two colors: only by combining the colors, an ellipsoid-like shape can be distinguished. It is for this reason we do not pre-calculate shapes: currently it is unknown how to combine colors to construct an object *a priori*. Instead, our index only stores the location and coloring information of dominant colors in images. During search, we reconstruct shapes *a posteriori* from the dominant colors and store those shapes in so-called structures. Our search engine only matches structures.

We allow two sets of colors to be selected in spots: fore- and background colors. Foreground colors are used to select the object the user is interested in, background colors are used to position the object in its background. In the face example, the eyes and mouth are embedded in a background of skin colors. For now, users can only indicate a single set of fore- or background colors, but we envision our system to be extended with support for conjunctions and disjunctions.

The embedding of foreground structures in backgrounds can be indicated in the four directions. Each of the directions can be selected separately or through any combination. This means that for the face example, users can select where they would like to find the background sky: above, next to or underneath the eyes and mouth. This embedding constraint allows users to remove those objects from the answer set that can never satisfy the request.

All spots (or in fact, structures) have a shape. Currently, there is a trivial shape 'gravity point' and a more elaborate shape 'ellipsoid.' The gravity point shape is primarily used for small area objects (*e.g.* the eyes and mouth of a face). The ellipsoid shape is used to describe whole areas. The skin of a face can be selected through an ellipsoid.

The set of spots inside an image form a so-called *multi-spot*. The multi-spot records the distance d and angle θ between the gravity or center point of the shapes. *I.e.*, the multi vector shows how to reach the next spot from the current spot.

4. THE DATABASE

For each image, the database stores the location and size of the dominant colors for an area in a quad-tree representation. The reason for using this type of index is primarily for optimization purposes: it turns out that with our quad-tree segmentation algorithm, a 100 K-pixel image can be approximated by only a few kilobytes worth of database tables.

The image index itself is stored in Monet, a main-memory database engine [1]. All image structures are laid out in main memory in such a manner that we do not have to worry about disk I/O performance. We run the database on a 32-processor, 64 Gb main-memory machine, which is capable of storing a color index for more than 1 M images. Note that at the moment (by far) we do not use the full capacity of our server's memory.

The quad-tree representation of the image is not used for the actual search process itself. Instead, when the database is accessed for searching and a number of fore- and background colors are presented, the quad trees are converted into structures through the help of a 128x128 image matrix: relevant colors are copied from the quad tree into the image matrix. Afterwards, the image matrix is processed to extract the relevant image *structures* or shapes, which are stored in Monet temporary tables.

Structures are the primary search data structure. For each matrix line, the structure describes the starting x and y location and the width and height of the line. All search engine steps know how to deal with these structures.

5. SPOT SEARCHING

Once a multi-spot query is formulated, the query is sent to the search engine for processing. In the database, a query processor parses the query, it derives the search parameters and drives the search operation by sending search requests to the actual database.

Database searching happens in several phases. First, our database index is combined with the user's selected colors, and the structures are synthesized from the quad trees. Based on these structures, gravity points are calculated, which are used for the background embedding (or enclosure) tests. Finally, all remaining structures are used for a multi-spot match and all foreground structures are discarded that do not satisfy the user selected multi-spot.

The reason for splitting up the search process in several phases in a pipeline is for optimization purposes. We try to prune away as many structures as possible in simple and cheap tests before applying expensive operations on the intermediate results. Also, caching techniques may help to reduce processing time by keeping intermediate search results available.

As said, in the first phase the quad-tree color index is traversed and, with the user selected colors, organized in structures. For each image considered, our search engine creates a data structure for all *closed* regions in either the fore- or background colors. The created structures are made available in our database for further processing.

Once the structures have been created, our search engine calculates the horizontal and vertical gravity points of each structure. These gravity points are the initial main point of interest for the structure and are used for testing the embedding of the foreground in the background structures. Note that the gravity point here does not have to be the real gravity point of the object: if the desired object is occluded or overlapped by an other object in almost same

color (*e.g.* a yellow and green balloon on green fields), the calculated gravity may be located at a different location due to the influence of the overlapping object. Although this may seem wrong, the effects are limited: the gravity points are, in this phase, only used for testing the embedding which does not change.

Next, the foreground gravity points are tested for their embedding in the background structures. For each image, all foreground gravity points are analyzed by scanning over the background structures once. Structures that do not satisfy the embedding constraint are discarded.

At the final step, all foreground structures are interrelated with each-other. Currently two types of shapes can be detected: gravity points and ellipsoids. The computed gravity points are used to correlate gravity points of multi-spots. The search engine adjusts the distances and angles of a multi-spot match in the original image to what is available in the considered image. For example if the distance between two spots in the original image is d_o, while the distance in the considered image is d_c, a multiplication of $m = \frac{d_c}{d_o}$ is used to match the other points in the multi-spot. Likewise, the search engine also calculates a constellation of the multi-spot α. This constellation represents the rotation of the multi-spot in other images. To allow for some degree of *fuzzy* matching, the user can supply a number of parameters that determine maximum offsets to both m and α.

Recall that the calculated gravity points are not correct. When an object is occluded by another similarly colored object, the calculated gravity point is shifted from the desired gravity point. This implies that candidate structures may be dropped while they would have matched the criteria. This problem is reduced by applying gravity point matching only to distinctive objects in the query. We will work on this problem for future versions of our search engine.

Other shapes, like ellipsoids, are matched differently. Since there can be many places where to position an ellipsoid even when m and α are restricted by the user, only a single instance of an ellipsoid is considered. First, a center point for the ellipsoid is calculated based on earlier matched gravity points.[1] Based on the center point, m, α and ϕ, the constellation of the ellipsoid, an internal representation of the desired ellipsoid is calculated. This internal representation of the ellipsoid is matched with the foreground structures. When a sufficiently large portion of the ellipsoid matches, the selected area is approved as a match.

6. RESULTS

We have not yet performed a formal evaluation of our system and our approach. Instead, we have selected a number of sample queries and fired those on our database. This section reviews the strengths and weaknesses learned from analyzing the results obtained.

We have formulated a number of queries with our query formulation tool.[2] One of the first queries we performed was a face detection query. Here we used a random image with a face from the COREL image set, and selected those parts from the image we thought were interesting for the request.

Figure 1 shows both the original image and the spots we have defined inside the face image. When faces are searched in an image database, the most characteristic parts of the face are the eyes, the mouth and the ellipsoid shape of the head. Hence, when formulating a query, those are precisely the parts of the image that are

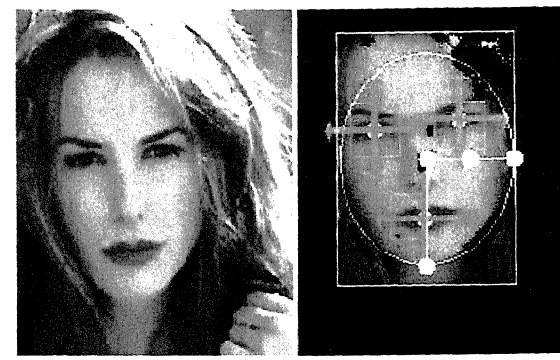

Figure 1: Face query.

selected as query spots. As for selecting shapes, by selecting fore- and background colors, shapes of objects become apparent. For example, when selecting the skin color in the face spot, all areas in this color are shown as a single shape. Likewise, the lips of the face have a distinctive red color.

Once fore- and background colors have been identified, the user can attach a base shape to the selected object. For the eyes and the mouth of the face it is sufficient to select a gravity point: a point that identifies the location of the object. An ellipsoid is used to identify the shape of the head.

The arrows in the image show the enclosure of the foreground object within the background object. Here, the eyes and the mouth need to be enclosed by a background (the skin) in all directions. Since we would like to select all images with a face, we do not specify a background for the ellipsoid: any background structure applies.

When this particular query is fired at a limited set of images in our database, a number of images are retrieved as is shown in Figure 2. Clearly, there are some good results for this request. The original query image is found, but also two additional images of faces. It is obvious that the first image is a match – both images are look-alikes. If, however, the histograms of the spots are compared, the images are radically different. So, this image would probably not be found with traditional whole-image search algorithms. The same goes the image from the Africa set in the COREL image set. The reason why this image matched is because in our color space, the color of skin is racially independent.

Clearly, there are some undesired results in the answer set. These images match for the simple reason that they match the query specification precisely. So, the query was not articulated sufficiently precise to remove these from the answer set.

As described earlier, we use the Itten-Runge color space for segmentation and searching. In particular, we use a limited set of this color space containing a total of 16 colors, ignoring saturation and brightness. The sparseness of this color space used causes a number of unwanted images to be selected easily. In the future, we will revert to other, more elaborate, color spaces such the L*a*b or L*u*v color spaces [5].

7. FUTURE WORK

The most important change that we would like to perform is to remove the dependency on gravity points to match shapes. Currently, at least two gravity points are required to match spots. In an

[1] Note that this implies that a multi-spot search must include at least two gravity point shapes.

[2] http://riem.cwi.nl:8080/~peterb/spotter.html

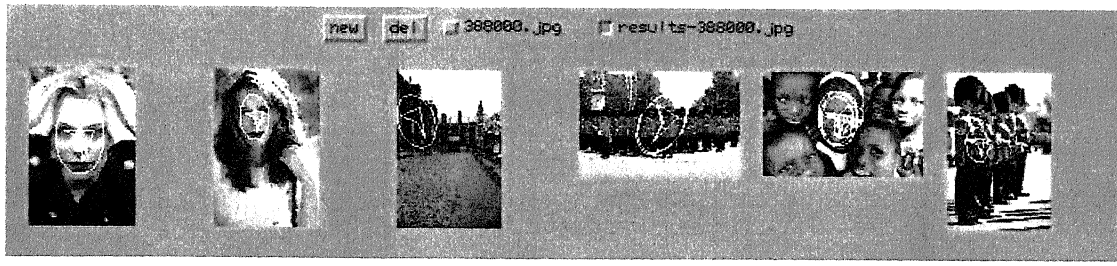

Figure 2: Face query results.

Figure 3: Butterfly queries.

image, many candidate places exist for matching arbitrary shapes, given a variable rotation and scaling. We use the gravity points to find a possible location of the shape, and then test if that area is covered by the shape's colors. However, given the earlier described problems with gravity points, we would like to match shapes, rather than gravity points. Consider, for example, Figure 3. The query one would like to ask here is: 'find all images with the five ellipsoids in the colors of the wings and body of the butterfly'. Currently we do not have support for this type of query.

Another limiting factor in our search engine is the reliance on a single set of fore- and background colors. Given, for example, the butterfly that is shown in Figure 3. The query formulated in this figure will only find other butterflies that have brown wings. It is quite likely, however, that a user finds other butterflies in different color settings similarly good results. Hence, the used fore- and background colors that are used in the selection criteria should not be limited to a single set. We would like to include conjunctive and disjunctive clauses as well in our queries.

8. SUMMARY

We have built a prototype of a search engine with which users can articulate their precise interest in a sub-image. With these precise descriptions, we can find sub-images rather than whole images. We think this is an important contribution to the image search field, which has up until now primarily focussed on whole image matching, or matching with global features.

We realize that our system is not finished. There are a number of areas where we can extend and improve our work. The most important improvement in our search engine will be sub-image matching on image structures, which are instantiated in an on-line manner.

But, given the results we have obtained so-far, we believe that our approach is a valid one.

9. REFERENCES

[1] P. A. Boncz and M. L. Kersten. MIL Primitives for Querying a Fragmented World. *VLDB Journal*, **8**(2):101-19, 1999.

[2] Peter Bosch, Arjen de Vries, Niels Nes, and Martin Kersten. A case for Image Quering through Image Spots. *SPIE 13th International Symposium: Electronic Imaging 2001* (San Jose, CA), pages 20–30. IS&T/SPIE, January 2001.

[3] Peter Bosch, Niels Nes, and Martin Kersten. Navigating through a forest of quad trees to spot images in a database. INS-R0007. Center for Mathematics and Computer Science (CWI), Amsterdam, 2000.

[4] C. Carson, M. Thomas, S. Belongie, J. M. Hellerstein, and J. Malik. Blobworld: A System for Region-Based Image Indexing and Retrieval. *Third International Conference Visual Information and Information Systems*. Springer-Verlag, 1999.

[5] Alberto Del Bimbo. Chapter 2, Image retrieval by colour similarity. In *Visual Information Retrieval*, pages 97-99. Morgan Kaufmann Publishers, Inc., 1999.

[6] Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jon Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, David Steele, and Peter Yanker. Query by Image and Video Content: the QBIC system. RJ-9949 (87908). IBM Research Division, Almaden Research Center, 30 March 1995.

[7] Jing Huang, S. Ravi Kumar, Mandar Mitra, Wei-Jing Zhu, and Ramin Zahib. Spatial Color Indexing and Applications. *International Journal of Computer Vision*, **35**(3):245–268, 1999.

[8] Greg Pass and Ramin Zahib. Histogram refinment for content-based image retrieval. *IEEE Workshop on Applications of Computer Vision*, pages 96–102. IEEE, 1996.

[9] Greg Pass and Ramin Zahib. Comparing Images Using Joint Histograms. *Journal of Multimedia Systems*, 7(3):234–240, 1999.

[10] Harpreet S. Sawhney and James L. Hafner. Efficient Color Histogram Indexing for Quadratic Form Distance Functions. RJ-9572 (83578). IBM Research Division, Almaden Research Center, 28 October 1993.

[11] John R. Smith and Shih-Fu Chang. Tools and Techniques for Color Image Retrieval. *SPIE*, volume 2670. SPIE, 1630-9.

[12] D. McG. Squire, W. Müller, H. Müller, and J. Raki. Content-based query of image databases, inspirations from text retrieval: inverted files, frequency-based weights and relevance feedback. *The 11th Scandinavian Conference on Image Analysis*, pages 7-11, June 1999.

[13] Markus Stricker and Alexander Dimai. Color Indexing with Weak Spatial Contraints. *SPIE'96*, pages 1–12, 1996.

[14] Michael Swain and Dana Ballard. Color indexing. *International Journal of Computer Vision*, 7(1), 1991.

[15] Khanh Vu, Kien A. Hua, and JungHwan Oh. A noise-free similiraty model for image retrieval systems. *Storage and Retrieval for Media Databases, SPIE 2001* (San Jose, CA), volume 4315, pages 1–11. SPIE, January 2001.