A case for Image Querying through Image Spots

Peter Bosch, Arjen de Vries, Niels Nes, Martin Kersten
CWI, Netherlands

ABSTRACT

We present an image spot query technique as an alternative for content-based image retrieval based on similarity over feature vectors. Image spots are selective parts of a query image designated by users as highly relevant for the desired answer set. Compared to traditional approaches, our technique allows users to search image databases for local (spatial, color and color transition) characteristics rather than global features.

When a user query is presented to our search engine, the engine does not impose any (similarity, ranking, cutoff) policy of its own on the answer set; it performs an exact match based on the query terms against the database. Semantic higher concepts such as weighing the relevance of query terms, is left to the user as a task while refining their query to reach the desired answer set. Given the hundreds of feature terms involved in query spots, refinement algorithms are to be encapsulated in separate applications, which act as an intermediary between our search engine and the users.

Keywords: content-based image retrieval, exact matching, query articulation, main-memory databases.

1. INTRODUCTION

In this paper we argue that image querying through image 'spots' is a better way of formulating an image query compared to traditional query formulation methods based on feature vector spaces. We define an image spot as a 'significant' portion of a query image and this spot is used to search alternatives in a (possibly) large image set. More precisely: a spot consists of one or more closed regions, or spot elements, in an image. When a spot consists of more than one spot element, we call the spot a *multi-spot*. Any part of an image can be a spot, and henceforth there are a huge number of possible spots per image. A system that enables searching on a collection of image spots can be seen as a third-generation content-based image retrieval system. Our system is such a third-generation system.

Recall that first generation image querying systems are based on color histogram matching. A sample image is characterized by its color/texture contents and images are compared by, for example, computing the Euclidean distance similarity function[1] between the images' color histograms. The recurring problem with this approach is that often spatial considerations are ignored and that only complete images can be compared. Images with the same color build-up as the sample image, but that are semantically different, are also presented as *good* answers in this query scheme.

Second generation content-based image retrieval approaches also consider spatial constraints and object layouts. A number of systems combine both color and/or texture contents of images with spatial information[2–4] and allow search operations on 'combined features.' Unfortunately, this feature database is computationally expensive to construct and search. Nevertheless, the tendency is that using spatial constraints improve the quality of the query results.

Third generation systems extend the former by enabling the user to formulate more precisely a compound query expression using image spots as focal points. Image spots in our system are construed in terms of the color components of salient objects as well as the spatial layout of these components when we are searching through our database. Contrary to second generation systems we do not store 'objects' in our database indexes: instead we only store the spatial location of dominant colors of images. By carefully matching dominant colors we reconstruct 'objects' while performing query searches in the database.

Although a spot-based approach dramatically increases both the query and answer domain, we can rely on the hypothesis that not all portions of an image are equally important for most query formulations. Especially in images containing many objects one requires more precision to articulate a query compared to traditional methods. Conversely, a multi-object query answer image may contain a small spot of high interest otherwise hidden behind the statistics of the complete image.

Correspondence: peterb@cwi.nl

To limit the scope of our project, our search engine currently performs exact spatial-color matches.* Users formulate a query in terms of spatial relationships between colors manifest in images. The search engine maps the user query request onto the indexes stored in the database and, based on maximum scaling and constellation offsets, only matches those images from the database which satisfy the user constraints precisely. A random sample and some statistics of the answer set are returned to help the user in refining the query.

The search engine is kept as simple as possible. It was our aim to abolish any search policy from the search engine and to make these explicit for and controllable by users or user applications. The prime reason is that hiding search policies inside the engine makes the search engine, in our view, too static: it is likely that with built-in policies the search engine can only be used for a limited number of types of search operations. By expelling idiosyncratic (black magic) search policies from the search engine, different kinds of user interfaces can make use of the same search engine. Semantically higher level concepts are also not the scope of our work: we acknowledge that they may be indexed as well and we envision applications using these concepts to prune the database or answer set to improve the response quality.

We are currently building up a database to hold more than 1 M images to perform spot-search experiments. We believe that only when systems are capable of storing, and searching through such large numbers of images, professional use of an image database can be fruitful. A large sample database avoids the trap that retrieval/similarity algorithms are biased to the few clusters emerging in a feature space built from just a few thousand (highly different) objects.

2. RELATED WORK

Many image retrieval systems use the (color) histogram approach. In this 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.*[1] or a more elaborate distance function such as the weighted quadratic distance function.[5] This section overviews a number of approaches.

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. In Faloutsos *et al.*[7] and Sawhney *et al.*[5] it is shown how to construct an efficient filter to reduce the computational consequences of the QBIC high dimensional histogram comparisons.

VisualSEEk[8] 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[9] 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.

Huang *et al.*[4] present the *color correlogram*. With a color correlogram, spatial correlation of colors is captured. Pairs of colors and their distances are kept in a histogram and distances between such histograms are computed. It turns out that this approach is quite effective to differentiate between color similar, but structurely different images. This work shows that using color transitions rather than colors is a good approach in finding similar images.

Pass *et al.*[2,3] present an extension on the histogram approach by defining classes of histograms through *color histogram refinement* and *color coherence vectors*. The idea is to construct histograms from a global histogram by grouping related features (*e.g.* textures) into the same histogram. In the refinement phase the image is split in the 75% inner and 25% outer pixels and two histograms are constructed, and in the second phase spatially connected pixels with the same color are grouped (the color coherence vector). Only related histograms are compared. It is shown that this approach leads to considerable better results when compared to plain histogram comparisons. In those cases where the coherence vectors performed badly, a poor choice of color space was accountable.

*We are in the process of integrating texture-based searches in our engine.

Gevers *et al.*[10] present the retrieval capacity of various color models for 500 images. It is shown that with 16 color and color transition bins, the recognition rate of 70 test images in the original set of images is between 80% and 97% for the various color models. In this test color histogram matching worked better than the color transitions histogram matching. In Stricker *et al.*[11] it is observed that 'Indexing by color histograms works only if the histograms are sparse, i.e., most of the images contain only a fraction of the number of colors of the color space.' It is for this reason that Gevers *et al.* used 16 bins: there was a good tradeoff between the sparseness of the histograms and the computational complexity to match the histograms.

We are considering over 1 M images. Given the constraints by Stricker *et al.* using color histogram matching requires a high dimensional color space to maintain sparseness. Unfortunately, as is shown by Beyer *et al.*,[12] increasing the dimensionality also reduces the effectiveness of the queries: all results have approximately the same distance to the query point, with the exception of the exact match.

Stricker *et al.*[13] show that color indexing and content-based image retrieval based on color moments, rather than color histograms and a minimal amount of spatial information leads to meaningful results. Their system defines five fuzzy regions (the center of the image and the four corners), and for each of these regions the first three moments of the color are measured and stored in the index. Querying is performed by searching the index on the moments. It is shown that this spatial information better results are achieved compared to queries without the spatial information.

The Viper[14] system uses many discrete features to represent the content of the images. Viper exploits inverted file indexing structures to manage several thousands of features per image, and matches images to a query image based on the absence and presence of features in both. Our approach differs from Viper in that we target for larger sets of images and we emphasize on image spots for the specification of queries.

3. IMAGE SPOTS

A spot may consist of a single spatially closed region, a spot element, or it may consist of multiple spot-elements (a multi-spot). We experimentally observed that single spot elements do not select objects from images well: too many images satisfy the constraints as imposed by a single spot element. Multi spots are more selective due to their spatial constraints. Given these observations, we think that the majority of image spot queries are construed from multi spots. The search engine has been constructed with multi-spots in mind.

Figure 1 presents a multi-spot in a sample image from the COREL[15] image set. When alternative faces are sought using this query image, one may identify the eyes and the lips as the significant regions of the image. The search is subsequently based on the features maintained in the database for all possible regions.

Since our database does not know anything about image objects (such as eyes and lips), we need to formulate our query in database index terms. As described before, our database holds the dominant color information with their spatial location. So, to formulate a query, the selected multi-spot is transformed into query based on colors. Such a transformation is performed through our spot-editor, which merely pans the regions of interest.

An example output of the spot-editor is shown in Figure 2. The spot-editor allows users to select from spots *foreground* and, when required, *background* colors of interest on a spot-element basis. In each spot element the color band marked with an 'x' at the top represents the foreground color, the bottom 'x' represents the selected background color.† The height of the bar represents the number of pixels for that color. Currently we have defined a background a color that *encloses* a foreground color in all directions. We realize that different types of spatial relationships between foreground and background colors are possible and we are extending our spot-editor to support such relationships. In terms of the example, the color of skin is a good candidate as a background color (color 0).

As is also shown in Figure 2, users can only select between a few colors. We use a reduced version of the Itten-Runge color space[16] with only 12 color and 4 grey scale bins for color selection. In our experience a color space with many bins (*e.g.* a 256-bin Hue color space) is far too selective for querying. With such high dimensional color spaces it is quite likely that only the original image satisfies a multi-spot color-based query. Although the 12+4 color space is currently built-in in the spot-editor, we can replace this color space for other color spaces when they prove more effective. For example, color spaces as introduced by Gevers[10] and Geusebroek[17] may replace and/or strengthen our color space.

Implicitly, by selecting the multi-spot and the foreground colors through the spot-editor, the user automatically selects the spatial constraints of the query. The spatial constraints between the spots in the multi-spots are determined by the gravity points

†Unfortunately, the bottom 'x'-es do not show on the gray scale paper version of this article. For all spot-elements, color bin 0 is selected as background.

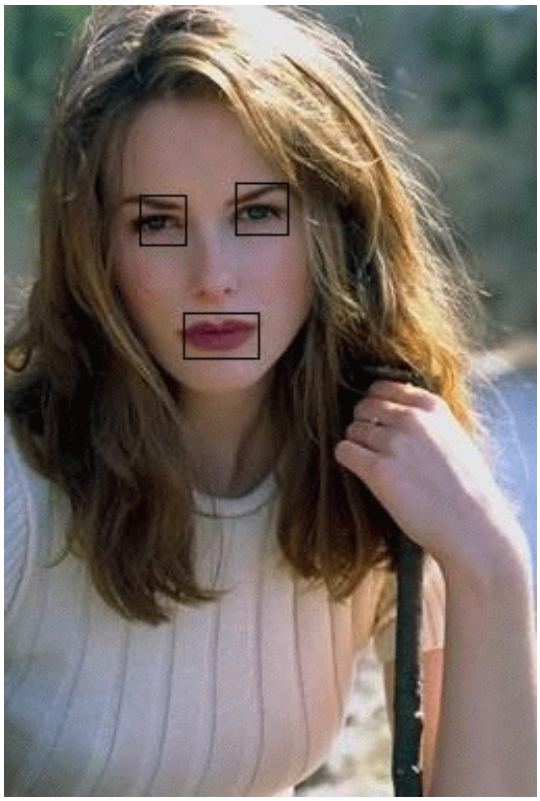

Figure 1. A multi-spot in a COREL image.

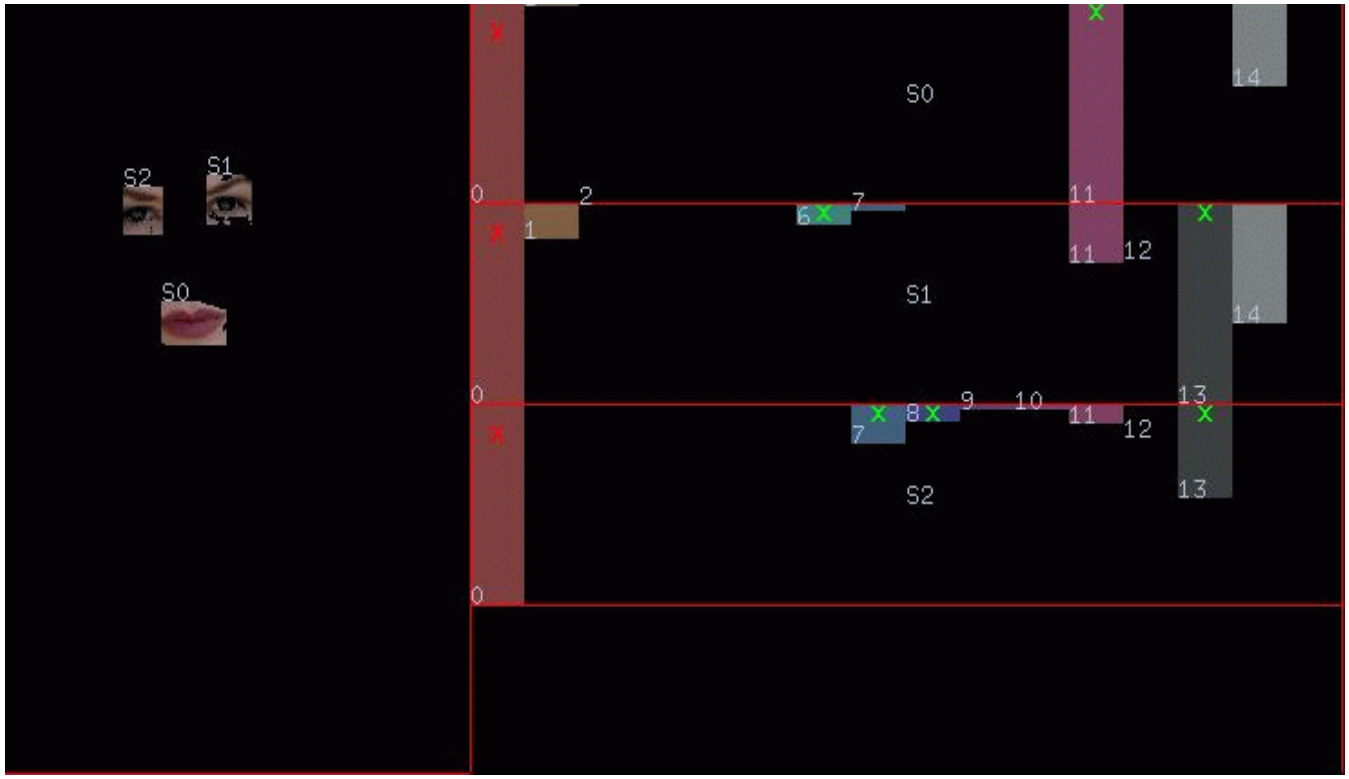

Figure 2. The spot-editor X11-window output.

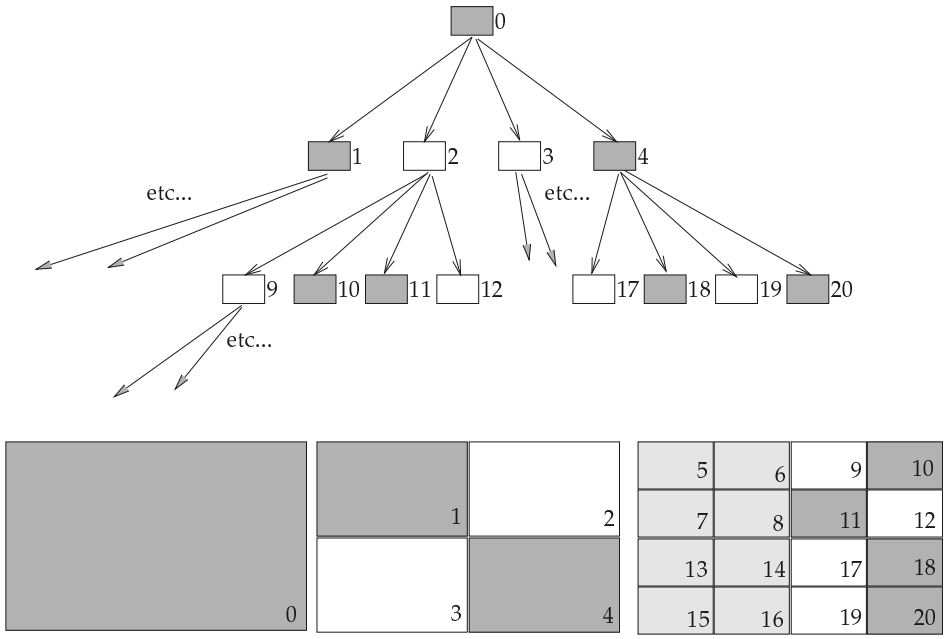

Figure 3. Quad-tree segmentation.

of the foreground colors. The inter-spot-element vector angles and the absolute distances between them characterize the color layout of the query.

What is not shown in Figure 2 is that additional distance and constellation offsets constraints can be given. Users can specify the margins of the foreground inter-sub-spot distances and how much inter-spot-element vector angles may be altered for a good match. For example, when the user selects 180 ° to be the maximum constellation offset, an image with an upside-down face is considered as a good candidate.

4. DATABASE INDEX

Our image database currently contains for each image a dominant-color quad tree,[18] much like as is done in Smith *et al.*[19] and by Lu *et al.*[20] The quad tree describes in each node of its tree, the location of the dominant feature and the feature itself (the features' color). An approximation of the image, or any significant region, can be reconstructed by a top-down materialization of the quad tree in a 2-dimensional image array.

For the color quad tree, we convert a standard RGB image to a 256-bin HSV color space and we determine the dominant color of the overall image. When this color is available for at least 10% of the pixels in all four quadrants of the image, a color feature with a location ID and color value is defined for that rectangle. Unlike previous approaches, we subsequently wipe out all pixels in the dominant color, and the algorithm proceeds with each of the four quadrants separately as a 'new' image. When not all quadrants have at least 10% of the dominant pixels, no feature is defined for that area and the quad tree leave remains empty.

Figure 3 shows a sample image quad tree and the area the quad tree describes. In this figure, shaded boxes represent features, while white boxes represent those areas where no feature is defined. At each level, features are numbered from left to right and top to bottom in a Z-ordering.[18] The figure presents the location IDs as well. The image shown in the example quad tree has a dominant color for the entire image (feature 0), it has a feature for the upper-left corner (feature 1) and bottom-right corner (feature 4). An image can be reconstructed by logically AND-ing all materialized features, *i.e.* by 'adding up' all rectangles as is shown in the bottom half of the example.

The reason for storing the images in quad trees is their storage efficiency. We only record the nodes with features in our database. Given the example in Figure 3, only features 0, 1, 4, 10, 11, 18 and 20 are recorded. By breaking up the images through the aforementioned segmentation algorithm, we can describe the 384x256 COREL images in, on average approximately 3700 quad tree nodes. Each point only requires three bytes of encoding: two bytes for the spatial location of the feature in the image and one byte to encode the contents of the node (*i.e.*, the color value). Compared to the original file size we require only

a fraction of the JPEG file size for storing all required color information of the image. More information on the segmentation algorithm can be found separately.[23]

We store all features in Monet, our main-memory database server.[24] Monet runs on a 32-node, 64 Gb main-memory machine, so when only color quad trees are considered, we can store more than 6 million images in main memory. As said earlier, we are currently integrating texture information in the form of quad trees as well. When both color and texture quad trees are considered the database our database can hold more than 2 million images in main memory.

Although we can store a fairly large amount of images in our database, this paper does not describe the problem of searching through millions of images. Instead, we assume the existence of functionality to reduce the large image set to a set of about 10,000 images that *may* be of interest for the exact match queries we are performing.[‡] Our search engine answers an exact match query on a sub-set of the images quickly enough so that users can browse interactively such a reduced set.

5. QUERY PROCESSING

Once the multi-spot query has been formulated, our database server matches the query to its index. We perform this search in a number of steps: first users can pre-select images based on color values, they can match spots using spot-element constraints and they can match the entire spot contents based on inter-spot-element spatial constraints.

We would like to see users to browse the image database much like people use the AltaVista search engine[§]: often users do not know which keyword works best when searching for a web-page a priori. Only after some experimentation, users find out which type of selection works best for their purposes. By allowing users to play with the selection criteria, they can learn *how* to articulate a query well. We would like to find a similar way of dealing with colors and spatial considerations: a priori one does not know what color feature, features or spatial constaints makes the spot element stand out.

Once image pre-selection has taken place and maximally 10,000 images are available for further analysis, we materialize the quad trees into free-formatted structures for exact matching purposes. These free-formatted structures describe all enclosed areas that satisfy a particular color constraint. For example, given the woman in Figure 1, the skin, lips and eye colors represent such constraints. Note that a fore- or background constraint may consist of multiple colors. For example, the woman's eyes are encoded in colors 6, 7, 13 and 7, 8, 13 for the left and right eye, respectively. Our spot-editor enables users to specify such constraints.

The reason for materializing is that our quad trees are not suitable for exact matching operations. When performing exact match operations, we need to match multi-spots based on what is actually represented in the image. To do so from a quad tree, we need to traverse the entire tree from top to bottom to reconstruct an approximation of the original image. In theory we could have kept the quad tree as basic storage mechanism while performing exact match operations. However, since the quad trees are not fully populated (as is shown by Figure 3), and no explicit parent/child references are kept in the database, finding neighboring features of an area implies at least one full tree traversal to build an index on the quad tree.

Once structures have been identified, the actual multi-spot matching can start. We allow the users to prune the set of structures through a variety of selection criteria: by adding more restriction criteria fewer results are returned, or by relaxing some of the criteria more results are returned. The following selection criteria exist:

- Weights. Make sure the relative weight of all foreground values from the original multi-spot correspond with a candidate multi-spot from the database. The user has the freedom to choose between various weight relationships ranging from a simple ordering (the number of pixels for the lips must be larger than those for the eyes) to precise measurements.
- Distance and constellations. Make sure that the distance and the angles of the vectors between the gravity points of the selected areas correspond to those in the query multi-spot. Here, the user can select a level of freedom in matching both distance and constellation. For each set of structures the search engine first matches the first two spot elements from the multi-spot, and determines a distance factor and constellation offset to allow for scaled and rotated multi-spots. The remainder of the multi-spot is matched with the adjusting distance factor and constallation offset. When all spots in a multi-spot match, the areas are considered for further processing.

[‡] Other sub-groups within our database group are looking into such issues.

[§] www.altavista.com

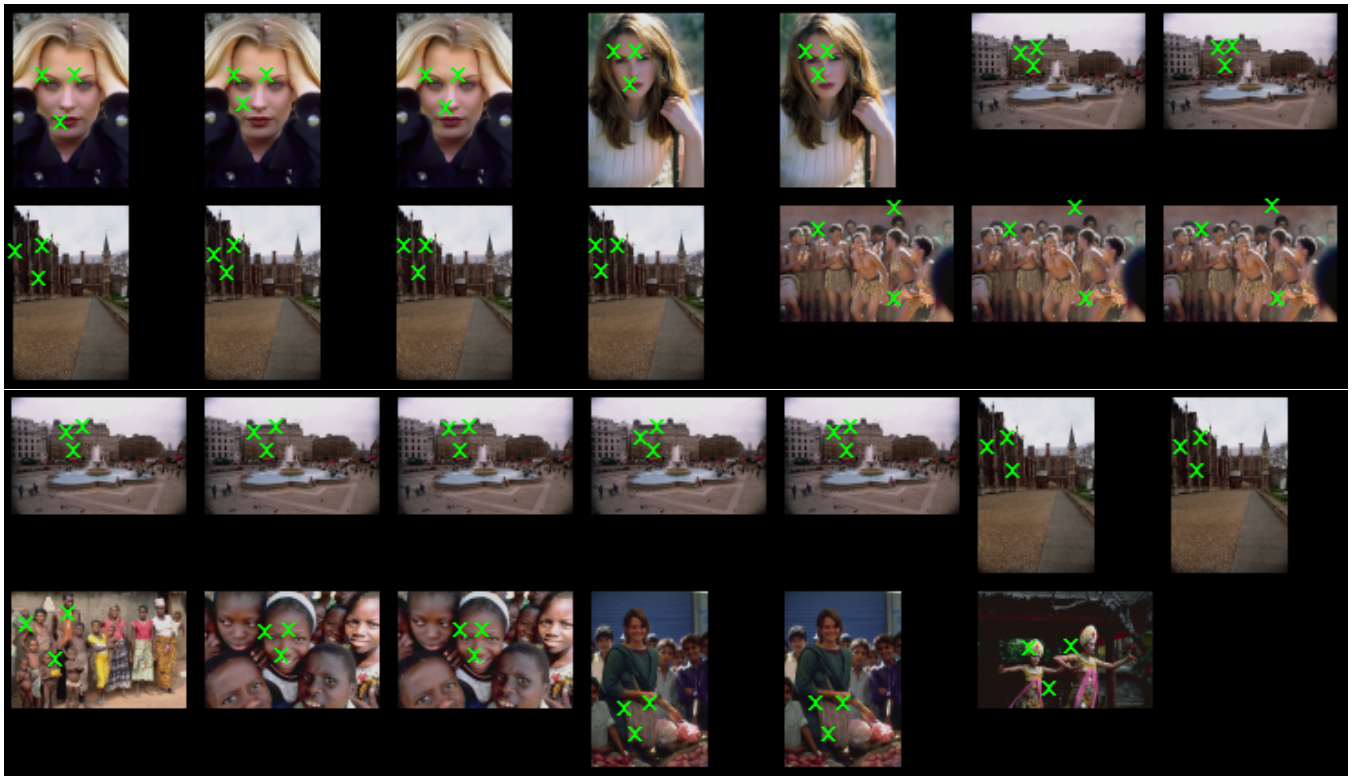

Figure 4. Results from a simple search on lips and eyes.

- Backgrounds. Make sure a selected foreground structure is enclosed by a background structure. The way the foreground is to be enclosed is determined by the user. For example, the woman's mouth and eyes from Figure 1 must be enclosed in all directions by the color of skin. This constraints works by identifying a background color in all directions from the gravity point of the foreground color.

The spot-editor drives the database search operations by sending selection and matching requests. Given the example in Figure 2, this query is evaluated as follows:

- Find all gravity points of the combined structures with color 6, 7, and 13 for the left eye;
- Find all gravity points of the combined structures with color 7, 8, and 13 for the right eye;
- Find all gravity points for the structures with color 11 for the lips;
- Make sure all these gravity points are enclosed by the structures with color 0 for the skin;
- Make sure the relative distance and vectors in database images correspond to the distance and vectors in the original multi-spot.

Figure 4 presents thumbnails from the answer set and the exact locations inside the thumbnails for the lips and eyes. It shows that there are a few good results, which are highlighted in Figure 5. The 'x' marks present where in the image the spot elements are found. Figure 4 also shows a number of matches inside images of buildings or images of streets. These results are, according to the original query, correct.

The images of buildings and streets are not what we were searching for. It turns out that in order to find other faces with eyes and lips, it is important to constrain the results even more with additional selection criteria. For this query type, it is likely that additional texture selection criteria improve the quality of the query answer set. As has been shown by Fleck *et al.*[21] and Forsyth *et al.*,[22] texture is a good selection criterium for detecting skin. It is relatively straightforward to implement a query to

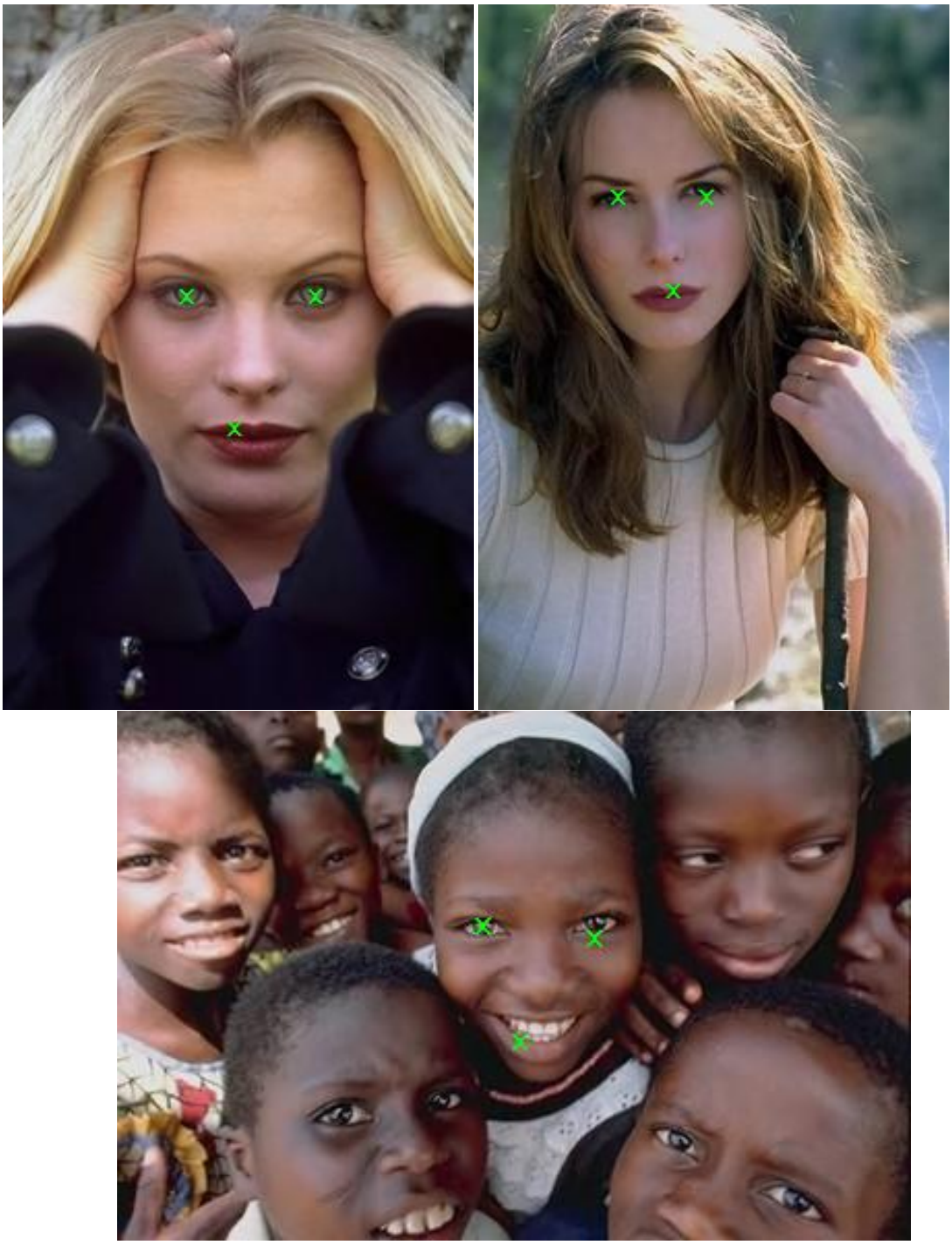

Figure 5. Samples from found images.

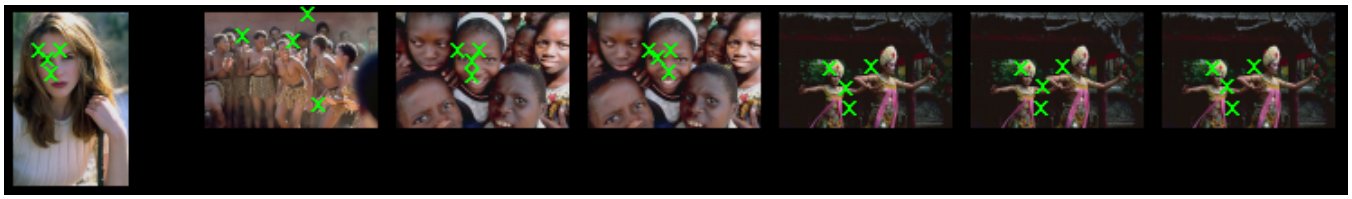

Figure 6. Refined results from a simple search on lips and eyes.

select on the color of the lips as foreground color and to reduce the number of candidate spots by constraining a skin colored background *and* a skin texture background.

Alternatively, when only considering colors, an extra spot element removes the building and the street images. When we define an extra spot element in the center of the three existing spot elements, the images that are shown in Figure 6 remain valid. The reason the blond woman is dropped from picture set is because we currently only search for gravity points of foreground colors, which is a built-in policy. This hidden policy will also be removed from the search engine and made explicit.

As for the example presented here, the processing speed of the search engine is currently too slow. To generate structures from the image quad trees based on the color selection, each materialization and detection of structures consumes about 80 ms per image on a 300 Mhz MIPS-R10K processor. We find about 220 structures per image based on one fore- and one background selection per spot element.

Once the structures have been materialized it takes about 30 ms per image to calculate the gravity points on the foreground structures, to perform the enclosure test with the backgrounds and to perform the distance and constellation selection. The background test reduces the number of candidates by about 97%. The distance and constellation selection reduces the number of final results by another 93%.

We realize that our current search engine is currently not fast enough for interactive use. However, we are currently using only a single processor for query evaluation, and we have not yet fine-tuned the search process. Several development tracks lay ahead to speed up our search process for interactive use.

6. DISCUSSION

Our approach is quite different from most approaches reported in the literature. The reason for this fresh start is that whole-image approaches do not solve the problem of finding sub-image areas, while often users are only interested in a specific image part. Secondly, common believe is that the computer vision problem at large will not be solved in the near future. Therefore, we conjuncture that much more research effort should be invested in helping the user to articulate his queries and to help the user in finding the right image through steering.

As a database group, we started out by using database techniques for solving the content-based image retrieval problem. In our first efforts[23] the database index in the form of the quad tree itself played an important role: it is an easy, though arbitrary, segmentation algorithm. The hypothesis was to rely on the filter capability of quad trees to reduce the search space quickly, e.g. by only analyzing high levels of the quad trees to reduce the search space to match spots. We also expected to use color transitions of neighboring features inside the quad trees as auxilirary selection criteria. Analytical results of the selection rate of color transitions (or color transition strings) are sufficiently high to reduce a large image set (more than 1 M images) quickly to proportional amounts.

Unfortunately, none of our approaches on raw quad trees worked (well). The fundamental problem with such an angle of attack is that by only analyzing a part of the index (*e.g.* top layers of a quad tree), one is not necessarily analyzing a representative part of an image, but merely a part of an image that may not exist: the detail makes the image. So, we had to revert to the 'old-fashioned' approach of analyzing the actual structures inside an image like BlobWorld[9] as a secondary phase.

Fortunately, our quad tree segmentation approach is not entirely useless. The number of features that are required for an approximation of an image are low through the quad tree segmentation approach. Without any compression, we can describe a 100 K pixel image in approximately 3,700 features with a storage requirement that is a fraction of the JPEG file size. However, given that we only use the quad tree for storing the coloring (or texture) information, other segmentation algorithms may work equally well.

Throughout this project we learned that it is important to match image spots in the database through an exact matching process. When one is searching for a particular object from a sample image inside other images, there are often a number of local features of the object that characterize the object. For example, when looking for faces the eyes, lips and the skin make the object distinguishable. Alternatively, when looking for balloons, the buckets, the color and texture of the balloon and the blue sky are interesting features to search for.

Being able to perform exact matches of sub-spots in a database of images differentiates our approach from (most) existing approaches. We allow the users to precisely describe *what* they are interested in, instead of enforcing a system wide (black box) search policy on the users. The search engine in our database simply performs the search based on the boundaries as given through the query.

The ability to refine queries when the results are a superset of the requested images allows users to browse the image database: it enables users to 'zoom in' on the requested image. In the example that we have given, we have only presented the results of a small set of images. However, if the input set is large (*e.g.* 10,000 images) and there are more results selected through a query than can be presented on a screen, possibly a number of good candidates are not shown. In fact, we contemplate on taken a random sample when there are more results than can be shown. This implies that there is a possibility that the desired image is not selected. By allowing the user to refine the request, it is likely that the desired image are shown in a later stage of the query process.

Often systems dealing with content based image retrieval discuss *recall* and *precision*. Here, recall means the fraction of the correctly found images on the total amount of relevant images and precision means the fraction of irrelevant images on the total amount of found images. We think recall and precision are not the terms to validate our search engine. Both recall and precision deal with a semantically higher levels than what can be searched for through our search engine. Instead we envision a layer on top of our search engine that helps the user to formulate queries based on these semantically higher concepts. This layer will support issues such as weighing the relevance of results and keep a history of queries and their results.

7. SUMMARY

In this paper we have presented multi-spot queries as an alternative for content-based image retrieval. Our approach is that by using local image features, users can search on sub-image areas rather than images as a whole. It is our thesis that these sub-images are of more interest to the users than whole images.

To zoom in on the desired images, we allow users to selectively prune the image set with color and spatial constraints. Through a multi-step search algorithm in our main-memory database, users can control the search process in all detail and continuously relax or refine the search. Our search engine must be a policy-free search engine, it only needs to provide the mechanism for searching to users or user applications.

Our system is not finished. Currently we have a prototype system available with color-based image features with which we can perform color-based retrieval experiments. It is imperative that we need to extend the feature space with different types of image features. We are currently integrating texture features in our system.

Acknowledgements

This project is supported by the Multimedia Information and Analysis (MIA) project, which is funded through the Dutch ICES-KIS program.

REFERENCES

1. M. Swain and D. Ballard, "Color indexing," *International Journal of Computer Vision* **7**(1), 1991.
2. G. Pass and R. Zahib, "Histogram refinment for content-based image retrieval," pp. 96–102, IEEE, 1996.
3. G. Pass and R. Zahib, "Comparing images using joint histograms," *Journal of Multimedia Systems* **7**(3), pp. 234–240, 1999.
4. J. Huang, S. R. Kumar, M. Mitra, W.-J. Zhu, and R. Zahib, "Spatial color indexing and applications," *International Journal of Computer Vision* **35**(3), pp. 245–268, 1999.
5. H. S. Sawhney and J. L. Hafner, "Efficient color histogram indexing for quadratic form distance functions," tech. rep., 1993.
6. M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker, "Query by image and video content: the qbic system," tech. rep., 1995.

7. C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz, "Efficient and effective querying by image content," *Intelligent Information Systems* **3**, pp. 231–62, 1994.
8. J. R. Smith and S.-F. Chang, "Tools and techniques for color image retrieval," vol. 2670, SPIE, 1996.
9. C. Carson, M. Thomas, S. Belongie, J. M. Hellerstein, and J. Malik, "Blobworld: A system for region-based image indexing and retrieval," Springer-Verlag, 1999.
10. T. Gevers and A. W. Smeulders, "Pictoseek: Combining color and shape invariant features for image retrieval," *IEEE Transactions on Image Processing* **9**(1), pp. 102–119, 2000.
11. M. Stricker and M. Swain, "The capacity of color histogram indexing," pp. 1–5, 1994.
12. K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is 'nearest neighbor' meaningful?," pp. 217–235, Springer-Verlag, (Berlin Heidelberg), 1999.
13. M. Stricker and A. Dimai, "Color indexing with weak spatial contraints," pp. 1–12, 1996.
14. D. M. 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," pp. 7–11, 1999.
15. Corel, *Individual Images*. Corel.
16. A. Del Bimbo, *Chapter 2, Image retrieval by colour similarity*, pp. 97–99. Morgan Kaufmann Publishers, Inc., San Francisco, 1999.
17. J.-M. Geusebroek, R. van den Boomgaard, A. W. Smeulders, and A. Dev, "Color and scale: The spatial structure of color images," 2000.
18. H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison Wesley, 1990.
19. J. R. Smith and S.-F. Chang, "Quad-tree segmentation for texture-based image query," ACM, 1994.
20. H. Lu, B.-C. Ooi, and K.-L. Tan, "Efficient image retrieval by color contents," No. 819 in Lecture Notes in Computer Science, pp. 95–108, Springer-Verlag, 1994.
21. M. Fleck, D. Forsyth, and C. Bregler, "Finding naked people," pp. 592–602, 1996.
22. D. A. Forsyth and M. M. Fleck, "Identifying nude pictures," 1996.
23. P. Bosch, N. Nes, and M. Kersten, "Navigating through a forest of quad trees to spot images in a database," tech. rep., 2000.
24. P. A. Boncz and M. L. Kersten, "Mil primitives for querying a fragmented world," *VLDB Journal* **8**(2), pp. 101–19, 1999.