# Visual Image Query

Krešimir Matković
VRVis Research Center in Vienna, Austria,
Matkovic@VRVis

László Neumann
Maros u. 36
H-1122 Budapest, Hungary
lneumann@cg.tuwien.ac.at

Johannes Siglaer
Institute for Design and Assessment of Technology,
TU Vienna, Austria
jsiglaer@pop.tuwien.ac.at

Martin Kompast
Institute for Design and Assessment of Technology,
TU Vienna, Austria
mkompast@pop.tuwien.ac.at

Werner Purgathofer
Institute of Computer Graphics and Algorithms,
TU Vienna, Austria
wp@cg.tuwien.ac.at

## ABSTRACT

The explosion of storage media size and bandwidth has led to huge image databases. Methods are needed to find a particular image based on a crude description by the user. Keywording is not only tedious, but also subjective and therefore often incorrect. Available visual query systems have different properties, and are mostly based on some image transformation. An alternative visual query system is introduced, which finds an image similar to a user drawn sketch, or to any other reference image. A descriptor is created for each image in the database, and for the query image. Descriptors are compared in order to find the best matches. Descriptors are computed by inserting a limited number of quasi-random rectangles in the image, and computing the average colors of the rectangles. Furthermore, a reduced color histogram is computed and stored in the descriptor. The difference between descriptors is calculated as the weighted average of CIE LUV differences between corresponding rectangles. Using the Contrast Sensitivity Function this average is adapted to the users perception. The metric used for comparing images operates in the original image space, which makes the whole algorithm intuitive and easy to understand, and enables the comparison of images sections, as well.

## Keywords

Image Retrieval, Color Layout Query, Digital Image Matching, Human Perception

## 1. INTRODUCTION

The amount of electronic images an average user is confronted with has exploded in the last decade. This trend seems to continue with further bandwidth and storage media size expansion. As a result of such an explosion huge image databases are quite common today. Unfortunately, as the size of the image database grows, the time required to find an image increases as well. When some critical number of images is reached, it is no longer possible to find a specific image fast, by using thumbnail view only. Of course, it is possible to describe images using lists of keywords, but it seems that there are just a few of us willing to type keywords for each scanned or downloaded image. Furthermore assigning keywords is not only tedious, but the description via textual attributes is not very objective. Another way to find a specific image or a group of similar pictures in a large database consists in using a visual query system. Such a system tries to find an image similar to an image that is drawn by the user, or that is supplied in some other way (existing image, low-quality scan...) An ideal system would be capable of finding a particular item, e.g. a company logo, or a human face, in a large database of arbitrary images. There is a lot more of research to be done before such a system is constructed.

Existing systems can find an image with similar color-layout, or can recognize a person if predefined constraints are met. So a person can only be identified if the image shows a portrait of the person and is compared within a database of portraits, and not of arbitrarily images. There are a lot of image query systems, either commercially available or still in research phase. Only the best known will be mentioned here, and Eakins et al.[4] give an excellent overview of almost all existing methods.

Probably the most well known systems are: QBIC[5] developed at IBM, VIR Image engine[1] from Virage, Inc. and Photobook Project[16] developed in the MIT Media Lab. Another work well known in the computer graphics community is Multi resolution Image Querying by Jacobs et al.[10]. This method was the first introduced to the community, and yields impressive results. Since our basic approach is similar to theirs, the differences to their method will be stressed out more explicitly.

A new visual image query method is introduced. It is simple to implement and simple to understand, and gives satisfactory results. This simplicity is the main strength of our method.

The main idea is to create image descriptors in a preprocessing phase, to store them in a database, and to use them for the query. When the user requests a query, a descriptor is computed for the new query image, and it is compared to the descriptors in the database. The results are sorted, and the best matches are shown to the user. That's exactly the same as Jacobs et al.[10] do. What makes our method different is the way how the descriptors are created, and the metric that is used in comparing the query with the target images. Our work relies strongly on the metric proposed by Neumann et al.[15]. We will not describe it in detail, due to space limitations, and the interested reader should have a look at the original paper.

The simplest solution would be to use the mean square error method to compare the query image with the target images. Although simple, this approach does not yield satisfactory results. In order to achieve better results, some aspects of human vision should be taken into account. Girod[7] has nicely described what's wrong with the mean square error. Adding, e.g. just a little offset to each pixel results in a relatively large error, although our visual system would consider these two images as almost the same. On the other hand, changing just a small area in the image significantly, would not be represented accordingly to our judgment. Just as many researchers before[17, 6, 10, 3] we used the Contrast Sensitivity Function, as proposed by Mannos et al.[11], as the dominant vision phenomena when developing our system.

The main idea is to decompose the images using a series of different sized rectangles. Sizes and positions of rectangles are distributed semi-randomly, just as described in Neumann et al.[15] Jacobs et al., on the other hand, use wavelets to decompose images. Once rectangles are placed in the image, the average color of every rectangle is computed, and the difference to the corresponding rectangle in a target image is computed. Differences are weighted according to the Contrast Sensitivity Function, because our visual system is not equally sensitive to differences at various spatial frequencies.

Two more mechanisms were added, which makes the method different from the query introduced by Jacobs et al.[10]. An important feature of the human visual system is that we are poor color estimators. There are people having so called absolute pitch who can estimate the sound frequency exactly, but there is no human possessing an ability to estimate color wavelength exactly. We can not remember colors exactly and therefore we cannot reproduce them later exactly either. In order to reduce errors caused by remembering a wrong color, we reduce colors in target and query images. The user is allowed to use only “basic” colors for sketching the image. Finally, we use a reduced histogram as an additional criterion in our query. The histogram is built on the reduced color set (actually the originally reduced color set is reduced once more), containing only nine main colors (like red, blue, ...). This reduction minimizes the error caused by not remembering exact colors even more.

Let us explain the Contrast Sensitivity Function, and the whole algorithm in some more detail.

## 2. CONTRAST SENSITIVITY FUNCTION

As already stated, contrast sensitivity will be one of the main human vision characteristics that is used in our metric. The model of Contrast Sensitivity Function introduced by Mannos and Sakrison[11] will be used. The Contrast Sensitivity Function tells us how sensitive we are to the various frequencies of visual stimuli. If the frequency of visual stimuli is too high we will not be able to recognize the stimuli pattern any more. Imagine an image consisting of vertical black and white stripes. If the stripes are very thin (i.e. a few thousand per millimeter) we will be unable to see individual stripes. All that we will see is a gray image. If the stripes then become wider and wider, there is a threshold width, from which on we are able to distinguish the stripes. The contrast sensitivity function proposed by Manos and Sakrison is

$$A(f) = 2.6 \cdot (0.0192 + 0.114 \cdot f) \cdot e^{-(0.114 \cdot f)^{1.1}} \tag{1}$$

$f$ in equation 1 is the spatial frequency of the visual stimuli given in cycles/degree. The function has a peak of value 1 approximately at $f = 8.0$ cycles/degree, and it is meaningless for frequencies

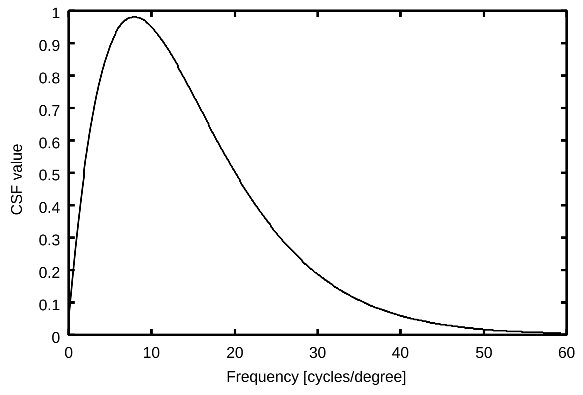


**Figure 1: Contrast sensitivity function**

above 60 cycles/degree. Figure 1 shows the contrast sensitivity function $A(f)$.

Since we use only image space, frequencies should be converted from cycles/degree to pixels/degree. We will use an assumption that a pixel pair corresponds to a cycle throughout this paper.

## 3. MAIN IDEA

The main idea of the visual image query algorithm is to create descriptors for target images, to save them, and then to create a descriptor for the query image, and compare it with saved descriptors. How are descriptors created?

The color depth of the image is reduced first. This is necessary to reduce errors caused by the inability to reproduce colors faithfully. After the color depth is reduced a limited number of rectangles of various sizes is placed in an image, the average color of each rectangle in CIE XYZ color space is computed, and finally converted to CIE LUV color coordinates. The series of LUV coordinates is part of the descriptor of an image.

Furthermore, a reduced histogram, containing only a small number of bins, is computed for the image, and saved in the descriptor as well. This histogram will be used to improve the query.

When the query image descriptor is compared with the target image descriptor, color differences are computed for each rectangle, using the CIE LUV color difference formula. These differences will be weighted according to the rectangle size using the contrast sensitivity function. In this way the differences that are more visible to us will be weighted stronger, and they will contribute more to the final distance. CIE LUV space was chosen as it is perceptually more uniform than CIE XYZ. For details about color spaces and conversions see[18, 9]. If there is some noise in the image, it will automatically be neglected by using the contrast sensitivity function, unless it is visible and influences our vision significantly. Actually, more visible differences will contribute to the error more significantly.

As the number of all possible rectangles of various sizes in an image is very large we use only a subset of all rectangles. The positions and orientations of the rectangles are chosen quasi randomly, which makes the whole process deterministic.

### 3.1 Algorithm Details

#### 3.1.1 *Descriptor Generation*

The first step in descriptor generation is the reduction of image size and color depth. Images are reduced to 128x128 pixel images.

In this way we do not need as many rectangles to cover the image as if it would have original size, and all images will be of the same size which assures that a particular rectangle represents the same area in all images. Of course, fine details are lost, but they are anyhow not of large interest in a query system. The next step is color reduction. The colors are reduced to a relatively small set of predefined colors. This set represents the whole color gamut, and consists of colors which we would remember more easily, or of colors we would be using in describing an image to someone. Actually, drawing the sketch is nothing else than describing the image, like saying there was a light red car in the front of medium yellow house.

In order to divide the color gamut in such a way, a color space that is perceptually uniform (or close to it), and intuitive is needed. The Coloroid color space [12, 13, 14], possesses both of these characteristics. It is perceptually uniform (note that there is no totally uniform color space, but there are only better or worse approximations of it), and has the advantage that a color can be described using three intuitive parameters (lightness, saturation and hue). It can be easily converted to CIE XYZ coordinates. We have chosen 8 basic hues: yellow, orange, red, magenta, violet, blue, cyan - cold green, and warm green. There are five lightness-saturation steps per hue. Beside these basic colors there are five neutral (gray) colors as well. A segmentation of the whole color gamut allows to classify every color as belonging to one of the selected representatives.

After having defined the reduced colors and having reduced the image size, the next step is the reduction of the color depth. This will be done using a slightly different approach than simply finding the closest color. The first step in selecting corresponding color to the original color is estimating its hue. Once, the hue is estimated (there are 9 possibilities, 8 colors + 1 gray), the nearest color from the available hue subset is chosen using the CIE LUV color difference formula. Finally the corresponding colors are converted to CIE XYZ coordinates in order to compute the average colors in the next step using a linear color space. Figure 2 shows an example of an original and a reduced color image.

The next step in descriptor computation is placing the rectangles in the image. A large number of rectangles will be needed, and the average color of each rectangle should be computed. In order to compute the average colors of rectangles fast summed area tables[2] are used. A separate table is built for each of the three CIE XYZ color components. That makes three tables, each containing a number of entries equal to the total number of pixels in the image. Element $T(i, j)$ of the table $T$ (see fig. 3) contains the sum of values of all pixels $X(x, y)$ such that $x \leq i$ and $y \leq j$. The average of the rectangle defined with points $(k, l)$ and $(i, j)$ such that $k \leq i$ and $l \leq j$ is then:

$$A = \frac{T(i,j) - T(k-1,j) - T(i,l-1) + T(k-1,l-1)}{(i-k+1)\cdot(j-l+1)} \tag{2}$$

Now, only the position and size of the rectangle must be given to compute the average with only a few operations.

The weighting of the particular rectangles according to the contrast sensitivity function is done implicitly using importance sampling for chosing the size of the rectangle. The integral of the contrast sensitivity function $g(f) = \int_{0^+}^{f} A(x) \cdot dx$ is precomputed and normalized by dividing all values with $g(60)$ (see fig. 4).

Randomly sampling over the domain of the inverse of $g(f)$ produces a frequency distribution as desired, so that no additional weights are necessary to get a distribution proportional to $A(f)$ (see bottom line in fig. 4).

The idea is to select the position on the y axis quasi-randomly, doing so, the metric has some meaning during the computation process as well. Once the size of a rectangle is determined, the orientation, i.e. longer and shorter side size, should be determined. The size of the rectangle corresponds to the rectangle's diagonal, and the maximum allowed ratio of the longer to the shorter side of the rectangle is the golden section ratio ($\phi = 1.618034\ldots$)[15]. The orientation of the rectangle is determined by choosing the angle between the diagonal and the horizontal axis. This angle is in the range $[\beta_{min}, \beta_{max}]$ (see fig. 5 ) and a particular angle is chosen quasi randomly. When the size and the orientation of the rectangle are known, the position of the rectangle can be determined. The rectangle position in the image is also determined by quasi random determination of its lower left corner. The position domain for this procedure is reduced, such that the complete rectangle lies within the image.

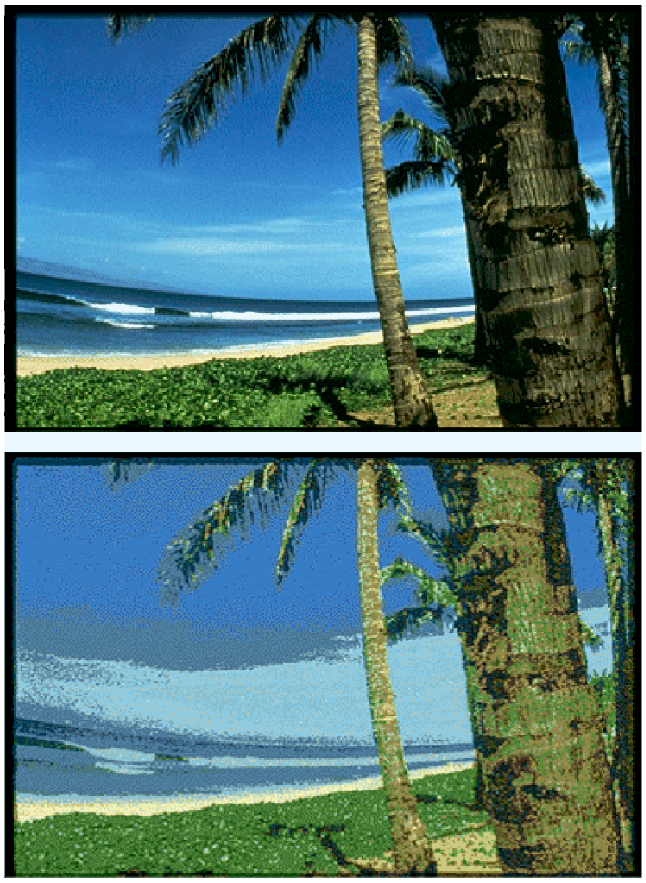


**Figure 2: Original and reduced color image**

The Halton sequence[8] is used to compute quasi-random numbers for this $4-$dimensional quasi random problem.

The Halton sequence for the N-dimensional point $x_m$ is defined as:

$$x_m = (\phi_2(m), \phi_3(m), ..., \phi_{p_{N-1}}(m), \phi_{p_N}(m)) \tag{3}$$

where $p_i$ refers to the $i$th prime number, and the function $\phi_r(m)$ is the radical-inverse function of $m$ to the base $r$. The value of the radical-inverse function $\phi_r(m)$ is obtained by simply reflecting the digits of $m$ written in base $r$ around the decimal point. Therefore if $m$ is:

$$m = a_0 \cdot r^0 + a_1 \cdot r^1 + ... + a_n \cdot r^n \tag{4}$$

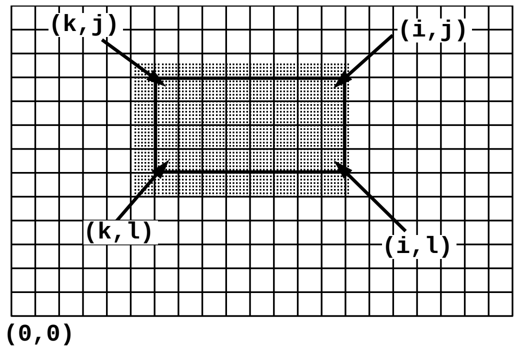


**Figure 3: Summed area table**

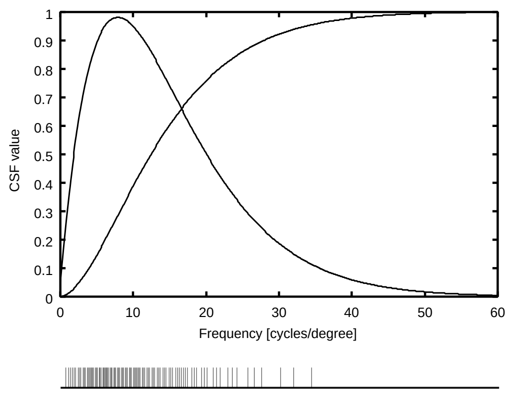


**Figure 4: Contrast sensitivity function and its integral**

the radical inverse function $\phi_r(m)$ is:

$$\phi_r(m) = a_0 \cdot r^{-1} + a_1 \cdot r^{-2} + ... + a_n \cdot r^{-(n+1)} \quad (5)$$

As our problem is $4-$dimensional we need the $\phi_2$, $\phi_3$, $\phi_5$ and $\phi_7$ functions. The size will be chosen using $\phi_2$, the orientation using $\phi_3$ and the position in the image using $\phi_5$ and $\phi_7$.

The rectangles are placed in the image, the average color is computed and 1000 (the number of rectangles) L,u,v triplets make the descriptor of an image.

As stated above, a reduced histogram is calculated as well, in order to improve the query. To make this control mechanism simple and fast, the histogram will consist only of the main hue bins, meaning there will be only one histogram bin for variations having the same hue. This makes 8 color bins, plus the gray bin. The reduced histogram consisting of nine values, is stored in the descriptor as well.

### *3.1.2 Image Query*

In a preprocessing phase the descriptors for all images of the database are generated and stored in a database. When starting the actual image query the descriptor database has to be loaded first and after that the user sketches or loads an image, and the descriptor for the sketch is computed. The 1000 rectangles of the query image descriptor are compared to the rectangles of every target descriptor, and the average difference is computed. Since descriptors represent CIE LUV colors, CIE LUV difference formulae will be used to compare two rectangles.

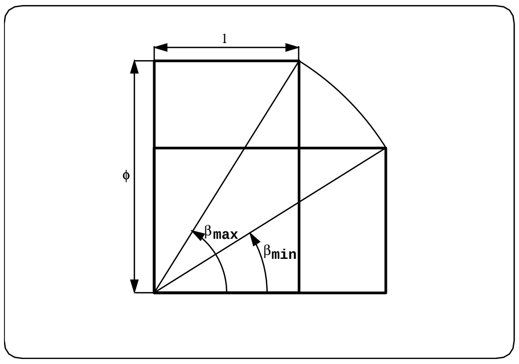


**Figure 5: Possible orientations of the rectangles**

The query can be additionally steered by using a so called histogram check. This check is an additional filter, which excludes images from the result list, which have an incompatible color histogram. It is assumed that certain basic features, like the existence of a bright color spot, are mandatory for an image to be a result candidate. In this way the set of potential result images is reduced largely. E.g., if the user has drawn an orange area on a blue background, all images containing blue but no orange can be considered as bad guesses, although the actual difference in the two metrics may be relatively small. In order to allow some tolerance the neighboring bins in the target histograms will also be taken into consideration. Meaning, when comparing a histogram bin from a query image with the corresponding target histogram bin, the neighboring bins in the target histogram are added to the original bin. By doing so, if the user draws an orange spot it will be checked if there are orange, yellow or red spots in the target image. For each histogram bin which is not zero in the user supplied image, corresponding bins in the database images histograms will be checked. If the database image bin has significantly less entries, the image is considered to be a bad guess. The threshold for relative bin difference can be set by the user, and our experience shows it is OK to use a threshold level of 20%, i.e. checking if the number of entries in the target image histogram bin is less than 20% of entries in the query histogram bin. In order to reduce the error when the user draws a red area, and there was e.g. an orange area in the image, the neighboring bins will be taken into account. When a red bin is compared, it is actually being compared with the weighted sum of orange, red and magenta bins in the target image. The user can set the weight of the neighboring bins between $0.0$ and $1.0$. Note that this check is done only in one way. It is checked if target images contain query colors, but it is not checked if the query image contains target images colors. The user will often remember only few most significant colors, and the image will contain some additional colors as well.

Let us make it clearer using an example. Lets assume that the user remembers an image of an orange fish in the sea. Of course the user is not able to reproduce the image exactly, so he/she draws a sketch containing a blue background and an orange “fish”. The query is started without the histogram check. Results are shown in figure 6. The user used a different blue color, and since there were some blue images in the database, closer blues are ranked better, than our “fish” image. If there were more blue images, “fish” could be ranked even worse. Now, the histogram check is included. Note, that a lot of blue images have no orange histogram component, and the user has used some orange, so all images containing no orange can be considered as not wanted. The result of the query, with

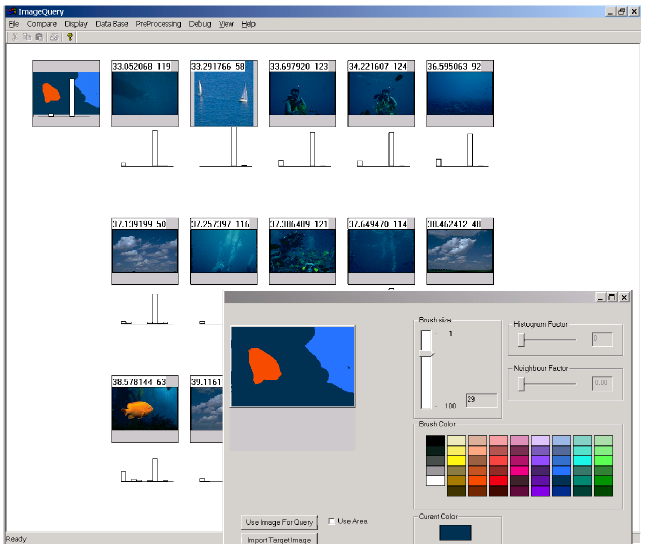


**Figure 6: Histogram check disabled**

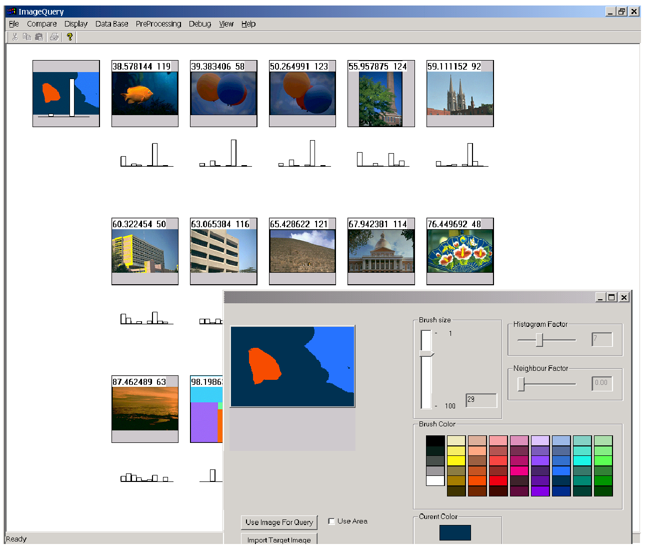


**Figure 7: Histogram check enabled**

histogram check is shown in figure 7. The "fish" image is a clear winner now.

An additional feature is implemented, too. If the user remembers only that there was a specific color layout in a part of the image, the user can limit the search to any rectangular area. Since the whole algorithm works in the original space, the implementation is straightforward.

## 4. ALGORITHM SUMMARY

Let us summarize the algorithm now. The integration of the Contrast Sensitivity Function can be precomputed and stored in an array. The descriptor database should be computed as next. There are lots of images, and a descriptor musts be computed for each of them. The image is resized to a predefined size first (e.g. 128 x 128). Color depth is reduced, and the r, g, b values are transformed to CIE XYZ values. Summed area tables are built for X, Y, and Z color components. The Halton series can be precomputed or easily computed on the fly. The size of a rectangle is determined, its orientation and position in the image. Average X, Y, and Z of the rectangles are computed using (2) and colors are converted to CIE LUV color space. These CIE LUV values represent the first part of a descriptor. The reduced histogram can be computed during color conversion, and it represents the second part of the descriptors. When all descriptors are computed, the descriptor database can be saved. It is always possible to add new images to the database.

The actual query starts with the database load. Next, the user supplies a query image to the system either by drawing a sketch, or selecting an existing image. Now the descriptor is computed for the supplied image, and the average differences of the query descriptor to all target descriptors are computed. Target images are sorted according to the difference, and the images are shown to the user. Additionally a histogram check is done to remove very unlikely results.

The whole process can be accelerated by checking the histograms first, and then computing the differences only if the histogram check did not fail. Another possibility to speed up the process is to limit the search only to the best, e.g. 100, matches. When the average difference of the last image in the top 100 group is known, any further query that exceeds this limit before all 1000 rectangles are checked, does not need to be processed further. The total difference can not be smaller, since there are no negative differences.

## 5. RESULTS

The query was implemented in C++ on a standard Windows2000 computer. Time needed to search through the database of 3549 images was app. 2.9 seconds on a PentiumIII 650MHz machine. The search was applied to the whole database always, without using the speedup possibilities described above.

After several tries the user understands how the query works, and results are just as expected. It can be used to find images with similar color layout as the proposed image, helping in finding the wanted image. The histogram check improves the query significantly, and it was almost never disabled in our experiments. Besides for finding an image which the user remembers the system can be used to find images with a desired color appearance, e.g. uniform or any other pattern.

Results of our query are shown in figures 8 to 12. In the first example we were trying to find a sunset image. With the histogram check enabled, the query was really successful. Figure 9 shows another example, where an existing image was used as query image in order to find similar images. Finally figure 10 shows the result of trying to find blue images. A search with limited area, where we are looking for images with yellow middle part is illustrated in figures 11 and 12.

All these figures are actual screen-shots from our tool. The 20 best ranked images are displayed, and the next screen can be displayed on demand, and so on. The user can create the descriptors database using this tool as well. Weighting factors for neighboring bins in histogram check, and histogram check threshold can be set as well. A simple dialog shown in figure 12 which can be used to produce quick sketches is a part of the tool, too.

## 6. CONCLUSION AND FUTURE WORK

We have presented an alternative visual image query system. The idea is related to that proposed by Jacobs et al.[10], but it differs in the metric used, and in two additional help mechanisms: color

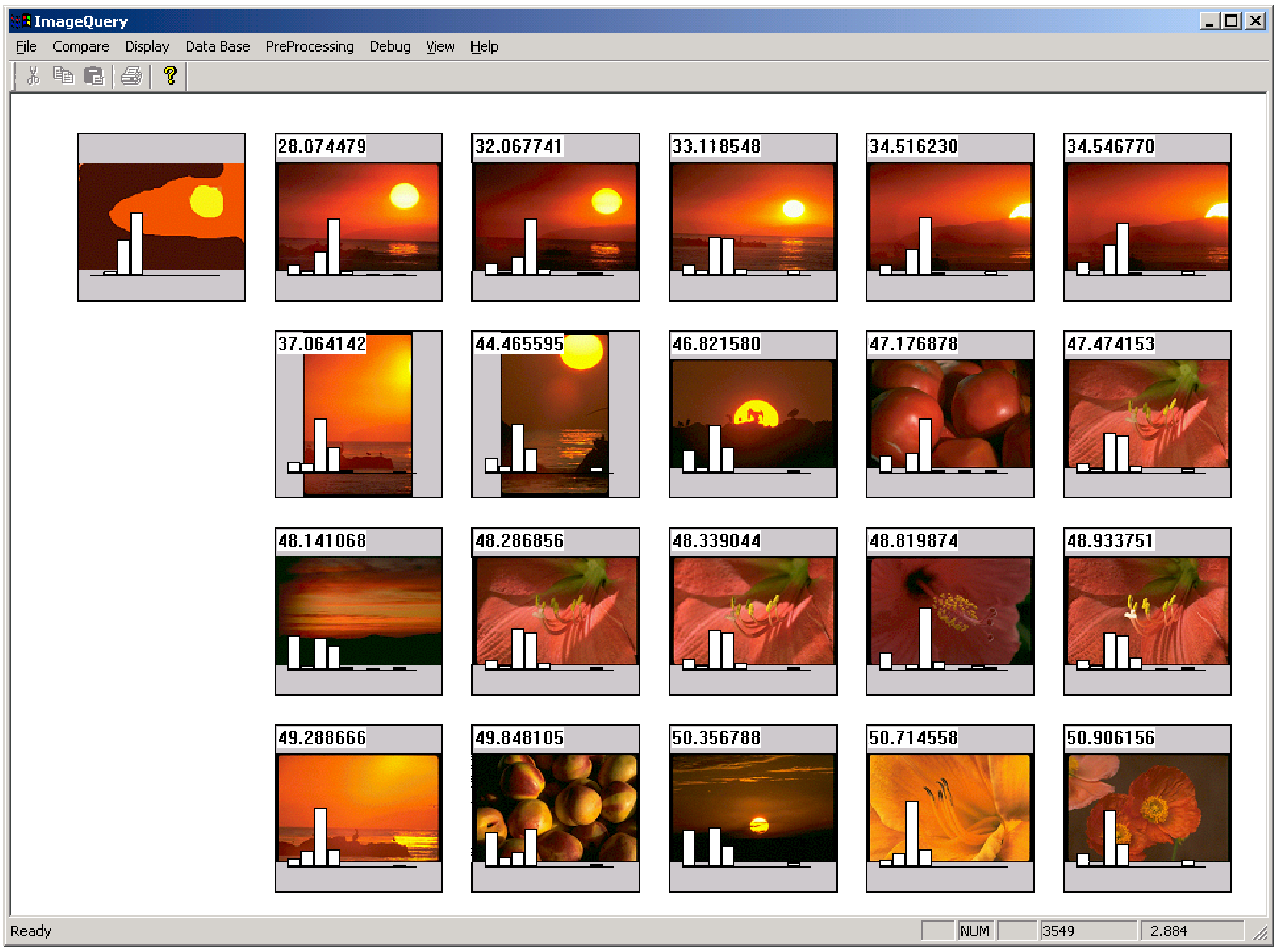


**Figure 8: Searching for a sunset**

reduction and the histogram check. The metric used is similar to the metric proposed by Neumann et al.[15]. This type of image query is not able to find images containing a specific pattern, e.g. black and white stripes, or containing a particular sub image e.g. a company logo, at an unknown position in the image.

The method is appropriate to find an image similar to the user drawn sketch, or any other image supplied to the system. The histogram check improves the query significantly.

The whole process is done in the original image space (there is no need to transform images using Fourier, wavelet or any other transform), which makes the whole method intuitive, and easy to understand.

We do not know if the use of another color system for color reduction would improve the query, or the color system chosen does not influence the query significantly. Furthermore, it is possible that some more advanced histogram check algorithm would improve the query further, but it would make it certainly less intuitive and harder to understand.

## 7. ACKNOWLEDGEMENTS

The Authors would like to thank Ina Wagner for her helpful comments and support. This work was partly sponsored by the Desarte - Esprit LTR Project NO 31870. The authors thank the Imagination company from Vienna, as well. Parts of this work were carried out in the scope of applied research at the VRVis Research Center in Vienna (`http://www.VRVis.at/`), Austria, which is funded by an Austrian governmental research program called K plus.

## 8. REFERENCES

[1] Gupta A. The virage image search engine: an open framework for image management. In *Storage and Retrieval for Image and Video Databases IV*, volume 2670 of *SPIE proceedings series*, pages 76–87, 1996.

[2] F. C. Crow. Summed-Area Table for Texture Mapping. *Computer Graphics (Proceedings of Siggraph '84)*, 18(3):207–212, July 1984.

[3] S. Daly. The visible difference predictor: An algorithm for the assessment of image fidelity. In A. B. Watson, editor, *Digital Images and Human Vision*, pages 179–206. MIT Press, 1993.

[4] J. P. Eakins and M. E. Graham. Content-based image retrieval, a report to the jisc technology applications programme, www.unn.ac.uk/iidr/research/cbir/report.html, 1999.

[5] M. Flickner, H. Sawhney, W. Niblack, J. Sahley, Q. Hiang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steel, and P. Yanker. Query by image and video content: the qbic system. *IEEE Computer*, 28(9):23–32, 1995.

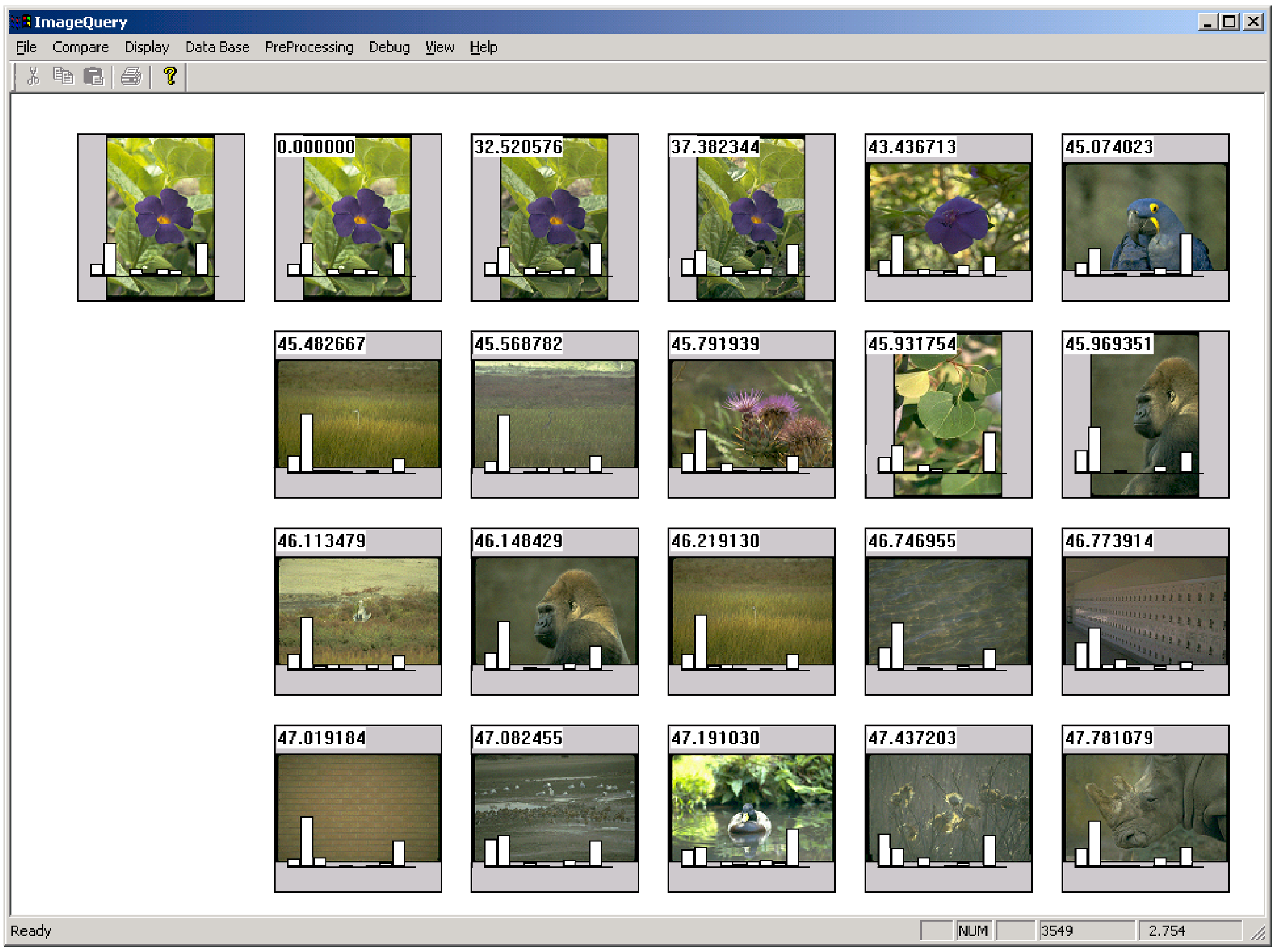


**Figure 9: Existing image as query image**

[6] Ajeetkumar Gaddipatti, Raghu Machiraju, and Roni Yagel. Steering image generation with wavelet based perceptual metric. *Computer Graphics Forum(Proceedings of Eurographics '97)*, 16(3):241–251, September 1997.

[7] B. Girod. What's wrong with mean-squared error? In A. B. Watson, editor, *Digital Images and Human Vision*, pages 207–220. MIT Press, 1993.

[8] Andrew S. Glasner. *Principles of Digital Image Synthesis*. Morgan Kaufmann, 1995.

[9] R. W. G. Hunt. *Measuring Color*. Ellis Horwood, 2nd edition, 1992.

[10] Charles E. Jacobs, Adam Finkelstein, and David H. Salesin. Fast multiresolution image querying. *Computer Graphics (Proceedings of Siggraph '95)*, 29(Annual Conference Series):277–286, November 1995.

[11] J. L. Mannos and D. J. Sakrison. The effects of a visual fidelity criterion on the encoding of images. *IEEE Transactions on Information Theory*, 20(4):525–535, 1974.

[12] A. Nemcsics. Coloroid color system. *Color Research and Application*, 5:113–120, 1980.

[13] A. Nemcsics. Color space of the coloroid color system. *Color Research and Application*, 12:135–146, 1987.

[14] A. Nemcsics. Colour dynamics, 1993.

[15] László Neumann, Krešimir Matkovć, and Werner Purgathofer. Perception based color image difference. *Computer Graphics Forum(Proceedings of Eurographics '98)*, 17(3):233–241, September 1998.

[16] A. Pentland, R. Picard, and S. Sclaroff. Photobook: tools for content-based manipulation of image databases. *International Journal of Computer Vision*, 18(3):233–254, 1996.

[17] H. Rushmeier, G. Ward, C. Piatko, P. Sanders, and B. Rust. Comparing real and synthetic images: Some ideas about metrics. In P. M. Hanrahan and W. Purgathofer, editors, *Rendering Techniques '95 (Proceedings of the Eurographics Workshop)*, pages 82–91. Springer Verlag, Vienna, 1995.

[18] G. Wyszecki and W. S. Stiles. *Color Science, Concept and Methods, Quantitive Data and Formulae*. John Wiley and Sons, 2nd edition, 1992.

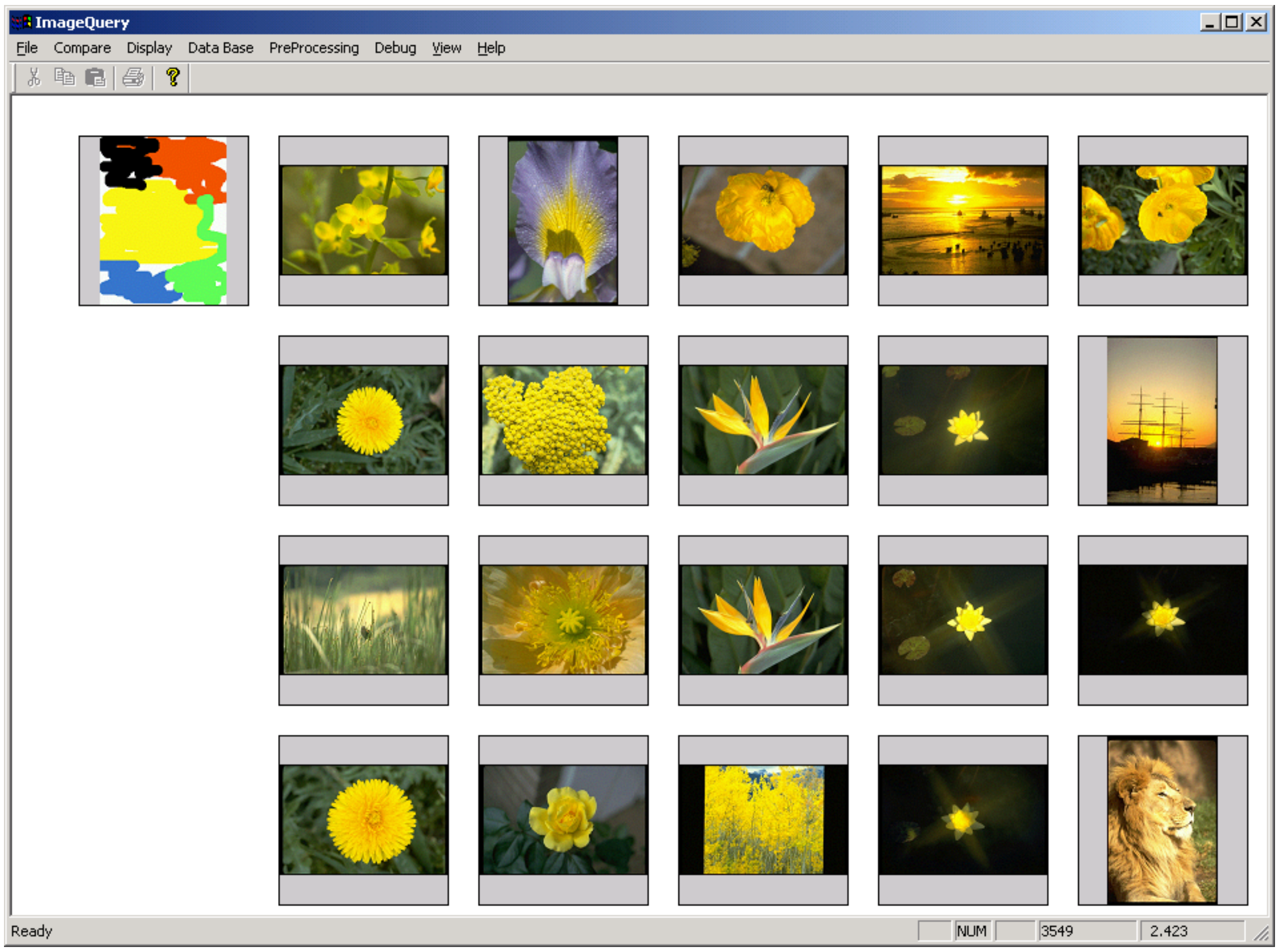

ImageQuery
File Compare Display Data Base PreProcessing Debug View Help
Ready
NUM
3549
2.423

**Figure 11: Limited area query**

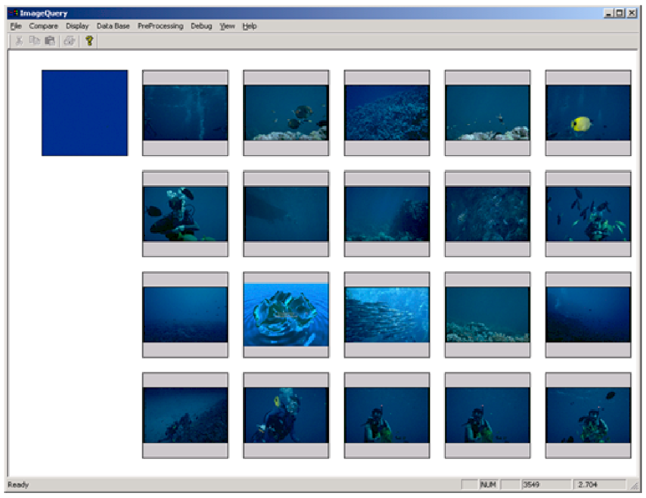
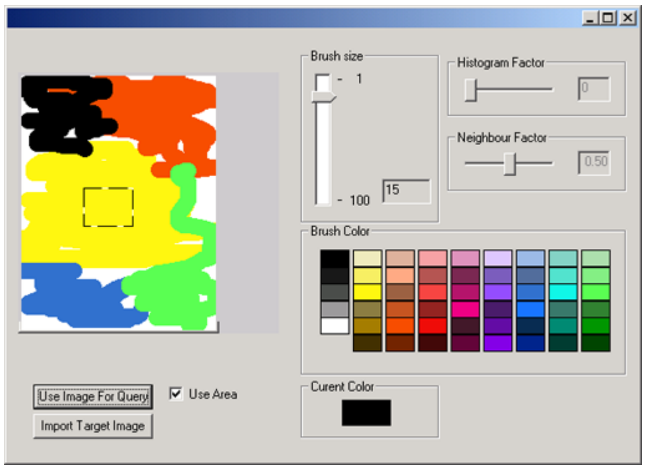

ImageQuery
File Compare Display Data Base PreProcessing Debug View Help
Ready
NUM
3549
2.704

**Figure 10: Blue images 1 to 20**



Brush size 1 - 100: 15

Histogram Factor: 0

Neighbour Factor: 0.50

[✓ Use Area]

Curent Color

**Figure 12: Sketch dialog with area enabled**