Fast Multiresolution Image Querying

Charles E. Jacobs   Adam Finkelstein   David H. Salesin

Department of Computer Science and Engineering
University of Washington
Box 352350
Seattle, WA 98195-2350

Technical Report 95-01-06
24 January 1995

Revised 20 May 1995

Proceedings of SIGGRAPH 95 (Los Angeles, California, August 6-11, 1995).
In Computer Graphics Proceedings, Annual Conference Series, 1995,
ACM SIGGRAPH, New York, 1995.

Fast Multiresolution Image Querying

Charles E. Jacobs *Adam Finkelstein* *David H. Salesin*

Department of Computer Science and Engineering
University of Washington
Seattle, Washington 98195

Abstract

We present a method for searching in an image database using a query image that is similar to the intended target. The query image may be a hand-drawn sketch or a (potentially low-quality) scan of the image to be retrieved. Our searching algorithm makes use of multiresolution wavelet decompositions of the query and database images. The coefficients of these decompositions are distilled into small "signatures" for each image. We introduce an "image querying metric" that operates on these signatures. This metric essentially compares how many significant wavelet coefficients the query has in common with potential targets. The metric includes parameters that can be tuned, using a statistical analysis, to accommodate the kinds of image distortions found in different types of image queries. The resulting algorithm is simple, requires very little storage overhead for the database of signatures, and is fast enough to be performed on a database of 20,000 images at interactive rates (on standard desktop machines) as a query is sketched. Our experiments with hundreds of queries in databases of 1000 and 20,000 images show dramatic improvement, in both speed and success rate, over using a conventional L^1, L^2, or color histogram norm.

CR Categories and Subject Descriptors: I.4.0 [Image Processing]: General — Image processing software; I.3.6 [Computer Graphics]: Methodology and Techniques — Interaction Techniques.

Additional Key Words: content-based retrieval, image databases, image indexing, image metrics, query by content, query by example, similarity retrieval, sketch retrieval, wavelets.

1 Introduction

With the explosion of desktop publishing, the ubiquity of color scanners and digital media, and the advent of the World Wide Web, people now have easy access to tens of thousands of digital images. This trend is likely to continue, providing more and more people with access to increasingly large image databases.

As the size of these databases grows, traditional methods of interaction break down. For example, while it is relatively easy for a person to quickly look over a few hundred "thumbnail" images to find a specific image query, it is much harder to locate that query among several thousand. Exhaustive search quickly breaks down as an effective strategy when the database becomes sufficiently large.

One commonly-employed searching strategy is to index the image database with keywords. However, such an approach is also fraught with difficulties. First, it requires a person to manually tag all the images with keys, a time-consuming task. Second, as Niblack *et al.* point out [20], this keyword approach has the problem that some visual aspects are inherently difficult to describe, while others are equally well described in many different ways. In addition, it may be difficult for the user to guess which visual aspects have been indexed.

In this paper, we explore an alternative strategy for searching an image database, in which the query is expressed either as a low-resolution image from a scanner or video camera, or as a rough sketch of the image painted by the user. This basic approach to image querying has been referred to in a variety of ways, including "query by content" [1, 4, 20], "query by example" [10, 13, 14], "similarity retrieval" [6, 16, 17, 21, 35] and "sketch retrieval" [14]. Note that this type of content-based querying can also be applied in conjunction with keyword-based querying or any other existing approach.

Several factors make this problem difficult to solve. The "query" image is typically very different from the "target" image, so the retrieval method must allow for some distortions. If the query is scanned, it may suffer artifacts such as color shift, poor resolution, dithering effects, and misregistration. If the query is painted, it is limited by perceptual error in both shape and color, as well as by the artistic prowess and patience of the user. For these reasons, straightforward approaches such as L^1 or L^2 image metrics are not very effective in discriminating the target image from the rest of the database. In order to match such imperfect queries more effectively, a kind of "image querying metric" must be developed that accommodates these distortions and yet distinguishes the target image from the rest of the database. In addition, the retrieval should ideally be fast enough to handle databases with tens of thousands of images at interactive rates.

In this paper, we describe how a Haar wavelet decomposition of the query and database images can be used to match a content-based query both quickly and effectively. The input to our retrieval method is a sketched or scanned image, intended to be an approximation to the image being retrieved. Since the input is only approximate, the approach we have taken is to present the user with a small set of the most promising target images as output, rather than with a single "correct" match. We have found that 20 images (the number of slides on a slide sheet) are about the most that can be scanned quickly and reliably by a user in search of the target.

In order to perform this ranking, we define an *image querying metric* that makes use of truncated, quantized versions of the wavelet decompositions, which we call *signatures*. The signatures contain only the most significant information about each image. The image querying metric essentially compares how many significant wavelet coefficients the query has in common with potential targets. We show how the metric can be tuned, using statistical techniques, to discriminate most effectively for different types of content-based image querying, such as scanned or hand-painted images. We also present a novel database organization for computing this metric extremely fast. (Our system processes a 128×128 image query on a database of 20,000 images in under 1/2 second on an SGI Indy R4400; by contrast, searching the same database using an L^1 metric takes over 14 minutes.) Finally, we evaluate the results of applying our tuned image querying metric on hundreds of queries in databases of 1000 and 20,000 images.

The content-based querying method we describe has applications in many different domains, including graphic design, architecture [30], TV production [27], multimedia [29], ubiquitous computing [36], art history [13], geology [26], satellite image databases [16], and medical imaging [15]. For example, a graphic designer may want to find an image that is stored on her own system using a painted query. She may also want to find out if a supplier of ultra-high-resolution digital images has a particular image in its database, using a low-resolution scanned query. In the realm of ubiquitous computing, a computer may need to find a given document in its database, given a video image of a page of that document, scanned in from the real-world environment. In all of these applications, improving the technology for content-based querying is an important and acknowledged challenge [8].

1.1 Related work

Previous approaches to content-based image querying have applied such properties as color histograms [33], texture analysis [12], and shape features like circularity and major-axis orientation of regions in the image [7], as well as combinations of these techniques.

One of the most notable systems for querying by image content, called "QBIC," was developed at IBM [20] and is now available commercially. The emphasis in QBIC is in allowing a user to compose a query based on a variety of different visual attributes; for example, the user might specify a particular color composition (x% of color 1, y% of color 2, etc.), a particular texture, some shape features, and a rough sketch of dominant edges in the target image, along with relative weights for all of these attributes. The QBIC system also allows users to annotate database images by outlining key features to search on. By contrast, the emphasis in our work is in searching directly from a query image, without any further specifications from the user — either about the database images or about the particulars of the search itself.

The work of Hirata and Kato [10] is perhaps the most like our own in its style of user interaction. In their system, called "query by visual example" (QVE), edge extraction is performed on user queries. These edges are matched against those of the database images in a fairly complex process that allows for corresponding edges to be shifted or deformed with respect to each other.

It is difficult to directly compare our results with these previous methods, since running times are rarely reported, and since the number of tests reported and the size of the databases being searched have generally been quite small. From the little information that has been provided, it appears that the success rate of our method is at least as good as that of other systems that work from a simple user sketch.

To our knowledge, we are the first to use a multiresolution approach for solving this problem. Among other advantages, our approach allows queries to be specified at any resolution (potentially different from that of the target); moreover, the running time and storage of our method are independent of the resolutions of the database images. In addition, the signature information required by our algorithm can be extracted from a wavelet-compressed version of the image directly, allowing the signature database to be created conveniently from a set of compressed images. Finally, our algorithm is much simpler to implement and to use than most previous approaches.

1.2 Overview of paper

In the next section, we discuss our approach to image querying in more detail and define an "image querying metric" that can be used for searching with imprecise queries. Section 3 describes the image querying algorithm in detail; the algorithm is simple enough that almost all of the code is included here. Section 4 describes the application we have built on top of this algorithm, and gives some examples of its use. Section 5 describes the results of our tests, and Section 6 outlines some areas for future research. Finally, the appendix discusses the statistical technique, "logit," that we used to tune the weights of our metric.

2 Developing a metric for image querying

Consider the problem of computing the distance between a query image Q and a potential target image T. The most obvious metrics to consider are the L^1 or L^2 norms:

$$||Q,T||_1 = \sum_{i,j} |Q[i,j] - T[i,j]| \tag{1}$$

$$||Q,T||_2 = \left(\sum_{i,j} (Q[i,j] - T[i,j])^2 \right)^{1/2} \tag{2}$$

However, these metrics are not only expensive to compute, but they are also fairly ineffective when it comes to matching an inexact query image in a large database of potential targets. For example, in our experience with scanned queries (described in Section 5), the L^1 and L^2 error metrics rank their intended target image in the highest 1% of the database only 3% of the time. (This rank is computed by sorting the database according to its L^1 or L^2 distance from the query, and evaluating the intended target's position in the sorted list.)

On the other hand, the target of the query image is almost always readily discernible to the human eye, despite such potential artifacts as color shifts, misregistration, dithering effects, and distortion (which, taken together, account for the relatively poor performance of the L^1 and L^2 metrics). The solution, it would seem, is to try to find an image metric that is "tuned" for the kind of errors present in image querying; that is, we would like a metric that counts primarily those types of differences that a human would use for discriminating images, but that gives much less weight to the types of errors that a human would ignore for this task. This problem is related to that of finding a good perceptual error metric for images, although, to our knowledge, most previous work in this area has been devoted primarily to minimizing image artifacts, for example, in image compression [11, 24, 34].

Since there is no obvious "correct" metric to use for image querying, we are faced with the problem of constructing one from scratch, using (informed) trial and error. The rest of this section describes the issues we addressed in developing our image querying metric.

2.1 A multiresolution approach

Our goal was to construct an image metric that is fast to compute, that requires little storage for each database image, and that improves significantly upon the L^1 or L^2 metrics in discriminating the targets of inexact queries. For several reasons, we hypothesized that a two-dimensional wavelet decomposition of the images [31, 32] would provide a good foundation on which to build such a metric:

- Wavelet decompositions allow for very good image approximation with just a few coefficients. This property has been exploited for lossy image compression [3]. Typically, in these schemes, just the wavelet coefficients with the largest magnitude are used.

- Wavelet decompositions can be used to extract and encode edge information [19]. Edges are likely to be among the key features of a user-painted query.

- The coefficients of a wavelet decomposition provide information that is independent of the original image resolution. Thus, a wavelet-based scheme allows the resolutions of the query and the target to be effectively decoupled.

- Wavelet decompositions are fast and easy to compute, requiring linear time in the size of the image and very little code.

2.2 Components of the metric

Given that we wish to use a wavelet approach, there are a number of issues that still need to be addressed:

1. **Color space.** We need to choose a color space in which to represent the images and perform the decomposition. (The same issue arises for L^1 and L^2 image metrics.) We decided to try a number of different color spaces: RGB, HSV, and YIQ. Ultimately, YIQ turned out to be the most effective of the three for our data, as reported in Figure 4 of Section 5.

2. **Wavelet type.** We chose Haar wavelets, both because they are fastest to compute and simplest to implement. In addition, user-painted queries (at least with our simple interface) tend to have large constant-colored regions, which are well represented by this basis. One drawback of the Haar basis for lossy compression is that it tends to produce blocky image artifacts for high compression rates. In our application, however, the results of the decomposition are never viewed, so these artifacts are of no concern. We have not experimented with other wavelet bases; others may work as well as or better than Haar (although will undoubtedly be slower).

3. **Decomposition type.** We need to choose either a "standard" or "non-standard" type of two-dimensional wavelet decomposition [2, 31]. In the Haar basis the non-standard basis functions are square, whereas the standard basis functions are rectangular. We would therefore expect the non-standard basis to be better at identifying features that are about as wide as they are high, and the standard basis to work best for images containing lines and other rectangular features. As reported in Figure 4 of Section 5, we tried both types of decomposition with all three color spaces, and found that the standard basis works best on our data, for both scanned and painted queries.

4. **Truncation.** For a 128×128 image, there are $128^2 = 16,384$ different wavelet coefficients for each color channel. Rather than using all of these coefficients in the metric, it is preferable to "truncate" the sequence, keeping only the coefficients with largest magnitude. This truncation both accelerates the search for a query and reduces storage for the database. Surprisingly, truncating the coefficients also appears to *improve* the discriminatory power of the metric, probably because it allows the metric to consider only the most significant features — which are the ones most likely to match a user's painted query — and to ignore any mismatches in the fine detail, which the user, most likely, would have been unable to accurately re-create. We experimented with different levels of truncation and found that storing the 60 largest-magnitude coefficients in each channel worked best for our painted queries, while 40 coefficients worked best for our scanned queries.

5. **Quantization.** Like truncation, the quantization of each wavelet coefficient can serve several purposes: speeding the search, reducing the storage, and actually improving the discriminatory power of the metric. The quantized coefficients retain little or no data about the precise magnitudes of major features in the images; however, the mere presence or absence of such features appears to have more discriminatory power for image querying than the features' precise magnitudes. We found that quantizing each significant coefficient to just two levels — $+1$, representing large positive coefficients; or -1, representing large negative coefficients — works remarkably well. This simple classification scheme also allows for a very fast comparison algorithm, as discussed in Section 3.

6. **Normalization.** The normalization of the wavelet basis functions is related to the magnitude of the computed wavelet coefficients: as the amplitude of each basis function increases, the size of that basis function's corresponding coefficient decreases accordingly. We chose a normalization factor that makes all wavelets orthonormal to each other. This normalization factor has the effect of emphasizing differences mostly at coarser scales. Because changing the normalization factor requires rebuilding the entire database of signatures, we have not experimented further with this degree of freedom.

2.3 The "image querying metric"

In order to write down the resulting metric, we must introduce some notation. First, let us now think of Q and T as representing just a single color channel of the wavelet decomposition of the query and target images. Let $Q[0,0]$ and $T[0,0]$ be the *scaling function coefficients* corresponding to the overall average intensity of that color channel. Further, let $\tilde{Q}[i,j]$ and $\tilde{T}[i,j]$ represent the $[i,j]$-th *truncated, quantized wavelet coefficients* of Q and T; these values are either -1, 0, or $+1$. For convenience, we will define $\tilde{Q}[0,0]$ and $\tilde{T}[0,0]$, which do not correspond to any wavelet coefficient, to be 0.

A suitable metric for image querying can then be written as

$$||Q,T|| \quad = \quad w_{0,0}\,|Q[0,0]-T[0,0]| + \sum_{i,j} w_{i,j}\left|\tilde{Q}[i,j]-\tilde{T}[i,j]\right|$$

We can simplify this metric in a number of ways.

First, we have found the metric to be just as effective if the difference between the wavelet coefficients $|\tilde{Q}[i,j]-\tilde{T}[i,j]|$ is replaced by $(\tilde{Q}[i,j] \neq \tilde{T}[i,j])$, where the expression $(a \neq b)$ is interpreted as 1 if $a \neq b$, and 0 otherwise. This expression will be faster to compute in our algorithm.

Second, we would like to group terms together into "buckets" so that only a small number of weights $w_{i,j}$ need to be determined experimentally. We group the terms according to the scale of the wavelet functions to which they correspond, using a simple bucketing function $bin(i,j)$, described in detail in Section 3.

Finally, in order to make the metric even faster to evaluate over many different target images, we only consider terms in which the *query* has a non-zero wavelet coefficient $\tilde{Q}[i,j]$. A potential benefit of this approach is that it allows for a query without much detail to match a very detailed target image quite closely; however, it does not allow a detailed query to match a target that does not contain that same detail. We felt that this asymmetry might better capture the form of most painted image queries. (Note that this last modification technically disqualifies our "metric" from being a metric at all, since metrics, by definition, are symmetric. Nevertheless, for lack of a better term, we will continue to use the word "metric" in the rest of this paper.)

Thus, the final "L^q" *image querying metric* $||Q,T||_q$ is given by

$$w_0\,|Q[0,0]-T[0,0]| \;+ \sum_{i,j\,:\,\tilde{Q}[i,j]\neq 0} w_{bin(i,\,j)}\left(\tilde{Q}[i,j]\neq \tilde{T}[i,j]\right) \quad (3)$$

The weights w_b in equation (3) provide a convenient mechanism for tuning the metric to different databases and styles of image querying. The actual weights we use are given in Section 3, while the method we use for their computation is described in the appendix.

2.4 Fast computation of the image querying metric

To actually compute the L^q metric over a database of images, it is generally quicker to count the number of *matching* $\tilde{Q}$ and $\tilde{T}$ coefficients, rather than *mismatching* coefficients, since we expect the vast majority of database images not to match the query image well at all. It is therefore convenient to rewrite the summation in (3) in terms of an "equality" operator $(a=b)$, which evaluates to 1 when $a=b$, and 0 otherwise. Using this operator, the summation

$$\sum_{i,j\,:\,\tilde{Q}[i,j]\neq 0} w_k\left(\tilde{Q}\neq\tilde{T}\right)$$

in equation (3) can be rewritten as

$$\sum_{i,j\,:\,\tilde{Q}[i,j]\neq 0} w_k \;-\; \sum_{i,j\,:\,\tilde{Q}[i,j]\neq 0} w_k\left(\tilde{Q}=\tilde{T}\right)$$

Since the first part of this expression $\sum w_k$ is independent of $\tilde{T}$, we can ignore it for the purposes of ranking the different target images in L^q. It therefore suffices to compute the expression

$$w_0\,|Q[0,0]-T[0,0]| \;- \sum_{i,j\,:\,\tilde{Q}[i,j]\neq 0} w_{bin(i,\,j)}\left(\tilde{Q}[i,j]=\tilde{T}[i,j]\right) \quad (4)$$

This expression is just a weighted sum of the difference in the average color between Q and T, and the number of stored wavelet coefficients of T whose indices and signs match those of Q.

3 The algorithm

The final algorithm is a straightforward embodiment of the L^q metric as given in equation (4), applied to the problem of finding a given query in a large database of images. The complexity of the algorithm is linear in the number of database images. The constant factor in front of this linear term is small, as discussed in Section 5.

At a high level, the algorithm can be described as follows: In a preprocessing step, we perform a standard two-dimensional Haar wavelet decomposition [2, 31] of every image in the database, and store just the overall average color and the indices and signs of the m largest-magnitude wavelet coefficients. The indices for all of the database images are then organized into a single data structure in the program that optimizes searching. Then, for each query image, we perform the same wavelet decomposition, and again throw away all but the average color and the largest m coefficients. The score for each target image T is then computed by evaluating expression (4).

The rest of the section describes this algorithm in more detail.

3.1 Preprocessing step

A standard two-dimensional Haar wavelet decomposition of an image is very simple to code. It involves a one-dimensional decomposition on each row of the image, followed by a one-dimensional decomposition on each column of the result.

The following pseudocode performs this one-dimensional decomposition on an array A of h elements, with h a power of two:

```
proc DecomposeArray(A : array[0..h−1] of color):
    A ← A/√h
    while h > 1 do:
        h ← h/2
        for i ← 0 to h−1 do:
            A′[i] ← (A[2i] + A[2i+1])/√2
            A′[h+i] ← (A[2i] − A[2i+1])/√2
        end for
        A ← A′
    end while
end proc
```

In the pseudocode above, the entries of A are assumed to be 3-dimensional color components, each in the range $[0,1]$. The various arithmetic operations are performed on the separate color components individually.

An entire $r\times r$ image T can thus be decomposed as follows:

```
proc DecomposeImage(T : array[0..r−1, 0..r−1] of color):
    for row ← 1 to r do:
        DecomposeArray(T[row, 0..r−1])
    end for
    for col ← 1 to r do:
        DecomposeArray(T[0..r−1, col])
    end for
end proc
```

(In practice, the *DecomposeImage* routine is best implemented by decomposing each row, then transposing the matrix, decomposing each row again, and transposing back.)

After the decomposition process, the entry $T[0,0]$ is proportional to the average color of the overall image, while the other entries of T contain the *wavelet coefficients*. (These coefficients are sufficient for reconstructing the original image T, although we will have no need to do so in this application.)

Finally, we store only $T[0,0]$ and the indices and signs of the largest m wavelet coefficients of T. To optimize the search process, the remaining m wavelet coefficients for *all* of the database images are organized into a set of six arrays, called the *search arrays*, with one array for every combination of sign (+ or −) and color channel (such as R, G, and B).

For example, let $\mathcal{D}^c_+$ denote the "positive" search array for the color channel c. Each element $\mathcal{D}^c_+[i,j]$ of this array contains a list of all images T having a large positive wavelet coefficient $T[i,j]$ in color channel c. Similarly, each element $\mathcal{D}^c_-[i,j]$ of the "negative" search array points to a list of images with large negative coefficients in c.

These six arrays are used to speed the search for a particular query, as described in the next section. In our implementation, the search arrays are created as a preprocess for a given database and stored on disk. We use a small stand-alone program to add new images to the database incrementally. This program performs the wavelet decomposition for each new image, finds the largest m coefficients, and augments the database search arrays accordingly.

3.2 Querying

The querying step is straightforward. For a given query image Q, we perform the same wavelet decomposition described in the previous section. Again, we keep just the overall average color and the indices and signs of the largest m coefficients in each color channel.

To compute a score, we loop through each color channel c. We first

Figure 1: The image querying application. The user paints a query in the large rectangular window, and the 20 highest-ranked targets appear in the small windows on the right. To avoid copyright infringements, the database for this example contains only 96 images (all created by artists who have been dead more than 75 years). Because the database is so limited, only the intended target (in the upper-left small window) appears to match the query very closely.

compute the differences between the query's average intensity in that channel $Q^c[0,0]$ and those of the database images. Next, for each of the m non-zero, truncated wavelet coefficients $\tilde{Q}^c[i,j]$, we search through the list corresponding to those database images containing the same large-magnitude coefficient and sign, and update each of those image's scores accordingly:

func *ScoreQuery*$(Q : \mathit{array}[0..r-1, 0..r-1]$ *of color*; $m : \mathit{int})$:
    DecomposeImage(Q)
    Initialize $\mathit{scores}[i] \leftarrow 0$ for all i
    for each color channel c **do**:
        for each database image T **do**:
            $\mathit{scores}[\mathit{index}(T)] \mathrel{+}= w^c[0] * |Q^c[0,0] - T^c[0,0]|$
        end for
        $\tilde{Q} \leftarrow \mathit{TruncateCoefficients}(Q, m)$
        for each non-zero coefficient $\tilde{Q}^c[i,j]$ **do**
            if $\tilde{Q}^c[i,j] > 0$ **then**
                $\mathit{list} \leftarrow \mathcal{D}^c_+[i,j]$
            else
                $\mathit{list} \leftarrow \mathcal{D}^c_-[i,j]$
            end if
            for each element ℓ of *list* **do**
                $\mathit{scores}[\mathit{index}(\ell)] \mathrel{-}= w^c[\mathit{bin}(i,j)]$
            end for
        end for
    end for
    return *scores*
end func

The function $\mathit{bin}(i,j)$ provides a way of grouping different coefficients into a small number of bins, with each bin weighted by some constant $w[b]$. For a given set of bins, the best weights $w[b]$ can be found experimentally, as discussed in the appendix. The larger the training set, the more weights that can be used. The size of our training set was sufficient for 18 weights: 6 per color channel.

In our implementation, we use the function

$$\mathit{bin}(i,j) \quad := \quad \min\{\max\{i,j\},5\}.$$

For our database of images, a good set of weights, using the YIQ color space and standard decomposition, was found to be:

	Painted			*Scanned*		
b	$w^Y[b]$	$w^I[b]$	$w^Q[b]$	$w^Y[b]$	$w^I[b]$	$w^Q[b]$
0	4.04	15.14	22.62	5.00	19.21	34.37
1	0.78	0.92	0.40	0.83	1.26	0.36
2	0.46	0.53	0.63	1.01	0.44	0.45
3	0.42	0.26	0.25	0.52	0.53	0.14
4	0.41	0.14	0.15	0.47	0.28	0.18
5	0.32	0.07	0.38	0.30	0.14	0.27

(All scaling function coefficients in our implementation are reals in the range $[0,1]$, so their differences tend to be smaller than the differences of the truncated, quantized wavelet coefficients. Thus, the weights on the scaling functions $w[0]$ have relatively large magnitudes because they generally multiply smaller quantities.)

As a final step, our algorithm examines the list of scores, which may be positive or negative. The smallest (typically, the most negative) scores are considered to be the closest matches. We use a "HeapSelect" algorithm [23] to find the 20 closest matches in linear time.

4 The application

We have built a simple interactive application that incorporates our image querying algorithm. The program is written in C^{++}, using OpenGL and Motif. It runs on SGI workstations.

A screen dump of the running application is shown in Figure 1. The user paints an image query in the large rectangular area on the left side of the application window. When the query is complete, the user presses the "Match" button. The system then tests the query against all the images in the database and displays the 20 top-ranked targets in the small windows on the right. (The highest-ranked target is displayed in the upper left, the second-highest target to its right, and so on, in row-major order.)

For convenience, the user may paint on a "canvas" of any aspect ratio. However, our application does not currently use this information in performing the match. Instead, the painted query is internally rescaled to a square aspect ratio and searched against a database in which all images have been similarly rescaled as a preprocess. We discuss how a user-specified aspect ratio might also be used to improve the match in Section 6.

Figure 2(a) shows an example of a painted query, along with the L^q rank of its intended target (c) in databases of 1093 and 20,558 images.

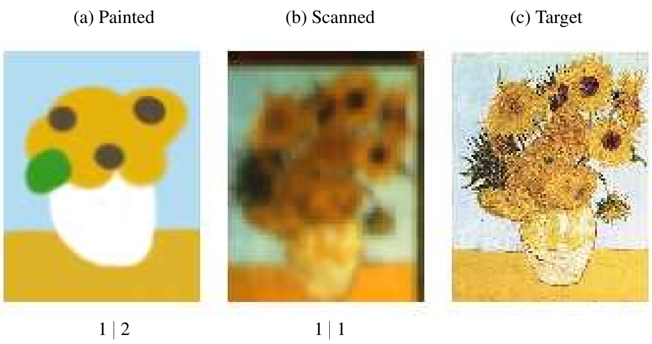

Figure 2: Queries and their target: (a) a query painted from memory; (b) a scanned query; and (c) their intended target. Below the queries, the L^q ranks of the intended target are shown for two databases of sizes $1093 \mid 20{,}558$.

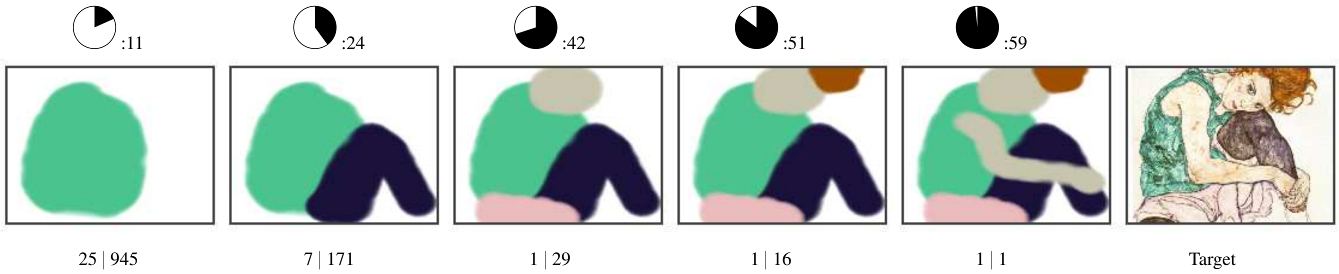

Figure 3: Progression of an interactive query. Above each partially-formed query is the actual time (in seconds) at which the snapshot was taken. Below each query are the L^q ranks of the intended target for databases of sizes $1093 \mid 20,558$.

Rather than painting a query, the user may also click on any of the displayed target images to serve as a subsequent query, or use any stored image file as a query. Figure 2(b) shows an example of using a low-quality scanned image as a query, again with its L^q rank in the two databases.

Because the retrieval time is so fast (under 1/2 second in a database of 20,000 images), we have also implemented an "interactive" mode, in which the 20 top-ranked target images are updated whenever the user pauses for a half-second or more. Figure 3 shows the progression of an interactive query, along with the actual time at which each snapshot was taken and the L^q rank of the intended target at that moment in the two different databases.

5 Results

To evaluate our image querying algorithm, we collected three types of query data.

The first set, called "scanned queries," were obtained by printing out small $1/2" \times 1/2"$ thumbnails of our database images, using a full-color Tektronix Phaser IISDX printer at 300dpi, and then scanning them back into the system using a Hewlett-Packard ScanJet IIc scanner. As a result of these steps, the scanned images became somewhat altered; in our case, the scanned images generally appeared fuzzier, darker, and slightly misregistered from the originals. An example of a scanned query is shown in Figure 2(b). We gathered 270 such queries, of which 100 were reserved for evaluating our metric, and the other 170 were used as a training set.

The second set, called "painted queries," were obtained by asking 20 subjects, most of whom were first-time users of the system, to paint complete image queries, in the non-interactive mode, while looking at thumbnail-sized versions of the images they were attempting to retrieve. We also gathered 270 of these queries and divided them into evaluation and training sets in the same fashion.

The third set, called "memory queries," were gathered in order to see how well this style of querying might work if users were not looking at small versions of the images they wanted to retrieve, but instead were attempting to retrieve images from memory. To obtain these queries, we asked each subject to initially examine two targets T_1 and T_2, and paint a query for T_1, which we threw away. The subject was then asked to iteratively examine a target T_{i+1} (starting with $i = 2$) and paint query T_i, which had not been viewed since before query T_{i-1} was painted. In this way, we hoped to get a more accurate idea of how well a user might do if attempting to retrieve a familiar image from memory, without being able to see the image directly. An example of a memory query is shown in Figure 2(a). We gathered 100 of these queries, which were used for evaluation only.

5.1 Training

Each training set was subdivided into 2 equal sets. The first training set of 85 queries was used to determine the weights of the image querying metric, as described in the appendix. The second training set of 85 queries was used to find the optimal color space, decomposition type, and number m of coefficients to store for each image. We performed an exhaustive search over all three dimensions, using color spaces RGB, HSV, and YIQ; standard and non-standard wavelet decompositions; and $m = 10, 20, 30, \ldots, 100$. For each combination, we found weights using the first set of images, and then tested these weights on the second set of images, using "the percentage of intended targets that were ranked among the top 1% in our database of 1093 images" as the evaluation function.

The results of these tests for scanned and painted queries are reported in Figure 4. For scanned queries, 40 coefficients with a standard decomposition and YIQ worked best. The same configuration, except with 60 coefficients, worked best for painted queries. This latter configuration was used for testing the success of memory queries as well.

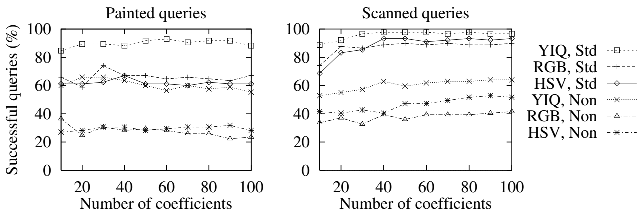

Figure 4: Choosing among color spaces (RGB, HSV, or YIQ), wavelet decomposition type (standard or non-standard), and number of coefficients.

5.2 Performance on actual queries

Using the weights obtained from the training set, we then evaluated the performance using the remaining 100 queries of each type. The graphs in Figure 5 compare our L^q metric to the L^1 and L^2 metrics and to a color histogram metric L^c, for a database of 1093 images. The three graphs show, from left to right: scanned queries (using 40 coefficients), painted queries (using 60 coefficients), and memory queries (using 60 coefficients).

The L^1 and L^2 metrics in these graphs were computed on both the full-resolution images (128×128 pixels) and on averaged-down versions (8×8 pixels), which have roughly the same amount of data as the 60-coefficient L^q metric. The color histogram metric L^c was computed by quantizing the pixel colors into a set of $6 \times 6 \times 6$ bins in RGB space, and then computing an L^1 metric over the number of pixels falling in each bin for the query versus the target.

Results for all six methods are reported by giving the percentage of queries y that were ranked among the top x% of the database images,

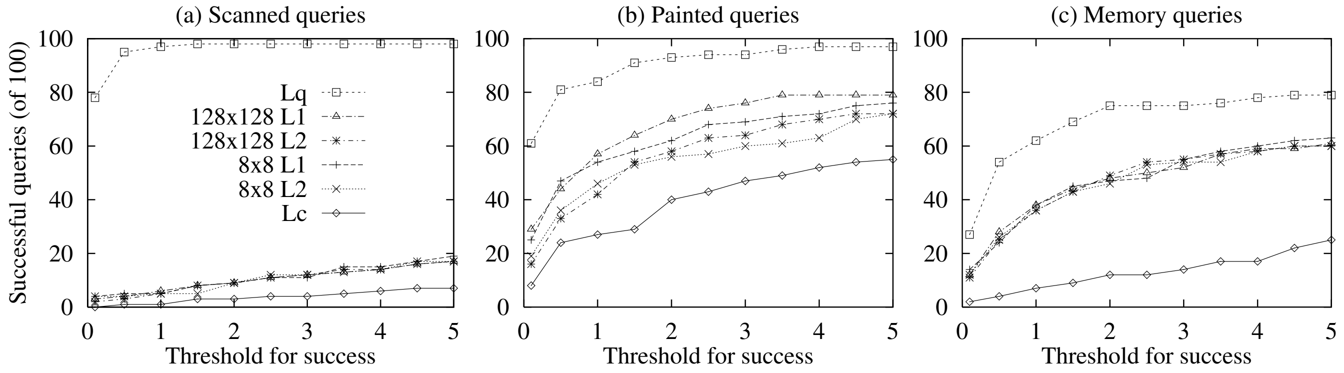

Figure 5: Comparison of L^q metric against L^1, L^2, and a color histogram metric L^c. The percentage of queries y ranked in the top x% of the database are plotted on the x and y axes.

with x and y plotted on the x- and y-axes. For example, the leftmost data point of each curve, at $x = 1/1093 \approx 0.09\%$, reports the percentage of queries whose intended targets were ranked in first place for each of the six methods; the data points at $x = 1\%$ report the percentage of queries whose intended targets were ranked among the top $\lfloor 0.01 * 1093 \rfloor = 10$ images; and so on.

Note that the scanned queries perform remarkably poorly under the L^1, L^2 and L^c metrics. These poor scores are probably due to the fact that the scanned queries were generally darker than their intended targets, and so matched many incorrect (darker) images in the database more closely.

5.3 Robustness with respect to distortions

In order to test more precisely how robust the different metrics are with respect to some of the distortions one might find in image querying, we devised the following suite of tests. In the first test, 100 randomly chosen color images from the database were scaled by a factor s ranging from 1 to 2 and used as a query. In the second test, the same images were rotated by a factor r between $0°$ and $45°$. In the third test, the same images were translated in a random direction by a distance t between 0 and 0.5 times the width of the query. In the fourth test, the colors of these images were uniformly shifted in normalized RGB space in a random direction by a distance c between 0 and 1. In the final test, all four of these transformations were applied for each test, in the order scale/rotate/translate/color-shift, with s, r, t, and c ranging as in the other tests. For all five tests, in cases where a border of the distorted image was undefined by the transformation (which occurs for rotations and translations), the image was padded with its overall average color. In cases where the color shift would lie outside the RGB cube, the color was clamped to $[0,1]^3$.

The top row of Figure 6 shows the results of these five tests. The curves in these graphs report the percentage of queries whose intended targets were ranked in the top 1% of the 1093-image database. Note that the L^q metric performs as well as or better than all other methods, except for L^c. However, as expected, the L^c metric does very poorly for color shifts, severely reducing this metric's utility in situations where a query's color is not always true. The bottom row shows the same five tests, but applied to each of our 100 scanned, painted, and memory queries — all with the L^q metric.

5.4 Effect of database size

We also wanted to test how well our method would perform as the size of the database was increased. We therefore gathered 19,465 images from the World Wide Web, using the WebCrawler [22] to find files on the Web with a ".gif" extension. We computed a signature and thumbnail for each image and stored the resulting database locally, along with a "URL" for each image — a pointer back to the original Web site. The resulting application is a kind of graphical "Web browser," in which a user can paint a query and very quickly see the images on the Web that match it most closely. Clicking on one of these thumbnail images calls up the full-resolution original from the Web.

In order to check how well our metric performed, we created a set of 20 nested databases, with each database containing our original 1093 images plus increasingly large subsets of the Web database. The largest such database had 20,558 images. For each of the three sets of 100 queries, we then evaluated how many of those queries would find their intended target in the top 1% of the different nested databases. We found that the number of queries matching their correct targets by this criterion remained almost perfectly constant in all three cases, with the number of correctly matching queries varying by at most 2% across the different database sizes.

5.5 Speed of evaluation

We measured the speed of our program by running 68 queries 100 times each, with databases ranging in size from $n = 1093$ to $n = 20,558$, and with the number of coefficients ranging from $m = 20$ to $m = 1000$. A regression analysis indicates that the running time is linear in both m and n, with each query requiring approximately $190 + 0.11m + 0.012n$ milliseconds to process on an SGI Indy R4400. This running time includes the time to decompose a 128×128-pixel query, score all n images in the database according to the L^q metric, and find the 20 top-ranked targets.

As two points of comparison, Table 1 reports the average running time of our algorithm to that of the other methods surveyed for finding a query using $m = 20$ coefficients per channel in databases of size $n = 1093$ and $n = 20,558$ images. In all cases, the times reported do not include any preprocessing that can be performed on the database images alone.

Metric	*Time* $n = 1093$	$n = 20,558$
L^q	0.19	0.44
L^1 (8×8)	0.66	7.46
L^2 (8×8)	0.68	6.39
L^1 (128×128)	47.46	892.60 (est.)
L^2 (128×128)	42.04	790.80 (est.)
L^c	0.47	5.03

Table 1: Average times (in seconds) to match a single query in databases of 1093 and 20,558 images under different metrics.

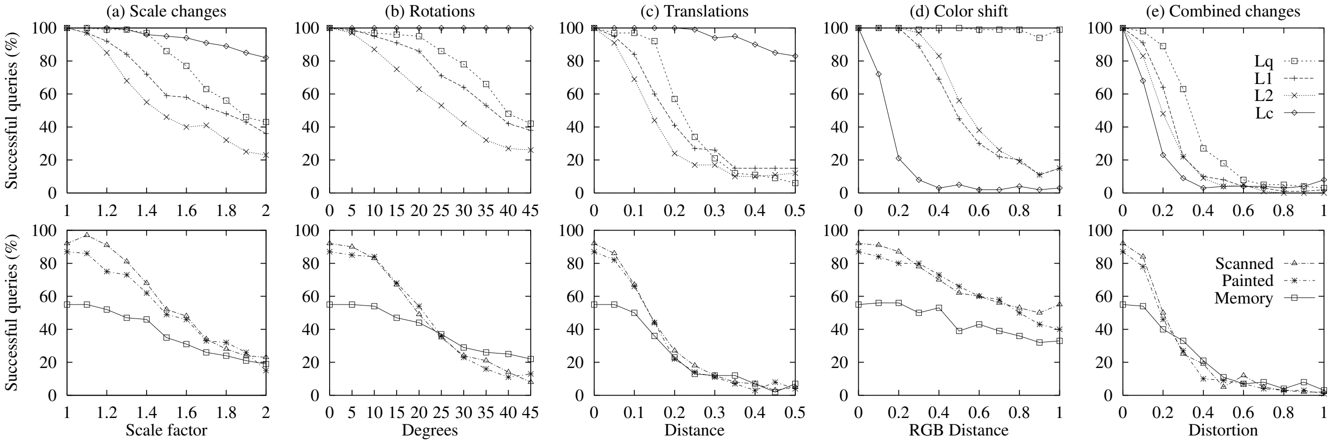

Figure 6: Robustness of various querying metrics with respect to different types of image distortions: (a) Scale changes; (b) Rotations; (c) Translations; (d) Color shifts; (e) All four effects combined. The top row (legend at upper right) compares the L^q metric to L^1, L^2 and L^c, using the target itself as the original undistorted query. The bottom row (legend at lower right) shows the same five tests applied to each of our 100 scanned, painted, and memory queries, using the L^q metric.

5.6 Interactive queries

To test the speed of interactive queries, we asked users to paint in the interactive mode, and we kept track of how long it took for the intended target to appear among the top 20 images in the database. For these tests, we used just $m = 20$ significant coefficients.

For the first such test, we had 5 users paint a total of 106 interactive queries, allowing them to look at thumbnails of the intended targets. The overall median time to retrieve the target images was 20 seconds.

Next, in order to see how this median query time might vary with database size, we asked 2 users to paint a total of 21 interactive queries in our database of 20,558 images. For each query, the application kept a log of each paint stroke and the time at which it was drawn. We then used these logs to simulate how quickly the same query would bring up the intended target among the top 20 images in databases of various sizes. The results are shown in Figure 7.

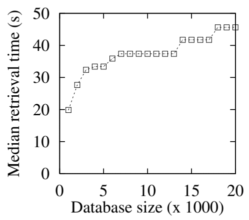

Figure 7: The effect of database size on median interactive query time.

To see if painting from memory would affect retrieval time, we selected 20 target images and, for each subject, we randomly divided these targets into two equal sets. Each subject was then asked to paint the 10 images from the first set while looking at a thumbnail of the image, and the 10 images from the second set from memory, in the style described for "memory queries" above. We used 3 subjects in this experiment. We found that the median query time increased from 18 seconds when the subjects were looking at the thumbnails, to 22 seconds when the queries were painted from memory.

In our experience with interactive querying, we have observed that users will typically be able to sketch all the information they know about an image in a minute or less, whether they are looking at a thumbnail or painting from memory. In most cases, the query succeeds within this short time. If the query fails to bring up the intended target within a minute or so, users will typically try adding some random details, which sometimes help in bringing up the image. If this tactic fails, users will simply give up and, in a real system, would presumably fall back on some other method of searching for the image. (In this case, we report an "infinite" query time.)

We have observed two benefits of painting queries interactively. First, the time to retrieve an image is generally reduced because the user simply paints until the target image appears, rather than painting until the query image seems finished. Second, the interactive mode subtly helps "train" the user to find images more efficiently, because the application is always providing feedback about the relative effectiveness of an unfinished query while it is being painted.

6 Discussion and future work

The algorithm we have described is extremely fast, requires only a small amount of data to be stored for each target image, and is remarkably effective. It is also fairly easy to understand and implement. Finally, it has parameters that can be tuned for a given database or type of query image.

Although this new image searching method has substantial advantages over previous approaches, its ultimate utility may depend to a large extent on the size of the image database being searched. Our tests suggest that, for the majority of non-interactive queries, our method will be able to pinpoint the correct target to within a 1%-sized subset of the overall database, regardless of the database's size. Thus, for a database of 100 images, it is easy to pull up the correct image precisely. However, for a database of 20,000 images, the user is still left with a list of 200 potential matches that must be searched visually, or by some other means. On the other hand, with interactive querying, even for a 20,000-image database it is still possible to place the target into the top 20 images the majority of the time. Nonetheless, creating a good query becomes increasingly difficult as the database grows. For a large enough database, even this interactive style of querying would begin to require more precision than most users can provide.

We have tried to perform a number of different tests to measure the success and robustness of our image querying metric. However, it is easy to envision many more tests that would be interesting to perform. One interesting test would be to try to quantify the degree to which different training sets affect our metric's sensitivity to various image distortions. For example, in querying images from memory, colors are less likely to be accurate. Presumably, a training set of "memory queries" would therefore reduce the metric's sensitivity to color accuracy. How significant is this effect? In addition, it would be interesting to examine whether providing separate training sets for individual users or for particular databases would make a significant difference in the metric's discriminatory power.

Our method also has some limitations, which we hope to address in future work. For example, while it is fairly robust with respect to a large degree of distortion in the query image, our metric does not currently allow for general pattern matching of a small query, such as an icon or company logo, inside some larger database image.

Here are some other areas for future research:

Aspect ratio. Currently, we allow users to choose an aspect ratio for their query; however, this aspect ratio is not used in the search itself. It would be straightforward to add an extra term to our image querying metric for the similarity of aspect ratio. The weight for this term could be found experimentally at the same time as the other weights are computed.

Perceptually-based spaces. It would be interesting to try using a perceptually uniform color space, such as CIE LUV or TekHVC [5], to see if it improves the effectiveness of our metric. In the same vein, it may help to compute differences on logarithmically-scaled intensities, which is closer to the way intensity is perceived [9].

Image clusters. Images in a large database appear to be "clustered" in terms of their proximity under our image querying metric. For example, using a portrait as a query image in our Web database selects portraits almost exclusively as targets. By contrast, using a "planet" image pulls up other planets. It would be interesting to perform some statistical clustering on the database and then show the user some representative images from the center of each cluster. These could be used either as querying keys, or merely as a way of providing an overview of the contents of the database.

Multiple metrics. In our experience with the system, we have noticed that a good query will bring up the target image, no matter which color space and decomposition method (standard or non-standard) is used; however, the false matches found in these different spaces all tend to be very different. This observation leads us to wonder whether it is possible to develop a more effective method by combining the results of searching in different color spaces and decomposition types, perhaps taking the average of the ranks in the different spaces (or, alternately, the worst of the ranks), as the rank chosen by the overall metric.

Affine transformation and partial queries. As discussed above, a very interesting (and more difficult) direction for future research is to begin exploring methods for handling general affine transformations of the query image or for searching on partial queries. The "shiftable transforms," described by Simoncelli *et al.* [28], which allow for multiresolution transforms with translational, rotational, and scale invariance, may be helpful in these respects. Another idea for specifying partial queries would be to make use of the alpha channel of the query for specifying the portions of the query and target images over which the L^q metric should be computed.

Video querying. We would like to extend our method to the problem of searching for a given frame in a video sequence. The simplest solution would be to consider each frame of the video as a separate image in the database and to apply our method directly. A more interesting solution would be to explore using a three-dimensional multiresolution decomposition of the video sequence, combined perhaps with some form of motion compensation, in order to take better advantage of the extra coherence among the video frames.

Acknowledgements

We would like to thank Leo Guibas, Dan Huttenlocher, and Eric Saund for many useful discussions during the initial phase of this project; Jutta Joesch and Werner Stuetzle for guiding us to the logit statistical model; Jack Tumblin and Matthew Turk for informative background on perception and matching; Karin Scholz for advice on the user interface; Ronen Barzel and Maureen Stone for advice on the paper; and Michael Cohen for helpful discussions along the way. We would also like to thank Anh Phan for help in scanning images; Brian Pinkerton for help in using the WebCrawler to find still more; and Piotr Brzezicki, Dennis Dean, Stuart Denman, Glenn Fleishman, George Forman, James Gunterman, Li-Wei He, Greg Heil, David Keppel, Diane King, Andrea Lingenfelter, Gene Morgan, Jonathan Shakes, Taweewan Siwadune, Christina Tonitto, Connie Wellnitz, Erin Wilson, Michael Wong, and Mikala Woodward for trying out their artistic talents as subjects in our tests.

This work was supported by an Alfred P. Sloan Research Fellowship (BR-3495), an NSF Young Investigator award (CCR-9357790), an ONR Young Investigator award (N00014-95-1-0728), a grant from the University of Washington Royalty Research Fund (65-9731), and industrial gifts from Adobe, Microsoft, and Xerox.

References

[1] R. Barber, W. Equitz, W. Niblack, D. Petkovic, and P. Yanker. Efficient query by image content for very large image databases. In *Digest of Papers. COMPCON Spring '93*, pages 17–19, San Francisco, CA, USA, 1993.

[2] G. Beylkin, R. Coifman, and V. Rokhlin. Fast wavelet transforms and numerical algorithms I. *Communications on Pure and Applied Mathematics*, 44:141–183, 1991.

[3] R. DeVore, B. Jawerth, and B. Lucier. Image compression through wavelet transform coding. *IEEE Transactions on Information Theory*, 38(2):719–746, March 1992.

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

[5] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Prentice-Hall, 1990.

[6] T. Gevers and A. W. M. Smuelders. An approach to image retrieval for image databases. In V. Marik, J. Lazansky, and R. R. Wagner, editors, *Database and Expert Systems Applicatons (DEXA '93)*, pages 615–626, Prague, Czechoslovakia, 1993.

[7] Yihong Gong, Hongjiang Zhang, H. C. Chuan, and M. Sakauchi. An image database system with content capturing and fast image indexing abilities. In *Proceedings of the International Conference on Multimedia Computing and Systems*, pages 121–130. IEEE, 1994.

[8] William I. Grosky, Rajiv Mehrotra, F. Golshani, H. V. Jagadish, Ramesh Jain, and Wayne Niblack. Research directions in image database management. In *Eighth International Conference on Data Engineering*, pages 146–148. IEEE, 1992.

[9] Donald Hearn and M. Pauline Baker. *Computer Graphics*. Addison-Wesley Publishing Company, Inc., 1994.

[10] K. Hirata and T. Kato. Query by visual example — content based image retrieval. In A. Pirotte, C. Delobel, and G. Gottlob, editors, *Advances in Database Technology (EDBT '92)*, pages 56–71, Vienna, Austria, 1992.

[11] N. Jayant, J. Johnston, and R. Safranek. Perceptual coding of images. In *Proceedings of the SPIE — The International Society for Optical Engineering*, volume 1913, pages 168–178, 1993.

[12] Atreyi Kankanhalli, Hong Jiang Zhang, and Chien Yong Low. Using texture for image retrieval. In *International Conference on Automation, Robotics and Computer Vision*. IEE, 1994.

[13] T. Kato. Database architecture for content-based image retrieval. In *Proceedings of the SPIE — The International Society for Optical Engineering*, volume 1662, pages 112–123, San Jose, CA, USA, 1992.

[14] T. Kato, T. Kurita, N. Otsu, and K. Hirata. A sketch retrieval method for full color image database — query by visual example. In *Proceedings of the 11th IAPR International Conference on Pattern Recognition*, pages 530–533, Los Alamitos, CA, USA, 1992.

[15] Patrick M. Kelly and T. Michael Cannon. CANDID: Comparison Algorithm for Navigating Digital Image Databases. In *Proceedings of the Seventh International Working Conference on Scientific and Statistical Database Management Storage and Retrieval for Image and Video Databases*. IEEE, 1994.

[16] A. Kitamoto, C. Zhou, and M. Takagi. Similarity retrieval of NOAA satellite imagery by graph matching. In *Storage and Retrieval for Image and Video Databases*, pages 60–73, San Jose, CA, USA, 1993.

[17] J. Liang and C. C. Chang. Similarity retrieval on pictorial databases based upon module operation. In S. Moon and H. Ikeda, editors, *Database Systems for Advanced Applications*, pages 19–26, Taejon, South Korea, 1993.

[18] G. S. Maddala. *Introduction to Econometrics*. Macmillan Publishing Company, second edition, 1992.

[19] Stephane Mallat and Sifen Zhong. Wavelet transform maxima and multiscale edges. In Ruskai, et al, editor, *Wavelets and Their Applications*, pages 67–104. Jones and Bartlett Publishers, Inc., Boston, 1992.

[20] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, P. Yanker, C. Faloutsos, and G. Taubin. The QBIC project: Querying images by content using color, texture, and shape. In *Storage and Retrieval for Image and Video Databases*, pages 173–187. SPIE, 1993.

[21] G. Petraglia, M. Sebillo, M. Tucci, and G. Tortora. Rotation invariant iconic indexing for image database retrieval. In S. Impedovo, editor, *Proceedings of the 7th International Conference on Image Analysis and Processing*, pages 271–278, Monopoli, Italy, 1993.

[22] Brian Pinkerton. Finding what people want: Experiences with the WebCrawler. In *The Second International WWW Conference '94: Mosaic and the Web*, October 1994.

[23] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Fetterling. *Numerical Recipes*. Cambridge University Press, second edition, 1992.

[24] T. R. Reed, V. R. Algazi, G. E. Forrd, and I. Hussain. Perceptually-based coding of monochrome and color still images. In *DCC '92 — Data Compression Conference*, pages 142–51, 1992.

[25] SAS Institute Inc. *SAS/STAT User's Guide, Version 6, Fourth Edition, Volume 2*. SAS Institute Inc., 1989.

[26] R. Shann, D. Davis, J. Oakley, and F. White. Detection and characterisation of Carboniferous Foraminifera for content-based retrieval from an image database. In *Storage and Retrieval for Image and Video Databases*, volume 1908, pages 188–197. SPIE, 1993.

[27] M. Shibata and S. Inoue. Associative retrieval method for image database. *Transactions of the Institute of Electronics, Information and Communication Engineers D-II*, J73D-II:526–34, 1990.

[28] E. P. Simoncelli, W. T. Freeman, E. H. Adelson, and D. J. Heeger. Shiftable multiscale transforms. *IEEE Transactions on Information Theory*, 38:587–607, 1992.

[29] Stephen W. Smoliar and Hong Jiang Zhang. Content-based video indexing and retrieval. *IEEE Multimedia*, 1(2):62–72, 1994.

[30] P. L. Stanchev, A. W. M. Smeulders, and F. C. A. Groen. An approach to image indexing of documents. *IFIP Transactions A (Computer Science and Technology)*, A-7:63–77, 1992.

[31] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. Wavelets for computer graphics: A primer, Part I. *IEEE Computer Graphics and Applications*, 15(3):76–84, May 1995.

[32] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. Wavelets for computer graphics: A primer, Part II. *IEEE Computer Graphics and Applications*, 15(4), July 1995. In press.

[33] Michael J. Swain. Interactive indexing into image databases. In *Storage and Retrieval for Image and Video Databases*, volume 1908, pages 95–103. SPIE, 1993.

[34] Patrick C. Teo and David J. Heeger. Perceptual image distortion. In *Human Vision, Visual Processing and Digital Display V, IS&T/SPIE's Symposium on Electronic Imaging: Science & Technology*, 1994. In press.

[35] Chen Wu Tzong and Chin Chen Chang. Application of geometric hashing to iconic database retrieval. *Pattern Recognition Letters*, 15(9):871–876, 1994.

[36] M. Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7):74–84, 1993.

A Tuning the weights of the metric

Recall that our L^q metric (3) involves a linear combination of terms. In this section, we discuss how a good set of weights w_k for these terms can be found.

The most straightforward approach for finding these weights is to use some form of multidimensional continuous optimization over these variables, such as Powell's method [23], using an evaluation function like "the number of queries that ranked their intended target in the upper 1% of the database." The difficulty is that this kind of evaluation function is fairly slow to compute (on the order of many seconds), since we would like to perform the evaluation over a large number of queries.

An alternative approach is to assume a regression model and to perform a kind of least-squares fit to the data [23]. For each pair ℓ of query and target images, we record an equation of the form:

$$r_\ell \quad = \quad v + \sum_k w_k t_{k,\ell} + u_\ell$$

where r_ℓ is either 1 or 0, depending on whether or not the query and target are intended to match; $t_{k,\ell}$ is the sum of the terms of equation (3) in bucket k; variables v and w_k are the unknowns to be found by the least-square fit; and u_ℓ is an error term to make the equality hold.

However, there are a number of problems with this method. The first problem is primarily an aesthetic one: once we have computed the weights v and w_k, they will give results that are in general neither 0 nor 1 — and in fact, may not even lie in the interval $[0,1]$. In that case, we are left with the problem of interpreting what these other values should mean. The second problem is even more serious. The difficulty is that when collecting the data for tuning the weights, it is much easier to create data for mismatches than for matches, since in a database of 1000 images, every query corresponds to 999 mismatches and only a single match. If we use all of this data, however, the least-squares fit will tend to give weights that are skewed toward finding mismatches, since the best least-squares fit to the data will be to make every query–target pair score very close to 0. The alternative, using equal-sized sets of matched and mismatched image pairs, means throwing out a lot of perfectly useful and inexpensive data.

For these reasons, we use an approach from statistics called the *logit* model [18]. In the logit model, we assume a regression model of the form:

$$r^*_\ell \quad = \quad v + \sum_k w_k t_{k,\ell} + u_\ell$$

where r^*_ℓ is called a "latent variable," which is not observed directly. Observed instead is a dummy variable r_ℓ, defined by

$$r_\ell \quad = \quad \begin{cases} 1 & \text{if } r^*_\ell > 0 \\ 0 & \text{otherwise} \end{cases}$$

The idea behind the logit model is that there exists some underlying continuous variable r^*_ℓ (such as the "perceptual closeness" of two images Q and T) that is difficult to measure directly. The continuous variable r^*_ℓ determines a binary outcome r_ℓ (such as "image T is the intended target of the query Q"), which is easily measured. The logit model provides weights w_k, which can be used to compute the *probability* that a given r^*_ℓ produces a positive outcome r_ℓ.

Indeed, under the assumptions of the logit model, the probability P_ℓ that the query–target pair ℓ is indeed a match, is given by

$$P_\ell = F\left(v + \sum_k w_k t_{k,\ell}\right) \qquad \text{where} \qquad F(x) = \frac{e^x}{1+e^x}$$

Once the weights are found, since $F(x)$ is monotonic and v is constant for all query–target pairs ℓ, it suffices to compute the expression

$$\sum_k w_k t_{k,\ell}$$

in order to rank the targets in order of decreasing probability of a match.

To compute the weights, we use the logit procedure in SAS [25]. It takes SAS about 30 seconds on an IBM RS/6000 to find appropriate weights for an input of 85 matches and 8500 (randomly chosen) mismatches. While these weights are not necessarily optimal with respect to our preferred evaluation function, they appear to give very good results, and they can be computed much more quickly than performing a multidimensional continuous optimization directly.