

# Content-based Image Retrieval: Color and Edges

Robert S. Gray  
*Department of Computer Science*  
*Dartmouth College*  
*Hanover, NH 03755*

*E-mail: robert.s.gray@dartmouth.edu*

## Abstract

One of the tools that will be essential for future electronic publishing is a powerful image retrieval system. The author should be able to search an image database for images that convey the desired information or mood; a reader should be able to search a corpus of published work for images that are relevant to his or her needs. Most commercial image retrieval systems associate keywords or text with each image and require the user to enter a keyword or textual description of the desired image. This text-based approach has numerous drawbacks – associating keywords or text with each image is a tedious task; some image features may not be mentioned in the textual description; some features are “nearly impossible to describe with text”; and some features can be described in widely different ways [Na93a]. In an effort to overcome these problems and improve retrieval performance, researchers have focused more and more on *content-based* image retrieval in which retrieval is accomplished by comparing image features directly rather than textual descriptions of the image features. Features that are commonly used in content-based retrieval include color, shape, texture and edges. In this paper we describe a simple content-based system that retrieves color images on the basis of their color distributions and edge characteristics. The system uses two retrieval techniques that have been described in the literature – i.e. *histogram intersection* to compare color distributions and *sketch comparison* to compare edge characteristics. The performance of the system is evaluated and various extensions to the existing techniques are proposed.

## 1 Introduction

One of the tools that will be essential for future electronic publishing is a powerful image retrieval system. The author should be able to search an image database for images that convey the desired information or mood; the reader should be able to search a corpus of published work for images that are relevant to his or her needs. Most commercial image retrieval systems associate keywords or text with each image in the corpus and require the user to enter a keyword or textual description of the desired image. Standard text retrieval techniques are then used to identify the relevant images. Unfortunately this text-based approach to image retrieval has numerous drawbacks [Na93a]. Associating keywords or text with each image is a tedious and time-consuming task since it must be done manually or at best semi-automatically; image processing technology is not advanced enough to allow the automatic construction of textual image descriptions except in well-defined and tightly focused domains. Some image features may not be mentioned in the textual description due to design decision or indexer error; these image features do not exist from the standpoint of the retrieval system and any query that mentions them will fail. Some features are “nearly impossible to describe with text” [Na93a]; for example many textures and shapes defy easy description. Finally different indexers – or even the same indexer – may describe the same feature with different terms or different features with the same term; these are the standard text retrieval problems of synonymy and polysemy.

In an effort to overcome the problems of the text-based approach and improve retrieval performance, researchers have focused more and more on *content-based* image retrieval in which retrieval is accomplished by comparing image features directly rather than textual descriptions of the image features. It is hoped that content-based techniques can provide the basis for powerful “*query by example*” retrieval systems. For example the user might provide a sample picture and request similar pictures, a picture of an object and request pictures that contain the object, a set of colors and request images that contain those colors, and so on. Features that are commonly used in content-based image retrieval include color, shape, texture and edges.

---

<sup>1</sup>Partially supported by AFOSR contract F49620-93-1-0266 and AFOSR/DARPA 89-0536

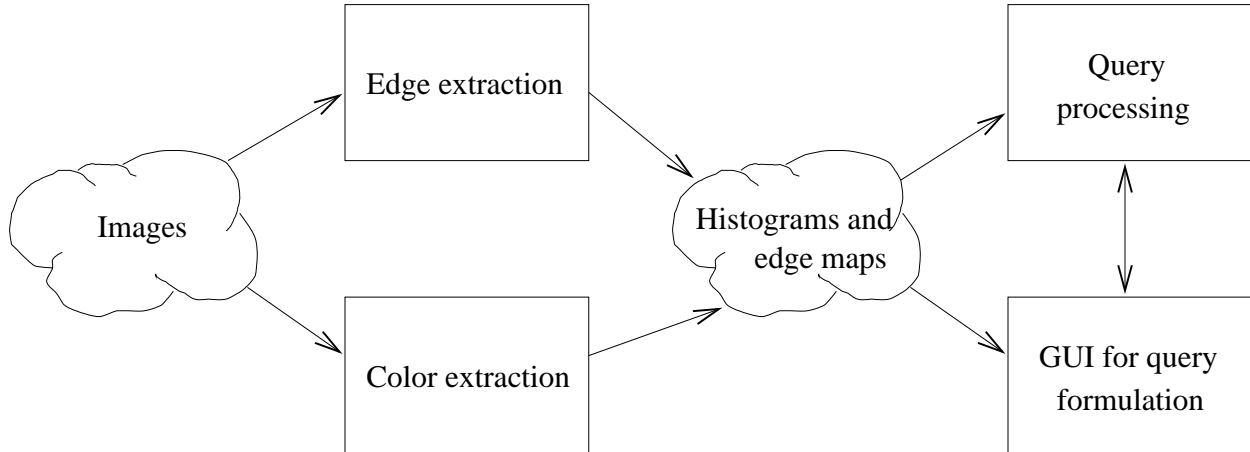


Figure 1: The architecture of the image retrieval system – the main modules are edge extraction, color extraction, query processing and user interface.

In this paper we describe a simple content-based system that retrieves color images on the basis of their color distributions and edge characteristics. The system is *fully automatic* as no manual intervention is required during the indexing process. Full automation – along with effective retrieval performance – is the primary goal of the system since a manual or semi-automatic indexing process is error-prone and time-intensive. The system does not develop any novel retrieval techniques but instead uses existing techniques that have been described in the literature – i.e. *histogram intersection* [SB91, Swa93] is used to compare color distributions and *sketch comparison* [HK92, KKO92] is used to compare edge characteristics. It is hoped that the system will highlight potential avenues of research and serve as a testbed for future work. To this end, the performance of the system is evaluated and various extensions to the existing retrieval techniques are proposed. The next section describes the implementation of the system. The remaining sections discuss the weaknesses of the current implementation and methods for addressing these weaknesses.

## 2 Implementation

The system is implemented as four modules – edge extraction, color extraction, query processing and user interface. The color and edge extraction modules construct a set of histograms and an edge map for each image. No manual intervention is required during the extraction process. The query processing module uses histogram intersection [SB91, Swa93] to compare histograms and sketch comparison [KKO92, HK92] to compare edge maps. The user interface provides a graphical front end. The four modules are shown in figure 1 and described below.

### 2.1 Edge extraction

The *edge extraction* module originally used the edge detection algorithm from [KKO92, HK92] which identifies the edges in an RGB image that are *clearly perceptible* to a human viewer. First the RGB image is reduced to thumbnail size and median filtered. Then four gradients – one for each major orientation – are calculated for each pixel in the thumbnail. The gradient masks are shown in figure 2. Each gradient is scaled by the reciprocal of the local intensity power  $|I_{ij}|$  which is defined as

$$|I_{ij}| = \left\{ \frac{1}{9} \sum_{r=i-1}^{i+1} \sum_{s=j-1}^{j+1} \bar{p}_{rs}^2 \right\}^{\frac{1}{2}}$$

1	1	1	1	0	-1	1	1	0	0	1	1
0	0	0	1	0	-1	1	0	-1	-1	0	1
-1	-1	-1	1	0	-1	0	-1	-1	-1	-1	0

Figure 2: The gradient masks that are used in the edge detection algorithm – the first gradient detects horizontal edges; the second gradient detects vertical edges; the third gradient detects diagonal edges that slope downward to the right; the fourth gradient detects diagonal edges that slope upward to the right.

where  $(i,j)$  is the pixel for which we are calculating the gradient and  $\bar{p}_{r,s}$  is the vector of RGB intensity values at pixel  $(r,s)$ . This scaling factor is a simple application of the Weber-Fechner law to the RGB color space. The Weber-Fechner law states that the “contrast sensitivity of the human eye is proportional to the log-scale of the intensity value” [HK92]).

The overall gradient for each pixel is taken to be whichever of the four gradients has the maximum absolute value. These maximal gradients are used to identify the edge pixels. First the algorithm calculates the average and standard deviation of the gradient magnitudes over the entire thumbnail image. All pixels for which the gradient magnitude is greater than the average plus one standard deviation are marked as *global edge candidates*. Then the algorithm filters the set of global edge candidates by examining the local context of each candidate. It calculates the average and standard deviation of the gradient magnitudes over a small window centered on the global edge candidate and keeps the candidate only if its gradient magnitude is greater than the local average plus one local standard deviation. An edge map is then constructed in which the pixel is on if it is one of the final edge candidates and off otherwise. The goal of this technique is to extract only those edges that are *clearly perceptible* within the image as a whole and within their local section of the image. The resulting edge map should be generally similar to a human’s impression of the image [HK92].

The algorithm was applied to a test collection of forty-eight outdoor scenes that are sold as part of the Microsoft Scenes screen saver; one of the scenes is shown in figure 3. The results were generally poor as the algorithm missed many clearly perceptible edges. This suggested that the local intensity scaling factor was an insufficient transformation of the RGB color space and motivated a switch from the RGB color space to the CIE-LUV color space. The CIE-LUV color space has the advantage that the distance between two points in the space is approximately proportionally to the perceptual distance between the two corresponding colors (as expressed by human viewers) [FVDFH91]. The conversion from the RGB color space to the CIE-LUV color space is straightforward except for the necessity of choosing a prototypical red, green and blue. Details of the conversion can be found in [FVDFH91]. Our revised edge detection algorithm first converts the RGB image to a CIE-LUV image and then uses the detection algorithm as described above except that the gradients are no longer scaled by the local intensity power.

The revised algorithm performed much better although edge map quality remained poor for a significant minority of the images. Figure 4 shows a few of the test images and their corresponding edge maps. Evaluation of edge map quality is necessarily subjective, but broadly speaking 45 percent (22 out of 48) of the maps contained every important edge (plus a small amount of noise); another 45 percent (22 out of 48) contained some important edges and some unimportant edges; and the remaining 10 percent (4 out of 48) contained no important edges. The main problem with the images that fall into the latter two categories is that many of their important edges are *clearly perceptible* only when texture or domain knowledge is considered. The edge detection algorithm considers color only.

The edge detection algorithm has parameters that can be used to control the size of the edge maps, the size of the median filtering window, and the size of the local window used to filter global edge candidates. We used a 64 x 64 edge maps, a 3 x 3 window for median filtering, and a 7 x 7 local window for filtering



Figure 3: A sample image from the image database – this image is used as the query in figures 6 and 8. The sky is deep blue with white clouds; the mountain is gray; the pine trees are dark green and the two meadows are light green. Note the complexity of the image. The only sharp boundary is the profile of the mountain.

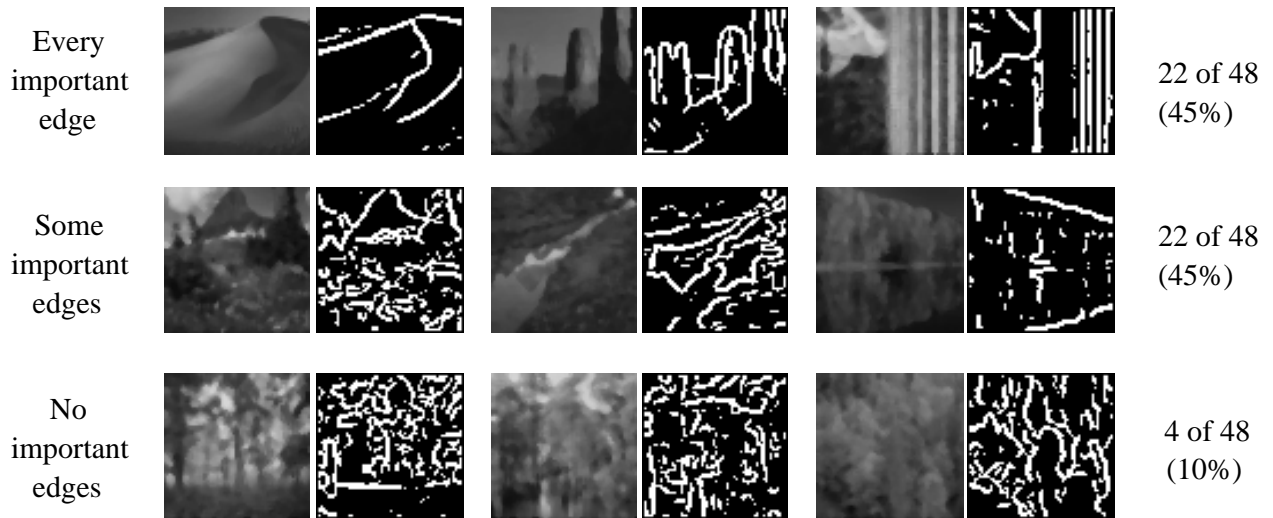


Figure 4: The performance of the edge detection algorithm – the edge maps are grouped into three broad categories depending on how many important edges they captured: every important edge (plus a small amount of noise); some important edges and some unimportant edges; and no important edges. The number of edge abstracts that fall into each category is shown at the right.

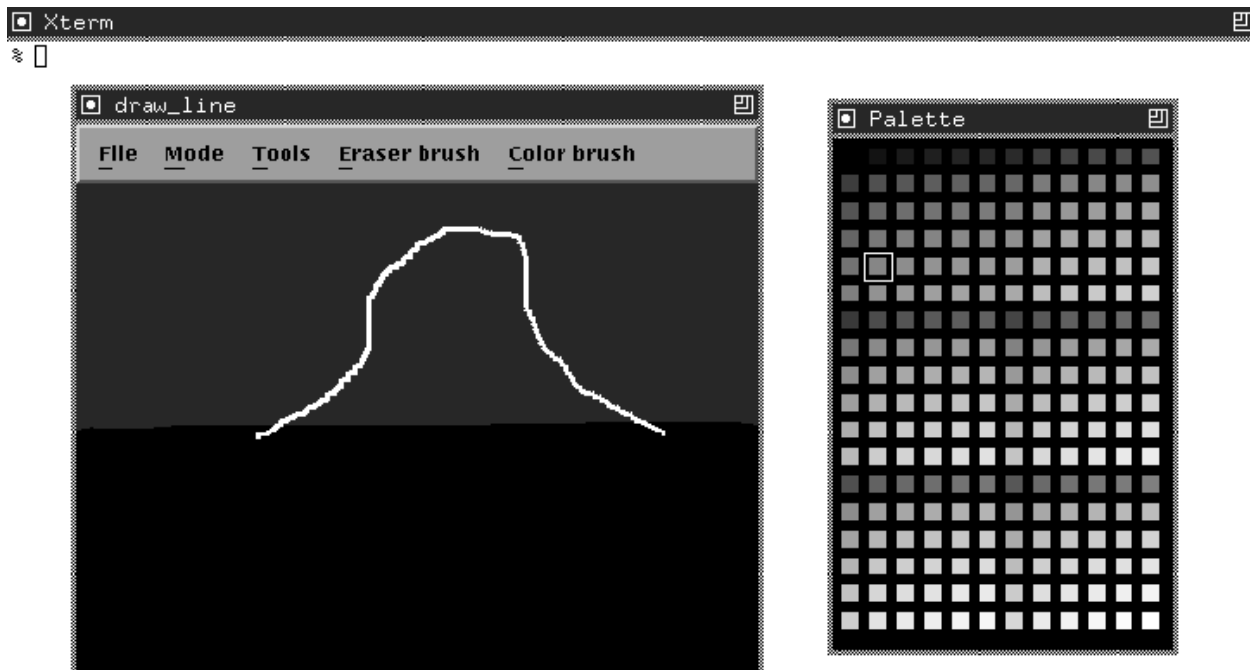


Figure 5: The GUI that allows the user to specify queries interactively

the global edge candidates. These values were used in [HK92]. Modified values did not produce significant improvements in retrieval performance or edge map quality.

## 2.2 Color extraction

The *color extraction* module divides each image into non-overlapping subareas as in [CLP94] and then constructs a three-axis histogram for the overall image and for each subarea. The module can produce either RGB or CIE-LUV histograms and provides parameters to control the number of histogram buckets along each color axis as well as the number and position of the subareas. We used CIE-LUV histograms, had 8 histogram buckets along each color axis and divided the image into four equal-sized subareas, one for each quadrant of the image. These choices provide reasonable retrieval performance. CIE-LUV histograms are computationally more expensive than RGB histograms but provide a more useful view of the color distribution since the distance between two histogram buckets is approximately proportional to the perceptual distance between the colors mapped into those buckets. This is the same advantage that motivated the switch to the CIE-LUV color space when performing edge detection.

## 2.3 Query processing

The *query processing* module accepts a single set of image histograms and a single edge map as a query. Then it identifies those images that have similar histograms and edge maps. The module computes color and edge similarity scores for each image and takes a weighted average of the two scores to get an overall similarity score.

The system uses histogram intersection [SB91, Swa93] to compute the color similarity score. For each image the *color extraction* module constructs one histogram for the overall image and one histogram for each subarea as described above. The *query processing* accepts a set of histograms as input – one histogram for the overall query and one for each subarea – and compares these histograms against each set of image histograms. The

module either compares the overall histograms using histogram intersection *or* compares each pair of subarea histograms using histogram intersection and then takes a weighted average of the subarea similarity scores. Histogram intersection defines the similarity between an image histogram  $I$  and a query histogram  $Q$  as

$$S(I, Q) = \frac{\sum_{j=1}^n \min(I_j, Q_j)}{\sum_{j=1}^n Q_j}$$

where  $n$  is the number of buckets in the histograms,  $I_j$  is the number of pixels in bucket  $j$  of the image histogram, and  $Q_j$  is the number of pixels in bucket  $j$  of the query histogram. Histogram intersection was originally developed to identify which images *contain* a given object (a prototypical image of the object is provided as a query). Thus the purpose of the *min* in the similarity measure is to filter out the background pixels in each image, leaving only those pixels that might belong to the object. Histogram intersection has been shown to be insensitive to “change in image resolution, histogram size, occlusion, depth and viewpoint” [Swa93] and has provided excellent performance when finding images that contain a given object [SB91]. This result should hold when the technique is used to determine the similarity between two equal sized images.

The system uses a slightly modified version of sketch comparison [HK92, KKOH92] to compute the edge similarity score. The *edge extraction* module constructs an edge map for each image as described above. The *query processing* module accepts an edge map as input and compares this map against each of the image maps. First the query edge map is divided into small non-overlapping blocks. Each query block is correlated with a small neighborhood of blocks in the image edge map. The neighborhood is centered on the image block that exactly corresponds to the query block. The correlation between a query and image block is defined as a sum of weights where there is one weight for each of six possible cases – query edge lined up with image edge; query edge lined up with an image blank; query edge lined up with a position that is off the image; query blank lined up with an image edge; query blank lined up with an image blank; and query blank lined up with a position that is off the image. Computing this correlation is simply a matter of performing a pixel-by-pixel comparison of the query and image blocks. The maximum correlation over all image blocks in the local neighborhood is taken to be the correlation score for the query block. In other words the algorithm tries various shifts of the query block and chooses the shift that provides the best match. Thus there is some flexibility in the matching process; the query module can retrieve images that have edges similar to those of the query but in slightly different positions. The correlation scores for the query blocks are summed and divided by the maximum possible sum for that particular query to get the edge similarity score.

The color and edge similarity scores are real numbers between 0 and 1 where 1 indicates maximum similarity and 0 indicates maximum dissimilarity. The final similarity score is between 0 and 1 since it is just a weighted average of the edge and color scores. Once the system has calculated the overall similarity score for each image, it sorts the images in order of decreasing similarity for presentation to the user.

The various weights are parameters that can be specified at query time. We weighted the edge and color similarity scores equally; weighted each subarea equally; and used query edge/image edge, query edge/image blank, query edge/off image, query blank/image edge, query blank/image blank and query blank/off image weights of 10, -3, -1, -3, 1 and -1 respectively. These weights take into account the fact that a match between two edges is far more important than a match between two blanks. In addition the sketch comparison algorithm uses 8 x 8 blocks and defines the local neighborhood of image blocks to be all blocks that are 4 or fewer pixels away from the center block.

## 2.4 Graphical user interface

Users can interactively “draw” their own queries using the GUI shown in figure 5. The drawing area is essentially two logical drawing areas laid on top of each other – one drawing area is for color and the other is for edges. The edge drawing area is on top of the color drawing area so edges are never hidden behind colors. In the figure the user has drawn the outline of a mountain in the edge drawing area and has filled the top half of the color drawing area with blue in order to indicate sky. Once the user has finished the query, the color drawing is histogrammed, the edge drawing is turned into an edge map, and the histograms and

edge map become the input to the query module. The query module returns a ranked list of images which is presented to the user as an ordered list of image filenames (our initial focus has been on the retrieval algorithms rather than GUI development).

## 2.5 Implementation notes

The total system currently consists of approximately 6000 lines of C++ code written and compiled on DEC workstations (although it should be trivially portable to other architectures). More than 1500 of these lines are for the graphical user interface. A complete code listing is available upon request.

## 3 Preliminary evaluation

A preliminary evaluation suggests several problems with the retrieval techniques. The system was evaluated with a database of forty-eight outdoor scenes that are sold as part of the Microsoft Scenes screen saver; one of the scenes is shown in figure 3. These are the same scenes that were used to evaluate the edge detection algorithm. Figures 6–9 show the results of four specific queries that were made against this image database. Two queries involve only color and two queries involve only edges; queries that involve *both* color and edges seemed premature in light of the retrieval problems that were identified. In general the results of these four queries are typical of the retrieval behavior exhibited by the system. The results and associated problems are discussed below. Methods for addressing the problems are discussed in the next section.

Figure 6 shows the result of submitting an image to the retrieval system and requesting images that have a similar color distribution. The query image is the first image in the test collection and is shown in figure 3. In this case we compared the query to each image on a subarea basis and averaged the subarea similarity scores to get the overall color similarity score. The similarity score for each subarea was calculated with *histogram intersection* as discussed in the implementation section. The retrieval results are reasonably good in that seven of the top ten images have a composition similar to that of the query image – green grass or trees at the bottom, blue sky at the top and in most instances a mountainous region in the center. In addition there were no relevant images ranked below the top ten. However three of the images clearly do not belong. The image with ranking 0.32 has no blue sky or gray mountain – it scores highly because it has lots of green at the bottom and several regions of small gray rocks at the top. The images with ranking 0.44 and 0.28 have no blue sky or gray mountain – they score highly because they have lots of green at the bottom.

This query illustrates the first two problems with our color retrieval mechanism. First there is no way of specifying that the *absence* of a certain color from a region means that the image is irrelevant. For example we would like to specify that the absence of blue sky means that the image is irrelevant. This can not be accomplished merely by increasing the weight of the two subareas at the top of the image since then we might retrieve images that have blue sky but no green grass. Rather we need to indicate that a negative result in one subarea overrides even the most positive result in another subarea. Second the algorithm does not provide effective localization – i.e. it does not take into account that a color might need to appear in a specific location within a subarea – especially since the subareas are large with respect to the image. For example the gray mountain in the query image matches well against several widely separated regions of gray rock in one of the highly ranked irrelevant images.

Figure 7 shows the result of submitting a hand-drawn color image to the system. The user has drawn a monochrome blue region at the top and a monochrome green region at the bottom in an effort to retrieve images that have blue sky at the top and green grass or trees at the bottom. As before we compared subareas and averaged the subarea similarity scores to get the overall similarity score. The results are poor. Only six images have a nonzero similarity score and only one of these six can be considered relevant to the query (and this relevant image has the lowest score).

The first five images illustrate the same problem discussed above. There is no way to specify that the absence of a certain color from a certain region means complete irrelevance. Thus we retrieve the images with ranks 0.06, 0.11 and 0.24 which contain green at the bottom but no blue at the top and the images with ranks

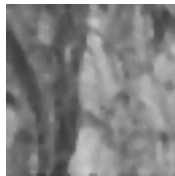
## Query



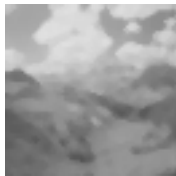
## Results



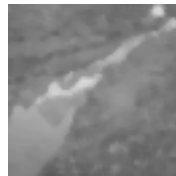
1.00



0.44



0.37



0.32



0.30



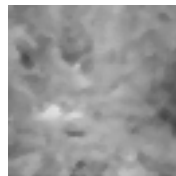
0.30



0.30



0.29



0.28



0.27

Figure 6: Here we are using the image shown in figure 3 as a *color query* – i.e. we want to find all images that have a similar color distribution. The ten highest-ranked images are shown. As expected the highest-ranked image is the query image itself.

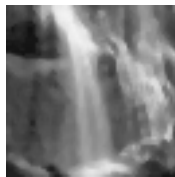
## Query



## Results



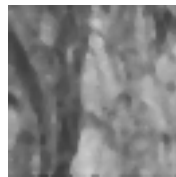
0.35



0.24



0.15



0.11



0.06



0.06

Figure 7: Here we are using a hand-drawn image as a *color query*. The query consists of a monochlor blue region at the top and a monochlor green region at the bottom – it is hoped that this query will retrieve images that have blue sky at the top and green grass or trees at the bottom. The six highest-ranked images are shown. All other images had a rank of zero.

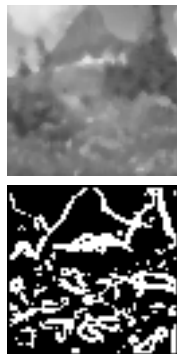
0.35 and 0.15 which contain blue at the top but no green at the bottom. However a more critical problem is that only one relevant image received nonzero similarity score. The other relevant images – even the most relevant image which has an unbroken green field at the bottom, blue sky at the top and a flat horizon – had similarity scores of zero. The problem is that the histogram intersection algorithm performs *exact* color match. Only a single blue and a single green are used in the query; all images that do not contain the exact same shades of blue and green have an empty intersection with the query and therefore a similarity score of zero. Furthermore an image that contains twenty different shades of green will have a small similarity score when compared against an image that contains only of one those shades even if each image contains the same amount of green. This problem did not arise with the previous query since that query was an image taken directly from the database and contained enough different shades of blue, green and gray to ensure at least some match with all relevant images. However the problem can arise when using certain real-world images as queries and is critical when using hand-drawn images as queries.

Figures 8 and 9 are best considered together. Figure 8 shows the result of submitting the first image in the test collection and requesting images that have similar edge characteristics. The results are poor in the sense that there are relatively high ranks on irrelevant images and that there is little discrimination on the basis of rank between relevant and irrelevant images. Figures 9 shows the results of submitting a hand-drawn query of a mountain top in an effort to retrieve images that contain mountains (particularly a desired target image that contains an almost identical mountain top). In this case the weights in the sketch comparison algorithm were adjusted such that blank regions of the query were treated as “don’t care” regions. Otherwise the query retrieves images that contain mainly blank space. The target image was successfully retrieved but again there are relatively high ranks on irrelevant images and there is little rank discrimination between relevant and irrelevant images. In addition the relevant images that score highly – aside from the target image – score highly by chance. The query edge pixels line up with image edge pixels that correspond to foliage boundaries, cloud boundaries and so on.

There are three problems. The first problem is that the test collection is small. The results are partially an artifact of the fact that there are only a few images relevant to each query. However this problem is minor in comparison to the other two. The second problem is that many edge maps contain extraneous edges that are perceptually prominent on the basis of color but spurious when one considers domain knowledge. In addition some edge maps miss important edges that are prominent on the basis of texture or domain knowledge but not on the basis of color. Fifty-five percent of the edge maps contain some unimportant edges or miss some important ones (see the implementation section). For example the edge map for an image of trees through fog essentially contains edges that trace each individual leaf cluster and no other edges. It would be more reasonable to have an edge that followed each tree trunk. Unfortunately the tree trunks are far less prominent than the leaf clusters on the basis of color alone since the trunks blend into the fog. [HK92] and [KKOH92] have cleaner edge maps due to the nature of their images. They use a collection of paintings – primarily landscapes and portraits – that tend to have far sharper color boundaries than our outdoor photographs.

The third problem is the most critical and has a drastic effect on retrieval performance. The sketch comparison algorithm divides the query and image edge maps into blocks and then compares the blocks on a pixel by pixel basis under various shifts. This pixel-by-pixel approach leads to nonintuitive results as shown in figure 10. The edge maps shown in the figure were constructed by hand but are representative of some of the actual edge maps for the test collection. Parts (a), (b) and (c) of the figure are representative of the edge maps for hillsides. Part (d) is representative of the edge map for a rosebush. Parts (e) and (f) are representative of edge maps for the tops of pine trees. The similarity scores clearly do not indicate the true similarity between the edge maps. The problem is that the sketch comparison algorithm ignores higher level features such as edge orientation, shape and connectedness. Instead it simply counts the number of edge pixels that can be aligned by shifting query blocks around on top of the image (more precisely the similarity score is proportional to the number of edge pixels that can be aligned). The result is that often one edge in the query scores highly when matched against several disconnected edges in the image *or* an edge in the query scores poorly against a highly similar edge in the image since they are just different enough that only a few edge pixels line up. The method appears to perform well enough for retrieval of a *known* or *remembered* target image on the basis of a *well-drawn* sketch or for retrieval of similar images in a larger database than

## Query



## Results

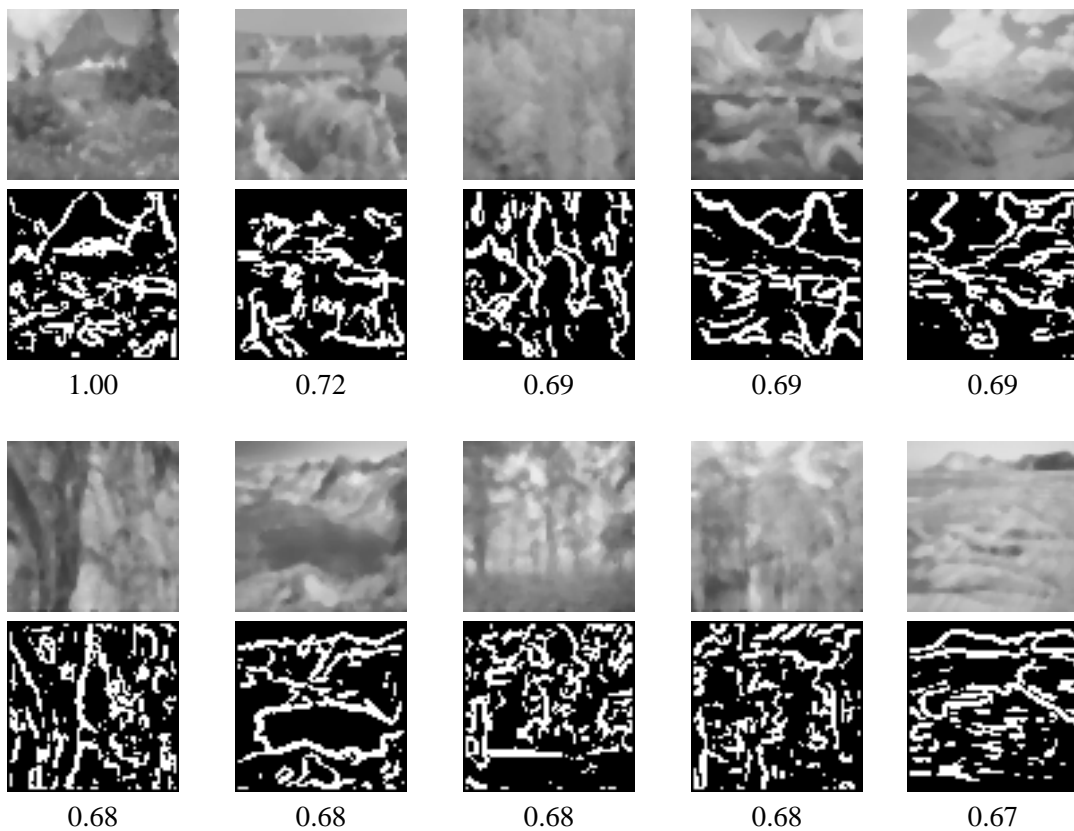


Figure 8: Here we are using the image shown in figure 3 as an *edge query* – i.e. we want to find all images that have a similar pattern of edges. The ten highest-ranked images are shown. As expected the highest-ranked image is the query image itself.

Query



Desired image



Results



0.89



0.89



0.85



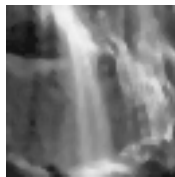
0.83



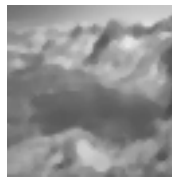
0.82



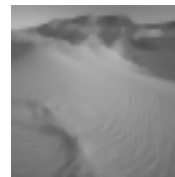
0.82



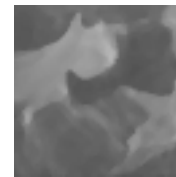
0.81



0.80



0.79



0.78



Figure 9: Here we are using a hand-drawn query as an *edge query*. It is hoped that this query will retrieve images that have mountain peaks in the center (particularly the indicated “desired” image which the author was looking at when he drew the query). The ten highest-ranked images are shown.

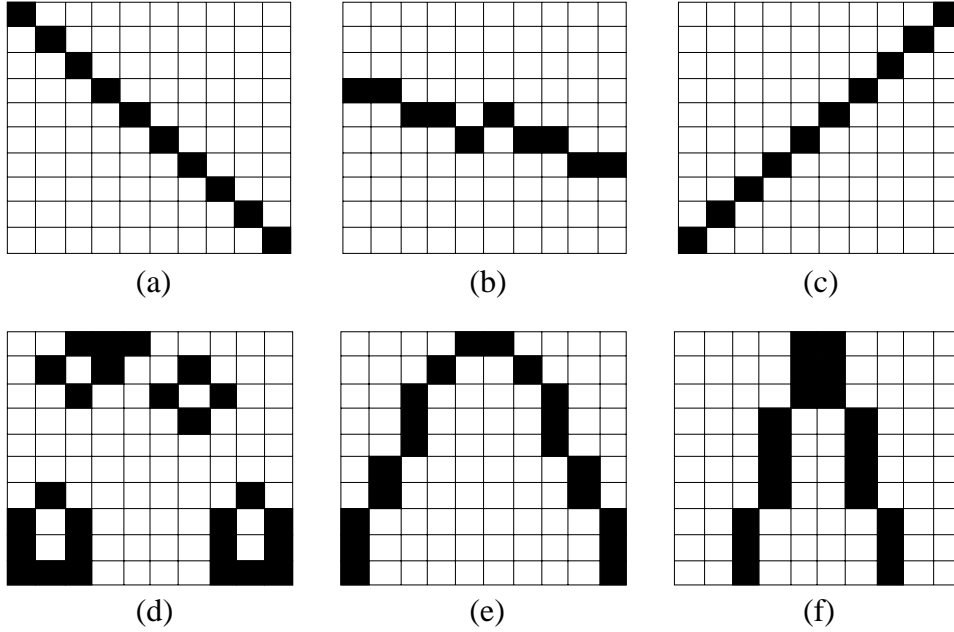


Figure 10: The fundamental weakness of the edge-based retrieval technique is that it performs a pixel by pixel comparison of the edge maps – therefore it reports that  $b$  is equally similar to  $a$  and  $c$  (in each case two edge pixels line up under the best possible shift) and that  $e$  is more similar to  $d$  than to  $f$  (thirteen edge pixels line up between  $d$  and  $e$  under the best possible shift but only eight edge pixels line up between  $e$  and  $f$ ).

ours. These are the situations that were considered in [HK92] and [KKOH92] where sketch comparison was observed to provide reasonable performance. However it clearly does not provide reasonable performance in the case of arbitrary queries against arbitrary image databases.

## 4 Future work

### 4.1 Color

#### 4.1.1 Weighting

The color retrieval technique has several weaknesses that were discussed in the evaluation. First there is no way for the user to specify that the absence of a color from a certain region means that the image is irrelevant. There are many similar situations. For example there is no way to specify that a region must contain *either* blue *or* green. The underlying problem is that the color similarity score is just a weighted average of the subarea similarity scores which means that the searcher has only the coarsest level of control over query behavior. This problem can be partially solved without abandoning the subarea scheme. The overall similarity score should be a nonlinear function of the subarea similarity scores so that the score for a particular subarea can have any desired effect on the overall score. In addition each subarea similarity score should be a nonlinear function of one *or more* histogram intersections.

#### 4.1.2 Localization

Unfortunately there remains an inherent weakness in the subarea scheme. The subareas are large relative to the image size and the system compares subareas by comparing just their histograms. Therefore a subarea

with a pine tree on the left will score highly when matched against a subarea with a pine tree on the right. More critically a subarea that contains a single large gray area will score highly when matched against a subarea that contains several small gray areas. The problem is that the retrieval mechanism provides no color localization below the subarea level. The obvious solution is to significantly increase the number of subareas. This causes a drastic increase in computational and storage requirements. A better solution is to leave the number of subareas unchanged and have the system perform a direct comparison of the image and query if there is sufficient similarity between the histograms. Either approach allows the user to specify that a color should appear as a connected region with a specific shape and position. However we should not limit ourselves to saying that location is always important. For each query the user should be able to specify that the location of certain colors is unimportant or unimportant to a certain degree – i.e. this color should appear somewhere within this region of the image – without losing the ability to specify that a color should form a connected region with a specific shape. Neither of the approaches can efficiently support this behavior. The only recourse is to run multiple queries where each query is a transformation of the given query. This is computationally intractable for large databases or complicated queries.

To handle the problem of color localization – and the problem of specifying on a color by color basis whether location matters – many researchers use segments rather than subareas or a combination of segments and subareas [CLP94, Na93b, GZCS94]. Each image is divided into segments such that each segment contains approximately a single uniform color or a single object. The color distribution of each segment is represented as a weighted centroid or as a histogram. The query is specified as a set of colored segments and the query segments are matched against the image segments – features such as segment size, location and shape are compared in addition to segment color. Dividing an image into object-based segments can not be done automatically with current image processing technology so it is more attractive to segment the image into regions of uniform color. In either case the system should be prepared to match a single query segment against multiple image segments or vice versa since the user might specify the query segments at a finer or coarser grain than the image segments. In addition to handling localization, the use of segments will make it easier to provide the fine-grained query control that was discussed in the *weighting* subsection since query behavior can be specified on a segment by segment basis – e.g. this segment can occur anywhere within this region while this segment is at a fixed location; this segment must be present in order for the image to be relevant; the size of this segment can vary; the shape of this segment can be anything from a thin oval to a full circle; these two segments can be anywhere but must be adjacent and so on. Some of these are simply instances of the localization problem while others involve other segment features beside location; the subarea-based approach can handle them only by submitting multiple queries which is computationally unattractive. We plan to move to a segment-based scheme in order to provide localization and fine-grained query control.

### 4.1.3 Exact color match

The third problem with the color retrieval is that histogram intersection performs an exact color match. It must be modified to allow inexact color match so that one shade of green will match similar shades of green. This is relatively simple in the case of histogram intersection. Rather than intersect a query bucket with an image bucket, the algorithm would intersect the query bucket with a neighborhood of image buckets. Different weights would be used for different buckets and care must be taken to not double-count pixels during the intersection process. Note that reducing the number of histogram buckets along each color axis would not have the same effect. It would lead to inexact color match for colors mapped into the center of buckets but not for colors mapped to the edge; in addition the user would not be able to specify inexact color match on one query and exact color match on the next.

### 4.1.4 Efficiency

Efficiency of histogram intersection is not a large concern at this point although response time will be poor for large databases. In this case there is a much more efficient version of histogram intersection called *incremental histogram intersection* that intersects the buckets in order of decreasing pixel count and stops

if it determines that the similarity between the histograms can not possibly be more than some threshold [SB91]. However efficiency of the segment comparison algorithm – of which histogram intersection is just a part – will be critical when we move to segment-based retrieval. Reasonable efficiency will require an excellent representation for segment features as well as a hierarchical or cluster-based retrieval scheme. It should not be hard to implement a hierarchical scheme. For example comparing the overall query histogram with the overall image histograms can immediately eliminate most of the database from consideration. Cluster-based retrieval will be more difficult since we must choose the color features used in the clustering process carefully. For example, if we cluster the images according to dominant color and then request images that contain a small region of blue in one corner, we have gained nothing; we must search the entire database.

#### 4.1.5 Summary

We plan to move to a segmented-based scheme in order to provide localization and fine-grained query control. Histogram intersection needs to be modified to allow inexact color match. Eventually we will need to incorporate a hierarchical or cluster-based scheme for the sake of search efficiency. In addition we would like to evaluate other color retrieval schemes that have been presented in the literature. For example [CLP94] uses the color pair technique and has achieved good results in a small database; QBIC uses a matrix-based technique that takes the product of a difference histogram and a set of perceptual color distances, but unfortunately there is no analysis of retrieval performance [Na93b]; [GZCS94] reduces an entire histogram to a single integer key by first transforming the histogram into a hyper-polygon and then taking a weighted sum of the angles and edge lengths, but again there is no analysis of retrieval performance. The technique of [GZCS94] will be exceptionally useful if it provides reasonable retrieval performance since then the first few levels of a hierarchical or cluster-based retrieval scheme would involve comparison of integer pairs rather than histogram pairs.

## 4.2 Edges

### 4.2.1 Edge detection

The edge detection algorithm does not construct good edge maps for every image as discussed in implementation and evaluation sections. There are two main problems with the edge maps. First the edge maps for some images contain “extraneous” edges that carry no useful information for retrieval purposes even though they are perceptually prominent in the image. Some of these extraneous edges arise due to “color noise” – i.e. a small group of pixels whose color is sharply different than all their neighbors. More aggressive median filtering and a thinning procedure to strip out the shortest edges [KKOH92, HK92] will eliminate some of these extraneous edges. In addition we should experiment with the gradient threshold to see if a higher threshold will give better results – e.g. keep only those edge pixels whose gradient strength is greater than the average plus *two* standard deviations. However preliminary experimentation suggests that increasing the threshold will eliminate not only some of the extraneous edges but portions of good edges. A multi-resolution edge detection scheme might lead to some improvement but the improvement is liable to be small in relation to the computational effort.

None of the techniques above will come close to solving the whole problem since many of the extraneous edges are as perceptually prominent as the valid edges and are relatively long. However five heuristics are immediately apparent. Edges that cross a large portion of the image – such as the horizon – tend to be far more important than edges that wind around and around in a small portion of the image – such as the boundary of a complex cluster of leaves. Longer edges – such as the profile of a mountain – tend to be far more important than shorter edges – such as the profile of a small boulder in front of the mountain. Edges that surround a large region – such as a large boulder – tend to be far more important than edges that surround a small region – such as a cluster of a few flowers. Short offshoots of long edges tend to be unimportant. Lots of short edges concentrated in a small region tend to be unimportant. In order to apply these heuristics we need to determine edge lengths and edge geometries. Extracting the lengths and geometries requires grouping the edge pixels into edges. This could involve an edge tracing algorithm applied

to either the original image or to an intermediate edge map. It might also be reasonable to segment the original image on the basis of color or texture or both and use the segment boundaries as the edges. A final thought is to use a reinforcement scheme in which edge points reinforce all connected edge points in some fashion; then a much higher threshold can be used when selecting the final edges. The reinforcement scheme could be combined easily with the edge tracing scheme.

Any of the heuristics are likely to eliminate valid edges as well as extraneous edges if applied too stringently so experimentation will be necessary to determine the appropriate balance between eliminating extraneous edges and keeping good edges. None of these heuristics will achieve good edge detection for every image since some edges are extraneous to humans only because humans *know* what objects are shown in the image. However the heuristics should improve the edge maps for many images.

The second problem is that good edges or portions of good edges are missing from some edge maps. An edge tracing algorithm could jump over and fill in small gaps. A reinforcement algorithm might reinforce all adjacent neighbors of strong edge points. If we have obtained a higher level representation of edges – i.e. perhaps the edges are represented as paths rather than pixel maps – through tracing or segmentation, we might connect all edges whose endpoints are sufficiently close and whose ends have sufficiently similar orientations. Considering texture in addition to color will capture some additional important edges. Again none of these approaches will provide good detection for every image since some edges are important only when one considers which objects are shown in the image.

Before implementing and comparing these approaches, it would be useful to determine how much performance improvement can be realized through cleaner edge abstracts. Thus we plan to manually outline the edges in a small test collection and run retrieval experiments against that collection. This manual outlining is unattractive as a permanent solution since it violates the goal of automatic indexing, but it will allow us to obtain a rough measure of the maximum performance improvement before we invest substantial coding effort.

#### 4.2.2 Sketch comparison

The *sketch comparison* algorithm is the weak point in the retrieval scheme. Sketch comparison calculates the similarity between two edge maps with a pixel by pixel comparison of the edge abstracts. The similarity score is proportional only to the number of edge pixels that can be made to line up under small shifts. This leads to incorrect results in many common situations as shown in figure 10. There are two contradictory problems. First, although blocks of pixels are allowed to shift in search of the best match, the pixels within the block are fixed so it is often possible to line up only a few pixels of two highly similar edges. Second it is possible for a single edge in one image to match multiple portions of distinct edges in another image. The contradiction lies in the fact that increasing the mobility of blocks and pixels during the matching process eases the first problem but makes the second far worse; restricting their mobility has the opposite effect.

It appears that the pixel-by-pixel comparison approach must be abandoned in order to support arbitrary, *possibly poorly drawn* queries in heterogeneous databases. An intermediate step is to keep the pixel-oriented edge maps but to make the query edge-oriented – e.g. the query edges are represented as a set of paths rather than as an array of pixels. It is easy to obtain this set of paths when the user hand-draws the query; it is more difficult when the query is a sample image from the database since it requires the use of edge tracing or segmentation techniques as mentioned above. The similarity score for a given *edge* in the query could be a function of the number of edge pixels that it covers in the image, whether those pixels are connected or disconnected, and how much the query edge needs to deform in order to move into position over the pixels. This concept is similar to the idea of “active snakes” which are often used in interactive outlining applications [Na93b]. A more drastic step would be to make the edge maps edge-based as well. Then the problem of comparing a query to an image would become a problem of comparing the feature values corresponding to the various edges. Possible edge features are location, orientation, length, turning rate and so on. This would again require a more complicated edge detection algorithm but we might be using such an algorithm anyways for the purpose of cleaning extraneous edges out of the edge abstracts. In addition an entirely edge-based approach will make it easier to provide fine-grained query control – e.g. this edge can appear anywhere within this region of the image, this edge must be in this position but can have either of

two orientations and so on. The segment-based approach to color retrieval has the same advantage.

### 4.2.3 Efficiency

The remaining problem with *sketch comparison* is efficiency. The algorithm is faster than one might expect. However the pixel-by-pixel technique involves a quarter of a million pixel comparisons just to compute the similarity between two 64 x 64 edge abstracts if the algorithm uses 8 x 8 blocks and allows a maximum shift of 4 pixels in any direction. The efficiency should improve somewhat with a careful implementation of the “active snake” or feature-based schemes mentioned above. However it seems clear that a hierarchical or clustering retrieval scheme will be needed as we move towards larger databases. We are attempting to determine which features of the edges will be useful in forming hierarchies or clusters. At a minimum images could be grouped according to which image regions contained no edges. However much more sophisticated schemes are possible.

### 4.2.4 Summary

We plan to move towards a path-based representation of the edge maps rather than a pixel-based representation in order to improve retrieval performance and provide fine-grained query control. In addition we are experimenting with various heuristics for filtering extraneous edges out of the edge maps and capturing more valid edges. Finally it is essential that we develop a hierarchical or cluster-based retrieval scheme.

## 4.3 System evaluation

The preliminary evaluation used a test collection of only forty-eight images. We must move to a larger test collection in order to perform a detailed evaluation of the system and of the extensions that have been proposed. A larger test collection is particularly essential for examining the time requirements of the retrieval mechanisms. We are currently building a large image corpus that can be used for this and other image retrieval work. However the small test collection should not be abandoned since it is useful for exploring the situation in which there are only a few relevant images per query and in which images tend to be highly distinct from each other.

In addition we need to explore how well the system performs with databases of gray-scale or black and white images since many archival image collections have no color. Any color retrieval mechanism will provide far less discriminatory power since it will be histogramming only a range of grays or only two colors (in which case the histogram is reduced to a measure of image *density*). However edges – and other features such as texture and shape – should work well with colorless databases.

Finally we need to evaluate the graphical user interface (GUI) in terms of the ease with which the user can construct an appropriate query. This issue will become more acute as we incorporate additional features such as texture and shape into the system since the user must be able to easily combine multiple features into a single query. In addition the user must be able to easily specify the characteristics of and the relationships between the various parts of the query. The current interface is simplistic and will need substantial reengineering to achieve these goals.

## 4.4 Text, texture and shape

Color and edges are only two of a wide range of image features that can be used in a content-based retrieval system. One common feature that is used in content-based retrieval is texture. [SC94] uses quad-tree segmentation to divide an image into blocks of approximately uniform texture. Feature vectors for the textures are computed from mean and variance measures produced by a QMF wavelet decomposition. The user queries the image database by selecting a desired texture from a set of prototypical textures. The database is searched for images that contain blocks of the desired texture. The approach works well on a collection of synthetic images but has not yet been tested on real-world images [SC94].

The QBIC project [Na93b, Na93a] allows texture-based retrieval although it uses different features. *Coarseness* measures the scale of the texture and is computed with moving windows of several sizes; *contrast* describes the “vividness” of the texture and is calculated from the gray-level histogram; and *directionality* measures whether the image has a “favored” direction and is computed from the gradient directions in the image [Na93b]. The author notes that many other texture features were either too expensive to compute or ill-suited to heterogeneous collections of images [Na93b]. The user queries the database by providing a swatch of the desired texture which is then matched against the images. Unfortunately the authors do not provide an analysis of retrieval performance.

A second common feature is the shape of the objects in the images. In the QBIC project [Na93b] a combination of area, circularity, eccentricity, major axis orientation and moment invariants are used as shape features. In [GZCS94] only circularity and major axis orientation are used. In QBIC the user draws the desired shape. Then the system computes the features of the query shape and matches the query features against the features of each shape in the images. In [GZCS94] the user does not draw the shape but rather specifies the values of the two shape parameters directly. Unfortunately both systems require that the images be segmented along *object* boundaries in order to identify the shapes that a user is likely to use when retrieving images. QBIC resorts to a manual approach in which a human manually outlines the desired shapes using an interactive “shrink-wrap” utility [Na93b]. In [GZCS94] the images are segmented automatically on the basis of color but a postprocessing step is required to recover from over-segmentation; the postprocessing step is not described but is probably manual. Both authors provide no analysis of retrieval performance.

Texture and shape will be incorporated into our retrieval system as soon as some of the weaknesses of the color and edge retrieval mechanisms have been addressed. Shape will be more difficult to incorporate since it appears to rely on segmenting images along object boundaries. However it may be possible – although nontrivial – to develop a representation that allows a single query shape to match against multiple image shapes and vice versa. This would maintain our goal of automatic indexing and retrieval since an image would not need to be segmented along object boundaries (current image processing technology can not accurately divide an arbitrary image along object boundaries).

The use of additional image features such as texture and shape will improve retrieval performance. However text should not be ignored since many images must have text associated with them anyways (e.g. captions for images in a book) and many queries are impossible to answer without examining this text. For example finding all photographs taken in Paris is impossible without looking at the photograph captions. Allowing the user to search any available text will be a critical addition to the retrieval system despite the errors and inconsistencies in normal text. A simple text retrieval mechanism will be incorporated soon.

## 5 Conclusion

This paper has presented a content-based image retrieval system. The system retrieves color images on the basis of their color and edge characteristics. Two existing retrieval techniques – *histogram intersection* and *sketch comparison* – are used to compare color distributions and edge maps. The performance of the system shows some promise – particularly with respect to color – but falls far short of the performance that is required for practical electronic publishing. Various methods to address the weaknesses of the retrieval techniques have been mentioned – most notably moving towards a segment-based scheme for color matching and towards a path-based scheme for edge matching – and will be explored in future work. In addition many other image features can be used for content-based retrieval. Texture and shape are the two most common and will eventually be incorporated into the system. A simple text retrieval mechanism will also be added.

## 6 Acknowledgements

Many thanks to Jing Feng and Professor Fillia Makedon for useful discussions; to my advisor, Professor George Cybenko, for his encouragement and support; and, as always, to Jennifer and Stephen Gray for reminding me that there is life outside graduate school.

## References

- [CLP94] Tat-Seng Chua, Swee-Kiew Lim, and Hung-Keng Pung. Content-based retrieval of images. In *Multimedia 94*, pages 211–218, San Francisco, California, 1994. ACM.
- [FVDFH91] James D. Foley, Andries Van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, Reading, Massachusetts, second edition, 1991.
- [GZCS94] 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, Boston, Massachusetts, 1994. IEEE.
- [HK92] Kyoji Hirata and Toshikazu Kato. Query by visual example. In *Advances in Database Technology EDBT 1992, Third International Conference on Extending Database Technology*, pages 56–71, Vienna, Austria, 1992. Springer-Verlag.
- [KKOH92] Toshikazu Kato, Takio Kurita, Nobuyaki Otsu, and Kyoji Hirata. A sketch retrieval method for full color image databases. In *International Conference on Pattern Recognition (ICPR)*, pages 530–533, The Hague, The Netherlands, 1992. IAPR.
- [Na93a] Wayne Niblack and all. The QBIC project: Querying images by content using color, texture and shape. *SPIE*, 1908:173–187, 1993.
- [Na93b] Wayne Niblack and all. The QBIC project: Querying images by content using color, texture and shape. Research Report RJ 9203 (81511), IBM Research Divison, Almaden Research Center, San Jose, California, 1993.
- [SB91] Michael J. Swain and Dana H. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.
- [SC94] John R. Smith and Shih-Fu Chang. Quad-tree segmentation for texture-based image query. In *Multimedia 94*, pages 279–286, San Francisco, California, 1994. ACM.
- [Swa93] Michael J. Swain. Interactive indexing into image databases. *SPIE*, 1908:95–103, 1993.