RETRIEVAL OF PETROGLYPH IMAGES

FROM A DATABASE

USING IMAGE INFORMATION

THESIS

Presented to the Graduate College of
Southwest Texas State University
in Partial Fulfillment of
the Requirements

For the Degree

Master of SCIENCE

By

Cynthia G. Huyser, B. A.

San Marcos, Texas
December 2002

DEDICATION

To my heart partner, Debra Winegarten,

and our friend, Josi Mata, who loves archaeology

ACKNOWLEDGEMENTS

I would like to thank my parents, Gloria Huyser and the late Willis Huyser, for helping me understand the importance of learning.

I am deeply grateful to my thesis advisor, Prof. Wilbon Davis, for introducing me to the field of computer vision and for his support, encouragement and guidance in the pursuit of my thesis topic. I would also like to express my appreciation to Dr. Tom McCabe and Dr. Robert Ogden, whose classes helped give my computer science education at Southwest Texas a firm foundation, for their participation as members of my thesis committee.

Finally, I want to express my deepest thanks to my partner, Debra Winegarten, for her unwavering love, support, and encouragement during my graduate education.

This manuscript was presented to the Graduate College of Southwest Texas State University on December 2, 2002.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

ABSTRACT

RETRIEVAL OF PETROGLYPH IMAGES

FROM A DATABASE USING IMAGE INFORMATION

A method of retrieving petroglyph images from a database using image information is proposed. Image preprocessing prior to retrieval is discussed, and a method for automated threshold extraction is developed. Image retrieval using a line-histogram method developed by Huet and Hancock (1998, 1998) and a centroid-radii model based on the work of Tan, Ooi, and Thiang (2000) are investigated. The centroid-radii model, when modified to accommodate missing image information and used with normalized Euclidean distance measure, was effective in retrieving similar images from a field of 437 images when the distance between images was less than 30. Preprocessing and retrieval procedures were implemented in C code. The automated threshold extraction is $O(n + m^2)$, where n is the number of pixels in the bounding box of a selected region-of-interest and m is the number of grayscale intensities in which the image is encoded; extraction of the centroid-radii distance measure is O(n), where n is the maximum dimension of a side of the bounding box, and calculation of the distance between two images in the database is $O(n^2)$ in the number of radii used to describe an image. A user survey of perceived similarity between query images and resulting retrieval sets demonstrated a direct correlation between the average distance of the retrieved image to the query image and

the number of images in the retrieval set perceived as similar to the query image. A cluster analysis performed using the distance measure showed that the most meaningful clustering of images occurs at small distances.

by

CYNTHIA G. HUYSER, B. A.

Southwest Texas State University

December 2002

SUPERVISING PROFESSOR: WILBON DAVIS

CHAPTER I

INTRODUCTION

1.1 Overview

The retrieval of petroglyph images from a database using image information poses a number of challenges. In this thesis, automated image preparation, processing, and retrieval are explored, and a cluster analysis done with the distance measure used. Human perception of retrieved versus query objects is discussed in regard to the distance measure employed in the retrieval. A second retrieval method, found to be less successful, is also explored.

1.2 The domain of petroglyph images

The domain of petroglyph images affords a variety of challenges: the color palette is limited; the images are not uniform; images are presented in a variety of scales and orientations; the images are subject to varying degrees of deterioration; lighting of images in outdoor locations varies with time of day and surface orientation. These challenges form a set of boundary conditions for the retrieval of images in this domain: the

methodology must be scale-, rotation-, and translation-invariant, and it must rely neither on image color nor a fixed amount of image contrast.

1.3 The image database

The image database used for experimentation consists of 437 areas of interest selected from 157 digitized photographs. All but one of the photographs was taken by the author at Petroglyph National Monument in Albuquerque, New Mexico in October 1997 and January 2001. Professor Wilbon Davis, the author's thesis advisor, took the remaining photograph from which query images were selected at Big Bend National Park. The photographs were scanned on a flatbed scanner and saved as 16-bit grayscale images.

1.4 Document structure

The structure of this paper is as follows: Chapter 2 provides a review of the literature pertaining to image retrieval pertinent to the domain. In Chapter 3, the avenues employed to approach this problem are Chapter. Section 4 describes preprocessing applied prior to the application of image retrieval approaches. Chapter 5 describes the outcome of the application of a line-angle histogram approach to retrieval of images in this domain. Chapter 6 describes the outcome of the application of a centroid-radii retrieval methodology to this domain. Chapter 7 discusses the development platform and performance issues. Chapter 8 describes the results of cluster analysis applied to the

centroid-radii data. In Chapter 9, human perception of retrieved images is discussed vis-à-vis the distance measure employed in the centroid-radii model. Chapter 10 discusses the relationships between image similarity, noise, and the distance measure. Chapter 11 discusses areas for further research. Chapter 12 gives conclusions. A bibliography and appendix follow.

CHAPTER II

RELATED LITERATURE

2.1 Survey of content-based image retrieval applicable to the domain

A number of rotation-, translation-, and scale-invariant image retrieval techniques that do not rely on color were explored. Incorporation of user feedback in image retrieval systems and retrieval efficiency issues were also investigated. An important overview of content-based image retrieval in the 20th century is given by Smeulders, Worring, Santini, Gupta, and Jain (2000).

Milios and Petrakis (1995) take a dynamic programming approach to shape retrieval using convex/concave segments defined at inflection points of a closed curve. Cinque and Lombardi (1995) describe a multiresolution approach to shape recognition using a diffusion process. Huet and Hancock (1998, 1999) describe content-based retrieval that indexes images with a two dimensional relational histogram based on the relative sizes of and angles between nearest-neighbor line segments, and with Cross (1999) use a Bayesian matching algorithm for similarity. Gdalyahu and Weinshall (1999) develop a syntactic representation of curves which they then apply to the problem of silhouette

matching. Abate, Nappi, Tortora, and Tucci (1999) describe an image retrieval system for CT scans in which spatial relationships between image elements and query-by-sketch are utilized.

Tan, Ooi, and Thiang (2000) describe the indexing of shapes in databases using the centroid-radii model, based on the work of Chang, Hwang, and Buehrer (1991); this model is amenable to a multiresolution retrieval approach. Abbasi, Mokhtarian, and Kittler (2000) describe an enhanced method for curvature scale space shape retrieval in the domain of images with shallow concavities. Kim and Kim (2000) develop retrieval based on the similarity distance between Zernike moments. Vailaya, Figueiredo, Jain, and Zhang (2001) introduce a Bayesian framework for image categorization. Weber and Casasent (2001) use a nonlinear combination of Gabor filters with an extended piecewise quadratic neural network for detecting objects in the presence of clutter and noise. Wu and Narasimhalu (1998) propose a fuzzy image database model in which fuzzy queries are used for retrieval.

2.2 Methods to improve retrieval efficiency

Santini and Jain (2000) describe the El Niño database system, which incorporates user ranking of image similarity. Adjeroh and Lee (2000) apply the statistical concept of occupancy to image retrieval where images are represented as multidimensional histograms. Chen, Bouman, and Dalton (2000) explore the use of hierarchical tree

structures to improve image retrieval in a large database (>10,000 images). Zhu, Ramsey, and Chen (2000) describe content-based image retrieval incorporating Gabor filters and image compression within a context of large (> 25 megabyte) file size.

CHAPTER III

AVENUES OF APPROACH

3.1 Two methods of implementation

Two methods of image retrieval were implemented and tested on the image database described above. One is the line-angle histogram approach described by Huet and Hancock (1998, 1999), the other the centroid-radii model discussed by Tan, Ooi, and Thiang (2000).

3.2 The line-angle histogram approach

Huet and Hancock (1998, 1999) define a method of image retrieval based on the line content in an image. Detected lines are inspected to determine the 6 nearest neighbors to each line in the image; the angle between each and the relative length of the lines vis-à-vis their point of intersection form the descriptors used to populate a histogram used for image retrieval. This line-angle histogram method was chosen because of the inherently discontinuous nature of the images in the petroglyph domain, where gaps in images make generation of an adequate contour difficult.

The following steps comprise the line-angle histogram described by Huet and Hancock (1999):

1. Extract lines from drawings (line extraction methodology discussed in Section 5 below).
2. Compute pairwise geometric attributes for 6 nearest neighbors (with 6 being an empirically derived number that maximizes retrieval accuracy), where the distance between two lines is calculated as the Euclidean distance between their midpoints:
    a. The relative angle between two lines is given as

$$\alpha_{ab,cd} = \arccos\left[\frac{x_{ab} \cdot x_{cd}}{|x_{ab}||x_{cd}|}\right]$$

where x_{ab} and x_{cd} are vectors denoting the line segments ab and cd, pointed away from their point of intersection.

    b. The length ratio attribute utilizes the ratio between the "oriented baseline vector x_{ab} and the vector x_{lb} joining the end (b) of the segment (ab) to the intersection of the segment (cd)." The length ratio attribute is given by:

$$v_{ab,cd} = \frac{1}{\frac{1}{2} + \frac{D_{lb}}{D_{ab}}}$$

3. Enter line information in image histogram. Histogram is 2-D structure:

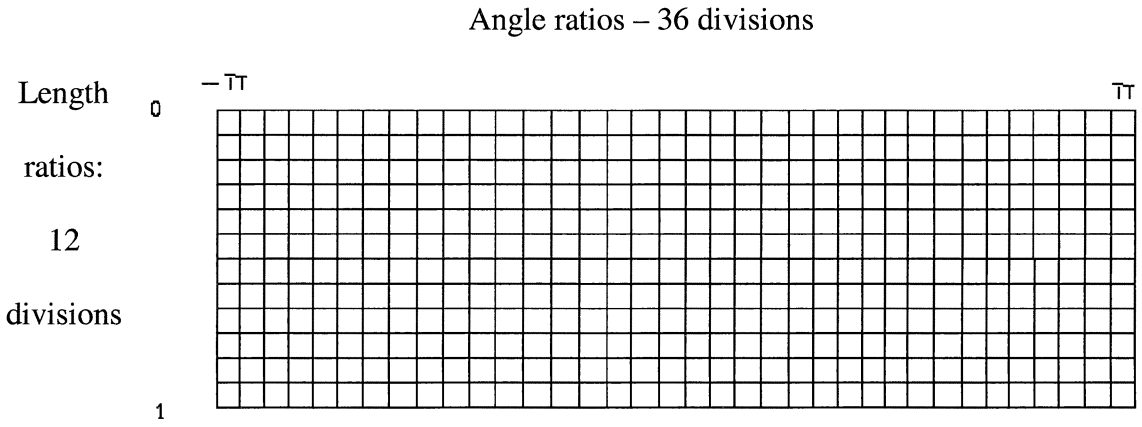


Figure 3.1: Line-Angle Histogram

As the relative length ratios and angles are computed for the 6 nearest neighbors of each line segment, the corresponding histogram bins are incremented.

4. Retrieval is based on minimum Bhattacharrya distance to the query pattern:

$$B(Q,M) = -\ln \sum_{I=1}^{n_A} \sum_{J=1}^{n_R} \sqrt{h_Q(I,J) \times h_D(I,J)}$$

To define an image's distance to itself as zero, the image histogram is normalized by dividing the number in each bin by the sum of all histogram bins, so that the sum over the histogram is 1.

Implementation details for the line-histogram method are discussed in Section 5.

3.3 The Centroid-Radii Approach

The centroid-radii model described by Tan, Ooi, and Thiang, based on the work of Chang, Hwang, and Buehrer (1991), is a contour-based approach that generates a series of descriptors by measuring the distance from the centroid of the image along a number of radii to the image contour. The number of radii used can be any power of 2 greater than or equal to 4; greater numbers of radii give better image description, while the use of numbers of radii that are powers of 2 lends itself to a multi-scale retrieval approach in which the retrieval tree can be pared by first calculating potential retrievals based on smaller numbers of radii. The centroid-radii model was chosen as a retrieval method because of the relative simplicity of its application within the petroglyph domain, and the fact that it could be modified to handle some of the discontinuities present in the domain.

The centroid-radii model described by Tan, Ooi, and Thiang (2000) is comprised of the following steps:

1. Locate the centroid of the image
2. Find the periphery of the image
3. Calculate the distance from the centroid to the periphery of the image at each of 2^n angles, where $n \geq 2$; these are the radii used for the distance measure
4. For each image, calculate the minimum distance by applying the Euclidean distance measure

$$\text{distance} = \min_{0 \leq k_{n_A - 1}} \left| \sqrt{\sum_{t=1}^{n_A} (r_{1,t} - r_{2,t+k \bmod n_A})^2} \right|$$

The sum of each image's radii is normalized to a common number for the distance calculation.

Implementation details for the centroid-radii model are discussed in Section 6.

CHAPTER IV

IMAGE PREPROCESSING

4.1 Region-of-interest selection

Each image region of interest begins as a selection by the user in GIMP. An alpha mask layer is then added to the image, and the selected area masked so that only the region of interest is visible. A black background layer is then added, and the resulting image flattened to produce a new image in which non-selected portions are black. Figures 4.1 and 4.2 below illustrate an input image and the image resulting from selection, masking and flattening.

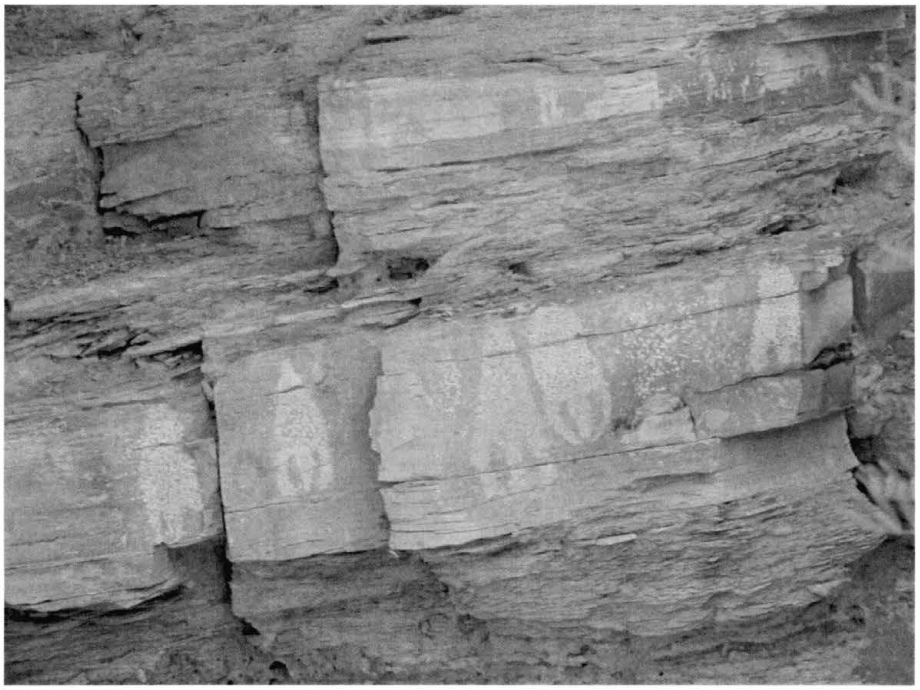


Figure 4.1: Original image, saved to 16-bit grayscale

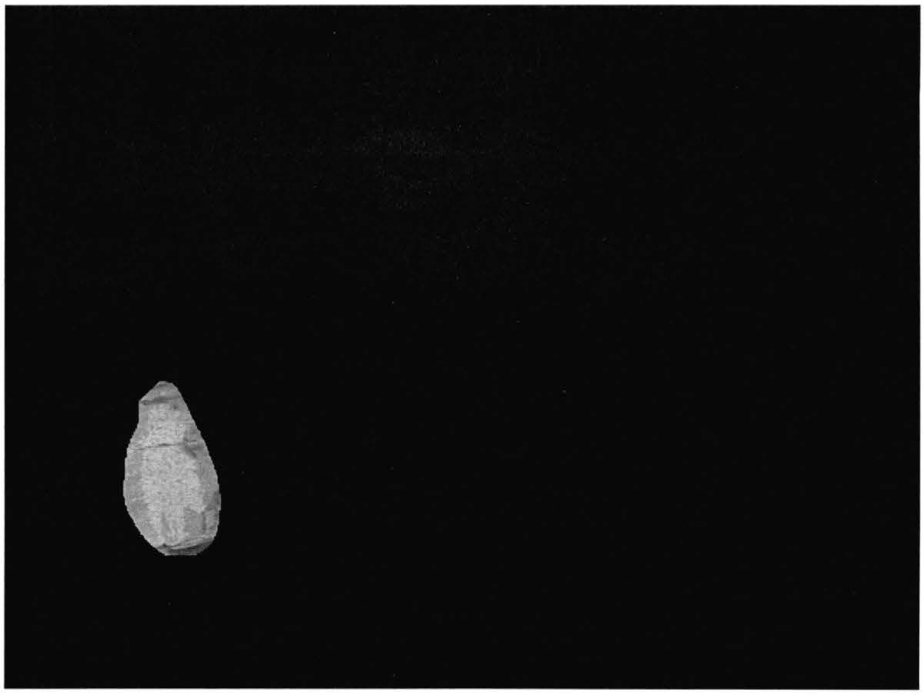


Figure 4.2: Image with selected region of interest after masking and flattening

4.2 Binarization

The image which now contains only the region of interest on a black background is then cropped using GIMP's *autocrop* plug-in, which trims the image dimensions so that the region of interest is enclosed within its bounding rectangle. The new image is then normalized to a common maximum dimension. After normalization, a Minkowski erosion is applied to the image; since the erosion operates on the dark pixels, it functions as a dilation of the pixels of interest. The image is then thresholded and blurred with a Gaussian infinite impulse response filter of radius determined by the density of the white pixels in the thresholded region of interest. Figure 4.3 below illustrates the binarized image resulting from these preprocessing operations.

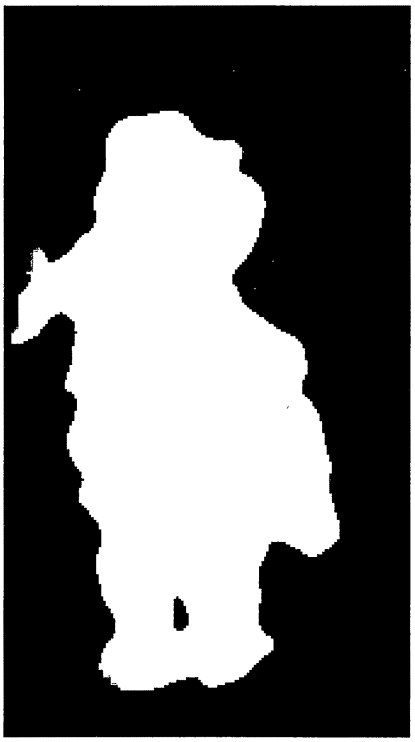


Figure 4.3: Normalized, preprocessed image used for retrieval

4.3 Automated thresholding

In automating image thresholding, histograms of pixel intensities for a number of images were studied. It was found that most histograms are multimodal, with the background pixel intensities making up the major mode. The point at which images should be thresholded was found to be at minimum occurring after the major mode, although this minimum could be quite subtle; Figure 4.4 shows the grayscale intensity histogram for the image in Figure 4.2.

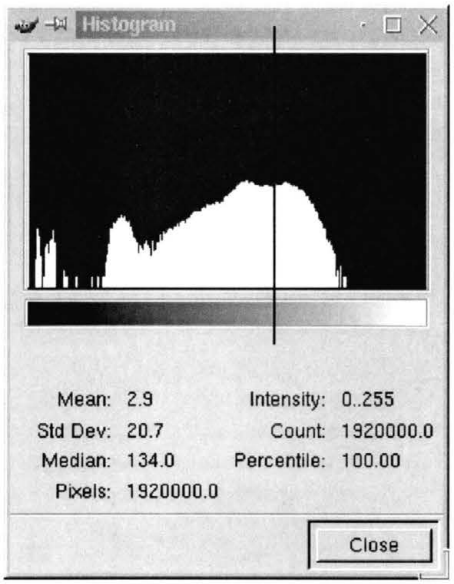


Figure 4.4: Histogram for Figure 4.2. Line marks best threshold to pick up foreground pixels.

The following algorithm was applied in determining the lower bound for the threshold:

1. Estimate the maxima and minima along the curve of the histogram of pixel intensities by averaging a window of 5 intensities to either side of a given point and using the difference between the averages as a proxy for the slope. While averaging, keep track of the maximum average intensity (this is the major mode).
2. Do a first pass noise removal for maxima and minima by discarding those maximum/minimum pairs that are within a window length (5 points) in distance. If there are the same number of minima and maxima, they are compared at corresponding indices; if there is an offset between the number of maxima and minima, each is compared in turn from an alignment of the top index to an alignment of the bottom index.
3. For the remaining maxima and minima, remove any pairs where they differ by window/2 points (each minimum is compared against each maximum).
4. Set the lower threshold to the first minimum occurring after the maximum major mode, if it exists. If no such maximum exists, choose the minimum after the first remaining maximum. Calculate the number of pixels that would remain given the application of this threshold. If fewer than 10% of the pixels would be white, adjust the threshold downward until 10% would be white.
5. Apply a Gaussian IIR blur of radius 6 to 12 to the image, with radius chosen by density of white pixels in image (greater density = larger blur radius), then apply the threshold determined above.

The upper bound of the threshold was taken to be 255. Figure 4.5 illustrates the estimation of tangent to the intensity histogram.

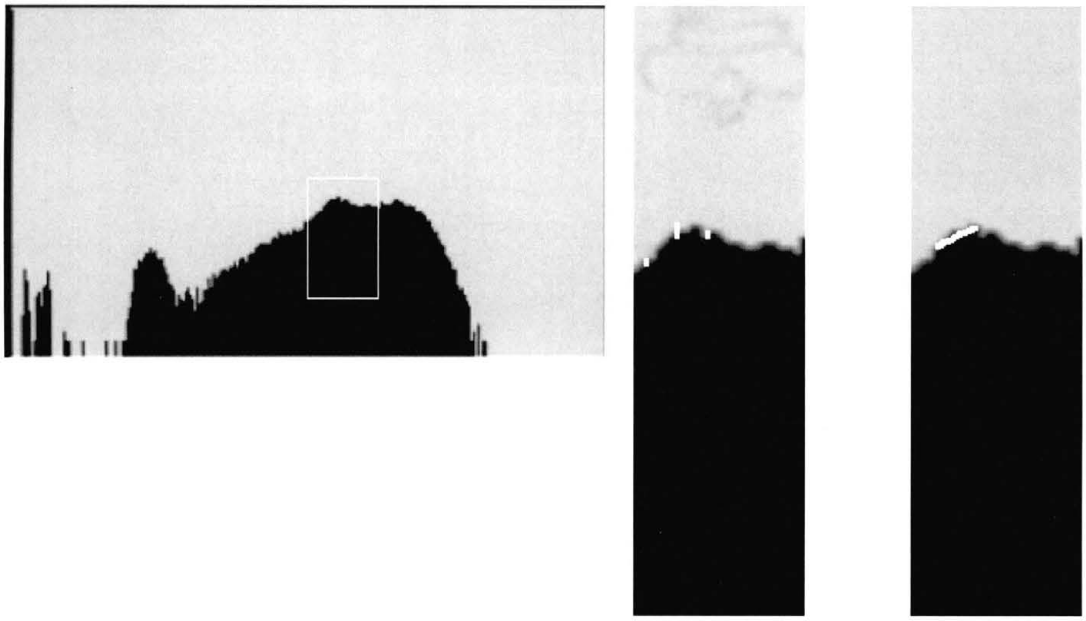


Figure 4.5: Histogram for Figure 4.2 (color inverted). Left: white box shows selected area for tangent demonstration. Center: intervals around center point where tangent will be estimated. Right: tangent estimation as interpolated line between means of left and right intervals, showing positive slope

Adjusting the size of the blur radius in direct proportion to density of high intensity pixels preserved most important image lines while reducing noise, while thresholding from a minimum point in the threshold contour directly following the maximum mode preserved an adequate amount of image information for most images. Figure 4.6 shows standard output during thresholding.

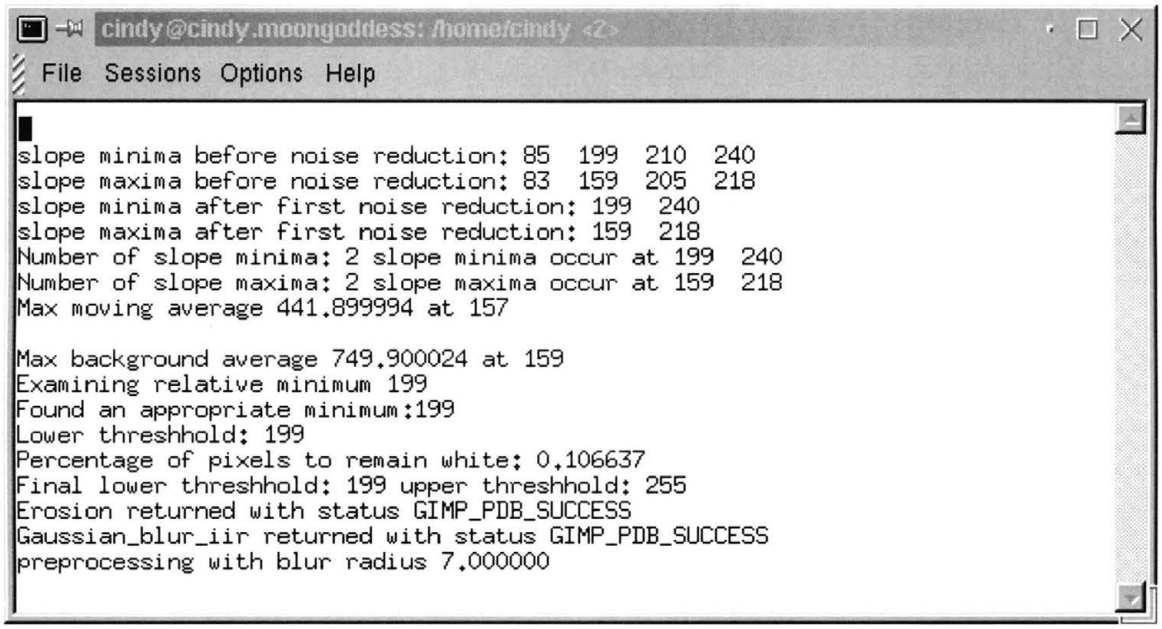


Figure 4.6: Screen shot of output from automated thresholding, showing removal of extraneous minima and maxima, calculation of threshold point, and choice of blur radius.

CHAPTER V

APPLICATION OF THE LINE-ANGLE HISTOGRAM APPROACH

5.1 Line segment recognition and calculation of line-angle histogram attributes

The first step of the line-angle histogram approach is line segment recognition; prior to taking this step, each image is preprocessed as described above. A series of 10 5 x 5 convolution masks as described in Sonka, Hlavac, and Boyle (1999) is then used to identify line segments in the images; Figure 5.1 shows one such matrix.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & 0 \\ 0 & -1 & 2 & 2 & 0 \\ 0 & 2 & -1 & -1 & 0 \\ 0 & -1 & 0 & 0 & 0 \end{bmatrix}$$

Figure 5.1: Sample line-detection convolution matrix

To adjust for incompleteness in lines due to noise, collinear lines with gaps smaller than 5 pixels in length are joined. The resulting lines in the images are then compared to find the six nearest-neighbors, their points of intersection, relative lengths and included angles; the histogram described in Section 3 above is populated with this information. The distance between two lines is calculated as the Euclidean distance between their

midpoints. Figure 5.2 illustrates the calculation of relative line lengths and the angle between them.

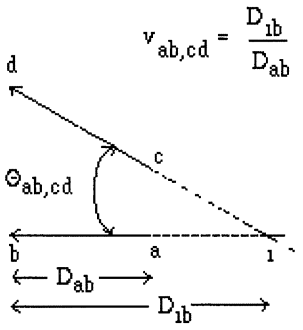


Figure 5.2: Calculation of line-angle histogram attributes

Once all relative line length – angle sets of neighbors have been determined, the distance between each of the images is calculated using the Bhattacharrya distance on normalized histograms. Each histogram is normalized by dividing the contents of its bins by the sum of the bin contents over the histogram, so that the Bhattacharrya distance from an image to itself is 0.

5.2 Retrieval results

The retrievals using this measure were less than satisfactory. A rough hand-categorization of images was used to gauge the effectiveness of the retrieval; for the preponderance of image types, the number of similar images retrieved was not significantly greater than their (random) incidence within the database, and a handful of images were retrieved by almost every query.

The failure of this method can be explained by the fact that the line-angle histograms for nearly all the images were sparsely populated, with zeroes in at least 75% of the bins. The Bhattacharrya distance measure favors well-populated line-angle matrices, since bins containing zeroes essentially drop out of the distance measure. The few images that were retrieved consistently had better-populated line-angle histograms.

CHAPTER VI

APPLICATION OF THE CENTROID-RADII APPROACH

6.1 Centroid-radii calculation

As in the case of the line-histogram method, some preprocessing was applied to enhance the measurements made with this method. Once the preprocessing is completed, the centroid-radii information is calculated using the following algorithm:

1. Find the bounding rectangle for the image (first row, last row, leftmost, rightmost): n x m
2. Initialize two arrays of length n (for the left and right edges) and two arrays of length m (for top and bottom edges) with the value –1, standing for "uninitialized."
3. Map the left and right edges into the arrays from Step 2 by scanning the image by rows to find the leftmost and rightmost image pixels with a value of 255, and storing the corresponding column number in the edge array; scan by columns for topmost and bottommost white pixels, and store the corresponding row numbers in the top and bottom edge arrays. When the mapping is complete, array entries with a value of –1 indicate a gap in the image contour.

4. The edge mapping contains some redundancy, as some edge pixels will also be stored in the adjacent edge array. Figure 6.1 illustrates the location of redundant edge points in a sample petroglyph contour.

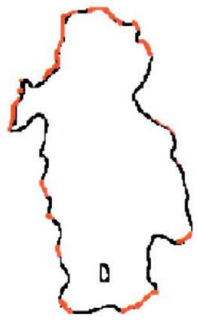


Figure 6.1: Redundant Edge Points Stored in Adjacent Edges (red)

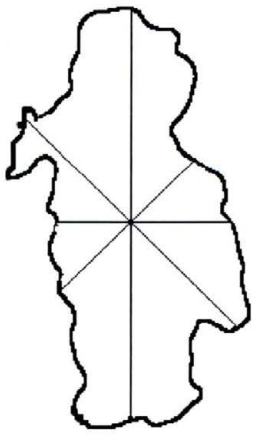
Calculate the mean of all unique edge points; this is the preliminary centroid of the image.

5. If any points are uninitialized,
    a. find the length of the uninitialized segment
    b. if the length of the uninitialized segment is 15 or fewer pixels, fill in the missing pixels with a linear interpolation of the existing points on either side of the gap; leave gaps of 15 or more pixels uninitialized
6. Recalculate the centroid, using the interpolated points as well as the original unique points. Comparing the recalculated centroid to the initial centroid acts as a "sanity check" regarding the effects of gap interpolation on centroid calculation.
7. Each radius will pass through the centroid (p, q) so that any point that intersects the edge of the image contour must satisfy (to within 1 pixel) the equation

$$y - q = s(x - p), s = \tan(\theta)$$

where θ is the angle of the radius passing through the centroid.

Lengths of radii for angles 0, $\frac{\pi}{2}, \pi,$ and $\frac{3}{2}\pi$ can be determined directly; for each of the other angles, search the appropriate quadrant (two adjacent sides) for a point that satisfies the above equation within one unit (since the image contour pixels are always integer values, unlike the calculated intersection values). If more than one such intersection point is encountered, the one with the maximum distance from the centroid is used for the radius calculation; if no such point is found, the intersection is marked "empty". Figure 6.2 illustrates a case in which all radius-contour intersections were found.



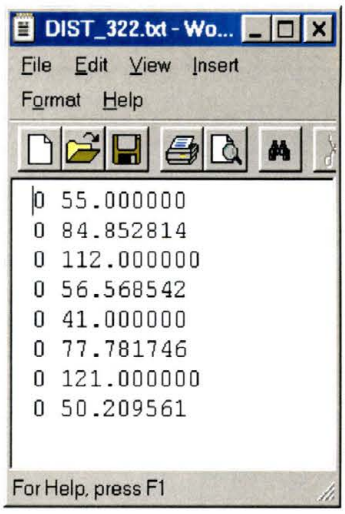


Figure 6.2: Centroid-radii view with complete contour and 8 radii. Each entry in left column of distance file is 0, meaning "not empty"; right column entries are distances.

As mentioned in describing the measurement of radii, there are images in which one or more of the radius measures were missing, either due to a gap in the image or contour not lying in the direction of the radius from the centroid. Figure 6.3 illustrates such an image. Additionally, the method used to determine intersection of a radius with the image contour fails to detect intersection when it would occur in the interpolated space between adjacent contour pixels; this is an area for improvement in the implementation.

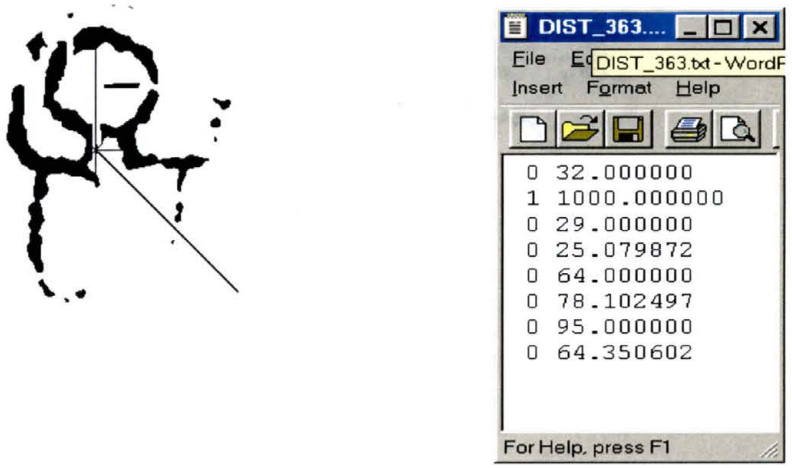


Figure 6.3: Illustration of an image with centroid-radius gaps. Note "1" in left column of distance file marking the empty radius bin.

6.2 Generating retrievals using centroid-radii information

Once all image centroid-radii lengths have been calculated, retrievals are determined by calculating the minimum distances between an image and each other image in the database and using the closer of these images as the retrievals for the given image. In the current implementation, 64 radii are calculated for each image. The differences in the lengths of each image's radii to the other images' corresponding radii are calculated; all

such differences resulting from the rotation of one image with respect to another are calculated, and the minimum distance taken as the distance between the images. The formula used for the calculation is:

$$\text{distance} = \min_{0 \le k_{n_A-1}} \left| \sqrt{\sum_{t=1}^{n_A} (r_{1,t} - r_{2,t+k \bmod n_A})^2} \right|$$

where k is an offset that varies from 0 to $n_A - 1$. The distances are in fact normalized to 1000:

$$r_{1,t} = \frac{r_t * 1000}{\sum r_t}$$

Since 64 radii are used, this normalization leads to an average radius of 15.625, so that differences in radii of more than roughly 7% should be magnified in the distance measure.

In deriving the distance between two images, the number of radii in corresponding positions at each rotation is counted; if fewer than half the available radius measurements can be compared, it is assumed that not enough information is known to judge whether the two images are similar, and the distance at that rotation is set arbitrarily high (500). If half or more of the radii can be compared, then those available for comparison are used to derive the distance between the images, with no penalty for missing information.

6.3 Retrieval results

The retrievals using this method were much more satisfactory than those using the line-angle histogram method. For most images, 15 to 50% of the closest 20 retrievals (of a possible field of 436) resembled the query image; in a number of cases, when an image existed at different resolutions within the database, the first image retrieved by similarity was the corresponding image at the different resolution. Of the 437 images, 21 retrieved no similar image, and 51 images were never found in the retrieval sets of other images. Figure 6.4 shows the frequency with which images in the database were retrieved in the set of top 20 retrievals made by all images in the database. Figures 6.5a, 6.5b, and 6.5c show sample retrievals.

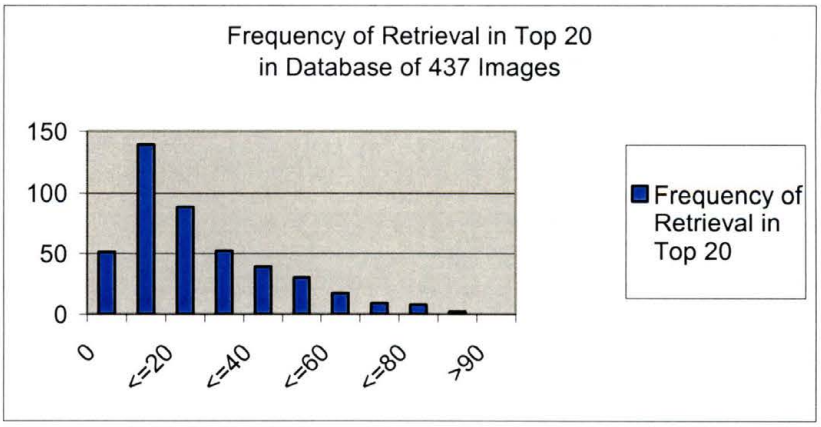


Figure 6.4: Image Frequency of Retrieval in Top 20

SAMPLE RETRIEVALS

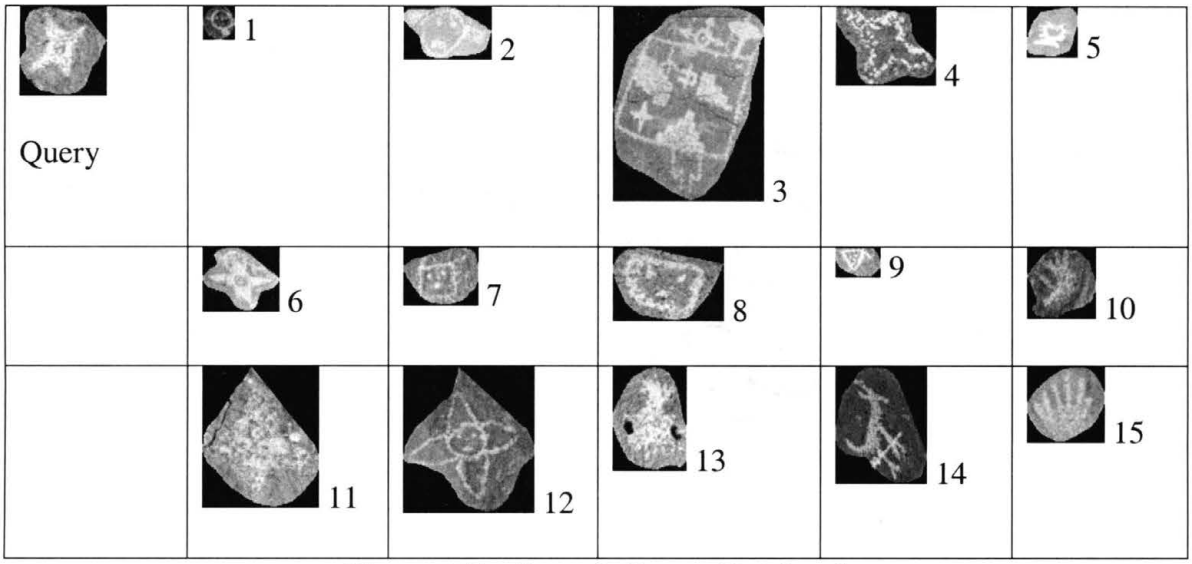


Figure 6.5a: Sample Image Retrieval

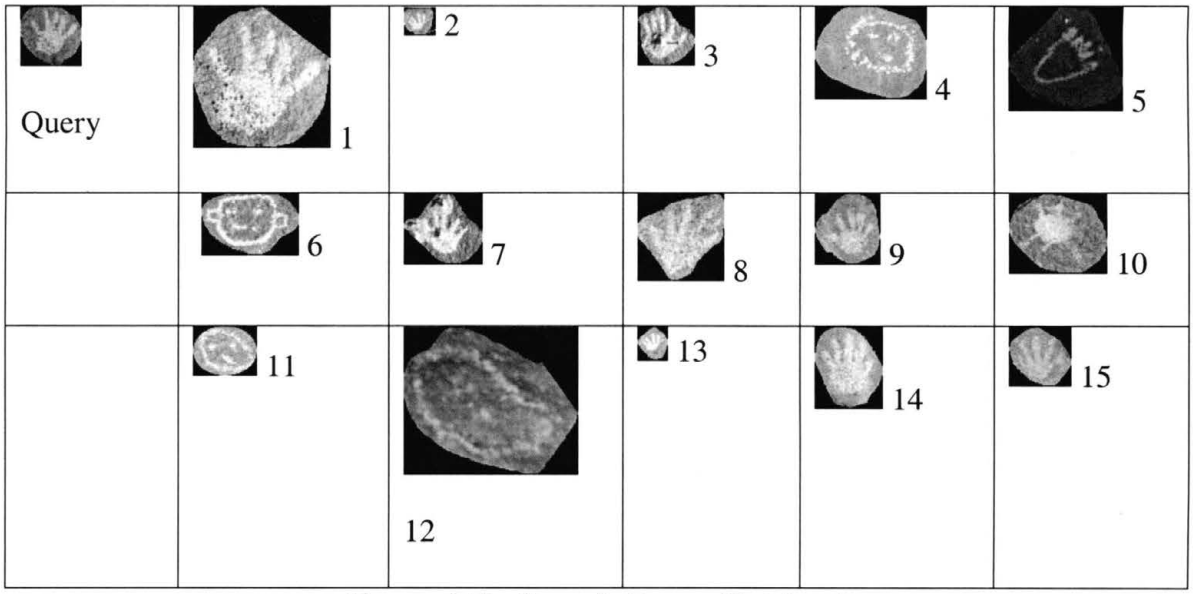


Figure 6.5b: Sample Image Retrieval

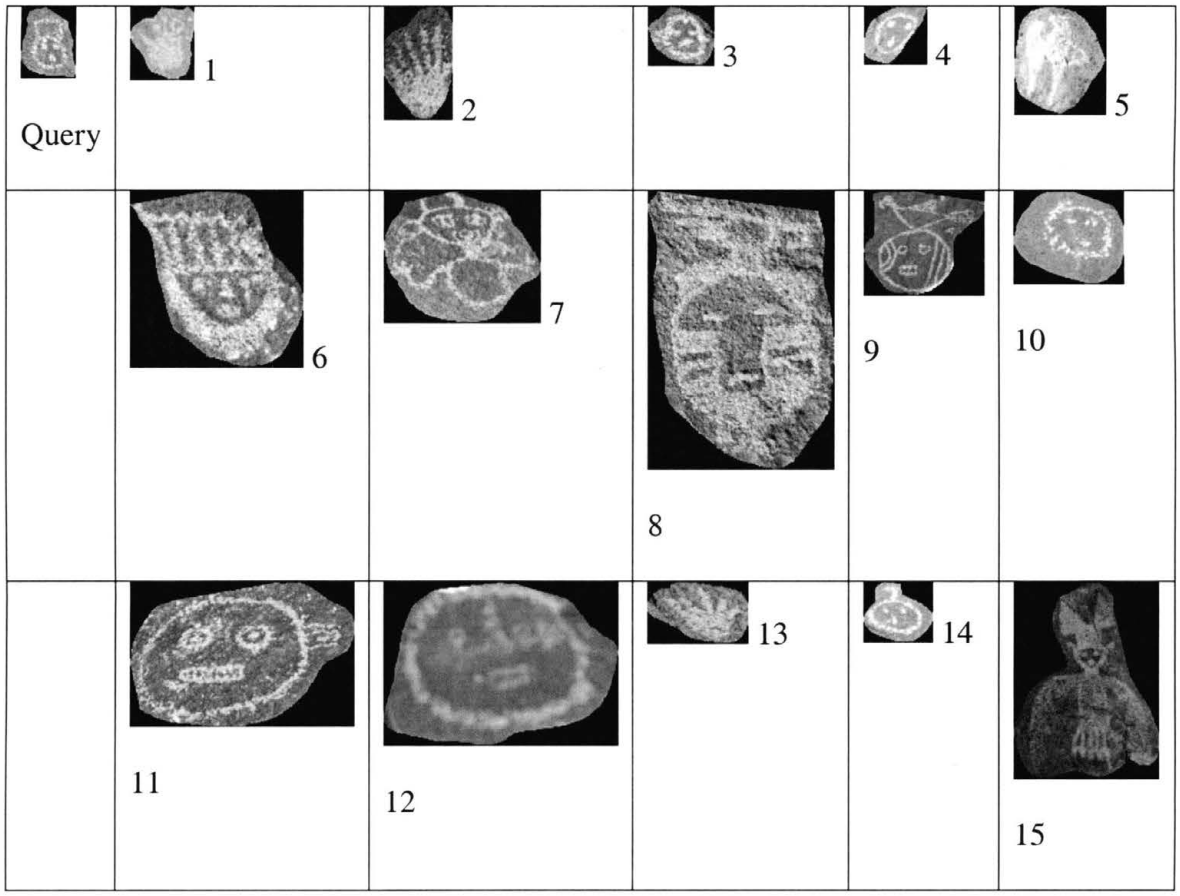


Figure 6.5c: Sample Image Retrieval

CHAPTER VII

PERFORMANCE

The algorithms described above were implemented in the C programming language, primarily as plug-ins to GIMP (the GNU Image Manipulation Program). Automated thresholding, centroid-radii calculation, line-detection and line-angle histogram formation were all accomplished in this plug-in format, while distance calculations, retrieval generation and cluster analysis were written as stand-alone programs. Postgresql was used for the image database.

Bulk image processing and calculation of centroid-radii was orchestrated via PyGimp python scripts, allowing all 437 images in the database to be processed in single script runs.

The machine on which the programs were tested contains a single 677 MHz Pentium-III processor and 256 MB of RAM running Linux Red Hat 6.2.

Program Name	Average Run Time (s)	Notes
centroid.pgc	0.04087 *	time per image processed
calculate_c_distance.c	66.08	calculation of distances between all images in database. Includes run time for b_dist3.c
find_thresh.c	0.05638 *	calls erode_matrix.c

Table 7.1: Instrumented Average Run Times

* Called plug-ins and scripting add a significant amount of time to this figure, generated by calling clock()/CLOCKS_PER_SEC just prior to program termination.

Program Name	O-notation
centroid.pgc	O(n) in number of pixels
calculate_c_distance.c	$O(n^2)$ in number of database images
b_dist.c	$O(n^2)$ in number of radii
cluster_analysis.pgc	$O(n^2)$ in number of database images
find_thresh.c	O(n) in number of pixels
erode_matrix.c	O(n) in number of pixels

Table 7.2: O-notation Performance

CHAPTER VIII

CLUSTER ANALYSIS OF RETRIEVALS VIA CENTROID-RADII APPROACH

8.1 Cluster analysis methodology

A cluster analysis of the retrievals done via the centroid-radii approach was made using a centroid distance measure for the clustering.

Centroid Clustering Algorithm:

```
initialize clusters to each contain a single image
while (more than one cluster)
    find the clusters closest in distance
    merge the clusters by averaging their radii in the alignment that
      produced the minimum distance
```

8.2 Cluster formation and distribution

The progress of cluster formation is shown below in Figure 8.1. The maximum distance required to subsume a viable singleton cluster (i.e., an image with at least half

the centroid-radii points available for distance measure) was 170; agglomeration of clusters continued until all viable clusters were clustered together (distance: 494) and the non-viable clusters began to be subsumed. Table A1, containing details regarding cluster distribution, may be found in the Appendix.

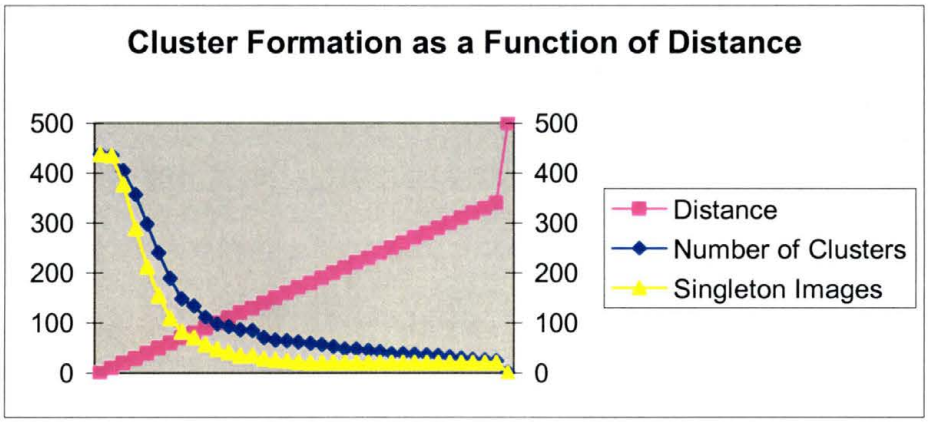


Figure 8.1: Cluster Formation as a Function of Distance

Table A2 of the Appendix illustrates clustering of singleton images at distances below 80. Most clustering of singleton images with strong similarity occurs at distances below 40. By the time the minimum clustering distance has reached 60, 75% of all singleton images have been clustered with at least one other image.

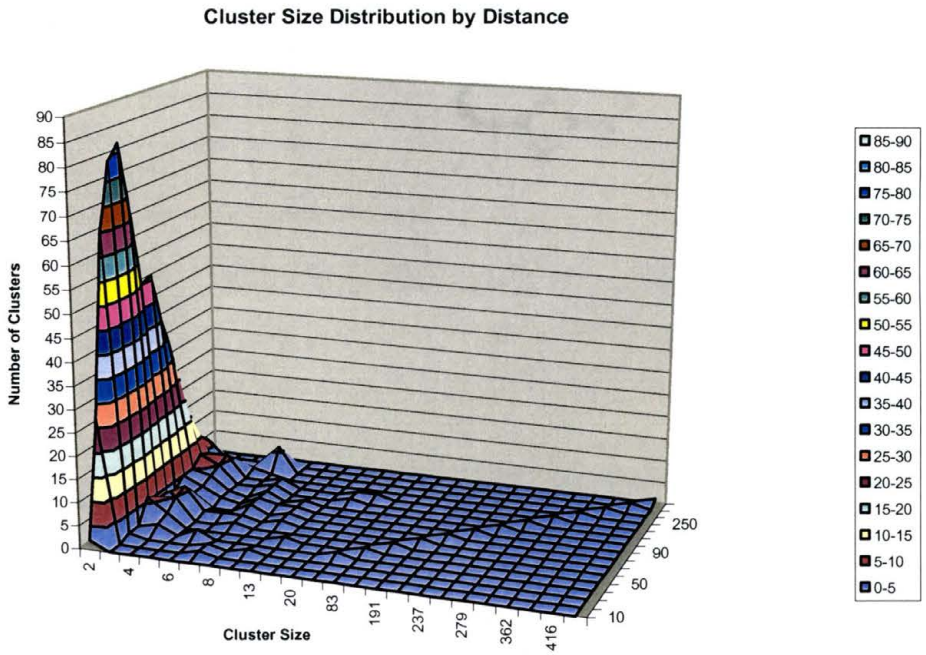


Figure 8.2: Cluster Size Distribution by Agglomeration Distance

In the agglomerative clustering that takes place when distance is 20 or greater, a single large cluster is favored. As can be seen in Figure 8.2, a number of clusters of intermediate size form when during the clustering process, but one large cluster dominates. For example, at distance of 50, the largest cluster contains 83 images; the next largest contains 13. Images whose contours approximate a circle dominate early agglomerative cluster formation. Figures 8.3 and 8.4 illustrate relatively large clusters at distances below 30.

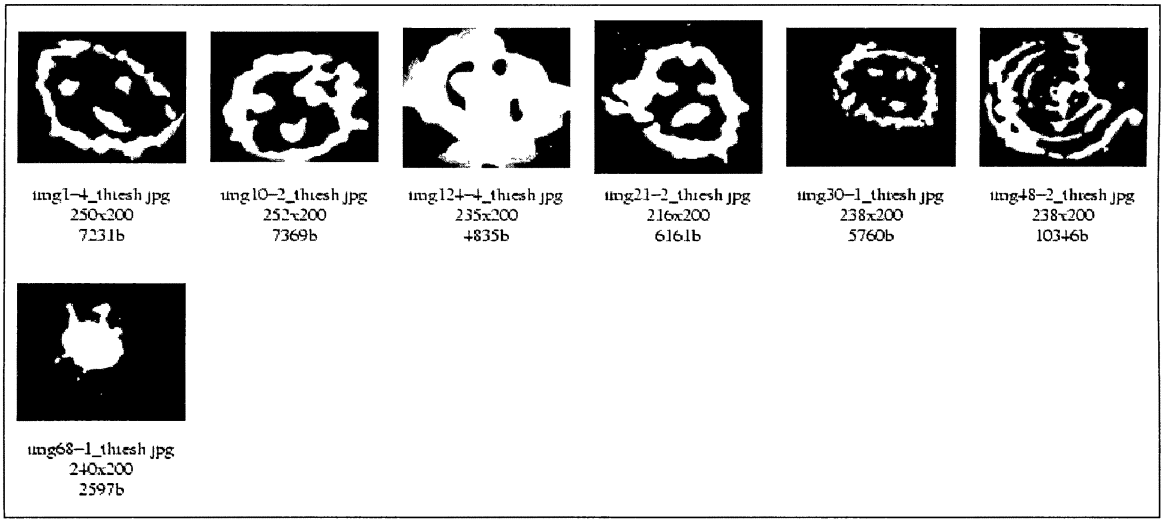


Figure 8.3: Largest Cluster at Distance 20.006056

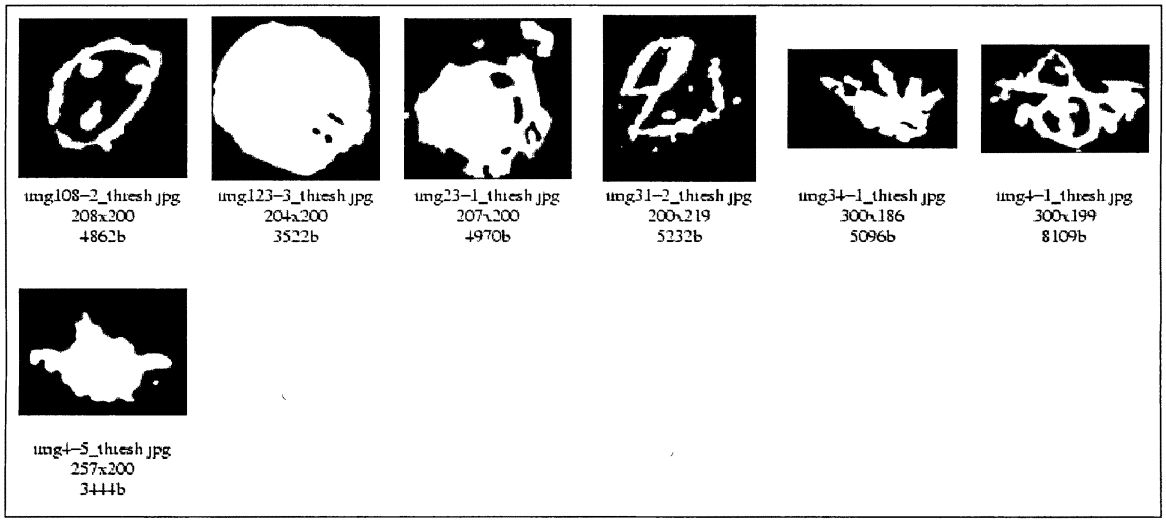


Figure 8.4: Second Largest Cluster at Distance 29.25824

The means of obtaining the centroid of images in a cluster encourages some growth of distance between clusters. As images with gaps are subsumed, "empty" radius entries are replaced by available corresponding non-zero entries; since the "empty" entry contributes nothing toward the distance measure, a newly-formed cluster will almost certainly have a

greater distance from other clusters than a subsumed cluster that contained an "empty" radius entry.

The most meaningful clustering appears at the smallest distances. As the distance overcome in cluster agglomeration increases, the disparity in clustered images increases. Figure 8.5 illustrates the heterogeneous nature of the largest cluster with increase in distance.

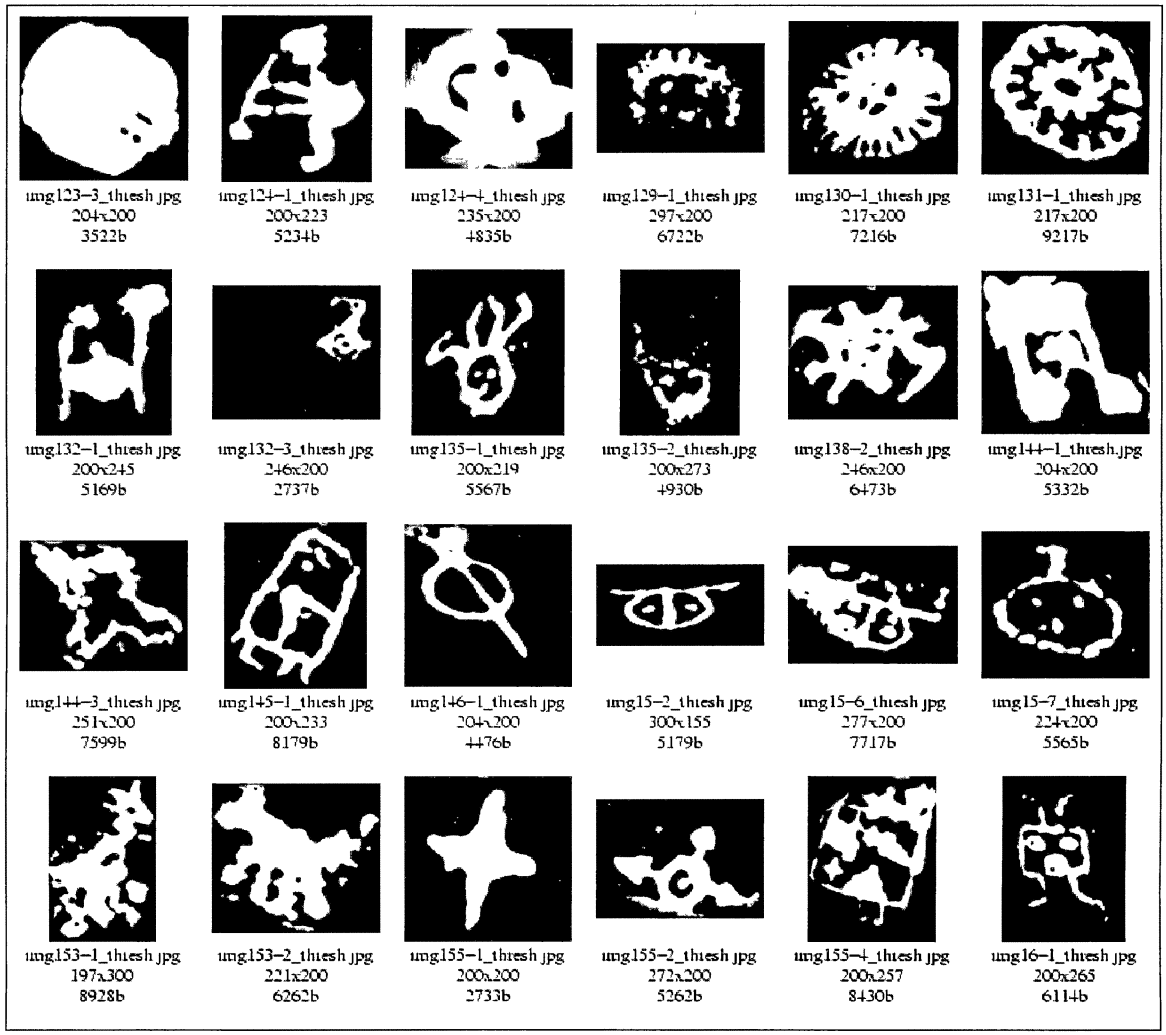


Figure 8.5: Heterogeneous Clustering At Distance 58.929016 (24 of 152 images in cluster shown)

CHAPTER IX

HUMAN PERCEPTION OF RETRIEVALS AND THE DISTANCE MEASURE

9.1 User survey

In order to measure the relationship between the distance measure implemented and human perception of its effectiveness, a survey was used in which users ranked retrievals of preprocessed petroglyph images as to their perceived closeness to the query image. For a series of 20 image retrievals, each user was presented a page with the query image and its top 10 retrievals from the database, and was asked to rank the retrieved images' perceived closeness to the query image on a scale of 1 to 10. Users were also asked to indicate how many of the retrieved images seemed to resemble the query image. Figure 9.1 shows a single page from the survey; a complete copy of the survey used can be found in the Appendix.

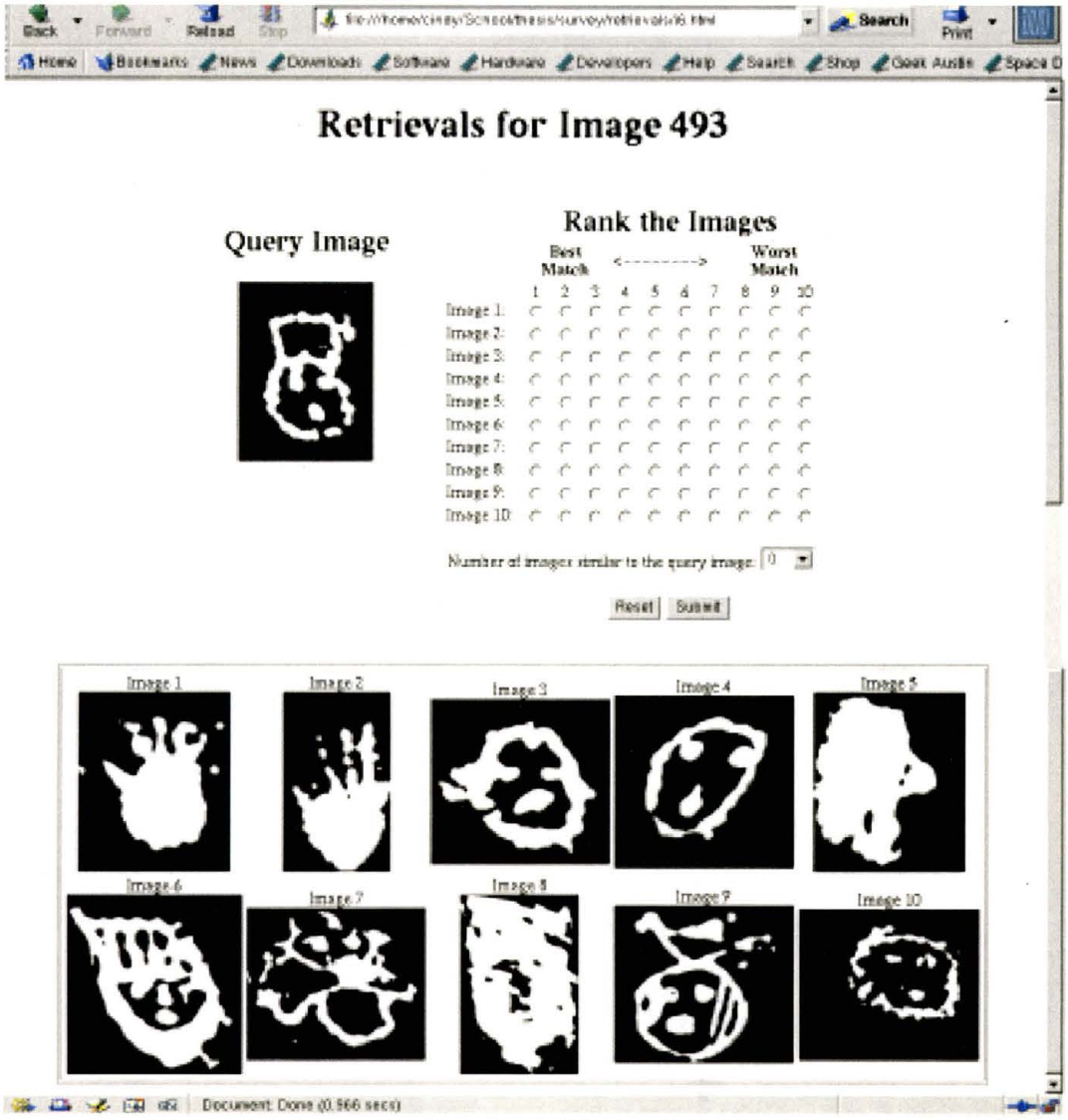


Figure 9.1: Sample page from Survey

9.2 Survey results

The number of images perceived as similar to the query by the survey takers ranged from 0 to 8. Some users estimations were decidedly pessimistic, with more than half of the sample set judged as having no similar retrievals; most users perceived all images as

having at least one similar retrieval and on average at least two retrieval sets were perceived as containing 5 or 6 similar images. Figure 9.2 illustrates the survey results; the image numbers are the auto-generated index numbers that identify the image in the database.

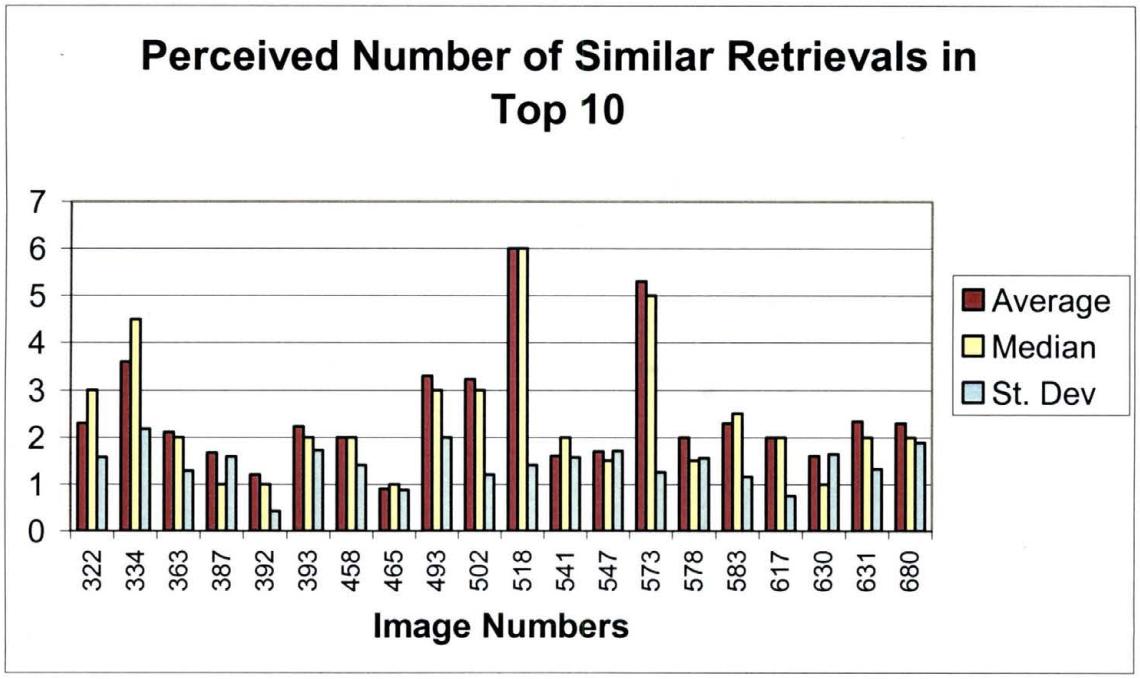


Figure 9.2: User-perceived similarity to query image in first 10 retrievals

An examination of the distances of the retrieved images to the query images shows a direct correlation between the average image distance and the perceived number of retrievals matching the query. Figure 9.3 shows that the perceived number of matches in the retrieval set is inversely proportional to the average distance of the retrieved image, verifying the distance measure's performance in retrieving similar images.

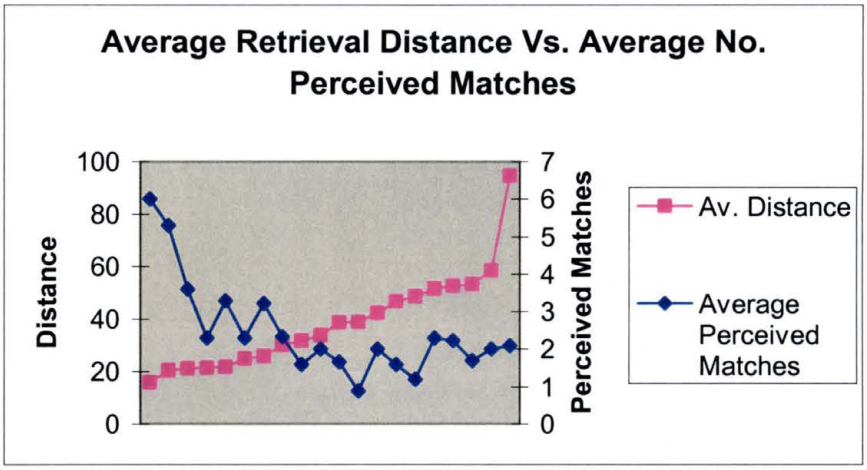


Figure 9.3: Average Retrieval Distance Vs. Average Number of Perceived Matches (from set of top 10 retrievals from query image)

CHAPTER X

IMAGE SIMILARITY, NOISE, AND THE DISTANCE MEASURE

As illustrated by the results of the survey, it is clear that the best retrieval results occur when the distance between the query object and the potential retrieval is less than 30. There are instances, however, in which the image that is closest to the query image is at a distance of greater than 30. Figure 10.1 shows one such case.

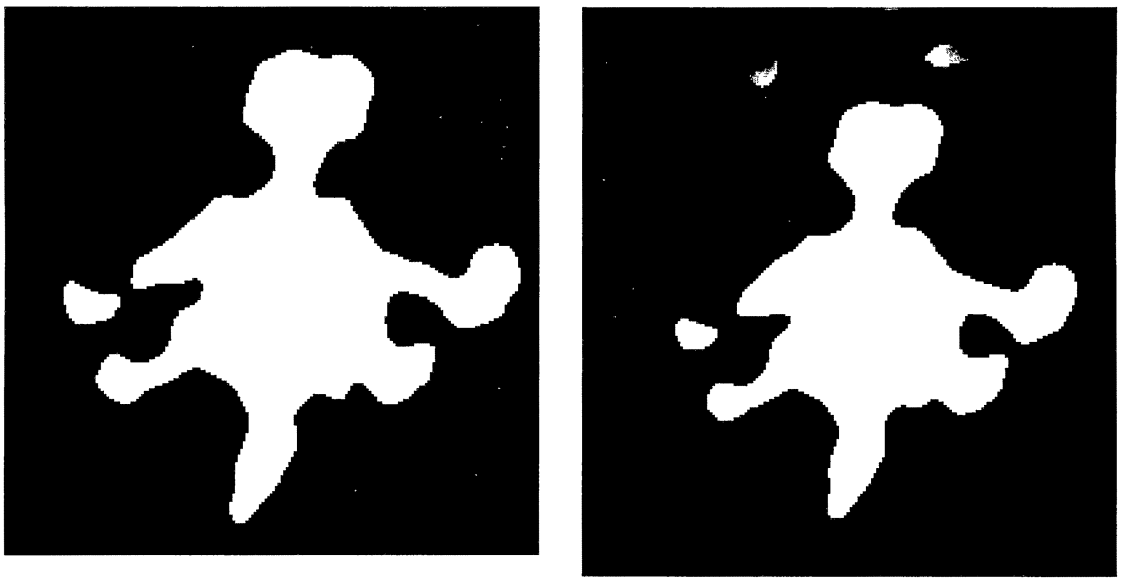


Figure 10.1: Query/Retrieval Pair at Distance of 32.52247

The noise that appears at the top of the right figure is likely responsible for the relatively large distance between these obviously similar images.

One approach to reducing the influence of such noise would be to introduce a statistical analysis of the contour that would recognize and ignore edge points when there are other, closer points that more closely follow the direction and curvature of adjacent portions of the image. Such an approach would also be useful in better describing those images in which a gap on one side of the image causes an inaccurate interpretation of the best contour on that side.

Another approach to improving the association of such images would be to introduce a function to allow the user to reorder the contents of a retrieval set to better describe the relationships between the query and retrieved images. The application of a negative weight to images proposed by the user as closer to the query image than their original retrieval position would suggest would have the desired effect.

CHAPTER XI

AREAS FOR FURTHER RESEARCH

11.1 Improvements to petroglyph retrieval

While the centroid-radii model is effective in retrieving a number of closely related images in the petroglyph domain, its retrieval is strictly contour-based; a more robust method might include image information located within the contour. This centroid-radii model is scale, translation, and rotation independent, but it does not detect mirror images; the addition of a mirror-image measurement in the image-to-image distance calculation would be an appropriate enhancement.

While the implementation of the centroid-radii model demonstrated here does to some extent address the issue of contour gaps, a method for detecting and closing gaps that occur on one side of an image (rather than across the image) should be explored. The current implementation localizes detection of the intersection of the image contour with centroid radii to a one-pixel neighborhood of contour pixels, and would be improved by checking for contour-radius intersections occurring in the interpolated area between pixels. The centroid-radii model demonstrated here could also be improved by the

development of a weighting scheme that would respond to user-directed reordering of an image in the retrieval set to ensure an improved retrieval for the image. The usefulness of the centroid-radii measurements as histograms, rather than plots, should also be investigated.

11.2 Improvements to binarization

Although the automated thresholding procedure implemented here was effective in binarizing most of the images in the database in a meaningful way, a few images suffered the loss of most of their useful information. An iterative process that adjusts threshold until at least half the available radii intersect with the image contour would likely remedy loss of information via inappropriately-high threshold setting; a means of setting a meaningful upper threshold limit (instead of fixing it at 255) should also be explored.

CHAPTER XII

CONCLUSIONS

The retrieval of images from the domain of petroglyphs is a difficult problem. Promising methods of retrieval must be scale-, rotation-, and translation-invariant, must be robust to noise, and must rely on image intensity rather than color.

Having tested the line-angle histogram method of image retrieval given by Huet and Hancock (1999) and the centroid-radii method described by Kian-Lee, Ooi, and Thiang (2000) on a database of 437 petroglyph images, we conclude that the centroid-radii method is the more effective of the two. The Euclidean distance measure employed with this method produced effective retrievals, particularly for distances below 30; a user survey of retrieval perceptions verified this by demonstrating an inverse relationship between number of perceived matches in a retrieval set and the distances from the query image to the retrieved images.

Preparatory to applying these methods, the images were binarized using a combination of Minkowski erosion (i.e., dilation of high-intensity pixels), the addition of Gaussian IIR blur and thresholding. Adjusting the size of the blur radius in direct proportion to density

of high intensity pixels preserved most important image lines while reducing noise, while thresholding from a minimum point in the threshold contour directly following the maximum mode preserved an adequate amount of image information for most images.

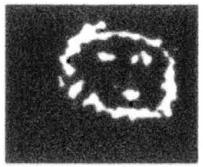
APPENDIX A

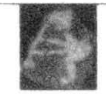
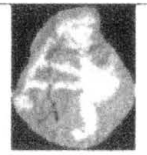
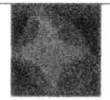
1. Table A1: Cluster Size Distribution
2. Table A2: Examples of Clustering Singleton Images

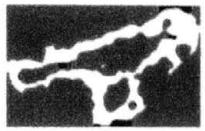
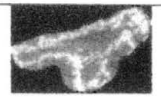
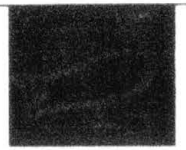
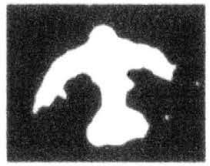
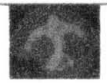
	Cluster																								
Distance	2	3	4	5	6	7	8	9	13	17	20	41	83	152	191	214	237	250	279	28	362	364	416	437	
10	2																								
20	27					1																			
30	65	1				1		1																	
40	79	1	2		2							1													
50	82	6	3						1				1												
60	67	4	5			1								1											
70	50	4	6	1	1	1		1							1										
80	51	3	3			2		1								1									
90	40	5	3	1		2	1	1									1								
100	32	5	4	1		2		1										1							
150	20	7	5	1	1	2				1									1						
200	12	6	6	1	1	2				1	1									1					
250	8		3	2	1																1				
300	6	2	2	2	6																	1			
340	0																						1		
500																								1	

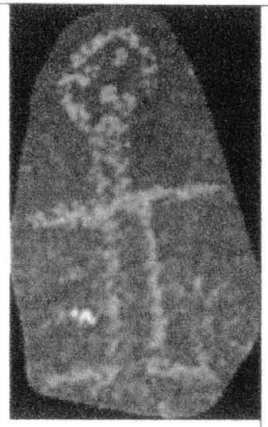
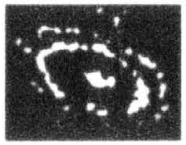
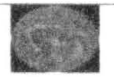
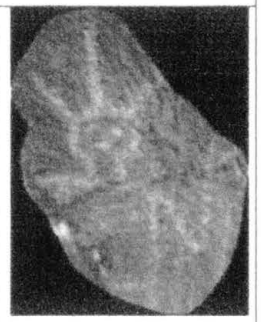
Table A1 -- Cluster Size Distribution

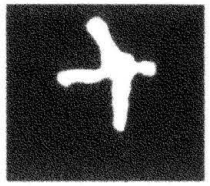
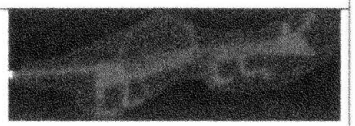
Table A2: Examples of Clustering Singletons

Distance	Example	Original (left)	Original (right)
9.105599	 img24-2_thresh.jpg 200x233 3395b  img52-2_thresh.jpg 227x200 3141b		
10.674561	 img21-2_thresh.jpg 216x200 6161b  img30-1_thresh.jpg 238x200 5760b		

16.106159	 img45-1_thresh.jpg 204x200 4445b	 img53-2_thresh.jpg 200x223 4141b		
18.553049	 img124-1_thresh.jpg 200x223 5234b	 img6-1_thresh.jpg 200x230 5269b		
28.013758	 img155-1_thresh.jpg 200x200 2733b	 img91-1_thresh.jpg 200x200 3369b		

27.173050	 img87-3_thresh.jpg 300x189 8247b	 img90-1_thresh.jpg 240x200 7047b		
34.648552	 img69-1_thresh.jpg 153x300 3750b	 img88-1_thresh.jpg 248x200 4320b		
35.853813	 img18-5_thresh.jpg 300x113 3945b	 img91-3_thresh.jpg 148x300 6282b		

43.873907	 img120-1_thresh.jpg 184x300 8355b	 img127-1_thresh.jpg 160x300 5069b		
40.832878	 img1-6_thresh.jpg 260x200 7888b	 img116-1_thresh.jpg 200x255 804+b		

58.096222	 img106-3_thresh.jpg 227x200 2721b	 img6-2_thresh.jpg 200x273 4240b		
64.276283	 img155-3_thresh.jpg 300x88 4394b	 img81-3_thresh.jpg 145x300 5251b		
76.085603	 img119-1_thresh.jpg 189x300 6211b	 img73-2_thresh.jpg 300x99 3983b		

APPENDIX B

1. Sample Image Retrieval Survey
2. Spreadsheet: Image Retrieval Survey Results

Image Retrieval Accuracy Survey

Introduction

This survey is part of a Master's Thesis in Computer Science in which the goal is to retrieve petroglyph images from a database using only the information in the images themselves. The images, which have different contrast levels and significant amounts of noise, are preprocessed before they are retrieved from the database.

The purpose of this survey is to estimate the efficacy of the image retrieval method used by having survey participants look at the preprocessed query images and their retrievals and rank those retrievals.

Doing the Survey

10 retrieved images are presented for each query image; each should be ranked as to the order in which the survey participant thinks the image should have been retrieved, with "1" indicating it is the closest image to the query, and "10" indicating the farthest. Additionally, survey participants are asked to indicate what number of the 10 images presented appear to be similar to the query image.

Thank you for participating in this survey!

If you have any questions, please don't hesitate to contact me:

Cindy Huyser
447-7752
chuyser@io.com

Thesis advisor:

Wilbon Davis, Professor of Computer Science
Southwest Texas State University

Retrievals for Image 322

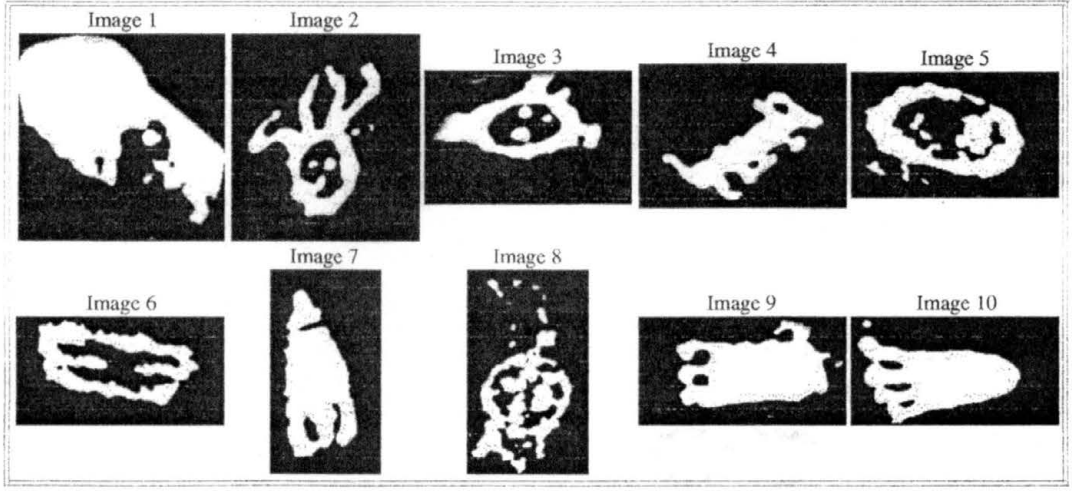
Query Image

Rank the Images

	Best Match 1	2	3	4	5	6	7	8	9	Worst Match 10
Image 1:	○	○	○	○	○	○	○	○	○	○
Image 2:	○	○	○	○	○	○	○	○	○	○
Image 3:	○	○	○	○	○	○	○	○	○	○
Image 4:	○	○	○	○	○	○	○	○	○	○
Image 5:	○	○	○	○	○	○	○	○	○	○
Image 6:	○	○	○	○	○	○	○	○	○	○
Image 7:	○	○	○	○	○	○	○	○	○	○
Image 8:	○	○	○	○	○	○	○	○	○	○
Image 9:	○	○	○	○	○	○	○	○	○	○
Image 10:	○	○	○	○	○	○	○	○	○	○

Number of images similar to the query image: [ ]

Reset Submit



Retrievals for Image 334

Query Image

Rank the Images

	Best Match 1	2	3	4	5	6	7	8	9	Worst Match 10
Image 1:	○	○	○	○	○	○	○	○	○	○
Image 2:	○	○	○	○	○	○	○	○	○	○
Image 3:	○	○	○	○	○	○	○	○	○	○
Image 4:	○	○	○	○	○	○	○	○	○	○
Image 5:	○	○	○	○	○	○	○	○	○	○
Image 6:	○	○	○	○	○	○	○	○	○	○
Image 7:	○	○	○	○	○	○	○	○	○	○
Image 8:	○	○	○	○	○	○	○	○	○	○
Image 9:	○	○	○	○	○	○	○	○	○	○
Image 10:	○	○	○	○	○	○	○	○	○	○

Number of images similar to the query image: [ ]

Reset Submit

Image 1

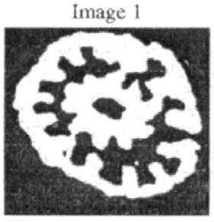


Image 2

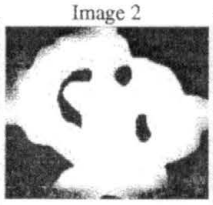


Image 3

Image 4

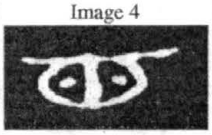


Image 5

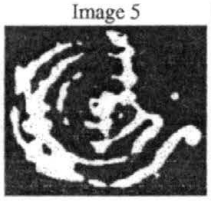


Image 6

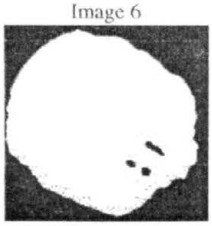


Image 7

Image 8

Image 9

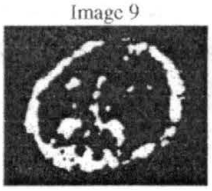


Image 10

Retrievals for Image 363

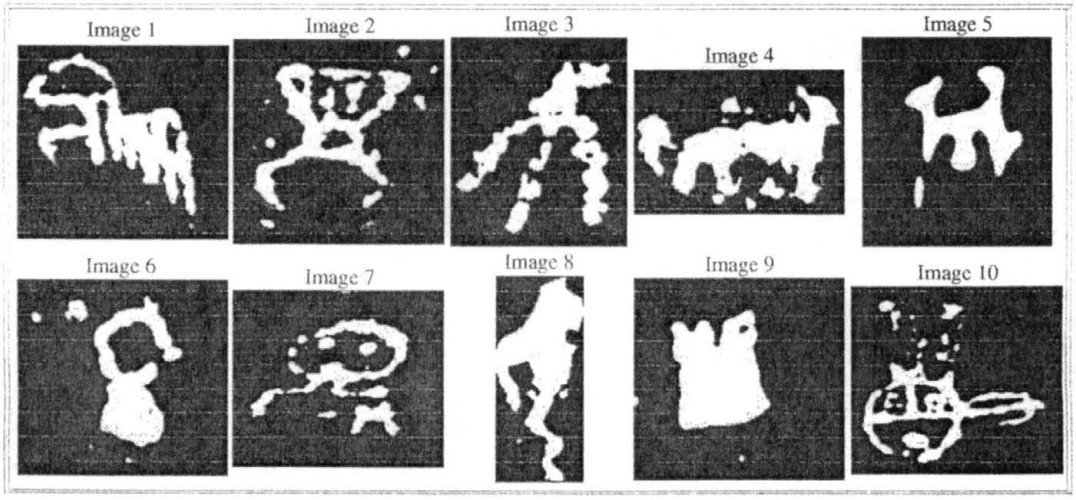
Query Image

Rank the Images

	Best Match 1	2	3	4	5	6	7	8	9	Worst Match 10
Image 1:	○	○	○	○	○	○	○	○	○	○
Image 2:	○	○	○	○	○	○	○	○	○	○
Image 3:	○	○	○	○	○	○	○	○	○	○
Image 4:	○	○	○	○	○	○	○	○	○	○
Image 5:	○	○	○	○	○	○	○	○	○	○
Image 6:	○	○	○	○	○	○	○	○	○	○
Image 7:	○	○	○	○	○	○	○	○	○	○
Image 8:	○	○	○	○	○	○	○	○	○	○
Image 9:	○	○	○	○	○	○	○	○	○	○
Image 10:	○	○	○	○	○	○	○	○	○	○

Number of images similar to the query image: [ ]

[Reset] [Submit]



Retrievals for Image 393

Query Image

Rank the Images

	Best Match 1	2	3	4	5	6	7	8	9	Worst Match 10
Image 1:	○	○	○	○	○	○	○	○	○	○
Image 2:	○	○	○	○	○	○	○	○	○	○
Image 3:	○	○	○	○	○	○	○	○	○	○
Image 4:	○	○	○	○	○	○	○	○	○	○
Image 5:	○	○	○	○	○	○	○	○	○	○
Image 6:	○	○	○	○	○	○	○	○	○	○
Image 7:	○	○	○	○	○	○	○	○	○	○
Image 8:	○	○	○	○	○	○	○	○	○	○
Image 9:	○	○	○	○	○	○	○	○	○	○
Image 10:	○	○	○	○	○	○	○	○	○	○

Number of images similar to the query image: [ ]

[Reset] [Submit]

Image 1

Image 2

Image 3

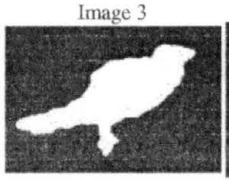


Image 4

Image 5

Image 6

Image 7

Image 8

Image 9

Image 10

Retrievals for Image 465

Query Image

Rank the Images

	Best Match 1	2	3	4	5	6	7	8	9	Worst Match 10
Image 1:	○	○	○	○	○	○	○	○	○	○
Image 2:	○	○	○	○	○	○	○	○	○	○
Image 3:	○	○	○	○	○	○	○	○	○	○
Image 4:	○	○	○	○	○	○	○	○	○	○
Image 5:	○	○	○	○	○	○	○	○	○	○
Image 6:	○	○	○	○	○	○	○	○	○	○
Image 7:	○	○	○	○	○	○	○	○	○	○
Image 8:	○	○	○	○	○	○	○	○	○	○
Image 9:	○	○	○	○	○	○	○	○	○	○
Image 10:	○	○	○	○	○	○	○	○	○	○

Number of images similar to the query image: [ ]

[Reset] [Submit]

Image 1

Image 2

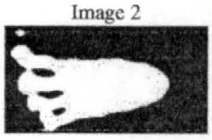


Image 3

Image 4

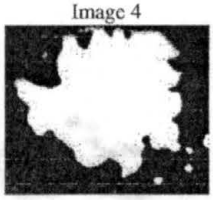


Image 5

Image 6

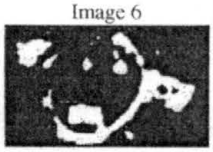


Image 7

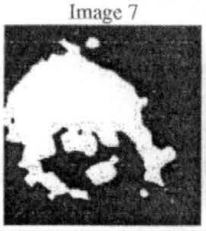


Image 8

Image 9

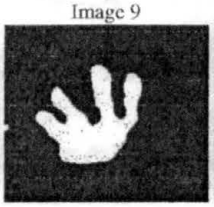
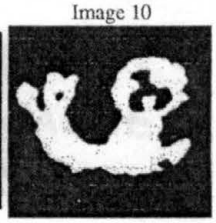


Image 10



Retrievals for Image 493

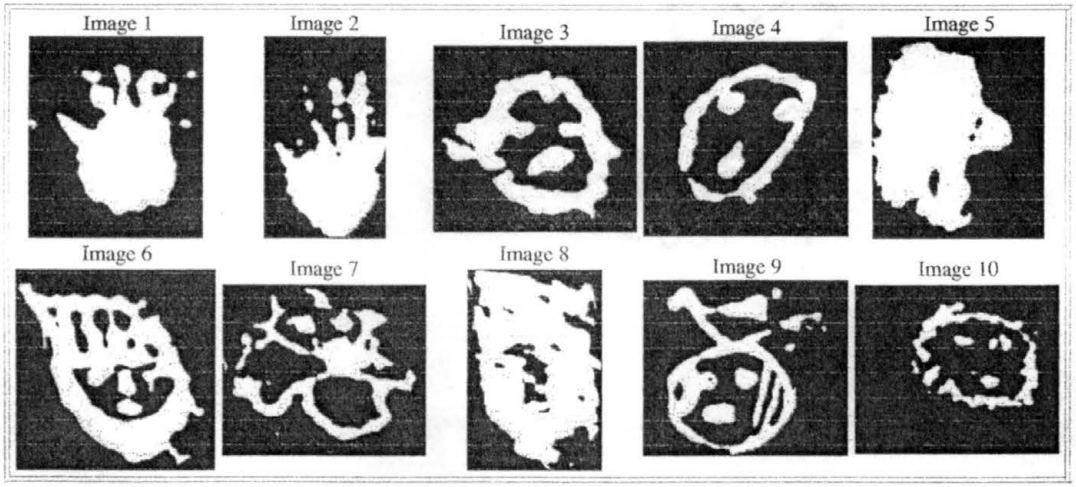
Query Image

Rank the Images

	Best Match 1	2	3	4	5	6	7	8	9	Worst Match 10
Image 1:	○	○	○	○	○	○	○	○	○	○
Image 2:	○	○	○	○	○	○	○	○	○	○
Image 3:	○	○	○	○	○	○	○	○	○	○
Image 4:	○	○	○	○	○	○	○	○	○	○
Image 5:	○	○	○	○	○	○	○	○	○	○
Image 6:	○	○	○	○	○	○	○	○	○	○
Image 7:	○	○	○	○	○	○	○	○	○	○
Image 8:	○	○	○	○	○	○	○	○	○	○
Image 9:	○	○	○	○	○	○	○	○	○	○
Image 10:	○	○	○	○	○	○	○	○	○	○

Number of images similar to the query image: [ ]

Reset Submit



Retrievals for Image 573

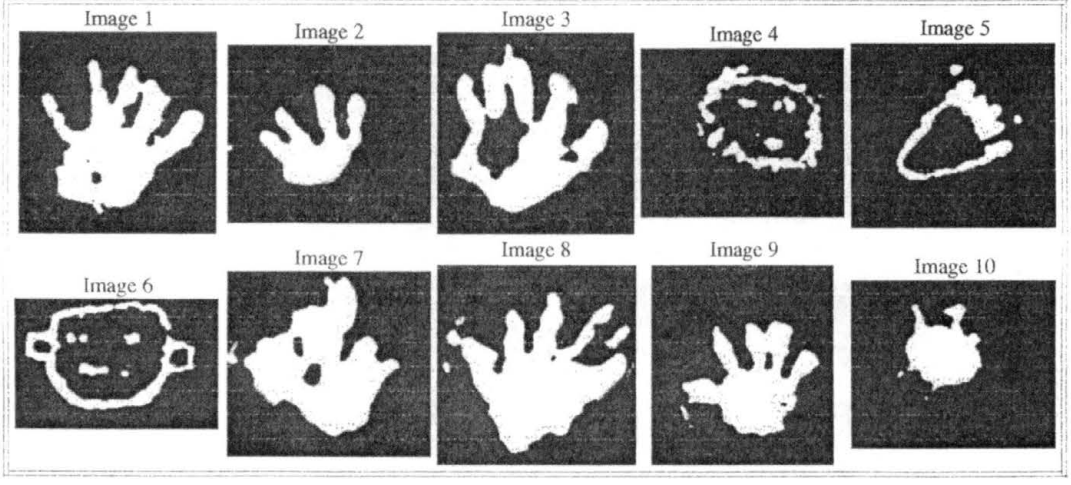
Query Image

Rank the Images

	Best Match 1	2	3	4	5	6	7	8	9	Worst Match 10
Image 1:	○	○	○	○	○	○	○	○	○	○
Image 2:	○	○	○	○	○	○	○	○	○	○
Image 3:	○	○	○	○	○	○	○	○	○	○
Image 4:	○	○	○	○	○	○	○	○	○	○
Image 5:	○	○	○	○	○	○	○	○	○	○
Image 6:	○	○	○	○	○	○	○	○	○	○
Image 7:	○	○	○	○	○	○	○	○	○	○
Image 8:	○	○	○	○	○	○	○	○	○	○
Image 9:	○	○	○	○	○	○	○	○	○	○
Image 10:	○	○	○	○	○	○	○	○	○	○

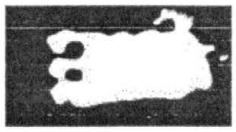
Number of images similar to the query image: [ ]

Reset Submit



Retrievals for Image 583

Query Image

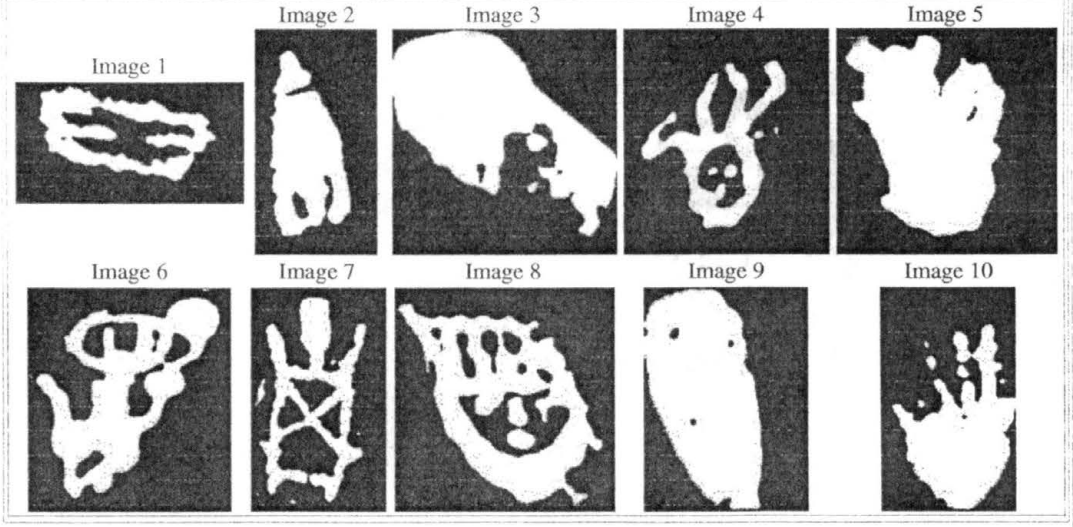


Rank the Images

	Best Match			<-------->					Worst Match
	1	2	3	4	5	6	7	8	9	10
Image 1:	○	○	○	○	○	○	○	○	○	○
Image 2:	○	○	○	○	○	○	○	○	○	○
Image 3:	○	○	○	○	○	○	○	○	○	○
Image 4:	○	○	○	○	○	○	○	○	○	○
Image 5:	○	○	○	○	○	○	○	○	○	○
Image 6:	○	○	○	○	○	○	○	○	○	○
Image 7:	○	○	○	○	○	○	○	○	○	○
Image 8:	○	○	○	○	○	○	○	○	○	○
Image 9:	○	○	○	○	○	○	○	○	○	○
Image 10:	○	○	○	○	○	○	○	○	○	○

Number of images similar to the query image: [ ]

Reset Submit



Retrievals for Image 387

Query Image

Rank the Images

	Best Match			<------->					Worst Match	
	1	2	3	4	5	6	7	8	9	10
Image 1:	○	○	○	○	○	○	○	○	○	○
Image 2:	○	○	○	○	○	○	○	○	○	○
Image 3:	○	○	○	○	○	○	○	○	○	○
Image 4:	○	○	○	○	○	○	○	○	○	○
Image 5:	○	○	○	○	○	○	○	○	○	○
Image 6:	○	○	○	○	○	○	○	○	○	○
Image 7:	○	○	○	○	○	○	○	○	○	○
Image 8:	○	○	○	○	○	○	○	○	○	○
Image 9:	○	○	○	○	○	○	○	○	○	○
Image 10:	○	○	○	○	○	○	○	○	○	○

Number of images similar to the query image:

Reset Submit

Image 1

Image 2

Image 3

Image 4

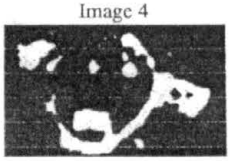


Image 5

Image 6

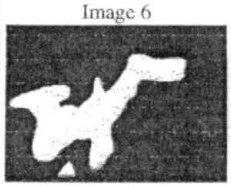


Image 7

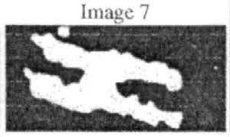


Image 8

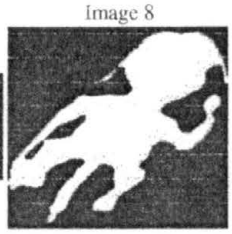
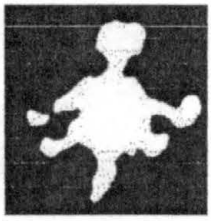


Image 9

Image 10

Retrievals for Image 578

Query Image

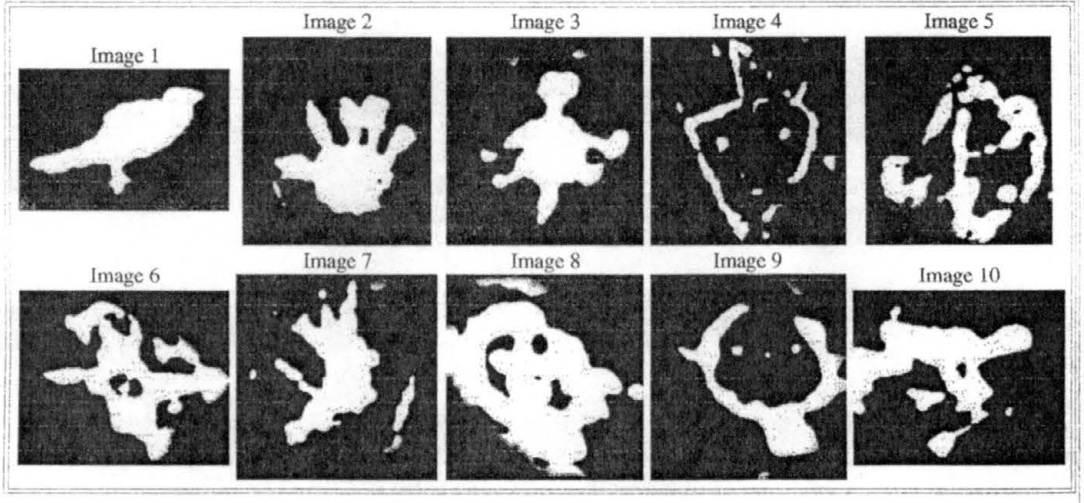


Rank the Images

	Best Match			<-------->						Worst Match
	1	2	3	4	5	6	7	8	9	10
Image 1:	○	○	○	○	○	○	○	○	○	○
Image 2:	○	○	○	○	○	○	○	○	○	○
Image 3:	○	○	○	○	○	○	○	○	○	○
Image 4:	○	○	○	○	○	○	○	○	○	○
Image 5:	○	○	○	○	○	○	○	○	○	○
Image 6:	○	○	○	○	○	○	○	○	○	○
Image 7:	○	○	○	○	○	○	○	○	○	○
Image 8:	○	○	○	○	○	○	○	○	○	○
Image 9:	○	○	○	○	○	○	○	○	○	○
Image 10:	○	○	○	○	○	○	○	○	○	○

Number of images similar to the query image:

Reset Submit



Retrievals for Image 630

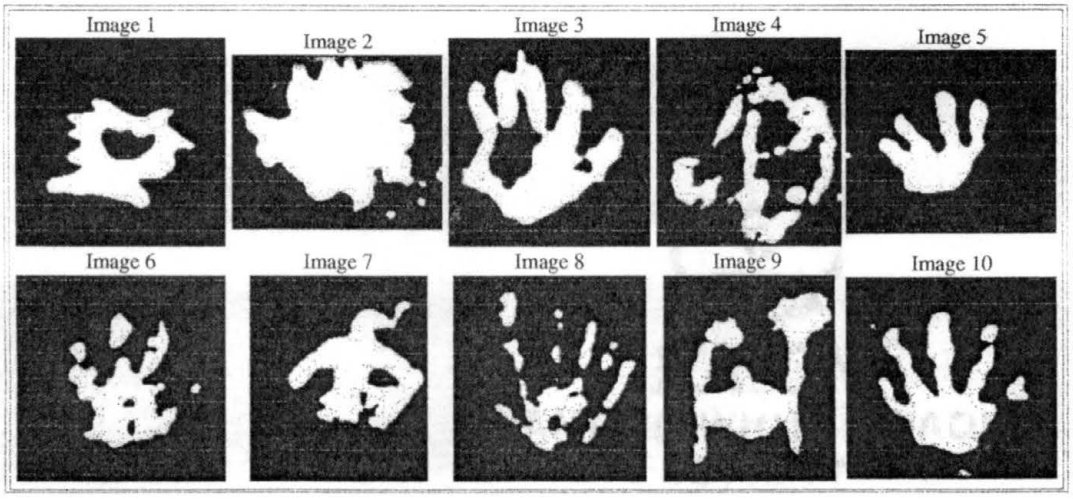
Query Image

Rank the Images

	Best Match 1	2	3	4	5	6	7	8	9	Worst Match 10
Image 1:	○	○	○	○	○	○	○	○	○	○
Image 2:	○	○	○	○	○	○	○	○	○	○
Image 3:	○	○	○	○	○	○	○	○	○	○
Image 4:	○	○	○	○	○	○	○	○	○	○
Image 5:	○	○	○	○	○	○	○	○	○	○
Image 6:	○	○	○	○	○	○	○	○	○	○
Image 7:	○	○	○	○	○	○	○	○	○	○
Image 8:	○	○	○	○	○	○	○	○	○	○
Image 9:	○	○	○	○	○	○	○	○	○	○
Image 10:	○	○	○	○	○	○	○	○	○	○

Number of images similar to the query image: [ ]

Reset Submit



Retrievals for Image 392

Query Image

Rank the Images

	Best Match 1	2	3	4	5	6	7	8	9	Worst Match 10
Image 1:	○	○	○	○	○	○	○	○	○	○
Image 2:	○	○	○	○	○	○	○	○	○	○
Image 3:	○	○	○	○	○	○	○	○	○	○
Image 4:	○	○	○	○	○	○	○	○	○	○
Image 5:	○	○	○	○	○	○	○	○	○	○
Image 6:	○	○	○	○	○	○	○	○	○	○
Image 7:	○	○	○	○	○	○	○	○	○	○
Image 8:	○	○	○	○	○	○	○	○	○	○
Image 9:	○	○	○	○	○	○	○	○	○	○
Image 10:	○	○	○	○	○	○	○	○	○	○

Number of images similar to the query image: [ ]

Reset Submit

Image 1

Image 2

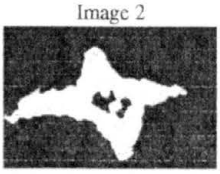


Image 3

Image 4

Image 5

Image 6

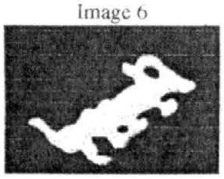


Image 7

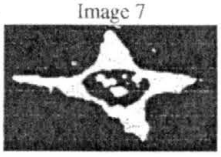


Image 8

Image 9

Image 10

Retrievals for Image 680

Query Image

Rank the Images

	Best Match 1	2	3	4	5	6	7	8	9	Worst Match 10
Image 1:	○	○	○	○	○	○	○	○	○	○
Image 2:	○	○	○	○	○	○	○	○	○	○
Image 3:	○	○	○	○	○	○	○	○	○	○
Image 4:	○	○	○	○	○	○	○	○	○	○
Image 5:	○	○	○	○	○	○	○	○	○	○
Image 6:	○	○	○	○	○	○	○	○	○	○
Image 7:	○	○	○	○	○	○	○	○	○	○
Image 8:	○	○	○	○	○	○	○	○	○	○
Image 9:	○	○	○	○	○	○	○	○	○	○
Image 10:	○	○	○	○	○	○	○	○	○	○

Number of images similar to the query image: [ ]

[Reset] [Submit]

Image 1

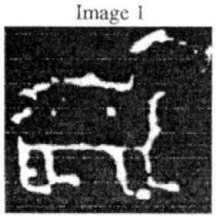


Image 2

Image 3

Image 4

Image 5

Image 6

Image 7

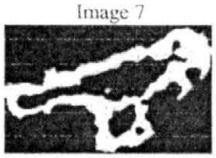


Image 8

Image 9

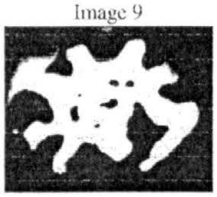


Image 10

Retrievals for Image 547

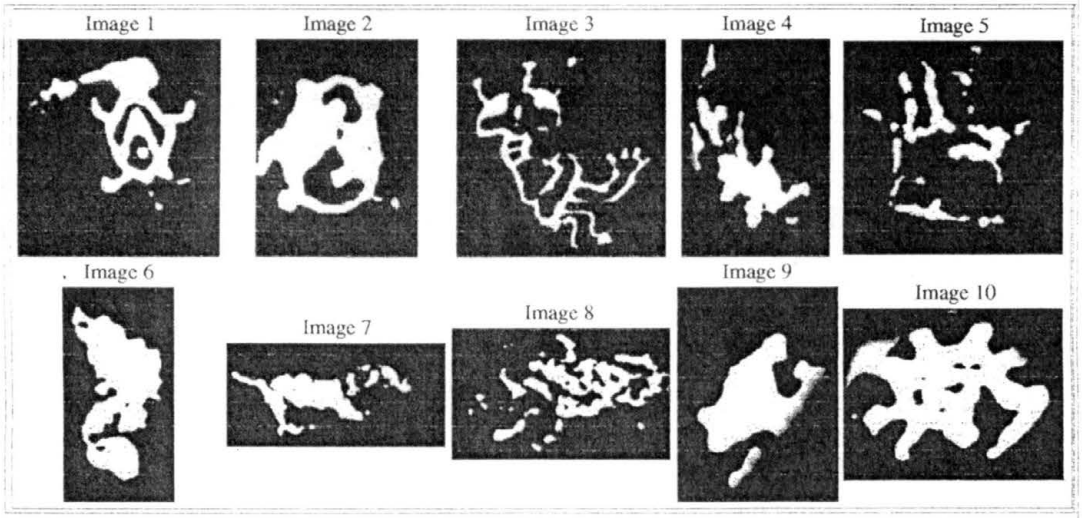
Query Image

Rank the Images

	Best Match 1	2	3	4	5	6	7	8	9	Worst Match 10
Image 1:	○	○	○	○	○	○	○	○	○	○
Image 2:	○	○	○	○	○	○	○	○	○	○
Image 3:	○	○	○	○	○	○	○	○	○	○
Image 4:	○	○	○	○	○	○	○	○	○	○
Image 5:	○	○	○	○	○	○	○	○	○	○
Image 6:	○	○	○	○	○	○	○	○	○	○
Image 7:	○	○	○	○	○	○	○	○	○	○
Image 8:	○	○	○	○	○	○	○	○	○	○
Image 9:	○	○	○	○	○	○	○	○	○	○
Image 10:	○	○	○	○	○	○	○	○	○	○

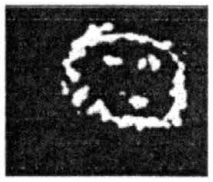
Number of images similar to the query image: [ ]

Reset Submit



Retrievals for Image 518

Query Image



Rank the Images

	Best Match 1	2	3	4	5	6	7	8	9	Worst Match 10
Image 1:	○	○	○	○	○	○	○	○	○	○
Image 2:	○	○	○	○	○	○	○	○	○	○
Image 3:	○	○	○	○	○	○	○	○	○	○
Image 4:	○	○	○	○	○	○	○	○	○	○
Image 5:	○	○	○	○	○	○	○	○	○	○
Image 6:	○	○	○	○	○	○	○	○	○	○
Image 7:	○	○	○	○	○	○	○	○	○	○
Image 8:	○	○	○	○	○	○	○	○	○	○
Image 9:	○	○	○	○	○	○	○	○	○	○
Image 10:	○	○	○	○	○	○	○	○	○	○

Number of images similar to the query image: [ ]

[Reset] [Submit]

Image 1

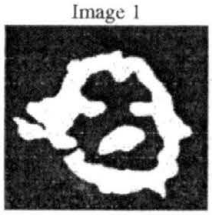


Image 2

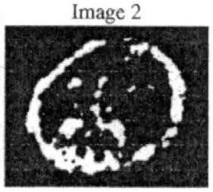


Image 3

Image 4

Image 5

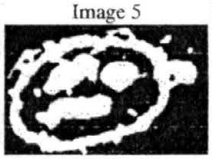


Image 6

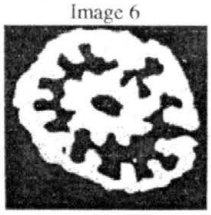


Image 7

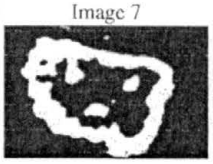


Image 8

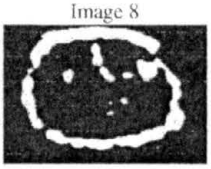


Image 9

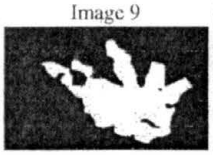


Image 10

Retrievals for Image 458

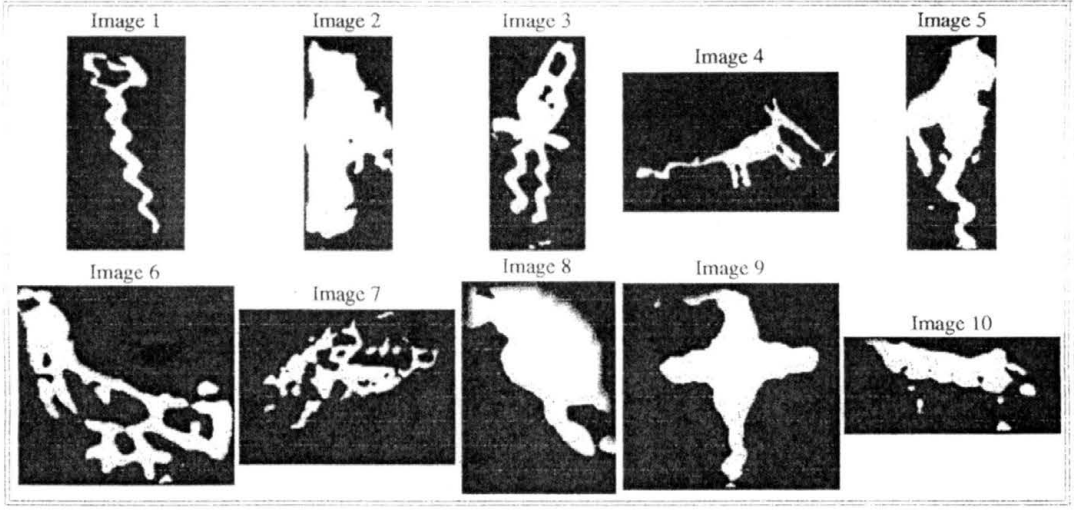
Query Image

Rank the Images

	Best Match 1	2	3	4	5	6	7	8	9	Worst Match 10
Image 1:	○	○	○	○	○	○	○	○	○	○
Image 2:	○	○	○	○	○	○	○	○	○	○
Image 3:	○	○	○	○	○	○	○	○	○	○
Image 4:	○	○	○	○	○	○	○	○	○	○
Image 5:	○	○	○	○	○	○	○	○	○	○
Image 6:	○	○	○	○	○	○	○	○	○	○
Image 7:	○	○	○	○	○	○	○	○	○	○
Image 8:	○	○	○	○	○	○	○	○	○	○
Image 9:	○	○	○	○	○	○	○	○	○	○
Image 10:	○	○	○	○	○	○	○	○	○	○

Number of images similar to the query image: [ ]

Reset Submit



Retrievals for Image 502

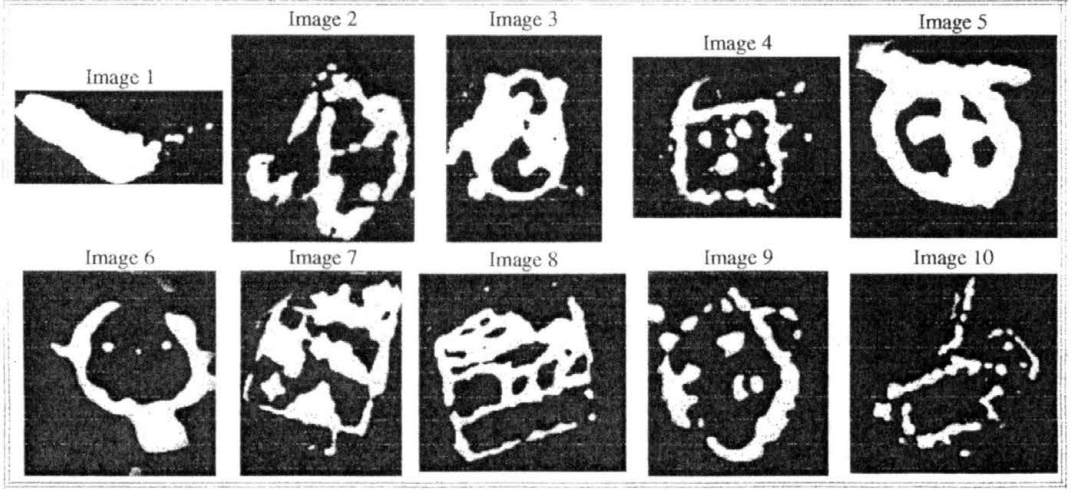
Query Image

Rank the Images

	Best Match 1	2	3	4	5	6	7	8	9	Worst Match 10
Image 1:	○	○	○	○	○	○	○	○	○	○
Image 2:	○	○	○	○	○	○	○	○	○	○
Image 3:	○	○	○	○	○	○	○	○	○	○
Image 4:	○	○	○	○	○	○	○	○	○	○
Image 5:	○	○	○	○	○	○	○	○	○	○
Image 6:	○	○	○	○	○	○	○	○	○	○
Image 7:	○	○	○	○	○	○	○	○	○	○
Image 8:	○	○	○	○	○	○	○	○	○	○
Image 9:	○	○	○	○	○	○	○	○	○	○
Image 10:	○	○	○	○	○	○	○	○	○	○

Number of images similar to the query image: [ ]

[Reset] [Submit]



Retrievals for Image 541

Query Image

Rank the Images

	Best Match 1	2	3	4	5	6	7	8	9	Worst Match 10
Image 1:	○	○	○	○	○	○	○	○	○	○
Image 2:	○	○	○	○	○	○	○	○	○	○
Image 3:	○	○	○	○	○	○	○	○	○	○
Image 4:	○	○	○	○	○	○	○	○	○	○
Image 5:	○	○	○	○	○	○	○	○	○	○
Image 6:	○	○	○	○	○	○	○	○	○	○
Image 7:	○	○	○	○	○	○	○	○	○	○
Image 8:	○	○	○	○	○	○	○	○	○	○
Image 9:	○	○	○	○	○	○	○	○	○	○
Image 10:	○	○	○	○	○	○	○	○	○	○

Number of images similar to the query image:

Reset Submit

Image 1

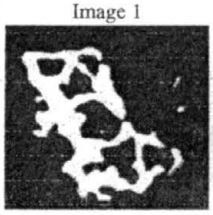


Image 2

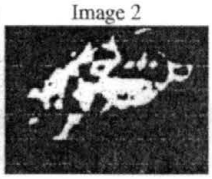


Image 3

Image 4

Image 5

Image 6

Image 7

Image 8

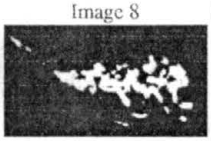


Image 9

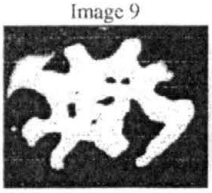
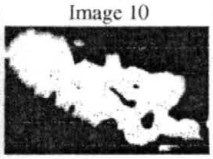


Image 10



Retrievals for Image 631

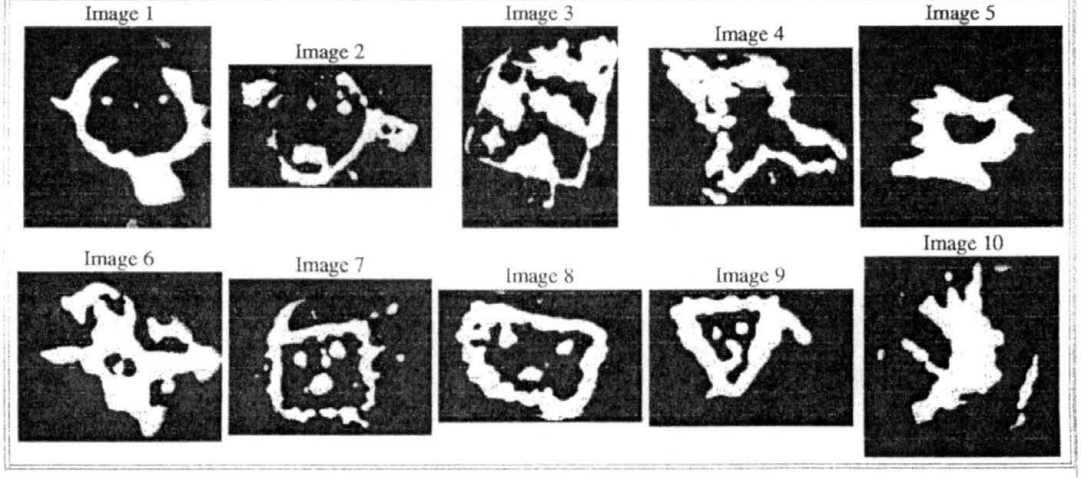
Query Image

Rank the Images

	Best Match 1	2	3	4	5	6	7	8	9	Worst Match 10
Image 1:	○	○	○	○	○	○	○	○	○	○
Image 2:	○	○	○	○	○	○	○	○	○	○
Image 3:	○	○	○	○	○	○	○	○	○	○
Image 4:	○	○	○	○	○	○	○	○	○	○
Image 5:	○	○	○	○	○	○	○	○	○	○
Image 6:	○	○	○	○	○	○	○	○	○	○
Image 7:	○	○	○	○	○	○	○	○	○	○
Image 8:	○	○	○	○	○	○	○	○	○	○
Image 9:	○	○	○	○	○	○	○	○	○	○
Image 10:	○	○	○	○	○	○	○	○	○	○

Number of images similar to the query image:

Reset Submit



Retrievals for Image 617

Query Image

Rank the Images

	Best Match 1	2	3	4	5	6	7	8	9	Worst Match 10
Image 1:	○	○	○	○	○	○	○	○	○	○
Image 2:	○	○	○	○	○	○	○	○	○	○
Image 3:	○	○	○	○	○	○	○	○	○	○
Image 4:	○	○	○	○	○	○	○	○	○	○
Image 5:	○	○	○	○	○	○	○	○	○	○
Image 6:	○	○	○	○	○	○	○	○	○	○
Image 7:	○	○	○	○	○	○	○	○	○	○
Image 8:	○	○	○	○	○	○	○	○	○	○
Image 9:	○	○	○	○	○	○	○	○	○	○
Image 10:	○	○	○	○	○	○	○	○	○	○

Number of images similar to the query image:

Reset Submit

Image 1

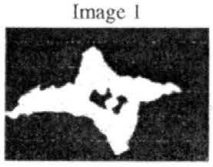


Image 2

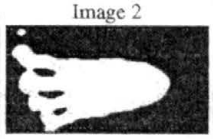


Image 3

Image 4

Image 5

Image 6

Image 7

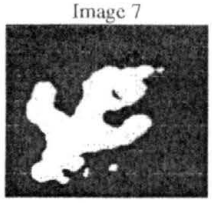


Image 8

Image 9

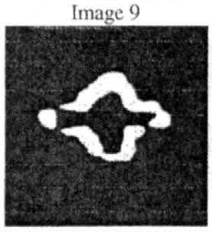
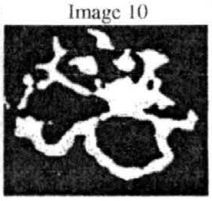


Image 10



retrieval_survey.xls

Image Number	retrieval 1	retrieval 2	retrieval 3	retrieval 4	retrieval 5	retrieval 6	retrieval 7	retrieval 8	retrieval 9	retrieval 10	num similar
322	6	10	8	2	9	7	1	3	4	5	?
322	6	9	10	2	10	8	1	7	3	3	?
322	10	10	10	10	10	10	1	2	10	3	3
322	7	9	10	4	6	5	1	8	2	3	3
322	3	4	5	2	6	8	1	7	9	10	1
322	4	10	9	5	8	6	3	7	1	2	3
322	4	10	8	7	6	5	2	3	1	9	3
322	10	10	10	7	10	10	6	10	8	8	0
322	6	10	10	3	10	10	2	10	6	6	2
322	3	10	6	5	9	2	1	4	7	8	0
322	6	9	10	5	8	2	1	7	3	4	5
322	7	10	10	6	8	8	4	7	2	2	3
Average	6	9.25	8.8333333	4.8333333	8.3333333	6.75	2	6.25	4.6666667	5.25	2.3
Median	6	10	10	5	8.5	7.5	1	7	3.5	4.5	3
St. Dev	2.335496832	1.7122553	1.7494588	2.4432963	1.6143298	2.8324419	1.5954481	2.66714011	3.2003788	2.86435777	1.567021236
Distance	21.897667	21.905838	22.072393	23.876501	24.170147	25.004499	26.64226	27.481829	28.762674	28.95192	25.0765728
334	1	3	2	9	4	7	8	6	5	10	?
334	2	4	5	9	1	6	10	8	3	7	?
334	4	10	1	10	3	10	10	10	2	4	5
334	1	3	2	10	5	4	7	8	6	9	6
334	1	10	2	9	3	6	8	5	4	10	0
334	3	7	4	8	5	9	2	1	6	10	5
334	3	6	1	10	4	5	7	8	2	9	4
334	5	8	7	7	9	9	9	8	7	10	0
334	1	3	1	10	4	6	10	2	1	1	4
334	1	4	2	10	8	6	7	5	3	9	2
334	2	4	1	10	5	10	10	10	3	10	5
334	3	5	2	4	4	7	8	9	6	9	5
Average	2.25	5.5833333	2.5	8.8333333	4.5833333	7.0833333	8	6.66666667	4	8.16666667	3.6
Median	2	4.5	2	9.5	4	6.5	8	8	3.5	9	4.5
St. Dev	1.356801051	2.6097138	1.8829377	1.8006733	2.1514618	1.9752253	2.2563043	2.9336088	1.9540168	2.85508584	2.170509413
	16.210419	19.683203	21.181129	21.524185	21.574804	22.234989	22.356632	22.852808	22.926365	23.671568	21.4216102

retrieval_survey.xls

Image Number	retrieval 1	retrieval 2	retrieval 3	retrieval 4	retrieval 5	retrieval 6	retrieval 7	retrieval 8	retrieval 9	retrieval 10	num similar
363	5	4	6	10	7	1	2	8	9	3	?
363	6	3	5	7	8	2	1	9	10	4	?
363	10	10	10	10	10	2	1	10	10	3	3
363	5	3	10	10	10	2	1	10	10	4	2
363	2	4	6	5	9	3	1	7	9	10	0
363	5	4	6	10	8	1	2	3	7	9	3
363	4	3	6	7	9	2	1	8	10	5	2
363	8	8	10	10	10	7	7	10	10	9	0
363	4	3	7	10	4	4	1	10	10	4	2
363	5	3	10	10	6	4	1	10	10	2	2
363	4	2	10	10	10	3	1	10	10	4	4
363	10	9	10	10	10	7	5	9	10	3	3
Average	5.666666667	4.6666667	8	9.0833333	8.4166667	3.1666667	2	8.66666667	9.5833333	5	2.1
Median	5	3.5	8.5	10	9	2.5	1	9.5	10	4	2
St. Dev	2.46182982	2.7080128	2.1320072	1.7298625	1.9286516	2.0375267	1.9540168	2.0597146	0.9003366	2.73030135	1.286683938
Distance	82.121925	86.4468	88.345116	88.46903	92.541054	97.865517	99.723503	101.166679	104.34519	104.406921	94.5431737
387	10	10	10	5	6	10	10	10	10	10	?
387	7	10	9	1	4	6	5	8	2	3	?
387	2	8	1	5	4	3	10	6	7	7	?
387	1	10	10	2	10	10	10	10	10	10	2
387	2	10	10	4	10	1	10	10	10	3	1
387	4	9	3	8	7	1	6	5	2	10	0
387	6	10	5	9	1	4	3	7	8	2	1
387	4	7	3	8	9	1	6	5	2	10	0
387	10	10	10	1	10	2	3	3	1	10	5
387	10	10	10	3	10	2	10	10	1	10	2
387	10	10	1	2	3	10	10	10	10	10	3
387	9	9	8	9	3	7	8	8	9	10	1
Average	6.25	9.4166667	6.6666667	4.75	6.4166667	4.75	7.5833333	7.66666667	6	7.91666667	1.666666667
Median	6.5	10	8.5	4.5	6.5	3.5	9	8	7.5	10	1
St. Dev	3.545163158	0.9962049	3.7739137	3.0785179	3.3427896	3.6958207	2.8431204	2.46182982	4	3.28794861	1.58113883
	24.811646	29.708384	32.070389	39.671932	40.482018	41.601639	44.581474	44.67284	44.89439	45.124729	38.7619441

retrieval_survey.xls

Image Number	retrieval 1	retrieval 2	retrieval 3	retrieval 4	retrieval 5	retrieval 6	retrieval 7	retrieval 8	retrieval 9	retrieval 10	num similar
392	1	9	5	6	7	2	10	3	8	4	?
392	1	6	10	5	3	2	8	10	9	10	?
392	1	10	10	10	10	10	10	2	10	10	2
392	1	10	10	10	10	10	10	10	10	10	1
392	1	3	2	7	9	4	5	6	8	10	1
392	1	4	7	5	10	2	8	6	9	3	1
392	1	4	8	6	7	2	5	3	9	10	1
392	1	10	10	10	10	10	10	10	10	10	1
392	1	7	7	10	10	5	10	10	10	10	1
392	1	10	10	10	10	2	10	4	10	3	1
392	1	10	10	10	10	10	10	10	2	10	2
392	1	9	9	9	9	9	9	9	9	9	1
Average	1	7.6666667	8.1666667	8.1666667	8.75	5.6666667	8.75	6.91666667	8.6666667	8.25	1.2
Median	1	9	9.5	9.5	10	4.5	10	7.5	9	10	1
St. Dev	0	2.7413777	2.5524795	2.1672493	2.1373305	3.7739137	1.912875	3.26018218	2.2292817	2.98861476	0.421637021
Distance	18.553049	42.2449	51.218941	51.468952	52.594193	52.904213	53.728409	54.276657	55.360973	55.821186	48.8171473
393	6	5	1	3	10	4	9	2	7	8	?
393	5	3	7	1	10	6	4	8	9	2	?
393	3	2	6	5	8	3	1	4	9	10	?
393	10	10	10	1	10	2	10	10	10	10	2
393	8	9	2	3	4	5	7	1	6	10	0
393	8	3	7	2	6	4	1	10	5	9	2
393	3	8	6	5	9	1	2	4	7	10	1
393	10	10	9	10	10	10	10	10	10	10	0
393	2	10	1	1	10	10	10	10	10	10	3
393	3	10	1	2	5	4	10	10	10	10	5
393	2	10	1	4	10	3	10	10	10	10	4
393	4	7	6	5	9	5	8	8	9	9	3
Average	5.333333333	7.25	4.75	3.5	8.4166667	4.75	6.8333333	7.25	8.5	9	2.222222222
Median	4.5	8.5	6	3	9.5	4	8.5	9	9	10	2
St. Dev	2.994945237	3.1658691	3.3608711	2.5761141	2.1933094	2.8001623	3.76185	3.49350046	1.7837652	2.29624199	1.715938357
Distance	45.952152	49.784122	50.03447	53.265659	53.656536	53.795555	54.358574	54.918922	55.148479	56.326412	52.7240881

retrieval_survey.xls

Image Number	retrieval 1	retrieval 2	retrieval 3	retrieval 4	retrieval 5	retrieval 6	retrieval 7	retrieval 8	retrieval 9	retrieval 10	num similar
458	3	4	5	2	8	6	7	10	9	1	?
458	3	9	5	2	4	7	6	8	10	1	?
458	4	9	2	1	3	6	7	9	10	8	?
458	3	10	10	2	10	10	10	10	10	1	3
458	10	10	10	10	10	10	10	10	10	1	1
458	4	8	7	1	3	6	5	10	9	2	2
458	1	10	7	2	4	8	9	6	5	3	0
458	7	10	9	2	6	10	10	10	10	5	1
458	10	10	6	2	10	10	9	10	10	2	2
458	5	10	6	2	3	10	10	4	7	1	4
458	1	10	10	2	4	10	10	10	10	3	4
458	9	9	9	4	9	9	9	9	9	9	1
Average	5	9.0833333	7.1666667	2.6666667	6.1666667	8.5	8.5	8.83333333	9.0833333	3.08333333	2
Median	4	10	7	2	5	9 5	9	10	10	2	2
St. Dev	3.247376564	1.7298625	2.5166115	2.4246212	3.0100841	1.7837652	1.7837652	1.94624736	1.5642793	2.81096338	1.414213562
Distance	52.478928	53.144867	54.153698	56.088356	58.881855	60.506794	61.034572	62.222763	62.93394	64.73724	58.6183013
465	7	5	6	8	9	2	10	4	3	1	?
465	7	8	3	9	10	2	4	5	6	1	?
465	10	10	1	10	10	2	10	10	10	10	2
465	10	10	1	10	10	2	10	10	10	3	2
465	10	9	3	7	8	4	5	6	2	3	0
465	10	7	1	9	8	6	3	2	4	5	1
465	6	4	7	9	10	2	8	6	5	1	0
465	10	10	10	10	10	10	10	8	10	8	0
465	10	10	3	10	10	6	10	10	6	2	2
465	10	10	2	10	10	10	10	3	4	1	1
465	10	10	1	10	10	10	10	10	10	10	1
465	8	9	7	9	9	9	8	8	9	9	0
Average	9	8.5	3.75	9 25	9.5	5.4166667	8.1666667	6.83333333	6.5833333	4.5	0.9
Median	10	9.5	3	9.5	10	5	10	7	6	3	1
St. Dev	1.53741223	2.1105794	3.01888	0.9653073	0.797724	3.5280263	2.6571801	2.91807325	3.0587678	3.72948936	0.875595036
Distance	34.95274	35.342789	36.993549	38.13121	39.349045	39.885277	40.630341	40.714931	41.345966	41.486637	38.8832485

retrieval_survey.xls

Image Number	retrieval 1	retrieval 2	retrieval 3	retrieval 4	retrieval 5	retrieval 6	retrieval 7	retrieval 8	retrieval 9	retrieval 10	num similar
493	7	9	3	1	8	4	6	10	5	2	?
493	10	10	1	10	10	2	9	3	10	10	?
493	10	10	10	10	10	2	10	10	1	10	2
493	7	8	2	5	10	4	10	6	1	3	4
493	7	8	5	1	10	4	6	9	2	3	1
493	3	4	6	5	7	2	10	1	8	9	3
493	7	6	2	1	10	5	8	9	3	4	3
493	10	10	7	7	9	7	10	9	8	9	0
493	10	10	1	1	10	3	2	10	3	3	6
493	7	8	5	3	9	1	10	6	2	4	5
493	5	10	10	6	10	2	10	1	4	3	6
493	8	8	5	6	9	9	4	10	3	4	3
Average	7.583333333	8.4166667	4.75	4.6666667	9.3333333	3.75	7.9166667	7	4.1666667	5.33333333	3.3
Median	7	8.5	5	5	10	3.5	9.5	9	3	4	3
St. Dev	2.193309386	1.880925	3.1370223	3.3393884	0.9847319	2.3403574	2.7784343	3.54195632	2.9797295	3.14305391	2.002775851
Distance	17.353168	17.375229	20.164511	22.075676	22.612637	22.859665	23.95101	23.358385	23.590153	23.600597	21.6941031
502	6	4	5	2	9	3	7	8	1	10	1
502	10	4	5	3	6	2	8	7	1	9	3
502	8	3	5	2	6	4	9	10	1	7	3
502	10	10	10	2	10	3	10	10	1	10	3
502	10	2	10	3	10	10	10	10	1	10	3
502	8	5	4	1	3	6	10	9	2	7	5
502	10	4	8	2	9	3	7	6	1	5	1
502	10	10	10	4	6	5	10	10	3	10	4
502	10	3	5	2	4	3	6	6	3	5	4
502	8	3	7	2	5	1	10	9	4	6	4
502	10	2	10	10	10	10	10	10	1	10	2
502	9	8	9	4	9	6	9	9	9	3	3
Average	9.083333333	4.8333333	7.3333333	3.0833333	7.25	4.6666667	8.8333333	8.66666667	2.3333333	7.66666667	3.222222222
Median	10	4	7.5	2	7.5	3.5	9.5	9	1	8	3
St. Dev	1.311372171	2.8867513	2.4246212	2.3532698	2.5271256	2.9024546	1.4668044	1.55699789	2.348436	2.49848439	1.201850425
Distance	23.828186	23.977325	24.032459	25.106201	26.172144	26.258217	27.020056	27.956804	28.022846	28.20632	26.0580558

retrieval_survey.xls

Image Number	retrieval 1	retrieval 2	retrieval 3	retrieval 4	retrieval 5	retrieval 6	retrieval 7	retrieval 8	retrieval 9	retrieval 10	num similar
518	8	2	10	3	4	7	6	1	9	5	3
518	5	3	10	4	6	8	2	1	9	7	3
518	10	2	10	3	3	9	4	1	9	8	3
518	10	3	10	2	10	10	4	1	10	10	4
518	2	4	8	6	5	9	1	3	10	7	8
518	3	4	10	2	5	8	1	6	9	7	6
518	6	2	10	5	4	9	3	1	8	7	4
518	8	6	9	3	5	10	1	2	10	7	6
518	2	1	10	2	1	5	2	1	10	3	7
518	2	7	9	5	6	8	3	1	10	4	7
518	6	3	10	5	4	7	2	1	10	10	7
518	4	3	7	2	4	6	7	3	9	5	5
Average	5.5	3.3333333	9.4166667	3.5	4.75	8	3	1.83333333	9.4166667	6.66666667	6
Median	5.5	3	10	3	4.5	8	2.5	1	9.5	7	6
St. Dev	3	1.7232809	0.9962049	1.4459976	2.1373305	1.5374122	1.9540168	1.52752523	0.6685579	2.14617348	1.414213562
	10.674562	13.330877	15.189759	15.345491	16.68466	17.216282	17.343973	17.959778	18.065002	18.495089	16.03054732
541	8	7	2	9	4	6	1	3	10	5	?
541	8	7	1	5	6	6	2	3	10	9	?
541	10	10	1	10	4	10	2	3	10	10	4
541	10	10	10	10	10	10	10	10	10	10	0
541	10	7	1	3	6	5	2	4	8	9	0
541	6	8	1	9	4	5	2	3	10	7	2
541	9	5	1	8	3	7	2	4	10	6	2
541	10	10	5	10	10	10	6	10	10	10	0
541	2	5	2	5	6	7	8	10	10	10	2
541	8	7	1	9	4	5	2	3	10	6	0
541	10	10	3	10	4	10	1	2	10	10	4
541	9	9	4	9	8	8	4	9	9	9	2
Average	8.333333333	7.9166667	2.6666667	8.0833333	5.75	7.4166667	3.5	5.33333333	9.75	8.41666667	1.6
Median	9	7.5	1.5	9	5	7	2	3.5	10	9	2
St. Dev	2.348435972	1.880925	2.6742317	2.3915888	2.4167973	2.1087839	2.9387691	3.31205329	0.6215816	1.88092498	1.577621275
	32.95739	36.111881	43.803448	47.400543	47.499493	47.689411	52.478928	52.69151	52.774189	55.022381	46.8429174

retrieval_survey.xls

Image Number	retrieval 1	retrieval 2	retrieval 3	retrieval 4	retrieval 5	retrieval 6	retrieval 7	retrieval 8	retrieval 9	retrieval 10	num similar
547	9	10	5	2	4	3	1	7	6	8	?
547	4	7	2	3	9	1	5	6	8	10	?
547	10	10	1	2	10	3	10	10	10	10	3
547	10	10	10	2	10	4	1	3	10	10	4
547	3	4	10	8	9	1	7	6	5	2	0
547	3	10	1	2	6	7	4	5	8	9	1
547	6	10	1	2	9	4	3	5	8	7	0
547	10	10	10	10	10	5	6	10	9	9	0
547	8	10	1	3	10	3	10	10	9	10	3
547	5	7	8	3	10	1	4	9	6	2	0
547	10	10	10	3	10	2	1	4	10	10	4
547	8	9	4	4	9	7	6	9	9	9	2
Average	7.166666667	8.9166667	5.25	3.6666667	8.8333333	3.4166667	4.8333333	7	8.1666667	8	1.7
Median	8	10	4.5	3	9.5	3	4.5	6.5	8.5	9	1.5
St. Dev	2.823065173	1.9286516	4.0704032	2.6053558	1.898963	2.1087839	3.1574827	2.5226249	1.6966991	2.95419578	1.702938637
Distance	49.83749	50.828465	52.369942	52.504925	52.873722	52.959053	53.005112	56.233284	56.808304	56.841461	53.4261758
573	1	6	2	9	8	10	5	4	3	7	3
573	1	3	3	8	9	10	10	3	2	7	3
573	3	10	2	10	10	10	10	10	1	10	3
573	1	5	3	10	10	10	6	4	2	7	6
573	1	3	4	8	9	10	6	5	2	7	5
573	1	5	2	9	6	10	7	4	3	8	5
573	1	4	3	8	7	10	6	5	2	9	5
573	1	4	5	9	8	10	6	3	2	7	5
573	1	1	1	10	3	10	2	1	1	3	8
573	1	4	2	9	7	10	6	5	3	8	6
573	1	5	2	10	10	10	10	4	3	10	5
573	1	3	2	6	7	7	7	2	2	7	5
Average	1.166666667	4.4166667	2.5833333	8.8333333	7.8333333	9.75	6.75	4.16666667	2.1666667	7.5	5.3
Median	1	4	2	9	8	10	6	4	2	7	5
St. Dev	0.577350269	2.1933094	1.0836247	1.1934163	2.0375267	0.8660254	2.3403574	2.20879784	0.7177406	1.83402191	1.251665557
	13.335711	18.987604	19.307379	21.008276	21.150614	21.657022	21.989332	22.495743	22.571808	22.728682	20.5232171

Image Number	retrieval 1	retrieval 2	retrieval 3	retrieval 4	retrieval 5	retrieval 6	retrieval 7	retrieval 8	retrieval 9	retrieval 10	num similar
578	7	6	1	8	10	5	4	9	3	2	?
578	10	10	1	10	9	2	4	7	8	10	?
578	10	3	1	10	10	10	2	10	10	10	3
578	10	3	1	10	10	2	10	10	10	10	2
578	6	2	1	10	9	5	3	8	7	4	1
578	7	9	1	5	6	2	3	10	8	4	1
578	6	2	1	10	8	3	7	5	9	4	1
578	6	2	1	10	9	4	3	7	8	5	1
578	2	2	1	10	10	2	4	7	10	2	6
578	7	4	1	10	9	2	8	6	5	3	1
578	10	2	1	10	10	10	10	10	10	10	2
578	10	4	1	8	8	8	6	8	8	9	2
Average	7.583333333	4.0833333	1	9.25	9	4.5833333	5.3333333	8.08333333	8	6.08333333	2
Median	7	3	1	10	9	3.5	4	8	8	4.5	1.5
St. Dev	2.503028469	2.8109634	0	1.544786	1.2060454	3.1176429	2.8069179	1.72986249	2.1742292	3.39674532	1.56347192
Distance	29.014378	32.441334	32.52247	32.822289	33.197044	34.422512	35.640335	36.141193	36.311005	36.329308	33.8841868
583	3	1	6	10	2	8	9	7	4	5	3
583	1	2	9	8	9	10	4	10	3	5	3
583	1	2	10	10	10	10	3	10	10	10	3
583	1	2	10	10	3	10	10	10	4	5	2
583	4	1	3	10	5	8	9	7	2	6	0
583	5	1	9	8	3	7	6	10	4	2	2
583	2	1	4	10	5	7	9	8	3	6	2
583	2	1	10	10	6	10	10	10	3	8	3
583	2	3	10	10	8	10	10	10	4	4	4
583	6	1	5	10	2	10	7	10	4	3	1
583	2	1	10	10	10	10	10	10	3	10	3
583	3	1	7	9	9	9	8	8	2	7	3
Average	2.666666667	1.4166667	7.75	9.5833333	6	9.0833333	7.9166667	9.16666667	3.8333333	5.91666667	2.3
Median	2	1	9	10	5.5	10	9	10	3.5	5.5	2.5
St. Dev	1.61432977	0.6685579	2.6328346	0.7929615	3.1042493	1.2401124	2.4293034	1.26730446	2.081666	2.50302847	1.159501809
	13.969042	18.583557	19.029232	19.345053	22.335873	22.550026	23.524883	24.657656	25.387058	25.832863	21.5215243

Image Number	retrieval 1	retrieval 2	retrieval 3	retrieval 4	retrieval 5	retrieval 6	retrieval 7	retrieval 8	retrieval 9	retrieval 10	num similar
617	1	7	10	10	9	5	3	8	2	4	?
617	1	9	8	7	10	6	5	4	2	3	?
617	1	9	5	4	7	8	3	10	2	6	?
617	1	7	9	4	10	8	2	6	3	9	?
617	1	10	10	10	10	10	2	10	10	10	2
617	1	8	4	9	10	5	3	6	2	7	2
617	1	5	8	7	9	4	3	6	2	10	1
617	1	10	10	10	10	10	5	10	4	10	1
617	1	10	10	8	10	10	3	8	3	5	3
617	1	9	6	4	8	5	7	3	2	10	2
617	1	10	10	10	10	10	3	10	2	10	3
617	2	9	9	9	9	9	9	9	4	9	2
Average	1.083333333	8.5833333	8.25	7.6666667	9.3333333	7.5	4	7.5	3.1666667	7.75	2
Median	1	9	9	8.5	10	8	3	8	2	9	2
St. Dev	0.288675135	1.5642793	2.1373305	2.4618298	0.9847319	2.3548789	2.1320072	2.46797672	2.2896341	2.63283463	0.755928946
Distance	37.195591	40.8577	41.800522	42.195827	42.400806	43.241375	43.528225	43.651119	44.619335	45.143528	42.4634028
630	4	9	7	10	6	2	1	8	3	5	3
630	8	7	5	10	3	10	1	10	2	9	3
630	10	10	3	10	4	1	2	10	10	6	5
630	10	10	10	10	10	1	2	10	10	10	2
630	6	8	9	10	4	2	1	7	3	5	0
630	5	10	6	9	4	7	1	8	2	3	1
630	2	7	4	8	5	1	6	9	10	6	0
630	10	10	10	10	10	4	5	10	10	6	1
630	7	10	4	10	7	1	1	10	2	3	4
630	8	10	7	9	6	1	3	5	2	4	1
630	10	10	10	10	10	10	1	10	10	10	1
630	8	10	9	8	9	9	5	8	3	9	1
Average	7.333333333	9.25	7	9.5	6.5	4.0833333	2.4166667	8.75	5.5833333	6.33333333	1.6
Median	8	10	7	10	6	2	1.5	9.5	3	6	1
St. Dev	2.640018365	1.2154311	2.5936987	0.797724	2.6457513	3.800917	1.880925	1.60255478	3.918681	2.57022579	1.646545205
	28.601717	31.04698	31.152096	31.814449	31.821949	32.25848	32.687893	32.813713	32.898365	33.108055	31.8203697

retrieval_survey.xls

Image Number	retrieval 1	retrieval 2	retrieval 3	retrieval 4	retrieval 5	retrieval 6	retrieval 7	retrieval 8	retrieval 9	retrieval 10	num similar
631	2	8	9	3	7	1	10	6	5	4	1
631	8	6	9	5	1	7	4	3	2	10	3
631	10	9	5	1	3	2	4	5	6	8	3
631	10	10	10	7	3	2	4	10	10	10	4
631	10	10	10	3	2	1	10	10	10	10	1
631	6	7	8	9	2	3	4	5	1	10	4
631	9	10	7	8	1	2	6	4	5	3	0
631	10	10	10	7	6	4	10	10	5	10	2
631	9	10	4	4	2	3	4	4	4	2	3
631	7	10	8	3	2	1	5	6	4	10	2
631	10	10	10	2	3	1	10	10	10	10	3
631	9	9	9	9	8	2	7	7	9	3	2
Average	8.333333333	9.0833333	8.25	5.0833333	3.3333333	2.4166667	6.5	6.66666667	5.9166667	7.5	2.333333333
Median	9	10	9	4.5	2.5	2	5.5	6	5	10	2
St. Dev	2.386832566	1.3789544	2.0056738	2.8109634	2.348436	1.7298625	2.7468991	2.67423169	3.1466673	3.39786029	1.322875656
-	25.893467	26.978565	27.494106	27.816525	31.202972	31.504745	32.508636	32.816513	32.4127	33.087124	30.1715353
680	1	7	3	5	6	4	2	8	10	9	3
680	1	10	5	6	3	10	2	7	9	8	3
680	2	10	3	10	10	4	10	1	10	10	4
680	2	10	10	10	10	1	10	10	10	10	2
680	1	2	3	6	8	4	5	10	10	7	0
680	1	3	5	6	7	4	2	8	9	10	2
680	4	1	2	6	7	3	5	8	10	9	1
680	10	7	6	10	10	5	10	10	10	10	0
680	1	3	10	10	3	5	1	10	10	3	5
680	2	5	3	7	8	6	1	4	10	9	3
680	2	5	10	10	10	3	1	4	10	10	5
680	3	9	9	9	9	9	9	9	9	9	1
Average	2.5	6	5.75	7.9166667	7.5833333	4.8333333	4.8333333	7.41666667	9.75	8.66666667	2.3
Median	2	6	5	8	8	4	3.5	8	10	9	2
St. Dev	2.540579748	3.3028913	3.1658691	2.0652243	2.5390884	2.5166115	3.8807997	2.93747985	0.452267	2.01509455	1.888562063
	47.172535	47.60997	48.1381	50.248177	52.615398	53.005112	53.35117	54.807537	55.290562	56.069897	51.8308458

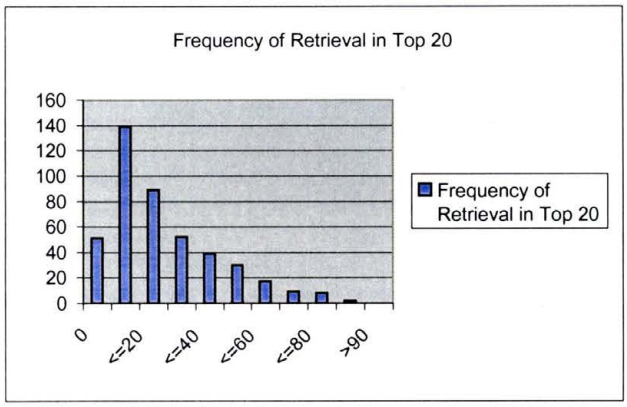
APPENDIX C

Spreadsheet: Image Retrieval Statistics

Name	# Retrievals
bb-5_thresh.xcf	0
img1-3_thresh.xcf	0
img104-1_thresh.xcf	0
img104-4_thresh.xcf	0
img105-1_thresh.xcf	0
img111-1_thresh.xcf	0
img113-1_thresh.xcf	0
img136-1_thresh.xcf	0
img137-1_thresh.xcf	0
img137-4_thresh.xcf	0
img143-1_thresh.xcf	0
img143-2_thresh.xcf	0
img150-1_thresh.xcf	0
img151-1_thresh.xcf	0
img151-2_thresh.xcf	0
img2-2_thresh.xcf	0
img2-3_thresh.xcf	0
img22-3_thresh.xcf	0
img22-4_thresh.xcf	0
img24-1_thresh.xcf	0
img27-2_thresh.xcf	0
img33-2_thresh.xcf	0
img33-3_thresh.xcf	0
img33-4_thresh.xcf	0
img33-5_thresh.xcf	0
img35-1_thresh.xcf	0
img37-4_thresh.xcf	0
img39-2_thresh.xcf	0
img42-3_thresh.xcf	0
img43-1_thresh.xcf	0
img43-2_thresh.xcf	0
img45-10_thresh.xcf	0
img51-2_thresh.xcf	0
img51-5_thresh.xcf	0
img52-1_thresh.xcf	0
img56-3_thresh.xcf	0
img59-1_thresh.xcf	0
img59-6_thresh.xcf	0
img65-3_thresh.xcf	0
img69-2_thresh.xcf	0
img72-3_thresh.xcf	0
img73-1_thresh.xcf	0
img73-3_thresh.xcf	0
img77-1_thresh.xcf	0
img81-11_thresh.xcf	0
img88-5_thresh.xcf	0
img88-7_thresh.xcf	0
img9-2_thresh.xcf	0
img97-1_thresh.xcf	0
img99-2_thresh.xcf	0
img99-3_thresh.xcf	0
img101-2_thresh.xcf	1
img118-4_thresh.xcf	1
img121-3_thresh.xcf	1
img15-5_thresh.xcf	1
img21-3_thresh.xcf	1

Frequency of Retrievals in Top 20

0	51
<=10	139
<=20	89
<=30	52
<=40	39
<=50	30
<=60	17
<=70	9
<=80	8
<=90	2
>90	0



Name	# Retrievals
img40-1_thresh.xcf	1
img46-2_thresh.xcf	1
img48-3_thresh.xcf	1
img51-4_thresh.xcf	1
img51-8_thresh xcf	1
img52-3_thresh.xcf	1
img55-1_thresh xcf	1
img81-1_thresh.xcf	1
img81-9_thresh.xcf	1
img93-6_thresh.xcf	1
img117-2_thresh.xcf	2
img119-1_thresh.xcf	2
img121-5_thresh.xcf	2
img125-2_thresh.xcf	2
img13-2_thresh xcf	2
img144-2_thresh.xcf	2
img144-4_thresh xcf	2
img146-1_thresh xcf	2
img153-4_thresh.xcf	2
img155-2_thresh xcf	2
img26-2_thresh xcf	2
img27-3_thresh.xcf	2
img36-1_thresh xcf	2
img45-4_thresh xcf	2
img45-9_thresh.xcf	2
img48-1_thresh.xcf	2
img63-1_thresh.xcf	2
img7-3_thresh.xcf	2
img70-2_thresh xcf	2
img70-8_thresh xcf	2
img79-5_thresh xcf	2
img91-2_thresh xcf	2
img99-6_thresh xcf	2
img114-1_thresh xcf	3
img118-3_thresh.xcf	3
img123-1_thresh xcf	3
img15-4_thresh.xcf	3
img19-4_thresh xcf	3
img32-1_thresh.xcf	3
img53-6_thresh.xcf	3
img6-3_thresh xcf	3
img76-3_thresh.xcf	3
img88-6_thresh xcf	3
img92-2_thresh.xcf	3
img93-5_thresh xcf	3
img104-2_thresh xcf	4
img118-2_thresh.xcf	4
img12-1_thresh.xcf	4
img148-1_thresh.xcf	4
img155-3_thresh.xcf	4
img24-3_thresh xcf	4
img41-1_thresh.xcf	4
img54-1_thresh.xcf	4
img64-1_thresh xcf	4
img72-6_thresh.xcf	4
img81-3_thresh.xcf	4

Name	# Retrievals
img91-5_thresh xcf	4
img133-1_thresh xcf	5
img21-5_thresh xcf	5
img21-6_thresh xcf	5
img50-5_thresh xcf	5
img67-2_thresh xcf	5
img72-5_thresh xcf	5
img81-2_thresh xcf	5
img84-1_thresh xcf	5
img86-3_thresh xcf	5
bb-3_thresh xcf	6
img108-3_thresh xcf	6
img124-1_thresh xcf	6
img132-3_thresh xcf	6
img142-1_thresh xcf	6
img15-8_thresh xcf	6
img153-1_thresh xcf	6
img4-4_thresh xcf	6
img56-4_thresh xcf	6
img58-1_thresh xcf	6
img60-1_thresh xcf	6
img71-1_thresh xcf	6
img79-3_thresh xcf	6
img81-6_thresh xcf	6
img81-7_thresh.xcf	6
img88-4_thresh xcf	6
img1-1_thresh xcf	7
img155-1_thresh xcf	7
img22-6_thresh xcf	7
img27-1_thresh xcf	7
img31-2_thresh xcf	7
img65-2_thresh xcf	7
img7-1_thresh xcf	7
img70-6_thresh xcf	7
img74-1_thresh xcf	7
img82-1_thresh xcf	7
img85-1_thresh xcf	7
img91-3_thresh xcf	7
img106-3_thresh xcf	8
img123-2_thresh xcf	8
img125-3_thresh xcf	8
img147-1_thresh xcf	8
img21-4_thresh xcf	8
img26-3_thresh xcf	8
img32-2_thresh xcf	8
img33-1_thresh xcf	8
img34-2_thresh xcf	8
img57-2_thresh xcf	8
img59-5_thresh xcf	8
img6-2_thresh xcf	8
img86-4_thresh xcf	8
img87-1_thresh xcf	8
img99-8_thresh xcf	8
img107-1_thresh xcf	9
img129-1_thresh xcf	9
img31-1_thresh xcf	9

Name	# Retrievals
img42-1_thresh xcf	9
img59-7_thresh xcf	9
img73-2_thresh.xcf	9
img76-1_thresh xcf	9
img76-2_thresh xcf	9
img99-1_thresh xcf	9
img99-4_thresh xcf	9
img99-7_thresh xcf	9
img100-1_thresh xcf	10
img104-3_thresh.xcf	10
img126-1_thresh xcf	10
img128-1_thresh xcf	10
img139-1_thresh xcf	10
img152-1_thresh.xcf	10
img26-1_thresh xcf	10
img42-2_thresh xcf	10
img45-3_thresh xcf	10
img49-2_thresh xcf	10
img52-5_thresh xcf	10
img57-1_thresh xcf	10
img6-1_thresh.xcf	10
img85-2_thresh xcf	10
img103-1_thresh xcf	11
img106-1_thresh xcf	11
img115-3_thresh xcf	11
img120-1_thresh xcf	11
img148-2_thresh xcf	11
img151-3_thresh xcf	11
img2-1_thresh xcf	11
img37-2_thresh xcf	11
img4-3_thresh.xcf	11
img45-5_thresh xcf	11
img52-4_thresh xcf	11
img6-4_thresh.xcf	11
img66-2_thresh xcf	11
img109-1_thresh xcf	12
img12-2_thresh xcf	12
img121-2_thresh xcf	12
img134-1_thresh xcf	12
img16-1_thresh xcf	12
img16-2_thresh xcf	12
img18-5_thresh xcf	12
img19-2_thresh xcf	12
img49-1_thresh xcf	12
img67-1_thresh xcf	12
img72-2_thresh xcf	12
img88-1_thresh xcf	12
img91-1_thresh xcf	12
img94-1_thresh xcf	12
img10-3_thresh.xcf	13
img148-3_thresh xcf	13
img37-3_thresh xcf	13
img45-7_thresh xcf	13
img10-2_thresh xcf	14
img101-1_thresh xcf	14
img121-1_thresh xcf	14

Name	# Retrievals
img137-2_thresh xcf	14
img19-3_thresh xcf	14
img25-3_thresh xcf	14
img28-2_thresh.xcf	14
img4-1_thresh.xcf	14
img46-1_thresh.xcf	14
img51-6_thresh.xcf	14
img89-1_thresh xcf	14
img93-3_thresh xcf	14
img117-1_thresh.xcf	15
img122-3_thresh xcf	15
img144-1_thresh xcf	15
img153-3_thresh xcf	15
img17-1_thresh.xcf	15
img22-1_thresh xcf	15
img48-2_thresh xcf	15
img72-1_thresh.xcf	15
img80-1_thresh xcf	15
img88-2_thresh.xcf	15
img88-8_thresh xcf	15
img11-1_thresh xcf	16
img144-5_thresh xcf	16
img16-3_thresh xcf	16
img3-2_thresh xcf	16
img59-2_thresh xcf	16
img61-1_thresh xcf	16
img70-9_thresh xcf	16
img78-1_thresh xcf	16
img79-2_thresh.xcf	16
img91-4_thresh xcf	16
img106-2_thresh xcf	17
img115-1_thresh.xcf	17
img116-1_thresh xcf	17
img122-1_thresh xcf	17
img124-2_thresh.xcf	17
img29-3_thresh xcf	17
img4-5_thresh.xcf	17
img56-2_thresh xcf	17
img93-1_thresh xcf	17
img1-6_thresh xcf	18
img154-1_thresh xcf	18
img18-3_thresh.xcf	18
img44-3_thresh xcf	18
img53-9_thresh xcf	18
img95-1_thresh xcf	18
img110-1_thresh xcf	19
img124-4_thresh xcf	19
img50-4_thresh xcf	19
img59-4_thresh xcf	19
img63-3_thresh xcf	19
img74-2_thresh.xcf	19
img130-1_thresh.xcf	20
img14-1_thresh.xcf	20
img144-3_thresh.xcf	20
img34-3_thresh xcf	20
img44-1_thresh.xcf	20

Name	# Retrıevals
ımg66-1_thresh xcf	20
ımg140-1_thresh xcf	21
ımg15-3_thresh xcf	21
ımg45-8_thresh xcf	21
ımg47-1_thresh xcf	21
ımg65-1_thresh.xcf	21
ımg83-1_thresh xcf	21
ımg87-2_thresh xcf	21
ımg9-3_thresh xcf	21
ımg93-2_thresh xcf	21
ımg93-4_thresh.xcf	21
ımg10-1_thresh xcf	22
ımg32-3_thresh xcf	22
ımg70-1_thresh xcf	22
ımg96-1_thresh xcf	22
ımg18-2_thresh xcf	23
ımg28-1_thresh xcf	23
ımg45-2_thresh xcf	23
ımg50-3_thresh xcf	23
ımg51-7_thresh xcf	23
ımg6-5_thresh xcf	23
ımg61-3_thresh xcf	23
ımg72-4_thresh xcf	23
ımg86-2_thresh xcf	23
ımg39-1_thresh xcf	24
ımg69-1_thresh xcf	24
bb-1_thresh xcf	25
ımg115-2_thresh xcf	25
ımg132-1_thresh xcf	25
ımg75-1_thresh xcf	25
ımg118-1_thresh xcf	26
ımg38-1_thresh xcf	26
ımg49-3_thresh xcf	26
ımg79-1_thresh xcf	26
ımg1-2_thresh xcf	27
ımg122-2_thresh xcf	27
ımg50-1_thresh xcf	27
ımg53-3_thresh xcf	27
ımg79-4_thresh xcf	27
ımg92-1_thresh xcf	27
ımg15-7_thresh xcf	28
ımg18-1_thresh xcf	28
ımg23-1_thresh xcf	28
ımg46-3_thresh xcf	28
ımg53-4_thresh xcf	28
ımg98-2_thresh xcf	28
ımg108-1_thresh xcf	29
ımg117-3_thresh xcf	29
ımg13-1_thresh.xcf	29
ımg50-2_thresh xcf	29
ımg24-2_thresh xcf	30
ımg42-4_thresh xcf	30
ımg86-1_thresh xcf	30
ımg133-2_thresh xcf	31
ımg149-1_thresh xcf	31
ımg70-7_thresh xcf	31

Name	# Retrievals
img81-10_thresh xcf	31
img81-4_thresh.xcf	31
img9-1_thresh.xcf	31
img102-1_thresh xcf	32
img51-3_thresh.xcf	32
img70-5_thresh.xcf	32
img106-4_thresh xcf	33
img25-2_thresh xcf	33
img53-8_thresh.xcf	33
img30-2_thresh xcf	34
img53-1_thresh.xcf	34
img56-1_thresh.xcf	34
img78-2_thresh xcf	34
img88-3_thresh xcf	34
img121-4_thresh.xcf	35
img127-1_thresh.xcf	35
img15-1_thresh xcf	35
img45-6_thresh.xcf	35
img90-1_thresh xcf	35
img138-2_thresh.xcf	36
img22-2_thresh xcf	36
img127-2_thresh xcf	37
img146-2_thresh xcf	37
img18-4_thresh xcf	38
img32-4_thresh xcf	38
img98-1_thresh xcf	38
img81-8_thresh xcf	39
img83-2_thresh xcf	39
img96-2_thresh.xcf	39
img134-2_thresh xcf	40
img153-2_thresh xcf	40
img3-1_thresh xcf	40
img37-5_thresh xcf	40
img41-2_thresh xcf	40
img67-3_thresh.xcf	40
img81-5_thresh xcf	40
img1-5_thresh.xcf	41
img123-3_thresh xcf	41
img135-1_thresh xcf	41
img138-1_thresh xcf	41
img34-1_thresh xcf	41
img63-2_thresh xcf	41
img70-4_thresh xcf	41
img8-1_thresh.xcf	41
img68-1_thresh xcf	42
img29-2_thresh xcf	43
img96-3_thresh xcf	43
bb-4_thresh xcf	45
img1-4_thresh.xcf	45
img60-2_thresh xcf	45
img69-3_thresh xcf	45
img77-2_thresh xcf	45
bb-2_thresh xcf	46
img110-2_thresh xcf	46
img15-2_thresh xcf	46
img52-2_thresh xcf	47

Name	# Retrievals
img53-7_thresh.xcf	47
img62-1_thresh.xcf	47
img131-1_thresh.xcf	48
img136-2_thresh.xcf	48
img70-3_thresh.xcf	48
img145-1_thresh.xcf	49
img44-2_thresh.xcf	49
img51-1_thresh.xcf	50
img7-2_thresh.xcf	50
img74-3_thresh.xcf	50
img100-2_thresh.xcf	51
img117-4_thresh.xcf	51
img29-1_thresh.xcf	51
img53-5_thresh.xcf	51
img21-1_thresh.xcf	52
img99-5_thresh.xcf	53
img155-4_thresh.xcf	54
img125-1_thresh.xcf	55
img59-3_thresh.xcf	55
img19-1_thresh.xcf	57
img24-4_thresh.xcf	57
img101-3_thresh.xcf	58
img110-3_thresh.xcf	58
img15-6_thresh.xcf	58
img25-1_thresh.xcf	58
img132-2_thresh.xcf	60
img135-2_thresh.xcf	60
img112-1_thresh.xcf	61
img87-3_thresh.xcf	61
img108-2_thresh.xcf	63
img127-3_thresh.xcf	63
img45-1_thresh.xcf	65
img20-1_thresh.xcf	67
img30-1_thresh.xcf	67
img124-3_thresh.xcf	68
img61-2_thresh.xcf	70
img4-2_thresh.xcf	73
img69-4_thresh.xcf	73
img137-3_thresh.xcf	74
img21-2_thresh.xcf	74
img37-1_thresh.xcf	74
img5-1_thresh.xcf	76
img53-2_thresh.xcf	77
img21-7_thresh.xcf	80
img15-9_thresh.xcf	87

APPENDIX D

C Code:

1. b_dist.c: line-angle distance calculation
2. b_dist.h: line-angle distance calculation
3. b_dist3.c: centroid-radii distance calculation
4. b_dist3.h: centroid-radii distance calculation
5. calculate_c_distance.pgc: centroid-radii distance calculation driver
6. calculate_distance.pgc: line-angle distance calculation driver
7. centroid.pgc: centroid-radii calculations (GIMP plug-in)
8. erodematrix.c: Minkowski erosion (GIMP plug-in)
9. find_thresh.pgc: preprocessing (GIMP plug-in)
10. line-detect.pgc: line detection (GIMP plug-in)
11. sel2mask.c: selection-to-mask for preprocessing (GIMP plug-in)
12. cluster_analysis.pgc: cluster_analysis

```
/* b_dist.c */

/* Calculate the Bhattacharyya distance between two histograms */

/* Formula: -ln (sum over i) (sumover j) sqrt(h1[i][j] x h2[i][j])

   where h1 and h2 are normalized histograms

*/

#include <stdio.h>
#include <math.h>
#include "b_dist.h"


int read_histogram(char *hist_name, int hist[36][12], char *img_name)
{
    FILE *hist_file;
    int line_count=0;
    int items_read=0;
    char buf[100]={0};

    hist_file = fopen(hist_name, "r");
    if(! hist_file)
    {
      printf("Failed to open%s\n", hist_name);
      return 0;
    }

    while (!feof(hist_file) && (line_count < 36))
    {

        fgets(buf, 100, hist_file);
          items_read = sscanf(buf,
"%d%*c%d%*c%d%*c%d%*c%d%*c%d%*c%d%*c%d%*c%d%*c%d%*c%d%*c%d",
                              &hist[line_count][0],
                              &hist[line_count][1],
```

```
                                &hist[line_count][2],
                              &hist[line_count][3],
                              &hist[line_count][4],
                              &hist[line_count][5],
                              &hist[line_count][6],
                              &hist[line_count][7],
                              &hist[line_count][8],
                              &hist[line_count][9],
                              &hist[line_count][10],
                              &hist[line_count][11]);
          line_count++;

   }

   if(hist_file)
   {
     fgets(img_name, 100, hist_file);
     fclose(hist_file);
   }

   return line_count;
}

int sum_histogram(int hist[36][12], char *hist_name, char *img_name)
{
     int i, j, sum, num_zeroes;

     sum = 0;
     num_zeroes = 0;

     for(i=0; i<36; i++)
     {
          for(j=0; j<12; j++)
          {
            sum +=  hist[i][j];
            if(!hist[i][j])
            num_zeroes++;
```

```
            }
        }

        //printf("Number of zeroes in histogram %s (%s): %d\n",
hist_name,img_name, num_zeroes);

        return sum;
}

void print_histogram(int hist[36][12])
{
        int i, j;

        printf("Histogram contents:\n");
        for(i=0; i<36; i++)
        {
                for(j=0; j<12; j++)
                {
                   if(j<11)
                    printf("%d ", hist[i][j]);
                   else
                    printf("%d\n", hist[i][j]);
                }
        }
}


double calculate_b_dist(char *hist1, char *hist2)
{
    int  A[36][12];
    int  B[36][12];
    int sumA;
    int sumB;
    int i,j;
    int histogram_success;
    double difference_sum = 0;
    double result=0;
```

```
    char imgA_name[100]={0};
    char imgB_name[100]={0};

    histogram_success = read_histogram(hist1, A, imgA_name);
    if(!histogram_success)
    {
      printf("Unsuccessful read of histogram %s\n", hist1);
      return -1;
    }
    sumA = sum_histogram(A, hist1, imgA_name);


    histogram_success = read_histogram(hist2, B, imgB_name);
    if(!histogram_success)
    {
      printf("Unsuccessful read of histogram %s\n", hist2);
      return -1;
    }
    sumB = sum_histogram(B, hist2, imgB_name);

    for(i=0; i<36; i++)
    {
       for(j=0; j<12; j++)
       {
        difference_sum += sqrt(((double)A[i][j]/(double)sumA) *
                          ((double)B[i][j]/(double)sumB));
       }
    }


    result = -1 * log(difference_sum);

    return result;
}
```

This page is intentionally blank.

```
/* b_dist.h */

#ifndef B_DIST
#define B_DIST

double calculate_b_dist(char *hist1, char *hist2);

#endif
```

```
/* b_dist3.c */

/* Calculate the Bhattacharyya distance between the two centroid radii */

/* Formula: (sum over i) sqrt(h1[i] x h2[i])

   where h1 and h2 are normalized centroids

*/

#include <stdio.h>
#include <math.h>
#include "b_dist3.h"

#define NUM_ANGLES 32

int read_distances(char *dist_name, double dist[NUM_ANGLES],
                                        int empty[NUM_ANGLES], char *img_name)
{
   FILE *dist_file;
   int line_count=0;
   int items_read=0;
   char buf[100]={0};

   dist_file = fopen(dist_name, "r");
   if(! dist_file)
   {
    printf("Failed to open%s\n", dist_name);
    return 0;
   }
```

```
  while (!feof(dist_file) && (line_count < NUM_ANGLES))
  {
            fgets(buf, 100, dist_file);
      items_read = sscanf(buf, "%d%*c%lf",
                          &empty[line_count], &dist[line_count]);
      line_count++;
  }

  if(dist_file)
  {
   fgets(img_name, 100, dist_file);
   fclose(dist_file);
  }

  return line_count;
}

double sum_distances(double dist[NUM_ANGLES], int empty[NUM_ANGLES],
                     char *dist_name, char *img_name, int *num_empty)
{
   int i, j, empty_count;
   double sum;

   empty_count = sum = 0;

   for(i=0; i<NUM_ANGLES; i++)
   {
          if(!empty[i])
             sum +=  dist[i];
```

```
            else
                empty_count++;
    }

    *num_empty = empty_count;

    return sum;
}

void print_distances(double dist[NUM_ANGLES], int empty[NUM_ANGLES])
{
    int i, j;

    printf("Distances:\n");
    for(i=0; i<NUM_ANGLES; i++)
    {
      printf("%f %d\n", dist[i], empty[i]);
    }
}


struct dist_info *calculate_b_dist3(char *dist1, char *dist2)
{
   double  A[NUM_ANGLES];
   int A_empty[NUM_ANGLES];
   double  B[NUM_ANGLES];
   int B_empty[NUM_ANGLES];
   double sumA;
   double sumB;
   int i,j,n, empty_A, empty_B;
```

```
int distances_success, best_angle_index;
double difference_sum = 0;
double result;
char imgA_name[100]={0};
char imgB_name[100]={0};
double smallest_distance;
int empty_compare_sum;

struct dist_info *distance;

distance = (struct dist_info *)malloc(sizeof(struct dist_info));
smallest_distance=500;

distances_success = read_distances(dist1, A, A_empty, imgA_name);
if(!distances_success)
{
        printf("Unsuccessful read of distances %s\n", dist1);
        distance->distance = -1;
        distance->index = -1;
        return distance;
}

sumA = sum_distances(A, A_empty, dist1, imgA_name, &empty_A);

distances_success = read_distances(dist2, B, B_empty, imgB_name);
if(!distances_success)
{
        printf("Unsuccessful read of distances %s\n", dist2);
        distance->distance = -1;
        distance->index = -1;
```

```
            return distance;
    }

    sumB = sum_distances(B, B_empty, dist2, imgB_name, &empty_B);

    for(j=0; j<NUM_ANGLES; j++)
    {
      difference_sum=0;
      empty_compare_sum = 0;
      for(i=0; i<NUM_ANGLES; i++)
      {
            n = i+j;
            if(n >= NUM_ANGLES)
              n = n-NUM_ANGLES;
       if((!A_empty[i]) && (!B_empty[n]))
       {
            difference_sum += (((double)A[i]*1000/(double)sumA) -
                                    ((double)B[n]*1000/(double)sumB))
*
                                    (((double)A[i]*1000/(double)sumA)
-

((double)B[n]*1000/(double)sumB));
       }
            else
              empty_compare_sum++;
      }

      if (empty_compare_sum > NUM_ANGLES/2)
      {
```

```
                result = 500;
                best_angle_index = 0;
        }
        else
          result = sqrt(difference_sum);

        if(result < smallest_distance)
        {
                smallest_distance = result;
                best_angle_index = n;
        }
    }

    distance->distance = smallest_distance;
    distance->index = best_angle_index;

    return distance;
}
```

```
/* b_dist3.h */

#ifndef B_DIST3
#define B_DIST3

struct dist_info{
   double distance;
   int index;
};

struct dist_info *calculate_b_dist3(char *hist1, char *hist2);

#endif
```

```
/* calculate_c_distance.pgc */

/* Driver file for calculation of Euclidean distances
  between centroid radii distances in the glyph database*/

/* This version has no image type info */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "b_dist3.h"

//#define PRINT_LONG
//#define DEBUG
//#define DEBUG_SORT

struct img_info{
   double distance;
   int img_num;
};

struct distribution_info{
   char name[20];
   int retrievals;
};

struct retrieval_info{
   float sum;
   int best;
   char name[20];
```

```
  int type_num;
};

void sort_by_density(int query_image, int *zero_list, int num_zeroes);

int check_db_error(struct sqlca sqlerrstruct)
{
  if(sqlerrstruct.sqlcode != 0)
  {
    if(sqlerrstruct.sqlcode == 100)
      printf("End of cursor/no data found\n");
    else
    {
      printf("SQL error number %d:\n", (int)sqlerrstruct.sqlcode);
      printf("Line %d: Error: %s\n", (int)sqlerrstruct.sqlerrm.sqlerrml,
                                                    sqlerrstruct.sqlerrm.sqlerrmc);
    }
  }

  return (int)sqlerrstruct.sqlcode;
}

void check_error(int err_num, char *msg)
{

  if(err_num)
  {
    printf("Error %d: %s\n", err_num, msg);
    if(err_num < 0)
      exit(1);
  }
```

```
  #ifdef DEBUG
   else
     printf("%s\n", msg);
  #endif
}

void connect_to_db(struct sqlca sqlerrstruct)
{
   int err_num;

   exec sql connect to glyph;
   err_num = check_db_error(sqlerrstruct);
   check_error(err_num, "Connect to db");
}

void disconnect_from_db(struct sqlca sqlerrstruct)
{
   int err_num;

   exec sql disconnect all;
   err_num = check_db_error(sqlerrstruct);
   check_error(err_num, "Disconnecting from db");
}

int comp(const void *a, const void *b)
{
   double difference;

   difference = ((struct img_info *)a)->distance - ((struct img_info *)b)->distance;
   if(difference < 0)
     return -1;
```

```
    else if(difference)
      return 1;
    else
      return 0;
}

int get_num_images(struct sqlca sqlerrstruct)
{
    int error_num;

    exec sql begin declare section;
    int num_images;
    exec sql end declare section;

    exec sql select count(*) into :num_images from images;
    error_num = check_db_error(sqlca);
    check_error(error_num, "Getting number of images");

    return num_images;
}

void get_short_name(char *long_name, char* short_name)
{
  int name_len, i, slash_index;

  // Clear out short name string holder
  for(i=0; i<(strlen(short_name)); i++)
    short_name[i] = 0;

  slash_index=0;
  name_len = strlen(long_name);
```

```
  for(i=name_len; i>=0; i--)
  {
    if(long_name[i] == '/')
    {
      slash_index = i;
      break;
    }
  }

  strcpy(short_name, long_name+i+1);
  #ifdef DEBUG
            printf("Short name for retrieval is %s\n", short_name);
  #endif
}

void get_jpg_name(char *short_name, char* jpg_name)
{
  int name_len, i, dot_index;

  // Clear out short name string holder
  for(i=0; i<(strlen(jpg_name)); i++)
   jpg_name[i] = 0;

  dot_index=0;
  name_len = strlen(short_name);
  for(i=name_len; i>=0; i--)
  {
    if(short_name[i] == '.')
    {
      dot_index = i;
      break;
```

```
    }
  }

  strncpy(jpg_name, short_name, i+1);
  strcpy(jpg_name+i+1, "jpg");
}

void show_image_distribution(int max_img_index,
                                                    struct distribution_info
*match_distribution,
                                                    int num_no_retrievals)
{
  int i, j;
  FILE *stats_file;
  char *file_name="/tmp/stats.txt";

  stats_file=fopen(file_name, "a");
  if(! stats_file)
    printf("Unable to open %s for writing\n", file_name);

  fprintf(stats_file, "<BR>");
  fprintf(stats_file, "<H2>Image Retrieval Distribution</H2>\n<PRE>");

  // Show distribution of all image retrievals
  printf("Name\t\t # Retrievals\n");
  fprintf(stats_file, "Name\t\t # Retrievals\n");
  printf("----\t\t ------------\n");
  fprintf(stats_file, "----\t\t ------------\n");
  for(i=0; i<max_img_index; i++)
  {
          if(match_distribution[i].retrievals > -1)
```

```
            {
        printf("%s\t%8d ", match_distribution[i].name,
                                        match_distribution[i].retrievals);
        fprintf(stats_file, "%s\t%8d ", match_distribution[i].name,
                                        match_distribution[i].retrievals);
              for(j=0; j<match_distribution[i].retrievals; j+=5)
              {
                        printf("*");
                        fprintf(stats_file, "*");
              }
              printf("\n");
              fprintf(stats_file, "\n");
            }
   }

   fprintf(stats_file, "\nNumber of images for which there were no retrievals: %d\n",
num_no_retrievals);
   printf("\nNumber of images for which there were no retrievals: %d\n",
num_no_retrievals);

   fprintf(stats_file, "</PRE>");
   fclose(stats_file);
}

void print_stats()
{
   FILE *stat_file, *stat_temp;
   char buf[100]={0};

   char *file_name="/tmp/retrieval/statistics.html";
   char *temp_name="/tmp/stats.txt";
```

```
stat_temp=fopen(temp_name, "r");
stat_file=fopen(file_name, "w");

if(! stat_temp)
{
 printf("Couldn't open %s file for reading\n", temp_name);
 return;
}
if(! stat_file)
{
 printf("Couldn't open %s file for writing\n", file_name);
 return;
}


fprintf(stat_file,"<HTML><HEAD><TITLE>Image Retrieval
Statistics</TITLE></HEAD>\n");
fprintf(stat_file,"<BODY BGCOLOR=#FFFFFF><CENTER><H1>Image Retrieval
Statistics</H1></CENTER>\n");


while(!feof(stat_temp))
{
  fgets(buf, 100, stat_temp);
  fprintf(stat_file, "%s", buf);
}

fprintf(stat_file, "</BODY></HTML>");

if(stat_temp)
```

```
    fclose(stat_temp);
  if(stat_file)
    fclose(stat_file);
}

void print_header_no_type(char *jpg_name, int image_number)
{
  char file_name[30]={0};
  char short_name[20]={0};
  char *index_file="/tmp/retrieval/index.html";
  FILE *ret_file, *ind_file;

  sprintf(short_name, "img_%d.html", image_number);
  sprintf(file_name, "/tmp/retrieval/%s", short_name);

  ret_file=fopen(file_name, "w");
  if(! ret_file)
  {
   printf("Failed to open %s for writing\n", file_name);
   return;
  }

  ind_file=fopen(index_file, "a");
  if(! ind_file)
  {
   printf("Failed to open %s for writing\n", index_file);
   return;
  }

  fprintf(ind_file, "<A HREF='%s'>%s</A><BR>\n", short_name, short_name);
```

```
   fprintf(ret_file,"<HTML><HEAD><TITLE>Top 20 Retrievals for Image
%d</TITLE></HEAD>\n", image_number);
   fprintf(ret_file,"<BODY BGCOLOR=#FFFFFF><CENTER><H1>Top 20 Retrievals
for Image %d</H1>\n", image_number);

   fprintf(ret_file,"<H2>Image %d (%s)\n", image_number, jpg_name);
   fprintf(ret_file,"</H2>\n<IMG SRC='../masked_jpg/%s'>\n", jpg_name);

   fclose(ind_file);
   fclose(ret_file);

}

void index_header()
{
  FILE *ind_file;
  char *index_file="/tmp/retrieval/index.html";

  ind_file=fopen(index_file, "w");
  if(! ind_file)
  {
   printf("Failed to open %s for writing\n", index_file);
   return;
  }
   fprintf(ind_file,"<HTML><HEAD><TITLE>Retrieval of Petroglyph Images from an
Image Database</TITLE></HEAD>\n");
   fprintf(ind_file,"<BODY BGCOLOR=#FFFFFF><CENTER><H1>Retrieval of
Petroglyph Images from an Image Database</H1>\n");
   fprintf(ind_file,"<H2>Retrieval Methodology and Statistics</H2>\n");

   fprintf(ind_file, "<BR><BR><A HREF='retrieval.html'>Retrieval
```

```
methodology</A>");
   fprintf(ind_file, "<BR><BR><A HREF='statistics.html'>Retrieval statistics</A>");
   fprintf(ind_file,"<H2>Retrievals</H2>\n");
   fclose(ind_file);
}

void finish_file(int image_number)
{
  char file_name[30]={0};
  FILE *ret_file, *ind_file;
  char *index_file="/tmp/retrieval/index.html";

  sprintf(file_name, "/tmp/retrieval/img_%d.html", image_number);

  ret_file=fopen(file_name, "a");
  if(! ret_file)
  {
   printf("Failed to open %s for writing\n", file_name);
   return;
  }
  ind_file=fopen(index_file, "a");
  if(! ind_file)
  {
   printf("Failed to open %s for writing\n", index_file);
   return;
  }

   fprintf(ret_file, "<BR><BR><A HREF='index.html'>BACK</A>");
   fprintf(ret_file, "</BODY></HTML>");
   fprintf(ind_file, "</BODY></HTML>");
   fclose(ret_file);
```

```
    fclose(ind_file);
}
void print_retrieval_no_type(int query_image, char *jpg_name,
                                        double distance, int ret_num)
{
  char file_name[30]={0};
  FILE *ret_file;

  sprintf(file_name, "/tmp/retrieval/img_%d.html", query_image);

  ret_file=fopen(file_name, "a");
  if(! ret_file)
  {
   printf("Failed to open %s for writing\n", file_name);
   return;
  }

   fprintf(ret_file, "<H2>Retrieval %d: %s</H2>\n", ret_num, jpg_name);
   fprintf(ret_file, "<IMG SRC='../masked_jpg/%s'><BR>\n", jpg_name);
   fprintf(ret_file, "distance: %f\n", (float)distance);
   fclose(ret_file);
}

int main(void)
{
   double distance, *matrix, same_distance;
   int i, err_num, j, index1, index2, k, num_matches;
   int query_image;
   int num_images, matrix_base;
   struct distribution_info *match_distribution;
   char file1[30], file2[30];
```

```
    struct img_info *img_list, *tmp_img_list;
    char short_name[20] = {0};
    char jpg_name[20] = {0};
    int same_count, *same_list, l, last_sorted, num_no_retrievals;

    exec sql begin declare section;
    int image_nums[500];
    char long_name[100];
    int image_number;
    int num_all_images;
    int max_img_index;
    exec sql end declare section;

    exec sql include sqlca;


    for(i=0; i<500; i++)
                image_nums[i] = -1;

    connect_to_db(sqlca);
    num_images = get_num_images(sqlca);

    printf("Found %d images in the images table\n", num_images);

    exec sql select image_num into :image_nums from images;

    err_num = check_db_error(sqlca);
    check_error(err_num, "Getting image numbers");
#ifdef DEBUG
    for(i=0; i<500; i++)
    {
```

```
    printf("image_nums[%d]: %d\n", i, image_nums[i]);
    if(image_nums[i] == 430)
            printf("Index into image_nums for 430: %d\n", i);
  }
#endif

  // Prepare for some statistics:
  // How many images of each type in the db?
  // How many images total?
  // What's the max image index?

  exec sql select count(*) into :num_all_images from images;
  err_num = check_db_error(sqlca);
  check_error(err_num, "Getting number of images");

  printf("Number of images: %d\n", num_all_images);

  exec sql select MAX(image_num) into :max_img_index from images;
  err_num = check_db_error(sqlca);
  check_error(err_num, "Getting highest image index");

  // Memory allocation section
  matrix = malloc(sizeof(double[num_images * num_images]));
  img_list = malloc(sizeof(struct img_info[num_images * num_images]));
  tmp_img_list = malloc(sizeof(struct img_info[num_images]));
  match_distribution = malloc(sizeof(struct distribution_info[max_img_index]));

  // Initialize structs for statistics
  for(i=0; i<max_img_index; i++)
  {
   match_distribution[i].retrievals=-1;
```

```
 strcpy(match_distribution[i].name,"");
}

for(i=0; i<num_images; i++)
{
            for(j=0; j<num_images; j++)
            {
              index1 = (i * num_images)+j;
              matrix[index1]=-1;
            }
}

num_no_retrievals=0;

// Populate n x n matrix of distances

for(matrix_base=0; matrix_base < num_images; matrix_base++)
{
  for(i=0; i<num_images; i++)
  {
             // Measurements are symmetric across the diagonal
             index1 = (matrix_base * num_images)+i;
             index2 = (i * num_images)+matrix_base;
             if(matrix[index1] == -1)
             {
               if(matrix_base == i)
               {
                 matrix[index1] = 0;
                          img_list[index1].img_num = image_nums[matrix_base];
                          img_list[index1].distance = 0;
      }
```

```
                else
                {
                  sprintf(file1, "/tmp/distances/dist_%d", image_nums[matrix_base]);
                  sprintf(file2, "/tmp/distances/dist_%d", image_nums[i]);
                  distance = calculate_b_dist3(file1, file2);
                        if(distance == -1)
                        {
                          printf("Error getting distance\n");
                          return 1;
                        }
          matrix[index1] = distance;
          matrix[index2] = distance;
                        img_list[index1].img_num = image_nums[i];
                        img_list[index1].distance = distance;
                        img_list[index2].img_num = image_nums[matrix_base];
                        img_list[index2].distance = distance;
                }
      }
    }
  }

  // For each image, sort the distance results
  for(i=0; i<num_images; i++)
  {
     for(j=0; j<num_images; j++)
     {
               index1 = i*num_images + j;
               tmp_img_list[j] = img_list[index1];
             }
            qsort((void *)tmp_img_list, (size_t)num_images, (size_t)(sizeof(*
tmp_img_list)), comp);
```

```
            for(j=0; j<num_images; j++)
            {
              index1 = i*num_images + j;
              img_list[index1] = tmp_img_list[j];
            }

   }

   index_header();
   last_sorted = 0;
   // Get statistics on retrieval of 20 closest images
   for(i=0; i<num_images; i++)
   {
            num_matches=0;
            #ifdef PRINT_LONG
            printf("\nTop 20 distances from image_num %d:\n", image_nums[i]);
            #endif
     for(j=0; j<21; j++)
            {
       index1= (i * num_images) + j;
              image_number = img_list[index1].img_num;

              if((j==0) && (image_number != image_nums[i]))
              {
                        printf("Found an erroneous entry: image_number=%d
image_nums[i]=%d\n", image_number, image_nums[i]);
                        printf("Need to look from %d to %d\n", index1+1,
num_images*(i+1));
                        printf("Looking to swap image numbers\n");
                        // Find the index of the image
                        image_number = image_nums[i];
```

```
                for(k=index1+1; k<num_images*(i+1); k++)
                {
                  printf("Searching: img_num = %d\n", img_list[k].img_num);
                  if(img_list[k].img_num == image_number)
                  {
                        printf("Found index of image at %d\n", k);
                        printf("Moving other images down\n");
                        for(l=0; l<k; l++)
                        {
                          img_list[k-l] = img_list[k-l-1];
                        }
                        img_list[index1].img_num = image_nums[i];
                        break;
                  }
                }
        }

        if(j>last_sorted)
        {
          // Sort by density for those images with identical distance
// from query -- look ahead to find them
          #ifdef DEBUG_SORT
                printf("Looking for identical distances to sort\n");
          #endif
          same_count = 1;
          same_distance = img_list[index1+j].distance;
          for(k=index1+1+j; k<num_images*(i+1); k++)
          {
            if(img_list[k].distance == same_distance)
                  same_count++;
            else
```

```
                    break;
                }
                if(same_count > 2)
                {
                        #ifdef DEBUG_SORT
                         printf("Sorting same distances in retrievals from %d\n",
image_nums[i]);
                        #endif
                  last_sorted = index1+j+same_count-1;
                  same_list = malloc(sizeof(int[same_count]));
                  for(l=0; l<same_count; l++)
                  {
                        k=index1+1+l+j;
                        if(img_list[k].distance > same_distance)
                          break;
                        same_list[l]= img_list[k].img_num;
                  }
                  sort_by_density(image_nums[i], same_list, same_count);
                  for(l=0; l<same_count; l++)
                  {
                        k=index1+1+l+j;
                        if(img_list[k].distance > same_distance)
                          break;
                        img_list[k].img_num = same_list[l];
                  }
                  free(same_list);
                }
              }

        for(k=0; k<(strlen(long_name)); k++)
          long_name[k] = 0;
```

```
        exec sql select image_name into :long_name from images where image_num =
:image_number ;
        check_error(err_num, "Getting image name");
                strcpy(short_name, "");
        get_short_name(long_name, short_name);
                strcpy(jpg_name, "");
        get_jpg_name(short_name, jpg_name);
                #ifdef DEBUG
                   printf("image_number: %d j: %d\n", image_number, j);
                        printf("jpg_name: %s\n", jpg_name);
                        printf("Distance: %f\n", img_list[index1].distance);
                #endif
                if(j == 0)
                {
                  print_header_no_type(jpg_name, image_number);
                  query_image = image_number;
                }
                else
                {
                  if(img_list[index1].distance < 500)
                          print_retrieval_no_type(query_image, jpg_name,
                                                  img_list[index1].distance, j);
                  else if(j==1)
                          num_no_retrievals++;

                }
                if(!strcmp(match_distribution[image_number-1].name,""))
                  strcpy(match_distribution[image_number-1].name,short_name);
                if(img_list[index1].distance < 500)
                  match_distribution[image_number-1].retrievals++;
              }
```

```
            finish_file(query_image);
    }


    show_image_distribution(max_img_index, match_distribution, num_no_retrievals);

    print_stats();

    disconnect_from_db(sqlca);

    // Memory deallocation
    free(matrix);
    free(img_list);
    free(tmp_img_list);
    free(match_distribution);

    return 0;
}

void sort_by_density(int query_image, int *zero_list, int num_zeroes)
{
    float density_diff;
    struct img_info *densities;
    int i, err_num;

    exec sql begin declare section;
    int image_number;
    int qry_image;
    float query_density;
    float zero_density;
    exec sql end declare section;
```

```
    exec sql include sqlca;

    densities = malloc(sizeof(struct img_info[num_zeroes]));
    qry_image = query_image;

    exec sql select density into :query_density from images where image_num =
:qry_image;
    err_num = check_db_error(sqlca);

    check_error(err_num, "Getting query image density");

    for(i=0; i<num_zeroes; i++)
    {
      image_number = zero_list[i];
      exec sql select density into :zero_density from images where image_num =
:image_number;
      density_diff = fabs(query_density - zero_density);
      densities[i].distance = density_diff;
      densities[i].img_num = image_number;
    }

    qsort((void *)densities, (size_t)num_zeroes, (size_t)(sizeof(* densities)), comp);

    for(i=0; i<num_zeroes; i++)
    {
      zero_list[i] = densities[i].img_num;
    }

    free(densities);
}
```

```
/* calculate_distance.pgc */

/* Driver file for calculation of Bhattacharyya distances
   between histograms in the glyph database*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "b_dist.h"

//#define PRINT_LONG

struct img_info{
    double distance;
    int img_num;
};

struct distribution_info{
    char name[20];
    int retrievals;
};

struct retrieval_info{
    float sum;
    int best;
    char name[20];
    int type_num;
};

int check_db_error(struct sqlca sqlerrstruct)
{
```

```
  if(sqlerrstruct.sqlcode != 0)
  {
    if(sqlerrstruct.sqlcode == 100)
      printf("End of cursor/no data found\n");
    else
    {
      printf("SQL error number %d:\n", (int)sqlerrstruct.sqlcode);
      printf("Line %d: Error: %s\n", (int)sqlerrstruct.sqlerrm.sqlerrml,
                                                    sqlerrstruct.sqlerrm.sqlerrmc);
    }
  }

  return (int)sqlerrstruct.sqlcode;
}

void check_error(int err_num, char *msg)
{

   if(err_num)
   {
     printf("Error %d: %s\n", err_num, msg);
     if(err_num < 0)
       exit(1);
   }
  #ifdef DEBUG
   else
     printf("%s\n", msg);
  #endif
}

void connect_to_db(struct sqlca sqlerrstruct)
```

```
{
   int err_num;

   exec sql connect to glyph;
   err_num = check_db_error(sqlerrstruct);
   check_error(err_num, "Connect to db");
}

void disconnect_from_db(struct sqlca sqlerrstruct)
{
   int err_num;

   exec sql disconnect all;
   err_num = check_db_error(sqlerrstruct);
   check_error(err_num, "Disconnecting from db");
}

int comp(const void *a, const void *b)
{
   double difference;

   difference = ((struct img_info *)a)->distance - ((struct img_info *)b)->distance;
   if(difference < 0)
     return -1;
   else if(difference)
     return 1;
   else
     return 0;
}

int get_num_images(struct sqlca sqlerrstruct)
```

```
{
   int error_num;

   exec sql begin declare section;
   int num_images;
   exec sql end declare section;

   exec sql select count(*) into :num_images from images;
   error_num = check_db_error(sqlca);
   check_error(error_num, "Getting number of images");

   return num_images;
}

void get_short_name(char *long_name, char* short_name)
{
  int name_len, i, slash_index;

  // Clear out short name string holder
  for(i=0; i<(strlen(short_name)); i++)
    short_name[i] = 0;

  slash_index=0;
  name_len = strlen(long_name);
  for(i=name_len; i>=0; i--)
  {
    if(long_name[i] == '/')
    {
      slash_index = i;
      break;
    }
```

```
  }

  strcpy(short_name, long_name+i+1);
}

int main()
{
   double distance, *matrix;
   int i, err_num, j, index1, index2, k, num_matches, num_to_match;
   int type_distribution, img_type_index;
   int num_images, matrix_base, *types_array, *type_nums_array;
   struct distribution_info *match_distribution;
   char file1[30], file2[30];
   struct img_info *img_list, *tmp_img_list;
   char short_name[20] = {0};
   char type_name[20] = {0};
   float percent_of_matches, percent_in_db;
   struct retrieval_info *retrieval_sums;

   exec sql begin declare section;
   int image_nums[500];
   char long_name[100];
   char img_type[20];
   int image_number, *img_type_counts, *img_type_nums;
   int num_img_types;
   int num_all_images;
   int img_type_num;
   int max_img_index;
   exec sql end declare section;

   exec sql include sqlca;
```

```
connect_to_db(sqlca);
num_images = get_num_images(sqlca);

printf("Found %d images in the images table\n", num_images);

exec sql select image_num into :image_nums from images;

err_num = check_db_error(sqlca);
check_error(err_num, "Getting image numbers");

// Prepare for some statistics:
// How many images of each type in the db?
// How many images total?
// What's the max image index?

exec sql select count(*) into :num_img_types from image_types;
err_num = check_db_error(sqlca);
check_error(err_num, "Getting number of image types");

exec sql select count(*) into :num_all_images from images;
err_num = check_db_error(sqlca);
check_error(err_num, "Getting number of images");

printf("Number of images: %d\n", num_all_images);

exec sql select MAX(image_num) into :max_img_index from images;
err_num = check_db_error(sqlca);
check_error(err_num, "Getting highest image index");

// Memory allocation section
```

```
   matrix = malloc(sizeof(double[num_images * num_images]));
   img_list = malloc(sizeof(struct img_info[num_images * num_images]));
   tmp_img_list = malloc(sizeof(struct img_info[num_images]));
   types_array = malloc(sizeof(int[num_img_types]));
   retrieval_sums=malloc(sizeof(struct retrieval_info[num_img_types]));
   type_nums_array = malloc(sizeof(int[num_img_types]));
   match_distribution = malloc(sizeof(struct distribution_info[max_img_index]));

   img_type_counts = types_array;
   img_type_nums = type_nums_array;

   // Get distribution of image types in db
   exec sql select count(img_type) into :img_type_counts from images group by
img_type ;
   err_num = check_db_error(sqlca);
   check_error(err_num, "Populating image type count");

   // Get list of type_nums
   exec sql select type_num into :img_type_nums from image_types order by type_num;
   err_num = check_db_error(sqlca);
   check_error(err_num, "Populating image type numbers");

#ifdef PRINT_LONG
   // Display type distribution
   for(i=0; i<num_img_types; i++)
     printf("Image type %d count %d\n", img_type_nums[i], img_type_counts[i]);
#endif


   // Initialize structs for statistics
   for(i=0; i<max_img_index; i++)
```

```
{
 match_distribution[i].retrievals=-1;
 strcpy(match_distribution[i].name,"");
}

for(i=0; i<num_img_types; i++)
{
 retrieval_sums[i].sum=0;
 retrieval_sums[i].best=0;
 retrieval_sums[i].type_num = img_type_nums[i];
 strcpy(retrieval_sums[i].name,"");
}

for(i=0; i<num_images; i++)
{
            for(j=0; j<num_images; j++)
            {
              index1 = (i * num_images)+j;
              matrix[index1]=-1;
            }
}

// Populate n x n matrix of distances

for(matrix_base=0; matrix_base < num_images; matrix_base++)
{
  for(i=0; i<num_images; i++)
  {
              // Measurements are symmetric across the diagonal
              index1 = (matrix_base * num_images)+i;
              index2 = (i * num_images)+matrix_base;
```

```
            if(matrix[index1] == -1)
            {
              if(matrix_base == i)
              {
                matrix[index1] = 0;
                        img_list[index1].img_num = image_nums[matrix_base];
                        img_list[index1].distance = 0;
      }
              else
              {
                sprintf(file1, "/tmp/histograms/hist_%d", image_nums[matrix_base]);
                sprintf(file2, "/tmp/histograms/hist_%d", image_nums[i]);
                distance = calculate_b_dist(file1, file2);
        matrix[index1] = distance;
        matrix[index2] = distance;
                        img_list[index1].img_num = image_nums[i];
                        img_list[index1].distance = distance;
                        img_list[index2].img_num = image_nums[matrix_base];
                        img_list[index2].distance = distance;
              }
    }
 }
}

// For each image, sort the distance results
for(i=0; i<num_images; i++)
{
   for(j=0; j<num_images; j++)
   {
            index1 = i*num_images + j;
            tmp_img_list[j] = img_list[index1];
```

```
                }
                qsort((void *)tmp_img_list, (size_t)num_images, (size_t)(sizeof(*
tmp_img_list)), comp);
                for(j=0; j<num_images; j++)
                {
                  index1 = i*num_images + j;
                  img_list[index1] = tmp_img_list[j];
                }

  }

  // Get statistics on retrieval of 20 closest images
  for(i=0; i<num_images; i++)
  {
                num_matches=0;
                #ifdef PRINT_LONG
                printf("\nTop 20 distances from image_num %d:\n", image_nums[i]);
                #endif
    for(j=0; j<21; j++)
                {
      index1= (i * num_images) + j;
                  image_number = img_list[index1].img_num;
      for(k=0; k<(strlen(long_name)); k++)
        long_name[k] = 0;
                  strcpy(img_type, "");
      exec sql select image_name, type_name, img_type into :long_name, :img_type,
:img_type_num from images, image_types where image_num = :image_number and
image_types.type_num = images.img_type;
      check_error(err_num, "Getting image name");
      get_short_name(long_name, short_name);
                  if(j == 0)
```

```
                {
                  num_to_match = img_type_num;
                  for(k=0; k<num_img_types; k++)
                            if(img_type_nums[k] == num_to_match)
                    {
                              img_type_index=k;
                              break;
                            }
                  type_distribution = img_type_counts[img_type_index];
                  strcpy(type_name, img_type);
                  if(!strcmp(retrieval_sums[img_type_index].name,""))
                            strcpy(retrieval_sums[img_type_index].name, img_type);
                }
                else
                {
                            if(img_type_num == num_to_match)
                      num_matches++;
                }
                if(!strcmp(match_distribution[image_number-1].name,""))
                  strcpy(match_distribution[image_number-1].name,short_name);
                match_distribution[image_number-1].retrievals++;
              #ifdef PRINT_LONG
                printf("%s  distance: %f type: %s \n",
                                        short_name, (float)img_list[index1].distance,
img_type);
              #endif
              }
              percent_of_matches=(float)num_matches/20;
              percent_in_db=(float)type_distribution/(float)num_all_images;
              retrieval_sums[img_type_index].sum+=percent_of_matches;
              if(percent_of_matches >= percent_in_db+0.05)
```

```
                retrieval_sums[img_type_index].best++;
            #ifdef PRINT_LONG
            printf("%s is %f of image database (%d of %d) and %f of top 20
matches\n", type_name, percent_in_db,type_distribution, num_all_images,
percent_of_matches);
            #endif
   }

   // Show distribution of all image retrievals
   for(i=0; i<max_img_index; i++)
   {
            if(match_distribution[i].retrievals > -1)
            {
            #ifdef PRINT_LONG
        printf("%s retrieved %d times\n", match_distribution[i].name,
                                        match_distribution[i].retrievals);
            #endif
            }
   }

   // Show statistics by image type for retrieval
   printf("    \t %15s         Average \t      \t\t# Better  \n", "\t");
   printf("Type\t %15s\t  Retrieval\t Random\t \tThan Random\n", "Name");
   printf("----\t %15s\t  ---------\t ------\t \t-----------\n", "----");
   for(i=0; i<num_img_types; i++)
   {
    printf("%d\t %15s\t %f\t %f\t %d/%d\n", retrieval_sums[i].type_num,
retrieval_sums[i].name,
retrieval_sums[i].sum/(float)img_type_counts[i],(float)img_type_counts[i]/(float)num_all
_images, retrieval_sums[i].best, img_type_counts[i]);
   }
```

```
    disconnect_from_db(sqlca);

    // Memory deallocation
    free(matrix);
    free(img_list);
    free(tmp_img_list);
    free(types_array);
    free(type_nums_array);
    free(retrieval_sums);
    free(match_distribution);

    return 0;
}
```

```
/* Centroid-radii plug-in for the GIMP -- Version 0.1
 * Copyright (C) 2001 Cindy Huyser <chuyser@io.com>
 *
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 *
 * The GNU General Public License is also available from
 * http://www.fsf.org/copyleft/gpl.html
 *
 * CHANGELOG:
 *
 *
 *
 */

#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <math.h>
```

```
#include <string.h>
#include <sys/types.h>
#ifdef HAVE_UNISTD_H
#include <unistd.h>
#endif

#include <gtk/gtk.h>

#include <libgimp/gimp.h>
#include <libgimp/gimpui.h>

#include <libgimp/stdplugins-intl.h>
#define NUM_ANGLES 64

#define DEBUG

struct point{
  glong x;
  glong y;
};

GimpDrawable *drawable;


/* Declare local functions. */
static void query (void);
static void run   (gchar     *name,
                   gint       nparams,
                   GimpParam  *param,
                   gint      *nreturn_vals,
                   GimpParam **return_vals);
```

```
static double euclidean_distance(struct point *a, struct point *b)
{
  if((b->x == -1)||(b->y==-1)||(a->x==-1)||(a->y==-1))
                return -1;
  return sqrt(((a->x - b->x)*(a->x - b->x)) + ((a->y - b->y) * (a->y - b->y)));
}

static void find_max_intersection(gdouble angle, struct point *centroid,
                                                gint *side, gint begin, gint end,
                                                struct point *intersection,
                                                gint search_type, gint quadrant,
                                                gdouble *distance)
{
   // m = tan(theta), where theta is the angle of interest
   // y = m(x - centroid->x) + centroid->y
   //
   // Search left from 3pi/4 to 5pi/4
   // Search right from 0 to pi/4 and 7pi/4 to 0
   // for searches left or right, look for y where x satisfies
   // x = (y-centroid->y)/m + centroid->x

  gint i, sign;
  gdouble m, wanted_x, wanted_y, max_distance;
  gboolean found=FALSE;
  struct point temp_intersection;

  sign = 1;

  max_distance = -1000;
  printf("Looking for intersection with line through centroid and angle %f pi\n",
```

```
angle/M_PI);
  printf("Search_type: %d\n", search_type);
  m = tan(angle);
  for(i=begin; i<=end; i++)
  {
    if(search_type == 0) // left or right
    {
              wanted_y = m * (side[i] - centroid->x) + centroid->y;
              wanted_x =  (i - centroid->y)/m + centroid->x;
     #ifdef VERBOSE_DEBUG
              {
       printf("\nChecking intersection with col %d row %d\n", side[i], i);
               printf("wanted_y: %f actual y: %d\n", wanted_y, side[i]);
               printf("wanted_x: %f actual x: %d\n", wanted_x, i);
              }
              #endif

              if(((side[i]<wanted_x+1) && (side[i]>wanted_x-1)) ||
               ((i<wanted_y+1) && (i>wanted_y-1)))
              {
                found=TRUE;
                #ifdef DEBUG
                printf("Found intersection: x=%d, y=%d\n", side[i], i);
                #endif
                temp_intersection.x=side[i];
                temp_intersection.y=i;
                switch(quadrant)
                {
                        case 1: if ((temp_intersection.y < centroid->y) &&
                                        (temp_intersection.x < centroid->x))
                                       sign = -1;
```

```
                                    break;
                            case 2: if ((temp_intersection.y < centroid->y) &&
                                        (temp_intersection.x > centroid->x))
                                    sign = -1;
                                    break;
                            case 3: if ((temp_intersection.y > centroid->y) &&
                                        (temp_intersection.x > centroid->x))
                                    sign = -1;
                                    break;
                            case 4: if ((temp_intersection.y > centroid->y) &&
                                        (temp_intersection.x < centroid->x))
                                    sign = -1;
    }
    *distance = sign * euclidean_distance(centroid, &temp_intersection);
              if(*distance > max_distance)
              {
                max_distance = *distance;
                intersection->x=side[i];
                intersection->y=i;
              }

            }
}
else  // Search top/bottom
{
            wanted_x =  (side[i] - centroid->y)/m + centroid->x;
            wanted_y = m * (i - centroid->x) + centroid->y;
 #ifdef VERBOSE_DEBUG
            {
    printf("\nChecking intersection with col %d row %d\n", i, side[i]);
              printf("wanted_x: %f actual x: %d\n", wanted_x, i);
```

```
        printf("wanted_y: %f actual y: %d\n", wanted_y, side[i]);
      }
        #endif
      if(((i<wanted_x+1) && (i>wanted_x-1)) ||
       ((side[i]<wanted_y+1) && (side[i]>wanted_y-1)))
      {
        found=TRUE;
        #ifdef DEBUG
        printf("Found intersection: y=%d, x=%d\n", side[i], i);
        #endif
        temp_intersection.y=side[i];
        temp_intersection.x=i;
        switch(quadrant)
        {
                case 1: if ((temp_intersection.y < centroid->y) &&
                                (temp_intersection.x < centroid->x))
                                sign = -1;
                                break;
                case 2: if ((temp_intersection.y < centroid->y) &&
                                (temp_intersection.x > centroid->x))
                                sign = -1;
                                break;
                case 3: if ((temp_intersection.y > centroid->y) &&
                                (temp_intersection.x > centroid->x))
                                sign = -1;
                                break;
                case 4: if ((temp_intersection.y > centroid->y) &&
                                (temp_intersection.x < centroid->x))
                                sign = -1;
}
*distance = sign * euclidean_distance(centroid, &temp_intersection);
```

```
                if(*distance > max_distance)
                {
                  printf("New max distance:%f\n", *distance);
                  max_distance = *distance;
                  intersection->y=side[i];
                  intersection->x=i;
                }
              }
    }
  }
  if(!found)
  {
    printf("Intersection not found\n");
    intersection->x=-1;
    intersection->y=-1;
  }
}

static void interpolate_missing_distances(gdouble *distances,
                                                    gint *empty)
{
  struct interpol{
              gint first_missing;
              gint num_missing;
              struct interpol *next;
  };

  gint last_good, i, last_missing, count;
  gdouble first_value, last_value, factor;
  struct interpol *head, *node, *new_node;
  gint first_wrap_num, last_wrap_num, denominator;
```

```
gboolean begin_wrap, end_wrap;

head=node=new_node=NULL;

last_missing=NUM_ANGLES;

printf("Distances before interpolation:\n");
for(i=0; i<NUM_ANGLES; i++)
   printf("%d: %f\n", i, (float)distances[i]);


// Build list of interpolations

for(i=0; i<NUM_ANGLES; i++)
{
            if(empty[i])
            {
              if((i==0)||(last_good == i-1))
              {
                printf("Found an area to interpolate at %d\n", i);
                new_node = malloc(sizeof(struct interpol));
                if(!head)
                {
                            head = new_node;
                            node = head;
                            node->next = NULL;
                }
                else
                {
                            node->next = new_node;
                            node = node->next;
```

```
                        node->next = NULL;
                        new_node = NULL;

                    }
                    node->first_missing = i;
                    node->num_missing = 1;
                }
                else
                {
                  node->num_missing++;
                }
                last_missing = i;
            }
            else
            {
                if(last_missing == i-1)
                {
                        printf("Finished with area to interpolate at %d\n", i);
                }
                last_good = i;
            }
}

node = head;
while(node)
{
            begin_wrap = end_wrap = FALSE;

            if(node->first_missing > 0)
            {
                first_value = distances[node->first_missing -1];
```

```
            first_wrap_num = 0;
          }
          else
          {
            begin_wrap = TRUE;
            for(i=NUM_ANGLES-1; i>=node->first_missing+node->num_missing;i--
)
            {
                    if(!empty[i])
                    {
                      first_value = distances[i];
                      first_wrap_num = NUM_ANGLES-i-1;
                      break;
                    }
            }
          }
          printf("First missing: %d num_missing: %d\n", node->first_missing, node-
>num_missing);
          if((node->first_missing + node->num_missing) < NUM_ANGLES)

          {
            last_value = distances[node->first_missing + node->num_missing];
            last_wrap_num = 0;
          }
          else
          {
            printf("Found node with last angle value empty\n");
            end_wrap = TRUE;
            for(i=0; i<node->first_missing + node->num_missing; i++)
            {
                    if(!empty[i])
```

```
                        {
                          last_value = distances[i];
                          last_wrap_num = i;
                          break;
                        }
                  }
                }

                denominator = node->num_missing + first_wrap_num + last_wrap_num +
1;
                factor = (last_value - first_value)/(gdouble)denominator;

      printf("About to interpolate using first_value %f node->num_missing %d factor
%f\n", (float)first_value, node->num_missing, (float)factor);

                if(begin_wrap)
                {
                  count=0;
                  for(i=NUM_ANGLES-first_wrap_num; i<node->num_missing;
i=(i+1)%NUM_ANGLES)
                  {
                          count++;
                          distances[i] = first_value + count*factor;
                          printf("Begin wrap; new distance %d is %f\n", i,
(float)distances[i]);
                  }
                }
                else if(end_wrap)
                {
                  count=0;
                  for(i=node->first_missing; i<=NUM_ANGLES + last_wrap_num; i++)
```

```
                {
                        count++;
                        distances[i%NUM_ANGLES] = first_value + count*factor;
                        printf("End wrap; new distance %d is %f\n", i,
(float)distances[i%NUM_ANGLES]);
                }
              }
              else
              {

                count=0;
                for(i=node->first_missing; i<node->first_missing+node->num_missing;
i++)
                {
                        count++;
                        distances[i] = first_value + count*factor;
                        printf("No wrap; new distance %d is %f\n", i,
(float)distances[i]);
                }
                printf("Done with this interpolation\n");
              }
              node = node->next;
   }


  printf("Distances after interpolation:\n");
  for(i=0; i<NUM_ANGLES; i++)
     printf("%d: %f\n", i, (float)distances[i]);

  node = head;
```

```
  while(node)
  {
    new_node = node;
    node = node->next;
    free(new_node);
  }
}

static void find_distances(gdouble *distances, gint *empty,
                           struct point *centroid,
                           gint left_edge, gint right_edge,
                           gint first_row, gint last_row,
                           gint *left, gint *right,
                           gint *top, gint *bottom)
{
  // Derive the lines representing radii from the centroid
  // pi = M_PI (from math.h)
  // m = tan(theta), where theta is the angle of interest
  //
  // Search top from pi/4 to 3pi/4
  // Search bottom from 5pi/4 to 7pi/4
  // for searches top or bottom, look for x where y satisfies
  // y = m(x - centroid->x) + centroid->y
  //
  // Search left from 3pi/4 to 5pi/4
  // Search right from 0 to pi/4 and 7pi/4 to 0
  // for searches left or right, look for y where x satisfies
  // x = centroid->y/m + centroid->x
  //
  // Searches on (2a-1)*pi/4 should turn up solutions regardless
  // of which contiguous edge is searched
```

```
struct point intersection;
gint i, quadrant, sign;
gdouble angles[NUM_ANGLES], distance;

sign = 1;

// Search types: 0 = left/right, 1 = top/bottom

for(i=0; i<NUM_ANGLES; i++)
{
        angles[i] = (2 * M_PI * i)/NUM_ANGLES;
}

for(i=0; i<NUM_ANGLES; i++)
{
  printf("\nProcessing angle %f pi\n", angles[i]/M_PI);
  if(i < NUM_ANGLES/4)
    quadrant=1;
      else if(i < NUM_ANGLES/2)
        quadrant=2;
      else if(i < 3*NUM_ANGLES/4)
        quadrant=3;
      else
        quadrant=4;
      switch(i){
        case 0: intersection.y = centroid->y;
                intersection.x = right[centroid->y];
                distance = intersection.x - centroid->x;
                break;
        case NUM_ANGLES/4: intersection.x = centroid->x;
```

```
                    intersection.y = bottom[centroid->x];
                    distance = intersection.y - centroid->y;
                    break;
            case NUM_ANGLES/2: intersection.y = centroid->y;
                    intersection.x = left[centroid->y];
                    distance = centroid->x - intersection.x;
                    break;
            case 3*NUM_ANGLES/4: intersection.x = centroid->x;
                    intersection.y = top[centroid->x];
                    distance = centroid->y - intersection.y;
                    break;
            default: if(quadrant==1)
                    {
                      find_max_intersection(angles[i], centroid, right, first_row,
last_row, &intersection, 0, quadrant, &distance);
                if((intersection.x == -1)||(intersection.y==-1))
                        find_max_intersection(angles[i], centroid, bottom,
left_edge, right_edge, &intersection, 1, quadrant, &distance);
                    }
                    else if(quadrant==2)
                    {
                      find_max_intersection(angles[i], centroid, bottom,
left_edge, right_edge, &intersection, 1, quadrant, &distance);
                if((intersection.x == -1)||(intersection.y==-1))
                        find_max_intersection(angles[i], centroid, left,
first_row, last_row, &intersection, 0, quadrant, &distance);
                    }
                    else if(quadrant==3)
                    {
                      find_max_intersection(angles[i], centroid, left, first_row,
last_row, &intersection, 0, quadrant, &distance);
```

```
                if((intersection.x == -1)||(intersection.y==-1))
                        find_max_intersection(angles[i], centroid, top, left_edge,
right_edge, &intersection, 1, quadrant, &distance);
                    }
                    else if(quadrant==4)
                    {
                      find_max_intersection(angles[i], centroid, top, left_edge,
right_edge, &intersection, 1, quadrant, &distance);
                if((intersection.x == -1)||(intersection.y==-1))
                        find_max_intersection(angles[i], centroid, right,
first_row, last_row, &intersection, 0, quadrant, &distance);
                    }
                    else
                      printf("ERROR: Not assigned to proper quadrant\n");

    }
        printf("Calculating distance from centroid (%ld, %ld) to col %ld row
%ld\n", centroid->x, centroid->y, intersection.x, intersection.y);

    // Mark as "no intersection" if white pixel wasn't there
    if(((intersection.x == -1)||(intersection.y==-1)) || (distance < 0))
    //if((intersection.x == -1)||(intersection.y==-1))
        {
          empty[i] = 1;
          distances[i]=1000;
        }
        else
          distances[i] = distance;
        printf("Distance: %f\n", (float)distances[i]);
  }
  //interpolate_missing_distances(distances, empty);
```

```
}

static void print_distances(gdouble *distance, gint *empty)
{
  gint i;

  for(i=0; i<NUM_ANGLES; i++)
  {
              if(!empty[i])
                printf("Distance[%d] = %f\n", i, (float)distance[i]);
              else
                printf("Distance[%d] was not calculated\n", i);
  }
}


static void
check_db_error(struct sqlca sqlerrstruct)
{
  if(!sqlerrstruct.sqlcode)
    printf("No error detected\n");
  else if(sqlerrstruct.sqlcode == 100)
    printf("End of cursor/no data found\n");
  else
  {
    printf("SQL error number %d:\n", (int)sqlerrstruct.sqlcode);
    printf("Line %d: Error: %s\n", (int)sqlerrstruct.sqlerrm.sqlerrml,
                        sqlerrstruct.sqlerrm.sqlerrmc);
  }
}
```

```
static void store_distances(gdouble *distance, gint *empty);

static void store_density(gdouble density);

static struct point* calculate_centroid(
                                        gint first_row, gint last_row,
                                        gint left_edge, gint right_edge,
                                        gint *top, gint *bottom,
                                        gint *left, gint *right)
{
  gint i,j, redundant_count;
  glong colsum, rowsum, num_edge_pts;
  struct point *the_centroid;

  colsum=rowsum=num_edge_pts=redundant_count=0;

  for(i=first_row; i<=last_row; i++)
  {
    if(right[i]!=-1)
    {
      colsum+=right[i];
      rowsum+=i;
      num_edge_pts++;
    }
    if(left[i]!=-1)
    {
      colsum+=left[i];
      rowsum+=i;
      num_edge_pts++;
    }
  }
```

```
for(i=left_edge; i<=right_edge; i++)
{
  if(top[i]!=-1)
  {
    colsum+=i;
    rowsum+=top[i];
    num_edge_pts++;
  }
  if(bottom[i]!=-1)
  {
    colsum+=i;
    rowsum+=bottom[i];
    num_edge_pts++;
  }
}
printf("centroid before removing redundancy: x=%ld y=%ld\n",
      colsum/num_edge_pts, rowsum/num_edge_pts);

// Remove redundant points
for(i=first_row; i<=last_row; i++)
{
  for(j=left_edge; j<=right_edge; j++)
  {
            if((top[j]==i)&&(left[i]==j))
            {
              redundant_count++;
              colsum-=j;
              rowsum-=i;
            }
  }
```

```
  for(j=right_edge; j>=left_edge; j--)
  {
            if((top[j]==i)&&(right[i]==j))
            {
              redundant_count++;
              colsum-=j;
              rowsum-=i;
            }
  }
}
for(i=last_row; i>=first_row; i--)
{
  for(j=left_edge; j<=right_edge; j++)
  {
            if((bottom[j]==i)&&(left[i]==j))
            {
              redundant_count++;
              colsum-=j;
              rowsum-=i;
            }
  }
  for(j=right_edge; j>=left_edge; j--)
  {
            if((bottom[j]==i)&&(right[i]==j))
            {
              redundant_count++;
              colsum-=j;
              rowsum-=i;
            }
  }
}
```

```
    printf("Removed %d redundant points\n", redundant_count);

    the_centroid = malloc(sizeof(struct point));
    the_centroid->x = colsum/(num_edge_pts-redundant_count);
    the_centroid->y = rowsum/(num_edge_pts-redundant_count);

    printf("centroid after removing redundancy: x=%ld y=%ld\n",
          the_centroid->x, the_centroid->y);

    return the_centroid;
}

static void interpolate_edge(gint begin, gint end, gint *edge, char *type)
{
    gint i, j, uninitialized_count, uninitialized_begin;
    gint begin_value, end_value, total_interpolated, factor;

    uninitialized_count=0;
    uninitialized_begin=0;
    total_interpolated=0;

    printf("%s contour:\n", type);

    for(i=begin; i<=end; i++)
    {
      if(edge[i] == -1)
      {
        uninitialized_count++;
        if(i>0)
        {
```

```
        if(edge[i-1]!=-1)
                {
                  uninitialized_begin = i;
                  begin_value=edge[i-1];
                        printf("Storing begin_value %d at %d\n", begin_value, i-1);
                }
               }
               if(i<end)
               {
                 if(edge[i+1] != -1)
                 {
                        end_value = edge[i+1];
                        printf("Storing end_value %d at %d\n", end_value, i+1);
                        factor= (end_value - begin_value)/(uninitialized_count+1);
                        if(uninitialized_count < 16)
                        {
                          printf("Using begin value %d end value %d, factor %d\n",
begin_value, end_value, factor);
                          for(j=0; j<uninitialized_count; j++)
                          {
                           edge[uninitialized_begin+j] = begin_value + (j+1)*factor;
                          }
                        }
                        total_interpolated += uninitialized_count;
                        uninitialized_count=0;
                 }
               }
    }
  }

  printf("Total interpolated in %s contour: %d\n", type, total_interpolated);
```

```
}

static gint show_edge(gint begin, gint end, gint *edge, char *type)
{
   gint i, uninitialized_count;

   uninitialized_count=0;
   printf("%s contour:\n", type);

   for(i=begin; i<=end; i++)
   {
     if(edge[i] == -1)
     {
       uninitialized_count++;
     }

/*
     if((!strcmp(type, "right")) || !(strcmp(type, "left")))
       printf("row %d col %d\n", i, edge[i]);
     else
       printf("col %d row %d\n", i, edge[i]);
*/
   }

   printf("Number uninitialized in %s contour: %d\n", type, uninitialized_count);
   return uninitialized_count;
}

static void doit       (void);

GimpPlugInInfo PLUG_IN_INFO =
```

```
{
 NULL,   /* init_proc  */
 NULL,   /* quit_proc  */
 query,  /* query_proc */
 run,    /* run_proc   */
};

gint bytes;
gint run_flag = 0;
gint32 image;


MAIN ()

static void
query (void)
{
 static GimpParamDef args[] =
 {
  { GIMP_PDB_INT32,    "run_mode", "Interactive, non-interactive" },
  { GIMP_PDB_IMAGE,    "image",    "Input image " },
  { GIMP_PDB_DRAWABLE, "drawable", "Input drawable" },
 };
 static gint nargs = sizeof (args) / sizeof (args[0]);

 gimp_install_procedure ("plug_in_centroid",
                                        "Generate centroid-radius image statistics",
                                        "",
                                        "Cindy Huyser",
                                        "Cindy Huyser",
                                        "2002",
```

```
                                    N_("<Image>/Filters/Centroid"),
                                    "GRAY*",
                                    GIMP_PLUGIN,
                                    nargs, 0,
                                    args, NULL);
}

static void
run (gchar      *name,
     gint        nparams,
     GimpParam  *param,
     gint       *nreturn_vals,
     GimpParam **return_vals)
{
  static GimpParam  values[1];
  GimpRunModeType   run_mode;
  GimpPDBStatusType status = GIMP_PDB_SUCCESS;

  INIT_I18N_UI();

  (void) name; /* Shut up warnings about unused parameters. */

  *nreturn_vals = 1;
  *return_vals = values;

  run_mode = param[0].data.d_int32;
  image = param[1].data.d_image;

  /*  Get the specified drawable  */
  drawable = gimp_drawable_get (param[2].data.d_drawable);
```

```
/*  The plug-in is not able to handle images smaller than 3 pixels  */
if (drawable->width < 3 || drawable->height < 3)
  {
   g_message (_("Centroid detection does not work\non layers smaller than 3 pixels."));
   status = GIMP_PDB_EXECUTION_ERROR;
   values[0].type = GIMP_PDB_STATUS;
   values[0].data.d_status = status;
   return;
  }

if (run_mode == GIMP_RUN_NONINTERACTIVE)
  {
   if (nparams != 3)
              status = GIMP_PDB_CALLING_ERROR;
  }

if (status == GIMP_PDB_SUCCESS)
  {
   /*  Make sure that the drawable is gray */
   if (gimp_drawable_is_gray(drawable->id))
              {
               gimp_tile_cache_ntiles (2 * (drawable->width /
                                                    gimp_tile_width () + 1));

               doit ();

              }
   else
              {
               status = GIMP_PDB_EXECUTION_ERROR;
              }
```

```
    gimp_drawable_detach (drawable);
  }

 values[0].type = GIMP_PDB_STATUS;
 values[0].data.d_status = status;
}


static void
doit (void)
{
 GimpPixelRgn  srcPR;
 gint         width, height, row, col;
 gint         i, left_edge, right_edge, first_row, last_row;
 guchar       *srcrow, *srccol;
 gint      *top;
 gint      *bottom;
 gint      *left;
 gint      *right;
 gint         xoff;
 gint                        last_left, last_top, empty[NUM_ANGLES];
 gdouble      distances[NUM_ANGLES];
 gboolean     leftset, topset, first_top_found;
 gboolean     prev_had_white;
 struct point  *calculated_centroid;
 gint         num_uninterpolated;
 gint32       num_white_pixels, total_pixels;
 gdouble      density;


 /* Get the size of the input image.
```

```
 */
width  = drawable->width;
height = drawable->height;
bytes  = drawable->bpp;

srcrow = g_new (guchar, width);
srccol = g_new (guchar, height);
top = g_new (gint, width);
bottom = g_new (gint, width);
left = g_new (gint, height);
right = g_new (gint, height);

// Initialize arrays to values telling where no value set
for(i=0; i<width; i++)
{
 top[i]=-1;
 bottom[i]=-1;
}
for(i=0; i<height; i++)
{
 left[i]=-1;
 right[i]=-1;
}

/*  initialize the pixel region  */
gimp_pixel_rgn_init (&srcPR, drawable, 0, 0, width, height, FALSE, FALSE);

/* initialize source arrays */
gimp_pixel_rgn_get_row (&srcPR, srcrow, 0, 0, width);

leftset = topset = first_top_found = prev_had_white = FALSE;
```

```
num_white_pixels = total_pixels = 0;

left_edge = width;
first_row = height;
last_row = right_edge = 0;
last_left = last_top = 0;

total_pixels = width * height;

/* Find the bounding box of the image */
/* Get the left and right edge info, too */
for (row = 0; row < height; row++)
{
   xoff = 0;

   for (col = 0; col < width; col++)
   {
                if(srcrow[xoff] == 255)
                {
                        num_white_pixels++;
                        if(!first_top_found)
                        {
                          first_top_found = TRUE;
                          first_row = row;
                        }
                        if((col == 0) || ((!leftset) && (last_left==0)))
                        {
                          leftset=TRUE;
                          left[row] = col;
                          if(col < left_edge)
                            left_edge = col;
```

```
                    }
                    if(col == width-1)
                    {
                      right[row] = col;
                      right_edge = col;
                    }
            }
            else
            {
                    if(last_left==255)
                    {
                      right[row] = col-1;
                      if(col-1 > right_edge)
                        right_edge = col-1;
                    }
            }
            last_left = srcrow[xoff];
            xoff+=bytes;
}
// Were any white pixels in row? if not, previous might be last row
if(!leftset)
{
        if(prev_had_white)
          last_row = row-1;
        prev_had_white = FALSE;
}
else
{
        prev_had_white = TRUE;
}
```

```
  leftset = FALSE;
  last_left = 0;
  if(row < height-1)
    gimp_pixel_rgn_get_row (&srcPR, srcrow, 0, row + 1, width);

 }

density = (gdouble)num_white_pixels/(gdouble)total_pixels;

// Now, get the column info
prev_had_white = FALSE;
gimp_pixel_rgn_get_col(&srcPR, srccol, 0, 0, height);

for (col = 0; col < width; col++)
{
  xoff = 0;

  for (row = 0; row < height; row++)
  {

            if(srccol[xoff] == 255)
            {
                    if((row == 0) || ((!topset) && (last_top==0)))
                    {
                      topset=TRUE;
                      top[col] = row;
                    }
                    if(row==height - 1)
                    {
                      bottom[col] = row;
                    }
```

```
            }
            else
            {
                        if(last_top==255)
                        {
                          bottom[col] = row-1;
                        }
            }
            last_top = srccol[xoff];
            xoff+=bytes;
}


// Were any white pixels in col? if not, previous might be last col
if(!topset)
{
            if(prev_had_white)
              last_top = col-1;
            prev_had_white = FALSE;
}
else
{
            prev_had_white = TRUE;
}

topset = FALSE;
last_top = 0;

if(col < (width-1))
   gimp_pixel_rgn_get_col(&srcPR, srccol, col+1, 0, height);
```

```
}

printf("Image boundaries: left %d right %d top %d bottom %d\n",
                        left_edge, right_edge, first_row, last_row);


num_uninterpolated = show_edge(first_row, last_row, left, "left");
if(num_uninterpolated>0)
{
   interpolate_edge(first_row, last_row, left, "left");
   num_uninterpolated = 0;
}
show_edge(first_row, last_row, left, "left");

num_uninterpolated = show_edge(first_row, last_row, right, "right");
if(num_uninterpolated>0)
{
   interpolate_edge(first_row, last_row, right, "right");
            num_uninterpolated = 0;
}
show_edge(first_row, last_row, left, "left");

num_uninterpolated = show_edge(left_edge, right_edge, top, "top");
if(num_uninterpolated>0)
{
   interpolate_edge(left_edge, right_edge, top, "top");
            num_uninterpolated = 0;
}
show_edge(first_row, last_row, left, "left");

num_uninterpolated = show_edge(left_edge, right_edge, bottom, "bottom");
```

```
if(num_uninterpolated>0)
{
   interpolate_edge(left_edge, right_edge, bottom, "bottom");
            num_uninterpolated = 0;
}
show_edge(first_row, last_row, left, "left");

// Approximate centroid
calculated_centroid = calculate_centroid(first_row, last_row,
                                                                left_edge, right_edge,
                                                                top, bottom,
                                                                left, right);

printf("Approximate centroid: col %ld row %ld\n", calculated_centroid->x,
               calculated_centroid->y);

// Set all to "found"
for(i=0; i<NUM_ANGLES; i++)
            empty[i]=0;

find_distances(distances, empty, calculated_centroid,
                                left_edge, right_edge, first_row, last_row,
                                left, right, top, bottom);

//print_distances(distances, empty);

store_distances(distances, empty);
store_density(density);

g_free(top);
g_free(bottom);
```

```
    g_free(left);
    g_free(right);
    g_free(srcrow);
    g_free(srccol);

    printf("Done processing\n");

}

static void store_density(gdouble density)
{
  exec sql begin declare section;
  char img_name[100];
  int result;
  float img_density;
  exec sql end declare section;

  img_density = (float)density;
  strcpy(img_name, gimp_image_get_filename(image));
  printf("Storing density %f for image name %s\n", img_density, img_name);

  exec sql include sqlca;
  exec sql connect to glyph;

  check_db_error(sqlca);

  exec sql select count(*) into :result from images where image_name=:img_name;

  check_db_error(sqlca);
  if(!result)
  {
```

```
    printf("Error: no record exists\n");
    return;
  }
  printf("Found a record\n");

  exec sql select image_num into :result from images where image_name=:img_name;
  check_db_error(sqlca);
  printf("Image has image_num %d\n", result);

  printf("Storing density\n");
  exec sql update images set density=:img_density where image_num=:result;
  check_db_error(sqlca);

  exec sql commit;
  check_db_error(sqlca);

  exec sql disconnect all;
  check_db_error(sqlca);
}

static void store_distances(gdouble *distance, gint *empty)
{
  FILE *dist_file;
  gchar *dist_path = "/tmp/distances/";
  gchar dist_name[100] = {0};
  gint i;

  exec sql begin declare section;
  char img_name[100];
  int result;
  exec sql end declare section;
```

```
strcpy(img_name, gimp_image_get_filename(image));
printf("The image name is %s\n", img_name);

exec sql include sqlca;
exec sql connect to glyph;

check_db_error(sqlca);

exec sql select count(*) into :result from images where image_name = :img_name;

check_db_error(sqlca);
if(result)
  printf("Found a record\n");
else
{
   printf("No record exists\n");
   exec sql insert into images values(nextval('"images_image_num_seq"'), :img_name);
   check_db_error(sqlca);
   exec sql commit;
   check_db_error(sqlca);
}

exec sql select image_num into :result from images where image_name = :img_name;
check_db_error(sqlca);
printf("Image has image_num %d\n", result);
sprintf(dist_name, "%sdist_%d", dist_path, result);
printf("Distances file name: %s\n", dist_name);

dist_file = fopen(dist_name, "w");
if(! dist_file)
```

```
    printf("Distances file failed to open\n");

  for(i=0; i<NUM_ANGLES; i++)
  {
    if(dist_file)
    {
                fprintf(dist_file, "%d %f\n", empty[i], distance[i]);
    }
  }

  if(dist_file)
  {
    fprintf(dist_file, "%s", img_name);
    fclose(dist_file);
  }

  exec sql disconnect all;
  check_db_error(sqlca);
}
```

```
/* Minkowski Erode plug-in for the GIMP -- Version 0.1
 * Copyright (C) 2001 Cindy Huyser <chuyser@io.com>
 *
 * Based heavily on:
 * Convolution Matrix plug-in for the GIMP -- Version 0.13
 * Copyright (C) 1997 Lauri Alanko <la@iki.fi>
 *
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 *
 * The GNU General Public License is also available from
 * http://www.fsf.org/copyleft/gpl.html
 *
 * CHANGELOG:
 *
 * v0.1         01.18.2002
 *              Initial release.
 *
 *
```

```
 */

#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <sys/types.h>
#ifdef HAVE_UNISTD_H
#include <unistd.h>
#endif

#include <gtk/gtk.h>

#include <libgimp/gimp.h>
#include <libgimp/gimpui.h>

#include <libgimp/stdplugins-intl.h>


typedef enum
{
  EXTEND,
  WRAP,
  CLEAR,
  MIRROR
} BorderMode;

GimpDrawable *drawable;
gint32        image;

gchar * const bmode_labels[] =
```

```
{
 N_("Extend"),
 N_("Wrap"),
 N_("Crop")
};

/* Declare local functions. */
static void query (void);
static void run   (gchar     *name,
                   gint       nparams,
                   GimpParam  *param,
                   gint      *nreturn_vals,
                   GimpParam **return_vals);

static gint dialog (void);

static void doit       (void);
static void check_config (void);

GimpPlugInInfo PLUG_IN_INFO =
{
 NULL,   /* init_proc  */
 NULL,   /* quit_proc  */
 query,  /* query_proc */
 run,    /* run_proc   */
};

gint bytes;
gint sx1, sy1, sx2, sy2;
gint run_flag = 0;
```

```
typedef struct
{
 gfloat    matrix[3][3];
 BorderMode bmode;
} config;

const config default_config =
{
 {
  { 0.0, 0.0, 0.0 },
  { 0.0, 1.0, 0.0 },
  { 0.0, 0.0, 0.0 }
 },              /* matrix */
 CLEAR,          /* border-mode */
};

config my_config;

struct
{
 GtkWidget *matrix[3][3];
 GtkWidget *bmode[3];
 GtkWidget *ok;
} my_widgets;


MAIN ()

static void
query (void)
{
```

```
 static GimpParamDef args[] =
 {
  { GIMP_PDB_INT32,    "run_mode", "Interactive, non-interactive" },
  { GIMP_PDB_IMAGE,    "image",    "Input image " },
  { GIMP_PDB_DRAWABLE, "drawable", "Input drawable" },
  { GIMP_PDB_INT32, "argc_matrix", "The number of elements in the following array.
Should be always 9." },
  { GIMP_PDB_FLOATARRAY, "matrix", "The 3x3 erosion matrix" },
  { GIMP_PDB_INT32, "bmode", "Mode for treating image borders" }
 };
 static gint nargs = sizeof (args) / sizeof (args[0]);

 gimp_install_procedure ("plug_in_erode",
                         "Minkowski erode using a matrix",
                         "",
                         "Cindy Huyser",
                         "Cindy Huyser",
                         "2001",
                         N_("<Image>/Filters/Minkowski/Erode"),
                         "GRAY*",
                         GIMP_PLUGIN,
                         nargs, 0,
                         args, NULL);
}

static void
run (gchar      *name,
   gint       nparams,
   GimpParam  *param,
   gint       *nreturn_vals,
   GimpParam **return_vals)
```

```
{

 static GimpParam  values[1];
 GimpRunModeType   run_mode;
 GimpPDBStatusType status = GIMP_PDB_SUCCESS;
 gint              x, y;
 values[0].type = GIMP_PDB_STATUS;

 INIT_I18N_UI();

 (void) name; /* Shut up warnings about unused parameters. */

 *nreturn_vals = 1;
 *return_vals = values;

 run_mode = param[0].data.d_int32;
 image = param[1].data.d_int32;


 /*  Get the specified drawable  */
 drawable = gimp_drawable_get (param[2].data.d_drawable);

 /*  The plug-in is not able to handle images smaller than 3 pixels  */


 if (drawable->width < 3 || drawable->height < 3)
  {
    g_message (_("Eroding does not work\non layers smaller than 3 pixels."));
    status = GIMP_PDB_EXECUTION_ERROR;
    values[0].data.d_status = status;
    return;
```

```
 }


my_config = default_config;
if (run_mode == GIMP_RUN_NONINTERACTIVE)
 {
  if (nparams != 6)
            status = GIMP_PDB_CALLING_ERROR;
  else
            {
             if (param[3].data.d_int32 != 9) {
              status = GIMP_PDB_CALLING_ERROR;
             } else {
              for (y = 0; y < 3; y++)
                for (x = 0; x < 3; x++)
                        my_config.matrix[x][y]=param[4].data.d_floatarray[y*3+x];
             }

             my_config.bmode    = param[5].data.d_int32;

             check_config ();
            }
 }
else
 {
  gimp_get_data ("plug_in_erode", &my_config);

  if (run_mode == GIMP_RUN_INTERACTIVE)
            {
             /* Oh boy. We get to do a dialog box, because we can't really
              * expect the user to set us up with the right values using gdb.
```

```
            */

            check_config ();
            if (!dialog())
              {
               /* The dialog was closed, or something similarly evil happened. */
               status = GIMP_PDB_EXECUTION_ERROR;
              }
           }
 }

if (status == GIMP_PDB_SUCCESS)
 {

  /*  Make sure that the drawable is gray  */
  if (gimp_drawable_is_gray(drawable->id))
           {
            gimp_progress_init (_("Applying Minkowski erode"));
            gimp_tile_cache_ntiles (2 * (drawable->width /
                                                  gimp_tile_width () + 1));

            doit ();

            if (run_mode != GIMP_RUN_NONINTERACTIVE)
              gimp_displays_flush ();

            if (run_mode == GIMP_RUN_INTERACTIVE)
              gimp_set_data ("plug_in_erode", &my_config, sizeof (my_config));
           }
  else
           {
```

```
                status = GIMP_PDB_EXECUTION_ERROR;
              }
    gimp_drawable_detach (drawable);
   }
 values[0].data.d_status = status;
}


/*  A generic wrapper to gimp_pixel_rgn_get_row which handles unlimited
 *  wrapping or gives you transparent regions outside the image
 */

static void
my_get_row (GimpPixelRgn *PR,
            guchar      *dest,
            gint         x,
            gint         y,
            gint         w)
{
 gint width, height, bytes;
 gint i;

 width  = PR->drawable->width;
 height = PR->drawable->height;
 bytes  = PR->drawable->bpp;

 /* Y-wrappings */

 switch (my_config.bmode)
  {
  case WRAP:
```

```
  /* Wrapped, so we get the proper row from the other side */
  while (y < 0) /* This is the _sure_ way to wrap. :) */
            y += height;
  while (y >= height)
            y -= height;
  break;

 case CLEAR:
  /* Beyond borders, so set full transparent. */
  if (y < 0 || y >= height)
            {
             memset (dest, 0, w * bytes);
             return; /* Done, so back. */
            }
 case MIRROR:
  /* The border lines are _not_ duplicated in the mirror image */
  /* is this right? */
  while (y < 0 || y >= height)
            {
             if (y < 0)
               y = -y; /* y=-y-1 */
             if (y >= height)
//             y = 2 * height - y - 2; /* y=2*height-y-1 */
               y = height - (y % height);
            }
  break;

 case EXTEND:
  y = CLAMP (y , 0 , height - 1);
  break;
 }
```

```
switch (my_config.bmode)
 {
 case CLEAR:
  if (x < 0)
              {
               i = MIN (w, -x);
               memset (dest, 0, i * bytes);
               dest += i * bytes;
               w -= i;
               x += i;
              }
  if (w)
              {
               i = MIN (w, width);
               gimp_pixel_rgn_get_row (PR, dest, x, y, i);
               dest += i * bytes;
               w -= i;
               x += i;
              }
  if (w)
              memset (dest, 0, w * bytes);
  break;

 case WRAP:
  while (x < 0)
              x += width;
  i = MIN (w, width - x);
  gimp_pixel_rgn_get_row (PR, dest, x, y, i);
  w -= i;
  dest += i * bytes;
```

```
 x = 0;
 while (w)
            {
             i = MIN (w, width);
             gimp_pixel_rgn_get_row (PR, dest, x, y, i);
             w -= i;
             x += i;
             dest += i * bytes;
            }
 break;

case EXTEND:
 if (x < 0)
            {
             gimp_pixel_rgn_get_pixel (PR, dest, 0, y);
             x++;
             w--;
             dest += bytes;

             while (x < 0 && w)
              {
               for (i = 0; i < bytes; i++)
                           {
                            *dest = *(dest - bytes);
                            dest++;
                           }
               x++;
               w--;
              }
            }
 if (w)
```

```
            {
             i = MIN (w, width);
             gimp_pixel_rgn_get_row (PR, dest, x, y, i);
             w -= i;
             x += i;
             dest += i * bytes;
            }
    while (w)
            {
             for (i = 0; i < bytes; i++)
               {
                *dest= *(dest - bytes);
                dest++;
               }
             x++;
             w--;
            }
    break;

  case MIRROR: /* Not yet handled */
    break;
  }
}

static gfloat
calc_erosion (guchar **srcrow,
              gint    xoff,
              gint    i)
{
// Using max to erode darker features
```

```
 static gint bytes      = 0;

 gfloat max     = 0;
 gfloat temp;
 gint   x, y;

 if (!bytes)
  {
   bytes = drawable->bpp;
  }
 for (y = 0; y < 3; y++)
  for (x = 0; x < 3; x++)
   {
              temp = my_config.matrix[x][y];
              if (temp > 0)
              {
                temp *= srcrow[y][xoff + x * bytes];
                max = MAX(max, temp);
              }
   }

 return max;
}

static void
doit (void)
{
 GimpPixelRgn  srcPR, dest1PR;
 gint        width, height, row, col;
 gint        w, h, i;
 guchar      *destrow[2];
```

```
guchar      *srcrow[3];
guchar      *temprow;
gfloat      erosion;
gfloat       update_amt;
gint        xoff;

/* Get the input area. This is the bounding box of the selection in
 *  the image (or the entire image if there is no selection). Only
 *  operating on the input area is simply an optimization. It doesn't
 *  need to be done for correct operation. (It simply makes it go
 *  faster, since fewer pixels need to be operated on).
 */

gimp_drawable_mask_bounds (drawable->id, &sx1, &sy1, &sx2, &sy2);
w = sx2 - sx1;
h = sy2 - sy1;

/* Get the size of the input image. (This will/must be the same
 *  as the size of the output image.
 */
width  = drawable->width;
height = drawable->height;
bytes  = drawable->bpp;

for (i = 0; i < 3; i++)
  srcrow[i] = g_new (guchar, (w + 2) * bytes);
for (i = 0; i < 2; i++)
  destrow[i]= g_new (guchar, w * bytes);

// Do the erosion work
/*  initialize the pixel regions  */
```

```
gimp_pixel_rgn_init (&srcPR, drawable,
                                sx1 - 1, sy1 - 1, w + 2, h + 2, FALSE, FALSE);
gimp_pixel_rgn_init (&dest1PR, drawable, sx1, sy1, w, h, TRUE, TRUE);

/* initialize source arrays */
for (i = 0; i < 3; i++)
 my_get_row (&srcPR, srcrow[i], sx1 - 1, sy1 + i - 1, w + 2);

for (row = sy1; row < sy2; row++)
 {
  xoff = 0;

  for (col = sx1; col < sx2; col++)
          for (i = 0; i < bytes; i++)
           {
            erosion = calc_erosion(srcrow, xoff, i);

            destrow[1][xoff]= (guchar) CLAMP (erosion, 0, 255);
            xoff++;
           }

  if (row > sy1 + 1)
          gimp_pixel_rgn_set_row (&dest1PR, destrow[0], sx1, row - 1, w);

  temprow = destrow[0];
  destrow[0] = destrow[1];
  destrow[1] = temprow;
  temprow = srcrow[0];

  for (i = 0; i < 2; i++)
          srcrow[i] = srcrow[i + 1];
```

```
    srcrow[2] = temprow;
    my_get_row (&srcPR, srcrow[2], sx1 - 1, row + 2, w + 2);
    update_amt = (double)(row - sy1)/(h);
    gimp_progress_update (update_amt);
}

 /* put the final rows in the buffer in place */
 if (h > 1)
   gimp_pixel_rgn_set_row (&dest1PR, destrow[0], sx1, row - 2, w);
 if (h > 2)
   gimp_pixel_rgn_set_row (&dest1PR, destrow[1], sx1, row - 1, w);

 /*  update the timred region  */
 gimp_drawable_flush (drawable);
 gimp_drawable_merge_shadow (drawable->id, TRUE);
 gimp_drawable_update (drawable->id, sx1, sy1, sx2 - sx1, sy2 - sy1);
}

/*****************************************************
 * GUI stuff
 */

static void
fprint (gfloat  f,
                gchar  *buffer)
{
 gint i, t;

 sprintf (buffer, "%.7f", f);
```

```
  for (t = 0; buffer[t] != '.'; t++);

  i = t + 1;

  while (buffer[i] != '\0')
    {
     if (buffer[i] != '0')
                t = i + 1;

     i++;
    }

  buffer[t] = '\0';
}

static void
redraw_matrix (void)
{
  gint x,y;
  gchar buffer[12];

  for (y = 0; y < 3; y++)
    for (x = 0; x < 3; x++)
     {
                fprint (my_config.matrix[x][y], buffer);

                gtk_entry_set_text (GTK_ENTRY (my_widgets.matrix[x][y]), buffer);
     }
}

static void
```

```
redraw_bmode (void)
{
 gtk_toggle_button_set_active
   (GTK_TOGGLE_BUTTON (my_widgets.bmode[my_config.bmode]), TRUE);
}

static void
redraw_all (void)
{
 redraw_matrix ();
 redraw_bmode ();
}

static void
check_matrix (void)
{
 gint     x, y;
 gboolean  valid = FALSE;
 gfloat    sum   = 0.0;

 for (y = 0; y < 3; y++)
   for (x = 0; x < 3; x++)
     {
               sum += my_config.matrix[x][y];
               if (my_config.matrix[x][y] != 0.0)
                 valid = TRUE;
     }
}

static void
ok_callback (GtkWidget *widget,
```

```
                    gpointer   data)
{
 run_flag = TRUE;

 gtk_widget_destroy (GTK_WIDGET (data));
}

/* Checks that the configuration is valid for the image type */
static void
check_config (void)
{
 if (!gimp_drawable_has_alpha (drawable->id))
   {
    my_config.bmode      = EXTEND;
   }
}

static void
defaults_callback (GtkWidget *widget,
                                gpointer   data)
{
 my_config = default_config;
 check_config ();
 redraw_all ();
}

static void
entry_callback (GtkWidget *widget,
                               gpointer   data)
{
 gfloat *value = (gfloat *) data;
```

```
  *value = atof (gtk_entry_get_text (GTK_ENTRY (widget)));

  check_matrix ();
}

static void
my_bmode_callback (GtkWidget *widget,
                   gpointer   data)
{
  my_config.bmode = (int) data - 1;
}

static gint
dialog (void)
{
  GtkWidget *dlg;
  GtkWidget *main_hbox;
  GtkWidget *table;
  GtkWidget *entry;
  GtkWidget *button;
  GtkWidget *box;
  GtkWidget *inbox;
  GtkWidget *yetanotherbox;
  GtkWidget *frame;
  gchar      buffer[32];
  gint       x, y, i;
  GSList    *group;

  gimp_ui_init ("erodematrix", FALSE);
```

```
 dlg = gimp_dialog_new (_("Minkowski Erode"), "erodematrix",
                                        gimp_standard_help_func,
"filters/erodematrix.html",
                                        GTK_WIN_POS_MOUSE,
                                        FALSE, TRUE, FALSE,

                                        _("OK"), ok_callback,
                                        NULL, NULL, &my_widgets.ok, TRUE,
FALSE,
                                        _("Reset"), defaults_callback,
                                        NULL, 1, NULL, FALSE, FALSE,
                                        _("Cancel"), gtk_widget_destroy,
                                        NULL, 1, NULL, FALSE, TRUE,

                                        NULL);

 gtk_signal_connect (GTK_OBJECT (dlg), "destroy",
                                GTK_SIGNAL_FUNC (gtk_main_quit),
                                NULL);

 main_hbox = gtk_hbox_new (FALSE, 4);
 gtk_container_set_border_width (GTK_CONTAINER (main_hbox), 6);
 gtk_box_pack_start (GTK_BOX (GTK_DIALOG (dlg)->vbox), main_hbox,
                                TRUE, TRUE, 0);

 yetanotherbox = gtk_vbox_new (FALSE, 4);
 gtk_box_pack_start (GTK_BOX (main_hbox), yetanotherbox, FALSE, FALSE, 0);

 frame = gtk_frame_new (_("Matrix"));
 gtk_frame_set_shadow_type (GTK_FRAME (frame), GTK_SHADOW_ETCHED_IN);
 gtk_box_pack_start (GTK_BOX (yetanotherbox), frame, FALSE, FALSE, 0);
```

```
inbox = gtk_vbox_new (FALSE, 6);
gtk_container_set_border_width (GTK_CONTAINER (inbox), 4);
gtk_container_add (GTK_CONTAINER (frame), inbox);

table = gtk_table_new (5, 5, FALSE);
gtk_table_set_row_spacings (GTK_TABLE (table), 4);
gtk_table_set_col_spacings (GTK_TABLE (table), 4);
gtk_box_pack_start (GTK_BOX (inbox), table, FALSE, FALSE, 0);

for (y = 0; y < 3; y++)
  for (x = 0; x < 3; x++)
    {
            my_widgets.matrix[x][y] = entry = gtk_entry_new ();
            gtk_widget_set_usize (entry, 40, 0);
            gtk_table_attach (GTK_TABLE (table), entry, x, x+1, y, y+1,
                                    GTK_EXPAND | GTK_FILL, GTK_EXPAND |
GTK_FILL, 0, 0);
            gtk_entry_set_text (GTK_ENTRY (entry), buffer);
            gtk_signal_connect (GTK_OBJECT (entry), "changed",
                                    GTK_SIGNAL_FUNC (entry_callback),
                                    &my_config.matrix[x][y]);
            gtk_widget_show (entry);
    }

gtk_widget_show (table);
gtk_widget_show (inbox);
gtk_widget_show (frame);

gtk_widget_show (yetanotherbox);
```

```
inbox = gtk_vbox_new (FALSE, 4);
gtk_box_pack_start (GTK_BOX (main_hbox), inbox, FALSE, FALSE, 0);

frame = gtk_frame_new (_("Border"));
gtk_frame_set_shadow_type (GTK_FRAME (frame), GTK_SHADOW_ETCHED_IN);
gtk_box_pack_start (GTK_BOX (inbox), frame, FALSE, FALSE, 0);

box = gtk_vbox_new (FALSE, 1);
gtk_container_set_border_width (GTK_CONTAINER (box), 2);
gtk_container_add (GTK_CONTAINER (frame), box);

group = NULL;

for (i = 0; i < 3; i++)
 {
  my_widgets.bmode[i] = button =
            gtk_radio_button_new_with_label (group, gettext (bmode_labels[i]));
  group = gtk_radio_button_group (GTK_RADIO_BUTTON (button));
  gtk_box_pack_start (GTK_BOX (box), button, FALSE, FALSE, 0);
  gtk_widget_show (button);
  gtk_signal_connect (GTK_OBJECT (button), "toggled",
                                        (GtkSignalFunc) my_bmode_callback,
                                        (gpointer) (i + 1));
 }

gtk_widget_show (frame);
gtk_widget_show (box);

gtk_widget_show (inbox);

gtk_widget_show (main_hbox);
```

```
  gtk_widget_show (dlg);
  redraw_all ();

  gtk_main ();
  gdk_flush ();

  return run_flag;
}
```

```
/* Find Threshhold plug-in for the GIMP -- Version 0.1
 * Copyright (C) 2001 Cindy Huyser <chuyser@io.com>
 *
 * Purpose: test threshhold finding before and after scaling of image
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 *
 * The GNU General Public License is also available from
 * http://www.fsf.org/copyleft/gpl.html
 *
 */

#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <sys/types.h>
#ifdef HAVE_UNISTD_H
#include <unistd.h>
```

```
#endif

#include <gtk/gtk.h>

#include <libgimp/gimp.h>
#include <libgimp/gimpui.h>

#include <libgimp/stdplugins-intl.h>


//#define DEBUG
//#define EDGE_DEBUG
//#define MASK_TRIM_DEBUG


GimpDrawable *drawable;
gint32 image;


/* Declare local functions. */
static void query (void);
static void run   (gchar     *name,
                   gint       nparams,
                   GimpParam  *param,
                   gint      *nreturn_vals,
                   GimpParam **return_vals);

GimpPlugInInfo PLUG_IN_INFO =
{
 NULL,   /* init_proc  */
 NULL,   /* quit_proc  */
```

```
 query,  /* query_proc */
 run,    /* run_proc   */
};

gint run_flag = 0;


MAIN ()

static void
query (void)
{
 static GimpParamDef args[] =
 {
  { GIMP_PDB_INT32,    "run_mode", "Interactive, non-interactive" },
  { GIMP_PDB_IMAGE,    "image",    "Input image" },
  { GIMP_PDB_DRAWABLE, "drawable", "Input drawable" },
 };
 static gint nargs = sizeof (args) / sizeof (args[0]);

 gimp_install_procedure ("plug_in_find_thresh",
                                          "Find Threshhold",
                                          "",
                                          "Cindy Huyser",
                                          "Cindy Huyser",
                                          "2002",
                                          N_("<Image>/Filters/Find Threshhold"),
                                          "GRAY*",
                                          GIMP_PLUGIN,
                                          nargs, 0,
                                          args, NULL);
```

```
}

static void
doit(gint maxFeatures);

static void
run (gchar      *name,
     gint       nparams,
     GimpParam  *param,
     gint       *nreturn_vals,
     GimpParam **return_vals)
{
  static GimpParam  values[1];
  GimpRunModeType   run_mode;
  GimpPDBStatusType status = GIMP_PDB_SUCCESS;
  gint32 activeLayer;
  gint32                    maxFeatures = 10;
  gboolean    gotData;

  gotData = FALSE;


  INIT_I18N_UI();

  (void) name; /* Shut up warnings about unused parameters. */

  *nreturn_vals = 1;
  *return_vals = values;

  run_mode = param[0].data.d_int32;
```

```
 /*  Get the specified drawable  */
 image = param[1].data.d_image;
 activeLayer = gimp_image_get_active_layer(image);
 drawable = gimp_drawable_get (activeLayer);

 /*  The plug-in is not able to handle images smaller than 3 pixels  */
 if (drawable->width < 3 || drawable->height < 3)
  {
   g_message (_("Mask build does not work\non layers smaller than 3 pixels."));
   status = GIMP_PDB_EXECUTION_ERROR;
   values[0].type = GIMP_PDB_STATUS;
   values[0].data.d_status = status;
   return;
  }

 if (nparams != 3)
  status = GIMP_PDB_CALLING_ERROR;


 if (status == GIMP_PDB_SUCCESS)
  {
   /*  Make sure that the drawable has an alpha channel */
   if (gimp_drawable_has_alpha(drawable->id))
            {
             gimp_progress_init (_("Building mask"));

      gimp_tile_cache_ntiles (256 * (MAX(drawable->width,drawable->height)) /
gimp_tile_width() + 1);

             doit(maxFeatures);
```

```
                gimp_displays_flush ();

              }
    else
              {
               printf("Didn't find an alpha channel\n");
               status = GIMP_PDB_EXECUTION_ERROR;
              }
    gimp_drawable_detach (drawable);
   }

 values[0].type = GIMP_PDB_STATUS;
 values[0].data.d_status = status;
}

static gboolean
show_status(gchar *procedure_name, GimpParam *return_val);

static gboolean
preprocess(gint32 imageID, gint32 drawableID, gdouble blur);

static gint
findFeatures(gint *first_mask_line, gint *last_mask_line,
                                gint maxFeatures, GArray *lbounds[],
                                GArray *rbounds[],
                                GimpPixelRgn *maskPR,
                                gint *featureList, gint width, gint height,
                                gint bytes);

static void
showStoredEdges(gint maxFeatures, GArray *lbounds[], GArray  *rbounds[],
```

```
                                gint first_mask_line, gint last_mask_line);

static gfloat
get_blur_radius(gdouble percentage);

static gint
getThreshholdLevel(gint * u_thresh, gint maxFeatures,
                                GArray *lbounds[], GArray *rbounds[], gdouble
*percentage,
                                GimpPixelRgn *bgPR, GimpPixelRgn *maskPR, gint
*featurelist,
                                gint bg_bytes, gint bytes, gint width, gint height);


typedef struct{
   gint edge;
  } boundary;

static void
doit(gint maxFeatures)
{

 GArray      *lbounds[maxFeatures], *rbounds[maxFeatures];
 gint                    i, num_features, num_layers, *layer_list;
 gint                    width, height, bytes, bg_bytes, bg_height, bg_width;
 gint                    *featureList;
 gulong      arraysize;
 boundary    boundaryInst;
 GimpDrawable  *wk_drawable, *bg_drawable, *ac_drawable, *orig_drawable;
 GimpPixelRgn  maskPR, bgPR;
 gint32      activeLayer, workingLayer, imageID;
```

```
gboolean    add_OK, setMaskOK, preprocessOK, lowerOK;
gint                    first_mask_line, last_mask_line, lowerThresh, upperThresh;
GimpParam               *return_vals;
gint                    nreturn_vals;
gboolean    result=TRUE;
gint                    max_dim, min_dim, new_height, new_width;
gfloat      blur_radius;
gdouble     scaling_factor, target_ratio, img_ratio, percentage;

////////////////////////////////////////////////////////
//  Initial processing: layer add and preprocess
////////////////////////////////////////////////////////

////////////////////////////////////////////////////////
//  Autocrop and resize
////////////////////////////////////////////////////////


activeLayer = gimp_image_get_active_layer(image);
// Test: swap the processing of added and original layers
// Copy active layer and add it
workingLayer = gimp_layer_copy(activeLayer);
add_OK = gimp_image_add_layer(image, workingLayer, -1);

if(!add_OK)
        printf("Failure occurred while adding image\n");

layer_list = gimp_image_get_layers(image, &num_layers);

for(i=0; i<num_layers; i++)
  printf("The layers are: %d %s\n", i, gimp_layer_get_name(layer_list[i]));
```

```
  wk_drawable = gimp_drawable_get(gimp_layer_mask(activeLayer));
  ac_drawable = gimp_drawable_get(activeLayer);
  return_vals = gimp_run_procedure("plug_in_autocrop",
                                                    &nreturn_vals,
                                                    GIMP_PDB_INT32,
GIMP_RUN_NONINTERACTIVE,
                                                    GIMP_PDB_IMAGE, image,
                                                    GIMP_PDB_DRAWABLE,
wk_drawable->id,
                                                    GIMP_PDB_END);

  result &= show_status("autocrop", return_vals);

  gimp_drawable_flush(wk_drawable);
  gimp_drawable_update(wk_drawable->id, 0, 0, wk_drawable->width, wk_drawable-
>height);
  gimp_drawable_flush(ac_drawable);
  gimp_drawable_update(ac_drawable->id, 0, 0, ac_drawable->width, ac_drawable-
>height);

  wk_drawable = gimp_drawable_get(gimp_layer_mask(activeLayer));
  width = wk_drawable->width;
  height = wk_drawable->height;
  bytes  = wk_drawable->bpp;

// Prepare to preprocess background of copy
  if((gimp_layer_get_edit_mask(workingLayer)))
            setMaskOK = gimp_layer_set_edit_mask(workingLayer, 0);

  bg_drawable = gimp_drawable_get(workingLayer);
  imageID = gimp_drawable_image(bg_drawable->id);
```

```
if(!(gimp_layer_get_edit_mask(activeLayer)))
        setMaskOK = gimp_layer_set_edit_mask(activeLayer, 1);

// We want the changes drawn to the original layer mask
wk_drawable = gimp_drawable_get(gimp_layer_mask(activeLayer));

bg_drawable = gimp_drawable_get(workingLayer);
printf("Got the working layer drawable\n");

/* Get the size of the input image. (This will/must be the same
 *  as the size of the output image.
 */

width  = wk_drawable->width;
height = wk_drawable->height;
bytes  = wk_drawable->bpp;
bg_bytes = bg_drawable->bpp;
bg_width = bg_drawable->width;
bg_height = bg_drawable->height;

printf("Dimensions of copied layer drawable: width %d height %d bytes %d\n", width,
height, bytes);
printf("Dimensions of active layer drawable: width %d height %d bytes %d\n",
bg_width, bg_height, bg_bytes);

////////////////////////////////////////////////////
// Re-scale, and re-sample for threshhold level
////////////////////////////////////////////////////

// Rescale
```

```
max_dim = 300;
min_dim = 200;
target_ratio = 1.5;
scaling_factor = 1;

if(width > height)
{
 img_ratio = (gdouble)width/(gdouble)height;

 if(img_ratio > target_ratio)
 {
            new_width = max_dim;
            scaling_factor = (gdouble)max_dim/(gdouble)width;
            new_height = (gint)((gdouble)height * scaling_factor);
 }
 else
 {
            new_height = min_dim;
            scaling_factor = (gdouble)min_dim/(gdouble)height;
            new_width = (gint)((gdouble)width * scaling_factor);
 }
}
else
{
 img_ratio = (gdouble)height/(gdouble)width;
 if(img_ratio > target_ratio)
 {
            new_height = max_dim;
            scaling_factor = (gdouble)max_dim/(gdouble)height;
            new_width = (gint)((gdouble)width * scaling_factor);
 }
```

```
  else
  {
            new_width = min_dim;
            scaling_factor = (gdouble)min_dim/(gdouble)width;
            new_height = (gint)((gdouble)height * scaling_factor);
  }
 }

 gimp_image_scale(image, new_width, new_height);
 gimp_drawable_flush(wk_drawable);
 gimp_drawable_flush(bg_drawable);
 gimp_drawable_update(wk_drawable->id, 0, 0, wk_drawable->width, wk_drawable-
>height);
 gimp_drawable_update(bg_drawable->id, 0, 0, bg_drawable->width, bg_drawable-
>height);

 if(!(gimp_layer_get_edit_mask(activeLayer)))
            setMaskOK = gimp_layer_set_edit_mask(activeLayer, 1);

 // We want the changes drawn to the original layer mask
 wk_drawable = gimp_drawable_get(gimp_layer_mask(activeLayer));
 ac_drawable = gimp_drawable_get(activeLayer);

 bg_drawable = gimp_drawable_get(workingLayer);
 imageID = gimp_drawable_image(bg_drawable->id);
 printf("Got the working layer drawable\n");

 width  = wk_drawable->width;
 height = wk_drawable->height;
 bytes  = wk_drawable->bpp;
 bg_bytes = bg_drawable->bpp;
```

```
 bg_width = bg_drawable->width;
 bg_height = bg_drawable->height;

 printf("Dimensions of scaled copied layer drawable: width %d height %d bytes %d\n",
width, height, bytes);
 printf("Dimensions of scaled active layer drawable: width %d height %d bytes %d\n",
bg_width, bg_height, bg_bytes);

 /*  initialize the pixel regions  */

 gimp_pixel_rgn_init (&maskPR, wk_drawable,
                      0, 0, width, height, FALSE, FALSE);


 printf("Initialized the pixel regions\n");
 arraysize = (gulong)(sizeof(boundaryInst));

 for(i=0; i<maxFeatures; i++)
 {
   lbounds[i] = g_array_new(FALSE, FALSE, arraysize);
   rbounds[i] = g_array_new(FALSE, FALSE, arraysize);
 };

 // Allocate memory for array to keep track of number of features per row
 featureList = g_new0(gint, height);

 num_features = findFeatures(&first_mask_line, &last_mask_line,
                             maxFeatures, lbounds, rbounds, &maskPR,
                             featureList, width, height,
                             bytes);
```

```
 printf("Found features\n");

 #ifdef DEBUG
            printf("Pass #1: about to show edges\n");
 #endif

#ifdef EDGE_DEBUG
  printf("Finished with first pass\n");
  printf("Number of features found: %d\n", num_features);
  printf("First mask line: %d  last mask line: %d\n", first_mask_line, last_mask_line);
#endif


 // Threshhold based on intensities created in preprocessing
 if((gimp_layer_get_edit_mask(workingLayer)))
 {
            setMaskOK = gimp_layer_set_edit_mask(workingLayer, 0);

    if(!setMaskOK)
            printf("Unable to set edit mask for working layer\n");
    else
            printf("Set edit mask for working layer OK\n");
 }

 bg_drawable = gimp_drawable_get(workingLayer);
 orig_drawable = gimp_drawable_get(layer_list[2]);
 gimp_pixel_rgn_init (&bgPR, orig_drawable,
                        0, 0, width, height, FALSE, FALSE);

 printf("About to get threshhold level\n");
```

```
 lowerThresh = getThreshholdLevel(&upperThresh, maxFeatures, lbounds, rbounds,
                                                    &percentage, &bgPR, &maskPR,
featureList,
                                                    bg_bytes, bytes, width, height);

 #ifdef DEBUG
            printf("Threshhold levels: lower %d upper %d\n", lowerThresh,
upperThresh);
 #endif


 blur_radius = get_blur_radius(percentage);

 preprocessOK = preprocess(imageID, bg_drawable->id, blur_radius);
 printf("preprocessing with blur radius %f\n", (float)blur_radius);
 if(!preprocessOK)
   printf("Error in erode/blur routine\n");

gimp_threshold(bg_drawable->id, lowerThresh, upperThresh);
gimp_drawable_flush(bg_drawable);
gimp_drawable_merge_shadow(bg_drawable->id, TRUE);
gimp_drawable_update(bg_drawable->id, 0, 0, width, height);


layer_list = gimp_image_get_layers(image, &num_layers);

for(i=0; i<num_layers; i++)
   printf("The layers are: %d %s\n", i, gimp_layer_get_name(layer_list[i]));

printf("Removing layer\n");
lowerOK= gimp_image_remove_layer(imageID, layer_list[1]);
```

```
 if(!lowerOK)
   printf("Couldn't remove layer\n");
 else
   printf("Layer removed\n");

 activeLayer = gimp_image_flatten(imageID);
 drawable = gimp_drawable_get(activeLayer);

 gimp_drawable_flush(drawable);
 gimp_drawable_update(drawable->id, 0, 0, width, height);
#ifdef DEBUG
                printf("Releasing memory\n");
#endif


  // Release memory allocated for arrays
  for(i=0; i<maxFeatures; i++)
  {
   g_array_free(lbounds[i], TRUE);
   g_array_free(rbounds[i], TRUE);
  }

  // Release dynamically-allocated memory
   g_free(featureList);

   printf("Done processing\n");

 //  update the timred region  //

/*
// Put the scratch image in the background to view mask changes
```

```
 lowerOK = gimp_image_lower_layer(imageID, bg_drawable->id);
 if(!lowerOK)
   printf("Error lowering scratch layer\n");
*/

}

static gboolean
show_status(gchar *procedure_name, GimpParam *return_val)
{
 gboolean result = TRUE;

 printf("%s returned with status ", procedure_name);
 switch(return_val[0].data.d_status)
 {
   case 0: printf("GIMP_PDB_EXECUTION_ERROR\n");
                result = FALSE;
                break;
   case 1: printf("GIMP_PDB_CALLING_ERROR\n");
                result = FALSE;
                break;
   case 2: printf("GIMP_PDB_PASS_THROUGH\n");
                result = FALSE;
                break;
   case 3: printf("GIMP_PDB_SUCCESS\n");
                break;
   case 4: printf("GIMP_PDB_CANCEL\n");
                result = FALSE;
 }

 return result;
```

```
}


static gboolean
preprocess(gint32 imageID, gint32 drawableID, gdouble blur)
{
 gint                nreturn_vals;
 GimpParam           *return_vals;
 gdouble     matrix[9] = {1,1,1,1,1,1,1,1,1};
 gboolean result = TRUE;

// 1) Erode with full 3x3 matrix
// 2) Gaussian blur IIR


 return_vals = gimp_run_procedure("plug_in_erode",
                                  &nreturn_vals,
                                  GIMP_PDB_INT32, GIMP_RUN_NONINTERACTIVE,
                                  GIMP_PDB_IMAGE, imageID,
                                  GIMP_PDB_DRAWABLE, drawableID,
                                  GIMP_PDB_INT32, 9,
                                  GIMP_PDB_FLOATARRAY, matrix,
                                  GIMP_PDB_INT32, 0,
                                  GIMP_PDB_END);

 result &= show_status("Erosion", return_vals);
 if(nreturn_vals != 1)
 {
   printf("Number of return values not 1\n");
   result = FALSE;
 }
```

```
 return_vals = gimp_run_procedure("plug_in_gauss_iir",
                                  &nreturn_vals,
                                  GIMP_PDB_INT32, GIMP_RUN_NONINTERACTIVE,
                                  GIMP_PDB_IMAGE, imageID,
                                  GIMP_PDB_DRAWABLE, drawableID,
                                  GIMP_PDB_FLOAT, blur,
                                  GIMP_PDB_INT32, 1,
                                  GIMP_PDB_INT32, 1,
                                  GIMP_PDB_END);


 result &= show_status("Gaussian_blur_iir", return_vals);
 if(nreturn_vals != 1)
 {
   printf("Number of return values not 1\n");
   result = FALSE;
 }

 return result;
}

static gint
findFeatures(gint *first_mask_line, gint *last_mask_line,
             gint maxFeatures, GArray *lbounds[], GArray *rbounds[],
             GimpPixelRgn *maskPR,
             gint *featureList, gint width, gint height,
             gint bytes)
{
 gint row, col, lfeaturenum, rfeaturenum, maxFeatureNum;
 gint lastmask, maskbyte, i;
```

```
gboolean newL, newR, sawLeftEdge;
gint maxRfeaturenum, maxLfeaturenum, xoff;
guchar *maskrow;
boundary boundaryInst, *LBInst, *RBInst;

*first_mask_line=-1;
boundaryInst.edge=-1;
maxRfeaturenum = 0;
maxLfeaturenum = 0;
maxFeatureNum=0;
RBInst = NULL;
LBInst = NULL;

maskrow = g_new (guchar, width * bytes);

for (row = 0; row < height; row++)
{
  // Initialize the row monitoring variables

  lastmask = 0;    // Value of last mask pixel
  maskbyte = -1;   // Value of current mask pixel
  lfeaturenum = -1;  // Number of left mask edges, per row
  rfeaturenum = -1;  // Number of right mask edges, per row
  newL = FALSE;    // Flag for left edge seen, but in_pix not set
  newR = FALSE;    // Flag for right edge seen, but out_pix not set
  sawLeftEdge = FALSE;

  // Add a struct to each array to correspond to the new row
  for(i=0; i<maxFeatures; i++)
  {
   g_array_append_val(lbounds[i], boundaryInst);
```

```
 g_array_append_val(rbounds[i], boundaryInst);
}
// grab the row or column

gimp_pixel_rgn_get_row (maskPR, maskrow, 0, row, width);


xoff = 0;

lastmask=0;
for (col = 0; col < width; col++)
{
        // Working strictly with grayscale values, threshholded

        // For the mask, there is only one byte
        maskbyte = maskrow[xoff];

 // Make sure all mask bytes treated as 0 or 1
        if((maskbyte > 0) && (maskbyte < 128))
          maskbyte = 0;
        else if(maskbyte < 255)
          maskbyte = 255;

 //if((row>150)&&(row<161))
         //  printf("Row %d col %d mask: %d\n", row, col, maskbyte);

        if (maskbyte != lastmask)
        {
          if (lastmask == 0) // Found left boundary of feature
          {
                  if (*first_mask_line == -1)
```

```
                    *first_mask_line = row;
                    *last_mask_line = row;

                    newL = TRUE;
                    lfeaturenum++;
                    //printf("lfeaturenum: %d\n", lfeaturenum);
                    sawLeftEdge = TRUE;

                    LBInst = &g_array_index(lbounds[lfeaturenum], boundary,
row);
                    LBInst->edge = col;
            #ifdef EDGE_DEBUG
                    {
                      printf("Left edge found in row %d at %d\n", row, col);
                    }
            #endif
                    maxLfeaturenum = MAX(maxLfeaturenum,lfeaturenum+1);
        }
          else if((maskbyte == 0)||(col == width-1)) // Found right boundary of
feature
          {
           if(sawLeftEdge)
           {
                    newR = TRUE;
                    rfeaturenum++;
            //      printf("rfeaturenum: %d\n", rfeaturenum);

                    RBInst = &g_array_index(rbounds[rfeaturenum], boundary,
row);
                    RBInst->edge = col;
```

```
            #ifdef EDGE_DEBUG
                        {
                          printf("Right edge found in row %d at %d\n", row, col);
                        }
            #endif
            #ifdef EDGE_DEBUG
                          printf("Featurenum: %d Right edge value: %d\n",
rfeaturenum, RBInst->edge);
            #endif
                        sawLeftEdge = FALSE;
                        maxRfeaturenum = MAX(maxRfeaturenum,rfeaturenum+1);
               }
              }
            }
            else if(col == width-1) // Found right boundary of feature
            {
               if(sawLeftEdge)
               {
                        newR = TRUE;
                        rfeaturenum++;
            //          printf("rfeaturenum: %d\n", rfeaturenum);

                        RBInst = &g_array_index(rbounds[rfeaturenum], boundary,
row);
                        RBInst->edge = col;

            #ifdef EDGE_DEBUG
                        {
                          printf("Right edge found in row %d at %d\n", row, col);
                        }
            #endif
```

```
            #ifdef EDGE_DEBUG
                        printf("Featurenum: %d Right edge value: %d\n",
rfeaturenum, RBInst->edge);
            #endif
                        sawLeftEdge = FALSE;
                        maxRfeaturenum = MAX(maxRfeaturenum,rfeaturenum+1);
                }
            }

      lastmask = maskbyte;

      xoff++;

    }

    maxFeatureNum = MAX(lfeaturenum, rfeaturenum);
    // Keep track of how many features this row
    //printf("MAX(lfeaturenum, rfeaturenum)=%d\n", maxFeatureNum);
    featureList[row] = maxFeatureNum + 1;
    //printf("row=%d\n", row);

    gimp_progress_update ((double) (row) / 2 * height);
  }


  g_free(maskrow);

//  printf("Number of features: right %d  left %d\n", maxRfeaturenum,
maxLfeaturenum);
  return MAX(maxRfeaturenum, maxLfeaturenum);
}
```

```
static void
showStoredEdges(gint maxFeatures, GArray *lbounds[], GArray *rbounds[],
                                gint first_mask_line, gint last_mask_line)
{
  gint prev_L, prev_R, L_stored_edge, R_stored_edge;
  gint i, row;
  boundary *LBInst, *RBInst;

  LBInst = NULL;
  RBInst = NULL;
  prev_L = 0;
  prev_R = 0;
  L_stored_edge = 0;
  R_stored_edge = 0;

  #ifdef DEBUG
            printf("In showStoredEdges\n");
  #endif
  for(i=0; i<maxFeatures; i++)
  {
#ifdef EDGE_DEBUG
   printf("\n--- Displaying edges for feature %d ---\n", i);
#endif
   for(row=first_mask_line; row<last_mask_line+1;row++)
   {
            LBInst = &g_array_index(lbounds[i], boundary, row);
            if(LBInst->edge != -1)
            {
#ifdef EDGE_DEBUG
              printf("Left edge at row %d col %d\n", row, LBInst->edge);
```

```
#endif
                L_stored_edge++;
              }
              RBInst = &g_array_index(rbounds[i], boundary, row);
              if(RBInst->edge != -1)
              {
#ifdef EDGE_DEBUG
                printf("Right edge at row %d col %d\n", row, RBInst->edge);
#endif
                R_stored_edge++;
              }
   }
#ifdef EDGE_DEBUG
   printf("Number of left edge markers after %d loops: %d\n", i+1, L_stored_edge);
   printf("Number of right edge markers after %d loops: %d\n", i+1, R_stored_edge);
#endif

   if(!((L_stored_edge == prev_L) && (R_stored_edge == prev_R)))
   {
              prev_L = L_stored_edge;
              prev_R = R_stored_edge;
   }
   else
              i=maxFeatures;
  }

#ifdef EDGE_DEBUG
   printf("Total number of left edge markers found: %d\n", L_stored_edge);
   printf("Total number of right edge markers found: %d\n", R_stored_edge);
#endif
  #ifdef DEBUG
```

```
            printf("Leaving showStoredEdges\n");
  #endif
}

static gfloat get_blur_radius(gdouble percentage)
{
  gfloat radius;

  if(percentage == 0.1)
    radius = 6.0;
  else if(percentage < 0.15)
    radius = 7.0;
  else if(percentage < 0.20)
            radius = 8.0;
  else if(percentage < 0.25)
            radius = 9.0;
  else if(percentage < 0.35)
            radius = 10.0;
  else
            radius = 12.0;

  return radius;
}

static gint
getThreshholdLevel(gint *u_thresh, gint maxFeatures,
                   GArray *lbounds[], GArray *rbounds[], gdouble
*percentage,
                   GimpPixelRgn *bgPR, GimpPixelRgn *maskPR, gint
*featureList,
                   gint bg_bytes, gint bytes, gint width, gint height)
```

```
{
 gint row, left_edge, right_edge, i, j;
 gint bg_byte, mask_byte;
 guchar *bgrow, *maskrow;
 gint threshhold_level = -1;
 gint32 pixel_count = 0;
 gint32 bg_pixel_count = 0;
 gint32 white_pixel_count = 0;
 gint32 levels[256], bglevels[256];
 boundary *LBInst, *RBInst;

 // Variables for autothreshhold
 gboolean no_close_min = FALSE;
 gint rel_sl_min[40];
 gint rel_sl_max[40];
 gint bad_sl_min[40];
 gint bad_sl_max[40];
 gint num_sl_min = 0;
 gint num_sl_max = 0;
 gint num_good_sl_min = 0;
 gint num_good_sl_max = 0;
 gint target;
 gint poss_max, poss_min, divisor, moving_sl_r_sum, moving_sl_l_sum;
 gint poss_sl_max, poss_sl_min, sl_offset;
 gdouble moving_slope[256];
 gdouble moving_average[256], moving_bg_average[256];
 gdouble max_moving_average, max_moving_bg_average;
 gint max_index, max_bg_index;

 gint bottom_sl_index, top_sl_index, window_offset;
 gint32 moving_sum;
```

```
FILE *coverage;
char *cov_file="/tmp/coverage";
char stat[20];
coverage = fopen(cov_file, "a");
if (!coverage)
  printf("Couldn't open coverage file");

poss_min = poss_max = moving_sum = 0;
poss_sl_min = poss_sl_max = 0;
divisor = 0;
rel_sl_min[0] = -1;
rel_sl_max[0] = -1;

for(i=0; i<40; i++)
{
  bad_sl_min[i]=0;
  bad_sl_max[i]=0;
}

//printf("l_thresh: %d\n", l_thresh);

// Histogram structure
for (i=0; i<256; i++)
{
 levels[i] = 0;
 bglevels[i] = 0;
}


printf("Looking at bgPR info: width: %d height: %d, origin x: %d, origin y: %d, bpp:
%d, rowstride: %d\n", bgPR->w, bgPR->h, bgPR->x, bgPR->y, bgPR->bpp, bgPR-
```

```
>rowstride);
 printf("Width: %d\n", width);
 bgrow = g_new (guchar, width * bg_bytes);
 maskrow = g_new (guchar, width * bytes);

 // Fill the histogram and total the number of pixels found
 // Look only at ROI
 for(row = 0; row < height; row++)
 {
  //  printf("Working on row %d\n", row);
    gimp_pixel_rgn_get_row(bgPR, bgrow, 0, row, width);
    gimp_pixel_rgn_get_row(maskPR, maskrow, 0, row, width);

   // printf("There are %d features in row %d\n", featureList[row], row);
    for (i=0; i<featureList[row]; i++)
    {
                LBInst = &g_array_index(lbounds[i], boundary, row);
                RBInst = &g_array_index(rbounds[i], boundary, row);

                left_edge = LBInst->edge;
                right_edge = RBInst->edge;

                //printf("Processing row %d from %d to %d\n", row, left_edge, right_edge);
                for(j=left_edge; j<right_edge+1; j++)
                {
                  bg_byte = bgrow[j*bg_bytes];
                  mask_byte = maskrow[j*bytes];
                  if(mask_byte==255)
                  {
                    levels[bg_byte]++;
                    //printf("Col: %d Adding to levels[%d]\n", j, bg_byte);
```

```
                pixel_count++;
              }
            }
  }
}

// Estimate maxima and minima using slope of approximate tangent
// to histogram of ROI

moving_sl_r_sum = 0;
moving_sl_l_sum = 0;
window_offset = 5;
max_moving_average = 0;
max_index = 0;
moving_sl_r_sum = 0;
moving_sl_l_sum = 0;

for(i=0; i<256; i++)
{
 if((i>window_offset-1) && (i<256 - window_offset))
 {
            bottom_sl_index = i-window_offset;
            top_sl_index = i+window_offset;
   if(i==window_offset)
   {
     for(j=bottom_sl_index; j<i; j++)
            {
              //printf("lsum: Adding %d from levels[%d]\n", levels[j], j);
              moving_sl_l_sum += levels[j];
            }
     for(j=i+1; j<=top_sl_index; j++)
```

```
                {
                  //printf("rsum: Adding %d from levels[%d]\n", levels[j], j);
                  moving_sl_r_sum += levels[j];
                }
        }
                else
                {
                  moving_sl_l_sum -= levels[bottom_sl_index-1];
                  moving_sl_l_sum += levels[i-1];
                  moving_sl_r_sum -= levels[i];
                  moving_sl_r_sum += levels[top_sl_index];
                }
                moving_slope[i] = (gdouble)(moving_sl_r_sum -
moving_sl_l_sum)/(2*(gdouble)window_offset);
                moving_average[i] = (gdouble)(moving_sl_r_sum +
moving_sl_l_sum)/(2*(gdouble)window_offset);
        if(moving_average[i]>max_moving_average)
                {
                  max_moving_average = moving_average[i];
                  max_index = i;
                }
      if((moving_slope[i] <= 0) && (moving_slope[i-1] > 0))
      {
                   rel_sl_max[num_sl_max] = i;
                   num_sl_max++;
      }
      else if((moving_slope[i] >= 0) && (moving_slope[i-1] < 0))
      {
                   rel_sl_min[num_sl_min] = i;
                   num_sl_min++;
      }
```

```
  }
  else
            moving_slope[i] = 0;
  //printf("Moving_slope[%d] = %f\n", i, (float)moving_slope[i]);
 }

printf("Checking initial number: bglevels[0]=%d\n", bglevels[0]);

// Add processing for background to get max moving average
// This will be baseline for threshholding
 for(row = 0; row < height; row++)
 {
  gimp_pixel_rgn_get_row(bgPR, bgrow, 0, row, width);
  gimp_pixel_rgn_get_row(maskPR, maskrow, 0, row, width);
  for(i=0; i<width; i++)
  {
            bg_byte = bgrow[i*bg_bytes];
            bglevels[bg_byte]++;
           // printf("Col: %d Adding to bglevels[%d]\n", i, bg_byte);
            bg_pixel_count++;
  }
 }

 // For comparison, get max moving average of entire background
 moving_sl_r_sum = 0;
 moving_sl_l_sum = 0;
 window_offset = 5;
 max_moving_bg_average = 0;
 max_bg_index = 0;
 moving_sl_r_sum = 0;
 moving_sl_l_sum = 0;
```

```
for(i=0; i<256; i++)
{
 if((i>window_offset-1) && (i<256 - window_offset))
 {
            bottom_sl_index = i-window_offset;
            top_sl_index = i+window_offset;
    if(i==window_offset)
    {
      for(j=bottom_sl_index; j<i; j++)
            {
              //printf("lsum: Adding %d from bglevels[%d]\n", bglevels[j], j);
              moving_sl_l_sum += bglevels[j];
            }
      for(j=i+1; j<=top_sl_index; j++)
            {
             // printf("rsum: Adding %d from bglevels[%d]\n", bglevels[j], j);
              moving_sl_r_sum += bglevels[j];
            }
    }
            else
            {
              moving_sl_l_sum -= bglevels[bottom_sl_index-1];
             //  printf("lsum: Subtracting %d from bglevels[%d]\n",
bglevels[bottom_sl_index-1], bottom_sl_index-1);
              moving_sl_l_sum += bglevels[i-1];
             // printf("lsum: Adding %d from bglevels[%d]\n", bglevels[i-1], i-1);
              moving_sl_r_sum -= bglevels[i];
             // printf("rsum: Subtracting %d from bglevels[%d]\n", bglevels[i], i);
              moving_sl_r_sum += bglevels[top_sl_index];
              //printf("rsum: Adding %d from bglevels[%d]\n", bglevels[top_sl_index],
```

```
top_sl_index);
                }
                moving_bg_average[i] = (gdouble)(moving_sl_r_sum +
moving_sl_l_sum)/(2*(gdouble)window_offset);
      //printf("moving_bg_average[%d]: %f\n", i, (float)moving_bg_average[i]);
      if(moving_bg_average[i]>max_moving_bg_average)
                {
                  max_moving_bg_average = moving_bg_average[i];
                 //  printf("New max_moving_bg_average: %f at index %d\n",
(float)max_moving_bg_average, max_bg_index);
                  max_bg_index = i;
                }
    }
  }


  printf("\nNumber of slope minima before noise reduction: %d slope minima occur at ",
num_sl_min+1);
  for(i=0; i<num_sl_min; i++)
  {
      printf("%d  ", rel_sl_min[i]);
  }
  printf("\nNumber of slope maxima before noise reduction: %d slope maxima occur at ",
num_sl_max+1);

  for(i=0; i<num_sl_max; i++)
  {
      printf("%d  ", rel_sl_max[i]);
  }

  num_good_sl_min = num_sl_min;
```

```
num_good_sl_max = num_sl_max;

// First pass at removing noisy maxima/minima:
// Remove all max/min that are as close together as the 1+the window
// and have equal indices
// Also remove those in a distance if half the window where the
// corresponding indices are offset by 1

for(i=0; i<num_sl_min; i++)
{
  if(i<num_sl_max)
  {
            if(abs(rel_sl_max[i] - rel_sl_min[i]) <= (window_offset+1))
            {
              bad_sl_min[i] = 1;
              bad_sl_max[i] = 1;
              num_good_sl_min--;
              num_good_sl_max--;
    }
            if(i>0)
            {
              if(abs(rel_sl_min[i-1] - rel_sl_max[i]) <= (window_offset+1)/2)
              {
                if(!bad_sl_min[i-1])
                {
                  bad_sl_min[i-1] = 1;
                  num_good_sl_min--;
                }
                if(!bad_sl_max[i])
                {
                  bad_sl_max[i] = 1;
```

```
                    num_good_sl_max--;
                  }
                }
              }
    }
    else if((i>0) && (i<(num_sl_max+1)))
    {
                if(abs(rel_sl_max[i-1] - rel_sl_min[i]) <= (window_offset+1)/2)
                {
                  if(!bad_sl_min[i])
                  {
                    bad_sl_min[i] = 1;
                    num_good_sl_min--;
                  }
                  if(!bad_sl_max[i-1])
                  {
                    bad_sl_max[i-1] = 1;
                    num_good_sl_max--;
                  }
                }
    }
  }

  printf("\nNumber of slope minima after first noise reduction: %d slope minima occur at
", num_good_sl_min);
  for(i=0; i<num_sl_min; i++)
  {
    if(!bad_sl_min[i])
      printf("%d  ", rel_sl_min[i]);
  }
  printf("\nNumber of slope maxima after first noise reduction: %d slope maxima occur at
```

```
", num_good_sl_max);
 for(i=0; i<num_sl_max; i++)
 {
   if(!bad_sl_max[i])
     printf("%d  ", rel_sl_max[i]);
 }

 // Remove remaining max/min within (window_offset+1)/2,
 // comparing from greatest disparity in max/min index length down
 // to even indices
 if(num_good_sl_min > num_good_sl_max)
 {
   sl_offset = num_good_sl_min - num_good_sl_max;
   for(j=0; j<sl_offset+1; j++)
   {
     for(i=num_sl_min-1; i>j-1; i--)
     {
                if(bad_sl_min[i] || bad_sl_max[i-j])
                  continue;
                if(abs(rel_sl_min[i] - rel_sl_max[i-j]) <= (window_offset+1)/2)
                {
                  bad_sl_min[i] = 1;
                  bad_sl_max[i-j] = 1;
                  num_good_sl_min--;
                  num_good_sl_max--;
                  printf("removing rel_sl_min:%d rel_sl_max%d\n", rel_sl_min[i],
rel_sl_max[i-j]);
                }
     }
   }
 }
```

```
else
{
  sl_offset = num_good_sl_max - num_good_sl_min;
  for(j=0; j<sl_offset+1; j++)
  {
    for(i=num_sl_max-1; i>sl_offset-1; i--)
    {
            if(bad_sl_max[i]|| bad_sl_min[i-j])
              continue;
            if(abs(rel_sl_max[i] - rel_sl_min[i-j]) <= (window_offset+1)/2)
            {
              bad_sl_min[i-j] = 1;
              bad_sl_max[i] = 1;
              num_good_sl_min--;
              num_good_sl_max--;
              printf("removing rel_sl_min:%d rel_sl_max%d\n", rel_sl_min[i],
rel_sl_max[i-j]);
            }
    }
  }
}

printf("\nNumber of slope minima: %d slope minima occur at ", num_good_sl_min);
for(i=0; i<num_sl_min; i++)
{
  if(!bad_sl_min[i])
    printf("%d  ", rel_sl_min[i]);
}
printf("\nNumber of slope maxima: %d slope maxima occur at ", num_good_sl_max);
for(i=0; i<num_sl_max; i++)
{
```

```
  if(!bad_sl_max[i])
    printf("%d  ", rel_sl_max[i]);
 }

 printf("\nMax moving average %f at %d\n", (float)max_moving_average, max_index);
 printf("\nMax background average %f at %d\n", (float)max_moving_bg_average,
max_bg_index);

 // Determine lower threshhold level
 for(i=0; i<num_sl_min; i++)
 {
  if(!bad_sl_min[i])
  {
            printf("Examining relative minimum %d\n", rel_sl_min[i]);
            if(rel_sl_min[i] > max_bg_index)
            {
              printf("Found an appropriate minimum:%d\n", rel_sl_min[i]);
              break;
            }
  }
 }
 if(i < num_sl_min)
 {
  if((rel_sl_min[i] - max_bg_index) < 60)
    threshhold_level = rel_sl_min[i];
  else
  {
            if((rel_sl_min[i] - max_bg_index) >= 120)
              threshhold_level = (int)(rel_sl_min[i] + 3*max_bg_index)/4;
            else if((rel_sl_min[i] - max_bg_index) >= 90)
              threshhold_level = (int)(rel_sl_min[i] + 2*max_bg_index)/3;
```

```
            else
              threshhold_level = (int)(rel_sl_min[i] + max_bg_index)/2;
            no_close_min = TRUE;
  }
}
else
  threshhold_level = (int)(max_bg_index + 255)/2;

printf("Lower threshhold: %d\n", threshhold_level);

// Now that we know the threshhold, determine the percentage of pixels
// falling above it

for(i=255; i>threshhold_level; i--)
{
  white_pixel_count+=levels[i];
}

*percentage = (gdouble)white_pixel_count/(gdouble)pixel_count;

printf("Percentage of pixels to remain white: %f\n", (float)*percentage);
sprintf(stat, "%f\n", (float)*percentage);
fprintf(coverage, stat);

if((*percentage < 0.1))
{
  target = (gint)(0.1 * pixel_count);
  white_pixel_count = 0;
  for(i=255; white_pixel_count < target; i--)
    white_pixel_count += levels[i];
```

```
    printf("Recalculated threshhold: %d\n", i);
    if(i>max_bg_index)
      threshhold_level = i;
    else
      threshhold_level = max_bg_index;

    *percentage = 0.1;
    printf("Recalculated percentage of pixels to remain white: %f\n", (float)*percentage);
  }

 *u_thresh = 255;

  j = i+1;
  while(j<num_sl_min-1)
  {
    if(rel_sl_min[j] - threshhold_level > 60)
      break;
    j++;
  }
  if(j < (num_sl_min-1))
    *u_thresh = rel_sl_min[j];

  printf("Final lower threshhold: %d upper threshhold: %d\n", threshhold_level,
*u_thresh);
  g_free(bgrow);
  fclose(coverage);
  return threshhold_level;
}
```

```
/* Line detection plug-in for the GIMP -- Version 0.1
 * Copyright © 2002 Cindy Huyser <chuyser@io.com>
 * Based  on convolution plug-in, copyright (C) 1997 Lauri Alanko <la@iki.fi>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 *
 * The GNU General Public License is also available from
 * http://www.fsf.org/copyleft/gpl.html
 *
 *
 * CHANGELOG:
 *
 *
 * TODO:
 *
 * ** Build masks for line detection (done)
 * ** Decide on output for lines (beside the image) (done)
 * ** Store midpoints of lines (done)
 * ** Build colinear lines
```

```
 * ** Calculate 6 nearest neighbors (done) (still crashes occasionally)
 * ** Calculate relative angles for nearest neighbors (done)
 * ** Calculate relative line lengths of nearest neighbors (done)
 */

#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <sys/types.h>
#ifdef HAVE_UNISTD_H
#include <unistd.h>
#include <math.h>
#endif

#include <gtk/gtk.h>

#include <libgimp/gimp.h>
#include <libgimp/gimpui.h>

#include "libgimp/stdplugins-intl.h"
#include <libpq-fe.h>
#include <libpgeasy.h>

//#define DEBUG
//#define DEBUG_LINES
//#define DEBUG_NEIGHBOR
//#define DEBUG_LENGTH
//#define DEBUG_ANGLE
//#define DEBUG_LRATIO
#define DENOISE
```

```
//#define DEBUG_INTERCEPT


#define VERTICAL 5   // Indices of lines detected
#define HORIZONTAL 0

GimpDrawable *drawable;
gint32 image;

gchar * const line_labels[] =
{
 N_("0 degrees"),
 N_("20 degrees"),
 N_("45 degrees"),
 N_("60 degrees"),
 N_("75 degrees"),
 N_("90 degrees"),
 N_("105 degrees"),
 N_("120 degrees"),
 N_("135 degrees"),
 N_("160 degrees")
};

/* Declare local functions. */
static void query (void);
static void run   (gchar     *name,
                   gint       nparams,
                   GimpParam  *param,
                   gint       *nreturn_vals,
                   GimpParam **return_vals);
```

```
static gint dialog (void);

static void doit       (void);
static void check_config (void);

GimpPlugInInfo PLUG_IN_INFO =
{
 NULL,   /* init_proc  */
 NULL,   /* quit_proc  */
 query,  /* query_proc */
 run,    /* run_proc   */
};

gint sx1, sy1, sx2, sy2;
gint run_flag = 0;

typedef struct
{
 gint      lines[10];
} config;

typedef struct
{
 gint          matrix[5][5];
} line_mask;

/*
typedef struct
{
 gint begin_x;
 gint begin_y;
```

```
  gint end_x;
  gint end_y;
  node *next;
} node;
*/

// Default: detect horizontal lines
const config default_config =
  {{ 1, 0, 0, 1, 0, 0, 0, 0, 0, 0 }};

const int line_0[5][5] =
  {
    { 0, 0, 0, 0, 0},
    { 0,-1,-1,-1, 0},
    { 0, 2, 2, 2, 0},
    { 0,-1,-1,-1, 0},
    { 0, 0, 0, 0, 0}
  };

const int line_1[5][5] =
  {
    {0, 0, 0, 0, 0},
    {0, 0,-1,-1, 0},
    {0,-1, 2, 2, 0},
    {0, 2,-1,-1, 0},
    {0,-1, 0, 0, 0}
  };

const int line_2[5][5] =
  {
    { 0, 0, 0, 0, 0},
```

```
  { 0, 0,-1, 2,-1},
  { 0,-1, 2,-1, 0},
  {-1, 2,-1, 0, 0},
  { 0, 0, 0, 0, 0}
 };

const int line_3[5][5] =
 {
  { 0, 0, 0, 0, 0},
  { 0, 0,-1, 2,-1},
  { 0,-1, 2,-1, 0},
  { 0,-1, 2,-1, 0},
  { 0, 0, 0, 0, 0}
 };

const int line_4[5][5] =
 {
  { 0, 0, 0, 0, 0},
  { 0,-1, 2,-1, 0},
  { 0,-1, 2,-1, 0},
  {-1, 2,-1, 0, 0},
  { 0, 0, 0, 0, 0}
 };

const int line_5[5][5] =
 {
  { 0, 0, 0, 0, 0},
  { 0,-1, 2,-1, 0},
  { 0,-1, 2,-1, 0},
  { 0,-1, 2,-1, 0},
  { 0, 0, 0, 0, 0}
```

```
};

const int line_6[5][5] =
{
  { 0, 0, 0, 0, 0},
  {-1, 2,-1, 0, 0},
  { 0,-1, 2,-1, 0},
  { 0,-1, 2,-1, 0},
  { 0, 0, 0, 0, 0}
};

const int line_7[5][5] =
{
  { 0, 0, 0, 0, 0},
  { 0,-1, 2,-1, 0},
  { 0,-1, 2,-1, 0},
  { 0, 0,-1, 2,-1},
  { 0, 0, 0, 0, 0}
};

const int line_8[5][5] =
{
  { 0, 0, 0, 0, 0},
  {-1, 2,-1, 0, 0},
  { 0,-1, 2,-1, 0},
  { 0, 0,-1, 2,-1},
  { 0, 0, 0, 0, 0}
};

const int line_9[5][5] =
{
```

```
   { 0, 0, 0, 0, 0},
   { 0,-1,-1, 0, 0},
   { 0, 2, 2,-1, 0},
   { 0,-1,-1, 2, 0},
   { 0, 0, 0,-1, 0}
  };

config my_config;

struct
{
  GtkWidget *lines[10];
  GtkWidget *ok;
} my_widgets;


MAIN ()

static void
query (void)
{
  static GimpParamDef args[] =
  {
    { GIMP_PDB_INT32,    "run_mode", "Interactive, non-interactive" },
    { GIMP_PDB_IMAGE,    "image",    "Input image " },
    { GIMP_PDB_DRAWABLE, "drawable", "Input drawable" },
    { GIMP_PDB_INT32, "argc_lines", "The number of elements in the following array.
Should be always 10." },
    { GIMP_PDB_INT8ARRAY, "lines", "The lines chosen for detection (1 = chosen, 0 =
not chosen. Angle of lines by index: [0]: 0, [1]: 20, [2]: 45, [3]: 60, [4]: 75, [5]: 90, [6]:
105, [7]: 120, [8]: 135, [9]: 160" },
```

```
  };
  static gint nargs = sizeof (args) / sizeof (args[0]);

  gimp_install_procedure ("plug_in_line_detect",
                                        "A line detector for thresholded images",
                                        "",
                                        "Cindy Huyser",
                                        "Cindy Huyser",
                                        "2002",
                                        N_("<Image>/Filters/Generic/Line detector..."),
                                        "GRAY*",
                                        GIMP_PLUGIN,
                                        nargs, 0,
                                        args, NULL);
}

static void
run (gchar      *name,
     gint       nparams,
     GimpParam  *param,
     gint       *nreturn_vals,
     GimpParam **return_vals)
{
  static GimpParam  values[1];
  GimpRunModeType   run_mode;
  GimpPDBStatusType status = GIMP_PDB_SUCCESS;
  gint              x;

  INIT_I18N_UI();

  (void) name; /* Shut up warnings about unused parameters. */
```

```
*nreturn_vals = 1;
*return_vals = values;

run_mode = param[0].data.d_int32;

/*  Get the specified image  */
image = param[1].data.d_image;

/*  Get the specified drawable  */
drawable = gimp_drawable_get (param[2].data.d_drawable);

/*  The plug-in is not able to handle images smaller than 3 pixels  */
if (drawable->width < 3 || drawable->height < 3)
 {
  g_message (_("Line detector does not work\non layers smaller than 3 pixels."));
  status = GIMP_PDB_EXECUTION_ERROR;
  values[0].type = GIMP_PDB_STATUS;
  values[0].data.d_status = status;
  return;
 }

my_config = default_config;
if (run_mode == GIMP_RUN_NONINTERACTIVE)
 {
  if (nparams != 5)
            status = GIMP_PDB_CALLING_ERROR;
  else
            {
             if (param[3].data.d_int32 != 10) {
               status = GIMP_PDB_CALLING_ERROR;
```

```
            } else {
               for (x = 0; x < 10; x++)
               {
                        my_config.lines[x]=param[4].data.d_int8array[x];
               }
            }

          }
 }
else
 {
  gimp_get_data ("plug_in_line_detect", &my_config);

  if (run_mode == GIMP_RUN_INTERACTIVE)
          {
           /*  Oh boy. We get to do a dialog box, because we can't really
            *  expect the user to set us up with the right values using gdb.
            */
           check_config ();
           if (!dialog())
            {
             /* The dialog was closed, or something similarly evil happened. */
             status = GIMP_PDB_EXECUTION_ERROR;
            }
          }
 }

if (status == GIMP_PDB_SUCCESS)
 {
  /*  Make sure that the drawable is gray or RGB color  */
  if (gimp_drawable_is_gray(drawable->id))
```

```
            {
             gimp_progress_init (_("Applying line detection"));
             gimp_tile_cache_ntiles (2 * (drawable->width /
                                                  gimp_tile_width () + 1));

             doit ();

             if (run_mode != GIMP_RUN_NONINTERACTIVE)
               gimp_displays_flush ();

             if (run_mode == GIMP_RUN_INTERACTIVE)
               gimp_set_data ("plug_in_line_detect", &my_config, sizeof (my_config));
            }
    else
            {
             status = GIMP_PDB_EXECUTION_ERROR;
            }
    gimp_drawable_detach (drawable);
   }

 values[0].type = GIMP_PDB_STATUS;
 values[0].data.d_status = status;
}

/*  A generic wrapper to gimp_pixel_rgn_get_row which handles unlimited
 *  wrapping or gives you transparent regions outside the image
 */

#ifdef DEBUG
static void show_location(gint row, gint col)
{
```

```
  printf("Processing row %d col %d\n", row, col);
}
#endif

#ifdef DEBUG
static void
show_source_array(guchar **srcrow, gint xoff, gint bytes)
{
  gint x, y;

  printf("Source array elements processed:");
  for(y=0; y<5; y++)
  {
   printf("\n");
   for(x=0; x<5; x++)
   {
              if(srcrow[y][xoff + x * bytes] > 99)
                printf("%d ", srcrow[y][xoff + x * bytes]);
              else if(srcrow[y][xoff + x * bytes] > 9)
                printf(" %d ", srcrow[y][xoff + x * bytes]);
              else
                printf("  %d ", srcrow[y][xoff + x * bytes]);

   }
  }
  printf("\n");
}
#endif

#ifdef DEBUG
static void
```

```
show_line_matrix(const gint line_mx[5][5])
{
  gint x, y;

  printf("Line matrix elements");
  for(y=0; y<5; y++)
  {
   printf("\n");
   for(x=0; x<5; x++)
   {
              if(line_mx[y][x] < 0)
                printf("%d ", line_mx[y][x]);
              else
                printf(" %d ", line_mx[y][x]);
   }
  }
  printf("\n");
}
#endif

struct pixel{
  gint row;
  gint col;
};

struct midpt{
  gfloat row;
  gfloat col;
};

struct neighbor{
```

```
  gfloat distance;
  struct list_el * line;
  gint index;
  struct neighbor * next;
};

struct list_el{
  gint length;
  struct pixel begin;
  struct pixel end;
  struct midpt midpoint;
  struct list_el * next;
  struct neighbor * n_list;
  gint num_neighbors;
  gint weak_pts;
  gdouble intercept;
};

static gint get_angle_index(gdouble gamma)
{
  gint angle_index;

  angle_index = 0;

  while(gamma > ((1 + (gdouble)angle_index) * 2 * M_PI/36))
    angle_index++;

  return angle_index;
}

static gint get_length_index(gdouble length_ratio)
```

```
{
  gint length_index = 0;

  while (length_ratio > ((1 + (gdouble)length_index))/ 12)
    length_index++;

  return length_index;
}

static gfloat compute_distance(struct list_el *L1, struct list_el *L2)
{
  gfloat row_diff;
  gfloat col_diff;

  row_diff = L1->midpoint.row - L2->midpoint.row;
  col_diff = L1->midpoint.col - L2->midpoint.col;

  return sqrt(row_diff * row_diff + col_diff * col_diff);
}

static void add_to_neighbor_list(struct list_el *L,
                                 gint line_index,
                                 gfloat distance,
                                 struct neighbor ** n_list,
                                 gint num_neighbors)
{
  struct neighbor *n_ptr, *n_head, *new_n, *n_prev, *n_high, n_item;
  gint traversed = 0;
  gint node_num = 0;

#ifdef DEBUG_NEIGHBOR
```

```
  printf("Inside add_to_neighbor_list\n");
#endif

  n_ptr = *n_list;
  n_head = *n_list;
  n_prev = NULL;

  if(num_neighbors < 6)
  {
            new_n = (struct neighbor *)malloc(sizeof(n_item));
            if(!(new_n))
            {
              printf("Failed to allocate memory for new neighbor\n");
              return;
            }
            else
            {
              new_n->distance = distance;
              new_n->line = L;
              new_n->index = line_index;
            }

            if(!n_head)
            {
              n_head = new_n;
              n_head->next = NULL;
              *n_list = n_head;
            }
            else
            {
              if(distance > (n_head->distance))
```

```
                {
                        n_head = new_n;
                        n_head->next = n_ptr;
                        *n_list = n_head;
                }
                else
                {
                        while(n_ptr && (distance < n_ptr->distance))
                        {
                          n_prev = n_ptr;
                          n_ptr = n_ptr->next;
                          node_num++;
                        }
                        new_n->next = n_ptr;
                        n_prev->next = new_n;
                }
              }
              #ifdef DEBUG_NEIGHBOR
              printf("Adding neighbor node in position %d\n", node_num);
              #endif
  }
  else
  {
#ifdef DEBUG_NEIGHBOR
              printf("Finding place for neighbor with distance %f\n", distance);
#endif

              n_high = n_head;
              n_ptr = n_head;
#ifdef DEBUG_NEIGHBOR
              printf("Distance of first neighbor: %f\n", n_ptr->distance);
```

```
#endif
            if((distance < n_ptr->distance) && (distance >= n_ptr->next->distance))
            {
                        n_head->distance = distance;
                        n_head->line = L;
                        n_head->index = line_index;
            }
            else
            {
               n_head = n_head->next;
               n_ptr = n_head;
#ifdef DEBUG_NEIGHBOR
               printf("Not an end point: looking for place in list\n");
#endif
               while(traversed < (num_neighbors - 1))
               {
                 traversed++;
#ifdef DEBUG_NEIGHBOR
                        printf("Traversing: %d\n", traversed);
                        printf("Distance of neighbor %d: %f\n", traversed, n_ptr-
>distance);
#endif
                 if((distance < n_ptr->distance) &&
                  ((n_ptr->next == NULL) || (distance >= n_ptr->next->distance)))

                 {
                          n_high->distance = distance;
                          n_high->line = L;
                          n_high->index = line_index;
                          n_high->next = n_ptr->next;
                          n_ptr->next = n_high;
```

```
                        #ifdef DEBUG_NEIGHBOR
                          printf("Inserting neighbor in place %d\n", traversed);
                        #endif
                          break;
                      }

                      n_ptr = n_ptr->next;
                    }
                  }
                  *n_list = n_head;
  }
}

static gboolean in_neighbor_list(struct list_el *L,
                                                    struct neighbor *n_list)
{
  gboolean in_list = FALSE;

  struct neighbor *n_ptr;

  n_ptr = n_list;
  if (!n_ptr)
  {
    return in_list;
  }

  while(n_ptr)
  {
    // Check to see if pointing to same address
    if(n_ptr->line == L)
    {
```

```
            in_list = TRUE;
            break;
    }
    n_ptr = n_ptr->next;
  }

  return in_list;
}

static void find_nearest_neighbors(gint curr_index,
                                   struct list_el **curr_line,
                                   struct list_el **line_list)
{
  gint i, line_count;
  gfloat distance, compare_distance;
  struct list_el *list_ptr;
  struct neighbor *n_head;

#ifdef DEBUG_NEIGHBOR
  printf("Finding nearest neighbors\n");
#endif
  for(i=0; i<10; i++)
  {
    line_count = 0;
    list_ptr = line_list[i];
    while(list_ptr)
    {
#ifdef DEBUG_NEIGHBOR
            printf("Looking at line %d\n", line_count);
#endif
            if((i==curr_index) && (list_ptr == *curr_line))
```

```
                {
                 list_ptr = list_ptr->next;
                 continue;
                }

                distance = compute_distance(*curr_line, list_ptr);
                if(!(in_neighbor_list(list_ptr, (*curr_line)->n_list)))
                {
#ifdef DEBUG_NEIGHBOR
                 printf("num_neighbors: %d\n", (*curr_line)->num_neighbors);
#endif
                 if((*curr_line)->num_neighbors < 6)
                 {
#ifdef DEBUG_NEIGHBOR
                        printf("Adding a neighbor to line: distance %f\n", distance);
#endif
                        add_to_neighbor_list(list_ptr, i, distance,
                                                        &((*curr_line)->n_list),
                                                        (*curr_line)-
>num_neighbors);
                        (*curr_line)->num_neighbors++;
                 }
                 else
                 {
                        n_head = (*curr_line)->n_list;
                        compare_distance = n_head->distance;
                  #ifdef DEBUG_NEIGHBOR
                        printf("About to add neighbor: distance is %f\n", distance);
                  #endif
                    if(distance < compare_distance)
                    {
```

```
                        add_to_neighbor_list(list_ptr, i, distance,
                                                    &((*curr_line)->n_list),
                                                    (*curr_line)-
>num_neighbors);
                    }
                        else
                        {
                        #ifdef DEBUG_NEIGHBOR
                          printf("Distance %f too great -- don't add this node\n",
                                                    distance);
                        #endif
                        }
                 }
                }
                list_ptr = list_ptr->next;
                line_count++;
    }
  }
}

#ifdef DEBUG_NEIGHBORS
static void print_nearest_neighbors(struct list_el *line)
{
   struct neighbor *n_ptr;
   struct list_el *line_ref;
   gint count = 0;

   if(!line)
            return;

   if(!(line->n_list))
```

```
            printf("No nearest neighbors for line beginning at (%d, %d) and ending at
(%d, %d)\n", line->begin.row, line->begin.col, line->end.row, line->end.col);
   else
   {
            printf("Nearest neighbors for line beginning at (%d, %d) and ending at (%d,
%d):\n", line->begin.row, line->begin.col, line->end.row, line->end.col);
            n_ptr = line->n_list;
            while(n_ptr)
            {
              line_ref = n_ptr->line;
              printf("%d) Line from (%d, %d) to (%d, %d) at index %d: distance %f\n",
count,
                              line_ref->begin.row, line_ref->begin.col,
                              line_ref->end.row, line_ref->end.col, n_ptr->index,
                              n_ptr->distance);
              n_ptr = n_ptr->next;
              count++;
            }
    }
}
#endif

static gdouble get_slope(gint index)
{
   gdouble slope = 0;
   switch(index)
   {
            case 0: slope = 0;
                        break;
            case 1: slope = 0.5;
                        break;
```

```
                case 2: slope = 1;
                        break;
                case 3: slope = 2;
                        break;
                case 4: slope = 4;
                        break;
                case 6: slope = -4;
                        break;
                case 7: slope = -2;
                        break;
                case 8: slope = -1;
                        break;
                case 9: slope = -0.5;
                        break;
                default: printf("get_slope called with bad index\n");
    }
    return slope;
}

static void
print_line_pts(struct list_el *line_ptr)
{

                printf("Begin: (%d, %d)  End: (%d, %d) Intercept: %f\n",
                        line_ptr->begin.row, line_ptr->begin.col,
                        line_ptr->end.row, line_ptr->end.col,
                        (float)line_ptr->intercept);
}

static void calculate_intercept(struct list_el *line)
{
```

```
    gint x1, x2, y1, y2, index;
    gdouble m, b;
    gboolean have_intercept = FALSE;

    index = -1;

    // Default values to quiet compiler
    m = b = 0;

    x1 = line->begin.col;
    y1 = line->begin.row;
    x2 = line->end.col;
    y2 = line->end.row;
#ifdef DEBUG_INTERCEPT
    printf("Inside calculate_intercept:");
    print_line_pts(line);
#endif

    if(!(x1 - x2))
            index = VERTICAL;

    if(!(y1 - y2))
            index = HORIZONTAL;

    // For vertical and horizontal, store
    // x- and y- intercepts
    if(index != VERTICAL)
            m = (((gdouble)y1 - (gdouble)y2) /
        ((gdouble)x1 - (gdouble)x2));
    else
    {
```

```
            b = x1;
            have_intercept = TRUE;
    }

    if(index == HORIZONTAL)
    {
            b = y1;
            have_intercept = TRUE;
    }

    if(!have_intercept){
      if(x1)
      {
        b = y1/(m * x1);
      }
      else
      {
              b = y2/(m * x2);
            }
    }

    line->intercept = b;

    #ifdef DEBUG_INTERCEPT
      printf("calculated intercept: %f\n", (float)b);
    #endif
}

static void get_intercept(struct list_el *line, gint index)
{
    gint x1, x2, y1, y2;
```

```
    gdouble m, b;
    gboolean have_intercept = FALSE;

    // Default values to quiet compiler
    m = b = 0;

    x1 = line->begin.col;
    y1 = line->begin.row;
    x2 = line->end.col;
    y2 = line->end.row;
#ifdef DEBUG_INTERCEPT
    printf("Inside get_intercept:");
    print_line_pts(line);
#endif

    // For vertical and horizontal, store
    // x- and y- intercepts
    if(index != VERTICAL)
      m = get_slope(index);
    else
    {
            b = x1;
            have_intercept = TRUE;
    }

    if(index == HORIZONTAL)
    {
            b = y1;
            have_intercept = TRUE;
    }
```

```
    if(!have_intercept){
      if(x1)
      {
        b = y1/(m * x1);
      }
      else
      {
                b = y2/(m * x2);
              }
    }

    line->intercept = b;

    #ifdef DEBUG_INTERCEPT
      printf("intercept: %f\n", (float)b);
    #endif
}

static gdouble line_length(gdouble first_x,
                                        gdouble second_x,
                                        gdouble first_y,
                                        gdouble second_y)
{
    // Calculates the Euclidean distance between two points
    gdouble distance;

    #ifdef DEBUG_LENGTH
    printf("Calculating line length: first_x: %f, second_x: %f first_y %f, second_y %f\n",
first_x, second_x, first_y, second_y);
    #endif
    distance = sqrt(((first_x - second_x) * (first_x - second_x))
```

```
                        + ((first_y - second_y) * (first_y - second_y)));

    return distance;

}

static void initialize_histogram(gint histogram[12][36])
{
    gint i, j;

    for(i=0; i<12; i++)
      for(j=0; j<36; j++)
        histogram[i][j] = 0;
}

static void fill_histogram(gint histogram[12][36],
                           struct list_el *line_ptr,
                           gint index)
{
    gdouble b1, b2;   // Offsets for line equations
    gdouble m1, m2;   // Slopes for line equations, based on index of line
    gdouble x, y;     // Coordinates of intersecting point
    gdouble Ax1, Ay1, Ax2, Ay2; // Start and end points of baseline
    gdouble Bx1, By1, Bx2, By2; // Start and end points of neighbor
    gdouble test_x, test_Ay, test_By;  // For CW/CCW determination
    gdouble A, B;    // Magnitude of each vector
    gdouble dA1, dA2, dB1, dB2; // Distances from point of intersection
                                // to endpoints of baseline, neighbor
    gdouble Ax, Ay, Bx, By;     // Magnitudes in each direction for
                                // baseline and neighbor
    gint angle_index, length_index;
```

```
struct neighbor *neighbor_ptr;
struct list_el *neighbor_line;
gdouble gamma;   // Included angle between baseline and neighbor
gdouble cos_num, length_ratio;
gboolean A_begin_high, B_begin_high, A_begin_first, CCW;
gdouble ib;  // Distance from intersection to endpt of baseline
gboolean have_x, have_y;

// Default values to quiet compiler
m1 = m2 = x = y = 0;

have_x = have_y = FALSE;

Ax1 = line_ptr->begin.col;
Ay1 = line_ptr->begin.row;
Ax2 = line_ptr->end.col;
Ay2 = line_ptr->end.row;

// To do: handle the cases (vertical, horizontal) for
// baseline and neighbor

// Calculate baseline slope and offset
if(index != VERTICAL)
  m1 = get_slope(index);
else
{
        x = Ax1;
        have_x = TRUE;
}

if(index == HORIZONTAL)
```

```
{
        y = b1 = Ay1;
        have_y = TRUE;
}

b1 = line_ptr->intercept;

// Calculate magnitude of baseline vector
A = line_length(Ax1, Ax2, Ay1, Ay2);

neighbor_ptr = line_ptr->n_list;
while(neighbor_ptr)
{
        neighbor_line = neighbor_ptr->line;
        Bx1 = neighbor_line->begin.col;
        By1 = neighbor_line->begin.row;
        Bx2 = neighbor_line->end.col;
        By2 = neighbor_line->end.row;

        // Calculate neighbor slope and offset
 if(neighbor_ptr->index != VERTICAL)
   m2 = get_slope(neighbor_ptr->index);
 else
        {
          x = Ax2;
          have_x = TRUE;
        }

 if(neighbor_ptr->index == HORIZONTAL)
        {
          y = b2 = Ay2;
```

```
        have_y = TRUE;
      }

      b2 = neighbor_line->intercept;

      // Calculate magnitude of neighbor vector
      B = line_length(Bx1, Bx2, By1, By2);

      // Find point of intersection of two lines --
      // only if the lines are not parallel!
      if(index != neighbor_ptr->index)
      {
if(!have_x)
        {
          #ifdef DEBUG_LRATIO
             printf("x: (b2(%f) - b1(%f))/(m1(%f) - m2(%f))\n", b2, b1, m1, m2);
          #endif
          x = (b2 - b1)/(m1 - m2);
        }

        if(!have_y)
        {
          printf("y: (m1(%f) * x) + b1(%f))\n", m1, b1);
          y = (m1 * x) + b1;
        }

        // To do:
        // Calculate distances from intersection to end points; mark closest
        // Using closest, calculate magnitudes in x and y directions
        // Using magnitudes, calculate included angle
        // Calculate length ratio using distance from baseline endpoint to
```

```
        // intersection
        // Put entry in histogram

// Calculate distances to end points
        dA1 = line_length(Ax1, x, Ay1, y);
        dA2 = line_length(Ax2, x, Ay2, y);

        if(dA1 < dA2)
        {
                // Begin point of baseline is (Ax1, Ay1)
                Ax = Ax2 - Ax1;
                Ay = Ay2 - Ay1;
                A_begin_first = TRUE;
        }
        else
        {
                // Begin point of baseline is (Ax2, Ay2)
                Ax = Ax1 - Ax2;
                Ay = Ay1 - Ay2;
                A_begin_first = FALSE;
        }
        if (Ay > 0)
          A_begin_high = FALSE;
        else
          A_begin_high = TRUE;

        dB1 = line_length(Bx1, x, By1, y);
        dB2 = line_length(Bx2, x, By2, y);

        if(dB1 < dB2)
        {
```

```
                    // Begin point of neighbor is (Bx1, By1)
                    Bx = Bx2 - Bx1;
                    By = By2 - By1;
                }
                else
                {
                    // Begin point of neighbor is (Bx2, By2)
                    Bx = Bx1 - Bx2;
                    By = By1 - By2;
                }
                if (By > 0)
                  B_begin_high = FALSE;
                else
                  B_begin_high = TRUE;

                // Calculate included angle between baseline and neighbor
                cos_num = ((Ax * Bx) + (Ay * By))/(A * B);
                gamma = acos(cos_num);

              #ifdef DEBUG
                printf("Ax: %f Ay: %f Bx: %f By:%f A: %f B: %f\n",
                        Ax, Ay, Bx, By, A, B);
              #endif
              #ifdef DEBUG_ANGLE
                printf("Included angle: %f = %f from cosine %f\n", gamma, gamma * 180
/ M_PI, cos_num);
              #endif

              // Determine if angle should be positive or negative:
              // is the angle from baseline to neighbor clockwise (+)
              // or counterclockwise (-)?
```

```
// From point of intersection, take one unit in the x direction
// that the neighbor points from the intersection; if the
// corresponding y is below that for the baseline with the same
// x, then clockwise/counterclockwise is determined by:
// clockwise: base has positive slope, begins low, or base
// has negative slope, begins high
// counterclockwise: base has positive slope, begins high, or base
// has negative slope, begins low.
// For values above the baseline y, take the opposite of what
// would happen for the value being below.
// The x direction is one unit in the direction of the sign of Bx.
// Algorithm:
// Calculate neighbor y from given slope, intercept, and x one unit
// in the direction of the sign of Bx from the pt of intersection
// Calculate baseline y for same x
// If neighbor y is below baseline y, use procedure above, doing
// opposite if above baseline y.

  if(Bx < 0)
    test_x = x - 1;
  else
    test_x = x + 1;

  test_Ay = (m1 * test_x) + b1;
  test_By = (m2 * test_x) + b2;

  CCW = FALSE;
  if(A_begin_high)
  {
    if(((m1 >=0) && (test_Ay >= test_By)) ||
       ((m1 < 0) && (test_Ay < test_By)))
```

```
    {
        // counterclockwise
        CCW = TRUE;
    }
  }
  else
  {
    if(((m1 >= 0) && (test_Ay < test_By)) ||
      ((m1 < 0) && (test_Ay >= test_By)))
    {
        CCW = TRUE;
    }
  }

  // If counterclockwise, add pi to gamma
  if (CCW)
  {
    gamma = gamma + M_PI;
    // If adjusted angle is about 2*PI, make it zero
    if((gamma >= ((2 * M_PI) - 0.00001)) &&
      (gamma <= ((2 * M_PI) + 0.00001)))
      gamma = 0;
  #ifdef DEBUG_ANGLE
    printf("Adjusted included angle: %f = %f\n", gamma, gamma * 180 /
M_PI);
  #endif
  }

  angle_index = get_angle_index(gamma);
#ifdef DEBUG_ANGLE
  printf("Calculated angle index is %d for gamma %f\n", angle_index,
```

```
gamma);
#endif

  // Get length ratio

  // Calculate distance from far endpoint of baseline to
  // point of intersection with neighbor

  if(A_begin_first)
      ib = line_length(Ax2, x, Ay2, y);
  else
      ib = line_length(Ax1, x, Ay1, y);

  length_ratio = 1 / (0.5 + (ib / A));

#ifdef DEBUG_LRATIO
  printf("Ax2: %f x: %f Ay2: %f y: %f\n", Ax2, x, Ay2, y);
  printf("ib = %f; A = %f\n", ib, A);
#endif

  length_index = get_length_index(length_ratio);
}
else
{
   // If lines are parallel, length and angle indices are zero
   angle_index = 0;
   length_index = 0;
   length_ratio = -1;
}

#ifdef DEBUG_ANGLE
```

```
            printf("Angle index is %d\n", angle_index);
            #endif
            #ifdef DEBUG_LRATIO
            printf("Length index is %d for length ratio %f\n", length_index,
length_ratio);
            #endif

            histogram[length_index][angle_index]++;

            neighbor_ptr = neighbor_ptr->next;
    }
}

static void get_last_coords(gint *last_row, gint *last_col,
                                        gint *alt_last_row, gint *alt_last_col,
                                        gint *alt_last_row2, gint *alt_last_col2,
                                gint row, gint col, gint index)
{
  // Get the coordinates of the last pixel in the line
  // (before the anchor)
  // Assume last pixel will be coords closest to anchor

  gint i, j, k;
  gint alt_j, alt_k, alt2_j, alt2_k;

  // Give some default values to quiet compiler output
  i = j = k = 0;
  alt_j = alt_k = alt2_j = alt2_k = 0;

  switch(index)
  {
```

```
case 0: j = 2;
                k = 1;
                // Some masks don't have alternate last row
                alt_j = j;
                alt_k = k;
                alt2_j = j;
                alt2_k = k;
                break;
case 1: j = 1;
                k = 3;
                alt_j = j;
                alt_k = 4;
                alt2_j = j;
                alt2_k = 4;
                break;
case 2: j = 1;
                k = 3;
                alt_j = 1;
                alt_k = 3;
                alt2_j = 1;
                alt2_k = 3;
                break;
case 3: j = 1;
                k = 2;
                alt_j = 0;
                alt_k = 3;
                alt2_j = 1;
                alt2_k = 3;
                break;
case 4: j = 0;
                k = 3;
```

```
                alt_j = 1;
                alt_k = k;
                alt2_j = 1;
                alt2_k = 2;
                break;
case 5: j = 1;
                k = 2;
                alt_j = j;
                alt_k = k;
                alt2_j = j;
                alt2_k = k;
                break;
case 6: j = 1;
                k = 1;
                alt_j = j;
                alt_k = 2;
                alt2_j = 0;
                alt2_k = 1;
                break;
case 7: j = 1;
                k = 2;
                alt_j = j;
                alt_k = 1;
                alt2_j = 0;
                alt2_k = 1;
                break;
case 8: j = 1;
                k = 1;
                alt_j = j;
                alt_k = k;
                alt2_j = j;
```

```
                        alt2_k = k;
                        break;
                case 9: j = 1;
                        k = 2;
                        alt_j = 1;
                        alt_k = 0;
                        alt2_j = 1;
                        alt2_k = 1;
        }

        if(i<0)
        {
                printf("Error calculating previous line pixel\n");
                printf("Index = %d, j = %d, k = %d, i = %d\n", index, j, k, i);
        }

        // Need to alter for column processing of line_1

        // Determine row offset from center pixel
        *last_row = row + (j - 2);
        *last_col = col + (k - 2);
        *alt_last_row = row + (alt_j - 2);
        *alt_last_col = col + (alt_k - 2);
        *alt_last_row2 = row + (alt2_j - 2);
        *alt_last_col2 = col + (alt2_k - 2);

}


static void
print_line_list(struct list_el *line_list)
```

```
{
  struct list_el *list_ptr;

  printf("Coordinates shown in (row, col) format\n");
  list_ptr = line_list;

  while(list_ptr != NULL)
  {
            printf("Begin: (%d, %d)  End: (%d, %d)\n",
                        list_ptr->begin.row, list_ptr->begin.col,
                        list_ptr->end.row, list_ptr->end.col);
            list_ptr = list_ptr->next;
  }
}

#ifdef DEBUG_LINES
static void
print_lines(struct list_el *line_list[10])
{
  gint i, count;
  struct list_el *list_ptr;

  printf("Coordinates shown in (row, col) format\n");
  for(i=0; i<10; i++)
  {
    if(!my_config.lines[i])
            continue;
    count = 0;
    list_ptr = (struct list_el *)line_list[i];

    while(list_ptr != NULL)
```

```
    {
            printf("Begin: (%d, %d)  End: (%d, %d)  Midpoint: (%.1f, %.1f)\n\t
Length: %d  Nearest neighbors: %d\n",
                    list_ptr->begin.row, list_ptr->begin.col,
                    list_ptr->end.row, list_ptr->end.col,
                    list_ptr->midpoint.row, list_ptr->midpoint.col,
                    list_ptr->length,
                    list_ptr->num_neighbors);
            count++;
            list_ptr = (struct list_el *)list_ptr->next;
    }
    printf("%d lines stored in index %d\n", count, i);
  }
}
#endif

static gint count_lines_in_list(struct list_el *line_list)
{
  gint count;
  struct list_el *list_ptr;

#ifdef DEBUG
  printf("In count_lines_in_list()\n");
#endif

  count = 0;
  list_ptr = line_list;
  while(list_ptr)
  {
    list_ptr = list_ptr->next;
    count++;
```

```
  }

  return count;
}

static gboolean are_colinear(struct list_el *line1,
                                          struct list_el *line2)
{
  gboolean answer = (abs(line1->intercept - line2->intercept)<2);
//  printf("are_colinear = %d\n", (int)answer);
  return answer;
}

static void grow_lines(struct list_el *line_list, gint index)
{
  gint num_lines, curr_index;
  gint *line_array, lp_index;
  gint max_distance, removed_lines, line_distance;
  struct list_el *line_head, *line_ptr, *remove_ptr, *current_test;

  line_head = line_list;
  line_ptr = line_head;
  current_test = line_head;
  remove_ptr = NULL;
  removed_lines = 0;
  curr_index = lp_index = 0;
  line_distance = 0;
  max_distance = 5;  // Maximum bridging distance across lines

  num_lines = count_lines_in_list(line_head);
  if(num_lines > 0)
```

```
  printf("Number of lines in list: %d\n", num_lines);
else
  return;

print_line_list(line_head);

line_array = (gint *)g_malloc0(sizeof(gint[num_lines]));

while(line_ptr)
{
  current_test = line_ptr->next;
  curr_index = lp_index + 1;

  if(!line_array[lp_index])
  {
            line_array[lp_index] = 1;
    while(current_test)
    {
              if(current_test->begin.row - line_ptr->begin.row > max_distance)
                        break;
              line_distance = line_length(current_test->begin.row,
                                                  line_ptr->end.row,
                                                  current_test->begin.col,
                                                  line_ptr->end.col);
              if((are_colinear(current_test, line_ptr)) &&
                (line_distance < max_distance))
                {
                        printf("Found colinear lines:\n");
                        print_line_pts(line_ptr);
                        print_line_pts(current_test);
                  printf("line distance: %f\n", (gfloat)line_distance);
```

```
                    line_ptr->end.row = current_test->end.row;
                    line_ptr->end.col = current_test->end.col;
                    line_ptr->length = line_length(line_ptr->begin.row,
                                                   line_ptr->end.row,
                                                   line_ptr->begin.col,
                                                   line_ptr->end.col);
                        calculate_intercept(line_ptr);
                printf("Agglomerated line: ");
                        print_line_pts(line_ptr);
                        line_array[curr_index] = 2;
                line_ptr->next = current_test->next;

                        remove_ptr = current_test;
                        remove_ptr->next = NULL;

                        current_test = line_ptr->next;
                curr_index++;


                g_free(remove_ptr);
                        removed_lines++;

              }
            else
            {
              current_test = current_test->next;
              curr_index++;
```

```
                }
            }
        }
        else
        {
                printf("Not entering loop: line_array[lp_index] = %d\n",
                        line_array[lp_index]);
        }
        line_ptr = line_ptr->next;
        lp_index++;
    }

    g_free(line_array);
    printf("Number of lines in list after line growth: %d\n",
          count_lines_in_list(line_head));

    print_line_list(line_head);
}


static void
add_line_to_list(gint row, gint col, struct list_el **line_list, gint index)
{
    gboolean found = FALSE;
    gint last_row, last_col, alt_last_row, alt_last_col;
    gint alt_last_row2, alt_last_col2;
    struct list_el *new_line, *list_ptr, *list_head, *last_ptr;
    struct list_el item;
    static gint line_total;
    gboolean qualified_to_add = FALSE;
    gboolean qualified_to_add_weak = FALSE;
```

```
  list_ptr = new_line = last_ptr = NULL;
#ifdef DEBUG_LINES
  printf("List_ptr points to address %ld\n", (long)list_ptr);
  printf("Line_list points to address %ld\n", (long)*line_list);

  printf("\nIn add_line_to_list(), line index %d\n", index);
#endif

  get_last_coords(&last_row, &last_col, &alt_last_row, &alt_last_col,
                                &alt_last_row2, &alt_last_col2, row, col, index);

#ifdef DEBUG_LINES
  printf("Coordinates to match: last_row: %d, last_col: %d\n", last_row, last_col);
  printf("Alternate coords to match: alt_last_row: %d, alt_last_col: %d\n", alt_last_row,
alt_last_col);
  printf("2nd alternate coords to match: alt_last_row2: %d, alt_last_col2: %d\n",
alt_last_row2, alt_last_col2);
#endif

  list_head = *line_list;
  list_ptr = *line_list;
  if(!*line_list)
            line_total = 0;

  if(!*line_list)
  {
            #ifdef DEBUG_LINES
            printf("line_list is NULL\n");
            #endif
  }
```

```
  while(list_ptr)
  {
    #ifdef DEBUG_LINES
    printf("Current list end: row: %d, col: %d\n", list_ptr->end.row,
                                                                    list_ptr-
>end.col);
    #endif
    qualified_to_add = ((list_ptr->end.row == last_row) &&
                        (list_ptr->end.col == last_col));
    qualified_to_add_weak = ((list_ptr->end.row == alt_last_row) &&
                        (list_ptr->end.col == alt_last_col) &&
                                (list_ptr->weak_pts <= (gint)(list_ptr-
>length)* 2/3)) || ((list_ptr->end.row == alt_last_row2) &&
                        (list_ptr->end.col == alt_last_col2) &&
                                (list_ptr->weak_pts <= (gint)(list_ptr-
>length)* 2/3));

    if(qualified_to_add || qualified_to_add_weak)
    {
            #ifdef DEBUG_LINES
              printf("Qualified to add\n");
              printf("Found matching endpoint; appending to existing line\n");
            #endif
               list_ptr->end.row = row;
               list_ptr->end.col = col;
               list_ptr->length++;
               list_ptr->midpoint.row = ((gfloat)list_ptr->begin.row +
                                                        (gfloat)list_ptr-
>end.row)/2;
               list_ptr->midpoint.col = ((gfloat)list_ptr->begin.col +
```

```
                                                    (gfloat)list_ptr-
>end.col)/2;
                if(qualified_to_add_weak)
                {
                        list_ptr->weak_pts++;
                        #ifdef DEBUG_LINES
                          printf("Adding weak point to line\n");
                        #endif
                }
                found = TRUE;
                break;
    }
    last_ptr = list_ptr;
    list_ptr = (struct list_el *)list_ptr->next;
  }

  // Add new line node to list if not already found
  if(!found)
  {
    #ifdef DEBUG_LINES
    printf("Adding new line\n");
              #endif
    if(list_head)
    {
    #ifdef DEBUG_LINES
              printf("Linking to next ptr\n");
    #endif
      new_line = (struct list_el *)malloc(sizeof(item));
              last_ptr->next = new_line;
              // Make sure we're still hooked to head element;
              *line_list = list_head;
```

```
    }
    else
    {
    #ifdef DEBUG_LINES
                printf("Creating first node\n");
    #endif
      *line_list = (struct list_el *)malloc(sizeof(item));
                if(! (*line_list))
                  printf("Trouble in river city!\n");
                new_line = *line_list;
    }
    #ifdef DEBUG_LINES
    printf("Filling node information\n");
    #endif
    new_line->begin.row = row;
    new_line->begin.col = col;
    new_line->end.row = row;
    new_line->end.col = col;
    new_line->midpoint.row = (gfloat)row;
    new_line->midpoint.col = (gfloat)col;
    new_line->length = 1;
    new_line->num_neighbors = 0;
    new_line->n_list = NULL;
    new_line->next = NULL;
    new_line->num_neighbors = 0;
    new_line->weak_pts = 0;

#ifdef DEBUG_LINES
    printf("New line: end row: %d, end col: %d\n", new_line->end.row,

new_line->end.col);
```

```
    if(*line_list == NULL)
      printf("After line assignment, list pointer is null\n");
    else
      printf("line_list not null after line assignment\n");
#endif

    line_total++;

#ifdef DEBUG_LINES
    printf("Now there are %d lines on index %d\n", line_total, index);
#endif
  }
}


static void
my_get_row (GimpPixelRgn *PR,
            guchar      *dest,
            gint         x,
            gint         y,
            gint         w)
{
 gint width, height, bytes;
 gint i;

 width  = PR->drawable->width;
 height = PR->drawable->height;
 bytes  = PR->drawable->bpp;

 /* Y-wrapping */
```

```
  y = CLAMP (y , 0 , height - 1);

/* X-wrapping */
  if (x < 0)
            {
             gimp_pixel_rgn_get_pixel (PR, dest, 0, y);
             x++;
             w--;
             dest += bytes;

             while (x < 0 && w)
              {
               for (i = 0; i < bytes; i++)
                        {
                         *dest = *(dest - bytes);
                         dest++;
                        }
               x++;
               w--;
              }
            }
  if (w)
            {
             i = MIN (w, width);
             gimp_pixel_rgn_get_row (PR, dest, x, y, i);
             w -= i;
             dest += i * bytes;
            }
  while (w)
            {
             for (i = 0; i < bytes; i++)
```

```
                {
                 *dest= *(dest - bytes);
                 dest++;
                }
               x++;
               w--;
              }
}

static gfloat calculate(guchar **srcrow,
                                        gint xoff,
                                        gint bytes,
                                        gint row,
                                        gint col,
                                        const gint line_mx[5][5])
{
              gfloat temp;
              gfloat sum = 0;
              gint x, y;

//            for(y = 0; y < 5; y++)
//              for(x = 0; x<5; x++)
//              {
//                      printf("line[y(%d)][x(%d)] = %d\n", y, x, line_mx[y][x]);
//              }

              for(y = 0; y < 5; y++)
                for(x = 0; x<5; x++)
                {
                  temp = line_mx[y][x];
```

```
                temp *= srcrow[y][xoff + x * bytes];
                if(temp != 0)
                {
                  //printf("Value of temp toward sum: %f\n", temp);
                }
                sum += temp;
                //printf("Sum at step %d: %f\n", (5*y)+x+1, sum);
              }

            // Eliminate pixels where a single pixel of support exists
            // within the mask
            sum = sum / 3;
            if (sum < 255)
              sum = 0;

#ifdef DEBUG
            if(sum != 0)
            {
                    show_location(row, col);
                    show_line_matrix(line_mx);
                    show_source_array(srcrow, xoff, bytes);
                    printf("Value to return: %f\n\n", sum);
            }
#endif
            return sum;
}

#ifdef DENOISE
static void
remove_noise(struct list_el **line_list, gint index, gint min_line_length)
{
```

```
struct list_el *remove_ptr, *list_ptr, *list_head, *last_ptr;

list_ptr = list_head = last_ptr = NULL;
remove_ptr = NULL;

#ifdef DEBUG
  printf("In remove_noise\n");
#endif

if(!my_config.lines[index])
 return;

list_head = *line_list;
list_ptr = *line_list;
if(!*line_list)
            return;

while(list_ptr)
{
  if(list_ptr->length < min_line_length)
  {
   remove_ptr = list_ptr;
            if(remove_ptr == list_head)
            {
              list_head = list_ptr->next;
              *line_list = list_head;
            }
            else
      last_ptr->next = list_ptr->next;
   list_ptr = (struct list_el *)list_ptr->next;
            free(remove_ptr);
```

```
    }
    else
    {
            last_ptr = list_ptr;
            list_ptr = list_ptr->next;
    }
  }
}
#endif

static gfloat
calcmatrix (guchar **srcrow,
            gint    xoff,
            gint    bytes,
            gint    row,
            gint    col,
            struct list_el *line_list[10])
{
 gint   k;
 gfloat sum     = 0;
 gfloat prev_sum = 0;

 for(k = 0; k<10; k++)
 {
  if(my_config.lines[k] != 0)
  {
   switch(k)
   {
            case 0:
                    sum = calculate(srcrow, xoff, bytes, row, col, line_0);
            //      printf("Calculating line 0: %f\n", sum);
```

```
                break;
case 1:
                sum = calculate(srcrow, xoff, bytes, row, col, line_1);
//              printf("Calculating line 1: %f\n", sum);
                break;
case 2:
                sum = calculate(srcrow, xoff, bytes, row, col, line_2);
//              printf("Calculating line 2: %f\n", sum);
                break;
case 3:
                sum = calculate(srcrow, xoff, bytes, row, col, line_3);
//              printf("Calculating line 3: %f\n", sum);
                break;
case 4:
                sum = calculate(srcrow, xoff, bytes, row, col, line_4);
//              printf("Calculating line 4: %f\n", sum);
                break;
case 5:
                sum = calculate(srcrow, xoff, bytes, row, col, line_5);
//              printf("Calculating line 5: %f\n", sum);
                break;
case 6:
                sum = calculate(srcrow, xoff, bytes, row, col, line_6);
//              printf("Calculating line 6: %f\n", sum);
                break;
case 7:
                sum = calculate(srcrow, xoff, bytes, row, col, line_7);
//              printf("Calculating line 7: %f\n", sum);
                break;
case 8:
                sum = calculate(srcrow, xoff, bytes, row, col, line_8);
```

```
                //              printf("Calculating line 8: %f\n", sum);
                                break;
                case 9:
                                sum = calculate(srcrow, xoff, bytes, row, col, line_9);
                //              printf("Calculating line 9: %f\n", sum);
                                break;

                }

    if(sum)
    {
                add_line_to_list(row, col, &(line_list[k]), k);
    }

    sum = MAX(sum, prev_sum);
    prev_sum = sum;
   }
  }

 return sum;
}

static void
check_db_error(struct sqlca sqlerrstruct)
{
  if(!sqlerrstruct.sqlcode)
    printf("No error detected\n");
  else if(sqlerrstruct.sqlcode == 100)
    printf("End of cursor/no data found\n");
  else
  {
```

```
    printf("SQL error number %d:\n", (int)sqlerrstruct.sqlcode);
    printf("Line %d: Error: %s\n", (int)sqlerrstruct.sqlerrm.sqlerrml,
                                                    sqlerrstruct.sqlerrm.sqlerrmc);
  }
}

static void
doit (void)
{
 GimpPixelRgn  srcPR, destPR;
 gint        width, height, row, col;
 gint        w, h, i, j, bytes;
 guchar      *destrow[3];
 guchar      *srcrow[5];
 guchar      *temprow;
 gfloat      sum;
 gint        xoff;
 gint                        min_line_length;
 struct list_el *lines[10], *line_ptr, *line_head;
 gint                        histogram[12][36];
 gchar                       hist_name[100] = {0};
 gchar                       *hist_path = "/tmp/histograms/";
 FILE                        *hist_file;

 exec sql begin declare section;

 char img_name[100];
 int result;

 exec sql end declare section;
```

```
strcpy(img_name, gimp_image_get_filename(image));
//strcpy(img_name, "simple");
printf("The image name is %s\n", img_name);

exec sql include sqlca;

exec sql connect to glyph;

check_db_error(sqlca);

min_line_length = 3;

for(i=0; i<10; i++)
  lines[i] = NULL;

/* Get the input area. This is the bounding box of the selection in
 *  the image (or the entire image if there is no selection). Only
 *  operating on the input area is simply an optimization. It doesn't
 *  need to be done for correct operation. (It simply makes it go
 *  faster, since fewer pixels need to be operated on).
 */
gimp_drawable_mask_bounds (drawable->id, &sx1, &sy1, &sx2, &sy2);
w = sx2 - sx1;
h = sy2 - sy1;

/* Get the size of the input image. (This will/must be the same
 *  as the size of the output image.
 */
width  = drawable->width;
height = drawable->height;
bytes  = drawable->bpp;
```

```
for (i = 0; i < 5; i++)
  srcrow[i] = g_new (guchar, (w + 4) * bytes);
for (i = 0; i < 3; i++)
  destrow[i]= g_new (guchar, w * bytes);

/*  initialize the pixel regions  */
gimp_pixel_rgn_init (&srcPR, drawable,
                                    sx1 - 2, sy1 - 2, w + 4, h + 4, FALSE, FALSE);
gimp_pixel_rgn_init (&destPR, drawable, sx1, sy1, w, h, TRUE, TRUE);

/* initialize source arrays */
for (i = 0; i < 5; i++)
  my_get_row (&srcPR, srcrow[i], sx1 - 2, sy1 + i - 2, w + 4);

for (row = sy1; row < sy2; row++)
  {
    xoff = 0;

    for (col = sx1; col < sx2; col++)
    {
                {
                for (i = 0; i < bytes; i++)
                  {
                    if(gimp_drawable_has_alpha(drawable->id))
                    {
                      if(i == bytes - 1)
                                  sum = srcrow[2][xoff + 2 * bytes];
                      else
                      {
                        sum = calcmatrix(srcrow, xoff, i, row, col, lines);
```

```
                }
              }
              else
              {
                sum = calcmatrix(srcrow, xoff, bytes, row, col, lines);
              }

              destrow[2][xoff]= (guchar) CLAMP (sum, 0, 255);
              xoff++;
            }
          }
  }

  if (row > sy1 + 1)
          gimp_pixel_rgn_set_row (&destPR, destrow[0], sx1, row - 2, w);

  temprow = destrow[0];
  destrow[0] = destrow[1];
  destrow[1] = destrow[2];
  destrow[2] = temprow;
  temprow = srcrow[0];

  for (i = 0; i < 4; i++)
          srcrow[i] = srcrow[i + 1];

  srcrow[4] = temprow;
  my_get_row (&srcPR, srcrow[4], sx1 - 2, row + 3, w + 4);

  gimp_progress_update ((double) (row - sy1) / h);
}
```

```
  if (h > 2)
    gimp_pixel_rgn_set_row (&destPR, destrow[0], sx1, row - 2, w);
  if (h > 1)
    gimp_pixel_rgn_set_row (&destPR, destrow[1], sx1, row - 1, w);


// print_lines(lines);
// for(i=0; i<10; i++)
//    printf("Number of lines in index %d: %d\n", i, count_lines_in_list(lines[i]));
#ifdef DENOISE
  for(i=0; i<10; i++)
    remove_noise(&(lines[i]), i, min_line_length);
#endif
  #ifdef DEBUG_LINES
  print_lines(lines);
  #endif
  for(i=0; i<10; i++)
  {
    if(count_lines_in_list(lines[i]) > 0)
      printf("Number of lines in index %d: %d\n", i, count_lines_in_list(lines[i]));
  }

  initialize_histogram(histogram);

  for(i=0; i<10; i++)
  {
    line_head = lines[i];
    line_ptr = lines[i];
    printf("Processing line index %d\n", i);
    while(line_ptr)
```

```
  {
    get_intercept(line_ptr, i);
            line_ptr = line_ptr->next;
  }
  for(j=0; j<5; j++)
    grow_lines(lines[i], i);
  line_ptr = line_head;
  while(line_ptr)
  {
            find_nearest_neighbors(i, &line_ptr, lines);
            line_ptr = line_ptr->next;
  }
  line_ptr = line_head;
  while(line_ptr)
  {
            #ifdef DEBUG_NEIGHBORS
            print_nearest_neighbors(line_ptr);
            #endif
            fill_histogram(histogram, line_ptr, i);
            line_ptr = line_ptr->next;
  }
}


exec sql select count(*) into :result from images where image_name = :img_name;

check_db_error(sqlca);

if(result)
 printf("Found a record");
else
```

```
{
 printf("No record exists\n");
 exec sql insert into images values(nextval('"images_image_num_seq"'),:img_name);
 check_db_error(sqlca);
 exec sql commit;
 check_db_error(sqlca);
}

exec sql select image_num into :result from images where image_name = :img_name;

check_db_error(sqlca);

printf("Image has image_num %d\n", result);
sprintf(hist_name, "%shist_%d", hist_path,result);

printf("Histogram file name: %s\n", hist_name);

hist_file = fopen(hist_name, "w");

if(! hist_file)
 printf("Histogram file failed to open\n");

for(j=0; j<36; j++)
{
  for(i=0; i<12; i++)
  {
    if(hist_file){
      if(i < 11)
        fprintf(hist_file, "%d,", histogram[i][j]);
      else
        fprintf(hist_file, "%d", histogram[i][j]);
```

```
            }

            if(histogram[i][j] > 0)
              printf("Histogram[%d][%d] contains [%d]\n", i, j, histogram[i][j]);
    }
    if(hist_file)
      fprintf(hist_file, "\n");
  }

  if(hist_file)
  {
    fprintf(hist_file, "%s", img_name);
    fclose(hist_file);
  }

  exec sql disconnect all;

  check_db_error(sqlca);


  /*  update the timred region  */
  gimp_drawable_flush (drawable);
  gimp_drawable_merge_shadow (drawable->id, TRUE);
  gimp_drawable_update (drawable->id, sx1, sy1, sx2 - sx1, sy2 - sy1);
}

/*****************************************************
 * GUI stuff
 */

static void
```

```
redraw_lines (void)
{
 gint i;

 for (i = 0; i < 10; i++)
   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (my_widgets.lines[i]),
                                                    my_config.lines[i] > 0);
}

static void
redraw_all (void)
{
 redraw_lines();
}

static void
ok_callback (GtkWidget *widget,
             gpointer   data)
{
 run_flag = TRUE;

 gtk_widget_destroy (GTK_WIDGET (data));
}

/* Checks that the configuration is valid for the image type */
static void
check_config (void)
{
 gint i;

 for (i = 0; i < 10; i++)
```

```
   if (my_config.lines[i] < 0)
     my_config.lines[i] = 0;
}

static void
defaults_callback (GtkWidget *widget,
                                gpointer   data)
{
 my_config = default_config;
 check_config ();
 redraw_all ();
}


static void
my_toggle_callback (GtkWidget *widget,
                                gpointer   data)
{
 gint val = GTK_TOGGLE_BUTTON (widget)->active;

 if (val)
   *(gint *) data = TRUE;
 else
   *(gint *) data = FALSE;

}

static gint
dialog (void)
{
 GtkWidget *dlg;
```

```
  GtkWidget *main_vbox;
  GtkWidget *button;
  GtkWidget *box;
  GtkWidget *frame;
  gint      i;

  gimp_ui_init ("line_detect", FALSE);

  dlg = gimp_dialog_new (_("Line Detection"), "line_detect",
                                        gimp_standard_help_func,
"filters/line_detect.html",
                                        GTK_WIN_POS_MOUSE,
                                        FALSE, TRUE, FALSE,

                                        _("OK"), ok_callback,
                                        NULL, NULL, &my_widgets.ok, TRUE,
FALSE,
                                        _("Reset"), defaults_callback,
                                        NULL, 1, NULL, FALSE, FALSE,
                                        _("Cancel"), gtk_widget_destroy,
                                        NULL, 1, NULL, FALSE, TRUE,

                                        NULL);

  gtk_signal_connect (GTK_OBJECT (dlg), "destroy",
                                GTK_SIGNAL_FUNC (gtk_main_quit),
                                NULL);

  main_vbox = gtk_vbox_new (FALSE, 4);
  gtk_container_set_border_width (GTK_CONTAINER (main_vbox), 6);
  gtk_box_pack_start (GTK_BOX (GTK_DIALOG (dlg)->vbox), main_vbox,
```

```
                                TRUE, TRUE, 0);

frame = gtk_frame_new (_("Line to Detect"));
gtk_frame_set_shadow_type (GTK_FRAME (frame), GTK_SHADOW_ETCHED_IN);
gtk_box_pack_start (GTK_BOX (main_vbox), frame, FALSE, FALSE, 0);

gtk_widget_show (frame);

box = gtk_vbox_new (FALSE, 1);
gtk_container_set_border_width (GTK_CONTAINER (box), 2);
gtk_container_add (GTK_CONTAINER (frame), box);

for (i = 0; i < 10; i++)
 {
  my_widgets.lines[i] = button =
            gtk_check_button_new_with_label (gettext (line_labels[i]));
  if (my_config.lines[i] < 0)
            gtk_widget_set_sensitive (button, FALSE);
  gtk_signal_connect (GTK_OBJECT (button), "toggled",
                                        (GtkSignalFunc)my_toggle_callback,
                                        &my_config.lines[i]);
  gtk_box_pack_start (GTK_BOX (box), button, FALSE, FALSE, 0);
  gtk_widget_show (button);
 }

gtk_widget_show(box);
gtk_widget_show (frame);

gtk_widget_show (main_vbox);

gtk_widget_show (dlg);
```

```
  redraw_all ();

  gtk_main ();
  gdk_flush ();

  return run_flag;
}
```

```
/* Selection-to-Mask plug-in for the GIMP -- Version 0.1
 * Copyright (C) 2001 Cindy Huyser <chuyser@io.com>
 *
 * Based in part on:
 * Convolution Matrix plug-in for the GIMP -- Version 0.13
 * Copyright (C) 1997 Lauri Alanko <la@iki.fi>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 *
 * The GNU General Public License is also available from
 * http://www.fsf.org/copyleft/gpl.html
 *
 * CHANGELOG:
 *
 * v0.1         03.06.2002
 *              Initial release.
 *
 *
 */
```

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <sys/types.h>
#ifdef HAVE_UNISTD_H
#include <unistd.h>
#endif

#include <gtk/gtk.h>

#include <libgimp/gimp.h>
#include <libgimp/gimpui.h>

#include <libgimp/stdplugins-intl.h>

//#define DEBUG

GimpDrawable *drawable;
gint32 image;


/* Declare local functions. */
static void query (void);
static void run   (gchar      *name,
                   gint        nparams,
                   GimpParam  *param,
                   gint       *nreturn_vals,
                   GimpParam **return_vals);
```

```
static void make_mask        (void);

GimpPlugInInfo PLUG_IN_INFO =
{
 NULL,   /* init_proc  */
 NULL,   /* quit_proc  */
 query,  /* query_proc */
 run,    /* run_proc   */
};

gint bytes, bg_bytes;
gint sx1, sy1, sx2, sy2;
gint run_flag = 0;


MAIN ()

static void
query (void)
{
 static GimpParamDef args[] =
 {
  { GIMP_PDB_INT32,    "run_mode", "Interactive, non-interactive" },
  { GIMP_PDB_IMAGE,    "image",    "Input image" },
  { GIMP_PDB_DRAWABLE, "drawable", "Input drawable [unused]" },
 };
 static gint nargs = sizeof (args) / sizeof (args[0]);

 gimp_install_procedure ("plug_in_selection-to-mask",
                                          "Selection to Mask",
```

```
                                    "",
                                    "Cindy Huyser",
                                    "Cindy Huyser",
                                    "2002",
                                    N_("<Image>/Selection to Mask"),
                                    "GRAY*, RGB*",
                                    GIMP_PLUGIN,
                                    nargs, 0,
                                    args, NULL);
}

static void
run (gchar      *name,
     gint        nparams,
     GimpParam  *param,
     gint       *nreturn_vals,
     GimpParam **return_vals)
{
  static GimpParam  values[1];
  GimpRunModeType   run_mode;
  GimpPDBStatusType status = GIMP_PDB_SUCCESS;
  gint32 activeLayer;
  GimpImageType imgType;

  INIT_I18N_UI();

  (void) name; /* Shut up warnings about unused parameters. */

  *nreturn_vals = 1;
  *return_vals = values;
```

```
  run_mode = param[0].data.d_int32;

  /*  Get the specified drawable  */
  image = param[1].data.d_image;
  activeLayer = gimp_image_get_active_layer(image);
  drawable = gimp_drawable_get (activeLayer);

  imgType = gimp_drawable_type(drawable->id);

  /* Won't work on indexed images */
  if((imgType == GIMP_INDEXED_IMAGE)||(imgType ==
GIMP_INDEXEDA_IMAGE))
    status = GIMP_PDB_CALLING_ERROR;

  if (nparams != 3)
    status = GIMP_PDB_CALLING_ERROR;

  if (status == GIMP_PDB_SUCCESS)
   {
                gimp_progress_init (_("Making mask from selection"));

                make_mask ();

                if (run_mode != GIMP_RUN_NONINTERACTIVE)
                  gimp_displays_flush ();

    gimp_drawable_detach (drawable);
   }

  values[0].type = GIMP_PDB_STATUS;
  values[0].data.d_status = status;
```

```
}

static void
make_mask (void)
{
 gint32        activeLayer, workingLayer;
 gboolean      add_OK, add_alpha_OK, lower_layer_OK, copy_OK;
 gboolean      anchorOK, add_mask_OK,deleteOK;
 GimpDrawable  *wk_drawable;
 gint32        mask_copy_channel, floating_sel, newMask, wkImage;

 /* Steps for manually converting a selection to a layer mask:

              1) Save the selection to a channel
              2) Duplicate original layer
              3) Add alpha to original and duplicate layers
              4) Add layer mask (transparent/black) to duplicate layer
              5) Move duplicate layer down
              6) Make selection mask copy channel active
              7) Copy mask copy channel
              8) Make mask of duplicate layer active
              9) Paste copy of mask copy channel on duplicate layer
              10) Anchor layer
              11) Make selection mask copy channel active
              12) Delete selection mask copy
 */

 // 1

 mask_copy_channel = gimp_selection_save(image);
```

```
// 2

activeLayer = gimp_image_get_active_layer(image);
workingLayer = gimp_layer_copy(activeLayer);
add_OK = gimp_image_add_layer(image, workingLayer, -1);
wkImage = gimp_drawable_image(workingLayer);

if(!add_OK)
        printf("Failure occurred while adding layer copy\n");
#ifdef DEBUG
else
        printf("Copy of initial layer added OK\n");
#endif


// 3

add_alpha_OK = gimp_layer_add_alpha(activeLayer);
if(!add_alpha_OK)
        printf("Failure occurred while adding alpha to active layer\n");
#ifdef DEBUG
else
        printf("Alpha layer added OK to active layer\n");
#endif

add_alpha_OK = gimp_layer_add_alpha(workingLayer);
if(!add_alpha_OK)
        printf("Failure occurred while adding alpha to working layer\n");
#ifdef DEBUG
else
```

```
        printf("Alpha layer added OK to working layer\n");
#endif


// 4

newMask = gimp_layer_create_mask(workingLayer, GIMP_BLACK_MASK);
add_mask_OK = gimp_image_add_layer_mask(wkImage, workingLayer, newMask);
if(!add_mask_OK)
        printf("Failure occurred while adding alpha to working layer\n");
#ifdef DEBUG
else
        printf("Mask layer added OK\n");
#endif

wk_drawable = gimp_drawable_get(workingLayer);
gimp_drawable_flush (wk_drawable);
gimp_drawable_update (wk_drawable->id, 0, 0, wk_drawable->width,
                                        wk_drawable->height);


// 5

lower_layer_OK = gimp_image_lower_layer(wkImage, workingLayer);
if(!lower_layer_OK)
        printf("Failure occurred while lowering working layer\n");
#ifdef DEBUG
else
        printf("Working layer lowered OK; value of lower_layer_OK: %d\n", \
                        lower_layer_OK);
#endif
```

```
// 6 & 7

copy_OK = gimp_edit_copy(mask_copy_channel);
if(!copy_OK)
        printf("Failure occurred while copying channel\n");
#ifdef DEBUG
else
        printf("Edit mask channel copied OK\n");
#endif


// 8 & 9

wk_drawable = gimp_drawable_get(gimp_layer_mask(workingLayer));
floating_sel = gimp_edit_paste(wk_drawable->id, FALSE);


// 10

anchorOK = gimp_floating_sel_anchor(floating_sel);
if(!anchorOK)
        printf("Failure occurred anchoring pasted mask to layer\n");
#ifdef DEBUG
else
        printf("Pasted layer anchored OK\n");
#endif

// 11 & 12
```

```
deleteOK = gimp_image_remove_channel(image, mask_copy_channel);
if(!deleteOK)
        printf("Failure occurred deleting mask copy channel\n");
#ifdef DEBUG
else
        printf("Mask copy channel deleted OK\n");
#endif

gimp_drawable_flush (wk_drawable);
gimp_drawable_update (wk_drawable->id, 0, 0, wk_drawable->width,
                                        wk_drawable->height);

}
```

```
/* cluster_analysis.pgc */

/* Driver file for cluster analysis of centroid histograms
  in the glyph database*/

/* This version prints only file names */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
//#include <sqlca.h>
#include "b_dist3.h"

#define NUM_ANGLES 64
//#define IMAGE_LIMIT 135
#define CLUSTER_LIMIT 1
#define DISTANCE_THRESHOLD 100  // Display of cluster threshold

//#define PRINT_LONG
//#define DEBUG_UPDATE
//#define DEBUG_VERBOSE
//#define DEBUG

struct img_info{
   float distance;
   int img_num;
   int index;
};

struct distribution_info{
   char name[20];
```

```
    int retrievals;
};

struct retrieval_info{
    float sum;
    int best;
    char name[20];
    int type_num;
};

struct cluster_info{
    int img_num;
    float distance;
    int *points_to;
    int pointed_to_by;
    int num_points_to;
    struct cluster_info *next;
};

struct cluster_node{
    char *distance_file_name;
    float distance;
    int level;
    int img_num;
    int index;
    int num_merged;
    struct cluster_node* merged_node_list;
    struct cluster_node* next;
    struct cluster_node* previous;
};
```

```
int check_db_error(struct sqlca sqlerrstruct);
void check_error(int err_num, char *msg);
void connect_to_db(struct sqlca sqlerrstruct);
void disconnect_from_db(struct sqlca sqlerrstruct);
int comp(const void *a, const void *b);
int get_num_images(struct sqlca sqlerrstruct);
void get_short_name(char *long_name, char* short_name);
void get_jpg_name(char *short_name, char* jpg_name);
void show_retrieval_stats(struct retrieval_info *retrieval_sums,
                          int *img_type_counts, int num_img_types,
                          int num_images);
void print_stats();
void print_header(char *jpg_name, int image_number);
void index_header();
void finish_index();
void print_retrieval(int query_image, char *jpg_name,
                     float distance, int ret_num);

void initialize_active_list(struct cluster_node **active_list,
                            int *image_nums,
                            int num_images);

void initialize_cluster_list(struct cluster_info **cluster_list,
                             int *image_nums,
                             int num_images);

int get_smallest_retrieval_list(struct img_info *img_list, int num_images,
                                struct cluster_node **best_list, int *image_nums,
                                int *subsumed, int *index_list,
                                struct cluster_node *inactive_list);
```

```
int mark_subsumed(int img_num, int *index_list, int *subsumed);

int has_been_subsumed(int subsumed_marker);

void update_subsumed_num(int img_num, int sum_subsuming,
                         int *index_list, int *subsumed);

void add_to_list(struct cluster_node **list,
                 struct cluster_node *new_cluster_ptr);

void move_to_inactive_list(struct cluster_node **active_list,
                           struct cluster_node **inactive_list,
                           struct cluster_node *merged_ptr);

int count_nodes_on_list(struct cluster_node *list);

void show_nodes(struct cluster_node *list, int *subsumed, int *index_list);

void update_distance_matrix(struct cluster_node *new_active_cluster,
                            struct cluster_node *active_list,
                            struct cluster_node *inactive_list,
                            struct img_info *img_list,
                            int *index_list, int num_images, int
*subsumed);

void show_distance_matrix(struct img_info *img_list, int num_images);

void show_relevant_distance_matrix(struct img_info *img_list, int num_images,
                                   struct cluster_node *inactive_list);

int is_inactive(int img_num, struct cluster_node *inactive_list);
```

```
void add_to_new_cluster_list(struct cluster_node *new_cluster,
                             struct cluster_node *best_list,
                             int *index_list, struct cluster_info **cluster_list,
                             int num_images);

void display_new_cluster_list(struct cluster_info **cluster_list);

void display_current_cluster(struct cluster_info **cluster_list,
                             int cluster_num, char *spacing);

void display_by_distance_new_cluster_list(struct cluster_info **cluster_list);

void display_by_distance_current_cluster(struct cluster_info **cluster_list,
                             int cluster_num);

void print_by_distance_new_cluster_list(struct cluster_info **cluster_list,
                                        char *filename);

void print_by_distance_current_cluster(struct cluster_info **cluster_list,
                             int cluster_num, char *filename);

void show_cluster_list_stats(struct cluster_info **cluster_list);

void initialize_unavailable(int *unavailable, int *image_nums, int num_images);

int available(int img_num, int *unavailable);

void print_html_current_cluster(struct cluster_info **cluster_list, int cluster_num, char
*filename);
```

```
void print_html_cluster_list(struct cluster_info **cluster_list, char *filename);

void print_cluster_structures(struct cluster_info **cluster_list, int threshold_num, char
*filename);

static int print_count = 0;
static int num_to_print = 0;
static int display_count = 0;

int main()
{
   int i, j, err_num, index1, threshold, num_best, prev_subsumed;
   int last_index, search_depth, *merged_clusters, *unavailable;
   int num_images, depth_of_search, index_list[1000], potential_img_num;
   int *subsumed, usable_best_num, usable_img_num, aligned_index, first_threshold;
   float distances_1[NUM_ANGLES], distances_2[NUM_ANGLES];
   float distance_sum[NUM_ANGLES];
   int empty_1[NUM_ANGLES], empty_2[NUM_ANGLES],
empty_sum[NUM_ANGLES];
   int node_factor, sum_subsuming, highest_subsumed_level, num_inactive_nodes;
   int count_merged;
   int loop_count = 0;
   int cluster_count = 0;
   char file1[30], file2[30], file_new[30], cluster_file[50], cluster_html_file[60];
   char base_name[20] = {0};
   struct img_info *img_list;
   char short_name[20] = {0};
   char jpg_name[20] = {0};
   char buf[100] = {0};
   char distance_buf[50] = {0};
   int items_read = 0;
```

```
    FILE *distance_matrix, *dist1, *dist2, *dist_new;
    char *matrix_file = "/tmp/distance_matrix";
    struct cluster_node *active_list, *inactive_list, *best_list;
    struct cluster_node *new_active_node;
    struct cluster_node *cluster_ptr, *active_ptr, *remove_ptr, *best_ptr;
    struct cluster_node *remove_merged, *curr_merged, *merged_ptr, *best_head;
    struct cluster_info *cluster_list, *list_head, *list_ptr, *remove_list_ptr;

    exec sql begin declare section;
    int image_nums[500];
    char long_name[100];
    int max_img_index;
    exec sql end declare section;

    exec sql include sqlca;

    connect_to_db(sqlca);
    num_images = get_num_images(sqlca);

    threshold = DISTANCE_THRESHOLD;  // Distance at which we'll display clusters
    first_threshold = DISTANCE_THRESHOLD/10;  // Distance at which we'll display
clusters

    printf("Found %d images in the images table\n", num_images);

    // FOR TESTING -- a small subset
    #ifdef IMAGE_LIMIT
    num_images = IMAGE_LIMIT;
    #endif
    cluster_count = num_images;
```

```
exec sql select image_num into :image_nums from images;

err_num = check_db_error(sqlca);
check_error(err_num, "Getting image numbers");

// Prepare for some statistics:
// How many images of each type in the db?
// How many images total?
// What's the max image index?

exec sql select MAX(image_num) into :max_img_index from images;
err_num = check_db_error(sqlca);
check_error(err_num, "Getting highest image index");

active_list = inactive_list = best_list = curr_merged = merged_ptr = NULL;
active_ptr = cluster_ptr = remove_ptr = best_ptr = remove_merged = NULL;

initialize_active_list(&active_list, image_nums, num_images);
initialize_cluster_list(&cluster_list, image_nums, num_images);

printf("Number to print: %d\n", num_to_print);

if(active_list == NULL)
        printf("Returning with null active list\n");

if(cluster_list == NULL)
   printf("Returning with null cluster list\n");

// Memory allocation section
img_list = malloc(sizeof(struct img_info[num_images * num_images]));
merged_clusters = malloc(sizeof(int[num_images]));
```

```
subsumed = malloc(sizeof(int[num_images]));
unavailable = malloc(sizeof(int[num_images * 4]));
initialize_unavailable(unavailable, image_nums, num_images);

for(i=0; i<1000; i++)
{
        index_list[i] = -1;
}

for(i=0; i<num_images; i++)
{
        // Mark all clusters not merged
        merged_clusters[i] = 0;
        subsumed[i] = 0;

        // Provide a lookup from image num to index in num_images
        index_list[image_nums[i]] = i;
}

distance_matrix = fopen(matrix_file, "r");
if(!distance_matrix)
{
        printf("Failed to open distance matrix file %s\n", matrix_file);
        return 1;
}

// Populate matrix of distances
index1 = 0;
printf("Populating distance matrices\n");
while(!feof(distance_matrix))
{
```

```
            fgets(buf, 100, distance_matrix);
            if(!strcmp(buf, "\n"))
              continue;
            items_read = sscanf(buf, "%d%*c%f%*c%d",&img_list[index1].img_num,
                                        &img_list[index1].distance,
&img_list[index1].index);
            index1++;
   }

   if(distance_matrix)
            fclose(distance_matrix);

   num_inactive_nodes = count_nodes_on_list(inactive_list);
  //  printf("Number of nodes on inactive list: %d\n", num_inactive_nodes);

#ifdef DEBUG
   printf("Initial distance matrix:\n");
   //show_relevant_distance_matrix(img_list, num_images, inactive_list);
   show_distance_matrix(img_list, num_images);
#endif

///////// Begin clustering loop ///

  while(cluster_count > CLUSTER_LIMIT)
  {
#ifdef DEBUG
   printf("\n**********************\n");
   printf("Loop %d: cluster_count %d\n", loop_count, cluster_count);
   printf("\n**********************\n");
#endif
```

```
   loop_count++;

   last_index = 0;

   num_best = get_smallest_retrieval_list(img_list, num_images, &best_list,
                                                    image_nums, subsumed, index_list,
inactive_list);

#ifdef DEBUG
   printf("Number of clusters created: %d\n", num_best);
#endif

   if(!best_list)
   {
            printf("Quitting loop: best list is null\n");
            break;
   }
#ifdef DEBUG
   else
            printf("Best list is not null\n");
#endif


   best_head = best_list;
   best_ptr = best_head;

   index_header();
   while(best_ptr)
   {
            // Merging steps:
            // Make a new image number = image_number + 1000 * (max level + 1)
```

```
            // Merge the distance files:
            //  Align the distance files at the indices of minimum distance
            //  Use the mean of existing distances for the new distance
            //  Mark the merged files as out of play
            //  Calculate the distance between the new merged file and the
            //  remaining distances
            //  Replace entries in the distance table with the merged entries

            // There are two methods to consider:
            // a centroid method (merging the two distance files)
            // and unweighted pair-groups method average

            // ** Centroid method: create a new distance file from the
            //   exising distance files, and replace the distances in the
            //   primary merge node with new distances.  Must keep track
            //   of number so far subsumed.

            // ** Unweighted pair-groups method average: calculate average
            //   distances between each point in a cluster and all other points
            //   in another cluster.  No new files created, but new data structure
            //   must be used to hold averages for use

            // This implementation uses the centroid method

            sum_subsuming = 0;
            highest_subsumed_level = 0;
            count_merged = 0;
#ifdef DEBUG
            printf("image number %d to be combined with %d clusters\n", best_ptr-
>img_num, best_ptr->num_merged);
#endif
```

```
            // We will need to agglomerate distances for the creation of the new
            // file. Read the distances from the files.

            sprintf(file1, "/tmp/distances/dist_%d", best_ptr->img_num);
    dist1 = fopen(file1, "r");
    if(!dist1)
    {
              printf("Failed to open distance file %s\n", file1);
              return 1;
    }
            usable_best_num = (best_ptr->img_num)%1000;
            node_factor = 1 + subsumed[index_list[usable_best_num]];
            i = 0;
#ifdef DEBUG
            printf("Reading distances for main node; factor is %d\n", node_factor);
#endif
            while((!feof(dist1)) && (i < NUM_ANGLES))
            {
              fgets(buf, 100, dist1);
              if(feof(dist1))
                 break;
              items_read = sscanf(buf, "%d%*c%f",&empty_1[i], &distances_1[i]);
              empty_sum[i] = empty_1[i];
              if(!empty_1[i])
              {
                distance_sum[i] = (float)node_factor * distances_1[i];

            #ifdef DEBUG
                printf("Reading distance_1[%d] as %f\n", i,(float)distances_1[i]);
                printf("Setting distance_sum[%d] to %f\n", i, (float)distance_sum[i]);
```

```
#endif
  }
  i++;
}

if(dist1)
  fclose(dist1);

if(best_ptr->img_num/1000 > highest_subsumed_level)
{
#ifdef DEBUG
  printf("Taking highest subsumed level from best_ptr image number
%d\n", best_ptr->img_num);
#endif
        highest_subsumed_level = best_ptr->img_num/1000;
}

merged_ptr = best_ptr->merged_node_list;

#ifdef DEBUG
printf("Number to be merged with this node: %d\n", best_ptr-
>num_merged);
#endif
while(merged_ptr && (count_merged < best_ptr->num_merged))
{
#ifdef DEBUG
  printf("merge with image number %d at index %d\n",
        merged_ptr->img_num, merged_ptr->index);
#endif

  count_merged++;
```

```
                // Read in distances from file to be merged
                sprintf(file2, "/tmp/distances/dist_%d", merged_ptr->img_num);
        dist2 = fopen(file2, "r");
        if(!dist2)
        {
                  printf("Failed to open distance file %s\n", file2);
                  return 1;
        }

                i = 0;
#ifdef DEBUG
                printf("Reading distances for merging node\n");
#endif
                while((!feof(dist2)) && (i < NUM_ANGLES))
                {
                  fgets(buf, 100, dist2);
                  if(feof(dist2))
                            break;
                  items_read = sscanf(buf, "%d%*c%f",&empty_2[i], &distances_2[i]);
                  i++;
                }

                // Mark merged as subsumed
                usable_img_num = (merged_ptr->img_num)%1000;
                prev_subsumed = mark_subsumed(usable_img_num, index_list,
subsumed);
                // Count the node we're subsuming as well as all the nodes it had
                // subsumed
                sum_subsuming += (1+prev_subsumed);

               // printf("Number subsuming with this node: %d sum: %d\n",
```

```
1+prev_subsumed, sum_subsuming);

                // Align on minimum distance index, and populate sum
                // If one of two is not empty, then the result should be not empty
                for(i=0; i<NUM_ANGLES; i++)
                {
                  aligned_index = (i+merged_ptr->index)%NUM_ANGLES;

                  empty_sum[i] &= empty_2[aligned_index];
                  if(!empty_2[aligned_index])
                                {
                     distance_sum[i] += (float)(1+ prev_subsumed) *
distances_2[aligned_index];
                                #ifdef DEBUG
                     printf("Reading distance_2[%d] as %f from index %d\n", i,
(float)distances_2[aligned_index], aligned_index);
                     printf("Adding %f to distance_sum[%d]\n",
(float)(1+prev_subsumed)*distances_2[aligned_index], i);
                                #endif
                                 }
                }

                if(merged_ptr->level > highest_subsumed_level)
                {
                                printf("Taking highest subsumed level from merged_ptr level
%d on image %d\n", merged_ptr->level, merged_ptr->img_num);
                                highest_subsumed_level = merged_ptr->level;
                }

#ifdef DEBUG
                printf("Moving from active to inactive list\n");
```

```
#endif
                move_to_inactive_list(&active_list, &inactive_list, merged_ptr);
                cluster_count--;

                merged_ptr = merged_ptr->next;
                depth_of_search++;
              }
#ifdef DEBUG
              printf("Creating a new active list node\n");
#endif

              // Create an new active list node for the new cluster

              new_active_node = (struct cluster_node *)malloc(sizeof(struct
cluster_node));

              new_active_node->level = highest_subsumed_level + 1;
              potential_img_num = (best_ptr->img_num)%1000 + new_active_node-
>level * 1000;
              while(!available(potential_img_num, unavailable))
                potential_img_num += 1000;
              new_active_node->img_num = potential_img_num;
              unavailable[num_to_print] = potential_img_num;
              new_active_node->distance_file_name = file_new;
              new_active_node->index = 0;
              new_active_node->distance = 0;
              new_active_node->num_merged = 0;
              new_active_node->merged_node_list = best_list;

#ifdef DEBUG
              printf("Adding to cluster list\n");
```

```
#endif
                add_to_new_cluster_list(new_active_node, best_list, index_list,
                                                &cluster_list, num_images);

                // Create the new distance file
                sprintf(file_new, "/tmp/distances/dist_%d", new_active_node->img_num);
        dist_new = fopen(file_new, "w");
        if(!dist_new)
        {
                  printf("Failed to open distance file %s\n", file_new);
                  return 1;
        }
#ifdef DEBUG
                printf("Created new distance file %s\n", file_new);
#endif

                // Write entries to new distance file
                for(i=0; i<NUM_ANGLES; i++)
                {
                  sprintf(distance_buf, "%d\t%f\n", empty_sum[i],
distance_sum[i]/(float)(node_factor+sum_subsuming));
                  fprintf(dist_new, distance_buf);
                }
                fprintf(dist_new, "cluster_%d\n", new_active_node->img_num);

                if(dist_new)
                  fclose(dist_new);

                // Update node distances
                update_distance_matrix(new_active_node, active_list,
                                                        inactive_list, img_list,
```

```
                                                    index_list, num_images, subsumed);

#ifdef DEBUG_VERBOSE
    // Display the clusters
            printf("Active and inactive lists before moving nodes:\n");
            printf("Active nodes:\n");
            show_nodes(active_list, subsumed, index_list);
            printf("Inactive nodes:\n");
            show_nodes(inactive_list, subsumed, index_list);
#endif
            // Move the current nodes to the inactive list

#ifdef DEBUG
            printf("Moving from active to inactive list\n");
#endif
            move_to_inactive_list(&active_list, &inactive_list, best_ptr);

#ifdef DEBUG
            printf("Adding to active list\n");
#endif
            add_to_list(&active_list, new_active_node);

#ifdef DEBUG_VERBOSE
    // Display the clusters
            printf("Active and inactive lists after moving nodes:\n");
            printf("Active nodes:\n");
            show_nodes(active_list, subsumed, index_list);
            printf("Inactive nodes:\n");
            show_nodes(inactive_list, subsumed, index_list);
#endif
```

```
#ifdef DEBUG
            // Show updated distance matrix
            printf("Updated distance matrix:\n");
      //show_relevant_distance_matrix(img_list, num_images, inactive_list);
      show_distance_matrix(img_list, num_images);
#endif

            if(best_ptr->distance <= threshold)
            {
              if(best_ptr->distance > first_threshold)
              {
                printf("Displaying clusters at distance of %d\n", first_threshold);
                display_by_distance_new_cluster_list(&cluster_list);
                printf("%d of %d displayed\n", display_count, num_to_print);
/*
                sprintf(base_name, "cluster_%d", first_threshold);
                sprintf(cluster_file, "/tmp/clusters/%s", base_name);
                print_by_distance_new_cluster_list(&cluster_list, cluster_file);
                print_cluster_structures(&cluster_list, first_threshold, base_name);
*/
                display_count = 0;
                first_threshold += DISTANCE_THRESHOLD/10;
              }
            }
            else if(best_ptr->distance > threshold)
            {
              // Add function to display clusters on Web pages
              printf("Displaying clusters at distance of %d\n", threshold);
              display_by_distance_new_cluster_list(&cluster_list);
              printf("%d of %d displayed\n", display_count, num_to_print);
/*
```

```
                sprintf(base_name, "cluster_%d", threshold);
                sprintf(cluster_file, "/tmp/clusters/%s", base_name);
                print_by_distance_new_cluster_list(&cluster_list, cluster_file);
                print_cluster_structures(&cluster_list, threshold, base_name);
*/
                display_count = 0;
                threshold += DISTANCE_THRESHOLD;
            }
            if (cluster_count == CLUSTER_LIMIT)
            {
        show_cluster_list_stats(&cluster_list);
                printf("Displaying final cluster list\n");
                display_by_distance_new_cluster_list(&cluster_list);
                printf("%d of %d displayed\n", display_count, num_to_print);
/*
                strcpy(base_name, "final_cluster");
                sprintf(cluster_file, "/tmp/clusters/%s", base_name);
                print_by_distance_new_cluster_list(&cluster_list, cluster_file);
                print_cluster_structures(&cluster_list, 500, base_name);
*/
                finish_index();
            }

            update_subsumed_num(best_ptr->img_num, sum_subsuming, index_list,
subsumed);

            #ifdef DEBUG
            // Test
            dist_new = fopen(file_new, "r");
            if(!dist_new)
            {
```

```
            printf("Couldn't open file %s for reading\n", file_new);
          }
          else
          {
            printf("Contents of new distance file:\n");
            do{
                    fgets(buf, 100, dist_new);
                    if(feof(dist_new))
                      break;
                    printf("%s", buf);
            }while(!feof(dist_new));
          }
          if(dist_new)
            fclose(dist_new);
          #endif

#ifdef DEBUG
          printf("Checking depth of search\n");
#endif
          if(depth_of_search > search_depth)
            search_depth = depth_of_search;
          best_ptr = best_ptr->next;
  }

  num_inactive_nodes = count_nodes_on_list(inactive_list);
#ifdef DEBUG
  printf("Number of nodes on active list: %d\n", count_nodes_on_list(active_list));
  printf("Number of nodes on inactive list: %d\n", num_inactive_nodes);
#endif

  // Clean up
```

```
    if(best_ptr)
    {
            remove_ptr = best_list;
            while(remove_ptr)
            {
                remove_merged = remove_ptr->merged_node_list;
                // Free memory allocated for list of merged clusters
                while(remove_merged)
                {
                        curr_merged = remove_merged->next;
                        free(remove_merged);
                        remove_merged = curr_merged;
                }
                // Free memory allocated for list of best clusters
                cluster_ptr = remove_ptr->next;
                free(remove_ptr);
                remove_ptr = cluster_ptr;
        }
    }
    // Now, tie up loose ends
    best_list = best_ptr = curr_merged = new_active_node = NULL;
  }

//// End of clustering loop ////

    disconnect_from_db(sqlca);


    // Memory deallocation
    free(img_list);
    free(merged_clusters);
```

```
   list_head = cluster_list;
   list_ptr = list_head;
   while(list_ptr)
   {
            remove_list_ptr = list_ptr;
            list_ptr = list_ptr->next;
            remove_list_ptr->next = NULL;
            free(remove_list_ptr);
   }

   printf("Finished processing\n");

   return 0;
}

void initialize_cluster_list(struct cluster_info **cluster_list,
                                                int *image_nums,
                                                int num_images)
{
   int i;
   struct cluster_info *new_cluster, *curr, *head;
   head = *cluster_list;
   curr = head;


   for(i=0; i<num_images; i++)
   {
      num_to_print++;
            new_cluster = malloc(sizeof(struct cluster_info));
            if(!new_cluster)
```

```
            printf("MEMORY ALLOCATION ERROR\n");
          if(head)
          {
            curr->next = new_cluster;
            curr = curr->next;
            new_cluster = NULL;
          }
          else
          {
            head = new_cluster;
            curr = new_cluster;
            curr->next = NULL;
          }

          curr->distance = 0;
          curr->next = NULL;
          curr->points_to = NULL;
          curr->num_points_to = 0;
          curr->img_num = image_nums[i];
   curr->pointed_to_by = -1;
}

*cluster_list = head;
new_cluster = NULL;
curr = NULL;

i = 0;
curr = *cluster_list;
while(curr)
{
          i++;
```

```
            curr = curr->next;
    }

    curr = NULL;
    head = NULL;
    printf("%d cluster list nodes initialized\n", i);

}

void initialize_active_list(struct cluster_node **active_list,
                                                    int *image_nums,
                                                    int num_images)
{
    int i;
    char distance_file_name[40];
    struct cluster_node *new_cluster, *curr, *head;
    head = *active_list;
    curr = head;
    for(i=0; i<num_images; i++)
    {
            new_cluster = malloc(sizeof(struct cluster_node));
            if(!new_cluster)
              printf("MEMORY ALLOCATION ERROR\n");
            if(*active_list)
            {
              curr->next = new_cluster;
              new_cluster->previous = curr;
              curr = curr->next;
              new_cluster = NULL;
            }
            else
```

```
            {
              head = new_cluster;
              curr = new_cluster;
              curr->next = NULL;
              curr->previous = NULL;
              *active_list = new_cluster;
            }

            curr->distance = -1;
            curr->level = 0;
            curr->next = NULL;
            curr->merged_node_list = NULL;
            curr->img_num = image_nums[i];
   curr->num_merged = 0;
   curr->merged_node_list = NULL;

            sprintf(distance_file_name, "/tmp/distances/dist_%d", image_nums[i]);
            curr->distance_file_name = distance_file_name;
}

*active_list = head;
new_cluster = NULL;
curr = NULL;

i = 0;
curr = *active_list;
while(curr)
{
            i++;
            curr = curr->next;
}
```

```
    curr = NULL;
    head = NULL;
    printf("%d nodes initialized\n", i);

}

void print_retrieval(int query_image, char *jpg_name,
                                        float distance, int ret_num)
{
  char file_name[30]={0};
  FILE *ret_file;

  sprintf(file_name, "/tmp/clusters/img_%d.html", query_image);

  ret_file=fopen(file_name, "a");
  if(! ret_file)
  {
   printf("Failed to open %s for writing\n", file_name);
   return;
  }

   fprintf(ret_file, "<H2>Retrieval %d: %s</H2>", ret_num, jpg_name);
   fprintf(ret_file, "<IMG SRC='../masked_jpg/%s'><BR>", jpg_name);
   fprintf(ret_file, "distance: %f", (float)distance);
   fclose(ret_file);
}

void finish_index(void)
{
  FILE *ind_file;
```

```
  char *index_file="/tmp/clusters/index.html";

  ind_file=fopen(index_file, "a");
  if(! ind_file)
  {
   printf("Failed to open %s for writing\n", index_file);
   return;
  }

   fprintf(ind_file, "</BODY></HTML>");
   fclose(ind_file);
}

void index_header()
{
  FILE *ind_file;
  char *index_file="/tmp/clusters/index.html";

  ind_file=fopen(index_file, "w");
  if(! ind_file)
  {
   printf("Failed to open %s for writing\n", index_file);
   return;
  }
   fprintf(ind_file,"<HTML><HEAD><TITLE>Cluster Analysis of Retrieval of
Petroglyph Images from an Image Database</TITLE></HEAD>\n");
   fprintf(ind_file,"<BODY BGCOLOR=#FFFFFF><CENTER><H1>Cluster Analysis
of Retrieval of Petroglyph Images from an Image Database</H1>\n");
   fprintf(ind_file,"<H2>Cluster Methodology</H2>\n");

   fprintf(ind_file, "</CENTER>The clustering methodology used for these trials was a
```

```
centroid methodology.  A distance matrix was calculated between all images in the
database; the matrix was then searched for the images having the least distance between
them.  The distance files for these two images were then agglomerated on the index
producing least distance to form a new distance file representing the cluster; distances
between the cluster and the the other images/clusters in the database are then
recalculated.  The clustering process ends when a single cluster remains.<BR><BR>\n");
   fprintf(ind_file,"<CENTER><H2>Retrievals</H2>\n");
   fclose(ind_file);
}

void print_header(char *jpg_name, int image_number)
{
  char file_name[30]={0};
  char short_name[20]={0};
  char *index_file="/tmp/retrieval/index.html";
  FILE *ret_file, *ind_file;

  sprintf(short_name, "img_%d.html", image_number);
  sprintf(file_name, "/tmp/retrieval/%s", short_name);

  ret_file=fopen(file_name, "w");
  if(! ret_file)
  {
   printf("Failed to open %s for writing\n", file_name);
   return;
  }

  ind_file=fopen(index_file, "a");
  if(! ind_file)
  {
   printf("Failed to open %s for writing\n", index_file);
```

```
    return;
  }

  fprintf(ind_file, "<A HREF='%s'>%s</A><BR>\n", short_name, short_name);

  fprintf(ret_file,"<HTML><HEAD><TITLE>Top 20 Retrievals for Image
%d</TITLE></HEAD>\n", image_number);
  fprintf(ret_file,"<BODY BGCOLOR=#FFFFFF><CENTER><H1>Top 20 Retrievals
for Image %d</H1>\n", image_number);

  fprintf(ret_file,"<H2>Image %d (%s)\n", image_number, jpg_name);
  fprintf(ret_file,"</H2>\n<IMG SRC='../masked_jpg/%s'>\n", jpg_name);

  fclose(ind_file);
  fclose(ret_file);

}

void print_cluster_structures(struct cluster_info **cluster_list, int threshold_num, char
*filename)
{
  FILE *cluster_file, *cluster_temp, *cluster_index;
  char buf[100]={0};
  char index_buf[150] = {0};
  char file_name[100] = {0};
  char temp_name[100] = {0};
  char *index_name = "/tmp/clusters/index.html";

  sprintf(temp_name, "/tmp/clusters/%s", filename);
  sprintf(file_name, "/tmp/clusters/%s.html", filename);
```

```
  cluster_temp=fopen(temp_name, "r");
  cluster_file=fopen(file_name, "w");
  cluster_index=fopen(index_name, "a");

  if(! cluster_temp)
  {
   printf("Couldn't open %s file for reading\n", temp_name);
   return;
  }
  if(! cluster_file)
  {
   printf("Couldn't open %s file for writing\n", file_name);
   return;
  }

  if(! cluster_index)
  {
   printf("Couldn't open %s file for writing\n", file_name);
   return;
  }

  sprintf(index_buf, "<A HREF=\"%s\">Clusters at distance %d</A><BR>\n",
file_name, threshold_num);
  fprintf(cluster_index, index_buf);

  fprintf(cluster_file,"<HTML><HEAD><TITLE>Image Retrieval Clusters at Distance
%d</TITLE></HEAD>\n", threshold_num);
  fprintf(cluster_file,"<BODY BGCOLOR=#FFFFFF><CENTER><H1>Image
Retrieval Clusters at Distance %d</H1></CENTER>\n", threshold_num);

  print_html_cluster_list(cluster_list, file_name);
```

```
   fprintf(cluster_file, "<H2>Cluster Retrieval Structure</H2>\n");

   fprintf(cluster_file, "<PRE>");
   while(!feof(cluster_temp))
   {
     fgets(buf, 100, cluster_temp);
     fprintf(cluster_file, "%s", buf);
   }
   fprintf(cluster_file, "</PRE>");

   fprintf(cluster_file, "<BR><BR><A HREF='index.html'>BACK</A>");
   fprintf(cluster_file, "</BODY></HTML>");

   fclose(cluster_temp);
   fclose(cluster_file);
   fclose(cluster_index);
}

void show_retrieval_stats(struct retrieval_info *retrieval_sums,
                                        int *img_type_counts, int num_img_types,
                                        int num_images)
{
   int i;
   FILE *stats_file;
   char *file_name="/tmp/stats.txt";

   stats_file=fopen(file_name, "w");

   fprintf(stats_file, "<H2>Image Retrieval by Type</H2>\n<PRE>");
   // Show statistics by image type for retrieval
```

```
   printf("    \t %15s          Average \t      \t\t# Better  \n", "\t");
   fprintf(stats_file, "    \t %15s          Average \t      \t\t# Better  \n", "\t");
   printf("Type\t %15s\t  Retrieval\t Random\t \tThan Random\n", "Name");
   fprintf(stats_file,"Type\t %15s\t  Retrieval\t Random\t \tThan Random\n", "Name");
   printf("----\t %15s\t  ---------\t ------\t \t----------\n", "----");
   fprintf(stats_file, "----\t %15s\t  ---------\t ------\t \t----------\n", "----");
   for(i=0; i<num_img_types; i++)
   {
    printf("%d\t %15s\t %f\t %f\t %d/%d\n", retrieval_sums[i].type_num,
retrieval_sums[i].name,
retrieval_sums[i].sum/(float)img_type_counts[i],(float)img_type_counts[i]/(float)num_i
mages, retrieval_sums[i].best, img_type_counts[i]);
    fprintf(stats_file, "%d\t %15s\t %f\t %f\t %d/%d\n", retrieval_sums[i].type_num,
retrieval_sums[i].name,
retrieval_sums[i].sum/(float)img_type_counts[i],(float)img_type_counts[i]/(float)num_i
mages, retrieval_sums[i].best, img_type_counts[i]);
   }

   fprintf(stats_file, "</PRE>");
   fclose(stats_file);
}

void get_jpg_name(char *short_name, char* jpg_name)
{
  int name_len, i, dot_index;

  // Clear out short name string holder
  for(i=0; i<(strlen(jpg_name)); i++)
   jpg_name[i] = 0;

  dot_index=0;
```

```
  name_len = strlen(short_name);
  for(i=name_len; i>=0; i--)
  {
    if(short_name[i] == '.')
  // Modified to break at _thresh
  //   if(short_name[i] == '_')
    {
      dot_index = i;
      break;
    }
  }

  strncpy(jpg_name, short_name, i+1);
  strcpy(jpg_name+i+1, "jpg");
}

void get_short_name(char *long_name, char* short_name)
{
  int name_len, i, slash_index;

  // Clear out short name string holder
  for(i=0; i<(strlen(short_name)); i++)
   short_name[i] = 0;

  slash_index=0;
  name_len = strlen(long_name);
  for(i=name_len; i>=0; i--)
  {
    if(long_name[i] == '/')
    {
      slash_index = i;
```

```
        break;
      }
    }

  strcpy(short_name, long_name+i+1);
}

int get_num_images(struct sqlca sqlerrstruct)
{
   int error_num;

   exec sql begin declare section;
   int num_images;
   exec sql end declare section;

   exec sql select count(*) into :num_images from images;
   error_num = check_db_error(sqlca);
   check_error(error_num, "Getting number of images");

   return num_images;
}

int comp(const void *a, const void *b)
{
   float difference;

   difference = ((struct img_info *)a)->distance - ((struct img_info *)b)->distance;
   if(difference < 0)
     return -1;
   else if(difference)
     return 1;
```

```
    else
      return 0;
}

void check_error(int err_num, char *msg)
{

    if(err_num)
    {
      printf("Error %d: %s\n", err_num, msg);
      if(err_num < 0)
        exit(1);
    }
  #ifdef DEBUG
    else
      printf("%s\n", msg);
  #endif
}

void connect_to_db(struct sqlca sqlerrstruct)
{
    int err_num;

    exec sql connect to glyph;
    err_num = check_db_error(sqlerrstruct);
    check_error(err_num, "Connect to db");
}

void disconnect_from_db(struct sqlca sqlerrstruct)
{
    int err_num;
```

```
   exec sql disconnect all;
   err_num = check_db_error(sqlerrstruct);
   check_error(err_num, "Disconnecting from db");
}

int check_db_error(struct sqlca sqlerrstruct)
{
  if(sqlerrstruct.sqlcode != 0)
  {
    if(sqlerrstruct.sqlcode == 100)
      printf("End of cursor/no data found\n");
    else
    {
      printf("SQL error number %d:\n", (int)sqlerrstruct.sqlcode);
      printf("Line %d: Error: %s\n", (int)sqlerrstruct.sqlerrm.sqlerrml,
                                                    sqlerrstruct.sqlerrm.sqlerrmc);
    }
  }

  return (int)sqlerrstruct.sqlcode;
}

int get_smallest_retrieval_list(struct img_info *img_list, int num_images,
                                                    struct cluster_node **best_list, int
*image_nums,
                                                    int *subsumed, int *index_list,
                                                    struct cluster_node *inactive_list)
{

   struct cluster_node *best_head;
```

```
   struct cluster_node *remove_merged, *curr_merged, *merged_ptr;
   struct cluster_node *cluster_ptr, *remove_ptr, *best_ptr;
   int num_best, search_depth, num_inspected;
   int depth_of_search, i, j, index1, usable_image_num;
   int merge_index, image_number, index2, last_index, best_index;
   int num_rows_skipped, num_img_skipped;
   float curr_dist, best_distance, prev_dist, smallest_distance;

   // Find smallest retrieval distances
   smallest_distance = 501;
   num_best = 0;
   best_index = 0;
   search_depth = 1;
   depth_of_search = 0;
   last_index = 0;
   num_rows_skipped = 0;

#ifdef DEBUG
   printf("--- Finding minimum retrieval distances ---\n");
#endif
   for(i=0; i<num_images; i++)
   {
              num_img_skipped = 0;
              if(is_inactive(img_list[last_index].img_num, inactive_list))
              {
                num_rows_skipped++;
                last_index += num_images - i;
                continue;
              }
              num_inspected = 0;
              for(j = 1; j<search_depth+1; j++)
```

```
        {
          if(j>=(num_images-i))
                    break;
      index1= last_index + j;
          image_number = img_list[index1].img_num;
          curr_dist = img_list[index1].distance;
          merge_index = img_list[index1].index;

          if(!image_number)
          {
            printf("Error in img_list: image_number = 0\n");
            return 0;
          }

          usable_image_num = image_number%1000;
          if(is_inactive(image_number, inactive_list))
          {
                    num_img_skipped++;
                    search_depth++;
                    continue;
          }

          num_inspected++;

          best_distance = curr_dist;
          if (best_distance <= smallest_distance)
          {
            // If new best distance, clear out old list
            if(best_distance < smallest_distance)
            {
#ifdef DEBUG
```

```
                    printf("Renovating best_list: new best distance %f\n",
(float)best_distance);
                    printf("from img_num %d to img_num %d\n",
image_nums[i], image_number);
#endif
                    num_best = 0;
                smallest_distance = best_distance;
                    best_index = i;
                if(*best_list)
                {
                    remove_ptr = *best_list;
                    while(remove_ptr)
                    {
                      remove_merged = remove_ptr->merged_node_list;
                      // Free memory allocated for list of merged clusters
                      while(remove_merged)
                      {
                              curr_merged = remove_merged->next;
                              free(remove_merged);
                              remove_merged = curr_merged;
                      }
                      // Free memory allocated for list of best clusters
                            remove_merged = NULL;
                      cluster_ptr = remove_ptr->next;
                      free(remove_ptr);
                      remove_ptr = cluster_ptr;
                    }
                    *best_list = NULL;
                    best_head = NULL;
                    cluster_ptr = NULL;
                    merged_ptr = NULL;
```

```
                }
        }
#ifdef DEBUG
        else
        {
                printf("Adding additional node to merged list\n");
        }
#endif
        num_best++;

#ifdef DEBUG
        printf("Allocating memory for first entry on merged cluster list\n");
#endif
        // Allocate memory for first entry on merged cluster list
        merged_ptr = malloc(sizeof(struct cluster_node));
        merged_ptr->distance = smallest_distance;
        merged_ptr->img_num = image_number;
        merged_ptr->level = subsumed[index_list[usable_image_num]];
        merged_ptr->index = merge_index;
        merged_ptr->next = NULL;
        merged_ptr->merged_node_list = NULL;
        merged_ptr->previous = NULL;

#ifdef DEBUG
        printf("Allocating memory for first node on best distance list\n");
#endif
        // Allocate memory for new node on best distance list
        cluster_ptr = malloc(sizeof(struct cluster_node));
        cluster_ptr->distance = smallest_distance;
        cluster_ptr->img_num = img_list[last_index].img_num;
        cluster_ptr->merged_node_list = merged_ptr;
```

```
                cluster_ptr->num_merged = 1;
                cluster_ptr->next = NULL;
                cluster_ptr->previous = NULL;

#ifdef DEBUG
                printf("Assigning list pointer to first node cluster\n");
#endif

                if(*best_list)
                {
#ifdef DEBUG
                        printf("best_list not null at time of assignment\n");
#endif
                        best_head = *best_list;
                        while(*best_list)
                        {
                          *best_list = (*best_list)->next;
                        }
                        *best_list = cluster_ptr;
                }
                else
                {
#ifdef DEBUG
                        printf("best_list null at time of assignment\n");
#endif
                        *best_list = cluster_ptr;
                  best_head = *best_list;
                }


                // Go down the list of retrievals for this image, looking
```

```
                // for distance matches
        for(j=search_depth+1; j<num_images-i; j++)
                {
                        prev_dist = curr_dist;
              index2 = last_index + j;
                        image_number = img_list[index2].img_num;
                  merge_index = img_list[index2].index;
                        curr_dist = img_list[index2].distance;
                       #ifdef DEBUG
                        printf("image number %d has distance %f at index %d\n",
image_number, curr_dist, merge_index);
                       #endif
                  usable_image_num = image_number%1000;

                  if(is_inactive(image_number, inactive_list))
                        {
                          num_img_skipped++;
                          continue;
                        }
                        if(curr_dist > prev_dist)
                        {
                          curr_dist = prev_dist;
                          break;
                        }
                        else if(curr_dist == prev_dist)
                        {
                    best_ptr = malloc(sizeof(struct cluster_node));
                          best_ptr->img_num = image_number;
                          best_ptr->index = merge_index;
                          best_ptr->previous = merged_ptr;
                          best_ptr->next = NULL;
```

```
                        best_ptr->merged_node_list = NULL;
                        merged_ptr->next = best_ptr;
                        cluster_ptr->num_merged++;
                        best_ptr = NULL;
                      }
                  }

                }
                else
                {
                        // Stop looking if this image has greater distance than smallest
                        break;
                }
              }

              last_index += num_images - i;
#ifdef DEBUG
              if(num_img_skipped)
                printf("Number of images skipped row %d: %d\n", i, num_img_skipped);
#endif
   }

#ifdef DEBUG
   printf("Number of rows skipped: %d of %d\n", num_rows_skipped, num_images);
#endif

   return num_best;
}

int mark_subsumed(int img_num, int *index_list, int *subsumed)
{
```

```
    int num_previously_subsumed;

    // Clusters will be build with img_nums that are multiples of 1000
    // higher than base image
    #ifdef DEBUG
      printf("marking img_num %d subsumed\n", img_num);
    #endif
    num_previously_subsumed = subsumed[index_list[img_num]];
    subsumed[index_list[img_num]] = -1;

    return num_previously_subsumed;
}

int has_been_subsumed(int subsumed_marker)
{
    if(subsumed_marker < 0)
            return 1;
    else
            return 0;
}
void update_subsumed_num(int img_num, int sum_subsuming,
                                        int *index_list, int *subsumed)
{
    int usable_img_num;

    // Clusters will be build with img_nums that are multiples of 1000
    // higher than base image
    usable_img_num = img_num%1000;
    subsumed[index_list[usable_img_num]] += sum_subsuming;
#ifdef DEBUG
    printf("Setting subsumed to %d for image num %d\n",
```

```
subsumed[index_list[usable_img_num]], img_num);
#endif
}

void add_to_list(struct cluster_node **list,
                                        struct cluster_node *new_cluster_ptr)
{
   struct cluster_node *head, *ptr, *tail;
   int usable_ptr_img_num, usable_new_img_num;
   int added = 0;

   // Adds cluster in numeric order
#ifdef DEBUG
   printf("Adding new cluster to list\n");
#endif

   usable_new_img_num = new_cluster_ptr->img_num%1000;

   head = *list;
   ptr = head;

   if(!ptr)
   {
#ifdef DEBUG
            printf("Adding to head\n");
#endif
            head = new_cluster_ptr;
            new_cluster_ptr->next = NULL;
            new_cluster_ptr->previous = NULL;
      *list = head;
            return;
```

```
    }

    while(ptr)
    {
        usable_ptr_img_num = ptr->img_num%1000;

        // Find tail (if we run through list)
                tail = ptr;

                if(usable_new_img_num > usable_ptr_img_num)
                {
          ptr = ptr->next;
                    continue;
                }
                else
                {
         new_cluster_ptr->next = ptr;
#ifdef DEBUG
                    printf("Adding node in line: ptr is %d new is %d\n", usable_ptr_img_num,
usable_new_img_num);
#endif
         new_cluster_ptr->previous = ptr->previous;
                    if(ptr->previous)
                    {
                      (ptr->previous)->next = new_cluster_ptr;
                    }
                    else
                    {
#ifdef DEBUG
                                printf("Actually, adding to head\n");
#endif
```

```
                        head = new_cluster_ptr;
                    }
                    ptr->previous = new_cluster_ptr;
                    added = 1;
                    break;
                }
    }

    if(!added)
    {
#ifdef DEBUG
        printf("Adding node to tail: new is %d\n", usable_new_img_num);
#endif
        if(tail)
          tail->next = new_cluster_ptr;
        else
                head = new_cluster_ptr;
        new_cluster_ptr->previous = tail;
        new_cluster_ptr->next = NULL;
    }

    *list = head;
}

void move_to_inactive_list(struct cluster_node **active_list,
                                        struct cluster_node **inactive_list,
                                        struct cluster_node *merged_ptr)
{
    struct cluster_node *active_head, *active_ptr, *remove_ptr;
    struct cluster_node *before_node, *after_node;
    int found=0;
```

```
    active_head = *active_list;
    active_ptr = active_head;

    if(!merged_ptr)
    {
            printf("Error: can't move a null pointer to/from lists\n");
            return;
    }

    // Find node to remove
    while(active_ptr)
    {
            if((active_ptr->img_num) == (merged_ptr->img_num))
            {
                found = 1;
                remove_ptr = active_ptr;
                before_node = active_ptr->previous;
                after_node = active_ptr->next;
                if(before_node)
                  before_node->next = after_node;
                else
                        active_head = after_node;
                if(after_node)
                        after_node->previous = before_node;

                remove_ptr->next = NULL;
                remove_ptr->previous = NULL;

#ifdef DEBUG
                printf("Adding to inactive list\n");
#endif
```

```
                add_to_list(inactive_list, remove_ptr);
                break;
            }
            active_ptr = active_ptr->next;
    }
#ifdef DEBUG
    if(!found)
            printf("Active list node with image number %d not found\n", merged_ptr-
>img_num);
    else
            printf("Moved list node with image number %d from active list\n",
merged_ptr->img_num);
#endif

    *active_list = active_head;
}

int count_nodes_on_list(struct cluster_node *list)
{
    struct cluster_node *list_head;
    int num_nodes = 0;

    list_head = list;

    while(list_head)
    {
            list_head = list_head->next;
            num_nodes++;
    }

    return num_nodes;
```

```
}

void show_nodes(struct cluster_node *list, int *subsumed, int *index_list)
{
   struct cluster_node *list_head, *list_ptr;
   int i, usable_img_num;
   char buf[40] = {0};

   list_head = list;
   list_ptr = list_head;

   while(list_ptr)
   {
     usable_img_num = list_ptr->img_num%1000;
     if(list_ptr->level == 0)
        printf("L%d: img_num %d\n", list_ptr->level, list_ptr->img_num);
     else
     {
                for(i=0; i<list_ptr->level; i++)
                  strcat(buf, "\t");
        printf("%sL%d: img_num %d\n", buf, list_ptr->level, list_ptr->img_num);
                strcpy(buf, "");
     }

     list_ptr = list_ptr->next;
   }
}

void update_distance_matrix(struct cluster_node *new_active_cluster,
                                                struct cluster_node *active_list,
                                                struct cluster_node *inactive_list,
```

```
                                        struct img_info *img_list,
                                        int *index_list, int num_images, int *subsumed)
{
  struct img_info *new_distances_below_index, *new_distances_from_index;
  struct cluster_node *active_head, *active_ptr;
  struct dist_info *calculated_distance;
  int num_active, num_at_or_above_new, active_index, index1, index2, i, j;
  int usable_active_img_num, usable_cluster_img_num, last_index, index;
  int index_correction, next_correction,shift_index;
  int num_updated, too_low, too_high;
  int prev_active_img_num, first_img_num, prev_img_num, expected_img_num;
  float new_distance;
  char file1[30], file2[30];

  // Line 1171

  // Get the base image number for the very first image in the list
  first_img_num = img_list[0].img_num%1000;

  // Get the base image number from the new cluster
  usable_cluster_img_num = (new_active_cluster->img_num)%1000;

  active_index = index_list[usable_cluster_img_num];
  num_active = count_nodes_on_list(active_list);
  num_at_or_above_new = num_images - active_index;

  // Allocate 2 arrays: 1 for distances at or above index, the other for
  // those below.  Second one will be sorted and used to replace entries on
  // corresponding index; first will be used to update existing indices

  // Need to know the number of subsumed nodes before and after the
```

```
   // active index

   num_updated = 0;

   new_distances_below_index = (struct img_info *)malloc(sizeof(struct
img_info[active_index]));
   new_distances_from_index = (struct img_info *)malloc(sizeof(struct
img_info[num_at_or_above_new]));

#ifdef DEBUG
   printf("Size of below_index array: %d\n", active_index);
   printf("Size of on_or_after_index array: %d\n", num_at_or_above_new);
#endif

   active_head = active_list;
   active_ptr = active_head;


   index1 = 0;
   index2 = 0;

   // Create file name from new cluster image number
   sprintf(file1, "/tmp/distances/dist_%d", new_active_cluster->img_num);

   while(active_ptr)
   {
            if(active_ptr == active_head)
              expected_img_num = first_img_num;
            else
              expected_img_num = prev_active_img_num + 1;
```

```
            usable_active_img_num = active_ptr->img_num%1000;

            // If not on active list, make sure old img_num is inserted
            // in proper place in sequence
            if(usable_active_img_num != expected_img_num)
            {
#ifdef DEBUG
              printf("Out-of-order substitution: prev=%d curr=%d\n",
prev_active_img_num, usable_active_img_num);
#endif
              usable_active_img_num = expected_img_num;
              if(usable_active_img_num < usable_cluster_img_num)
              {
                new_distances_below_index[index2].img_num =
usable_active_img_num;
                new_distances_below_index[index2].distance = 501;
                new_distances_below_index[index2].index = 0;
                index2++;
              }else{
                new_distances_from_index[index1].img_num =
usable_active_img_num;
                new_distances_from_index[index1].distance = 501;
                new_distances_from_index[index1].index = 0;
                index1++;
              }
              prev_active_img_num = usable_active_img_num;
              num_updated++;
              continue;
            }

            if(usable_active_img_num == usable_cluster_img_num)
```

```
            {
              new_distances_from_index[index1].img_num = new_active_cluster-
>img_num;
              new_distances_from_index[index1].distance = 0.0;
              new_distances_from_index[index1].index = 0;
              index1++;
            }
            else
            {
              // Create file name from active node image number
        sprintf(file2, "/tmp/distances/dist_%d", active_ptr->img_num);
              calculated_distance = calculate_b_dist3(file1, file2);

              if(usable_active_img_num < usable_cluster_img_num)
              {
                new_distances_below_index[index2].img_num = active_ptr->img_num;
                if(!is_inactive(active_ptr->img_num, inactive_list))
                {
                  new_distances_below_index[index2].distance = calculated_distance-
>distance;
                  new_distances_below_index[index2].index = calculated_distance-
>index;
                }
                else
                {
                        new_distances_below_index[index2].index = 0;
                        new_distances_below_index[index2].distance = 501;
                }
                index2++;
              }
              else
```

```
                {
                  new_distances_from_index[index1].img_num = active_ptr->img_num;
                  if(!is_inactive(active_ptr->img_num, inactive_list))
                  {
                    new_distances_from_index[index1].distance = calculated_distance-
>distance;
                    new_distances_from_index[index1].index = calculated_distance-
>index;
                  }
                  else
                  {
                          new_distances_from_index[index2].index = 0;
                          new_distances_from_index[index2].distance = 501;
                  }

                  index1++;
                }
              }
              num_updated++;
              prev_img_num = active_ptr->img_num;
              prev_active_img_num = usable_active_img_num;
              active_ptr = active_ptr->next;
  }

  expected_img_num = (prev_active_img_num)%1000 + 1;
  if(num_updated < num_images)
  {
#ifdef DEBUG
              printf("num_images: %d num_updated: %d prev_active_img_num %d,
expected_img_num: %d\n", num_images, num_updated, prev_active_img_num,
expected_img_num);
```

```
#endif
                for(i=num_updated; i<num_images; i++)
                {
#ifdef DEBUG
                  printf("Filling in missing number at end: %d\n", expected_img_num);
#endif
                  if(expected_img_num < usable_cluster_img_num)
                  {
                    new_distances_below_index[index2].img_num = expected_img_num;
                    new_distances_below_index[index2].index = 0;
                    new_distances_below_index[index2].distance = 501;
                    index2++;
                  }
                  else
                  {
                    new_distances_from_index[index1].img_num = expected_img_num;
                    new_distances_from_index[index1].index = 0;
                    new_distances_from_index[index1].distance = 501;
                    index1++;
                  }
                  expected_img_num++;
                }
  }

#ifdef DEBUG
  printf("Distances from cluster %d to indices below it:\n", new_active_cluster-
>img_num);

  for(i=0; i<active_index; i++)
  {
                printf("img_num %d\t", new_distances_below_index[i].img_num);
```

```
            if(new_distances_below_index[i].distance == 501)
              printf("-- inactive --\n");
            else
            {
              printf("distance %f\t", new_distances_below_index[i].distance);
              printf("index %d", new_distances_below_index[i].index);
              printf("\n");
            }
  }

  printf("Distances from cluster %d to indices at or above it:\n", new_active_cluster-
>img_num);
  for(i=0; i<num_at_or_above_new; i++)
  {
            printf("img_num %d\t", new_distances_from_index[i].img_num);
            if(new_distances_from_index[i].distance == 501)
              printf("-- inactive --\n");
            else
            {
              printf("distance %f\t", new_distances_from_index[i].distance);
              printf("index %d", new_distances_from_index[i].index);
      printf("\n");
            }
  }

#endif

  // Modify the image list       // Line 1241
  // Begin by updating indices below the active one
  last_index = 0;
  for(i=0; i<active_index; i++)
```

```
  {
            #ifdef DEBUG
              printf("Row %d: before update\n", i);
              for(j=0; j<num_images-i; j++)
              {
                index = last_index + j;
                printf("img_num %d distance %f\n", img_list[index].img_num,
img_list[index].distance);
              }
            #endif

            index_correction = 0;
            too_low = 0;
            too_high = 0;
            new_distance = new_distances_below_index[i].distance;
            // Look for the entry with the active cluster image in it
            // Replace the values with those of the active cluster
            // If this now makes the list out of order, reorder the
            // necessary elements
            for(j=1; j<num_images - i; j++)
            {
               index = last_index + j;

              if((img_list[index].img_num)%1000 == usable_cluster_img_num)
              {
                // Now that we've found the original placement, test to
                // see how things need to shift
                if(j<num_images-i-1)
                        if(img_list[index+1].distance < new_distance)
                        {
                          too_low = 1;
```

```
                    index_correction = 1;
                while((new_distance > img_list[index+index_correction].distance)
&& ((index+index_correction-last_index) < num_images-i))
                {
                                index_correction++;
                                if(index + index_correction - last_index ==
num_images -i)

                                {
                                  index_correction--;
                                  break;
                                }
                                next_correction = index_correction+1;
                                if(img_list[index+next_correction].distance >=
new_distance)
                                  break;
                }
                        }


            if(j>1)
                    if(img_list[index-1].distance > new_distance)
                    {
                      too_high = 1;
                      index_correction = -1;
                while((new_distance < img_list[index+index_correction].distance)
&& ((index+index_correction-last_index) > last_index))
                {
                                 index_correction--;
                                 if(index + index_correction - last_index ==
last_index)
```

```
                                        {
                                          index_correction++;
                                          break;
                                }
                                        next_correction = index_correction-1;
                                        if(img_list[index+next_correction].distance <=
new_distance)
                                          break;
                    }
                  }

#ifdef DEBUG
                if(too_low)
                        printf("Current placement at %d is too low for distance %f\n",
j, new_distance);

                if(too_high)
                        printf("Current placement at %d is too high for distance %f\n",
j, new_distance);

                if(index_correction)
            printf("Index correction in row %d of matrix: %d\n", i, index_correction);
                else
                         printf("Found no index correction\n");
#endif

                if(index_correction < 0)
                {
                         for(shift_index = index; shift_index>index+index_correction;
shift_index--)
                         {
```

```
                        img_list[shift_index].img_num = img_list[shift_index-
1].img_num;
                        img_list[shift_index].distance = img_list[shift_index-
1].distance;
                        img_list[shift_index].index = img_list[shift_index-1].index;
                    }
                }
                else if(index_correction > 0)
                {
                    for(shift_index = index; shift_index<index+index_correction;
shift_index++)
                    {
                        img_list[shift_index].img_num =
img_list[shift_index+1].img_num;
                        img_list[shift_index].distance =
img_list[shift_index+1].distance;
                        img_list[shift_index].index = img_list[shift_index+1].index;
                    }
                }

                img_list[index+index_correction].img_num = new_active_cluster-
>img_num;
                img_list[index+index_correction].distance =
new_distances_below_index[i].distance;
                img_list[index+index_correction].index =
new_distances_below_index[i].index;
                break;
              }

            }
```

```
            #ifdef DEBUG
              printf("Row %d: after update\n", i);
              for(j=0; j<num_images-i; j++)
              {
                index = last_index + j;
                printf("img_num %d distance %f\n", img_list[index].img_num,
img_list[index].distance);
              }
            #endif


            last_index += num_images - i;
  }

  // Sort the values at or above the active index
  qsort((void *)new_distances_from_index, (size_t)num_at_or_above_new,
            (size_t)(sizeof(* new_distances_from_index)), comp);

  #ifdef DEBUG_UPDATE
            printf("Active cluster row %d: before update\n", i);
            for(j=0; j<num_at_or_above_new; j++)
            {
              index = last_index + j;
              printf("img_num %d distance %f\n", img_list[index].img_num,
img_list[index].distance);
            }
  #endif

  // Now, update the row that begins with the lead image for the cluster
#ifdef DEBUG
  printf("Max index for list: %d\n", num_at_or_above_new);
```

```
#endif
   for(j=0; j<num_at_or_above_new; j++)
   {
                index = last_index+j;
                index_correction=0;
                if(j==0)
                {
                  if(usable_cluster_img_num != img_list[index].img_num%1000)
                  {
#ifdef DEBUG
                    printf("Initial image out of order!\n");
#endif
                    index_correction = 0;
                    while(usable_cluster_img_num !=
img_list[index+index_correction].img_num%1000)
                    {
                              index_correction++;
                    }
                    // Swap out-of-order elements
                    img_list[index+index_correction].distance = img_list[index].distance;
                    img_list[index+index_correction].img_num = img_list[index].img_num;
                    img_list[index+index_correction].index = img_list[index].index;
#ifdef DEBUG
                    printf("Swapped elements %d and %d\n", index,
index+index_correction);
#endif
                    index_correction = 0;
                  }
                }

                img_list[index].img_num = new_distances_from_index[j].img_num;
```

```
            img_list[index].distance = new_distances_from_index[j].distance;
            img_list[index].index = new_distances_from_index[j].index;
    }

    #ifdef DEBUG_UPDATE
            printf("Active cluster row %d: after update\n", i);
            for(j=0; j<num_at_or_above_new; j++)
            {
              index = last_index + j;
              printf("img_num %d distance %f", img_list[index].img_num,
img_list[index].distance);
              if(is_inactive(img_list[index].img_num, inactive_list))
              {

                printf(" (inactive)");
              }
              printf("\n");
            }
    #endif

    free(new_distances_below_index);
    free(new_distances_from_index);

}

void show_distance_matrix(struct img_info *img_list, int num_images)
{
  // Display current distance array

    int i, j, last_index, index;
```

```
   last_index = 0;
   printf("Distance matrix contents:\n");
   for(i=0; i<num_images; i++)
   {
              for(j=0; j<num_images-i; j++)
              {
                 index = last_index + j;
                 printf("%d\t%f\t%d\n", img_list[index].img_num,
                              img_list[index].distance, img_list[index].index);

              }
              last_index += num_images-i;
              printf("\n");
   }
}

void show_relevant_distance_matrix(struct img_info *img_list, int num_images,
                                                  struct cluster_node *inactive_list)
{
  // Display current distance array

   int i, j, last_index, index, skipped_img_count;

   last_index = 0;
   printf("Relevant distance matrix contents:\n");
   for(i=0; i<num_images; i++)
   {
              skipped_img_count = 0;
              if(is_inactive(img_list[last_index].img_num, inactive_list))
              {
                printf("Skipping row with image %d\n", img_list[last_index].img_num);
```

```
                last_index += num_images-i;
                continue;
            }
            for(j=0; j<num_images-i; j++)
            {
                index = last_index + j;
                if(is_inactive(img_list[index].img_num, inactive_list))
                {
                        skipped_img_count++;
                        continue;
                }
                printf("%d\t%f\t%d\n", img_list[index].img_num,
                        img_list[index].distance, img_list[index].index);

            }
            printf("skipping %d images in row %d\n", skipped_img_count, i);
            last_index += num_images-i;
            printf("\n");
    }
}

int is_inactive(int image_num, struct cluster_node *inactive_list)
{
  int inactive = 0;
  struct cluster_node *inactive_ptr;

  inactive_ptr = inactive_list;

  while(inactive_ptr)
  {
    if(inactive_ptr->img_num == image_num)
```

```
    {
            inactive = 1;
            break;
    }
    inactive_ptr = inactive_ptr->next;
  }

  return inactive;
}

void add_to_new_cluster_list(struct cluster_node *new_cluster,
                             struct cluster_node *best_list,
                             int *index_list, struct cluster_info **cluster_list,
                             int num_images)
{
   struct cluster_info *list_head, *list_ptr, *add_ptr, *add_head;
   struct cluster_node *best_head, *best_ptr, *merged_ptr;
   int num_pointed_to, i, num_created, num_marked;

#ifdef DEBUG
   printf("Inside add_to_new_cluster_list\n");
#endif

   list_head = *cluster_list;
   list_ptr = list_head;

   add_ptr = add_head = NULL;

   best_head = best_list;
   best_ptr = best_head;
```

```
   num_created = 0;

   while(best_ptr)
   {
    list_head = *cluster_list;
    list_ptr = list_head;
    num_pointed_to = 1;

    num_to_print++;
    // Allocate new node for cluster information
#ifdef DEBUG
    printf("Created cluster to copy best_ptr\n");
    printf("Copy node has img_num %d\n", new_cluster->img_num);
#endif
    add_ptr = malloc(sizeof(struct cluster_info));
    add_ptr->img_num = new_cluster->img_num;
    add_ptr->distance = best_ptr->distance;
    add_ptr->num_points_to = best_ptr->num_merged+1;
    add_ptr->points_to = malloc(sizeof(int[add_ptr->num_points_to]));
    add_ptr->points_to[0] = best_ptr->img_num;
    add_ptr->pointed_to_by = -1;
    add_ptr->next = NULL;

    merged_ptr = best_ptr->merged_node_list;
    while(merged_ptr)
    {
      // Fill "points_to" array with image numbers being referenced
      if(num_pointed_to < best_ptr->num_merged+1)
              {
                add_ptr->points_to[num_pointed_to] = merged_ptr->img_num;
```

```
            num_pointed_to++;
          }
          merged_ptr = merged_ptr->next;
}

num_created++;

// Add the newly-created node(s) to the list at the head
add_ptr->next = list_head;
add_head = add_ptr;
list_head = add_head;
*cluster_list = list_head;

// Mark "pointed_to" points
list_ptr = list_head;
num_marked = 0;
while(list_ptr)
{
          for(i=0; i<num_pointed_to; i++)
          {
            if(add_ptr->points_to[i] == list_ptr->img_num)
            {
              num_marked++;
              list_ptr->pointed_to_by = add_ptr->img_num;
              break;
            }
          }
          if(num_marked == num_pointed_to)
          {
          #ifdef DEBUG
              printf("All pointed-to marked\n");
```

```
                #endif
                    break;
                }
                list_ptr = list_ptr->next;
    }
    if(num_marked != num_pointed_to)
      printf("Error: num marked (%d) != num pointed to (%d)\n", num_marked,
num_pointed_to);


    best_ptr = best_ptr->next;
  }

#ifdef DEBUG
  printf("%d new nodes added\n", num_created);
#endif

}

void display_new_cluster_list(struct cluster_info **cluster_list)
{
    struct cluster_info *list_head, *list_ptr;
    int i;
    char *spacing = "\t";

    list_head = *cluster_list;
    list_ptr = list_head;

    while(list_ptr)
    {
                if(list_ptr->pointed_to_by == -1)
```

```
            {
              printf("\n\nimg_num %d", list_ptr->img_num);
              printf("--\n");
              if(list_ptr->distance > 0)
                printf("%f\n", list_ptr->distance);
              else
                printf("\n");
              display_count++;

              if(list_ptr->num_points_to)
              {
                for(i=0; i< list_ptr->num_points_to; i++)
                {
                  display_current_cluster(cluster_list, list_ptr->points_to[i],
                                                                    spacing);
                }
              }
            }

            list_ptr = list_ptr->next;
    }

}

void display_current_cluster(struct cluster_info **cluster_list,
                                                    int cluster_num, char *spacing)
{
   struct cluster_info *list_head, *list_ptr;
   char dash_buf[4] = {0};
   char space_buf[440] = {0};
   int i;
```

```
// Prepare spacing for each level
strcpy(space_buf, spacing);
strcat(space_buf, "  ");

// First, find the cluster
list_head = *cluster_list;
list_ptr = list_head;

while(list_ptr)
 {
   if(list_ptr->img_num == cluster_num)
            {
              if(list_ptr->num_points_to)
                strcpy(dash_buf, "--");

              if(list_ptr->pointed_to_by != -1)
              {
                display_count++;
                printf("%s--%d%s\n", spacing, list_ptr->img_num, dash_buf);
                if(list_ptr->distance > 0)
                  printf("%s%f\n", spacing, list_ptr->distance);
              }
              if(list_ptr->num_points_to)
              {
                for(i=0; i<list_ptr->num_points_to; i++)
                {
                  display_current_cluster(cluster_list, list_ptr->points_to[i],
                                                                space_buf);
                }
              }
```

```
                break;
            }

            list_ptr = list_ptr->next;
    }
}

void show_cluster_list_stats(struct cluster_info **cluster_list)
{
  int total_clusters, num_pointed_to, i;
  struct cluster_info *list_head, *list_ptr;

  list_head = *cluster_list;
  list_ptr = list_head;
  num_pointed_to = 0;
  total_clusters = 0;

  while(list_ptr)
  {
    total_clusters++;
    printf("cluster %d: pointed_to_by: %d num_points_to: %d\n", list_ptr->img_num,
list_ptr->pointed_to_by, list_ptr->num_points_to);
    if(list_ptr->num_points_to)
    {
      for(i=0; i<list_ptr->num_points_to; i++)
            printf("points_to[%d]: %d\n", i, list_ptr->points_to[i]);
    }

    if(list_ptr->pointed_to_by != -1)
     num_pointed_to++;
```

```
    list_ptr = list_ptr->next;
  }
  printf("%d total clusters in list, %d of which are pointed to\n", total_clusters,
num_pointed_to);

}
int available(int img_num, int *unavailable)
{
  int i, unassigned;

  unassigned = 1;

  for(i = 0; i<num_to_print; i++)
  {
    if(unavailable[i] == img_num)
    {
      unassigned = 0;
            break;
    }
  }

  return unassigned;
}
void initialize_unavailable(int *unavailable, int *image_nums, int num_images)
{
  int i;

  for(i=0; i<num_images; i++)
  {
    unavailable[i] = image_nums[i];
```

```
  }
}

void display_by_distance_new_cluster_list(struct cluster_info **cluster_list)
{
    struct cluster_info *list_head, *list_ptr;
    int i;
    char space_buf[100] = {0};

    list_head = *cluster_list;
    list_ptr = list_head;

    while(list_ptr)
    {
                if(list_ptr->pointed_to_by == -1)
                {
                  strcpy(space_buf, "");
                  for(i=0; i<list_ptr->distance/10; i++)
                    strcat(space_buf, " ");
                  //printf("\n--------------------------------------------------\n%s--img_num
%d\n", space_buf, list_ptr->img_num);
                  printf("\n--------------------------------------------------\n);
               /*
                  if(list_ptr->distance > 0)
                    printf("%s%f\n", space_buf, list_ptr->distance);
                  else
               */
                    printf("\n");
                  display_count++;

                  if(list_ptr->num_points_to)
```

```
                {
                  for(i=0; i< list_ptr->num_points_to; i++)
                  {
                    display_by_distance_current_cluster(cluster_list, list_ptr-
>points_to[i]);
                  }
                }
              }

              list_ptr = list_ptr->next;
    }

}

void print_by_distance_new_cluster_list(struct cluster_info **cluster_list, char
*filename)
{
    struct cluster_info *list_head, *list_ptr;
    int i;
    char space_buf[60] = {0};
    char print_buf[500] = {0};
    char temp_buf[100] = {0};
    FILE *output;

    list_head = *cluster_list;
    list_ptr = list_head;

    output = fopen(filename, "a");
    if(!output)
    {
            printf("Failed to open %s for writing\n", filename);
```

```
            return;
    }

    while(list_ptr)
    {
            if(list_ptr->pointed_to_by == -1)
            {
              strcpy(space_buf, "");
              strcpy(print_buf, "");
              strcpy(temp_buf, "");
              for(i=0; i<list_ptr->distance/10; i++)
                strcat(space_buf, " ");
              sprintf(print_buf, "\n-------------------------------------------------\n%s--
img_num %d\n", space_buf, list_ptr->img_num);
              if(list_ptr->distance > 0)
                sprintf(temp_buf, "%s%f\n", space_buf, list_ptr->distance);
              else
                sprintf(temp_buf, "\n");
              strcat(print_buf, temp_buf);
              printf("Printing %s\n", print_buf);
              fprintf(output, print_buf);
              print_count++;

              if(list_ptr->num_points_to)
              {
                for(i=0; i< list_ptr->num_points_to; i++)
                {
                  print_by_distance_current_cluster(cluster_list, list_ptr->points_to[i],
filename);
                }
              }
```

```
            }

            list_ptr = list_ptr->next;
    }

    fclose(output);
}

void print_html_cluster_list(struct cluster_info **cluster_list, char *filename)
{
    struct cluster_info *list_head, *list_ptr;
    int i, err_num;
    char print_buf[200] = {0};
    char error_buf[100] = {0};
    char short_name[30] = {0};
    char jpg_name[30] = {0};
    FILE *output;

    exec sql begin declare section;
    char name_buf[100] = {0};
    int usable_img_num;
    exec sql end declare section;

    exec sql include sqlca;

    list_head = *cluster_list;
    list_ptr = list_head;

/*
    output = fopen(filename, "a");
    if(!output)
```

```
    {
            printf("Failed to open %s for writing\n", filename);
            return;
    }

*/
    while(list_ptr)
    {
             if(list_ptr->pointed_to_by == -1)
             {
               usable_img_num = list_ptr->img_num%1000;
               if(list_ptr->num_points_to)
               {
                 for(i=0; i< list_ptr->num_points_to; i++)
                 {
                   print_html_current_cluster(cluster_list, list_ptr->points_to[i],
filename);
                 }
               }
/*
               if(list_ptr->distance)
               {
                 exec sql select image_name into :name_buf from images where
image_num = :usable_img_num;
                 sprintf(error_buf, "Getting image name for number %d",
usable_img_num);
                 err_num = check_db_error(sqlca);
                 check_error(err_num, error_buf);
                 get_short_name(name_buf, short_name);
                 get_jpg_name(short_name, jpg_name);
```

```
                    strcpy(print_buf, "");
                    sprintf(print_buf, "\n<BR><HR><BR>\n<IMG
SRC=\"/tmp/masked_jpg/%s\">\n<BR>%s Distance: %f", jpg_name, short_name,
list_ptr->distance) ;
                    fprintf(output, print_buf);
                  }

                  if(list_ptr->num_points_to)
                  {
                    for(i=0; i< list_ptr->num_points_to; i++)
                    {
                      print_html_current_cluster(cluster_list, list_ptr->points_to[i],
filename);
                    }
                  }
*/
                }

                list_ptr = list_ptr->next;
    }

//    fclose(output);
}


void display_by_distance_current_cluster(struct cluster_info **cluster_list,
                                         int cluster_num)
{
  struct cluster_info *list_head, *list_ptr;
  char dash_buf[4] = {0};
  char space_buf[100] = {0};
```

```
char error_buf[100] = {0};
char short_name[40] = {0};
char jpg_name[40] = {0};
int i, err_num;

exec sql begin declare section;
int target_img_num;
char name_buf[100] = {0};
exec sql end declare section;

// First, find the cluster
list_head = *cluster_list;
list_ptr = list_head;

while(list_ptr)
 {
   if(list_ptr->img_num == cluster_num)
            {
              target_img_num = list_ptr->img_num%1000;

              if(list_ptr->num_points_to)
                strcpy(dash_buf, "--");

              if(list_ptr->pointed_to_by != -1)
              {
                display_count++;
                strcpy(space_buf, "");
                for(i=0; i<list_ptr->distance/10; i++)
                          strcat(space_buf, " ");
                if(target_img_num == list_ptr->img_num)
                {
```

```
                    exec sql select image_name into :name_buf from images where
image_num = :target_img_num;
                    err_num = check_db_error(sqlca);
                    sprintf(error_buf, "Getting image name for number %d",
target_img_num);
                    check_error(err_num, error_buf);
                    get_short_name(name_buf, short_name);
                    get_jpg_name(short_name, jpg_name);
                  //}
                  //printf("%s%s%d--%s\n", space_buf, dash_buf, list_ptr->img_num,
jpg_name);
                  printf("%s\n", space_buf, dash_buf, list_ptr->img_num, jpg_name);
                  }
                  if(list_ptr->distance > 0)
                    printf("%s%f\n", space_buf, list_ptr->distance);
                }
                if(list_ptr->num_points_to)
                {
                  for(i=0; i<list_ptr->num_points_to; i++)
                  {
                    display_by_distance_current_cluster(cluster_list, list_ptr-
>points_to[i]);
                  }
                }

                break;
              }

              list_ptr = list_ptr->next;
  }
}
```

```
void print_by_distance_current_cluster(struct cluster_info **cluster_list,
                                       int cluster_num, char *filename)
{
   struct cluster_info *list_head, *list_ptr;
   char dash_buf[4] = {0};
   char space_buf[60] = {0};
   char temp_buf[100] = {0};
   char print_buf[500] = {0};
   int i;
   FILE *output;

   output = fopen(filename, "a");
   if(!output)
   {
            printf("Failed to open %s for appending\n", filename);
            return;
   }

   // First, find the cluster
   list_head = *cluster_list;
   list_ptr = list_head;

   while(list_ptr)
    {
      if(list_ptr->img_num == cluster_num)
            {
              strcpy(dash_buf, "");
              strcpy(space_buf, "");
              strcpy(print_buf, "");
```

```
                strcpy(temp_buf, "");
                if(list_ptr->num_points_to)
                  strcpy(dash_buf, "--");

                if(list_ptr->pointed_to_by != -1)
                {
                  for(i=0; i<list_ptr->distance/10; i++)
                              strcat(space_buf, " ");
                  print_count++;
                  sprintf(print_buf, "%s%s%d--\n", space_buf, dash_buf, list_ptr-
>img_num);
                  if(list_ptr->distance > 0)
                    sprintf(temp_buf, "%s%f\n", space_buf, list_ptr->distance);
                  strcat(print_buf, temp_buf);
                  printf("Printing %s\n", print_buf);
                  fprintf(output, print_buf);
                }
                if(list_ptr->num_points_to)
                {
                  fclose(output);
                  for(i=0; i<list_ptr->num_points_to; i++)
                  {
                    print_by_distance_current_cluster(cluster_list, list_ptr->points_to[i],
filename);
                  }
                }

                break;
              }

              list_ptr = list_ptr->next;
```

```
    }
    if(output)
      fclose(output);
}

void print_html_current_cluster(struct cluster_info **cluster_list, int cluster_num, char
*filename)
{
    struct cluster_info *list_head, *list_ptr;
    char print_buf[500] = {0};
    char temp_buf[100] = {0};
    char short_name[30] = {0};
    char jpg_name[30] = {0};
    char error_buf[100] = {0};
    int i, err_num;
    FILE *output;

    exec sql begin declare section;
    char name_buf[100] = {0};
    int target_img_num;
    exec sql end declare section;

    exec sql include sqlca;

    output = fopen(filename, "a");
    if(!output)
    {
                printf("Failed to open %s for appending\n", filename);
                return;
    }
```

```
   // First, find the cluster
   list_head = *cluster_list;
   list_ptr = list_head;

   while(list_ptr)
    {
                target_img_num = list_ptr->img_num;
      if(target_img_num == cluster_num)
                {
                  strcpy(print_buf, "");

                  if(target_img_num == target_img_num%1000)
                  {
                    exec sql select image_name into :name_buf from images where
image_num = :target_img_num;
                    err_num = check_db_error(sqlca);
                    sprintf(error_buf, "Getting image name for number %d",
target_img_num);
                    check_error(err_num, error_buf);
                    get_short_name(name_buf, short_name);
                    get_jpg_name(short_name, jpg_name);

                    sprintf(print_buf, "<IMG SRC =\"/tmp/masked_jpg/%s\">\n",
jpg_name);
                    if(list_ptr->distance > 0)
                      sprintf(temp_buf, "<BR>%s Distance: %f\n", short_name, list_ptr-
>distance);
                    strcat(print_buf, temp_buf);
                    fprintf(output, print_buf);
                  }
```

```
                if(list_ptr->num_points_to)
                {
                  fclose(output);
                  for(i=0; i<list_ptr->num_points_to; i++)
                  {
                    print_by_distance_current_cluster(cluster_list, list_ptr->points_to[i],
filename);
                  }
                }

                break;
              }

              list_ptr = list_ptr->next;
  }
  if(output)
    fclose(output);
}
```

BIBLIOGRAPHY

Abate, A. F., Nappi, M., Tortora, G., and Tucci, M. (1999). IME: an image management environment with content-based access. *Image and Vision Computing* 17, 967-980.

Abbasi, S., Mokhtarian, F., and Kittler, J. (2000). Enhancing CSS-based shape retrieval for objects with shallow concavities. *Image and Vision Computing* 18, 199-211.

Adjeroh, D. A., and Lee, M. C. (2000). An occupancy model for image retrieval and similarity evaluation. *IEEE Transactions on Image Processing* 9:(1), 120-131.

Chang, C. C., Hwang, S. M., and Buehrer, D. J. (1991). A shape recognition scheme based on relative distances of feature points from the centroid. *Pattern Recognition* 24:(11). 1053-1063.

Chen, J.-Y., Bouman, C. A. , and Dalton, J. C. (2000). Hierarchical browsing and search of large image databases. *IEEE Transactions on Image Processing* 9:(3) No. 3. (March 2000).

Cinque, L. and Lombardi, L. (1995). Shape description and recognition by a multi-resolution approach. *Image and Vision Computing* 13:(8), 599-607.

Gdalyahu, Y. and Weinshall, D. (1999). Flexible syntactic matching of curves and its application to automatic hierarchical classification of silhouettes. *IEEE Transactions of Pattern Analysis and Machine Intelligence* 21:(12), 1312 – 1328.

Huet, B. and Hancock, E. R. (1998). Relational histograms for shape indexing. *6th International Conference on Computer Vision.* 563-569.

Huet, B. and Hancock, E. R. (1999*).* Line pattern retrieval using relational histograms. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21:(12). 1363-1370.

Huet, B., Cross, A. D. J., and Hancock. E. R. (1999). Shape retrieval by inexact graph matching. *IEEE International Conference on Multimedia Computing and Systems* Vol 1. 772-776.

Kian-Lee T., Ooi B. C., and Thiang L. F. (2000). Indexing shapes in image databases using the centroid-radii model. *Data and Knowledge Engineering* 32, 271-289.

Kim, W. Y. and Kim, Y. S. (2000). A region-based shape descriptor using Zernike moments. *Signal Processing – Image Communication* 16:(1-2), 95-102.

Mayya, N. and Rajan, V. T. (1995). An efficient shape representation scheme using Voronois skeletons. *Pattern Recognition Letters* 16, 147-160.

Milios, E. and Petrakis, E. G. M. (2000). Shape retrieval based on dynamic programming. *IEEE Transactions on Image Processing* 5:(1), 141-147.

Santini, S. and Jain, R. (2000). Integrated browsing and querying for image databases. *IEEE Multimedia* 7:(3)26-39.

Smeulders, A. W. M., Worring, M., Santini, S., Gupta, A., and Jain, R. (2000). Content-based image retrieval at the end of the early years. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22:(12).

Sonka, M., Hlavac, V. and Boyle, R. (1999). *Image processing, analysis, and machine vision* (pp. 95-96). Pacific Grove: PWS Publishing.

Vailaya, A., Figueiredo, M. A. T., Jain, A. K., and Zhang, Hong-Jiang (2001). Image classification for content-based indexing. *IEEE Transactions on Image Processing* 10:(1), 117 – 130.

Weber, D. M. and Casasent, D. P. (2001). Quadratic Gabor filters for object detection. *IEEE Transactions on Image Processing,* 10:(2), 218 – 230.

Wu, J. K., and Narasimhalu, A. D. (1998). Fuzzy content-based retrieval in image databases. *Information Processing & Management,* 34:(5), 513-534.

VITA

Cynthia Gaye Huyser was born in Detroit, Michigan, on October 16, 1959, the daughter of Gloria N. and Willis C. Huyser. After receiving her diploma from Lamphere High School in Madison Heights, Michigan, in 1977, she entered Tri-State University in Angola, Indiana, where she completed her Bachelor of Arts degree in English May of 1981. She began her computer studies in August, 1996, when she entered Austin Community College in Austin, Texas, and continued them at Southwest Texas State University in January of 1999. She was employed by the City of Austin's electric utility for 15 years, where she became the first female power plant control board operator and the first female power plant operations supervisor. From March 1999 to June 2000, she was employed by Travis County as a PC/Lan Administrator, and from June 2000 through October 2002 worked as a Research/Engineering Scientist Assistant for the University of Texas' Applied Research Laboratories in the Space and Geophysics laboratory. Since November 2002 she has been employed as an assistant engineer with J3S, Inc., a consulting and research company.

Permanent Address: 603 South 3rd Street
Austin, Texas 78704

This thesis was typed by Cynthia G. Huyser.