Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

BINARY IMAGE DECOMPOSITION AND COMPRESSION USING MATHEMATICAL MORPHOLOGY FOR OBJECT RECOGNITION

by
Venu Gopal Gogusetti

A Thesis
Submitted to the Faculty Institute of Technology
In Partial Fulfillment of Requirements for Degree of
Master of Science

Department of Computer and Information Science

January 1992

ABSTRACT

Title of Thesis: Binary Image Decomposition and Compression Using Mathematical Morphology for Object Recognition.

Name of Candidate: Venu Gopal Gogusetti
Master of Science in Computer & Information Sciences, 1991

Thesis Directed by: Dr. Frank Y. Shih
Assistant Professor
Department of Computer & Information Sciences
New Jersey Institute of Technology, Newark, N J 07102

Image segmentation has been studied for several years. There are several segmentation techniques which are fast and effective but all of them are lacking the property of invariency to shift, rotation and sizing. In this study a new process is introduced which overcomes the shift, size and rotation variance and the compressed data can be used for object recognition.

Euclidean distance measurement is used in the compression process which is rotation invarient but is expensive in terms of time. Eucledean distance transformation is calculated using optimal double two scan algorithm with gray scale morphology, a new method developed by Dr. Shih and Mr. Wu. Mathematical morphology provides an effective tool for image analysis. Many parallel image computers support basic morphological operations. A new image segmentation technique is presented, which is more usefull in object recognition as it is invarient to shif, orientation and is proportional to sized images.

APPROVAL SHEET

Title of Thesis: **Binary Image Decomposition and Compression Using Mathematical Morphology for Object Recognition**

Name of Candidate: **Venu Gopal Gogusetti**
Master of Science in Computer and Information Sciences

Thesis and Abstract Approved: 12/20/91

Dr. Frank Y. Shih Date

Assistant Professor
Department of Computer and Information Sciences
New Jersey Institute of Technology
Newark, New Jersey 07102

BIOGRAPHICAL SKETCH

Author: Venu Gopal Gogusetti

Degree: Master of Science in Computer Science

Date: January 1992

Date of Birth:

Date of Birth:

Undergraduate and Graduate Education

- Master of Science in Computer Science
  New Jersey Institute of Technology
  Newark, New Jersey, 1992

- Master of Science in GE,
  College of Engineering
  Andhra University, India, 1986

- Bachelor of Science in Computer Science,
  College of Science,
  Oamania University, India, 1984

Major: Computer Science

This thesis work is dedicated to
my parents,
my wife Sharmila
and friends

ACKNOWLEDGEMENT

I would like to take this opportunity to express my sincere gratitude to my advisor, Dr. Frank Y. Shih, for his remarkable guidance, moral support and kindness throughout the course of this work.

I gratefully acknowledge the valuable discussions with my colleague Mr. Kumar Chebrolu.

I am thankful to my wife Sharmila, all friends and family members for supporting and encouraging me to complete my work successfully.

TABLE OFCONTENTS

Chapter 1 Introduction

In image compression or enhancement, the desired output is a picture, an approximation or an improved version of the input image. To generate description of any picture, a picture has to be segmented in to parts which are simple and unique. The unique property is thus used in identifying the feature or the object. Segmentation is basically pixel classification. The image is segmented in to subsets by assigning individual pixell classes. A good segmentation algorithm should allow shape description.

1.1 Compression

The necessity of compressing an image is the storage requirement. For a two dimensional binary image of NxM size needs N*M bits of array. Image transmitted over communication links and stored on the disks in computers generally takes significant space and time. Different techniques have been used to reduce this storage requirements. But most of them lack the property of producing the similar data if there is any shift or rotation in the image. There are several compression algorithms available for binary image processing. In this study a new technique is proposed and studied for image compression which produces the similar data for the same image with shift or rotation of the object and a proportional data for sized image.

1.1.1 Evaluation of different Compression Schemes

Runlength coding is one of the most practical and simplest methods for binary images. Each row of a picture consists of a sequence of maximal runs of points such that the points in each each run will have the same value. Thus the row is completely determined. As far as binary image is concerned, by specifying the value of the first runin the row and lengths of all runs, because the values are alternate.

In chain coding, each chain is composed of line segments connecting adjacent pixels in horizontal, vertical or diagonal directions. A closed boundary is chain coded from an arbitrary starting point, the initial border point coordinate is stored and the boarder around the region is then followed back to the first point. And this is repeated for each separate region. A region can be defined as a set of all connected points.

In MAT representation, with each point P of the image the set upright squares of oddsized length centered at P is associated. The largest such square that is contained in the object is called maximal block. It is easy to see that if a set is specified with centers P, and radii r(P) of the maximal blocks, the image is completely determined.

In the quad trees representation case, maximal blocks can be of any size and at any position, they are analogous to runs in one dimensional case. If the picture has all one value, the root node is labelled with that value. Otherwise four descendent nodes to the rootnode are added representing four quadrants of the image. This process is repeated until no further nodes are generated. This scheme is conceptually very clear and well structured. But it produces very different tree for a slight shift in the object making it useless for image comparision.

Axial representations of shape is another method, where an object is represented by a spine and structuring element. if the spine or axis is dilated with the structring element which is generally a disk. The disadvantage is that same image can be represented by several spines and structuring elements. Which makes it useless for object recognition because of lack of uniqueness. And also it is difficult to represent splines in the equation forms, and complicated calculations involved in reproducing the spline.

1.2 Morphological Shape Decomposition

Shape description is very important issue in image pattern analysis and recognition. It provides description of objects according to their shape which can be used for pictorial pattern recognition. Shape description can also provide techniques suitable for image coding that permit image transmission at low bit rates and can be stored in relatively small disk space. So far all the segmentation techniques are effective in reduction of size, but for object recognition or imaghe comparision. Morphological decomposition of an object can be dodne as a union of disks. The principle of this segmentation technique is to calculate the radius of the largest disc, which fits in the object. The location of the radius and the center of the disk in the image is noted and the then the disk is subtracted from the image. This process is repeated until the remaining image is blank or background. The result of this segmentation technique is a set of coordinates and radii.

The whole procedure can be described in a mathematical way by morphology. Let $G(R^n)$ be the class of topological open sets on Euclidean space R^n. Let $X \in G$ be an open non empty set containing no half space. The spacial occupancy of this set is the shape to be decomposed. The problem is to find a set of open sets { X1, ..., Xn} whose union is X:

$$X = \bigcup_{r>0} S_r(X) \oplus rB$$

The image comparison and recognition part will use the resulting co-ordinates and radii. Two objects are said to be same when the distance between disks and the radii of the first image are proportional to the distance between disks and the radii of the second image. This eliminates the variance in size ane orientation.

But to calculate the radius and the co-ordinates of the largest disk fits in the image, the eucledean distance transformation is used. The eucledean distance transformation was a time consuming and expensive process until Dr. Shih and Wu came out with a new algorithm called "Double two scan algorithm" which is independent of the object size and a non recursive process. Eucledean distance transformation is very sensitive to noise. To eliminate noise morphological functions dilation, erosion, closing and opening are used.

Chapter 2

XView User Interface and Mathematical Morphology

2.1 XView

XView is a user interface toolkit to support interactive, graphics based application running under the X Window System. XView stands for X Window-System-based Visual/ Integrated Environmant for Work Stations. This toolkit is developed by Sun Microsystems, Inc. and is derived from the earlier toolkits for the SunView Windowing system. XView provides a set of pre-built, user interface objects like canvases, scrollbars, menus and control panels. The appearence and functionality of these objects follow thw OPEN LOOK Graphical User Interface (GUI) specifications. OPEN LOOK provides user with a simple, consistant and efficient interface for performing tasks within an application. XView is based on Xlib, the lowest level of X w\Window System available to the programmer. X Window System controls a bit mapped display in which every pixel is individually controllable. This allows displaying pictures very easyly in addition to text. X takes user input from a pointer, which is useually a mouse.

2.2 Object-oriented Programming

XView is an object-oriented toolkit. Objects are represented in a class hierarchy. Objects are opaque data types. Objects have attributes which can be set via message passing functions. Objects may have callback procedures that are triggered by events.

XView defines classes of objects in a tree heirarchy. Frame is a subclass of the more general class window, is a subclass of drawable, whcih is a user interface object classes, is a subclass of the Generic Object class. Figure 2.1 shows the Xview class heirarchy and the relationships between classes. Each class has identifying features that make it unique from other classes or packages.

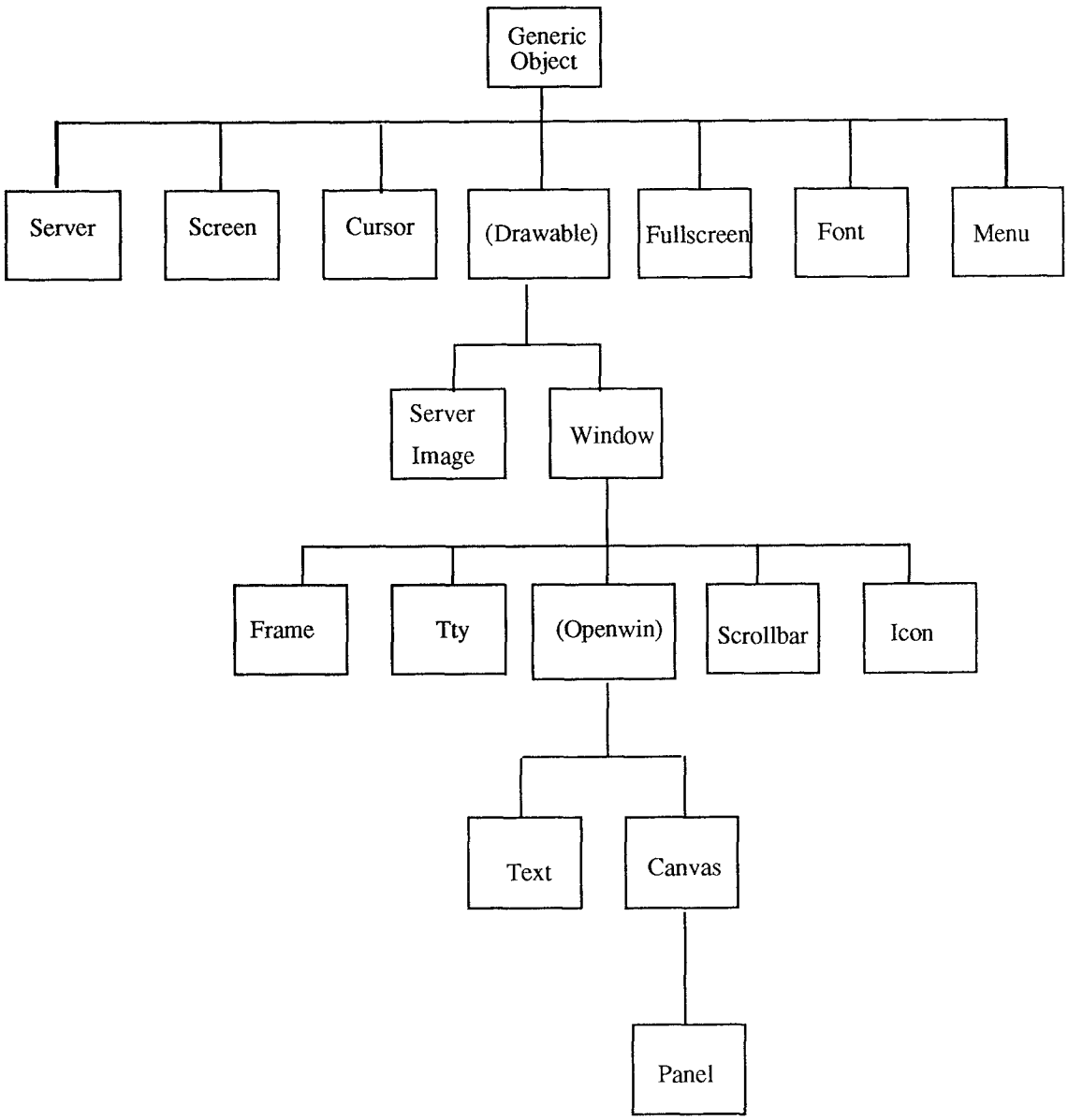


Fig. 2.1 XView Class heirarchy.

2.3 XView and Morphology

Xview has been efficiently used in this work which makes morphology so simple and user friendly in image processing, that even if the user does not know anything about morphology, he can play around by loading image and doing some processing with the morphological operators by just clicking the buttons.

2.4 The Menu

The menu displayed on the screen after running the program 'morphology' has Load Image, Disp Orig Image, EDTransform, Dilation, Erosion, Closing, Opening, Reverse, Enlarge, Compress, Reconstruct, Save Image, Clear Image and Quit. Of all these buttons the Dilation, Erosion, Closing and Opening has sub menu buttons called menu action items for selecting appropriate structuring elements.

2.4.1 Load Image

If Load Image button is activated by clicking a sub window is popped up for taking the input file name of the image. This window will display the format of input filename and will display error message if it finds difficult to open that file, and the window is cleared (killed) if the file is opened with out any problem.

2.4.2 Display Original Image

The second button is to display the original image. This button provides the user with the originally loaded image from the file so that if the user makes any mistake in using the morphological operators he can come back to step one by clicking this button. This button frees all the processed images it has in the memory.

2.4.3 Euclidean Distance Transform

The next button ED Transform is to apply euclidean distance transformation to the image. This button uses the most efficient back propagation double two scan algorithm and

calculates the euclidean distance.

2.4.4 Morphological Operations

The next four buttons are the basic morphological operators called dilation, erosion, closing and opening. All these four buttons have a sub menu to select proper structuring element to do the operation. The sub-menu will popup as one of those buttons is activated by clicking with the mouse.

2.4.5 Reversing Image

The button reverse can be clicked to transform the image background (non object) to foreground (object) and foreground (object) to back ground (non object). This button is added because in some scanned images the object has zero value and the background has one value, so that user can load the image as it is and perform the regular operations by reversing the image.

2.4.5 Enlarge

The button enlarge can be used to magnify the image digitally. If this button is activated the image size becomes X4 i.e. the number of rows and cloumns are doubled. As the limitation to display is 512x512 pixels the image is enlarged first and only the first 512 rows and colums are displayed if the size of the image is larger than 512x512 after enlargement.

2.4.6 Enlarge

The button compress is used to decompose the image in to set of disks, where the union of disks will produce the original image back. The segmentation procedure uses the euclidean transform algorithm to calculate the radius of the largest disk which fits in the object and subtrcts the disk from the image. The display function is used after subtracting every disk from the image so that user can see how the algorithm is working on that

particular image. The compressed data, in other words the coordinates and radii (squares of radii used so that thay can be represented by integers) of these disks are written in a file called 'RES'.

2.4.7 Reconstruction

The reconstruct button is used to reconstruct the image, where the data is read from the file RES which is created by the compress program. This data file consists the size of the image and coordinates and radii (squares) of the disks, where the union of the disks produce the image.

2.4.8 Save Image

The save image button saves the processed image which is displayed. This button pops a subwindow where it waits for a filename to be entered. The size of the image is concatinated automatically by the program so that the user need not remember the image size after processing.

2.4.9 Clear and Quit Buttons

The clear image button clears the display screen and returns a blank display area. Where as the last button, quit is used to quit the program.

This XView interface makes the whole program so interactive and displays the resulting image so efficiently, that the user need not type much and image processing using morphology is made so simple.

Chapter 3 Mathematical Morphology

3.1 Morphology

An Image can be represented by a set of pixels. The morphological is a process where two sets are at work, or it can also be said that it is a process where two images are interacted. The image being processed is referred to as the active image and the other image being a kernel is referred as the structuring element. Each structuring element has pre-designed shape, which can be thought as a probe or filter of the active image. Active image can be modified or processed using various structuring elements. Mathematical morphology provides an approach to the processing of digital images which is based on shape. Appropriately used, mathematical morphological operations tend to simplify image data preserving their essential shape characteristics and eliminating irrelevancies. As the identification of objects, object features, and assembly defects correlate directly with shape, it becomes apparent that the natural processing approach to deal with the machine vision recognition process and the visually guided robot problem is mathematical morphology.

3.2 Dilation and Erosion

The language of mathematical morphology is that of set theory. Sets in mathematical morphology represent the shapes which are manifested on binary or gray tone images. The set of all the black pixels in a black and white image, (a binary image)

constitutes a complete description of the binary image. Sets in Euclidean 2-space denote foreground regions in binary images. Sets in Euclidean 3-space may denote time varying binary imagery or static gray scale imagery am binary solids. Mathematical morphological transformations apply to sets of any dimensions, those like Euclidean N- space, or those like its discrete or digitized equivalent.

Those points in a set being morphologically transformed are considered as the selected points and those in the complement set are considered as not selected. Hence, morphology from this point of view is binary morphology. We begin our discussion with the binary morphological operations of dilation and erosion.

3.2.1 Dilation

A Dilation is the morphological transformation which combines two sets using vector addition of set elements. If A and B are sets in N- space(E^N) with elements 'a' and 'b', respectively, a=(a_1, ...,a_N) and b=(b_1,. . . . . ,b_N) being N-tuples of element coordinates, then the dilation of A by B is the set of all possible vector sums of pairs of elements, one coming from A and one coming from B.

Definition: Let A and B be subsets of E^N . The dilation of A by B is denoted by $A \oplus B$ and is defined by

$$A \oplus B = \{ c \in E^N \mid c = a + b \text{ for some } a \in A \text{ and } b \in B \}$$

Dialation as a set theoretic operation was proposed by H.Minkowski [20] to characterize integral measures of certain open sets. Dilation is an image processing operation used to smoothout the image reducing noise.

3.2.3 Erosion

Erosion is the morphological duel to the dilation. It is the morphological transformation which combines two sets using the vector subtraction of set elements. If A and B are sets in Euclidean N-space, Then the erosion of A by B is the set of all elements x for which $x + b \in A$ for every $b \in A$. Erosion is also said to be as shrink or reduce instead.

$$A \ominus B = \{ x \in E^N \mid x + b \in A \text{ for every } b \in A \}$$

3.2.4 Opening and Closing

Dilations and erosions are employed in pairs, either dilation of an image followed by the erosion of dilated result, or the image erosion followed by dilation. In either case, the result of iteratively applied dilations and erosions is an elimination of specific image details smaller than the structuring element without the global geometric distortion of unsuppressed features. An opening of an image with a disk structuring element smoothes the contour, break narrow isthmuses and eliminates narrow islands and sharp peaks or capes. A closing of an image with a disk structuring element smooths the contours, fuses narrow breaks and long thin gulfs, eliminates small holes and fills gaps on the contours. Of particular significance is the fact that image transformations employing iteratively applied dilations and erosions are idempotent. That is their re-application effects no further changes to the previously transformed result.

Opening of image B by structuring element K is denoted by BoK and is defined by

$$BoK = (B \ominus K) \oplus K .$$

Closing of image B by structuring element K is denoted by B • K and is defined by

$$B \bullet K = (B \oplus K) \ominus K .$$

If B is unchanged by opening it with K, then B is said to be open with respect to K. If B is unchanged by closing it with K, then B is said to be closed with respect to K.

3.3 Gray Scale Morphology

The binary morphological operations of dilation, erosion, opening and closing are naturally extended to gray scale imagery by the use of a minimum and maximum operators. The gray scale

dilations is defined as the durface of the dilation with umbras. Dilation can be computed in terms of maximum operation and a set of addition operations. A similar plan is followed for erosion which can be evaluated in terms of a minimum operation and a set of subtraction operations.

3.3.1 Gray scale dilation:

Let $:f;F \rightarrow E$ and $k;K \rightarrow E$

Gray scale dilation can be defined as $f \oplus_g k;F \oplus_g K \rightarrow E$

can be computed by

$$f \oplus_g k\,(x) \quad = \max \{ \text{ f (f(x-z) + k(z) } \} \text{ for all } z \in K \text{ and } x - z \in F$$

3.3.2 Gray scale erosion:

Let $:f;F \rightarrow E$ and $k;K \rightarrow E$

Gray scale erosion can be defined as $f \quad \ominus_g K \rightarrow E;\ F \qquad \ominus_g K \rightarrow E$

can be computed by

$$f \ominus_g k\,(x) \quad = \min \{ \text{ f (f(x+z) - k(z) } \} \text{ for all } z \in K \text{ and } x + z \in F$$

3.4 Back Propagation Dilation and Erosion

The new type of morphological operators introduced by Shih[xx] are back propagation dilation and back propagation erosion. Let N(O) be the set of the neighbors that preceede O and itself in a scanning sequence of the picture within a window of a structuring element. The back propagation dilation of 'f' by 'k' is denoted by f $\circledoplus$ k is defined as

$$(f \circledoplus k)(x,y) = \max\{[f \circledoplus k\ (x-m,y-n)] + k(n,m)\},$$

$$\text{for all } (m,n) \in K \text{ and } (x-m,y-n) \in N \cap F.$$

The back propagation erosion of 'f' by 'k' is denoted by f $\circledcirc$ k is defined as

$$(f \circledcirc k)(x,y) = \min\{[f \circledcirc k\ (x+m,y+n)] + k(n,m)\},$$

$$\text{for all } (m,n) \in K \text{ and } (x+m,y+n) \in N \cap F.$$

Since the back propagated operations adopts the results of the preceeding scanned neighbors to be involved in its computation, its output inherently depends on the image scanning sequence.

Chapter 4

Distance Transformation

A distance transformation converts a digital binary image which consists of object (foreground) and non-object (background) pixels, in to a gray level image in which all object pixels have a value corresponding to the minimum distance to the background. The distance computation is in fact a global computation, often prohibitively costly. There fore a decomposition strategy using only local operations is useful. There are six types of distance measures, they are City-block, Chessboard, Euclidean, Octogonal, Chamfer3-4 and Chamfer5-7-11. Three types of distance measures in image processing are often used are Euclidean, Citi-block and Chessboard.

Citi-block and Chessboard distances are easy to compute, and they can be recursively accumulated by cosidering only a small neighborhood. The algorithms for citiblock and chessboard are widely developed. But the biggest disadvantage of these distance transformation algorithms is that they are very sensitive to the orientation of the object.

4.1 Euclidean Distance Transform

Euclidean distance transform is a global operation and a very complex recursive algorithm. In euclidean distance transformation each object pixcell is calculated to the nearest background to the floating level. Euclidean distance transformation is very useful in object recognition and inspection because of the metric accuracy and rotation invarience.

4.2 Double Two-Scan Algorithm

With the new algorithm developed by Dr. Shih and Wu, using gray scale morphology we can compute euclidean distance by doing four erosions with small (3X3) structuring elements, called "Optimal Double Two-Scan Algorithm " which is indipendent of the object size. This algorithm employs only four back-propagation grayscale morphological erosions to implement euclidean distance transformation in order to avoid time consuming iterations. k1 and g1(l) are the raster scan structuring elements, and they are used in the image scanning left-to-right and top-to-bottom. The k2 and g2(l) are the inversed raster scan structuring elements and they are used in the image scanning right-to-left and bottom- to-top.

The structuring elements used in the first two steps are to acheive the chessboard distance transformation. The k1 and k2 are given as follows:

$$k1 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 0 & x \\ x & x & x \end{bmatrix}$$

$$k2 = \begin{bmatrix} x & x & x \\ x & 0 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

The structuring elements g1(l) and g2(l) used in the third and fourth steps are functions of the parameter l, that is

$$g(l)1 = \begin{bmatrix} -(4l-2) & -(2l-1) & -(4l-2) \\ -(2l-1) & 0 & x \\ x & x & x \end{bmatrix}$$

$$g(l)2 = \begin{bmatrix} x & x & x \\ x & 0 & -(2l-1) \\ -(4l-2) & -(2l-1) & -(4l-2) \end{bmatrix}$$

4.2.1 Double Two Scan Algorithm in a nut shell

1 $D = f \ominus k1$

2 $E = f \ominus k2$

3 For $i = 1$, length, 1 do

For $j = 1$, length, 1 do

$l = D\ ij$

$Rij = f\ ij \ominus g1(l)$

4 For i = length, 1, 1 do

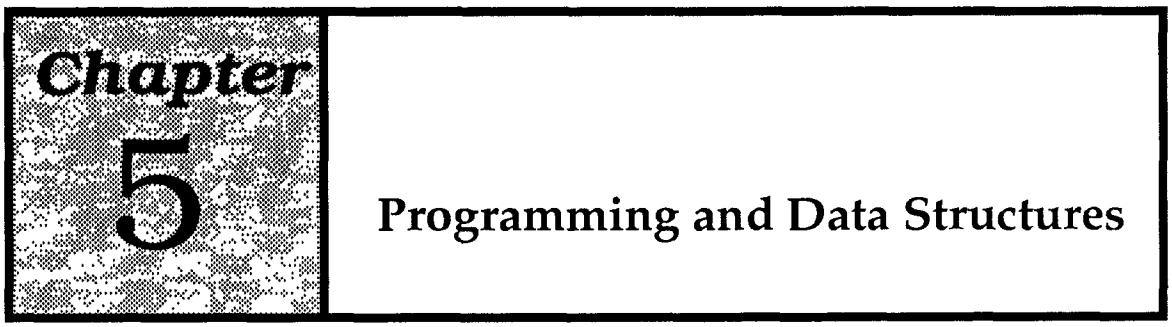
For j = length, 1, 1 do

$l = E\ ij$

$Q\ ij = Rij \ominus g1(l)$

if ($Q\ ij > 1$) $Q\ ij = \sqrt{Q\ ij}$

In the third step, the image D is used as the index value l to obtain g1(l). In the fourth step the image E is used as the index value l to obtain g2(l) . Where after the end of fourth step we get the euclidean distance in Q fot the image f. The complexity of the program is of the order of P, where P is the size of the image.



This Morphological Toolkit is developed using 'C' language and inter faced with OPENLOOK/ XView Graphical User Interface. Xlib is used for image displaying. An image can be read as on e byte charecter form for a graylevel image or one bit per pixel for for binary image. In this case an intiger is scanned for the pixel value, which makes things lot easier, but the image file occupies excess disk space.

An image is created as structure which contains an array of intigers of 512 by 512 size, and two more integers row and column for indicating the actual size of the image. This row and column is passed along with the structure to limit the un-necessary oprations beyond the input image size.

Some cases long integer has been declared because of the square of the distance. To have a backup of the original image for the button 'Dispay Original Image', a copy of the image is made to the mallocated structure pointed by Orig. when this button is clicked the activated sub procedure frees the image_F, the one which is processed by all procedures, and Image_C, the swapping pointer to the structure. Then the original image is copied and here goes to the step one.

The morpphological operators Dilation, Erosion, Opening and Closing are implemented with simple null matrix of Mx1 or NxN where N is an odd sized integer.

These procedures have only one limitation, i.e. no of calling menu buttons, and action procedures. These morphological operators can be used to eliminate noise, fill holes and remove sharp extensions.

The Euclidean Distance Transform is performed using Double Two-Scan backpropagation erosion with simple 3x3 structurin element, which takes only four scans to calculate the squares of the euclidean distance. As the squares of some distances like 500 go beyond the maxint value of the system, long intiger has been declared.

Enlarge is a simple function where the size of the image is doubled in both directions row and column from r to R and c to C where R (C) = 2r (2c) and the integer division of the R (C) gives the value of corresponding r(c). Thus digital enlargement gives a good image. There is alimitation in this package, ofcourse the maximum size, can be increased, but set 512x512. So even if the R(C) value goes beyond 512 only 512x512 image is selected for further processing.

Reversing an image is another simple algorithm, particularly for a binary image. All we have to do is just change the value from 1 to 0 and vice versa. This can also be extende for a gray scale image (or even to the color image), just by subtracting the gray values from the maximum gray value and assigning to that pixel. This function needs one scan of the image.

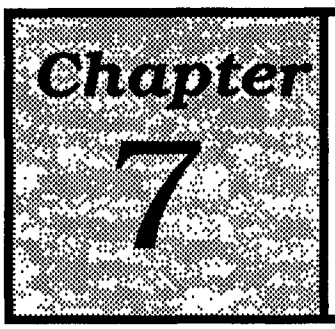
Compress is the most time consuming algorithm in this toolkit, as it uses the Euclidean Disatance Transform several times, depending on the complexity of the image. In this algorithm. The word segmentation or decomposition may well suit, instead of compress, but most of the compact object cases the resulting data of compression is simple and unique. The resulting coordinates and square of the radius is written in a file named RES. This file will have some integers, first line will have two integers representing the row and column size of the image, followed by three integers representing the coordinates and the square of the radii.

The reconstruction program uses the file RES created by the compression algorithm. First it allocates the memory for the image and then it creates disks with the radii

and coordinates. This algorithm is a simple one where disks are created and or'd with the resulting image. This kind of reconstruction produces the same original image without any loss of information, if all disks with radius value one are considered.

Save image is the program where at any stage the processed image can be saved by using this button. A sub window is popped to enter the filename. This window will also display that the file is saved under such and such name, and The program concatinates the column and row values to the file name, so the user need not remember the size of image. All the user need to enter is simple name.

Quit leaves the program destroying the window and closing all files. Some results of the processing are in the appendix B.



Bibliography

[1] Frank Y.Shih, and Hong Wu, " Optimization on Euclidean Distance Transformation Using Gray Scale Morphology"

[2] Ionnis Pitas and Anastasios N. Venetsanopoulos, " Morphological Shape Decomposition", IEEE transactions on PAMI, VOl 12, No. 1, January 1990.

[3] Frank Y. Shih and O. Robert Mitchell, " A Mathematical Morphology Approach to Euclidean Distance Transformation", IEEE Transactions on Signal Processing, to appear.

[4] Robert M. Haralick, Stanley R. Sternberg and Xinhua Zhuang, " Image analysis Using Mathematical Morphology", IEEE transactions on PAMI, VOl 9, No. 4, July 1987.

[5] H.J.A.M. Heijmans and C Ronse, " The Algebraic Basis of Mathematical Morphology", Computer Vision, Graphics and Image Processing, 50, 245-295, 1990

[6] Jean Serra, " Introduction to Mathematical Morphology", Computer Vision, Graphics and Image Processing, 35, 283-305, 1986

[7] Frank Y. Shih and Chandra S. Prathuri, " Shape Extraction from Size Histogram", proceedings, Sixth Israel Conference on AI and Computer Vision, December 1989.

[8] Jianning Xu, " Decomposion ofConvex Polygonal Morphological Structuring Elements into Neighborhood Subsets", IEEE transactions on PAMI, VOl 13, No. 2, February 1991.

[9] Petros Maragos and Ronald W. Schafer, " Morphological System for Multidimensional Signal Processing", Proceedings of the IEEE, Vol 78, No 4, April 1990.

[10] Ben-Kwei Jang, Ronald T. Chin, " Analysis of Thinning Algorithms Using Mathematical Morphology", IEEE transactions on PAMI, VOl 12, No. 6, June 1990.

[11] Frank Y. Shih and O. Robert Mitchell, " Decomposition of Gray Scale Morphological Structuring elements", Presented at IEEE workshop on Computer Vision, Florida, Nov. 30- Dec. 2, 1987

[12] Frank Y. Shih, " A New Compression Technique Based on Geometric Features", Proceedings of International Conference on Image Processing, Singapore, September 1989.

[13] Frank Y. shih and O. Robert Mitchell, " Automated Fast Recognition and Location of Arbitrarily Shaped Objects by Image Morphology", presented at IEEE Conference on computer Vision & Pattern Recognition, June 1988

[14] Azriel Rosenfield, " Axial Representation of Shape", Computer Vision, Graphics and image Processing, 33, 156-173, 1986.

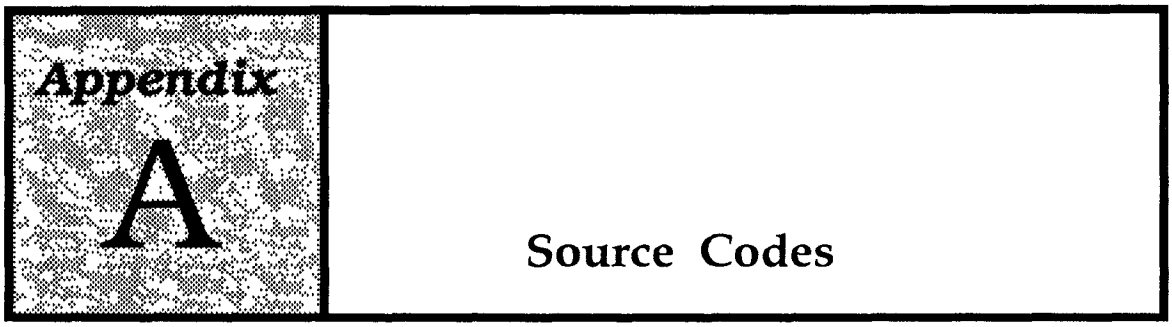
[15] W. K. Pratt, " Digital Image Processing", John wiley, 1978.

[16] Azriel Rosenfield and Avinash C. Kak, " Digital Picture Processing", Academic press, 1982.

[17] R. C. Gonzalez and P. Wintz, " Digital Image Processing", Addison-Wesley, 1987.

[18] O'Reilly and Associates , " The X Windows System in a nut shell"

[19] Dan Heller, O'Rrilly Series, Vol. VII, " The XView Programming Manual"



Source Codes

1. include files.
2. menu.c
3. edt.c
4. in_out.c
5. close_open.c
6. dilation.c
7. erosion.c
8. close_open.c
9. enlarge.c
10. recon.c
11. reversing.c
12. makefile

```
/*****************************************************/
##/* make file for the programs and the exec file is morphology */
/*****************************************************/
##
##
EXEC=morphology
##
##
CFLAGS=-O -I/usr/openwin/include -I/usr/csd/include
LIBS=-lxview-lolgx -lX11 -L/usr/openwin/lib -L/usr/csd/lib -lm
##
OBJECTS=menu.o\
        edt.o\
        in_out.o\
        recon.o\
        enlarge.o\
        reversing.o\
        dilation.o\
        erosion.o\
        close_open.o\
        save_image.o\
        com.o
##
$(EXEC):$(OBJECTS)
        cc      -o      $(EXEC)$(OBJECTS) $(LIBS)
```

```
/*****************************************************/
/*  mydefs.h,   where the image description and declaration is made*/
/*****************************************************/

#define MAX_SIZE 512

#define MIN2(I, J)(I < J) ? I : J
#define MIN3(I, J, k)MIN2(MIN2(I , J),k)


struct Image {
long pxls[MAX_SIZE][MAX_SIZE];
int row;
int col;
};

struct point {
int i;
int j;
int MAXi;
int MAXj;
long r;
}/* global image declerations */
;
struct Image *Orig, *img_F, *img_C, *img_S;
```

```
/*****************************************************/
/* include library files for XView and Xlib          */
/*****************************************************/
/* ********** include_defs.h ************ */

#include<stdio.h>
#include<ctype.h>
#include<math.h>
#include<strings.h>
#include<string.h>

#include<xview/xview.h>
#include<xview/frame.h>
#include<xview/panel.h>
#include<xview/canvas.h>
#include<xview/font.h>
#include<X11/Xlib.h>
#include<X11/Xutil.h>
#include<xview/rect.h>
#include<xview/xv_xrect.h>
#include<xview/cms.h>
#include<xview/svrimage.h>
#include<xview/textsw.h>

FILE *fopen(),*fin;

/* ********************* ******************** */
```

```
/* ************************************************** */
/* Xview application source code for user interactive
menu driven morphlogial image processing and image
display */
/* ************************************************** */
#include<stdio.h>
#include"include_defs.h"
#include"mydefs.h"

Frame  loframe, frame;
Menu DIL, EROD, CLOS, OPEN;
Display *display;
Canvas canvas;
Cms cms;
XID xid;
GC gc;
XImage *img;
unsigned long *color;
Panel panel;
static Xv_singlecolor colorty[] = {
                { 255, 255, 255},
                { 255, 0, 0},

                { 0, 255, 255},
                { 0, 0, 0}};
XFontStruct *font;
XGCValues gc_val;
static char stipple[] = {0xAA, 0xAA, 0x55, 0x55};
Textsw textsw;

extern struct Image *reverse();
extern struct Image *EDT();
extern struct Image *erode();
extern struct Image *dilate();
extern struct Image *closing();
extern struct Image *opening();
```

```
extern struct Image *reconstruct();
extern struct Image *enlarge();
extern compress_img();
extern save();

char infile[20];
int row,col;
char *name;
char *str;
char *s1, *s2, *s3;

/* ********************************************** */
/* reads filename from a sub window and extracts
the image width abd length from the filename   */
/* ********************************************** */


read_file(item, event)
Panel_item item;
Event *event;
{
char buf[20];
char *filename = (char *)xv_get(item, PANEL_VALUE);
int i,j;

Frame frame = (Frame)xv_get(xv_get(item,PANEL_PARENT_PANEL),XV_OWNER);


if ((fin = fopen(filename,"r")) == NULL)  {
            sprintf(buf,"cannot open file '%s'",filename);
          printf("file %s not found\n",filename);
            xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);
            return PANEL_NONE;
      }
str = (char *)calloc(1, sizeof(char));
strcpy( str,filename);
```

```
s1 = strtok(str,".");
str = (char *) 0;
s2 = strtok(str,"r");
str = (char *) 0;
s3 = strtok(str,"\0");
str = (char *) 0;
if (s3 == (char *)0)
      row = col  = atoi(s2);
else {
col = atoi(s2);
row = atoi(s3);
}

sprintf(buf,"                          ");
      xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);

printf(" row = %d\t col = %d\n",row,col);

free ( img_F);
if ( img_F != NULL )
         puts ( " unable to free img_f " );

img_F = ((struct Image *)malloc( sizeof(struct Image)));
free ( Orig );
Orig = ((struct Image *)malloc( sizeof(struct Image)));

img_F->row = row;
img_F->col = col;

read_img ( fin, img_F );
img_display(img_F);
  fclose( fin);
Orig->row = img_F->row;
Orig->col = img_F->col;
  for (i=0;i<Orig->row;i++)
```

```
   for (j=0;j<Orig->col;j++)
    Orig->pxls[i][j] = img_F->pxls[i][j];

      xv_destroy_safe(loframe);

}
/* ********************************************** */
/* pops up a sub window for the purpose of entring
the image file name and if correct file name is mensioned
it loads the image, otherwise it displays message
and waits for the correct filename*/
/* ********************************************** */

 void load()
{
      Panel lpanel;
Menu lmenu;
      Rect lrect;
      Xv_singlecolor lbg, lfg;
  lbg.red = 0, lbg.green = 255, lbg.blue = 127;
      lfg.red = 255, lfg.green = 0, lfg.blue = 0;

      lrect.r_top = 100;
      lrect.r_left = 750;
      lrect.r_width = 390;
      lrect.r_height = 135;

   loframe = (Frame)xv_create(NULL, FRAME,
                        FRAME_LABEL, "Load File",
                        FRAME_SHOW_RESIZE_CORNER, FALSE,
                        FRAME_NO_CONFIRM, TRUE,
                        FRAME_SHOW_FOOTER, TRUE,
                        FRAME_INHERIT_COLORS, TRUE,
                        FRAME_FOREGROUND_COLOR, &lfg,
                        FRAME_BACKGROUND_COLOR, &lbg,
                        NULL);
```

```
frame_set_rect(loframe, &lrect);
        lpanel = (Panel)xv_create(loframe, PANEL, NULL);

xv_create(lpanel, PANEL_TEXT,
                PANEL_LABEL_STRING,"File Name:",
                PANEL_LAYOUT,PANEL_HORIZONTAL,
                PANEL_VALUE_DISPLAY_LENGTH, 20,
                PANEL_NOTIFY_PROC, read_file,
                NULL);

xv_create(lpanel, PANEL_MESSAGE,
                PANEL_NEXT_ROW,-1,
                PANEL_LABEL_STRING,"Format: name.COL.rROW (or) name.size (if
COL=ROW)",
                PANEL_LABEL_BOLD, TRUE,
                NULL);

xv_set(loframe, XV_SHOW, TRUE, NULL);
}


/* ********************************************** */
/* each function activates a respective program
passing appropriate input image and freeing old
images and displaying the result*/
/* ********************************************** */
redisp()
{
int i, j;
 if (img_C != Orig)
free(img_C);
 if (img_F != Orig)
free(img_F);
img_F = ((struct Image *)malloc( sizeof(struct Image)));
img_F->row = Orig->row;
```

```
img_F->col = Orig->col;
 for (i=0;i<img_F->row;i++)
   for (j=0;j<img_F->col;j++)
    img_F->pxls[i][j] = Orig->pxls[i][j];
img_C = img_F;
img_display(img_F);
}
/* ********************************************** */
quit()
{
 printf("\n All the output is in the file named RES.\n");
free(Orig);
free(img_F);
free(img_S);
free(img_C);
 xv_destroy_safe( frame);
}
/* ********************************************** */
EcDt()
{
struct Image *temp;
int i, j, k;
double K;

temp = ((struct Image *)malloc( sizeof(struct Image)));
temp->row = img_F->row;
temp->col = img_F->col;
if (img_C != img_F)
free(img_C);
img_C = EDT(img_F);
for (i=0;i<img_C->row;i++)
 for(j=0;j<img_C->col;j++)
{ K = sqrt((double)(img_C->pxls[i][j]));
        temp->pxls[i][j] = K;
}
img_display(temp);
```

```
free (temp);
}
/* ********************************************** */
/* ********************************************** */
revers_image()
{
if (img_C != img_F)
free(img_C);
img_C = reverse(img_F);
free(img_F);
img_display(img_C);
img_F= img_C;
}
/* ********************************************** */
ENLARGE()
{
if (img_C != img_F)
free(img_C);
img_C = enlarge(img_F);
img_display(img_C);
free(img_F);
img_F=img_C;
}
/* ********************************************** */
/* Crude method of calling a function with differt
valuese, activated by menu buttons */
/* ********************************************** */
D2()
{
if (img_C != img_F)
free(img_C);
img_F = dilate(img_C,2);
img_display(img_F);
}
/* ********************************************** */
D3()
```

```
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F = dilate(img_C,3);
img_display(img_F);
}
/* ********************************************** */
D4()
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F = dilate(img_C,4);
img_display(img_F);
}
/* ********************************************** */
D5()
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F = dilate(img_C,5);
img_display(img_F);
}
/* ********************************************** */
D6()
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F = dilate(img_C,6);
img_display(img_F);
}
/* ********************************************** */
D7()
```

```
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F = dilate(img_C,7);
img_display(img_F);
}
/* ********************************************** */
D8()
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F = dilate(img_C,8);
img_display(img_F);
}
/* ********************************************** */
D9()
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F = dilate(img_C,9);
img_display(img_F);
}
/* ********************************************** */
E2()
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F = erode(img_C,2);
img_display(img_F);
}
/* ********************************************** */
E3()
```

```
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F = erode(img_C,3);
img_display(img_F);
}
/* ********************************************** */
E4()
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F =  erode(img_C,4);
img_display(img_F);
}
/* ********************************************** */
E5()
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F = erode (img_C,5);
img_display(img_F);
}
/* ********************************************** */
E6()
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F =  erode(img_C,6);
img_display(img_F);
}
/* ********************************************** */
E7()
```

```
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F = erode(img_C,7);
img_display(img_F);
}
/* ********************************************** */
E8()
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F = erode (img_C,8);
img_display(img_F);
}
/* ********************************************** */
E9()
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F = erode (img_C,9);
img_display(img_F);
}
/* ********************************************** */
O2()
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F = opening(img_C,2);
img_display(img_F);
}
/* ********************************************** */
O3()
```

```
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F = opening(img_C,3);
img_display(img_F);
}
/* ********************************************** */
O4()
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F =  opening(img_C,4);
img_display(img_F);
}
/* ********************************************** */
O5()
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F = opening(img_C,5);
img_display(img_F);
}
/* ********************************************** */
O6()
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F = opening (img_C,6);
img_display(img_F);
}
/* ********************************************** */
O7()
```

```
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F = opening(img_C,7);
img_display(img_F);
}
/* ********************************************** */
O8()
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F = opening (img_C,8);
img_display(img_F);
}
/* ********************************************** */
O9()
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F =  opening(img_C,9);
img_display(img_F);
}
/* ********************************************** */
C2()
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F = closing(img_C,2);
img_display(img_F);
}
/* ********************************************** */
C3()
```

```
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F = closing(img_C,3);
img_display(img_F);
}
/* ********************************************** */
C4()
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F =  closing (img_C,4);
img_display(img_F);
}
/* ********************************************** */
C5()
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F =  closing(img_C,5);
img_display(img_F);
}
/* ********************************************** */
C6()
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F =  closing(img_C,6);
img_display(img_F);
}
/* ********************************************** */
C7()
```

```
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F = closing(img_C,7);
img_display(img_F);
}
/* ********************************************** */
C8()
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F =  closing(img_C,8);
img_display(img_F);
}
/* ********************************************** */
C9()
{
if (img_C != img_F)
free(img_C);
img_C = img_F;
img_F = closing(img_C,9);
img_display(img_F);
}
/* ********************************************** */
clear()
{
  XClearArea(display, xid, 0,0,650,800, TRUE);
  xv_destroy_safe( textsw);
}
/* ********************************************** */
/* clears display area and displays the image     */
/* ********************************************** */
img_display(image)
struct Image *image;
```

```
{
 int i,j,k;
 XClearArea(display, xid, 0,0,640,800, TRUE);
 xv_destroy_safe( textsw);
 img = XGetImage( display,xid,0,0,640,640,-1,ZPixmap);

 for( i=0;i<image->row;i++)
  for( j=0;j<image->col;j++)
  {
   k = image->pxls[i][j];
   XPutPixel( img,j,i,k);
  }
  XPutImage( display,xid,gc,img,0,0,50,50,640,640);
}
/* ********************************************** */
/* reads image filename and stroks the size of the
image and then creates display frame, buttons, and
menu for interaction*/
/* ********************************************** */
main( argc,argv)
 int argc;
 char *argv[];
{
 Xv_Singlecolor ffg, fbg;

 Rect rect;
 rect.r_top = rect.r_left = 50;
 rect.r_width = 800;
 rect.r_height = 800;
 ffg.red =180, ffg.green = 0, ffg.blue = 100;
 fbg.red = 180, fbg.green = 230, fbg.blue = 190;

 xv_init( XV_INIT_ARGC_PTR_ARGV,&argc,argv,NULL);
 frame = ( Frame) xv_create( NULL,FRAME,
                FRAME_LABEL,"Compression and Object Recognition",
                FRAME_INHERIT_COLORS,TRUE,
```

```
                    XV_X,50,XV_Y,50,
                    XV_WIDTH,800,XV_HEIGHT,800,
                    FRAME_FOREGROUND_COLOR, &ffg,
                    FRAME_BACKGROUND_COLOR, &fbg,
                    NULL);
        frame_set_rect(frame, &rect);

        cms = ( Cms) xv_create(
NULL,CMS,CMS_CONTROL_CMS,TRUE,CMS_SIZE,CMS_CONTROL_COLORS+4,CMS_
COLORS,colorty,NULL);

        panel = ( Panel) xv_create( frame,PANEL,XV_X, 650,XV_Y,100,NULL);

        display = ( Display *)xv_get( frame,XV_DISPLAY);




      DIL = (Menu)xv_create(NULL, MENU,
                        MENU_ACTION_ITEM,"3 X 3",D3,
                        MENU_ACTION_ITEM,"5 X 5",D5,
                        MENU_ACTION_ITEM,"7 X 7",D7,
                        MENU_ACTION_ITEM,"9 X 9",D9,
                        MENU_ACTION_ITEM,"Horiz",D2,
                        MENU_ACTION_ITEM,"Virti",D4,
                        MENU_ACTION_ITEM,"Diag45",D6,
                        MENU_ACTION_ITEM,"DIAG135",D8,
                        NULL);

          EROD = (Menu)xv_create(NULL, MENU,
                        MENU_ACTION_ITEM,"3 X 3",E3,
                        MENU_ACTION_ITEM,"5 X 5",E5,
                        MENU_ACTION_ITEM,"7 X 7",E7,
                        MENU_ACTION_ITEM,"9 X 9",E9,
                        MENU_ACTION_ITEM,"Horiz",E2,
                        MENU_ACTION_ITEM,"Virti",E4,
```

```
                MENU_ACTION_ITEM,"Diag45",E6,
                MENU_ACTION_ITEM,"DIAG135",E8,
                NULL);

    CLOS = (Menu)xv_create(NULL, MENU,
                MENU_ACTION_ITEM,"3 X 3",C3,
                MENU_ACTION_ITEM,"5 X 5",C5,
                MENU_ACTION_ITEM,"7 X 7",C7,
                MENU_ACTION_ITEM,"9 X 9",C9,
                MENU_ACTION_ITEM,"Horiz",C2,
                MENU_ACTION_ITEM,"Virti",C4,
                MENU_ACTION_ITEM,"Diag45",C6,
                MENU_ACTION_ITEM,"DIAG135",C8,
                NULL);

    OPEN = (Menu)xv_create(NULL, MENU,
                MENU_ACTION_ITEM,"3 X 3",O3,
                MENU_ACTION_ITEM,"5 X 5",O5,
                MENU_ACTION_ITEM,"7 X 7",O7,
                MENU_ACTION_ITEM,"9 X 9",O9,
                MENU_ACTION_ITEM,"Horiz",O2,
                MENU_ACTION_ITEM,"Virti",O4,
                MENU_ACTION_ITEM,"Diag45",O6,
                MENU_ACTION_ITEM,"DIAG135",O8,
                NULL);




( void) xv_create( panel,PANEL_BUTTON,
            PANEL_NEXT_ROW,-1,
            PANEL_LABEL_STRING, "Load image",
            PANEL_NOTIFY_PROC, load,
            NULL);

( void) xv_create( panel,PANEL_BUTTON,
```

```
                PANEL_NEXT_ROW,-1,
                PANEL_LABEL_STRING, "Disp Orig Image",
                PANEL_NOTIFY_PROC, redisp,
                NULL);

( void) xv_create( panel,PANEL_BUTTON,
                PANEL_NEXT_ROW,-1,
                PANEL_LABEL_STRING, "ED Transform",
                PANEL_NOTIFY_PROC,EcDt,
                NULL);

( void) xv_create( panel,PANEL_BUTTON,
                PANEL_NEXT_ROW,-1,
                PANEL_LABEL_STRING, " Dilation",
                PANEL_ITEM_MENU,DIL,
                NULL);

( void) xv_create( panel,PANEL_BUTTON,
                PANEL_NEXT_ROW,-1,
                PANEL_LABEL_STRING, " Erosion   ",
                PANEL_ITEM_MENU,EROD,
                NULL);

( void) xv_create( panel,PANEL_BUTTON,
                PANEL_NEXT_ROW,-1,
                PANEL_LABEL_STRING, " Closing   ",
                PANEL_ITEM_MENU,CLOS,
                NULL);

( void) xv_create( panel,PANEL_BUTTON,
                PANEL_NEXT_ROW,-1,
                PANEL_LABEL_STRING, " Opening   ",
                PANEL_ITEM_MENU,OPEN,
                NULL);

( void) xv_create( panel,PANEL_BUTTON,
```

```
                PANEL_NEXT_ROW,-1,
                PANEL_LABEL_STRING, "Reverse",
                PANEL_NOTIFY_PROC,revers_image,
                NULL);

( void) xv_create( panel,PANEL_BUTTON,
                PANEL_NEXT_ROW,-1,
                PANEL_LABEL_STRING, "Enlarge",
                PANEL_NOTIFY_PROC,ENLARGE,
                NULL);

( void) xv_create( panel,PANEL_BUTTON,
                PANEL_NEXT_ROW,-1,
                PANEL_LABEL_STRING, "Compress   ",
                PANEL_NOTIFY_PROC,compress_img,
                NULL);

( void) xv_create( panel,PANEL_BUTTON,
                PANEL_NEXT_ROW,-1,
                PANEL_LABEL_STRING, "Reconstruct ",
                PANEL_NOTIFY_PROC,reconstruct,
                NULL);

 ( void) xv_create( panel,PANEL_BUTTON,
                PANEL_NEXT_ROW,-1,
                PANEL_LABEL_STRING, "Save Image",
                PANEL_NOTIFY_PROC, save,
                NULL);

 ( void) xv_create( panel,PANEL_BUTTON,
                PANEL_NEXT_ROW,-1,
                PANEL_LABEL_STRING, "Clear Image",
                PANEL_NOTIFY_PROC,clear,
                NULL);

( void) xv_create( panel,PANEL_BUTTON,
```

```
                PANEL_NEXT_ROW,-1,
                PANEL_LABEL_STRING, "Quit  ",
                PANEL_NOTIFY_PROC, quit,
                NULL);
  window_fit( panel);
  window_fit( panel);
  window_fit( panel);

  canvas = xv_create( frame,CANVAS,
                XV_X,10, XV_Y,10,
                XV_WIDTH,640,
                XV_HEIGHT,800,
                CANVAS_X_PAINT_WINDOW,TRUE,
                WIN_DYNAMIC_VISUAL, TRUE, WIN_CMS,cms, NULL);


  window_fit( frame);
  xid = ( XID)xv_get( canvas_paint_window( canvas),XV_XID);
  font = XLoadQueryFont( display,"fixed");
  gc_val.font = font->fid;
  gc_val.stipple = XCreateBitmapFromData( display, xid,stipple,16,2);
  gc= XCreateGC( display, xid, GCFont | GCStipple, &gc_val);
  color = ( unsigned long *)xv_get( canvas,WIN_X_COLOR_INDICES);

  xv_main_loop( frame);
}
/* ********************************************** */


/* ************END OF MAIN PROGRAM*************** */
```

```
/*************************************************/
/* This program calculates the euclidean distance transformation
using back propagation double scan algorithm    */

/* ********************************************** */

#include <stdio.h>

#include "mydefs.h"

struct Image *D_IMAGE;
struct Image *E_IMAGE;
struct Image *R_IMAGE;
struct Image *Q_IMAGE;

extern struct Image *Make_Image();
int Str_elm[3][3] ;

 /* ********************************************** */

struct Image *onetomax(img)
struct Image *img;

{
struct Image *temp;
int i, j;

temp = Make_Image();
temp->col = img->col;
temp->row = img->row;


for (i=0; i< img->row; i++)
        for(j=0; j<img->col; j++)
        {
        if(img->pxls[i][j]==1)
```

```
                temp->pxls[i][j]=1000000;
                else temp->pxls[i][j]=0;
        }
return(temp);
}
/*  * ************************** */

void ST_EL_G(X,L)
int X, L;

{
int i, j;
if (L==0)
 L = 1;
Str_elm[1][1] = 0;
if (X == 1)
{       Str_elm[0][0] = 2 - (4 * L);
        Str_elm[0][1] = 1 - (2 * L) ;
        Str_elm[0][2] = 2 - (4 * L) ;
        Str_elm[1][0] = 1 - (2 * L) ;
        Str_elm[1][2] = -1000000;
        Str_elm[2][0] = -1000000;
        Str_elm[2][1] = -1000000;
        Str_elm[2][2] = -1000000;
}
else if (X == 2)
{       Str_elm[0][0] = -1000000;
        Str_elm[0][1] = -1000000;
        Str_elm[0][2] = -1000000;
        Str_elm[1][0] = -1000000;
        Str_elm[1][2] = 1 - (2 * L);
        Str_elm[2][0] = 2 - (4 * L);
        Str_elm[2][1] = 1 - (2 * L);
        Str_elm[2][2] = 2 - (4 * L);
}
```

```
}
/*  ********************************************** */

void ST_EL_K(X)
int X;

{
Str_elm[1][1] = 0;
if (X == 1)
{       Str_elm[0][0] = -1;
        Str_elm[0][1] = -1;
        Str_elm[0][2] = -1;
        Str_elm[1][0] = -1;
        Str_elm[1][2] = -1000000;
        Str_elm[2][0] = -1000000;
        Str_elm[2][1] = -1000000;
        Str_elm[2][2] = -1000000;
}
else if (X == 2)
{       Str_elm[0][0] = -1000000;
        Str_elm[0][1] = -1000000;
        Str_elm[0][2] = -1000000;
        Str_elm[1][0] = -1000000;
        Str_elm[1][2] = -1;
        Str_elm[2][0] = -1;
        Str_elm[2][1] = -1;
        Str_elm[2][2] = -1;
}
  else  { fprintf(stderr,"Selected argument value is not valid");
        exit(1); }

}

/*  ********************************************** */

/* This procedure is used to produce the lookup table for the 'l' value
```

```
generate structuring elements for the third and fourth step.
This procedure is used to for D_image and E_image with structuring element
k1 and k2                */
/* ********************************************** */

struct Image *First_scan(KX, img)
int KX;
struct Image *img;
{
        struct Image *T_image;
        int I, J, i, j, k;
        int ti, tj;
        long stmp, t[3][3];
ST_EL_K(KX);

T_image = Make_Image();
T_image->row = img->row;
T_image->col = img->col;
        for (i=0; i<img->row; i++)
                for (j=0; j<img->col; j++)
                T_image->pxls[i][j] = 0;

for (I=0; I<img->row; I++)
for (J=0; J<img->col; J++)

{
        if (KX == 2)
        { i = img->row - I;
          j = img->col - J; }
        else {  i = I;
                j = J;}

        if (img->pxls[i][j] == 1) k = 1000000;
        else k = img->pxls[i][j] ;
                t[1][1] = k - Str_elm[1][1];
```

```
if ( i != 0 && i != img->row && j != 0 && j != img->col)
 {
                t[0][0] = T_image->pxls[i-1][j-1]-Str_elm[0][0];
                t[0][1] = T_image->pxls[i-1][j]-Str_elm[0][1];
                t[0][2] = T_image->pxls[i-1][j+1]-Str_elm[0][2];
                t[1][0] = T_image->pxls[i][j-1]-Str_elm[1][0];
                t[1][2] = T_image->pxls[i][j+1]-Str_elm[1][2];
                t[2][0] = T_image->pxls[i+1][j-1]-Str_elm[2][0];
                t[2][1] = T_image->pxls[i+1][j]-Str_elm[2][1];
                t[2][2] = T_image->pxls[i+1][j+1]-Str_elm[2][2];
}


else {

        if ( i == 0 && j != 0 && j != img->col) {
                t[0][0] = 0 - Str_elm[0][0];
                t[0][1] = 0 - Str_elm[0][1];
                t[0][2] = 0 - Str_elm[0][2];
                t[1][0] = T_image->pxls[i][j-1]-Str_elm[1][0];
                t[1][2] = T_image->pxls[i][j+1]-Str_elm[1][2];
                t[2][0] = T_image->pxls[i+1][j-1]-Str_elm[2][0];
                t[2][1] = T_image->pxls[i+1][j]-Str_elm[2][1];
                t[2][2] = T_image->pxls[i+1][j+1]-Str_elm[2][2];
                                                        }
else    if ( j == 0 && i != 0 && i != img->row) {
                t[0][0] = 0 - Str_elm[0][0];
                t[0][1] = T_image->pxls[i-1][j]-Str_elm[0][1];
                t[0][2] = T_image->pxls[i-1][j+1]-Str_elm[0][2];
                t[1][0] = 0 - Str_elm[1][0];
                t[1][2] = T_image->pxls[i][j+1]-Str_elm[1][2];
                t[2][0] = 0 - Str_elm[2][0];
                t[2][1] = T_image->pxls[i+1][j-1]-Str_elm[2][1];
                t[2][2] = T_image->pxls[i+1][j+1]-Str_elm[2][2];
                                                        }
 else    if (i == img->row && j != 0 && j != img->col) {
```

```
                t[0][0] = T_image->pxls[i-1][j-1]-Str_elm[0][0];
                t[0][1] = T_image->pxls[i-1][j]-Str_elm[0][1];
                t[0][2] = T_image->pxls[i-1][j+1]-Str_elm[0][2];
                t[1][0] = T_image->pxls[i][j-1]-Str_elm[1][0];
                t[1][2] = T_image->pxls[i][j+1]-Str_elm[1][2];
                t[2][0] = 0 - Str_elm[2][0];
                t[2][1] = 0 - Str_elm[2][1];
                t[2][2] = 0 - Str_elm[2][2];
                                                        }

else    if ( j == img->col && i != 0 && i != img->row) {
                t[0][0] = T_image->pxls[i-1][j-1]-Str_elm[0][0];
                t[0][1] = T_image->pxls[i-1][j]-Str_elm[0][1];
                t[0][2] = 0 - Str_elm[0][2];
                t[1][0] = T_image->pxls[i][j-1]-Str_elm[1][0];
                t[1][2] = 0 - Str_elm[1][2];
                t[2][0] = T_image->pxls[i+1][j-1]-Str_elm[2][0];
                t[2][1] = T_image->pxls[i+1][j]-Str_elm[2][1];
                t[2][2] = 0 - Str_elm[2][2];
                                                        }

else    if ( i == 0 && j == 0 ) {
                t[0][0] = 0 - Str_elm[0][0];
                t[0][1] = 0 - Str_elm[0][1];
                t[0][2] = 0 - Str_elm[0][2];
                t[1][0] = 0 - Str_elm[1][0];
                t[1][2] = T_image->pxls[i][j+1]-Str_elm[1][2];
                t[2][0] = 0 - Str_elm[2][0];
                t[2][1] = T_image->pxls[i+1][j]-Str_elm[2][1];
                t[2][2] = T_image->pxls[i+1][j+1]-Str_elm[2][2];
                                                        }
else    if ( i == img->row && j == 0 ) {
                t[0][0] = 0 - Str_elm[0][0];
                t[0][1] = T_image->pxls[i-1][j]-Str_elm[0][1];
                t[0][2] = T_image->pxls[i-1][j+1]-Str_elm[0][2];
                t[1][0] = 0 - Str_elm[1][0];
```

```
                t[1][2] = T_image->pxls[i][j+1]-Str_elm[1][2];
                t[2][0] = 0 - Str_elm[2][0];
                t[2][1] = 0 - Str_elm[2][1];
                t[2][2] = 0 - Str_elm[2][2];
                                                        }
else    if ( i == 0 && j == img->col) {
                t[0][0] = 0 - Str_elm[0][0];
                t[0][1] = 0 - Str_elm[0][1];
                t[0][2] = 0 - Str_elm[0][2];
                t[1][0] = T_image->pxls[i][j-1]-Str_elm[1][0];
                t[1][2] = 0 - Str_elm[1][2];
                t[2][0] = T_image->pxls[i+1][j-1]-Str_elm[2][0];
                t[2][1] = T_image->pxls[i+1][j]-Str_elm[2][1];
                t[2][2] = 0 - Str_elm[2][2];
                                                        }
else    if  ( i == img->row && j == img->col) {
                t[0][0] = T_image->pxls[i-1][j-1]-Str_elm[0][0];
                t[0][1] = T_image->pxls[i-1][j]-Str_elm[0][1];
                t[0][2] = 0 - Str_elm[0][2];
                t[1][0] = T_image->pxls[i][j-1]-Str_elm[1][0];
                t[1][2] = 0 - Str_elm[1][2];
                t[2][0] = 0 - Str_elm[2][0];
                t[2][1] = 0 - Str_elm[2][1];
                t[2][2] = 0 - Str_elm[2][2];
                                                        }

    }
stmp = 1000000;
for(ti=0;ti<3;ti++)
 for(tj=0;tj<3;tj++)
  stmp= MIN2(stmp,t[ti][tj]);
T_image->pxls[i][j] = stmp;
}
return(T_image);
}
```

```
/* ********************************************** */
/* This procedure uses the result of First_scan for the generation of
the structuring elements and gives out the resultin euclidean transform
after scannin the image from left-to right_&_top_to_bottom and
right_to_left_&_and bottm_to_top    */

/* ********************************************** */



struct Image *Second_scan(KX, img, LUKUP_IMAGE)
int KX;
struct Image *img;
struct Image *LUKUP_IMAGE;
{
        struct Image *T_image;
        int I, J, L, i, j;
        int ti, tj;
        long stmp, t[3][3];

T_image = Make_Image();
T_image->row = img->row;
T_image->col = img->col;
        for (i=0; i<img->row; i++)
                for (j=0; j<img->col; j++)
                T_image->pxls[i][j] = 0;
for (I=0; I<img->row; I++)
for (J=0; J<img->col; J++)


{
        if (KX == 2)
        { i = img->row - I;
          j = img->col - J; }
        else {  i = I;
```

```
                j = J;}
L = LUKUP_IMAGE->pxls[i][j];
ST_EL_G(KX,L);

                t[1][1] = img->pxls[i][j] - Str_elm[1][1];

if ( i != 0 && i != img->row && j != 0 && j != img->col)
 {
                t[0][0] = T_image->pxls[i-1][j-1]-Str_elm[0][0];
                t[0][1] = T_image->pxls[i-1][j]-Str_elm[0][1];
                t[0][2] = T_image->pxls[i-1][j+1]-Str_elm[0][2];
                t[1][0] = T_image->pxls[i][j-1]-Str_elm[1][0];
                t[1][2] = T_image->pxls[i][j+1]-Str_elm[1][2];
                t[2][0] = T_image->pxls[i+1][j-1]-Str_elm[2][0];
                t[2][1] = T_image->pxls[i+1][j]-Str_elm[2][1];
                t[2][2] = T_image->pxls[i+1][j+1]-Str_elm[2][2];
}


else {

        if ( i == 0 && j != 0 && j != img->col) {
                t[0][0] = 0 - Str_elm[0][0];
                t[0][1] = 0 - Str_elm[0][1];
                t[0][2] = 0 - Str_elm[0][2];
                t[1][0] = T_image->pxls[i][j-1]-Str_elm[1][0];
                t[1][2] = T_image->pxls[i][j+1]-Str_elm[1][2];
                t[2][0] = T_image->pxls[i+1][j-1]-Str_elm[2][0];
                t[2][1] = T_image->pxls[i+1][j]-Str_elm[2][1];
                t[2][2] = T_image->pxls[i+1][j+1]-Str_elm[2][2];
                                                        }
else    if ( j == 0 && i != 0 && i != img->row) {
                t[0][0] = 0 - Str_elm[0][0];
                t[0][1] = T_image->pxls[i-1][j]-Str_elm[0][1];
                t[0][2] = T_image->pxls[i-1][j+1]-Str_elm[0][2];
                t[1][0] = 0 - Str_elm[1][0];
```

```
                t[1][2] = T_image->pxls[i][j+1]-Str_elm[1][2];
                t[2][0] = 0 - Str_elm[2][0];
                t[2][1] = T_image->pxls[i+1][j-1]-Str_elm[2][1];
                t[2][2] = T_image->pxls[i+1][j+1]-Str_elm[2][2];
                                                        }
else    if (i == img->row && j != 0 && j != img->col) {
                t[0][0] = T_image->pxls[i-1][j-1]-Str_elm[0][0];
                t[0][1] = T_image->pxls[i-1][j]-Str_elm[0][1];
                t[0][2] = T_image->pxls[i-1][j+1]-Str_elm[0][2];
                t[1][0] = T_image->pxls[i][j-1]-Str_elm[1][0];
                t[1][2] = T_image->pxls[i][j+1]-Str_elm[1][2];
                t[2][0] = 0 - Str_elm[2][0];
                t[2][1] = 0 - Str_elm[2][1];
                t[2][2] = 0 - Str_elm[2][2];
                                                        }

else    if ( j == img->col && i != 0 && i != img->row) {
                t[0][0] = T_image->pxls[i-1][j-1]-Str_elm[0][0];
                t[0][1] = T_image->pxls[i-1][j]-Str_elm[0][1];
                t[0][2] = 0 - Str_elm[0][2];
                t[1][0] = T_image->pxls[i][j-1]-Str_elm[1][0];
                t[1][2] = 0 - Str_elm[1][2];
                t[2][0] = T_image->pxls[i+1][j-1]-Str_elm[2][0];
                t[2][1] = T_image->pxls[i+1][j]-Str_elm[2][1];
                t[2][2] = 0 - Str_elm[2][2];
                                                        }

else    if ( i == 0 && j == 0 ) {
                t[0][0] = 0 - Str_elm[0][0];
                t[0][1] = 0 - Str_elm[0][1];
                t[0][2] = 0 - Str_elm[0][2];
                t[1][0] = 0 - Str_elm[1][0];
                t[1][2] = T_image->pxls[i][j+1]-Str_elm[1][2];
                t[2][0] = 0 - Str_elm[2][0];
                t[2][1] = T_image->pxls[i+1][j]-Str_elm[2][1];
                t[2][2] = T_image->pxls[i+1][j+1]-Str_elm[2][2];
```

```
                                }
else    if ( i == img->row && j == 0 ) {
                t[0][0] = 0 - Str_elm[0][0];
                t[0][1] = T_image->pxls[i-1][j]-Str_elm[0][1];
                t[0][2] = T_image->pxls[i-1][j+1]-Str_elm[0][2];
                t[1][0] = 0 - Str_elm[1][0];
                t[1][2] = T_image->pxls[i][j+1]-Str_elm[1][2];
                t[2][0] = 0 - Str_elm[2][0];
                t[2][1] = 0 - Str_elm[2][1];
                t[2][2] = 0 - Str_elm[2][2];
                                                        }
else    if ( i == 0 && j == img->col) {
                t[0][0] = 0 - Str_elm[0][0];
                t[0][1] = 0 - Str_elm[0][1];
                t[0][2] = 0 - Str_elm[0][2];
                t[1][0] = T_image->pxls[i][j-1]-Str_elm[1][0];
                t[1][2] = 0 - Str_elm[1][2];
                t[2][0] = T_image->pxls[i+1][j-1]-Str_elm[2][0];
                t[2][1] = T_image->pxls[i+1][j]-Str_elm[2][1];
                t[2][2] = 0 - Str_elm[2][2];
                                                        }
else    if  ( i == img->row && j == img->col) {
                t[0][0] = T_image->pxls[i-1][j-1]-Str_elm[0][0];
                t[0][1] = T_image->pxls[i-1][j]-Str_elm[0][1];
                t[0][2] = 0 - Str_elm[0][2];
                t[1][0] = T_image->pxls[i][j-1]-Str_elm[1][0];
                t[1][2] = 0 - Str_elm[1][2];
                t[2][0] = 0 - Str_elm[2][0];
                t[2][1] = 0 - Str_elm[2][1];
                t[2][2] = 0 - Str_elm[2][2];
                                                        }

   }
stmp = 1000000;
for(ti=0;ti<3;ti++)
 for(tj=0;tj<3;tj++)
```

```
 stmp= MIN2(stmp,t[ti][tj]);
T_image->pxls[i][j] = stmp;
}
return(T_image);
}


/* ********************************************
This sub_main program for Euclidean Distance Transformation
Shows the clear picture of the algorithm.
******************************************** */




struct Image *EDT(img_A)
struct Image *img_A;
{
int i, j;
struct Image  *img_B;

D_IMAGE = First_scan(1,img_A);
E_IMAGE = First_scan(2,img_A);
img_B = onetomax(img_A);
R_IMAGE = Second_scan(1,img_B,D_IMAGE);
Q_IMAGE = Second_scan(2,R_IMAGE,E_IMAGE);
free(R_IMAGE);
free(D_IMAGE);
free(E_IMAGE);
free(img_B);
   return(Q_IMAGE);
}


/************************************************/
```

```
/* ***********************************************/
/* This program is used to decompose a given image using
Euclidean distance transform and finding the largest
disk fits in the object and subtracts form the image,
final result will be the coordinates and radii
of the disks which can be used to reconstruct the
original image.*/
/************************************************** */

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

struct Image *Q_IMAGE;
struct Image *imgC;
struct Image *M_IMAGE;
extern img_display();
extern read_img();
extern write_image();
extern struct Image *EDT();
extern struct Image *reconstruct();

FILE *res_out;

struct Image *Make_Image()
{
return((struct Image *)malloc( sizeof(struct Image)));
}
/*  ******************************************** */
/* Converts a gray scale image to binary image*    */
/*  ******************************************** */

struct Image *gray_to_bin(img)
struct Image *img;
```

```
{
struct Image *temp;
int i, j;

temp = Make_Image();
temp->col = img->col;
temp->row = img->row;


for (i=0; i< img->row; i++)
        for(j=0; j<img->col; j++)
                if(img->pxls[i][j]>=1)
                temp->pxls[i][j]=1;
                else temp->pxls[i][j]=0;
return(temp);
}
/*  ********************************************** */
/* Searches the EDT image for maximum value       */
/*  ********************************************** */

struct point *MAX_RAD(image)
struct Image *image;
{
int m,  n, x, y;
struct point *P;
P = (struct point *)malloc(sizeof(struct point));

P->r = 0;
for (m=0; m< image->row; m++)
  for(n=0; n<image->col; n++)
        if(image->pxls[m][n] > P->r)
        {       P->i = m;
                P->j = n;
                P->r =  image->pxls[m][n];
        }
P->MAXi = image->row;
```

```
P->MAXj = image->col;
 return(P);
}


/*  ********************************************** */
/*  Creates a disk with the given radius and coordinates
and the size of the image so that can be or'd easily */
/*  ********************************************** */
struct Image *CREATE_DISK(P)
struct point *P;
{
struct Image *img;
int i, j, k;
double K;

img = Make_Image();
  K = sqrt((double)(P->r));
k = K/1;
if ((K-k)>0.5 ) k++;

for (i=0; i < P->MAXi; i++)
for (j=0; j < P->MAXj ; j++)
if (P->r > ((i - P->i ) * (i - P->i ) + (j - P->j ) * (j - P->j)))
   img->pxls[i][j] = 1;
else img->pxls[i][j] = 0;

img->row = P->MAXi;
img->col = P->MAXj;
printf("\n");
return(img);
}
/*  ********************************************** */
/*  writes in the RES file the coordinates and the
square of the radius which is used in reconstruction*/
/*  ********************************************** */
```

```
void write_pt(out,pt)
FILE *out;
struct point *pt;
{
fprintf(out,"%d  ",pt->i);
fprintf(out,"%d  ",pt->j);
fprintf(out,"%d \n",pt->r);
}
/*  ********************************************** */
/*  direct binary image subtraction */
/*  ********************************************** */
struct Image *sub_image(img1,img2)
struct Image *img1, *img2;

{
struct Image *img3;
int temp, i, j;

img3 = Make_Image();

for (i=0; i<img1->row; i++)
for (j=0; j<img1->col; j++)
        img3->pxls[i][j] = ((temp = img1->pxls[i][j] - img2->pxls[i][j]) < 0)? 0 : temp;

img3->row = img1->row;
img3->col = img1->col;
return(img3);
}

/*  ********************************************** */
/* segmentation using EDT and disk subtraction is
applied through this procedure */
/*  ********************************************** */

compress_img()
```

```
{
int i, j ;
struct point *Pt;
struct Image *img_A, *img_B, *img_D ;

if (( res_out = fopen("RES", "w")) == NULL)
{
fprintf(stderr,"cannot open file RES to write\n");
exit(2);
}

img_A = EDT(img_F);
fprintf(res_out,"%d  %d \n",img_F->row,img_F->col);

Pt = MAX_RAD(img_A);
while ((Pt->r) > 1)
{
write_pt(res_out,Pt);
    img_B = CREATE_DISK(Pt);

    img_D = gray_to_bin(img_A);
    if (imgC != NULL) free(imgC);
    imgC = sub_image(img_D,img_B);


img_display(imgC);

          free(img_A);
          free(img_B);
          free(img_D);
         img_A = EDT(imgC);
         Pt = MAX_RAD(img_A);

}
for (i=0; i< imgC->row; i++)
        for(j=0; j<imgC->col; j++)
```

```
if (imgC->pxls[i][j]){
        imgC->pxls[i][j] = 0;
        fprintf(res_out,"%d  %d  %c\n",i,j,'1');
                        }
img_display(imgC);

free(img_C);
img_C = imgC;

fclose(res_out);

}

/*****************************************************/
```

```
/* ************************************************/
/* These procedures are used to read from a file
and to write in a file
print_image prints the image values on the screen*/
/************************************************/
#include <stdio.h>
#include <math.h>
#include "mydefs.h"
 /* ********************************************** */

read_img(fp, img)
FILE *fp;
struct Image *img;

{
int i, j, t;
for (i=0; i< img->row; i++)
        for(j=0; j<img->col; j++)
                fscanf(fp,"%d",&img->pxls[i][j]);

}

/*  ********************************************** */

print_image(img)
struct Image *img;
{
int i, j;
for (i=0; i< img->row; i++)
{
        for(j=0; j<img->col; j++)
                printf("%d ",img->pxls[i][j]);
printf("\n");
}
}
```

```
/*  * ************************** */


write_img(outf,img)
FILE *outf;
struct Image *img;
{
int i, j;
for (i=0; i< img->row; i++)
{
        for(j=0; j<img->col; j++)
                fprintf(outf,"%d ",img->pxls[i][j]);
fprintf(outf,"\n");
}
}


/*************************************************/
```

```
/*************************************************/
/* This program dilates a given image with the selected
structuring element.  The structuring element list has
horizontal, vertical, diagonal45, diagonal135, 3X3, 5X5,
7X7 and 9X9.., where are the X_by_X procedure can handle
any odd square structuring element */


/* ************************************** */
#include <stdio.h>
#include "mydefs.h"

int i,j, stel;
struct Image *di_img;

/* ************************************** */

void X_by_X(imgA)
struct Image *imgA;
{
int a, b, st, x;

st = stel/2;

for(i=0;i<imgA->row;i++)
 for(j=0;j<imgA->col;j++)
{
  x = 0;

   for (a=0;a<=st;a++)
   for (b=0;b<=st;b++)
   {
         if(!( ((i-a)<0) && ((j-b)<0)) )
                  if(imgA->pxls[i-a][j-b])
                  x=1;
         if(!( ((i+a)>imgA->row) && ((j+b)>imgA->col)) )
```

```
                if(imgA->pxls[i+a][j+b])
                x=1;
  }

        if(imgA->pxls[i][j])
        x=1;
   di_img->pxls[i][j]=x;
      }
}

/* ****************************************** */
 /* horizontal ST_EL  [0 0 0] */

/* ****************************************** */
void TWO(imgB)
struct Image *imgB;
{
for (i=0;i<imgB->row;i++)
for (j=0;j<imgB->col;j++)
        {
        if (i != 0 || i != imgB->row)
         {
         if ( imgB->pxls[i-1][j] || imgB->pxls[i][j] || imgB->pxls[i+1][j])
         di_img->pxls[i][j] = 1;
         else di_img->pxls[i][j] = 0;
         }
         else if (i == 0)
         {
         if ( imgB->pxls[i][j] || imgB->pxls[i+1][j])
         di_img->pxls[i][j] = 1;
         else di_img->pxls[i][j] = 0;
         }
         else if (i == imgB->row)
         {
         if ( imgB->pxls[i-1][j] || imgB->pxls[i][j])
         di_img->pxls[i][j] = 1;
```

```
            else di_img->pxls[i][j] = 0;
            }
            else di_img->pxls[i][j] = 0;
          }
}

/* ****************************************** */
/* virtical ST_EL
[0]
[0]
[0] */
/* ****************************************** */
void FOUR(imgC)
struct Image *imgC;

{
for (i=0;i<imgC->row;i++)
for (j=0;j<imgC->col;j++)
          {
          if (j != 0 || j != imgC->row)
            {
            if ( imgC->pxls[i][j-1] || imgC->pxls[i][j] || imgC->pxls[i][j+1])
            di_img->pxls[i][j] = 1;
            else di_img->pxls[i][j] = 0;
            }

            else if (j == 0)
            {
            if ( imgC->pxls[i][j] || imgC->pxls[i][j+1])
            di_img->pxls[i][j] = 1;
            else di_img->pxls[i][j] = 0;
            }
            else if (j == imgC->col)
            {
            if ( imgC->pxls[i][j-1] || imgC->pxls[i][j])
            di_img->pxls[i][j] = 1;
```

```
        else di_img->pxls[i][j] = 0;
        }
        else di_img->pxls[i][j] = 0;

      }
}

/* ************************************** */
/* 135 deg. ST_EL     [0]
                          [0]
                                [0]  */
/* ************************************** */
void EIGHT(imgD)
struct Image *imgD;
{
for (i=0;i<imgD->row;i++)
for (j=0;j<imgD->col;j++)
      {
      if (j != 0 && j != imgD->row && i != 0 && i != imgD->row)
       {
       if ( imgD->pxls[i-1][j-1] || imgD->pxls[i][j] || imgD->pxls[i+1][j+1])
       di_img->pxls[i][j] = 1;
       else di_img->pxls[i][j] = 0;
       }

       else if ( (j == 0 || i == 0) &&  i != imgD->row && j != imgD->col)
       {
       if ( imgD->pxls[i][j] || imgD->pxls[i+1][j+1])
       di_img->pxls[i][j] = 1;
       else di_img->pxls[i][j] = 0;
       }
       else if ( (i == imgD->row || j == imgD->col) && j != 0 && i != 0)
       {
       if ( imgD->pxls[i-1][j-1] || imgD->pxls[i][j])
       di_img->pxls[i][j] = 1;
       else di_img->pxls[i][j] = 0;
```

```
        }
        else  di_img->pxls[i][j] = imgD->pxls[i][j];
       }
}

/* ****************************************** */
            /* 45 deg. ST_EL       [0]
                                  [0]
                               [0]  */
/* ****************************************** */

void SIX(imgE)
struct Image *imgE;
{
for (i=0;i<imgE->row;i++)
for (j=0;j<imgE->col;j++)
        {
       if (j != 0 && j != imgE->row && i != 0 && i != imgE->row)
        {
        if ( imgE->pxls[i+1][j-1] || imgE->pxls[i][j] || imgE->pxls[i-1][j+1])
        di_img->pxls[i][j] = 1;
        else di_img->pxls[i][j] = 0;
        }

        else if ((j == imgE->col || i == 0) &&  i != imgE->row && j != 0)
        {
        if ( imgE->pxls[i][j] || imgE->pxls[i+1][j-1])
        di_img->pxls[i][j] = 1;
        else di_img->pxls[i][j] = 0;
        }
        else if (( j == 0 || i == imgE->row) && i != 0 && j != imgE->row)
        {
        if ( imgE->pxls[i-1][j+1] || imgE->pxls[i][j])
        di_img->pxls[i][j] = 1;
        else di_img->pxls[i][j] = 0;
        }
```

```
        else  di_img->pxls[i][j] = imgE->pxls[i][j];
       }
}

/* ********************************************** */
struct Image *dilate(in_img,st_el)
struct Image *in_img;
int st_el;
{

di_img = (struct Image *)malloc(sizeof(struct Image));
di_img->row = in_img->row;
di_img->col = in_img->col;
stel=st_el;
if ( st_el == 3 || st_el == 5 || st_el == 7 || st_el == 9 )
        X_by_X(in_img);
else if(st_el == 2)
        TWO(in_img);
else if(st_el == 4)
        FOUR(in_img);
else if(st_el == 6)
        SIX(in_img);
else if(st_el == 8)
        EIGHT(in_img);
else { printf ("wrong selection of ST_EL\n");
        return(in_img);
        }
return(di_img);
}


/************************************************/
```

```
/*****************************************************/
/* This program erodes a given image with the selected
structuring element. The structuring element list has
horizontal, vertical, diagonal45, diagonal135, 3X3, 5X5,
7X7 and 9X9.., where are the X_by_X procedure can handle
any odd square structuring element */


/* ************************************** */
#include <stdio.h>
#include "mydefs.h"

int i,j, stel;
struct Image *di_img;

/* ************************************** */

void K_by_K(imgA)
struct Image *imgA;
{
int a, b, st, x;

st = stel/2;

for(i=0;i<imgA->row;i++)
 for(j=0;j<imgA->col;j++)
{
  x = 1;

  for (a=0;a<=st;a++)
  for (b=0;b<=st;b++)
  {
        if(!( ((i-a)<0) && ((j-b)<0)) )
                if(!imgA->pxls[i-a][j-b])
                x=0;
        if(!( ((i+a)>imgA->row) && ((j+b)>imgA->col)) )
```

```
                if(!imgA->pxls[i+a][j+b])
                x=0;
  }

        if(!imgA->pxls[i][j])
        x=0;
   di_img->pxls[i][j]=x;
      }
}

/* ****************************************** */
 /* horizontal ST_EL  [0 0 0] */

/* ****************************************** */
void Two(imgB)
struct Image *imgB;
{
for (i=0;i<imgB->row;i++)
for (j=0;j<imgB->col;j++)
        {
        if (i != 0 || i != imgB->row)
         {
         if (  !imgB->pxls[i-1][j] ||  !imgB->pxls[i][j] ||  !imgB->pxls[i+1][j])
         di_img->pxls[i][j] = 0;
         else di_img->pxls[i][j] = 1;
         }
         else if (i == 0)
         {
         if (  !imgB->pxls[i][j] ||  !imgB->pxls[i+1][j])
         di_img->pxls[i][j] = 0;
         else di_img->pxls[i][j] = 1;
         }
         else if (i == imgB->row)
         {
         if (  !imgB->pxls[i-1][j] ||  !imgB->pxls[i][j])
         di_img->pxls[i][j] = 0;
```

```
            else di_img->pxls[i][j] = 1;
            }
            else di_img->pxls[i][j] = 0;
          }


}

/* ****************************************** */
/* virtical ST_EL
[0]
[0]
[0] */
/* ****************************************** */
void Four(imgC)
struct Image *imgC;

{
for (i=0;i<imgC->row;i++)
for (j=0;j<imgC->col;j++)
          {
          if (j != 0 || j != imgC->row)
            {
            if ( !imgC->pxls[i][j-1] || !imgC->pxls[i][j] || !imgC->pxls[i][j+1])
            di_img->pxls[i][j] = 0;
            else di_img->pxls[i][j] = 1;
            }

            else if (j == 0)
            {
            if ( !imgC->pxls[i][j] || !imgC->pxls[i][j+1])
            di_img->pxls[i][j] = 0;
            else di_img->pxls[i][j] = 1;
            }
            else if (j == imgC->col)
            {
```

```
        if ( !imgC->pxls[i][j-1] || !imgC->pxls[i][j])
        di_img->pxls[i][j] = 0;
        else di_img->pxls[i][j] = 1;
        }
        else di_img->pxls[i][j] = 0;

      }
}

/* ****************************************** */
/* 135 deg. ST_EL     [0]
                              [0]
                                    [0]  */
/* ****************************************** */
void Eight(imgD)
struct Image *imgD;
{
for (i=0;i<imgD->row;i++)
for (j=0;j<imgD->col;j++)
      {
      if (j != 0 && j != imgD->row && i != 0 && i != imgD->row)
       {
       if ( !imgD->pxls[i-1][j-1] || !imgD->pxls[i][j] || !imgD->pxls[i+1][j+1])
       di_img->pxls[i][j] = 0;
       else di_img->pxls[i][j] = 1;
       }

       else if ( (j == 0 || i == 0) &&  i != imgD->row && j != imgD->col)
       {
       if ( !imgD->pxls[i][j] || !imgD->pxls[i+1][j+1])
       di_img->pxls[i][j] = 0;
       else di_img->pxls[i][j] = 1;
       }
       else if ( (i == imgD->row || j == imgD->col) && j != 0 && i != 0)
       {
       if ( !imgD->pxls[i-1][j-1] || !imgD->pxls[i][j])
```

```
          di_img->pxls[i][j] = 0;
          else di_img->pxls[i][j] = 1;
          }
          else  di_img->pxls[i][j] = imgD->pxls[i][j];
         }
}

/* ****************************************** */
              /* 45 deg. ST_EL              [0]
                                    [0]
                           [0]  */
/* ****************************************** */

void Six(imgE)
struct Image *imgE;
{
for (i=0;i<imgE->row;i++)
for (j=0;j<imgE->col;j++)
         {
         if (j != 0 && j != imgE->row && i != 0 && i != imgE->row)
          {
          if ( !imgE->pxls[i+1][j-1] || !imgE->pxls[i][j] || !imgE->pxls[i-1][j+1])
          di_img->pxls[i][j] = 0;
          else di_img->pxls[i][j] = 1;
          }

          else if ((j == imgE->col || i == 0) &&  i != imgE->row && j != 0)
          {
          if ( !imgE->pxls[i][j] || !imgE->pxls[i+1][j-1])
          di_img->pxls[i][j] = 0;
          else di_img->pxls[i][j] = 1;
          }
          else if (( j == 0 || i == imgE->row) && i != 0 && j != imgE->row)
          {
          if ( !imgE->pxls[i-1][j+1] || !imgE->pxls[i][j])
          di_img->pxls[i][j] = 0;
```

```
            else di_img->pxls[i][j] = 1;
            }
            else  di_img->pxls[i][j] = imgE->pxls[i][j];
           }
}

/* ************************************************** */
struct Image *erode (in_img,st_el)
struct Image *in_img;
int st_el;
{

di_img = (struct Image *)malloc(sizeof(struct Image));
di_img->row = in_img->row;
di_img->col = in_img->col;
stel=st_el;
if ( st_el == 3 || st_el == 5 || st_el == 7 || st_el == 9 )
        K_by_K(in_img);
else if(st_el == 2)
        Two(in_img);
else if(st_el == 4)
        Four(in_img);
else if(st_el == 6)
        Six(in_img);
else if(st_el == 8)
        Eight(in_img);
else { printf ("wrong selection of ST_EL\n");
        return(in_img);
        }
return(di_img);
}
/**************************************************/
```

```
/*************************************************/
/*    ******************** CLOSE_OPEN *************/
#include <stdio.h>
#include "mydefs.h"
extern struct Image *dilate();
extern struct Image *erode();


 /* ********************************************/

/* This program calls the dialtion and erosion
procedures acoordingly for opening and closing*/
/********************************************** */

struct Image *closing(img,k)
struct Image *img;
int k;
{
        struct Image *T_image;
T_image = erode(dilate(img,k),k);
return(T_image);
}

struct Image *opening(img,k)
struct Image *img;
{
        struct Image *T_image;
T_image = dilate(erode(img,k),k);
return(T_image);
}

/*************************************************/
```

```
/* * ************************************************/
/* This procedure converts object to background and
back ground to object of a given image */

/*  ********************************************** */
#include <stdio.h>
#include "mydefs.h"

 /* ********************************************** */

struct Image *reverse(img)
struct Image *img;
{
        struct Image *T_image;
        int  i, j;

T_image = (struct Image *)malloc(sizeof(struct Image));
T_image->row = img->row;
T_image->col = img->col;
        for (i=0; i<img->row; i++)
                for (j=0; j<img->col; j++)
                if(img->pxls[i][j])
                T_image->pxls[i][j] = 0;
                else
                T_image->pxls[i][j] = 1;

return(T_image);
}


/************************************************/
```

```
/*****************************************************/
/*This procedure reconstructs the compressed image
i.e. coordinates and radii from the file RES */
/* ***************************************** */
#include <stdio.h>
#include <math.h>
#include "mydefs.h"

extern Make_Image();
struct Image *image;

/*  ***************************************** */

add_disk(ii,jj,rr)
int ii,jj,rr;
{
      int i, j, k;
      double K;

      K = sqrt((double)(rr));
      k = K/1;
      if ((K-k)>0.5 ) k++;

      for (i=(ii - k);i <( ii + k + 1); i++)
      for (j=(jj - k);j <( jj + k + 1); j++)
        {
      if (rr > ((i - ii ) * (i - ii ) + (j - jj ) * (j - jj)))
         image->pxls[i][j] = 1;
        }
}
/*  ***************************************** */

reconstruct()

{
      int i, j;
```

```
FILE *fp, *fopen();
long x,y,r;
char c;

image = (struct Image *)calloc( 1,sizeof(struct Image));
fp = fopen("RES", "r");

fscanf(fp,"%d %d", &x,&y);
image->row = x;
image->col = y;
printf("%d %d \n", image->row,image->col);
while ( fscanf(fp,"%d %d %d",&x,&y,&r) != EOF )
{
add_disk(x,y,r);
img_display(image);
}
img_F=image;
}



/*****************************************************/
```

```
/*************************************************/
/*  This function pops a window to enter the filename
for saving the current image.  This function will
add the dot extentions which give the size of the image
by itself based on the image saved. this feature is very
usefule as the function enlarge is availabel and user
need not remember the number of times the image is
enlarged or the multiplications.  */
 /* ******************************************** */


#include<stdio.h>
#include"mydefs.h"
#include"include_defs.h"

Frame saveframe;
char *str;
char *s1, *s2, *s3;

 /* ******************************************** */


quit_save()
{
      xv_destroy_safe(saveframe);
}

 /* ******************************************** */


save_img(item, event)
Panel_item item;
Event *event;
{
char *filename = (char *)xv_get(item, PANEL_VALUE);
Frame frame = (Frame)xv_get(xv_get(item,PANEL_PARENT_PANEL),XV_OWNER);
```

```
char buf1[20], buf[20];
int ii, jj;
FILE *fp1;

sprintf(buf,"                    ");
      xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);

if (img_C->col == img_C->row)
      sprintf(buf1,"%s.%d",filename, img_C->col );
else
      sprintf(buf1,"%s.%d.r%d",filename, img_C->col, img_C->row);

if ((fp1 = fopen(buf1,"w")) != NULL)  {
            sprintf(buf,"Saved  in file '%s'", buf1);
          printf("file %s is saved\n",buf1);
            xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);
}
else
            return PANEL_NONE;


     for ( ii=0;ii<img_C->row;ii++)
{
       for ( jj=0;jj<img_C->col;jj++)
         fprintf(fp1,"%d ",img_C->pxls[ii][jj]);
 fprintf(fp1,"\n");
}
sleep ( 5 );
xv_destroy_safe(saveframe);

fclose ( fp1 );
}

 /* ********************************************** */
```

```
void save()
{
    Panel lpanel;
Menu lmenu;
    Rect lrect;
    Xv_singlecolor lbg, lfg;
  lbg.red = 0, lbg.green = 255, lbg.blue = 127;
    lfg.red = 255, lfg.green = 0, lfg.blue = 0;

    lrect.r_top = 400;
    lrect.r_left = 750;
    lrect.r_width = 390;
    lrect.r_height = 135;

   saveframe = (Frame)xv_create(NULL, FRAME,
                        FRAME_LABEL, "SAVE IMAGE",
                        FRAME_SHOW_RESIZE_CORNER, FALSE,
                        FRAME_NO_CONFIRM, TRUE,
                        FRAME_SHOW_FOOTER, TRUE,
                        FRAME_INHERIT_COLORS, TRUE,
                        FRAME_FOREGROUND_COLOR, &lfg,
                        FRAME_BACKGROUND_COLOR, &lbg,
                        NULL);

frame_set_rect(saveframe, &lrect);
    lpanel = (Panel)xv_create(saveframe, PANEL, NULL);

xv_create(lpanel, PANEL_TEXT,
                PANEL_LABEL_STRING,"File Name:",
                PANEL_LAYOUT,PANEL_HORIZONTAL,
                PANEL_VALUE_DISPLAY_LENGTH, 20,
                PANEL_NOTIFY_PROC, save_img,
                NULL);

xv_create(lpanel, PANEL_MESSAGE,
                PANEL_NEXT_ROW,-1,
```

```
                PANEL_LABEL_STRING,"Just enter the file",
                PANEL_LABEL_BOLD, TRUE,
                NULL);

        xv_create(lpanel, PANEL_BUTTON,
                PANEL_NEXT_ROW,-1,
                XV_X, 290,
                PANEL_LABEL_STRING,"Quit",
                PANEL_LABEL_BOLD, TRUE,
                PANEL_NOTIFY_PROC, quit_save,
                NULL);

xv_set(saveframe, XV_SHOW, TRUE, NULL);

}

/* ********************************************** */
```

Appendix B

Results

1. Input image 1, with little noise and Morphological operations

2. Input image 2, with little noise and Morphological operations

3. Some screen dumps to show the window and Toolkit.

IMAGE F the input

```
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 0 0 0 0
0 0 1 1 1 1 1 1 0 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
```

IMAGE A the EDT output

```
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 0 0
0 0 1 4 4 4 4 4 4 1 0 0
0 0 1 4 9 9 5 2 1 1 0 0
0 0 1 4 8 5 4 1 0 0 0 0
0 0 1 4 5 2 1 1 0 0 0 0
0 0 1 4 4 1 0 0 0 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
```

IMAGE B the max disk of input

```
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
```

IMAGE C the remaining image

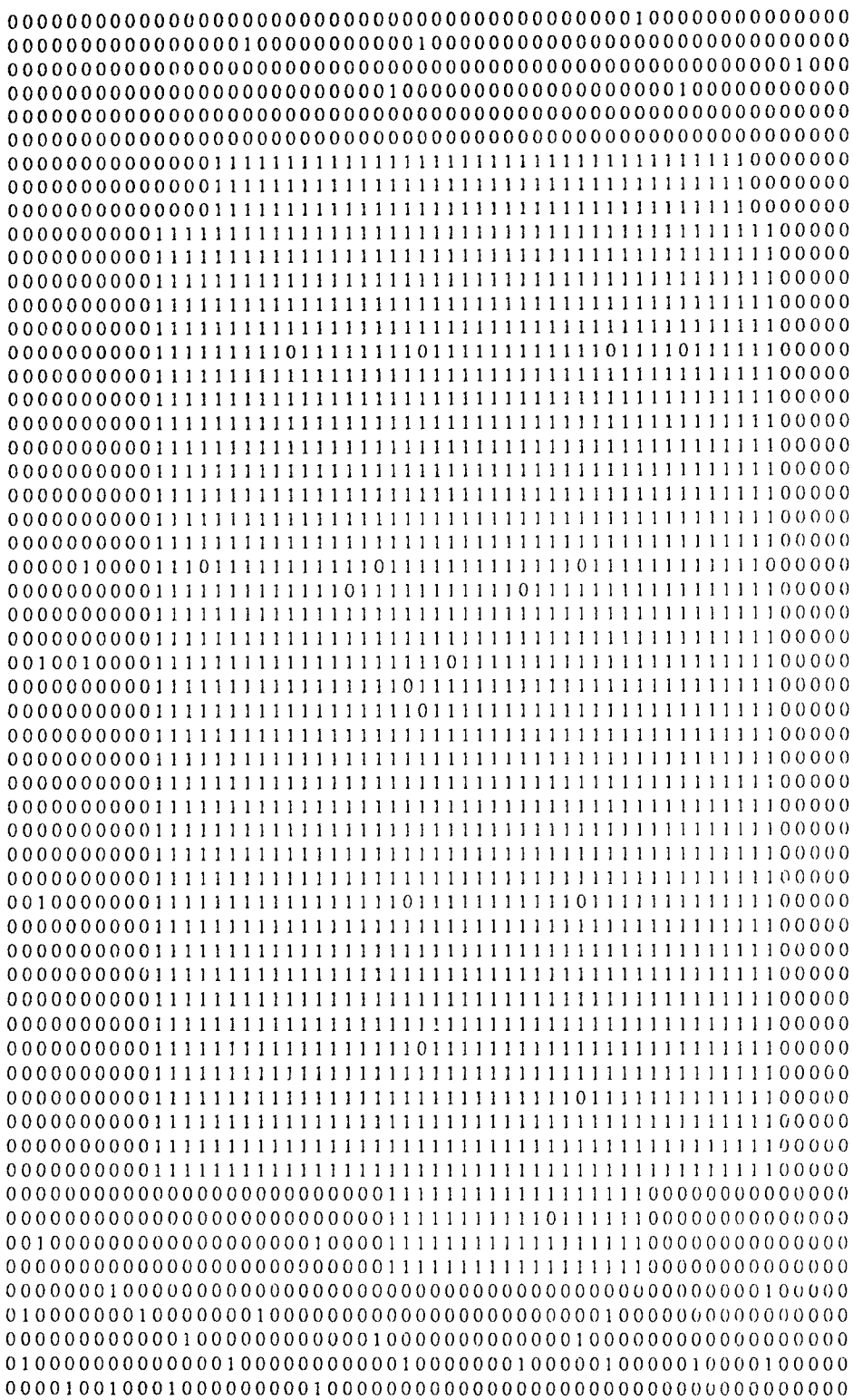
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0

IMAGE B the max disk of input

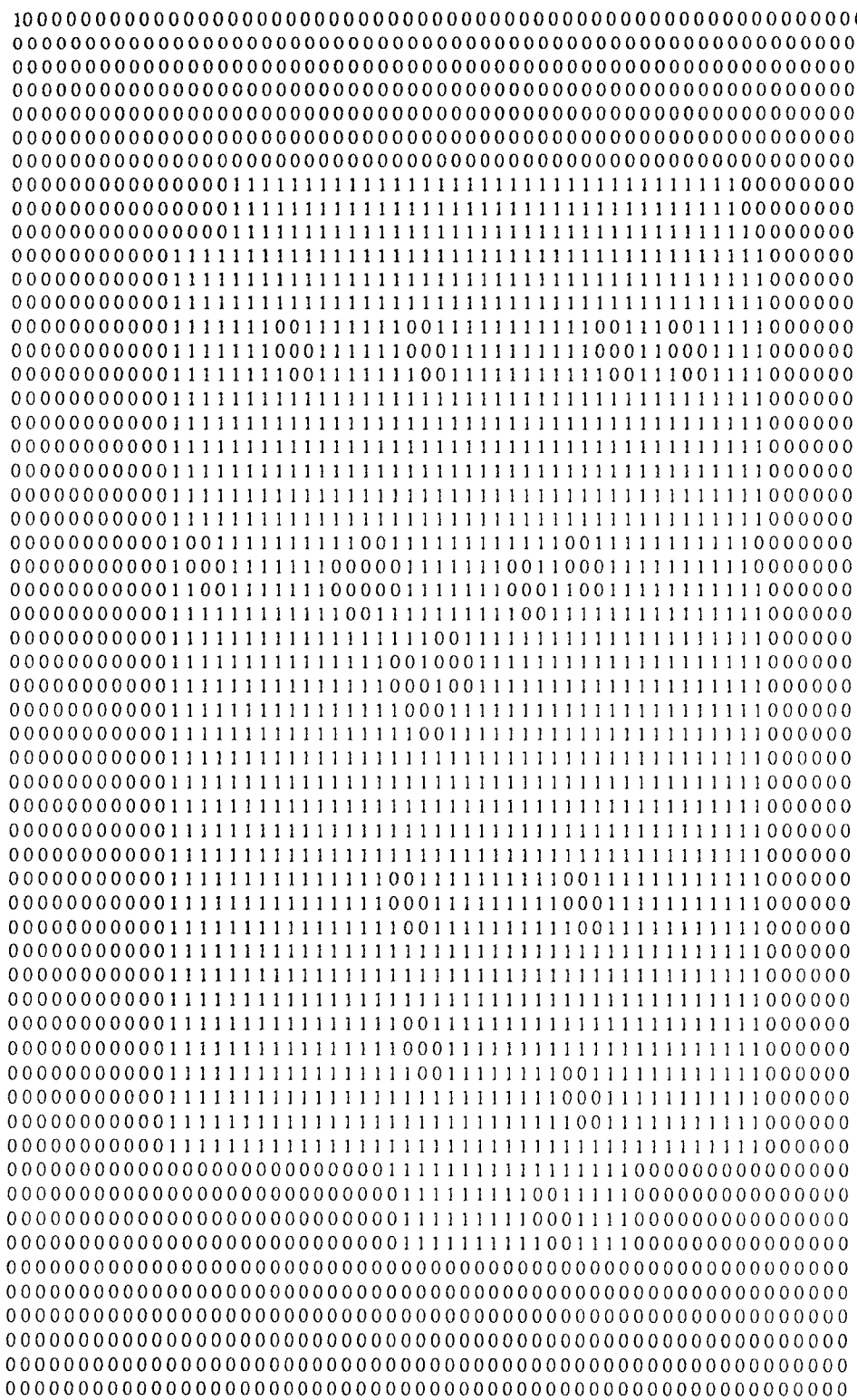
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0

IMAGE C the remaining image

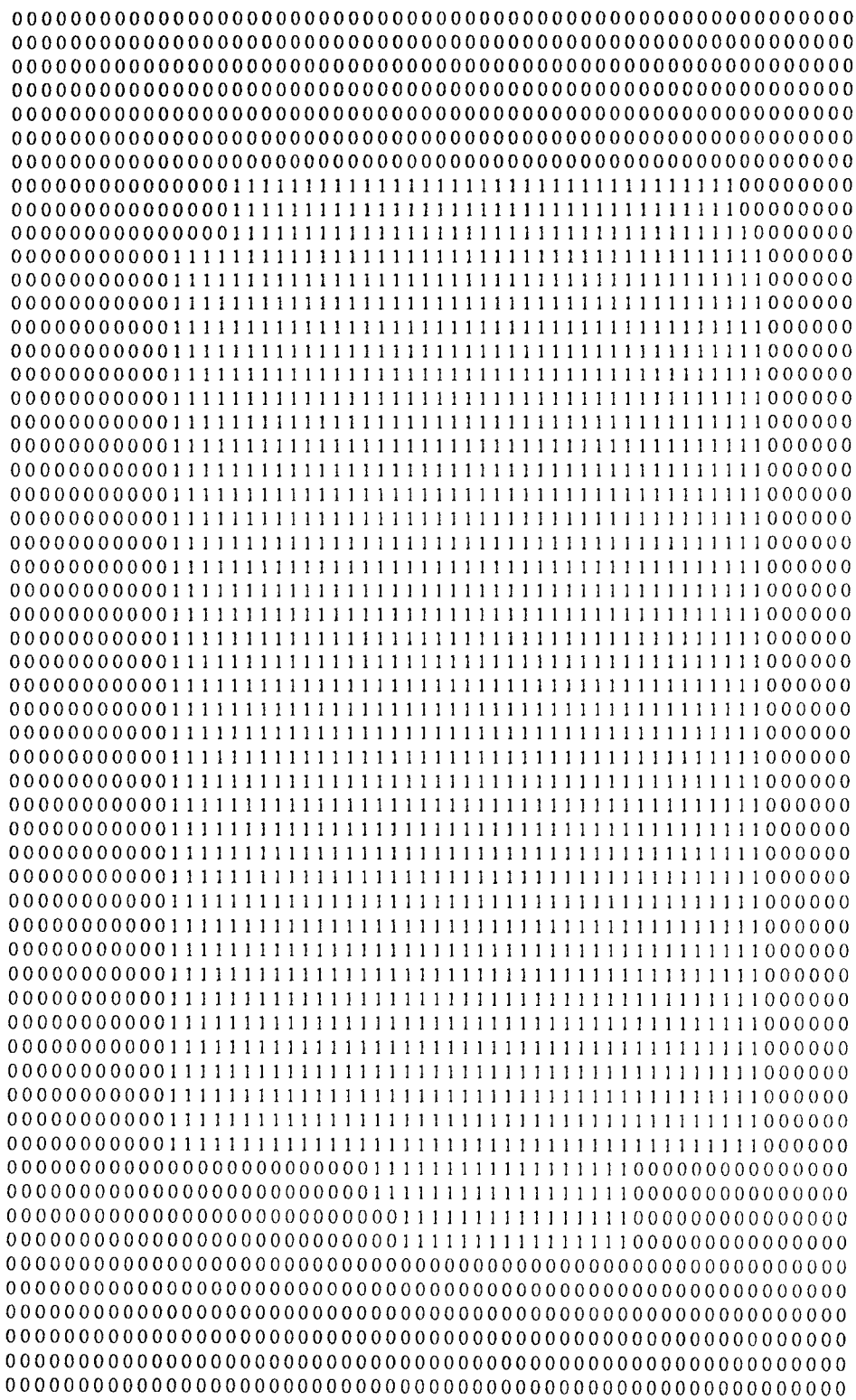
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0



Input image with noise



After erosion with 3x3 structuring Element



After closing with 5x5 structuring element

Fig. 1 The canvas and the menu.

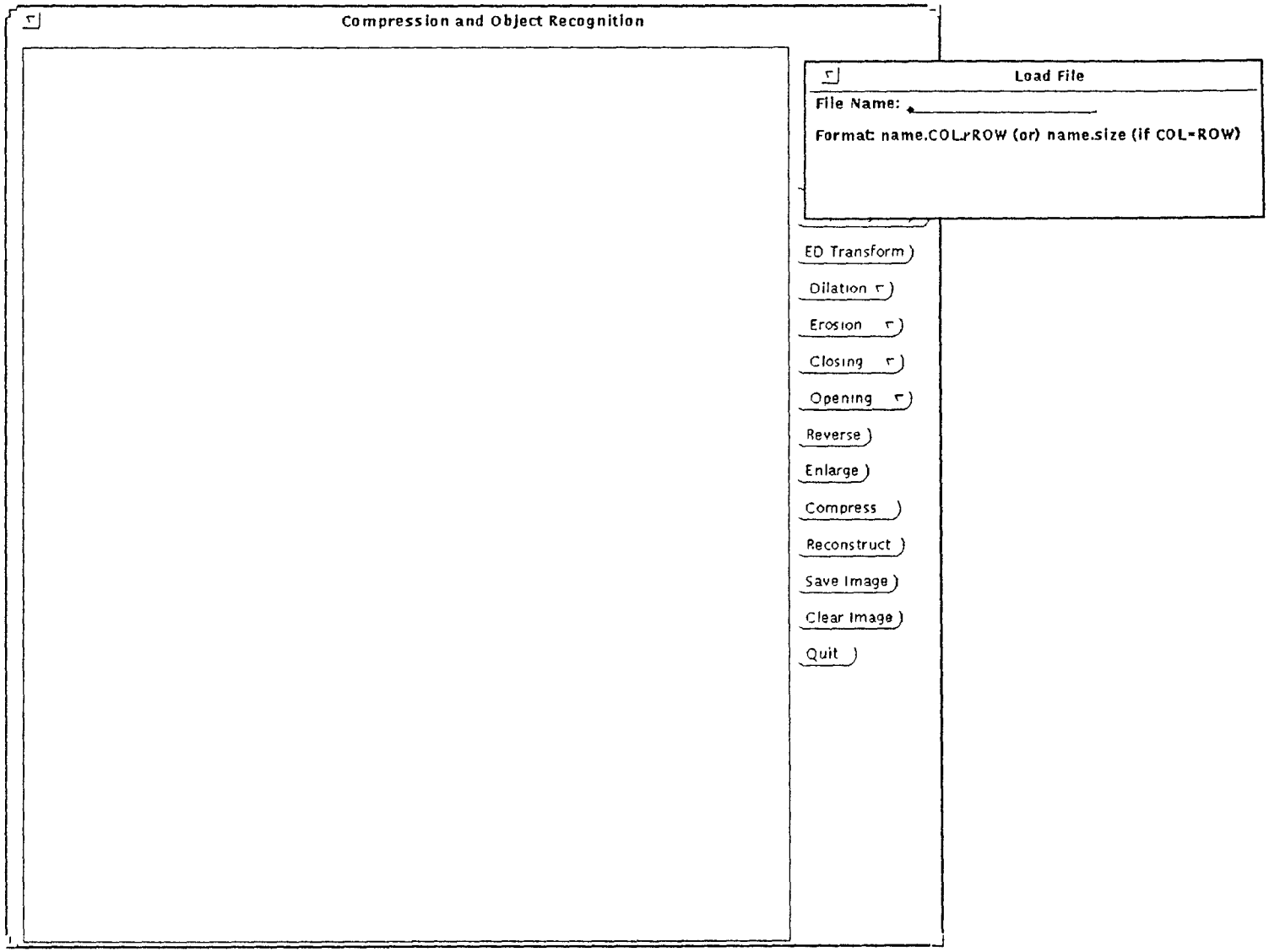


Fig. 2: With the popup window to load image

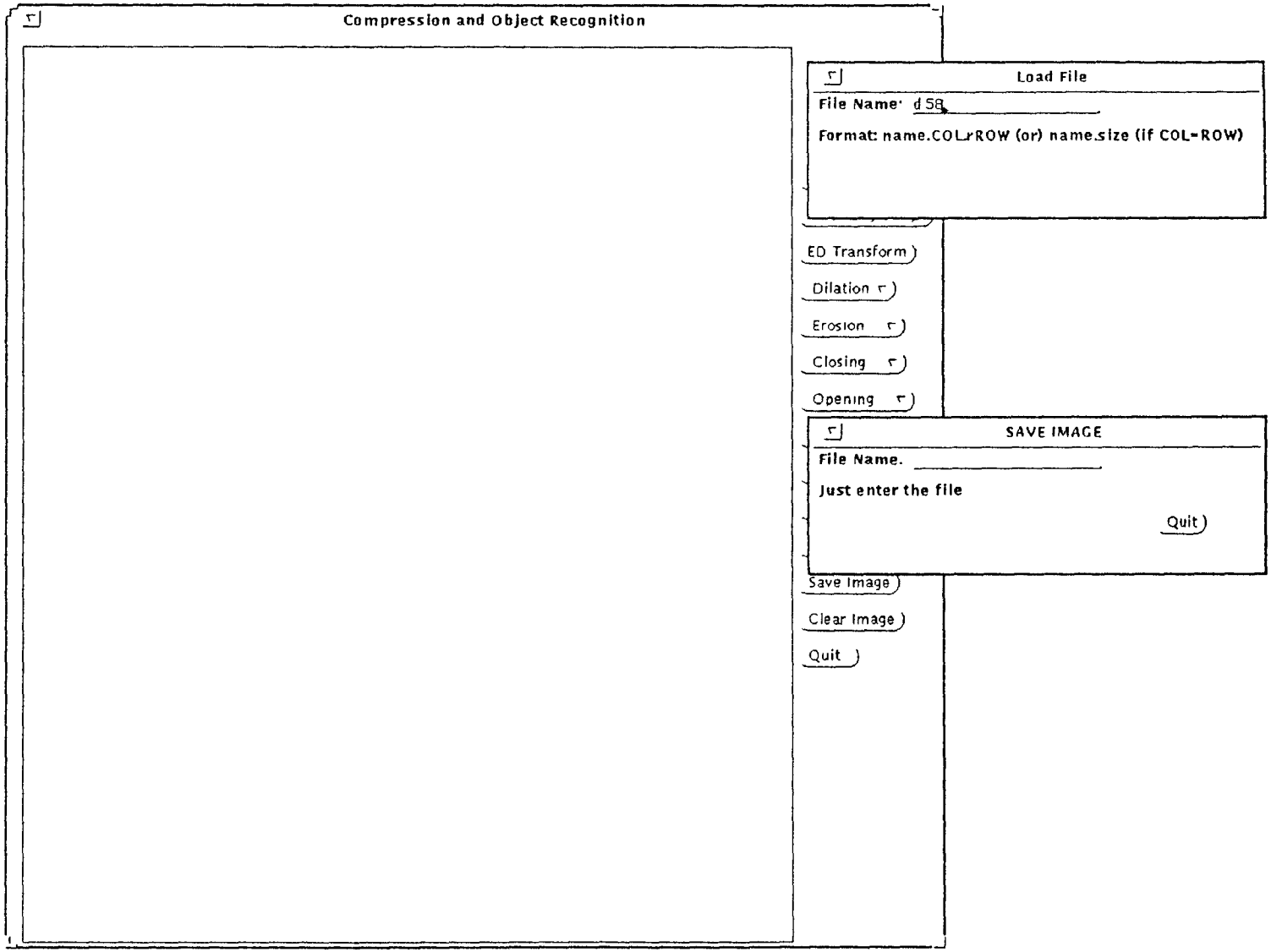


Fig. 3: Popup sub windos to load image abd save image

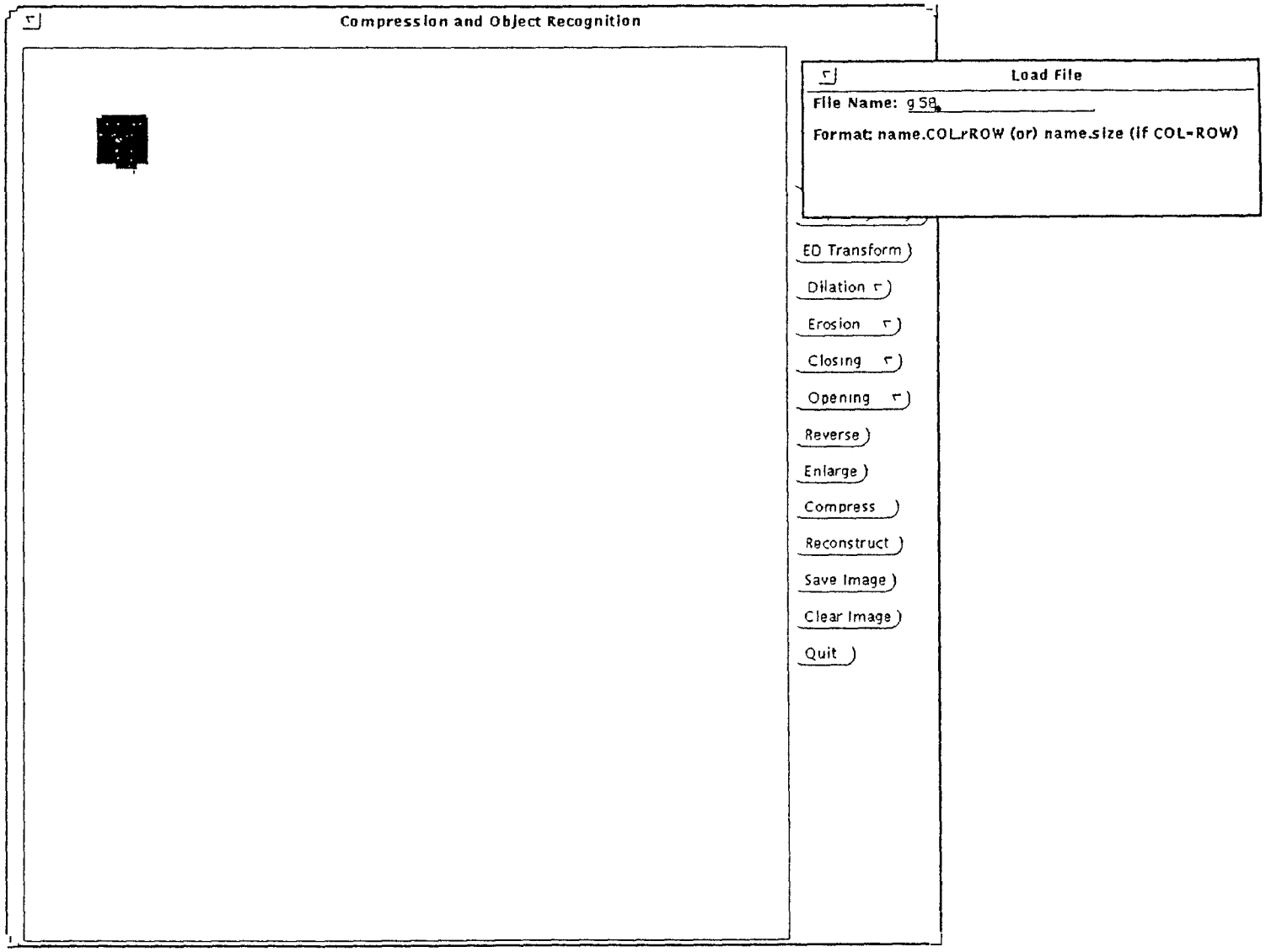


Fig. 4: After loading a noisy image

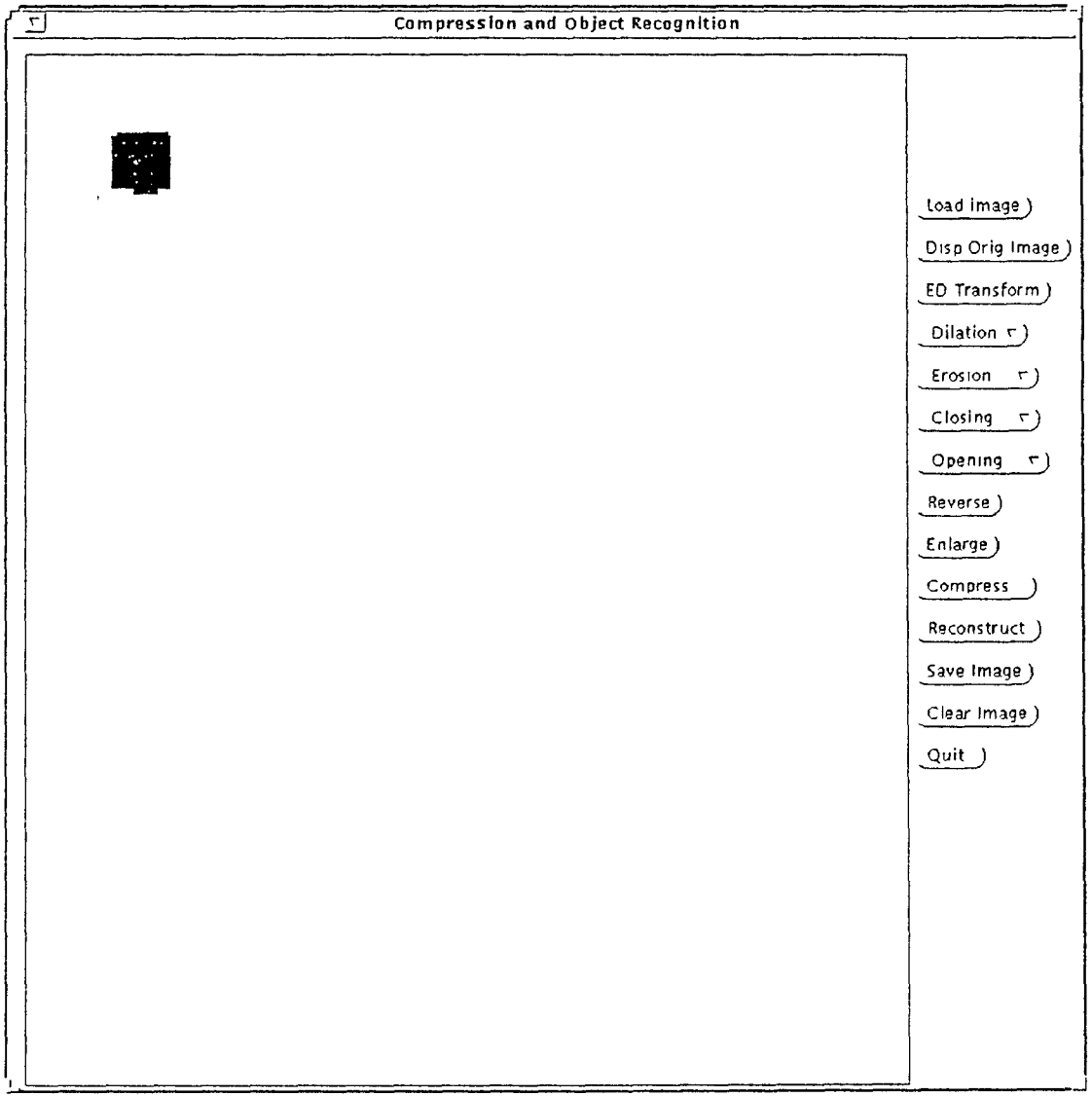


Fig 5: After loading image

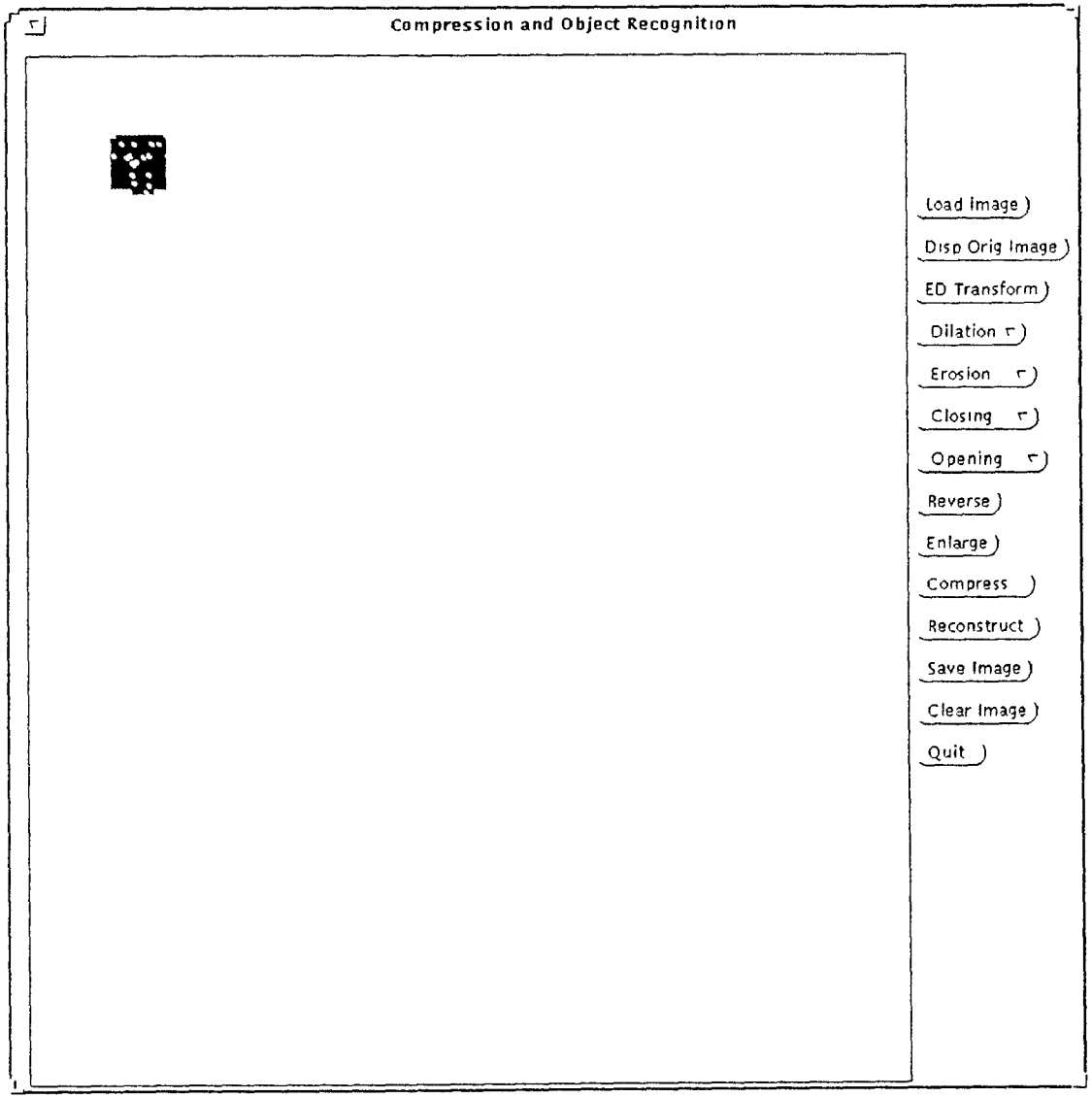


Fig 6: After erosion with 3x3 structuring element

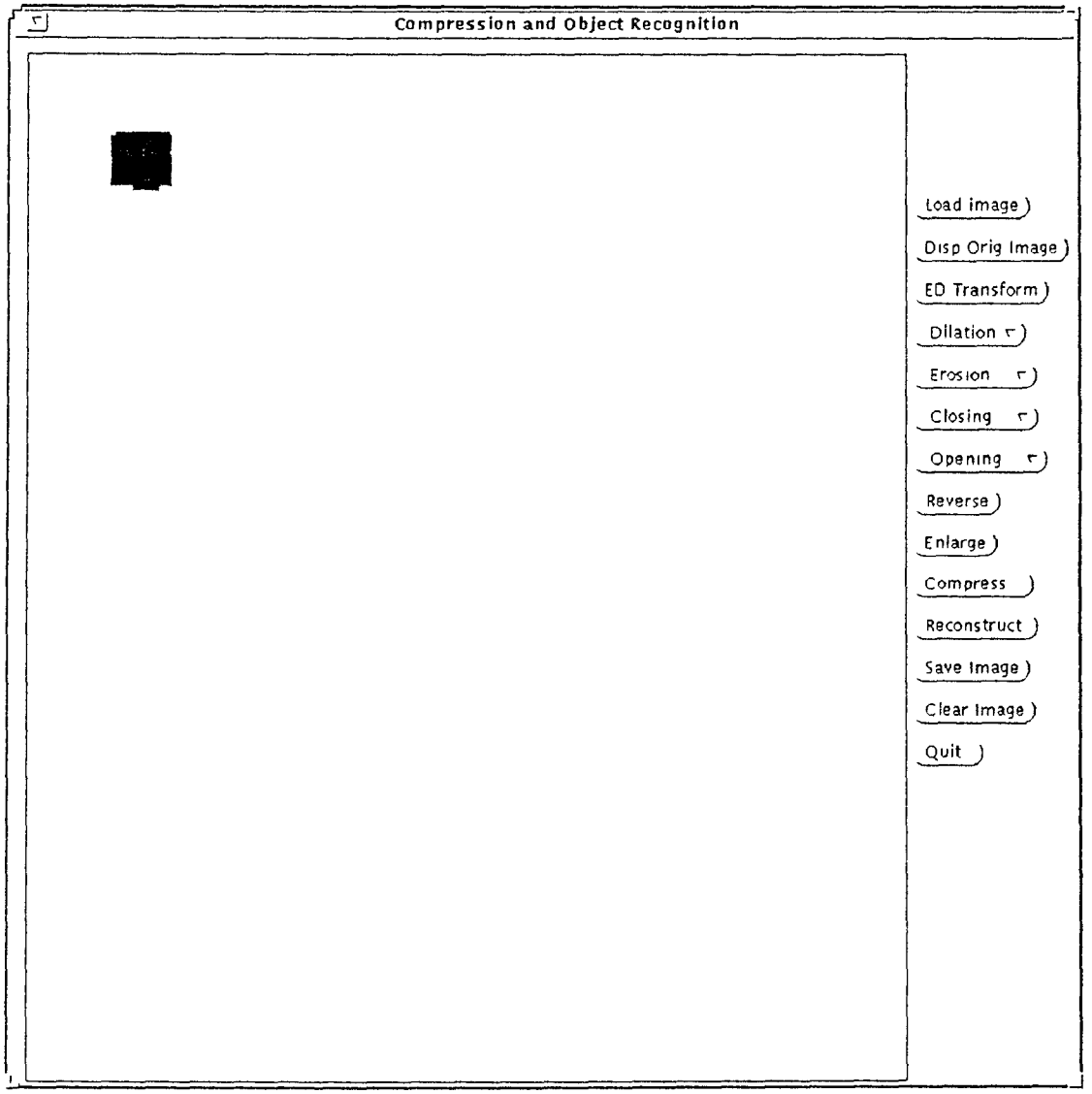


Fig. 7: After dilation, i.e after closing the original image

File : RES

58 58
28 31 435
10 47 13
12 13 9
44 13 9
44 49 9
9 17 7
49 40 5
8 21 4
8 41 4
14 50 4
16 12 4
40 12 4
40 50 4
46 17 4
46 45 4
50 28 4
50 31 4
50 34 4
17 50 2
47 42 2
50 37 2

Compressed data where the square of the radius of the disk is greater than 2

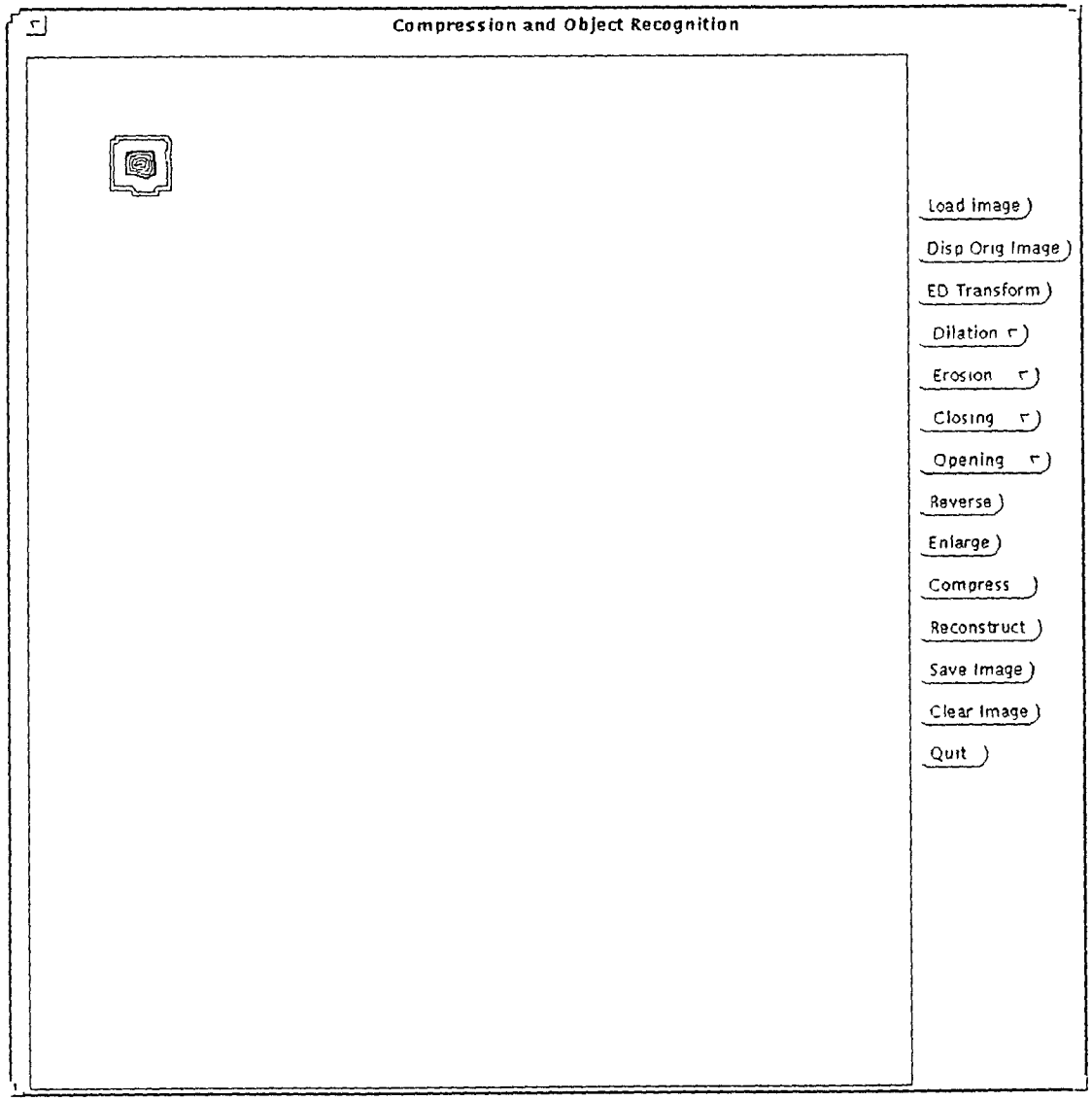


Fig. 8: The Euclidean Transform

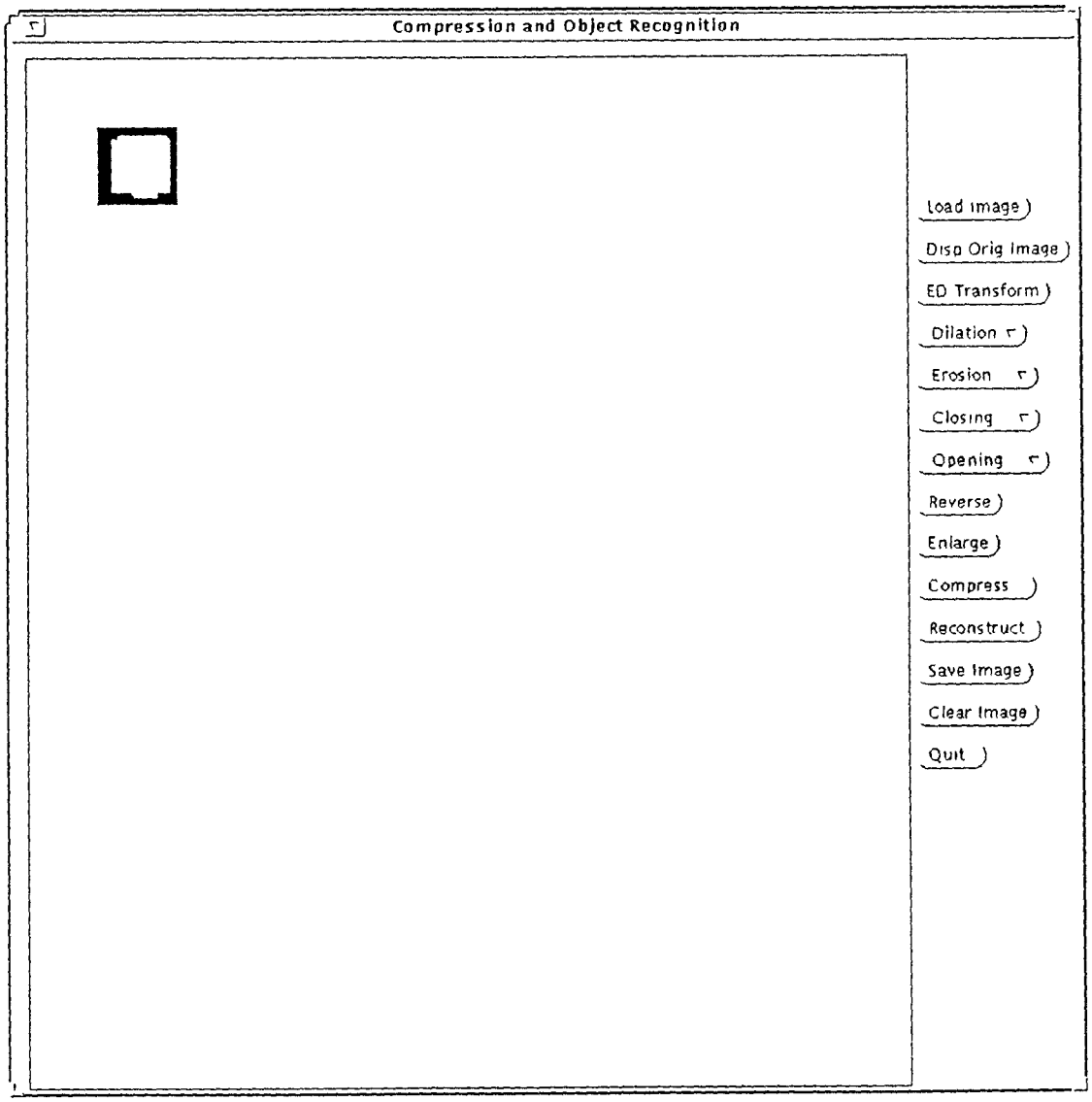


Fig 9: Reversed image

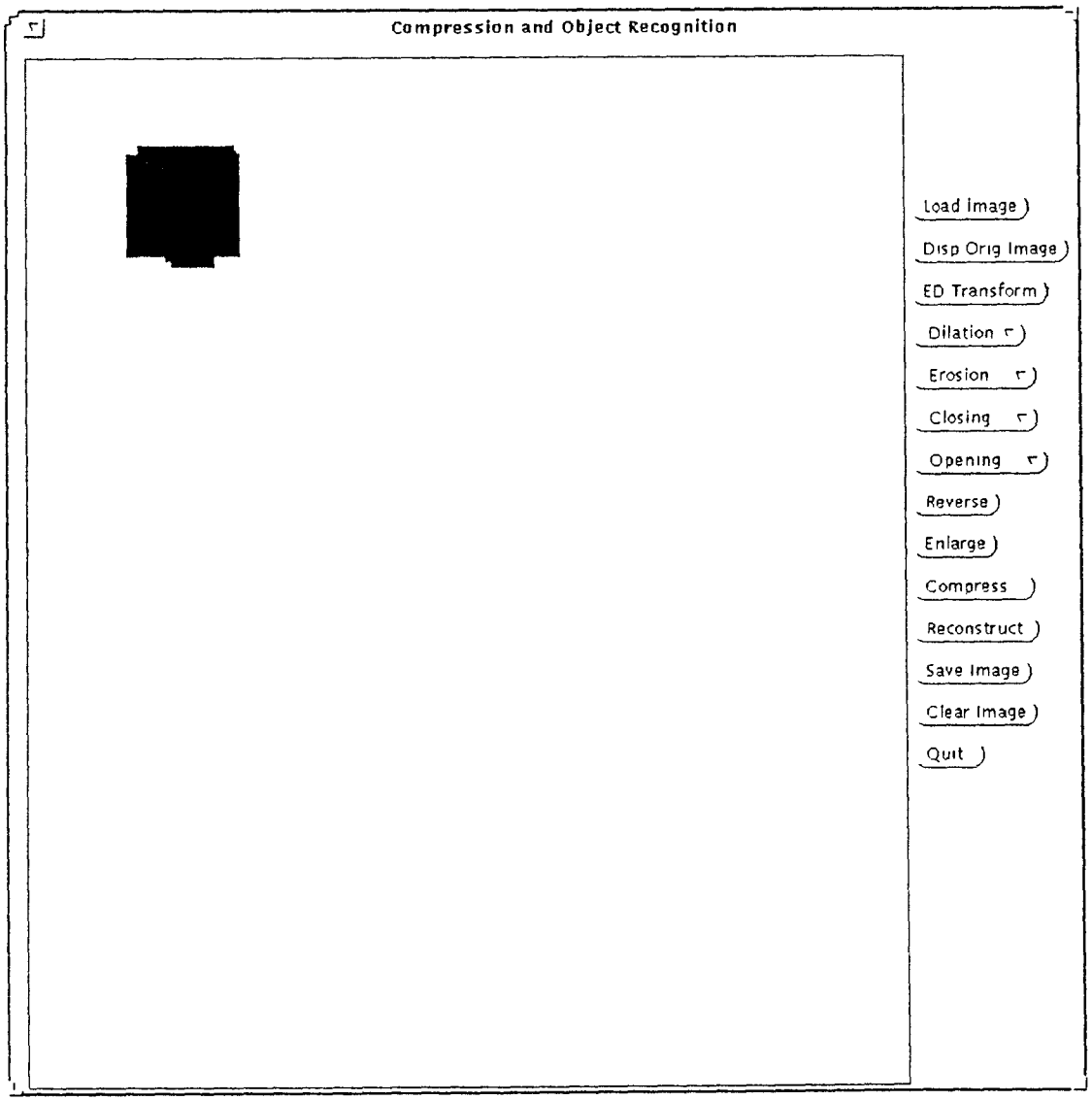


Fig. 10: Enlarged image

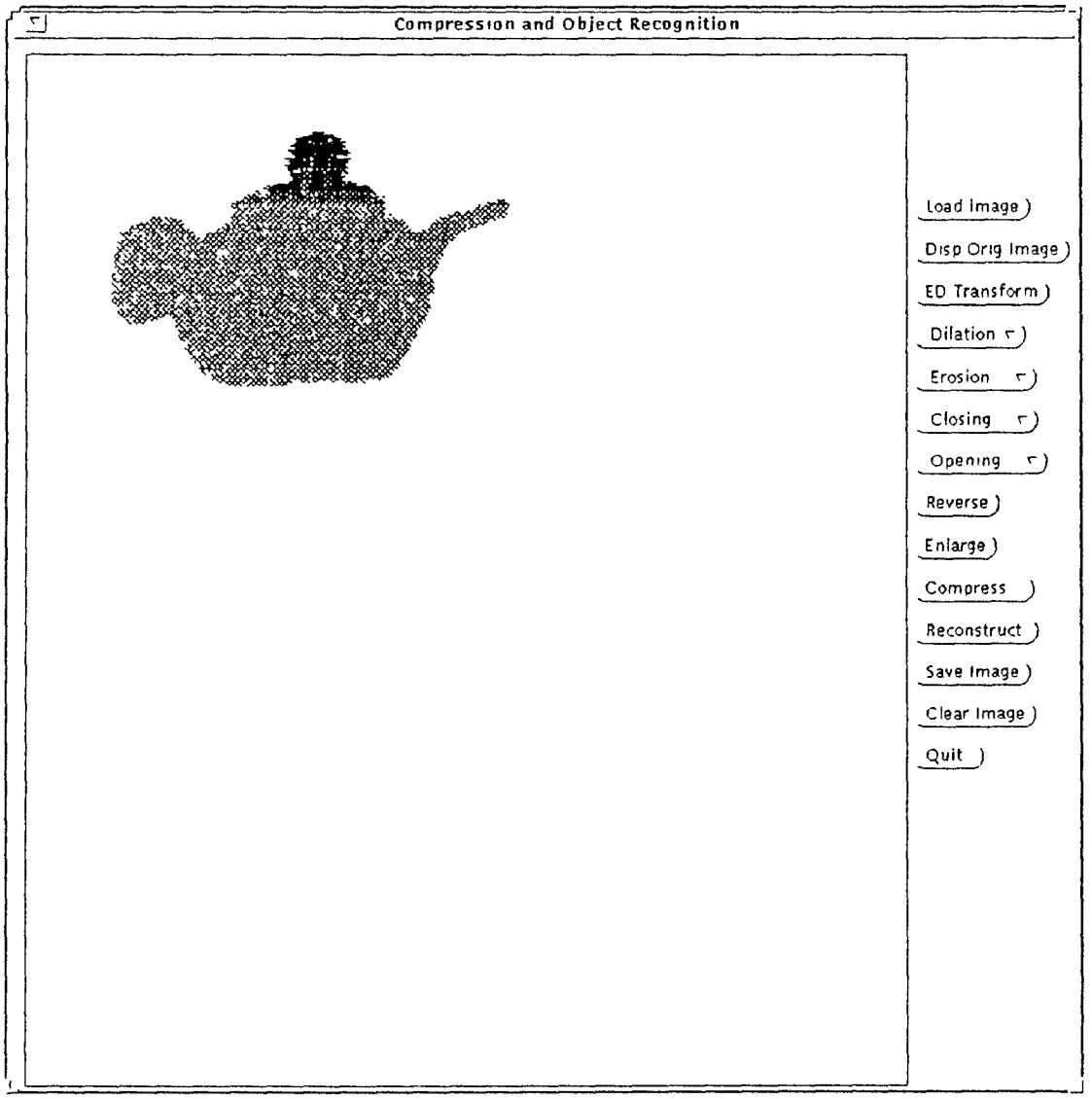


Fig. 11: Loaded noisy image 'Kettle'

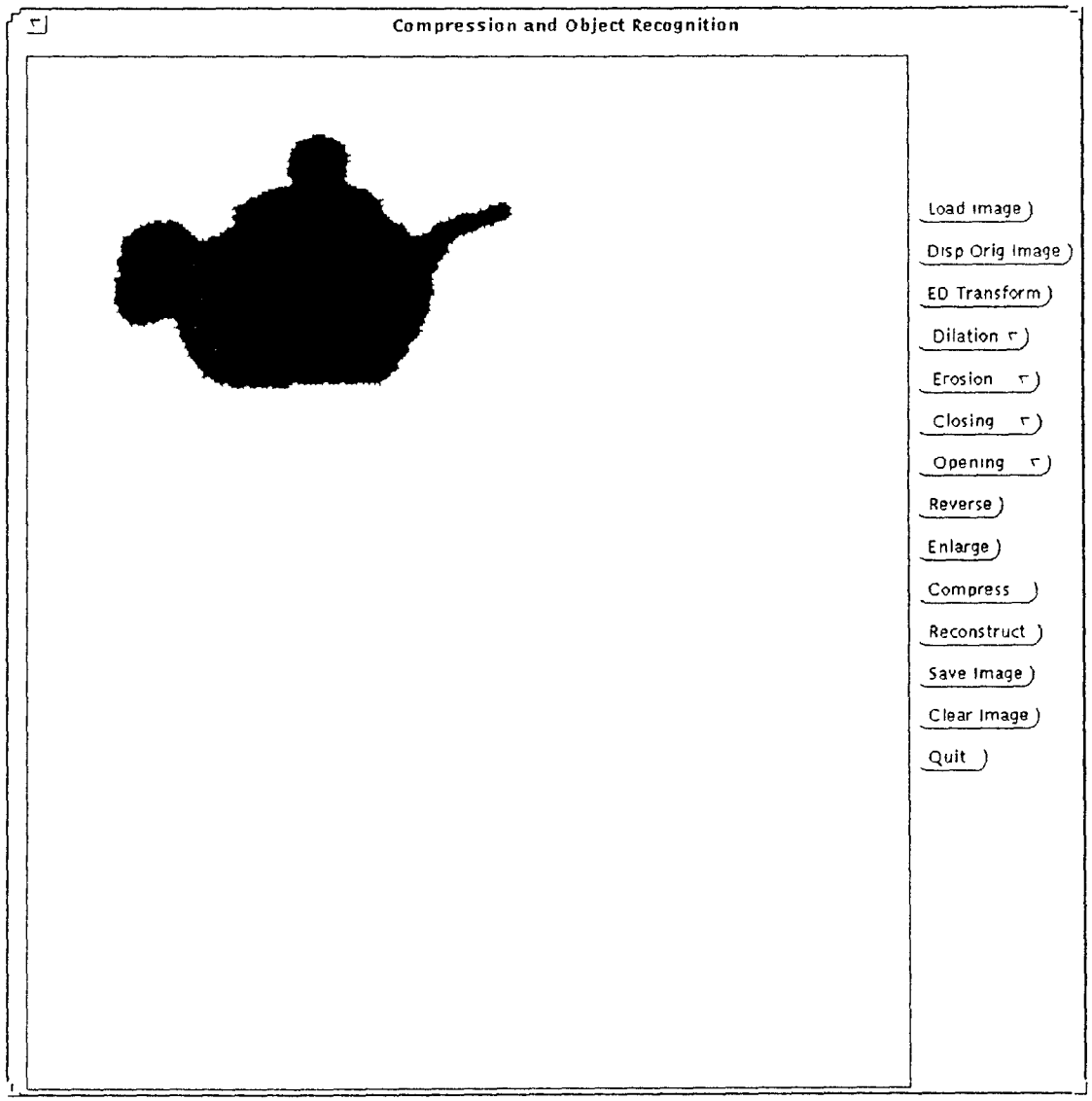


Fig. 12: After closing with 5x5 structuring element

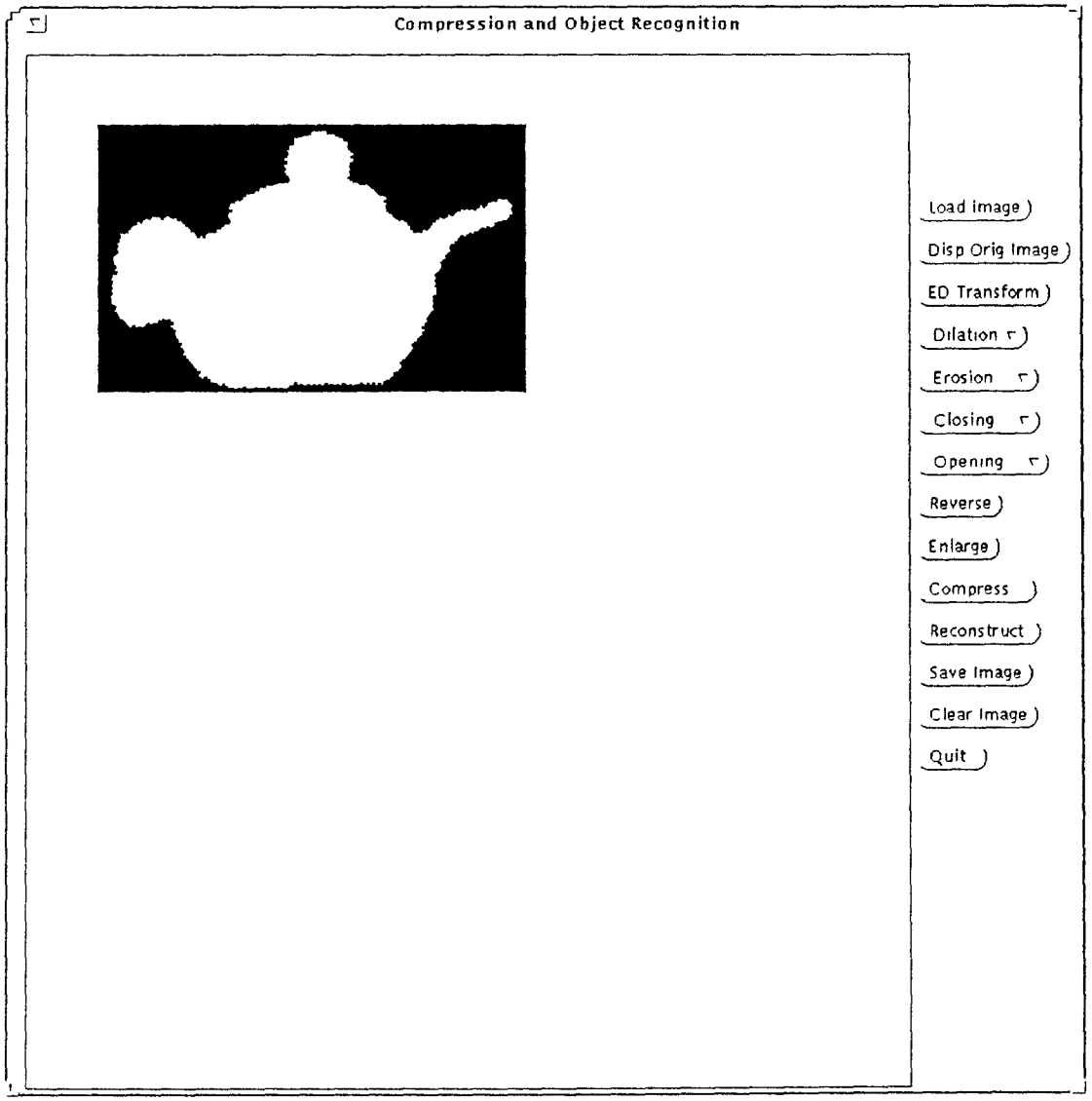


Fig. 13: The reversed image

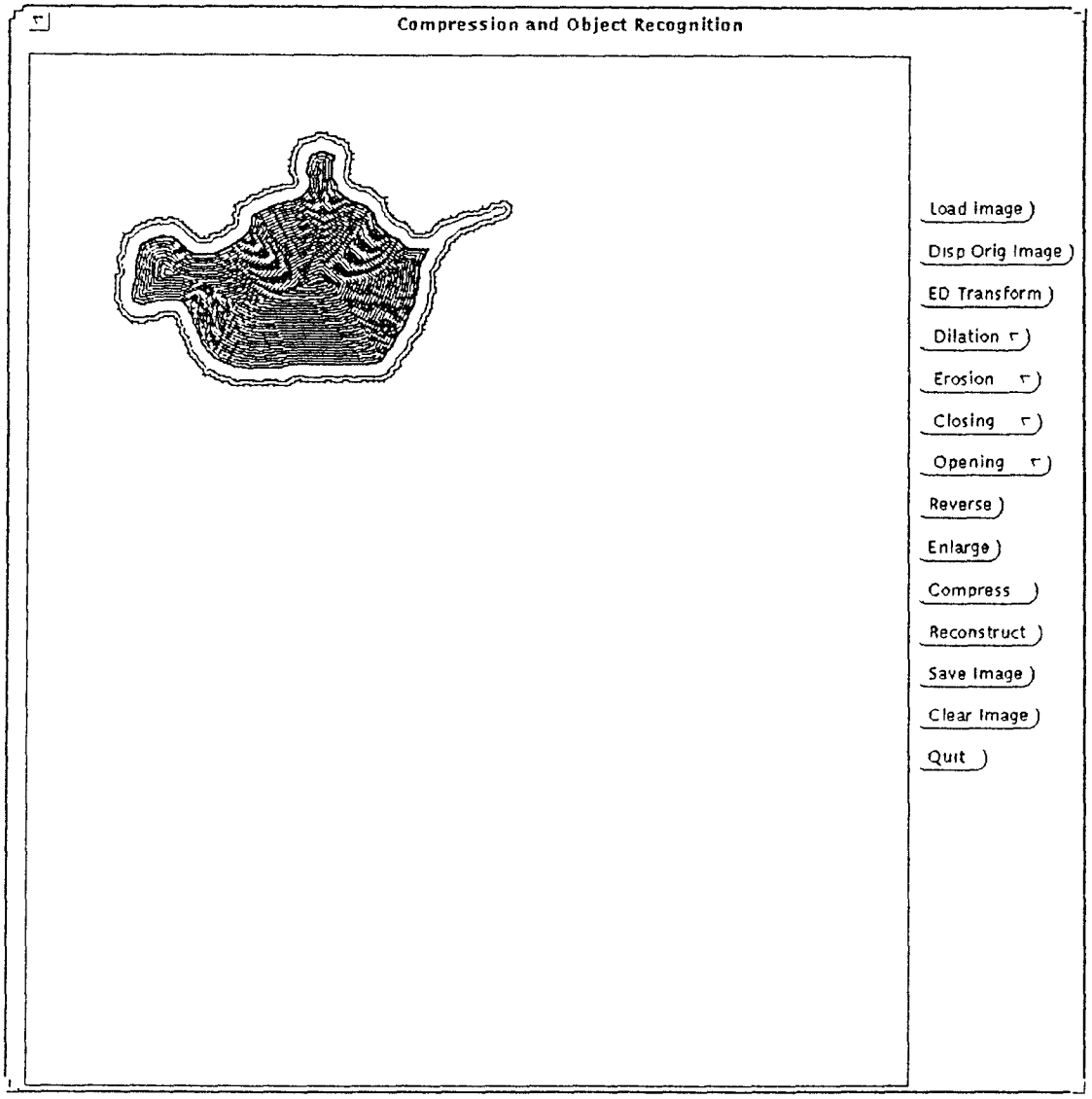


Fig. 14: The Euclidean Distance Transform

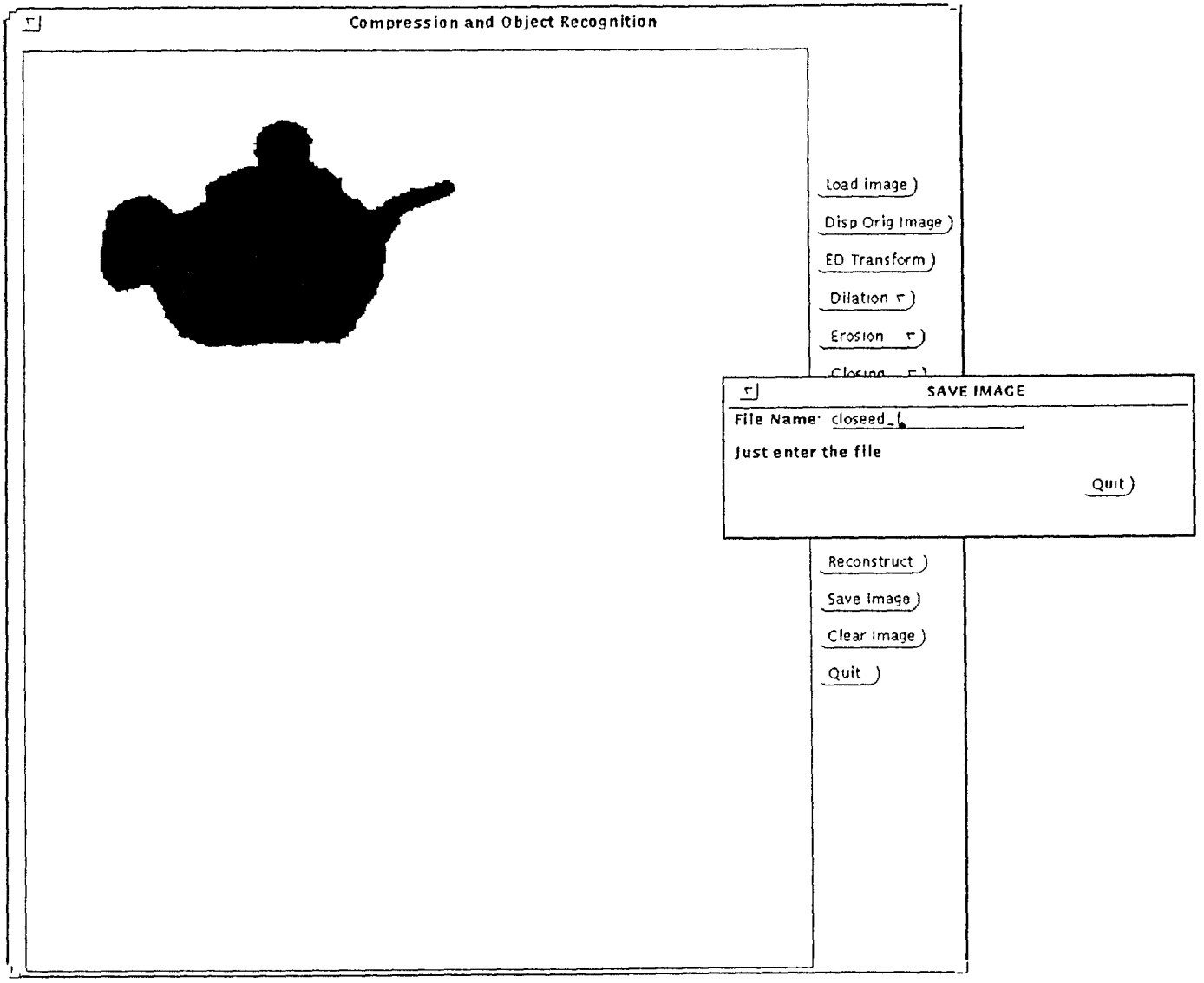


Fig. 15 Saving the closed image

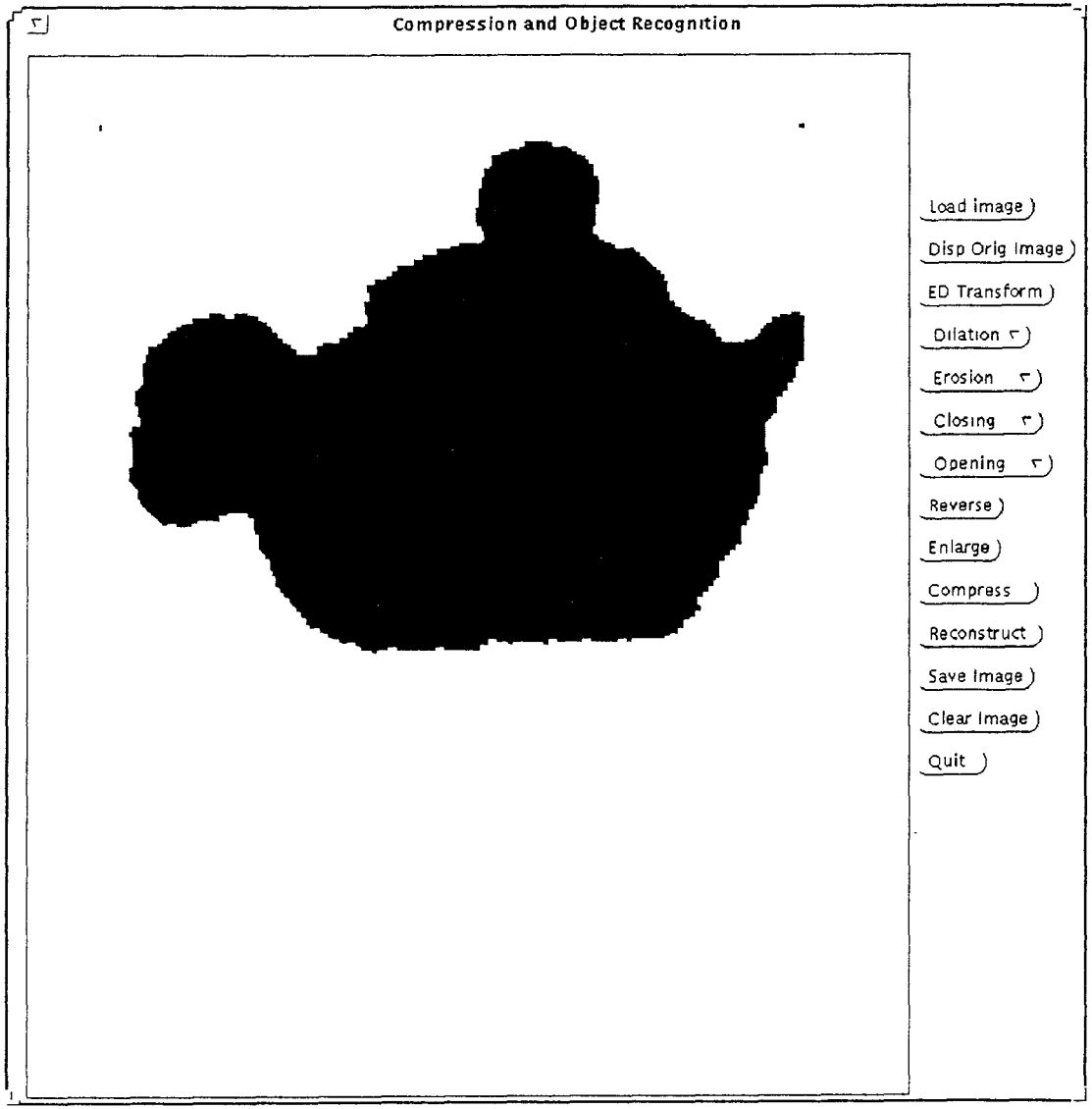


Fig. 16: After enlargement of the opened image by 9x9 structuring element