IMAGE ANALYSIS USING MATHEMATICAL MORPHOLOGY

by

KYUNG HYUN YOO

B.S., Hanyang University, KOREA, 1980

A REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

ELECTRICAL ENGINEERING

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1989

Approved by:

D.V. Satish Chandra

Major professor

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: Introduction

The word morphology refers to the study of form and structure. The morphological analysis of black-and-white images was initiated by Georges Matheron[1] in the mid 1960's, in his study of porous materials. Mathematical morphology provides an approach to the processing of digital images in terms of some predetermined geometric shape known as a structuring element. In mathematical morphology we study the manner in which the structuring element fits into the image. Therefore, it has the intrinsic ability to quantitatively analyze object shapes in both two and three dimensions.

Mathematical morphology treats images from the point of view of set theory. Geometrically it distinguishes itself from other image processing techniques such as syntactic techniques and signal processing techniques. Syntactic technique is based on generative grammars. These grammars establish a set of production rules which will produce the shape from certain symbols. In general these grammars tend to be quite complex for representing global properties of shape. Signal processing techniques make use of Fourier and other orthogonal transformations for image analysis. Just as a one-dimensional signal can be represented by an orthogonal series of basis functions, an image can be represented by a series of two dimensional basis functions called basis images. These basis images can be generated by unitary matrices. The series coefficients in orthogonal series expansion can be used for image processing.

In mathematical morphology, an image will be considered as a set of points and the operations, which are based on logical relations between pixels, rather than arithmetic ones, come from set theory. These are dilation and erosion, and relate directly to shape. It can be used in the areas of noise cleaning, image enhancement, feature extraction and shape analysis. Morphological operations are non-reversible. In other words, morphological operations can simplify image data which has usually too much information, preserving their essential shape characteristics, and eliminating irrelavancies. Therefore, shape description by mathematical morphology can also provide techniques suitable for image coding, that permit image transmission at low bit rates.

The main purpose of this project is to develop algorithms and software for image analysis using morphological operations. These programs are written on the Electrical and Computer Engineering Department's AT&T 3B2 computer, using the C language. The effectiveness of these moporphological operations in image analysis applications such as noise cleaning, edge detection, region filling and image representation is also studied.

CHAPTER 2: Mathematical morphology

The objectives of computer vision or image processing are often to segment an image into objects and textures and to extract certain information for image understanding or classification. Mathematical morphological techniques are based on the analysis of images in terms of some predetermined structuring elements. Mathematical morphology provides an algebraic formulation for applying the structuring elements to the image. Both the image and the structuring elements are considered as sets of points and the sequence of different structuring elements applied to an image gives the information about the geometric measurement of the image. Such knowledge will greatly depend on the choice of structuring elements.

An image can be considered as a set, X, of points or pixels. With each set of points of x of the space E, a set B(x) called a structuring element can be chosen. Every set X in E can be modified by some B(x) in several ways.
The most important ones are as follows;

$$\text{dilation of } X = \{\, x \mid B(x) \uparrow X \,\}$$

$$\text{erosion of } X = \{ x \mid B(x) \subset X \}$$

The dilation of X by B(x) is the set of all the points x such that B(x) hits (denoted by $\Uparrow$) X. The erosion of X by B(x) is the set of all the points x such that B(x) is included in X. This is shown in the Figure 2-1. These two operations will play a major role in morphological analysis in image processing.

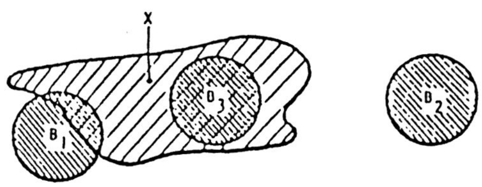


Figure 2-1 B1 hits X ,B2 misses X and B3 is included in X

2.1 Dilation and erosion

The language of mathematical morphology is that of set theory. Sets in mathematical morphology represent shapes. Sets in Euclidean 2-space denote binary images and sets in Euclidean 3-space may denote gray scale images. Sets in higher dimensional space may denote additional image information, such as color, etc. Morphological transformations apply to sets of any dimension. Sets in Euclidean N-space, or its digitized equivalent, the set of N-tuples of integers, denoted by Z will be considered as belonging to E. In the following sections we define some important morphological terms.

DEF 1 : Dilation > Dilation is the morphological transformation which combines two sets using vector addition of set elements. Let A and B be subset of N-space. The dilation of A by B is denoted by $A \oplus B$ and is defined by

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

The dilation operation is commutative and associative,i.e,

$$A \oplus B = B \oplus A \quad \text{(Commutative)}$$

$$A \oplus (B \oplus C) = (A \oplus B) \oplus C \quad \text{(associative)}$$

Using the associative property, dilation an image A by large structuring element, which itself can be expressed as the dilation of B by C, can be computed by successive dilation by B and C. This operation saves much operation time. Dilation by disk structuring elements corresponds to isotropic expansion algorithms common to binary image processing. This is shown in the Figure 2-2 and Figure 2-3.

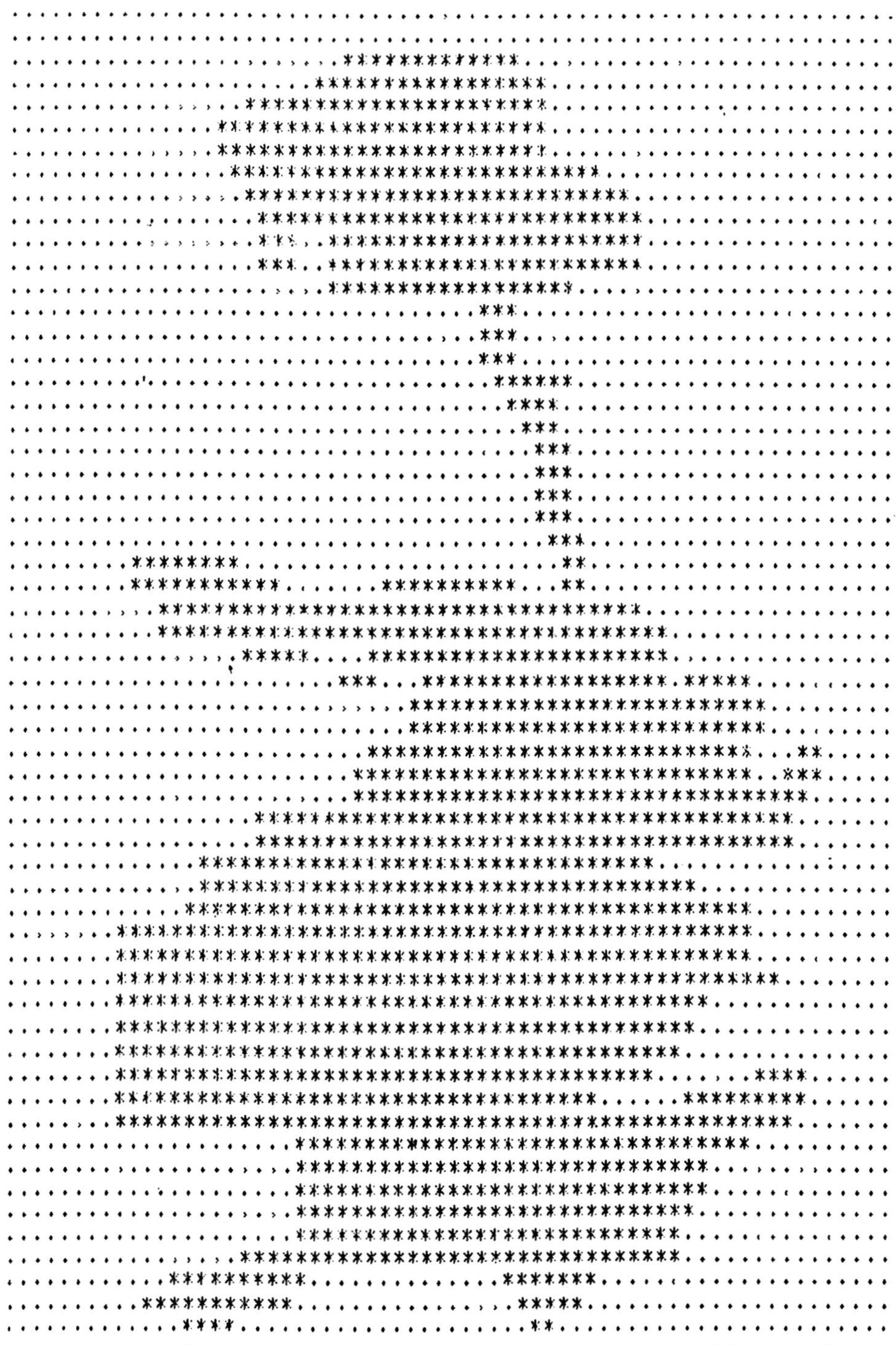


Figure 2-2 Sample image for morphological operations

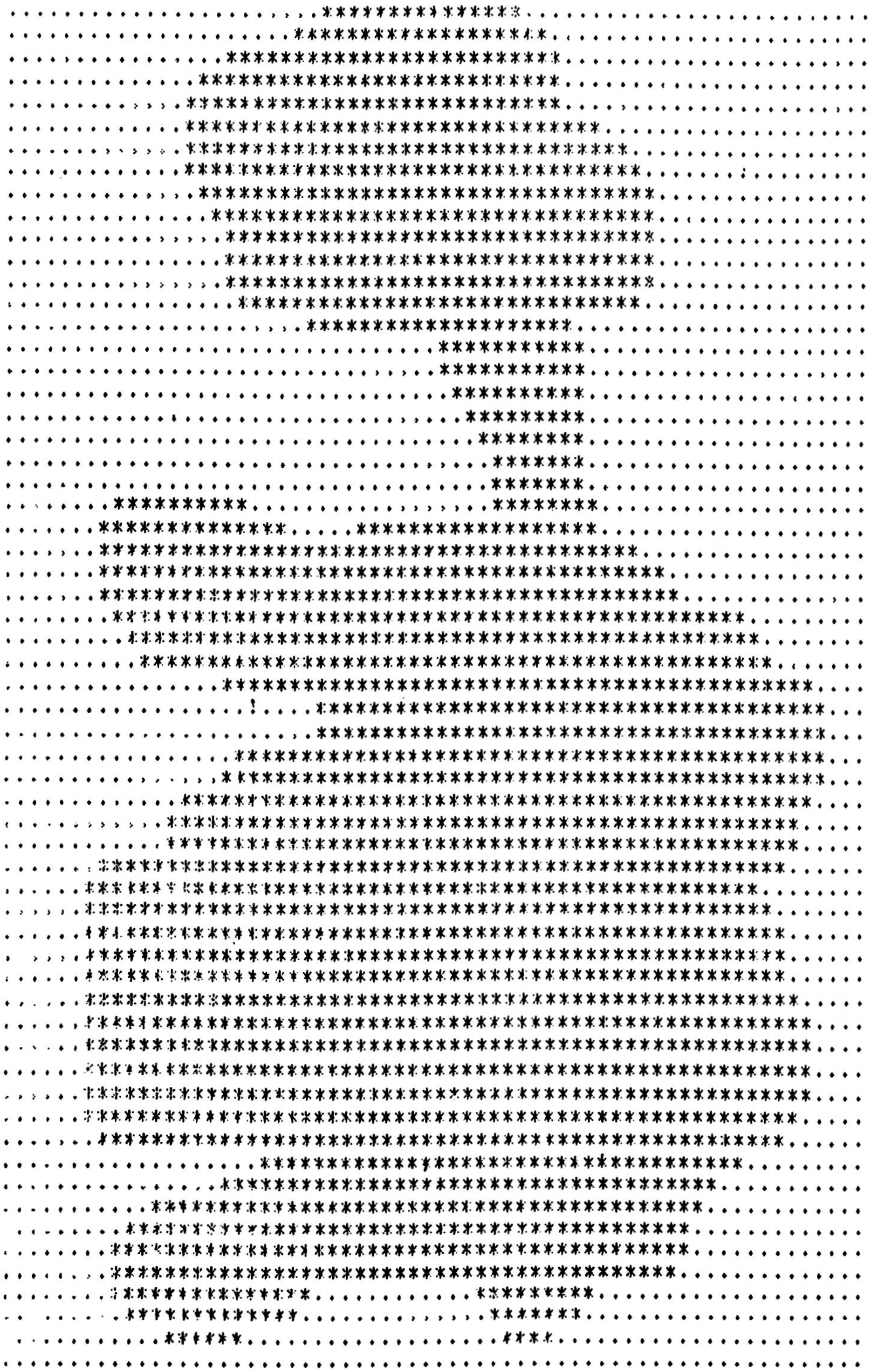


Figure 2-3 Dilation of sample image by circle

DEF 2 : Translation> Let A be a subset of N-space and $x \in E$. The translation of A by x is denoted by $(A)x$ and is defined by

$$(A)x = \{c \in E \mid c = a + x \text{ for some } a \in A\}.$$

It is important to note that dilating a shifted image shifts the dilated image by the same amount. This property is called translation invariance of dilation,i.e.,

$$(A)x \oplus B = (A \oplus B)x$$

The dilation of A by B can be computed as the union of translations of A by the elements of B. From the definition of translation, we can easily show that

$$X \oplus \{t\} = (X)t$$

$$A \oplus (B)t = (A \oplus B)t$$

Since $A \oplus (B \cup C) = (A \oplus B) \cup (A \oplus C)$ it follows directly that

$$A \oplus B = \bigcup_{b \in B} (A)b$$

Using this property dilation operation can be implemented in hardware easily.

DEF 3 : Erosion > Erosion is the morphological dual of dilation. It is the morphological transformation which combines two sets using the vector subtraction of set elements. The erosion of A by B is denoted by $A \ominus B$ and is defined by

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

Structuring element B may be visualized as a probe which slides across the image A, testing the spatial nature of A at every point. Erosion of A by B can be computed as the intersection of all translations of A by the points -b, where $b \in B$.

$$A \ominus B = \bigcap_{b \in B} (A)_{-b}.$$

Like dilation, erosion is translation invariant, i.e.,

$$(A)_x \ominus B = (A \ominus B)_x$$

This erosion operation results in a shrunken image. This is shown in Figure 2-4.

DEF 4 : Reflection > Let $B \subseteq E$. The reflection of B is denoted by $\bar{B}$ and is defined by

$$\bar{B} = \{x \mid \text{for some } b \in B,\ x = -b\}.$$

Erosion and dilation are dual operations with regard to complement. Eroding A is equivalent to taking the complement of the dilation of A^c.

$$(A \ominus B)^c = A^c \oplus B^c$$

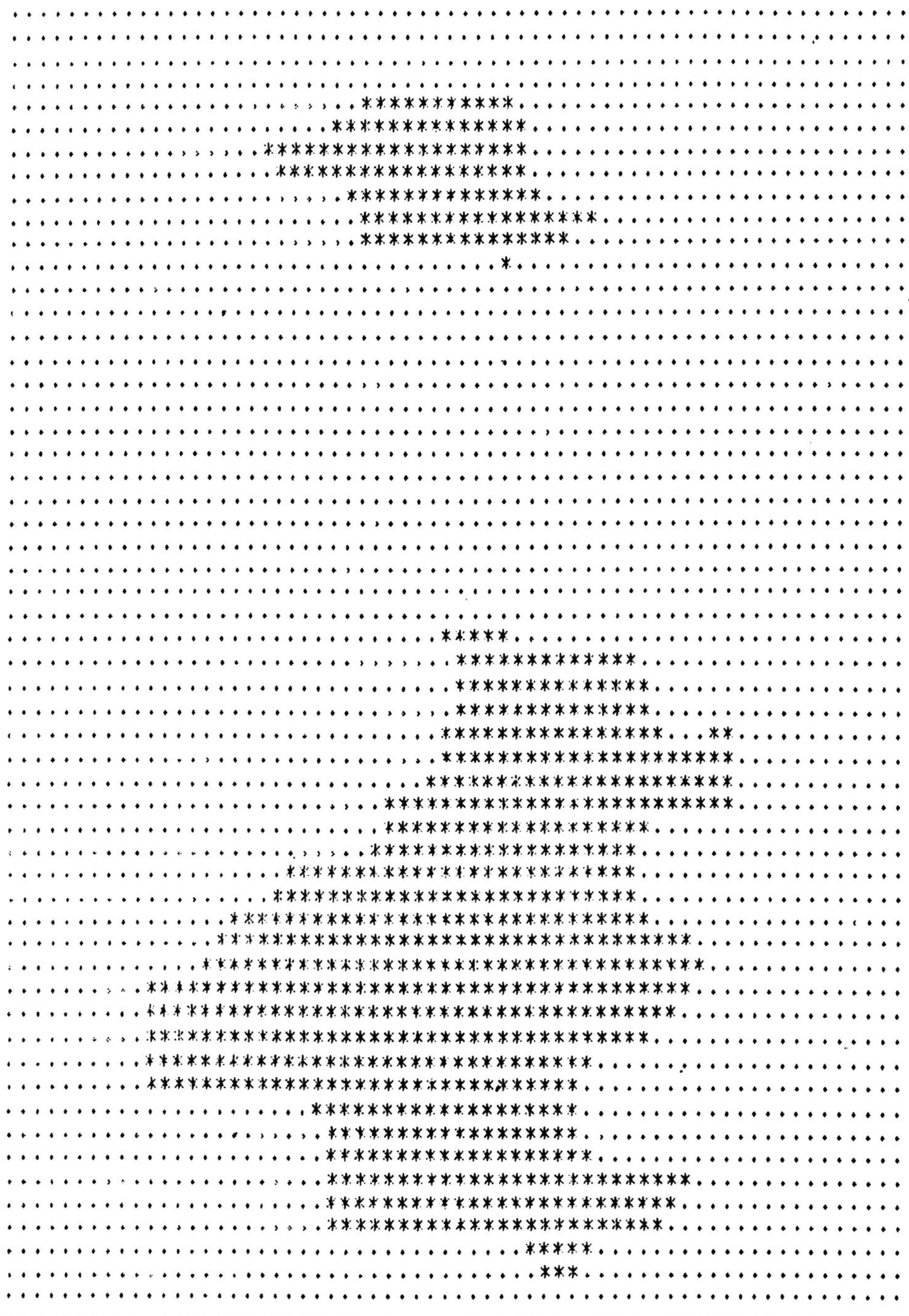


Figure 2-4 Erosion of sample image by circle

2.2 Opening and closing

In practice, dilation and erosion are usually employed in pairs, either dilation of an image followed by the erosion of the dilated result, or image erosion followed by dilation. The result of iteratively applied dilations and erosions is an elimination of specific image detail smaller than the structuring element without the global geometric distortion. For example, opening an image with a disk structuring element smooths the contour, breaks narrow isthmuses, and eliminates small islands and sharp peaks. Closing an image with a disk structuring element smooths the contours, fuses narrow breaks and long thin gulfs, eliminates small holes and fill gaps on the contours. The particular significance of opening and closing is that image transformations employing these operations are idempotent, that is, their reapplication effects no futher change to the previously transformed result.

DEF 5 : Opening > The opening of image A by structuring element B is denoted by $A \circ B$ and is defined as

$$A \circ B = (A \ominus B) \oplus B$$

DEF 6 : Closing > The closing of image A by structuring element B is denoted by $A \bullet B$ and is defined as

$$A \bullet B = (A \oplus B) \ominus B$$

If A is unchanged by opening it with A, we say that A is open with respect to B, while if A is unchanged by closing it with B, then A is closed with respect to B.

Opening and closing are both idempotent also, i.e.,

$$(A \circ B) \circ B = A \circ B$$

$$(A \bullet B) \bullet B = A \bullet B$$

Like erosion and dilation, closing and opening are dual transformations.

$$(A \bullet B)^c = A^c \circ \bar{B}$$

Figure 2-5 and Figure 2-6 show the opening and closing operations.

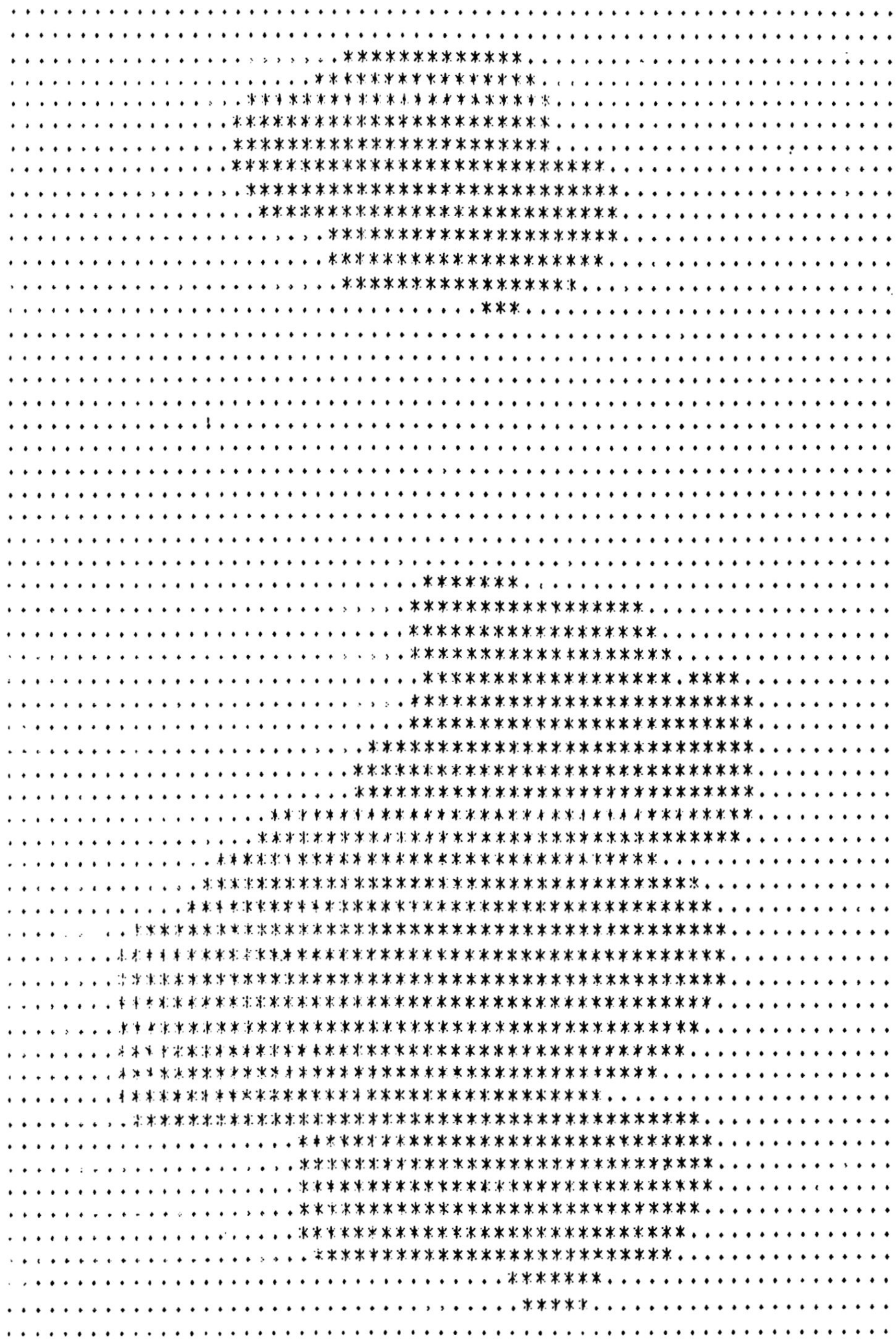


Figure 2-5 Opening of sample image by circle

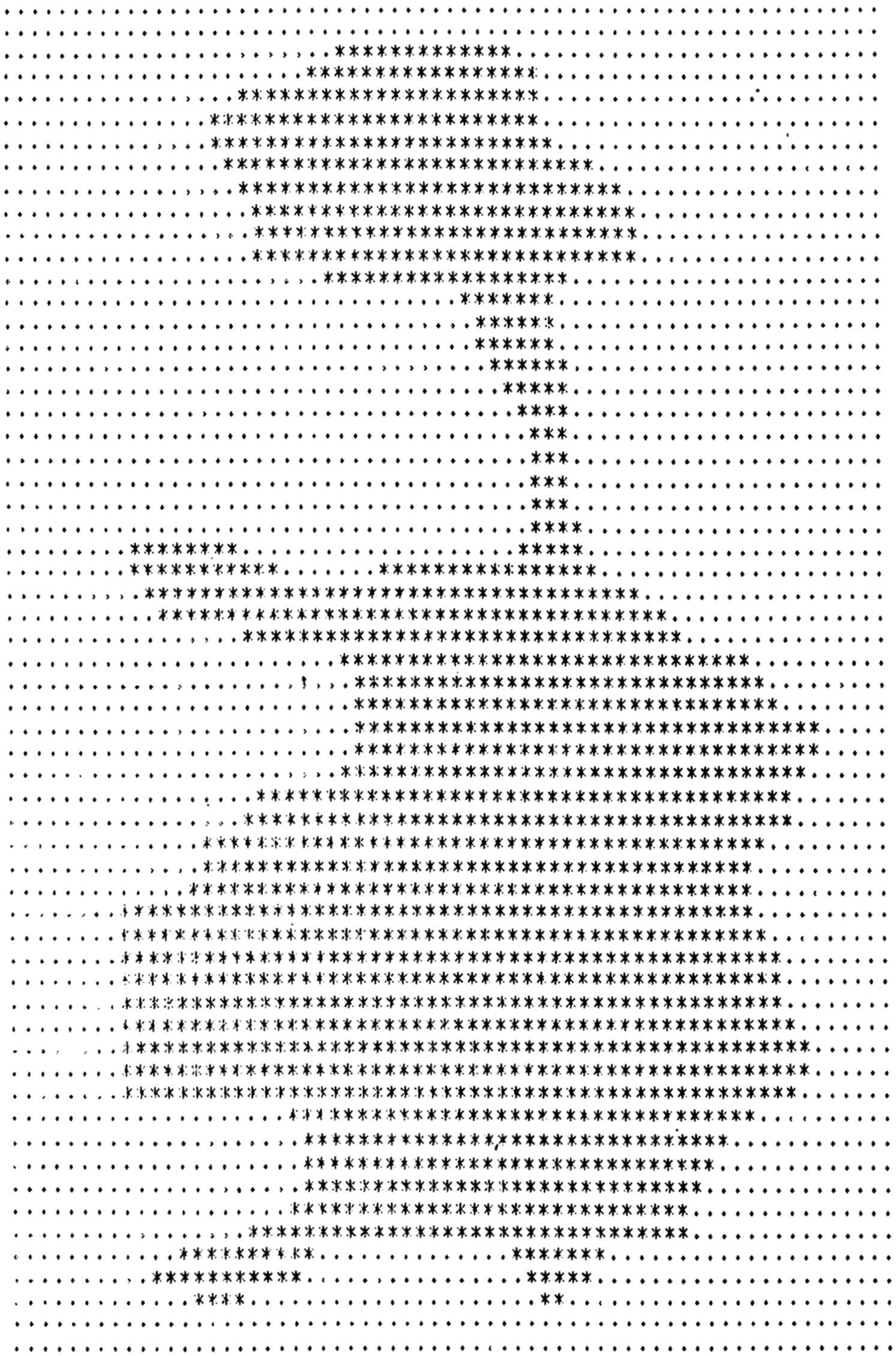


Figure 2-6 Closing of sample image by circle

CHAPTER 3: Digital morphology

3.1 Image definitions

A digital image refers to a two dimensional light-intensity function denoted as f(x,y), where x and y are independent variables denoting the spatial coordinates. The value of f at any point (x,y) is proportional to the brightness of that point. This brightness is called the gray level and the specific coordinates (x,y) are refered to as the picture elements or pixels. In order to generate the computer program of morphological operations, systematic structure of image representation is needed. Matrix representation is a suitable form of digital image representation.

A digital image is obtained by assigning a real number, which refers to a gray level value, to each pixel in some collection of pixels. A digital image f is defined as a function f:D->R, where D is called the domain of the image f, and R is called the codomain. Very often, the domain of a digital image will be rectangular in shape and contains a finite number of elements. In such a case a digital image will be represented in a manner similar to a matrix or a two-dimensional array. Each element of the matrix represents the gray level value of the pixel at that location. The location of a pixel in the matrix is identified by its spatial coordinates.

3.2 Software development

First, imagine two stacks called DOMAIN and RANGE. Each stack contains the same number of entries, the first containing ordered pairs(i,j) and second containing real numbers. Together the stacks implicitly contain an image, for if they were popped simultaneously, the corresponding series of numbers would form a location of pixel together with its gray level. It can be written in the form of a program as follows:

```
typedef struct position
    {
    int x,
        y;
    } POSITION;

typedef struct image
    {
    int         size,
                range[4000];
    POSITION    domain[4000];
    } IMAGE;
```

With this convenient representation, several basic operations can be implemented.

3.3 Fundamental operators

The following six operators can be considered as the fundamental operations which will be applied to each pixel in an image.

EXTMAX - This function compares two images in a pixelwise manner and outputs the maximum, or highest, gray value at each pixel at which both input images are defined.

MIN - This function compares two images in a pixelwise manner and outputs the image which is an intersection of the domain instead of their union. The resulting image has the pixels with lowest gray value.

TRANS - This function has the image of f and two integer i and j as inputs and the image that is identical to f but moved over i pixels to the right(along x-axis) and j pixels down(along y-axis) as output.

NINETY - This function leaves the gray values of an input image intact while altering the domain of the image. **NINETY** rotates an input image 90 degrees in the counterclockwise direction about the origin. (In this report, this function will be used only for structuring elements.)

SUB - This function subtracts an image from an other image. When we subtract an image B from an image A, **SUB** generates same image as A except those pixels of which domain is same in image B.

COMP - This function generates a complementary image of the input image.

Now, the basic operations, dilation and erosion, in mathematical morphology can be obtained from these fundamental functions. Suppose A represents an image and B represents a structuring element. Dilation and erosion can be expressed by the following equations:

$$\text{Dilation} : A \oplus B = \bigcup_{b \in B} (A)b$$

$$\text{Erosion} : A \ominus B = \bigcap_{b \in B} (A)-b$$

From these two expression, dilation and erosion can be implemented as in the following block diagram of Figure 3-1. Based on the definitions of opening and closing, these operations can be implemented as shown in the block diagram of Figure 3-2.

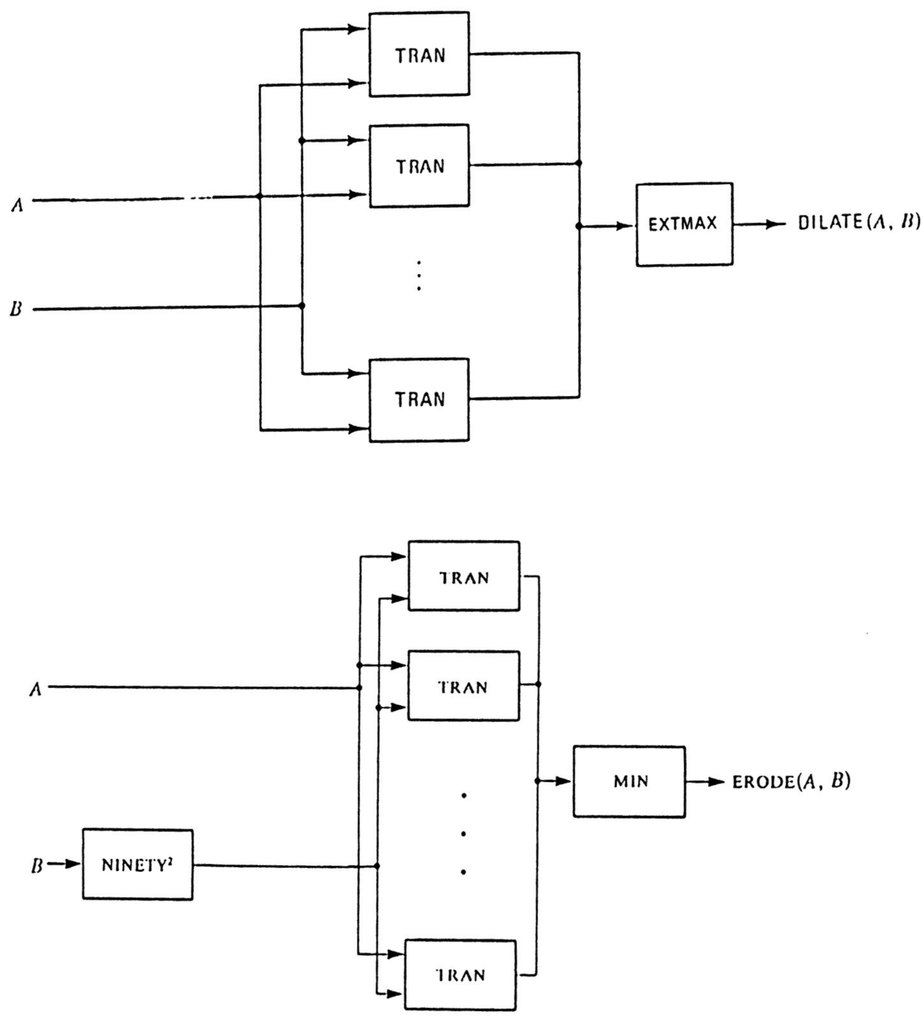


Figure 3-1 Block diagram of dilation and erosion

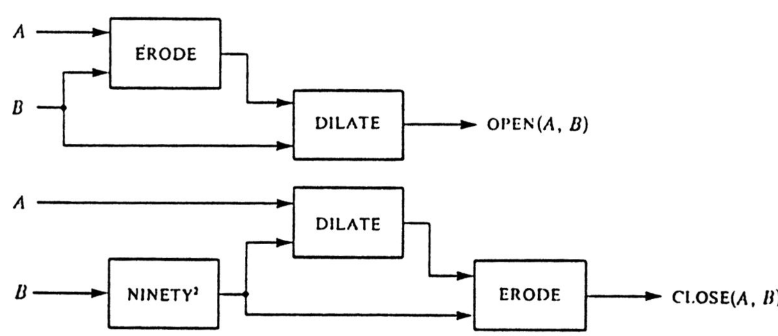


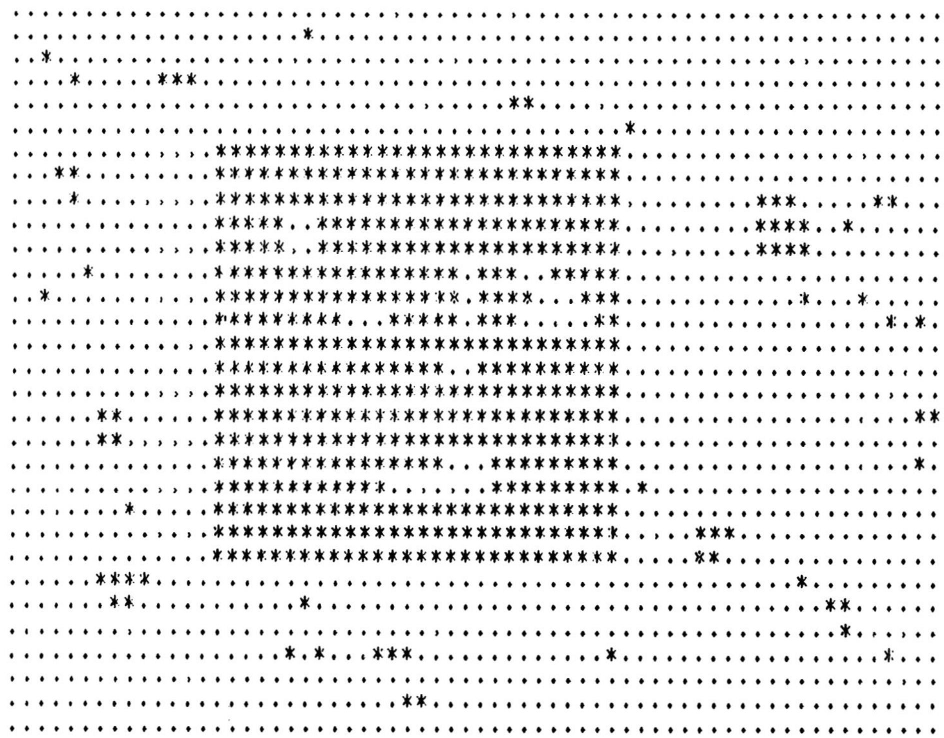
Figure 3-2 Block diagram of opening and closing

CHAPTER 4: Image analysis applications

In this section some applications of morphological methods in digital image analysis are presented. These applications include noise cleaning, edge detection, region filling and image representation.

4.1 Noise Cleaning

The opening of an image A by a convex set B cuts down the peaks of A, whereas the closing of A by B fills up the valleys of A. For an image contaminated by salt-and-pepper noise, closing and opening operations can remove this noise. Opening operation suppresses the background noise spike, and closing after opening (X) cleans interior noise spike. For removing background noise spike, larger structuring element is better than smaller one. However, large structuring element may cause unacceptable distortion. So, the noise cleaning operation depends on the choice of the structuring element. It is reasonable that a large spot in backgroud is not considered as noise because it is bigger than the structuring element. The noise cleaning operation is shown in Figure 4-1 and Figure 4-2.



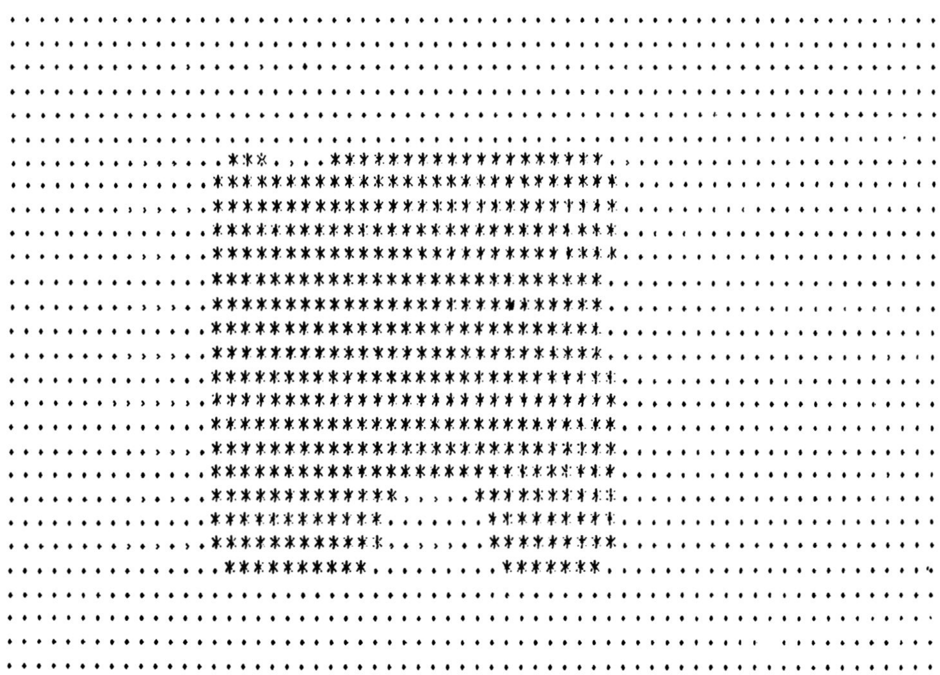


Figure 4-1 Sample image with noise and image cleaned by circle

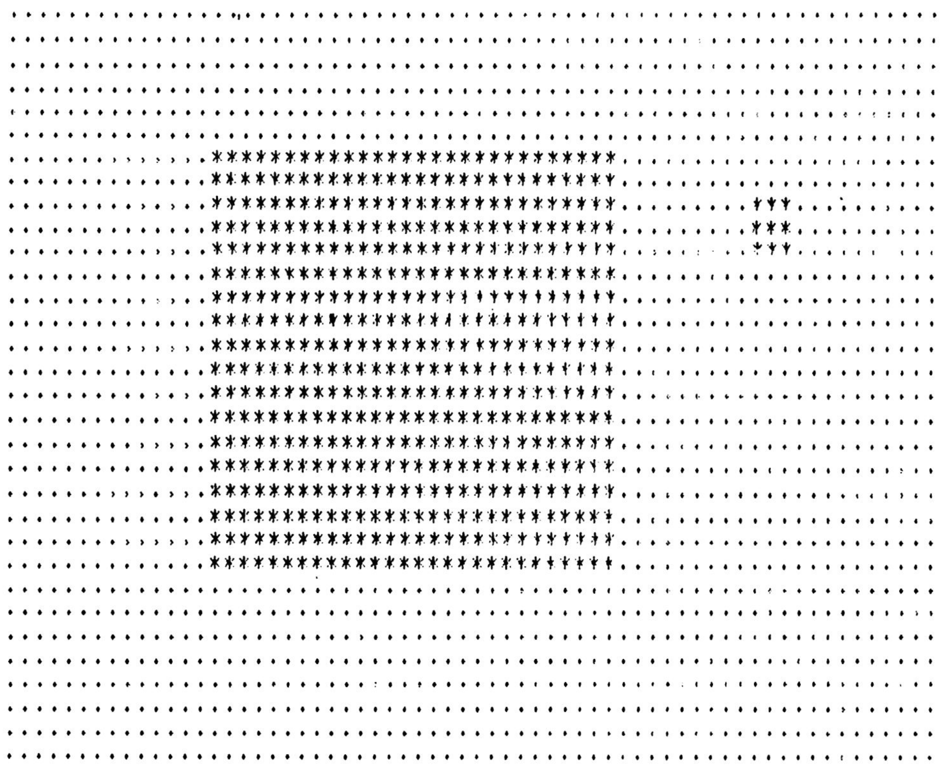


Figure 4-2 Image cleaned by square

This closing-opening operation is comparable to median filtering. However, it needs less computation than median filtering.

4-2 Edge detection

Consider a structuring element B of unit size. Then nB denotes a structuring element of size n obtained by dilating B by itself n times. If B is symmetric, the erosion of A by B denotes a shrunken image of A. Again, the image difference A - $(A \ominus B)$ gives the boundary of a binary image. If we use nB for erosion, orientation and size of n will determine the orientation and thickness of boundary. The edge detection process is illustrated in Figure 4-3 and Figure 4-4.

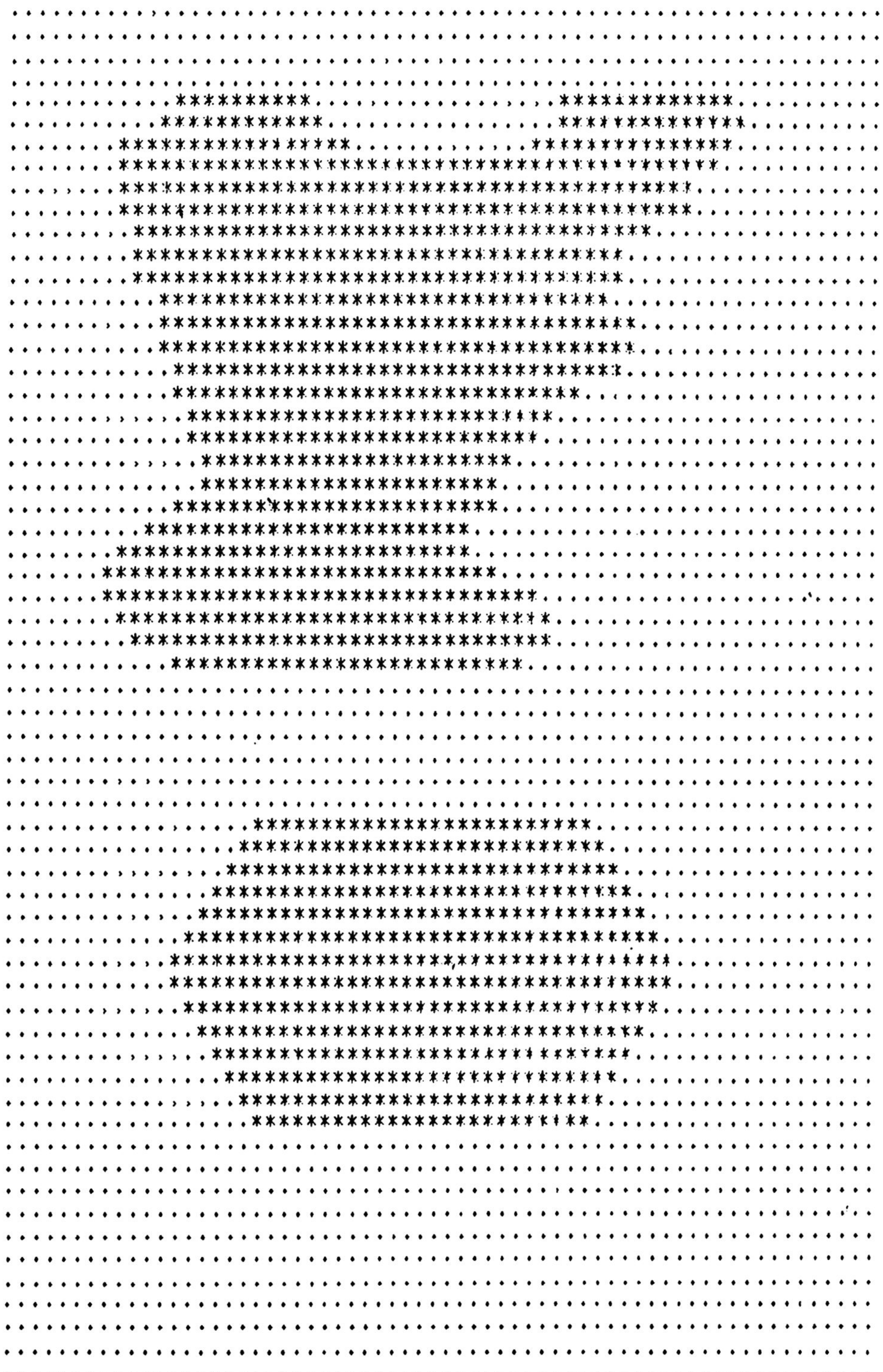


Figure 4-3 Sample image

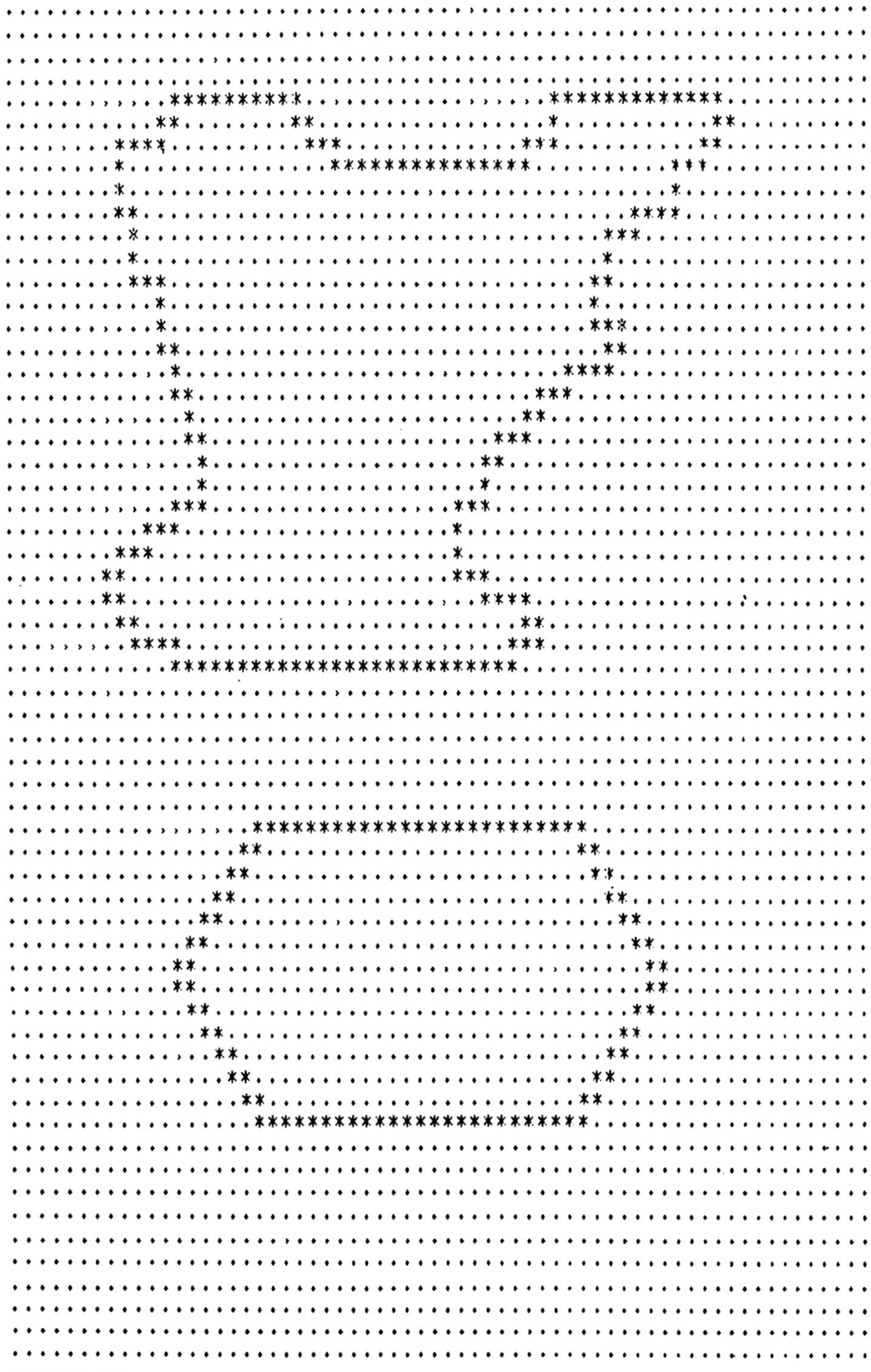


Figure 4-4 Edge image of sample in Figure 4-3

4-3 Region filling

This operation requires only dilation and intersection. Suppose we have an image A which is the boundary of two disjoint regions and we know a point P inside one of two regions. After dilating the point P by a small symmetric and convex structuring element, intersect this intermediate result with X^c(Figure 4-5). This X^c limits the result of dilating effect inside the region. Iteration of dilation and intersection will make the image fully filled. Figure 4-6 and Figure 4-7 show the results after 10 and 17 iterations of dilation respectively. It is important that the structuring element should be small with regard to the thickness of boundaries.

4.4 Morphological skeleton and minimal skeleton

The skeleton is a topologically equivalent thinned version of image. The skeleton SK(X) of a continuous image object X, viewed as subsets of 2-D continuous space, is defined as the centers of the maximal disks inscribable inside X. A disk is maximal if it is not properly contained in any other disk totally included in X. Hence, a maximal disk must touch the boundary of the object X at least at two different points. Some examples of skeleton are shown in Figure 4-8. This skeleton is a caricature containing

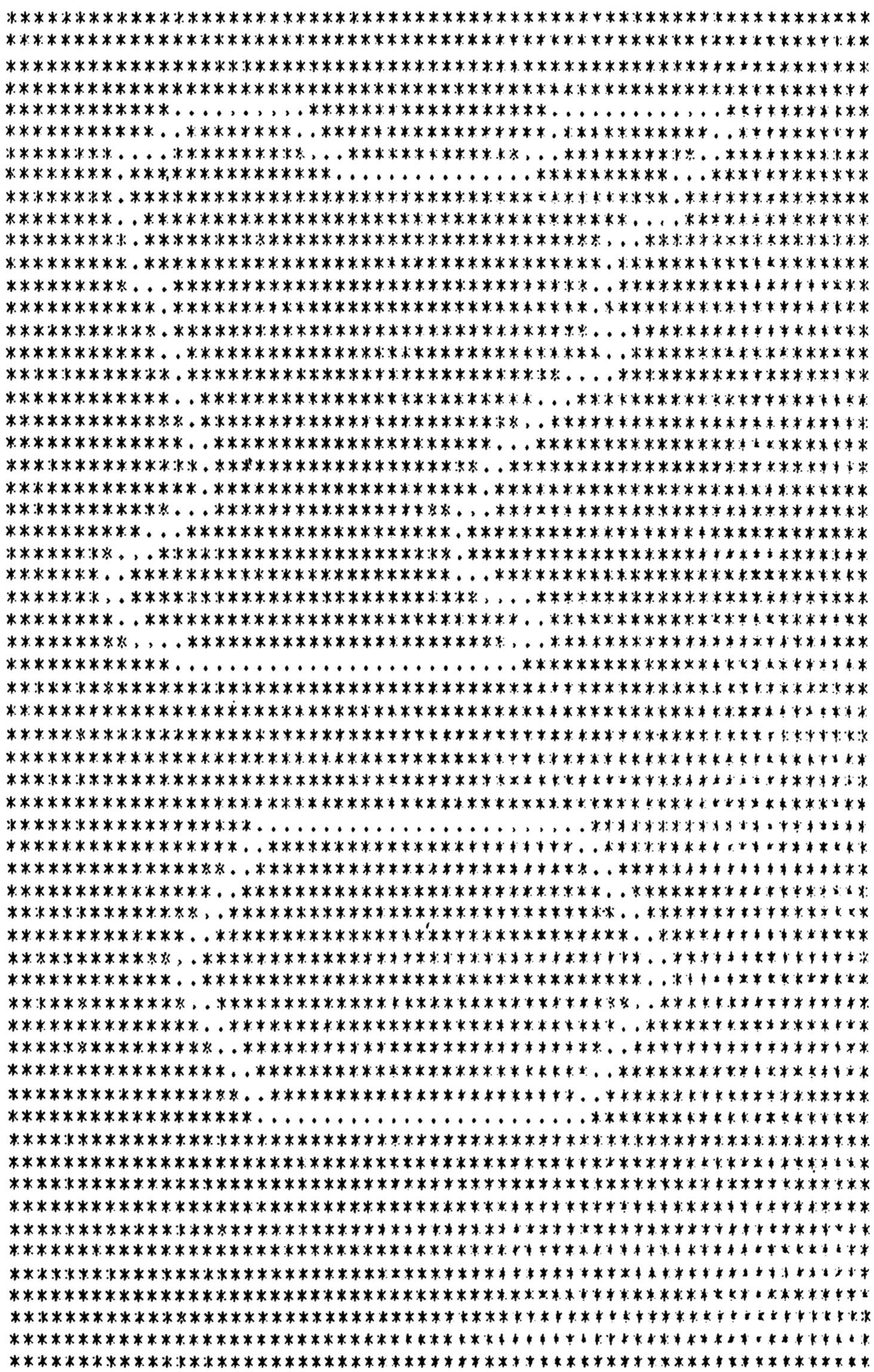


Figure 4-5 Complement of edge image in Figure 4-4

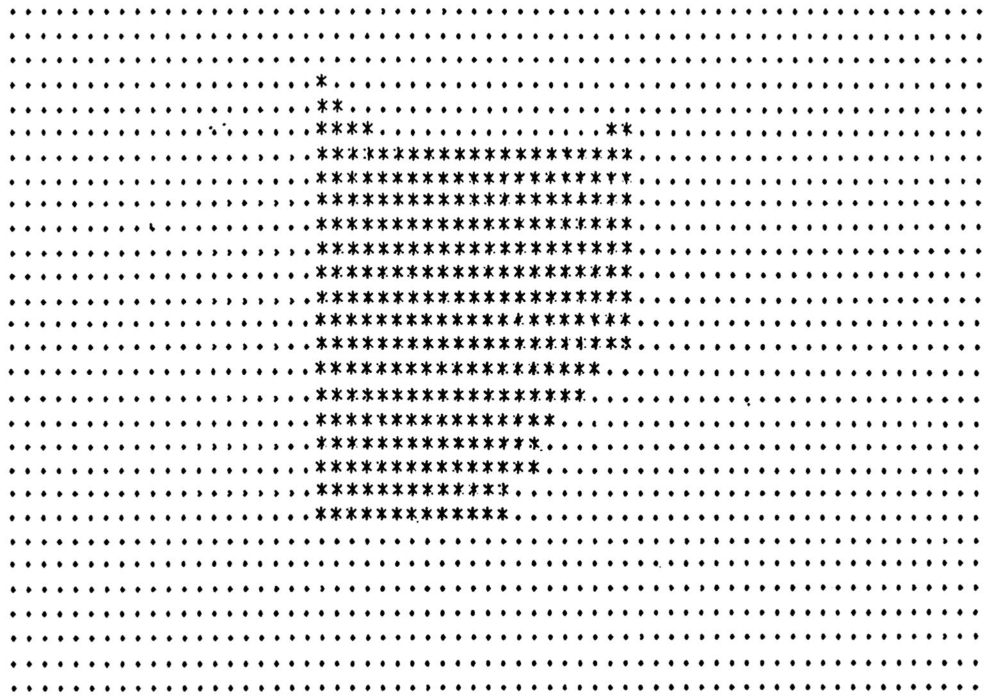


Figure 4-6 Image after 10 iterations of dilation

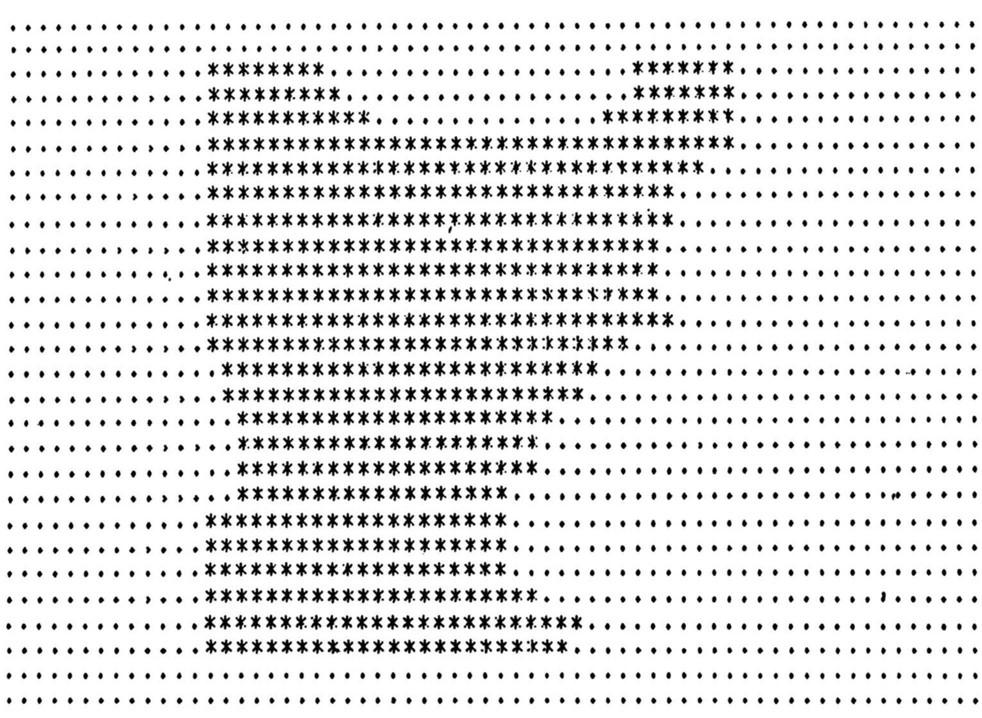


Figure 4-7 Image after 17 iterations of dilation

information about the shape, size and orientation.
The skeleton SK(X) can be obtained from the set union of Sr(X), r > 0, which denotes the rth skeleton subset, i.e., the set of the centers of the maximal disks whose radius is equal to r. These skeleton subsets can be obtained by using morphoplogical erosion and openings. The skeleton SK(X) is equal to

$$SK(X) = \bigcup Sr(X)$$
$$= \bigcup[(X \ominus rB) - (X \ominus rB)drB]$$

where rB denotes the disk of radius r and drB is a disk of infinitesimally small radius dr. The boundaries of the eroded sets $(X \ominus rB)$ can be viewed as the propagating wave fronts of Blum's grassfires where the propagation time coincides with the radius r. Subtracting from these eroded versions of X their opening by drB retains only the angular points, which are points of the skeleton. The original set X can be reconstructed as the union for all r > 0 of the

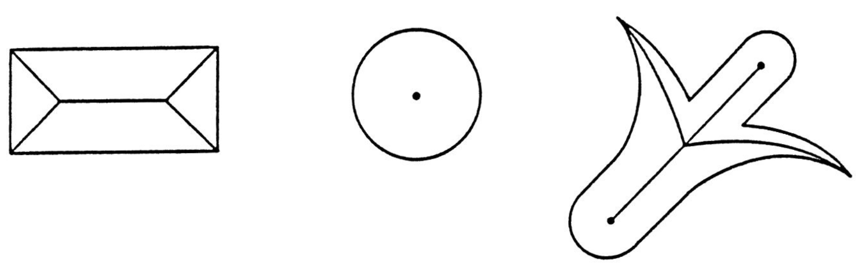


Fig 4-8 Example of skeleton

subsets $Sr(X)$ dilated by the open disks rB, respectively. For digital image Serra[2] provided an algorithm for morphological skeleton $SK(X)$ of a discrete binary image X sampled on a hexagonal grid,

$$Sn(X) = (X \ominus nB) - (X \ominus nB)B$$

$$n = 0,1,2 \ldots,N$$

$$SK(X) = \bigcup_{n=0}^{N} Sn(X)$$

where $Sn(X)$ denotes the nth skeleton subset of X.

The hexagon is very good approximation to a circle, but in rectangularly sampled binary image this algorithm can be used by using symmetric convex structuring elements, such as the CIRCLE, SQUARE in Figure 4-9. If we considered these elements to have a discrete radius one, then, as in the case of the discrete hexagon, we can form similarly shaped elements of discrete radius n. Then nth skeleton subsets is obtained by eroding X by nB, and then keeping from every eroded set $(X \ominus nB)$ only those parts which consist of

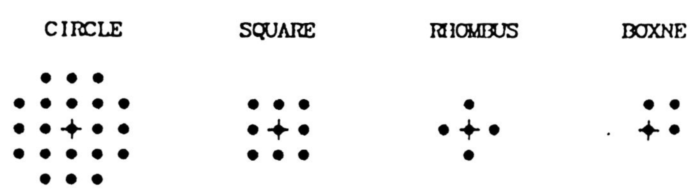


Fig 4-9 Examples of structuring element

angular points and lines without thickeness. These parts are the only ones remaining after the set difference between $(X \ominus nB)$ and its opening $(X \ominus nB)b$.
By the properties of erosion, the erosion of X by nB can be done much faster by successively eroding X by B n times. Using this method, algorithm for skeletonizing of digital image can be described as follows.

Let EROS1, EROS2, OPEN denote three accumulator sets large enough to hold the image object and its background, then skeletonization can be achieved by the following steps;

step1: n = 0, EROS1 = X

step2: EROS2 = EROS1 $\ominus$ B

step3: if EROS2 = $\emptyset$

then N = n, SN(X) = EROS1 and STOP.

step4: OPEN = EROS1 $\oplus$ B

step5: Sn(X) = EROS1 - OPEN

step6: n = n + 1, EROS1 = EROS2

and go to step 2.

Exact reconstruction of the image from skeleton SK(X) can be achieved using the following steps;

step1: n = N, A = 0

step2: A = A $\bigcup$ Sn(X)

step3: if n = 0 STOP, otherwise A = A $\oplus$ B

step4: n = n - 1 and go to step2.

It may be possible to remove some points of the skeleton and still reconstruct the image exactly. It will be called a minimal skeleton, which is a subset of the original skeleton guaranteeing the exact reconstruction of the entire image.

Let X be the original image and let Sn(X), n = 0,1,..,N be its skeleton subsets with respect to a structuring element nB. For each skeleton subset index n, shift nB to all the points of Sn(X). This operation is equivalent to dilation of each points in Sn(X) by B 2n times. This is the characteristic function of the set nB. After that add algebraically all these contributions for all points of Sn(X) and for all n. This will make gray scale image, pgf(X), whose region is identical with the original image.

Now, in order to decide to remove a certain point which can be removed, check first whether the value of the pgf(X) at all the points of the region of support of the respective shifted characteristic function is >= 2. If so, this region of image is supported by skeleton point more than one time. Therefore, this skeleton point can be removed and subtracted algebraically from pgf(X). This operation is repeated until all the skeleton points have been searched.

The remaining skeleton points represent the minimal skeleton. Figure 4-10 shows the skeleton of a sample image and Figure 4-11 shows its minimal skeleton.

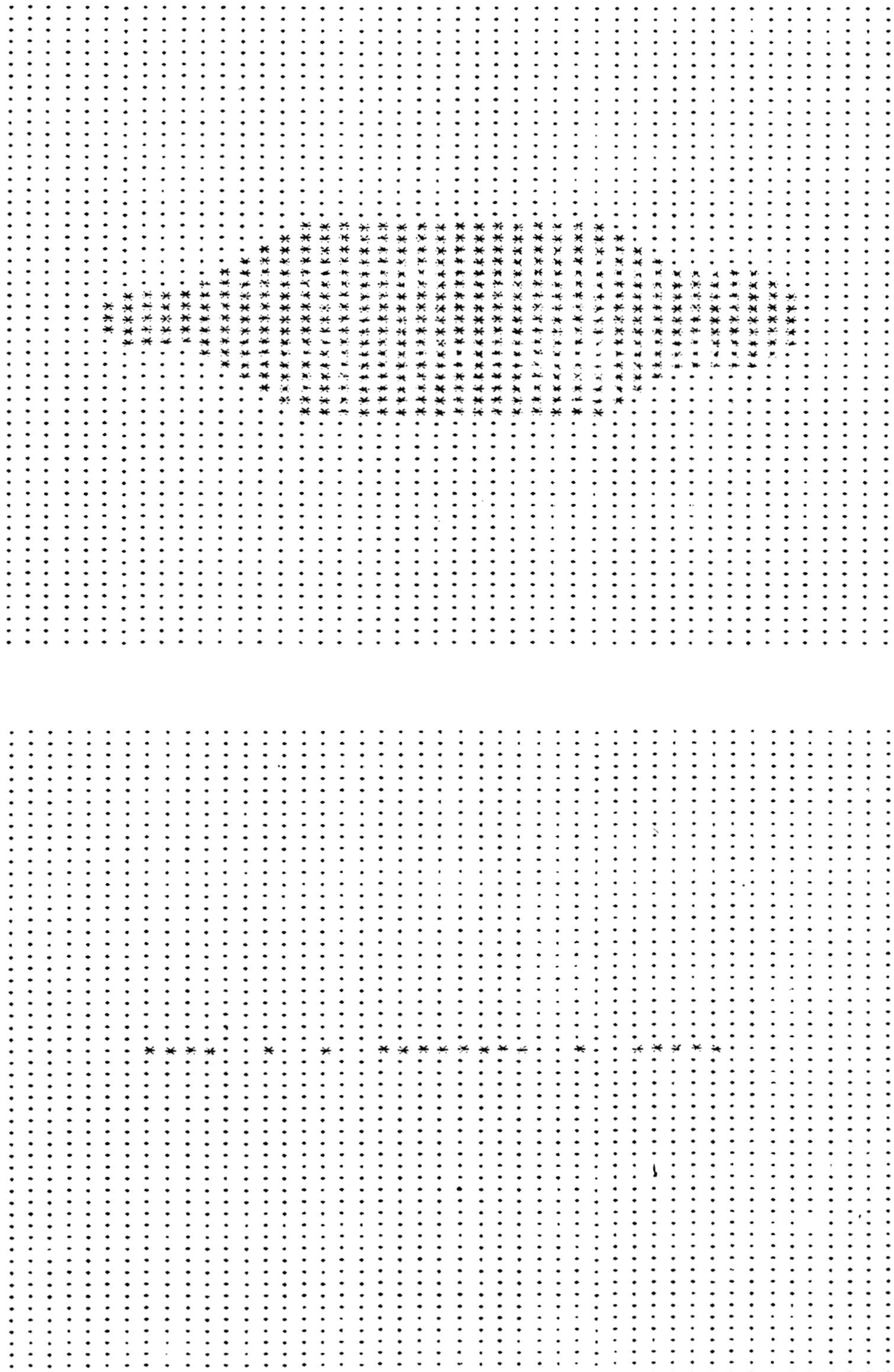


Figure 4-10 Sample image and its skeleton

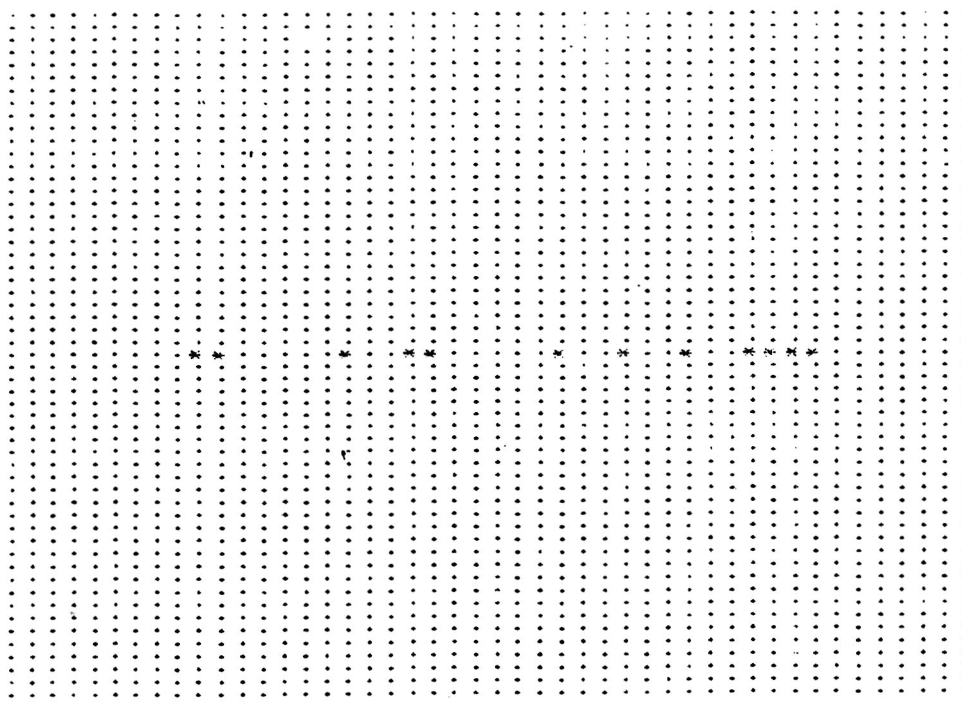


Figure 4-11 Minimal skeleton of Figure 4-10

CHAPTER 5: Summary and conclusions

The purpose of this report is to analyze images using mathematical morphology. The fundamental operations are implemented in computer program and some useful applications are tested by this program. As a structuring element, circle and square were used as these are convex sets and also symmetric.

For simplicity, binary images were tested. Morphlogical operations can easily be extended to gray scale images. The final results fully depends on the size and shape of the structuring element. Therefore, success of mathematical morphological methods depends on what kinds of structuring elements are used and how morphological operations are combined. Skeletonization of an image is more complex than other applications, and can be implemented by a combination of basic operations. Although minimal skeleton has less pixels, it can reconstruct the original image exactly. Using the algorithm which was used in this report, it is possible to reduce the number of pixels in the skeleton to obtain a minimal skeleton. With certain tolerance, the number of pixels in the mimimal skeleton can be reduced. However, unique way to do this is very important and not usually easy.

CHAPTER 6: References

[1] G. Matheron, Random Sets and Integrated Geometry. New York: Wiley, 1975.

[2] J. Serra, Image Analysis and Mathematical Morphology. New York: Academic Press, 1982.

[3] ________, "Introduction to Mathematical Morphology," Comput. Vision Graphics Image Prcess., vol. 35, pp. 283-305, Sept. 1986.

[4] C. R. Giardiana, and E. R. Dougherty, Morphological Methods in Image and Signal Processing. New Jersey: Prentice Hall, 1987.

[5] R. M. Halalick, S. R. Sternberg, and X. Zhuang, "Image Analysis Using Mathmatical Morphology," IEEE Trans. Pattern Anal. Machine Intell., vol. PAMI-9, pp. 532-550, July 1987.

[6] Z. Zhou, and A. N. Venetsanopoulos, "Morphological Skeleton Reprsentation and Shape Recognition," Proceeding : ICASSP-88, pp. 948-951, April 11-14, 1988.

[7] P. A. Maragos and R. W. Schafer, "Morphological Skeleton Representation and Coding of Binary Images," IEEE Trans. ASSP vol.34 pp. 1228-1244, Oct.1986.

[8] ________,________, "Applications of Morphological Filtering to Image Analysis and Processing," in Proc. IEEE ICASSP-86, Tokyo, pp.39.6.1-39.6.4.

[9] ________,________, "Morphological Skeleton Representation and Coding of Binary Images," in Proc. IEEE ICASSP, Sand Diego CA, Apr,1984, pp. 29.2.1-29.2.4

[10] ________,________, "A Unification of Linear, Median, Order-statistics and MorphologicalFilters under Mathematical Morphology," in Proc. IEEE ICASSP-85, Tampa Fla., March 1985, pp. 34.8.1-34.8.4.

[11] T. Pavlidis, "A Review of Algorithms for Shape Analysis", Comput. Graphics and Image Processing, vol. 7, no 2, April. 1978, pp. 243-258.

[12] I. Pitas, and A. N. Venetsanopoulos, "Shape Decomposition by Mathmatical Morphology," IEEE First Inter. Conf. Computer Vision, pp. 621-625, June 8-17, 1987

[13] T. R. Esselman, and J. G. Verly, "Some Applications of Mathematical Morphology to Range Imagery," Proceedings ICASSP-87, pp.245-248, April 6-9, 1987.

[14] P. K. Ghosh, "A Mathematical Model for Shape Description Using MinkowskiOperators," Comput. Vision, Graphics, Image Processing, vol. 44, pp.

239-269, Dec. 1988.

[15] S. R. Sternberg, "Grayscale Morphology," Comput. Vision Graphics Image Process., vol. 35, pp. 333-335, Sep. 1986.

[16] F. Meyer, "Automatic Screening of Cytological Specimens," Comput. Vision Graphics Image Process., vol. 35, pp. 356-369, Sept. 1986.

[17] X. Zhung, and R. M. Haralick, "Morphological Structuring Element Decomposition," Comput. Vision Graphics Image Process., vol. 35, pp. 370-382, Sep. 1986.

[18] F. Y. -C. Shih, and O. R. Michell, "Threshold Decomposition of Gray-scale Morphology into Binary Morphology," IEEE Trans. Pattern Anal. Machine Intell., vol. PAMI-11, pp. 31-42, Jan.1989.

[19] J. B. T. M. Roerdink, and H. J. A. M. Heijmans, "Mathematical Morphology for Structures without Translation Symmetry," Signal Processing, pp. 271-277, 1988.

[20] S. R. Sternberg, "Biomedical Image Processing," IEEE Computer, vol.16, pp. 22-34, Jan, 1983.

[21] J. G. Very, P. L. Van Hove, R. L. Walton, and D. E. Dudgeon, "Silhoutte Understanding System," in Proc. IEEE Conf. ASSP. ICASSP 86, Tokyo, pp. 1457-1460.

[22] R. S. Acharya, and R. Laurette, "Mathematical Morphology for 3-D Image Analysis," Proceedings: ICASSP-88, pp. 952 - 955, April 11-14, 1988

[23] F. Y. -C. Shih, and O. R. Michell, "Industrial parts recognition and inspection by image morphology," in Proc. IEEE Int. Conf. Robotics Automat., Philadelphia, PA, April 1988

[24] ______________, "Automated fast recognition and location of arbitrary shaped objects by image morphology," in Proc. Comp. Vision and Pattern Recognition Conf., Ann Arbor, MI, pp. 774-779, June 5-7, 1988.

APPENDIX : Computer Program Listings

This appendix includes the computer listings of some of the important routines written for this study. These routines are written to compute basic morphological operations and skeleton of image. These computer listings include only the routines which are relevent to the report.

```
/*****************************************************************
 *      Department of Electrical and Computer Engineering
 *                Kansas State University
 *           AT&T UNIX C Source file name : m_morp.h
 *****************************************************************
 *
 *
 *  DESCRIPTION:     This is a header file for all morphological
 *                   operation programs. This file has the
 *                   definition of IMAGE structure.
 *
*****************************************************************/

#define MAX 64                         /* Size of image (MAX X MAX)   */

typedef struct position                /* Type definiton of POSITION */
    {
    int x,
        y;
    } POSITION;

typedef struct image                   /* Type definition of IMAGE    */
    {
    int          size,
                 range[4000];
    POSITION     domain[4000];
    } IMAGE;

typedef struct str_elemt               /* Type definition of          */
    {                                  /* structuring element         */
    int          size,
                 range[21];
    POSITION     domain[21];
    } STR_ELEMT;

#define  NORMAL  0
#define  ERROR   1
```

```
/*****************************************************************
 *      Department of Electrical and Computer Engineering
 *                  Kansas State University
 *            AT&T UNIX C Source file name : morp.c
 *****************************************************************
 *
 *
 *   FUNCTION:          main()
 *
 *
 *   DESCRIPTION:       In this program morphological operations
 *                      will be performed, such as dilation, erosion,
 *                      opening and closing.
 *
 *
 *   DOCUMENTATION
 *   FILE:              None
 *
 *
 *   ARGUMENTS:         None
 *
 *
 *   RETURN:            None
 *
 *
 *   FUNCTION
 *   CALLED:            dilate()
 *                      erode()
 *                      make_im()
 *                      make_out()
 *                      sub()
 *                      comp()
 *                      ninety()
 *
 *
 *   AUTHOR:            Kyung Hyun Yoo
 *
 *
 *   DATE CREATED:      30 March 1989
 *
 *   REVISION:
 *
 ****************************************************************/
#include <stdio.h>
#include "m_morp.h"

main()
{
    extern     IMAGE *make_im(),
                     *dilate(),
                     *erode(),
                     *comp(),
                     *sub();

    extern     int   make_out(),
```

```
                    ninety();

    int         i,                      /* General purpose counter      */
                job,                    /* Selected job number          */
                sel;                    /* Selected structuring         */
                                        /* element                      */

    char        name[10];               /* Input file name              */

    IMAGE       buffer,
                *pt_test,               /* Pointer to input image file */
                *pt_sub,                /* Pointer to image in sub      */
                *pt_close,              /* Pointer to image in close    */
                *pt_open;               /* Pointer to image in open     */

    STR_ELEMT *pt_se;

    static      STR_ELEMT CIRCLE = {21, {1,1,1,1,1,1,1,1,1,1,1,
                                    1,1,1,1,1,1,1,1,1,1}, {{0,0},
                                    {0,-1},{0,-2},{0,1},{0,2},
                                    {1,0},{1,-1},{1,-2},{1,1},
                                    {1,2},{2,0},{2,-1},{2,1},
                                    {-1,0},{-1,-1},{-1,-2},{-1,1},
                                    {-1,2},{-2,0},{-2,-1},{-2,1}}},

                        SQUARE = {9, {1,1,1,1,1,1,1,1,1}, {{0,0},
                                    {-1,0},{-1,1},{0,-1},{-1,-1},
                                    {0,1},{1,-1},{1,0},{1,1}}},

                        RHOMBUS = {5, {1,1,1,1,1}, {{0,0},{0,-1},
                                    {-1,0},{1,0},{0,1}}},

                        BOXNE = {4, {1,1,1,1}, {{0,0},{0,1},
                                  {1,0},{1,1}}};

/*-------------------------------------------------------------------*/
/*  Make structure element                                           */
/*-------------------------------------------------------------------*/
    printf("    Select structuring element:\n");
    printf("    1. CIRCLE\n");
    printf("    2. SQUARE\n");
    printf("    3. RHOMBUS\n");
    printf("    4. BOXNE\n");

    scanf("%d", &sel);

    switch(sel)
        {
        case 1:
            pt_se = &CIRCLE;
            break;
        case 2:
            pt_se = &SQUARE;
            break;
        case 3:
            pt_se = &RHOMBUS;
```

```
            break;
        case 4:
            pt_se = &BOXNE;
            break;
        default:
            pt_se = &CIRCLE;
            break;
        }

/*-----------------------------------------------------------------*/
/*  Main body :select morphological operations and input files */
/*-----------------------------------------------------------------*/

    printf("     Enter the number you want to do:\n");
    printf("     1. Dilation\n     2. Erosion\n     3. Open \n");
    printf("     4. Close\n    5. Subtract\n     6. Comp\n");
    printf("     7. min\n     8. Exit\n");
    scanf("%d",&Job);

    while(Job != 8)
        {
        printf("     Enter the file name:");
        scanf("%s",name);

        switch(Job)
            {
            case 1:                             /* Dilation          */
                {
                pt_test = make_im(name);
                make_out(dilate(pt_test, pt_se));
                break;
                }

            case 2:                             /* Erosion           */
                {
                pt_test = make_im(name);
                make_out(erode(pt_test, pt_se));
                break;
                }

            case 3:                             /* Opening           */
                {
                pt_test = make_im(name);
                pt_open = erode(pt_test, pt_se);
                make_out(dilate(pt_open, pt_se));
                break;
                }

            case 4:                             /* Closing           */
                {
                pt_test = make_im(name);
                ninety(pt_se);
                ninety(pt_se);
                pt_close = dilate(pt_test, pt_se);
                make_out(erode(pt_close, pt_se));
                break;
```

```
            }

        case 5:                             /* Substraction      */
            {
            pt_sub = &buffer;
            pt_test = make_im(name);
            buffer = *pt_test;
            printf("        Enter the file name:");
            scanf("%s",name);
            pt_test = make_im(name);
            make_out(sub(pt_sub, pt_test));
            break;
            }

        case 6:                             /* Complement        */
            {
            pt_test = make_im(name);
            make_out(comp(pt_test));
            break;
            }

        case 7:                             /* Min operation     */
            {
            pt_sub = &buffer;
            pt_test = make_im(name);
            buffer = *pt_test;
            printf("Enter the file name:");
            scanf("%s",name);
            pt_test = make_im(name);
            make_out(min(pt_sub, pt_test));
            break;
            }
        default:
            break;
        }

    printf("     Enter the number you want to do:\n");
    printf("     1. Dilation\n     2. Erosion\n     3. Open \n");
    printf("     4. Close\n     5. Subtract\n     6. Comp\n");
    printf("     7. Min \n     8. Exit\n");
    scanf("%d",&Job);
    }
  return(0);
}
```

```
/*****************************************************************
 *     Department of Electrical and Computer Engineering
 *                Kansas State University
 *          AT&T UNIX C Source file name : skeleton.c
 *****************************************************************
 *
 *
 *  FUNCTION:          main()
 *
 *
 *  DESCRIPTION:       This program opens input file and find
 *                     skeleton and minimal skeleton image. User can
 *                     choose the structuring element.
 *
 *
 *  DOCUMENTATION
 *  FILE:              None
 *
 *
 *  ARGUMENTS:         None
 *
 *
 *  RETURN:            None
 *
 *
 *  FUNCTION
 *  CALLED:            make_im(),
 *                     make_ou(),
 *                     dilate(),
 *                     erode(),
 *                     recon(),
 *                     sub(),
 *                     min_sk_glob()
 *
 *
 *  AUTHOR:            Kyung Hyun Yoo
 *
 *
 *  DATE CREATED:      20 March 1989
 *
 *
 *  REVISION:
 *
*****************************************************************/
#include <stdio.h>
#include "m_morp.h"

main()
{
extern     IMAGE *dilate(),
                 *erode(),
                 *make_im(),
                 *min_sk_glob(),
                 *recon(),
                 *sub();
```

```
extern     int    make_out();

           int    i,
                  j,
                  k,
                  n,
                  sel;          /* For structure element selecting */

           char   name[10];     /* Input file name                 */

static     IMAGE  s[10];

           IMAGE  m_skel,
                  temp,
                  *eros1,
                  *eros2,
                  *open,
                  *pt_buf;

    STR_ELEMT *pt_se;

    static STR_ELEMT CIRCLE = {21, {1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                               1,1,1,1,1,1,1},{{0,0},{0,-1},
                               {0,-2},{0,1},{0,2},{1,0},{1,-1},
                               {1,-2},{1,1},{1,2},{2,0},{2,-1},
                               {2,1},{-1,0},{-1,-1},{-1,-2},
                               {-1,1},{-1,2},{-2,0},{-2,-1},
                               {-2,1}}},

                     SQUARE = {9, {1,1,1,1,1,1,1,1,1}, {{-1,-1},
                               {-1,0},{-1,1},{0,-1},{0,0},{0,1},
                               {1,-1},{1,0},{1,1}}},

                     RHOMBUS = {5, {1,1,1,1,1}, {{0,-1},{-1,0},
                                {0,0},{1,0},{0,1}}};

/*-------------------------------------------------------------*/
/*  Select structure element                                    */
/*-------------------------------------------------------------*/
    printf("    Select structuring element:\n");
    printf("    1. CIRCLE\n");
    printf("    2. SQUARE\n");
    printf("    3. RHOMBUS\n");

    scanf("%d", &sel);

    switch(sel)
        {
        case 1:
            pt_se = &CIRCLE;
            break;
        case 2:
            pt_se = &SQUARE;
            break;
        case 3:
            pt_se = &RHOMBUS;
```

```
            break;
        default:
            pt_se = &CIRCLE;
            break;
        }

/*---------------------------------------------------------------*/
/*   Make IMAGE from input image file                            */
/*---------------------------------------------------------------*/
    printf(" Enter the file name:");
    scanf("%s",name);

    n = 0;                                  /* Maximum size of SE */

    eros1 = &temp;
    *eros1 = *make_im(name);
    eros2 = erode(eros1, pt_se);

/*---------------------------------------------------------------*/
/*  Main boby of skeleton algorithms                             */
/*---------------------------------------------------------------*/
    while(eros2->size != 0)
        {
        open = dilate(eros2, pt_se);

        pt_buf = sub(eros1, open);

/*---------------------------------------------------------------*/
/*  Make skeleton subsets                                        */
/*---------------------------------------------------------------*/
        *(s + n) = *pt_buf;

        *eros1 = *eros2;

        eros2 = erode(eros1, pt_se);

        n++;
        }

    *(s + n) = *eros1;

/*---------------------------------------------------------------*/
/*  Make skeleton                                                */
/*---------------------------------------------------------------*/
    i = 0;
    m_skel.size = 0;
    for(j = 0; j <= n; j++)
        {
        m_skel.size += s[j].size;
        for( k = 0; k < s[j].size; k++)
            {
            m_skel.domain[i].x = s[j].domain[k].x;
            m_skel.domain[i].y = s[j].domain[k].y;
            m_skel.range[i] = s[j].range[k];
            i++;
            }
```

```
        }

    printf("Now skeleton image was made.\n");
    pt_buf = &m_skel;

    make_out(pt_buf);

/*    make_out(recon(s,pt_se,n));
    scanf("%d",&dumm);
*/
/*-----------------------------------------------------------------*/
/*  Make minimum skeleton                                          */
/*-----------------------------------------------------------------*/
    make_out(min_sk_glob(s, pt_se, n));

/*    make_out(recon(s,pt_se,n));*/

    return(0);
}
```

```
/*****************************************************************
 *      Department of Electrical and Computer Engineering
 *                Kansas State University
 *           AT&T UNIX C Source file name : dilate.c
 *****************************************************************
 *
 *
 *   FUNCTION:         dilate(f, s)
 *
 *
 *   DESCRIPTION:      This function take an image and a structure
 *                     element as inputs and output a dilated image
 *
 *
 *   DOCUMENTATION
 *   FILE:             None
 *
 *
 *   ARGUMENTS:
 *
 *       f             (input) IMAGE *
 *                     Pointer to input IMAGE which will be dilated.
 *
 *       s             (input) STR_ELEMT *
 *                     Pointer to the structre element which will be
 *                     used for dilaion.
 *
 *
 *   RETURN:           IMAGE *
 *                     This function will return the pointer to
 *                     IMAGE which will represent the dilated image.
 *
 *
 *   FUNCTION
 *   CALLED:           trans(),
 *                     extmax()
 *
 *
 *   AUTHOR:           Kyung Hyun Yoo
 *
 *
 *   DATE CREATED:     20 March 1989
 *
 *
 *   REVISION:
 *
 *****************************************************************/
#include "m_morp.h"

IMAGE dilated;                  /* Dilated image                  */
IMAGE *dilate(f, s)

IMAGE     *f;                   /* Input image                    */
STR_ELEMT *s;                   /* Structure element which will   */
                                /* be used                        */
```

```
{
    extern IMAGE *trans(),
                 *extmax();

    IMAGE  *pt_dil;

    int    i,
           x,                   /* Position of pixel in structure */
           y;                   /* element                        */

    pt_dil = &dilated;

/*--------------------------------------------------------------*/
/*  Make dilated image by taking union of transed image         */
/*--------------------------------------------------------------*/
    for(i = 0; i < s->size; i++)
        {
        x = s->domain[i].x;
        y = s->domain[i].y;

        pt_dil = extmax(trans(f, x, y), pt_dil);
        }

    return(pt_dil);
}
```

```
/*****************************************************************
 *     Department of Electrical and Computer Engineering
 *              Kansas State University
 *          AT&T UNIX C Source file name : erode.c
 *****************************************************************
 *
 *
 *  FUNCTION:        erode(f, s)
 *
 *
 *  DESCRIPTION:     This function take an image and a structure
 *                   element as inputs and output an eroded image
 *
 *
 *  DOCUMENTATION
 *  FILE:            None.
 *
 *
 *  ARGUMENTS:
 *
 *      f            (input) IMAGE *
 *                   Ponter to input structure IMAGE which will
 *                   be eroded.
 *
 *      s            (input) IMAGE *
 *                   Pointer to structure element which will be
 *                   used for erosion.
 *
 *
 *  RETURN:          IMAGE *
 *                   This function will return the pointer to
 *                   structure IMAGE which will represent the
 *                   eroded image.
 *
 *
 *  FUNCTION
 *  CALLED:          min()
 *                   ninety()
 *                   trans()
 *
 *
 *  AUTHOR:          Kyung Hyun Yoo
 *
 *
 *  DATE CREATED:
 *
 *
 *  REVISION:
 *
 *****************************************************************/

#include "m_morp.h"

IMAGE eroded;
IMAGE *erode(f, s)
```

```
IMAGE       *f;
STR_ELEMT   *s;
{
    extern IMAGE *min(),
                 *trans();

    extern int   ninety();

    int     i,
            j,
            k,
            x,
            y;

    IMAGE   *pt_buffer,
            *pt_erod;

    pt_erod = &eroded;

    ninety(s);
    ninety(s);

/*---------------------------------------------------------------*/
/*  Initialize all pixels in IMAGE mined to 1                    */
/*---------------------------------------------------------------*/
    k = 0;

    for(i = 0; i < MAX; i++)
        for(j = 0; j < MAX; j++)
            {
            pt_erod->domain[k].x = i;
            pt_erod->domain[k].y = j;
            pt_erod->range[k++] = 1;
            }

    pt_erod->size = MAX * MAX;

/*---------------------------------------------------------------*/
/*  Erode operation ( IMAGE AND opertion )                       */
/*---------------------------------------------------------------*/
    for(i = 0; i < s->size; i++)
        {
        x = s->domain[i].x;
        y = s->domain[i].y;

        pt_buffer = min(trans(f, x, y), pt_erod);

        *pt_erod = *pt_buffer;
        }

    return(pt_erod);
}
```

```
/****************************************************************
 *      Department of Electrical and Computer Engineering
 *                  Kansas State University
 *             AT&T UNIX C Source file name : min_sk_glob.c
 ****************************************************************
 *
 *
 *  FUNCTION:          min_sk_glob(sk_sub, pt_se, n)
 *
 *
 *  DESCRIPTION:       This function make minimal skeleton image
 *                     from skeleton subsets array.
 *
 *
 *  DOCUMENTATION
 *  FILE:              None
 *
 *
 *  ARGUMENTS:
 *
 *      sk_sub         (input) IMAGE[]
 *                     Skeleton subset array of input image
 *
 *      pt_se          (input) SE_ELEMT
 *                     Structuring element corresponding to
 *                     skeleton subsets
 *
 *      n              (input)
 *                     Size of skeleton subsets array
 *
 *
 *  RETURN:            IMAGE *
 *                     Pointer to minimal skeleton image
 *
 *
 *  FUNCTION
 *  CALLED:            dilate()
 *                     make_out()
 *
 *
 *  AUTHOR:            Kyung Hyun Yoo
 *
 *
 *  DATE CREATED:      30 March 1989
 *
 *
 *  REVISION:
 *
 ****************************************************************/

#include "m_morp.h"

IMAGE min_sk;
IMAGE *min_sk_glob(sk_sub, pt_se, n)

IMAGE       sk_sub[];            /* Array of skeleton subsets     */
```

```
STR_ELEMT   *pt_se;                /* Structure element              */
int         n;                     /* Numbers of skeleton subsets    */
{
    extern IMAGE  *dilate();
    extern int    make_out();

    IMAGE   min_sk_sub[10],        /* Array of minimal skeleton      */
                                   /* subsets                        */
            *pt_buffer,            /* For temporary storage          */
            *pt_dummy,             /* For temporary storage          */
            *pt_temp,              /* For temporary storage          */
            psf,                   /* pseudo gray level function     */
            temp;

    static int pixel[MAX][MAX];

    int     i,j,k,l,               /* Counter                        */
            x,y,                   /* Location of pixels             */
            flag,
            size;                  /* Size of IMAGE                  */

/*---------------------------------------------------------------*/
/*  For 0th skeleton                                             */
/*---------------------------------------------------------------*/
    min_sk_sub[0] = sk_sub[0];

/*---------------------------------------------------------------*/
/*  Make modified subsets (dialte(dilate(sk_sub)))               */
/*---------------------------------------------------------------*/
printf("In modify section\n");
    for(i = 1; i <= n; i++)
        {
        pt_dummy = sk_sub + i;
        j = pt_dummy->size;
        pt_buffer = min_sk_sub + i;
        pt_buffer->size = 0;

        /*-------------------------------------------------------*/
        /*  Make minimal skeleton subsets which have the gray    */
        /*  level. Shifted version of each pixels by structure   */
        /*  element makes this gray level.                       */
        /*-------------------------------------------------------*/

        while(j > 0)
            {   printf("make minimal skeleton\n");
            pt_temp = &temp;
            temp.size = 1;
            temp.domain[0].x = pt_dummy->domain[j-1].x;
            temp.domain[0].y = pt_dummy->domain[j-1].y;
            temp.range[0] = pt_dummy->range[j-1];

            pt_temp = dilate(pt_temp, pt_se);

            k = i - 1;

            while(k > 0)    /* Shifting structure element is     */
```

```
                         /* equal to taking dilation 2 times*/
            {
            pt_temp = dilate(pt_temp, pt_se);
            k--;
            }

        size = pt_buffer->size;

        flag = 0;
        for(k = 0; k < pt_temp->size; k++)
            {
            for(l = 0; l < size; l++)
                {
                if((pt_buffer->domain[l].x ==
                                        pt_temp->domain[k].x)&&
                    (pt_buffer->domain[l].y ==
                                        pt_temp->domain[k].y))
                    {
                    flag = 1;
                    pt_buffer->range[l] += 1;
                    break;
                    }
              }

            if(flag == 0)
               {
               pt_buffer->domain[pt_buffer->size].x =
                                        pt_temp->domain[k].x;
               pt_buffer->domain[pt_buffer->size].y =
                                        pt_temp->domain[k].y;
               pt_buffer->range[pt_buffer->size] =
                                        pt_temp->range[k];
               pt_buffer->size++;
               }
            else
               flag = 0;

            }
        j--;
        }
    }
/*-------------------------------------------------------------*/
/*  Make pgf  (pseudo graytone function)                        */
/*-------------------------------------------------------------*/
printf("In pgf section\n");
    pt_dummy = &pgf;
    pt_dummy->size = 0;
    flag = 0;

    pt_buffer = min_sk_sub;
    *pt_dummy = *pt_buffer;          /* For 0th skeleton            */

    for(i = 1; i <= n; i++)
        {
        pt_buffer = min_sk_sub + i;
        size = pt_dummy->size;
```

```
        for(j = 0; j < pt_buffer->size; j++)
                                  /* Add all minimal skeleton */
                                  /* subsets                  */
            {
            for(k = 0; k < size; k++)
                {
                if((pt_buffer->domain[j].x ==
                                  pt_dummy->domain[k].x)&&
                   (pt_buffer->domain[j].y ==
                                  pt_dummy->domain[k].y))
                    {
                    flag = 1;
                    pt_dummy->range[k] += pt_buffer->range[j];
                    break;
                    }
              }
            if(flag == 0)
                {
                pt_dummy->domain[pt_dummy->size].x =
                                  pt_buffer->domain[j].x;
                pt_dummy->domain[pt_dummy->size].y =
                                  pt_buffer->domain[j].y;
                pt_dummy->range[pt_dummy->size] =
                                  pt_buffer->range[j];
                pt_dummy->size++;
                }
            else
                flag = 0;

            }
        }

/*----------------------------------------------------------*/
/*  Make pixel form                                         */
/*----------------------------------------------------------*/
printf("In pixel section\n");
    for(i = 0; i < pt_dummy->size; i++)
        {
        x = pt_dummy->domain[i].x;
        y = pt_dummy->domain[i].y;
        pixel[x][y] = pt_dummy->range[i];
        }

/*----------------------------------------------------------*/
/*  Check the contributionsof pixels in skeleton substs to  */
/*  psf function. If this contribution is greater than 2,   */
/*  that pixel can be removed.                              */
/*----------------------------------------------------------*/
    for(i = 1; i <= n; i++)
        {
        pt_dummy = sk_sub + i;
        j = pt_dummy->size;
        while(j > 0)
            {               printf("In checking section\n");
            pt_temp = &temp;
```

```
                temp.size = 1;
                temp.domain[0].x = pt_dummy->domain[j-1].x;
                temp.domain[0].y = pt_dummy->domain[j-1].y;
                temp.range[0] = pt_dummy->range[j-1];

                pt_temp = dilate(pt_temp,pt_se);

                k = i - 1;

                while(k > 0)
                    {
                    pt_temp = dilate(pt_temp, pt_se);
                    k--;
                    }

                for(k = 0; k < pt_temp->size; k++)
                    {
                    x = pt_temp->domain[k].x;
                    y = pt_temp->domain[k].y;

                    if (pixel[x][y] < 2)
                        flag = 1;
                    }

                if(flag == 0)
                    {
                    for(k = 0; k < pt_temp->size; k++)
                        {
                        x = pt_temp->domain[k].x;
                        y = pt_temp->domain[k].y;
                        pixel[x][y] -= 1;
                        }
                    pt_dummy->domain[j-1].x = -1;
                    pt_dummy->domain[j-1].y = -1;
                    pt_dummy->range[j-1] = 0;
                    }
                else
                    flag = 0;

                j--;
                }
            }

/*-----------------------------------------------------------------*/
/*  Make minimal skeleton image from subsets                        */
/*-----------------------------------------------------------------*/
    i = 0;
    temp.size = 0;
    for(j = 0; j <= n; j++)
        {
        for(k = 0; k < sk_sub[j].size; k++)
            {
            if ((sk_sub[j].domain[k].x >= 0) &&
                (sk_sub[j].domain[k].y >= 0))
              {
              temp.domain[i].x = sk_sub[j].domain[k].x;
```

```
                    temp.domain[i].y = sk_sub[j].domain[k].y;
                    temp.range[i] = sk_sub[j].range[k];
                    i++;
                    }
               }
            }
      temp.size = i;
      pt_dummy = &temp;

      return(pt_dummy);
}
```

```
/*****************************************************************
 *      Department of Electrical and Computer Engineering
 *                  Kansas State University
 *            AT&T UNIX C Source file name : trans.c
 *****************************************************************
 *
 *
 *   FUNCTION:          trans(f, x_val, y_val)
 *
 *
 *   DESCRIPTION:       This function move the input image f over
 *                      i pixels to the right and j pixels down.
 *
 *
 *   DOCUMENTATION
 *   FILE:              None.
 *
 *
 *   ARGUMENTS:
 *
 *       f              (input) IMAGE *
 *                      Pointer to input IMAGE which will be moved.
 *
 *       x_val          (input) int
 *                      Image will be moved along the x_axis by this
 *                      value.
 *
 *       y_val          (input) int
 *                      Image will be moved along the y_axis by this
 *                      value.
 *
 *
 *   RETURN:            IMAGE *
 *                      This function will return the pointer to
 *                      IMAGE which was moved by x_val and y_val.
 *
 *
 *   FUNCTION
 *   CALLED:            None.
 *
 *
 *   AUTHOR:            Kyung Hyun Yoo
 *
 *
 *   DATE CREATED:      20 March 1989
 *
 *
 *   REVISION:          None
 *
 *****************************************************************/
#include "m_morp.h"

IMAGE    transed;                   /* IMAGE which was moved       */
IMAGE    *trans(f, x_val, y_val)

IMAGE    *f;                        /* Image which will be moved   */
```

```
int      x_val,                    /* Trans value x-axis          */
         y_val;                    /* Trans value y-axis          */
{
    int       i;                   /* General purpose counter     */

    IMAGE    *pt_trans;

    pt_trans = &transed;

    *pt_trans = *f;

/*-----------------------------------------------------------------*/
/*  Check input image                                              */
/*-----------------------------------------------------------------*/
    if(f->size <= 0)
        {
        printf("  Error: Size of input IMAGE is less than 0\n");
        exit(1);
        }

/*-----------------------------------------------------------------*/
/*  Make transition image of input                                 */
/*-----------------------------------------------------------------*/
    i = 0;
    while(i < f->size)
        {
        if((pt_trans->domain[i].x = f->domain[i].x + x_val) >= 64)
            pt_trans->domain[i].x -= 64;

        if((pt_trans->domain[i].y = f->domain[i].y + y_val) >= 64)
            pt_trans->domain[i].y -= 64;

        i++;
        }

    return(pt_trans);
}
```

```
/*************************************************************
 *      Department of Electrical and Computer Engineering
 *                 Kansas State University
 *           AT&T UNIX C Source file name : extmax.c
 *************************************************************
 *
 *
 *   FUNCTION:         extmax(f, g)
 *
 *
 *   DESCRIPTION:      This function compares two input images in a
 *                     pixelwise manner and output the maximum gray
 *                     value at each pixel.
 *
 *
 *   DOCUMENTATION
 *   FILE:             None
 *
 *
 *   ARGUMENTS:
 *
 *       f             (input) IMAGE *
 *                     Input IMAGE which will be compared
 *
 *       g             (input) IMAGE *
 *                     Input IMAGE which will be compared
 *
 *
 *   RETURN:           IMAGE *
 *                     This function returns a IMAGE which has the
 *                     maximum gray value of input images.
 *
 *
 *   FUNCTION
 *   CALLED:           None
 *
 *
 *   AUTHOR:           Kyung Hyun Yoo
 *
 *
 *   DATE CREATED:     20 March 1989
 *
 *
 *   REVISION:
 *
 *************************************************************/

#include "m_morp.h"

IMAGE   extmaxed;
IMAGE   *extmax(f, g)

IMAGE   *f,                     /* Pointer to input IMAGE     */
        *g;                     /* Pointer to input IMAGE     */

{
```

```
    int     i,                          /* General purpose counter     */
            j,                          /* General purpose counter     */
            flag;

    IMAGE *pt_extmax;

    pt_extmax = &extmaxed;

    flag = 0;

/*-----------------------------------------------------------------------*/
/*  Check IMAGE g. If IMAGE g has no pixel, output IMAGE will            */
/*  be the same IMAGE as input IMAGE f                                   */
/*-----------------------------------------------------------------------*/
    if(g->size == 0)
        *pt_extmax = *f;

/*-----------------------------------------------------------------------*/
/*  Check the gray value of each pixels and output the highest */
/*  gray value                                                           */
/*-----------------------------------------------------------------------*/
    else
        {
        *pt_extmax = *g;

        for(i = 0; i <f->size; i++)
            {
            for(j = 0; j < g->size; j++)
                {
                if((f->domain[i].x == pt_extmax->domain[j].x) &&
                   (f->domain[i].y == pt_extmax->domain[j].y))
                    {
                    flag = 1;
                    if(f->range[i] > extmaxed.range[j])
                        extmaxed.range[j] = f->range[i];
                    break;
                    }
                }
            if(flag == 0)
                {
                extmaxed.domain[extmaxed.size] = f->domain[i];
                extmaxed.range[extmaxed.size] = f->range[i];
                extmaxed.size++;
                }
            else
                flag = 0;
            }
        }
   return(pt_extmax);
}
```

```
/****************************************************************
 *      Department of Electrical and Computer Engineering
 *                 Kansas State University
 *            AT&T UNIX C Source file name : min.c
 ****************************************************************
 *
 *
 *   FUNCTION:          min(f, g)
 *
 *
 *   DESCRIPTION:       This function compares two images in a
 *                      pixelwise manner and outputs the maximum gray
 *                      value at each pixels.
 *
 *
 *   DOCUMENTATION
 *   FILE:              None.
 *
 *
 *   ARGUMENTS:
 *
 *       f              (input) IMAGE *
 *                      Input IMAGE which will be compared.
 *
 *       g              (input) IMAGE *
 *                      Input IMAGE which will be compared.
 *
 *
 *   RETURN:            IMAGE *
 *                      This function returns IMAGE which has the
 *                      lowest gray level of input IMAGEs.
 *
 *
 *   FUNCTION
 *   CALLED:            None.
 *
 *
 *   AUTHOR:            Kyung Hyun Yoo
 *
 *
 *   DATE CREATED:      20 March 1989
 *
 *
 *   REVISION:
 *
 ****************************************************************/
#include "m_morp.h"

IMAGE minimum;
IMAGE *min(f, g)

IMAGE *f,                       /* Pointer to input image       */
      *g;                       /* Pointer to input image       */
{
    int   i,                    /* General purpose counter      */
          j,                    /* General purpose counter      */
```

```
        flag;

    IMAGE *pt_min;

    flag = 0;
    pt_min = &minimum;

/*-------------------------------------------------------------*/
/*  Take an intersection of input images and lowest gray level */
/*  for each pixels                                            */
/*-------------------------------------------------------------*/
    minimum.size = 0;

    for(i = 0; i < f->size; i++)
        {
        for(j = 0; j < g->size; j++)
            {
            if((f->domain[i].x == g->domain[j].x) &&
               (f->domain[i].y == g->domain[j].y))
                {
                minimum.domain[minimum.size].x = f->domain[i].x;
                minimum.domain[minimum.size].y = f->domain[i].y;
                minimum.range[minimum.size] = f->range[i];
                minimum.size++;
                }
            }
        }
    return(pt_min);
}
```

```
/*****************************************************************
 *     Department of Electrical and Computer Engineering
 *                Kansas State University
 *          AT&T UNIX C Source file name : ninety.c
 *****************************************************************
 *
 *
 *  FUNCTION:         ninety(s)
 *
 *
 *  DESCRIPTION:      This function rotate structure element
 *                    90 degrees.
 *
 *
 *  DOCUMENTATION
 *  FILE:             None.
 *
 *
 *  ARGUMENTS:
 *
 *      s             (input/output) STR_ELEMT *
 *                    Structure element which will be rotate.
 *
 *  RETURN:           int
 *                    NORMAL
 *                    ERROR
 *
 *
 *  FUNCTION
 *  CALLED:           None.
 *
 *
 *  AUTHOR:           Kyung Hyun Yoo
 *
 *
 *  DATE CREATED:     10 March 1989
 *
 *
 *  REVISION:         None
 *
 *****************************************************************/
#include "m_morp.h"

int ninety(s)

STR_ELEMT *s;                  /* Pointer to structure element */
{
    int i,                     /* General purpose counter       */
        temp;                  /* Temporary buffer              */

    if(s->size <= 0)
        {
        printf("  Error: Size of input IMAGE is less than 0\n");
        exit(1);
        }
```

```
/*****************************************************************/
/*   Rotate 90 degrees                                           */
/*****************************************************************/
     for(i = 0; i < s->size; i++)
          {
          temp = s->domain[i].y;
          s->domain[i].y = s->domain[i].x;
          s->domain[i].x = -1 * temp;
          }
     return(NORMAL);
}
```

```
/***************************************************************
 *      Department of Electrical and Computer Engineering
 *                 Kansas State University
 *           AT&T UNIX C Source file name : make_im.c
 ***************************************************************
 *
 *
 *   FUNCTION:        make_im(name)
 *
 *
 *   DESCRIPTION:     This function makes structure IMAGE from
 *                    input image file. Input image file is a
 *                    binary image which was expressed by "1" and
 *                    "0". Size of image is 64 * 64 pixels.
 *
 *
 *   DOCUMENTATION
 *   FILE:            None.
 *
 *
 *   ARGUMENTS:
 *
 *       name         (input) char[]
 *                    Name of input image file.
 *
 *
 *   RETURN:          *IMAGE
 *                    This function will return the pointer to
 *                    structure IMAGE.
 *
 *
 *   FUNCTION
 *   CALLED:          None.
 *
 *
 *   AUTHOR:          Kyung Hyun Yoo
 *
 *
 *   DATE CREATED:    20 March 1989
 *
 *
 *   REVISION:        None.
 *
 ***************************************************************/
#include <stdio.h>
#include "m_morp.h"

IMAGE original;              /* IMAGE which will represent input */
                             /* image file.                      */

IMAGE *make_im(name)

char  name[];                /* Name of input image file         */
{
    int    c,                /* Integer value of each pixel      */
           i,                /* General purpose counter          */
```

```
        j,                  /* General purpose counter              */
        k,                  /* General purpose counter              */
        pixel[MAX][MAX];    /* Array of pixel value                 */

    IMAGE *pt_im;

    FILE  *infile;

    pt_im = &original;

/*---------------------------------------------------------------*/
/*  Open input file                                              */
/*---------------------------------------------------------------*/
    if((infile = fopen(name, "r")) == (FILE *) NULL)
        {
        printf(stderr," Error: cannot open file (make_im())\n");
        exit(1);
        }

/*---------------------------------------------------------------*/
/*  Make pixel array from input file ( MAX = 64 )                */
/*---------------------------------------------------------------*/
    for(j = 0; j < MAX; j++)
        for(i = 0; i < MAX; i++)
            pixel[j][i] = 0;

    for(j = 0; j < MAX; j++)
        for(i = 0; i < MAX; i++)
            {
            if((c = getc(infile)) == EOF)
                pixel[j][i] = EOF;

            else if(c != '\n')
                pixel[j][i] = c - 48;

            else
                i--;
            }

/*---------------------------------------------------------------*/
/*  Make structure IMAGE from pixel array                        */
/*---------------------------------------------------------------*/
    k = 0;
    pt_im->size = 0;

    for(j = 0; j < MAX; j++)
        for(i = 0; i < MAX; i++)
            {
            if(pixel[j][i] >= 1)
                {
                pt_im->domain[k].x = i;
                pt_im->domain[k].y = j;
                pt_im->range[k++] = pixel[j][i];
                pt_im->size++;
                }
            }
```

```
/*****************************************************************
 *      Department of Electrical and Computer Engineering
 *                 Kansas State University
 *           AT&T UNIX C Source file name : make_out.c
 *****************************************************************
 *
 *
 *   FUNCTION:          make_out(pt_im)
 *
 *
 *   DESCRIPTION:       This function converts structure IMAGE to
 *                      image file which has 64 * 64 pixels. Each
 *                      pixel will be represented as a number, which
 *                      is the gray level.
 *
 *
 *   DOCUMENTATION
 *   FILE:              None.
 *
 *
 *   ARGUMENTS:
 *
 *       pt_im          (input) IMAGE *
 *                      Pointer to input structure IMAGE.
 *
 *
 *   RETURN:            int
 *                      NORMAL
 *                      ERROR
 *
 *
 *   FUNCTION
 *   CALLED:            None.
 *
 *
 *   AUTHOR:            Kyung Hyun Yoo
 *
 *
 *   DATE CREATED:      20 March 1989
 *
 *
 *   REVISION:          None.
 *
 ****************************************************************/
#include <stdio.h>
#include "m_morp.h"

int make_out(pt_im)

IMAGE *pt_im;
{
    char  name[10];             /* Name of output image file    */

    int   i,                    /* General purpose counter      */
          j,                    /* General purpose counter      */
          x,                    /* Location of pixel            */
```

```
        y,                        /* Location of pixel              */
        pixel[MAX][MAX];          /* Array of pixel value           */

    FILE  *outfile;

/*----------------------------------------------------------------*/
/*  Open output file                                              */
/*----------------------------------------------------------------*/
    printf("You are going to make image file\n\n");
    printf("Enter the file name within 10 characters: ");
    scanf("%s",name);

    if((outfile = fopen(name,"w")) == (FILE *) NULL)
        {
        printf("  Error: cannot open file (make_out())\n");
        exit(1);
        }

/*----------------------------------------------------------------*/
/*  Make pixel array from structure IMAGE                         */
/*----------------------------------------------------------------*/
    for(j = 0; j < MAX; j++)
        for(i = 0; i < MAX; i++)
            pixel[j][i] = 0;

    for(i = 0; i < pt_im->size; i++)
        {
        x = pt_im->domain[i].x;
        y = pt_im->domain[i].y;
        if((x >= 0) && (y >= 0))
            pixel[y][x] = pt_im->range[i];
        else
            return(ERROR);
        }

/*----------------------------------------------------------------*/
/*  Make output image file from pixel array                       */
/*----------------------------------------------------------------*/
    for(j = 0; j < MAX; j++)
        {
        for(i = 0; i < MAX; i++)
            {
            if(pixel[j][i] == 0)
                fprintf(outfile,"0");
            else if(pixel[j][i] >= 1)
                fprintf(outfile,"%d",pixel[j][i]);
            else
                fprintf(outfile,"*");
            }
        fprintf(outfile,"\n");
        }

    fclose(outfile);

    return(NORMAL);
}
```

```
/*****************************************************************
 *     Department of Electrical and Computer Engineering
 *                Kansas State University
 *          AT&T UNIX C Source file name : sub.c
 *****************************************************************
 *
 *
 *  FUNCTION:         sub(pict1, pict2)
 *
 *
 *  DESCRIPTION:      This function subtract a image from the other
 *                    image.
 *
 *
 *  DOCUMENTATION
 *  FILE:             None.
 *
 *
 *  ARGUMENTS:
 *
 *      pict1         (input) IMAGE*
 *                    Minuend IMAGE
 *
 *      pict2         (input) IMAGE*
 *                    Subtrahend IMAGE
 *
 *
 *  RETURN:           This function returns the pointer to
 *                    subtracted image.
 *
 *
 *  FUNCTION
 *  CALLED:           None.
 *
 *
 *  AUTHOR:           Kyung Hyun Yoo
 *
 *
 *  DATE CREATED:     20 March 1989
 *
 *
 *  REVISION:
 *
 ****************************************************************/
#include <stdio.h>
#include "m_morp.h"

IMAGE subed;

IMAGE *sub(pict1, pict2)

IMAGE *pict1,                       /* Input minuend image    */
      *pict2;                       /* Input subtrhend image  */

{
    static int pixel1[MAX][MAX],
```

```
                pixel2[MAX][MAX];

    int     x,                              /* Location of pixel        */
            y,                              /* points                   */
            i,                              /* Counter                  */
            j,                              /* Counter                  */
            k,                              /* Counter                  */
            point;                          /* Gray value difference    */

    IMAGE *out;

    out = &subed;

/*----------------------------------------------------------------*/
/*  Initialize the pixel arrays                                   */
/*----------------------------------------------------------------*/
    for(i = 0; i < MAX; i++)
        for(j = 0; j < MAX; j++)
            {
            pixel1[i][j] = 0;
            pixel2[i][j] = 0;
            }

/*----------------------------------------------------------------*/
/*  Make pixel arrays for input images                            */
/*----------------------------------------------------------------*/
    for(i = 0; i < pict1->size; i++)
        {
        x = pict1->domain[i].x;
        y = pict1->domain[i].y;
        pixel1[x][y] = 1;
        }

    for(i = 0; i < pict2->size; i++)
        {
        x = pict2->domain[i].x;
        y = pict2->domain[i].y;
        pixel2[x][y] = 1;
        }

    k = 0;

/*----------------------------------------------------------------*/
/*  Subtract gray level                                           */
/*----------------------------------------------------------------*/
    for(i = 0; i < MAX; i++)
        for(j = 0; j < MAX; j++)
            {
            point = pixel1[i][j] - pixel2[i][j];
            if (point == 1)
                {
                out->domain[k].x = i;
                out->domain[k].y = j;
                out->range[k] = 1;
                k++;
                }
```

```
                }
        out->size = k;

        return(out);
}
```

IMAGE ANALYSIS USING MATHEMATICAL MORPHOLOGY

by

KYUNG HYUN YOO

B.S., Hanyang University, KOREA, 1980

AN ABSTRACT OF A REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

ELECTRICAL ENGINEERING

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1989

ABSTRACT

This report presents the application of morphological techniques to binary image analysis. Mathematical morphology provides an approach to the processing of digital image in terms of some predetermined geometric shape known as a structuring element. A brief discussion of mathematical morphology is included as a background along with some definitions of basic morphological terms. The programs for basic morphological oprations are developed using C language on AT&T 3B2 computer. The results of the application of morphological techniques to applications such as noise cleaning, edge detection, region filling and image representation are also presented.