

To the Graduate Council:

I am submitting herewith a thesis written by Chien-Chyun Lee entitled "Boundary Adjacency Tracing and Its Application To Data Base Compression." I recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Michael G. Thomason
Michael G. Thomason, Major Professor

We have read this thesis
and recommend its acceptance:

R. C. Gwyer
E. H. Hall

Accepted for the Council:

L. Evans
Vice Chancellor
Graduate Studies and Research

Thesis

79

.L44

cop. 2

BOUNDARY ADJACENCY TRACING AND ITS APPLICATION TO
DATA BASE COMPRESSION

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Chien-Chyun Lee

June 1979

DEDICATION

This Thesis is dedicated
to my parents, Mr. Feng-She Lee and Mrs. Su-Ying Ling Lee,
and to my wife, Li-Na Wang Lee.

ACKNOWLEDGEMENTS

The author wishes to acknowledge with deepest appreciation his thesis advisor, Dr. Michael G. Thomason, for his advice and patience. Appreciation is also extended to committee members Dr. Ernest L. Hall and Dr. Ralph C. Gonzalez for their valuable suggestions and for review of this thesis. The author wants to express special gratitude to Dr. Ernest L. Hall for giving assistance in research and for his friendly attitude. He has never missed a chance to provide his assistance, encouragement, and constructive criticism when need in the preparation of this thesis.

The author wishes to express his appreciation to Mr. Juin-Jet Hwang for his valuable heuristic suggestions and for his review of the thesis. The author also wishes to extend his gratitude to Mr. Jim Killingsworth for his help in editing the thesis.

Finally, it gives the author special pleasure to express his gratitude to his wife, Li-Na Wang, for the many ways in which she has contributed to the preparation of this thesis.

ABSTRACT

Based on a well-segmented image, a series of methods leading to the compression of the boundary adjacency data base without any additional distortion is presented. The size of the compressed version for a regular image is $1/12$ to $1/3$ the original data base.

A well-segmented image has two-dimensional pixels clustered into regions. An easy method for extracting the singular boundary pixels, which requires special attention due to aspects of sequencing the boundary pixels, is discussed. In addition, a convenient way for identifying these boundary pixels is presented.

The storage structure of the boundary adjacency data base is a three-level hierarchical structure based on index and SRA (Stored Record Address) concept. A simple way to trace the adjacency relationships of a region to other regions and extract the boundary pixels of each adjacent edge of this region at the same time is presented.

The sequencing of the boundary pixels of the adjacent edges of a region provides a way to compress the boundary adjacency data base. Using a circular-linked list structure, the entire set of boundary pixels of a region is sequenced clockwise, with the singular boundary pixels being sequenced recurrently. Then, with reference to the sequenced boundary, a method that orders (in a clockwise sequence) the boundary pixels of each adjacent edge in the adjacency data base is developed. The possibility that the adjacent edge of a region would be cut into several separable sections is considered. Finally, three

ways using a direction vector code to compress the boundary adjacency data base effectively are presented, compared, and discussed.

The sequencing of the boundary, the detecting of the boundary adjacency, and the coding of the adjacent edge might be directed toward such further studies as relationship analysis of the scene content, vertex locating, reconstruction of the scene, and the shape description.

TABLE OF CONTENTS

SECTION	PAGE
I. INTRODUCTION	1
II. DEFINITION AND CONVENTION	6
III. BOUNDARY PIXEL TRACING	11
IV. BOUNDARY PIXEL SEQUENCING	17
V. BOUNDARY ADJACENCY DETECTING	30
VI. ADJACENT EDGE SEQUENCING	36
VII. CODING SCHEME FOR BOUNDARY ADJACENT DATA BASE COMPRESSION	53
VIII. EXPERIMENTAL RESULTS AND DISCUSSION	67
IX. APPLICATION AND CONCLUSION	80
LIST OF REFERENCES	82
APPENDIX	85
VITA	130

LIST OF FIGURES

FIGURE	PAGE
1. A 3x3 template showing inner pixel (j,i)	7
2. Circular-linked list	10
3. Flowchart for boundary pixel tracing algorithm	14
4. Priorities for sequencing 8 possible eight-connective boundary pixels	19
5. An example for a region containing singular boundary pixels	21
6. Example of some regular boundary pixels that need to be recurrently sequenced	24
7. Flowchart for boundary pixel sequencing algorithm	26
8. The storage structure of the boundary adjacency data base	31
9. Flowchart for boundary adjacency detecting algorithm	34
10. Six adjacent edges of region 4	38
11. Flowchart for adjacent edge sequencing algorithm	42
12. Original image picture	46
13. Segmented and averaged image picture	47
14. Boundaries generated from sequenced boundary pixel data base	48
15. Adjacent edges generated from sequenced boundary adjacency data base	49
16. Direction codes for compression coding schemes	54
17. Example for the first coding scheme	57
18. Example for the second coding scheme	58
19. Super-direction codes for the third coding scheme	62
20. Example for the third coding scheme	66

I. INTRODUCTION

Most of the advanced scene content analysis techniques require that the picture be subdivided or segmented in some way. One general approach is to segment the image into regions corresponding to objects or parts of objects or to background. Once the image has been segmented, these regions can be represented and described in various ways. These descriptions provide properties or features that are useful, either directly for object classification or indirectly as a step toward scene description.

Since every region in an image has its boundary adjacent to other regions, much of the important information of a scene is contained in these boundaries. Thus, the boundary is often considered to be a fundamental primitive in the scene. This characteristic results in a widely-used approach to segmentation based on detecting edges or curves in the pictures. Such features may be the boundaries or parts of boundaries of the regions of interest, or they may represent features of interest themselves. Once such boundaries are found, they can also be represented and described in various ways.

Since one objective of the segmentation of an image is to facilitate the detection of the relationships that are important aspects of the scene content [1,5,10,12,13,18,21], and since the boundary is a fundamental primitive in the scene, a carefully constructed boundary adjacency data base is very important for scene content analysis.

Over the past decade, considerable effort has been devoted to solving the problem of boundary detection. Many of the proposed

boundary detection methods apply local operations to every point of the input picture, in order to extract short edges or boundaries of a region. Some of them use either the gradient method (the first derivatives) or the Laplacian method (the second derivatives) [7,14,19]. Other operations allow the elimination of most of the local noise [8,18,20].

When small edges have been obtained, they must be connected to form meaningful boundaries. In order to detect meaningful boundaries in a noise image picture, it is necessary to have global information about the boundaries in the image and about the structure of the objects they represent. Usually this global information is heuristic and is embedded implicitly in the computer algorithm. However, this approach has the disadvantage the minor changes in the characteristics of the sought boundaries of regions may require extensive reprogramming [16].

In order to avoid such disadvantages and to provide a simple way of detecting boundary adjacency, a series of methods are introduced in the following sections for tracing the boundaries of regions and then setting up a boundary adjacency data base. These methods are based on an image presegmented by the histogram classification and GLECS (Global Local Edge Coincidence Segmentation) techniques [11,12,13,21].

There are six advantages to this approach:

1. The boundaries of the regions in the image are always closed.
2. The efficiency of the algorithm reduces the computing time and the memory space needed to a size compatible with a typical mini-computer.
3. The boundary adjacency data base for scene content analysis can be easily developed.

4. Sequencing the boundary contributes to the compression of the boundary adjacency data base.

5. Shape descriptions of scene content may be made [10].

6. Vertices of interest can be located [11].

Once a boundary in the image has been extracted, it can be represented and described in a variety of ways. As is the case with region representations, some methods preserve all of the information about the contour, while other preserve only what is of interest for a particular task. The image processed by the computer is basically a two-dimensional array of data in which one datum is called a "pixel." Therefore, the pixels representing the boundaries of the regions in the image form fundamental primitives in the boundary adjacency data base.

Several techniques that view the boundary as a smooth curve have been introduced during the past ten years [7,16,18]. One basic way to represent a smooth curve mathematically is to give its slope or curvature as a function of arc length. Such representations are known as intrinsic equations for the curve. The so-called "chain code" developed by Freeman is a typical example of such intrinsic equations for the discrete curve [4]. This representation of boundaries may be used to compress the boundary adjacency data base but will lose some information about the original boundaries and generate some distortion when used for reconstruction. This distortion results from the fact that the pixels representing the boundaries in the image are always discrete and finite in number, whereas the true boundaries are continuous.

Based on the discrete data viewpoint, a series of methods are applied in this thesis to sequence the data representing each adjacent edge in the boundary adjacency data base; then, by using the direction vector coding technique, the boundary adjacency data base can be greatly compressed without any additional distortion.

This thesis introduces three techniques for compression of the boundary adjacency data base. All of these are based on direction vector coding, but with different coding schemes. Four representative images are included to provide comparisons of these three coding schemes.

The first coding scheme results in a compression to $1/4$ to $1/3$ the original size. The second coding scheme assures that the result of the compression is on the order of $1/62$ to $1/2$. The third coding scheme, which combines the first coding scheme with the second, assures that the result of compression is on the order of $1/12$ to $1/3$ the original size.

This thesis is divided into nine sections following this one. In Section II, some definitions and notational conventions are provided, and in Section III, the method for tracing the boundary pixels is presented. Section IV describes the paradigm for sequencing boundary pixels, while Section V provides a method for quickly setting up the boundary adjacency data base. In Section VI, the strategy for sequencing the adjacent edges clockwise in the boundary adjacency data base is described, and Section VII deals with three coding schemes for the compression of the sequenced boundary adjacency data base. In Section VIII, the experimental compression results for the three coding schemes are discussed. Section IX contains concluding remarks and points out

areas recommended for future research. Finally, the Appendix provides the computer programs for the algorithms which follow in the body of this thesis.

The computer programs shown in the Appendix are written in FORTRAN language. The input presegmented image is a 128×128 array (16384 pixels) of data, in which a number ranging from -128 to 127 is stored for each pixel's value. For a 256-word (one word equals two bytes) block size, 40 block storage space is required for an image file, since each datum is a LOGICAL*1 byte. The first eight blocks of the image file are used for storing the descriptive information of the picture, i.e., title, size, date, convolution. The remaining 32 blocks, amounting to 16384 LOGICAL*1 bytes, are used to store sequentially the data of 16384 pixels of the image picture from row 1 to row 128 and in each row from column 1 to column 128.

Before describing the following algorithms, some conventions needed to avoid ambiguity are presented.

II. DEFINITION AND CONVENTION

Some of the terms which are used in the following text are defined in this section. Terms such as node, circular-linked list, and sequential linear list have been widely used in describing data structures. Some new terms, such as inner pixel, regular boundary pixel, and singular boundary pixel are also introduced for convenience.

Pixel

An image can be considered to be a set of data in a matrix whose row and column indices identify a point in the picture. The elements of such a matrix are called pixels [5].

Inner Pixel

A pixel $P(j,i)$ on a 3×3 template is defined as an inner pixel, if

$$P(j,i) = P(j+1,i) = P(j-1,i) = P(j,i+1) = P(j,i-1).$$

(See Figure 1.)

Boundary Pixel

A pixel which is not an inner pixel is called a boundary pixel.

Eight-Connective

On a 3×3 template, each of the locations $(j+1,i)$, $(j+1,i+1)$, $(j,i+1)$, $(j-1,i+1)$, $(j-1,i)$, $(j-1,i-1)$, $(j,i-1)$, and $(j+1,i-1)$ is eight-connective to location (j,i) (Figure 1).

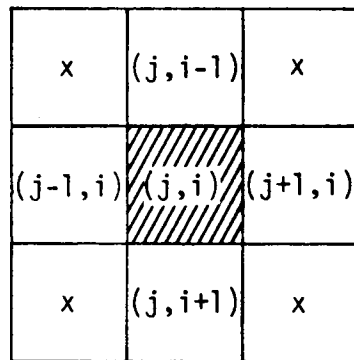


Figure 1. A 3x3 template showing inner pixel (j,i).

Adjacent Pixel and Adjacency

A pixel which is eight-connective to a reference pixel on a 3×3 template is defined as an adjacent pixel to this reference pixel.

Region

A region in the image is a set of adjacent pixels with the same value.

Singular Boundary Pixel

A boundary pixel of region R to which no inner pixel of region R is adjacent is called a singular boundary pixel of region R.

Regular Boundary Pixel

A boundary pixel which is not a singular boundary pixel is called a regular boundary pixel.

Adjacent Edge

The adjacent edge of region A to region B is a set of the boundary pixels of region A, whose pixels are adjacent to the pixels of region B.

Boundary

The union set of the adjacent edges of region A (in other words, the set of the boundary pixels of region A) is called the boundary of region A.

Node

An element in a list, which contains at least one field, such as an INFO field or a LINK field, is called a node in this list [15].

Circular-Linked List

A circular-linked list is one whose nodes have at least one LINK field and are linked one by one, such that no matter at which node a trace is started, the original node can be traced through its linkage information [15] (Figure 2).

Sequential Linear List

A sequential linear list (sometimes called simply a sequential list) is a set of N nodes (N is greater than or equal to 0) that are linearly ordered [15].

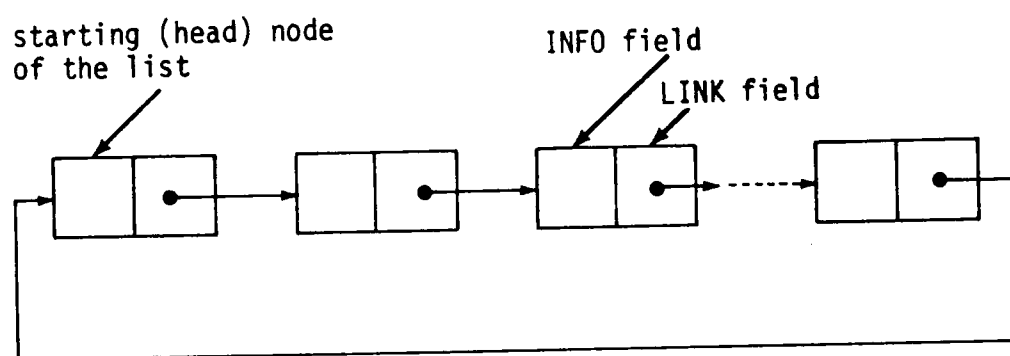


Figure 2. Circular-linked list.

III. BOUNDARY PIXEL TRACING

A simple method for boundary pixel tracing based on the segmented image will be introduced in this section. The segmented image contains an $N \times N$ array of pixels ($N = 128$ is assumed for convenient description of the following algorithm.). In the segmented image, each pixel contains a number indicating to which region it belongs instead of the grey level number shown in the original image.

The algorithm shown below is sequenced from left to right and from top to bottom once as it traces the boundary pixels. The output of this algorithm is a circular-linked list representation of the boundary pixels. A circular-linked list data structure is used to facilitate the boundary pixel sequencing algorithm described in the next section. The set of these lists may be viewed as a boundary pixel data base with a key as the region number.

The region number is used as an index of the region pointer, and through the region pointer all the boundary pixels of the corresponding region can be extracted by following the linkage order in the corresponding circular-linked list. Once a boundary pixel is extracted, its location on the image array is stored in the INFO field of a node, which is then inserted into the circular-linked list at the right side of the last node. The use of the circular-linked list data structure results in dynamic storage capability and efficient utilization of the memory space in a mini-computer.

Throughout this thesis it is assumed that the datum of a pixel is represented by a LOGICAL*1 byte. For a LOGICAL*1 byte (eight bits),

numbers ranging from -128 to 127 can be represented. The maximum number of the regions is 256; therefore, it is always necessary to have one block of (256) INTEGER*2 words of memory space for the region pointers in the boundary pixel data base.

The boundary pixel data base keeps track of the locations of the boundary pixels in the image array. When a boundary pixel is stored in a node, its two-dimensional location (x,y) will be transformed into a one-dimensional location (z) through the formula

$$z = x + (y - 1) * N,$$

for an N*N image array.

Each node in the circular-linked list contains two INTEGER*2 words. One word stores the one-dimensional location of the boundary pixel, and the other word stores the linkage pointing to the next node in the same circular-linked list.

In order to reduce computing time, a temporary N*N array of LOGICAL*1 byte corresponding to the N*N image array, which shows which pixels are inner pixels, regular boundary pixels, or singular boundary pixels, is used to store the index information about the pixels. An index number "1" is used to indicate the inner pixels; "2" indicates regular boundary pixels; and "3" indicates singular boundary pixels. Since the original image array is assumed in the first quadrant, a negative number for one-dimensional location is used to differentiate between the singular boundary pixel and the regular boundary pixel. The detection and isolation of the singular boundary pixels are helpful for the boundary pixel sequencing algorithm in the next section.

Since the first row and column and the last row and column in the image array correspond to the frame of the picture, this algorithm starts to scan all pixels from the second row to the next to the last row and from the second column to the next to the last column. A number "-5" is stored in the region pointer to indicate that the corresponding circular-linked list is empty.

A concise flowchart of the above algorithm is shown in Figure 3. Algorithm steps are as follows:

STEP 1: Scan each pixel to detect whether it is an inner pixel or a boundary pixel. If it is an inner pixel, store number "1" in the corresponding element of the temporary index array; otherwise, store number "2."

STEP 2: By referring to the temporary index array, scan all boundary pixels to detect which are singular boundary pixels and which are regular boundary pixels. For each singular boundary pixel, change the number "2" stored in the corresponding element of the temporary index array to number "3."

STEP 3: Scan the next pixel. If all pixels have been scanned, go to STEP 9; otherwise, go to STEP 4.

STEP 4: If the pixel being scanned is an inner pixel, go back to STEP 3; otherwise, go to STEP 5.

STEP 5: Transform the two-dimensional location of the pixel being scanned to the one-dimensional location representation.

STEP 6: If the pixel being scanned is a regular boundary pixel, go to STEP 8; otherwise, go to STEP 7.

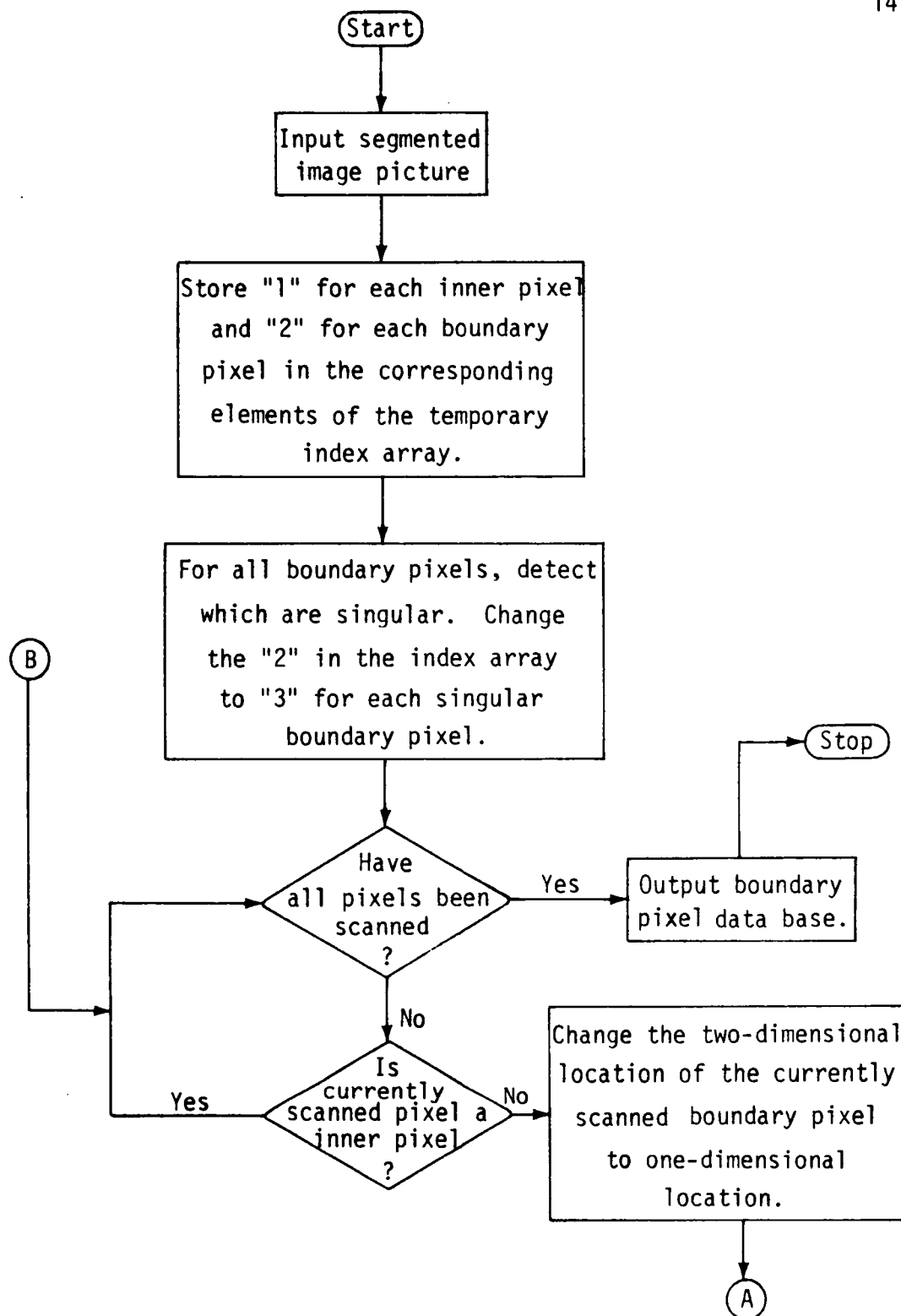


Figure 3. Flowchart for boundary pixel tracing algorithm.

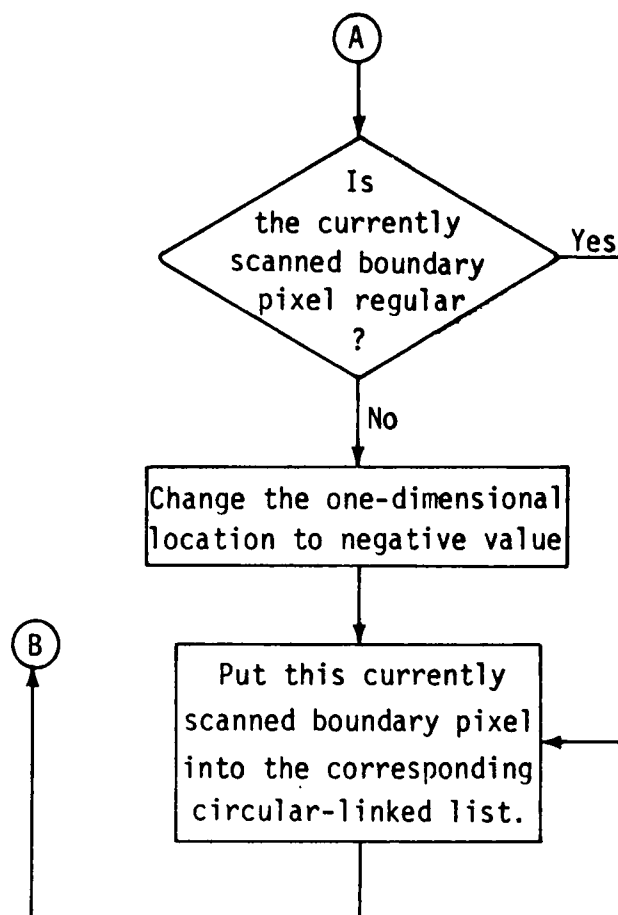


Figure 3 (Continued).

STEP 7: Change the value of the one-dimensional location of the pixel being scanned to a negative value to indicate that it is a singular boundary pixel.

STEP 8: Create a node to store the one-dimensional location of the pixel being scanned and link it to the last inserted node of the corresponding circular-linked list, then go to STEP 3.

STEP 9: Output the boundary pixel data base.

IV. BOUNDARY PIXEL SEQUENCING

Boundary pixel sequencing can be widely used in image data processing for such applications as the study of shape description in scene content analysis [3,5,10], the location of vertices [15], and the compression of the boundary (adjacency) data base.

It is the activity in which all pixels on the boundary of a region are sequenced in the order in which they appeared on the boundary of a region in the picture. In other words, they are sequenced in such an order that a pencil could be used to draw an outline of the region in the picture.

The boundary pixel sequencing algorithm described below is based on the boundary pixel data base set up as a circular-linked data structure. The boundary pixel data base is processed, region by region, by using the region number as a key to extract the boundary pixels from its corresponding circular-linked list. The output of this algorithm (sequenced boundary pixel data base) is also set up as a circular-linked list data structure to prepare the way for the algorithm of the adjacent edge sequencing described in Section VI.

The determination of the priority of the boundary pixels to be sequenced is the key step in this sequencing algorithm. The following strategy for determining such priority will force all boundary pixels to be sequenced in a clockwise direction if the operation starts at the boundary pixel which is on the top-leftmost location of the boundary of a region. The boundary pixel tracing algorithm previously described assures this starting point because the tracing algorithm follows the sequence left-to-right and top-to-bottom.

In the following strategy description, List A will be used to indicate the original circular-linked list in the boundary pixel data base; List B will indicate the corresponding circular-linked list in the sequenced boundary pixel data base. At the beginning, the first boundary pixel extracted from List A is put directly into the corresponding List B. As soon as an additional boundary pixel in List A is ready to be put into List B (i.e., sequenced), it is immediately taken off List A. (The node associated with this sequenced boundary pixel is deleted from List A.) Once a boundary pixel is removed from List A, it serves as a reference boundary pixel for those pixels remaining in List A in order to determine the priority of the sequencing. This priority is decided by the following strategy. (See Figure 4.)

If the two-dimensional location of the reference boundary pixel is (j,i) , then the four possible four-connective boundary pixels [locations $(j+1,i)$, $(j-1,i)$, $(j,i+1)$, and $(j,i-1)$] adjacent to the reference boundary pixel have priority in sequencing over the remaining four possible eight-connective boundary pixels [locations $(j+1,i+1)$, $(j-1,i+1)$, $(j+1,i-1)$, and $(j-1,i-1)$]. Location $(j+1,i)$ has first priority to be sequenced; location $(j,i+1)$, second priority; location $(j-1,i)$, third priority; and location $(j,i-1)$, fourth priority. For the remaining four possible pixels, location $(j+1,i+1)$ has first priority to be sequenced; location $(j-1,i+1)$, second priority; location $(j-1,i-1)$, third priority; and location $(j+1,i-1)$, fourth priority. If a boundary pixel is not adjacent to the reference boundary pixel, it has no priority to be sequenced.

A three-element buffer array, `INTEGER*2 BUFR(3)`, is used to store temporarily the information about the boundary pixel likely to be

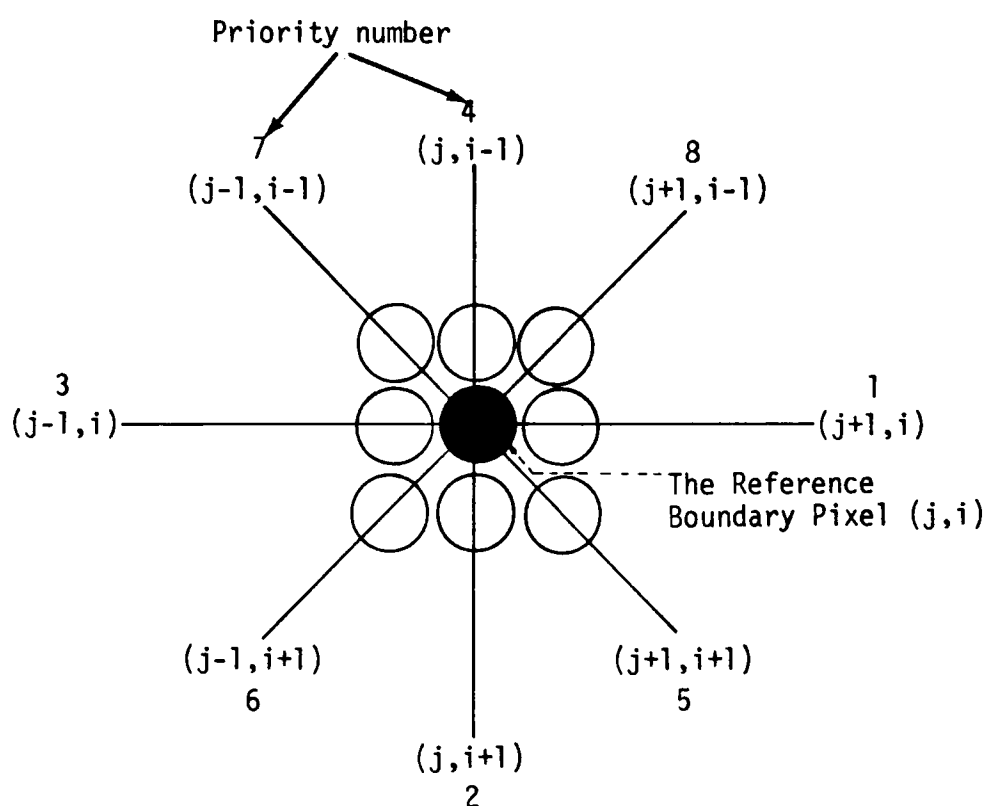


Figure 4. Priorities for sequencing 8 possible eight-connective boundary pixels. (Note: small number indicates higher priority of sequencing.)

sequenced. BUFR(1) stores the memory location of the node just previous to the likely sequenced boundary pixel in List A. BUFR(2) stores the memory location of the node associated with this likely sequenced boundary pixel in List A. BUFR(3) stores a number indicating the priority of this boundary pixel.

All remaining boundary pixels in List A are traced to decide which boundary pixel has the highest sequencing priority. If a boundary pixel has higher priority than the boundary pixel in the buffer array, then the contents of the buffer array will be replaced with the information about the boundary pixel with higher priority. After all of the boundary pixels remaining in List A have been traced, the information stored in the buffer array shows which boundary pixel in List A will be sequenced next. The memory locations stored in BUFR(1) and BUFR(2) assure a straightforward way to insert the new node into List B and to delete the node from List A.

The possible existence of singular boundary pixels gives rise to a problem: If a set of boundary pixels contains singular boundary pixels, the set usually cannot be sequenced as a closed chain which allows the use of one-way sequencing without repetition. Figure 5 illustrates such a problem.

In order to keep the sequential order continuous, such singular boundary pixels are sequenced recurrently. The detection scheme for the singular boundary pixels is simply based on the definition of the singular boundary pixel shown in Section II. In order to assure that the number of the recurrently sequenced boundary pixels is as small as possible, the singular boundary pixels are sequenced after all the regular boundary pixels.

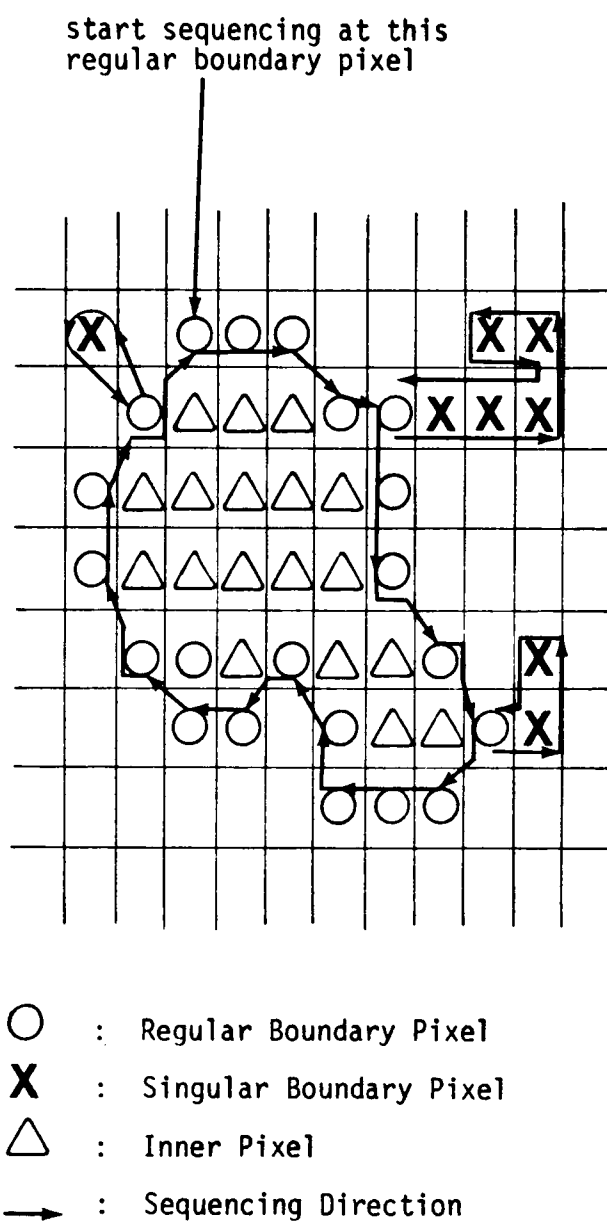


Figure 5. An example for a region containing singular boundary pixels.

When the singular boundary pixels are sequenced, a temporary circular-linked list, called List C in the following description, is used to assure that the singular boundary pixels are recurrently inserted into List B. If, after all regular boundary pixels have been sequenced, List A is not empty, then singular boundary pixels exist, and the following operations are performed.

In the first operation, the sequenced boundary pixels are traced in List B. In this operation, each time a pixel in List B is taken as a reference boundary pixel to List A, the pixels in List A are traced to determine which pixel in List A is adjacent to this reference boundary pixel. If there is a pixel in List A (referred to as pixel S) adjacent to such a reference boundary pixel in List B (referred to as pixel R), then the memory locations of the pixel S and the pixel R are kept; otherwise, the next pixel in List B is taken as a reference boundary pixel, and this operation is repeated.

In the second operation, starting at pixel S in List A and following the previously described sequencing scheme, the singular boundary pixels are sequenced in List A, and the sequenced singular boundary pixels are kept in List C. The second operation stops when no more singular boundary pixels in List A can be sequenced.

In the third operation, by copying all the singular boundary pixels in the List C from the first pixel to the last pixel and then from the second to the last pixel back to the first one, all the singular boundary pixels in List C are inserted into List B after the node associated with pixel R. If List A is still not empty, these three operations are repeated.

Even if no singular boundary pixels are involved, there still exists a problem: If a region is very narrow, its boundary pixels probably cannot be sequenced into a closed chain. Figure 6 is an example showing such a problem.

In Figure 6, the symbol of triangle stands for an inner pixel of some region (referred to as region A) in a picture. Circles a through l represent some of the boundary pixels of region A. It is clear that all of these are regular boundary pixels, because they are all adjacent to some inner pixels. The arrow shows the direction of sequencing. According to the previously described sequencing scheme, the operation of the sequencing will stop at the boundary pixel h, because no pixel remaining in List A is adjacent to pixel h. However, if pixel d and pixel e are repeatedly sequenced, the operation of the sequencing will continue to pixel i, from there to pixel j, pixel k, pixel l, and so on.

In order to pick up exactly such pixels as d and e for repeated sequencing, the following operations are applied. The "adjacency order" of the regular boundary pixel D is defined as the number of the regular boundary pixels adjacent to the regular boundary pixel D. For example, in Figure 6 boundary pixels d and e have adjacency order 3. Before starting to sequence the boundary pixels, the adjacency order of each regular boundary pixel must be detected. If the adjacency order of some regular boundary pixel is higher than 2, its one-dimensional location and adjacency order must be kept in the buffer array.

If the sequencing operation stops without having formed a closed chain, the operation of the sequencing is transferred from List A to the buffer array. The buffer array is traced to determine which boundary

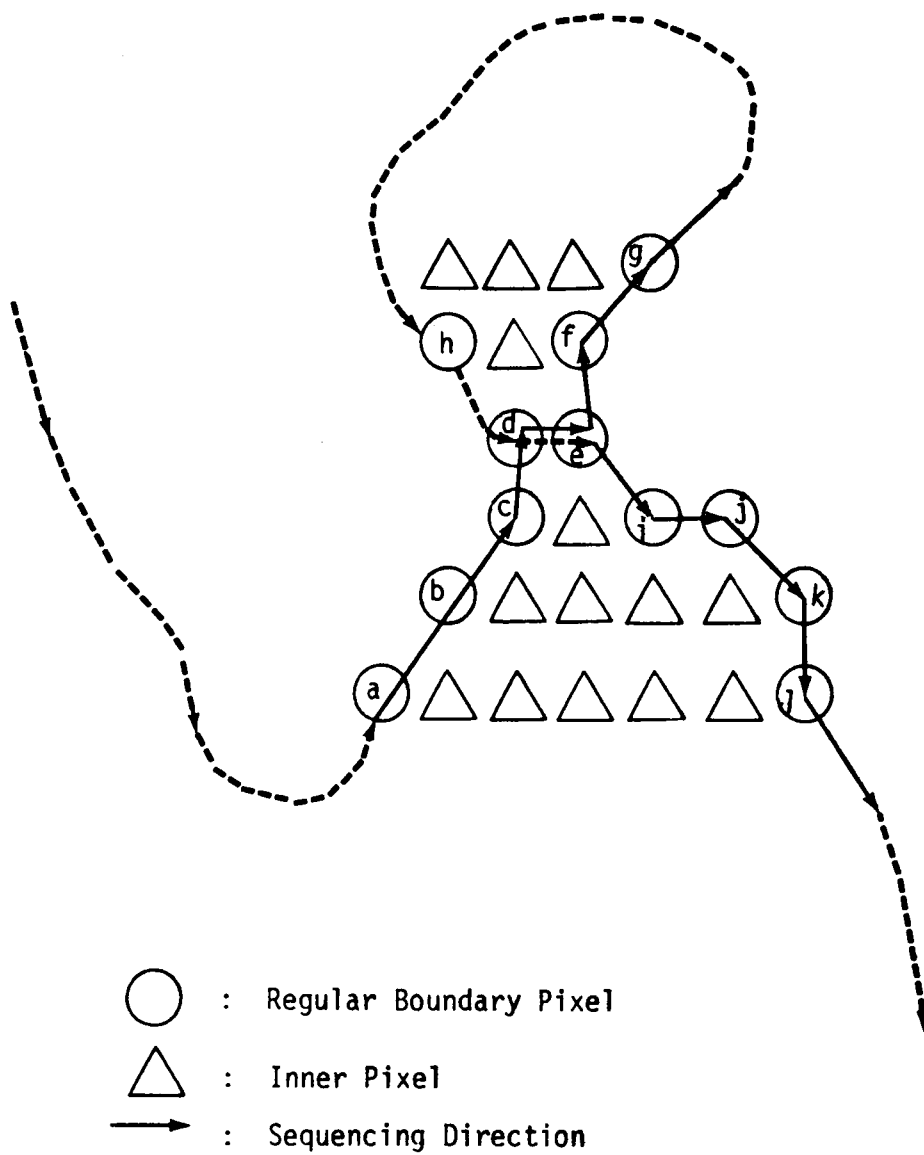


Figure 6. Example of some regular boundary pixels that need to be recurrently sequenced.

pixel with higher adjacency order is adjacent to the reference boundary pixel. When such a boundary pixel in the buffer array has been put into List B, it is taken off the buffer array. Then the operation of sequencing is transferred from the buffer array back to List A. If the operation of sequencing still stops without having formed a closed chain, it will again be transferred to the buffer array.

For example, in Figure 6 the sequencing operation stops first at pixel h and is transferred to the buffer array to get pixel d as the next sequenced boundary pixel; then the sequencing operation is transferred back to List A. The sequencing operation then stops at pixel d and is again transferred to the buffer array to get pixel e as the next sequenced boundary pixel. At this time, when the sequencing operation is again transferred back to List A, it will keep on sequencing pixel i, pixel j, pixel l, and so on.

Figure 5 is an example of this boundary pixel sequencing algorithm. Figure 7 shows a brief flowchart of this algorithm, whose step descriptions are as follows:

STEP 1: Input List A (the output of the boundary pixel tracing algorithm).

STEP 2: For each regular boundary pixel, compute adjacency order. For each pixel with adjacency order greater than 2, put its location and adjacency order into the buffer array.

STEP 3: If List A is empty, output List B; otherwise, go to STEP 4.

STEP 4: If List A contains no regular boundary pixel, go to STEP 10, otherwise, go to STEP 5.

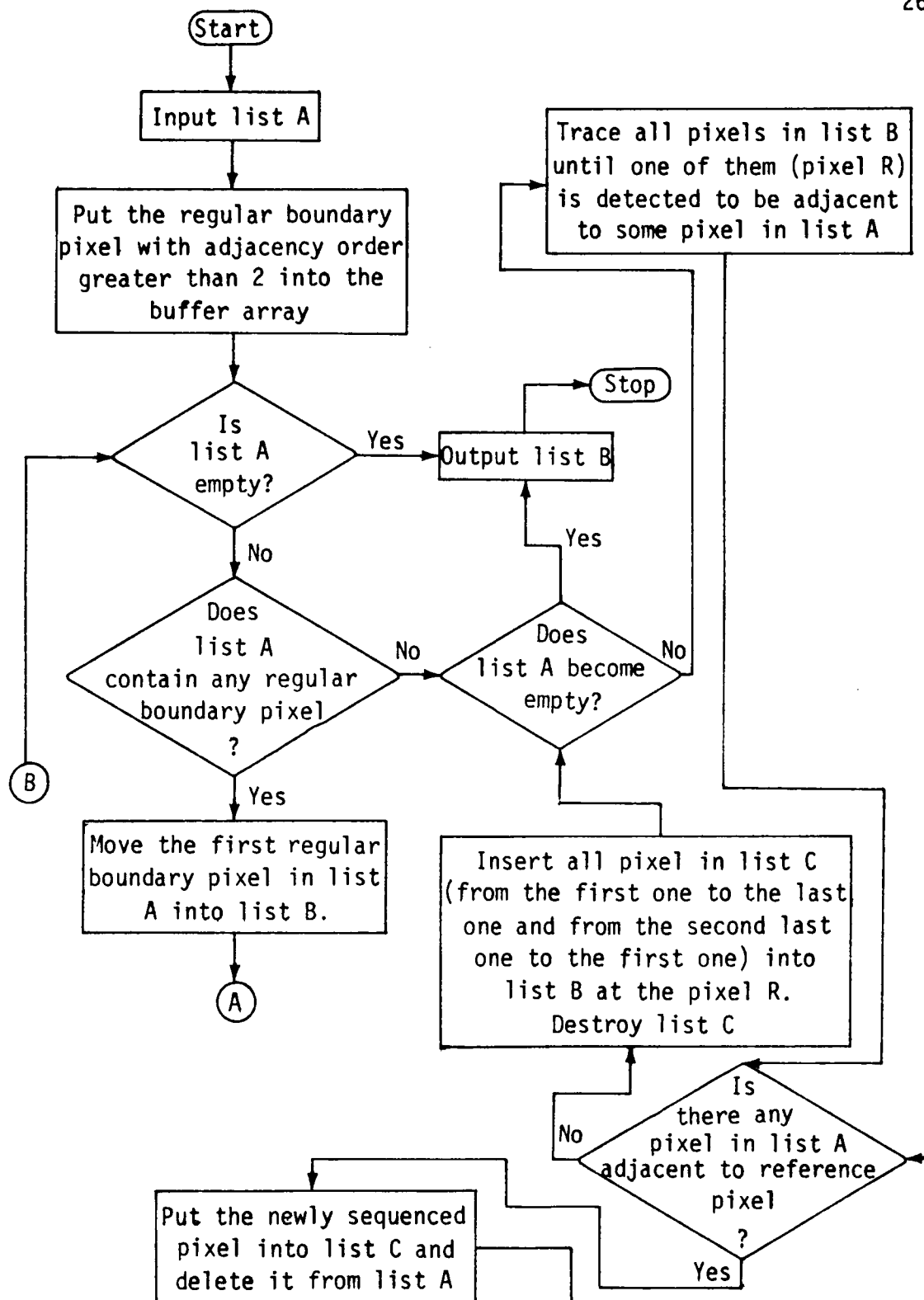


Figure 7. Flowchart for boundary pixel sequencing algorithm.

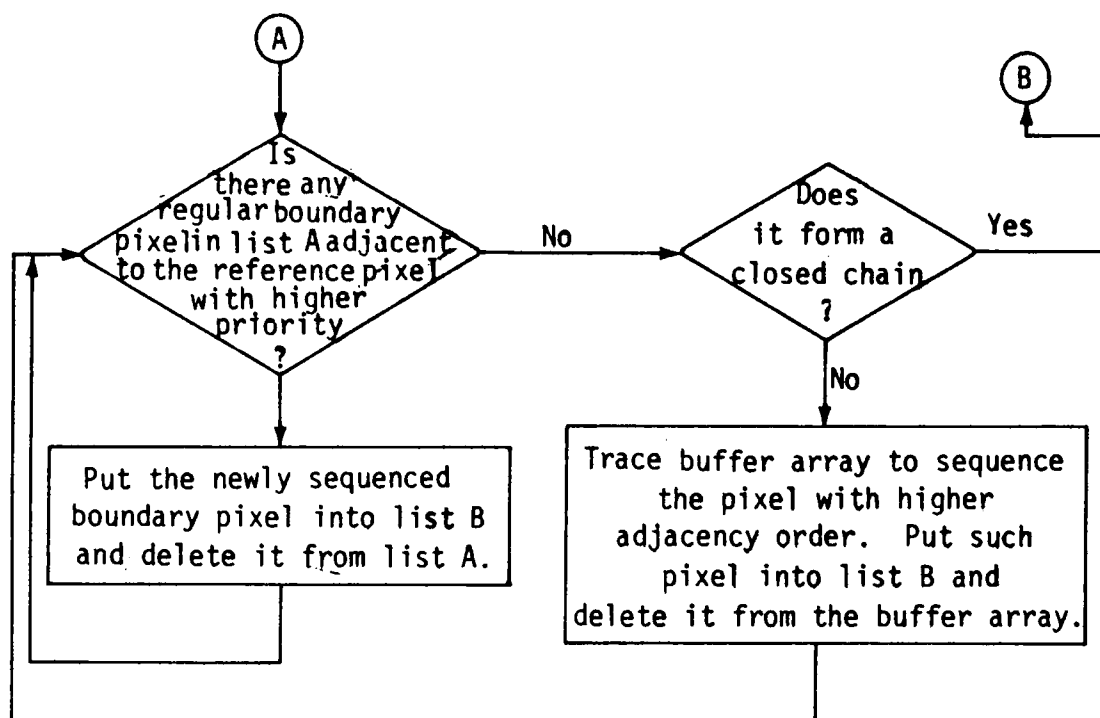


Figure 7 (Continued).

STEP 5: Get the first regular boundary pixel from List A. Put it into List B and delete it from List A. Refer to it as the current reference boundary pixel.

STEP 6: Trace all regular boundary pixels in List A to see which one should be taken as the next sequenced boundary pixel. If no pixel is adjacent to the reference boundary pixel, go to STEP 8; otherwise, go to STEP 7.

STEP 7: Put the newly sequenced boundary pixel into List B and delete it from List A. Refer to it as the current reference boundary pixel. Go back to STEP 6.

STEP 8: If a closed chain has been formed, go back to STEP 3; otherwise, go to STEP 9.

STEP 9: Trace all pixels in the buffer array to see which pixel adjacent to the current reference boundary pixel has greater adjacency order. Put this pixel into List B and delete it from the buffer array. Refer to it as the current reference boundary pixel. Go back to STEP 6.

STEP 10: If List A becomes empty, output List B; otherwise, go to STEP 11.

STEP 11: Trace all pixels in List B until one of them, called pixel R, is found to be adjacent to some pixel in List A. Refer to it as the current reference boundary pixel.

STEP 12: Trace all pixels in List A to see which one has priority to be taken as the next sequenced boundary pixel. If no pixel is adjacent to the reference boundary pixel, go to STEP 14; otherwise, go to STEP 13.

STEP 13: Put the newly sequenced pixel into List C and delete it from List A. Refer to it as the current reference boundary pixel. Go back to STEP 12.

STEP 14: Insert all pixels into List C (from the first one to the last one and from the second to the last one back to the first one) into List B at the pixel R. Then destroy List C. Go back to STEP 10.

V. BOUNDARY ADJACENCY DETECTING

This section introduces a simple algorithm to quickly detect the adjacency of the boundaries. The output of such an algorithm is called a boundary adjacency data base, which keeps the adjacency information and the pixels of the adjacent edges. Its storage structure is a three-level hierarchical structured based on the index and SRA (Stored Record Address) concept [2]. Figure 8 shows this storage structure.

The data stored in each level of this storage structure are always in sequential list representation. Each sequential list has a node containing a number "-5" (or "0") for an end mark. The first level is a one-dimensional array containing 256 elements. Each element is an INTEGER*2 word for a region pointer storing the starting position of the corresponding sequential list in the second level. The region number is used as the index in this array.

The second level is a two-dimensional array for the "adjacency table." The element in the adjacency table contains two INTEGER*2 words. The first word stores the adjacent region number, and the second word is a pointer storing the starting position of the corresponding sequential list in the third level. For example, in Figure 8 the fact that the element REGPTR(i) in the first level points to the element x containing adjacent region number j in the second level means that in region i there is an edge adjacent to region j, and the boundary pixels of such an edge can be extracted from the third level of the data base through the pointer of the element x in the adjacency table. Furthermore, from the

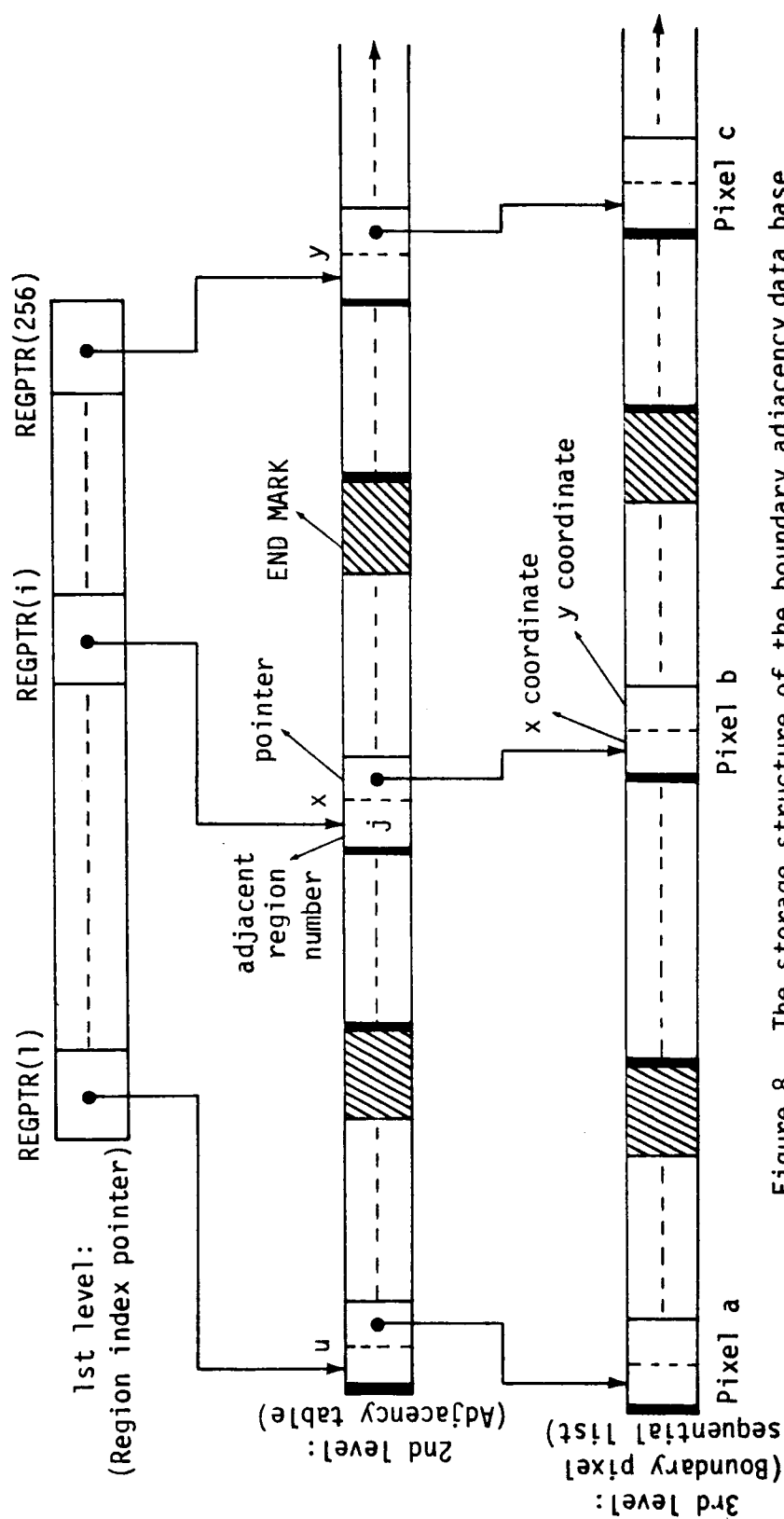


Figure 8. The storage structure of the boundary adjacency data base.

element x in the adjacency table, all the adjacent edges of region i can be extracted from the sequential list in the table.

The third level is a two-dimensional array for storing the boundary pixels of the adjacent edges in the sequential lists. Each adjacent edge corresponds to a sequential list. The element of this array contains two LOGICAL*1 bytes. The first and second bytes store the x coordinate and y coordinate locations of the pixel in the image array respectively.

Since the following algorithm is based on the left-to-right and top-to-bottom operational order, the boundary pixels of the adjacent edge in the sequential list are not in sequential order. In the next section, the method for sequencing the boundary pixels of the adjacent edge is described.

To aid the description, suppose that the adjacency of region c is being detected. Once a boundary pixel of region c , called pixel h , is detected, its eight-connective pixels are put into a buffer. Then, for these eight-connective pixels in the buffer, if there is more than one pixel belonging to region i (i is not equal to c), these pixels of region i are eliminated, such that only one pixel of region i is left in the buffer; and if there are pixels belonging to region c , all of them are eliminated. Thus the pixels remaining in the buffer will indicate to which regions the boundary pixel h is adjacent, and the number of the pixels remaining in the buffer will show how many regions to which the boundary pixel h is adjacent.

If the boundary pixel h is adjacent to more than one region, it is probably an end point of the adjacent edge. If boundary pixel h is

detected to be adjacent to n regions (n can range from 1 to 8), then the next operation is to put boundary pixel h into the n temporary circular-linked lists corresponding to its n adjacent regions respectively. After all the boundary pixels of region c have been traced, the pixels stored in the temporary circular-linked lists are transferred respectively to the corresponding sequential lists in the third level of the data base.

Figure 9 shows the concise flowchart of this algorithm, whose steps are summarized as follows:

STEP 1: Detect how many regions need to be processed.

STEP 2: If all regions have been processed, output the boundary adjacency data base; otherwise, go to STEP 3 to process the next region (refer to it as region c).

STEP 3: If all boundary pixels have been traced, go to STEP 7; otherwise, repeat this step until one boundary pixel is detected; then go to STEP 4.

STEP 4: Put the eight eight-connective pixels of the currently detected boundary pixel into the buffer.

STEP 5: For these eight pixels in the buffer, if there is more than one pixel belonging to region i (i is not equal to c), eliminate these pixels of region i from the buffer, such that only one pixel of region i is left in the buffer; and if there are pixels belonging to region c , eliminate all of them.

STEP 6: If the currently detected boundary pixel is adjacent to n regions (n can range from 1 to 8), put it into the n temporary circular-linked lists corresponding to its n adjacent regions respectively. Then go back to STEP 3.

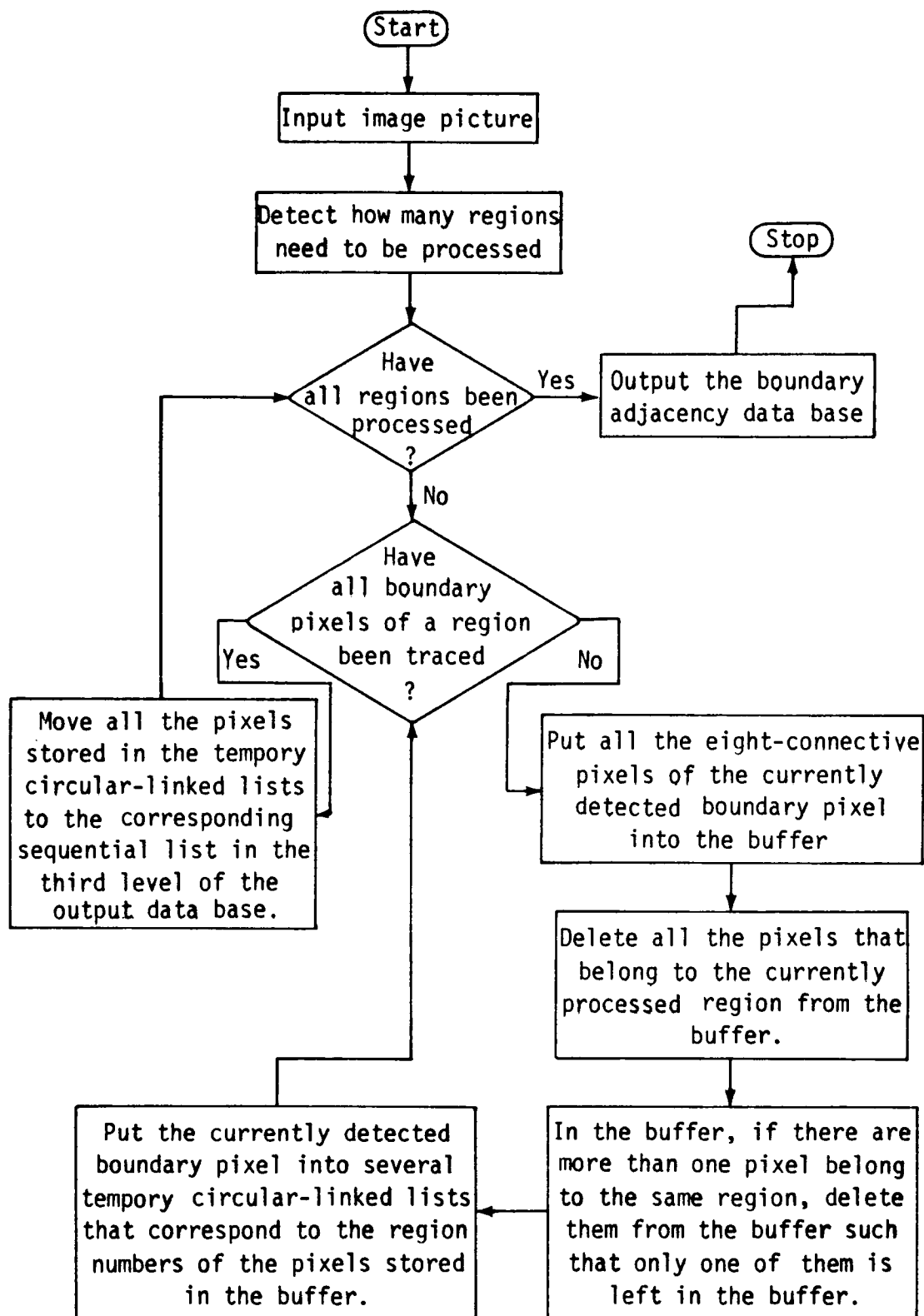


Figure 9. Flowchart for boundary adjacency detecting algorithm.

STEP 7: Transfer the pixels stored in the temporary circular-linked lists to the corresponding sequential list in the third level of the boundary adjacency data base. (The adjacency information can be recorded in the adjacency table.) Then go back to STEP 2.

VI. ADJACENT EDGE SEQUENCING

The previous section introduced a simple way to detect the adjacency of the boundaries and the storage structure of the boundary adjacency data base; but the boundary pixels of the adjacent edges stored in the data base have not been placed in sequential order. In this section, a method for sequencing the adjacent edges stored in this boundary adjacency data base is presented.

The output of the sequencing algorithm described in Section IV has prepared the way for sequencing these adjacent edges in such a boundary adjacency data base, for all the boundary pixels of all adjacent edges have been globally sequenced into several closed boundaries, such that each closed boundary represents one region in the picture. Therefore, an algorithm is now introduced that uses the sequenced boundary data base shown in Section IV as the global reference information for the adjacent edge sequencing. The adjacent edge sequencing algorithm has two input data bases. One is the output data base of the algorithm shown in Section IV, i.e., the sequenced boundary pixel data base; the other is that of the algorithm shown in Section V., i.e., the boundary adjacency data base.

The output data base of this algorithm is called the sequenced boundary adjacency data base. The storage data structure of this data base is almost the same as that of the boundary adjacency data base shown in Figure 8, except that the element in the third level of the data base is changed to an INTEGER*2 word instead of two LOGICAL*1 bytes.

This adjacent edge sequencing algorithm is still based upon a region-by-region processing procedure. For a given region number, the corresponding circular-linked list (representing the region's sequenced closed boundary) can be extracted from the sequenced boundary pixel data base. The adjacency and all the pixels of an adjacent edge of the corresponding region can also be extracted from the adjacency table of the boundary adjacency data base.

For example, if region 4 in Figure 10 is being processed, and if all the boundary pixels of this region have been extracted in the corresponding circular-linked list, then the regions which are adjacent to region 4 can be determined from the adjacency table in the boundary adjacency data base. The adjacent regions are thus regions 2, 3, 4, and 6.

Figure 10 shows that region 4 has two separated edges adjacent to region 2 and two separated edges adjacent to region 5. Pixel a is the current starting node of the corresponding circular-linked list. Pixels b, c, e, f, i, m, n, q, u, and v are the end points of these six adjacent edges (identified by the squares in the figure). The end point of the adjacent edge is defined as the boundary pixel which is adjacent to more than one region. Since the boundary pixels are sequenced in a clockwise direction, pixels b, e, i, m, q, and u are the starting end points of the corresponding adjacent edges. The starting end point of an adjacent edge is defined as the first traced end point of that adjacent edge in a clockwise direction.

Finding the starting end point of the adjacent edge is an important part of this adjacent edge sequencing algorithm. Once the starting

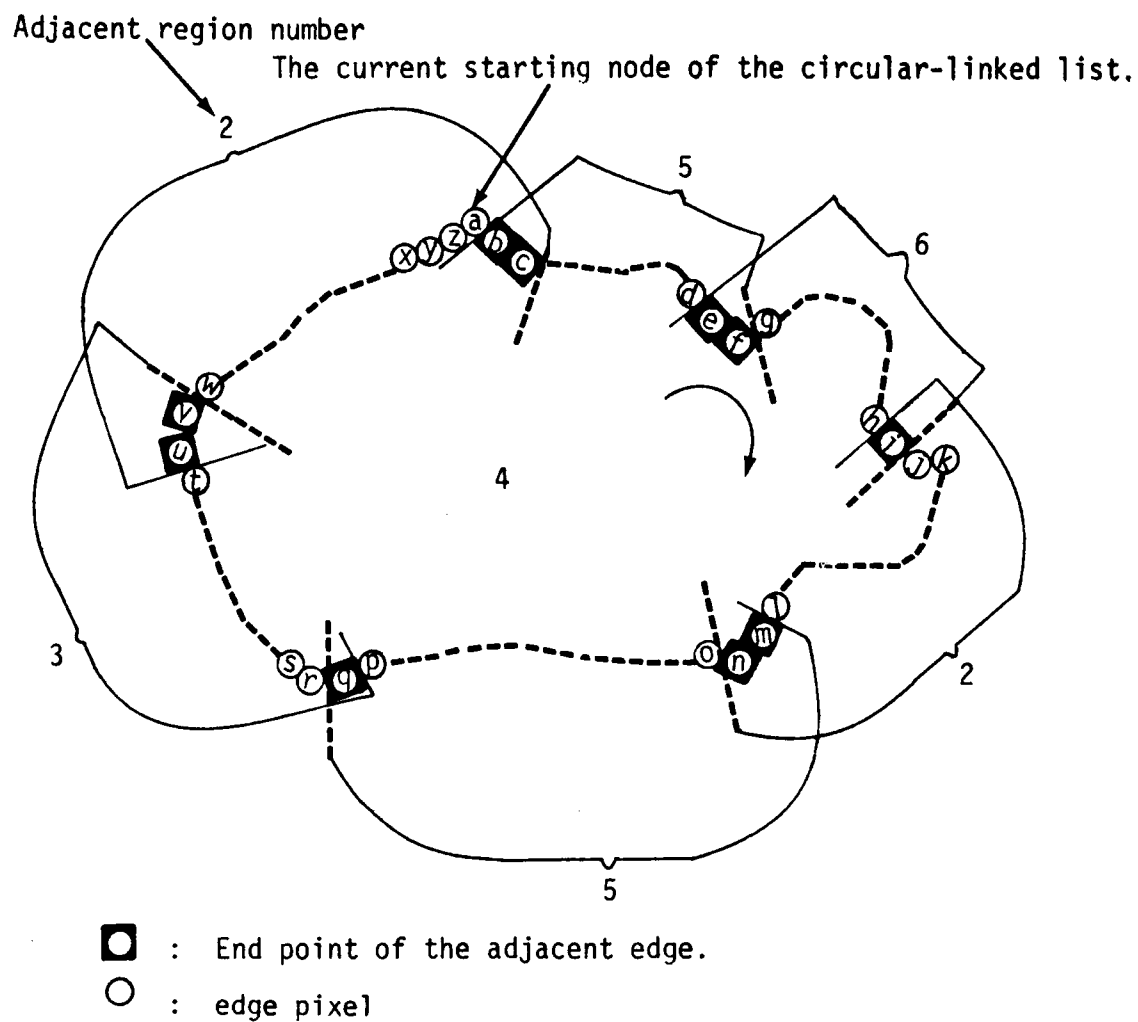


Figure 10. Six adjacent edges of region 4.

end point of an adjacent edge has been found, the sequencing of that adjacent edge is straightforward.

For example, in Figure 10, supposing that adjacent edge 2 (i.e., the adjacent edge between region 4 and its adjacent region 2) is being processed and that the starting end point pixel *i* has been found, then from pixel *i* in the corresponding circular-linked list the sequenced boundary pixels *j*, *k*, ---, *l*, *m*, and *n*, which also belong to adjacent edge 2, can be extracted. Thus, one separable part of adjacent edge 2 has been sequenced. However, there is probably another separable part of adjacent edge 2. From pixel *o* in the corresponding circular-linked list, the remaining nodes in the list must be traced until pixel *h* is traced. In this example, pixel *u* will be found to be the starting end point of another separable part of adjacent edge 2.

By the same method, from pixel *u* the sequenced boundary pixels *v*, *w*, ---, *x*, *y*, *z*, *a*, *b*, and *c* can be extracted from the same circular-linked list. Thus, another separable part of adjacent edge 2 has also been sequenced.

After all separable parts of adjacent edge 2 have been sequenced, whether there is another region adjacent to region 4 can be determined by reference to the adjacency table. If no more regions are adjacent to region 4, the next region, i.e., region 5, will be processed. Once a separable part of an adjacent edge has been sequenced, it is stored in a sequential list in the third level of the output data base. Then a node storing its corresponding adjacent region number will be put into the adjacency table in the second level of the data base.

To find the first starting end point of an adjacent edge (i.e., the starting end point of the first separable part of an adjacent edge), two variables, IFLAG and JOPTLN, are first declared. (See program SEQEDG.FOR shown in the Appendix.) IFLAG acts as a flag in the algorithm. JOPTLN is the one-dimensional location of the boundary pixel which is the candidate of the starting end point of an adjacent edge. IFLAG and JOPTLN must first be reinitialized to zero whenever the starting end point of an adjacent edge is found. The candidates for the starting end point of an adjacent edge are the same as those edge pixels stored in the corresponding sequential list in the third level of the boundary adjacency data base.

Once it is determined that a node is not a candidate, IFLAG is set to "1." If a candidate is not elected, IFLAG must be set back to "0." The necessary conditions for the candidate to be elected as the starting end point of an adjacent edge are that IFLAG is "1" and that JOPTLN is "0."

After all the nodes in the circular-linked list are traced, if no candidate satisfies these necessary conditions, the current starting node of the list is shifted to the last non-candidate node in this tracing operation. Then, from this new starting node, the procedures are repeated to select the candidates for the starting end point of the corresponding adjacent edge. If all the nodes in the circular-linked list are candidates, i.e., the whole boundary of the corresponding region is itself the adjacent edge, then the current starting node is regarded as the starting end point of this adjacent edge.

For example, in Figure 10 adjacent edge 2 is being processed; the current starting node of the corresponding circular-linked list is pixel a.

IFLAG and JOPTLN have been initialized to be "0." The procedures finding the starting end point of the adjacent edge 2 are begun at pixel a. Since pixel a is a candidate, it is checked for the necessary conditions mentioned before. Here pixel a cannot be elected because IFLAG is still "0." Since pixel a is a candidate, JOPTLN is changed to the one-dimensional location of pixel a in the picture. Now that JOPTLN is not "0," even if IFLAG is "1," there will be no candidates that satisfy the conditions for the starting end point of the adjacent edge 2 when all the nodes in the circular-linked list are traced. Therefore, the current starting node of the circular-linked list is changed to the last encountered non-candidate pixel t. Then IFLAG and JOPTLN are reinitialized to "0."

The above procedures are repeated at the new starting node, pixel t. Then the candidate, pixel u, satisfies the conditions. It is the first starting end point of the adjacent edge 2.

The step descriptions of this adjacent edge sequencing algorithm are shown below, and its brief flowchart is shown in Figure 11.

STEP 1: Input the sequenced boundary pixel data base. (The circular-linked list in this data base is List A.)

STEP 2: Input boundary adjacency data base. (The sequential list in this data base is List B.)

STEP 3: If all regions have been processed, output the sequenced boundary adjacency data base; otherwise, go to STEP 4 to process the next region.

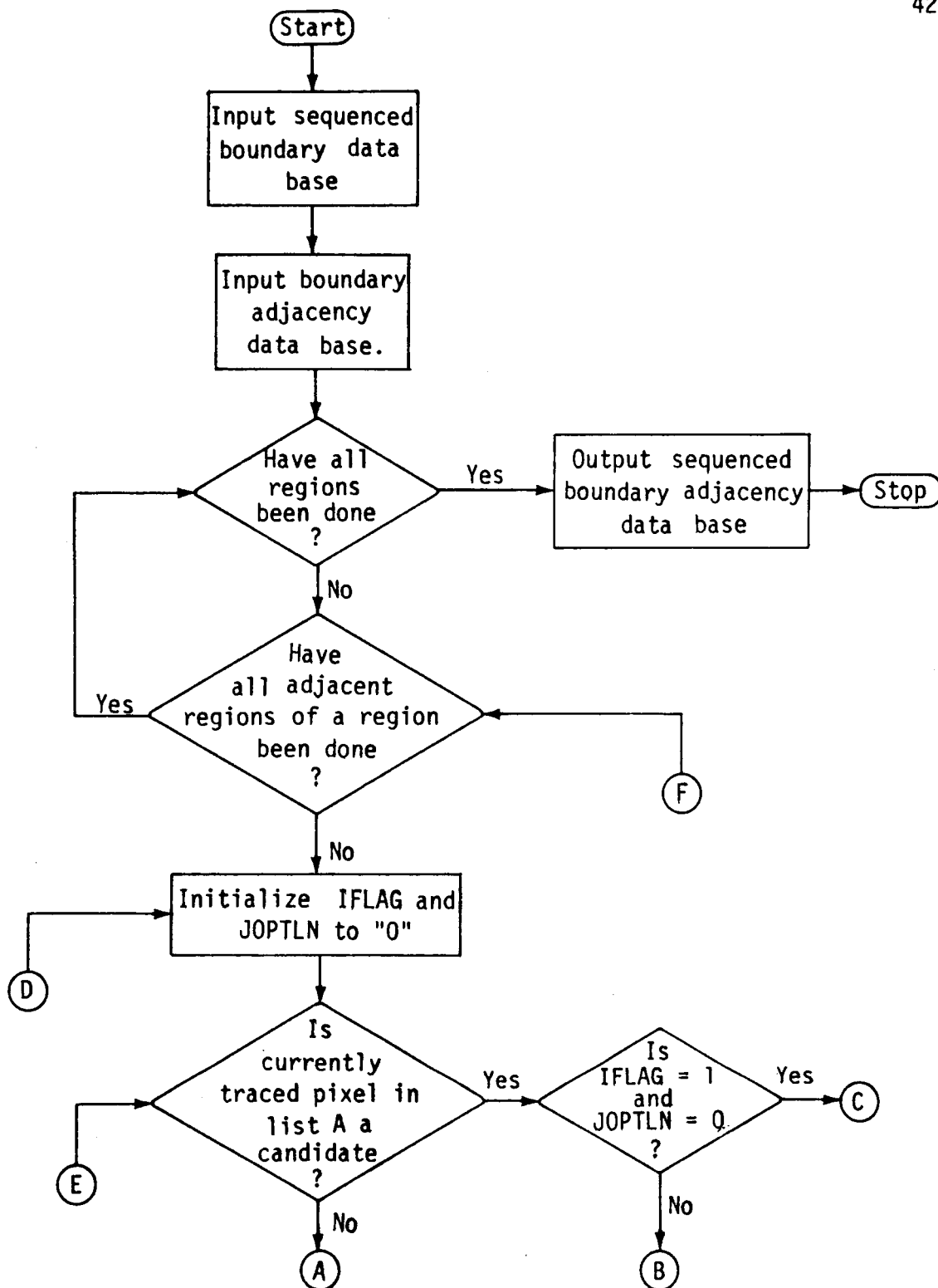


Figure 11. Flowchart for adjacent edge sequencing algorithm.

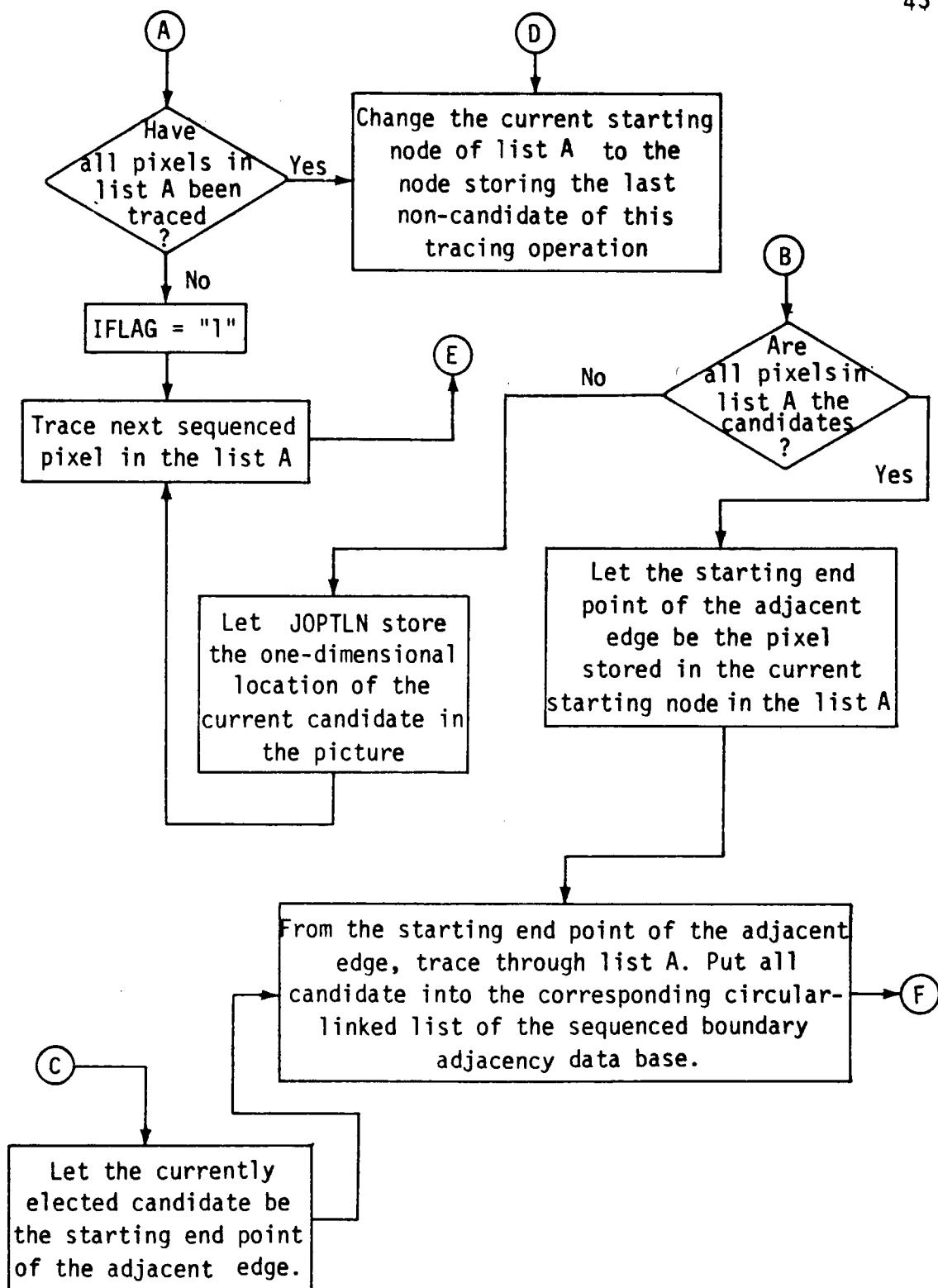


Figure 11 (Continued).

STEP 4: If all adjacent edges of an adjacent region of the currently processed region have been sequenced, go back to STEP 3; otherwise, go to STEP 5 to sequence the next adjacent region's adjacent edges.

STEP 5: Initialize IFLAG and JOPTLN to "0."

STEP 6: If the currently scanned pixel in List A is a candidate (i.e., the same pixel is in List B), go to STEP 8; otherwise, go to STEP 7.

STEP 7: If all pixels in List A have been traced, to go STEP 9; otherwise, go to STEP 13.

STEP 8: Call this candidate pixel *s*. If *s* satisfies the conditions $IFLAG = 1$ and $JOPTLN = 0$, go to STEP 15; otherwise, go to STEP 10.

STEP 9: Change the current starting node of List A to the node storing the last non-candidate pixel of this tracing operation. Then go back to STEP 5.

STEP 10: If all pixels in List A are candidates, go to STEP 12; otherwise, go to STEP 11.

STEP 11: Let JOPTLN store the one-dimensional location of the pixel *s* in the picture. Then go to STEP 14.

STEP 12: Let the starting end point of the adjacent edge be the pixel stored in the current starting node in List A. Then go to STEP 16.

STEP 13: Let $IFLAG = 1$. Go to STEP 14.

STEP 14: Trace the next sequenced pixel in List A. Then go back to STEP 6.

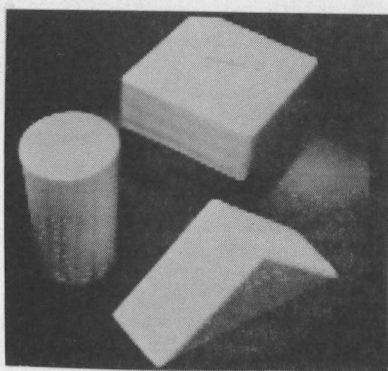
STEP 15: Let pixel *s* be the starting end point of the adjacent edge. Then go to STEP 16.

STEP 16: From the starting end point of the adjacent edge, trace through List A. Put all the candidates into the corresponding sequential list of the sequenced boundary adjacency data base in the same order as they appeared in List A. Then go back to STEP 4.

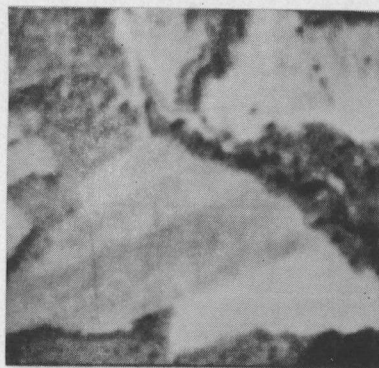
Four pictures in Figure 12 illustrate the above algorithm. The first picture is a simple scene consisting of a block, a pyramid, and a cylinder. The second picture is an aerial photograph of AVA, provided by SAMSO (Space and Missile System Organization). The third picture is a Hughes Building from DARPA (Defense Advanced Research Project Agency). The fourth picture is the Kodak Girl portrait.

Each of these four pictures contains 128×128 pixels, each having one of 256 grey level values. They are segmented into several labeled regions; then the grey level values of the pixels of each labeled region are averaged, as shown in Figure 13.

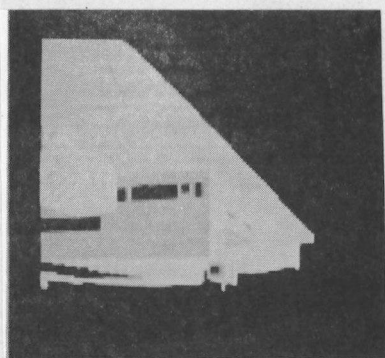
The boundaries of each segmented image extracted from the corresponding sequenced boundary pixel data base are shown in Figure 14. The four pictures in Figure 15 display the adjacent edges extracted from the corresponding sequenced boundary adjacency data base. The edges adjacent to the frames of the pictures are not shown, because this redundant information is not stored in the data base.



(a)



(b)

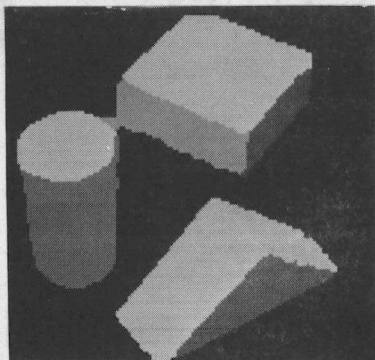


(c)

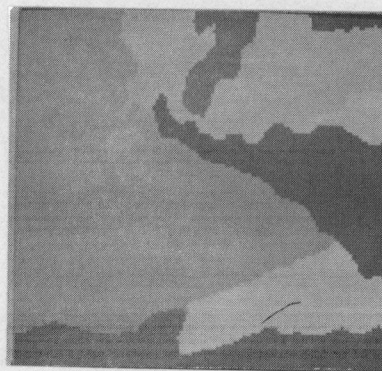


(d)

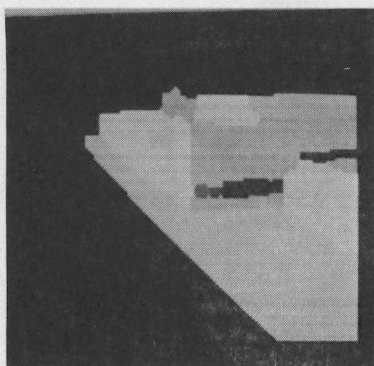
Figure 12. Original image picture. (a) Simple example scene, (b) AVA aerial photograph, (c) Hughes building, and (d) Kodak girl.



(a)



(b)

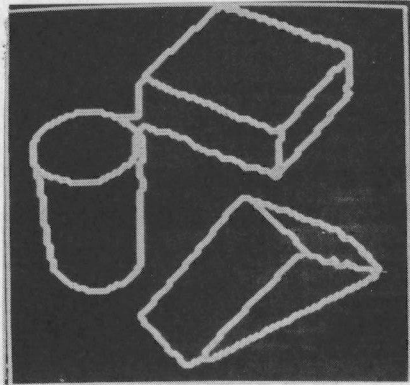


(c)

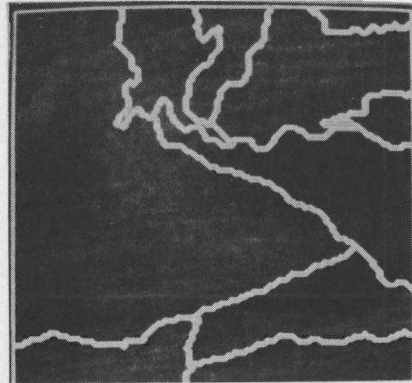


(d)

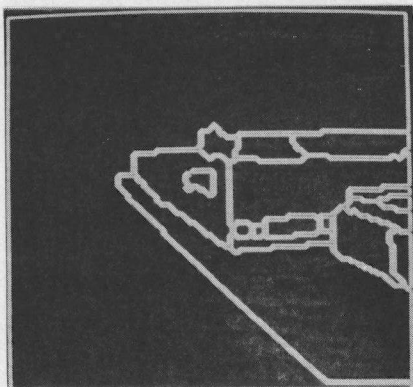
Figure 13. Segmented and averaged image picture. (a) Simple example scene, (b) AVA, (c) Hughes building, and (d) Kodak girl.



(a)



(b)

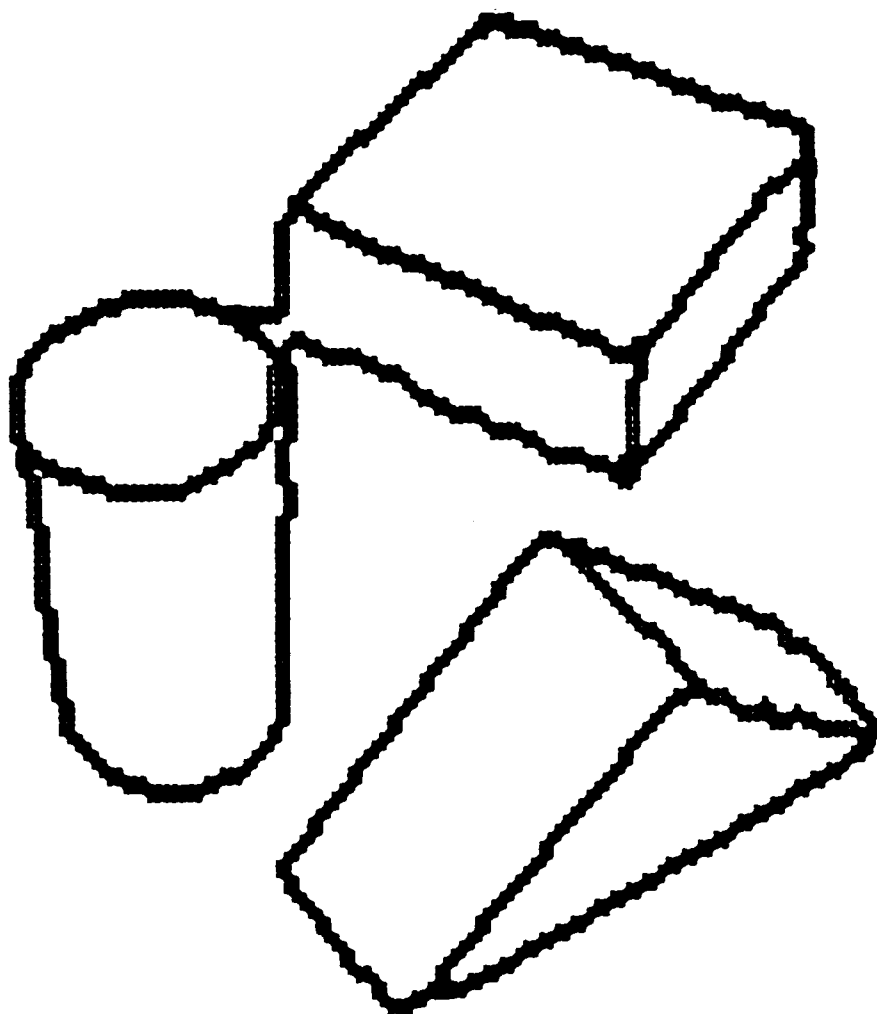


(c)



(d)

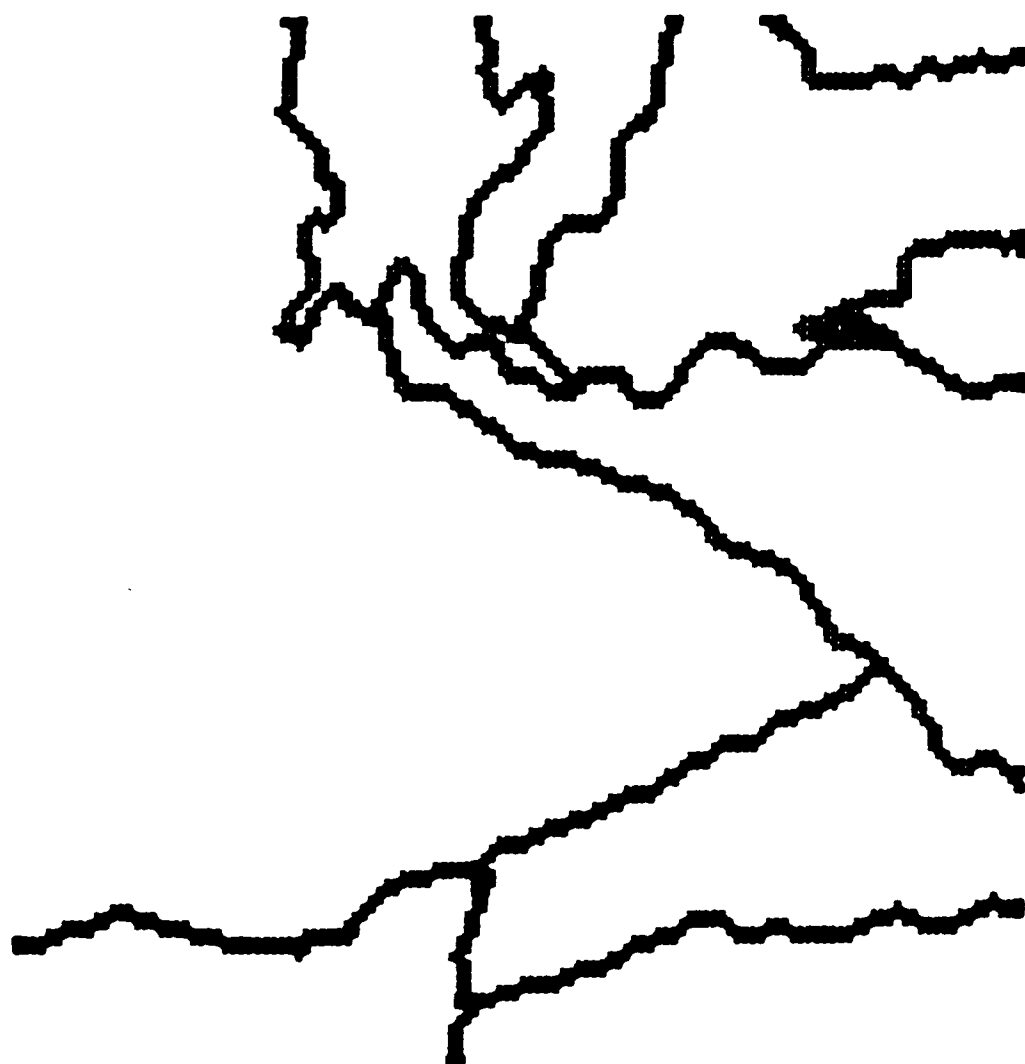
Figure 14. Boundaries generated from sequenced boundary pixel data base. (a) Simple example scene, (b) AVA, (c) Hughes building, and (d) Kodak girl.



(a) Simple example scene

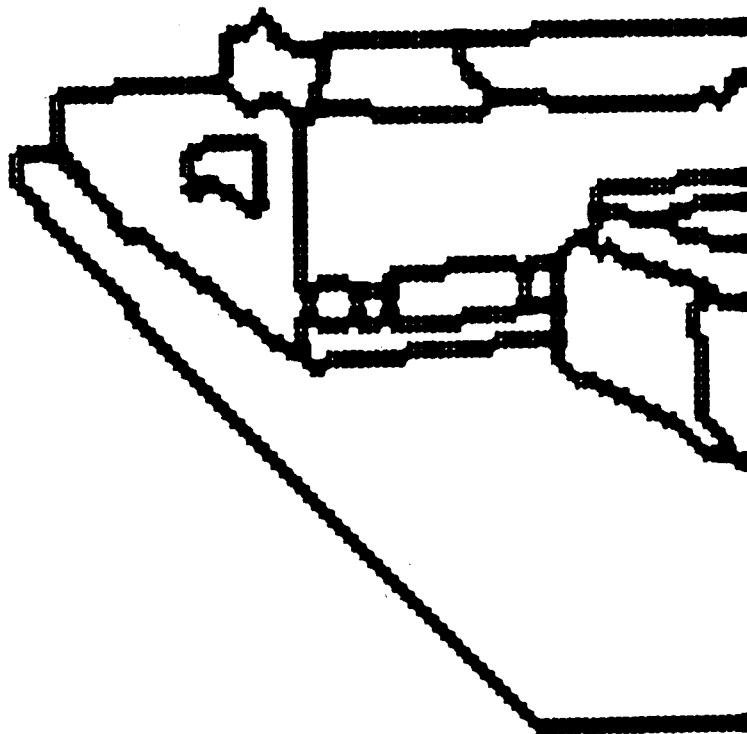
■ : End point of the adjacent edge.

Figure 15. Adjacent edges generated from sequenced boundary adjacency data base.



(b) AVA

Figure 15 (continued)



(c) Hughes building

Figure 15 (continued)



(d) Kodak girl

Figure 15 (continued)

VII. CODING SCHEME FOR BOUNDARY ADJACENCY DATA BASE COMPRESSION

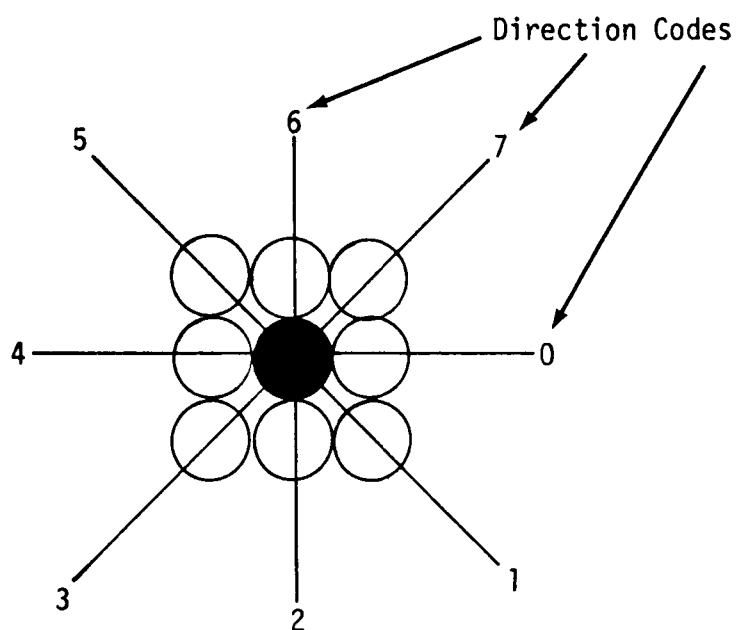
In this section, three simple coding schemes based on a direction code are presented for the compression of the sequenced boundary adjacency data base. The storage data structure of the compressed data base is a three-level hierarchical structure very similar to that of the boundary adjacency data base in Figure 8, page 31.

The first and second level storage structures of the compressed data base are also used to store the region pointers and the adjacency table respectively. The third level storage structure of the compressed data base is also in sequential list representation; however, its elements are one-LOGICAL*1 bytes, whereas those of the boundary adjacency data base are two-LOGICAL*1 bytes.

Since it is necessary to know the location of the starting end poing of an adjacent edge in order to identify the adjacent edges, its x and y coordinates are stored in the first and second elements of the corresponding sequential list in the third level of the compressed data base. Then the coding numbers which represent the compressed data are stored in the remaining elements of the corresponding sequential list.

The following three data base compression schemes basically use the direction codes to represent the direction of sequencing the edge pixels. For each pixel, eight numbers can be assigned to correspond to its eight eight-connective adjacent pixels. Figure 16 shows the eight direction codes used in the following coding schemes.

To determine the direction code, let JOPTLN be the one-dimensional location of the reference pixel in the 128*128 image and LOCATN be that



: Reference pixel



: Eight-connective adjacent pixel of the reference pixel

Figure 16. Direction codes for compression coding schemes.

of the eight-connective adjacent pixel of the reference pixel. Let
 $DIFR = LOCATN = JOPTLN$. Then,

$DIFR = 1$ corresponds to Direction Code = 0;

$DIFR = 129$ corresponds to Direction Code = 1;

$DIFR = 128$ corresponds to Direction Code = 2;

$DIFR = 127$ corresponds to Direction Code = 3;

$DIFR = -1$ corresponds to Direction Code = 4;

$DIFR = -129$ corresponds to Direction Code = 5;

$DIFR = -128$ corresponds to Direction Code = 6;

$DIFR = -127$ corresponds to Direction Code = 7.

In the following coding algorithms, once the x and the y coordinate locations of the starting end point of an adjacent edge are stored respectively in the first and the second bytes of the corresponding sequential list (called the "Code List") in the third level of the compressed data base, this starting end point is regarded as the reference pixel of its following sequenced adjacent edge pixel kept in the sequential list of the third level of the sequenced boundary adjacency data base. Then, as soon as a pixel has been processed for coding, it is regarded as the reference pixel of its following sequenced adjacent edge pixel. Therefore, every edge pixel except the starting end point of the adjacent edge has a corresponding direction code ranging from 0 to 7.

The first coding scheme sequentially combines two successive direction codes into one code by the following coding equation:

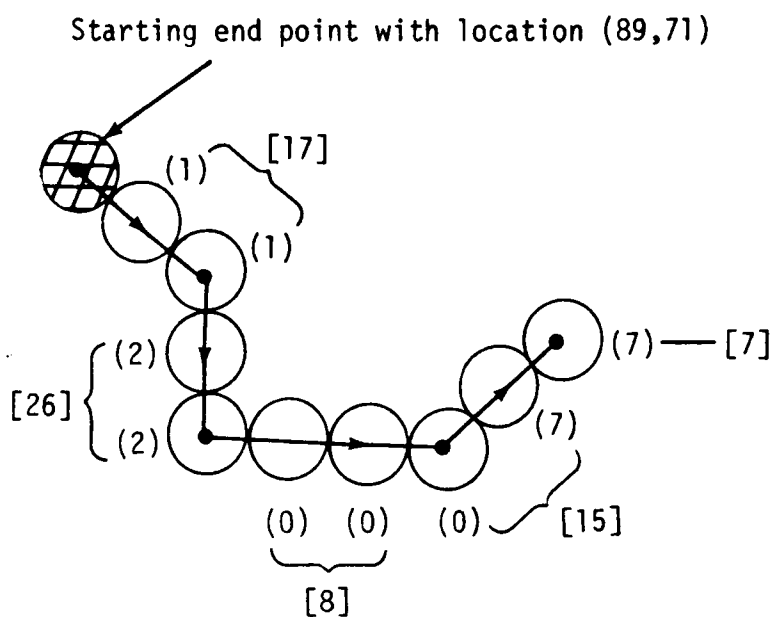
$$CODE = (DIRCD1 + 1) * 8 + DIRCD2, \quad (7-1)$$

where DIRCD1 represents the first one of the two successive direction codes, and DIRCD2 is the second. CODE is the number which will be stored in a LOGICAL*1 byte of the corresponding Code List.

If the total number of the direction codes of an adjacent edge is an odd number, the last direction code of this adjacent edge cannot be combined into one code by using Equation (7-1). In this case, CODE is set equal to this last direction code. Since CODE is always positive, "-128" is used to represent the end mark of the Code List. For example, in Figure 17 the numbers in parentheses represent the direction code, and the numbers in brackets represent the codes stored in the Code List. Thus, the Code List for the adjacent edge in Figure 17 will be "89, 71, 17, 26, 8, 15, 7, -128," where 89 and 71 are the x and y coordinates of the starting end point.

The second coding scheme is basically the coding of the direction vectors. A direction vector is a section of an adjacent edge in which all edge pixels have the same direction codes. The number of the pixels of a direction vector is called the "length" of this vector. For example, the first direction vector of the adjacent edge shown in Figure 18 has length "6" and direction code "2."

Since the element of the Code List is a LOGICAL*1 byte (eight bits), the number represented by a byte for the code can only range from -128 to 127. In order to code the direction vector as long as possible, the eight possible kinds of direction vectors are divided into two parts. The first part, consisting of direction codes 0, 1, 2, and 3, can range from 1 to 127; the second part, consisting of direction codes 4, 5, 6, and 7, can range from -1 to -128. The number "0" is used to represent



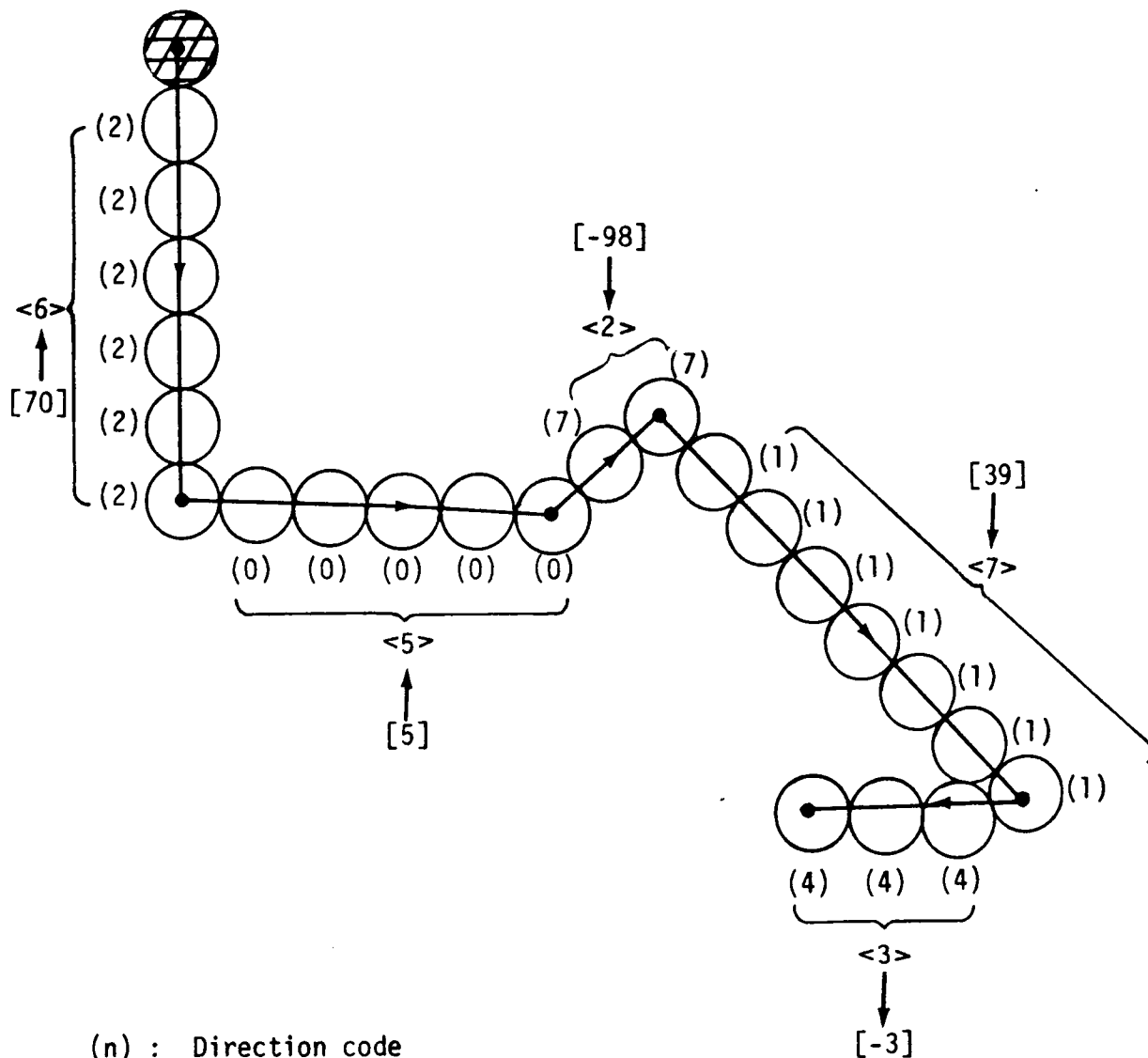
(n) : Direction code

[m] : Code stored in the Code List.

Code List : "89,71,17,26,8,15,7,-128"

Figure 17. Example for the first coding scheme.

Starting end point with location (23,45)



(n) : Direction code

<m> : The length of the direction vector

[1] : Code stored in the Code List

Code List : "23,45,70,5,-98,39,-3,0"

Figure 18. Example for the second coding scheme.

the end mark of the Code List. The maximum length of the direction vector is "31" because four different kinds of direction vectors with different lengths ranging from 1 up to 31 can be represented by the numbers from 1 to 124 (-1 to -124) without exceeding the range from 1 to 127 (-1 to -127). Therefore, if a direction vector's length exceeds 31, it is divided into two direction vectors. For example, if there is a direction vector with length 70, it will be sequentially divided into three direction vectors with length 31, 31, and 8. If the maximum length of the direction vector is allowed to be 32, direction vectors in the ranges from 0 to 127 and -1 to -128 may be coded; however, a problem might arise, in that no more numbers can be used to represent the end marks of the Code Lists. This illustrates the reason why the maximum length of the direction vector for the second coding scheme can only be as large as 31.

The second coding scheme codes the direction vectors of the adjacent edge by the following coding equations:

$$\begin{aligned} \text{CODE} &= (\text{DIRCOD} * 32) + \text{LENGTH} \\ &\text{for } 0 \leq \text{DIRCOD} \leq 3; \end{aligned} \tag{7-2}$$

and

$$\begin{aligned} \text{CODE} &= -\{[(\text{DIRCOD} - 4) * 32] + \text{LENGTH}\} \\ &\text{for } 4 \leq \text{DIRCOD} \leq 7; \end{aligned} \tag{7-3}$$

where DIRCOD represents the direction code of the direction vector, LENGTH is the length of the direction vector, and the CODE stands for the number which will be stored in a LOGICAL*1 byte of the corresponding Code List.

For example, in Figure 18 the numbers in parentheses are the direction codes of the corresponding direction vector, and the numbers in "< >'s" represent the length of the corresponding direction vector. Their codes are shown in brackets. Thus, the Code List representing the adjacent edge shown in Figure 18 will be "23, 45, 70, 5, -98, 39, -3, and 0," where 23 and 45 are the x and y coordinates of the starting end point.

The third coding scheme is also basically the coding of the direction vectors but is developed from the second coding scheme by introducing more direction codes and reducing the maximum length of the direction vector. If the direction vector has length "1" (called "direction vector A"), it is combined with its next direction vector (called "direction vector B") into a "super-direction vector" with the length the same as that of its next direction vector. Thus, the direction code of the super-direction vector can be regarded as a number corresponding to the combination (DRCOD1,DRCOD2), where DRCOD1 is the direction code of direction vector A, and DRCOD2 is the direction code of the direction vector B.

Since DRCOD1 and DRCOD2 have eight different codes respectively, their combination (DRCOD1, DRCOD2) will have 64 (8*8) different representations. Each representation corresponds to a different code. Thus, there must be 64 different numbers for coding these combinations. However, some of these 64 combinations are redundant.

For example, consider the combination of (0,4) for (DRCOD1,DRCOD2). Direction 0 and 4 are opposite, but two adjacent edge pixels cannot be sequenced in mutually opposite directions by the previously described

boundary pixel sequencing algorithm. Therefore, such codes are not implemented.

Furthermore, there are combinations that very rarely occur in practical processing (e.g., the combinations (0,3), (2,5), (6,3), etc.). These combinations are associated with the single singular boundary pixels in some section of an adjacent edge. The occurrences of the single singular boundary pixels are very rare. Therefore, super-direction codes are not assigned to these combinations either. If such a combination does unfortunately occur, it will be coded by the method shown in the second coding scheme, since there is not super-direction code assigned to it.

Figure 19 shows all the super-direction codes implemented in the third coding scheme. Since the original-sense direction code is a number ranging from 0 to 7, the super-direction code begins at the number "8." In Figure 19, the numbers in parentheses represent the original-sense direction code, and the numbers in brackets stand for the corresponding super-direction codes. The first arrow of each super-direction vector represents direction vector A which is of length "1," and the second arrow represents direction vector B. The direction of the arrows shows the order of the coding operation.

To avoid ambiguity, the term "advanced direction vector" is used to imply the collective representation of the original-sense direction vector considered in the second coding scheme and the previously introduced super-direction vector. Thus, the advanced direction code also implies the original-sense direction code and the super-direction code.

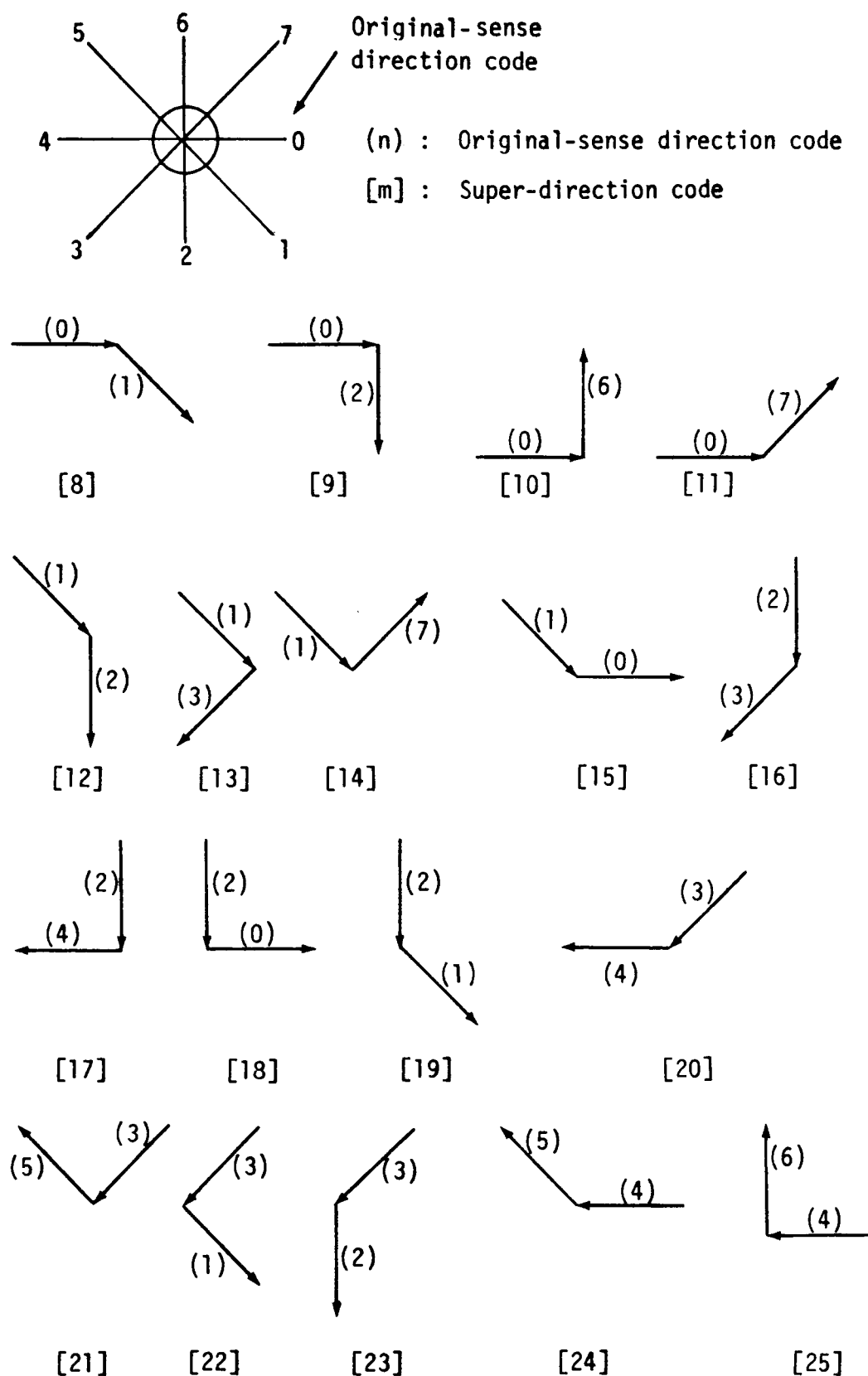


Figure 19. Super-direction codes for the third coding scheme.

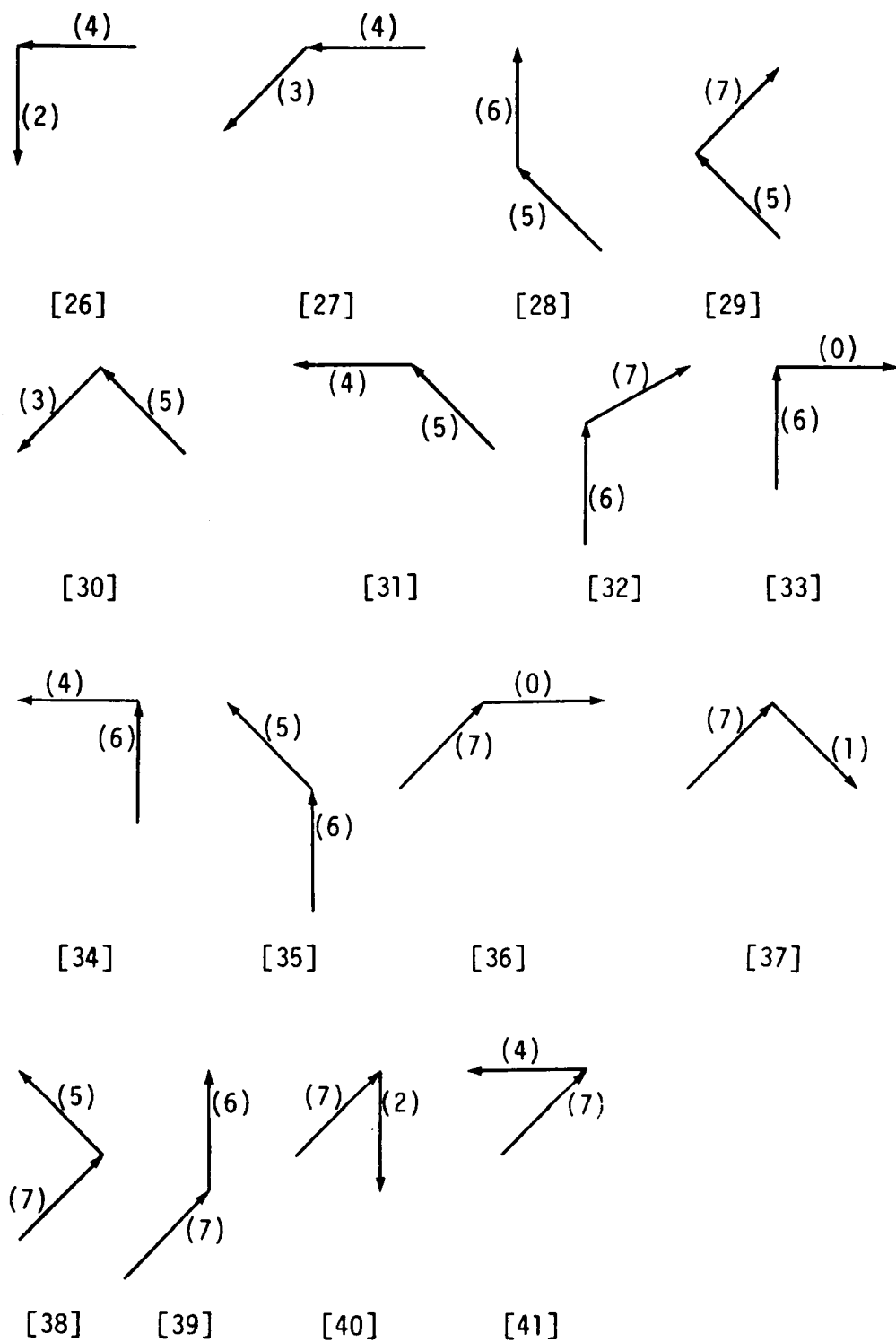


Figure 19 (Continued).

Since the total number of the advanced direction codes is 42, the maximum length of the direction vector that can be coded in this coding scheme is 6.

In the third coding scheme, the advanced direction vectors with advanced direction codes ranging from 0 to 20 are coded by a number ranging from 0 to 125, and those with advanced direction codes ranging from 21 to 41 are coded by a number ranging from -1 to -126. The number "-128" is used to indicate the end mark of the Code List.

Since in this coding scheme the maximum length of the advanced direction vector can only be as large as 6, those advanced direction vectors with lengths longer than the maximum length 6 are divided into two advanced direction vectors. The first one has length 6. If the length of the second one happens to be 1, then the second one is combined into another super-direction vector. If the length of the second one is still greater than 6, the second one is divided again in the same way.

The third coding scheme codes the advanced direction vectors of the adjacent edge by the following coding equations:

$$\begin{aligned} \text{CODE} &= (\text{DIRCOD} * 6) + \text{LENGTH} - 1 \\ &\text{for } 0 \leq \text{DIRCOD} \leq 20; \end{aligned} \tag{7-4}$$

and

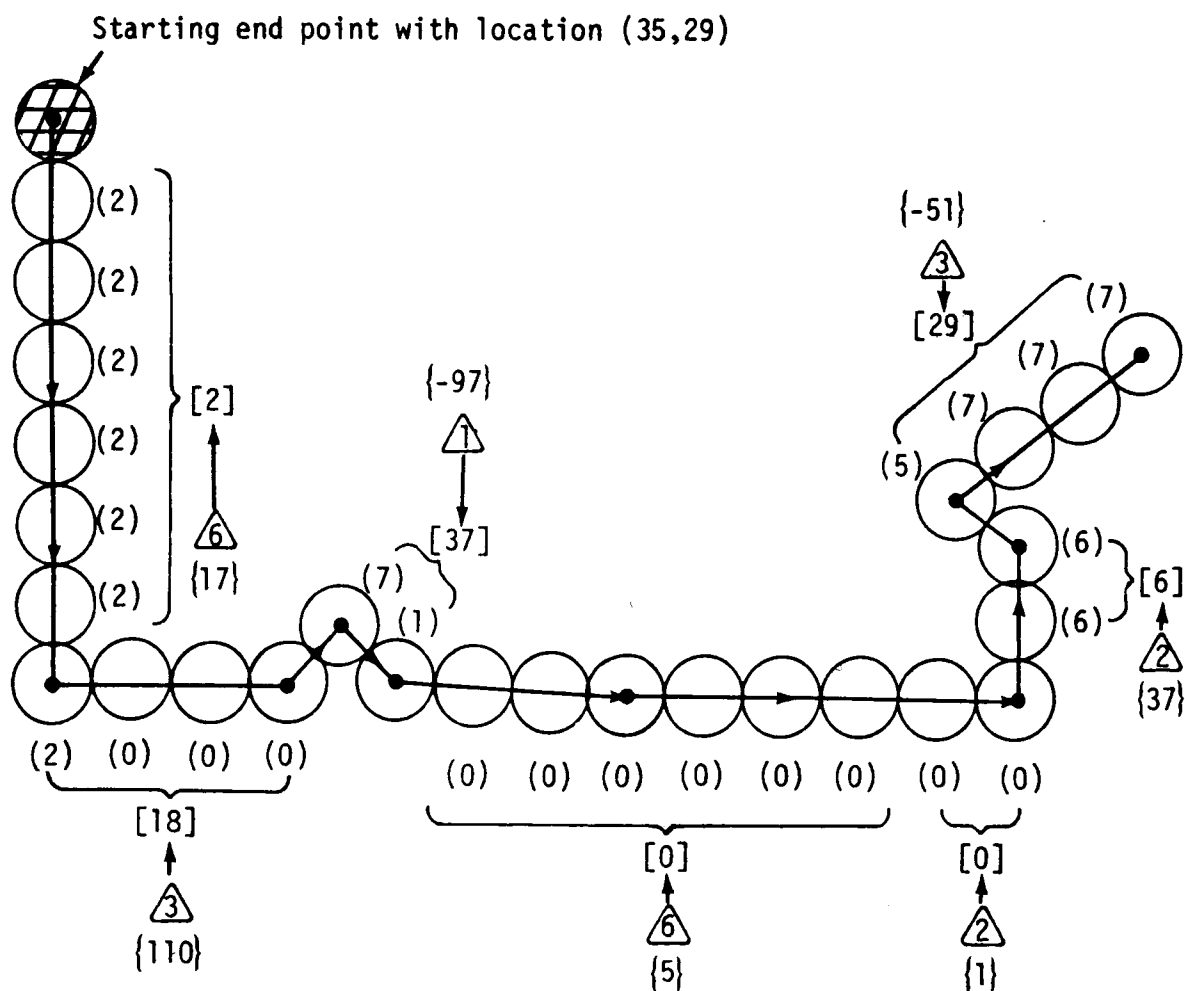
$$\begin{aligned} \text{CODE} &= -\{[(\text{DIRCOD} - 21) * 6] + \text{LENGTH}\} \\ &\text{for } 21 \leq \text{DIRCOD} \leq 41; \end{aligned} \tag{7-5}$$

where DIRCOD represents the advanced direction code of the advanced direction vector, LENGTH stands for the length of the advanced direction

vector, and the CODE is the number stored in a LOGICAL*1 byte of the corresponding Code List.

For example, in Figure 20 the numbers in parentheses represent the original-sense direction codes. The numbers in brackets stand for the advanced direction codes. The numbers in the triangles are the lengths of the advanced directions vectors, and the number in the "{ }'s" is the code stored in the Code List. Thus, the Code List corresponding to the adjacent edge shown in Figure 20 will be "35, 29, 17, 110, -97, 5, 1, 37, -51, and -128," where 35 and 29 are the x and y coordinates of the starting end point, and -128 is the end mark of the Code List.

From the example shown in Figure 20, the coding of the super-direction vector with super-direction code "-37" by the third coding scheme is found to have the same compression effect as the first coding scheme; i.e., two sequenced adjacent edge pixels can be represented by a LOGICAL*1 byte code. However, in any other advanced direction vectors shown in this example, the compression effect of the third coding scheme is similar to that of the second coding scheme. The quantitative comparison of these three coding schemes for a variety of examples will be discussed in the next section.



(n) : Original-sense direction code

[m] : Advanced direction code

$\triangle$: The length of the advanced direction vector

{p} : Code stored in the Code List.

Code List : "35,29,17,110,-97,5,1,37,-51,-128"

Figure 20. Example for the third coding scheme.

VIII. EXPERIMENTAL RESULTS AND DISCUSSION

Table I shows some experimental results of the algorithms described in the previous sections. Four pictures in Figure 12, page 46, are taken as the examples. These four pictures are segmented into four region-labeled pictures in Figure 13, page 47, by using the GLECS (Global Local Edge Coincidence Segmentation) technique [12,13]. The boundary pixel sequencing and boundary adjacency tracing algorithms are then applied to these region-labeled pictures to set up the sequenced boundary pixel data bases and the boundary adjacency data bases respectively. The adjacent edge sequencing algorithm is next applied to these data bases to establish the sequenced boundary adjacency data base, which is finally compressed by the three coding schemes.

Since the numbers of elements stored in the first and second levels of the compressed data base do not differ from those of the sequenced boundary adjacency data base, only the third level of the data base is presented for comparing the effect of the compression.

The comparison is classified into two categories, "Whole List" and "Real Body" in Table I. The Whole List category is a comparison involving the whole sequential list in the third level of the data base, and the Real Body category corresponds to the comparison involving only the elements making a contribution to the coding of the direction vectors. The comparison of the Real Body category does not include the elements for the end marks of the sequential lists and the elements representing the starting end points of the adjacent edges.

Table I. Experimental results of four example pictures by using coding schemes.

Sequential list in the third level of the corresponding data base Example pictures	Memory length for sequenced adjacent edges		First coding scheme				Second coding scheme				Third coding scheme				Number of adjacent edges
	Whole list*	Real body**	Whole list		Real body		Whole list		Real body		Whole list		Real body		
			Memo leng byte	Comp ratio	Memo leng byte	Comp ratio	Memo leng byte	Comp ratio	Memo leng byte	Comp ratio	Memo leng byte	Comp ratio	Memo leng byte	Comp ratio	
Simple example scene	1478 [2956]	1392 [2574]	836	0.282	707	0.254	859	0.291	730	0.262	586	0.198	457	0.164	43
AVA aerial photograph	1363 [2726]	1287 [2574]	767	0.281	653	0.253	830	0.304	716	0.278	574	0.211	460	0.178	38
Hughes building	1363 [2726]	1199 [2398]	859	0.315	613	0.256	641	0.235	395	0.165	573	0.210	327	0.136	82
Kodak girl	2792 [5584]	2476 [4952]	1748	0.313	1274	0.257	1992	0.357	1518	0.306	1444	0.259	970	0.196	158

*This item shows the length of memory storing all sequential lists in the 3rd level of the corresponding data base.

**This item shows the length of memory for those nodes having contribution to the coding of direction vector. It doesn't contain the memory for the nodes representing the end marks of the sequential lists nor the nodes representing the starting end points of the adjacent edges.

memo = memory leng = length

The compression ratio shows the compression effect of the coding schemes. A smaller compression ratio indicates greater compression effect.

For the first coding scheme, two edge pixels can be represented by a LOGICAL*1 byte code; however, in the sequenced boundary adjacency data base, one edge pixel is stored in an INTEGER*2 word (two bytes). Therefore, for the Real Body category, the compression ratio of the compressed data base to the sequenced boundary adjacency data base ideally must be equal to 0.25 (1:4). Nevertheless, if the number of the edge pixels (except the starting end point) of an adjacent edge is not an even number, one more byte is required to code the last edge pixel of this adjacent edge. For this reason, the compression ratios of the Real Body category for the first coding scheme shown in Table I are slightly more than 0.25. Furthermore, the more adjacent edges in the picture, the greater is the difference of compression ratio between experimental cases and ideal cases due to the presence of adjacency edges in the picture. Table I shows these characteristics of the first coding scheme.

For the second coding scheme, the compression ratio depends on the feature of the adjacent edge in the picture. If the adjacent edges in the picture are composed of relative long direction vectors, this picture is said to be "more regular." The compression ratio for more regular pictures will be smaller because for a larger direction vector more edge pixels are coded in a byte. In Figure 15, one sees that the Hughes Building (Kodak Girl) is the most (least) regular picture among the four pictures. The compression ratio for the Hughes Building will be the smallest, and that for the Kodak Girl will be the largest. Table I shows this characteristic of the second coding scheme.

For the ideal case, since the maximum length of the direction vector used in the second scheme is 31 (i.e., 31 edge pixels can be coded in a LOGICAL*1 byte), the compression ratio of the Real Body category can be as small as $1/62$. For the worst case, all the direction vectors of all adjacent edges in the picture have lengths of only 1, and the compression ratio of the Real Body category for the second coding scheme is as large as $1/2$.

For the third coding scheme, the super-direction vector combines the characteristics of the first and second coding schemes. For the same worst case as described above, since the length of every direction vector is 1, every two successive direction vectors can be combined into a super-direction vector. That is the same sense as two edge pixels being coded by a byte. Therefore, in this case the compression effect of the third coding scheme is the same as that of the first coding scheme. In the ideal case, since the maximum of the advanced direction vector is 6, by the same augmentation for the second coding scheme the compression ratio of this coding scheme thus can only be as small as $1/12$.

In the compressed data base, each starting end point of the adjacent edge is stored in two LOGICAL*1 bytes (x and y coordinate locations respectively), and each end mark of the sequential list is stored in a LOGICAL*1 byte; however, in the sequenced boundary adjacency data base, they are respectively stored in an INTEGER*2 word. Therefore, considering that the memory stores only the starting end points and the end marks, a one-byte gain of compression for an adjacent edge is achieved. (Four bytes are compressed to three bytes.) Compared with the previously discussed compression effects of those coding schemes (in the Real Body

category), this gain is much less than any of them, because the worst of them has at least one-byte gain of compression from an edge pixel. (The worst case among them is that every pixel is coded by a byte; however, in the sequenced boundary adjacency data base, each pixel is originally stored in a word.) This kind of compression gain will serve as a "counter-effect" to the compression effect considered in the Real Body category when it is considered in the Real Body Category. (Here the term "counter-effect" will be used to stand for such a compression gain.)

Actually, the compression effect considered in the Whole List category is the combination of this counter-effect and the Real Body category's compression effect. Therefore, the consideration of the counter-effect illustrates the reason why the compression effect of the Whole List category is always slightly worse than that of the Real Body category for any of the three coding schemes. Furthermore, this counter-effect will increase as the number of the adjacent edges of a picture increases; however, its increment will be gradually getting smaller when the number of the adjacent edges in a picture is getting larger. The proof of this property is described as follows:

It is assumed that the compression effect of one of the three coding schemes for some picture is being compared in the Whole List and the Real Body categories. The compression ratio of the Real Body category is originally known to be $1/r$, i.e.,

$$1/r = \frac{x \text{ (bytes)}}{a \text{ (bytes)}}, \quad (8-1)$$

where both x and a are the lengths of the memories making a contribution to the compression ratio of the Real Body category. They belong respectively to the compressed data base and to the sequenced boundary adjacency data base. Equation (8-1) implies that

$$a = rx \text{ (bytes).} \quad (8-2)$$

It is assumed that the number of the adjacent edges in this picture is known as y . Since the memories considered in the Real Body category do not contain the starting end points of the adjacent edges, and since each adjacent edge owns a starting end point, the number, p , of the pixels stored in the sequenced boundary adjacency data base is

$$p = a/2 + y = rx/2 + y \text{ (words).} \quad (8-3)$$

(Note: These edge pixels and the end marks of the sequential list are represented by words in the sequenced boundary adjacency data base.)

Also, the compression ratio, $1/R1$, of the Whole List category is originally

$$1/R1 = \frac{x + 2y + 1y}{a + 2y + 2y} = \frac{x + 3y \text{ (bytes)}}{rx + 4y \text{ (bytes)}}, \quad (8-4)$$

where $1y$ is the memory length for the end marks of the lists in the compressed data base. One of the two $2y$'s in the denominator of Equation (8-4) stands for the memory length of the end marks of the lists in the sequenced boundary adjacency data base. The other two $2y$'s represent the memory lengths for the starting end points of the adjacent edges of the corresponding data bases.

If the number of the adjacent edges of the original picture is increased by an increment z (the configuration of the adjacent edges is assumed not to be changed), the number, p' , of the pixels stored in the sequenced boundary adjacency data base will be increased as

$$p' = p + 2z \text{ (words)}, \quad (8-5)$$

because the pixels representing the two end points of the adjacent edge appear doubly in the data base.

It is also assumed that the compression ratio of the Real Body category for the "new" picture (i.e., the picture with the same configuration but with more adjacent edges) is still $1/r$. Under this condition the new compression ratio, $1/R_2$, of the Whole List category will be compared with the original compression ratio, $1/R_1$.

For the new picture, since the compression ratio of the Real Body category is also $1/r$, for the same reason as Equation (8-1) $1/r$ will also be

$$1/r = \frac{w \text{ (bytes)}}{b \text{ (bytes)}}, \quad (8-6)$$

where both w and b are the lengths of the memories having contributed to the compression ratio of the Real Body category for the new picture. They belong respectively to the compressed data base and to the sequenced boundary adjacency data base. Equation (8-6) implies that

$$b = rw \text{ (bytes)}. \quad (8-7)$$

Since the new picture has $(y + z)$ adjacent edges, by the same argument of Equation (8-3), p' will be

$$p' = b/2 + y + z = rw/2 + y + z \text{ (words)}. \quad (8-8)$$

From Equations (8-3), (8-5), and (8-8), it is obvious that

$$\begin{aligned} p' &= p + 2z \\ &= rx/2 + y + 2z \\ &= rw/2 + y + z \text{ (words)}. \end{aligned} \quad (8-9)$$

Equation (8-9) implies that

$$w = x + 2z/r. \quad (8-10)$$

By the same argument of Equation (8-4), $1/R_2$ will be

$$1/R_2 = \frac{w + 2(y + z) + 1(y + z)}{b + 2(y + z) + 2(y + z)} = \frac{w + 3y + 3z \text{ (bytes)}}{rw + 4y + 4z \text{ (bytes)}}. \quad (8-11)$$

In order to verify that $1/R_2$ is larger than $1/R_1$ under certain conditions, let

$$D = 1/R_2 - 1/R_1. \quad (8-12)$$

Then, by applying Equation (8-10), D will become

$$\begin{aligned} D &= \frac{w + 3y + 3z}{x + 4y + 4z} - \frac{x + 3y}{rx + 4y} \\ &= \frac{x + 2z/r + 3y + 3z}{(rx + 2z) + 4y + 4z} - \frac{x + 3y}{rx + 4y} \\ &= \frac{(x + 3y) + (2/r + 3)z}{rx + 4y + 6z} - \frac{x + 3y}{rx + 4y} \end{aligned}$$

$$\begin{aligned}
&= \frac{(rx + 4y)(2/r + 3)z - 6z(x + 3y)}{(rx + 4y + 6z)(rx + 4y)} \\
&= \frac{(2/r + 3)z}{rx + 4y + 6z} - \frac{x + 3y}{rx + 4y} \cdot \frac{6z}{rx + 4y + 6z} \\
&= \frac{(2/r + 3)z}{rx + 4y + 6z} - \frac{1}{R1} \cdot \frac{6z}{rx + 4y + 6z} \\
&= \frac{z}{rx + 4y + 6z} \quad 2/r + 3 - 6/R1.
\end{aligned}$$

Since all variables in Equation (8-13) are of positive value, it is sufficient that the sign of the value of $(2/r + 3 - 6/R1)$ implies the sign of D . Equation (8-13) shows that if $1/R1$ is less than $1/2$, then $(2/r + 3 - 6/R1)$ will be of positive value.

From the discussion presented before for the three coding schemes, any compression ratio of the three coding schemes for a picture is, in general, hardly greater than $1/2$. Therefore, under the condition that the compression ratio is not greater than $1/2$, the compression effect of the Whole List category for any of the three coding schemes will be reduced as the number of the adjacent edges is increased.

Furthermore, extracting from Equation (8-13),

$$D = \frac{z}{rx + 4y + 6z} \quad 2/r + 3 - 6/R1, \quad (8-14)$$

implies that as the increment z is getting larger, D will be getting smaller, because the increment for the numerator in the right side of Equation (8-14) is much less than that for the denominator in the same side. This argument thus agrees for the discussion mentioned previously.

The previous discussion claims that the counter-effect described is much less than any compression effect of the three coding schemes.

The mathematical argumentation is described as follows:

The counter-effect can be represented by the difference between the compression effects considered in the Whole List and the Real Body categories. In other words, the counter-effect, ANTI, can be expressed by the following equation:

$$\text{ANTI} = 1/R1 - 1/r. \quad (8-15)$$

The argumentation in fact is desired to show that the ratio of ANTI to $1/r$ can be much less than 1. Let Ra stand for this ratio. From Equations (8-4) and (8-5), Ra can be derived as follows:

$$\begin{aligned} Ra &= \frac{\text{ANTI}}{1/r} \\ &= (1/R1 - 1/r) \cdot r \\ &= \left(\frac{x + 3y}{rx + 4y} - 1/r \right) \cdot r \\ &= \frac{rx + 3ry - rx - 4y}{r \cdot (rx + 4y)} \cdot r \\ &= \frac{3ry - 4y}{rx + 4y} \\ &= \frac{y \cdot (3r - 4)}{y \cdot (rx/y + 4)} \\ &= \frac{3r - 4}{(x/y) \cdot r + 4}. \end{aligned} \quad (8-16)$$

However, in general, in a picture the number of the adjacent edges is much less than that of the edge pixels, so it is very possible that x/y can be much greater than 3. The data in Table I demonstrates that

for any of the four pictures, the x/y value is indeed very much greater than 3.

Theoretically, as the number of the adjacent edges is getting smaller, the possibility that x/y is much larger than 3 will be getting greater, because the average length of the adjacent edges will become larger and larger. Thus, the number of codes representing this adjacent edge will also become larger and larger. This behavior implies that there is the possibility of x/y being greater than 3 and that the value of x/y will also become larger and larger.

As discussed earlier and from Equation (8-16), it is obvious that the value of

$$R_a = \frac{3r - 4}{(x/y) - r + 4} \quad (8-17)$$

must be much less than 1 if x/y is much greater than 3. Thus, it has been shown that the counter-effect is much less than any compression effect of the three coding schemes.

Furthermore, corresponding to a fixed value of r , the value of R_a depends on that of x/y . If x/y is increased, R_a will be decreased. In other words, Equation (8-17) implies that a smaller amount of adjacent edges results in a smaller counter-effect. Thus, in this sense we find that the implication of Equation (8-17) coincides with that of Equation (8-14). It is noted that the function of the counter-effect between the compression effects of the Whole List and Real Body categories is not linear.

The above discussions and argumentations illustrate the phenomenon indicated by the data shown in Table I. For the first coding scheme, the

the compression effect considered in the Real Body category among the pictures is much more stable, so the relation of the compression effects between the Whole List and Real Body categories almost entirely depends upon the counter-effect. The phenomenon shown by the data in the item of the first coding scheme in Table I agrees for the previous arguments.

As the compression effect of the first coding scheme has nothing to do with the regularity of the picture, more attention will be paid to the other two schemes. Without any argument, it is obvious that the Hughes Building is the most regular of the four pictures; the Simple Example Scene is less regular; the AVA aerial photograph is even less regular; and the Kodak Girl is the least regular of the four. Table I shows that although the number of adjacent edges of the Hughes Building is more than those of the AVA and the Simple Example Scene, the compression ratios, either of the second coding scheme or the third coding scheme, of the Whole List category of the Hughes Building is less than those of the AVA and the Simple Example Scene. This experimental result coincides with the previous discussion.

In the previous argumentation, it is stated that the counter-effect is much less than the compression effect of the Real Body category. Since the Hughes Building is the most regular picture among the four pictures, its compression effect of the Real Body category is the best among them. Table I demonstrates that although the counter-effect of the Hughes Building is greater than those of the other three pictures (this fact can be determined from the value of x/y), its compression effect of the Whole List category in the second coding scheme is still better than those of the others.

However, in the item of the third coding scheme, it is found that the compression effect considered in the Whole List category for the Hughes Building is, on the contrary, worse than that for the Simple Example Scene and is almost the same as that for the AVA. The reason for this phenomenon is that the compression ratios of these three pictures in the Real Body category are so close to one another that the counter-effect runs over the compression effect considered in the Real Body category.

The compression between the experimental data associated with the AVA and the Simple Example Scene also demonstrates the same coincidence with the previous discussion. For Kodak Girl, because it has the largest number of adjacent edges and is the least regular picture among the four, it is obvious that its compression effect is the worst among them.

IX. APPLICATION AND CONCLUSION

A series of algorithms for establishing the sequenced boundary adjacency data base has been described. Among these algorithms, the boundary pixel sequencing algorithm shown in Section IV is the most difficult and time-consuming. It is the most important algorithm for the application of the data base compression using the three coding schemes described in previous sections. The second most important algorithm is the adjacent edge sequencing algorithm shown in Section VI. This algorithm assures the availability of sequencing all the adjacent edges of a picture in a clockwise direction. The boundary adjacency tracing algorithm described in Section V enables one to detect the adjacency of the boundaries very quickly.

Three coding schemes for the data base compression without any additional distortion have been described in Section VII and discussed in Section VIII. These coding schemes are all based on direction vector coding. Theoretically, for the compression effect of the Real Body category, the first coding scheme results in a compression of $1/4$ to $1/3$ the original size. The second coding scheme assures that the result of the compression is on the order of $1/62$ to $1/2$. The third coding scheme assures the compression effect of $1/12$ to $1/3$ the original size.

The compression ratio of the Whole List category is always a little larger than that of the Real Body category because of the counter-effect described in the last section. For a more regular picture, the compression effect of the second coding scheme is better than that of the first coding scheme. Since the third coding scheme combines the advantages of

the first and second coding schemes, it is the best coding scheme for the general pictures.

Further research might be directed toward detecting the vertex locations of the important edges (or regions) of the picture [11]. Further research might also be conducted to reconstruct the scene from the compressed data base. Since the direction vector code used in the coding scheme can be the representation of the element of the shape description, further research might also be contributed to identifying the shape of the objects.

LIST OF REFERENCES

LIST OF REFERENCES

1. C. R. Brice, C. L. Fennema, "Scene Analysis Using Regions, " Artificial Intelligence, Vol. 1, 1970, pp. 205-226.
2. C. J. Date, An Introduction to Database Systems, Addison-Wesley Publishing Co., 2nd edition, 1977.
3. H. Freeman, "Shape Description Via the Use of Critical Points," IEEE Computer Society Conference on Pattern Recognition and Image Processing, June 1977, pp. 168-197.
4. H. Freeman, "On the Encoding of Arbitrary Geometric Configuration," IRE Trans. on Electron Computer, Vol. EC-10, June 1961, pp. 260-268.
5. R. C. Gonzalez, P. Wintz, Digital Image Processing, Addison-Wesley Publishing Co., 1977.
6. E. L. Hall, Computer Image Processing and Recognition, Academic Press, New York, to appear.
7. M. H. Hueckel, "An Operator Which Locates Edges in Digitized Pictures," J. Ass. Compt. Mach., Vol. 18, January 1971, pp. 113-125.
8. M. H. Hueckel, "A Local Visual Operator Which Recognizes Edges and Lines," J. ACM., Vol. 20, October 1973, pp. 634-637.
9. Thomas S. Huang, Ohl J. Tretiak, Picture Bandwidth Compression, Gordon and Breach, Science Publishers, Inc., New York, 1972.
10. J. J. Hwang, E. L. Hall, "Scene Representation Using the Adjacency Matrix and Sampled Shapes of Region," Proc. IEEE Conference on Pattern Recognition and Image Processing, Chicago, Illinois, 1978.
11. J. J. Hwang, C. C. Lee, E. L. Hall, "Locating Vertices In A Scene," Proceeding of IEEE, Southeascon '79, April 1979, pp. 357-362.
12. J. J. Hwang, C. C. Lee, E. L. Hall, W. J. Nalesnik, and C. R. Archer, "Global Local Edge Coincidence Segmentation For Computer Tomography Images," submitted to Computer-Aided Analysis of Radiological Images, June 1979.
13. J. J. Hwang, C. C. Lee, E. L. Hall, "Segmentation of Solid Objects Using Global Local Edge Coincidence," submitted to IEEE Conference on Pattern Recognition and Image Processing, August 1979.
14. J. D. Kelly, "Edge Detection in Pictures by Computer Using Planning," Machine Intelligence, B. Meltzer and D. Michie, Eds., Vol. 6, 1971, pp. 397-409.

15. D. E. Knuth, Fundamental Algorithms, Vol. 1, 2nd edition, Addison-Wesley Publishing Co., 1975.
16. A. Martelli, "An Application of Heuristic Search Methods to Edge and Contour Detection," Commun. Ass. Compt. Mach., Vol. 19, February 1976, pp. 73-83.
17. William K. Pratt, Digital Image Processing, John Wiley and Sons, Inc., 1978.
18. A. Rosenfeld, M. Thurston, "Edge and Curve Detection for Visual Scene Analysis," IEEE Trans. Compt., Vol. C-20, May 1971, pp. 562-569.
19. A. Rosenfeld, "Picture Processing by Computer," Computer Surveys, 1, 3, New York, September 1969, pp. 147-176.
20. A. Rosenfeld, M. Thurston, Y-H Lee, "Edge and Curve Detection: Further Experiments," IEEE Trans. Compt., Vol. C-21, July 1972, pp. 677-715.
21. T. G. Saba, D. Bryant, E. L. Hall, "Scene Segmentation Using Uniform Clustering," Proceeding of IEEE, Southeascon '78, April 1978, pp. 217-221.

APPENDIX

```

C          ***  BNDSEQ.FOR  ***
C
C*****
C*  SUBJECT:  BOUNDARY PIXEL SEQUENCING ALGORITHM.  *
C*  PROGRAMMER:  CHIEN-CHYUN LEE.  *
C*  DATE:  OCTOBER 13, 1978.  *
C*  INPUT:  FILENAME MUST BE "*.PIC".  *
C*          THIS INPUT FILE IS A 128*128, 40 BLOCK SEGEMENTED  *
C*          AND REGION LABELED IMAGE PICTURE FILE.  *
C*  OUTPUT:  FILENAME MUST BE "*.DAT".  THE LENGTH OF THIS  *
C*          OUTPUT FILE IS VARIABLE.  *
C*          1. BLOCK 0: INTEGER*2 REGPTR(256) ARRAY FOR REGION  *
C*             POINTER.  EACH ELEMENT STORES A NUMBER STANDING  *
C*             FOR WHICH BLOCK IN THE OUTPUT FILE STARTS TO  *
C*             STORE THE CORRESPONDING REGION'S SEQUENCED  *
C*             BOUNDARY PIXELS.  *
C*          2. BLOCK 1 ---: ALL THE BOUNDARY PIXELS OF EACH  *
C*             REGION ARE STORED IN THE CIRCULAR-LINKED LIST.  *
C*             EACH NODE IN THE LIST HAS TWO WORDS.  THE FIRST  *
C*             WORD STORES THE ONE-DIMENSIONAL POSITION OF THE  *
C*             BOUNDARY PIXEL.  THE SECOND WORD STORES THE  *
C*             LINKAGE POINTER.  *
C*  NOTE:  COMPILING WITH SWITCH "/D" WILL PRINT OUT THE  *
C*          OUTPUT DATA FILE.  *
C*****
C
C
C
C
C$  REGPTR IS REGION POINTER ARRAY.
C
C$  LP (LPIC) IS ORIGINAL IMAGE ARRAY.
C
C$  IND IS INDEX ARRAY SHOWING INNER PIXEL, REGULAR OR SINGULR
C$  BOUNDARY PIXEL.
C
C$  ORDER IS ADJACENCY ORDER ARRAY.
C
C$  BNDPXL IS UNSEQUENCED BOUNDARY PIXEL ARRAY.
C
C$  SEQPXL IS SEQUENCED BOUNDARY PIXEL ARRAY.
C
C$  TEMSEQ IS TEMPORARY ARRAY FOR SEQUENCED BOUNDARY PIXELS.
C
0001  LOGICAL*1 LP(128,128),LPIC(16384),GREY,LINK,INFO,IND(128,128)
0002  LOGICAL*1 ORDER(384),IFLAG,JFLAG,KADJ
0003  INTEGER*2 BNDPXL(2,1280),SEQPXL(2,1536),REGPTR(256)
0004  INTEGER*2 TEMSEQ(2,1280),WORKSP(8192),STACK(384)
0005  INTEGER*2 FILSPC(39),DEFEXT(4),BUFR(3)
0006  DATA DEFEXT/4*3RPIC/
0007  EQUIVALENCE (LP(1,1),LPIC(1),SEQPXL(1,1),WORKSP(1))
0008  EQUIVALENCE (WORKSP(3073),TEMSEQ(1,1))
0009  EQUIVALENCE (WORKSP(5633),BNDPXL(1,1))
0010  COMMON REGPTR,WORKSP,IND

```

FORTRAN IV V01C-03G+ MON 21-MAY-79 02:31:53

PAGE 002

```

C*****
C
C$   INPUT AND OUTPUT ARRANGEMENTS.
C
0011   CALL PRINT('ENTER COMMAND STRING--"* . DAT<*. PIC"')
0012   CALL ICSI(FILSPC,DEFEXT,,,0)
0013   IX=IFETCH(FILSPC(1))
0014   IY=IFETCH(FILSPC(16))
0015   IF((IX+IY).NE.0) STOP ' ? BAD COMMAND STRING'
0017   ICH1=IGETC()
0018   ICH2=IGETC()
0019   IF(ICH1.LT.0.OR.ICH2.LT.0) STOP ' ? NO CHANNEL AVAILABLE'
0021   IF(IENTER(ICH1,FILSPC(1),200).NE.200) STOP ' ? NO SPACE OUT-1'
0023   IF(LOOKUP(ICH2,FILSPC(16),40).NE.40) STOP ' ? NO SUCH FILE-1'
0025   IF(IREADW(8192,LPIC,8,ICH2).NE.8192) STOP ' ? BAD READ-1'
C*****
C
C$   NOBLK IS THE NUMBER OF THE BLOCK WHICH FROM HERE THE
C$   SEQUENCED BOUNDARY PIXELS ARE STORED IN THE OUTPUT DATA BASE.
C
0027   LINK=2
0028   INFO=1
0029   NOBLK=1
C*****
C
C$   DECIDE HOW MANY REGIONS ARE IN THE PICTURE.
C
0030   GREY=-128
0031   DO 100 I=1,128
0032   DO 100 J=1,128
0033   IF(GREY.LT.LP(J,I)) GREY=LP(J,I)
0035 100   CONTINUE
0036   MAXREG=GREY
0037   MAXREG=MAXREG+128
0038   TYPE109,MAXREG
0039 109   FORMAT(' MAXIMUM REGIONS',I5)
C*****
C
C$   REGION POINTER IS INITIALIZED TO INDICATE "NULL" REGION.
C
0040   DO 150 I=1,256
0041   REGPTR(I)=-5
0042 150   CONTINUE
C*****
C*****
C
C$   THE SEQUENCING IS PROCESSED REGION BY REGION.
C
C$   WHEN IT STARTS TO PROCESS THE NEXT REGION, ALL ASSOCIATED
C$   VARIABLES NEED TO BE INITIALIZED.
C
C$   THE INDEX FOR INNER (REGULAR OR SINGULAR BOUNDARY) PIXEL
C$   SHOULD BE REDETECTED.
C

```

FORTRAN IV VOIC-036+ MON 21-MAY-79 02:31:53

PAGE 003

```

0043      DO 3000 KREG=1,MAXREG
0044          NOPXL=0
0045          IBPNTR=0
0046          IHEADB=1
0047          NOSTAK=0
0048          IREGNO=KREG-129
0049          GREY=IREGNO
0050          TYPE59,KREG
0051      D      WRITE(6,59) KREG
0052      59    FORMAT(1H0,20X,'$$$$$  TRACING REGION',I5)
C*****
C
C$      INITIALIZE STACK AND ORDER FOR STORING THE ADJACENCY ORDER.
C
0053      DO 2300 I=1,384
0054          STACK(I)=-5
0055          ORDER(I)=-5
0056      2300  CONTINUE
C*****
C
C$      INITIALIZE INDEX ARRAY.
C
0057      DO 2150 I=1,128
0058          DO 2150 J=1,128
0059              IND(J,I)=-1
0060      2150  CONTINUE
C*****
C
C$      INDEX 0 --- THE CURRENT REGION'S INNER PIXEL.
C
C$      INDEX 1 --- THE CURRENT REGION'S REGULAR BOUNDARY PIXEL.
C
0061      DO 2100 I=2,127
0062          DO 2100 J=2,127
0063              IF(LP(J,I).NE.GREY) GO TO 2100
0064              IND(J,I)=1
0065              IF(GREY.EQ.LP(J-1,I).AND.GREY.EQ.LP(J+1,I).AND.GREY.EQ.
- LP(J,I-1).AND.GREY.EQ.LP(J,I+1)) IND(J,I)=0
0068      2100  CONTINUE
0069          TYPE2109
0070      2109  FORMAT(1H0,5X,'$$$  REGULAR BOUNDARY PIXELS DECIDED !')
C*****
C
C$      INDEX 2 --- THE CURRENT REGION'S SINGULAR BOUNDARY PIXEL.
C
0071      DO 2200 I=2,127
0072          DO 2200 J=2,127
0073              IF(IND(J,I).NE.1) GO TO 2200
0075              IF(IND(J+1,I).EQ.0.OR.IND(J+1,I+1).EQ.0.OR.IND(J,I+1).EQ.0
- OR.IND(J-1,I+1).EQ.0.OR.IND(J-1,I).EQ.0.OR.IND(J-1,I-1)
- EQ.0.OR.IND(J,I-1).EQ.0.OR.IND(J+1,I-1).EQ.0) GO TO 2200
0077              IND(J,I)=2
0078      2200  CONTINUE
0079          TYPE2209

```

FORTRAN IV V01C-03G+ MON 21-MAY-79 02:31:53 PAGE 004

```

0080 2209  FORMAT(1H0,10X,'== SINGULAR BOUNDARY PIXELS DECIDED !')
      C*****
      C
      C$  COMPUTE THE ADJACENCY ORDER.
      C
      C$  IF A PIXEL'S ADJACENCY ORDER IS GREATER THAN 2, IT IS
      C$  STORED IN THE "STACK", AND ITS ADJACENCY ORDER
      C$  IS STORED IN THE "ORDER".
      C
0081      DO 2450 I=2,127
0082      DO 2450 J=2,127
0083      IF(IND(J,I).NE.1) GO TO 2450
0085      KADJ=0
0086      IF(IND(J+1,I).EQ.1) KADJ=KADJ+1
0088      IF(IND(J+1,I+1).EQ.1) KADJ=KADJ+1
0090      IF(IND(J,I+1).EQ.1) KADJ=KADJ+1
0092      IF(IND(J-1,I+1).EQ.1) KADJ=KADJ+1
0094      IF(IND(J-1,I).EQ.1) KADJ=KADJ+1
0096      IF(IND(J-1,I-1).EQ.1) KADJ=KADJ+1
0098      IF(IND(J,I-1).EQ.1) KADJ=KADJ+1
0100      IF(IND(J+1,I-1).EQ.1) KADJ=KADJ+1
0102      IF(KADJ.LE.2) GO TO 2450
0104      NOSTAK=NOSTAK+1
0105      IF(NOSTAK.GT.384) STOP '? STACK > 384'
0107      STACK(NOSTAK)=(I-1)*128+J
0108      ORDER(NOSTAK)=KADJ
0109 2450  CONTINUE
      C*****
      C
      C$  STORE ALL THE UNSEQUENCED BOUNDARY PIXELS IN THE CIRCULAR-LINKED
      C$  LIST IN THE "BNDPXL" ARRAY.
      C
      C$  SINGULAR BOUNDARY PIXEL IS REPRESENTED BY NEGATIVE POSITION.
      C
0110      DO 300 I=2,127
0111      DO 300 J=2,127
0112      IF(IND(J,I).LT.1) GO TO 300
0114  D  WRITE(6,169) J,I,KREG
0115  D169  FORMAT(' XCOD=',I4,5X,' YCOD=',I4,5X,' REGION',I4)
0116      LOCATN=(I-1)*128+J
0117      IF(IND(J,I).EQ.2) LOCATN=-LOCATN
0119      IBPNTR=IBPNTR+1
0120      NOPXL=IBPNTR
0121      IF(NOPXL.GT.1280) STOP '?? BNDPXL > 1280'
0123      BNDPXL(LINK,IBPNTR)=BNDPXL(LINK,IHEADB)
0124      BNDPXL(INFO,IBPNTR)=LOCATN
0125      BNDPXL(LINK,IHEADB)=IBPNTR
0126      IHEADB=IBPNTR
0127 300  CONTINUE
0128      TYPE209,NOPXL
0129 209  FORMAT('0 --->>,I6,' PIXELS TRACED !')
      C*****
      C
      C$  START THE SEQUENCING ALGORITHM.

```

FORTRAN IV

V01C-03G+ MON 21-MAY-79 02:31:53

PAGE 005

```

C
C*  ONCE A PIXEL IS SEQUENCED, IT IS DELETED FROM THE UNSEQUENCED
C*  CIRCULAR-LINKED LIST.
C
C*  "ISPNTN" IS A POINTER POINTING TO CIRCULAR-LINKED LIST
C*  FOR SEQUENCED BOUNDARY PIXELS.
C
C*  "ICPOTN" STANDS FOR THE CURRENT NODE.
C
C*  "IFIRST" STANDS FOR THE STARTING NODE.
C
0130      IF(IBPNTR.EQ.0) GO TO 3000
0132      TYPE309,KREG
0133      D  WRITE(6,309) KREG
0134      309  FORMAT('0***** SEQUENCING REGION',I5)
0135      IHEADS=1
0136      ISPNTN=0
0137      ICPOTN=BNDPXL(LINK,IHEADB)
0138      IPPOTN=IHEADB
0139      IFLAG=0
C*****
C
C*  FIND THE FIRST UNSEQUENCED REGULAR BOUNDARY PIXEL.
C
C*  IF NO REGULAR BOUNDARY PIXEL EXISTS AND "IFALG" NE. 0,
C*  I.E., ALL REGULAR BOUNDARY PIXELS HAVE BEEN SEQUENCED,
C*  THEN GO TO SEQUENCE THE SINGULAR BOUNDARY PIXEL.
C
0140      400  IFIRST=ICPOTN
0141      500  IF(BNDPXL(INFO,ICPOTN).GT.0) GO TO 250
0143      IPPOTN=ICPOTN
0144      ICPOTN=BNDPXL(LINK,ICPOTN)
0145      IF(ICPOTN.NE.IFIRST) GO TO 500
0147      550  BNDPXL(INFO,ICPOTN)=IABS(BNDPXL(INFO,ICPOTN))
0148      IPPOTN=ICPOTN
0149      ICPOTN=BNDPXL(LINK,ICPOTN)
0150      IF(ICPOTN.NE.IFIRST) GO TO 550
0152      IF(IFLAG.NE.0) GO TO 850
C*****
C
C*  PUT THIS REGULAR BOUNDARY PIXEL INTO THE SEQUENCED
C*  CIRCULAR-LINKED LIST.
C
C*  THIS PIXEL IS THE STARTING SEQUENCED PIXEL.
C
C*  IFLAG IS CHANGED TO 1 TO INDICATE REGULAR BOUNDARY
C*  PIXEL SEQUENCING.
C
0154      250  KLOC=BNDPXL(INFO,ICPOTN)
0155      600  ISPNTN=ISPNTN+1
0156      IFLAG=1
0157      SEQPXL(LINK,ISPNTN)=SEQPXL(LINK,IHEADS)
0158      SEQPXL(INFO,ISPNTN)=BNDPXL(INFO,ICPOTN)
0159      SEQPXL(LINK,IHEADS)=ISPNTN

```

FORTRAN IV

V01C-036+ MON 21-MAY-79 02:31:53

PAGE 006

```

0160      IHEADS=ISPNTR
      C*****
      C
      C$  "ICPOTN" .EQ. "IPPOTN" MEANS THE UNSEQUENCED CIRCULAR-LINKED
      C$  LIST BECOMES EMPTY.
      C
      C$  START TO TRACE UNSEQUENCED CIRCULAR-LINKED LIST TO DECIDE
      C$  WHICH PIXEL IS THE NEXT SEQUENCED PIXEL.
      C
0161      IF(ICPOTN.EQ.IPPOTN) GO TO 2800
0163      JCPTLN=BNDPXL(INFO,ICPOTN)
0164      ICPOTN=BNDPXL(LINK,ICPOTN)
0165      BNDPXL(LINK,IPPOTN)=ICPOTN
0166      NOPXL=NOPXL-1
0167 650    IFIRST=ICPOTN
0168      BUFR(1)=IPPOTN
0169      BUFR(2)=ICPOTN
0170      BUFR(3)=9
      C*****
      C
      C$  FOR EACH TRACED PIXEL, SUBROUTINE "KPRO" IS CALLED TO DETERMINE
      C$  ITS SEQUENCING PRIORITY.
      C
      C$  SMALLER PRIORITY NUMBER MEANS HIGHER PRIORITY.
      C
0171 700    JCPTLN=BNDPXL(INFO,ICPOTN)
0172      IF(JCPTLN.LT.0) GO TO 800
0174      JDIFER=JCPTLN-JOPTLN
0175      KPROTY=KPRO(JDIFER)
0176      IF(KPROTY.GT.BUFR(3)) GO TO 800
      C*****
      C
      C$  THE "BUFR" IS UPDATED TO STORE THE TRACED PIXEL OF HIGHER
      C$  SEQUENCING PRIORITY.
      C
      C$  AFTER TRACING OPERATION, "BUFR" ARRAY SHOWS WHICH PIXEL
      C$  IS THE NEXT SEQUENCED PIXEL.
      C
0178      BUFR(1)=IPPOTN
0179      BUFR(2)=ICPOTN
0180      BUFR(3)=KPROTY
0181 800    IPPOTN=ICPOTN
0182      ICPOTN=BNDPXL(LINK,ICPOTN)
0183      IF(ICPOTN.NE.IFIRST) GO TO 700
0185      IPPOTN=BUFR(1)
0186      ICPOTN=BUFR(2)
0187      KPROTY=BUFR(3)
0188      IF(KPROTY.NE.9) GO TO 600
      C*****
      C
      C$  IF NO PIXEL CAN BE SEQUENCED, CHECK IF CLOSED BOUNDARY
      C$  HAS BEEN FORMED.
      C
0190      LOCA=SEQPXL(INFO,IHEADS)

```

FORTRAN IV

VOIC-03G+ MON 21-MAY-79 02:31:53

PAGE 007

```

0191      JCPNTR=IHEADS
0192      JDIFER=IABS(LOCA-KLOC)
0193      IF(JDIFER.EQ. 1. OR. JDIFER.EQ. 127. OR. JDIFER.EQ. 128. OR.
- JDIFER.EQ. 129) GO TO 850
C*****
C
C$      IF THE SEQUENCED BOUNDARY HAS NOT BEEN CLOSED, TRACE
C$      "ORDER" AND "STACK" ARRAYS TO SEE IF THERE IS A PIXEL
C$      OF HIGHER ADJACENCY ORDER THAT CAN BE SEQUENCED NEXT.
C
C$      ONCE A PIXEL IS SEQUENCED, IT IS REMOVED FROM THE "STACK";
C$      THEN GO BACK TO TRACE UNSEQUENCED CIRCULAR-LINKED LIST.
C
0195      DO 750 NDDR=7,3,-1
0196      DO 750 NOSTAK=1,384
0197      IF(ORDER(NOSTAK) NE. NDDR) GO TO 750
0199      JDIFER=IABS(LOCA-STACK(NOSTAK))
0200      IF(JDIFER. NE. 1. AND. JDIFER. NE. 127. AND. JDIFER. NE. 128. AND.
- JDIFER. NE. 129) GO TO 750
0202      ISPNTR=ISPNTR+1
0203      IF(ISPNTR.GT. 1536) STOP '? SEQPXL > 1536'
0205      SEQPXL(LINK, ISPNTR)=SEQPXL(LINK, IHEADS)
0206      SEQPXL(INFO, ISPNTR)=STACK(NOSTAK)
0207      SEQPXL(LINK, IHEADS)=ISPNTR
0208      IHEADS=ISPNTR
0209      JOPTLN=STACK(NOSTAK)
0210      STACK(NOSTAK)=-5
0211      ORDER(NOSTAK)=-5
0212      GO TO 650
0213 750  CONTINUE
C*****
C
C$      START TO SEQUENCE THE SINGULAR BOUNDARY PIXEL.
C
C$      TRACE THE SEQUENCED CIRCULAR-LINKED LIST TO SEE IF THERE
C$      IS A PIXEL IN THE UNSEQUENCED CIRCULAR-LINKED LIST
C$      ADJACENT TO THE CURRENT TRACED SEQUENCED PIXEL
C$      (REFERENCE PIXEL).
C
C$      "JDIFER" .EQ. 1 OR "JDIFER" .EQ. 128 MEANS IT IS
C$      FOUR-CONNECTIVE.
C
C$      "JDIFER" .EQ. 127 OR "JDIFER" .EQ. 128 MEANS IT IS
C$      EIGHT-CONNECTIVE (BUT NOT FOUR-CONNECTIVE).
C
C$      IF NO PIXEL IN THE SEQUENCED CIRCULAR-LINKED LIST IS
C$      ADJACENT TO ANY REMAINING PIXEL IN THE UNSEQUENCED
C$      CIRCULAR-LINKED LIST, THEN A SEPARABLE CLOSED BOUNDARY
C$      EXISTS. THE SEQUENCING ALGORITHM IS REPEATED.
C
0214 850  JCPNTR=SEQPXL(LINK, IHEADS)
0215      JFPNTR=IHEADS
0216      JFIRST=JCPNTR
0217      IFIRST=ICPOTN

```

```

FORTRAN IV          VOIC-03G+  MON 21-MAY-79 02:31:53          PAGE 008

0218  900    JCPTLN=SEQPXL(INFO,JCPNTR)
0219  1000   JCPTLN=BNDPXL(INFO,ICPOTN)
0220        IF(JCPTLN.LT.0) GO TO 1050
0222        JDIFER=IABS(JCPTLN-JCPTLN)
0223        IF(JDIFER.EQ.1.OR.JDIFER.EQ.128) GO TO 1100
0225        IF(JDIFER.EQ.127.OR.JDIFER.EQ.129) GO TO 1800
0227  1050   IPPOTN=ICPOTN
0228        ICPOTN=BNDPXL(LINK,ICPOTN)
0229        IF(IFIRST.NE.ICPOTN) GO TO 1000
0231        JPPNTR=JCPNTR
0232        JCPNTR=SEQPXL(LINK,JCPNTR)
0233        IF(JFIRST.NE.JCPNTR) GO TO 900
0235        GO TO 400
C*****
C
C*    IF THE CURRENT TRACED PIXEL (PIXEL M) IS EIGHT-CONNECTIVE
C*    RATHER THAN FOUR-CONNECTIVE TO THE REFERENCE PIXEL, KEEP
C*    ON TRACING UNSEQUENCED CIRCULAR-LINKED LIST TO SEE IF ANY
C*    PIXEL IS ADJACENT TO THIS CURRENT TRACED PIXEL.
C
0236  1800   MC=ICPOTN
0237        MP=IPPOTN
0238        IFIRST=ICPOTN
0239        IPPOTN=ICPOTN
0240        ICPOTN=BNDPXL(LINK,ICPOTN)
0241        BUFR(3)=9
0242  1900   LOCATN=BNDPXL(INFO,ICPOTN)
0243        IF(LOCATN.LT.0) GO TO 2000
0245        JDIFER=LOCATN-JCPTLN
0246        KPROTY=KPRO(JDIFER)
0247        IF(KPROTY.GT.BUFR(3)) GO TO 2000
0249        BUFR(3)=KPROTY
0250  2000   IPPOTN=ICPOTN
0251        ICPOTN=BNDPXL(LINK,ICPOTN)
0252        IF(ICPOTN.NE.IFIRST) GO TO 1900
0254        ICPOTN=MC
0255        IPPOTN=MP
0256        KPROTY=BUFR(3)
0257        IF(KPROTY.NE.9) GO TO 1100
C*****
C
C*    IF NO PIXEL IS ADJACENT TO PIXEL M, TRACE NEXT PIXEL (PIXEL X)
C*    IN THE SEQUENCED CIRCULAR-LINKED LIST TO SEE IF IT IS
C*    FOUR-CONNECTIVE TO PIXEL M.
C
C*    IF PIXEL X IS FOUR-CONNECTIVE TO PIXEL M, CHANGE THE
C*    REFERENCE PIXEL TO PIXEL X.
C
0259        LOCATN=BNDPXL(INFO,ICPOTN)
0260        MC=JCPNTR
0261        MP=JPPNTR
0262        JPPNTR=JCPNTR
0263        JCPNTR=SEQPXL(LINK,JCPNTR)
0264        JCPTLN=SEQPXL(INFO,JCPNTR)

```

FORTRAN IV V01C-03G+ MON 21-MAY-79 02:31:53 PAGE 009

```

0265      JDIFER=IABS(JCPTLN-LOCATN)
0266      IF(JDIFER.EQ.1.OR.JDIFER.EQ.128) GO TO 1100
0268      JCPNTR=MC
0269      JPPNTR=MP
C*****
C
C$      PUT ALL SEQUENCED SINGULAR BOUNDARY PIXELS INTO TEMPORARY
C$      CIRCULAR-LINKED LIST.
C
0270 1100  IHEADT=1
0271      ITPNTR=0
0272 1200  ITPNTR=ITPNTR+1
0273      TEMSEQ(LINK,ITPNTR)=TEMSEQ(LINK,IHEADT)
0274      TEMSEQ(INFO,ITPNTR)=BNDPXL(INFO,ICPOTN)
0275      TEMSEQ(LINK,IHEADT)=ITPNTR
0276      IHEADT=ITPNTR
C*****
C
C$      START TO DECIDE WHICH SINGULAR BOUNDARY PIXEL IS THE
C$      NEXT SEQUENCED PIXEL.
C
0277      NOPXL=NOPXL-1
0278      IF(ICPOTN.EQ.IPPOTN) GO TO 1500
0280      JOPTLN=BNDPXL(INFO,ICPOTN)
0281      ICPOTN=BNDPXL(LINK,ICPOTN)
0282      BNDPXL(LINK,IPOTN)=ICPOTN
0283      IFIRST=ICPOTN
0284      BUFR(1)=IPOTN
0285      BUFR(2)=ICPOTN
0286      BUFR(3)=9
C*****
C
C$      KEEP ON SEQUENCING UNTIL NO MORE PIXELS CAN BE SEQUENCED.
C
0287 1300  JCPTLN=BNDPXL(INFO,ICPOTN)
0288      IF(JCPTLN.LT.0) GO TO 1400
0290      JDIFER=JCPTLN-JOPTLN
0291      KPROTY=KPRO(JDIFER)
0292      IF(KPROTY.GT.BUFR(3)) GO TO 1400
0294      BUFR(1)=IPOTN
0295      BUFR(2)=ICPOTN
0296      BUFR(3)=KPROTY
0297 1400  IPPOTN=ICPOTN
0298      ICPOTN=BNDPXL(LINK,ICPOTN)
0299      IF(ICPOTN.NE.IFIRST) GO TO 1300
0301      IPPOTN=BUFR(1)
0302      ICPOTN=BUFR(2)
0303      KPROTY=BUFR(3)
0304      IF(KPROTY.NE.9) GO TO 1200
C*****
C
C$      CHECK IF THE LAST PIXEL IN THE TEMPORARY CIRCULAR-LINKED
C$      LIST IS ADJACENT TO THE PIXEL NEXT TO THE REFERENCE PIXEL
C$      IN THE SEQUENCED CIRCULAR-LINKED LIST.

```

FORTRAN IV V01C-03G+ MON 21-MAY-79 02:31:53 PAGE 010

```

      C
0306 1500 JFLAG=0
0307      KCPNTR=TEMSEQ(LINK, IHEADT)
0308      KFIRST=KCPNTR
0309      JDIFER=TEMSEQ(INFO, IHEADT)-SEQPXL(INFO, SEQPXL(LINK, JCPNTR))
0310      KPROTY=KPRO(JDIFER)
0311      IF(KPROTY.EQ. 9) GO TO 1550
0313      JFLAG=1
0314      JPPNTR=JCPNTR
0315      JCPNTR=SEQPXL(LINK, JCPNTR)
0316      GO TO 1600
      C*****
      C
      C#   PUT ONE MORE REFERENCE PIXEL INTO THE SEQUENCED
      C#   CIRCULAR-LINKED LIST FOR INSERTING THE SEQUENCED SINGULAR
      C#   BOUNDARY PIXELS IN "FORTH AND BACK" SEQUENTIAL ORDER.
      C
0317 1550 ISPNTR=ISPNTR+1
0318      IF(ISPNTR.GT. 1536) STOP '? SEQPXL > 1536'
0320      SEQPXL(LINK, ISPNTR)=JCPNTR
0321      SEQPXL(INFO, ISPNTR)=SEQPXL(INFO, JCPNTR)
0322      SEQPXL(LINK, JPPNTR)=ISPNTR
0323      JPPNTR=ISPNTR
      C*****
      C
      C#   INSERT THE SEQUENCED SINGULAR BOUNDARY PIXELS INTO
      C#   SEQUENCED CIRCULAR-LINKED LIST IN "FORTH" SEQUENTIAL ORDER.
      C
0324 1600 ISPNTR=ISPNTR+1
0325      IF(ISPNTR.GT. 1536) STOP '? SEQPXL > 1536'
0327      SEQPXL(LINK, ISPNTR)=JCPNTR
0328      SEQPXL(INFO, ISPNTR)=TEMSEQ(INFO, KCPNTR)
0329      SEQPXL(LINK, JPPNTR)=ISPNTR
0330      JPPNTR=ISPNTR
0331      KCPNTR=TEMSEQ(LINK, KCPNTR)
0332      IF(KCPNTR.NE. KFIRST) GO TO 1600
      C*****
      C
      C#   INSERT THE SEQUENCED SINGULAR BOUNDARY PIXELS INTO THE
      C#   SEQUENCED CIRCULAR-LINKED LIST IN "BACK" SEQUENTIAL ORDER.
      C
0334      IF(JFLAG.EQ. 1) GO TO 1750
0336      IF(KCPNTR.EQ. TEMSEQ(LINK, KCPNTR)) GO TO 1750
0338      KFIRST=IHEADT
0339 1700 ISPNTR=ISPNTR+1
0340      IF(ISPNTR.GT. 1536) STOP '? SEQPXL > 1536'
0342      SEQPXL(LINK, ISPNTR)=JCPNTR
0343      SEQPXL(INFO, ISPNTR)=TEMSEQ(INFO, KCPNTR)
0344      SEQPXL(LINK, JPPNTR)=ISPNTR
0345      JCPNTR=ISPNTR
0346      KCPNTR=TEMSEQ(LINK, KCPNTR)
0347      IF(KCPNTR.NE. KFIRST) GO TO 1700
      C*****
      C

```

FORTRAN IV

V01C-03G+ MON 21-MAY-79 02:31:53

PAGE 011

```

C$  IF ALL SINGULAR BOUNDARY PIXELS HAVE BEEN SEQUENCED, PUT
C$  ALL THE SEQUENCED BOUNDARY PIXELS INTO THE OUTPUT DATA
C$  BASE; OTHERWISE, REPEAT THE SEQUENCING OPERATION OF
C$  SINGULAR BOUNDARY PIXEL.
C
0349 1750 IF(NOPXL.EQ.0) GO TO 2800
0351      GO TO 850
0352 2800 TYPE2809, ISPNTR
0353 2809 FORMAT('O ----> ', I5, ' PIXELS SEQUENCED !')
0354      ISPNTR=ISPNTR*2
0355      IX=MOD(ISPNTR,256)
0356      IF(IX.EQ.0) IX=256
0358      JBLKS=(ISPNTR-IX)/256+1
0359      NOWORD=256*JBLKS
0360      IF(IWRITW(NOWORD, SEQPXL, NOBLK, ICH1). NE. NOWORD)
- STOP '? BAD WRITE-1'
0362      REGPTR(KREG)=NOBLK
0363      NOBLK=NOBLK+JBLKS
C*****
C
C$  PRINT OUT ALL SEQUENCED BOUNDARY PIXELS.
C
0364 D      JCPNTR=SEQPXL(LINK, IHEADS)
0365 D      JFIRST=JCPNTR
0366 D2500 LOCATN=SEQPXL(INFO, JCPNTR)
0367 D      IX=MOD(LOCATN,128)
0368 D      IF(IX.EQ.0) IX=128
0370 D      IY=(LOCATN-IX)/128+1
0371 D      WRITE(6,169) IX, IY, KREG
0372 D      JCPNTR=SEQPXL(LINK, JCPNTR)
0373 D      IF(JCPNTR.NE. JFIRST) GO TO 2500
C*****
0375 IF(IREADW(8192, LPIC, 8, ICH2). NE. 8192) STOP '? BAD READ-2'
0377 3000 CONTINUE
0378 IF(IWRITW(256, REGPTR, 0, ICH1). NE. 256) STOP '? BAD WRITE-2'
0380 CALL CLOSEC(ICH1)
0381 STOP 'NORMAL END EXECUTION'
0382 END

```

FORTRAN IV V01C-03G+ MON 21-MAY-79 02:38:06

PAGE 001

```
0001      FUNCTION KPRO(JDIFER)
      C*****
      C
      C$   THIS SUBROUTINE DECIDES THE SEQUENCING PRIORITY
      C$   OF THE CURRENT TRACED PIXEL.
      C
0002      KPRO=9
0003      IF(JDIFER.EQ.1) KPRO=1
0005      IF(JDIFER.EQ.129) KPRO=5
0007      IF(JDIFER.EQ.128) KPRO=2
0009      IF(JDIFER.EQ.127) KPRO=6
0011      IF(JDIFER.EQ.-1) KPRO=3
0013      IF(JDIFER.EQ.-129) KPRO=7
0015      IF(JDIFER.EQ.-128) KPRO=4
0017      IF(JDIFER.EQ.-127) KPRO=8
0019      RETURN
0020      END
```

FORTRAN IV

V01C-03G+ MON 21-MAY-79 02:39:54

PAGE 001

```

C          *** ADJDCN.FOR ***
C
C*****
C* SUBJECT:  BOUNDARY ADJACENCY TRACING ALGORITHM.      *
C* PROGRAMMER:  CHIEN-CHYUN LEE.                        *
C* DATE:  OCTOBER 15, 1978.                             *
C* INPUT:  128*128 IMAGE PICTURE. (40 BLOCK FILE).      *
C*          THE PICTURE SHOULD BE SEGMENTED INTO LABELED *
C*          REGIONS.                                     *
C* OUTPUT:  40 BLOCK DATA FILE (FILENAME SHOULD BE *.DAT). *
C*          1. BLOCK 39: REGPTR(256) ARRAY FOR REGION POINTER. *
C*          2. BLOCK 32 -- BLOCK 38: SEQUENTIAL REGION *
C*             ADJACENCY TABLE.                         *
C*             NEGATIVE VALUES REPRESENT END MARKS.      *
C*          3. BLOCK 0 -- BLOCK 31: SEQUENTIAL LIST FOR *
C*             ADJACENT BOUNDARY PIXELS.                  *
C*             ZERO VALUES REPRESENT END MARKS.          *
C* %% NOTE:  1. EACH ELEMENT IN THE ADJACENCY TABLE HAS TWO *
C*             INTEGER*2 WORDS. FIRST WORD STORES A REGION *
C*             NUMBER FOR ADJACENT REGION. SECOND WORD IS A *
C*             POINTER POINTING TO THE SEQUENTIAL LIST.    *
C*             2. EACH ELEMENT IN THE SEQUENTIAL LIST HAS TWO *
C*             LOGICAL*1 BYTES. FIRST BYTE STORES X-COORDINATE *
C*             POSITION. SECOND BYTE STORES Y-COORDINATE *
C*             POSITION.                                     *
C*             3. COMPILING WITH SWITCH "/D" WILL PRINT OUT *
C*             THE OUTPUT DATA BASE OF THIS PROGRAM.      *
C*****
C
C
C
C$ "LP (LPIC)" IS THE ORIGINAL IMAGE ARRAY.
C
C$ "REGPTR" IS REGION POINTER ARRAY.
C
C$ "ADJTAB" ARRAY STANDS FOR ADJACENCY TABLE.
C
C$ "ADJPXL" ARRAY IS USED FOR STORING ADJACENT EDGE PIXELS.
C
C$ "BUFER" IS THE BUFFER FOR STORING THE REGION NUMBERS OF
C$ THE EIGHT-CONNECTIVE PIXELS OF THE DETECTED EDGE PIXEL.
C
C$ "WKSP1" AND "WKSP2" ARE THE WORKING SPACE TO STORE TEMPORARILY
C$ THE CIRCULAR-LINKED LISTS OF ADJACENT REGIONS OF THE CURRENTLY
C$ PROCESSED REGION.
C
0001 LOGICAL*1 LP(128,128),LPIC(16896),ADJPXL(2,2048),XREG
0002 LOGICAL*1 WKSP1(2,2048),XCOD,YCOD,ADJREG,PNTN
0003 INTEGER*2 REGPTR(256),ADJTAB(2,896),WKSP2(2048);BUFER(8)
0004 INTEGER*2 FILSPC(39),DEFEXT(4),TABPNR,TEMPNR(256)
0005 DATA DEFEXT/4*3RPIC/
0006 EQUIVALENCE (LP(1,1),LPIC(1))
C*****
C

```

FORTRAN IV

V01C-03G+ MON 21-MAY-79 02:39:54

PAGE 002

```

      C*   INPUT AND OUTPUT ARRANGEMENTS.
      C
0007      CALL PRINT('ENTER COMMAND STRING -- *.DAT<*. *')
0008      CALL ICSI(FILSPC,DEFEXT,,,0)
0009      IX=IFETCH(FILSPC(1))
0010      IY=IFETCH(FILSPC(16))
0011      IF((IX+IY).NE.0) STOP '? ERROR COMMAND STRING ?'
0013      ICH1=IGETC()
0014      ICH2=IGETC()
0015      IF(ICH1.LT.0.OR.ICH2.LT.0) STOP '? NO CHANNEL AVAILABLE'
0017      IF(IENTER(ICH1,FILSPC(1),40).NE.40) STOP '? NO SPACE OUT'
0019      IF(LOOKUP(ICH2,FILSPC(16),40).NE.40) STOP '? NO SUCH FILE'
0021      IF(IREADW(8192,LPIC,8,ICH2).NE.8192) STOP '? BAD READ-1'
      C*****
      C
      C*   "ADJREG" STANDS FOR ADJACENT REGION.
      C
      C*   "PNTR" IS THE POINTER IN THE ADJACENCY TABLE.
      C
      C*   "TABPNR" IS THE MEMORY CURSOR IN THE ADJACENCY TABLE.
      C
      C*   "LSTPNR" IS THE MEMORY CURSOR IN THE "ADJPXL" ARRAY.
      C
0023      XCOD=1
0024      YCOD=2
0025      ADJREG=1
0026      PNTR=2
0027      NTIME=0
0028      TABPNR=1
0029      LSTPNR=1
0030      NOBLK=0
      C*****
      C
      C*   INITIALIZE THE REGION POINTER TO INDICATE POINTING TO
      C*   "NULL" REGION.
      C
0031      DO 100 N=1,256
0032      REGPTR(N)=-5
0033  100 CONTINUE
      C*****
      C
      C*   DETERMINE HOW MANY REGIONS ARE IN THE PICTURE.
      C
0034      XREG=-128
0035      DO 200 I=1,128
0036      DO 200 J=1,128
0037      IF(LP(J,I).GT.XREG) XREG=LP(J,I)
0039  200 CONTINUE
0040      MAXREG=XREG
0041      MAXREG=MAXREG+129
0042      TYPE209,MAXREG
0043  209 FORMAT('O MAXIMUM REGION NUMBER IS ',I4)
      C*****
      C

```

FORTRAN IV

V01C-03G+ MON 21-MAY-79 02:39:54

PAGE 003

```

C$ THE BOUNDARY ADJACENCY TRACING ALGORITHM IS OPERATED
C$ REGION BY REGION.
C
C$ "TEMPNR" ARRAY IS USED TO STORE THE REGION POINTERS POINTING
C$ TO THE CIRCULAR-LINKED LISTS STORING THE EDGE PIXELS OF
C$ THE CORRESPONDING ADJACENT REGIONS IN THE TEMPORARY
C$ WORKING SPACE.
C
C$ WHEN IT STARTS TO PROCESS A REGION, "TEMPNR" MUST BE
C$ INITIALIZED TO INDICATE "NULL" REGION.
C
0044 DO 2000 KREG=2,MAXREG
0045 IWPNTR=1
0046 DO 300 N=1,MAXREG
0047 TEMPNR(N)=-5
0048 300 CONTINUE
0049 IREGNO=KREG-129
0050 XREG=IREGNO
0051 TYPE309,KREG
0052 309 FORMAT('O DETECT REGION ',I4)
C*****
C
C$ DETECT THE BOUNDARY PIXEL OF THE CURRENTLY PROCESSED REGION.
C
0053 DO 1200 I=2,127
0054 DO 1200 J=2,127
0055 IF(XREG.NE.LP(J,I)) GO TO 1200
0057 IF(XREG.EQ.LP(J-1,I).AND.XREG.EQ.LP(J+1,I).AND.XREG.EQ.
- LP(J,I-1).AND.XREG.EQ.LP(J,I+1)) GO TO 1200
0059 IF(IWPNTR.GT.2048) GO TO 2200
C*****
C
C$ PUT THE EIGHT-CONNECTIVE PIXELS OF THE CURRENTLY DETECTED
C$ BOUNDARY PIXEL INTO THE BUFFER.
C
0061 BUFER(1)=LP(J+1,I)
0062 BUFER(2)=LP(J+1,I+1)
0063 BUFER(3)=LP(J,I+1)
0064 BUFER(4)=LP(J-1,I+1)
0065 BUFER(5)=LP(J-1,I)
0066 BUFER(6)=LP(J-1,I-1)
0067 BUFER(7)=LP(J,I-1)
0068 BUFER(8)=LP(J+1,I-1)
C*****
C
C$ FOR THESE EIGHT PIXELS IN THE BUFFER, IF THERE IS
C$ MORE THAN ONE PIXEL BELONGING TO REGION "I," ELIMINATE
C$ THESE PIXELS OF THE REGION "I" FROM THE BUFFER SUCH THAT ONLY
C$ ONE PIXEL OF THE REGION "I" IS LEFT IN THE BUFFER.
C
0069 DO 400 K=1,7
0070 IF(BUFER(K).EQ.-555) GO TO 400
0072 DO 400 N=K+1,8
0073 IF(BUFER(K).EQ.BUFER(N)) BUFER(N)=-555

```

FORTRAN IV

V01C-03G+ MON 21-MAY-79 02:39:54

PAGE 004

```

0075 400  CONTINUE
      C*****
      C
      C$  IF THE BUFFER HAS A PIXEL BELONGING TO THE CURRENTLY
      C$  PROCESSED REGION, SKIP THIS PIXEL.
      C
      C$  COMPUTE TO HOW MANY REGIONS ARE THE CURRENTLY DETECTED
      C$  BOUNDARY PIXEL ADJACENT.
      C
0076      IADJ=0
0077      DO 500 K=1,8
0078      IF(BUFER(K).GE. -128. AND. BUFER(K).NE. XREG) IADJ=IADJ+1
0080 500  CONTINUE
      C*****
      C
      C$  IF THE CURRENTLY DETECTED BOUNDARY PIXEL IS ADJACENT TO
      C$  MORE THAN ONE REGION, IT IS PROBABLY THE END POINT
      C$  OF THE ADJACENT EDGE.
      C
      C$  CHANGE ITS LOCATION TO NEGATIVE VALUE.
      C
0081      IX=J
0082      IY=I
0083      IF(IADJ.EQ.1) GO TO 800
0085      IX=-IX
0086      IY=-IY
      C*****
      C
      C$  PUT THE CURRENTLY DETECTED BOUNDARY PIXEL INTO THE CIRCULAR-
      C$  LINKED LISTS OF THE CORRESPONDING ADJACENT REGIONS.
      C
      C$  GO TO DETECT THE NEXT BOUNDARY PIXEL.
      C
0087 800  DO 1200 K=1,8
0088      IF(BUFER(K).LE. -128. OR. BUFER(K).EQ. XREG) GO TO 1200
0090      IREG=BUFER(K)+129
0091      IF(TEMPNR(IREG).GT.0) GO TO 1000
0093      TEMPNR(IREG)=IWPNTR
0094 1000  WKSP2(IWPNTR)=WKSP2(TEMPNR(IREG))
0095      WKSP1(XCOD,IWPNTR)=IX
0096      WKSP1(YCOD,IWPNTR)=IY
0097      WKSP2(TEMPNR(IREG))=IWPNTR
0098      TEMPNR(IREG)=IWPNTR
0099      IWPNTR=IWPNTR+1
0100 1200  CONTINUE
      C*****
      C
      C$  TRANSFER THE EDGE PIXELS OF THE CIRCULAR-LINKED LISTS IN
      C$  THE WORKING SPACE TO THE CORRESPONDING SEQUENTIAL LISTS
      C$  OF THE OUTPUT DATA BASE.
      C
      C$  WHEN THE BUFFER OF THE OUTPUT DATA BASE IS FULL, OUTPUT
      C$  THE DATA IN THE BUFFER FIRST.
      C

```

FORTRAN IV VOIC-03G+ MON 21-MAY-79 02:39:54

PAGE 005

```

0101      DO 1800 IREG=1,MAXREG
0102      IF(TEMPNR(IREG).LT.0) GO TO 1800
0104      IF(TABPNR.GT.896) GO TO 2400
0106      IF(REGPTR(KREG).GT.0) GO TO 1300
0108      REGPTR(KREG)=TABPNR
0109 1300    ADJTAB(ADJREG,TABPNR)=IREG
0110      ADJTAB(PNTR,TABPNR)=2048*NTIME+LSTPNR
          C
0111      D      WRITE(6,1309) REGPTR(KREG),ADJTAB(ADJREG,TABPNR),
          D      - ADJTAB(PNTR,TABPNR),KREG
0112  D1309  FORMAT('0*** REGPTR = ',I6,5X,' ADJREG = ',I4,5X,
          D      - ' LSTPNR = ',I6,5X,' REGION ',I3)
          C
0113      TABPNR=TABPNR+1
0114      IFLAG=0
0115      IWPNTR=TEMPNR(IREG)
0116      ISTART=IWPNTR
0117 1400    IF(LSTPNR.LE.2048) GO TO 1500
0119 1450    IF(NOBLK.GE.32) GO TO 2600
0121      IF(IWRITE(2048,ADJPXL,NOBLK,ICH1).NE.2048) STOP '? BAD WRITE-1'
0123      NOBLK=NOBLK+8
0124      NTIME=NTIME+1
0125      LSTPNR=1
0126      IF(IFLAG.EQ.1) GO TO 1600
0128 1500    ADJPXL(XCOD,LSTPNR)=WKSP1(XCOD,IWPNTR)
0129      ADJPXL(YCOD,LSTPNR)=WKSP1(YCOD,IWPNTR)
          C
0130      D      WRITE(6,1509) ADJPXL(XCOD,LSTPNR),ADJPXL(YCOD,LSTPNR),LSTPNR
0131  D1509  FORMAT(20X,'XCOD = ',I4,5X,' YCOD = ',I4,5X,' ORDER = ',I6)
          C
0132      LSTPNR=LSTPNR+1
0133      IWPNTR=WKSP2(IWPNTR)
0134      IF(IWPNTR.NE.ISTART) GO TO 1400
0136      IF(LSTPNR.LE.2048) GO TO 1600
0138      IFLAG=1
0139      GO TO 1450
          C*****
          C
          C$      CLOSE THE SEQUENTIAL LIST OF THE ADJACENT EDGE PIXELS.
          C
0140 1600    ADJPXL(XCOD,LSTPNR)=0
0141      ADJPXL(YCOD,LSTPNR)=0
0142      LSTPNR=LSTPNR+1
0143 1800    CONTINUE
          C*****
          C
          C$      CLOSE THE SEQUENTIAL LIST IN THE ADJACENCY TABLE.
          C
0144      IF(TABPNR.GT.896) GO TO 2400
0146      ADJTAB(ADJREG,TABPNR)=-5
0147      ADJTAB(PNTR,TABPNR)=-5
0148      TABPNR=TABPNR+1
0149 2000    CONTINUE
          C*****

```

FORTRAN IV V01C-03G+ MON 21-MAY-79 02:39:54

PAGE 006

```
      C
      C*  OUTPUT THE BOUNDARY ADJACENCY DATA BASE.
      C
0150      IF(IWRITW(2048,ADJPXL,NOBLK,ICH1),NE.2048) STOP '? BAD WRITE-2'
0152      IF(IWRITW(256,REGPTR,39,ICH1),NE.256) STOP '? BAD WWRITE-3'
0154      IF(IWRITW(1792,ADJTAB,32,ICH1),NE.1792) STOP '? BAD WRITE-4'
0156      CALL CLOSEC(ICH1)
0157      STOP 'NORMAL END EXECUTION'
0158 2200 STOP '?? ERROR -- NO MORE WORKING SPACE'
0159 2400 STOP '?? ERROR -- NO MORE TABLE SPACE'
0160 2600 STOP '?? ERROR -- NO MORE DATA SPACE'
0161      END
```

FORTRAN IV

V01C-036+ MON 21-MAY-79 03:52:08

PAGE 001

```

C                                     *** SEQEDG.FOR ***
C
C*****
C* SUBJECT: ADJACENT EDGE SEQUENCING ALGORITHM. *
C* PROGRAMMER: CHIEN-CHYUN LEE. *
C* DATE: OCTOBER 22, 1978. *
C* INPUT: MUST BE "<*.DAT,*.DAT". *
C*      1. THE FIRST INPUT FILE SHOULD BE THE DATA FILE *
C*      GENERATED FROM THE ADJDCN.SAV PROGRAM. (REFER TO *
C*      ADJDCN.FOR). THE LENGTH OF THIS DATA FILE IS *
C*      40 BLOCKS. *
C*      2. THE SECOND INPUT FILE IS A DATA FILE GENERATED *
C*      FROM BNDSEQ.SAV PROGRAM. (REFER TO BNDSEQ.FOR). *
C*      THE LENGTH OF THIS DATA FILE IS VARIABLE. *
C* OUTPUT: FILENAME MUST BE "/*.DAT". 40 BLOCK LONG DATA *
C*      FILE. *
C*      1. BLOCK 0 -- BLOCK 31: SEQUENTIAL LIST FOR *
C*      SEQUENCED ADJACENT EDGE PIXELS. ZERO VALUE *
C*      REPRESENTS END MARK. *
C*      2. BLOCK 32 -- BLOCK 38: SEQUENTIAL REGION *
C*      ADJACENCY TABLE. NEGATIVE VALUES REPRESENT *
C*      MARKS. *
C*      3. BLOCK 39: INTEGER*2 REGPTR(256) ARRAY FOR *
C*      REGION POINTER. *
C* %% NOTE: *
C*      1. EACH ELEMENT IN THE ADJACENCY TABLE HAS TWO *
C*      INTEGER*2 WORDS. FIRST WORD STORES A NUMBER FOR *
C*      ADJACENT REGION. SECOND WORD IS A POINTER TO THE *
C*      SEQUENTIAL LIST. IF THE ADJACENT EDGE HAS ONE *
C*      MORE DISCONNECTED SECTION, EACH DISCONNECTED *
C*      SECTION WILL HAVE ITS OWN CORRESPONDING ELEMENT *
C*      IN THE ADJACENCY TABLE. *
C*      2. EACH ELEMENT IN THE SEQUENTIAL LIST IS A INTEGER*2 *
C*      WORD STORING ONE-DIMENSIONAL LOCATION OF THE *
C*      PIXEL. *
C*      3. COMPILING WITH SWITCH "/D" WILL PRINT OUT THE *
C*      OUTPUT DATA BASE. *
C*****
C
C
C
C$ "REGPTR" IS THE REGION POINTER ARRAY OF THE DATA BASE
C$ GENERATED FROM ADJDCN.FOR.
C
C$ "REGPNR" IS THE REGION POINTER ARRAY OF THE DATA BASE
C$ GENERATED FROM BNDSEQ.FOR. IT IS ALSO THE REGION POINTER
C$ ARRAY OF THE OUTPUT DATA BASE OF THIS PROGRAM.
C
C$ "ADJTAB" IS THE ADJACENCY TABLE GENERATED FROM ADJDCN.FOR.
C
C$ "ADJTBL" IS THE ADJACENCY TABLE OF THE OUTPUT DATA BASE
C$ OF THIS PROGRAM.
C
C$ "SEQPXL" IS AN ARRAY FOR THE SEQUENCED BOUNDARY PIXELS

```

FORTRAN IV

V01C-03G+ MON 21-MAY-79 02:43:20

PAGE 002

```

C* FROM ENDSEQ. FOR.
C
C* "SEQLST" IS AN ARRAY FOR SEQUENCED ADJACENT EDGE PIXELS.
C
C* "ADJPXL" IS AN ARRAY FOR THE UNSEQUENCED ADJACENT EDGE
C* PIXELS FROM ADJDCN. FOR.
C
0001 LOGICAL*1 ADJREG, PNTR, XCOD, YCOD, ADJPXL(2, 8192)
0002 INTEGER*2 REGPTR(256), ADJTAB(2, 896), REGPNR(256), ADJTBL(2, 896)
0003 INTEGER*2 SEGPXL(2, 1536), SEQLST(8192), WORKSP(10240)
0004 INTEGER*2 FILSPC(39), DEFEXT(4)
0005 DATA DEFEXT/4*3RDAT/
0006 EQUIVALENCE (WORKSP(1), ADJPXL(1, 1)), (WORKSP(8193), ADJTAB(1, 1))
0007 EQUIVALENCE (WORKSP(9985), REGPTR(1))
0008 COMMON WORKSP, REGPNR, SEGPXL, SEQLST, ADJTBL
C*****
C
C* INPUT AND OUTPUT ARRANGEMENTS.
C
0009 CALL PRINT('ENTER COMMAND STRING---*. DAT<*. DAT, *. DAT')
0010 CALL ICSI(FILSPC, DEFEXT, , , 0)
0011 IX=IFETCH(FILSPC(1))
0012 IY=IFETCH(FILSPC(16))
0013 IZ=IFETCH(FILSPC(20))
0014 IF((IX+IY+IZ).NE.0) STOP '?BAD COMMAND STRING'
0015 ICH1=IGETC()
0016 ICH2=IGETC()
0017 ICH3=IGETC()
0018 IF(ICH1.LT.0.OR.ICH2.LT.0.OR.ICH3.LT.0) STOP '? NO CHANNEL
- AVAILABLE'
0021 IF(IENTER(ICH1, FILSPC(1), 40).NE.40) STOP '? NO SPACE OUT-1'
0023 IF(LOOKUP(ICH2, FILSPC(16), 40).NE.40) STOP '? NO SUCH FILE-1'
0025 MAXBLK=LOOKUP(ICH3, FILSPC(20))
0026 IF(MAXBLK.LT.0) STOP '? NO SUCH FILE-2'
0028 IF(IREADW(10240, WORKSP, 0, ICH2).NE.10240) STOP '? BAD READ-1'
0030 IF(IREADW(256, REGPNR, 0, ICH3).NE.256) STOP '? BAD READ-2'
C*****
C
C* "ADJREG" STANDS FOR ADJACENT REGION IN THE ADJACENCY TABLE.
C
C* "PNTR" STANDS FOR THE POINTER IN THE ADJACENCY TABLE.
C
C* "ISPNTN" IS THE MEMORY CURSOR ASSOCIATED WITH
C* "SEQLST" ARRAY.
C
C* "ITPNTR" IS THE MEMORY CURSOR ASSOCIATED WITH
C* "ADJTBL" ARRAY.
C
0032 ADJREG=1
0033 PNTR=2
0034 INFO=1
0035 LINK=2
0036 XCOD=1
0037 YCOD=2

```

FORTRAN IV VOIC-03G+ MON 21-MAY-79 02:43:20

PAGE 003

```

0038      ISPNTR=0
0039      IIPNTR=0
      C*****
      C
      C$    THE SEQUENCING OPERATION IS PROCESSED REGION BY REGION.
      C
      C$    INPUT ALL THE SEQUENCED BOUNDARY PIXELS OF A REGION.
      C
0040      DO 3000 KREG=1,256
0041      IF(REGPNR(KREG).LT.0) GO TO 3000
0043      TYPE59,KREG
0044      D    WRITE(6,59) KREG
0045      59   FORMAT(1H0,10X,'***** DOING REGION ',I4)
0046      NOBLK=REGPNR(KREG)
0047      LBLK=MAXBLK-NOBLK
0048      IF(LBLK.GT.12) LBLK=12
0050      NOWORD=LBLK*256
0051      IF(IREADW(NOWORD,SEQPXL,NOBLK,ICH3).NE.NOWORD)
      - STOP '? BAD READ-3'
      C*****
      C
      C$    "REGPTR" IS INITIALIZED FOR THE USED OF THE OUTPUT.
      C
      C$    "KTPNTR" IS THE MEMORY CURSOR OF THE INPUT ADJACENCY TABLE.
      C
      C$    THE KEY STEP OF THIS PROGRAM IS TO FIND THE FIRST STARTING
      C$    END POINT OF THE ADJACENT EDGE.
      C
0053      REGPNR(KREG)=-5
0054      IPPOTN=1
0055      ICPOTN=1
0056      KTPNTR=REGPTR(KREG)
0057      ISTART=ICPOTN
      C*****
      C
      C$    THE CANDIDATES OF THE FIRST STARTING END POINT OF THE ADJACENT
      C$    EDGE ARE STORED IN "ADJPXL" ARRAY.
      C
      C$    TRACE THE SEQUENCED BOUNDARY PIXELS TO FIND THE CANDIDATE.
      C
      C$    "IFLAG" IS INITIALIZED TO ZERO TO INDICATE THE CANDIDATES
      C$    HAVE NOT YET HAD RIGHT TO BE ELECTED AS A STARTING END
      C$    POINT OF THE ADJACENT EDGE.
      C
      C$    "JOPTLN" KEEPS THE MEMORY LOCATION OF THE LAST CANDIDATE.
      C$    IT IS SET TO 0 BEFORE THE TRACING OPERATION IS STARTED.
      C
0058      500  IDOREG=ADJTAB(ADJREG,KTPNTR)
0059      IF(IDOREG.LT.0) GO TO 2300
0061      IFLAG=0
0062      JOPTLN=0
0063      TYPE509,IDOREG
0064      509  FORMAT(1H0,5X,'ADJACENT REGION',I4)
0065      600  KAPNTR=ADJTAB(PNTR,KTPNTR)

```

FORTRAN IV V01C-03G+ MON 21-MAY-79 02:43:20

PAGE 004

```

0066      LOCATN=SEQPXL(INFO,ICPOTN)
0067 800   IX=ADJPXL(XCOD,KAPNTR)
0068      IF(IX.EQ.0) GO TO 1000
0070      IY=ADJPXL(YCOD,KAPNTR)
0071      IXX=IABS(IX)
0072      IYY=IABS(IY)
0073      LOCPXL=(IYY-1)*128+IXX
0074      IF(LOCPXL.EQ.LOCATN) GO TO 1200
0076      KAPNTR=KAPNTR+1
0077      GO TO 800
C*****
C
C$   IF A NON-CANDIDATE SEQUENCED BOUNDARY PIXEL IS
C$   TRACED, THEN ALL THE CANDIDATES START TO HAVE
C$   RIGHT TO BE ELECTED.
C
C$   "NEXSAT" KEEPS THE MEMORY LOCATION OF THE LAST
C$   NON-CANDIDATE PIXEL IN THE SEQUENCED CIRCULAR-
C$   LINKED LIST.
C
0078 1000  IF(IFLAG.EQ.0) NEXSAT=ICPOTN
0080      IFLAG=1
0081 900   IPPOTN=ICPOTN
0082      ICPOTN=SEQPXL(LINK,ICPOTN)
0083      IF(ICPOTN.NE.ISTART) GO TO 600
0085      ISTART=NEXSAT
0086      IPPOTN=NEXSAT
0087      ICPOTN=SEQPXL(LINK,NEXSAT)
0088      JOPTLN=0
0089      GO TO 600
C*****
C
C$   THE CANDIDATE IS ELECTED IN THE FOLLOWING WAYS.
C
0090 1200  IF(IFLAG.EQ.1.AND.JOPTLN.EQ.0) GO TO 1600
0092      IF(IFLAG.EQ.0.AND.JOPTLN.NE.0) GO TO 1250
0094      IF(IFLAG.EQ.0.AND.JOPTLN.EQ.0) GO TO 1100
0096      JDIFER=IABS(JOPTLN-LOCATN)
0097      IF(JDIFER.NE.1.AND.JDIFER.NE.127.AND.JDIFER.NE.128.AND.
- JDIFER.NE.129) GO TO 1600
0099      NEXSAT=ICPOTN
0100      JOPTLN=LOCATN
0101      IFLAG=0
0102      GO TO 900
0103 1250  JDIFER=IABS(JOPTLN-LOCATN)
0104      IF(JDIFER.NE.1.AND.JDIFER.NE.127.AND.JDIFER.NE.128.AND.
- JDIFER.NE.129) GO TO 1600
0106 1100  JOPTLN=LOCATN
0107      IPPOTN=ICPOTN
0108      ICPOTN=SEQPXL(LINK,ICPOTN)
0109      IF(ICPOTN.NE.ISTART) GO TO 600
0111      LOCATN=SEQPXL(INFO,ICPOTN)
C*****
C

```

FORTRAN IV

V01C-03G+ MON 21-MAY-79 02:43:20

PAGE 005

```

C*   WHEN THE FIRST STARTING END POINT OF THE ADJACENT EDGE
C*   IS DETERMINED, THE ADJACENCY INFORMATION OF THE
C*   SEQUENCED ADJACENT EDGE IS FIRST STORED IN THE
C*   ADJACENCY TABLE OF THE OUTPUT DATA BASE.
C
C*   THE STARTING END POINT IS SECOND STORED IN THE
C*   SEQUENTIAL LIST OF THE OUTPUT DATA BASE.
C
0112 1600 INNER=0
0113      ISTART=ICPOTN
0114      ITPNTR=ITPNTR+1
0115      IF(ITPNTR.GT.896) STOP '? OUT OF TABLE, > 896'
0117      IF(REGPNR(KREG).LT.0) REGPNR(KREG)=ITPNTR
0119      ADJTBL(ADJREG,ITPNTR)=IDOREG
0120      ISPNTN=ISPNTN+1
0121      IF(ISPNTN.GT.8192) STOP '? OUT OF LIST SPACE, > 8192'
0123      ADJTBL(PNTR,ITPNTR)=ISPNTN
0124      SEQLST(ISPNTN)=LOCATN
0125      JOPTLN=LOCATN
C*****
C
C*   FROM THE STARTING END POINT, PUT ALL THE CANDIDATES
C*   INTO THE SEQUENTIAL LIST.
C
C*   THE POSSIBILITY OF HAVING THE SEPARABLE PARTS OF THE
C*   ADJACENT EDGE IS CONSIDERED.
C
C*   THE FOLLOWING COMANDS JUST DETECT WHICH SEQUENCED PIXEL
C*   IS A CANDIDATE.
C
0126 1650 JFLAG=0
0127 1700 IPPOTN=ICPOTN
0128      ICPOTN=SEQPXL(LINK,ICPOTN)
0129 1800 IF(ICPOTN.EQ.ISTART) GO TO 2500
0131      JCPTLN=SEQPXL(INFO,ICPOTN)
0132      KAPNTR=ADJTAB(PNTR,KTPNTR)
0133 1900 IX=ADJPXL(XCOD,KAPNTR)
0134      IF(IX.NE.0) GO TO 1950
0136      JFLAG=1
0137      GO TO 1700
0138 1950 IY=ADJPXL(YCOD,KAPNTR)
0139      IXX=IABS(IX)
0140      IYY=IABS(IY)
0141      LOCPXL=(IYY-1)*128+IXX
0142      IF(LOCPXL.EQ.JCPTLN) GO TO 2000
0144      KAPNTR=KAPNTR+1
0145      GO TO 1900
C*****
C
C*   CHECK IF SEPARABLE PART OF THE ADJACENT EDGE EXISTS.
C
0146 2000 IF(JFLAG.EQ.0) GO TO 2050
0148      JDIFER=IABS(JCPTLN-JOPTLN)
0149      IF(JDIFER.NE.1.AND.JDIFER.NE.127.AND.JDIFER.NE.128.AND.

```

FORTRAN IV

V01C-03G+ MON 21-MAY-79 02:43:20

PAGE 006

```

      - JDIFER. NE. 129) GO TO 2325
0151      JFLAG=0
0152 2050 IF(IX. LT. 0) GO TO 2200
0154      IF(INNER. LE. 2) GO TO 2100
0156      ICPOTN=SEQPXL(LINK, ICPOTN)
0157      SEQPXL(LINK, IPPOTN)=ICPOTN
0158      GO TO 2300
      C*****
      C
      C* IF A SEPARABLE PART OF THE ADJACENT EDGE EXISTS,
      C* CLOSE THE CURRENT SEQUENTIAL LIST AND GENERATE
      C* ANOTHER LIST FOR NEXT PART OF THE ADJACENT EDGE;
      C* OTHERWISE, PUT THE CURRENTLY DETECTED CANDIDATE
      C* INTO THE LIST.
      C
0159 2100 INNER=INNER+1
0160 2200 IPPOTN=ICPOTN
0161      ICPOTN=SEQPXL(LINK, ICPOTN)
0162 2300 JDIFER=IABS(JOPTLN-JCPTLN)
0163      IF(JDIFER. EQ. 1. OR. JDIFER. EQ. 127. OR. JDIFER. EQ. 128. OR.
      - JDIFER. EQ. 129) GO TO 2400
0165 2325 IF(SEQLST(ISPNTR). EQ. 0) GO TO 2350
0167      ISPNTR=ISPNTR+1
0168      IF(ISPNTR. GT. 8192) STOP '? OUT OF LIST SPACE, > 8192'
0170      SEQLST(ISPNTR)=0
0171 2350 ISPNTR=ISPNTR+1
0172      IF(ISPNTR. GT. 8192) STOP '? OUT OF LIST SPACE, >8192'
0174      SEQLST(ISPNTR)=JCPTLN
0175      JOPTLN=JCPTLN
0176      ITPNTR=ITPNTR+1
0177      IF(ITPNTR. GT. 896) STOP '? OUT OF TABLE, > 896'
0179      ADJTBL(ADJREG, ITPNTR)=IDOREG
0180      ADJTBL(PNTR, ITPNTR)=ISPNTR
0181      IF(JFLAG. EQ. 1) GO TO 1650
0183      GO TO 1800
0184 2400 ISPNTR=ISPNTR+1
0185      IF(ISPNTR. GT. 8192) STOP '? OUT OF LIST SPACE, > 8192'
0187      SEQLST(ISPNTR)=JCPTLN
0188      JOPTLN=JCPTLN
0189      GO TO 1800
      C*****
      C
      C* CLOSE THE SEQUENTIAL LIST OF THE SEQUENCED EDGE PIXELS.
      C
      C* GO TO SEQUENCE NEXT ADJACENT EDGE.
      C
0190 2500 IF(SEQLST(ISPNTR). EQ. 0) GO TO 2550
0192      ISPNTR=ISPNTR+1
0193      IF(ISPNTR. GT. 8192) STOP '? OUT OF LIST SPACE, > 8192'
0195      SEQLST(ISPNTR)=0
0196 2550 KTPNTR=KTPNTR+1
0197      GO TO 500
      C*****
      C

```

FORTRAN IV

V01C-03G+ MON 21-MAY-79 02:43:20

PAGE 007

```

      C*   CLOSE THE SEQUENTIAL LIST IN THE ADJACENCY TABLE.
      C
      C*   GO TO PROCESS NEXT REGION.
      C
0198 2800 ITPNTR=11PNTR+1
0199      IF(ITPNTR.GT.896) STOP '? OUT OF TABLE, > 896'
0201      ADJTBL(ADJREG,ITPNTR)=-5
0202      ADJTBL(PNTR,ITPNTR)=-5
      C*****
      C
      C*   PRINT OUT THE OUTPUT DATA BASE.
      C
0203 D      MTPNTR=REGPNR(KREG)
0204 D3100 IDOREG=ADJTBL(ADJREG,MTPNTR)
0205 D      IF(IDOREG.LT.0) GO TO 3000
0207 D      WRITE(6,3109) IDOREG
0208 D3109 FORMAT(1H0,5X,'#### ADJACENT REGION',I4)
0209 D      MSPNTR=ADJTBL(PNTR,MTPNTR)
0210 D3200 LOCATN=SEQLIST(MSPNTR)
0211 D      IF(LOCATN.EQ.0) GO TO 3300
0213 D      IX=MOD(LOCATN,128)
0214 D      IF(IX.EQ.0) IX=128
0216 D      IY=(LOCATN-IX)/128+1
0217 D      WRITE(6,3209) IX,IY
0218 D3209 FORMAT(' XCOD =',I4,5X,'YCOD =',I4)
0219 D      MSPNTR=MSPNTR+1
0220 D      GO TO 3200
0221 D3300 MTPNTR=MTPNTR+1
0222 D      GO TO 3100
      C*****
0223 3000 CONTINUE
0224      NOBYTE=ISPNTN*2
0225      TYPE3009,ISPNTN,NOBYTE
0226 3009 FORMAT(1H0,'*** TOTALLY',I6,' WORDS ----> ',I6,' BYTES')
      C*****
      C
      C*   OUTPUT THE SEQUENCED BOUNDARY ADJACENCY DATA BASE.
      C
0227      IF(IWRITW(8192,SEQLIST,0,ICH1).NE.8192) STOP '? BAD WRITE-1'
0229      IF(IWRITW(1792,ADJTBL,32,ICH1).NE.1792) STOP '? BAD WRITE-2'
0231      IF(IWRITW(256,REGPNR,39,ICH1).NE.256) STOP '? BAD WRITE-3'
0233      CALL CLOSEC(ICH1)
0234      STOP 'NORMAL END EXECUTION'
0235      END

```

FORTRAN IV

VOIC-03G+ MON 21-MAY-79 02:47:27

PAGE 001

```

C          *** RCACOD.FOR ***
C
C*****
C* SUBJECT:  FIRST CODING SCHEME FOR COMPRESSION.
C* PROGRAMMER:  CHIEN-CHYUN LEE.
C* DATE:  NOVEMBER 14, 1979.
C* INPUT:  FILENAME MUST BE *.DAT.
C*          THIS INPUT DATA FILE IS THE OUTPUT DATA FILE
C*          GENERATED FROM THE SEQEDG.SAV PROGRAM. (REFER TO
C*          SEQEDG.FOR). THE LENGTH OF THIS INPUT DATA FILE IS
C*          40 BLOCK LONG.
C* OUTPUT:  FILENAME MUST BE *.DAT.
C*          THE LENGTH OF THIS OUTPUT DATA FILE IS 13 BLOCKS.
C*          1. BLOCK 0 -- BLOCK 4: LOGICAL*1 DIRVCR(2560)
C*             ARRAY FOR SEQUENTIAL LIST TO STORE DIRECTION
C*             VECTOR CODE. VALUE -128 STANDS FOR END MARK OF
C*             EACH SEQUENTIAL LIST.
C*          2. BLOCK 5 -- BLOCK 11: INTEGER*2 CHANTB(2,896)
C*             ARRAY FOR CHAIN CODE TABLE (BY SEQUENTIAL LIST
C*             STRUCTURE). NEGATIVE VALUE STANDS FOR END MARK
C*             OF EACH SEQUENTIAL LIST. EACH NODE IN THE LIST
C*             HAS TWO WORDS. FIRST WORD STORES A NUMBER FOR
C*             ADJACENT REGION. SECOND WORD STORES THE POINTER
C*             TO ITS CORRESPONDING SEQUENTIAL LIST FOR DIREC-
C*             TION VECTOR CODE.
C*          3. BLOCK 12: INTEGER*2 REGPTR(256) ARRAY FOR REGION
C*             POINTER.
C* NOTE:  COMPILING WITH SWITCH "/D" WILL PRINT OUT THE
C*         OUTPUT DATA BASE.
C*****
C
C
C
C* "DIRVCR" IS THE DIRECTION VECTOR CODE ARRAY FOR CODE LIST.
C
C* "REGPTR" IS REGION POINTER ARRAY.
C
C* "ADJTBL" ARRAY IS THE ADJACENCY TABLE OF THE INPUT DATA BASE.
C
C* "SEQLIST" ARRAY STORES THE SEQUENTIAL LIST OF THE SEQUENCED
C*   ADJACENT EDGE.
C
C* "CHANTB" ARRAY IS THE ADJACENCY TABLE OF THE OUTPUT DATA BASE.
C
0001      LOGICAL*1 DIRVCR(2560)
0002      INTEGER*2 REGPTR(256), ADJTBL(2,896), SEQLST(8192), DIRCD1
0003      INTEGER*2 FILSPC(39), DEFEXT(4), DIRCOD, CODE, CHANTB(2,896)
0004      DATA DEFEXT/4*3RDAT/
0005      EQUIVALENCE (ADJTBL(1,1), CHANTB(1,1))
C*****
C
C* INPUT AND OUTPUT ARRANGEMENTS.
C
0006      CALL PRINT('ENTER COMMAND STRING -- *.DAT<*.DAT')

```

FORTRAN IV

V01C-036+ MON 21-MAY-79 02:47:27

PAGE 002

```

0007      CALL ICSI(FILSPC,DEFEXT,,0)
0008      IX=IFETCH(FILSPC(1))
0009      IY=IFETCH(FILSPC(16))
0010      IF((IX+IY).NE.0) STOP '? BAD COMMAND STRING'
0012      ICH1=IGETC()
0013      ICH2=IGETC()
0014      IF(ICH1.LT.0.OR.ICH2.LT.0) STOP '? NO CHANNEL AVAILABLE'
0016      IF(1ENTER(ICH1,FILSPC(1),13).NE.13) STOP '? NO SPACE OUT'
0018      IF(LOOKUP(ICH2,FILSPC(16),40).NE.40) STOP '? NO SUCH FILE'
0020      IF(IREADW(8192,SEQLST,0,ICH2).NE.8192) STOP '? BAD READ-1'
0022      IF(IREADW(1792,ADJTBL,32,ICH2).NE.1792) STOP '? BAD READ-2'
0024      IF(IREADW(256,REGPTR,39,ICH2).NE.256) STOP '? BAD READ-3'
C*****
C
C$  "ADJREG" STANDS FOR ADJACENT REGION.
C
C$  "PNTR" STANDS FOR POINTER IN THE ADJACENCY TABLE.
C
C$  "NCOD" KEEPS THE NUMBER OF THE CODE IN THE CODE LIST.
C
0026      ADJREG=1
0027      PNTR=2
0028      IDPNTR=0
0029      NCOD=0
C*****
C
C$  THE CODING OPERATION IS PROCESSED REGION BY REGION,
C$  EDGE BY EDGE.
C
C$  "DIRCD1" IS THE DIRECTION CODE OF THE FIRST ONE OF THE TWO
C$  SUCCESSIVE PIXELS.
C
C$  "ISPNTR" IS THE MEMORY CURSOR IN THE "SEQLST" ARRAY.
C
C$  "IDPNTR" IS THE MEMORY CURSOR IN THE "DIRVCR" ARRAY.
C
C$  "IAPNTR" IS THE MEMORY CURSOR IN THE "ADJTBL" ARRAY
C
C$  ONCE IT STARTS TO CODE THE NEXT TWO SUCCESSIVE PIXELS,
C$  "DIRCD1" MUST BE INITIALIZED.
C
C$  ONCE IT STARTS TO CODE AN ADJACENT EDGE, THE X AND THE Y
C$  COORDINATE LOCATIONS OF THE STARTING END POINT OF THIS
C$  ADJACENT EDGE MUST BE PUT INTO THE FIRST AND THE SECOND
C$  ELEMENTS OF THE CORRESPONDING CODE LIST RESPECTIVELY.
C
0030      DO 2000 KREG=1,256
0031      IF(REGPTR(KREG).LT.0) GO TO 2000
0033      IAPNTR=REGPTR(KREG)
0034      DIRCD1=-5
0035 200  IDOREG=ADJTBL(ADJREG,IAPNTR)
0036      IF(IDOREG.LT.0) GO TO 2000
0038      TYPE109,IDOREG,KREG
0039 109  FORMAT(1H0,5X,'DOING REGION',I4,' ----->  REGION',I4)

```

FORTRAN IV

V01C-03G+ MON 21-MAY-79 02:47:27

PAGE 003

```

0040      ISPNTR=ADJTB(L(PNTR, IAPNTR)
0041      LOCATN=SEQLST(ISPNTR)
0042      IX=MOD(LOCATN, 128)
0043      IF(IX.EQ.0) IX=128
0045      IY=(LOCATN-IX)/128+1
0046      IDPNTR=IDPNTR+1
0047      IF((IDPNTR+1).GT.2560) STOP '? NO MORE SPACE FOR CODE. > 2560'
0049      IHEAD=IDPNTR
0050      DIRVCR(IDPNTR)=IX
0051      IDPNTR=IDPNTR+1
0052      DIRVCR(IDPNTR)=IY
0053      CHANTB(PNTR, IAPNTR)=IHEAD
      C
0054      D      WRITE(6,209) IX,IY,KREG, IDOREG, IAPNTR, IHEAD
0055      D209  FORMAT('O ***** START AT XCOD=',I4,', YCOD=',I4,',', ' REGION',
      D      -  I4,', -----> REGION',I4,',,10X,' IAPNTR=',I6,5X,' IHEAD=',I6,/)
      C
      C*****
      C
      C$      DETERMINE THE DIRECTION CODE OF THE CURRENTLY TRACED
      C$      PIXEL.
      C
      C$      IF THE FIRST DIRECTION CODE OF THE DIRECTION VECTORS OF
      C$      THE TWO SUCCESSIVE PIXELS HAS BEEN DETERMINED, GO TO
      C$      CODE THESE TWO SUCCESSIVE PIXELS.
      C
0056      400  JOPTLN=LOCATN
0057      ISPNTR=ISPNTR+1
0058      LOCATN=SEQLST(ISPNTR)
0059      IF(LOCATN.EQ.0) GO TO 900
0061      JDIFER=LOCATN-JOPTLN
0062      500  DIRCOD=9
0063      IF(JDIFER.EQ.1) DIRCOD=0
0065      IF(JDIFER.EQ.129) DIRCOD=1
0067      IF(JDIFER.EQ.128) DIRCOD=2
0069      IF(JDIFER.EQ.127) DIRCOD=3
0071      IF(JDIFER.EQ.-1) DIRCOD=4
0073      IF(JDIFER.EQ.-129) DIRCOD=5
0075      IF(JDIFER.EQ.-128) DIRCOD=6
0077      IF(JDIFER.EQ.-127) DIRCOD=7
0079      IF(DIRCOD.EQ.9) STOP '? BAD INPUT DATA'
0081      IF(DIRCD1.NE.-5) GO TO 700
0083      DIRCD1=DIRCOD
0084      GO TO 400
      C*****
      C
      C$      CODE THE TWO SUCCESSIVE DIRECTION CODES IN THE FOLLOWING
      C$      WAY AND PUT THE CODE INTO THE CODE LIST.
      C
      C$      GO TO DETERMINE THE DIRECTION CODE OF THE TWO SUCCESSIVE
      C$      PIXELS.
      C
0085      700  CODE=(DIRCOD+1)*8+DIRCD1
0086      NCOD=NCOD+1

```

FORTRAN IV V01C-03G+ MON 21-MAY-79 02:47:27 PAGE 004

```

0087      IDPNTR=IDPNTR+1
0088      IF(IDPNTR.GT.2560) STOP '? NO MORE SPACE FOR CODE, > 2560'
0090      DIRVCR(IDPNTR)=CODE
      C
0091      D      WRITE(6,809) CODE, IDPNTR, DIRCD1, DIRCOD
0092      D809  FORMAT(5X, 'CODE =', I5, 5X, 'IDPNTR =', I6, 5X, '1ST DIRECTION =', I2,
      D      -   , 5X, 'NEXT DIRECTION =', I3)
      C
0093      DIRCD1=-5
0094      GO TO 400
      C*****
      C
      C$      PUT THE LAST CODE INTO THE CODE LIST AND CLOSE THE CODE LIST.
      C
      C$      GO TO CODE THE NEXT ADJACENT EDGE.
      C
0095  900      IF(DIRCD1.NE.-5) GO TO 1000
0097      CODE=-128
0098      IDPNTR=IDPNTR+1
0099      IF(IDPNTR.GT.2560) STOP '? NO MORE SPACE FOR CODE, > 2560'
0101      DIRVCR(IDPNTR)=CODE
      C
0102      D      WRITE(6,909) CODE, IDPNTR
0103      D909  FORMAT(5X, 'CODE =', I5, 5X, 'IDPNTR =', I6, 5X, 30('*'))
      C
0104      IAPNTR=IAPNTR+1
0105      GO TO 200
0106  1000     CODE=DIRCD1
0107      NCOD=NCOD+1
0108      IDPNTR=IDPNTR+1
0109      IF(IDPNTR.GT.2560) STOP '? NO MORE SPACE FOR CODE, > 2560'
0111      DIRVCR(IDPNTR)=CODE
      C
0112      D      WRITE(6,1009) CODE, IDPNTR, DIRCD1
0113      D1009 FORMAT(5X, 'CODE =', I5, 5X, 'IDPNTR =', I6, 5X, '1ST DIRECTION =',
      D      -   , I2, 5X, 10('*'))
      C
0114      DIRCD1=-5
0115      GO TO 900
0116  2000     CONTINUE
      C*****
0117      TYPE2009, IDPNTR, NCOD
0118  2009     FORMAT(1H0, 5X, '***==* CODE SEQUENTIAL LISTS TOTALLY', I6, ' BYTES',
      -   /15X, 'BUT REALLY JUST', I6, ' BYTES FOR CODE'//)
      C*****
      C
      C$      OUTPUT THE COMPRESSED DATA BASE.
      C
0119      IF(IWRITW(1280, DIRVCR, 0, ICH1).NE.1280) STOP '? BAD WRITE-1'
0121      IF(IWRITW(1024, CHANTB, 5, ICH1).NE.1024) STOP '? BAD WRITE-2'
0123      IF(IWRITW(256, REOPTR, 12, ICH1).NE.256) STOP '? BAD WRITE-3'
0125      CALL CLOSEC(ICH1)
0126      STOP 'NORMAL END EXECUTION'
0127      END

```

FORTRAN IV

V01C-03G+ MON 21-MAY-79 02:58:18

PAGE 001

```

C                      *** CHACOD.FOR ***
C
C*****
C* SUBJECT:  SECOND CODING SCHEME FOR COMPRESSION.      *
C* PROGRAMMER:  CHIEN-CHYUN LEE.                        *
C* DATE:  OCTOBER 25, 1978.                             *
C* INPUT:  FILENAME MUST BE *.DAT.                      *
C*          THIS INPUT DATA FILE IS THE OUTPUT DATA FILE *
C*          GENERATED FROM THE SEQEDG.SAV PROGRAM. (REFER TO *
C*          SEQEDG.FOR).  THE LENGTH OF THIS INPUT DATA FILE IS *
C*          40 BLOCK LONG.                               *
C* OUTPUT:  FILENAME MUST BE *.DAT.                     *
C*          THE LENGTH OF THIS OUTPUT DATA FILE IS 13 BLOCKS. *
C*          1. BLOCK 0 -- BLOCK 4: LOGICAL*1 DIRVCR(2560) *
C*             ARRAY FOR SEQUENTIAL LIST TO STORE DIRECTION *
C*             VECTOR CODE. ZERO VALUE STANDS FOR END MARK OF *
C*             EACH SEQUENTIAL LIST.                       *
C*          2. BLOCK 5 -- BLOCK 11: INTEGER*2 CHANTB(2,896) *
C*             ARRAY FOR CHAIN CODE TABLE (BY SEQUENTIAL LIST *
C*             STRUCTURE). NEGATIVE VALUE STANDS FOR END MARK *
C*             OF EACH SEQUENTIAL LIST. EACH NODE IN THE LIST *
C*             HAS TWO WORDS. FIRST WORD STORES A NUMBER FOR *
C*             ADJACENT REGION. SECOND WORD STORES THE POINTER *
C*             TO ITS CORRESPONDING SEQUENTIAL LIST FOR DIREC- *
C*             TION VECTOR CODE.                           *
C*          3. BLOCK 12: INTEGER*2 REGPTR(256) ARRAY FOR REGION *
C*             POINTER.                                     *
C* NOTE:  COMPILING WITH SWITCH "/D" WILL PRINT OUT THE *
C*        OUTPUT DATA BASE.                             *
C*****
C
C
C
C$ "DIRVCR" IS DIRECTION VECTOR CODE ARRAY FOR CODE LIST.
C
C$ "REGPTR" IS REGION POINTER ARRAY.
C
C$ "ADJTLB" ARRAY IS THE ADJACENCY TABLE OF THE INPUT DATA BASE.
C
C$ "SEQLST" ARRAY STORES THE SEQUENTIAL LIST OF THE SEQUENCED
C$ ADJACENT EDGE.
C
C$ "CHANTB" ARRAY IS THE ADJACENCY TABLE OF THE OUTPUT DATA BASE.
C
0001 LOGICAL*1 DIRVCR(2560),BUFER(2)
0002 INTEGER*2 REGPTR(256),ADJTLB(2,896),SEQLST(8192)
0003 INTEGER*2 FILSPC(39),DEFEXT(4),DIRCOD,CODE,CHANTB(2,896)
0004 DATA DEFEXT/4*3RDAT/
0005 EQUIVALENCE (ADJTLB(1,1),CHANTB(1,1))
C*****
C
C$ INPUT AND OUTPUT ARRANGEMENTS.
C
0006 CALL PRINT('ENTER COMMAND STRING -- *.DAT<*.DAT')

```

FORTRAN IV V01C-03G+ MON 21-MAY-79 02:58:18 PAGE 002

```

0007      CALL IC5I(FILSPC,DEFEXT,,,0)
0008      IX=IFETCH(FILSPC(1))
0009      IY=IFETCH(FILSPC(16))
0010      IF((IX+IY).NE.0) STOP '? BAD COMMAND STRING'
0012      ICH1=IGETC()
0013      ICH2=IGETC()
0014      IF(ICH1.LT.0.OR.ICH2.LT.0) STOP '? NO CHANNEL AVAILABLE'
0016      IF(IENTER(ICH1,FILSPC(1),13).NE.13) STOP '? NO SPACE OUT'
0018      IF(LOOKUP(ICH2,FILSPC(16),40).NE.40) STOP '? NO SUCH FILE'
0020      IF(IREADW(8192,SEQLST,0,ICH2).NE.8192) STOP '? BAD READ-1'
0022      IF(IREADW(1792,ADJTBL,32,ICH2).NE.1792) STOP '? BAD READ-2'
0024      IF(IREADW(256,REGPTR,39,ICH2).NE.256) STOP '? BAD READ-3'
C*****
C
C$      "ADJREG" STANDS FOR ADJACENT REGION.
C
C$      "PNTR" STANDS FOR POINTER IN THE ADJACENCY TABLE.
C
C$      "NCOD" KEEPS THE NUMBER OF THE CODE IN THE CODE LIST.
C
0026      ADJREG=1
0027      PNTR=2
0028      IDPNTR=0
0029      KPXLN=0
0030      NCOD=0
C*****
C
C$      THE CODE OPERATION IS PROCESSED REGION BY REGION,
C$      EDGE BY EDGE.
C
C$      "BUFER(1)" KEEPS THE DIRECTION CODE OF THE DIRECTION VECTOR.
C
C$      "BUFER(2)" KEEPS THE LENGTH OF THE DIRECTION VECTOR.
C
C$      "ISPNTN" IS THE MEMORY CURSOR IN THE "SEQLST" ARRAY.
C
C$      "IDPNTR" IS THE MEMORY CURSOR IN THE "DIRVCR" ARRAY.
C
C$      "IAPNTR" IS THE MEMORY CURSOR IN THE "ADJTBL" ARRAY.
C
C$      ONCE IT STARTS TO CODE A NEW DIRECTION VECTOR, "BUFER"
C$      MUST BE INITIALIZED.
C
C$      ONCE IT STARTS TO CODE AN ADJACENT EDGE, THE X AND THE Y
C$      COORDINATE LOCATIONS OF THE STARTING END POINT OF THIS
C$      ADJACENT EDGE MUST BE PUT INTO THE FIRST AND THE SECOND
C$      ELEMENTS OF THE CORRESPONDING CODE LIST RESPECTIVELY.
C
0031      DO 2000 KREG=1,256
0032      IF(REGPTR(KREG).LT.0) GO TO 2000
0034      IAPNTR=REGPTR(KREG)
0035      BUFER(1)=-1
0036      BUFER(2)=0
0037 200  IDOREG=ADJTBL(ADJREG,IAPNTR)

```

FORTRAN IV

V01C-03G+ MON 21-MAY-79 02:58:18

PAGE 003

```

0038      IF(IDOREG.LT.0) GO TO 2000
0040      TYPE109, IDOREG, KREG
0041 109    FORMAT(1H0, 5X, 'DOING REGION', I4, ' -----> REGION', I4)
0042      ISPNTN=ADJTB(LPNTR, IAPNTR)
0043      LOCATN=SEQLST(ISPNTN)
0044      IX=MOD(LOCATN, 128)
0045      IF(IX.EQ.0) IX=128
0046      IY=(LOCATN-IX)/128+1
0047      IDPNTR=IDPNTR+1
0048      IF((IDPNTR+1).GT.2560) STOP '? NO MORE SPACE FOR CODE, > 2560'
0049      IHEAD=IDPNTR
0050      DIRVCR(IDPNTR)=IX
0051      IDPNTR=IDPNTR+1
0052      DIRVCR(IDPNTR)=IY
0053      CHANTB(LPNTR, IAPNTR)=IHEAD
0054      KPXLNO=KPXLNO+1
0055      C
0056      D    WRITE(6, 209) IX, IY, KREG, IDOREG, IAPNTR, IHEAD
0057 209    FORMAT('0 ***** START AT XCOD=', I4, ', YCOD=', I4, '//, ' REGION',
0058      D    - I4, ' -----> REGION', I4, '//, 10X, ' IAPNTR=', I6, 5X, ' IHEAD=', I6, '//)
0059      C
0060      C*****
0061      C
0062      C$    DETERMINE THE DIRECTION CODE OF THE CURRENTLY TRACED PIXEL.
0063      C
0064      C$    CHECK IF NEW DIRECTION VECTOR COMES OUT.
0065      C
0066      C$    THE MAXIMUM LENGTH OF THE DIRECTION VECTOR IS 31.
0067      C
0068      C$    IF THE LENGTH OF THE DIRECTION VECTOR EXCEEDS 31, BREAK IT
0069      C$    INTO TWO DIRECTION VECTORS.
0070      C
0071      C$    "DIRCOD" IS THE DIRECTION CODE OF THE CURRENTLY TRACED PIXEL.
0072      C
0073 400      JOPTLN=LOCATN
0074      ISPNTN=ISPNTN+1
0075      LOCATN=SEQLST(ISPNTN)
0076      IF(LOCATN.EQ.0) GO TO 900
0077      KPXLNO=KPXLNO+1
0078      JDIFER=LOCATN-JOPTLN
0079 500      DIRCOD=9
0080      IF(JDIFER.EQ.1) DIRCOD=0
0081      IF(JDIFER.EQ.129) DIRCOD=1
0082      IF(JDIFER.EQ.128) DIRCOD=2
0083      IF(JDIFER.EQ.127) DIRCOD=3
0084      IF(JDIFER.EQ.-1) DIRCOD=4
0085      IF(JDIFER.EQ.-129) DIRCOD=5
0086      IF(JDIFER.EQ.-128) DIRCOD=6
0087      IF(JDIFER.EQ.-127) DIRCOD=7
0088      IF(DIRCOD.EQ.9) STOP '? BAD INPUT DATA'
0089      IF(BUFER(1).EQ.-1) BUFER(1)=DIRCOD
0090      IF(BUFER(1).NE.DIRCOD) GO TO 600
0091      IF(BUFER(2).EQ.31) GO TO 600
0092      BUFER(2)=BUFER(2)+1

```

FORTRAN IV V01C-03G+ MON 21-MAY-79 02:58:18

PAGE 004

```

0092      GO TO 400
C*****
C
C$      IF THE DIRECTION CODE OF THE DIRECTION VECTOR RANGES FROM
C$      0 TO 3, THE DIRECTION VECTOR IS CODED IN THE FOLLOWING WAY.
C
0093 600      IF(BUFER(1).GE.4) GO TO 700
0095      CODE=BUFER(1)*32+BUFER(2)
0096      GO TO 800
C*****
C
C$      IF THE DIRECTION CODE OF THE DIRECTION VECTOR RANGES
C$      FROM 4 TO 7, THE DIRECTION VECTOR IS CODED IN THE FOLLOWING
C$      WAY.
C
0097 700      CODE=-((BUFER(1)-4)*32+BUFER(2))
C
C$      PUT THE CODE OF THE DIRECTION VECTOR INTO THE CODE LIST.
C
0098 800      IDPNTR=IDPNTR+1
0099      NCOD=NCOD+1
0100      IF(IDPNTR.GT.2560) STOP '? NO MORE SPACE FOR CODE, > 2560'
0102      DIRVCR(IDPNTR)=CODE
C
0103 D      WRITE(6,809) CODE, IDPNTR, BUFER(1), BUFER(2)
0104 D809      FORMAT(5X, 'CODE =', I5, 5X, 'IDPNTR =', I6, 5X, 'DIRECTION =', I2, 5X,
D      - 'LENGTH =', I3)
C
0105      BUFER(1)=-1
0106      BUFER(2)=0
0107      GO TO 500
C*****
C
C$      PUT THE CODE OF THE LAST DIRECTION VECTOR INTO THE CODE LIST.
C
C$      CLOSE THE CODE LIST AND GO TO CODE THE NEXT ADJACENT EDGE.
C
0108 900      IF(BUFER(1).GE.4) GO TO 1100
0110      CODE=BUFER(1)*32+BUFER(2)
0111      GO TO 1300
0112 1100      CODE=-((BUFER(1)-4)*32+BUFER(2))
0113 1300      IDPNTR=IDPNTR+1
0114      NCOD=NCOD+1
0115      IF(IDPNTR.GT.2560) STOP '? NO MORE SPACE FOR CODE, > 2560'
0117      DIRVCR(IDPNTR)=CODE
0118 D      WRITE(6,809) CODE, IDPNTR, BUFER(1), BUFER(2)
0119      CODE=0
0120      IDPNTR=IDPNTR+1
0121      IF(IDPNTR.GT.2560) STOP '? NO MORE SPACE FOR CODE, > 2560'
0123      DIRVCR(IDPNTR)=CODE
C
0124 D      WRITE(6,1309) CODE, IDPNTR
0125 D1309      FORMAT(5X, 'CODE =', I5, 5X, 'IDPNTR =', I6, 5X, 30('*'))
C

```

FORTTRAN IV V01C-03G+ MON 21-MAY-79 02:58:18

PAGE 005

```

0126      BUFER(1)=-1
0127      BUFER(2)=0
0128      IAPNTR=IAPN1R+1
0129      GO TO 200
0130 2000  CONTINUE
C*****
0131      TYPE2009, IDPNTR, NCOD
0132 2009  FORMAT(1H0, 5X, '***==** CODE SEQUENTIAL LISTS TOTALLY', I6, ' BYTES',
-       /15X, 'BUT REALLY JUST', I6, ' BYTES FOR CODE')
0133      JPXL=KPXLND*2
0134      TYPE2019, KPXLND, JPXL
0135 2019  FORMAT(1H0, 5X, '$--$ TOTALLY', I6, ' PIXELS ----> ', I6, ' BYTES')
C*****
C
C$      OUTPUT THE COMPRESSED DATA BASE.
C
0136      IF(IWRITW(1280, DIRVCR, 0, ICH1).NE. 1280) STOP '? BAD WRITE-1'
0138      IF(IWRITW(1024, CHANTB, 5, ICH1).NE. 1024) STOP '? BAD WRITE-2'
0140      IF(IWRITW(256, REGPTR, 12, ICH1).NE. 256) STOP '? BAD WRITE-3'
0142      CALL CLOSEC(ICH1)
0143      STOP 'NORMAL END EXECUTION'
0144      END

```

FORTRAN IV

V01C-03G+ MON 21-MAY-79 03:11:39

PAGE 001

```

C                               *** ACACOD.FOR ***
C
C*****
C* SUBJECT:  THIRD CODING SCHEME FOR COMPRESSION.
C* PROGRAMMER:  CHIEN-CHYUN LEE.
C* DATE:  NOVEMBER 12, 1978.
C* INPUT:  FILENAME MUST BE *.DAT.
C*          THIS INPUT DATA FILE IS THE OUTPUT DATA FILE
C*          GENERATED FROM THE CHACOD.SAV PROGRAM. (REFER TO
C*          CHACOD.FOR). THE LENGTH OF THIS INPUT DATA FILE IS
C*          13 BLOCK LONG.
C*          THE DATA STRUCTURE OF THE INPUT FILE IS THE SAME AS
C*          THAT OF THE OUTPUT FILE, BUT THE END MARK OF THE
C*          CODING SEQUENTIAL LIST IS VALUE 0.
C* OUTPUT:  FILENAME MUST BE *.DAT.
C*          THE LENGTH OF THIS OUTPUT DATA FILE IS 10 BLOCKS.
C*          1. BLOCK 0 -- BLOCK 4: LOGICAL*1 DIRVCR(2560)
C*             ARRAY FOR SEQUENTIAL LIST TO STORE DIRECTION
C*             VECTOR CODE. VALUE -128 STANDS FOR END MARK OF
C*             EACH SEQUENTIAL LIST.
C*          2. BLOCK 5 -- BLOCK 11: INTEGER*2 CHANTB(2,896)
C*             ARRAY FOR CHAIN CODE TABLE (BY SEQUENTIAL LIST
C*             STRUCTURE). NEGATIVE VALUE STANDS FOR END MARK
C*             OF EACH SEQUENTIAL LIST. EACH NODE IN THE LIST
C*             HAS TWO WORDS. FIRST WORD STORES A NUMBER FOR
C*             ADJACENT REGION. SECOND WORD STORES THE POINTER
C*             TO ITS CORRESPONDING SEQUENTIAL LIST FOR DIREC-
C*             TION VECTOR CODE.
C*          3. BLOCK 12: INTEGER*2 REGPTR(256) ARRAY FOR REGION
C*             POINTER.
C* NOTE:
C*          1. THIS PROGRAM IS THE SECOND PART OF THE THIRD
C*             CODING SCHEME. THE FIRST PART IS THE SAME AS
C*             "CHACOD.FOR" PROGRAM.
C*          2. COMPILING WITH SWITCH "/D" WILL PRINT OUT THE
C*             OUTPUT DATA BASE OF THIS PROGRAM.
C*****
C
C
C* "DIRVCR" IS THE DIRECTION VECTOR CODE ARRAY FOR THE CODE LIST
C*   OF THE INPUT DATA BASE.
C
C* "NEWVCR" IS THE DIRECTION VECTOR CODE ARRAY FOR THE THE CODE
C*   LIST IN THE OUTPUT DATA BASE OF THIS PROGRAM.
C
C* "REGPTR" IS THE REGION POINIER ARRAY.
C
C* "CHANTB" ARRAY IS THE ADJACENCY TABLE.
C
0001      LOGICAL*1 DIRVCR(2560),NEWVCR(2560)
0002      INTEGER*2 REGPTR(256),CHANTB(2,896)
0003      INTEGER*2 FILSPC(39),DEFEXT(4),DIRCOD, CODE, DIRCD1
0004      DATA DEFEXT/4*3RDAT/
0005      COMMON LENTH, DIRCOD

```

FORTRAN IV

V01C-03G+ MON 21-MAY-79 03:11:39

PAGE 002

```

C*****
C
C$   INPUT AND OUTPUT ARRANGEMENTS.
C
0006   CALL PRINT('ENTER COMMAND STRING -- *.DAT<*.DAT')
0007   CALL ICSI(FILSPC,DEFEXT,,,0)
0008   IX=IFETCH(FILSPC(1))
0009   IY=IFETCH(FILSPC(16))
0010   IF((IX+IY).NE.0) STOP '? BAD COMMAND STRING'
0012   ICH1=IGETC()
0013   ICH2=IGETC()
0014   IF(ICH1.LT.0.OR.ICH2.LT.0) STOP '? NO CHANNEL AVAILABLE'
0016   IF(IENTER(ICH1,FILSPC(1),13).NE.13) STOP '? NO SPACE OUT'
0018   IF(LOOKUP(ICH2,FILSPC(16),13).NE.13) STOP '? NO SUCH FILE'
0020   IF(IREADW(1280,DIRVCR,0,ICH2).NE.1280) STOP '? BAD READ-1'
0022   IF(IREADW(1792,CHANTB,5,ICH2).NE.1792) STOP '? BAD READ-2'
0024   IF(IREADW(256,REGPTR,12,ICH2).NE.256) STOP '? BAD READ-3'
C*****
C
C$   "ADJREG" STANDS FOR THE ADJACENCY REGION.
C
C$   "PNTR" STANDS FOR POINTER IN THE ADJACENCY TABLE.
C
C$   "NCOO" KEEPS THE NUMBER OF THE CODE IN THE CODE LIST.
C
0026   ADJREG=1
0027   PNTR=2
0028   INPNTR=0
0029   MAXLEN=6
0030   NOCOD=0
C*****
C
C$   THE CODING OPERATION IS PROCESSED REGION BY REGION,
C$   EDGE BY EDGE.
C
C$   "ICPNTR" IS THE MEMORY CURSOR IN THE "CHANTB" ARRAY.
C
C$   "IDPNTR" IS THE MEMORY CURSOR IN THE CODE LIST OF THE INPUT
C$   DATA BASE.
C
C$   "INPNTR" IS THE MEMORY CURSOR IN THE CODE LIST OF THE OUTPUT
C$   DATA BASE.
C
0031   DO 3000 KREG=1,256
0032   ICPNTR=REGPTR(KREG)
0033   IF(ICPNTR.LT.0) GO TO 3000
0035 200 IDOREG=CHANTB(ADJREG,ICPNTR)
0036   IF(IDOREG.LT.0) GO TO 3000
0038   TYPE109,IDOREG,KREG
0039 109 FORMAT(1H0,5X,'DOING REGION',I4,' -----> REGION',I4)
0040   IDPNTR=CHANTB(PNTR,ICPNTR)
0041   IX=DIRVCR(IDPNTR)
0042   IDPNTR=IDPNTR+1
0043   IY=DIRVCR(IDPNTR)

```

FORTRAN IV

V01C-03G+ MON 21-MAY-79 03:11:39

PAGE 003

```

0044      INPNTR=INPNTR+1
0045      IF((INPNTR+1).GT.2560) STOP '? NO MORE SPACE FOR CODE, > 2560'
0047      IHEAD=INPNTR
0048      NEWDVR(INPNTR)=IX
0049      INPNTR=INPNTR+1
0050      NEWDVR(INPNTR)=IY
0051      CHANTB(PNTR,ICPNTR)=IHEAD
      C
0052      D      WRITE(6,209) IX,IY,KREG,IDOREG,ICPNTR,IHEAD
0053      D209  FORMAT('O ***** START AT XCOD=',I4,', YCOD=',I4,'/5X, 'REGION',I4,
      D      - ' /-----> REGION',I4,'/15X, 'ICPNTR=',I6,5X, 'IHEAD=',I6,'//')
      C
      C*****
      C
      C$      SUBROUTINE "DECODE" IS CALLED TO DECODE THE INPUT DATA (THE
      C$      CODE GENERATED BY THE SECOND CODING SCHEME).
      C
      C$      IF THE LENGTH OF THE INPUT DIRECTION VECTOR IS LESS THAN 6,
      C$      DETECT THE NEXT INPUT DIRECTION VECTOR TO SEE IF ITS DIRECTION
      C$      CODE IS THE SAME AS THAT OF THE CURRENTLY DETECTED INPUT
      C$      DIRECTION VECTOR.
      C
      C$      IF THEIR DIRECTION CODES ARE THE SAME, THEIR LENGTHS
      C$      ARE SUMMED TOGETHER, THEN GO TO DETECT THE NEXT INPUT
      C$      DIRECTION VECTOR.
      C
0054      400  IDPNTR=IDPNTR+1
0055      CODE=DIRVCR(IDPNTR)
0056      IF(CODE.EQ.0) GO TO 1600
0058      CALL DECODE(CODE)
0059      600  IF(LENTH.GT.6) GO TO 700
0061      DIRCD1=DIRCOD
0062      KLENTH=LENTH
0063      CODE=DIRVCR(IDPNTR+1)
0064      IF(CODE.EQ.0) GO TO 750
0066      CALL DECODE(CODE)
0067      IF(DIRCOD.NE.DIRCD1) GO TO 350
0069      LENTH=LENTH+KLENTH
0070      IDPNTR=IDPNTR+1
0071      GO TO 600
      C*****
      C
      C$      CHECK IF THE LENGTH OF THE CURRENTLY DETECTED INPUT DIRECTION
      C$      VECTOR IS 1.
      C
      C$      IF THE LENGTH IS NOT 1, THEN CODE THE CURRENTLY DETECTED
      C$      INPUT DIRECTION VECTOR IN THE FOLLOWING WAY.
      C
0072      850  DIRCOD=DIRCD1
0073      LENTH=KLENTH
0074      750  IF(LENTH.EQ.1) GO TO 800
0076      650  NCODE=(DIRCOD*6)+(LENTH-1)
0077      NOCOD=NOCOD+1
0078      INPNTR=INPNTR+1

```

FORTRAN IV V01C-03G+ MON 21-MAY-79 03:11:39 PAGE 004

```

0079      IF(INPNTR.GT.2560) STOP '? NO MORE SPACE FOR CODE, > 2560'
0081      NEWDVR(INPNTR)=NCODE
      C
0082      D      WRITE(6,609) NCODE, INPNTR, DIRCOD, LENTH
0083      D609   FORMAT(5X, 'CODE =', I5, 5X, 'INPNTR =', I6, 5X, 'DIRECTION =', I3,
      D      - 5X, 'LENGTH =', I2)
      C
0084      GO TO 400
      C*****
      C
      C$      BREAK THE INPUT DIRECTION VECTOR INTO TWO VECTORS.
      C
      C$      THE FIRST VECTOR WITH LENGTH 6 IS RECODED IN THE FOLLOWING
      C$      WAY.
      C
      C$      THE SECOND VECTOR WILL BE CHECKED IF ITS LENGTH IS STILL
      C$      GREATER THAN 6.
      C
      C$      IF ITS LENGTH IS NOT GREATER THAN 6, IT WILL BE CHECKED IF
      C$      IT CAN BE COMBINED WITH ITS NEXT INPUT DIRECTION VECTOR.
      C
0085      700   NCODE=(DIRCOD*6)+5
0086      NCCOD=NCCOD+1
0087      INPNTR=INPNTR+1
0088      IF(INPNTR.GT.2560) STOP '? NO MORE SPACE FOR CODE, > 2560'
0090      NEWDVR(INPNTR)=NCODE
0091      LENTH=LENTH-6
0092      D      WRITE(6,609) NCODE, INPNTR, DIRCOD, MAXLEN
0093      GO TO 600
      C*****
      C
      C$      GET THE NEXT INPUT DIRECTION VECTOR AND CALL "DECODE"
      C$      TO DECODE IT.
      C
      C$      COMBINE IT WITH THE LAST DIRECTION VECTOR INTO A SUPER-
      C$      DIRECTION VECTOR.
      C
      C$      IF THEY CANNOT BE COMBINED TOGETHER (NO SUPER-DIRECTION
      C$      CODE IS ASSIGNED TO THEIR COMBINATION), THE LAST
      C$      DIRECTION VECTOR WILL BE CODED INDIVIDUALLY.
      C
      C$      "NEWDIR" IS THE SUPER-DIRECTION CODE.
      C
0094      800   DIRCD1=DIRCOD
0095      IDPNTR=IDPNTR+1
0096      CODE=DIRVCR(IDPNTR)
0097      IF(CODE.EQ.0) GO TO 1500
0099      CALL DECODE(CODE)
0100      NEWDIR=NDIR(DIRCD1)
0101      IF(NEWDIR.LT.0) GO TO 1800
      C*****
      C
      C$      CHECK ITS NEXT INPUT DIRECTION VECTOR TO SEE IF THEY
      C$      HAVE THE SAME DIRECTION CODE.

```

FORTRAN IV

V01C-03B+ MON 21-MAY-79 03:11:39

PAGE 005

```

C
C$ IF THEY HAVE THE SAME DIRECTION CODE, THEIR LENGTHS ARE
C$ SUMMED TOGETHER.
C
C$ THE LENGTH OF THE SUPER-DIRECTION VECTOR IS ASSIGNED AS
C$ THE SAME LENGTH OF THE CURRENTLY DETECTED DIRECTION VECTOR.
C
C$ IF THE LENGTH OF THE SUPER-DIRECTION VECTOR IS GREATER THAN
C$ 6, THE SUPER-DIRECTION VECTOR WILL BE BROKEN INTO TWO
C$ VECTORS.
C
0103 1050 IF(LENTH.GT.6) GO TO 1100
0105 DIRCD1=DIRCOD
0106 KLENTH=LENTH
0107 CODE=DIRVCR(IDPNTR+1)
0108 IF(CODE.EQ.0) GO TO 950
0110 CALL DECODE(CODE)
0111 IF(DIRCOD.NE.DIRCD1) GO TO 1150
0113 LENTH=LENTH+KLENTH
0114 IDPNTR=IDPNTR+1
0115 GO TO 1050
0116 1150 DIRCOD=DIRCD1
0117 LENTH=KLENTH
C*****
C
C$ IF THE SUPER-DIRECTION CODE RANGES FROM 8 TO 20, THE
C$ SUPER-DIRECTION VECTOR IS CODED IN THE FOLLOWING WAY.
C
0118 950 IF(NEWDIR.GT.20) GO TO 1000
0120 NCODE=(NEWDIR*6)+(LENTH-1)
0121 INPNTR=INPNTR+1
0122 NCCOD=NCCOD+1
0123 IF(INPNTR.GT.2560) STOP '? NO MORE SPACE FOR CODE. > 2560'
0125 NEWDVR(INPNTR)=NCODE
0126 D WRITE(6,609) NCODE, INPNTR, NEWDIR, LENTH
0127 GO TO 400
C*****
C
C$ IF THE SUPER-DIRECTION CODE RANGES FROM 21 TO 41, THE
C$ SUPER-DIRECTION VECTOR IS CODED IN THE FOLLOWING WAY.
C
0128 1000 NCODE=-(((NEWDIR-21)*6)+(LENTH-1)+1)
0129 GO TO 900
C*****
C
C$ BREAK THE SUPER-DIRECTION VECTOR INTO TWO VECTORS.
C
C$ THE FIRST VECTOR IS STILL A SUPER-DIRECTION VECTOR WITH
C$ LENGTH 6.
C
C$ THE SECOND VECTOR IS THE ORIGINAL-SENSE DIRECTION VECTOR,
C$ WHICH WILL BE CHECKED IF IT CAN BE COMBINED WITH ITS NEXT
C$ INPUT DIRECTION VECTOR.
C

```

FORTRAN IV V01C-03G+ MON 21-MAY-79 03:11:39 PAGE 006

```

0130 1100 IF(NEWDIR.GT.20) GO TO 1300
0132      NCODE=(NEWDIR*6)+5
0133 1200 INPNTR=INPNTR+1
0134      NOCOD=NOCOD+1
0135      IF(INPNTR.GT.2560) STOP '? NO MORE SPACE FOR CODE, > 2560'
0137      NEWDVR(INPNTR)=NCODE
0138 D    WRITE(6,609) NCODE, INPNTR, NEWDIR, MAXLEN
0139      LENTH=LENTH-6
0140      GO TO 600
0141 1300 NCODE=-(((NEWDIR-21)*6)+6)
0142      GO TO 1200

C*****
C
C$      CODE THE LAST INPUT DIRECTION VECTOR IN THE FOLLOWING WAY.
C
C$      GO TO CODE THE NEXT ADJACENT EDGE.
C
0143 1500 NCODE=(DIRCOD*6)+(LENTH-1)
0144      INPNTR=INPNTR+1
0145      NOCOD=NOCOD+1
0146      IF(INPNTR.GT.2560) STOP '? NO MORE SPACE FOR CODE, > 2560'
0148      NEWDVR(INPNTR)=NCODE
0149 D    WRITE(6,609) NCODE, INPNTR, DIRCOD, LENTH
0150 1600 NCOUE=-128
0151      INPNTR=INPNTR+1
0152      IF(INPNTR.GT.2560) STOP '? NO MORE SPACE FOR CODE, >2560'
0154      NEWDVR(INPNTR)=NCODE
C
0155 D    WRITE(6,1609) NCODE, INPNTR
0156 D1609 FORMAT(5X, 'CODE =', I5, 5X, 'INPNTR =', I6, 5X, 30(' '))
0157      ICPNTR=ICPNTR+1
C
0158      GO TO 200

C*****
C
C$      IF TWO DIRECTION VECTORS CAN NOT BE COMBINED INTO A
C$      SUPER-DIRECTION VECTOR, CHANGE THE CURSOR TO THE PREVIOUS
C$      DIRECTION VECTOR AND GO TO CODE THE PREVIOUS DIRECTION VECTOR
C$      INDIVIDUALLY.
C
0159 1800 DIRCOD=DIRCOD1
0160      LENTH=1
0161      IDPNTR=IDPNTR-1
0162      GO TO 650
0163 3000 CONTINUE

C*****
0164      TYPE3009, INPNTR, NOCOD
0165 3009 FORMAT(1H0, 5X, '***** CODE SEQUENTIAL LISTS TOTALLY', I6, ' BYTES',
- /15X, 'BUT REALLY JUST', I6, ' BYTES FOR CODE')
C*****
C
C$      OUTPUT THE COMPRESSED DATA BASE.
C
0166      IF(IWRITW(1280, NEWDVR, 0, ICH1).NE.1280) STOP '? BAD WRITE-1'

```

```
FORTRAN IV      V01C-03G+  MON 21-MAY-79 03:01:27      PAGE 007

0168      IF(IWRITW(1792,CHANTB,5,ICH1).NE.1792) STOP '? BAD WRITE-2'
0170      IF(IWRITW(256,REGPTR,12,ICH1).NE.256) STOP '? BAD WRITE-3'
0172      CALL CLOSEC(ICH1)
0173      STOP 'NORMAL END EXECUTION'
0174      END
```

FORTRAN IV

V01C-03G+ MON 21-MAY-79 03:04:51

PAGE 001

```

0001      SUBROUTINE DECODE(CODE)
C*****
C
C
C      THIS SUBROUTINE DECODE THE INPUT DATA GENERATED BY THE SECOND
C      CODING SCHEME.
C
C      "DIRCOD" IS THE DIRECTION CODE OF THE INPUT DIRECTION
C      VECTOR.
C
C      "LENGTH" IS THE LENGTH OF THE INPUT DIRECTION VECTOR.
C
C
0002      INTEGER*2 CODE, DIRCOD
0003      COMMON LENTH, DIRCOD
C*****
0004      IF(CODE.GT.0) GO TO 100
0006      CODE=-CODE
0007      LENTH=MOD(CODE,32)
0008      DIRCOD=(CODE-LENTH)/32
0009      DIRCOD=DIRCOD+4
0010      RETURN
0011 100  LENTH=MOD(CODE,32)
0012      DIRCOD=(CODE-LENTH)/32
0013      RETURN
0014      END

```

FORTRAN IV V01C-03G+ MON 21-MAY-79 03:15:30

PAGE 001

```

0001      FUNCTION NDIR(DIRCD1)
C*****
C
C
C
C$      THIS FUNCTION SUBROUTINE DETERMINES THE SUPER-DIRECTION
C$      CODE.
C
C$      "DIRCD1" IS THE DIRECTION CODE OF THE DIRECTION VECTOR
C$      WITH LENGTH 1.
C
C$      "DIRCOD" IS THE DIRECTION CODE OF THE NEXT DIRECTION
C$      VECTOR.
C
C$      IF THE TWO DIRECTION VECTOR CANNOT BE COMBINED INTO A
C$      SUPER-DIRECTION VECTOR (I.E., NO SUPER-DIRECTION CODE IS
C$      ASSIGNED TO THEIR DIRECTION CODES' COMBINATION), THE SUBROUTINE
C$      RETURNS THE VALUE "-5" TO INDICATE THE COMBINATION
C$      FAILED.
C
C
C
0002      INTEGER*2 DIRCD1, DIRCOD
0003      COMMON LENGTH, DIRCOD
C*****
0004      NDIR=-5
0005      IF(DIRCD1.EQ.7) GO TO 700
0007      IF(DIRCD1.EQ.6) GO TO 600
0009      IF(DIRCD1.EQ.5) GO TO 500
0011      IF(DIRCD1.EQ.4) GO TO 400
0013      IF(DIRCD1.EQ.3) GO TO 300
0015      IF(DIRCD1.EQ.2) GO TO 200
0017      IF(DIRCD1.EQ.1) GO TO 100
C*****
0019      IF(DIRCOD.EQ.1) NDIR=8
0021      IF(DIRCOD.EQ.2) NDIR=9
0023      IF(DIRCOD.EQ.6) NDIR=10
0025      IF(DIRCOD.EQ.7) NDIR=11
0027      RETURN
C*****
0028 100  IF(DIRCOD.EQ.2) NDIR=12
0030      IF(DIRCOD.EQ.3) NDIR=13
0032      IF(DIRCOD.EQ.7) NDIR=14
0034      IF(DIRCOD.EQ.0) NDIR=15
0036      RETURN
C*****
0037 200  IF(DIRCOD.EQ.3) NDIR=16
0039      IF(DIRCOD.EQ.4) NDIR=17
0041      IF(DIRCOD.EQ.0) NDIR=18
0043      IF(DIRCOD.EQ.1) NDIR=19
0045      RETURN
C*****
0046 300  IF(DIRCOD.EQ.4) NDIR=20
0048      IF(DIRCOD.EQ.5) NDIR=21

```

FORTRAN IV V01C-03G+ MON 21-MAY-79 03:15:30

PAGE 002

```

0050      IF(DIRCOD.EQ.1) NDIR=22
0052      IF(DIRCOD.EQ.2) NDIR=23
0054      RETURN
C*****
0055  400   IF(DIRCOD.EQ.5) NDIR=24
0057      IF(DIRCOD.EQ.6) NDIR=25
0059      IF(DIRCOD.EQ.2) NDIR=26
0061      IF(DIRCOD.EQ.3) NDIR=27
0063      RETURN
C*****
0064  500   IF(DIRCOD.EQ.6) NDIR=28
0066      IF(DIRCOD.EQ.7) NDIR=29
0068      IF(DIRCOD.EQ.3) NDIR=30
0070      IF(DIRCOD.EQ.4) NDIR=31
0072      RETURN
C*****
0073  600   IF(DIRCOD.EQ.7) NDIR=32
0075      IF(DIRCOD.EQ.0) NDIR=33
0077      IF(DIRCOD.EQ.4) NDIR=34
0079      IF(DIRCOD.EQ.5) NDIR=35
0081      RETURN
C*****
0082  700   IF(DIRCOD.EQ.0) NDIR=36
0084      IF(DIRCOD.EQ.1) NDIR=37
0086      IF(DIRCOD.EQ.5) NDIR=38
0088      IF(DIRCOD.EQ.6) NDIR=39
0090      IF(DIRCOD.EQ.2) NDIR=40
0092      IF(DIRCOD.EQ.4) NDIR=41
0094      RETURN
0095      END

```

VITA

Chien-Chyun Lee was born on January 6, 1954, at Chiao-Tou, Kaohsiung, Taiwan, China. He graduated from City Kaohsiung 2nd Junior High School in Kaohsiung City, Taiwan, China, in 1968; from Provincial Tainan First Senior High School in Tainan City, Taiwan, China, in 1971 and from National Taiwan University in Taipei, Taiwan, China in 1976 with a B.A. in Mathematics. He worked as a teaching assistant in the Mathematics Department of National Taiwan University during 1975-1976, and as a research assistant of Dr. Feng-Yao Lee during the summer of 1977. He has been working as a research assistant at the University of Tennessee Image and Pattern Analysis Laboratory with Drs. E. L. Hall, D. W. Bouldin, M. G. Thomason and R. C. Gonzalez since 1977.