

EXTRACTION OF LINES AND REGIONS FROM GREY TONE LINE DRAWING
IMAGES

by

Krishna Arvind

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE
in
Computer Science

APPROVED:

Dr. L. T. Watson

Dr. R. W. Ehrich

Dr. J. P. Bixler

Dr. R. V. Foutz

December, 1983
Blacksburg, Virginia

EXTRACTION OF LINES AND REGIONS FROM GREY TONE
LINE DRAWING IMAGES

by

Krishna Arvind

(ABSTRACT)

An algorithm is described for extracting lines from grey level digitizations of industrial drawings. The algorithm is robust, noniterative, sequential, and includes procedures for differentiating shaded areas from lines. Examples are given for complex regions of a typical mechanical drawing.

ACKNOWLEDGEMENTS

First and foremost I would like to thank my advisor Dr. Layne T. Watson for his untiring patience and the valuable assistance he provided without which this work would not have been possible. Thanks are also due to Dr. J. P. Bixler, Dr. Roger W. Ehrich and Dr. Rex Hartson who assisted me as members of my advisory committee. I would also like to take this opportunity to thank Dr. Haralick and his associates at the Spatial Data Analysis Laboratory for their helpful suggestions from time to time.

On the non academic front, I would like to thank my parents, my brother, my brother-in-law and sister, and Dr. P. R. S. Kishore who all helped me study in the U.S.A. And last, but not least, I dedicate this thesis to my parents who made everything possible.

TABLE OF CONTENTS

EXTRACTION OF LINES AND REGIONS FROM GREY TONE LINE DRAWING IMAGES	ii
ACKNOWLEDGEMENTS	iii

Chapter

	<u>page</u>
I. INTRODUCTION	1
II. ALGORITHMS	6
DOUBLE ADAPTIVE THRESHOLDING	6
REGION DETERMINATION	10
REGION EXTRACTION	12
LINE EXTRACTION	12
III. IMPLEMENTATION OF THE PARALLEL LINE TRACKER	19
DATA STRUCTURES	19
IMAGE BUFFER	19
VECTOR ARRAY	21
OPERATION OF THE TRACKER	22
LIMITATIONS	24
ENHANCEMENTS	24
IV. RESULTS	25
V. CONCLUSION	40
REFERENCES	42

Appendix

	<u>page</u>
A. PROGRAM LISTINGS FOR ALGORITHMS	47
VITA.	141

LIST OF FIGURES

<u>Figure</u>	<u>page</u>
1. Real world image	29
2. Close up of coil section	30
3. Bobbin, after filtering and sampling, before DAT . . .	31
4. Result of sharpening operator EHNUM on bobbin . . .	32
5. Topographic labelling of bobbin	33
6. Coil, after filtering and sampling, before DAT . . .	34
7. Topographic labelling of coil	35
8. Result of double adaptive threshold (DAT) algorithm on coil	36
9. Regions extracted from coil	37
10. Output of line tracker on coil (regions removed) .	38
11. Same as Figure 10 but showing only tracked lines . .	39

Chapter I

INTRODUCTION

There are many applications such as mapping, drafting, image compression, and computer vision that require robust algorithms for extracting lines and boundaries from images. While a very substantial effort has been made to solve that problem under the adverse image conditions typical in computer vision applications, most popular techniques have many computational disadvantages when image conditions are good. When there is no need to be concerned with serious line fragmentation due to noise, it is possible to deal more directly with the problems of line drawing semantics, including curvature, endpoints, junctions, intersections, variable width, smoothness, straightness, and problems of approximation and encoding.

The particular application that motivated the work reported here is the problem of converting industrial line drawings from hard copy into highly compressed graphical representations involving a small number of primitives such as lines, curves, and regions. This problem vanishes when electronic drafting systems are used to generate the original drawings. However, the reality is that many technical drawings are still created and communicated on

paper, and relatively little use is being made of electronically stored representations. As a consequence, revision and updating is difficult to do, and field documentation of large, high-technology systems such as aircraft, space vehicles, buildings, and computers all too frequently consists of containers, files, or even entire rooms full of hard copy reproductions.

The algorithms described in this paper make use of grey level digitizations rather than binary digitizations to maximize the amount of image information available to the interpretation procedures and to minimize the intelligence required of the sensors themselves. The resolution is high, and the images are assumed to be large - at least 1024 by 1024 picture elements. No attempt is made to repair poorly executed drawings. Because of the large data volume and in order to facilitate processing on inexpensive microcomputers, the use of iterative algorithms, including grey level and binary thinning, is precluded, and every possible attempt is made to make processing sequential by scan line. It is also assumed that the line drawings to be processed may contain small, solid, shaded (not textured) regions or regions of parallel adjacent lines too fine to resolve individually. Before presenting specific algorithms, a review of some of the existing approaches for extracting lines from high-contrast digitizations is given.

Currently there are a number of systems for the extracting and encoding of line structured data. These systems frequently contain dedicated and often expensive hardware such as fast optical scanners that do fast local raster scanning of image data. Black et. al. [2], discuss a general purpose follower for line structured data which is table driven using a PEPR flying spot scanner. Fulford et. al. [12], describe the FASTRAK system, which is an interactive line following digitizer, scanning a reduction of a map with a laser beam. The system depends upon human interaction and intervention for starting lines and guiding the tracker along noisy or ambiguous lines. SysCan, a system described by Leberl et. al. [22], features KartoScan, a raster scanner using white light and a CCD array sensor. It is an operational automated system which converts maps and drawings into digital format, and the digital data is edited, stored, retrieved and generally manipulated.

Holdermann et. al. [17], Peleg et. al. [34], Ting et. al. [43], Wang et. al. [45], and Weszka [47], describe general preprocessing techniques such as thresholding and other forms of filtering applied to binary and gray tone images.

Complete systems for the extraction of line structured data from binary and gray scale images are described in the

literature. A coding method for the vector representation of engineering drawings is discussed by Ramachandran [35]. This algorithm does not distinguish between lines and regions, and moreover, the final form of extracted information, which is an approximation, is not suitable for parametric representation (by splines, for example). Compression ratios of about 35:1 were achieved, which are not very satisfactory for line drawings. Woetzel [50], describes an automatic method for the scanning of cartographic maps and extracting the linework. The method only works on binary images and uses a fast thinning algorithm which may distort the line structure (this is not very critical for maps). Furthermore regions cannot be extracted. Dudani [8], describes a contour following algorithm that can be adapted as a line following algorithm. However, the algorithm does not handle thick lines.

The present work incorporates well known techniques for encoding, digital transmission, and ridge detection. For completeness a few relevant references are mentioned here. The paper by Freeman [10], is a good tutorial on line drawings and also explains the concept of chain codes. Graham [14], and Huang [18], discuss methods for digital transmission. Maxwell [26], attempts to evolve natural descriptors for line drawings for efficient human-computer

communication in the domain of computer graphics. Watson et. al. [46], and Haralick et. al. [16], describe a method for the topographic labelling of gray scale image characteristics such as peaks, ridges, valleys, etc., which can be applied to line drawings.

This thesis describes a system which extracts the linework and solid regions from large gray tone images of line drawings using noniterative fast algorithms with minimal storage requirements.

Chapter II

ALGORITHMS

A large digitized image produced by a collimated white light scanner was initially blurred by a Gaussian filter and then resampled at every other pixel (2:1 sampling) to produce the input image that was used to develop the algorithms. This image (see Figure 1) was of size 1024 by 1024, and consisted of thin and thick lines and a few regions with a uniform gray tone value except at the borders. The gray tone intensity surface of a line was ridge-like due to the Gaussian filtering. The gray tones in the image ranged from 0 to 255 in value, and average line intensities varied (from 50 to 250) in different parts of the image. Line separation was as low as one pixel in some portions of the image, and the valleys that separated these close lines had gray tone values in between those of the lines and the zero background.

2.1 DOUBLE ADAPTIVE THRESHOLDING

Because of the small differences in height between valleys and ridges (formally defined in Haralick et. al. [16]) and variation in average intensities over the image, simple thresholding using a single cutoff value fails to

resolve all the lines and suggests the use of local adaptive thresholding. A good local characteristic is the average of pixel values in a neighborhood centered around the pixel being tested. Thresholding can be described as determining which are the object pixels (typically represented by ones in the output image) and which are the background pixels (represented by zeros in the output image). A strategy whereby the threshold cutoff was computed by multiplying some constant by the average gray tone over a square n by n window centered at the current pixel produced very good results. Decreasing the window size has the effect of increasing the resolution, making the thresholding more sensitive to local features. Since low background pixel values have the effect of depressing the computed average, thereby diluting large peaks, only those pixels in the neighborhood which are greater than a certain low threshold are used in the computation of the average. This lower threshold is chosen to be slightly higher than the background pixel values. Thus there are two thresholds, the upper threshold which is computed by multiplying the window average by a cutoff factor, and the lower threshold--hence the name double adaptive thresholding. By choosing a value slightly greater than one for the cutoff factor, only the ridgeline pixels of the convex ridges have values more than

the computed cutoff and therefore appear in the output as object pixels. Region pixels do not pass the thresholding condition because of the flat nature of the regions. Here we make a small modification of the algorithm. Pixels which do not pass the thresholding condition but which are greater than a certain region threshold (chosen to be consistent with region pixel values) are marked as region pixels in the output. Note that this double adaptive thresholding is a procedure to extract structure from a given image, and not an intelligent program to restore interpretations that have been lost because of noise in some ideal image.

Thresholding that normally produces a binary image does not produce lines exactly one pixel thick, making line tracking rather difficult. One course of action would be to thin the binary image produced by the thresholding. A large variety of thinning algorithms are described in the literature, each has its own flavor from the point of view of the accuracy and aesthetic appeal of the skeletons they produce from the original binary image. Often the result is unappealing, as for example when curved lines are reduced to thin jagged lines. The problem is not with the thinning, but with the fact that once we create a binary image from thresholding, the information contained in gray tones of the original image is lost and no distinction can be made

between weak and strong object pixels which are all ones. Hence, a grey tone skeleton is needed, as produced by, for example, the grey scale medial axis transformation of Wang and Rosenfeld [45]. This algorithm was tried and found to decompose variable width lines, besides not dealing with irregularly shaped regions. Hence, in the double adaptive thresholding algorithm, all pixels which are above their computed cutoff values (i.e., the object pixels) are represented in the output by their original gray tone values. Region pixels are represented by the negative of their original gray tone values. The reason for retaining the gray tone values for region pixels will be apparent later when we describe region pruning.

The double adaptive thresholding algorithm (DAT) works well for lines when their gray tone intensity surfaces have a convex ridge shape, which can be artificially produced by Gaussian filtering, local averaging, or other known blurring techniques. The DAT algorithm is given below.

```

n := 3 (* window size *)
factor := 1.063
region_threshold := 200
low_threshold := 6
avg := average of all pixel values
      > low_theshold in an n by n
      neighborhood of the current

```

```

        pixel
cur_pixel := current input pixel
        value
out_pixel := current output pixel
        value

IF cur_pixel <= ( avg * factor )
THEN IF cur_pixel > region_threshold
      THEN out_pixel := ( -cur_pixel )
      ELSE out_pixel := 0
ELSE out_pixel := cur_pixel.

```

2.2 REGION DETERMINATION

The DAT, though it handles regions well has the effect of marking some stray pixels as region pixels. In addition, there may also be holes at the boundaries of the regions. In the next two steps, called 'growing' and 'shrinking', stray region pixels are eliminated and small holes in the regions are filled. Both these steps are based on the connectivity of region pixels. In the 'growing' operation each non-zero non-region pixel is marked as a region pixel if at least $k=3$ of its eight neighbors are region pixels. This has the effect of filling up small holes and growing the regions. Stray region pixels are not affected because of their low region connectivity. In the 'shrinking' operation, which is

complementary to and follows the 'growing' operation, a region pixel is changed to a non-region pixel if no more than 3 of its eight neighbors are region pixels. Stray region pixels are converted to non-region (i.e., line) pixels in this operation. The two steps described above which constitute region determination are shown below.

```

n := 3          (* window size *)
cur_pixel := current input pixel value
out_pixel := current output pixel value
num_neg := number of negative neighbors
           in an n by n neighborhood of
           the current pixel not counting
           the current pixel.

const1 := 3
const2 := 4

Grow:
IF num_neg >= const1 AND cur_pixel > 0
THEN out_pixel := ( -cur_pixel ) (* region *)
else out_pixel := ( cur_pixel ).

Shrink: (* with output of Grow as input *)
IF cur_pixel < 0 AND num_neg >= const2
THEN out_pixel := cur_pixel (* region *)
ELSE out_pixel := abs ( cur_pixel )

```

2.3 REGION EXTRACTION

The region determination step is followed by region extraction and line extraction. The regions can be represented by following and encoding their contours. Dudani [8], describes a method for region extraction using boundary following. Alternatively, simple run length coding or one of its more complex variations could be used. Such schemes work well because of good correlation of pixel runs between adjacent scan lines in the regions.

2.4 LINE EXTRACTION

Using a simple thinning algorithm to reduce the line width to one pixel followed by a simple line tracker is unsatisfactory for the reasons given above. A more complicated line tracker which tracks the ridges of the gray tone intensity surfaces of the lines is described below in two steps.

1. Finding the starting pixel for a new line.

The image is scanned from left to right and top to bottom. Thus, line by line each pixel is examined to check whether it continues a line or is a candidate for starting a new line. The following conditions must be satisfied by an unmarked pixel (called the candidate pixel) in order to be a starting point of a line:

- a) Its value is more than a certain threshold to indicate which pixel values are background and which are not.
- b) Its value is more than a factor (a good value is 0.7) times the average of its non-zero marked or unmarked eight neighbors. (When a pixel is tracked it is marked.)
- c) Its value is more than a factor (0.9) times the average of its marked eight neighbors. This condition ensures that the pixel is strong compared to nearby tracked vectors.
- d) Let pixel P1 be the unmarked, untested eight neighbor of the candidate pixel with maximum gray tone value. This selection implies a probable direction for a new line. Directions of the eight neighbors with respect to the candidate pixel are labeled from 0 through 7 in an anticlockwise direction with the northwest neighbour being labeled as 0. Let d be the direction of P1 with respect to the candidate pixel. P1 must also satisfy conditions a) through c).
- e) Let P11, P12, P13 be the neighbors in the directions d-1, d, d+1 (modulo 8) respectively from P1. At least one of these pixels P1j must not

be marked, must not have a marked eight neighbor in the directions $s-1$ and $s+1$ from P_1 (s is the direction of P_{1j} from P_1), and must satisfy conditions a) through c). Otherwise consider P_1 tested, and attempt to satisfy d) and e) with a different P_1 , until all possibilities have been tested.

If conditions a) through e) are satisfied by the candidate pixel, then it is marked along with P_1 and the line tracker is invoked to continue tracking from pixel P_1 .

2. Tracking a line

Let P_1 represent the previously marked pixel, P_2 the current marked pixel, and P_3 the next pixel sought by the line tracker. Let d_1 be the direction to P_1 from the pixel marked prior to P_1 , d_2 the direction from P_1 to P_2 , and d_3 the direction from P_2 to P_3 . From among the unmarked neighbors of P_2 in the directions $d_2-2 \pmod{8}$, $d_2-1 \pmod{8}$, d_2 , $d_2+1 \pmod{8}$, $d_2+2 \pmod{8}$, P_3 is chosen as the pixel (if it exists) with maximum gray tone value. P_3 is marked if it satisfies the conditions:

- a) P_3 's gray tone value is greater than a certain threshold which is slightly more than typical background values.

b) P3 has no marked eight neighbors in the directions $d3-1 \pmod{8}$ and $d3+1 \pmod{8}$ from P2.

c) $\min \{ \text{abs}(d3-d1), 8-\text{abs}(d3-d1) \} \leq 2$.

If P3 does not exist or does not satisfy conditions a) through c), then the current vector is terminated and the next line continuation pixel or line starting pixel is sought. If no such pixel is found, the line tracker moves down to the next scan line.

The algorithm for the line tracker is shown below.

STEP 1. Determination of the starting point.

cur_pix := current pixel

VALUE (cur_pix) := value of current pixel

cur_val := VALUE (cur_pix)

n := 3 (* window size *)

avg_marked := average of all the marked pixel
values in an n by n neighborhood
of the current pixel

avg_nonzero := average of all the non-zero
pixel values in an n by n
neighborhood of the current pixel

cutoff1 := 0.9 * avg_marked

cutoff2 := 0.7 * avg_nonzero

CUTOFF (cur_pix) := MAX (50, cutoff1, cutoff2)

IF cur_pix is not marked AND

```

cur_val > CUTOFF ( cur_pix )
THEN WHILE ( some unmarked neighbor of cur_pix
            not tested ) DO

BEGIN

temp_pix := unmarked, untested neighbor of
            cur_pix with maximum value

temp_val := VALUE ( temp_pix )
IF temp_val > CUTOFF ( temp_pix )
THEN
BEGIN

cur_dir := direction from cur_pix to
            temp_pix

pix1 := pixel in dir. cur_dir-1 from temp_pix
pix2 := pixel in dir. cur_dir from temp_pix
pix3 := pixel in dir. cur_dir+1 from temp_pix
IF VALUE ( pix1 ) > CUTOFF ( pix1 ) AND pix1
            not marked AND pix1 has no adjacent
            marked neighbors (* adjacent means
            in the direction from temp_pix +1
            or -1 *)

OR VALUE ( pix2 ) > CUTOFF ( pix2 ) AND pix2
            not marked AND pix2 has no adjacent
            marked neighbors

OR VALUE ( pix3 ) > CUTOFF ( pix3 ) AND pix3
            not marked AND pix3 has no adjacent

```

marked neighbors

THEN mark cur_pix, temp_pix, and call tracker

END

ELSE designate temp_pix as tested

END (* WHILE *)

STEP 2. Tracking the vector.

cur_dir := current direction

prev_dir := previous direction

threshold := 50

next_pix := unmarked neighbor in direction d from
 cur_pix, $\text{abs} (d - \text{cur_dir} \pmod{8}) \leq 2$,
 with maximum value.

IF next_pix exists

THEN

BEGIN

 next_dir := direction from cur_pix to next_pix

 next_val := VALUE (next_pix)

END

ELSE next_val := 0

IF next_val > threshold AND

$\text{abs} (\text{next_dir} - \text{prev_dir} \pmod{8}) \leq 2$ AND

 neighbors of next_pix in directions

 next_dir-1, next_dir+1 from cur_pix are not

 marked

```
THEN
BEGIN
    mark next_pix
    cur_pix := next_pix
    prev_dir := cur_dir
    cur_dir := next_dir
    continue tracking the current vector
END
ELSE
BEGIN
    terminate tracking the current vector
    determine the starting point of next vector
END.
```

The line tracker, in addition to marking the pixels in the image, also outputs the new chain code or absolute coordinates of the pixels. The output format depends on the user's requirements. Skiansky et. al. [41], discuss a method for fast polygonal approximation of the pixels as the pixels are being tracked. With their scheme a straight line would be represented by its two end points.

Chapter III

IMPLEMENTATION OF THE PARALLEL LINE TRACKER

The parallel tracker was implemented on the VAX 11/780 system using 'RATFOR', the language recommended for image processing software at the Spatial Data Analysis Laboratory at Virginia Tech. Two very important data structures for realizing the tracker are described in the next section.

3.1 DATA STRUCTURES

3.1.1 IMAGE BUFFER

This is a buffer used to contain a number of scan lines in primary memory at the same time. It is an array IMGBUF(NBUF, NPPL) interpreted as NBUF scan lines, each such line being of length NPPL, corresponding to the number of pixels per line in the image. Thus at any time IMGBUF would contain a horizontal slice of the image. To facilitate updating of the buffer, pointers INDEX(1) through INDEX(NBUF) are used to point to line 1 through line NBUF of the portion of the image. Therefore, the nth line in the image portion is not in line n of the image buffer - its physical line number in IMGBUF is contained in the variable INDEX(n).

Each time the buffer is updated, vectors which were previously tracked up to line NBUF-1 are now tracked until

they either terminate or reach line NBUF. Line 4 is scanned from pixel 1 to pixel NPPL and each pixel is tested for starting a vector. Line 4 rather than line 2 is chosen to take care of the unlikely event that a vector may track vertically upwards for a short distance without having been detected during earlier scans. For such an event our requirement that we test three levels of neighbours translates to the necessity of lines 3, 2, and 1 which contain pixels that are needed to test the candidate pixel in line 4.

Morover, for any given pixel, if a choice exists between using it for continuing an existing vector and using it for the process of testing a pixel (which may be the same) as a starting point, then priority is given to continuing an existing vector, so that vectors are as long as possible. For this reason, all existing vectors are tracked first to an extent that further tracking of these vectors would not affect the testing of pixels for starting points. This requirement coupled with the fact that three levels of neighbors are to be tested leads to the need of separating line 4 and line NBUF by at least 3 lines. This gives the minimum value of NBUF to be 8 so that after each buffer update vectors which were previously tracked up to line 7 are now tracked up to line 8, and then line 4 is scanned for

starting points where again all new vectors found are either terminated or tracked up to line 8. However, the larger the size of NBUF the less likely that curved or looping lines will be broken and detected as multiple lines.

3.1.2 VECTOR ARRAY

This is used for storing the tracked vectors. It is an array VCTARR(MAXVCT, MAXLEN) interpreted as MAXVCT vectors each of maximum length MAXLEN. This gives the tracker the ability to track a maximum of MAXVCT vectors simultaneously.

Each vector in VCTARR consists of the following information in positions 1 to 8. This information is necessary for tracking and ordering the vectors.

1. DLINK :- This is a pointer to the next vector in VCTARR. The vectors in VCTARR are interpreted as a linked list of vectors, starting with the oldest vector found (still being tracked) to the most recently found vector. Thus, in the linked list, the vectors are ordered in the order in which their starting points were found.
2. DIRLST :- The direction the vector took in finding the pixel previous to the last pixel.
3. DIRCUR :- The direction the vector took in finding the last pixel.

4. PTRCUR :- A pointer to the next available slot for storing the direction the vector takes in finding the next pixel.
5. XVCT :- The absolute x-coordinate of the starting point of the vector.
6. YVCT :- The absolute y-coordinate of the starting point of the vector.
7. XCUR :- The absolute x-coordinate of the last tracked pixel of the vector.
8. YCUR :- The absolute y-coordinate of the last tracked pixel of the vector.

Positions 9 to MAXLEN are used to store the successive directions (i.e., the chain code) of the vector.

3.2 OPERATION OF THE TRACKER

The action of the parallel tracker can be explained as a cyclic process consisting of 3 phases which are explained below.

1. Phase 1 :- Starting from the first vector in the linked list of vectors each vector (which would have already been tracked upto line NBUF-1) is tracked until it either terminates or reaches line NBUF. When a vector terminates it is written out so that an empty slot is created which can be used for a new

vector later. The DLINK pointer of the previous vector is updated to point to the next vector.

2. Phase 2 :- Line 4 is scanned from pixel 1 to pixel NPPL. For each pixel, if it satisfies the criteria for starting a vector, then a new vector slot in VCTARR is allotted. The DLINK pointer of the last tracked vector is made to point to the position of this new vector in VCTARR. The new vector is now tracked until it either terminates or reaches line NBUF. If it terminates then the vector is written out and the linked list of vectors is updated as in phase 1.
3. Phase 3 :- The buffer is updated by reading a new line. This is done by first rotating the pointers INDEX(1) through INDEX(NBUF). INDEX(1) is made equal to INDEX(2), INDEX(2) is made equal to INDEX(3)... and INDEX(NBUF) is made equal to the original value of INDEX(1). The new line is now read into the line pointed to by INDEX(NBUF). With this the buffer has been updated by one scan line and we now return to Phase 1.

3.3 LIMITATIONS

1. If a vector causes overflow of VCTARR then it is terminated and a new vector is begun.
2. Certain curved or looped lines may be broken up into two or more lines.
3. Only MAXVCT vectors can be tracked simultaneously.

3.4 ENHANCEMENTS

1. The problem of breaking up some curved lines can be alleviated by imparting to the program the ability to link up vectors created from the same continuous line.
2. Polygonal approximation as described in [41] can be included in the tracker. In this case, however, rather than chain codes, the absolute coordinates of the approximate polygon vertices will be stored.

Chapter IV

RESULTS

Fig. 1 shows a real world picture (i. e. before digitization) of one of the images used for testing the algorithms. Fig. 2 shows a close up view of the spool of wire in the image which was produced by digitization of the real world image. This is the most crucial part of the image since the line separation here is very low, and the lines merge at the edge of the coil to form textured regions. The lines are sparse in most of the image except in the spool of wire. A 1024 by 1024 test image (to which the algorithms were applied) was obtained by blurring the larger image using a Gaussian filter of standard deviation 1.039, and then resampling the filtered image at every other pixel. Compression ratios higher than 2:1 cause Moire patterns in the spool of wire, attributed to the classical problem of aliasing.

The test image corresponded very closely with what would have been produced by some industrial imaging hardware already in place. The larger image was taken as the starting point simply for the purposes of comparison and testing.

Fig. 3 displays the gray tone values of a portion of the spool of wire, where the upper portion resembles a "bobbin".

The ridge pixels of some lines have been highlighted. Note the presence of a considerable amount of noise, some of which was present in the original image, and the rest was added by the filtering. A well known iterative sharpening algorithm EHNUM, (an iterative algorithm which replaces each pixel's gray tone value by the nearest of the max or min of its eight neighbors' gray tone values) was applied to this noisy image. The result is illustrated in Fig. 4. Although the image in Fig. 4 does have a cleaner, sharper appearance, the lines are now rather jagged because the gray tone values of many insignificant pixels have now been raised. There is a more serious defect, namely, although there are only two vertical lines at the top in the input image Fig. 3, the sharpening algorithm EHNUM has found three (highlighted in Fig. 4).

Fig. 5 shows the result of applying the topographic labeling algorithm [16], to the image in Fig. 3. The algorithm fits a two-dimensional cubic polynomial to the gray tone values in a square 5 by 5 window (this window size can be changed) centered around each pixel. This polynomial represents a surface which best fits, in a discrete least squares sense, the pixel data in the window. The topography of the surface at the position of the central pixel is now determined using partial derivatives of the

polynomial. By tuning threshold parameters in the algorithm a good representation of the ridges (flagged by asterisks in Fig. 5) was obtained.

Fig. 6 shows a dense part of the coil with a typical line highlighted. The lines are not well resolved and there is a lot of noise between the lines. The topographic labeling algorithm was applied to this image with the same parameter values that were used to obtain the image in Fig. 5 (see Fig. 7). The detection of the gray tone surface ridges, which clearly exist in Fig. 6, is obviously extremely poor in Fig. 7, where what should have been a line is highlighted. This illustrates the sensitive nature of the algorithm to the threshold parameter values, making it unsuitable for the present application.

The DAT was used with very good results on the image in Fig. 6. Fig. 8 shows this result, in which the lines have been accentuated very well. The same line is highlighted in Figs. 6, 7, and 8. The negative pixels represent the regions on which the GROW and SHRINK algorithms were applied to finally get the regions shown in Fig. 9. These extracted regions were quite consistent with the perceived regions in the original unprocessed image.

The line tracker was now applied to the image in Fig. 8 after the regions (shown in Fig. 9) had been removed.

Pixels which the line tracker marked are shown with negative values in Fig. 10. For further illustration, the same tracked image is shown in Fig. 11, but as a binary image indicating the pixels marked by the tracker. Observe that the tracked lines very faithfully represent the lines in the original unprocessed image.

The algorithms were also applied to a number of other test images and very satisfactory results were obtained, thus attesting to the robust nature of the algorithms.

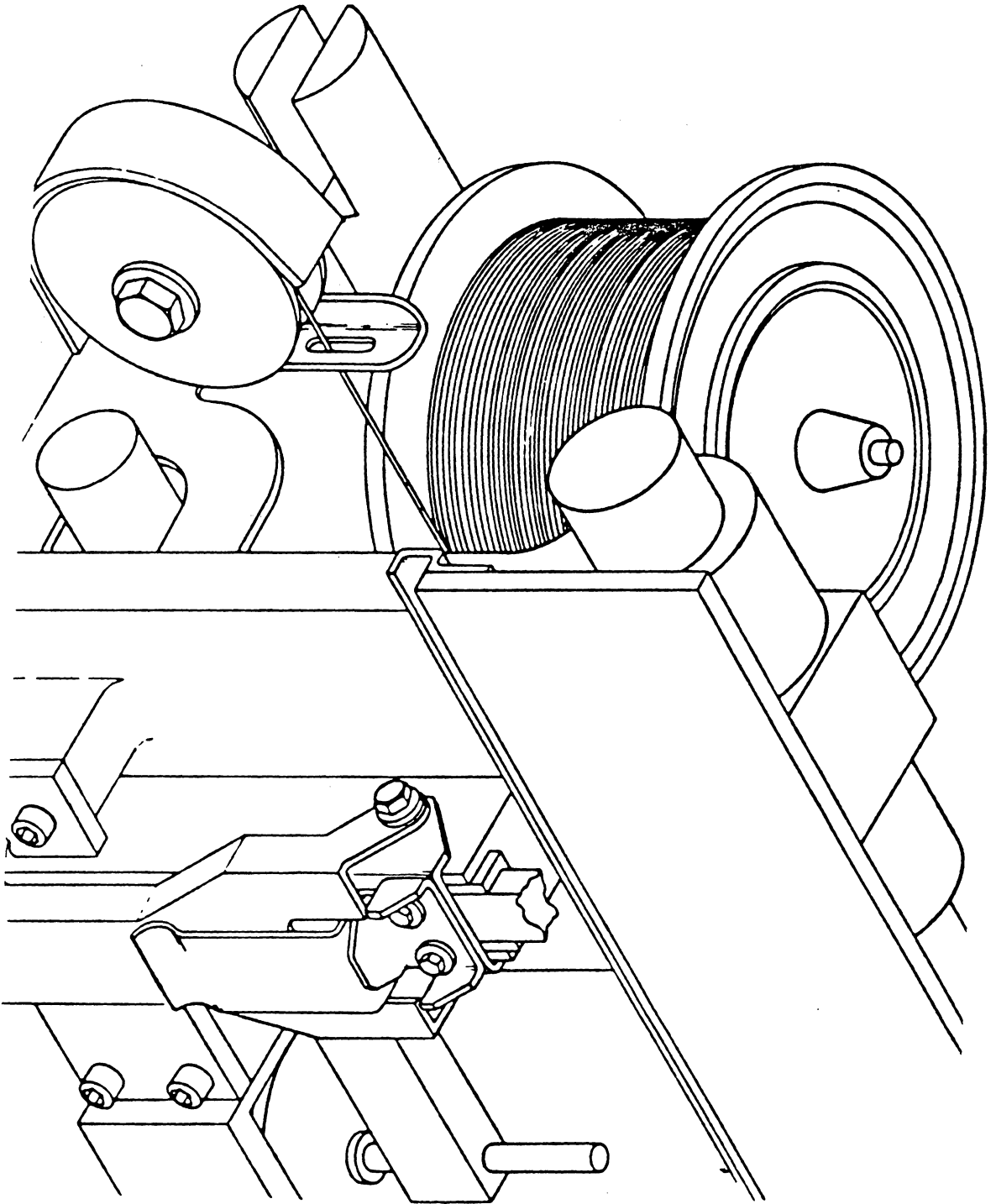


Figure 1: Real world image

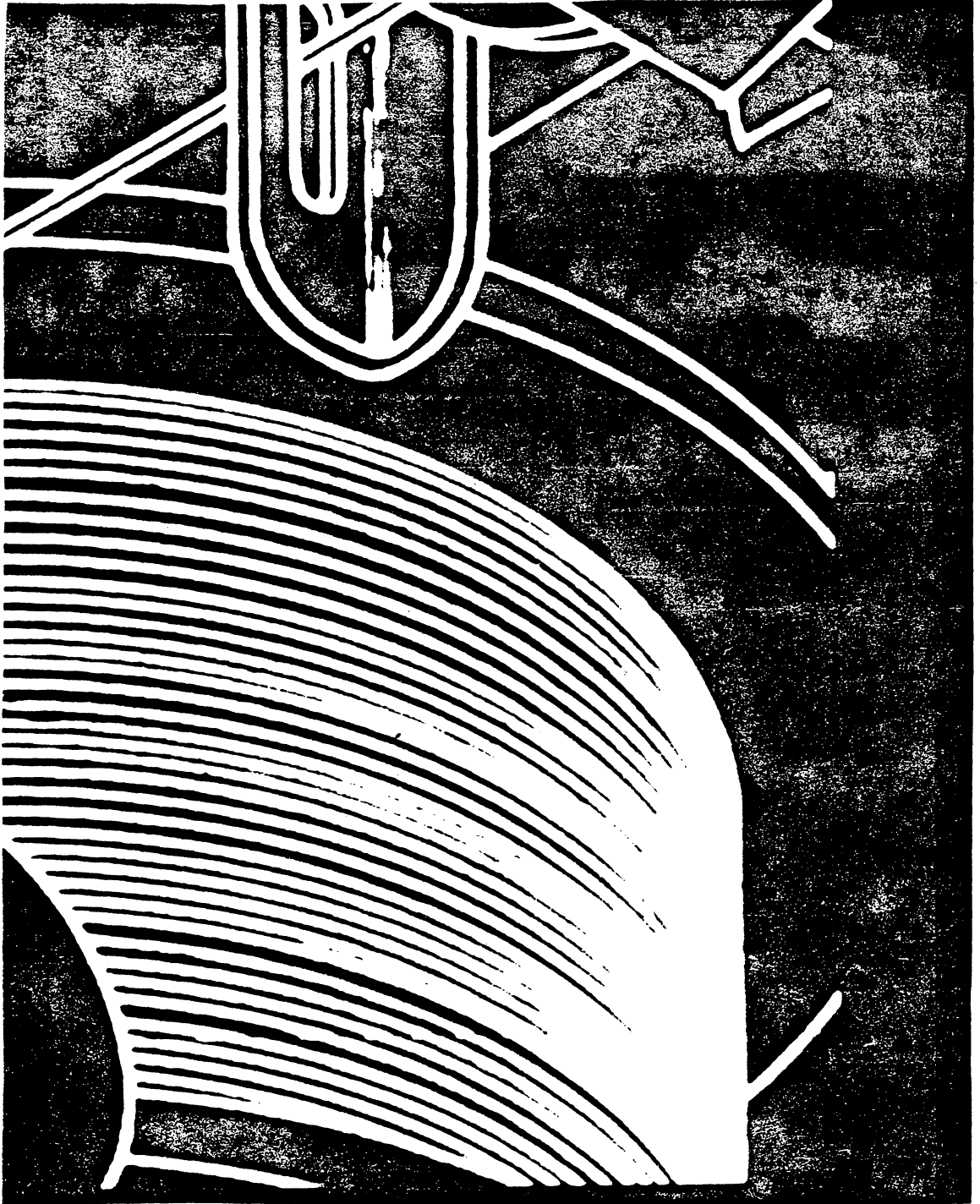


Figure 2: Close up of coil section

0	0	0	4	78	67	34	35	89	16	0	0	0	0	0
0	0	0	4	80	67	78	41	53	6	0	0	0	0	0
0	0	0	5	89	84	92	51	62	8	0	0	0	0	0
0	0	0	5	91	101	118	59	85	14	0	0	0	0	0
0	0	0	4	84	88	125	61	92	16	0	0	0	0	0
0	0	0	4	71	64	130	66	95	14	0	0	0	0	0
0	0	0	3	57	53	127	62	78	10	0	0	0	0	0
0	0	0	3	46	47	136	64	85	10	0	0	0	0	0
0	0	0	4	74	71	147	69	104	16	0	0	0	0	0
0	0	0	5	87	91	151	70	107	17	0	0	0	0	0
0	0	0	5	95	106	155	72	109	16	0	0	0	0	0
0	0	0	5	88	103	153	74	129	23	0	0	0	0	0
0	0	0	5	90	106	159	76	143	28	0	0	0	0	1
0	0	0	5	95	112	162	75	135	25	0	0	0	0	22
0	0	0	5	94	107	164	75	129	23	0	0	0	12	101
0	0	0	5	93	100	163	77	139	26	0	1	15	97	209
2	0	0	5	99	109	166	78	143	27	1	21	105	214	201
60	17	3	7	106	115	168	85	144	30	33	132	221	199	79
200	145	74	46	145	131	179	117	174	98	165	227	186	73	9
125	195	208	192	221	189	219	204	237	227	224	147	48	8	24
13	44	107	176	223	238	245	242	235	193	102	25	9	44	142
1	1	7	30	74	113	125	116	94	41	12	31	100	191	225
42	14	3	1	3	5	6	6	10	26	72	168	222	201	114
196	141	77	32	21	24	33	49	109	171	214	204	130	56	10
158	208	211	183	165	172	194	210	233	216	152	66	12	2	0
20	50	107	143	179	189	184	181	152	74	19	3	0	0	0
0	1	5	13	27	31	29	27	17	4	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
44	20	6	1	0	0	0	0	0	0	0	0	0	0	0
200	159	101	43	15	4	1	0	0	0	0	0	0	0	0
132	184	212	196	144	90	40	15	3	1	0	0	0	0	0
83	69	95	137	192	209	192	142	83	32	8	2	0	0	0
212	189	136	79	66	90	131	183	204	177	119	61	23	6	1
107	155	207	210	176	114	60	56	91	152	199	204	164	98	37
36	26	62	113	157	199	199	161	105	63	63	118	184	211	184
178	117	60	32	27	55	117	176	196	187	135	73	61	96	156
163	200	202	172	107	48	29	32	51	119	186	198	172	114	65

Figure 3: Bobbin, after filtering and sampling, before DAT

1	1	9	1	*7*	*7*	9	1	*7*	1	9	1	1	1	1
1	1	9	1	*7*	*7*	1	1	*7*	1	9	1	1	1	1
1	1	9	1	1	*7*	1	1	1	1	9	1	1	1	1
1	1	9	1	1	*7*	1	*7*	1	1	9	1	1	1	1
1	1	9	1	1	*7*	1	*7*	1	1	9	1	1	1	1
1	1	9	1	1	*7*	1	*7*	1	1	9	1	1	1	1
1	1	9	1	1	*7*	1	*7*	1	1	9	1	1	1	1
1	1	9	1	1	*7*	1	*7*	1	1	9	1	1	1	1
1	1	9	1	1	*7*	1	*7*	1	1	9	1	1	1	1
1	1	9	9	1	*7*	1	*7*	1	1	9	1	1	9	9
1	1	9	9	1	*7*	1	*7*	1	1	9	1	1	9	9
1	1	9	9	1	*7*	1	*7*	1	1	9	9	9	9	1
9	9	9	9	1	*7*	1	*7*	1	1	9	9	1	*7*	*11*
9	9	10	9	1	*7*	1	*7*	9	9	9	1	*7*	*7*	*7*
7	1	10	9	9	*11*	9	*11*	9	9	9	*7*	*7*	*7*	1
7	*7*	*11*	1	1	*7*	1	*7*	1	*7*	*11*	*11*	*7*	1	9
7	*7*	*11*	*7*	*7*	8	*7*	8	*7*	*7*	*11*	*7*	9	9	9
9	1	*7*	*7*	*7*	8	*7*	*7*	*7*	*7*	9	9	9	9	*7*
9	9	9	9	9	1	1	9	9	9	9	1	*7*	*7*	*7*
1	9	9	9	9	9	9	9	9	9	1	*7*	*7*	*7*	*7*
7	*7*	*7*	1	9	9	9	9	1	*7*	*7*	*7*	*7*	1	9
7	*7*	*7*	*7*	*7*	*7*	*7*	*7*	*7*	*7*	*7*	*7*	*7*	1	9
9	1	*7*	*7*	*7*	*7*	*7*	*7*	*7*	*7*	1	1	9	9	9
9	9	9	9	9	9	9	9	9	9	9	9	9	1	1
1	9	9	9	9	9	9	9	9	9	9	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1	1	1	1	1
9	9	9	9	1	1	1	1	1	1	1	1	1	1	1
1	9	9	9	9	9	9	9	1	1	1	1	1	1	1
7	*7*	*7*	1	9	9	9	9	9	9	9	1	1	1	1
7	*7*	*7*	*7*	*7*	*7*	1	9	9	9	9	9	1	1	1
1	1	1	1	*7*	*7*	*7*	*7*	*7*	1	9	9	9	9	9
7	*7*	1	1	1	1	1	1	*7*	*7*	*7*	1	1	9	9
1	1	*7*	*7*	*7*	1	1	1	1	*7*	*7*	*7*	*7*	*7*	1
9	9	9	1	1	*7*	*7*	*7*	1	1	1	1	*7*	*7*	*7*
7	1	9	9	9	9	1	*7*	*7*	*7*	1	1	*7*	*7*	*7*
7	*7*	*7*	*7*	1	9	9	9	9	1	1	*7*	1	1	1

Figure 5: Topographic labelling of bobbin

117	208	166	62	77	196	178	62	131	235	254	255	255	253	234
61	110	211	177	61	84	185	143	70	146	238	254	255	255	252
175	81	108	199	179	77	74	162	149	61	155	244	255	255	255
214	196	86	89	201	189	63	56	175	95	52	188	249	255	255
179	220	201	95	91	198	178	67	87	149	54	73	204	252	255
227	196	219	205	91	87	192	177	52	82	131	53	94	225	254
221	233	198	218	203	100	91	196	153	60	101	146	90	152	243
205	223	236	214	241	214	106	97	200	172	64	123	188	136	199
230	216	236	247	244	212	202	92	98	207	151	73	176	206	141
152	231	239	227	236	186	210	204	104	125	214	163	90	193	196
110	146	227	220	211	236	218	239	213	109	139	219	147	111	213
212	136	135	214	224	230	245	251	249	204	98	155	213	109	152
199	223	145	134	220	233	228	251	248	247	188	93	179	194	109
76	197	231	165	131	212	243	253	254	255	246	183	113	203	188
107	92	186	229	153	119	213	252	255	255	254	244	176	151	224
211	126	75	165	225	152	116	214	253	255	254	252	242	179	191
150	216	142	71	167	222	146	120	225	254	255	255	254	231	153
67	145	217	148	71	148	222	155	155	236	254	255	255	252	206
163	81	133	213	152	77	184	226	158	168	240	255	255	255	242
214	183	86	131	212	163	107	192	226	159	176	243	255	255	254
91	206	191	93	121	213	177	105	199	222	136	183	247	255	255
41	88	204	200	103	114	203	183	136	214	208	134	206	252	255
151	48	88	202	206	105	101	210	179	142	222	200	148	229	254
217	165	65	85	200	206	94	122	215	170	161	230	196	188	244
132	216	192	73	82	201	202	99	138	221	178	195	238	199	217
116	157	220	165	57	109	213	188	93	175	226	181	231	236	201
207	165	128	197	188	103	94	194	199	130	162	227	243	251	240
162	223	177	137	210	208	95	91	203	207	123	191	248	255	254
155	167	226	198	151	214	209	101	92	207	204	130	205	252	255
229	182	159	221	204	157	214	204	86	104	219	197	150	230	254
184	230	176	148	217	202	148	205	199	99	137	226	198	187	245
121	160	227	181	159	219	192	127	208	202	101	151	231	194	218
208	132	164	231	207	165	218	175	128	210	194	101	187	233	193
200	214	141	160	229	211	189	222	176	145	222	177	112	212	231
121	186	217	143	157	232	221	199	229	198	184	226	163	146	232
192	110	167	216	161	173	233	218	192	233	199	195	227	151	166
249	204	123	162	224	173	174	233	217	212	237	200	207	213	130
255	252	213	128	175	227	180	174	234	224	224	235	190	234	210
253	255	252	218	150	167	223	176	185	238	233	245	243	251	248
211	251	255	253	223	138	176	231	191	183	240	253	255	255	255
78	198	249	255	253	230	169	184	229	175	195	248	255	255	255
17	66	177	244	255	254	230	148	178	223	178	216	252	255	255
95	20	42	158	242	255	253	220	157	214	233	206	238	254	255
209	115	27	37	156	242	255	253	233	194	223	238	224	249	255

Figure 6: Coil, after filtering and sampling, before DAT

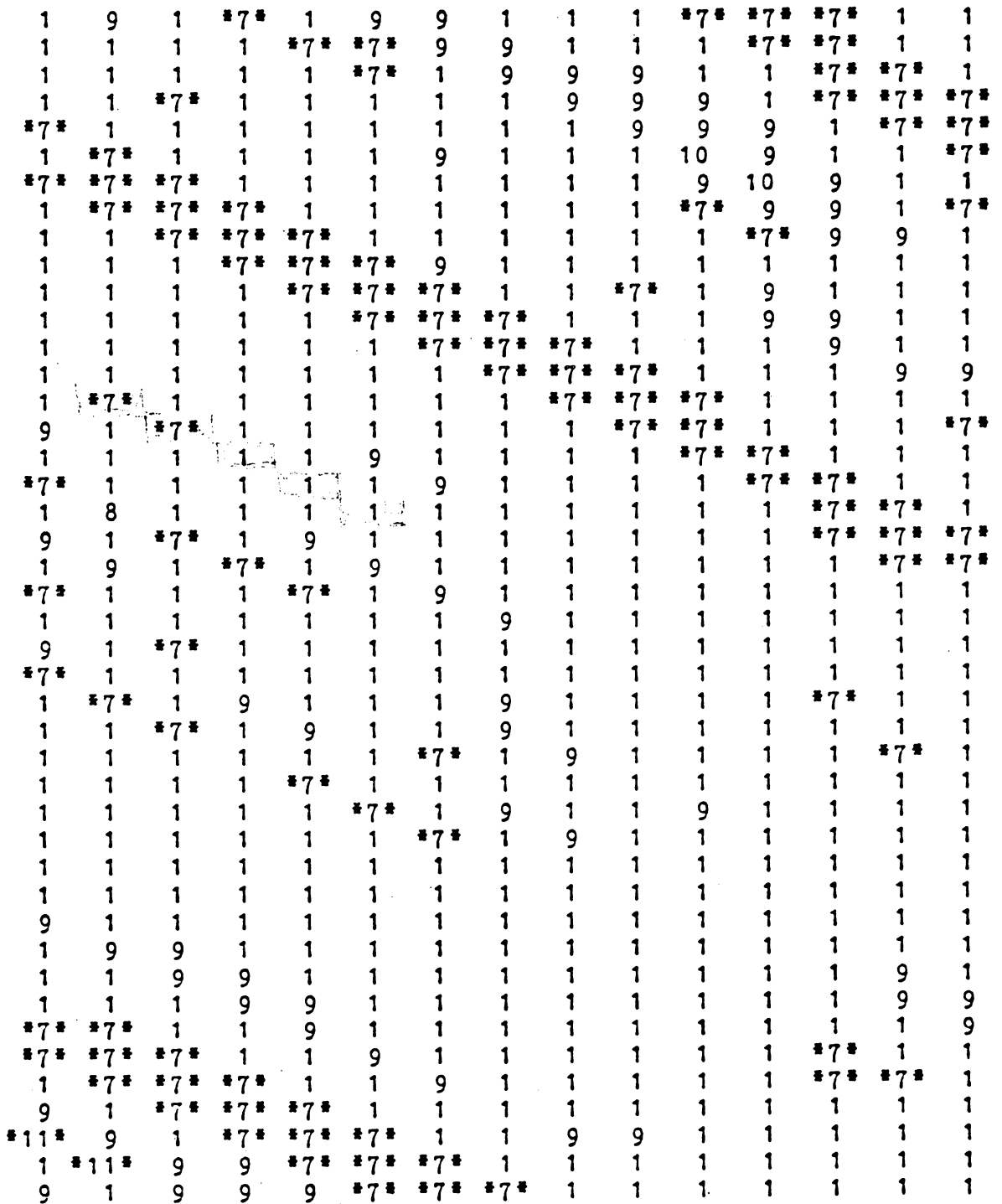


Figure 7: Topographic labelling of coil

```

0 208 166 0 0 196 178 0 0 235 254-255-255-253-234
0 0 211 177 0 0 185 143 0 0 238-254-255-255-252
175 0 0 199 179 0 0 162 149 0 0 244-255-255-255
214 196 0 0 201 189 0 0 175 0 0 188 249-255-255
0 220 201 0 0 198 178 0 0 149 0 0 204 252-255
227 0 219 205 0 0 192 177 0 0 131 0 0 225 254
-221 233 0 218 203 0 0 196 153 0 0 146 0 0 243
-205-223-236-214 241 214 0 0 200 172 0 0 188 0 0
230-216-236-247 244-212 202 0 0 207 0 0 176 206 0
0 231 239-227-236 0-210 204 0 0 214 163 0 193 196
0 0 227-220-211-236-218 239 213 0 0 219 0 0 213
212 0 0 214-224-230-245-251 249 204 0 0 213 0 0
199 223 0 0 220 233-228-251-248 247 0 0 179 194 0
0 197 231 0 0 212 243-253-254-255 246 0 0 203 188
0 0 186 229 0 0 213 252-255-255-254 244 0 0 224
211 0 0 0 225 0 0 214 253-255-254-252 242 0 0
0 216 0 0 167 222 0 0 225 254-255-255-254-231 0
0 0 217 0 0 0 222 0 0 236-254-255-255 252-206
163 0 0 213 0 0 184 226 0 0 240-255-255-255 242
214 183 0 0 212 0 0 192 226 0 0 243-255-255-254
0 206 191 0 0 213 177 0 199 222 0 0 247-255-255
0 0 204 200 0 0 203 183 0 214 208 0-206 252-255
151 0 0 202 206 0 0 210 0 0 222 0 0-229-254
217 165 0 0 200 206 0 0 215 0 0 230 0 0-244
0 216 192 0 0 201 202 0 0 221 0 0 238 0-217
0 0 220 165 0 0 213 188 0 0 226 0-231-236-201
207 0 0 197 188 0 0 194 199 0 0 227-243-251-240
0 223 0 0 210 208 0 0 203 207 0 0 248-255-254
0 0 226 0 0 214 209 0 0 207 204 0-205 252-255
229 0 0 221 204 0 214 204 0 0 219 0 0-230-254
0 230 0 0 217-202 0 205 199 0 0 226 0 0-245
0 0 227 0 0 219 0 0 208 202 0 0 231 0-218
208 0 0 231-207 0 218 0 0 210 194 0 0 233 0
200 214 0 0 229-211 0 222 0 0 222 0 0 212-231
0 186 217 0 0 232-221 0 229 0 0 226 0 0 232
0 0 0 216 0 0 233-218 0 233 0 0 227 0 0
249-204 0 0 224 0 0 233-217-212 237 0-207 213 0
-255 252-213 0 0 227 0 0 234-224-224-235 0 234-210
253-255 252-218 0 0 223 0 0 238-233-245-243-251 248
211 251-255 253-223 0 0 231 0 0 240-253-255-255-255
0 198 249-255 253 230 0 0 229 0 0 248-255-255-255
0 0 177 244 255 254 230 0 0 223 0-216-252-255-255
0 0 0 0 242 255 253-220 0-214 233-206-238-254-255
209 0 0 0 156 242 255 253-233 0-223-238-224-249-255

```

Figure 8: Result of double adaptive threshold (DAT) algorithm on coil

0	0	0	0	0	0	0	0	0	0	0	254	255	255	253	234
0	0	0	0	0	0	0	0	0	0	0	0	254	255	255	252
0	0	0	0	0	0	0	0	0	0	0	0	244	255	255	255
0	0	0	0	0	0	0	0	0	0	0	0	0	249	255	255
0	0	0	0	0	0	0	0	0	0	0	0	0	0	252	255
227	0	0	0	0	0	0	0	0	0	0	0	0	0	0	254
221	233	0	0	0	0	0	0	0	0	0	0	0	0	0	0
205	223	236	214	241	0	0	0	0	0	0	0	0	0	0	0
0	216	236	247	244	212	0	0	0	0	0	0	0	0	0	0
0	0	239	227	236	0	210	0	0	0	0	0	0	0	0	0
0	0	0	220	211	236	218	239	0	0	0	0	0	0	0	0
0	0	0	0	224	230	245	251	249	0	0	0	0	0	0	0
0	0	0	0	0	233	228	251	248	247	0	0	0	0	0	0
0	0	0	0	0	0	243	253	254	255	246	0	0	0	0	0
0	0	0	0	0	0	0	252	255	255	254	244	0	0	0	0
0	0	0	0	0	0	0	0	253	255	254	252	242	0	0	0
0	0	0	0	0	0	0	0	0	254	255	255	254	231	0	0
0	0	0	0	0	0	0	0	0	0	254	255	255	252	206	0
0	0	0	0	0	0	0	0	0	0	240	255	255	255	242	0
0	0	0	0	0	0	0	0	0	0	0	243	255	255	254	0
0	0	0	0	0	0	0	0	0	0	0	0	247	255	255	0
0	0	0	0	0	0	0	0	0	0	0	0	0	206	252	255
0	0	0	0	0	0	0	0	0	0	0	0	0	0	229	254
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	244
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	217
0	0	0	0	0	0	0	0	0	0	0	0	0	0	236	201
0	0	0	0	0	0	0	0	0	0	0	0	0	243	251	240
0	0	0	0	0	0	0	0	0	0	0	0	0	248	255	254
0	0	0	0	0	0	0	0	0	0	0	0	0	205	252	255
0	0	0	0	0	0	0	0	0	0	0	0	0	0	230	254
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	245
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	218
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
249	204	0	0	0	0	0	0	217	212	237	0	0	0	0	0
255	252	213	0	0	0	0	0	234	224	224	235	0	234	210	0
0	255	252	218	0	0	0	0	0	238	233	245	243	251	248	0
0	0	255	253	0	0	0	0	0	0	240	253	255	255	255	0
0	0	0	0	0	0	0	0	0	0	0	248	255	255	255	0
0	0	0	0	0	0	0	0	0	0	0	216	252	255	255	0
0	0	0	0	0	0	0	0	0	0	233	206	238	254	255	0
0	0	0	0	0	0	0	253	233	0	223	238	224	249	255	0

Figure 9: Regions extracted from coil

```

0-208 166 0 0-196 178 0 0-235 0 0 0 0 0
0 0-211 177 0 0-185 143 0 0-238 0 0 0 0
175 0 0-199 179 0 0-162 149 0 0 0 0 0
-214 196 0 0-201 189 0 0-175 0 0-188 0 0 0
0-220 201 0 0-198 178 0 0-149 0 0-204 0 0
0 0-219 205 0 0-192 177 0 0-131 0 0-225 0
0 0 0-218-203 0 0-196 153 0 0-146 0 0-243
0 0 0 0 0-214 0 0-200 172 0 0-188 0 0
-230 0 0 0 0 0-202 0 0-207 0 0 176-206 0
0-231 0 0 0 0 0-204 0 0-214 163 0-193-196
0 0-227 0 0 0 0 0-213 0 0-219 0 0-213
-212 0 0-214 0 0 0 0 0-204 0 0-213 0 0
199-223 0 0-220 0 0 0 0 0 0 0 179-194 0
0 197-231 0 0-212 0 0 0 0 0 0 0-203-188
0 0 186-229 0 0-213 0 0 0 0 0 0-224
-211 0 0 0-225 0 0-214 0 0 0 0 0 0
0-216 0 0 167-222 0 0-225 0 0 0 0 0 0
0 0-217 0 0 0-222 0 0-236 0 0 0 0 0
163 0 0-213 0 0 184-226 0 0 0 0 0 0
-214 183 0 0-212 0 0 192-226 0 0 0 0 0 0
0-206 191 0 0-213 177 0 199-222 0 0 0 0 0
0 0-204 200 0 0-203 183 0-214 208 0 0 0 0
151 0 0-202-206 0 0-210 0 0-222 0 0 0 0
-217 165 0 0 200-206 0 0-215 0 0-230 0 0 0
0-216 192 0 0 201-202 0 0-221 0 0-238 0 0
0 0-220 165 0 0-213 188 0 0-226 0-231 0 0
-207 0 0-197 188 0 0-194 199 0 0-227 0 0 0
0-223 0 0-210 208 0 0-203 207 0 0 0 0 0
0 0-226 0 0-214 209 0 0-207 204 0 0 0 0
-229 0 0-221 204 0-214 204 0 0-219 0 0 0 0
0-230 0 0-217 202 0-205 199 0 0-226 0 0 0
0 0-227 0 0-219 0 0-208 202 0 0-231 0 0
-208 0 0-231 207 0-218 0 0-210 194 0 0-233 0
200-214 0 0-229 211 0-222 0 0-222 0 0 212-231
0 186-217 0 0-232 221 0-229 0 0-226 0 0 232
0 0 0-216 0 0-233 218 0-233 0 0-227 0 0
0 0 0 0-224 0 0-233 0 0 0 207-213 0
0 0 0 0 0-227 0 0 0 0 0 0 0 0
-253 0 0 0 0 0-223 0 0 0 0 0 0 0 0
211-251 0 0 223 0 0-231 0 0 0 0 0 0 0 0
0 198-249-255-253 230 0 0-229 0 0 0 0 0 0
0 0 177 244-255-254 230 0 0-223 0 0 0 0 0
0 0 0 0 242-255 253 220 0-214 0 0 0 0 0
-209 0 0 0 156 242-255 0 0 0 0 0 0 0 0

```

Figure 10: Output of line tracker on coil (regions removed)

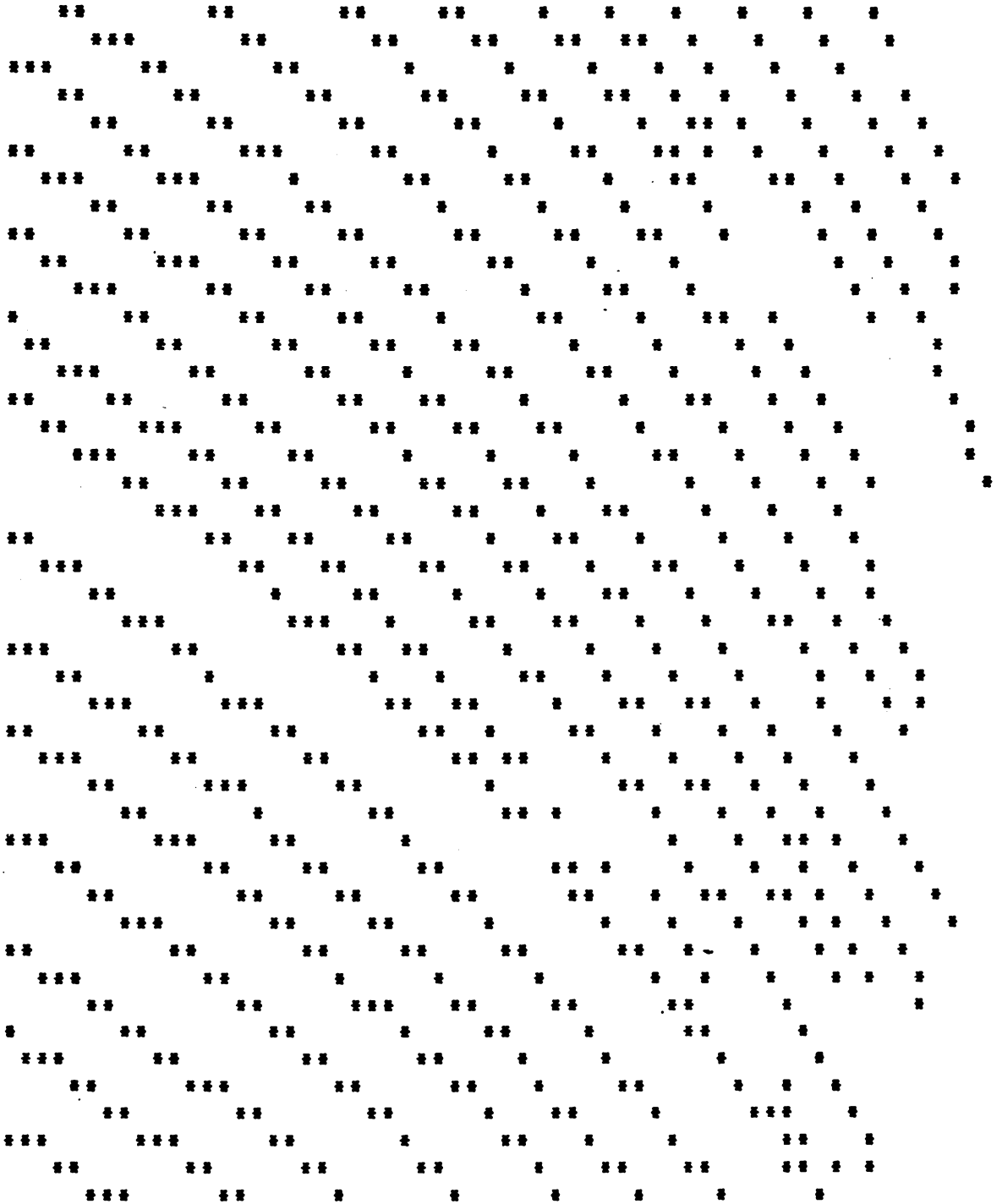


Figure 11: Same as Figure 10 but showing only tracked lines

Chapter V

CONCLUSION

The difficulties associated with real digitized line drawing images, as opposed to artificially generated images, are significant. Many papers on line drawings have used binary images as their starting point, but the present work shows that producing a good binary image from a real digitized gray tone image is highly nontrivial. Any practical production algorithm must clearly begin with noisy gray tone images. Standard thinning, sharpening, and medial axis transformations were tried, and (despite occasional exemplary performances) none were found to be uniformly good.

For certain industrial applications it may not be practical or economic to connect the imaging equipment to mainframe computers with high speed transmission lines. Thus the low level processing (extraction of the lines and regions) must be done with limited local compute power and storage. The DAT algorithm presented here meets these requirements by being reasonably cheap and noniterative, although several sequential passes through the image are required. Numerous experiments on different types of line drawings also strongly suggest that a good algorithm must be

adaptive, and the adaptive nature of the DAT is crucial to its success.

The overall problem being addressed here for real digitized gray tone images of line drawings consists of (1) recognition and extraction of lines and solid regions (textured regions are not considered), (2) the compression and transmission of the line and region data, and (3) the high level representation (e.g., as graphics primitives) of the line drawing (lines, curves, regions). This thesis represents a solution to (1) requiring only limited local compute power and storage. The next goals are an economic solution to (2) using slow transmission speeds (300 baud), and a powerful and sophisticated high level encoding of the line drawing as graphical primitives in, e.g., CDC's TUTOR system.

REFERENCES

1. Aoki, M., 'An Algorithm for Compressing Binary Pictures Using Uniform Rectangular Regions', Unpublished Conference Proceedings.
2. Black, W., T. P. Clement, J. F. Harris, B. Llewellyn, G. Preston, 'A General Purpose Follower for Line-Structured Data', Unpublished Conference Proceedings.
3. Burr, D. J., 'Elastic Matching Of Line Drawings', IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-3, No. 6, Nov. 1981, pp 708-713.
4. Chow, C. K., B. L. Deekshatulu, L. S. Loh, 'Some Computer Experiments in Picture Processing for Data Compaction', Computer Graphics and Image Processing, 3, 1974, pp 203-214.
5. Clement, T. P., 'The Extraction of Line Structured Data from Engineering Drawings', Pattern Recognition, Vol. 14, No. 16, 1981, pp 43-52.
6. Davisson, L. D., 'The Theoretical Analysis of Data Compression Systems', Proceedings of the IEEE, 56, 1968, pp 176-186.
7. Deutsch, E. S., 'Thinning Algorithms on Rectangular, Hexagonal and Triangular Arrays', Commn. of ACM, 15, No. 9, 1972, pp 827-837.
8. Dudani, S. A., 'Region Extraction Using Boundary Following', Unpublished Conference Proceedings.
9. Ehrich, R. W., 'A Symmetric Hysteresis Smoothing Algorithm that Preserves Principal Features', Computer Graphics and Image Processing, Vol. 8, 1978, pp 121-126.
10. Freeman, H., 'Computer Processing of Line Drawings', Computing Surveys, Vol. 6, No. 1, March, 1974, pp 59-97.
11. Freeman, H., 'On the Encoding of Arbitrary Geometric Configurations', IRE Tran. Electron. Comput., Vol. EC-10, June, 1961, pp 260-268.
12. Fulford, M. C., 'The Fastrak Automatic Digitizing System', Pattern Recognition, Vol. 14, No. 16, 1981, pp 65-74.

13. Gluss, B., 'A Line Segment Curve Fitting Algorithm Related to Optimal Encoding of Information', *Information and Control*, 5, (3), Sept., 1962, pp 261-267.
14. Graham, D. N., 'Image Transmission by Two Dimensional Contour Coding', *Proceedings of IEEE*, 55, (3), March, 1967, pp 336-346.
15. Haralick, R. M., L. T. Watson, 'A Facet Model for Image Data', *Computer Graphics and Image Processing*, 15, 1981, pp 113-129.
16. Haralick, R. M., L. T. Watson, T. J. Laffey, 'The Topographic Primal Sketch', *Internat. J. Robotics Res.*, 2, 1983, pp 50-72.
17. Holdermann, F., H. Kazmierczak, 'Preprocessing of Gray Scale Pictures', *Computer Graphics and Image Processing*, 1, (1), April, 1972, pp 66-80.
18. Huang, T. S., 'Digital Transmission of Halftone Pictures', *Computer Graphics and Image Processing*, 3, 1974, pp 195-202.
19. Jarvis, J. F., 'Feature Recognition in Line Drawings Using Regular Expressions', *Unpublished Conference Proceedings*.
20. Kim, C. E., 'On Cellular Straight Line Segments', *Computer Graphics and Image Processing*, 18, 1982, pp 369-381.
21. Kortman, C. M., 'Redundancy Reduction - a Practical Method of Data Compression', *Proceedings of IEEE*, 55, 1967, pp 253-263.
22. Leberl, F. W., D. Olson, 'Raster Scanning for Operational Digitizing of Graphical Data', *Photogrammetric Engineering and Remote Sensing*, Vol. 48, No. 4, April, 1982, pp 615-627.
23. Lee, H. C., K. S. Fu, 'Using the FFT to Determine Digital Straight Line Chain Codes', *Computer Graphics and Image Processing*, 18, 1982, pp 359-368.
24. Levi, G., U. Montonari, 'A Gray Weighted Skeleton', *Information and Control*, 17, 1970, pp 169-182.

25. Martelli, A., 'Edge Detection Using Heuristic Search Methods', Computer Graphics and Image Processing, 1, (2), August, 1972, pp 169-182.
26. Maxwell, P. C., 'The Perception and Description of Line Drawings by Computer', Computer Graphics and Image Processing, 1, (1), April, 1972, pp 31-46.
27. Merrill, R. D., 'Representation of Contours and Regions for Efficient Computer Search', Comm. ACM, 16, (2), Feb., 1973, pp 69-82.
28. Montonari, U., 'Continuous Skeletons from Digitized Images', J. ACM, 16, (14), Oct., 1969, pp 534-549.
29. Murthy, L. S. N., K. J. Udupa, 'A Search Algorithm for Skeletization of Thick Patterns', Computer Graphics and Image Processing, Vol. 3, 1974, pp 247-259.
30. Nagao, M., T. Matsuyama, 'Edge Preserving Smoothing', Computer Graphics and Image Processing, 9, 1979, pp 394-407.
31. Oliviero, A., G. Scarpetta, 'A New Approach to Contour Coding', Computer Graphics and Image Processing, 15, 1981, pp 87-92.
32. Pavlidis, T., 'Techniques for Optimal Compaction of Pictures and Maps', Computer Graphics and Image Processing, 3, 1974, pp 215-224.
33. Pavlidis, T., 'Segmentation of Pictures and Maps Through Functinal Approximations', Computer Graphics and Image Processing, 1, 1972, pp 360-372.
34. Peleg, S., A. Rosenfeld, 'A Min Max Medial Axis Transformation' IEEE Tran. on Pattern Analysis and Machine Intelligence, Vol. PAMI-1, 1979, pp 88-89.
35. Ramachandran, K., 'Coding Method for Vector Representation of Engineering Drawings', Proceedings of the IEEE, Vol 68, No. 7, July, 1980, pp 813-817.
36. Ramer, U., 'An Iterative Procedure for the Polygonal Approximation of Plane Curves', Computer Graphics and Image Processing, Vol. 1, 1972, pp 244-256.
37. Rosenfeld, A., 'Connectivity in Digital Pictures', J. ACM, 17, 1970, pp 146-160.

38. Rosenfeld, A., 'Arcs and Curves in Digital Pictures', J. ACM, 20, 1, Jan., 1973, pp 81-87.
39. Rosenfeld, A., J. L. Pfaltz, 'Sequential Operations in Digital Picture Processing', J. ACM, Vol. 13, No. 4, 1966, pp 471-494.
40. Rosenfeld, A., 'Digital Straight Line Segments', IEEE Trans. Comput., C-23, 1974, pp 1264-1269.
41. Skiansky, J., V. Gonzales, 'Fast Polygonal Approximation of Digitized Curves', Pattern Recognition, Vol. 12, 1981, pp 327-331.
42. Suetens, P., P. Dierckx, R. Riessens, A. Oosterlinck, 'A Semiautomatic Digitization Method and the use of Spline Functions in Processing Line Drawings', Computer Graphics and Image Processing, 15, 1981, pp 390-400.
43. Ting, D., B. Prasada, 'Digital Processing Techniques for Encoding of Graphics' Proceedings of the IEEE, Vol. 68, No. 7, July, 1980, pp 756-767.
44. Triendl, E. E., 'Skeletonization of Noisy Handrawn Symbols Using Parallel Operations', Pattern Recognition, 2, (3), Sept., 1970, pp 215-226.
45. Wang, S., A. Y. Wu, A. Rosenfeld, 'Image Approximation from Gray Scale Medial Axes', IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI-3, No. 6, November, 1981, pp 687-697.
46. Watson, L. T., T. J. Laffey, R. M. Haralick, 'Topographic Classification of Digital Image Intensity Surfaces Using Generalized Splines and the Discrete Cosine Transformation', Computer Vision, Graphics and Image Processing., to appear.
47. Weszka, J., 'A Survey of Threshold Selection Techniques', Computer Graphics and Image Processing, 7, 1978, pp 259-265.
48. Williams, C. M., 'An Efficient Algorithm for the Piecewise Approximation of Planar Curves', Computer Graphics and Image Processing, Vol. 8, 1978, pp 286-293.
49. Wintz, P. A., 'Transform Picture Coding', IEEE Proc., July, 1972, pp 809-820.

50. Woetzel, G., 'A Fast and Economic Scan to Line Conversion Algorithm', Computer Graphics, Vol. 12, 1978, pp 125-129.
51. Wu, L. D., 'On the Chain Code of a Line', IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI-4, No. 3, May, 1982, pp 347-353.
52. Yalabik, N., D. B. Cooper, 'Compression of Contour Data through Exploiting Curve-To-Curve Dependence', Unpublished Conference Proceedings.

Appendix A

PROGRAM LISTINGS FOR ALGORITHMS

The following appendix contains the program code for all the algorithms that were discussed. The routines are written in RATFOR and conform to the specifications of the GIPSY system in use at the SDA laboratory at Virginia Tech. All the algorithms were implemented as GIPSY operators.

DESCRIPTION OF DAT.

*DAT Double threshold a numeric image.

VERSION: A.01 DATE: 06-01-80 CONTACT: K. ARVIND

ACTION: With a user specified factor and window size 'DAT' thresholds a numeric image and produces a numeric output image. The cutoff for each pixel is computed by multiplying the factor with the average of all pixels greater than or equal to the double threshold in the window centred over the current pixel. If the value of the current pixel is greater than its cutoff then the output pixel is made equal to the input. If it is less than the cutoff but greater than the texture threshold then also it is made equal to the input but with a negative sign, signifying a region or texture pixel. If none of these conditions are satisfied then the output pixel is made zero.

SOURCE: Disk, SIF.

DESTINATION: Disk, SIF.

FLAGS: None.

- QUESTIONS: (1) User is asked which bands to threshold on.
Note: this is not asked if the image file contains only one band.
- (2) User is asked for the texture threshold. If the current pixel is below the computed cutoff but above or equal to this threshold then it is marked as texture by assigning to it a value equal to the negative of the input pixel.
- (3) User is asked for the factor to be used in computing the cutoff. For each pixel in consideration the cutoff is computed by calculating the mean of the pixels in a user defined window (centered over the pixel in consideration) and then multiplying the mean by this factor.
- (4) User is asked for the column size of the window to be used. This has to be an odd integer.
- (5) User is asked for the row size of the window to be used. This has to be an odd integer.
- (6) User is asked for the double threshold to be used. Only pixel values above or equal to this will be used in computing the mean over the window.

COMMAND STRING EXAMPLE:

DAT OUT.SIF < IN.SIF

```
IMAGE IN.SIF HAS <NNUM> NUMERIC BANDS
HOW MANY DO YOU WISH TO SELECT (>0) -- 2
SELECTION 1 (1-<NNUM>) -- 1
SELECTION 2 (1-<NNUM>) -- 3
ENTER THRESHOLD FOR TEXTURE ( D=256, 1-256 ) - 200
ENTER FACTOR FOR COMPUTING CUTOFF ( >0.0 ) -- 1.25
ENTER COLUMN SIZE FOR WINDOW ( D=3 ) --
ENTER ROW SIZE FOR WINDOW ( D=3 ) --
ENTER DOUBLE THRS ( D=5 ) --
```

Bands '1' and '3' of 'IN.SIF' will be thresholded using
(1.25 * window mean) for the cutoff. OUT.SIF will be a two
band numeric image.

COMMENTS: For a detailed explanation refer to 'EXTRACTION OF
LINES AND REGIONS FROM GREY TONE IMAGES OF LINE
DRAWINGS' : Masters Thesis submitted by Krishna
Arvind to the Faculty of Computer Science in
January 1984.

RUNFILE FOR TESTING DAT

\$! TESTING DAT

\$ \$ \$ \$ \$

! CREATE A SIMPLE NUMERIC IMAGE

```
EXSIF
OPEN TESTIN.SIF
Y
0 0 0 0
8 5 5 1
1 6 256 5
5 1 0 255
1 0 0 0
0 0 0 0 0
0 190 190 190 0
0 210 210 210 0
0 240 240 240 0
0 0 0 0 0
DONE
```

\$ \$ \$ \$ \$

! DO THE THRESHOLDING NOW

```
DAT TESTIN.SIF > TESTOUT.SIF
200
1.0
3
3
5
```

\$ \$ \$ \$ \$

! CREATE A TEST INPUT TO COMPARE THE OUTPUT WITH.

```
EXSIF
OPEN COMPAR.SIF
Y
0 0 0 0
9 5 5 1
1 6 511 5
5 1 -255 255
1 0 0 0
0 0 0 0 0
0 0 0 0 0
0 -210 -210 -210 0
0 240 240 240 0
0 0 0 0 0
DONE
```

\$ \$ \$ \$ \$

! COMPARE THE OUTPUT WITH THE TEST INPUT. THE DIFFERENCE SHOULD BE ZERO.

```
RMS TT < TESTOUT.SIF, COMPAR.SIF
```

```
$ !  
$ ! DELETE TEMPORARY FILES  
$ !  
$      DELETE TESTIN.SIF  
$      DELETE COMPAR.SIF  
$      DELETE TESTOUT.SIF  
$      EXIT
```

LINK FILE FOR DAT.
\$ GENCMD DAT DDAT, DATASC, DAT

```

#--DDAT          DRIVER FOR THE DAT GIPSY OPERATOR.          MID
#
# IDENTIFICATION
#
#   TITLE          DDAT
#   AUTHOR         KRISHNA ARVIND
#   VERSION        A.01
#   DATE           20-DEC-1983
#   LANGUAGE       RATFOR
#   SYSTEM         VAX 11/780
#
# PURPOSE
#
#   THIS ROUTINE IS THE DRIVER FOR THE DAT GIPSY OPERATOR.
#
# ENTRY POINT
#
#   CALL DDAT( IWORK, * )
#
# ARGUMENT LISTING
#
#   IWORK  ARRAY          COMMON STORAGE AREA
#   ALTRET INTEGER        ALTERNATE RETURN.
#
# INCLUDE FILES/COMMONS
#
#   MACA1  INCLUDE      GIPSY GENERAL SYMBOL DEFINITIONS
#   GIPCOM COMMON      GIPSY COMMAND LINE COMMON DATA
#   ERROR  COMMON      ERROR TRACE STACK COMMON DATA
#
# ROUTINES CALLED
#
#   PPUSH      PUSH PROGRAM NAME ONTO ERROR STACK          (GIPSY PRIMITIVE)
#   RDKINL     INITIALIZE AND ACCESS A STANDARD IMAGE      (GIPSY PRIMITIVE)
#               FILE.
#   CLOSE      CLOSE A FILE.                               (GIPSY PRIMITIVE)
#   NMBNDS     NUMBER OF BANDS TO BE SELECTED              (GIPSY PRIMITIVE)
#   GETWP      GET WORK ARRAY POINTER.                     (GIPSY PRIMITIVE)
#   BANDSP     BANDS SPECIFICATION                         (GIPSY PRIMITIVE)
#   DATASC     TO GET THE PARAMETERS FOR THE DAT OPERATOR.
#               (USER ROUTINE)
#   OSALOC     CHECK AND EXTEND USER WORK SPACE            (GIPSY OS KERNEL)
#   DAT        TO DO THE DOUBLE ADAPTIVE THRESHOLDING.
#               (USER ROUTINE)
#   PPOP       POP PROGRAM NAME FROM THE ERROR STACK
#               (GIPSY PRIMITIVE)
#   OSGIEV     GET EVENT VARIABLE (IEV) FROM OS SAVE

```

```

#                               AREA                               (GIPSY OS KERNEL)
#
# *****
#
# INCLUDE MACA1
# SUBROUTINE DDAT ( IWORK, * )
#
#                               TYPE STATEMENTS
#
# IMPLICIT INTEGER(A-Z)
#
# REAL FTR
# LOGICAL FLG, AFLAG
#
#
# INTEGER IDENT(.IDLENGTH)
# INTEGER IWORK (.ARB)
#
# INCLUDE GIPCOM
# INCLUDE ERROR
#
# EQUIVALENCE (NPPL, IDENT(.IDNPPL)), (NLIN, IDENT(.IDNLINS))
# EQUIVALENCE ( UFLAG(1), AFLAG )
#
#
#                               STACK SUBROUTINE NAME IN ERROR STACK
#
#
# CALL PPUSH ( "DDAT" )
#
#                               OPEN INPUT FILE TO GET SOME PARAMETERS.
#
# CALL RDKINL ( FD11, IDENT, .OLD, IEV, %9999 )
# CALL CLOSE (FD11)
#
#
#                               FIND OUT HOW MANY AND WHICH NUMERIC
#                               BANDS TO PROCESS.
#
# FLG = .FALSE.
# CALL NMBNDS ( FD11, IDENT, -1, NUMB, AFLAG, FLG, IEV, %9999 )
#
# NXT = 1
# IMAGS = GETWP( NXT, .INTMODE, NUMB )

```

```

CALL BANDSP ( IWORK(IMAGS), NUMB, NUMB, IDENT, -1, AFLAG, IEV, %9999 )
#
#           CALL DATASC TO DETERMINE THE THRESHOLD.
#
CALL DATASC ( IDENT, RWSIZ, CWSIZ, DBLEVL, THTXT, FTR, IEV, %9999 )
#
#
#           SET UP THE WORK ARRAY POINTERS
#
IY = GETWP( NXT, .INTMODE, NPPL )
IS = GETWP( NXT, .INTMODE, NPPL )
TOTSIZ = RWSIZ * NPPL
PSIZ = RWSIZ
MEAN = GETWP( NXT, .INTMODE, TOTSIZ )
POINT = GETWP( NXT, .INTMODE, PSIZ )
SIZ2 = NPPL
#
IF ( .OK .EQ OSALOC( NXT ))
    GOTO 9010
#
#           CALL DAT FOR PROCESSING.
#
CALL DAT ( FD11, FD01, FTR, IWORK(IY), IWORK(IS), NPPL, NLIN, IWORK(MEAN),
          NUMB, IWORK(IMAGS), RWSIZ, CWSIZ, DBLEVL, SIZ2, IWORK(POINT),
          PSIZ, THTXT, IEV, %9999 )
#
#           CLOSE FILES, POP ROUTINE NAME AND TAKE
#           NORMAL RETURN
#
CALL CLOSE ( FD11 )
CALL CLOSE ( FD01 )
#
CALL PPOP
#
RETURN
#
#
#           ABNORMAL CONDITIONS
#
9010  CONTINUE
#
#           ERROR IN LOWER OS ROUTINE
#
IEV = OSGIEV(IEV)
#
#           CLOSE INPUT FILE AND RETURN
#
9999  CONTINUE

```

```
# CALL CLOSE ( FD11 )  
CALL CLOSE ( FD01 )  
RETURN 1  
#  
END
```



```

*****
#
#
# INCLUDE MACA1
SUBROUTINE DATASC ( IDENT, RWSIZ, CWSIZ, DBLEVL, THTXT, FTR, IEV, * )
#
#
# TYPE STATEMENTS
#
# IMPLICIT INTEGER(A-Z)
#
#
# INTEGER IDENT(.IDLNGTH)
REAL FTR
#
# INCLUDE GIPCOM
INCLUDE TTCOM
#
#
#
#
# STACK SUBROUTINE NAME IN ERROR STACK
#
# CALL PPUSH ( "DATASC" )
#
#
# CHECK THAT WE HAVE FILE IN LINE FORM
#
# IF ( IDENT(.IDNCOLS) /= IDENT(.IDNPPL) !
#     IDENT(.IDNROWS) /= 1 )
#     GO TO 9000
#
#
# NOW ASK USER FOR PARAMETERS
#
#
# CALL RNGETI('ENTER THRESHOLD FOR TEXTURE.', 1, IDENT(.IDMAX)+1,
#           IDENT(.IDMAX)+1, THTXT, IEV, %9010 )
# CALL RNGETR( 'ENTER FACTOR FOR COMPUTING CUTOFF.', 0, .LARGEREAL,
#           .9, FTR, IEV, %9010 )
# CALL RNGETI( 'ENTER COLUMN SIZE FOR WINDOW.', 1, IDENT(.IDNPPL), 3,
#           CWSIZ, IEV, %9010 )
# CALL RNGETI( 'ENTER ROW SIZE FOR WINDOW.', 1, IDENT(.IDNLINS), 3,
#           RWSIZ, IEV, %9010 )
# CALL RNGETI( 'ENTER DOUBLE THRS. LEVEL.', 0, IDENT(.IDMAX)+1, 5,
#           DBLEVL, IEV, %9010 )
#
#
# RETURN TO THE CALLING ROUTINE.
#
# CALL PPOP
RETURN

```

```
#  
#  
#  
# 9000 CONTINUE  
#  
#  
# IEV = - 5001  
# RETURN 1  
# 9010 CONTINUE  
#  
#  
#  
#  
# RETURN 1  
#  
# END
```

ERROR CONDITIONS

INPUT FILE NOT IN LINE FORMAT

ERROR IN RNGET

```

#--DAT          DOES THE DOUBLE ADAPTIVE THRESHOLDING.          MID
#
# IDENTIFICATION
#
#   TITLE          DAT
#   AUTHOR         KRISHNA ARVIND
#   VERSION        A.01
#   DATE           20-DEC-1983
#   LANGUAGE       RATFOR
#   SYSTEM         VAX 11/780
#
# PURPOSE
#
#   DOES THE DOUBLE ADAPTIVE THRESHOLDING.
#
# ENTRY POINT
#
#   CALL ( FD11, FDO1, FTR, IY, IR, NPPL, NLIN, MEAN, NUMB, IMAGS,
#         RWSIZ, CWSIZ, DBLEVL, SIZ2, P, PSIZ, THTXT, IEV, ALTRET )
#
# ARGUMENT LISTING
#
#   FD11   CHR.ARR   INPUT FILE DESCRIPTOR.
#   FDO1   CHR.ARR   OUTPUT FILE DESCRIPTOR.
#   FTR    REAL      FACTOR USED FOR COMPUTING THE CUTOFF.
#   IY     INT.ARR   ARRAY TO HOLD ONE LINE OF ONE BAND OF
#                   INPUT.
#   IR     INT.ARR   ARRAY TO HOLD ONE LINE OF A BAND OF
#                   OUTPUT.
#   NPPL   INTEGER   NUMBER OF PIXELS PER LINE.
#   NLIN   INTEGER   NUMBER OF LINES IN THE IMAGE. ALSO SIZE
#                   OF MEAN.
#   MEAN   INT.ARR   BUFFER TO HOLD THE INPUT IMAGE IN ORDER
#                   TO COMPUTE THE WINDOW MEAN.
#   NUMB   INTEGER   NUMBER OF BANDS SPECIFIED.
#   IMAGS  INT.ARR   CONTAINS THE SPECIFIED BAND NUMBERS.
#   RWSIZ  INTEGER   ODD ROW SIZE OF THE RUNNING WINDOW.
#   CWSIZ  INTEGER   ODD COLUMN SIZE OF THE RUNNING WINDOW.
#   DBLEVL INTEGER   DOUBLE THRESHOLD LEVEL.
#   SIZ2   INTEGER   COLUMN SIZE OF THE ARRAY MEAN.
#   P      INTEGER   ARRAY FOR THE POINTERS OF THE LINE
#                   BUFFER.
#   PSIZ   INTEGER   SIZE OF THE ARRAY P.
#   THTXT  INTEGER   THRESHOLD FOR TEXTURE.
#   IEV    INTEGER   INTEGER EVENT VARIABLE.
#   ALTRET INTEGER   ALTERNATE RETURN.
#
# INCLUDE FILES/COMMONS

```

```

#
# MACA1 INCLUDE GIPSY GENERAL SYMBOL DEFINITIONS
#
# ROUTINES CALLED
#
# PPUSH PUSH PROGRAM NAME ONTO ERROR STACK (GIPSY PRIMITIVE)
#
# CPYIDR COPY A RANDOM FILE INTO A TEMPORARY SEQUENTIAL FILE. (GIPSY PRIMITIVE)
#
# DSCNAM WRITE DESCRIPTOR RECORD -- NAME RECORD. (GIPSY PRIMITIVE)
#
# PDSCR PUT DESCRIPTOR RECORD - REAL NUMBER. (GIPSY PRIMITIVE)
#
# PDSCI PUT DESCRIPTOR RECORD - INTEGER. (GIPSY PRIMITIVE)
#
# COPYDS COPY DESCRIPTOR RECORDS FROM TEMP FILE. (GIPSY PRIMITIVE)
#
# RREAD READ A BLOCK OF A STANDARD IMAGE FILE (SIF). (GIPSY PRIMITIVE)
#
# RWRITE WRITE A BLOCK OF A STANDARD IMAGE FILE (SIF). (GIPSY PRIMITIVE)
#
# PPOP POP PROGRAM NAME FROM THE ERROR STACK (GIPSY PRIMITIVE)
#
# CLOSE CLOSE A FILE. (GIPSY PRIMITIVE)
#
# *****
#
# INCLUDE MACA1
# SUBROUTINE DAT (FDI1, FDO1, FTR, IY, IR, NPPL, NLIN, MEAN, NUMB, IMAGS,
# RWSIZ, CWSIZ, DBLEVL, SIZ2, P, PSIZ, THTXT, IEV, *)
#
#
# TYPE STATEMENTS
#
# IMPLICIT INTEGER (A-Z)
# CHARACTER FDI1(.FDLENGTH), FDO1(.FDLENGTH)
# REAL FTR,RNUMPL,RNUMLN,RTOT,AM
#
# INTEGER IY(NPPL),IR(NPPL),MEAN(RWSIZ,SIZ2),IMAGS(NUMB),P(PSIZ)
# INTEGER IDENT(.IDLENGTH), JDENT(.IDLENGTH)
# EQUIVALENCE (IDENT(.IDNPPL), NUMPPL), (IDENT(.IDNLINS), NUMLIN)
#
#
#
#

```



```

LBA = RWSIZ/2
CBA = CWSIZ/2
#
#                               SET UP THE POINTERS AND WORK ARRAYS.
#
DO I = 1, RWSIZ
  $(
    P(I) = I
    DO K = 1, NUMPPL
      MEAN(I,K) = 0
    $)
#
DO I = 1, LBA
  $(
    CALL RREAD (FDI1, IY, IMNO, I, IDENT, .WAIT, IEV, %9999)
    DO J = 1, NUMPPL
      MEAN(I + LBA, J) = IY(J)
    $)
#
#                               START PROCESSING THE IMAGE
#
DO I = 1, NLIN
  $(
    LINENO = I + LBA
    IF (LINENO > NLIN )
      DO J = 1, NUMPPL
        MEAN(P(RWSIZ), J) = 0
      ELSE
        $(
          CALL RREAD (FDI1, IY, IMNO, LINENO, IDENT, .WAIT, IEV, %9999)
          DO J = 1, NUMPPL
            MEAN(P(RWSIZ), J) = IY(J)
          $)
#
TOTSUM = 0
TOTPPL = 0
#
DO J = 1, CBA + 1
  DO K = 1, RWSIZ
    $(
      IF (MEAN(K, J) >= DBLEVL)
        $(
          TOTSUM = TOTSUM + MEAN(K, J)
          TOTPPL = TOTPPL + 1
        $)
    $)
  TBORD = I - LBA
  IF (TBORD < 1)
    TBORD = 1
  BBORD = I + LBA

```

```

IF (BBORD > NLIN)
  BBORD = NLIN
RPPL = 0
LPPL = 0
RSUM = 0
LSUM = 0

```

```

#
#
#

```

START PROCESSING THE LINE

```

DO J = 1, NUMPPL
  $(
  TOTSUM = TOTSUM + RSUM - LSUM
  TOTPPL = TOTPPL + RPPL - LPPL
  RSUM = 0
  LSUM = 0
  RPPL = 0
  LPPL = 0

  LBORD = J - CBA
  IF(LBORD < 1)
    LBORD = 1
  ELSE
    $(
    DO K = 1, RWSIZ
      $(
      IF (MEAN(P(K),LBORD) >= DBLEVL)
        $(
        LSUM = LSUM + MEAN(P(K),LBORD)
        LPPL = LPPL + 1
        $)
      $)
    $)
  RBORD = J + CBA + 1
  IF(RBORD > NUMPPL)
    $(
    RBORD = NUMPPL
    RSUM = 0
    $)
  ELSE
    $(
    DO K = 1, RWSIZ
      $(
      IF (MEAN(P(K),RBORD) >= DBLEVL)
        $(
        RSUM = RSUM + MEAN(P(K),RBORD)
        RPPL = RPPL + 1
        $)
      $)
    $)
  $)

```

```

#

```

```

RTOT = TOTSUM
RNUMPL = TOTPPL
IF (RNUMPL /= 0.0)
  RTOT = RTOT/RNUMPL
ELSE
  RTOT = 10000.0
CUTOFF = RTOT * FTR + 0.5
IF (MEAN(P(LBA + 1),J) > CUTOFF)
  IR(J) = MEAN(P(LBA + 1),J)
ELSE
  IF (MEAN(P(LBA + 1),J) > THTXT)
    IR(J) = -MEAN(P(LBA+1),J)
  ELSE
    IR(J) = 0
S)
#
#           WRITE THE LINE OUT
CALL RWRITE (FD01, IR, NB, I, JDENT, .WAIT, IEV, %9999)
#
#           ROTATE THE POINTERS
#
TEMP = P(1)
DO K = 1, RWSIZ - 1
  P(K) = P(K + 1)
P(RWSIZ) = TEMP
S)
#
#           GO BACK FOR ANOTHER BAND
#
# $)
#
#           WE ARE DONE.  CLOSE FILES AND RETURN
#
CALL CLOSE (FDI1)
CALL CLOSE (FD01)
#
CALL PPOP
#
RETURN
#
#           ERROR CONDITIONS
#
9999  CONTINUE
#
#           CLOSE INPUT FILE AND TAKE ERROR RETURN
#
CALL CLOSE (FDI1)
CALL CLOSE (FD01)
#
RETURN 1
#

```

END

DESCRIPTION OF GROW.

*GROW Grow the regions in a numeric image.

VERSION: A.01 DATE: 05-1-83 CONTACT: K. ARVIND

ACTION: The DAT operator produces a numeric image with region pixels marked negative. The GROW operator is used to grow these regions in a user specified number of iterations. This growing is mainly to fill up holes in the regions. The grown regions can be pruned later with the DELT operator. The algorithm involves computing the number of negative neighbours in a 3 by 3 neighbourhood of the current pixel. If this value is greater than or equal to a threshold specified by the user, then the current pixels value is changed to negative if it is positive and nonzero.

SOURCE: Disk, SIF.
DESTINATION: Disk, SIF.

FLAGS: None.

- QUESTIONS: (1) User is asked which bands to process.
 Note: this is not asked if the image file contains only one band.
- (2) User is asked for the number of iterations for growing.
- (3) User is asked the threshold for growing. This is the minimum number of neighbours in each iteration required to change a pixels value from positive to negative.

COMMAND STRING EXAMPLE:

GROW OUT.SIF < IN.SIF

ENTER NUMBER OF ITERATIONS (D=2, 1-15) - 1
ENTER THRESHOLD FOR GROWING (D=3, 0-8) - 3

Band '1' of IN.SIF will be grown. A positive nonzero pixel in the input will be written as negative if it has more than or equal 3 negative neighbours. All other pixels have the same value and sign in the output. The output file OUT.SIF will consist of one band.

COMMENTS: For a detailed explanation refer to 'EXTRACTION OF LINES AND REGIONS FROM GREY TONE IMAGES OF LINE DRAWINGS' : Masters Thesis submitted by Krishna Arvind to the Faculty of Computer Science in January 1984.

RUN FILE FOR GROW.

\$\$\$! TESTING GROW

\$\$\$!

\$\$\$! CREATE A SIMPLE NUMERIC IMAGE

\$\$\$!

\$\$\$

EXSIF

OPEN TESTIN.SIF

Y

0 0 0 0

9 5 5 1

1 6 511 5

5 1 -255 255

1 0 0 0

0 0 0 0 0

0 -200 -200 -200 0

0 200 200 200 0

0 200 200 200 0

0 0 0 0 0

DONE

\$\$\$!

\$\$\$!

\$\$\$!

\$\$\$

DO THE GROW OPERATION NOW

GROW TESTIN.SIF > TESTOUT.SIF

1

3

\$\$\$!

\$\$\$!

\$\$\$!

\$\$\$

CREATE AN INPUT IMAGE TO COMPARE THE OUTPUT.

EXSIF

OPEN COMPAR.SIF

Y

0 0 0 0

9 5 5 1

1 6 511 5

5 1 -255 255

1 0 0 0

0 0 0 0 0

0 -200 -200 -200 0

0 200 -200 200 0

0 200 200 200 0

0 0 0 0 0

DONE

\$\$\$!

\$\$\$!

\$\$\$!

\$\$\$!

\$\$\$!

\$\$\$

COMPARE THE OUTPUT WITH THE TEST INPUT. THE DIFFERENCE SHOULD BE ZERO.

RMS TT < COMPAR.SIF, TESTOUT.SIF

DELETE TEMPORARY FILES

DELETE TESTIN.SIF

\$
\$
\$

DELETE COMPAR.SIF
DELETE TESTOUT.SIF
EXIT

LINK FILE FOR GROW.
\$ GENCMD GROW DGROW, GROW


```

#--GROW          DOES THE GROW OPERATION.          MID
#
# IDENTIFICATION
#
#   TITLE          GROW
#   AUTHOR        KRISHNA ARVIND
#   VERSION       A.01
#   DATE          20-DEC-1983
#   LANGUAGE      RATFOR
#   SYSTEM        VAX 11/780
#
# PURPOSE
#
#   DOES A GROWING OPERATION OF THE PIXELS BASED ON THE CONNECTIVITY
#   OF OTHER REGION PIXELS.
#
# ENTRY POINT
#
#   CALL GROW ( FDI, FDO, NUMB, IMAGS, NBUF, IILINE, IOLINE, NPPL,
#   IEV, ALTRET )
#
# ARGUMENT LISTING
#
#   FDI    CHR.ARR    INPUT FILE DESCRIPTOR.
#   FDO    CHR.ARR    OUTPUT FILE DESCRIPTOR.
#   NUMB   INTEGER    THE NUMBER OF SPECIFIED BANDS.
#   IMAGS  INT.ARR    CONTAINS THE NUMBERS OF THE SPECIFIED
#                     BANDS.
#   NBUF   INTEGER    SIZE OF THE LINE BUFFER IILINE.
#   IILINE ARR.INT    INPUT LINE BUFFER.
#   IOLINE ARR.INT    INPUT OUTPUT LINE BUFFER.
#   NPPL   INTEGER    NUMBER OF PIXELS PER LINE.
#   IEV    INTEGER    INTEGER EVENT VARIABLE.
#   ALTRET INTEGER    ALTERNATE RETURN
#
# INCLUDE FILES/COMMONS
#
#   MACA1    INCLUDE  GIPSY GENERAL SYMBOL DEFINITIONS
#   TTCOM    COMMON   FD'S FOR TERMINAL AND RUNFILE IO
#
# ROUTINES CALLED
#
#   PPUSH    PUSH PROGRAM NAME ONTO ERROR STACK          (GIPSY PRIMITIVE)
#   RDKINL   INITIALIZE AND ACCESS A STANDARD IMAGE \
#             FILE.                                       (GIPSY PRIMITIVE)
#   CLOSE    CLOSE A FILE.                                (GIPSY PRIMITIVE)
#   CPYIDR   COPY A RANDOM FILE INTO A TEMPORARY

```



```

JDENT(.IDNPPL) = NPPL
JDENT(.IDNLINS) = NLINS
JDENT(.IDNCOLS) = NCOLS
JDENT(.IDNROWS) = NROWS
JDENT(.IDMODE) = MODE
JDENT(.IDMIN) = IDENT(.IDMIN)
JDENT(.IDMAX) = IDENT(.IDMAX)
JDENT(.IDNBDS) = NUMB
JDENT(.IDNSBDS) = 0

```

```

#
#
#
#

```

```

OPEN DESCRIPTOR RECORDS FROM INPUT IMAGE,
ROUTINE NAME, AND PARAMETERS TO TEMPORARY
SEQUENTIAL FILE.

```

```

CALL CPYIDR(FDI, IDENT, .OPNTMP, IEV, %9998)
CALL DSCNAM('GROW', IEV, %9998)
CALL COPYDS(FDO, JDENT, IEV, %9998)

```

```

#

```

```

CALL RNGETI('ENTER NUMBER OF ITERATIONS.', 1, 15, 2, ITER, IEV, %9010)
CALL RNGETI('ENTER THRESHOLD FOR GROWING.', 1, 0, 8, 3, THRS, IEV, %9010)

```

```

#
#
#
#
#
#

```

```

ACTUAL NUMBER CRUNCHIN'

```

```

PROCESS EACH BAND.

```

```

DO NB = 1, NUMB
  $(
    IMNO = IMAGS(NB)
    DO PASS = 1, ITER
      $(

```

```

#
#
#

```

```

SET UP THE BUFFER POINTERS.

```

```

INDEX(1) = 1
INDEX(2) = 2
INDEX(3) = 3

```

```

#
#
#

```

```

READ IN THE LINES.

```

```

IF (PASS == 1)
  CALL RREAD(FDI, IOLINE, IMNO, 1, IDENT, .WAIT, IEV, %9999)
ELSE
  CALL RREAD(FDO, IOLINE, NB, 1, JDENT, .WAIT, IEV, %9999)
DO M = 1, NPPL
  $(
    IILINE(1, M) = 0
    IILINE(2, M) = IOLINE(M)

```

#

```

S)
DO I = 2, NLINS + 1
S(
IF (I > NLINS)
DO M = 1, NPPL
IILINE(INDEX(3),M) = 0
ELSE
S(
IF (PASS == 1)
CALL RREAD(FDI, IOLINE, IMNO, I, IDENT, .WAIT, IEV, %9999)
ELSE
CALL RREAD(FDO, IOLINE, NB, I, JDENT, .WAIT, IEV, %9999)
DO M = 1, NPPL
IILINE(INDEX(3),M) = IOLINE(M)
S)

```


#

PROCESS THE CURRENT LINE.

```

DO K = 1, NPPL
S(
IF (K > 1)
S(
TEMP(1) = IILINE(INDEX(1), K-1)
TEMP(7) = IILINE(INDEX(3), K-1)
TEMP(8) = IILINE(INDEX(2), K-1)
S)
ELSE
S(
TEMP(1) = 0
TEMP(7) = 0
TEMP(8) = 0
S)
IF (K < NPPL)
S(
TEMP(3) = IILINE(INDEX(1), K+1)
TEMP(4) = IILINE(INDEX(2), K+1)
TEMP(5) = IILINE(INDEX(3), K+1)
S)
ELSE
S(
TEMP(3) = 0
TEMP(4) = 0
TEMP(5) = 0
S)
TEMP(2) = IILINE(INDEX(1), K)
TEMP(6) = IILINE(INDEX(3), K)
IOLINE(K) = IILINE(INDEX(2), K)
NUM = 0

```


DESCRIPTION OF DELT.

*DELT Prune and remove the regions in a numeric image.

VERSION: A.01 DATE: 05-1-83 CONTACT: K. ARVIND

ACTION: The GROW operator grow the regions in the image which are detected and marked as negative valued pixels by the DAT operator. The growing is used to fill up holes in the regions. The DELT operator is used to reduce ie. to prune back the regions to the original size and also to remove isolated noise like regions. These pruned regions are then removed from the image. The algorithm involves computing the number of negative neighbours in a 3 by 3 neighbourhood of the current pixel. If this value is less than a user given threshold and the current pixel is negative then the current pixels value is changed to positive else it is made zero. Otherwise the value remains unchanged.

SOURCE: Disk, SIF
DESTINATION: Disk, SIF

FLAGS: None.

QUESTIONS: (1) User is asked which bands to process.
Note: this is not asked if the image file contains only one band.
(2) User is asked for the threshold for pruning. If the number of negative neighbours of a pixel is less than this number then if it is negative it is changed to positive otherwise it is made zero.

COMMAND STRING EXAMPLE:

DELT IN.SIF > OUT.SIF

ENTER THRESHOLD FOR DELETING (D = 4, 1-8) - 4

Band '1' of IN.SIF will be pruned. A negative pixel in the input will be written as positive in the output if it has less than 4 negative neighbours, otherwise it is written as zero. All other pixels have the same value and sign in the output. The output file OUT.SIF will consist of one band.

COMMENTS: The DELT operation usually follows the DAT and GROW operations and can be used to reduce the regions back to the original size and remove isolated noise like regions. For a detailed explanation refer to

'EXTRACTION OF LINES AND REGIONS FROM GREY TONE IMAGES
OF LINE DRAWINGS' : Masters Thesis submitted by Krishna
Arvind to the Faculty of Computer Science in
January 1984.

```

RUN FILE FOR DELT.
$ ! TESTING DELT
$ !
$ ! CREATE A SIMPLE NUMERIC IMAGE
$ !
$ EXSIF
$ OPEN TESTIN.SIF
$ Y
$ 0 0 0 0
$ 9 5 5 1
$ 1 6 511 5
$ 5 1 -255 255
$ 1 0 0 0
0 0 0 0
0 -200 -200 -200 0
0 -200 200 -200 0
0 200 200 200 0
0 0 0 0 0
DONE
$ !
$ ! DO THE DELT OPERATION NOW
$ !
$ ! DELT TESTIN.SIF > TESTOUT.SIF
$ 4
$ !
$ ! CREATE A TEST INPUT TO COMPARE THE OUTPUT WITH.
$ !
$ EXSIF
$ OPEN COMPAR.SIF
$ Y
$ 0 0 0 0
$ 8 5 5 1
$ 1 6 256 5
$ 5 1 0 255
$ 0 0 0 0
0 0 0 0
0 200 0 200 0
0 200 200 200 0
0 200 200 200 0
0 0 0 0 0
DONE
$ !
$ ! COMPARE THE TEST INPUT WITH THE OUTPUT. THE DIFFERENCE SHOULD BE ZERO.
$ !
$ ! RMS TT < TESTOUT.SIF, COMPAR.SIF
$ !
$ ! DELETE TEMPORARY FILES
$ !
$ ! DELETE TESTIN.SIF

```

\$
\$
\$

DELETE COMPAR.SIF
DELETE TESTOUT.SIF
EXIT

LINK FILE FOR DELT.
\$ GENCMD DELT DDEL, DELT

```

#--DDELT          DRIVER FOR THE DELT OPERATOR          MID
#
# IDENTIFICATION
#
#   TITLE          DDELT
#   AUTHOR         KRISHNA ARVIND
#   VERSION        A.01
#   DATE           20-DEC-1983
#   LANGUAGE       RATFOR
#   SYSTEM         VAX 11/780
#
#
# PURPOSE
#
#   DRIVER FOR THE DELT OPERATOR. ALLOCATES STORAGE AND CHECKS THE
#   INPUT IMAGE FILE.
#
# ENTRY POINT
#
#   CALL DDELT ( WORK, ALTRET )
#
# ARGUMENT LISTING
#
#   WORK   ARRAY          COMMON STORAGE AREA.
#   ALTRET INTGER        ALTERNATE RETURN.
#
# INCLUDE FILES/COMMONS
#
#   MACA1   INCLUDE      GIPSY GENERAL SYMBOL DEFINITIONS
#   ERROR   COMMON       ERROR TRACE STACK COMMON DATA
#   GIPCOM  COMMON       GIPSY COMMAND LINE COMMON DATA
#   TTCOM   COMMON       FD'S FOR TERMINAL AND RUNFILE IO
#
# ROUTINES CALLED
#
#   PPUSH      PUSH PROGRAM NAME ONTO ERROR STACK
#              (GIPSY PRIMITIVE)
#   RDKINL     INITIALIZE AND ACCESS A STANDARD IMAGE
#              FILE.
#              (GIPSY PRIMITIVE)
#   CLOSE      CLOSE A FILE.
#              (GIPSY PRIMITIVE)
#   COMTIN     HAS USER INPUT COMMENT DESCRIPTOR RECORDS
#              (GIPSY PRIMITIVE)
#   NMBNDS     NUMBER OF BANDS TO BE SELECTED
#              (GIPSY PRIMITIVE)
#   GETWP      GET WORK ARRAY POINTER.
#              (GIPSY PRIMITIVE)
#   BANDSP     BANDS SPECIFICATION
#              (GIPSY PRIMITIVE)
#   OSALOC     CHECK AND EXTEND USER WORK SPACE
#              (GIPSY OS KERNEL)
#   DELT       DOES THE ACTUAL DELT OPERATION.
#              (USER ROUTINE)
#   PPOP       POP PROGRAM NAME FROM THE ERROR STACK
#              (GIPSY PRIMITIVE)

```

```

#
# *****
#
# SUBROUTINE DDEL(TWORK,*)
#   IMPLICIT INTEGER (A-Z)
#   INCLUDE MACA1
#   INCLUDE GIPCOM
#   INCLUDE ERROR
#   INCLUDE TTCOM
#
#   INTEGER WORK(.ARB), NUMB
#   INTEGER IDENT(.IDLENGTH)
#   LOGICAL FLG, AFLAG
#
#   EQUIVALENCE (NBITS, IDENT(.IDNBITS))
#   EQUIVALENCE (NPPL, IDENT(.IDNPPL))
#   EQUIVALENCE (NLINS, IDENT(.IDNLINS))
#   EQUIVALENCE (NCOLS, IDENT(.IDNCOLS))
#   EQUIVALENCE (NROWS, IDENT(.IDNROWS))
#   EQUIVALENCE (NTBND, IDENT(.IDNBND))
#   EQUIVALENCE (NSBND, IDENT(.IDNSBND))
#   EQUIVALENCE (MODE, IDENT(.IDMODE))
#
#   CALL PPUSH("DDEL")
#
#
#           SET UP INPUT FILES
#
#   CALL RDKINL(FD11, IDENT, .OLD, IEV, %9999)
#   CALL CLOSE(FD11)
#
#
#           CHECK THE INPUT FILE
#
#   IF (MODE ^= 0 & MODE ^= 1 & MODE ^= 2)
#     GO TO 9000
#   IF (NPPL ^= NCOLS | NROWS ^= 1)
#     GO TO 9010
#   IF (NTBND - NSBND < 0)
#     GO TO 9020
#
#
#           GET THE USER COMMENTS
#
#   CALL COMTIN(%9999)

```


#

FIND OUT HOW MANY AND WHICH NUMERIC
BANDS TO PROCESS.

FLG = .FALSE.
CALL NMBNDS (FD11, IDENT, -1, NUMB, AFLAG, FLG, IEV, %9999)

#

NXT = 1
IMAGS = GETWP(NXT, .INTMODE, NUMB)
CALL BANDSP (WORK(IMAGS), NUMB, NUMB, IDENT, -1, AFLAG, IEV, %9999)
NBUF = 3 * NPPL
ILINE = GETWP(NXT, MODE, NBUF)
OLINE = GETWP(NXT, MODE, NPPL)
IF (0 /= OSALOC(NXT))
GO TO 9030

#

DO THE ACTUAL NUMBER CRUNCHING AND MUNCHING

CALL DELT(FD11, FD01, NUMB, WORK(IMAGS), NBUF, WORK(ILINE),
WORK(OLINE), NPPL, IEV, %9998)

#

CALL PPOP
CALL CLOSE(FD11)
CALL CLOSE(FD01)
RETURN

#

ABNORMAL CONDITIONS

9000 CONTINUE

#

NOT AN INTEGER OR REAL FILE

IEV = -2012
GO TO 9998

#

9010 CONTINUE

##

NOT IN LINE FORMAT

IEV = -5001
GO TO 9998

#


```

#--DELT                DOES THE DELT OPERATION.                MID
#
# IDENTIFICATION
#
#   TITLE              DELT
#   AUTHOR             KRISHNA ARVIND
#   VERSION            A.01
#   DATE               20-DEC-1983
#   LANGUAGE           RATFOR
#   SYSTEM             VAX 11/780
#
# PURPOSE
#
#   DOES THE ACTUAL DELT OPERATION. DELTETES REGION PIXELS BASED ON
#   CONNECTIVITY TO OTHER NEIGHBOURING REGION PIXELS.
#
# ENTRY POINT
#
#   CALL DELT ( FDI, FDO, NUMB, IMAGS, NBUF, IILINE, IOLINE, NPPL,
#   IEV, ALTRET )
#
# ARGUMENT LISTING
#
#   FDI      CHR.ARR      INPUT FILE DESCRIPTOR.
#   FDO      CHR.ARR      OUTPUT FILE DESCRIPTOR.
#   NUMB     INTEGER      NUMBER OF BANDS SPECIFIED.
#   IMAGS    INT.ARR      CONTAINS THE NUMBERS OF THE BANDS
#                       SPECIFIED.
#   IILINE   INT.ARR      INPUT LINE BUFFER.
#   IOLINE   INT.ARR      INPUT OUTPUT LINE BUFFER.
#   NPPL     INTEGER      NUMBER OF PIXELS PER LINE.
#   IEV      INTEGER      INTEGER EVENT VARIABLE.
#   ALTRET   INTEGR      ALTERNATE RETURN.
#
# INCLUDE FILES/COMMONS
#
#   MACA1     INCLUDE     GIPSY GENERAL SYMBOL DEFINITIONS
#   TTCOM     COMMON      FD'S FOR TERMINAL AND RUNFILE IO
#
# ROUTINES CALLED
#
#   PPUSH     PUSH PROGRAM NAME ONTO ERROR STACK                (GIPSY PRIMITIVE)
#   RDKINL    INITIALIZE AND ACCESS A STANDARD IMAGE            (GIPSY PRIMITIVE)
#               FILE.
#   CLOSE     CLOSE A FILE.                                     (GIPSY PRIMITIVE)
#   CPYIDR    COPY A RANDOM FILE INTO A TEMPORARY                (GIPSY PRIMITIVE)
#               SEQUENTIAL FILE.

```



```
JDENT(.IDNCOLS) = NCOLS
JDENT(.IDNROWS) = NROWS
JDENT(.IDMODE) = 0
JDENT(.IDMIN) = 0
JDENT(.IDMAX) = IDENT(.IDMAX)
JDENT(.IDNBND) = NUMB
JDENT(.IDNSBND) = 0
```

```
#
#
#
#
```

```
OPEN DESCRIPTOR RECORDS FROM INPUT
IMAGE, ROUTINE NAME, AND PARAMETERS
TO TEMPORARY SEQUENTIAL FILE.
```

```
CALL CPYIDR(FDI, IDENT, .OPNTMP, IEV, %9998)
CALL DSCNAM('DELT', IEV, %9998)
CALL COPYDS(FDO, JDENT, IEV, %9998)
```

```
#
```

```
CALL RNGETI('ENTER THRESHOLD FOR DELETING.', 0, 8, 4, THRS, IEV, %9010)
```

```
#
#
#
#
#
#
```

```
ACTUAL NUMBER CRUNCHIN'
```

```
SET UP THE INITIAL 3-LINE BUFFER .
```

```
PROCESS EACH BAND IN THE IMAGE.
```

```
DO NB = 1, NUMB
```

```
$(
  IMNO = IMAGS(NB)
  INDEX(1) = 1
  INDEX(2) = 2
  INDEX(3) = 3
```

```
#
```

```
CALL RREAD(FDI, IOLINE, IMNO, 1, IDENT, .WAIT, IEV, %9999)
```

```
DO M = 1, NPPL
```

```
$(
  IILINE(1, M) = 0
  IILINE(2, M) = IOLINE(M)
$)
```

```
#
```

```
DO I = 2, NLINS+1
```

```
$(
  IF (I > NLINS)
    DO M = 1, NPPL
      IILINE(INDEX(3), M) = 0
```

```
ELSE
```

```
$(
  CALL RREAD(FDI, IOLINE, IMNO, I, IDENT, .WAIT, IEV, %9999)
  DO M = 1, NPPL
    IILINE(INDEX(3), M) = IOLINE(M)
```

```

$)
DO K = 1, NPPL
$(
  IF (K > 1)
    $(
      TEMP(1) = IILINE(INDEX(1), K-1)
      TEMP(7) = IILINE(INDEX(3), K-1)
      TEMP(8) = IILINE(INDEX(2), K-1)
    $)
  ELSE
    $(
      TEMP(1) = 0
      TEMP(7) = 0
      TEMP(8) = 0
    $)
  IF (K < NPPL)
    $(
      TEMP(3) = IILINE(INDEX(1), K+1)
      TEMP(4) = IILINE(INDEX(2), K+1)
      TEMP(5) = IILINE(INDEX(3), K+1)
    $)
  ELSE
    $(
      TEMP(3) = 0
      TEMP(4) = 0
      TEMP(5) = 0
    $)
  TEMP(2) = IILINE(INDEX(1), K)
  TEMP(6) = IILINE(INDEX(3), K)
  NUM = 0
  DO J = 1, 8
    IF (TEMP(J) < 0)
      NUM = NUM+1
  IOLINE(K) = IILINE(INDEX(2), K)
  IF (IILINE(INDEX(2), K) < 0)
    IF (NUM >= THRS)
      IOLINE(K) = 0
    ELSE
      IOLINE(K) = -IILINE(INDEX(2), K)
  $)
CALL RWRITE(FDO, IOLINE, NB, I-1, JDENT, .WAIT, IEV, %9999)
#
TTEMP = INDEX(1)
INDEX(1) = INDEX(2)
INDEX(2) = INDEX(3)
INDEX(3) = TTEMP
$)
$)
CALL CLOSE(FDI)

```


DESCRIPTION OF GREG.

*GREG Get regions from a numeric image.

VERSION: A.01 DATE: 05-1-83 CONTACT: K. ARVIND

ACTION: The GROW operator grow the regions in the image which are detected and marked as negative valued pixels by the DAT operator. The growing is used to fill up holes in the regions. The GREG operator is used to get the regions from the image after the GROW operator has been used. The GREG operator complements the DELT operator which extracts the linework and deletes the regions. The algorithm involves computing the number of negative neighbours in a 3 by 3 neighbourhood of the current pixel. If this value is greater than or equal to a user given threshold and the current pixel is negative then the current pixels value is changed to positive. In all other cases it becomes zero.

SOURCE: Disk, SIF.
DESTINATION: Disk, SIF.

FLAGS: None.

QUESTIONS: (1) User is asked which bands to process.
 Note: this is not asked if the image file contains only one band.
QUESTIONS: (2) User is asked for the threshold . If the number of negative neighbours of a pixel is greater than or equal to this number then if it is negative it is changed to positive otherwise it is made zero.

COMMAND STRING EXAMPLE:

GREG IN.SIF > OUT.SIF

ENTER THRESHOLD (D=4, 0-8).- 4

Regions will be extracted from band '1' of IN.SIF. A negative pixel in the input will be written as positive in the output if it has 4 or more negative neighbours, otherwise it is written as zero. The output file OUT.SIF will consist of one band.

COMMENTS: For a detailed explanation refer to 'EXTRACTION OF LINES AND REGIONS FROM GREY TONE IMAGES OF LINE DRAWINGS' : Masters Thesis submitted by Krishna Arvind to the Faculty of Computer Science in January 1984.

```

RUN FILE FOR GREG.
$ ! TESTING GREG
$ !
$ ! CREATE A SIMPLE NUMERIC IMAGE
$ !
$ EXSIF
$ OPEN TESTIN.SIF
$ Y
$ 0 0 0 0
$ 9 5 5 1
$ 1 6 511 5
$ 5 1 -255 255
$ 1 0 0 0
0 0 0 0
0 -200 -200 -200 0
0 -200 200 -200 0
0 200 200 200 0
0 0 0 0 0
$ DONE
$ !
$ ! DO THE GREG OPERATION NOW
$ !
$ GREG TESTIN.SIF > TESTOUT.SIF
$ 4
$ !
$ ! CREATE A TEST INPUT IMAGE TO COMPARE THE OUTPUT WITH.
$ !
$ EXSIF
$ OPEN COMPAR.SIF
$ Y
$ 0 0 0 0
$ 8 5 5 1
$ 1 6 256 5
$ 5 1 0 255
$ 0 0 0 0
0 0 0 0 0
0 0 200 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
$ DONE
$ !
$ ! COMPARE THE TEST INPUT WITH THE OUTPUT. THE DIFFERENCE SHOULD BE ZERO.
$ !
$ RMS IT < TESTOUT.SIF, COMPAR.SIF
$ !
$ ! DELETE TEMPORARY FILES
$ !
$ DELETE TESTIN.SIF

```

\$
\$
\$

DELETE COMPAR.SIF
DELETE TESTOUT.SIF
EXIT

LINK FILE FOR GREG.
\$GENCMD GREG DGREG,GREG

```

#--DGREG          DRIVER FOR THE OPERATOR GREG.          MID
#
# IDENTIFICATION
#
#   TITLE          DGREG
#   AUTHOR         KRISHNA ARVIND
#   VERSION        A.01
#   DATE           20-DEC-1983
#   LANGUAGE       RATFOR
#   SYSTEM         VAX 11/780
#
# PURPOSE
#
#   THIS ROUTINE IS A DRIVER FOR THE GREG OPERATOR. IT ALLOCATES
#   STORAGE AND CHECKS THE INPUT IMAGE FILE.
#
# ENTRY POINT
#
#   CALL DGREG ( WORK, ALTRET )
#
# ARGUMENT LISTING
#
#   WORK   ARRAY          COMMON STORAGE AREA
#   ALTRET INTEGER        ATERNATE RETURN
#
# INCLUDE FILES/COMMONS
#
#   MACA1   INCLUDE      GIPSY GENERAL SYMBOL DEFINITIONS
#   GIPCOM  COMMON       GIPSY COMMAND LINE COMMON DATA
#   TTCOM  COMMON       FD'S FOR TERMINAL AND RUNFILE IO
#   ERROR  COMMON       ERROR TRACE STACK COMMON DATA
#
# ROUTINES CALLED
#
#   PPUSH      PUSH PROGRAM NAME ONTO ERROR STACK
#              (GIPSY PRIMITIVE)
#   RDKINL    INITIALIZE AND ACCESS A STANDARD IMAGE
#              FILE.
#              (GIPSY PRIMITIVE)
#   CLOSE     CLOSE A FILE.
#              (GIPSY PRIMITIVE)
#   COMTIN    HAS USER INPUT COMMENT DESCRIPTOR RECORDS
#              (GIPSY PRIMITIVE)
#   NMBNDS   NUMBER OF BANDS TO BE SELECTED
#              (GIPSY PRIMITIVE)
#   BANDSP   BANDS SPECIFICATION
#              (GIPSY PRIMITIVE)
#   GETWP    GET WORK ARRAY POINTER.
#              (GIPSY PRIMITIVE)
#   OSALOC   CHECK AND EXTEND USER WORK SPACE
#              (GIPSY OS KERNEL)
#   GREG     DOES THE ACTUAL GREG OPERATION.
#              (USER ROUTINE)
#   PPOP     POP PROGRAM NAME FROM THE ERROR STACK
#              (GIPSY PRIMITIVE)

```


##

FIND OUT HOW MANY AND WHICH NUMERIC
BANDS TO PROCESS.

FLG = .FALSE.
CALL NMBNDS (FD11, IDENT, -1, NUMB, AFLAG, FLG, IEV, %9999)

#

NXT = 1
IMAGS = GETWP(NXT, .INTMODE, NUMB)
CALL BANDSP (WORK(IMAGS), NUMB, NUMB, IDENT, -1, AFLAG, IEV, %9999)
NBUF = 3 * NPPL
ILINE = GETWP(NXT,MODE,NBUF)
OLINE = GETWP(NXT,MODE,NPPL)
IF (O ^= OSALOC(NXT))
GO TO 9030

##

DO THE ACTUAL NUMBER CRUNCHING AND MUNCHING

CALL GREG(FD11,FDO1,NUMB,WORK(IMAGS),NBUF,WORK(ILINE),
WORK(OLINE),NPPL,IEV,%9998)

##

CALL PPOP
CALL CLOSE(FD11)
CALL CLOSE(FD01)
RETURN

##

ABNORMAL CONDITIONS

9000 CONTINUE

##

NOT AN INTEGER OR REAL FILE

IEV = -2012
GO TO 9998

#

9010 CONTINUE

##

NOT IN LINE FORMAT

IEV = -5001
GO TO 9998

#


```

#--GREG                DOES THE ACTUAL GREG OPERATION.                MID
#
# IDENTIFICATION
#
# TITLE                GREG
# AUTHOR               KRISHNA ARVIND
# VERSION              A.01
# DATE                 20-DEC-1983
# LANGUAGE             RATFOR
# SYSTEM               VAX 11/780
#
#
# - PURPOSE
#
# GETS THE REGION PIXELS FROM THE IMAGE. FIRST DOES A DELETE
# OPERATION ON THE REGION PIXELS BASED ON THEIR CONNECTIVITY TO
# OTHER REGION PIXELS.
#
# ENTRY POINT
#
# CALL GREG ( FDI, FDO, NUMB, IMAGS, NBUF, IILINE, IOLINE, NPPL,
# IEV, ALTRET )
#
# ARGUMENT LISTING
#
# FDI    CHR.ARR      INPUT FILE DESCRIPTOR.
# FDO    CHR.ARR      OUTPUT FILE DESCRIPTOR.
# NUMB   INTEGER      NUMBER OF BANDS SPECIFIED.
# IMAGS  INT.ARR      CONTAINS THE NUMBERS OF THE SPECIFIED
#                   BANDS.
# NBUF   INTEGER      SIZE OF THE BUFFER IILINE.
# IILINE INT.ARR      INPUT LINE BUFFER.
# IOLINE INT.ARR      INPUT OUTPUT LINE BUFFER.
# NPPL   INTEGER      NUMBER OF PIXELS PER LINE.
# IEV    INTEGER      INTEGER EVENT VARIABLE.
# ALTRET INTEGER      ALTERNATE RETURN.
#
# INCLUDE FILES/COMMONS
#
# MACA1  INCLUDE     GIPSY GENERAL SYMBOL DEFINITIONS
# TTCOM  COMMON      FD'S FOR TERMINAL AND RUNFILE IO
#
# ROUTINES CALLED
#
# PPUH   PUSH PROGRAM NAME ONTO ERROR STACK (GIPSY PRIMITIVE)
#
# RDKINL INITIALIZE AND ACCESS A STANDARD IMAGE (GIPSY PRIMITIVE)
# FILE.
#
# CLOSE  CLOSE A FILE. (GIPSY PRIMITIVE)

```



```

        JDENT(1) = 0
#
        JDENT(.IDNPPL) = NPPL
        JDENT(.IDNLINS) = NLINS
        JDENT(.IDNCOLS) = NCOLS
        JDENT(.IDNROWS) = NROWS
        JDENT(.IDMODE) = 0
        JDENT(.IDMIN) = 0
        JDENT(.IDMAX) = IDENT(.IDMAX)
        JDENT(.IDNBDS) = NUMB
        JDENT(.IDNSBDS) = 0
#
# OPEN DESCRIPTOR RECORDS FROM INPUT IMAGE ,ROUTINE NAME,
# AND PARAMETERS TO TEMPORARY SEQUENTIAL FILE.
#
        CALL CPYIDR(FDI, IDENT, .OPNTMP, IEV, %9998)
        CALL DSCNAM('GREG', IEV, %9998)
        CALL COPYDS(FDO, JDENT, IEV, %9998)
#
        CALL RNGETI('ENTER THRESHOLD.', 0, 8, 4, THRS, IEV, %9010)
#
# ACTUAL NUMBER CRUNCHIN'
#
# SET UP THE INITIAL 3-LINE BUFFER .
#
# PROCESS EACH BAND IN THE IMAGE.
#
DO NB = 1, NUMB
    $(
        IMNO = IMAGS(NB)
        INDEX(1) = 1
        INDEX(2) = 2
        INDEX(3) = 3
#
        CALL RREAD(FDI, IOLINE, IMNO, 1, IDENT, .WAIT, IEV, %9999)
        DO M = 1, NPPL
            $(
                IILINE(1, M) = 0
                IILINE(2, M) = IOLINE(M)
            $)
#
        DO I = 2, NLINS+1
            $(
                IF (I > NLINS)
                    DO M = 1, NPPL
                        IILINE(INDEX(3), M) = 0
                ELSE
                    $(

```

```

CALL RREAD(FDI, IOLINE, IMNO, I, IDENT, .WAIT, IEV, %9999)
DO M = 1, NPPL
  IILINE(INDEX(3), M) = IOLINE(M)
$)
DO K = 1, NPPL
  $(
  IF (K > 1)
    $(
    TEMP(1) = IILINE(INDEX(1), K-1)
    TEMP(7) = IILINE(INDEX(3), K-1)
    TEMP(8) = IILINE(INDEX(2), K-1)
    $)
  ELSE
    $(
    TEMP(1) = 0
    TEMP(7) = 0
    TEMP(8) = 0
    $)
  IF (K < NPPL)
    $(
    TEMP(3) = IILINE(INDEX(1), K+1)
    TEMP(4) = IILINE(INDEX(2), K+1)
    TEMP(5) = IILINE(INDEX(3), K+1)
    $)
  ELSE
    $(
    TEMP(3) = 0
    TEMP(4) = 0
    TEMP(5) = 0
    $)
  TEMP(2) = IILINE(INDEX(1), K)
  TEMP(6) = IILINE(INDEX(3), K)
  NUM = 0
  DO J = 1, 8
    IF (TEMP(J) < 0)
      NUM = NUM+1
  IOLINE(K) = 0
  IF (IILINE(INDEX(2), K) < 0)
    IF (NUM >= THRS)
      IOLINE(K) = -IILINE(INDEX(2), K)
  $)
CALL RWRITE(FDO, IOLINE, NB, I-1, JDENT, .WAIT, IEV, %9999)
#
TTEMP = INDEX(1)
INDEX(1) = INDEX(2)
INDEX(2) = INDEX(3)
INDEX(3) = TTEMP
$)
$)

```

```
CALL CLOSE(FDI)
CALL CLOSE(FDO)
CALL PPOP
RETURN
#
# ABNORMAL CONDITIONS
#
9010 CONTINUE
#
# INPUT FILES ARE NOT OF THE SAME SIZE
#
#
# IEV = -5021
# GO TO 9999
#
9020 CONTINUE
#
# GO TO 9999
9998 CONTINUE
#
# READ OR WRITE ERROR
#
#
# CALL CLOSE(FDI)
# CALL CLOSE(FDO)
#
9999 RETURN 1
END
```

DESCRIPTION OF PLTRKR.

*PLTRKR Track the peaks of lines in a grey tone line drawing image.

VERSION: A.01

DATE: 12-20-83

AUTHOR: KRISHNA ARVIND

ACTION: This command takes a grey tone image of a line drawing and tracks the peaks of each line (vector). The chain codes of all tracked lines are output. A good strategy before using this command is to first blur the line drawing image by local averaging or using a Gaussian filter. If the lines are very thick this may be followed by thresholding using the DAT operator which also produces a numeric image. By this preprocessing the lines assume a ridge-like shape with a well defined medially located peak line which can be conveniently tracked using the PLTRKR operator.

SOURCE: Disk, input file name.

DESTINATION: Disk, output file name.

FLAGS: NONE

QUESTIONS: (1) Bands to process.

NOTE: This is not asked if the image file contains only one band.

- (2) The threshold factor for marked pixels. The tracker marks the pixels that it tracks (in a temporary buffer). In order to start a vector from a (candidate) pixel the pixels value must be greater than this factor times the average of its marked eight neighbours. In other words the candidate pixel must be strong enough compared to adjacent vectors (i. e. marked pixels). This can be ensured by using a value nearly equal to 1.0 for this factor.
- (3) The threshold factor for all pixels. In order to start a vector from a (candidate) pixel the pixels value must be greater than this factor times the average of its eight neighbours (marked or unmarked). By choosing a value less than or greater than 1.0 it can be ensured that the starting pixel is strong compared to all its eight neighbours.
- (4) The threshold for lines. This should be slightly more than the maximum expected background value (or noise). Thus a pixel must be greater than this value in order to be considered for tracking.

COMMAND STRING EXAMPLES:

BORDER INPUT.SIF > OUTPUT.SEQ

```
ENTER MARKED THRESH (D = 0.9, 0.0 - 100.0) -- 0.85
ENTER ALL THRESH (D = 0.7, 0.0 - 100.0) -- 0.75
ENTER LINE THRESH (D = 50, 0 - 255) -- 60
```

EXAMPLE 1

The input file INPUT.SIF is a one band line drawing image. The chain codes of the lines (vectors) in this image are output to OUTPUT.SEQ which is a sequential file of record length 80.

COMMENTS: The output file will be sequential with a record length of 80. The format of the data is as follows :
LINE1 : LENGTH OF VECTOR = (Length of 1st tracked vector)
LINE2 : STARTING COL = (Starting column of 1st tracked vector)
LINE3 : STARTING ROW = (Starting row of 1st tracked vector)
LINE4 : Chain code of 1st tracked vector.....
LINE5 :
.
.
.
LINE n : .. LENGTH OF VECTOR = (Length of 2nd tracked vector)
.
.

ALGORITHM: (For a detailed explanation refer to 'EXTRACTION OF LINES AND REGIONS FROM GREY TONE IMAGES OF LINE DRAWINGS' : Masters thesis submitted by Krishna Arvind to the Faculty of Computer Science in December 1983.)

STEP 1: Finding the starting pixel for a new line.

The image is scanned from left to right and top to bottom. Thus, line by line each pixel is examined to check whether it continues a line or is a candidate for starting a new line. The following conditions must be satisfied by an unmarked pixel (called the candidate pixel) in order to be a starting point of a line:

- a) Its value is more than a certain threshold to indicate which pixel values are background and which are not. (This is the threshold for lines asked in question no. 3)
- b) Its value is more than a factor (a good value is 0.7) times the average of its non-zero marked or unmarked eight neighbours. When a pixel is tracked it is marked. (This is the threshold factor for all pixels asked in question no. 2)
- c) Its value is more than a factor (a good value is 0.9)

- times the average of its marked neighbours. This condition ensures that the pixel is strong compared to nearby tracked vectors. (This is the threshold factor for marked pixels asked in question no. 1)
- d) Let pixel P1 be the unmarked, untested eight neighbour of the candidate pixel with maximum grey tone value. This selection implies a probable direction for a new line. Directions of the eight neighbours are labeled from 0-7 in an anticlockwise direction with 1 being the northwest direction. Let d be the direction of P1 with respect to the candidate pixel. P1 must also satisfy conditions a) through c).
- e) Let P11, P12, P13 be the neighbours in the directions d-1, d, d+1 (modulo 8) respectively from P1. At least one of these pixels P1j must not be marked, must not have a marked neighbour in the directions s-1 and s+1 from P1 (s is the direction of P1j from P1), and must satisfy conditions a) through c). Otherwise consider P1 tested, attempt to satisfy d) and e) with a different P1, until all possibilities have tested. If conditions a) through e) are satisfied by the candidate pixel, then it is marked along with P1 and the line tracker is invoked to continue tracking from pixel P1.

STEP 2: Tracking a line.

Let P1 represent the previously marked pixel, P2 the current marked pixel, and P3 the next pixel sought by the line tracker. Let d1 be the direction to P1 from the pixel marked prior to P1, d2 the direction from P1 to P2 and d3 the direction from P2 to P3. From among the unmarked neighbours of P2 in the directions d2-2 (mod 8), d2-1 (mod 8), d2, d2+1 (mod 8), d2+2 (mod 8), P3 is chosen as the pixel (if it exists) with maximum grey tone value. P3 is marked if it satisfies the conditions:

- a) P3's grey tone value is greater than a certain threshold which is slightly more than typical background values. (This is the threshold for lines asked in question no. 3)
- b) P3 has no marked eight neighbours in the directions d3-1 (mod 8) and d3+1 (mod 8) from P2.
- c) $\text{MIN} \left[\text{abs}(d3-d1), 8-\text{abs}(d3-d1) \right] \leq 2$.

If P3 does not exist or does not satisfy conditions a) through c) then the current vector is terminated and the next line continuation pixel or line starting pixel is sought. If no such pixel is found, the line tracker moves down to the next scan line.

The tracker is implemented in a parallel fashion. Only one pass is made through the image and all the vectors are tracked from top to bottom.

LIMITATIONS:

- 1) The maximum allowable vector length is 800 pixels.
- 2) Each scan line can contain a maximum of 100 vectors.
- 3) Certain curved or looped lines may be broken up into two or more lines.

(For an explanation of the exact implementation of the tracker refer to the Masters thesis referred to above)

```

RUN FILE FOR PLTRKR.
$ ! TESTING PLTRKR
$ !
$ ! CREATE A SIMPLE NUMERIC IMAGE
$ !
$ EXSIF
  OPEN TESTIN.SIF
  Y
  0 0 0 0
  8 5 5 1
  1 6 256 5
  5 1 0 255
  1 0 0 0
250 100 0 0 0
100 250 100 0 0
0 100 250 100 0
0 0 100 250 100
0 0 0 100 250
  DONE
$ !
$ ! DO THE PLTRKR OPERATION NOW
$ !
$ PLTRKR TESTIN.SIF > TT
  0.9
  0.7
  50
$ !
$ ! THE OUTPUT SHOULD LOOK LIKE
$ !
$ ! LENGTH OF VECTOR = 5
$ ! STARTING COL = 1
$ ! STARTING ROW = 1
$ ! CHAIN CODE OF VECTOR =
$ ! 5555
$ !
$ ! DELETE TEMPORARY FILES
$ !
$ DELETE TESTIN.SIF
$ EXIT

```

LINK FILE FOR PLTRKR.
\$GENCMD PLTRKR DTRKR.RAT, PLTRKR, ADVVCT, SRTTRK, TRKVCT, TSTSRT, FNDSLTL, PASTHS, MOD8

```

#--DTRKR          DRIVER FOR THE PARALLEL TRACKER PLTRKR          MID
#
# IDENTIFICATION
#
#   TITLE          DTRKR
#   AUTHOR         KRISHNA ARVIND
#   VERSION        A.01
#   DATE           19-DEC-1983
#   LANGUAGE       RATFOR
#   SYSTEM         VAX 11/780
#
#
# PURPOSE
#
#   THIS ROUTINE IS A DRIVER FOR THE PARALLEL TRACKER PLTRKR.
#
# ENTRY POINT
#
#   CALL DTRKR( WORK, * )
#
# ARGUMENT LISTING
#
#   WORK   INT.ARR   WORK ARRAY
#   ALTRET INTEGER   ALTERNATE RETURN
#
# INCLUDE FILES/COMMONS
#
#   MACA1   INCLUDE   GIPSY GENERAL SYMBOL DEFINITIONS
#   GIPCOM  COMMON    GIPSY COMMAND LINE COMMON DATA
#   ERROR   COMMON    ERROR TRACE STACK COMMON DATA
#   TTCOM   COMMON    FD'S FOR TERMINAL AND RUNFILE IO
#
# ROUTINES CALLED
#
#   PPUSH   PUSH PROGRAM NAME ONTO ERROR STACK          (GIPSY PRIMITIVE)
#
#   RDKINL  INITIALIZE AND ACCESS A STANDARD IMAGE
#           FILE.                                       (GIPSY PRIMITIVE)
#
#   NMBNDS  NUMBER OF BANDS TO BE SELECTED             (GIPSY PRIMITIVE)
#   BANDSP  BANDS SPECIFICATION                       (GIPSY PRIMITIVE)
#   CLOSE   CLOSE A FILE.                             (GIPSY PRIMITIVE)
#   PPOP    POP PROGRAM NAME FROM THE ERROR STACK
#           (GIPSY PRIMITIVE)
#
#   OSPINF  PUT PARAMETER VALUES INTO FILE DESCRIPTOR
#           (GIPSY OS KERNEL)
#
#   GETWP   GET WORK ARRAY POINTER.                   (GIPSY PRIMITIVE)
#   COMTIN  HAS USER INPUT COMMENT DESCRIPTOR RECORDS
#           (GIPSY PRIMITIVE)
#
#   OSALOC  CHECK AND EXTEND USER WORK SPACE          (GIPSY OS KERNEL)

```

```

#          PLTRKR          INITIALIZES AND RUNS THE TRACKER          (USER ROUTINE)
#
#*****
#
#          SUBROUTINE DTRKR(WORK,*)
#          IMPLICIT INTEGER (A-Z)
#          INCLUDE MACAT
#          INCLUDE GIPCOM
#          INCLUDE ERROR
#          INCLUDE TTCOM
#
#          INTEGER WORK(.ARB)
#          INTEGER IDENT(.IDLENGTH)
#          LOGICAL FLG,AFLAG
#
#          EQUIVALENCE (NBITS, IDENT(.IDNBITS))
#          EQUIVALENCE (NPPL, IDENT(.IDNPPL))
#          EQUIVALENCE (NLINS, IDENT(.IDNLINS))
#          EQUIVALENCE (NCOLS, IDENT(.IDNCOLS))
#          EQUIVALENCE (NROWS, IDENT(.IDNROWS))
#          EQUIVALENCE (NTBND, IDENT(.IDNBND))
#          EQUIVALENCE (NSBND, IDENT(.IDNSBND))
#          EQUIVALENCE (MODE, IDENT(.IDMODE))
#
#          CALL PPUSH("DTRKR")
#
#          SET UP INPUT FILES
#
#          CALL RDKINL(FD11, IDENT, .OLD, IEV, %9999)
#          CALL CLOSE(FD11)
#
#          ATTEMPT TO OPEN OUTPUT FILE.
#
#          IEV = OSPINF(FD01, .MODE, .SYSMODE)
#          IF (.OK ^= IEV)
#              GO TO 9999
#
#          IEV = OSPINF(FD01, .LREC, 80)
#          IF (.OK ^= IEV)
#              GO TO 9999
#
#          IEV = OSOPNS(FD01, .OUTPUT)
#          IF (.OK ^= IEV)

```


#

9999 RETURN 1
END

#

#

#

```

#--ADVCT          ADVANCE THE VECTORS                      MID
#
# IDENTIFICATION
#
#   TITLE          ADVCT
#   AUTHOR         KRISHNA ARVIND
#   VERSION        A.01
#   DATE           20-DEC-1983
#   LANGUAGE       RATFOR
#   SYSTEM         VAX 11/780
#
# PURPOSE
#
#   THIS ROUTINE ADVANCES THE VECTORS AFTER THE BUFFER HAS BEEN
#   UPDATED.
#
# ENTRY POINT
#
#   CALL ADVCT ( VCTARR, VCTARX, VCTARY, IMGBUF, IMGBFX, IMGBFY,
#   INDEX, WRTFLG, FDO, IEV, ALTRET )
#
# ARGUMENT LISTING
#
#   VCTARR  INT.ARR      ARRAY FOR STORING THE CHAIN CODES OF THE
#                   VECTORS.
#   VCTARX  INTEGER      DIMENSION OF VCTARR. CORRESPONDS TO THE
#                   NUMBER OF CHAIN CODES PER TRACKED VECTOR
#   VCTARY  INTEGER      DIMENSION OF VCTARR. CORRESPONDS TO THE
#                   MAXIMUM NUMBER OF VECTORS THAT CAN BE
#                   TRACKED SIMULTANEOUSLY.
#   IMGBUF  INT.ARR      IMAGE BUFFER.
#   IMGBFX  INTEGER      DIMENSION OF IMGBUF. CORRESPONDS TO THE
#                   NUMBER OF PIXELS PER LINE IN THE IMAGE.
#   IMGBFY  INTEGER      DIMENSION OF IMGBUF. CORRESPONDS TO THE
#                   NUMBER OF LINES OF THE IMAGE IN THE
#                   BUFFER.
#   INDEX   INT.ARR      ARRAY OF POINTERS USED FOR BUFFER
#                   ROTATION.
#   WRTFLG  ARR.LOG      ARRAY OF FLAGS FOR INDICATING IF BUFFER
#                   LINES HAVE BEEN CHANGED.
#   FDO     CHR.ARR      OUTPUT FILE DESCRIPTOR.
#   IEV     INTEGER      INTEGER EVENT VARIABLE.
#   ALTRET  INTEGER      ALTERNATE RETURN.
#
# INCLUDE FILES/COMMONS
#
#   STATUS   COMMON      TRACKER STATE IDENTIFIERS.
#   MACA1    INCLUDE     GIPSY GENERAL SYMBOL DEFINITIONS

```

```

#
# ROUTINES CALLED
#
#     PPUSH          PUSH PROGRAM NAME ONTO ERROR STACK
#                                     (GIPSY PRIMITIVE)
#     TRKVCT        ACTUALLY TRACKS THE VECTORS.      (USER ROUTINE)
#     PPOP          POP PROGRAM NAME FROM THE ERROR STACK
#                                     (GIPSY PRIMITIVE)
#
# *****
#
# SUBROUTINE ADVVCT(VCTARR,VCTARX,VCTARY,IMGBUF,IMGBFX,IMGBFY,INDEX,
#                 WRTFLG,FDO,IEV,*)
#
# IMPLICIT INTEGER (A-Z)
# INCLUDE MACA1
# CHARACTER FDO(.FDLENGTH)
# INTEGER VCTARR(VCTARX,VCTARY),IMGBUF(IMGBFX,IMGBFY),INDEX(IMGBFY)
# LOGICAL WRTFLG(IMGBFY)
#
# COMMON /STATUS/ MAXVCT,CURVCT,PRVVCT,NXTVCT,FSTVCT,NEWVCT,SRTLIN,VCTLIN,
#              DLINK,DIRLST,DIRCUR,PTRCUR,XVCT,YVCT,XCUR,YCUR,
#              XOFF(8),YOFF(8),ALLTHS,MRKTHS,ETHS
#
#     CALL PPUSH("ADVVCT")
#     PRVVCT = 0
#     CURVCT = FSTVCT
#     ENDVCT = .FALSE.
#
#                                     FOR EACH VECTOR IN VCTARR, THE LINKED LIST OF
#                                     VECTORS, CALL TRKVCT TO FURTHER TRACK THE VECTOR.
#
#     WHILE (.NOT.(CURVCT == 0 | CURVCT > MAXVCT ))
#         $
#         NXTVCT = VCTARR(CURVCT,DLINK)
#         CALL TRKVCT(VCTARR,VCTARX,VCTARY,IMGBUF,IMGBFX,IMGBFY,INDEX,
#                   WRTFLG,FDO,IEV,%9999)
#         CURVCT = NXTVCT
#         $
#
#     CALL PPOP
#     RETURN
#
# 9999 CONTINUE
#     RETURN 1
#

```

END

```

#--SRTRRK      FINDS VECTOR STARTING POINTS IN CURRENT SCAN LINE.  MID
#
# IDENTIFICATION
#
# TITLE          SRTRRK
# AUTHOR         KRISHNA ARVIND
# VERSION        A.01
# DATE           20-DEC-1983
# LANGUAGE       RATFOR
# SYSTEM         VAX 11/780
#
# PURPOSE
#
# THIS ROUTINE FINDS STARTING POINTS FOR VECTORS IN THE CURRENT
# SCAN LINE AND FOR EACH POINT FOUND CALLS THE TRACKER.
#
# ENTRY POINT
#
# CALL SRTRRK ( VCTARR, VCTARX, VCTARY, IMGBUF, IMGBFX, IMGBFY,
# INDEX, WRNFLG, FDO, IEV, ALTRET )
#
# ARGUMENT LISTING
#
# VCTARR INT.ARR      ARRAY FOR STORING THE CHAIN CODES OF THE
#                      VECTORS.
# VCTARX INTEGER     DIMENSION OF VCTARR. CORRESPONDS TO THE
#                      NUMBER OF CHAIN CODES PER TRACKED VECTOR
# VCTARY INTEGER     DIMENSION OF VCTARR. CORRESPONDS TO THE
#                      MAXIMUM NUMBER OF VECTORS THAT CAN BE
#                      TRACKED SIMULTANEOUSLY.
#
# IMGBUF INT.ARR     IMAGE BUFFER.
# IMGBFX INTEGER     DIMENSION OF IMGBUF. CORRESPONDS TO THE
#                      NUMBER OF PIXELS PER LINE IN THE IMAGE.
# IMGBFY INTEGER     DIMENSION OF IMGBUF. CORRESPONDS TO THE
#                      NUMBER OF NUMBER OF LINES OF THE IMAGE
#                      IN THE BUFFER.
#
# INDEX INT.ARR      ARRAY OF POINTERS USED FOR BUFFER
#                      ROTATION.
# WRNFLG ARR.LOG     ARRAY OF FLAGS USED FOR INDICATING IF
#                      BUFFER LINES HAVE BEEN CHANGED.
#
# FDO CHR.ARR        OUTPUT FILE DESCRIPTOR.
# IEV INTEGER        INTEGER EVENT VARIABLE.
# ALTRET INTEGER     ALTERNATE RETURN.
#
# INCLUDE FILES/COMMONS
#
# STATUS COMMON      TRACKER STATE IDENTIFIERS.
# MACA1 INCLUDE      GIPSY GENERAL SYMBOL DEFINITIONS

```

```

#
# ROUTINES CALLED
#
#       PPU$H       PUSH PROGRAM NAME ONTO ERROR STACK
#                                     (GIPSY PRIMITIVE)
#       TSTSRT     TESTS THE CANDIDATE PIXEL FOR STARTING
#                   A VECTOR.
#                                     (USER ROUTINE)
#       PPOP       POP PROGRAM NAME FROM THE ERROR STACK
#                                     (GIPSY PRIMITIVE)
#
# *****
#
SUBROUTINE SRTRK(VCTARR,VCTARX,VCTARY,IMGBUF,IMGBFX,IMGBFY,INDEX,
                WRTFLG,FDO,IEV,*)
#
# IMPLICIT INTEGER (A-Z)
# INCLUDE MACA1
# CHARACTER FDO(.FDLENGTH)
# INTEGER VCTARR(VCTARX,VCTARY),IMGBUF(IMGBFX,IMGBFY),INDEX(IMGBFY)
# LOGICAL WRTFLG(IMGBFY)
# COMMON /STATUS/ MAXVCT,CURVCT,PRVVCT,NXTVCT,FSTVCT,NEWVCT,SRTLIN,VCTLIN,
#             DLINK,DIRLST,DIRCUR,PTRCUR,XVCT,YVCT,XCUR,YCUR,
#             XOFF(8),YOFF(8),ALLTHS,MRKTHS,ETHS
#
# CALL PPU$H("SRTRK")
#
#             FOR EACH PIXEL IN THE SCAN LINE CALL TSTSRT FOR
#             DETERMINING WHETHER A VECTOR CAN BE STARTED.
#
# DO I = 2, IMGBFX - 1
#   IF (IMGBUF(I,INDEX(4)) > ETHS)
#     CALL TSTSRT(I,VCTARR,VCTARX,VCTARY,IMGBUF,IMGBFX,IMGBFY,INDEX,
#               WRTFLG,FDO,IEV,%9999)
#
# CALL PPOP
# RETURN
#
# 9999 CONTINUE
# RETURN 1
#
# END

```

```

#--TRKVCT          TRACKS THE CURRENT VECTOR.                                MID
#
# IDENTIFICATION
#
#   TITLE          TRKVCT
#   AUTHOR         KRISHNA ARVIND
#   VERSION        A.01
#   DATE           20-DEC-1983
#   LANGUAGE       RATFOR
#   SYSTEM         VAX 11/780
#
# PURPOSE
#
#   THIS ROUTINE TRACKS THE CURRENT VECTOR. THE CURRENT VECTOR
#   IS ESTABLISHED EITHER BY THE ROUTINE TSTSRT OR BY THE
#   ROUTINE ADVVCT.
#
# ENTRY POINT
#
#   CALL TRKVCT ( VCTARR, VCTARX, VCTARY, IMGBUF, IMGBFX, IMGBFY,
#   INDEX, WRTFLG, FDO, IEV, ALTRET )
#
# ARGUMENT LISTING
#
#   VCTARR  INT.ARR      ARRAY FOR STORING THE CHAIN CODES OF THE
#                       VECTORS.
#   VCTARX  INTEGER     DIMENSION OF VCTARR. CORRESPONDS TO THE
#                       NUMBER OF CHAIN CODES PER TRACKED VECTOR
#   VCTARY  INTEGER     DIMENSION OF VCTARR. CORRESPONDS TO THE
#                       MAXIMUM NUMBER OF VECTORS THAT CAN BE
#                       TRACKED SIMULTANEOUSLY.
#   IMGBUF  INT.ARR     IMAGE BUFFER.
#   IMGBFX  INTEGER     DIMENSION OF IMGBUF. CORRESPONDS TO THE
#                       NUMBER OF PIXELS PER LINE IN THE IMAGE.
#   IMGBFY  INTEGER     DIMENSION OF IMGBUF. CORRESPONDS TO THE
#                       NUMBER OF NUMBER OF LINES OF THE IMAGE
#                       IN THE BUFFER.
#   INDEX   INT.ARR     ARRAY OF POINTERS USED FOR BUFFER
#                       ROTATION.
#   WRTFLG  ARR.LOG     ARRAY OF FLAGS USED FOR INDICATING IF
#                       BUFFER LINES HAVE BEEN CHANGED.
#   FDO     CHR.ARR     OUTPUT FILE DESCRIPTOR.
#   IEV     INTEGER     INTEGER EVENT VARIABLE.
#   ALTRET  INTEGER     ALTERNATE RETURN.
#
# INCLUDE FILES/COMMONS
#
#   STATUS   COMMON     TRACKER STATE IDENTIFIERS.

```

```

# MACA1 INCLUDE GIPSY GENERAL SYMBOL DEFINITIONS
#
# ROUTINES CALLED
#
# PPUH PUSH PROGRAM NAME ONTO ERROR STACK (GIPSY PRIMITIVE)
# MOD8 CONVERTS AN INTEGER TO MODULO 8 (USER ROUTINE)
# ARITHMETIC.
# PUTPS WRITE A PACKED CHARACTER STRING TO (GIPSY PRIMITIVE)
# A FILE.
# PUTEOL PUT END OF LINE TO OUTPUT ROUTINES (GIPSY PRIMITIVE)
# (GIPSY PRIMITIVE)
# PUTI WRITE AN INTEGER. (GIPSY PRIMITIVE)
# PPOP POP PROGRAM NAME FROM THE ERROR STACK (GIPSY PRIMITIVE)
#
# *****
#
# SUBROUTINE TRKVCT(VCTARR,VCTARX,VCTARY,IMGBUF,IMGBFX,IMGBFY,INDEX,
# WRTFLG,FDO,IEV,*)
# IMPLICIT INTEGER (A-Z)
# INCLUDE MACA1
# CHARACTER FDO(.FDLENGTH)
# INTEGER VCTARR(VCTARX,VCTARY),IMGBUF(IMGBFX,IMGBFY),DIROFF(5),INDEX(IMGBFY)
# LOGICAL ENDVCT,WRTFLG(IMGBFY),TEST
#
# COMMON /STATUS/ MAXVCT,CURVCT,PRVCT,NXTVCT,FSTVCT,NEWVCT,SRTLIN,VCTLIN,
# DLINK,DIRLST,DIRCUR,PTRCUR,XVCT,YVCT,XCUR,YCUR,
# XOFF(8),YOFF(8),ALLTHS,MRKTHS,ETHS
#
# CALL PPUH("TRKVCT")
#
# DIROFF GIVES OFFSETS WHICH WHEN ADDED TO THE CURRENT
# DIRECTION OF THE TRACKER GIVES A SEQUENCE OF DIRECTIONS
# WHICH ARE TO BE TESTED IN ORDER OF PRIORITY TO FIND
# THE NEXT PIXEL FOR THE TRACKER.
#
# DIROFF(1) = 0
# DIROFF(2) = -1
# DIROFF(3) = +1
# DIROFF(4) = -2
# DIROFF(5) = +2
#
# ESTABLISH THE PARAMETERS FOR THE CURRENT VECTOR.
#
# CURPTR = VCTARR(CURVCT,PTRCUR)

```



```

PIX1 = IMGBUF(XBUF+XOFF(DIR1), INDEX(YBUF+YOFF(DIR1)))
PIX2 = IMGBUF(XBUF+XOFF(DIR2), INDEX(YBUF+YOFF(DIR2)))

#
#
#
CHECK IF THE ADJACENT NEIGHBOURS ARE NOT MARKED.

IF (PIX1 >= 0 & PIX2 >= 0)
  ENDVCT = .FALSE.
ELSE
  $(
#
#
#
    SINCE ADJACENT NEIGHBOURS ARE MARKED THE VECTOR WILL
    BE TERMINATED. MARK THE LAST PIXEL FOR THIS CASE.

    LSTDIR = CURDIR
    CURDIR = NEWDIR
    VCTARR(CURVCT, CURPTR) = NEWDIR
    CURPTR = CURPTR+1
    VCTARR(CURVCT, PTRCUR) = CURPTR
    VCTARR(CURVCT, XCUR) = VCTARR(CURVCT, XCUR) + XOFF(NEWDIR)
    VCTARR(CURVCT, YCUR) = VCTARR(CURVCT, YCUR) + YOFF(NEWDIR)
    VCTLEN = VCTLEN + 1
    VCTARR(CURVCT, DIRLST) = VCTARR(CURVCT, DIRCUR)
    VCTARR(CURVCT, DIRCUR) = NEWDIR
    XBUF = XBUF + XOFF(NEWDIR)
    YBUF = YBUF + YOFF(NEWDIR)
    IMGBUF(XBUF, INDEX(YBUF)) = -IMGBUF(XBUF, INDEX(YBUF))
    WRNFLG(YBUF) = .TRUE.
  $)
  $(
IF (VCTLEN > 2 & .NOT. ENDVCT)
  $(
#
#
#
    CHECK THE DIRECTIONS FOR THE LOOPING CONDITION.

    IF (NEWDIR > LSTDIR)
      $(
        DIFF1 = NEWDIR - LSTDIR
        DIFF2 = LSTDIR - NEWDIR + 8
      $)
    ELSE
      $(
        DIFF1 = LSTDIR - NEWDIR
        DIFF2 = NEWDIR - LSTDIR + 8
      $)
    DIFF = JMINO(DIFF1, DIFF2)
    IF (DIFF > 2)
      ENDVCT = .TRUE.
  $)
IF (ENDVCT)

```


#

```
$(
    THE VECTOR HAS TERMINATED SO WRITE OUT THE
    VECTOR DATA.
IF (FSTVCT == CURVCT)
    FSTVCT = VCTARR(CURVCT,DLINK)
LIM = VCTARR(CURVCT, PTRCUR) - 1
IF (VCTLEN > 2)
    $(
        IF (.OK ^= PUTPS(FDO," "))
            GO TO 9999
        IF (.OK ^= PUTEOL(FDO))
            GO TO 9999
        IF (.OK ^= PUTPS(FDO,"LENGTH OF VECTOR = "))
            GO TO 9999
        IF (.OK ^= PUTI(FDO,VCTARR(CURVCT, PTRCUR)-8,4))
            GO TO 9999
        IF (.OK ^= PUTEOL(FDO))
            GO TO 9999
        IF (.OK ^= PUTPS(FDO,"STARTING COL = "))
            GO TO 9999
        IF (.OK ^= PUTI(FDO,VCTARR(CURVCT,XVCT),4))
            GO TO 9999
        IF (.OK ^= PUTEOL(FDO))
            GO TO 9999
        IF (.OK ^= PUTPS(FDO,"STARTING ROW = "))
            GO TO 9999
        IF (.OK ^= PUTI(FDO,VCTARR(CURVCT,YVCT),4))
            GO TO 9999
        IF (.OK ^= PUTEOL(FDO))
            GO TO 9999
        IF (.OK ^= PUTPS(FDO,"CHAIN CODE OF VECTOR = "))
            GO TO 9999
        IF (.OK ^= PUTEOL(FDO))
            GO TO 9999
        I = 9
        REPEAT
            $(
                COUNT = 1
                WHILE (COUNT <= 70 & I <= LIM)
                    $(
                        IF (.OK ^= PUTI(FDO,VCTARR(CURVCT,I),1))
                            GO TO 9999
                        I = I + 1
                        COUNT = COUNT + 1
                    )
                IF (.OK ^= PUTEOL(FDO))
                    GO TO 9999
            )
    )

```

```

        $)
        UNTIL ( I > LIM)
        $)
#
#
#
        REARRANGE THE LINKED LIST OF VECTORS.
        IF (.NOT.(PRVVCT == 0))
            VCTARR(PRVVCT,DLINK) = VCTARR(CURVCT,DLINK)
        DO I = 1,LIM
            VCTARR(CURVCT,I) = 0
        $)
    ELSE
        $(
#
#
#
#
            CONTINUE TRACKING. WRITE INTO VCTARR THE RECENTLY
            FOUND DIRECTJON. ALSO WRITE THE OTHER PARAMETERS
            FOR TRACKING.

            LSTDIR = CURDIR
            CURDIR = NEWDIR
            VCTARR(CURVCT,CURPTR) = NEWDIR
            CURPTR = CURPTR + 1
            VCTARR(CURVCT,PTRCUR) = CURPTR
            VCTARR(CURVCT,XCUR) = VCTARR(CURVCT,XCUR) + XOFF(NEWDIR)
            VCTARR(CURVCT,YCUR) = VCTARR(CURVCT,YCUR) + YOFF(NEWDIR)
            VCTLEN = VCTLEN + 1
            VCTARR(CURVCT,DIRLST) = VCTARR(CURVCT,DIRCUR)
            VCTARR(CURVCT,DIRCUR) = NEWDIR
            XBUF = XBUF + XOFF(NEWDIR)
            YBUF = YBUF + YOFF(NEWDIR)

#
#
#
            MARK THE NEWLY FOUND PIXEL.

            IMGBUF(XBUF,INDEX(YBUF)) = -IMGBUF(XBUF,INDEX(YBUF))
            WRTFLG(YBUF) = .TRUE.
        $)
    $)
    IF (.NOT. ENDVCT)
        PRVVCT = CURVCT
#
#
        CALL PPOP
        RETURN
#
9999 CONTINUE
    IEV = OSGIEV(IEV)
    RETURN 1
    END

```



```

#
# STATUS COMMON TRACKER STATE IDENTIFIERS.
# MACA1 INCLUDE GIPSY GENERAL SYMBOL DEFINITIONS
#
# ROUTINES CALLED
#
# PPUSH PUSH PROGRAM NAME ONTO ERROR STACK (GIPSY PRIMITIVE)
# TRKVCT TRACKS EACH VECTOR. (USER ROUTINE)
# PASTHS TESTS THE THRESHOLD CONDITIONS. (USER ROUTINE)
# FNDSLT FINDS AN EMPTY SLOT IN VCTARR. (USER ROUTINE)
# MOD8 CONVERTS AN INTEGER TO MODULO 8 ARITHMETIC. (USER ROUTINE)
# PPOP POP PROGRAM NAME FROM THE ERROR STACK (GIPSY PRIMITIVE)
#
# *****
#
# SUBROUTINE TSTSRT(XBUF1,VCTARR,VCTARX,VCTARY,IMGBUF,IMGBFX,IMGBFY,
# INDEX,WRTFLG,FDO,IEV,*)
#
# IMPLICIT INTEGER (A-Z)
# INCLUDE MACA1
# CHARACTER FDO(.FDLENGTH)
# INTEGER VCTARR(VCTARX,VCTARY),IMGBUF(IMGBFX,IMGBFY),INDEX(IMGBFY)
# INTEGER NGB(8)
# LOGICAL FOUND,PASTHS,WRTFLG(IMGBFY)
# COMMON /STATUS/ MAXVCT,CURVCT,PRVVCT,NXTVCT,FSTVCT,NEWVCT,SRTLIN,VCTLIN,
# DLINK,DIRLST,DIRCUR,PTRCUR,XVCT,YVCT,XCUR,YCUR,
# XOFF(8),YOFF(8),ALLTHS,MRKTHS,ETHS
#
# CALL PPUSH("TSTSRT")
#
# ESTABLISH LINE 4 AS THE LINE FOR TESTING
#
# YBUF1 = 4
#
# TEST IF THE CANDIDATE POINT PASSES THE THRESHOLD
# CONDITIONS
#
# IF (PASTHS(XBUF1,YBUF1,IMGBUF,IMGBFX,IMGBFY,INDEX))
# S(
# MAX = ETHS
# LSTDIR = 0
#
# FIND THE MAXIMUM OF THE 8 NEIGHBOURS OF THE

```

```

#                                     CANDIDATE POINT
#
DO I = 1,8
  $(
    NGB(I) = IMGBUF(XBUF1+XOFF(I), INDEX(YBUF1+YOFF(I)))
    IF (NGB(I) > MAX)
      $(
        MAX = NGB(I)
        LSTDIR = I
      $)
  $)
FOUND = .FALSE.

#                                     LOOP TO TEST EACH SUCCESSION MAXIMUM.
#
#
WHILE (MAX > ETHS & .NOT. FOUND)
  $(
    XBUF2 = XBUF1 + XOFF(LSTDIR)
    YBUF2 = YBUF1 + YOFF(LSTDIR)
    NUM = 1

#                                     TEST IF THE MAXIMUM PIXEL PASSES THE THRESHOLD
#                                     CONDITIONS.
#
IF (PASTHS(XBUF2, YBUF2, IMGBUF, IMGBFX, IMGBFY, INDEX))

#                                     LOOP TO TEST EACH OF THE THREE NEIGHBOURS OF THE
#                                     MAXIMUM.
#
WHILE (NUM <= 3 & .NOT. FOUND)
  $(
    CURDIR = MOD8(LSTDIR+NUM-2)
    XBUF3 = XBUF2 + XOFF(CURDIR)
    YBUF3 = YBUF2 + YOFF(CURDIR)
    IF (IMGBUF(XBUF3, INDEX(YBUF3)) > ETHS)
      $(
        XADJ1 = XBUF2+XOFF(MOD8(CURDIR-1))
        YADJ1 = YBUF2+YOFF(MOD8(CURDIR-1))
        XADJ2 = XBUF2+XOFF(MOD8(CURDIR+1))
        YADJ2 = YBUF2+YOFF(MOD8(CURDIR+1))

#                                     TEST THE NEIGHBOUR OF THE MAXIMUM FOR PASSING
#                                     THE THRESHOLD CONDITIONS.
#
IF (IMGBUF(XADJ1, INDEX(YADJ1)) >= 0 & IMGBUF(XADJ2,
        INDEX(YADJ2)) >= 0)
      IF (PASTHS(XBUF3, YBUF3, IMGBUF, IMGBFX, IMGBFY, INDEX))
        $(

```



```

#--FNDSLTT      FINDS AN EMPTY SLOT IN VCTARR.                                MID
#
# IDENTIFICATION
#
# TITLE          FNDSTL
# AUTHOR         KRISHNA ARVIND
# VERSION        A.01
# DATE           20-DEC-1983
# LANGUAGE       RATFOR
# SYSTEM         VAX 11/780
#
# PURPOSE
#
# THIS ROUTINE FINDS AND RETURNS AN EMPTY SLOT IN THE
# ARRAY VCTARR.
#
# ENTRY POINT
#
# CALL FNDSTL ( VCTARR, VCTARX, VCTARY )
#
# ARGUMENT LISTING
#
# VCTARR INT.ARR      ARRAY FOR STORING THE CHAIN CODES OF THE
#                     VECTORS.
# VCTARX INTEGER     DIMENSION OF VCTARR. CORRESPONDS TO THE
#                     NUMBER OF CHAIN CODES PER TRACKED VECTOR
# VCTARY INTEGER     DIMENSION OF VCTARR. CORRESPONDS TO THE
#                     MAXIMUM NUMBER OF VECTORS THAT CAN BE
#                     TRACKED SIMULTANEOUSLY.
#
# INCLUDE FILES/COMMONS.
#
# STATUS COMMON      TRACKER STATE IDENTIFIERS.
#
# ROUTINES CALLED
#
# PPUSH             PUSH PROGRAM NAME ONTO ERROR STACK
#                                     (GIPSY PRIMITIVE)
# PPOP              POP PROGRAM NAME FROM THE ERROR STACK
#                                     (GIPSY PRIMITIVE)
#
# *****
#
# SUBROUTINE FNDSTL(VCTARR,VCTARX,VCTARY)
# IMPLICIT INTEGER (A-Z)
# INTEGER VCTARR(VCTARX,VCTARY)
# LOGICAL FOUND

```

```

COMMON /STATUS/ MAXVCT, CURVCT, PRVVCT, NXTVCT, FSTVCT, NEWVCT, SRTLIN, VCTLIN,
                DLINK, DIRLST, DIRCUR, PTRCUR, XVCT, YVCT, XCUR, YCUR,
                XOFF(8), YOFF(8), ALLTHS, MRKTHS, ETHS
#
CALL PPUSH("FNDSL")
IF (PRVVCT == 0)
    NEWVCT = 1
ELSE
    $(
    FOUND = .FALSE.
    NXT = PRVVCT
#
#                               LOOP THROUGH THE LINKED LIST OF VECTORS IN
#                               VCTARR UNTIL A SLOT IS FOUND.
#
REPEAT
    $(
    IF (VCTARR(NXT, PTRCUR) == 0)
        $(
        NEWVCT = NXT
        FOUND = .TRUE.
        $)
    NXT = NXT + 1
    IF (NXT > MAXVCT)
        NXT = NXT - MAXVCT
    $)
UNTIL (NXT == PRVVCT | FOUND )
$)
#
#
CALL PPOP
RETURN
END

```

```

#--PASTHS          TESTS THE THRESHOLD CONDITIONS FOR THE PIXEL.          MID
#
# IDENTIFICATION
#
#   TITLE           PASTHS
#   AUTHOR          KRISHNA ARVIND
#   VERSION         A.01
#   DATE            20-DEC-1983
#   LANGUAGE        RATFOR
#   SYSTEM          VAX 11/780
#
# PURPOSE
#
#   THIS ROUTINE TESTS THE THRESHOLD CONDITIONS FOR THE
#   GIVEN PIXEL.
#
# ENTRY POINT
#
#   CALL PASTHS ( XBUF, YBUF, IMGBUF, IMGBFX, IMGBFY, INDEX )
#
# ARGUMENT LISTING
#
#   XBUF    INTEGER    COL POSITION OF THE PIXEL TO BE TESTED.
#   YBUF    INTEGER    ROW POSITION OF THE PIXEL TO BE TESTED.
#   IMGBUF  INT.ARR    IMAGE BUFFER.
#   IMGBFX  INTEGER    DIMENSION OF IMGBUF. CORRESPONDS TO THE
#                       NUMBER OF PIXELS PER LINE IN THE IMAGE.
#   IMGBFY  INTEGER    DIMENSION OF IMGBUF. CORRESPONDS TO THE
#                       NUMBER OF NUMBER OF LINES OF THE IMAGE
#                       IN THE BUFFER.
#   INDEX   INT.ARR    ARRAY OF POINTERS USED FOR BUFFER
#                       ROTATION.
#
# INCLUDE FILES/COMMONS
#
#   STATUS    COMMON    TRACKER STATE IDENTIFIERS.
#
# ROUTINES CALLED
#
#   PPUSH     PUSH PROGRAM NAME ONTO ERROR STACK
#                                     (GIPSY PRIMITIVE)
#   PPOP     POP PROGRAM NAME FROM THE ERROR STACK
#                                     (GIPSY PRIMITIVE)
#
# *****
#
#

```

```

LOGICAL FUNCTION PASTHS(XBUF, YBUF, IMGBUF, IMGBFX, IMGBFY, INDEX)
  IMPLICIT INTEGER (A-Z)
  INTEGER IMGBUF(IMGBFX, IMGBFY), INDEX(IMGBFY)
  REAL ALLTHS, MRKTHS
  COMMON /STATUS/ MAXVCT, CURVCT, PRVVCT, NXTVCT, FSTVCT, NEWVCT, SRTLIN, VCTLIN,
    DLINK, DIRLST, DIRCUR, PTRCUR, XVCT, YVCT, XCUR, YCUR,
    XOFF(8), YOFF(8), ALLTHS, MRKTHS, ETHS

#
CALL PPUSH("PASTHS")
NUMMRK = 0
NUMALL = 0
MRKSUM = 0
ALLSUM = 0
DO I = 1, 8
  $(
  XN = XBUF + XOFF(I)
  YN = YBUF + YOFF(I)
#
#           FIND THE SUMS OF THE MARKED NEIGHBOURS AND ALL
#           THE NEIGHBOURS.
#
  IF (IMGBUF(XN, INDEX(YN)) > 0)
    $(
    ALLSUM = ALLSUM + IMGBUF(XN, INDEX(YN))
    NUMALL = NUMALL + 1
    $)
  ELSE
    IF (IMGBUF(XN, INDEX(YN)) < 0)
      $(
      ALLSUM = ALLSUM - IMGBUF(XN, INDEX(YN))
      NUMALL = NUMALL + 1
      MRKSUM = MRKSUM - IMGBUF(XN, INDEX(YN))
      NUMMRK = NUMMRK + 1
      $)
    $)
#
#           COMPUTE THE CUTOFFS.
#
  IF (NUMALL == 0)
    CTFALL = 0
  ELSE
    CTFALL = ALLSUM * ALLTHS / NUMALL
  IF (NUMMRK == 0)
    CTFMRK = 0
  ELSE
    CTFMRK = MRKSUM * MRKTHS / NUMMRK
  MAX = JMAX0(CTFALL, CTFMRK)
  IF (MAX < IMGBUF(XBUF, INDEX(YBUF)))
    PASTHS = .TRUE.

```

```
ELSE  
  PASTHS = .FALSE.  
#  
CALL PPOP  
RETURN  
END
```

```

#--MOD8          RETURNS THE VALUE MODULO 8.          MID
#
# IDENTIFICATION
#
#   TITLE          MOD8
#   AUTHOR         KRISHNA ARVIND
#   VERSION        A.01
#   DATE          20-DEC-1983
#   LANGUAGE       RATFOR
#   SYSTEM        VAX 11/780
#
# PURPOSE
#
#   THIS ROUTINE COMPUTES THE VALUE OF AN INTEGER MODULO 8.
#
# ENTRY POINT
#
#   MOD8 ( DIR )
#
# ARGUMENT LISTING
#
#   DIR    INTEGER    THE DIRECTION WHICH HAS TO BE CONVERTED
#                       TO MODULO 8.
#
# INCLUDE FILES/COMMONS.
#
#   NONE
#
# ROUTINES CALLED
#
#   PPUSH    PUSH PROGRAM NAME ONTO ERROR STACK    (GIPSY PRIMITIVE)
#   PPOP     POP PROGRAM NAME FROM THE ERROR STACK (GIPSY PRIMITIVE)
#
# *****
#
# INTEGER FUNCTION MOD8(DIR)
#   IMPLICIT INTEGER (A-Z)
#
#   CALL PPUSH("MOD8")
#   IF (DIR > 8)
#     MOD8 = DIR - 8
#   ELSE
#     IF (DIR < 1)
#       MOD8 = DIR + 8
#     ELSE

```

```
MOD8 = DIR  
#  
CALL PPOP  
RETURN  
END
```

**The vita has been removed from
the scanned document**