ROAD DETECTION ON RADAR IMAGERY

by

Jungwhan John Kim

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 and Applications

APPROVED:

R. M. Haralick, Chairman

L. G. Shapiro, Co-chairman

R. W. Ehrich

July, 1985

Blacksburg, Virginia

ROAD DETECTION ON RADAR IMAGERY

by

Jungwhan John Kim

R. M. Haralick, Chairman

Computer Science and Applications

(ABSTRACT)

A facet based road network detection procedure is described for radar imagery. The procedure includes a line detection part and a road detection and connection part. The line detection part analytically detects linear features using a facet valley finding technique. Valleys are defined as zero crossings of the first directional derivatives of a bicubic facet model taken in a direction extremizing the second directional derivative. The road detection and connection part statistically screens the linear features on a component by component basis, and then optimally connects the screened linear features using a dynamic programming algorithm.

This thesis also includes as a preprocessing technique for noisy images, an adaptive noise removal algorithm, and suggests a practical method of estimating a local noise variance.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincere gratitude to all members of my commitee.

I especially thank my supervisor, for his support and extensive comments on this work.

I would also like to thank and his family for their encouragement and patience during my stay at Blacksburg.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER 1 INTRODUCTION

The computer recognition of roads on remotely sensed imagery is one of the fundamental tasks in remote sensing. Many attempts have been made on satellite images or aerial photographs, but few on radar images. The goal is to locate roads (possibly a road network) on radar imagery. The radar image differs from other remotely sensed images in that it is formed by an active system, while others are formed by passive systems. As a result, it has different properties and may need somewhat different image processing techniques. Here we discuss a facet based road detection procedure for radar imagery.

In this chapter, an overview of this thesis is given following a short discussion of a radar system and its characteristics.

1.1 RADAR IMAGE SYSTEM

RADAR is the acronym for Radio Detection and Ranging. Radar has been developed for navigation, target detection, and reconnaissance imagery since the reflection of radio waves from solid objects was discovered.

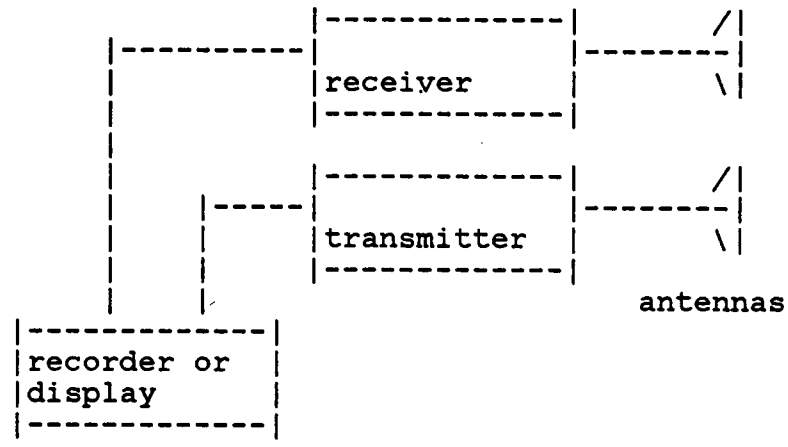


Figure 1. Structure of radar system

Radar imagery is acquired by a complex system. A typical radar system consists of a transmitter, a receiver, antennas, and a recorder (see Figure 1). The transmitter emits a series of short pulses of, generally, a single wavelength through the transmitting antenna. A portion of the energy incident upon a surface is scattered back toward the receiving antenna. The receiver takes the output of the receiving antenna and amplifies its strength. The result is sequentially recorded in the recorder or displayed as a raw image for the user. To obtain a photo-like radar image, the recorded backscatters are further compiled and processed through a complicated imaging system, the study of which is beyond the scope of this thesis. A typical radar image is shown in Figure 2.

Figure 2. A typical radar image

The backscatter from the terrain to the receiving antenna is called radar return. A strong radar return makes a bright image. The intensity of radar returns in a system is determined by the terrain properties such as complex dielectric constant and surface roughness. The effect of dielectric properties on the radar returns are usually not very strong and are therefore neglected in the radar image interpretation. The surface roughness of terrain is generally the dominant factor in determining the return intensity, which is in turn the gray-tone of the image. Surfaces of terrain can be grouped into three categories according to the pattern of the radar returns [1]:

A smooth surface - it specularly reflects all of the incident energy in the opposite direction.

A rough surface - it diffusely scatters the incident energy in all directions.

An intermediate surface - it specularly reflects a portion of the incident energy and diffusely scatters a portion.

The classes of surfaces are closely related to radar wavelength and to the depression angle, which is defined as an angle between the horizontal line and the incident line [1]. At a longer wavelength, the reflection of the incident energy tends to be more specular so that a resolution cell once classified at a shorter wavelength as a rough surface may have to be reclassified as an intermediate or smooth surface. The relationship between depression angle and surface roughness is shown in Figure 3.

Rough surfaces produce almost uniformly strong returns at all depression angles. On the other hand, smooth surfaces produce little or no returns at most of the depression angles except at very high angles near vertical, where the returns are very strong. As a result, smooth surfaces such as still-water surfaces, dry lake beds, paved roads, and airport runways usually show up dark in radar images, whereas rough surfaces such as coarse gravel show up bright. In addition to rough

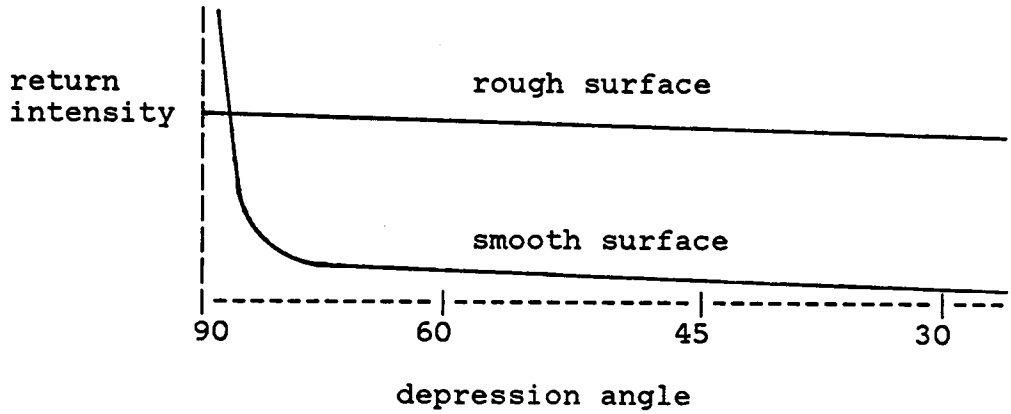


Figure 3. Return intensity: a function of surface roughness and depression angle

surfaces, metallic objects and corner reflectors such as bridges and buildings also produce strong radar returns.

Other important factors influencing the appearance of radar images are shadows and highlights which are produced due to the topography of the terrain and the look direction. For example, the oblique illumination of a Side Looking Airborne Radar (SLAR) produces strong returns, called highlights, from the sides of ridges facing the antenna, and no returns, called shadows, from the other sides (see Figure 4).
A radar look direction can enhance or subdue linear features on a radar image.

In this section, a radar system and its characteristics were described which gave us some insight into how a road appears

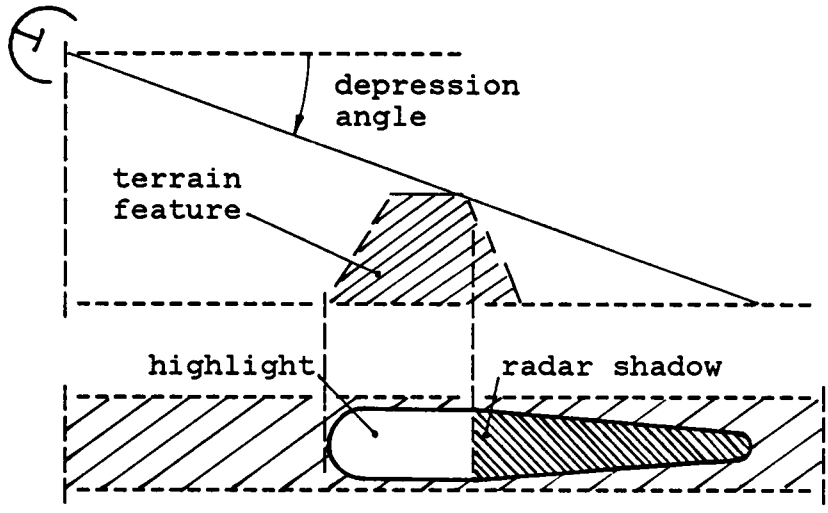


Figure 4. Highlight and shadow

in a radar image. Our road detection project has been started based on this insight and general road characteristics. The next section gives an overview of the road detection procedure.

1.2 OVERVIEW OF THESIS

The ultimate goal of this thesis is to build an effective procedure which can locate road networks on radar imagery. This goal is divided into three subgoals. The first subgoal is to detect linear features on a radar image since we know roads appear as linear features in an image. The detected linear features may include non-road objects such as rivers, water passages, radar shadows, noise, and various artifacts. The second subgoal is to screen those non-road linear features. The surviving linear features, called road segments, are not usually connected to each other. The third subgoal is to connect those road segments.

The first subgoal starts with approximating a discrete digital image as a continuous gray-tone intensity surface, which is expressed as a linear combination of tensor products of two set of discrete orthogonal polynomials. Using these continuous surfaces, linear features can be obtained analytically by looking for the first directional derivative zero-crossing.

For the second subgoal, the relationship between the properties of roads appearing in a radar image and the statistics of the maximally connected linear features, called compo-

nents, is analyzed. The properties of roads are connectivity, local straightness, contrast, etc. The statistics derived from a component are the number of pixels, average contrast, average angle difference, average gray-tone value, etc. Using these properties and statistics, the road segments are differentiated from other linear features.

The third subgoal employs a dynamic programming algorithm which finds the minimum cost paths between disconnected road segments. The cost used here is evaluated for each pixel by the cost evaluation function which measures how close the statistics of a pixel are to those of an assumed real road pixel.

Noisy images, of course, are preprocessed before being processed by the above procedure. For the preprocessing, a noise removal algorithm using the local gradient and statistics is employed. This algorithm filters a noisy radar image efficiently without contrast loss.

The noise removal algorithm is discussed in Chapter 2. Chapters 3 and 4 deal with the road detection procedure, which is the main body of the thesis. Chapter 4 elaborates the principle and application of the dynamic programming in image processing. Finally, Chapter 5 shows the experimental results on a few SAR (Synthetic Aperture Radar) images.

CHAPTER 2 NOISE REMOVAL ON RADAR IMAGERY

2.1 INTRODUCTION

Noise removal is usually accompanied by image blurring which manifests itself in smeared edges or loss of subtle details. In general, image blurring becomes more serious as the window size of the noise removal operator increases. Image blurring can be divided into two kinds; blurring edges between heterogeneous regions and blurring texture in a homogeneous region. To solve such blurring problems, many adaptive filtering algorithms have been suggested. One of those algorithms is discussed here. The algorithm was originally suggested by Lee [2], [3] under the assumption that the sample mean and variance of a pixel equal the local mean and variance of its neighborhood. The neighborhood is redefined adaptively according to the edge orientation in a high contrast area such that the neighborhood contains only one side of a possible edge. Hence noise is removed along an edge and the edge is enhanced.

In the following section, this algorithm is summarized and then one practical method of estimating a local noise variance is presented.

2.2 ALGORITHM

Let Z_{ij} be the observed brightness of the pixel (i,j) and X_{ij} be the true brightness of the pixel before noise degradation.

a. For the additive noise case:

$$Z_{ij} = X_{ij} + W_{ij} \tag{2.2.1}$$

where W_{ij} is white noise with a zero mean and variance S^2. The mean $\overline{X}_{ij}$ and variance Q_{ij} are approximated by the local mean and variance in a chosen window centered around (i,j).

$$\overline{X}_{ij} = \overline{Z}_{ij} \tag{2.2.2}$$

$$Q_{ij} = E [(Z_{ij} - \overline{Z}_{ij})^2] - S^2 \tag{2.2.3}$$

The local variance Q_{ij} is thresholded such that the area with a higher value of Q_{ij} than a certain threshold is regarded as containing an edge. The threshold is image dependent (e.g. for the radar images, the threshold of 5,000,000.0 gave the best results). The estimated brightness of the pixel (X^*_{ij}) is obtained by minimizing the mean square error.

$$X^*_{ij} = \bar{X}_{ij} + K_{ij} (Z_{ij} - \bar{X}_{ij}) \tag{2.2.4}$$

where

$$K_{ij} = \frac{Q_{ij}}{Q_{ij} + S^2} \tag{2.2.5}$$

In implementation, the noise variance S^2 is estimated by the average of the smallest five local variances in a block. Hence, the value of Q_{ij} will be always positive and the value K_{ij} is between one and zero.

b. For the multiplicative noise case:

$$Z_{ij} = X_{ij} U_{ij} \tag{2.2.6}$$

where

$$E [U_{ij}] = \bar{U}_{ij} \tag{2.2.7}$$

The mean and variance of X_{ij} are given by

$$\bar{X}_{ij} = \bar{Z}_{ij} / \bar{U}_{ij} \tag{2.2.8}$$

$$Q_{ij} = \frac{VAR(Z_{ij}) + Z^2_{ij}}{S^2 + \overline{U}^2} - \overline{X}^2_{ij} \qquad (2.2.9)$$

The estimated true brightness of the pixel (X^*_{ij}) is obtained by applying the Kalman filtering algorithm to the above equations.

$$X^*_{ij} = \overline{X}_{ij} + K_{ij} (Z_{ij} - \overline{U}_{ij} \overline{X}_{ij}) \qquad (2.2.10)$$

where

$$K_{ij} = \frac{\overline{U}_{ij} Q_{ij}}{\overline{X}^2_{ij} S^2 + \overline{U}^2_{ij} Q_{ij}} \qquad (2.2.11)$$

Note that there still remains an unsolved problem; that is how to estimate the noise variance. Because the noise variance is unknown and spatially variant in most situations, it is not easy to estimate it correctly. Theoretically, the minimum local variance in the local area may be a good estimate of noise variance. The idea is that the local noise variance can be estimated by the local variance of a flat (or almost flat) area. During our experiments, it was found that this algorithm was sensitive to the estimated noise variance. If the noise variance is obtained from too small a block there may not be many flat areas to select on that block. Consequently the noise variance does not reflect the true situation, and hence the algorithm is not very effec-

tive. On the other hand, if the noise variance is obtained from too large a block it may be too global and it is not a local variance. Hence, the subtle details tend to be lost. Another big factor for choosing a good method of estimating a local noise variance is the processing time (i.e. cpu time). In our experiments it varied from about 1 minute to 40 minutes for processing a 256 x 256-pixel image depending on the method of estimating a local noise variance. One possible solution to this problem is to take one fixed estimate for every local noise variance corresponding to every pixel in the same row of the input image. The fixed estimate for the local noise variance corresponding to the pixel of the ith row and the jth column can be obtained by ordering the local variances (Q_{i1}, Q_{i2} , ..., Q_{in} ; n is the number of pixels in a row.) in size, choosing several of the smallest local variances, and averaging them. The local variance corresponding to a pixel is defined as the variance of the pixel values in the neighborhood of the pixel. Only one noise variance is estimated for each row so that a great amount of cpu time is saved. Although one noise variance for each row sounds too global, it turns out to be a computationally efficient method with high quality of noise removal and edge enhancement. The definition of a neighborhood using a local gradient is described next.

This algorithm assumes that for any pixel with a high local variance over a certain threshold, there exists an edge in its neighborhood. By this assumption, in a high contrast region (i.e. edge area), the local statistics obtained from either side of an edge are more reliable than those obtained from its whole neighborhood. For such a high local variance pixel, its neighborhood is redefined in the following way.

1. The direction of a possible edge is obtained by applying directional gradient masks to the window centered around the pixel.
2. The side of the edge that the center pixel belongs to is determined.
3. The side of the edge determined in the above step is defined as the neighborhood of the pixel.

In implementation, four directional gradient masks were used. Hence only four directions of an edge and eight types of subarea as a redefined neighborhood were possible (see Figure 5). The threshold for the local variance determines the presence of an edge in the window centered around each pixel such that a lower threshold gives more edge areas for which the neighborhoods are redefined.

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

Type 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
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
```

Type 2

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

Type 3

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

Type 4

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

Type 5

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

Type 6

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

Type 7

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

Type 8

Figure 5. Eight types of neighborhood: The neighborhood is redefined in a high contrast region as a area marked by 1's when a 7 x 7 window is used.

In the next section, four noise removal algorithms including the one described above and their filtering results on radar images are discussed briefly. The comparison among those filtering techniques is then summarized at the end of the section.

2.3 EXPERIMENTS

A set of radar images was used as test images. This set consists of twenty eight areas in Alabama and North Carolina. Each image consists of 512 x 512 pixels. Four different noise removal filters were tried on each of the images. The four filters were the weighted median filter, the average image filter, the Gaussian filter, and the adaptive filter which was introduced in the previous section.

WEIGHTED MEDIAN FILTER

The weighted median filter takes the median value of all the pixel values in a window centered around each pixel. The pixels are weighted such that a pixel with a weight, say 3, is counted 3 times in computing the median value. The traditional median filter is a special case of this filter with equal wights. Generally this filter removes noise without image blurring when the window size is not greater than 3 x 3 pixels. However, for an image which needs a larger size window or an image whose pixels have the values close to the median of their neighborhood, this filter does not work effectively.

Two weighted windows were used during the experiments (see Figure 6). Most of the radar image noise were removed more

efficiently by using a 3 x 3-pixel window than a 5 x 5-pixel window. It was also found that iterative application of this filter on the images does not have a better effect on the noise removal than filtering once with a larger size window.

EQUALLY WEIGHTED AVERAGE FILTER

This filter takes the average value of all the pixels in a window centered around a pixel as the value for the pixel. It removes noise very efficiently on a homogeneous region containing a coarse textural pattern at a very fast processing speed. However, on a heterogeneous region or even a homogeneous region with a fine textural pattern, it blurs the image significantly. Unfortunately, most images of our experiments contain many such regions and hence the results were not very good. During the experiments, 3 x 3-pixel and 5 x 5-pixel windows were used.

GAUSSIAN FILTER

The Gaussian filter takes an average value of weighted pixels in a window centered around a pixel as the value for the pixel. Each pixel value in a window has a weight which associates with the value of a normal probability density function such that the pixels far away from the center pixel have smaller weights. With this filter, the contrast of the image

1 1 1 1 3 1 1 1 1	1 1 1 1 1 1 2 2 2 1 1 2 4 2 1 1 2 2 2 1 1 1 1 1 1
3 x 3 window	5 x 5 window

Figure 6. Weighted windows used in the experiments

is maintained and the image has a good visual appearance. The noise is generally removed for any kind of texture. However the image is easily blurred because the filter has the property of averaging. In our experiments, it removed the noise well on most of the images with good visual effect but with some image blurring. The 5 x 5-pixel window with 0.75 standard deviation was used.

ADAPTIVE FILTER

Since the neighborhood of each pixel is redefined adaptively within a chosen window, the average number of pixels involved in computing an output for each pixel is relatively smaller than that of any other filter when the same size window is used. Therefore it does not blur an image so much as the others. It enhances edges and contrast of an image. In homogeneous regions, however, it still tends to blur an image and wash out details such as thin lines, fine textural patterns, or weak edges. This undesirable blurring in homoge-

neous regions can be reduced by lowering the threshold for the local variance which determines the presence of an edge. In our experiments, most of the radar images were contrast stretched and edge enhanced with a minimal loss of detail.

SUMMARY OF EXPERIMENTS AND COMMENTS

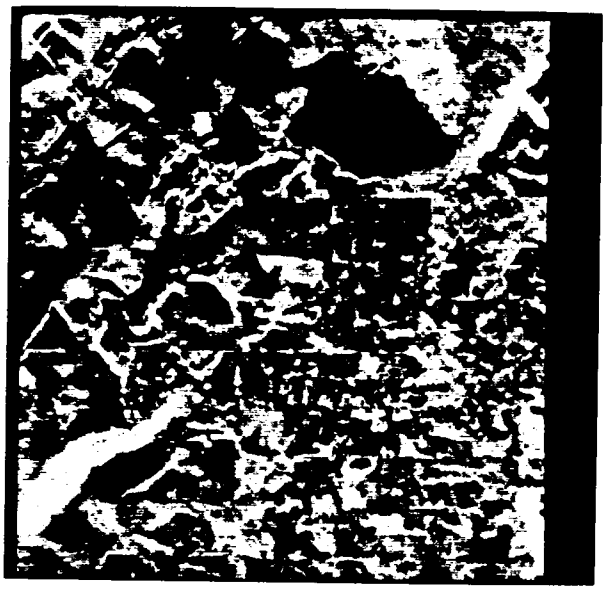
The filtering results on radar images using the above four filters can be summarized on the basis of their visual effect (see Figure 7). The visual effect is judged in four aspects: noise removal, contrast stretching, edge enhancement, and preservation of texture.

The adaptive filter using local gradient and local statistics removes noise without contrast loss or edge blurring. In a homogeneous region it still tends to blur an image because it counteracts the blurring problem only in heterogeneous regions. The image blurring, however, can be reduced by lowering the threshold on the local variance which decides whether a region is homogeneous or heterogeneous. As summarized in Figure 7, the weighted median filter works better than the adaptive filter in terms of preservation of texture. This fact suggests a study of various adaptive filters such as an adaptive weighted median filter, adaptive Gaussian filter, adaptive averaging filter or a combination of them. The final results on a few radar images, by the adaptive

	noise removal	contrast stretch	edge enhancement	texture preservation
Weighted median	good	fair	fair	good
Equal wgt averaging filter	fair	fair	fair	fair
Gaussian filter	good	excellent	fair	fair
Adaptive filter	good	excellent	excellent	fair

Figure 7. Comparison of the filters: The comparison is based upon the filtering results of the set of radar images

filter introduced in this report, are shown in Figures 8 through 10.



a. Original Image

b. Filtered Image

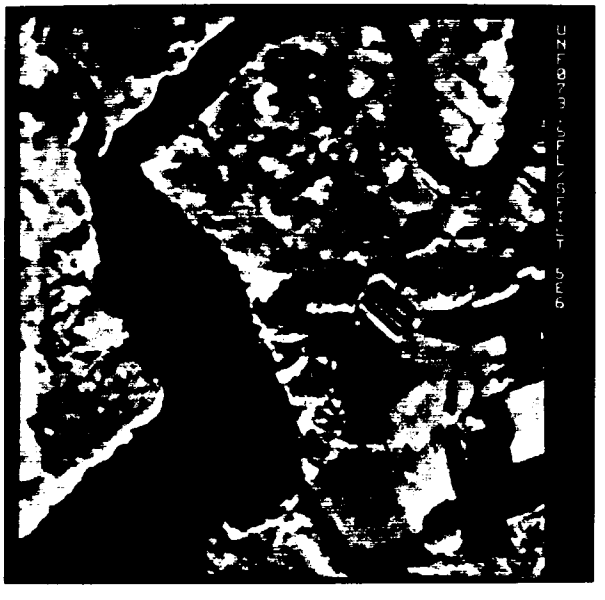
Figure 8. Huntsville area: Local variance threshold = 5,000,000

a. Original Image

b. Filtered Image

Figure 9. Small town: Local variance threshold = 5,000,000

a. Original Image



b. Filtered Image

Figure 10. Elizabeth city area: Local variance threshold = 5,000,000

CHAPTER 3 LINE DETECTION

3.1 INTRODUCTION

To detect linear features, a modified version of the ridge-valley algorithm originally suggested by Haralick [4] is used. In this chapter, the algorithm is discussed from the view point of road detection on a radar image. A road on a radar image usually appears as a dark line which can be loosely defined as a connected sequence of pixels having significantly lower gray-tone intensity than the neighbors. In a continuous gray-tone surface, the dark line would be viewed as a valley which could be computed analytically; the bottom of a valley would occur where the first directional derivative of the surface has a zero-crossing in a direction extremizing a positive second derivative. Such a surface can be obtained from a discrete image by a facet model generating function which is discussed in Section 2. Then, a facet based line detection algorithm is discussed in Section 3.

3.2 FACET MODEL

The idea of using a fitted continuous model in image processing was first proposed by Prewitt [13]. Haralick [2] calls

such a fitted model for image data a facet model. A facet model can be produced by a facet generating function K;

$$K : D(I) \times O \times \{B\} \times \{I\} \dashrightarrow f^{*}(R) \qquad (3.2.12)$$

The D(I) is a discretely sampled data function for a fixed neighborhood. The parameter O is the order of fit. For example, when using a second order fit, the resulting facet model will be a quadratic polynomial. The index set {I} is the domain on which the discretely sampled data function D(I) and the set of basis functions {B} are defined. The function $f^{*}(R)$, a real valued function, is called a facet model for the neighborhood. Here, we discuss how the function K maps arguments into a facet model when {B} is a set of discrete Chebyshev polynomials, O, a third order of fit, and {I}, a two-dimensional integer index set $\{(r,c) \mid -n \le r \le n \text{ and } -m \le c \le m\}$. Since the fit is third order and {I} is a two-dimensional index set, it is assumed that the underlying continuous gray-tone intensity surface from which D(I) is discretely sampled, takes the parametric form of a bicubic polynomial defined in the row-column coordinates. The canonical form for the function f can be expressed by

$$\begin{aligned} f(r,c) = {} & k_1 + k_2 r + k_3 c + k_4 r^2 + k_5 rc + k_6 c^2 \\ & + k_7 r^3 + k_8 r^2 c + k_9 rc^2 + k_{10} c^3 \end{aligned} \qquad (3.2.13)$$

The function f^* is then obtained by estimating the coefficients of Eq. (3.2.13) from the discrete samples of D(I). The least square error method directly applied to Eq. (3.2.13) can give the solution to the problem of estimating the coefficients. However, the computation is more complex and less numerically stable in the canonical form. An alternative is to express f^* in a linear combination of discretely orthogonal polynomials {B}. Let the basis {B} be $\{R_0, \ldots, R_{N-1}\}$. Then, the function f^* is expressed by

$$f^*(r,c) = \sum_{n=0}^{N-1} b_n R_n(r,c) \tag{3.2.14}$$

By the least square error method, the coefficients $b_0, \ldots, b_{N-1}$ of Eq. (3.2.14) are estimated instead of the coefficients $k_1, \ldots, k_{10}$ of Eq. (3.2.13). By doing so, we can use the orthogonality of the basis functions {B}, and hence achieve much less complex computation.

The discrete Chebyshev polynomials defined on the integer index set {(r,c)} are constructed inductively in the following way. Let P_n be the nth order polynomial. Define $P_0 = 1$ and suppose $P_1(r), \ldots, P_{n-1}(r)$ have been constructed. Then $P_n(r)$ must be orthogonal to each polynomial $P_0(r), \ldots, P_{n-1}(r)$. That is,

$$\sum_{r} P_k(r)\, P_n(r) = 0, \text{ for all } k \le n-1 \qquad (3.2.15)$$

where $P_n(r) = r^n + a_{n-1}r^{n-1} + \ldots + a_1 r + a_0$. Solving the unknowns $a_0, \ldots, a_{n-1}$ from the n linear equations, we construct $P_n(r)$.

The set $\{P_0(r), \ldots, P_n(r)\}$ is a one-dimensional discrete orthogonal polynomial set for the integer index set $\{r\}$. A two-dimensional discrete orthogonal polynomial set for the integer index set $\{(r,c)\}$ can be easily constructed by taking the tensor products of two one-dimensional sets, $\{P_0(r), \ldots, P_n(r)\}$ and $\{Q_0(c), \ldots, Q_n(c)\}$. For example, on the two-dimensional index set $\{-1, 0, 1\} \times \{-1, 0, 1\}$, the discrete Chebyshev polynomial set can be defined as the tensor products of $\{1, r, r^2 - 2/3\}$ and $\{1, c, c^2 - 2/3\}$, which yields $\{1, r, c, r^2 - 2/3, rc, c^2 - 2/3, r(c^2 - 2/3), c(r^2 - 2/3), (r^2 - 2/3)(c^2 - 2/3)\}$.

Now we go back to the problem of estimating the coefficients of Eq. (3.2.14). Let $D(r,c)$ be the discretely observed data function. Then, the residual square error is expressed by

$$e^2 = \sum_{(r,c)} [D(r,c) - \sum_{n=0}^{N-1} b_n R_n(r,c)]^2 \qquad (3.2.16)$$

The coefficient, b_i of the ith order discrete orthogonal polynomial $R_i(r,c)$ can be computed by taking the partial derivative of e^2 with respect to b_i, setting it to zero and solving the resulting equation.

$$\frac{\partial e^2}{\partial b_i} = -2 \sum_{(r,c)} [D(r,c) - \sum_{n=0}^{N-1} b_n R_n(r,c)] [R_i(r,c)] \quad (3.2.17)$$

Using the orthogonality of the polynomials $R_0(r,c)$, ..., $R_{N-1}(r,c)$, we know

$$\sum_{(r,c)} [R_j(r,c) R_i(r,c)] = 0, \text{ for all } j \neq i \quad (3.2.18)$$

Thus the Eq. (3.2.17) is reduced to

$$\frac{\partial e^2}{\partial b_i} = 2 b_i \sum_{(r,c)} [R_i(r,c)]^2 - 2 \sum_{(r,c)} [D(r,c) R_i(r,c)] \quad (3.2.19)$$

Setting the partial derivative of e^2 with respect to b_i to zero the estimate b_i^* of the coefficient b_i is obtained.

$$b_i^* = \sum_{(r,c)} [D(r,c) R_i(r,c)] / \sum_{(r,c)} [R_i(r,c)]^2 \quad (3.2.20)$$

It should be noted that each coefficient b_i ($i \leq N-1$) is just a linear combination of the data values $D(r,c)$. In other words, b_i^* is simply obtained by multiplying the data value $D(r,c)$ by the weight

$$R_i(r,c) / \sum_{(r,c)} [R_i(r,c)]^2 \tag{3.2.21}$$

for each index (r,c). Once the coefficient b_i ($i \le N-1$) have been computed, the estimate f^* of the underlying continuous function f is given by

$$f^*(r,c) = \sum_{n=0}^{N-1} b_n^* R_n(r,c) \tag{3.2.22}$$

Eq.(3.2.22) gives the estimates of the coefficients k_1, ..., k_{10} of the bicubic polynomial Eq. (3.2.13).

In this section, we have discussed one instance of facet models which can be generated by the facet generating function K. It is notable that various facet models can be generated by changing arguments in the function K. Image processing techniques based on a facet model have been developed. Among them are the directional derivative edge detector [6] and the ridge-valley finder [4]. In the next section we discuss a line detection algorithm based on a bicubic facet model, which is the modified version of the ridge-valley finder, and is used for the detection of lines having a constant width.

3.3 ALGORITHM

The canonical form of the fitted bicubic surface for each pixel's neighborhood of an input image is

$$f(r,c) = k_1 + k_2 r + k_3 c + k_4 r^2 + k_5 rc + k_6 c^2 + k_7 r^3 + k_8 r^2 c + k_9 rc^2 + k_{10} c^3 \qquad (3.3.23)$$

The first directional derivative of a function f in the direction α, f'_α is defined by

$$f'_\alpha(r,c) = \lim_{d \to 0} \frac{f(r + d \sin\alpha,\ c + d \cos\alpha) - f(r,c)}{d} = \frac{\partial f}{\partial r}(r,c) \sin\alpha + \frac{\partial f}{\partial c}(r,c) \cos\alpha \qquad (3.3.24)$$

Similarly, the second directional derivative of f is defined by

$$f''_\alpha(r,c) = \frac{\partial^2 f}{\partial r^2}(r,c) \sin^2\alpha + 2 \frac{\partial^2 f}{\partial r \partial c}(r,c) \sin\alpha \cos\alpha + \frac{\partial^2 f}{\partial c^2}(r,c) \cos^2\alpha \qquad (3.3.25)$$

The direction α for which f''_α is an extremum can be determined by setting the derivative of f''_α with respect to α to zero.

$$\frac{\partial f''_\alpha}{\partial \alpha} = \left(\frac{\partial^2 f}{\partial r^2} - \frac{\partial^2 f}{\partial c^2} \right) \sin 2\alpha + 2 \frac{\partial^2 f}{\partial r \partial c} \cos 2\alpha \qquad (3.3.26)$$

Solving Eq. (3.3.26) we obtain

$$\sin 2\alpha = \pm (-2\partial^2 f/\partial r \partial c)/D \quad \text{and}$$

$$\cos 2\alpha = \mp (\partial^2 f/\partial r^2 - \partial^2 f/\partial c^2)/D \qquad (3.3.27)$$

where

$$D = [\ 4\ (\partial^2 f/\partial r \partial c)^2 + (\partial^2 f/\partial r^2 - \partial^2 f/\partial c^2)^2]^{0.5}$$

Substituting Eq. (3.3.23) into Eq. (3.3.27), we can get the direction α at the center of each pixel, which is the direction normal to an assumed valley line through the pixel.

$$\alpha = -\ 0.5 \tan^{-1} k_5/(k_6 - k_4) \qquad (3.3.28)$$

Using the angle α to constrain a position (r, c), we know

$$r = \rho \sin\alpha \quad \text{and} \quad c = \rho \cos\alpha$$

Thus, the cross-section of the surface in the direction α through the origin is given by

$$f_\alpha(\rho) = A\rho^3 + B\rho^2 + C\rho + k_1 \qquad (3.3.29)$$

where

$$A = k_7 \sin^3\alpha + k_8 \sin^2\alpha \cos\alpha + k_9 \cos^2\alpha \sin\alpha + k_{10} \cos^3\alpha,$$

$$B = k_4 \sin^2\alpha + K_5 \sin\alpha \cos\alpha + k_6 \cos^2\alpha, \text{ and}$$

$$C = k_2 \sin\alpha + k_3 \cos\alpha. \qquad (3.3.30)$$

The typical curve of Eq. (3.3.29) is shown in Figure 11. In the figure R1 and R2 are the zero-crossings of Eq. (3.3.29) and L and U are the lower and upper limit of domain which is determined by a window size. Assume that R1 is closer to the center of pixel than R2. Then we can declare the pixel as a line pixel if all of the following conditions are satisfied:

1. The first derivative of Eq. (3.3.29) has a zero-crossing sufficiently near the pixel's center and the second derivative is positive at the zero-crossing point;

$$|R1| < \text{threshold}_1 \quad \text{and} \quad \frac{d^2 f_\alpha(R1)}{d\rho^2} > 0 \qquad (3.3.31)$$

2. The second derivative is big enough;

$$\left| \frac{d^2 f_\alpha(0)}{d\rho^2} \right| > \text{threshold}_2 \qquad (3.3.32)$$

3. The gray-tone of the pixel is within a proper line

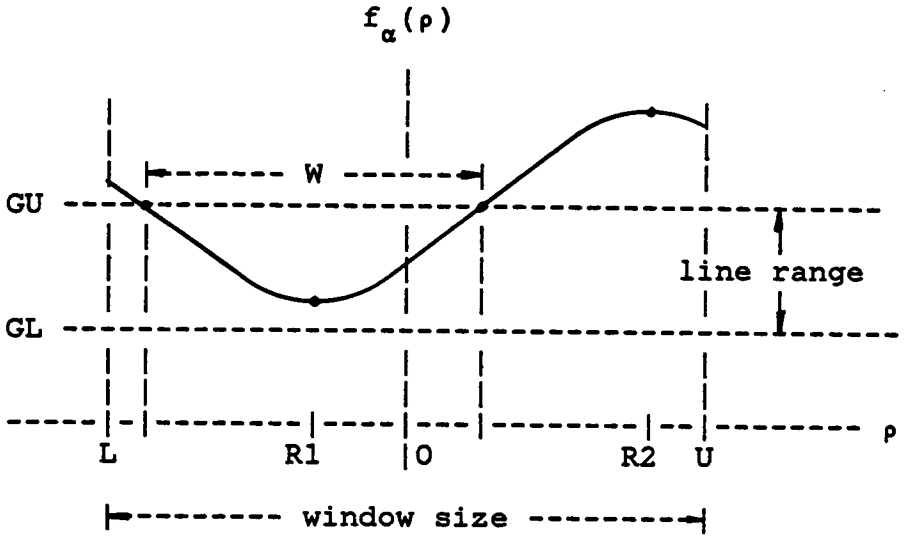


Figure 11. Cross section of surface

range;

$$threshold_3 \le f_\alpha(R1) \le threshold_4 \qquad (3.3.33)$$

4. The contrast of the pixel with its surrounding is sufficiently large;

$$MIN\ [f_\alpha(L),\ f_\alpha(R2)] - f_\alpha(R1) > threshold_5 \qquad (3.3.34)$$

(The thresholds are provided by the user)

Once a line pixel is identified, it is assigned four property values which are shown in Figure 11 and defined as follows:

1. Depth, $D = f_\alpha(R1)$
2. Strength, $S = MIN\ [f_\alpha(L),\ f_\alpha(R2)] - f_\alpha(R1)$
3. Angle, $A = \alpha$
4. Width, W = the number of pixels whose gray-tone values are in the line range.

These property values can be utilized in further processing. For example, by setting W to 2 we can detect two pixel wide lines and then by restricting A we can find two pixel wide lines at a certain orientation.

In this section we have discussed the line pixel detection algorithm on a pixel by pixel base. That is, based on the bicubic facet model for each pixel, we looked for a zero-crossing point which satisfies a set of constraints. If such a point was found within its neighborhood the pixel was declared to be a line pixel.

3.4 EXPERIMENTS

The line detection algorithm was tested on a few SAR images. The results are dependent on the combination of the thresholds. Figure 12 through Figure 14 show the effects of the thresholds for which a 100 x 100 subimage shown in Figure 33 was used as a test image.

The $threshold_1$ is called radius. Figure 12 shows how the radius affects the results. It is noted that a bigger radius value gives thicker lines. For the purpose of this thesis, a radius between 1.2 and 1.5 turned out to be the most effective.

The $threshold_2$, called curvature, has the effect of eliminating the artifacts but also has a tendency to break a line into shorter pieces, as shown in Figure 13. A curvature around 750 was best for the image with a gray-tone scale of 0 through 65,535.

The $threshold_3$ and $threshold_4$ give the gray-tone bounds within which line pixels should lie. Using these thresholds we obtain lines in our intended gray-tone range. They have the same effect as the intensity line operator described in Fischler, et al [7]. Their effects on the result are shown in Figure 14.

$Threshold_5$, called contrast, reflects the human visual recognition of lines. That is, a vivid line in a bright area may not be as perceptible as it is in a darker area. As a result, the higher contrast threshold gets rid of relatively vague lines and tends to loose useful information. For the next process which requires a lot of information, this threshold had to be lowered.

The overall results of the line detection technique are shown in Figures 15 and 16.

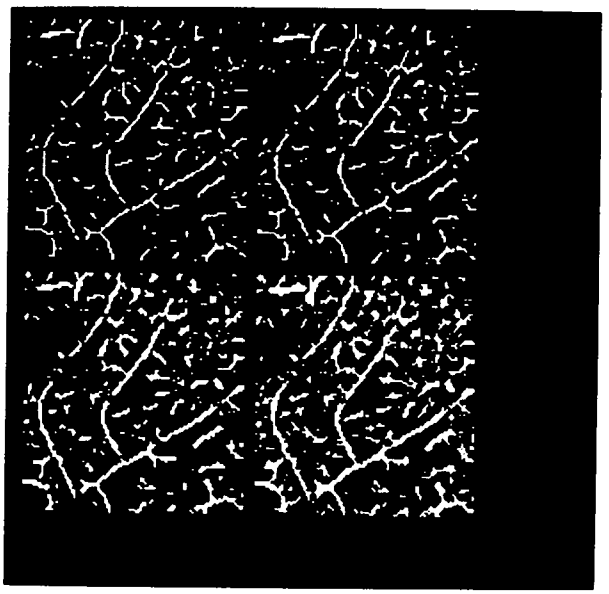


Figure 12. The effect of radius: Upper left (radius = 0.75), Upper right (radius = 1.00), Lower left (radius = 1.50), Lower right (radius = 2.00)

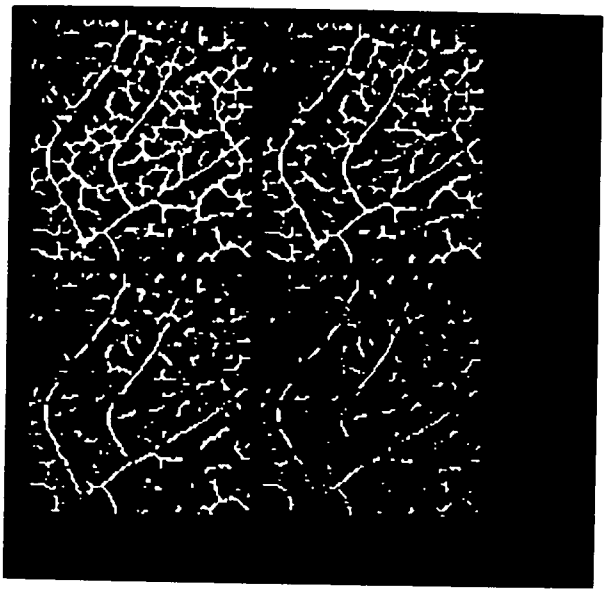


Figure 13. The effect of curvature: Upper left (curvature = 100), Upper right (curvature = 500), Lower left (curvature = 750), Lower right (curvature = 1000)

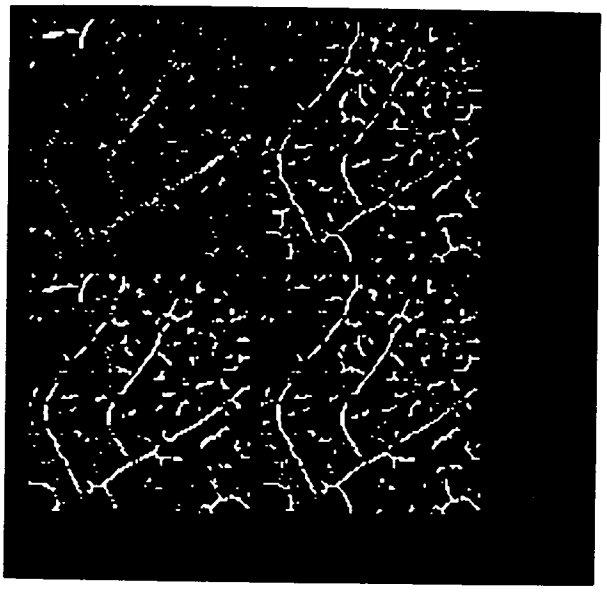
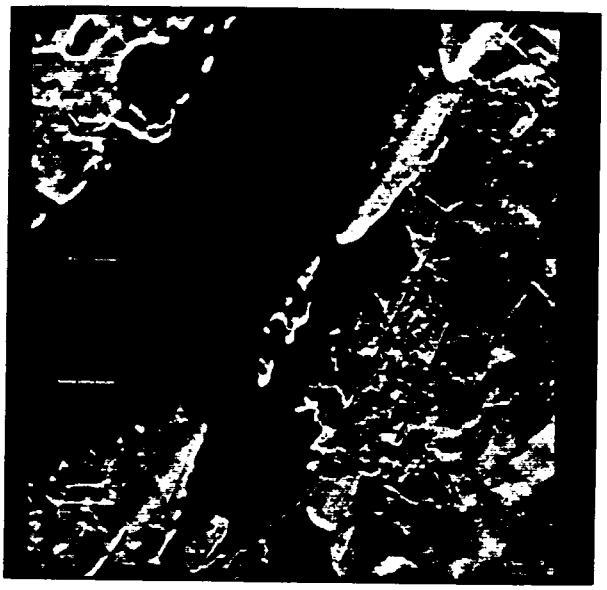
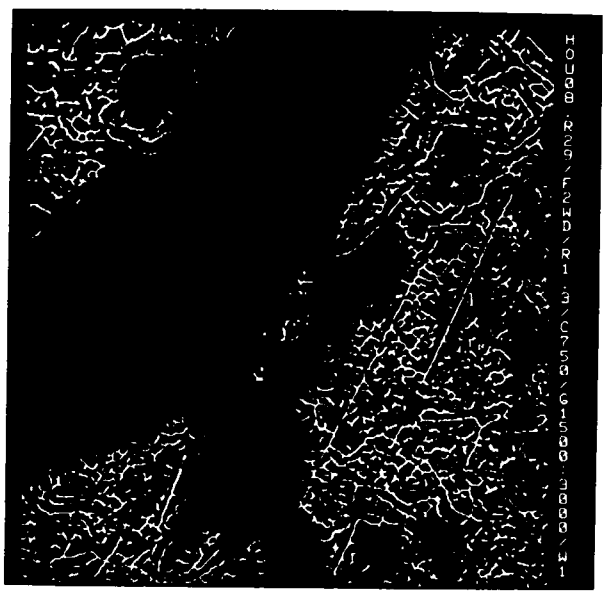


Figure 14. The effect of gray-tone range: Upper left (equiv. range = 1000 -- 2000), Upper right (equiv. range = 2000 -- 3000), Lower left (equiv. range = 1500 -- 2500), Lower right (equiv. range = 1800 -- 2700)



a. Original image



b. Line image

Figure 15. Huntsville area: Curvature = 750, Radius = 1.2, Equiv. range = 1500 thru 3000, Contrast = 800, Width = 1 thru 4

a. Original image

b. Line image

Figure 16. Elizabeth city area: Curvature = 750, Radius = 1.4, Equiv. range = 1800 thru 2700, Contrast = 1000, Width = 1 thru 4

CHAPTER 4 ROAD DETECTION AND CONNECTION

4.1 INTRODUCTION

The line detector discussed in the previous chapter gives plenty of information about a line pixel such as strength, gray-tone value, angle, and width. However, the information provided by the detector includes many non-road linear objects. At the beginning of this research, these non-road linear objects seemed to be easy to remove by adjusting the thresholds of the line detector. This resulted in the waste of time in search of a proper combination of the thresholds. But it was soon learned and the idea of screening the linear features on a connected component by connected component basis, not on a pixel by pixel basis developed. The screening technique uses the statistics of each connected component. To get the necessary statistics efficiently, the line pixels are connected first, using 8-connectivity. Each maximally connected set of pixels is called a component. Each component is then assigned a unique integer identifier. The statistics can be derived from each component. The surviving components after the screening procedure using the statistics are called road segments. The pixels belonging to a road segment are called road pixels. The details are discussed in Section 2.

Other important factors in the screening process are the road characteristics appearing in radar imagery. Those characteristics are:

1. The road should be connected.
2. The road should be a linear feature.
3. The road should be sufficiently straight or smoothly curved.
4. The road should be contrasted with its surroundings.
5. The road should have a sufficiently low gray-tone intensity.

On the basis of these road characteristics in conjunction with the statistics, the components are screened.

4.2 ROAD DETECTION

Road detection is closely related to road connection and can be viewed as its preprocess. Here the statistics which are the screening criteria for the components on the basis of the road characteristics are defined and discussed. First, a simple example shows the effect of screening process on the following connection.

If the road connection procedure did not exist and the system ended with road segments detected from the linear features, the road detection part would have been greatly different.

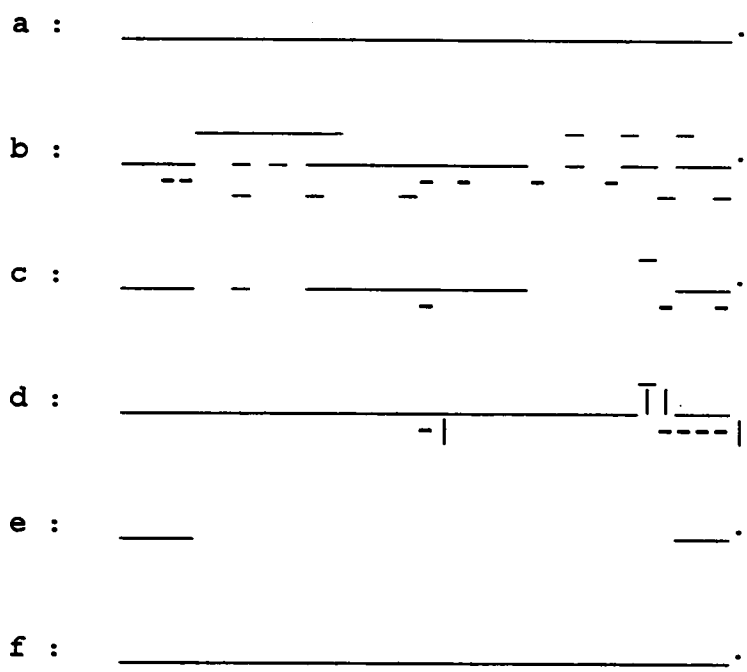


Figure 17. Detection and connection

Figure 17 illustrates the difference through the example of a very simple road network (Figure 17a). Figure 17b shows the linear features detected from Figure 17a. In the system without a connection procedure, the road detector would be designed to detect as many road segments as possible, even though the result may include a few non-road linear features (see Figure 17c). In the system with a connection part, however, the result shown in Figure 17c is not very desirable because the connection procedure will connect all segments including falsely marked road segments, which yields a different network (Figure 17d). To make it worse, it would take

a longer processing time than the result with fewer segments because the connection procedure would be invoked more frequently. The most desirable result of the road detection in the system with a connection part is shown in Figure 17e, which requires only one invocation of the connection procedure to reconstruct the original network (Figure 17f). Therefore, the road detection in the system with a connection procedure should give the smallest number of road segments that can reconstruct the original network. The road detection procedure used in this thesis can be viewed as a thresholding procedure with a simple combining mechanism. This procedure uses the statistics to threshold the components, and then combines any pair of surviving components which are one pixel apart, to reduce the number of road segments. These statistics are derived from each component. Those are the number of pixels in the component (N), the mean line strength (MS), the standard deviation of line strengths (DS), the mean angle difference (MB), the mean gray-tone value (MG), and the standard deviation of gray-tones (DG). These statistics are totally dependent on the set of thresholds used in the previous line detection. The road detection is based upon the relation between these statistics and road characteristics appearing in radar imagery. The intuitive meaning of each statistic is discussed next.

(1) The number of line pixels in a component (N): This value is important for identifying the real road segment. A road network is completely connected and the things that break the network into pieces are the noise in the image and the possible error in our bicubic surface. In any noisy area in a image, unless the area is completely damaged by the noise, a road segment would have at least a certain number of connected pixels. Under the assumption that the N value of a non-road component is relatively small, a component with a sufficiently large N is considered to be a road segment. The threshold against N should be neither too small nor too large because too small a threshold would cause non-road components to be mistaken for road segments, and too large a threshold would result in failure to pick up road segments. An appropriate value of the threshold against N may vary from image to image. In these experiments the most effective value was found to be between 8 and 12.

(2) The mean line strength of a component (MS): The mean line strength of a component is defined as the average of strengths of pixels belonging to the component. The strength of a pixel is, as defined in the previous chapter, the difference in the value of two extreme points of the cross-sectioned surface in a direction extremizing the second directional derivative.

The MS value indicates the contrast of a component with its neighboring region running parallel to the component. The contrast of a component for a likely road segment is assumed higher than that of a non-road component. This assumption is not always true and so the real road components are thrown away as well as non-road components when thresholding against this parameter. However, in many cases, giving up a few real road segments can eliminate many non-road components. The line pixels in the abandoned road segments are re-used for connecting the road segments later in the road connection part.

(3) The standard deviation of line strengths (DS): The standard deviation of line strengths of a component is defined as the standard deviation of the strengths of pixels belonging to the component.

The line strength (contrast) of each pixel in a component for a likely road segment is more uniform than that in a noisy component because the gray-tone intensity of a radar image is locally homogeneous. Hence, the smaller DS value a component has, the more likely the component is a road segment, especially when the component is not so large that it will lie on two or more different regions.

(4) The mean angle difference in a component (MB): The mean angle difference in a component is defined as the average of the angle differences of pixels belonging to the component. The angle difference of a pixel is defined only within a component as a difference in angles between adjacent pixels of the same component. Since a line pixel usually has more than one adjacent pixel with the same label, the priority of the adjacent pixels are set up in a way that the angle difference of two adjacent pixels may not be used twice if possible. The priority set-up used in these experiments is shown in Figure 18.

The pixel with the highest priority (the lowest number) is chosen as the adjacent pixel of center pixel X if the pixel and the center pixel are in the same component.

The MB value is a good measurement for the curvature of a component. Ideally, a perfect circle composed of 36 pixels may have 10° MB and a straight line has 0° MB. In reality, the MB value does not reflect the true situation accurately. The inaccuracy is mainly due to the error involved in the discrete data. In our experiments, components with MB values less than 15° were considered as candidates to be real road segments.

7	4	8
3	X	1
6	2	5

Figure 18. The priorities of the eight adjacent pixels

(5) The mean gray-tone value of a component (MG): The mean gray-tone value of a component is defined as the average of the gray-tone values of pixels belonging to the component.

This value is another important tool for differentiating a component for a possible road segment from a non-road component. Even if a component has a relatively high MS value without a sufficient darkness, the component cannot be a real road segment. It is because the surface roughness of a road segment does not vary enough to change the gray-tone, place to place, in the same image. The MG value of a road component should be in a certain range of gray-tone. Although this range is dependent upon an image brightness it is usually between 22,000 and 28,000 on a gray-tone scale of 0 to 65,535 for the collection of radar images in our library.

(6) The standard deviation of gray-tone values (DG): The standard deviation of gray-tone values of a component is defined as the standard deviation of gray-tone values of pixels belonging to the component.

This is a supplementary tool to the mean gray-tone. The gray-tone values of a non-road component are assumed to have a greater standard deviation than those of a component for a likely road segment. Re-thresholding the components against the DG values can keep the surviving non-road components from being mistaken for road components.

4.3 ROAD CONNECTION

To complete the road networks, all the detected road segments are connected using the non-road pixels, whether they are line pixels or not. Among many possible connecting paths between two road segments, the path with the maximum likelihood (or the minimum cost) is chosen by a dynamic programming method. To do so all non-road pixels are evaluated by a cost evaluation function. The technique of dynamic programming and the strategy of cost evaluation function are the main subject of this section.

4.3.1 DYNAMIC PROGRAMMING

Dynamic programming is a very useful approach to optimization. It is based on the "principle of optimality"; any subpolicy of an optimum policy must be optimum itself [8]. The optimality principle is applied to finding the minimum cost path between two road segments. Similarly it holds that

any subpath of the optimum (or the minimum cost) path from one (starting) segment to another (goal) segment must be optimum. The relationship of the subpath to the whole path can be defined recursively. The technique for solving an optimum path is generally described and then a tailor-made algorithm for our purpose is presented.

As shown in Figure 19, the minimum cost path from S to T is sought. Let X_i (i = 1, 2, ..., n) be a set of pixels, one of which must be part of a possible minimum cost path from S to T.

Let g(a,b) be the function which gives the minimum cost from a to b. Now suppose that the minimum cost path between S and T contains X_k. Then, the minimum cost from S to T is given by

$$g(S,T) = g(S, X_k) + g(X_k, T) \tag{4.3.35}$$

In general, the k of X_k is unknown in advance and the minimum cost is rewritten by

$$g(S,T) = \underset{i}{\mathrm{MIN}} [g(S, X_i) + g(X_i, T)] \tag{4.3.36}$$

Consider the special case where X_i's are the eight neighbors of the goal T and $g(X_i, T)$ is the constant, CT for all X_i's,

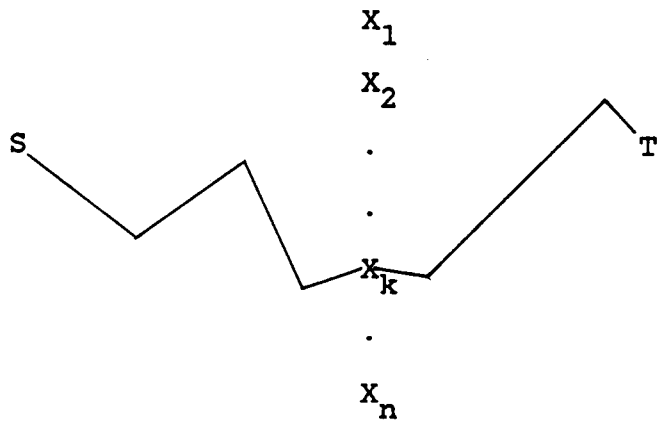


Figure 19. The minimum paths between S and X_i and T

which is the cost of advancing from each of the X_i's to T (see Figure 20)

Then, the minimum cost from S to T can be expressed by

$$g(S, T) = \underset{i}{\mathrm{MIN}}\ g(S, X_i) + CT \tag{4.3.37}$$

Similarly each of the $g(S, X_i)$'s is reduced to

$$g(S, X_i) = \underset{j}{\mathrm{MIN}}\ g(S, Y_j) + CX_i \tag{4.3.38}$$

where Y_j ($j = 1, 2, \ldots, 8$) are the eight neighbors of X_i and CX_i is the cost of advancing from the Y_j's to X_i. Accordingly, we obtain the minimum cost from S to T by

$$g(S, T) = \underset{i}{\mathrm{MIN}}\ (\ \underset{j}{\mathrm{MIN}}\ g(S, Y_j) + CX_i) + CT \tag{4.3.39}$$

.	S	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	X_1	X_2	X_3	.
.	.	.	X_4	T	X_5	.
.	.	.	X_6	X_7	X_8	.
.	.	.	.	.	.	.

Figure 20. The special arrangement of S, X_i, and T

This procedure is repeated recursively until we get

$$g(S, T) = \underset{i}{\mathrm{MIN}} \left(\underset{j}{\mathrm{MIN}} \left(\ldots \underset{n}{\mathrm{MIN}}\, g(S, Z_n) + \ldots \right. \right. \left. \left. \ldots \right) + CX_i \right) + CT \qquad (4.3.40)$$

where Z_n's are the eight neighbors of the starting pixel, S and $g(S, Z_n)$'s are the known values, CZ_n's, the costs of advancing from S to Z_n's.

This basic idea has been implemented in many application areas since it was presented by Ford [9]. The F^* algorithm used in Fischler, et al. [7] is an implementation nicely fitted to sequential image processing. It is noted that a similar optimization method (e.g. the A^* algorithm in Duda and Hart [10]) can be derived from the best first search algorithm. Our implementation can be thought as a extended version of the F^*, called F^o.

In the experiments run in this research, the F^o algorithm is applied iteratively in the manner of minimum cost spanning tree. Each road segment (a set of pixels) plays the role of a vertex in a graph. Each result of the F^o algorithm applied to each pair of segments plays the role of a weighted edge between vertices. The implementation details are presented here. The prerequisite is a cost image on which the algorithm is exercised. The cost image is computed by a cost evaluation function which will be discussed in the next section. For the time being, the algorithm is assumed to have it. The sets of starting pixels and goal pixels are specified by the real road segment image in which every road segment has a unique label. Consider the following procedure;

While the number of road segments is greater than 1

1. Set a segment to the starting segment.
2. Set the others to the goal segments.
3. Set every element of S to infinity except the starting segment whose elements are set to 0.
4. Find the minimum cost path.
5. Assign the same label to the minimum cost connected segments.

Each iteration of this procedure involves top-to-bottom (and bottom-to-top) passes using the cost image, C, and updates the total cost from the starting segment to each pixel, cre-

ating the image S according to the left-to-right rule, Eq. (4.3.41), and then the right-to-left rule, Eq. (4.3.42), both shown for the top-to-bottom pass.

$$
\begin{aligned}
S(i, j) = \mathrm{MIN}\ [\ & S(i, j), \\
& C(i, j) + \mathrm{MIN}\ [\ S(i-1, j-1), \\
& \qquad S(i-1, j), \\
& \qquad S(i-1, j+1), \\
& \qquad S(i,j-1)\ ]\ ] \qquad (4.3.41)
\end{aligned}
$$

$$S(i, j) = \mathrm{MIN}\ [\ S(i, j),\ \ C(i, j) + S(i, j+1)\ ] \qquad (4.3.42)$$

This is then followed by a bottom-to-top pass using the mirror image equation to equation (4.3.41) and (4.3.42). It usually needs more than one top-to-bottom and bottom-to-top pass until there is a condition of no change. If the optimum path is composed of n row monotonic subpaths, then n alternating passes between a top-to-bottom pass and bottom-to-top pass (see Appendix A for the proof) are required.

The following example shows the operation of the minimum cost path finding algorithm from two sets of source pixels (subscripted by s1 or s2) to two sets of goal pixels (subscripted by g3 or g4). In the global sense, this example can be viewed as the first step of the minimum cost spanning tree through

the four vertices (segments subscripted by s1, s2, g3, or g4) with two designated starting vertices (segments subscripted by s1 and s2). The cost image C(i, j) is shown in Figure 21. The image S is practically set to 999 instead of infinity. During the first top-to-bottom pass, the first row of the image S will appear as follows.

999	999	999	999	999	999	999	999;	initially
999	999	999	999	0	2	1	0;	after applying Eq.(4.3.41)
16	14	9	3	0	2	1	0;	after applying Eq.(4.3.41)

The second row of the image S will appear as follows.

999	999	999	999	999	999	999	999;	initially
7	10	4	5	2	0	1	7;	after applying Eq.(4.3.41)
8	5	4	5	2	0	1	7;	after applying Eq.(4.3.41)

The rest of the image S is updated similarly applying Eq. (4.3.41), and then Eq. (4.3.42). The total cost image S(i, j) after each pass is shown in Figures 22 thru 24. As shown in Figure 23 and Figure 24, there are no updates at the third pass. Accordingly the algorithm is terminated right after the third pass. Since the minimum cost to goal segment g3 (value 4) is less than the minimum cost to goal segment g4 (value 6) the minimum cost path is tracked reversely from g3 by substracting the cost (in the image C) from the total cost

(in the image S), marking the neighbor which has the same value as the substracted total cost, and repeating this process until the neighbor becomes the pixel of any source segment. Thus the minimum cost path (the first edge to span) is found between segments subscripted s1 and g3, and its elements are bracketed in Figure 24. Those segments including the path between them become a new segment for the next iteration (spanning). It should be noted that the algorithm can be terminated even after the second pass. It is because the smallest of the updated values, 5 is greater than the minimum cost, 4 after the second pass (see the asterisked values in Figure 23). In these experiments, both of the terminating conditions were implemented.

4.3.2 COST EVALUATION FUNCTION

The cost of a pixel is defined as the expense for advancing to the pixel from any of its eight neighbors. The basic strategies of cost evaluation are:

(1) A non-line pixel is penalized enough not to be chosen as a part of road network unless absolutely necessary.

(2) A line pixel is rewarded according to its attributes which are the strength (S), angle difference (B), and gray-tone distance (D).

2	5	6	3	0_{s1}	2	1	0_{s2}
3	1	1	5	2	0_{s1}	1	7
1	3	3	3	1	6	7	5
3	4	2	7	2	5	6	4
2	1	1	1	2	1	2	1
3	1	0_{g3}	0_{g3}	2	4	3	2
4	3	5	7	0_{g3}	2	5	7
2	0_{g4}	1	1	3	2	2	1

Figure 21. A sample cost image, C(i, j)

The strength of a line pixel (S) is defined as in the earlier chapter; that is, S is the difference in the values of two extreme points of the cross-sectioned surface in a direction extremizing the second directional derivative. The definition of the angle difference of a line pixel (B) is slightly different from that of angle difference in a component which was discussed in the previous section. The angle difference of a line pixel is defined as follows. First, two neighbors are selected such that they best coincide with the angle of a pixel. Next, two differences of angles are taken: between the angle of the pixel and the angle of each neighbor. The larger difference of the two is then the angle difference of a line pixel. Suppose that the zero angle direction is set to the positive column direction and the an-

16	14	9	3	0_{s1}	2	1	0_{s2}
8	5	4	5	2	0_{s1}	1	7
6	7	7	4	1	6	7	6
9	10	6	8	3	6	12	10
8	6	5	4	5	4	6	7
8	5	4_{g3}	4_{g3}	6	8	7	8
9	7	9	11	4_{g3}	6	11	14
8	6_{g4}	6	5	7	6	8	9

Figure 22. Image of total cost, S(i, j): after the first pass (top to bottom)

gles increase clockwise as shown in Figure 25, and that the line pixel 5 has an angle closer to 45° than to 90°. Then, the two neighbors of pixel 5 become pixel 1 and pixel 9 and the angle difference of pixel 5 is given by

B = MAX [abs (angle of pixel 5 - angle of pixel 1),
abs (angle of pixel 5 - angle of pixel 9)]

(4.3.43)

The angle difference of a line pixel is a measurement for the consistency of direction with its adjacent pixel and for the curvature of a possible line going through the pixel.

7	9	9	3	0_{s1}	2	1	0_{s2}
8	5	4	5	2	0_{s1}	1	7
6	7	7	4	1	6	7	6
8	9	6	8	3	6	10	10
7	5^{*}	5	4	5	4	6	7
8	5	4_{g3}	4_{g3}	6	8	7	8
9	7	9	11	4_{g3}	6	11	14
8	6_{g4}	6	5	7	6	8	9

Figure 23. Image of total cost, S(i, j): after the second pass (bottom to top)

The gray-tone distance of a line pixel (D) is defined as the absolute difference in the gray-tone value of the pixel and an assumed mean gray-tone value of real road segments, which can be decided experimentally during the road detection part discussed in the earlier section.

Since the three attributes of a line pixel are equally important they should have equal parts in the cost evaluation. However, since they are hardly sensed equally by an actual program, it is proper that their effects on the cost evaluation be proportional to their sensitivities and accuracies. For example, in our detection procedure, the angle of a line pixel is not so accurate as the other attributes. Their pro-

7	9	9	3	0_{s1}	2	1	0_{s2}
8	5	4	5	2	0_{s1}	1	7
6	7	7	4	[1]	6	7	6
8	9	6	8	[3]	6	10	10
7	5	5	[4]	5	4	6	7
8	5	4_{g3}	4_{g3}	6	8	7	8
9	7	9	11	4_{g3}	6	11	14
8	6_{g4}	6	5	7	6	8	9

Figure 24. Image of total cost, S(i, j): after the third pass (top to bottom)

portion in our case turned out to be well described by the following equation,

$$S : B : D = 10 : 2 : 5 \tag{4.3.44}$$

In other words, when the others are fixed, the variation of the angle affects the cost at the magnitude of 1 to 2. Similarly the strength and the gray-tone distance solely affect the cost at the magnitude of 1 to 10 and 1 to 5, respectively. Based on the above strategies, the cost of a road pixel is set to zero and the cost of a non-line pixel to 1000. The cost for a non-road line pixel (i, j) is given by

$$C(i, j) = \frac{g(B_{ij})\ h(D_{ij})}{f(S_{ij})} \tag{4.3.45}$$

```
1   2   3
4   5   6   ------> 0°
7   8   9
    |
    |
    ↓
   90°
```

Figure 25. The angle direction

The functions g, h, f in Eq. (4.3.45) are defined as follows.

$$g(B) = \begin{cases} 5 & B < LB \\ \dfrac{5*B}{UB - LB} & LB \le B \le UB \\ 10 & B > UB \end{cases} \qquad (4.3.46)$$

where LB and UB are lower and upper limits of angle difference (see Figure 26).

$$h(D) = \begin{cases} 2 & D < LD \\ \dfrac{8*D}{UD - LD} & LD \le D \le UD \\ 10 & D > UD \end{cases} \qquad (4.3.47)$$

where LD and UD are lower and upper limits of gray-tone distance (see Figure 27).

$$f(S) = \begin{cases} 1 & S < LS \\ \dfrac{9*S}{US - LS} & LS \le S \le US \\ 10 & S > US \end{cases} \qquad (4.3.48)$$

where LS and US are lower and upper limits of line strength (see Figure 28).

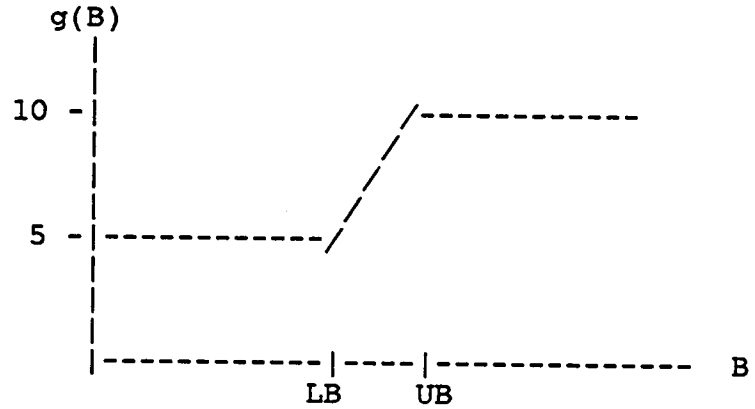


Figure 26. The angle difference function

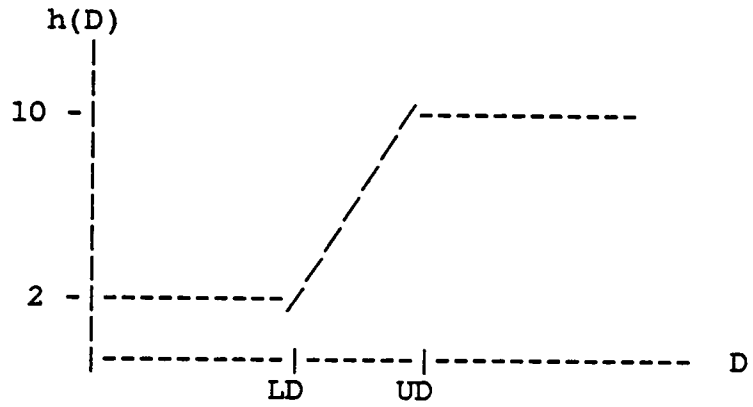


Figure 27. The gray-tone distance function

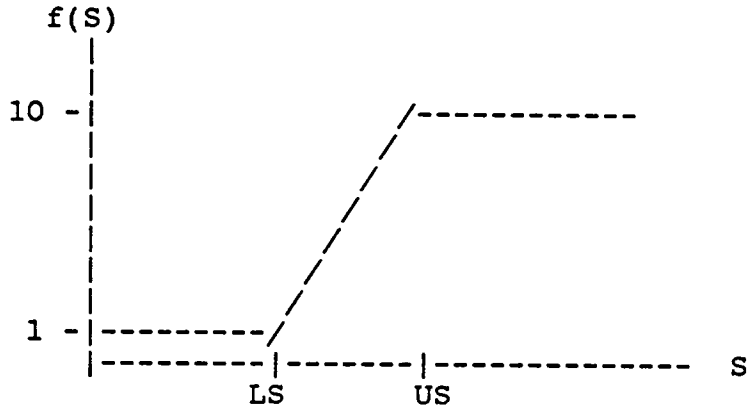


Figure 28. The strength function

CHAPTER 5 FINAL RESULTS

The procedure was first tested on a controlled data set. The data set is composed of a small noise-free image and seven images created by adding Gaussian noise with zero mean and various standard deviations to the noise-free image (see Figure 29). The pixel value of the road structure in the noise-free image was set to 75 and that of the background to 175. The resulting line images and network images are shown in Figures 30 and 31. The reconstructed network images were compared with a perfect network image in terms of RMS errors as shown in Figure 32.

For the test on natural images, a few SAR images were used as test images. Most of our experiments were carried out on 100 x 100-pixel subimages of them. Figures 33 through 36 show an original subimage, a facet based line image, a road image which is screened from the line image on the basis of road characteristics appearing in radar imagery and statistics of connected components, and a reconstructed network image as a final result. Figures 37 through 40 show a 512 x 512-pixel image and its line image, screened road image, and reconstucted road network image.

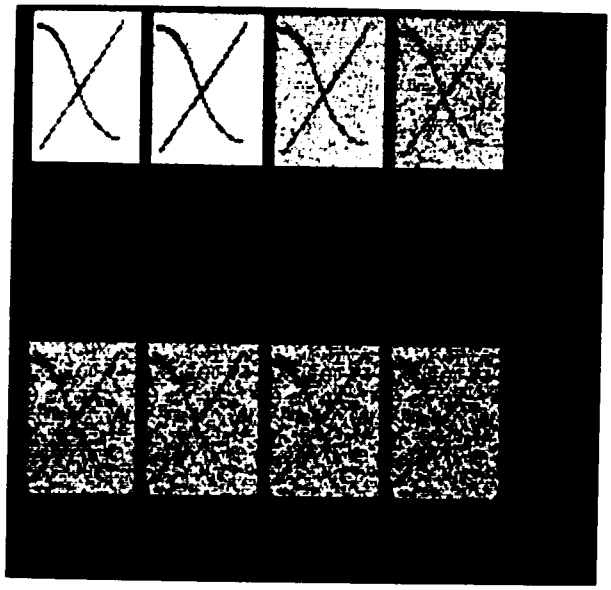


Figure 29. X images (controlled data): 50 x 64-pixel images. From left to right and top to bottom the standard deviations of added Gaussian noise are 0 (noise-free), 10, 20, 30, 40, 50, 60, and 70. (The means of the noise are zero)

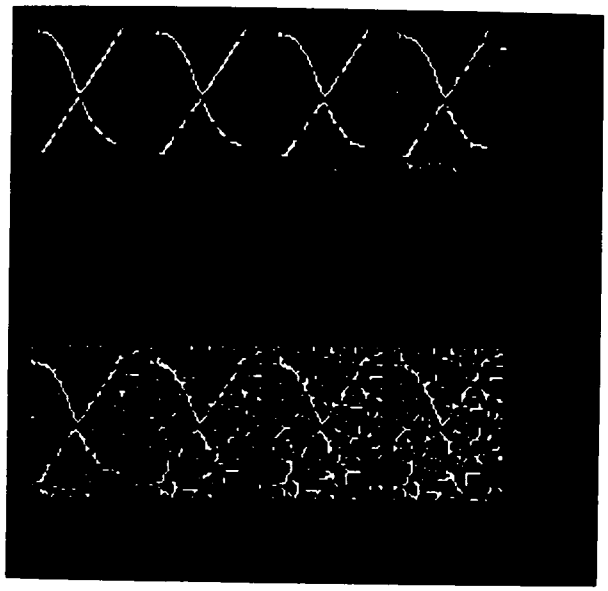


Figure 30. Line images: 50 x 64-pixel images

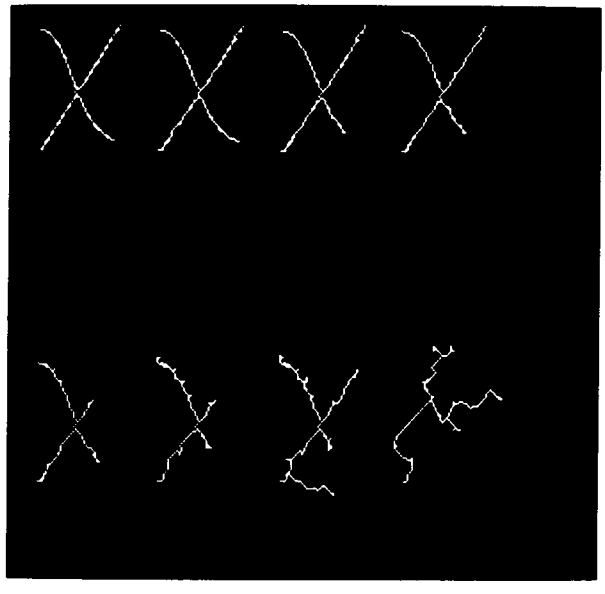


Figure 31. Reconstructed X image: 50 x 64-pixel image

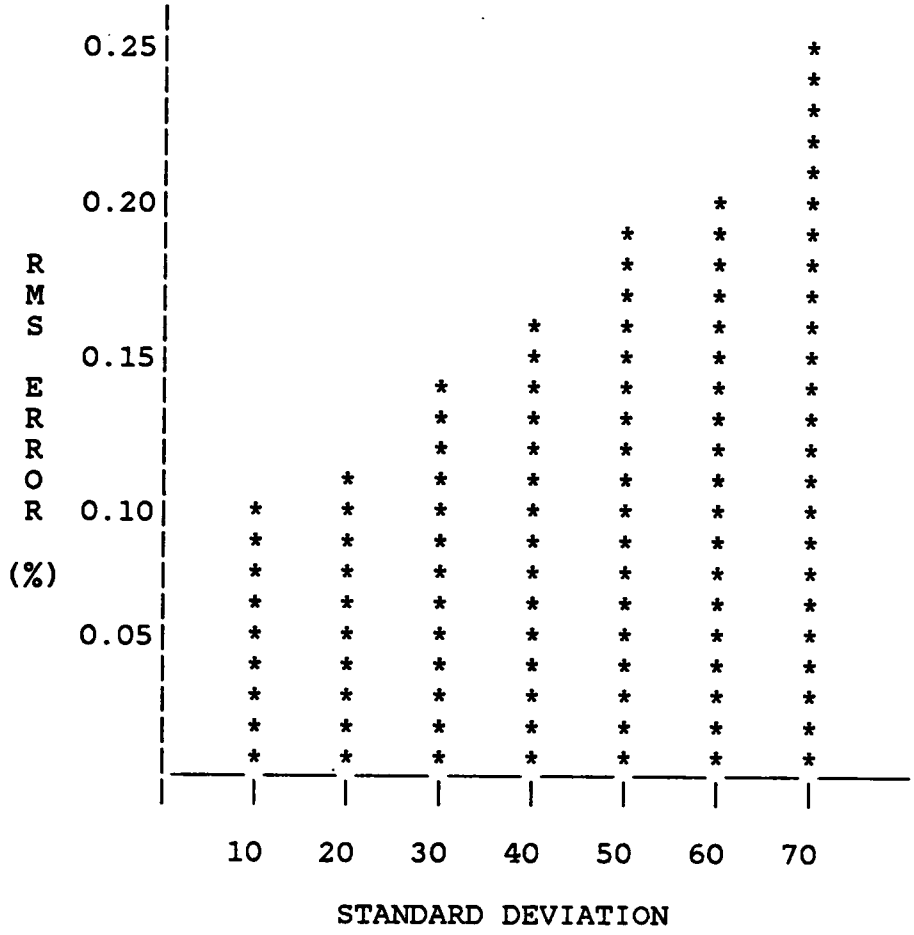


Figure 32. The performance of the procedure: As a function of the noise standard deviation

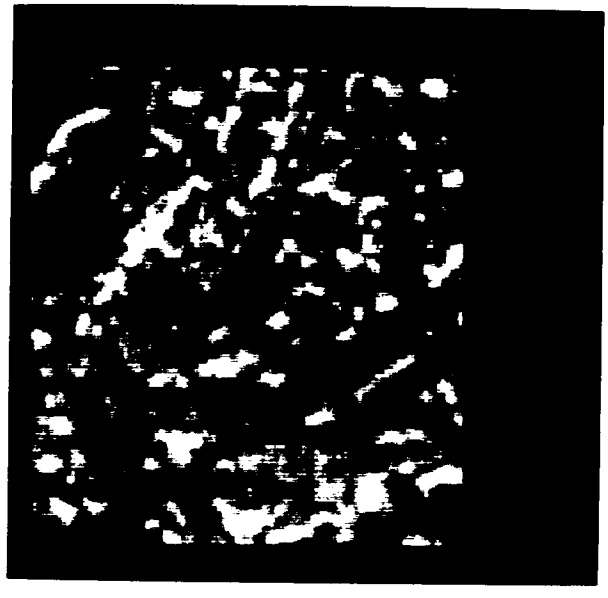


Figure 33. Subimage-1: 100 x 100-pixel image

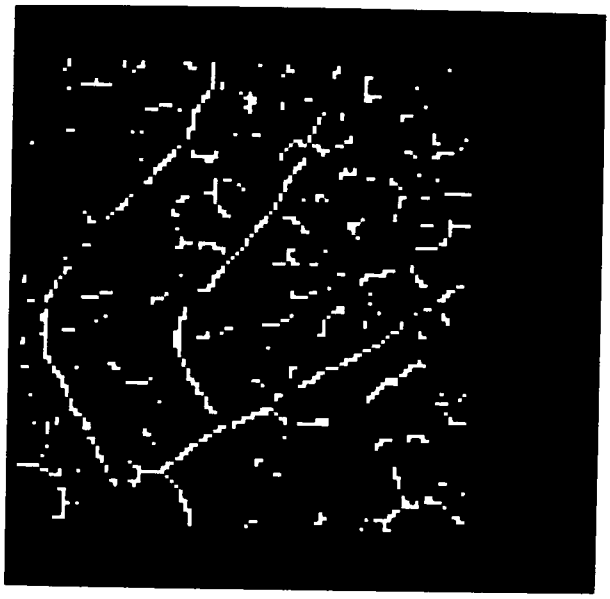


Figure 34. Line image: Radius = 1.0, Curvature = 750, Equiv. range = 1800 -- 2700 Width = 1 -- 4, Contrast = 100

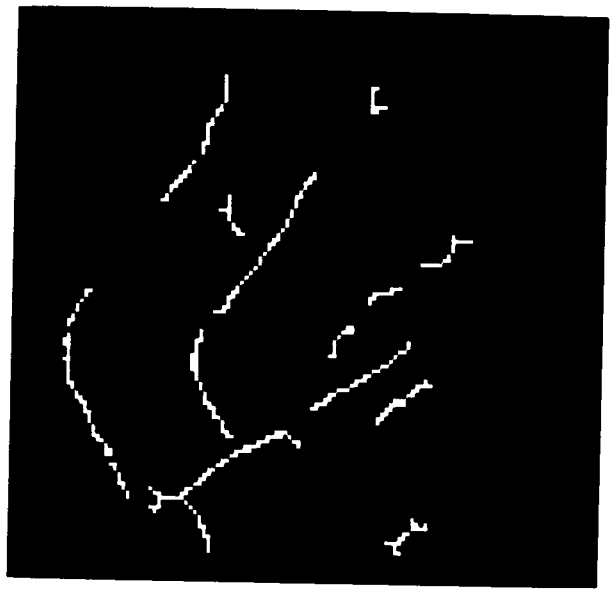


Figure 35. Screened road image: N = 15, MS = 1200, MB = 20, MG = 25,000

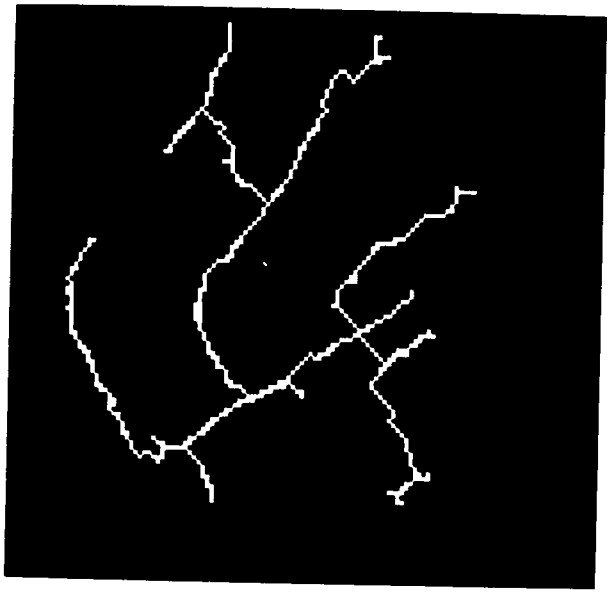


Figure 36. Reconstructed Network image

Figure 37. Elizabeth City area: 512 x 512-pixel image

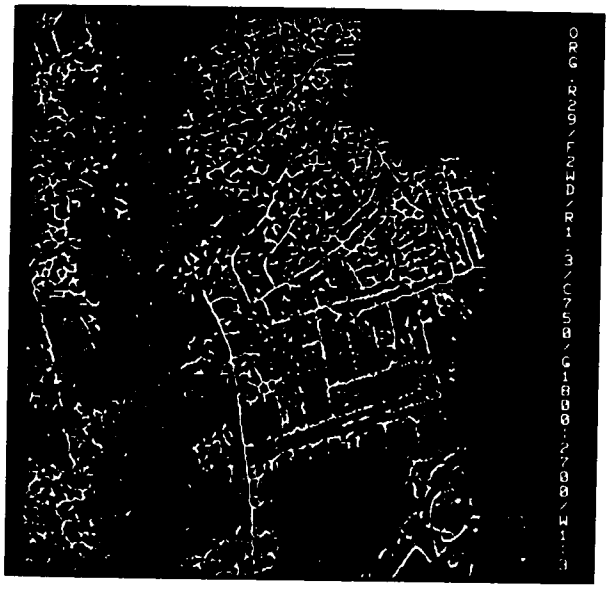


Figure 38. Line image: Radius = 1.3, Curvature = 750, Window size = 9 x 9, Equi. range = 1800 -- 2700, Width = 1 -- 4, Contrast = 50

Figure 39. Screened road image: N = 16, MS = 1250, MB = 15, MG = 25,000

Figure 40. Reconstructed road network image: There are still missing roads and falsely connected roads.

CHAPTER 6 RELATED WORK

There are not many published papers closely related to the work reported in this thesis. However, it is worth summarizing some related approaches to the problem of line detection and continuation, since road network detection involves both road segment detection and road connection techniques.

Fischler, et al. [7] integrated several road segment techniques. They defined operators classified either as type I operator that will rarely mistake artifacts as road segments but often miss true instances, or as type II operator that will measure parameters of all true instances but may falsely classify and parameterize non-instances. Type I operators produce the skeleton of a road network, which is then completed by an optimization algorithm, on the basis of a cost array evaluated by the results of type I and type II operators.

Bajcsy and Tavakoli [11] start with a strip detector which finds the connected road pixels (strip) by thresholding gray-tone values. The strips are further screened by thresholding according to their contrasts with adjacent regions, and their own widths and lengths. A road segment (piece) consists of two or more connected strips. Then road

segments are connected according to continuity and proximity criteria between adjacent disconnected road segments.

There are many papers regarding the continuation problem for local lines or edges (called global lines or edges). VanderBrug [12] pruned and enhanced local linear features iteratively, pixel by pixel, according to the direction and strength of adjacent line pixels. To connect the local line segments he represented them as a graph, and defined a merge merit function for selecting pairs of segments to merge. The function is based on the distance, angle alignment, smooth continuation, and strength of local line segment. He used a stage by stage relaxation method for merging line segments until there are no acceptable pairs to merge.

Ramer [14] used a tree search algorithm to find a global edge. He first finds edge and line elements called strokes. The strokes that align to form a curve segment are grouped into a streak by representing a stroke image as a directed graph and then searching the minimum cost path from a initial node to a goal node.

Martelli [15] viewed a global edge detection problem as a problem of finding an optimal path in a weighted graph. He used a heuristic tree search algorithm to find an optimal path on the basis of edge properties.

Montanari [16] viewed the line continuation problem as a multistage decision problem. A mutistage decision problem is best solved by dynamic programming. He was successful in the detection of one line with a fixed length from a noisy picture on the basis of figure of merit which reflects the curvature of a line. As in Martelli's work, he had to search for the initial line segment in an exhaustive manner.

Gold and Smith [17] [16] used a stage by stage dynamic programming algorithm to detect nearly straight lines in a noisy picture. They were able to reduce the large cpu time and storage space required (the inherent problem of the dynamic programming) by restricting the search area into small regions called angular bins.

Ehrich [18] constructs global low curvature lines by growing local line segment optimally a fixed number of pixels at a time using dynamic programming. One of his main contributions was a data organization that reduced redundant local line information so that large images could be processed quickly. His merit function used in the dynamic programming algorithm is based on the confidence of the presence of line segments, the curvature of one to another, and the smoothness of continuation.

In this thesis the approach to the line continuation problem is also dynamic programming. However, the problem is viewed differently, not as a multistage decision problem, but as a minimum cost path problem. Hence the processing time depends not on the length of a line but on the shape of a line (see Appendix A), and the required storage space is linearly proportional to the area of an image. Most of road segment detection techniques including all of the algorithms mentioned above are based on discrete line or edge detectors that have to use many directional and discrete masks to obtain local edge information. However, since the road segment detection scheme used in this thesis is based on a facet model, the required information is computed analytically. This is the main difference from earlier local feature detection techniques.

CHAPTER 7 CONCLUSIONS AND SUGGESTIONS FOR FURTHER WORK

This thesis began with the analysis of road characteristics appearing in radar imagery and proceeded to the construction of a road network detection procedure.

Chapter 2 discusses a noise removal algorithm using local gradient and statistics. By using this algorithm with a practical method of estimating a local noise variance, it was possible to filter a noisy radar image efficiently without contrast loss.

In Chapter 3, it was shown that the kernel of a facet model could be viewed as the function which can be used to compute a facet model according to its arguments such as basis function, the order of fit, index set, and an input digital image. Using this function, a bicubic facet model was obtained. It was then found that linear features could be analytically detected on the basis of the bicubic facet model. This analytical detection technique made it possible to detect and parameterize most of the attributes of linear features such as line orientation, strength, and width.

In Chapter 4, the problem of screening the detected linear features on a pixel by pixel basis was addressed. The al-

ternative was to screen them on a component by component basis. This method uses the statistics of each maximally connected set of pixels, called a component, and the road characteristics appearing in radar imagery. Using this method, it was possible to obtain a relatively small number of real road segments. In Section 4.3, it was shown that these road segments could be connected by dynamic programming. This algorithm, based on the "principle of optimality", gave a minimum cost path between two sets of road segments. The costs used in this algorithm were evaluated by the cost evaluation function which measured a merit for each pixel. The dynamic programming algorithm was applied iteratively as many times as the number of the paths required to connect all the road segments.

There are a few directions in which the work in this thesis could be continued. The most obvious is the method of applying the connection procedure. The method used in this thesis was to apply the procedure to the whole image. This required a relatively long processing time, and hence it was necessary to reduce the number of road segments as much as possible. Since the locations of the starting road segments were known, if the locations of the goal road segments could be correctly guessed using the global context, the search area and the processing time could be greatly reduced.

The cost evaluation function described in Section 4.3 could be made more sophisticated and accurate so that the connection procedure would connect road segments through non-existent paths less frequently.

Another possible improvement would be to selet a more suitable fitting surface. The surface used in this thesis was the bicubic polynomial, which was too general for our line dection purpose. By using a biquadratic or a constrained form of a bicubic polynomial, it may be possible to reduce the computational complexity. For instance, if the fitted surface was of the form,

$$f(r,c) = A\,(r\cos\alpha + c\sin\alpha)^3 + B\,(r\cos\alpha + c\sin\alpha)2 + C\,(r\cos\alpha + c\sin\alpha) + D \qquad (7.0.49)$$

the cross-section of the surface would be directly obtained simply by replacing $(r\cos\alpha + c\sin\alpha)$ with ρ, where α would be the direction normal to an assumed line as discussed in Section 3.3. It may require deeper mathematical insight as well as laborious research to find proper basis functions, but the considerably faster computation would be worth it.

APPENDIX A. THE PROOF OF THE REQUIRED NUMBER OF PASSES

Definition 1: If the row index for the pixels from one end to the other end along a path increases (or decreases) monotonically, the path is called 'i-monotonic' (or 'd-monotonic') or simply 'monotonic'.

Definition 2: Application of the Eq. (4.3.41) and Eq. (4.3.42) to a whole image either in a top-to-bottom manner or in a bottom-to-top manner as described in Section 4.3 is called a pass through the image.

Notation 1: S denotes the image of total cost for the path from a starting pixel A to each pixel and S(B) denotes the total cost for the path from A to B.

Notation 2: O denotes the image of the optimum total cost for the optimum path from a starting pixel A to each pixel and O(B) denotes the optimum cost for the optimum path from A to B.

Notation 3: S_i denotes the image generated after the ith pass through the image S.

Lemma 1: If there exists an unique optimum path from the starting pixel A to the goal pixel G and there are two pixels X and Y on the optimum path such that for every pixel x from A up to X along the path, $S(x) = O(x)$, and for every pixel g beyond X to G along the path, $S(g) >= O(g)$, and if the optimal subpath from X to Y is monotonic, then after a single pass through S, every pixel y from the starting pixel A up to Y along the path will have the optimal cost ($S_1(y) = O(y)$).

Proof: (By induction over the sequence number of pixels along the path) Assume that the optimal subpath is i-monotonic and its sequence is X (= y_0), y_1, y_2, ... y_{n-1}, Y (= y_n). Let $r(y_i)$ and $c(y_i)$ denote the row and the column index for the pixel y_i. Then the following inequalities hold for $i >= 1$.

$$r(y_{i-1}) <= r(y_i) <= r(y_{i-1}) + 1$$

$$c(y_{i-1}) - 1 <= c(y_i) <= c(y_{i-1}) + 1$$

(base case) Because $S(y) = O(y)$ and the above inequalities for $i = 1$ hold, it is clear that $S_1(y_1)$ becomes $O(y_1)$ during

the top-to-bottom pass of applying Eq.(4.3.41) and Eq.(4.3.42) to the image S.

(induction) Assume that $S_1(y_k)$ becomes $O(y_k)$ during the top-to-bottom pass. When applying Eq.(4.3.42) to y_{k+1}, y_k contains $O(y_k)$ since the path from X to Y is monotonic and the above inequalities hold for $i = k + 1$. Therefore, $S_1(y_{k+1})$ equals to $O(y_{k+1})$ after that.

Q.E.D.

Theorem 1: For an unique optimal path which is composed of n monotonic subpaths P_i ($i = 1, 2, \ldots, n$), it requires at least n passes, to assure the convergence, through the total cost image where the starting pixel is assigned the value of 0 and the others assigned an infinite number, initially.

Proof: (By induction over the number of passes) Let the pixel X_k be the pixel of monotonicity change between P_k and P_{k+1} Also let X_0 and X_n be the starting and the goal pixel. Then it suffices to show that for every pixel g from the starting pixel to the goal pixel along the optimum path, $S_n(g) = O(g)$.

(base case: after the first pass) Since $S(X_0) = O(X_0) = 0$, and P_1 is monotonic, for every pixel x_1 in P_1, $S_1(x_1) = O(x_1)$ by the Lemma 1. Now assume that there exists a pixel u, say in P_j such that $S_1(u) = O(u)$. It means then that there exists

another optimum path from X_0 to X_n which does not include the pixels $X_1, \ldots, X_{j-1}$. This contradicts the uniqueness stated in the antecedent. Hence, after the first pass only the pixels up to X_1 along the optimal path will have the optimal cost.

(induction) Assume that after kth pass, for every pixel x_k from the starting pixel up to X_k, $S_k(x_k) = O(x_k)$. Then, by the fact that P_{k+1} is monotonic and the $S_k(X_k) = O(X_k)$, we know that after a single pass through S_k, for every pixel x_{k+1} from the starting pixel up to X_{k+1}, $(S_k)_1(x_{k+1}) = S_{k+1}(x_{k+1}) = O(x_{k+1})$ [Lemma 1]. Furthermore, the other pixels beyond X_k up to the goal pixel will not have yet the optimal cost since otherwise, it would contradict the uniqueness in the similar way as above. Therefore, only after the (k + 1)th pass through the initial total cost image S, every pixel up to X_{k+1} along the path will have the optimal cost.

Q.E.D.

APPENDIX B. DOCUMENTATIONS, RUN-FILES, AND LISTINGS

This appendix contains the documentations, run-files, and program listings for the four commands which were used in the thesis. Those commands are FLINE (Facet LINE detector), ROAD (Road finder), COST (Cost evaluation), and MCPATH (Minimum Cost PATH). This appendix also includes the run-file, THESIS.RUN which creates all the subimages from the gray-tone image to the final road network image appearing in Chapter 5.

```
$ !
$ !  THESIS.RUN
$ !
$ !  This run file is to show the way the 100 x 100 gray-tone image,
$ !  SB.SIF, appeared in the thesis body is processed to get the final
$ !  road network image.
$ !
$ !
$ !  GET THE CUBIC FACET PARAMETERS
$ !
$    FACET2 SB.SIF > SB.F29 (W)
     9
     9
     3
     1.5

$ !
$ !  RUN THE LINE COMMAND TO GET A LINE IMAGE
$ !
$    FLINE SB.F29 > SB.LIN
     2
     1.3
     750
     1800
     2700
     1
     3
$ !
$ !  GET THE CONNECTED COMPONENT IMAGE OUT OF THE LINE IMAGE
$ !
$    CONCM SB.LIN > SB.CMP
     2
     1
     3
$ !
$ !  SCREEN THE CONNECTED COMPONENTS BASED ON THE ROAD CHARACTERISTICS
$ !
$    ROAD SB.LIN, SB.CMP, SB.SIF > SB.ROD
     10
     1250
     20
     25000
$ !
$ !  GET THE CONNECTED COMPONENT OF SCREENED IMAGE
$ !
$    CONCM SB.ROD > SB.RCM
     1
$ !
$ !  TO MAKE IT SURE THAT THE GENERATED COMPONENT IMAGE HAS LABELS
$ !  CONSECUTIVELY STARTING FROM 1
$ !
$    RELBL SB.RCM > SB.RLB (S)
```

```
      0
$ !
$ !   GET THE COST IMAGE
$ !
$     COST SB.LIN, SB.RLB, SB.SIF > SB.CST
      5
      30
      22500
      2500
      5000
      500
      3000
$ !
$ !   NOW, READY FOR LINKING THE SCREENED LINE SEGMENTS IN THE LINE
$ !   IMAGE
$ !
$     MCPATH SB.CST > SB.NET
      100
      1000
$ !
$ !   THE FINAL RESULT OF THE WHOLE PROCESS IS SB.NET
$ !
$     EXIT
```

*FLINE LINE FINDER

VERSION: A.02 DATE: 11-27-84 AUTHOR: J.J.KIM

ACTION: This command takes the central neighborhood cubic facet parameters for each pixel's central neighborhood and uses them to define a line in the direction of maximum magnitude of second directional derivative. There are two such directions which are 90 degrees apart. If in one of these directions a zero crossing of the first directional derivative occurs sufficiently close to the pixel's center a bright line or dark line is declared depending on the sign of the second directional derivative.
To avoid misclassifications due to noise or fitting artifacts additional requirements for a pixel to be labeled as a line are that: the gradient angle change taken in the direction extremizing second directional derivative must exceed 15 degrees; the curvature must exceed a certain threshold; the width must be within a certain range; and so the gray-tone level.

This command generates two numeric bands and two symbolic bands as follows:

(Band 1: Numeric) Strengh
(Band 2: Numeric) Direction
(Band 3: Symbolic) Line image
(Band 4: Symbolic) Width

```
SOURCE: Disk, input file name.

DESTINATION: Disk, output file name

QUESTIONS: (1) Enter the line type, Bright line:1, Dark line:2
               (d=2, 1 or 2)
           (2) Enter event radius (d=1.2, .1 to 2)
           (3) Enter curvature threshold (d=750. 0. to 100000.)
           (4) Enter gray-tone lower limit (d=1000, 0 to 65000)
           (5) Enter gray-tone upper limit (d=3000, 0 to 65000)
           (6) Enter width lower limit (d=1, 1 to 6)
           (7) Enter width upper limit (d=6, 1 to 6)

COMMAND STRING EXAMPLE:

    FLINE BAR.LIN < BAR.CUB

    Enter line type, bright:1,dark:2 (D=2, 1 to 2) 2
    Enter event radius (D=1.2, .1 to 2.) 1.2
    Enter curvature threshold (D=750.0, 0 to 10000.) 750.
    Enter gray-tone lower limit (d=1000, 0 to 65000) 1800
    Enter gray-tone upper limit (D=3000, 0 to 65000) 2700
    Enter width lower limit (D=1, 1 to 6) 1
    Enter width upper limit (D=1, 1 to 6) 1

    The image containing the coefficients of the cubic fit is BAR.CUB
    and the created  line  image is BAR.LIN.
    If a zero-crossing of the first directional derivative taken in the
    direction of the maximum magnitude of second directional
    derivative exists within a distance 1.2 of the pixel's center,
    the curvature is greater than 750, and its equivalent gray-tone is
    between 1800 and 2700, then a dark-line with one pixel width is
    declared at that pixel's position. Non-line pixels are marked with
    zeros.

REFERENCES
    GIPSY_DOC:RIDVAL.DOC, AUTHOR'S MASTER THESIS CHAPTER 3


$$!
$$! FLINE.RUN
$$!
$$!     MAKE A PARABOLA FACING UP
$$!
$       MKBAR LINE.IMG
        50
        V
        L
$       ARITHM LINE.SUB < LINE.IMG (S)
        25
```

```
        N
$       IMGPWR LINE.PWR < LINE.SUB
        2
        100
$ !
$$ !            GET THE CUBIC FACET PARAMETERS
$$ !
$        FACET2 LINE.CUB < LINE.PWR (W)
         9
         9
         3
         1.5
$ !
$$ !            MARK THE DARK LINES
$$ !
$        FLINE LINE.LIN < LINE.CUB
         2
         1.0
         1.0
         0
         10
         1
         6
$ !
$$ !            TO TEST THE RESULT WE WILL MAKE ALL DARK LINE PIXELS
$$ !            TAKE A 1 AND THEN COMPARE
$$ !
$         TRSLD LINE.TRS < LINE.PWR
          0
$ !
$$ !
$$ !            TO COMPARE WE WILL MAKE ALL DARK LINE PIXELS TAKE
$$ !            THE VALUE 2
$$ !
$       EXSIF
        OPEN LINE.TRS
        MID 18 0
        Y
        DONE
$       ARITHM LINE2.TRS < LINE.TRS (M)
        2
        N
$        EXSIF
         OPEN LINE2.TRS
         MID 18 1
         Y
         DONE
$        RMS TT < LINE2.TRS,LINE.LIN
$ !
$$ !            NOW TEST FOR THE BRIGHT LINES
$$ !
$$ !
$$ !            INVERT THE PARABOLA
$ !
```

```
$        INVIMG LINEM.IMG < LINE.PWR
$        ARITHM LINEINV.IMG < LINEM.IMG (A)
         1000
         N
$ !
$$ !             GET THE CUBIC FACET PARAMETERS
$$ !
$        FACET2 LINEINV.CUB < LINEINV.IMG (W)
         9
         9
         3
         1.5

$$ !
$$ !             MARK THE BRIGHT LINES
$$ !
$        FLINE LINEINV.LIN < LINEINV.CUB
         1
         1.0
         1.0
         6300
         6400
         1
         6
$ !
$        EXSIF
         OPEN LINE.TRS
         MID 18 1
         Y
         DONE
         RMS TT < LINEINV.LIN,LINE.TRS
$$ !
$       DELETE
        LINE.IMG
        LINE.SUB
        LINE.PWR
        LINE.CUB
        LINE.LIN
        LINE.TRS
        LINEM.IMG
        LINEINV.IMG
        LINEINV.CUB
        LINEINV.LIN
        LINE2.TRS

$!
$       EXIT
```

```
#--DFLINE        DRIVER FOR THE LINE        COMMAND                MID
#
```

```
#  IDENTIFICATION
#
#       TITLE           DFLINE
#       AUTHOR          JUNGWHAN JOHN KIM
#       VERSION         C.01
#       DATE            OCTOBER 28, 1985
#       LANGUAGE        RATFOR
#       SYSTEM          VAX-11-780
#
#  UPDATE
#
#       UPDATE
#       AUTHOR
#       DATE
#       VERSION
#       PURPOSE
#
#  PURPOSE
#
#       THE PURPOSE OF THIS COMMAND IS TO TAKE THE ELEVEN BAND OUTPUT
#       OF THE COMMAND FACET2 AND DECTECT THE LINES ON IT.
#
#  ENTRY POINT
#
#       DFLINE( WORK, ERRET )
#
#  ARGUMENT LISTING
#
#       WORK    INT.ARR     WORKING STORAGE ARRAY
#       ERRET   INT         ALTERNATE ERROR RETURN
#
#  INCLUDE FILES/COMMONS
#
#       MACA1     INCLUDE     GIPSY TOKEN DEFINTIONS INCLUDE
#                             FILE FOR A1 CHARACTERS.
#       GIPCOM    COMMON      GIPSY COMMAND LINE INFORMATION
#       ERROR     COMMON      GIPSY ERROR STACK INFORMATION
#
#  ROUTINES CALLED
#
#       RDKINL  OPEN A SIF AND FILL IN THE IDENT        (GIPSY ROUTINE)
#               BLOCK.
#       CLOSE   ROUTINE TO CLOSE A FILE.                (GIPSY ROUTINE)
#       OSALOC  ROUTINE TO CHECK FOR EXCEEDED ARRAY     (GIPSY ROUTINE)
#               SIZE.
#       NFLINE   THIS IS THE NUMBER CRUNCHER ROUTINE     (GIPSY ROUTINE)
#               FOR THE COMMAND LINE
#       PPOP    REMOVES THE PROGRAM NAME FROM THE       (GIPSY ROUTINE)
#               ERROR STACK.
#       PPUSH   PLACES THE PROGRAM NAME ONTO THE ERROR  (GIPSY ROUTINE)
#               STACK.
#
#  REMARKS
#
#
```

```
#
#*****************************************************************
#
#
#
      INCLUDE MACA1
      SUBROUTINE  DFLINE ( WORK, *)
#
#
      IMPLICIT INTEGER (A-Z)
#
      INCLUDE GIPCOM
      INCLUDE ERROR
#
      INTEGER WORK ( .ARB )
      INTEGER IDENT ( .IDLENGTH )
      REAL GRADTH,FRATIO,RADIUS,CRVTHR,ANGTHR
      REAL DGRAD,GMAX,DMAX,DRADIUS,DCURVE,DANG,DRV
#
      EQUIVALENCE  (NPPL, IDENT(.IDNPPL)),
                   (NLINS, IDENT(.IDNLINS))
      EQUIVALENCE  (NCOLS, IDENT(.IDNCOLS)),
                   (NROWS, IDENT(.IDNROWS))
      EQUIVALENCE  (NTBND, IDENT(.IDNBNDS))
      EQUIVALENCE  (MODE, IDENT(.IDMODE))
#
      COMMON /RDV/ROW,COL
#
      DATA DGRAD/5./
      DATA DCURVE/750.0/
      DATA DANG/15./
      DATA DFRAT/.25/
      DATA DRADIUS/1.2/
      DATA ROW/-1/
#
      CALL PPUSH ("DFLINE ")
#
      GRADTH  = DGRAD
      ANGTHR  = DANG
      FRATIO  = DFRAT
#
#                          SET UP INPUT FILE
#
      CALL RDKINL( FDI1, IDENT, .READONLY, IEV, %9998 )
      CALL CLOSE(FDI1)
#
#                          CHECK THE INPUT FILE
#
      IF( MODE ¬= 0 & MODE ¬= .INTMODE ) GO TO 9000
      IF (NPPL ¬= NCOLS ! NROWS ¬= 1)  GO TO 9010
      IF (NTBND ¬= 11) GO TO 9020
#
#                          SET UP BUFFERS
#
      NXT = 1
```

```
      IBUFF = GETWP (NXT,MODE,NPPL*10)
      OBUFF = GETWP (NXT,MODE,NPPL)
      SBUFF=GETWP(NXT,MODE,NPPL)
      ABUFF=GETWP(NXT,MODE,NPPL)
      WBUFF=GETWP(NXT,MODE,NPPL)
#
#                          NOW ALLOCATE THAT SPACE
#
      IF( .OK ¬= OSALOC(NXT)) GO TO 9030
#
#                          GET  PARAMETERS
#
      ANGTHR = ANGTHR * .017453
      CALL RNGETI ('Enter line type, Bright:1,Dark:2.',1,2,2,BW,IEV,%9998)
      CALL RNGETR ('Enter event radius.',.1,2.,DRADIUS,RADIUS,IEV,%9998)
      CALL RNGETR('Curvature threshold.',0.,100000.,DCURVE,
                  CRVTHR,IEV,%9998)
      CALL RNGETI('Enter gray-tone lower limit.',0,65000,1000,LG,
                                                  IEV,%9998)
      CALL RNGETI('Enter gray-tone upper limit.',LG,65000,LG,UG,
                                                  IEV,%9998)
      CALL RNGETI('Enter width lower limit.',1,7,1,LW,IEV,%9998)
      CALL RNGETI('Enter width upper limit.',LW,7,6,UW,IEV,%9998)
#
      CALL COMTIN(%9998)
#
444   CONTINUE
#
#                          CALL THE NUMBER CRUNCHER
#
      CALL NFLINE (FD11, FD01, WORK(IBUFF), WORK(OBUFF), NPPL,WORK(SBUFF),
       WORK(ABUFF),WORK(WBUFF),RADIUS,CRVTHR, ANGTHR, GRADTH,
       LG,UG,LW,UW,BW,IEV,%9998)
#
      CALL PPOP
      RETURN
#
#                          ABNORMAL CONDITIONS
#
9000  CONTINUE
#
#                          NOT AN INTEGER FILE
#
      IEV = -2012
      GO TO 9998
#
9010  CONTINUE
#
#                          NOT IN LINE FORMAT
#
      IEV = -5001
      GO TO 9998
#
9020  CONTINUE
```

```
#
#                                NO NUMERIC BANDS
#
      IEV = -5018
      GO TO 9998
#
 9030 CONTINUE
#
#                                NOT ENOUGH WORK SPACE
#
      IEV = -5010
#
 9998 CONTINUE
#
#                                ERROR IN SUBPROGRAM
#
      CALL CLOSE ( FDI1 )
      CALL CLOSE ( FDO1 )
#
#                                ABNORMAL RETURN
#
 9999 CONTINUE
      RETURN 1
      END


#--NFLINE        NUMBER CRUNCHER FOR THE LINE        COMMAND        MID
#
# IDENTIFICATION
#
#       TITLE           NFLINE
#       AUTHOR          JUNGWHAN JOHN KIM
#       VERSION         C.01
#       DATE            OCTOBER 28, 1984
#       LANGUAGE        RATFOR
#       SYSTEM          VAX-11-780
#
#  UPDATE
#
#       UPDATE
#       AUTHOR
#       DATE
#       VERSION
#       PURPOSE
#
#
#
#  PURPOSE
#
#       THIS ROUTINE IS THE NUMBER CRUNCHER FOR THE COMMAND LINE
#
```

```
#  ENTRY POINT
#
#      NFLINE ( FDI, FDO, IIN, IOUT, NPPL, SBUFF, ABUFF, WBUFF, RADIUS, CRVTHR,
#             ANGTHR,GRADTH,LG,UG,LW,UW,BW,IEV, ERRET)
#
#  ARGUMENT LISTING
#
#      FDI      CHR.ARR      INPUT FILE DESCRIPTOR
#      FDO      CHR.ARR      OUTPUT FILE DESCRIPTOR
#      IIN      INT.ARR      INPUT BUFFER ARRAY
#      IOUT     INT.ARR      OUTPUT BUFFER ARRAY
#      NPPL     INT          NPPL OF INPUT FILE
#      SBUFF    INT.ARR      ANGLE OUTPUT BUFFER ARRAY
#      ABUFF    INT.ARR      CURVATURE DIRECTION OUTPUT BUFFER ARRAY
#      WBUFF    INT.ARR      LINE WIDTH BUFFER ARRAY
#      RADIUS   REAL         DISTANCE OF RIDGE OR VALLEY TO CENTER
#                            OF PIXEL
#      CRVTHR   REAL         CURVATURE THRESHOLD WHICH MUST BE EXCEEDED
#                            IN ORDER FOR CURVATURE TO BE CONSIDERED
#                            NON-ZERO.
#      ANGTHR   REAL         GRADIENT ANGLE DIFFERENCE THRESHOLD
#      GRADTH   REAL         GRADIENT THRESHOLD
#      LG       INT          GRAY-TONE LOWER LIMIT FOR A LINE
#      UG       INT          GRAY-TONE UPPER LIMIT FOR A LINE
#      LW       INT          WIDTH LOWER LIMIT IN THE NUMBER OF PIXELS
#                            FOR A LINE
#      UW       INT          WIDTH UPPER LIMIT IN THE NUMBER OF PIXELS
#                            FOR A LINE
#      BW       INT          TYPE OF LINE (BLACK OR WHITE)
#      IEV      INT          INTEGER EVENT VARIBLE
#      ERRET    INT          ALTERNATE ERROR RETURN
#
#  INCLUDE FILES/COMMONS
#
#      MACA1      INCLUDE      GIPSY TOKEN DEFINTIONS INCLUDE
#                              FILE FOR A1 CHARACTERS.
#
#  ROUTINES CALLED
#
#      PPUSH    PLACES THE PROGRAM NAME ONTO THE ERROR  (GIPSY ROUTINE)
#               STACK.
#      CPYIDR   ROUTINE TO OPEN THE INPUT FILE AND      (GIPSY ROUTINE)
#               COPY THE DESCRIPTOR RECORDS.
#      DSCNAM   WRITES SUBROUTINE NAME INTO THE         (GIPSY ROUTINE)
#               DESCRIPTOR RECORDS.
#      PDSCR    WRITES A REAL NUMBER TO THE             (GIPSY ROUTINE)
#               DESCRIPTOR RECORDS.
#      COPYDS   COPY THE DESCRIPTOR RECORDS FROM TEMP   (GIPSY ROUTINE)
#               FILE TO SIF & OPEN OUTPUT FILE.
#      GDSCAX   GETS VALUES FROM THE DESCRIPTOR RECORDS (GIPSY ROUTINE)
#      RREAD    READ A RECORD FROM A SIF.               (GIPSY ROUTINE)
#      RWRITE   WRITE A RECORD TO A SIF.                (GIPSY ROUTINE)
#      CLOSE    ROUTINE TO CLOSE A FILE.                (GIPSY ROUTINE)
#      PPOP     REMOVES THE PROGRAM NAME FROM THE       (GIPSY ROUTINE)
#               ERROR STACK.
```

```
#
#
#*****************************************************************
#
#
      INCLUDE MACA1
      SUBROUTINE NFLINE  ( FDI, FDO, IIN, IOUT, NPPL, SBUFF, ABUFF,WBUFF,
                         RADIUS,CRVTHR,ANGTHR,GRADTH,
                         LG,UG,LW,UW,BW,IEV,*)
#
      IMPLICIT INTEGER ( A - Z )
      INCLUDE DSCAX
      CHARACTER FDI ( .FDLENGTH )
      CHARACTER FDO ( .FDLENGTH )
#
      INTEGER IDENT ( .IDLENGTH ), JDENT ( .IDLENGTH )
      INTEGER IIN(NPPL,10),IOUT(NPPL),SBUFF(NPPL),ABUFF(NPPL),WBUFF(NPPL)
      INTEGER NUMDR
      LOGICAL INITX
      REAL SIZE
      REAL CRVTHR,ANGTHR,THR,RADIUS,FLOAT,FCTDSC(12),SCALE(12)
      REAL GRADTH,STR,DIR,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10
#
      EQUIVALENCE  ( NLIN, IDENT(.IDNLINS) ), ( MODE, IDENT(.IDMODE) )
#
      COMMON /RDV/ROW,COL
#
      CALL PPUSH ( "NFLINE " )
      NUMDR = 1
#
#
#                    OPEN INPUT FILES
#
#                    COPY DESCRIPTOR RECORDS FROM INPUT
#                    IMAGE, ROUTINE NAME, AND PARAMETERS
#                    TO TEMPORARY SEQUENTIAL FILE.
#
      CALL CPYIDR ( FDI, IDENT, .OPNTMP, IEV, %9998 )
#
      CALL DSCNAM ( 'NFLINE', IEV, %9998 )
#
      CALL PDSCR ( 'RADIUS .', RADIUS, IEV, %9998 )
      CALL PDSCR ( 'CRVTHR .',CRVTHR,IEV,%9998)
      CALL PDSCR ( 'ANGTHR .',DEPTHR,IEV,%9998)
      CALL PDSCR ( 'GRADTH .',GRADTH,IEV,%9998)
      CALL PDSCR ( 'LG .',LG,IEV,%9998)
      CALL PDSCR ( 'UG .',UG,IEV,%9998)
      CALL PDSCR ( 'LW .',LW,IEV,%9998)
      CALL PDSCR ( 'UG .',UG,IEV,%9998)
#
#                    OPEN OUTPUT IMAGE AND COPY TO IT THE
#                    DESCRIPTOR RECORDS FROM TEMPORARY FILE
#
      DO I = 1, .IDLENGTH
        JDENT ( I ) = 0
```

```
#
        IBAND = 3
        JDENT ( .IDNPPL ) = NPPL
        JDENT ( .IDNLINS ) = NLIN
        JDENT ( .IDNCOLS ) = NPPL
        JDENT ( .IDNROWS ) = 1
        JDENT ( .IDNBNDS ) = 4
        JDENT ( .IDNSBNDS ) = 2
        JDENT( .IDMODE ) = 0
        JDENT(.IDMIN)=-999999
        JDENT(.IDMAX)=999999
#
#
        CALL COPYDS ( FDO, JDENT, IEV, %9998 )
#
        CALL GDSCAX(FDI,.CAXSLOPEFAC,FCTDSC,IDENT,FCTLNE,NUMDR,IEV,%9998)
#
        IF(FCTLNE¬=13) GO TO 9000
#
        DO I=3,12
        $(
        IF(FCTDSC(I)<=0.) GO TO 9000
        SCALE(I)=1./FCTDSC(I)
        $)
        SIZE=(FCTDSC(1)-1.)*.5
#
#                    ACTUAL NUMBER CRUNCHING
#
         DO I = 1,NLIN
             $(
             DO J = 1,10
                 CALL RREAD( FDI, IIN(1,J), J, I, IDENT,
                             .WAIT, IEV, %9998 )
#
             DO K = 1,NPPL
                 $(
                 R1 = FLOAT( IIN(K,1) )* SCALE(3)
                 R2 = FLOAT( IIN(K,2) )* SCALE(4)
                 R3 = FLOAT( IIN(K,3) )* SCALE(5)
                 R4 = FLOAT( IIN(K,4) )* SCALE(6)
                 R5 = FLOAT( IIN(K,5) )* SCALE(7)
                 R6 = FLOAT( IIN(K,6) )* SCALE(8)
                 R7 = FLOAT( IIN(K,7) )* SCALE(9)
                 R8 = FLOAT( IIN(K,8) )* SCALE(10)
                 R9 = FLOAT( IIN(K,9) )* SCALE(11)
                 R10 = FLOAT( IIN(K,10) )* SCALE(12)
                 IOUT(K) = XFLINE( BW,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,RADIUS,
                 CRVTHR,ANGTHR,GRADTH,STR,DIR,WTH,
                 LG,UG,SIZE)
#
#
#
#
                IF (BW == 2)
                   IF (DIR > -90.0) DIR = DIR + 90.0
```

```
                ELSE              DIR = DIR + 270.0
            IF (IOUT(K) == BW)
                $(
                SBUFF(K) = STR
                ABUFF(K) = DIR
                WBUFF(K) = WTH
                $)
            ELSE
                $(
                SBUFF(K) = 0
                ABUFF(K) = 0
                WBUFF(K) = 0
                $)
            IF ((WTH < LW) | (WTH > UW))
               IOUT(K) = 0
             $)
        CALL RWRITE(FDO,SBUFF,1,I,JDENT,.WAIT,IEV,%9998)
        CALL RWRITE(FDO,ABUFF,2,I,JDENT,.WAIT,IEV,%9998)
        CALL RWRITE(FDO, IOUT,3,I,JDENT,.WAIT,IEV,%9998)
        CALL RWRITE(FDO,WBUFF,4,I,JDENT,.WAIT,IEV,%9998)
        $)
    CALL CLOSE (FDO)
#
#
    CALL CLOSE ( FDI )
#
    CALL PPOP
    RETURN
#
#                    ABNORMAL CONDITIONS
#
9000 CONTINUE
#
#                         BAD VALUE IN DESCRIPTOR RECORD
#
    IEV=-5037
 9998 CONTINUE
#
#                    READ OR WRITE ERROR
#
    CALL CLOSE ( FDI )
    CALL CLOSE ( FDO )
#
    RETURN 1
#
    END


#--XFLINE     LINE DETECTOR BASED ON 0-CROSSINGS OF 1ST DERIVATIVE  MID
#
#  IDENTIFICATION
```

```
#
#       TITLE           XFLINE
#       AUTHOR          JUNGWHAN JOHN KIM
#       VERSION         C.01
#       DATE            OCTOBER 28,1984
#       LANGUAGE        RATFOR
#       SYSTEM          VAX-11 780
#
#  UPDATE
#
#       UPDATE
#       AUTHOR
#       DATE
#       VERSION
#       PURPOSE
#
#  PURPOSE
#
#       THIS PROCEDURE DETECTS LINES;
#       ZERO CROSSINGS OF THE FIRST DERIVATIVE ARE
#       DETECTED IN DIRECTIONS WHICH EXTREMIZE THE SECOND
#       DIRECTIONAL DERIVATIVE.
#
#  ENTRY POINT
#
#       XFLINE ( BW, K1, K2, K3, K4, K5, K6, K7, K8, K9, K10, RADIUS, CRVTHR
#                ANGTHR,GRADTH,STR,DIR,WTH,LG,UG,SIZE)
#
#  ARGUMENT LISTING
#
#       K1      REAL        CONSTANT
#       K2      REAL        COEFFICIENT OF R
#       K3      REAL        COEFFICIENT OF C
#       K4      REAL        COEFFICIENT OF R**2
#       K5      REAL        COEFFICIENT OF R*C
#       K6      REAL        COEFFICIENT OF C**2
#       K7      REAL        COEFFICIENT OF R**3
#       K8      REAL        COEFFICIENT OF R**2*C
#       K9      REAL        COEFFICIENT OF R*C**2
#       K10     REAL        COEFFICIENT OF C**3
#       RADIUS  REAL        RADIUS WITHIN WHICH SECOND DIRECTIONAL
#                           DERIVATIVE MUST TAKE A ZERO IN THE
#                           DIRECTION OF THE GRADIENT
#       CRVTHR  REAL        THRESHOLD CURVATURE
#                           MUST EXCEED TO COUNT AS NON-ZERO.
#       ANGTHR  REAL        THRESHOLD GRADIENT ANGLE CHANGE TAKEN IN
#                           DIRECTION EXTREMIZING SECOND DIRECTIONAL
#                           DERIVATIVE MUST EXCEED TO COUNT AS NON-ZERO
#       GRADTH  REAL        GRADIENT THRESHOLD
#       STR     REAL        STRENGTH OF A LINE; ABSOLUTE DEPTH OR HEIGHT
#       DIR     REAL        DIRECTION OF A LINE
#       LG      INT         LOWER LIMIT OF GRAY-TONE RANGE CONSIDERED
#                           AS HAVING LINES
#       UG      INT         UPPER LIMIT OF GRAY-TONE RANGE
#       WTH     INT         WIDTH OF A LINE IN THE NUMBER OF PIXELS
```

```
#       SIZE    REAL        NEIGHBORHOOD RIGHT END POINT
#
#  ROUTINES CALLED
#
#       ATAN2   ARC TANGENT                                 (SYSTEM ROUTINE)
#       ZERCRO  FIRST DERIVATIVE ZERO CROSSING              (USER)
#       ANGLE   GRADIENT ANGLE DIFFERENCE                   (USER)
#
#
#*******************************************************************
#
#
        INCLUDE MACA1
        FUNCTION XFLINE (BW,K1,K2,K3,K4,K5,K6,K7,K8,K9,K10,RADIUS,CRVTHR,
            ANGTHR,GRADTH,STR,DIR,WTH,LG,UG,SIZE)
#
        INTEGER XFLINE, ZERCRO, BW
        INTEGER RCLASS,CLASS,CLASSA, CLASSB,LG,UG,WTH
        INTEGER RIDGE, VALLEY, JJ
        REAL K1,K2,K3,K4,K5,K6,K7,K8,K9,K10,RADIUS,CRVTHR,ANGTHR,GRADTH
        REAL SLRAT,STR, DIR, ROOTAS,ROOTAL,ROOTBS,ROOTBL
        REAL SIZE,EPS,ABS,GR(7),FLOAT
        REAL RVTHET,THETAA,THETAB,ATAN2,D,VALUE,SQRT
        REAL GRAD,CURVEA,CURVEB
        REAL RADDEG,CRVEAM,CRVEBM
        REAL AMN,AMX,CS,SN,A,B,ANGLE
        REAL T1,T2,T3,DER1,DER2,TA1,TA2,TA3,TB1,TB2,TB3,DEPTHA,DEPTHB
        REAL SNA,CSA,SNB,CSB,DEPTHX
#
#
        INCLUDE TTCOM
        COMMON/ROOT/ROOTL,ROOTS,DEP
#
        DATA RIDGE/1/, VALLEY/2/
        DATA EPS/1.E-3/
        DATA SLRAT/.333/
        DATA RADDEG/57.29577951/
#
#                                   IN A 5 PIXEL PERFECT EDGE
#                                   THE EXTREMA OCCUR AT +-SQRT(7/3)
#                                   THE DISTANCE BETWEEN EXTREMA
#                                   NORMALIZED BY THE NEIGHBORHOOD SIZE
#                                   IS 2.*SQRT(7/3)/5.
#                                   DSTNRM IS THE INVERSE SQUARE OF THIS.
#                                   DSTNRM=25/(4*7/3)
        DATA DSTNRM,DSMALL,DLARGE/2.678571,.64,1.5625/
#
#                                   FIRST AND SECOND DERIVATIVES OF
#                                   T0+T1*X+T2*X**2+T3*X**3
#
        DER1(X,T1,T2,T3)=(3.*T3*X+2.*T2)*X+T1
        DER2(X,T2,T3)=6.*T3*X+2.*T2
        GRAY(X,K1,T1,T2,T3)=K1+(T1+(T2+T3*X)*X)*X
#
```

```
#                                CUT THE BIVARIATE CUBIC
#                                K1+K2*R+K3*C+K4*R**2+K5*R*C+K6*C**2
#                                +K7*R**3+K8*R**2*C+K9*R*C**2+K10*C**3
#                                BY A VERTICAL PLANE IN DIRECTION
#                                (SN,CS).  THE RESULTING CURVE IS
#                                K1+T1*X+T2*X**2+T3*X**3 WHERE
#                                SN*X=R AND CS*X=C
#
      T3(SN,CS) =  (K7*SN+K8*CS)*SN*SN + (K9*SN+K10*CS)*CS*CS
      T2(SN,CS) =  (K4*SN+K5*CS)*SN +K6*CS*CS
      T1(SN,CS) = K2*SN+K3*CS
#
#                                SECOND PARTIAL DERIVATIVES OF
#                                BIVARIATE CUBIC EVALUATED AT
#                                (RO,CO).
#
      HRR(RO,CO)   =2.*K4+6.*K7*RO+2.*K8*CO
      HRC(RO,CO)   =K5+2.*K8*RO+2.*K9*CO
      HCC(RO,CO)   =2.*K6+2.*K9*RO+6.*K10*CO
#
#                                GRADIENT DIRECTION CHANGE
#                                TAKEN IN A DIRECTION (SN,CS)
#                                AND MULTIPLIED BY GRADIENT MAGNITUDE
#
      ANGLEG(SN,CS,RO,CO)= ABS(-CE*HRR(RO,CO)*SN-CE*HRC(RO,CO)*CS+
                           +SE*HRC(RO,CO)*SN+SE*HCC(RO,CO)*CS)
#
#                                FIRST PARTIALS EVALUATED AT (RO,CO)
#
      RDER(RO,CO)= K2+(2.*K4+2.*K8*CO+3.*K7*RO)*RO+(K5+K9*CO)*CO
      CDER(RO,CO)= K3+(2.*K6+2.*K9*RO+3.*K10*CO)*CO+(K5+K8*RO)*RO
#
      CALL PPUSH("XFLINE")
#
#                           DETERMINE FIRST ANGLE WHICH
#                           EXTREMIZES SECOND DIRECTIONAL DERIVATIVE
#
      IF(ABS(K5) > EPS)
         $(
         TH=(K6-K4)/K5
         T=1./(ABS(TH)+SQRT(1+TH*TH))
         IF(TH < 0.) T=-T
         $)
      ELSE T=0.
#
      CSA=1./SQRT(1+T*T)
      SNA=T*CSA
#
      CSB=-SNA
      SNB=CSA
#
      GRAD = SQRT(K2*K2+K3*K3)
#
      XINFLE = -999.
      DIR  = 360.
```

```
      ANGLEA = 0
      ANGLEB = 0
      CLASS  = 0
      CLASSA = 0
      CLASSB = 0
      XFLINE = 0
      DEPTHX = 0.
      DEPTHA = 0.
      DEPTHB = 0.
      STR=0.
      WIDTH = 0
      WIDTHA=0
      WIDTHB=0
      WTH = 0
      THETAA = 0.
      THETAB = 0.
#
#
#     RVFLAG=.TRUE.
#
      TA1=T1(SNA,CSA)
      TA2=T2(SNA,CSA)
      TA3=T3(SNA,CSA)
#
#                                DETERMINE THE LOCATIONS OF THE
#                                ZERO-CROSSINGS
#
      CLASS=ZERCRO(TA1,TA2,TA3,RADIUS,SIZE,DEPTHA)
      DEPTHX=DEPTHA
      ROOTAS=ROOTS
      ROOTAL=ROOTL
#
#                                GET THE WIDTH OF A LINE IF IT IS
#                                A LINE PIXEL
#
      GR(4) = GRAY(ROOTS,K1,TA1,TA2,TA3)
      IF (((GR(4) <= FLOAT(UG))&(GR(4) >= FLOAT(LG)))& (CLASS == BW))
         $(
         WIDTH = 0
         DO JJ = 1, 7
            $(
            GR(JJ) = GRAY(ROOTS+FLOAT(JJ-4), K1, TA1, TA2, TA3)
            IF ((BW == 2) & (GR(JJ) <= FLOAT(UG)))
               WIDTH = WIDTH + 1
            ELSE IF ((BW == 1) & (GR(JJ) >= FLOAT(LG)))
               WIDTH = WIDTH + 1
            ELSE
               WIDTH = WIDTH + 0
            $)

         $)
      WIDTHA = WIDTH
      IF(CLASS==0)                X=0
      ELSE                        X=ROOTAS
#
```

```
      CURVEA=DER2(X,TA2,TA3)
      CRVEAM = ABS(CURVEA)
#
#                                IS THE CURVATURE HIGH ENOUGH
#
      IF(CRVEAM < CRVTHR)GO TO 3000
      IF(CLASS==0)GO TO 3000
#                               CALCULATE THE RATE OF CHANGE OF
#                               GRADIENT ANGLE DIRECTION TAKEN
#                               IN THE DIRECTIN OF (SNA,CSA)
#
#
      RO = ROOTAS*SNA
      CO = ROOTAS*CSA
      SE = RDER(RO,CO)
      CE = CDER(RO,CO)
      GRADA = SQRT(SE*SE+CE*CE)
#
      IF (GRADA > GRADTH)
        $(
        SE = SE/GRADA
        CE = CE/GRADA
        $)
      ELSE
        $(
        SE = -CSA
        CE = SNA
        $)
#
#
     GRDCHA = ANGLEG(SNA,CSA,RO,CO)
     IF (GRDCHA >= 1000.*GRAD) ANGLEA = 1000.
     ELSE                      ANGLEA = GRDCHA/GRAD
#
#
#                              CHECK THE DIFFERENCE IN ANGLES
#                              JUST ON EITHER SIDE OF THE RIDGE
#                              VALLEY CANDIDATE
#
     IF (ANGLEA > ANGTHR)
        CLASSA = CLASS
     IF ((CLASSA == RIDGE & SNA < 0.) !
         (CLASSA == VALLEY & SNA >= 0.))
       $(
       IF (SNA == 0. ) SNA = EPS
       SNA = -SNA
       CSA = -CSA
       $)
#
      THETAA=RADDEG*ATAN2(SNA,CSA)
      RVTHET = THETAA
      WTH = WIDTHA
3000  CONTINUE
#
#
```

```
      TB1=T1(SNB,CSB)
      TB2=T2(SNB,CSB)
      TB3=T3(SNB,CSB)
#
#                                DETERMINE THE LOCATIONS OF THE
#                                ZERO-CROSSINGS
#
      WIDTH = 0
      CLASS=ZERCRO(TB1,TB2,TB3,RADIUS,SIZE,DEPTHB)
      ROOTBS=ROOTS
      ROOTBL=ROOTL
#
#                                GET THE WIDTH OF A LINE IF IT IS
#                                A LINE PIXEL
#
      GR(4) = GRAY(ROOTS,K1,TB1,TB2,TB3)
      IF (((GR(4) <= FLOAT(UG))&(GR(4) >= FLOAT(LG)))& (CLASS == BW))
         $(
         WIDTH = 0
         DO JJ = 1, 7
           $(
           GR(JJ) = GRAY(ROOTS+FLOAT(JJ-4), K1, TB1, TB2, TB3)
           IF ((BW == 2) & (GR(JJ) <= FLOAT(UG)))
              WIDTH = WIDTH + 1
           ELSE IF ((BW == 1) & (GR(JJ) >= FLOAT(LG)))
              WIDTH = WIDTH + 1
           ELSE
              WIDTH = WIDTH + 0
           $)
         $)
      WIDTHB = WIDTH
      IF(CLASS==0)                X=0
      ELSE                        X=ROOTBS
#
      CURVEB=DER2(X,TB2,TB3)
      CRVEBM = ABS(CURVEB)
#
#                                IS THE CURVATURE HIGH ENOUGH
#
      IF(CRVEBM < CRVTHR)GO TO 4000
      IF(CLASS==0)GO TO 4000
#
      RO = ROOTBS*SNB
      CO = ROOTBS*CSB
      SE = RDER(RO,CO)
      CE = CDER(RO,CO)
      GRADB = SQRT(SE*SE + CE*CE)
#
      IF (GRADB > GRADTH)
        $(
        SE = SE/GRADB
        CE = CE/GRADB
        $)
      ELSE
        $(
```

```
          SE = -CSB
          CE = SNB
          $)
#
#
        GRDCHB=ANGLEG(SNB,CSB,RO,CO)
        IF(GRDCHB>=1000*GRAD) ANGLEB=1000.
        ELSE                  ANGLEB=GRDCHB/GRAD
#
#
#                                IS THE GRADIENT ANGLE DIFFERENCE
#                                GREAT ENOUGH
#
        IF(ANGLEB > ANGTHR)
          CLASSB=CLASS
          IF((CLASSB==RIDGE&SNB<0.)!(((CLASSB==VALLEY)!(CLASSB==FALT))&SNB>=0.))
              $(
              IF ( SNB == 0. ) SNB = EPS
              SNB=-SNB
              CSB=-CSB
              $)
        THETAB=ATAN2(SNB,CSB)*RADDEG
4000    CONTINUE
        IF  (CLASSA == 0 & CLASSB == 0)
           $(
           XFLINE = 0
           DEPTHX = 0.
           RVTHET = 0.
           WTH    = 0
           GO TO 1000
           $)
        IF  (CLASSA == 0 & CLASSB ¬= 0)
           $(
           XFLINE = CLASSB
           DEPTHX = DEPTHB
           RVTHET = THETAB
           WTH   = WIDTHB
           GO TO 1000
           $)
        IF ( CLASSA ¬= 0 & CLASSB == 0 )
           $(
           XFLINE=CLASSA
           DEPTHX = DEPTHA
           RVTHET = THETAA
           WTH   = WIDTHA
           GO TO 1000
           $)
#
#                                  BOTH DIRECTIONS HAVE AN ASSIGNMENT
#
#
        IF (( CLASSA == RIDGE & CLASSB == RIDGE ) !
            ( CLASSA == VALLEY & CLASSB == VALLEY))
           $(
           XFLINE  = CLASSA
```

```
            IF (ABS(DEPTHA) > ABS(DEPTHB))
              $(
              DEPTHX = DEPTHA
              RVTHET = THETAA
              WTH    = WIDTHA
              $)
            ELSE
              $(
              DEPTHX = DEPTHB
              RVTHET = THETAB
              WTH    = WIDTHB
              $)
            GO TO 1000
            $)
         IF ( ( CLASSA == RIDGE  & CLASSB == VALLEY ) !
              ( CLASSA == VALLEY & CLASSB == RIDGE  ) )
            $(
              IF ( ABS(CURVEB) > ABS(CURVEA) )
                  $(
                  XFLINE=CLASSB
                  DEPTHX = DEPTHB
                  RVTHET = THETAB
                  WTH    = WIDTHB
                  $)
              ELSE
                  $(
                  XFLINE=CLASSA
                  DEPTHX = DEPTHA
                  RVTHET = THETAA
                  WTH    = WIDTHA
                  $)
            $)
1000      CONTINUE
#
        STR = DEPTHX

        DIR = RVTHET
#
#                                NORMAL RETURN
#
        CALL PPOP
        RETURN
        END



#--ZERCRO       ZERO CROSSING OF FIRST DIRECTIONAL DERIVATIVE        MID
#
#  IDENTIFICATION
#
#       TITLE           ZERCRO
#       AUTHOR          JUNGWHAN JOHN KIM
```

```
#       VERSION         A.01
#       DATE            OCTOBER 28, 1984
#       LANGUAGE        RATFOR
#       SYSTEM          VAX 11/780
#
#
#  UPDATE
#
#       UPDATE
#       AUTHOR
#       DATE
#       VERSION
#       PURPOSE
#
#  PURPOSE
#
#       GIVEN A CUBIC POLYNOMIAL EXPRESSION FOR THE FIRST DIRECTIONAL
#       DERIVATIVE, THIS FUNCTION SUBPROGRAM DETERMINES IF THERE IS
#       A ZERO-CROSSING OF THE FIRST DIRECTIONAL DERIVATIVE WITHIN A
#       GIVEN RADIUS.  IT RETURNS A 0 IF THERE IS NONE.
#       IT RETURNS A 1 IF THERE IS A WEAK LINE AND A 2 IF
#       THERE IS A STRONG LINE AND A. THE LINE PIXEL IS
#       ONLY DECLARED IF THE INFLECTION POINT IF MORE THAN 1.0 AWAY
#       FROM THE PIXEL CENTER AND IF THE DEPTH IS GREAT
#       ENOUGH.
#
#  ENTRY POINT
#
#       ZERCRO(TA1,TA2,TA3,RADIUS,SIZE,DEPTH)
#
#  ARGUMENT LISTING
#
#       TA1     REAL            COEFFICIENT OF X
#       TA2     REAL            COEFFICIENT OF X**2
#       TA3     REAL            COEFFICIENT OF X**3
#       RADIUS  REAL            RADIUS WITHIN WHICH THE ZERO IS SOUGHT
#       SIZE    REAL            RIGHT END POINT OF NEIGHBORHOOD
#       DEPTH   REAL            RELATIVE EXTREMA DEPTH
#
#  ROUTINES CALLED
#
#  COMMENT
#
#        THE DEPTH OF A VALLEY IS THE HEIGHT OF WATER THAT COULD FILL
#      THE VALLEY UP TO THW SMALLER OF THE HEIGHT OF THE ENDPOINT OR
#      THE INFLECTION POINT.  THE RANGE IS THE HIGHEST POINT MINUS THE
#      THE LOWEST POINT ON THE DOMAIN OF THE NEIGHBORHOOD INTERVAL.
#      THE DEPTH RATIO IS THE DEPTH/RANGE.
#
#
#*********************************************************************
#
#
#       FUNCTION ZERCRO(TA1,TA2,TA3,RADIUS,SIZE,DEPTH)
#
```

```
      REAL TA1,TA2,TA3,ROOTL,ROOTS,AMIN1,AMAX1,ABS,DIST,DISTHR
      REAL SIZE,EPS,VLE,VRE,VI,VS,VL,HIGH,LOW,RANGE,XINFLE,X
      REAL VALUE,RADIUS,DEPTH,GTHETA
      INTEGER ZERCRO,RIDGE,VALLEY,FLAT,CL
      COMMON /ROOT/ ROOTL,ROOTS
#
      VALUE(X)= (TA1+(TA2+TA3*X)*X)*X
      DATA RIDGE,VALLEY/1,2/
      DATA EPS/1.E-3/
      DATA DISTHR /1.0/
#
#     CALL PPUSH("ZERCRO")
      ZERCRO=0
      DEPTH=0.
#
#                                 DETERMINE THE LOCATIONS OF THE
#                                 ZERO-CROSSINGS OF FIRST DERIVATIVE
#
      A2=3.*TA3
      A1=2.*TA2
      A0=TA1
#
#
#                                 IS THE POLYNOMIAL QUADRATIC?
#
      IF(ABS(A2)>0.)
#
#                                 IS THE DISCRIMINANT POSITIVE OR
#                                 NEGATIVE?
#
        $(
        D=A1*A1-4.*A2*A0
        IF(D >= 0.)
#
#                                 DISCRIMINANT IS POSITIVE; THERE
#                                 ARE TWO ROOTS
#
          $(
          NUMRT = 2
          IF(A1 > 0.)            ROOTL = (-A1 - SQRT(D))/(2.*A2)
          ELSE                   ROOTL = (-A1 + SQRT(D))/(2.*A2)
          IF(ABS(ROOTL) > 0.)    ROOTS = A0/(A2*ROOTL)
          ELSE                   ROOTS = 0.
          $)
        ELSE
#
#                                 DISCRIMINANT IS NEGATIVE; THERE
#                                 ARE NO REAL ROOTS
#
          $(
          NUMRT = 0
          ROOTL = -9999.
          ROOTS = -9999.
          GO TO 1
          $)
```

```
          $)
#       ELSE
#
#                                   POLYNOMIAL IS NOT QUADRATIC
#
          $(
          IF(ABS(A1) > 0.)
#
#                                   POLYNOMIAL IS FIRST ORDER
#
            $(
            NUMRT = 1
            ROOTL = 9999.
            ROOTS = -A0/A1
            $)
          ELSE
#
#                                   POLYNOMIAL IS ZERO ORDER
#
            $(
            NUMRT = 0
            ROOTL = -9999.
            ROOTS = -9999.
            GO TO 1
            $)
          $)
#
1       CONTINUE
#
        IF(NUMRT==0) GO TO 10
#
#                                 BOUND THE ROOTS TO WITHIN THE
#                                 NEIGHBORHOOD
#
        IF(ROOTL<-SIZE)ROOTL=-SIZE
        IF(ROOTL> SIZE)ROOTL= SIZE
        IF(ROOTS<-SIZE)ROOTS=-SIZE
        IF(ROOTS> SIZE)ROOTS= SIZE
#
#                                 IS THE ZERO-CROSSING CLOSE ENOUGH
#
        IF(ABS(ROOTS)<RADIUS)
          $(
          IF  (NUMRT == 1)
            $(
            VLE = VALUE(-SIZE)
            VRE = VALUE( SIZE)
            VS  = VALUE(ROOTS)
            DEPTH = AMAX1(VLE,VRE) - VS
            IF     (TA2 > 0. )   ZERCRO = VALLEY
            ELSE                 ZERCRO = RIDGE
            GO TO 20
            $)
          IF  (NUMRT == 2)
```

```
$(
IF (ABS(ROOTL-ROOTS) < 1.5)
 $(
 ZERCRO = 0
 GO TO 20
 $)
#
#                                THERE IS A VALLEY IF THE 2ND DERIVATIVE
#                                IS POSITIVE, OTHERWISE IT IS A RIDGE
#
IF (A2*ROOTS+A1 > 0.)
 ZERCRO = VALLEY
ELSE
 ZERCRO = RIDGE
#
#                                IS DEPTH GREAT ENOUGH
#
IF(ABS(TA3)>EPS)
 $(
#
#                                DETERMINE THE INFLECTION POINT
#
 XINFLE=-TA2/A2
#
#                                FOR A GOOD RIDGE OR VALLEY, THE
#                                INFLECTION POINT MUST BE FAR AWAY
#                                FROM THE PIXEL CENTER.
#
 DIST=ABS(XINFLE)
 IF(DIST< DISTHR)
    $(
    ZERCRO=0
    GO TO 20
    $)
#
#                                BOUND THE INFLECTION  TO WITHIN
#                                THE NEIGHBORHOOD
#
 IF(XINFLE<-SIZE)XINFLE=-SIZE
 IF(XINFLE>SIZE )XINFLE=SIZE
#
#                                DETERMINE THE VALUES AT THE
#                                LEFT AND RIGHT ENDPOINTS OF NEIGHBORHOOD
#
 VLE=VALUE(-SIZE)
 VRE=VALUE(SIZE)
 VI=VALUE(XINFLE)
 VS =VALUE(ROOTS)
 VL =VALUE(ROOTL)
#
#
#
#
#
 IF(ROOTS<ROOTL)
```

```
                IF(ZERCRO==VALLEY)
                   DEPTH=AMIN1(VLE,VI)-VS
                ELSE                DEPTH=VS-AMAX1(VLE,VI)
#
             ELSE
                IF(ZERCRO==VALLEY)
                   DEPTH=AMIN1(VRE,VI)-VS
                ELSE                DEPTH=VS-AMAX1(VRE,VI)
#

            $)
           ELSE
            $(
            VLE = VALUE(-SIZE)
            VRE = VALUE( SIZE)
            VS  = VALUE(ROOTS)
            DEPTH = AMAX1(VLE,VRE) - VS
            $)
           $)
        $)
      GO TO 20
#
10    CONTINUE
#
      ROOTS=-SIZE
      ROOTL=SIZE
20    CONTINUE
#
#     CALL PPOP
      RETURN
      END
```

```
*ROAD      Road finder

VERSION:   A.01    DATE: 11-27-83      AUTHOR: J. J. KIM

ACTION:    This command takes a line image (created by the FLINE command), the
           connected component image of the line image (by the CONCM command),
           and the original sif image from which the line image is originated.
           Then, it generates the most probable road image by screening the
           line image, component by component, on the basis of the road's
           characteristics as follows:
                (1) Connectivity -- the number of pixels in a component
                (2) Straightness -- mean angle difference of a component
                (3) Contrast -- mean strenght of a component
                (4) Gray-tone -- mean gray-tone of a component
           If two output files are input, it generates a property file
           into the second output file. The properties for each connected
           component of the connected component image are as follows:
                 1. the number of pixels in a component
```

2. the mean gray-tone value
3. the standard deviation of gray-tone values
4. the mean strength (contrast)
5. the standard deviation of strengths
6. the mean angle difference
7. the standard dviation of angle differences

SOURCE: Disk, input file name

DESTINATION: Disk, output file name

QUESTION: (1) Enter threshold for connectivity (d=20, 1 to MAXIMUM NO.)
The reply to this question should be the minimum number of pixels in a component in order to be a road segment.
(2) Enter threshold for contrast (d= 1000. 0. to 10000.)
The reply to this question should be the mean strengh of a component in order to be considered as a road segment.
(3) Enter threshold for straightness (d=15. 0. to 180.)
The reply to this question should be the mean angle difference of a component to be considered as a road segment.
(4) Enter threshold for gray-tone (d=25000 0 to 65000)
The reply to this question should be the mean gray-tone values of a component to be considered as a road segment.

COMMAND STRING EXAMPLE:

```
ROAD BAR.ROD < BAR.LIN, BAR.CMP, BAR.SIF

Enter threshold for connectivity (D=20, 1 to 2500) 6
Enter threshold for contrast (D=1000. 0. to 10000.) 1000.
Enter threshold for straighness (D=15. 0 to 180.) 25.
Enter threshold for gray-tone (D=25000, 0 to 65000) 25000
```

BAR.LIN contains 4 bands : Strengh band, Angle band, Line image band, and Width band. BAR.CMP is a connectec component image of the line image of BAR.LIN. BAR.SIF is a original bar image. The created road image is BAR.ROD. In this example, in order for a component to be a road segment, the component contains at least 6 pixels, its mean contrast must exceed 1000, its straightness, 25 and its mean gray-tone ,25000.

```
ROAD BAR.ROD, BAR.PRP < BAR.LIN, BAR.CMP, BAR.SIF

Enter threshold for connectivity (D=20, 1 to 2500) 6
Enter threshold for contrast (D=1000. 0. to 10000.) 1000.
Enter threshold for straightness (D=15. 0. to 180.) 25.
Enter threshold for gray-tone (D=25000, 0 to 65000) 25000
```

Same as above except it generates an extra file BAR.PRP in which the seven properties for each connected component of BAR.CMP are stored.

REFERENCE:
Author's master thesis chapter 4.

```
$ !!
$ !!     ROAD.RUN
$ !!
$ !!     MAKE A PARABOLA FACING UP
$ !!
$       MKBAR ROAD.IMG
        50
        V
        L
$       ARITHM ROAD.SUB < ROAD.IMG (S)
        25
        N
$       IMGPWR ROAD.PWR < ROAD.SUB
        2
        100
$ !!
$ !!            GET THE CUBIC FACET PARAMETERS
$ !!
$        FACET2 ROAD.CUB < ROAD.PWR (W)
         9
         9
         3
         1.5

$ !!
$ !!            GET A LINE IMAGE
$ !!
$       FLINE ROAD.LIN < ROAD.CUB
         2
         1.0
         1.0
         0
         10
         1
         6
$ !!
$ !!            GET A CONNECTED COMPONENT IMAGE
$ !!
$       CONCM ROAD.CMP < ROAD.LIN
         2
         1
         3
$ !!
$ !!            RUN THE ROAD COMMAND TO CREATE A ROAD IMAGE
$ !!
$       ROAD ROAD.ROD < ROAD.LIN, ROAD.CMP, ROAD.PWR
        15
        100
        25
        100
```

```
$ !
$ !             TO TEST THE RESULT WE WILL MAKE A TEST IMAGE IN WHICH
$ !             EVERY PIXEL OF THE CONNECTED COMPONENT IS ASSGINED 1
$ !             AND THE BACKGROUND IS ASSGINED 0.
$ !
$       TRSLD ROAD.TRS < ROAD.PWR
        0
$ !
$ !             THERE SHOULD BE NO DIFFERENCE
$ !
$       RMS TT < ROAD.TRS,ROAD.ROD
$ !
$      DELETE
       ROAD.IMG
       ROAD.SUB
       ROAD.PWR
       ROAD.CUB
       ROAD.LIN
       ROAD.CMP
       ROAD.ROD
       ROAD.TRS

$!
$      EXIT


#--DROAD        DRIVER OF COMMAND ROAD                                  MID
#
#  IDENTIFICATION
#
#       TITLE           DROAD
#       AUTHOR          JUNGWHAN KIM
#       VERSION         A.01
#       DATE            10-20-1984
#       LANGUAGE        RATFOR
#       SYSTEM          VAX 11/780
#
#  UPDATE
#
#       UPDATE
#       AUTHOR
#       DATE
#       PURPOSE
#
#  PURPOSE
#
#
#  ENTRY POINT
#
#      DROAD (WORK, *)
#
```

```
#  ARGUMENT LISTING
#
#      WORK     INT.ARR        WORKING STORAGE ARRAY
#      *        INT            ALTERNATE ERROR 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)
#      OSGTNM       GET NAME FROM FILE DESCRIPTOR      (GIPSY OS KERNEL)
#      RDKINL       INITIALIZE AND ACCESS A STANDARD IMAGE
#                   FILE.                              (GIPSY PRIMITIVE)
#      OSALOC       CHECK AND EXTEND USER WORK SPACE   (GIPSY OS KERNEL)
#      RNGETI       GET AN INTEGER WITHIN A RANGE (WITH
#                   PROMPT).                           (GIPSY PRIMITIVE)
#      RNGETR       GET AN REAL WITHIN A RANGE (WITH PROMPT).
#                                                      (GIPSY PRIMITIVE)
#      GETWP        GET WORK ARRAY POINTER.            (GIPSY PRIMITIVE)
#      NROAD        THIS IS THE NUMBER CRUNCHER ROUTINE
#                   FOR THE COMMAND "ROAD".            (GIPSY ROUTINE)
#      PPOP         POP PROGRAM NAME FROM THE ERROR STACK
#                                                      (GIPSY PRIMITIVE)
#      CLOSE        CLOSE A FILE.                      (GIPSY PRIMITIVE)
#
#*********************************************************************
#
#
#
#      INCLUDE MACA1
#
#      SUBROUTINE DROAD (WORK, *)
#
#      IMPLICIT INTEGER (A-Z)
#
#      INCLUDE GIPCOM
#      INCLUDE ERROR
#      INCLUDE TTCOM
#
#      CHARACTER NAME (.FILENAMELENGTH)
#
#      INTEGER IDENT1(.IDLENGTH)
#      INTEGER IDENT2(.IDLENGTH)
#      INTEGER IDENT3(.IDLENGTH)
#      INTEGER WORK (.ARB)
#
#
```

```
      INTEGER TRSLD
      REAL TLDSTR,TLDANV,TLDGREY
#
      LOGICAL ONE
#
#
      EQUIVALENCE (NPPL, IDENT1(.IDNPPL)), (NLINS, IDENT1(.IDNLINS))
      EQUIVALENCE (NCOLS, IDENT1(.IDNCOLS)), (NROWS, IDENT1(.IDNROWS))
      EQUIVALENCE (NBND, IDENT1(.IDNBNDS)), (MODE, IDENT1(.IDMODE))
      EQUIVALENCE (NSBND, IDENT1(.IDNSBNDS)), (MAXLBL, IDENT2(.IDMAX))
#
#                          PUSH PROGRAM NAME ONTO ERROR STACK
#
      CALL PPUSH ("DROAD")
#
#                          CHECK THE NUMBER OF OUTPUT FILES (1 OR 2)
#
      ONE = (.OK ¬= OSGTNM(FDO2,NAME))
#
      IF (ONE)
         IF (.OK ¬= OSTFIL(FDO2)) GO TO 9020
#
#                          SET UP INPUT FILE
#
      CALL RDKINL (FDI1, IDENT1, .READONLY, IEV, %9999)
      CALL CLOSE(FDI1)
      CALL RDKINL (FDI2, IDENT2, .READONLY, IEV, %9999)
      CALL CLOSE(FDI2)
      CALL RDKINL (FDI3, IDENT3, .READONLY, IEV, %9999)
      CALL CLOSE(FDI3)
#
#
#
#                          CHECK THE INPUT FILE
#
      IF (NPPL ¬= NCOLS | NROWS ¬= 1) GO TO 9010
      IF (NSBND == 0) GO TO 9020
      IF (MODE ¬= .INTMODE) GO TO 9000
      IF (NBND < 3) GO TO 9020
#
#
#
#                          SET UP BUFFERS
#
      NXT = 1
      SBUFF = GETWP (NXT, .INTMODE, 3*(NPPL+2))
      ABUFF = GETWP (NXT, .INTMODE, 3*(NPPL+2))
      IBUFF = GETWP (NXT, .INTMODE, 3*(NPPL+2))
      OBUFF = GETWP (NXT, .INTMODE, NPPL+2)
      INP   = GETWP (NXT, .INTMODE, NPPL)
      INP3  = GETWP (NXT, .INTMODE, 3*(NPPL+2))
      OCCUR = GETWP (NXT, .REALMODE, 7*MAXLBL)
      IMAGE = GETWP (NXT, .INTMODE, NPPL)
      ST    = getwp (nxt, .INTMODE, NPPL)
```

```
      AN    = getwp (nxt, .INTMODE, 3*(NPPL+2))
#
#                                ALLOCATE THAT SPACE
#
      IF  (.OK ¬= OSALOC (NXT))
         GO TO 9030
#
#                                GET THE USER PARAMETERS
#
      STRBND = 1
      ANGBND = 2
      RDBND  = 3
      CALL RNGETI("Enter threshold for connectivity ",
                  1, NPPL*NPPL, 20, TRSLD, IEV, %9999)
      CALL RNGETR("Enter threshold for contrast ",
                  0.0,10000.0,1000.0,tldstr,IEV,%9999)
#
      CALL RNGETR("Enter threshold for straightness ",
                  0.0,180.0 ,15.0,tldanv,iev,%9999)
      CALL RNGETR("Enter threshold for gray-tone ",
                  0.0,65000.0,25000.0,TLDGREY,IEV,%9999)
#                               CALL THE NUMBER CRUNCHER
#
      NPPL2 = NPPL + 2
      CALL NROAD (FDI1, FDI2, FDI3, FDO1, FDO2, WORK(SBUFF), WORK(ABUFF),
                  WORK(IBUFF), WORK(OBUFF), WORK(INP), WORK(INP3),
                  WORK(OCCUR), WORK(IMAGE),WORK(ST),WORK(AN), MAXLBL, TRSLD,
                  TLDSTR,TLDANV,TLDGREY,STRBND,
                  ANGBND, RDBND, NPPL, NPPL2,TTT, IEV, %9998)
#
#                                NORMAL RETURN
#
      CALL CLOSE (FDO1)
      IF (ONE)
         IEV = OSCLOS(FDO2, .DELETE)
      ELSE
         CALL CLOSE (FDO2)
      CALL CLOSE (FDI1)
      CALL CLOSE (FDI2)
      CALL CLOSE (FDI3)
      CALL PPOP
      RETURN
#
#                                ABNORMAL CONDITIONS
#
 9000 CONTINUE
#
#                                NOT AN INTEGER FILE
#
      IEV = -2012
      GO TO 9998
#
 9010 CONTINUE
#
#                                NOT IN LINE FORMAT
```

```
#
      IEV = -5001
      GO TO 9998
#
 9020 CONTINUE
#
#                                BANDS SPECIFIED INCORRECTLY
#
      IEV = -5018
      GO TO 9998
#
 9030 CONTINUE
#
#                                NOT ENOUGH WORK SPACE
#
      IEV = -5010
      GO TO 9999
#
 9998 CONTINUE
#
#                                ERROR IN SUBPROGRAM
#
#
#                                ABNORMAL RETURN
#
 9999 CONTINUE
      CALL CLOSE (FD01)
      IF (ONE)
         IEV = OSCLOS(FD02, .DELETE)
      ELSE
         CALL CLOSE (FD02)
      CALL CLOSE (FD11)
      CALL CLOSE (FD12)
      CALL CLOSE (FD13)
      RETURN 1
      END


#--NROAD        NUMBER CRUNCHER FOR THE THINEG COMMAND              MID
#
#  IDENTIFICATION
#
#       TITLE          NROAD
#       AUTHOR         JUNGWHAN KIM
#       VERSION        A.01
#       DATE           10-20-1984
#       LANGUAGE       RATFOR
#       SYSTEM         VAX 11/780
#
#  UPDATE
#       UPDATE
```

```
#       AUTHOR
#       DATE
#       PURPOSE
#
#  PURPOSE
#
#
#  ENTRY POINT
#
#       NROAD  (FDI1, FDI2,FDI3,FDO,FDO2, SBUFF, ABUFF, IBUFF, OBUFF,
#               INP, INP3, OCCUR, IMAGE,ST, AN,MAXLBL,TRSLD,TLDSTR,
#               TLDANV,TLDGREY,STRBND,ANGBND,RDBND,NPPL,NPPL2,TTT,
#               IEV, *)
#
#  ARGUMENT LISTING
#
#       FDI1     CHR.ARR        INPUT FILE DESCRIPTOR
#       FDO      CHR.ARR        OUTPUT FILE DESCRIPTOR
#       SBUFF    INT.ARR        INPUT BUFFER
#       ABUFF    INT.ARR        INPUT BUFFER
#       IBUFF    INT.ARR        INPUT BUFFER
#       OBUFF    INT.ARR        OUTPUT BUFFER
#       TRSLD    REAL           GRADIENT THRESHOLD
#       BKGRND   INT            BACKGROUND SYMBOL
#       NPPL     INT            NPPL OF INPUT FILE
#       IEV      INT            INTEGER EVENT VARIABLE
#       *        INT            ALTERNATE ERROR RETURN
#
#  INCLUDE FILES/COMMONS
#
#       MACA1      INCLUDE     GIPSY GENERAL SYMBOL DEFINITIONS
#       DSCAX      INCLUDE     TYPE CODES FOR AUXILLIARY DATA IN DE
#                              SCRIPTOR RECORDS.
#       TTCOM      COMMON      FD'S FOR TERMINAL AND RUNFILE IO
#
#  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)
#       COPYDS       COPY DESCRIPTOR RECORDS FROM TEMP FILE.
#                                                    (GIPSY PRIMITIVE)
#       GDSCAX       GET AUXILIARY DATA FROM DESCRIPTOR
#                    RECORDS                         (GIPSY PRIMITIVE)
#       AREAD        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)
#
#
```

```
#*****************************************************************
#
#
#
#       INCLUDE MACA1
#
        SUBROUTINE NROAD (FDI1, FDI2, FDI3, FDO,FDO2, SBUFF, ABUFF, IBUFF,
                          OBUFF, INP,INP3, OCCUR, IMAGE,ST, AN,
                          MAXLBL, TRSLD, TLDSTR,TLDANV,TLDGREY,STRBND,
                          ANGBND, RDBND, NPPL, NPPL2, TTT, IEV, *)
#
        IMPLICIT INTEGER (A-Z)
#
        INCLUDE DSCAX
        INCLUDE TTCOM
        INCLUDE PFCOM
        CHARACTER FDI1 (.FDLENGTH)
        CHARACTER FDI2 (.FDLENGTH)
        CHARACTER FDI3 (.FDLENGTH)
        CHARACTER FDO (.FDLENGTH)
        CHARACTER FDO2 (.FDLENGTH)
#
#
        INTEGER IDENT1(.IDLENGTH), IDENT2(.IDLENGTH), IDENT3(.IDLENGTH)
        INTEGER JDENT(.IDLENGTH), OHEAD(.HEADLENGTH)
        INTEGER SBUFF(NPPL2, 3), ABUFF(NPPL2, 3), IBUFF(NPPL2, 3)
        INTEGER OBUFF(NPPL), INP(NPPL),INP3(NPPL2,3)
        INTEGER IMAGE(NPPL),ST(NPPL),AN(NPPL2,3)
        INTEGER CAN, AAN
        REAL TTEST,SPOOL, ELM, SQELM, LONGST,CMPANM,CMPANV,CMPSTM,CMPSTV
        REAL SUMOSQE, SUMELM,SUMOSQS,SUMSST,SUMOSQA,SUMAAN,SST,SQSST,FAAN,SQAAN
        REAL TLDSTR,TLDANV,ROADANM,ROADANV,ROADSTM,ROADSTV,TLDGREY
        REAL OCCUR(MAXLBL,7)
        REAL PRPO(7)
#
        EQUIVALENCE (NLIN, IDENT1(.IDNLINS))
#
        DATA PI    / 3.14159 /
#
#
#
#                           PUSH PROGRAM NAME ONTO ERROR STACK
        CALL PPUSH("NROAD")
#
#                           OPEN INPUT FILES
#
#                           COPY DESCRIPTOR RECORDS FROM INPUT
#                           IMAGE, ROUTINE NAME, AND PARAMETRS
#                           TO TEMPORARY SEQUENTIAL FILE.
#                           OPEN INPUT FILES
#
        CALL CPYIDR (FDI1, IDENT1, .OPNTMP, IEV, %9998)
        CALL CPYIDR (FDI2, IDENT2, .NOOPNTMP, IEV, %9998)
```

```
      CALL CPYIDR (FDI3, IDENT3, .NOOPNTMP, IEV, %9998)
#
      CALL DSCNAM ("ROAD", IEV, %9998)
#
#
#
#
#
                              OPEN OUTPUT IMAGE AND COPY TO IT THE
                              DESCRIPTOR RECORDS FROM TEMPORARY FILE
      DO I = 1, .IDLENGTH
        JDENT(I) = 0
#
      JDENT(.IDNPPL    ) = NPPL
      JDENT(.IDNLINS   ) = NLIN
      JDENT(.IDNCOLS   ) = NPPL
      JDENT(.IDNROWS   ) = 1
      JDENT(.IDNBNDS)    = 1
      JDENT(.IDNSBNDS ) = 1
      JDENT(.IDNBITS)    = 20
#
      CALL COPYDS (FDO, JDENT, IEV, %9998)
#
#
#
#
#
#
#
#
#
                              INITIALIZE THE PROPERTY ARRAY WHICH FOR
                              EACH COMPONENT, WILL CONTAIN THE NUMBER
                              OF PIXELS, SUM AND SQUARE SUM OF GRAY-TONE,
                              STRENGTH, AND ANGLE DIFFERENCE. THESE
                              VALUES ARE USED FOR COMPUTING THE MEAN AND
                              STANDARD DEVIATION OF GRAY-TONE, CONTRAST,
                              AND STRAIGHTNESS.

      DO II = 1, MAXLBL
         DO JJ = 1,7
        OCCUR(II,JJ) = 0.0001
      SM = 0.0
      SMOSQ = 0.0
      DO I = 1, NLIN
        $(
        CALL RREAD (FDI3, IMAGE,1,I, IDENT3, .WAIT, IEV, %9999 )
        CALL RREAD (FDI1, ST,  1, I, IDENT1, .WAIT, IEV, %9999 )
        DO K = 1, 3
           $(
            CALL AREAD (FDI2, INP3(1,K),NPPL2, 1, 1, 1, 1, 1,
                        I+K-1, IDENT2, .WAIT, IEV, %9999)
            CALL AREAD (FDI1, AN(1,K), NPPL2, 1, 1, 1, 1, 2,
                        I+K-1, IDENT1, .WAIT, IEV, %9999)
            $)
        DO  J = 1, NPPL
            $(
            LABL = INP3(J+1,2)
            CAN  = AN (J+1,2)
            ELM  = FLOAT(IMAGE(J))/10000.0
            SQELM = ELM * ELM
            SST  = FLOAT(ST(J))/100.0
            SQSST = SST * SST
            IF (LABL > 0)
```

```
                $(
                OCCUR(LABL,1) = OCCUR(LABL,1) + 1.
                OCCUR(LABL,2) = OCCUR(LABL,2) + ELM
                OCCUR(LABL,3) = OCCUR(LABL,3) + SQELM
                OCCUR(LABL,4) = OCCUR(LABL,4) + SST
                OCCUR(LABL,5) = OCCUR(LABL,5) + SQSST
                IF      (LABL == INP3(J+2,2)) AAN = ABS(CAN-AN(J+2,2))
                ELSE IF (LABL == INP3(J+1,3)) AAN = ABS(CAN-AN(J+1,3))
                ELSE IF (LABL == INP3(J  ,2)) AAN = ABS(CAN-AN(J  ,2))
                ELSE IF (LABL == INP3(J+1,1)) AAN = ABS(CAN-AN(J+1,1))
                ELSE IF (LABL == INP3(J+2,3)) AAN = ABS(CAN-AN(J+2,3))
                ELSE IF (LABL == INP3(J  ,3)) AAN = ABS(CAN-AN(J  ,3))
                ELSE IF (LABL == INP3(J  ,1)) AAN = ABS(CAN-AN(J  ,1))
                ELSE IF (LABL == INP3(J+2,1)) AAN = ABS(CAN-AN(J+2,1))
                ELSE                          AAN = 30
                FAAN = FLOAT(AAN)
                SQAAN = FAAN * FAAN
                OCCUR(LABL,6) = OCCUR(LABL,6) + FAAN
                OCCUR(LABL,7) = OCCUR(LABL,7) + SQAAN
                $)
            $)
        $)
#
#
#
#
        LONGST = 0.001
        SUMELM    = 0.
        SUMOSQE = 0.
        SUMSST  = 0.
        SUMOSQS = 0.
        SUMAAN  = 0.
        SUMOSQA = 0.
        FTRSLD = FLOAT(TRSLD)
#
#                          COMPUTE THE MEAN AND STANDARD DEVIATION
#                          OF GRAY-TONE, CONTRAST,AND ANGLE
#                          DIFFERENCE FOR EACH LABLED COMPONENT.
#
#
#
        DO I = 1, MAXLBL
            $(
            OCCUR(I,4) = OCCUR(I,4)/OCCUR(I,1)
            IF (OCCUR(I,1) < 1.) OCCUR(I,5)= 0.01
            ELSE
                OCCUR(I,5) = (OCCUR(I,5) -
                        OCCUR(I,1)*(OCCUR(I,4))*(OCCUR(I,4)))/OCCUR(I,1)
            OCCUR(I,6) = OCCUR(I,6)/OCCUR(I,1)
            IF (OCCUR(I,1) < 1.) OCCUR(I,7) = 0.01
            ELSE
                OCCUR(I,7) = (OCCUR(I,7) -
                        OCCUR(I,1)*(OCCUR(I,6))*(OCCUR(I,6)))/OCCUR(I,1)
            OCCUR(I,2) = OCCUR(I,2)/OCCUR(I,1)
            IF (OCCUR(I,1) < 1.) OCCUR(I,3) = 0.01
            ELSE
```

```
            OCCUR(I,3) = (OCCUR(I,3) -
                    OCCUR(I,1)*(OCCUR(I,2))*(OCCUR(I,2)))/OCCUR(I,1)
        IF (OCCUR(I,4) > TLDSTR/100.0)
            $(
            LONGST = LONGST + OCCUR(I,1)
            SUMSST = SUMSST + OCCUR(I,4)*OCCUR(I,1)
            SUMOSQS= SUMOSQS+ OCCUR(I,5)*OCCUR(I,1)
            SUMOSQA= SUMOSQA+ OCCUR(I,7)*OCCUR(I,1)
            $)
        $)
    ROADSTM = SUMSST/LONGST
    ROADANV = SUMOSQA/LONGST
    ROADSTV = SUMOSQS/LONGST
#
#                           MAKE PROPERTY FILE
#
#                           SET UP OUTPUT IHEAD ARRAY
#
    DO I = 1, HEADLENGTH
        OHEAD (I) = 0
    OHEAD(.IDLOWL) = 1
    OHEAD(.IDHIL)  = MAXLBL
    OHEAD(.IDNPRP) = 7
    OHEAD(.IDLREC) = 7
    OHEAD(.IDWINDX)   = NPPL
    OHEAD(.IDWINDY)  = NLIN
    OHEAD(.IDPMODE) = .REALMODE
#
#                           OPEN OUTPUT PROPERTY FILE
#
    CALL PFOPEN (FD02, OHEAD, .NEW, IEV, %9999)
#
#                           COPY FROM OCCUR ARRAY
    DO I = 1, MAXLBL
      $(
       DO J = 1, 7
          PRPO(J) = OCCUR(I,J)
       CALL PWRITE (FD02, PRPO, I, OHEAD, .WAIT, IEV, %9999)
      $)
#
#
#
#                           SCREEN THE LABLED COMPONENTS USING
#                           THE THRESHOLDS AND THEIR STATISTICS.
#
#
    DO I = 2, NLIN + 1
      $(
      DO K = 1, 3
      CALL AREAD(FD12,INP3(1,K),NPPL2,1,1,1,1,1,
                 I+K-2,IDENT2, .WAIT,IEV,%9999)
      DO L = 2, NPPL+1
         $(
          NEWLBL = 0
          LABL = INP3(L,2)
```

```
            IF (LABL ¬= 0)
               $(
               IF (((OCCUR(LABL,1) >= FLOAT(TRSLD))&(
                 OCCUR(LABL,2)<TLDGREY/10000.0))&(
                 (OCCUR(LABL,4)>TLDSTR/100.0)&(OCCUR(LABL,6)<TLDANV)))
                  NEWLBL = MAXLBL + 2
               ELSE IF ((OCCUR(LABL,1) >= FLOAT(TRSLD))&(
                        OCCUR(LABL,7) < TLDANV*TLDANV))
                  $(
                  CMPANV = OCCUR(LABL,7) # ANGLE VARIANCE
                  CMPSTV= OCCUR(LABL,5) # STRENGTH VARIANCE
                  IF ((ROADANV/CMPANV>.4)&(ROADANV/CMPANV<3.))
                     $(
                     SPOOL = SQRT((LONGST*ROADSTV*ROADSTV+OCCUR(LABL,1)*
                                   CMPSTV*CMPSTV)/(LONGST+OCCUR(LABL,1)))
                     TTEST = ABS(ROADSTM-OCCUR(LABL,4))/(
                             SPOOL*SQRT(1./LONGST/OCCUR(LABL,1)))
                     IF (TTEST<2.326)
                        NEWLBL = MAXLBL + 2
                     $)
                  $)
               $)
            OBUFF(L-1) = NEWLBL + LABL
         $)
      CALL RWRITE(FDO, OBUFF, 1, I-1, JDENT, .WAIT, IEV, %9999)
      $)
#
#
#                          IF TWO PIXELS ARE FROM DIFFERENT COMPONENTS
#                          AND THEY ARE ONE PIXEL APAPRT, THEN CONNECT
#                          THEM.
#
#
   DO I = 2, NLIN + 1
      $(
      DO K = 1, 3
      CALL AREAD(FDO,INP3(1,K),NPPL2,1,1,1,1,1,
                 I+K-2,JDENT, .WAIT,IEV,%9999)
      DO L = 2, NPPL+1
         $(
          NEWLBL = 0
          LABL = INP3(L, 2)
          IF (LABL ¬= 0)
            $(
            IF (LABL > MAXLBL + 2)
              NEWLBL = 1
            ELSE
              NEWLBL = 0
            $)
#
#
#                          THE PIXEL IS NOT LABLED: CONSIDER
#                          ITS 3 X 3 NEIGHBORHOOD AND CHECK
#                          IF ANY PAIR OF THE NEIGHBORS SATISFY
#                          THE CONNECTING CONDITION. IF SO,
```

```
#
#                    LABEL THE CENTER PIXEL BY ASSIGNING
#                    A NON-ZERO LABEL.
#
#
          ELSE IF ((INP3(L-1,1) ¬= 0) & (INP3(L+1,3) ¬= 0))
                $(
                LBL1 = INP3(L-1,1)
                LBL2 = INP3(L+1,3)
                IF ((LBL1 ¬= LBL2) &
                    ((LBL1 > MAXLBL+2) & (LBL2 > MAXLBL+2)))
                   NEWLBL = 2
                $)
          ELSE IF ((INP3(L+1,1) ¬= 0) & (INP3(L-1,3) ¬= 0))
                $(
                LBL1 = INP3(L+1,1)
                LBL2 = INP3(L-1,3)
                IF ((LBL1 ¬= LBL2) &
                    ((LBL1 > MAXLBL+2) & (LBL2 > MAXLBL+2)))
                   NEWLBL = 2
                $)
          ELSE IF ((INP3(L,1) ¬= 0) & (INP3(L,3) ¬= 0))
                $(
                LBL1 = INP3(L,1)
                LBL2 = INP3(L,3)
                IF ((LBL1 ¬= LBL2) &
                    ((LBL1 > MAXLBL+2) & (LBL2 > MAXLBL+2)))
                   NEWLBL = 2
                $)
          ELSE IF ((INP3(L+1,2) ¬= 0) & (INP3(L-1,2) ¬= 0))
                $(
                LBL1 = INP3(L+1,2)
                LBL2 = INP3(L-1,2)
                IF ((LBL1 ¬= LBL2) &
                    ((LBL1 > MAXLBL+2) & (LBL2 > MAXLBL+2)))
                   NEWLBL = 2
                $)
          ELSE IF ((INP3(L,1) ¬= 0) & (INP3(L-1,3) ¬= 0))
                $(
                LBL1 = INP3(L,1)
                LBL2 = INP3(L-1,3)
                IF ((LBL1 ¬= LBL2) &
                    ((LBL1 > MAXLBL+2) & (LBL2 > MAXLBL+2)))
                   NEWLBL = 2
                $)
          ELSE IF ((INP3(L,1) ¬= 0) & (INP3(L+1,3) ¬= 0))
                $(
                LBL1 = INP3(L,1)
                LBL2 = INP3(L+1,3)
                IF ((LBL1 ¬= LBL2) &
                    ((LBL1 > MAXLBL+2) & (LBL2 > MAXLBL+2)))
                   NEWLBL = 2
                $)
          ELSE IF ((INP3(L,3) ¬= 0) & (INP3(L-1,1) ¬= 0))
                $(
                LBL1 = INP3(L,3)
```

```
                LBL2 = INP3(L-1,1)
                IF ((LBL1 ¬= LBL2) &
                    ((LBL1 > MAXLBL+2) & (LBL2 > MAXLBL+2)))
                   NEWLBL = 2
                $)
            ELSE IF ((INP3(L,3) ¬= 0) & (INP3(L+1,1) ¬= 0))
                $(
                LBL1 = INP3(L,3)
                LBL2 = INP3(L+1,1)
                IF ((LBL1 ¬= LBL2) &
                    ((LBL1 > MAXLBL+2) & (LBL2 > MAXLBL+2)))
                   NEWLBL = 2
                $)
            ELSE IF ((INP3(L-1,2) ¬= 0) & (INP3(L+1,1) ¬= 0))
                $(
                LBL1 = INP3(L-1,2)
                LBL2 = INP3(L+1,1)
                IF ((LBL1 ¬= LBL2) &
                    ((LBL1 > MAXLBL+2) & (LBL2 > MAXLBL+2)))
                   NEWLBL = 2
                $)
            ELSE IF ((INP3(L-1,2) ¬= 0) & (INP3(L+1,3) ¬= 0))
                $(
                LBL1 = INP3(L-1,2)
                LBL2 = INP3(L+1,3)
                IF ((LBL1 ¬= LBL2) &
                    ((LBL1 > MAXLBL+2) & (LBL2 > MAXLBL+2)))
                   NEWLBL = 2
                $)
            ELSE IF ((INP3(L+1,2) ¬= 0) & (INP3(L-1,1) ¬= 0))
                $(
                LBL1 = INP3(L+1,2)
                LBL2 = INP3(L-1,1)
                IF ((LBL1 ¬= LBL2) &
                    ((LBL1 > MAXLBL+2) & (LBL2 > MAXLBL+2)))
                   NEWLBL = 2
                $)
            ELSE IF ((INP3(L+1,2) ¬= 0) & (INP3(L-1,3) ¬= 0))
                $(
                LBL1 = INP3(L+1,2)
                LBL2 = INP3(L-1,3)
                IF ((LBL1 ¬= LBL2) &
                    ((LBL1 > MAXLBL+2) & (LBL2 > MAXLBL+2)))
                   NEWLBL = 2
                $)

          IF (NEWLBL ¬= 0)
              OBUFF(L-1) = 1
          ELSE
              OBUFF(L-1) = 0
        $)
      CALL RWRITE(FDO, OBUFF, 1, I-1, JDENT, .WAIT, IEV, %9999)
      $)
#
#                          NORMAL RETURN
```

```
#
      CALL PPOP
      RETURN
#
#                                ABNORMAL RETURN
#
 9998 CONTINUE
#
#                                READ OR WRITE ERROR
#
#
 9999 CONTINUE
      RETURN 1
#
      END
```

```
*COST      CREATE A COST IAMGE FOR RDNET COMMAND

 VERSION:  A.01   DATE: 6-15-85     AUTHOR: J. J. KIM

 ACTION:   This command takes three input images as follows:
             The first input image: Line image which can be
                      created by the FLINE command.
             The second input image: Connected component image
                      whose background is labeled with zero
                      and every connected componet is labeled
                      with 1 thru the number of the components
                      consecutively.
             The third input image: Original sif image from which
                      the line image is originated.
           Then, this command assigns a value between 0 and 99 to
           any line pixel which is not the part of a connected
           component, the value of 100 to any non-line pixel, and
           the value of 100 + its label number to every line pixel
           which is the part of a connected component.

           The cost value between 0 and 99 is computed by the weights
           of the strength, angle difference, and gray-tone value.
           The weights are provided by the users.

SOURCE:    Disk, input file name

DESTINATION: Disk, output file name

QUESTION:  (1) Enter lower bound of angle difference(d=15, 0 to 90)
           The reply to this question should be the minimum angle
           difference for prorating the cost. Every pixel with
           the value below this minimum is assigned the best cost
           regarding angle difference which is 5.
           (2) Enter upper ound of angle difference(d=lower bound + 90,
```

lower bound to 180)
The reply to this question should be the maximum angle difference for prorating the cost. Every pixel with the value above this maximum is assigned the worst cost regarding angle difference which is 10.
(3) Enter mid gray-tone value (d=22500, 0 to 65000)
The reply to this question should be the median gray-tone value for the possible lines.
(4) Enter lower bound of gray-tone distance (d=2500, 0 to 10000)
The reply to this question should be the minimum distance of the gray-tone from the mid gray-tone of previous question. Every pixel with the value which differs from the mid gray-tone less than this bound will be assigned the best score.
(5) Enter upper bound of gray-tone distance
(d=lower bound + 10000, lower bound to 65000)
The reply to this question should be the maximum distance of the gray-tone from the mid gray-tone. Every pixel with the value which differs from the mid gray-tone more than this bound will be assigned the worst score.
(6) Enter lower bound of strength (d=250, 0 to 3000)
The reply to this question should be the minimum strength for prorating the cost. Every pixel with the value below this minimum will be assigned the worst score.
(7) Enter upper bound of strength (d=lower bound + 3000, lower bound to 25000)
The reply to this question should be the maximum strength for prorating the cost score. Every pixel with the value greater than this maximum will be assigned the best score regarding the line strength.

COMMAND STRING EXAMPLE:

```
COST COST.LIN,COST.LBL,COST.SIF > COST.CST

Enter the lower bound of angle difference,
(D=15, 0 to 90)   10
Enter the upper bound of angle difference,
(D=100, 10 to 180) 45
Enter the mid gray-tone value,
(D=22500, 10000 to 40000) 22500
Enter the lower bound of gray-tone distance,
(D=2500, 0 to 10000) 2500
Enter the upper bound of gray-tone distance,
(D=12500, 2500 to 65000) 5000
Enter the lower bound of line strength
(D=250, 0 to 3000) 250
Enter the upper bound of line strength,
(D=3250, 250 to 25000) 3250
```

Cost.lin is a line image created by the FLINE command, Cost.lbl is a connected component image which is created by screening the line image (by ROAD command), and connecting the screened line image (and possibly relabeling them), COST.sif is a sif image from which the line image is originated.

The resulting cost image is COST.CST.

ALGORITHM:

```
For every line pixel not being used as the part of a connected
component:
COST = (GA * HD)/FS - 1
GA   = 5 when the angle difference is less than the lower
                bound of angle difference user provides
     = 10 when the angle difference is greater than the
                upper bound of angle difference user provides
     = ((5*(angle difference - lower bound)/(upper -lower)) + 5
HD   = 2 when the gray-tone distance is less than the lower
                bound of gray-tone distance user provides
     = 10 when the gray-tone distance is greater than the
                upper bound of gray-tone distance user provides
     = ((8*(gray-tone distance - lower bound)/(upper-lower)) + 2
FS   = 1 when the line strength is less than the lower
                bound of line strength user provides
     = 9 when the line strength is greater than the upper
                bound of line strength user provides
     = ((9*(line strength - lower bound)/)upper - lower)) + 1
For every non-line pixel:
COST = 100
For every line pixel being used as the part of a connected
component:
COST = 100 + the label of the pixel
```

REFERENCE:

AUTHOR's master thesis chapter 4

```
$ !
$$ ! COST.RUN
$$ !
$$ !     MAKE A PARABOLA FACING UP
$$ !
$       MKBAR COST.IMG
        50
        V
        L
$       ARITHM COST.SUB < COST.IMG (S)
        25
        N
$       IMGPWR COST.PWR < COST.SUB
        2
        100
$ !
$$ !            GET THE CUBIC FACET PARAMETERS
$$ !
$       FACET2 COST.CUB < COST.PWR (W)
        9
```

```
            9
            3
            1.5
$ !
$$ !!             GET A LINE IMAGE
$$ !!
$           FLINE COST.LIN < COST.CUB
            2
            1.0
            1.0
            0
            10
            1
            6
$ !
$$ !!             GET A CONNECTED COMPONENT IMAGE
$$ !!
$           CONCM COST.CMP < COST.LIN
            2
            1
            3
$ !
$$ !!             RUN THE COST COMMAND TO CREATE A COST IMAGE
$$ !!
$           COST COST.CST < COST.LIN, COST.CMP, COST.PWR
            15
            45
            250
            50
            500
            10
            100
$ !
$$ !!             TO TEST THE RESULT WE WILL MAKE A TEST IMAGE IN WHICH
$$ !!             EVERY PIXEL OF THE CONNECTED COMPONENT IS ASSGINED 101 1
$$ !!             AND THE BACKGROUND IS ASSGINED 100.
$$ !!
$           TRSLD COST.TRS < COST.PWR
            0
$           EXSIF
            OPEN COST.TRS
            MID 18 0
            Y
            DONE
$           ARITHM COST2.TRS < COST.TRS (A)
            100
            N
$           EXSIF
            OPEN COST2.TRS
            MID 18 1
            Y
            DONE
$ !
$ !               THERE SHOULD BE NO DIFFERENCE
```

```
$ !
$$      RMS TT < COST2.TRS,COST.CST
$$ !
$       DELETE
        COST.IMG
        COST.SUB
        COST.PWR
        COST.CUB
        COST.LIN
        COST.CMP
        COST.CST
        COST.TRS
        COST2.TRS

$!
$       EXIT


#--DCOST      DRIVER OF COMMAND COST                                    MID
#
#  IDENTIFICATION
#
#       TITLE           DCOST
#       AUTHOR          JUNGWHAN KIM
#       VERSION         A.01
#       DATE            11-20-1984
#       LANGUAGE        RATFOR
#       SYSTEM          VAX 11/780
#
#  UPDATE
#
#       UPDATE          # 1
#       AUTHOR          JUNGWHAN KIM
#       DATE
#       PURPOSE
#
#  PURPOSE
#
#
#  ENTRY POINT
#
#       DROAD (WORK, *)
#
#  ARGUMENT LISTING
#
#       WORK    INT.ARR         WORKING STORAGE ARRAY
#       *       INT             ALTERNATE ERROR 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)
#       OSGTNM      GET NAME FROM FILE DESCRIPTOR        (GIPSY OS KERNEL)
#       RDKINL      INITIALIZE AND ACCESS A STANDARD IMAGE
#                   FILE.                                (GIPSY PRIMITIVE)
#       OSALOC      CHECK AND EXTEND USER WORK SPACE     (GIPSY OS KERNEL)
#       RNGETI      GET AN INTEGER WITHIN A RANGE (WITH
#                   PROMPT).                             (GIPSY PRIMITIVE)
#       RNGETR      GET AN REAL WITHIN A RANGE (WITH PROMPT).
#                                                        (GIPSY PRIMITIVE)
#       GETWP       GET WORK ARRAY POINTER.              (GIPSY PRIMITIVE)
#       NCOST      THIS IS THE NUMBER CRUNCHER ROUTINE
#                   FOR THE COMMAND "ROAD".              (GIPSY ROUTINE)
#       PPOP        POP PROGRAM NAME FROM THE ERROR STACK
#                                                        (GIPSY PRIMITIVE)
#       CLOSE       CLOSE A FILE.                        (GIPSY PRIMITIVE)
#
#
#*****************************************************************
#
#
#
#
#       INCLUDE MACA1
#
#       SUBROUTINE DCOST (WORK, *)
#
#       IMPLICIT INTEGER (A-Z)
#
#       INCLUDE GIPCOM
#       INCLUDE ERROR
#       INCLUDE TTCOM
#
#       CHARACTER NAME (.FILENAMELENGTH)
#
#       INTEGER IDENT1(.IDLENGTH)
#       INTEGER IDENT2(.IDLENGTH)
#       INTEGER IDENT3(.IDLENGTH)
#       INTEGER WORK (.ARB)
#
#
#
#
#       EQUIVALENCE (NPPL, IDENT1(.IDNPPL)), (NLINS, IDENT1(.IDNLINS))
#       EQUIVALENCE (NCOLS, IDENT1(.IDNCOLS)), (NROWS, IDENT1(.IDNROWS))
#       EQUIVALENCE (NBND, IDENT1(.IDNBNDS)), (MODE, IDENT1(.IDMODE))
#       EQUIVALENCE (NSBND, IDENT1(.IDNSBNDS)), (MAXLBL, IDENT2(.IDMAX))
#
#
```

```
#                          PUSH PROGRAM NAME ONTO ERROR STACK
#
      CALL PPUSH ("DCOST")
#
#
#                          SET UP INPUT FILE
#
      CALL RDKINL (FDI1, IDENT1, .READONLY, IEV, %9999)
      CALL CLOSE(FDI1)
      CALL RDKINL (FDI2, IDENT2, .READONLY, IEV, %9999)
      CALL CLOSE(FDI2)
      CALL RDKINL (FDI3, IDENT3, .READONLY, IEV, %9999)
      CALL CLOSE(FDI3)
#
#
#
#                          CHECK THE INPUT FILE
#
      IF (NPPL ¬= NCOLS ! NROWS ¬= 1) GO TO 9010
      IF (NSBND == 0) GO TO 9020
      IF (MODE ¬= .INTMODE) GO TO 9000
      IF (NBND < 3) GO TO 9020
#
#
#
#                          SET UP BUFFERS
#
      NXT = 1
      ST    = GETWP (NXT, .INTMODE, NPPL)
      AN    = GETWP (NXT, .INTMODE, 3*(NPPL+2))
      GR    = GETWP (NXT, .INTMODE, NPPL)
      MAP   = GETWP (NXT, .INTMODE, NPPL)
      AD    = GETWP (NXT, .INTMODE, NPPL)
      RV    = GETWP (NXT, .INTMODE, NPPL)
      PCOST = GETWP (NXT, .INTMODE, NPPL)
#
#                          ALLOCATE THAT SPACE
#
      IF  (.OK ¬= OSALOC (NXT))
         GO TO 9030
#
#                          GET THE USER PARAMETERS
#
      CALL RNGETI("Enter lower bound of angle difference", 0,
                  55, 15, LA, IEV, %9999)
      CALL RNGETI("Enter upper bound of angle difference", LA,
                  180, LA + 45, UA, IEV,%9999)
      CALL RNGETI("Enter mid gray-tone value", 0, 40000,
                  22500, TRSGR, IEV, %9999)
      CALL RNGETI("Enter lower bound of gray-tone distance", 0,
                  10000, 2500, LD,IEV,%9999)
#
      CALL RNGETI("Enter upper bound of gray-tone distance", LD,
                  65000, LD + 10000, UD, IEV,%9999)
```

```
      CALL RNGETI("Enter lower bound of line strength", 0,
                  3000, 250, LS, IEV,%9999)
      CALL RNGETI("Enter upper bound of line strength", LS,
                  25000, LS + 3000, US, IEV, %9999)
#
#                                CALL THE NUMBER CRUNCHER
#
      NPPL2 = NPPL + 2
      CALL NCOST (FDI1, FDI2, FDI3, FDO1, WORK(ST), WORK(AN),
                  WORK(GR), WORK(MAP), WORK(AD), WORK(RV), WORK(PCOST),
                  LA, UA, TRSGR, LD, UD, LS, US,
                  NPPL, NPPL2, IEV, %9998)
#
#                                NORMAL RETURN
#
      CALL CLOSE (FDO1)
      CALL CLOSE (FDI1)
      CALL CLOSE (FDI2)
      CALL CLOSE (FDI3)
      CALL PPOP
      RETURN
#
#                                ABNORMAL CONDITIONS
#
 9000 CONTINUE
#
#                                NOT AN INTEGER FILE
#
      IEV = -2012
      GO TO 9998
#
 9010 CONTINUE
#
#                                NOT IN LINE FORMAT
#
      IEV = -5001
      GO TO 9998
#
 9020 CONTINUE
#
#                                BANDS SPECIFIED INCORRECTLY
#
      IEV = -5018
      GO TO 9998
#
 9030 CONTINUE
#
#                                NOT ENOUGH WORK SPACE
#
      IEV = -5010
      GO TO 9999
#
 9998 CONTINUE
#
#                                ERROR IN SUBPROGRAM
```

```
#
#
#                                   ABNORMAL RETURN
#
 9999   CONTINUE
        CALL CLOSE (FDO1)
        CALL CLOSE (FDI1)
        CALL CLOSE (FDI2)
        CALL CLOSE (FDI3)
        RETURN 1
        END


#--NCOST        NUMBER CRUNCHER FOR THE COST COMMAND                    MID
#
#  IDENTIFICATION
#
#       TITLE           NCOST
#       AUTHOR          JUNGWHAN KIM
#       VERSION         A.01
#       DATE            11-20-1984
#       LANGUAGE        RATFOR
#       SYSTEM          VAX 11/780
#
#  UPDATE
#       UPDATE          # 1
#       AUTHOR          JUNGWHAN KIM
#       DATE
#       PURPOSE
#
#  PURPOSE
#
#
#  ENTRY POINT
#
#       NCOST (FDI1, FDI2, FDI3, FDO, ST, AN, GR, MAP,
#              AD, RV, PCOST, LA, UA, TRSGR, LD, UD, LS, US,
#              NPPL, NPPL2, IEV, *)
#
#  ARGUMENT LISTING
#
#       FDI1    CHR.ARR         INPUT FILE DESCRIPTOR
#       FDI2    CHR.ARR         INPUT FILE DESCRIPTOR
#       FDI3    CHR.ARR         INPUT FILE DESCRIPTOR
#       FDO     CHR.ARR         OUTPUT FILE DESCRIPTOR
#       ST      INT.ARR         INPUT BUFFER
#       AN      INT.ARR         INPUT BUFFER
#       GR      INT.ARR         INPUT BUFFER
#       MAP     INT.ARR         I/O BUFFER
#       AD      INT.ARR         WORK BUFFER
#       RV      INT.ARR         INPUT BUFFER
```

```
#       PCOST   INT.ARR         OUTPUT BUFFER
#       TCOST   INT.ARR         OUTPUT BUFFER
#       NPPL    INT             NPPL OF INPUT FILE
#       IEV     INT             INTEGER EVENT VARIABLE
#       *       INT             ALTERNATE ERROR RETURN
#
#  INCLUDE FILES/COMMONS
#
#       MACA1     INCLUDE     GIPSY GENERAL SYMBOL DEFINITIONS
#       DSCAX     INCLUDE     TYPE CODES FOR AUXILLIARY DATA IN DE
#                             SCRIPTOR RECORDS.
#       TTCOM     COMMON      FD'S FOR TERMINAL AND RUNFILE IO
#
#  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)
#       COPYDS      COPY DESCRIPTOR RECORDS FROM TEMP FILE.
#                                                       (GIPSY PRIMITIVE)
#       GDSCAX      GET AUXILIARY DATA FROM DESCRIPTOR
#                   RECORDS                             (GIPSY PRIMITIVE)
#       AREAD       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)
#
#*****************************************************************
#
#
#
#       INCLUDE MACA1
#
        SUBROUTINE NCOST (FDI1, FDI2, FDI3, FDO1, ST, AN, GR,
                          MAP, AD,  RV, PCOST, LA, UA, TRSGR, LD, UD,
                          LS, US,
                          NPPL, NPPL2, IEV, *)
#
        IMPLICIT INTEGER (A-Z)
#
        INCLUDE DSCAX
        INCLUDE TTCOM
        CHARACTER FDI1 (.FDLENGTH)
        CHARACTER FDI2 (.FDLENGTH)
        CHARACTER FDI3 (.FDLENGTH)
        CHARACTER FDO1 (.FDLENGTH)
#
#
```

```
      INTEGER IDENT1(.IDLENGTH), IDENT2(.IDLENGTH), IDENT3(.IDLENGTH)
      INTEGER JDENT(.IDLENGTH)
      INTEGER ST(NPPL), AN(NPPL2, 3), GR(NPPL), MAP(NPPL)
      INTEGER AD(NPPL), RV(NPPL), PCOST(NPPL)
#
      EQUIVALENCE (NLIN, IDENT1(.IDNLINS))
#
#
#
#                          PUSH PROGRAM NAME ONTO ERROR STACK
      CALL PPUSH("NCOST")
#
#                          OPEN INPUT FILES
#
#                          COPY DESCRIPTOR RECORDS FROM INPUT
#                          IMAGE, ROUTINE NAME, AND PARAMETRS
#                          TO TEMPORARY SEQUENTIAL FILE.
#                          OPEN INPUT FILES
#
      CALL CPYIDR (FDI1, IDENT1, .OPNTMP, IEV, %9998)
      CALL CPYIDR (FDI2, IDENT2, .NOOPNTMP, IEV, %9998)
      CALL CPYIDR (FDI3, IDENT3, .NOOPNTMP, IEV, %9998)
#
      CALL DSCNAM ("COST", IEV, %9998)
#
#
#                          OPEN OUTPUT IMAGE AND COPY TO IT THE
#                          DESCRIPTOR RECORDS FROM TEMPORARY FILE
#
      DO I = 1, .IDLENGTH
        JDENT(I) = 0
#
      JDENT(.IDNPPL   ) = NPPL
      JDENT(.IDNLINS  ) = NLIN
      JDENT(.IDNCOLS  ) = NPPL
      JDENT(.IDNROWS  ) = 1
      JDENT(.IDNBNDS)   = 1
      JDENT(.IDNSBNDS ) = 1
#
      CALL COPYDS (FDO1, JDENT, IEV, %9998)
#
      DO I = 1, NLIN
        $(
        CALL RREAD (FDI2, MAP, 1, I, IDENT2, .WAIT, IEV, %9999)
        CALL RREAD (FDI3, GR, 1, I, IDENT3, .WAIT, IEV, %9999 )
        CALL RREAD (FDI1, ST,  1, I, IDENT1, .WAIT, IEV, %9999 )
        DO K = 1, 3
            CALL AREAD (FDI1, AN(1,K), NPPL2, 1, 1, 1, 1, 2,
                        I+K-1, IDENT1, .WAIT, IEV, %9999)
        CALL RREAD (FDI1, RV, 3, I, IDENT1, .WAIT, IEV, %9999)
#
#
#                          COMPUTE THE ANGLE DIFFERENCE FOR
```

```
#                                EACH PIXEL
#
#
          DO  J = 2, NPPL + 1
            IF      ((AN(J,2) <= 22) ! ((AN(J,2) >= 158)&(AN(J,2) ¬= 360)))
              AD(J-1) = MAX(ABS(AN(J,2)-AN(J-1,2)), ABS(AN(J,2)-AN(J+1,2)))
            ELSE IF ((AN(J,2) >= 23) & (AN(J,2) <= 67))
              AD(J-1) = MAX(ABS(AN(J,2)-AN(J-1,1)), ABS(AN(J,2)-AN(J+1,3)))
            ELSE IF ((AN(J,2) >= 68) & (AN(J,2) <= 112))
              AD(J-1) = MAX(ABS(AN(J,2)-AN(J,1)), ABS(AN(J,2)-AN(J,3)))
            ELSE IF ((AN(J,2) >= 113) & (AN(J,2) <= 157))
              AD(J-1) = MAX(ABS(AN(J,2)-AN(J+1,1)), ABS(AN(J,2)-AN(J-1,3)))
            ELSE
              AD(J-1) = AN(J,2)
#
#
#                          COMPUTE THE COST FOR EACH LINE PIXEL,
#                          ASSIGN THE COST OF 100 TO EACH NON-LINE PIXEL,
#                          ASSIGN THE COST OF 100 + ITS LABEL NUMBER TO
#                          EACH LINE PIXEL WHICH IS THE PART OF A LINE
#                          SEGMENT. NOTE THAT A LINE SEGMENT IS LABELED
#                          AN UNIQUE NUMBER OTHER THAN 0 IN THE SECOND
#                          INPUT FILE.
#
#
          DO J = 1, NPPL
            $(
            IF (MAP(J) == 0)
              $(
              IF (RV(J) ¬= 0)
                $(
                IF (AD(J) <= LA)
                    GA = 5
                ELSE IF (AD(J) >= UA)
                    GA = 10
                ELSE
                    GA =  ((5*(AD(J)-LA))/(UA-LA)) + 5
                DEL = ABS(TRSGR - GR(J))
                IF (DEL <= LD)
                    HD = 2
                ELSE IF (DEL >= UD)
                    HD = 10
                ELSE
                    HD = ((8*(DEL-LD))/(UD-LD)) + 2
                IF (ST(J) <= LS)
                    FS = 1
                ELSE IF (ST(J) >= US)
                    FS = 10
                ELSE
                    FS = ((9*(ST(J)-LS))/(US-LS)) + 1
                PCOST(J) = (GA*HD)/FS - 1
                $)
              ELSE
                PCOST(J) = 100
              $)
```

```
            ELSE
              PCOST(J) = MAP(J) + 100
            $)
          CALL RWRITE (FDO1, PCOST, 1, I, JDENT, .WAIT, IEV, %9999)
          $)
#
#                                NORMAL RETURN
#
        CALL PPOP
        RETURN
#
#                                ABNORMAL RETURN
#
 9998   CONTINUE
#
#                                READ OR WRITE ERROR
#
#
 9999   CONTINUE
        RETURN 1
#
        END
```

```
*MCPATH    MINIMUM COST PATH FINDER

 VERSION:  A.01   DATE: 06-15-85              AUTHOR: J. J. KIM

 ACTION:   This command takes a one-band cost image of which
           pixels represent three different types depending on
           their values as follows:

           pixel value 0 -- (NPCP-1) : A path-candidate pixel
                                       which is not used for an object,
                                       but will be possibly used for a
                                       a path between two objects.
                                       cost = pixel value + 1
           pixel value NPCP  : A non-path-candidate pixel which
                                       will be penalized by a high
                                       cost. The NPCP value and the
                                       high cost PENALTY value are to
                                       be provided by user.
                                       cost = PENALTY
           pixel value (NPCP+1) -- MAX: An object pixel which
                                       is a part of an object.
                                       cost = 0

           Then, this command connects all the objects
           with the minimum cost paths using the dynamic
           programming algorithm.
```

```
SOURCE:  Disk, input file name

DESTINATION: Disk, output file name

QUESTION: (1) Enter the value assigned to a non-path-candidate
              pixel in the cost image (d=100, 0 to 1000000)
          The reply to this question is consistent with
          an input cost image.
          (2) Enter penalty cost for a non-pathe-candidate
              pixel (d=NPCP*10, NPCP+1 to NPCP*20)
          The reply to this question should be sufficiently
          high so that when linking objects non-path-candidate
          pixels will be least likely chosen.

COMMAND STRING EXAMPLE:

          MCPATH MCPATH.NET < MCPATH.CST

          Enter the value assigned to a non-path-candidate
          pixel in the cost image (d=100, 0 to 1000000) 100

          Enter the penalty cost for a non-path-candidate
          pixel (D=1000, 101 to 2000) 1000

          MCPATH.CST is a cost image in which the pixels
          with the values of 100 represent non-path-candidate
          lixels, the pixels with the value of 0 thru 99
          represent path-candidate pixels not being used as
          a part of an object, and the pixels with the values
          of 101 or greater represent object pixels being used
          as a part of objects.
          The created linked object image is MCPATH.NET in
          which all the objects are connected as a minimum
          cost spanning tree. Note that a non-line pixel is
          penalized by assigning the cost of 1000, which means
          that 9 line pixels with the highest cost (100) are
          prefered to 1 non-line pixel.

REFERENCES:
          Fischler's paper on CGIP 15, 1981
          Author's master thesis chapter 4


$ !
$ ! MCPATH.RUN
$ !
$ !    MAKE A COST IMAGE USING PARABOLA FACING UP
$ !
$      MKBAR MCPATH.IMG (A)
       20
       20
```

```
        20
        V
        L
$       ARITHM MCPATH.SUB < MCPATH.IMG (S)
        10
        N
$$ !
$$ !!           FOR A COST IMAGE
$$ !!
$       IMGPWR MCPATH.CST < MCPATH.SUB
        2
        1
$$ !
$$ !            FOR A TEST IAMGE
$       IMGPWR MCPATH.TST < MCPATH.SUB
        2
        1
$$ !
$$ !!           TO MAKE THE PROPER COST IMAGE FOR THIS COMMAND
$$ !!           WE WILL MAKE THE TWO END POINT OF THE 11TH COLUMN
$$ !!           TAKE THE VALUE 101 AND 102.
$$ !!
$       EXSIF MCPATH.CST
        PROT OFF
        MID 16 102
        MOD 1 11
        101
        S
        MOD 20 11
        102
        S
        PROT ON
        DONE
        Y
$$ !
$$ !!           NOW RUN THE MCPATH COMMAND TO THIS COST IMAGE
$$ !!
$       MCPATH MCPATH.NET <MCPATH.CST
        100
        1000
$$ !
$$ !!           TO TEST THE RESULT WE WILL MAKE A PROPER TEST
$$ !!           IMAGE IN WHICH EACH PIXEL OF THE LINE TAKE A
$$ !!           VALUE OF 2
$$ !!
$       TRSLD MCPATH.TRS < MCPATH.TST
        0
$       EXSIF
        OPEN MCPATH.TRS
        MID 18 0
        Y
        DONE
$       ARITHM MCPATH2.TRS < MCPATH.TRS (M)
        2
        N
```

```
$       EXSIF
        OPEN MCPATH2.TRS
        MID 18 1
        Y
        DONE
$ !
$$ !
$$ !            THERE SHOULD BE A DIFFERENCE OF 0
$$ !
$$ !
$$ !    RMS TT < MCPATH2.TRS,MCPATH.NET
$$ !
$       DELETE
        MCPATH.IMG
        MCPATH.SUB
        MCPATH.CST
        MCPATH.TST
        MCPATH.NET
        MCPATH.TRS
        MCPATH2.TRS

$!
$       EXIT


#--DMCP      DRIVER OF COMMAND MCPATH                                MID
#
#  IDENTIFICATION
#
#       TITLE           DMCP
#       AUTHOR          JUNGWHAN KIM
#       VERSION         A.01
#       DATE            12-01-1984
#       LANGUAGE        RATFOR
#       SYSTEM          VAX 11/780
#
#  UPDATE
#
#       UPDATE          # 1
#       AUTHOR          JUNGWHAN KIM
#       DATE
#       PURPOSE
#
#  PURPOSE
#
#
#  ENTRY POINT
#
#       DMCP (WORK, *)
#
#  ARGUMENT LISTING
```

```
#
#       WORK    INT.ARR         WORKING STORAGE ARRAY
#       *       INT             ALTERNATE ERROR 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)
#       OSGTNM      GET NAME FROM FILE DESCRIPTOR       (GIPSY OS KERNEL)
#       RDKINL      INITIALIZE AND ACCESS A STANDARD IMAGE
#                   FILE.                               (GIPSY PRIMITIVE)
#       OSALOC      CHECK AND EXTEND USER WORK SPACE    (GIPSY OS KERNEL)
#       RNGETI      GET AN INTEGER WITHIN A RANGE (WITH
#                   PROMPT).                            (GIPSY PRIMITIVE)
#       RNGETR      GET AN REAL WITHIN A RANGE (WITH PROMPT).
#                                                       (GIPSY PRIMITIVE)
#       GETWP       GET WORK ARRAY POINTER.             (GIPSY PRIMITIVE)
#       NMCP      THIS IS THE NUMBER CRUNCHER ROUTINE
#                   FOR THE COMMAND "MCPATH".            (GIPSY ROUTINE)
#       PPOP        POP PROGRAM NAME FROM THE ERROR STACK
#                                                       (GIPSY PRIMITIVE)
#       CLOSE       CLOSE A FILE.                       (GIPSY PRIMITIVE)
#
#
#*****************************************************************
#
#
#
#
        INCLUDE MACA1
#
        SUBROUTINE DMCP (WORK, *)
#
        IMPLICIT INTEGER (A-Z)
#
        INCLUDE GIPCOM
        INCLUDE ERROR
        INCLUDE TTCOM
#
        CHARACTER NAME (.FILENAMELENGTH)
#
        INTEGER IDENT1(.IDLENGTH)
        INTEGER WORK (.ARB)
#
#
#
#
        EQUIVALENCE (NPPL, IDENT1(.IDNPPL)), (NLIN, IDENT1(.IDNLINS))
```

```
      EQUIVALENCE (NCOLS, IDENT1(.IDNCOLS)), (NROWS, IDENT1(.IDNROWS))
      EQUIVALENCE (NBND, IDENT1(.IDNBNDS)), (MODE, IDENT1(.IDMODE))
      EQUIVALENCE (NSBND, IDENT1(.IDNSBNDS)), (MAXLBL, IDENT1(.IDMAX))
#
#
#                          PUSH PROGRAM NAME ONTO ERROR STACK
#
      CALL PPUSH ("DMCP")
#
#
#                          SET UP INPUT FILE
#
      CALL RDKINL (FDI1, IDENT1, .READONLY, IEV, %9999)
      CALL CLOSE(FDI1)
#
#
#                          CHECK THE INPUT FILE
#
      IF (NPPL ¬= NCOLS ! NROWS ¬= 1) GO TO 9010
#
#
#                          SET UP BUFFERS
#
      NXT = 1
      MAP   = GETWP (NXT, .INTMODE, NPPL+2)
      C     = GETWP (NXT, .INTMODE, NPPL+2)
      T     = GETWP (NXT, .INTMODE, NPPL+2)
      TS    = GETWP (NXT, .INTMODE, NPPL+2)
      TT    = GETWP (NXT, .INTMODE, NPPL)
      CC    = GETWP (NXT, .INTMODE, NPPL)
      CCC   = GETWP (NXT, .INTMODE, (NPPL+2)*3)
      TTT   = GETWP (NXT, .INTMODE, (NPPL+2)*3)
      MM    = GETWP (NXT, .INTMODE, NPPL)
      MMM   = GETWP (NXT, .INTMODE, (NPPL+2)*3)
#
#                          ALLOCATE THAT SPACE
#
      IF  (.OK ¬= OSALOC (NXT))
         GO TO 9030
#
#                          GET THE USER PARAMETERS
#
      IF (.OK ¬= PUTF (FDRUNO,
            "Enter the value assigned to%N"))
         GO TO 9999
      CALL RNGETI("a non-path-candidate pixel in the cost image", 0,
                  1000000, 100, NPCP, IEV, %9999)
      IF (.OK ¬= PUTF (FDRUNO,
            "Enter penalty cost for%N"))
         GO TO 9999
      CALL RNGETI("a non-path-candidate pixel", NPCP + 1,
                  NPCP * 20, NPCP * 10, PENALTY, IEV, %9999)
```

```
#
#                                   CALL THE NUMBER CRUNCHER
#
      NPPL2 = NPPL + 2
      CALL NMCP (FDI1, FDO1, WORK(MAP), WORK(C), WORK(T),
                 WORK(TS), WORK(TT), WORK(CC), WORK(CCC), WORK(TTT),
                 WORK(MM), WORK(MMM), MAXLBL, NLIN,
                 NPPL, NPPL2, PENALTY, NPCP, IEV, %9998)
#
#                                   NORMAL RETURN
#
      CALL CLOSE (FDO1)
      CALL CLOSE (FDI1)
      CALL PPOP
      RETURN
#
#                                   ABNORMAL CONDITIONS
#
 9000 CONTINUE
#
#                                   NOT AN INTEGER FILE
#
      IEV = -2012
      GO TO 9998
#
 9010 CONTINUE
#
#                                   NOT IN LINE FORMAT
#
      IEV = -5001
      GO TO 9998
#
 9020 CONTINUE
#
#                                   BANDS SPECIFIED INCORRECTLY
#
      IEV = -5018
      GO TO 9998
#
 9030 CONTINUE
#
#                                   NOT ENOUGH WORK SPACE
#
      IEV = -5010
      GO TO 9999
#
 9998 CONTINUE
#
#                                   ERROR IN SUBPROGRAM
#
#
#                                   ABNORMAL RETURN
#
 9999 CONTINUE
      CALL CLOSE (FDO1)
```

```
      CALL CLOSE (FDI1)
      RETURN 1
      END


#--NMCP       NUMBER CRUNCHER FOR THE MCPATH  COMMAND                MID
#
#  IDENTIFICATION
#
#      TITLE           NMCP
#      AUTHOR          JUNGWHAN KIM
#      VERSION         A.01
#      DATE            12-10-1984
#      LANGUAGE        RATFOR
#      SYSTEM          VAX 11/780
#
#  UPDATE
#      UPDATE          # 1
#      AUTHOR          JUNGWHAN KIM
#      DATE
#      PURPOSE
#
#  PURPOSE
#
#
#  ENTRY POINT
#
#      NMCP  (FDI1 FDO1, MAP, C, T, CC, TTT, MM, MAXLBL,
#                NPPL, NPPL2, PENALTY, NPCP, IEV, *)
#
#  ARGUMENT LISTING
#
#      FDI     CHR.ARR        INPUT FILE DESCRIPTOR
#      FDO     CHR.ARR        OUTPUT FILE DESCRIPTOR
#      MAP     INT.ARR        I/O BUFFER
#      C       INT.ARR        I/O BUFFER
#      T       INT.ARR        I/O BUFFER
#      TS      INT.ARR        I/O BUFFER
#      TT      INT.ARR        I/O BUFFER
#      CC      INT.ARR        I/O BUFFER
#      CCC     INT.ARR        I/O BUFFER
#      TTT     INT.ARR        I/O BUFFER
#      MM      INT.ARR        I/O BUFFER
#      NPPL    INT            NPPL OF INPUT FILE
#      NPPL2   INT            NPPL + 2
#      IEV     INT            INTEGER EVENT VARIABLE
#      *       INT            ALTERNATE ERROR RETURN
#
#  INCLUDE FILES/COMMONS
#
#      MACA1     INCLUDE     GIPSY GENERAL SYMBOL DEFINITIONS
```

```
#       DSCAX     INCLUDE     TYPE CODES FOR AUXILLIARY DATA IN DE
#                             SCRIPTOR RECORDS.
#       TTCOM     COMMON      FD'S FOR TERMINAL AND RUNFILE IO
#
#  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)
#       COPYDS      COPY DESCRIPTOR RECORDS FROM TEMP FILE.
#                                                        (GIPSY PRIMITIVE)
#       GDSCAX      GET AUXILIARY DATA FROM DESCRIPTOR
#                   RECORDS                              (GIPSY PRIMITIVE)
#       AREAD       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)
#
#
#*********************************************************************
#
#
#
#       INCLUDE MACA1
#
#       SUBROUTINE NMCP (FDI1, FDO1, MAP, C, T, TS, TT, CC, CCC, TTT, MM,
#                  MMM, MAXLBL, NLIN, NPPL, NPPL2, PENALTY, NPCP, IEV, *)
#
#       IMPLICIT INTEGER (A-Z)
#
#       INCLUDE DSCAX
#       INCLUDE TTCOM
#       CHARACTER FDI1 (.FDLENGTH)
#       CHARACTER FDO1 (.FDLENGTH)
#
#
#       INTEGER IDENT(.IDLENGTH)
#       INTEGER JDENT(.IDLENGTH)
#       INTEGER MAP(NPPL2), C(NPPL2), T(NPPL2), TS(NPPL2), CCC(NPPL2, 3)
#       INTEGER TT(NPPL), CC(NPPL), TTT(NPPL2,3), MM(NPPL), MMM(NPPL2,3)
#
#       LOGICAL UPDATE, LUDT, FIRST, TOPDOWN, NOTYET
#
#       DATA INF/999999/
#
#       EQUIVALENCE (NLIN, IDENT(.IDNLINS))
#
#
```

```
#
#                                PUSH PROGRAM NAME ONTO ERROR STACK
#
      CALL PPUSH("NMCP")
#
#                                OPEN INPUT FILES
#
#                                COPY DESCRIPTOR RECORDS FROM INPUT
#                                IMAGE, ROUTINE NAME, AND PARAMETRS
#                                TO TEMPORARY SEQUENTIAL FILE.
#                                OPEN INPUT FILES
#
      CALL CPYIDR (FDI1, IDENT, .OPNTMP, IEV, %9998)
#
      CALL DSCNAM ("MCPATH", IEV, %9998)
#
#                                OPEN OUTPUT IMAGE AND COPY TO IT THE
#                                DESCRIPTOR RECORDS FROM TEMPORARY FILE
#
      DO I = 1, .IDLENGTH
        JDENT(I) = 0
#
      JDENT(.IDNPPL    ) = NPPL
      JDENT(.IDNLINS   ) = NLIN
      JDENT(.IDNCOLS   ) = NPPL
      JDENT(.IDNROWS   ) = 1
      JDENT(.IDNBNDS)    = 3
      JDENT(.IDNSBNDS  ) = 1
      JDENT(.IDNBITS)    = 24
#
      CALL COPYDS (FDO1, JDENT, IEV, %9998)
#
#                                INITIALIZE THREE OUTPUT BANDS
#                                ACCORDING TO THE INPUT VALUES
#                                OF COST IMAGE AS FOLLOWS:
#                                 0 -- (NPCP-1) : COST
#                                 NPCP    : NON-PATH-CANDIDATE PIXEL
#                                 (NPCP+1) -- MAX : AN OBJECT PIXEL
#
      DO ROW = 1, NLIN
        $(
        CALL RREAD(FDI1, CC, 1, ROW, IDENT, .WAIT, IEV, %9999)
        DO COL = 1, NPPL
           $(
           IF (CC(COL) == NPCP)
#
#                                THIS PIXEL IS NOT A PATH-CANDIDATE
#                                PIXEL AND ASSIGNED A HIGH PENALTY COST
#
              $(
              CC(COL) = PENALTY
```

```
            MM(COL) = 0
            $)
          ELSE IF (CC(COL) > NPCP)
#
#                                THIS IS A PIXEL OF AN OBJECT WHICH
#                                WILL BE CONNECTED TO ANOTHER OBJECT BY
#                                THIS DYNAMIC PROGRAMMING.
#
#
            $(
            MM(COL) = CC(COL) - NPCP
            CC(COL) = 0
            $)
          ELSE
#
#                                THIS IS A CANDIDATE-PATH PIXEL WHICH
#                                MAY BE USED AS A PART OF PATH BETWEEN
#                                TWO OBJECT.
#
#
            $(
            CC(COL) = CC(COL) + 1
            MM(COL) = 0
            $)
          TT(COL) = INF
          $)
        CALL RWRITE(FDO1, CC, 1, ROW, JDENT, .WAIT, IEV, %9999)
        CALL RWRITE(FDO1, TT, 2, ROW, JDENT, .WAIT, IEV, %9999)
        CALL RWRITE(FDO1, MM, 3, ROW, JDENT, .WAIT, IEV, %9999)
        $)
      LSEGNO = MAXLBL - (NPCP + 1)
      DO SEGNO = 1, LSEGNO
        $(
        MINCST = INF
        MINUDT = INF/2
        TOPDOWN = .FALSE.
        FIRST   = .TRUE.
        UPDATE  = .TRUE.
        NOTYET  = .TRUE.
        GROW    = 1
        GCOL    = 1
        WHILE (UPDATE & (MINUDT < MINCST))
          $(
          UPDATE = .FALSE.
          MINUDT = INF
          IF (TOPDOWN) TOPDOWN = .FALSE.
          ELSE         TOPDOWN = .TRUE.
#
          DO ROW = 2, NLIN + 1
#
#
#                                ADJUST LOOP VARIANTS ACCORDING TO
#                                WHETHER TOP-DOWN OR NOT(BOTTOM-UP)
#
```

```
#
                $(
                LUDT = .FALSE.
                IF (TOPDOWN)
#
#                       TOPDOWN PROCESS: ADJUST LOOP VARIANT
#                                        FOR ROW
#
                  $(
                  L = ROW
                  LS = L - 1
                  $)
                ELSE
#
#                       BOTTOMUP PROCESS: ADJUST LOOP VARIANT
#                                         FOR ROW
#
                  $(
                  L = NLIN - ROW + 3
                  LS = L + 1
                  $)
#
#
                CALL AREAD (FD01, MAP, NPPL2, 1, 1, 1, 1, 3,
                            L, JDENT, .WAIT, IEV, %9999)
                CALL AREAD (FD01, C,   NPPL2, 1, 1, 1, 1, 1,
                            L, JDENT, .WAIT, IEV, %9999)
                CALL AREAD (FD01, T,   NPPL2, 1, 1, 1, 1, 2,
                            L, JDENT, .WAIT, IEV, %9999)
                CALL AREAD (FD01, TS,  NPPL2, 1, 1, 1, 1, 2,
                            LS,JDENT, .WAIT, IEV, %9999)
                DO COL = 2, NPPL + 1
                   $(
                   IF (TOPDOWN)
#
#                       TOPDOWN PROCESS: ADJUST LOOP VARIANT
#                                        FOR LEFT-TO-RIGHT
#
                     $(
                     I = COL
                     BI = I - 1
                     AI = I + 1
                     $)
                   ELSE
#
#                       BOTTOMUP PROCESS: ADJUST LOOP VARIANT
#                                         FOR RIGHT-TO-LEFT
#
                     $(
                     I = NPPL - COL + 3
                     BI = I + 1
                     AI = I - 1
                     $)
#
#
```

```
#
               IF (COL == 2) TBI = T(BI)
               IF (MAP(I) ¬= SEGNO)
                 $(
                 TEMP = T(I)
                 T(I) = MIN(MIN(TS(BI), TS(I), TS(AI), TBI) + C(I), TEMP)
                 IF (TEMP > T(I))
                   $(
                   UPDATE = .TRUE.
                   LUDT = .TRUE.
                   IF (MINUDT > T(I)) MINUDT = T(I)
                   $)
                 IF ((MAP(I) >SEGNO) & (MINCST > T(I)))
                   $(
                   MINCST = T(I)
                   GROW   = L - 1
                   GCOL   = I - 1
                   $)
                 $)
               ELSE
                 T(I) = 0
               TBI = T(I)
               $)
#
#
#
#                          AGAIN, SAME PROCESS IN REVERSE DIRECTION
#                          ON THE SAME CURRENT ROW
#
#
#
            DO COL = 2, NPPL + 1
               $(
               IF (TOPDOWN)
#
#
#                          TOPDOWN PROCESS: ADJUST LOOP VARIANT
#                                           FOR RIGHT-TO-LEFT
#
#                          AFTER THIS, TOPDOWN LEFT-TO-RIGHT AND
#                          RIGHT-TO-LEFT PROCESS FOR THE CURRENT
#                          ROW IS FINISHED.
#
                 $(
                 I = NPPL - COL + 3
                 BI = I + 1
                 $)
               ELSE
#
#                          BOTTOMUP PROCESS: ADJUST LOOP VARIANT
#                                            FOR LEFT-TO-RIGHT
#
#                          AFTER THIS, BOTTOMUP RIGHT-TO-LEFT AND
#                          LEFT-TO-RIGHT PROCESS FOR THE CURRENT
#                          ROW IS FINISHED.
```

```
#
#
                      $(
                      I = COL
                      BI = I - 1
                      $)
#
#
#
                    IF (COL == 2) TBI = T(BI)
                    TEMP = T(I)
                    T(I) = MIN(TBI + C(I), TEMP)
                    IF (TEMP > T(I))
                      $(
                      UPDATE = .TRUE.
                      LUDT = .TRUE.
                      IF (MINUDT > T(I))  MINUDT = T(I)
                      $)
                    IF ((MAP(I) > SEGNO) & (MINCST > T(I)))
                      $(
                      MINCST = T(I)
                      GROW = L - 1
                      GCOL = I - 1
                      $)
                    TT(I-1) = T(I)
                    TBI = T(I)
                    $)
#
                CALL RWRITE (FD01, TT, 2, L-1,.JDENT, .WAIT, IEV, %9999)
                $)
#
#
#
                FIRST = .FALSE.
             $)
#
#
#
#
#                              MINIMUM PATH IS TRACED
#                              FROM THE GOAL SEGMENT
#
#
#
#
          GR1 = GROW + 1
          GC1 = GCOL + 1
          DO K = 1, 3
            $(
            CALL AREAD (FD01, CCC(1, K), NPPL2, 1, 1, 1, 1, 1, GR1 + K - 2,
                        JDENT, .WAIT, IEV, %9999)
            CALL AREAD (FD01, TTT(1, K), NPPL2, 1, 1, 1, 1, 2, GR1 + K - 2,
                        JDENT, .WAIT, IEV, %9999)
            CALL AREAD (FD01, MMM(1, K), NPPL2, 1, 1, 1, 1, 3, GR1 + K - 2,
                        JDENT, .WAIT, IEV, %9999)
```

```
  $)
GOALNO = MMM(GC1,2)
STEP = 1
WHILE ((MMM(GC1,2) ¬= SEGNO)&(STEP < NPPL))
  $(
  BT = TTT(GC1,2) - CCC(GC1,2)
  OLDGR1 = GR1
  IF (STEP > 1)
    $(
    CCC(GC1,2) = 0
    MMM(GC1,2) = GOALNO
    DO KK = 2, NPPL + 1
      $(
      MM(KK-1) = MMM(KK,2)
      CC(KK-1) = CCC(KK,2)
      $)
    CALL RWRITE (FDO1, MM, 3, GR1-1, JDENT, .WAIT, IEV, %9999)
    CALL RWRITE (FDO1, CC, 1, GR1-1, JDENT, .WAIT, IEV, %9999)
    $)
  X = 2
  Y = NPPL + 1
  Z = NLIN + 1
  IF      (((TTT(GC1-1, 1) == BT)&((GR1 ¬= X)&(GC1 ¬= X)))
          &(MMM(GC1-1,1) ¬= GOALNO))
          $(
          GR1 = GR1 - 1
          GC1 = GC1 - 1
          $)
  ELSE IF (((TTT(GC1,   1) == BT)&(GR1 ¬= X))
           &(MMM(GC1,1) ¬= GOALNO))
          GR1 =GR1 - 1
  ELSE IF (((TTT(GC1+1, 1) == BT)&((GR1 ¬= X)&(GC1 ¬= Y)))
          &(MMM(GC1+1,1) ¬= GOALNO))
          $(
          GR1 = GR1 - 1
          GC1 = GC1 + 1
          $)
  ELSE IF (((TTT(GC1-1, 2) == BT)&(GC1 ¬= X))
          &(MMM(GC1-1,2) ¬= GOALNO))
          GC1 = GC1 - 1
  ELSE IF (((TTT(GC1+1, 2) == BT)&(GC1 ¬= Y))
          &(MMM(GC1+1,2) ¬= GOALNO))
          GC1 = GC1 + 1
  ELSE IF (((TTT(GC1-1, 3) == BT)&((GR1 ¬= Z)&(GC1 ¬= X)))
          &(MMM(GC1-1,3) ¬= GOALNO))
            $(
          GR1 = GR1 + 1
          GC1 = GC1 - 1
            $)
  ELSE IF (((TTT(GC1,   3) == BT)&(GR1 ¬= Z))
          &(MMM(GC1,3) ¬= GOALNO))
          GR1 = GR1 + 1
  ELSE IF (((TTT(GC1+1, 3) == BT)&((GR1 ¬= Z)&(GC1 ¬= Y)))
           &(MMM(GC1+1,3) ¬= GOALNO))
            $(
```

```
                GR1 = GR1 + 1
                GC1 = GC1 + 1
                  $)
        ELSE
                  $(
                  STEP = NPPL
                  $)
        IF (GR1 ¬= OLDGR1)
         DO K = 1, 3
           $(
           CALL AREAD (FDO1, CCC(1, K), NPPL2, 1, 1, 1, 1, 1, GR1 + K - 2,
                       JDENT, .WAIT, IEV, %9999)
           CALL AREAD (FDO1, TTT(1, K), NPPL2, 1, 1, 1, 1, 2, GR1 + K - 2,
                       JDENT, .WAIT, IEV, %9999)
           CALL AREAD (FDO1, MMM(1, K), NPPL2, 1, 1, 1, 1, 3, GR1 + K - 2,
                       JDENT, .WAIT, IEV, %9999)
           $)
        STEP = STEP + 1
        $)
#
#                          REINITIALIZATION OF TOTAL COST IMAGE, TT
      DO ROW = 1, NLIN
        $(
        CALL RREAD (FDO1, MM, 3, ROW, JDENT, .WAIT, IEV, %9999)
        DO COL = 1, NPPL
          $(
          IF (MM(COL) == SEGNO) MM(COL) = GOALNO
          TT(COL) = INF
          $)
        CALL RWRITE (FDO1, TT, 2, ROW, JDENT, .WAIT, IEV, %9999)
        CALL RWRITE (FDO1, MM, 3, ROW, JDENT, .WAIT, IEV, %9999)
        $)
#
#
#                          NOW THE MINIMUM COST PATH BETWEEN ONE PAIR
#                          OF SEGMENTS IS FOUND AND THOSE SEGMENTS
#                          INCLUDING THE PATH ARE TREATED AS ONE
#                          CONNECTED SEGMENT THEREAFTER.
#
#                          FOR OTHER POSSIBLE PAIRS OF SEGMENT,
#                          REPEAT THIS PROCESS AGAIN.
#
#
#
      $)
#
#                            NORMAL RETURN
#
    CALL PPOP
    RETURN
#
#                            ABNORMAL RETURN
#
    9998 CONTINUE
```

```
#
#                                   READ OR WRITE ERROR
#
      9999 CONTINUE
      RETURN 1
#
      END
```

REFERENCES

1. F. F. Sabins, Jr., Remote sensing, W. H. Freeman and Co., San Francisco, 1978

2. J. S. Lee, "Digital image enhancement and noise filtering by the use of local statistics," IEEE Trans. PAMI PAMI-2, NO. 2, 1980, pp 165-168

3. J. S. Lee, "Refined filtering of image noise using local statistics," Computer Graphics and Image Processing 15, 1981, pp 380-389

4. R. M. Haralick, "Ridges and valleys on digital images," CVGIP 21, 1983, pp 28-38

5. R. M. Haralick and L. Watson, "A facet model for image data," CGIP 15, 1981, pp 113-129

6. R. M. Haralick, "Digital step edges from zero crossing of second directional derivatives," IEEE Trans. PAMI PAMI-6, NO. 1, 1984, pp 58- 68

7. M. A. Fischler, J. M. Tenenbaum, and H. C. Wolf, "Detection of roads and linear structure in low-resolution

aerial imagery using a multisource knowledge integration technique," CGIP 15, 1981, pp 201-223

8. T. C. Hu, Combinatorial Algorithms, Addison-Wesley, 1982

9. L. R. Ford, Jr., Network Flow Theory, The RAND Corp., P-923, 1956

10. R. O. Duda and P. E. Hart, Pattern Classification and Scene Analysis, Wiley, New York, 1973

11. R. Bajcsy and M. Tavakoli, "Computer recognition of road from satellite pictures," IEEE Trans. Syst., Man, Cybern. SMC-16, No. 9, 1976, pp 623-637 pp 37-44

12. G. J. VanderBrug, "Curve representation and mapping," Univ. of Maryland, College Park, MD, Tech. Rep. TR-561 Aug. 1977

13. J. Prewitt, "Object enhancement and extraction," in Picture Processing and Psychopictorics, B. Lipkin and A. Rosenfeld, Eds. New York, Academic, 1970, pp 75-149

14. U. Ramer, "Computer edge extraction from photographs of curved objects," New York Univ., New York, NY, Tech. Rep. CRL-34, Dec. 1973

15. A. Martelli, "Edge detection using heuristic search methods," Comput. Graph. and Image Process., vol. 1. Aug. 1972, pp169-182

16. U. Montanari, "On the optimal detection of curves in noisy pitures," J. Assoc. Comput. Mach., vol. 18. may 1971, pp 335-345

17. T. S. Gold and H. P. Smith, "A dynamic programming method for the detection of nearly straight lines in noisy pictures," IEEE Comput. Repository, R-74-167 Aug. 1974

18. R. W. Ehrich, "Detection of Global edges in textured image," IEEE Trans. on computers, vol. C-26 No. 6. June 1977, pp 589-603

The vita has been removed from the scanned document