8-2001

Development of an Arc View™ Extension to Measure Local Fractal Dimension

Christopher Page Caird

DEVELOPMENT OF AN ARC VIEW™ EXTENSION TO MEASURE LOCAL FRACTAL DIMENSION

by

Christopher Page Caird

A Thesis
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Arts
Department of Geography

Western Michigan University
Kalamazoo, Michigan
August 2001

ACKNOWLEDGMENTS

I would like to express my deepest appreciation to Ernest Anderson. Without Mr. Anderson's assistance, the completion of this thesis would not have been feasible. Without his unending patience and time, the development of the algorithm known as MWFDTPM would never have existed and I would not have completed the goals set out in my thesis. I cannot express the depth of my gratitude.

Secondly, I would like to recognize Gregory Anderson and the GIS Research Center at Western Michigan University for the set of orthophotos of Holland, MI used in the analysis. Greg provided much more than just the data set. He permitted significant use of his machines to run the program and obtain the results. He also provided invaluable and vast expertise on the use of GIS as well as continual friendship and humor to get me through the hard parts associated with the development of this application. It is absolutely inconceivable that I could have accomplished this work without Greg's help.

With Charles 'Jay' Emerson as my advisor, I was able to see this thesis materialize and take the necessary form that would lead to the completion of this project. I really want to thank Jay for his patience and guidance throughout this project from beginning to end. Since he first introduced me to the concept of fractals, Jay has inspired and motivated me towards setting and meeting the necessary goals.

I would also like to thank Changchun Yang a computer science master's student at Western Michigan University. Throughout my struggle to understand the Avenue scripting language, he was always available to provide fresh insight and technical assistance towards meeting my desired objectives.

In conclusion, I would like to dedicate this thesis to my mom and dad, Donald and Judith Caird. Thank you dad and mom for pushing me to study so hard when I was a kid. None of the success I have experienced would be possible without such strong loving parents. I owe you both such a debt of gratitude that is difficult to express and impossible to ever repay.

Christopher Page Caird

DEVELOPMENT OF AN ARCVIEW™ EXTENSION TO MEASURE LOCAL FRACTAL DIMENSION

Christopher Page Caird

Western Michigan University, 2001

Texture is defined as the visible cue suggesting the smoothness or coarseness associated with image tone or color (Emerson, 1999). Fractal dimension (D) offers insight towards understanding texture in the spatial as well as spectral context of an image (Mandelbrot, 1982). Fractal dimension is best understood as a range of values between two and three that are used to quantitatively describe a surface. (D) values near 2.0 represent a uniform area containing similar pixel values. In contrast, (D) values near 3.0 represent a domain that is spectrally complex. This complexity stems from light and dark pixels in close proximity to one another. The Triangular Prism Method (Clarke, 1986) was originally designed to compute a fractal dimension for an entire image. By applying a moving window to the triangular prism method, a local filter is created thus allowing grid images to be generated that are composed entirely of fractal dimensions. These images show recognizable areas that have high and low complexity. The results of this technique have shown that using the triangular prism method to compute the fractal dimension aids in characterizing the texture of an image.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

List of Figures - continued

CHAPTER I

INTRODUCTION

With the advent of earth orbiting satellites, scientists have been able to study the earth's patterns and its associated processes at greater area detail then ever thought possible. As aerial and space technology advances, the precision and accuracy available from earth orbiting satellites and aerial photography require similar advances in interpretation techniques. The type of information challenges professionals to devise new methods or improve upon old ones in order to analyze the recognition elements associated with image interpretation. As microcomputers develop a greater capacity to handle larger volumes of data at multiple resolutions, automated analysis techniques are sought to resolve the increasing complexity.

The amount of detail shown within an aerial photograph or satellite image depends heavily upon scale. Scale is the ratio of the distance on an image to that of the real world. One of the central arguments surrounding scale is that data may carry different information when presented at different scales (Bian, 1997). Within the context of remote sensing, many processes and patterns may appear homogeneous at one spatial scale and heterogeneous at another (Quattrochi, 1993). Changes in spatial scale often result in new and significant changes in the spatial properties of an image. This can result from aggregation or disaggregation of data when used with single data sets as well as multiple data sets. The information obtained from spatial data is well defined for some types of digital data while it may not be well defined for other types (Quattrochi & Goodchild, 1997). This problem of scale presents significant obstacles for the assessment of complexity within an image. As the science of interpretation

advances, there is a growing need for automated tools and procedures that will bridge the gap to allow researchers and analysts to use techniques that can operate at multiple scales and multiple resolutions.

Within the past 20 years, the emergence and widespread use of Geographic Information Systems (GIS), has prompted interest in scale as well as the assessment of image complexity (Quattrochi & Goodchild, 1997). This growing interest surrounds fundamental questions regarding accuracy, error modeling, and data structures. These issues are largely a result of the increasing availability of remote sensing and other spatial data at the local, regional, and global levels. With numerous options of earth-imaging satellites and airborne imaging systems, scientists are faced with the challenge of effectively combining and analyzing image complexity at various spatial, temporal, and spectral resolutions.

One of the primary problems facing investigators of image intricacy is how to properly handle multiscale data to prevent the distribution of erroneous information. Undesirable effects can result when analysts routinely rescale and resample complex images to appropriate spatial resolutions (Quattrochi et al, 1997). The continual merging of multiscale information from various sources has become a critically important research obstacle. The changes to information within the image must be understood in order to overcome issues surrounding the accuracy and precision of the data.

Analysts must also overcome the problem of determining an appropriate scale required for analysis and interpretation of image complexity. Obtaining image information from complex landscapes requires distinguishing heterogeneous areas from homogeneous structures. Discerning specific areas requires analysts to associate the image complexity with the pattern, structure, and function of the landscape. Prior

to performing landscape analysis, a detailed understanding of image texture is required.

CHAPTER II

TEXTURE MEASUREMENT METHODS

Explanation of Texture

The discussion of image complexity is referred to in terms of the texture and pattern of an image. Texture is defined as the visual impression of coarseness or smoothness resulting from the variability or uniformity of image tone or color (Avery & Berlin, 1992). Pattern is known as the general repetition of characteristic forms found within cultural and physical features (Lillesand and Kiefer 1994). It has long been possible to distinguish textural features but not quantify many characteristics to be used for classification. Prior to the work of Haralick and others (Haralick et al, 1973; Pratt et al, 1978; Goodchild, 1980), a series of adjectives were often used to characterize landscape complexity. The imprecise nature of terms like 'mottled' or 'stippled' lead to numerous errors and difficulty in conveying image complexity and pattern. This subjective method of distinguishing texture differences has progressed towards a quantitative means of accurately characterizing different textures and patterns. The advances pioneered by Haralick et al (1973), Goodchild (1980), and Woodcock and Strahler (1987) have brought about several classification improvements designed to quantify the texture within an image via the use of several types of textural or contextural information methods (Pesaresi, 2000).

There are four principal approaches used in image processing to describe the pattern elements of texture: statistical, structural, spectral (Gonzalez and Wintz, 1987)

and the manipulation of the spatial frequency within an image (Lillesand and Kiefer, 1994). Statistical methods yield characterizations of textures such as smooth, rough, etc. through a quantitative assessment. Structural techniques examine the arrangement of landscape patterns in an image, which include the distinction between human induced physical changes and those of naturally occurring processes. Spectral features are used to describe the average tonal variations in various bands (Haralick et al, 1973). Spectral processes and algorithms are designed to detect narrow peaks associated with tonal changes in the periodic patterns of an image (Gonzalez and Wintz, 1987). Spectral filters are designed to observe these tonal changes by blocking or passing energy over various spectral ranges.

Spatial filters act to emphasize or de-emphasize image data at various spatial frequencies. Rough areas of an image are defined by high spatial variability over a small distance. Low spatial frequency is associated with the smooth fluctuations between pixels over a large distance. Low pass filters are designed to emphasize changes in brightness over large areas rather then highlight the local detail within an image. In contrast, high pass filters are designed to emphasize the detailed high frequency elements of an image and de-emphasize the low frequency information (Lillesand and Kiefer, 1994).

Focal functions compute an output grid where the output value at each location is a function of the input cell(s) in some specified neighborhood of the location. Focal spatial filtering occurs when pixel values in an original image are modified on the basis of the gray levels of the surrounding pixels (Lillesand and Kiefer, 1994). For example, a simple low pass filter can be instructed to send a moving window across an image that results in a second image whose digital numbers (DN) at each pixel corresponds to the average value obtained from each of

the moving window. This will result in a smaller second image resulting from the averaging of values within a window.

In contrast, a high pass filter can be illustrated by subtracting a low pass filtered image from the original image (Lillesand and Kiefer, 1994). A high pass filter is based upon the notion that a low frequency image smoothes the detail of an original while emphasizing areas with high brightness values. The high frequency image is associated with enhancing the spatial detail while ignoring the bright areas within an image. By subtracting the values of a low pass filter from an original image, the end result is a smoothing function with the lighter and darker areas showing less contrast.

There are several algorithms designed to examine contextural information from pictorial features. The following methods represent a few of the most widely used techniques to examine image complexity: gray-level co-occurrence matrix (GLCO), Fourier Transforms, local variance, and the fractal dimension.

Gray-Level Co-occurrence Matrix

The gray-level co-occurrence matrix (GLCO) developed by Haralick et al (1973) is based upon repeated occurrences of gray-level patterns in an image. These repeated occurrences are rooted in the notion that texture and tone are not independent concepts. Haralick proposed decomposing image texture into two basic dimensions: tonal primitives and structure (Haralick, 1979). Tonal primitives describe the pixel intensities in a localized area while structure refers to the spatial organization of the tonal primitives thus allowing tone and structure to completely specify the image texture (Haralick, 1979).

The essential component of GLCO involves the use of arrays termed angular nearest neighbor gray-tone spatial dependence matrices for the measurement of texture (Haralick et al, 1973). The arrays are defined by the nearest neighbor pixels meaning there can only be eight pixels within one computation (Figure 1). The matrices that are created are a function of the angular relationship between neighboring pixels as well as a function of the differences between them (Figure 2). Using the GLCO matrix it becomes possible to detect the presence of given texture patterns by the relative frequencies at which two neighboring pixel cells are separated by a distance on an image (Gonzalez and Wintz, 1987). This configuration of occurrence varies rapidly in fine textures verses much more slowly in coarse textures. Co-occurrence matrices are advantageous because they take into consideration spatial properties of their neighboring pixels.

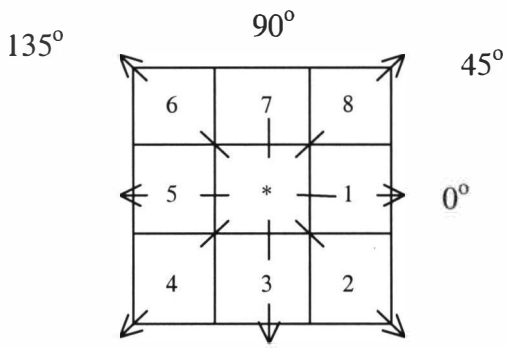


Figure 1. Horizontal Nearest Neighbor Pixel Cells.

Emphasis of the explanation of GLCO will be focused upon using the horizontal nearest neighbor method. The horizontal nearest neighbor method refers to pixels 1 and 5 (Figure 1). An example of a co-occurrence matrix can be described by

using a 4 X 4 digital image with four gray tones from 0 to3 (Figure 2). First, the number of different pixels are determined followed by ranking the pixel values (Figure 3). This arrangement shows the general form of the any gray-tone spatial dependence matrix. The element in the (2,1) position of the distance 1 horizontal P_0

J

M = I

0	0	1	1
0	0	1	1
0	2	2	2
2	2	2	3

Figure 2. 4 x 4 Matrix of a Digital Image.

matrix of Figure 4 is the total number of times two gray tones of value 2 and 1 occurred horizontally adjacent to each other (Figure 4). To determine the frequency, the pairs of pixels are counted such that the first pixel of the pair has gray tone 2 and the second pixel of the pair has gray tone 1. The frequencies are counts which describe how often two pixels located at a fixed geometric position relative to one another have a particular pair of gray levels. Each entry in M with i rows and j columns is a count of how many times a pixel with gray level i is situated a fixed distance from, and at a fixed orientation to a pixel with a gray level j in the original image (Haralick, 1979). The operator is applied to the image to yield the co-occurrence matrix. The number of rows or columns in the matrix is equal to the number of gray levels in the original image. In this example, every time gray level i is immediately to the left of gray level j, M(i,j) is incremented by one. In this example,

Gray Tone

Gray Tone

	0	**1**	**2**	**3**
0	#(0,0)	#(0,1)	#(0,2)	#(0,3)
1	#(1,0)	#(1,1)	#(1,2)	#(1,3)
2	#(2,0)	#(2,1)	#(2,2)	#(2,3)
3	#(3,0)	#(3,1)	#(3,2)	#(3,3)

Figure 3. General Gray-Tone Spatial-Dependence Matrix.

the operator is not bi-directional which means that the matrix M(i,j) is not equal to M(j,i).

$$P_0\text{, Horizontal} = \begin{vmatrix} 4 & 2 & 1 & 0 \\ 2 & 4 & 0 & 0 \\ 1 & 0 & 6 & 1 \\ 0 & 0 & 1 & 2 \end{vmatrix}$$

Figure 4. Co-occurrence Matrix Using 1 Gray-Tone.

Once the pattern is determined, texture classification can be based on criteria derived from the gray-tone spatial dependence matrix. GLCO matrix has three major weaknesses. It is: computationally demanding, memory intensive, and lacks the ability to capture primitive shapes well, particularly if they are large (Carr and Mirranda, 1998).

Fourier Transforms

The use of the Fourier transform has played a major role in image processing for many years due to its ability to solve a wide range of image processing problems. The Fourier transform is designed to examine the frequency domain of an image by separating it into various spatial frequency components. Conceptually, this type of transform operates by fitting a continuous function through the discrete digital number values of an image as if they were plotted along each row and column (Lillesand and Kiefer, 1994). The light and dark reflectance values along any given row or column can be described by combinations of sine and cosine waves with various amplitudes, frequencies and phases (Figure 5).

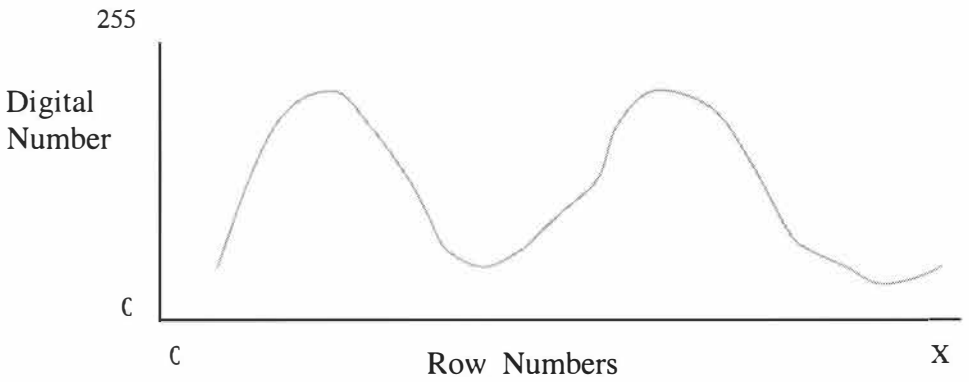


Figure 5. Digital Numbers Plotted Against Each Row of Numbers Within an Image.

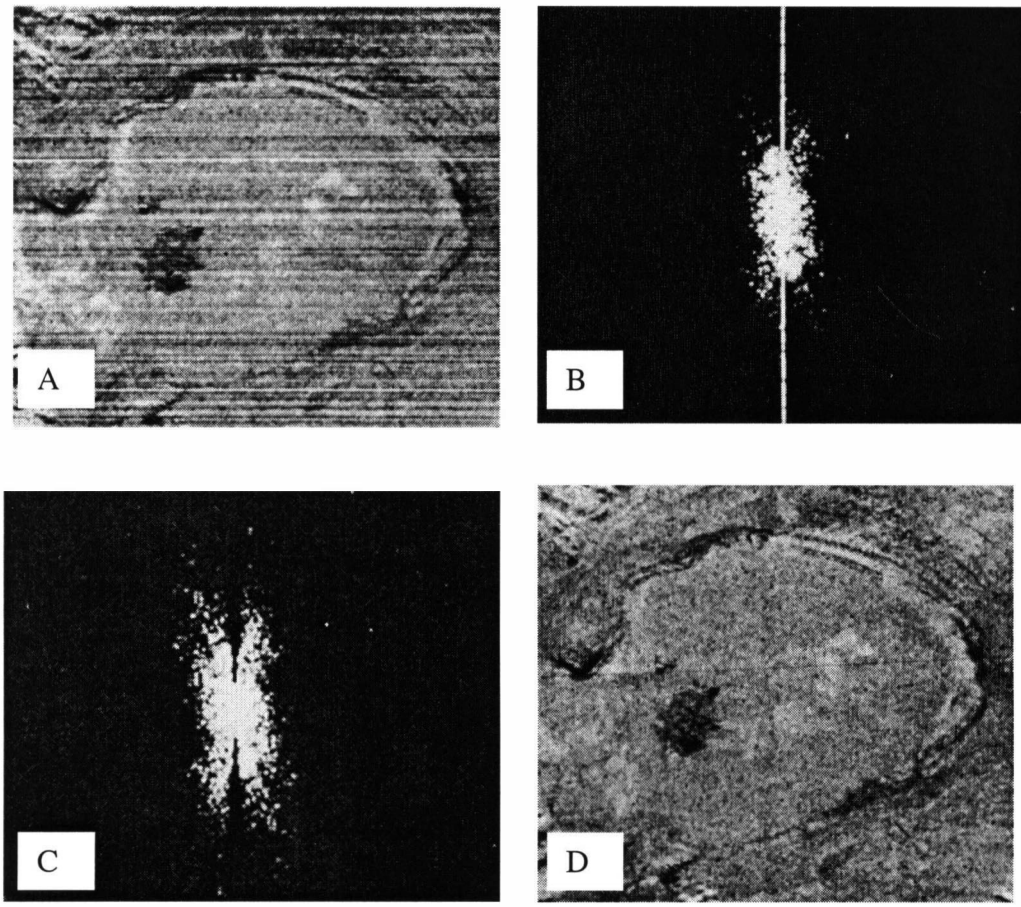
The Fourier transform results from the computation of the amplitude and phase for each possible spatial frequency in an image (Lillesand and Kiefer, 1994). After an image is separated into its component spatial frequencies (Figure 6a), the

values obtained can be displayed in a two-dimensional scatter plot known as the Fourier spectrum (Figure 6b). This output displays the lower frequencies at the center and a continual increase in higher frequencies as one moves out from the center. The main advantage to this method is its invertibility, which allows the function to transform between the Fourier spectrum and the data without the loss of information (Lillesand and Kiefer, 1994).

This invertibility allows the Fourier transform to assist in a number of image processing applications. For example, a filter can be applied directly to the power spectrum followed by an inverse transform. The results obtained display significant image enhancement.

The reduction of noise within an image is another common application of the Fourier transform. It is possible to perform what is known as a wedge block filter either in the horizontal or vertical direction (Lillesand and Kiefer, 1994). This method allows interpreters to take an original image with significant horizontal noise, generate a transform that would likely show a band of frequencies tending in the vertical direction. A vertical wedge block filter can be applied to the transform and the inverse Fourier transform can be used to regenerate the image with significantly less horizontal noise (Figure 6c and 6d).

The two major disadvantages associated with the Fourier transforms are the complicated computations as well as the memory requirements, which slows the process significantly (Lillesand and Kiefer, 1994). The number of calculations required are proportional to the squared number of sample arrays. Since the most general form of the Fourier transforms requires both complex floating point multiples and the calculation of sine and cosine functions, it is important to choose the number of sample arrays conservatively so that the computation is completed in a reasonable



Legend. A = Airborne multispectral image, B = Fourier spectrum

C = Wedge block filter, D = Inverse transform with reduced noise

Figure 6. Fourier Transform Using the Wedge Block Filter. (Lillesand & Kiefer, 1994)

amount of time (Lam and Decola, 1993).

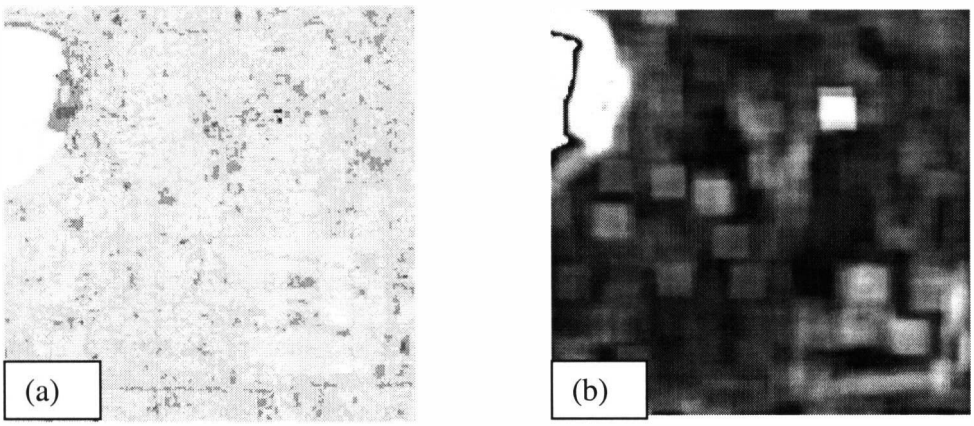
Local Variance

An alternative method for quantifying texture that is not as computationally intensive as the Fourier transforms is local variance (Woodcock and Strahler, 1987). This method is intended to determine the spectral band at which the maximum variability occurs in an image within a user specified moving window as it passes through an entire image. The logic behind local variance is based on the notion that if spatial resolution is finer then the objects in view, then objects are composed of several pixels and most pixels will be correlated with their neighbors. This situation causes the local variance to be low. When objects in an image are nearly the same size as the pixel cell size, the values of adjacent cells tend to be different causing the local variance to increase (Figure 7).

In Figure 7, local variance gives an indication of how typical of a whole distribution the mean is within a specified window. In this image, a window size of 15 X 15 was chosen to determine the dispersion of pixel values. Where the lake meets the shore the greatest amount of spread is observed. The dark line indicates the shore front while the bright values the left and right indicate the edge that separates the city from the lake.

Local variance is particularly sensitive to the window size, which makes it particularly useful during edge detection. An edge detector acts to highlight big changes while gradual changes tend to be flattened out. This method is a function of the sizes and spatial relationships of the objects within the scene. A larger window

size will cause the number and size of landscape features to exhibit higher local variance. For example, areas containing a mix of urban, suburban, and forested



Legend. A = Original image, B = Local Variance using a 15 X 15 window

Figure 7. Variance of Lake Charles, Louisiana Band 3.

landscapes display high local image variance or bright areas. Knowing which areas represent high variance can potentially lead interpreters towards determining what may be an appropriate scale or spatial resolution for a particular application. There are questions that remain to be answered in order to determine if this method aids in image interpretation and can be combined with other spectral or contextural applications. It is important to note that local variance works on single image bands at a time.

CHAPTER III

FRACTAL DIMENSION

Fractal Background

Fractal analysis has been suggested as one of the greatest additions to the scientific community because of its potential usefulness for modeling natural phenomena at local levels (Jaggi et al, 1993). In 1977, Mandelbrot (1977) first introduced the concept of fractals with his essay identifying a family of shapes that describe the irregular and fragmented patterns in nature. He stated that fractals are described by any function for which the Hausdorff-Besicovich dimension exceeds the topological dimension (Mandelbrot, 1977). According to Xia and Clarke (1997), this definition was thought to be unsatisfactory early in its conception. The reason for disagreement was because it failed to include a number of sets that were regarded as fractals. Later, Mandelbrot retracted this definition in favor of an idea that came to be known as self-similarity(Xia and Clarke, 1994). Self-similarity exists when both the overall shape and the cascade that generates each portion of the shape is geometrically similar to the whole (Mandelbrot, 1982).

The fractal concept gathered substantial attention as a means for characterizing remotely sensed images and the applications associated with the analysis of spatial and spectral phenomena (Clarke, 1986; Jaggi et al, 1993; De Cola, 1989; Emerson et al, 1999; Collins& Woodcock, 1999). The use of fractal dimension

to improve analysis of spectrally and spatially complex images has seen significant strides (Lam, 1990; Jaggi et al 1993; Quattrochi et al, 1997; Emerson et al, 1999).

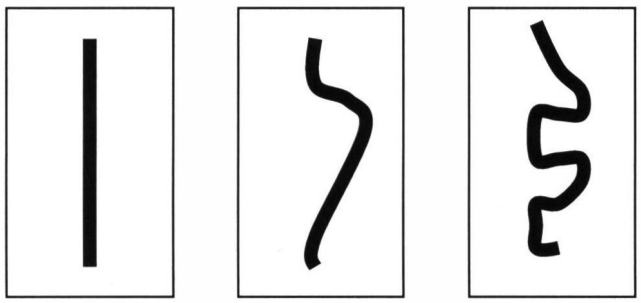
Topological dimension (D_t) and the fractal dimension (D) are not only computationally distinct but also provide dissimilar information during analysis of texture. Topological dimension is described as the number of distinct coordinates needed to specify a position on or within an object (Xia and Clarke, 1997). In Euclidean geometry, the topological dimension (D_t) has a null, or nothing set. The null is assigned the dimension of -1 while countable sets have a value of 0 representing a point, 1 representing a line, and 2 representing a surface. The difference between countable sets and uncountable sets is that countable sets are a finite set of integers while those that are uncountable are not. These infinite sets are split into the countable and uncountable ones. If a set is countable and infinite, then its elements can be listed as:

$$x_n = f(n) \qquad [1]$$

The fact that the elements can always be listed as a countable set means it is possible to work with them one at a time. Since the null set has the dimension -1, any countable set like a point has the dimension of 0. A line has the dimension of 1 since it can be separated by a countable set of two points and a plane is assigned a dimension of 2 because three points can separate a surface. These topological dimensions, (D_t), are expressed with integer values and remain constant regardless of the complexity of the line or surface under scrutiny.

The extent to which patterns are fragmented and irregular is expressed in terms of fractal dimension (D) (Mandelbrot, 1983). One of the more important

aspects associated with (D) focuses on the idea that it is always greater then it's Euclidean counterpart. The fractal dimension (D) of a line can range between 0 and 1 for shapes that exhibit varied complexity. A line is an example of where the amount of complexity exhibited can change within a specified area. In the case of a line, it can be between the value of 1 and 2 depending upon its irregularity. The more contorted the straight line becomes the more space it fills within the box (Figure 8).



(a) (b) (c)

Legend. A = Estimated D = 1.0, B = Estimated D = 1.3,
C = Estimated D = 1.7

Figure 8. Estimated Fractal Dimension of a Line.

A surface can display fractal dimension values between 2 and 3 with the higher values representing greater surface complexity (Mandelbrot, 1977). Examination of a surface that reveals a value of 2.0 indicates the surface is completely smooth while a value close to 3.0 suggests extremely coarse. A good example of a smooth surface would be a flat sheet of white paper, which would have a fractal dimension near 2.0. By crumpling the paper into a ball, a complex surface

results from the increase of light and dark portions that are in close proximity. The fractal dimension value would approach 3.0 (Figure 9).

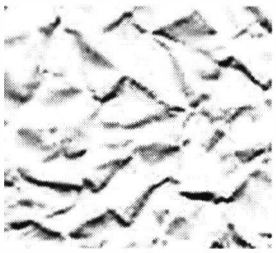


Figure 9. Crumbled Paper as Complex Surface.

The property of self-similarity is defined by the invariance of a shape under certain transformations of scale. Self-similarity is explained by the irregular patterns that repeat exactly or statistically as the scale of observation varies (Goodchild et al, 1987). An example of an object that displays exact self-similarity is the fractal shape of the Cantor set (Figure 10).

Figure 10. Cantor Set.

Although the Cantor set is to be thought of as the "final row" in this picture, the entire picture must be considered altogether. It is important to note that at each stage the picture is "doubled" into two copies that precisely resemble the whole. However, at each stage the new set becomes two-thirds smaller then the original. With these

properties, the Cantor set exhibits self-similarity at every scale over a uniform reduction of scale and qualifies the Cantor set as a fractal with Hausdorf dimension given by the equation below where the 2 represents the two remaining pieces when one-third is removed from the whole:

$$\text{Log2/Log3} = 0.63092973.... \qquad [2]$$

Areas and surfaces can be considered self-similar when an object within the image lack clues to its scale. Most real world objects are not self-similar at all scales and can be modeled by stochastic fractals where self-similar properties have inexact patterns (Xia and Clarke, 1997). For example, Mandelbrot introduced the concept of fractal curves using a stretch of coastline along Britain (Mandelbrot, 1982). Close scrutiny of the English coastline at a scale of 1/100,000 reveals numerous irregular and complex inlets (Lam, 1990). Reexamination at 1/10,000 shows more detail of similar complexity to that of 1/100,000. This complexity exhibits properties of stochastic self-similarity. This fundamental concept when applied to curves and surfaces allows for portions to be considered as a reduced scale of the image (Lam, 1990). Thus, analysts can begin to apply the properties of fractals to those of real world spatial questions.

CHAPTER IV

TEXTURE MEASUREMENT USING THE FRACTAL DIMENSION

Variogram

There are several viable techniques for computing the fractal dimension including the variogram, the isarithm, and the triangular prism. These methods utilize the fractal dimension as an index of image texture.

The variogram is a method used to measure image complexity using the fractal dimension. Variograms are used to describe how changes in brightness vary with distance and have been used significantly in *kriging*, a spatial interpolation technique (Lam and DeCola, 1993). For any continuous surface of dimension $2 \leq D \leq 3$, the expected value of the squared height difference between two points is expressed by:

$$E[Zp - Zq]^2 = k\left(d_{pq}\right)^{2H} \qquad [3]$$

where Zp and Zq are the brightness values at the points p and q and d_{pq} is known as the lag spacing or lag increment. The lag spacing is the distance between successive brightness values (Isaaks and Srivastava, 1989). The H parameter in this equation describes the complexity and has values between 0 and 1 (Lam and DeCola, 1993). The parameter H describes the complexity of the surface and indicates with large

values, a strong tendency to return to neighboring pixel values which results in a smooth surface (Lam and DeCola, 1993). In contrast, a small H returns a complex surface with white and dark pixel values in close proximity. If the variance is computed for different distances, the D can be estimated from the slope b of the regression between the logarithms of variance and distance so that (Lam and DeCola, 1993):

$$D = 3 - H = 3 - b/2 \quad [4]$$

The variogram is a plot of incremental variance between two locations along the ordinate against the distance between the two locations along the abscissa (Xia and Clarke, 1997). The variogram offers insight towards understanding image complexity and pattern. There is a single major advantage and two disadvantages to using the variogram method. The advantage is that it can be used upon samples of regular and irregular grid data to increase processing speed. The major disadvantage to the variogram is that it often does not behave linearly in the log-log plot at all scales. Often, the ends of the log-log plot will slope downward, which results in D being greater then 3.0. The other disadvantage of the variogram is that it is not appropriate for detecting scale ranges within images (Lam and Decola, 1993). This method requires several complicated computations compared to that of the isarithm method. The isarithm technique is rooted in that distance is examined using changes in increments between neighboring cells.

Isarithm

The isarithm method for measuring fractal dimension is based on the principle that as the cartographic scale of a curve increases more detail becomes visible and the length of the line increases at a consistent rate (Clarke and Schweizer, 1991). The rate of increase is determined by calculating of the slope of the log-log plot of the entire length of the line against the length of the measurement instrument or step-size. The fractal dimension is equal to 1 minus the slope of the regression line (Figure 11). Using the argument that $D_{SURFACE} = D_{ISARITHM} + 1$, a modification of the line-divider technique can be used to measure the $D_{ISARTHM}$. For example, a technique developed by Lam and DeCola (1993) requires users to specify an input matrix of pixel values to be used as z-values for a surface. This is followed by indicating an isarithmic interval to determine the number of isarithms to be examined. The technique concludes by requiring a particular number of step sizes. Step sizes are defined by units of grids that proceed by the number of grids (1, 2, 4, 8, etc) (Lam and DeCola, 1993). The procedure is designed to compare neighboring cells along the rows or columns and examines if the pairs are both black or both white. If they are not, an isarithm is said to be lying between the two neighboring cells and a boundary counter is incremented. Using the total number of boundary pairs, the length of the isarithmic line can be approximated. This process is continued until the specified number of walks or step sizes has been completed. After the counting of the boundary cells per walk size, the rate of increase in line length can be calculated by the slope of the regression line of a log-log plot of the entire length of the line against the length of the step size (Figure 11). This process computes the corresponding D value using 2 – b, where b is the slope of the line from the log-log regression (Clarke et al, 1991). The fractal

dimension of the surface is the average of the D values for those isarithms for which $R^2 \geq 0.9$ (Lam and DeCola, 1993).

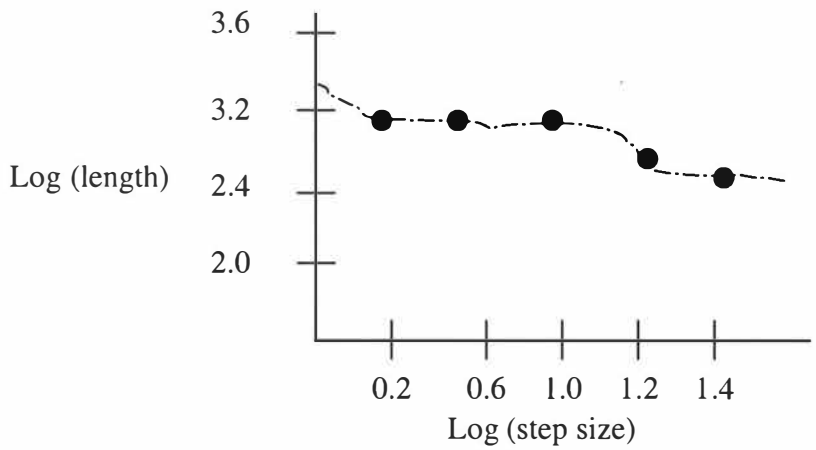


Figure 11. Log-Log Regression of Isarithm Method.

Triangular Prism

The triangular prism has also been repeatedly shown to provide fractal measures without being overly sensitive to noise within an image (Qiu et al, 1999; Jaggi et al, 1993; Clark et al, 1991; Lam, 1990). This method first introduced by Clarke (1986) requires that an image be located on a square grid defined by (x, y) coordinates with z-values corresponding to the digital numbers of the pixels using one band (Figure 12). From the corners of a specified kernel, reflectance values are obtained from the four corners (e.g. $\overline{AZ}, \overline{BF}, \overline{CG}, \& \overline{DH}$). These values are used to determine the average elevation that is used to determine the z value for the interpolated center point (e.g. $\overline{MO}$). Drawing a line from each corner to the center results in dividing the top into four distinct triangles. The last step requires

computation of the surface areas of the triangles. By repeating this method, Clarke (1986) was able to establish a relationship between the total area of the surface and the spacing of the squares.

The points E, F, G, and H in the figure are locations at the four corner pixels on a grid. The points A, B, C, and D are the ends of the lines that extend in the 'z' plane according to the reflectance values of the corner pixels. A vertical line is then drawn from the center base of the square and labeled $\overline{MO}$. M is the middle coordinate (center pixel of square) with a new value O determined from the average digital numbers of the four corner pixels (Jaggi et al, 1993) (Figure 13).

$$\overline{MO} = \left(\overline{EA} + \overline{FB} + \overline{GC} + \overline{HD}\right)/4 \qquad [5]$$

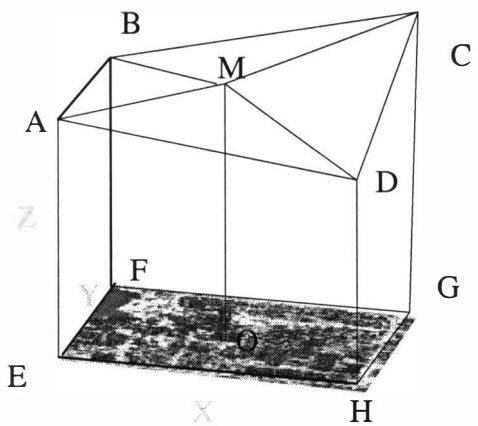


Figure 12. Three Dimensional Triangular Prism Method.

By doing this, we obtain 4 triangular prisms (Figure 13). The area of these surfaces can be computed by using trigonometric functions and Heron's formula (Clarke, 1986). Heron's formula is used to provide a means of calculating the area when only the lengths of three sides are known. This method is described by a given triangle with sides of lengths a, b, and c the area can be computed by:

$$Area = \sqrt{[s(s-a)(s-b)(s-c)]} \qquad [6]$$

Where S is the semi-perimeter,

$$S = ((a+b+c)/2) \qquad [7]$$

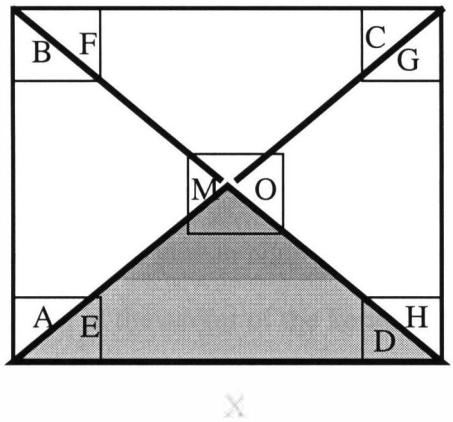


Figure 13. Top View Calculation of Triangular Prism Method.

Clarke and Jaggi used the triangular prism method to compute the fractal dimension of entire images (Clarke, 1986; Jaggi et al, 1993). Using this method at the global level yields one measurement of fractal dimensions for a whole image or specified subsets of an image.

Local Fractals

Convolution is a mathematical operation that is fundamental to many common image-processing operations. Convolution provides a way of `multiplying together' two arrays of numbers, generally of different sizes, but of the same dimensionality, to produce a third array of numbers of the same dimensionality (Fisher et al, 2000). This can be used in image processing to implement operators whose output pixel values are simple linear combinations of certain input pixel values.

Convolving an image involves two specific procedures. First, a moving window must be established that contains an array of coefficients or weighting factors. These arrays are referred to as operators or kernels and are normally an odd number of pixels in size (Lillesand and Kiefer, 1994). Second, the kernel must move through all the positions where the kernel fits entirely within the boundaries of the image. Each kernel position relates to a single output pixel. This value is calculated by multiplying together the kernel value and the underlying image pixel value for each of the cells in the kernel, and then adds all these numbers together (Fisher et al, 2000). The digital number at the center of the kernel in a second convoluted output image is obtained by multiplying each coefficient in the kernel by the corresponding digital number in the original image and adding all the resulting products (Lillesand and Kiefer, 1994). The influence convolution may have on an image depends directly upon the size of the kernel used and the value of the coefficients contained within the kernel.

In contrast to the global fractal dimension, local fractal dimensions are a series of values determined within an overlapping or non-overlapping moving

window. First, a square window of odd size is specified. A center pixel value is used as the peak and requires an odd size window from which to generate the four triangles. The fractal dimension is calculated for the window and is then used in an output image as a pixel with the assigned value and the process is repeated. By combining the set of fractal dimension, the results can be represented in an entirely new grid image. It is important to note that the new output image is shorter on all sides by:

$$\#\ \text{Rows} = (\text{Truncate}(\text{Image Size}-\text{Window Size})/\ \text{Window Gap})+1 \quad [8]$$

$$\#\ \text{Columns} = (\text{Truncate}(\text{Image Size}-\text{Window Size})/\ \text{Window Gap})+1 \quad [9]$$

The focus of this work centers upon assessing landscape complexity using the triangular prism method to determine local fractal dimension for texture classification. The advantage of this method is based upon its computational ease as well as the simplicity of the graphical output generated from plane geometry. The triangular prism method developed by Clarke (1986) was designed to measure natural surface area as an alternative to the line-divider method.

Purpose Statement

The purpose of this research is to construct a spatial analysis filter using the triangular prism method. This method is used to compute the fractal dimension (D) that can be used to quantitatively characterize image texture. This filter, titled Fractex, is built as an extension to the Geographic Information System ArcView™. This local filter is designed to pass a moving window throughout an image and

compute the fractal dimension values for a specified window size. These values are used to generate a second image that is quantitatively and visually assessed in order to determine the level at which the image no longer provides viable information.

CHAPTER V

METHODS

Methods Background

This work describes the alteration of the triangular prism method for measuring the fractal dimension of an image. Prior to alteration, this technique has been shown to successfully compute the fractal dimensions from remotely sensed images of digital terrain data. Jaggi et al (1993) developed an interactive triangular prism technique based upon the work done by Clarke (1986). This method is a self contained program for use in the Image Characterization and Modeling System (ICAMS) (Quattrochi et al, 1997). The program titled triangular.exe was written in C and designed to be used through Arc/INFO and Intergraph software packages and also as a stand-alone application.

The extension to ArcView known as Fractex expands upon the work of Quattrochi et al (1997) through a series of modifications to the triangular executable using an object-oriented scripting language known as Avenue. The modified executable is renamed to be MWFDTPM.exe (Moving Window using Fractal Dimension of the Triangular Prism Method). Fractex functions on a menu driven geographic information system (GIS) software package known as ArcView, versions 3.0 and higher. ArcView is a Geographic Information System designed and distributed by the Environmental Systems Research Institute (ESRI) allowing users to perform numerous mapping and spatial analysis functions. An extension is defined as a kind of object database that can be used to provide new functionality to ArcView

without altering existing projects. Avenue is designed to simplify the development of complete extensions like Fractex. ESRI's window-based software enhances fractal analysis significantly with the assistance of numerous spatial analysis tools and extensions.

Summary of Software Flow

The main objective of this research is to provide a means of characterizing the texture of an image using the local triangular prism method in an ArcView environment. In order to assess the complexity of an image, it is important to understand how the Avenue script and C code function together. In order to characterize texture, a script was written in an object oriented scripting language called Avenue. It was designed to accept ten user-input parameters, perform a remote procedure call (RPC) to the C code, and display results within an active theme window.

The script begins by checking for critical components required to compute the fractal dimension. The first step involves checking to see if the Spatial Analyst extension is loaded. The ArcView Spatial Analyst is a tool used for understanding spatial relationships within data sets. The main component of the Spatial Analyst is the grid theme. The grid theme is the raster equivalent of the image theme displayed within a view. The Spatial Analyst is required in order to generate the single band grid theme from the original image. The grid theme is required for spatial analysis functionality within ArcView. Prior to using Fractex, the original image displayed in ArcView must be converted have one band extracted as a grid in order to display the output results.

The script proceeds by gathering a range of attributes associated with the newly created grid theme. Information obtained includes the projection, the cell size, the extent, the number of rows and columns, the origin, and the upper right coordinates of the grid (Figure 14). These parameters will be used for two specific tasks. The first task is the creation of two text files. One text file, titled fractal.txt, contains a header with the newly appended fractal dimensions. This text file is imported into ArcView as a newly created grid theme for image analysis. The second text file, titled debug.txt, provides detailed information regarding each window and the computations associated with it. The information in debug.txt includes the dimensions of the window, the step size, the R-squared value, and the fractal dimension for each window. Located at the bottom of the debug.txt file is the average fractal dimension for all the windows and the average R-squared value for the entire image. When the program is initiated, the two text files known as fractal.txt and debug.txt are generated in tandem via the Avenue script.

The next portion of the script focuses on the creation of ten windows that allow the user to tailor the moving window computation. The first input parameter includes the path location of the stored file without the image extension (eg: a:\hol_15) and the second asks for the extension of that file (eg: gis). Fractex can accept four file types: BIL, LAN, GIS, and DAT. BIL stands for band interleaved by line. BIL is a form of data storage in which each record in the file contains a scanned line (row) of data for one band. All bands of data for a given line are stored consecutively with the file. LAN is an image data file, usually with multiple bands, containing data file values for a digital image. LAN and GIS files are ERDAS format images and usually contain remotely sensed data or a scanned image. GIS stands for a geographic information systems file. This is a single band image file that usually

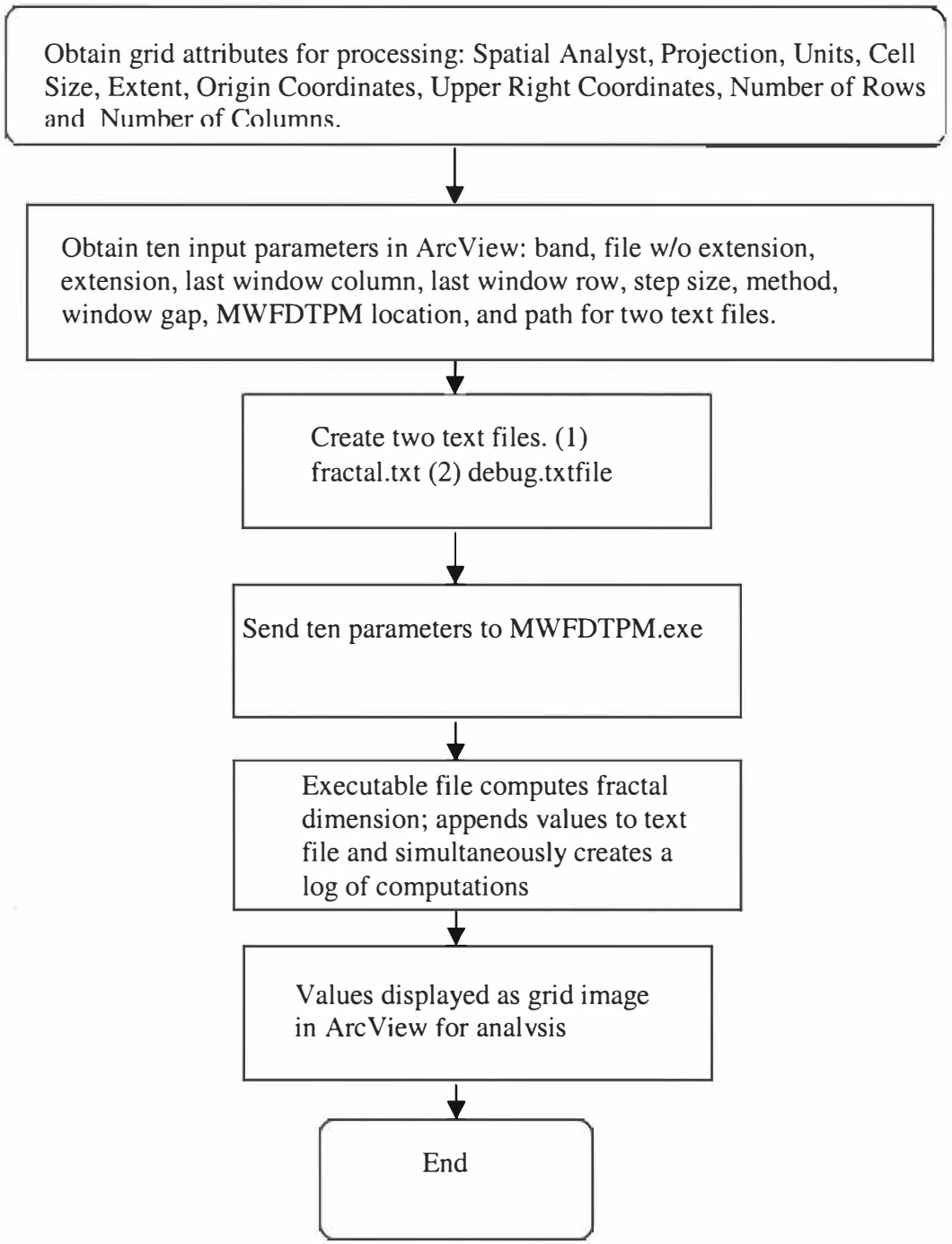


Figure 14. Flow Diagram of Fractex.

contains data file values that correspond to GIS classes. DAT are data files. These are provided with ERDAS software and store various types of data. The user is also requested to designate the path name for the location of MWFDTPM.exe as well as where to place the two text files.

Subsequent to entering the text file location, Fractex requires the band chosen for the analysis. The following two input criteria require the user to select a step size and a regression method of arithmetic or logarithmic. The fractal computation method is used to compute the fractal dimension at geometrically decreasing square sizes. The methods that are available for computing the fractal dimension are either a linear or logarithmic method.

The next two inputs to Fractex specify the window size to be used. The user is asked for the upper right and lower left dimension values the window. The moving window begins at the upper left corner of the image (Figure 15). The window 'moves' by maintaining the original size but uses different coordinates to step over row major based upon a window gap. Row major is defined by repositioning the window along a specified horizontal axis of the grid.

Fractex requires a specified window gap size to determine the number of columns the new window will be located to the right of the first window This pattern continues row major until it reaches the end of the row (Figure 16). When the algorithm reaches the end of the row and there are not the required rows and columns, the algorithm will stop and begin a new row. The new row uses the same window parameters directly below the first window at the specified window gap (Figure 17). This new row will also continue moving row major to the right until the entire image has been analyzed.

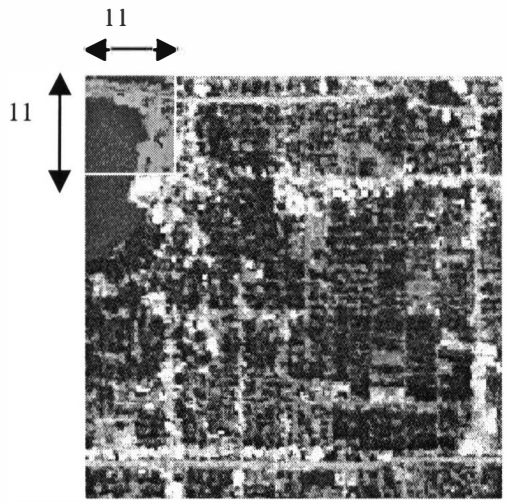


Figure 15. Beginning Position of Window.

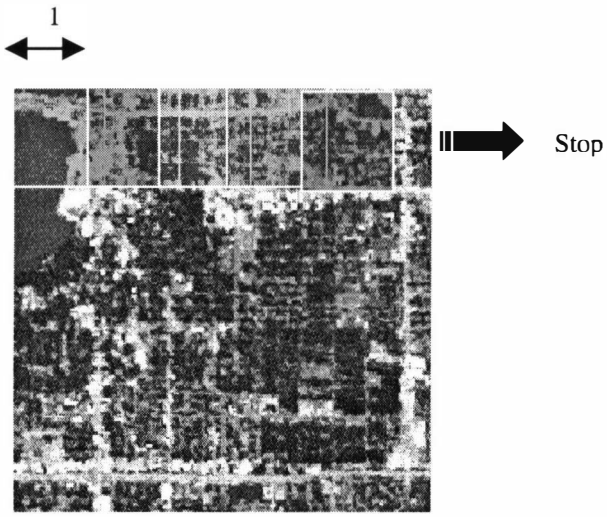


Figure 16. Moving Window Using a Window Gap of One.

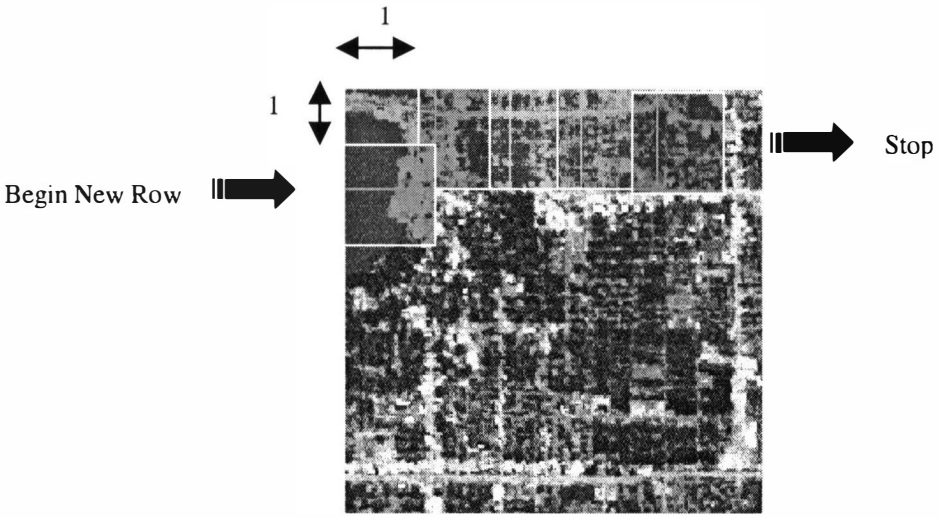


Figure 17. Moving Window Beginning New Row Lower by Window Gap.

The final task occurs during the remote procedure call (RPC) associated with the computation of the fractal dimension. The remote procedure call (RPC) allows the client (Avenue script) to trigger procedures running in another program known as a server. The C++ program written by Anderson (2001) MWFDTPM.exe acts as the server. An RPC server resides on a particular host machine and provides a set of services to clients. RPC clients connect to a RPC server either locally or through a network and initiate the specified services from that server. Through a local connection, multiple events occur in rapid succession. During the computation of each fractal dimension for each window, the program is simultaneously appending values to the two text files fractal.txt and debug.txt.

The new program is altered from the original triangular.exe by computing a single gain value for the entire image rather then a gain value for each individual window. In the original triangular.exe (Quattrochi et al, 1997), a gain value is determined from the minimum and maximum values within the specified window.

Gain values are used to scale the corners of the window such that the values for the entire image are stretched from 0 to 255. While the MWFDTPM.exe generates a series of moving windows, a single gain value is determined for the entire image and subsequently used for all individual window computations allowing for a much smoother more accurate color stretch observed in the output image (Anderson, 2001). This stretch also allows for relative comparisons to be made within an image.

A minor change to triangular.exe involved some of the calculations that were happening on the inside of the loop. A few calculations produced identical results every time the program was executed. Movement of a variable calculation was determined by inspection of source code. The criteria involved determining whether or not a variable calculation was constant for the duration of the loop. Functions were moved outside of the loop to clarify the purpose of the function and to speed up performance. The number of moved variables was limited and resulted in only a slight increase in processing time (Anderson, 2001).

The second minor alteration involved the display of the results from the computation of the fractal dimension. In the original triangular.exe results were displayed on the screen. In the altered version, all values were sent to the two text files previously discussed.

CHAPTER VI

RESULTS

Validation

Three test images were examined to test the validity of Fractex as a viable extension. The analysis began with a close scrutiny of the generated ideal fractal surfaces as a means of testing the precision of this extension. The final site used to examine the viability of Fractex was an orthophoto of Holland, Michigan because it contained a wide range of land uses.

All images examined using Fractex used identical window parameters except for minor changes in window size (Table 1). The window gap used in the calculations was always 1 and the fractal computation method was arithmetic unless otherwise noted. The step size of 9 was maintained throughout the program because it is the maximum number steps even though most computations will not use all nine.

Simulated Fractal Surfaces

Ideal fractal surfaces can be generated using a method known as 'fractional Brownian'. Brownian motion is a sophisticated random number generator, based on a process in plants discovered by Robert Brown in 1827 (Lee and Hoon, 2001).

Table 1.

Table of Parameters Used During Analysis

Band	1*
File w/o Extension	Variable
Extension	Variable
Last Window Column	Variable
Last Window Row	Variable
Step-Size	1
Method	Arithmatic
Window Gap	1
* Indicates Analysis of Lake Charles used Band 3	

According to Einstein, bodies of microscopic particles are suspended in a liquid and perform irregular thermal movements called Brownian molecular motion and are easily observable in a microscope (Mandelbrot and Ness, 1968). Chapter three introduced how fractal patterns develop when a simple example like the Cantor set is transformed continually on smaller and smaller scales. Brownian motion is able to produce these fractal patterns based on the property of self-similarity. For example, a pen can be used to mark dots at random on a sheet of paper to produce an image. However, instead of being completely random, the movement of the pen from one location to the next is selected randomly from a set of rules each having a fixed probability of being chosen (Lee and Hoon, 2001). As a result, the fractional Brownian motion theory can describe naturally occurring rough surfaces by combining fractals and Brownian motion to produce fractal patterns. This term is given to a class of variograms from:

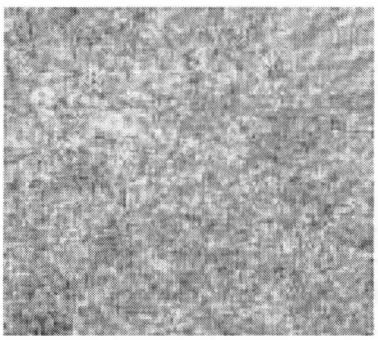
$$\gamma(h) = h^{2H} \qquad [10]$$

that have surface fractal dimensions equal to 3 – H (Lam and DeCola, 1993). The H parameter represents values from 0 to 1. For large H values, the results tend to exhibit small differences between neighboring points. In the case of small H values, the generated surface tends to be highly irregular. According to Lam and DeCola (1993), an H value of 0.5 produces a surface that is statistically self similar.

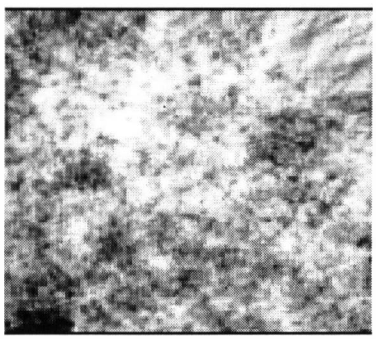
Three surfaces were generated using the shear displacement method titled SURF_GEN and written in FORTRAN by Lam and DeCola (1993). The shear displacement method takes a grid matrix, initializes it to a uniform value ($z = 0$), and generates a succession of random lines across the grid surface (Goodchild, 1980; Goodchild, 1982). Across each line, the surface is faulted vertically to form a cliff. This method of faulting is repeated until numerous lines are generated between points (Emerson et al, 1999). Each cliff's height is determined by the user specified parameter (H) so that the variance between two points is proportional to their distance.

The simulated fractal surfaces were the first set of images analyzed using Fractex. Three surfaces were generated to test the accuracy and validity of the algorithm. The first one uses H01 and an H value of 0.1 represents the most complex simulated surface (Figure18A). The surface outcome returned an area with light and dark pixels in close proximity to one another. Increasing the H parameter to 0.5 results in a progressively smoother surface (Figure. 18B). This set shows an even transition between high and low reflectance values. The second surface titled H05 (Figure 18B) is a surface with a known fractal dimension of 2.5. The third image uses an H value of 0.7 and returns a known fractal dimension of 2.3. The third image resembles a surface of a cloud and is titled H07 (Figure 18C). As a result, these

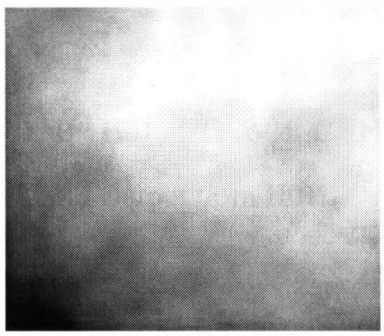
generated fractional Brownian surfaces can be used as a null hypothesis in order to test fractal techniques as well as a means of comparison against real world images.



A



B



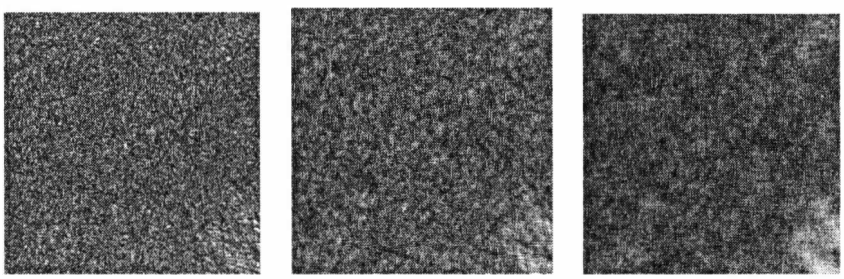
C

Legend. A = H=0.1 D = 2.9, B = H=0.5 D=2.5, C = H=0.7 D=2.3

Figure 18. Simulated Fractal Surfaces Generated by the Sheer Displacement Method.

A series of grid images were generated from the simulated surfaces based upon the local fractal dimension and displayed in ArcView. A mean fractal dimension for the whole image is obtained in Fractex by summing the fractal dimension value computed for each window and dividing by that number to obtain a singular value for the entire grid output. In simulated surface H01, we observe a high range of values that are as low as 2.086 and as high as 3.007 associated with this rough surface at

window size of 11 X 11 (Figure 19). As the window size increases, two significant responses occurred. The first event is observed in the lower right hand corner of the image and shows very low fractal dimensions. This was followed by a significant decrease in the range of values displayed in the associated theme in ArcView. It is important to note the uniformity of the output grids as window size increases.

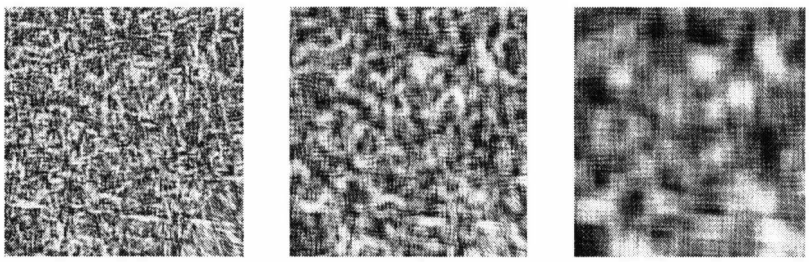


A B C

Legend. A= H01 at 11 X 11, B= H01 at 21 X 21, C= H01 at 49 X49

Figure 19. Average Fractal Grid Output from H01.

Using Figure 18B with a known fractal dimension of 2.5 returned the grid output known as H05. These images showed notable complexity with an even distribution of high and low values of fractal dimension in close proximity to one another. There is a significant contrast between the expected fractal dimension of 2.5 and the average fractal dimension that approaches 2.2. Increasing the window size causes a reduction in precision of Fractex (Figure 20). The surface displayed several significant and easily recognizable pockets of high and low fractal dimensions with a reduction in variability and detail when using a window size of 49x49. Using progressively larger window sizes the areas of low complexity tend clump together and have less uniform distribution.



A B C

Legend. A= H05 at 11 X 11, B= H05 at 21 X 21, C= H05 at 49 X49

Figure 20. Average Fractal Grid Output from H05.

H09 returned an average fractal dimension of 2.0 while the known fractal dimension was 2.1. The range of values used to obtain this average showed a very small spread between the highest and lowest fractal dimension (<0.043). A unique response began to emerge in H09 as a localized region of higher fractal dimensions was seen in the middle of the image with a gradual blend of white near the upper right and low left of the output .

Legend. A= H09 at 11 X 11, B= H09 at 21 X 21, C= H09 at 49 X49

Figure 21. Average Fractal Grid Output from H09.

The outcome obtained form the analysis shows significantly less average fractal dimension values when compared to those of the known fractal dimenion (Figure 22). The contrast between the expected fractal dimension and the observed is a result of using the known global fractal dimension from the simulated surfaces and comparing it with an average fractal dimension. These results, however, show that even as window size increases the average fractal dimension does not change drastically in any of the simulated surfaces. The major result observed is the major drop from the known fractal dimension to the average fractal dimension. Even though these results are significantly less then expected, comparisons can still be made between the other real image.

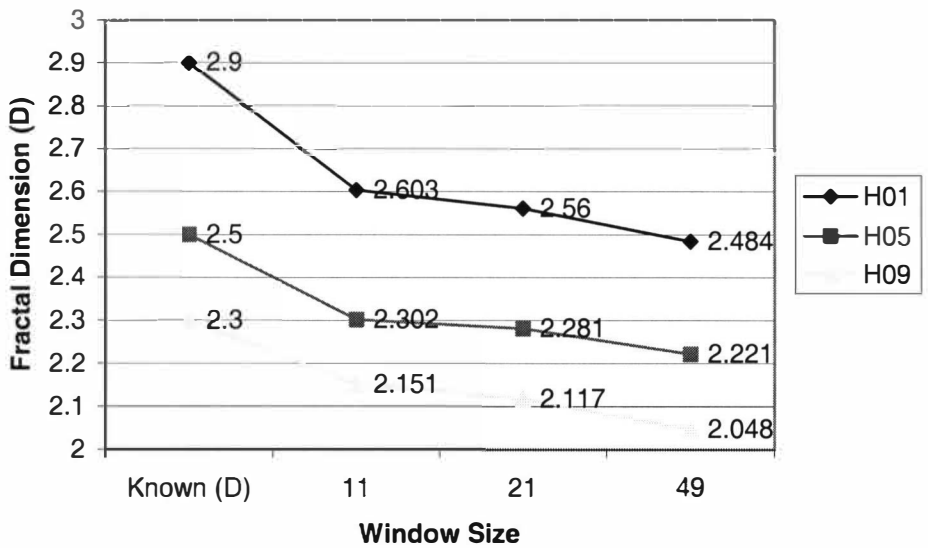


Figure 22. Average Fractal Dimension of Simulated Surfaces.

Holland, Michigan Characteristics

The orthographic photos taken in 1999 of Holland, Michigan were taken by GRW. Inc through a contract with the United States Army Corps of Engineers for multiple purposes (USACE and GRW, 1999). Orthographic photos resemble photos but contain properties of a map. Like maps, they have one scale and like photographs they show the terrain in actual detail. These two properties allow analysts to interpret the orthophoto but also obtain true distances, angles, and areas. The GIS Research Center at Western Michigan University obtained the images from a contract held with the United States Army Corps of Engineers in 2000 in order to measure bluff line erosion.

The image of Holland is made up of six one square kilometer tiles encompassing the lower southwest portion of Ottawa County and the northwest portion of Allegan county (Figure 23). This image of Holland, Michigan was selected because it contained a wide variety of land uses with various degrees of complexity. This area contains a lake, forest cover, and urban setting as well as several agricultural plots. The highway running north and south nearly splitting the image in half is US-31 and the highway running east and west is Michigan highway 21. The lake just west of US-31 is Lake Macatawa which is fed by the Macatawa River. In the Southeastern portion of the image below the urban areas a forest is visible and beyond that a few monoculture agricultural tracts can be identified.

The original six tiles were mosaiced together using the geospatial analysis product TNTmips (MicroImages, 2000). The original image at 15,183 X 9,685(Figure 24A) has 0.3 meter resolution. It was resampled to 1 meter at 4555 X 2906 (Figure 24B), 5 meter 911X 581(Figure 24C), 15 meter 304 X 194 (Figure 24D), and 30

meter resolution 152 X 97(Figure 24E). All of these images were resampled using cubic convolution and converted to a single band ERDAS GIS format that would allow it to be analyzed in Fractex.

The resampled 8-bit raster images of Holland allowed for meaningful comparisons to be made between images after using Fractex. The changes in resolution between images showed significant contrasts between the original and the fractal dimension output images. Unlike the ideal fractal surfaces, real world images such as Holland, Michigan represent various fractal dimensions at changing resolutions. As Holland is resampled towards less clarity, it becomes evident that the recognizable visual information obtained from the fractal dimension images becomes increasingly difficult to interpret.

The type and amount of information to be gathered using Fractex becomes more difficult to interpret as the window size is increased and the resolution is decreased. A general examination of Holland at 15-meter resolution shows the output images becoming more homogenous while recognizable features tend to diminish at increasing window sizes (Figure 25). Upon closer inspection of Figure 25A, Lake Macatawa and US-31 have easily identifiable features because of the low complexity of those surfaces. In the lower right hand corner of Figure25A there is a pocket of low image complexity where the forest stands and agricultural plots are located. Progressing to Figure 25B, Figure 25C, and Figure 25D, the low image complexity of agricultural tracts begins to expand. The presence of a new area in the northeast portion of the image becomes more dominant as an area if decreasing complexity. This area is the Muskegon state game area. In Figure25E and Figure 25F, the game area has a substantial smoothing influence throughout the northern portion of the

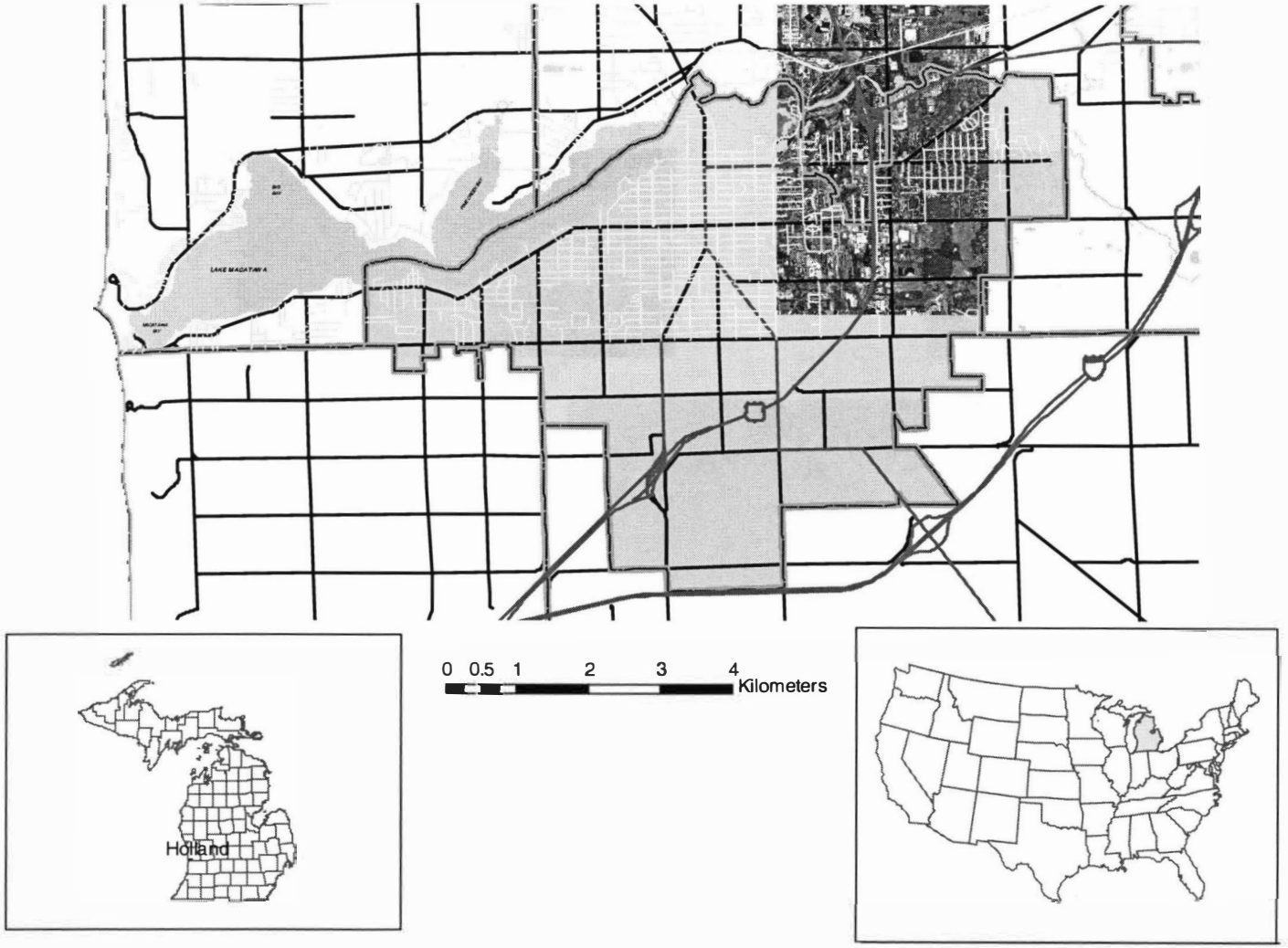
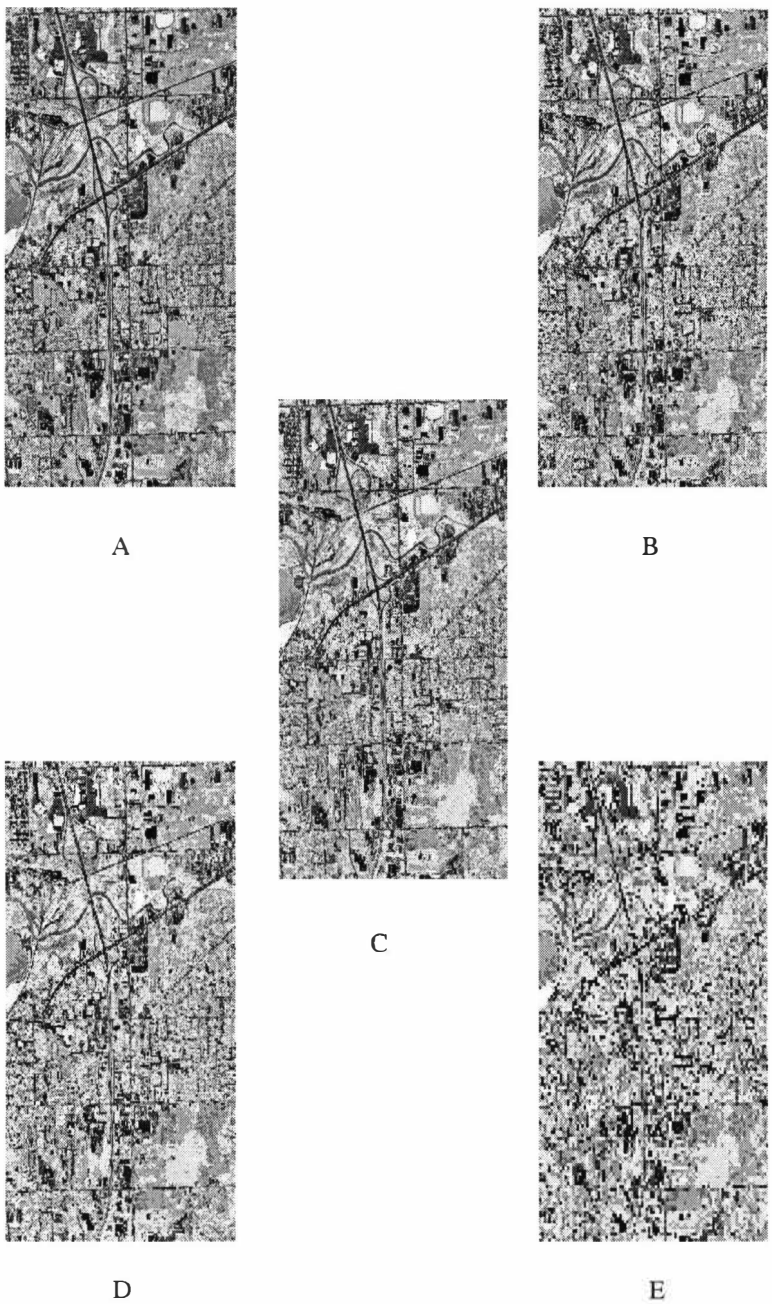


Figure 23. Holland, Michigan.



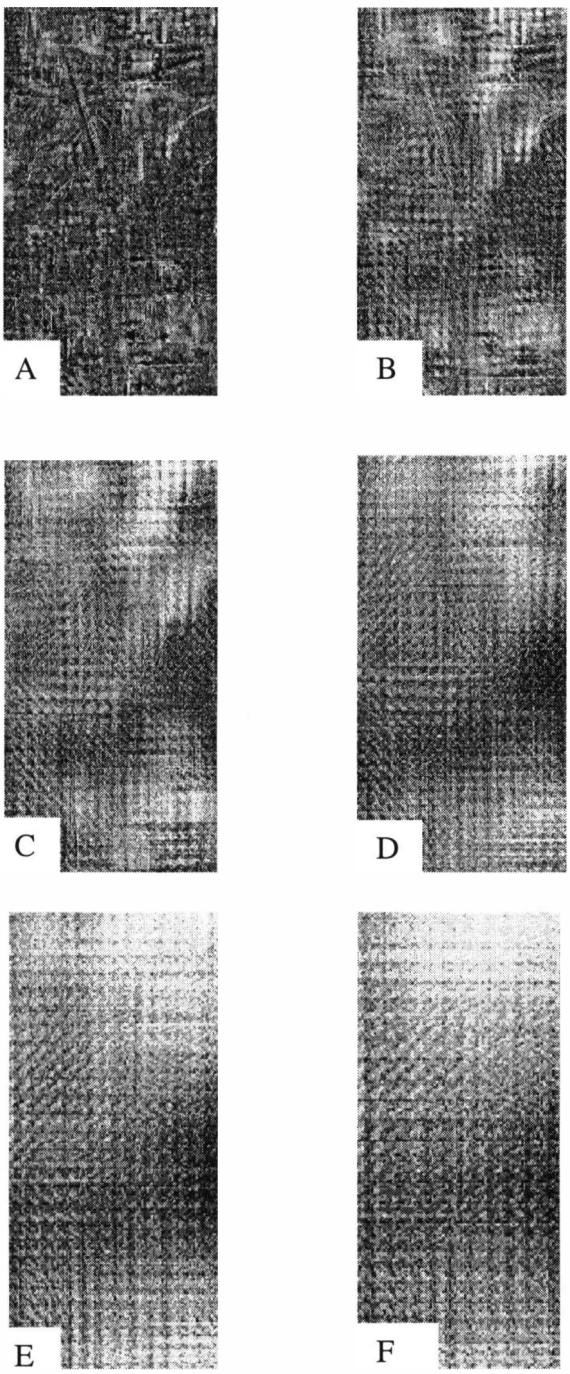
Legend. A= Holland original at 0.3 meter, B= Resampled to 1 meter, C= Resampled to 5 meters, D= Resampled to 15 meters, E= Resampled to 30 meters

Figure 24. Resampled Images of Holland, Michigan.

image. Overall, the observed average fractal dimension is lower because Fractex is a smoothing function.

The second trend involves the shrinking range of fractal dimension values obtained within an image. With small window sizes (11X11), a wide range of values (1.92-3.03) is seen in the 15 meter resolution of Holland that used a window size of 11 x 11. Comparing the 15 meter resolution using 49 x 49 window sizes (15hol_49) with a range of (2.03 – 2.59) reaffirms the notion that larger window sizes have a smoothing effect upon the output grid. In the case of 15hol_11, the fractal dimension is 2.555 while the dimension for 15hol_49 is 2.426. As the window size increases and the output grid becomes smaller, the fractal dimension value of an entire image tends to shrink and approach a single fractal dimension. For example, window sizes with subsequently larger increments in Holland_15 returned values (Figure 23) that approached a value near 2.450.

A third observed outcome was obtained upon examination of a continuous window size of 11 X 11 across three resampled Holland images. As images are resampled to include more actual ground area per pixel, the expected result would suggest that by maintaining a set window size a smoothing effect would result. However, Figure 23 shows that as images are resampled, they tend to exhibit properties of higher complexity. It is interesting to point out that Holland_15m and Holland_30m maintain close average fractal dimensions between them at low window sizes. As the window size increases to 49 X 49, there is a significant increase in the spread of the average fractal dimension between the two images.



Legend. A= 11x11 D= 2.596, B= 21x21 D =2.546, C= 35x35 D = 2.528, D= 51x51 D = 2.43, E= 75x75 D = 2.454, F= 101x101 D = 2.450

Figure 25. Fractal Grid Output from Holland, Michigan at 15 Meter Resolution.

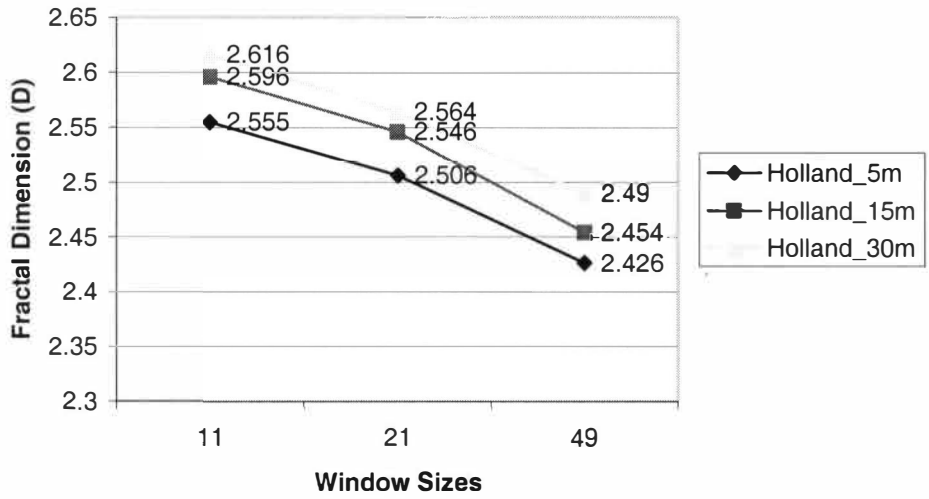


Figure 26. Average Fractal Dimension for Holland, Michigan.

DISCUSSION

With more than 500,000 users of ArcView GIS worldwide (ESRI, 2001), Fractex will possibly experience widespread adoption to assist texture classification. The use of this extension is accomplished with only the low cost of ArcView compared to that of other more costly expenses associated with other software platforms or stand-alone products. It is important to mention that this extension only works with versions of ArcView that are 3.0 and higher and must have the Spatial Analyst extension.

The real functionality of this utility is rooted in the simplicity of Fractex. Fractex offers users of numerous disciplines the ability to perform complex fractal analysis within a simple and easy to understand extension. This current version also has the capability to supplement the addition of more features. The incorporation of additional features for geospatial analysis, like Geary's C and Morans I (Cliff and Ord, 1981), will be easily integrated. Fractex presents an alternitive in the

development and the testing of practical applications of fractal analysis in terms of image texture. The potential usefulness of this technique will hopefully reach all manor of texture classification experts.

CHAPTER VII

CONCLUSIONS

Final Comments

The rationale, design, and major functions of the extension Fractex have been described as a tool for multiscaled analysis of landscape characteristics. The development and distribution of the extension Fractex offers another alternative to determining the fractal dimension of an image. The algorithm has shown that it functions adequately within ArcView to generate a grid image based upon the fractal dimensions of a moving window. The output grid, however, is a smaller area then the original and therefore caution is suggested when including target features near the edge. The boundary of the new grid is dependent upon the window size, window gap and the step size. The program also returned an average fractal dimension for entire grid image sets. These grid images and the average fractal dimension were compared with known fractal dimensions of simulated surfaces. The results indicated that there was a considerable loss of precision between the known and the expected values. Even though there was a significant drop, the average fractal dimensions of the simulated surfaces maintained a relatively uniform fractal dimension at large and small window sizes.

Based upon the notion that analysis of a real world image would likely provide average fractal dimension values lower then a global fractal dimension, comparisons were made only upon images of Holland, Michigan. The analysis of

Holland, Michigan examined the effects resampling of images had upon the fractal dimension. The evidence suggested that resampling to coarser resolutions caused the fractal dimension to rise. One resampled image of Holland, Michigan was studied to examine the effects of increasing window size and the complexity of an image.

A major difficulty associated with fractal analysis when applied to earth system remote sensing stems from the notion that different algorithms return different dimension values (Tate, 1998). Often there is variability associated within a single method resulting from various input parameters that affect the final fractal dimension. This work only examined the use of the triangular prism method at the local level. However, Fractex can be expanded to include other methods like the isarithm and variogram. Analyses of the output fractal dimensions require expert knowledge of the target system especially when attempting to draw major conclusions from the results. The emphasis upon expert knowledge is often overlooked in favor of the processes surrounding remote sensing and geographic information system research. In the case of fractal analysis, expert knowledge cannot be understated.

Further Research

There are at least three major areas that need to be examined in future research. First, do automation tools improve land cover classification? It is important to determine if Fractex aids in the providing real clues to quantifying texture. There needs to be further examination of other simulated surfaces to assess image complexity. Second, there is the need for further development involving multi-scale analysis and the effects it has upon texture classification. It is important that research

be performed to determine the boundaries of certain characteristics within an image. For example, determining quantitatively where the edge of an urban area meets a rural area is an important issue in land use management. Third, determining what is an optimal scale to perform image analysis upon the system, target or area is useful and can be developed. Knowing the resolution that can provides the most information to determine the most accurate fractal dimension is needed.

Perhaps the most limiting factor associated with the fractal dimension is the lack of understanding of what is being actually measured and the meanings of these values. It is critically important that analysts compare fractal results with results from spatial statistics to assess the usefulness and accuracy of the information obtained.

Appendix A

Avenue Source Code

```
'Name: FractalDimensionbyMovingWindow
'
'Title: Determine Fractal Dimension using moving windows
'
'Topics: Spatial Analyst
'
'Christopher Caird
'Western Michigan University.
'
'Requires: a View must be the active document, an integer grid theme must be
'the active theme.
'
'Self:
'
'Returns:


 'Check the availability of Spatial Analyst Extension

test=Extension.Find("Spatial Analyst")
if (test=NIL) then
msgbox.error("You must have the spatial analyst extension loaded","Clip by ~
        Moving Windows")
 return nil
end
```

```
'Get the View and selected Grid

theView=av.GetActiveDoc
theGTheme=theView.GetActiveThemes.Get(0)

if (theGTheme.Is(GTheme).Not) then
  msgbox.error("You must select the grid as the active theme.","Aborting")
  return nil
end


'Get the projection information

theProjection=theView.GetProjection
theDisplayUnits=theView.GetDisplay.GetUnits
theUnits=theView.GetUnits


'Get the attributes of the Grid

theGrid=theGTheme.GetGrid
if (theGrid.IsInteger.Not) then
  msgbox.error("You must select the integer grid for this analysis.","Aborting")
```

```
  return nil
end

theCellSize=theGrid.GetCellSize
theCellSize.SetFormat("d")
theExtent=theGrid.GetExtent

ImageRow = theExtent.GetBottom
ImageColumn = theExtent.GetRight

theOrigin=theExtent.ReturnOrigin
originx=theOrigin.getx
originy=theOrigin.gety



gridrows=theGrid.GetNumRowsAndCols.get(0)
gridcols=thegrid.GetNumRowsAndCols.get(1)


theSize=theExtent.ReturnSize
theExtentUR=Point.Make(originx+theSize.Getx,originy+theSize.Gety)
URx=theExtentUR.Getx
URy=theExtentUR.Gety
```

```
NODATA_value = -9999
NODATA_value = nodata_value.AsString



dftBand=1

Band=Msgbox.Input("Specify the Band Width of the LAN image (1-
7),"+NL+"Example: 1","Image",dftBand.AsString)
Band1=Band.AsString
if (Band.contains("aString" ))then
 Msgbox.Error("Band must be a Number.","")
 return nil
 end



'dftfirstRow=1

'firstRow=Msgbox.Input("Specify the first row,"+NL+"Example:
1","Image",dftfirstRow.AsString)
'firstRow1=firstRow.AsString
'if (firstRow.contains("aString" ))then
'  Msgbox.Error("Row must be a Number.","")
' return nil
```

```
' end
firstRow = 1

dftlastRow=10

lastRow=Msgbox.Input("Specify the last row,"+NL+"Example:
50","Image",dftlastRow.AsString)
lastRow1=lastRow.AsString
if (lastRow.contains("aString" ))then
 Msgbox.Error("Last Row must be a Number.","")
return nil
end

oddlist = {1,3,5,7,9}
lastdig = lastRow.right(1).AsNumber
chkval = oddlist.findbyvalue(lastdig)
'msgbox.info(chkval.asstring,"")

if (chkval = -1) then
 msgBox.error("This value is Even","")
 return nil
 end

lastRow = lastRow.AsNumber
```

```
'dftleftColumn=1

'leftColumn=Msgbox.Input("Specify the first column,"+NL+"Example:~
        1","Image",dftleftColumn.AsString)
'leftColumn1=leftColumn.AsString
'if (leftColumn.contains("aString" ))then
'  Msgbox.Error("Left Column must be a Number.","")
' return nil
' end
leftColumn = 1


dftrightColumn=10

rightColumn=Msgbox.Input("Specify the right column for window,"+NL+ ~
        "Example: 50","Image",dftrightColumn.AsString)
rightColumn1=rightColumn.AsString
if (rightColumn.contains("aString" ))then
  Msgbox.Error("Right Column must be a Number.","")
 return nil
 end

oddlist = {1,3,5,7,9}
lastdig = rightColumn.right(1).AsNumber
chkval = oddlist.findbyvalue(lastdig)
```

```
'msgbox.info(chkval.asstring,"")

if (chkval = -1) then
  msgBox.error("This value is Even","")
  return nil
  end

rightColumn=rightColumn.AsNumber


dftstepSize=9

stepSize=Msgbox.Input("What's the Step Size Needed in Order to Compute 'D'?:","
Step Size", dftstepSize.AsString)
stepSize=stepSize.AsNumber


dftwinGap = 1

winGap=Msgbox.Input("What's the Amount of Gap Between Windows :"+NL+"Gap
is Defined in Terms of number of Columns Between Successive
Windows("+theCellSize.AsString+").","  ",dftwinGap.AsString)
winGap=winGap.AsNumber

'  Obtain image location
```

```
dftimageLoc= "d:\fractal\h051024a"

imageLoc=Msgbox.Input("Enter the Name of the Image without the ~
        Extension"+NL+"and the name of the drive it is stored in."+NL+ ~
        "Example: a:\lkch84","Image",dftimageLoc.AsString)

if (Not (imageLoc.contains(":\"))) then
 Msgbox.Error("Location of image must include the drive it is stored in","")
 return nil
 end

if (imageLoc.contains(".")) then
 Msgbox.Error("Image Must not have Extension.","")
 return nil
 end


' Obtain extension

dftexten = "lan"
myList = { "lan", "gis", "dat", "bil"  }
if (myList = nil) then
   msgbox.info(myList++"You must choose one type.","  Aborting")
      end
```

```
myList = MsgBox.Choiceasstring (myList, "Choose the extension of the image.", ~
        dftexten.AsString)



' Fractal computation method

dftimageMethod = 1

imageMethod=Msgbox.Input("Chose the method for computing the Fractal ~
        dimension (0 - exponential, 1 - arithmetic),"+NL+ ~
        "Must be a 0 or 1. ","Image",dftimageMethod.AsString)
imageMethod1=imageMethod.AsString
if ((imageMethod1 = "0") or ( imageMethod1 = "1" )) then
  msu=0
else
  Msgbox.Error("Size must be a (1) or (0).","")
 return nil
end

'Information changes for text file header

gridcols = (((gridcols-rightColumn)/winGap.floor) + 1).setformat( "d" )
gridrows = (((gridrows-lastRow)/winGap.floor) + 1).setformat( "d" )
xorigin = originx+((1/2)*lastRow*theCellSize)
```

```
yorigin = originy + ((1/2) * lastRow*theCellSize)


' Write the text file header
    txtFileName = "d:\fractal\fractal.txt"
    theFN = txtFileName.AsFileName
    theTestFile = TextFile.Make(theFN, #FILE_PERM_WRITE)
    theList = {"line1","line2","line3","line4","line5"}
    theList = theList.AsString
    theTestFile.Write(theList,0)

    If( theTestFile = nil ) Then
     MsgBox.Error("Cannot open file:" + theFN.GetFullName, "")
     Exit
    End

    ' Write the file header and the Fractal
        Str2 = "ncols"
        Str3 = "nrows"
        Str4 = "xllcorner"
        Str5 = "yllcorner"
        Str6 = "cellsize"
        Str7 = "NODATA_value"

    theTestFile.Write(Str2, 5)
```

```
    theTestFile.Write(" " + gridcols.AsString, gridcols.AsString.Count +1)
    theTestFile.WriteElt(NL)

    theTestFile.Write(Str3, 5)
    theTestFile.Write(" " + gridrows.AsString, gridrows.AsString.Count +1)
    theTestFile.WriteElt(NL)

    theTestFile.Write(Str4, 9)
    theTestFile.Write(" " + xorigin.AsString, xorigin.AsString.Count +1)
    theTestFile.WriteElt(NL)

    theTestFile.Write(Str5, 9)
    theTestFile.Write(" " + yorigin.AsString, yorigin.AsString.Count +1)
    theTestFile.WriteElt(NL)

    theTestFile.Write(Str6, 8)
    theTestFile.Write(" " + theCellSize.AsString, theCellSize.AsString.Count +1)
    theTestFile.WriteElt(NL)

    theTestFile.Write(Str7, 12)
    theTestFile.Write(" " + NODATA_value.AsString, 6)
    theTestFile.WriteElt(NL)
theTestFile.Close

'Msgbox.info("The number of windows in ~
```

```
        longitude:"++vmaxNumx.AsString+NL+"The number of windows in ~
        latitude:"++vmaxNumy.AsString,"Information on window number")

OriginRow = FirstRow
OriginColumn = LeftColumn
BoxWidth = RightColumn - LeftColumn
BoxHeight = LastRow - FirstRow
FirstRow = FirstRow

'msgbox.info((ury-originy).asstring, (urx-originx).asstring)

StepV = ((ury-originy)/theCellSize - lastRow ) / WinGap
StepH = ((urx-originx)/theCellSize - RightColumn) / WinGap
StepV = StepV + 1
StepH = StepH + 1
StepV = StepV.floor
StepH = StepH.floor
'StepH will be used as paramter number 10 in our execute file.
'msgbox.info(stepv.asstring, steph.asstring)


  system.ExecuteSynchronous( "d:\fractal\MWFDTPM_OneGain.exe" + " " + ~
        imageLoc.AsString + " " + myList.AsString + " " + Band.AsString + " " + ~
        stepSize.AsString + " " + imageMethod.AsString + " " + StepV.AsString + ~
```

```
" " + StepH.AsString + " " + WinGap.AsString + " " + OriginRow.AsString ~
+ " " + OriginColumn.AsString + " " + BoxHeight.AsString + " " + ~
 BoxWidth.AsString)
```

Appendix B

C Programming Source Code

The C source code for the Moving Window by Fractal Dimension of Triangular Prism (MWFDTPM.exe) is listed below. The executable must contain all modules listed below in order operate properly. The modules are numbered and given a name for reference. When recompiling the code make sure to delete out the numbered rows that are underlined.

1. globals.cpp

```
#include <stdio.h>

// DANGER! This same structure is declared in erdas.h, I copied it here trying to ~
get the program to compile.
// Don't leave it this way, fix it!/FIXED IT.
#define ERDAS_HEAD_SIZE 128
typedef struct _ERDAS {
    char head[6];
    unsigned short ipack;
    unsigned short nbands;
    unsigned char unused0[6]; /*unsigned short unused0[3];*/
    unsigned long icols;
    unsigned long irows;
    long xstart;
```

```
    long ystart;
    unsigned char unused1[56]; /* int unused1[14];*/
    unsigned short maptyp;
    unsigned short nclass;
    unsigned char unused2[20];
    float xmap;
    float ymap;
    float xcell;
    float ycell;
} ERDAS;

ERDAS ers;

int up,below,left,right;
int z_interval,walk,direction;
int no_is_line;
float zmin,zmax;
//unsigned short int zmin,zmax;
float z_contour[256],b[256],r[256],dimen[256];
int flag[256];
```

```
int **a;



FILE   *fp;
FILE   *fout;
char head[6];
unsigned short ipack;
unsigned short nbands;
unsigned long icols;
unsigned long irows;
long xstart;
long ystart;
unsigned short maptyp;
unsigned short nclass;
float xmap;
float ymap;
float xcell;
float ycell;

float xmin;
float ymin;
float xmax;
float ymax;
```

```
unsigned short int **z;

unsigned char unused0[6]; /*unsigned short unused0[3];*/
unsigned char unused1[56]; /* int unused1[14];*/
unsigned char unused2[20];

float *xx, *yy, *zz;
float *dist, *hvar;
float *sumzd,*gb;
int *ncase;
int num,num_group;
int np;
int select_points,band;
float rang;

double area[100], resolution[100];
int step_num, method;
```

2. MWFDTP.DSW

```
// This is a profiling version of the main program. It isn't used under normal
// circumstances, but I wanted to keep it around.

/*
 * MWFDTPM
 *
 * Moving Window Fractal Dimension by Triangular Prism Method
 *
 * Revision History:
 *   04/27/2001 EJA Created for Chris Caird from "testtriangular" by Wei Zhao.
 *   05/03/2001 EJA Moved to a C++ project.
 *   06/06/2001 EJA Did final cleanup for first release.
 */



// If defined, the gain will be calculated for the whole image instead of per-window.
#define ONEGAIN
// If defined, progress through the calculation will be shown
#define SHOWPROGRESS



char szVersion[] = "1.1.5";



#include "triangular.h"



main (int argc, char *argv[])
  {
  long steps;
  long StepV, StepH;
  long WinGap;
  long OriginRow, OriginColumn;
  long BoxHeight, BoxWidth;
  long Row, Column;
```

```
  // Print the banner
#ifdef ONEGAIN
  printf("MWFDTPM v%s - Gain calculated once for the whole image.\n", ~
        szVersion);
#else
  printf("MWFDTPM v%s - Gain calculated for each window.\n", szVersion);
#endif
  printf("Moving Window Fractal Dimension by Triangular Prism  ~
        Method\nCopyright(c) 2001 Western Michigan University Geography ~
        Department\nBased upon the algorithim developed by Nina Lam ~
        (Louisana State University)\n\n");

  if(argc != 13)
    {
    printf("USAGE: %s <inputfile> <option> <band> <steps> <method>  ~
        <StepV> <StepH> <WinGap> <OriginRow> <OriginColumn>  ~
        <BoxHeight> <BoxWidth>\n", argv[0]);
    printf("      <inputfile> is an image file without the extension.\n");
    printf("      <option> specifies a type to be calculated.\n");
    printf("        lan - a Erdas format image.\n");
    printf("        gis - a Erdas format image.\n");
    printf("        dat - a ASCII format data.\n");
    printf("        bil - Band Interleaved by Line.\n");
    printf("      <band> The band to read from the input file.\n");
    printf("      <steps> The number of steps to iterate the prism size over.\n");
    printf("      <method> The method to increase prism size by:  ~
        0 is doubling, anything else is incrementing by 1.\n");
    printf("      <StepV> The number of rows of FDs to calculate.\n");
    printf("      <StepH> The number of columns of FDs to calculate.\n");
    printf("      <WinGap> Increment between window edges for each  ~
        FD calculation.\n");
    printf("      <OriginRow> The starting row.\n");
    printf("      <OriginColumn> The starting column.\n");
    printf("      <BoxHeight> The height of the window.\n");
    printf("      <BoxWidth> The width of the window.\n");
    exit(1);
    }

  band=atoi(argv[3]);
  steps=atoi(argv[4]);
  method=atoi(argv[5]);

  //TODO: It should be possible to calculate StepV and StepH from the ~
```

```
        image dimensions, OriginRow, OriginColumn, BoxHeight,  ~
       BoxWidth, and WinGap...
StepV = atoi(argv[6]);
StepH = atoi(argv[7]);
WinGap = atoi(argv[8]);
OriginRow = atoi(argv[9]);
OriginColumn = atoi(argv[10]);
BoxHeight = atoi(argv[11]);
BoxWidth = atoi(argv[12]);

char szDrive[_MAX_DRIVE];
char szDir[_MAX_DIR];
char *pszFullPath = _fullpath(NULL, argv[1], 0);
if (!pszFullPath)
  {
  fprintf(stderr, "ERROR: The filename \"%s\" is invalid\n", argv[1]);
  exit(-1);
  }
_splitpath(pszFullPath, szDrive, szDir, NULL, NULL);

FILE *pfDebug;
char szFile[_MAX_PATH];
_makepath(szFile, szDrive, szDir, "debug", ".txt");
if (!(pfDebug = fopen(szFile, "w")))
  {
  fprintf(stderr, "ERROR: The debug file \"%s\" could not be opened  ~
       for writing.\n", szFile);
  exit(-1);
  }

fprintf(pfDebug, "Run Parameters\n");
fprintf(pfDebug, "data file:\t%s\n", pszFullPath);
fprintf(pfDebug, "data type:\t%s\n", argv[2]);
fprintf(pfDebug, "band:    \t%d\n", band);
fprintf(pfDebug, "steps:   \t%d\n", steps);
fprintf(pfDebug, "method:  \t%d\n", method);
fprintf(pfDebug, "StepV:   \t%d\n", StepV);
fprintf(pfDebug, "StepH:   \t%d\n", StepH);
fprintf(pfDebug, "WinGap:  \t%d\n", WinGap);
fprintf(pfDebug, "OriginRow:\t%d\n", OriginRow);
fprintf(pfDebug, "OriginCol:\t%d\n", OriginColumn);
fprintf(pfDebug, "BoxWidth: \t%d\n", BoxWidth);
fprintf(pfDebug, "BoxHeight:\t%d\n", BoxHeight);
fprintf(pfDebug, "\nup\tbelow\tleft\tright\tNumStps\tR-Sq\tF.D.\n");
```

```
  FILE *pfFractalDimension;
  _makepath(szFile, szDrive, szDir, "fractal", ".txt");
  if (!(pfFractalDimension = fopen(szFile, "a")))
    {
    fprintf(stderr, "ERROR: The fractal dimension file \"%s\" could not be opened for
writing.\n", szFile);
    fclose(pfDebug);
    exit(-1);
    }

  double dResultSum = 0.0;
  double dRSquaredSum = 0.0;
  long cResults = 0;

  Open_File(pszFullPath, argv[2], band);
#ifdef ONEGAIN
  Find_Z_MINMAX(0, irows - 1, 0, icols - 1); // Only calculate one gain
#endif
  for (Row = 0; Row < StepV; ++Row)
    {
    up = Row * WinGap + (OriginRow - 1);
    below = up + BoxHeight;

#ifdef SHOWPROGRESS
    putchar('*');
#endif
    for (Column = 0; Column < StepH; ++Column)
      {
      float fResult;
      float fRSquared;

      left = Column * WinGap + (OriginColumn - 1);
      right = left + BoxWidth;

      fprintf(pfDebug, "%d\t%d\t%d\t%d\t", up + 1, below + 1, left + 1, right + 1);

      // Perform calculations for the current window
#ifndef ONEGAIN
      Find_Z_MINMAX(up, below, left, right); // Calculate one gain per window
#endif
      Compute_triangular(steps, method);
      Line_Fit(pfDebug, fResult, fRSquared);
```

```
        // Update the data that is used to calculate averages
        dResultSum += fResult;
        dRSquaredSum += fRSquared;
        ++cResults;

        // Write the current window's information to the output files
        fprintf(pfFractalDimension, "%.3f ", fResult);
        fprintf(pfDebug, "%.3f\n", fResult);
        }
      fprintf(pfFractalDimension, "\n");
      }

#ifdef SHOWPROGRESS
  putchar('\n');
#endif

  // Calculate and write averages to the output file(s).
  fprintf(pfDebug, "\nAverage fractal dimension %0.3lf\n", dResultSum/cResults);
  fprintf(pfDebug, "\nAverage R Squared %0.3lf\n", dRSquaredSum/cResults);

  Close_File();
  fclose(pfDebug);
  fclose(pfFractalDimension);
  free(pszFullPath);

  return 0;
  }
```

3. triangular.h

```
/*-------------------------------------------------------------
 File:triangular.h
 Date: 7/26/95
 Purpose:Computation of Fractal Dimension of remotely-sensed images
        using a modified form of the triangular prism surface area
        method.

 Author: Wei Zhao
-------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>
```

```
#include "isarithm.h"
#include "variogram.h"
#include "erdas.h"


extern double area[100], resolution[100];
extern int step_num, method;


void Compute_triangular(int steps, int method);
void Line_Fit(FILE *pfDebug, float& fFractalDimension, float& fRSquared);
```

4. erdas.h

```
/*******************************************************
 File: erdas.h
 Date: July 5, 1995
 Purpose: Erdas LAN/GIS image file head information

 Author: Wei Zhao
*******************************************************/

#define ERDAS_HEAD_SIZE 128
typedef struct _ERDAS {
      char head[6];
      unsigned short ipack;
      unsigned short nbands;
      unsigned char unused0[6]; /*unsigned short unused0[3];*/
      unsigned long icols;
      unsigned long irows;
      long xstart;
      long ystart;
      unsigned char unused1[56]; /* int unused1[14];*/
      unsigned short maptyp;
      unsigned short nclass;
      unsigned char unused2[20];
      float xmap;
      float ymap;
      float xcell;
```

```
    float ycell;
} ERDAS;

extern ERDAS ers;

extern FILE   *fp;
extern FILE   *fout;
extern char head[6];
extern unsigned short ipack;
extern unsigned short nbands;
extern unsigned long icols;
extern unsigned long irows;
extern long xstart;
extern long ystart;
extern unsigned short maptyp;
extern unsigned short nclass;
extern float xmap;
extern float ymap;
extern float xcell;
extern float ycell;

extern float xmin;
extern float ymin;
extern float xmax;
extern float ymax;
extern unsigned short int **z;

extern unsigned char unused0[6]; /*unsigned short unused0[3];*/
extern unsigned char unused1[56]; /* int unused1[14];*/
extern unsigned char unused2[20];

extern int Open_Erdas(char *erdas_path);
extern int Read_Erdas_HDR();
extern int Read_Erdas(int band, unsigned short int **z);
extern int Close_Erdas();
extern unsigned short int  **TwoArrayAlloc(int row, int col);
extern void TwoArrayFree(unsigned short int **x);
extern int **TwoArrayAlloc_int(int row, int col);
extern void TwoArrayFree_int(int **x);
extern float **TwoArrayAlloc_float(int row, int col);
extern void TwoArrayFree_float(float **x);
extern char **TwoArrayAlloc_char(int row, int col);
extern void TwoArrayFree_char(char **x);
```

5. fractal_isarithm.c

```
/*-------------------------------------------------------
 File: fractal_isarithm.c
 Date: July 12, 1995
 Purpose:  The isarithm method for fractal surface measurement

 Author: Wei Zhao
-------------------------------------------------------*/

#include "isarithm.h"
#include "variogram.h"
#include "erdas.h"

/*-------------------------------------------------------
Find image Z value and calculate the number of isarithm
lines
-------------------------------------------------------*/
void Find_Z_MINMAX (int up, int below, int left, int right)
  {
  int i,j;
  float zTemp;

  zmin = 9999.0;
  zmax = -999.0;

  for (j = up; j <= below; j++)
    for (i = left; i <= right; i++)
      {
      zTemp = z[j][i];

      if (zTemp < zmin)
        zmin = zTemp;

      if (zTemp > zmax)
        zmax = zTemp;
      }

  //SHHH printf("MINIMUM Z = %f\n",zmin);
  //SHHH printf("MAXIMUM Z = %f\n",zmax);
```

```
  return;
  }

/*-------------------------------------------------------
 Calculate every isarithm
-------------------------------------------------------*/
void Compute_Isarithm(int z_interval, int walk, int direction)
{
int max_walk;
int i,j,k,ii,jj,zi,zj;
int icnt,in_line,out_line;
double d1,d2,r1,bsum,rsum;
int is_line;
int no_of_rows, no_of_cols;
int sum;
int no_of_walk;
char **z_count;
float x,y,sumx,sumy,sumxy,sqx,sqy;
float avgbc, average_b,average_r,average_dimen;

no_is_line = ((zmax-zmin)/z_interval) - .5;
max_walk=pow(2,walk-1);
//SHHH printf("NO. OF ISARITHM LINES = %d\n",no_is_line);
//SHHH printf("MAXIMUM WALK SIZE = %d\n",max_walk);

no_of_rows=below-up+1;
no_of_cols=right-left+1;

//printf("no_of_rows %d, no_of_cols %d\n",no_of_rows, no_of_cols);

z_count=TwoArrayAlloc_char(no_of_rows, no_of_cols);
a=TwoArrayAlloc_int(no_is_line,walk);


for(is_line=0;is_line<no_is_line;is_line++)
{
 z_contour[is_line]=zmin+(is_line+1)*z_interval;
 for(i=up,zi=0;i<=below,zi<no_of_rows;i++,zi++){
  for(j=left,zj=0;j<=right,zj<no_of_cols;j++,zj++)
  {
    if(z[i][j]<=z_contour[is_line]) z_count[zi][zj]='1';
    else z_count[zi][zj]='0';
   }
  }
```

```
sumx=sqx=sumy=sumxy=sqy=0;
for(k=0;k<walk;k++)
{
 sum = 0;
 no_of_walk=pow(2,k);  //sampling method, exponentially!
 if((direction==0)||(direction==2))
 {
  for(i=0;i<no_of_rows;i++)
    for(j=0;j<no_of_cols;j+=no_of_walk)
    {
        jj=j+no_of_walk;
        if(jj>=no_of_cols) break;
        if(z_count[i][j]!=z_count[i][jj]) sum++;
     }
 }

 if((direction==1)||(direction==2))
 {
  for(j=0;j<no_of_cols;j++)
    for(i=0;i<no_of_rows;i+=no_of_walk){
        ii=i+no_of_walk;
        if(ii>=no_of_rows) break;
        if(z_count[i][j]!=z_count[ii][j]) sum++;
     }
  }

  if(sum==0)
    break;
  else
  {
   switch(direction)
   {
    case 0: avgbc=(float)sum/no_of_rows;break;
    case 1: avgbc=(float)sum/no_of_cols;break;
    case 2: avgbc=(float)sum/(no_of_rows+no_of_cols);break;
    default: break;
   }
   a[is_line][k]=sum;
   x=log10(no_of_walk);
   y=log10(avgbc);
   sumx+=x;
   sumy+=y;
   sumxy+=x*y;
   sqx+=x*x;
```

```
    sqy+=y*y;
   }
  }
  if(sum==0)
  {
   flag[is_line]=0;
   b[is_line]=0.0;
   r[is_line]=0.0;
   dimen[is_line]=0.0;
   continue;
  }

  flag[is_line]=1;
  icnt = 0;
  for(k=0;k<walk;k++)
    if(a[is_line][0]==a[is_line][k]) icnt++;

  if(walk!=icnt)
  {
   d1=sumxy-((sumx*sumy)/walk);
   d2=sqx-((sumx*sumx)/walk);
   r1=d1/sqrt(d2*(sqy-sumy*sumy/walk));
   b[is_line]=d1/d2;
   r[is_line]=r1*r1;
   dimen[is_line]=2-b[is_line];
  }
  else
  {
   b[is_line]=0.0;
   r[is_line]=0.0;
   dimen[is_line]=0.0;
  }
}

bsum=0.;
rsum=0.;
in_line=no_is_line;
//!!we need printf("  ****************************************\n");
//!! we need printf("  Contour  Z_value    B       D     RSQ\n");

for(is_line=0;is_line<no_is_line;is_line++)
{
//!!we need printf("  %4d   %8.2f
%8.4f%8.4f%8.4f\n",is_line,z_contour[is_line],b[is_line],dimen[is_line],r[is_line]);
```

```
// if(flag[is_line]!=0&&r[is_line]>0.7&&r[is_line]<=1.0)
 if(flag[is_line]!=0&&r[is_line]>=0.9&&r[is_line]<=1.0)
 {
   bsum += b[is_line];
   rsum += r[is_line];
 }
 else
  in_line--;
}
        out_line=no_is_line-in_line;
        //SHHH printf("  ****************************************\n");
        //SHHH printf("  NO. OF ISARITHM LINES INCLUDED = %d\n",in_line);
        //SHHH printf("  NO. OF ISARITHM LINES NOT INCLUDED =
%d\n",out_line);

if(in_line){
        average_b=bsum/in_line;
        average_r=rsum/in_line;
        average_dimen=2.-average_b;
        //SHHH printf("  Slope = %f\n",average_b);
        //SHHH printf("  R-SQ = %f\n",average_r);
        //SHHH printf("  Dimension = %f\n",average_dimen);
}
else {
        //SHHH printf("  Slope = N/A\n");
        //SHHH printf("  R-SQ = N/A\n");
        //SHHH printf("  Dimension = N/A\n");
}
//SHHH printf("  ****************************************\n\n");
//printf("    Calculation by = %s\n",argv[4]);

Close_File();
TwoArrayFree_char(z_count);
return;
}

/*-------------------------------------------------------------
 Summary the every isarithm the result
-------------------------------------------------------------*/
void Summary_Isarithm(int walk_interval)
{
int is_line;
int k;
int cell_size[20];
```

```
for(is_line=0;is_line<no_is_line;is_line+=walk_interval)
{
 //SHHH ("  Enter isarithm line number ===> %d\n",is_line);
 //SHHH printf("                    Total\n");
 //SHHH printf("Isarithm line    No. of boundary cells   Walk size\n");
 //SHHH printf("-------------    ----------------------  ---------\n");

 for(k=0;k<walk;k++)
 {
  cell_size[k]=pow(2,k);
  //SHHH printf("   %8.2f         %6d
%2d\n",z_contour[is_line],a[is_line][k],cell_size[k]);
 }

 //SHHH switch(direction)
 //SHHH {
  //SHHH case 0: printf("by columns\n");break;
  //SHHH case 1: printf("by rows\n");break;
  //SHHH case 2: printf("by columns and rows\n");break;
  //SHHH default: break;
 //SHHH }
 //SHHH printf(" D = %6.4f\n",dimen[is_line]);
 //SHHH printf(" R = %8.6f\n",r[is_line]);
 //SHHH printf("  Summary  %d  of %d\n", is_line, no_is_line);

}

return;
}
/*-------------------------------------------------------------
 Close_Isarithm
-------------------------------------------------------------*/
void Close_Isarithm()
{
TwoArrayFree_int(a);
return;
}
```

6. isarithm.h

```
/*-------------------------------------------------------
 File: isarithm.h
```

```
 Date: July 12, 1995
 Author: Wei Zhao
-----------------------------------------------------------*/

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

//#define WHITE 1
//#define BlACK 0

extern int up,below,left,right;
extern int z_interval,walk,direction;
extern int no_is_line;
extern float zmin,zmax;
//unsigned short int zmin,zmax;
extern float z_contour[256],b[256],r[256],dimen[256];
extern int flag[256];
extern int **a;

void Find_Z_MINMAX(int up,int below,int left,int right);
void Compute_Isarithm(int interval, int walk, int direction);
void Summary_Isarithm(int walk_interval);
void Close_Isarithm();
```

7. read_erdas.c

```
/*********************************************************
File: read_erdas.c
Date: July 5, 1995
Purpose: Read Erdas LAN/GIS format image file

Author: Wei Zhao
*********************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
```

```
#include "erdas.h"
#include "variogram.h"

//unsigned short int zmin=999, zmax=0;
/*****************************************************
 Open_Erdas(): Open Erdas LAN/GIS files
 *****************************************************/
int Open_Erdas(char *path)
{
if((fp=fopen(path,"rb"))==NULL){
  printf("Cannot find the input file: %s.\n",path);
  exit(1);
}
return 0;
}

/*****************************************************
Close_Erdas(): Close opened Erdas image file
*****************************************************/
int Close_Erdas()
{
TwoArrayFree(z);
fclose(fp);
return 0;

}

/*****************************************************
Read header information of Erdas Lan/Gis format image
*****************************************************/
int Read_Erdas_HDR()
{
fseek(fp,0,SEEK_SET);
fread(head,sizeof(char),6,fp);
//fseek(fp,6,SEEK_SET);

fread(&ipack,sizeof(short),1,fp);
fread(&nbands,sizeof(short),1,fp);

fread(unused0,sizeof(char),6,fp);
//fseek(fp,16,SEEK_SET);
fread(&icols,sizeof(long),1,fp);
fread(&irows,sizeof(long),1,fp);
fread(&xstart,sizeof(long),1,fp);
```

```
fread(&ystart,sizeof(long),1,fp);

fread(unused1,sizeof(char),56,fp);
//fseek(fp,88,SEEK_SET);
fread(&maptyp,sizeof(short),1,fp);
fread(&nclass,sizeof(short),1,fp);

fread(unused2,sizeof(char),20,fp);
//fseek(fp,112,SEEK_SET);
fread(&xmap,sizeof(float),1,fp);
fread(&ymap,sizeof(float),1,fp);
fread(&xcell,sizeof(float),1,fp);
fread(&ycell,sizeof(float),1,fp);

xmin=xmap;
ymax=ymap;
xmax=xmin+xcell*icols;
ymin=ymax-ycell*irows;

// !!we need
printf("The Erdas image head information:\n");
switch(ipack) {
        case 1: printf("IPACK:4 bits/pixel\n");break;
        case 2: printf("IPACK:16 bits/pixel\n");break;
        default: printf("IPACK: 8 bits/pixel\n");break;
}
printf("NBANDS:%d bands\n",nbands);
printf("ICOLS:%ld pixels\n",icols);
printf("IROWS:%ld pixels\n",irows);
printf("xstart:%ld \n",xstart);
printf("ystart:%ld \n",ystart);
printf("maptype:%d \n",maptyp);
printf("nclass:%d \n",nclass);
printf("xmap:%f \n",xmap);
printf("ymap:%f \n",ymap);
printf("xcell:%f \n",xcell);
printf("ycell:%f \n",ycell);
printf("\n");

return 0;
}
```

```
//TODO: If the requested band doesn't exist in the file, throw error.
int Read_Erdas(int band, unsigned short int **z)
  {
  unsigned int i, nRow;
  unsigned char *buf;

  buf = (unsigned char *)malloc(icols * sizeof(unsigned char));

  fseek(fp,128,SEEK_SET);

  int cUnusedBandsAfterDesiredBand = 0;
  int cUnusedBandsBeforeDesiredBand = band - 1;

  for (nRow = 0; nRow < irows; nRow++)
    {
    int cBandsToSkip;

    cBandsToSkip = cUnusedBandsBeforeDesiredBand +
cUnusedBandsAfterDesiredBand;
    cUnusedBandsAfterDesiredBand = nbands - band;

    fseek(fp, cBandsToSkip * icols, SEEK_CUR);
    fread(buf, 1, icols, fp);

    for (i = 0; i < icols; i++)
      z[nRow][i] = (int) buf[i]; //Just BYTE not Short, or Float format
    }

  free((unsigned char *)buf);
  return 0;
  }



/************************************************
Allocate dynamic 2-D array
************************************************/
unsigned short int **TwoArrayAlloc(int row, int col)
{
 unsigned short int *x, **y;
 int n;
 x=(unsigned short int *)calloc(row*col, sizeof(short int));
 y=(unsigned short int **)calloc(row, sizeof(short int *));
 for(n=0;n<row;++n)
```

```
    y[n]=&x[col*n];
  return y;
}

void TwoArrayFree(unsigned short int **x)
{
  free(x[0]);
  free(x);
}

char **TwoArrayAlloc_char(int row, int col)
{
  char *x, **y;
  int n;
  x=(char *)calloc(row*col, sizeof(char));
  y=(char **)calloc(row, sizeof(char *));
  for(n=0;n<row;++n)
    y[n]=&x[col*n];
  return y;
}

void TwoArrayFree_char(char **x)
{
  free(x[0]);
  free(x);
}

int **TwoArrayAlloc_int(int row, int col)
{
  int *x, **y;
  int n;
  x=(int *)calloc(row*col, sizeof(int));
  y=(int **)calloc(row, sizeof(int *));
  for(n=0;n<row;++n)
    y[n]=&x[col*n];
  return y;
}

void TwoArrayFree_int(int **x)
{
  free(x[0]);
  free(x);
}
```

```
float **TwoArrayAlloc_float(int row, int col)
{
 float *x, **y;
 int n;
 x=(float *)calloc(row*col, sizeof(float));
 y=(float **)calloc(row, sizeof(float *));
 for(n=0;n<row;++n)
   y[n]=&x[col*n];
 return y;
}

void TwoArrayFree_float(float **x)
{
 free(x[0]);
 free(x);
}
```

8. variogram.h

```
/*-------------------------------------------------------
 File: variogram.h
 Date: July 5, 1995
 Purpose: The variogram method for fractal surface measurement

 Author: Wei Zhao - 01/28/95
-------------------------------------------------------*/
extern float *xx, *yy, *zz;
extern float *dist, *hvar;
extern float *sumzd,*gb;
extern int *ncase;
extern int num,num_group;
extern int np;
extern int select_points,band;
extern float rang;
extern unsigned short int **z;

void Open_File(char *input, char *filetype, int band);
void Sampling_Method_1(int select_points);
void Sampling_Method_2(int select_points);
void Sampling_Method_3(int select_points, int sub_numbers);
void Sampling_Cov(char *outputfile);
extern void Close_File(void);
void Dist_Vari(int num_group);
void Write_Variogram(char *outputfile);
```

```
void Vari_Fractal(char *outputfile, int num_group, int break_point1, ~
        int break_point2);
void Vari_Fractal_Max(char *outputfile, int num_group);
void Close_Variogram();
extern void Read_Dat(char *input);
extern void Read_Bil(char *input);
```

9. Microsoft developer studio_1

```
Microsoft Developer Studio Workspace File, Format Version 6.00
# WARNING: DO NOT EDIT OR DELETE THIS WORKSPACE FILE!

###############################################################################

Project: "MWFDTPM"=.\MWFDTPM.dsp - Package Owner=<4>

Package=<5>
{{{
}}}

Package=<4>
{{{
}}}

###############################################################################

Global:

Package=<5>
{{{
}}}

Package=<3>
{{{
}}}

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

10. read_dat

```
#include <stdio.h>
#include <stdlib.h>

#include "variogram.h"
#include "erdas.h"

void Read_Dat(char *input)
{
int i,j;
int zz;
unsigned char *buf;

if((fp = fopen(input,"rt"))==NULL)
{
 fprintf(stderr,"Cannot find the input file: %d\n",input);
 exit(1);
}

fscanf(fp,"%d %d",&irows,&icols);
xmin=0.;
ymax=irows-1;
//z=TwoArrayAlloc(irows,icols);
buf=(unsigned char *)malloc(icols*sizeof(unsigned char));
for(j=0;j<irows;j++){
 for(i=0;i<icols;i++)
 {
 fscanf(fp,"%d\n",&zz);
 //z[j][i]=zz;
//  printf("z[%d][%d]=%d\n",j,i,z[j][i]);
 buf[i]=(unsigned char)zz;
 }
 fwrite(buf,1,icols,fout);
}

//fclose(fp);
return;
}


void Read_Bil(char *input)
{
int i,j;
int zz;
unsigned char *buf;
```

```
if((fp = fopen(input,"r"))==NULL)
{
  fprintf(stderr,"Cannot find the input file: %d\n",input);
  exit(1);
}
/*
fscanf(fp,"%d %d",&irows,&icols);
xmin=0.;
ymax=irows-1;
*/

//irows=1600;
//icols=1600;

buf=(unsigned char *)malloc(icols*sizeof(unsigned char));
z=TwoArrayAlloc(irows,icols);

for(j=0;j<irows;j++){
  fread(buf,sizeof(char),icols,fp);
  for(i=0;i<icols;i++)
  {
  z[j][i]=buf[i];
//  printf("z[%d][%d]=%d\n",j,i,z[j][i]);
  }
}
fclose(fp);
return;

}
```

11. Microsoft developer studio_2

```
# Microsoft Developer Studio Project File - Name="MWFDTPM" –  ~
        Package Owner=<4>
# Microsoft Developer Studio Generated Build File, Format Version 6.00
# ** DO NOT EDIT **

# TARGTYPE "Win32 (x86) Console Application" 0x0103

CFG=MWFDTPM - Win32 Debug
!MESSAGE This is not a valid makefile. To build this project using NMAKE,
!MESSAGE use the Export Makefile command and run
!MESSAGE
```

```
!MESSAGE NMAKE /f "MWFDTPM.mak".
!MESSAGE
!MESSAGE You can specify a configuration when running NMAKE
!MESSAGE by defining the macro CFG on the command line. For example:
!MESSAGE
!MESSAGE NMAKE /f "MWFDTPM.mak" CFG="MWFDTPM - Win32 Debug"
!MESSAGE
!MESSAGE Possible choices for configuration are:
!MESSAGE
!MESSAGE "MWFDTPM - Win32 Release" (based on "Win32 (x86) Console
Application")
!MESSAGE "MWFDTPM - Win32 Debug" (based on "Win32 (x86) Console
Application")
!MESSAGE

# Begin Project
# PROP AllowPerConfigDependencies 0
# PROP Scc_ProjName ""
# PROP Scc_LocalPath ""
CPP=cl.exe
RSC=rc.exe

!IF  "$(CFG)" == "MWFDTPM - Win32 Release"

# PROP BASE Use_MFC 0
# PROP BASE Use_Debug_Libraries 0
# PROP BASE Output_Dir "Release"
# PROP BASE Intermediate_Dir "Release"
# PROP BASE Target_Dir ""
# PROP Use_MFC 0
# PROP Use_Debug_Libraries 0
# PROP Output_Dir "Release"
# PROP Intermediate_Dir "Release"
# PROP Target_Dir ""
# ADD BASE CPP /nologo /W3 /GX /O2 /D "WIN32" /D "NDEBUG"  ~
        /D "_CONSOLE" /D "_MBCS" /YX /FD /c
# ADD CPP /nologo /GB /W3 /GX- /O2 /D "WIN32" /D "NDEBUG"  ~
        /D "_CONSOLE" /D "_MBCS" /FR /YX /FD /c
# ADD BASE RSC /l 0x409 /d "NDEBUG"
# ADD RSC /l 0x409 /d "NDEBUG"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
LINK32=link.exe
```

```
# ADD BASE LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib ~
        comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib ~
        uuid.lib odbc32.lib odbccp32.lib kernel32.lib user32.lib ~
        gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ~
        ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib ~
        /nologo /subsystem:console /machine:I386
# ADD LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib ~
        comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib ~
        uuid.lib odbc32.lib odbccp32.lib kernel32.lib user32.lib ~
        gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ~
        ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib ~
        /nologo /subsystem:console /machine:I386

!ELSEIF  "$(CFG)" == "MWFDTPM - Win32 Debug"

# PROP BASE Use_MFC 0
# PROP BASE Use_Debug_Libraries 1
# PROP BASE Output_Dir "Debug"
# PROP BASE Intermediate_Dir "Debug"
# PROP BASE Target_Dir ""
# PROP Use_MFC 0
# PROP Use_Debug_Libraries 1
# PROP Output_Dir "Debug"
# PROP Intermediate_Dir "Debug"
# PROP Target_Dir ""
# ADD BASE CPP /nologo /W3 /Gm /GX /ZI /Od /D "WIN32" /D ~
        "_DEBUG" /D "_CONSOLE" /D "_MBCS" /YX /FD /GZ /c
# ADD CPP /nologo /W3 /Gm /GX /ZI /Od /D "WIN32" /D ~
        "_DEBUG" /D "_CONSOLE" /D "_MBCS" /YX /FD /GZ /c
# ADD BASE RSC /l 0x409 /d "_DEBUG"
# ADD RSC /l 0x409 /d "_DEBUG"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
LINK32=link.exe
# ADD BASE LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib ~
        comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib ~
        uuid.lib odbc32.lib odbccp32.lib kernel32.lib user32.lib ~
        gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ~
        ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib ~
        /nologo /subsystem:console /debug /machine:I386 /pdbtype:sept
# ADD LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib ~
        comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib ~
        uuid.lib odbc32.lib odbccp32.lib kernel32.lib user32.lib ~
```

```
        gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib  ~
        ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib ~
        /nologo /subsystem:console /debug /machine:I386 /pdbtype:sept

!ENDIF

# Begin Target

# Name "MWFDTPM - Win32 Release"
# Name "MWFDTPM - Win32 Debug"
# Begin Group "Source Files"

# PROP Default_Filter "cpp;c;cxx;rc;def;r;odl;idl;hpj;bat"
# Begin Source File

SOURCE=.\fractal_isarithm.cpp
# End Source File
# Begin Source File

SOURCE=.\fractal_triangular.cpp
# End Source File
# Begin Source File

SOURCE=.\fractal_variogram.cpp
# End Source File
# Begin Source File

SOURCE=.\globals.cpp
# End Source File
# Begin Source File

SOURCE=.\mwfdtpm.cpp
# End Source File
# Begin Source File

SOURCE=.\read_dat.cpp
# End Source File
# Begin Source File

SOURCE=.\read_erdas.cpp
# End Source File
# End Group
# Begin Group "Header Files"
```

```
# PROP Default_Filter "h;hpp;hxx;hm;inl"
# Begin Source File

SOURCE=.\erdas.h
# End Source File
# Begin Source File

SOURCE=.\isarithm.h
# End Source File
# Begin Source File

SOURCE=.\Timeout.h
# End Source File
# Begin Source File

SOURCE=.\triangular.h
# End Source File
# Begin Source File

SOURCE=.\variogram.h
# End Source File
# End Group
# Begin Group "Resource Files"

# PROP Default_Filter "ico;cur;bmp;dlg;rc2;rct;bin;rgs;gif;jpg;jpeg;jpe"
# End Group
# End Target
# End Project
```

12. fractal_variogram

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <time.h>

#include "variogram.h"
#include "erdas.h"
```

```
#define nTYPE 6

void Open_File(char *input, char *filetype, int band)
  {
  int nFileType;
  char inputfile[100];
  char *TYPE[nTYPE]={"tif","lan","gis","arc","dat","bil"};

  if ((band > 7) || (band < 1))
    {
    printf("ERROR: Band number is invalid.\n");
    exit(1);
    }

  for(nFileType = 0; nFileType < nTYPE; ++nFileType)
    if(!strcmp(filetype, TYPE[nFileType]))
      break;

  strcpy(inputfile,input);
  strcat(inputfile,".");
  strcat(inputfile,filetype);
  printf("FILE NAME: %s\n",inputfile);

  switch(nFileType)
    {
    /*  case 0:  // tif
      Read_Tif();
      break;*/

    case 1:  // lan
      Open_Erdas(inputfile);
      Read_Erdas_HDR();
      z=TwoArrayAlloc(irows,icols);
      Read_Erdas(band,z);
      //        xcell=1.0;
      //        ycell=1.0;
      break;

    case 2:  // gis
      Open_Erdas(inputfile);
      Read_Erdas_HDR();
      z=TwoArrayAlloc(irows,icols);
      //      z=(unsigned short int *)malloc(icols*irows*sizeof(unsigned short int));
      Read_Erdas(band,z);
```

```
      break;

    /*case 3:  // arc
      Read_Grid();
      break;*/

    case 4:  // dat
      xcell=1.0;
      ycell=1.0;
      Read_Dat(inputfile);
      break;

    case 5:  // bil (Band Interleaved by Line?)
      xcell=1.0;
      ycell=1.0;
      printf("Please input rows and columns: for example> 201 346 \n");
      printf("> ");
      scanf("%d %d",&irows,&icols);
      //  irows=icols=1600;
      Read_Bil(inputfile);
      break;

    default:
      printf("ERROR: The file format \"%s\" is unsupported.\n", filetype);
      exit(1);
    }

  return;
  }

  /*-------------------------------------------------------------------
  Read X, Y, and Z values of all pixels; Select only a sample of points
  for computing the variogram using a regular sampling method
-------------------------------------------------------------------*/
void Sampling_Method_1(int select_points)
  {
  int i,j;
  int unit = 1;
  int k = 0;
  int matrix;

  num = 0;
  matrix = irows*icols/select_points + 1;
```

```
  printf("irows = %d, icols = %d, matrix of points selected = %d\n", irows,
icols,matrix);

  zz = (float *)calloc(matrix, sizeof(float));
  xx = (float *)calloc(matrix, sizeof(float));
  yy = (float *)calloc(matrix, sizeof(float));

  if( zz == NULL || xx == NULL || yy == NULL)
    {
    printf("Memory overflow!\n");
    exit(1);
    }

  for(j=0;j<irows;j++)
    for(i=0;i<icols;i++)
      {
      //  printf("z[%d][%d]=%d\n",j,i,z[j][i]);
      if(k%select_points==0)
        {
        zz[num]=z[j][i]; //this is a limited condition!! need to modified!
        xx[num]= xmin + i*xcell;
        yy[num]= ymax - j*ycell;
        num++;
        }
      k++;
      }

  printf("total number of points = %d\n",k);
  printf("Number of points selected = %d\n", num);

  Close_File();
  return;
  }

  /*-----------------------------------------------------------------
  Read X, Y, and Z values of all pixels; Select only a sample of points
  for computing the variogram using a regular sampling method
-----------------------------------------------------------------*/
void Sampling_Method_2(int select_points)
  {
  int i,j;
  int unit = 1;
  int k=0;
  int matrix;
```

```
  num = 0;
  matrix = (irows/select_points + 1)*(icols/select_points + 1);
  printf("irows = %d, icols = %d, matrix of points selected = %d\n",  ~
         irows, icols,matrix);
  zz = (float *)calloc(matrix, sizeof(float));
  xx = (float *)calloc(matrix, sizeof(float));
  yy = (float *)calloc(matrix, sizeof(float));

  if( zz == NULL || xx == NULL || yy == NULL)
    {
    printf("Memory overflow!\n");
    exit(1);
    }
  for(j=0;j<irows;j++)
    for(i=0;i<icols;i++)
      {
      if(j%select_points==0&&i%select_points==0)
        {
        zz[num]=z[j][i]; //this is a limited condition!! need to modified!
        xx[num]= xmin + i*xcell;
        yy[num]= ymax - j*ycell;
        num++;
        }
      k++;
      }

  printf("total number of points = %d\n",k);
  printf("Number of points selected = %d\n", num);

  Close_File();
  return;
  }
  /*-------------------------------------------------------------------
  Read X,Y, and Z values of all pixels; select only a sample of points
  for computing the variogram using a regular and random sampling method
-------------------------------------------------------------------*/
void Sampling_Method_3(int select_points, int sub_numbers)
  {
  int i,j,i_random,j_random;
  int unit=1;
  int k;
  int matrix;
  int random_100_1,random_100_2;
```

```
num=0;
matrix=(irows/select_points+1)*(icols/select_points+1)*sub_numbers;
printf("irows = %d, icols = %d, matrix of points selected = %d\n",  ~
        irows, icols,matrix);
zz = (float *)calloc(matrix, sizeof(float));
xx = (float *)calloc(matrix, sizeof(float));
yy = (float *)calloc(matrix, sizeof(float));

if( zz == NULL || xx == NULL || yy == NULL)
  {
  printf("Memory overflow!\n");
  exit(1);
  }
// randomize();
for(j=0;j<irows;j+=select_points)
  for(i=0;i<icols;i+=select_points)
    {
    for(k=0;k<sub_numbers;k++)
      {
      random_100_1=rand()%select_points;
      random_100_2=rand()%select_points;
      i_random=i+random_100_1;
      j_random=j+random_100_2;
      //    printf("%d %d %f %f [%d][%d]\n",random_100_1,  ~
                random_100_2, rand_fraction_1,rand_fraction_2,  ~
                 j_random,i_random);
      if(i_random<icols&&j_random<irows)
        {
        zz[num]=z[j_random][i_random]; //this is a limited condition!!  Modify!
        xx[num]= xmin + i_random*xcell;
        yy[num]= ymax - j_random*ycell;
        num++;
        }
      }
    }

//printf("total number of points = %d\n",k);
printf("Number of points selected = %d\n", num);

Close_File();
return;
}
/*---------------------------------------------------------------
```

```
   Sampling_Cov(): Write a sampling coverage
 ------------------------------------------------------------*/
 void Sampling_Cov(char *outputfile)
   {
   int i;
   FILE *fp_smp;
   char smp_file[50];

   strcpy(smp_file,outputfile);
   strcat(smp_file,".smp");
   if ((fp_smp = fopen(smp_file, "wt"))==NULL)
     {
     fprintf(stderr, "Cannot open output file %s.\n",smp_file);
     return;
     }
   for(i=0;i<num;i++)
     fprintf(fp_smp,"%d %f %f %f\n",i,xx[i],yy[i],zz[i]);
   fprintf(fp_smp,"END\n");
   fclose(fp_smp);
   return;
   }

   /*-----------------------------------------------------------------
   Close_File():Close file
 -----------------------------------------------------------------*/
 void Close_File(void)
   {
   Close_Erdas();
   return;
   }
   /*-----------------------------------------------------------------
   Calculate distances and squared z_value differences between
   any two of the selected points.
 -----------------------------------------------------------------*/
 void Dist_Vari(int num_group)
   {
   int i,j,k=0;
   float sidea,sideb;
   float distance_max=-1.0;
   float distance_min=99999.0;
   int matrix;
   int npc=0;
   int maxnp;
   int perc,percent;
```

```
matrix = (num-1)*num/2;
printf("triangular matrix of points selected = %d\nRunning... ", matrix);
//putchar('%');
//printf("Running... \n");

dist = (float *)calloc(matrix,sizeof(float));
hvar = (float *)calloc(matrix,sizeof(float));

if( dist == NULL || hvar == NULL )
  {
  printf("Memory overflow!\n");
  exit(1);
  }

for(j=0;j<num-1;j++)
  {
  /*percent=(float)j/(float)(num-1)*50.0+0.5;
  printf("\b\b\b\b");
  printf("%3d",percent);
  putchar('%');
  */
  for(i=j+1;i<num;i++)
    {
    /*  perc=j%4;
    switch(perc){
    case 0: printf("\b/");break;
    case 1: printf("\b-");break;
    case 2: printf("\b\\");break;
    case 3: printf("\b|");break;
    }

     printf("\b\b\b\b");
     printf("%3d",percent);
     putchar('%');
    */
    sidea=(xx[i]-xx[j])*(xx[i]-xx[j]);
    sideb=(yy[i]-yy[j])*(yy[i]-yy[j]);
    dist[k]=sqrt(sidea+sideb);
    hvar[k]=(zz[i]-zz[j])*(zz[i]-zz[j]);
    if(dist[k]>distance_max)
      {
      distance_max=dist[k];
      maxnp=k;
```

```
      }
    else if(dist[k]<distance_min)
      distance_min=dist[k];
    k++;
    }
  }

free(yy);
free(xx);
free(zz);

np=k;

/*if(k!=matrix)
{
printf("Program not good!\n");
printf("k=%d\n",k);
exit(1);
}
*/

/*-------------------------------------------------------------
Determine the number of distance groups and their boundaries
-------------------------------------------------------------*/
rang=(distance_max-distance_min)/num_group;

sumzd=(float *)malloc((num_group+1)*sizeof(float));
ncase=(int *)malloc((num_group+1)*sizeof(int));
gb=(float *)malloc((num_group+1)*sizeof(float));


for(i=0;i<=num_group;i++)
  {
  sumzd[i]=0.0;
  ncase[i]=0;
  gb[i]=distance_min+rang*i;
  }

gb[num_group]=gb[num_group]+0.5;

/*-----------------------------------------------------------------
Group the distance array, find  out number of cases falling within
each group, sum up the squared z-value differences.
-----------------------------------------------------------------*/
```

```
    for(j=0;j<np;j++)
      {
      /*  percent=(float)j/(float)np*50.0+50.5;
      printf("\b\b\b\b");
      printf("%3d",percent);
      putchar('%');
      */
      for(i=0;i<num_group;i++)
        {

        /*  perc=i%4;
        switch(perc){
        case 0: printf("\b/");break;
        case 1: printf("\b-");break;
        case 2: printf("\b\\");break;
        case 3: printf("\b|");break;
        }
        */
        //the sort method is not good, please search to a better one.
        if(dist[j]>=gb[i]&&dist[j]<gb[i+1])
          {
          ncase[i]++;
          sumzd[i]+=hvar[j];
          }
        }
      }

    for(i=0;i<num_group;i++)
      npc+=ncase[i];

    //printf("Number of point pairs = %d\n",k);
    printf("\rMinimum distance between two points = %f\n", distance_min);
    //printf("Minimum distance between two points = %f\n", distance_min);
    printf("Maximum distance between two points = %f\n", distance_max);
    printf("point_pairs classified = %d\n",npc);

    return;
    }

/*------------------------------------------------------------
Compute the mean variance and mid-point distances for each group.
Write the result to an output file.
------------------------------------------------------------*/
void Write_Variogram(char *outputfile)
```

```
{
int i,j;
float middle_point, avezd;
float *log_distance, *log_variance;
float log_dist_min,log_dist_max,log_vari_min,log_vari_max;
FILE *fp_out;

strcat(outputfile,".var");
if ((fp_out = fopen(outputfile, "wt"))==NULL)
  {
  fprintf(stderr, "Cannot open output file.\n");
  return;
  }

fprintf(fp_out,"Band=%3d, Every Nth point=%3d,  ~
      Number of distance groups=%4d\n",band, select_points, num_group);
//fprintf(fp_out," Group #   # of cases   log[dist]   log[mean variance]\n");
printf(" Group #   # of cases   log[dist]   log[mean variance]\n");

log_distance=(float *)malloc(num_group*sizeof(float));
log_variance=(float *)malloc(num_group*sizeof(float));

if( log_distance == NULL || log_variance == NULL )
  {
  printf("Memory overflow!\n");
  exit(1);
  }

log_dist_min=9999.;
log_vari_min=9999.;
log_dist_max=0.;
log_vari_max=0.;

for(i=0;i<num_group;i++)
  {
  middle_point=gb[i]+rang*0.5;
  log_distance[i]=log10(middle_point);
  if(ncase[i]==0)
    {
    log_variance[i]=0.0;
    }
  else
    {
    avezd=sumzd[i]/ncase[i];
```

```
      log_variance[i]=log10(avezd);
      }
    if(log_dist_min>log_distance[i])log_dist_min=log_distance[i];
    if(log_dist_max<log_distance[i])log_dist_max=log_distance[i];
    if(log_vari_min>log_variance[i])log_vari_min=log_variance[i];
    if(log_vari_max<log_variance[i])log_vari_max=log_variance[i];
    }

  fprintf(fp_out,"log_dist_min log_dist_max log_vari_min log_vari_max\n");
  printf("log_dist_min log_dist_max log_vari_min log_vari_max\n");
  fprintf(fp_out,"%10.4f  %10.4f  %10.4f ~
        %10.4f\n",log_dist_min,log_dist_max,log_vari_min,log_vari_max);
  printf("%10.4f  %10.4f  %10.4f ~
        %10.4f\n",log_dist_min,log_dist_max,log_vari_min,log_vari_max);

  fprintf(fp_out," Group #  # of cases  log[dist]  log[mean variance]\n");
  printf(" Group #  # of cases  log[dist]  log[mean variance]\n");
  for(i=0;i<num_group;i++)
    {
    fprintf(fp_out,"  %3d    %6d    %10.4f
%10.4f\n",i+1,ncase[i],log_distance[i],log_variance[i]);
    printf("  %3d    %6d    %10.4f
%10.4f\n",i+1,ncase[i],log_distance[i],log_variance[i]);
    }

  free(log_variance);
  free(log_distance);
  fclose(fp_out);
  return;
  }

  /*-------------------------------------------------------------
  Input the breakpoint number and compute the regression up
  to and including that point
-------------------------------------------------------------*/
void Vari_Fractal(char *outfile, int num_group, int break_point1, int break_point2)
  {
  int i,j,l;
  int k=0;
  float sx,sy,sxy,sqx,sqy;
  float d1,d2,d3,beta,alpha,fd,r,rsq;
  float *log_distance, *log_variance;
  FILE *fp_out,*fp_in;
  char out_var[100], out_fct[100];
```

```
char flag[100];
float xba,yba;

sx=sy=sxy=sqx=sqy=0.0;

strcpy(out_var,outfile);
strcat(out_var,".var");
strcpy(out_fct,outfile);
strcat(out_fct,".fct");

if ((fp_in = fopen(out_var, "rt"))==NULL)
  {
  fprintf(stderr, "Cannot open output file %s.\n",out_var);
  return;
  }

if ((fp_out = fopen(out_fct, "wt"))==NULL)
  {
  fprintf(stderr, "Cannot open output file %s.\n",out_fct);
  return;
  }

log_distance=(float *)malloc(num_group*sizeof(float));
log_variance=(float *)malloc(num_group*sizeof(float));

fgets(flag,63,fp_in);
fgets(flag,53,fp_in);
fgets(flag,51,fp_in);
fgets(flag,56,fp_in);
for(i=0;i<=num_group;i++)
  fscanf(fp_in,"%d%d%f%f\n",&l,&j,&log_distance[i],&log_variance[i]);


for(i=break_point1;i<break_point2;i++)
  {
  if(log_variance[i]==0.0) continue;
  sx+=log_distance[i];
  sy+=log_variance[i];
  sxy+=log_distance[i]*log_variance[i];
  sqx+=log_distance[i]*log_distance[i];
  sqy+=log_variance[i]*log_variance[i];
  k++;
  }
d1=k*sxy-sx*sy;
```

```
  d2=k*sqx-sx*sx;
  beta=d1/d2;
  fd=3.0-beta/2.0;
  d3=k*sqy-sy*sy;
  r=d1/sqrt(d2*d3);
  rsq=r*r;

  xba=sx/k;
  yba=sy/k;
  alpha=yba-beta*xba;

  fprintf(fp_out," # points included in regression = %d\n",k);
  fprintf(fp_out," regression slope = %f\n", beta);
  fprintf(fp_out," fractal dimension= %f\n",fd);
  fprintf(fp_out," r-square = %f\n",rsq);
  fprintf(fp_out," alpha = %f\n",alpha);
  fprintf(fp_out," %f %f\n",xba,yba);

  printf("# points included in regression = %d\n",k);
  printf("regression slope=%f\n", beta);
  printf("fractal dimension=%f\n",fd);
  printf("r-square = %f\n",rsq);
  printf("alpha = %f\n",alpha);
  printf(" %f %f\n",xba,yba);

  free(log_variance);
  free(log_distance);
  fclose(fp_out);
  fclose(fp_in);
  return;
  }

  /*-------------------------------------------------------
  automatically choose break_point2 based on slope changing
  into negative value and assume between 0 and this  breakpoints
  regression value will be maximum
-------------------------------------------------------*/
void Vari_Fractal_Max(char *outfile, int num_group)
  {

  int i,j,l,n;
  int k=0;
  float sx,sy,sxy,sqx,sqy;
  float d1,d2,d3,beta,alpha,fd,r,rsq;
```

```
float *log_distance, *log_variance;
FILE *fp_out,*fp_in;
char out_var[100], out_fct[100];
char flag[100];
float xba,yba;
int break_point1, break_point2;
float rrs[100],rrs_max;


sx=sy=sxy=sqx=sqy=0.0;

strcpy(out_var,outfile);
strcat(out_var,".var");
strcpy(out_fct,outfile);
strcat(out_fct,".fct");

if ((fp_in = fopen(out_var, "rt"))==NULL)
  {
  fprintf(stderr, "Cannot open output file %s.\n",out_var);
  return;
  }

if ((fp_out = fopen(out_fct, "wt"))==NULL)
  {
  fprintf(stderr, "Cannot open output file %s.\n",out_fct);
  return;
  }

log_distance=(float *)malloc(num_group*sizeof(float));
log_variance=(float *)malloc(num_group*sizeof(float));

fgets(flag,63,fp_in);
fgets(flag,53,fp_in);
fgets(flag,51,fp_in);
fgets(flag,56,fp_in);
for(i=0;i<=num_group;i++)
  fscanf(fp_in,"%d%d%f%f\n",&l,&j,&log_distance[i],&log_variance[i]);

for(n=0;n<num_group-5;n++){
  break_point2=n+5;

  k=0;
  sx=sy=sxy=sqx=sqy=0.0;
  //  for(i=break_point1;i<break_point2;i++)
```

```
  for(i=0;i<break_point2;i++)
    {
    if(log_variance[i]==0.0) continue;
    sx+=log_distance[i];
    sy+=log_variance[i];
    sxy+=log_distance[i]*log_variance[i];
    sqx+=log_distance[i]*log_distance[i];
    sqy+=log_variance[i]*log_variance[i];
    k++;
    }
  d1=k*sxy-sx*sy;
  d2=k*sqx-sx*sx;
  beta=d1/d2;
  fd=3.0-beta/2.0;

  d3=k*sqy-sy*sy;
  r=d1/sqrt(d2*d3);
  rrs[num_group-6-n]=r*r;
  printf("rrs[1-%2d]=%8.6f, dimension[1-%2d]=%8.6f\n",n+6,rrs ~
      [num_group-6-n],n+6,fd);
  }

  /*
  for(n=0;n<num_group-5;n++){
  printf("rrs[1-%d]=%f\n",num_group-n,rrs[n]);
  }
*/

break_point2=5;
for(i=2;i<num_group-7;i++) {
  if((rrs[i]>rrs[i-1])&&(rrs[i]>rrs[i+1])&&(rrs[i]>0.5)){
    //     (rrs[i-1]>rrs[i-2])&&(rrs[i+1]>rrs[i+2])) {
    //   if(rrs[i]>=0.6){
    break_point2=num_group-i-1;
    break;
    }
  }

printf("break_point2=%d\n",break_point2);

k=0;
sx=sy=sxy=sqx=sqy=0.0;
for(i=0;i<break_point2;i++)
  {
```

```
    if(log_variance[i]==0.0) continue;
    sx+=log_distance[i];
    sy+=log_variance[i];
    sxy+=log_distance[i]*log_variance[i];
    sqx+=log_distance[i]*log_distance[i];
    sqy+=log_variance[i]*log_variance[i];
    k++;
    }
  d1=k*sxy-sx*sy;
  d2=k*sqx-sx*sx;
  beta=d1/d2;
  fd=3.0-beta/2.0;
  d3=k*sqy-sy*sy;

  r=d1/sqrt(d2*d3);
  rsq=r*r;

  xba=sx/k;
  yba=sy/k;
  alpha=yba-beta*xba;

  fprintf(fp_out," # points included in regression = %d\n",k);
  fprintf(fp_out," regression slope = %f\n", beta);
  fprintf(fp_out," fractal dimension= %f\n",fd);
  fprintf(fp_out," r-square = %f\n",rsq);
  fprintf(fp_out," alpha = %f\n",alpha);
  fprintf(fp_out," %f %f\n",xba,yba);

  printf("# points included in regression = %d\n",k);
  printf("regression slope=%f\n", beta);
  printf("fractal dimension=%f\n",fd);
  printf("r-square = %f\n",rsq);
  printf("alpha = %f\n",alpha);
  printf(" %f %f\n",xba,yba);

  free(log_variance);
  free(log_distance);
  fclose(fp_out);
  fclose(fp_in);
  return;
}

/*-----------------------------------------------------
Close_Variogram:
```

```
----------------------------------------------------*/
void Close_Variogram()
  {
  free(gb);
  free(ncase);
  free(sumzd);
  free(dist);
  free(hvar);
  return;
  }
```

13. fractal_triangular

```
#include "triangular.h"



void Compute_triangular(int steps, int method)
  {
  int i,j;
  int iter, step=1, no_of_blocks[50];
  double side, diag, gain;
  double a, b, c, d, e, u, v, x, y, o, p, q, r;
  double sa, sb, sc, sd, aa, ab, ac, ad, surface_area;
  float pixel_size;
  int nrows,ncols;

  step_num = steps;
  nrows = below - up + 1;
  ncols = right - left + 1;

  //pixel_size=xcell;
  pixel_size = 1;
  gain = 255.0 / (zmax - zmin);
  for(iter=1;iter<=steps;iter++)
    {
    surface_area=0.0;
    no_of_blocks[iter]=0;
    side = (float) pixel_size * step;
    diag = (float) side * sqrt(2.0) / 2.0;
    for (i = up; i <= below; i += step)
      {
      if ((i + step) > below)
        break;
```

```
  for(j = left; j <= right; j += step)
    {
    if ((j + step) > right)
      break;

    a = (z[i][j] - zmin) * gain;
    b = (z[i][j + step] - zmin) * gain;
    d = (z[i + step][j] - zmin) * gain;
    c = (z[i + step][j + step] - zmin) * gain;

    e = 0.25 * (a + b + c + d);

    double dSideSquared = side * side;
    u = sqrt((a - b) * (a - b) + dSideSquared);
    v = sqrt((b - c) * (b - c) + dSideSquared);
    x = sqrt((c - d) * (c - d) + dSideSquared);
    y = sqrt((a - d) * (a - d) + dSideSquared);

    double dDiagSquared = diag * diag;
    o = sqrt((a - e) * (a - e) + dDiagSquared);
    p = sqrt((b - e) * (b - e) + dDiagSquared);
    q = sqrt((c - e) * (c - e) + dDiagSquared);
    r = sqrt((d - e) * (d - e) + dDiagSquared);

    sa = 0.5 * (u + p + o);
    sb = 0.5 * (v + p + q);
    sc = 0.5 * (x + q + r);
    sd = 0.5 * (y + o + r);

    aa = sqrt(fabs(sa * (sa - u) * (sa - p) * (sa - o)));
    ab = sqrt(fabs(sb * (sb - v) * (sb - p) * (sb - q)));
    ac = sqrt(fabs(sc * (sc - x) * (sc - q) * (sc - r)));
    ad = sqrt(fabs(sd * (sd - y) * (sd - o) * (sd - r)));

    surface_area += aa + ab + ac + ad;
    no_of_blocks[iter]++;
    }
  }
area[iter]=surface_area;
resolution[iter]=side*side;

if(method==0)
  step*=2;
else
```

```
      step++;

    if (step >= nrows || step >= ncols)
      {
      step_num = iter;
      break;
      }
    }
  }



void Line_Fit(FILE *pfDebug, float& fFractalDimension, float& fRSquared)
  {
  int n;
  double resavg=0.0,areaavg=0.0,cross=0.0,sumres=0.0,sumarea=0.0;
  double dimension, alpha, beta, r;

  for (n = 1; n <= step_num; n++)
    {
    resavg += log(resolution[n]);
    areaavg += log(area[n]);
    }

  if (step_num < 3)
    {
    printf("\n Too few calculated data points for regression\n");
    exit(0);
    }

  resavg/=(float)step_num;
  areaavg/=(float)step_num;

  for (n=1;n<=step_num;n++)
    {
    cross+=((log(resolution[n])-resavg)*(log(area[n])-areaavg));
    sumres+=((log(resolution[n])-resavg)*(log(resolution[n])-resavg));
    sumarea+=((log(area[n])-areaavg)*(log(area[n])-areaavg));
    }
  r=cross/sqrt(sumres*sumarea);
  beta=r*sqrt(sumarea)/sqrt(sumres);
  alpha=areaavg-(beta*resavg);
  dimension=2.0-beta;
```

```
fprintf(pfDebug, "%d\t%.3f\t", step_num, r*r);

fFractalDimension = (float)dimension;
fRSquared = (float)(r * r);
}
```

BIBLIOGRAPHY

Anderson, E. J. (2001). Moving Window of Fractal Dimension by Triangular Prism Method (MWFDTPM-version 1.0.5.)

Atkinson, P.M. & Tate, N.J. (2000). Spatial Scale Problems and Geostatistical Solutions: A Review. Professional Geographer, 52, 607-623.

Bian, L. (1997). Multiscale Nature of Spatial Data in Scaling Up Environmental Models. In D.A. Quattrochi & M.F. Goodchild (Eds.), Scale in Remote Sensing and GIS (pp.13-26). Florida: CRC Press.

Brigham, E. O. (1988). The Fast Fourier Transform and Its Applications. New Jersey: Prentice-Hall.

Avery, T.A. & Berlin, G.L. (1992). Fundamentals of Remote Sensing and Airphoto Interpretation (5^{th} ed.). New York: Macmillan Publishing.

Cao C. & Lam, N.S.N. (1997). Understanding the Scale and Resolution Effects in RemoteSensing and GIS. In D.A. Quattrochi & M.F. Goodchild (Eds.), Scale in Remote Sensing and GIS (pp.57-72). Florida: CRC Press.

Chen, C.-C. & Chen, C.C. (1999). Filtering Methods for Texture Discrimination. Pattern Recognition Letters, 20, 783-790.

Chica-Olmo, M. & Abarca-Hernandez, F. (2000). Computing Geostatistical Image Texture for Remotely Sensed Data Classification. Computers & Geosciences, 26, 373-383.

Clarke, K.C., & Schweizer, D.M. (1991). Measuring the Fractal Dimension of Natural Surfaces using a Robust Fractal Estimator. Cartography and Geographic Information Systems, 18, 37-47.

Clarke, K.C. (1986). Computation of the Fractal Dimension of Topographic Surfaces Using the Triangular Prism Surface Area Method. Computers & Geosciences, 12, 713-722.

Cliff, A.D., & Ord, J.K. (1981). Spatial Processes: Models and Applications. London: Pion.

Collins, J. B. & Woodcock, C. E. (1999). Geostatistical Estimation of Resolution-Dependent Variance in Remotely Sensed Images. Photogrammetric Engineering & Remote Sensing, 65, 41-50.

De Cola, L. (1989). Fractal Analysis of a Classified Landsat Scene. Photogrammetric Engineering & Remote Sensing, 55, 601-610.

Emerson, C. W., Lam, N.S.N., & Quattrochi, D.A. (1999). Multi-Scale Fractal Analysis of Image Texture and Pattern. Photogrammetric Engineering & Remote Sensing, 65, 51-61.

Environmental Systems Research Institute. (2001). ESRI GIS & Mapping Software. Retrived May13th 2001 from ESRI Incorporated on the World Wide Web: http://www.esri.com/software/arcview/index.html

Fisher, R., Perkins S., Walker A., & Wolfart E. (2000). Mean Filter-Convolution. Retrieved June 28th 2001 from Hypermedia Image Processing Reference on the World Wide Web: http://www.dai.ed.ac.uk/HIPR2/convolve.htm

Gonzalez, R.C. & Wintz, P. (1987). Digital Image Processing. California: Addison-Wesley.

Goodchild, M.F. & Mark, D. M. (1987). The Fractal Nature of Geographic Phenomena. Annals of the Association of American Geographers, 77, 265-278.

Goodchild, M. F. (1980). Fractals and the Accuracy of Geographical Measures. Mathematical Geology, 12, 85-98.

Goodchild, M. F. (1986). Spatial Autocorrelation. Norwich: Geo Books.

Haralick, R.M. (1979). Statistical and structural approaches to texture. Proceedings of the IEEE, 67, 786-804.

Haralick, R.M., Shanmugam, K. & Dinstein, I. (1973). Texture Features for Image Classification. IEE Transactions on Systems, Man and Cybernetics, 6, 610-621.

Isaaks, E. H. & Srivastava, R. M. (1989). An Introduction to Applied Geostatistics. New York: Oxford University Press.

Jaggi, S., Quattrochi, D.A. & Lam, N.S.N. (1993). Implementation and Operation of Three Fractal Measurement Algorithms for Analysis of Remote-Sensing Data.

Computers and GeoSciences, 19, 745-767.

Lam, N.S.N. & De Cola, L. (1993). Fractals in Geography. New Jersey: Prentice Hall.

Lam, N.S.N. (1990). Description and Measurement of Landsat TM Images Using Fractals, Photogrammetric Engineering & Remote Sensing, 56, 187-195.

Lee, Y.K. & Hoon, K. (2001). Brownian Motion- The Research Goes On. Retrieved June 28th 2001 from The Department of Computing Imperial College of Science, Technology and Medicine University of London on the World Wide Web: http://www.doc.ic.ac.uk/~nd/surprise_95/journal/vol4/ykl/report.html

Lillesand, T.M. & Kiefer. R.W. (1994). Remote Sensing and Image Interpretation (3rd ed). New York: John Wiley & Sons, Inc.

Mandelbrot, B.B. (1982). Fractal Geometry of Nature. California: W. H. Freeman & Co.

Mandelbrot, B. B., & Van Ness, J. W. (1968). Fractional Brownian Motions, Fractional Noises and Applications. SIAM Review, 10, 422-437.

Mandelbrot, B.B. (1977). Fractals: Form, Chance and Dimension. California: W. H. Freeman & Co.

Pesariesi, M. (2000). Texture Analysis for Urban Pattern Recognition Using Fine-resolution Panchromatic Satellite Imagery, Geographical and Environmental Modelling, 4, 43-63.

Pratt, W. K. (1978). Digital Image Processing. New York: John Wiley & Sons.

Qiu, H., Lam, N.S.N., Quattrochi, D.A. & Gamon, J. A. (1999). Fractal Characterizationof Hyperspectral Imagery. Photogrammetric Engineering & Remote Sensing, 65, 63-71.

Quattrochi, D. A., & Goodchild, M. F. (1997). Scale, Multiscaling, Remote Sensing, and GIS, In D.A. Quattrochi & M.F. Goodchild (Eds.), Scale in Remote Sensing and GIS (pp. 1-11). Florida: CRC Press.

Quattrochi, D. A., Lam, N. S. N., Qiu, H., & Zhao, W. (1997). Image Characterization and Modeling System (ICAMS): A Geographic Information System for the Characterization and Modeling of Mulitscale Remote Sensing

Data. In D.A. Quattrochi & M.F. Goodchild (Eds.), Scale in Remote Sensing and GIS (pp. 361-394). Florida: CRC Press.

Rho, P., (2000). Extract by Moving Windows (extractbymovingwindows.ave). Texas A&M University, Texas.

Sali, E. & Wolfson, H. (1991). Texture Classification in Aerial Photographs and Satellite Data. Proceedings of the Seventh Israeli Conference, 494, 325-34.

Sonka, M., Hlavac, V., & Boyle, R. (1999). Image Processing, Analysis, and Machine Vision. Retrived April 14th 2001 from Brooks and Cole Publishing, on the World Wide Web: http://www.icaen.uiowa.edu/~dip/LECTURE/Texture1.html#cooccurrence

MicroImages Software. (2000). TNTmips - version 6.4. Nebraska.

Tate, N. J. (1998). Estimating the Fractal Dimension of Synthetic Topographic Surfaces. Computers and GeoSciences, 24, 325-334.

U.S. Army Corps of Engineers & GRW.Inc (USACE & GRW). (1999). Tracking Bluff Line Movement in Southwest Michigan 2000. Conducted by Western Michigan University, Geographic Information Systems Research Center, Michigan.

Woodcock, C.E. & Strahler, A.H. (1987). The Factor of Scale in Remote Sensing. Remote Sensing of the Environment, 21, 311-332.

Xia, Z.-G & Clarke, K.C. (1997). Scale, Approaches to Scaling of Geo-Spatial Data. In D.A. Quattrochi & M.F. Goodchild (Eds.), Scale in Remote Sensing and GIS (pp. 309-360). Florida: CRC Press.