The Geomorphological Characterisation of Digital Elevation Models

Thesis submitted for the degree of
Doctor of Philosophy
at the University of Leicester

by

Joseph Wood
Department of Geography
University of Leicester

March 1996

UMI Number: U077269

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

UMI U077269
Published by ProQuest LLC 2015. Copyright in the Dissertation held by the Author.
Microform Edition © ProQuest LLC.
All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346


LEICESTER UNIVERSITY
LIBRARY

Abstract

Techniques and issues are considered surrounding the characterisation of surface form represented by Digital Elevation Models (DEMs). A set of software tools suitable for use in a raster based Geographical Information System (GIS) is developed. Characterisation has three specific objectives, namely to identify spatial pattern, to identify scale dependency in form and to allow visualisation of results. An assessment is made of the characteristics of error in DEMs by identifying suitable quantitative measures and visualisation processes that may be enabled within a GIS. These are evaluated by contour threading a fractal surface and comparing four different spatial interpolations of the contours. The most effective error characterisations are found to be those that identify high frequency spatial pattern. Visualisation of spatial arrangement of DEM error is used to develop a deterministic error model based on local surface slope and aspect. DEMs are parameterised using first and second derivatives of quadratic surfaces fitted over a range of scales. This offers advantages over traditional methods based on a 3 by 3 local window, as geomorphometric form can be characterised at any scale. Morphometric parameters are combined to give a feature classification that may also be applied over a range of scales. Multi-scale measurements are combined to give a feature membership function that describes how properties change with scale. These functions are visualised using modal and entropy measures of variability. An additional method of visualising scale dependency is suggested that graphically represents statistical measures of spatial pattern over a variety of spatial lags. This is found most appropriate for detecting structural anisotropy in a surface. Characterisation tools are evaluated by applying them to uncorrelated surfaces, fractal surfaces and Ordnance Survey DEMs of Lake District, Peak District and Dartmoor.

Acknowledgements

I will not attempt to list all those to whom I owe thanks in producing this work. Perhaps I should simply say thanks to those in Bristol, Sheffield and Leicester who have made this all possible.

I would however, particularly like to acknowledge the inspiration and support offered by David Unwin and Peter Fisher without whom I really wouldn't have made it this far.

To Matthew

All UK Ordnance Survey DEMs used in this work are Crown Copyright. The assistance provided by the Ordnance Survey is gratefully acknowledged.

Preface

As I put the finishing touches to this Thesis, the film *An Englishman who Went up a Hill But Came Down a Mountain* has been enjoying success. It is based on the book by Christopher Monger and describes how, in 1917, an English surveyor was required to survey the topography around *Ffynnon Garw,* a small village in South Wales. The inhabitants of *Ffynnon Garw*, who for generations had long known that they lived at the foot of "the first mountain in Wales", were somewhat aggrieved to be told by an Englishman that their "mountain" was only a "hill"; it was 20 feet short of the 1000 feet required by the Ordnance Survey to classify it as a mountain. They promptly turned their hill into a mountain by building a 20 foot high soil mound on its summit.

If only life was that simple. The following 456 pages describe an attempt to develop the tools that will allow us to describe hills and mountains, valleys and dales, cols and passes, with perhaps a little more sophistication than by their elevation alone.

The Geomorphological Characterisation of Digital Elevation Models

Table of Contents

Chapter One - Introduction

One of the long standing aims of science has been to impose a rational, internally consistent, framework upon 'nature'. The construction and implementation of that framework is intended to help us understand and predict the nature it describes. Whilst the development of such frameworks may be from a variety of perspectives, the importance of *description* should not be underestimated. Description may be directly of nature, or of the processes that result in observed phenomena, or of the scientific framework itself. This thesis is an attempt to construct such a scientific framework, one which can be used to describe and understand landscape.

The overall aim of this work is to develop a set of tools that may be used to describe any geomorphological landscape in a discriminating and succinct manner. Many attempts have, of course, been made to do this previously. The process of *parameterisation* of landscape is to identify and measure those properties of landscape that are seen as most fundamental in its description of form. The *thesis* presented here is that in order to successfully characterise a geomorphological surface, it is necessary not only to identify the parameters of that surface, but to identify, describe and visualise, the *spatial* association between parameters. 'Tools' are developed from several scientific sub-disciplines in order to achieve this.

1.1 Surface Characterisation

Before embarking on a detailed description of the nature of this work, its scope is defined by addressing a number of underlying questions.

What should a characterisation of landscape attempt to achieve ?

The answer to this question touches on the nature of scientific method. Is it possible to identify an objective reality, that of the 'true' landscape ? If we assume the answer to this

question is yes, then characterisation becomes the process of identifying the optimal categorisation of that reality. However, the concept of 'landscape' embodies a host of ideas, not simply physically determined by geomorphological process, but culturally defined by its use and the preconceptions of the observer. As a consequence, a framework for objective testing of landscape characterisation is not possible without a stricter definition of terms.

It is more useful to consider an alternative assumption where our imposition of a scientific framework becomes *a* reality. The scheme we use to describe a landscape can be evaluated by its consistency with other scientific conventions, its discriminating power with respect to the applications it is orientated towards, and its own internal consistency. This is the approach that is taken during this study, both in the definition of landscape and in the evaluation of the tools developed to describe it.

How should landscape be modelled ?

In order to provide a somewhat objective scheme for development and evaluation of characterisation tools, a very specific definition of landscape will be used throughout this study. If we ignore our own 'filter of perception' in our appreciation of landscape, we can define any landscape in terms of its form (geomorphological), what lies upon it (landcover) and what it is used for (landuse). This study will be limited to a geomorphological treatment of landscape (see Figure 1.1).

In turn, a geomorphological landscape can be appreciated purely in terms of its measurable surface form (geomorphometric), the materials that make up the surface and sub-surface (material), and the processes that give rise to the geomorphometric and material characteristics (process). The former alone will be considered here. Of course, it should be recognised that a substantial part of geomorphological research has been devoted to relating these three elements together. The contribution offered by the current work is to provide a more effective mechanism for characterising form (geomorphometry).

The study of surface form, and more particularly, the taking of surface measurements from maps or models, has a long history dating back at least to the mid nineteenth century (Cayley, 1859; Maxwell, 1870). More recently, the accessibility of computing technology in the 1970's led to a renaissance in the approach, particularly with the work of Evans (1972, 1979) and Mark, (1975*a*, 1975*b*). The work reported in this volume represents what appears to be part of a second renaissance accompanied by the widespread availability of Geographical Information Systems.

Finally, within the context of GIS, surface geomorphometry is most commonly modelled either as a Digital Elevation Model (DEM), or as a Digital Terrain Model (DTM). Here, a DEM is defined as a regular two dimensional array of heights sampled above some datum that describes a surface. A DTM contains elevation information with the addition of some explicit coding of the surface characteristics such as breaks in slope, drainage divides etc. Since one of the aims of this study is to enable the production of such ancillary information, the work will be based upon DEMs only.

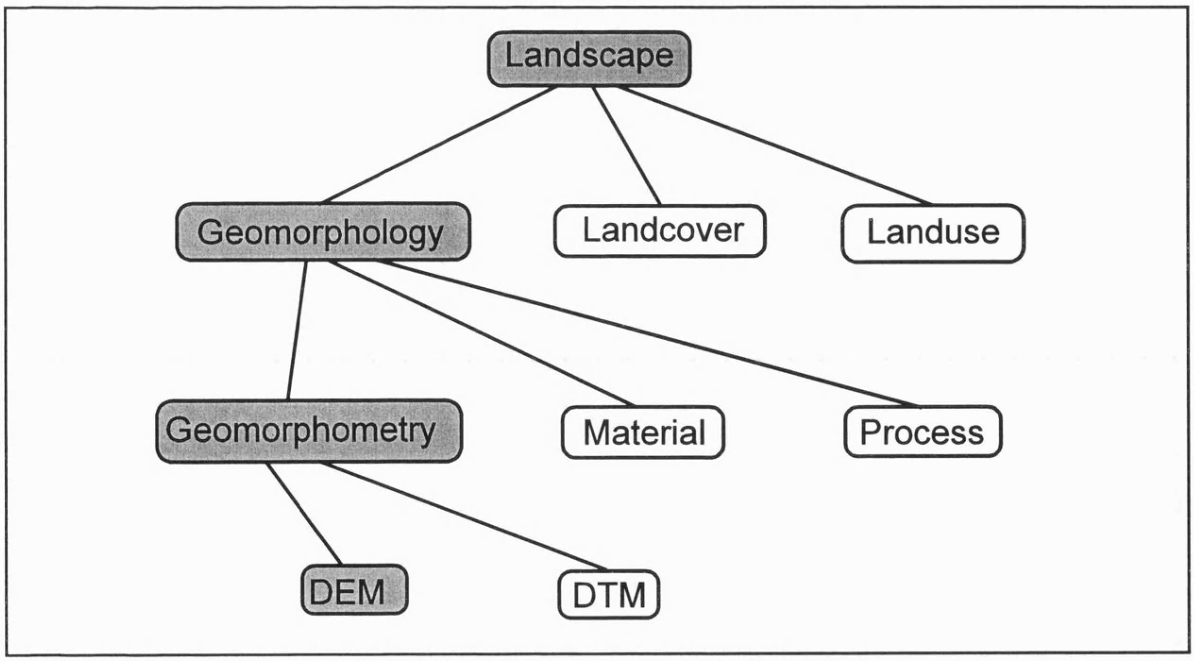


Figure 1.1 - The context of the Digital Elevation Model (DEM) as a representation of landscape.

How should tools for characterisation be built ?

Geomorphological characterisation tools can be derived using two broad approaches. The *theoretical approach* is orientated towards the construction of the tools themselves. Methodological development consists of evaluating existing tools, identifying weaknesses, and producing new or enhanced methodologies. Evaluation of tools produced using this approach is likely to be by assessment of internal and external inconsistencies, and discriminating power. Alternatively, an *empirical approach* may be adopted where landscapes with known characteristics are evaluated. Tools are developed that try to identify these characteristics. Evaluation is largely based on the effectiveness of characterisation in modelling known phenomena. There is a parallel here with the distinction between general and specific geomorphometry (see section 2.2).

The approach adopted here is of the former, theoretical one, inspired largely by the weaknesses in existing techniques for surface characterisation.

The intimacy of the link between *interpolation*, *generalisation*, and *characterisation* will become apparent throughout this work. In essence all three processes involve the same procedure. That is they all attempt to extract *information* from *data.* In the case of generalisation it is in order to represent information as efficiently as possible, with a minimum of data. For interpolation it is to extract the maximum amount of information necessary to inform the process of data creation. Characterisation can simply be regarded as the process of making information derived from data explicit. It is not surprising therefore, that much of the work considered in this study has origins in the fields of spatial interpolation and generalisation.

In summary, the aim of this study is to produce a set of techniques (or 'tools') that may be used to identify and discriminate between different surface forms. Throughout, gridded digital elevation models will be used as a surrogate for landscape, while recognising that they only model a very specific component - that of surface form.

More specifically, the study has the following (testable) objectives,

- To assess whether a DEM alone, contains useful information for geomorphological characterisation.

- To identify weaknesses in existing methods of surface characterisation and present new solutions to overcome these weaknesses.

- To assess the effect that elevation model uncertainty has on surface characterisation.

- To assess the effectiveness of *visualisation* as a means of conveying information on surface form and as a methodological process.

- To produce working software that may be used in a GIS environment for surface characterisation.

In meeting these objectives, a number of continuing themes become apparent; these are outlined below.

1.2 Scientific Visualisation

Although the term *Scientific Visualisation* has only received popular attention in the last decade, the geographical tradition of illustrating spatial (and non-spatial) information graphically, has predated the discipline by many centuries. Visualisation is used in this study in two contexts. Firstly, the figures presented in this volume should provide an effective means of communication of embodied ideas. Secondly, and more importantly, the visualisation *process* has been instrumental in the development of many of the ideas presented. It is one of the primary arguments of this thesis that the visualisation process provides an important methodological approach that is necessary for the effective use of the tools presented. That process involves a dynamic interaction between user and computer, a process that cannot be

fully conveyed in a written dissertation.

More specifically it is possible to enlarge on DiBiase's (1990) notions of visualisation in the public and private realms. Many of the techniques described in this volume have arisen from exploratory *ideation* (McKim, 1972) firmly in the private realm of visual thinking. Yet if the methodologies described are to have use elsewhere, their visualisation must be capable of being used in the public realm of visual communication. Thus graphical representation is used in a variety of contexts within this work. It ranges from the visual communication of DEM uncertainty (see Chapter 3) to the private realm of visual thinking implied by the use of textural lag diagrams (see Chapter 4). Visualisation should take a more important and central role than conventional statistical measures of surface characterisation.

1.3 Image Processing

The original impetus for this work was a recognition that many of the tasks confronted by the image processing community have similarities with those facing geomorphometry. Much of the image processing literature is given to issues of information extraction from 2-dimensional raster representations of (reflectance) surfaces. This relationship is illustrated in more detail in Figure 1.2.

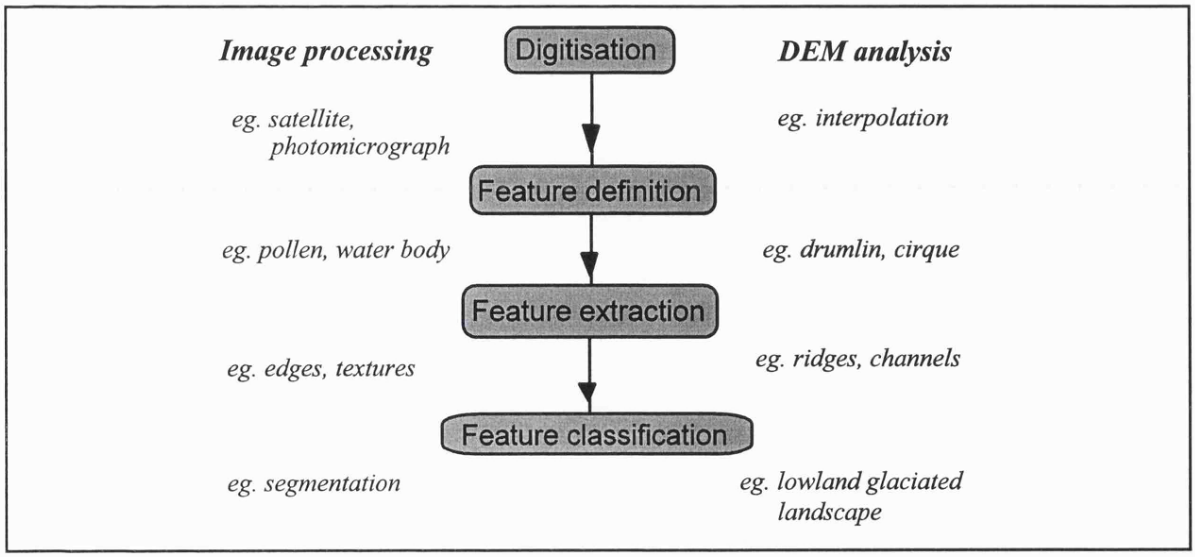


Figure 1.2 - The relationship between the characterisation of Digital Elevation Models and digital images.

Some of the ideas discussed here, particularly those of 'texture analysis' and 'edge detection' are fundamental to the disciplines. Techniques such as co-occurrence matrix analysis (see Chapter 4) are borrowed directly from the image processing literature. Others such as feature classification (see Chapter 5) have a geomorphological origin, yet could be equally appropriate for the segmentation of a grey-scale image. One of the problems of the somewhat parallel evolution of both disciplines is the different use of terminology adopted by each. Throughout this work, a geomorphometric nomenclature is chosen where possible, although it is readily acknowledged that many of the ideas used have arisen from both disciplines.

1.4 Uncertainty in Surface Models

It was the original intention of this work to consider only those techniques suitable for geomorphometric analysis. Yet within a few months of study it became apparent that in many cases it was difficult to separate the attributes of surface form that were due to geomorphological process from those due to model uncertainty. Consequently, it has become one of the objectives of this work to distinguish the two sets of influences. An entire chapter is given over to considering the techniques suitable for identifying uncertain or inaccurate elements of surface models (see Chapter 3). This is a *necessary* consideration and one that cannot be ignored if effective surface characterisation is to be achieved

It has only been possible to begin to separate the effect of uncertainty from geomorphological form because of its relatively unique spatial and scale based characteristics. The notions of scale and spatial distribution do themselves recur throughout this volume, and are perhaps central to the uniqueness of the approach adopted within.

1.5 Thesis Outline

Chapter 2 presents the research context from which this study has arisen. In making reference to literature several issues are considered. Firstly the research problem is more fully defined by considering attempts to characterise surfaces effectively, and more particularly their limitations. Secondly, the themes that run through this research are considered in more detail. It is hoped that by considering the research context, a research 'niche' is made apparent, one which is investigated in the following four chapters.

Chapter 3 is devoted to considering DEM uncertainty in greater depth. The first part considers how the effects of data uncertainty may be quantified and visualised. The emphasis is placed on the use of the interactive visualisation process and a consideration of spatial and scale dependent characterisation. The second part of the chapter considers an example of uncertainty modelling as a mechanism for separating data error from geomorphometric form. Again the use of visualisation as part of the research process is emphasised.

Chapter 4 covers in some technical detail the process of surface *parameterisation* using quadratic patches to model local surface form. The emphasis is placed on a computational implementation of the process. The second part of the chapter considers how parameterisation should be applied over a range of scales in order to provide effective surface characterisation. The third part of the chapter is a description of alternative tools for scale-based description. In particular, the notion of the 'lag-diagram' - a two-dimensional map of spatial association - is considered an important part of the visualisation of spatial behaviour.

Chapter 5 goes on to consider how some of the surface parameters discussed in the previous chapter can be used in a geomorphological context. In particular, the derivatives of quadratic surface patches are used to provide a classification of surface feature type. The second part of the chapter is a consideration of how some of the hydrological descriptions of a surface can be used to characterise surface form.

Chapter 6 consists of an evaluation of the tools developed in the previous three chapters. Many of the visualisations are new and do not allow intuitive interpretation. So one of the objectives of this chapter is to 'calibrate' the tools by applying them to surfaces with known properties. These range from simple geometric objects, uncorrelated Gaussian surfaces, through simple polynomial functions, to fractal surfaces. The tools are also evaluated by applying them to contrasting Ordnance Survey DEMs of the English Lake District, the Peak District and Dartmoor. The objective of this chapter is not to characterise the landscapes *per se*, but to assess the effectiveness of the tools developed.

The final chapter draws conclusions from the development and assessment of characterisation tools by re-evaluating the aims and objectives stated in the introduction. The continuing research themes that have arisen out of this work are identified and provisionally assessed.

Over 20,000 lines of C code have been produced as part of this research. It has been in the production of workable software, that by far the greatest effort has been expended. The emphasis in coding has been on clarity, as it is the description of algorithms that, in many cases, is as useful as a working system. It is for this reason that all C code has been included as an appendix to this volume. Frequent reference is made to these appendices, but where an algorithm is considered to be of more general use, it is also represented as a figure within the main body of text.

Chapter Two - Research Context

This chapter reviews and discusses the research that provides a context for this study. It is not meant to be an exhaustive review of the literature, but more an indication of the theoretical development of, and problems associated with, characterising surface form using Digital Elevation Models. More comprehensive reviews, collections and bibliographies include Pike (1993), Goudie (1990), (geomorphometry); Petrie and Kennie(1990), Weibel and Heller (1991), (Digital Elevation Models); and Wilkinson *et al* (1995), Unwin (1989), (fractals in the geosciences).

2.1 The Digital Elevation Model

The form of surface model used for this entire study is that of the Digital Elevation Model (DEM). Although the term is used inconsistently in the literature (Burrough, 1986; Weibel and Heller, 1991) it is strictly defined here consistent with the terms of Burrough (1986), as a *regular gridded matrix representation of the continuous variation of relief over space.* Georeferencing of elements that comprise the surface are implicitly defined by the ordering of elevation values within the matrix. No additional or explicit representation of surface form is recorded. The contrast should be noted between such surface models and the Digital Terrain Model (DTM) which includes some additional explicit representation of surface form. Examples of DTMs include the Triangulated Irregular Network (TIN) (Peucker, 1978), digital contours with 'form lines' (Ordnance Survey, 1992), and the 'richline model' of Douglas (1988) that uses ridge, valley and form lines to define an elevation model.

The attraction of a simple matrix of elevation values (amenable to processing with procedural computer languages) was one of the reasons for its uptake in the early 1970s as a model suitable for landscape analysis (eg Evans, 1972). More recently, its continued widespread use as a model of surface form may be attributed to its easy integration within a GIS environment (Weibel and Heller, 1990,1991).

There are a number of conceptual problems that must be addressed when considering DEMs as models of surface form. Firstly, despite being a model of continuous surface form, a DEM is a set of discrete elevation measurements of what is usually an undifferentiatable surface. The *fidelity* with which the DEM models the true surface will depend on surface roughness and DEM resolution. Yet, evidence from fractal research (see section 2.5) suggests that there will always be detail at a finer scale than that measured at the DEM resolution. This suggests that all DEMs implicitly model at a certain *scale* implied by the grid cell resolution. Although the scale dependency of measurement has been recognised and even modelled by many authors (eg Garbrecht and Martz, 1993; Ackerman, 1993; Hodgson, 1995), it still remains hidden in many aspects of DEM analysis.

A second conceptual problem must be addressed in considering what each elevation value within a gridded matrix represents. It is possible to conceive of the matrix as a collection of elevation samples at point locations. If this assumption is made, the model is not one of continuous surface form without invoking an additional assumption about the spatial relationship between recorded elevation values. Although the interpolation between grid cells is rarely addressed by those using DEM analysis, the form of implicit interpolation can have significant effects on analytical results. Fisher (1993) showed that DEM derived viewshed areas could vary drastically depending on how adjacent elevation values were interpolated to form a continuous surface. Kumler (1994) suggested that a linear interpolation between adjacent DEM cells produced the smallest residuals when compared with TIN models of the same area. He also recognised that different interpolation procedures gave rise to different elevation estimates. This problem has led to a consistent (quadratic) form of cell interpolation to be adopted for this study where a continuous model of surface form is required.

2.2 Geomorphometry

The measurement of shape has received much attention in many branches of science ranging from, biology and palaeontology (for example, Thompson, 1917), to mathematics and image

processing (Serra, 1982). The systematic measurement of topography from cartographic sources can be traced back at least as far as the mid-nineteenth century (Cayley, 1859). Significantly, the development of a science of surface measurement has been accompanied by the development of surface storage and representation methods. Cayley's analysis of "slope and contour lines" was only possible (and useful) because of the introduction of accurate contour representations of relief in the nineteenth century (Imhof, 1982). This section considers the science of *geomorphometry*, or the science "which treats the geometry of the landscape" (Chorley *et al*, 1957).

A substantial part of twentieth century geomorphology, at least since Tricart (1947), has been devoted to the measurement and quantification of topographic form, although as Pike (1993) has noted, this has proved less successful than the characterisation of geomorphological process. The notable exception has been in the quantification of channel form at the drainage basin scale (Horton, 1945; Strahler, 1964; Shreve, 1966). The quantification of network form has allowed the empirical relationships between measurements to be investigated, often with claims of universality (eg Horton, 1945; Schumm, 1956; Hack, 1957, Shreve, 1974). A criticism of this approach has been that the indices used (eg stream order) or the empirical relationships observed (eg basin mainstream length ratio) are insensitive to topographic variation (Abrahams, 1985; Knighton, 1984). Further, such empirical observations may be as much a function of scale of sampling and measurement as they are of geomorphological variation (eg Church and Mark, 1980; see also section 2.5 below).

The drainage network forms only a part of an overall description of topography. Several authors have attempted to place network quantification within a wider context of topographic form. The early work of Cayley (1859) and Maxwell (1870) considered the topological links between channels, ridges and local extrema. Mark (1979) considered the topological properties of ridge networks. Werner (1988) attempted to formalise the relationship between the channel network and interdigitating ridge network. Wolf (1989,1991) provided a similar formalisation using graph theory. However these topological descriptions do not consider the geometrical properties of surface form, nor do they provide descriptions of continuous variation over a surface.

2.2.1 General Geomorphometry

Evans (1972) made the distinction between attempts to quantify specific geomorphological features (*specific geomorphometry*) and *general geomorphometry,* "the measurement and analysis of those characteristics of landform which are applicable to any continuous rough surface" (Evans, 1972, p.18). The origins of this approach arise from the cartometric analysis of map data (eg Glock, 1932; Johnson, 1933; Wood and Snell, 1957, 1960) and the derivation of descriptive indices from them. Mark (1975*a*) suggests that what unifies this approach, is an attempt to quantify the concept of terrain "roughness". He suggests that the planimetric variation of roughness embodies two primary scales, namely the *grain* and *texture*. Grain refers to the longest significant wavelength of a topographic surface, texture, the shortest. Altimetric range has been characterised by various measures of *relief,* including local relief (altitude range over a local area, Smith, 1935); available relief (vertical difference between a former upland surface and degraded channel, Glock, 1932); and alternative measure of available relief (vertical difference between adjacent ridge and channel, Dury, 1951).

A number of significant problems arise from this form of quantification. Firstly, many of these early measures involved arbitrary or ambiguous definitions, such as 'significant wavelength', or 'former position of upland surface', making comparisons of measures derived by different authors difficult. Secondly there is significant dimensional overlap between measures, resulting in a redundancy in description. Attempts to remove descriptive redundancy using factor analysis (eg Lewis, 1969; King, 1968) or correlation (Doornkamp, 1968) have resolved this problem in part, although inadequate operational definition of indices make the interpretation of components difficult (Evans, 1972). A more systematic parameterisation of altimetric and planimetric variation was suggested and used by Evans (1972, 1979, 1980, 1984). He suggested that a unifying framework is provided by taking the first and second derivatives of altitude (slope, aspect, profile convexity and plan convexity). Dimensionally these measures, or *morphometric parameters*, are orthogonal and process related.

Evans (1979,1980) analysed the frequency distributions of morphometric parameters derived from DEMs of glaciated and mountainous terrains, and found they characteristically possessed

high mean and variability in slope, high variability in altitude, and a positive skew in profile and plan convexities. Frequency distribution based characterisations were carried out by Speight (1971) who analysed slope distributions and Ohmori and Hirano (1984) who identified characteristic hypsometric distributions. Evans (1984) applied factor analysis to the 4 momental measures (mean, standard deviation, skewness and kurtosis) of the altitude and its four derivatives (slope, aspect, profile and plan convexity). He identified 10 'key variables' from this analysis that allowed him to characterise local variations in terrain. Herdegen and Beran (1982) and Pike (1988) both used slope and curvature to characterise different landforms by constructing landform 'signatures' (Pike, 1988) from these parameters.

The process of landform *classification* is not one that is considered by this work, but it should be recognised that a successful surface characterisation provides the information with which to classify landform. Pike (1988) suggests that a successful surface parameterisation is necessary for a flexible terrain taxonomy. Geomorphometric classification of terrain has tended to be either into 'homogeneous regions' (eg Speight, 1968, 1973, 1976, Dikau, 1989) or the identification of specific geomorphological features (eg valley heads by Tribe, 1990). Richards (1981) highlights a number of problems with geomorphometry that apply to both approaches. In particular, the problem of scale of both spatial extent and resolution make single objective classifications of landscape unfeasible.

The notion of scale in geomorphological analysis is an important one, and forms the basis of this study. Evans (1979) recognised that many of the morphometric parameters measured from a DEM would vary with the size of grid used to model the surface. He distinguished this (sampling) effect from the size of area measured, which he found to be far less scale dependent. Frank *et al* (1986) found that scale dependency in operational definition of some surface features was sufficiently high to preclude objective definition altogether. In a manual measurement context, Gerrard and Robinson (1971) found high levels of variation in slope measurement with different sampling interval. Garbrecht and Martz (1993) analysed the effect of DEM grid size on automated drainage network identification concluding that the grid resolution should be no larger that 5% of the drainage basin area, if effective characterisation is to take place. What becomes clear from these (and many other) studies, is that it is unwise

to ignore the scale based effects of sampling interval when characterising surfaces.

2.3 Hydrological Characterisation

Probably the most widely examined and cited geomorphological feature extraction processes from DEMs concerns the derivation of hydrological and fluvial features from surface models. Consequently, it deserves special attention when reviewing existing surface characterisation work. The routing of water over the surface of a landscape represents a fundamental geomorphological process that is intimately tied to its form. The subdivision of the continuous surface into discrete hydrological units provides an important step in the geomorphological treatment of an elevation model. The literature on this specific process alone is large, Pike (1993) identifying over 100 refereed articles. Consequently, rather than provide a comprehensive (and repetitive) treatment of all papers in the field, this section attempts a categorisation of the techniques, identifying the major works in each category.

2.3.1 Manual Derivation of Drainage Networks

Prior to the use of automated hydrological characterisation from DEMs, features would either have to be measured directly in the field, or derived from secondary sources such as paper maps. Much of the rich literature on the pattern of drainage networks is based upon such extraction (eg. see Abrahams, 1984 for a review of techniques). If paper maps are used, drainage channels may be measured directly from the 'blue line' network, or inferred from contour crenellations. The accuracy of the blue line network clearly depends on the scale of the map source and quality of original surveying, but also the dynamism of the network itself. Networks with ephemeral streams may have particular symbolic representations (eg USGS 1:24000 topographic maps), or they may not be distinguished from permanent streams (eg, Ordnance Survey 1:50000 Landranger maps). The Ordnance Survey instructs its surveyors to represent the network at a constant hydrological state, at the "normal winter level" (Ovenden and Gregory, 1980).

If an alternative measure of drainage density is required, contour data may be examined so that channel form may be extracted. Many authors have questioned the accuracy of contour maps as sources of such information (eg. Drummond, 1974; Dunkerley, 1977; Mark, 1983) both because the accuracy of the contours themselves may be questionable and because there can be varying interpretations of the same contour data. Tribe (1990) noted that when two experienced geomorphologists were given the task of identifying the locations and areal extent of valley heads from a contour map, one consistently delimited a smaller head extent than the other.

2.3.2 Automated derivation of drainage networks from surface models

From the multitude of literature on ridge and channel extraction, it is possible to characterise all the methods according to five dichotomous classification criteria. These are summarised in Table 2.1. Criteria have been selected that are general enough to group any method of network extraction from any type or dimension of surface model. They could, for example, equally be used to classify the extraction of a least cost path from a cost surface, or migration path from a space-time 'volume'. This classification procedure was first described by Wood (1990) and is analogous to Lam's (1983) classification of spatial interpolation methods. In fact, the relationship between feature extraction and spatial interpolation is not an arbitrary one. Both involve the characterisation of as much useful *source* material as possible, from which a *target* is derived. The essential difference between spatial interpolation and feature extraction is that the former involves extracting a target that is the same quantity as the source (eg elevation from elevation) whereas the latter involves a further transformation (eg. from elevation to morphometric feature type).

(i) Topological / Geometrical

An analogy can be drawn between this dichotomous classification of feature extraction techniques and that of Lam's (1983) point/area interpolation dichotomy. Both define the metric used for the source and target. For interpolation the source and target are either 0-dimensional point space or 2-dimensional area space. For drainage network

extraction, the target is either n-dimensional space or a more abstract topological 'space'.

It has long been recognised that surface models contain important topological information that characterises a surface. Early work by Cayley (1859) and Maxwell (1870) described how any contour map describing surface form contains a set of unique topological relationships between *summits* (local maxima), *immits* (local minima), *bars* (lowest point dividing two immits), and *passes* (lowest point dividing two summits). From the topological connectivity of these point locations, the line features of *watercourses* and *watersheds*, and the areal features of *hills* and *dales*, could all be delimited. The exact distribution in space of these features is not fully defined, but rather the number of links and their degree of connectedness.

This topological characterisation has been developed more rigorously by Warntz (1966), Pfaltz (1976) and Wolf (1984, 1989, 1992). Wolf used the more standard classification of surface topology (described in Chapter 5) by considering bars and passes as having the same topological form. More significantly, he modelled these connectivity relationships using graph theory. Thus the topology of any surface could be described using a weighted surface network of pits, peaks and passes connected by ridges and channels. This has been used as the basis for generalisation (Mackaness and Beard, 1993; Wolf, 1989) and stream network identification (Wolf, 1992).

The topological characterisation of derived features as a concise description of surface form has been examined by many authors. Werner (1988) looked at the topological relationship between ridge and channel pattern within a drainage basin and derived a number of consistent relations. The topological characteristics of channel networks themselves have been examined in great detail, the most widespread approach incorporating the Random Topology Model (Shreve, 1966).

Network topology may not provide sufficient information to characterise hydrology for many applications. It must also be recognised that GIS, the platforms for most

automated extraction techniques, are not currently well equipped for processing topological surface data. Thus a distinction can be made between the techniques described above and the majority of extraction routines that are concerned with the *geometry* of hydrological features. Spatial location of surface features is required for much hydrological analysis (eg Lammers and Band, 1990), conversion to other data structures (Kumler, 1994) and realistic cartographic representation.

Conversion between topological and geometrical representation of drainage networks remains one of the outstanding problems of feature extraction. Wolf's surface network approach has limited application because it is difficult to attach spatial location to the processed weighted networks. Likewise, the fragmentation of networks produced by many of the geometric techniques (eg Peucker and Douglas, 1974; Jenson, 1985; Skidmore, 1990), makes the identification of topological relationships difficult. Two categories of solution to this problem have been adopted. Hutchinson (1989) describes a method of interpolating elevation using a `drainage enforcement algorithm' to force hydrological connectivity. This is done by identifying peaks and passes and forcing topological connectivity via channels that contain no pits. The alternative approach adopted by many authors (eg Band, 1986; Wood, 1990b) is to post-process the derived network to force topological connectivity. This may be in the form of line thinning (Skidmore, 1990), line joining and elimination (Wood, 1990*b*), combination of external data sources such as remotely sensed imagery (Riley, 1993).

(ii) Global / Local

Lam (1983) distinguishes between spatial interpolation routines that use some local neighbourhood in the interpolation of new values, and those that use the entire source. The same distinction may be made for the extraction of drainage features. As with spatial interpolation, the distinction is less a dichotomous one, more two ends of a continuum. For the purposes of classification, a distinction will be made here between three degrees of operation. Local extraction routines use a fixed window size that is less than the size of the entire surface mode. Quasi-local methods use an adaptive

window that is in most senses local, but may change size according to the characteristics of the surface mode. Global routines require information from the entire surface model for extraction.

Peucker and Douglas (1974) describe how a local 2 by 2 window may be passed over a raster while flagging the highest cell in each window. Cells that remain unflagged after the entire DEM has been scanned are described as channel cells. This procedure (which may be conveniently reversed to detect ridge cells) is based on the assumption that all channel cells will have at least once neighbouring cell of a higher elevation. This, and similar methods are referred to as *hillslope elimination methods* by Wood (1990a). The majority of local routines are based on more precise morphometric properties referred to as *concavity detection methods* by Wood (1990a). An alternate method suggested by Peucker and Douglas (1974), Jensen (1985), Band (1986) all use a fixed local 3 by 3 neighbourhood to identify 2-dimensional profiles from which channel shape (and other morphometric features) may be identified. Fixed local windows have been used ranging from 2 by 2 (Douglas and Peucker, 1974) to 7 by 7 (Zhang *et al.*, 1990).

Many authors recognise that using a fixed size window only allows channel features of a certain scale to be identified. Since this scale is arbitrarily defined by the resolution of the elevation model, it may not be always appropriate for feature extraction. Quasi-local methods allow the size of window used for sampling source data to vary according to the properties of the surface model. This variation can take several forms. Jensen and Domingue (1988) and Skidmore (1990) both describe methods similar to those of Jensen (1985) but where window size is allowed to expand in areas where profiles are ambiguous (eg. flat regions). Such methods have the potential to identify cross-sectional form at a variety of scales as the window expands. An alternative set of quasi-local techniques was adopted by Wood (1990b), where a variety of scales of surface form are examined using a range of window sizes.

Global feature extraction methods require the entire surface model to be examined

before any characterisation can be made. As with global spatial interpolation, every elevation value can potentially influence the outcome of characterisation. Collins (1975) used such a method by ordering all elevation values from highest to lowest before processing. This method, although computationally efficient disregards much of the natural spatial autocorrelation of real surfaces during extraction. Douglas (1986) shows that the method described by Collins may also produce incorrect results in certain circumstances. By far the most widely used group of methods of global hydrological characterisation is the *flow magnitude* method first described by Mark (1983a) and subsequently modified by many others (eg, Marks *et al.*, 1983; O'Callaghan and Mark, 1984; Band and Wood, 1988; Jensen and Domingue, 1988; Bennett and Armstrong, 1989; Band, 1989). This method (described in more detail below) involves traversing the entire surface model accumulating predicted surface flow. Drainage channels can be identified as made up of locations that have exceeded an (arbitrarily) defined flow magnitude threshold.

(iii) Exact / Approximate

This dichotomous distinction is similar but not identical to the exact/approximate distinction of spatial interpolation made by Lam (1983). Lam's distinction refers to the similarity between coincident source and target values. An exact interpolator will honour all source values such that spatially coincident source and targets will have identical values. Approximate interpolators may result in deviation between the two. Hydrological feature extraction can be regarded in a similar context. Given that all methods use some kind of morphometric characterisation, it possible to distinguish between those that honour elevation values precisely and those that make some approximation about local elevation values.

Approximate characterisation of elevation may be for a number of reasons. Several methods apply some preprocessing of the elevation model before extraction such as mean filtering (eg. Mark, 1983a; Band, 1986). Alternatively, the elevation may be

modified to ensure hydrological connectivity. Hutchinson (1989) forced interpolation of contours to eliminate any apparent pits modeled by contour lines. Morris and Flavin (1984) describe how planimetric position of contour lines may be altered so that they are consistent with a surveyed blue line network. Thus a deviation is introduced between original elevation values and those used for the extraction of hydrological features.

Two of the outstanding problems with feature extraction concern the characteristion of form of flat regions and 'noisy' elevation models. Several authors have suggested that a tolerance value should be associated with each elevation before using it to characterise morphometry (eg. Marks *et al.,* 1983; Skidmore, 1990). This is usually expressed as the vertical deviation required to define a real change in elevation. A third way of introducing approximate characterisation of surface form concerns those methods that model local neighbourhoods with some form of mathematical function or 'patch'. Evans (1979) suggested a bi-variate quadratic approximation of 3 by 3 neighbourhoods could be used to identify drainage features. Due to the overspecification of the six coefficients of the quadratic by the nine elevation values of the local neighbourhood, an approximate (but optimal) model of sampled elevations is produced. Mitasova and Mitas (1993) describe how a bivariate spline function can be used to detect channel features. Tension parameters are introduced that control the degree to which modelled values fit the original source data. Zhang *et al* (1990) apply a discrete cosine transformation to elevation, forming an approximate mathematical model of sampled points, before computing derivatives to find surface concavities.

Exact models honour all sampled points exactly during the extraction process. Zevenbergen and Thorne (1987) use a partial quartic expression to model a 3 by 3 local neighbourhood. Since this expression contains 9 coefficients, the model is specified precisely with no redundancy or deviation between sampled and modelled data. Other methods use the elevation values directly without patch modelling, pre-processing or using tolerance values (eg Jensen, 1985).

(iv) Direct / Indirect

In extracting hydrological features from an elevation model, two procedures may be adopted. Either the morphometry of the target can be identified and measured (eg. channel cross-section), or the target features may be associated with some other set of properties which can be in turn related to morphometry. There is a loose analogy here with the distinction between deterministic and stochastic interpolation methods. If the source and target values are both morphometric, it is possible to invoke a deterministic relationship between the two. If an indirect, intermediate rule must be invoked to relate the two, there is greater likelihood of indeterminacy.

Mark (1983a) defines a drainage channel as being made up of "*points at which fluvial processes are sufficiently dominant over slope processes*" (Mark, 1983a, p.169). If a measure of fluvial and slope processes can be made, so drainage channels can be (indirectly) identified. This indirect method of identifying drainage channels by the predicted volume of water flowing over their surface is the basis for many hydrological characterisations of surfaces (eg. O'Callaghan and Mark, 1984; Band and Wood, 1988; Band, 1989). Such methods are desirable when morphometry alone is not sufficient to characterise hydrology because other factors which may vary spatially have importance in hydrological modelling (eg Bathurst, 1986). Alternatively it may be that features may not be well defined morphometrically. For example drainage divides are relatively unambiguously defined in terms of their hydrological function, but may not be well expressed as morphometric ridge features. Conversely, channels may have a strong morphometric expression but have a widely varying hydrological role. Heavily dissected badlands in a semi-arid environment are particularly susceptible to such variation (eg loess plateaux described by Zonghu, 1986; Wood, 1990).

(v) Systematic / Recursive

The final dichotomy describes the way in which the feature extraction is applied spatially over the surface model. Systematic methods proceed in some orderly way that

is entirely independent of the characteristics of the source. Recursive methods are those which traverse the source in a pattern determined by the source itself. The parallel can be drawn with the gradual/abrupt distinction of Lam (1983). Gradual interpolation applies the same rules over the entire source whereas abrupt interpolation can involve the application of different rules at different points determined by 'barriers' in the source.

It should be noted that although the term recursive is used here, it does not necessarily follow that recursive algorithms are always used to implement recursive techniques. One of the most common recursive methods drainage network identification is to locate a drainage outflow point, recursively move upstream to the drainage divide and fall back to the outflow accumulating surface flow. This method has been implemented using recursive function calling by Marks *et al* (1983) and Band (1989), but non-recursively by O'Callaghan and Mark (1984).

A Classification of Hydrological Feature Extraction Techniques

Author	Topol/Geom	Loc/Quasi/Glob	Exact/Approx	Direct/Indirect	Sys/Recursive
Cayley (1859)	T	Q		D	R
Maxwell (1970)	T	Q		D	R
Pfaltz (1976)	T	Q		D	R
Wolf(1984,88)	T	Q		D	R
Band (1986)	G	L	E	D	S
Jenson (1985)	G	L	E	D	S
Peucker & Douglas (1974)[1]	G	L	E	D	S
Dikau (1989)	G	L	A	D	S
Evans (1979)	G	L	A	D	S
Zhang *et al* (1990)	G	L	A	D	S
Skidmore (1990)	G	Q	E	D	S
Speight (1968)	G	Q	E	D	S
Tribe (1990)	G	Q	E	D	S
Palacios-Velez *et al* (1986)	G	Q	E	D	S
Collins (1975)	G	G	E	D	R
Marks *et al* (1983)	G	G	E	D	R
Band (1989)	G	G	E	I	R
Band & Wood (1988)	G	G	E	I	R
Mark (1983a)	G	G	E	I	S
O'Callaghan & Mark (1984)	G	G	E	I	S
Jensen & Domingue (1988)	G	G	E	I	S
Bennett & Armstrong (1989)	G	L/G	E	D/I	S
Wood (1990b)	G	L/G	E	D/I	S

Where no entries are given, the method can be considered in either grouping. Where two alternatives are given, the method uses both groupings to produce the final extraction.

Table 2.1 A classification of some hydrological feature extraction techniques

2.4 Image Processing and Pattern Recognition

A consideration of the extraction of information from gridded elevation models cannot ignore the very similar process of information extraction from gridded digital imagery. In particular the image processing techniques of *texture analysis* and *pattern recognition* are considered here. This section will not consider the process of *image segmentation* (partition of homogeneous regions) or *mathematical morphology* (shape definition) which are more closely associated with specific geomorphology.

The analysis of *texture* within a digital image is closely allied to the geomorphometric

measurement of roughness. There appear to be no formal definitions of texture (Gonzalez and Woods, 1992) although many authors make the distinction between statistical texture, and structural components of an image (eg.Haralick, 1979; Lu and Fu, 1978; Tomita *et al*, 1982). Statistical texture is the description of the highest frequency of variation within an image, while structure is the arrangement of 'textural primitives' that make up an image (Gonzalez and Woods, 1992). It can be argued that the distinction is simply one of scale, such that the texture at one resolution becomes the structure at another. A large branch of image processing is concerned with spectral analysis of images where a (Fourier) transformation of an image allows periodicities at a range of scales to be examined (eg. Bajcsy, 1973).

Early measurements of statistical texture include analysis of frequency distributions (Darling and Joseph, 1968), and measures of autocorrelation (Kaizer, 1955) but, according to Haralick *et al* (1973), failed to adequately define or characterise texture. They proposed a method of texture analysis involving the computation of the *grey-tone spatial dependence matrix* (co-occurrence matrix) from a digital image. This two-dimensional matrix is, in effect, a measure of the spatial autocorrelation properties of an image. From this matrix, various summary statistics can be calculated measuring distinct textural properties such as entropy and linearity. Weszka *et al* (1976) assessed the effectiveness of these measures by comparison with Fourier techniques, concluding that in general, spectral analysis was less discriminating than the co-occurrence matrix measures.

It should be noted that the co-occurrence matrix is a resolution dependent measure. The degree of spatial association between grey levels in many images will depend in part on their distance of separation (a property measured by the variogram or correlogram (eg. Goodchild, 1986)). Such scale dependency has been measured using Fourier transforms, but the trigonometrical modelling of surface form makes this technique most suitable for images that comprise continuous and regular periodicities (Gonzalez and Woods, 1992). More recent developments in the use of *wavelet* transforms (Graps, 1995) may prove more promising in that they allow discrete non-periodic scale dependencies to be measured. Gallant and Hutchinson (1996) use such wavelet transforms to reconstruct DEMs over a (controlled) range of scales.

2.5 Fractals as models of Scale Dependency

Independently of the geostatistical tradition of characterising spatial dependency, there has arisen in the last 20 years, an alternative set of statistical tools that are able to describe aspects of scale dependency in spatial objects. Mandelbrot's persuasive treatises on *fractals* (Mandelbrot, 1975, 1977, 1982) have led to a prolific literature (over 100,000 articles according to Raffy, 1995) on scale-based characterisation of spatial and non-spatial phenomena. While much of the fractal literature is not strictly relevant to geomorphometry, there are a number of important ideas that have arisen from this field of study. It is ironic that one of the most persuasive aspects of Mandelbrot's work, according to Feder (1988), are the images created by Richard Voss (Voss, 1985) of "landscapes [that] look natural - one must believe that fractals somehow capture the essence of the surface topography of the earth" (Feder, 1988, p.3). In this context it is important to examine the relevant fractal literature in order to assess the contribution the theoretical advance has made to geomorphometry.

The essence of any fractal object is its self-similarity and can be defined as "a shape made of parts similar to the whole in some way" (Mandelbrot, 1986). More specifically, fractals can be defined in four contexts - deterministic or stochastic and self-similar or self-affine. Deterministic fractals can usually be defined by some mathematical function that is recursively applied over a wide range of scales. The result is a (usually geometric) object that contains scaled copies of itself. If these copies are identical bar their isotropic scaling and rotation, the fractal is termed self-similar (Mandelbrot, 1982). If the scaling process is itself anisotropic but linear, the fractal is termed self-affine (Loevejoy and Schertzer, 1995). Examples of deterministic self-similar fractals include the well known 19th century mathematical "monsters", the Koch Curve and Sierpinski Carpet (Mandelbrot, 1982). Although not directly relevant to geomorphometric characterisation, such fractals have been shown to be useful in developing spatial data structures to mimic surface spatial autocorrelation (Goodchild and Grandfield, 1983). Deterministic self-affine fractals have been largely used in image compression (Barnsely, 1988), botanical simulation (Govaerts and Verstraete, 1995) and atmospheric modelling (Lovejoy and Schertzer, 1995). More useful for the analysis and

description of terrain is the concept of statistical self similarity and self-affinity. Here measurable statistical parameters of an object are repeated at all scales. Thus, although a statistically self similar object will not contain exact copies of itself at all scales, the object would contain no geometric indication of its scale (Goodchild and Mark, 1987).

The fractal behaviour of surfaces (statistical self similarity) can be detected with many of the geostatistical techniques used for interpolation. In particular the variogram can be used to identify self-similar scaling since the log of variance plotted against the log of lag will yield a linear relationship for self-similar form (Rees, 1995). It is this behaviour that has one of the most important implications for spatial science in general and surface characterisation in particular. The so-called *Steinhaus Paradox* (Goodchild and Mark, 1987) is a recognition that the measurement of length increases with increased accuracy (Steinhaus, 1960). A similar observation known as the Schwartz area paradox (Feder, 1988) has been noted in the scale-dependent measurement of surface area of a volume by triangulation. Any truly self-similar object would have indeterminate measurable geometry since at whatever scale of measurement used, there will always be a finer level of detail that can only be approximated (underestimated) by that measurement. Richardson (1961) showed empirically that this is often the case when measuring cartographic lines. Schertzer and Lovejoy (1995) have shown this is also the case for measurements such as vegetation cover or albedo from increasingly higher resolution satellite imagery. The importance of this observation is that it suggests that any measurement of length (or area or volume) of a fractal object, must be accompanied by the scale at which the measurement was made if it is to have meaning. The corollary of this is that any raster based model of a surface, which by definition is recorded at a specific resolution scale, is only a measure of part of a fractal object. It should be with caution, that any analysis carried out at a particular raster resolution is extrapolated to any other scale.

The degree to which length measurements change with scale was investigated by Richardson (1961) by plotting the log of line length against log of measurement sampling interval. The resultant straight line relationship indicated a consistent power 'law'. Similar empirical results were found for time series data by Hurst *et al.* (1965) who observed through 'R/S analysis' (Rees, 1995), a constant power law relationship between measurement variance and scale.

Mandelbrot (1967, 1977) placed such observations in a fractal framework, by defining a fractional dimension $D = \log(n/n_0) / \log(\lambda/\lambda_0)$ where λ and λ_0 are two different length step sizes, and n and n_0 are the number of these steps that measure the length of a line. For any truly self-similar object the value D will be a constant regardless of the two scales chosen for measurement. In practice D would be estimated by taking measurement at a variety of scales and taking the best fit regression. This may be measured directly using the so-called *walking-dividers method* (eg. Schelberg *et al*, 1982), from Richardson plots, where gradient of the best fit line = 1-D (Goodchild and Mark, 1987), power spectra (Burrough, 1981; Pentland 1984; Huang and Turcotte, 1990), or the variogram (Mark and Aronson, 1984, Polidori, 1995).

A number of methods of measuring the fractal dimension of surfaces have been widely used. It has been suggested that a truly fractal surface of dimension D will contain profiles of dimension D-1 (Voss, 1988, p.30). These profiles may be measured in the XY plane (ie contours) (eg. Shelberg *et al*, 1983), or with vertical profiles (Rees, 1995). A number of other techniques rely on box or cell counting (eg Voss, 1988, Goodchild, 1982). Alternatively, a 2-dimensional extension of the walking-dividers method has been proposed by Clarke (1985) and modified to look at local variation by de Jong (1995). A comparative assessment of these techniques was carried out by Roy *et al* (1987).

Given that fractal dimension can be measured, it is possible to ask (and possibly answer) the more important question as to whether topography *is* fractal. Implicit in the argument of Wood and Snell (1957) is that the relief measured over a range of sampling scales can be used to predict relief characteristics at other scales. Mandelbrot (1967) provides some evidence that the British coastline is fractal suggesting a characteristic range of D=1.2 to 1.3. This implicitly suggests the terrain it bounds may also be so. Rather indirectly, he also suggests that landscapes are fractal since fractal simulations "look right" (Mandelbrot, 1982). Goodchild (1980) points out that although the west coast exhibits a relatively constant value of D (1.25), the east coast varies between 1.15 and 1.31. Burrough (1981) examined a wide variety of environmental data including topography identifying a characteristic (profile) dimension of 1.5. Clarke (1986) found the fractal dimension of the USGS DEM of the San Gabriel mountains, California to be 2.19. Huang and Turcotte found the mean fractal dimension from USGS 1

degree DEMs of Arizona to be 2.59 (Huang and Turcotte, 1989) and Oregon to be 2.586 with a standard deviation of 0.15 (Huang and Turcotte, 1990). Deitler and Zhang (1992) found the characteristic fractal dimension of Swiss alpine topography to be 2.43. Rees (1995) who examined topographic profiles of glaciated regions (Snowdonia, UK) and ice masses (Svalbard) using variograms, found D to vary between 1.1 (Snowdonia) through 1.3 (lowland UK terrain) to 1.4 for large ice masses.

What becomes apparent when comparing empirical results is that a single fractal dimension is not a very useful measure. It may fail to distinguish between contrasting topographies (eg. Huang and Turcotte, 1989, 1990; Fioravanti, 1995). The measure itself can depend on the procedure used to derive it (Roy *et al*, 1987). There is some degree of disagreement over what constitutes a 'typical' topographic value (2.3 - Shelberg *et al* (1983); 2.5 - Outcalt and Melton (1992); 2.2 - Voss (1988)). Most significantly, most empirical work suggests that landscapes do not possess a single fractal dimension, but a variety of values that change with scale. Of the 17 surfaces examined by Mark and Aronson (1984), 15 were said to show at least 2 characteristic values. At short lags ($<$ 1km) values were typically around 2.4, at larger lags, D appeared to be much higher (up to 2.8). Roy *et al* (1987) attribute this in part to sampling bias and possible anisotropic effects. Hakanson (1978), although not measuring fractal dimension explicitly noted a smoothing in lake shoreline as measurement size decreases. Outcalt and Melton (1992) noted a systematic spatial variation in D within DEMs.

Systematic variation in values of D can be interpreted in two ways. Polidori (1995) suggests a typical log-log variogram measured from a DEM will have three 'zones'. At the medium lags, self-similar (linear) behaviour will be evident, from which a fractal dimension could be derived. At larger lags, the variogram slope is unpredictable as the sample size rapidly decreases. At small lags, variance (and so fractal dimension) decreases more rapidly as the generalisation effects of DEM resolution begin to dominate. Alternative causes are attributable to geomorphic process. It is attractive to assume that since geomorphic process varies with scale, so might the surface form produced by such process. Mark and Aronson (1984) suggest the scales at which fractal dimension changes represent the change in dominance from one form of geomorphic process to another. Certainly, there is additional empirical evidence for

characteristic scales or breakpoints in fractal dimension. For example Dietler (1995) found Swiss alpine topography to be effectively made up of self-affine units of size 5km. Gilbert (1989) noted a general linear trend, but with local non linear stretches over limited scales.

While the overwhelming weight of evidence suggests that topography is at best fractal over a limited range of scales, the concept forces the recognition of two important points. Firstly, the scale at which surfaces are modelled and analysed will in part determine surface characterisation. Secondly, the very absence of scale independence is itself a useful diagnostic, be it through systematic change in fractal behaviour, or as characteristic scales of change. More recent development in the field of *multifractals* is an attempt to recognise and model scale and attribute dependencies. Lovejoy and Schertzer (1995) suggest that traditional (mono)fractal analysis is inappropriate for modelling what they regard as multifractal topography. They suggest that fractal dimension systematically decreases with altitude. Again this has an appeal to geomorphologists who acknowledge different upland and lowland process and form. Multifractal characterisation replaces a single scale invariant value with a scale and attribute dependent function D(q). Fioravanti (1995) uses such D(q) functions to characterise the texture of SAR images and shows the technique to be far more discriminating than either a single fractal dimension or other traditional texture measures (such as the co-occurrence matrix measures). Allied to the use of multifractal analysis is the use of wavelet transforms to specifically map discrete scale dependencies within the frequency domain (Graps, 1995).

Chapter Three - DEM Uncertainty.

The certainty with which we can assume a DEM represents true surface from, is a function partly of the conceptual limitations of the model (discussed in the previous chapter) and partly the quality of the data provided. This chapter considers methods of quantifying, visualising and modelling DEM uncertainty.

3.1 Quantifying DEM Data Uncertainty

This section provides a description and evaluation of uncertainty quantification methods commonly available, as well as the development of several new descriptors. They are presented in approximate order of completeness in their description. It is recognised that a compromise is sought between a concise and conveniently measured quantification and a comprehensive description. As a consequence, some descriptions (for example those requiring a second independently derived model of topography) may not be applicable in all contexts. Nevertheless, all methods described in this chapter highlight an important aspect of data quality. These measures are applied to several Ordnance Survey elevation models from which an evaluation of their uncertainty is made. Part of this experimental work was carried out at an Ordnance Survey 'Summer Workshop' from July-August, 1993 (Wood, 1993).

3.1.1 Momental Descriptions

The conventional descriptions of a frequency distribution that include measures of central tendency and dispersion may be used to describe the pattern of deviation between two sets of elevation data. These measures are part of a set of what are described as momental statistics (eg. Miller and Kahn, 1962, p.42). All of the following measures require two sets of data that describe the same elevation property. Although it is desirable that the datasets be independent with one being "of a higher accuracy" (SDTS, 1990), neither true independence, or judgements of relative accuracy are always necessary in order to derive a meaningful measure (eg. Yeoli, 1986).

(i) Root Mean Square Error (RMSE)

The most widely used measure for reporting accuracy is the root mean squared error (RMSE) used by, for example, both the USGS (USGS, 1987) and the Ordnance Survey (Ordnance Survey, 1992). It is a dispersion measure, being approximately equivalent to the average (absolute) deviation between two datasets.

$$RMSE=\sqrt{\frac{\Sigma(z_i-z_j)^2}{n}} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (1)$$

where z_i and z_j are two corresponding elevation values
n is the number of elevation pairs modelled.

The larger the value of the RMSE, the greater the difference between two sets of measurements of the same phenomenon. It would be usual therefore to use this as a quantification of the uncertainty of one or both sets of measurements. Its widespread use can be attributed to the relative ease of calculation and reporting (usually a single figure) and the ease with which the concept can be understood by most users of elevation data.

There are a number of problems with the measure and the way in which it is often derived. The index does not involve any description of the mean deviation between the two measures of elevation. Most interpretations of the value will involve an assumption of zero mean deviation - one which is not always valid (Li, 1993a, 1993b; Monckton, 1994). A zero mean without spatial variation also implies stationarity, which again is not always a desirable property of a general error model (Heuvelink, 1993; Cressie, 1993). Since the variance of elevation deviation is likely to be attributable to a different set of processes to variation in means, this single measure may hide important information.

Inevitably a single measure of dispersion describes nothing of the form of frequency

distribution. Properties such as balance between small and large deviations, skewness of the distribution, are not revealed by the measure. Because the distribution function cannot be modelled it is not possible to associate reliably, a probability of individual deviations in the distribution occurring by chance. Stochastic error modelling (eg Fisher, 1990,1991a; Heuvelink, 1993; Monckton, 1994) requires an assumption of the nature of frequency distribution function.

The most common use of the RMSE is to provide a single global measure of deviation. Consequently, there is no indication of spatial variation over the surface of the DEM. Empirical evidence of Carter (1989), Guth (1992), Wood (1993), and Monckton (1994) suggests that error can be positively spatially autocorrelated. Non-random spatial variation in error cannot be revealed by a single aspatial measure. When creating an error model, it is desirable to separate the spatial trends from the non-spatial error component (Goodchild, 1986; Heuvelink, 1993)

The RMSE has a dimension of [L], and is consequently usually measured in the same units as the original elevation data. This makes comparisons of RMSE values for areas with different relative relief values hazardous. The magnitude of the RMSE depends not only on our intuitive idea of error but on the variance of the true elevation distribution. This 'natural' variance will depend on relative relief, but also the spatial scale of measurements.

(ii) Accuracy Ratio

In order to eliminate relative relief effects from measurement deviation, the RMSE can be divided by a measure of relative relief. This is most conveniently achieved by dividing by the standard deviation of elevation measurements.

$$\textit{accuracy ratio} \quad a = \sqrt{\frac{\Sigma(z_i - z_j)^2}{\Sigma(z_i - \bar{z}_i)^2}} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(2)}$$

where z_i and z_j are measured as above
$\bar{z}_i$ is the average elevation of i.

This dimensionless ratio, although not reported in the literature, can be used for comparison at different spatial scales and between different terrain surfaces.

(iii) Mean and Standard Deviation

These measures allow systematic deviations between values to be separated from symmetrical dispersion (Li, 1988, 1993a, 1993b; Monckton, 1994).

$$\overline{dz_{ij}} = \frac{\Sigma z_i - \Sigma z_j}{n}$$

$$s = \sqrt{\frac{\Sigma[(z_i - z_j) - \overline{dz_{ij}}]^2}{n}} \qquad \text{. . . . . (3)}$$

where z_i and z_j are measured as above
dz_{ij} is the mean deviation between models
s is the standard deviation between models.

While these measures cannot identify non-stationarity, a global (DEM wide) trend may be measured as distinct from a more local error component. A non-zero mean value indicates that overestimates and underestimates of elevation are not equal, and that the overall accuracy of the model can be improved simply by subtracting the mean deviation from all elevation values. Measuring the systematic deviation in elevations can be used as the basis of *deterministic error modelling* (discussed in section 3.3).

(iv) Higher order moments

Measures of skewness and kurtosis have been used to characterise elevation distributions (Evans, 1979, 1980) and can be applied to the distribution of elevation model deviations.

$$\overline{dz_{ij}} = \frac{\Sigma z_i - \Sigma z_j}{n}$$

$$skewness = \frac{\Sigma[(z_i - z_j) - \overline{dz_{ij}}]^3}{ns^3}$$

$$kurtosis = \frac{\Sigma[z_i - z_j) - \overline{dz_{ij}}]^4}{ns^4} \qquad \text{. . . . . (4)}$$

While the use of higher order moments in error reporting is somewhat limited in the literature, these measures could provide a fuller description of the distribution of error while remaining parsimonious.

(v) Accuracy Histogram

In order to develop a reliable stochastic error model it is necessary to model the form of the accuracy distribution. While the moments described above help in determining the parameters of such a distribution, they do not fully describe it. If we regard model errors as independent random variables, the central limit theorem suggests distribution of those errors to be normal, with a mean of 0 and standard deviation (standard error) s defined as above. In such cases, the distribution could be defined by a single RMSE value. However, as suggested above, this is unlikely to be the case for many elevation models. By comparing the modelled and actual frequency distribution of accuracy, it is possible to determine the degree to which error is successfully represented as an independent random variable.

3.1.2 Spatial Measures

Many authors have suggested that the distribution of errors in elevation models will show some form of spatial patterning (eg, Guth, 1992; Li, 1993a; Wood and Fisher, 1993; Monckton, 1994). In order to both test this idea and build models upon it, it is necessary to provide descriptions of the spatial pattern of error.

(i) Spatial Autocorrelation

By identifying the degree of clustering of error, it is possible to produce stochastic models of error that reflect to some degree, the spatial pattern of accuracy (Monckton, 1994). Spatial autocorrelation is commonly measured using Moran's I statistic (Cliff and Ord, 1981) which provides a convenient measure of the degree of spatial association between similar error values.

$$I = \frac{n.\sum_{u=1}^{n}\sum_{v=1}^{n} w_{uv}\,(dz_u - \bar{dz})\,(dz_v - \bar{dz})}{\sum_{u=1}^{n}(dz_u - \bar{dz})^2 \sum_{u=1}^{n}\sum_{v=1}^{n} w_{uv}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \textbf{(5)}$$

where dz_u is the deviation between the two models for each cell;
dz_v is the deviation between the two models for some neighbouring cells;
w_{uv} is the weighting given to neighbouring cells;
dz is the average deviation between the two models.

The statistic indicates clustering of values (approximately I→1), random distribution (approximately I→0) and local separation of similar values (approximately I→-1). Exact limits are given in Goodchild (1986) and Bailey and Gatrell (1995). It can be used in the generation of stochastic error models or in the examination of residuals from deterministic models (see section 4.3).

(ii) Variograms and Correlograms

Single measures of spatial autocorrelation are somewhat limited because they provide only a global description of spatial association, and because they are confined to a single spatial scale (determined by the separation between dz_u and dz_v above). It is reasonable to expect that the degree of clustering of error will vary at different scales, and indeed this is supported by empirical evidence (Monkton, 1994). A more useful description of spatial pattern is the variogram or correlogram which describe the change in autocorrelation with scale. For a discussion of the problems associated with measuring variograms from rasters see section 4.5.

(iii) Accuracy surface

The fullest description of model uncertainty can be achieved by creating a continuous surface of measured accuracy. This may be calculated directly if two models of elevation are available (Z_i - Z_j). From this surface, any of the above measures may be calculated. Additionally, visualisation of the accuracy surface can reveal pattern that is not reflected in the statistical measures described (see section 3.2).

In practice, it is rarely the case that two full and equivalent models of elevation are available for comparison. For such cases two possible alternatives may be used each of which provides an estimate of the likely accuracy surface. If spot heights are available in addition to a DEM (for example, Monkton, 1994 digitized from a paper map), the difference in elevation values may be found at spot height locations. Where there is some ambiguity between location and DEM cell boundary a local planar or quadratic trend surface may be fitted through the local DEM cell neighbourhood. The value of this trend function can be calculated precisely at the spot height location. Once a series of accuracy values have been determined, it is useful to interpolate these over a continuous surface to aid visualisation and detection of trends (Wood, 1993). There are two major problems with this method, namely the representative selection of spot height locations and the arbitrary method of spatial interpolation. Spot heights included on topographic maps (eg Ordnance Survey 1:50000 and USGS 1:24000) are not randomly sampled. They tend to occur either along engineered features (usually roads), or at selected topographic locations (usually local peaks). Kumler (1994) observed similar problems with point based elevation values from USGS topographic data. The topographic bias of spot heights may undermine the representativeness of the values as there is evidence that local topography is not independent of elevation accuracy (Wood, 1993).

An alternative method of accuracy surface construction is available where the DEM is to be interpolated from source elevation data. By taking a series of selected samples from the source elevation data, a number of model realisations may be constructed. The variance in elevation between these models provides a measure of accuracy at all locations. Methods that fall into this category include the production of error variance surfaces as part of the Kriging process (eg. Isaaks and Srivastava, 1989) and multiple one-dimensional interpolation suggested by Yoeli (1986) and investigated by Wood and Fisher (1993) (see also section 3.2). These methods only work when all samples can be regarded as being equally representative of the interpolated surface.

3.1.3 Hypsometric Analysis

In many cases, DEM accuracy cannot be estimated because no alternative models or sources of elevation data exist. Where this is the case, an assessment of accuracy can still be made by identifying internal inconsistencies within the DEM. This can be achieved through visualisation (see section 3.2) or by analysis of the frequency histogram of elevation (the non-cumulative equivalent to the hypsometric curve identified in Monkhouse and Wilkinson, 1952). While analysis of this distribution has formed the basis for general geomorphometry (Evans 1972,1979,1980; Mark, 1975a), the hypsometric integral appears very sensitive to certain types of error.

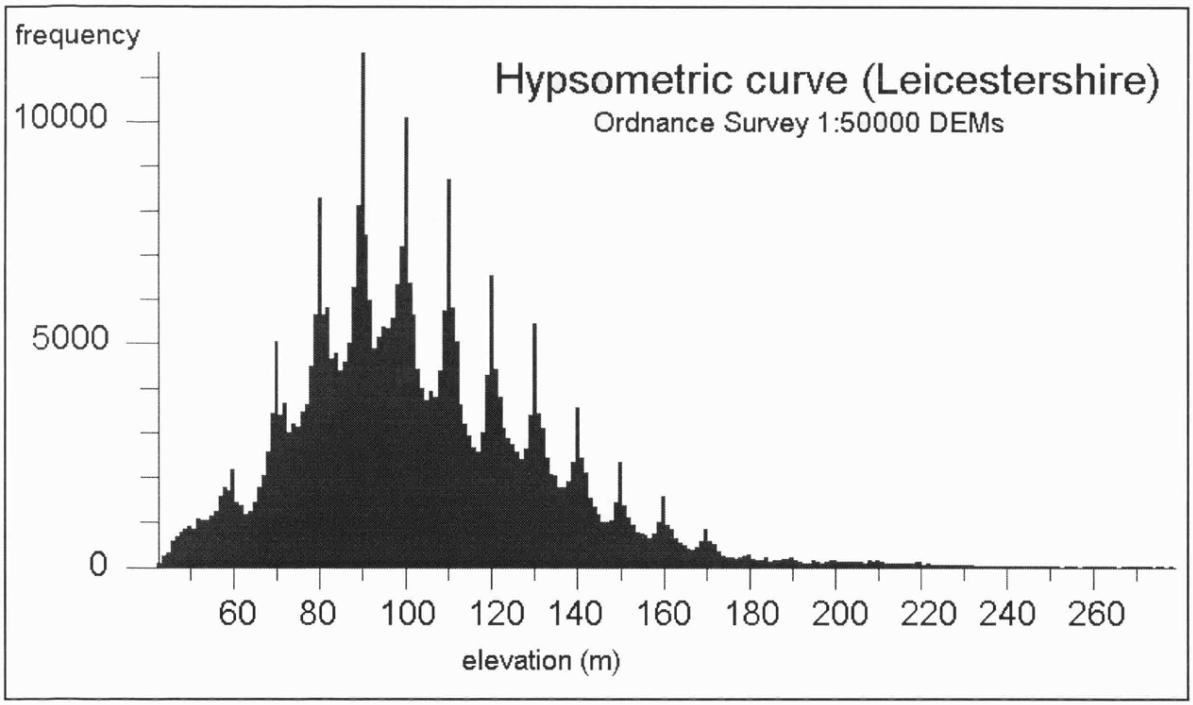


Figure 3.1 - Hypsometric curve for Leicestershire region showing systematic elevation bias.

Figure 3.1 shows the frequency histogram of an Ordnance Survey 1:50k DEM of Leicestershire. The elevation range is typical for lowland Britain, ranging between 0 and 260m above OD. What dominates however, is the 'spikiness' of the distribution with elevations of 60, 70, 80, 90m etc. being twice as common as 65m, 75m, 85m, and 95m etc. The elevations at which these peaks occur is unlikely to be a function of the Leicestershire landscape since 10m (the spike interval) is an arbitrarily defined distance.

Histograms were calculated for all Ordnance Survey 1:50k DEMs in Great Britain to see if this pattern occurred elsewhere, or if there was any evidence of a regional trend. The results were plotted on a single map using LaserScan software at the Ordnance Survey (Wood, 1993; Robinson, 1993). The X-axis of each histogram was scaled to cover the 20km tile it represented (see Figure 3.2).

Detailed examination of these histograms revealed that over 99% showed some degree of spikiness, and that these overrepresented elevations occurred in multiples of 10m. The consistency of this pattern clearly indicates a non-topographic cause since it occurs over such a wide region and variety of topographic areas. Given that all OS 1:50k DEMs were interpolated from contours that were recorded at 10m intervals, it seems reasonable to suggest that these contour values remain as 'artifacts' arising from the interpolation process. Such artifacts should also be regarded as erroneous since the height at which they were recorded is arbitrary (contours recorded at 2m, 12m, 22m etc. would have produced a different set of DEMs).

To quantify the degree of spikiness associated with each DEM, histograms were 'collapsed' into their modulo 10 equivalents. That is, each elevation value was divided by 10 and the remainder placed in a category 0 to 9. It would be expected that the frequency of values in each category would be approximately equal for a well interpolated surface with a sufficiently large range (approximately 30m+). Absences from this distribution indicate artifacts of interpolation. Figures 3.3*a-c* show the modulo distributions for the Leicestershire DEMs. Figure 3.3*a* shows the entire modulo distribution (including the frequency distribution or 'hammock plot'), indicating a clear excess of *mod0* and *mod9* values (red), and a deficit of *mod5* values (blue). The distributions of *mod0* and *mod5* cells are shown separately in Figures 3.3*b* and 3.3*c*. As would be expected for any spatially autocorrelated surface Figures 3.3*b* and 3.3*c* both show a contoured appearance. They also indicate flatter areas that may validly contribute to an excess of a particular modulo value (eg, the valley in north-east corner of the image). However, the *mod0* values appear consistently as thicker 'contours' than their interleaved *mod5* equivalents.

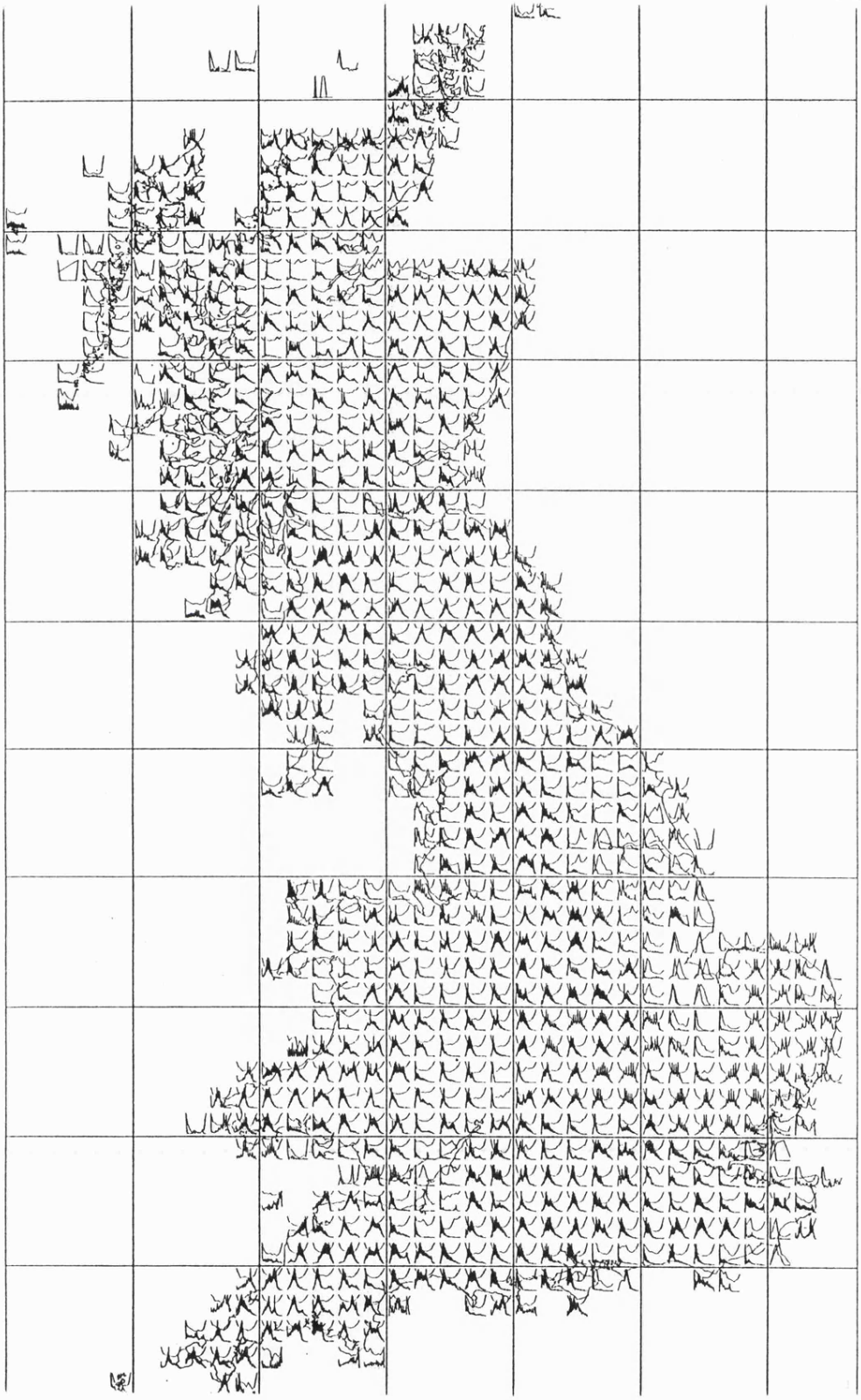
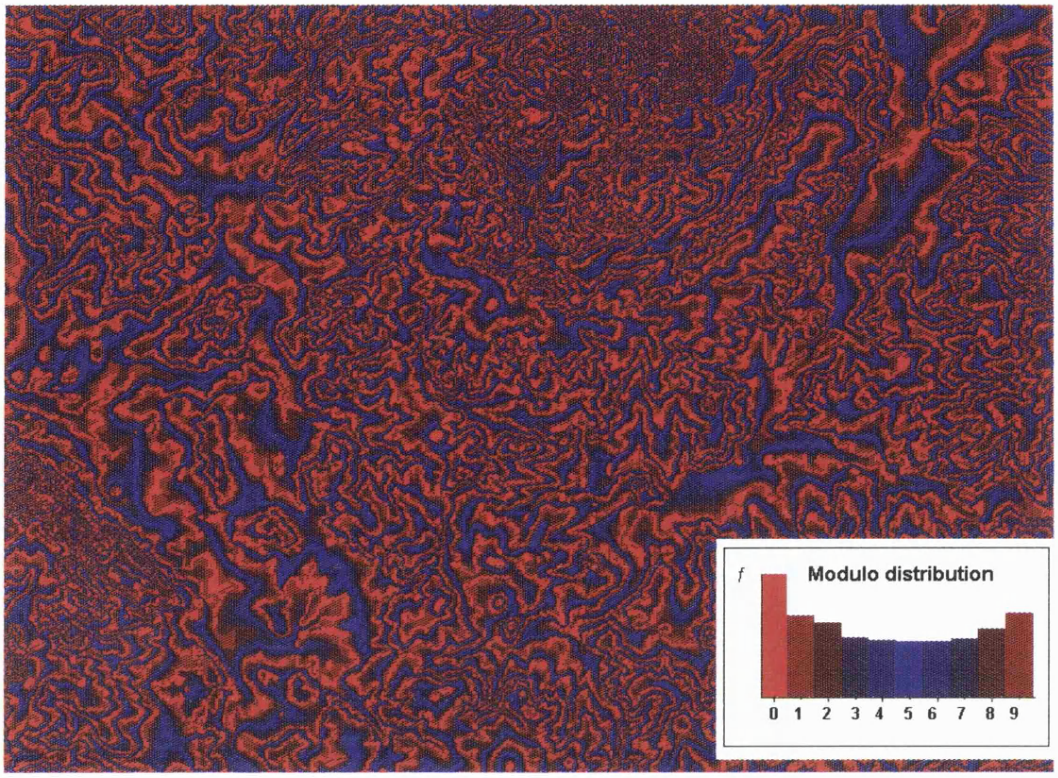
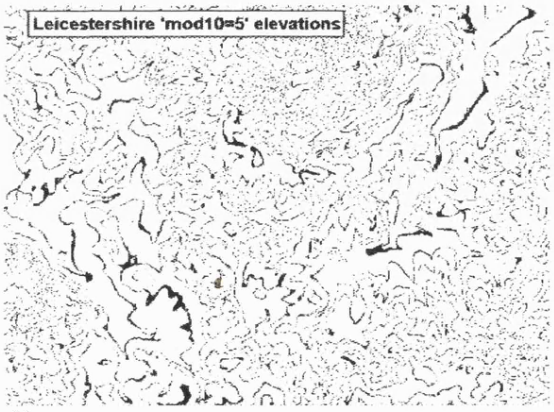


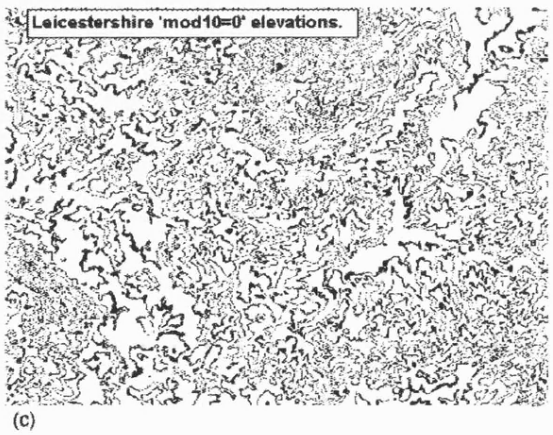
Figure 3.2 - Hypsometric curves and 'hammock plots' for entire GB 1:50k DEM coverage



(a)



(b)



(c)

Figure 3.3 (*a*) Leicestershire DEM - modulo values; (*b*) modulo 5 cells (ie 5m, 15m, 25m etc); (*c*) modulo 0 cells (10m, 20m, 30m etc.).

While histograms and hammock plots provide a qualitative description of interpolation artifacts, a quantitative measure of the phenomenon can be measured by comparing the observed and expected modulo frequencies assuming an even spread of remainder values. For the general case of a surface interpolated from contours of vertical interval n units, let the Hammock index, H be defined as,

$$H = \frac{(n \cdot f_0) - \sum_{i=1}^{n-1} f_i}{\sum_{i=0}^{n-1} f_i} \qquad \text{(6)}$$

That is, the area of the *hammock plot* between the frequency of *mod0* (f_0) cells and the frequency of all other modulo values (f_i). This is standardised by dividing by the area under the graph. Results will vary between *(i-1)* and -1. The limiting cases of H are shown, along with a typical OS DEM distribution in Figure 3.4.

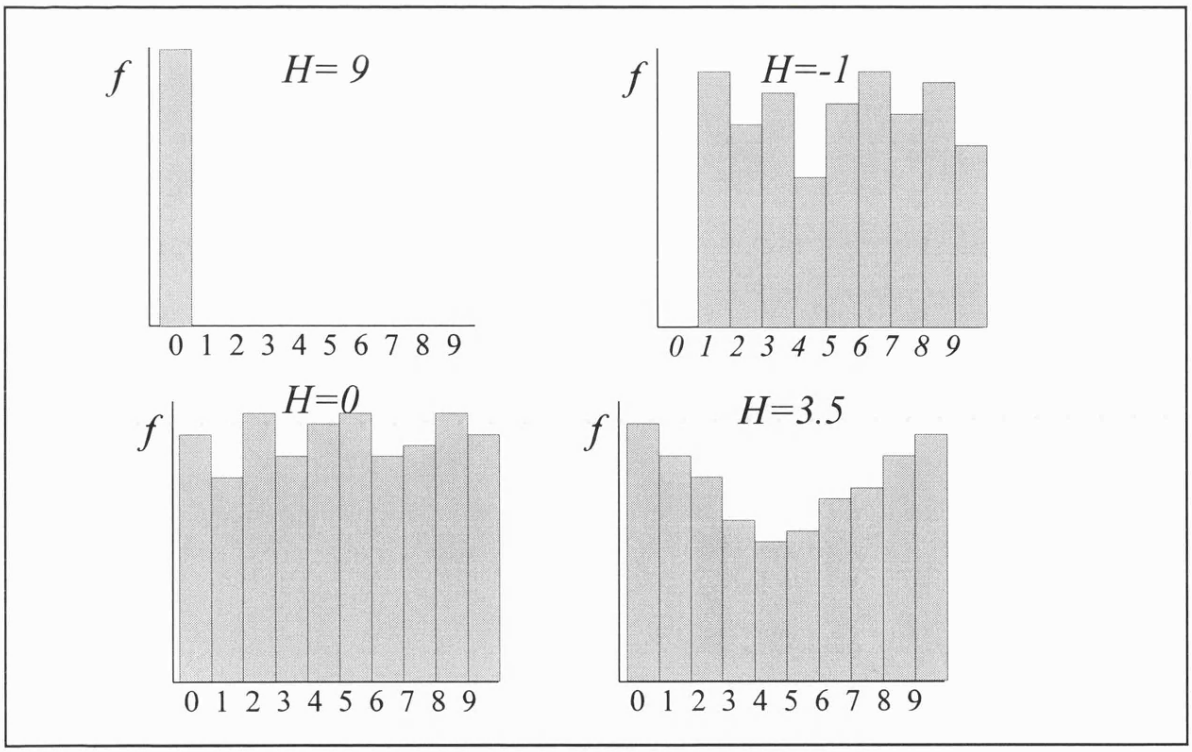


Figure 3.4 - Hammock plots with corresponding Hammock indices. The top row indicates the two limiting cases, the bottom, a random expectation (*H=0)* and a typical value for Ordnance Survey 1:50k DEMs (*H=3.5*).

Figure 3.5 shows this index plotted for all OS DEMs in Wales and western England. For greater clarity, each 20x20km DEM was divided into 5km quadrants, and the index calculated for each. Darker blue areas indicated a higher value of H, white indicating no bias and red, a negative value. Additionally, tiles have been superimposed on a shaded relief map of the region to indicate the relationship between contour artifacts and local relief.

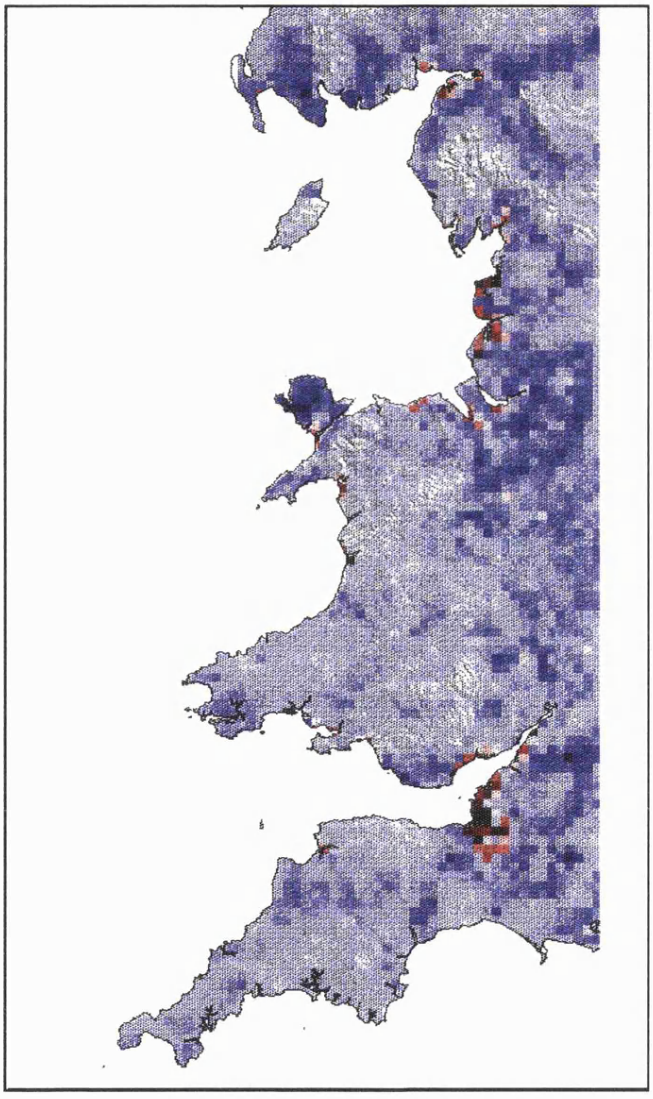


Figure 3.5 Hammock Index for Wales and western England.

The most striking observation from the hammock map is that nearly the entire map is blue. The only exceptions occur in very low lying relief where there is an insufficient elevation range for *H* to be valid. All Ordnance Survey 1:50k DEMs appear to be afflicted to a varying degree with contour artifacts. There appears to be a systematic relationship with local relief, with the darkest blue DEMs occurring in areas of low to moderate relief (eg Anglesey, Cheshire Plains and Northern Wiltshire). The effect appears least in areas of high local relief (eg, Lake District, Snowdonia and Brecon Beacons).

It has already been suggested that the overabundance of *mod0* values in a DEM is a response to the original 10m vertical contour interval before interpolation. Unfortunately, the precise interpolation procedure used by the Military Survey for the Ordnance Survey, has not been published or made available (see Wood, 1993). Therefore, the reasons for the relationship can only be hypothesised. Three likely explanations are given below.

(i) Exact Interpolators.

Using the nomenclature of Lam (1983), an exact interpolator is one that honours all available source data values. In the case of a DEM interpolated from contours, all original contour values would remain after interpolation. Thus, one would expect *H* to increase as the density of contours relative to the DEM resolution increases. However, the observation that high relief areas such as Snowdonia have relatively low values of H indicates a more subtle relationship.

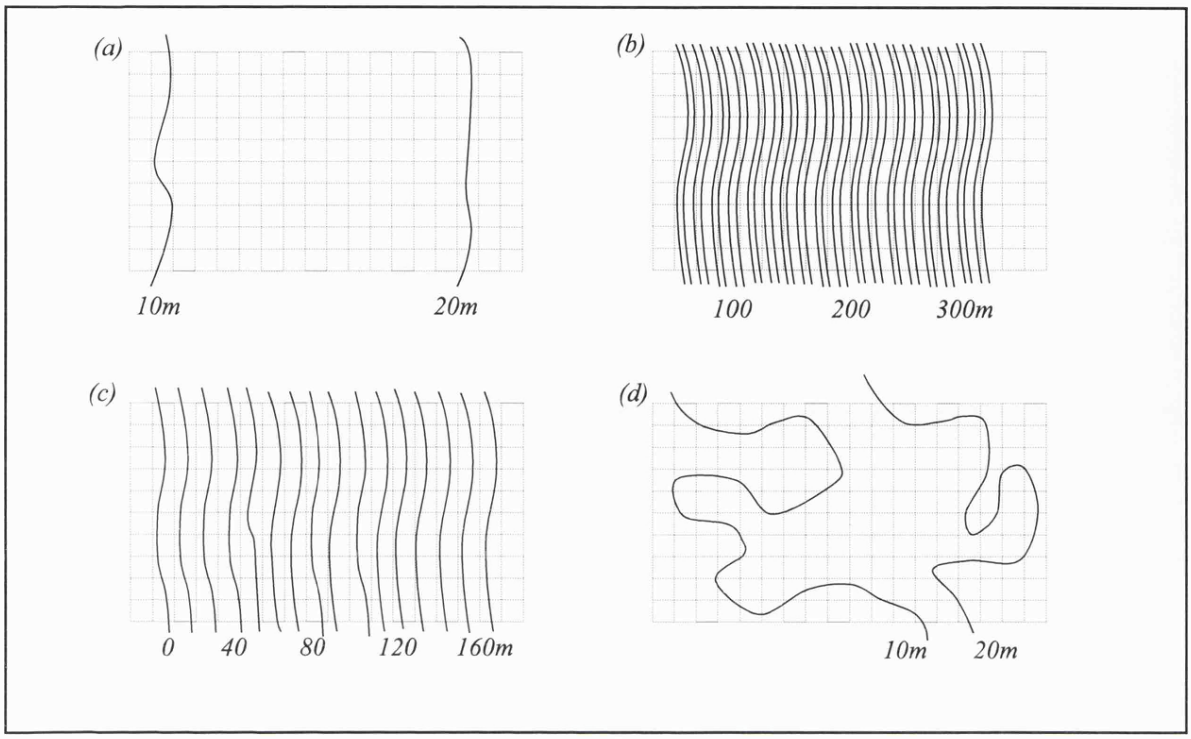


Figure 3.6 - Four contour distributions that give rise to high and low values of *H* (see text for details).

Figure 3.6 shows four alternative distributions of contour lines. In case 3.6*a* (shallow slope) where contours are widely spaced relative to the DEM resolution, only a small proportion of DEM cells coincide with the source data. The cells between will occupy the full range of modulo values. In case 3.6*b* (very steep slope) many contours occupy the same DEM cell. In this case, even if the interpolator is an exact one, only a single value can be returned for each cell. If this is based on any form of average, elevation values are likely to occupy a full range of modulo values. Case 3.6*c* (moderate slope) would provide the highest value of H since contours are (planimetricly) placed at the DEM resolution. For a DEM with 50m resolution and contour interval of 10m, this would correspond to a slope of between 1:5 (rook's case) and 1:7 (bishop's case), or between 8 and 11 degrees. This critical slope value would be somewhat less for case 3.6*d*, where contours 'wander' over the surface. The degree of space filling by contours can be quantified by the fractal dimension.

(ii) Contours as sets of independent points

The conventional arc-node structure that is used to store digital contours in most GIS

stores each contour line as a collection of p points that join $p-1$ line segments. Interpolation routines that treat these lines as samples of independent points will inevitably overrepresent contour elevations. Interpolation routines that search for the n nearest points will usually select most from the same contour line, producing characteristic artifacts (see Figures 3.14 to 3.18 for examples of this).

(iii) Contour Triangulation

The LaserScan software used by the Military Survey for the creation of Ordnance Survey DEMs relies, in part, on a triangulation of contour lines (Robinson, 1993). Figure 3.7*a* shows how two parallel contours are triangulated. Regardless of the density of points along contours, a relatively accurate linear interpolation will be applied between adjacent contour lines. In case 3.7*b* however, where crenellations occur, triangulation can be made across the same contour. This well known problem can be overcome by adding form lines before triangulation - a process that has undoubtedly occurred in the case of the Military Survey. However, form lines are usually only placed along major breaks in slope. There will be many cases of minor contour crenellation where erroneous triangulation will occur. The overall effect is to overrepresent contour elevation values after interpolation.

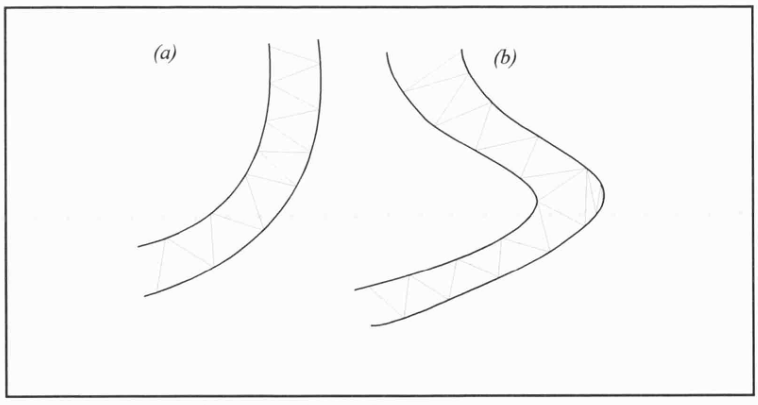


Figure 3.7 - Triangulation of adjacent contours.

3.2 Visualising DEM Data Uncertainty

It has already been demonstrated that plotting the histogram of elevation values from a DEM provides an immediate visual indication of certain types of DEM error. Visualisation provides a powerful mechanism for identifying the spatial distribution and possible causes of DEM uncertainty. This section demonstrates how the functionality available with most raster based GIS can be used to identify patterns of error. In particular, the emphasis is placed on those methods that allow internal inconsistencies to be identified where no 'ground truth' or data of a higher accuracy are available. The aim of this section is therefore twofold; to identify the data quality of the DEMs used as part of this work, and to demonstrate a simple methodology for identifying DEM quality for users of GIS.

3.2.1 Four Interpolation Methods

For the purposes of DEM quality validation, a fractal surface was created using spectral synthesis (Saupe, 1988; for more details on this method see section 6.1.3 and *r.frac.surf* in the Appendix). The resultant DEM, *fractal* with unit resolution, consisted of 200x200 cells, a mean value of 1719, standard deviation of 701, range from 1 to 3597, and fractal dimension of 2.01. This surface was treated as 'ground truth', with which four alternative surfaces were compared. Contours were threaded through *fractal* at 400m intervals using the GRASS function *r.contours*. These contours were interpolated using four interpolation methods to produce four DEMs all with possible error. *fractal* and its contour representation are shown in Figure 3.8. Contours were deliberately sparse relative to the DEM resolution so that the likelihood of interpolation error was increased. All methods of interpolation investigated here are likely to perform much better with 'real' data at appropriate raster resolutions.

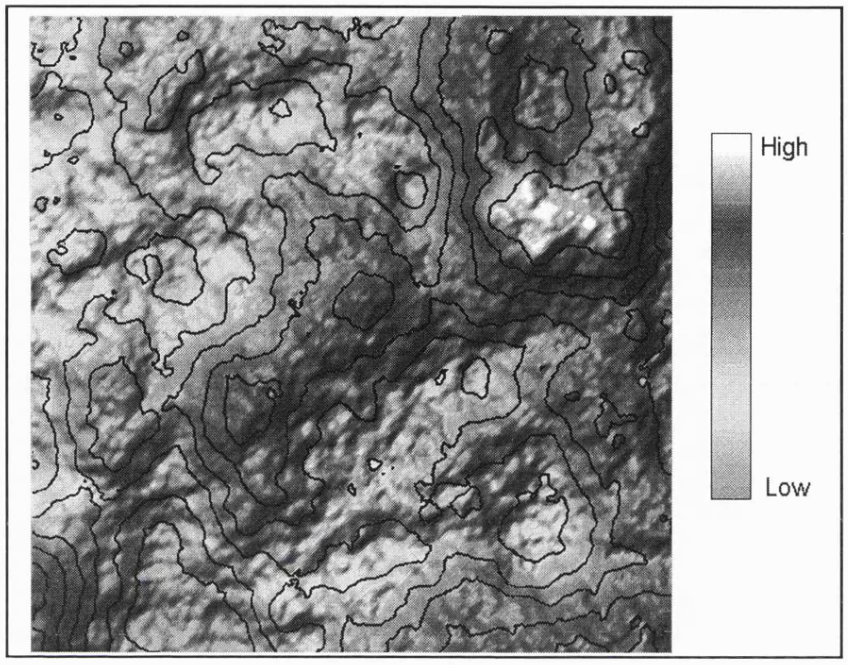


Figure 3.8 - *fractal* surface and 400m contours used for interpolation.

Three of the four interpolation methods were available in the GIS GRASS, the fourth, *v.surf.spline*, was created for the experiment and is discussed in more detail below.

(i) *s.surf.idw*

The simplest interpolation method used is also one of the commonest, using simple inverse distance squared weighting of source data to interpolate a target.

$$z = \frac{\sum_{i=1}^{i=n} (1/d_i^2 . z_i)}{\sum_{i=1}^{i=n} 1/d_i^2} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (7)$$

where z is the interpolated value,
d_i is the distance between z and each of n nearest points
z_i is one of the n nearest points.

Source points (z_i) are provided in the form of GRASS sites (point based) files - the point locations of the vector contour vertices. These were converted from line vectors into point values using the GRASS program *v.to.sites*. Inverse distance weighting of

nearest neighbours is best suited to a non-clustered distribution of source points. Clearly, contour vertices are clustered, and so it would be expected that this routine should perform poorly in the generation of an error free DEM.

(ii) *r.surf.contour*

GRASS provides an interpolation routine specifically designed for interpolating contour data. This routine, *r.surf.contour*, uses a contour floodfilling technique to overcome the irregular distribution of source values along contour lines.

$$z=\frac{(1/d_u.z_u)+(1/d_d.z_d)}{(1/d_u+1/d_d)} \qquad \text{(8)}$$

where z is the interpolated value,
d_u is the upslope distance to the nearest contour
d_d is the downslope distance to the nearest contour
z_u and z_d are the two bounding contour values.

This routine requires rasterised contour which were generated using the GRASS function *v.to.rast*. For each point to interpolate, a flood fill is generated from that cell until two unique contour values are found. A linear distance weighted average is then given between the two adjacent contours. Where an interpolated cell coincides with a contour, it is given the contour's value. Where interpolated cells are bounded by a single contour line (at local maxima or minima), the cell is also given the bounding contour value. Thus, without the addition of spot heights, it would be expected that interpolation using this method would produce truncated maxima and minima.

(iii) *s.surf.tps*

This GRASS routine interpolates using regularized 2-dimensional spline fitting with tension (Mitasova and Mitas, 1994). This bivariate procedure attempts to interpolate a smooth surface through all source data points. The degree to which local values are honoured is controlled by a tension parameter. Low tension gives a smoothed generalised surface, high tension honours source points precisely, but gives the appearance of a rubber sheet stretched over source points.

Additionally, for maximum computational efficiency, *s.surf.tps* uses a quadtree segmentation process that recursively subdivides the raster before searching for neighbouring values. The resolution and size of segmentation can be controlled by user defined parameters. The selection of smoothing, tension and segmentation parameters is a somewhat subjective process that depends on the nature of the modelled surface (Mitasova, 1992). For the purposes of this experiment, all default values were selected, remembering that the objective is not to produce an optimal interpolation, but a surface that may contain DEM error.

s.surf.tps parameters:

tension = 40.0	(for smooth surfaces)
smoothing = 0	(no extra smoothing)
dmin = 0.5	(minimum distance between source points)
segmax = 40	(maximum number of points per segment)
npmin = 150	(minimum number of points for interpolation)

(iv) *v.surf.spline*

A fourth interpolation routine was written based upon the method suggested by Yeoli (1986). It differs from the other three in that 1-dimensional interpolation is used along profiles across the contour lines. Four profiles at 45 degrees to each other were measured intersecting at each interpolated raster cell (Figure 3.9). A 1 dimensional cubic spline was fitted along each profile. The spline function between any two contour lines can be uniquely defined by the distance away from the interpolated cell and the heights of the two adjacent contour intersections with the profile and the second derivative of the function at the next bounding contours (Press *et al*, 1992). In practice, the spline function was calculated for each entire profile across the DEM before calculating its value at intersections with raster cells. The algorithm for spline fitting is based on that by Press *et al*, (1992, pp.113-116).

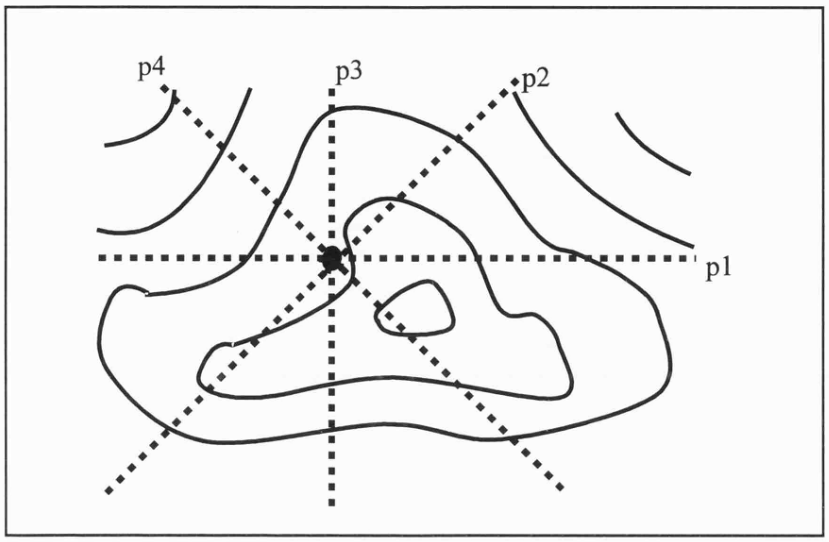


Figure 3.9 - Profiles used for 1-dimensional spline fitting (after Yeoli, 1986).

For each interpolated cell, two values are produced, the interpolated elevation value and a weighting factor that is based on the distance of separation between the interpolated cell and its bounding contours (see Figure 3.9). Thus, each cell is associated with eight values, four interpolated elevations (one for each profile direction) and four interpolation weights. The final elevation is the weighted average of the four interpolations. Additionally an RMSE of the interpolation at each cell can be estimated by measuring the weighted standard deviation of the four profile interpolations. Note that this is an measure of the variability in interpolation and not necessarily a measure of DEM accuracy (see table 3.3 for a comparison of the two).

For each interpolated location,

$$z = \frac{\sum_{i=1}^{i=4} z_i . w_i}{\sum_{i=1}^{i=4} w_i}$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{i=4} (z_i - z)^2 . w_i}{\sum_{i=1}^{i=4} w_i}} \qquad \text{. . . . . (9)}$$

where z is the final interpolated elevation,
z_i is the interpolated profile elevation
w_i is the profile interpolation weight
RMSE is the root mean squared error estimate.

It should be stressed that the RMSE value is only an estimate of data uncertainty and is based on the uncertainty associated with this particular algorithm. Its relationship with the true error (known in this case because we have the original surface *fractal* with which to compare elevations) is investigated in section 3.2.3 below. It also demonstrates how the interpolation can be improved by constraining the spline function.

3.2.2 Visualisation Techniques Available in GIS

The visualisation process is acknowledged to be an effective way of understanding spatial data (eg Hearnshaw and Unwin, 1994) and not least in the examination of spatial data error (Beard *et al*, 1991). This section examines the way in which commonly available raster GIS functionality may be used to visualise DEM uncertainty.

(i) 2D Raster Rendering

By far the most common method of displaying raster data in a GIS is to use a 2-dimensional colour coding where each elevation value is assigned a colour value. With respect to categorical data this is sometimes referred to as *k*-colour mapping (Unwin, 1981), or with respect to elevation, hypsometric mapping. The effect is to give the appearance of thick coloured contours filling the surface, which gives a general impression of topography but reveals little of the high frequency variation over the surface (Figure 3.10). The only exception to this is at distinct colour boundaries where some indication of high frequency variation is given.

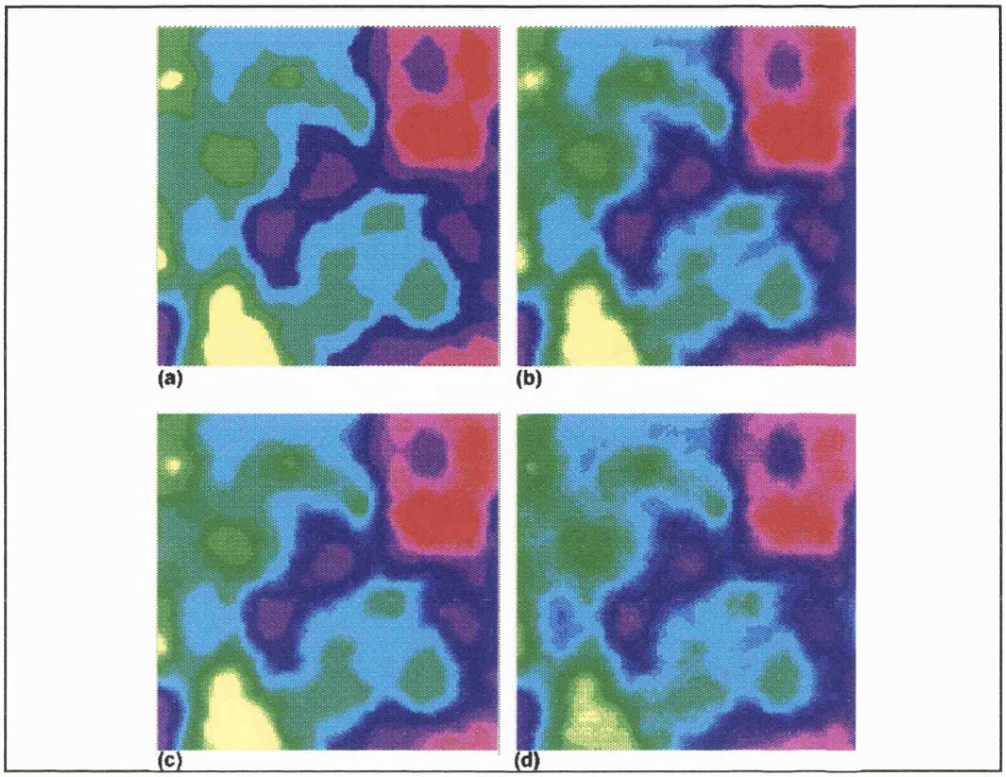


Figure 3.10 - 4 'raster rendered' interpolated surfaces (a) *s.surf.idw*; (b) *r.surf.contour*; (c) *s.surf.tps*; (d) *v.surf.spline*

Figure 3.10 shows the four interpolated surfaces rendered in this manner using GRASS' default colour scheme. Although some differences can be seen between the surfaces, it is difficult to make a valued judgement about the data quality of each, even when compared with the 'true' surface (see Figure 3.8 above). The only form of data error likely to be detected using this method is that of 'blunders' - significant localised deviations in elevation value. Even so, blunders may be hard to detect due to their minimal areal extent.

(ii) Bi-polar difference maps

In cases where two elevation surfaces are available, one of which is known to be of higher accuracy than the other, a 'difference map' may be produced using simple map algebra. One convenient method of visualising difference maps is to render in two dimensions using a bi-polar colour scheme. Two contrasting hues are used for positive and negative differences and value is used to indicate the strength of difference. This

may be achieved conveniently where a colour scheme can be defined by interpolating between colour values.

To create a bi-polar colour scheme for a difference map that varies between -92 and 85:

-100	*black*
-20	*blue*
0	*white*
20	*red*
100	*black*

Table 3.1 - Typical bi-polar linear interpolated colour scheme.

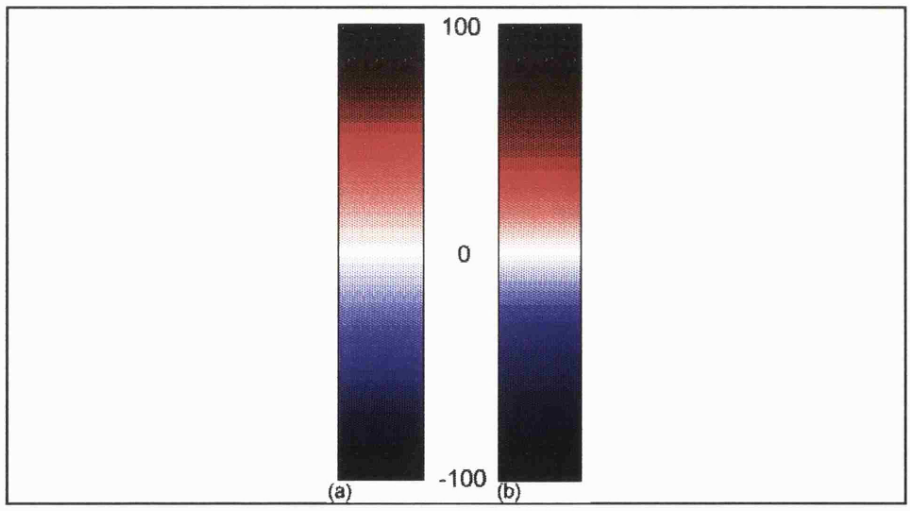


Figure 3.11 - Two bi-polar colour scales. (a) Linear scheme with red/blue set at +/- 50%; (b) Red/blue set to +/- 1 standard deviation of a normal distribution.

If a judgment is required of the balance between positive and negative values, the positive and negative limits of the colour scheme should have equal magnitude. The placement of the hue setting (+/- 20 in Table 3.1) will depend on the nature of the distribution (particularly the standard deviation). Figure 3.11 shows two variants of the colour scheme. Figure 3.11*a* shows a linear colour scale with the two hue settings at

one quarter of the distribution range. Figure 3.11*b* shows a colour scale more appropriate for normally distributed data with the hue settings at +/- one standard deviation. Note that although this colour scheme is bi-polar, both extrema are represented as black. In practice this does not cause any interpretation problems for spatially autocorrelated data where the transitions of colour provide an interpretive context. For poorly or negatively autocorrelated surfaces, more caution should be exercised in using this colour scheme. Figure 3.12 shows the four bi-polar colour maps for each of the four interpolations.Each interpolated surface is compared with the original fractal DEM at the same grid resolution. Note that the colour scheme is identical for all four maps to aid comparison.

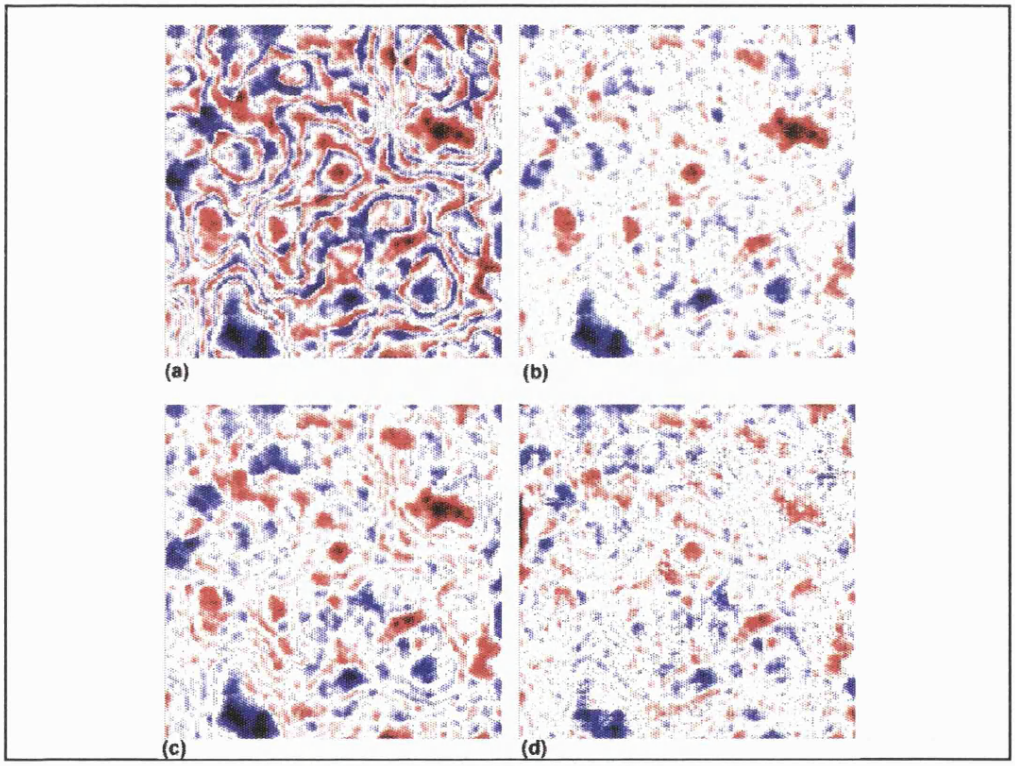


Figure 3.12 - Bi-polar difference surfaces each compared with original fractal surface. (a) *s.surf.idw*; (b) *r.surf.contour*; (c) *s.surf.tps*; (d) *v.surf.spline*

The most immediate impression given by these maps is that inverse distance weighting gives the highest overall difference and contour flood filling appears to result in the smallest overall error. The nature of the accuracy distributions appears to vary with

spline interpolation having the smallest range of difference values. All four difference maps appear to show very similar spatial distributions; variation is mostly confined to changes in magnitude. Although this set of maps is the only one discussed in this section that truly visualises model accuracy, in many cases some of the alternative visualisations discussed below reveal much more about the accuracy of the interpolation process.

(iii) Pseudo 3D Projection

Additional topographic information can be revealed by viewing elevation data obliquely. In order to do this each surface location must be transformed using a (pseudo) 3-dimensional projection.

Given a 3-dimensional viewing location in polar coordinates (λ ,ϕ, r) and surface location of (x,y,z), we can calculate the screen coordinates (X,Y) using homogeneous matrices as follows (Ammeraal, 1986, pp.60-73) :

$$\begin{pmatrix} x_e & y_e & z_e & 1 \end{pmatrix} = \begin{pmatrix} x & y & z & 1 \end{pmatrix} \begin{pmatrix} -\sin\lambda & -\cos\phi\cos\lambda & -\sin\phi\cos\lambda & 0 \\ \cos\lambda & -\cos\phi\sin\lambda & -\sin\phi\sin\lambda & 0 \\ 0 & \sin\phi & -\cos\phi & 0 \\ 0 & 0 & r & 1 \end{pmatrix} \quad \ldots\ldots\ldots \textbf{(10)}$$

where (x_e,y_e,z_e) are the 3-dimensional viewing coordinates.

$$X = d\,\frac{x_e}{z_e}\,, \quad Y = d\,\frac{y_e}{z_e} \quad \ldots\ldots\ldots \textbf{(11)}$$

where d is a scaling coefficient that represents the distance from viewpoint to screen.

Once this projection is applied to surface data, a number of different data types may be displayed (see Figure 3.13). Of the three vector data types displayed using this projection, the most useful for detecting error is the 'fishnet' where orthogonal lines are projected over the surface. The information revealed will depend on the density of these lines, but can often be an improvement over 2-dimensional raster rendering as

elevations are not quantized into colour bands. This form projection is appropriate for detecting blunders that have little areal extent but may vary greatly in elevation from their neighbours.

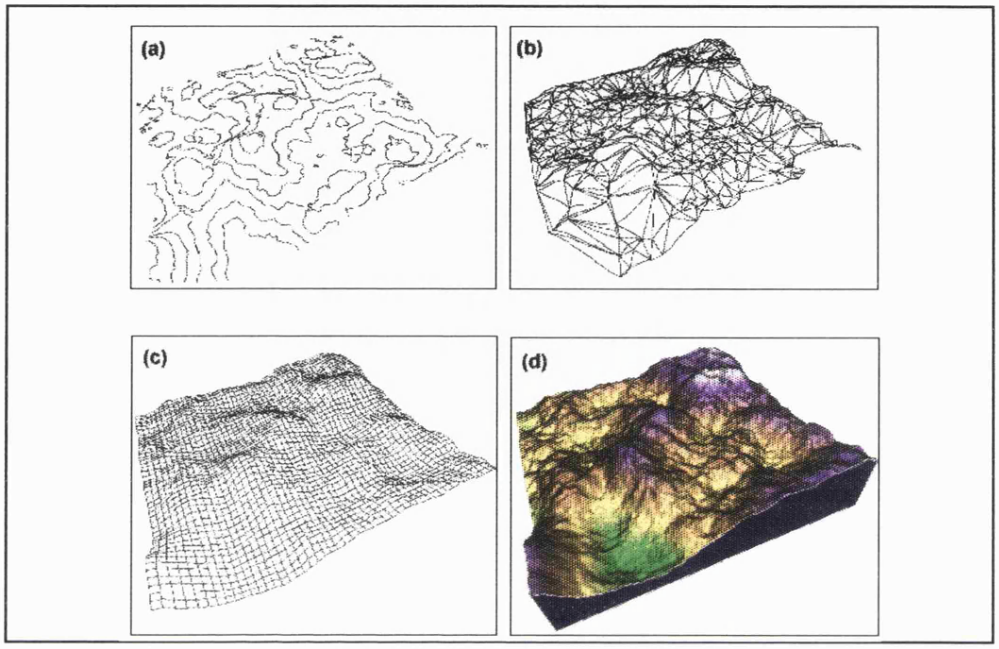


Figure 3.13 - 4 Surface models projected using a perspective view (a) Contours (b) Triangulated Irregular Network (TIN) (c) Vector lattice (d) Bi-variate drape (elevation and shaded relief).

Perhaps the most useful property of the 3-dimensional projection is in its ability to represent two variables. In addition to elevation, a second variable can be 'draped' onto the surface and coloured (Figure 3.13*d*). The remaining part of this section examines forms of raster processing that may be applied to a DEM and (optionally) draped upon its surface. The 'fishnet' is most useful for reinforcing surface form when shaded relief is not sufficient (see for example, Figure 4.8*a*).

(iv) Local Derivatives

The derivatives of a surface which characterise some of its fundamental properties are discussed more fully in section 4.2. They are also very sensitive to the types of internal errors produced by some interpolation routines. One of the commonest applications of

local derivatives is in the calculation of shaded relief (eg Yeoli, 1967; Brassel 1974). These methods use local windows for calculating shading (as opposed to more sophisticated ray-tracing techniques used in some visualisation systems) and as such are sensitive to the characteristically high frequency error patterns.

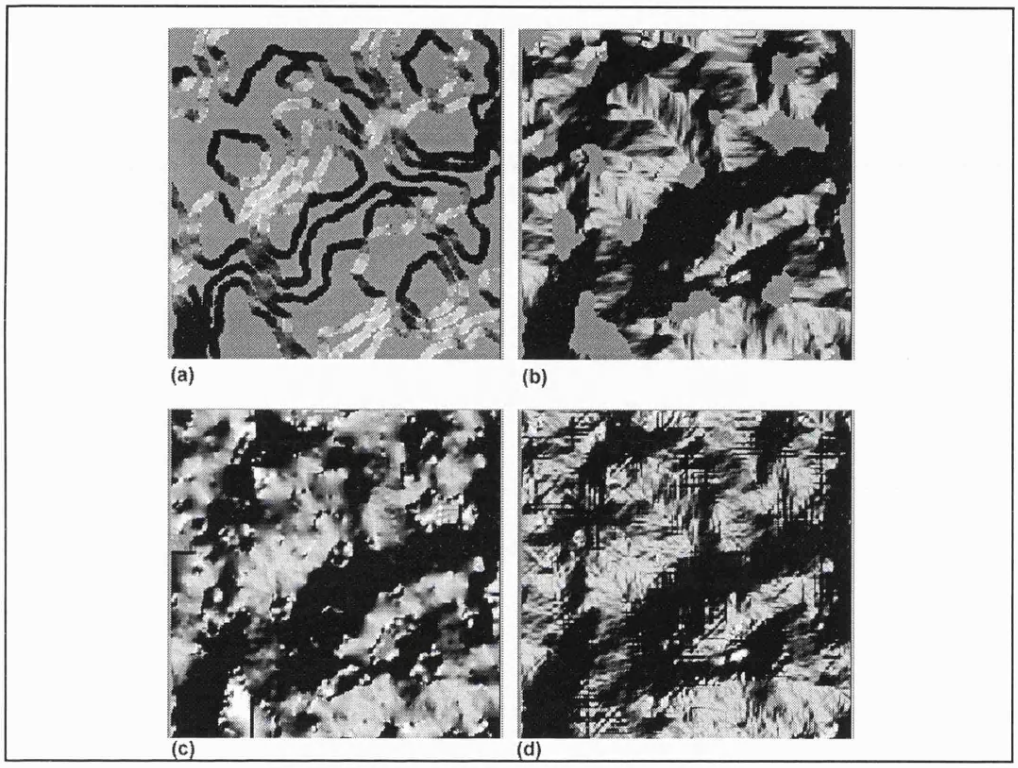


Figure 3.14 - Shaded relief from interpolated surfaces (a) *s.surf.idw*; (b) *r.surf.contour*; (c) *s.surf.tps*; (d) *v.surf.spline*

Figure 3.14 shows the shaded relief calculated for the four interpolated surfaces. In a similar way aspect (the slope azimuth) may be identified and if coloured using a bi-polar colour scheme, can give the appearance of shaded relief (Figure 3.15). Four different interpolation artifacts are revealed by this process. The flattened maxima and minima of the contour flood filling interpolation are immediately obvious in Figures 3.14*b* and 3.15*b*. This clearly indicates a need for the introduction of spot heights at turning points before interpolation. Equally striking are the terracing effects produced by the inverse distance weighting interpolation (Figures 3.14*a* and 3.15*a*). There is evidence of the quadtree segmentation procedure in the artifacts of Figures 3.14*c* and 3.15*c*. These can be seen as occasional striations running north-south and east-west

1/8th, 1/4, and 1/2 way along each edge. They are most apparent towards the edge of the images. Striations that correspond to the profiles taken by the 1-dimensional spline fitting routine can also be seen in Figures 3.14*d* and 3.15*d*.

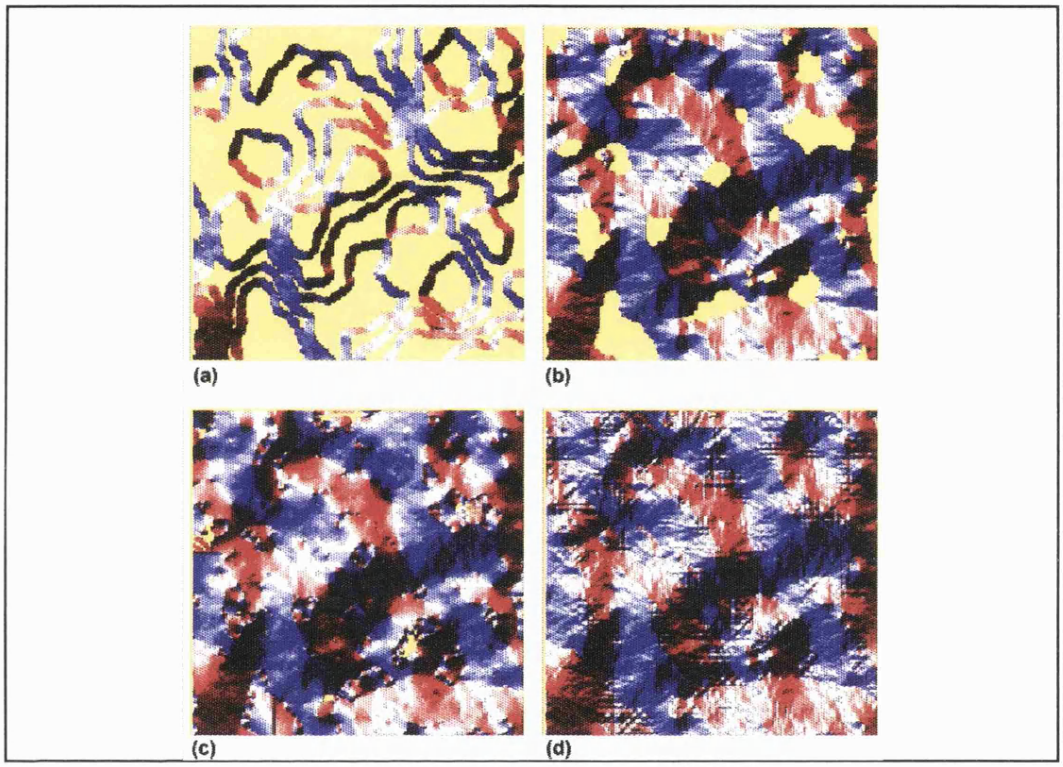


Figure 3.15 - Aspect calculated from interpolated surfaces. (a) *s.surf.idw*; (b) *r.surf.contour*; (c) *s.surf.tps*; (d) *v.surf.spline*

Some caution should be exercised when interpreting shaded relief and aspect maps since they involve identifying an arbitrary light source direction. This can preferentially highlight certain directions of change while hiding orthogonal change. Both these measures also give little or no indication of the magnitude of local change leading to a possible over or underestimate of data quality.

To investigate the magnitude of possible DEM error as well as its spatial distribution, derivatives orthogonal to the plane of the surface must be calculated. Figure 3.16 shows the slope map of the four interpolated surfaces and Figure 3.17, the second vertical derivative, profile convexity. The second derivative appears particularly

sensitive to local interpolation artifacts in that all four methods show the remains of the original contours used for interpolation. This is to be expected since the most significant artifact, contour terracing, is defined as a break in slope (a non-zero second derivative).

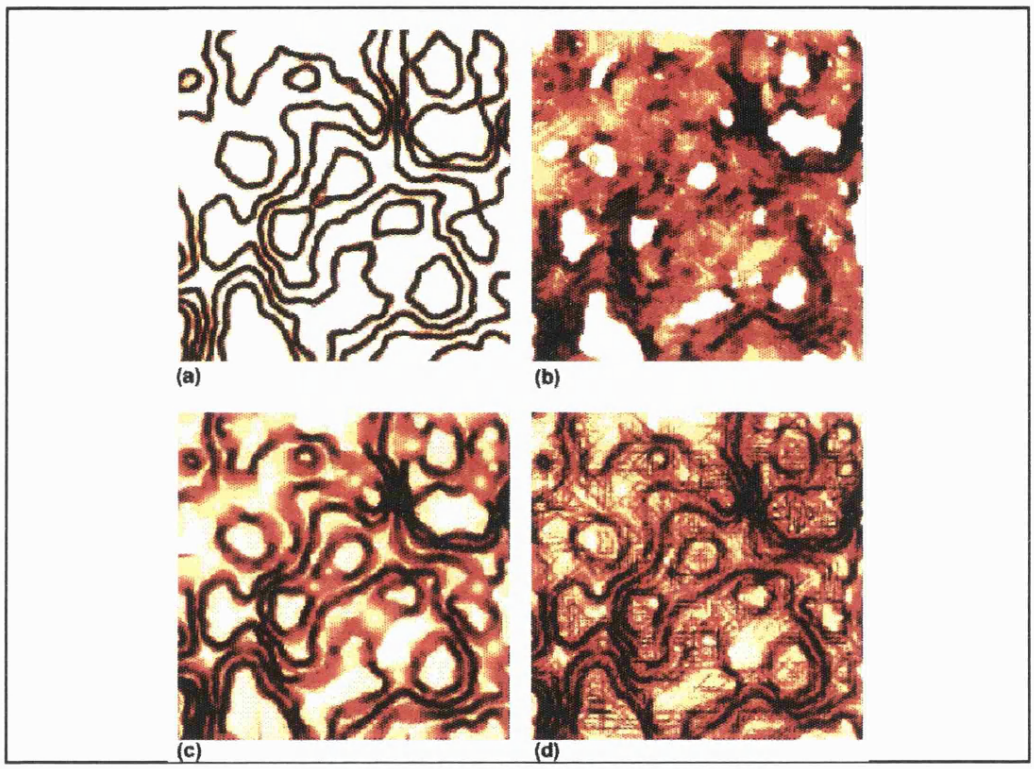


Figure 3.16 - Slope maps of interpolated surfaces (a) *s.surf.idw*; (b) *r.surf.contour*; (c) *s.surf.tps*; (d) *v.surf.spline*

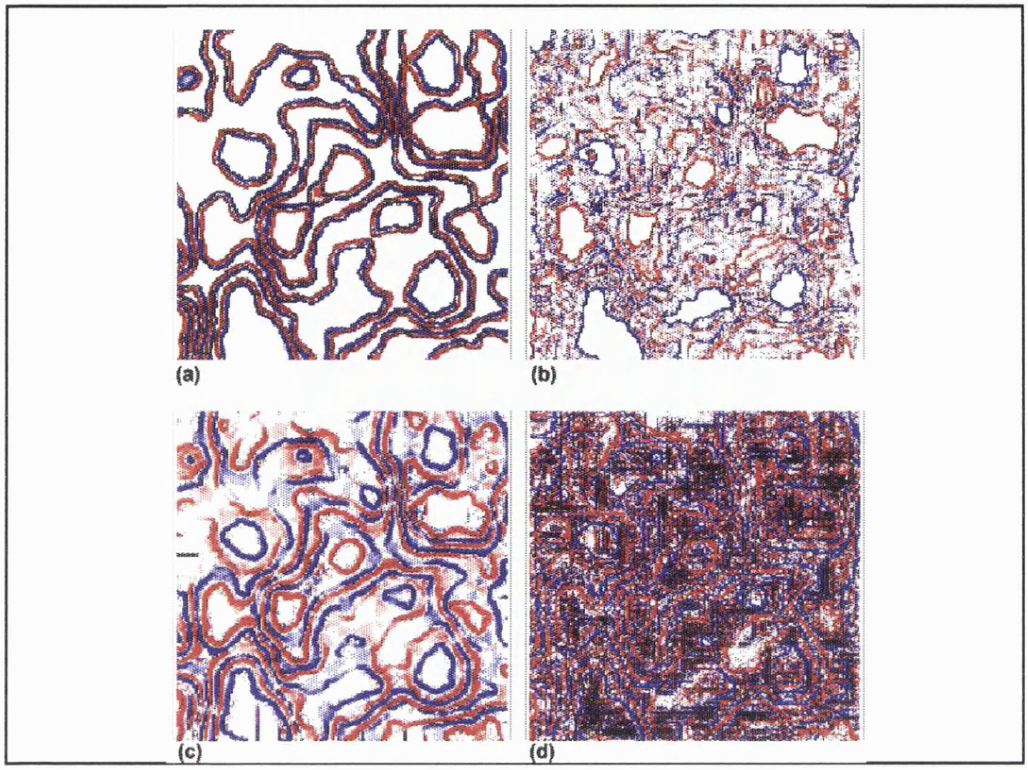


Figure 3.17 - Profile convexity of interpolated surfaces (a) *s.surf.idw*; (b) *r.surf.contour*; (c) *s.surf.tps*; (d) *v.surf.spline*

(v) Local Filters

A number of image processing local filters may be passed over a DEM in order to detect high frequency variation. One of the commonest is the high-pass Laplacian filter specifically designed for local edge-detection (Figure 3.18). It would be expected that this filter might produce a similar pattern to the slope images (Figure 3.16) since the Laplacian convolution is a discrete approximation of the first derivative (Schalkoff, 1989). For the images in Figure 3.18, a Laplacian kernel of 3x3 cells was taken from the original image to give a map of local high frequency change.

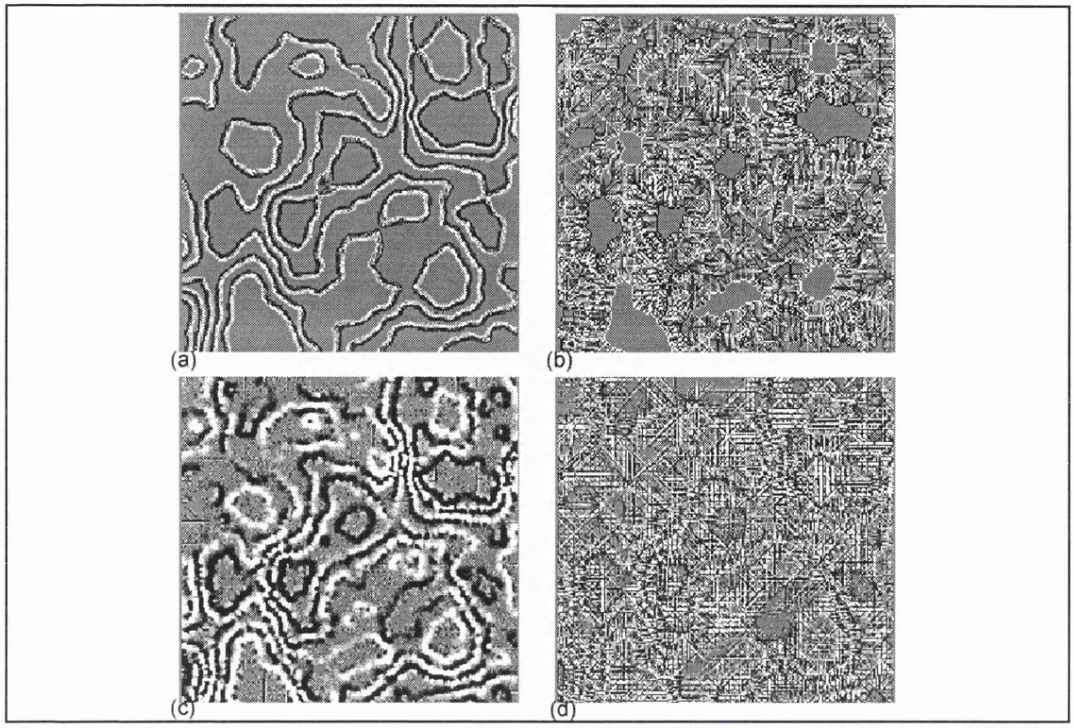


Figure 3.18 - Laplacian filter passed over interpolated surfaces (a) *s.surf.idw*; (b) *r.surf.contour*; (c) *s.surf.tps*; (d) *v.surf.spline*

The Laplacian filter picks out the contours in all four surfaces, making explicit the scale at which artifacts manifest. The finest scale (highest frequency) of contour artifacts appear on the two profile based interpolated surfaces (*r.surf.contour* and *v.surf.spline*). The contour artifacts produced by *s.surf.idw* appear at a lower frequency while *s.surf.tps* produces the lowest frequency (coarsest scale) artifacts. It is also apparent that the Laplacian filter emphasises the diagonal profiles produced by *r.surf.contour* - a pattern not apparent using the other visualisation techniques.

3.2.3 Constrained Spline Fitting - Detecting Algorithm Error Using Visualisation

Examination of the images of the surface interpolated using *v.surf.spline* reveals characteristic artifacts, particularly in the striations along profiles over the surface. This section demonstrates how visualisation of both the interpolated surface and the RMSE results can lead to improvements in the interpolation algorithm. It demonstrates how visualisation can be used not only to give a greater understanding of data, but also of algorithmic process.

(i) The original interpolator

The first implementation of *v.surf.spline* was as described in section 3.2.1 (iv) above. Detailed examination of the difference in elevation between each profile interpolation and the true surface (Figure 3.19) shows marked variation. In many cases differences of over 2 contour intervals are recorded. The cause of some of these larger differences which appear spatially autocorrelated along profiles is demonstrated in Figures 3.19 and 3.20.

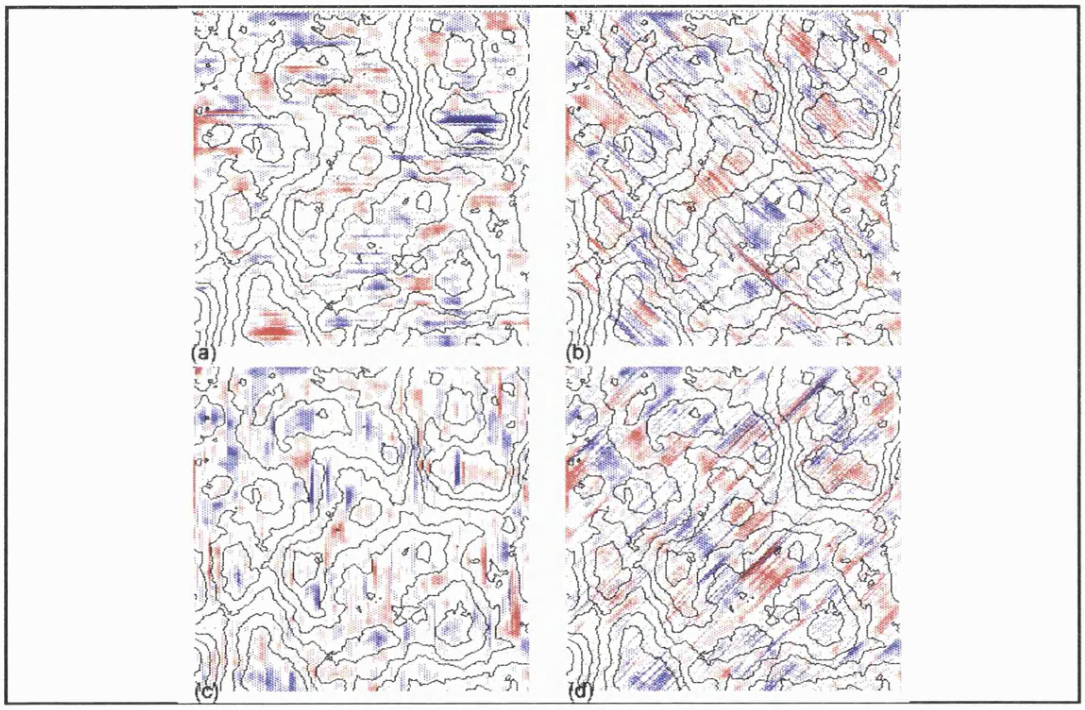


Figure 3.19 - Difference maps showing difference between surfaces interpolated in each of the four profile directions and the original surface *fractal.*

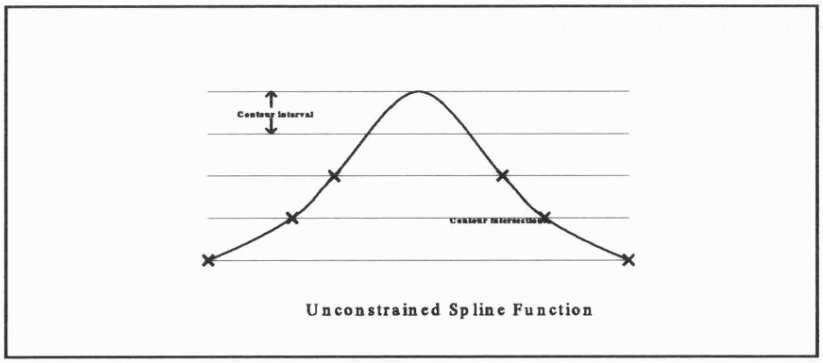


Figure 3.20 - Original profile spline interpolation showing an overshoot of at least 1 contour interval.

Significant overshooting or undershooting of the spline function can occur at local maxima and minima implied by bounding contours. It is possible to identify such over/under shoots because we know that between existing contours no interpolated value should exceed one contour interval (if it did, it would itself be bound by its own contour lines). There are two sets of conditions where we can identify such 'contour crossings' (see Table 3.2).

Illegal contour crossing conditions:
(**lbc** = left bounding contour elevation,
rbc = right bounding contour elevation,
z_i = interpolated elevation, **ci** = vertical contour interval)

---

z_i is illegal if neither of the following conditions are met:

1: **if (lbc != rbc)**
min(lbc,rbc) <= z_i <= max(lbc,rbc)

2: **if (lbc == rbc)**
lbc - ci <= z_i <= lbc + ci

Table 3.2 - Illegal contour crossing conditions for spline function constraint.

It is possible to constrain the interpolation function so that illegal contour crossings are eliminated. While there can be no guarantee that 'true' elevations fall between contours in this way, without reference to an external data source, this constraint, helps to maximise the information provided by the contour model.

(ii) Truncation Constraint

The simplest constraint on interpolation is to truncate any values that fail both conditions in Table 3.2. The constrained value is set to either *min(lbc,rbc)* or *max(lbc,rbc)* if condition *1* is not met. If condition *2* is not met then *zi* is set to either *lbc+ci* or *lbc-ci*. The result is a truncated profile of the type illustrated in Figure 3.21.

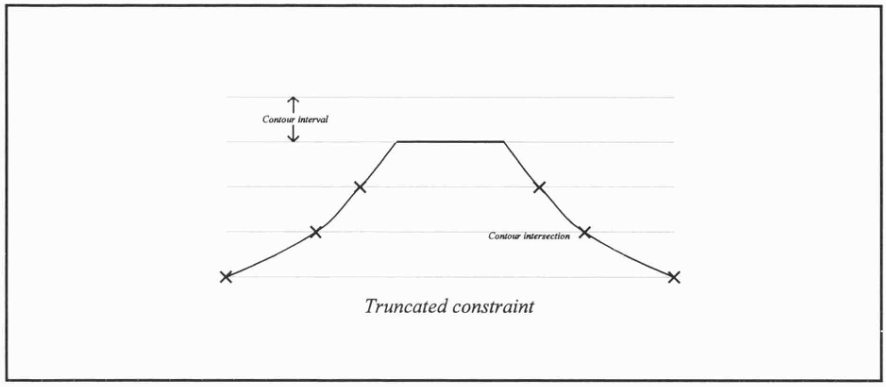


Figure 3.21 Constrained profile using simple truncation.

The constrained profiles have a smaller maximum deviation from the true surface(see Table 3.3) but still retain characteristic striations. Close examination of the RMSE surfaces (Figure 3.22*a*) reveals that the greatest errors appear along diagonal profiles increasing towards orthogonal contour lines.

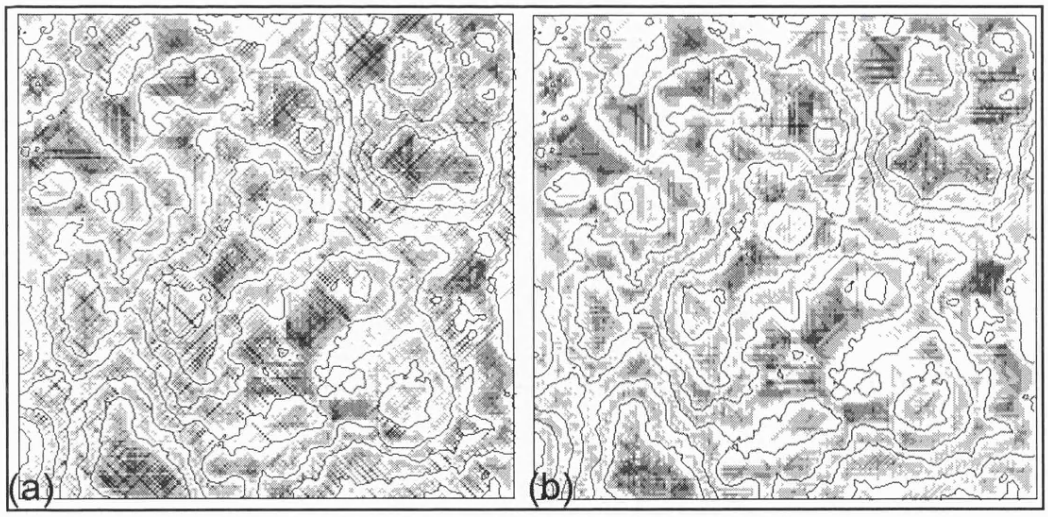


Figure 3.22 RMSE estimated from the 4 profile surfaces for *(a)* queen's-case rasterisation of contours and *(b)* rook's case rasterisation.

Visualisation reveals that the cause of this effect is the rasterisation of the vector contours before interpolation assumes queen's-case (8 way adjacency). When diagonal profiles are taken through the rasterised contour lines, diagonals can sometimes be 'missed' by the searching algorithm. Consequently, when the truncation constraint is applied, elevations are fixed within 1 contour interval of the *detected* adjacent contour not the *actual* bounding contours.

This problem can be solved by forcing rook's-case (4 way) rasterisation of contours before interpolation. The RMSE surface of the rook's-case is shown in Figure 3.22*b*.

(iii) Addnode Constraint

Although effective in reducing spline overshoot and undershoot, the major problem with spline truncation is that it produces characteristically flat topped profiles that are unlikely to reflect the true surface variation. A smoother surface can be produced by adding nodes at points of maximum over/undershoot set at their constrained value. The spline function is then recomputed with the extra node. This process is repeated iteratively until all interpolated points are within the acceptable contour limits. Profiles a produced of the type illustrated in Figures 3.23 and 3.24

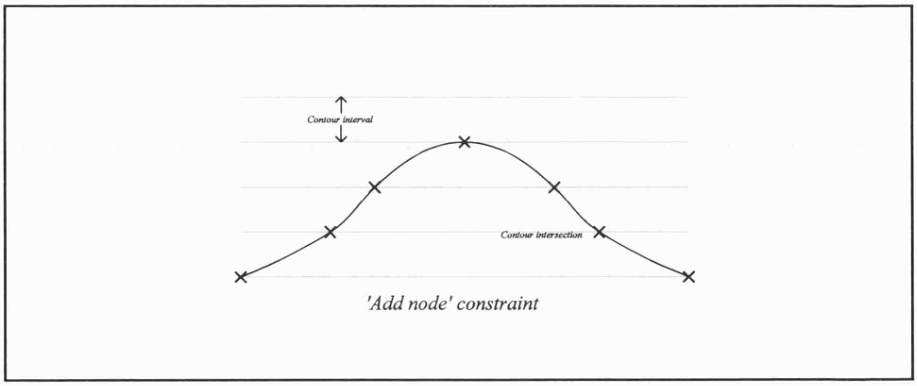


Figure 3.23 Profile constrained by adding node a point of maximum overshoot/undershoot.

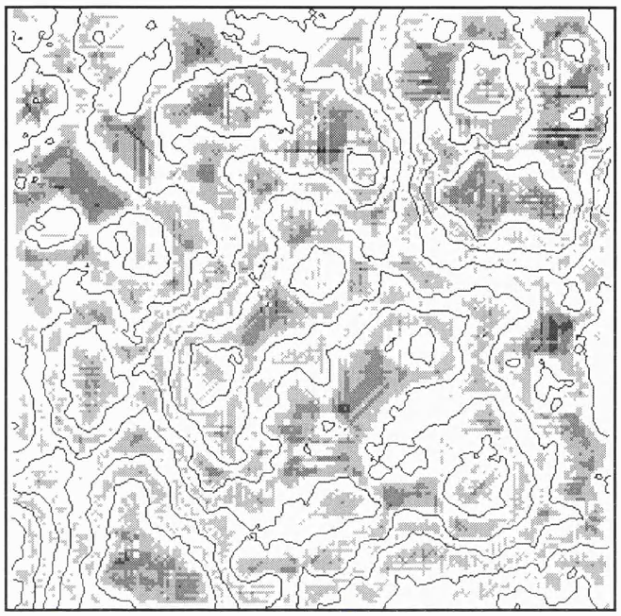


Figure 4.24 - Estimated RMSE of addnode spline interpolation (darker shade indicates higher RMSE)

Finally, a quantitative comparison may be made between all four variations of *v.surf.spline* by examining some of their global statistics and variance with the original surface *fractal*.

Surface	**$Diff_{mean}$**	**$Diff_{Stdev}$**	**$Diff_{min}$**	**$Diff_{max}$**	**RMSE**	**$RMSE_{max}$**
unconstrained (queen's-case)	-0.22	68.7	-381	579	53.3	658
unconstrained (rook's-case)	-1.20	70.3	-354	571	51.0	505
truncated (queen's-case)	2.48	70.1	-381	384	55.4	449
truncated (rook's-case)	-0.55	68.8	-329	309	49.0	261
addnode (queen's-case)	1.15	69.1	-383	327	52.1	395
addnode (rook's-case)	-0.51	68.8	-329	309	48.6	262

Table 3.3 Summary statistics of the six variations of the *v.surf.spline* interpolation.

3.3 Modelling DEM Uncertainty

Discussion has so far been limited to the description of DEM both quantitatively and qualitatively through visualisation. This section considers how the description of uncertainty can lead to effective modelling of that uncertainty. Once an uncertainty model has been built it can be used either to reduce the uncertainty associated with the elevation data, or to examine the effects of that uncertainty on subsequent analysis. This section will consider the former by demonstrating how a deterministic error model can be built by comparing an Ordnance Survey 1:50k DEM with photgrammetrically derived spot heights.

3.3.1 Establishing a Bi-Polar Accuracy Surface

The Ordnance Survey 1:50k DEM covering Snowdon, North Wales (centred around SH260840) was used as the basis for model generation. This high relief area contrasts with the areas most susceptible to spiky hypsometric plots (see Figure 3.5 above) and low relief areas where the integer storage of elevation to the nearest metre introduces terracing effects. In order to assess its accuracy, and in particular the spatial distribution of uncertainty, a second independently derived surface model was required. This was generated from photogrammetric stereo pairs by Ordnance Survey Photogrammetric Services. Sparse vegetation cover and lack of urbanisation ensured accurate stereo matching. Photogrammetric elevations were taken to coincide with the centre of each DEM cell. Generation procedures and statistics are summarised in Table 3.4 below.

Model	**Procedure**	**Planimetric resolution**	**Vertical resolution**	**Planimetric RMSE**	**Vertical RMSE**
SH64 (DEM)	MS contour interpolation	50m	1m	Unknown	Unknown
Photo	Semi-automatic stereo matching	50m	0.01m	1.38m	0.16m

Table 3.4 Two independent Snowdon elevation models

A bi-polar difference map between the two models was calculated using map algebra and is shown in Figure 3.25. Blue indicates the DEM gives higher elevation estimates than the more accurate photogrammetric source, red lower elevation estimates.

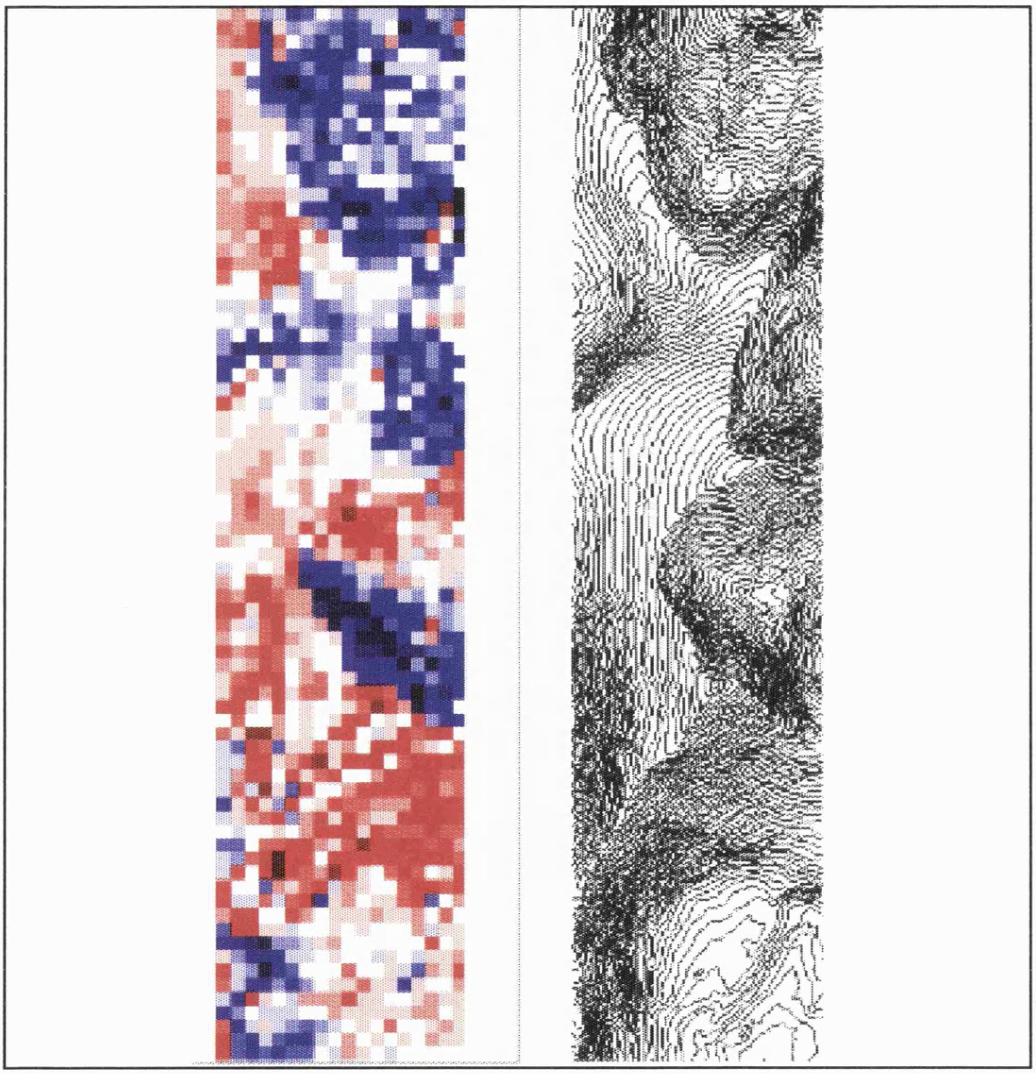


Figure 3.25 - Bi-polar difference map of Snowdon 1:50km DEM and photogrammetrically derived surface. 10m contours shown to provide topographic context.

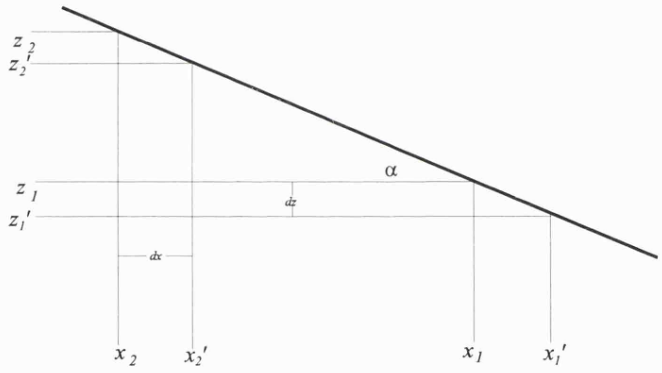
The difference map clearly indicates some form of positive spatial autocorrelation in elevation difference. Comparison with the contour map of the same area suggests this to be related in part to the aspect direction and slope. The cause of this pattern is hypothesised and modelled below.

3.3.2 A Planimetric Offset Error Model

Let us suppose that we suspect that the cause of elevation error may be due to some mis-registration of planimetric coordinates. This hypothesis appears to be supported by the Snowdon bi-polar difference map, and as will be shown later, is in fact the case. We can examine the relationship between planimetric and vertical error by considering firstly a simple 1-dimensional case, then the 2-dimensional DEM case.

(i) The One Dimensional Case

Consider a line of gradient α with respect to some horizontal axis, defined by two points x_1 and x_2 with heights of z_1 and z_2 respectively (Figure 3.26).



Suppose we wish to model the two elevations z_1 and z_2 by measuring two points at locations x_1 and x_2, but due to a systematic planimetric offset of dx we actually measure elevations at x_1' and x_2'. The planimetric displacement dx between true and measured locations will result in a vertical displacement dz,

$$\tan(\alpha)=dz/dx \quad \ldots\ldots\ldots \textbf{(12)}$$

Thus we can model the elevation error if we know the displacement and slope,

$$dz = dx.\tan(\alpha) \quad \ldots\ldots\ldots \textbf{(13)}$$

or we can calculate the planimetric displacement if we know the elevation error and slope,

$$dx = \frac{dz}{\tan(\alpha)} \quad \ldots\ldots\ldots \textbf{(14)}$$

(ii) The 2-dimensional case

The trivial example above serves to illustrate the relationship between slope, vertical displacement and planimetric displacement. For a DEM we must consider the two dimensional equivalent where planimetric offset and slope have two components.

The relationships (12), (13), and (14) will hold when planimetric displacement is in the direction of steepest slope. An offset perpendicular to this will have no effect on the vertical offset dz. Thus the strength of the relationship is determined by the azimuthal angle between the direction of planimetric offset and the steepest slope (aspect).

This relationship is best considered by measuring planimetric offset in polar coordinates:

Let γ be the angle of steepest slope (aspect) with respect to some arbitrary direction

γ_O be the angle of planimetric offset with respect to the same arbitrary direction

O be the magnitude of planimetric offset.

Thus, by combining with (13) above, for a given slope magnitude α

$$dz = O \,.\, \tan(\alpha) \,.\, \cos(\gamma - \gamma_O) \quad \ldots\ldots\ldots \textbf{(15)}$$

This important result allows us to *predict* elevation error if we can measure local slope, aspect, and planimetric displacement. However, it is only valid when (i) all error is due to planimetric displacement; (ii) the second derivative of altitude is approximately zero (ie planar) over the distance defined by O.

3.3.3 Measuring Planimetric Offset

If an accuracy surface can be produced, it is possible to estimate the contribution made by any planimetric offset. Because planimetric offset is only one possible cause of altimetric error and the assumptions (i) and (ii) in 3.3.2 above may not always be true, it is only possible to *estimate* the effect of planimetric offset. This may be done by fitting the offset model through sample points using least squares regression fitting.

Let $y = dz / \tan(\alpha)$,

$a = O.\cos(\gamma_O)$

$b = O.\sin(\gamma_O)$

where α, dz, O, and γ_O are defined as above.

Therefore,

$$\gamma_O = \arctan(b/a) \qquad \ldots\ldots\ldots \textbf{(16)}$$

$$O = \sqrt{a^2+b^2} \qquad \ldots\ldots\ldots \textbf{(17)}$$

From *(15)* above, we get:

$$\begin{aligned} y &= O.\cos(\gamma-\gamma_O) \\ &= O.\cos(\gamma).\cos(\gamma_O) + O.\sin(\gamma).\sin(\gamma_O) \end{aligned} \qquad \ldots\ldots\ldots \textbf{(18)}$$

Substituting a and b from above,

$$y = a.\cos(\gamma) + b.\sin(\gamma) \qquad \ldots\ldots\ldots \textbf{(19)}$$

Let χ^2 be the total squared deviation between the measured and modelled value of y.

$$\chi^2 = \sum [y - a.\cos(\gamma) - b.\sin(\gamma)]^2 \quad \ldots\ldots\ldots \textbf{(20)}$$

To minimise this deviation, the derivatives with respect to a and b will be zero.

$$\frac{d\chi^2}{da} = 0 = 2\sum [\cos(\gamma).(y - a.\cos(\gamma) - b.\sin(\gamma)]$$
$$\frac{d\chi^2}{db} = 0 = 2\sum [\sin(\gamma).(y - a.\cos(\gamma) - b.\sin(\gamma)] \quad \ldots\ldots\ldots \textbf{(21)}$$

Therefore,

$$\sum [y.\cos(\gamma)] = a\sum \cos^2(\gamma) + b\sum [\sin(\gamma).\cos(\gamma)]$$
$$\sum [y.\sin(\gamma)] = a\sum [\sin(\gamma).\cos(\gamma)] + b\sum \sin^2(\gamma) \quad \ldots\ldots\ldots \textbf{(22)}$$

Let the following measurable terms be defined,

$$YS = \sum [y.\sin(\gamma)]$$
$$YC = \sum [y.\cos(\gamma)]$$
$$SC = \sum [\sin(\gamma).\cos(\gamma)] \quad \ldots\ldots\ldots \textbf{(23)}$$
$$C^2 = \sum \cos^2(\gamma)$$
$$S^2 = \sum \sin^2(\gamma)$$

Substituting into *(22)*,

$$YC = a.C^2 + b.SC$$
$$YS = a.SC + b.S^2 \quad \ldots\ldots\ldots \textbf{(24)}$$

Multiplying by SC and C^2,

$$YC.SC = a.C^2.SC + b.(SC)^2$$
$$YS.C^2 = a.SC.C^2 + b.S^2.C^2 \quad \ldots\ldots\ldots \textbf{(25)}$$

$$b = \frac{YS.C^2 - YC.SC}{S^2.C^2 - (SC)^2} \quad \ldots\ldots\ldots \textbf{(26)}$$

Multiplying by S^2 and SC,

$$YC.S^2 = a.C^2.S^2 + b.SC.S^2$$
$$YS.SC = a.(SC)^2 + b.S^2.SC \quad \ldots\ldots\ldots \textbf{(27)}$$

$$a = \frac{YS.SC - YC.S^2}{S^2.C^2 - (SC)^2} \quad \ldots\ldots\ldots \textbf{(28)}$$

Substituting back into *(16)* and *(17)*, we can express the two parameters in terms of measurable quantities of a DEM.

$$\gamma_O = \arctan\left(\frac{YS.C^2 - YC.SC}{YC.S^2 - YS.SC}\right) \quad \ldots\ldots\ldots \textbf{(29)}$$

$$O = \frac{\sqrt{(YC.S^2 - YS.SC)^2 + (YS.C^2 - YC.SC)^2}}{(SC)^2 - C^2.S^2} \quad \ldots\ldots\ldots \textbf{(30)}$$

Since (29) and (30) are expressed entirely in terms of slope, aspect and error, the modelled planimetric offset may be found using map algebra (see Appendix).

3.3.4 Testing the Model

In order to test the effectiveness of the planimetric offset estimation, two mathematical surfaces were created, one with a known planimetric offset from the other:

$$\begin{aligned} z_1 &= 5000-[(x-50)^2 + (y-50)^2] \\ z_2 &= 5000-[(x-50.6)^2 + (y-50.25)^2] \end{aligned} \quad \ldots\ldots\ldots \textbf{(31)}$$

The two simple quadratic surfaces Z1 and Z2 are identical except that Z2 is planimetricly offset 0.6 units towards the x-axis and 0.25 towards the y-axis. Thus,

$$\begin{aligned} O &= \sqrt{0.6^2 + 0.25^2} = 0.65 \\ \gamma_O &= \arctan(0.25/0.6) = 202.62^o \textit{ anticlockwise of east} \end{aligned} \quad \ldots\ldots\ldots \textbf{(32)}$$

The two surfaces were created using *r.xy* - a program to create two rasters containing the x and y coordinates of each cell (see Appendix), and map algebra (*r.mapcalc*) to create the quadratic functions. Because GRASS stores all rasters as integers, *z* values were scaled by 10 and *(x,y)* coordinates by 1000. The two parameters were found using least squares to be

O = *649 (scaled by 1000)*

γ_O = *203 degrees anticlockwise of east*

The least squares fitting of the two offset parameters can be visualised as a regression plot where aspect direction is plotted against the vertical offset component *dz/tan(α)*. From *(15)* above, the best-fit regression line will be a cosine curve with magnitude O and phase offset of γ_O. The largest residuals occur towards the maxima and minima of the curve. These points correspond to the lowest slopes where it is difficult to ascertain the slope direction reliably.

An alternative visualisation of the relationship between slope, aspect and elevation error was adopted by Wood (1993) where, aspect and *dz/tan(α)* are represented as polar coordinates with each cell corresponding to a displacement vector. The resultant vector gives an estimate of the planimetric offset.

3.3.5 The Snowdon Error Model

The results of applying the regression model to the Snowdon DEMs (SH64 and photogrammetric rasterised spot heights) indicated,

O = *15m*

γ_O = *241 degrees anticlockwise of east*

The relationship is visualised in Figure 3.27. A planimetric offset of 15m at the 50m resolution indicates that for slopes of 45 degrees, elevation is systematically displaced by up to 15m. This is considerably higher than the reported global RMSE figure of 2-3m (Ordnance Survey, 1992). The systematic nature of the displacement means that the vertical error can be corrected by translating all cells 15m 61 degrees anticlockwise of east, and recalculating elevation assuming a planar displacement between the two measures. The new difference map is shown in Figure 3.28, and the effect on global RMSE is to reduce it by approximately 3m.

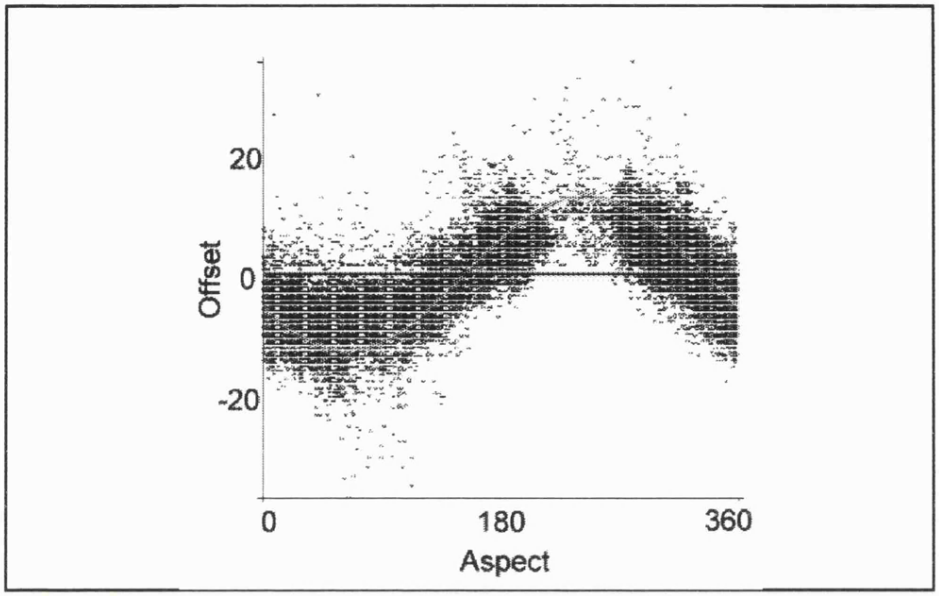


Figure 3.27 - Scatterplot of the aspect - vertical deviation relationship for the Snowdon DEM

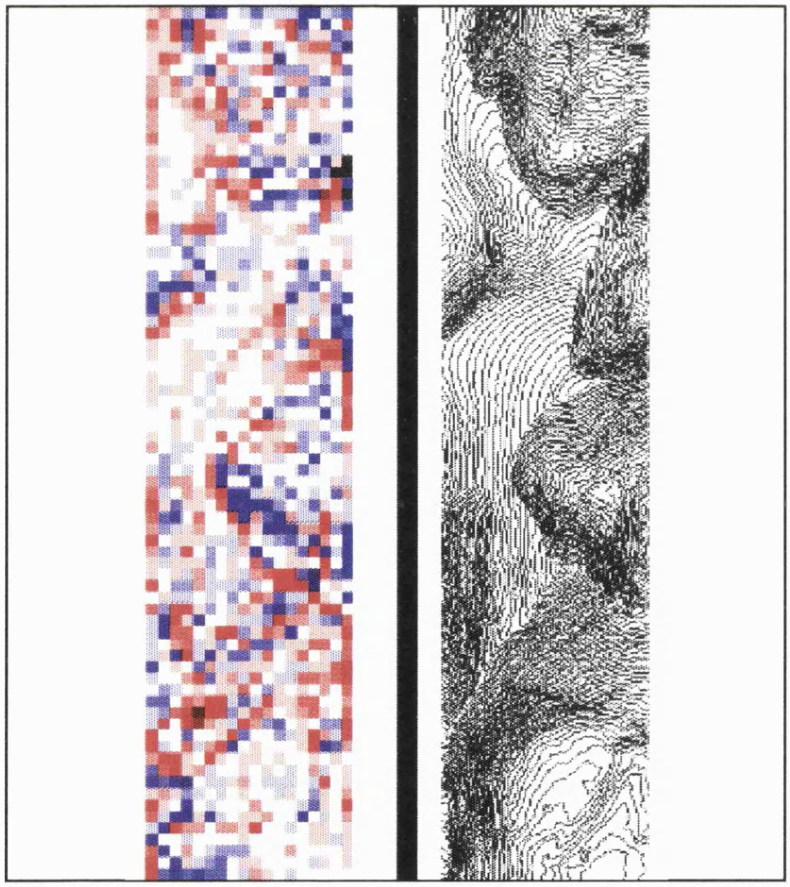


Figure 3.28 - Bi-polar difference map for modelled surface after planimetric offset (identical colour scheme to Fig. 3.25).

It must be borne in mind that the procedure outlined above is only applicable where slopes are sufficiently steep. In areas of little, or even moderate slope, a planimetric displacement has little effect on the vertical offset of elevation values. Additionally, as the test quadratic demonstrated, low slope values can give erroneous results due to a very small denominator in $dz/\tan(\alpha)$ and an unreliable estimate of aspect direction. Figure 3.28 demonstrates that there are still autocorrelated patches of higher accuracy loss suggesting some refinement of the model is possible to reduce error further. One of the causes of lower accuracy along sharp breaks in slope (especially ridges and channels) is that the assumption of planar slope (3.3.2) is not valid for these features. A more sophisticated displacement model based on quadratic surfaces may overcome this.

3.4 Summary of DEM Uncertainty Factors

This chapter has demonstrated that visualisation is an important part of DEM quality assessment. By visualising DEMs and their derivatives, it has been possible to identify a number of causes of model uncertainty. These are summarised in the tables below. Excluded from the discussion have been photogrammetric and surveying based sources of error (eg Firth effect), since it is elevation *model* error that is of primary interest.

1.	**INTEGER ROUNDING**
PROBLEM	Coastal and flat areas represented as `terraces'; slope calculation gives flat regions alternating with narrow bands of steeper slopes.
EXPLANATION	Elevation is stored as integers in metre units. Rounding of true elevation causes a classification into 1m bands.
DIAGNOSTIC	*Visualisation:* Shaded relief map, slope map, concavity map, edge enhancement filter. *Database:* Identify flat coastal regions (eg. elev <5m), check that elevation units are integer metres.
SOLUTION	*Data provider:* Scale z values by 10, or 100, *before* interpolation (scope for storing vertical scale factor in NTF). *Data User:* If stored as integers in GIS, scale by 10 or 100. Pass mean filter over surface. May have to do this iteratively. This is only really appropriate where there is no high frequency variation (otherwise meaningful data are removed).

2.	**INTERPOLATION TERRACING**
PROBLEM	DEM appears to have regular `terraces' over surface; slope calculation reveals flatter regions alternating with narrow bands of steeper slopes.
EXPLANATION	DEM was interpolated from contour data by an interpolation method that failed to interpret the contour data correctly. Contours may have been treated as a collection of independent point samples, or crenellations not correctly identified.
DIAGNOSTIC	*Visualisation:* Shaded relief map, slope map, concavity map, edge enhancement filter, frequency histogram *Database:* Calculate `hammock index' from frequency histogram data.
SOLUTION	Reinterpolate data adding formlines at all sharp contour crenellations. Ensure interpolation algorithm treats contours as lines rather than a collection of points.

3.	**TERRAIN FLATTENING**
PROBLEM	Local peaks not modelled at their correct height - appear to be flattened. Ridges and valleys modelled by the DEM not as `sharp' or `deep' as expected.
EXPLANATION	Insufficient terrain data at topographic `very important points' (VIPs) results in generalisation of these features.
DIAGNOSTIC	*Visualisation:* Shaded relief map - look for unnaturally flattened peaks. Create bi-polar difference surface by comparing with known elevation values and interpolating surface. Look for topographic association by overlaying contours or shaded relief.
SOLUTION	Ensure 5m contour VI is used if terrain is not too steep. Use all available spot height data for topographic VIPs. Automatically detect VIPs during interpolation process and retain.

4.	**PLANIMETRIC OFFSET**
PROBLEM	Error appears to be related to slope direction and steepness - one side of topographic features having overestimated elevation, the other being underestimated.
EXPLANATION	Source data for interpolation have been planimetrically offset.
DIAGNOSTIC	Requires reference to an independent data source. Produce accuracy surface of the two independent models, divide the vertical deviation by the tan of slope for the surface and look for topographic association.
SOLUTION	*Data provider:* Check planimetric accuracy of source data at each stage of data lineage; check transformation to raster based projection; transform data where necessary. *Data user:* Using accuracy surface, regress the offset function, estimate offset magnitude and direction, create model surface, correct original.

Chapter Four - The Parameterisation of Geomorphological Surfaces.

4.1 Aims of Parameterisation.

In common with one of the objectives of science, the description and modelling of surface form should be sufficiently precise, exhaustive, universal and orthogonal. Precision describes the degree to which a description is sensitive to variation. To describe a landscape verbally as 'rolling' is imprecise since it could be applied to a wide variety of landscapes with differing characteristics. A terrain description is exhaustive if it describes all aspects of the surface form. To describe a landscape as 'hilly' says nothing about the form of the surface not defined as hills. Universality relates to the appropriateness of a description in many contexts. To describe a part of a landscape as 'high' clearly depends on a local context with which to compare elevation. Finally, a useful set of terrain descriptors should be orthogonal in that as a set, there is no repetition or redundancy in description. This can be interpreted in a statistical sense (such as the use of factor analysis by Evans, 1984), a geometrical sense (such as profile and plan convexity described below, or the determination of eigenvalues (Gould, 1967)), or simply in a verbal sense (for example 'rough' and 'high' can be regarded as orthogonal descriptions).

The form of description investigated in this chapter is the *parameterisation* of a surface model. In this context, parameterisation has been defined as "the numerical description of continuous surface form" (Pike, 1993). More geomorphologically, it has been described as "a set of measurements that describe topographic form well enough to distinguish topographically disparate landscapes" (Pike, 1988). While both definitions hint at the aims of parameterisation, the measures described in this chapter are more closely associated with the statistical and geometrical definition of a parameter as "*a line or quantity which serves to determine a point, line, figure, or quantity in a class of such things*" (Chambers, 1990). Here, the identification of a unique and defining property of a surface model, is emphasised. Together surface

parameters should be as few as possible, but convey maximum information and be as widely applicable as possible.

For effective geomorphological parameterisation, two additional criteria are required. Firstly, terrain parameters should be sensitive to geomorphological *process* as well as form. For example, altitude, slope and surface curvature (0, 1st and 2nd derivatives) are all affected by or influence geomorphological process. It is difficult to justify measuring the third derivative as it is not clear how this would be related to surface process. Secondly, a complete surface parameterisation must include reference to scale-based characteristics. The effects due to the scale of sampling and the way in which the surface model is stored should be separated from true surface scale-dependencies as far as possible. More specifically, this chapter will consider a method of surface parameterisation that is not constrained by the grid resolution of the DEM.

4.1.1 General and Specific Geomorphometry

The distinction by Evans (1972) of the measurement of surface form into *specific* and *general* geomorphology, provides a framework for classifying morphometric parameterisation. The dichotomy itself is perhaps unduly polarised as many forms of measurement fall somewhere between the two extremes. We can consider the form of measurement on a continuum (Figure 4.1). Progression along the continuum involves measuring a generally increasing amount of spatial information.

The position along the continuum determines the balance between the some of the aims of parameterisation described in section 4.1. Towards the 'general' end, parameterisation is relatively universal, but lacking in precision. Specific geomorphometry is precise, but can only be applied in a limited context. The characterisation methods described in this and the following chapter include examples of morphometric parameter, morphometric feature and geomorphometric feature identification.

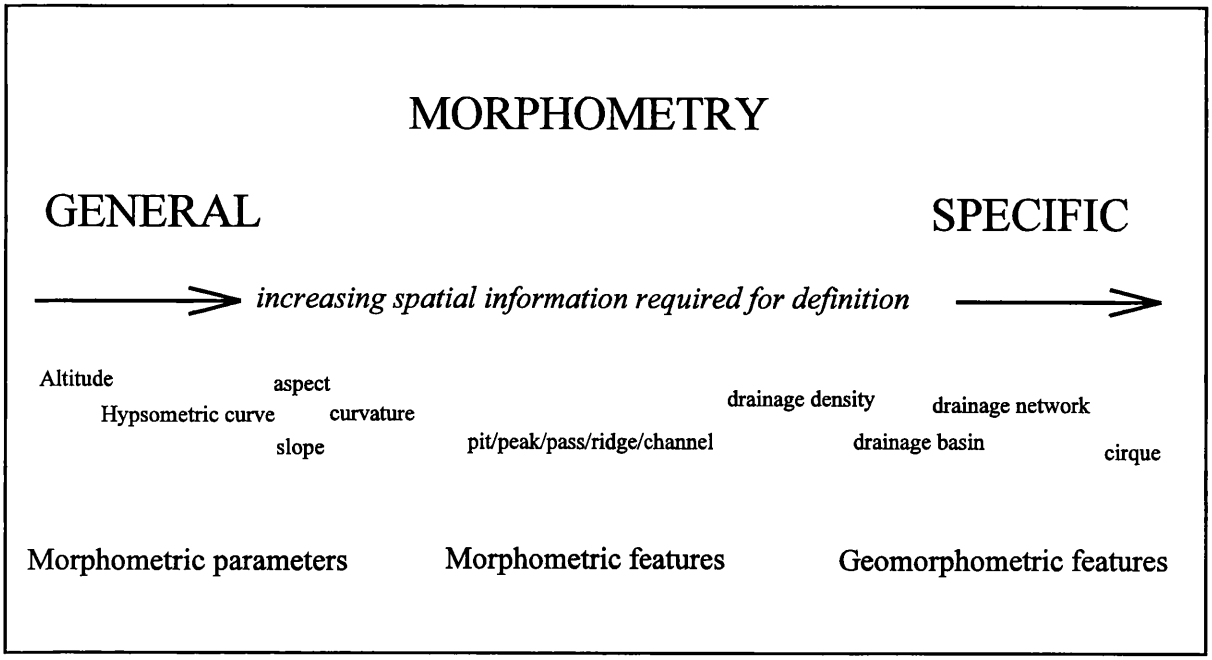


Figure 4.1 A classification of morphometric characterisation

4.2 Morphometric Parameterisation.

Evans (1979) considers five terrain parameters that may be defined for any two dimensional continuous surface.

elevation
slope, aspect
profile convexity, plan convexity

These correspond to groups of 0, 1st and 2nd order differentials, where the 1st and 2nd order functions have components in the XY and orthogonal planes. Whilst higher derivatives may also be extracted, there is no evidence that these have any geomorphological meaning. Additionally, higher order derivatives must be based on surface patches modelled by higher order polynomials if any component is to be extracted. The higher the order of the polynomial, the greater the number of points required to uniquely identify the necessary coefficients. Since it is only the property of the surface at the central point of each local patch that is required, the effect of generalisation increases with the order of polynomial.

Evans goes on to approximate the surface using a bivariate quadratic function in the form:

$$z=ax^2+by^2+cxy+dx+ey+f \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (1)$$

This can be written in the form of the general conic:

$$ax^2+2hxy+by^2+2jx+2ky+m=0 \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2)$$

where $h=c/2$, $j=d/2$, $k=e/2$, and $m=f-z$.

Instances of the general conic fall into one of three types, depending on the values of the coefficients a,b and h (Stephenson, 1973, p.463) *:*

	$ab-h^2>0$		*elliptic*
if,	$ab-h^2=0$	*conic is*	*parabolic*
	$ab-h^2<0$		*hyperbolic*

These general forms correspond to the morphometric feature types, and can be used as part of the feature identification process (see Figure 4.2 and Chapter 5). Contour lines through each of these surfaces describe conic sections and can be seen from Figure 4.2 to be elliptic for pits and peaks, parabolic for channels and ridges, and hyperbolic for passes. For the purposes of general geomorphometry however, terrain parameters can be defined by considering the partial differential equations of the general quadratic form.

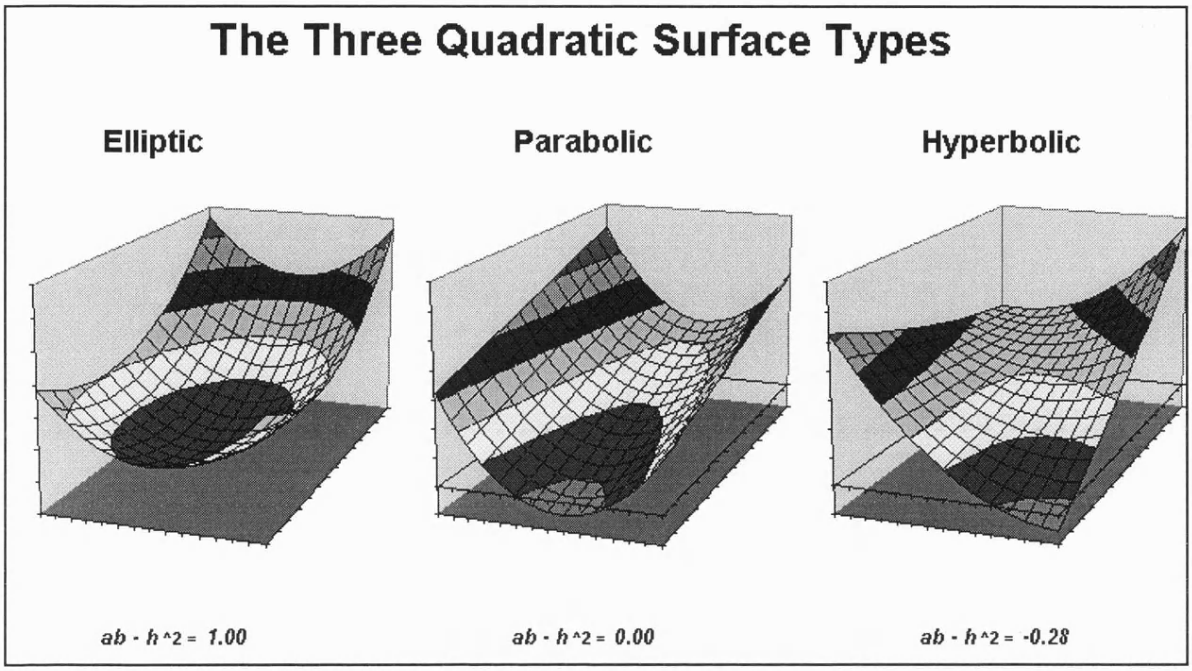


Figure 4.2 - A classification of quadratic surface types.

4.2.1 Slope and Aspect.

The rate of change of elevation in both the x and y directions can be used to identify the direction and magnitude of steepest gradient. These two parameters can be found by taking the partial first order derivatives of *(1)* above, with respect to x and y. Slope (magnitude) can be found by combining the two component partial derivatives:

$$\frac{dz}{dxy} = \sqrt{\left(\frac{\delta z}{\delta x}\right)^2 + \left(\frac{\delta z}{\delta y}\right)^2} \qquad \text{. . . . . (3)}$$

The partial derivatives for x and y are given as

$$\frac{\delta z}{\delta x} = 2ax + cy + d \qquad \frac{\delta z}{\delta y} = 2by + cx + e \qquad \text{. . . . . (4)}$$

Since we are only interested in the slope at the central point of the quadratic surface, by adopting a local coordinate system with the origin located at the point of interest, we can combine *(3)* and *(4)* where $x = y = 0$ *giving*,

$$\frac{dz}{dxy} = \sqrt{d^2 + e^2} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (5)$$

This slope value is more usually represented in degrees, giving:

$$slope = \arctan(\sqrt{d^2 + e^2}) \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (6)$$

This definition is consistent with that reported in the literature (eg Evans 1979,1980; Zevenbergen and Thorne, 1987; Skidmore, 1989). Likewise, aspect is simply the polar angle described by the two orthogonal partial derivatives:

$$aspect = \arctan(\frac{e}{d}) \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (7)$$

4.2.2 Surface Curvature.

If we wish to create a single measure of the second order derivatives we must derive that measure for an intersecting plane so as to reduce the expression to an ordinary differential one. Thus we have several choices depending on the orientation of this intersecting plane. The plane itself can be defined uniquely by two vectors.

profc or profile convexity (intersecting with the plane of the *Z* axis and aspect direction);

planc or plan convexity (intersecting with the *XY* plane);

longc or longitudinal curvature (intersecting with the plane of the slope normal and aspect direction)

crosc or cross-sectional curvature (intersecting with the plane of the slope normal and perpendicular aspect direction);

maxic or maximum curvature (in any plane);

minic or minimum curvature (in any plane);

meanc or mean curvature (in any plane).

The form of curvature that is most appropriate will depend partly on the nature of the surface patch being modelled. Computational and interpretive simplicity may dictate a single measure for an entire DEM. Most commonly, this would be either mean curvature (eg Mitasova, 1994), profile or plan convexity (eg. Evans, 1979). For geomorphological analysis it would be useful to know curvature along, for example, channel cross-sections and longitudinal profiles. The derivations of some of these measures are discussed below.

(i) Profile and Plan Convexity

Profile and plan convexity are useful in that they separate curvature out into two orthogonal components where the effects of gravitational process are either maximised (profile convexity) or minimised (plan convexity). Evans (1979) defines profile and plan convexity measures (of dimension $[L^{-1}]$) as follows,

$$profc = \frac{-200(ad^2+be^2+cde)}{(e^2+d^2)(1+d^2+e^2)^{1.5}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(8)}$$

$$planc = \frac{200(bd^2+ae^2-cde)}{(e^2+d^2)^{1.5}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(9)}$$

Because these two measures involve the calculation of the slope vector, they remain undefined for quadratic patches with zero gradient (ie. the planar components *d* and *e* are both zero). In such cases, alternative measures need to be substituted. In all such cases, there is no component to plan convexity, any curvature being entirely orthogonal to the *xy* plane. Evans (1979) suggests two measures of minimum and maximum profile convexity,

$$profc_{MAX} = -a-b + \sqrt{(a-b)^2+c^2} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(10)}$$

$$profc_{MIN} = -a-b - \sqrt{(a-b)^2+c^2} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(11)}$$

For the elliptic case, both values will have the same sign; for the parabolic case, one of the values will be zero; and for the hyperbolic case, the two curvatures will be in opposite directions.

(ii) Longitudinal and Cross-sectional convexity.

To calculate the mean convexity in the direction of slope, we can reduce the problem to an ordinary differential one by finding the intersection between the quadratic surface and the plane in which the slope normal and aspect direction both lie,

Let s be a vector whose magnitude is proportional to the slope at the centre of the quadratic ($x=y=0$) and s' be orthogonal to this in the same plane as two axes X and Y (see Figure 5.3). Let alpha be proportional to the aspect direction. We can consider the distance along the X and Y axes (which correspond to the axes of the DEM, but with a local origin centred around a DEM cell) in terms of s and alpha,

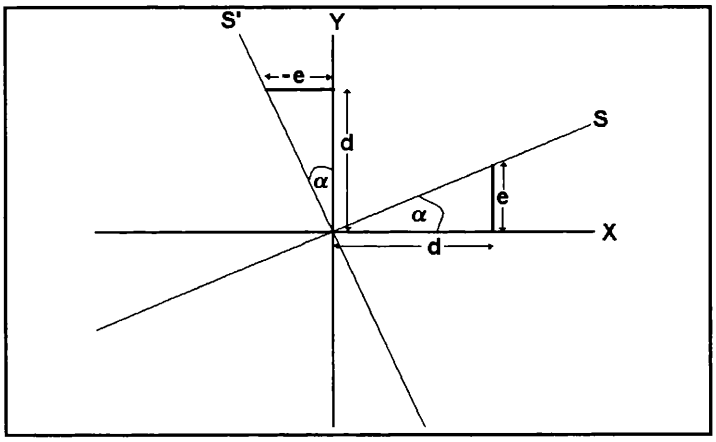


Figure 4.3 - Slope and aspect vectors

$$x=s.\cos(\alpha), \quad y=s.\sin(\alpha) \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \textbf{(12)}$$

From similar triangles,

$$\cos(\alpha)=\frac{d}{\sqrt{d^2+e^2}}, \quad \sin(\alpha)=\frac{e}{\sqrt{d^2+e^2}} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \textbf{(13)}$$

By combining ***(12)*** and ***(13)*** and substituting into ***(1)***, we get,

$$z=a\frac{s^2d^2}{d^2+e^2}+b\frac{s^2e^2}{d^2+e^2}+c\frac{s^2ed}{d^2+e^2}+d\frac{sd}{\sqrt{d^2+e^2}}+e\frac{se}{\sqrt{d^2+e^2}}+f \quad \ldots\ldots\ldots\ldots \textbf{(14)}$$

Which simplifies to,

$$z=(\frac{ad^2+be^2+cde}{d^2+e^2})s^2+(\sqrt{d^2+e^2})s+f \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \textbf{(15)}$$

The curvature of this univariate quadratic can be found by ordinary differentiation,

$$\frac{d^2z}{ds^2}=2(\frac{ad^2+be^2+cde}{d^2+e^2}) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \textbf{(16)}$$

By convention, surface concavity is given a negative sign. This down-slope curvature value will be referred to as longitudinal curvature, *longc*. Curvature in the orthogonal direction (cross sectional curvature *crosc*) is found through similar triangles so that *d* becomes *-e* and *e* becomes *d* (see Figure 4.3). Thus the two measures are,

$$longc=-2(\frac{ad^2+be^2+cde}{d^2+e^2}), \quad crosc=-2(\frac{bd^2+ae^2-cde}{d^2+e^2}) \ldots\ldots\ldots\ldots\ldots\ldots \textbf{(17)}$$

4.3 Limitations of Quadratic Modelling

Any order of polynomial could be used to model local surface patches, but a quadratic function can be regarded as a successful compromise between the number of data points required to uniquely model the surface and the fidelity with which the model fits the true surface. Planar facets would only require three points to define the polynomial so could therefore be modelled using a two by two local window. However, a planar model is not sufficient to identify surface specific points, or *morphometric features* (see Chapter 5).

Polynomials of order 3 or higher can model more convoluted surfaces including breaks of

slope (points of inflection), but require 10 points to define the surface. Since this would require a local window of more than 3 by 3 cells, the degree of surface generalisation becomes much greater. Thus the quadratic function appears the most widely and appropriately used model for general geomorphometric approximation.

Although capable of modelling all six morphometric feature types, there are a number of limitations to the quadratic model, particularly in the way the 6 coefficients are derived. Evans (1979) uses least squares to identify the optimum set of quadratic coefficients. This minimises the difference between modelled and measured values for the local window as a whole, even though it is only the parameter values at the centre of the patch that are of direct interest. As a consequence, the modelled surface does not necessarily pass through the local origin. Zevenbergen and Thorne (1987) argue that if the model is unconstrained at the origin (central data point in the window), it does not represent the land surface accurately.

There are a number of techniques available that reduce the residual value at the local origin. The modelled surface can be constrained by forcing it through the origin before minimising squared residuals (see section 4.4.4 below). Alternatively, a higher order of polynomial can be used so that there is no redundancy in data points. Zevenbergen and Thorne (1987) suggested a partial quartic expression which requires 9 coefficients. The unique solution derived from the 9 data points in a local window ensures fidelity at the origin and centre of each measured grid cell. A set of solutions adopted by other authors(eg Tobler, 1969; Struve, 1977 [both reported in Evans, 1979]) is to give measured cells in the local window different weights, perhaps excluding some cells altogether.

It becomes apparent that the statistically optimum set of coefficients that is used to model a surface patch may not necessarily be the most appropriate when identifying surface parameters at the centre of the surface. More particularly, it is important to assess whether there is any consistent pattern of residuals with respect to specific geomorphometric arrangements. That is, if residuals are not truly random, they are not true residuals in the statistical sense and we may consider reevaluating the form of our surface model. It was this process that was used to describe and model DEM uncertainty in section 3.3. Chapter 6 includes examination and

discussion of the residuals produced by quadratic approximation using a variety of calibrated and 'real' surfaces.

4.4 The Importance of Scale.

The techniques reviewed for morphometric characterisation of DEMs are all constrained by the resolution of the model. The information derived using these techniques is relevant only to the scale implied by the resolution of the DEM. Since this scale is often arbitrarily defined and not necessarily related to the scale of characterisation required, derived results may not always be appropriate. Indeed, most of the acknowledged problems of morphometric characterisation result from either variation at a finer scale than the DEM ('noisy data') or variations on a coarser scale ('flat regions'). What is required are characterisation techniques that are somewhat independent of the resolution of the database used to store topographic information.

From a geomorphological perspective, it would seem ludicrous to only consider surface variation at a fixed scale when an assessment of an entire landscape is desired. Our own judgements both scientifically and 'intuitively' rely on an appreciation of landscape at a variety of scales simultaneously. Variation that occurs with scale is in itself a useful landscape diagnostic. It is for these reasons, that the remains of this chapter will consider how *scale* can be incorporated into the automatic characterisation of DEMs.

4.4.1 Approaches to multi-scale parameterisation.

Given the utility of quadratic parameterisation of a surface, and the requirement that we must consider this parameterisation at a variety of scales, a number of possible solutions present themselves.

(i) Nearest neighbour sampling.

Tesseral based GIS have an implicit scale of processing that can be set either by the

resolution of the data, or by the resolution of a 'view' of those data. The processing resolution of GRASS can be set independently of the data resolution, for example. This provides a mechanism for a multi-scale approach whereby parameterisation is carried out at a number of viewing resolutions, each of which corresponds to a different scale of processing.

The process of resampling at a coarser resolution in GRASS is carried out by 'nearest neighbour' sampling (Shapiro *et al*, 1993). That is, the value of each raster cell at the new resolution is equal to the value of the original cell that is nearest to its centre. As can be seen from Figure 4.4, this can result in spurious surface models, especially when spatial autocorrelation is relatively low. Additionally, the variance of the resampled elevations tends to be reduced as outlying values of a normal distribution are less likely to be resampled than those nearer the centre of the distribution.

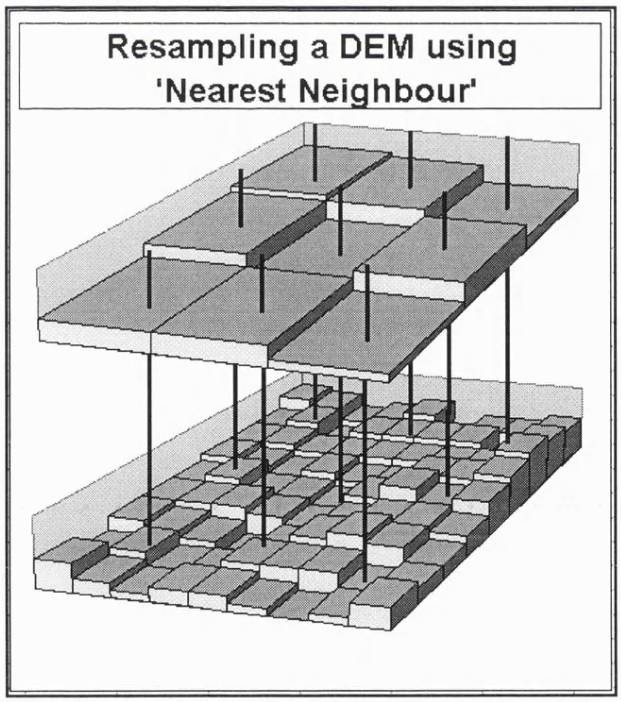


Figure 4.4 Effect of resampling at coarser resolution using nearest neighbour techniques.

(ii) Maximum likelihood sampling

An alternative and more suitable model is adopted here that uses the least squares criterion to change scale. A local quadratic trend surface may be fitted through the original data values from which terrain parameters may be estimated. If we assume residuals from such a model are normally distributed, this process of resampling, is in effect, a maximum likelihood estimation (Unwin, 1975).

It should be recognised that this technique is a generalisation of Evans' 3 by 3 quadratic parameterisation to an *n* by *n* parameterisation. Details of the process are given in the next section.

4.4.2 Multi-scale Quadratic Parameterisation

Consider the derivation of the 6 coefficients that define a quadratic trend surface:

$$z=ax^2+by^2+cxy+dx+ey+f \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(18)}$$

This is more conventionally expressed as a specific case of a more general polynomial expression:

$$z_i = b_0+b_1x_i+b_2y_i+b_3x_i^2+b_4x_iy_i+b_5y_i^2 \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(19)}$$

(eg. Unwin, 1975; Unwin and Wrigley, 1987; Press *et al*, 1988)

To solve this expression i (the number of points sampled) must be at least 6. If we limit the sample of elevation values to a square window centred around the point of interest, i will be 9, 16, 25, ... n^2 as the local window increases from the smallest size (3 by 3) to the largest size (n by n) determined by the smaller side of the DEM. Where $i = 9$, the resultant quadratic expression will be identical to Evans' quadratic derivation (Evans, 1979).

Evans states that for the 3 by 3 case, "*...the procedure [for fitting by least squares] is greatly simplified by the arrangement of data on the square grid. The six parameters are calculated by multiplying the 9 x 1 vector of altitudes by a 6 x 9 matrix of coefficients: matrix inversion*

is unnecessary." (Evans, 1979, p.28). He then goes on to give expressions for the 6 coefficients given in **(18)** as functions of the 9 cells in a 3x3 local window arranged as follows with a planimetric separation of g:

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

$$a = (z_1 + z_3 + z_4 + z_6 + z_7 + z_9)/6g^2 - (z_2 + z_5 + z_8)/3g^2$$
$$b = (z_1 + z_2 + z_3 + z_7 + z_8 + z_9)/6g^2 - (z_4 + z_5 + z_6)/3g^2$$
$$c = (z_3 + z_7 - z_1 - z_9)/4g^2$$
$$d = (z_3 + z_6 + z_9 - z_1 - z_4 - z_7)/6g$$
$$e = (z_1 + z_2 + z_3 - z_7 - z_8 - z_9)/6g$$
$$f = (2(z_2 + z_4 + z_6 + z_8) - (z_1 + z_3 + z_7 + z_9) + 5z_5)/9$$

(from Evans, 1979, p.29)

Whilst this simplification of the least squares solution can be applied to the 3 by 3 case, a matrix solution is required for the more general case of an n by n local window.

To solve the general case, the unknown coefficients are expressed as a set of 6 simultaneous equations, or *normal equations* (Unwin, 1975) *:*

$$\begin{aligned}
b_0N + b_1\sum x_i + b_2\sum y_i + b_3\sum x_i^2 + b_4\sum x_iy_i + b_5\sum y_i^2 &= \sum z_i \\
b_0\sum x_i + b_1\sum x_i^2 + b_2\sum x_iy_i + b_3\sum x_i^3 + b_4\sum x_i^2y_i + b_5\sum x_iy_i^2 &= \sum z_i.x_i \\
b_0\sum y_i + b_1\sum x_iy_i + b_2\sum y_i^2 + b_3\sum x_i^2y_i + b_4\sum x_iy_i^2 + b_5\sum y_i^3 &= \sum z_i.y_i \\
b_0\sum x_i^2 + b_1\sum x_i^3 + b_2\sum x_i^2y_i + b_3\sum x_i^4 + b_4\sum x_i^3y_i + b_5\sum x_i^2y_i^2 &= \sum z_i.x_i^2 \\
b_0\sum x_iy_i + b_1\sum x_i^2y_i + b_2\sum x_iy_i^2 + b_3\sum x_i^3y_i + b_4\sum x_i^2y_i^2 + b_5\sum x_iy_i^3 &= \sum z_i.x_iy_i \\
b_0\sum y_i^2 + b_1\sum x_iy_i^2 + b_2\sum y_i^3 + b_3\sum x_i^2y_i^2 + b_4\sum x_iy_i^3 + b_5\sum y_i^4 &= \sum z_i.y_i^2
\end{aligned}$$

(20)

where coefficients are given as **(19)** above.

The normal equations can be simplified due to the regular nature of the DEM sampling (as Evans, 1979, above). If all observed values z_i are taken from a local window with dimensions n by n, where n is an odd number, a local coordinate system can be defined with the origin at the central cell and a grid spacing of g. Thus for a 5 by 5 window, the coordinates of each cell become:

(-2g,-2g)	(-2g,-g)	(-2g, 0)	(-2g, g)	(-2g, 2g)
(-g,-2g)	(-g,-g)	(-g, 0)	(-g, g)	(-g, 2g)
(0,-2g)	(0,-g)	(0, 0)	(0, g)	(0, 2g)
(g,-2g)	(g,-g)	(g, 0)	(g, g)	(g, 2g)
(2g,-2g)	(2g,-g)	(2g, 0)	(2g, g)	(2g, 2g)

The symmetry of the coordinate system reduces all expressions without even exponents throughout to zero (as observed for the linear case by Unwin and Wrigley, 1987, p.352). In addition, the sum of x^2 is equivalent to the sum of y^2 and the sum of x^4 is equivalent to the sum of y^4. This matrix expression may also be reordered using Evans' notation (**(18)** above). The reordering of the normal equations so that the coefficients of the constant f (or b_0 in **(19)** above) lie in the last row and column make computation of the matrix inversion much simpler (see 4.4.4 below). By expressing the reordered and simplified normal equations in matrix form we are left with the expression:

$$\begin{pmatrix} \sum x_i^4 & \sum x_i^2 y_i^2 & 0 & 0 & 0 & \sum x_i^2 \\ \sum x_i^2 y_i^2 & \sum x_i^4 & 0 & 0 & 0 & \sum x_i^2 \\ 0 & 0 & \sum x_i^2 y_i^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sum x_i^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sum x_i^2 & 0 \\ \sum x_i^2 & \sum x_i^2 & 0 & 0 & 0 & N \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix} = \begin{pmatrix} \sum z_i . x_i^2 \\ \sum z_i . y_i^2 \\ \sum z_i . x_i y_i \\ \sum z_i . x_i \\ \sum z_i . y_i \\ \sum z_i \end{pmatrix} \qquad \textbf{(21)}$$

4.4.3 Computational Implementation

The four constants of the matrix of sums of squares of cross products need only be found once for any given window size. The can be found efficiently for any window size of n^2 cells involving approximately $2n$ calculations:

```
Name:       FindNormal()
Purpose:    Finds the elements of a matrix of squares and cross products
Globals:    n            side of window
            g            grid cell resolution
Locals :    edge         function of window size
            N            number of cells in window
            x2,x2y2,x4   non-zero elements of matrix
            i,i2         counters
```

```
edge = (n-1) / 2        /* Store this value to save computation */

for (i=1 to edge)
{
      i2 = i*i*g*g
      x2 = x2 + i2
      x4 = i2 * i2
}

N = n*n
x2 = x2 * 2* n
x4 = x4 * 2 * n
x2y2 = x2 * x2 / N
```

Algorithm 4.1

Once the elements have been found, the matrix can be inverted (again, only once for a given window size). This is achieved using LU decomposition and LU back substitution (Press et al, 1992). For flexibility and clarity, matrices are stored as double indirection pointers with unit offset, vectors as single indirection unit offset pointers (see *r.param.scale* in the Appendix).

4.4.4 Constrained Quadratic Approximation

If we wish to constrain our quadratic regression we can force the trend surface through the central cell of our local DEM window. The regression surface becomes an *exact interpolator* (at the centre of the function) to use the nomenclature of Lam (1983). This can be achieved by creating a local origin through the central cell in both X-Y and Z directions. Thus all other elevations in the local window are expressed as relative vertical changes from the central value. Once this coordinate system has been adopted, the trend surface is forced through the origin by dropping the quadratic coefficient f (or b_0) and all relevant elements of the normal equations (the bottom row and right-hand column of the matrix of cross products). Hence, (**21**) becomes:

$$\begin{pmatrix} \sum x_i^4 & \sum x_i^2 y_i^2 & 0 & 0 & 0 \\ \sum x_i^2 y_i^2 & \sum x_i^4 & 0 & 0 & 0 \\ 0 & 0 & \sum x_i^2 y_i^2 & 0 & 0 \\ 0 & 0 & 0 & \sum x_i^2 & 0 \\ 0 & 0 & 0 & 0 & \sum x_i^2 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} = \begin{pmatrix} \sum z_i . x_i^2 \\ \sum z_i . y_i^2 \\ \sum z_i . x_i y_i \\ \sum z_i . x_i \\ \sum z_i . y_i \end{pmatrix} \qquad \textbf{(22)}$$

Note that computationally, the solution of this matrix expression can be achieved through the LU decomposition and back substitution routines described in Section 4.4.3 using the full 6 by 6 matrix, but specifying the dimension of 5 in both routines.

Where there is good reason for requiring quadratic approximation though known values, this solution appears more elegant than the partial quartic expression resorted to by Zevenbergen and Thorne (1987). Firstly, there is no good geomorphological reason for resorting to polynomial expressions higher than order 2 (eg. Evans, 1979, Skidmore, 1989). Secondly, the effects of this constraint can be assessed directly by comparing parameters with those derived from the equivalent unconstrained quadratic models.

4.4.5 Weighted Least Squares

Unwin and Wrigley (1987) draw attention to the significance of points around the margins of a trend surface in determining the overall best fit. By quantifying *leverage* the effect of each control point on the resultant quadratic surface can be assessed (Unwin and Wrigley, 1987 p.353). In the context of modelling terrain, this effect would appear most significant for surfaces with low spatial autocorrelation. Here, isolated surface features exert greatest leverage when coincident with the outer corners of the kernel. This can manifest as regular gridlike features on the modelled surface (see Chapter 6).

To counter this problem, a weighting matrix can be incorporated into the determination of modelled quadratic surface. Different cells can be given differing importance when constructing the quadratic function through them, usually some reflection of the positive autocorrelation of most surfaces. This might typically be of an inverse gradient or inverse distance squared form sometimes used in mean filtering in image processing (eg Niblack, 1986, p.79).

In common with all inverse distance functions, an (arbitrary) decision has to be made about the weight of the central cell, which by strict definition would have an undefined weight of 1/0. To avoid the problem of infinite weights, unity is added to each distance in the following weighting function:

$$w_{ij} = \frac{1}{(d_{ij}+1)^n} \qquad \text{. . . . . (23)}$$

where d_{ij} is the Euclidean distance in grid cells to the central cell and n is an exponent ranging from 0 (no distance decay), through 1 (linear decay), to 2 (distance squared decay).

So, for example, the weights for a 5 by 5 kernel with an exponent of 1 would become:

1/3.83	1/3.24	1/3	1/3.24	1/3.83
1/3.24	1/2.41	1/2	1/2.41	1/3.24
1/3	1/2	1/1	1/2	1/3
1/3.24	1/2.41	1/2	1/2.41	1/3.24
1/3.83	1/3.24	1/3	1/3.24	1/3.83

For the most general case, where any weighting matrix may be used, we no longer have the property of symmetry when calculating the elements of the matrix of cross products (as in **(21)** above). Thus the general weighted normal equations matrix becomes,

$$\begin{pmatrix} \sum x_i^4 w_i & \sum x_i^2 y_i^2 w_i & \sum x_i^3 y_i w_i & \sum x_i^3 w_i & \sum x_i^2 y_i w_i & \sum x_i^2 w_i \\ \sum x_i^2 y_i^2 w_i & \sum y_i^4 w_i & \sum x_i y_i^3 w_i & \sum x_i y_i^2 w_i & \sum y_i^3 w_i & \sum y_i^2 w_i \\ \sum x_i^3 y_i w_i & \sum x_i y_i^3 w_i & \sum x_i^2 y_i^2 w_i & \sum x_i^2 y_i w_i & \sum x_i y_i^2 w_i & \sum x_i y_i w_i \\ \sum x_i^3 w_i & \sum x_i y_i^2 w_i & \sum x_i^2 y_i w_i & \sum x_i^2 w_i & \sum x_i y_i w_i & \sum x_i w_i \\ \sum x_i^2 y_i w_i & \sum y_i^3 w_i & \sum x_i y_i^2 w_i & \sum x_i y_i w_i & \sum y_i^2 w_i & \sum y_i w_i \\ \sum x_i^2 w_i & \sum y_i^2 w_i & \sum x_i y_i w_i & \sum x_i w_i & \sum y_i w_i & \sum w_i \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix} = \begin{pmatrix} \sum z_i . w_i . x_i^2 \\ \sum z_i . w_i . y_i^2 \\ \sum z_i . w_i . x_i y_i \\ \sum z_i . w_i . x_i \\ \sum z_i . w_i . y_i \\ \sum z_i . w_i \end{pmatrix} \quad \textbf{(24)}$$

This again may be solved using LU decomposition, for both the constrained and unconstrained cases.

4.5 Spatial Extensions of Terrain Parameters.

Modelling of surface patches with quadratic functions allow the derivative properties of neighbourhoods to be characterised. In particular, the preceding section provides a mechanism for making scale-based relationships explicit. There are however further sets of surface properties that may be characterised that can incorporate a greater degree of spatial

information. The notions of spatial autocorrelation and spatial texture are appropriate to DEM characterisation despite their more usual applications in geostatistics and image processing respectively. Spatial autocorrelation is a property that represents the degree of clustering of like values, while texture characterises the form of clustering. This section shows how both groups of measures can be extended spatially by taking measurements over a range of scales over a DEM.

4.5.1 Spatial Autocorrelation

The degree to which close neighbours over a surface share similar properties is characterised by spatial autocorrelation. Commonly, a single measure of spatial autocorrelation may be found for an entire surface that distinguishes generally smooth surfaces (highly positively autocorrelated), from rougher ones (lower spatial autocorrelation).

Let Z be a surface that is measured at locations i and at separations j, the spatial autocorrelation may be measured using the so-called Moran's I statistic:

$$I = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(w_{ij} \cdot c_{ij})}{\sum_{i=1}^{n} v_i \cdot \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}} \qquad \text{. . . . . (25)}$$

where the w_{ij} is the weighting given to each of the measurements over the surfac, c_{ij} and v_i are equivalent to measures of covariance and variance measured from the surface are defined as

$$c_{ij} = (z_i - \bar{z})(z_j - \bar{z})$$
$$v_i = \frac{(z_i - \bar{z})(z_i - \bar{z})}{n} \qquad \text{. . . . . (26)}$$

where z_i and z_j are the values of Z at locations i and j, z is the mean value of Z over all the locations i and j, and n is the number of measurements. This is the form most commonly

reported in the literature (eg Cliff and Ord, 1981; Goodchild, 1986; Bailey and Gatrell, 1995). Values of I usually range between ± 1, where $+1$ indicates strong positive autocorrelation, -1 a strong negative autocorrelation and 0 a random uncorrelated distribution. The exact limits depend on the sample size and weightings used (see Goodchild, 1986, p.16 and Bailey and Gatrell, 1995 p.270)

Measurements of I over a raster can be simplified if we only measure a single value at j for each location i and give each measurement a unit weighting. Additionally, by simplifying **(26)**, we get:

$$I = \frac{covar_{ij}}{var_i} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}[(z_i - \bar{z})(z_j - \bar{z})]}{\sum_{i=1}^{n}(z_i - \bar{z})^2} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \textbf{(27)}$$

While this measure is relatively simple to calculate in a raster GIS environment, it tells us very little since the measure is highly dependent on the separation or *lag* between i and j (eg. Goodchild, 1986). What is more useful is to calculate the measure at a variety of lags in order to measure how clustering changes with scale. It is this technique that forms the basis of much geostatistical interpolation (eg Jounel, 1978; Isaaks and Srivastava, 1989).

Consider the measurement of I over a raster surface with a lag of 1 grid cell unit to the right. All the locations i that can be measured are shown in Figure 4.5*a*. This includes the entire raster with the exception of the right-most column of grid cells. Likewise the lagged cells j consist of the entire raster bar the left-most column (Figure 4.5*b*). The intersection of these two sets (Figure 4.5*c*) comprises nearly the whole raster, thus,

$$\begin{aligned} var(Z_i) &\approx var(Z_j) \approx var(Z) \\ \bar{z}_i &\approx \bar{z}_j \approx \bar{Z} \end{aligned} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \textbf{(28)}$$

By definition, the mean and variance of all i, j and the whole raster will be equal if the

modelled surface possesses *stationarity* and *homoscedasticity*. Thus for small lags or surfaces with a non varying mean and variance the expressions **(25)** to **(27)** may be used appropriately.

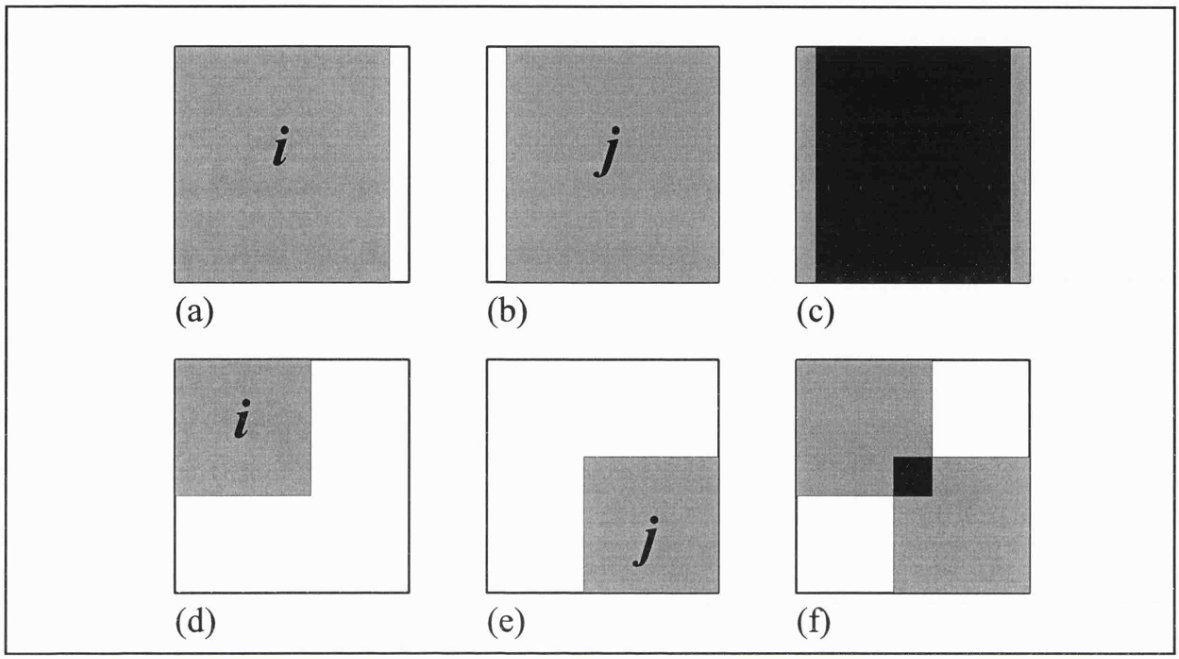


Figure 4.5 - Spatial extent of measured and lagged cells in spatial autocorrelation and variogram calculation.

Consider a larger lag of approximately half the length of one side of the raster in a south westerly direction. The measurable locations *i* occupy a smaller proportion of the entire raster (Figure 5.5*d*). Likewise the lagged locations *j* occupy an equally small, *but different* set of locations (Figure 5.5*e*). For a non-stationary heteroscedastic surface, the variance of *i* will not adequately reflect the variance of all measured cells ($i \cup j$), and as such expressions **(25)** to **(27)** are not valid measurements. It should be noted that such 'edge effects' will also be present for irregular subdivisions of space. Any lagged measure of spatial autocorrelation using this method should be treated with caution for non-stationary, heteroscedastic data.

An initially attractive alternative would appear to be to consider the strong association between spatial *auto*correlation and spatial *correlation* (Wartenberg, 1985). A measure could then be made of the association between *i* and *j* by calculating Pearson's Product Moment Correlation Coefficient:

$$r = \frac{covar_{ij}}{\sqrt{var_i \,.\, var_j}} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}[(z_i - \bar{z_i})(z_j - \bar{z_j})]}{\sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}(z_i - \bar{z_i})^2 \,.\, \Sigma(z_j - \bar{z_j})^2}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \textbf{(29)}$$

However, if the variance of i or the variance of j is 0, the correlation coefficient is undefined.

The alternative suggested here is that both i and j must be included in the measurement of variance. In the case illustrated by Figure 4.5*c* this could be found by calculating the variance of the entire raster. However, for larger diagonal lags (eg. Figure 4.5*f*), there are substantial portions of the raster that are not measured at all as part of covariance calculation. It is significant to note that these unmeasured portions of the raster are not just a function of lag magnitude, but of lag direction. Diagonal profiles to not 'pack' well into a square lattice arrangement (the implications of which are demonstrated in below).

Moran's I for any lag on a raster can therefore be represented as,

$$I = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}[(z_i - \overline{z_{i\cup j}})(z_j - \overline{z_{i\cup j}})]}{\sum_{i=1}^{n}\sum_{j=1}^{n}(z_{i\cup j} - \overline{z_{i\cup j}})^2} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \textbf{(30)}$$

This can be calculated for any lag within a raster with relative computational efficiency (see Algorithm 4.2)

```
Name:       MoranLag()
Purpose:    Calculates Moran' I for any lag in a raster
Globals:    nrows,ncols dimensions of raster
            xlag,ylag   Lag components
            zbar        Mean of raster for cells i U j
            raster[][]  Raster storing surface information
            moran[][]   Stores lagged Moran's I values
Locals :    row,col     Raster coordinates
            var         total variance
            vari,varj   Measured and lagged variance (but see comment)
            zi,zj       The two raster values to compare
```

```
for (row=0;row<nrows;row++)
{
      if ((row+ylag < nrows) AND (row+ylag >= 0))
      {
            for (col=0;col<ncols;col++)
            {
                  if ((col+xlag < ncols) && (col+xlag >= 0))
                  {
                      zi = raster[row][col]
                      zj = raster[row + ylag][col + xlag]
                      covar += (zi - zbar) * (zj - zbar)
                      vari  += (zi - zbar) * (zi - zbar)

                      if ( (row + ylag*2 >=nrows) OR
                           (row + ylag*2 < 0)     OR
                           (col + xlag*2 >=ncols) OR
                           (col + xlag*2 < 0))
                        varj  += (zj - zbar) * (zj - zbar)
                  }
            }
            var = (vari + varj)      /* Note that varj is in fact the */
                                     /* var(i U j) - var (i)          */
      }
}

if (var != 0.0)
      moran[ylag + (nrows/2)][xlag + (ncols/2)] = covar/var
else
      moran[ylag + (norws/2)][xlag + (ncols/2)] = 1.0
```

Algorithm 4.2

To consider possible edge effects in more detail, it is perhaps clearer to compare the modified

calculation of Moran's *I* described above with the more common geostatistical description of scale dependent autocorrelation, the *semivariogram.* The semivariogram function $\gamma(h)$ can be defined as,

$$\gamma(h) = \frac{1}{2n(h)} \sum_{i=1}^{n} \sum_{j=1}^{n} (z_i - z_j)^2 \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \textbf{(31)}$$

where h is the lag between measured cells, n is the number of pairs considered.

This measure is in effect the non standardised version of Geary's index of spatial autocorrelation (Goodchild, 1986). Note that, unlike the original definition of Moran's I given in **(27)**, but in common with the modified definition **(29)**, values of i and j can be interchanged giving identical results (see also Isaaks and Srivastava, 1989, p.60).

Figure 4.6 shows three surfaces (left column) from which both a 2 dimensional variogram Moran' *I* surface have been calculated. Instead of the more usual 1 dimensional profile, $\gamma(h)$ and *I* have been calculated separately for lags in different directions (a series of narrowly defined directional variograms). The results of each calculation are mapped back onto the raster metric, where the lag magnitude and direction are indicated by the location relative to the centre of the raster. The value of the statistic for each lag is colour coded on a linear scale, $\gamma(h)_{min}$, being white, $\gamma(h)_{max}$, black, and *I* having the standard blue-red bipolar colour scheme described in 3.2.2. This is thought to provide a very powerful visual diagnostic of scale dependency and anisotropy of a surface.

Edge effects can most clearly be seen with the variogram surfaces. Consider the centred circle of radius r (Figure 4.6 top left). Its variogram at lags of $\leq r$ is as expected, a gradual increase in variance with lag. As the lag further increases so the lines of equal variance become distorted towards a diamond shape. At the largest lags, maximum variance is confined to the four corners of the surface. This surface can be explained by the packing properties of diagonal lags in comparison with perpendicular lags. For larger diagonal lags, substantial parts of the surface are not measured at all (see Figure 4.5*f*). It is these unmeasured corners that in

the case of the centred circle contain flat regions that would, if measured, reduce overall variance. For perpendicular lags, even large ones, there is no part of the raster that remains unmeasured. A circle that is located in one corner gives an even more strongly anisotropic variogram (Figure 4.6 middle) for the same reasons. An uncorrelated Gaussian random surface is shown in Figure 4.6 (bottom row) along with its variogram. It would be expected that the variogram should have no structure since there is no spatial structure in the original. However, some structure appears evident (disregarding the obvious symmetry of all 2D variograms) as alternating patches of lighter and darker texture. This is due entirely to a systematic change in the sampled cells for larger lags.

The implications for interpreting both 2D and conventional variograms are important to consider here. Edge effects are most apparent here because large lags are included and highly heteroscedastic surfaces are tested. However, even an uncorrelated, stationary homoscedastic surface can give rise to artificial structure. They demonstrate that great care must be exercised in checking the representativeness of samples for variance calculation. This is particularly (but not exclusively) the case for raster based sampling.

The problem can be vastly reduced (but not eliminated) by measuring a standardised autocorrelation measure. The right column of Figure 4.6 shows the Moran's I surface for the same three test surfaces. Because each measure of covariance is standardised by the variance of all cells used in calculation, the effects of systematic spatial bias are reduced. It is only in these extreme heteroscedastic surfaces that any artifacts are produced.

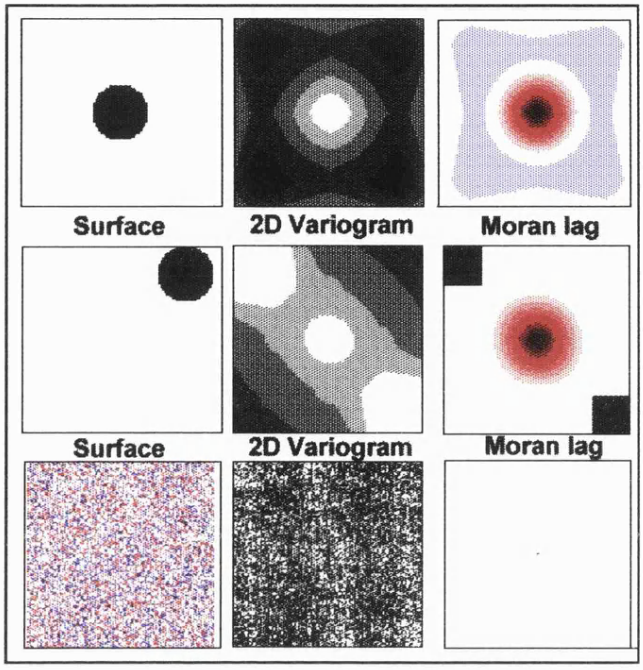


Figure 4.6 - Some geometric surfaces and their 2-dimensional variograms and 'lag' diagrams.

4.5.2 Co-occurrence Matrices

The problems of variogram interpretation introduced by edge effects are partly due to the application of the analysis in a raster-based framework. The development of both regionalised variable theory and measures of spatial autocorrelation arises from the characterisation of irregularly distributed data. Both these techniques have only later been adapted for a raster framework (eg, Cliff and Ord, 1981 and Cressie, 1993). An alternative set of techniques has arisen from the image processing domain, designed specifically for raster based surface data.

Co-occurrence matrices, or grey-tone spatial dependence matrices (Haralick *et al*, 1973) provide quantitative descriptions of the spatial and attribute relationships between cells within a gridded framework. Haralick *et al* (1973) point out that they offer a more refined description of spatial relationship than single spatial autocorrelation measures since they can be used for detecting specific textural features such as rippling, lineation and molling. While being

primarily developed as texture descriptors of grey-tone images, they may be equally well applied to raster DEMs.

Consider a raster surface, that is quantised into k classes. The co-occurrence matrix M consists of k by k cells where $M(x,y)$ consists of the probability of a cell on the original raster having a class M_x and a lagged (h) cell on the same raster having a class M_y. Lags have traditionally been set to first order adjacency in one of 4 possible profile directions, but in theory could be set to any value. Clearly the form of the co-occurrence matrix M will depend on the number of classes k and the lag h used, however there are a number of generalised properties of such matrices. For lagged profiles that extend on either side of measured cells, the resultant matrix will be approximately symmetrical along the leading diagonal. Deviations from symmetry will occur due to the edge effects discussed in section 4.5.1 above. Highly positively autocorrelated surfaces will have the majority of entries concentrated along the leading diagonal of the matrix (that is the class k of any cells similar or identical to the class of its near neighbour). If the 2-dimensional matrix is projected onto a 1-dimensional line down the leading diagonal, the result is equivalent to the probability adjusted frequency histogram. An uncorrelated surface comprising of independent random uniform deviates would result in a co-occurrence matrix of equi-probable cells.

The co-occurrence matrix can be easily calculated for any DEM using the algorithm presented in Algorithm 4.3. Note that for computational efficiency, the matrix is stored as the (integer) sum of co-occurrences rather than their (floating point) probabilities. This can be easily converted by dividing each cell by the total number of measured raster cells n.

```
Name:       Co_occur()
Purpose:    Calculates Co-occurrence matrix from a DEM
Globals:    nrows,ncols dimensions of raster
            xlag,ylag   Lag components
            zmin,zmax   Minimum and maximum raster values
            raster[][]  Raster storing surface information
            cooccur[][] Stores lagged Moran's I values
            quant()     Function to quantize distribution into k classes
Locals :    row,col     Raster coordinates
            zi,zj       The two raster values to compare
```

```
for (row=0;row<nrows;row++)
{
      if ((row+ylag < nrows) AND (row+ylag >= 0))
      {
            for (col=0;col<ncols;col++)
            {
                  if ((col+xlag < ncols) && (col+xlag >= 0))
                  {
                      zi = quant(raster[row][col],zmin,zmax)
                      zj = quant(raster[row+ylag][col+xlag],zmin,zmax)
                      cooccur[zi][zj] += 1
                  }
            }
      }
}
```

Algorithm 4.3

By convention, the co-occurrence matrix is used for calculating a number of textural measures. Haralick *et al* (1973) suggested 14 such features, but found it "hard to identify which specific textural characteristic is represented by each of these features" (Haralick *et al,* 1973, p.613). Weszka *et al* (1976) suggested a sub-set of these measures that corresponded to identifiable textural characteristics. It is a similar sub-set that is considered here, although even in selecting just 5 measures, it is clear that there is some degree of dimensional overlap in what they measure.

The six textural measurements considered here can be defined as follows,

$$contrast = \sum (i-j)^2 \, p(i,j) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (32)$$

$$angular\ second\ moment = \sum [p(i,j)]^2 \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (33)$$

$$entropy = \sum -p(i,j) \log[p(i,j)] \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (34)$$

$$assymetry = \sum [p(i,j) - p(j,i)]^2 \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (35)$$

$$inverse\ distance\ moment = \sum \frac{1}{1+(i-j)^2} \, p(i,j) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (36)$$

in all cases, i and j are the coordinates of the co-occurrence matrix and $p(i,j)$ is the probability associated with each cell ($M(i,j) / n$).

Two new developments that arise from such textural description are considered here. Firstly, it becomes apparent that the 14 texture measures originally suggested by Haralick *et al* (1973), in many cases, quantify information that can be found by visualising the co-occurrence matrix itself. In a scientific visualisation context, the graphical representation of the matrix may well provide important diagnostic information. A routine was written therefore, to represent the co-occurrence matrix of any surface as a grey tone raster (see Appendix). Two matrix visualisations are shown in figure 4.7. Figure 4.7*a* shows the matrix for a fractal surface with fractal dimension 2.10, Figure 4.7*b* shows the matrix for a surface with dimension 2.90. The reduced spatial autocorrelation of the latter is clearly visible as less concentration of values around the leading diagonal. Note that both images represent empty matrix cells (by far the most common) as mid-grey. This is to distinguish them from cells with a few entries (ie a non-zero, but low probability of co-occurrence).

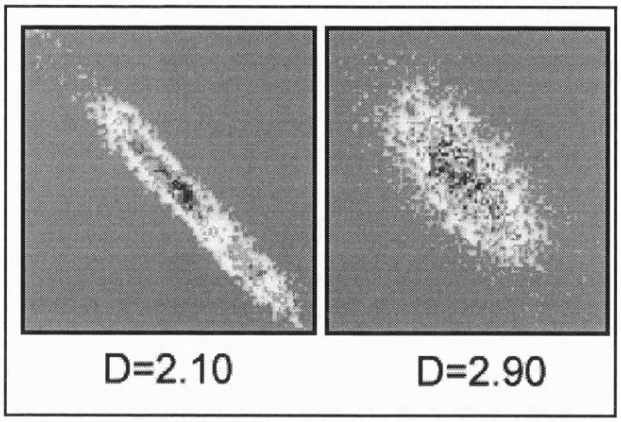


Figure 4.7 - Co-occurrence matrices visualised for 2 fractal surfaces of dimension 2.1 and 2.9.

The final development of the co-occurrence approach is similar to that used for measures of spatial autocorrelation. As with this measure, textural measures are highly dependent on the scale at which texture is measured. The discussion on the relative importance of *tone* and *texture* by Haralick *et al* (1973), is really just one of scale. What appears as texture at one scale, will appear as tone at another. Equally, the notion of *pattern* as an image characteristic address the same type of spatial arrangement at yet another scale. It therefore seems obvious to examine the relationship between texture measurements and scale. This is achieved here by constructing a similar *lag map* to the one described in section 4.5.1, but this time at each lag, one of the five texture measures described above is measured. Thus five maps of the explicit relationship between scale and textural components are created. An example of such a set is shown in Figure 4.8 where a fractal surface of dimension 2.10 is measured.

On of the problems in the interpretation of images such as those described in this section is that without experience, we have no basis with which to make comparisons. Chapter 6 therefore includes a 'calibration' of all tools and visualisations discussed in this chapter, along with an assessment of their utility.

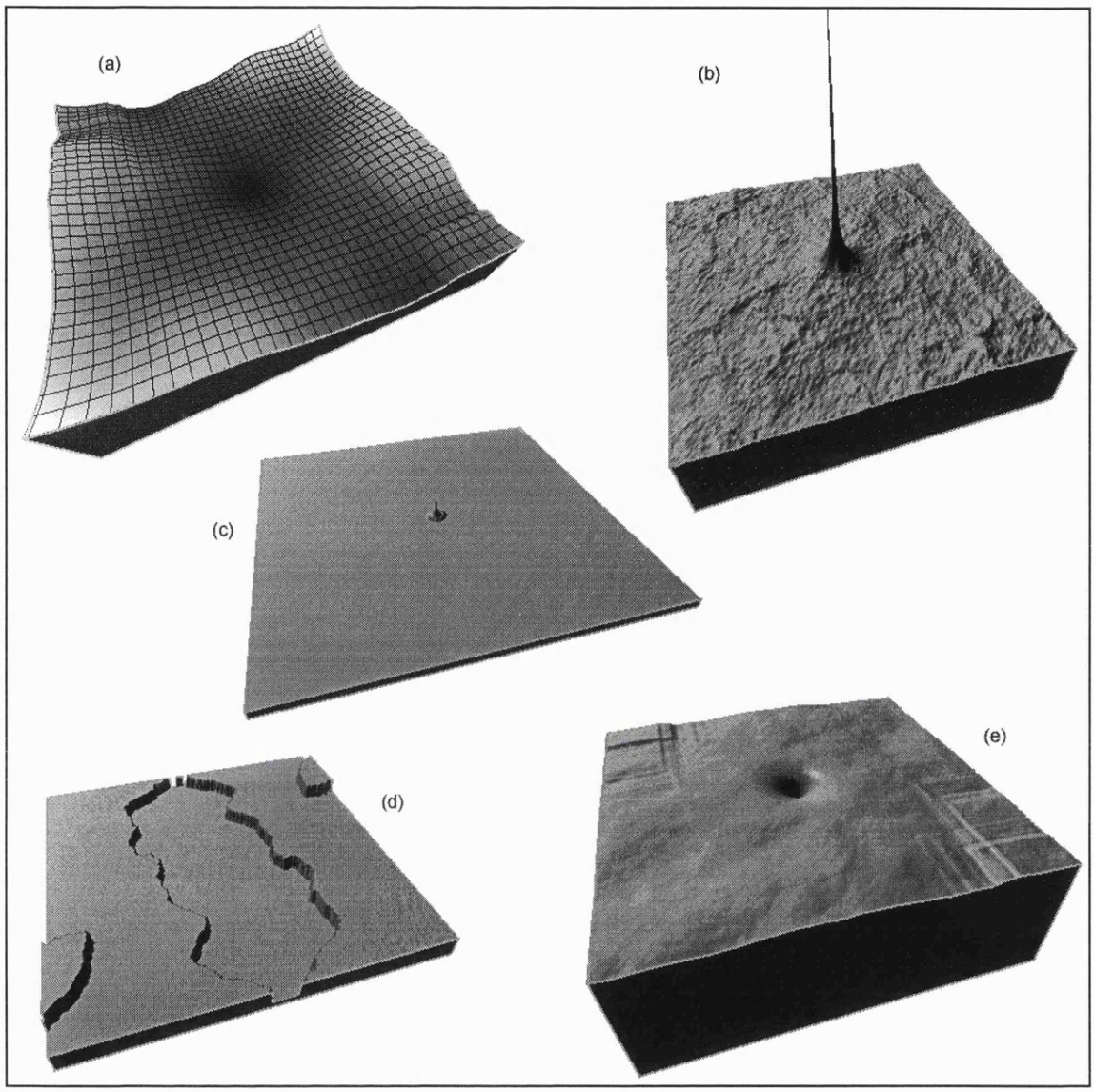


Figure 4.8 - 'lag maps' of 5 co-occurrence matrix texture measures. (a) contrast; (b) inverse distance moment; (c) angular second moment; (d) asymmetry; (e) entropy.

Chapter Five - Morphometric Characterisation

One of the overall aims of this study has been is to develop a set of tools that describe the general geomorphometry of a surface. On the whole, this is quite distinct from the process of identifying specific geomorphometric features such as cirques or floodplains. There are however, a number of surface features that may be used both in the specific and general geomorphometric identification process. These features can be thought of as *morphometric* features rather than *geomorphometric* in that they are characteristic of any surface.

The most widely used set of morphometric characteristics, is the subdivision of all points on a surface into one of *pits, peaks, channels, ridges, passes* and *planes* (see Figure 5.1). The names of these features suggest a geomorphological interpretation, but they may be unambiguously described in terms of rates of change of three orthogonal components (see Table 5.1). Note that the components x and y are not necessarily parallel to the axes of the DEM, but are in the direction of maximum and minimum profile convexity.

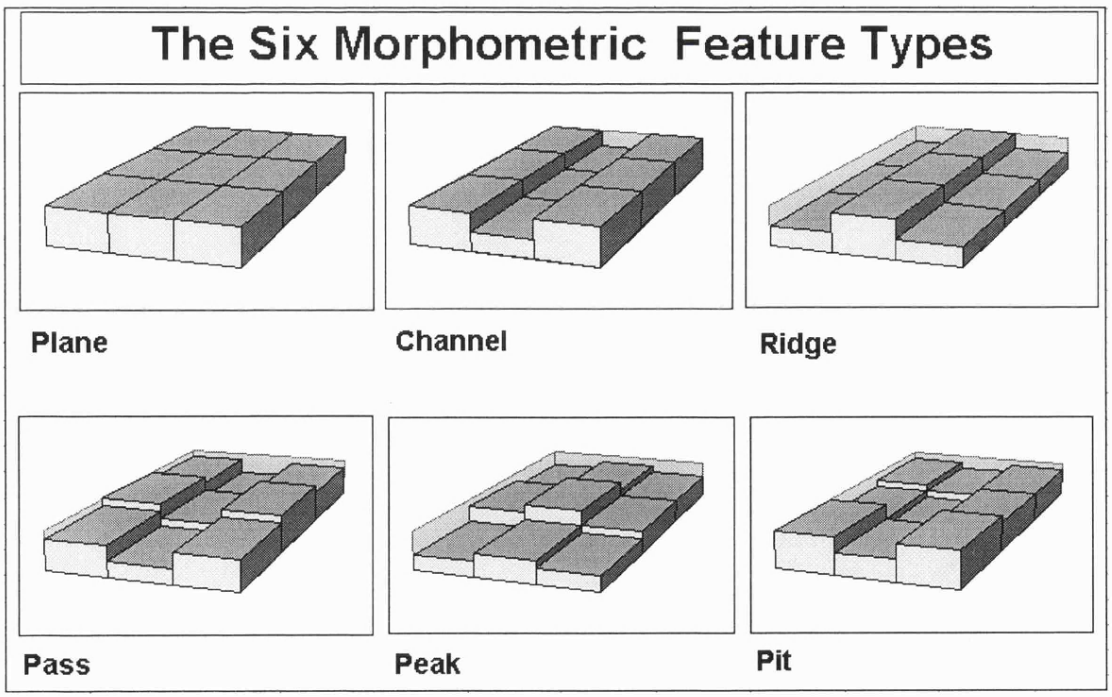


Figure 5.1 - The six categories of morphometric feature illustrated by the relationship between a central DEM cell and its eight neighbours.

Feature name	Derivative expression	Description
Peak	$\frac{\delta^2 z}{\delta x^2}>0, \quad \frac{\delta^2 z}{\delta y^2}>0$	Point that lies on a local convexity in all directions (all neighbours lower).
Ridge	$\frac{\delta^2 z}{\delta x^2}>0, \quad \frac{\delta^2 z}{\delta y^2}=0$	Point that lies on a local convexity that is orthogonal to a line with no convexity/concavity.
Pass	$\frac{\delta^2 z}{\delta x^2}>0, \quad \frac{\delta^2 z}{\delta y^2}<0$	Point that lies on a local convexity that is orthogonal to a local concavity.
Plane	$\frac{\delta^2 z}{\delta x^2}=0, \quad \frac{\delta^2 z}{\delta y^2}=0$	Points that do not lie on any surface concavity or convexity.
Channel	$\frac{\delta^2 z}{\delta x^2}<0, \quad \frac{\delta^2 z}{\delta y^2}=0$	Point that lies in a local concavity that is orthogonal to a line with no concavity/convexity.
Pit	$\frac{\delta^2 z}{\delta x^2}<0, \quad \frac{\delta^2 z}{\delta y^2}<0$	Point that lies in a local concavity in all directions (all neighbours higher).

Table 5.1 - Morphometric Features described by second derivatives.

The identification of these features forms the basis of the techniques described in this chapter for describing DEM characteristics. The first two sections describe how the features themselves may be identified. The third section extends the technique to extract multi-scale behaviour. The final section concentrates on the hydrological implications of the layout of these morphometric features. It is worth noting at this stage that this classification produces point-based categories (pits, passes, and peaks), two line-based categories (channels and ridges) and one area-based category (planes).

5.1 Feature Identification

The standard method of identifying morphometric features is to pass a local (usually 3 by 3) window over the DEM and examine the relationships between a central cell and its neighbours (eg. Peucker and Douglas, 1975; Evans, 1979). Alternative methods exist for terrain modelled with contour lines (eg. Maxwell, 1870; Tang, 1992), but will not be considered here. This section will consider how the multi-scaled parameterisation discussed in the previous chapter may be applied to morphometric feature identification.

5.1.1 Quadratic Approximation

For consistency with the general geomorphometric parameters identified previously, the second derivatives required to identify all six features are extracted using quadratic approximation of some local window (see section 4.4). Cross-sectional curvature (*crosc*) is used to characterise the second derivative as this is the convexity measure that is most closely related to geomorphological process. At locations with a non-zero slope, channels have a negative *crosc*, ridges a positive *crosc*, and (sloping) planes a *crosc* of zero. Additionally, we can measure the longitudinal curvature *longc* in order to define the three remaining feature types. Pits have a negative *crosc* and *longc*, peaks a positive *crosc* and *longc*, and passes *crosc* and *longc* with opposite signs.

For cases of zero slope, the slope direction (aspect), *crosc* and *longc* remain undefined. In such cases an alternative measure of convexity is required that is not based on slope direction. Young (1978) shows how the maximum and minimum convexity values can be derived from the quadratic coefficients *a, b* and *c* for this special case:

For a surface modelled by the quadratic,

$$z=ax^2+by^2+cxy+dx+ey+f \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (1)$$

where gradient is zero ($d^2 = e^2 = 0$), maximum and minimum convexity values are,

$$maxic = -a - b + \sqrt{(a-b)^2 + c^2}$$
$$minic = -a - b - \sqrt{(a-b)^2 + c^2} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2)$$

Together, *slope*, *crosc*, *longc*, *minic*, and *maxic* provide a complete and unique set of conditions for defining all six morphometric features (Table 5.2)

Feature	*slope*	*crosc*	*longc*	*maxic*	*minic*
Peak	0	#	#	+ve	+ve
	+ve	+ve	+ve	#	#
Ridge	0	#	#	+ve	0
	+ve	+ve	0	#	#
	+ve	0	+ve	#	#
Pass	0	#	#	+ve	-ve
	+ve	+ve	-ve	#	#
	+ve	-ve	+ve	#	#
Plane	0	#	#	0	0
	+ve	0	0	#	#
Channel	0	#	#	0	-ve
	+ve	-ve	0	#	#
	+ve	0	-ve	#	#
Pit	0	#	#	-ve	-ve
	+ve	-ve	-ve	#	#

Table 5.2 Morphometric features defined by the sign of five morphometric parameters. # indicates undefined, or not part of selection criteria.

It becomes necessary to check longitudinal curvature when identifying ridges and channels where the ridge/channel sides are of significantly different heights. In such cases the slope

direction is unlikely to be 'down-channel' (or along the ridge), but obliquely across the length of the feature. Additionally there is a possibility of misclassification for features similar to those illustrated in Figure 5.2. In Figure 5.2(*a*) the negative cross-sectional curvature (orthogonal to the slope vector represented by an arrow) suggests a channel, but the longitudinal curvature is also negative, forcing classification as a pit. Figure 5.2(*b*) shows a similar situation where a positive longitudinal curvature forces an apparent channel to be classified as a pass. The result of applying this classification algorithm to a real geomorphological surface is to overrepresent the numbers of point-based categories (peaks, passes and pits).

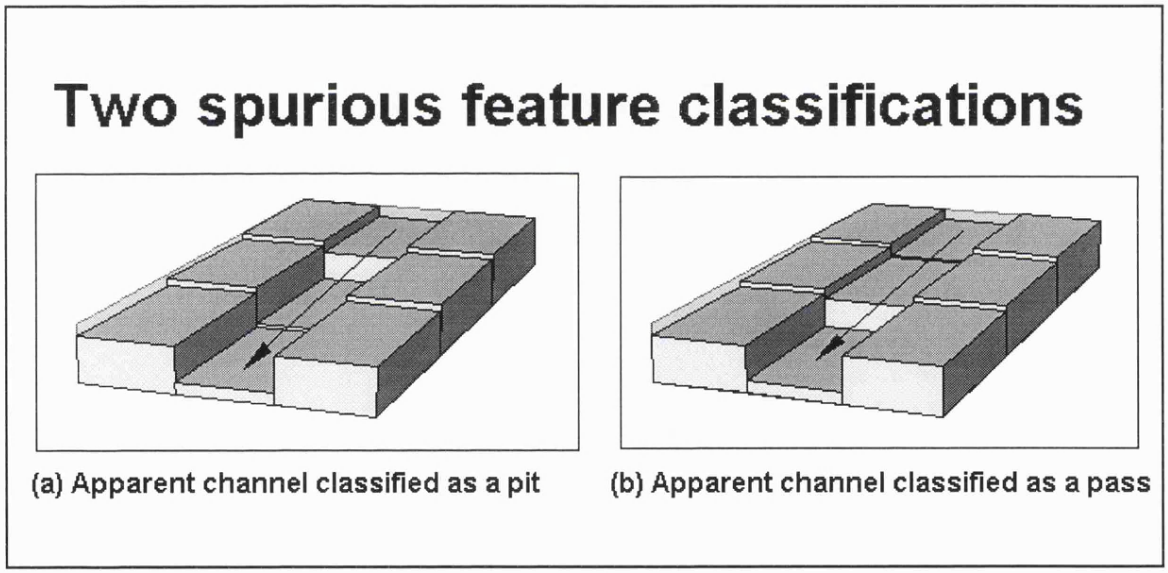


Figure 5.2 - Two misclassified features due to asymmetric neighbourhoods.

A simplified and improved algorithm for classification is suggested here that preserves the continuity of line-based channels and ridges to a far greater extent. This provides an advantage over traditional methods of feature selection based on logical comparison of neighbours (eg, Peucker and Douglas, 1975; Jensen, 1985; Bennet, 1989; Skidmore, 1990).

It is assumed that all locations that have a local slope must be either planar, form part of a

channel or form part of a ridge. Pits peaks and passes are assumed only to occur where local slope is zero. These assumptions are perhaps closer to our own models of the surface features. The anomalous classifications shown in Figure 5.2 are eliminated. The selection criteria are shown in Table 5.3.

Feature	*slope*	*crosc*	*maxic*	*minic*
Peak	0	#	+ve	+ve
Ridge	0	#	+ve	0
	+ve	+ve	#	#
Pass	0	#	+ve	-ve
Plane	0	#	0	0
	+ve	0	#	#
Channel	0	#	0	-ve
	+ve	-ve	#	#
Pit	0	#	-ve	-ve

Table 5.3 Simplified feature classification criteria.

These rules provide the basis for the morphometric classification considered in the rest of this chapter.

5.2 Slope and Curvature Tolerance

The method described above, if strictly applied will tend to produce surfaces that consist almost entirely of interdigitating channels and ridges. This is because (i) a quantised DEM will rarely produce truly planar facets with profile convexity components of zero (the only relatively common exception being flat regions such as lakes and coastal areas), and (ii) true peaks, passes and pits usually have an overall slope value when their neighbours are considered. Figure 5.3 shows how small variations in one or two values can change the nature of the feature detected. To overcome this two tolerance values are introduced that account for (i) and (ii) above.

Let *Tconvex* be the minimum *crosc* that represents a true cross-sectional convexity/concavity. In effect, this value defines the minimum concavity of a channel cross-section and convexity of drainage divides. Let *Tslope* be the minimum *slope* that represents a true slope. In effect, this value defines the minimum longitudinal channel or ridge slope in which local variations can occur, without breaking it up with a series of pits, passes and peaks. The values of these tolerances are somewhat arbitrary, yet it is necessary to have the flexibility to vary the feature selection process according to the nature of terrain. The algorithm representing these modified rules is shown in Algorithm 5.1 (coded as part of *r.param.scale* and *d.param.scale* in the Appendix). The results of varying the both tolerance values for a mountainous DEM are shown in Figure 5.4 as *small multiples* (Tufte, 1990). The position of each image within the figure indicates the magnitude of the two tolerance values.

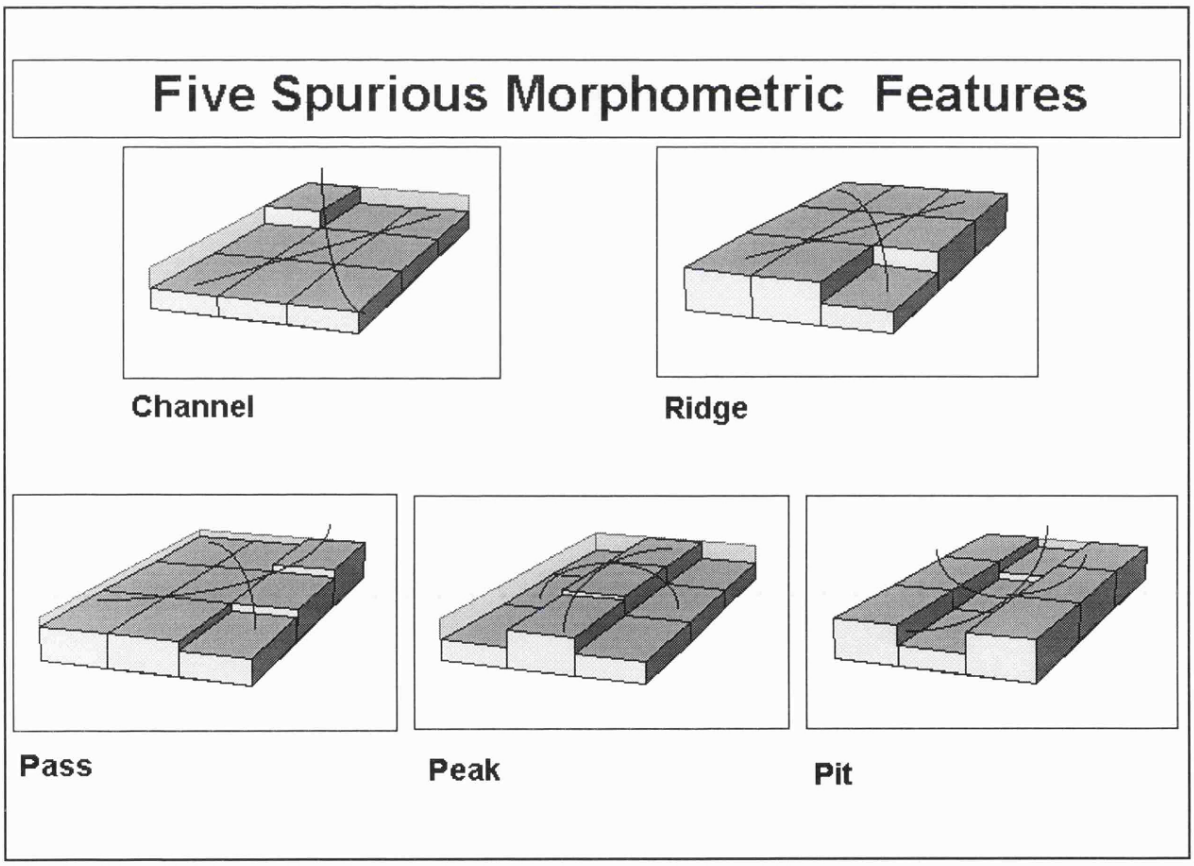


Figure 5.3 Misclassification of features without suitable tolerance values.

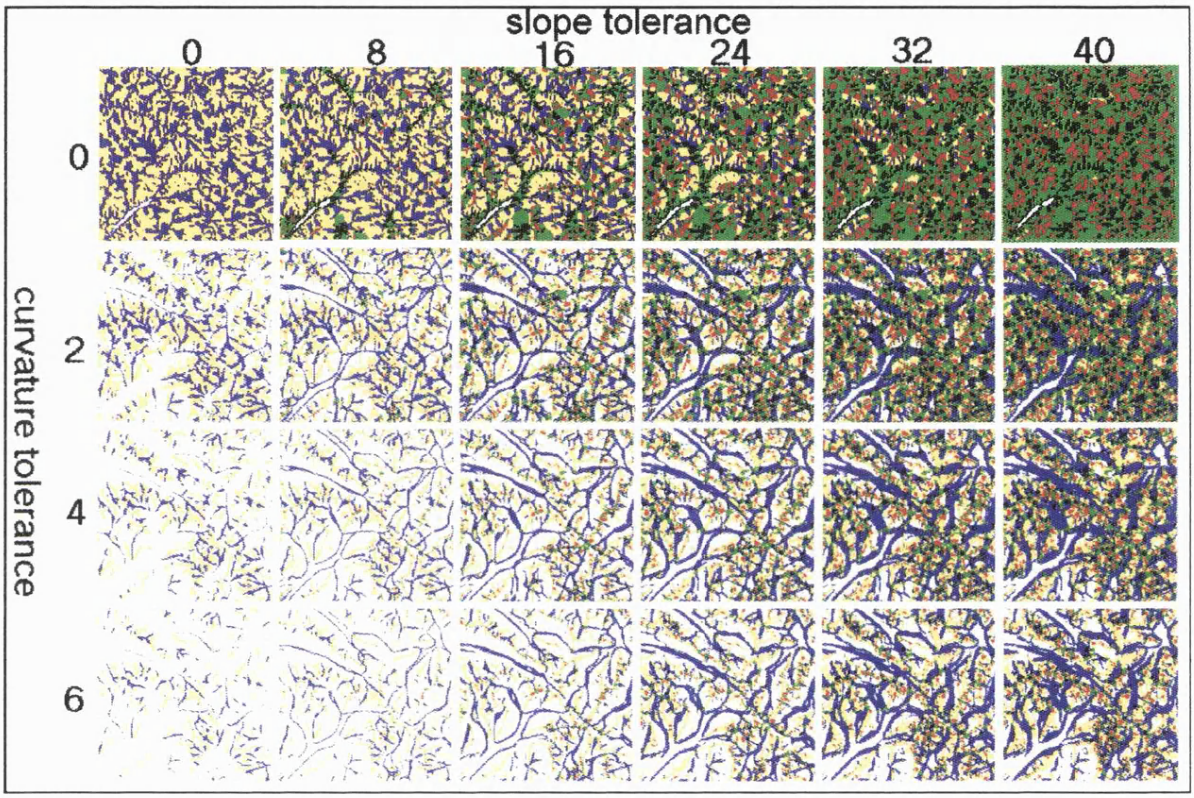


Figure 5.4 - Effect of tolerance values on feature classification. Columns represent a relaxing of the slope tolerance (degrees); rows represent increasing curvature tolerance (dimensionless). Features are classified as *pits* (black), *channels* (blue), *passes* (green), *ridges* (yellow), and *peaks* (red).

```
Name:          Mfeature(coeff,n)
Purpose:       Identifies the morphometric feature described by a bivariate quadratic
               function.
Parameters:    coeff[6]       Array holding the 6 quadratic coefficients a to f
               n              Size of side of local window
Globals:       Tslope  Minimum gradient that defines a non flat surface
               Tconvex        Minimum convexity defining a non-planar surface
               g              Grid resolution of DEM
Locals :       slope          Maximum gradient
               crosc          Cross-sectional convexity
               minic,maxic    Minimum and maximum convexity
```

```
Mfeature(coeff)
{
        /* Measure morphometric parameters */
        slope = atan(sqrt(d*d + e*e))
        crosc = n*g*(b*d*d + a*e*e - c*d*e)/(d*d + e*e)
        maxic = n*g*(-a-b+sqrt((a-b)*(a-b) + c*c))
        minic = n*g*(-a-b-sqrt((a-b)*(a-b) + c*c))

        /* Identify morphometric features */
        if (slope > Tslope)           /* Case 1: Surface is sloping */
        {
            if (crosc > Tconvex)
                return (RIDGE)
            if (crosc < -Tconvex)
                return (CHANNEL)
            else
                return (PLANAR)
        }
                                      /* Case 2: Surface is horizontal */
        if (maxic > Tconvex)
        {
            if (minic > Tconvex)
                return (PEAK)
            if (minic < -Tconvex)
                return (PASS)
            else
                return (RIDGE)
        }

        if (minic < -Tconvex)
        {
            if (maxic < -Tconvex)
                return (PIT)
            else
                return (CHANNEL)
        }

        return(PLANAR)
}
```

Algorithm 5.1 - Feature classification.

5.3 Multi-scale feature classification

The argument presented in the previous chapter was that one resolution dependent parameterisation is not sufficient to describe surface form. Surface parameters should be able to be expressed at a variety of sizes. Likewise, a rule-based classification of morphometric features based on those parameters should also be multi-scale. If we are to match morphometic 'peaks' to our own geomorphometric understanding, it is very unlikely that the scale defined by the DEM resolution will be sufficient. Thus the rules for feature classification defined above are scale independent. The parameters *a-f* in **(1)** can be found for any window size before being used for classification.

The result of applying a multi-scale approach is that each location has multiple feature attributes. These can be visualised in a number of ways. Animation may be used with 'time' representing change in spatial scale (an approach adopted by Wood *et al*, 1996 for characterising population density surfaces). Alternatively, small multiples may be used to show the 'slices' through this sequence (see Chapter 6 for examples of this). The third alternative that is adopted here for an interactive visualisation context, is the multi-scale 'probe'. A mouse is passed over a DEM so that when a mouse button is clicked, the feature classification of the relevant DEM cell is found over a range of window sizes. The result is a graphical representation of the *feature membership function.* Equally important is an equivalent representation of the morphometric parameters (*crosc* and *slope*) used for the classification. The C code for interactive probing in this way is shown in the Appendix (*d.param.scale*).

Figures 5.5 and 5.6 show an example of this probe applied to part of the Lake District DEM. The cross in the upper image of both figures indicates the point selected for query. In both cases this corresponds to Mickledore - a well known gap between England's two highest mountains, Scafell Pike (to the North East) and Scafell (to the South West). This location would probably be considered by most visitors to the location as a 'pass' with two steep gullies running orthogonally to the direction of the two peaks on either side. Yet if we are to classify

the point location, the type of feature should (and does) depend on the scale of interest. The lower image in both figures shows the relationship between the parameter or feature classification and local window size. The X-axis in both figures ranges from 3 x 3 cells at the origin to 69 x 69 cells at the right hand side. At the resolution of the DEM this corresponds to a spatial range of 150m to 3.5km. The extent of the kernel shown graphically in Figure 5.7 with kernels 3x3 to 29x29 shown as red squares, and kernel boundaries indicated for sizes 39x39, 49x49, 59x59 and 69x69.

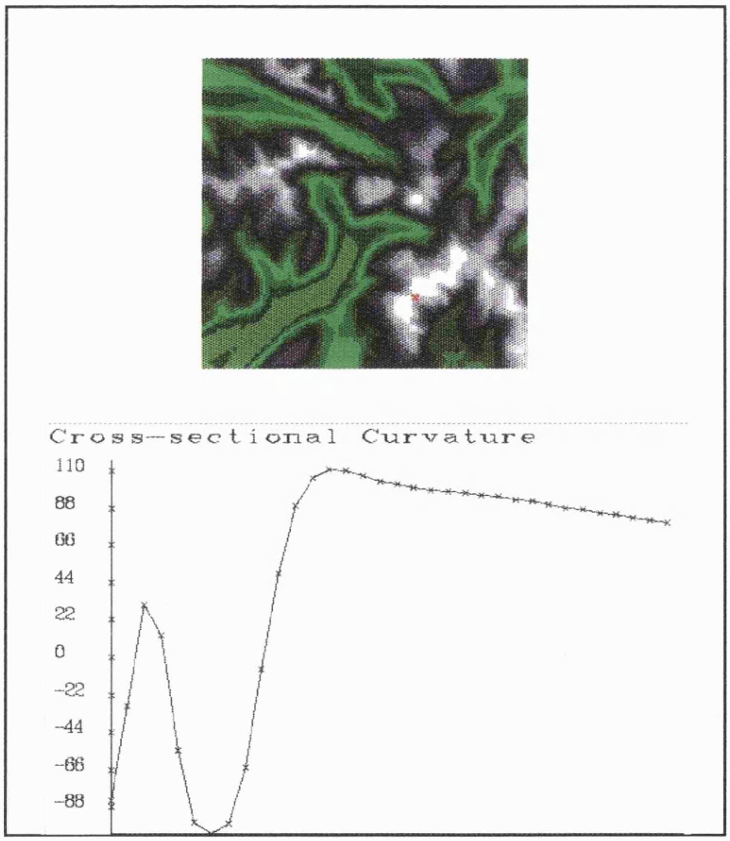


Figure 5.5 - Output from the *crosc* scale-based probe.

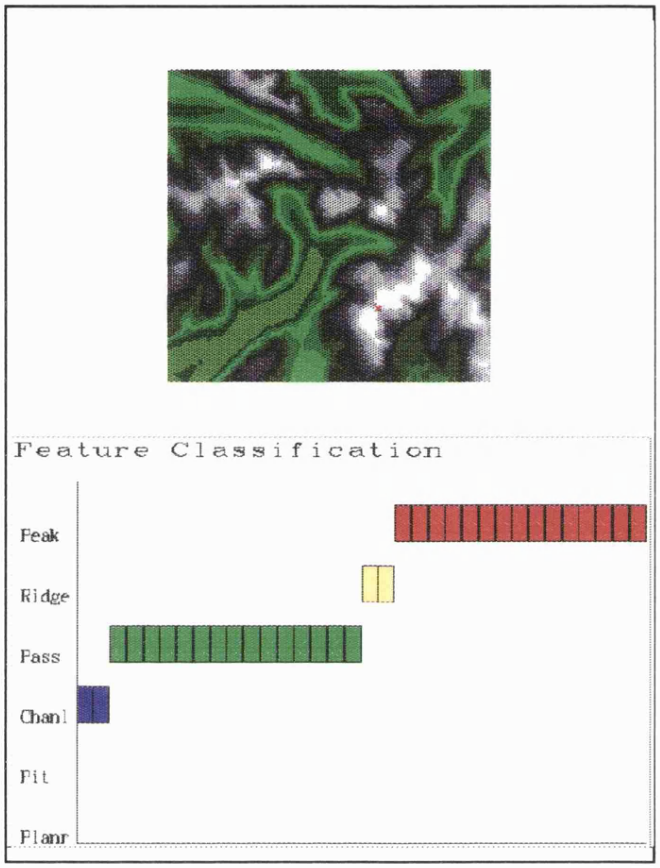


Figure 5.6 - Output from the feature classification scale-based probe.

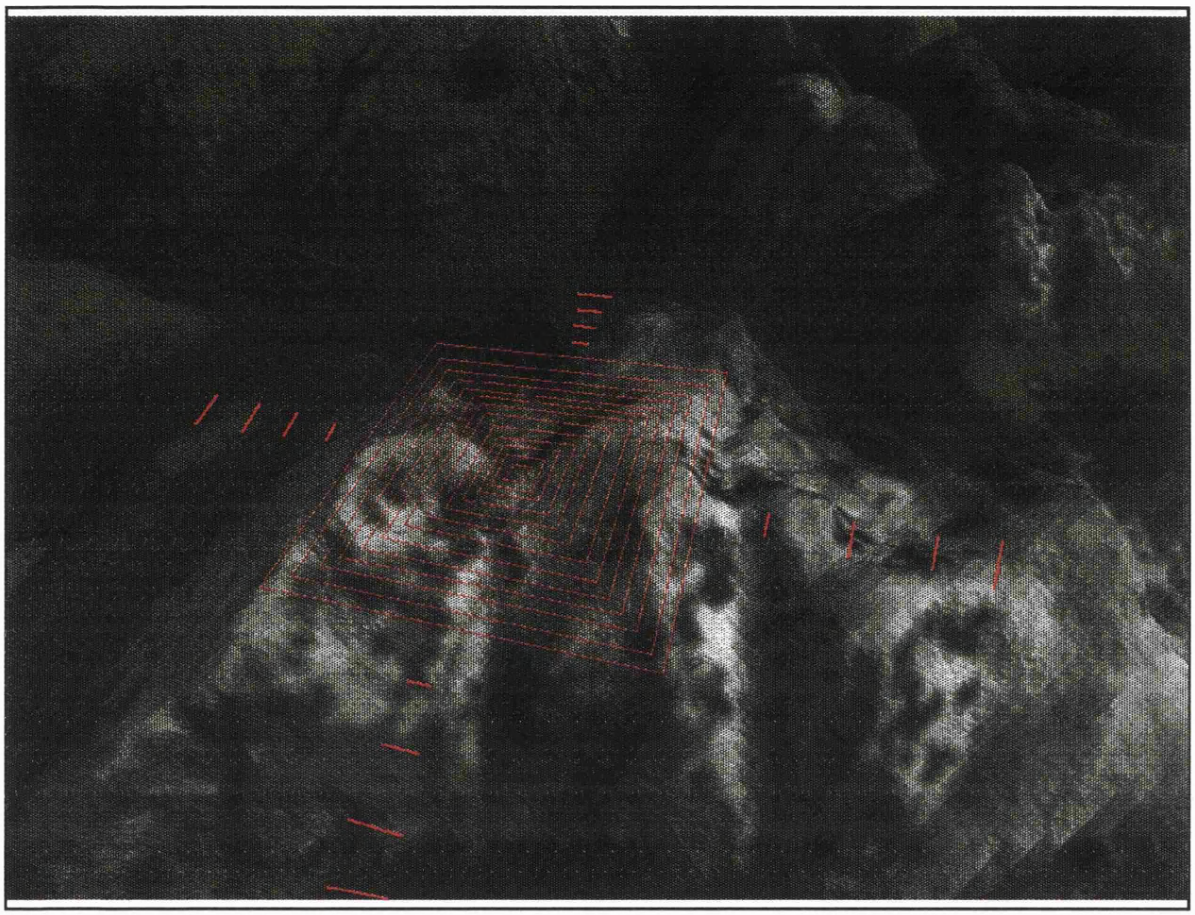


Figure 5.7 - Visualisation of DEM kernels superimposed on the Scafell range.

Both cross-sectional curvature and feature type vary with scale. Short lags would indicate that the location is part of a convex channel. As the scale is widened, it is revealed that this channel lies within a larger pass, that itself is part of an extended convex ridge. The largest lags indicate a peak feature. It is suggested that this scale based progression of characteristics is much more useful than a single morphometric parameter or classification. It provides a landform *signature* (Pike, 1988) that is more discriminating that a single feature classification, but sufficiently general to be of use in an analytical context.

There are likely to be contexts when a single measure or classification is required, so it is useful to at least quantify the variability of measurements. For (the ratio scale) parameters, the mean and standard deviation are calculated. For the categorical feature classification, the mode and scaled entropy is calculated,

$$E = -\sum_{i=1}^{n} p(f_i) \cdot \log[p(f_i)]$$
$$E_{max} = -\log(1/n) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (3)$$
$$E_s = \frac{E}{E_{max}}$$

where $p(f_i)$ is the proportion of measurements classified as feature type i and n is the number of feature types (6 in this study).

This may be used to distinguish locations that are consistently classified as the same feature ($E_s = 0$) from locations that have a high degree of scale dependency in their classification ($E_s = 1$). While this form of interrogation is useful in an interactive visualisation context, it is difficult to identify the spatial pattern of entropy across the image. To visualise both the spatial distribution and scale dependency of measures it is necessary to create several raster layers containing morphometric feature classification at a range of scales. Using simple map algebra, these maps can be combined to produce two new images, one of the modal classification, the other of entropy (see Algorithm 5.2). Thus it is possible to produce a feature membership map and a classification uncertainty map. These may be combined into a single hue-intensity image (see Chapter 6).

```
# Map Algebra script for calculating modal classification of morphometric
# features. Creates two maps, one of modal classification, one of entropy.
#
# Note: feat9, feat15, feat21 etc. are the map layers containing feature
#       classification at the corresponding kernel sizes.
#       pplne,ppit,pchan etc. contain the proportion of maps classified
#       in each feature category.

r.mapcalc << EOF

pplne =      (feat9 ==0) + (feat15==0) + (feat21 == 0) +
             (feat27 == 0) + (feat33 == 0) + (feat45 == 0)

ppit  =      (feat9 ==1) + (feat15==1) + (feat21 == 1) +
             (feat27 == 1) + (feat33 == 1) + (feat45 == 1)

pchan =      (feat9 ==2) + (feat15==2) + (feat21 == 2) +
             (feat27 == 2) + (feat33 == 2) + (feat45 == 2)

ppass =      (feat9 ==3) + (feat15==3) + (feat21 == 3) +
             (feat27 == 3) + (feat33 == 3) + (feat45 == 3)

pridg =      (feat9 ==4) + (feat15==4) + (feat21 == 4) +
             (feat27 == 4) + (feat33 == 4) + (feat45 == 4)

ppeak =      (feat9 ==5) + (feat15==5) + (feat21 == 5) +
             (feat27 == 5) + (feat33 == 5) + (feat45 == 5)

entrop = 1000*     ((pplne/6.0)*log(.001 + pplne/6.0) +
                   (ppit/6.0)*log(.001 + ppit/6.0)    +
                   (pchan/6.0)*log(.001 + pchan/6.0)  +
                   (ppass/6.0)*log(.001 + ppass/6.0)  +
                   (pridg/6.0)*log(.001 + pridg/6.0)  +
                   (ppeak/6.0)*log(.001 + ppeak/6.0))
                                              / log(1.0/6.0)

mode =       (max(pplne,ppit,pchan,ppass,pridg,ppeak)==pplne)*0 +
             (max(pplne,ppit,pchan,ppass,pridg,ppeak)==ppit)*1  +
             (max(pplne,ppit,pchan,ppass,pridg,ppeak)==pchan)*2 +
             (max(pplne,ppit,pchan,ppass,pridg,ppeak)==ppass)*3 +
             (max(pplne,ppit,pchan,ppass,pridg,ppeak)==pridg)*4 +
             (max(pplne,ppit,pchan,ppass,pridg,ppeak)==ppeak)*5

EOF
```

Algorithm 5.2 - Classification membership calculation using map algebra.

5.4 Hydrological Modelling

One of the most direct relationships between geomorphological form and process is that between fluvial and hydrological process and resultant features. This is one of the most widely examined relationships that may be derived from a DEM (see section 2.3). There are several reasons for this pertinent to the objectives of this study. Firstly, although fluvial process results in geomorphological form, the process may be inferred from the nature of that form at the scale of the DEM. This is a relatively rare two way relationship (compare, for example, with other process such as frost heave, longshore drift, weathering etc.). It suggests that the use of GIS-based analysis of DEMs can progress beyond the characterisation of form to an assessment of surface process. Consequently fluvial and hydrological modelling have many immediate applications in the GIS arena. Secondly, fluvially eroded channels are comparatively unambiguously defined geomorphological features that can be represented using a DEM. There is of course, a scale dependency in the effectiveness of this channel modelling. Thirdly, and most importantly in the context of this study, hydrological processes allow us to define important spatial properties of a surface. The drainage basin represents a fundamental geomorphological areal unit, providing both an hierarchical and exhaustive spatial sub-division. Its clear scale dependency is appropriate to the style of analysis adopted in this study, and so will be considered in a little more detail here.

This section will not consider the extraction of drainage networks explicitly. This has been investigated by many authors elsewhere (see section 2.3). It has already been suggested that many of the problems associated with drainage network identification are due to a failure to consider an appropriate range of scales in classification. Some of these problems could be overcome by adopting the multi-scale feature classification described in the previous section. The connected macro-scale valley networks identified with larger kernels can be used to force hydrological connectivity at the raster cell scale. For more details of this style of approach (albeit in a rather more primitive form) see Wood (1990*b*).

5.4.1 Drainage basin identification

The identification of the drainage basin is an important process in both characterising a surface and in defining spatial units that are appropriately related to geomorphological process. The method for basin identification adopted here also identifies two allied and important surface characteristics. Firstly, basin delineation implies the identification of drainage divides - a linear network whose topology and geometry provide useful summaries of a surface's form. Secondly, a modelled *flow magnitude* surface is produced. Each location on this surface has a value associated with the proportion of surface flow that might be expected to pass over it. This has long been used to identify drainage networks (eg Mark, 1983*a*). The topology of this surface could be characterised in much the same way as blue-line drainage networks (Shreve, 1966).

Most basin identification algorithms involve a 'basin climbing' approach where a basin outflow point is identified and the basin is recursively 'climbed' until all points flowing from the drainage divide have been covered (eg. Marks *et al,* 1983). This is broadly the approach adopted here (see *r.basin* in the Appendix) . There are, however, a number of problems associated with identifying drainage basins in this way. If a single outflow cell is missed at the base of a steep sided valley, diagonal or orthogonal basin edges can be produced running up the sides of the valley walls. If hydrological consistency is required, pits in the surface halt recursive algorithms that only travel 'upstream' from an outlet (eg Band, 1989). The result is either a basin identification routine that does not work, or the production of internal basins with no apparent outflow. Yet the occurrence of measured pits is relatively common in mountainous terrain where the valley neck is narrow (eg cirque formation). The valley neck itself may not be picked up at the (planimetric) scale of the DEM, or the terrain at the neck may be sufficiently flat to allow elevation uncertainty to dominate.

A number of possible solutions may be applied to this 'problem'. Pits can be filtered out as a preprocessing option. This may however, (arbitrarily) change elevation values unnecessarily. Alternatively, internal basins may be merged as a post-processing operation. This may be achieved by either 'excavating' cells that connect the base of a pit to its adjacent downstream

basin, or by flooding pits until outflow is redirected. The latter is adopted here (see *r.basin* in the Appendix). Figure 5.8 shows the effect of recursive pit removal on basin derivation for a 30x30km region of the Lake District. Superimposed on (*a*) and (*b*) is the distribution of flooded pits (in shades of red). In all cases, internal basins (*a*) contain at least one of these pits. The basins in Figure 5.8(*b*) correspond to the major valley systems of this part of the Lake District (clockwise from the top, *cyan,* Borrowdale; *mauve,* Thirlmere valley; *pink,* Ullswater valley; *blue*, Windermere/Coniston valleys; *purple*, Dunnerdale; *plum* Eskdale; *pink,* Wasdale).

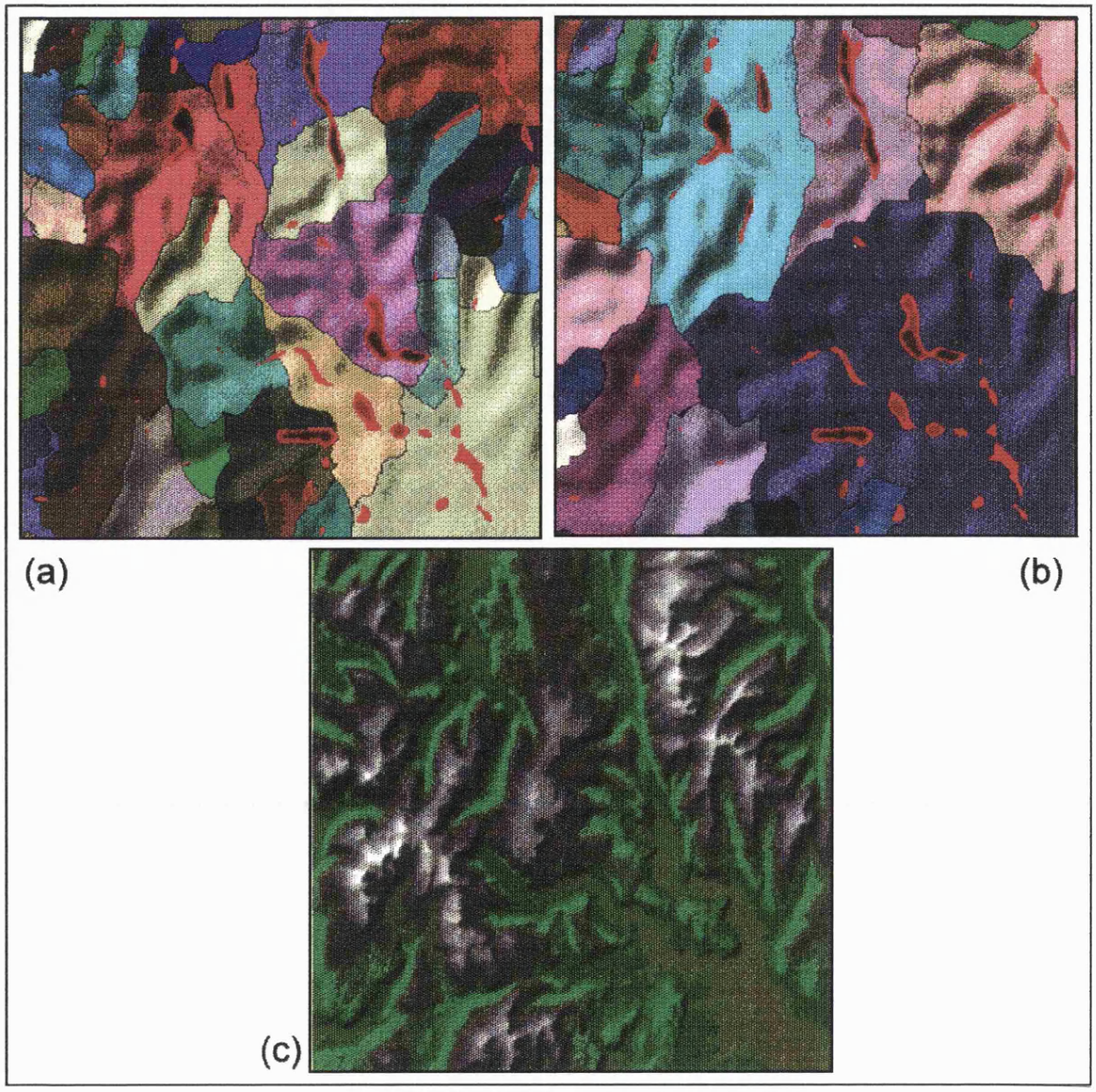


Figure 5.8 - The effect of 'hydrological enforcement' on drainage basin derivation (pit removal shown in red). (*a*) smoothed DEM without hydrological enforcement; (*b*) with hydrological enforcement; (*c*) topographic context.

For geomorphological characterisation (as opposed to hydrological analysis) it is arguable whether the identification of internal drainage basins is in fact a problem. Valley systems with narrow necks *should* be represented as pits at some scale. It is only a problem for hydrological modelling that the mechanism for water removal is expressed at a different scale. The relationship between basin delineation (not hydrologically enforced) and scale is shown in Figure 5.9. Quadratic approximation is used to smooth the DEM with kernel sizes up to 950m. The basin finding routine was then applied to the smoothed surfaces. As expected the number of basins decreases with generalisation, although not spatially consistent.

Figure 5.9 - The effect of scale based smoothing on drainage basin derivation. (*a*) original DEM; (*b*) 7x7 kernel; (*c*) 13x13 kernel; (*d*) 19x19 kernel.

Chapter Six - Results.

This chapter contains an assessment of the DEM characterisation tools that have been developed in the previous chapters. This takes three forms; a theoretical evaluation of a technique's validity; a calibration of the visualisation tools; and an investigation into a technique's utility through application.

A theoretical evaluation has, in part, already been discussed in the previous three chapters. Since many of the techniques used involve the quadratic approximation of local surface patches, the implications of this form of modelling will be assessed in greater detail. The more abstracted measurements and their visualisation (for example, the lag diagrams discussed in section 4.5) require some form of calibration where the geomorphometric analysis yields measurements that are not intuitively obvious. This is most effectively carried out by applying such measures to surfaces with simple and known properties. These can then be applied to surfaces whose properties are not known. Finally the utility of the techniques and methodology proposed is illustrated with the analysis of 'real' topographic surfaces.

6.1 Control Surfaces.

The evaluation of characterisation tools involves using some control surfaces whose properties are known or can be systematically altered. This section outlines their derivation and utility.

6.1.1 Uncorrelated Gaussian Surfaces.

If one of the objectives of surface characterisation is to identify spatial structure, it is useful to make comparison with surfaces that have no spatial structure at all. For the purpose of this study, a GRASS module, *r.gauss.surf* was written (see Appendix) that generates an uncorrelated random surface with a Gaussian (normal) distribution. The resolution, size, mean and standard deviation can all be controlled from the program.

The *Box-Muller* method is used for generating the random deviates with Gaussian distribution (see Press *et al*, 1992, pp.288-290). This involves transforming two uniform random deviates between 0 and 1 (x_1 and x_2) as follows,

$$y_1 = \sqrt{-2 \ln x_1}.\cos(2 \pi x_2)$$
$$y_2 = \sqrt{-2 \ln x_1}.\sin(2 \pi x_2) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \textbf{(1)}$$

$$G(y_1, y_2) = - \left[\frac{1}{\sqrt{2\pi}} e^{-y_1^2/2}\right]\left[\frac{1}{\sqrt{2\pi}} e^{-y_2^2/2}\right] \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \textbf{(2)}$$

6.1.2 Object Surfaces.

Alternatively, a control may be used that has a high degree of spatial structure. Such surfaces are particularly useful for calibrating lag diagrams (for example, see the discussion on measures of spatial autocorrelation, section 4.5.1).

To create simple geometric objects within GRASS, a module *r.xy* (see Appendix) was written that creates two raster surfaces based on the coordinates of each cell location. An origin is set at the bottom left corner, with unit coordinate displacement,

$$f(x,y) = x$$
$$f(x,y) = nrows - y \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \textbf{(3)}$$

These surfaces may be manipulated using map algebra (Tomlin, 1989; Shapiro and Westervelt, 1992), to create simple geometrical surface objects. For example to create a cylinder of height 20 units and of radius 50 units centred over a 201 x 201 cell raster, the following map algebra expression was issued using the GRASS module *r.mapcalc*,

```
circle = if ((x - 101)*(x-101) + (y-101)*(y-101) > 2500, 20, 0)
```

where **x** and **y** are raster layers created using *r.xy*.

To create a 20 x 20 x 20 unit cube in the top left corner of a 201 x 201 cell raster,

```
cube= if ( (x < 20) && (y>180), 20, 0)
```

6.1.3 Fractal Surfaces

The third form of control surface used in evaluation allows a degree of spatial structure to be controlled. It was originally anticipated that this could be achieved by creating polynomial surfaces of a user-defined order. However, since much of the surface modelling involves quadratic fitting, there is a danger of producing circular evidence. That is, there may be a tendency for quadratic models to fit other polynomial surfaces more effectively than, for example, trigonometric or 'real' surfaces.

Fractal surface generation was adopted as a more realistic but controllable model of topographic variation. By changing the fractal dimension of the surface, the degree of spatial structure can be varied. There are numerous methods of generating fractal surfaces (see section 2.5), but the one adopted here uses the spectral synthesis approach described by Saupe (1988), pp.105-109. This technique involves selecting scaled (Gaussian) random Fourier coefficients and performing the inverse Fourier transform. It has the advantage over the more common midpoint displacement methods which produce characteristic artifacts at distances 2^n units away from a local origin (Voss, 1988). More importantly for this work, this technique has been modified so that multiple surfaces may be realised with only selected Fourier coefficients (see *r.frac.surf* in the Appendix). The result is that the scale of fractal behaviour may be controlled as well as the fractal dimension itself. Figure 6.1 shows a fractal surface of dimension D=2.10 rendered with (*a*) 1/8th, (*b*) 1/4, (*c*) 1/2, and (*d*) all of the Fourier coefficients transformed.

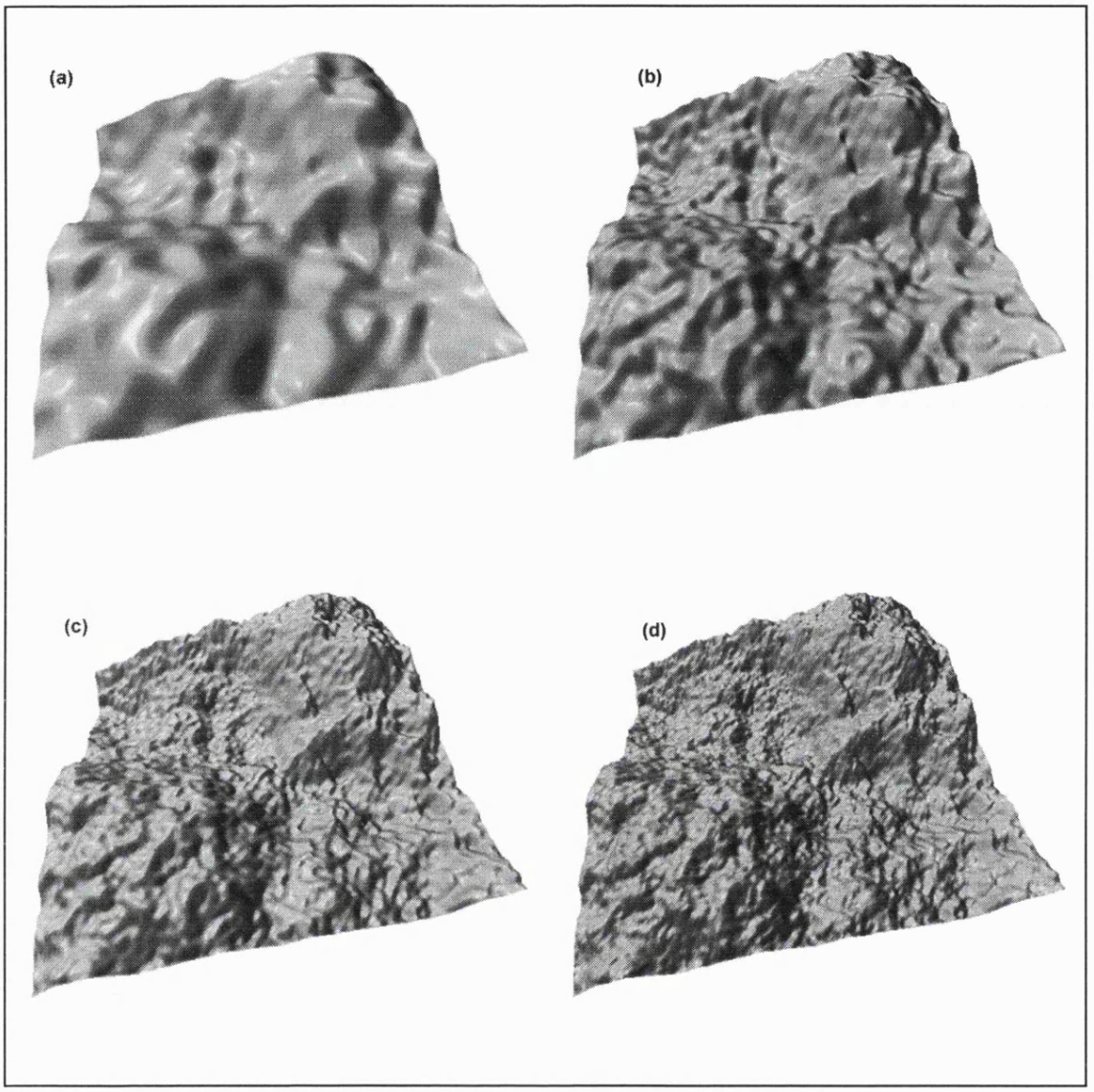


Figure 6.1 - Fractal surfaces created using Spectral Synthesis. Each image shows partial Fourier transformation of *(a)* 1/8, *(b)* 1/4, *(c)* 1/2, and *(d)* all, of the scaled Gaussian coefficients.

6.2 Terrain Models

The DEMs representing real terrain were all selected from the Ordnance Survey 1:50k coverage of the UK. Three areas for study were selected that show contrasting topographic variation. It should be remembered that the objective of this study is not to provide a comprehensive characterisation of the UK landscape, but to demonstrate the utility of the characterisation tools themselves.

6.2.1 Lake District

The following Ordnance Survey DEM tiles were used (with National Grid coordinates indicating extent of the coverage),

(360000, 540000)

NY02	**NY22**	**NY42**
NY00	**NY20**	**NY40**
SD08	**SD28**	**SD48**

(300000, 480000)

Each tile consists of 401 x 401 cells of size 50m x 50m, giving a 3600km^2 coverage of the major part of the Lake District. Various 'sub-windows' were selected for the evaluation below (and previous chapters).

The area represented is a mountainous previously glaciated environment ranging in altitude from sea level to 977m (England's highest mountain). The central upland region (NY20) is dominated by Borrowdale Volcanics, with Skiddaw Slates to the north (NY22) and Silurian shales and sandstones to the south (SD28).

6.2.2 Dartmoor

The following Ordnance Survey tiles covering central Dartmoor were selected,

(280000, 100000)

SX48	**SX68**
SX46	**SX66**

(240000, 060000)

The area is entirely within the surface outcrop of the Dartmoor granitic intrusion giving rise to a characteristic upland 'moor and tor' topography. The extent of the selected DEM is shown

in Figure 6.2 (15 x 14 km).

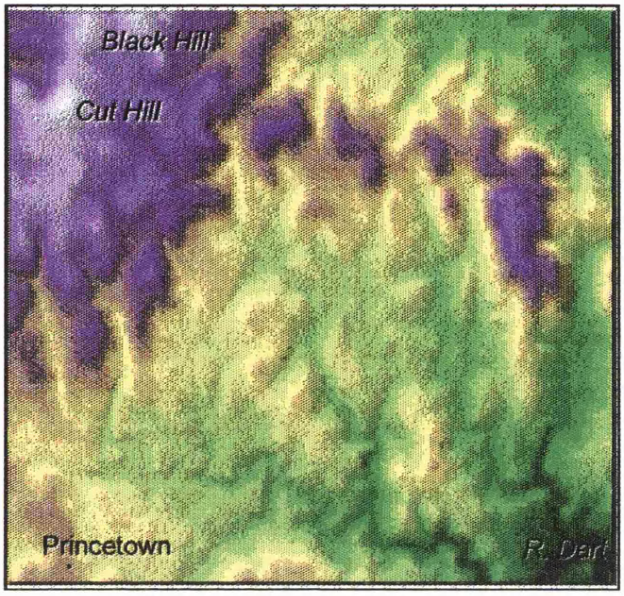


Figure 6.2 Dartmoor study area.

6.2.3 Peak District

The third area selected for study covers a wider 40 x 41km area around the south of the Peak District (40 x 41 km). The Derwent valley runs north-south across the map from the moorland in the High Peak area to the south of the region near Matlock (Figure 6.3). The geology is dominated by Carboniferous Limestone to the west of the Derwent and Millstone Grit to the east and north.

(440000, 400000)

SK08	**SK28**
SK06	**SK26**
SK04	**SK24**

(400000, 340000)

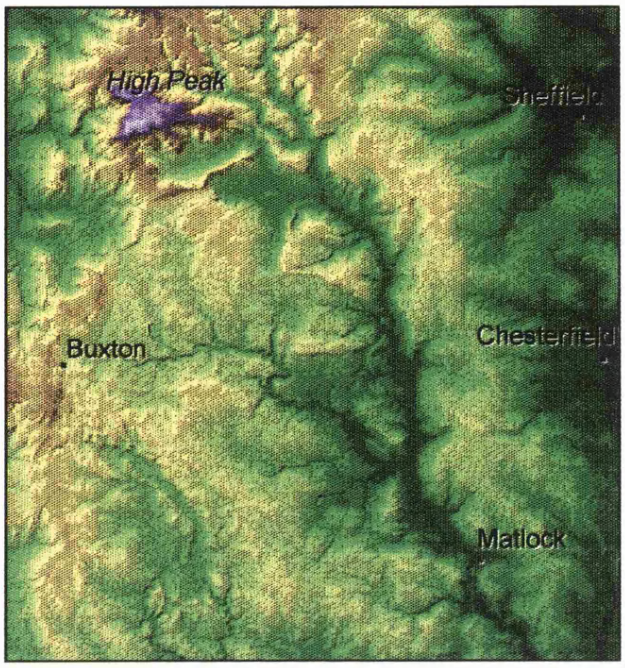


Figure 6.3 - Peak District study area.

6.3 Assessment of Quadratic Approximation Tools

6.3.1 Non-spatial analysis of residuals.

Even at the 3 x 3 kernel size, a quadratic function cannot precisely model the 9 values used in its construction. The question arises therefore, does the approximation lose vital surface information, or worse, does it introduce systematic artifacts into the surface model?

To test this question, the pattern of residuals between the quadratic parameter *f* (elevation at centre of the kernel) and the measured elevation from the DEM was examined. Three surfaces were used for analysis, each with a different degree of spatial structure. Two fractal surfaces of 200 x 200 cells were created, one with a low fractal dimension (D=2.01), the other with a high fractal dimension (D=2.9). A third uncorrelated Gaussian surface was created for comparison. Summary statistics for all three layers are shown in Table 6. 1.

	number of cells	mean	std. dev.
fractal (d=2.01)	40000	-47.32	859.44
fractal (d=2.90)	40000	-152.59	1899.20
Gaussian	40000	1.47	1002.13

Table 6.1 - Summary statistics of quadratic control surfaces.

Table 6.2 shows the mean and standard deviation of the residuals for various kernel sizes. As expected, the standard deviation increases with larger kernels (where the quadratic function becomes increasingly overspecified). The nature of the relationship between residuals and kernel size is shown in Figures 6.2 and 6.3.

	fractal d=2.01		fractal, d=2.90		Gaussian	
Kernel size	***mean***	***stdev***	***mean***	***stdev***	***mean***	***stdev***
3 x 3	0.001	4.11	0.007	302.45	-0.013	667.31
5 x 5	0.008	10.51	0.104	563.99	-0.006	922.67
7 x 7	0.017	16.02	0.478	688.82	-0.170	964.23
9 x 9	0.042	21.35	0.760	768.50	0.287	981.03
11 x 11	0.064	26.50	0.815	825.24	0.329	987.91
13 x 13	0.079	31.49	0.914	870.59	0.371	992.57
15 x 15	0.097	36.42	0.904	910.88	0.242	995.75
17 x 17	0.113	41.29	0.756	945.56	0.150	997.72
19 x 19	0.134	46.09	0.499	975.92	0.129	999.30
21 x 21	0.156	50.79	-0.118	1003.27	0.145	999.77
23 x 23	0.172	55.37	-1.026	1028.13	0.092	1000.50
25 x 25	0.182	59.80	-2.097	1051.89	0.129	1000.92

Table 6.2 - Summary statistics for quadratic approximation residuals for various kernel sizes.

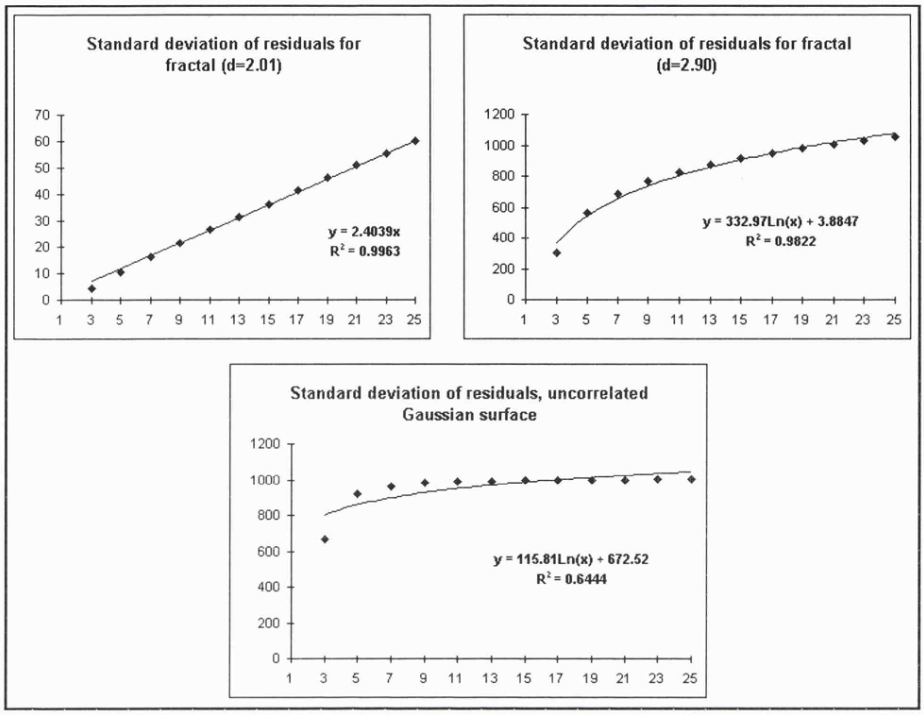


Figure 6.2 - Kernel - residual size relationships for three control surfaces

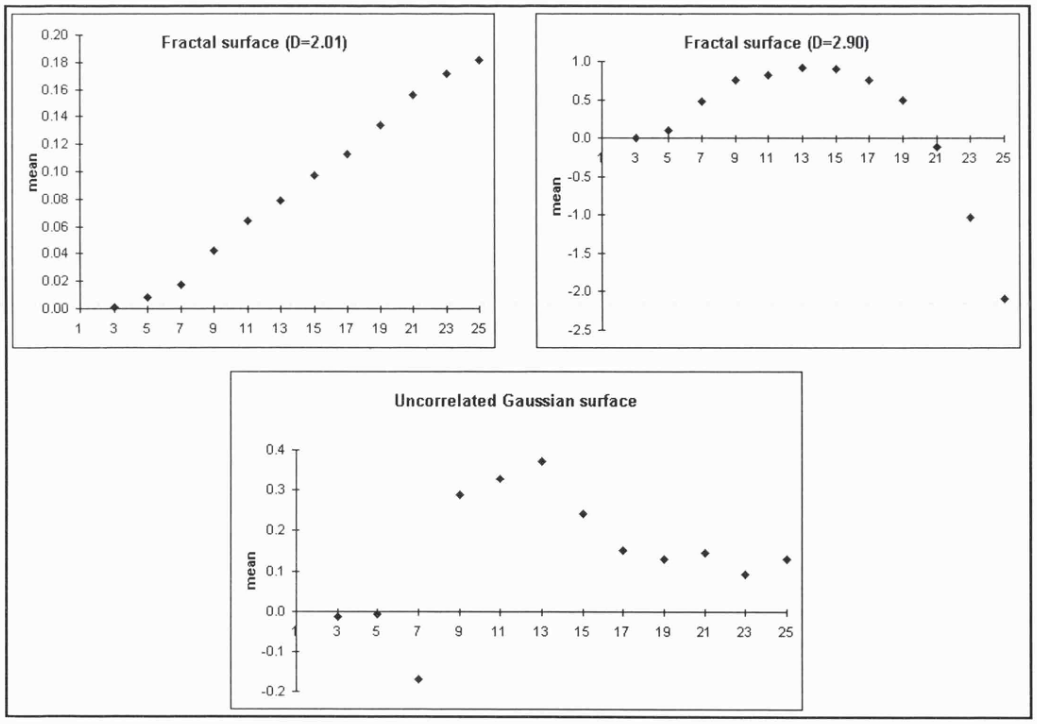


Figure 6.3 - Kernel size - mean residual relationships for three control surfaces.

For the surfaces with the greatest spatial autocorrelation (fractal D=2.01), there appears a well defined (R^2=0.996) linear relationship between kernel size and the magnitude of the residual between quadratic model and original elevation. This suggests that the degree of approximation of 'smooth' surfaces can be controlled entirely by kernel size. As the surface becomes 'rougher' there appears a well defined (R^2=0.902) log-linear relationship between residuals and kernel size. The transition from linear to log-linear relationship suggests that for rougher surfaces there is a degree of 'diminishing returns' with increasing spatial approximation. That is, most approximation occurs at smaller spatial scales with little further generalisation introduced by larger kernels. This relationship is exemplified by the case of the uncorrelated surface where there is little control over residual magnitude beyond a kernel size of 7 x 7 cells.

The relationship between kernel size and residual mean (Figure 6.3) should indicate any systematic over- or under- estimates of elevation. The magnitude of maximum residual mean is less than 0.5% of the residual standard deviation, and so in practice, will have very little systematic effect on elevation estimates. It is worth noting however, that the degree of over- or under-estimation of elevation appears to be dependent the topographic form as well as kernel size. For example, a DEM representing a single valley system will tend to produce consistently higher overestimates of elevation as kernel size is increased. Likewise, a tendency to regress towards a mean elevation will produce increasingly large underestimates of elevation for a DEM representing a single peak.

6.3.2 Spatial analysis of residuals

As with the discussion of DEM uncertainty (see Chapter 3), global measures of elevation variation are not sufficient to understand the pattern of residuals, since they are likely to vary over space. Figure 6.4 shows selected 'residual maps' for the quadratic approximation of the two fractal surfaces at different scales.

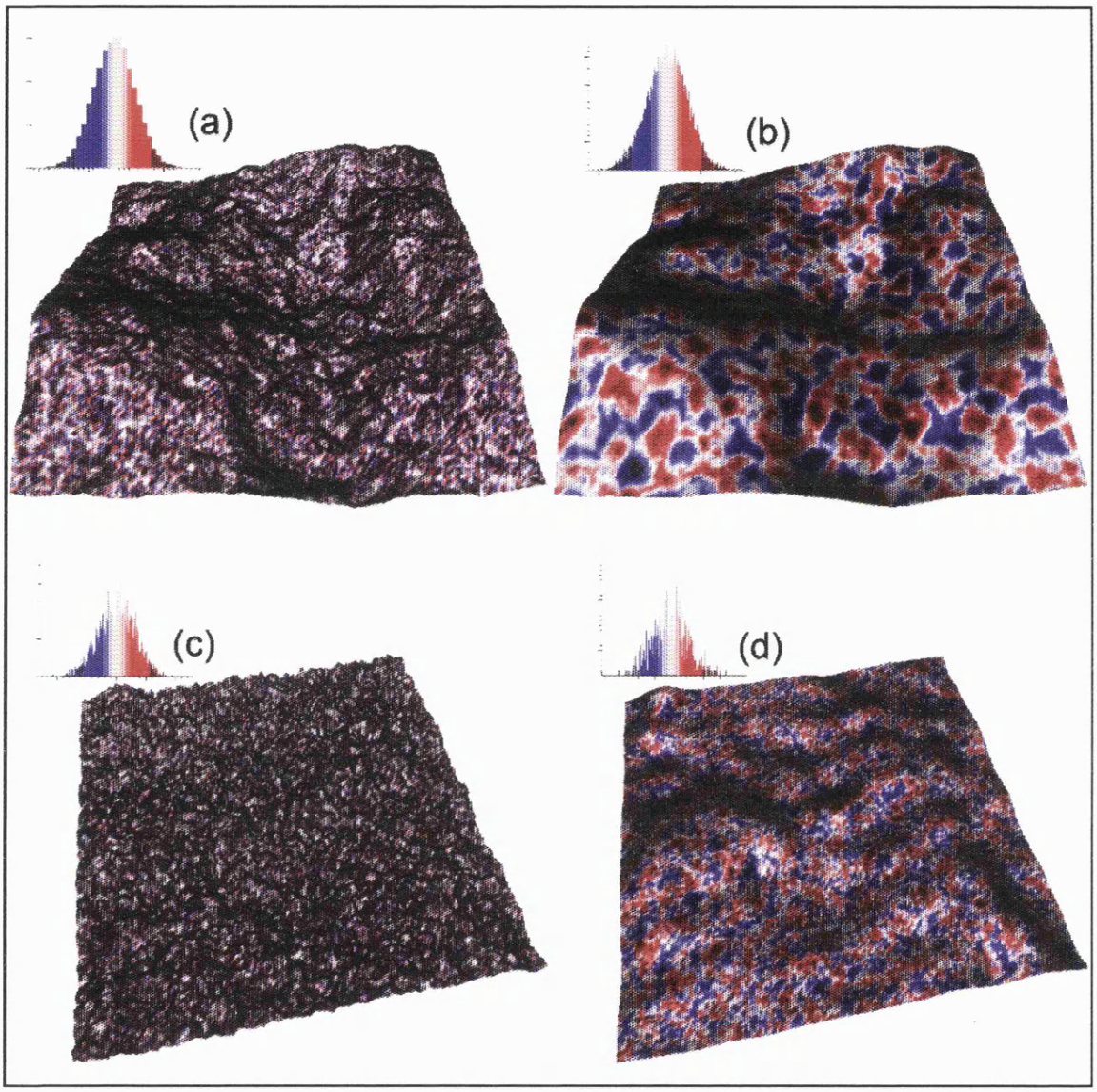


Figure 6.4 - Residual maps for four quadratic approximations. Colour scales are normalised so that pure red/blue is 1 standard deviation from mean. (*a*) 3x3 approximation of fractal D=2.01; (*b*) 25x25 approximation of D=2.01; (*c*) 3x3 approximation of fractal D=2.90; (*d*) 25x25 approximation of fractal D=2.90.

All four images in Figure 6.4 show some form of spatial clustering of residuals, that not unexpectedly corresponds with a change in kernel size. To describe the spatial pattern in more systematically, Moran's I lag diagrams (see section 4.5) were visualised. Figure 6.5 shows two selected examples corresponding to residuals produced by 25x25 and 13x13 cell kernels. Red indicates positive autocorrelation, blue negative. The magnitude of the statistic is additionally shown with selected profiles.

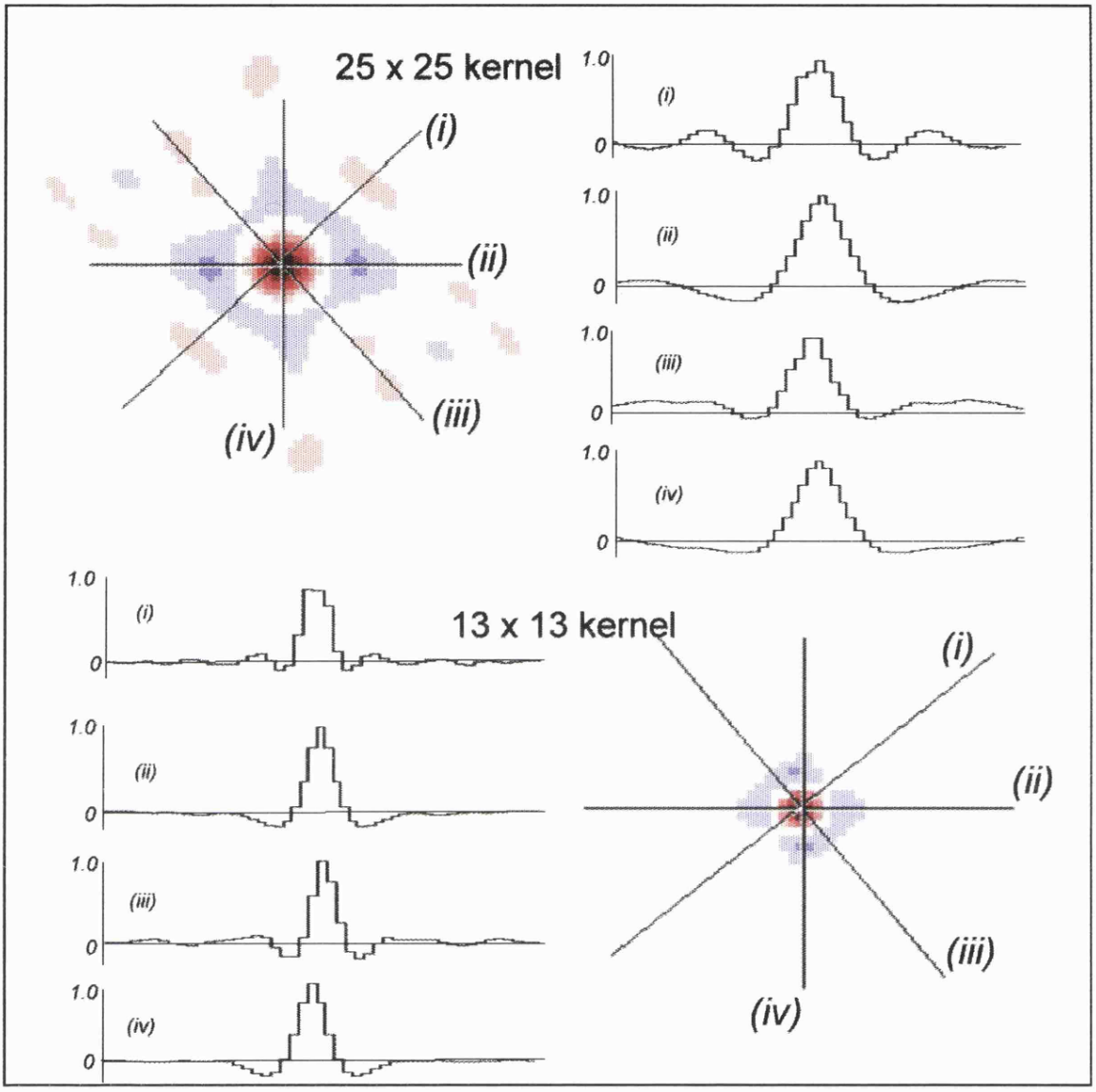


Figure 6.5 - Spatial autocorrelation lag diagrams and profiles for two residual surfaces.

The clustering seen in Figure 6.4 is described by the high spatial autocorrelation at low lags. Spatial autocorrelation is reduced as lag increases, reaching a minimum value before becoming approximately uncorrelated. In both cases, the innermost uncorrelated (white) band corresponds to approximately half the kernel size, while the maximum negative value is at approximately 2/3 the kernel width. There is also a degree of anisotropy shown by the lag diagrams as 'diamond' rather than circular bands of equal autocorrelation. This is due to the square shape of the kernel that was used without any form of distance weighting. These findings suggest that filters that do not have any kind of distance weighting may well have to

be treated with caution, especially when processing surfaces with low spatial autocorrelation at the scale of the filter.

6.3.3 Quadratic approximation as a surface generalisation tool

The effectiveness of quadratic approximation as a generalisation process was examined by comparing the smoothing effect of mean convolution filtering (the most common form of surface smoothing) with that of quadratic approximation.

Figure 6.6 shows the visual effect of quadratic approximation as a generalisation process, applied to the central Lake District (35km x 35km). Kernel sizes range from 7x7 (350m) to 25x25 cells (1.25km).

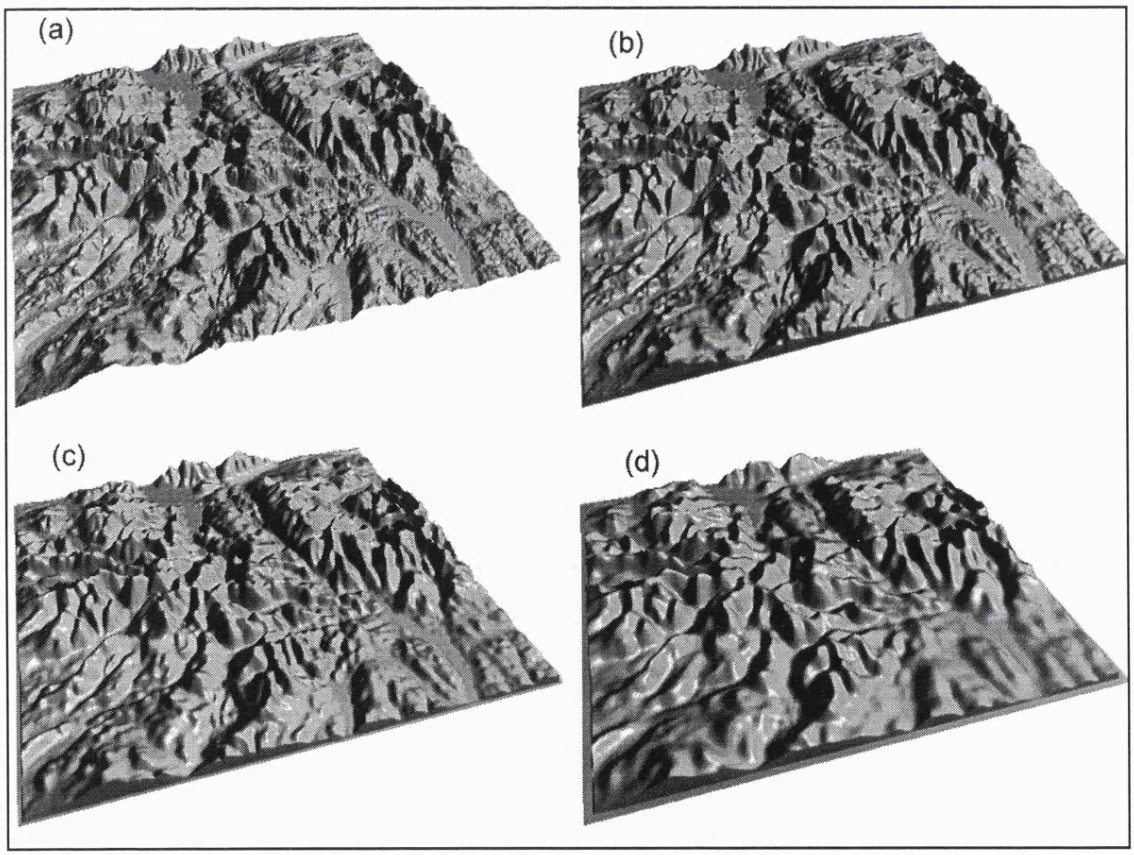


Figure 6.6 - Effect of quadratic approximation on terrain roughness. (*a*) original Lake District DEM; (*b*) 7x7 quadratic approximation kernel; (*c*) 13 x 13 kernel; (*d*) 25 x 25 kernel.

The effect of non-distance weighted quadratic smoothing on the frequency distribution properties of three surfaces was considered. The range and standard deviation of (i) an uncorrelated Gaussian surface, (ii) the Lake District DEM pictured in Figure 6.6; and (iii) the Leicestershire DEM were examined after filtering at all scales from 3x3 to 25x25 cells. Comparison was made with a non-distance weighted mean filter (using the GRASS program *r.mfilter*) at the same range of scales. Figure 6.8*a-f* show the relationship between elevation dispersion and kernel size. Note also, the non-linear smoothing effect of the Gaussian surface that was also observed in section 6.3.1.

As would be expected, the larger the filter, the greater the reduction in range and variance of elevation values. However quadratic smoothing retains the original global dispersion characteristics far more effectively than mean filtering. This is reinforced by visualising the smoothed surfaces. Figure 6.7 shows the south east corner of the Lake District DEM (Windermere/Ambleside) smoothed using both the 25x25 quadratic and mean filters. Quadratic approximation appears to preserve characteristics at the certain scales while smoothing detail at finer scales. Mean filtering would appear far more scale insensitive in its smoothing effects.

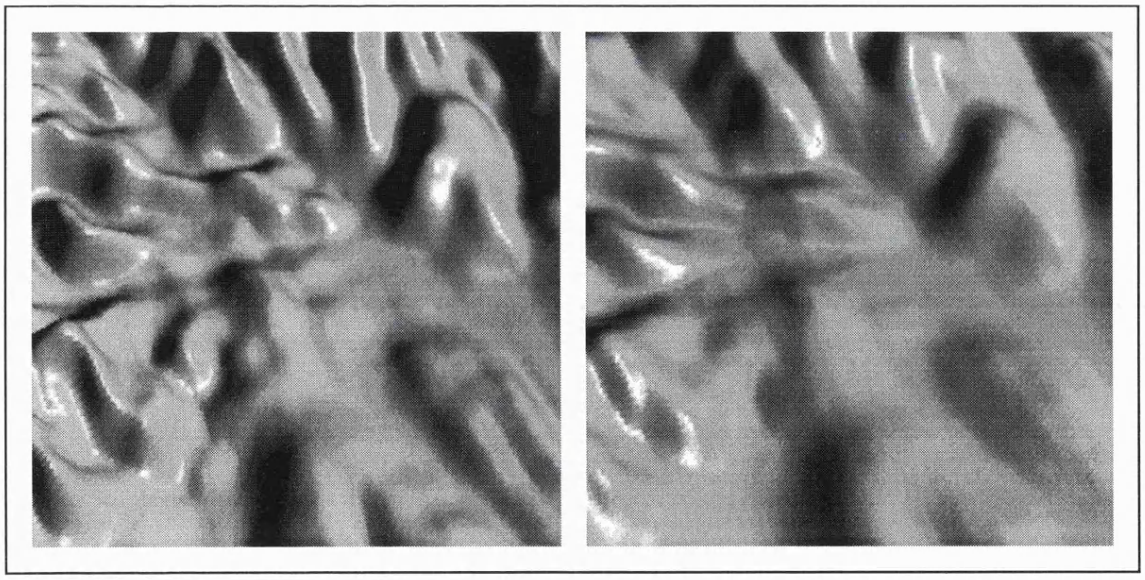
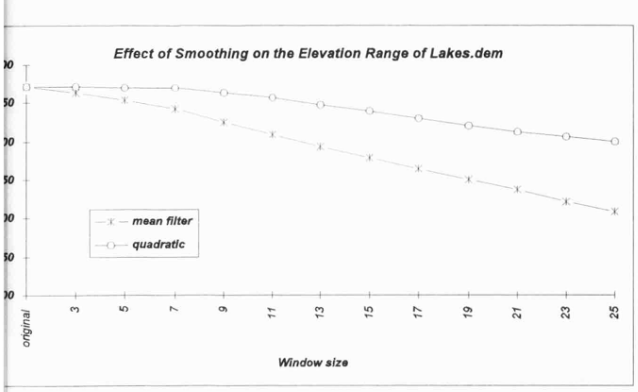


Figure 6.7 - Comparison of (*a*) quadratic approximation and (*b*) mean filtering as a smoothing process.



ure 6.8a

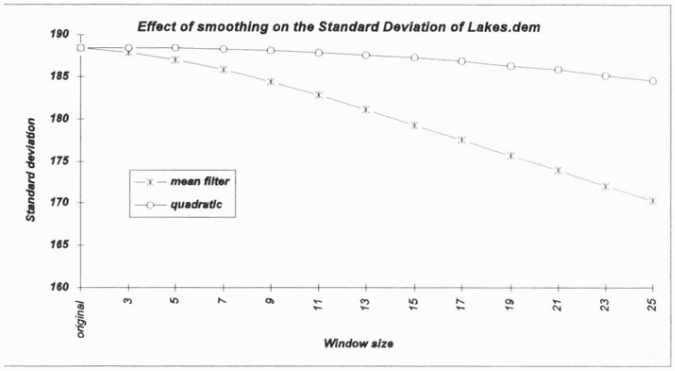
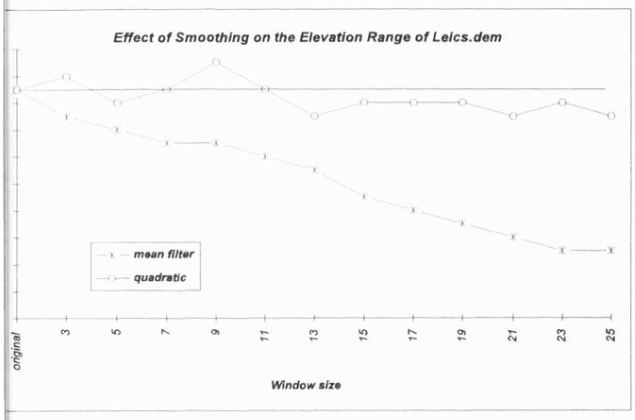


Figure 6.8b



ure 6.8c

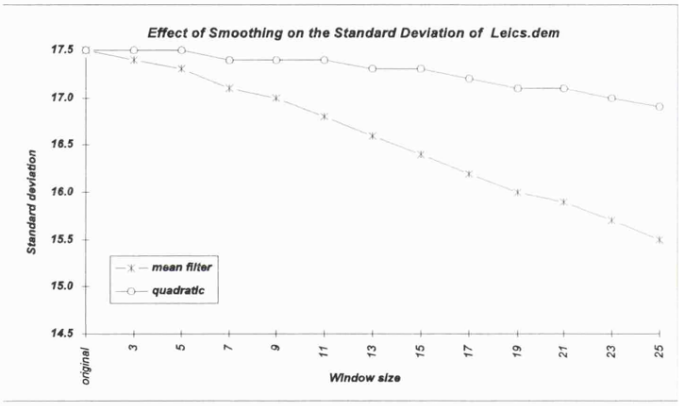
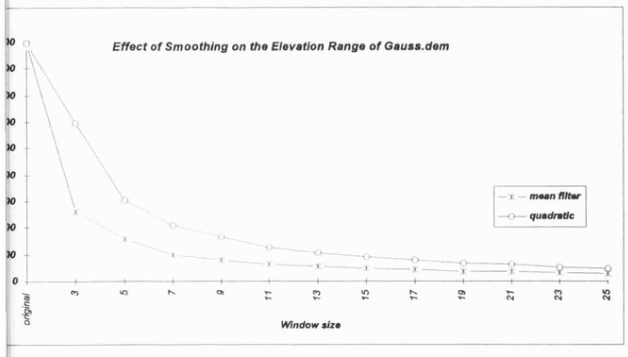


Figure 6.8d



ure 6.8e

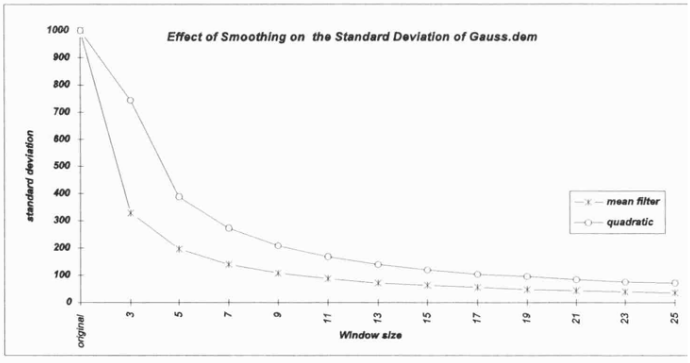


Figure 6.8e

6.4 Calibration of Spatial Dependence Measures.

Section 4.5 described how several tools were developed to visualise and measure the spatial structure of DEMs. This section evaluates their utility by 'calibrating' them with surfaces with known properties.

It was suggested that a lag diagram of Moran's *I* spatial autocorrelation index could reveal something of the relationship between spatial dependence and scale. To evaluate this, multiple realisations of a fractal surface were generated. Eight surfaces of fractal dimension D=2.10 were created, and eight of D=2.90. A lag diagram of Moran's *I* was generated for each surface. Figures 6.10 and 6.11 show the results for surfaces with D=2.10 and D=2.90 respectively. The bi-polar colour scheme used for all Moran diagrams is shown in Figure 6.9.

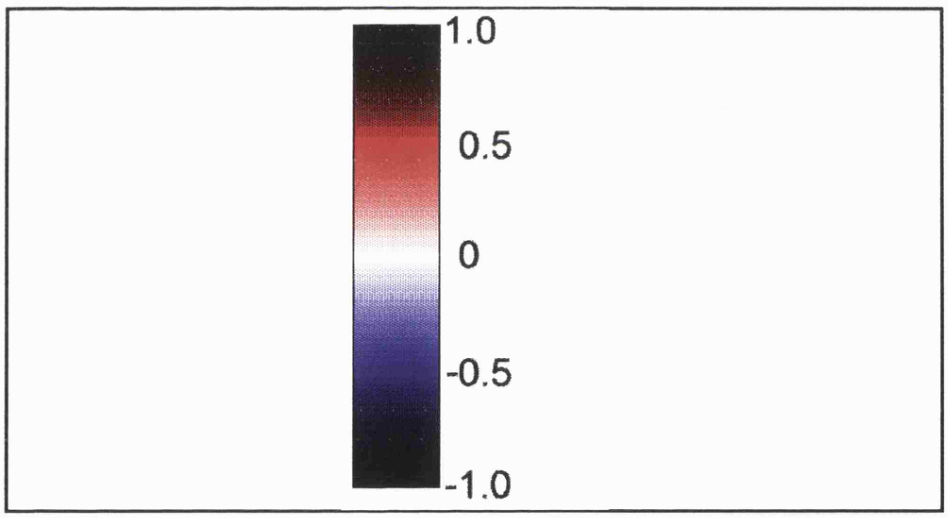


Figure 6.9 - Bi-polar colour scheme used for Moran's I lag diagrams.

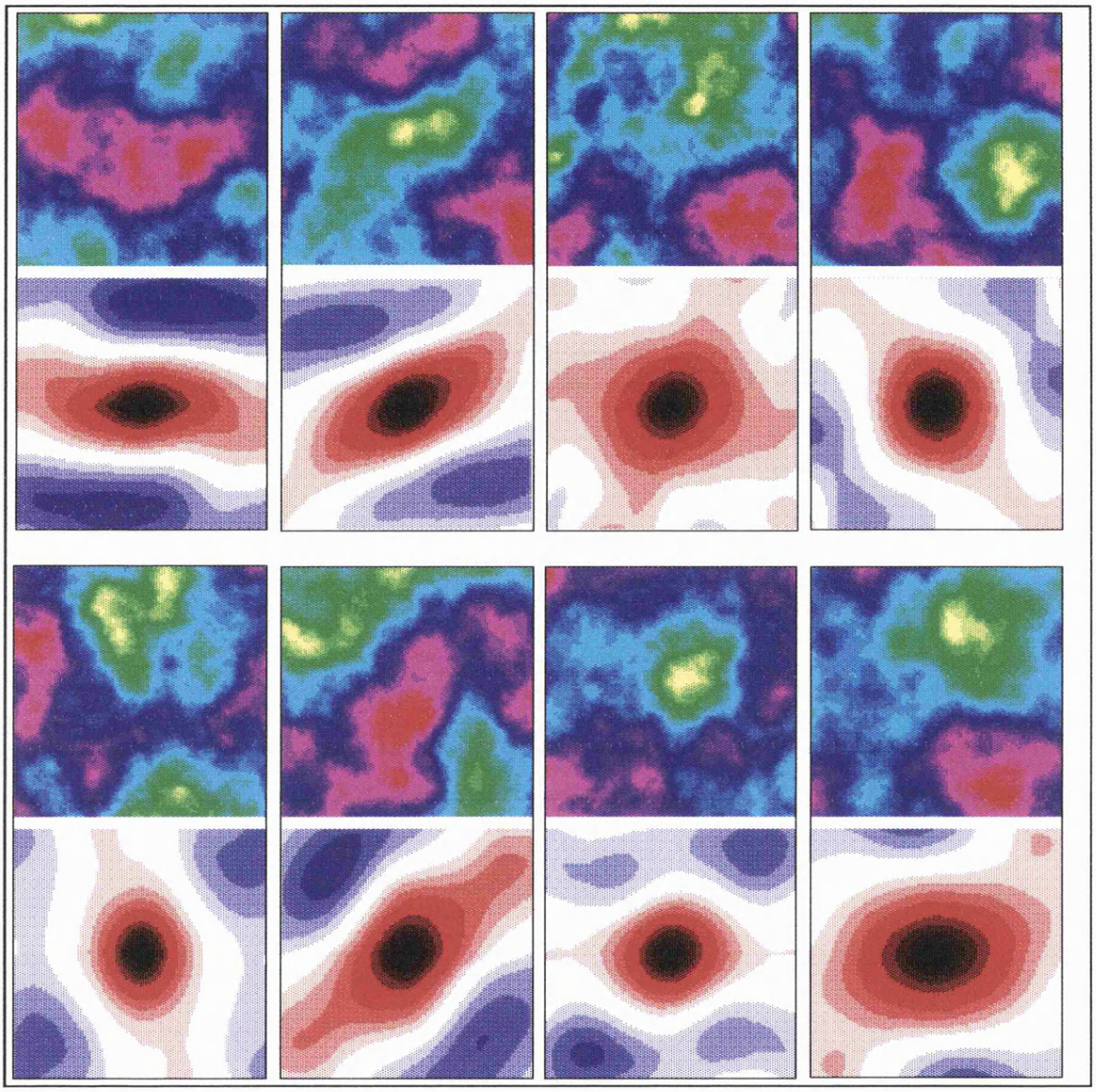


Figure 6.10 - 8 realisations of a fractal surface (D=2.10) with corresponding Moran's I lag diagram shown below.

The notion of a distinction between *texture* and *structure* is a common one in image processing (eg Gonzalez and Woods, 1992) and is useful to consider here. All the surfaces shown in Figure 6.10 have similar (fine scale) textural characteristics (determined by the fractal dimension), but structurally (course scale) they vary. That is, due to the random generation of Fourier coefficients, a surface may describe on a low frequency peak, pit, ridge, channel or combination of features. What distinguishes texture from structure is simply the scale at which variation is considered. All the images in Figure 6.10 show similar fine scale variation, in that the central portions of all lag diagrams are similar. Variation tends to occur away from the centre seen as

concentric zones of equal spatial autocorrelation. The most obvious feature of these larger lag measures, is the ability to represent anisotropy. The orientation of any coarse scale valley or ridge systems is pricked out by elongated bands of equal spatial autocorrelation. In several cases such bands tend to 'twist' with increasing lag suggesting a change of orientation with scale. It should be noted that such variation would not be detected using the traditional (one-dimensional) variogram. Surfaces without a dominant ridge or valley system tend to produce lag diagrams with a greater proportion of positively autocorrelated measurements.

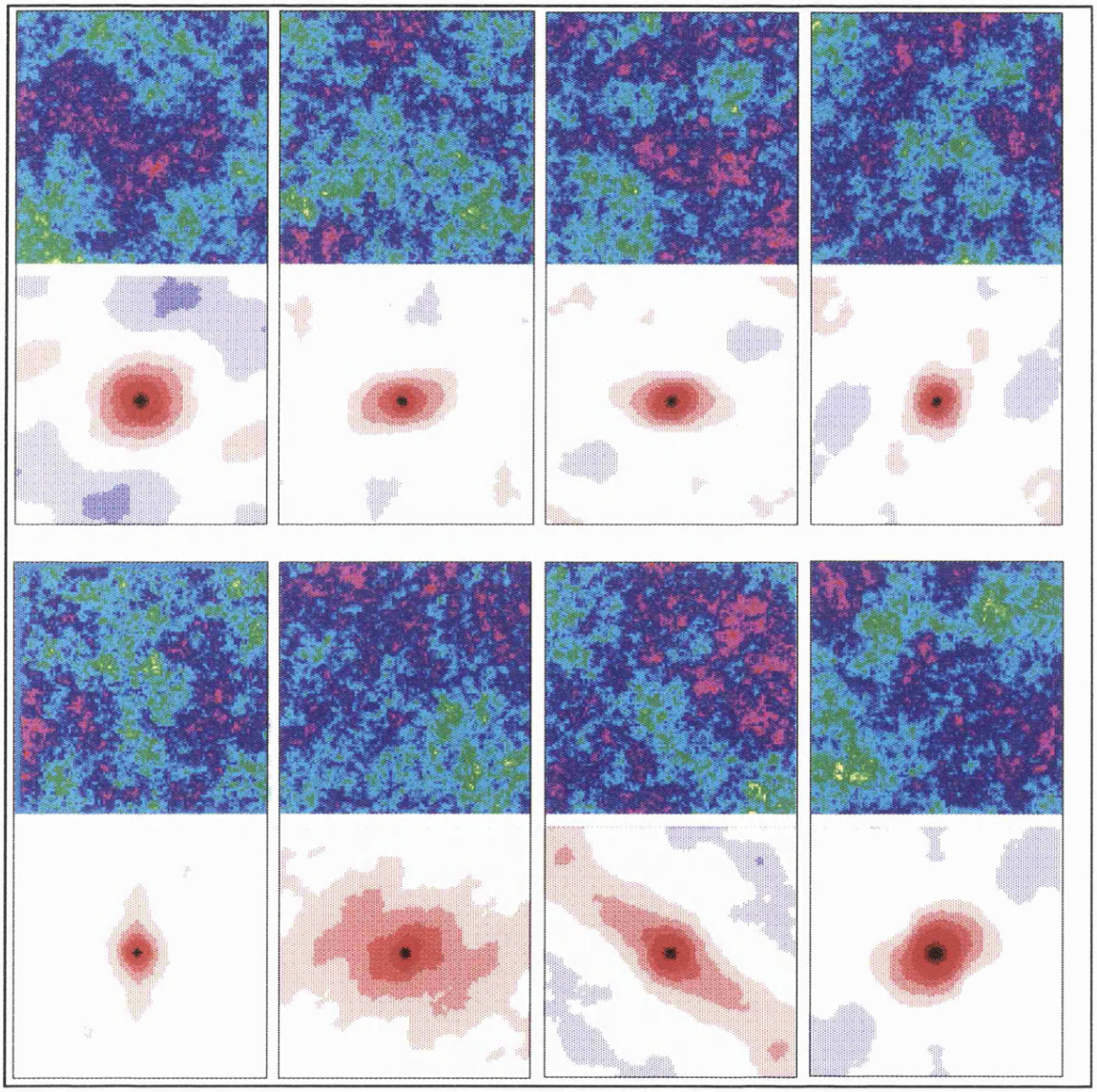


Figure 6.11 - 8 realisations of a fractal surface (D=2.9) with corresponding Moran's I lag diagram shown below.

The eight sets of images shown in Figure 6.11 show a marked contrast with the surfaces of lower

fractal dimension. Generally, the magnitude of either positive or negative autocorrelation is lower (images appear 'paler'). The characteristic diameter of high spatial autocorrelation is much smaller for these rougher surfaces. Coarse scale structure and anisotropy is still revealed, albeit less strongly.

It would appear that Moran's I lag diagrams are useful in detecting structural variation and anisotropy, and the relative dominance of textural and structural components of surface form. Although the terms texture and structure are used, these diagrams show the scale based continuum between the two rather than forcing an artificial dichotomy between them. To investigate the distinction between textural and structural variation further, a second set of fractal surfaces was produced. Using a single set of scaled Gaussian Fourier coefficients, selected coefficients were transformed. The effect was to generate a series of similar surfaces but each with increasingly high frequency detail added (see Figure 6.1 for an example).

Figure 6.12 shows two surfaces from the series, one with 1/8th of the Fourier coefficients transformed, the other with all coefficients. Surface texture is emphasised by combining elevation and local relief as a hue-intensity map. The Moran's *I* lag diagrams for both surfaces are shown below.

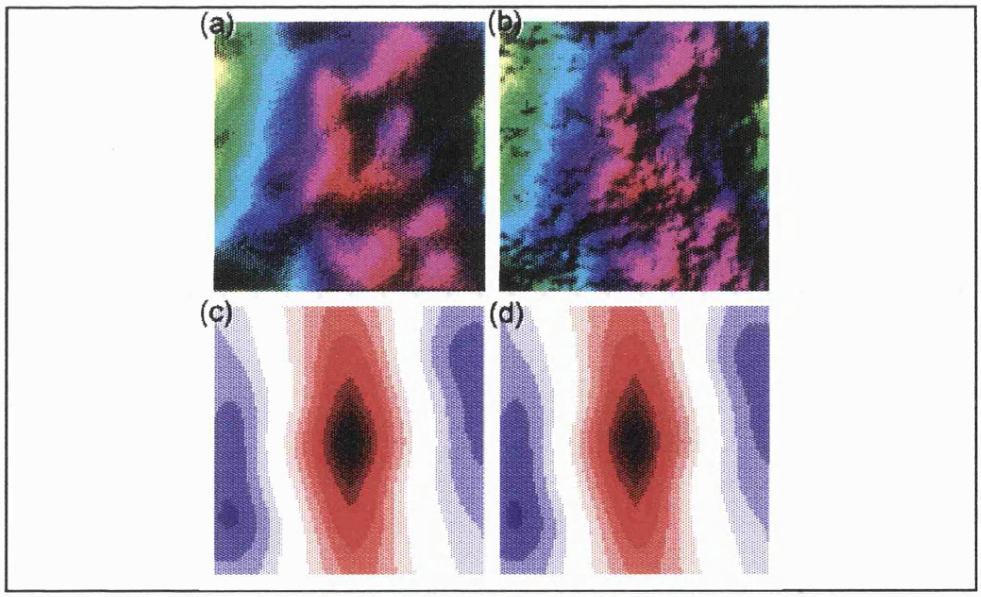


Figure 6.12 - Fractal surface (D=2.1) with (*a*) 1/8 and (*b*) all Fourier coefficients. Their Moran's I lag diagrams are shown below as (*c*) and (*d*).

The two lag diagrams appear very similar, suggesting that they are not suitable visualisations of high frequency textural variation. Indeed, visualising the original surfaces (Figure 6.12*a* and *b*) is much more useful. This demonstrates that such lag diagrams are dominated by, and therefore most suitable for examining, structural variation. The change in texture between the two surfaces occurs only in the immediate (dark) central portion of each image.

To examine finer scale textural variation , co-occurrence textural measures were investigated (see also section 4.5.2). The co-occurrence matrices for the two surfaces shown in Figure 6.12 were calculated and textural measures calculated from them. A visualisation of the two matrices is shown in Figure 6.13 and the derived texture measures are recorded in Table 6.3.

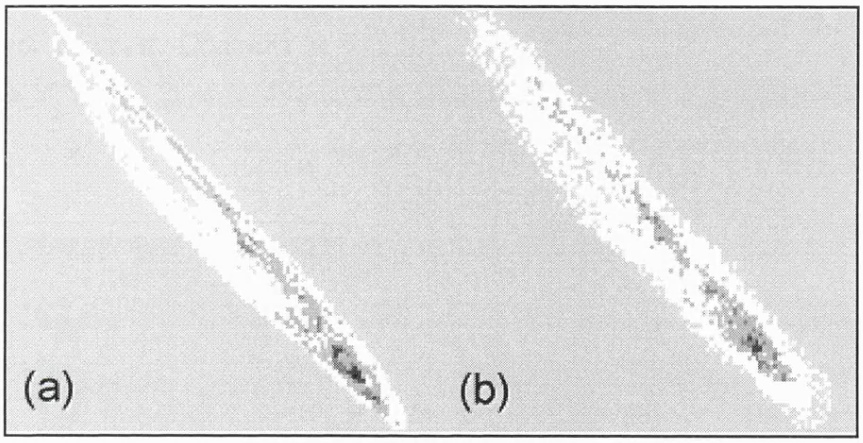


Figure 6.13 - Two co-occurrence matrices for fractal surface with (*a*) 1/8th, and (*b*) all Fourier coefficients transformed. Elevation is quantized into 100 categories, lag is one raster unit.

Texture Measure	**fractal (1/8th coeff.)**	**fractal (all coeff.)**
Contrast	22.85	26.03
Angular Second Moment	0.0012	0.0010
Entropy	6.99	7.15
Asymmetry	0.00071	0.00042
Inverse Distance Moment	0.064	0.068

Table 6.3 - Selected texture measures for the two fractal surfaces.

The global textural measures do appear to discriminate between the two surfaces although it is not clear which distinct texture properties each measure. The co-occurrence matrix itself also appears to vary between the surfaces with the rougher surface producing the more disperse matrix. However, it should be considered that both the matrix and the derived indices are measured at a single lag (unit offset in this case). It would be expected therefore, that they should show different patterns at different scales.

To determine any scale dependency in textural characteristics, lag diagrams were calculated for two of the measures - contrast and inverse distance moment. The others were excluded either because they are scaled variations of the same property or because they showed no scale dependency. Figure 6.14 shows the contrast lag diagram for the two surfaces. Darker shading indicates lower contrast. Contours at 500 unit intervals are included to emphasise surface variation.

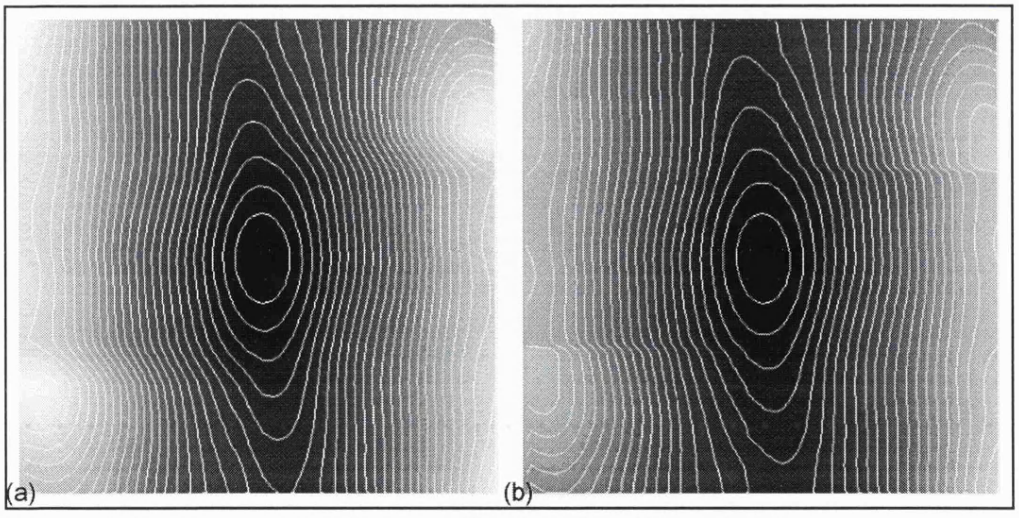


Figure 6.14 - Contrast lag diagrams for fractal surfaces with (*a*) 1/8th and (*b*) all Fourier coefficients.

Broadly, these two images appear similar, as they suffer from the same problem as Moran's *I* lag diagrams as representations of high frequency texture. They do however demonstrate the high degree of scale dependence of the measure, which ranges from order 10 at the lowest lags to order 10,000 at the largest lags. To investigate differences in texture the central portion of

the two images was enlarged and the percentage difference in contrast between the two was calculated (something that is easily accomplished in a raster GIS environment). Figure 6.15*a* shows the scaled difference between the contrast values of the fractal surface and its smoother equivalent with 1/8th of the Fourier coefficients. Figure 6.15*b* shows a similar difference map between the fractal and is equivalent surfaces with 1/4 of the Fourier coefficients. Both images show lags of up to 12 raster cells (17 along diagonals).

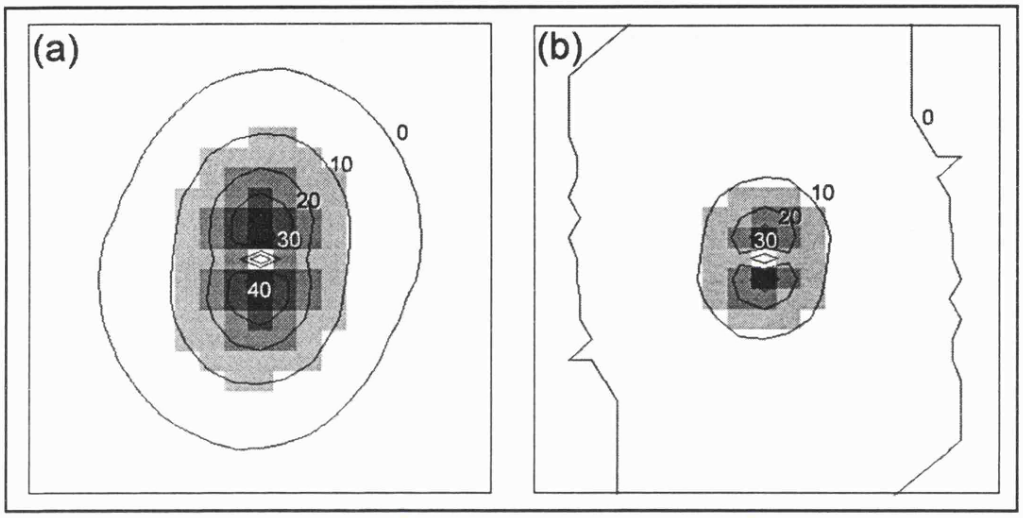


Figure 6.15 - Percentage difference in contrast measures between (*a*) fractal and fractal with 1/8th coefficients, and (*b*) fractal and fractal with 1/4 coefficients.

The difference map shows that proportionally, the greatest difference in texture occurs at the highest frequency (as would be expected). More significantly, a spatial resolution can be identified at which this difference becomes significant. A 10% difference occurs at a lag of approximately 4 raster units in Figure 6.15*a* and 2 raster units in Figure 6.15*b*. This corresponds with the doubling of Fourier coefficients between the two surfaces.

A similar proportional difference map was produced for the Inverse Distance Moment measure and is shown in Figure 6.16. The spatial distribution of difference appears contrasts with Figure 6.15, and does not appear to relate to scale in an obvious way. While demonstrating that the Inverse Distance Moment measures different texture properties to Contrast, it is not clear how its meaning should be interpreted.

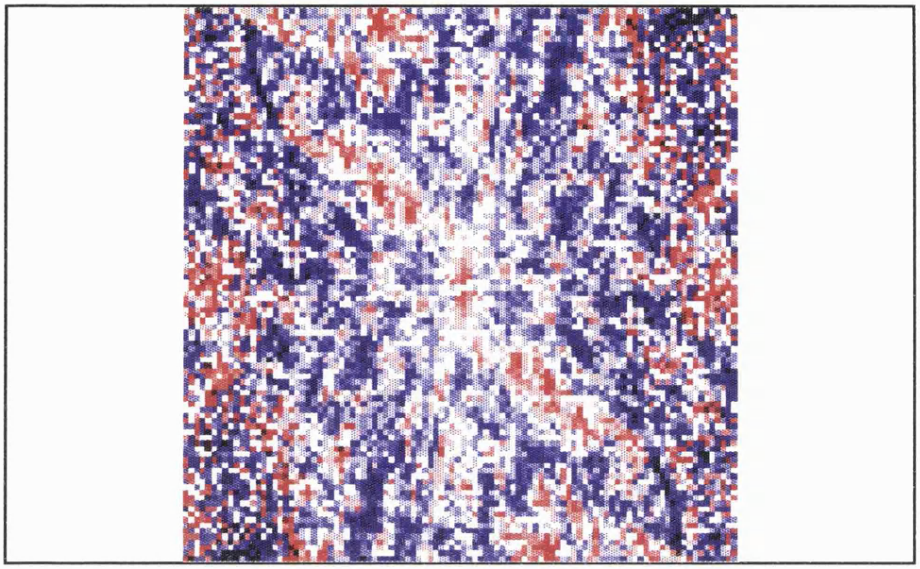


Figure 6.15 - Scaled differences between the Inverse Distance Moment lag maps of fractal and fractal with 1/8th transformed coefficients.

6.5 Terrain Characterisation.

This section describes how the multi-scale characterisation techniques developed can applied to DEMs representing 'real' terrain. The aim is not to provide a detailed geomorphological analysis, but to demonstrate how, with visual examples, these tools could be used as a mechanism for such analysis. In particular, it is the aim of this section to demonstrate that a multi-scaled characterisation reveals more (useful) information that traditional raster-based geomorphometric analysis.

Figure 6.16 shows the cross-sectional curvature derived from the Lake District DEM at kernel scales of 150m, 450m, 850m and 1.25km. The area represented in the figure is approximately 8 x 8 km with the Wasdale valley in the southwest and the Ennerdale valley running northwest - southeast along the north of the image.

It is clear that the pattern of local curvature varies considerably with scale. At the finest (150m) scale, a rather fragmented network of ridges and channels is revealed. Smaller, well dissected upland valley systems are relatively well defined, while the major valleys are not delineated. Much of the surface is represented as poorly autocorrelated minor concavities and convexities.

At a coarser scale, there is far greater spatial autocorrelation in measurement. Figure 6.16*d* for example shows an almost entirely connected valley network. This suggests that either the true surface varies in roughness with scale (ie not self-similar), or that there is a (random) noise or error component at the highest frequency modelled by the DEM.

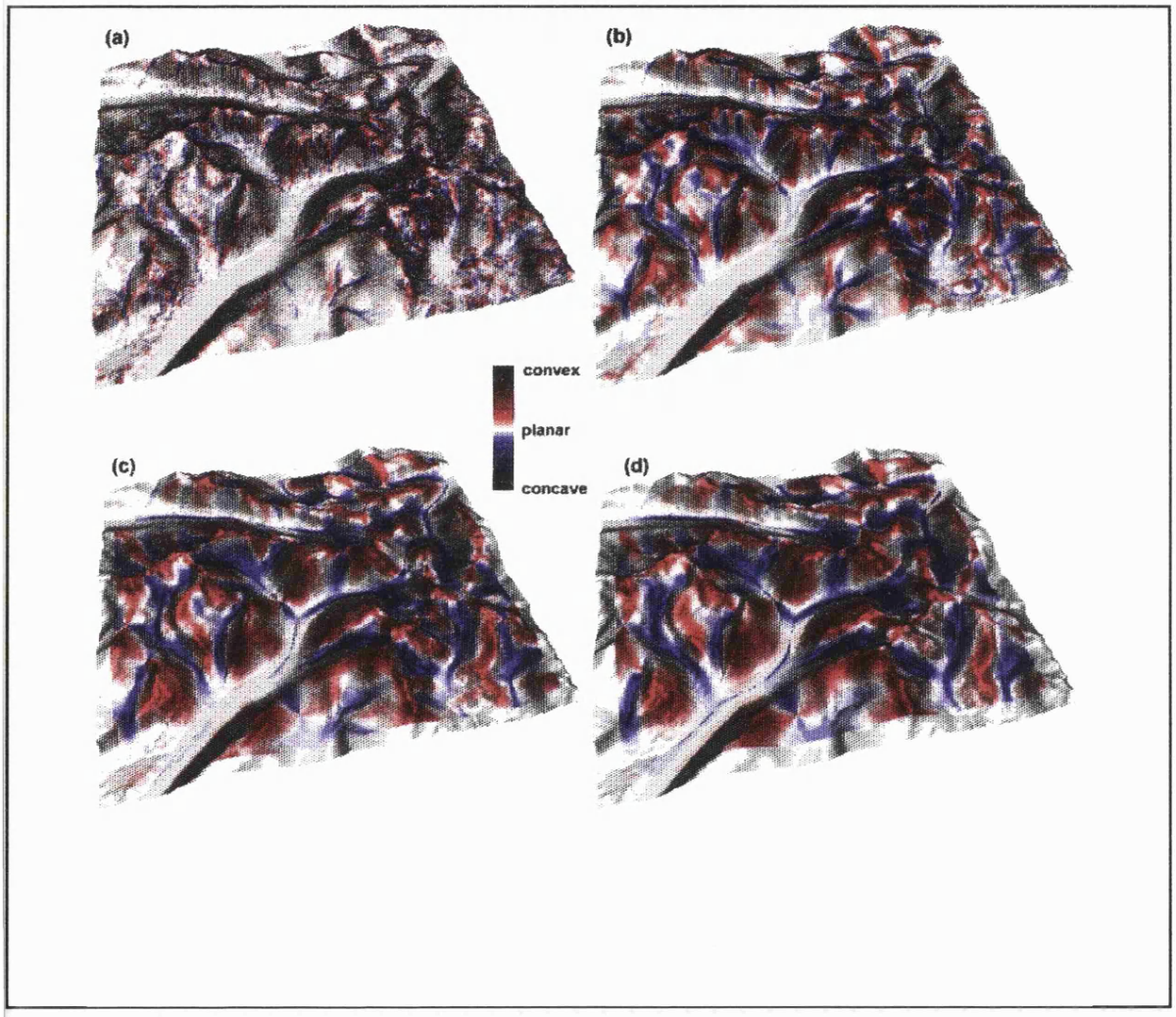


Figure 6.16 - Cross-sectional curvature maps of selected portion of the Lake District DEM. Kernel size set to (*a*) 3 x 3 (150m); (*b*) 9 x 9 (450m); (*c*) 17 x 17 (850m); and (*d*) 25 x 25 (1.25km).

The pattern of changing curvature with scale is not symmetrical for concavity and convexity measurements. The 'characteristic scale' beyond which networks are well connected appears to be finer for convexities than for concavities. At the 450m scale (Figure6.16*b*), ridges appear well defined while channels systems are represented discontinuously. At the 1.25km scale (Figure

6.16*d*) channels are still broadly linear features, while surface convexities are in many cases 'spread' over peak features. This asymmetry in pattern is itself a useful diagnostic feature. It is indicative of a glaciated mountain environment previously dominated by erosional processes around ridges and depositional processes along major valley systems.

The animated sequence of changing cross-sectional curvature with scale from which the four images in Figure 6.16 were taken, demonstrates the inadequacy of traditional (single scaled) raster based measurements. A large valley systems such as Wasdale (southwest of the image) is clearly of some geomorphological importance, yet would not be 'detected' by raster processing at the DEM resolution of 50m. It would seem an unnecessary constraint to consider geomorphological surface variation at one scale alone if a single DEM can reveal pattern over a range of scales.

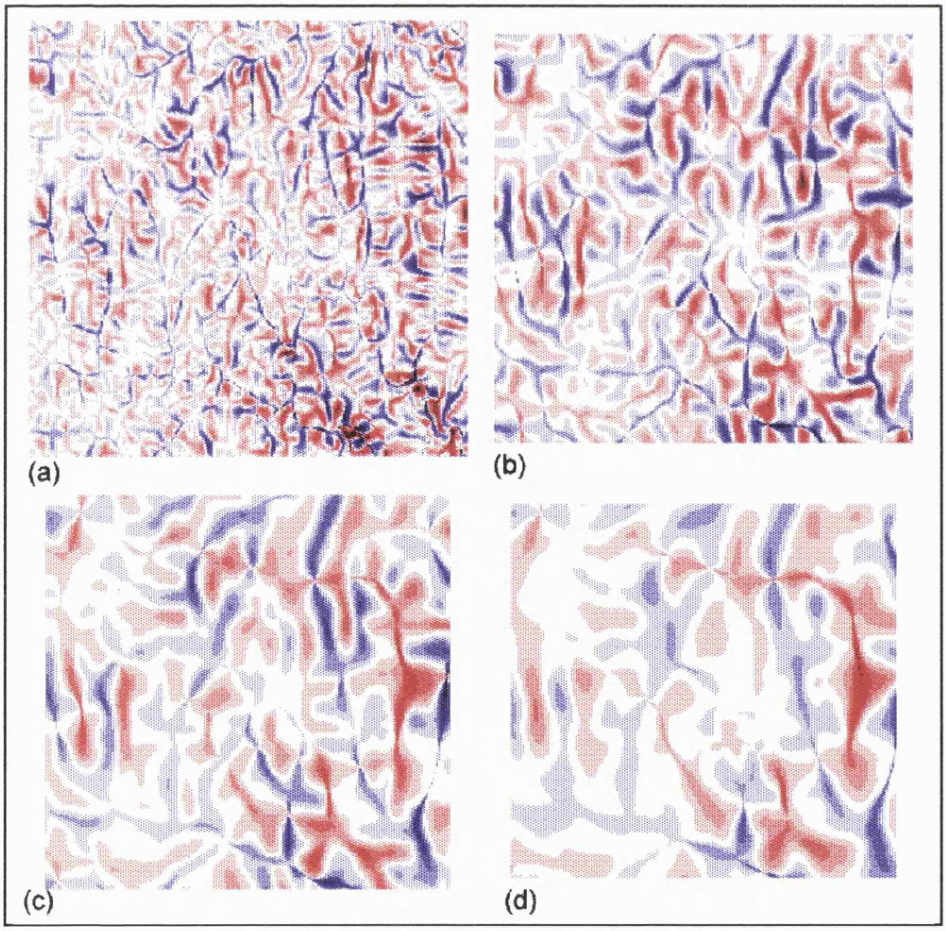


Figure 6.17 - Cross-sectional curvature for Dartmoor DEM. (*a*) 9x9 kernel; (*b*) 21 x 21 kernel; (*c*) 33 x 33 kernel; (*d*) 45 x 35 kernel.

Figure 6.17 shows a similar set of cross-sectional curvature values for the Dartmoor region. In this case the kernel size ranges up to 45 x 45 cells (2.3km) over an area of 14km x 14km. As with the Lake District example, the network pattern varies considerably with scale. Again, as with the Lake District, the change in curvature with scale appears asymmetric, but this time, channels are more well defined at finer scales than ridges. It also appears that there is greater spatial variation in convexity with scale than with concavity. This implies the existence of well defined 'V-shaped' channels that are expressed over a range of scales.

The degree to which surface measurements vary with scale was investigated interactively using *d.param.scale* (see Appendix). An indication of scale dependency in morphometric feature classification is shown in Figure 6.18.

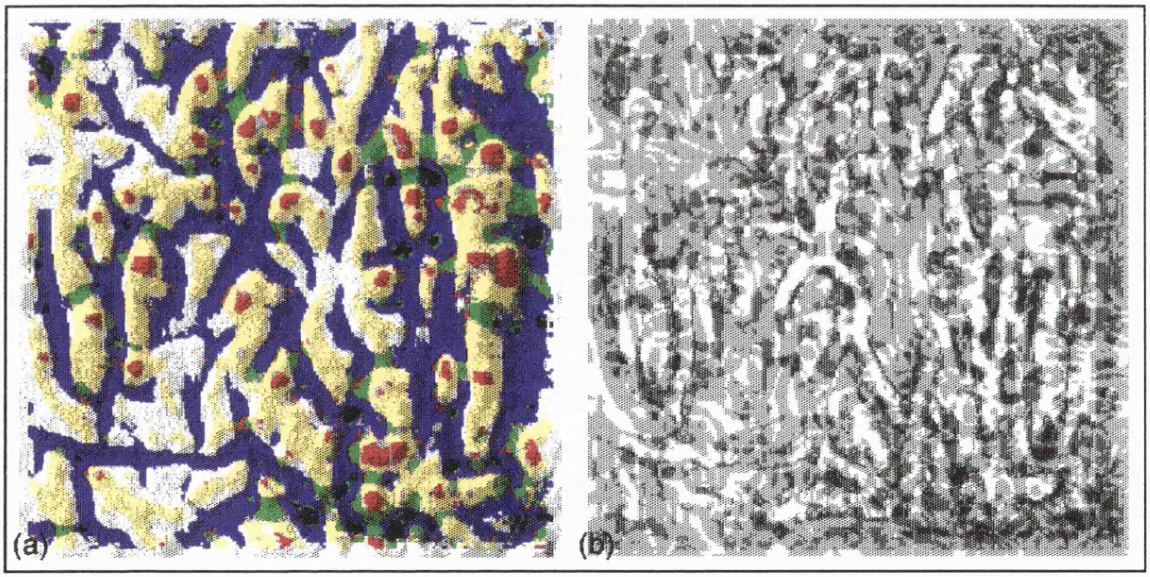


Figure 6.18 - Dartmoor modal feature classification. (*a)* morphometric features (pit -black, channel-blue, pass-green, ridge-yellow, peak-red, multimodal-grey); (*b*) classification entropy (dark = high entropy).

Figure 6.18*a* displays the most frequent feature classification of each DEM cell based on kernel sizes ranging from 9 x 9 to 45 x 45 cells. Where a cell has more than one mode, it is coloured grey. The relationship with relief is emphasised by combing this classification with shaded relief using an hue-intensity mapping. Figure 6.18*b* shows the confidence with which classification can be made by showing the scaled entropy. The lighter the image, the greater the consistency

of feature classification (see section 5.3). These may be combined in a single image using an hue-intensity mapping (Figure 6.19)

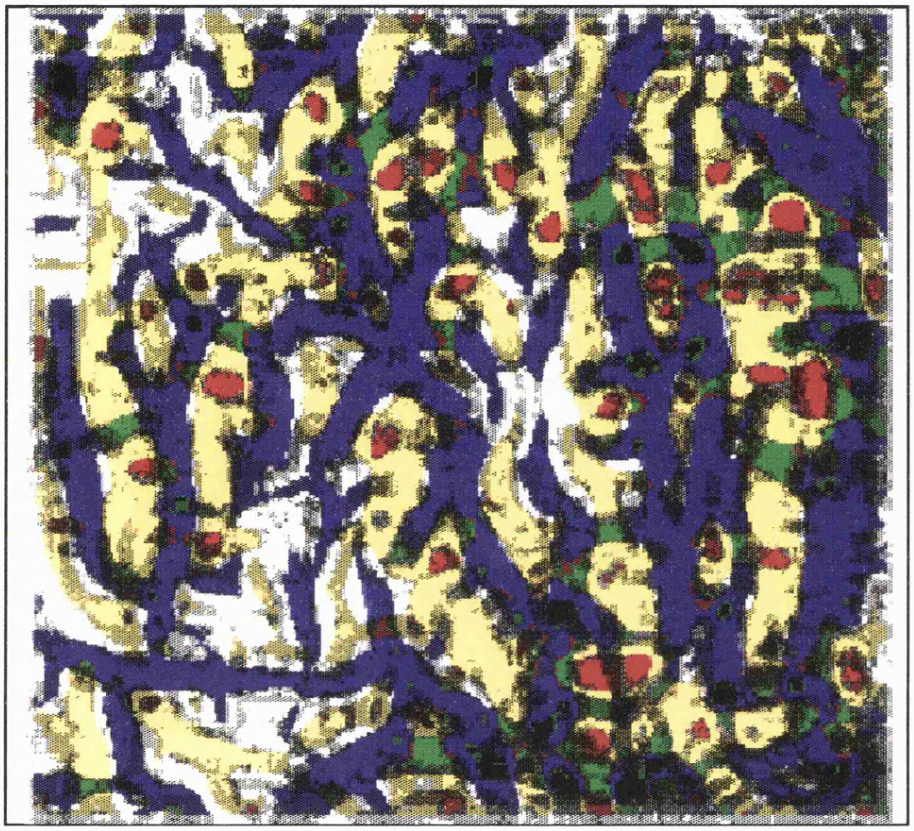


Figure 6.19 - Dartmoor feature classification certainty (see text).

Modal classifications are not all expressed at the same scale. That is, the size of each feature is not consistent over the surface. Features become progressively less well defined towards their edges, as the influence of a wider neighbourhood over a central cell's classification becomes increasingly dominant.

Figure 6.20 shows the scale dependency of feature classification for the Lake District DEM at four different kernel sizes. The slope and curvature tolerance values selected were 20 degrees and 2 respectively.

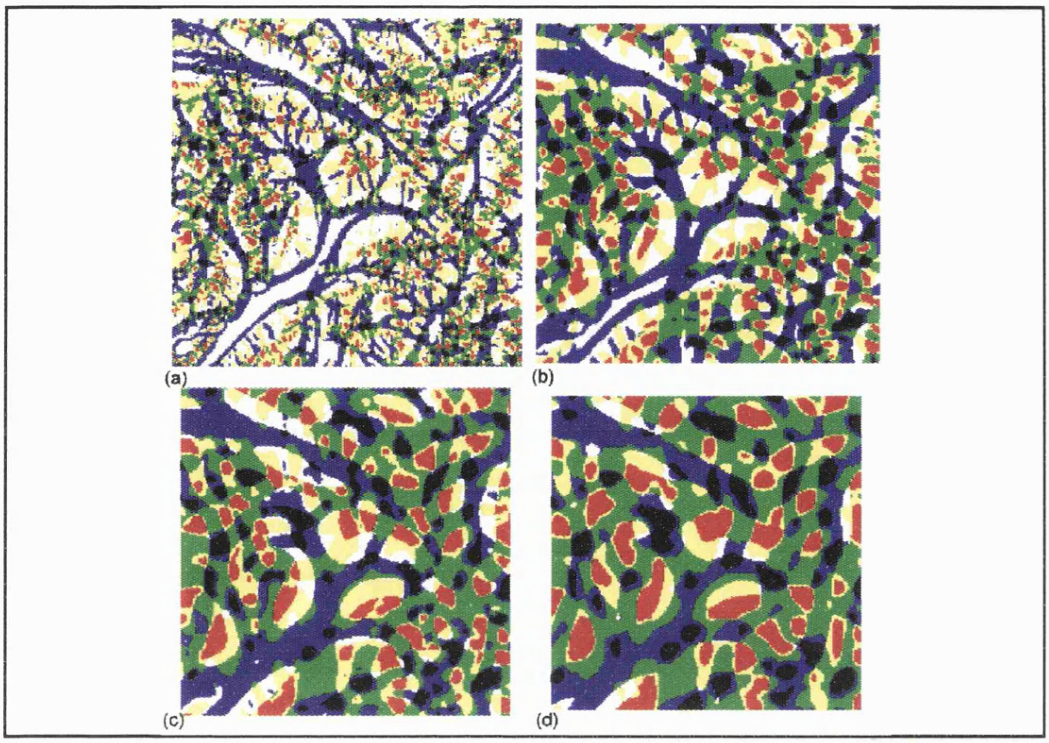


Figure 6.20 - Feature classification of the Lake District DEM using kernel sizes of (*a*) 5 x 5; (*b*) 11 x 11; (*c*)17 x 17; (*d*) 25 x 25.

The certainty of feature classification is shown in Figures 6.21 and 6.22 using the mode and entropy method discussed in section 5.3.

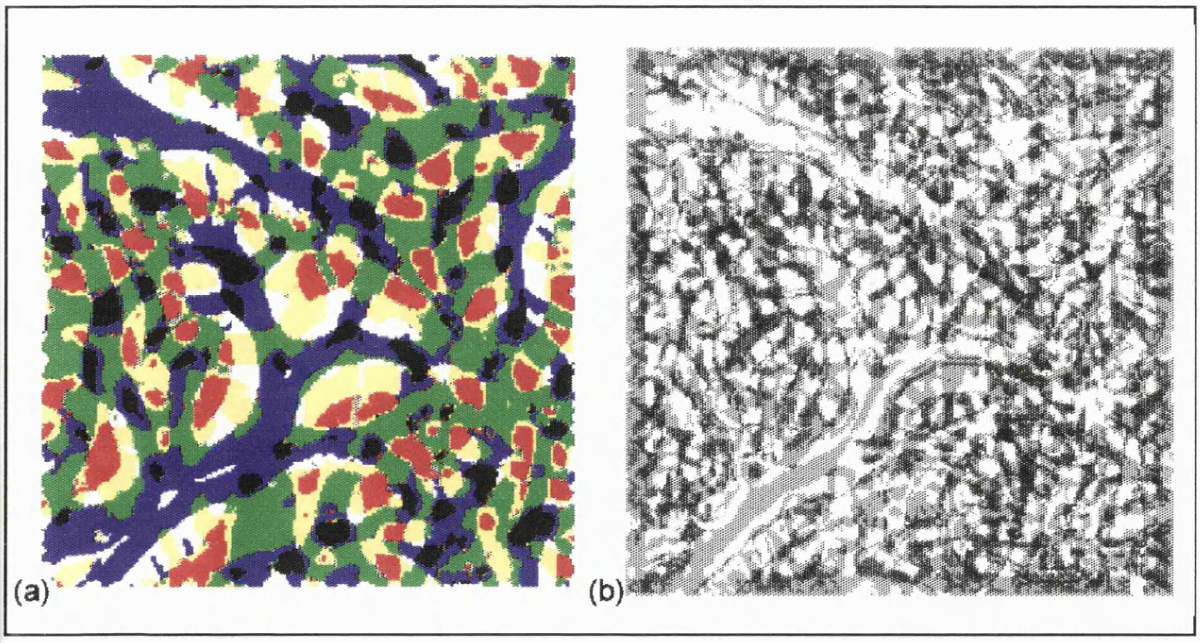


Figure 6.21 - Lake District feature classification (*a*) mode, and (*b*) entropy.

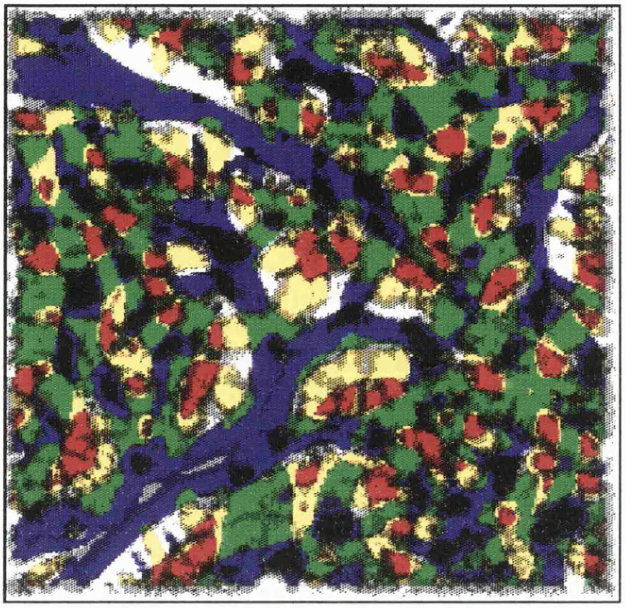


Figure 6.22 - Lake District feature classification certainty

One final example of feature classification is provided by the Peak District DEM shown in Figure 6.23. In this case a 15 x 15 cell kernel was passed over the 780 x 800 cell DEM. The distribution of peaks in particular is not uniform of the entire surface, with a higher density of peaks seen to the west and south of the image. This corresponds to the Carboniferous Limestone outcrop which contrasts with the flat topped moorland of Millstone Grit to the north and east. At this scale, the topological relationships between surface features may be of importance. Figure 6.24 shows the Delaunay triangulation of the thinned point features shown in Figure 6.23 (pits, passes and peaks) (see *r.feat.thin* and *v.delaunay* in the Appendix).

Figure 6.23 - Peak District surface features.

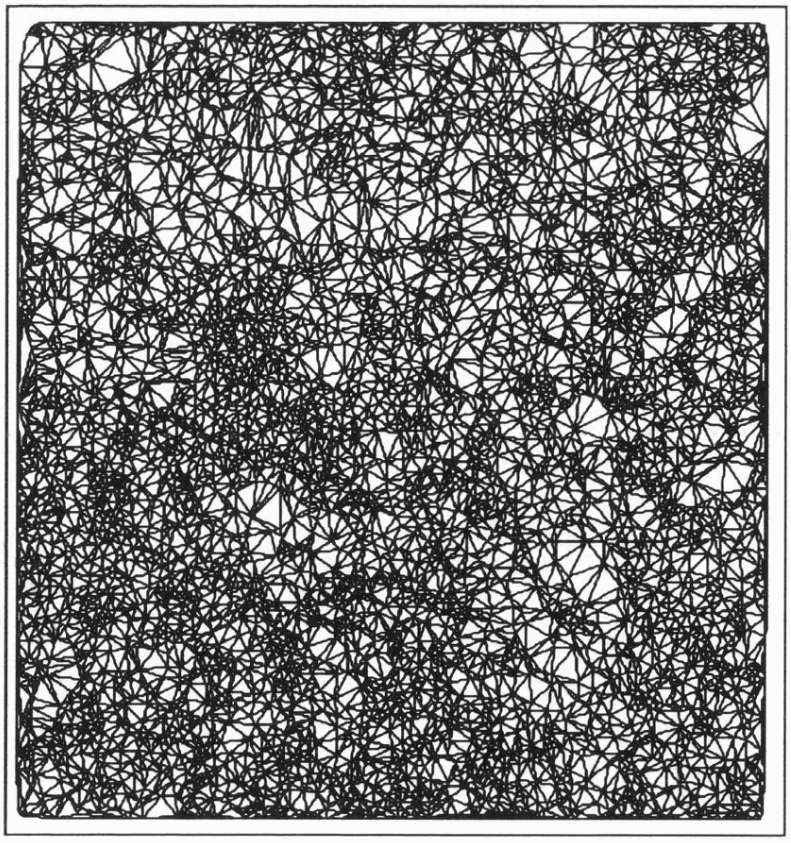


Figure 6.24 - Peak District feature triangulation

Chapter Seven - Conclusions and Further Work.

It has been the aim of this study to develop a new set of tools that describe landscape form. In particular, the *thesis* argued is that this can only be achieved effectively by (i) identifying spatial and scale based variation, and (ii) visualising such variation. This concluding chapter considers the success of the approach be re-evaluating the research objectives stated in the introduction, identifying weaknesses of the approach, and avenues for future research.

7.1 Re-evaluation of Research Aims and Objectives

To assess whether a DEM alone contains useful information for geomorphological characterisation.

This research has been deliberately confined to the analysis of regular matrices of elevation values. Clearly, additional sources of information such as those provided by satellite imagery, and ground survey, provide a more detailed model of landscape. Yet it has been made clear by this work that DEMs are highly information rich descriptions of surface form. It has been argued, that many of the previous attempts to extract information from DEMs have failed to utilise fully, the scale based information contained within the model.

By adopting the parameterisation of surface form first suggested by Evans (1972) it has been possible to examine and quantify local spatial variation systematically. This local description of morphology is sufficiently exhaustive to describe geomorphological form, whilst being sufficiently simple to relate to geomorphological process. Parameterisation has been extended by modelling over a range of scales that remove the surface model from the constraints implied by the DEM resolution. Thus it has been possible to create a more useful source of information for geomorphological analysis.

To identify weaknesses in existing methods of surface characterisation and present new methods to overcome these weaknesses.

The original form of geomorphometric analysis presented by Evans (1972) involved characterisation by quantifying the frequency distribution of surface parameters. Whilst this undoubtedly reveals important diagnostic information, it does little to characterise either the spatial distribution of parameters, or their scale dependency. Quantifications of scale dependencies such as Moran's I spatial autocorrelation index provide little extra information if they are used as a global summary statistic. The solution to this problem suggested by this work is that surface analysis is most effectively carried out within an interactive visualisation context. Consequently, many of the characterisation tools developed have been orientated towards visual representation rather than numerical summary.

More recent geomorphometric analysis has been within the context of the widespread use of Geographical Information Systems. In particular, much research has been devoted to the extraction of hydrological information from DEMs. One of the most significant problems accompanying much of this research has been the apparent mismatch between our own geomorphological understanding of a landscape, and that implied by the raster data model. The multi-scaled parameterisation approach developed here reduces the impact that data model has on subsequent analysis.

To assess the effect that elevation model uncertainty has on surface characterisation.

Two reasons have been identified for the mismatch between true topographic surface form, and its representation as a DEM within a Geographical Information System. Firstly, the model itself provides some conceptual limitations. It is not possible to represent fully, a continuous, undifferentiatable surface with a discrete, finite resolution elevation model. Secondly, the process of elevation interpolation required for DEM generation can lead to model error.

It has been demonstrated that the spatial manifestation of DEM error can, in many

circumstances, be detected in much the same way as true geomorphometric variation. The visualisation of hypsometric distributions demonstrated in Chapter 3, has led to the quantification of the terracing effects resulting from contour interpolation. The visualisation of first and second derivatives of interpolated surfaces has allowed the causes of interpolation error to be hypothesised. Visualising the spatial distribution of DEM error has facilitated the development of a deterministic error model based on planimetric offset of elevation data.

These processes have demonstrated that much the same procedure can be adopted when assessing the surface form that is a function of DEM error, and surface form that is a function of geomorphological process. Yet they have also allowed the two to be separated; error possessing a 'topographic signature' in much the same way as geomorphometry. DEM error has been shown empirically to be largely a high frequency phenomenon (usually over a scale of order 1-5 cells) often possessing artifacts of the original data source. Once surface variation is considered at a scale beyond that implied by the DEM resolution, error effects are minimised.

To assess the effectiveness of visualisation as a means of conveying information on surface form, and as a methodological process.

It has been argued that visualisation has been a necessary step in generating the ideas discussed in this study. The interactive generation of images within a GIS environment has allowed ideas to be hypothesised, tested and confirmed. It was used to hypothesise sources of contour interpolation error and to develop a deterministic error model.

Visualisation has been used to illustrate statistical properties of spatial association in the form of lag diagrams. Such patterns would have been revealed by numerical summary alone. Visualisation has been used as a mechanism for conveying complex multivariate information that could not be effectively achieved by other means. The construction of animated sequences (displayed here as *small multiples*) of feature classification patterns changing with scale, simultaneously convey the interrelationship between four geomorphometric quantities (altitude,

local relief, feature classification and scale dependency).

To produce working software that may be used in a GIS environment for surface characterisation.

The appendix to this volume includes the C source code for all software developed for this study. All modules are fully integrated with the GRASS GIS, allowing the benefits of existing GIS technology to be combined with the multiscale visualisation approach suggested here.

7.2 Further Work

7.2.1 Application

Although tools for geomorphometrical analysis have been developed as part of this study, they have not been used to provide a comprehensive or thorough geomorphological assessment of landscape. For the methodology suggested here to be of any use, it must allow new insights into surface form, rather than being used in a confirmatory context. This can only be achieved by considering the geomorphological context more thoroughly.

Multiscale parameterisation has been used to characterise population density surface properties (Wood *et al,* 1996). The use of quadratic generalisation appears particularly appropriate for such surfaces that exhibit a low spatial autocorrelation and strongly positive skew. The use of lag diagrams would be appropriate for analysing surfaces that exhibit anisotropic structural properties over a variety of scales. The rippling of fine grained channel bed form, or larger scale dune systems should be readily detectable using this method.

Work is being carried out in using multi-scaled feature classification as part of the modelling cognitive landscape evaluation. The ability to alter the scale at which features are classified makes this a particularly appropriate approach. A similar classification could be used as part of an automated cartographic generalisation process, both of landscape form and cartographic name placement.

7.2.2 Textural Analysis and Lag Diagrams.

One of the weaknesses in the use of textural lag diagrams was the difficulty in relating the measures with known geomorphological characteristics. It might be more appropriate to use the co-occurrence matrix as the basis for inferential hypothesis testing. The co-occurrence matrix has the form of a contingency table used for categorical data analysis. It would be useful to compare the matrix model with some expectation. Yet the standard Chi-squared expectation of independence is highly artificial for such a spatial distribution. Log-linear modelling provides a mechanism for alternative assumptions to be made about expectations. It would be useful to incorporate models of positive spatial autocorrelation into calculations of model expectation so that it would be possible distinguish 'expected' from 'unexpected' surface behaviour.

7.2.3 Data Structure Development

One of the important results to come from the feature classification process is that a simple Boolean classification of landscape features is not always appropriate. A feature membership function that describes the variation in classification with scale gives a more flexible alternative. Equally, a similar function could describe the change in any morphometric parameter with scale. It would be useful to incorporate such a function into subsequent GIS operations. Fuzzy logic would seem to provide a convenient mechanism for formalising the manipulation of such functions as part of a new type of elevation model.

An alternative elevation model could be developed based on the topological characteristics derived from multiscale quadratic modelling. In particular, the graph theoretic approach suggested by Wolf (1984) provides a parsimonious 'map' of the connectedness of surface features. It is possible to thin point features (ie pits, peaks, passes) and connect with thinned line features (ridges and valleys) as part of a weighted surface network. Such a model could itself be a useful surface characterisation, or alternatively be used for terrain generalisation.

Bibliography

Abrahams, A. D. (1984). Channel networks: A geomorphological perspective. *Water Resources Research*. 20 (2), 161-168.

Ackermann, F. (1993) Automatic generation of digital elevation models, 16 pp. *presented at OEEPE Commision B, DTM Accuracy Meeting, Southampton.*

Ammeraal, L. (1986) *Programming Principles in Computer Graphics*, London: Wiley.

Bailey, T. and Gatrell, A. (1995) *Interactive Spatial Data Analysis,* Essex: Longman

Bajcsy, R. (1973) Computer description of textured surfaces, *Proceedings, 1973 International Conference on Artificial Intelligence, pp.572-579*

Band, L.E. (1989) A terrain based watershed information system. *Hydrological Processes* 3, *pp.151-162*

Band, L. E. (1986). Topographic partition of watersheds with digital elevation models. *Water Resources Research 22(1), pp.15-24.*

Band, L. E. and Wood, E. F. (1988). Strategies for large scale, distributed hydrologic simulation. *Applied Mathematics and Computation. 27, pp.33-37.*

Barnsley, M.F. (1988) *Fractals Everywhere,* Academic Press.

Bathurst, J.C. (1986) Physically-based distributed modelling of an upland catchment using the Systeme Hydrologique Europeen, *Journal of Hydrology, 87, pp.79-102.*

Beard, M.K., Buttenfield, B.P., and Clapham, S.B. (1991) NCGIA research initiative 7: Visualisation of spatial data quality, *Technical Paper 91-26*, National Center for Geographic Information and Analysis, Santa Barbara

Bennett, D. A. and Armstrong, M. P. (1989). An inductive bit-mapped classification scheme for terrain feature extraction. *Proceedings of the GIS/LIS conference 1989, Orlando.* 1, *pp.59-68.*

Brassel, K. (1974) A model for automatic hill shading. *The American Cartographer* 1(1), *pp. 15-27*

Brown, D.G. and Bara, T. J. (1994) Recognition and reduction of systematic error in elevation and derivative surfaces from 7 1/2 minute DEMs, *Photogrammetric Engineering and Remote Sensing, 60 (2), pp.189-194.*

Burrough, P.A. (1981) Fractal dimensions of landscapes and other environmental data. *Nature, 294, pp.240-242.*

Burrough, P.A. (1986) *Principles of Geographical Information Systems for Land Resources Assessment*. OUP, Oxford, Ch.8, Methods of interpolation, *pp. 147-166.*

Burrough, P.A. (1993) Fractals and geostatistical methods in landscape studies, *in* **Lam, N. and De Cola, L.** *Fractals in Geography*, Englewood Cliffs, NJ: Prentice Hall.

Burrough, P.A. and Heuvelink, G.B.M. (1992) The sensitivity of Boolean and continuous (fuzzy) logical modelling to uncertain data. *Proceedings, EGIS 92, Munich 2, pp.1032-1041*

Carter, J.R. (1989) Relative errors identified in USGS DEMs. *Proceedings, Auto Carto 9,* American Congress on Surveying and mapping, Bethesda, Maryland, *pp.255-265*

Carter, J. (1992) The effect of data precision on the calculation of slope and aspect using gridded DEMs, *Cartographica, 29 (1), pp.22-34.*

Cayley, A. (1859). On contour and slope lines. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science.* XVIII, *pp. 264-268.*

CERL (1991) The GRASS 4.0 Reference Manual. U.S. Army Construction Engineering Research Laboratory, Champaign, Illinois.

Chambers (1990) *Chambers English Dictionary, 7th Edition*, Edinburgh: Chambers.

Chang, K. and Tsai, B. (1991) The effect of DEM resolution on slope and aspect mapping, *Cartography and Geographic Information Systems, 18, pp.69-77.*

Chrisman, N.R. (1989) Modeling error in overlaid categorical maps. *in* **Goodchild, M.F. and Gopal, S.** (eds.) (1989) *The Accuracy of Spatial Databases*. Taylor and Francis, London. *pp.21-34*

Chorley, R.J., Malm, D.E. and Pogorzelski, H.A. (1957) A new standard for estimating drainage basin shape, *American Journal of Science, 255(2) pp.138-141.*

Chorowicz, J., Ichoku, C., Raizanoff, S., Kim, Y., and Cervelle, B. (1992) A combined algorithm for automated drainage network extraction, *Water Resources Research, 28 (5), pp.1293-1302.*

Chorowicz, J., Kim, J., Manoussis, S., Rudant, J., Foin, P. and Veillet, I. (1989) A new technique for recognition of geological and geomorphological patterns in digital terrain models, *Remote Sensing of Environment, 29 (3), pp.229-239.*

Church, M. and Mark, D. (1980) On size and scale in geomorphology, *Progress in Physical Geography 4, pp.342-390*

Clarke, K.C. (1986) Computation of the fractal dimension of topographic surfaces using the triangular prism surface area method, *Computers and Geosciences, 12 (5), pp.713-722.*

Clarke, K. C. (1988). Scale based simulation of topographic relief. *The American Cartographer 15 (2), pp.171-181.*

Clarke, K., and Schweitzer, D. (1991) Measuring the fractal dimension of fractal surfaces using a robust fractal estimator, *Cartography and Geographic Information Systems, 18 (1)*

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

Collins, S. H. (1975). Terrain parameters directly from digital elevation models. *Canadian Surveyor* 29 (5), *pp.507-518.*

Cressie, N.A.C. (1993) *Statistics for Spatial Data.* Ney York: Wiley.

Culling, W.E.H. (1989) The characterization of regular/irregular surfaces in the soil-covered landscape by gaussian random fields, *Computers and Geosciences, 15 (2), pp.219-226.*

Darling, E. and Joseph, R. (1968) Pattern recognition from satellite altidudes, *IEEE Transactions on Systems, Science and Cybernetics, 4, pp.38-47.*

Datcu, M. and Seidel, K. (1995) Fractal and multi-resolution techniques for the understanding of geo-information, *in* **Wilkinson, G, Kanellopoulos, I. and Mégier, J.** *(eds) Fractals in Geosciences and Remote Sensing,* Joint Research Centre, Report EUR 16092 EN. *pp.56-84*

DiBiase, D. (1990) Visualisation in the earth sceinces, *Earth and Mineral Sciences, Pennsylvania State University, 59(2), pp.13-18.*

Dietler, G. (1995) The fractal structure of materials and surfaces, *in* **Wilkinson, G, Kanellopoulos, I. and Mégier, J.** *(eds) Fractals in Geosciences and Remote Sensing,* Joint Research Centre, Report EUR 16092 EN. *pp.325-341.*

Dietler, G. and Zhang, Y. (1992) Fractal aspects of the Swiss landscape, *Physica A, 191, pp.213-219.*

Dikau, R. (1989) The application of a digital relief model to landform analysis in geomorphology. in *Three dimensional applications in GIS*, **Raper, J.** ed., Taylor and Francis, London, *pp.51-77*

Doornkamp, J.C. (1968) The analysis of the morphometric properties of drainage basins by the Spearman's rank correlation technique, *in* **Slaymaker, H.O. (ed)** (1968) Morphometric analysis of maps, *British Geomorphological Research Group, Occasional Paper, pp.31-40.*

Douglas, D. H. (1986). Experiments to locate ridges and channels to create a new type of digital elevation model. *Cartographica 23(1), pp.29-61.*

Dunkerley, D. L. (1977). The frequency distributions of stream link lengths and the development of channel networks. *Journal of Geology, 1985, pp.459-470.*

Dury, G.H. (1951) Quantitive measurement of available relief and depth of dissection, *Geological Magazine, 88(5), pp.339-343.*

Dykes, J.A. (1994) Area-value data: New visual emphases and representations. in **Hearnshaw and Unwin** *(eds)* (1994) *Visualization in Geographical Information Systems,* Chichester: Wiley, *pp.103-114*

Evans, I. S. (1972). General geomorphometry, derivatives of altitude, and descriptive statistics. in **Chorley, R. J.** ed. *Spatial Analysis in Geomorphology,* Methuen, London. *pp.17-90.*

Evans, I.S. (1979) An integrated system of terrain analysis and slope mapping. *Final report on grant DA-ERO-591-73-G0040*, University of Durham, England.

Evans, I.S. (1980) An integrated system of terrain analysis and slope mapping. *Zeitschrift fur Geomorphologie*, Suppl-Bd 36, *pp.274-295.*

Evans, I.S. (1981) General geomorphometry, *in* **Goudie, A.S.** (ed.), *Geomorphological Tachniques, pp.31-37.*

Evans, I.S. (1984) Correlation structures and factor analysis in the investigation of data dimensionality: statistical properties of the Wessex land surface, England, *International Symposium on Spatial Data Handling, Zurich, 1, pp.98-116.*

Fairfield, J. and Leymarie, P. (1991) Drainage networks from grid digital elevation models, *Water Resources Research, 27(5), pp.709-717.*

Feder, J. (1988) *Fractals*, Plenum Press.

Frederiksen, P., Jacobi, O., and Kubik, K. (1984) Modelling and classifying terrain, *International Archives of Photogrammetry and Remote Sensing, XXV, Part A3a, pp.256-257.*

Fioravanti, S. (1995) Multifractals: theory and application to image texture recognition, *in* **Wilkinson, G, Kanellopoulos, I. and Mégier, J.** *(eds) Fractals in Geosciences and Remote Sensing,* Joint Research Centre, Report EUR 16092 EN. *pp.152-175*

Fisher, P.F. (1990) Simulation of error in digital elevation models. *Papers and Proceedings of the 13th Applied Geography Conference 13, pp. 37-43*

Fisher, P.F. (1991a) Modelling soil map-unit inclusions by Monte Carlo simulation, *International Journal of Geographical Information Systems, 5(2), pp.192-208.*

Fisher, P.F. (1991b) First experiments in viewshed uncertainty: The accuracy of the viewshed area. *Photogrammetric Engineering and Remote Sensing* 57 (10) *pp.*1321-1327

Fisher, P.F. (1992) First experiments in viewshed uncertainty: Simulating fuzzy viewsheds. *Photogrammetric Engineering and Remote Sensing* 58 (4) *pp.*345-352

Fisher P. (1993) Algorithm and implementation uncertainty in viewshed analysis, *International Journal of Geographical Information Systems, vol 7 (4) pp.331-347*

Fox, C.G. (1989) Empirically derived relationships between fractal dimension and power law form frequency spectra, *Pure and Applied Geophysics, 131 (1/2) pp.211-239.*

Frank, A., Palmer, B. and Robinson, V. (1986) Formal methods of the accurate definition of some fundamental terms in physical geography, *Proceedings, Second International Symposium on Spatial Data Handling, Seattle, pp.583-599.*

Gallant, J.C. and Hutchinson, M.F. (1996) Towards an understanding of landscape scale and structure, *Proceedings, Third International Conference on Integrating GIS and Environmental Modelling, http://www.ncgia.ucsb.edu/conf/santa_fe.html, Santa Barbara: National Center for Geographic Information and Analysis*

Garbrecht, J. and Martz, L. (1993) Grid size dependency of parameters from digital elevation models, *12 pp. (source unknown)*

Garbrecht, J. and Starks, P. (1995) Note on the use of USGS level 1 7.5-minute DEM coverages for landscape drainage analyses, *Photogrammetric Engineering and Remote Sensing, 61 (5), pp.519-522.*

Gardner, T.W., Sasowsky, K.C., and Day, R.L. (1990) Automated extraction of geomorphometric properties form digital elevation data, *Zeitschrift fur Geomorphologie Suppl. 80, pp.57-68.*

Gilbert, L. E. (1989) Are topographic data sets fractal ? *Pure and Applied Geophysics, 131(1/2), pp.241-254.*

Glock, W.S. (1932) Available relief as a factor of contol in the profile of a land form, *Journal of Geology, 40(1), pp.74-83.*

Gonzalez, R. C. and Woods, R.C. (1992) *Digitial Image Processing*, Reading, Mass: Addison-Wesley.

Goodchild, M.F. (1980) Fractals and the accuracy of geographical measures, *Mathematical Geology 12(2), pp.85-98*

Goodchild, M.F. (1982) The fractional Brownian process as a terrain simulation model, *Modelling and Simulation, 13, pp.1133-1137.*

Goodchild, M. F. (1986) Spatial Autocorrelation, *Concepts and Techniques in Modern Geography (CATMOG) 47*, GeoBooks, Norwich.

Goodchild, M. F. (1988). Lakes on fractal surfaces: A null hypothesis for lake-rich landscapes. *Mathematical Geology* 20,6, *pp.615-630.*

Goodchild, M.F. and Grandfield, A.W. (1983) Optimizing raster storage: An evaluation of four alternatives, *Proceedings, Sixth International Symposium on Automated Cartography, Ottowa (Auto-Carto VI), 2, pp.400-407.*

Goodchild, M.F. and Gopal, S. (eds.) (1989) *The Accuracy of Spatial Databases*. Taylor and Francis, London.

Goodchild, M.F., Guoqing, S. and Shiren, Y. (1992) Development and test of an error model for categorical data. *International Journal of Geographical Information Systems* 6 (2) *pp.87-104*

Goodchild, M. F. and Mark, D. M. (1987). The fractal nature of geographic phenomena. *Annals of the Association of American Geographers* 77(2), *pp.265-278.*

Goudie, A. (1990) *(ed) Geomorphological Techniques (second edition),* London: Unwin for the British Geomorphological Research Group.

Gould, P.R. (1967) On the geographical interpretation of eigenvalues, *Transactions of the Institute of British Geographers, 42, pp.53-86.*

Govaerts, Y and Verstraete, M. (1995) Applications of the L-systems to canopy reflectance modelling in a Monte Carlo ray tracing technique *in* **Wilkinson, G, Kanellopoulos, I. and Mégier, J.** *(eds) Fractals in Geosciences and Remote Sensing,* Joint Research Centre, Report EUR 16092 EN. *pp.211-236*

Graff, L.H. and Usery, E.L. (1993) Automated classification of generic terrain features in digital elevation models, *Photogrammetric Engineering and Remote Sensing, 59(9), pp.1409-1417.*

Graps, A. (1995) An introduction to wavelets, *IEEE Computational Science and Engineering, 2(2), 18 pp.*

Guth, P. (1992) Spatial analysis of DEM error, *Proceedings, ASPRS/ACSM Annual Meeting, Washington DC, pp.187-196.*

Haber, R.B. and McNabb, D.A. (1990) Visualisation idioms: a conceptual model for scientific visualisation systems, *in* Neilson, M., Shriver, B., and Rosenblaum, L. (1990) *Visualisation in scientific computing*, IEEE Computer Society Press.

Hack, J.T. (1957) Studies of longitudinal stream profiles in Virginia and Maryland, *US Geological Survey, Professional Paper294-B, pp.45-94.*

Haining, R., Griffith, D. and Bennett, R. (1983) Simulating two-dimensional autocorrelated surfaces, *Geographical Analysis, 15(3), pp.247-255*

Hakanson, L. (1978) The length of closed geomorphic lines, *Mathematical Geology 10, pp.141-167.*

Hannah, M.J. (1981) Error detection and correction in digital terrain models. *Photogrammetric Engineering and Remote Sensing 47(1), pp.63-69.*

Haralick, R.M., Shanmugam, K. and Dinstein, I. (1973) Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics, 3(6), pp.610-611.*

Haralick, R.M. (1979) Statistical and structural approaches to texture, *IEEE Transactions on Systems, Man and Cybernetics, 4(7), pp.394-396.*

Haralick, R.M. (1983) Ridge and valley on digital images, *Computer Vision, Graphics and Image Processing, 22(1), pp.28-38.*

Heerdegen, R.G. and Beran, M.A. (1982) Quantifying source areas through land surface curvature and slope, *Journal of Hydrology, 57 (3-4), pp.359-373.*

Heuvelink, G.B. (1993) *Error Propagation in Quantitiative Spatial Modelling. Applications in Geographical Information Systems,* University of Utrecht.

Hodgson, M.E. (1995) What cell size does the computed slope / aspect angle represent ? *Photogrammetric Engineering and Remote Sensing*, 61(5), *pp. 513-517.*

Horton, R.E. (1945) Erosional development of streams and their drainage basins, hydrophysical approach to quantitative morphology, *Bulleting, Geological Society of America, 56(3), pp.275-370.*

Huang, J. and Turcotte, D. (1989) Fractal mapping of digitized images - application to the topography of Arizona and comparisons with synthetic images, *Journal of Geophysical Research 94 (B6), pp.7491-7495*

Huang, J. and Turcotte, D. (1990) Fractal image analysis: application to the topography of Oregon and synthetic images, *Journal of the Optical Socisty of America A, 7(6), pp.1124-1130*

Hunter, G.J. and Goodchild, M.F. (1995) Dealing with error in spatial databases: A simple case study, *Photogrammetric Engineering and Remote Sensing, 61)5), pp.529-537.*

Hurst, H., Black, R., and Simaika, Y. (1965) *Long-Term Storage: An Experimental Study,* London: Constable.

Hutchinson, M.F. (1989). A new procedure for gridding elevation and stream line data with automatic removal of spurious pits. *Journal of Hydrology 106, pp.211-232.*

Imhof, E. (1982) *Cartographic Relief Presentation*, (ed Steward, H.J), Berlin: Walter de Gruyter.

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

Jarvis, R.S. (1981) Specific geomorphometry, *in* **Goudie, A.S.** (ed.), *Geomorphological Tachniques, pp.42-46.*

Jarvis, R. S. and Sham, C. H. (1981). Drainage network structure and the diameter magnitude relation. *Water Resources Research.* 17, *pp.1019-1027.*

Jenson, S. K. (1985). Automated detection of hydrologic basin characteristics from digital elevation model data. *Digital Representation of Spatial Knowledge, Auto-Carto 7., American society of Photogrammetry and the American Congress on Surveying and Mapping. pp.301-310.*

Jenson, S.K. (1991) Applications of hydrologic information automatically extracted from digital elevation models, *Hydrological Processes 5, pp.31-44.*

Jenson, S. K. and Domingue, J. O. (1988). Extracting topographic structure from digital elevation data for geographic information system analysis. *Photogrammetric Engineering and Remote Sensing 54(11), pp.1593-1600.*

Johnson, D. (1933) Available relief and texture of topography, a discussion, *Journal of Geology, 41(3), pp.293-305.*

de Jong, S. (1995) Mapping Spatial Variablility in Landscapes: An example using fractal dimensions, *in* **Wilkinson, G, Kanellopoulos, I. and Mégier, J.** *(eds) Fractals in Geosciences and Remote Sensing,* Joint Research Centre, Report EUR 16092 EN. *pp.176-210.*

Kaizer, H. (1955) A quantification of textures on aerial photographs, *Technical Note 121, AD 69484, Boston University Research Laboratory.*

Kidner, D.B., Jones, C.B., Knight, D.G. and Smith, D.H. (1990) Digital terrain models for radio path profiles. *Proceedings of the 4th International Symposium on Spatial Data Handling, Zurich, pp.240.*

King, C.A. (1968) A morphometric example of factor analysis *in* **Slaymaker, H.O. (ed)** (1968) Morphometric analysis of maps, *British Geomorphological Research Group, Occasional Pape, pp.41-48.*

Klinkenberg, B. (1992) Fractals and morphometric measures: is there a relationship ? *Geomorphology 5, pp.5-20*

Klinkenberg, B. and Goodchild, M. (1992) The fractal properties of topography: A comparison of methods, *Earth Surface Processes and Landforms, 17, pp.217-234.*

Knighton, A.D. (1984) *Fluvial Forms and Processes,* London: Arnold.

Kumler, M.P. (1994) An intensive comparison of traiangulated irregular networks (TINs) and Digital Elevation Models (DEMs), *Cartographica, Monograph, 31(2).*

Lam, N. (1983) Spatial Interpolation methods: A review, *American Cartographer, 10, pp.129-149.*

Lam, N. and De Cola, L. (1993) *Fractals in Geography*, Englewood Cliffs, NJ: Prentice Hall

Lammers, R. and Band, L. (1990) Automating object representation of drainage basins, *Computers and Geosciences, 16 (6), pp. 787-810*

Lavalle, D., Lovejoy, S., Schertzer, D. and Ladoy, D. (1993) Non-linear variability in landscape topography: Multifractal analysis and simulation, *in* **Lam, N. and De Cola, L.** *(eds) Fractals in Geography*, Englewood Cliffs, NJ: Prentice Hall

Lee, J., Snyder, P.K. and Fisher, P.F. (1992) Modelling the effect of data errors on feature extraction from digital elevation models. *Photogrammetric Engineering and Remote Sensing* 58(10) *pp.1461-1467*

Lévy-Véhel, J. (1995) Multifractal analysis of remotely sensed images, *in* **Wilkinson, G, Kanellopoulos, I. and Mégier, J.** *(eds) Fractals in Geosciences and Remote Sensing,* Joint Research Centre, Report EUR 16092 EN, *pp.85-101.*

Lewis, L.A. (1968) Analysis of surficial landform properties-the regionalization of Indiana into units of morphometric similarity, *Proceedings of the Indiana Acadamy of Science, 78, pp.317-328.*

Li, Z. (1988) On the measure of digital terrain model accuracy. *Photogrammetric Record, 72 (12), pp.873-877.*

Li, Z. (1993a) Theoretical models of the accuracy of digital terrain models: An evaluation and some observations, *Photogrammetric Record 14 (82), pp.651-659.*

Li, Z. (1993b) Mathematical models of the accuracy of digital terrain model surfaces linearly constructed from square gridded data, *Photogrammetric Record 14 (82), pp.661-673.*

Lifton, N.A. and Chase, C.G. (1992) Tectonic, climatic, and lithologic influences on landscape fractal dimension and hypsometry: Implications for landscape evolution in the San Gabriel mountains, California, *Geomorphology 5, pp.77-114.*

Lovejoy, S. and Schertzer, D. (1995) How bright is the coast of Brittany ?, *in* **Wilkinson, G, Kanellopoulos, I. and Mégier, J.** *(eds) Fractals in Geosciences and Remote Sensing,* Joint Research Centre, Report EUR 16092 EN. *pp.102-151*

Lu, S.Y. and Fu, K.S. (1978) A syntactic approach to texture analysis, *Computer Graphics and Image Processing, 7(3), pp.303-330*

MacEachren, A.M. and Davidson, J.V. (1987) Sampling and isometric mapping of continuous geographic surfaces. *The American Cartographer 14,(4) pp.299-320.*

Mackaness, W. and Beard, K. (1993) Use of graph theory to support map generalisation, *Cartography and Geographical Information Systems, 20(4), pp.210-221.*

McLaren, R.A., and Kennie, T.J.M. (1989) Visualisation of digital terrain models: techniques and applications. in *Three dimensional applications in GIS*, **Raper, J.** ed., Taylor and Francis, London, *pp.80-98*

Mandelbrot, B.B (1967) How long is the coastline of Britain ? Statistical self-similarity and fractional dimension, *Science 156, pp.636-638.*

Mandelbrot, B.B. (1975) *Les Objects Fractal: Forme, Hasard et Dimension*, Paris: Flammarion

Mandelbrot, B.B. (1977) *Fractals: Form, Chance and Dimension*, San Fransisco: WH Freeman

Mandelbrot, B.B. (1982) *The Fractal Geometry of Nature*, New York: WH Freeman

Mandelbrot, B.B. (1986) Self-affine fractal sets, *in* **Pietronero, L. and Tosatti, E.** (eds) *Fractals in Physics*, Holland, *pp.3-28*

Mark, D.M. (1975a) Geomorphometric parameters: a review and classification, *Geografiska Annaler 57 A, pp.165-177.*

Mark, D.M. (1975b) Computer analysis of topography: a comparison of terrain storage methods, *Geografiska Annaler 57 A, pp.179-188.*

Mark, D.M. (1979) Topology of ridge patterns, randomness and constraints, *Bulletin, Geological Society of America, 90(2)* pp.164-172.

Mark, D. M. (1983a). Automated detection of drainage networks from digital elevation models. in: **Wellar, B. S.** 1983 . *Automated Cartography:international perspectives on achievements and challenges. Ottawa, Auto-Carto Six* 2, *pp.168-177.*

Mark, D. M. (1983b). Relations between field-surveyed channel networks and map-based geomorphometric measures, Inez, Kentucky. *Annals of the Association of American Geographers.* 73, *pp.358-272.*

Mark, D. M. and Aronson, P.B. (1984) Scale-Dependent fractal dimensions of topographic surfaces: An empirical investigation with applications in geomorphology and computer mapping, *Mathematical Geology, 16 (7), pp.671-683.*

Marks, D., Dozier, J. and Frew, J. (1983). Automated basin delineation from digital terrain data. *NASA Technical Memorandum 84984.*

Martz, L.W. and DeJong, E. (1988) Catch: A Fortran program for measuring catchment area from digital elevation models, *Computers and Geosciences, 14(5), pp.627-640.*

Martz, L.W. and Garbrecht, J. (1992) Numerical definition of drainage network and subcatchment areas from Digital Elevation Models. *Computers and Geosciences, 18 (6), pp. 747-761*

Maxwell, J.C. (1870). On contour lines and measurements of heights. *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science. 40, pp.421-427.*

McKim, R.H. (1972) *Experiences in Visual Thinking*, Monterey, CA: Brooks/Cole.

Meisels, A., Raizman, S. and Karnieli, A. (1995) Skeletonizing a DEM into a drainage network, *Computers and Geosciences, 21 (1), pp.187-196.*

Miller, R.L. and Kahn, J.S. (1962) *Statistical Analysis in the Geological Sciences*, John Wiley, London.

Mitasova H. (1992) Surfaces and Modelling, *Grassclippings, The Journal of Open Geographical Information Systems, 6 (3), pp.16-18.*

Mitasova H. and Mitas, L. (1993) Interpolation by regularized spline with tension: I theory and implementation, *Mathematical Geology.*

Monckton, C. (1994) An investigation into the spatial structure of error in digital elevation data, *Innovations in GIS 1, pp.201-211,* London: Taylor and Francis.

Monkhouse, F.J. and Wilkinson, H.R. (1952) *Maps and Diagrams: Their Compilation and Construction.* 3rd edition, 1971, London:Methuen

Moore, I.D., Grayson, R.B. and Ladson, A.R (1991) Digital terrain modelling: A review of hydrological, geomorphological and biological applications, *Hydrological Processes 5, pp.3-30.*

Morris, D.G. and Flavin, R.W. (1984) A digital terrain model for hydrology, *4th International Sysmposium on Spatial Data Handling, Zurich, pp.250-262.*

Morris, D.G. and Heerdegen, R.G. (1988) Automatically derived catchment boundaries and channel networks and their hydrological applications. *Geomorphology 1, pp.131-141*

O'Callaghan, J. F. and Mark, D. M. (1984). The extraction of drainage networks from digital elevation data. *Computer Vision, Graphics and Image Processing. 28, pp.323-344.*

Ohmori, H. and Hirano, M. (1984) Mathematical explanation of some characteristics of altitude distributions of landforms in an equilibrium state, *Transactions, Japanese Geomorphological Union, 5(4), pp.293-310.*

Oliver, M., Webster, R. and Gerrard, J. (1989a) Geostatistics in physical geography. Part I: theory, *Transactions, Institute of British Geographers, 14, pp.259-269.*

Oliver, M., Webster, R. and Gerrard, J. (1989b) Geostatistics in physical geography. Part II: applications, *Transactions, Institute of British Geographers, 14, pp.270-286.*

Onorati, G., Poscolieri, M., Ventura, R., Chiarini, V. and Crucilla, U. (1992) The digital elevation model of Italy for geomorphology and structural geology, *Catena 19, pp.147-178.*

Ordnance Survey (1992) *1:50 000 Scale Height Data User Manual.* Ordnance Survey, Maybush, Southampton.

Ordnance Survey (1993) *A Technical Guide for Sample Digital Map Data in National Transfer Format Version 2.0.* Ordnance Survey, Maybush, Southampton.

Ostman, A. (1987) Accuracy estimation of digital elevation data banks, *Photogrammetric Engineering and Remote Sensing, 53 (4), pp.425-430*

Outcalt, S. and Melton, M. (1992) Geomorphic application of the Hausdorff-Besicovich dimension, *Earth Surface Processes and Landforms, 17, pp. 775-787.*

Ovenden, J. C. and Gregory, K. J. (1980) The permanence of stream networks in Britain. *Earth Surface Processes. 5, pp.47-60.*

Palacois-Velez, O. L. and Cuevas-Renaud, B. (1986). Automated river-course, ridge and basin delineation from digital elevation data. *Journal of Hydrology 86, pp.299-314.*

Peitgen, H.O. and Saupe, D. (1988) (eds) *The Science of Fractal Images*, pp.312, London: Springer-Verlag.

Pentland, A. P. (1984) Fractal based description of natural scenes, *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-6 (6), pp.661-674.*

Petrie, G. and Kennie, T.J. (1990) *(eds) Terrain Modelling in Surveying and Civil Engineering,* Caithness: Whittles.

Peucker, T.K. (1978) Data structures for digital terrain models - discussion and comparison, *in* **Dutton, G.** *(ed), Harvard Papers on Geographic Information Systems.*

Peucker, T. K. and Douglas, D. H. (1974). Detection of surface specific points by local parallel processing of discrete terrain elevation data. *Computer Graphics and Image Processing 4, pp.375-387.*

Pike, R.J. (1988) The geometric signature: Quantifying landslide terrain types from digital elevation models, *Mathematical Geology, 20 (5), pp.491-511.*

Pike, R.J. (1993) A bibliography of geomorphometry, *United States Geological Survey Open-File Report 93-262-A, pp. 132, Menlo Park, CA.*

Pfaltz, J. L. (1976). Surface networks. *Geographical Analysis 8(1), pp.77-93.*

Polidori, L. (1995) Fractal-based eevaluation of relief mapping techniques, *in* **Wilkinson, G, Kanellopoulos, I. and Mégier, J.** *(eds) Fractals in Geosciences and Remote Sensing,* Joint Research Centre, Report EUR 16092 EN. *pp.277-297.*

Polidori, L. and Chorowicz, J. (1990) Modelling terrestrial relief with Brownian motion: Application to digital elevation data evaluation and resampling, *EARSeL Symposium, Graz.*

Polidori, L., Chorowicz, J. and Guillande, R. (1991) Description of terrain as a fractal surface, and application to digital elevation model quality assessment, *Photogrammetric Engineering and Remote Sensing, 57 (10), pp.1329-1332*

Press, W.H., Flannery, B.P., Teukolsky, S.A. and Vetterling, W.T. (1992) *Numerical Recipes In C - The Art of Scientific Computing, Second Edition*, Cambridge University Press.

Qian, J., Ehrich, R.W., and Campbell, J.B. (1990) DNESYS - an expert system for automatic extraction fo drainage networks from digital elevation data, *IEEE Transactions on Geoscience and Remote Sensing, 28(1), pp.29-45.*

Quinn, P., Beven, K., Chevallier, P. and Planchon, O. (1991) The prediction of hillslope flow paths for distributed hydrological modelling using digital terrain models, *Hydrological Processes 5, pp.59-79.*

Rees, W.G. (1995) Characterisation and imaging of fractal topography, *in* **Wilkinson, G, Kanellopoulos, I. and Mégier, J.** *(eds) Fractals in Geosciences and Remote Sensing,* Joint Research Centre, Report EUR 16092 EN. *pp.298-324*

Richards, K.S. (1981) Introduction to morphometry, *in* **Goudie, A.S.** (ed.), *Geomorphological Tachniques, pp.25-30.*

Richardson, L.F. (1961) The problem of contiguity: an appendix of statistics of deadly quarrels, *General Systems Yearbook, 6, pp.139-187.*

Riley, S. (1993) The automated extraction of drainage networks from DEMs and remotely sensed images, *Unpublished MSc Thesis, University of Leicester.*

Robinson, G. (1993) Measuring the accuracy of Ordnance Survey digital elevation data, *Ordnance Survey Technical Report, Ordnance Survey, Southampton.*

Rosenfeld, A. and Kak, A. (1982). *Digital Picture Processing*. Academic Press, Orlando.

Roy, A., Gravel, G. and Gauthier, C. (1987) Measuring the dimension of surfaces: a review and appriasal of different methods, *Proceedings, 8th International Sysposium on Computer Assisted Cartography, Baltimore, Maryland, (Auto-Carto 8), pp.68-77.*

Saupe, D. (1988) Algorithms for random fractals, pp.71-133, *in* **Peitgen, H.O. and Saupe, D.** (1988) *(eds) The Science of Fractal Images*, London: Springer-Verlag.

Schalkoff, R.J. (1989) *Digital Image Processing and Computer Vision*, London: Wiley and Sons.

Schertzer, D. and Lovejoy, S. (1995) Standard and advanced multifractal techniques in remote sensing, *in*, **Wilkinson, G, Kanellopoulos, I. and Mégier, J.** *(eds) Fractals in Geosciences and Remote Sensing,* Joint Research Centre, Report EUR 16092 EN. *pp.11-40.*

Schumm, S.A. (1956) Evolution of drainage systems and slopes in badlands at Perth Amboy, New Jersey, *Bulletin, Geological Society of America, 67(5), pp.597-646.*

SDTS (1990) *The Spatial Data Transfer Standard.* US Government Printing Office, US Geological Survey Bulletin, Washington DC.

Seemuller, W.W. (1989) The extraction of ordered vector drainage networks from elevation data, *Computer Vision, Graphics and Image Processing, 47, pp.45-48.*

Serra, J. (1982) *Image Analyis and Mathematical Morphology*, New York: Academic Press

Shapiro, M. and Westervelt, J. (1992) *r.mapcalc: An algebra for GIS and Image Processing*, US Army Construction Engineering Research Laboratory, Champaign IL.

Shreve, R. L. (1966). Statistical law of stream numbers. *Journal of Geology.* 74, 17-37.

Shreve, R. L. (1974). Variation of mainstream length with basin area in river networks. *Water Resources Research. 10, pp.1167-1177.*

Skidmore, A.K. (1989) A comparison of techniques for calculating gradient and aspect from a gridded digital elevation model. *International Journal of Geographical Information Systems, 3(4), pp. 323-334*

Skidmore, A. K. (1990). Terrain position as mapped from a gridded digital elevation model. *International Journal of Geographical Information Systems 4(1), pp.33-49.*

Slaymaker, H.O. (ed) (1968) Morphometric analysis of maps, *British Geomorphological Research Group, Occasional Paper.*

Smith, G.H. (1935) The relative relief of Ohio: *Geographical Review, 25(2), pp.272-284*

Smith, T.R., Zhan, C. and Gao, P. (1990) A knowledge-based two-step procedure for extracting channel networks from noisy DEM data, *Computers and Geosciences 16(6), pp.777-786.*

Snow, R.S. and Mayer, L. (*eds*) (1993) Fractals in geomorphology, *Geomorphology 5 (1/2) (Special Issue)*

Speight, J. G. (1968). Parametric description of landform. in *Land Evaluation* (**G. A. Stewart Ed.**), *Proceedings of the CSIRO/UNESCO symposium, Canberra 26-31, pp.239-250.*

Speight, J.G. (1973). A parametric approach to landform regions, *Institute of British Geographers Special Publication 7: Progress in Geomorphology, pp.213-230.*

Speight, J.G. (1976). Numerical classification of landform elements from air photo data, *Zeitschrift fur Geomorphologie, pp.154-168.*

Srinivasan, R. and Engel, B.A. (1991) Effect of slope prediction methods on slope and erosion estimates, *Applied Engineering in Agriculture, 7 (6), pp.779-783.*

STDS (1990) *The Spatial Data Transfer Standard,* US Government Printing Office, Washington DC: USGS Bulletin.

Steinhaus, H. (1954) Length, shape and area, *Colloquium Mathematicum 3, pp.1-13.*

Steinhaus, H. (1960) *Mathematical Snapshots,* London: Oxford University Press.

Stephenson, G. (1973) *Mathematical Methods for Science Students,* London: Longman.

Strahler, A.N. (1964) Quantitative geomorphology of drainage basins and channel networks, *in* **Chow, V.** (ed) *Handbook of Applied Hydrology,NY: McGraw-Hill, pp.39-76*

Struve, H. (1977) *An Automated Procedure for Slope Map Construction, Final Report on project 4A152121A896, US Army Engineer Waterways Experiment Station, Vicksburg, Miss.*

Tamura, H., Mori, S., and Yamawaki, T. (1978) Textural features corresponding to visual perception. *IEEE Transactions on Systems, Man, and Cybernetics.* 8 (6) *pp.460-473.*

Tang, L. (1992) Automatic extraction of specific geomorphological elements from contours, *Proceedings, 5th International Symposium on Spatial Data Handling, Charleston, SC. Vol 2, pp.554-566.*

Tarboton, D.G., Bras, R.L., and Rodriques-Iturbe, I. (1991) On the extraction of channel networks from digital elevation data, *Hydrological Processes, 5, pp.81-100.*

Thompson, D.W (1917) *On Growth and Form,* Cambridge: Cambridge University Press.

Tobler, W.R. (1969) An analysis of a digitized surface in **Davis, C.M. (ed)** *A Study of the Land Type, Report DA-31-124-ARO-D-456, pp.59-76, Deptartment of Goegraphy, University of Michigan.*

Tomita, F., Shirai, Y., Tsuji, S. (1982) Description of texture by structural analysis, *IEEE Transactions on Pattern Analysis and Machine Intelligence, 4(2), pp.183-191.*

Tomlin C. (1989) *Geographic Information Systems and Cartographic Modelling.*

Tribe, A. (1990). Towards the automated recognition of landforms (valley heads) from digital elevation models. *Proceedings of the 4th International Symposium on Spatial Data Handling, 1990, Zurich, Switzerland.* 1, *pp.45-52.*

Tricart, J. (1947) Sur quelques indices geomorphometriques, *Comptes Rendues hebdomadaires des Seances de l'Academie des Sciences (Paris), 255, pp.747-749.*

Tufte, E.R. (1990) *Envisioning Information*, Cheshire, CN: Graphics Press.

USGS (1987) *Digital Elevation Models Data Users Guide.* Technical Instructions, Data Users Guide 5, National Mapping Program, United States Geological Survey, Reston, VA.

Unwin, D.J. (1975) Introduction to trend surface analysis, *Concepts and Techniques in Modern Geography (CATMOG) 5 pp.40.* Norwich:Geobooks.

Unwin, D.J. (1989) Fractals in the geosciences, *Computers and Geosciences, 15(2) pp.163-165.*

Unwin, D.J. and Wrigley, N. (1987) Towards a general theory of control point distribution effects in trend-surface models, *Computers and Geosciences Vol 13 (4), pp.351-355*

Venig-Meinesz, F.A. (1951) A remarkable feature of the Earth's topography, *Proceedings, K. Ned Akad, Wet Ser B Phys Sci, 54, pp.212-228.*

Voss, R.F. (1985) Random fractal forgeries, *in* **Earnshaw, R.A. (ed)**, *Fundamental Algorithms in Computer Graphics,* NATO ASI Series, F17, New York: Springer-Verlag, *pp.805-835,*

Voss, R.F. (1988) Fractals in nature: From characterisation to simulation, in **Peitgen, H. and Saupe D.** (*eds*) *The Science of Fractal Images, London: Springer-Verlag, pp.21-70.*

Walsh, S.J., Lightfoot, D.R., and Butler, D.R. (1987) Recognition and assessment of error in geographic information systems. *Photogrammetric Engineering and Remote Sensing. 53, pp.1423-1430.*

Warntz, W. (1966). The topology of socio-economic terrain and spatial flows. *Papers of the Regional Science Association. 17, pp.47-61.*

Wartenberg, D. (1985) Multivariate spatial correlation: A method for exploring geographical analysis, *Geographical Analysis, 17(4), pp.263-283.*

Weibel, R. and Heller, M. (1990). A framework for digital terrain modelling, *4th International Symposium on Spatial Data Handling, Zurich, pp.219-229*

Weibel, R. and Heller, M. (1991). *Digital Terrain Modelling.* in: **Maguire, D. J., Goodchild, M. F., and Rhind, D. W.** (eds.) *Geographical Information Systems: Principles and Applications, pp.269-297, Longman, London.*

Werner, C. (1988) Formal analysis of ridge and channel patterns in maturely eroded terrain. *Annals of the Association of American Geographers, 78 (2), pp. 253-270.*

Weszka, J, Dyer, C., Rosenfeld, A. (1976) A comparative study of texture measures for terrain classification, *IEEE Transactions on Systems, Man and Cybernetics, 6(4), pp.269-285.*

Wilkinson, G, Kanellopoulos, I. and Mégier, J. (1995) *(eds) Fractals in Geosciences and Remote Sensing,* Joint Research Centre, Report EUR 16092 EN.

Wolf, G.W. (1984). A mathematical model of cartographic generalization. *Geo-Processing 2, pp.271-286.*

Wolf, G.W. (1989). Data structures for the topological characterisation of topographic surfaces. *Paper presented at the Sixth European Colloquium of Theoretical and Quantitative Geography, Chantilly, France.*

Wolf, G.W. (1991) A Fortran subroutine for cartographic generalisation, *Computers and Geosciences, 17(10), pp.1359-1281.*

Wood, J.D. (1990a). Automated surface feature detection from digital elevation data. Part one: a review and classification. *Midlands Regional Research Report no. 20.*

Wood, J.D. (1990b) Automated surface feature detection from digital elevation data. Part two: Implementation. *Midlands Regional Research Laboratory Research Report no. 21.*

Wood J.D. (1993) Measuring and reporting the accuracy of Ordnance Survey digital elevation data, *Ordnance Survey Technical Report, Ordnance Survey, Southampton.*

Wood, J.D. (1995) Scale-based characterisation of Digital Elevation Models, *Proceedings, 3rd National Conference on GIS Research UK (GISRUK '95), Newcastle.*

Wood J. and Fisher P. (1993) Assessing interpolation accuracy in elevation models, *IEEE Computer Graphics and Applications*, 13 (2), *pp.48-56.*

Wood, J., Unwin, D., Stynes, K., Fisher, P. and Dykes, J. (1996) The use of the landscape metaphor in understanding spatial data, *Environment and Planning B, (in press).*

Wood, W.F. and Snell J.B. (1957) The dispersion of geomorphic data around measures of central tendency and its application, *US Army Quartermaster Research and Development Center, Research Study Report EA-8.*

Wood, W.F. and Snell J.B. (1960) A Quantitative system for classifying landforms, *US Army Quartermaster Research and Development Center, Technical Report EP-124.*

Young, M. (1978) Terrain analysis: program documentation. *Statistical Characterisation of altitude matrices by computer: Report 5 on grant DA-ERO-591-73-G0040*, University of Durham, England.

Yuan, L. P. and Vanderpool N. L. (1986). Drainage network simulation. *Computers and Geosciences* 12 (5), *pp.653-665.*

Yoeli, P. (1967) Mechanisation in analytical hill-shading. *Cartographic Journal* 4, *pp.82-88*

Yoeli, P. (1984) Computer assisted determination of the valley and ridge lines of digital terrain models, *International Yearbook of Cartography, 24, pp.197-206.*

Yoeli, P. (1986) Computer executed production of a regular grid of height points from digital contours, *The American Cartographer 13(3), pp.219-229*

Yokoya, N., Yamamoto, N. and Funakubo, N. (1989) Fractal-based analysis and interpolation of 3D natureal surface shapes and their application to terrain modelling, *Computer Vision, Graphics and Image Processing, 46, pp.284-302*

Zevenbergen, L.W. and Thorne, C.R. (1987) Quantitative analysis of land surface topography. *Earth Surface Processes and Landforms 12, pp.47-56.*

Zhang, M.C., Campbell, J.B. and Haralick, R.M. (1990). Automatic delineation of drainage basins within digital elevation data using the topographic primal sketch. *Mathematical Geology* 22 (2), *pp.189-209.*

Appendices

A.1 Introduction

Most of the work described has relied upon the functionality of the GIS GRASS (Geographical Resources Analysis Support System). GRASS is divided into independent command line *modules* each associated with a particular GIS operation. Modules are divided into 8 categories, each indicated by its prefix.

d prefix — eg. `d.rast`
Commands that relate to graphical display such as displaying rasters, vectors, histograms, clearing screen etc.

g prefix — eg. `g.list`
General file manipulation routines such as the removing of files, renaming files, listing files etc.

i prefix — eg. `i.cluster`
Image processing functionality such as image classification, rectification, principal components analysis etc.

m prefix — eg. `m.tape.examine`
Manipulation of input files such as reading tape header files, extraction of remotely sensed images from tape etc.

p prefix — eg. `p.map`
Paint commands that control output to hardcopy devices.

r prefix — eg. `r.buffer`
Raster based processing - the largest part of GRASS. Processing includes calculation of cost surfaces, map algebra, buffering etc.

s prefix — eg. `s.in.ascii`
Sites or point file manipulation including the interpolation of points values, linking point values to a database etc.

v prefix — eg `v.to.rast`
Vector related functionality, mostly input/output conversion. Includes conversion of Arc/Info files, topology building, digitizing etc.

In many cases existing GRASS functionality has been used in this work, particularly the display related modules and the map algebra module `r.mapcalc`. However, since many new methods of analysis have been developed for this work, it has been necessary to develop

additional modules that can be integrated with existing GRASS functionality. All these additional modules have been written in UNIX C for Irix operating system and take advantage of the GRASS programming libraries (Shapiro *et al*, 1993). The code for these modules is included in these appendices to provide detailed reference to algorithms and methods described. Modules have been grouped into 4 categories depending on their functionality.

1. Data creation modules

`r.frac.surf`	Creates fractal surface(s) as a GRASS raster using spectral synthesis.
`r.gauss.surf`	Creates a normally distributed (Gaussian) random surface with given mean and standard deviation.
`r.xy`	Creates two rasters containing x and y coordinates of cell location (used for creating polynomial surfaces with r.mapcalc).

2. File conversion utilities

`iff2sites`	Reads and converts LaserScan IFF point files into GRASS sites files.
`m.in.dti`	Reads and converts LaserScan raster DTI files into GRASS rasters.
`m.in.ntf`	Reads and converts Ordnance Survey NTF files into GRASS vector and raster files.
`r.to.sites`	Converts non-zero values in a raster into GRASS sites point files.

3. DEM analysis modules

`d.param.scale`	Allows interactive scale based interrogation of DEM parameters.
`r.basin`	Tessellates a DEM into drainage basins. Can also enforce hydrological connectivity.
`r.param.scale`	Calculates morphometric parameters or features at any scale from a DEM.
`v.surf.spline`	Interpolates contours to give a DEM and calculates RMSE for every cell.

4. Output statistics modules

`r.lags`	Calculates autocorrelation and texture analysis statistics at a variety of scales and produces a 'lag-diagram' of the results.
`r.comatrix`	Calculates and displays the co-occurrence matrix and associated measures from any raster map layer.
`r.statistics`	Calculates and reports simple univariate statistics of any raster map layer.

A.2 Data Creation Modules

r.frac.surf

GRASS module to create a fractal surface of a given fractal dimension. Uses spectral synthesis to generate surface. Can generate multiple realisations at different spectral frequencies.

Jo Wood, 19th October, 1994

frac.h

```
/*****************************************************************/
/***                          frac.h                           ***/
/***                Header file for use with r.frac            ***/
/***             Jo Wood, V 1.0  - 19th October, 1994          ***/
/*****************************************************************/

#include "gis.h"                /* This MUST be included in all GRASS   */
                                /* programs. It sets up the necessary   */
                                /* prototypes for GRASS library calls.  */

#define MAX(a,b) ((a)>(b) ? (a):(b))
#define SWAP(a,b) tempr=(a); (a)=(b); (b) = tempr

#define TWOPI 6.283185307179590 /* Constant value of 2 pi */

/* ------ Global variables ------ */

#ifndef MAIN
    extern                      /* Externally defined if not main()     */
#endif


char    *rast_out_name,         /* Name of the raster output file.      */
        *mapset_out;

#ifndef MAIN
    extern
#endif

int     fd_out,                 /* File descriptor of output raster     */
        Steps;                  /* Number of intermediate images.       */

#ifndef MAIN
    extern
#endif
double  H;                      /* Hausdorff-Besickovitch dimension.    */
```

main.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                        main() for r.frac                              ***/
/***         GRASS module to manipulate a raster map layer.                ***/
/***                              Jo Wood                                  ***/
/***                    v 1.0  19th October, 1994                          ***/
/***                                                                       ***/
/*****************************************************************************/

#define MAIN

#include "frac.h"

CELL main(argc,argv)
    int argc;
    char *argv[];
{
    /*---------------------------------------------------------------------*/
    /*                       GET INPUT FROM USER                           */
    /*---------------------------------------------------------------------*/

    interface(argc,argv);


    /*---------------------------------------------------------------------*/
    /*                      PROCESS RASTER FILES                           */
    /*---------------------------------------------------------------------*/

    process();

}
```

interface.c

```
/*****************************************************************************/
/***      Function to get input from user and check files can be opened    ***/
/***                                                                       ***/
/***                     Jo Wood,  V1.0, 13th September 1994               ***/
/*****************************************************************************/

#include "frac.h"

interface(argc,argv)

    int      argc;              /* Number of command line arguments     */
    char    *argv[];            /* Contents of command line arguments.  */

{
    /*---------------------------------------------------------------*/
    /*                              INITIALISE                       */
    /*---------------------------------------------------------------*/
```

```
    struct Option       *rast_out;      /* Structure for output raster  */

    struct Option       *frac_dim;      /* Fractal dimension of surface */

    struct Option       *num_images;    /* Number of images to produce. */

    G_gisinit (argv[0]);                /* Link with GRASS interface.   */


    /*--------------------------------------------------------------------*/
    /*                       SET PARSER OPTIONS                           */
    /*--------------------------------------------------------------------*/

    rast_out = G_define_option();
    frac_dim = G_define_option();
    num_images = G_define_option();

    /* Each option needs a 'key' (short description),
                          a 'description` (a longer one),
                          a 'type' (eg intiger, or string),
                      and an indication whether manditory or not */

    rast_out->key         = "out";
    rast_out->description = "Name of fractal surface raster layer";
    rast_out->type        = TYPE_STRING;
    rast_out->required    = YES;

    frac_dim->key         = "d";
    frac_dim->description = "Fractal dimension of surface (2 < D < 3)";
    frac_dim->type        = TYPE_DOUBLE;
    frac_dim->required    = NO;
    frac_dim->answer      = "2.05";

    num_images->key         = "n";
    num_images->description = "Number of intermediate images to produce";
    num_images->type        = TYPE_INTEGER;
    num_images->required    = NO;
    num_images->answer      = "0";

    if (G_parser(argc,argv))            /* Performs the prompting for   */
        exit(-1);                       /* keyboard input.              */

    rast_out_name  = rast_out->answer;
    sscanf(frac_dim->answer,"%lf",&H);
    H = 3.0 - H;
    Steps = atoi(num_images->answer) + 1;


    /*--------------------------------------------------------------------*/
    /*               CHECK OUTPUT RASTER FILE DOES NOT EXIST              */
    /*--------------------------------------------------------------------*/

    mapset_out = G_mapset();            /* Set output to current mapset */

    if (G_legal_filename(rast_out_name)==NULL)
```

```
    {
        char err[256];
        sprintf(err,"Illegal output file name. Please try another.");
        G_fatal_error(err);
    }
    else
    {
        if (G_find_cell2(rast_out_name,mapset_out) !=NULL)
        {
            char err[256];
            sprintf(err,"Raster map [%s] exists.\nPlease try another\n",
                                                        rast_out_name);
            G_fatal_error(err);
        }
    }

    /*--------------------------------------------------------------------*/
    /*              CHECK FRACTAL DIMENSION IS WITHIN LIMITS               */
    /*--------------------------------------------------------------------*/

    if ( (H <= 0) || (H >= 1))
    {
        char err[256];
        sprintf(err,"Fractal dimension of [%.2lf] must be between 2 and 3.",
                                                                3.0-H);
        G_fatal_error(err);
    }
}
```

open_files.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                            open_files()                               ***/
/***          Opens input and output raster files for r.frac.surf          ***/
/***                   Jo Wood, V1.0, 13th September, 1994                 ***/
/***                                                                       ***/
/*****************************************************************************/

#include "frac.h"

open_files()
{
    /* Open new file and set the output file descriptor. */

    if ( (fd_out=G_open_cell_new(rast_out_name)) <0)
    {
        char err[256];
        sprintf(err,"ERROR: Problem opening output file.");
        G_fatal_error(err);
    }

}
```

process.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                              process()                                ***/
/***          Reads in a raster files row by row for processing            ***/
/***                   Jo Wood, V1.0, 13th September, 1994                 ***/
/***                                                                       ***/
/*****************************************************************************/

#include "frac.h"

process()
{
    /*-------------------------------------------------------------------*/
    /*                            INITIALISE                             */
    /*-------------------------------------------------------------------*/

    int         nrows,          /* Will store the current number of     */
                ncols,          /* rows and columns in the raster.      */

                nn,             /* Size of raster to nearest power of 2.*/

                row,col;        /* Counts through each row and column   */
                                /* of the input raster.                 */

    double      *data[2];       /* Array holding complex data.          */


    /*-------------------------------------------------------------------*/
    /*                       GET DETAILS OF INPUT RASTER                 */
    /*-------------------------------------------------------------------*/

    nrows = G_window_rows();          /* Find out the number of rows and */
    ncols = G_window_cols();          /* columns of the raster view.     */

    nn = max_pow2(MAX(nrows,ncols)); /* Find smallest power of 2 that   */
                                     /* largest side of raster will fit.*/

    /*-------------------------------------------------------------------*/
    /*                      CREATE SQUARE ARRAY OF SIDE 2^n              */
    /*-------------------------------------------------------------------*/

    data[0] = (double *) G_malloc(nn*nn*sizeof(double));
    data[1] = (double *) G_malloc(nn*nn*sizeof(double));

    /*-------------------------------------------------------------------*/
    /*                   Apply spectral synthesis algorithm.             */
    /*-------------------------------------------------------------------*/

    specsyn(data,nn);

    free(data[0]);
    free(data[1]);
}
```

complex.c

```
/*****************************************************/
/*  Initialize real & complex components to zero   */
/*****************************************************/

data_reset(data,nn)
    double *data[2];
    int    nn;
{
    register double *dptr0=data[0], *dptr1=data[1];
             int    total_size = nn*nn,
                    count;

    for (count=0; count<total_size; count++)
        *dptr0++ = *dptr1++ =0.0;
}
```

gauss.c

```
/*********************************************************************/
/***  Random Number Generator (Gaussian:  mean=0.0 sigma=1.0) ***/
/***                                                           ***/
/***                                      [Press et al, 1988]  ***/
/***                                                           ***/
/*** Coded  Oct 23 1991                                       ***/
/*** Version 1.0                                              ***/
/***                                                           ***/
/*********************************************************************/

#include <math.h>

float gauss(seed)
int     seed;
{
        static int      iset=0;
        static float    gset;
        float           fac,r,v1,v2;
        float           rand1();

        if (iset==0)
        {
                do
                {
                        v1=2.0*rand1(seed)-1.0;
                        v2=2.0*rand1(seed)-1.0;
                        r=v1*v1+v2*v2;
                }
                while (r>=1.0);

                fac=sqrt(-2.0*log(r)/r);
                gset=v1*fac;
                iset=1;
```

```
                return(v2*fac);
        }
        else
        {
                iset=0;
                return gset;
        }
}
```

rand1.c

```
/*************************************************************/
/***    Random Number Generator (Uniform Deviates 0.0 -> 1.0) ***/
/***                                                          ***/
/*** Based on three linear congruential generators:          ***/
/***  One for most significant part,                         ***/
/***  One for the least significant,                         ***/
/***  One for a shuffling routine (Knuth, 1981).             ***/
/***                                    [Press et al, 1988]   ***/
/***                                                          ***/
/*** Coded  Oct 23 1991                                      ***/
/*** Version 1.0                                             ***/
/***                                                          ***/
/*************************************************************/

                        /* Arbitrary constants */
#define M1 259200
#define IA1 7141
#define IC1 54773
#define RM1 (1.0/M1)
#define M2 134456
#define IA2 8121
#define IC2 28411
#define RM2 (1.0/M2)
#define M3 243000
#define IA3 4561
#define IC3 51349


float rand1(seed)
int     seed;
{
        static long     ix1,ix2,ix3;
        float           temp;
        int             j;
        static float    r[98];
        static int      iff=0;

        if (seed < 0 || iff==0) /* Initialise if negative seed is given. */
                                /* This MUST be done on the first call.  */
        {
                iff=1;
                ix1=(IC1-seed) % M1;            /* Seed the 1st routine */
```

```
                ix1=(IA1*ix1+IC1) % M1; /* and use it to seed the */
                ix2=ix1 % M2;           /* 2nd and 3rd routines. */
                ix1=(IA1*ix1+IC1) % M1;
                ix3=ix1 % M3;

                for (j=1; j<=97; j++)
                {
                        ix1=(IA1*ix1+IC1) % M1;  /* Fill table with sequen- */
                        ix2=(IA2*ix2+IC2) % M2;  /* tial uniform deviates   */
                        r[j]=(ix1+ix2*RM2)*RM1;  /* from the 1st 2 routines */
                }
                seed=1;
        }

        ix1=(IA1*ix1+IC1) % M1; /* Generate the next number in the */
        ix2=(IA2*ix2+IC2) % M2; /* sequence of routine             */
        ix3=(IA3*ix3+IC3) % M3;

        j=1 + ((97*ix3) / M3);  /* Get random position ion array */
        temp = r[j];
        r[j]=(ix1+ix2*RM2)*RM1;        /* Refil array and return entry */
        return(temp);
}
```

spec_syn.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                              spec_syn()                               ***/
/***          Creates a fractal surface using spectral synthesis.          ***/
/***        Algorithm adapted from Peitgen and Sauper (1988), p.108.       ***/
/***                    Jo Wood, V1.0, 19th October, 1994                  ***/
/***       Modified to allow multiple realisations of same surface,        ***/
/***                    Jo Wood, V1.1 15th October, 1995.                  ***/
/***                                                                       ***/
/*****************************************************************************/

#include "frac.h"
#include <math.h>


specsyn(data,nn)
    double *data[2];    /* Array holding complex data to transform. */
    int    nn;          /* Size of array in one dimension.          */

{
    /* ---------------------------------------------------------- */
    /* ---                      Initialise                    --- */
    /* ---------------------------------------------------------- */

    int   row,col,              /* Counts through half the data array.  */
          row0,col0,            /* 'other half' of the array.           */
          coeff,                /* No. of Fourier coeffficents to calc. */
```

```
      seed;                    /* Random number seed.                    */

float randl(),                 /* Random number generators.              */
      gauss();



double phase,rad,              /* polar coordinates of Fourier coeff.    */
       *temp[2];

seed = -1*getpid();
randl(seed);                   /* Reset random number generator.         */

temp[0] = (double *) G_malloc(nn*nn*sizeof(double));
temp[1] = (double *) G_malloc(nn*nn*sizeof(double));

/* ------------------------------------------------------------------- */
/* ---                    Create fractal surface                   --- */
/* ------------------------------------------------------------------- */


/* Calculate all the preliminary random coefficients. */
/* ================================================== */

fprintf(stderr,"Preliminary surface calculations.\n");
data_reset(data,nn);

for (row=0; row <= nn/2; row++)
    for (col=0; col<= nn/2; col++)
    {
        /* Generate random Fourier coefficients. */

        phase = TWOPI*randl();

        if ((row != 0) || (col != 0))
            rad = pow(row*row + col*col,-(H+1)/2.0)*gauss();
        else
            rad = 0.0;

        /* Fill half the array with coefficients. */

        *(data[0] + row*nn + col) = rad * cos(phase);
        *(data[1] + row*nn + col) = rad * sin(phase);

        /* Fill other half of array with coefficients. */

        if (row == 0)
            row0 = 0;
        else
            row0 = nn-row;

        if (col == 0)
            col0 = 0;
        else
            col0 = nn-col;
```

```
        *(data[0] + row0*nn + col0) =  rad * cos(phase);
        *(data[1] + row0*nn + col0) = -rad * sin(phase);
    }

*(temp[1] + nn/2 )             =  0.0;
*(temp[1] + nn * nn/2 )        =  0.0;
*(temp[1] + nn * nn/2 + nn/2) =  0.0;

for (row=1; row < nn/2; row++)
    for (col=1; col< nn/2; col++)
    {
        phase = TWOPI*randl();
        rad   = pow(row*row + col*col,-(H+1)/2.0)*gauss();

        *(data[0] + row*nn + nn-col) = rad * cos(phase);
        *(data[1] + row*nn + nn-col) = rad * sin(phase);

        *(data[0] + (nn-row)*nn + col) =  rad * cos(phase);
        *(data[1] + (nn-row)*nn + col) = -rad * sin(phase);

    }


/* Transfer random coeffients to array before ifft transform */
/* ========================================================= */

for (coeff=0; coeff < Steps; coeff++)
{
    fprintf(stderr,"Calculating surface %d (of %d)\n",coeff+1,Steps);
    data_reset(temp,nn);

    for (row=0; row <= (coeff+1)*nn/(Steps*2); row++)
        for (col=0; col<= (coeff+1)*nn/(Steps*2); col++)
        {
            if (row == 0)
                row0 = 0;
            else
                row0 = nn-row;

            if (col == 0)
                col0 = 0;
            else
                col0 = nn-col;

            *(temp[0] + row*nn + col) = *(data[0] + row*nn + col);
            *(temp[1] + row*nn + col) = *(data[1] + row*nn + col);

            *(temp[0] + row0*nn + col0) = *(data[0] + row0*nn + col0);
            *(temp[1] + row0*nn + col0) = *(data[1] + row0*nn + col0);
        }

     for (row=1; row < (coeff+1)*nn/(Steps*2); row++)
        for (col=1; col< (coeff+1)*nn/(Steps*2); col++)
        {
            *(temp[0] + row*nn + nn-col) = *(data[0] + row*nn + nn-col);
```

```
                *(temp[1] + row*nn + nn-col) = *(data[1] + row*nn + nn-col);

                *(temp[0] + (nn-row)*nn + col) = *(data[0]+(nn-row)*nn + col);
                *(temp[1] + (nn-row)*nn + col) = *(data[1]+(nn-row)*nn + col);
            }

        fft(1,temp,nn*nn,nn,nn);                /* Perform inverse FFT and */
        write_rast(temp,nn,coeff+1);            /* write out raster.       */
    }

    /* Free memory. */
    /* ============ */

    free(temp[0]);
    free(temp[1]);
}
```

min_length.c

```
/*****************************************************************************/
/* MAX_POW2 : finds least power of 2 greater than or equal to number     ***/
/* Input arguments: n - unsigned integer, the number                     ***/
/* Output is an integer power of 2                                       ***/
/*****************************************************************************/

max_pow2(n)
int n;
{
    int p2, n1;

    n1 = n >> 1;
    p2 = 1;

    while (n1 > 0)
    {
        n1 >>= 1;
        p2 <<= 1;
    }
    if (p2 < n)
        p2 <<=1;

    return(p2);
}
```

fft.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                                fft()                                  ***/
/*** Calculactes the fast Fourier transform of an array of real numbers   ***/
/***   Algorithm derived from Cooley et al (1969).                         ***/
/***   Implemented in Press et al (1988), Numerical Recipes in C.          ***/
/***   GRASS coding, Ali R. Vali, University of Texas.                     ***/
/***   Modified by David B. Satnik, Central Washington University,         ***/
/***   Minor modifications,  Jo Wood, V1.0, 19th October, 1994             ***/
/***                                                                       ***/
/*****************************************************************************/

#include "frac.h"
#include <math.h>

/*****************************************************************************/
/*                                                                           */
/* Fast Fourier Transform Routine for two-dimensional array                  */
/* Adapted from N.M. Brenner Algorithm (Numerical Recipes in C               */
/*  pp. 407-412)                                                             */
/*                                                                           */
/* Input args: i_sign - direction of transform :                             */
/*                      -1 -> transform, +1 -> inverse transform             */
/*               DATA - pointer to a complex linear array in row major       */
/*                      order containing the data and the result.            */
/*               NN -   value of DATA dimension                              */
/*               dimc - value of image column dimension (max power of 2)     */
/*               dimr - value of image row dimension (max power of 2)        */
/*                                                                           */
/*****************************************************************************/

fft(i_sign,DATA,NN,dimc,dimr)
int    i_sign;
double *DATA[2];
int   dimc, dimr, NN;
{
    int    i, j, ip1, ip2, i1, i2, i3, i2rev, i3rev, ibit, ifp1, ifp2, k1, k2;
    int    N, Nprev, Nrem;
    double tempr, tempi, wr, wi, wpr, wpi, wtemp, theta, norm;
    double swr, swi;

    /* Apply normalization factor */

    norm = 1.0 / sqrt((double)NN);
    for (i = 0; i < NN; i++)
      for (j = 0; j < 2; j++)
        *(DATA[j]+i) *= norm;

    /* Begin FFT algorithm */
    Nprev = 1;
    N = dimr;

    for (i = 0; i < 2; i++)
```

```
{
    Nrem = NN / (N * Nprev);
    ip1 = Nprev * N;

    /* This is the bit reversal section */

    i2rev = 0;
    for (i2 = 0; i2 < ip1; i2 += Nprev)
    {
        if (i2 < i2rev)
        {
            for (i1 = i2; i1 < (i2+Nprev); i1++)
                for (i3 = i1; i3 < NN; i3 += ip1)
                {
                    i3rev = i2rev + i3 - i2;
                    tempr = *(DATA[0]+i3);
                    tempi = *(DATA[1]+i3);
                    *(DATA[0]+i3) = *(DATA[0]+i3rev);
                    *(DATA[1]+i3) = *(DATA[1]+i3rev);
                    *(DATA[0]+i3rev) = tempr;
                    *(DATA[1]+i3rev) = tempi;
                }
        }   /* end if */
        ibit = ip1 >> 1;
        while ((ibit >= Nprev) && (i2rev >= ibit))
        {
            i2rev -= ibit;
            ibit >>= 1;
        }
        i2rev += ibit;
    }   /* end bit reversal section */

    /* Begin Danielson-Lanczos section of algorithm */

    ifp1 = Nprev;
    while (ifp1 < ip1)
    {
        ifp2 = 2 * ifp1;
        theta = i_sign * TWOPI / (ifp2 / Nprev);
        wpr = cos(theta);
        wpi = sin(theta);
        wr = 1.0;
        wi = 0.0;
        for (i3 = 0; i3 < ifp1; i3 += Nprev)
        {
            for (i1 = i3; i1 < (i3+Nprev); i1++)
                for (i2 = i1; i2 < NN; i2 += ifp2)

                /* Danielson-Lanczos formula */
                {
                    k1 = i2;
                    k2 = k1 + ifp1;
                    swr = wr;
                    swi = wi;
                    tempr = swr * *(DATA[0]+k2) - swi * *(DATA[1]+k2);
```

```
                        tempi = swr * *(DATA[1]+k2) + swi * *(DATA[0]+k2);
                        *(DATA[0]+k2) = *(DATA[0]+k1) - tempr;
                        *(DATA[1]+k2) = *(DATA[1]+k1) - tempi;
                        *(DATA[0]+k1) = *(DATA[0]+k1) + tempr;
                        *(DATA[1]+k1) = *(DATA[1]+k1) + tempi;
                    }

                /* trigonometric recurrence */
                wtemp = wr;
                wr = wr * wpr - wi * wpi;
                wi = wi * wpr + wtemp * wpi;
            }
            ifp1 = ifp2;

        }

        Nprev = N;
        N = dimc;
    }

    return(0);
}
```

write_rast.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                            write_rast()                               ***/
/***  Extracts real component from complex array and writes as raster.     ***/
/***                  Jo Wood, V1.0, 20th October, 1994                    ***/
/***                                                                       ***/
/*****************************************************************************/

#include "frac.h"

write_rast(data,nn,step)
    double *data[2];            /* Array holding complex data.           */
    int    nn,                  /* Size of side of array.                */
           step;                /* Version of file to send.              */
{

    /*-----------------------------------------------------------------*/
    /*                            INITIALISE                           */
    /*-----------------------------------------------------------------*/

    CELL        *row_out;       /* Buffers to hold raster rows.          */

    char        file_name[128]; /* Name of each file to be written       */

    int         nrows,          /* Will store the current number of      */
                ncols,          /* rows and columns in the raster.       */

                row,col;        /* Counts through each row and column    */
                                /* of the input raster.                  */
```

```
    nrows = G_window_rows();     /* Find out the number of rows and      */
    ncols = G_window_cols();     /* columns of the raster view.          */

    row_out = G_allocate_cell_buf();

    /*--------------------------------------------------------------------*/
    /*         Open new file and set the output file descriptor.          */
    /*--------------------------------------------------------------------*/

    if (Steps != step)
        sprintf(file_name,"%s.%d",rast_out_name,step);
    else
        strcpy(file_name,rast_out_name);

    if ( (fd_out=G_open_cell_new(file_name)) <0)
    {
        char err[256];
        sprintf(err,"ERROR: Problem opening output file.");
        G_fatal_error(err);
    }

    /*--------------------------------------------------------------------*/
    /*  Extract real component of transform and save as a GRASS raster. */
    /*--------------------------------------------------------------------*/

    for(row=0; row<nrows; row++)
    {
        for(col = 0; col < ncols; col++)
            *(row_out + col) = (CELL) (*(data[0] + row*nn + col) *100000);

        G_put_map_row(fd_out,row_out);
    }

    G_close_cell(fd_out);
}
```

r.gauss.surf

GRASS module to produce a raster map layer of gaussian deviates whose mean and standard deviation can be expressed by the user. It is essentialy the same as r.rand.surf, but uses a gaussian random number generator instead. Both random number generators are taken from Press, Flannery, Teukolsky and Vetterling (1988) - Numerical Recipes in C.

Jo Wood, 24th October, 1991

main.c

```
/*
** Code Compiled by Jo Wood [JWO] 24th October 1991
** Department of Geography, university of Leicester
**
*/

#include "gis.h"

main(argc,argv)
    int argc;
    char *argv[];
{

    /****** INITIALISE ******/

    double          gauss_mean,gauss_sigma;


    struct Option   *out;       /* Structures required for G_parser()   */
                                /* call. These can be filled with the   */
    struct Option   *mean;      /* various defaults, mandatory paramtrs */
                                /* etc. for the GRASS user interface.   */
    struct Option   *sigma;

    G_gisinit (argv[0]);        /*      This GRASS library function MUST
                                        be called first to check for valid
                                        database and mapset. As usual argv[0]
                                        is the program name. This can be
                                        recalled using G_program_name(). */

    /****** SET PARSER OPTIONS ******/

    out   = G_define_option();  /* Request pointer for each option.     */
    mean  = G_define_option();  /* Mean of the distribution             */
    sigma = G_define_option();  /* Standard deviation of distribution.  */

    out->key            = "out";
    out->description    = "Name of the random surface to be produced";
    out->type           = TYPE_STRING;
    out->required       = YES;
```

```
    mean->key           = "mean";
    mean->description   = "Distribution mean";
    mean->type          =  TYPE_DOUBLE;
    mean->answer        = "0.0";

    sigma->key          = "sigma";
    sigma->description  = "Standard deviation";
    sigma->type         = TYPE_DOUBLE;
    sigma->answer       = "1.0";


    if (G_parser(argc,argv))
        exit(-1);               /*      Returns a 0 if sucessful        */

    sscanf(mean->answer,"%lf",&gauss_mean);
    sscanf(sigma->answer,"%lf",&gauss_sigma);


    /****** CHECK THE CELL FILE (OUT) DOES NOT ALREADY EXIST******/

    if (G_legal_filename(out->answer)==NULL)
    {
        char err[256];
        sprintf(err,"Illegal file name. Please try another.");
        G_fatal_error(err);
    }
    else
    {
        if (G_find_cell(out->answer,"") !=NULL)
        {
            char err[256];
            sprintf(err,
                "Raster map [%s] already exists.\nPlease try another.",out);
            G_fatal_error(err);
        }

    }

    /****** CREATE THE RANDOM CELL FILE  ******/

    gaussurf(out->answer,gauss_mean,gauss_sigma);


}
```

gaussurf.c

```
/*****************/
/** gaussurf()  **/
/*****************/

#include "gis.h"
#include <math.h>

gaussurf(out,mean,sigma)

char    *out;                   /* Name of cell files to be opened.     */
double  mean,sigma;             /* Distribution parameters.             */

{

        int     nrows,ncols;    /* Number of cell rows and columns      */



        CELL    *row_out;       /* Buffer just large enough to hold one */
                                /* row of the raster map layer.         */

        int     fd_out;         /* File descriptor - used to identify   */
                                /* open cell files.                     */


        int     row_count,col_count;

        float   rand1(),gauss();

        /****** INITIALISE RANDOM NUMBER GENERATOR ******/

        rand1(-1* getpid());

        /****** OPEN CELL FILES AND GET CELL DETAILS ******/

        fd_out = G_open_cell_new(out);

        nrows = G_window_rows();
        ncols = G_window_cols();

        row_out = G_allocate_cell_buf();


        /****** PASS THROUGH EACH CELL ASSIGNING RANDOM VALUE ******/

        for (row_count=0;row_count<nrows;row_count++)
        {
            for (col_count=0;col_count<ncols;col_count++)
                *(row_out+col_count) = rint((gauss(2742)*sigma)+mean);
                        /* NB: Must use rint() otherwise truncating
                               will double frequency of 0 cells. */

            /* Write contents row by row */
```

```
            G_put_map_row(fd_out,row_out);
        }


        /****** CLOSE THE CELL FILE ******/

        G_close_cell(fd_out);


}
```

gauss.c

```
/*************************************************************/
/***  Random Number Generator (Gaussian:  mean=0.0 sigma=1.0) ***/
/***                                                         ***/
/***                                    [Press et al, 1988]  ***/
/***                                                         ***/
/*** Coded  Oct 23 1991                                     ***/
/*** Version 1.0                                            ***/
/***                                                         ***/
/*************************************************************/

#include <math.h>

float gauss(seed)
int     seed;
{
        static int      iset=0;
        static float    gset;
        float           fac,r,v1,v2;
        float           randl();

        if (iset==0)
        {
                do
                {
                        v1=2.0*randl(seed)-1.0;
                        v2=2.0*randl(seed)-1.0;
                        r=v1*v1+v2*v2;
                }
                while (r>=1.0);

                fac=sqrt(-2.0*log(r)/r);
                gset=v1*fac;
                iset=1;

                return(v2*fac);
        }
        else
        {
                iset=0;
                return gset;
        }
}
```

rand1.c

```
/*************************************************************/
/***    Random Number Generator (Uniform Deviates 0.0 -> 1.0) ***/
/***                                                         ***/
/*** Based on three linear congruential generators:          ***/
/***  One for most significant part,                         ***/
/***  One for the least significant,                         ***/
/***  One for a shuffling routine (Knuth, 1981).             ***/
/***                                    [Press et al, 1988]  ***/
/***                                                         ***/
/*** Coded  Oct 23 1991                                     ***/
/*** Version 1.0                                            ***/
/***                                                         ***/
/*************************************************************/

                        /* Arbitrary constants */
#define M1 259200
#define IA1 7141
#define IC1 54773
#define RM1 (1.0/M1)
#define M2 134456
#define IA2 8121
#define IC2 28411
#define RM2 (1.0/M2)
#define M3 243000
#define IA3 4561
#define IC3 51349


float rand1(seed)
    int  seed;
{
        static long     ix1,ix2,ix3;
        float           temp;
        int             j;
        static float    r[98];
        static int      iff=0;

        if (seed < 0 || iff==0)         /* Init. if -ve seed is given.  */
                                        /* MUST be done on first call.  */
        {
                iff=1;
                ix1=(IC1-seed) % M1;            /* Seed the 1st routine */
                ix1=(IA1*ix1+IC1) % M1;         /* and use it to seed   */
                ix2=ix1 % M2;                   /* 2nd and 3rd routines */
                ix1=(IA1*ix1+IC1) % M1;
                ix3=ix1 % M3;

                for (j=1; j<=97; j++)
                {
                        ix1=(IA1*ix1+IC1) % M1;  /* Fill table with sequen- */
                        ix2=(IA2*ix2+IC2) % M2;  /* tial uniform deviates   */
                        r[j]=(ix1+ix2*RM2)*RM1;  /* from the 1st 2 routines */
                }
```

```
            seed=1;
        }

        ix1=(IA1*ix1+IC1) % M1; /* Generate the next number in the */
        ix2=(IA2*ix2+IC2) % M2; /* sequence of routine             */
        ix3=(IA3*ix3+IC3) % M3;

        j=1 + ((97*ix3) / M3);          /* Get random position in array */
        temp = r[j];
        r[j]=(ix1+ix2*RM2)*RM1;         /* Refil array and return entry */
        return(temp);
}
```

r.xy

GRASS module to produce two raster maps for use with r.mapcalc. One file contains the x-coordinates of each raster map cell, the other, the y-coordinates. These two cells can be used to produce mathematical functions in the form:

$$z = fn(x,y)$$

NOTE: x and y values are in RELATIVE coordinates to an origin of (0,0) at the bottom left corner. To transfrom back to georeferenced coordinates use r.mapcalc to add the relative offset to the origin.

Jo Wood, Department of Geography, 28th October 1991

main.c

```
/*
** Code Compiled by Jo Wood [JWO] 26th October 1991
** Midlands Regional Research Laboratory (ASSIST)
**
**
*/

#include "gis.h"

main(argc,argv)
    int argc;
    char *argv[];
{

    /****** INITIALISE ******/


    struct Option *x;    /* Structures required for the G_parser()       */
                         /* call. These can be filled with the           */
    struct Option *y;    /* various defaults, mandatory paramters        */
                         /* etc. for the GRASS user interface.           */

    G_gisinit (argv[0]);         /*      This GRASS library function MUST
                                      be called first to check for valid
                                      database and mapset. As usual argv[0]
                                      is the program name. This can be
                                      recalled using G_program_name(). */

    /****** SET PARSER OPTIONS ******/

    x  = G_define_option();      /* Request pointer for each option.     */
    y  = G_define_option();

    x->key              = "x";
    x->description      = "Name of raster map layer to hold x coord values";
    x->type             = TYPE_STRING;
```

```
    x->required = YES;

    y->key              = "y";
    y->description      = "Name of raster map layer to hold y coord values";
    y->type             = TYPE_STRING;
    y->required = YES;


    if (G_parser(argc,argv))
        exit(-1);               /*      Returns a 0 if sucessful        */


    /****** CHECK THE CELL FILES (X & Y) DO NOT ALREADY EXIST******/

    if (G_legal_filename(x->answer)==NULL)
    {
        char err[256];
        sprintf(err,"Illegal file name. Please try another.");
        G_fatal_error(err);
    }
    else
    {
        if (G_find_cell(x->answer,"") !=NULL)
        {
            char err[256];
            sprintf(err,
                "Raster [%s] already exists.\nPlease try another.",x->answer);
            G_fatal_error(err);
        }

    }

    if (G_legal_filename(y->answer)==NULL)
    {
        char err[256];
        sprintf(err,"Illegal file name. Please try another.");
        G_fatal_error(err);
    }
    else
    {
        if (G_find_cell(y->answer,"") !=NULL)
        {
            char err[256];
            sprintf(err,
                "Raster [%s] already exists.\nPlease try another.",y->answer);
            G_fatal_error(err);
        }

    }

    /****** CREATE THE X and Y CELL FILES  ******/

    createxy(x->answer,y->answer);
}
```

create_xy.c

```
/*****************/
/** createxy()  **/
/*****************/

#include "gis.h"

createxy(xname,yname)

char    *xname,*yname;          /* Name of cell files to be opened.     */

{
        int     nrows,ncols;    /* Number of cell rows and columns      */

        CELL    *x_out;         /* Buffer just large enough to hold one */
        CELL    *y_out;         /* row of the raster map layer.         */

        int     fd_xout;        /* File descriptor - used to identify   */
        int     fd_yout;        /* open cell files.                     */


        int     row_count,col_count;

        struct Cell_head        *window;

        struct Categories       xcats;
        struct Categories       ycats;


        int     northing;

        char    label[128];

        /****** OPEN CELL FILES AND GET CELL DETAILS ******/


        fd_xout = G_open_cell_new(xname);
        fd_yout = G_open_cell_new(yname);

        nrows = G_window_rows();
        ncols = G_window_cols();

        x_out = G_allocate_cell_buf();
        y_out = G_allocate_cell_buf();

        G_init_cats( (CELL) ncols, "Column Positions", &xcats);
        G_init_cats( (CELL) nrows, "Row Positions", &ycats);


        /****** PASS THROUGH EACH CELL ASSIGNING COORDINATE VALUE ******/

        /* Fill up the buffer of x values */
```

```
    for (col_count=0;col_count<ncols;col_count++)
    {
        *(x_out+col_count) =col_count;
                    /* NOTE: Call to GRASS function should be
                       G_col_to_easting((double)col_count+0.5,window);
                       but error in library. */

        sprintf(label,"%f",(float)col_count);

        G_set_cat (col_count,label,&xcats);
    }


    for (row_count=0;row_count<nrows;row_count++)
    {

        /* Find out what the y coordinate will be for each new row */

        northing = row_count;
                    /* NOTE: Call to GRASS function should be
                       G_row_to_northing((double)row_count+0.5,window);
                       but error in library. */

        sprintf(label,"%f",(float)row_count);

        G_set_cat(row_count,label,&ycats);


        /* Fill up the buffer of y values */

        for (col_count=0;col_count<ncols;col_count++)
            *(y_out+col_count) = northing;

        /* Write contents row by row */
        G_put_map_row(fd_xout,x_out);
        G_put_map_row(fd_yout,y_out);

    }

    /****** CLOSE THE CELL FILES ******/

    G_close_cell(fd_xout);
    G_close_cell(fd_yout);


    /****** WRITE CATEGORY SUPPORT FILES ******/

    G_write_cats(xname,&xcats);
    G_write_cats(yname,&ycats);

    G_free_cats(&xcats);
    G_free_cats(&ycats);
}
```

A.3 File Conversion Modules

iff2sites

Program to convert LaserScan IFF point files into ASCII (x,y,z) format for input into GRASS as 'sites' files..

Jo Wood, Department of Geography, 29th July, 1993.

main.c

```
/*****************************************************************************/
/** IFF2sites - Program to convert IFF text files into x,y,z text files **/
/** V0.9 Jo Wood, Written at Ordnance Survey, Southampton 29th July '93 **/
/**                                                                         **/
/*****************************************************************************/

#include <stdio.h>
#define CONTROL_POINT -9999

main(argc,argv)
    int argc;
    char *argv[];
{

    FILE        *fptr_in, *fptr_out;    /* Input and output files */

    char        text[128];              /* Line of text from file */

    float       x,y,
                x_orig,y_orig,
                z;                      /* Coordinates and elevation */

    int         dummy;                  /* Dummy variable */

    /* ------ Cheeck input and output file names have been given. ------ */

    if (argc < 3)
    {
        fprintf(stderr,"USAGE: iff2xyz <iff_file> <xyz_file> [G]\n");
        exit(-1);
    }

    if ( (fptr_in=fopen(argv[1],"r")) == NULL)
    {
        fprintf(stderr,
                "ERROR: Can't open IFF text file [%s] for reading\n",argv[1]);
        exit(-2);
    }

    if ( (fptr_out=fopen(argv[2],"w")) == NULL)
```

```
{
    fprintf(stderr,
            "ERROR: Can't open xyz text file [%s] for writing\n",argv[2]);
    exit(-2);
}


/* ------ Process Input file ------ */


/* ------ Seach Header for Local Origin (LO) ------*/

do
{
    if (fgets(text,sizeof(text),fptr_in) == NULL)
                                    /* <---- Scan each line in file */
        fprintf(stderr,"ERROR: No Local Origin found\n");
}
while (strncmp(text+4,"LO",2) != 0);
                                    /* <---- Look for Local Origin line */

sscanf(text+6,"%f %f",&x_orig,&y_orig);



while (fgets(text,sizeof(text),fptr_in) != NULL)
                                    /* <---- Scan each line in file */
{
    if (strncmp(text,"   NA",5) == 0)
                                    /* <---- Look for line before
                                                        attribute */
    {
        fgets(text,sizeof(text),fptr_in);
                                    /* <---- Read z value */
        if (strncmp(text,"   AT",5) == 0)
            sscanf(text+5,"%d %f",&dummy,&z);
        else
            z= CONTROL_POINT;

        do
        {
            fgets(text,sizeof(text),fptr_in);
                                    /* <---- Check we haven't reached
                                                    end of record */
            if (strncmp(text,"EF",2) == 0)
                fprintf(
    "WARNING: 'EF' line found between attributes and coordinates.\n
    Possible attribute mismatch.\n");
        }
        while(strncmp(text,"   DA",5) != 0);
                                    /* <---- Read in (x,y) coordinates */
        sscanf(text+5,"%f %f",&x,&y);

        /* Send to output file */
```

```
            if (z != CONTROL_POINT)
                if ( (argc==4) && (strncmp(argv[3],"G",1) == 0) )
                    fprintf(fptr_out,"%f|%f|# %d\n",
                                        x+x_orig,y+y_orig,(int)(z*1000.0));
                else
                    fprintf(fptr_out,"%f %f %f\n",x+x_orig,y+y_orig,z);
        }

        else if (strncmp(text,"ZS",2) == 0)
                        /* <---- Coords stored in a a Z string (ZS instead */
        {
            fgets(text,sizeof(text),fptr_in);
                        /* <---- Read x,y and z values */
            sscanf(text,"%f %f %f",&x,&y,&z);

            /* Send to output file */

            if ( (argc==4) && (strncmp(argv[3],"G",1) == 0) )
                fprintf(fptr_out,
                        "%f|%f|# %d\n",x+x_orig,y+y_orig,(int)(z*1000.0));
            else
                fprintf(fptr_out,"%f %f %f\n",x+x_orig,y+y_orig,z);
        }

    }


    /* ------ Close Both Files ------- */

    fclose(fptr_in);
    fclose(fptr_out);
}
```

m.in.dti

GRASS module that converts a LaserScan DTI DEM into a GRASS raster.

Jo Wood, Department of Geography, 3rd August 1993

dti.h

```
/*****************************************************************************/
/***                                                                       ***/
/***                    dti.h for user with m.in.dti                       ***/
/***                   V1.0  - Jo Wood, 3rd August 1993                    ***/
/***                                                                       ***/
/*****************************************************************************/

#include "gis.h"                /* This MUST be included in all GRASS   */
                                /* programs. It sets up the necessary   */
                                /* prototypes for GRASS library calls.  */


/* ------ Some shorthand defines for region information. ------ */

#define North (region.north)    /* When outputting any vector file it   */
#define South (region.south)    /* is advisable to fill in the vector   */
#define East (region.east)      /* header file with its geographical    */
#define West (region.west)      /* extent. This is stored in the region */
#define NS_Res (region.ns_res)  /* structure. For convenience, these    */
#define EW_Res (region.ew_res)  /* are referred to by their shorthand   */
#define Zone (region.zone)      /* define names.                        */


/* ------ Some DTI Parameters ------*/

#define HEAD_OFFSET 1632

/* ------ Global variables ------ */

#ifndef MAIN
    extern                      /* Externally defined if not main()     */
#endif

char            *dti_in_name,  /* Name of the raster file to convert.  */

                *rast_out_name, /* Name of the output sites file.      */

                *mapset_out;    /* Names of mapsets containing files.   */

#ifndef MAIN
    extern
#endif
```

```
int             fd_out;         /* File descriptor for raster file      */


#ifndef MAIN
    extern
#endif

FILE            *dti_fptr,    /* File descriptor for output sites file */
                *cell_fptr;
```

main.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                              m.in.dti                                 ***/
/***  GRASS module to convert Laser Scan DTI files into GRASS rasters.     ***/
/***                     Jo Wood, Dept of Geography                        ***/
/***                       V 1.0 3rd August 1993                           ***/
/***                                                                       ***/
/*****************************************************************************/

#define MAIN

#include "dti.h"

main(argc,argv)
    int argc;
    char *argv[];       /* Assuming GRASS is going to respond to some   */
                        /* input from the keyboard, the two arguments   */
                        /* are necessary. argc is an ARGument Count     */
                        /* that stores the number of words input. *argv */
                        /* is a POINTER to an array holding those words.*/
{

    /*---------------------------------------------------------------------*/
    /*                            INITIALISE                               */
    /*---------------------------------------------------------------------*/


    /*---------------------------------------------------------------------*/
    /*                        GET INPUT FROM USER                          */
    /*---------------------------------------------------------------------*/

    interface(argc,argv);


    /*---------------------------------------------------------------------*/
    /*                       CONVERT RASTER TO SITES                       */
    /*---------------------------------------------------------------------*/

    convert();
    calc_histo();
    set_colour();
```

```
    /*--------------------------------------------------------------------*/
    /*                          CLOSE DOWN FILES                          */
    /*--------------------------------------------------------------------*/

    fclose(dti_fptr);
}
```

interface.c

```
/*****************************************************************************/
/***      Function to get input from user and check files can be opened    ***/
/***                                                                       ***/
/***                    Jo Wood, V1.0 3rd August 1993                      ***/
/*****************************************************************************/

#include "dti.h"

interface(argc,argv)

    int      argc;              /* Number of command line arguments     */
    char    *argv[];            /* Contents of command line arguments.  */

{
    /*--------------------------------------------------------------------*/
    /*                             INITIALISE                             */
    /*--------------------------------------------------------------------*/


    struct Option       *dti_in,        /* Pointer to  structures holding  */
                        *rast_out;      /* the text to describe each option*/
                                        /* It also stores the user's input */

    G_gisinit (argv[0]);                /* GRASS library function called   */
                                        /* checks for valid command        */
                                        /* argv[0] is the program name.    */



    /*--------------------------------------------------------------------*/
    /*                          SET PARSER OPTIONS                        */
    /*--------------------------------------------------------------------*/

    dti_in   = G_define_option();       /* Pointer to memory for each option*/
    rast_out = G_define_option();

    /* Each option needs a 'key' (short description),
         a 'description` (a longer one),
         a 'type' (eg intiger, or string),
         and an indication whether manditory or not */

    dti_in->key          = "dti";
    dti_in->description  = "Laser Scan DTI file to convert";
    dti_in->type         = TYPE_STRING;
```

```
    dti_in->required     = YES;

    rast_out->key         = "out";
    rast_out->description = "Output raster file to produce";
    rast_out->type        = TYPE_STRING;
    rast_out->required    = YES;

    if (G_parser(argc,argv))            /* Performs the prompting for   */
        exit(-1);                       /* keyboard input.              */
                                        /* Returns 0 if sucessful.      */

    dti_in_name    = dti_in->answer;
    rast_out_name  = rast_out->answer;


    /*--------------------------------------------------------------------*/
    /*                     CHECK INPUT DTI FILE EXISTS                    */
    /*--------------------------------------------------------------------*/

    if ((dti_fptr=fopen(dti_in_name,"r")) == NULL)
    {
        char err[256];
        sprintf(err,"DTI file [%s] not available.",dti_in_name);
        G_fatal_error(err);
    }

    /*--------------------------------------------------------------------*/
    /*             CHECK OUTPUT RASTER DOES NOT EXIST AND OPEN             */
    /*--------------------------------------------------------------------*/

    mapset_out = G_mapset();            /* Set output to current mapset.  */

    if (G_legal_filename(rast_out_name)==NULL)
    {
        char err[256];
        sprintf(err,"Illegal raster file name. Please try another.");
        G_fatal_error(err);
    }
    else
    {
        if (G_find_cell2(rast_out_name,mapset_out) != NULL)
        {
            char err[256];
            sprintf(err,
                "Raster map layer [%s] already exists.\nPlease try another\n",
                                                    rast_out_name);
            G_fatal_error(err);
        }

    }

}
```

convert.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                              convert()                                ***/
/***            Reads in a DTI file and converts to a GRASS raster         ***/
/***                     Jo Wood V1.0 Department of Geography              ***/
/***                                                                       ***/
/*****************************************************************************/

#include "dti.h"

convert()
{

    /*-------------------------------------------------------------------*/
    /*                              INITIALISE                           */
    /*-------------------------------------------------------------------*/

    CELL        *row_buf;       /* GRASS raster row                      */

    struct      Cell_head  region;

    int         row,col;        /* Counts through each row and column    */
                                /* of the input raster.                  */


    char        header[HEAD_OFFSET], /* Offset into DTI file after header*/
                dti_cell[2],        /* Size in bytes of a raster row.    */
                interleave[2],  /* Blank to interleave with raster cell */
                command[120];

    double      south_offset=0.0,
                west_offset=0.0;

    interleave[0] = 0;
    interleave[1] = 0;

    /*-------------------------------------------------------------------*/
    /*                          SET GRASS REGION INFO                    */
    /*-------------------------------------------------------------------*/

    G_get_window(&region);


    offset(dti_in_name,&west_offset,&south_offset);

    south_offset -= 25.0;       /* Realign raster to centre of dti cell */
    west_offset  -= 25.0;

    North  = south_offset + 20050.0;
    South  = south_offset;
    East   = west_offset + 20050.0;
    West   = west_offset;
    NS_Res = 50.0;
```

```
    EW_Res = 50.0;

    G_set_window(&region);


    if ((fd_out = G_open_cell_new_uncompressed(rast_out_name)) <0)
    {
        char err[256];
        sprintf(err,"ERROR: Problem opening output raster file.");
        G_fatal_error(err);
    }


    G_close_cell(fd_out);


    /*--------------------------------------------------------------------*/
    /*                          READ DTI HEADER                           */
    /*--------------------------------------------------------------------*/

    fread(header,1,sizeof(header),dti_fptr);

    /*--------------------------------------------------------------------*/
    /*                        READ DTI RASTER DATA                        */
    /*--------------------------------------------------------------------*/

    G_remove("cell",rast_out_name);     /* Remove empty raster.          */

    cell_fptr=G_fopen_new("cell","TeMpFiLe");

    for (row = 0; row<401; row++)
        for (col=0; col<401; col++)
        {
            fread(dti_cell,1,sizeof(dti_cell),dti_fptr);

            fwrite(interleave,1,sizeof(interleave),cell_fptr);
            fwrite(&dti_cell[1],1,1,cell_fptr);      /* Swap Byte order */
            fwrite(&dti_cell[0],1,1,cell_fptr);
        }

    fclose(cell_fptr);

    /*--------------------------------------------------------------------*/
    /*                     ROTATE AND COMPRESS DATA,                      */
    /*--------------------------------------------------------------------*/

    sprintf(command,"m.rot90 input=$LOCATION/cell/TeMpFiLe
        output=$LOCATION/cell/%s bpc=4 rows=401 cols=401",rast_out_name);

    system(command);
    G_remove("cell","TeMpFiLe");

    sprintf(command,"r.compress %s",rast_out_name);
    system(command);
}
```

```
/*****************************************************************/
/* offset() -   Calculates map sheet offset                      */
/*              Converts full National Grid reference into       */
/*              numeric grid coordinates.                        */
/*****************************************************************/

offset(map_name,easting,northing)
    char        *map_name;      /* Map Name: eg NF02, tq64.dti etc.     */
    double      *easting,*northing;
{
    switch(*map_name)
    {
        case 'H': case 'h':             /* Initial Easting Offset       */
        case 'N': case 'n':
        case 'S': case 's':
            *easting=0.0;
            break;

        case 'J': case 'j':
        case 'O': case 'o':
        case 'T': case 't':
            *easting=500000.0;
            break;

        default:
            fprintf(stderr,"WARNING: Map Name not identified.\n");
            break;
    }


    switch(*map_name)
    {                                           /* Initial Northing Offset */
        case 'S': case 's':
        case 'T': case 't':
            *northing = 500000.0;
            break;

        case 'N': case 'n':
        case 'O': case 'o':
            *northing = 1000000.0;
            break;

        case 'H': case 'h':
        case 'J': case 'j':
            *northing = 1500000.0;
            break;

        default:
            fprintf(stderr,"WARNING: Map Name not identified.\n");
            break;
    }

    switch(*(map_name+1))                       /* Second Easting Offset */
    {
```

```
        case 'A': case 'a':
        case 'F': case 'f':
        case 'L': case 'l':
        case 'Q': case 'q':
        case 'V': case 'v':
            *easting += 0.0;
            break;

        case 'B': case 'b':
        case 'G': case 'g':
        case 'M': case 'm':
        case 'R': case 'r':
        case 'W': case 'w':
            *easting += 100000.0;
            break;

        case 'C': case 'c':
        case 'H': case 'h':
        case 'N': case 'n':
        case 'S': case 's':
        case 'X': case 'x':
            *easting += 200000.0;
            break;

        case 'D': case 'd':
        case 'J': case 'j':
        case 'O': case 'o':
        case 'T': case 't':
        case 'Y': case 'y':
            *easting += 300000.0;
            break;

        case 'E': case 'e':
        case 'K': case 'k':
        case 'P': case 'p':
        case 'U': case 'u':
        case 'Z': case 'z':
            *easting += 400000.0;
            break;

        default:
            fprintf(stderr,"WARNING: Map Name not identified.\n");
            break;
    }


    switch(*(map_name+1))                    /* Second Northing Offset */
    {
        case 'A': case 'a':
        case 'B': case 'b':
        case 'C': case 'c':
        case 'D': case 'd':
        case 'E': case 'e':
            *northing -= 100000.0;
            break;
```

```
        case 'F': case 'f':
        case 'G': case 'g':
        case 'H': case 'h':
        case 'J': case 'j':
        case 'K': case 'k':
            *northing -= 200000.0;
            break;

        case 'L': case 'l':
        case 'M': case 'm':
        case 'N': case 'n':
        case 'O': case 'o':
        case 'P': case 'p':
            *northing -= 300000.0;
            break;

        case 'Q': case 'q':
        case 'R': case 'r':
        case 'S': case 's':
        case 'T': case 't':
        case 'U': case 'u':
            *northing -= 400000.0;
            break;

        case 'V': case 'v':
        case 'W': case 'w':
        case 'X': case 'x':
        case 'Y': case 'y':
        case 'Z': case 'z':
            *northing -= 500000.0;
            break;

        default:
            fprintf(stderr,"WARNING: Map Name not identified.\n");
            break;
    }
                                /* Offset within each 100km Grid Square */
    *easting  += (dti_in_name[2] - '0')*10000.0;
    *northing += (dti_in_name[3] - '0')*10000.0;
}
```

calc_histo.c

```
/*****************************************************************/
/*                          calc_histo()                         */
/* Updates the histogram information of newly created raster.    */
/* Modified from GRASS function do_histogram() as part of r.support.    */
/*                                                               */
/*****************************************************************/

#include "dti.h"

calc_histo ()
{
```

```
    CELL                *cell;

    struct Cell_head    cellhd;
    struct Range        range;
    struct Histogram    histogram;


    struct Cell_stats   statf;
    int                 nrows, ncols,
                        row;

    int                 fd,i;

    if (G_get_cellhd (rast_out_name, mapset_out, &cellhd) < 0)
        return 0;
    G_set_window (&cellhd);
    fd = G_open_cell_old (rast_out_name, mapset_out);
    if (fd < 0)
        return 0;
    nrows = G_window_rows();
    ncols = G_window_cols();
    cell = G_allocate_cell_buf();

    G_init_cell_stats (&statf);
    for (row = 0; row < nrows; row++)
    {
        if (G_get_map_row_nomask (fd, cell, row) < 0)
            break;
        G_update_cell_stats (cell, ncols, &statf);
    }
    G_close_cell (fd);
    free (cell);

    if (row == nrows)
        G_write_histogram_cs (rast_out_name, &statf);
    G_free_cell_stats (&statf);


    G_init_range (&range);

    if (G_read_histogram (rast_out_name, mapset_out, &histogram) <= 0)
        return 1;

    i = G_get_histogram_num (&histogram);
    while (i >= 0)
        G_update_range (G_get_histogram_cat(i--,&histogram), &range);
    G_write_range (rast_out_name, &range);
}
```

set_colour.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                            set_colour()                               ***/
/***                  Sets colour table associated with DEM                ***/
/***                     Jo Wood V1.0, Dept. of Geography                  ***/
/***                                                                       ***/
/*****************************************************************************/

#include "dti.h"


#define SEA       0, 0, 140
#define COAST1  180,175,  0
#define COAST2  180,185,  0
#define LOWLAND1  0,140,  0
#define LOWLAND2  0,180, 10
#define LOWLAND3  0,240, 50
#define UPLAND1 100, 80, 40
#define UPLAND2 100, 80,120
#define PEAK    255,255,255


set_colour()
{

    /*-------------------------------------------------------------------*/
    /*                            INITIALISE                             */
    /*-------------------------------------------------------------------*/

    struct Colors       colours;        /* Structure to store colour table*/


    G_init_colors(&colours);            /* Initialise structure.          */

    /*-------------------------------------------------------------------*/
    /*                      CREATE AND WRITE COLOUR TABLE                */
    /*-------------------------------------------------------------------*/

    G_add_color_rule((CELL)0,SEA,       (CELL)1,COAST1,         &colours);
    G_add_color_rule((CELL)1,COAST1,    (CELL)5,COAST2,         &colours);
    G_add_color_rule((CELL)5,COAST2,    (CELL)120,LOWLAND1,     &colours);
    G_add_color_rule((CELL)120,LOWLAND1,(CELL)190,LOWLAND2,     &colours);
    G_add_color_rule((CELL)190,LOWLAND2,(CELL)250,LOWLAND3,     &colours);
    G_add_color_rule((CELL)250,LOWLAND3,(CELL)450,UPLAND1,      &colours);
    G_add_color_rule((CELL)450,UPLAND1, (CELL)550,UPLAND2,      &colours);
    G_add_color_rule((CELL)550,UPLAND2, (CELL)950,PEAK,         &colours);
    G_add_color_rule((CELL)950,PEAK,    (CELL)1500,PEAK,        &colours);


    G_write_colors(rast_out_name,mapset_out,&colours);
    G_free_colors(&colours);
}
```

m.in.ntf

GRASS module that converts Ordnace Survey NTF files into GRASS format.

Each line in an NTF file starts with a 2 digit record type. Included here is a list of supported and unsupported record types.

The following NTF record types are supported

00 Continuation record
01 Volume header record
02 Database header record
03 Data description record
04 Data format record
05 Feature classification record
07 Section header record
11 Name record
12 Name position record
14 Attribute record
15 Point record
16 Node record
21 Geometry record
22 3D Geometry record
23 Line record
40 Attribute description record
43 Text record
44 Text position record
45 Text representation record
50 Grid header record
51 Grid data record
99 Volume termination record

The following are not yet supported

24 Chain record
31 Polygon record
33 Complex polygon
34 Collection of features

Coding history:

V1.0 Jo Wood, Department of Geography, August 1992
V2.0 Jo Wood, Department of Geography, 29th May 1993
Reads both NTF 1.0 and NTF 2.0.
V2.1 Modified to read in large scale vector data. Minor bug fixes.
V2.2 Modified, 10th February, 1995. Minor bug fixes.

V2.3 Modified 18th October, 1995. Incorporates 'STRATEGI' vector files.
Minor bug modifications, 19th October, 1995.
23rd October, 1995.

Last modified 24th October - Fixed attribute labelling problem.

ntf.h

```
/*****************************************************************/
/***                          ntf.h                            ***/
/*** Header for programs examining Ordnance Survey NTF files  ***/
/*** To conform with BS 7567 (NTF V2.0)                       ***/
/*** Jo Wood, Department of Geograph, 26th May 1993           ***/
/*****************************************************************/

/*---------------------------------------------------------------------*/
/*                    HEADER STRUCTURES - 01 02 03 04 05.              */
/*---------------------------------------------------------------------*/

struct VOLHDREC
{
    char        REC_DESC_1;     /* 01 for Volume header record.              */
    char        DONOR[21];      /* Will usually be ORDNANCE SURVEY.          */
    char        RECIPIENT[21];  /* Customer name.                            */
    char        TRANDATE[9];    /* Date of processing.                       */
    int         SERIAL;         /* Not used.                                 */
    char        VOLNUM;         /* Volume number incremented for each vol.   */
    char        NTFLEVEL;       /* NTF level.                                */
    float       NTFVER;         /* NTF version (2.00)                        */
    char        NTFOR;          /* 'V' for variable length record.           */
    char        EOR_1;          /* '%' on mag tape ' ' if not terminated here.  */
    char        DIVIDER;        /* '\' used to terminate variable len txt fields*/
    char        CONT_MARK_1;    /* '0' No further records.                   */
    char        EOR_2;          /* '%' End of record.                        */
};

struct DBHDREC
{
    char        REC_DESC_1;     /* 02 for Database header record.            */
    char        DBNAME[21];     /* Indicates product name.                   */
    char        DDNAME[21];     /* Standard NTF data dictionary name.        */
    char        DDATE[9];       /* Date of standard data dictionary.         */
    char        DDBASE[21];     /* Not used.                                 */
```

```
    char        DDBDATE[9];     /* Not used.                                    */
    char        CONT_MARK_1;    /* '0' No further records '1' continuation.     */
    char        EOR_1;          /* '%' End of record.                           */
    char        REC_DESC_2;     /* 00 for continuation of Database header rec.  */
    char        FCNAME[21];     /* Not used.                                    */
    char        FCDATE[9];      /* Not used.                                    */
    char        DQNAME[21];     /* Not used.                                    */
    char        DQDATE[9];      /* Not used.                                    */
    char        DATA_MODEL;     /* Not used.                                    */
    char        CONT_MARK_2;    /* '0' No further records.                      */
    char        EOR_2;          /* '% End of record.                            */
};

struct DATADESC
{
    char        REC_DESC_1;     /* 03 for Data description record.              */
    char        FIELD_NAME[11]; /* Field name, eg. GRID_ID.                     */
    int         FWIDTH;         /* Field width, eg. 010.                        */
    char        FINTER[6];      /* Interpretation of field, eg. I10<2S          */
    char        FDESC[39];      /* Field description (variable length).         */
                                /* INCOMPLETE: rest merged with FDESC.          */
    char        CONT_MARK_1;    /* '0' No further records.                      */
    char        EOR_1;          /* '%' End of record.                           */
};

struct DATAFMT
{
    char        REC_DESC_1;     /* 04 for Data format record.                   */
    char        REC_TYPE;       /* Either 50 or 51 describing record defined.   */
    char        REC_NAME[11];   /* Record name.                                 */
    char        NUM_FIELD;      /* Number of fields in the record.              */
    char        FIELD_NAME[11]; /* Field name (repeating field).                */
    char        FUSE;           /* Field use (repeating field).                 */
    char        CONT_MARK_1;    /* '0' No further records '1' continuation.     */
    char        EOR_1;          /* '%' End of record.                           */
    char        REC_DESC_2;     /* 00 for continuation of Database header rec.  */
    char        COMMENT[77];    /* If continued, used for description.          */
                                /* INCOMPLETE: merged in single comment field.  */
    char        CONT_MARK_2;    /* '0' No further records.                      */
    char        EOR_2;          /* '% End of record.                            */
};

struct FEATCLASS
{
    char        REC_DESC_1;     /* 05 for feature classification record.        */
    int         FEAT_CODE;      /* Contains 4 digit feature code.               */
    char        CODE_COM[11];   /* Not used.                                    */
    char        STCLASS[21];    /* Not used.                                    */
    char        FEATDES[43];    /* Feature description.                         */
    char        CONT_MARK_1;    /* '0' No further records.                      */
    char        EOR_1;          /* '% End of record.                            */
};


/*------------------------------------------------------------------------------*/
```

```
/*                          SECTION HEADER STRUCTURES - 07.                    */
/*-------------------------------------------------------------------------*/

struct SECHREC
{
    char        REC_DESC_1;     /* 07 for Section Header Record.              */
    char        SECT_REF[11];   /* Section Reference, eg. SK22.               */
    char        COORD_TYPE;     /* '2' - for rectangular maps.                */
    char        STRUC_TYPE;     /* '1' - for vectors.                         */
    int         XY_LEN;         /* Length of (x,y) coordinates (0 for vectors). */
    char        XY_UNIT;        /* '2' - for metres.                          */
    float       XY_MULT;        /* (x,y) scaling factor (1.0 is default).     */
    int         Z_LEN;          /* Length of (z) coordinates (0 for vectors). */
    char        Z_UNIT;         /* '2' - for metres.                          */
    float       Z_MULT;         /* (z) scaling factor (1.0 is default).       */
    int         X_ORIG;         /* (x) coordinate of SW corner.               */
    int         Y_ORIG;         /* (y) coordinate of SW corner.               */
    int         Z_DATUM;        /* Vertical datum (usually 0)                 */
    char        CONT_MARK_1;    /* '0' No further records, '1' continuation.  */
    char        EOR_1;          /* '%' End of record.                         */
    char        REC_DESC_2;     /* 00 for continuation.                       */
    int         XMIN;           /* Minimum (x) value.                         */
    int         YMIN;           /* Minimum (y) value.                         */
    int         XMAX;           /* Maximum (x) value.                         */
    int         YMAX;           /* Maximum (y) value.                         */
    float       XY_ACC;         /* (x,y) accuracy - '0.0' Not used !          */
    float       Z_ACC;          /* (z) accuracy - eg '5.0'.                   */
    char        SURV_DATE[9];   /* Nominal date of survey.                    */
    char        LAST_AMND[9];   /* Date of last amendment.                    */
    char        COPYRIGHT[9];   /* Copyright date.                            */
    char        CONT_MARK_2;    /* '0' No further records or '1' Continuation */
    char        EOR_2;          /* '%' End of record.                         */
    char        REC_DESC_3;     /* 00 for continuation.                       */
    char        SQNAME[21];     /* Not used.                                  */
    char        SQDATE[9];      /* Not used.                                  */
    int         SCALE;          /* 1:1250, 1:2500, or 1:10000 scale data.     */
    float       GRID_OR_X;      /* Not used.                                  */
    float       GRID_OR_Y;      /* Not used.                                  */
    float       PROJ_OR_LAT;    /* Not used.                                  */
    float       PROJ_OR_LNG;    /* Not used.                                  */
    char        CONT_MARK_3;    /* '0' No further records or '1' Continuation */
    char        EOR_3;          /* '%' End of record.                         */
    char        REC_DESC_4;     /* 00 for continuation.                       */
    char        SPHER_NAME[10]; /* Not used.                                  */
    float       MAJOR_AXIS;     /* Not used.                                  */
    float       ECCENTRICITY;   /* Not used.                                  */
    float       FLATTENING;     /* Not used.                                  */
    char        PROJECTION[10]; /* Not used.                                  */
    int         PARAMETER_1;    /* Not used.                                  */
    char        P_TYPE;         /* Not used.                                  */
    int         PARAMETER_2;    /* Not used.                                  */
    char        DBANK_DATE[9];  /* Initial data banking date.                 */
    char        CONT_IND[9];    /* Defines map content/accuracy.              */
    char        CONT_MARK_4;    /* '0' No further records or '1' Continuation */
    char        EOR_4;          /* '%' End of record.                         */
```

```
    char        REC_DESC_5;     /* 00 for continuation.                          */
    char        BOUND_DATE[9];  /*                                               */
    char        DIG_U_DATE[9];  /*                                               */
    char        SPEC_REF[15];   /* Digitizing specification at time od digitn.   */
    char        EMATCH_N[2];    /* Northern Edge Matching ( ' ' if not matched)  */
    char        EMATCH_N_DATE[9];/* Date of edge match.                          */
    char        EMATCH_E[2];    /* Eastern Edge Matching ( ' ' if not matched)   */
    char        EMATCH_E_DATE[9];/* Date of edge match.                          */
    char        EMATCH_S[2];    /* Southern Edge Matching ( ' ' if not matched)  */
    char        EMATCH_S_DATE[9];/* Date of edge match.                          */
    char        EMATCH_W[2];    /* Western Edge Matching ( ' ' if not matched)   */
    char        EMATCH_W_DATE[9];/* Date of edge match.                          */
    int         HO_UNIT_COUNT;  /* Cumulative count of the change to the map.    */
    char        CONT_MARK_5;    /* '0' No further records or '1' Continuation    */
    char        EOR_5;          /* '%' End of record.                            */
                                /* NOTE: POSSIBLE FREE TEXT IN CONTINUATION RECS*/

};


/*-------------------------------------------------------------------------*/
/*                    TEXT INFORMATION - 11,12,43,44 and 45                */
/*-------------------------------------------------------------------------*/

struct NAMEREC
{
    char        REC_DESC_1;     /* 11 for Name Record.                           */
    int         NAME_ID;        /* Feature serial number.                        */
    char        TEXT_CODE[5];   /* Feature classification number.                */
    char        TEXT_LEN;       /* Number of characters in text string.          */
    char        TEXT[100];      /* The text string.                              */
    char        SECURITY;       /* Not used.                                     */
    char        CHG_TYPE;       /* Not used.                                     */
    char        CHG_DATE[9];    /* Not used.                                     */
    char        QLABEL;         /* Not used.                                     */
    char        SVY_DATE[9];    /* Not used.                                     */
    char        CONT_MARK_1;    /* '0' No further records or '1' Continuation    */
    char        EOR_1;          /* '%' End of record.                            */
};

struct NAMPOSTN
{
    char        REC_DESC_1;     /* 12 For Name Position Record.                  */
    int         FONT;           /* Font used for text string.                    */
    float       TEXT_HT;        /* Text height in mm.                            */
    char        DIG_POSTN;      /* Text location code (0-8).                     */
    float       ORIENT;         /* Text orientation (0.0 - 359.9 degrees).       */
    char        CONT_MARK_1;    /* '0' No further records.                       */
    char        EOR_1;          /* '%' End of record.                            */
};                              /* NOTE: SHOULD BE FOLLOWED BY GEOMETRY RECORD. */

struct TEXTREC
{
    char        REC_DESC_1;     /* 43 for Text Record.                           */
    int         TEXT_ID;        /* Sequential number of text record.             */
```

```
    char        NUM_SEL;        /* Number selected (01)                          */
    char        SELECT;         /* Selection (00)                                */
    int         TEXT_CODE;      /* (0000)                                        */
    int         TEXP_ID;        /* Sequential number of text position record.    */
    char        NUM_ATT;        /* Number of attributes selected (01)            */
    int         ATT_ID;         /* Sequential number of attribute record.        */
    char        CONT_MARK_1;    /* '0' No further records or '1' Continuation    */
    char        EOR_1;          /* '%' End of record.                            */
};

struct TEXTPOS
{
    char        REC_DESC_1;     /* 44 For Text Position Record.                  */
    int         TEXP_ID;        /* Sequential number of text position record.    */
    char        NUM_TEXR;       /* Number of text records selected (01)          */
    int         TEXR_ID;        /* Sequential num of text representation record.*/
    int         GEOM_ID;        /* Geometry record holding name position.        */
    char        CONT_MARK_1;    /* '0' No further records.                       */
    char        EOR_1;          /* '%' End of record.                            */
};

struct TEXTREP
{
    char        REC_DESC_1;     /* 45 For Text Representation Record.            */
    int         TEXR_ID;        /* Sequential num of text representation record.*/
    int         FONT;           /* Font used for text string.                    */
    float       TEXT_HT;        /* Text height in mm.                            */
    char        DIG_POSTN;      /* Text location code (0-8).                     */
    float       ORIENT;         /* Text orientation (0.0 - 359.9 degrees).       */
    char        CONT_MARK_1;    /* '0' No further records.                       */
    char        EOR_1;          /* '%' End of record.                            */
};


/*-------------------------------------------------------------------------*/
/*                      ATTRIBUTE DATA STRUCTURES - 14, 40.                */
/*-------------------------------------------------------------------------*/

struct ATTREC
{
    char        REC_DESC_1;     /* 14 for Attribute Record.                      */
    int         ATT_ID;         /* Attribute identifier.                         */
    char        VAL_TYPE[3];    /* Code for attribute mnemonic.                  */
    char        VALUE[70];      /* Attribute value (could be int or char).       */
    char        CONT_MARK_1;    /* '0' No further records.                       */
    char        EOR_1;          /* '%' End of record.                            */
};


struct ATTDESC
{
    char        REC_DESC_1;     /* 40 for Attribute Description record.          */
    char        VAL_TYPE[3];    /* Code for description type.                    */
    char        FWIDTH[4];      /* Field width of attribute name.                */
    char        FINTER[6];      /* Format of attribute name.                     */
```

```
    char        ATT_NAME[65];   /* Attribute Name.                              */
    char        DIVIDER_1;      /* Divides attribute name from description.     */
    char        FDESC[63];      /* Attribure description (optional).            */
    char        DIVIDER_2;      /* End of attribute description.                */
    char        CONT_MARK_1;    /* '0' No further records.                      */
    char        EOR_1;          /* '%' End of record.                           */
};



/*-------------------------------------------------------------------------*/
/*                VECTOR STRUCTURES - 15 16 21 22 23.                      */
/*-------------------------------------------------------------------------*/

struct POINTREC
{
    char        REC_DESC_1;     /* 15 for Point Record.                         */
    int         POINT_ID;       /* Point Identity.                              */
    int         GEOM_ID;        /* Geometry Identity.                           */
    char        NUM_ATT;        /* Number of attributes '01'.                   */
    int         ATT_ID;         /* Attribute identity.                          */
    char        CONT_MARK_1;    /* '0' No further records                       */
    char        EOR_1;          /* '%' End of record.                           */
};

struct POINTREC_C
{
    char        REC_DESC_1;     /* 15 for Point Record (Contour variation).     */
    int         POINT_ID;       /* Point Identity.                              */
    char        VAL_TYPE[3];    /* Will be HT for height values.                */
    int         VALUE;          /* Height value.                                */
    int         FEAT_CODE;      /* Feature code (200 for spot heights).         */
    char        CONT_MARK_1;    /* '0' No further records                       */
    char        EOR_1;          /* '%' End of record.                           */
};

struct POINTREC_L
{
    char        REC_DESC_1;     /* 15 for Point Record (Land-Line variation).   */
    int         POINT_ID;       /* Point Identity.                              */
    char        VAL_TYPE[3];    /* Will be DT for distance mnemonic.            */
    float       VALUE;          /* Orientation value (0.0 - 359.9).             */
    int         FEAT_CODE;      /* Feature code (1-400 for Land-Line).          */
    char        SECURITY;       /* Not used.                                    */
    char        CHG_TYPE;       /* Not used.                                    */
    char        CHG_DATE[7];    /* Not used.                                    */
    char        QLABEL;         /* Not used.                                    */
    char        SVY_DATE[7];    /* Not used.                                    */
    char        CONT_MARK_1;    /* '0' No further records                       */
    char        EOR_1;          /* '%' End of record.                           */
};


struct NODEREC
{
```

```
    char        REC_DESC_1;     /* 16 for Node Record.                          */
    int         NODE_ID;        /* Node Identity (same as for Point [15] ).     */
    int         GEOM_ID;        /* Geometry Identity (same as for Geom [21] ).  */
    int         NUM_LINKS;      /* Number of links at node (always <10).        */
                                /* INCOMPLETE                                   */
    char        CONT_MARK_1;    /* '0' No further records                       */
    char        EOR_1;          /* '%' End of record.                           */
};

struct GEOMREC
{
    char        REC_DESC_1;     /* 21 for two dimensional geometry record.      */
    int         GEOM_ID;        /* Geometry identity.                           */
    char        G_TYPE;         /* '1' for point features, '2' for lines.       */
    int         NUM_COORD;      /* Number of following coordinate pairs.        */
    int         X_COORD;        /* X coordinate or easting.                     */
    int         Y_COORD;        /* Y coordinate or northing.                    */
    char        Q_PLAN;         /* Not used.                                    */
    char        CONT_MARK_1;    /* '1' further record, '0' No firther records.  */
    char        EOR_1;          /* '%' End of record.                           */
};

struct GEOMREC2
{
    char        REC_DESC_1;     /* 22 for three dimensional geometry record.    */
    int         GEOM_ID;        /* Geometry identity.                           */
    char        G_TYPE;         /* '1' for point features, '2' for lines.       */
    int         NUM_COORD;      /* Number of following coordinate pairs.        */
    int         X_COORD;        /* X coordinate or easting.                     */
    int         Y_COORD;        /* Y coordinate or northing.                    */
    char        Q_PLAN;         /* Not used.                                    */
    int         Z_COORD;        /* Z coordinate or elevation.                   */
    char        QHT;            /* Not used.                                    */
    char        CONT_MARK_1;    /* '1' further record, '0' No firther records.  */
    char        EOR_1;          /* '%' End of record.                           */
};


struct LINEREC
{
    char        REC_DESC_1;     /* 23 for Line Record.                          */
    int         LINE_ID;        /* Line Identity.                               */
    int         GEOM_ID;        /* Geometry Identity.                           */
    char        NUM_ATT;        /* Number of attributes '01'.                   */
    int         ATT_ID;         /* Attribute identity.                          */
    char        CONT_MARK_1;    /* '0' No further records                       */
    char        EOR_1;          /* '%' End of record.                           */
};

struct LINEREC_C
{
    char        REC_DESC_1;     /* 23 for Line Record (Contour variation).      */
    int         LINE_ID;        /* Line Identity.                               */
    char        VAL_TYPE[3];    /* Will be HT for height values.                */
    int         VALUE;          /* Height value.                                */
```

```
    int         FEAT_CODE;      /* Feature code (201 for contours).          */
    char        CONT_MARK_1;    /* '0' No further records                    */
    char        EOR_1;          /* '%' End of record.                        */
};

struct LINEREC_L
{
    char        REC_DESC_1;     /* 23 for Line Record (Land-Line variation). */
    int         LINE_ID;        /* Line Identity.                            */
    char        VAL_TYPE[3];    /* Will be DT for distance mnemonic.         */
    int         VALUE;          /* Not used.                                 */
    int         FEAT_CODE;      /* Feature code (1-400 for Land-Line).       */
    char        SECURITY;       /* Not used.                                 */
    char        CHG_TYPE;       /* Not used.                                 */
    char        CHG_DATE[7];    /* Not used.                                 */
    char        QLABEL;         /* Not used.                                 */
    char        SVY_DATE[7];    /* Not used.                                 */
    char        CONT_MARK_1;    /* '0' No further records                    */
    char        EOR_1;          /* '%' End of record.                        */
};


/*---------------------------------------------------------------------------*/
/*                      DTM GRID STRUCTURES - 50 51.                         */
/*---------------------------------------------------------------------------*/

struct GRIDHREC
{
    char        REC_DESC_1;     /* 50 for Grid Header Record.                */
    int         GRID_ID;        /* 1km Grid Reference of map sheet square.   */
    int         N_COLUMNS;      /* Number of columns in Grid. (should be 401). */
    int         N_ROWS;         /* Number of rows in Grid. (should be 401).  */
    int         N_PLANES;       /* Number of planes in Grid. (should be 1).  */
    int         X_COORD_1;      /* Grid corner values.                       */
    int         Y_COORD_1;      /* Grid corner values.                       */
    int         Z_COORD_1;      /* Grid corner values.                       */
    int         X_COORD_2;      /* Grid corner values.                       */
    int         Y_COORD_2;      /* Grid corner values.                       */
    int         Z_COORD_2;      /* Grid corner values.                       */
    char        CONT_MARK_1;    /* First continuation mark.                  */
    char        EOR_1;          /* First record termination mark.            */
    char        REC_DESC_2;     /* Continuation descriptor.                  */
    int         X_COORD_3;      /* Grid corner values.                       */
    int         Y_COORD_3;      /* Grid corner values.                       */
    int         Z_COORD_3;      /* Grid corner values.                       */
    int         X_COORD_4;      /* Grid corner values.                       */
    int         Y_COORD_4;      /* Grid corner values.                       */
    int         Z_COORD_4;      /* Grid corner values.                       */
    char        CONT_MARK_2;    /* Second continuation mark.                 */
    char        EOR_2;          /* Second record termination mark.           */
    char        REC_DESC_3;     /* Continuation descriptor.                  */
    int         X_COORD_5;      /* Grid corner values.                       */
    int         Y_COORD_5;      /* Grid corner values.                       */
    int         Z_COORD_5;      /* Grid corner values.                       */
    int         X_COORD_6;      /* Grid corner values.                       */
```

```
    int         Y_COORD_6;      /* Grid corner values.                      */
    int         Z_COORD_6;      /* Grid corner values.                      */
    char        CONT_MARK_3;    /* Third continuation mark.                 */
    char        EOR_3;          /* Third record termination mark.           */
    char        REC_DESC_4;     /* Continuation descriptor.                 */
    int         X_COORD_7;      /* Grid corner values.                      */
    int         Y_COORD_7;      /* Grid corner values.                      */
    int         Z_COORD_7;      /* Grid corner values.                      */
    int         X_COORD_8;      /* Grid corner values.                      */
    int         Y_COORD_8;      /* Grid corner values.                      */
    int         Z_COORD_8;      /* Grid corner values.                      */
    char        CONT_MARK_4;    /* Fourth continuation mark.                */
    char        EOR_4;          /* Fourth record termination mark.          */
};


struct GRIDREC
{
    char        REC_DESC_1;     /* 51 for Grid Data Record.                 */
    int         GRID_ID;        /* Grid reference for mapsheet square.      */
    char        SURVEY[8];      /* Method and date of survey.               */
    char        CHANGE[8];      /* Type and date of change.                 */
    int         COL_START;      /* First column in record.                  */
    int         COL_END;        /* Last column in record.                   */
    int         ROW_START;      /* First row in record.                     */
    int         ROW_END;        /* Last row in record.                      */
    int         PLA_START;      /* First plane in record.                   */
    int         PLA_END;        /* Last plane in record.                    */
    char        COL_INV;        /* Column inversion ('0' = FALSE).          */
    char        ROW_INV;        /* Row inversion ('0'=FALSE).               */
    char        PLA_INV;        /* Plane inversion ('0' = FALSE).           */
    char        ORDER;          /* '1' = col,row,plane.                     */
    char        INTERPRET;      /* '1' = numerical interpretation of data.  */
    int         V_OFFSET;       /* Constant to be added to z values.        */
    float       V_SCALE;        /* Scaling factor for z values.             */
    char        CONT_MARK_1;    /* Continuation mark.                       */
    char        EOR_1;          /* Record termination mark.                 */
    char        REC_DESC_2;     /* Continuation descriptor.                 */
    int         N_GRIDVAL;      /* Number of grid values.                   */
    char        CONT_MARK_2;    /* Second continuation mark.                */
    char        EOR_2;          /* Second record termination mark.          */
    char        REC_DESC_3;     /* Continuation descriptor.                 */
    char        GRIDVAL_1[77];  /* 19 Grid values each of four digits.      */
    char        CONT_MARK_3;    /* Continuation mark.                       */
    char        EOR_3;          /* Record termination mark.                 */
    char        REC_DESC_4;     /* Final continuation descriptor.           */
    char        GRIDVAL_2[9];   /* 2 Grid values each of four digits.       */
    char        CONT_MARK_4;    /* Final continuation mark.                 */
    char        EOR_4;          /* Final record termination mark.           */
};


/*--------------------------------------------------------------------------*/
/*                          FOOTER STRUCTURES - 99.                         */
/*--------------------------------------------------------------------------*/
```

```
struct VOLTERM
{
    char        REC_DESC_1;     /* 99 for Volume Termination record.           */
    char        FREE_TEXT[77];  /* Either 'End of Transfer Set' or.            */
                                /* 'End of volume (n). Transfer continues on... */
    char        CONT_MARK_1;    /* '0' No further records.                     */
    char        EOR_1;          /* '% End of record.                           */
};
```

ntf_in.h

```
/*****************************************************************************/
/***                                                                       ***/
/***                            ntf_in.h                                   ***/
/***  Header file for use with m.in.ntf -                                  ***/
/***  Jo Wood, Dept of Geography, University of Leicester                  ***/
/***  V2.1  - 19th May, 1993                                               ***/
/***                                                                       ***/
/*****************************************************************************/

#include "gis.h"        /* This MUST be included in all GRASS programs  */
                        /* It sets up the necessary prototypes for all  */
                        /* GRASS library calls.                         */

#include "Vect.h"       /* Must be included in programs that manipulate */
                        /* vectors.                                     */

#include "dig_head.h"   /* Used for vector header information.          */

#include "ntf.h"

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

#define TRUE 1
#define FALSE 0
#define NO_ATTRIB -1

#define MODE_READ  0
#define MODE_WRITE 1
#define MODE_RW    2

#define TEXT_CODE 9999


/* Map Types */

#define CONTOUR 1
#define DEM 2
#define LANDLINE 3
#define BOUNDARYLINE 4
#define OSCAR 5
#define ONE250 6
```

```
#define ONE625 7
#define STRATEGI 8
#define TEXT_LABEL 9999
#define DONT_KNOW -1


/* ------ Global variables ------ */

#ifndef MAIN
    extern                      /* Externally defined if not main()     */
#endif

char            *ntf_in_name,   /* Name of the raster file to process.  */
                *file_out_name, /* Name of the raster output file.      */
                *mapset_out,    /* GRASS mapset holding output file.    */
                conversion_log, /* Determines if conversion log printed.*/
                outfile,        /* Determines if an outfile is created. */
                nodes,          /* Determines if nodes are transferred. */
                O_raster,       /* Determines if raster is already open */
                O_vector,       /* Determines if vector is already open */
                O_temp,
                O_spot,         /* Flags that determine whether files   */
                O_cont,         /*   are already open.                  */
                O_lake,         /*   Include:   Spot Heights, Contours, */
                O_break,        /*              Lakes, Breaklines,      */
                O_coast,        /*              Coastlines, Ridgelines, */
                                /*              and Form Lines.         */
                O_form,
                O_other,        /* Any other type of vector coverage.   */
                text[128],      /* Stores a line (record) of NTF file.  */
                H_organ[30],    /* Organisation supplying data.         */
                H_ddate[11],    /* Digitisation date                    */
                H_mname[40],    /* Title of map.                        */
                H_mdate[11],    /* Date of map survey.                  */
                V_name[40],     /* Vector object name.                  */
                Write_vect;     /* Flag to defermine if vector written  */

#ifndef MAIN
    extern                      /* Externally defined if not main()     */
#endif

float           XY_mult,        /* X, Y, and Z Scaling factors.         */
                Z_mult;


#ifndef MAIN
    extern                      /* Externally defined if not main()     */
#endif

FILE            *ntf_fptr,      /* File descriptor for input and        */
                *out_fptr,      /* output raster files.                 */
                *new_fptr,      /* New converted NTF 2.0 file.          */

                *fptr_spot,     /* File descriptor for vector files.    */
                *fptr_spot_att,
```

```
                *fptr_cont,
                *fptr_cont_att,
                *fptr_lake,
                *fptr_lake_att,
                *fptr_break,
                *fptr_break_att,
                *fptr_coast,
                *fptr_coast_att,
                *fptr_ridge,
                *fptr_ridge_att,
                *fptr_form,
                *fptr_form_att,
                *fptr_other,
                *fptr_other_att;


#ifndef MAIN
    extern                      /* Externally defined if not main()     */
#endif

int             line_number,    /* Current line of NTF file being read. */
                fd_out,         /* Raster output file descriptor.       */
                num_rlines,     /* Number of raster lines read.         */
                X_min,Y_min,    /* Boundaries of vector map.            */
                X_max,Y_max,
                X_origin,       /* SW corner of vector map.             */
                Y_origin,
                V_featcode,     /* Feature code of vector map.          */
                H_scale,        /* Original scale of map data.          */
                geom_type;      /* Point line or area type of vector    */



#ifndef MAIN
    extern                      /* Externally defined if not main()     */
#endif

struct Map_info  vect_info;     /* Structure to hold vector information */

#ifndef MAIN
    extern                      /* Externally defined if not main()     */
#endif

struct line_pnts *points;       /* Holds x,y coordinates of vector.     */

#ifndef MAIN
    extern                      /* Externally defined if not main()     */
#endif

struct Categories cats,         /* Stores vector categories.            */
                  feature_desc; /* Temporary structure for storing      */
                                /* feature descriptions.                */


#ifndef MAIN
```

```
    extern                          /* Externally defined if not main()     */
#endif

CELL            raster[401][401];
                                    /* Array holding entire raster tile.    */
```

main.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                                  m.in.ntf                             ***/
/***        GRASS module to read in any Ordnance Survey NTF data.          ***/
/***            Assumes National Transfer Format V2.0 (BS 7567).           ***/
/***                                                                       ***/
/***  Jo Wood, Project ASSIST, V2.1 19th May 1993                          ***/
/***                                                                       ***/
/*****************************************************************************/

#define MAIN

#include "ntf_in.h"             /* Must be included for program to work */

main(argc,argv)
    int argc;
    char *argv[];
{

    /*-------------------------------------------------------------------*/
    /*                  INITIALISE GLOBAL VARIABLES                      */
    /*-------------------------------------------------------------------*/

    O_raster    = FALSE;
    O_vector    = FALSE;

    O_cont      = FALSE;
    O_spot      = FALSE;
    O_lake      = FALSE;
    O_break     = FALSE;
    O_coast     = FALSE;
    O_form      = FALSE;

    num_rlines  = 0;

    X_min       = 0;
    Y_min       = 0;
    X_max       = 0;
    Y_max       = 0;
    X_origin    = 0;
    Y_origin    = 0;

    XY_mult     = 1.0;
    Z_mult      = 1.0;

    strcpy (H_organ,"Not Known                    ");
```

```
    strcpy (H_ddate,"          ");
    strcpy (H_mname,"Not Known                              ");
    strcpy (H_mdate,"          ");
    strcpy (V_name,"                                        ");
    H_scale = 0;

    points = Vect_new_line_struct();

    Write_vect = FALSE;

                        /* Initialise the two category classess, one for vector */
                        /* classes, the other for feature descriptions.         */
    G_init_cats((CELL)0,"Ordnance Survey NTF",&cats);
    G_init_cats((CELL)0,"Feature Descriptions",&feature_desc);


    /*--------------------------------------------------------------------*/
    /*                          GET INPUT FROM USER                       */
    /*--------------------------------------------------------------------*/

    interface(argc,argv);


    /*--------------------------------------------------------------------*/
    /*                       OPEN INPUT AND OUTPUT FILES                  */
    /*--------------------------------------------------------------------*/

    open_files();


    /*--------------------------------------------------------------------*/
    /*                           PROCESS NTF FILE                         */
    /*--------------------------------------------------------------------*/

    read_ntf();


    /*--------------------------------------------------------------------*/
    /*                  CLOSE ALL OPENED FILES AND FREE MEMORY            */
    /*--------------------------------------------------------------------*/

    if (outfile)
        close_vect();

    fclose(ntf_fptr);

    G_free_cats(&cats);
    G_free_cats(&feature_desc);


}
```

interface.c

```
/*****************************************************************************/
/***                            interface()                                ***/
/***    Function to get input from user and check files can be opened       ***/
/***                                                                       ***/
/***      Jo Wood, Department of Geography, V1.2, 7th February 1992        ***/
/*****************************************************************************/

#include "ntf_in.h"

interface(argc,argv)

    int      argc;              /* Number of command line arguments.    */
    char    *argv[];            /* Contents of command line arguments.  */

{
    /*-------------------------------------------------------------------*/
    /*                            INITIALISE                             */
    /*-------------------------------------------------------------------*/

    struct Option *ntf_in;      /* Pointer to a structure holding       */
                                /* all the text to describe each option.*/
                                /* It also stores the user's input.     */

    struct Option *file_out;


    struct Flag   *out_log;     /* Conversion log flag.                 */

    G_gisinit (argv[0]);        /* This GRASS library function MUST be
                                /* called to initialise the program.    */

    /*-------------------------------------------------------------------*/
    /*                        SET PARSER OPTIONS                         */
    /*-------------------------------------------------------------------*/

    /* Each option needs a 'key' (short description), a 'description`   */
    /* (a longer one), a 'type' (eg intiger, or string), and an         */
    /* indication whether manditory or not.                             */

    ntf_in                = G_define_option();
    ntf_in->key           = "ntf";
    ntf_in->description   = "NTF file to read";
    ntf_in->type          = TYPE_STRING;
    ntf_in->required      = YES;

    file_out              = G_define_option();
    file_out->key         = "out";
    file_out->description = "Output file";
    file_out->type        = TYPE_STRING;
    file_out->required    = NO;
    file_out->answer      ="No output file";
```

```
    out_log                = G_define_flag();
    out_log->key           ='l';
    out_log->description   ="Sends an NTF conversion log to standard output";

    if (G_parser(argc,argv))     /* Actually performs the prompting for  */
        exit(-1);                /* keyboard input. Returns 0 on sucess. */

                                 /* Transfer command line answers        */
                                 /* to variables.                        */
    ntf_in_name = ntf_in->answer;
    file_out_name = file_out->answer;

    conversion_log = out_log->answer;

    if (strcmp(file_out_name,"No output file"))
        outfile = TRUE;
    else
        outfile = FALSE;

}
```

open_files.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                            open_files()                               ***/
/***                       Opens NTF input file.                           ***/
/***               Jo Wood, Department of Geography, 19th May 1993         ***/
/***                                                                       ***/
/*****************************************************************************/

#include "ntf_in.h"


open_files()
{
    /* Open existing file and set the input file descriptor. */

    if ( (ntf_fptr=fopen(ntf_in_name,"r")) == NULL)
    {
        char err[256];
        sprintf(err,"Problem opening NTF file.");
        G_fatal_error(err);
    }

    line_number= 0;              /* Initialise line counter      */

}
```

open_rast.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                          open_raster()                                ***/
/***                Opens output GRASS raster file.                        ***/
/***             Jo Wood, Dept. of Geography, 29th May 1993                ***/
/***                                                                       ***/
/*****************************************************************************/

#include "ntf_in.h"

open_raster(xorigin,yorigin,nrows,ncols)
    int xorigin,yorigin,        /* Origin of SW corner of raster.       */
        nrows,ncols;            /* Number of rows and columns in raster.*/
{
    /*---------------------------------------------------------------------*/
    /*                            INITIALISE                               */
    /*---------------------------------------------------------------------*/

    struct  Cell_head   region;         /* Stores GRASS region information.     */


    /*---------------------------------------------------------------------*/
    /*                CHECK OUTPUT RASTER DOES NOT ALREADY EXIST            */
    /*---------------------------------------------------------------------*/

    mapset_out = G_mapset();            /* Set output to current mapset.        */

    if (G_legal_filename(file_out_name)==NULL)
    {
        char err[256];
        sprintf(err,"Illegal GRASS file name. Please try another.");
        G_fatal_error(err);
    }
    else
    {
        if (G_find_cell2(file_out_name,mapset_out) !=NULL)
        {
            char err[256];
            sprintf(err,"Raster map [%s] already exists.\n",file_out_name);
            G_fatal_error(err);
        }
    }


    /*---------------------------------------------------------------------*/
    /*                  CHECK CURRENT REGION AND CHANGE IF NECESSARY        */
    /*---------------------------------------------------------------------*/

    G_get_window(&region);

    if ( (region.ew_res != 50) || (region.ns_res != 50) ||
         (region.south != yorigin-25.0)                 ||
         (region.west  != xorigin-25.0)                 ||
```

```
        (region.north != yorigin + (50*(nrows-1) + 25))||
        (region.east  != xorigin + (50*(ncols-1) + 25))  )
    {
        char warn[256];
        sprintf(warn, "Current GRASS region does not match NTF grid.");
        G_warning(warn);

        region.ew_res = (double) 50.0;
        region.ns_res = (double) 50.0;
        region.south  = (double) (yorigin - 25.0);
        region.west   = (double) (xorigin - 25.0);
        region.north  = (double) (yorigin + (50*(nrows-1) + 25));
        region.east   = (double) (xorigin + (50*(ncols-1) + 25));

        G_set_window(&region);
    }


    /*--------------------------------------------------------------------*/
    /*                    OPEN THE NEW RASTER OUTPUT FILE                 */
    /*--------------------------------------------------------------------*/


    if ( (fd_out=G_open_cell_new(file_out_name)) <0)
    {
        char err[256];
        sprintf(err,"Problem opening output raster file.");
        G_fatal_error(err);
    }

    O_raster = TRUE;                    /* Flag to indicate file is open.       */
}
```

open_vect.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                           open_vector()                               ***/
/***                    Opens  GRASS vector files.                         ***/
/***                Jo Wood, Dept. of Geography, 30th May 1993.            ***/
/***                                                                       ***/
/*****************************************************************************/

#include "ntf_in.h"

open_vector(extension)
    char        *extension;             /* Filename extension.                  */
{
    /*--------------------------------------------------------------------*/
    /*                              INITIALISE                            */
    /*--------------------------------------------------------------------*/

    struct dig_head  vect_head;         /* Structure holding vector header info */
    struct Cell_head region;            /* Structure to hold region information */
```

```
char            text[255],        /* Used to hold a line of text.        */
                vect_out_name[80]; /* Name of vector to open.            */

strcpy(vect_out_name,file_out_name);
strcat(vect_out_name,extension);    /* Append extension to default file name*/

G_get_window(&region);


/*--------------------------------------------------------------------------*/
/*                          OPEN NEW VECTOR FILE                            */
/*--------------------------------------------------------------------------*/

mapset_out = G_mapset();            /* Current mapset.                      */

if (G_legal_filename(vect_out_name)==NULL)
{
    char err[256];
    sprintf(err,"Illegal vector file name. Please try another.");
    G_fatal_error(err);
}
else
{
    if (G_find_vector2(vect_out_name,mapset_out) !=NULL)
    {
        char err[256];
        sprintf(err,
            "Vector map [%s] exists.\nPlease try another.\n",vect_out_name);
        G_fatal_error(err);
    }
}

if (Vect_open_new(&vect_info,vect_out_name) < 0)
{
    char err[256];
    sprintf(err,"Unable to open vector map [%s].\n",vect_out_name);
    G_fatal_error(err);
}

if ( (vect_info.att_fp = G_fopen_new("dig_att",vect_out_name)) <0)
{
    char err[256];
    sprintf(err,"Unable to open vector attribute map [%s].\n",vect_out_name);
    G_fatal_error(err);
}

O_vector = TRUE;

/*--------------------------------------------------------------------------*/
/*                    WRITE OUT VECTOR HEADER INFORMATION                   */
/*--------------------------------------------------------------------------*/

strcpy(vect_head.organization,H_organ);
```

```
    strcpy(vect_head.date,H_ddate);

    sprintf(text,"created by m.in.ntf");
    strcpy(vect_head.your_name,text);

    strcpy(vect_head.map_name,H_mname);

    strcpy(vect_head.source_date,H_mdate);

    vect_head.orig_scale        = H_scale;
    vect_head.line_3[0]         = 0;
    vect_head.plani_zone        = region.zone;
    vect_head.digit_thresh      = 0;
    vect_head.map_thresh        = 0;

    vect_head.N    = (double)Y_max;
    vect_head.S    = (double)Y_min;
    vect_head.E    = (double)X_max;
    vect_head.W    = (double)X_min;


    Vect_copy_head_data(&vect_head,&(vect_info.head));
                                            /* This line MUST be included   */
                                            /* as it tells GRASS that the   */
                                            /* header has been updated.     */

}
```

old2new.c

```
/*****************************************************************************/
/***                            old2new()                                  ***/
/***              Converts NTF Version 1.1 to NTF Version 2.0              ***/
/***  Jo Wood, Project ASSIST, V2.0 3rd June 1993                          ***/
/***                                                                       ***/
/*****************************************************************************/

#include "ntf_in.h"

old2new()
{
    /*-------------------------------------------------------------------*/
    /*                           INITIALISE                              */
    /*-------------------------------------------------------------------*/

    char newfile[128],          /* Name of new NTF file to create.          */
         rec_desc,              /* Record Descriptor.                       */
         end_of_file=FALSE,     /* Flag indicating end of file.             */
         vol_num = 1,           /* Counts through multiple volume numbers.  */
         fixed_length = FALSE;  /* Flag indicating fixed/variable length files. */

    /*-------------------------------------------------------------------*/
    /*                GET NAME OF NEW NTF FILE TO CREATE                 */
    /*-------------------------------------------------------------------*/
```

```
    fprintf(stderr,"WARNING: Volume header indicates old NTF version.\n");

    do
    {
        fprintf(stderr,
                "Would you like to create a new NTF Version 2.00 file ?\n> ");
    }
    while ( G_gets(newfile) == 0);


    if( (strcmp(newfile,"YES") !=0) &&  (strcmp(newfile,"Yes") !=0) &&
        (strcmp(newfile,"yes") !=0) &&  (strcmp(newfile,"Y")   !=0) &&
        (strcmp(newfile,"y")   !=0) )
        exit(0);


    /* Prompt user for new file name. */

    do
    {
        fprintf(stderr,"Type in the name of the new NTF file to be created:\n> ");
    }
    while ( G_gets(newfile) == 0);


    /* Open new file */

    if ( (new_fptr=fopen(newfile,"r")) != NULL )
    {
        fprintf(stderr,
                "ERROR: File <%s> already exists. Please try another\n",newfile);
        fclose(new_fptr);
        exit(-1);
    }

    if ( (new_fptr=fopen(newfile,"w")) == NULL )
    {
        fprintf(stderr,"ERROR: Could not open output file <%s>\n",newfile);
        exit(-1);
    }


    /*-----------------------------------------------------------------*/
    /*                    CONVERT THE VOLUME HEADER RECORD             */
    /*-----------------------------------------------------------------*/

    strncpy(text+57,"0200",4);  /* Change Version number to 2.00              */

    if (*(text+61) == 'F')      /* Change from fixed to variable length records */
    {
        *(text+61) = 'V';
        fixed_length = TRUE;

        while(text[strlen(text) - 2] == ' ')
                                /* Remove surplus spaces from   */
```

```
    {                          /* end of line.                  */
        text[strlen(text) - 2] = text[strlen(text) - 1];
        text[strlen(text)-1] = NULL;
    }
}

strcpy(text+strlen(text),"%\n");
                           /* Add line terminator '%' to end.      */

if ((text[66] == '%') && (text[64] == ' ') ) /* Remove extra space. */
{
    text[64] = text[65];
    strcpy(text+65,"%\n");
}

fputs(text,new_fptr);              /* Write new record to file.    */

/*----------------------------------------------------------------------*/
/*                    CONVERT THE REST OF THE FILE                      */
/*----------------------------------------------------------------------*/

do
{
    rec_desc = read_line();

    if (fixed_length == TRUE)
        while(text[strlen(text) - 2] == ' ')
                                   /* Remove surplus spaces from   */
        {                          /* end of line.                 */
            text[strlen(text) - 2] = text[strlen(text) - 1];
            text[strlen(text) - 1] = NULL;
        }


    switch(rec_desc)
    {
        case 02:
            new02();
            break;

        case 03:
            new03();
            break;

        case 04:
            new04();
            break;

        case 05:
            new05();
            break;

        case 06:
            new06();
            break;
```

```
            case 07:
                new07();
                break;

            case 14:
                new14();
                break;

            case 15:
                new15();
                break;

            case 23:
                new23();
                break;

            case 40:
                new40();
                break;

            case 90:
                new90();
                break;

            case 99:
                if ( (text[strlen(text) - 1] == '0') ||
                     (text[strlen(text) - 2] == '0') )
                {
                    end_of_file = TRUE;
                    strcpy(text,"99End of Transfer Set0\%\n");
                }
                else
                {
                    sprintf(text,
                "99End of Volume (%d). Transfer Set continues on Volume (%d)"
                                                    ,vol_num,vol_num+1);
                    strcat (text,".1\%\n");
                    vol_num++;
                }
                fputs(text,new_fptr);           /* Write new record to file.    */
                break;

            default:
                strcpy(text+strlen(text),"%\n");/* Add line terminator '%' to end.*/
                fputs(text,new_fptr);           /* Write new record to file.    */
                break;
        }
    }
    while (end_of_file == FALSE);


    /*-----------------------------------------------------------------*/
    /*                      CLOSE NEW FILE AND QUIT                    */
    /*-----------------------------------------------------------------*/
```

```
    fclose (new_fptr);

    fprintf(stderr,"\n");
    fprintf(stderr,"*********************************************************\n");
    fprintf(stderr,"**           CONVERSION TO NTF V2.00 COMPLETE          **\n");
    fprintf(stderr,"*********************************************************\n");
    exit(0);

}
```

convert.c

```
/*************************************************************************/
/***                                                                   ***/
/***                  Conversion Routines to NTF V2.0                  ***/
/***            Jo Wood, Department of Geog, V2.0 12th June 1993       ***/
/***                                                                   ***/
/*************************************************************************/

new02() /* ADD 3 EMPTY ELEMENTS */
{

    strcpy(text+strlen(text),"%\n");     /* Add line terminator '%' to end.*/

    if (strncmp(text+2,"OS_1:625",8) == 0)       /* Change DBNAME field. */
        strncpy(text+2,"OS_ROUTEPLANNER_DATA",20);

    fputs(text,new_fptr);                /* Write new record to file.    */

    if ( text[strlen(text) - 3] == '1') /* Modify continuation record.  */
    {
        read_line();
        strcpy(text+30,"                    00000000000%\n");
        fputs(text,new_fptr);            /* Write new record to file.    */
    }
}

new03() /* ADD 3 EMPTY ELEMENTS */
{
    char        temp[80],                /* Temporarily stores new record. */

                cont_mark;

    strncpy(temp,text+50,strlen(text) - 51);
    temp[strlen(text)-51] = '\\';
    strncpy(temp+strlen(text)-50,text+20,6);
    strncpy(temp+strlen(text)-45,text+26,24);
    strcpy(temp+strlen(text)-23,"  0\%\n");

    cont_mark = text[strlen(text) - 3];

    strcpy(text+20,temp);
    fputs(text,new_fptr);                /* Write new record to file.    */
```

```
    if (cont_mark == '1')              /* Ignore any continuation lines. */
    {
        read_line();

        while (text[strlen(text)] == '1')
            read_line();
    }
}

new04() /* ADD ONE EMPTY ELEMENT */
{
    char        cont_mark,             /* Identifies continuation lines.       */
                posn=16,offset=0,      /* Position along string.               */
                temp[90];              /* Temporary storage of record.         */

    while(posn+11 < strlen(text)-10)
    {
        strncpy(temp+posn-16+offset,text+posn,10);
        temp[posn-6+offset] = ' ';
        posn +=10;
        offset ++;
    }
    strcpy(temp+posn-16+offset,text+posn);
    strcpy(text+16,temp);
    strcpy(text+strlen(text),"%\n");    /* Add line terminator '%' to end.      */
    fputs(text,new_fptr);              /* Write new record to file.            */

    while (text[strlen(text) - 3] == '1')/* Add extra element to               */
    {                                   /* conttinuation lines.                */
        read_line();
        offset=0;
        posn=2;
        while(posn+11 < strlen(text)-10)
        {
            strncpy(temp+posn-2+offset,text+posn,10);
            temp[posn+8+offset] = ' ';
            posn +=10;
            offset ++;
        }
        strcpy(temp+posn-2+offset,text+posn);
        strcpy(text+2,temp);
        strcpy(text+strlen(text),"%\n"); /* Add line terminator '%' to end.     */
        fputs(text,new_fptr);           /* Write new record to file.           */
    }
}


new05() /* ADD THE FEAT_CODE ELEMENT */
{
    char buffer[5],
         letter_code;
    int  feat_code;

    strcpy(text+strlen(text),"%\n");    /* Add line terminator '%' to end.      */
```

```
    if ((strncmp(text+2,"    ",4) == 0) &&      /* If attribute is referenced   */
        (strncmp(text+6,"          ",10) != 0)) /* by CODE_COM, convert it into */
                                                /* 4 digit FC                   */
    {

        /* To convert a 7 digit CODE_COM to a 4 digit FEAT_CODE:

                1. Feature code comprises of digits 3,4,5 & 6 of CODE_COM.
                2. Add the numeric equivalent (A=0, Z=25) of digit 1.
                3. Add 26 * digit 2.
                4. Add 260 for Line data, and 520 for Area data.

           eg.
                H11032L  becomes 1032 + 7 + (26*1) + 260 = 1325

           Note, there could be an overflow, if the central 4 digits > 9220

        */

        strncpy(buffer, text+8,4);              /* Copy main numeric code       */
        buffer[4] = NULL;
        feat_code = atoi(buffer);               /* Step 1.                      */

        letter_code = text[6];                  /* Step 2.                      */
        feat_code += letter_code - 'A';
        feat_code += 26 * text[7];              /* Step 3.                      */

        switch(text[12])                        /* Step 4.                      */
        {
            case 'P':
                feat_code += 0 * 260;
                break;
            case 'L':
                feat_code += 1 * 260;
                break;
            case 'A':
                feat_code += 2 * 260;
                break;
            default:
                fprintf(stderr,"WARNING: Non standard attribute code.\n");
                break;
        }

        sprintf(text+2,"%d",feat_code);
        text[6] = letter_code;
    }

    fputs(text,new_fptr);                       /* Write new record to file.    */

}


new06() /* IGNORE THIS RECORD */
{
    while (text[strlen(text)-1] == '1')
```

```
        read_line();
}

new07() /* CONVERT Z_DATUM TO 10 DIGITS */
{
    char        cont_mark;                      /* Identify continuation lines. */

    strcpy(text+74,"  1\%\n");
    fputs(text,new_fptr);                       /* Write new record to file.    */

    read_line();
    cont_mark = text[strlen(text)-1];
    strcpy(text+strlen(text)-1,"0%\n");         /* Add terminator '%' to end.   */
    fputs(text,new_fptr);                       /* Write new record to file.    */


    if (cont_mark == '1')
    {
        do                                      /* Read following lines.        */
        {
            read_line();
        }
        while (text[strlen(text)-1] == '1');
    }
}


new14() /* CONVERT ANY LC(LONG CODE) VALUES TO FC (FEAT_CODE) VALUES */
{
    char buffer[5],
         letter_code,
         new_text[80],
         new_textscan=0,
         orient,
         txt_string=FALSE;

    int  feat_code,
         textscan;

    do
    {
        for (textscan=0; textscan<strlen(text); textscan++)
        {
            if ((strncmp(text+textscan,"LC",2) ==0 ) && (txt_string == FALSE))
            {
                /* To convert a 7 digit CODE_COM to a 4 digit FEAT_CODE:

                1. Feature code comprises of digits 3,4,5 & 6 of CODE_COM.
                2. Add the numeric equivalent (A=0, Z=25) of digit 1.
                3. Add 26 * digit 2.
                4. Add 260 for Line data, and 520 for Area data.

                   eg.
                        H11032L  becomes 1032 + 7 + (26*1) + 260 = 1325
```

```
            Note, there could be an overflow, if the central 4 digits > 9220

            */

            strncpy(buffer, text+textscan+4,4);  /* Copy main numeric code  */
            buffer[4] = NULL;
            feat_code = atoi(buffer);            /* Step 1.                 */

            letter_code = text[textscan+2];      /* Step 2.                 */
            feat_code += letter_code - 'A';
            feat_code += 26 * text[textscan+3];  /* Step 3.                 */

            switch(text[textscan+8])             /* Step 4.                 */
            {
                case 'P':
                    feat_code += 0 * 260;
                    break;
                case 'L':
                    feat_code += 1 * 260;
                    break;
                case 'A':
                    feat_code += 2 * 260;
                    break;
                default:
                    fprintf(stderr,"WARNING: Non standard attribute code.\n");
                    break;
            }
            sprintf(new_text+new_textscan,"FC%d",feat_code);
            new_textscan += 6;
            textscan     += 9;
        }
        else
            if ((strncmp(text+textscan,"OR",2) == 0) && (txt_string == FALSE))
            {
                for (orient=0; orient<5; orient++)
                    *(new_text+new_textscan+orient) = *(text+textscan+orient);

                new_textscan += orient;
                textscan += orient+1;        /* Ignore last digit of number. */
            }
            else
            {
                *(new_text+new_textscan) = *(text+textscan);
                new_textscan++;

                if ((strncmp(text+textscan,"PN",2) == 0) ||
                    (strncmp(text+textscan,"NU",2) == 0) )
                    txt_string = TRUE;
                else
                    if (*(text+textscan) == '\\')
                        txt_string = FALSE;
            }
    }

}
```

```
    while (*(text+strlen(text)-1) == 1);    /* Repeat for continuation lines.    */

    strcpy(new_text+new_textscan,"%\n");    /* Add line terminator '%' to end.   */

    fputs(new_text,new_fptr);               /* Write new record to file.         */
}


new15() /* REMOVE NAME ID AND ADD NUM_ATT */
{
    char cont_mark;                         /* Identifies continuation lines.    */
    cont_mark = text[strlen(text) - 1];

    strcpy(text+strlen(text),"%\n");        /* Add line terminator '%' to end.   */

    if (text[27] == '%')                    /* Default configuration             */
    {
        text[16] = text[14];                /* Move ATT_ID forward two chars.    */
        text[17] = text[15];
        text[18] = text[16];
        text[19] = text[17];
        text[20] = text[18];
        text[21] = text[19];

        text[14] = '0';                 /* Add NUM_ATT                           */
        text[15] = '1';


        text[22] = text[26];            /* Move terminator back 4 spaces.        */
        strcpy(text+23,"%\n");
    }
    else
        if ((text[37] == '%') && (text[35] == ' ') ) /* Remove extra space.      */
        {
            text[35] = text[36];
            strcpy(text+36,"%\n");
        }

    fputs(text,new_fptr);               /* Write new record to file.             */

}


new23() /* REMOVE NAME ID AND ADD NUM_ATT */
{
    char cont_mark;                     /* Identifies continuation lines.        */
    cont_mark = text[strlen(text) - 1];

    strcpy(text+strlen(text),"%\n");    /* Add line terminator '%' to end.       */

    if (text[27] == '%')                /* Default configuration                 */
    {
        text[16] = text[14];            /* Move ATT_ID forward two characters.   */
        text[17] = text[15];
        text[18] = text[16];
```

```
        text[19] = text[17];
        text[20] = text[18];
        text[21] = text[19];

        text[14] = '0';                 /* Add NUM_ATT                          */
        text[15] = '1';


        text[22] = text[26];            /* Move terminator back 4 spaces.       */
        strcpy(text+23,"%\n");
    }
    else
        if ((text[37] == '%') && (text[35] == ' ') ) /* Remove extra space.     */
        {
            text[35] = text[36];
            strcpy(text+36,"%\n");
        }

    fputs(text,new_fptr);                       /* Write new record to file.    */
}

new40() /* ADD SECOND DIVIDER TO END OF STRING */
{
    char cont_mark;                             /* Identify continuation lines. */
    cont_mark = text[strlen(text) - 1];


    strcpy(text+strlen(text)-1,"\\0%\n");       /* Add second '\' to string and */
                                                /* line terminator '%' to end.  */
    text[strlen(text) - 3] = cont_mark;

    fputs(text,new_fptr);                       /* Write new record to file.    */

}



new90() /* IGNORE THIS RECORD */
{
    while (text[strlen(text)-1] == '1')
        read_line();
}
```

read_ntf.c

```
/*************************************************************************/
/***                          read_ntf()                               ***/
/**    Function to read in Ordnance Survey NTF record description data. ***/
/***              Assumes National Transfer Format V2.0 (BS 7567).     ***/
/***                                                                   ***/
/***  Jo Wood, Dept of Geography, V2.0 19th May 1993                   ***/
/***                                                                   ***/
/*************************************************************************/

#include "ntf_in.h"

read_ntf()
{
    /*-----------------------------------------------------------------*/
    /*                          INITIALISE                             */
    /*-----------------------------------------------------------------*/

    char        rec_desc,       /* Record Descriptor.                  */
                cont_mark,      /* Continuation code - 0 is no continue,
                                                        1 to continue. */

                end_of_file = FALSE;
                                /* Flag -> all volumes have been read.  */

    /*-----------------------------------------------------------------*/
    /*                    SCAN EACH RECORD IN NTF FILE                 */
    /*-----------------------------------------------------------------*/

    do
    {
        rec_desc = read_line();

        switch (rec_desc)
        {
            case 1:                     /* MAP HEADER INFORMATION       */
                vol_head();
                break;
            case 2:
                db_head();
                break;
            case 3:
                data_desc();
                break;
            case 4:
                data_fmt();
                break;
            case 5:
                feat_class();
                break;

            case 7:                     /* SECTION HEADER INFORMATION   */
                sect_head();
                break;
```

```
case(11):                       /* TEXT INFORMATION              */
    name_rec();
    break;
case(12):
    name_postn();
    break;
case(43):
    text_rec();
    break;
case(44):
    text_postn();
    break;
case(45):
    text_rep();
    break;


case(14):                       /* ATTRIBUTE INFORMATION         */
    att_rec();
    break;
case 40:
    att_desc();
    break;

case 15:                        /* VECTOR COORD INFORMATION      */
    point_rec();
    break;
case 16:
    node_rec();
    break;
case 21:
    geom_rec(NO_ATTRIB,NO_ATTRIB);
    break;
case 22:
    geom_rec2(NO_ATTRIB,NO_ATTRIB);
    break;
case 23:
    line_rec();
    break;

case 50:                        /* RASTER INFORMATION            */
    grid_head();
    break;

case 51:
    grid_rec();
    break;

case 99:                        /* VOLUME TERMINATION INFO       */
    if (vol_term() == 0)
        end_of_file = TRUE;
    break;

case 0:                         /* ERRORS                        */
    cont_error();
```

```
                break;
            default:
                unknown_desc(rec_desc);
                break;
        }
    }
    while (end_of_file == FALSE);

}
```

read_line.c

```
/*****************************************************************************/
/***                          read_line()                                  ***/
/***   Reads a line from an NTF file and returns the record descriptor     ***/
/***   Jo Wood, Department of Geography, V2.0 27th May 1993                ***/
/*****************************************************************************/

#include "ntf_in.h"

int read_line()
{
    /* ------ Initialise ------ */

    char rec_desc[3],           /* Stores 2 digit ASCII record descriptor       */
         *posn_ptr=NULL;        /* Points to contrl-M character in string.      */

    rec_desc[2] =NULL;


    /* ------ Read in a line (record) from the NTF file ------ */

    if (fgets(text,128,ntf_fptr) == NULL)
    {
        fprintf(stderr,"ERROR during read of NTF file\n");
        exit(-1);
    }

    line_number ++;             /* Accumulate line counter.                     */


    /* ------ Strip line of Ctrl-M (Return) Characters ------ */

    posn_ptr = strchr(text,13);
    if (posn_ptr != NULL)
       *(posn_ptr) = NULL;      /* Terminate string at first Ctrl-M.            */


    /* ------ Find and return the Record Descriptor ------ */

    strncpy(rec_desc,text,2);           /* Copy first two characters.           */
    return (atoi(rec_desc));            /* Convert into integer.                */
}
```

errors.c

```
/*****************************************************************************/
/***                            cont_error()                               ***/
/*** Identifies and reports error if unexpected continuation code found.  ***/
/***                                                                       ***/
/***  Jo Wood, Project ASSIST, V1.0 27th May 1993                          ***/
/***                                                                       ***/
/*****************************************************************************/

cont_error()
{
    /*-------------------------------------------------------------------*/
    /*                          INITIALISE                               */
    /*-------------------------------------------------------------------*/

    char        *posn_ptr;      /* Position in string of continuation code.     */
    char        cont_mark;      /* Continuation code.                           */

    fprintf(stderr,
        "WARNING: Line %d - Record contains unexpected continuation code.\n",
                                                            line_number);

    /*-------------------------------------------------------------------*/
    /*                  IDENTIFY CONTINUATION CODE                       */
    /*-------------------------------------------------------------------*/

    if ((posn_ptr = strrchr(text,'%')) == NULL)
        fprintf(stderr,
        "WARNING: Line %d - No record terminator found in line.\n",line_number);

    cont_mark = *(posn_ptr-1) - '0';

    /*-------------------------------------------------------------------*/
    /*                   READ CONTINUATION LINES                         */
    /*-------------------------------------------------------------------*/

    if (cont_mark != 0)
        while (read_blank() != 0)       /* Read following continuation lines.   */
        ;

}


/*****************************************************************************/
/***                           unknown_desc()                              ***/
/*** Identifies and reports error if unknown description code is found.  ***/
/***                                                                       ***/
/***  Jo Wood, Department of Geography, V1.0 27th May 1993                 ***/
/***                                                                       ***/
/*****************************************************************************/

unknown_desc(rec_desc)
    char rec_desc;              /* The unknown record descriptor.       */
```

```
{
    /*-----------------------------------------------------------------*/
    /*                          INITIALISE                             */
    /*-----------------------------------------------------------------*/

    char        *posn_ptr;      /* Position in string of continuation code.     */
    char        cont_mark;      /* Continuation code.                           */

    fprintf(stderr,"WARNING: Line %d - Unknown record descriptor [%d].\n",
                                                    line_number,rec_desc);

    /*-----------------------------------------------------------------*/
    /*                IDENTIFY CONTINUATION CODE                       */
    /*-----------------------------------------------------------------*/

    if ((posn_ptr = strrchr(text,'%')) == NULL)
        fprintf(stderr,
        "WARNING: Line %d - No record terminator found in line.\n",line_number);

    cont_mark = *(posn_ptr-1) - '0';

    /*-----------------------------------------------------------------*/
    /*                    READ CONTINUATION LINES                      */
    /*-----------------------------------------------------------------*/
    if (cont_mark != 0)
        while (read_blank() != 0)       /* Read following continuation lines.   */
        ;
}



/*************************************************************************/
/***                          read_blank()                             ***/
/***       Reads in an NTF record when the contents are to be ignored.  ***/
/***                                                                    ***/
/***              Jo Wood, Dept. of Geography, V2.0 27th May 1993       ***/
/***                                                                    ***/
/*************************************************************************/

read_blank()
{
    /*-----------------------------------------------------------------*/
    /*                          INITIALISE                             */
    /*-----------------------------------------------------------------*/



    char    cont_mark,          /* Continuation code ('0' or '1').              */
            rec_desc_txt[3],    /* Stores 2 digit ASCII record descriptor       */
            rec_desc,           /* Integer record descriptor.                   */
            *posn_ptr;          /* Position of characters in string.            */

    rec_desc_txt[2] = NULL;
```

```
    /*--------------------------------------------------------------------*/
    /*                  READ A LINE FROM THE NTF FILE                     */
    /*--------------------------------------------------------------------*/


    /* ------ Read in a line (record) from the NTF file ------ */

    if (fgets(text,128,ntf_fptr) == NULL)
    {
        fprintf(stderr,"ERROR during read of NTF line.\n");
        exit(-1);
    }
    line_number++;              /* Accumulate line counter.                */

    /*--------------------------------------------------------------------*/
    /*                      CHECK RECORD DESCRIPTOR                       */
    /*--------------------------------------------------------------------*/

    strncpy(rec_desc_txt,text,2);       /* Copy first two characters.      */
    rec_desc = atoi(rec_desc_txt);      /* Convert into integer.           */
    if (rec_desc != 0)
        fprintf(stderr,
        "WARNING: Line %d - Record descriptor is %d for continuation line.\n",
                                                    line_number,rec_desc);

    /*--------------------------------------------------------------------*/
    /*                     IDENTIFY CONTINUATION CODE                     */
    /*--------------------------------------------------------------------*/

    if ((posn_ptr = strrchr(text,'%')) == NULL)
        fprintf(stderr,
        "WARNING: Line %d - No record terminator found in continuation line.\n",
                                                        line_number);

    cont_mark = *(posn_ptr-1) - '0';

    return(cont_mark);
}
```

write_rast.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                          write_raster()                               ***/
/***                Writes out and closes  GRASS raster file.              ***/
/***                Jo Wood, Dept of Geography, 29th May 1993.             ***/
/***                                                                       ***/
/*****************************************************************************/

#include "ntf_in.h"

write_raster()
{
```

```
    /* ------ Initialise ------ */

    int    rast_row,rast_col;           /* Counts through each raster row.   */

    CELL  *row_buf;
    row_buf  = G_allocate_cell_buf();   /* Allocate enough memory to store   */
                                        /* one raster row.                   */

    /* ------ Copy raster to GRASS buffer and write out ------ */

    for (rast_row=0;rast_row<401; rast_row++)
    {
        row_buf = raster[rast_row];
        G_put_map_row(fd_out, row_buf); /* Write the row buffer to           */
                                        /* the output raster.                */
    }

    /* ------ Close GRASS raster ------ */

    G_close_cell(fd_out);
    G_init_cats((CELL)0,"Elevation",&cats);
    G_write_cats(file_out_name,&cats);

    /* ------ Update file name so new one may be opened ------ */

    strcat(file_out_name,"%");

    O_raster     = FALSE;               /* Flag to indicate file is now closed  */
    num_rlines   = 0;

}
```

date.c

```
/*****************************************************************************/
/***                                get_date()                             ***/
/***          Converts numerical date into 10 character text date          ***/
/***                Jo Wood, Dept. of Geography, 31st May 1993.            ***/
/***                                                                       ***/
/*****************************************************************************/

#include "ntf.h"

get_date(date_num)
    char        *date_num;              /* Text string holding numerical     */
                                        /* version of date: YYYYMMDD         */
{

     char       date_txt[11],
                pair[3];                /* Pair of digits YY or MM or DD     */


     strcpy(date_txt,"          ");
```

```
 strcpy(pair,"  ");


/* Check for NULL strings */

if ( (strcmp(date_num,"00000000")==0) || (strcmp(date_num,"        ")==0) )
{
    strcpy(date_num,"Not Given");
    return(0);
}


/* Get Day */

strncpy(date_txt,date_num+6,2);
if (*(date_num+7) == '1')
    strncpy(date_txt+2,"st",2);
else
    if (*(date_num+7) == '2')
        strncpy(date_txt+2,"nd",2);
    else
        if (*(date_num+7) == '3')
            strncpy(date_txt+2,"rd",2);
        else
            strncpy(date_txt+2,"th",2);


/* Get Month */

if (strncmp(date_num+4,"01",2) == 0 )
    strncpy(date_txt+4," Jan",4);
else
    if (strncmp(date_num+4,"02",2) == 0 )
        strncpy(date_txt+4," Feb",4);
    else
        if (strncmp(date_num+4,"03",2) == 0 )
            strncpy(date_txt+4," Mar",4);
        else
            if (strncmp(date_num+4,"04",2) == 0 )
                strncpy(date_txt+4," Apr",4);
            else
                if (strncmp(date_num+4,"05",2) == 0 )
                    strncpy(date_txt+4," May",4);
                else
                    if (strncmp(date_num+4,"06",2) == 0 )
                        strncpy(date_txt+4," Jun",4);
                    else
                        if (strncmp(date_num+4,"07",2) == 0 )
                            strncpy(date_txt+4," Jul",4);
                        else
                            if (strncmp(date_num+4,"08",2) == 0 )
                                strncpy(date_txt+4," Aug",4);
                            else
                                if (strncmp(date_num+4,"09",2) == 0 )
                                    strncpy(date_txt+4," Sep",4);
```

```
                                    else
                                        if (strncmp(date_num+4,"10",2) == 0 )
                                            strncpy(date_txt+4," Oct",4);
                                        else
                                            if (strncmp(date_num+4,"11",2) == 0 )
                                                strncpy(date_txt+4," Nov",4);
                                            else
                                                if (strncmp(date_num+4,"12",2) ==0 )
                                                    strncpy(date_txt+4," Dec",4);
    /* Get Year */
    strncpy(date_txt+8,date_num+2,2);

    strcpy(date_num,date_txt);
}
```

write_vect.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                          write_vector()                               ***/
/***                Writes out to GRASS vector files.                      ***/
/***              Jo Wood, Dept of Geography, 30th May 1993.               ***/
/***                                                                       ***/
/*****************************************************************************/

#include "ntf_in.h"

write_vector(geom_type,attribute)
    int         geom_type,              /* Point, Line or Area              */
                attribute;              /* Numerical attribute.             */
{
    /*-------------------------------------------------------------------*/
    /*                           INITIALISE                              */
    /*-------------------------------------------------------------------*/

    FILE        *vect_fptr,             /* Points to one of the opened vect maps*/
                *vect_fptr_att;

    /*-------------------------------------------------------------------*/
    /*                  FIND OUT WHICH VECTOR TO WRITE TO                */
    /*-------------------------------------------------------------------*/

    switch (V_featcode)                 /* Each feature code written out to a   */
    {                                   /*  separate vector map layer.          */

        case 200:                       /* Spot heights.                    */
            if (O_spot == FALSE)        /* If false, haven't been opened yet.   */
            {
                open_vector(".spot");
                fptr_spot = vect_info.dig_fp;
                fptr_spot_att = vect_info.att_fp;
                O_spot = TRUE;
            }
```

```
    vect_fptr = fptr_spot;      /* This is the file to write to.        */
    vect_fptr_att = fptr_spot_att;
    break;

case 201:                       /* Contour lines.                       */
    if (O_cont == FALSE)        /* If false, haven't been opened yet.   */
    {
        open_vector(".cont");
        fptr_cont = vect_info.dig_fp;
        fptr_cont_att = vect_info.att_fp;
        O_cont = TRUE;
    }
    vect_fptr = fptr_cont;      /* This is the file to write to.        */
    vect_fptr_att = fptr_cont_att;
    break;

case 202:                       /* Lake boundaries.                     */
    if (O_lake == FALSE)        /* If false, haven't been opened yet.   */
    {
        open_vector(".lake");
        fptr_lake = vect_info.dig_fp;
        fptr_lake_att = vect_info.att_fp;
        O_lake = TRUE;
    }
    vect_fptr = fptr_lake;      /* This is the file to write to.        */
    vect_fptr_att = fptr_lake_att;
    break;

case 203:                       /* Break lines.                         */
    if (O_break == FALSE)       /* If false, haven't been opened yet.   */
    {
        open_vector(".break");
        fptr_break = vect_info.dig_fp;
        fptr_break_att = vect_info.att_fp;
        O_break = TRUE;
    }
    vect_fptr = fptr_break;     /* This is the file to write to.        */
    vect_fptr_att = fptr_break_att;
    break;

case 204:                       /* Coastlines.                          */
    if (O_coast == FALSE)       /* If false, haven't been opened yet.   */
    {
        open_vector(".coast");
        fptr_coast = vect_info.dig_fp;
        fptr_coast_att = vect_info.att_fp;
        O_coast = TRUE;
    }
    vect_fptr = fptr_coast;     /* This is the file to write to.        */
    vect_fptr_att =fptr_coast_att;
    break;

case 205:                       /* Ridge lines.                         */
    if (O_temp == FALSE)        /* If false, haven't been opened yet.   */
    {
```

```
            open_vector(".ridge");
            fptr_ridge = vect_info.dig_fp;
            fptr_ridge_att = vect_info.att_fp;
            O_temp = TRUE;
        }
        vect_fptr = fptr_ridge;     /* This is the file to write to.        */
        vect_fptr_att = fptr_ridge_att;

        break;

    case 207:                       /* Form lines.                          */
        if (O_form == FALSE)        /* If false, haven't been opened yet.   */
        {
            open_vector(".form");
            fptr_form = vect_info.dig_fp;
            fptr_form_att = vect_info.att_fp;
            O_form = TRUE;
        }
        vect_fptr = fptr_form;      /* This is the file to write to.        */
        vect_fptr_att = fptr_form_att;
        break;

    default:
        if (O_other == FALSE)       /* If false, haven't been opened yet.   */
        {
            open_vector("");
            fptr_other = vect_info.dig_fp;
            fptr_other_att = vect_info.att_fp;
            O_other = TRUE;
        }
        vect_fptr = fptr_other;     /* This is the file to write to.        */
        vect_fptr_att = fptr_other_att;
        break;
}

/*-------------------------------------------------------------------------*/
/*                  WRITE OUT X,Y COORDINATES AND ATTRIBUTES                */
/*-------------------------------------------------------------------------*/

if (vect_fptr == NULL)
    return(0);

vect_info.dig_fp = vect_fptr;
vect_info.att_fp = vect_fptr_att;
Vect_write_line(&vect_info,geom_type,points);

if (geom_type == DOT)
    write_att(vect_fptr_att, 'P', *(points->x), *(points->y), attribute);
else
    if (geom_type == LINE)
        write_att(vect_fptr_att, 'L',
                        *(points->x+1), *(points->y+1), attribute);
    else
        if (geom_type == AREA)
            write_att(vect_fptr_att, 'A', *(points->x), *(points->y),
```

```
attribute);

}
```

close_vect.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                          close_vect()                                 ***/
/***              Closes down all opened vector files.                     ***/
/***             Jo Wood, Dept. of Geography, 12 June 1993                 ***/
/***                                                                       ***/
/*****************************************************************************/

#include "ntf_in.h"


close_vect()
{

    if (O_spot == TRUE)
    {
        vect_info.dig_fp = fptr_spot;
        if (vect_info.mode == MODE_WRITE || vect_info.mode == MODE_RW)
                Vect__write_head_binary (&vect_info, &(vect_info.head));
        fclose(fptr_spot);
        fclose(fptr_spot_att);

        G_free_cats(&cats);                     /* Just save category title */
        G_init_cats((CELL)0,"Elevation",&cats);

        O_spot = FALSE;
    }

    if (O_cont == TRUE)
    {
        vect_info.dig_fp = fptr_cont;
        if (vect_info.mode == MODE_WRITE || vect_info.mode == MODE_RW)
                Vect__write_head_binary (&vect_info, &(vect_info.head));
        fclose(fptr_cont);
        fclose(fptr_cont_att);

        G_free_cats(&cats);                     /* Just save category title */
        G_init_cats((CELL)0,"Elevation",&cats);

        O_cont = FALSE;
    }

    if (O_lake == TRUE)
    {
        vect_info.dig_fp = fptr_lake;
        if (vect_info.mode == MODE_WRITE || vect_info.mode == MODE_RW)
                Vect__write_head_binary (&vect_info, &(vect_info.head));
```

```
    fclose(fptr_lake);
    fclose(fptr_lake_att);

    G_free_cats(&cats);                    /* Just save category title */
    G_init_cats((CELL)0,"Elevation",&cats);

    O_lake = FALSE;
}

if (O_break == TRUE)
{
    vect_info.dig_fp = fptr_break;
    if (vect_info.mode == MODE_WRITE || vect_info.mode == MODE_RW)
            Vect__write_head_binary (&vect_info, &(vect_info.head));
    fclose(fptr_break);
    fclose(fptr_break_att);

    G_free_cats(&cats);                    /* Just save category title */
    G_init_cats((CELL)0,"Elevation",&cats);

    O_break = FALSE;
}

if (O_coast == TRUE)
{
    vect_info.dig_fp = fptr_coast;
    if (vect_info.mode == MODE_WRITE || vect_info.mode == MODE_RW)
            Vect__write_head_binary (&vect_info, &(vect_info.head));
    fclose(fptr_coast);
    fclose(fptr_coast_att);

    G_free_cats(&cats);                    /* Just save category title */
    G_init_cats((CELL)0,"Elevation",&cats);

    O_coast = FALSE;
}

if (O_temp == TRUE)
{
    vect_info.dig_fp = fptr_ridge;
    if (vect_info.mode == MODE_WRITE || vect_info.mode == MODE_RW)
            Vect__write_head_binary (&vect_info, &(vect_info.head));
    fclose(fptr_ridge);
    fclose(fptr_ridge_att);

    G_free_cats(&cats);                    /* Just save category title */
    G_init_cats((CELL)0,"Elevation",&cats);

    O_temp = FALSE;
}

if (O_form == TRUE)
{
    vect_info.dig_fp = fptr_form;
    if (vect_info.mode == MODE_WRITE || vect_info.mode == MODE_RW)
```

```
            Vect__write_head_binary (&vect_info, &(vect_info.head));
        fclose(fptr_form);
        fclose(fptr_form_att);

        G_free_cats(&cats);                    /* Just save category title */
        G_init_cats((CELL)0,"Elevation",&cats);

        O_form = FALSE;
    }


    if (O_other == TRUE)
    {
        vect_info.dig_fp = fptr_other;
        if (vect_info.mode == MODE_WRITE || vect_info.mode == MODE_RW)
            Vect__write_head_binary (&vect_info, &(vect_info.head));
        fclose(fptr_other);
        fclose(fptr_other_att);

        G_write_vector_cats(file_out_name,&cats);
        O_other = FALSE;
    }

    if (O_vector == TRUE)
    {
        strcat(file_out_name,"%");      /* Change name of output files.        */
        strcpy (H_organ,"Not Known                    ");
        strcpy (H_ddate,"          ");
        strcpy (H_mname,"Not Known                              ");
        strcpy (H_mdate,"          ");
    }


}
```

01vol_head.c

```
/*****************************************************************************/
/***                              vol_head()                               ***/
/*** Reads in Volume header Record (01) into structure for processing    ***/
/***                                                                       ***/
/***  Jo Wood, Department of Geography, V2.0 27th May 1993                 ***/
/***                                                                       ***/
/*****************************************************************************/

#include "ntf_in.h"

vol_head()
{
    /*-------------------------------------------------------------------*/
    /*                          INITIALISE                               */
    /*-------------------------------------------------------------------*/
```

```
struct  VOLHDREC    volhdrec;       /* Volume header structure.             */
char                buffer[80];     /* Temporary storage of text data       */

/* Initialise all string structures */

strcpy(volhdrec.DONOR,      "                    ");
strcpy(volhdrec.RECIPIENT,  "                    ");
strcpy(volhdrec.TRANDATE,   "        ");

/*--------------------------------------------------------------------*/
/*                  SCAN RECORD INTO STRUCTURE                        */
/*--------------------------------------------------------------------*/

volhdrec.REC_DESC_1 =       1;
strncpy(volhdrec.DONOR,     text+2,20);
strncpy(volhdrec.RECIPIENT, text+22,20);
strncpy(volhdrec.TRANDATE,  text+42,8);
strncpy(buffer,             text+50,4);
    buffer[4] = NULL;
    volhdrec.SERIAL =       atoi(buffer);
strncpy(buffer,             text+54,2);
    buffer[2] = NULL;
    volhdrec.VOLNUM =       atoi(buffer);
volhdrec.NTFLEVEL =         text[56] - '0';
strncpy(buffer,             text+57,4);
    buffer[4] = NULL;
    volhdrec.NTFVER =       atoi(buffer) / 100.0;
volhdrec.NTFOR =            text[61];
volhdrec.EOR_1 =            text[62];

if (volhdrec.EOR_1 != '%')
{
    volhdrec.DIVIDER =      text[63];
    volhdrec.CONT_MARK_1 =  text[64] - '0';
    volhdrec.EOR_2 =        text[65];
}
else
{
    volhdrec.DIVIDER =      ' ';
    volhdrec.CONT_MARK_1 =  0;
    volhdrec.EOR_2 =        '%';
}

/*--------------------------------------------------------------------*/
/*                        CHECK NTF VERSION                           */
/*--------------------------------------------------------------------*/

if (volhdrec.NTFVER < 2.0)
    old2new();
else
    if (volhdrec.NTFVER != 2.0)
        fprintf(stderr,
"WARNING: Line %d - Volume header indicates unknown NTF version [%.2f].\n",
                                        line_number,volhdrec.NTFVER);
```

```
    /*-------------------------------------------------------------------*/
    /*                    CHECK INTEGRITY OF RECORD                      */
    /*-------------------------------------------------------------------*/

    if (volhdrec.CONT_MARK_1 != 0)
    {
        fprintf(stderr,
        "WARNING: Line %d - Volume header record has continuation mark.\n",
                                                            line_number);
        while (read_blank() != 0)       /* Read following continuation lines.   */
        ;
    }

    if (volhdrec.EOR_2 != '%')
        fprintf(stderr,
        "WARNING: Line %d - Volume header record has no line terminator.\n",
                                                            line_number);


    /*-------------------------------------------------------------------*/
    /*                    OUTPUT CONTENTS OF RECORD                      */
    /*-------------------------------------------------------------------*/

    if (conversion_log)          /* Print out log if requested.                */
    {
        printf("\n\n");
        printf("      National Transfer Format - Conversion Log\n");
        printf("      ========================================\n");
        printf("\n");
        printf("Donor:                  %s\n",volhdrec.DONOR);
        printf("Processed for           %s\n",volhdrec.RECIPIENT);
        printf("Extracted from volume   %d\n",volhdrec.VOLNUM);
        printf("Uses NTF                Version %.2f\n",volhdrec.NTFVER);
        printf("                        Level %d\n",volhdrec.NTFLEVEL);
    }

    if (outfile)
    {
        strcpy(H_organ,volhdrec.DONOR);
        strcpy(H_ddate,volhdrec.TRANDATE);
        get_date(H_ddate);
    }
}
```

02db_head.c

```
/*************************************************************************/
/***                            db_head()                              ***/
/***  Reads in Database Header Record (02) into structure for processing ***/
/***                                                                   ***/
/***  Jo Wood, Dept of Geography, V1.0 27th May 1993                   ***/
/***                                                                   ***/
/*************************************************************************/
```

```
#include "ntf_in.h"

db_head()
{
    /*-----------------------------------------------------------------*/
    /*                          INITIALISE                             */
    /*-----------------------------------------------------------------*/

    struct  DBHDREC     dbhdrec;        /* Database header structure.          */
    char                buffer[80];     /* Temporary storage of text data      */

    /* Initialise all string structures */

    strcpy(dbhdrec.DBNAME,      "                    ");
    strcpy(dbhdrec.DDNAME,      "                    ");
    strcpy(dbhdrec.DDATE,       "        ");
    strcpy(dbhdrec.DDBASE,      "                    ");
    strcpy(dbhdrec.DDBDATE,     "        ");
    strcpy(dbhdrec.FCNAME,      "                    ");
    strcpy(dbhdrec.FCDATE,      "        ");
    strcpy(dbhdrec.DQNAME,      "                    ");
    strcpy(dbhdrec.DQDATE,      "        ");


    /*-----------------------------------------------------------------*/
    /*                    SCAN RECORD INTO STRUCTURE                   */
    /*-----------------------------------------------------------------*/

    dbhdrec.REC_DESC_1  =       2;
    strncpy(dbhdrec.DBNAME,     text+2,20);
    strncpy(dbhdrec.DDNAME,     text+22,20);
    strncpy(dbhdrec.DDATE,      text+42,8);
    strncpy(dbhdrec.DDBASE,     text+50,20);
    strncpy(dbhdrec.DDBDATE,    text+70,8);
    dbhdrec.CONT_MARK_1 =       text[78] - '0';
    dbhdrec.EOR_1 =             text[79];

    if (dbhdrec.CONT_MARK_1 == 1)
    {
        if ( (dbhdrec.REC_DESC_2 = read_line()) != 0)
            fprintf(stderr, "WARNING: Line %d -
                Continuation of Database header has unexpected descriptor [%d] \n",
                                                                line_number);
        strncpy(dbhdrec.FCNAME, text+2,20);
        strncpy(dbhdrec.FCDATE, text+22,8);
        strncpy(dbhdrec.DQNAME, text+30,20);
        strncpy(dbhdrec.DQDATE, text+50,8);
        strncpy(buffer,         text+58,2);
            buffer[2] = NULL;
            dbhdrec.DATA_MODEL =atoi(buffer);
        dbhdrec.CONT_MARK_2 =   text[60] - '0';
        dbhdrec.EOR_2 =         text[61];
    }
    else
        dbhdrec.EOR_2 =         '%';
```

```
    /*-------------------------------------------------------------------*/
    /*                      CHECK INTEGRITY OF RECORD                    */
    /*-------------------------------------------------------------------*/

    if (dbhdrec.CONT_MARK_2 != 0)
    {
        fprintf(stderr,
        "WARNING: Line %d - Database header record has second continuation mark.\n",
                                                              line_number);
        while (read_blank() != 0)       /* Read following continuation lines.   */
        ;
    }

    if (dbhdrec.EOR_2 != '%')
        fprintf(stderr,
        "WARNING: Line %d - Database header record has no line terminator.\n",
                                                            ine_number);


    /*-------------------------------------------------------------------*/
    /*                      OUTPUT CONTENTS OF RECORD                    */
    /*-------------------------------------------------------------------*/

    if (conversion_log)         /* Print out log if requested.                */
    {
        printf("Database Header:\n");
        printf("----------------\n");
        printf("\tDatabase Name:\t\t%s\n",dbhdrec.DBNAME);
        printf("\tData Dictionary Name:\t%s\n",dbhdrec.DDNAME);
    }

    strcpy(H_mname,dbhdrec.DBNAME);

}
```

03data_desc.c

```
/*************************************************************************/
/***                            data_desc()                            ***/
/*** Reads in Data Description Record (03) into structure for processing ***/
/***                                                                   ***/
/***  Jo Wood, Department of Geography, V2.0 27th May 1993             ***/
/***                                                                   ***/
/*************************************************************************/

#include "ntf_in.h"

data_desc()
{
    /*-------------------------------------------------------------------*/
    /*                          INITIALISE                               */
    /*-------------------------------------------------------------------*/
```

```
    struct  DATADESC    datadesc;       /* Volume header structure.             */
    char                buffer[80],     /* Temporary storage of text data       */
                        *posn_ptr;      /* Points to '\' at end of descriptor.  */

    /* Initialise all string structures */

    strcpy(datadesc.FIELD_NAME, "          ");
    strcpy(datadesc.FINTER,     "     ");
    strcpy(datadesc.FDESC,      "                                        ");

    /*-------------------------------------------------------------------*/
    /*                       SCAN RECORD INTO STRUCTURE                  */
    /*-------------------------------------------------------------------*/

    datadesc.REC_DESC_1 =       3;
    strncpy(datadesc.FIELD_NAME,text+2,10);
    strncpy(buffer,             text+12,3);
        buffer[3] = NULL;
        datadesc.FWIDTH =       atoi(buffer);
    strncpy(datadesc.FINTER,    text+15,5);

    if ( (posn_ptr = strrchr(text,'\\')) == NULL)
        fprintf(stderr,
        "WARNING: Line %d - Data Description Record has no divider.\n",line_number);
    else
        strncpy(datadesc.FDESC, text+20,posn_ptr - (text+20));

    if ( (posn_ptr = strrchr(text,'%')) == NULL)
        fprintf(stderr,
"WARNING: Line %d - Data Description Record has no terminator.\n",line_number);

    datadesc.CONT_MARK_1 =      *(posn_ptr-1) - '0';
    datadesc.EOR_1 =            *posn_ptr;


    /*-------------------------------------------------------------------*/
    /*                      CHECK INTEGRITY OF RECORD                    */
    /*-------------------------------------------------------------------*/

    if (datadesc.CONT_MARK_1 != 0)
    {
        fprintf(stderr,
        "WARNING: Line %d - Data Description Record has continuation mark.\n",
                                                          line_number);
        while (read_blank() != 0)       /* Read following continuation lines.   */
        ;
    }

    /*-------------------------------------------------------------------*/
    /*                      OUTPUT CONTENTS OF RECORD                    */
    /*-------------------------------------------------------------------*/

    if (conversion_log)         /* Print out log if requested.                  */
    {
```

```
        printf("Data Description Record:\n");
        printf("------------------------\n");
        printf("\tField name:\t\t%s\n",datadesc.FIELD_NAME);
        printf("\tField description\t%s\n",datadesc.FDESC);
    }

}
```

04data_fmt.c

```
/*****************************************************************************/
/***                              data_fmt()                               ***/
/***    Reads in Data Format Record (04) into structure for processing     ***/
/***                                                                       ***/
/***  Jo Wood, Department of Geography, V2.0 27th May 1993                 ***/
/***                                                                       ***/
/*****************************************************************************/

#include "ntf_in.h"

data_fmt()
{
    /*-------------------------------------------------------------------*/
    /*                          INITIALISE                               */
    /*-------------------------------------------------------------------*/

    struct  DATAFMT     datafmt;        /* Data format record structure.        */
    char                buffer[80],     /* Temporary storage of text data       */
                        *posn_ptr;      /* Position of record terminator.       */

    /* Initialise all string structures */

    strcpy(datafmt.REC_NAME,    "          ");
    strcpy(datafmt.FIELD_NAME,  "          ");

    /*-------------------------------------------------------------------*/
    /*                    SCAN RECORD INTO STRUCTURE                     */
    /*-------------------------------------------------------------------*/

    datafmt.REC_DESC_1  =       4;
    strncpy(buffer,             text+2,2);
        buffer[2] = NULL;
        datafmt.REC_TYPE =      atoi(buffer);
    strncpy(datafmt.REC_NAME,   text+4,10);
    strncpy(buffer,             text+14,2);
        buffer[2] = NULL;
        datafmt.NUM_FIELD =     atoi(buffer);
    strncpy(datafmt.FIELD_NAME, text+16,10);
    datafmt.FUSE =              text[26];

    if ( (posn_ptr = strrchr(text,'%')) == NULL)
        fprintf(stderr,
        "WARNING: Line %d - Data Format Record has no terminator.\n",line_number);
```

```
    datafmt.CONT_MARK_1 =       *(posn_ptr-1) - '0';
    datafmt.EOR_1 =             *posn_ptr;


    /*-------------------------------------------------------------------*/
    /*                        CHECK INTEGRITY OF RECORD                  */
    /*-------------------------------------------------------------------*/

    if (datafmt.CONT_MARK_1 != 0)
        while (read_blank() != 0)       /* Read following continuation lines.   */
        ;

    /*-------------------------------------------------------------------*/
    /*                        OUTPUT CONTENTS OF RECORD                  */
    /*-------------------------------------------------------------------*/

    if (conversion_log)         /* Print out log if requested.                  */
    {
        printf("Data Format for record %d:\n", datafmt.REC_TYPE);
        printf("-------------------------\n");
        printf("\tRecord name:\t\t%s\n",datafmt.REC_NAME);
        printf("\tField name:\t\t%s\n",datafmt.FIELD_NAME);
    }


}
```

05feat_class.c

```
/*************************************************************************/
/***                              feat_class()                         ***/
/***  Reads in Feature Classification Record (05) for processing       ***/
/***                                                                   ***/
/***  Jo Wood, Department of Geography, V2.0 27th May 1993             ***/
/***                                                                   ***/
/*************************************************************************/

#include "ntf_in.h"

feat_class()
{
    /*-------------------------------------------------------------------*/
    /*                           INITIALISE                              */
    /*-------------------------------------------------------------------*/

    struct  FEATCLASS   featclass;      /* Feature classification record struct.*/
    char                buffer[80],     /* Temporary storage of text data       */
                        *posn_ptr;      /* Position of terminator in record.    */

    /* Initialise all string structures */

    strcpy(featclass.CODE_COM,  "          ");
    strcpy(featclass.STCLASS,   "                    ");
    strcpy(featclass.FEATDES,   "                                        ");
```

```
    /*-------------------------------------------------------------------*/
    /*                        SCAN RECORD INTO STRUCTURE                 */
    /*-------------------------------------------------------------------*/

    featclass.REC_DESC_1        =       5;
    strncpy(buffer,             text+2,4);
        buffer[4] = NULL;
        featclass.FEAT_CODE =   atoi(buffer);
    strncpy(featclass.CODE_COM, text+6,10);
    strncpy(featclass.STCLASS,  text+16,20);

    if ( (posn_ptr = strrchr(text,'%')) == NULL)
        fprintf(stderr,
        "WARNING: Line %d - Feature Classification Record has no terminator.\n",
                                                        line_number);

                                    /* Ignore last \ in description record. */
    strncpy(featclass.FEATDES,  text+36,(posn_ptr-2) - (text+36));
    featclass.CONT_MARK_1 =     *(posn_ptr-1) - '0';
    featclass.EOR_1 =           *posn_ptr;


    /*-------------------------------------------------------------------*/
    /*                        CHECK INTEGRITY OF RECORD                  */
    /*-------------------------------------------------------------------*/

    if (featclass.CONT_MARK_1 != 0)
    {
        fprintf(stderr,
        "WARNING: Line %d - Feature Classification Record has continuation mark.\n",
                                                        line_number);
        while (read_blank() != 0)       /* Read following continuation lines.   */
        ;
    }

    /*-------------------------------------------------------------------*/
    /*                        OUTPUT CONTENTS OF RECORD                  */
    /*-------------------------------------------------------------------*/

    if (conversion_log)         /* Print out log if requested.                  */
    {
        printf("Feature Classification:\n");
        printf("----------------------\n");
        printf("\tCode:\t\t\t%d\n",featclass.FEAT_CODE);
        printf("\tDescription\t\t%s\n",featclass.FEATDES);
    }

                                /* Store feature description and code.          */
    if (outfile)
       G_set_cat(featclass.FEAT_CODE,featclass.FEATDES,&feature_desc);

}
```

07sect_head.c

```
/*****************************************************************************/
/***                              sect_head()                              ***/
/***  Reads in Section Header Record (07) into structure for processing    ***/
/***                                                                       ***/
/***  Jo Wood, Dept. of Geography, V2.0 28th May 1993                      ***/
/***                                                                       ***/
/*****************************************************************************/

#include "ntf_in.h"

sect_head()
{
    /*-------------------------------------------------------------------*/
    /*                            INITIALISE                             */
    /*-------------------------------------------------------------------*/

    struct  SECHREC     sechrec;        /* Section header structure.          */
    char                buffer[80];     /* Temporary storage of text data     */

    /* Initialise all string structures */

    strcpy(sechrec.SECT_REF,    "          ");
    strcpy(sechrec.SURV_DATE,   "        ");
    strcpy(sechrec.LAST_AMND,   "        ");
    strcpy(sechrec.COPYRIGHT,   "        ");

    /*-------------------------------------------------------------------*/
    /*                       SCAN RECORD INTO STRUCTURE                  */
    /*-------------------------------------------------------------------*/

    sechrec.REC_DESC_1  =       7;
    strncpy(sechrec.SECT_REF,   text+2,10);
    sechrec.COORD_TYPE =        text[12] - '0';
    sechrec.STRUC_TYPE =        text[13] - '0';
    strncpy(buffer,             text+14,5);
        buffer[5] = NULL;
        sechrec.XY_LEN =        atoi(buffer);
    sechrec.XY_UNIT =           text[19] - '0';
    strncpy(buffer,             text+20,10);
        buffer[10] = NULL;
        sechrec.XY_MULT =       atoi(buffer)/1000.0;
    strncpy(buffer,             text+30,5);
        buffer[5] = NULL;
        sechrec.Z_LEN =         atoi(buffer);
    sechrec.Z_UNIT =            text[35] - '0';
    strncpy(buffer,             text+36,10);
        buffer[10] = NULL;
        sechrec.Z_MULT =        atoi(buffer)/1000.0;
    strncpy(buffer,             text+46,10);
        buffer[10] = NULL;
        sechrec.X_ORIG =        atoi(buffer);
    strncpy(buffer,             text+56,10);
        buffer[10] = NULL;
```

```
        sechrec.Y_ORIG =        atoi(buffer);
    strncpy(buffer,             text+66,10);
        buffer[10] = NULL;
        sechrec.Z_DATUM =       atoi(buffer);
    sechrec.CONT_MARK_1 =       text[76] - '0';
    sechrec.EOR_1 =             text[77];

    if (sechrec.CONT_MARK_1 == 1)
    {
        if ( (sechrec.REC_DESC_2 = read_line()) != 0)
            fprintf(stderr,
"WARNING: Line %d - Continuatn of Section header has unexpected descriptor [%d] \n"
                                                ,line_number,sechrec.REC_DESC_2);
        strncpy(buffer,         text+2,10);
            buffer[10] = NULL;
            sechrec.XMIN =      atoi(buffer);
        strncpy(buffer,         text+12,10);
            buffer[10] = NULL;
            sechrec.YMIN =      atoi(buffer);
        strncpy(buffer,         text+22,10);
            buffer[10] = NULL;
            sechrec.XMAX =      atoi(buffer);
        strncpy(buffer,         text+32,10);
            buffer[10] = NULL;
            sechrec.YMAX =      atoi(buffer);
        strncpy(buffer,         text+42,5);
            buffer[5] = NULL;
            sechrec.XY_ACC =    atoi(buffer)/100.0;
        strncpy(buffer,         text+47,5);
            buffer[5] = NULL;
            sechrec.Z_ACC =     atoi(buffer)/100.0;
        strncpy(sechrec.SURV_DATE,text+52,8);
        strncpy(sechrec.LAST_AMND,text+60,8);
        strncpy(sechrec.COPYRIGHT,text+68,8);
        sechrec.CONT_MARK_2 =   text[76] - '0';
        sechrec.EOR_2 =         text[77];

        if (sechrec.CONT_MARK_2 == 1)
        {
            if ( (sechrec.REC_DESC_3 = read_line()) != 0)
                fprintf(stderr,
"WARNING: Line %d - Continuatn of Section header has unexpected descriptor [%d] \n"
                                                ,line_number,sechrec.REC_DESC_3);
            strncpy(sechrec.SQNAME,     text+2,20);
            strncpy(sechrec.SQDATE,     text+22,8);
            strncpy(buffer,             text+30,9);
                buffer[9] = NULL;
                sechrec.SCALE =         atoi(buffer);
            strncpy(buffer,             text+39,10);
                buffer[10] = NULL;
                sechrec.GRID_OR_X =     atoi(buffer)/1000.0;
            strncpy(buffer,             text+49,10);
                buffer[10] = NULL;
                sechrec.GRID_OR_Y =     atoi(buffer)/1000.0;
            strncpy(buffer,             text+59,8);
```

```
            buffer[8] = NULL;
            sechrec.PROJ_OR_LAT =   atoi(buffer)/10.0;
        strncpy(buffer,             text+67,8);
            buffer[8] = NULL;
            sechrec.PROJ_OR_LNG =   atoi(buffer)/10.0;
        sechrec.CONT_MARK_3 =       text[75] - '0';
        sechrec.EOR_3 =             text[76];
    }
    else
    {
        if (strncmp(H_mname,"OSCAR",5) == 0)
            sechrec.SCALE = 10000;
        else
            if (strncmp(H_mname,"OS_ROUTEPLANNER",15) == 0)
                sechrec.SCALE = 625000;
            else
                if (strncmp(H_mname,"OS_TRAVELMASTER",15) == 0)
                    sechrec.SCALE = 250000;
                else
                    sechrec.SCALE = 0;

        sechrec.CONT_MARK_3 = 0;
        sechrec.EOR_3 = '%';
    }
}
else
{
    sechrec.EOR_3 = '%';
    sechrec.CONT_MARK_3 = 0;
}


/*-------------------------------------------------------------------*/
/*                        CHECK INTEGRITY OF RECORD                  */
/*-------------------------------------------------------------------*/

if (sechrec.EOR_3 != '%')
    fprintf(stderr,
    "WARNING: Line %d - Section header record has no line terminator.\n",
                                                        line_number);

if (sechrec.CONT_MARK_3 != 0)
    while (read_blank() != 0)       /* Read following continuation lines.   */
    ;

/*-------------------------------------------------------------------*/
/*                        OUTPUT CONTENTS OF RECORD                  */
/*-------------------------------------------------------------------*/

if (conversion_log)                 /* Print out log if requested.          */
{
    printf("\n");
    printf("Section Header:\n");
    printf("---------------\n");
    printf("\tDatabase Name:\t\t%s\n",sechrec.SECT_REF);
```

```
        printf("\tMeasurement units:");
        if (sechrec.XY_UNIT == 2)
            printf("\tmetres.\n");
        else
            printf("\tunknown.\n");
        printf("\tOrigin (SW corner):\t(%10d,%10d)\n",
                                                sechrec.X_ORIG,sechrec.Y_ORIG);
        printf("\tHorizontal Accuracy:");
        if (sechrec.XY_ACC == 0.0)
            printf("\tnot recorded.\n");
        else
            printf("\t%.2f\n",sechrec.XY_ACC);
        printf("\tVertical Accuracy:");
        if (sechrec.Z_ACC == 0.0)
            printf("\tnot recorded.\n");
        else
            printf("\t%.2f\n",sechrec.Z_ACC);
    }

    if (outfile)
    {
        close_vect();                   /* Close any previously opened vectors. */

        X_origin = sechrec.X_ORIG;      /* Identify map boundaries.             */
        Y_origin = sechrec.Y_ORIG;

        X_min = sechrec.X_ORIG + (sechrec.XMIN*sechrec.XY_MULT);
        Y_min = sechrec.Y_ORIG + (sechrec.YMIN*sechrec.XY_MULT);
        X_max = sechrec.X_ORIG + ( (sechrec.XMAX - sechrec.XMIN)*sechrec.XY_MULT);
        Y_max = sechrec.Y_ORIG + ( (sechrec.YMAX - sechrec.YMIN)*sechrec.XY_MULT);

        XY_mult = sechrec.XY_MULT;
        Z_mult = sechrec.Z_MULT;

        strcat(H_mname," - ");
        strcat(H_mname,sechrec.SECT_REF);
        strcpy(H_mdate,sechrec.SURV_DATE);
        get_date(H_mdate);
        H_scale = sechrec.SCALE;
    }
}
```

11name_rec.c

```
/*****************************************************************************/
/***                          name_rec()                                   ***/
/***    Reads in the Name Record (11) into structure for processing        ***/
/***                                                                       ***/
/***  Jo Wood, Dept. of Geography, V2.1 16th June 1993.                    ***/
/***                                                                       ***/
/*****************************************************************************/

#include "ntf_in.h"
```

```
name_rec()
{
    /*-------------------------------------------------------------------*/
    /*                          INITIALISE                               */
    /*-------------------------------------------------------------------*/

    struct  NAMEREC     namerec;        /* Name record structure.              */
    char                buffer[80],     /* Temporary storage of text data      */
                        rec_desc,       /* Record descriptor.                  */
                        *posn_ptr;      /* Points to line terminator.          */

    /* Initialise all string structures */

    strcpy(namerec.TEXT,"
                                    ");


    /*-------------------------------------------------------------------*/
    /*                    SCAN RECORD INTO STRUCTURE                     */
    /*-------------------------------------------------------------------*/

    namerec.REC_DESC_1 =        11;

    /********* NOTE : INCOMPLETE **********/

    posn_ptr = strrchr(text,'%');
    namerec.CONT_MARK_1 =       *(posn_ptr-1) - '0';
    namerec.EOR_1 =             *(posn_ptr);

    /*-------------------------------------------------------------------*/
    /*                      CHECK INTEGRITY OF RECORD                    */
    /*-------------------------------------------------------------------*/

    if (namerec.CONT_MARK_1 != 0)
        while (read_blank() != 0)       /* Read following continuation lines.  */
        ;
}
```

12name_postn.c

```
/*************************************************************************/
/***                          name_postn()                             ***/
/***  Reads in Name Position Record (12) into structure for processing ***/
/***                                                                   ***/
/***  Jo Wood, Dept. of Geography, V2.1 16th June 1993                 ***/
/***                                                                   ***/
/*************************************************************************/

#include "ntf_in.h"

name_postn()
{
```

```
    /*-------------------------------------------------------------------*/
    /*                          INITIALISE                               */
    /*-------------------------------------------------------------------*/

    struct  NAMPOSTN    nampostn;       /* Name record structure.          */
    char                rec_desc,       /* Record descriptor.              */
                        *posn_ptr;      /* Position of terminator in record.  */


    /*-------------------------------------------------------------------*/
    /*                  SCAN RECORD INTO STRUCTURE                       */
    /*-------------------------------------------------------------------*/

    nampostn.REC_DESC_1 =       12;

    /********* NOTE : INCOMPLETE **********/

    if ( (posn_ptr = strrchr(text,'%')) == NULL)
        fprintf(stderr,
        "WARNING: Line %d - Name Position Record has no terminator.\n",
                                                            line_number);
    nampostn.CONT_MARK_1 =      *(posn_ptr-1) - '0';
    nampostn.EOR_1 =            *posn_ptr;


    /*-------------------------------------------------------------------*/
    /*                    CHECK INTEGRITY OF RECORD                      */
    /*-------------------------------------------------------------------*/

    if (nampostn.CONT_MARK_1 != 0)
    {
        fprintf(stderr,
"WARNING: Line %d - Name Position Record has unexpected continuation mark.\n",
                                                            line_number);
        while (read_blank() != 0)       /* Read following continuation lines.  */
        ;
    }

    /*-------------------------------------------------------------------*/
    /*                 READ GEOMETRY ASSOCIATED WITH TEXT                */
    /*-------------------------------------------------------------------*/

    if( (rec_desc = read_line()) != 21)
    {
        if (rec_desc == 99)
            vol_term();         /* Allow for merged volumes between records    */

        if (read_line() != 21)
        {
            fprintf(stderr,
"WARNING: Line %d - Name Position Record is not followed by geometry record.\n",
                                                            line_number);
            return(0);
        }
```

```
    }

    geom_rec(0,TEXT_LABEL);
}
```

14att_rec.c

```
/*************************************************************************/
/***                          att_rec()                                ***/
/***    Reads in Attribute Record (14) into structure for processing   ***/
/***                                                                   ***/
/***  Jo Wood, Dept. of Geography, V2.1 21st June 1993                 ***/
/***                                                                   ***/
/*************************************************************************/

#include "ntf_in.h"

att_rec()
{
    /*-----------------------------------------------------------------*/
    /*                        INITIALISE                               */
    /*-----------------------------------------------------------------*/

    struct  ATTREC      attrec;         /* Attribute record structure.          */

    char                buffer[80],     /* Temporary buffer for storing text.   */
                        *posn_ptr,      /* Points to line terminatr in string.  */
                        offset,loffset, /* Offset along text line.              */
                        label[80],      /* Category labels for vector object.   */
                        vect_name[80],
                        writecats = TRUE;

    int                 value;          /* Value of attribute (Height only)     */

    /* Initialise string structures */

    strcpy(attrec.VAL_TYPE,"  ");
    label[0] = NULL;
    strcpy(vect_name," ");


    /*-----------------------------------------------------------------*/
    /*                  SCAN RECORD INTO STRUCTURE                     */
    /*-----------------------------------------------------------------*/

    attrec.REC_DESC_1   =       14;
    strncpy(buffer,             text+2,6);
        buffer[6] = NULL;
        attrec.ATT_ID =         atoi(buffer);

    offset = 8;                         /* First value type in text string.     */
    loffset =0;
                                        /* Scan through value types.            */
```

```
do
{
    strncpy(attrec.VAL_TYPE,text+offset,2);

    if (strcmp(attrec.VAL_TYPE,"FC") == 0)  /* Feature Code */
    {
        strncpy(buffer,text+offset+2,4);
        buffer[4] = NULL;
        V_featcode = atoi(buffer);

        offset += 6;
    }


    else if (strcmp(attrec.VAL_TYPE,"HT") == 0)  /* Height  */
    {
        strncpy(buffer,text+offset+2,11);
        buffer[11] = NULL;
        value = atoi(buffer);

        if (Write_vect == TRUE)
        {
            write_vector(geom_type,value);
            Write_vect = FALSE;
        }
        sprintf(buffer,"%dm",value);
        G_set_cat(value,buffer,&cats);

        offset += 13;
        writecats = FALSE;

    }


    else if (strcmp(attrec.VAL_TYPE,"SC") == 0)   /* Scale  */
    {
        switch(*(text+offset+2))
        {
            case 'A':
                H_scale = 2500;
                break;

            case 'B':
                H_scale = 10000;
                break;

            case 'C':
                H_scale = 50000;
                break;

            case 'D':
                H_scale = 100000;
                break;

            default:
```

```
            fprintf(stderr,
                "WARNING: Line %d - Unknown digitization scale.\n",
                                                    line_number);
            break;
        }
        offset += 3;
    }

    else if (strcmp(attrec.VAL_TYPE,"SY") == 0)      /* Date (ignored) */
        offset +=8;

    else if (strcmp(attrec.VAL_TYPE,"LL") == 0)      /* Link Length (ignored)*/
        offset +=7;

    else if (strcmp(attrec.VAL_TYPE,"PN") == 0)      /* Proper Name          */
    {
        do
        {
            vect_name[loffset] = *(text + offset + loffset + 2);
            loffset++;
        }
        while(*(text + offset + loffset + 2) != '\\');

        vect_name[loffset] = ' ';
        loffset++;
        vect_name[loffset] = NULL;
        offset += loffset + 2;
    }

    else if (strcmp(attrec.VAL_TYPE,"RN") == 0)      /* Road Number.         */
    {
        strcat(vect_name," ");
        strncat(vect_name,text+offset+2,8);
        vect_name[loffset+9] = NULL;
        offset+=10;
    }

    else if (strcmp(attrec.VAL_TYPE,"FW") == 0)      /* Form of way (ignord) */
        offset += 3;

    else if (strcmp(attrec.VAL_TYPE,"RB") == 0)      /* Rep. point (bounded) */
        offset += 3;

    else if (strcmp(attrec.VAL_TYPE,"RU") == 0)      /* Rep. pnt (unbounded) */
        offset += 3;

    else if (strcmp(attrec.VAL_TYPE,"OR") == 0)      /* Orientation (ignord) */
        offset += 6;

    else if (strcmp(attrec.VAL_TYPE,"NU") == 0)      /* Numbered feature.    */
    {
        do
        {
            vect_name[loffset] = *(text + offset + loffset + 2);
            loffset++;
```

```
        }
        while(*(text + offset + loffset + 2) != '\\');

        vect_name[loffset] = ' ';
        loffset++;
        vect_name[loffset] = NULL;
        offset += loffset + 2;
    }

    else if (strcmp(attrec.VAL_TYPE,"TC") == 0)      /* Orientation (ignord) */
        offset += 6;

    else if (strcmp(attrec.VAL_TYPE,"TL") == 0)      /* Orientation (ignord) */
        offset += 4;

    else if (strcmp(attrec.VAL_TYPE,"TX") == 0)      /* Text string          */
    {
        loffset=0;
        do
        {
            vect_name[loffset] = *(text + offset + loffset + 2);
            loffset++;
        }
        while(*(text + offset + loffset + 2) != '\\');

        vect_name[loffset] = NULL;
        loffset++;
        offset += loffset + 2;
    }

    else
    {
fprintf(stderr, "WARNING: Line %d - Unrecognised attribute value type.\n",
                                                          line_number);
        offset +=1;
    }

    if ( (*(text+offset+1) == '%') && (*(text+offset) == '1'))
    {
        read_line();                /* If at end of line and it has a       */
        offset = 2;                 /* continuation code read next line.    */
    }
}
while( *(text+offset+1) != '%');

posn_ptr = strrchr(text,'%');       /* Search for line terminator.          */

attrec.CONT_MARK_1 =        *(posn_ptr-1) - '0';
attrec.EOR_1 =              *(posn_ptr);


/*-------------------------------------------------------------------*/
/*                      CHECK INTEGRITY OF RECORD                    */
/*-------------------------------------------------------------------*/
```

```
    if (attrec.EOR_1 != '%')
        fprintf(stderr,
        "WARNING: Line %d - Attribute record has no line terminator.\n",
                                                            line_number);

    if (attrec.CONT_MARK_1 != 0)
    {
        while (read_blank() != 0)       /* Read following continuation lines.   */
        ;
    }

    /*-------------------------------------------------------------------*/
    /*                      OUTPUT CONTENTS OF RECORD                    */
    /*-------------------------------------------------------------------*/

    if (writecats == TRUE)
    {
        strcpy(label," ");
        strcpy(label,vect_name);
        strcat(label,"- ");
        strcat(label,G_get_cat(V_featcode,&feature_desc));

        G_set_cat(attrec.ATT_ID,label,&cats);
    }
}
```

15point_rec.c

```
/*****************************************************************************/
/***                            point_rec()                                ***/
/***    Reads in the Point Record (15) into structure for processing       ***/
/***                                                                       ***/
/***  Jo Wood, Dept. of Geography, V2.1 15th June 1993                     ***/
/***                                                                       ***/
/*****************************************************************************/

#include "ntf_in.h"

point_rec()
{
    /*-------------------------------------------------------------------*/
    /*  IDENTIFY WHICH VARIATION OF POINT RECORD STRUCTURE IS REQUIRED    */
    /*-------------------------------------------------------------------*/

    if (text[21] == '%')                /* Contour Variation.                */
        point_rec_c();
    else
        if (text[23] == '%')            /* Default (OSCAR, 1:250k, 1:625k).  */
            point_rec_def();
        else
            if (text[36] == '%')        /* Land-Line Variation.              */
                point_rec_l();
            else
```

```
                fprintf(stderr,
                        "WARNING: Line %d - Can't identify Point Record Type.\n"
                                                                ,line_number);
}


/*************************************************************************/
/***                                                                   ***/
/*** Point record structure - Default format.                          ***/
/***                                                                   ***/
/*************************************************************************/

point_rec_def()
{
    /*-------------------------------------------------------------------*/
    /*                          INITIALISE                               */
    /*-------------------------------------------------------------------*/

    struct  POINTREC    pointrec;       /* Point record structure.          */
    char                buffer[80],     /* Temporary storage of text data   */
                        rec_desc,       /* Record descriptor.               */
                        maptype;        /* Type of map data to read.        */

    /*-------------------------------------------------------------------*/
    /*                    SCAN RECORD INTO STRUCTURE                     */
    /*-------------------------------------------------------------------*/

    pointrec.REC_DESC_1 =       15;

    strncpy(buffer,             text+2,6);
        buffer[6] = NULL;
        pointrec.POINT_ID =     atoi(buffer);
    strncpy(buffer,             text+8,6);
        buffer[6] = NULL;
        pointrec.GEOM_ID =      atoi(buffer);
    strncpy(buffer,             text+14,2);
        buffer[2] = NULL;
        pointrec.NUM_ATT =      atoi(buffer);
    strncpy(buffer,             text+16,6);
        buffer[6] = NULL;
        pointrec.ATT_ID =       atoi(buffer);

    pointrec.CONT_MARK_1 =      text[22] - '0';
    pointrec.EOR_1 =            text[23];

    /*-------------------------------------------------------------------*/
    /*                      CHECK INTEGRITY OF RECORD                    */
    /*-------------------------------------------------------------------*/

    if (pointrec.CONT_MARK_1 != 0)
    {
        fprintf(stderr,
        "WARNING: Line %d - Point record has unexpected continuation mark.\n",
                                                            line_number);
        while (read_blank() != 0)       /* Read following continuation lines.   */
```

```
        ;
    }

    /*-----------------------------------------------------------------*/
    /*                  READ GEOMETRY ASSOCIATED WITH POINT            */
    /*-----------------------------------------------------------------*/

    if( (rec_desc = read_line()) != 21)
    {
        if (rec_desc == 99)
            vol_term();                 /* Allow merged volumes between records */

        if (rec_desc == 14)
        {
                                        /* Ignore Attribute description record  */
                                        /* if it does not follow geometry rec.  */
            return(0);
        }


        if (read_line() != 21)
        {
            fprintf(stderr,
"WARNING: Line %d - Point record is not followed by geometry record.\n",
                                                            line_number);
            return(0);
        }

    }

    if (strncmp(H_mname,"OSCAR",5) == 0)
        maptype = OSCAR;
    else
        if (strncmp(H_mname,"OS_ROUTEPLANNER",15) == 0)
            maptype = ONE625;
        else
            if (strncmp(H_mname,"OS_TRAVELMASTER",15) == 0)
                maptype = ONE250;
            else
                if ((strncmp(H_mname,"Strategi",8) == 0) ||
                    (strncmp(H_mname,"STRATEGI",8) ==0))
                    maptype = STRATEGI;
                else
                    maptype = NO_ATTRIB;

    geom_rec(pointrec.GEOM_ID,maptype);
}


/*****************************************************************/
/***                                                           ***/
/*** Point record structure - Contour variation.               ***/
/***                                                           ***/
/*****************************************************************/
```

```
point_rec_c()
{
    /*-----------------------------------------------------------------*/
    /*                          INITIALISE                             */
    /*-----------------------------------------------------------------*/

    struct  POINTREC_C  pointrec_c;     /* Point record structure.          */
    char                buffer[80],     /* Temporary storage of text data   */
                        rec_desc;       /* Record descriptor.               */

    /* Initialise all string structures */

    strcpy(pointrec_c.VAL_TYPE, "  ");


    /*-----------------------------------------------------------------*/
    /*                    SCAN RECORD INTO STRUCTURE                   */
    /*-----------------------------------------------------------------*/

    pointrec_c.REC_DESC_1 =     15;
    strncpy(buffer,             text+2,6);
        buffer[6] = NULL;
        pointrec_c.POINT_ID =   atoi(buffer);
    strncpy(pointrec_c.VAL_TYPE,text+8,2);
    strncpy(buffer,             text+10,6);
        buffer[6] = NULL;
        pointrec_c.VALUE =      atoi(buffer);
    strncpy(buffer,             text+16,4);
        buffer[4] = NULL;
        pointrec_c.FEAT_CODE =  atoi(buffer);

    pointrec_c.CONT_MARK_1 =    text[20] - '0';
    pointrec_c.EOR_1 =          text[21];

    /*-----------------------------------------------------------------*/
    /*                      CHECK INTEGRITY OF RECORD                  */
    /*-----------------------------------------------------------------*/

    if (pointrec_c.CONT_MARK_1 != 0)
    {
        fprintf(stderr,
        "WARNING: Line %d - Point record has unexpected continuation mark.\n",
                                                          line_number);
        while (read_blank() != 0)       /* Read following continuation lines.   */
        ;
    }

    /*-----------------------------------------------------------------*/
    /*                  READ GEOMETRY ASSOCIATED WITH POINT            */
    /*-----------------------------------------------------------------*/

    if( (rec_desc = read_line()) != 21)
    {
        if (rec_desc == 99)
            vol_term();                 /* Allow merged volumes between records */
```

```
        if (read_line() != 21)
        {
            fprintf(stderr,
        "WARNING: Line %d - Point record is not followed by geometry record.\n",
                                                                line_number);
            return(0);
        }

    }

    V_featcode = pointrec_c.FEAT_CODE;

    geom_rec(pointrec_c.VALUE,CONTOUR);
}


/*********************************************************/
/***                                                   ***/
/*** Point record structure - Land-Line variation.     ***/
/***                                                   ***/
/*********************************************************/

point_rec_l()
{
    /*-------------------------------------------------------------------*/
    /*                          INITIALISE                               */
    /*-------------------------------------------------------------------*/

    struct  POINTREC_L  pointrec_l;     /* Point record structure.           */
    char                buffer[80],     /* Temporary storage of text data    */
                        rec_desc;       /* Record descriptor.                */

    /* Initialise all string structures */

    strcpy(pointrec_l.VAL_TYPE, "  ");


    /*-------------------------------------------------------------------*/
    /*                    SCAN RECORD INTO STRUCTURE                     */
    /*-------------------------------------------------------------------*/

    pointrec_l.REC_DESC_1 =     15;
    strncpy(buffer,             text+2,6);
        buffer[6] = NULL;
        pointrec_l.POINT_ID =   atoi(buffer);
    strncpy(pointrec_l.VAL_TYPE,text+8,2);
    strncpy(buffer,             text+10,6);
        buffer[6] = NULL;
        pointrec_l.VALUE =      atoi(buffer)/10.0;
    strncpy(buffer,             text+16,4);
        buffer[4] = NULL;
        pointrec_l.FEAT_CODE =  atoi(buffer);

    /******** INCOMPLETE **********/
```

```
    pointrec_1.CONT_MARK_1 =    text[35] - '0';
    pointrec_1.EOR_1 =          text[36];

    /*-------------------------------------------------------------------*/
    /*                      CHECK INTEGRITY OF RECORD                    */
    /*-------------------------------------------------------------------*/

    if (pointrec_1.CONT_MARK_1 != 0)
    {
        fprintf(stderr,
        "WARNING: Line %d - Point record has unexpected continuation mark.\n",
                                                            line_number);
        while (read_blank() != 0)       /* Read following continuation lines.   */
        ;
    }

    /*-------------------------------------------------------------------*/
    /*                  READ GEOMETRY ASSOCIATED WITH POINT              */
    /*-------------------------------------------------------------------*/

    if( ((rec_desc = read_line()) != 21) && (rec_desc != 22))
    {
        if (rec_desc == 99)
            vol_term();                 /* Allow merged volumes between records */

        if (((rec_desc=read_line() != 21)) && (rec_desc != 22))
        {
            fprintf(stderr,
        "WARNING: Line %d - Point record is not followed by geometry record.\n",
                                                            line_number);
            return(0);
        }

    }

    V_featcode = pointrec_1.FEAT_CODE;

    if (rec_desc == 21)
        geom_rec(pointrec_1.POINT_ID,DONT_KNOW); /* 2D Geometry Variation       */
                                                 /* Have chaged from FEAT_CODE  */
    else
        geom_rec2(pointrec_1.FEAT_CODE,LANDLINE); /* 3D Geometry Variation      */
}
```

16node_rec.c

```
/*************************************************************************/
/***                              node_rec()                           ***/
/***      Reads in the Node Record (16) into structure for processing   ***/
/***                                                                    ***/
/***  Jo Wood, Dept. of Geography, V2.1 21st June 1993                  ***/
/***                                                                    ***/
/*************************************************************************/
```

```
#include "ntf_in.h"

node_rec()
{
    /*-----------------------------------------------------------------*/
    /*                          INITIALISE                             */
    /*-----------------------------------------------------------------*/

    struct  NODEREC     noderec;        /* Node record structure.              */

    char                buffer[80],     /* Temporary buffer for storing text.  */
                        *posn_ptr,      /* Points to line terminator in string.*/
                        rec_desc;       /* Record descriptor.                  */


    /*-----------------------------------------------------------------*/
    /*                    SCAN RECORD INTO STRUCTURE                   */
    /*-----------------------------------------------------------------*/

    noderec.REC_DESC_1  =       16;
    strncpy(buffer,             text+2,6);
        buffer[6] = NULL;
        noderec.NODE_ID =       atoi(buffer);
    strncpy(buffer,             text+8,6);
        buffer[6] = NULL;
        noderec.GEOM_ID =       atoi(buffer);
    strncpy(buffer,             text+14,4);
        buffer[4] = NULL;
        noderec.NUM_LINKS =     atoi(buffer);

    /******** INCOMPLETE  ********/



    posn_ptr = strrchr(text,'%');       /* Search for line terminator.         */

    noderec.CONT_MARK_1 =       *(posn_ptr-1) - '0';
    noderec.EOR_1 =             *(posn_ptr);


    /*-----------------------------------------------------------------*/
    /*                      CHECK INTEGRITY OF RECORD                  */
    /*-----------------------------------------------------------------*/

    if (noderec.EOR_1 != '%')
        fprintf(stderr,
        "WARNING: Line %d - Node data record has no line terminator.\n",
                                                            line_number);

    if (noderec.CONT_MARK_1 != 0)
    {
        while (read_blank() != 0)       /* Read following continuation lines.  */
        ;
    }
```

```
    /*-------------------------------------------------------------------*/
    /*                  READ GEOMETRY ASSOCIATED WITH NODE               */
    /*-------------------------------------------------------------------*/

    if( (rec_desc = read_line()) != 21)
    {
        if (rec_desc == 99)
            vol_term();                 /* Allow merged volumes between records */

        if (read_line() != 21)
        {
            fprintf(stderr,
        "WARNING: Line %d - Node record is not followed by geometry record.\n",
                                                                    line_number);
            return(0);
        }

    }

    /******* Ignore the contents of node geometry for time being ******/

    /* ie, don't call geom_rec() */

    /* Should precede a geometry record */
}
```

21geom_rec.c

```
/*************************************************************************/
/***                          geom_rec()                                 ***/
/***  Reads in the Geometry Record (21) into structure for processing    ***/
/***                                                                     ***/
/***  Jo Wood, Dept of Geography, V2.0 30th May 1993                     ***/
/***                                                                     ***/
/*************************************************************************/

#include "ntf_in.h"

geom_rec(value,maptype)
    int value,                          /* Numerical attribute of vector to be
read.   */
        maptype;                        /* Type of map being converted.
        */
{

    /*-------------------------------------------------------------------*/
    /*                          INITIALISE                               */
    /*-------------------------------------------------------------------*/

    struct  GEOMREC     geomrec;        /* Geometry record structure.          */
    char                buffer[80],     /* Temporary storage of text data      */
                        *posn_ptr;      /* Points to line terminator in string. */
    int                 coords;         /* Counts through each coordinate
                                           pair in object                      */
```

```
    char                coord_digits=10;/* Number of digits representing coord. */


    /*-------------------------------------------------------------------*/
    /*       WRITE OUT PREVIOUS VECTOR (IF THERE IS ONE TO WRITE)        */
    /*-------------------------------------------------------------------*/

    if (outfile)
        if (Write_vect == TRUE)
        {
            write_vector(geom_type,value);
            Write_vect = FALSE;
        }

    /*-------------------------------------------------------------------*/
    /*     MAKE SURE THAT A POINT OR LINE RECORD HAS JUST BEEN READ      */
    /*-------------------------------------------------------------------*/

    if ( (value==NO_ATTRIB) && (maptype==NO_ATTRIB) )
    {
        fprintf(stderr,
"WARNING: Line %d - Geometry record has no preceding point or line record\n",
                                                              line_number);
        if ( (posn_ptr=strrchr(text,'%')) == NULL)
            fprintf(stderr,
"WARNING: Line %d - Geometry record has no line terminator\n",line_number);

        if (*(posn_ptr-1) != 0)
            while (read_blank() != 0)   /* Read following continuation lines.   */
            ;
        return(0);
    }


    /*-------------------------------------------------------------------*/
    /*                      IDENTIFY THE TYPE OF MAP TO CONVERT          */
    /*-------------------------------------------------------------------*/

    switch (maptype)
    {
        case CONTOUR:
            coord_digits = 10;
            break;

        case LANDLINE:
        case BOUNDARYLINE:
        case ONE625:
            coord_digits = 6;
            break;

        case ONE250:
        case STRATEGI:
            coord_digits = 5;
            break;
```

```
        case OSCAR:
            coord_digits = 4;
            break;

        case TEXT_LABEL:                /* Ignore for the moment */
            return(1);

        case DONT_KNOW:
            coord_digits = guess();
            break;

        default:
            fprintf(stderr,
        "WARNING: Line %d - Geometry record associated with unknown record type\n",
                                                                line_number);
            break;
    }


    /*-------------------------------------------------------------------*/
    /*                      SCAN RECORD INTO STRUCTURE                   */
    /*-------------------------------------------------------------------*/

    geomrec.REC_DESC_1 =        21;
    strncpy(buffer,             text+2,6);
        buffer[6] = NULL;
        geomrec.GEOM_ID =       atoi(buffer);
    geomrec.G_TYPE =            text[8] - '0';
    strncpy(buffer,             text+9,4);
        buffer[4] = NULL;
        geomrec.NUM_COORD =     atoi(buffer);


    /* Allocate enough memory to hold coordinates of segment.   */

    if ( dig_alloc_points(points,geomrec.NUM_COORD) < 0)
        fprintf(stderr,
"WARNING: Line %d - Not enough memory to hold entire vector segment.\n",
                                                            line_number);


    posn_ptr = text+13;                 /* Set position to first coord pair.    */

    for (coords=0; coords<geomrec.NUM_COORD; coords++)
    {
        strncpy(buffer,         posn_ptr,coord_digits);
            buffer[coord_digits] = NULL;
            geomrec.X_COORD =   atoi(buffer);

        if (*(posn_ptr+coord_digits+1) != '%')
                                    /* If not at the end of the record,     */
            posn_ptr += coord_digits;   /* move to the y coord in the record.   */
        else
        {
            if (read_line() == 0)       /* If at the end of a record, read      */
```

```
                posn_ptr = text+2;      /* next record and position ptr.        */
            else
                fprintf(stderr,
"WARNING: Line %d - Geometry record does not have expected continuatn descriptr.\n",
                                                                line_number);
        }

        strncpy(buffer,         posn_ptr,coord_digits);
            buffer[coord_digits] = NULL;
            geomrec.Y_COORD =   atoi(buffer);

        if (*(posn_ptr+coord_digits+1) != '%')
                                        /* If not at the end of the record,     */
            posn_ptr += coord_digits;   /* move to the QPlan element in record  */
        else
        {
            if (read_line() == 0)       /* If at the end of a record, read      */
                posn_ptr = text+2;      /* next record and position ptr.        */
            else
                fprintf(stderr,
"WARNING: Line %d - Geometry record does not have expected continuatn descriptr.\n",
                                                                line_number);
        }

        geomrec.Q_PLAN =        *(posn_ptr);

        if (outfile)
        {
            points->x[coords] = (double)(X_origin + XY_mult*geomrec.X_COORD);
            points->y[coords] = (double)(Y_origin + XY_mult*geomrec.Y_COORD);
            points->n_points = coords+1;
        }

        if (*(posn_ptr+2) != '%')       /* If not at the end of the record,     */
            posn_ptr += 1;              /* move to next coord pair in record.   */
        else
            if (*(posn_ptr+1) == '1')   /* If continuation, read next line and  */
            {                           /* set pointer to first coordinate pair.*/

                if (read_line() == 0)   /* If at the end of a record, read      */
                    posn_ptr = text+2;  /* next record and position ptr.        */
                else
                    fprintf(stderr,
"WARNING: Line %d - Geometry record does not have expected continuatn descriptr.\n",
                                                                line_number);
            }

    }
    geomrec.CONT_MARK_1 =       *(posn_ptr+1) - '0';
    geomrec.EOR_1 =             *(posn_ptr+2);


    /*-----------------------------------------------------------------*/
    /*                      CHECK INTEGRITY OF RECORD                  */
    /*-----------------------------------------------------------------*/
```

```
    if (geomrec.CONT_MARK_1 != 0)
    {
        fprintf(stderr,
        "WARNING: Line %d - Geometry record has unexpected continuation mark.\n",

line_number);
        while (read_blank() != 0)       /* Read following continuation lines.   */
        ;
    }

    /*-------------------------------------------------------------------*/
    /*                      WRITE OUT VECTOR COORDINATES                 */
    /*-------------------------------------------------------------------*/

    if (outfile)
    {
        if (geomrec.G_TYPE == 1)
            geom_type = DOT;
        else
            if (geomrec.G_TYPE == 2)
                geom_type = LINE;
            else
            {
                fprintf(stderr,
"WARNING: Line %d - Unknown geometry type for vector object (will assume line).\n",
                                                                    line_number);
                geom_type = LINE;
            }
        Write_vect = TRUE;
    }
}


guess()
{
        if (text[23] == '%')    /* One pair only */
            return(4);

        if (text[25] == '%')    /* One pair only */
            return(5);

        if (text[27] == '%')    /* One pair only */
            return(6);

        if (text[35] == '%')    /* One pair only */
            return(10);

        if (text[36] == '%')            /* Two pairs only */
            return(5);

        if (text[40] == '%')            /* Two pairs only */
            return(6);

        if (text[47] == '%')                    /* Three pairs only */
            return(5);
```

```
        if (text[53] == '%')                    /* Three pairs only */
            return(6);

        if (text[56] == '%')            /* Two pairs only */
            return(10);

        if (text[58] == '%')                            /* Four pairs only */
            return(5);

        if (text[66] == '%')                            /* Four pairs only */
            return(6);

        if (text[69] == '%')                                /* Five pairs only */
            return(5);

        if (text[77] == '%')                    /* Three pairs only */
            return(10);

        if (text[79] == '%')                                /* Five pairs only */
            return(6);


        fprintf(stderr,
"WARNING: Line %d - Can't guess number of digits in geometry record
                                                (will assume 4).\n",line_number);
        return(4);
}
```

22geom_rec.c

```
/*************************************************************************/
/***                              geom_rec()                           ***/
/***    Reads in three dimensional Geometry Record (22) into structure  ***/
/***                                                                   ***/
/***  Jo Wood, Dept of Geography, V2.2 29th July 1993                  ***/
/***                                                                   ***/
/*************************************************************************/

#include "ntf_in.h"

geom_rec2(value,maptype)
    int value,                  /* Numerical attribute of vector to be read.    */
        maptype;                /* Type of map being converted.                 */
{

    /*-----------------------------------------------------------------*/
    /*                         INITIALISE                              */
    /*-----------------------------------------------------------------*/

    struct  GEOMREC2    geomrec;        /* Geometry record structure.           */
    char                buffer[80],     /* Temporary storage of text data       */
                        *posn_ptr;      /* Points to line terminator in string. */
    int                 coords,         /* Counts through each coord in object. */
```

```
                    geom_type;      /* Point line or area vector object.    */
    char            coord_digits=10;/* Number of digits of coordinate.      */


    /*--------------------------------------------------------------------*/
    /*      MAKE SURE THAT A POINT OR LINE RECORD HAS JUST BEEN READ      */
    /*--------------------------------------------------------------------*/

    if ( (value==NO_ATTRIB) && (maptype==NO_ATTRIB) )
    {
        fprintf(stderr,
"WARNING: Line %d - 3D Geometry record has no preceding point or line record\n",
                                                                line_number);
        if ( (posn_ptr=strrchr(text,'%')) == NULL)
            fprintf(stderr,
    "WARNING: Line %d - 3D Geometry record has no line terminator\n",line_number);

        if (*(posn_ptr-1) != 0)
            while (read_blank() != 0)   /* Read following continuation lines.   */
            ;
        return(0);
    }

    /*--------------------------------------------------------------------*/
    /*                  IDENTIFY THE TYPE OF MAP TO CONVERT               */
    /*--------------------------------------------------------------------*/

    switch (maptype)
    {
        case LANDLINE:
            coord_digits = 6;
            break;

        default:
            fprintf(stderr,
"WARNING: Line %d - 3D Geometry record associated with unknown record type\n",
                                                                line_number);
            break;
    }


    /*--------------------------------------------------------------------*/
    /*                        SCAN RECORD INTO STRUCTURE                  */
    /*--------------------------------------------------------------------*/

    geomrec.REC_DESC_1 =        22;
    strncpy(buffer,             text+2,6);
        buffer[6] = NULL;
        geomrec.GEOM_ID =       atoi(buffer);
    geomrec.G_TYPE =            text[8] - '0';
    strncpy(buffer,             text+9,4);
        buffer[4] = NULL;
        geomrec.NUM_COORD =     atoi(buffer);
```

```
    /* Allocate enough memory to hold coordinates of segment.   */

    if ( dig_alloc_points(points,geomrec.NUM_COORD) < 0)
        fprintf(stderr,
"WARNING: Line %d - Not enough memory to hold entire vector segment.\n",
                                                            line_number);


    posn_ptr = text+13;                 /* Set position to first coord pair.    */

    for (coords=0; coords<geomrec.NUM_COORD; coords++)
    {
        strncpy(buffer,         posn_ptr,coord_digits);
            buffer[coord_digits] = NULL;
            geomrec.X_COORD =   atoi(buffer);

        if (*(posn_ptr+coord_digits+1) != '%')
                                    /* If not at the end of the record,     */
            posn_ptr += coord_digits;   /* move to the y coord in the record.   */
        else
        {
            if (read_line() == 0)       /* If at the end of a record, read      */
                posn_ptr = text+2;      /* next record and position ptr.        */
            else
                fprintf(stderr, "WARNING: Line %d -
        3D Geometry record does not have expected continuation descriptor.\n",
                                                            line_number);
        }

        strncpy(buffer,         posn_ptr,coord_digits);
            buffer[coord_digits] = NULL;
            geomrec.Y_COORD =   atoi(buffer);

        if (*(posn_ptr+coord_digits+1) != '%')
                                    /* If not at the end of the record,     */
            posn_ptr += coord_digits;   /* move to the QPlan element.           */
        else
        {
            if (read_line() == 0)       /* If at the end of a record, read      */
                posn_ptr = text+2;      /* next record and position ptr.        */
            else
                fprintf(stderr, "WARNING: Line %d -
        3D Geometry record does not have expected continuation descriptor.\n",
                                                            line_number);
        }

        geomrec.Q_PLAN =        *(posn_ptr);

        if (*(posn_ptr+2) != '%')       /* If we at the end of the record,      */
            posn_ptr += 1;              /* move to z coordinate in the record.  */
        else
            if (*(posn_ptr+1) == '1')   /* If continuation code, read next line */
            {                           /* set the pointer to first coord pair. */

                if (read_line() == 0)   /* If at the end of a record, read      */
```

```
            posn_ptr = text+2;  /* next record and position ptr.        */
        else
            fprintf(stderr,"WARNING: Line %d -
    3D Geometry record does not have expected continuation descriptor.\n",
                                                          line_number);
        }
        else
            break;

    strncpy(buffer,         posn_ptr,coord_digits);
        buffer[coord_digits] = NULL;
        geomrec.Z_COORD =   atoi(buffer);

    if (*(posn_ptr+coord_digits+1) != '%')
                                  /* If not at the end of the record,     */
        posn_ptr += coord_digits;  /* move to the QHT element in record    */
    else
    {
        if (read_line() == 0)      /* If at the end of a record, read      */
            posn_ptr = text+2;     /* next record and position ptr.        */
        else
            fprintf(stderr,"WARNING: Line %d -
    3D Geometry record does not have expected continuation descriptor.\n",
                                                          line_number);
    }

    geomrec.QHT =   *(posn_ptr);

    if (outfile)
    {
        points->x[coords] = (double)(X_origin + XY_mult*geomrec.X_COORD);
        points->y[coords] = (double)(Y_origin + XY_mult*geomrec.Y_COORD);
        points->n_points = coords+1;
    }

    if (*(posn_ptr+2) != '%')      /* If not at the end of the record,     */
        posn_ptr += 1;             /* move to next coord pair in record.   */
    else
        if (*(posn_ptr+1) == '1')  /* If continuation code, read next line */
        {                          /* set pointer to the first coord pair. */

            if (read_line() == 0)  /* If at the end of a record, read      */
                posn_ptr = text+2; /* next record and position ptr.        */
            else
                fprintf(stderr,"WARNING: Line %d -
    3D Geometry record does not have expected continuation descriptor.\n",
                                                          line_number);
        }

}
geomrec.CONT_MARK_1 =       *(posn_ptr+1) - '0';
geomrec.EOR_1 =             *(posn_ptr+2);
```

```
    /*-------------------------------------------------------------------*/
    /*                        CHECK INTEGRITY OF RECORD                  */
    /*-------------------------------------------------------------------*/

    if (geomrec.CONT_MARK_1 != 0)
    {
        fprintf(stderr,
"WARNING: Line %d - 3D  Geometry record has unexpected continuation mark.\n",
                                                              line_number);
        while (read_blank() != 0)       /* Read following continuation lines.   */
        ;
    }

    /*-------------------------------------------------------------------*/
    /*                       WRITE OUT VECTOR COORDINATES                */
    /*-------------------------------------------------------------------*/

    if (outfile)
    {
        if (geomrec.G_TYPE == 1)
            geom_type = DOT;
        else
            if (geomrec.G_TYPE == 2)
                geom_type = LINE;
            else
            {
                fprintf(stderr,
"WARNING: Line %d - Unknown geometry type for vector object (will assume line).\n",
                                                              line_number);
                geom_type = LINE;
            }
        write_vector(geom_type,value);

    }
}
```

23line_rec.c

```
/*************************************************************************/
/***                              line_rec()                           ***/
/***    Reads in the Line Record (23) into structure for processing    ***/
/***                                                                   ***/
/***  Jo Wood, Dept of Geography, V2.0 27th May 1993                   ***/
/***                                                                   ***/
/*************************************************************************/

#include "ntf_in.h"

line_rec()
{
    /*-------------------------------------------------------------------*/
    /*  IDENTIFY WHICH VARIATION OF LINE RECORD STRUCTURE IS REQUIRED     */
    /*-------------------------------------------------------------------*/

    if (text[21] == '%')                /* Contour Variation.                */
```

```
        line_rec_c();
    else
        if (text[23] == '%')            /* Default (OSCAR, 1:250k, 1:625k)      */
            line_rec_def();
        else
            if(text[36] == '%')         /* Land-Line Variation.                 */
                line_rec_l();
            else
                fprintf(stderr,
        "WARNING: Line %d - Can't identify Line Record Type.\n" ,line_number);
}


/*************************************************************************/
/***                                                                   ***/
/*** Line record structure - Default format.                           ***/
/***                                                                   ***/
/*************************************************************************/

line_rec_def()
{
    /*-------------------------------------------------------------------*/
    /*                          INITIALISE                               */
    /*-------------------------------------------------------------------*/

    struct  LINEREC     linerec;        /* Line record structure.               */
    char                buffer[80],     /* Temporary storage of text data       */
                        rec_desc,       /* Record descriptor.                   */
                        maptype;        /* Type of map data to read.            */


    /*-------------------------------------------------------------------*/
    /*                    SCAN RECORD INTO STRUCTURE                     */
    /*-------------------------------------------------------------------*/

    linerec.REC_DESC_1 =        23;

    strncpy(buffer,             text+2,6);
        buffer[6] = NULL;
        linerec.LINE_ID =       atoi(buffer);
    strncpy(buffer,             text+8,6);
        buffer[6] = NULL;
        linerec.GEOM_ID =       atoi(buffer);
    strncpy(buffer,             text+14,2);
        buffer[2] = NULL;
        linerec.NUM_ATT =       atoi(buffer);
    strncpy(buffer,             text+16,6);
        buffer[6] = NULL;
        linerec.ATT_ID =        atoi(buffer);

    linerec.CONT_MARK_1 =       text[22] - '0';
    linerec.EOR_1 =             text[23];
```

```
    /*-------------------------------------------------------------------*/
    /*                    CHECK INTEGRITY OF RECORD                      */
    /*-------------------------------------------------------------------*/

    if (linerec.CONT_MARK_1 != 0)
    {
        fprintf(stderr,
        "WARNING: Line %d - Line record has unexpected continuation mark.\n",
                                                              line_number);
        while (read_blank() != 0)       /* Read following continuation lines.   */
        ;
    }

    /*-------------------------------------------------------------------*/
    /*                 READ GEOMETRY ASSOCIATED WITH POINT               */
    /*-------------------------------------------------------------------*/

    if( (rec_desc = read_line()) != 21)
    {
        if (rec_desc == 99)
            vol_term();                 /* Allow merged volumes between records */

        if (read_line() != 21)
        {
            fprintf(stderr,
        "WARNING: Line %d - Line record is not followed by geometry record.\n",
                                                              line_number);
            return(0);
        }

    }

    if (strncmp(H_mname,"OSCAR",5) == 0)
        maptype = OSCAR;
    else
        if (strncmp(H_mname,"OS_ROUTEPLANNER",15) == 0)
            maptype = ONE625;
        else
            if (strncmp(H_mname,"OS_TRAVELMASTER",15) == 0)
                maptype = ONE250;
            else
                if ((strncmp(H_mname,"Strategi",8) == 0) ||
                    (strncmp(H_mname,"STRATEGI",8) ==0))
                    maptype = STRATEGI;
                else
                    maptype = NO_ATTRIB;

    geom_rec(linerec.GEOM_ID,maptype);
}

/*************************************************************************/
/***                                                                   ***/
/*** Line record structure - Contour variation.                        ***/
/***                                                                   ***/
/*************************************************************************/
```

```
line_rec_c()
{
    /*-----------------------------------------------------------------*/
    /*                          INITIALISE                             */
    /*-----------------------------------------------------------------*/

    struct  LINEREC_C   linerec_c;      /* Line record structure.          */
    char                buffer[80],     /* Temporary storage of text data. */
                        rec_desc;       /* Record descriptor.              */

    /* Initialise string structure */

    strcpy(linerec_c.VAL_TYPE,  "  ");


    /*-----------------------------------------------------------------*/
    /*                    SCAN RECORD INTO STRUCTURE                   */
    /*-----------------------------------------------------------------*/

    linerec_c.REC_DESC_1 =      23;
    strncpy(buffer,             text+2,6);
        buffer[6] = NULL;
        linerec_c.LINE_ID =     atoi(buffer);
    strncpy(linerec_c.VAL_TYPE,text+8,2);
    strncpy(buffer,             text+10,6);
        buffer[6] = NULL;
        linerec_c.VALUE =       atoi(buffer);
    strncpy(buffer,             text+16,4);
        buffer[4] = NULL;
        linerec_c.FEAT_CODE =   atoi(buffer);

    linerec_c.CONT_MARK_1 =     text[20] - '0';
    linerec_c.EOR_1 =           text[21];

    /*-----------------------------------------------------------------*/
    /*                     CHECK INTEGRITY OF RECORD                   */
    /*-----------------------------------------------------------------*/

    if (linerec_c.CONT_MARK_1 != 0)
    {
        fprintf(stderr,
        "WARNING: Line %d - Line record has unexpected continuation mark.\n",
                                                          line_number);
        while (read_blank() != 0)       /* Read following continuation lines.  */
        ;
    }

    /*-----------------------------------------------------------------*/
    /*                  READ GEOMETRY ASSOCIATED WITH LINE             */
    /*-----------------------------------------------------------------*/

    if( (rec_desc = read_line()) != 21)
    {
        if (rec_desc == 99)
            vol_term();                 /* Allow merged volumes between records */
```

```
        if (read_line() != 21)
        {
            fprintf(stderr,
        "WARNING: Line %d - Line record is not followed by geometry record.\n",
                                                                line_number);
            return(0);
        }

    }

    V_featcode = linerec_c.FEAT_CODE;

    geom_rec(linerec_c.VALUE,CONTOUR);
}


/*************************************************************************/
/***                                                                   ***/
/*** Line record structure - Land-Line variation.                      ***/
/***                                                                   ***/
/*************************************************************************/

line_rec_l()
{
    /*-------------------------------------------------------------------*/
    /*                          INITIALISE                               */
    /*-------------------------------------------------------------------*/

    struct  LINEREC_L   linerec_l;      /* Line record structure.            */
    char                buffer[80],     /* Temporary storage of text data.   */
                        rec_desc;       /* Record descriptor.                */

    /* Initialise string structure */

    strcpy(linerec_l.VAL_TYPE,  "  ");


    /*-------------------------------------------------------------------*/
    /*                    SCAN RECORD INTO STRUCTURE                     */
    /*-------------------------------------------------------------------*/

    linerec_l.REC_DESC_1 =      23;
    strncpy(buffer,             text+2,6);
        buffer[6] = NULL;
        linerec_l.LINE_ID =     atoi(buffer);
    strncpy(linerec_l.VAL_TYPE,text+8,2);
    strncpy(buffer,             text+10,6);
        buffer[6] = NULL;
        linerec_l.VALUE =       atoi(buffer);
    strncpy(buffer,             text+16,4);
        buffer[4] = NULL;
        linerec_l.FEAT_CODE =   atoi(buffer);


    /********** INCOMPLETE **********/
```

```
    linerec_1.CONT_MARK_1 =     text[35] - '0';
    linerec_1.EOR_1 =           text[36];

    /*-------------------------------------------------------------------*/
    /*                      CHECK INTEGRITY OF RECORD                    */
    /*-------------------------------------------------------------------*/

    if (linerec_1.CONT_MARK_1 != 0)
    {
        fprintf(stderr,
        "WARNING: Line %d - Line record has unexpected continuation mark.\n",
                                                              line_number);
        while (read_blank() != 0)       /* Read following continuation lines.   */
        ;
    }

    /*-------------------------------------------------------------------*/
    /*                   READ GEOMETRY ASSOCIATED WITH LINE              */
    /*-------------------------------------------------------------------*/

    if( ((rec_desc = read_line()) != 21) && (rec_desc != 22))
    {
        if (rec_desc == 99)
            vol_term();                 /* Allow merged volumes between records */

        if (((rec_desc=read_line()) != 21) && (rec_desc != 22))
        {
            fprintf(stderr,
        "WARNING: Line %d - Line record is not followed by geometry record.\n",
                                                              line_number);
            return(0);
        }

    }

    V_featcode = linerec_1.FEAT_CODE;

    if (rec_desc == 21)
        geom_rec(linerec_1.LINE_ID,DONT_KNOW);  /* 2D Geometry */
                                                /* Have Changd from FEAT_CODE */
    else
        geom_rec2(linerec_1.FEAT_CODE,LANDLINE);        /* 3D Geometry */
}
```

40att_desc.c

```
/*************************************************************************/
/***                                att_desc()                         ***/
/***  Reads in Attribute Description Record (40)for processing         ***/
/***                                                                    ***/
/***  Jo Wood, Dept of Geography, V2.0 28th May 1993                    ***/
/***                                                                    ***/
/*************************************************************************/
```

```
#include "ntf_in.h"

att_desc()
{
    /*-----------------------------------------------------------------*/
    /*                          INITIALISE                             */
    /*-----------------------------------------------------------------*/

    struct  ATTDESC     attdesc;        /* Attribute description record.        */
    char                buffer[80],     /* Temporary storage of text data.      */
                        *posn_ptr1,     /* Position of first divider in record. */
                        *posn_ptr2;     /* Position of divider and terminator.  */

    /* Initialise all string structures */

    strcpy(attdesc.VAL_TYPE,    "  ");
    strcpy(attdesc.FWIDTH,      "   ");
    strcpy(attdesc.FINTER,      "     ");
    strcpy(attdesc.ATT_NAME,    "                                                ");
    strcpy(attdesc.FDESC,       "                                                ");

    /*-----------------------------------------------------------------*/
    /*                    SCAN RECORD INTO STRUCTURE                   */
    /*-----------------------------------------------------------------*/

    attdesc.REC_DESC_1 =        5;
    strncpy(attdesc.FWIDTH,     text+2,2);
    strncpy(attdesc.FWIDTH,     text+4,3);
    strncpy(attdesc.FINTER,     text+7,5);

    if ( (posn_ptr1 = strchr(text,'\\')) == NULL)
        fprintf(stderr,
        "WARNING: Line %d - Attribute Description Record has no divider.\n",
                                                        line_number);
    strncpy(attdesc.ATT_NAME,   text+12,posn_ptr1 - (text+12));
    if (*(posn_ptr1+2) != '%')
    {
        if (( (posn_ptr2 = strrchr(text,'\\')) == NULL) ||
              (posn_ptr2 == posn_ptr1) )
            fprintf(stderr,
         "WARNING: Line %d - Attribute Description Record has no second divider.\n",
                                                        line_number);
        strncpy(attdesc.FDESC,  posn_ptr1+1,(posn_ptr2-1)-posn_ptr1);
    }

    if ( (posn_ptr2 = strrchr(text,'%')) == NULL)
        fprintf(stderr,
        "WARNING: Line %d - Attribute Description Record has no terminator.\n",
                                                        line_number);
    attdesc.CONT_MARK_1 =       *(posn_ptr2-1) - '0';
    attdesc.EOR_1 =             *posn_ptr2;
```

```
    /*-------------------------------------------------------------------*/
    /*                      CHECK INTEGRITY OF RECORD                    */
    /*-------------------------------------------------------------------*/

    if (attdesc.CONT_MARK_1 != 0)
    {
        fprintf(stderr,
        "WARNING: Line %d - Attribute Description Record has continuation mark.\n",
                                                              line_number);
        while (read_blank() != 0)       /* Read following continuation lines.   */
        ;
    }

    /*-------------------------------------------------------------------*/
    /*                      OUTPUT CONTENTS OF RECORD                    */
    /*-------------------------------------------------------------------*/

    if (conversion_log)          /* Print out log if requested.                */
    {
        printf("Attribute Description:\n");
        printf("----------------------\n");
        printf("\tName:\t\t\t%s\n",attdesc.ATT_NAME);
        printf("\tDescription:\t\t%s\n",attdesc.FDESC);
    }

}
```

43text_rec.c

```
/*************************************************************************/
/***                              text_rec()                           ***/
/***    Reads in the Text Record (43) into structure for processing    ***/
/***                                                                   ***/
/***  Jo Wood, Dept of Geography, V2.2 4th July 1993                   ***/
/***                                                                   ***/
/*************************************************************************/

#include "ntf_in.h"

text_rec()
{
    /*-------------------------------------------------------------------*/
    /*                           INITIALISE                              */
    /*-------------------------------------------------------------------*/

    struct  TEXTREC     textrec;        /* Text structure.                  */
    char                buffer[80],     /* Temporary storage of text data   */
                        *posn_ptr;      /* Points to line terminator.       */
    /* INCOMPLETE */


    /*-------------------------------------------------------------------*/
    /*                     SCAN RECORD INTO STRUCTURE                    */
    /*-------------------------------------------------------------------*/
```

```
    /* INCOMPLETE */


    posn_ptr =  strrchr(text,'%');

    textrec.EOR_1 = *posn_ptr;
    textrec.CONT_MARK_1 = *(posn_ptr - 1) - '0';

    /*-------------------------------------------------------------------*/
    /*                    CHECK INTEGRITY OF RECORD                      */
    /*-------------------------------------------------------------------*/

    if (textrec.EOR_1 != '%')
        fprintf(stderr,
        "WARNING: Line %d - Text record has no line terminator.\n", line_number);

    if (textrec.CONT_MARK_1 != 0)
    {
        fprintf(stderr,
        "WARNING: Line %d - Text record has second continuation mark.\n",
                                                          line_number);
        while (read_blank() != 0)       /* Read following continuation lines.   */
        ;
    }

 }
```

44text_postn.c

```
/*************************************************************************/
/***                          text_postn()                             ***/
/***  Reads in Text Position Record (44) into structure for processing  ***/
/***                                                                   ***/
/***  Jo Wood, Dept of Geography, V2.0 4th July 1993                   ***/
/***                                                                   ***/
/*************************************************************************/

#include "ntf_in.h"

text_postn()
{
    /*-------------------------------------------------------------------*/
    /*                          INITIALISE                               */
    /*-------------------------------------------------------------------*/

    struct  TEXTPOS     textpos;        /* Text position structure.          */
    char                buffer[80],     /* Temporary storage of text data    */
                        *posn_ptr;      /* Points to line terminator.        */
    /* INCOMPLETE */

    /*-------------------------------------------------------------------*/
    /*                   SCAN RECORD INTO STRUCTURE                      */
    /*-------------------------------------------------------------------*/
```

```
    /* INCOMPLETE */


    posn_ptr =  strrchr(text,'%');

    textpos.EOR_1 = *posn_ptr;
    textpos.CONT_MARK_1 = *(posn_ptr - 1) - '0';

    /*-------------------------------------------------------------------*/
    /*                       CHECK INTEGRITY OF RECORD                   */
    /*-------------------------------------------------------------------*/


    if (textpos.EOR_1 != '%')
        fprintf(stderr,
        "WARNING: Line %d - Text position record has no line terminator.\n",
                                                            line_number);


    if (textpos.CONT_MARK_1 != 0)
    {
        fprintf(stderr,
        "WARNING: Line %d - Text position record has second continuation mark.\n",
                                                            line_number);
        while (read_blank() != 0)       /* Read following continuation lines.   */
        ;
    }


 }
```

45text_rep.c

```
/*************************************************************************/
/***                          text_rep()                                 ***/
/*** Reads Text Representation Record (45) into structure for processing ***/
/***                                                                     ***/
/***  Jo Wood, Dept of Geography, V2.1 4th July, 1993                    ***/
/***                                                                     ***/
/*************************************************************************/

#include "ntf_in.h"

text_rep()
{
    /*-------------------------------------------------------------------*/
    /*                          INITIALISE                               */
    /*-------------------------------------------------------------------*/

    struct  TEXTREP     textrep;        /* Text representation structure.       */
    char                buffer[80],     /* Temporary storage of text data       */
                        *posn_ptr;      /* Points to line terminator.           */
```

```
    char                rec_desc;       /* Record descriptor.                 */

    int                 maptype;        /* Type of map to reconstruct.        */

    /* INCOMPLETE */


    /*--------------------------------------------------------------------*/
    /*                        SCAN RECORD INTO STRUCTURE                  */
    /*--------------------------------------------------------------------*/

    /* INCOMPLETE */


    posn_ptr =  strrchr(text,'%');

    textrep.EOR_1 = *posn_ptr;
    textrep.CONT_MARK_1 = *(posn_ptr - 1) - '0';


    /*--------------------------------------------------------------------*/
    /*                        CHECK INTEGRITY OF RECORD                   */
    /*--------------------------------------------------------------------*/

    if (textrep.EOR_1 != '%')
        fprintf(stderr,
        "WARNING: Line %d - Text representation record has no line terminator.\n",
                                                              line_number);

    if (textrep.CONT_MARK_1 != 0)
    {
        fprintf(stderr,
"WARNING: Line %d - Text representation record has second continuation mark.\n",
                                                              line_number);
        while (read_blank() != 0)       /* Read following continuation lines.  */
        ;
    }

    /*--------------------------------------------------------------------*/
    /*                  READ GEOMETRY ASSOCIATED WITH TEXT                */
    /*--------------------------------------------------------------------*/

    if( (rec_desc = read_line()) != 21)
    {
        if (rec_desc == 99)
            vol_term();                 /* Allow merged volumes between records */

        if (read_line() != 21)
        {
            fprintf(stderr, "WARNING: Line %d -
        Text Representation Record is not followed by geometry record.\n",
                                                                  line_number);
            return(0);
        }
    }
```

```
    if (strncmp(H_mname,"OSCAR",5) == 0)
        maptype = OSCAR;
    else
        if (strncmp(H_mname,"OS_ROUTEPLANNER",15) == 0)
            maptype = ONE625;
        else
            if (strncmp(H_mname,"OS_TRAVELMASTER",15) == 0)
                maptype = ONE250;
            else
                if ((strncmp(H_mname,"Strategi",8) == 0) ||
                    (strncmp(H_mname,"STRATEGI",8) ==0))
                    maptype = STRATEGI;
                else
                    maptype = NO_ATTRIB;

    geom_rec(0,maptype);

 }
```

50grid_head.c

```
/*****************************************************************************/
/***                          grid_head()                                  ***/
/***   Reads the Grid Header Record (50) into structure for processing     ***/
/***                                                                       ***/
/***  Jo Wood, Dept. of Geography, V2.0 28th May 1993                      ***/
/***                                                                       ***/
/*****************************************************************************/

#include "ntf_in.h"

grid_head()
{
    /*-----------------------------------------------------------------*/
    /*                          INITIALISE                             */
    /*-----------------------------------------------------------------*/

    struct  GRIDHREC    gridhrec;       /* Grid header structure.              */
    char                buffer[80];     /* Temporary storage of text data      */


    /*-----------------------------------------------------------------*/
    /*                    SCAN RECORD INTO STRUCTURE                   */
    /*-----------------------------------------------------------------*/

    gridhrec.REC_DESC_1 =       50;
    strncpy(buffer,             text+2,10);
        buffer[10] = NULL;
        gridhrec.GRID_ID =      atoi(buffer);
    strncpy(buffer,             text+12,4);
        buffer[4] = NULL;
        gridhrec.N_COLUMNS =    atoi(buffer);
    strncpy(buffer,             text+16,4);
```

```
        buffer[4] = NULL;
        gridhrec.N_ROWS =       atoi(buffer);
    strncpy(buffer,             text+20,4);
        buffer[4] = NULL;
        gridhrec.N_PLANES =     atoi(buffer);
    strncpy(buffer,             text+24,10);
        buffer[10] = NULL;
        gridhrec.X_COORD_1 =    atoi(buffer);
    strncpy(buffer,             text+34,10);
        buffer[10] = NULL;
        gridhrec.Y_COORD_1 =    atoi(buffer);
    strncpy(buffer,             text+44,6);
        buffer[6] = NULL;
        gridhrec.Z_COORD_1 =    atoi(buffer);
    strncpy(buffer,             text+50,10);
        buffer[10] = NULL;
        gridhrec.X_COORD_2 =    atoi(buffer);
    strncpy(buffer,             text+60,10);
        buffer[10] = NULL;
        gridhrec.Y_COORD_2 =    atoi(buffer);
    strncpy(buffer,             text+70,6);
        buffer[6] = NULL;
        gridhrec.Z_COORD_2 =    atoi(buffer);

    gridhrec.CONT_MARK_1 =      text[76] - '0';
    gridhrec.EOR_1 =            text[77];

    if (gridhrec.CONT_MARK_1 == 1)
    {
        if ( (gridhrec.REC_DESC_2 = read_line()) != 0)
            fprintf(stderr,
"WARNING: Line %d - Continuation of Grid header has unexpected descriptor [%d] \n",
                                                gridhrec.REC_DESC_2,line_number);
        strncpy(buffer,         text+2,10);
            buffer[10] = NULL;
            gridhrec.X_COORD_3 =atoi(buffer);
        strncpy(buffer,         text+12,10);
            buffer[10] = NULL;
            gridhrec.Y_COORD_3 =atoi(buffer);
        strncpy(buffer,         text+22,6);
            buffer[6] = NULL;
            gridhrec.Z_COORD_3 =atoi(buffer);
        strncpy(buffer,         text+28,10);
            buffer[10] = NULL;
            gridhrec.X_COORD_4 =atoi(buffer);
        strncpy(buffer,         text+38,10);
            buffer[10] = NULL;
            gridhrec.Y_COORD_4 =atoi(buffer);
        strncpy(buffer,         text+48,6);
            buffer[6] = NULL;
            gridhrec.Z_COORD_4 =atoi(buffer);

        gridhrec.CONT_MARK_2 =  text[54] - '0';
        gridhrec.EOR_2 =        text[55];
```

```
        if (gridhrec.CONT_MARK_2 == 1)
        {
            if ( (gridhrec.REC_DESC_3 = read_line()) != 0)
                fprintf(stderr,
"WARNING: Line %d - Continuation of Grid header has unexpected descriptor [%d] \n",
                                                gridhrec.REC_DESC_3,line_number);
            strncpy(buffer,             text+2,10);
                buffer[10] = NULL;
                gridhrec.X_COORD_5 =atoi(buffer);
            strncpy(buffer,             text+12,10);
                buffer[10] = NULL;
                gridhrec.Y_COORD_5 =atoi(buffer);
            strncpy(buffer,             text+22,6);
                buffer[6] = NULL;
                gridhrec.Z_COORD_5 =atoi(buffer);
            strncpy(buffer,             text+28,10);
                buffer[10] = NULL;
                gridhrec.X_COORD_6 =atoi(buffer);
            strncpy(buffer,             text+38,10);
                buffer[10] = NULL;
                gridhrec.Y_COORD_6 =atoi(buffer);
            strncpy(buffer,             text+48,6);
                buffer[6] = NULL;
                gridhrec.Z_COORD_6 =atoi(buffer);

            gridhrec.CONT_MARK_3 =      text[54] - '0';
            gridhrec.EOR_3 =            text[55];

            if (gridhrec.CONT_MARK_3 == 1)
            {
                if ( (gridhrec.REC_DESC_4 = read_line()) != 0)
                    fprintf(stderr,
"WARNING: Line %d - Continuation of Grid header has unexpected descriptor [%d] \n",
                                                gridhrec.REC_DESC_4,line_number);
                strncpy(buffer,         text+2,10);
                    buffer[10] = NULL;
                    gridhrec.X_COORD_7 =atoi(buffer);
                strncpy(buffer,         text+12,10);
                    buffer[10] = NULL;
                    gridhrec.Y_COORD_7 =atoi(buffer);
                strncpy(buffer,         text+22,6);
                    buffer[6] = NULL;
                    gridhrec.Z_COORD_7 =atoi(buffer);
                strncpy(buffer,         text+28,10);
                    buffer[10] = NULL;
                    gridhrec.X_COORD_8 =atoi(buffer);
                strncpy(buffer,         text+38,10);
                    buffer[10] = NULL;
                    gridhrec.Y_COORD_8 =atoi(buffer);
                strncpy(buffer,         text+48,6);
                    buffer[6] = NULL;
                    gridhrec.Z_COORD_8 =atoi(buffer);

                gridhrec.CONT_MARK_4 =  text[54] - '0';
                gridhrec.EOR_4 =        text[55];
```

```
            }
        }
    }
    else
        fprintf(stderr,
        "WARNING: Line %d - Expected Grid header continuation mark is missing.\n",
                                                                line_number);


    /*---------------------------------------------------------------*/
    /*                      CHECK INTEGRITY OF RECORD                */
    /*---------------------------------------------------------------*/

    if (gridhrec.CONT_MARK_4 != 0)
    {
        fprintf(stderr,
        "WARNING: Line %d - Grid header record has fourth continuation mark.\n",
                                                                line_number);
        while (read_blank() != 0)       /* Read following continuation lines.   */
        ;
    }

    if (gridhrec.EOR_4 != '%')
        fprintf(stderr,
        "WARNING: Line %d - Grid header record has no line terminator.\n",
                                                                line_number);

    if ( (gridhrec.N_ROWS != 401) || (gridhrec.N_COLUMNS != 401) )
        fprintf(stderr,
"WARNING: Line %d - Number of rows and cols not equal to 401: (r,c) = (%d,%d).\n",
                              line_number,gridhrec.N_ROWS,gridhrec.N_COLUMNS);


    /*---------------------------------------------------------------*/
    /*            OUTPUT CONTENTS OF RECORD TO CONVERSION LOG         */
    /*---------------------------------------------------------------*/

    if (conversion_log)         /* Print out log if requested.                */
    {
        printf("Grid Header:\n");
        printf("----------------\n");
        printf("\tMap Square:\t%10d\n",gridhrec.GRID_ID);
        printf("\tNo of rows:\t%d\n",gridhrec.N_ROWS);
        printf("\tNo of columns:\t%d\n",gridhrec.N_COLUMNS);
        printf("\tCorners:\t(%10d,%10d)\n",gridhrec.X_COORD_1,gridhrec.Y_COORD_1);
        printf("\t\t\t(%10d,%10d)\n",gridhrec.X_COORD_2,gridhrec.Y_COORD_2);
        printf("\t\t\t(%10d,%10d)\n",gridhrec.X_COORD_3,gridhrec.Y_COORD_3);
        printf("\t\t\t(%10d,%10d)\n",gridhrec.X_COORD_4,gridhrec.Y_COORD_4);
        printf("\n");
    }


    /*---------------------------------------------------------------*/
    /*               WRITE OUT NECESSARY INFORMATION TO GRASS        */
    /*---------------------------------------------------------------*/
```

```
    /* Grid Header Record [50] must mean we need to open a raster */

    if (outfile)
    {
        if (O_raster)
            fprintf(stderr,
    "WARNING: Line %d - More than one Grid Header Record found.\n",line_number);
        else
            open_raster(gridhrec.X_COORD_1,gridhrec.Y_COORD_1,
                                                gridhrec.N_ROWS,gridhrec.N_COLUMNS);
    }

}
```

51grid_rec.c

```
/*****************************************************************************/
/***                              grid_rec()                               ***/
/***     Reads the Grid Data Record (51) into structure for processing     ***/
/***                                                                       ***/
/***  Jo Wood, Dept of Geography, V2.0 28th May 1993                       ***/
/***                                                                       ***/
/*****************************************************************************/

#include "ntf_in.h"

grid_rec()
{
    /*-----------------------------------------------------------------*/
    /*                          INITIALISE                             */
    /*-----------------------------------------------------------------*/

    struct  GRIDREC     gridrec;        /* Grid data record structure.          */
    char                buffer[80],     /* Temporary storage of text data       */
                        gridvals[1604]; /* Holds a line of the raster.          */
    int                 num_lines,      /* Counts through body of 21 grid recs. */
                        col;

    /*-----------------------------------------------------------------*/
    /*                     SCAN RECORD INTO STRUCTURE                  */
    /*-----------------------------------------------------------------*/

    gridrec.REC_DESC_1  =       50;
    strncpy(buffer,             text+2,10);
        buffer[10] = NULL;
        gridrec.GRID_ID =       atoi(buffer);
    strncpy(gridrec.SURVEY,     text+12,7);
    strncpy(gridrec.CHANGE,     text+19,7);
    strncpy(buffer,             text+26,4);
        buffer[4] = NULL;
        gridrec.COL_START =     atoi(buffer);
    strncpy(buffer,             text+30,4);
        buffer[4] = NULL;
```

```
        gridrec.COL_END =       atoi(buffer);
    strncpy(buffer,             text+34,4);
        buffer[4] = NULL;
        gridrec.ROW_START =     atoi(buffer);
    strncpy(buffer,             text+38,4);
        buffer[10] = NULL;
        gridrec.ROW_END =       atoi(buffer);
    strncpy(buffer,             text+42,4);
        buffer[10] = NULL;
        gridrec.PLA_START =     atoi(buffer);
    strncpy(buffer,             text+46,4);
        buffer[6] = NULL;
        gridrec.PLA_END =       atoi(buffer);
    gridrec.COL_INV =           text[50] - '0';
    gridrec.ROW_INV =           text[51] - '0';
    gridrec.PLA_INV =           text[52] - '0';
    gridrec.ORDER =             text[53] - '0';
    gridrec.INTERPRET =         text[54] - '0';
    strncpy(buffer,             text+55,10);
        buffer[10] = NULL;
        gridrec.V_OFFSET =      atoi(buffer);
    strncpy(buffer,             text+65,10);
        buffer[10] = NULL;
        gridrec.V_SCALE =       atoi(buffer) / 1000.0;
    gridrec.CONT_MARK_1 =       text[75] - '0';
    gridrec.EOR_1 =             text[76];

    if (gridrec.CONT_MARK_1 == 1)
    {
        if ( (gridrec.REC_DESC_2 = read_line()) != 0)
            fprintf(stderr,
"WARNING: Line %d - Continuation of Grid data has unexpected descriptor [%d] \n",
                                                gridrec.REC_DESC_2,line_number);
        strncpy(buffer,         text+2,8);
            buffer[8] = NULL;
            gridrec.N_GRIDVAL =atoi(buffer);
        gridrec.CONT_MARK_2 =   text[10] - '0';
        gridrec.EOR_2 = text[11];
    }
    else
    {
        fprintf(stderr,
        "WARNING: Line %d - Expected Grid header continuation mark is missing.\n",
                                                                line_number);
        gridrec.CONT_MARK_2 =   1;
    }

    if (gridrec.CONT_MARK_2 == 1)
    {
        for (num_lines=0; num_lines<21; num_lines++)
        {
            if ( (gridrec.REC_DESC_3 = read_line()) != 0)
                fprintf(stderr,
"WARNING: Line %d - Continuation of Grid data has unexpected descriptor [%d] \n",
                                                gridrec.REC_DESC_3,line_number);
```

```
            strncpy(gridrec.GRIDVAL_1,  text+2,76);
            gridrec.CONT_MARK_3 =       text[10] - '0';
            gridrec.EOR_3 =             text[11];

            /* Add text string to gridvals */
            if (outfile)
                strncpy(gridvals + num_lines*76,gridrec.GRIDVAL_1,76);
        }
        if ( (gridrec.REC_DESC_4 = read_line()) != 0)
            fprintf(stderr,
"WARNING: Line %d - Continuation of Grid data has unexpected descriptor [%d] \n",
                                                gridrec.REC_DESC_4,line_number);
        strncpy(gridrec.GRIDVAL_2,      text+2,8);
        gridrec.CONT_MARK_4 =   text[10] - '0';
        gridrec.EOR_4 =         text[11];

        /* Write out contents to raster array */
        if (outfile)
            strncpy(gridvals + 1596,gridrec.GRIDVAL_2,8);
    }
    else
        fprintf(stderr,
        "WARNING: Line %d - Expected Grid header continuation mark is missing.\n",
                                                            line_number);

    /*-----------------------------------------------------------------*/
    /*                       CHECK INTEGRITY OF RECORD                 */
    /*-----------------------------------------------------------------*/

    if (gridrec.CONT_MARK_4 != 0)
    {
        fprintf(stderr,
        "WARNING: Line %d - Grid data record has too many continuation marks.\n",
                                                            line_number);
        while (read_blank() != 0)       /* Read following continuation lines.   */
        ;
    }

    if (gridrec.EOR_4 != '%')
        fprintf(stderr,
        "WARNING: Line %d - Grid data record has no line terminator.\n",
                                                            line_number);


    /*-----------------------------------------------------------------*/
    /*                       OUTPUT CONTENTS OF RECORD                 */
    /*-----------------------------------------------------------------*/

    if (conversion_log)         /* Print out log if requested.                 */
    {
        printf("Grid Data:\tFirst Row:\tLast Row\tFirst Col\tLast Col\t\n");
        printf("          \t  %d\t\t  %d\t\t  %d\t\t  %d\n",
                                    gridrec.ROW_START,gridrec.ROW_END,
                                    gridrec.COL_START,gridrec.COL_END);
    }
```

```
    if (outfile)
    {
        col = gridrec.COL_START;
        printf("%d ",col);
        fill_raster(gridvals,col);

        num_rlines++;
        if (num_rlines == 401)  /* When entire raster is read, output to GRASS  */
            write_raster();
    }

}



/*************************************************************************/
/***                          fill_raster()                            ***/
/***  Extracts the raster values from a set of 22 Grid Data Records      ***/
/***                                                                   ***/
/*************************************************************************/


fill_raster(gridvals,col)
    char        gridvals[1604];         /* Text string containing raster values.*/
    int col;
{
    int         scany;                  /* Scans a line through raster.         */

    char        gridval_txt[5];         /* Stores a single grid value as text.  */
    strncpy (gridval_txt,"    ");


    for (scany=401; scany>=1; scany--)
    {
        strncpy(gridval_txt,gridvals,4);
        raster[scany-1][col-1] = (CELL)atoi(gridval_txt);

        gridvals += 4;                  /* Move to next number.                 */

    }
}
```

99vol_term.c

```
/*************************************************************************/
/***                          vol_term()                               ***/
/*** Reads Volume Termination Record (99) into structure for processing  ***/
/***                                                                   ***/
/*** Jo Wood, Dept. of Geography, V1.0 27th May 1993                     ***/
/***                                                                   ***/
/*************************************************************************/

#include "ntf_in.h"
```

```
vol_term()
{
    /*-------------------------------------------------------------------*/
    /*                          INITIALISE                               */
    /*-------------------------------------------------------------------*/

    struct  VOLTERM     volterm;        /* Feature classificatn record struct.  */
    char                buffer[80],     /* Temporary storage of text data       */
                        *posn_ptr;      /* Points to record termination mark.   */

    strcpy(volterm.FREE_TEXT,   "                                                ");

    /*-------------------------------------------------------------------*/
    /*                     SCAN RECORD INTO STRUCTURE                    */
    /*-------------------------------------------------------------------*/

    volterm.REC_DESC_1 =        99;

    if ( (posn_ptr = strrchr(text,'%')) == NULL)
        fprintf(stderr,
        "WARNING: Line %d - Volume Termination Record has no terminator.\n",
                                                          line_number);

    strncpy(volterm.FREE_TEXT,  text+2, (posn_ptr-1) - (text+2) );
    volterm.CONT_MARK_1 =       *(posn_ptr-1) - '0';
    volterm.EOR_1 =             *posn_ptr;


    /*-------------------------------------------------------------------------*/
    /*                       CHECK INTEGRITY OF RECORD                         */
    /*-------------------------------------------------------------------------*/

    if (volterm.CONT_MARK_1 != 0)
        if (read_line() != 1)   /* Read following volume header record.        */
            fprintf(stderr,
            "WARNING: Line %d - Volume Header Record is missing.\n",line_number);


    /*-------------------------------------------------------------------------*/
    /*                      OUTPUT CONTENTS OF RECORD                          */
    /*-------------------------------------------------------------------------*/

    if (conversion_log)         /* Print out log if requested.                 */
    {
       printf("\n");
       printf("*************************************************************\n");
       printf("\t%s\n",volterm.FREE_TEXT);
       printf("*************************************************************\n");
    }

    return (volterm.CONT_MARK_1);
}
```

r.to.sites

GRASS module that converts the non-zero values of a raster file into a GRASS sites (point vector) file.

Jo Wood, Department of Geography, 2nd August 1993

rtosites.h

```
/*****************************************************************/
/***                                                           ***/
/***                rtosites.h - For use with r.to.sites       ***/
/***                  V1.0  - Jo Wood, 2nd August 1993         ***/
/***                                                           ***/
/*****************************************************************/

#include "gis.h"                /* This MUST be included in all GRASS   */
                                /* programs. It sets up the necessary   */
                                /* prototypes for GRASS library calls.  */


/* ------ Some shorthand defines for region information. ------ */

#define North (region.north)    /* When outputting any vector file it   */
#define South (region.south)    /* is advisable to fill in the vector   */
#define East (region.east)      /* header file with its geographical    */
#define West (region.west)      /* extent. This is stored in the region */
#define NS_Res (region.ns_res)  /* structure. For convenience, these    */
#define EW_Res (region.ew_res)  /* are referred to by their shorthand   */
#define Zone (region.zone)      /* define names.                        */


/* ------ Global variables ------ */

#ifndef MAIN
    extern                      /* Externally defined if not main()     */
#endif

char            *rast_in_name,  /* Name of the raster file to convert.  */

                *sites_out_name, /* Name of the output sites file.      */

                *mapset_in,     /* Names of mapsets containing files.   */
                *mapset_out;

#ifndef MAIN
    extern
#endif

int             fd_in;          /* File descriptor for raster file      */
```

```
#ifndef MAIN
    extern
#endif

FILE            *sites_fptr;    /* File descriptor for output sites file */
```

main.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                            r.to.sites                                 ***/
/***    GRASS module to convert non-zero raster values into a sites file   ***/
/***                      Jo Wood, Ordnance Survey                         ***/
/***                        V 0.9 2nd August 1993                          ***/
/***                                                                       ***/
/*****************************************************************************/

#define MAIN

#include "rtosites.h"

main(argc,argv)
    int argc;
    char *argv[];       /* Assuming GRASS is going to respond to some   */
                        /* input from the keyboard, the two arguments   */
                        /* are necessary. argc is an ARGument Count     */
                        /* that stores the number of words input. *argv */
                        /* is a POINTER to an array holding those words.*/
{
    /*---------------------------------------------------------------------*/
    /*                          GET INPUT FROM USER                        */
    /*---------------------------------------------------------------------*/

    interface(argc,argv);

    /*---------------------------------------------------------------------*/
    /*                       OPEN INPUT AND OUTPUT FILES                   */
    /*---------------------------------------------------------------------*/

    open_files();

    /*---------------------------------------------------------------------*/
    /*                         CONVERT RASTER TO SITES                     */
    /*---------------------------------------------------------------------*/

    convert();

    /*---------------------------------------------------------------------*/
    /*                           CLOSE DOWN FILES                          */
    /*---------------------------------------------------------------------*/

    G_unopen_cell(fd_in);
    fclose(sites_fptr);
}
```

interface.c

```
/*****************************************************************************/
/***      Function to get input from user and check files can be opened    ***/
/***                                                                       ***/
/***                    Jo Wood, V1.0 2nd August 1993                      ***/
/*****************************************************************************/

#include "rtosites.h"

interface(argc,argv)

    int      argc;              /* Number of command line arguments     */
    char    *argv[];            /* Contents of command line arguments.  */

{
    /*---------------------------------------------------------------------*/
    /*                              INITIALISE                             */
    /*---------------------------------------------------------------------*/


    struct Option       *rast_in,       /* Pointer to  structures holding  */
                        *sites_out;     /* the text to describe each option*/
                                        /* It also stores the user's input */

    G_gisinit (argv[0]);                /* GRASS library function called   */
                                        /* checks for valid command        */
                                        /* argv[0] is the program name.    */


    /*---------------------------------------------------------------------*/
    /*                            SET PARSER OPTIONS                       */
    /*---------------------------------------------------------------------*/

    rast_in    = G_define_option();     /* Pointer to memory for each option*/
    sites_out  = G_define_option();

    /* Each option needs a 'key' (short description),
         a 'description` (a longer one),
         a 'type' (eg intiger, or string),
         and an indication whether manditory or not */

    rast_in->key          = "rast";
    rast_in->description  = "Raster map layer to convert";
    rast_in->type         = TYPE_STRING;
    rast_in->required     = YES;

    sites_out->key         = "sites";
    sites_out->description = "Output sites file to produce";
    sites_out->type        = TYPE_STRING;
    sites_out->required    = YES;

    if (G_parser(argc,argv))            /* Performs the prompting for   */
        exit(-1);                       /* keyboard input.              */
                                        /* Returns 0 if sucessful.      */
```

```
    rast_in_name    = rast_in->answer;
    sites_out_name  = sites_out->answer;


    /*--------------------------------------------------------------------*/
    /*                    CHECK INPUT RASTER FILES EXIST                  */
    /*--------------------------------------------------------------------*/

    if ((mapset_in=G_find_cell2(rast_in_name,""))==NULL)
    {
        char err[256];
        sprintf(err,"Raster map layer [%s] not available.",rast_in_name);
        G_fatal_error(err);
    }

    /*--------------------------------------------------------------------*/
    /*                  CHECK OUTPUT SITES FILE DOES NOT EXIST            */
    /*--------------------------------------------------------------------*/

    mapset_out = G_mapset();            /* Set output to current mapset.  */

    if (G_legal_filename(sites_out_name)==NULL)
    {
        char err[256];
        sprintf(err,"Illegal sites file name. Please try another.");
        G_fatal_error(err);
    }
    else
    {
        if (G_find_file("sites_lists",sites_out_name,mapset_out) !=NULL)
        {
            char err[256];
            sprintf(err,"Sites file [%s] already exists.\nPlease try another\n",
                                                                sites_out_name);
            G_fatal_error(err);
        }
    }

}
```

open_files.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                            open_files()                               ***/
/***        Opens input raster and output sites files for r.to.sites       ***/
/***                Jo Wood, Dept of Geography 2nd August 1993             ***/
/***                                                                       ***/
/*****************************************************************************/

#include "rtosites.h"

open_files()
{
```

```
    /* Open existing raster file and set the input file descriptor */

    if ( (fd_in=G_open_cell_old(rast_in_name,mapset_in)) <0)
    {
        char err[256];
        sprintf(err,"ERROR: Problem opening input raster file.");
        G_fatal_error(err);
    }


    /* Open new sites file and set the output file descriptor. */

    if ( (sites_fptr=G_fopen_sites_new(sites_out_name)) == NULL)
    {
        char err[256];
        sprintf(err,"ERROR: Problem opening output sites file.");
        G_fatal_error(err);
    }

}
```

convert.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                            convert()                                  ***/
/***            Reads in a raster file and output sites file               ***/
/***                    Jo Wood V1.0 Department of Geography               ***/
/***                                                                       ***/
/*****************************************************************************/

#include "rtosites.h"

convert()
{

    /*-------------------------------------------------------------------*/
    /*                            INITIALISE                             */
    /*-------------------------------------------------------------------*/

    CELL        *row_buf;       /* Buffer to hold raster row */

    struct      Cell_head  region;

    int         nrows,          /* Will store the current number of     */
                ncols,          /* rows and columns in the raster.      */

                row,col;        /* Counts through each row and column   */
                                /* of the input raster.                 */

    double      easting,        /* Coordinates of centre of each        */
                northing;       /* Raster cell.                         */

    char        desc[80];       /* Description of each locatn (eg elev) */
```

```
    /*-------------------------------------------------------------------*/
    /*                   GET DETAILS OF INPUT RASTER                     */
    /*-------------------------------------------------------------------*/

    row_buf = G_allocate_cell_buf();     /* Initialise raster row buffer */

    nrows = G_window_rows();             /* Find out the number of rows  */
    ncols = G_window_cols();             /* and columns of the raster.   */

    G_get_window(&region);


    /*-------------------------------------------------------------------*/
    /*          PROCESS INPUT RASTER AND WRITE OUT SITES FILE            */
    /*-------------------------------------------------------------------*/

    for(row=0; row<nrows; row++)
    {
        G_get_map_row(fd_in,row_buf, row);

        for(col = 0; col < ncols; col++)
        {
            easting = West + (col*EW_Res) + (EW_Res/2.0);
            northing = North - (row*NS_Res) - (NS_Res/2.0);

            sprintf(desc,"# %d",(int)(*(row_buf+col)));

            if (*(row_buf+col) != (CELL)0)
                G_put_site(sites_fptr,easting,northing,desc);
        }
    }
}
```

A.4 DEM Analysis Modules

d.param.scale

GRASS module that allows interactive indentification of terrain parameters and features with a mouse. Reports membership classes for variable window sizes and their variance or entropy. Based on a modified version of r.param.scale.

Jo Wood, 4th October, 1995

param.h

```
/*****************************************************************************/
/***                                                                       ***/
/***                            param.h                                    ***/
/***  Header file for use with r.param.scale                               ***/
/***  Jo Wood, ASSIST, Dept of Geography, University of Leicester          ***/
/***  V1.0  - 7th February, 1993                                           ***/
/***                                                                       ***/
/*****************************************************************************/

#include "gis.h"                /* This MUST be included in all GRASS   */
                                /* programs. It sets up the necessary   */
                                /* prototypes for GRASS library calls.  */
#include <math.h>

#define EDGE ((wsize-1)/2)      /* Number of rows/cols that make up the */
                                /* 'blank' edge around raster.          */
#define MAX_WSIZE 69            /* Maximum dimensions of window.        */

                                /* Rounds any number to its nearest     */
                                /* integer. (rint is SGI specific, so   */
                                /* for other platforms, you  may need   */
                                /* to change this macro.                */
#define RINT(x) (CELL)rint((double)(x))

                                /* Some useful labels.                  */
#define TRUE 1
#define FALSE 0

#define RAD2DEG 57.29578
#define DEG2RAD 0.017453293

#define TINY 1.0e-20;

#define FLAT ((CELL)0)
#define PIT ((CELL)1)
#define CHANNEL ((CELL)2)
#define PASS ((CELL)3)
#define RIDGE ((CELL)4)
#define PEAK ((CELL)5)
```

```
#define NUM_CATS ((CELL)6)
#define EMAX 1.7917595          /* Maximum entropy for 6 categories */

#define ELEV   1
#define SLOPE  2
#define ASPECT 3
#define PROFC  4
#define PLANC  5
#define LONGC  6
#define CROSC  7
#define MINIC  8
#define MAXIC  9
#define FEATURE 10

/* The six quadratic coefficients are stored in the array coeff */

#define C_A coeff[1]
#define C_B coeff[2]
#define C_C coeff[3]
#define C_D coeff[4]
#define C_E coeff[5]
#define C_F coeff[6]

/* ------ Declare functions ----- */

float   *vector(),              /* Reserves memory for 1D matrix.       */
        **matrix();             /* Reserves memory for 2D matrix.       */

int     *ivector();             /* Reserves memory for 1D int matrix.   */

CELL    param();                /* Calculates terrain parameters.       */


double  D_d_to_a_row(),         /* GRASS functions that should have     */
        D_d_to_a_col();         /* been prototyped in gis.h but were    */
                                /* not, so I have here.                 */


/* ------ Global variables ------ */

#ifndef MAIN
    extern                      /* Externally defined if not main()     */
#endif

char            *rast_in_name,  /* Name of the raster file to process.  */

                *mapset_in,     /* Mapset containing DEM.               */

                constrained,    /* Flag that forces quadtratic through  */
                                /* the central cell of the window.      */

                dem_frame[128], /* Names of graphics frames.            */
                graph_frame[128];

#ifndef MAIN
```

```
    extern                      /* Externally defined if not main()     */
#endif

int             fd_in,          /* File descriptor for input file.      */

                wsize,          /* Size of local processing window.     */

                mparam;         /* Morphometric parameter to calculate. */


#ifndef MAIN
    extern                      /* Externally defined if not main()     */
#endif

double          resoln,         /* Planimetric resolution.              */
                exponent,       /* Distance weighting exponent.         */
                zscale,         /* Vertical scaling factor.             */

                slope_tol,      /* Vertical tolerences for surface      */
                curve_tol;      /* feature identification.              */
```

main.c

```
/*****************************************************************************/
/**                                                                         **/
/**                            r.param.scale                                **/
/**       GRASS module for extracting multi-scale surface parameters.       **/
/**                                                                         **/
/**                                                                         **/
/**                  Jo Wood, V 1.1, 11th December, 1994                    **/
/**                                                                         **/
/*****************************************************************************/

#define MAIN

#include "param.h"

main(argc,argv)
    int argc;
    char *argv[];
{


    /*-------------------------------------------------------------------*/
    /*                        GET INPUT FROM USER                        */
    /*-------------------------------------------------------------------*/

    interface(argc,argv);


    /*-------------------------------------------------------------------*/
    /*                  OPEN INPUT AND OUTPUT RASTER FILES                */
    /*-------------------------------------------------------------------*/
```

```
    open_files();
    init_graphics();


    /*--------------------------------------------------------------------*/
    /*                PROCESS SURFACE FOR FEATURE DETECTION               */
    /*--------------------------------------------------------------------*/

    process();

    /*--------------------------------------------------------------------*/
    /*                CLOSE ALL OPENED FILES AND FREE MEMORY              */
    /*--------------------------------------------------------------------*/

    close_down();

}
```

interface.c

```
/*****************************************************************************/
/***                             interface()                               ***/
/*** Function to get input from user and check files can be opened           ***/
/***                                                                       ***/
/***   Jo Wood, Department of Geography, V1.2, 7th February 1992           ***/
/*****************************************************************************/

#include "param.h"

interface(argc,argv)

    int      argc;              /* Number of command line arguments.    */
    char    *argv[];            /* Contents of command line arguments.  */

{
    /*--------------------------------------------------------------------*/
    /*                         INITIALISE                                 */
    /*--------------------------------------------------------------------*/

    struct Option  *rast_in,    /* Name of input file from command line.*/
                   *tol1_val,   /* Tolerance values for feature         */
                   *tol2_val,   /* detection (slope and curvature).     */
                   *win_size,   /* Max size of side of local window.    */
                   *parameter,  /* Morphometric parameter to calculate. */
                   *expon,      /* Inverse distance exponent for weight.*/
                   *vert_sc;    /* Vertical scaling factor.             */

    struct Flag    *constr;     /* Forces quadratic through the central */
                                /* cell of local window if selected.    */

    G_gisinit (argv[0]);        /* GRASS function which MUST be called  */
                                /* first to check for valid database    */
                                /* and mapset and prompt user for input.*/
```

```
/*-------------------------------------------------------------------*/
/*                      SET PARSER OPTIONS                           */
/*-------------------------------------------------------------------*/

rast_in   = G_define_option();      /* Request memory for each option.*/
tol1_val  = G_define_option();
tol2_val  = G_define_option();
win_size  = G_define_option();
parameter = G_define_option();
expon     = G_define_option();
vert_sc   = G_define_option();

constr    = G_define_flag();

rast_in->key          = "in";
rast_in->description  = "Raster surface layer to interrogate";
rast_in->type         = TYPE_STRING;
rast_in->required     = YES;

tol1_val->key         = "s_tol";
tol1_val->description = "Slope tolerance that defines a `flat'
                         surface (degrees)";
tol1_val->type        = TYPE_DOUBLE;
tol1_val->required    = NO;
tol1_val->answer      = "1.0";

tol2_val->key         = "c_tol";
tol2_val->description = "Curvature tolerance that defines `planar'
                         surface";
tol2_val->type        = TYPE_DOUBLE;
tol2_val->required    = NO;
tol2_val->answer      = "1.0";

win_size->key         = "size";
win_size->description = "Size of processing window (odd number only)";
win_size->type        = TYPE_INTEGER;
win_size->required    = NO;
win_size->answer      = "15";

parameter->key        = "param";
parameter->description= "Morphometric parameter to calculate";
parameter->type       = TYPE_STRING;
parameter->required   = NO;
parameter->options    = "elev,slope,aspect,profc,planc,longc,
                         crosc,minic,maxic,feature";
parameter->answer     = "elev";

expon->key            = "exp";
expon->description    = "Exponent for distance weighting (0.0-4.0)";
expon->type           = TYPE_DOUBLE;
expon->required       = NO;
expon->answer         = "0.0";

vert_sc->key          = "zscale";
vert_sc->description  = "Vertical scaling factor";
```

```
vert_sc->type          = TYPE_DOUBLE;
vert_sc->required      = NO;
vert_sc->answer        = "1.0";

constr->key            = 'c';
constr->description    = "Constrain model through central window cell";


if (G_parser(argc,argv))     /* Actually performs the prompting for  */
    exit(-1);                /* keyboard input.                      */

rast_in_name  = rast_in->answer;    /* Store results globally.       */
wsize         = atoi(win_size->answer);
constrained   = constr->answer;
sscanf(expon->answer,"%lf",&exponent);
sscanf(vert_sc->answer,"%lf",&zscale);
sscanf(tol1_val->answer,"%lf",&slope_tol);
sscanf(tol2_val->answer,"%lf",&curve_tol);

if ((exponent<0.0) || (exponent >4.0))
    exponent = 0.0;

if (zscale == 0.0)
    zscale = 1;

if (!strcmp(parameter->answer,"elev"))
        mparam = ELEV;
else
    if (!strcmp(parameter->answer,"slope"))
        mparam = SLOPE;
    else
        if (!strcmp(parameter->answer,"aspect"))
            mparam = ASPECT;
        else
            if (!strcmp(parameter->answer,"profc"))
                mparam = PROFC;
            else
                if (!strcmp(parameter->answer,"planc"))
                    mparam = PLANC;
                else
                    if (!strcmp(parameter->answer,"crosc"))
                        mparam = CROSC;
                    else
                        if (!strcmp(parameter->answer,"longc"))
                            mparam = LONGC;
                        else
                            if (!strcmp(parameter->answer,"maxic"))
                                mparam = MAXIC;
                            else
                                if (!strcmp(parameter->answer,"minic"))
                                 mparam = MINIC;
                                else
                                 if (!strcmp(parameter->answer,"feature"))
                                        mparam = FEATURE;
                                 else
```

```
                                    {
                                        G_warning(
        "Morphometric parameter not recognised. Assuming `Elevation'");
                                        mparam  = ELEV;
                                    }

    /*--------------------------------------------------------------------*/
    /*                    CHECK INPUT RASTER FILE EXISTS                  */
    /*--------------------------------------------------------------------*/


    if ((mapset_in=G_find_cell2(rast_in_name,""))==NULL)
    {
        char err[256];
        sprintf(err,"Raster map [%s] not available.",rast_in_name);
        G_fatal_error(err);
    }

    /*--------------------------------------------------------------------*/
    /*                CHECK WINDOW SIZE IS NOT EVEN OR TOO LARGE          */
    /*--------------------------------------------------------------------*/

    if ( (wsize/2 != (wsize-1)/2) || (wsize > MAX_WSIZE) )
    {
        char err[256];
        sprintf(err,"Inappropriate window size (too big or even)");
        G_fatal_error(err);
    }
}
```

open_files.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                           open_files()                                ***/
/***                Opens input and output raster files.                   ***/
/***               Jo Wood, Project ASSIST, 24th January 1993              ***/
/***                                                                       ***/
/*****************************************************************************/

#include "param.h"


open_files()
{
    /* Open existing file and set the input file descriptor. */

    if ( (fd_in=G_open_cell_old(rast_in_name,mapset_in)) <0)
    {
        char err[256];
        sprintf(err,"ERROR: Problem opening input file.");
        G_fatal_error(err);
    }
}
```

init_graphics.c

```
/*****************************************************************/
/***                                                           ***/
/***                      init_graphics()                      ***/
/***  Initialises all graphics calles for interactive mouse query.***/
/***          Jo Wood,Dept. of Geography, 4th December 1995    ***/
/***                                                           ***/
/*****************************************************************/

#include "param.h"

init_graphics()
{

    /*---------------------------------------------------------------*/
    /*                          INITIALISE                           */
    /*---------------------------------------------------------------*/

    int   bot,top,lef,rit;              /* Window boundaries.        */

    *dem_frame   = 0;                   /* Initialise frame names.   */
    *graph_frame = 0;


    /*---------------------------------------------------------------*/
    /*        OPEN GRAPHICS DRIVER AND CHECK WINDOW COORDINATES       */
    /*---------------------------------------------------------------*/

    system("d.frame -e; d.erase white");
    R_open_driver();

    D_get_screen_window(&top,&bot,&lef,&rit);   /* Get window coords.  */

    D_new_window(graph_frame,bot/2 + 4, bot-1, lef+4, rit-4);


    D_new_window(dem_frame, top+1, bot/2, (rit-lef)/4, 3*(rit-lef)/4);
    D_set_cur_wind(dem_frame);
    Dcell(rast_in_name,mapset_in,0);
}
```

display_graph.c

```
/*****************************************************************/
/***                                                           ***/
/***                       disp_graph()                        ***/
/***            Displays graph of parameter values with scale. ***/
/***             Jo Wood,Dept. of Geography, 4th December 1995 ***/
/***                                                           ***/
/*****************************************************************/
```

```
#include "param.h"


disp_graph(scr_x,scr_y,param_ptr)
    int scr_x,scr_y,                    /* Mouse screen coords.          */
    *param_ptr;                         /* Array storing paramtr values */
{

    /*-------------------------------------------------------------------*/
    /*                          INITIALISE                               */
    /*-------------------------------------------------------------------*/

    int   bot,top,lef,rit,              /* Window boundaries.            */
          x,y,                          /* Plotting coorinates.          */
          n,                            /* Number of samples.            */

          yaxis,yinc,                   /* Y axis label counter and inc.*/

          size,                         /* Local window sizes.           */

          param,                        /* Parameter value.              */
          pmin=99999,
          pmax=-99999;                  /* Range of parameter values.    */

    char  text[128];                    /* Text buffer for labels.       */

    double mean   = 0.0,                /* Distribution measures.        */
           stdev  = 0.0,
           entrop = 0.0,
           fp[6];                       /* Feature probabilities.        */

    for (size=0; size<6; size++)        /* Initialise feature probs.     */
        fp[size] = 0.0;

    n = (wsize-1)/2;                    /* Number of samples.            */


    /*-------------------------------------------------------------------*/
    /*                  HIGHLIGHT CURRENT SELECTION                      */
    /*-------------------------------------------------------------------*/

    R_standard_color(D_translate_color("red"));
    plot_cross(scr_x,scr_y);


    /*-------------------------------------------------------------------*/
    /*                       SETUP GRAPH WINDOW                          */
    /*-------------------------------------------------------------------*/

    D_set_cur_wind(graph_frame);
    D_get_screen_window(&top,&bot,&lef,&rit);
    D_set_clip_window(top,bot,lef,rit);

    top +=2;                            /* Allow small outer border.     */
    bot -=2;
```

```
lef +=2;
rit -=2;

R_standard_color(D_translate_color("white"));
D_erase_window();


/*-------------------------------------------------------------------*/
/*                          LABEL GRAPH                              */
/*-------------------------------------------------------------------*/

R_standard_color(D_translate_color("black"));
R_font("romanc");
R_text_size((rit-lef)/25,(bot-top)/25);
R_move_abs(lef,top + (bot-top)/25);

switch(mparam)
{
    case(ELEV):
        sprintf(text,"Elevation");
        break;
    case(SLOPE):
        sprintf(text,"Slope");
        break;
    case(ASPECT):
        sprintf(text,"Aspect");
        break;
    case(PROFC):
        sprintf(text,"Profile Curvature");
        break;
    case(PLANC):
        sprintf(text,"Plan Curvature");
        break;
    case(LONGC):
        sprintf(text,"Longitudinal Curviture");
        break;
    case(CROSC):
        sprintf(text,"Cross-sectional Curvature");
        break;
    case(MINIC):
        sprintf(text,"Minimum Curvature");
        break;
    case(MAXIC):
        sprintf(text,"Maximum Curvature");
        break;
    case(FEATURE):
        sprintf(text,"Feature Classification");
        break;
    default:
        fprintf(stderr,"Error: Unknown terrain parameter.\n");
        exit();
}
R_text(text);

lef += (rit-lef)/10;
```

```
    top += (bot-top)/10;

    D_move_abs(lef,bot);
    D_cont_abs(rit,bot);
    D_move_abs(lef,bot);
    D_cont_abs(lef,top);

    R_text_size((rit-lef)/40,(bot-top)/25);


    /*---------------------------------------------------------------------*/
    /*                          LABEL AXES                                 */
    /*---------------------------------------------------------------------*/

    if (mparam==FEATURE)
    {
        pmin=0;
        pmax=NUM_CATS-1;

        R_move_abs(lef-(rit-lef)/10,bot - FLAT*(bot-top)/6);
        R_text("Planr");

        R_move_abs(lef-(rit-lef)/10,bot - PIT*(bot-top)/6);
        R_text("Pit");

        R_move_abs(lef-(rit-lef)/10,bot - CHANNEL*(bot-top)/6);
        R_text("Chanl");

        R_move_abs(lef-(rit-lef)/10,bot - PASS*(bot-top)/6);
        R_text("Pass");

        R_move_abs(lef-(rit-lef)/10,bot - RIDGE*(bot-top)/6);
        R_text("Ridge");

        R_move_abs(lef-(rit-lef)/10,bot - PEAK*(bot-top)/6);
        R_text("Peak");
    }
    else
    {                   /* Get parameter range and graph scaling.      */
        for (size=3; size<=wsize; size += 2)
        {
            param = *(param_ptr + (size-3)/2);

            if (param < pmin)
                pmin = param;

            if (param > pmax)
                pmax = param;
        }

        yinc = (int)ceil((pmax-pmin)/10.0);

        for (yaxis = pmin; yaxis <=pmax; yaxis++)
            if (yaxis%yinc == 0)
            {
```

```
            R_move_abs(lef-(rit-lef)/10,bot - (yaxis-pmin)*(bot-top)/
                                                    (1+pmax-pmin));
            sprintf(text,"%d",yaxis);
            R_text(text);
            plot_cross(lef,bot- (yaxis-pmin)*(bot-top)/(1 + pmax-pmin));
        }
    }

/*-------------------------------------------------------------------*/
/*          PLOT GRAPH VALUES AND CALCULATE PRIMARY STATS            */
/*-------------------------------------------------------------------*/


param = *(param_ptr + (size-3)/2);
D_move_abs(lef,bot - (param-pmin)*(bot-top)/(1+ pmax-pmin));

for (size=3; size<=wsize; size += 2)
{
    param = *(param_ptr + (size-3)/2);
    x = lef + (size-3)*(rit-lef)/(wsize-1);
    y = bot - (param-pmin)*(bot-top)/(1+pmax-pmin);

    if (mparam==FEATURE)
    {
        plot_box(x,y,top,bot,lef,rit,param);
        fp[param]++;
    }
    else
    {
        D_cont_abs(x,y);
        plot_cross(x,y);
        mean += param;
    }
}

/*-------------------------------------------------------------------*/
/*          CALULATE AND DISPLAY SECONDARY STATISTICS                */
/*-------------------------------------------------------------------*/

if (mparam == FEATURE)
{
    for (size=0; size<6; size++)    /* Rescale feature probabilities*/
    {
        fp[size] /= n;
        if (fp[size] != 0.0)
            entrop += -fp[size]*log(fp[size]);
    }
    printf("Scaled Entropy = %.3lf\n", entrop/EMAX);
}
else
{
    mean /= n;

    for (size=3; size<=wsize; size += 2)
    {
```

```
            param = *(param_ptr + (size-3)/2);
            stdev += (mean-param)*(mean-param);
        }
        stdev = sqrt(stdev/n);

        printf("Mean = %.3lf\n Stdev = %.3lf\n",mean,stdev);
    }


    /*--------------------------------------------------------------------*/
    /*                CLOSE DOWN AND RESET TO DEM WINDOW                  */
    /*--------------------------------------------------------------------*/


    R_flush();
    D_set_cur_wind(dem_frame);
    D_set_clip_window_to_map_window();

}

/*********************************************************************/
/***                      Cross Plotting function.                 ***/
/*********************************************************************/


plot_cross(scr_x,scr_y)
    int scr_x,scr_y;            /* Centre of cross to plot.     */
{
    D_move_abs(scr_x-2,scr_y-2);
    D_cont_rel(5,5);
    D_move_abs(scr_x-2,scr_y+2);
    D_cont_rel(5,-5);
    D_move_abs(scr_x,scr_y);
}


/*********************************************************************/
/***                      Box Plotting function.                   ***/
/*********************************************************************/


plot_box(scr_x,scr_y,top,bot,lef,rit,feature)
    int scr_x,scr_y,            /* Centre of box to plot.       */
        bot,top,lef,rit,        /* Graph boundaries.            */
        feature;                /* Morphometric feature.        */
{
    int n;
    n = (wsize-1)/2;            /* Number of samples.           */

    switch (feature)
    {
        case FLAT:
            R_standard_color(D_translate_color("gray"));
            break;
        case PIT:
```

```
            R_standard_color(D_translate_color("black"));
            break;
        case CHANNEL:
            R_standard_color(D_translate_color("blue"));
            break;
        case PASS:
            R_standard_color(D_translate_color("green"));
            break;
        case RIDGE:
            R_standard_color(D_translate_color("yellow"));
            break;
        case PEAK:
            R_standard_color(D_translate_color("red"));
            break;
        default:
            fprintf(stderr,"Error: Unknown terrain feature.\n");
            exit();
    }

    R_box_abs(scr_x,scr_y-(bot-top)/10,scr_x+(rit-lef)/n,scr_y);

    R_standard_color(D_translate_color("black"));
    D_move_abs(scr_x,scr_y);
    D_cont_abs(scr_x,scr_y-(bot-top)/10);
    D_cont_abs(scr_x+(rit-lef)/n,scr_y-(bot-top)/10);
    D_cont_abs(scr_x+(rit-lef)/n,scr_y);
    D_cont_abs(scr_x,scr_y);

}
```

process.c

```
/*****************************************************************/
/***                                                           ***/
/***                        process()                          ***/
/***          Reads in a raster file row by row for processing.***/
/***           Jo Wood, Project ASSIST, V1.0 7th February 1993 ***/
/***                                                           ***/
/*****************************************************************/

#include "param.h"

process()
{

    /*---------------------------------------------------------------*/
    /*                          INITIALISE                           */
    /*---------------------------------------------------------------*/


    CELL        *rast_ptr,      /* Buffer large enough to hold DEM.     */

                *window_ptr,    /* Stores local terrain window.         */
```

```
            centre;          /* Elevation of central cell in window. */

struct Cell_head region;     /* Structure to hold region information */

int         nrows,           /* Will store the current number of     */
            ncols,           /* rows and columns in the raster.      */

            row,             /* Counts through each row of raster.   */

            scr_x,scr_y,     /* Mouse screen coordinates.            */
            dem_x,dem_y,     /* Mouse DEM coordinates.               */

            button,          /* Mouse button pressed.                */

            wind_row,        /* Counts through each row and column   */
            wind_col,        /* of the local neighbourhood window.   */

            mwsize = wsize, /* Maximum local window size.            */

            *param_ptr,      /* Stores parameters at each scale.     */

            *index_ptr;      /* Row permutation vector for LU decomp.*/

float       **normal_ptr,    /* Cross-products matrix.               */
            *obs_ptr;        /* Observed vector.                     */

double      *weight_ptr;     /* Weighting matrix for observed values.*/


/*--------------------------------------------------------------------*/
/*                  GET RASTER AND WINDOW DETAILS                     */
/*--------------------------------------------------------------------*/

G_get_window(&region);       /* Fill out the region structure (the   */
                             /* geographical limits etc.)            */

nrows = G_window_rows();     /* Find out the number of rows and      */
ncols = G_window_cols();     /* columns of the raster.               */


if ((region.ew_res/region.ns_res >= 1.01) ||
    (region.ns_res/region.ew_res >= 1.01))
{
    G_warning("E-W and N-S grid resolutions are different.
                    Taking average.");
    resoln = (region.ns_res + region.ew_res)/2;
}
else
    resoln = region.ns_res;


/*--------------------------------------------------------------------*/
/*          RESERVE MEMORY TO HOLD Z VALUES AND MATRICES              */
/*--------------------------------------------------------------------*/
```

```
                              /* Reserver enough memory for raster.   */
rast_ptr   = (CELL *) G_malloc(ncols*nrows*sizeof(CELL));

                              /* Reserve enough memory for local wind.*/
window_ptr = (CELL *) G_malloc(wsize*wsize*sizeof(CELL));

                              /* Reserve enough memory for parameters.*/
param_ptr  = (int *) G_malloc(mwsize*sizeof(int));

                              /* Reserve enough memory weights matrix.*/
weight_ptr = (double *) G_malloc(wsize*wsize*sizeof(double));

normal_ptr = matrix(1,6,1,6); /* Allocate memory for 6*6 matrix     */
index_ptr  = ivector(1,6);    /* and for 1D vector holding indices  */
obs_ptr    = vector(1,6);     /* and for 1D vector holding observed z */

                              /* Read entire raster into memory.    */
for (row=0; row<nrows; row++)
    G_get_map_row(fd_in,rast_ptr + row*ncols,row);



/* ------------------------------------------------------------- */
/* -            CONTROL LOOP BASED ON MOUSE LOCATION           - */
/* ------------------------------------------------------------- */

do
{
                    /* Get array coordinates from mouse location.   */
    R_get_location_with_pointer(&scr_x,&scr_y,&button);
    dem_x = (int)D_d_to_a_col((double)scr_x);
    dem_y = (int)D_d_to_a_row((double)scr_y);

                    /* Only procede if correct mouse button is
                       pressed and mouse is within DEM.             */
    if ((button == 3) || (dem_x < 1) || (dem_x > ncols-2) ||
                         (dem_y < 1) || (dem_y > nrows-2))
        continue;

    for (wsize = 3; wsize <=mwsize; wsize += 2)
    {
                    /* Only procede if local window can be
                       extracted around the mouse location.         */
        if ( (dem_x < EDGE) || (dem_x > ncols-EDGE-1) ||
             (dem_y < EDGE) || (dem_y > nrows-EDGE-1))
            break;


        /* ------------------------------------------------------ */
        /* -         CALCULATE LEAST SQUARES COEFFICIENTS       - */
        /* ------------------------------------------------------ */

        /*--- Calculate weighting matrix. ---*/

        find_weight(weight_ptr);
```

```
/*--- Find normal equations in matrix form. ---*/

find_normal(normal_ptr,weight_ptr);


/*--- Apply LU decomposition to normal equations. ---*/

/* To constrain the quadratic through the central cell, ignore
   the calculations involving the coefficient f. Since these are
   all in the last row and column of the matrix, simply
   redimension.*/

if (constrained)
    ludcomp(normal_ptr,5,index_ptr);
else
    ludcomp(normal_ptr,6,index_ptr);


/*-------------------------------------------------------------*/
/*  PROCESS LOCAL WINDOW AND CALCULATE MORPHOMETRIC PARAM.  */
/*-------------------------------------------------------------*/

                            /* Find central z value */
centre = *(rast_ptr + dem_y*ncols + dem_x);

for (wind_row=0; wind_row<wsize; wind_row++)
    for (wind_col=0; wind_col<wsize; wind_col++)

                    /* Express all window values relative   */
                    /* to the central elevation.            */
        *(window_ptr+(wind_row*wsize)+wind_col) =
            *(rast_ptr+(dem_y+wind_row-EDGE)*ncols +
                            dem_x+wind_col-EDGE) - centre;


/*--- Use LU back substitution to solve normal equations. ---*/

find_obs(window_ptr,obs_ptr,weight_ptr);

if (constrained)
    lubksub(normal_ptr,5,index_ptr,obs_ptr);
else
    lubksub(normal_ptr,6,index_ptr,obs_ptr);


/*--- Calculate parameter based on quad. coefficients. ---*/

if (mparam == FEATURE)
    *(param_ptr + (wsize-3)/2) = feature(obs_ptr);
else
    if (mparam == ELEV)
        *(param_ptr + (wsize-3)/2) =
                            param(mparam,obs_ptr) + centre;
    else
        *(param_ptr + (wsize-3)/2) = param(mparam,obs_ptr);
```

```
        }

        /*-------------------------------------------------------------*/
        /*          REPORT PARAMETER VALUES AS GRAPH AND TEXT          */
        /*-------------------------------------------------------------*/

        wsize -= 2;             /* Set wsize back to original.         */
        disp_graph(scr_x,scr_y,param_ptr);

    }
    while (button != 3);



    /*-------------------------------------------------------------*/
    /*       FREE MEMORY TO STORE RASTER, LOCAL WINDOW AND MATRICES    */
    /*-------------------------------------------------------------*/

    free(rast_ptr);
    free(window_ptr);
    free(param_ptr);
    free_matrix(normal_ptr,1,6,1,6);
    free_vector(obs_ptr,1,6);
    free_ivector(index_ptr,1,6);
}
```

param.c

```
/*************************************************************************/
/***                                                                   ***/
/***                          param()                                  ***/
/*** Returns a terrain parameter based on 6 quadratic coefficents     ***/
/*** that define a local trend surface.                               ***/
/***  Jo Wood, Department of Geography, V2.0 15th December, 1994      ***/
/***                                                                   ***/
/*************************************************************************/

#include "param.h"
#include <math.h>


CELL param(ptype,coeff)
    int ptype;                  /* Type of terrain parameter to calculate */
    float *coeff;               /* Set of six quadratic coefficents.      */

{

    /* Quadratic function in the form of

                z = ax^2 + by^2 + cxy + dx + ey +f                         */

    double      a=C_A*zscale,           /* Rescale coefficients if a      */
                b=C_B*zscale,           /* Z scaling is required.         */
                c=C_C*zscale,
```

```
            d=C_D*zscale,
            e=C_E*zscale,
            f=C_F;                  /* f does not need rescaling as   */
                                    /* it is only used for smoothing. */

    switch(ptype)
    {
        case ELEV:
            return(RINT(f));
            break;

        case SLOPE:
            return(RINT(atan(sqrt(d*d + e*e))*RAD2DEG));
            break;

        case ASPECT:
            return(RINT(atan2(e,d)*RAD2DEG));
            break;

        case PROFC:
            if ((d == 0) && (e == 0))
                return((CELL)0);
            else
                return(RINT(-200.0*resoln*wsize*(a*d*d + b*e*e + c*e*d) /
                          ((e*e + d*d) * pow(1.0 + d*d + e*e,1.5)) ));
            break;

        case PLANC:
            if ((d == 0) && (e == 0))
                return((CELL)0);
            else
                return(RINT(200.0*resoln*wsize*(b*d*d + a*e*e - c*d*e) /
                                  powf(e*e + d*d,1.5) ));
            break;

        case LONGC:
            if ((d == 0) && (e ==0))
                return((CELL) 0);
            else
                return(RINT(-200.0*resoln*wsize*(a*d*d + b*e*e + c*d*e)/
                                                        (d*d + e*e)));
        case CROSC:
            if ((d == 0) && (e ==0))
                return((CELL) 0);
            else
                return(RINT(-200.0*resoln*wsize*(b*d*d + a*e*e - c*d*e)/
                                                        (d*d + e*e)));

        case MINIC:
            return(RINT(200.0*resoln*wsize*(-a-b-sqrt((a-b)*(a-b) +
                                                         c*c))));

        case MAXIC:
            return(RINT(200.0*resoln*wsize*(-a-b+sqrt((a-b)*(a-b) +
                                                         c*c))));
```

```
        default:
                return((CELL)0);
    }
}
```

feature.c

```
/*********************************************************************/
/***                                                               ***/
/***                          feature()                            ***/
/*** Returns a terrain feature based on 6 quadratic coefficents    ***/
/*** that define a local trend surface.                            ***/
/***     Jo Wood, Department of Geography, V2.1 30th March, 1995   ***/
/***                                                               ***/
/*********************************************************************/

#include "param.h"
#include <math.h>


CELL feature(coeff)
    float *coeff;               /* Set of six quadratic coefficents.    */

{

    /* Quadratic function in the form of

                z = ax^2 + by^2 + cxy + dx + ey +f                      */

    double      a=C_A*zscale,   /* Scale parameters if necessary.       */
                b=C_B*zscale,
                c=C_C*zscale,
                d=C_D*zscale,
                e=C_E*zscale,
                f=C_F*zscale;

    double maxic,minic,         /* Minimium and maximum curvature.      */
           slope,               /* Slope.                               */
           crosc;               /* Cross-sectional curvature.           */

    minic = 20*wsize*resoln*(-a-b-sqrt((a-b)*(a-b) + c*c));
    maxic = 20*wsize*resoln*(-a-b+sqrt((a-b)*(a-b) + c*c));
    slope = RAD2DEG*atan(sqrtf((d*d) + (e*e)));
    crosc = -20*wsize*resoln*(b*d*d + a*e*e - c*d*e)/(d*d + e*e);


    /* Case 1: Surface is sloping. Cannot be a peak,pass or pit. Therefore
               calculate the cross-sectional curvature to characterise as
               channel, ridge or planar.                                */

    if (slope > slope_tol)
    {
        if (crosc > curve_tol)
            return(RIDGE);
```

```
        if (crosc < -curve_tol)
            return(CHANNEL);

        else
            return(FLAT);
    }


    /* Case 2: Surface has (approximately) vertical slope normal. Feature
               can be of any type.                                          */

    if (maxic > curve_tol)
    {
        if (minic > curve_tol)
            return(PEAK);

        if (minic < -curve_tol)
            return(PASS);

        else
            return (RIDGE);
    }
    else
        if (minic < -curve_tol)
        {
            if (maxic < -curve_tol)
                return (PIT);

            else
                return (CHANNEL);
        }

    return (FLAT);
}
```

find_normal.c

```
/*****************************************************************/
/* find_normal() - Function to find the set of normal equations */
/*                 that allow a quadratic trend surface to be   */
/*                 fitted through N  points using least squares */
/*                 V.1.0, Jo Wood, 27th November, 1994.         */
/*****************************************************************/

#include "param.h"

find_normal(normal,w)
    float **normal;             /* Matrix of cross-products.    */
    double *w;                  /* Weights matrix.              */
{

    int edge=EDGE;              /* Store (wsize-1)/2 to save    */
                                /* on computation               */
```

```
float  x,y,                    /* Local coordinates of window. */
       x1=0,y1=0,              /* coefficients of X-products.  */
       x2=0,y2=0,
       x3=0,y3=0,
       x4=0,y4=0,
       xy2=0, x2y=0,
       xy3=0, x3y=0,
       x2y2=0,xy=0,
       N=0;

int   row,col;                 /* Pass through local window.   */


/* Initialise sums-of-squares and cross products matrix */

for (row=1; row<=6; row++)
    for (col=1; col<=6;col++)
        normal[row][col] = 0.0;


/* Calculate matrix of sums of squares and cross products */

for (row=0; row<wsize; row++)
    for (col=0; col<wsize; col++)
    {
       x = resoln*(col-EDGE);
       y = resoln*(row-EDGE);

       x4   += x*x*x*x* *(w + row*wsize + col);
       x2y2 += x*x*y*y* *(w + row*wsize + col);
       x3y  += x*x*x*y* *(w + row*wsize + col);
       x3   += x*x*x*   *(w + row*wsize + col);
       x2y  += x*x*y*   *(w + row*wsize + col);
       x2   += x*x*     *(w + row*wsize + col);

       y4   += y*y*y*y* *(w + row*wsize + col);
       xy3  += x*y*y*y* *(w + row*wsize + col);
       xy2  += x*y*y*   *(w + row*wsize + col);
       y3   += y*y*y*   *(w + row*wsize + col);
       y2   += y*y*     *(w + row*wsize + col);

       xy   += x*y*     *(w + row*wsize + col);

       x1   += x*       *(w + row*wsize + col);

       y1   += y*       *(w + row*wsize + col);

       N    +=          *(w + row*wsize + col);

    }

/* --- Store cross-product matrix elements. ---*/

normal[1][1] = x4;
normal[1][2] = normal[2][1] = x2y2;
```

```
    normal[1][3] = normal[3][1] = x3y;
    normal[1][4] = normal[4][1] = x3;
    normal[1][5] = normal[5][1] = x2y;
    normal[1][6] = normal[6][1] = x2;

    normal[2][2] = y4;
    normal[2][3] = normal[3][2] = xy3;
    normal[2][4] = normal[4][2] = xy2;
    normal[2][5] = normal[5][2] = y3;
    normal[2][6] = normal[6][2] = y2;

    normal[3][3] = x2y2;
    normal[3][4] = normal[4][3] = x2y;
    normal[3][5] = normal[5][3] = xy2;
    normal[3][6] = normal[6][3] = xy;

    normal[4][4] = x2;
    normal[4][5] = normal[5][4] = xy;
    normal[4][6] = normal[6][4] = x1;

    normal[5][5] = y2;
    normal[5][6] = normal[6][5] = y1;

    normal[6][6] = N;
}


/*****************************************************************/
/* find_obs() - Function to find the observed vector as part of */
/*              the set of normal equations for least squares.  */
/*              V.1.0, Jo Wood, 11th December, 1994.            */
/*****************************************************************/

find_obs(z,obs,w)
    CELL  *z;                   /* Local window of elevs.       */
    float *obs;                 /* Observed column vector.      */
    double  *w;                 /* Weighting matrix.            */

{

    int row,col,                /* Counts through local window. */
        edge=EDGE,              /* EDGE = (wsize-1)/2.          */
        offset;                 /* Array offset for weights & z */

    float x,y;                  /* Local window coordinates.    */

    for (row=1;row<=6; row++)   /* Initialise column vector.    */
        obs[row] = 0.0;


    for (row=0;row<wsize; row++)
        for (col=0; col<wsize; col++)
        {
            x = resoln*(col-EDGE);
            y = resoln*(row-EDGE);
```

```
            offset = row*wsize + col;

            obs[1] += *(w + offset) * *(z + offset) * x*x ;
            obs[2] += *(w + offset) * *(z + offset) * y*y ;
            obs[3] += *(w + offset) * *(z + offset) * x*y ;
            obs[4] += *(w + offset) * *(z + offset) * x ;
            obs[5] += *(w + offset) * *(z + offset) * y ;

            if (!constrained)    /* If constrained, should remain 0.0 */
                obs[6] += *(w + offset) * *(z + offset) ;
        }
}


/*************************************************************/
/* find_weight() Function to find the weightings matrix for the */
/*               observed cell values.                       */
/*               Uses an inverse distance function that can be  */
/*               calibrated with an exponent (0= no decay,   */
/*               1=linear decay, 2=squared distance decay etc.) */
/*               V.1.1, Jo Wood, 11th May, 1995.             */
/*************************************************************/

find_weight(weight_ptr)
    double *weight_ptr;
{
    int row,col;                /* Counts through the rows and  */
                                /* columns of weights matrix.   */

    double dist,                /* Distance to centre of kernel.*/
           sum_weight=0;        /* Sum of weights.              */


    /* --- Find inverse distance of all cells to centre. ---*/

    for (row=0; row<wsize; row++)
        for (col=0; col<wsize; col++)
        {
            dist = 1.0 /
                pow(sqrt((EDGE-col)*(EDGE-col) +
(EDGE-row)*(EDGE-row))+1.0,exponent);
            *(weight_ptr + row*wsize + col) = dist;
        }
}

/*************************************************************/
/* lubksub() - Function to perform forward and back substitution*/
/*               on an LU decomposed matrix. Used for solving a  */
/*               set of linear equations.                    */
/*               Adapted from Press et al (1988), p.44.      */
/*               V.1.0, Jo Wood, 10th December, 1994.        */
/*************************************************************/


lubksub(a,n,index,b)
```

```
    int     n,          /* Size of side of matrix               */
            *index;     /* Row permutation effected by pivoting */
    float   **a,        /* Matrix to be decomposed.             */
            *b;         /* Input as RHS vector, output as soln. */

{
    int    i, ii=0, ip, j;
    float  sum;

    for (i=1; i<=n; i++)
    {
        ip    = index[i];
        sum   = b[ip];
        b[ip] = b[i];

        if (ii)
            for (j=ii; j<=i-1; j++)
                sum -= a[i][j] * b[j];
        else
            if (sum)
                ii=i;

        b[i] = sum;
    }

    for (i=n; i>=1; i--)
    {
        sum = b[i];
        for (j=i+1; j<=n; j++)
            sum -= a[i][j] * b[j];

        b[i] = sum/a[i][i];
    }
}

/*************************************************************/
/* ludcomp() - Function to perform LU decomposition of a matrix.*/
/*             Used to find inverse of the matrix when solving */
/*             sets of linear equations.                       */
/*             Adapted from Press et al (1988), pp.43-45.      */
/*             V.1.0, Jo Wood, 9th December, 1994.             */
/*************************************************************/

ludcomp(a,n,index)
    int     n,          /* Size of side of matrix               */
            *index;     /* Row permutation effected by pivoting */
    float   **a;        /* Matrix to be decomposed.             */

{
    int     i, imax,j,k;
    float   big,dum,sum,temp,
            *vv;

    vv = vector(1,n);
```

```
for (i=1; i<=n; i++)     /* Loop over rows, get scaling info.*/
{
    big = 0.0;
    for (j=1; j<=n; j++)
        if ((temp=fabs(a[i][j])) > big)
            big = temp;

    if (big == 0.0)
    {
        fprintf(stderr,"Singular matrix - can't perform LU
                                                    decomposition\n");
        exit(-1);
    }

    vv[i] = 1.0/big;     /* Save the scaling. */
}


for (j=1; j<=n; j++)
{
    for (i=1; i<j; i++)
    {
        sum=a[i][j];
        for (k=1; k<i; k++)
            sum -= a[i][k] * a[k][j];
        a[i][j] = sum;
    }

    big=0.0;

    for (i=j; i<=n; i++)
    {
        sum=a[i][j];
        for (k=1; k<j; k++)
            sum -= a[i][k] * a[k][j];
        a[i][j] = sum;

        if ( (dum=vv[i]*fabs(sum)) >= big)
        {
            big = dum;
            imax = i;
        }
    }

    if (j != imax)
    {
        for (k=1; k<=n; k++)
        {
            dum = a[imax][k];
            a[imax][k] = a[j][k];
            a[j][k] = dum;
        }
        vv[imax] = vv[j];
    }
```

```
        index[j] = imax;
        if (a[j][j] == 0.0)
            a[j][j] = TINY;

        if (j != n)
        {
            dum = 1.0/(a[j][j]);

            for (i=j+1; i<=n; i++)
                a[i][j] *= dum;
        }
    }

    free_vector(vv,1,n);
}
```

nrutil.c

```
/*************************************************************/
/* nrutil.c     Functions to reserve memory fo rthe storage of  */
/*              matrices and vectors. From Press et al (1988)   */
/*              See Appendix D pp.705-709.                      */
/*              V.1.0, Jo Wood, 9th December, 1994.             */
/*************************************************************/

#include <stdio.h>

/*************************************************************/
/* Set up a 1D floating point matrix with range [nl to nh]      */
/* See Press et al (1988) p.705.                                */
/*************************************************************/

float *vector(nl,nh)
    int nl,nh;
{
     float *v;

     v = (float *) malloc((unsigned) (nh-nl+1)*sizeof(float));

     if (!v)
     {
        fprintf(stderr,"ERROR: Can't allocate memory for vector\n");
        exit(-1);
     }

     return (v-nl);
}


/*************************************************************/
/* Set up a 1D integer matrix with range [nl to nh]             */
/* See Press et al (1988) p.706.                                */
/*************************************************************/
```

```
int *ivector(nl,nh)
    int nl,nh;
{
     int *v;

     v = (int *) malloc((unsigned) (nh-nl+1)*sizeof(int));

     if (!v)
     {
        fprintf(stderr,"ERROR: Can't allocate memory for vector\n");
        exit(-1);
     }

     return (v-nl);
}


/*************************************************************/
/* Set up a 2D floating point matrix                         */
/* with range [nrl to nrh][ncl to nch]                       */
/* See Press et al (1988) p.706.                             */
/*************************************************************/

float **matrix(nrl,nrh,ncl,nch)
    int nrl,nrh,ncl,nch;
{
    int     i;
    float **m;

    m = (float **) malloc((unsigned) (nch-ncl+1)*sizeof(float));

    if (!m)
    {
        fprintf(stderr,"ERROR: Can't allocate memory for matrix rows\n");
        exit(-1);
    }

    for (i=nrl; i<=nrh; i++)
    {
        m[i] = (float *) malloc((unsigned) (nch-ncl+1)*sizeof(float));

        if (!m[i])
        {
            fprintf(stderr,
                        "ERROR: Can't allocate memory for matrix columns\n");
            exit(-1);
        }
        m[i] -= ncl;
    }

    return (m);
}
```

```
/*****************************************************************/
/* Free memory used for storing 1D floating point matrix.        */
/* See Press et al (1988) p.707.                                 */
/*****************************************************************/

free_vector(v,nl,nh)
    float *v;
    int   nl,nh;
{
    free((char *) (v+nl));
}


/*****************************************************************/
/* Free memory used for storing 1D integer matrix.               */
/* See Press et al (1988) p.707.                                 */
/*****************************************************************/

free_ivector(v,nl,nh)
    int *v,
         nl,nh;
{
    free((char *) (v+nl));
}


/*****************************************************************/
/* Free memory used for storing 2D floating point matrix.        */
/* See Press et al (1988) p.708.                                 */
/*****************************************************************/

free_matrix(m,nrl,nrh,ncl,nch)
    float **m;
    int   nrl,nrh,ncl,nch;
{
    int i;

    for (i=nrh; i>=nrl; i--)
        free ((char *) (m[i]+ncl));

    free((char *) (m+nrl));
}
```

close_down.c

```
/*****************************************************************/
/***                                                           ***/
/***                        close_down()                       ***/
/***  Closes all input and output raster files and frees memory ***/
/***       Jo Wood, Dept. of Geography, 7th February 1993      ***/
/***                                                           ***/
/*****************************************************************/

#include "param.h"
```

```
close_down()
{
    /* Close connection with existing input raster. */

    G_unopen_cell(fd_in);

    R_close_driver();

}
```

r.basin

Calculates drainage basins by recursive backtracking. Allows slope tolerance values to be specified to alter scale of sub-division. Can also produce 'hydrologically enforced' DEMs by flooding internal pits.

V1.2 Jo Wood, 13th September, 1994

basin.h

```
/*****************************************************************/
/***                          basin.h                          ***/
/***                Header file for use with r.basin           ***/
/***             Jo Wood, V 1.0  - 13th September, 1994        ***/
/*****************************************************************/

#include "gis.h"                /* This MUST be included in all GRASS   */
                                /* programs. It sets up the necessary   */
                                /* prototypes for GRASS library calls.  */
#include <limits.h>

#define VISITED 128
#define RAISED 64

#define NO_OUTLET -1
#define EDGE 1
#define NOT_EDGE 2

#define TRUE 1
#define FALSE 0

struct Elevation
{
    unsigned short z;           /* Elevation value (0-65535)            */
      signed char  drainto;     /* Location of cell to drain into.      */
    unsigned char  flags;       /* Various flags (up to 8)              */
    unsigned short basin;       /* Drainage basin index (0-65535).      */
};

/* ------ Global variables ------ */

#ifndef MAIN
    extern                      /* Externally defined if not main()     */
#endif

char    *RastInName,            /* Name of DEM to process.              */
        *RastOutName,           /* Name of the raster output file.      */

        *MapsetIn,              /* Names of mapsets containing the      */
        *MapsetOut;             /* files to be processed.               */

        NoPits;                 /* Flags hydrological correction of DEM */
```

```
#ifndef MAIN
    extern
#endif

int     FdIn,                   /* File descriptor for input and        */
        FdOut,                  /* output raster files.                 */

        nRows,nCols,            /* Number of rows and columns in DEM.   */
        Neighbour[8],           /* Holds location of a cell's 8 neighs. */

        zTol;                   /* Vertical tolerance for slope calcn.  */

        BasinNum;               /* Drainage basin index.                */

#ifndef MAIN
    extern
#endif

struct Elevation *Dem;          /* Array holding entire DEM and flags.  */
```

main.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                        main.c for r.basin                             ***/
/***         GRASS module to identify drainage basins in a DEM             ***/
/***                             Jo Wood                                   ***/
/***                    v 1.2  13th September, 1994                        ***/
/***                                                                       ***/
/*****************************************************************************/

#define MAIN

#include "basin.h"

CELL main(argc,argv)
    int argc;
    char *argv[];
{
    /*---------------------------------------------------------------------*/
    /*                        INITIALISE                                   */
    /*---------------------------------------------------------------------*/

    BasinNum = 1;

    /*---------------------------------------------------------------------*/
    /*                      GET INPUT FROM USER                            */
    /*---------------------------------------------------------------------*/

    interface(argc,argv);

    /*---------------------------------------------------------------------*/
```

```
    /*                OPEN INPUT AND OUTPUT RASTER FILES                    */
    /*------------------------------------------------------------------------*/

    open_files();


    /*------------------------------------------------------------------------*/
    /*                      PROCESS RASTER FILES                              */
    /*------------------------------------------------------------------------*/

    process();

    /*------------------------------------------------------------------------*/
    /*                    CLOSE ALL OPENED FILES AND FREE MEMORY              */
    /*------------------------------------------------------------------------*/

    close_down();

}
```

interface.c

```
/*****************************************************************************/
/***      Function to get input from user and check files can be opened    ***/
/***                                                                       ***/
/***                    Jo Wood,  V1.0, 13th September 1994                ***/
/*****************************************************************************/

#include "basin.h"

interface(argc,argv)

    int      argc;              /* Number of command line arguments     */
    char    *argv[];            /* Contents of command line arguments.  */

{
    /*------------------------------------------------------------------------*/
    /*                                INITIALISE                              */
    /*------------------------------------------------------------------------*/

    struct Option       *rast_in;
    struct Option       *rast_out;
    struct Option       *tolerance;
    struct Flag         *no_pits;

    G_gisinit (argv[0]);               /* Link with GRASS interface.      */



    /*------------------------------------------------------------------------*/
    /*                             SET PARSER OPTIONS                         */
    /*------------------------------------------------------------------------*/
```

```
    rast_in  = G_define_option();       /* Pointer to reserved memory for  */
    rast_out = G_define_option();       /* each command line option.       */
    tolerance= G_define_option();
    no_pits  = G_define_flag();

    /* Each option needs a 'key' (short description),
                          a 'description` (a longer one),
                          a 'type' (eg intiger, or string),
                      and an indication whether manditory or not */

    rast_in->key          = "dem";
    rast_in->description  = "Digital Elevation Model to process";
    rast_in->type         = TYPE_STRING;
    rast_in->required     = YES;

    rast_out->key         = "out";
    rast_out->description = "Output raster layer containing drainage basins";
    rast_out->type        = TYPE_STRING;
    rast_out->required    = YES;

    tolerance->key        = "tol";
    tolerance->description= "Vertical tolerance value";
    tolerance->type       = TYPE_INTEGER;
    tolerance->required   = NO;
    tolerance->answer     = "0";

    no_pits->key          = 'h';
    no_pits->description  = "Produce hydrologically correct DEM";

    if (G_parser(argc,argv))            /* Performs the prompting for      */
        exit(-1);                       /* keyboard input.                 */

    RastInName  = rast_in->answer;      /* Now that keyboard input has     */
    RastOutName = rast_out->answer;     /* been parsed, place the contents */
    zTol = atoi(tolerance->answer);     /* contents into global variables. */
    if (no_pits->answer)
        NoPits = TRUE;

    /*--------------------------------------------------------------------*/
    /*                    CHECK INPUT RASTER FILES EXIST                  */
    /*--------------------------------------------------------------------*/

    if ((MapsetIn=G_find_cell2(RastInName,""))==NULL)
    {
        char err[256];
        sprintf(err,"Raster map [%s] not available.",RastInName);
        G_fatal_error(err);
    }



    /*--------------------------------------------------------------------*/
    /*                  CHECK OUTPUT RASTER FILE DOES NOT EXIST           */
    /*--------------------------------------------------------------------*/
```

```
    MapsetOut = G_mapset();             /* Set output to current mapset.  */

    if (G_legal_filename(RastOutName)==NULL)
    {
        char err[256];
        sprintf(err,"Illegal output file name. Please try another.");
        G_fatal_error(err);
    }
    else
    {
        if (G_find_cell2(RastOutName,MapsetOut) !=NULL)
        {
            char err[256];
            sprintf(err,
                "Raster map [%s] exists.\nPlease try another\n",RastOutName);
            G_fatal_error(err);
        }
    }
}
```

open_files.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                            open_files()                               ***/
/***          Opens input and output raster files for r.basin              ***/
/***                   Jo Wood, V1.0, 13th September, 1994                 ***/
/***                                                                       ***/
/*****************************************************************************/

#include "basin.h"

open_files()
{
    /* Open existing file and set the input file descriptors */

    if ( (FdIn=G_open_cell_old(RastInName,MapsetIn)) <0)
    {
        char err[256];
        sprintf(err,"ERROR: Problem opening DEM file.");
        G_fatal_error(err);
    }

    /* Open new file and set the output file descriptor. */

    if ( (FdOut=G_open_cell_new(RastOutName)) <0)
    {
        char err[256];
        sprintf(err,"ERROR: Problem opening output file.");
        G_fatal_error(err);
    }
}
```

process.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                            process()                                  ***/
/***          Reads in a raster files row by row for processing            ***/
/***                  Jo Wood, V1.0, 13th September, 1994                  ***/
/***                                                                       ***/
/*****************************************************************************/

#include "basin.h"

process()
{

    /*-------------------------------------------------------------------*/
    /*                            INITIALISE                             */
    /*-------------------------------------------------------------------*/

    CELL        *row_in,        /* Buffers to hold raster rows.          */
                *row_out;

    int         row,col,        /* Counts through each row and column    */
                                /* of the input raster.                  */

                out_cell,       /* Location of drainage basin outlet.    */
                status,         /* Status of drainage basin outlet.      */
                offset;         /* Counts through DEM to reset flags.    */

    char        altered;        /* Flags if any cells have been altered */

    /*-------------------------------------------------------------------*/
    /*                      GET DETAILS OF INPUT RASTER                  */
    /*-------------------------------------------------------------------*/

    row_in  = G_allocate_cell_buf();
    row_out = G_allocate_cell_buf(); /* Allocate row buffer for output. */

    nRows = G_window_rows();         /* Find out the number of rows and */
    nCols = G_window_cols();         /* columns of the raster view.     */


    /*-------------------------------------------------------------------*/
    /*                UPDATE ARRAY HOLDING NEIGHBOUR POSITIONS           */
    /*-------------------------------------------------------------------*/

    Neighbour[0] =  1 - 0;
    Neighbour[1] =  1 - nCols;
    Neighbour[2] =  0 - nCols;
    Neighbour[3] = -1 - nCols;
    Neighbour[4] = -1 - 0;
    Neighbour[5] = -1 + nCols;
    Neighbour[6] =  0 + nCols;
    Neighbour[7] =  1 + nCols;
```

```
/*--------------------------------------------------------------------*/
/*              RESERVE ENOUGH MEMORY AND STORE ENTIRE DEM            */
/*--------------------------------------------------------------------*/

                                        /* Reserve the memory.        */
Dem = (CELL) G_malloc(sizeof(struct Elevation)*nRows*nCols);

                                    /* Transfer DEM to memory by row.  */
for(row=0; row<nRows; row++)
{
    G_get_map_row(FdIn, row_in, row);

    for (col=0; col<nCols; col++)
        (Dem+row*nCols+col)->z = (unsigned short) *(row_in + col);
}

/*--------------------------------------------------------------------*/
/*      PROCESS INPUT RASTER AND WRITE OUT RASTER LINE BY LINE        */
/*--------------------------------------------------------------------*/

find_drainto();                    /* Fill out the drainto structure. */


if (NoPits)
{
    row=0;
                                    /* Correct the DEM. */
    do
    {
        altered = FALSE;

        while( (status = find_basin(&out_cell)) )
        {
            if (status == NOT_EDGE)
            {
                raise_cell(out_cell);
                altered=TRUE;
            }
        }

        for(offset=0;offset<nRows*nCols; offset++)
            (Dem + offset)->flags = (Dem + offset)->flags & ~VISITED;

        printf("\n");
        fflush(stdout);
    }
    while (altered == TRUE);

    for(row=0; row<nRows; row++)
    {
        for(col = 0; col < nCols; col++)
            *(row_out + col) = (Dem + row*nCols+col)->z;

        G_put_map_row(FdOut,row_out);
```

```
        }
    }
    else
    {                                   /* Find Basins. */
        while (find_basin(&out_cell))
            BasinNum++;
                                        /* Transfer memory to output file.  */
        for(row=0; row<nRows; row++)
        {
            for(col = 0; col < nCols; col++)
                if ((Dem + row*nCols + col)->flags & VISITED)
                    *(row_out + col) = (Dem + row*nCols+col)->basin;
                else
                    *(row_out + col) = 0;

            G_put_map_row(FdOut,row_out);
        }
    }

    free(Dem);
}
```

drainto.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                          find_drainto()                               ***/
/***             Identifies the cell into which each cell drains.          ***/
/***                   Jo Wood, V1.0, 15th September, 1994                 ***/
/***                                                                       ***/
/*****************************************************************************/

#include "basin.h"

find_drainto()
{

    /*-------------------------------------------------------------------*/
    /*                           INITIALISE                              */
    /*-------------------------------------------------------------------*/

    int offset,                 /* Neighbouring cell offset.             */
        centre,                 /* Location of central cell in window.   */
        drainto,                /* Location of cell to drain to.         */
        row,col,                /* Counts through rows and cols of DEM.  */
        drop,max_drop;          /* Drop and max drop to draining cell.   */


    /*-------------------------------------------------------------------*/
    /*                     SCAN THROUGH ENNTIRE DEM                      */
    /*-------------------------------------------------------------------*/

    for (row=1; row<nRows-1; row++)
```

```
    for (col=1; col<nCols-1; col++)
    {
        centre = row*nCols+col;
        max_drop = -1;
        drainto = NO_OUTLET;

        for(offset=0; offset<8; offset++)
        {
            drop= (Dem + centre)->z -
                            (Dem + centre + Neighbour[offset])->z;

            if (offset%2)
                drop *= 0.707;

            if (drop >= max_drop)
                if (max_drop == 0)
                    drainto = NO_OUTLET;
                else
                {
                    max_drop=drop;
                    drainto = offset;
                }
        }

        (Dem + centre)->drainto = drainto;
    }


for (col=1;col<nCols;col++)
{
    centre = col;
    max_drop = -1;
    drainto = NO_OUTLET;

    for(offset=0; offset<8; offset++)
    {
        if ((offset !=1) && (offset !=2) && (offset!=3))
        {
            drop= (Dem + centre)->z -
                            (Dem + centre + Neighbour[offset])->z;
            if (offset%2)
                drop *= 0.707;

            if (drop >= max_drop)
                if (max_drop == 0)
                    drainto = NO_OUTLET;
                else
                {
                    max_drop=drop;
                    drainto = offset;
                }
        }
    }
    (Dem + centre)->drainto = drainto;
```

```
    centre = (nRows-1)*nCols + col;
    max_drop = -1;
    drainto = NO_OUTLET;

    for(offset=0; offset<5; offset++)
    {
        drop= (Dem + centre)->z - (Dem + centre + Neighbour[offset])->z;
        if (offset%2)
            drop *= 0.707;

        if (drop >= max_drop)
            if (max_drop == 0)
                drainto = NO_OUTLET;
            else
            {
                max_drop=drop;
                drainto = offset;
            }
    }
    (Dem + centre)->drainto = drainto;
}


for (row=1; row<nRows; row++)
{
    centre = row*nCols;
    max_drop = -1;
    drainto = NO_OUTLET;

    for(offset=0; offset<8; offset++)
    {
        if ((offset !=3) && (offset !=4) && (offset!=5))
        {
            drop= (Dem + centre)->z -
                            (Dem + centre + Neighbour[offset])->z;
            if (offset%2)
                drop *= 0.707;

            if (drop >= max_drop)
                if (max_drop == 0)
                    drainto = NO_OUTLET;
                else
                {
                    max_drop=drop;
                    drainto = offset;
                }
        }
    }

    (Dem + centre)->drainto = drainto;

    centre = (row+1)*nCols -1;
    max_drop = -1;
    drainto = NO_OUTLET;
```

```
        for(offset=0; offset<8; offset++)
        {
            if ((offset !=1) && (offset !=0) && (offset!=7))
            {
                drop= (Dem + centre)->z -
                                (Dem + centre + Neighbour[offset])->z;
                if (offset%2)
                    drop *= 0.707;

                if (drop >= max_drop)
                    if (max_drop == 0)
                        drainto = NO_OUTLET;
                    else
                    {
                        max_drop=drop;
                        drainto = offset;
                    }
            }
        }

        (Dem + centre)->drainto = drainto;
    }
}
```

edge.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                           is_edge()                                   ***/
/***        Tests to see if a location is on the edge of a DEM.            ***/
/***                  Jo Wood, V1.0, 15th September, 1994                  ***/
/***                                                                       ***/
/*****************************************************************************/

#include "basin.h"

int is_edge(location)
   int location;                /* Location of cell to be tested.       */
{
    if ( (location <  nCols)             ||      /* Top row.             */
         (location >= nCols*(nRows-1))   ||      /* Bottom row.          */
         (location % nCols     == 0)     ||      /* Left row.            */
         ((location+1) % nCols == 0))            /* Right row.           */
        return(TRUE);
    else
        return(FALSE);
}
```

find_basin.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                          find_basin()                                 ***/
/***    Scans up and down DEM to find drainage basins.                     ***/
/***                  Jo Wood, V1.0, 13th September, 1994                  ***/
/***                                                                       ***/
/*****************************************************************************/

#include "basin.h"

int find_basin(out_cell)
    int *out_cell;              /* Location of outlet cell.              */
{

    /*-----------------------------------------------------------------*/
    /*                          FLAG OUTLET CELL                       */
    /*-----------------------------------------------------------------*/

    *out_cell = find_outlet();

    if (*out_cell == NO_OUTLET)
        return(FALSE);
    else
    {
        move_up(*out_cell);

        if (!is_edge(*out_cell))
            return(NOT_EDGE);
        else
            return(EDGE);
    }
}
```

find_outlet.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                            find_outlet()                              ***/
/***        Scans up and down DEM to find drainage basin outlet.           ***/
/***                  Jo Wood, V1.0, 13th September, 1994                  ***/
/***                                                                       ***/
/*****************************************************************************/

#include "basin.h"

int find_outlet()
{
```

```
    /*--------------------------------------------------------------------*/
    /*                            INITIALISE                              */
    /*--------------------------------------------------------------------*/

    int offset,                 /* Counts through DEM cells.              */
        out_cell,               /* Location of outlet cell in DEM.        */
        num_cells = nRows*nCols;/* Number of cells in DEM.                */

    CELL lowest = INT_MAX;      /* Lowest cell in DEM (probably oulet).   */
                                /* NOTE: Assumes type CELL is int         */

    char newcell = FALSE;       /* Flag if any new cells to process.      */


    /*--------------------------------------------------------------------*/
    /*                          SCAN DEM FOR LOWEST CELL                  */
    /*--------------------------------------------------------------------*/

    for (offset=0; offset < num_cells; offset++)
        if ( !((Dem + offset)->flags & VISITED) )
            if ((Dem + offset)->z <= lowest)
                if (((Dem + offset)->z < lowest) || (is_edge(offset)))
                {
                    lowest = (Dem+offset)->z;
                    out_cell = offset;
                    newcell = TRUE;
                }


    /*--------------------------------------------------------------------*/
    /*                     RETURN LOCATION OF LOWEST CELL                 */
    /*--------------------------------------------------------------------*/

    if (newcell == TRUE)
        return (out_cell);
    else
        return (NO_OUTLET);
}
```

raise_cell.c

```
/**************************************************************************/
/***                                                                    ***/
/***                          raise_cell()                              ***/
/***    Raises DEM cells if they are at the base of an internal basin.  ***/
/***                 Jo Wood, V1.0, 15th September, 1994                ***/
/***                                                                    ***/
/**************************************************************************/

#include "basin.h"

raise_cell(location)
   int location;                /* Location of outlet cell.               */
{
```

```
    /*--------------------------------------------------------------------*/
    /*                              INITIALISE                            */
    /*--------------------------------------------------------------------*/

    unsigned short lowest = 65535;      /* Lowest neighbour.              */
    int  offset;                /* Counts through all DEM cells.          */


    /*--------------------------------------------------------------------*/
    /*          SET CENTRAL CELL TO LOWEST OF THE 8 NEIGHBOURS            */
    /*--------------------------------------------------------------------*/

    find_lowest(location,&lowest);

    set_to_lowest(location,lowest);

    printf(".");
    fflush(stdout);
}


/***********************************************/
/* find_lowest() - Recursive downflow detection */
/***********************************************/

find_lowest(location1,lowest)
    int location1;
    unsigned short *lowest;
{
    int offset,                 /* Counts through neighbouring cells     */
        location2;              /* Rel. location of neighbouring cell    */

    (Dem + location1)->flags = (Dem + location1)->flags | RAISED;


    for (offset=0; offset <8; offset++)
    {
        location2 = location1 + Neighbour[offset];

        if ((Dem + location2)->z < *lowest)
        {
            if ((Dem + location2)->z > (Dem + location1)->z)
                *lowest = (Dem + location2)->z;
            else
                if ((!is_edge(location2)) &&
                    (!((Dem+location2)->flags & RAISED)))
                    find_lowest(location2,lowest);
        }
    }
}

/***************************************/
/* set_to_lowest() - Recursive function */
/***************************************/
```

```
set_to_lowest(location1,lowest)
    int location1;
    unsigned short lowest;
{
    int offset,                 /* Counts through neighbouring cells    */
        location2;              /* Rel. location of neighbouring cell   */

    (Dem + location1)->z = lowest;
    (Dem + location1)->flags = (Dem + location1)->flags & ~RAISED;

    for (offset=0; offset <8; offset++)
    {
        location2 = location1 + Neighbour[offset];

        if (((Dem + location2)->z < lowest) && (!is_edge(location2)))
            set_to_lowest(location2,lowest);
    }
}
```

move_up.c

```
/****************************************************************************/
/***                                                                      ***/
/***                           move_up()                                  ***/
/***    Traverses a drainage basin by moving up from any given position ***/
/***                  Jo Wood, V1.0, 13th September, 1994                 ***/
/***                                                                      ***/
/****************************************************************************/

#include "basin.h"

move_up(location1)
   int location1;       /* Location of central cell in local window.    */
{

    /*--------------------------------------------------------------------*/
    /*                              INITIALISE                            */
    /*--------------------------------------------------------------------*/

    int offset,                 /* Location index of neighbouring cell. */
        location2;              /* Location of neighbouring cell.       */

    (Dem + location1)->flags = (Dem + location1)->flags | VISITED;
    (Dem + location1)->basin = BasinNum;

    /*--------------------------------------------------------------------*/
    /*                         SCAN ALL 8 NEIGHBOURS                      */
    /*--------------------------------------------------------------------*/

    for (offset=0; offset <8; offset++)
    {
        location2 = location1 + Neighbour[offset];

        if ((abs((location2 % nCols) - (location1 % nCols)) <= 1) &&
```

```
            (location2 >=0) && (location2 < nRows*nCols) &&

            (!((Dem + location2)->flags & VISITED)) &&

            ((Dem + location2)->z >= (Dem+location1)->z  -zTol)  &&

            ((Neighbour[(Dem + location2)->drainto] +
                                              Neighbour[offset] == 0)||
             ((Dem + location2)->drainto == NO_OUTLET)))

               move_up(location2);
    }
}
```

r.param.scale

GRASS module that extracts terrain parameters from a DEM. Uses a multi-scalar approach by taking fitting quadratic parameters to any size window (via least squares).

Jo Wood, 27th November, 1994.

Modified to include constrained fitting, April, 1995

Modified to include weighting matrix and double precision arithmetic, 9th May, 1995.

Modified 23rd May to inclue two separate tolerance values for feature detection.

param.h

```
/*****************************************************************************/
/***                                                                       ***/
/***                            param.h                                    ***/
/***  Header file for use with r.param.scale                               ***/
/***  Jo Wood, Department of Geography, University of Leicester            ***/
/***  V1.0  - 7th February, 1993                                           ***/
/***                                                                       ***/
/*****************************************************************************/

#include "gis.h"                /* This MUST be included in all GRASS   */
                                /* programs. It sets up the necessary   */
                                /* prototypes for GRASS library calls.  */
#include <math.h>

#define EDGE ((wsize-1)/2)      /* Number of rows/cols that make up the */
                                /* 'blank' edge around raster.          */
#define MAX_WSIZE 69            /* Maximum dimensions of window.        */

                                /* Some useful labels.                  */
#define TRUE 1
#define FALSE 0

#define RAD2DEG 57.29578
#define DEG2RAD 0.017453293

#define TINY 1.0e-20;

#define FLAT ((CELL)0)
#define PIT ((CELL)1)
#define CHANNEL ((CELL)2)
#define PASS ((CELL)3)
#define RIDGE ((CELL)4)
#define PEAK ((CELL)5)
```

```
#define NUM_CATS ((CELL)6)

#define ELEV   1
#define SLOPE  2
#define ASPECT 3
#define PROFC  4
#define PLANC  5
#define LONGC  6
#define CROSC  7
#define MINIC  8
#define MAXIC  9
#define FEATURE 10

/* The six quadratic coefficients are stored in the array coeff */

#define C_A coeff[1]
#define C_B coeff[2]
#define C_C coeff[3]
#define C_D coeff[4]
#define C_E coeff[5]
#define C_F coeff[6]

/* ------ Declare functions ----- */

float   *vector(),              /* Reserves memory for 1D matrix.       */
        **matrix();             /* Reserves memory for 2D matrix.       */

int     *ivector();             /* Reserves memory for 1D int matrix.   */

CELL    param();                /* Calculates terrain parameters.       */

/* ------ Global variables ------ */

#ifndef MAIN
    extern                      /* Externally defined if not main()     */
#endif

char            *rast_in_name,  /* Name of the raster file to process.  */
                *rast_out_name, /* Name of the raster output file.      */

                *mapset_in,     /* If no problems, these will be names  */
                *mapset_out,    /* of mapsets containing the files to   */
                                /* be processed. Otherwise, error code. */

                constrained;    /* Flag that forces quadtratic through  */
                                /* the central cell of the window.      */

#ifndef MAIN
    extern                      /* Externally defined if not main()     */
#endif

int             fd_in,          /* File descriptor for input and        */
                fd_out,         /* output raster files.                 */

                wsize,          /* Size of local processing window.     */
```

```
                mparam;         /* Morphometric parameter to calculate. */


#ifndef MAIN
    extern                      /* Externally defined if not main()     */
#endif

double          resoln,         /* Planimetric resolution.              */
                exponent,       /* Distance weighting exponent.         */
                zscale,         /* Vertical scaling factor.             */

                slope_tol,      /* Vertical tolerences for surface      */
                curve_tol;      /* feature identification.              */
```

main.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                            r.param.scale                              ***/
/***      GRASS module for extracting multi-scale surface parameters.      ***/
/**                                                                        ***/
/**                                                                        ***/
/**                       Jo Wood, V 1.1, 11th December, 1994              ***/
/**                                                                        ***/
/*****************************************************************************/

#define MAIN

#include "param.h"

main(argc,argv)
    int argc;
    char *argv[];
{

    /*-------------------------------------------------------------------*/
    /*                           INITIALISE                              */
    /*-------------------------------------------------------------------*/



    /*-------------------------------------------------------------------*/
    /*                        GET INPUT FROM USER                        */
    /*-------------------------------------------------------------------*/

    interface(argc,argv);


    /*-------------------------------------------------------------------*/
    /*                OPEN INPUT AND OUTPUT RASTER FILES                 */
    /*-------------------------------------------------------------------*/

    open_files();
```

```
    /*--------------------------------------------------------------------*/
    /*               PROCESS SURFACE FOR FEATURE DETECTION                */
    /*--------------------------------------------------------------------*/

    process();

    /*--------------------------------------------------------------------*/
    /*               CLOSE ALL OPENED FILES AND FREE MEMORY               */
    /*--------------------------------------------------------------------*/

    close_down();

    if (mparam == FEATURE)
    {
        write_cols();
        write_cats();
    }

}
```

interface.c

```
/*****************************************************************************/
/***                              interface()                              ***/
/***    Function to get input from user and check files can be opened      ***/
/***                                                                       ***/
/***      Jo Wood, Department of Geography, V1.2, 7th February 1992        ***/
/*****************************************************************************/

#include "param.h"

interface(argc,argv)

    int      argc;              /* Number of command line arguments.    */
    char    *argv[];            /* Contents of command line arguments.  */

{
    /*--------------------------------------------------------------------*/
    /*                           INITIALISE                               */
    /*--------------------------------------------------------------------*/

    struct Option  *rast_in,    /* Name of input file from command line.*/
                   *rast_out,   /* Holds name of output file.           */
                   *tol1_val,   /* Tolerance values for feature         */
                   *tol2_val,   /* detection (slope and curvature).     */
                   *win_size,   /* Size of side of local window.        */
                   *parameter,  /* Morphometric parameter to calculate. */
                   *expon,      /* Inverse distance exponent for weight.*/
                   *vert_sc;    /* Vertical scaling factor.             */

    struct Flag    *constr;     /* Forces quadratic through the central */
                                /* cell of local window if selected.    */
```

```
    G_gisinit (argv[0]);          /* GRASS function which MUST be called  */
                                  /* first to check for valid database    */
                                  /* and mapset and prompt user for input.*/

    /*---------------------------------------------------------------------*/
    /*                        SET PARSER OPTIONS                           */
    /*---------------------------------------------------------------------*/

    rast_in   = G_define_option();      /* Request mem for each option. */
    rast_out  = G_define_option();
    tol1_val  = G_define_option();
    tol2_val  = G_define_option();
    win_size  = G_define_option();
    parameter = G_define_option();
    expon     = G_define_option();
    vert_sc   = G_define_option();

    constr    = G_define_flag();


    rast_in->key          = "in";
    rast_in->description  = "Raster surface layer to process";
    rast_in->type         = TYPE_STRING;
    rast_in->required     = YES;


    rast_out->key         = "out";
    rast_out->description = "Output raster containing morphometric parameter";
    rast_out->type        = TYPE_STRING;
    rast_out->required    = YES;

    tol1_val->key         = "s_tol";
    tol1_val->description = "Slope tolerance defining 'flat' surface (degs)";
    tol1_val->type        = TYPE_DOUBLE;
    tol1_val->required    = NO;
    tol1_val->answer      = "1.0";

    tol2_val->key         = "c_tol";
    tol2_val->description = "Curvature tol. that defines 'planar' surface";
    tol2_val->type        = TYPE_DOUBLE;
    tol2_val->required    = NO;
    tol2_val->answer      = "1.0";

    win_size->key         = "size";
    win_size->description = "Size of processing window (odd number only)";
    win_size->type        = TYPE_INTEGER;
    win_size->required    = NO;
    win_size->answer      = "3";

    parameter->key        = "param";
    parameter->description= "Morphometric parameter to calculate";
    parameter->type       = TYPE_STRING;
    parameter->required   = NO;
    parameter->options    =
         "elev,slope,aspect,profc,planc,longc,crosc,minic,maxic,feature";
```

```
    parameter->answer     = "elev";

    expon->key            = "exp";
    expon->description    = "Exponent for distance weighting (0.0-4.0)";
    expon->type           = TYPE_DOUBLE;
    expon->required       = NO;
    expon->answer         = "0.0";

    vert_sc->key          = "zscale";
    vert_sc->description  = "Vertical scaling factor";
    vert_sc->type         = TYPE_DOUBLE;
    vert_sc->required     = NO;
    vert_sc->answer       = "1.0";

    constr->key           = 'c';
    constr->description   = "Constrain model through central window cell";


    if (G_parser(argc,argv))            /* Performs the prompting for   */
        exit(-1);                       /* keyboard input.              */

    rast_in_name  = rast_in->answer;    /* Place contents into strings  */
    rast_out_name = rast_out->answer;
    wsize         = atoi(win_size->answer);
    constrained   = constr->answer;
    sscanf(expon->answer,"%lf",&exponent);
    sscanf(vert_sc->answer,"%lf",&zscale);
    sscanf(tol1_val->answer,"%lf",&slope_tol);
    sscanf(tol2_val->answer,"%lf",&curve_tol);

    if ((exponent<0.0) || (exponent >4.0))
        exponent = 0.0;

    if (zscale == 0.0)
        zscale = 1;

    if (!strcmp(parameter->answer,"elev"))
            mparam = ELEV;
    else
        if (!strcmp(parameter->answer,"slope"))
            mparam = SLOPE;
        else
            if (!strcmp(parameter->answer,"aspect"))
                mparam = ASPECT;
            else
                if (!strcmp(parameter->answer,"profc"))
                    mparam = PROFC;
                else
                    if (!strcmp(parameter->answer,"planc"))
                        mparam = PLANC;
                    else
                        if (!strcmp(parameter->answer,"crosc"))
                            mparam = CROSC;
                        else
                            if (!strcmp(parameter->answer,"longc"))
```

```
                        mparam = LONGC;
                    else
                        if (!strcmp(parameter->answer,"maxic"))
                            mparam = MAXIC;
                        else
                            if (!strcmp(parameter->answer,"minic"))
                                mparam = MINIC;
                            else
                                if (!strcmp(parameter->answer,
                                                        "feature"))
                                    mparam = FEATURE;
                                else
                                {
                                    G_warning("Morphometric parameter not
                                      recognised. Assuming `Elevation'");
                                    mparam  = ELEV;
                                }

/*--------------------------------------------------------------------*/
/*                  CHECK INPUT RASTER FILE EXISTS                    */
/*--------------------------------------------------------------------*/


if ((mapset_in=G_find_cell2(rast_in_name,""))==NULL)
{
    char err[256];
    sprintf(err,"Raster map [%s] not available.",rast_in_name);
    G_fatal_error(err);
}


/*--------------------------------------------------------------------*/
/*              CHECK OUTPUT RASTER FILE DOES NOT EXIST               */
/*--------------------------------------------------------------------*/

mapset_out = G_mapset();    /* Set output to current mapset.          */

if (G_legal_filename(rast_out_name)==NULL)
{
    char err[256];
    sprintf(err,"Illegal file name. Please try another.");
    G_fatal_error(err);
}
else
{
    if (G_find_cell2(rast_out_name,mapset_out) !=NULL)
    {
        char err[256];
        sprintf(err,
    "Raster map [%s] exists.\nPlease try another\n",rast_out_name);
        G_fatal_error(err);
    }
}
```

```
    /*--------------------------------------------------------------------*/
    /*              CHECK WINDOW SIZE IS NOT EVEN OR TOO LARGE             */
    /*--------------------------------------------------------------------*/

    if ( (wsize/2 != (wsize-1)/2) || (wsize > MAX_WSIZE) )
    {
        char err[256];
        sprintf(err,"Inappropriate window size (too big or even)");
        G_fatal_error(err);
    }

}
```

open_files.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                            open_files()                               ***/
/***                  Opens input and output raster files.                 ***/
/***               Jo Wood, Project ASSIST, 24th January 1993              ***/
/***                                                                       ***/
/*****************************************************************************/

#include "param.h"


open_files()
{
    /* Open existing file and set the input file descriptor. */

    if ( (fd_in=G_open_cell_old(rast_in_name,mapset_in)) <0)
    {
        char err[256];
        sprintf(err,"ERROR: Problem opening input file.");
        G_fatal_error(err);
    }

    /* Open new file and set the output file descriptor. */

    if ( (fd_out=G_open_cell_new(rast_out_name)) <0)
    {
        char err[256];
        sprintf(err,"ERROR: Problem opening output file.");
        G_fatal_error(err);
    }
}
```

find_normal.c

```
/*************************************************************/
/* find_normal() - Function to find the set of normal equations */
/*                 that allow a quadratic trend surface to be   */
/*                 fitted through N  points using least squares */
/*                 V.1.0, Jo Wood, 27th November, 1994.         */
/*************************************************************/

#include "param.h"

find_normal(normal,w)
    float **normal;             /* Matrix of cross-products.    */
    double *w;                  /* Weights matrix.              */
{

    int edge=EDGE;              /* Store (wsize-1)/2 to save    */
                                /* on computation               */

    float  x,y,                 /* Local coordinates of window. */
           x1=0,y1=0,           /* coefficients of X-products.  */
           x2=0,y2=0,
           x3=0,y3=0,
           x4=0,y4=0,
           xy2=0, x2y=0,
           xy3=0, x3y=0,
           x2y2=0,xy=0,
           N=0;

    int   row,col;              /* Pass through local window.   */


    /* Initialise sums-of-squares and cross products matrix */

    for (row=1; row<=6; row++)
        for (col=1; col<=6;col++)
            normal[row][col] = 0.0;


    /* Calculate matrix of sums of squares and cross products */

    for (row=0; row<wsize; row++)
        for (col=0; col<wsize; col++)
        {
           x = resoln*(col-EDGE);
           y = resoln*(row-EDGE);

           x4   += x*x*x*x* *(w + row*wsize + col);
           x2y2 += x*x*y*y* *(w + row*wsize + col);
           x3y  += x*x*x*y* *(w + row*wsize + col);
           x3   += x*x*x*   *(w + row*wsize + col);
           x2y  += x*x*y*   *(w + row*wsize + col);
           x2   += x*x*     *(w + row*wsize + col);
```

```
            y4   += y*y*y*y* *(w + row*wsize + col);
            xy3  += x*y*y*y* *(w + row*wsize + col);
            xy2  += x*y*y*   *(w + row*wsize + col);
            y3   += y*y*y*   *(w + row*wsize + col);
            y2   += y*y*     *(w + row*wsize + col);

            xy   += x*y*     *(w + row*wsize + col);

            x1   += x*       *(w + row*wsize + col);

            y1   += y*       *(w + row*wsize + col);

            N    +=          *(w + row*wsize + col);

        }

    /* --- Store cross-product matrix elements. ---*/

     normal[1][1] = x4;
     normal[1][2] = normal[2][1] = x2y2;
     normal[1][3] = normal[3][1] = x3y;
     normal[1][4] = normal[4][1] = x3;
     normal[1][5] = normal[5][1] = x2y;
     normal[1][6] = normal[6][1] = x2;

     normal[2][2] = y4;
     normal[2][3] = normal[3][2] = xy3;
     normal[2][4] = normal[4][2] = xy2;
     normal[2][5] = normal[5][2] = y3;
     normal[2][6] = normal[6][2] = y2;

     normal[3][3] = x2y2;
     normal[3][4] = normal[4][3] = x2y;
     normal[3][5] = normal[5][3] = xy2;
     normal[3][6] = normal[6][3] = xy;

     normal[4][4] = x2;
     normal[4][5] = normal[5][4] = xy;
     normal[4][6] = normal[6][4] = x1;

     normal[5][5] = y2;
     normal[5][6] = normal[6][5] = y1;

     normal[6][6] = N;
}


/*************************************************************/
/* find_obs() - Function to find the observed vector as part of */
/*              the set of normal equations for least squares.  */
/*              V.1.0, Jo Wood, 11th December, 1994.            */
/*************************************************************/

find_obs(z,obs,w)
    CELL  *z;                   /* Local window of elevs.       */
```

```
    float *obs;                 /* Observed column vector.      */
    double  *w;                 /* Weighting matrix.            */

{

    int row,col,                /* Counts through local window. */
        edge=EDGE,              /* EDGE = (wsize-1)/2.          */
        offset;                 /* Array offset for weights & z */

    float x,y;                  /* Local window coordinates.    */

    for (row=1;row<=6; row++)   /* Initialise column vector.    */
        obs[row] = 0.0;


    for (row=0;row<wsize; row++)
        for (col=0; col<wsize; col++)
        {
            x = resoln*(col-EDGE);
            y = resoln*(row-EDGE);
            offset = row*wsize + col;

            obs[1] += *(w + offset) * *(z + offset) * x*x ;
            obs[2] += *(w + offset) * *(z + offset) * y*y ;
            obs[3] += *(w + offset) * *(z + offset) * x*y ;
            obs[4] += *(w + offset) * *(z + offset) * x ;
            obs[5] += *(w + offset) * *(z + offset) * y ;

            if (!constrained)   /* If constrained, should remain 0.0 */
                obs[6] += *(w + offset) * *(z + offset) ;
        }
}


/*****************************************************************/
/* find_weight() Function to find the weightings matrix for the */
/*               observed cell values.                           */
/*               Uses an inverse distance function that can be  */
/*               calibrated with an exponent (0= no decay,      */
/*               1=linear decay, 2=squared distance decay etc.) */
/*               V.1.1, Jo Wood, 11th May, 1995.                */
/*****************************************************************/

find_weight(weight_ptr)
    double *weight_ptr;
{
    int row,col;                /* Counts through the rows and  */
                                /* columns of weights matrix.   */

    double dist,                /* Distance to centre of kernel.*/
           sum_weight=0;        /* Sum of weights.              */


    /* --- Find inverse distance of all cells to centre. ---*/
```

```
    for (row=0; row<wsize; row++)
        for (col=0; col<wsize; col++)
        {
            dist = 1.0 /
                pow(sqrt((EDGE-col)*(EDGE-col) +
                                        (EDGE-row)*(EDGE-row))+1.0,exponent);
            *(weight_ptr + row*wsize + col) = dist;
        }
}
```

lubacksub.c

```
/*************************************************************/
/* lubksub() - Function to perform forward and back substitution*/
/*             on an LU decomposed matrix. Used for solving a  */
/*             set of linear equations.                        */
/*             Adapted from Press et al (1988), p.44.          */
/*             V.1.0, Jo Wood, 10th December, 1994.            */
/*************************************************************/


lubksub(a,n,index,b)
    int     n,          /* Size of side of matrix               */
            *index;     /* Row permutation effected by pivoting */
    float   **a,        /* Matrix to be decomposed.             */
            *b;         /* Input as RHS vector, output as soln. */

{
    int    i, ii=0, ip, j;
    float  sum;

    for (i=1; i<=n; i++)
    {
        ip    = index[i];
        sum   = b[ip];
        b[ip] = b[i];

        if (ii)
            for (j=ii; j<=i-1; j++)
                sum -= a[i][j] * b[j];
        else
            if (sum)
                ii=i;

        b[i] = sum;
    }

    for (i=n; i>=1; i--)
    {
        sum = b[i];
        for (j=i+1; j<=n; j++)
            sum -= a[i][j] * b[j];
```

```
        b[i] = sum/a[i][i];
    }
}
```

ludcomp.c

```
/*************************************************************/
/* ludcomp() - Function to perform LU decomposition of a matrix.*/
/*             Used to find inverse of the matrix when solving */
/*             sets of linear equations.                       */
/*             Adapted from Press et al (1988), pp.43-45.      */
/*             V.1.0, Jo Wood, 9th December, 1994.             */
/*************************************************************/

#include "param.h"
#include <math.h>


ludcomp(a,n,index)
    int     n,          /* Size of side of matrix               */
            *index;     /* Row permutation effected by pivoting */
    float   **a;        /* Matrix to be decomposed.             */

{
    int     i, imax,j,k;
    float   big,dum,sum,temp,
            *vv;


    vv = vector(1,n);

    for (i=1; i<=n; i++)    /* Loop over rows, get scaling info.*/
    {
        big = 0.0;
        for (j=1; j<=n; j++)
            if ((temp=fabs(a[i][j])) > big)
                big = temp;

        if (big == 0.0)
        {
            fprintf(stderr,"Singular matrix - can't perform LU decomposition\n");
            exit(-1);
        }

        vv[i] = 1.0/big;    /* Save the scaling. */
    }


    for (j=1; j<=n; j++)
    {
        for (i=1; i<j; i++)
        {
            sum=a[i][j];
            for (k=1; k<i; k++)
```

```
                sum -= a[i][k] * a[k][j];
            a[i][j] = sum;
        }

        big=0.0;

        for (i=j; i<=n; i++)
        {
            sum=a[i][j];
            for (k=1; k<j; k++)
                sum -= a[i][k] * a[k][j];
            a[i][j] = sum;

            if ( (dum=vv[i]*fabs(sum)) >= big)
            {
                big = dum;
                imax = i;
            }
        }

        if (j != imax)
        {
            for (k=1; k<=n; k++)
            {
                dum = a[imax][k];
                a[imax][k] = a[j][k];
                a[j][k] = dum;
            }
            vv[imax] = vv[j];
        }

        index[j] = imax;
        if (a[j][j] == 0.0)
            a[j][j] = TINY;

        if (j != n)
        {
            dum = 1.0/(a[j][j]);

            for (i=j+1; i<=n; i++)
                a[i][j] *= dum;
        }
    }

    free_vector(vv,1,n);
}
```

nrutils.c

```
/*************************************************************/
/* nrutil.c     Functions to reserve memory fo rthe storage of  */
/*              matrices and vectors. From Press et al (1988)   */
/*              See Appendix D pp.705-709.                      */
/*              V.1.0, Jo Wood, 9th December, 1994.             */
/*************************************************************/
```

```
#include <stdio.h>

/*************************************************************/
/* Set up a 1D floating point matrix with range [nl to nh]   */
/* See Press et al (1988) p.705.                             */
/*************************************************************/

float *vector(nl,nh)
    int nl,nh;
{
     float *v;

     v = (float *) malloc((unsigned) (nh-nl+1)*sizeof(float));

     if (!v)
     {
        fprintf(stderr,"ERROR: Can't allocate memory for vector\n");
        exit(-1);
     }

     return (v-nl);
}


/*************************************************************/
/* Set up a 1D integer matrix with range [nl to nh]          */
/* See Press et al (1988) p.706.                             */
/*************************************************************/

int *ivector(nl,nh)
    int nl,nh;
{
     int *v;

     v = (int *) malloc((unsigned) (nh-nl+1)*sizeof(int));

     if (!v)
     {
        fprintf(stderr,"ERROR: Can't allocate memory for vector\n");
        exit(-1);
     }

     return (v-nl);
}


/*************************************************************/
/* Set up a 2D floating point matrix                         */
/* with range [nrl to nrh][ncl to nch]                       */
/* See Press et al (1988) p.706.                             */
/*************************************************************/

float **matrix(nrl,nrh,ncl,nch)
    int nrl,nrh,ncl,nch;
{
```

```
    int     i;
    float **m;

    m = (float **) malloc((unsigned) (nch-ncl+1)*sizeof(float));

    if (!m)
    {
        fprintf(stderr,"ERROR: Can't allocate memory for matrix rows\n");
        exit(-1);
    }

    for (i=nrl; i<=nrh; i++)
    {
        m[i] = (float *) malloc((unsigned) (nch-ncl+1)*sizeof(float));

        if (!m[i])
        {
            fprintf(stderr,"ERROR: Can't allocate memory for matrix columns\n");
            exit(-1);
        }
        m[i] -= ncl;
    }

    return (m);
}


/*************************************************************/
/* Free memory used for storing 1D floating point matrix.    */
/* See Press et al (1988) p.707.                             */
/*************************************************************/

free_vector(v,nl,nh)
    float *v;
    int   nl,nh;
{
    free((char *) (v+nl));
}


/*************************************************************/
/* Free memory used for storing 1D integer matrix.           */
/* See Press et al (1988) p.707.                             */
/*************************************************************/

free_ivector(v,nl,nh)
    int *v,
         nl,nh;
{
    free((char *) (v+nl));
}
```

```
/*************************************************************/
/* Free memory used for storing 2D floating point matrix.    */
/* See Press et al (1988) p.708.                             */
/*************************************************************/

free_matrix(m,nrl,nrh,ncl,nch)
    float **m;
    int   nrl,nrh,ncl,nch;
{
    int i;

    for (i=nrh; i>=nrl; i--)
        free ((char *) (m[i]+ncl));

    free((char *) (m+nrl));
}
```

process.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                            process()                                  ***/
/***          Reads in a raster file row by row for processing.            ***/
/***      Jo Wood, Department of Geography, V1.0 7th February 1993         ***/
/***                                                                       ***/
/*****************************************************************************/

#include "param.h"

process()
{

    /*-------------------------------------------------------------*/
    /*                          INITIALISE                         */
    /*-------------------------------------------------------------*/


    CELL        *row_in,        /* Buffer large enough to hold `wsize'  */
                *row_out,       /* raster rows. When GRASS reads in a   */
                                /* raster row, each element is of type  */
                                /* CELL (probably of type int).         */

                *window_ptr,    /* Stores local terrain window.         */

                centre;         /* Elevation of central cell in window. */

    struct Cell_head region;    /* Structure to hold region information */
    struct Categories cats;     /* Category file structure for raster   */

    int         nrows,          /* Will store the current number of     */
                ncols,          /* rows and columns in the raster.      */
```

```
            row,col,        /* Counts through each row and column   */
                            /* of the input raster.                 */

            wind_row,       /* Counts through each row and column   */
            wind_col,       /* of the local neighbourhood window.   */

            row_mem,        /* Memory to hold one raster row.       */

            *index_ptr;     /* Row permutation vector for LU decomp.*/

float       **normal_ptr,   /* Cross-products matrix.               */
            *obs_ptr;       /* Observed vector.                     */

double      *weight_ptr;    /* Weighting matrix for observed values.*/


/*--------------------------------------------------------------------*/
/*                    GET RASTER AND WINDOW DETAILS                   */
/*--------------------------------------------------------------------*/

G_get_window(&region);      /* Fill out the region structure (the   */
                            /* geographical limits etc.)            */

nrows = G_window_rows();    /* Find out the number of rows and      */
ncols = G_window_cols();    /* columns of the raster.               */


if ((region.ew_res/region.ns_res >= 1.01) || /* If EW and NS res are*/
    (region.ns_res/region.ew_res >= 1.01))   /* >1% diff, warn user.*/
{
    G_warning("E-W and N-S grid resolutions are different. Taking average.");
    resoln = (region.ns_res + region.ew_res)/2;
}
else
    resoln = region.ns_res;



/*--------------------------------------------------------------------*/
/*            RESERVE MEMORY TO HOLD Z VALUES AND MATRICES            */
/*--------------------------------------------------------------------*/


row_in = (CELL *) G_malloc(ncols*sizeof(CELL)*wsize);
                            /* Reserve `wsize' rows of memory.      */

row_out = G_allocate_cell_buf();    /* Initialise output row buffer. */

window_ptr = (CELL *) G_malloc(wsize*wsize*sizeof(CELL));
                            /* Reserve enough memory for local wind.*/

weight_ptr = (double *) G_malloc(wsize*wsize*sizeof(double));
                            /* Reserve enough memory weights matrix.*/

normal_ptr = matrix(1,6,1,6);       /* Allocate mem for 6*6 matrix  */
```

```
    index_ptr  = ivector(1,6);  /* and for 1D vector holding indices    */
    obs_ptr    = vector(1,6);   /* and for 1D vector holding observed z */


    /* ------------------------------------------------------------- */
    /* -            CALCULATE LEAST SQUARES COEFFICIENTS           - */
    /* ------------------------------------------------------------- */

    /*--- Calculate weighting matrix. ---*/

    find_weight(weight_ptr);

    /* Initial coefficients need only be found once since they are
       constant for any given window size. The only element that
       changes is the observed vector (RHS of normal equations). */

    /*--- Find normal equations in matrix form. ---*/

    find_normal(normal_ptr,weight_ptr);


    /*--- Apply LU decomposition to normal equations. ---*/

    if (constrained)
        ludcomp(normal_ptr,5,index_ptr);   /* To constrain the quadtratic
                                              through the central cell, ignore
                                              the calculations involving the
                                              coefficient f. Since these are
                                              all in the last row and column of
                                              the matrix, simply redimension. */
    else
        ludcomp(normal_ptr,6,index_ptr);

    /*-------------------------------------------------------------*/
    /*      PROCESS INPUT RASTER AND WRITE OUT RASTER LINE BY LINE      */
    /*-------------------------------------------------------------*/

    G_zero_cell_buf(row_out);
    for (wind_row=0; wind_row<EDGE; wind_row++)
        G_put_map_row(fd_out,row_out);  /* Write out edge cells as zeros. */

    for (wind_row=0; wind_row<wsize-1; wind_row++)
        G_get_map_row(fd_in,row_in+(wind_row*ncols),wind_row);
                                        /* Read in enough of the 1st rows */
                                        /* to allow window to be examined.*/

    for (row=EDGE; row<(nrows-EDGE); row++)
    {
        G_percent(row+1,nrows-EDGE,2);

        G_get_map_row(fd_in,row_in+((wsize-1)*ncols),row+EDGE);

        for (col=EDGE; col<(ncols-EDGE); col++)
        {
                                        /* Find central z value */
```

```
            centre = *(row_in + EDGE*ncols + col);

            for (wind_row=0; wind_row<wsize; wind_row++)
                for (wind_col=0; wind_col<wsize; wind_col++)
                                        /* Express window values relative */
                                        /* to the central elevation.      */
                    *(window_ptr+(wind_row*wsize)+wind_col) =
                        *(row_in+(wind_row*ncols)+col+wind_col-EDGE) - centre;

            /*--- Use LU back substitution to solve normal equations. ---*/

            find_obs(window_ptr,obs_ptr,weight_ptr);

            if (constrained)
                lubksub(normal_ptr,5,index_ptr,obs_ptr);
            else
                lubksub(normal_ptr,6,index_ptr,obs_ptr);

            /*--- Calculate terrain parameter based on quad. coefficients. ---*/

            if (mparam == FEATURE)
                *(row_out+col) = feature(obs_ptr);
            else
                *(row_out+col) = param(mparam,obs_ptr);

            if (mparam == ELEV)
                *(row_out+col) += centre;       /* Add central elevation back */

        }
        G_put_map_row(fd_out,row_out);          /* Write the row buffer to    */
                                                /* the utput raster.          */

                                /* 'Shuffle' rows down one, and read in */
                                /*  one new row.                        */
        for (wind_row=0;wind_row<wsize-1;wind_row++)
           for (col=0; col<ncols; col++)
            *(row_in+(wind_row*ncols)+col) = *(row_in+((wind_row+1)*ncols)+col);
    }

    G_zero_cell_buf(row_out);
    for (wind_row=0; wind_row<EDGE; wind_row++)
        G_put_map_row(fd_out,row_out);  /* Write out edge cells as zeros.   */

    /*--------------------------------------------------------------------*/
    /*  FREE MEMORY USED TO STORE RASTER ROWS, LOCAL WINDOW AND MATRICES  */
    /*--------------------------------------------------------------------*/

    free(row_in);
    free(row_out);
    free(window_ptr);
    free_matrix(normal_ptr,1,6,1,6);
    free_vector(obs_ptr,1,6);
    free_ivector(index_ptr,1,6);
}
```

param.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                              param()                                  ***/
/***     Returns a terrain parameter based on the 6 quadratic coefficents  ***/
/***     that define a local trend surface.                                ***/
/***     Jo Wood, Department of Geography, V2.0 15th December, 1994        ***/
/***                                                                       ***/
/*****************************************************************************/

#include "param.h"
#include <math.h>


CELL param(ptype,coeff)
    int ptype;                  /* Type of terrain parameter to calculate */
    float *coeff;               /* Set of six quadratic coefficents.      */

{

    /* Quadratic function in the form of

                z = ax^2 + by^2 + cxy + dx + ey +f                          */

    double      a=C_A*zscale,           /* Rescale coefficients if a      */
                b=C_B*zscale,           /* Z scaling is required.         */
                c=C_C*zscale,
                d=C_D*zscale,
                e=C_E*zscale,
                f=C_F;                  /* f does not need rescaling as   */
                                        /* it is only used for smoothing. */

    switch(ptype)
    {
        case ELEV:
                return((CELL)rint(f));
                break;

        case SLOPE:
                return((CELL)rint(atan(sqrt(d*d + e*e))*RAD2DEG));
                break;

        case ASPECT:
                return((CELL)rint(atan2(e,d)*RAD2DEG));
                break;

        case PROFC:
                if ((d == 0) && (e == 0))
                  return((CELL)0);
                else
                  return((CELL)rint(-200.0*resoln*wsize*(a*d*d + b*e*e + c*e*d)/
                                    ((e*e + d*d) * pow(1.0 + d*d + e*e,1.5)) ));
                break;
```

```
        case PLANC:
                if ((d == 0) && (e == 0))
                  return((CELL)0);
                else
                  return((CELL)rint(200.0*resoln*wsize*(b*d*d + a*e*e - c*d*e) /
                                    powf(e*e + d*d,1.5) ));
                break;

        case LONGC:
                if ((d == 0) && (e ==0))
                    return((CELL) 0);
                else
                    return((CELL)rint(-20.0*resoln*wsize*(a*d*d + b*e*e + c*d*e)/
                                                        (d*d + e*e)));
        case CROSC:
                if ((d == 0) && (e ==0))
                    return((CELL) 0);
                else
                    return((CELL)rint(-20.0*resoln*wsize*(b*d*d + a*e*e - c*d*e)/
                                                        (d*d + e*e)));

        case MINIC:
                return((CELL)rint
                        (20.0*resoln*wsize*(-a-b-sqrt((a-b)*(a-b)+c*c))));

        case MAXIC:
                return((CELL)rint
                        (20.0*resoln*wsize*(-a-b+sqrt((a-b)*(a-b) + c*c))));

        default:
                return((CELL)0);
    }
}
```

feature.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                            feature()                                  ***/
/***     Returns a terrain feature based on the 6 quadratic coefficents    ***/
/***     that define a local trend surface.                                ***/
/***     Jo Wood, Department of Geography, V2.1 30th March, 1995           ***/
/***                                                                       ***/
/*****************************************************************************/

#include "param.h"
#include <math.h>


CELL feature(coeff)
    float *coeff;               /* Set of six quadratic coefficents.    */

{
```

```
/* Quadratic function in the form of

            z = ax^2 + by^2 + cxy + dx + ey +f                    */

double      a=C_A*zscale,    /* Scale parameters if necessary.     */
            b=C_B*zscale,
            c=C_C*zscale,
            d=C_D*zscale,
            e=C_E*zscale,
            f=C_F*zscale;

double maxic,minic,          /* Minimium and maximum curvature.    */
       slope,                /* Slope.                             */
       crosc;                /* Cross-sectional curvature.         */

minic = 20*wsize*resoln*(-a-b-sqrt((a-b)*(a-b) + c*c));
maxic = 20*wsize*resoln*(-a-b+sqrt((a-b)*(a-b) + c*c));
slope = RAD2DEG*atan(sqrtf((d*d) + (e*e)));
crosc = -20*wsize*resoln*(b*d*d + a*e*e - c*d*e)/(d*d + e*e);


/* Case 1: Surface is sloping. Cannot be a peak,pass or pit. Therefore
           calculate the cross-sectional curvature to characterise as
           channel, ridge or planar.                                  */

if (slope > slope_tol)
{
    if (crosc > curve_tol)
        return(RIDGE);

    if (crosc < -curve_tol)
        return(CHANNEL);

    else
        return(FLAT);
}


/* Case 2: Surface has (approximately) vertical slope normal. Feature
           can be of any type.                                        */

if (maxic > curve_tol)
{
    if (minic > curve_tol)
        return(PEAK);

    if (minic < -curve_tol)
        return(PASS);

    else
        return (RIDGE);
}
else
    if (minic < -curve_tol)
    {
```

```
            if (maxic < -curve_tol)
                return (PIT);

            else
                return (CHANNEL);
        }

    return (FLAT);
}
```

write_cats.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                            write_cats()                               ***/
/***            Writes out category file for morphometric features         ***/
/***               Jo Wood, Project ASSIST, 7th February 1995              ***/
/***                                                                       ***/
/*****************************************************************************/

#include "param.h"

write_cats()
{
    /*-------------------------------------------------------------------------*/
    /*                             INITIALISE                                  */
    /*-------------------------------------------------------------------------*/

    struct Categories   cats;

    G_init_cats(NUM_CATS,"Surface Features",&cats);

    /*-------------------------------------------------------------------------*/
    /*                      FILL OUT CATEGORIES STRUCTURE                      */
    /*-------------------------------------------------------------------------*/


    G_set_cat(PIT,     " Pit",           &cats);
    G_set_cat(PEAK,    " Peak",          &cats);
    G_set_cat(RIDGE,   " Ridge",         &cats);
    G_set_cat(CHANNEL," Channel",        &cats);
    G_set_cat(PASS,    " Pass (saddle)",&cats);
    G_set_cat(FLAT,    " Planar",        &cats);


    /*-------------------------------------------------------------------------*/
    /*                     WRITE OUT CATEGORIES STRUCTURE                      */
    /*-------------------------------------------------------------------------*/

    if (G_write_cats(rast_out_name,&cats) <=0)
    {
        char warn[255];
        sprintf(warn,"Can't write category file for <%s>",rast_out_name);
```

```
        G_warning(warn);
    }

    G_free_cats(&cats);

}
```

write_cols.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                          write_cols()                                 ***/
/***             Writes out colour file for morphometric features          ***/
/***              Jo Wood, Dept. of Geography, 21st February 1995          ***/
/***                                                                       ***/
/*****************************************************************************/

#include "param.h"

write_cols()
{
    /*---------------------------------------------------------------------*/
    /*                           INITIALISE                                */
    /*---------------------------------------------------------------------*/

    struct Colors       colours;

    G_init_colors(&colours);

    /*---------------------------------------------------------------------*/
    /*                      FILL OUT COLORS STRUCTURE                      */
    /*---------------------------------------------------------------------*/

    G_add_color_rule(   FLAT,    255,255,255,                   /* White      */
                        PIT,     0,  0,  0,  &colours);         /* Black      */
    G_add_color_rule(   CHANNEL,0,   0,  255,                   /* Blue       */
                        PASS,    0,  255,0,  &colours);         /* Green      */
    G_add_color_rule(   RIDGE,   255,255,0,                     /* Yellow     */
                        PEAK,    255,0,  0,  &colours);         /* Red        */


    /*---------------------------------------------------------------------*/
    /*                      WRITE OUT COLORS STRUCTURE                     */
    /*---------------------------------------------------------------------*/

    G_write_colors(rast_out_name,mapset_out,&colours);

    G_free_colors(&colours);

}
```

close_down.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                            close_down()                               ***/
/***       Closes all input and output raster files and frees memory.      ***/
/***           Jo Wood, Dept. of Geography, 7th February 1993              ***/
/***                                                                       ***/
/*****************************************************************************/

#include "param.h"


close_down()
{
    /* Close connection with existing input raster. */

    G_unopen_cell(fd_in);

    /* Write output raster file and close connection. */

    G_close_cell(fd_out);
}
```

v.surf.spline

GRASS module to interpolate vector contour data by fitting a cubic spline function to profiles in four directions. It has two advantages over other interpolation techniques: (1) The RMS Error can be calculated for EACH CELL in the DEM as well as an overall RMSE. Spatial variation in error can examined; (2) A cubic spline has the property of 2nd derivative being continuous therefore having implications for slope measurements. Terracing effects should be minimised.

Jo Wood, 29th November 1991

Modified 23rd July 1995 to conform to standard GRASS module layout (main.c, interface.c, process.c etc.)

Modified July, 1995 to incorporate 'addnode' spline constraint.

main.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                            v.surf.spline                              ***/
/*** GRASS module to interpolate vector contour data using constrained  ***/
/*** spline fitting. Can be used to calculate RMS error for each of the ***/
/*** interpolated cells (Yeoli, 1986).                                   ***/
/***                                                                       ***/
/*** Jo Wood, Department of Geography, V1.0 5th December, 1991           ***/
/*** V2.0 Modified to conform to GRASS module structure, 23rd July 1995 ***/
/***                                                                       ***/
/*****************************************************************************/

#define MAIN

#include "spline.h"

main(argc,argv)
    int argc;
    char *argv[];
{
    /*---------------------------------------------------------------*/
    /*                          INITIALISE                           */
    /*---------------------------------------------------------------*/

    struct Map_info     vmap;           /* Vector map information.      */
    interface(argc,argv);               /* Parse command line input.    */

    /*---------------------------------------------------------------*/
    /*                          OPEN FILES                           */
    /*---------------------------------------------------------------*/

    open_vect(vmap);
```

```
    /*-------------------------------------------------------------------*/
    /*                  INTERPOLATE VECTOR CONTOURS                      */
    /*-------------------------------------------------------------------*/

    process();
}
```

interface.c

```
/*****************************************************************************/
/***                           interface()                                 ***/
/***  Parses input from keyboard and checks files exist/don't exist.       ***/
/***  V1.2 23rd July, 1995.                                                ***/
/***                                                                       ***/
/*****************************************************************************/

#include "spline.h"

interface(argc,argv)
    int argc;
    char *argv[];
{
    /*-------------------------------------------------------------------*/
    /*                           INITIALISE                              */
    /*-------------------------------------------------------------------*/

    struct Option  *vect_in,    /* Structures required for G_parser()    */
                   *rast_out,
                   *interv;     /* Contour interval (for constrained     */
                                /* spline fitting.                       */
    struct Flag    *rooks_case, /* Select rooks case rasterising.        */
                   *simple_con; /* Select simple spline consstraint.     */

    G_gisinit (argv[0]);        /* Initialise GRASS module.              */


    /*-------------------------------------------------------------------*/
    /*                       SET PARSER OPTIONS                          */
    /*-------------------------------------------------------------------*/

    vect_in  = G_define_option();       /* Pointer for each option.      */
    rast_out = G_define_option();
    interv   = G_define_option();

    rooks_case= G_define_flag();
    simple_con= G_define_flag();

    vect_in->key                = "in";
    vect_in->description        = "vector contour map to be interpolated";
    vect_in->type               = TYPE_STRING;
    vect_in->required           = YES;
```

```
    rast_out->key               = "out";
    rast_out->description       = "Resultant Digital Elevation Model";
    rast_out->type              = TYPE_STRING;
    rast_out->required          = YES;

    interv->key                 = "interval";
    interv->description         = "Contour interval (0 for no interval
                                   constraint)";
    interv->type                = TYPE_INTEGER;
    interv->answer              = "0";

    rooks_case->key             = 'r';
    rooks_case->description     = "Rasterise contours using rooks case
                                   adjacency";

    simple_con->key             = 's';
    simple_con->description     = "Constrain interpolation using simple
                                   truncation";

    if (G_parser(argc,argv))
        exit(-1);               /* Returns a 0 if sucessful            */

    vect_in_name  = vect_in->answer;
    rast_out_name = rast_out->answer;
    interval      = atoi(interv->answer);
    truncation    = simple_con->answer;
    rooks         = rooks_case->answer;

    /*--------------------------------------------------------------------*/
    /*                     CHECK INPUT VECTOR FILE EXISTS                 */
    /*--------------------------------------------------------------------*/

    if ((mapset_in=G_find_vector(vect_in_name,""))==NULL)
    {
        char err[256];
        sprintf(err,"Vector map [%s] not available.",vect_in_name);
        G_fatal_error(err);
    }

    /*--------------------------------------------------------------------*/
    /*             CHECK THE OUTPUT RASTER DOES NOT ALREADY EXIST         */
    /*--------------------------------------------------------------------*/

    mapset_out = G_mapset();            /* Set output to current mapset.*/

    if (G_legal_filename(rast_out)==NULL)
    {
        char err[256];
        sprintf(err,"Illegal file name. Please try another.");
        G_fatal_error(err);
    }
    else
    {
        if (G_find_cell(rast_out_name,mapset_out) !=NULL)
        {
```

```
            char err[256];
            sprintf(err,"Raster map [%s] already exists.\nPlease try another.",
                                                              rast_out_name);
            G_fatal_error(err);
        }
    }
}
```

open_vect.c

```
/*****************************************************************************/
/***                              open_vect()                              ***/
/***            Opens vector file and checks that it has topology.         ***/
/***                                                                       ***/
/*** Jo Wood, Department of Geography, V1.0 5th December, 1991             ***/
/*** V2.0 Modified to conform to GRASS module structure, 23rd July 1995    ***/
/***                                                                       ***/
/*****************************************************************************/

#include "spline.h"

open_vect(vmap)
    struct Map_info vmap;       /* Stores vector information.            */
{
    if (Vect_open_old(&vmap,vect_in_name,mapset_in)<2)
    {
        char err[256];
        sprintf(err,"Vector file %s needs topology building.\nRun v.support.",
                vect_in_name);
        G_fatal_error(err);
    }
    Vect_close(&vmap);
}
```

process.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                                process()                              ***/
/*** Function to process vector for spline interpolation.                  ***/
/*** Jo Wood, Department of Geography, V1.0 December 1991                  ***/
/*** Modified for rooks-case rasterisation, V1.2, 7th March, 1992          ***/
/*** Modified for standard GRASS module structure, V2.0, 23rd July 1995    ***/
/***                                                                       ***/
/*****************************************************************************/

#include "spline.h"

process()
{
```

```
/*--------------------------------------------------------------------*/
/*                          INITIALISE                                */
/*--------------------------------------------------------------------*/

char        cmd[127],               /* System command buffer.        */
            rcont_name[127],        /* Rasterized contour file.      */
            temp[127],              /* Temporary file name.          */
            *rmapset;               /* Raster mapset.                */

int         new_ncols,
            profile,
            nprofiles=4,
            fd_new;

nrows = G_window_rows();
ncols = G_window_cols();

/*--------------------------------------------------------------------*/
/*            RASTERIZE CONTOURS USING GRASS FUNCTION                 */
/*--------------------------------------------------------------------*/

/* Create temporary rasterized contour layer */
strcpy(rcont_name,rast_out_name);
strcat(rcont_name,".contours");

printf("\n\nSTEP 1 - Rasterizing vectors\n         ");
sprintf(cmd,"v.to.rast input=%s output=%s",vect_in_name,rcont_name);
system(cmd);

if (rooks)                  /* rasterise using rooks case adjacency */
{
    sprintf(cmd,"rooks.sh %s",rcont_name);
    system(cmd);
}


/*--------------------------------------------------------------------*/
/*                  INTERPOLATE AND ROTATE CONTOURS                   */
/*--------------------------------------------------------------------*/


/* 1. INTERPOLATE ALONG EDGES AND STORE RASTERIZED CONTOURS */
interpol_edges(rcont_name);


/* 2. INTERPOLATE AND STORE PROFILES */
for (profile=1;profile<=nprofiles;profile++)
    interpol_layer(rast_out_name,rcont_name,profile);


/* 3. CALCULATE FINAL INTERPOLATED MAP AND RMSE SURFACE */
printf("\n\nSTEP 4 -
           Calculating weighted average of surfaces and RMSE \n\t");
sprintf(cmd,"rmse.sh %s",rast_out_name);
system(cmd);
```

```
    /* 4. REMOVE TEMPORARY FILES */
    sprintf(cmd,"tidy.sh %s",rast_out_name);
    system(cmd);

}
```

interpol_edges.c

```
/*****************************************************************************/
/***                        interpol_edges()                               ***/
/*** Function to interpolate along raster contour edges.                    ***/
/*** Jo Wood, V1.0, December, 1991, V1.2 March 7th, 1992                    ***/
/*** V2.0 Modified to conform to GRASS module strcture, 23rd July, 1995     ***/
/***                                                                       ***/
/*****************************************************************************/

#include "spline.h"

interpol_edges(rcont_name)

char    *rcont_name;
{
    /*-------------------------------------------------------------------*/
    /*                          INITIALISE                               */
    /*-------------------------------------------------------------------*/

    char                rcont_temp[127],
                        cmd[127],
                        *rmapset;

    int                 row,col,
                        fd_in,
                        fd_out;

    CELL                *cont_row,
                        *weight_row,
                        *weight_col,
                        *cont_west,
                        *cont_east;

    printf("\nSTEP 2 - Calculating splines for edges of contour map      ");

    /*-------------------------------------------------------------------*/
    /*      OPEN RASTER FILE CONTAING CONTOURS FOR INPUT AND OUTPUT      */
    /*-------------------------------------------------------------------*/

    if ((rmapset=G_find_cell(rcont_name,"")) == NULL)
    {
        char err[256];
        sprintf(err,"Rasterized contours [%s] could not be opened\n",rcont_name);
        G_fatal_error(err);
    }

    sprintf(rcont_temp,"%s.temp",rcont_name);
```

```
fd_in    = G_open_cell_old(rcont_name,rmapset);
fd_out   = G_open_cell_new(rcont_temp,rmapset);


/*--------------------------------------------------------------------*/
/*     ALLOCATE MEMORY FOR BUFFERS HOLDING EDGE INFORMATION           */
/*--------------------------------------------------------------------*/

cont_row    = G_allocate_cell_buf();
cont_west   = (CELL *) malloc(nrows*sizeof(CELL));
cont_east   = (CELL *) malloc(nrows*sizeof(CELL));
weight_row  = G_allocate_cell_buf();
weight_col  = (CELL *) malloc(nrows*sizeof(CELL));


/*--------------------------------------------------------------------*/
/*                READ IN BOUNDARY CONTOUR INFORMATION                */
/*--------------------------------------------------------------------*/

for (row=0; row<nrows; row++)
{
    G_get_map_row(fd_in,cont_row,row);

    *(cont_west+row) = *(cont_row);
    *(cont_east+row) = *(cont_row+ncols-1);
}

inter_spline(nrows,cont_west,weight_col,3);
inter_spline(nrows,cont_east,weight_col,3);


/*--------------------------------------------------------------------*/
/*             STORE CONTOURS AND INTERPOLATED EDGES                  */
/*--------------------------------------------------------------------*/

G_get_map_row(fd_in,cont_row,0);            /* North edge.            */
inter_spline(ncols,cont_row,weight_row,1);

                                            /* Corners are average    */
                                            /* of intersecting edge   */
                                            /* profiles.              */
*cont_row = (*cont_row + *cont_west)/2;
*(cont_row + ncols-1) = (*(cont_row + ncols-1) + *cont_east)/2;


G_put_map_row(fd_out,cont_row);

for (row=1;row<nrows-1;row++)               /* Most of raster.        */
{
    G_get_map_row(fd_in,cont_row,row);
    *(cont_row)             = *(cont_west+row);
    *(cont_row + ncols-1)   = *(cont_east+row);
    G_put_map_row(fd_out,cont_row);
}
```

```
    G_get_map_row(fd_in,cont_row,nrows-1);      /* South edge.          */
    inter_spline(ncols,cont_row,weight_row,1);
                                                /* Corners are average  */
                                                /* of intersecting edge */
                                                /* profiles.            */
    *cont_row = (*cont_row + *(cont_west+nrows-1))/2;
    *(cont_row + ncols-1) = (*(cont_row + ncols-1) + *(cont_east+nrows-1))/2;

    G_put_map_row(fd_out,cont_row);


    /*---------------------------------------------------------------*/
    /*                          CLOSE DOWN                           */
    /*---------------------------------------------------------------*/

    G_close_cell(fd_in);
    G_close_cell(fd_out);

    sprintf(cmd,"g.remove %s\n",rcont_name);
    system(cmd);
    sprintf(cmd,"g.rename %s,%s\n",rcont_temp,rcont_name);
    system(cmd);

    free(cont_row);
    free(weight_row);
    free(weight_col);
    free(cont_west);
    free(cont_east);
}
```

interpol_layer.c

```
/*************************************************************************/
/***                      interpol_layer()                             ***/
/*** Function to read in and interpolate between rasterized contours    ***/
/*** Jo Wood, V1.0, December, 1991, V1.2 March 7th, 1992                ***/
/*** V2.0 Modified to conform to GRASS module strcture, 23rd July, 1995 ***/
/*** V2.1 Modified to allow any profile angle, 28th July, 1995.         ***/
/***                                                                   ***/
/*************************************************************************/

#include "spline.h"

interpol_layer(dem_name,rcont_name,profile)

char    *dem_name,              /* Name of DEM and rasterised contours. */
        *rcont_name;
int     profile;                /* Profile direction code.              */
{
    /*---------------------------------------------------------------*/
    /*                          INITIALISE                           */
    /*---------------------------------------------------------------*/
```

```
char        profile_name[127],      /* Names of files storing DEM   */
            weight_name[127];       /* profiles and weights.        */

int         row,col,                /* Counts through raster cells. */
            fd_rcont,               /* Rasterised contour file desc.*/
            fd_dem,                 /* DEM and weights profile file */
            fd_weight,              /* desciptor.                   */
            ncells;                 /* Number of cells in profile.  */

CELL        *dem_prof,              /* Stores oblique profiles.     */
            *weight_prof,
            *dem_ptr,               /* Stores entire rasters.       */
            *weight_ptr;

/*--------------------------------------------------------------------*/
/*                  OPEN RASTER FILE CONTAING CONTOURS                */
/*--------------------------------------------------------------------*/

if ((mapset_in=G_find_cell(rcont_name,"")) == NULL)
{
    char err[256];
    sprintf(err,"Rasterized contours could not be opened!\n");
    G_fatal_error(err);
}

fd_rcont  = G_open_cell_old(rcont_name,mapset_in);


/*--------------------------------------------------------------------*/
/*     ALLOCATE MEMORY FOR ENTIRE DEM AND READ IN CONTOURS            */
/*--------------------------------------------------------------------*/


                                        /* Oblique profiles.    */
weight_prof = (int *) G_malloc((ncols+nrows)*sizeof(int));
dem_prof    = (CELL *) G_malloc((ncols+nrows)*sizeof(CELL));

                                        /* Entire rasters.      */
dem_ptr     = (CELL *) G_malloc(ncols*nrows*sizeof(CELL));
weight_ptr  = (int *) G_malloc(ncols*nrows*sizeof(int));

                    /* Read in contours and initialise weights.     */
for (row=0; row<nrows; row++)
{
    G_get_map_row(fd_rcont,dem_ptr + ncols*row,row);

    for (col=0; col<ncols; col++)
        *(weight_ptr + row*ncols + col) = -1;
}


/*--------------------------------------------------------------------*/
/*                INTERPOLATE SPARSE PROFILE                          */
/*--------------------------------------------------------------------*/
```

```
    printf("\nSTEP 3 - Calculating splines for profile number %d\t",profile);

    /* Open files to store profile interpolation */

    sprintf(profile_name,"%s.%d",dem_name,profile);
    sprintf(weight_name,"%s.%d.weights",dem_name,profile);

    if ((fd_dem = G_open_cell_new(profile_name,mapset_out))==NULL)
    {
        char err[256];
        sprintf(err,"Trouble opening [%s].", profile_name);
        G_fatal_error(err);
    }

    if ((fd_weight = G_open_cell_new(weight_name,mapset_out))==NULL)
    {
        char err[256];
        sprintf(err,"Trouble opening [%s].", weight_name);
        G_fatal_error(err);
    }
                                        /* Gather profiles for raster.  */
    for (row=0;row<nrows;row++)
    {
        for (col=0; col<ncols; col++)
            if (*(weight_ptr + row*ncols + col) == -1)
            {
                gather_profile(dem_ptr,dem_prof,row,col,&ncells,profile);

                if (ncells >2)
                {
                    inter_spline(ncells,dem_prof,weight_prof,profile);

                    G_percent(row,nrows,5);

                    fill_profile(dem_ptr,weight_ptr,dem_prof,weight_prof,
                                 row,col,profile);
                }

            }

        /* Save interpolated row to disk */
        G_put_map_row(fd_dem,dem_ptr + ncols*row);
        G_put_map_row(fd_weight,weight_ptr + ncols*row);
    }
    G_percent(nrows,nrows,1);


    /*-------------------------------------------------------------*/
    /*                          CLOSE DOWN                         */
    /*-------------------------------------------------------------*/

    G_close_cell(fd_dem);
    G_close_cell(fd_weight);
    G_unopen_cell(fd_rcont);
```

```
    free(dem_ptr);
    free(weight_ptr);
    free(dem_prof);
    free(weight_prof);
}
```

interpol_spline.c

```
/*****************************************************************************/
/***                        inter_spline()                                 ***/
/*** Calls spline interpolation but adds constraint between contours.    ***/
/*** Jo Wood, V1.0, December, 1991, V1.2 March 20th, 1992                ***/
/*** V2.0 Modified to conform to GRASS module strcture, 23rd July, 1995 ***/
/***                                                                       ***/
/*****************************************************************************/

#include "spline.h"

inter_spline(ncells,dem_prof,weight_prof,profile)

int     ncells;
CELL    *dem_prof;
int     *weight_prof,
        profile;

{
    /*-------------------------------------------------------------------*/
    /*                          INITIALISE                               */
    /*-------------------------------------------------------------------*/

    int         *x;     /* Three arrays holding tabulated */
    CELL        *z;     /* functions of form z = f(x)     */
    float       *zdash; /* and zdash = f'(x).             */

    int         col,
                count=0,
                overshoot=0,
                overshoot_max=0,
                elev, weight;

    float       slope_left  = 0.99e31,
                slope_right = 0.99e31;

    /* Reserve memory for arrays */

    x =         (int *)   malloc((ncells+1)*sizeof(int));
    z =         (CELL *)  malloc((ncells+1)*sizeof(CELL));
    zdash =     (float *) malloc((ncells+1)*sizeof(float));


    /*-------------------------------------------------------------------*/
    /*                          INTERPOLATE                              */
    /*-------------------------------------------------------------------*/
```

```
            /* Initialise spline routine by calculating the second
                    derivatives of the interpolating function.      */

contract(x,z,dem_prof,ncells,&count);
init_spline(x,z,count,slope_left,slope_right,zdash);

            /* Gather weights for constrained & unconstrained case. */
for (col=0;col<ncells;col++)
{
    spline(x,z,zdash,count,col,&elev,weight_prof+col);

    if (*(weight_prof+col) >=100000) /* Set contour weights to 0    */
        *(weight_prof+col) = 0;

            /* Cells are further apart for diagonal profiles,
                    so reduce weighting be a factor of root two     */
    if (profile == 2 || profile == 4)
        *(weight_prof+col)*=0.707;
}

/* For constrained case, add points to the profile where they would */
/* be exceeded in the unconstrained case.                           */

if ((interval) && (!truncation))
{
    for (col=0;col<ncells;col++)
    {
        overshoot =spline(x,z,zdash,count,col,&elev,&weight);

        if (abs(overshoot) > abs(overshoot_max))
            overshoot_max = overshoot;

        if ((overshoot) && (abs(overshoot) < abs(overshoot_max)))
        {
            *(dem_prof + col) = elev - overshoot;
            contract(x,z,dem_prof,ncells,&count);

            /* Re-initialise with new point. */
            printf("Re itiitalising at column %d (of %d)
                    (overshoot of %d), max overshoot =%d\n",
                                col,ncells,overshoot,overshoot_max);

            init_spline(x,z,count,slope_left,slope_right,zdash);
            overshoot=0;
            overshoot_max=0;
            col=0;
        }
    }
}

            /* Gather weights for constrained & unconstrained case. */

init_spline(x,z,count,slope_left,slope_right,zdash);
```

```
    for (col=0;col<ncells;col++)
    {
        overshoot = 0;
        overshoot = spline(x,z,zdash,count,col,dem_prof + col,&weight);

        if (overshoot && truncation)
           *(dem_prof + col) -= overshoot;
    }

    /*--------------------------------------------------------------------*/
    /*                          FREE MEMORY                               */
    /*--------------------------------------------------------------------*/

    free(x);
    free(z);
    free(zdash);
}




/*--------------------------------------------------------------------*/
/*                      CONTRACT SPARSE ARRAY                         */
/*--------------------------------------------------------------------*/

contract(x,z,prof,ncells,count)
    int  *x;
    CELL *z,
         *prof;
    int  ncells;
    int  *count;
{
    int col;
    *count=0;

    /*** First convert sparse row into array that just
         holds non zero values. NOTE this means that
         'genuine' zero values will be lost.          ***/

    for (col=0;col<ncells;col++)
        if (*(prof+col) != 0)
        {
          (*count)++;
          *(x+ *count) = col;
          *(z+ *count) = *(prof+col);
        }

}
```

profile.c

```
/****************************************************************************/
/***                          profile.c                                   ***/
/*** Functions to gather and fill oblique profiles across rasters.      ***/
/*** V1.1 Modified to allow any profile angle, 31st July, 1995.         ***/
/***                                                                      ***/
/****************************************************************************/

#include "spline.h"

gather_profile(cont_ptr,profile_ptr,row,col,ncells_ptr,profile)
    CELL *cont_ptr,             /* Rasterized contour values.           */
         *profile_ptr;          /* Profile across DEM.                  */
    int  row,col,               /* Location in DEM to extend profile.   */
         *ncells_ptr,           /* Number of cells in profile.          */
         profile;               /* Profile direction code.              */
{
    int xoffset,                /* Displacements along profile direction*/
        yoffset,
        offset=0,
        x=col,                  /* Position in profile.                 */
        y=row;


    /*--------------------------------------------------------------------*/
    /*                  CALCULATE PROFILE OFFSETS                         */
    /*--------------------------------------------------------------------*/

    switch (profile)
    {
        case (1):
            xoffset=1;          /* Horizontal profile direction.        */
            yoffset=0;
            break;

        case (2):
            xoffset=1;          /* NW to SE diagonal profile direction. */
            yoffset=1;
            break;

        case (3):
            xoffset=0;          /* Vertical profile direction.  */
            yoffset=1;
            break;

        case (4):
            xoffset=1;          /* NE to SW diagonal profile direction. */
            yoffset=-1;
            break;

        default:
            fprintf(stderr,"Error: Unknown profile direction\n");
            exit(-1);
    }
```

```
                                /* Move to start of profile.            */
    while ((x>0) && (y>0) && (x<ncols-1) && (y<nrows-1))
    {
        x -= xoffset;
        y -= yoffset;
    }

                                /* Fill in sparse profile.              */
    while ((x+xoffset <= ncols) && (y+yoffset <= nrows) &&
           (x+xoffset >= -1) && (y+yoffset >= -1))
    {
        *(profile_ptr + offset) = *(cont_ptr + y*ncols+x);
        offset++;
        x += xoffset;
        y += yoffset;
    }

    *ncells_ptr = offset;
}

fill_profile(dem_ptr,weight_ptr,dem_prof,weight_prof,row,col,profile)
    CELL *dem_ptr;              /* Rasterized contour values.           */
    int  *weight_ptr;           /* interpolation weights.               */
    CELL *dem_prof;             /* Profiles across DEM.                 */
    int  *weight_prof,
         row,col,               /* Location in DEM to extend profile.   */
         profile;               /* Profile direction code.              */
{

    int xoffset,                /* Displacements along profile direction*/
        yoffset,
        offset=0,               /* Profile offset.                      */

        x=col,                  /* Position in profile.                 */
        y=row;

    switch (profile)
    {
        case (1):
            xoffset=1;          /* Horizontal profile direction.        */
            yoffset=0;
            break;

        case (2):
            xoffset=1;          /* NW to SE diagonal profile direction. */
            yoffset=1;
            break;

        case (3):
            xoffset=0;          /* Vertical profile direction.  */
            yoffset=1;
            break;

        case (4):
```

```
            xoffset=1;          /* NE to SW diagonal profile direction. */
            yoffset=-1;
            break;

        default:
            fprintf(stderr,"Error: Unknown profile direction\n");
            exit(-1);
    }


                                /* Move to start of profile.            */
    while ((x>0) && (y>0) && (x<ncols-1) && (y<nrows-1))
    {
        x -= xoffset;
        y -= yoffset;
    }

                                /* Put interpolated profile in DEM      */
    while ((x+xoffset <= ncols) && (y+yoffset <= nrows) &&
           (x+xoffset >= -1) && (y+yoffset >= -1))
    {
        *(dem_ptr + y*ncols+x) = *(dem_prof + offset);
        *(weight_ptr + y*ncols+x) = *(weight_prof + offset);
        offset++;
        x += xoffset;
        y += yoffset;
    }
}
```

spline.c

```
/*****************************************************************************/
/***                  spline related functions                             ***/
/*** Taken from Press, Flannery, Teukolsky & Vetterling (1988) p.85          ***/
/*** Modified to incorporate constraints and GRASS functionality            ***/
/*** Jo Wood, v1.0, December, 1991, v2.0, July 1995.                        ***/
/***                                                                       ***/
/*****************************************************************************/

#include "spline.h"

#define F (float)

init_spline(x,z,nvals,slope_left,slope_right,zdash)

int     *x;             /* Tabulated x values */
CELL    *z;             /* Tabulated z values [ z = fn(x)] */
int     nvals;          /* Number of tabulated values */
float   slope_left,     /* LHS boundary slope */
        slope_right,    /* RHS slope (if >0.99+E30, 2nd deriv=0) */
        *zdash;         /* Tabulated z derivatives [zdash = dz/dx] */
{
    int         i,k;
    float       p,qn,
```

```
                sig,
                un,
                *u,
                *vector();

    u=vector(1,nvals-1);

    if (slope_left > 0.99e30)
        *(zdash+1) = u[1] = 0.0;
    else
    {
        *(zdash+1) = -0.5;
        u[1] = (3.0/(F*(x+2)-F*(x+1)))*((F*(z+2)-F*(z+1))/
                (F*(x+2)-F*(x+1))-slope_left);
    }

    for (i=2;i<=nvals-1;i++)
    {
        sig = (F*(x+i)-F*(x+i-1))/(F*(x+i+1)-F*(x+i-1));
        p = sig*(*(zdash+i-1))+2.0;
        *(zdash+i) = (sig-1.0)/p;
        u[i] = (F*(z+i+1)-F*(z+i))/(F*(x+i+1)-F*(x+i)) - (F*(z+i)-F*(z+i-1))/
                (F*(x+i)-F*(x+i-1));
        u[i] = (6.0*u[i]/(F*(x+i+1)-F*(x+i-1)) - sig*u[i-1])/p;
    }

    if (slope_right > 0.99e30)
        qn=un=0.0;
    else
    {
        qn=0.5;
        un = (3.0/(F*(x+nvals)- F*(x+nvals-1))) *
                (slope_right-(F*(z+nvals)- F*(z+nvals-1))/
                (F*(x+nvals)- F*(x+nvals-1)));
    }

    *(zdash+nvals) = (un-qn*u[nvals-1])/(qn*(*(zdash+nvals-1))+1.0);

    for (k=nvals-1;k>=1;k--)
        *(zdash+k) = *(zdash+k)*(*(zdash+k+1))+u[k];

    free_vector(u,1,nvals-1);

}



/*** vector() - See PRESS et al (1988) p705 ***/
/*** Allocates a float vector with range [nl...nh] ***/

float   *vector(nl,nh)
int     nl,nh;

{
```

```
    float       *v;

    v = (float *)malloc((unsigned) (nh-nl+1)*sizeof(float));
    return (v-nl);
}


/*** free_vector() - See PRESS et al (1988) p707 ***/
/*** Frees a float vector allocated by vector ***/

free_vector(v,nl,nh)
float   *v;
int     nl,nh;

{
    free((char*) (v+nl));
}



/*************************************************************/
/***                        spline()                       ***/
/*************************************************************/

spline(x,z,zdash,nvals,xi,zi,weight)

int     *x;             /* Tabulated x values */
CELL    *z;             /* Tabulated z values [ z = fn(x)] */
float   *zdash;         /* Tabulated z derivatives [zdash = dz/dx] */
int     nvals,          /* Number of tabulated values */
        xi;             /* The specic position of to be interpolated */
CELL    *zi;            /* The interpolated z value */
int     *weight;        /* The weighting given to the interpolated value */

{
    /*** ALGORITHM FROM PRESS et al. (1988) pp94-98 ***/

    int         klo=1,
                khi=nvals,
                k;

char temp[128];

    float       h,b,a;

    while (khi-klo >1)
    {
        k = (khi+klo) >> 1;

        if (F*(x+k) > F xi)
            khi=k;
        else
            klo=k;

    }
```

```
    h=F*(x+khi) - F*(x+klo);

    *weight = 100/(F xi+.001-(F*(x+klo))) + 100/(F*(x+khi)+0.001-F xi);


    if (h==0.0)
        G_fatal_error("Bad input to routine SPLINE");

    a = (F*(x+khi)- F xi)/h;
    b = (F xi- F*(x+klo))/h;

    *zi = a*(F*(z+klo)) + b*(F*(z+khi)) + ((a*a*a-a)*(*(zdash+klo)) +
                                          (b*b*b-b)*(*(zdash+khi)))*(h*h)/6.0;


    /* Check for contour overshoots and undershoots */
    /* -------------------------------------------- */

    if (interval)
    {

        /* Case One: At edge of spline function */
        /* ------------------------------------ */

        if (klo == -11)                          /* Left edge of spline functn. */
        {
            if ((*zi - *(z+khi)) >interval)      /* Overshoot.   */
            {
                printf("**Over (1) %d<-- %d -->%d\n",*(z+klo),*zi,*(z+khi));
                return ((*zi - *(z+khi) - interval)+1);
            }

            if ((*(z+khi) - *zi) >interval)      /* Undershoot.  */
            {
                printf("**Undr (2) %d<-- %d -->%d\n",*(z+klo),*zi,*(z+khi));
                return ((*zi - *(z+khi) + interval) -1);
            }
        }

        if (khi == nvals+11111)                  /* Right edge of spline functn.*/
        {
            if ((*zi - *(z+klo)) > interval)     /* Overshoot.   */
            {
                printf("**Over (3) %d<-- %d -->%d\n",*(z+klo),*zi,*(z+khi));
                return ((*zi - *(z+klo) - interval) +1);
            }

            if ((*(z+klo) - *zi) >interval)      /* Undershoot.  */
                return ((*zi - *(z+klo) + interval) -1);
            }
        }

        /* Case two: Between different value contours */
        /* ------------------------------------------ */
```

```
        if (abs(*(z+klo) - *(z+khi)) == interval)
                                /* If contours are sufficiently different*/
        {                       /* constrain spline between them.        */
            if (*zi < MIN(*(z+klo),*(z+khi)))   /* Undershoot.  */
                return ((*zi - MIN(*(z+klo),*(z+khi))) - 1);

            if (*zi > MAX(*(z+klo), *(z+khi)))  /* Overshoot */
                return ((*zi - MAX(*(z+klo),*(z+khi))) + 1);
        }
        else
        {
            /* Case three: Between identical value contours */
            /* -------------------------------------------- */

                            /* If contours are the same, constrain  */
                            /* within one contour interval.         */
            if (*zi - *(z+klo) > interval)      /* Overshoot. */
                return (((*zi - *(z+klo))- interval) +1);

            if (*(z+klo) - *zi  > interval)     /* Undershoot. */
                return (((*zi - *(z+klo)) + interval) -1);
        }
    }

    /* Default: No constraint necessary */
    /* -------------------------------- */

    return (0);
}
```

RMSE.sh

```
###############################################################################
#### Script to calculate final interpolated DEM from four sets of profiles   ####
#### created by v.surf.spline. Also calculates the RMSE for each cell in DEM ####
#### Jo Wood, v1.0 Deceber, 1991, v1.2 March 1992, v2.0 July 1995.           ####
####                                                                         ####
###############################################################################


#### Check that correct number of arguments have been given.

if test "$1" = "" || test "$2" != ""
then
        echo "Usage: $0 <dem_name>" >&2
        exit 1
fi

#### Check that user is running GRASS

if test "$GISRC" = ""
then
```

```
        echo "You must be running GRASS to execute $0" >&2
        exit 1
fi

#### Check GRASS variables are set

eval `g.gisenv`
: ${GISBASE?} ${GISDBASE?} ${LOCATION_NAME?} ${MAPSET?}

#### Envoke r.mapcalc to do the calculations

# DEM is the weighted average of the four interpolated profiles.
# The weight of each profile depends on the distance to the nearest contours.
# The nearer to a contour line the greater the weight.
# RMSE is the weighted standard deviation of the four profile elevations.

r.mapcalc <<EOF

$1 = if ($1.contours,$1.contours,($1.1*$1.1.weights + $1.2*$1.2.weights +       \
                                  $1.3*$1.3.weights + $1.4*$1.4.weights)/       \
                ($1.1.weights + $1.2.weights + $1.3.weights + $1.4.weights))    \

$1.rmse = if ($1.contours,0,sqrt(                                               \
                (($1.1-$1)*($1.1-$1)*$1.1.weights*$1.1.weights) +               \
                 (($1.2-$1)*($1.2-$1)*$1.2.weights*$1.2.weights) +              \
                 (($1.3-$1)*($1.3-$1)*$1.3.weights*$1.3.weights) +              \
                 (($1.4-$1)*($1.4-$1)*$1.4.weights*$1.4.weights)) /             \
                (($1.1.weights*$1.1.weights) + ($1.2.weights*$1.2.weights) +    \
                 ($1.3.weights*$1.3.weights) + ($1.4.weights*$1.4.weights))     \
                                ))
EOF
```

tidy.sh

```
##############################################################################
#### Script to remove files created by v.surf.spline                      ####
#### Jo Wood, v1.0 Deceber, 1991, v1.2 March 1992, v2.0 July 1995.        ####
####                                                                      ####
##############################################################################


#### Check that correct number of arguments have been given.

if test "$1" = "" || test "$2" != ""
then
        echo "Usage: $0 <dem_name>" >&2
        exit 1
fi

#### Check that user is running GRASS

if test "$GISRC" = ""
then
        echo "You must be running GRASS to execute $0" >&2
```

```
        exit 1
fi

#### Check GRASS variables are set

eval `g.gisenv`
: ${GISBASE?} ${GISDBASE?} ${LOCATION_NAME?} ${MAPSET?}

g.remove $1.1,$1.2,$1.3,$1.4,$1.contours
g.remove $1.1.weights,$1.2.weights,$1.3.weights,$1.4.weights
```

A.5 Output Statistics Modules

r.comatrix

GRASS module to display a co-occurrence matrix and calculate simple texture measures.

main.c

```
/*
** Code Compiled by Jo Wood [JWO] 29th November 1991
** Midlands Regional Research Laboratory (ASSIST)
**
**
*/

#include "gis.h"

main(argc,argv)
    int argc;
    char *argv[];
{

    /****** INITIALISE ******/

    struct Option   *map;       /* Structures required for the G_parser()*/
                                /* call. These can be filled with the   */

    struct Option   *glevs;
                                /* etc. for the GRASS user interface.   */
    struct Option   *out;

    char            *image;     /* Name of raster file to be manipulated. */
    char            *matrix;    /* Name of resultant network              */
    char            *mapset;    /* File status of layer file to be manip. */

    G_gisinit (argv[0]);        /*      This GRASS library function MUST
                                        be called first to check for valid
                                        database and mapset. As usual argv[0]
                                        is the program name. This can be
                                        recalled using G_program_name(). */


    /****** SET PARSER OPTIONS ******/

    map   = G_define_option();  /* Request pointer to memory for each option */
    glevs = G_define_option();  /* Number of z values (dimensions of matrix)    */
    out   = G_define_option();  /* Name of layer storing matrix         */

    map->key            = "image";
    map->description    = "Raster map layer to be examined";
    map->type           = TYPE_STRING;
    map->required       = YES;
```

```
    glevs->key          = "glevs";
    glevs->description  = "Number of grey level (z) `bins'";
    glevs->type         = TYPE_INTEGER;
    glevs->answer       = "128";

    out->key            = "matrix";
    out->description    = "Resultant co-occurrence matrix name";
    out->type           = TYPE_STRING;
    out->required       = YES;

    if (G_parser(argc,argv))
        exit(-1);               /*      Returns a 0 if sucessful                */

    image  = map->answer;
    matrix = out->answer;



    /****** CHECK THE CELL FILE EXISTS******/

    if ((mapset=G_find_cell2(image,""))==NULL)
    {
        char err[256];
        sprintf(err,"Raster map [%s] not available.",image);
        G_fatal_error(err);
    }

    /****** CHECK THE CELL FILE (matrix) DOES NOT ALREADY EXIST******/

    if (G_legal_filename(matrix)==NULL)
    {
        char err[256];
        sprintf(err,"Illegal file name. Please try another.");
        G_fatal_error(err);
    }
    else
    {
        if (G_find_cell(matrix,"") !=NULL)
        {
            char err[256];
            sprintf(err,
                "Raster map [%s] already exists.\nPlease try another.",matrix);
            G_fatal_error(err);
        }

    }

    /****** MANIPULATE THE CELL FILE (DEM) ******/

    comatrix(mapset,image,matrix,atoi(glevs->answer));


}
```

comatrix.c

```
/*****************************************************************/
/***                          cmatrix                          ***/
/*** Function to extract a co-occurrence matrix from an image ***/
/***                                                           ***/
/*****************************************************************/


/*---------------------------------------------------------------
  | V2.1 17th July 1992                                         |
  |    Set to ignore zero values. Change range.c and accumulation |
  | conditions to remove option. Should build this into a -z flag |
  | callable from GRASS. Note commented out normalisation of  the |
  | matrix contents with respect to expected value. Should add  a |
  | log-linear modelling capcability to the matrix.             |
   ---------------------------------------------------------------*/

/*---------------------------------------------------------------
  | V2.0  29th November 1991                                     |
  |   Present version is passed (1) Pointer to the layer to be  |
  | examined; (2) the pathnames of the current mapset; (3) the  |
  | name of the new matrix to be created; (4) the relative; (5) |
  | the number of `grey levels' into which the image is quantized.|
  |                                                             |
  |   Future improvements include (1) Separate lag calculation off|
  | into a function able to deal with more complex calculations. |
  | eg. average over various lags, distance weghted lags etc.;  |
  | (2) Ability to deal with 3D matrices, with lag increasing in |
  | the z dimension; (3) More sophisticated quantizing function. |
  | eg. histogram equalisation before quantizing. Would allow a |
  | more evenly spaced matrix to be produced. (4) Add to library |
  | of matrix measures; (5) Textual summary of matrix with matrix |
  | measures included.                                          |
   ---------------------------------------------------------------*/


#include "gis.h"
#include <math.h>

comatrix(mapset,image,matrix,glevs)

char    *mapset;                /* Mapset holding cell file to be opened.      */
char    *image, *matrix;        /* Names of cell files to be opened.           */
int     glevs;                  /* Number of grey level bins.                  */

{
        struct Range    *range;
        CELL            *row_in1,*row_in2,*row_out;

        int             fd_in,fd_out;

        int             cocr[255][255];         /* max matrix size      */
        int             sumx[255],sumy[255];
```

```
int             x,y,nrows,ncols;
CELL            zmin,zmax;
int             cell,neigh;             /* matrix coordinates   */

float           obs,
                exp;

float           contrast(),con,
                asmoment(),asmo,
                asymmetry(),asym,
                entropy(),ent,
                idmoment(),idm;          /* matrix measures     */

int             total = 0;

int             xoff=1, yoff=0;


/****** OPEN CELL FILES AND GET CELL DETAILS ******/

/* Min and max z values */

getrange(image,mapset,&zmin,&zmax);

fd_in  = G_open_cell_old(image,mapset);
fd_out = G_open_cell_new(matrix);

nrows = G_window_rows();
ncols = G_window_cols();

row_in1 = G_allocate_cell_buf();
row_in2 = G_allocate_cell_buf();


/*** Initialise co-occurrence matrix ***/

for (y=0; y<255; y++)
{
   sumx[y] = sumy[y] = 0;
   for (x=0; x<255; x++)
      cocr[y][x] = 0;
}


/*** Create co-occurrence matrix ***/


for (y=0;y<nrows;y++)
{
  G_get_map_row(fd_in,row_in1,y);
  if ((y+yoff >=0) && (y+yoff < ncols))
    G_get_map_row(fd_in,row_in2,y+yoff);

  for (x=0;x<ncols;x++)
    if ((x+xoff >=0) && (x+xoff < ncols)
```

```
        && (y+yoff >=0) && (y+yoff < ncols)
        && (*(row_in1+x)!=0) && (*(row_in2+x+xoff)!=0))
    {
        cell = quant(*(row_in1+x),zmin,zmax,glevs);
        neigh = quant(*(row_in2+x+xoff),zmin,zmax,glevs);

        /* Calculate marginal totals */

        total++;

        sumx[cell]++;
        sumy[neigh]++;

        /* Update matrix */

        cocr[cell][neigh]++;
    }
}


/* Print contents on screen if not too big */

if (glevs<17)
{
    for (y=0; y<glevs;y++)
    {
      printf("\n\n");
      for (x=0; x<glevs; x++)
         if( cocr[y][x] == 0)
            printf(" [ -- ] ");
         else
            printf("%.5f ",(float)cocr[y][x]/total);
    }
    printf("\n");
}


/*** Calculate some textural features (See Haralick et al (73) ) ***/

for(y=0; y<glevs; y++)
  for (x=0; x<glevs; x++)
  {
    con += contrast(x,y,(float)cocr[y][x]/total);
    asmo += asmoment((float)cocr[y][x]/total);
    ent += entropy((float)cocr[y][x]/total);
    asym += asymmetry((float)cocr[y][x]/total,(float)cocr[x][y]/total);
    idm += idmoment(x,y,(float)cocr[y][x]/total);
  }

printf("Contrast = %f.\n",con);
printf("Angular Second Moment = %f.\n",asmo);
printf("Entropy = %f\n",ent);
printf("Asymmetry = %f\n",asym);
printf("Inverse Difference Moment = %f\n",idm);
```

```
        /*** Write integer matrix out as cell file ***/

        G_unopen_cell(fd_in);
        fd_out = G_open_cell_new(matrix);
        row_out = G_allocate_cell_buf();

        zmin =  999999999;
        zmax = -999999999;

        for (y=0;y<glevs;y++)
           for (x=0;x<glevs;x++)
           {
                if (cocr[y][x] < zmin)
                    zmin = cocr[y][x];
                if (cocr[y][x] > zmax)
                    zmax = cocr[y][x];
           }

        for (y=0;y<nrows;y++)
        {
           for (x=0;x<ncols;x++)
              *(row_out+x) = cocr[glevs*y/(nrows-1)][glevs*x/(ncols-1)];

           G_put_map_row(fd_out,row_out);
        }

        G_close_cell(fd_out);

}


/****************************************/
/* quantize z value in glevs grey levels */
/****************************************/

quant(n,zmin,zmax,glevs)
int     n,glevs,zmin,zmax;
{

        return ((glevs-1)*(float)(n-zmin)/(zmax-zmin));
}


/*****************************/
/*** Some texture measures ***/
/*****************************/

float contrast(x,y,z)
int x,y;
float z;
{
        return((x-y)*(x-y)*z);
}
```

```
float asmoment(z)
float z;
{
        return(z*z);
}

float entropy(z)
float z;
{
        if (z != 0.0)
          return (-1*(z*log(z)));
        else
          return (0);
}

float asymmetry(z1,z2)
float z1,z2;
{
        return ((z1-z2)*(z1-z2));
}

float idmoment(x,y,z)
int x,y;
float z;
{
        return((1/(1+((x-y)*(x-y))))*z);
}
```

range.c

```
#include <gis.h>

getrange(image,mapset,min,max)
char    *image,*mapset;
CELL    *min, *max ;

{
        int                     fd;
        CELL                    *array,
                                *ptr;
        int                     x, y,
                                nrows,ncols,
                                first;

        fd = G_open_cell_old(image,mapset);

        nrows = G_window_rows();
        ncols = G_window_cols();


        array  = G_allocate_cell_buf() ;

        fprintf(stderr, "Finding min and max values ...");
        first = 1;
```

```
        for(y=0; y<nrows; y++)
        {
                G_percent (y, nrows, 2);
                G_get_map_row (fd, array, y) ;
                for(x=ncols, ptr=array; x; x--, ptr++)
                {
                        if (first)
                        {
                            *min = *max = *ptr;
                            *min +=99999;
                            first = 0;
                        }

                        if ( (*ptr < *min) && (*ptr !=0) ) *min = *ptr ;
                        if (*ptr > *max) *max = *ptr ;
                }
        }
        G_percent (y, nrows, 2);
        G_close_cell(fd);
        free(array) ;
}
```

r.lags

GRASS function to calculate various spatial dependence measures for all possible lags within a given image. Each measure is made by comparing two cells only for each lag azimuth and distance. The whole image is read once for each cell in the image, therefore is very computationally expensive. A Sparc 2 would typically take 1 hour to calculate a 150x150 image.

V1.0 written 17.6.92 to calculate Moran autocorrelation statistic.

V2.0 written 18.7.92 to also calulate Haralick's grey-tone spatial dependence textural measures.

V2.1 modified 18.9.95 to conform to standard GRASS program structure.

lags.h

```
/****************************************************************************/
/***                                                                      ***/
/***                            lags.h                                    ***/
/***  Header file for use with r.lags -                                   ***/
/***  Jo Wood, Dept of Geography, University of Leicester                 ***/
/***  V1.0  - 7th February, 1993                                          ***/
/***                                                                      ***/
/****************************************************************************/

#include "gis.h"                /* This MUST be included in all GRASS   */
                                /* programs. It sets up the necessary   */
                                /* prototypes for GRASS library calls.  */
#include <math.h>

#define TRUE 1
#define FALSE 0

#define MORAN 1
#define TEXTURAL 2

#define GLEVS 80

#define MIN(x,y)  ((x)<(y)? (x):(y))
#define MAX(x,y)  ((x)>(y)? (x):(y))

/* ------ Global variables ------ */

#ifndef MAIN
    extern                      /* Externally defined if not main()     */
#endif

char            *rast_in_name,  /* Name of the raster file to process.  */
                *rast_out1_name,/* Name of the raster output files.     */
                rast_out2_name[80],
                con_name[80],
```

```
                asmo_name[80],
                ent_name[80],
                asym_name[80],
                idm_name[80],

                *mapset_in,     /* If no problems, these will be names  */
                *mapset_out,    /* of mapsets containing the files to   */
                                /* be processed. Otherwise, error code. */

                measure,        /* Type of spatial dependence measure   */

                non_zero,       /* Flag that indicates exclusion of     */
                                /* zero values from calculations.       */
                vario;          /* Flag indicating variogram to be      */
                                /* plotted.                             */

#ifndef MAIN
    extern                      /* Externally defined if not main()     */
#endif

int             fd_in,          /* File descriptor for input and        */
                fd1_out,        /* output raster files.                 */
                fd2_out,
                fd_con,
                fd_asmo,
                fd_ent,
                fd_asym,
                fd_ent,
                fd_idm;

#ifndef MAIN
    extern                      /* Externally defined if not main()     */
#endif

double          contrast(),     /* Textural measurements.               */
                asmoment(),
                entropy(),
                asymmetry(),
                idmoment();
```

main.c

```
/*****************************************************************************/
/**                                                                         **/
/**                              r.lags                                     **/
/**  GRASS module for 2D lag graphs from spatial dependence measures.       **/
/**                                                                         **/
/**                                                                         **/
/**               Jo Wood, V 2.1, 18th September, 1995                      **/
/**                                                                         **/
/*****************************************************************************/

#define MAIN
```

```
#include "lags.h"

main(argc,argv)
    int argc;
    char *argv[];
{

    /*--------------------------------------------------------------------*/
    /*                            INITIALISE                              */
    /*--------------------------------------------------------------------*/

    /*--------------------------------------------------------------------*/
    /*                        GET INPUT FROM USER                         */
    /*--------------------------------------------------------------------*/

    interface(argc,argv);


    /*--------------------------------------------------------------------*/
    /*                 OPEN INPUT AND OUTPUT RASTER FILES                 */
    /*--------------------------------------------------------------------*/

    open_files();


    /*--------------------------------------------------------------------*/
    /*               PROCESS SURFACE FOR FEATURE DETECTION                */
    /*--------------------------------------------------------------------*/

    if (measure==MORAN)
        autocorr();
    else
        if (measure==TEXTURAL)
            comatrix();

    /*--------------------------------------------------------------------*/
    /*               CLOSE ALL OPENED FILES AND FREE MEMORY               */
    /*--------------------------------------------------------------------*/

    close_down();

    write_cols();

}
```

interface.c

```
/**************************************************************************/
/***                            interface()                             ***/
/***   Function to get input from user and check files can be opened    ***/
/***                                                                    ***/
/***      Jo Wood, Department of Geography, V1.2, 7th February 1992     ***/
/**************************************************************************/
```

```
#include "lags.h"

interface(argc,argv)

    int      argc;              /* Number of command line arguments.    */
    char    *argv[];            /* Contents of command line arguments.  */

{
    /*--------------------------------------------------------------------*/
    /*                          INITIALISE                                */
    /*--------------------------------------------------------------------*/

    struct Option *rast_in,     /* Name of raster to measure.           */
                  *rast_out,    /* Name of raster holding lag map.      */
                  *meas;        /* Type of spatial dependence measure.  */

    struct Flag *nozero;        /* Exclude non-zero values from calcn.  */
    struct Flag *variog;        /* Calculate semi-varigram instead.     */

    G_gisinit (argv[0]);        /* GRASS function which MUST be called  */
                                /* first to check for valid database    */
                                /* and mapset and prompt user for input.*/

    /*--------------------------------------------------------------------*/
    /*                      SET PARSER OPTIONS                            */
    /*--------------------------------------------------------------------*/

    rast_in     = G_define_option();  /* Request memory for options.    */
    rast_out    = G_define_option();
    meas        = G_define_option();

    nozero    = G_define_flag();
    variog    = G_define_flag();

    /* Each option has a 'key' (short descript), a 'description` (longer) */
    /* a 'type' (eg int, or string), and an indication whether
    /* manditory or not. */

    rast_in->key          = "in";
    rast_in->description  = "Raster surface layer to measure";
    rast_in->type         = TYPE_STRING;
    rast_in->required     = YES;


    rast_out->key         = "out";
    rast_out->description = "Output raster containing lag map";
    rast_out->type        = TYPE_STRING;
    rast_out->required    = YES;

    meas->key             = "measure";
    meas->description     = "Spatial dependence measure (Moran's I or texture)";
    meas->type            = TYPE_STRING;
    meas->options         = "moran,texture";
    meas->required        = NO;
    meas->answer          = "moran";
```

```
nozero->key           = 'n';
nozero->description   = "Calculate non-zero values only";

variog->key           = 'v';
variog->description   = "Calculate semi-variogram instead";


if (G_parser(argc,argv))    /* Actually performs the prompting for  */
    exit(-1);               /* keyboard input.                      */

rast_in_name  = rast_in->answer;    /* Store file names globally.   */
rast_out1_name = rast_out->answer;
strcpy(rast_out2_name,rast_out->answer);
strcat(rast_out2_name,".ns");

if (strcmp(meas->answer,"texture")==0)
    measure=TEXTURAL;
else
    measure=MORAN;

vario = variog->answer;

/*--------------------------------------------------------------------*/
/*                  CHECK INPUT RASTER FILE EXISTS                    */
/*--------------------------------------------------------------------*/


if ((mapset_in=G_find_cell2(rast_in_name,""))==NULL)
{
    char err[256];
    sprintf(err,"Raster map [%s] not available.",rast_in_name);
    G_fatal_error(err);
}


/*--------------------------------------------------------------------*/
/*             CHECK OUTPUT RASTER FILE DOES NOT EXIST                */
/*--------------------------------------------------------------------*/

mapset_out = G_mapset();            /* Set output to current mapset.*/

if (G_legal_filename(rast_out1_name)==NULL)
{
    char err[256];
    sprintf(err,"Illegal file name. Please try another.");
    G_fatal_error(err);
}
else
{
    if (G_find_cell2(rast_out1_name,mapset_out) !=NULL)
    {
        char err[256];
        sprintf(err,"Raster map [%s] exists.\nPlease try another\n",
            rast_out1_name);
        G_fatal_error(err);
```

```
        }

        if (G_find_cell2(rast_out2_name,mapset_out) !=NULL)
        {
            char err[256];
            sprintf(err,"Raster map [%s] exists.\nPlease try another\n",
                rast_out2_name);
            G_fatal_error(err);
        }
    }
}
```

open_files.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                            open_files()                               ***/
/***                  Opens input and output raster files.                 ***/
/***               Jo Wood, Project ASSIST, 24th January 1993              ***/
/***                                                                       ***/
/*****************************************************************************/

#include "lags.h"


open_files()
{
    char err[256];      /* Sting holding error message.         */


    /* Open existing file and set the input file descriptor. */
    /* ----------------------------------------------------- */

    if ( (fd_in=G_open_cell_old(rast_in_name,mapset_in)) <0)
    {
        sprintf(err,"Problem opening input file [%s].",rast_in_name);
        G_fatal_error(err);
    }


    /* Open new files and set the output file descriptor. */
    /* -------------------------------------------------- */

    if (measure == MORAN)
    {
        if ( (fd1_out=G_open_cell_new(rast_out1_name,mapset_out)) <0)
        {
            sprintf(err,"Problem opening output file [%s].",rast_out1_name);
            G_fatal_error(err);
    }

    if (measure == TEXTURAL)
    {
        strcpy(con_name,rast_out1_name);
```

```
        strcat(con_name,".con");

        if ( (fd_con = G_open_cell_new(con_name,mapset_out)) <0)
        {
            sprintf(err,"Problem opening output file [%s].",con_name);
            G_fatal_error(err);
        }


        strcpy(asmo_name,rast_out1_name);
        strcat(asmo_name,".asmo");

        if ( (fd_asmo = G_open_cell_new(asmo_name,mapset_out)) <0)
        {
            sprintf(err,"Problem opening output file [%s].",asmo_name);
            G_fatal_error(err);
        }


        strcpy(ent_name,rast_out1_name);
        strcat(ent_name,".ent");

        if ( (fd_ent = G_open_cell_new(ent_name,mapset_out)) <0)
        {
            sprintf(err,"Problem opening output file [%s].",ent_name);
            G_fatal_error(err);
        }


        strcpy(asym_name,rast_out1_name);
        strcat(asym_name,".asym");

        if ( (fd_asym = G_open_cell_new(asym_name,mapset_out)) <0)
        {
            sprintf(err,"Problem opening output file [%s].",asym_name);
            G_fatal_error(err);
        }


        strcpy(idm_name,rast_out1_name);
        strcat(idm_name,".idm");

        if ( (fd_idm = G_open_cell_new(idm_name,mapset_out)) <0)
        {
            sprintf(err,"Problem opening output file [%s].",idm_name);
            G_fatal_error(err);
        }
    }
}
```

comatrix.c

```
/*****************************************************************************/
/*                                comatrix()                                 */
/* Function to calulate various co-occurrence matrix texture measures.       */
/* Jo Wood, V1.0 December, 1992, V1.1 October, 1995.                         */
/*                                                                           */
/*****************************************************************************/

#include "lags.h"

comatrix()
{
    /*-----------------------------------------------------------------------*/
    /*                           INITIALISE                                  */
    /*-----------------------------------------------------------------------*/

    CELL    *raster,            /* Buffer hold one raster row.              */
            zi,zj,              /* The two raster values to compare.        */
            zmin,zmax,          /* Range of raster values.                  */
            *con_row,           /* Buffers storing texture measures.        */
            *asmo_row,
            *ent_row,
            *asym_row,
            *idm_row;

    int     row,col,
            nrows,ncols,        /* Number of cell rows and columns.         */
            xoffset,yoffset,    /* 2D lag vector displacement.              */
            *cocr,              /* Raster to hold co-occurrence matrix.     */
            n;                  /* The number of pairs to consider.         */

    double  con,                /* The five texture measures.               */
            asmo,
            ent,
            asym,
            idm;

    nrows = G_window_rows();    /* Find dimensions of raster.               */
    ncols = G_window_cols();

                                /* Reserve memory for entire raster         */
                                /* co-occurrence matrix, and texture        */
                                /* measures.                                */
    raster = (CELL*) G_malloc(nrows*ncols*sizeof(CELL));
    cocr   = (int *) G_malloc(GLEVS*GLEVS*sizeof(int));

    con_row  = G_allocate_cell_buf();
    asmo_row = G_allocate_cell_buf();
    ent_row  = G_allocate_cell_buf();
    asym_row = G_allocate_cell_buf();
    idm_row  = G_allocate_cell_buf();
```

```
/*-------------------------------------------------------------------*/
/*           READ IN RASTER AND CREATE CO-OCCURRENCE MATRIX          */
/*-------------------------------------------------------------------*/


printf("STAGE ONE - Reading in data\n");

for (row=0;row<nrows; row++)
    G_get_map_row(fd_in,raster+ncols*row,row);


printf("STAGE TWO - Calculating co-occurrence matrix\n");


for(yoffset=-1*(nrows/2);yoffset<(nrows/2);yoffset++)
{
    for(xoffset=-1*(ncols/2);xoffset<(ncols/2);xoffset++)
    {
        G_percent(yoffset+(nrows/2),nrows,1);

        /* Find range of data. */
        /* ------------------- */

        init_matrix(cocr,GLEVS);     /* Initialise co-occurence matrix */
        n    = 0;                    /* Reset counters.                */
        con  = 0.0;
        asmo = 0.0;
        ent  = 0.0;
        asym = 0.0;
        idm  = 0.0;
        zmin =  999999;
        zmax = -999999;

        for (row=0;row<nrows;row++)
            if ((row+yoffset<nrows) && (row+yoffset>=0))
                for (col=0;col<ncols;col++)
                    if ((col+xoffset<ncols) && (col+xoffset>=0))
                    {
                        n++;
                        zi = *(raster + row*ncols + col);
                        zj = *(raster + (row+yoffset)*ncols + col+xoffset);

                        if ( MIN(zi,zj) < zmin)
                            zmin = MIN(zi,zj);

                        if (MAX(zi,zj) > zmax)
                            zmax = MAX(zi,zj);
                    }

        /* Calculate co-occurrence matrix. */
        /* ------------------------------- */

        for (row=0;row<nrows;row++)
            if ((row+yoffset<nrows) && (row+yoffset>=0))
                for (col=0;col<ncols;col++)
```

```
                    if ((col+xoffset<ncols) && (col+xoffset>=0))
                    {
                        zi =quant(*(raster + row*ncols + col),
                                                zmin,zmax,GLEVS);
                        zj =quant(*(raster+(row+yoffset)*ncols+col+xoffset),
                                                zmin,zmax,GLEVS);

                        *(cocr + zj*GLEVS + zi) = *(cocr + zj*GLEVS + zi)+1;
                    }

        /* Calculate matrix statistics. */
        /* ---------------------------- */

        for(row=0; row<GLEVS; row++)
            for (col=0; col<GLEVS; col++)
            {
                con  += contrast(row,col,
                            (double) *(cocr + row*GLEVS + col)/(double)n);
                asmo += asmoment(
                            (double) *(cocr + row*GLEVS + col)/(double)n);
                ent  += entropy(
                            (double) *(cocr + row*GLEVS + col)/(double)n);
                asym += asymmetry(
                            (double) *(cocr + col*GLEVS + row)/(double)n,
                            (double) *(cocr + row*GLEVS + col)/(double)n);
                idm  += idmoment(row,col,
                            (double) *(cocr + row*GLEVS + col)/(double)n);
            }

        con_row[xoffset+(ncols/2)] = rint(10*con);
        asmo_row[xoffset+(ncols/2)] = rint(1000*asmo);
        ent_row[xoffset+(ncols/2)] = rint(1000*ent);
        asym_row[xoffset+(ncols/2)] = rint(1000*asym);
        idm_row[xoffset+(ncols/2)] = rint(1000*idm);

    }

    /* Output raster rows. */
    /* ------------------- */

    G_put_map_row(fd_con,con_row);
    G_put_map_row(fd_asmo,asmo_row);
    G_put_map_row(fd_ent,ent_row);
    G_put_map_row(fd_asym,asym_row);
    G_put_map_row(fd_idm,idm_row);

}

free(raster);
free(cocr);
}
```

autocorr.c

```
/*****************************************************************************/
/*                                autocorr()                                 */
/*          Function to calulate Moran's I at all possible lags              */
/*                                                                           */
/*****************************************************************************/

#include "lags.h"

autocorr()
{
    /*-----------------------------------------------------------------*/
    /*                          INITIALISE                             */
    /*-----------------------------------------------------------------*/

    CELL    *raster,            /* Buffers hold one raster row.         */
            *moran,             /* Moran measure.                       */
            *samples;           /* Number of samples at each lag.       */

    int     row,col,
            nrows,ncols;        /* Number of cell rows and columns.     */

    int     n=0,                /* Sample size of i or j.               */
            zi,zj;              /* The two raster values.               */

    double  zibar=0.0,          /* Moment parameters of i and j.        */
            zjbar=0.0,
            vari=0.0,
            varj=0.0,
            var=0.0,
            covar=0.0;          /* Covaraince measure                   */

    int     xoffset,            /* 2D lag vector displacement.          */
            yoffset;

    nrows = G_window_rows();    /* Find dimensions of raster.           */
    ncols = G_window_cols();

                                /* Reserve memory for entire raster.    */
    raster = (CELL*) G_malloc(nrows*ncols*sizeof(CELL));

    moran   = G_allocate_cell_buf();
    samples = G_allocate_cell_buf();



    /*-----------------------------------------------------------------*/
    /*              CALCULATE AUTOCORRELATION STATISTICS               */
    /*-----------------------------------------------------------------*/


    printf("STAGE ONE - Reading in data\n");
```

```
for (row=0;row<nrows; row++)
    G_get_map_row(fd_in,raster+ncols*row,row);


printf("STAGE TWO - Calculating statistics\n");

for(yoffset=-1*(nrows/2);yoffset<(nrows/2);yoffset++)
{
    for(xoffset=-1*(ncols/2);xoffset<(ncols/2);xoffset++)
    {
        G_percent(yoffset+(nrows/2),nrows,1);


        /* First pass - find number of samples and their mean */
        /* -------------------------------------------------- */

        for (row=0;row<nrows;row++)
            if ((row+yoffset<nrows) && (row+yoffset>=0))
                for (col=0;col<ncols;col++)
                    if ((col+xoffset<ncols) && (col+xoffset>=0))
                    {
                        zi = *(raster + row*ncols + col);
                        zj = *(raster + (row+yoffset)*ncols + col+xoffset);

                        if (vario)
                            var += (zi-zj)*(zi-zj);
                        else
                        {
                            zibar += zi;
                            zjbar += zj;
                        }
                        n++;
                    }

        if (vario)                  /* For variogram, we can leave loop */
        {                           /* before calculating co-variance.  */
            moran[xoffset+(ncols/2)] = var/(2*n);
            var = 0.0;
            n = 0;
            continue;
        }

    if (vario)
       fprintf(stderr,"Error: shouldnt be here.");

        zibar /= n;
        zjbar /= n;

        /* Second pass - find variance and co-variance  */
        /* -------------------------------------------- */

        for (row=0;row<nrows;row++)
            if ((row+yoffset<nrows) && (row+yoffset>=0))
                for (col=0;col<ncols;col++)
                    if ((col+xoffset<ncols) && (col+xoffset>=0))
```

```
                        {
                            zi = *(raster + row*ncols + col);
                            zj = *(raster + (row+yoffset)*ncols + col+xoffset);
                            covar += (zi - zibar) * (zj - zjbar);
                            vari  += (zi - zibar) * (zi - zibar);

                            if ( (row + yoffset*2 >=nrows) ||
                                 (row + yoffset*2 < 0)     ||
                                 (col + xoffset*2 >=ncols) ||
                                 (col + xoffset*2 < 0))
                                varj  += (zj - zjbar) * (zj - zjbar);

                        }
            var = (vari + varj);        /* Note that varj is in fact the */
                                        /* var of var(i U j) - var (i)   */

            if (var != 0.0)
                moran[xoffset+(ncols/2)] = 1000.0*covar/var;
            else
                moran[xoffset+(ncols/2)] = 1000;

            zibar= 0.0;
            zjbar= 0.0;
            n    = 0;
            covar= 0.0;
            vari = 0.0;
            varj = 0.0;

        }

        /* Output raster row */
        /* ----------------- */

        G_put_map_row(fd1_out,moran);
    }

    free(raster);
}
```

measures.c

```
/*****************************************************************/
/*                                                               */
/*          measures - Various co-occurrence matrix measures     */
/*        Jo Wood, V1.0, December 1992, V1.1 October, 15th, 1995 */
/*                                                               */
/*****************************************************************/

#include "lags.h"
```

```
/************************************************************************/
/* init_matrix() - Fill a co-occurence matrix with zeros                */
/************************************************************************/

init_matrix(cocr,glevs)
int     *cocr,                  /* Pointer to co-occurrence matrix.     */
        glevs;                  /* Dimension of matrix.                 */
{
    int posn;                   /* Position in matrix.                  */

    for (posn=0; posn <glevs*glevs; posn++)
        *(cocr + posn) = 0;
}


/************************************************************************/
/* quant() - Quantize z value in glevs grey levels                      */
/************************************************************************/

quant(n,zmin,zmax,glevs)
CELL    n,                      /* Value to quantize.                   */
        zmin,zmax;              /* Range from which z is drawn.         */
int     glevs;                  /* Number of classes to quantize into.  */
{
    return ((glevs-1)*(float)(n-zmin)/(zmax-zmin));
}




double contrast(x,y,z)
int x,y;
double z;

{
   return((x-y)*(x-y)*z);
}

double asmoment(z)
double z;
{
    return(z*z);
}

double entropy(z)
double z;
{
    if (z != 0.0)
        return (-1*(z*log(z)));
    else
        return (0.0);
}

double asymmetry(z1,z2)
double z1,z2;
```

```
{
    return ((z1-z2)*(z1-z2));
}

double idmoment(x,y,z)
int x,y;
double z;
{
    return((1/(1+((x-y)*(x-y))))*z);
}
```

close_down.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                              close_down()                             ***/
/***       Closes all input and output raster files and frees memory.      ***/
/***               Jo Wood, Project ASSIST, 7th February 1993              ***/
/***                                                                       ***/
/*****************************************************************************/

#include "lags.h"


close_down()
{
    /* Close connection with existing input raster. */

    G_unopen_cell(fd_in);

    /* Write output raster file and close connection. */


    if (measure==MORAN)
    {
        G_close_cell(fd1_out);

    }

    if (measure==TEXTURAL)
    {
        G_close_cell(fd_con);
        G_close_cell(fd_asmo);
        G_close_cell(fd_ent);
        G_close_cell(fd_asym);
        G_close_cell(fd_idm);
    }
}
```

write_cols.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                            write_cols()                               ***/
/***             Writes out colour file for lag maps                       ***/
/***               Jo Wood, Project ASSIST, 21st February 1995             ***/
/***                                                                       ***/
/*****************************************************************************/

#include "lags.h"


write_cols()
{
    /*-------------------------------------------------------------------------*/
    /*                            INITIALISE                                   */
    /*-------------------------------------------------------------------------*/

    struct Colors       colours;

    G_init_colors(&colours);

    /*-------------------------------------------------------------------------*/
    /*                       FILL OUT COLORS STRUCTURE                         */
    /*-------------------------------------------------------------------------*/
    switch(measure)
    {
        case MORAN:

            G_add_color_rule(-1000,   0,   0,   0,                 /* Black */
                              -500,   0,   0,255, &colours);       /* Blue  */
            G_add_color_rule( -500,   0,   0,255,                  /* Blue  */
                                 0,255,255,255, &colours);       /* White */
            G_add_color_rule(    0,255,255,255,                    /* White */
                               500,255,   0,   0, &colours);       /* Red   */
            G_add_color_rule(  500,255,   0,   0,                  /* Red   */
                              1000,   0,   0,   0, &colours);      /* Black */
            break;


        default:
            break;
    }

    /*-------------------------------------------------------------------------*/
    /*                      WRITE OUT COLORS STRUCTURE                         */
    /*-------------------------------------------------------------------------*/

    G_write_colors(rast_out1_name,mapset_out,&colours);

    G_free_colors(&colours);

}
```

r.statistics

Program to calculate simple univariate statistics from a raster layer.
Jo Wood, 26th March, 1995.

Modified, 6th July, 1995 to include skewness and kurtosis.

Modified, 9th July, 1995 to allow exclusion of zero values.

stats.h

```
/*****************************************************************/
/***                          stats.h                          ***/
/***                Header file for use with r.stats           ***/
/***              Jo Wood, V 1.0  - 13th September, 1994       ***/
/*****************************************************************/

#include "gis.h"                /* This MUST be included in all GRASS   */
                                /* programs. It sets up the necessary   */
                                /* prototypes for GRASS library calls.  */

#include <math.h>               /* For the sqrt() function in mathlib.  */
#include <limits.h>             /* Stores min and maximum values for    */
                                /* different data types.                */

/* ------ Global variables ------ */

#ifndef MAIN
    extern                      /* Externally defined if not main()     */
#endif

char    *rast_in_name,          /* Name of raster to process.           */

        *mapset_in,             /* Names of mapset containing the       */
                                /* file to be processed.                */

        no_zero;                /* Flag indicating exclusion of zeros.  */

#ifndef MAIN
    extern
#endif

int     fd_rast_in;             /* File descriptor for input raster.    */
```

main.c

```
/*****************************************************************************/
/***                                                                       ***/
/***          r.statistics - Calculates univarite stats for a raster.      ***/
/***                              Jo Wood                                  ***/
/***                     v 2.1  27th March, 1995.                          ***/
/***                                                                       ***/
/*****************************************************************************/

#define MAIN

#include "stats.h"

main(argc,argv)
    int argc;
    char *argv[];
{
    /*-------------------------------------------------------------------*/
    /*                       GET INPUT FROM USER                         */
    /*-------------------------------------------------------------------*/

    interface(argc,argv);

    /*-------------------------------------------------------------------*/
    /*                      OPEN INPUT RASTER FILE                       */
    /*-------------------------------------------------------------------*/

    open_files();


    /*-------------------------------------------------------------------*/
    /*                        PROCESS RASTER FILE                        */
    /*-------------------------------------------------------------------*/

    process();

    /*-------------------------------------------------------------------*/
    /*                    CLOSE ALL OPENED FILES AND FREE MEMORY         */
    /*-------------------------------------------------------------------*/

    close_down();

}
```

interface.c

```
/*****************************************************************************/
/***      Function to get input from user and check files can be opened    ***/
/***                                                                       ***/
/***                     Jo Wood,  V1.0, 13th September 1994               ***/
/*****************************************************************************/

#include "stats.h"

interface(argc,argv)

    int      argc;              /* Number of command line arguments     */
    char    *argv[];            /* Contents of command line arguments.  */

{
    /*-------------------------------------------------------------------*/
    /*                           INITIALISE                              */
    /*-------------------------------------------------------------------*/

    struct Option       *rast_in;       /* Pointer to  structures holding  */
    struct Flag         *zero_flg;      /* the text to describe each option*/

    G_gisinit (argv[0]);                /* Link with GRASS interface.      */


    /*-------------------------------------------------------------------*/
    /*                        SET PARSER OPTIONS                         */
    /*-------------------------------------------------------------------*/

    rast_in  = G_define_option();
    zero_flg = G_define_flag();

    /* Each option needs a 'key' (short description),
                         a 'description` (a longer one),
                         a 'type' (eg intiger, or string),
                    and an indication whether manditory or not */

    rast_in->key          = "rast";
    rast_in->description  = "raster layer to process";
    rast_in->type         = TYPE_STRING;
    rast_in->required     = YES;

    zero_flg->key         = 'z';
    zero_flg->description = "exclude zero values";

    if (G_parser(argc,argv))            /* Performs the prompting for      */
        exit(-1);                       /* keyboard input.                 */

    rast_in_name  = rast_in->answer;    /* Now that keyboard input has     */
    no_zero = zero_flg->answer;         /* been parsed, place the contents */
                                        /* into string arrays.             */
```

```
    /*--------------------------------------------------------------------*/
    /*                    CHECK INPUT RASTER FILES EXIST                  */
    /*--------------------------------------------------------------------*/

    if ((mapset_in=G_find_cell2(rast_in_name,""))==NULL)
    {
        char err[256];
        sprintf(err,"Raster map [%s] not available.",rast_in_name);
        G_fatal_error(err);
    }

}
```

open_files.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                           open_files()                                ***/
/***                Opens input raster file for r.statistics               ***/
/***                   Jo Wood, V1.0, 13th September, 1994                 ***/
/***                                                                       ***/
/*****************************************************************************/

#include "stats.h"

open_files()
{
    /* Open existing files and set the input file descriptors */

    if ( (fd_rast_in=G_open_cell_old(rast_in_name,mapset_in)) <0)
    {
        char err[256];
        sprintf(err,"ERROR: Problem opening raster file.");
        G_fatal_error(err);
    }
}
```

process.c

```
/*****************************************************************************/
/***                                                                       ***/
/***                            process()                                  ***/
/***          Reads in a raster files row by row for processing            ***/
/***                  Jo Wood, V1.0, 13th September, 1994                  ***/
/***                                                                       ***/
/*****************************************************************************/

#include "stats.h"

process()
{
```

```
/*--------------------------------------------------------------------*/
/*                            INITIALISE                              */
/*--------------------------------------------------------------------*/

CELL        *row_in;        /* Buffer to hold raster row.             */

int         nrows,          /* Will store the current number of       */
            ncols,          /* rows and columns in the raster.        */

            n=0,            /* Number of samples.                     */

            row,col;        /* Counts through each row and column     */
                            /* of the input raster.                   */

CELL        min = INT_MAX,  /* Raster statistics.                     */
            max = INT_MIN,
            range,
            z;

double      mean  = 0.0,
            stdev = 0.0,
            skew  = 0.0,
            kurt  = 0.0;


/*--------------------------------------------------------------------*/
/*                      GET DETAILS OF INPUT RASTER                   */
/*--------------------------------------------------------------------*/

row_in = G_allocate_cell_buf(); /* Allocate row buffers for I/O.      */

nrows  = G_window_rows();       /* Find out the number of rows and    */
ncols  = G_window_cols();       /* columns of the raster view.        */


/*--------------------------------------------------------------------*/
/*      PROCESS INPUT RASTER ROW BY ROW AND CALCULATE STATISTICS       */
/*--------------------------------------------------------------------*/

for(row=0; row<nrows; row++)
{
    G_get_map_row(fd_rast_in, row_in, row);

    for(col = 0; col < ncols; col++)
    {
        z = *(row_in + col);

        if ((no_zero) && (z == 0))
            ;               /* Do nothing */
        else
        {
            n++;

            /* ---- Range statistics ---- */
```

```
            if (max < z)
                max = z;
            else
                if (min > z)
                    min = z;


            /* ---- Mean statistic ---- */

            mean += z;
        }
    }
}
mean /=  n;
range = max - min;

for(row=0; row<nrows; row++)
{
    G_get_map_row(fd_rast_in, row_in, row);

    for(col = 0; col < ncols; col++)
    {
        z = *(row_in + col);

        if ((no_zero) && (z == 0))
            ;               /* Do nothing */
        else
        {
            /* ---- Standard deviation ---- */

            stdev += ((z - mean)*(z - mean));
        }
    }
}
stdev = sqrt(stdev/n);


for(row=0; row<nrows; row++)
{
    G_get_map_row(fd_rast_in, row_in, row);

    for(col = 0; col < ncols; col++)
    {
        z = *(row_in + col);

        if ((no_zero) && (z == 0))
            ;               /* Do nothing */
        else
        {
            /* ---- Skewness and Kurtosis ---- */

            skew  += ((z - mean)*(z - mean)*(z - mean))/
                                        (stdev*stdev*stdev);
            kurt  += ((z - mean)*(z - mean)*(z - mean)*(z - mean))/
                                        (stdev*stdev*stdev*stdev);
```

```
            }
        }
    }

    skew = skew / n;
    kurt = kurt / n;

    /*---------------------------------------------------------------------*/
    /*                    DISPLAY UNIVARIATE STATITICS                     */
    /*---------------------------------------------------------------------*/

    printf("\nUNIVARIATE STATISTICS\n====================\n");
    printf("%s\n",rast_in_name);
    printf("--------------------\n");
    printf("n       = %d ",n);
    if (no_zero)
        printf("(excluding zeros)\n");
    else
        printf("\n");
    printf("Mean    = %.4f\nStd dev = %.4f \n", mean,stdev);
    printf("--------------------\n");
    printf("Skewness= %.4f\nKurtosis= %.4f \n", skew,kurt);
    printf("--------------------\n");
    printf("Minimum = %d \nMaximum = %d\nRange   = %d \n", min,max,range);
    printf("====================\n");

}
```

close_down.c

```
/***************************************************************************/
/***                                                                     ***/
/***                            close_down()                             ***/
/***       Closes all input and output raster files and frees memory.    ***/
/***                   Jo Wood, V1.0, 13th September, 1994               ***/
/***                                                                     ***/
/***************************************************************************/

#include "stats.h"

close_down()
{

    /* Close connection with existing input rasters */

    G_unopen_cell(fd_rast_in);
}
```