Visual Force

4th Year Project Report

Richard D. Grey

June 2nd, 1993.

Departments of Artificial Intelligence and Computer Science

Abstract

This project report describes an approach used to automatically determine the lines of visual force which correspond to the convex and concave features in a natural landscape. Using digital terrain data as input, ridges and valleys (the convex and concave features) are identified — by estimating the surface fit around each pixel — and tracked using a Canny, non–maximal suppression type algorithm. The lines of visual force are then determined according to feature strength and orientation, to be plotted on top of a wireframe mesh representing the landscape.

Acknowledgements

I would like to thank the following people for all their help throughout the year : Bob Fisher, my project supervisor, for the guided help he so willingly gave in advising me on the best approach to tackling both the project itself and the final thesis; David Wren for starting me off with the Singular Value Decomposition method, and also providing a couple of necessary and useful references; Andrew Fitzgibbon for help with all the SVD library routines from "Numerical Recipes in C" and with deriving the mathematical formulae used in calculating pixel orientations; and all the departmental computing staff for keeping the system going, thereby allowing me no room for excuses!

Contents

List of Figures

Chapter 1

Introduction

1.1 What is Visual Force ?

Visual force is a principle embodied in art, architecture and graphic design which causes the eye and mind to react in a predictable and dynamic way. By exploiting these predictable responses, the illusion of motion, for example, can be introduced through the interaction of carefully designed patterns and shapes.

In a landscape, visual force occurs naturally, automatically drawing an observer's eye up concave features — valleys and gullies — and down the convex features — ridges and spars.

1.2 The Rôle of the Forestry Commission

Vegetation growing naturally tends to follow the lines of visual force by rising higher in hollows (the concave features) than on the more exposed, convex features. This natural pattern of growth works with the observing eye to create an overall, harmonious and non–disrupted view of the land.

For this reason, it is important that artificially planted forests are designed so as not to upset the natural balance or look out of place. A well unified relationship between the visual forces and the planted vegetation must be created and maintained and so, the Forestry Commission must carefully plan the design of forests along the lines of visual force.

1.3 Project Motivation

Presently, this form of land appraisal is done manually by the Forestry Commission where a landscape architect, beginning with a contour map, identifies the concave and convex features (essentially the valleys and ridges) and plots the lines of visual force which are correspondingly tied to those features.

This process can take anything up to a whole day, depending on the landform under consideration. Hence, the aim of this project is to automate and speed up the identification, spatial location and quantification of the lines of visual force.

1.4 Design Approach

The problem of identifying the lines of visual force in a landscape can be sub–divided into a number of smaller, distinct problems. Each smaller problem can be thought of as a separate module applying its own solution towards the overall, larger task.

The various different stages necessary in planning can be visualised as shown in *Figure 1.1* and can be described as follows — the letters corresponding to boxes in the figure :

[a] Firstly, the data to be used must be in a suitable format for input into the system. This involves extracting the required information and the removal of noise and/or finer detail which may cause difficulties.

[b] From the data, the local surface shape about each pixel, and then the curvature, can be calculated.

[c] Using the surface curvature information from [b] the orientation of the curvature directions at each pixel can then be estimated.

[d] Stage [b] also provides the necessary curvature information to be able to classify each pixel into one of a number of types (valley, ridge, other).

[e] Combining the results from [b] *&* [c] enables the grouping of similar pixels into distinct features, which can then be tracked and identified.

[f] Having identified the features in the landscape, the visual saliency of each can be calculated, depending on their scale and irregularity.

[g] The final stage sees the graphical results of plotting the calculated lines of visual force on top of a mesh which represents the landscape data.

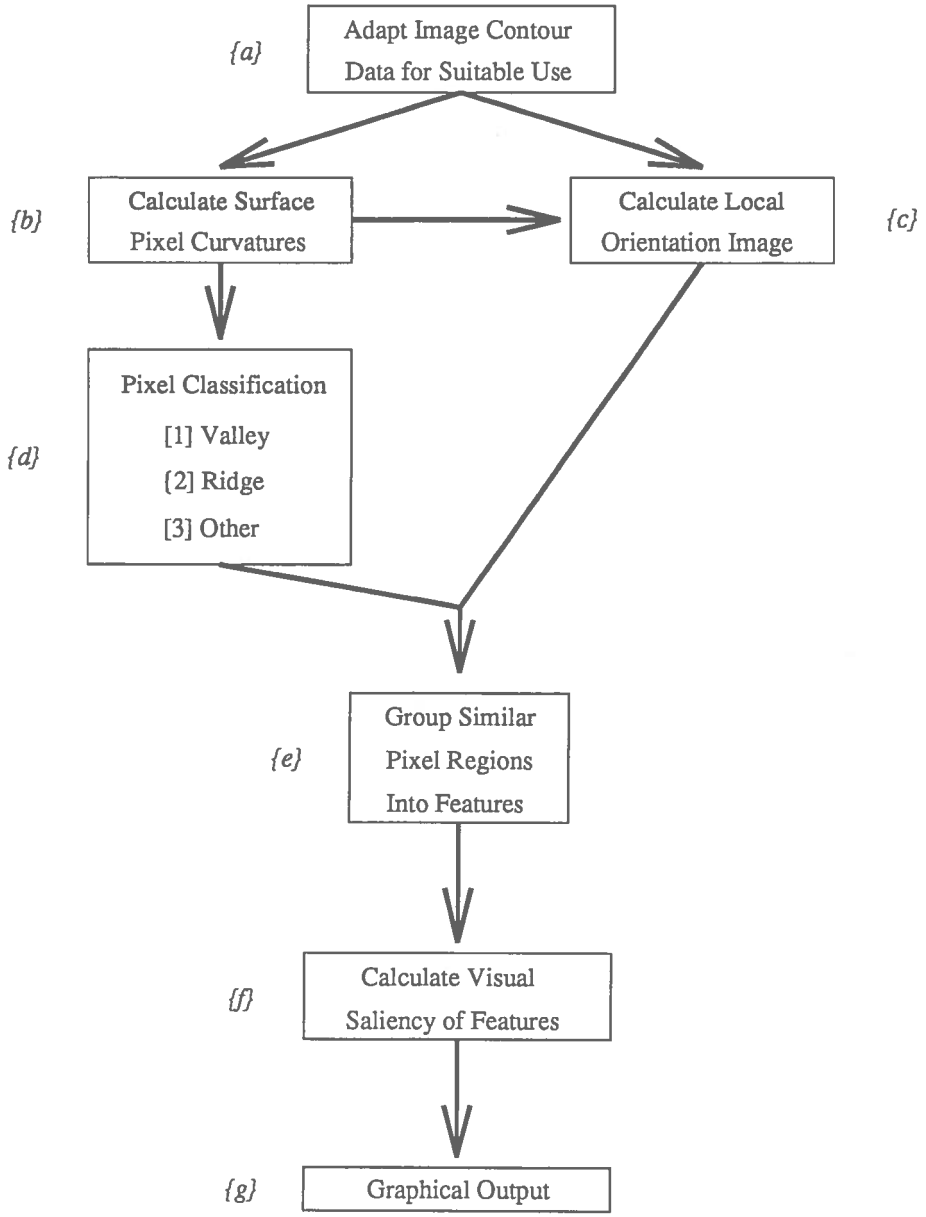

Figure 1.1: *Structured Plan.*

1.5 Project Report Overview

The body of the report describes the research results on identifying the lines of visual force, detailing the choices made, problems encountered along the way, results found and conclusions formed for each of the stages outlined in the structured plan.

Chapter 2 provides further background to the project, dealing with how visual force can actually be represented, along with the reasons behind the choice of data to be used with the system, and how overall performance may be measured.

Chapter 3 details the conversion of the original data into a form suitable for manipulation (Stage [a]), with Chapter 4 covering the mathematical methods used in Stages [b], [c] *&* [d] for calculating the pixel curvature, orientation and classification information.

Chapter 5 deals with the Canny non–maximal suppression type algorithm used to isolate the ridge apexes and valley minima in the data (Stage [e]), while Chapter 6 outlines the approach to be used in calculating the visual saliency (Stage [f]) of the features.

A critical analysis of the performance is provided in Chapter 7 along with some pointers and ideas towards further work. Also included is a brief summary of an alternative approach to ridge finding which could be adapted and extended.

Chapter 2

Background

2.1 Representing Visual Force

We know that the aspects of visual force are tied to the concave and convex features in a natural landscape, but how can these forces be represented?

As stated earlier, the visual forces which occur naturally draw the eye up concave and down convex slopes. It makes sense then to show visual forces through the use of arrows, directed in the way the observing eye is drawn. Within the Forestry Commission, these arrows are colour–coded : green for visual forces pulling the eye up the concave features; red for those pushing the eye downwards.

However, directed arrows only indicate the ways in which the visual forces flow — they give no measure of the strength of the forces. This strength information is easily represented though by combining the scale and irregularity of the feature in order to determine a width for the force representing arrows. The scale of a feature (depth of a valley or height of a ridge) can be found from the contour values being worked with, however, heuristics are often needed and employed in determining the irregularities.

In short then, visual forces are depicted by means of directed arrows whose width is a measure of the degree of concavity or convexity of the underlying feature slope. Green arrows depict attracting, red arrows repellent forces.

2.2 A Region to Work With

What use is a system unless it has been built to take into account the problems that real data can present? In order to overcome such possibilities and give the system wide enough scope to cover all eventualities, it is necessary to provide a suitable test region with which to work. Such an area should offer a wide diversity of feature type, of differing scale and irregularity.

For this reason, the area chosen to be used is a region of the Cullin Mountains on the Isle of Skye, in Scotland. Two Ordnance Survey (O.S.) National Grid Squares — references **NG40** *&* **NG42** — cover a suitable portion of these mountains. They provide ample opportunity for locating ridges and valleys in order to identify the visual forces, thus acting as a good testbed for an automated system.

2.3 The Data

Having decided on a suitable test area, it was then necessary to obtain the relevant data for input into the system. However, real data, representing the chosen region, is far too complex for initial algorithm development due to the level of detail it contains. We need to be sure that the techniques work correctly on simple cases before we can possibly apply them to data representative of a natural landscape.

2.3.1 Synthetic Test Data

To enable testing of the techniques to be employed, non–complex data needed to be used. This ideal, noise–free data was generated artificially and could be used to represent simple, precise landform features (ridges and valleys) to which the visual forces are so closely tied.

To allow for all cases where natural features could have any arbitrary orientation, the synthetic test data was created using the following formulae :

$$\textbf{valley generation} \ : z = 1 + (x \sin(\theta) - y \cos(\theta))^2$$

$$\textbf{ridge generation} \ : z = 10 - (x \sin(\theta) - y \cos(\theta))^2$$

This generates an (x, y) array of z height values, with θ specifying the angle at which the feature is orientated about the y–axis. *Figure 2.1* shows an artificially created ridge, while *Figure 2.2* shows a valley, both generated using the above formulae[1].

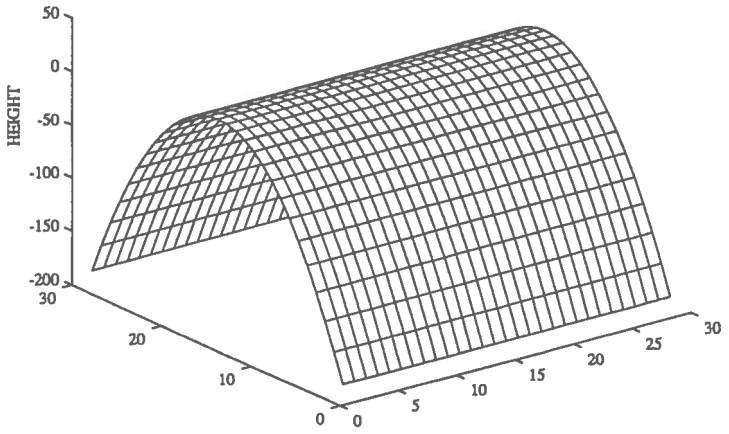

Figure 2.1: *Synthetic Test Ridge.*

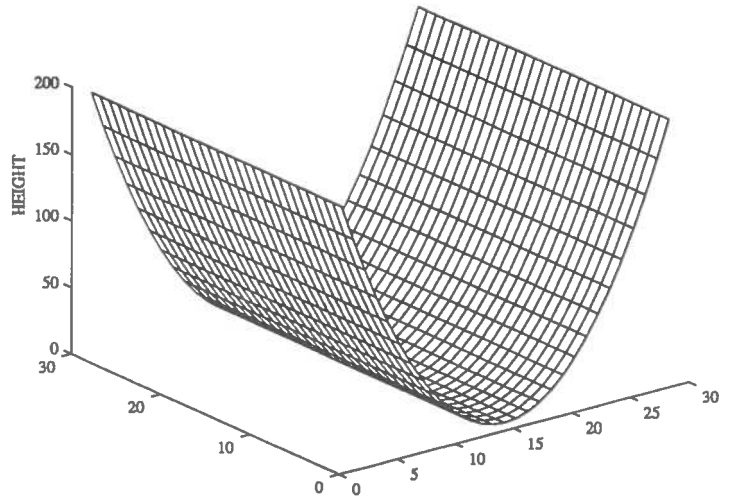

Figure 2.2: *Synthetic Test Valley.*

[1] Plotted using the *Matlab* package, ©The Maths Works, Inc.

As well as using just single–feature data, a small *"landscape"* was also artificially created. This data is just a convolution of the valley and ridge data, but without any orientation specification. This data would allow us to envisage any problems that might be encountered when different feature types, and thus differing visual force aspects, interact. *Figure 2.3* below, shows what this data looks like.

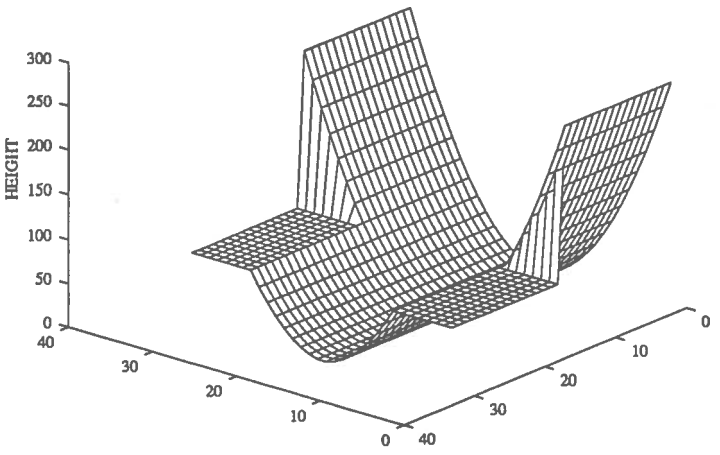

Figure 2.3: *Synthetic Landscape.*

2.3.2 Real Data

The real data to be used with the system comes from the Ordnance Survey who have transformed every 20km^2 O.S. National Grid Square into digital terrain data. Each grid square dataset is purchasable, under licence, at a cost of £25, with an additional price of £48 to cover the storage media. Funding for the **NG40** *&* **NG42** datasets came from the Department of Artificial Intelligence here at the University of Edinburgh.

The digital terrain data comes in National Transfer Format (NTF) and consists of header information detailing the region it covers, and a stream of height values, taken at 100 metre intervals. However, in this original format, it is not quite suitable for use and so needed to be adapted before it could be used.

The adaptation of the original digital terrain data was done by Dr. R. B. Fisher, the project supervisor. The new format is described in detail in *Appendix A*.

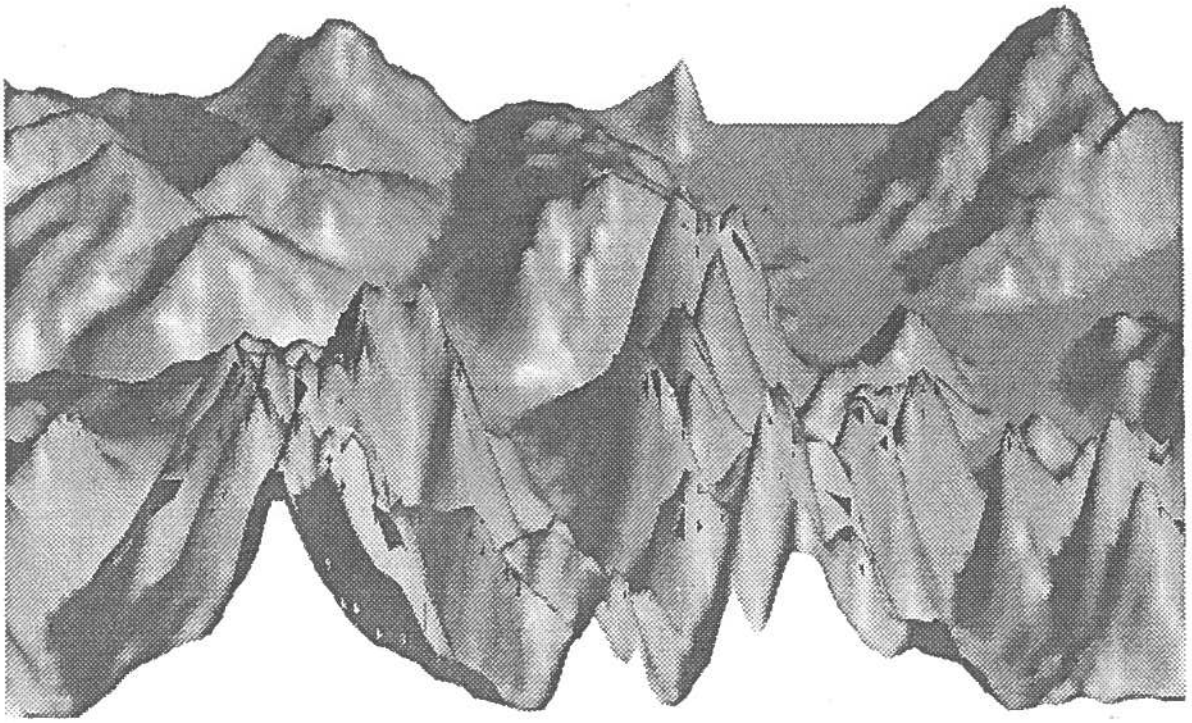

Figure 2.4: **NG42** *Region Landscape.*

Figure 2.4 shows what this data looks like in real life, however, it is only an approximation. The above image was produced from a 100×100 reduced size dataset, which took an average over every four values in the original 401×401 sized data, hence the jaggedness of the taller peaks.

2.3.3 Packaging the Data for Input

The data, be it real or artificial, is input into the system as a 2–dimensional (x, y) array of z height values. However, for the benefit of producing graphical results along each stage of the process, these arrays have been packaged up into HIPS image files.

Numerical value arrays which represent both the original data and subsequently processed information, make much better sense when displayed in graphical form. Packaging up these arrays into HIPS format allows the use of library routines, available within the Department, to display the information in a more meaningful way.

With regard to the real data, the images shown in the main body of the report all relate to the **NG42** region, produced at the various system stages. *Appendix B* collects together all the corresponding images for the **NG40** region.

2.4 Measuring System Performance

Using the artificial data, it is quite easy to gauge how effectively visual forces can be identified, albeit in these simple cases. The artificially created ridges and valleys are used to check that such features can be located and identified at any arbitrary orientation, while the test landscape allows the examination of problems that may arise when different features are closely interwoven.

However, obtaining a thorough measurement of the performance of the system, when used with real data representing a natural landscape, can only be achieved through collaboration with an expert — the landscape architect.

The initial decision to choose the region of the Isle of Skye as input into the system was made in conjunction with Mr. Simon Bell, a senior landscape architect with The Forestry Commission in Edinburgh. It was decided that a traditional, manual appraisal of the area, done by him, would be compared with any output to gauge how effectively the system identifies and quantifies the lines of visual force.

Chapter 3

Preprocessing the Data

The first main stage of building an automatic system capable of identifying the aspects of visual force is the preprocessing of the data. This stage converts the data from the format described in *Appendix A*, into one that the system can make the best use of. Preprocessing of the data occurs in two stages :

- Converting the data into HIPS image format.
- Smoothing to reduce the complexity.

3.1 HIPS Image Format

The HIPS image software, available within the Artificial Intelligence Department, provides a convenient way of packaging data into a format which allows it to be viewed graphically.

A HIPS image consists of a header which contains a number of fields describing the data, and the data itself. The most useful fields in the header describe the image size by specifying the number of rows and columns, and how the the data is represented — whether as bytes, integers, floating–point values or in some other format.

By representing data as an $N \times M$ array of a single type, a HIPS image can then be generated and displayed using the available software library routines. Treating the data as an array allows further arrays to be produced by the various stages of system processing. These too can be displayed as HIPS images which allows the user to see how the system progresses in determining the lines of visual force.

3.1.1 Representing Data as HIPS Images

Two HIPS images, representing the regions **NG40** and **NG42** are generated by reading the 401 height values in each of the 401 blocks into a 401×401 size array. The values entered into the array are integer values ranging from -100 metres to 1500 metres, however they are stored as floating–point values for reasons that will become apparent in §3.2.1. The appropriate header information is written and the two — header and array — are then appended together to form the stored HIPS image.

A contour map of the array representing the **NG42** region, as plotted by Matlab^{tm}, is shown in *Figure 3.1*. *Figure 3.2* shows the result of packaging and displaying the **NG42** data as a HIPS image. The image acts like a solid grey–level contour map where different heights are represented by differing intensities. Lower values are depicted by the darker intensities; sea–level pixels being represented by a distinguishable, uniform grey.

The synthetic test data is also packaged in a similar way. Each HIPS image has its own header which details the number of rows and columns in each image. This information can be accessed by the system, allowing it to adjust to the smaller test images accordingly.

3.2 Smoothing the Data

The data which represents the regions of the Isle of Skye were chosen so as to provide bold enough features with which to work. However, the nature of this data — and natural landscape in general — provides a level of detail that is too complex for producing good, distinct results. For this reason, the data needs to be smoothed into a much more "manageable" format.

Smoothing removes isolated points of noise which, in landscape data, correspond to isolated pixels whose values differ markedly from their closest neighbours. Since the digital terrain data is measured at 100 metres intervals, adjacent pixels could quite easily have very different height values. This could cause severe problems during later stages of processing and so such differences must be minimised.

3.2.1 Gaussian Convolution

Gaussian Convolution provides a method for smoothing by looking at the values within a local neighbourhood and adjusting the value of the central pixel accordingly.

The process involves convolving an operator mask with every image data point which calculates an average weighted sum of all the pixels the mask covers. The Gaussian Convolution mask is :

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 12 & 2 \\ 1 & 2 & 1 \end{bmatrix} \div 24$$

It is because of this average smoothing that the original integer height values are stored as floating–point numbers, as mentioned in §3.1.1.

Each application of the operator mask alters the value of a single pixel. However, the mask must be applied to every pixel in the image before any changes can be made, otherwise each successive application would be considering incorrect values. For this reason, the changed values are stored in a new data array, while the mask is applied to the original data values.

3.2.2 Number of Passes and the Problems

Obviously, a single pass of the operator over the entire image will not render much effect with regard to reducing the complexity of the data, and so a number of passes must be made. Each subsequent pass creates a new data array which, in turn, is used for the next operation.

But how many passes should we make? Every time we make a smoothing pass, we reduce the level of complexity — too few passes and we are no better off than we were originally; too many and we smooth the data to a level that it becomes useless.

There is no ideal number. The more smoothing passes made, the more smaller features tend to "disappear" from the data. This is a good thing in that ideally we should be trying to identify the larger features which contribute more to the visual forces, to a greater extent. However, the smaller features are just as important and provide a necessary detail, yet too few passes and the level of detail remains too confusing to work with.

The smoothing process is invaluable in reducing the complexity of the natural landscape data, however, not all pixels should be smoothed. Consider coastline sea–level pixels whose neighbouring height values could be measured at the top of a cliff. Applying the Gaussian operator to these pixels would effectively cause a landslide into the sea as the average weighting calculations would effect a continual increase to an initially zero value. Further smoothing passes would cause this landslide effect to extend further and further into the sea — clearly an undesirable side–effect — and so, sea–level pixels are not considered for smoothing.

3.3 Results Summary

As mentioned earlier, it is difficult to decide on exactly how many passes of the smoothing operator to make. Clearly, a decision on the optimal number needed to maintain sufficient detail without causing too much confusion, can only be made through hindsight by viewing the results achieved at later stages.

A command line parameter, supplied to data conversion program `preprocess.cxx`, allows the production of datasets which are smoothed to varying degrees. These different datasets are then used to explore how smoothing affects the later stages of processing and how many smoothing passes can be tolerated in order to achieve the best results.

The following page depicts the original **NG42** region as it comes without any smoothing, through both a contour map *(Figure 3.1)* and the HIPS range image *(Figure 3.2)*. Having smoothed the image both 20 and 80 times — contour map *Figures 3.3 & 3.4* — I have drawn the conclusion, from subsequent processing, that the data smoothed 40 times reduces the noise level factor considerably yet maintains a sufficient level of detail to be used constructively. The contour map and view of this data are depicted in *Figures 3.5 & 3.6.*

Appendix C.1 contains the code `preprocess.cxx` which allows the user to specify the number of passes required to smooth the data. It then creates the required HIPS format image using the `array2hips.cxx` routine. This routine is used extensively throughout all stages — the code can be found in *Appendix C.4.*

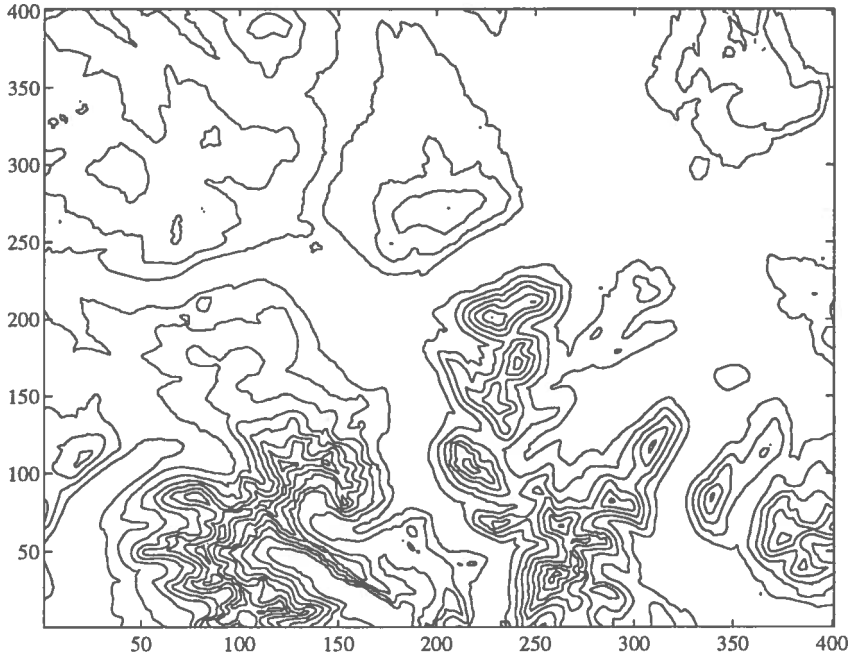

Figure 3.1: **NG42** *Contour Map (Original).*

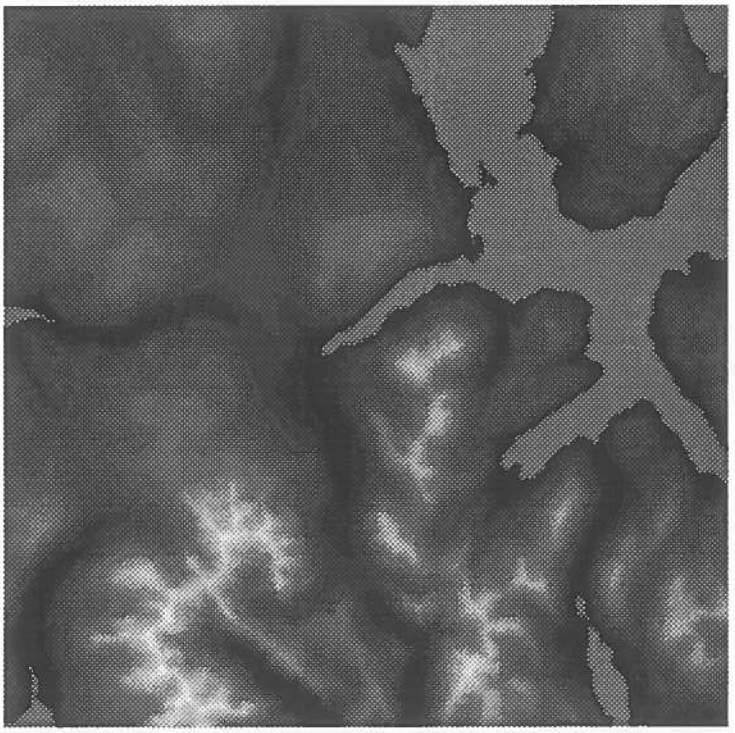

Figure 3.2: **NG42** *Range Image (Original).*

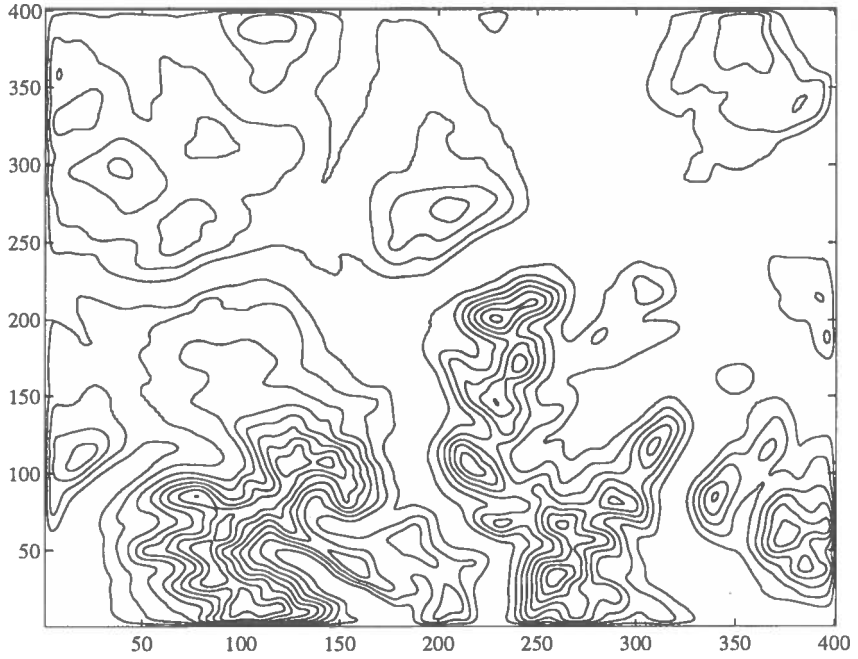

Figure 3.3: **NG42** *Contour Map (Smoothed* $\times 20$*).*

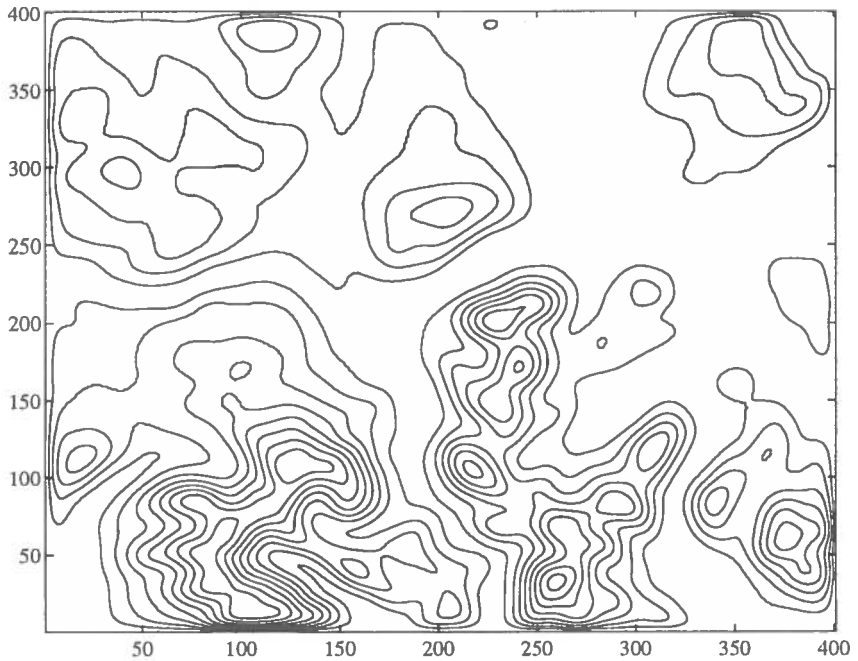

Figure 3.4: **NG42** *Contour Map (Smoothed* $\times 80$*).*

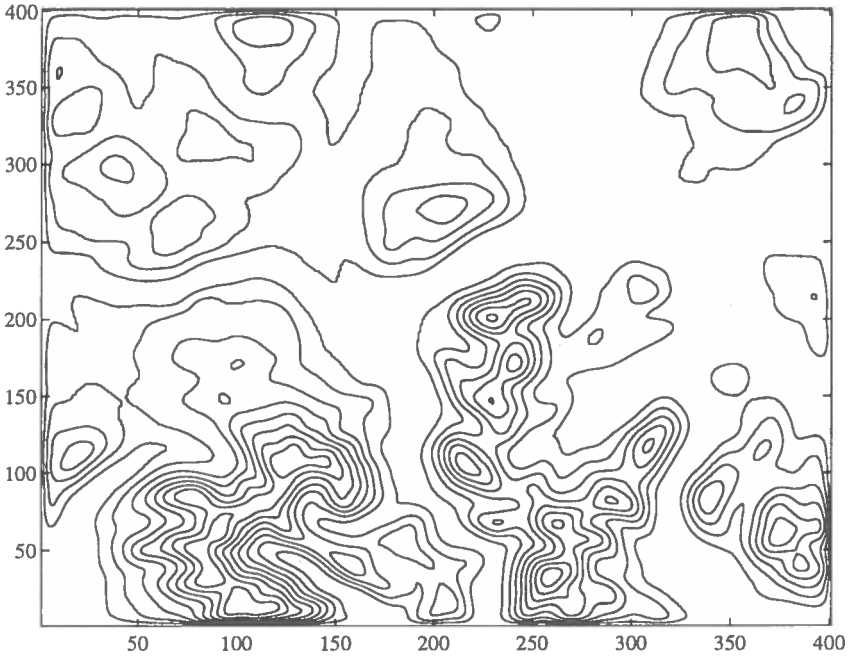

Figure 3.5: **NG42** *Contour Map (Smoothed* $\times 40$*).*

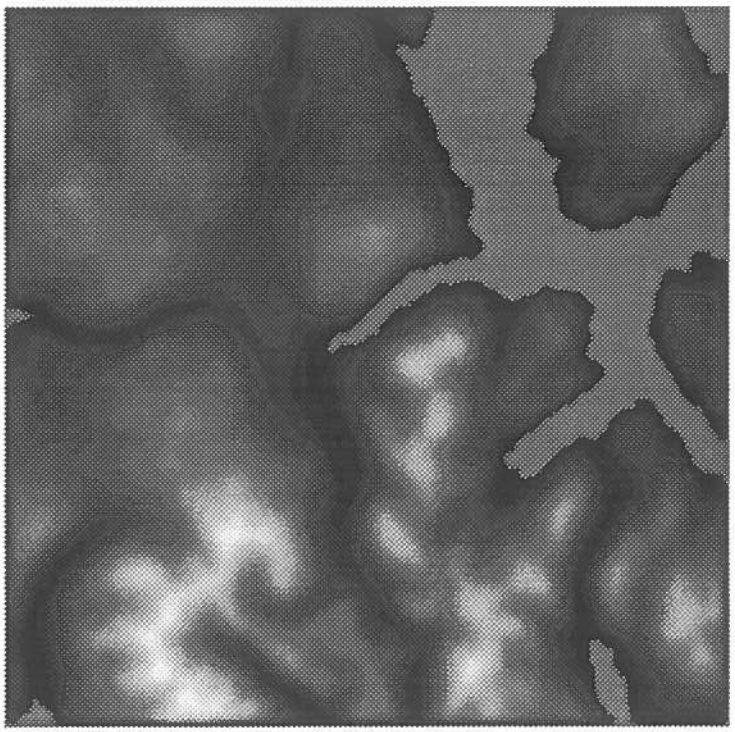

Figure 3.6: **NG42** *Range Image (Smoothed* $\times 40$*).*

Chapter 4

Calculating Local Curvature, Shape and Orientation

4.1 Fitting a Bi-Quadratic Surface

The way in which to determine the local shape of an individual pixel is to estimate the shape of a model surface which best *fits* the pixel and its neighbouring data points. The coefficients of the parametric equation which describes this surface can then be combined, according to known mathematical equations, to give both the local shape and orientation information that is needed.

The method described below fits a bi–quadratic surface about every image pixel and calculates the corresponding coefficients for the representative parametric equation. The equation for a bi–quadratic surface is :

$$z = a + bx + cy + dx^2 + ey^2 + fxy \tag{4.1}$$

which yields the six coefficients ($a \rightarrow f$). However, the subsequent formulae for calculating the local pixel shape and orientation are easier to describe in terms of the derivatives of the surface :

$$z_x = b + 2dx + fy, \quad z_y = c + 2ey + fx,$$

$$z_{xx} = 2d, \quad z_{yy} = 2e, \quad z_{xy} = f \tag{4.2}$$

4.1.1 A Window on Neighbourhood Regions

Estimating a surface fit can only be done with a number of data points and so the neighbourhood of a pixel must be defined. Here, this neighbourhood is a user–defined $N \times N$ pixel size window[1], centered around the data point in question. The data values within this window are then used to estimate the surface shape.

Greater accuracy in estimating the fit can be achieved by using a larger window, however, a tradeoff exists. With any window, we lose a $\frac{1}{2}(N-1)$ border of pixels around the image as there, the $N \times N$ window lies partially outside the dataset and we cannot determine the elevation values which lie outside the data currently being processed. Larger windows will therefore have wider borders and we lose more image pixels.

Further improvements in accuracy can also be made (although this is not always the case — see Ch.7, §2) by transposing the window to the origin so that the pixel coordinates become $[x = 0, y = 0]$. This eases the calculations involved with respect to machine dependent floating–point inaccuracies and changes the x and y first derivatives of the surface from equation *(4.2)* into :

$$z_x = b, \qquad z_y = c \tag{4.3}$$

4.1.2 The Solution : Singular Value Decomposition

Given the information contained in the window, how then can the bi–quadratic surface fit be estimated to yield the necessary coefficients? A well known method used for this, the parametric modelling of data, is singular value decomposition (SVD). Mathematical library routines for implementing SVD exist and are called by the system to solve the many dimensional minimisation problem which estimating the parameters of a surface presents.

Using SVD to estimate these parameters is a robust and well–documented solution that improves on other approaches such as the least–squares method. See **[Pres 88]** for a general discussion on how, and why, singular value decomposition is implemented, along with the library routines used.

[1]The default window size is 3×3.

4.2 Using Mean and Gaussian Curvatures

4.2.1 H & K Calculation

Having fitted a bi–quadratic surface around every image pixel, the corresponding Mean and Gaussian curvatures (see [I&J 85]) can be calculated from the associated parametric coefficients. The Mean Curvature, **H**, for a given (x, y) pixel is given by :

$$\mathbf{H} = \frac{2d + 2e + 2d(c + 2ey + fx)^2 + 2e(b + 2dx + fy)^2 - 2f(c + 2ey + fx)(b + 2dx + fy)}{2(1 + (c + 2ey + fx)^2 + (b + 2dx + fy)^2)^{\frac{3}{2}}} \tag{4.4}$$

with the Gaussian Curvature, **K**, by :

$$\mathbf{K} = \frac{4de - f^2}{(1 + (c + 2ey + fx)^2 + (b + 2dx + fy)^2)^2} \tag{4.5}$$

However, the relevant equations can be expressed more concisely from the surface derivatives given in equations *(4.2)* and *(4.3)*, and these are in fact the equations used by the system :

$$\mathbf{H} = \frac{z_{xx} + z_{yy} + z_{xx}z_y^2 + z_{yy}z_x^2 - 2z_x z_y z_{xy}}{2(1 + z_x^2 + z_y^2)^{\frac{3}{2}}} \tag{4.6}$$

$$\mathbf{K} = \frac{z_{xx}z_{yy} - z_{xy}^2}{(1 + z_x^2 + z_y^2)^2} \tag{4.7}$$

4.2.2 Local Shape Classification

The shape of any surface can be classified based on the signs of the Mean and Gaussian curvatures for that surface, as shown by [B&J 85]. Nine different combinations of the signs of **H** and **K** being positive, negative or equal to zero, result in eight distinct classification types with the 9^{th} possibility, $[\mathbf{H} = 0, \mathbf{K} > 0]$, never arising (but see §4.2.3). The possible classifications are shown in *Figure 4.1.*

Using the bi–quadratic surface fit, the shape at each image data point can be estimated and we can then calculate the local shape classification. This is done after calculating the **H** and **K** values for the individual pixels, however, for reasons that will become clear in §4.2.4, not all possible classifications are applied.

H	K	CLASS	GENERAL
H < 0	K > 0	Peak	Ridge
H < 0	K = 0	Ridge	
H < 0	K < 0	Saddle Ridge	
H = 0	K > 0	Unclassified	Uninteresting
H = 0	K = 0	Plane	
H = 0	K < 0	Minimal	
H > 0	K < 0	Saddle Valley	Valley
H > 0	K = 0	Valley	
H > 0	K > 0	Pit	

Figure 4.1: *Classification Types.*

4.2.3 Thresholding

Local shape classification based on the values of the Mean and Gaussian Curvatures is subject to two problems. The first is noise, where random small perturbations can change local shape. The second is that flat surfaces, ie. surfaces with zero **H** and **K** values, very rarely occur naturally in the real world.

As a results of these factors, the "exactly zero" constraint used in classification must be relaxed. This is done by the user supplying a tolerance value ε which allows approximations to flat surfaces to be made while also removing the possibility of errors arising due to noise. Effectively, this thresholding procedure reduces values, in the range limited by $\pm\varepsilon$, to zero for the purposes of classification, ie. :

$$\text{value} \in [-\varepsilon, \ldots, +\varepsilon] \Rightarrow \text{value} = 0 \tag{4.8}$$

Since we are evaluating separately whether the two different values are equal to zero or not, a single threshold cannot represent both required deviations — the **H** and **K** values are independent. **[Cai2 90]** discusses the relationship between two thresholds, $\mathcal{H}_\varepsilon$ for the Mean curvature values and $\mathcal{K}_\varepsilon$ to be used with the Gaussian values.

Taking this into account, the user specified threshold is treated as $\mathcal{H}_\varepsilon$ and the corresponding $\mathcal{K}_\varepsilon$ is calculated according equation *(4.9)* :

$$\mathcal{K}_\varepsilon = (2 \times (\frac{1}{M} \sum_{i=1}^{M} H_i) \times \mathcal{H}_\varepsilon) + \mathcal{H}_\varepsilon^2 \tag{4.9}$$

where M is the number of data points in the image being processed.

Unfortunately, by using thresholds it is then possible for the 9^{th} (invalid) classification to arise when $[\mathbf{H} = 0, \mathbf{K} > 0] \pm\varepsilon$. Luckily however, we are not concerned with this, or either of the two surface classification types which do not contribute to visual forces — planes and minimals. We only need to identify the ridge and valley types.

4.2.4 Generalising Shape Classification

Although the preceding discussion details what can be achieved and the importance of using both curvature descriptions, hindsight has shown that Gaussian curvature information is actually unnecessary. Local shape classification can be generalised into simplified classes which detail enough information needed for the purposes of detecting visual forces in a natural landscape.

The reason for this is that the lines of visual force are, as described earlier, inherently tied to the convex and concave features in general — there is no need to distinguish between different types of each. Peaks, ridges and saddle ridges are all examples of convex features; pits, valleys and saddle valleys of concave features.

By observing the sign of the Mean curvature values in *Figure 4.1*, it can be seen that negative **H** values correspond to positive cylinders — the ridge–like features — while negative cylinders, the valley–like features, have positive **H** values. Flat surfaces (planes) and minimals, which do not contribute to the aspects of visual force, have a zero **H** value and can thus be "ignored." However, for the reasons given in §4.2.3, a threshold still needs to be used and this remains as the user–specified ε, input to the system.

It is interesting to note that if **H** was actually calculated as $-\mathbf{H}$, then the more intuitive association of positive cylinders with positive **H** values, and negative cylinders with negative values, could be used. However, the choice of the sign of curvature is arbitrary and these standard formula produce negative values for positive cylinders.

4.3 Local Pixel Orientation

Having successfully determined local shape information for every image pixel, it is then necessary to obtain the local orientation details of each. Mean and Gaussian curvatures tell nothing about the orientation or degree of curvature of a surface directly. This information can only be obtained when we know something about the two principal curvatures — the direction of the minimum curvature, which runs along a surface, can give an indication of the orientation of the ridge or valley; the maximum curvature across a surface allowing the degree of concavity or convexity to be measured.

4.3.1 Calculating Principal Curvature Magnitudes

Once again, the coefficients of the parametric equation describing the bi–quadratic surface fit can be used in calculating the maximum and minimum curvatures. As before, the equations are best described in terms of the surface derivatives given in equations *(4.2)* and *(4.3)*.

First of all, I define the Weingarten mapping matrix, **W**, (see **[B&J 85]**) :

$$\mathbf{W} = \frac{1}{EG - F^2} \begin{pmatrix} GL - FM & GM - FN \\ EM - FL & EN - FM \end{pmatrix} = \frac{1}{S} \begin{pmatrix} A & B \\ C & D \end{pmatrix} \tag{4.10}$$

where :

$$E = 1 + z_x^2, \qquad F = z_x z_y, \qquad G = 1 + z_y^2,$$

$$L = \frac{z_{xx}}{\sqrt{1 + z_x^2 + z_y^2}}, \qquad M = \frac{z_{xy}}{\sqrt{1 + z_x^2 + z_y^2}}, \qquad N = \frac{z_{yy}}{\sqrt{1 + z_x^2 + z_y^2}}$$

The matrix **W** has the property that its eigenvalues are the principal curvatures, C_p, while the eigenvectors give the (u, v) curvature directions in the (x, y) plane, ie. :

$$\mathbf{W} \begin{pmatrix} u_p \\ v_p \end{pmatrix} = C_p \begin{pmatrix} u_p \\ v_p \end{pmatrix} \tag{4.11}$$

By using equation *(4.11)* it is possible to calculate the two principal curvature direction vectors in two dimensions, of which the minimum curvature vector gives the local orientation information. However, knowing these direction vectors does not allow us to

distinguish which corresponds to which principal curvature. The curvature magnitudes enable us to make the distinction, but the (u, v) components alone do not give enough information to calculate the true magnitude values.

A different approach, which first calculates the principal curvature magnitudes is used. From this, the smaller magnitude can then be determined and used in calculating the correct direction vector. By combining the Weingarten matrix elements from equation *(4.10)* a formula can be expressed to calculate the two magnitudes. The formula used, derived with the help of Andrew Fitzgibbon, is :

$$C_p = \frac{1}{2}\left[(D + A) \pm \sqrt{(D - A)^2 + 4BC}\right] \tag{4.12}$$

4.3.2 Orientation from Principal Curvatures

By distinguishing between the two principal curvatures, a local orientation can then be determined. The orientation required is represented by the principal curvature which has the smaller magnitude.

Calculating the (u, v) components of the minimum curvature direction vector can be done, again using the matrix elements from **W**. The formula :

$$\frac{v}{u} = \frac{(S \times C_{min_p}) - A}{B} \tag{4.13}$$

represents the valley / ridge direction vector, orientated at an angle ϕ anti–clockwise from the y–axis, ie. :

$$\tan(\phi) = \frac{v}{u}$$

4.4 Cosine Shading

From the bi–quadratic surface parameters, we can also generate a cosine shaded image which makes the structure of the range image (Ch.3, §1.1, *Figures 3.2 & 3.6*) much clearer to the human observer. In a cosine shaded image, the degree of shading of a surface is related to the cosine of the angle the surface normal makes with the light source vector, as stated by Lambert's Cosine Law for diffuse reflection — see **[Fole 91]** for further details.

For the range images, there is only a single viewing direction — from directly overhead — with the light source considered to be at infinity. This allows the formula for calculating the cosine shaded value of an image pixel to be expressed simply in terms of the surface derivatives :

$$\frac{1}{\sqrt{1 + z_x^2 + z_y^2}} \tag{4.14}$$

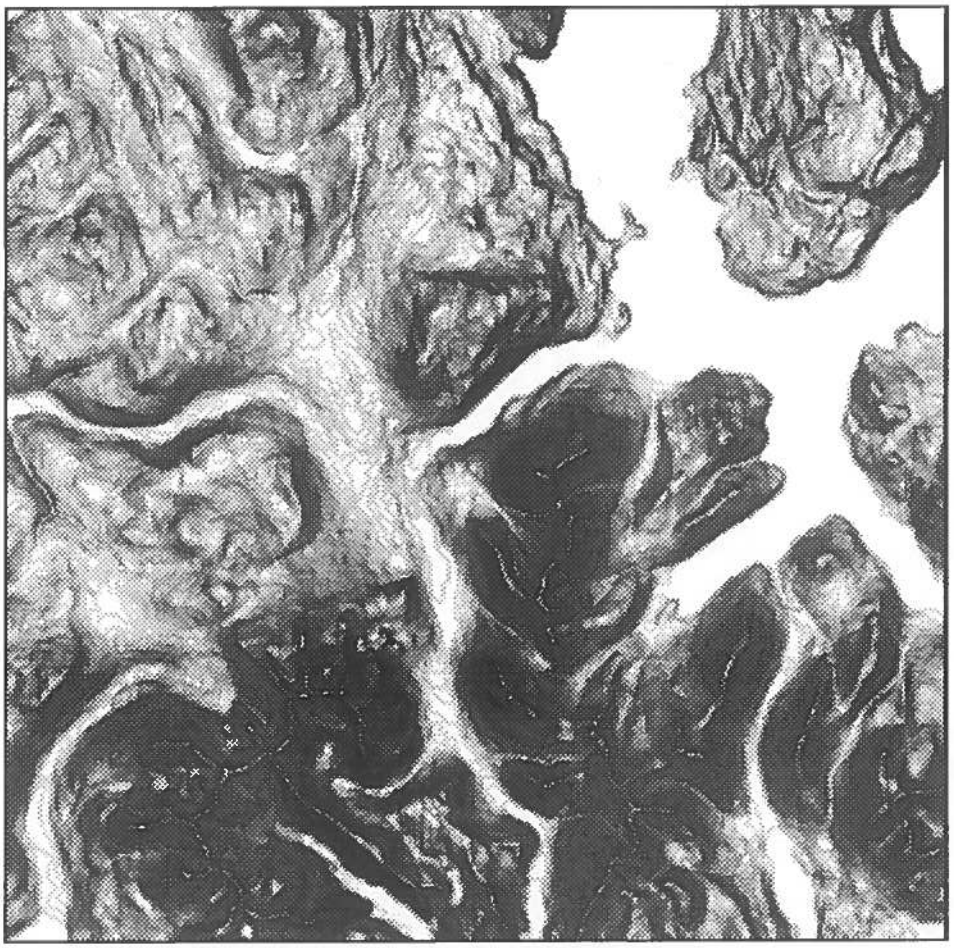

Figure 4.2: **NG42** *Cosine Shaded Image.*

Figure 4.2 shows the cosine shaded image corresponding to the original **NG42** range data. The greater the angle between a surface normal and the light source vector, the smaller the cosine and hence, the cosine shaded value calculated by equation *(4.14)* increases. This results in surfaces with normals pointing straight–up towards the viewpoint (ie. out of the page), such as the sea, appearing brightest (whitest) in the image, with steeper faced surfaces appearing darker.

4.5 Results Summary

The entire process of calculating local curvature, shape and orientation information, as described in the previous sections, is contained in the second program section which uses the various SVD library routines mentioned. The *pseudo*–algorithm below summarises the whole process.

Pseudo–algorithm :

```
for every image pixel do
{
     extract NxN window;
     generate bi-quadratic parameters using SVD;
     calculate Mean curvature, H, value;
     determine orientation from minimum curvature;
     calculate cosine shaded value;
 }
 dump H and orientation details for later use;
 generate cosine shaded image;
```

Generating an **H** classification for every image pixel is essential as it allows us to distinguish between the two main features types needed to identify the different aspects of visual force. The classification generalises local shape information into one of three categories : ridge, valley or other. These classification types are represented in the following figures as black, white and grey respectively.

Figure 4.3 highlights how the classification works with respect to the 32×32 testimage shown in *Figure 2.3 (p.8)*. However, from this we can see the difficulty arising in trying to classify flat surfaces correctly without using a threshold, but, by using $\varepsilon = 0.1$ in *Figure 4.4*, a better classification can be generated.

It has become obvious from using the thresholding procedure mentioned in §4.2.3, that regions of maximum curvature — the ridge apexes and valley minima — have mean curvature values which deviate the greatest from zero. This fact allows us to broaden the range of values which classify surfaces as being non–contributory to visual force, enough for us to produce **H** classifications which tend towards depicting just the valley

minima and ridge apexes. Unfortunately though, choosing a threshold which identifies only the maxima and minima is difficult (see *Figures 4.4* and *4.5*).

Figure 4.5 uses a smaller threshold than in *Figure 4.4* of $\varepsilon = 0.01$. This reduces the valley and ridge classified points into smaller regions, more associated with the corresponding minima and apexes.

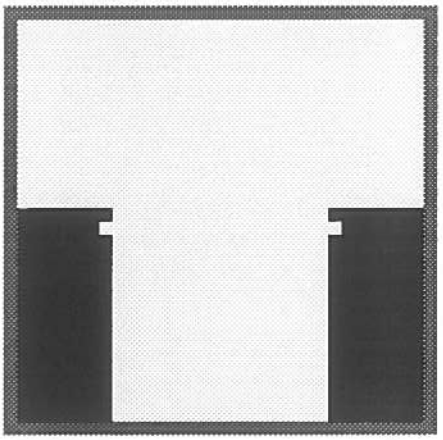

Figure 4.3: *Test Image Classification (No Threshold).*

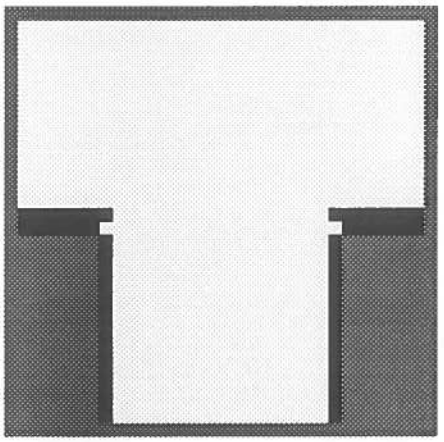

Figure 4.4: *Test Image Classification (Threshold = 0.1).*

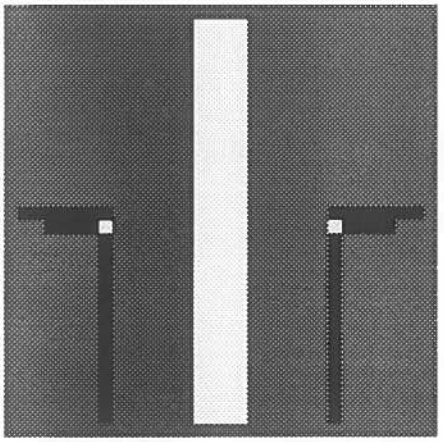

Figure 4.5: *Test Image Classification (Threshold = 0.01).*
white ⇒ valley; black ⇒ ridge; grey ⇒ flat.

Figures 4.6 → 4.10 show the HIPS images which represent possible **H** classifications for the **NG42** region.

Figure 4.6 shows the level of complexity we are dealing with in the original data while *Figures 4.7 & 4.8* show the benefits of smoothing this data 40 and 80 times respectively. Although the smoother images allow us to see patterns emerging in the grouping of similar feature types, more so in *Figure 4.8*, specific features are not obviously apparent, nor are the feature maxima and minima which determine the lines of visual force.

Figure 4.9 shows the original data, classified with a tolerance value of $\varepsilon = 0.1$, and from this we can begin to visualise the distinct feature tracks emerging (ridges in black; valleys in white). But still, the level of detail is too complex. The classified image produced using the data smoothed 40 times is shown in *Figure 4.10*. Here, again using a threshold value of $\varepsilon = 0.1$, the distinct classification of ridge apexes and valley minima is more apparent.

It may seem from *Figures 4.9* and *4.10* that we could proceed to calculate the visual saliency directly from these results, however, this is not the case. Not only is it difficult to chose an exact threshold which isolates the ridges apexes and valley minima alone — as in *Figure 4.4* which uses the same threshold — but, by specifying the tolerance value, we also classify surfaces with mean curvature values within the $\pm\varepsilon$ bounded range, as not contributing to visual force.

In otherwords, this method only allows the identification of ridges and valleys with $\mathbf{H} < 0.1$ and $\mathbf{H} > 0.1$ respectively. More gently sloping terrain may produce adequate feature definitions which have **H** values outside these constraints, and so will not be picked–up by thresholding.

In the case of the **NG42** and **NG40** regions being used, chosen for the boldness of features, the rare existence of these shallower features do not present a major problem. However, for less "rugged" landscapes the problem cannot be ignored. A better method for identifying <u>all</u> ridge apexes and valley minima is needed. Chapter 5 addresses this problem.

The code, `process.cxx`, and the routines for calculating the Mean curvature values (`hkcode.cxx`) used for classification purposes, and the orientation information, (`orient.cxx`), are included in their entirety in *Appendix C.2.*

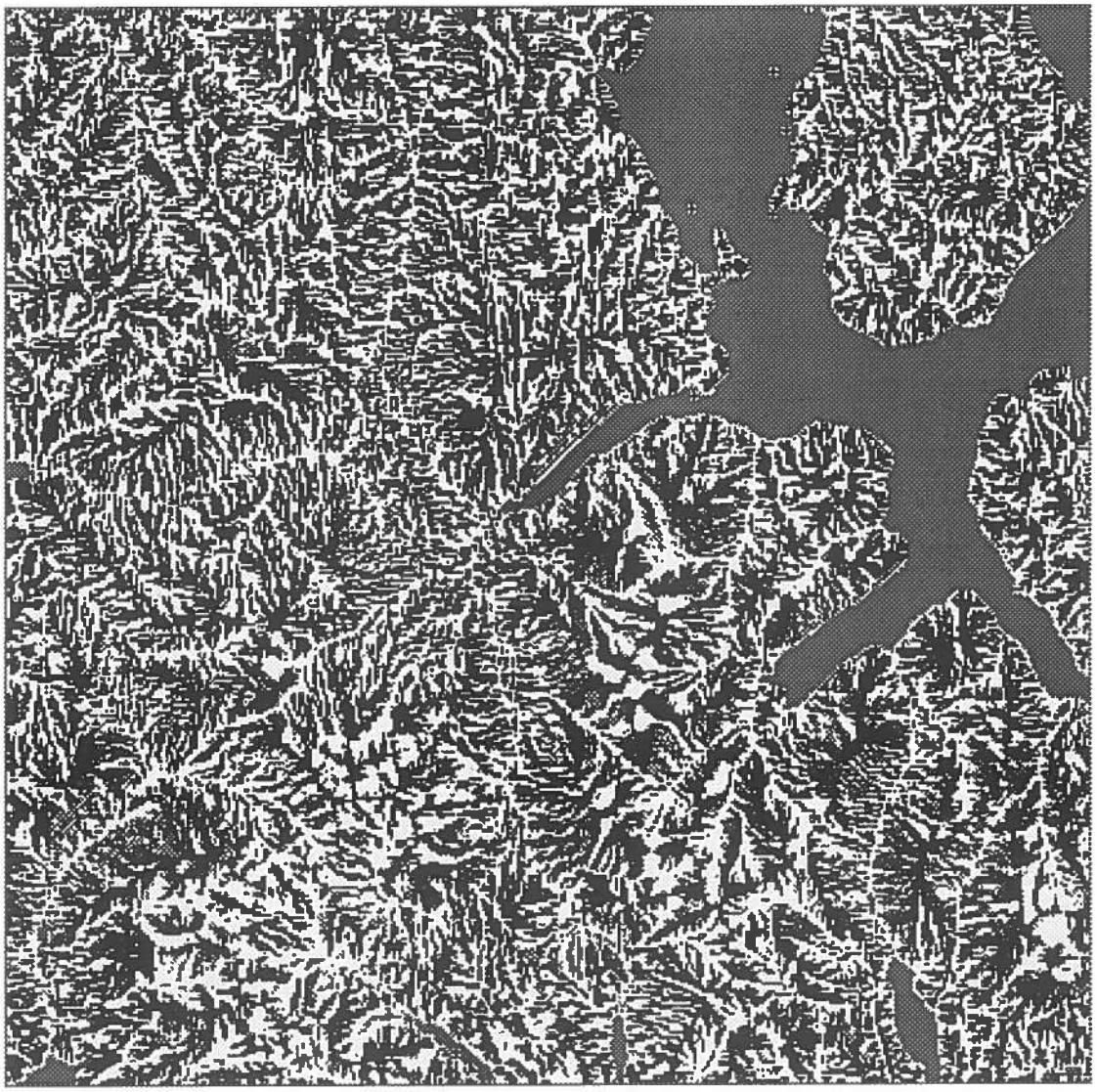

Figure 4.6: **NG42** *Classified Image (Original).*
white ⇒ *valley; black* ⇒ *ridge; grey* ⇒ *flat.*

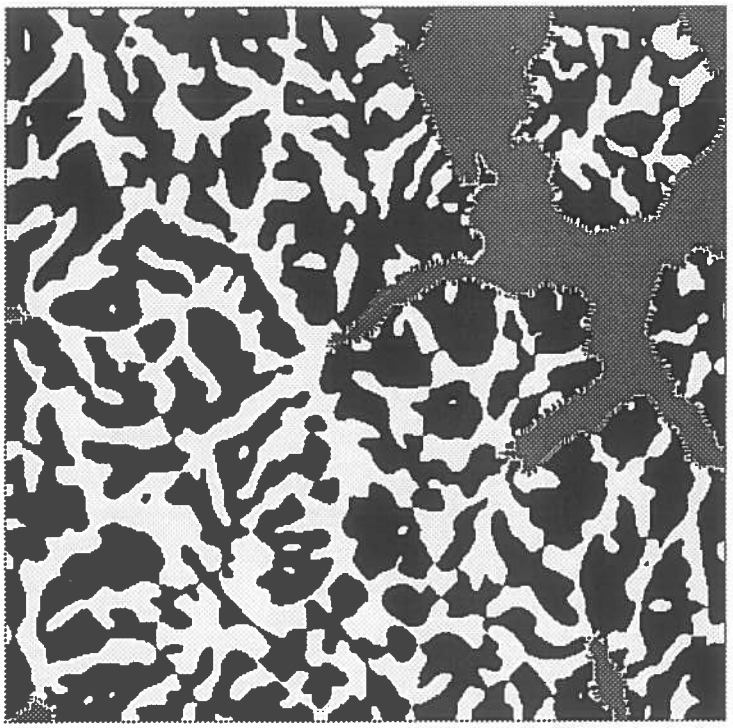

Figure 4.7: **NG42** *Classified Image (Smoothed* $\times 40$*).*

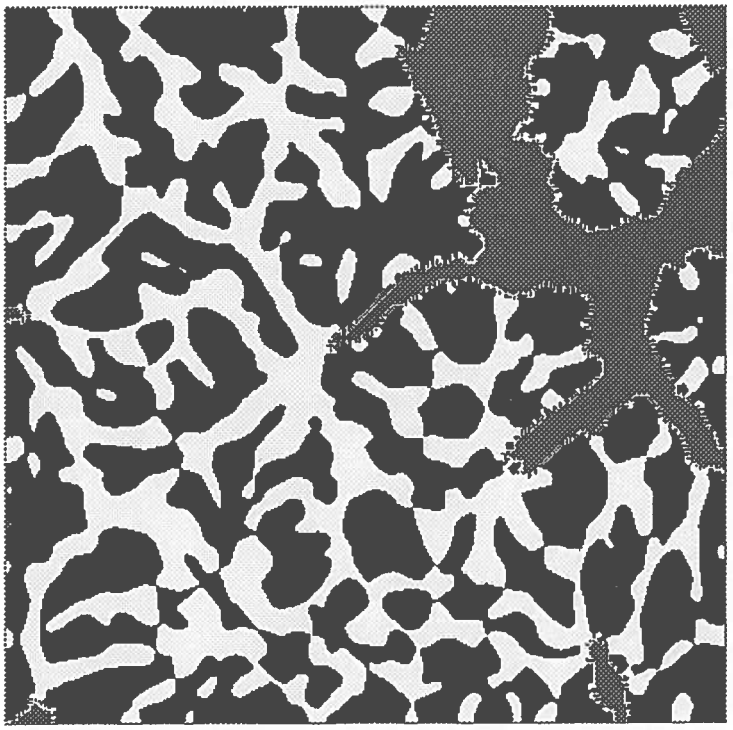

Figure 4.8: **NG42** *Classified Image (Smoothed* $\times 80$*).*
white $\Rightarrow$ *valley; black* $\Rightarrow$ *ridge; grey* $\Rightarrow$ *flat.*

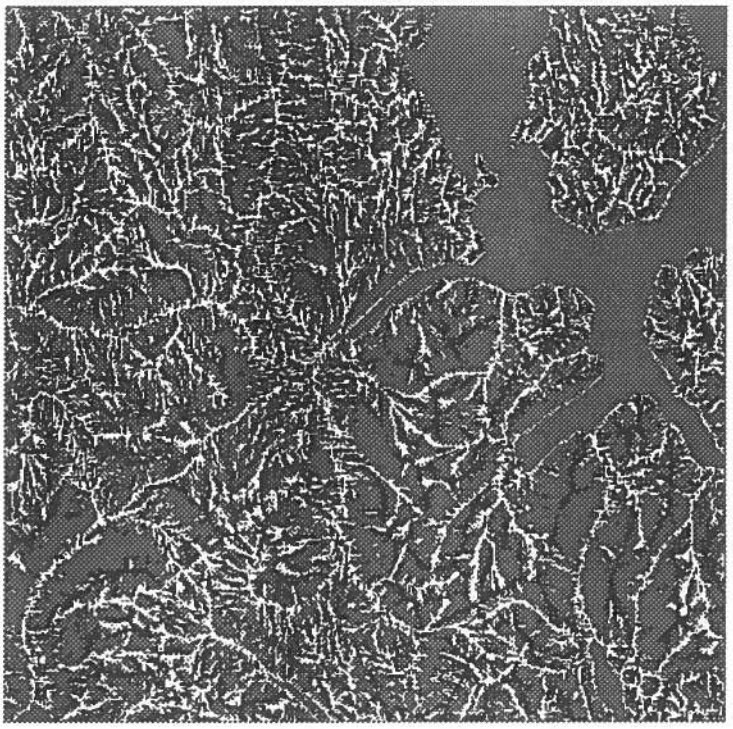

Figure 4.9: *Thresholded* **NG42** *Classified Image (Original).*

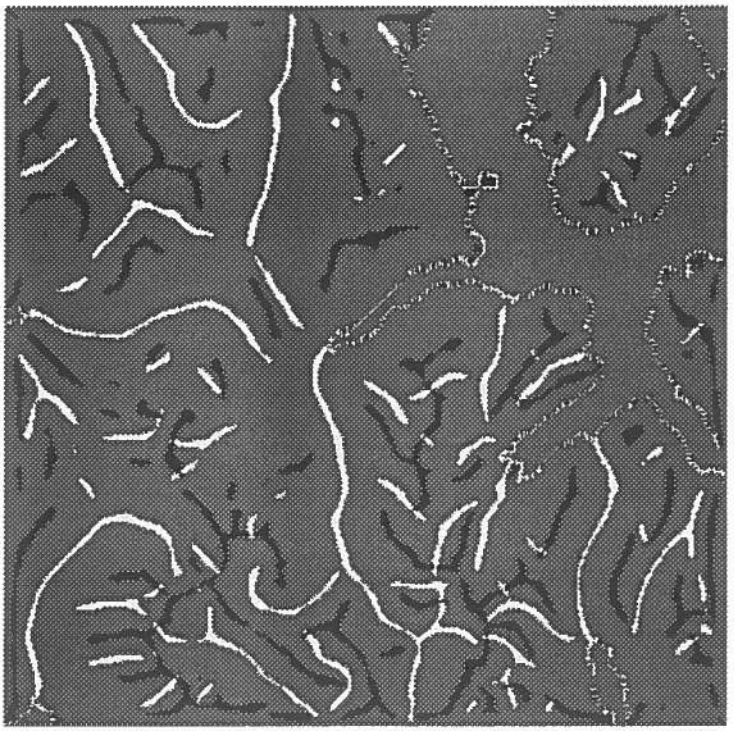

Figure 4.10: *Thresholded* **NG42** *Classified Image (Smoothed* $\times 40$*).*
white $\Rightarrow$ *valley; black* $\Rightarrow$ *ridge; grey* $\Rightarrow$ *flat.*

Chapter 5

Feature Identification & Tracking

Having concluded in the previous chapter that classification based on thresholded Mean curvature values is prone to "ignoring" shallow sloped features, a more reliable method is needed. An adapted version of the standard Canny non–maximal suppression algorithm ([**Cann 86**]) used in image edge detection, provides a basis for such a solution which enables the finding of all possible ridge apexes and valley minima.

5.1 Non–Maximal Suppression

A non–maximal suppression algorithm uses orientation information in deciding whether a pixel should be suppressed or not. The decision is made based on the values of two immediate neighbours on either side of the pixel under consideration, chosen depending on the pixel orientation. If the pixel's value is strictly less than both the neighbouring values (ie. not maximal), then that pixel is suppressed.

Using the original, non–thresholded **H** classification from Ch.4, §2, the above method can be used to identify both ridge apexes and valley minima. Identifying minima is done by checking that the height value of a classified valley pixel is less than the neighbouring values along the line of maximum curvature. The orientation of the maximum curvature, which represents the curvature across a feature, is easily determined being perpendicular to the minimum curvature — the earlier calculated pixel orientation (Ch.4, §3). If the value is not minimal, then it is suppressed. With classified ridge pixels, the opposite occurs — height values must be maximal if they are to survive suppression.

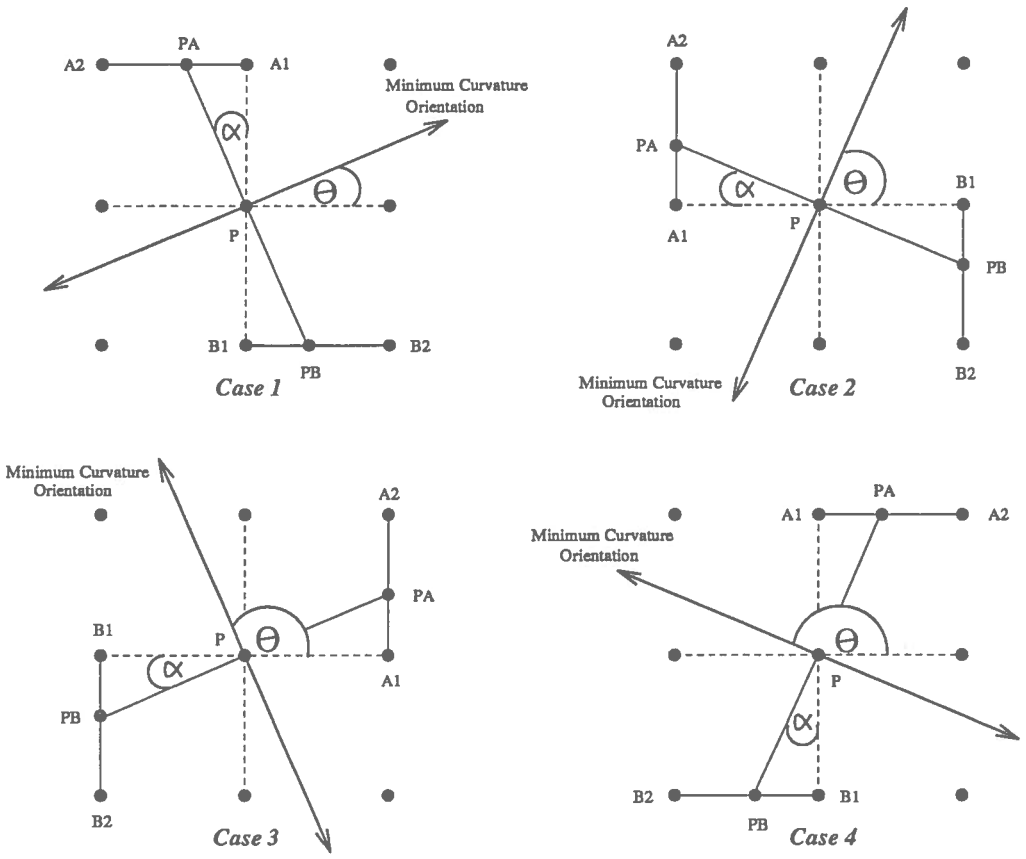

Figure 5.1: *The Four Orientation Dependant Cases.*

5.1.1 Interpolation

However, using non–maximal suppression is not so straight forward. The maximum curvature of a pixel may not be orientated at an ideal angle so that its line goes through exact data points, it may fall between pixels. For this reason, a method of interpolating values must be used to estimate values which are otherwise unknown.

Four cases can be envisaged — shown in *Figure 5.1* above — depending on the angle θ, the minimum curvature makes with the x–axis :

- Case [1] : $0^o \leq \theta \leq 45^o$
- Case [2] : $45^o < \theta \leq 90^o$
- Case [3] : $90^o < \theta \leq 135^o$
- Case [4] : $135^o < \theta < 180^o$

Having calculated the orientation of each pixel, a decision can be made on which of the four cases holds. From this, we can use the correct pixel values, $A1_{val}$ & $A2_{val}$ and $B1_{val}$ & $B2_{val}$, in order to calculate the interpolated values, PA_{val} and PB_{val} respectively. These two values lie along the line of maximum curvature and are used in the comparison with P_{val} to determine whether P is to be suppressed or not.

Each interpolated point is calculated as a distance weighted sum of the two pixels it lies between. All pixels are unit distance apart, and so these distances can be calculated based on the tan of the angle α which subtends the corresponding line segment :

$$|A1\ PA| = |B1\ PB| = \tan(\alpha) \tag{5.1}$$

$$|A2\ PA| = |B2\ PB| = (1 - \tan(\alpha)) \tag{5.2}$$

Angle α is related, case dependent, to the orientation angle θ by way of similar angles between the axes and the perpendicular curvatures — Case [1] $\alpha = \theta$; Case [2] $\alpha = (90 - \theta)$; Case [3] $\alpha = (\theta - 90)$; Case [4] $\alpha = (180 - \theta)$.

The interpolated value PA_{val} is calculated according to equation *(5.3)* below which sums the values of $A1_{val}$ & $A2_{val}$, weighted by the distances from PA as calculated by equations *(5.1)* and *(5.2)*. Calculation of PB_{val} occurs in similar fashion.

$$PA_{val} = (1 - \tan(\alpha))A1_{val} + \tan(\alpha)A2_{val} \tag{5.3}$$

$$PB_{val} = (1 - \tan(\alpha))B1_{val} + \tan(\alpha)B2_{val} \tag{5.4}$$

Each valley pixel is compared with interpolated values PA_{val} and PB_{val} to see if it is less than both, while a ridge pixel value must be greater. If it is, then the pixel is considered to be a minima (or apex) and need not be suppressed.

5.1.2 Relaxing the Suppression Constraint

Non–maximal suppression works well in that it considers every possible point as being a candidate apex (if classified as a ridge pixel), or minima (if a valley pixel), irrespective of the feature strength or slope. However problems do exist which must be taken care of in order to produce an image that contains all correct maxima and minima.

One problem is that of strictness — how strictly should we enforce the constraint that minima must be less than, and maxima greater than, both interpolated values? It is possible that the apex of a ridge (or valley minima) may cover two, equal height image pixels which are classed as ridge pixels. In this case, both pixels must not be suppressed, even though neither is strictly maximal. Therefore, the constraints must be relaxed to allow maxima to be greater than *or equal*, and minima to be less than *or equal*, to the interpolated values.

However, relaxing the constraints gives rise to another problem as the surface fit procedure classifies coastline sea–level pixels as being valley instances. Such a pixel will have a lower height than its coastline neighbour and a height equal to its sea–level neighbour, so, it will be considered a minima. Although this is a problem caused by the surface fit and subsequent classification, it is much easier to solve during the suppression stage by automatically suppressing all sea–level pixels — determined by their zero height values.

5.2 False Suppression at Junctions

Another major problem inherent in using non–maximal suppression, as highlighted in [**DuLi 89**], is the fact that the method fails to connect features at junctions where they should otherwise be connected. This is because the apex height value of one ridge is often less than that of another where the two (or more) form a junction. According to the algorithm, such a pixel is suppressed as it is not locally maximal in the neighbourhood of the other feature apex. This *false* suppression is undesirable.

5.2.1 Reconnecting Junctions

[**DuLi 89**] suggests an approach to solving the connectivity problem. In the case of falsely suppressed ridges, a probe is temporarily extended from an apex endpoint in the direction of that pixel's orientation (minimum curvature). Provided height values along the probe continue to increase and the relative orientation remains within a specified deviation from pixel to pixel, the probe can be treated as an extension to the apex which is now known to have falsely ended as a result of suppression.

However, an extended ridge apex probe must be carefully examined to ensure that it does not cross classified valley pixels or previously identified minima. The argument for extending the valley minima is equivalent.

5.2.2 Results and Associated Problems

Valley minima and ridge apexes are identified in the image data by considering every point for suppression. All valley pixels which are non–minimal, and ridge pixels which are non-maximal, are suppressed, leaving identified feature tracks in the image.

Figure 5.2 below depicts the identified valley minima in the synthetic testimage (*Figure 2.3, p.14*) with the ridge apexes shown in *Figure 5.3*.

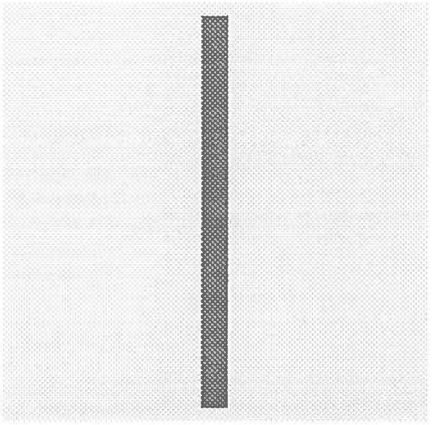

Figure 5.2: Testimage Valley Minima.

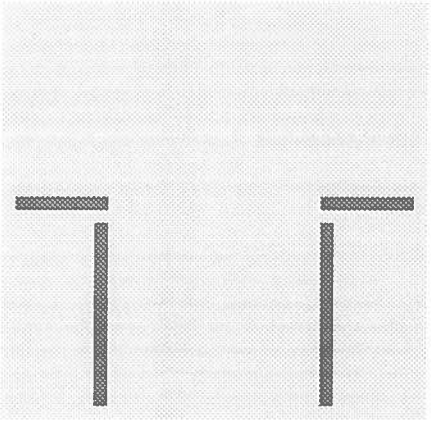

Figure 5.3: Testimage Unconnected Ridge Apexes.

For the NG42 region data (smoothed 40 times) the results of suppression are displayed in *Figure 5.5* — showing the identified valley minima — and in *Figure 5.6*, which depicts the ridge apexes.

Falsely suppressed maxima and minima at junctions have been extended using available standard library HIPS image processing routines `accrete` and `erode`. Although this is not the approach described in §5.2.1 above, it approximates the same operation by replacing pixels which do not conform with their immediate neighbours.

The procedure effectively connects up small pixel gaps in the maxima and minima tracks. The size of the gap is determined by an argument, `n`, supplied to `accrete` and `erode` which replaces differing neighbours upto 2`n` pixels away. A value of `n`=1 was used for connecting up the ridge apex tracks in the testimage, the results of which are shown in *Figure 5.4* below. For the NG42 region tracks of *Figures 5.5* and *5.6*, a value of `n`=2 was used to connect gaps of upto four pixels.

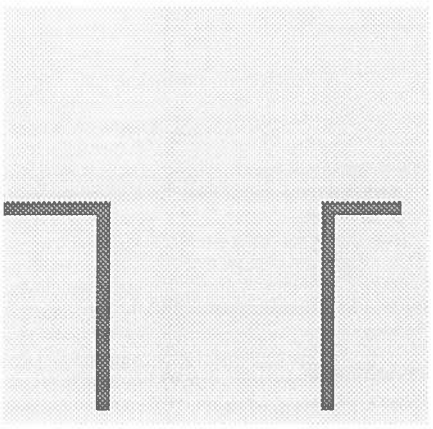

Figure 5.4: Connecting Up Track Gaps in the Testimage.

As can be seen in *Figures 5.7* and *5.8* which show the new connections, the valley minima and ridge maxima are no longer isolated thin tracks. This is because using `accrete` and `erode` effectively extends single pixels into wider regions, and so, the now wider tracks need to be thinned. The thinning process is performed by another library routine `lthin2` which produces the results depicted in *Figures 5.9* and *5.10*, however the effects often produce tracks which are not representative of real maxima and minima (this problem is addressed at a later stage, see §5.4 and Ch.7, §2.4).

5.3 Removing Isolated Tracks

It is both important and necessary to remove isolated feature tracks that arise from noise and features which contain only a small number of points. This is mainly because the visual forces tied to these small tracks will be overwhelmed by the forces associated with the larger, more striking features.

Isolated track removal is a simple procedure that can be easily implemented using stacks. This is best described in terms of a *pseudo*–algorithm :

```
do scan image until next apex/minima found
{
   remove current point from image;
   place point on new tracklist (length = 1);
   push 8 neighbours onto stack;

   while (not stack_empty) do
   {
      pop stack top;
      if (not popped point == apex/minima) then continue;
      else
      {
         remove point from image;
         place point on tracklist;
         increment length;
         push 8 neighbours onto stack;
      }
   }

   if (length < some minimum) then delete track;
}
```

The process creates a data structure which is an array of pointers to lists of track coordinates. These tracks all have lengths greater than or equal to the "some minimum"

which is a value specified by the user as input to the system. *Figures 5.11* and *5.12* show the results of removing all tracks in the **NG42** image consisting of less than 20 pixel points.

The method works well and is useful in that by specifying a minimum length requirement, we can ultimately identify only visual forces of a certain length and above, if necessary.

The code for the above algorithm to remove small tracks (`remove.cxx`) is included in *Appendix C.3* which also contains the non–maximal suppression code `suppress.cxx`.

5.4 Conclusions

Overall, the adapted non–maximal suppression algorithm works well in identifying all possible ridge apexes and valley minima, regardless of the width, length or slope of the associated features. All features are considered equal candidates for providing the evidence which will eventually lead to determining and identifying the lines of visual force in a natural landscape.

However, the solution used in reconnecting up feature tracks, as caused by the failure of the algorithm at junctions, is far from ideal. We can see the usefulness of the method in *Figure 5.4*, but, we can also see the drawbacks — in this case a border pixel (which we have no information about) has been "identified" as a ridge apex, while a previously identified maxima pixel has been removed.

In the testimage, the problem is easy to see, yet, in the real data, the addition of false (and the removal of true) maxima and minima is not so clearly apparent. A number of obviously wrong maxima/minima pixels can be noticed which cross the longer tracks perpendicularly at various points *Figures 5.9, 5.10, 5.11 & 5.12.*

Yet these tracked errors only play a small part. While isolated tracks can be painlessly removed, the original major tracks remain, more or less intact. The hope is that generation of the actual lines of visual force from these, more influential tracks, will help to override the minor side–effect anomalies which are produced.

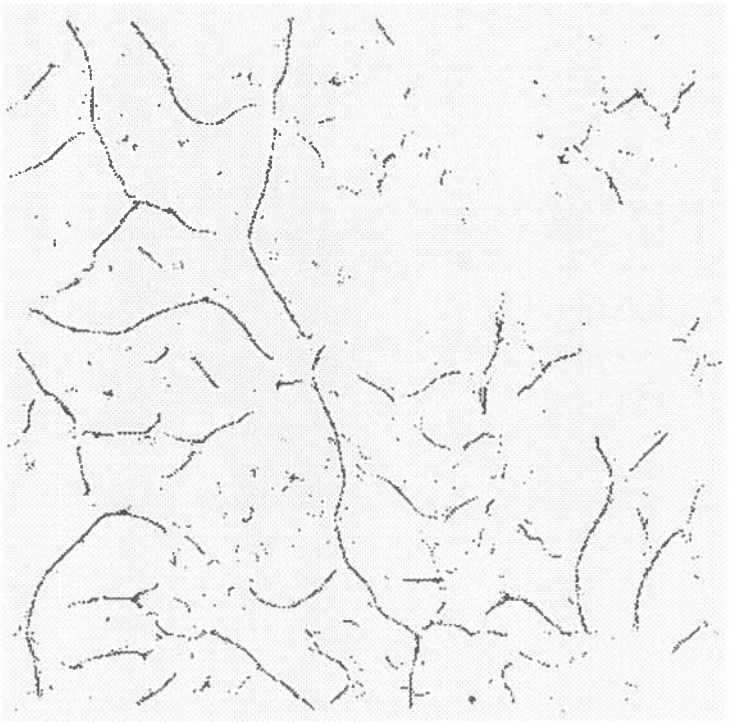

Figure 5.5: **NG42** *(Smoothed ×40) Unconnected Valley Minima.*

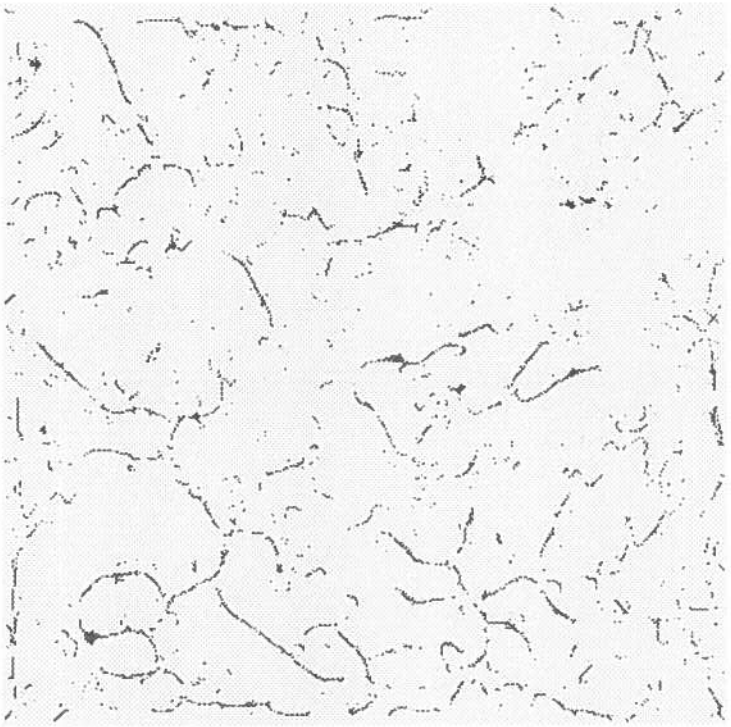

Figure 5.6: **NG42** *(Smoothed ×40) Unconnected Ridge Apexes.*

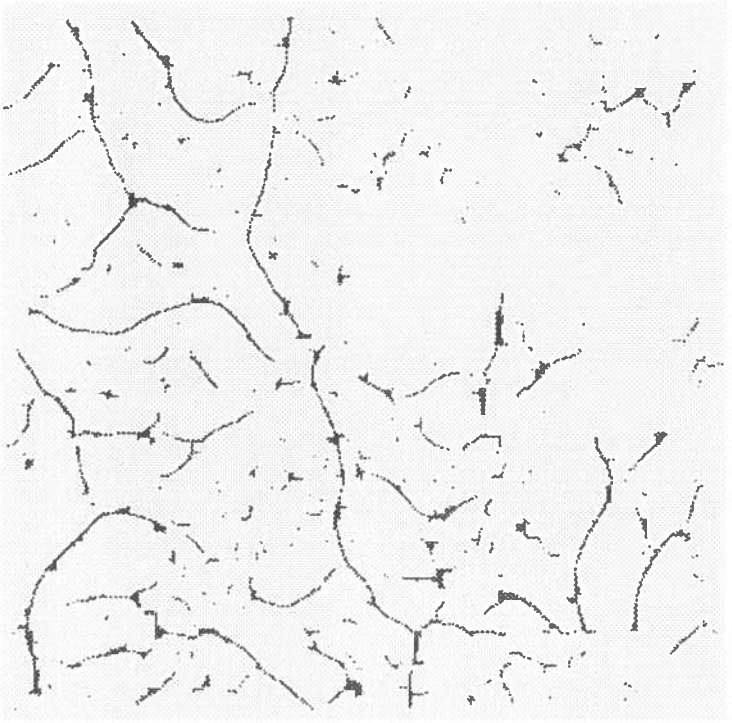

Figure 5.7: *Connecting Gaps in the* **NG42** *Valley Minima Tracks.*

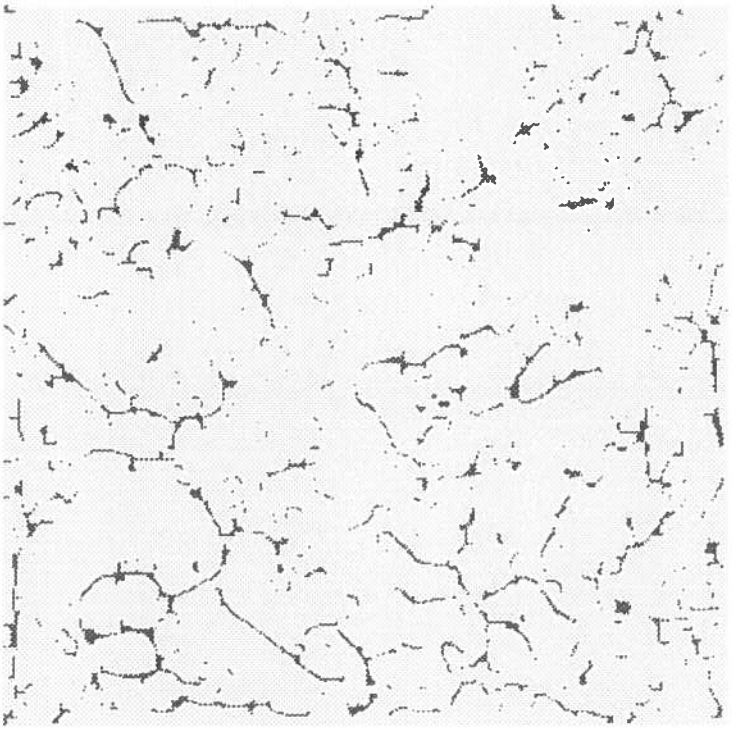

Figure 5.8: *Connecting Gaps in the* **NG42** *Ridge Apex Tracks.*

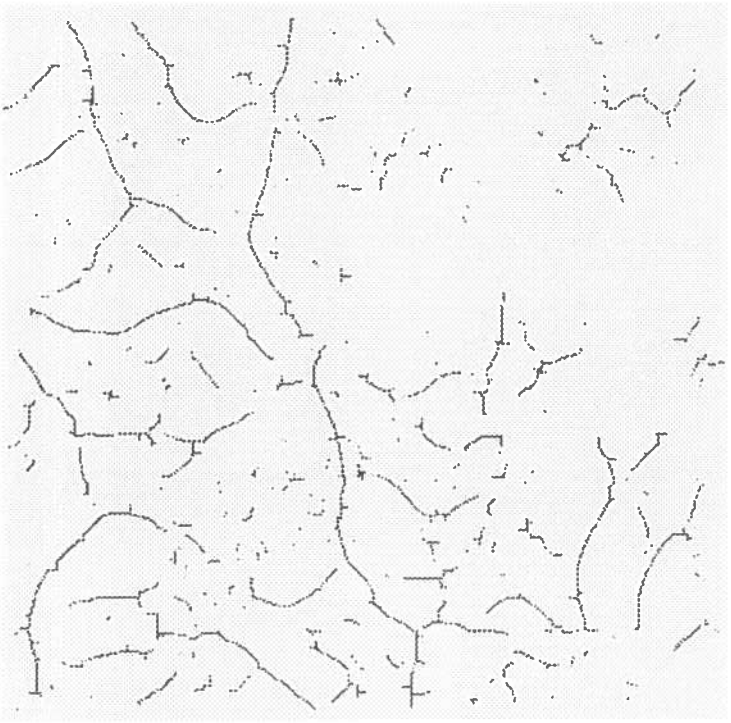

Figure 5.9: *The Effect of Track Thinning (***NG42** *Valley Minima).*

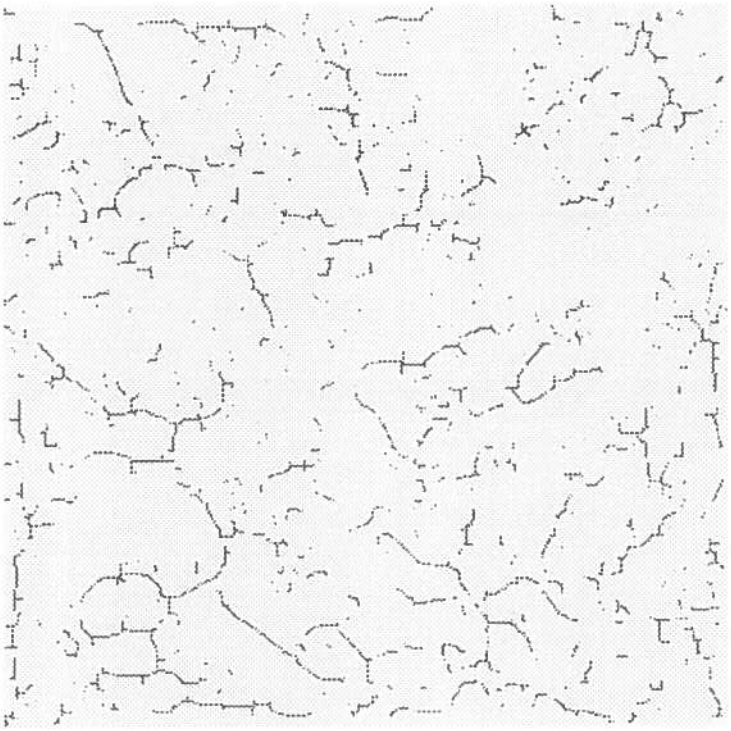

Figure 5.10: *The Effect of Track Thinning (***NG42** *Ridge Apexes).*

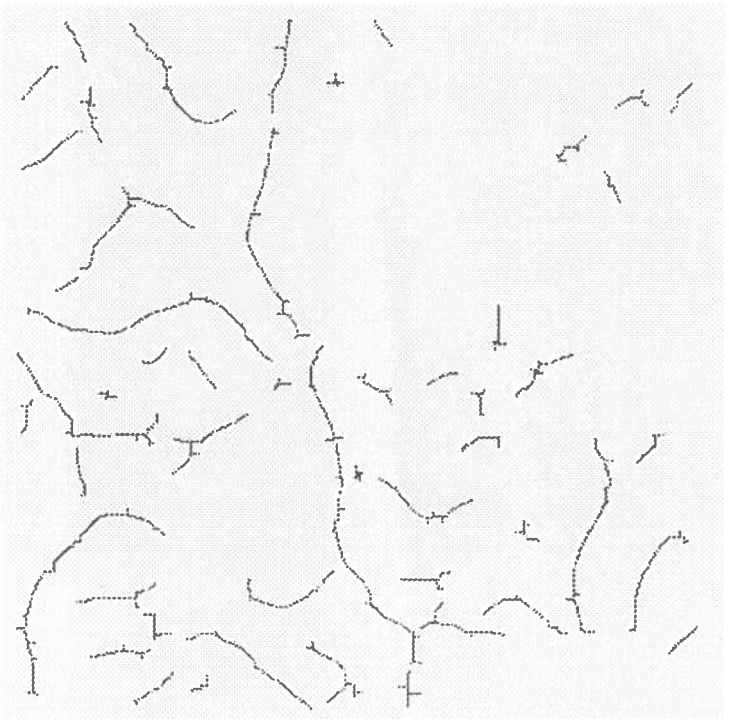

Figure 5.11: **NG42** *Valley Minima of < 20 Pixels Removed.*

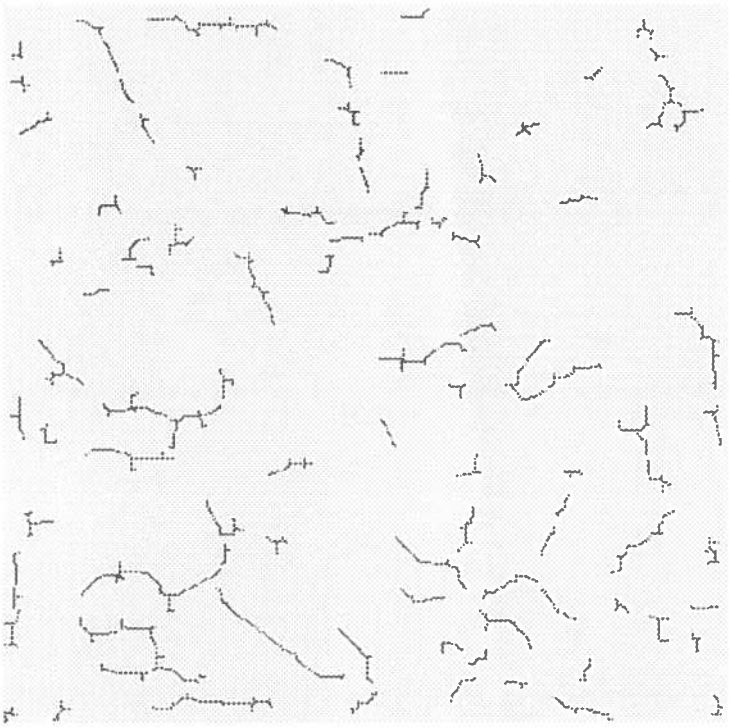

Figure 5.12: **NG42** *Ridge Apexes of < 20 Pixels Removed.*

Chapter 6

Calculating Visual Force

In this chapter I will attempt to describe an approach towards how the aspects of visual force can be calculated. This method will use the information currently available to the system, both in terms of the data originally presented and the results arrived at up to this stage.

6.1 Feature Strength as an Indicator

Not only do lines of visual force follow the features in a natural landscape — up concave slopes and down convex, but the strength of force is closely related to the scale and irregularity of the features to which they are tied.

The visual saliency of a feature can be described as a function of both the feature's width and relative height (or depth). Determining the actual visual forces associated with each feature is thus performed as a related function of the feature's visual saliency.

6.1.1 Ridge Height / Valley Depth

Calculating the height of a ridge (or depth of a valley) provides us with the first starting point, but the question is how can we measure this factor? Clearly, the maxima height point alone is not a suitable indicator since it does not give the relative height.

In order to do this, we need to identify the extremities of the ridge, ie. the bounding local minima. These points (x_1, y_1) and (x_2, y_2) are shown in *Figures 6.1 & 6.2* overleaf, and yield the height values z_1 and z_2 respectively.

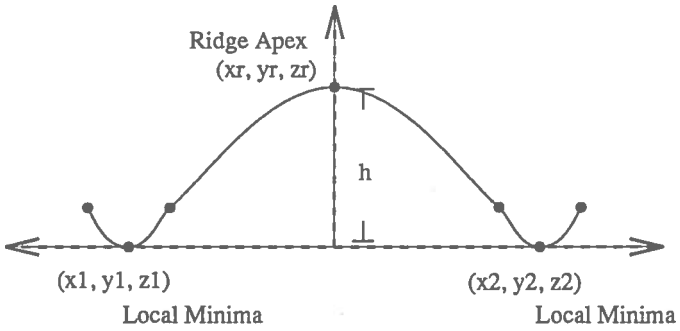

Figure 6.1: *Ridge Apex Cross Section.*

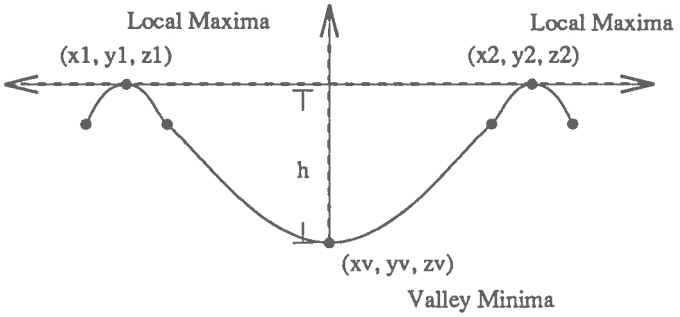

Figure 6.2: *Valley Minima Cross Section.*

The local minima that bound the apex are across the ridge — in a direction indicated by the maximum curvature. From the known pixel orientation information calculated in Chapter 4, §3, the maximum curvature direction can be calculated, being perpendicular to the orientation.

The height values at the corresponding minima points can then be averaged and compared with the height value z_r at (x_r, y_r) to indicate the ridge height, h. A similar procedure is used in determining the depth of a valley.

6.1.2 Width

The bounding minima of a ridge, or maxima of a valley, can also be used in determining that feature's width. This time, since we know that data points are 100 metres apart, the x_1 and x_2 values can be combined to give a measure of the width across the direction of maximum curvature.

6.1.3 Averaging Values and Smoothing

Calculating the visual saliency and using this information to determine the visual force at a single pixel, however, is probably not the best approach. A ridge (or valley) might vary dramatically in either width or height anywhere along its track. But we wish to represent the corresponding line of visual force with a smoothly changing arrow (whose width is an indication of the strength of force) and not a ragged one of differing widths at each pixel point.

Two options to help achieve such a gradual change in the force representing arrow width are averaging and smoothing. Having produced the width and height values at each pixel along the ridge track, an average could be taken to arrive at a mean value. This could be a single average over all values for a small track, or a number of averages for longer tracks which uses interpolated values to maintain a gradual change in width.

Another method is to calculate the visual saliency at each point and then the corresponding strength of visual force. Each value could then be smoothed depending on the values of its two immediate neighbours.

6.2 Implementation

The above ideas can be implemented to calculate the visual saliency of each identified valley minima and ridge apex track. This information can then be combined with the feature orientation and relative position in order to determine the width of the visual force representing arrow

Pseudo–algorithm :

```
// calculate visual force
   for each identified track do
      for each point i do
         Vf_i  = F(orientation_i, position_i, height_i, width_i);
// smooth a bit
   for each point i (1 ≤ i ≤ N - 1) do
      Vf'_i  = F(Vf_{i-1}, Vf_i, Vf_{i+1});
```

Although the algorithm itself is simple, as shown in the *pseudo*–code, the difficulty lies in calculating the information with which to produce the ideal results.

The method in general does not take into account anything about the irregularity of a feature. Visual saliency is calculated from information local to a point, with perhaps some influence from its neighbours through averaging or smoothing. However, there is no notion of how a feature may be changing further down (or previously along) the track — a factor which also can affect the lines of visual force.

Also, without any results, it is impossible to gauge whether calculating visual force as $\texttt{Vf}_i$ is the appropriate method. It may be better to adopt a function such as :

$$\texttt{Vf}_i = \sqrt{(height_i \times width_i)}$$

These methods of calculating visual force, as a function of height and width, is subject to one major drawback. The values give a measure of the feature cross section and so, will treat deep and narrow features like wide shallow ones. This may not be ideal and suggests the need to adopt a more complex approach to calculating visual force.

A better understanding of how visual forces are related to feature scale and irregularity, which may involve heuristic methods, is needed.

Chapter 7

Concluding Comments

7.1 Overall Performance

Unfortunately, without having calculated the visual saliency of the features in the region landscape, it was not possible to compare the automatically identified visual forces with those found manually — as done using the traditional method of land appraisal by the landscape architect Mr. Simon Bell.

However, since the method of estimating visual force is closely tied to the identification and tracking of valley minima and ridge apexes, it seems appropriate to mention how effectively this task of identification is performed.

Figures 7.1 and *7.2* overleaf show overlays of the the tracked valley minima and ridge apexes (*Figures 5.11 & 5.12, p.50*) on the **NG42** region HIPS range image (*Figure 3.6, p.23*). Remembering that lighter values in the range image correspond to higher points in the data (which are more likely to be ridges), it can be seen how well the method of non–maximal suppression manages to identify the minima and maxima.

The major features which provide the strongest evidence for visual forces are clearly defined, albeit with small deviations from the real tracks — a side–effect of the thinning process used in Chapter 5, §2.2 (p.43). However, these tracks should not be a significant cause for concern. Strength of force is related to the tracked feature's scale and irregularity and so, the forces associated with the minor deviations will inherently be smaller and thus, be overwhelmed by the stronger forces which correspond to the true, larger feature maxima and minima.

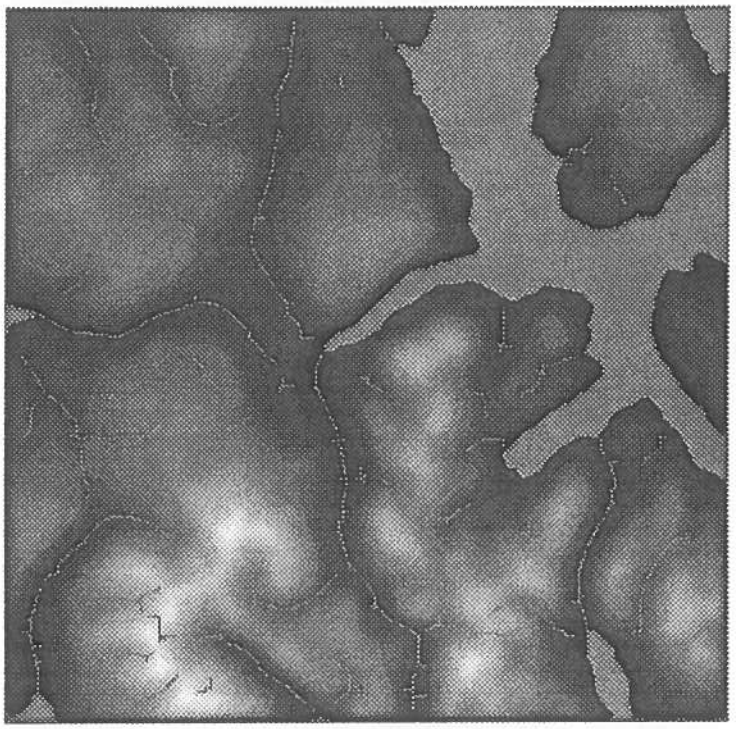

Figure 7.1: **NG42** *Overlaid Valley Minima.*

Figure 7.2: **NG42** *Overlaid Ridge Apexes.*

7.2 Criticisms and the Room for Improvement

There are a number of areas where I see shortfalls in the performance of the system. Each of these shortcomings are described below, pointing out the reasons for the criticism and how, in turn, they can be used as a basis for further work to aid in improving the overall effectiveness of an automatic system.

7.2.1 Real Data Measurements.

The digital terrain data, which represents natural landscapes, is the initial starting point from which we can proceed to identify visual force.

The problem associated with the digital terrain data is related to its measurement and how well it represents the landscape. Points in the data are Ordnance Survey height values taken at 100 metre intervals. Unfortunately, this scale of measuring interval is too large. Since natural landscape can change dramatically over only a few metres, such changes cannot be recorded in such large steps — a smaller interval is needed.

This is especially true of the regions of the Cullin Mountains, **NG40** and **NG42**, chosen because of their rugged, bold features. For all we know, height values between actual data points could be radically different — for example a 90 metre wide gorge between measured points — a possibility not considered when we interpolate between points during the non–maximal suppression stage. However, for less rugged landscapes that slope more gently, intervals of 100 metres are adequate. This suggests the need, perhaps, for a more dynamic scale.

Identified maxima and minima may be "real" in accordance with the data points we have access to, yet they may not be the true maxima and minima of the natural landscape. Worse still, some maxima and minima may be missed entirely. The effect of this is that lines of visual force are identified where real lines (which are not found) do not actually exist.

But what can be done? More precise data is needed to work with, but the question remains how precise ? Such data is not extensively available.

7.2.2 Automatic Parameter Determination.

At various stages in the system, the user can specify certain parameters — the number of smoothing iterations to perform (Ch.3, §2.2, p.19); the threshold value for use in the **H** classification process (Ch.4, §2.3, p.27); the minimum length requirement for isolated track removal (Ch.5, §3, p.45).

But, an automatic system should be able to perform without intervention in that the best choice of parameter in the above, and possibly other, cases is made independently by the system. Automatic determination of such parameters can be done effectively in some cases, but not so easily in others.

For example, the number of smoothing passes needed to remove minute and complex detail from the original digital terrain data, yet leave enough to produce sufficient results, is only made through hindsight on behalf of the user (though even this can be helpful compared to spending a day calculating the visual force by hand). To decide on the number of passes needed automatically would involve the system being able to gauge the effectiveness of the results produced at later stages, and performing "roll–back" to smooth the data fewer or more times, as required.

In the case of deciding on a minimum length requirement to be used with the removal of isolated tracked valley minima and ridge apexes, a value can be more easily determined using a histogram method. In tracking all the maxima and minima, the length of each unique track is determined. From a collection of track lengths, the system could then decide on the best minimum length requirement to be used in order to remove the smaller tracks.

7.2.3 Bi–Quadratic Surface Fit Accuracy.

Two approaches to improving the accuracy of the estimated bi–quadratic surface fit can be considered, as hinted at in Chapter 4, §1.1 (p.25) : [1] using a window whose size changes dynamically over more complex data; and [2] the calculation of mean residuals as a method of determining whether transposing the window to the origin produces a better estimate than calculating the surface fit with the window in–place.

Dynamic Neighbourhood Windows.

The more points we have in a window, the more evidence we have for estimating a bi–quadratic surface fit and so, the accuracy of an estimate increases. Where we have more rugged data which is difficult to characterise in terms of only a few number of points, a larger window may improve the accuracy of the fit.

For this reason, it would seem to make sense to dynamically alter the size of the window being used, depending on the complexity of the data being examined. Mean residuals (see below) could be calculated to give a measure of the fit accuracy, but the problem here is in automatically determining the data complexity in order to adjust the window? What size should the window be adjusted to?

I have examined the results of processing the **NG42** region using the default 3×3 window, as compared to using both a 5×5 and 7×7 window over all the image. By counting the number of minima and maxima tracks found alone, I have concluded that the larger windows do not make any substantial difference to the number, or types of features found. However, this is with data which varies dramatically in places. In the case of more gradual changing landscape, differently sized windows may prove more effective in the long run.

Of course, more complex surface shapes cannot necessarily be represented accurately by a bi–quadratic fit — a more complex parametric model may be needed. A better surface to fit would include zx, zy and z^2 terms, although calculating the corresponding coefficients is somewhat more more of a problem!

Mean Residuals and Window Transposition

It was stated that transposing the neighbourhood window to the origin, so that the pixel coordinates became $[x = 0, y = 0]$, greatly improved the accuracy of the estimated surface fit. This is not always the case.

The accuracy of the bi–quadratic surface fit around the data points in the window can be measured. Singular Value Decomposition returns the parametric coefficients of the estimated surface and these can then be used in conjunction with the window values to get an overall mean residual value.

The mean residual $\mathcal{R}$ is calculated by averaging over the differences between the M z–values in a window, z_i, and the z–values returned by the parametric equation for each (x, y)–coordinate pair. Each calculation uses the SVD estimated coefficients (a $\rightarrow$ f):

$$\mathcal{R} = \frac{1}{M}\left[\sum_{i=1}^{M}(a + bx_i + cy_i + dx_i^2 + ey_i^2 + fx_iy_i) - z_i\right] \tag{7.1}$$

Mean residuals for both the original and transposed window can then be compared and used to choose the more accurately estimated surface fit — the smaller residual indicating greater accuracy.

Discontinuities in the set of data points to be fitted, such as a cliff face, can also lead to major surface fit inaccuracies. By checking that mean residuals remain within a certain threshold limit, such problem cases can be identified, and the system (or user) notified of the need to produce a better fit using a more accurate method.

7.2.4 Maxima / Minima Connections.

Another major critical point is the method used in connecting up gaps in the valley minima and ridge apex tracks. As shown in *Figures 5.4 (p.44), 5.7 & 5.8 (p.48)*, the method of widening the tracks to "subsume" small gaps works fine in terms of performing the connection. However, this method does not take into account that it may be connecting valley minima across ridge apexes, or vice–versa, while also identifying pixels which are not necessarily representative of true. maxima or minima

Had more time been available, it would have been much better to adopt Du Li's approach ([**DuLi 89**]) of examining endpoint orientations and extending tracks accordingly. The method involves adding new pixels to a track provided they follow a similar orientation, change in height value and are of the same classification. All these details are currently available within the system.

7.3 Further Work

So, apart from some fine tuning methods which could be employed, as described in the previous section, what further work remains to be done?

Obviously, the first problem to be tackled is that of calculating the visual saliency of the features, and hence the associated visual forces. This can be done along the lines as suggested in Chapter 6, however I feel that more work is needed to determine the scale and irregularity of features, along with a better understanding of how the strength of depicted visual forces is related to this information.

Having determined the aspects of visual force, the next hurdle would be to produce a 3–dimensional wireframe plot, representative of the landscape under consideration. The visual forces could then be plotted on top of the mesh — green for forces which push the eye up concave slopes, red for those that attract the eye down convex features — thus allowing the user to see how the forces interact with the lie of the land.

Adopting this approach would clearly be better than just viewing the directions of force, as seen in a 2-dimensional environment from directly above, which is, perhaps, suggested by the graphical images produced at each stage of the process so far. Further to this, being able to view the plot from any arbitrary direction would give a clearer indication of how lines of visual forces follow the concave and convex features in natural landscapes.

Chapter 8

Bibliography

- **[F&W 83]** : Martin A. Fischler and Helen C. Wolf — "Machine Perception of Linear Structure" — *in Proceedings of the 8^{th} International Joint Conference on Artificial Intelligence : Karlsruhe, Germany, August 8^{th} - 12^{th}, 1983, pp 1010–1013.*

- **[B&J 85]** : Paul Besl and Ramesh Jain — "Range Image Understanding" — *in IEEE Conference on Computer Vision and Pattern Recognition, 1985.*

- **[I&J 85]** : D. J. Ittner and A. K. Jain — "3–D Surface Discrimination from Local Curvature Measures" — *in IEEE Conference on Computer Vision and Pattern Recognition, 1985.*

- **[Cann 86]** : J. Canny — "A Computational Approach to Edge Detection" — *in IEEE PAMI Vol.8, No.1, 1986, pp 679–698.*

- **[Pres 88]** : W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling — "Numerical Recipes in C" — *Cambridge University Press, 1988, pp 60ff & 517ff.*

- **[DuLi 89]** : Du Li, G. D. Sullivan & K. D. Baker — "Edge Detection at Junctions" — *in Proceedings of the 5^{th} Alvey Vision Conference, 1989, pp 121–125.*

- **[Cai1 90]** : L. D. Cai — "Scale–Based Understanding Using Diffusion Smoothing" — *Unpublished PhD thesis, Dept. of Artificial Intelligence, Univ. of Edinburgh, 1990.*

- **[Cai2 90]** : L. D. Cai — "An Estimate of the Relationship Between Zero Thresholds of Gaussian Curvature and Mean Curvature" — *in Proceedings of the* 4^{th} *IMA Conference on the Mathematics of Surfaces : Bath, UK, September* 14^{th} - 17^{th}, *1990.*

- **[Fole 91]** : J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes — "Computer Graphics : Principles and Practice (Second Edition)" — *Addison–Wesley, 1991, pp 723–725.*

- **[FCom 92]** : Forestry Commission — "Forest Landscape Design Guidelines" — *pp 4–5, 26–7.*

- **[Wren 92]** : David Wren — "Finding Robot Grasping Points from Range Data" — *MSc thesis, Dept. of Artificial Intelligence, Univ. of Edinburgh, 1992.*

Appendix A

Digital Terrain Data Format

The data input into the system is a *massaged* version of the original digital terrain data, consisting of two separate files — representing the two O.S. National Grid Squares (**NG40** & **NG42**). It is structured (and read in) as follows :

- each file consists of 401 blocks of data arranged in a West → East direction, each containing 401 height values.
- each block starts with a row having "51" as characters 0–1 and characters 27–29 encoding which block it is (1–401). The rest of the row is ignored.
- the next 21 rows have 19 4–character height records, followed by a "1", going in a South → North direction.
- the height records, encoded in **ASCII**, represent the elevation in metres ranging from -100m to 1500m.
- each block ends with a 23^{rd} row that has 2 final 4–character height records (ie. the 400^{th} and 401^{st}), followed by a "0".
- a typical block example is shown overleaf along with a graphical representation of how the data is arranged.

```
begin                          block number
  ↓                                ↓
 51  01400820 900201 000000 200 200    1 401    1    100011                  10001  ← row 1
 329 337 334 335 339 339 330 322 315 310 308 302 293 285 276 270 263 257 2521 ← row 2
 246 242 239 237 238 239 242 249 261 274 284 295 303 308 312 307 302 300 2991 ← row 3
 296 291 296 299 305 314 313 286 244 228 201 184 168 155 145 135 127 128 1241 ← row 4
       ⋮                                ⋮                                ⋮       ⋮
 170 169 164 162 162 170 170 174 175 174 171 171 170 169 164 155 149 147 1471 ← row 20
 146 146 146 146 145 144 140 134 125 116 100  98  97  90  84  81  88  93  931 ← row 21
  93  91  90  86  76  68  66  63  61  58  57  56  56  54  53  52  50  48  481 ← row 22
  48  470                                                                   ↑  ← row 23
       ↑                                                                 end char
     end char
```

Figure A.1: *Example Real Data Block No. 200.*

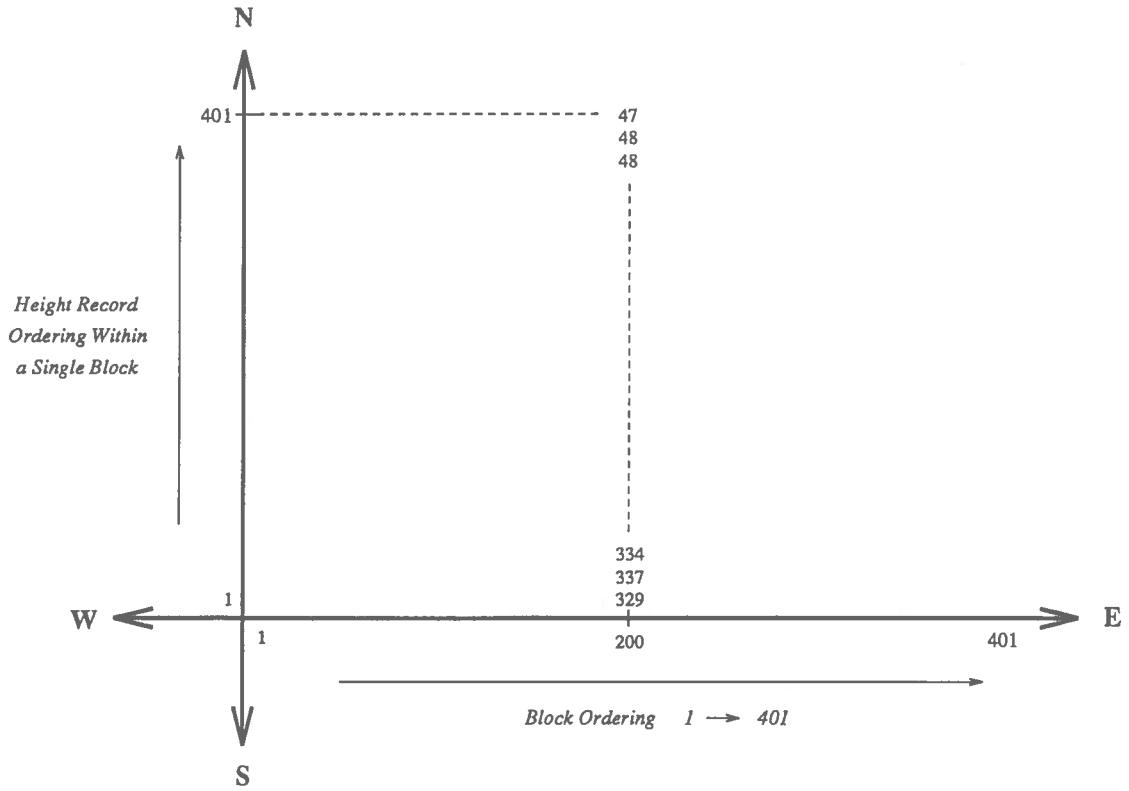

Figure A.2: *Arrangement of Data.*

Appendix B

NG40 Figures

In the main body of the report, all figures are related only to the results of processing O.S. Grid Square reference **NG42**. The second Grid Square reference **NG40** was also used extensively throughout, and all the corresponding figures are collected here along with the corresponding **NG42** figure references.

The figures included are :

- **NG40** Region Landscape — *Figure 2.4.*
- **NG40** Contour Map (Original) — *Figure 3.1.*
- **NG40** Range Image (Original) — *Figure 3.2.*
- **NG40** Contour Map (Smoothed ×20) — *Figure 3.3.*
- **NG40** Contour Map (Smoothed ×80) — *Figure 3.4.*
- **NG40** Contour Map (Smoothed ×40) — *Figure 3.5.*
- **NG40** Range Image (Smoothed ×40) — *Figure 3.6.*
- **NG42** Cosine Shaded Image — *Figure 4.2.*
- **NG40** Classified Image (Original) — *Figure 4.6.*
- **NG40** Classified Image (Smoothed ×40) — *Figure 4.7.*
- **NG40** Classified Image (Smoothed ×80) — *Figure 4.8.*

- Thresholded **NG40** Classified Image (Original) — *Figure 4.9.*
- Thresholded **NG40** Classified Image (Smoothed ×40) — *Figure 4.10.*
- **NG40** (Smoothed ×40) Unconnected Valley Minima — *Figure 5.5.*
- **NG40** (Smoothed ×40) Unconnected Ridge Apexes — *Figure 5.6.*
- Connecting Gaps in the **NG40** Valley Minima Tracks. — *Figure 5.7.*
- Connecting Gaps in the **NG40** Ridge Apex Tracks. — *Figure 5.8.*
- The Effect of Track Thinning (**NG40** Valley Minima) — *Figure 5.9.*
- The Effect of Track Thinning (**NG40** Ridge Apexes) — *Figure 5.10.*
- **NG40** Valley Minima of < 20 Pixels Removed — *Figure 5.11.*
- **NG40** Ridge Apexes of < 20 Pixels Removed — *Figure 5.12.*
- **NG40** Overlaid Valley Minima — *Figure 7.1.*
- **NG40** Overlaid Ridge Apexes — *Figure 7.2.*

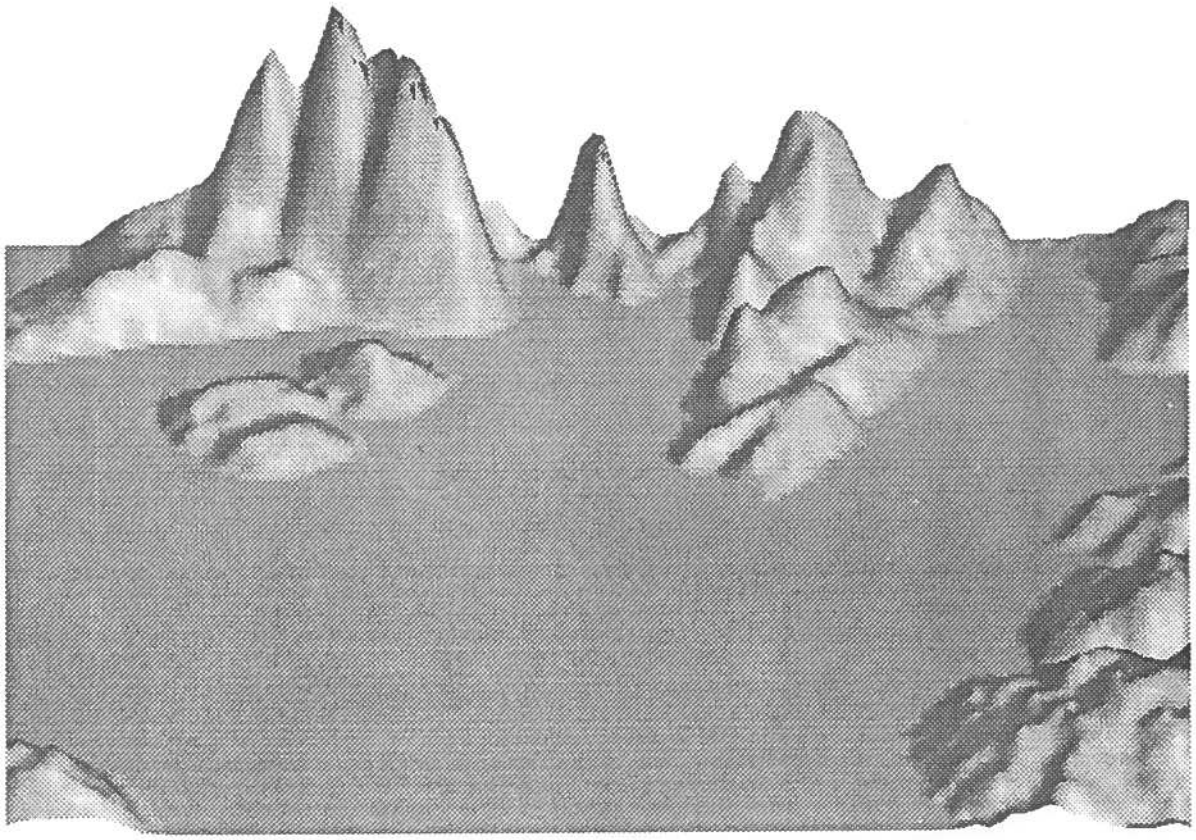

Figure B.1: **NG40** *Region Landscape.*

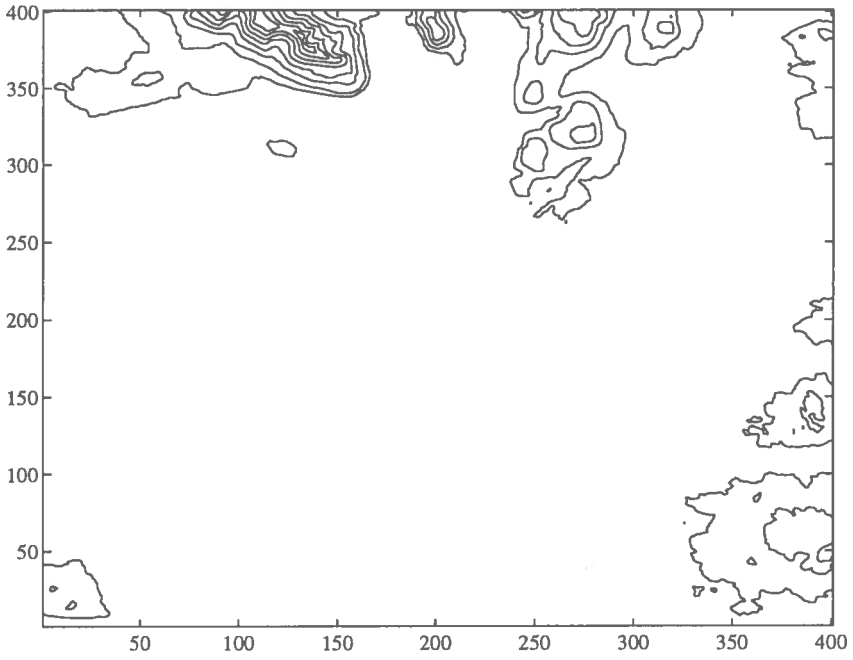

Figure B.2: **NG40** *Contour Map (Original).*

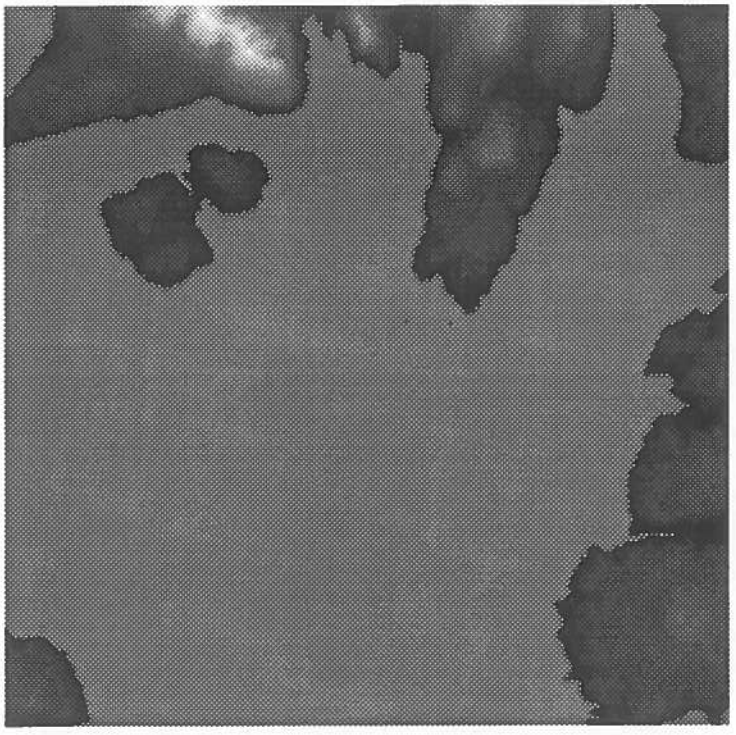

Figure B.3: **NG40** *Range Image (Original).*

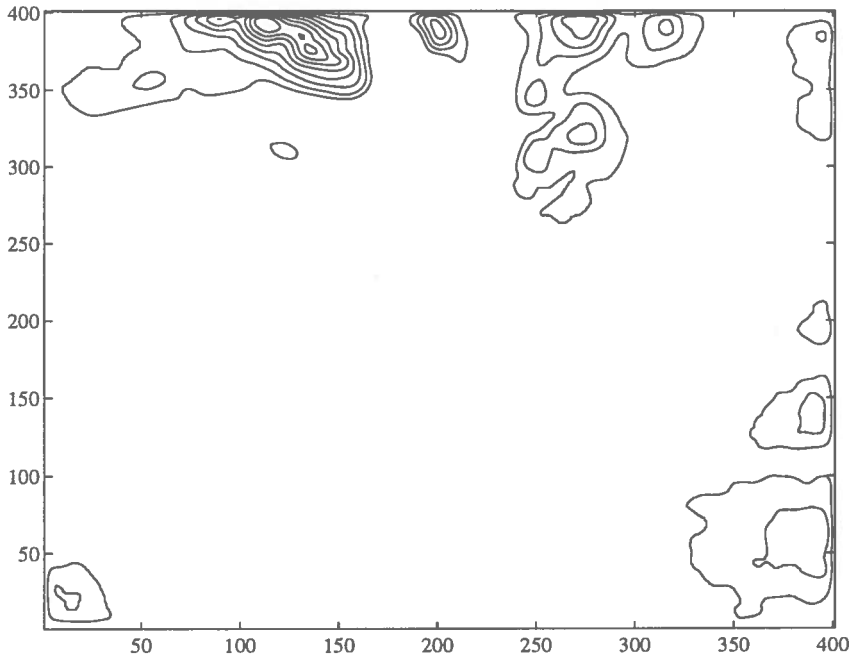

Figure B.4: **NG40** *Contour Map (Smoothed* $\times 20$*).*

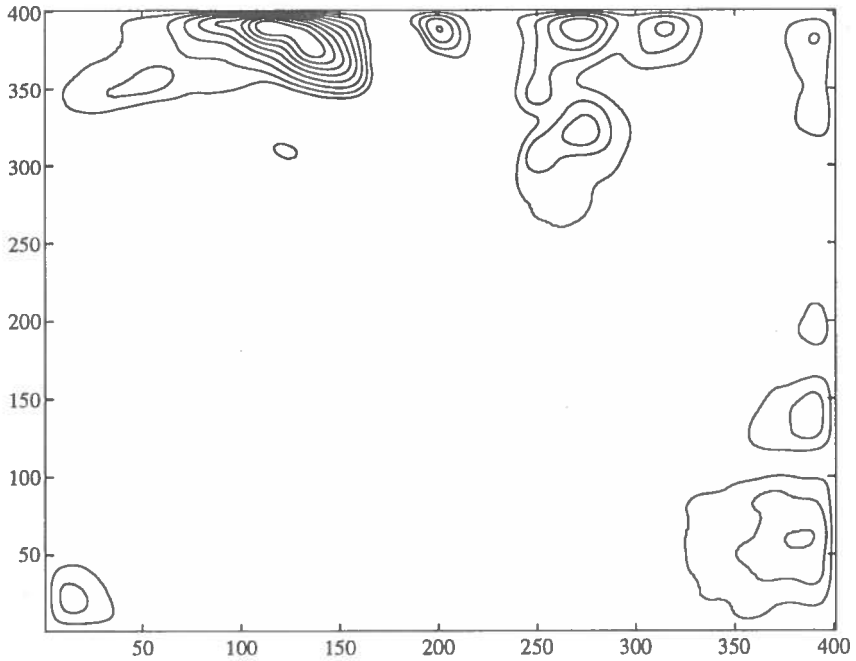

Figure B.5: **NG40** *Contour Map (Smoothed* $\times 80$*).*