DOCTOR OF PHILOSOPHY

Contouring algorithms with terrain mapping applications

Griffiths, Dylan Wyn

Award date:
2010

Awarding institution:
University of Wales, Bangor

CONTOURING ALGORITHMS WITH TERRAIN MAPPING APPLICATIONS

Dylan Wyn Griffiths
School of Mathematics, University of Wales, Bangor.
December 2010

PRIFYSGOL
BANGOR
UNIVERSITY

Thesis submitted to the University of Wales, Bangor
in Candidature for the degree of Doctor of Philosophy

Abstract

When studying geological models below the Earth's surface, or indeed the surface itself, we often wish to concentrate on particular values on the terrain. This is often visualised be means of contour lines. One of the challenges of producing contour lines is the estimation of values between data points – especially if data points are sparse. Another challenge is to produce smooth, 'sensible' contours – contours which do not cross or have sharp corners or loops.

In this thesis we investigate some of the interpolation methods for contouring, highlighting their advantages and limitations, and compare the outputs produced. We propose that our new contouring algorithm described in this thesis will run faster than existing contouring methods, whilst also being able to contour difficult areas such as at discontinuities.

We present the main algorithms used in the new contouring program, Amlin, which utilises the same data structures as TetSim [44] and builds on them to produce the contour outputs shown in this thesis. This leads to a new method for contouring, which uses an interpolating subdivisions scheme based on the Butterfly scheme, which generates C^1-continuous surfaces from arbitrary meshes.

The modified butterfly scheme used in the Amlin program expands beyond the original domain, whilst still respecting the original nodes, including those at the boundary. This can be seen as either an advantage, or a disadvantage, depending on the results required. The expansion beyond the boundary enables us to estimate the nature of the domain, and hence the contour, beyond the boundary. This can be particularly advantageous if the domain contains missing data and holes, as the expansion into these holes makes it more straightforward to estimate the values of the missing data. The problem with this expansion is that if we wished to retain the original domain we would need to perform some trimming or even retriangulation near the boundary.

We present a new method of butterfly subdivision which is constrained to the original boundary. This is especially important when the domain forms part of a

larger data set, where expanding beyond the boundary would cause an overlap of data.

We discuss the extent to which the hypothesis has been proved within this thesis, and for the methods implemented, results and outputs are presented, along with comparisons, suggestions for improvement and further work.

Acknowledgements

I would like to begin by expressing many thanks to my supervisor over the years, Dr Gareth Roberts, for giving me the opportunity to further my academic research and for providing motivation when the future of my work was uncertain. Thanks to my other supervisor, Dr Barrie Wells, for providing the background information for my research, as well as enabling me to use the data sets used in this thesis.

My thanks also to all of the members of the former Mathematics department, who have all helped me during my time as an undergraduate. Also, thank you to my examiners, Dr Nigel John and Prof Horst Holstein, and Prof Holstein's colleague Dr Alan Reid, for their feedback and suggestions for improvement.

Thanks to Dr Ricki Walker for sharing his knowledge of C++, the TetSim program, and for sharing room 423 over the last few years, making it a less lonely place to work.

Many thanks to Liz Du Pré of the Dyslexia Unit, as the study skills and research strategies I learned whilst helping dyslexic students with their mathematics proved to be invaluable in my own research. Thank you also to Anna Story, the note-taking coordinator at the university, for providing me with work when I was short of money and for giving me the opportunity to attend computing lectures and acquire more C++ programming skills. I also gratefully acknowledge the financial support I have received from the ESF for the first three years of my postgraduate studies.

Outside academia, the support of all my family and friends has helped me to stay the course over the past few years. I would especially like to thank my parents, Gareth and Susan, for their love and encouragement, and my sister Siân. Thanks to Matthew, Sam and David for providing me with an occasional break from my studies, and for helping me to relive my undergraduate days.

Finally, I would like to thank my fiancée, Jen, who has helped me to keep focussed on the task at hand and for believing in me when I didn't. She has also been the person with whom I could swap ideas and discuss some of the problems. Thank you all, love Dylan.

Contents

Chapter 1

Introduction and Thesis Outline

1.1 Introduction

1.1.1 Introduction to contouring

When studying geological models below the Earth's surface, or indeed the surface itself, it is reasonable to assume that the area we are studying will not all lie on a fixed plane. For models of the surface of the Earth, we often use the word *terrain* to describe the differences in the elevation of the land. The terrain of a region is particularly important—for environmental, agricultural, geological and many more research areas, and is usually visualised by means of a perspective drawing or with contour lines. Contour lines show lines of constant value (or *level sets*) for some function $f(x, y)$, which in the case of terrain is the elevation (z-value). Clearly we do not know the elevation of every point on the Earth; we only know those values where we have measured. This means that when we are looking at a terrain, we only know the value of a function $z = f(x, y)$ at a finite set of sample points. If we wished to find the value of the elevation at a point which is not in the data set, we would need to approximate it.

The simplest, and most naïve form of estimating the values uses the value nearest to the point we are interested in, although this will certainly produce erratic values for successful contouring, and the surface will contain discontinuities. Other, more involved methods use a weighted average of the surrounding points, which,

depending on the weighting used, often produces a smooth, continuous surface. Once we have a set of points of equal value, the logical step would be to join these points to produce a contour map of the data. Although the contour lines are an expression of a continuous and unbroken surface, they are based on measurements made at the sample points, and so there may be areas between the sample points that vary greatly, such as a deep canyon that was not recorded. As we do not know about the areas in-between the data points, we assume that such data are continuous when we produce the contour map. The contours are also affected by the way the grid or triangulation is formed, as well as which grid or triangulation is used, and so the surface that is generated may not be unique.

Surface uniqueness may not be a significant problem since we are already estimating the positions of the contours, although problems would arise if we required a unique surface in every occasion.

There are various methods in producing the contours, from hand-drawn to computer-generated contour maps. A common method for drawing computer-generated contour maps is to interpolate a grid of uniformly spaced values and then contour this grid. This method is particularly advantageous when it comes to computation, and most interpolation methods produce meaningful contour maps, although they do not always respect the original data points. This is particularly problematic when the data are clustered or has large regions where there is no data. An alternative to gridding involves joining the data points to form a triangulation. This has a distinct advantage as it ensures we have the original data in our data set. As with gridding, we can interpolate between the points to form a contour, with the added advantage that as we are using the data points as corners of our triangles and hence guarantee that we respect the data points. Gridding methods do not always respect the data points, although there are methods to ensure the original data is retained.

When we are considering contouring methods, the simplest involves drawing straight lines between the points that are of equal value, although this is not ideal and often produces contours which appear jagged and have sharp corners. Jagged and sharp cornered contours, as well as not being aesthetically pleasing, are generally

not a good representation of real data. Most geological data flows smoothly from one area to another, where rock types, salt concentrations etc. would alter in a relatively smooth manner. However, there are exceptions to this rule, the main exception being at a fault, although contours would normally end at a fault line and continue from a different point at the other side of the fault. For hand-drawn contours, the contours are usually curves so the problem of jaggedness is not an issue, although hand-drawing contours is considerably more time consuming. For computer-generated contour maps, there are many methods used to smooth out the contours, although this smoothing still may not remove all of the sharp corners and jagged edges. One such method involves computing finer and finer grids until the contours appear smooth. The finer grids would require extra points and extra data – ideally this would come from the source data, where we would start from a finer initial grid. Otherwise, some interpolation or approximation to the data would be required. Another method fits curves to the data points such that the contours are continuous, do not cross and do not contain loops.

In general, smoothing functions are mathematical interpolators or approximators that are designed to improve the appearance of contours without introducing errors such as anomalous curves, loops or crossing contours. If the data we wish to contour are almost or completely sampled (i.e. the sample data points are the population data), then the computer is as likely, if not less likely to produce errors in the contour map, and certainly produces the contours in a shorter space of time than a human can. If we were to have sparse data, often the hand-drawn contour maps are considered to be 'more realistic' as the human drawing would normally have background knowledge of the data and can instinctively reason whether to include an anomalous data point. Computers, however, would require this 'instinct' to be programmed in, and this may be unique to the particular data set. In both cases, the contours are subject to some error, although these errors might be minimal. Hand-drawing contours is a subjective process, and human error can result in misleading contours, especially when the human 'instinct' is incorrect. Similarly, computer-generated contouring is subject to the stability of the algorithm that is being used, as well as the accuracy of the programmer responsible for encoding the algorithm.

Algorithm design can be utilised to improve stability in the algorithm, although this does not remove the risk of human error at the programming stage.

The main contributions of this thesis are:

- Further investigation of natural neighbour bases for interpolation used in contouring algorithms. This is generally ignored as triangulations are normally used as the basis. Natural neighbour bases are an invaluable alternative, especially if the data changes, as natural neighbour bases do not require retriangulation.

- Implementation of the butterfly subdivision scheme for contouring algorithms. The butterfly scheme is normally used for subdivision surfaces, but as contours are level sets on a surface we propose that the butterfly subdivision scheme can be used for contouring algorithms.

  Using the butterfly subdivision contouring algorithm, we have a choice of estimating points beyond the boundary, or to use the novel constrained butterfly subdivision scheme to constrain all data points within the original boundaries of the data set.

- The use of the Worsey-Farin algorithm used over a Powell-Sabin triangle subdivision, referred to in this thesis as *WFPS*, as used by Walker in the TetSim program [44]

- The creation of the Amlin contouring program, which utilises the data structures of Walker's TetSim program.

- Comparisons are made between linear contours, straight line contours over a Powell-Sabin subdivision, WFPS, contours using the natural neighbour method, contours over an unconstrained butterfly subdivision and contours over a constrained butterfly subdivision. Timings for each are also compared. For all contour methods, timings were performed on a 2GHz Athlon CPU with 768MB of RAM.

1.1.2 Thesis motivation

The problem

When considering geological data, the data are usually in the form of two- or three-dimensional scattered points to which attributes are attached. Unlike datasets from areas such as mechanical engineering, geoscientific data often have a highly irregular distribution. For example, bathymetric data are collected at a high sampling rate along each ship's direction of travel, but there can be a very long distance between two lines of data. Geological data are gathered from boreholes and therefore usually has a large amount of data vertically but very little horizontally. In order to model the data sets, interpolation needs to be performed to estimate the value of an attribute at unsampled locations.

Geological data often contains faulted surfaces – surfaces containing unknown discontinuities along lines known *a priori*, and the surface itself can be folded, resulting in multi-valued surfaces. Other difficulties appear when the data are affected by random disturbances. The discontinuity of a faulted surface means that we may have two or more different values at one point, although when a surface is faulted, it is not defined beyond the fault. As stated in Bolondi et al. [6]:

> A map is completely accurate if the proper contour lines touch the correspondent data points, even if they do not lie on the grid nodes. At present, to the knowledge of the authors, the only way of guaranteeing such a behaviour is to introduce an irregular grid containing among its nodes all of the data points.

Bolondi et al imply that irregular grids containing all of the data points are more likely to produce an accurate map of the data.

Alternatively, we could use a method such as kriging [9, 35], which optimises interpolation between the data points using the statistical nature of the surface. Measured points are used to describe properties of the surface, which can then be applied to estimate missing locations. However, kriging has increased time and computational demands, and is also an approximator or estimator, not an interpolator since it does not respect the original data points. Since the data points

are often sparse, it would be logical to respect the data where we have values, and to interpolate between these. Kriging is also unsuitable where we have specific areas of interest whose results are statistically anomalous and have results which differ greatly to neighbouring points. This is likely to occur during our investigations so in order to prevent this from occurring we do not use kriging in this thesis.

Hypothesis. *In this thesis we propose that our new contouring algorithm will run faster than existing contouring methods, whilst also being able to contour difficult areas such as at discontinuities.*

Although we could use gridding and an approximating contour algorithm, as geological data is often sparse it would be preferential to honour the data wherever possible. Because of this we will compare our algorithm with contouring algorithms that interpolate rather than approximate, and produce smooth contours with no jagged edges or corners. We also require the contouring algorithms to agree with Bolondi et al and produce contours which touch the corresponding data points. We will investigate a selection of current contouring methods, highlighting their potential benefits and restrictions. We will look at straight-line contours over the original data, as well as the possibility of using curved contours to produce a smooth contour map. We will also propose a new contouring method, and compare the results with other methods to determine whether, using specific criteria, this new method is faster than existing methods and more versatile in the applications to which it can be used. Three-dimensional data and irregular sampling are not specifically covered in this thesis, although many of the techniques discussed can be applied to these scenarios. A fictitious discontinuity is introduced for one data set to investigate and compare the outputs produced by each contouring algorithm. Again, these algorithms can be applied to data sets where real discontinuities occur, such as at fault lines.

Applications

Analysing data is notoriously difficult in the geosciences. The main problem facing the geologist is the impossibility of making continuous observations in the subsurface

domain, except for occasional access to drill cores or underground works [23]. Below surface data collection is an expensive operation and so this often results in sparse data sets which may include faults and discontinuities. There may also be many different types of surface below the Earth's surface and so we may wish to compare the differences between these surfaces, whether it is the varying salt concentrations, temperatures, pressures, or for any types of data collected.

The most straightforward methods to analyse results is to visualise it. From this point of view, contouring is the more fundamental operation as it immediately presents the data in an accessible format. Contours indicate areas which vary greatly, as well as areas which are relatively uniform. This enables the geologist to concentrate further analysis in certain regions rather than analysing the data as a whole, saving valuable time for future analysis.

The TetSim program can be used to read in data as well as the properties and attributes of the data. After analysis the data can be read by Amlin for fast contouring using a method decided by the user. Of the methods available in Amlin, contours over a butterfly subdivided domain show the most potential, as the butterfly subdivision algorithm can be used either to estimate missing data from the input and even beyond the boundaries of the data, or to be constrained within the original data.

1.2 Thesis outline

We now give a brief description of the contents of each chapter of this thesis. In Chapter 2, we present a short summary of the background theory which provide the foundations for the work in later chapters. We introduce the background to interpolation methods, from basic interpolation which is weighted by a simple average of the surrounding data, to interpolation based on the natural neighbours and Voronoi diagrams. Interpolation is required due to the nature of most data sets – we do not have regularly arranged data points. Even if we do have regularly arranged data points, we still may wish to consider areas between these points. Obtaining data for geological models below the Earth's surface is difficult and expensive, and so

the data collected is often sparse. In order to produce reliable contoured output, interpolation is used. The Sibson interpolation function is defined, both over regular and irregular nodal point arrangements, and will be used in the following chapters. This is followed by introducing contouring methods which can be used to improve on linear contours. A few related methods are discussed, along with reasons why we choose to use Bézier curves as our smooth contouring function.

Chapter 3 introduces the Worsey-Farin algorithm – a contouring algorithm which produces smooth, continuous, contours over a triangulated data set. The triangulation used is subdivided by the six-triangle subdivision as described by Powell and Sabin [33]. We then utilise Bézier ordinates over the Powell-Sabin subdivision in order to compute the Worsey-Farin algorithm. The Worsey-Farin algorithm requires normal vectors at the nodal points, and since we do not have this information in our data, we discuss different methods to estimate these normals. In *TetSim*, Walker [44] chose Nelson Max's method as the preferred method of normal estimation [29], whereas for the natural neighbour method, it is logical to use the *Weighted by Voronoi area* method, since the Voronoi Areas are already calculated. Finally, we produce contour maps of two data sets and compare the two contouring methods with each other. We see that both methods produce similar, yet not identical contour outputs.

We propose a new method of contouring in Chapter 4, this time based on a subdivision scheme as opposed to the algorithm that is built on the subdivision. We briefly mention a trivial subdivision scheme, before proceeding to a scheme known as Butterfly subdivision. The butterfly scheme is named due to the shape of the map of neighbours used during evaluation. The original scheme devised by Dyn et al [18] was general in that it could not subdivide in areas where the area does not look like the butterfly-shaped stencil. In 1996, Zorin et al published an extension to the butterfly scheme known as the modified butterfly scheme [54], which developed rules for cases which were not covered by Dyn et al's original butterfly scheme. We use Zorin et al's extension to the original butterfly scheme throughout the chapter, highlighting the methods used for when we encounter special cases. We use Zorin et al's method as a basis to propose a new method of producing contours and produce

a contour map of one of the data sets in the previous chapter. This is used to show that the Butterfly Subdivision scheme can be used for contouring algorithms and has many benefits over existing methods. The Butterfly Subdivision scheme uses the data structures of Walker's TetSim program and so it is more straightforward to compare outputs between the different methods of contouring.

Finally, in Chapter 5 we compare all of the contouring methods we have seen in previous chapters, using the data sets we have previously seen. We highlight differences between the contour maps, as well as showing that the methods we have investigated can cope with missing data. We also look at real data, taken from the Irish Sea, and compare the contouring methods from Chapters 2 and 3 with the hand-drawn contour maps of the data. The different contouring methods produce similar, although not identical outputs, which satisfy the criteria required and so often it is up to computational time to decide which method is the preferred one for given data.

Chapter 2

Surface approximation / interpolation

2.1 Introduction

Until the arrival of computers, interpolation for contouring was limited to methods that could be easily implemented by hand. These algorithms can now be automated on modern hardware. Apart from the "educated guess" method of drawing contours, there are many manual methods of interpolating data that allow an estimate of elevation. Watson [47] describes these manual methods, the simplest of which involves weighting each of the N data points by $1/N$. This implies that we have a level plane, at the average height, as a representative surface over the region. If the height measurement for a vertex (x_i, y_i) is $F(x_i, y_i)$, then a level plane L over the region has an elevation

$$L = \sum_{i=1}^{N} \frac{1}{N} F(x_i, y_i).$$

This has poor local agreement with most of the data, although globally it offers a good estimate of the total volume. Such a surface cannot be contoured, as a level plane has no variation, and so the variability implied by the data cannot be displayed.

Distance-based weighted averages offer an alternative solution to this problem. These methods are probably the easiest to implement and so are the most abundant in interpolation literature. Various computer adaptations of these methods have been created, including Inverse Distance Weighted Observations (known as IDWO)

and Inverse Distance Weighted Gradients (IDWG) [31]. Inverse distance weighting models use the notion that observations further away should have a lower contribution than those which are near the point of interest. The simplest model involves dividing each of the observations $p(x_i, y_i)$ by its distance d_{ij} from the target point, $p(x_j, y_j)$, such as in the following equation:

$$p(x_j, y_j) = k_j \sum_{i=1}^{n} \frac{1}{d_{ij}} p(x_i, y_i), \qquad k_j = \sum_{i=1}^{n} \frac{1}{d_{ij}}$$

A lot of imaging software packages use this type of model for interpolation, as it is straightforward to implement and simple to understand. Due to the availability of these methods, we will not investigate them further, and so the references can provide the interested reader with a concise knowledge of the particular methods.

An interpolation method that can improve on the above techniques divides the region into polygonal prisms centred around each data point. The height of each prism is determined by the height of its representative point [47]. These polygonal prisms are piecewise constant and so give perfect local pointwise agreement, although global values can vary considerably, as the size of each polygon can differ greatly. An improvement on this would be to use natural neighbour polygons – polygonal regions whose boundaries form perpendicular bisectors of the straight line join to certain neighbouring data points, as we will see later. As these polygons are unique for each data set, they are easily reproducible. The methods described above produce polygons with flat tops, and so we would be looking for a method that produces more satisfactory results.

Another method, known as *convergent gridding* [24], uses a coarse grid which is initially assigned to the data, and then refined many times until the surface converges to a specified smoothness. As the refining process may need to be performed many times, this may not be a favourable method when time is an issue.

A fitted function method, which fits a surface to the data using a least squares method, is more straightforward to implement when it comes to local support, and such methods are also used for gradient estimation, which we will see later. Fitted function methods refer to a class of computer interpolation methods that use a polynomial expression for a surface that fits the data, either locally or globally.

These methods are applied in two stages: first we determine the parameters of the function, and then we use these parameters to interpolate. Fitted functions can be applied to either gridded or scattered data, using a linear combination of elemental surfaces, known as *basis functions*.

Other methods of computer interpolation include triangle and rectangle-based methods, as well as neighbourhood-based methods. These are described in the following sections.

2.2 Triangle-based interpolation

2.2.1 Barycentric coordinates

A more continuous solution can be found using a triangulation of the data, which is equivalent to 'slanting the tops' of the triangular prisms mentioned previously. In other words, we can use a mesh of planar triangles defined by neighbouring three-dimensional data points. Each vertex of the triangle is anchored at a data point, enabling exact local agreement, and then we can apply barycentric coordinates to the data at the vertices of the triangle, giving a weighted average method of interpolation.

Any point P in the plane can be expressed in terms of barycentric coordinates with respect to any triangle ABC of a triangulation $\mathcal{T}$

$$P(x,y) = \sum_{i=1}^{3} w_i(x_i, y_i), \qquad \text{where } \sum_{i=1}^{3} w_i = 1.$$

This forms a 3×3 linear system which has the unique solution

$$\begin{aligned} w_1 &= \frac{\text{area}(P,B,C)}{\text{area}(A,B,C)} \\ w_2 &= \frac{\text{area}(A,P,C)}{\text{area}(A,B,C)} \\ w_3 &= \frac{\text{area}(A,B,P)}{\text{area}(A,B,C)} \end{aligned}$$

Therefore the height of surfaces at a point $F(x,y)$ within a triangle is given by

$$F(x,y) = \sum_{i=1}^{3} w_i(x,y) f(x_i, y_i), \tag{2.2.1}$$

where the weight $w_i(x, y)$ is the i^{th} barycentric coordinate of the interpolated point (x, y) with respect to the triangle, and $f(x_i, y_i)$ the value of f observed at the i^{th} vertex (x_i, y_i). These triangles enable the construction of an isoline map as the interpolated surface is piecewise linear. A rectangular grid could be used in a similar way to the triangulation, giving a fast and easy method of interpolation. However, the major drawback of both the triangular and rectangular methods is that in general the slope of the interpolated surface is discontinuous along each triangle edge [30], so the contouring would contain jagged lines unless the grid was dense.

The use of barycentric coordinates suggests that for general approaches to the interpolation, the weight applied to a distant data point should be less than that for one that is closer. For a set of N points, we could use an inverse data weighting to find the height of a surface F at a point (x, y) by using the formula

$$F(x, y) = \sum_{i=1}^{N} w_i(x, y) f(x_i, y_i) = \sum_{i=1}^{N} f(x_i, y_i) \frac{1/d_i(x, y)}{\sum_{j=1}^{N} 1/d_j(x, y)}$$

where $d_i(x, y)$ is the distance from (x, y) to the i^{th} data point (x_i, y_i). This is the generalisation of Equation (2.2.1). The surface generated by this method would touch each data point, forming cone-like peaks and troughs around them. The slope of the surface is discontinuous at the data points, but continuous everywhere else. This is a fairly accurate method of interpolating the data – surface accuracy is lost at the data points although we know the values at the points and so can manually include them. The price to pay for this accuracy is the large number of calculations required over a simple domain.

This is one method used in *Finite Element interpolation*, as we would triangulate the convex hull of the domain using some triangulation, and then interpolate using an interpolant on each triangle. A commonly used interpolant is Clough-Tocher [11, 52, 51, 34], although we will not be using this in this thesis. A Clough-Tocher subdivision of a triangle forms three subtriangles by inserting a vertex anywhere inside the triangle, then three polynomial patches are determined by the three data points and their estimated gradients. This construction gives us a piecewise quadratic approximation to the function, where the function has been

approximated by a series of quadratic function pieces. Once the triangulation has been completed, the interpolation is very efficient. We can also extend this to higher dimensions as the triangulations still hold, as well as the computed interpolants. The triangulation in adaptive or time-evolution Finite Element interpolation can cause problems, however, as moving the data points may introduce errors in the interpolant if we do not retriangulate the domain. The repeated triangulations are a major drawback of Finite Element interpolation, as the triangulation may require a significant proportion of the computing time, especially for fine meshes.

2.2.2 Delaunay triangulation

A commonly used triangulation for Finite Element interpolation is known as a *Delaunay triangulation.*

A triangulation $\mathcal{T}$ is a Delaunay triangulation of the data set if and only if the circumcircle of any triangle of $\mathcal{T}$ does not contain any point of the data in its interior. Delaunay triangulations maximise the minimum angle of all the angles of the triangles in the triangulation and so they tend to avoid thin triangles wherever possible [39].

In addition to the interpolation methods mentioned here, we can also interpolate using the dual of the Delaunay triangulation – Voronoi tessellation, which we will investigate in the next section.

2.3 Natural Neighbour interpolation

Since the Finite Element method was first implemented, it has become one of the most commonly used numerical methods in engineering and mechanics. This is mainly due to its versatility and ease with which it can be used to solve otherwise impossible problems. One main drawback in the use of the Finite Element method is the time taken for the meshing and remeshing of the domain. Since Finite Element models rely on a mesh, this cannot be avoided, and for large sets of data the meshing algorithm can be very time consuming. Meshes also cause problems for models that contain cracks, or areas where there is a large change in data concentrated over a

small part of the domain. These problems may result in a mesh that gives inaccurate or even unusable solutions [38, 36].

To eliminate this problem, other methods that do not rely on a triangulation have been developed. These include the Element Free Galerkin (EFG) [2], Meshless Local Petrov-Galerkin (MLPG) [27], Moving Least-Squares (MLS) [50], and the Natural Element Method [10, 39]. These are all members of a class of methods known as "Meshless Methods", many of which have been investigated by Belytschko et al [3], De Vuyst et al [13] and Cueto et al [10]. In the following section we will concentrate on an interpolation method using natural neighbours [39].

2.3.1 Natural Neighbours

Consider the set of finite distinct points $N = \{n_1, n_2, \ldots, n_M\}$ in $\mathbb{R}^m$ with positions $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M$. Just as neighbours are the people who live around one particular place, natural neighbour interpolation uses the point (or points) that lie around the area of interest. The nearest neighbour is the closest point to the area of interest. In some cases, such as points located on a circle, a point can have more than one nearest neighbour, although generally there is only one nearest neighbour, where there can be any number of natural neighbours. A definition in two dimensions is given below, although it can be altered for n-dimensions by replacing the two dimensional real domain $\mathbb{R}^2$ with the n-dimensional domain $\mathbb{R}^n$ [41].

Definition 2.3.1. *Consider a point* $\mathbf{x} \in \mathbb{R}^2$. *The point* $\mathbf{x}_i$ *is the nearest neighbour of* $\mathbf{x}$ *if*

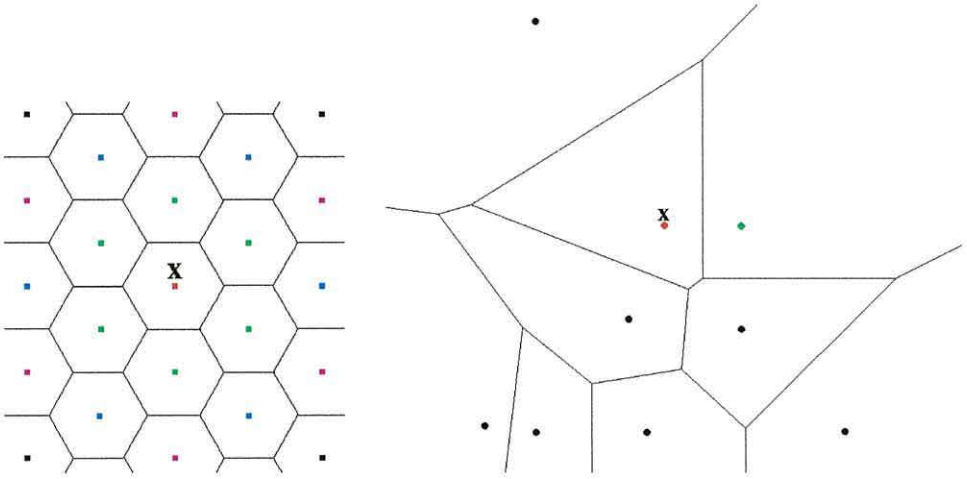
$$d(\mathbf{x}, \mathbf{x}_i) < d(\mathbf{x}, \mathbf{x}_j) \quad \forall\, j \neq i$$

where $d(\mathbf{x}_i, \mathbf{x}_j)$ *is the Euclidean distance between* $\mathbf{x}_i$ *and* $\mathbf{x}_j$.

Figure 2.1 demonstrates the nearest neighbours, where the nearest neighbours to point $\mathbf{x}$ are those indicated in green.

2.3.2 Voronoi Diagrams

Consider again the set of distinct points, $P = \{n_1, n_2, \ldots, n_N\}$ in $\mathbb{R}^2$. The Voronoi diagram of these points places each point into a separate region, or *cell*, where each



(a) Regularly arranged points

(b) 9 randomly placed points

Figure 2.1: Voronoi diagrams for two sets of data. Nearest neighbours are given in green.

point in that cell is closer to the point of that cell than any other on the diagram. These regions are called *Voronoi cells* and mathematically, for each cell T_i we have that

$$T_i = \{\mathbf{x} \in \mathbb{R}^2 \ : \ d(\mathbf{x}, \mathbf{x}_i) < d(\mathbf{x}, \mathbf{x}_j) \quad \forall\, j \neq i\}$$

again where $d(\mathbf{x}_i, \mathbf{x}_j)$ is the Euclidean distance between $\mathbf{x}_i$ and $\mathbf{x}_j$. The Voronoi diagrams for two sets of points are given in Figure 2.1.

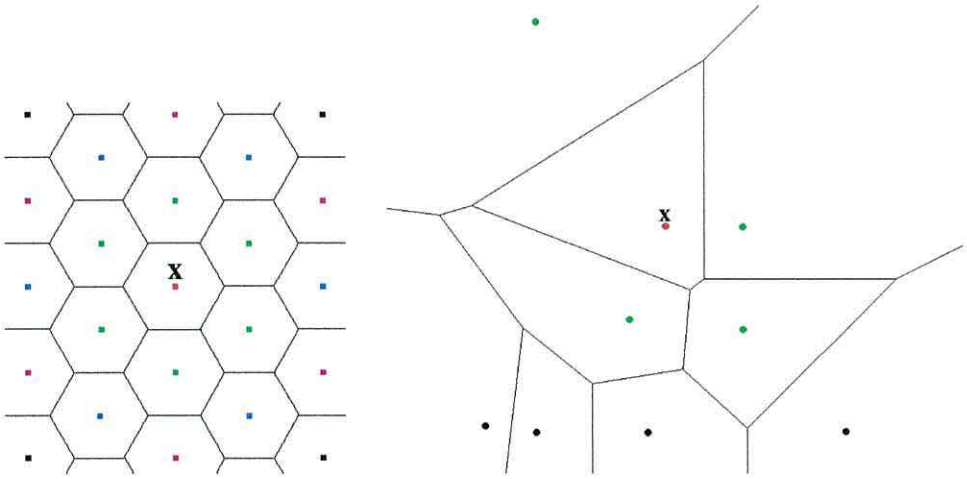
If a point $\mathbf{x}$ was to fall on a line dividing two regions, then it would not have a unique nearest neighbour.

Using the Voronoi diagrams and Definition 2.3.1 from the above, we can complete the definition of natural neighbours.

Definition 2.3.2. *Any two points are said to be natural neighbours if their Voronoi cells have a common boundary.*

Note from this definition that the natural neighbours of a point are not necessarily the same as its nearest neighbours, as two points which are close together may not share a common boundary. In Figure 2.1(a) the points were arranged in a regular pattern, and so the nearest neighbours and natural neighbours are the same points. In Figure 2.1(b), however, the nearest neighbour for the point $\mathbf{x}$ was one point in

the diagram, but the natural neighbours include the other three points surrounding **x**. Note that some points may be closer to **x**, but as the Voronoi cells do not share a boundary they are not considered to be natural neighbours. The natural neighbours for the points given in Figure 2.1 are given in Figure 2.2.



(a) Regularly arranged points

(b) 9 randomly placed points

Figure 2.2: Voronoi diagrams for two sets of data. Natural neighbours are given in green.

If we were to connect each point to its natural neighbours we obtain the Delaunay triangulation. Delaunay triangulations are used in Finite Element meshing, and are the topological duals of Voronoi diagrams [42].

Now that we have a way of presenting and interpreting the data, we need an interpolating function. There are two main interpolating functions [40], known as Sibson [37, 20] and non-Sibson (Laplace), and we will focus on Sibson interpolation – this is refered to as *natural neighbour interpolation* by Watson [47, 48].

2.4 Interpolating functions

2.4.1 Sibson interpolation

Suppose we wish to find the natural neighbour coordinates (or *shape function*) of a general point **x** using the nodal set $\{n_1, \ldots, n_9\}$ in Figure 2.1(b) on page 16. If

$\mathbf{x}$ was tessellated along with the nine points in the diagram, then a new region containing $\mathbf{x}$ would be created as illustrated in Figure 2.3.

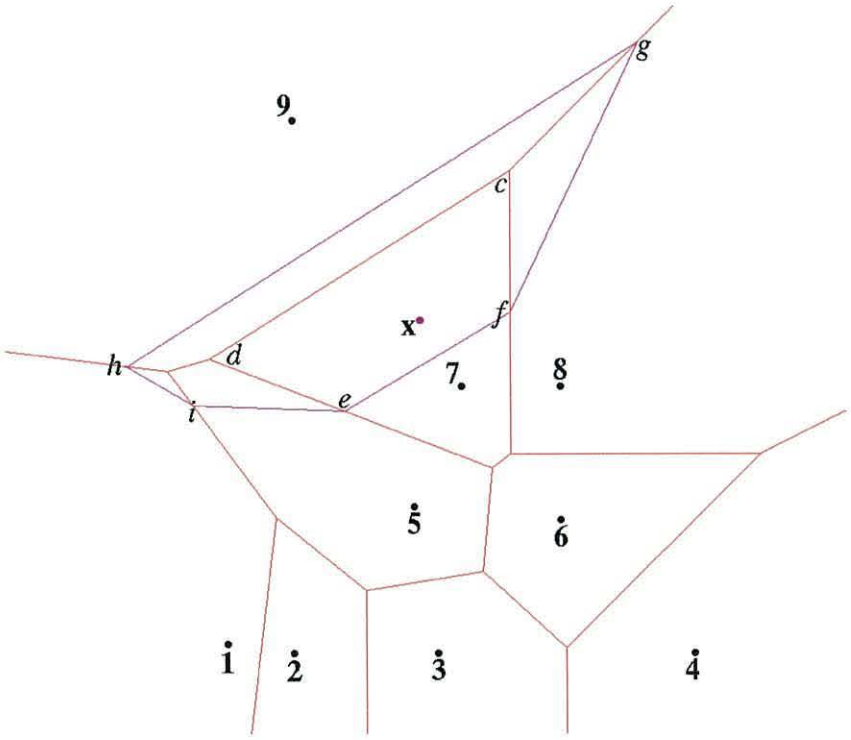


Figure 2.3: Voronoi diagram for the nine points and $\mathbf{x}$.

The natural neighbour coordinates of $\mathbf{x}$ with respect to a natural neighbour I are defined as the ratio of the area of overlap of their Voronoi cells to the total area of the Voronoi cell of $\mathbf{x}$, or

$$\phi_I(\mathbf{x}) = \frac{A_I(\mathbf{x})}{A(\mathbf{x})}, \qquad A(\mathbf{x}) = \sum_{J=1}^{M} A_J(\mathbf{x})$$

where I ranges from 1 to M, and M is the number of natural neighbours of $\mathbf{x}$. If $\mathbf{x}$ coincides with a point $\mathbf{x}_I$, then $A_I(\mathbf{x}) = A(\mathbf{x})$ giving $\phi_I(\mathbf{x}) = 1$, and zero for all other shape functions. A visualisation of the shape function is given in Figure 2.4. The shape function $\phi_7(\mathbf{x})$ is shown, where

$$\phi_7(\mathbf{x}) = \frac{A_7(\mathbf{x})}{A(\mathbf{x})}, \text{ or } \frac{A_{cdef}}{A_{efghi}}.$$

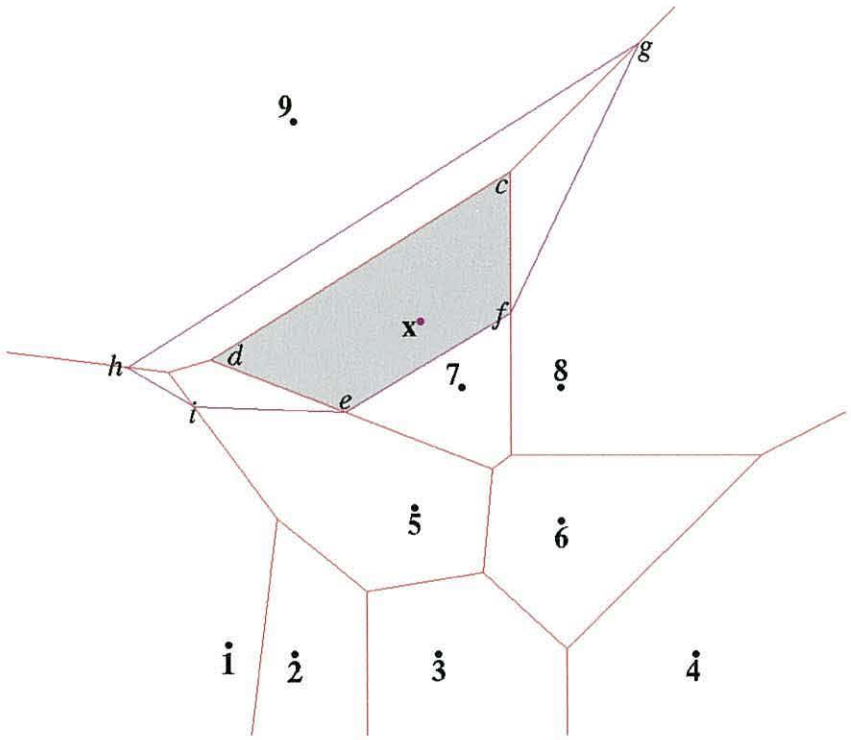


Figure 2.4: Sibson shape function for $\phi_7(\mathbf{x})$.

By the definition of the shape function, we have that [8]

$$0 \leq \phi_I \leq 1 \tag{2.4.1}$$

$$\phi_I(\mathbf{x}_J) = \delta_{IJ} \tag{2.4.2}$$

$$\sum_{I=1}^{M} \phi_I(\mathbf{x}) = 1. \tag{2.4.3}$$

Equation (2.4.2) implies that the natural element interpolant passes directly through the nodal values, and so in a Galerkin implementation the nodal unknowns are the nodal displacements. The Natural Element and Finite Element shape functions both share this same property. This is not the case for approximations in most other meshless methods, as the nodal unknowns are not necessarily the nodal displacements.

Natural neighbour shape functions also satisfy the local coordinate property [37], namely

$$\mathbf{x} = \sum_{I=1}^{M} \phi_I(\mathbf{x})\mathbf{x}_I \tag{2.4.4}$$

which means that the shape functions can reproduce the geometrical coordinates

exactly. The linear consistency conditions are satisfied by Equations (2.4.3) and (2.4.4).

In order to calculate $A_I(\mathbf{x})$ we are able to choose from many methods available. Lasserre's method [26] is simple to implement and can be used to calculate volumes of Voronoi diagrams in n-dimensions for any integer n. The area is calculated around a given point, and so clearly the point needs to be inside the area to be calculated. This is acceptable for calculating areas of first-order Voronoi cells, such as those of **5**, **6**, and **7** in Figure 2.4. It is, however, unsuitable for calculating some areas of overlap, such as the areas of *dcgh* and *cfg*. The algorithm does work for the area *cdef*, but only because point **x** lies inside the area.

Another method is the calculation of the area of each polygon, given by

$$A_I(\mathbf{x}) = \frac{1}{2}\sum_{i=0}^{N-1}(x_i y_{i+1} - x_{i+1}y_i), \tag{2.4.5}$$

where $x_N \equiv x_0$ and $y_N \equiv y_0$, although this area calculation is only valid in two dimensions. We are only interested in two dimensional functions, so this restriction is not a problem. To understand the calculation of the shape function, let us consider points under two arrangements - regular and irregular. We will first consider points arranged in a regular grid.

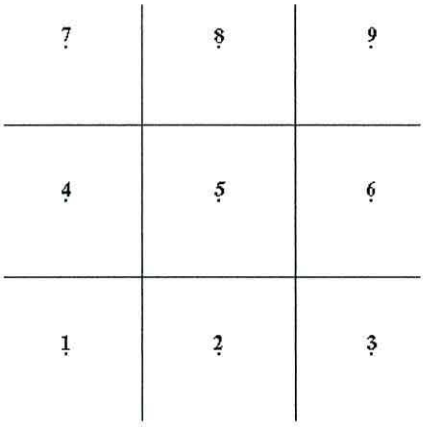
2.4.2 Contouring regular grids

Consider a Voronoi diagram of nine points arranged in a regular grid. The location of the points and the Voronoi cells are shown in (Figure 2.5(a)). A Sibson shape function may be associated with each point, denoted $\phi_{\mathbf{1}}, \ldots, \phi_{\mathbf{9}}$. The value of a shape function at a general point $\mathbf{p}$ is calculated as a ratio of areas. For example

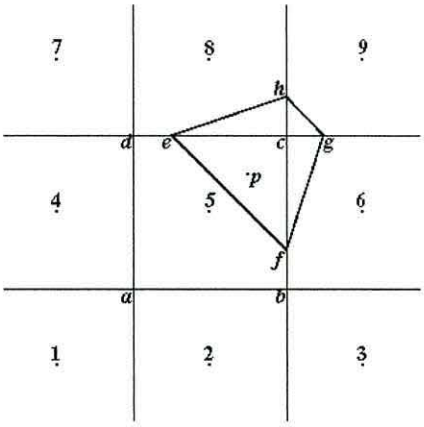
$$\phi_{\mathbf{5}}(\mathbf{p}) = \frac{A_5(\mathbf{p})}{A(\mathbf{p})},$$

where $A(\mathbf{p})$ is the area of the new Voronoi cell surrounding the point $\mathbf{p}$ and $A_{\mathbf{5}}(\mathbf{p})$ is its overlap with the original unperturbed Voronoi cell surrounding point **5**. The area $A_I(\mathbf{p})$ of an N-sided polygon can be calculated by

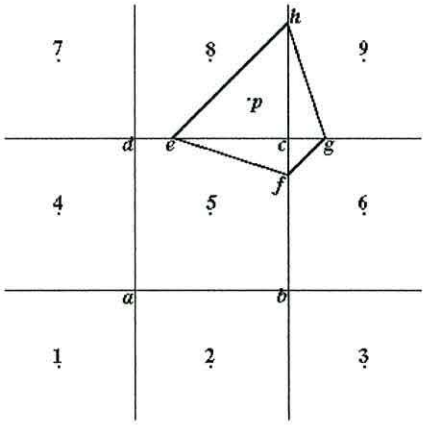
$$A_I(\mathbf{p}) = \frac{1}{2}\sum_{i=0}^{N-1}(x_i y_{i+1} - x_{i+1}y_i),$$



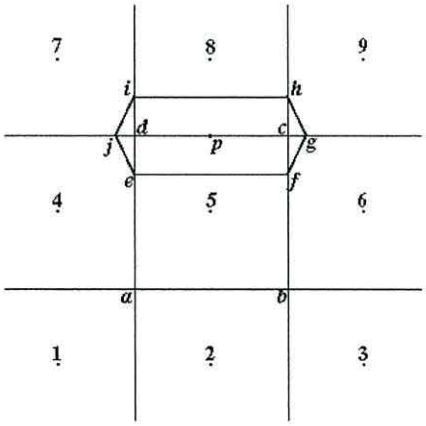
(a) Original Voronoi Diagram



(b) Case 1: Quadrilateral created by a point inside the Voronoi cell **5**



(c) Case 2: Quadrilateral created by a point outside the Voronoi cell **5**



(d) Case 3: Hexagon created by a point on the boundary of **5** and **8**

Figure 2.5: Voronoi Diagrams for use with the Sibson shape function

taking i mod N. So for cases 1 and 2 (Figures 2.5(b) and 2.5(c)) we have

$$\phi_5(\mathbf{p}) = \frac{x_c y_e - x_e y_c + x_e y_f - x_f y_e + x_f y_c - x_c y_f}{x_e y_f - x_f y_e + x_f y_g - x_g y_f + x_g y_h - x_h y_g + x_h y_e - x_e y_h} \tag{2.4.6}$$

and similarly for case 3 (Figure 2.5(d)) we have

$$\phi_5(\mathbf{p}) = \frac{x_c y_d - x_d y_c + x_d y_e - y_e x_d + x_e y_f - x_f y_e + x_f y_c - x_c y_f}{x_e y_f - x_f y_e + x_f y_g - x_g y_f + x_g y_h - x_h y_g + x_h y_i - x_i y_h + x_i y_j - x_j y_i + x_j y_e - x_e y_j}. \tag{2.4.7}$$

Assuming the original nine points of the Voronoi diagram are $\mathbf{1} = (0,0)$, $\mathbf{2} = (1,0)$, $\mathbf{3} = (2,0)$, ..., $\mathbf{8} = (1,2)$, $\mathbf{9} = (2,2)$, then for case 2, a suitable point $\mathbf{p}$ is $(1.25, 1.75)$, and so $\phi_5(\mathbf{p})$ can be calculated as follows:

$$\phi_5(\mathbf{p}) = \frac{2.25 - 1.125 + .9375 - 2.25 + 2.25 - 1.875}{0.9375 - 2.25 + 2.25 - 2.1875 + 3.9375 - 2.25 + 2.25 - 1.6875} = \frac{0.1875}{1} = 0.1875.$$

Similarly, for case 1 we can choose $\mathbf{p} = (1.25, 1.25)$ to obtain $\phi_5(\mathbf{p}) = 0.5625$ and case 3 with $\mathbf{p} = (1, \frac{3}{2})$ gives us $\phi_5(\mathbf{p}) = \frac{4}{9}$.

As many methods use triangulations as a basis for producing contoured output, it is logical that the dual of the triangulation could also be used to produce contoured output. This has largely been ignored in contouring literature and so we will investigate this further.

Contouring using the natural neighbour algorithm

To determine contours (level sets) of the slope function ϕ_5 at height h, it is necessary to find all points $\mathbf{p}$ such that $\phi_5(\mathbf{p}) = h$. For $h = 0.1875$ the point $\mathbf{p} = (1.25, 1.75)$ in case 2 is one possible solution and thus lies on the contour $\phi_5(\mathbf{p}) = 0.1875$. Due to the number of calculations involved, the other values can be found by solving the problem using the Maple mathematical program [28]. Maple, and its counterparts enables users to enter formulae in traditional mathematical notation. There is extensive support for numeric computations, to arbitrary precision, as well as symbolic computation.

The points e, f, g, and h can be found using the knowledge that g is the circumcentre of **6**, **9**, and $\mathbf{p}$, h is the circumcentre of **8**, **9**, and $\mathbf{p}$, etc.

Consider cases 1 and 2. Both cases have four natural neighbours, namely **5**, **6**, **8**, and **9**. This is because in both cases, $\mathbf{p}$ lies in the region given in Figure 2.6.

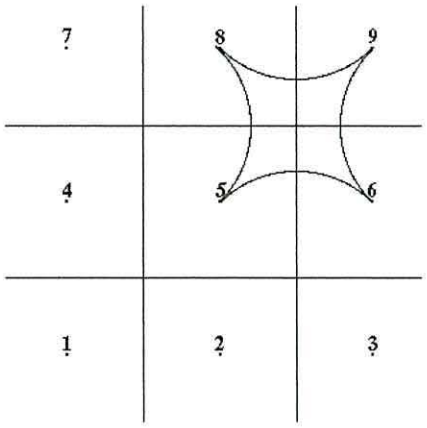


Figure 2.6: Region where point **p** has **5**, **6**, **8**, and **9** as neighbours

The variation of natural neighbours for a regular grid has been found by Sukumar [39], and Figure 2.7 shows the number of neighbours n for each point located in the convex hull of the grid.

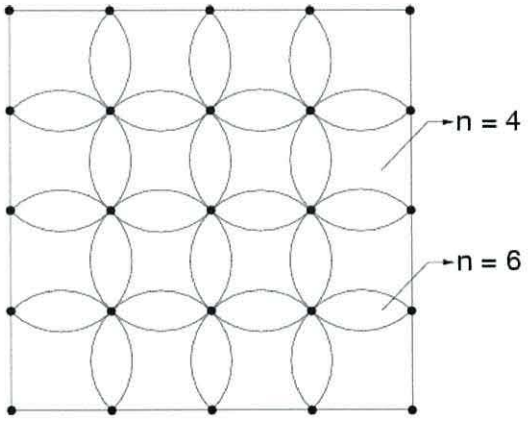


Figure 2.7: Variation of n natural neighbours for a regular grid.

To find the contour lines of the shape function, namely all points where the function ϕ_5 is equal to some value, then we use either Equation (2.4.6) or Equation (2.4.7), depending on the position of the point.

Solving $\phi_5 = 0.1875$ using Maple for points with four neighbours provides four solutions as given in Figure 2.8 and two of the plots for six neighbours are given in Figure 2.9. For the areas where we have six neighbours, the limitations of the Maple program results in two of the four plots being unavailable, since these plots have two y values for each given x value. Due to the regularity and symmetry of the original grid and resulting plots, it is easy to determine the shape of the missing

plots.

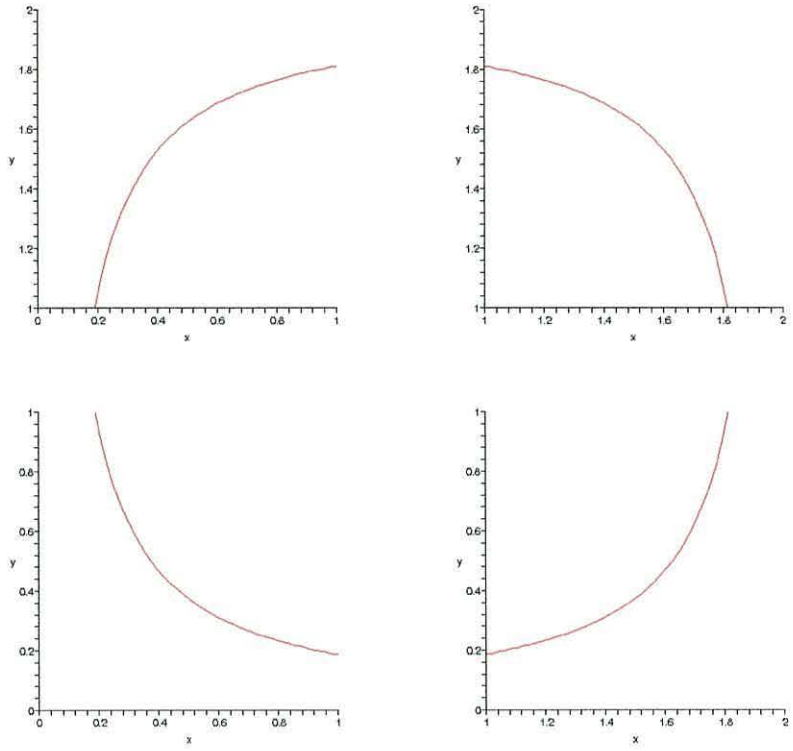


Figure 2.8: Plots for $n = 4$ natural neighbours over the domain $[0, 2] \times [0, 2]$

By merging all of the above plots we can obtain a visualisation of the contour as given in Figure 2.10(a) below. If we included the two missing plots and removed the plots at points outside their respective domains, we would have the contour plot given in Figure 2.10(b).

Although the contour in Figure 2.10(b) appears to be circular, it is not a perfect circle, as shown in Figure 2.11, where the blue contour is drawn on top of a red circle of the same radius. The red circle was a circle as perfect as resolution would allow – if the blue contour was circular, no part of the red circle should be seen at identical resolution. We would not, however, expect the blue contour to be a perfect circle, as the level of $\phi_{\mathbf{5}} = 0.1875$ may be close enough to zero to be influenced by neighbouring regions. As the value of $\phi_{\mathbf{5}}$ increases towards 1 (but not equalling 1) we would expect the contour to approach a perfect circle.

Changing the values of k for $\phi_{\mathbf{5}} = k$ also changes the size of the contour. As we would expect, a value of $k = 1$ produces a single point at $\mathbf{5}$ $(= (1, 1))$, and as

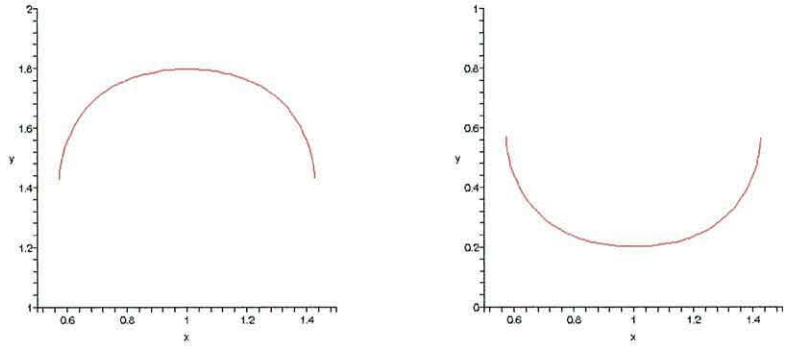
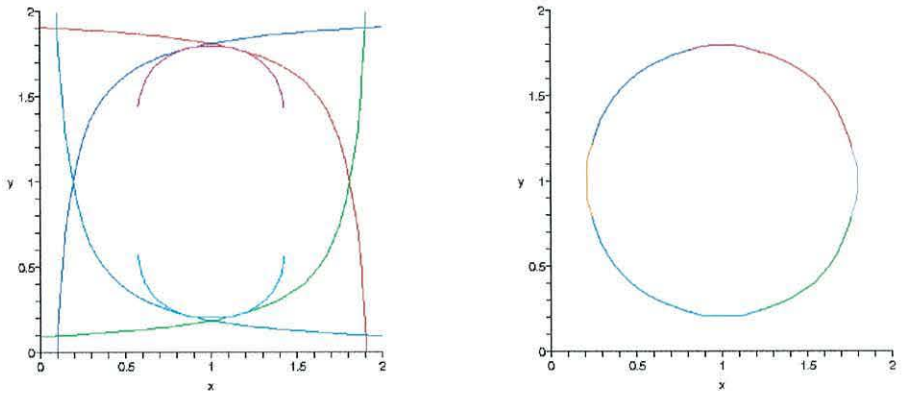


Figure 2.9: Plots for $n = 6$ natural neighbours over the domain $[0.5, 1.5] \times [0, 2]$.



(a) Merged plots produced by Maple (b) Contour produced from the plots

Figure 2.10: Maple plots for $\phi_5 = 0.1875$.

$k \to 0$, the contour expands towards the boundary of the support of ϕ_5. Contours that move beyond the convex hull of $[0, 2] \times [0, 2]$ depend on external points which were originally ignored in the Maple code. Although these points have zero value, they are needed to ensure that the calculations of neighbours are correct.

Ignoring the external points for $\phi_5 = 0.0001$ produces the contour shown in Figure 2.12, which is smooth and correct, since we have reached the convex hull of the data set. As a point approaches the boundary of the convex hull from its interior, Sibson's interpolant becomes piecewise linear. This is due to the areas in Sibson's interpolant becoming infinite, reducing to linear interpolation between the boundary points.

Since the outer points do exist, these cannot be ignored, and so including these points produces a different solution.

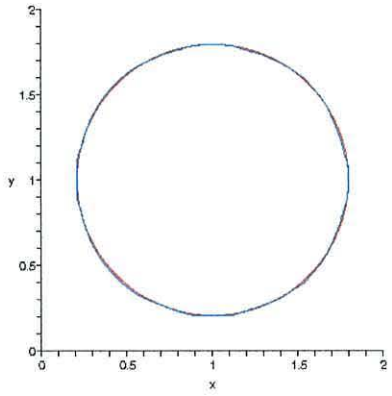


Figure 2.11: Comparison between the contour and a circle of the same radius.

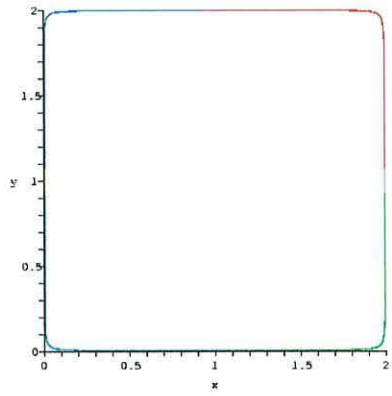


Figure 2.12: Contour map of $\phi_5 = 0.0001$.

Figure 2.13 shows the support of the same function, but this time the support lies totally inside the convex hull of the domain. Using the same coordinates as before now gives a domain of $[-1, 3] \times [-1, 3]$, and we can recalculate the shape function to produce the contour for $\phi_5 = 0.0001$, as seen in Figure 2.14.

For a regular grid, we know from Figure 2.7 that all points inside the grid have either four or six natural neighbours, depending on the position of the points in question. To determine the number of neighbours each point has, we can assume that all points have four neighbours, apart from ones lying in the lens-shaped regions — namely those points that lie inside two circumcircles.

In order to test to see whether a point $\mathbf{p}$ lies between two circumcircles, we need to know the circumcentres of these circles. Assuming that the centres of circles Q and R are at $\mathbf{q}$ and $\mathbf{r}$ respectively, then

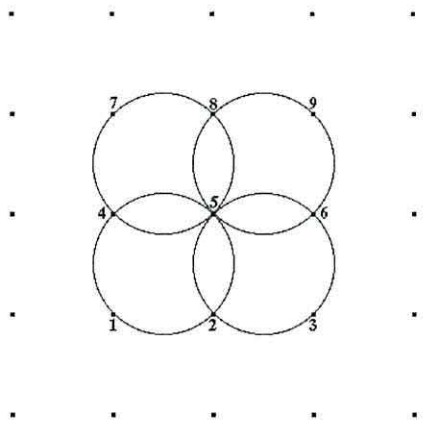


Figure 2.13: Support of $\phi_{\mathbf{5}}$ for a grid of 25 points.

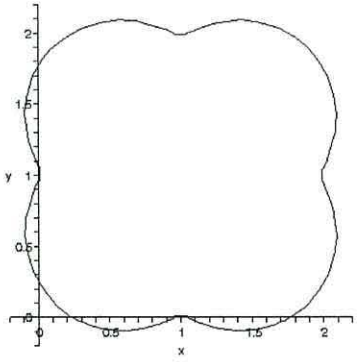


Figure 2.14: Contour map of $\phi_{\mathbf{5}} = 0.0001$.

if $\quad |\overrightarrow{\mathbf{pq}}| < \text{radius of } Q \text{ and } |\overrightarrow{\mathbf{pr}}| < \text{radius of } R$

then $\mathbf{p}$ has 6 neighbours

else $\mathbf{p}$ has 4 neighbours

As we can now determine the number of neighbours a point has on a regular grid, the next logical step is to determine the number of neighbours for points on an irregular grid.

2.4.3 Contouring irregular grids

Consider a Voronoi diagram for a set of nine randomly placed points as given in Figure 2.15, with coordinates given in Table 2.1.

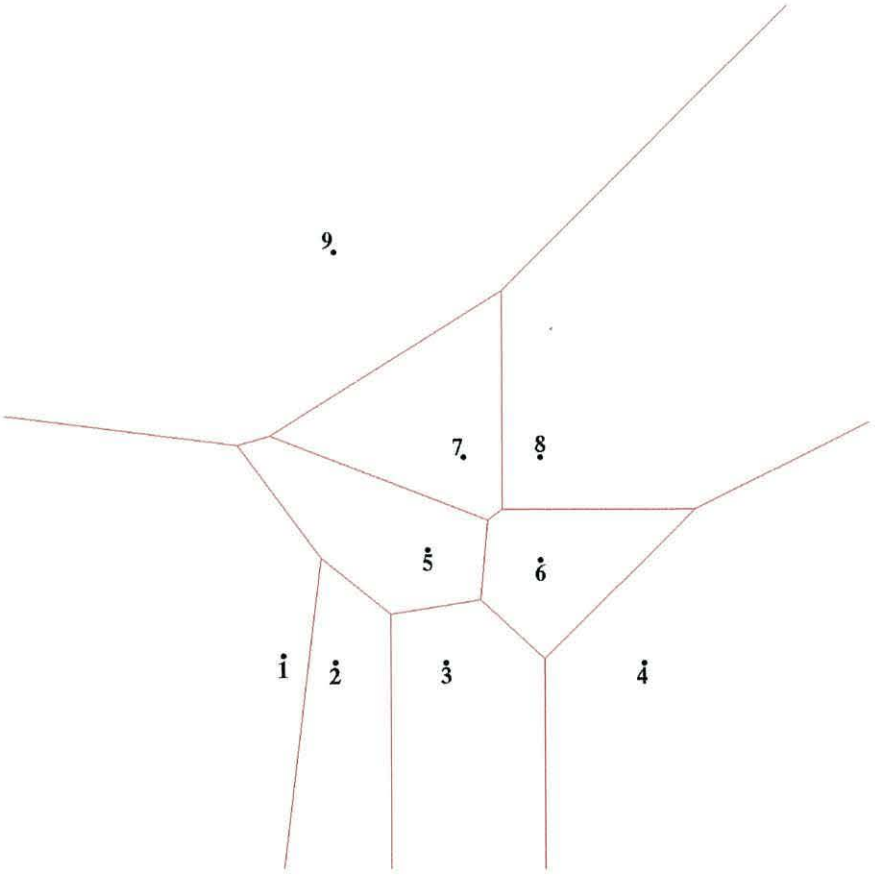


Figure 2.15: Voronoi Diagram for 9 randomly placed points.

Suppose we wish to calculate the shape function $\phi(\mathbf{p})$ for one point in the domain, say point **7**. For points arranged in a regular grid, the shape function was calculated using the knowledge that if the point $\mathbf{p}$ is inside two circumcircles then $\mathbf{p}$ has

Table 2.1: Coordinates of the 9 points.

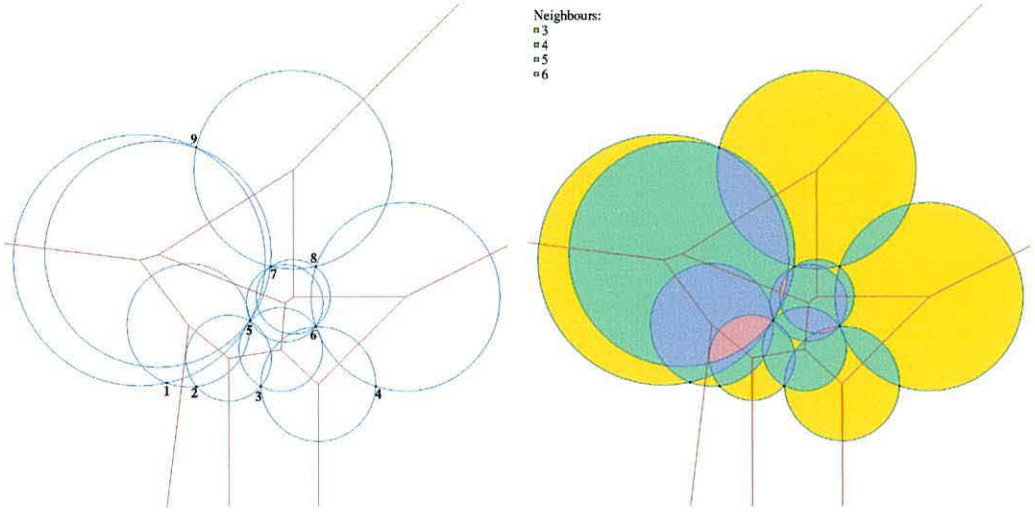
Point	1	2	3	4	5
Coordinate	(-0.5, 0.06)	(0, 0)	(1.08, 0)	(3, 0)	(0.9, 1.1)
Point	6	7	8	9	
Coordinate	(2, 1)	(1.25, 2)	(2, 2)	(0, 4)	

six neighbours, and four neighbours otherwise. A similar method is adopted for randomly placed points, although more consideration is needed for the location of the point compared to its neighbours, as well as the neighbours themselves, since the number of neighbours is no longer restricted to only four or six.

In the regular grid, four points lie on each circumcircle, and so any point that lies exclusively inside that region has four natural neighbours. Where a point lies inside two circumcircles, that particular point has six neighbours since there are six points used to make these circumcircles (four points each, two of them occurring twice). For an irregular grid, again the points make up each circumcircle, and so a point lying exclusively inside that region would have three natural neighbours. A point lying inside two circumcircles would have four natural neighbours, since four distinct points would be used (three points each, two occurring twice) to make the two circles. Generally, for a randomly placed data set, a point lying inside the union of n circumcircles has $(n+2)$ natural neighbours. This claim does not hold for some special cases, such as data points that are all arranged on the same circle. For this case, each point inside the circle would have *all* of the data points as neighbours. Special cases of this type are rare and can be dealt with separately, as and when they occur. The circumcircles for the data set example and the corresponding number of neighbours is given in Figure 2.16.

The support of a point is given in a similar manner as for a regular grid, namely by first looking at all of the points forming circumcircles with the point in question. These are shown in Figure 2.17, and we can see from Figure 2.17 that the points used in the calculation of the shape function ϕ_7 include points **5**, **6**, **8**, and **9**.

In addition to these points, points **1** and **2** also need to be included in the shape function calculations, since there are two circumcircles formed from points **1**, **2**,



(a) Circumcircles (b) Colour-coded Neighbours

Figure 2.16: Voronoi diagrams for the data set.

and **5**, and **1**, **5**, and **9** which overlap the Voronoi region **7**. Now that we have the number of neighbours for a point, in addition to what these neighbours are, we can calculate the contour at $\phi_{\mathbf{7}} = k$, where k is some given value.

As there are many small regions within the Voronoi region **7**, we will assume that these regions have insignificant influence over the whole region, and for the time being, these will be ignored. Our calculations will therefore use points located in the larger areas given in Figure 2.17.

Inserting the point **p** into one of the four areas marked with example points a, b, c, and d on Figure 2.17 produces four possible Voronoi diagrams as given in Figure 2.18. We can now calculate the value of **p** in these regions using the same ideas and notation from the calculations over regular grids.

The four equations to be solved are as follows:

$$\phi_{\mathbf{7}}(\mathbf{a}) = \frac{x_c y_d - x_d y_c + x_d y_e - x_e y_d + x_e y_c - x_c y_e}{x_d y_e - x_e y_d + x_e y_f - x_f y_e + x_f y_d - x_d y_f} = k. \tag{2.4.8}$$

$$\phi_{\mathbf{7}}(\mathbf{b}) = \frac{x_c y_d - x_d y_c + x_d y_e - x_e y_d + x_e y_c - x_c y_e}{x_d y_e - x_e y_d + x_e y_f - x_f y_e + x_f y_g - x_g y_f + x_g y_d - x_d y_g} = k. \tag{2.4.9}$$

$$\phi_{\mathbf{7}}(\mathbf{c}) = \frac{x_c y_d - x_d y_c + x_d y_e - x_e y_d + x_e y_f - x_f y_e + x_f y_c - x_c y_f}{x_e y_f - x_f y_e + x_f y_g - x_g y_f + x_g y_h - x_h y_g + x_h y_e - x_e y_h} = k. \tag{2.4.10}$$

$$\phi_{\mathbf{7}}(\mathbf{d}) = \frac{x_c y_d - x_d y_c + x_d y_e - x_e y_d + x_e y_f - x_f y_e + x_f y_c - x_c y_f}{x_e y_f - x_f y_e + x_f y_g - x_g y_f + x_g y_h - x_h y_g + x_h y_i - x_i y_h + x_i y_e - x_e y_i} = k. \tag{2.4.11}$$

These equations correspond to the points in Figures 2.18(a) to (d) respectively. As the points are arranged irregularly, the solutions to these problems are left in

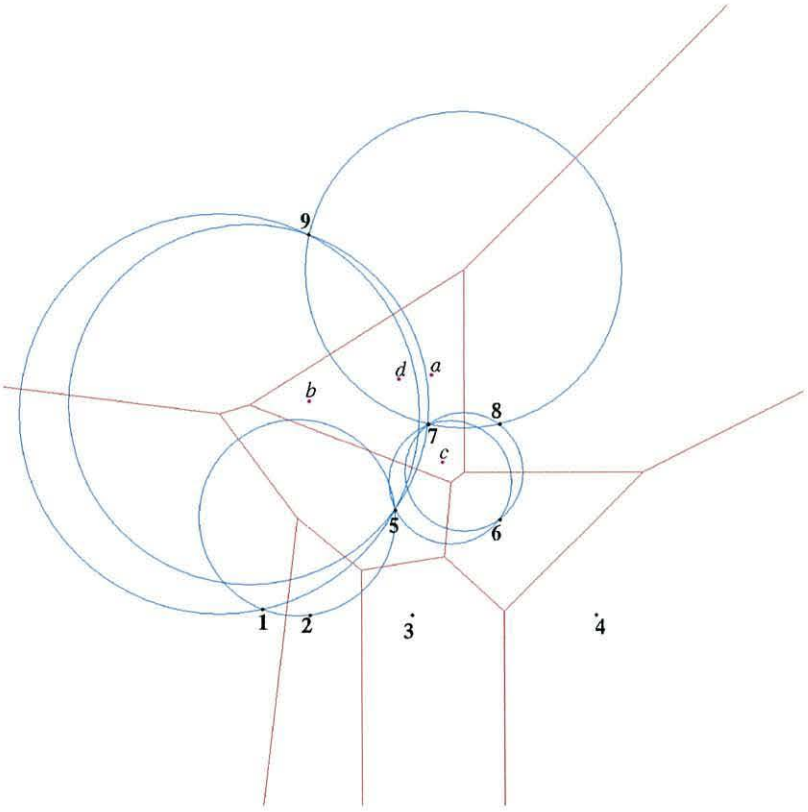
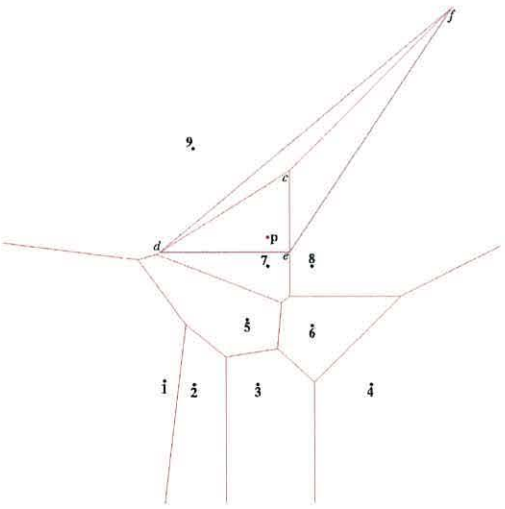


Figure 2.17: Support for point **7**.

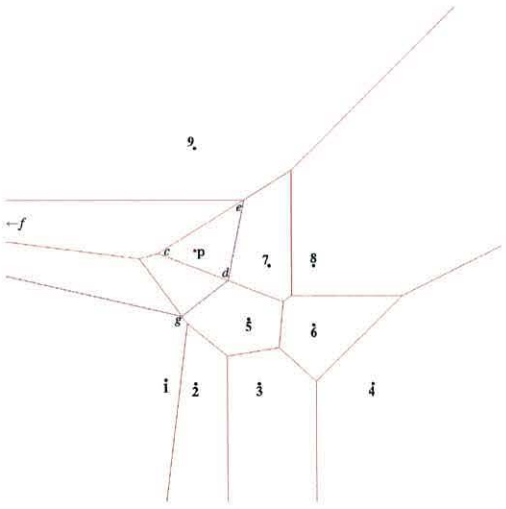
its simplest form by Maple, resulting in many complex equations. The solution to (2.4.8), however, produces a linear expression, and can be solved with ease. This occurred since we were comparing the ratio of areas of one triangle to another in Equation (2.4.8), and for these triangles, two of the three points were the same. If this was to arise elsewhere over the domain, then it is likely that a similar result would be produced.

As the Voronoi region for point **7** is not a square domain, we need to use a domain that includes the whole of the region, thus also including some points from regions **5**, **6**, and **9**. Running Maple for two values of ϕ_7 over this domain produces graphs given in Figure 2.19.

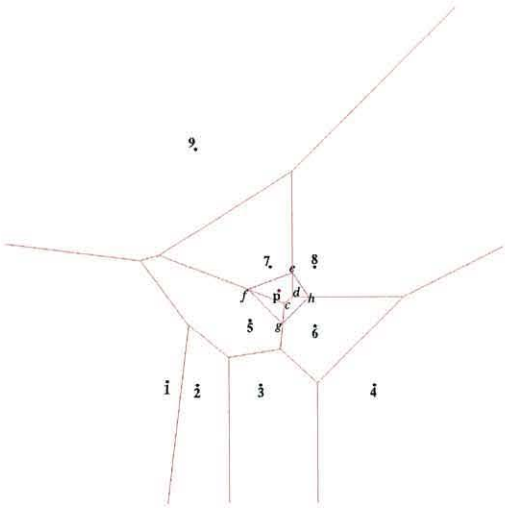
When $\phi_7 = 1$, then as expected, the four curves meet at a single point at **7** $(= (1.25, 2))$ as given in Figure 2.19(a). For $\phi_7 = 0.5625$, however, the combined contours appear to give strange results, although these are not necessarily meaningless for the whole contour. If we were to retain only the curves in their respective domains around point **7** and remove the remainder, then we would be left with the



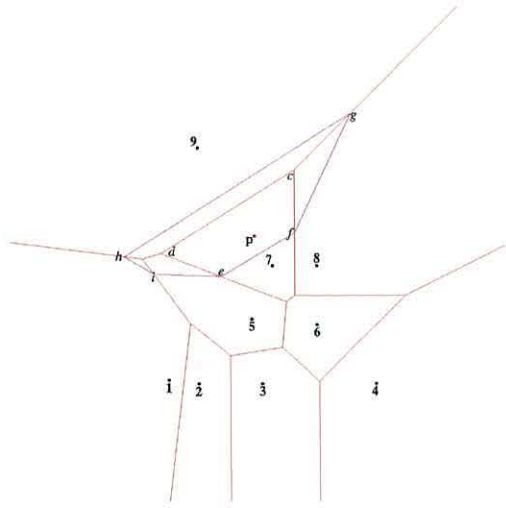
(a) $\mathbf{p} = (1.25, 2.5)$



(b) $\mathbf{p} = (0, 2.25)$

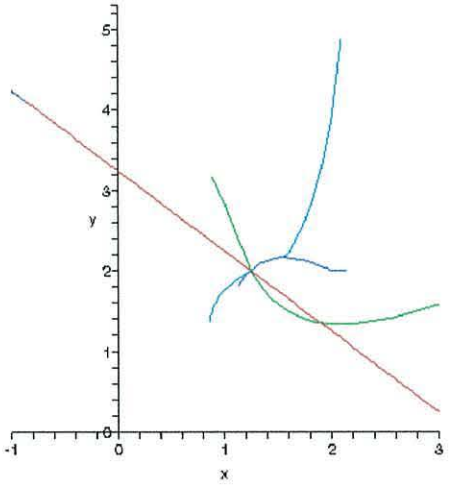


(c) $\mathbf{p} = (1.4, 1.6)$

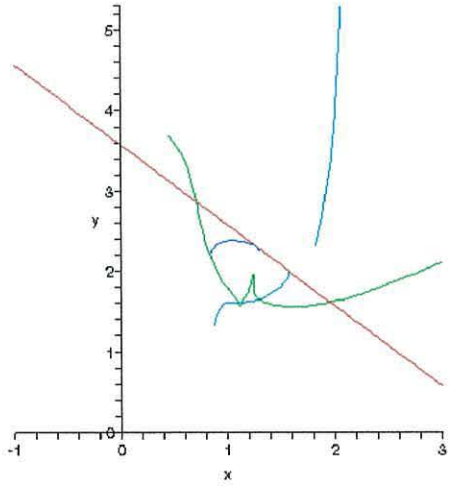


(d) $\mathbf{p} = (0.95, 2.5)$

Figure 2.18: Point $\mathbf{p}$ inserted at various points over the Voronoi cell **7**.



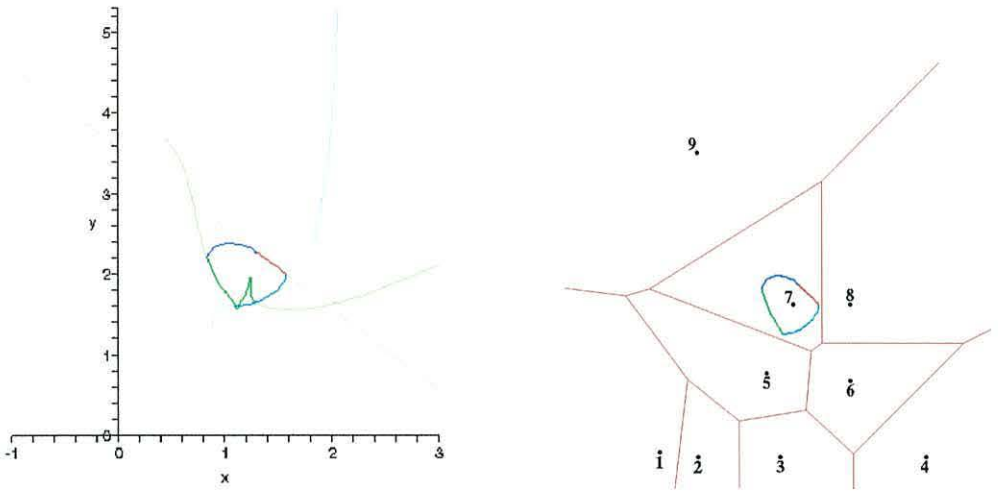
(a) $\phi_7 = 1$



(b) $\phi_7 = 0.5625$

Figure 2.19: Maple plots of $\phi_7 = k$ for two different values of k.

graph given in Figure 2.20(a). The anomalous spike in the green contour was the result of a floating-point error, and removing this enables us to produce a contour for $\phi_7 = 0.5625$ as given in Figure 2.20(b).



(a) Contours retained around point **7**

(b) Contour for point **7**

Figure 2.20: Contours for $\phi_7 = 0.5625$.

Although we have a working contour for the shape function, it is possible that parts of the contour curve may be incorrect. This is because some of the points can lie outside the convex hull of the domain. Figure 2.21 illustrates this problem, where points can lie either above the line $(\mathbf{8}, \mathbf{9})$ or to the left of the line $(\mathbf{1}, \mathbf{9})$, whilst still lying inside the Voronoi region for point **7**. When a point reaches the convex hull, only linear interpolation is necessary, and so the shape function calculations can be simplified.

Beyond the convex hull, shape function calculations are undefined since the areas of interest become infinite, and so as the point $\mathbf{p}$ in Figure 2.18(b) approaches c, contours reduce from a polynomial, to linear, to meaningless. This provides one possible explanation as to why the green contours in Figures 2.19(b) and 2.20 are spiked, and why the shape function curves around a different area to the point used for the calculations. Figure 2.22 shows that as $k \to 0$ for $\phi_7 = k$, this problem increases as other points move outside the convex hull. One important factor that could be addressed to improve the accuracy of the contour concerns the small regions within the Voronoi diagram. These were assumed to have insignificant

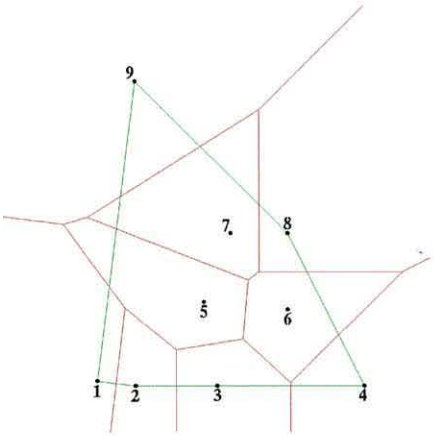


Figure 2.21: Convex hull for the nine points of the Voronoi diagram.

influence over the whole region, but if these regions were not ignored, a more realistic contour diagram would be produced, although more consideration would be needed to determine the exact location of the points used in the Sibson shape function calculation. This ensures that the points being calculated would have the correct number of neighbours at the correct locations.

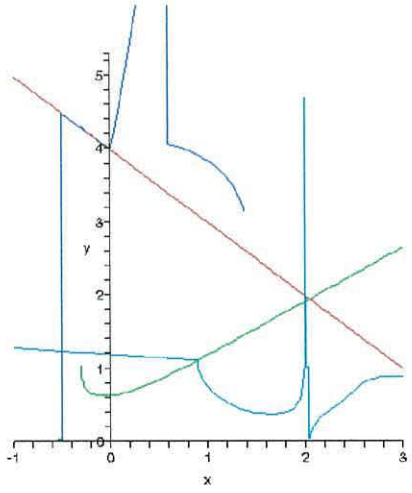


Figure 2.22: 'Contours' for $\phi_7 = 0.01$.

2.5 Interpolation for Smooth Contours

When looking at a standard contour map, such as the one given in Figure 2.23, we can see that the contours provide information regarding the elevation of the terrain

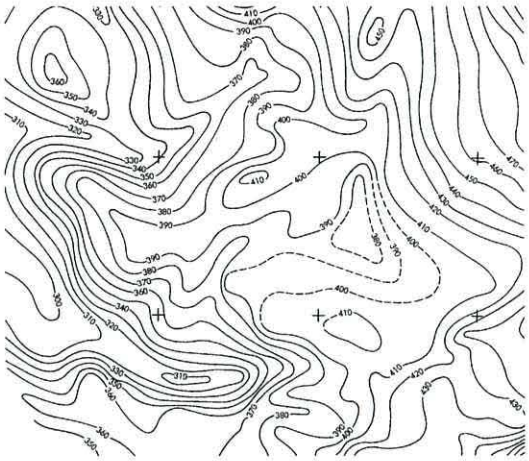


Figure 2.23: Sample contour map of a typical terrain. In this example, the numbers represent elevation above sea level. (For illustration purposes only)

we are interested in. As it is not practical to measure every point over the terrain, the measurement of these elevations would have generally been calculated at specific points, and smooth lines drawn through those of equal height. The points would normally be meshed, and elevation values would be interpolated over the mesh. As we are introducing interpolated values, parts of the terrain may not be accurately described, and so the contours produced are an estimation of the true data. We do not know the exact terrain between data points, and so opting to use straight lines to join points of equal value is currently one of the most accurate methods of displaying the results. This is shown in Figure 2.24, where the blue contour is drawn using piecewise linear sections around a coarse triangulation.

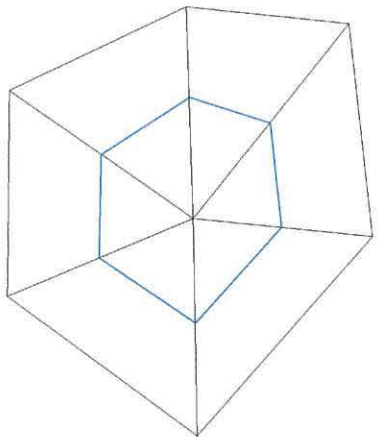


Figure 2.24: Simple linear contour on a small triangulation.

Although piecewise linear contours produce good results for the data, it is not aesthetically pleasing to the eye, as we are aware that real data rarely has corners and straight lines, and so it would be preferential if we could smooth the data in some way.

One method would be to subdivide the triangles into smaller triangles, and to either measure or estimate the data at the new triangle corners. This produces smoother contours, but still has corners and straight lines. We could repeat the subdivision on these sub-triangles to produce smaller triangles, but this produces a large amount of data which may be estimations derived from estimations. Another problem arises as to deciding when to stop the subdivision to produce a contour map which is smooth enough.

Another method would be to assess the data by hand and smooth the lines by redrawing the contours. This requires human intervention, and so problems may arise since it is possible for two people to produce two different contour maps from the same data.

We could also use linear interpolation to estimate the point where the contour line intersects the triangle sides. A smooth curve is then drawn through these points. However, doing this can cause problems as it is possible for contours of different levels to cross each other. A solution to this would be to provide an algorithm to plot the contours of a function with a continuous first derivative, hence removing this problem.

In the next section we will investigate methods which produce smooth contours from the outset, which removes the requirement for further subdivision.

2.6 Bézier Curves

Bézier curves, as mentioned above, produce smoooth contours and do not require further subdivision. Because of this, Bézier curves are ideal for contouring algorithms and we will use contouring algorithms using Bézier curves to give smooth derivatives over the interpolation domain.

Bézier curves were invented independently by both de Casteljau and Bézier, who

were engineers for two different car companies in France in the late 1950s and early 1960s. These curves are relatively easy to describe and control. The idea underlying Bézier curves lies in the weighting of the parametric functions by the coordinates of certain intermediate points. These intermediate points enable the formation of curved triangles, i.e. triangles with curved faces, as opposed to linear triangles, and improve on the accuracy of Finite Element interpolation.

The curved triangles can be used to interpolate the corner points exactly, as well as the slopes at these corners, ensuring that the corner slopes match the slopes of the straight lines between each end point and its nearest intermediate data point. These curves are known as *linear Bézier curves*. Quadratic and higher order Bézier curves are also available, and many applications need true space curves, so we need to create a general polynomial curve of arbitrary degree n. An n^{th} order Bézier curve requires $n+1$ control points, so a cubic Bézier curve can be described parametrically as

$$\begin{aligned} x(t) &= a_x(1-t)^3 + 3b_x(1-t)^2t + 3c_x(1-t)t^2 + d_xt^3, \\ y(t) &= a_y(1-t)^3 + 3b_y(1-t)^2t + 3c_y(1-t)t^2 + d_yt^3, \\ 0 &\leq t \leq 1, \end{aligned}$$

where the coefficients are the coordinates of the four control points $A(a_x, a_y)$, $B(b_x, b_y)$, $C(c_x, c_y)$, and $D(d_x, d_y)$. The points A and D correspond to the end points of the curve, and the intermediate points B and C determine the tangential direction at the two end points.

The case for a general polynomial curve of degree n is defined using de Casteljau's algorithm below.

Algorithm 2.6.1 (de Casteljau).

Given $\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^3$ *and* $t \in \mathbb{R}$.

Set $\mathbf{b}_i^r(t) = (1-t)\mathbf{b}_i^{r-1}(t) + t\,\mathbf{b}_{i+1}^{r-1}(t)$ *and* $\mathbf{b}_i^0(t) = \mathbf{b}_i$,
where $r = 1, \ldots, n$ *and* $i = 0, \ldots, (n-r)$.

Then $\mathbf{b}_0^n(t)$ *is the point with parameter* t *on the Bézier curve* $\mathbf{b}^n$,
hence $\mathbf{b}^n(t) = \mathbf{b}_0^n(t)$.

The polygon formed by $\mathbf{b}_0, \ldots, \mathbf{b}_n$ is called the *Bézier polygon* or *control polygon*

of the curve $\mathbf{b}^n$. Similarly, the vertices of the polygon are called *Bézier points* or *control points.*

The intermediate coefficients $\mathbf{b}_i^r(t)$ can be written as a triangular array of points known as the *de Casteljau scheme.* If we were to have a cubic curve, the points could be arranged as follows:

$$\begin{array}{cccc} \mathbf{b}_0 & & & \\ \mathbf{b}_1 & \mathbf{b}_0^1 & & \\ \mathbf{b}_2 & \mathbf{b}_1^1 & \mathbf{b}_0^2 & \\ \mathbf{b}_3 & \mathbf{b}_2^1 & \mathbf{b}_1^2 & \mathbf{b}_0^3. \end{array}$$

This array design suggests that a two-dimensional array is required for the de Casteljau algorithm. However, it is possible to use just the left-most column and to overwrite these values when required.

Using Bézier curves in the de Casteljau algorithm enables us to infer many of the important properties of Bézier curves in the algorithm:

- Bézier curves do not change under affine maps or affine parameter transformations, and so the order of applying the map or transformation with the computation of the points $\mathbf{b}_i$ is not important.

- For $t \in [0,1]$, $\mathbf{b}^n(t)$ lies in the convex hull of the polygon since each $\mathbf{b}_i^r$ is obtained using a combination of other internal points and so we do not produce any points outside the convex hull of the $\mathbf{b}_i$. This convex hull property gives us an important consequence that a planar control polygon always generates a planar curve.

- The Bézier curve passes through $\mathbf{b}_0$ and $\mathbf{b}_r$, and for the cases $t = 0$ and $t = 1$, we can easily verify that $\mathbf{b}^n(0) = \mathbf{b}_0$ and $\mathbf{b}^n(1) = \mathbf{b}_n$. This means that we can interpolate the end points exactly, which is essential for most interpolation domains.

For higher values of n this algorithm is not computationally efficient, although it is a very stable method as well as an important tool for further investigations. Typically, we will be using values no higher than $n = 2$, and so efficiency for higher

values of n is not an issue. When $t = 0$, we can write the control points of the Bézier polygon as a Taylor expansion given by

$$\binom{n}{i} \Delta^i \mathbf{b}_0,$$

where Δ^i are the forward differences, namely

$$\begin{aligned}
&\Delta^0 \mathbf{b}_0 = \mathbf{b}_0 \\
&\Delta^1 \mathbf{b}_0 = \Delta \mathbf{b}_0 = \mathbf{b}_1 - \mathbf{b}_0 \\
&\Delta^2 \mathbf{b}_0 = \Delta (\Delta \mathbf{b}_0) = \Delta \mathbf{b}_1 - \Delta \mathbf{b}_0 = \mathbf{b}_2 - 2\mathbf{b}_1 + \mathbf{b}_0 \\
&\quad \text{etc.}
\end{aligned}$$

The Bézier points can also be expanded in terms of Bernstein polynomials $B_i^n(t)$:

$$\mathbf{b}_0 B_0^n(t) + \mathbf{b}_1 B_1^n(t) + \cdots + \mathbf{b}_n B_n^n(t),$$

where Bernstein polynomials of degree n are defined by

$$B_i^n(t) = \binom{n}{i} (1-t)^{n-i} t^i, \quad i = 0, 1, \ldots, n$$

and t is the local coordinate of the interpolated curve segment. Similar to de Casteljau's algorithm, we have that the $B_i^n(t)$ are invariant under affine transformations, lie in the convex hull, and can be created by repeated linear interpolation since we can also define the $B_i^n(t)$ as

$$B_i^n(t) = (1-t) B_i^{n-1}(t) + t\, B_{i-1}^{n-1}(t).$$

To find the derivatives with respect to a point u of a Bézier curve, we have two possible directions.

For the de Casteljau scheme we have the first and second derivatives given respectively by

$$\frac{n}{\Delta u_0} \Delta \mathbf{b}_{n-1}^{n-1} \quad \text{and} \quad \frac{n(n-1)}{(\Delta u_0)^2} \Delta^2 \mathbf{b}_{n-2}^{n-2}$$

since the de Casteljau steps commute.

For the case of derivatives of Bernstein polynomials, the r^{th} derivative is given by

$$\frac{n!}{(n-r)!} \cdot \frac{1}{(\Delta u_0)^r} \big(\Delta^r \mathbf{b}_0 B_0^{n-r}(t) + \cdots + \Delta^r \mathbf{b}_{n-r} B_{n-r}^{n-r}(t) \big).$$

At the endpoint $u = u_0$, $t = 0$, and so the Bernstein representation reduces to just

$$\frac{n!}{(n-r)!} \cdot \frac{1}{(\Delta u_0)^r} \Delta^r \mathbf{b}_0.$$

From the above we can deduce that the tangent vector at one end point $u = u_0$ is $\frac{n}{\Delta u_0}(\mathbf{b}_1 - \mathbf{b}_0)$, and similarly at the other end point $u = u_1$ we have a tangent vector of $\frac{n}{\Delta u_0}(\mathbf{b}_n - \mathbf{b}_{n-1})$. The Bézier polygons therefore provide an idea of the shape of the curve that they define, and although the derivatives can be calculated more efficiently using other methods, the above is sufficient for the time being.

In addition to differentiation, we can also integrate Bézier curves, and it is relatively simple to show that

$$\int_0^1 B_i^n(t)dt = \frac{1}{n+1}, \quad \text{for all } i.$$

So for the interval $[u_0, u_1]$ over the curve $f(u)$ we have

$$\int_{u_0}^{u_1} f(u)du = \frac{u_1 - u_0}{n+1}\big(\mathbf{b}_0 + \mathbf{b}_1 + \cdots + \mathbf{b}_n\big).$$

The above equation implies that the definite integral can be found by multiplying the interval length by the average of the Bézier points $\mathbf{b}_i$.

Just as we can use integers from real numbers to form rational numbers, we are also able to use Bézier curves to form rational Bézier curves.

A rational Bézier curve can be expressed in terms of Bézier polynomials as

$$r(u) = \frac{\beta x(t)}{\beta(t)} = \frac{\beta_0 \mathbf{b}_0 B_0^n(t) + \cdots + \beta_n \mathbf{b}_n B_n^n(t)}{\beta_0 B_0^n(t) + \cdots + \beta_n B_n^n(t)}, \tag{2.6.1}$$

where $\beta x(t)$ is the weighted function of $x(t)$ and again t is the local coordinate of the interpolated curve segment [21]. The Bézier points $\mathbf{b}_i$ are each assigned a weight β_i such that if β_i is large enough compared to β_{i-1} and β_{i+1}, then the curve is skewed towards $\mathbf{b}_i$. The curve lies in the convex hull of the $\mathbf{b}_i$ if all of the β_i have the same sign.

As with standard Bézier curves, a rational linear transformation changes the defining Bézier polygon and weights, but not the shape and degree of a rational Bézier curve. This transformation can be defined by three arbitrary points on the curve, together with the corresponding parameter values. In particular, we can

assign the value $t = \infty$ to a point at infinity which in turn causes the degree of the denominator $\beta(t)$ to reduce to $(n-1)$ or less. The point at $t = \frac{1}{2}$ is called the '*shoulder*' s, and we can find this value of s along with the $\mathbf{b}_i$s by solving a homogeneous linear system to determine the weights β_i.

For a rational quadratic Bézier curve, Equation (2.6.1) can be simplified to

$$\mathbf{b}(t) = \frac{\mathbf{b}_0 B_0^2(t) + w\mathbf{b}_1 B_1^2(t) + \mathbf{b}_2 B_2^2(t)}{B_0^2(t) + wB_1^2(t) + B_2^2(t)}, \tag{2.6.2}$$

where w determines the shape of the quadratic contour. If $w < 1$, the curve is an ellipse, if $w = 1$, it is a parabola, and if $w > 1$ we have a hyperbola. We can describe the whole contour using many small Bézier curves, each of which can be parametrised over the interval $[0, 1]$.

Worsey and Farin [49] described an algorithm which takes advantage of these properties, and we shall see their method in Section 3.4. The algorithm pieces together many small rational Bézier curves, controlled using intersections between triangle edges and their neighbouring incentres as control points, as well as the incentres themselves. The triangulation surface is smooth and continuous, and using control points from this surface should produce a continuous set of Bézier curves.

Before we look into the algorithm, we must first look at other methods which could be used to interpolate the data. Although these methods will not be investigated further, they are included for completion and could be used in place of Bézier curves in the methods described in the next few chapters, although Bézier curves are more straightforward to code and are sufficient to produce the outputs required.

2.7 B-Splines

A Spline curve is a piecewise polynomial curve that has certain differentiability constraints. Spline functions are formed by joining polynomials together at fixed points called *knots*. These knots can be thought of in the same way as a knot joining two different pieces of string together. Clearly they must join at the same point, and mathematically these two polynomials are required to join smoothly. In the

most common case, this means that the derivatives must match up to the order one less than the degree. (If they matched up to the derivative whose order equalled the degree, they would be the same polynomial.) Thus a spline function defined in this way has one extra degree of freedom than a polynomial defined over the entire interval.

A B-spline curve (or Basis-spline curve) is a generalisation of Bézier curves, and has many properties derived from Bézier curves, including partition of unity, positivity and local support, as well as being $(n-1)$-times continuously differentiable for a given n. A B-spline curve is given by

$$s(u) = \sum_i \mathbf{d}_i N_i^n(u)$$

where the $\mathbf{d}_i$ are the *control points* or *de Boor points* of the B-splines $N_i^n(u)$ forming the de Boor polygon. The normalised B-splines $N_i^n(u)$ are piecewise polynomials of degree n defined recursively by

$$N_i^0 = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_i^n(u) = \frac{u - u_i}{u_{i+n} - u_i} N_i^{n-1}(u) + \frac{u_{i+n+1} - u}{u_{i+n+1} - u_{i+1}} N_{i+1}^{n-1}(u).$$

It is possible that the equations above can yield $\frac{u}{0}$, and so for completeness this is defined as zero. $N_i^n(u)$ is a step function equal to zero everywhere except on the interval $u_i \leq u < u_{i+1}$. This interval is known as the i^{th} knot span, and can have zero length since knots do not need to be distinct.

B-spline curves possess the four properties previously mentioned, as well as the convex hull property, since any point of the curve lies in the convex hull of the de Boor points defining it. The knot vector uniquely defines the B-spline curves, as this should be clear from the equations given above. The relation between the number of knots $(p+1)$, the degree (n) of N_i^n, and the number of control points $(m+1)$ is given by $p = m + n + 1$. The sequence of knots in the knot vector U is assumed to be non-decreasing, i.e. $u_i \leq u_{i+1}$. Each successive pair of knots represents an interval $[u_i, u_{i+1})$ for the parameter values to calculate a segment of a shape.

If p knots $u_j = \cdots = u_{j+p-1}$ are equal, then the B-spline curve becomes only C^{n-p} continuous at u_j, and in order to ensure that a B-spline curve has non-vanishing support we require that $p \leq n+1$. The local support property implies that a change of one control point d_i only affects a limited part of the B-spline curve.

If for example, $U = \{0,0,0,1,1,1\}$, then we have a B-spline curve of degree 2. Considering u only in the range $[0,1)$ we have

$$\begin{aligned} N_0^0 &= 0 & N_0^1 &= \tfrac{u-0}{0-0}N_0^0 + \tfrac{0-u}{0-0}N_1^0 &&= 0 \\ N_1^0 &= 0 & N_1^1 &= \tfrac{u-0}{0-0}N_1^0 + \tfrac{1-u}{1-0}N_2^0 &&= 1-u \\ N_2^0 &= 1 & N_2^1 &= \tfrac{u-0}{1-0}N_2^0 + \tfrac{1-u}{1-1}N_3^0 &&= u \\ N_3^0 &= 0 & N_3^1 &= \tfrac{u-1}{1-1}N_3^0 + \tfrac{1-u}{1-1}N_4^0 &&= 0 \\ N_4^0 &= 0 \end{aligned}$$

$$\begin{aligned} N_0^2 &= \tfrac{u-0}{0-0}N_0^1 + \tfrac{1-u}{1-0}N_1^1 &&= (1-u)^2 \\ N_1^2 &= \tfrac{u-0}{1-0}N_1^1 + \tfrac{1-u}{1-1}N_2^1 &&= 2u(1-u) \\ N_2^2 &= \tfrac{u-1}{1-1}N_2^1 + \tfrac{1-u}{1-1}N_3^1 &&= u^2 \end{aligned}$$

These are the Bernstein polynomials as used in Bézier curves. B-splines may be thought of as a generalisation of the Bézier representation if

$$U = \{\underbrace{0,\ldots,0}_{n+1},\underbrace{1,\ldots,1}_{n+1}\}.$$

In the de Casteljau algorithm we used recursion to construct the $B_i^n(t)$ for the evaluation of $x(u)$ at a given value t in a Bézier curve. We can do the same for B-splines using the de Boor algorithm.

The de Boor algorithm is the generalisation of the de Casteljau algorithm. If $x \in [u_l, u_{l+1})$, then all $N_i^n(u)$ vanish at x except those with $l \in \{l-n,\ldots,l\}$. The point $s(x)$ is found by repeated linear interpolation of

$$\mathbf{d}_i^k = (1-\alpha_i^k)\mathbf{d}_{i-1}^{k-1} + \alpha_i^k \mathbf{d}_i^{k-1}, \qquad \alpha_i^k = \frac{x-u_i}{u_{i+n+1-k}-u_i}$$

where $\mathbf{d}_i^0 = \mathbf{d}_i$ and $\mathbf{d}_n^k = s(x)$. As with Bézier curves, we can differentiate B-spline curves of degree n, defined over a partition $\{u_k\}$ to obtain a B-spline curve of degree $(n-1)$:

$$D_u s(u) = \sum_i \mathbf{d}_i^{(1)} N_i^{n-1}(u),$$

defined over the same partition, where

$$\mathbf{d}_i^{(1)} = n\frac{\mathbf{d}_i - \mathbf{d}_{i-1}}{u_{i+n} - u_i}.$$

The $D_u s(x)$ can be found using the de Boor algorithm, and to differentiate further, we just repeat the method the required number of times.

We can find all derivatives simultaneously by using Böhm's algorithm [5] and is given below.

Algorithm 2.7.1 (Böhm).

$$\begin{array}{lllllll}
\mathbf{d}_{l-n} & = & \mathbf{d}_{l-n,0} & \mathbf{d}_{l-n,1} & \cdots & \mathbf{d}_{l-n,n-1} & \mathbf{d}_{l-n,n} \\
\mathbf{d}_{l-n+1} & = & \mathbf{d}_{l-n+1,0} & \mathbf{d}_{l-n+1,1} & \cdots & \mathbf{d}_{l-n+1,n-1} & \mathbf{d}_{l-n+1,n} \\
\vdots & & & \vdots & \ddots & & \vdots \\
\mathbf{d}_{l-1} & = & \mathbf{d}_{l-1,0} & \mathbf{d}_{l-1,1} & \cdots & \mathbf{d}_{l-1,n-1} & \mathbf{d}_{l-1,n} \\
\mathbf{d}_{l} & = & \mathbf{d}_{l,0} & \mathbf{d}_{l,1} & \cdots & \mathbf{d}_{l,n-1} & \mathbf{d}_{l,n}
\end{array}$$

Given the array above representing the set of simultaneous equations for the derivatives, and starting with $\mathbf{d}_{i,0} = \mathbf{d}_i$, *solve the lower left triangle of the array by solving*

$$\mathbf{d}_{i,k}(u_{i+n+1-k} - u_i) = \mathbf{d}_{i,k-1} - \mathbf{d}_{i-1,k-1},$$

followed by the upper right triangle by solving

$$\mathbf{d}_{i,k}(x - u_{i+k}) = \mathbf{d}_{i+1,k} - \mathbf{d}_{i,k-1}.$$

This gives us the derivative $D_u^r s(x)$, *where*

$$D_u^r s(x) = n \cdots (n-r)\,\mathbf{d}_{l-n+r,n}.$$

We can also find the derivatives using the Taylor expansion

$$\sum_{i=0}^{n} \binom{n}{i} \mathbf{d}_{l-n+i,n}(u-x)^i$$

giving the de Boor points $\mathbf{d}_{l-n}, \ldots, \mathbf{d}_e$.

As with Bézier curves, integration can be performed on B-splines, where the integrand of a normalised B-spline $N_i^n(u)$ is given by

$$\int_{-\infty}^{\infty} N_i^n(u)du = \frac{1}{n+1}(u_{i+n+1} - u_i).$$

If we make u_a and u_b into knots of multiplicity $n+1$, then for the B-spline function $s(u) = \sum_i \mathbf{d}_i N_i^n(u)$ we obtain

$$I = \int_{u_a}^{u_b} s(u)du = \frac{1}{n+1} \sum_i \mathbf{d}_i (u_{i+n+1} - u_i).$$

If we define the shoulder s in a similar manner for B-splines as for Bézier curves, then we are able to define rational B-splines using $s(u)$ as:

$$r(u) = \frac{\delta s(u)}{\delta(u)} = \frac{\sum_i \delta_i \mathbf{d}_i N_i^n(u)}{\sum_i \delta_i N_i^n(u)},$$

where the algorithms which are previously mentioned are applied simultaneously to the numerator and denominator.

A special type of rational B-splines, known as Non-Uniform Rational B-Splines (or *NURBS*) are seen by many as the most general curve scheme, as they are extremely powerful and complex. The use of multiple knots, repeated control points and rational weights all add to the complexity, and we will see the advantages of using NURBS in the following section.

2.8 NURBS

NURBS are industry-standard tools used to design and display geometrical curves [32, 43]. Since NURBS are generalisations of B-splines, then they should have all properties of B-splines. These include the following:

- NURBS use one form to mathematically represent standard analytical shapes and free form shapes.
- They provide the flexibility to design a large variety of shapes.
- Solutions can be evaluated reasonably quickly by accurate and numerically stable algorithms.
- NURBS are invariant under affine and perspective transforms.
- In addition to being generalisations of B-splines, it follows that NURBS are also generalisations of non-rational and rational Bézier curves and surfaces.

One of the original drawbacks of using NURBS is that extra computer memory is required to define traditional shapes such as circles and ellipses, although this extra memory may not be a significant issue in today's hardware. The extra memory requirement comes from parameters in addition to the control points, although these are required to allow for the desired flexibility for defining parametric shapes.

NURBS are defined by control points and their necessarily associated weights. A NURBS curve $C(u)$ is defined as

$$C(u) = \sum_i R_i^n(u)\mathbf{p}_i$$

where $\mathbf{p}_i$ are the vector control points and $R_i^n(u)$ is defined as

$$R_i^n(u) = \frac{N_i^n(u)w_i}{\sum_{j=0}^{n} N_j^n(u)w_j}.$$

The w_i are the weights associated with the control points and $N_i^n(u)$ are the normalised B-spline basis functions of degree n as given in Equation (2.7.1) on page 43.

For NURBS, the knot vectors u_i in the B-spline basis functions need not have the same intervals, i.e. the knot spacing is non-uniform, leading to a non-periodic knot vector of the form

$$U = \{a, \ldots, a, u_{k+1}, \ldots, u_{m-k-1}, b, \ldots, b\}$$

where a and b are repeated $k+1$ times. The multiplicity of a knot affects the continuity of the parameters at this knot. Non-periodic B-splines, such as NURBS, are C^∞ continuously differentiable in the interior of the knot span, and C^{k-M-1} continuously differentiable at a knot, where M is the multiplicity of the knot. The end knot points for NURBS (u_k, u_{k+1}) with multiplicity $k+1$ coincide with the end control points $\mathbf{p}_0$, $\mathbf{p}_n$.

Since it is possible for the knot-spacing to be non-uniform, the B-splines are no longer the same over each interval (u_i, u_{i+1}) and the degree of the B-spline can vary. If we consider the whole range of parameter values represented by the knot vector, the different B-splines build up continuous overlapping blending functions $N_i^n(u)$ as defined in Equation (2.7.1) on page 43 over this range. Similar to previously defined functions, these blending functions have the following properties:

1. $N_i^n(u) \geq 0$ for all i, n, u.

2. $N_i^n(u) = 0$ if $u \notin [u_i, u_{i+1})$, implying local support of $k+1$ knot spans, where $N_i^n(u) \neq 0$.

3. If u is in the interval $[u_i, u_{i+1})$, the non-vanishing blending functions are $N_{i-n}^n(u), \ldots, N_i^n(u)$.

4. $\sum_{j=i-n}^{i} N_j^n(u) = \sum_{i=0}^{n} N_i^n(u) = 1$ (partition of unity).

5. In case of multiple knots, $\frac{u}{0}$ is defined as zero.

1. and 4. imply the convex hull property, and 2. and 3. suggest that $k+1$ successive control points define a segment of a shape, and a control point is involved in $k+1$ neighbouring shape segments defined over the interval given in 2. We can use these properties in the following example, showing how the shape of a NURBS curve changes as the weight w changes.

Consider a NURBS curve with a control polygon given by

$$\mathbf{p}_0 = [0 \;\; 0] \qquad \mathbf{p}_1 = [1 \;\; 2] \qquad \mathbf{p}_2 = \left[\tfrac{5}{2} \;\; 0\right] \qquad \mathbf{p}_3 = [4 \;\; 2] \qquad \mathbf{p}_4 = [0 \;\; 0]\,.$$

Suppose we wish to determine the point at $t = \frac{3}{2}$ for the second degree NURBS curve with weights given by $w = [1 \;\; 1 \;\; 0 \;\; 1 \;\; 1]$.

The corresponding knot vector is $[0 \;\; 0 \;\; 0 \;\; 1 \;\; 2 \;\; 3 \;\; 3 \;\; 3]$, and the curves are composed of three piecewise rational quadratics—one for each of the interior intervals of the knot vector. As we are looking at $t = \frac{3}{2}$, then the basis functions on the interval $1 \leq t < 2$ are:

$$\begin{aligned}
N_3^0(t) &= 1 \\
N_2^1(t) &= 2 - t \qquad & N_3^1(t) &= t - 1 \\
N_1^2(t) &= \tfrac{(2-t)^2}{2} \qquad & N_2^2(t) &= \tfrac{t(2-t)}{2} + \tfrac{(3-t)(t-1)}{2} \qquad & N_3^2(t) &= \tfrac{(t-1)^2}{2}.
\end{aligned}$$

All other $N_i^j(t)$ values which are not shown have the value zero. The denominator of the NURBS basis functions is given by

$$\begin{aligned}\sum_j N_j^n(t)w_j &= w_0N_0^2(t) + w_1N_1^2(t) + w_2N_2^2(t) + w_3N_3^2(t) + w_4N_4^2(t) \\ &= 0 + \tfrac{(2-t)^2}{2} + 0 + \tfrac{(t-1)^2}{2} + 0 \\ &= \frac{2t^2 - 6t + 5}{2}.\end{aligned}$$

Using this denominator, we can calculate the $R_i^n(t)$ to be:

$$\begin{aligned}R_0^2(t) &= 0 \\ R_1^2(t) &= \frac{(2-t)^2/2}{(2t^2-6t+5)/2} = \frac{(2-t)^2}{2t^2-6t+5} \\ R_2^2(t) &= 0 \\ R_3^2(t) &= \frac{(t-1)^2/2}{(2t^2-6t+5)/2} = \frac{(t-1)^2}{2t^2-6t+5} \\ R_4^2(t) &= 0,\end{aligned}$$

and so when $t = \frac{3}{2}$ we have

$$R_0^2(\tfrac{3}{2}) = 0 \qquad R_1^2(\tfrac{3}{2}) = \frac{1}{2} \qquad R_2^2(\tfrac{3}{2}) = 0 \qquad R_3^2(\tfrac{3}{2}) = \frac{1}{2} \qquad R_4^2(\tfrac{3}{2}) = 0.$$

The corresponding point on the NURBS curve is $P(\frac{3}{2}) = \frac{1}{2}\begin{bmatrix}1 & 2\end{bmatrix} + \frac{1}{2}\begin{bmatrix}4 & 2\end{bmatrix} = \begin{bmatrix}\frac{5}{2} & 2\end{bmatrix}$.

Similarly if we were to change the weights to $w = [1 \;\; 1 \;\; 5 \;\; 1 \;\; 1]$, we would obtain a denominator of

$$\tfrac{(2-t)^2}{2} + 5\left(\tfrac{t(2-t)}{2} + \tfrac{(3-t)(t-1)}{2}\right) + \tfrac{(t-1)^2}{2} = -4t^2 + 12t - 5,$$

giving new values of $R_i^n(t)$:

$$\begin{aligned}R_0^2(t) &= 0 \\ R_1^2(t) &= \frac{(2-t)^2}{2(-4t^2+12t-5)} \\ R_2^2(t) &= \frac{5t(2-t) + 5(3-t)(t-1)}{2(-4t^2+12t-5)} = \frac{5(-2t^2+6t-3)}{2(-4t^2+12t-5)} \\ R_3^2(t) &= \frac{(t-1)^2}{2(-4t^2+12t-5)} \\ R_4^2(t) &= 0.\end{aligned}$$

Now when $t = \frac{3}{2}$, we have

$$R_0^2(\tfrac{3}{2}) = 0 \qquad R_1^2(\tfrac{3}{2}) = \frac{1}{32} \qquad R_2^2(\tfrac{3}{2}) = \frac{30}{32} \qquad R_3^2(\tfrac{3}{2}) = \frac{1}{32} \qquad R_4^2(\tfrac{3}{2}) = 0,$$

giving a point on the NURBS curve at $\frac{1}{32}[1 \;\; 2] + \frac{30}{32}\left[\frac{5}{2} \;\; 0\right] + \frac{1}{32}[4 \;\; 2] = \left[\frac{5}{2} \;\; \frac{1}{8}\right]$.

NURBS curves for the weights $\left[1 \;\; 1 \;\; \frac{1}{4} \;\; 1 \;\; 1\right]$ and $[1 \;\; 1 \;\; 1 \;\; 1 \;\; 1]$ have also been calculated, giving respective points of $\left[\frac{5}{2} \;\; \frac{8}{7}\right]$ and $\left[\frac{5}{2} \;\; \frac{1}{2}\right]$. A graph of the four NURBS curves can be seen in Figure 2.25.

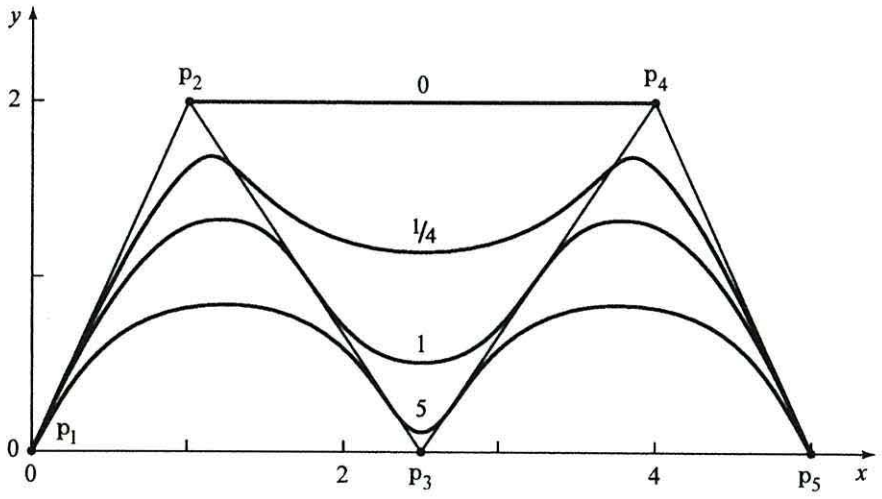


Figure 2.25: NURBS curves for weights $[1 \;\; 1 \;\; w \;\; 1 \;\; 1]$, where $w = 0, \frac{1}{4}, 1, 5$.

Although NURBS is the most general method for drawing curves, we will not be going into this level of generality. We could use B-splines to draw contours, although the contouring method described in the following chapter only requires Bézier curves in order to draw smooth contours.

2.9 Summary

In this chapter, we have provided the basis for the methods discussed in the following chapters. We have reviewed many surface approximation/interpolation methods, focusing on triangle-based interpolation. Straight-line contours are able to contour data sets where other methods may not be able to, although often these contours are jagged and contain sharp corners. This led to investigating natural neighbour bases

as a new method of interpolation for contouring algorithms. Straight-line contours using the natural neighbour algorithm improve on straight-line contours over a standard triangulation, although we will see they can often still appear jagged. If the mesh is retriangulated with a finer triangulation, the natural neighbour algorithm produces smoother contours. Although the natural neighbour algorithm acts locally, resulting in a faster retriangulation, the triangulation process itself must produce a Delaunay triangulation for the natural neighbour algorithm to work. This means that any subdivision will not be sufficient for a contour map, and often data which has been provided already with a triangulation would either need a check to ensure that the triangulation is Delaunay, or would need to be retriangulated. This could be seen as an inefficient method since it may discard information which can be useful, particularly the connectivity between nodes.

We also looked at a standard method of interpolation – Bézier curves, as well as its generalisations into B-splines and NURBS. As B-splines and NURBS are not required for the contouring algorithms in this thesis, they are mentioned for completeness and not investigated further. Worsey and Farin [49] describe an algorithm which pieces together many small rational Bézier curves, controlled using intersections between triangle edges and their neighbouring incentres as control points, as well as the incentres themselves. The triangulation surface is smooth and continuous, and using control points from this surface should produce a continuous set of Bézier curves. We shall see this algorithm in action in the following chapter.

Chapter 3

The Worsey-Farin contouring algorithm

This chapter focuses on a major area of contouring that I have been investigating, and provides a basis for the contouring methods used in the following chapter.

3.1 Introduction

Farin in 1986 [19] provided a simple algorithm for describing a contour over a triangle, where the contour has only one section over that triangle, and the contour intersects the boundary of the triangle at two points which are not on the same edge of the triangle. Clearly not all contours are of this form, and so in order to produce accurate contours for the entire triangulation, it may be necessary to subdivide the triangulation so that it satisfies the above conditions for each triangle. Worsey and Farin in 1990 [49] extended Farin's original algorithm to handle all possible contour types over a triangle, and can also handle closed contours within triangles, cases where the contour has disjoint sections inside a triangle and therefore more than two intersections with the boundary, and other degenerate cases.

The algorithm involves subdividing each triangle into six (or twelve) subtriangles and computing all intersections of the contour with the boundary of each triangle to provide us with a list of points defining end points of rational quadratic Bézier curves. The contours are then produced by plotting these Bézier curves, using normal estimation to decide on the shape of each curve. The methods used for this contouring are described in the following sections.

3.2 Powell-Sabin Six Triangle Subdivision

This subdivision is described in Powell and Sabin [33]. Given height values and their respective first derivatives at all the vertices of a triangulation, it is possible to find a method of defining the values to be a piecewise quadratic on each triangle which, when combined with other triangles, produces an approximating function that is continuous with a continuous first derivative over all triangles. Powell and Sabin present many methods of constructing such a piecewise quadratic approximation, and we will see one of these methods put to use. Given a triangulation $\mathcal{T}$ of a point set P, we choose one interior point in each triangle of $\mathcal{T}$ so that, if the two triangles have a common edge, then the line joining their interior points intersects the common edge between its vertices. For example, it is adequate to choose the incentre of each triangle. Given a triangle ABC, let a be the length (Euclidean distance) of the edge from B to C, b be the length of the edge from C to A, and c be the length of the edge from A to B. Then the coordinates of the incentre of the triangles are given by

$$\left(\frac{aA[x]+bB[x]+cC[x]}{a+b+c}, \frac{aA[y]+bB[y]+cC[y]}{a+b+c}\right),$$

where $A[x]$ denotes the x coordinate of the point A, etc. ABC is a typical triangle of $\mathcal{T}$, and O is the interior point chosen, which in our case is the incentre. Initially, Powell and Sabin considered using the circumcentre as the interior point, although it was noted that the circumcentre is only an interior point of a triangle if the triangle is acute. Further problems arise when the circumcentre lies on an edge of the triangle, and so the incentre is a better point to consider since it does not suffer these problems.

The points P, Q and R are the midpoints of the sides of ABC when the sides are part of the perimeter of the whole triangulation. Otherwise we let P be the point where OO' cuts BC, where O' is the interior point that has been chosen in the other triangle of $\mathcal{T}$ that has the side BC (and similarly for Q and R).

We now have a triangulation where every triangle is subdivided into six smaller triangles, as shown in Figure 3.1.

The interpolation method given below is applied on each triangle, and so the

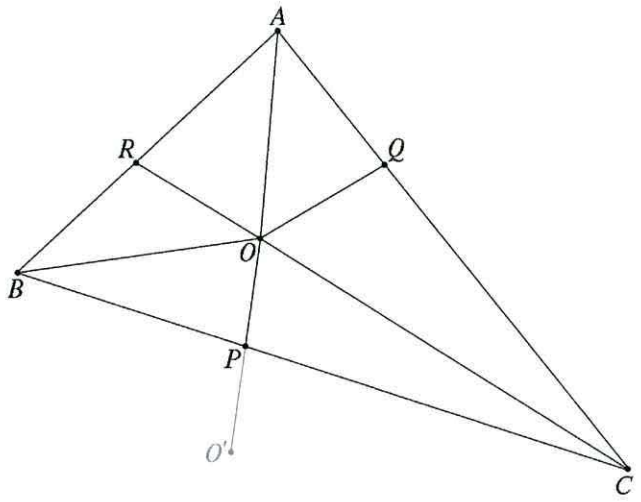


Figure 3.1: 6-triangle Powell-Sabin Subdivision.

result is a piecewise quadratic function that is C^1 continuous and interpolates the data.

Although the points P, Q and R could lie anywhere on the sides BC, CA and AB, it is normal to place them at the mid-points of the sides. The intersection of OO' with BC for our choice of point P (and similarly Q and R) for interior points normally lie near the midpoints and so we achieve similar results. If we were to let P lie away from the mid-point, say one third of the way along BC, then the maximum error when approximating a cubic polynomial increases by a factor of $\frac{27}{8}$ [33]. Similar arguments can be made for points Q and R.

3.3 Bézier Ordinates

Consider a triangle T belonging to a triangulation $\mathcal{T}$, with function values $z = f(x, y)$ defined at each vertex. Unit normal vectors at the vertices are also either known or estimated by some normal vector estimation method, which we shall consider in Section 3.4.3. A quadratic Bézier control net consists of six Bézier ordinates – three at the vertices of the triangle, and three at the midpoints of each edge.

The associated Bézier ordinates to these six control points are of two distinct types – ordinates b_i whose subscripts contain one 2 and all other zeroes (such as

$b_{0,2,0}$), or ordinates b_i whose subscripts have two 1s and one zero (such as $b_{1,1,0}$). These ordinates respectively represent the vertices and tangent planes, and hence are referred to as vertex ordinates and tangent ordinates. The control points and associated Bézier ordinate values (b_i) for a quadratic Bernstein-Bézier triangular patch are shown in Figure 3.2.

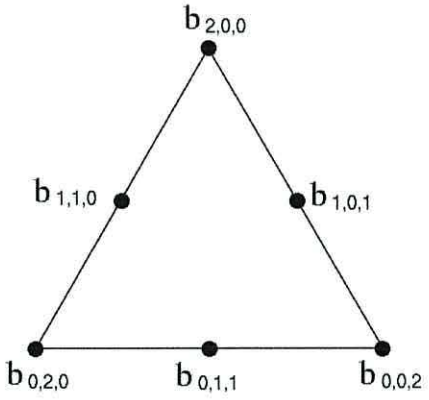


Figure 3.2: Quadratic Bernstein-Bézier triangular patch.

The values of the Bézier ordinates at the vertices are simply equal to the function values at these vertices. The values of the Bézier ordinates at the midpoints are calculated as follows:

All the points on a plane satisfy an equation

$$Ax + By + Cz = D.$$

The first three plane coefficients A, B and C are simply the respective x, y and z components of the unit normal vector of the plane. D is then obtained from the dot product of the normal vector with any point on the plane. The tangent plane at a vertex is therefore

$$z = \frac{D}{C} + \frac{-A}{C}x + \frac{-B}{C}y, \quad C \neq 0, \tag{3.3.1}$$

where x and y are the coordinates of the vertex in question. The remaining Bézier ordinate values come from a projection from the midpoint of the edge to the tangent plane at the vertex.

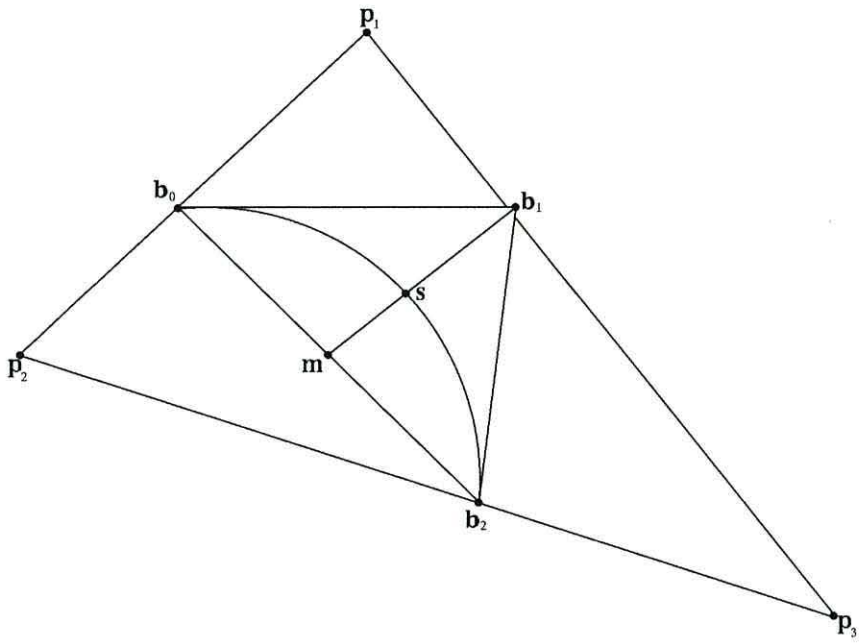


Figure 3.3: Bézier curve over a triangle.

3.4 The Worsey-Farin algorithm

Consider a quadratic polynomial function Q, defined over a domain triangle T with vertices p_1, p_2, and p_3 as shown in Figure 3.3. Q can be written in terms of Bernstein polynomials

$$Q(b_1, b_2, b_3) = \sum_{i+j+k=2} \frac{2!}{i!j!k!} w_{ijk} b_1^i b_2^j b_3^k, \tag{3.4.1}$$

where (b_1, b_2, b_3) is a local barycentric coordinate system defined by these vertices, whose barycentric coordinates are (b_1^i, b_2^j, b_3^k) and $i, j, k \geq 0$. The real coefficients w_{ijk} are the Bézier ordinates. As we would like to contour Q, then we need to solve

$$\{(b_1, b_2, b_3) \in T : Q(b_1, b_2, b_3) = c\},$$

for some contour level c. This means that we need to solve the equation

$$\sum_{i+j+k=2} \frac{2!}{i!j!k!} w_{ijk} b_1^i b_2^j b_3^k = c, \tag{3.4.2}$$

again with $i, j, k \geq 0$. As in Equation (2.6.2), the contour can be parametrised as a rational quadratic of the form

$$\mathbf{b}(t) = \frac{\mathbf{b}_0 B_0^2(t) + w\mathbf{b}_1 B_1^2(t) + \mathbf{b}_2 B_2^2(t)}{B_0^2(t) + w B_1^2(t) + B_2^2(t)},$$

where the $\mathbf{b}_i$ form the Bézier polygon of the curve, the $B_i^2(t)$ are the quadratic Bernstein polynomials, and w is the weight associated with the control point $\mathbf{b}_1$, and so we now need to find $\mathbf{b}_0$, $\mathbf{b}_1$, $\mathbf{b}_2$, and w. Farin [19] provides a simple algorithm for describing a contour over a triangle, where the contour has only one section over that triangle, and the contour intersects the boundary of the triangle at two points which are not on the same edge of the triangle. This is the contour as shown in Figure 3.3. Clearly not all contours are of this form, and so in order to produce accurate contours for the entire triangulation, it may be necessary to subdivide the triangulation so that it satisfies the above conditions for each triangle. Worsey and Farin [49] extended Farin's original algorithm to handle all possible contour types over a triangle, and can also handle closed contours within triangles, cases where the contour has disjoint sections inside a triangle and therefore more than two intersections with the boundary, and other degenerate cases.

If we were to compute all intersections of the contour with the boundary of a triangle, we would be able to use the Worsey-Farin algorithm to decide on the connectivity of the boundary points. The nature of the connection between the boundary points are determined in the next section.

3.4.1 Connecting boundary points

Definition 3.4.1 (Worsey and Farin Definition 3.1). *Starting at one vertex of the domain triangle T, and moving anticlockwise around each edge of this triangle, label the boundary points as $c_1, c_2, \ldots, c_n$; $n \leq 6$. For each boundary point c_i, label the barycentric coordinates of this point as (b_1^i, b_2^i, b_3^i).*

Once we have determined the boundary points, it is necessary to determine which boundary points are joined by a contour section. Worsey and Farin use the following lemma to help:

Lemma 3.4.2 (Worsey-Farin Lemma 3.1). *The two points c_k and c_l; $k, l \in \{1, 2, \ldots, n\}$* may *be connected by a section of the contour lying in the interior of the triangle T if and only if the indices k and l are adjacent entries in the cycle $(1, 2, \ldots, n)$.*

The proof of this lemma follows from realising that as the points are arranged in a cyclic order, joining non-adjacent points would usually require that the contour sections must not cross each other. Although this lemma rules out this possibility, it also removes the possibility of contours as given in Figure 3.4. As a result of

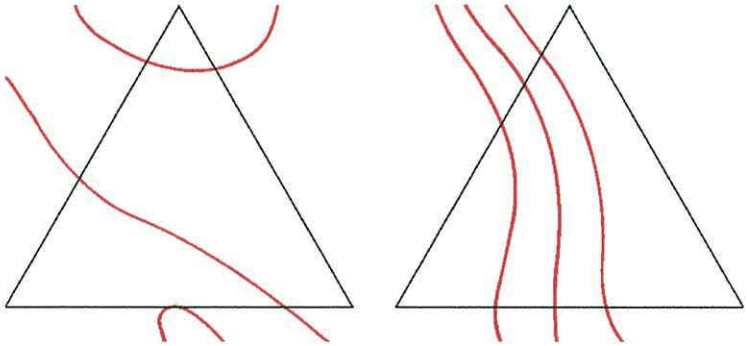


Figure 3.4: Contours not possible from Lemma 3.4.2.

Lemma 3.4.2, we will only need to check neighbouring points for connectivity, and so we only need to consider connectivity for points c_m and c_{m+1}, where $c_{n+1} \equiv c_1$. The idea is to check and see if a line parallel to the line joining c_m and c_{m+1} is tangent to the contour inside the triangle T. If it is, we check to see if it conflicts with any of the other boundary points $c_i; i \neq m, m+1$. Although the idea is the same whether or not the adjacent boundary points lie on the same edge, the two cases will be described separately for ease of reading.

Connecting boundary points on the same edge

As the barycentric coordinates of the edges are parametrised in a cyclic order, then we only need to consider one of these edges and adjust the parameters for the other edges accordingly.

Suppose we are testing whether or not two points are connected by a contour section as given in Figure 3.5.

Assume that the boundary points c_m and c_{m+1} lie on the edge opposite vertex p_2. Now consider the line that passes through a point $(1-\lambda_2,\ \lambda_2,\ 0)$ with direction $(-1, 0, 1)$, i.e. lines that are parallel to the edge opposite p_2 (the edge $\overrightarrow{p_1p_3}$). This

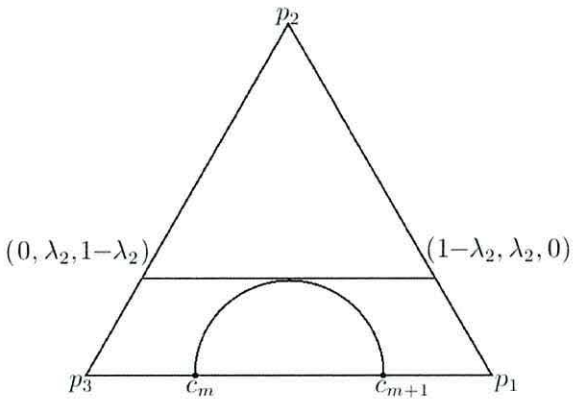


Figure 3.5: Connection between two boundary points on the same edge.

line can be parametrised using barycentric coordinates as

$$\begin{aligned}\lambda_2(t) &= (1-\lambda_2,\ \lambda_2,\ 0) + t(-1, 0, 1) \\ &= (1-\lambda_2-t,\ \lambda_2,\ t).\end{aligned}$$

If we were to substitute this parametrisation into Equation (3.4.2), we have

$$\alpha t^2 + \beta t + \gamma = 0,$$

where

$$\begin{aligned}\alpha &= w_{002} + w_{200} - 2w_{101} \\ \beta(\lambda_2) &= \beta_0 + \beta_1\lambda_2 \\ &= 2(w_{101} - w_{200}) + 2(w_{200} + w_{011} - w_{110} - w_{101})\lambda_2 \\ \gamma(\lambda_2) &= \gamma_0 + \gamma_1\lambda_2 + \gamma_2\lambda_2 \\ &= w_{200} - \mathrm{c} + 2(w_{110} - w_{200})\lambda_2 + (w_{200} + w_{020} - 2w_{110})\lambda_2,\end{aligned}$$

and c is the contour level.

The values for each λ_i can be found by solving the quadratic equation

$$\beta^2 - 4\alpha\gamma = 0.$$

The roots are λ_i^1 and λ_i^2 with $\lambda_i^1 \leq \lambda_i^2$. We define λ_i^* to be the smallest non-negative value of λ_i^1 and λ_i^2, and we can now use the following lemma to determine whether there is a contour to be drawn between boundary points c_m and c_{m+1}.

Lemma 3.4.3 (Worsey-Farin Lemma 3.2). *The points c_m and c_{m+1} on an edge of the domain triangle are connected by a contour section over the interior of the triangle iff the following three conditions are satisfied:*

1. *λ_i^1 and λ_i^2 are real,*

2. *$0 < \lambda^* < 1;\ 0 < (1 - \lambda^* - t^*) < 1;\ 0 < t^* < 1$ where $t^* = -(\beta(\lambda^*)/2\alpha)$, and*

3. *$\lambda^* < b_j;\ j \in \{1, 2, \ldots, n\},\ j \neq m, m+1.$*

If these conditions are satisfied, then we can parametrise the contour as two rational quadratic Bézier curves. The end points are c_m and $l_i(t^*)$ for the first section, and $l_i(t^*)$ and c_{m+1} for the second section. In both cases $\lambda_i = \lambda_i^*$.

Connecting boundary points on different edges

The method for checking boundary points on different edges is essentially the same as for checking on the same edge. However, the tangent lines in this case will no longer be parallel to an edge. This slightly alters the problem, but as before, suppose we are checking to see if two boundary points c_m and c_{m+1} (on different edges) are connected by a contour section over the interior of T. Without loss of generality, assume that they lie on the edges of the triangle T opposite vertices p_2 and p_3 respectively, as shown in Figure 3.6.

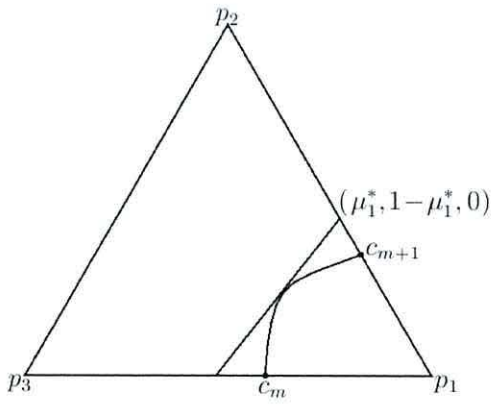


Figure 3.6: Connection between two boundary points on different edges.

Consider a line parametrisation of one of the triangle edges, given by

$$\begin{aligned} l_1(t) &= (\mu_1,\ 0,\ 1-\mu_1) + t(d_1,\ d_2,\ d_3) \\ &= (\mu_1 + td_1,\ td_2,\ 1-\mu_1 + td_3) \end{aligned}$$

with d_i defined to be $d_i = b_i^m - b_i^{m+1}$ for $i = 1, 2, 3$. Inserting these boundary coordinates into Equation (3.4.2) gives

$$\alpha t^2 + \beta t + \gamma = 0,$$

where

$$\begin{aligned} \alpha &= w_{200}d_1^2 + w_{020}d_2^2 + w_{002}d_3^2 + 2(w_{110}d_1d_2 + w_{101}d_1d_3 + w_{011}d_2d_3) \\ \beta(\mu_1) &= \beta_0 + \beta_1\mu_1 \\ &= 2(w_{002}d_3 + w_{101}d_1 + w_{011}d_2) \\ &\quad + 2(-w_{002}d_3 + w_{200}d_1 - w_{011}d_2 + w_{101}(d_3 - d_1) + w_{110}d_2)\mu_1 \\ \gamma(\mu_1) &= \gamma_0 + \gamma_1\mu_1 + \gamma_2\mu_1^2 \\ &= w_{002} - c + 2(w_{101} - w_{002})\mu_1 + (w_{002} + w_{200} - 2w_{101})\mu_1^2, \end{aligned}$$

and c is the contour level. Comparing this with the previous case, we can find values for each μ_i by solving the quadratic equation

$$\beta^2 - 4\alpha\gamma = 0.$$

Again the roots are μ_i^1 and μ_i^2 with $\mu_i^1 \leq \mu_i^2$, and we define μ_i^* to be the smallest non-negative value of μ_i^1 and μ_i^2. We can now use the following lemmas to determine whether there is a contour to be drawn between boundary points c_m and c_{m+1}.

Lemma 3.4.4 (Worsey-Farin Lemma 3.3). *The straight line connecting c_m and c_{m+1} is part of the contour iff*

$$\alpha = \beta(b_k^{m+1}) = \gamma(b_k^{m+1}) = 0,$$

where $k = 1,\ 2$ or $3,\ k \neq i, j$.

Lemma 3.4.5 (Worsey-Farin Lemma 3.4). *If Lemma 3.4.4 above does not hold, then let μ_i^1 and μ_i^2 be the roots as defined previously. The points c_m and c_{m+1} on two edges of the triangle are connected by a contour section over the interior of the triangle iff for $\mu_1^* = \mu_1^1$ or $\mu_1^* = \mu_1^2$,*

1. μ_1^1 *and* μ_1^2 *are real,*

2. $\mu_1^* < 1$,

3. $0 < (\mu_1^* + t^* d_1) < 1$; $0 < t^* d_2 < 1$; $0 < (1 - \mu_1^* + t^* d_3) < 1$ *where* $t^* = -(\beta(\mu_1^*)/2\alpha)$, *and*

4. $b_2^j - (b_2^j/d_2)d_3 \notin \left[\min(\mu_1^*, b_2^{m+1}), \max(\mu_1^*, b_2^{m+1})\right]$;
   $j \in \{1, 2, \ldots, n\}$, $j \neq m, m+1$.

If Lemma 3.4.5 is satisfied, then the contour can again be parametrised as two rational quadratic Bézier curves. The end points are c_m and $l_i(t^*)$ for the first section, and $l_i(t^*)$ and c_{m+1} for the second section. In both cases $\mu_i = \mu_i^*$.

The above case can be repeated for the other two sides of the triangle, with parametrisations given by $l_2(t) = (1 - \mu_1 + td_1,\ \mu_1 + td_2,\ td_3)$ and $l_3(t) = (td_1,\ 1 - \mu_3 + td_2,\ \mu_3 + td_3)$.

Now that we have these steps, we can use the Worsey-Farin algorithm to produce the contours for the domain.

In Worsey and Farin [49], the test to see whether a quadratic Q is elliptic had some terms omitted. This was highlighted and proved in Bloomquist [4]. The Worsey-Farin algorithm below has these extra terms included.

This algorithm and subsequent methods have been coded as a joint project between myself and Walker [44, 45].

3.4.2 The Worsey-Farin algorithm

Algorithm 3.4.6.

1. *Let* Q *be the quadratic as defined in equation* (3.4.1).

2. *Given the six Bézier ordinates of the triangle*

$$W = \{w_{200}, w_{020}, w_{002}, w_{110}, w_{011}, w_{101}\},$$

define

$$w_{\min} = \min(W) \qquad w_{\max} = \max(W).$$

For the contour level c, if

$$w_{\min} \leq c \leq w_{\max}$$

is not satisfied, then the plane $z = c$ cannot intersect the quadratic Q, and so there is no contour to be plotted over the triangle.

3. *Flag whether or not Q is elliptic. If the surface is elliptic then it may have closed contours, otherwise it does not. The surface is elliptic if and only if*

$$\begin{aligned} &w_{110}^2 + w_{101}^2 + w_{011}^2 + 2(w_{200}w_{011} + w_{020}w_{101} + w_{002}w_{110}) \\ &\quad < w_{200}w_{002} + w_{200}w_{020} + w_{020}w_{002} + 2(w_{110}w_{101} + w_{110}w_{011} + w_{101}w_{011}). \end{aligned}$$

4. *Compute all intersections of the contour with the boundary of the triangle. There are n such intersections, with $0 \leq n \leq 6$. The intersections are found by solving a quadratic equation for each edge of the triangle. It is also possible for the triangle edge to be the tangent of the point lying on that edge. We let n_t (where $n_t \leq 3$) denote the total number of points for which this is the case. We can now analyse the value of n (and, if required, n_t) to determine the type of contour that is to be produced.*

   (a) *$n = 0$. Either there is no contour, or it is a closed contour entirely inside the triangle. If Q is elliptic then we need to check for a closed contour, and if there is, it can be parametrised.*

   (b) *$n = 1$. There are two possible solutions. Either it is a single point on the edge of the triangle, or it is a closed contour that is tangent to the boundary. Again, if Q is elliptic, we need to check for a closed contour, and act accordingly. If not, then it is the single point and the contour has finished over the triangle.*

   (c) *$n = n_t = 2$. This is similar to the above case, where either it is two single points on the contour, or it is a closed contour that is tangent to two boundaries.*

   (d) *$n = n_t = 3$. In this case, there must be a closed contour on the triangle. This is easy to contour since the points of the triangle are the control points of the curve, and can be used accordingly.*

(e) This is the most commonly occurring case where the others do not apply. We start at a vertex of the triangle and move anticlockwise around the boundary. Label the contour points $c_1, c_2, \ldots, c_n$ *(*$n \leq 6$*) and check whether point* i *is connected to point* $i+1$ *(as given in Section 3.4.1), where* $i = 1, 2, \ldots, n$ *and* $(n+1) \equiv 1$*. If they are connected, then we can parametrise the section as two rational quadratic Bézier curves. If the two points are not connected, then proceed with the next consecutive pair or points. If a point* c_i *is not connected to either* c_{i-1} *or* c_{i+1}*, then* c_i *is labelled as a single isolated point.*

These cases account for nearly all possibilities, and so the Worsey-Farin algorithm is an efficient method to produce piecewise quadratic contours over a triangulated domain. The exceptions would be if we were to have contours shown in Figure 3.4. These are more likely to occur in coarse triangulations. If we suspect that we may have some of these contours, we could use an alternative contouring method, or produce a finer triangulation over the domain. The Powell-Sabin triangle subdivisions normally eliminate these cases, although it does not guarantee that these extreme cases would not occur.

Once the algorithm is complete, we have a list of points defining end points of rational quadratic Bézier curves to be plotted. We can now construct the Bézier curves for each pair of end points.

Label the end points $\mathbf{b}_0$ and $\mathbf{b}_2$. Estimate the normals at these two end points. Given that the point $\mathbf{b}_0$ (or $\mathbf{b}_2$) lies on an edge or in the interior of the triangle, we can estimate the normal as given in Section 3.4.3.

We can now calculate the location of the middle Bézier point $\mathbf{b}_1$. A method for doing this is described in Worsey and Farin [49], where we can calculate the intersection of the tangent planes at $\mathbf{b}_0$ and $\mathbf{b}_2$ with the plane $z = c$, where c is the contour level.

For $\mathbf{b}_0$ and $\mathbf{b}_2$, we calculate the equations of the tangent planes using the method

described in Section 3.3. This gives us the tangent planes

$$z = Z_0 + X_0 x + Y_0 y$$
$$z = Z_2 + X_2 x + Y_2 y,$$

which, comparing with Equation (3.3.1) gives

$$Z_i = \frac{D}{C}, \qquad X_i = \frac{-A}{C}, \qquad Y_i = \frac{-B}{C}.$$

This results in three quadratic equations of the form $Au = b$, where

$$A = \begin{bmatrix} X_0 & Y_0 & -1 \\ X_2 & Y_2 & -1 \\ 0 & 0 & 1 \end{bmatrix}, \qquad u = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \qquad b = \begin{bmatrix} -Z_0 \\ -Z_2 \\ c \end{bmatrix}.$$

This method gives the correct intersection of the two tangent planes with the contour plane, by consistently providing a satisfactory middle Bézier point $\mathbf{b}_1$.

3.4.3 Estimating normals

As the normal vectors are unknown over the domain, we need to provide reasonable estimates using one of the many methods available. These vectors need to be calculated before the Worsey-Farin algorithm, and so the *Amlin* program calculates the normal vectors immediately after generating the triangulation. At the nodes of the triangulation, we can approximate the values by taking an appropriate weighted average of the normals of the contiguous facets [1]. This results in a normal $\mathbf{n}$ on a triangle given by

$$\mathbf{n} = \frac{1}{|N|} \sum_{i \in N} (w_i \cdot \mathbf{u}_i)$$

where N is the set of indices i such that the unit normal vector neighbouring the facet of interest $\mathbf{u}_i$ for the i^{th} facet is weighted by w_i.

There are many methods used to determine the weighting w_i. For the *Weighted by area* method, each $\mathbf{u}_i$ is weighted by the area of each corresponding facet. The *Weighted by angle* method uses the magnitude of the vertex angle of the corresponding facet. For the *Weighted by Voronoi area* method, each $\mathbf{u}_i$ is weighted

by the area of the Voronoi region of the corresponding facet. Nelson Max's method [29] assigns a weight to each $\mathbf{u}_i$ by

$$w_i = \frac{\sin(\alpha_i)}{|V_i||V_{i+1}|}$$

where α_i is the vertex angle, and V_i and V_{i+1} are the edges of the i^{th} facet extending from the node n. For an unweighted method, set $w_i = 1$ for all $i \in N$.

Now that we are able to estimate the normals at the nodes, we need to find ways of estimating normals at any point on each triangle, including the triangle edges. The simplest method is linear estimation, where we can use the barycentric coordinates (u, v, w) of the triangle to get a normal estimation of

$$N(u, v, w) = \frac{N}{\|N\|}, \qquad N = uN_1 + vN_2 + wN_3,$$

where N_1, N_2, and N_3 are the normals at the three corners of the triangle.

For the triangle as a whole, given N_1, N_2, and N_3 as defined above, we can calculate the normal of the triangle using

$$\begin{aligned}
N_x &= \big((N_{z_3} - N_{z_1})(N_{y_2} - N_{y_1})\big) - \big((N_{z_2} - N_{z_1})(N_{y_3} - N_{y_1})\big) \\
N_y &= \big((N_{z_2} - N_{z_1})(N_{x_3} - N_{x_1})\big) - \big((N_{x_2} - N_{x_1})(N_{z_3} - N_{z_1})\big) \\
N_z &= \big((N_{x_2} - N_{x_1})(N_{y_3} - N_{y_1})\big) - \big((N_{y_2} - N_{y_1})(N_{x_3} - N_{x_1})\big),
\end{aligned}$$

where N_x, N_y, and N_z are the respective x, y, and z components of the triangle normal, and N_{x_i}, N_{y_i}, and N_{z_i} are the respective x, y, and z normal components of the three corners of the triangle.

3.5 Contouring

The current contouring method, as implemented by Walker [44], uses these methods to estimate the normals at the nodes and points inside the triangle. The 'TetSim' program specifically utilises Nelson Max's method for weighting the node normals [29], before subdividing the triangulation using a method given by Powell and Sabin [33]. The node and triangle normals then need to be estimated on these subdivided triangles, and it is these triangles that are used in the Worsey-Farin algorithm to calculate the piecewise quadratic contours [49].

It is also possible to contour the domain using natural neighbours. The Worsey-Farin algorithm favours Nelson Max's node normal weighting method, although when contouring using natural neighbours, it would be logical to use natural neighbours to estimate the normals of the triangles. The normals of the triangles are estimated by finding the natural neighbours of the incentre of the triangle, and calculating the derivative of the Voronoi areas A_I by differentiating Equation (2.4.5) on page 20 to give

$$A_{I,x} = \frac{1}{2}\sum_{i=0}^{N-1}\left(\frac{\partial x_i}{\partial x}y_{i+1} + x_i\frac{\partial y_{i+1}}{\partial x}\right) - \left(\frac{\partial x_{i+1}}{\partial x}y_i + x_{i+1}\frac{\partial y_i}{\partial x}\right) \tag{3.5.1}$$

$$A_{I,y} = \frac{1}{2}\sum_{i=0}^{N-1}\left(\frac{\partial x_i}{\partial y}y_{i+1} + x_i\frac{\partial y_{i+1}}{\partial y}\right) - \left(\frac{\partial x_{i+1}}{\partial y}y_i + x_{i+1}\frac{\partial y_i}{\partial y}\right) \tag{3.5.2}$$

where again $x_N \equiv x_0$ and $y_N \equiv y_0$.

As ϕ_I is defined to be

$$\phi_I(\mathbf{x}) = \frac{A_I(\mathbf{x})}{A(\mathbf{x})},$$

then differentiating ϕ_I with respect to x and y gives

$$\phi_{I,x}(\mathbf{x}) = \frac{A_{I,x}(\mathbf{x}) - \phi_I(\mathbf{x})A_{,x}(\mathbf{x})}{A(\mathbf{x})} \quad \text{and} \quad \phi_{I,y}(\mathbf{x}) = \frac{A_{I,y}(\mathbf{x}) - \phi_I(\mathbf{x})A_{,y}(\mathbf{x})}{A(\mathbf{x})},$$

where $A_{I,x}$ and $A_{I,y}$ are defined in Equations (3.5.1) and (3.5.2) respectively. These can then be used to give gradients in the x, y, and z directions using

$$\Delta_x = \sum_I z_I\phi_{I,x}(\mathbf{x})$$

$$\Delta_y = \sum_I z_I\phi_{I,y}(\mathbf{x})$$

$$\Delta_z = \sum_I z_I - x_I\Delta_x - y_I\Delta_y,$$

which can then be used to derive the normals for the triangles. As the gradient is not defined at the nodes in a Voronoi diagram, then we cannot use natural neighbours to estimate the node normals. Instead we need to rely on the previous method for normal estimations.

3.6 Comparing contouring methods

In order to compare the contouring methods we have mentioned, we will consider two data sets – one is a stratigraphic horizon with a discontinuity (inclined fault) showing rapid elevation change (subsequently referred to as the "stratigraphic horizon" data set), and the other is a surface with minor perturbations – this is especially useful since data sets which are almost flat in the third dimension are notoriously difficult to contour.

For the stratigraphic horizon data set, we will contour the maximum burial depth values over the horizon. The original mesh is given in Figure 3.7(a), contours using straight lines and the natural neighbour method are in Figure 3.7(b), and contours produced using the Worsey-Farin algorithm over a Powell-Sabin subdivision [44] is in Figure 3.7(d). Larger versions of the contour maps are given in Appendix A. The Worsey-Farin algorithm used over a Powell-Sabin triangle subdivision will subsequently referred to as *WFPS*.

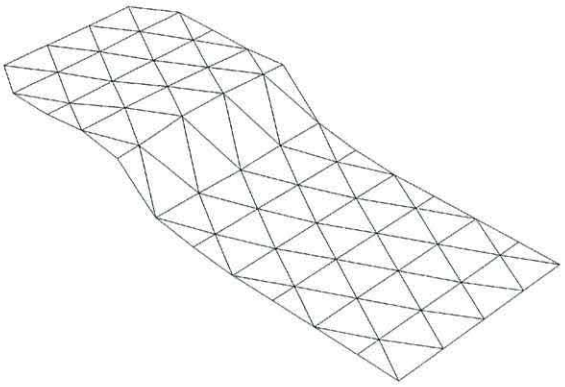
Clearly the contours using the natural neighbour algorithm are not as smooth as those produced over a Powell-Sabin triangle subdivision. If we were to retriangulate the original data set so that it has a similar number of triangles as the subdivision, we can see that the natural neighbour contour map produces comparable results. This is shown in Figure 3.7(c).

For the surface with minor perturbations we are contouring elevation values, as shown in Figure 3.8. Larger versions of the contour maps are given in Appendix B.

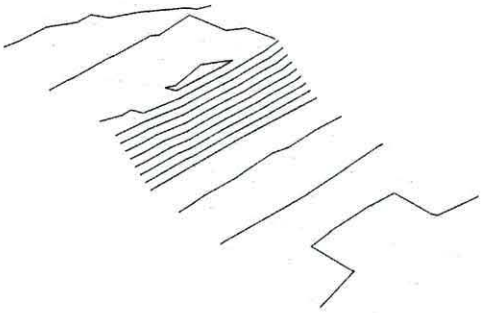
We see from Figure 3.8(b) that a straight line contour map over the subdivided data set gives improved results over the original coarse triangulation in Figure 3.8(a), although Bézier curves over the subdivided triangles improve this further (3.8(c)).

If we were to consider computational power, then both the WFPS and the natural neighbour contouring over a finer triangulation take a similar amount of time.

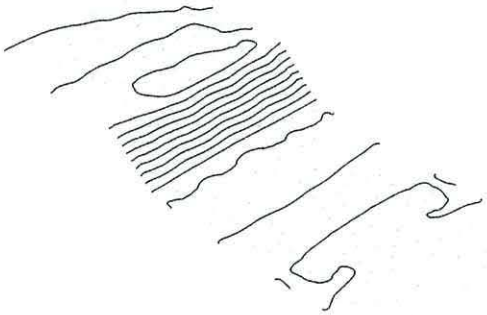
As stated previously, the locality of the natural neighbour method means that if we wished to consider only part of the domain, we would only need to retriangulate a small area, leaving the rest of the domain relatively coarse. This is particularly



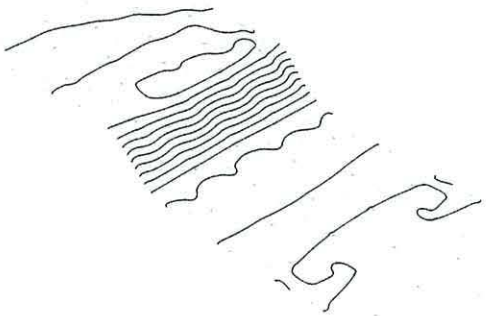
(a) Original Triangulation



(b) Contours using the Natural Neighbour algorithm

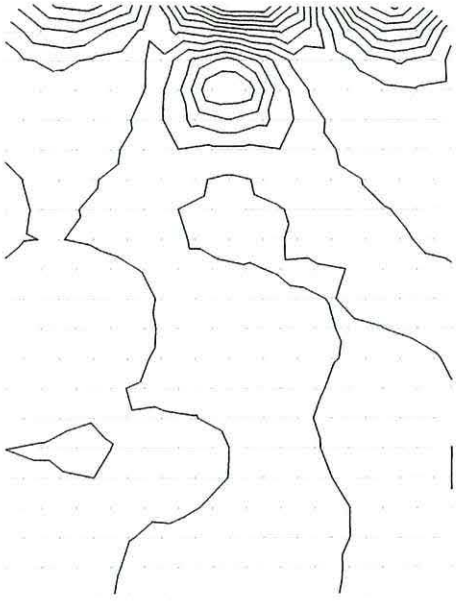


(c) Contours using the Natural Neighbour algorithm over a retriangulated domain

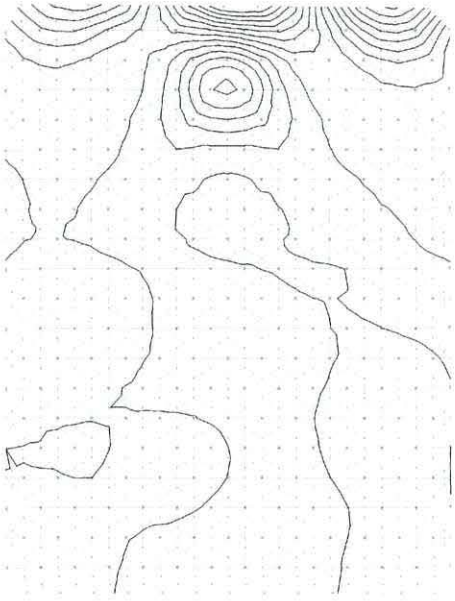


(d) Bézier Curves over a Powell-Sabin Triangle Subdivision

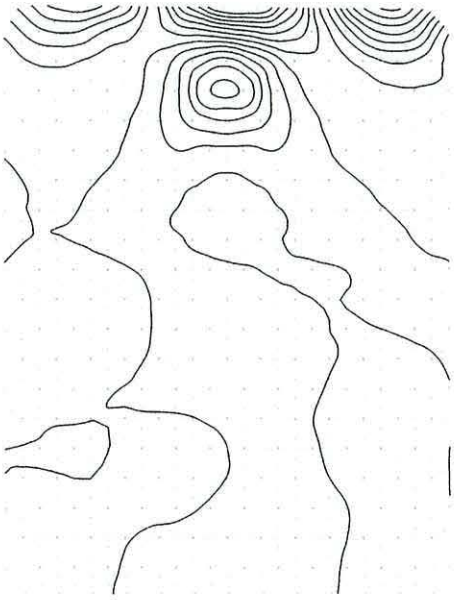
Figure 3.7: Contouring Maximum Burial Depth Values Over A Stratigraphic Horizon



(a) Linear contours over the original triangulation



(b) Straight Line Contours over a Powell-Sabin Triangle Subdivision



(c) Bézier Curves over a Powell-Sabin Triangle Subdivision

Figure 3.8: Contouring Elevation Values Over A Surface With Minor Perturbations

useful if we know *a priori* of particular areas that would have many contours. This retriangulation would result in the natural neighbour contouring method being faster than WFPS, as the local retriangulation would need to be subdivided again using Powell-Sabin subdivision to ensure contour continuity.

3.7 Summary

In this chapter we have investigated the Worsey-Farin algorithm and used the structure in Walker's TetSim program [44] to produce contour outputs via Amlin. Natural neighbour contours produced using TetSim were not as smooth as WFPS over the original domain, although this was expected as the triangulation used for natural neighbour contouring had not been subdivided. When we used a domain retriangulated to the same density as the Powell-Sabin subdivided triangulation, the natural neighbour contours were comparable to the WFPS contours.

In the following chapter we shall investigate a third method, which uses a subdivision surface. The method subdivides triangles using Butterfly Subdivision and is built on the data structures of the TetSim program. Using the existing TetSim data structures therefore makes it more straightforward to compare the outputs and computation time of each method to determine the most efficient method for contouring data.

Chapter 4

Butterfly Subdivision

4.1 Introduction

Subdivision surfaces are a way to describe a surface using a piecewise polygonal model. The surface and its piecewise polygonal model can be of any shape or size, although unlike polygonal models the surface itself can be perfectly smooth. Subdivision surface schemes allow you to take the original polygonal model and produce an approximation of the surface by adding vertices and subdividing existing polygons. Subdivision surfaces are typically used in the gaming and animation industries. The spread between high and low powered computing forces game developers to cater for the general public who used low powered computers, whilst also including features which make the most of the advanced hardware of the hardcore gamer. Animators such as Pixar require the models on screen to be smooth without any obvious jagged edges or other flaws. If the camera was to zoom in on the model, it would be easier to subdivide the polygons of the model than it would be to create multiple copies of the model with varying resolutions. This has become even more important with the advent of high definition and 3D films. Pixar used Catmull-Clark surfaces for their animation in Geri's Game and further information can be found in [14].

This is where subdivision schemes are important, and one important and useful feature of these schemes is *locality*, so no global system of equations needs to be solved.

In this chapter we use subdivision surfaces in a novel way – we use the surfaces as a basis for contouring algorithms. This method could theoretically be used for any data over a surface, although in this thesis we will concentrate on geological surface data.

We now define the basis of subdivision surfaces. Clearly a subdivision surface is a surface generated through some form of subdivision. Every subdivision surface starts with an original polygonal surface, or *control net*. Then the surface is subdivided into additional polygons and the vertices are moved according to some set of rules. The rules for moving the vertices are different from scheme to scheme, and it is these rules that determine the properties of the surface. The rules of most schemes, including the ones discussed below, involve retaining the original vertices, optionally moving them, and introducing new vertices. There are schemes that remove the old vertices at each step, but these are rarely used and will not be investigated here.

The most common interpolating scheme used in subdivision is based on piecewise linears. Unfortunately piecewise linear interpolation of the original sparse data is often not smooth enough for many applications, including contouring, as previously mentioned in Section 2.5.

With Bézier or B-spline patches, modelling complex surfaces involves trying to cover them with triangular or rectangular patches. This is not a simple task and often not possible since some of the patch edges are not allowed to be degenerate. Furthermore, changing values of the object can make continuity very difficult, and it is likely that the model will show creases and artefacts near patch seams.

This is where subdivision surfaces can be useful. It is possible to make a subdivision surface out of any arbitrary (preferably closed) mesh, which means that subdivision surfaces can consist of arbitrary topology, due to the fact that the control net and the eventual surface (or *limit surface*) are topologically the same. On top of that, since the mesh produces a single surface, it is possible to alter the control net without worrying about seams or other continuity issues.

Almost every subdivision scheme has C^1 continuity everywhere. Some have C^2 continuity in some places, but the majority have areas where C^1 is the highest

attainable level of continuity. Rigorously proved continuity properties are a major advantage of some subdivision schemes.

We could theoretically model any surface with as many polygons as we wanted, although the polygons which have none of the original data points would have interpolated nodes and edges. These nodes and edges may or may not be interpolated correctly and so would need some restriction as to how much the interpolated data can deviate from simple linear interpolation. In the Worsey-Farin algorithm, Walker [44] used a Powell-Sabin subdivision [33] which estimated normals at the nodes to determine the location and slope of the new nodes and triangles. The reason for using a subdivision model is that additional polygons can be added in order to get closer to the target limit surface, so as to satisfy smoothness or visual criteria.

Subdivision schemes that produce satisfactory contour maps generally have some, if not all, of the following conditions

- Interpolation: The original mesh vertices are retained and new vertices are interpolated between these mesh vertices. All vertices are on the limit surface.
- Locality: As stated previously, the neighbourhood used to create new vertices should be as small as possible to enable fast algorithms with no global system of equations.
- Symmetry: The scheme should exhibit some type of symmetries as the local mesh topology.
- Generality: The scheme should work for all triangulations, including triangles at boundaries.
- Smoothness: For higher-order continuity, we require the scheme to reproduce polynomials up to some power.
- Simplicity: For ease of use, the scheme should only require simple data structures.

While the degree of continuity is generally the same for all subdivision schemes, there are a number of characteristics that vary notably between schemes. One

important aspect of a scheme is whether it is an approximating scheme or an interpolating scheme. If it is an approximating scheme, the vertices of the control net may not lie on the surface itself. So, at each step of subdivision, the existing vertices in the control net are moved closer to the limit surface. The benefit of an approximating scheme is that the resulting surface tends to be very fair, i.e. having few undulations and ripples. Even if the control net is of very high frequency with sharp points, the scheme will tend to smooth it out, as the sharpest points move the furthest onto the limit surface. On the other hand, this can be a disadvantage, as it is harder to envision the end result while building a control net. It may be hard to craft more undulating, rippling surfaces as the scheme attempts to smooth them out and hence can make the scheme difficult to work with.

If it is an interpolating scheme, the vertices of the control net actually lie on the limit surface. This means that at each recursive step, the existing vertices of the control net are not moved. The benefit of this is that it can be much more obvious from a control net what the limit surface will look like, since the control net vertices are all on the surface. However, it can sometimes be difficult to get an interpolating surface to look exactly the way you defined, as the surface can develop unsightly bulges in areas where it strains to interpolate the vertices and still maintain its continuity. This is usually not a tremendous problem.

The preferred vertex valence is another property of subdivision schemes. The valence of a vertex is the number of edges coming out of it, which is usually 6. Most vertices produced by a scheme during subdivision have the same valence. Vertices of a valence 6 are the *regular* vertices of a scheme. Vertices of any other valence are known as *extraordinary* vertices. Their effect depends on the subdivision scheme, but historically there have been problems analysing the limit surface near extraordinary vertices.

Most schemes never produce extraordinary internal vertices during subdivision, so the number of extraordinary vertices is set by the original control net and never changes. This is common for triangular schemes, as they all tend to split the triangles in the same way—by adding new vertices along the edges and breaking each triangle into four smaller triangles. Vertices on the boundary of the domain

usually have valence 4.

4.2 The Polyhedral Scheme

The polyhedral scheme is a simple subdivision scheme, where you subdivide by adding new vertices along the midpoints of each edge, and then break each existing triangle into four triangles using the new edge vertices. A simple example is shown in Figure 4.1. The problem with linear interpolation, of course, is that it does not produce smooth surfaces, nor does it change the shape of the control net.

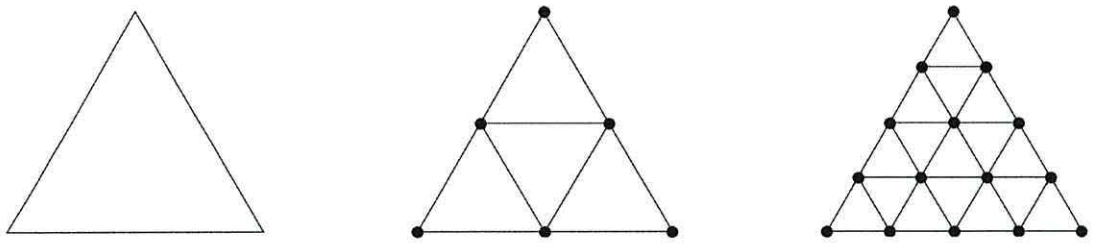


Figure 4.1: Two stages of triangle subdivision for the polyhedral scheme

The scheme is clearly interpolating since it does not move the vertices once created. It is also triangular, since it operates on a triangular mesh. Furthermore, the scheme is uniform since the edge's location does not affect the rules used to subdivide it, and stationary since the same midpoint subdivision is used each time. The surface is only C^0 continuous, since along the edges of polygons it does not have a well-defined tangent plane. The regular vertices of this scheme are of valence 6, which is the valence of new vertices created by the scheme. However, this scheme is simple enough that it does not suffer due to its extraordinary vertices.

The evaluation of the scheme is fairly trivial and it is possible to evaluate it recursively using the subdivision rules. No further evaluation is required, since the points are already on the limit surface.

4.3 The Butterfly Scheme

The next scheme is known as the butterfly subdivision scheme, or, in its current form, the modified butterfly scheme. It shares some similarities with the polyhedral scheme, but has some differences, notably that it is C^1 continuous and therefore produces a smooth surface.

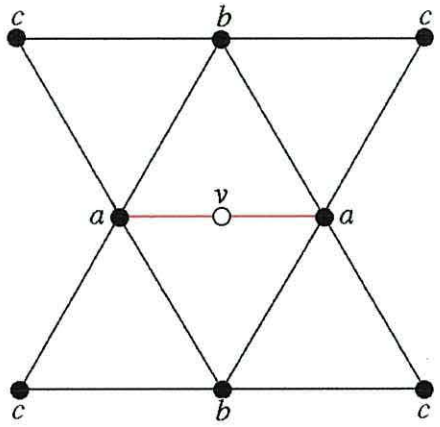


Figure 4.2: The eight-point stencil for the original butterfly scheme.

The butterfly scheme [15, 18] is named due to the shape of the stencil, or map of neighbours used during evaluation. This is given in Figure 4.2. The scheme is interpolating and triangular, and so only adds vertices, $\mathbf{v}$, along the edges of existing triangles. The rules for adding those vertices are simple, and the support is compact. For each edge, sum up the vertices in the stencil-shaped area around that edge, weighting each one by a predetermined weight w. This gives

$$z = \sum_{i=1}^{N} w_i z_i,$$

where

$$\sum w_i = 1.$$

The weights used, corresponding to the vertex labelling in Figure 4.2, are

$$w_a = \frac{1}{2} \qquad w_b = \frac{1}{8} + 2w \qquad w_c = -\frac{1}{16} - w$$

In this case, w is a tension parameter, which controls how "tightly" the limit surface is pulled towards the control net. If $w = -\frac{1}{16}$, then $w_b, w_c = 0$ and the scheme simply linearly interpolates the end points and the surface is not smooth.

In its original form, Dyn et al did not make it clear what happens when the area around an edge does not look like the butterfly stencil. Specifically, if either of the edges' end points is of a valence less than 5, then we do not have sufficient information to use the scheme, leaving no choice but to choose $w = -\frac{1}{16}$ near that area, resulting in a surface that is not smooth near those extraordinary points. This means that while the surface is smooth almost everywhere, there will be isolated jagged points which would stand out visually.

If we refer to the conditions given on page 74, we can see that the original butterfly subdivision scheme satisfies all requirements except for the generality, since it cannot cope with edges which do not look like the butterfly stencil.

In 1993, Dyn et al extended the butterfly scheme to use a ten-point stencil [16], so that the default case was the one shown in Figure 4.3. We now have a new weighting scheme for $\mathbf{v}$, where the new weights are

$$w_a = \frac{1}{2} - w \qquad w_b = \frac{1}{8} + 2w \qquad w_c = -\frac{1}{16} - w \qquad w_d = w.$$

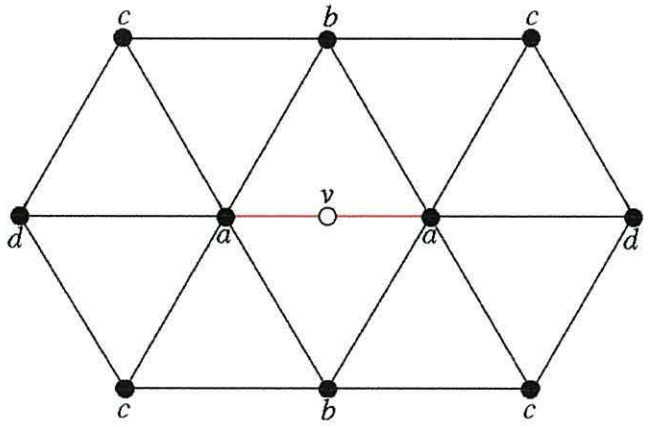


Figure 4.3: The ten-point stencil from the modified butterfly scheme.

Note that by adding w to the **d** points and subtracting it from the **a** points, the stencil's total weighting still adds up to 1 which ensures invariance under affine transformations. Intuitively, this is important because it means that the new point will be in the neighbourhood of the ones used to generate it. If the weights summed

to a different value, say 2, then the point could be twice as far from the origin as the points used to generate it, which would be undesirable.

This new scheme also reduces to the old scheme as a subset—choosing $w = 0$ results in the same rule set as the eight-point butterfly stencil. However, this extension did not address the smoothness problem at extraordinary vertices.

In 1996, Zorin et al published an extension of the butterfly scheme known as the modified butterfly scheme [54]. The primary intent of their extension was to develop rules to use for extraordinary vertices, making the surface C^1 continuous everywhere.

If both of the end points of the edge are regular valence-6 vertices, the scheme uses the standard butterfly's ten-point stencil with the same weights.

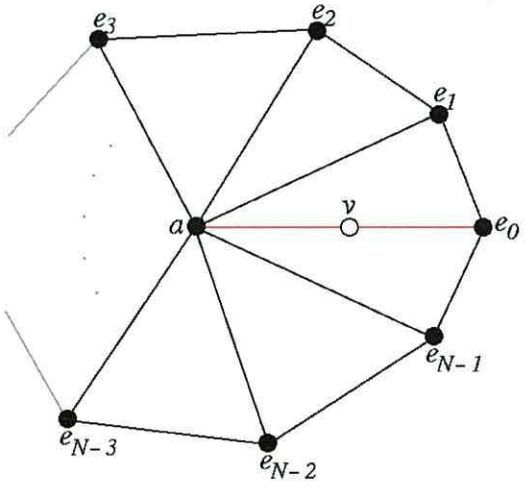


Figure 4.4: The stencil for extraordinary vertices in the modified butterfly scheme. $\mathbf{e}_0$ is a regular vertex of valence 3, whereas vertex $\mathbf{a}$ has N edges, $\mathbf{e}_0 \ldots \mathbf{e}_{N-1}$.

If only one of the end points is extraordinary, the new vertex is computed by the weighted sum of the extraordinary vertex and its neighbours (as shown in Figure 4.4), that is the value of $\mathbf{v}$ can be found by calculating

$$\mathbf{v} = w_a \mathbf{a} + \sum_{k=0}^{N-1} w_{e_k} \mathbf{e}_k,$$

where given the extraordinary vertex's valence of N, the weights used are:

$$N = 3 : w_a = \frac{3}{4},\ w_{e_0} = \frac{5}{12},\ w_{e_1} = -\frac{1}{12},\ w_{e_2} = -\frac{1}{12}$$
$$N = 4 : w_a = \frac{3}{4},\ w_{e_0} = \frac{3}{8},\ w_{e_1} = 0,\ w_{e_2} = -\frac{1}{8},\ w_{e_3} = 0$$
$$N \geq 5 : w_a = \frac{3}{4},\ w_{e_k} = \left(0.25 + \cos\frac{2\pi k}{N} + 0.5 * \cos\frac{4\pi k}{N}\right)/N$$

The full justification for these weights is given in Zorin's thesis [53].

If both end points of the edge are extraordinary, then evaluate the vertex once for each endpoint using the appropriate weights from above, and average the resulting two candidates.

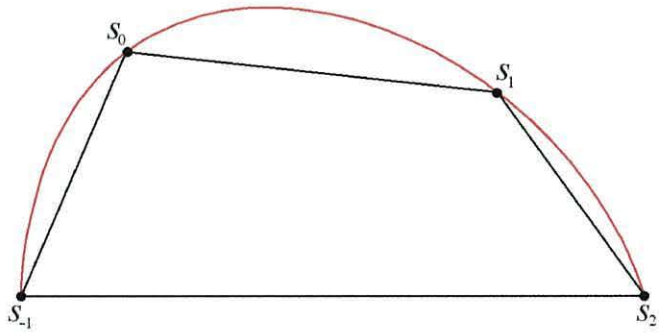


Figure 4.5: The one-dimensional four point interpolatory scheme [17].

Boundary edges are subdivided using the 1-dimensional four point scheme [17] ($s_{-1} = \frac{-1}{16}$, $s_0 = \frac{9}{16}$, $s_1 = \frac{9}{16}$, $s_2 = \frac{-1}{16}$). In this case only other edge points participate in the stencil. A consequence of this rule is that two separate meshes, whose boundaries are identical, will have a matching boundary curve after subdivision. Edges which are not on the boundary but which have a vertex which is on the boundary are subdivided as before while any vertices in the stencil which would be on the other side of the boundary are replaced with "virtual" vertices. These are constructed as required by reflecting vertices across the boundary.

The butterfly scheme is interpolating as points in a control net also lie on the limit surface—the subdivision process does not move existing vertices. It is also triangular as it operates on triangular control nets. It is stationary as it uses the same set of rules every time it subdivides the net, and uniform because every section of the net is subdivided with the same set of rules.

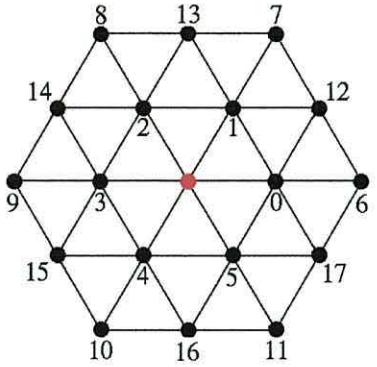


Figure 4.6: The stencil used to calculate the normals of a regular vertex.

As with other contouring schemes, we will sometimes need to calculate/estimate the normals at the nodes. For regular vertices, the process involves the first and second nearest neighbours of the vertex. This results in 18 vertices, and so the scalars, corresponding to the indexing shown in Figure 4.6, are:

$$\begin{aligned} l_0 &= \left\{16, -8, -8, 16, -8, -8, -\frac{8\sqrt{3}}{3}, \frac{4\sqrt{3}}{3}, \frac{4\sqrt{3}}{3}, \right. \\ &\qquad \left. -\frac{8\sqrt{3}}{3}, \frac{4\sqrt{3}}{3}, \frac{4\sqrt{3}}{3}, 1, -\frac{1}{2}, -\frac{1}{2}, 1, -\frac{1}{2}, -\frac{1}{2}\right\} \\ l_1 &= \left\{0, 8, -8, 0, 8, -8, 0, -\frac{4\sqrt{3}}{3}, \frac{4\sqrt{3}}{3}, \right. \\ &\qquad \left. 0, -\frac{4\sqrt{3}}{3}, \frac{4\sqrt{3}}{3}, 0, \frac{1}{2}, -\frac{1}{2}, 0, \frac{1}{2}, -\frac{1}{2}\right\} \end{aligned}$$

Multiplying the vertices by l_0 and l_1 gives us two different tangent vectors. Taking the normalised cross product of these vectors gives us our normal. For extraordinary vertices the normal is actually easier to find, as it depends only on the nearest neighbours of the vertex. The two tangent vectors in this case can be found as

$$t_0 = \sum_{i=0}^{N-1} \mathbf{e}_k \cos\frac{2\pi k}{N}, \quad t_1 = \sum_{i=0}^{N-1} \mathbf{e}_k \sin\frac{2\pi k}{N}$$

Here, t_0 and t_1 are the tangents, N is the vertex valence, and $\mathbf{e}_k$ is the k^{th} neighbour point of the vertex in question, where $\mathbf{e}_0$ is an arbitrary point with $\mathbf{e}_{k+1}$ lying in an anti-clockwise direction to $\mathbf{e}_k$. Crossing the two resulting vectors gives

$$t_0 \times t_1 = \sum_{k<p} \sin\frac{2(p-k)\pi}{N} \mathbf{e}_k \times \mathbf{e}_p = \sum_{m=0}^{N} \sin\frac{2m\pi}{N} \sum_n \mathbf{e}_n \times \mathbf{e}_{(n+m) \bmod N}.$$

Similar to regular vertices, the cross product is a weighted sum of normals of nodes which are first or second nearest neighbours to the node. Normalising our result of the cross product produces the vertex normal. Now that we have the rules for recursively evaluating the surface and node normals, we can perform the subdivision.

For subdivision: given a net, we need to subdivide it into a more refined net. Working from the modified butterfly rules, this is fairly straightforward. We need to add a vertex along each edge of the net, then split each triangular face into four faces using the new vertices.

The first step requires us to add new vertices along each edge. There is no fast and simple way to find all the edges unless we store them explicitly. An edge needs to be able to tell us about its end points since we need to use those in the butterfly stencil for computing the new vertex. Furthermore, the stencil extends to the end points' neighbours, so the end point vertices need to know about the edges they are connected to.

The second step, breaking existing faces into new faces, requires that the faces know about their vertices, and such information is already present. The faces also need to know about their edges. While this is possible by looking through the face vertices for all their edges and fishing through those, that would require a fair amount more work for every lookup, and so the edge data are also explicitly stored with each face. This leads to increased computer memory requirement. Our data structure now has arrays of vertices, edges, and faces. Vertices know about their edges, edges know about their vertices, and faces know about their vertices and edges.

The data structure we will be using is based wholly on locality, so that the time it takes to find one vertex given another is proportional to the number of edges between them. The complete subdivision steps are given below.

- Tessellate the surface.
- Create an edge-vertex map so that we can locate new vertices along edges.
- Create an edge-edge map so that we can pair the half-edges made when the edge is split.

- Tessellate the edges.
- Build the new faces.
- Once the subdivision process is complete, generate the normals at the vertices if required.

For the edge subdivision we iterate over the edges. At each edge, check the valences of the end point vertices to determine which subdivision rules to use, then apply the relevant rules, produce the new vertex and add it to the vertices array.

Constructing the new faces is more involved, as it requires more work to ensure that when creating the four new faces their vertices are all arranged in anti-clockwise order and have the correct edges. Each face contains the data for its corner vertices and edges. From the lookup tables created while subdividing edges, we also know the new vertices and new edges.

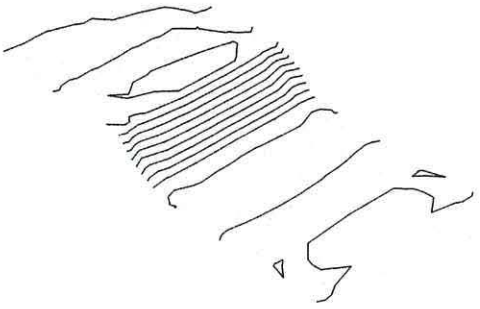
We can now use these steps to subdivide as many times as required. Each subdivision increases the polygon count by a factor of four. Only after we complete the subdivision do we calculate the vertex normals. Iterating over the vertices with the equations for the tangent vectors given previously finds our normals.

The butterfly subdivision method as a basis for contouring has not been utilised in the literature, and so we believe that this forms a contribution to contouring methods – providing an additional method to use as the main contouring algorithm, or an algorithm to use when the preferred method fails.

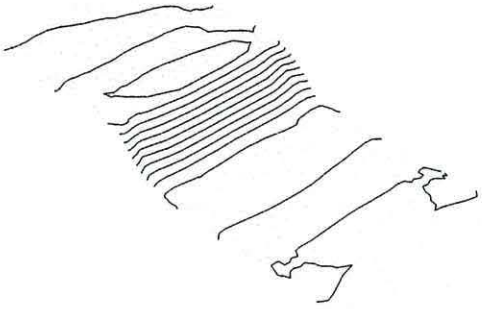
As we have the steps to perform the subdivision, we can produce contour maps of the stratigraphic horizon data sets given in the previous chapter. This is given in Figure 4.7. Larger versions of the contour maps are given in Appendix A.

Table 4.1 shows the run time taken to produce each of the contours given in figure 4.7. Timings were performed on a 2GHz Athlon CPU with 768MB of RAM.

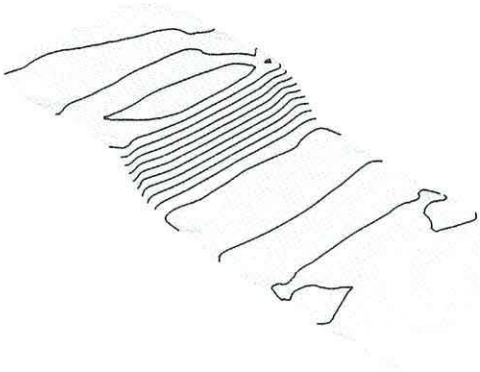
We can see in Figure 4.7 that subdividing the triangulation many times increases the appearance of the smoothness of the contours, although beyond a subdivision level, the extra triangles slow down the algorithm significantly with little improvement over the contour map. The meshing algorithm could be sped up by improved algorithm design, although subdividing beyond the third recursion shows no obvious



(a) First Recursion



(b) Second Recursion



(c) Third Recursion

(d) Fourth Recursion

(e) Fifth Recursion

Figure 4.7: Contour Maps of Maximum Burial Depth Values Over A Stratigraphic Horizon, Produced Over Successive Recursions of the Butterfly Subdivision Algorithm

Table 4.1: Run times of the contouring methods used for Figure 4.7.

Recursion Level	No. of Nodes	Meshing Time	Contouring Time	Total Time
a) First	224	0.241 s	0.047 s	0.288 s
b) Second	835	0.865 s	0.071 s	0.936 s
c) Third	3221	2.148 s	0.156 s	2.304 s
d) Fourth	12649	7.262 s	0.231 s	7.493 s
e) Fifth	50115	18.064 s	0.497 s	18.561 s

improvement in the contouring. Comparing the subdivisions with the WFPS method in the previous chapter we see that recursively subdividing to the third level is usually sufficient to produce a comparable contour map.

In the next chapter we shall examine whether this novel scheme can run at a speed comparable to the other methods in this thesis. For the remainder of this chapter we will discuss further areas of research into the butterfly subdivision scheme.

4.4 Adaptive subdivision

As the subdivision scheme described so far in this chapter subdivides the whole domain, it may be worth considering whether we can adaptively subdivide based on the number of data points in a specific area.

The problem with adaptive solutions for subdivision surfaces is that they do not easily present a closed-form parametrisation. The only easy way to tessellate them is through recursion. As we recurse, we are converging on a limit surface – the same limit surface regardless of which tessellation method used.

If we tessellate adaptively, we have changed the control net. Some of the control net may be at a higher level of tessellation than the rest, and so our net is no longer converging on the same surface so the underlying surface is now fundamentally different.

Furthermore, although this issue could be resolved in some way, the motivation to do so may be lacking. As we have already seen, recursively subdividing to the third level is usually sufficient to produce a comparable contour map. If this is the case when using adaptive subdivision we may often end up with a data set that

has been subdivided to the second or third recursion in some areas, with a third or fourth level of recursion in others – the extra work to adaptively subdivide may have negligible benefits.

When we have data sets with a large concentration of data in some areas more than others, we could start with an *adaptive triangulation* and then use the Butterfly Subdivision scheme on that triangulation. This could be a mid-point between Butterfly Subdivision and full adaptive subdivision.

4.5 Controlling expansion beyond the boundary

We can improve the butterfly subdivision further by controlling the expansion of the nodes beyond the original boundary. Due to the nature of the butterfly subdivision algorithm, the expansion will always occur, and so the simplest method to solve this is to perform a constrained subdivision and project any outlying points back onto the boundary. The projection onto the boundary is as follows.

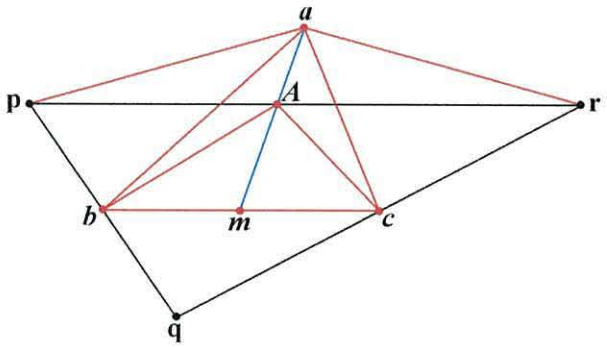


Figure 4.8: Constraining the outlying points to lie on the boundary.

During the subdivision, the triangle **pqr** is subdivided into four triangles, **pba**, **abc**, **qcb** and **rac**, as shown in Figure 4.8. At each boundary edge, point **a** lies beyond the boundary. Denote the (x, y, z) components of point **a** as a_x, a_y and a_z respectively. Let **m** be the mid-point of points **b** and **c**. Put $\mathbf{u} = \mathbf{m} - \mathbf{a}$ and $\mathbf{v} = \mathbf{r} - \mathbf{p}$ so that the equation of the line **am** is $\mathbf{L} = \mathbf{a} + \alpha\mathbf{u}$ and the equation of the line **pr** is $L = \mathbf{p} + \beta\mathbf{v}$ for some values $\alpha, \beta \in \mathbb{R}$. We are considering the triangle in two dimensions and so we project onto the x, y-plane and solve for the intersection

of the lines **am** and **pr**. For **am**,

$$x = a_x + \alpha u_x \text{ and } y = a_y + \alpha u_y.$$

For **pr**,

$$x = p_x + \alpha v_x \text{ and } y = p_y + \alpha v_y.$$

Where the lines intersect we have two simultaneous equations

$$a_x + \alpha u_x = p_x + \beta v_x$$
$$a_y + \alpha u_y = p_x + \beta v_y$$

which can be rearranged to

$$u_x\alpha + v_x\beta = p_x + a_x$$
$$u_y\alpha + v_y\beta = p_y + a_y.$$

These have solutions

$$\alpha = (-v_x(p_y - a_y) + v_y(p_x - a_x)) \,/\, (-u_x v_y + u_y v_x)$$
$$\beta = (\, u_x(p_y - a_y) + u_y(p_x - a_x)) \,/\, (-u_x v_y + u_y v_x)$$

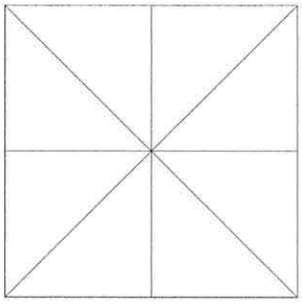
This gives the required point A, where

$$\mathbf{A} = \mathbf{a} + \alpha\mathbf{u} \qquad \text{or } \mathbf{A} = \mathbf{p} + \beta\mathbf{v}.$$

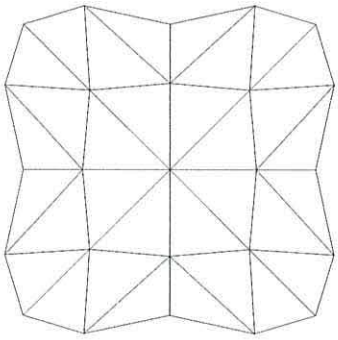
We then replace **a** by **A** in the subdivision, which gives a subdivision that lies within the x, y projection of the original subdivision.

This is shown in Figure 4.9, where we start off with a 9-point data set and subdivide whilst constraining the subdivision.

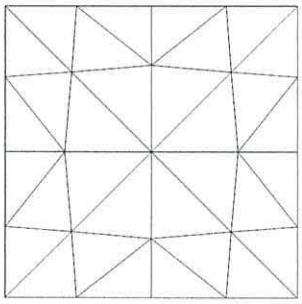
Applying the constrained butterfly subdivision algorithm to our data sets gives us the results shown in Figure 4.10. Larger versions of the contour maps are given in Appendix D. Note that if a boundary triangle changes during the constraining process, then as a result any contour lying on that triangle also changes. This may be undesirable to the user, who can easily revert to unconstrained butterfly subdivision if required.



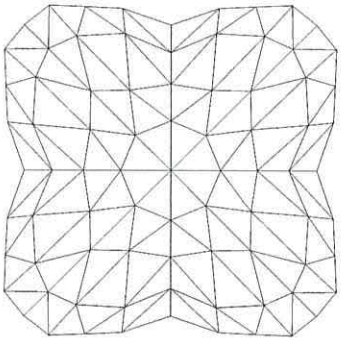
(a) Original mesh of the nine-point data set



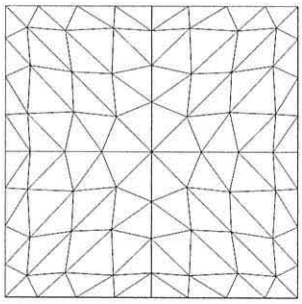
(b) First level of subdivision, unconstrained



(c) First level of subdivision, constrained

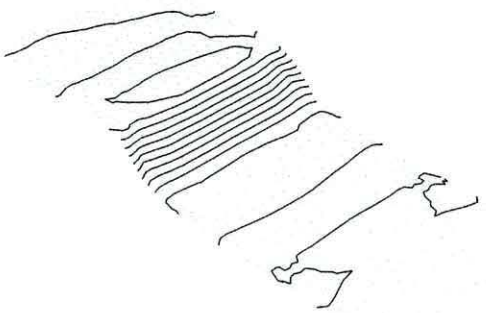


(d) Second level of subdivision, unconstrained

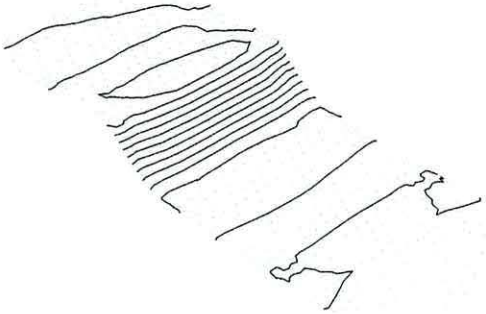


(e) Second level of subdivision, constrained

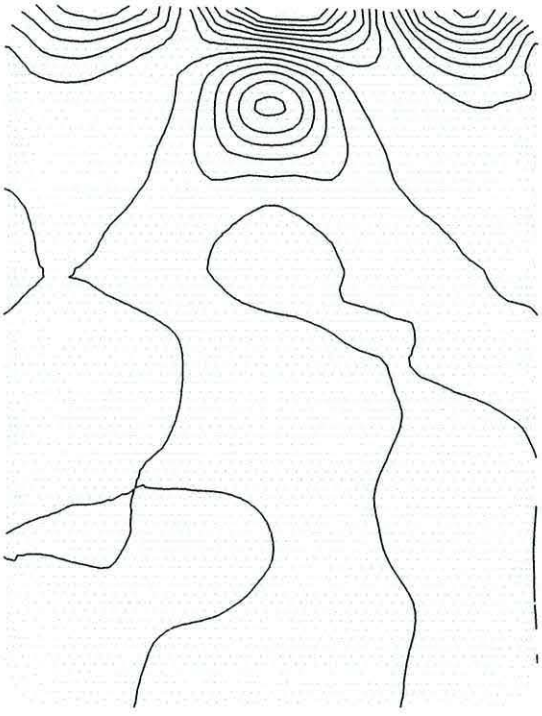
Figure 4.9: Constraining a nine-point data set when subdividing.



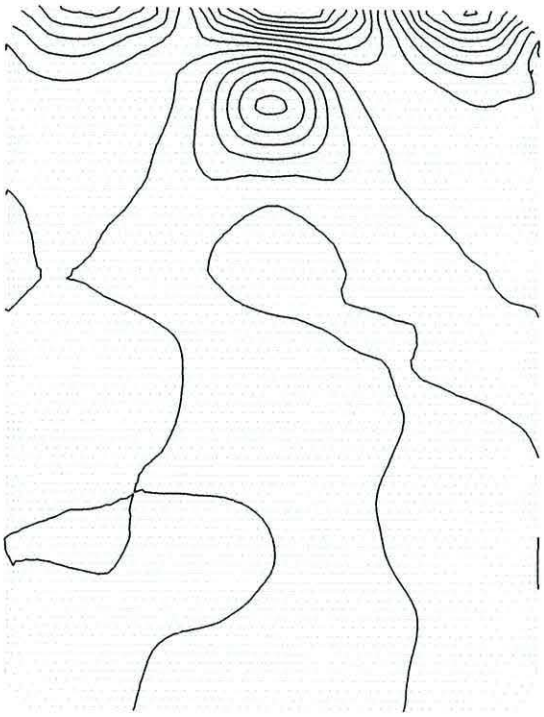
(a) Stratigraphic Horizon, unconstrained



(b) Stratigraphic Horizon, constrained



(c) Surface With Minor Perturbations, unconstrained



(d) Surface With Minor Perturbations, constrained

Figure 4.10: Second level of constrained subdivision for our two data sets.

A constrained butterfly subdivision is preferable to an unconstrained butterfly subdivision if we wished to retain the original domain, especially if the domain forms part of a larger data set, where expansion beyond the boundary would cause an overlap of data. The main cost for this advantage is the computational time taken to find the nodes which lie outside the boundary, and the algorithm used to constrain them.

For the Stratigraphic Horizon data set, the run times for the five steps of both unconstrained and constrained butterfly subdivision schemes are given in Table 4.2. We can see from the table that the current constraining algorithm increases the

Table 4.2: Run times of the contouring methods used for the Stratigraphic Horizon data set.

Unconstrained Butterfly Subdivision

Recursion Level	No. of Nodes	Meshing Time	Contouring Time	Total Time
a) First	224	0.241 s	0.047 s	0.288 s
b) Second	835	0.865 s	0.071 s	0.936 s
c) Third	3221	2.148 s	0.156 s	2.304 s
d) Fourth	12649	7.262 s	0.231 s	7.493 s
e) Fifth	50115	18.064 s	0.497 s	18.561 s

Constrained Butterfly Subdivision

Recursion Level	No. of Nodes	Meshing Time	Constraining Time	Contouring Time	Total
a) First	224	0.241 s	0.368 s	0.047 s	0.656 s
b) Second	835	0.865 s	1.422 s	0.071 s	2.358 s
c) Third	3221	2.148 s	3.269 s	0.156 s	5.573 s
d) Fourth	12649	7.262 s	11.070 s	0.231 s	18.563 s
e) Fifth	50115	18.064 s	28.518 s	0.497 s	47.079 s

computational time significantly – more than doubling the total run time at each stage. This is mainly due to the constraining algorithm searching through every edge and node to see whether the nodes could potentially lie outside the boundary. Once these nodes are determined, the algorithm can then constrain them to the boundary. It is possible that during the subdivision process, the boundary nodes could be stored as a separate list of nodes, so that the constraining algorithm could check just these nodes and constrain them when necessary. This would significantly reduce the computational time and enable the constrained butterfly subdivision to be a viable candidate for contouring.

4.6 Summary

In this chapter we have investigated a selection of subdivision schemes, concentrating on the butterfly subdivision scheme. The original subdivision scheme devised by Dyn et al [18] was restricted to a finite number of triangulations and so we discussed Dyn et al's extended butterfly scheme, and then Zorin et al's modified butterfly scheme. This enabled us to obtain C^1 continuous surfaces, including at the boundary. Once we had C^1 continuous surfaces, we used the butterfly subdivision method to produce a contour map of our data sets. This has not previously been done in the available literature, and so this thesis gave a new method for contouring data.

When we compared the contour maps of the third and fourth recursions of the stratigraphic horizon data (Figure 4.7), we only noticed a minor difference in the contours, and comparing the fourth and fifth recursion contour maps we noticed that there is no real difference in output. For general use, the third recursion is usually sufficient to produce a smooth contour, with the fourth recursion being used when we observe any remaining jagged contours.

We have seen that the butterfly subdivision in its original form is different to the other contouring methods as the boundary expands beyond the original domain, whilst still respecting the original nodes, including those at the boundary. This can be seen as either an advantage, or a disadvantage, depending on the results required. The expansion beyond the boundary enables us to estimate the nature of the domain, and hence the contour, beyond the boundary. This can be particularly advantageous if the domain contains missing data and holes, as the expansion into these holes makes it more straightforward to estimate the values of the missing data. The problem with this expansion is that if we wished to retain the original domain we would need to perform some trimming or even retriangulation near the boundary. This is especially important when the domain forms part of a larger data set, where expanding beyond the boundary would cause an overlap of data.

Constraining the boundary removes this problem, although the computational time taken to find and move the nodes which lie beyond the boundary is significantly higher – especially when performing a constrained butterfly subdivision over many

recursions.

The current constraining algorithm increases the computational time significantly – more than doubling the total run time at each stage. It is possible to significantly reduce the computational time during the subdivision process, by storing the boundary nodes as a separate list of nodes, so that the constraining algorithm could check just these nodes and constrain them when necessary. This would enable the constrained butterfly subdivision to be a viable candidate for contouring. However, the current data structures do not allow for this scenario, and so to improve on algorithm efficiency a new data structure would need to be devised.

In addition to constraining the boundary, we could also adaptively subdivide based on areas of interest. The problem with adaptive solutions for subdivision surfaces is that, unlike patches such as Bézier patches, subdivision surfaces do not easily provide a closed-form parametrisation. The only easy way to tessellate them is through recursion. We rely on the fact that as we recursively subdivide, we are converging on a limit surface. Regardless of how we tessellate, we should be converging on the same limit surface.

If we were to adaptively tessellate, we have changed the control net. Some of the net might be at a higher level of tessellation than the rest, which means that our net is no longer converging to the same surface.

Now that we have studied some ways of subdividing and contouring data, we can compare the results of each method with the methods discussed in previous chapters. We shall see this in the following chapter.

Chapter 5

Comparing the contouring methods

5.1 The Amlin Contouring Algorithm

In this thesis we have used the data structures in Walker's TetSim program [44] to produce contour outputs via the Amlin contouring program, devised as part of this thesis. The relevant parts of TetSim are described in the following section, along with the additional structures required by Amlin.

The Amlin data structure

In the TetSim program, calculations are completed over *TIN*s – Triangulated Irregular Networks. A TIN model represents a surface as a set of contiguous, non-overlapping triangles. Within each triangle the surface is represented by a plane.

For the purposes of contouring, TetSim contains an additional data structure known as a *ContourMap*. This is an unused part of TetSim, required by Amlin, in order to calculate and produce contours over a surface. The ContourMap class copies the data from the TINs in the data ready for contouring. In addition, the ContourMap class contains the options for each contour map, such as which method to use and which attributes to assign, as well as a class for the Powell-Sabin Subdivision (*PSSubdivision*) and Butterfly Subdivision (*ButterflySurface*). The ContourMap class also contains the list of normals used to calculate the contours, as well as the algorithms used in the contouring.

Each TIN stores the list of triangles, nodes and edges, and the maximum and

minimum values of all nodes within the TIN. If required, each PSSubdivision and ButterflySurface stores its TIN inside the class, which is then a separate TIN stored in the ContourMap. Each of the subdivision classes contain the algorithm required to perform the relevant subdivision, and each ButterflySurface contains its own unique control net.

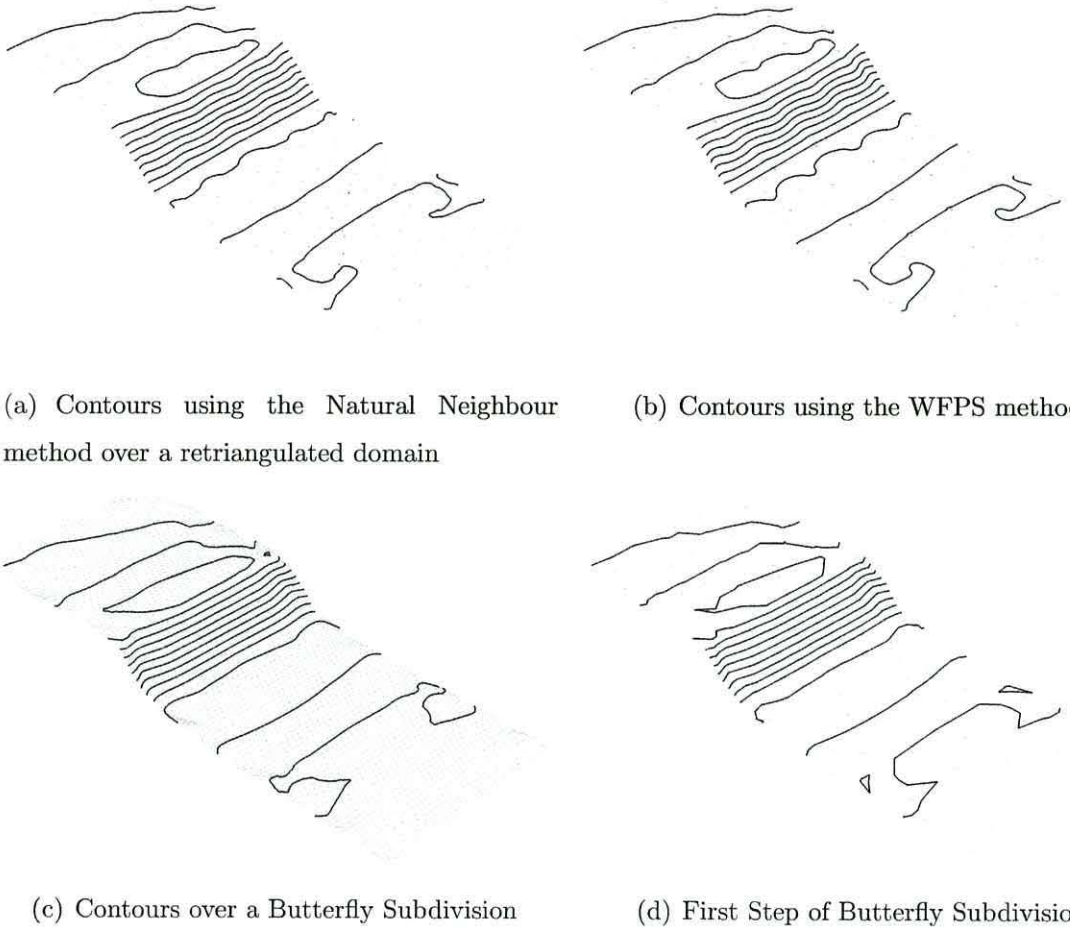
If we were to produce a contour output for a Butterfly Subdivided contour map, the program would complete the following steps:

1. Read in data and contour options.
2. Generate TIN (using TetSim algorithms) and store in ContourMap.
3. For each level of the Butterfly subdivision:
   (a) Subdivide the surface using the TIN stored in ContourMap, as described in Section 4.3.
   (b) Create an edge-vertex map so that we can locate new vertices along edges.
   (c) Create an edge-edge map so that we can pair the half-edges made when the edge is split.
   (d) Tessellate the edges.
   (e) Build the new faces.
   (f) Once the subdivision process is complete, generate the normals at the vertices if required.
   (g) *If required*, constrain the output to the original surface.
   (h) Replace the original TIN stored in ContourMap with this new triangulation.
4. Calculate the contours over the surface and output to a file.

As we can see from the list above, there is some inefficiency in the programming, which may result in the algorithms running slower than optimal. We shall see this effect more clearly in the following section.

5.2 Results

For our given data sets, we have contour maps using the natural neighbour method, the Worsey-Farin method over a Powell-Sabin subdivision (WFPS) and straight-line contours over a butterfly subdivision. Larger versions of the contour maps are given in the appendices. The various contour maps for the stratigraphic horizon data are given in Figure 5.1, and the run times of the methods are given in Table 5.1.



(a) Contours using the Natural Neighbour method over a retriangulated domain

(b) Contours using the WFPS method

(c) Contours over a Butterfly Subdivision

(d) First Step of Butterfly Subdivision

Figure 5.1: Comparison of the Butterfly Subdivision with the WFPS and Natural Neighbour contouring methods

Clearly there are differences between the contour diagrams, especially in the looped contour in the upper half of the diagram and on the sides of the lower half.

In the lower half of the figures, the algorithms come to differing conclusions as to how the contours should be joined. This is made clearer in the first step of the butterfly subdivision (Figure 5.1(d)), where the algorithm produces two loops. In

Table 5.1: Run times of the contouring methods used for Figure 5.1.

Contouring Method	No. of Nodes	Meshing Time	Contouring Time	Total Time
a) Natural Neighbour	305	0.601 s	0.291 s	0.892 s
b) WFPS	329	0.617 s	0.306 s	0.923 s
c) Unconstrained Butterfly, Third Step	3221	2.148 s	0.156 s	2.304 s
d) Unconstrained Butterfly, First Step	224	0.241 s	0.047 s	0.288 s

producing a smoother contouring, WFPS decides that these are separate contour lines to the main contour, whereas recursive butterfly subdivisions include them with the main contour.

The looped contours produced by the different contouring methods again are not identical, due to the locations and values of the added vertices. The differing algorithms introduce new vertices by estimating or interpolating the intermediate values, and the contour is drawn considering these vertex values. As contours are subject to interpretation, we can not surmise that any of the contours are incorrect. For this particular data, however, the WFPS or Natural Neighbour contour maps may be preferred as they may appear to be more aesthetically pleasing. In contrast to this, the wavy contour at the centre of the surface may not appear to be as realistic as the equivalent contour over the butterfly-subdivided surface. This is due to the values of the vertices on the original triangulation oscillating between different values. The butterfly subdivision smooths this oscillation, whereas the WFPS algorithm reproduces it. Both methods produce areas which are arguably 'more realistic' and areas which are 'less realistic' than hand-contoured data, and so preference to either of the algorithms is subject to the user's prior knowledge.

Comparing the timings of the methods, clearly contours over the first step of the unconstrained butterfly subdivision run faster, although we can see from Figure 5.1 this contouring has jagged edges and sharp corners. After the third subdivision, the contours are smoother and sharp corners are minimised. The cost of this smoothing is the time taken to run the algorithm – almost two and a half times longer than the WFPS method. This is due to the increased number of nodes created by the algorithm. If we were to compare the numbers of nodes, WFPS takes almost a second to run through and calculate 329 nodes whereas the third step of the butterfly subdivision calculates ten times as many nodes in 2.3 seconds. After this example

it is clear that we would have to balance between smooth, aesthetically pleasing contours and a contouring algorithm that runs in the shortest time. Before we compare the run times of the algorithms any further, we must next investigate how the algorithms deal with missing data.

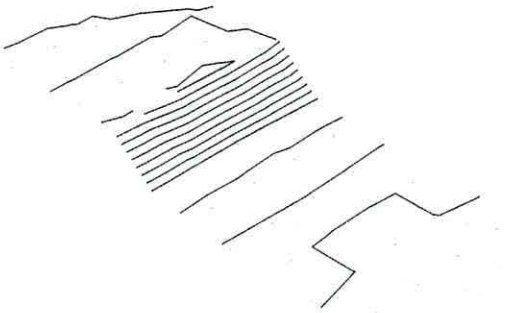
If we were to receive a data set which contains missing data such as holes. or areas where data should have been collected but instead the data was missing or had to be rejected (for example, where the measuring instrument failed and produced clearly anomalous data), the algorithms are able to cope with this and the respective contours are given in Figure 5.2. We can see that the hole appears to be smaller in the butterfly subdivision, and when we overlay the original triangle on the subdivision (Figure 5.2(d)), we see that this is, in fact, true. This is due to the smoothing nature of the subdivision process, as described previously. This could either be used as an advantage where we can estimate contours over missing data and effectively 'fill in the gaps', or we could trim the contours where they no longer lie inside the original domain and hence produce a completed contour map over the original triangulation.

The surface with minor perturbations dataset, shown in Figure 5.3, highlights the difference between the coarseness of the mesh, along with the differences between straight line and Bézier curved contours. The run times of the methods are given in Table 5.2.

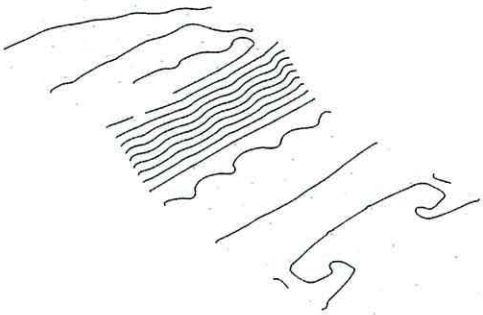
Table 5.2: Run times of the contouring methods used for Figure 5.3.

Contouring Method	No. of Nodes	Meshing Time	Contouring Time	Total Time
a) Natural Neighbour	1589	2.894 s	0.385 s	3.279 s
b) Straight Line over PS subdivision	1673	3.017 s	0.981 s	3.998 s
c) WFPS	1673	3.017 s	0.726 s	3.743 s
d) Unconstrained Butterfly, First Step	1137	1.089 s	0.203 s	1.292 s
e) Unconstrained Butterfly, Third Step	4417	9.476 s	0.510 s	9.986 s

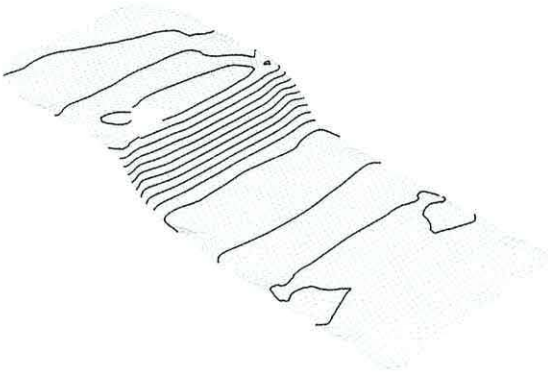
In Figure 5.3(a), we can clearly see that straight-line contours over a triangulation which has not been subdivided produces the least realistic contours, as this contains many jagged contours which often taper off to a point. Doing a simple subdivision, such as Powell-Sabin in Figure 5.3(b), improves the straight-line contours somewhat, with a reduced number of contours with sharp corners.



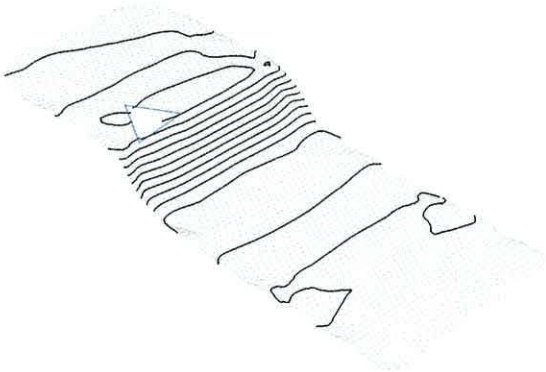
(a) Natural Neighbour Contours over the original triangulation



(b) Contours using the WFPS method

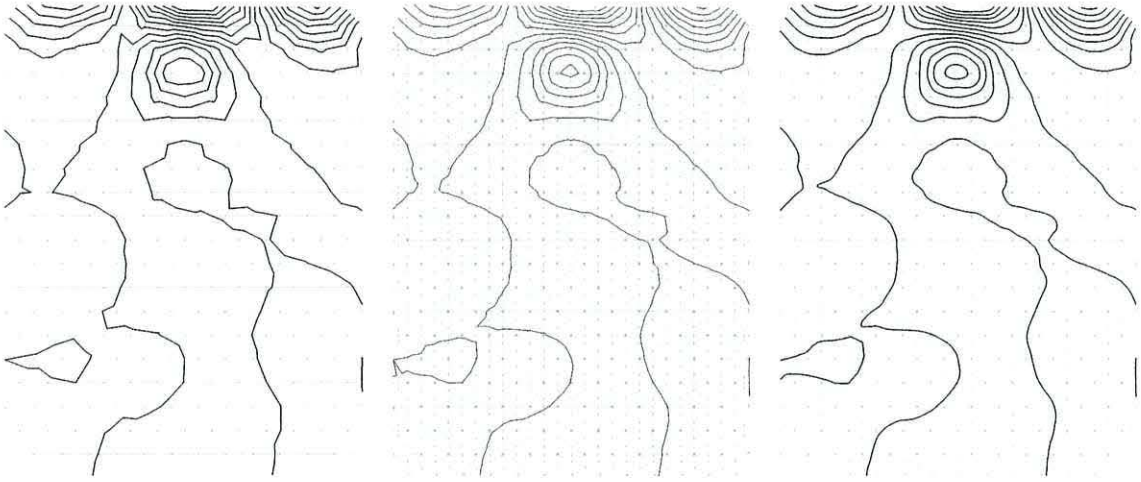


(c) Contours over a Butterfly Subdivision



(d) Contours over a Butterfly Subdivision, with the original hole overlaid

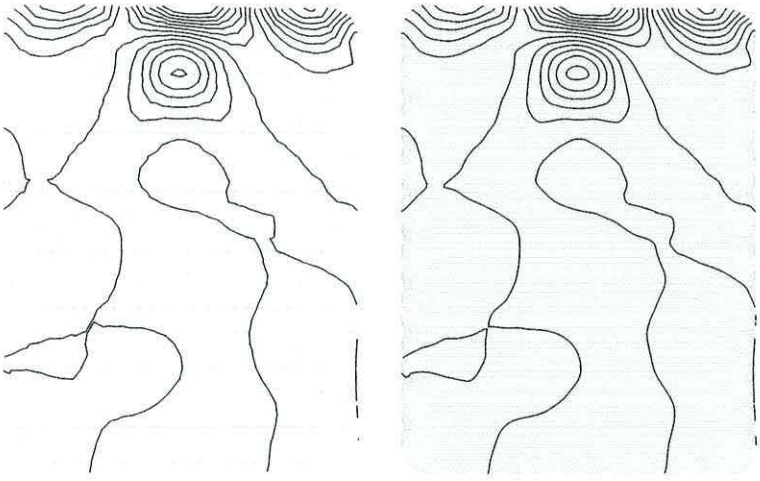
Figure 5.2: Contour Maps of Altitude Values Over A Stratigraphic Horizon, Produced Over Successive Recursions of the Butterfly Subdivision Algorithm



(a) Contours using the Natural Neighbour method over a retriangulated domain

(b) Straight Line Contours over a Powell-Sabin Triangle Subdivision

(c) Contours using the WFPS method



(d) Contours over the first level of Butterfly Subdivision

(e) Contours over the third level of Butterfly Subdivision

Figure 5.3: Comparison of the Butterfly Subdivision with the WFPS and Natural Neighbour contouring methods

Performing the Worsey-Farin algorithm over this triangle subdivision smooths the contours, producing a satisfactory contour map of the data. When using a butterfly subdivision, however, the first level of subdivision produces contours which differ to the other contouring methods (Figure 5.3(d)). This is particularly true in the lower-left area of the diagram where originally we have two contours which pass close to each other, but by the third level of subdivision we see that the connection between these two contours changes, producing different contour lines (Figure 5.3(e)). Comparing the final contour map with the WFPS contours we can see that both contours are similar, and so both of these methods produce comparable outputs.

Comparing the timings of the methods, again contours over the first step of the unconstrained butterfly subdivision run faster, although we can see from Figure 5.3 this contouring still has jagged edges and sharp corners. As the data sets get more complex and the number of nodes increase, we can see that the first stage of the butterfly subdivision algorithm outperforms the other algorithms when it comes to speed. After the third subdivision, however, the contours are again smoother and sharp corners are minimised, although again there is extra run time for the third subdivision.

In addition to these sample data sets, we can also compare the contouring methods to a hand-drawn contour map. The data set is from Boomer data taken in the Irish Sea. There are two sets of data which have been contoured—the depth of the seabed below datum sea-level and the depth to first reflector, i.e. the first discontinuity in wave velocity (after the sea bed). The method of data collection involved taking measurements from the rear of a boat travelling on the surface of the water. The boat travelled in a series of almost parallel lines, resulting in many data points parallel to the boat, but few perpendicular to the direction of travel. This is common in geological modelling and repeats the importance of interpolation between data points.

The WFPS and straight-line contours over a butterfly subdivision are in Figure 5.4, and the hand-drawn contours are in Figure 5.5. Comparing the contour methods with the hand-drawn contours, we see that both the butterfly subdivision and WFPS

methods produce contour maps which are visually similar to each other, and each have 11 contour levels. The hand-drawn contour maps only have 6 contour levels and so it is more difficult to directly compare with the computerised contour maps, although we can see that the contour maps are comparable, particularly the contours for the depth to first reflector. The contours for the depth of the seabed below datum sea-level, however, differs in a few places. On the hand-drawn contour map we have points in a region which have similar values, apart from one point lying within this region which has a value significantly different to the rest. This is not unusual with real data, and so it is up to the user to decide whether this is significant or erroneous data with respect to the area where the data was collected. When using the data to produce computerised contours, it was impractical to use the whole data set. As the data are arranged primarily in rows, the resulting triangulations would consist mainly of thin triangles with very little change in data values from one triangle to the next. Node reduction was performed on the data, which removed a high proportion of the nodes that provided no extra information, as well as some of the 'erroneous' nodes. This is reflected in the contour maps as some of the contours drawn in the hand-drawn contour maps are not present in the computerised contours. Both computerised contours contain extra contours on the left-hand side of the diagram. These are represented on the hand-drawn contours by isolated contour sections, and as we do not have the data between these sections, either contour is acceptable. From the data sets we have investigated, there is evidence to show that both WFPS and butterfly subdivision-based contouring algorithms produce smoother, improved contours compared to straight-line contours on the original domain. From the Boomer data we can see that these contouring methods are comparable to hand-drawn contours, producing similar, but not identical results. It is almost impossible for the contouring methods to produce results which are identical to hand-drawn contours, as often the person who is producing the hand-drawn contours has prior knowledge of the area where the data were collected, and may interpret the data differently to the computer methods.

When comparing the timings of the algorithms, again butterfly subdivision takes around two and a half times longer to run than WFPS.

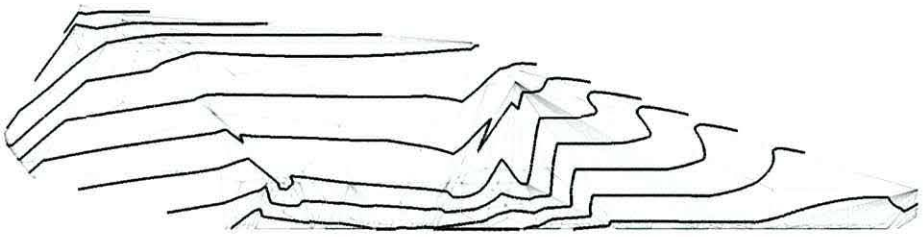
Table 5.3: Run times of the contouring methods used for Figure 5.4.

Contouring Method	No. of Nodes	Meshing Time	Contouring Time	Total Time
a) WFPS, depth of seabed	371	0.705 s	0.349 s	1.054 s
b) WFPS, depth to first reflector	371	0.705 s	0.337 s	1.042 s
c) Butterfly, depth of seabed	3801	2.443 s	0.181 s	2.624 s
d) Butterfly, depth to first reflector	3801	2.443 s	0.175 s	2.618 s

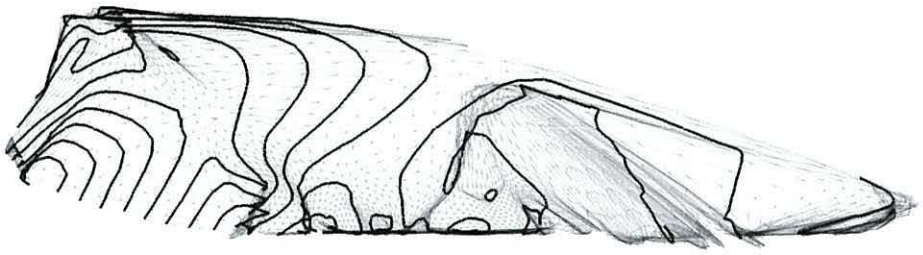
In all of the examples given, butterfly subdivision contouring methods takes a significantly longer than the other methods which have been investigated. However, if we were to consider the first level of recursion the algorithm in fact runs significantly faster, at a cost of smoothness of contours. This supports the hypothesis which claims that the new butterfly algorithm runs faster than existing contouring algorithms. If we are concerned about producing smooth contours, then we must consider further recursions of the butterfly subdivision algorithm. In its current state, further recursions are time-consuming, meaning we can no longer claim that the algorithm is faster than other methods.

The meshing for butterfly subdivision appears to be the cause of this extra time, and so it would be helpful to investigate the code behind the meshing algorithm to see whether any improvements could be made. We will comment on this further in the next chapter.

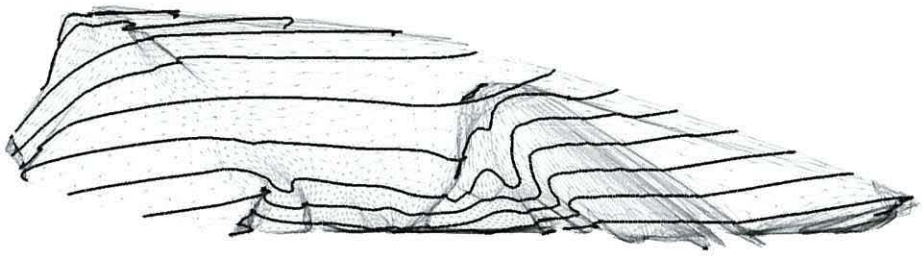
(a) WFPS contour map for the depth of the seabed



(b) WFPS contour map for the depth to first reflector

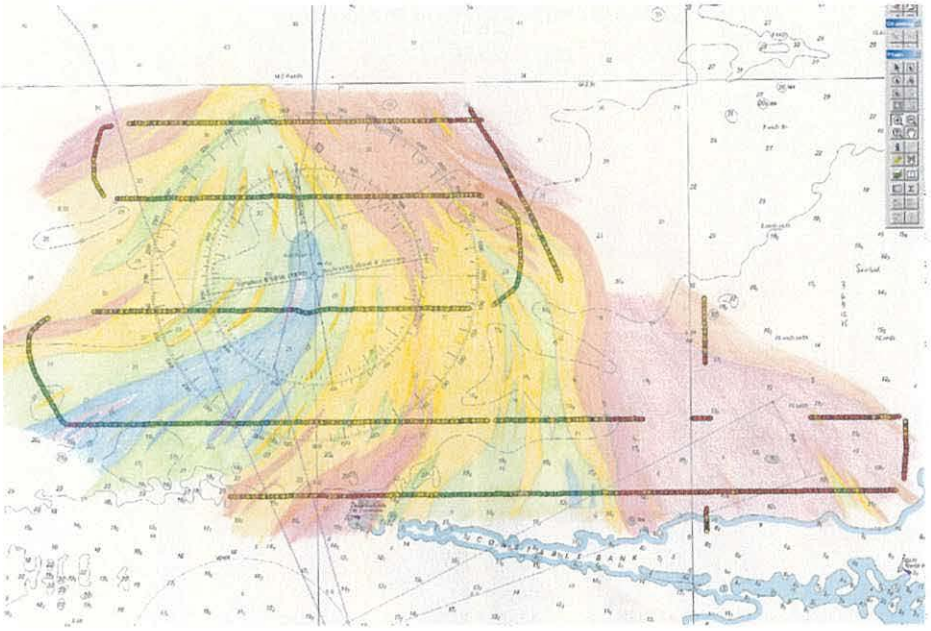


(c) Butterfly subdivision contour map for the depth of the seabed

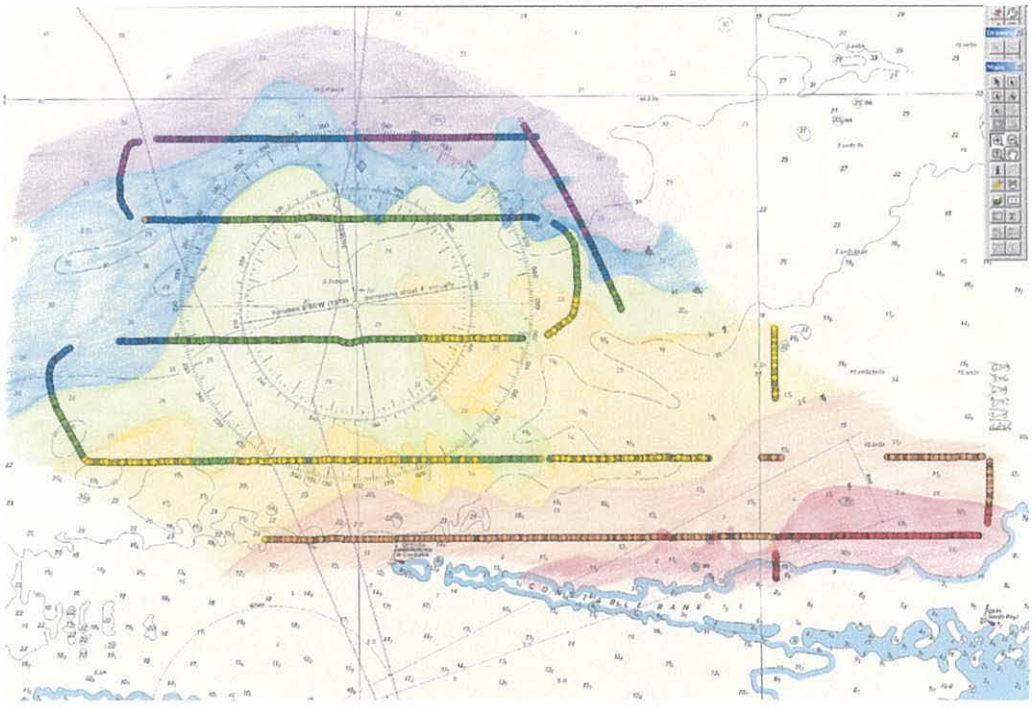


(d) Butterfly subdivision contour map for the depth to first reflector

Figure 5.4: WFPS and Butterfly subdivision contour maps of the Boomer data



(a) Depth of the seabed



(b) Depth to first reflector

Figure 5.5: Hand-drawn contour maps of the Boomer data

Chapter 6

Conclusions

6.1 Summary

In this thesis, we have analysed various methods that are used to produce contour maps over a given data set. In this chapter, we aim to summarise the achievements of this thesis and highlight how they contribute to the literature on contouring algorithms. We also discuss areas for further research. All contouring methods have some restrictions or undesirable effects, and these have been highlighted in the chapters.

In chapter 2, we reviewed many surface approximation/interpolation methods, such as triangle based interpolation. This led to investigating natural neighbour bases as a new method of interpolation for contouring algorithms. We also looked at a standard method of interpolation – Bézier curves, as well as its generalisations into B-splines and NURBS. As B-splines and NURBS are not required for the contouring algorithms, they are mentioned for completeness and not investigated further. This chapter provided the basis for the methods discussed in the future chapters.

Straight-line contours are able to contour data sets where other methods may not be able to, although often these contours are jagged and contain sharp corners. Straight-line contours using the natural neighbour algorithm improve on straight-line contours over a standard triangulation, but can often still appear jagged. If the mesh is retriangulated with a finer triangulation, the natural neighbour algorithm produces smoother contours. Although the natural neighbour algorithm acts locally,

resulting in a faster retriangulation, the triangulation process itself must produce a Delaunay triangulation for the natural neighbour algorithm to work. This means that any subdivision will not be sufficient for a contour map, and often data which has been provided already with a triangulation would either need a check to ensure that the triangulation is Delaunay, or would need to be retriangulated. This could be seen as an inefficient method since it may discard information which can be useful, particularly the connectivity between nodes.

In chapter 3 we began to examine contouring algorithms, starting with the Worsey-Farin contouring algorithm. This algorithm produces a C^1 continuous interpolation if the input triangulation has been subdivided. For simplicity, we chose the Powell-Sabin six triangle subdivision, using the incentre as the interior point. The choice of incentre guaranteed that the interior point would indeed lie inside the interior of each triangle, which was not always the case when using the circumcentre. We then gave a definition of Bézier ordinates used for the Worsey-Farin algorithm.

We then investigated the nature of the connection between boundary points on each triangle, giving a sketch proof of Lemma 3.1 from Worsey and Farin [49]. In order to make the algorithm more readable the two cases of connecting the boundary points are reproduced in a different format to the original, as well as reproducing the orignal lemmas required for the algorithm.

The Worsey-Farin algorithm has been reproduced, including the correction by Bloomquist [4], later in the chapter. The original lemmas and full algorithm is provided for completeness and for the main reason that the contours produced by the Worsey-Farin algorithm are the major source of comparison to the butterfly subdivision.

Due to the nature of the Worsey-Farin algorithm, some contouring combinations cannot be drawn. Some examples of these are given in Figure 6.1. If these cases do occur, the data can be subdivided again so that the contours lie in different triangles, although it is difficult to detect when we would need to do this.

At the end of chapter 3 we compared the contours given by three different methods, and we saw that the contours produced using the Worsey-Farin algorithm were smooth. This was a significant improvement over both the straight line

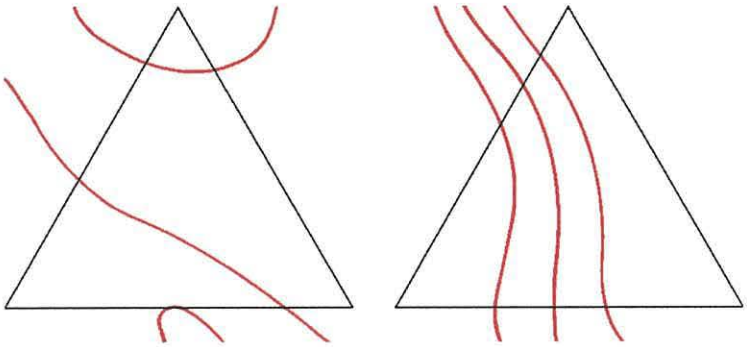


Figure 6.1: Contours not possible using WFPS.

contours and contours using the Natural Neighbour algorithm, although once the domain was retriangulated to produce triangles of a similar size to that of the Powell-Sabin subdivision, the Natural Neighbour contours were almost identical. This showed that the Natural Neighbour algorithm can produce similar results for similar data sets.

Chapter 4 concentrated on investigating the butterfly subdivision method. The original subdivision scheme devised by Dyn et al [18] was restricted to a finite number of triangulations and so we investigated Dyn et al's extended butterfly scheme, and then Zorin et al's modified butterfly scheme. This enabled us to obtain C^1 continuous surfaces, including at the boundary. Once we had C^1 continuous surfaces, we used the butterfly subdivision method to produce a contour map of our data sets. This has not previously been done in the available literature, and so this thesis gave a different method for contouring data.

When we compared the contour maps of the third and fourth recursions of the stratigraphic horizon data (Figure 6.2), we only noticed a minor difference in the contours. Comparing the fourth and fifth recursion contour maps we noticed that there is no real difference in output. For general use, the third recursion is usually sufficient to produce a smooth contour, with the fourth recursion being used when we observe any remaining jagged contours.

The butterfly subdivision in its original form is different to the other contouring methods as the boundary expands beyond the original domain, whilst still respecting the original nodes, including those at the boundary. This can be seen as either an advantage, or a disadvantage, depending on the results required. The expansion

beyond the boundary enables us to estimate the nature of the domain, and hence the contour, beyond the boundary. This can be particularly advantageous if the domain contains missing data and holes, as the expansion into these holes makes it more straightforward to estimate the values of the missing data. The problem with this expansion is that if we wished to retain the original domain we would need to perform some trimming or even retriangulation near the boundary. This is especially important when the domain forms part of a larger data set, where expanding beyond the boundary would cause an overlap of data.

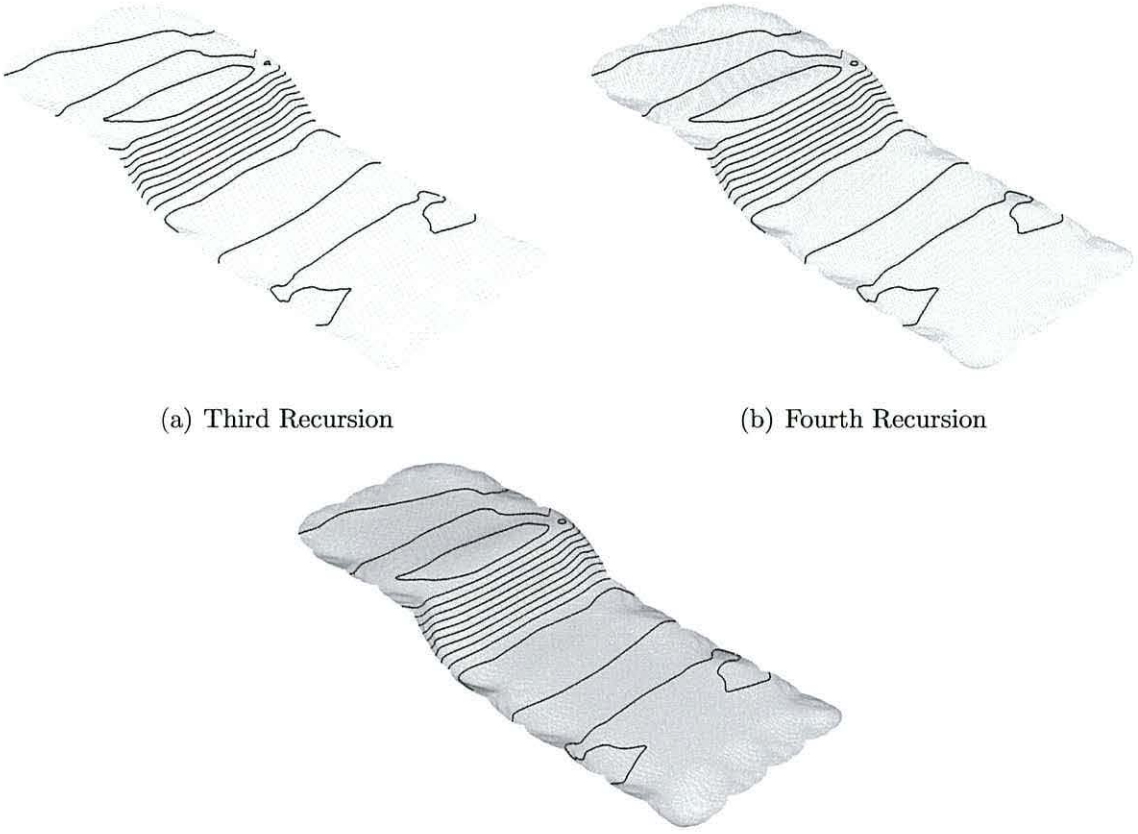
Constraining the boundary removes this problem, although the computational time taken to find and move the nodes which lie beyond the boundary is significantly higher – especially when performing a constrained butterfly subdivision over many recursions.

The current constraining algorithm increases the computational time significantly – more than doubling the total run time at each stage. This is mainly due to the constraining algorithm searching through every edge and node to see whether the nodes could potentially lie outside the boundary. Once these nodes are determined, the algorithm can then constrain them to the boundary. It is possible that during the subdivision process, the boundary nodes could be stored as a separate list of nodes, so that the constraining algorithm could check just these nodes and constrain them when necessary. This would significantly reduce the computational time and enable the constrained butterfly subdivision to be a viable candidate for contouring. However, the current data structures do not allow for this scenario, and so to improve on algorithm efficiency a new data structure would need to be devised.

In addition to constraining the boundary, we could also adaptively subdivide based on areas of interest. The problem with adaptive solutions for subdivision surfaces is that, unlike patches such as Bézier patches, subdivision surfaces do not easily provide a closed-form parametrisation. The only easy way to tessellate them is through recursion. We rely on the fact that as we recursively subdivide, we are converging on a limit surface. Regardless of how we tessellate, we should be converging on the same limit surface.

If we were to adaptively tessellate, we have changed the control net. Some of

the net might be at a higher level of tessellation than the rest, which means that our net is no longer converging to the same surface. This is a worst-case scenario for scalable geometry as it produces errors which cannot be simply avoided.



(a) Third Recursion

(b) Fourth Recursion

(c) Fifth Recursion

Figure 6.2: Contour Maps of Maximum Burial Depth Values Over A Stratigraphic Horizon, Produced Over Successive Recursions of the Butterfly Subdivision Algorithm

The main aim of this thesis was to prove that our new butterfly subdivision contouring algorithm will run faster than existing contouring methods, whilst also being able to contour difficult areas such as at discontinuities.

We have seen in the five examples of this thesis that on the same number of nodes the Butterfly subdivision algorithm is faster than natural neighbour contours, straight line contours over a Powell-Sabin subdivision and the WFPS contouring algorithm, proving the hypothesis to some extent. However, the drawback is that the contours do not appear to be as smooth as the other methods. In order for the

contours to be as smooth as the other methods, further subdivisions are required and so in its current form this is the time-consuming part of the algorithm. If smoothness is used as a requirement for contouring, then we cannot currently prove the hypothesis to be completely true. This falls short of demonstrating universal improvement for the algorithm, but is consistent with the state aims within the confines of PhD research. However, on this result we may have one (or both) of the following scenarios:

- The current subdivision algorithm is inefficient, with many areas of code which can be streamlined. This could be the main cause of the time-consuming part of the algorithm. Improved algorithm design will decrease the runtime of the program, and may result in completely proving the hypothesis.

- There may not be one algorithm which can produce smooth contours over all a data sets, and run faster than all other contouring algorithms. If this is the case then the original aims may not be realisable, and so the way forward may be to develop fast codes for specific classes of problems, rather than seek one algorithmic solution that excels in all contouring problems.

In addition to the main aims, the following contributions have been made:

- We have investigated natural neighbour bases for interpolation for contouring algorithms. This is generally ignored since triangulations are normally used as the basis. Natural neighbour bases provide a reasonable alternative since natural neighbours arise from the dual of the Delaunay triangulation, and provide comparable results. If the data changes and requires retriangulation in specific regions, the natural neighbour algorithm proves invaluable as a retriangulation is not required – saving computing time over interpolations that require a mesh.

- We also investigated and implemented the butterfly subdivision scheme for contouring algorithms. The butterfly scheme is normally used for surface approximation, but as contours are level sets on a surface it follows that contouring algorithms could be used on a butterfly subdivided surface. These

contours are comparable to other contouring algorithms, and due to the nature of the subdivision, contours over the butterfly scheme enable us to estimate contours beyond the boundary as well as over areas where we have missing data.

- If the expansion beyond the boundaries is not required, we have derived and implemented a novel constrained butterfly subdivision scheme for contouring algorithms. This offers the same flexibility as the standard, unconstrained subdivision scheme, with the additional property that it respects the original boundaries of the input data. We believe that the results obtained in this thesis will be of use to persons wishing to use a contouring algorithm to estimate beyond the boundary, as well as within the original constraints of the boundary. We also offer an alternative method of contouring which can be used when other methods fail. We are confident that both the unconstrained and constrained butterfly subdivisions would cope with general surfaces in three dimensions, although we have not proven that this will be the case for all surfaces.

- The main contouring algorithms mentioned in this thesis, namely contours using natural neighbour bases, the WFPS contouring algorithm, and both the unconstrained and constrained butterfly subdivisions can be found as part of the Amlin contouring program.

6.2 Further work

Following the investigations described in this thesis, a number of projects could be taken up, including the following listed below:

- Adaptive tessellation – the current adaptive tessellation changes the original triangulation, and does not converge to the same surface as recursive subdivision. This is because we have changed the control net. Some of the net might be at a higher level of tessellation than the rest, which means that our net is no longer converging to the same surface. It is likely that an algorithm could be

produced to either counteract the changing of the control net, or to prevent it changing from the outset, although as discussed in this thesis the benefits of adaptive Butterfly Subdivision may outweigh the cost of doing so.

- The Butterfly Subdivision schemes described here have not been used over folded surfaces. It has yet to be proven whether Butterfly Subdivided surfaces can (or indeed cannot) cope with folded surfaces.

- The expansion in the unconstrained Butterfly Subdivision was shown to give a good estimate in the data containing a manually introduced hole. The interesting part would be to investigate whether the Butterfly Subdivision always gave a good estimate beyond the original data boundary, and whether the expansion could be used for data which is significantly further than the original data set.

- The current code for constraining the boundaries of the data set is inefficient as it searches through all edges (internal and external) to see which external edges have been expanded beyond the boundary. This search happens at each stage of the subdivision and so clearly there is scope for increasing the efficiency of the program at this point.

- In its current form, the butterfly subdivided contouring algorithm runs slower than the other algorithms, although this is due to the inefficiency of the computer coding rather than the algorithm itself. This is further slowed when constraining all nodes to the boundary of the data set, as the current coding searches through the complete data rather than just the edges and nodes which could potentially lie beyond the boundary. If this part of the coding was made more efficient the program would run significantly faster – especially on large data sets and fine meshes.

- Even if program efficiency is improved, there may not be one algorithm which can produce smooth contours over all a data sets, and run faster than all other contouring algorithms. If this is the case then the next stage would be

to develop fast codes for specific classes of problems, rather than seek one algorithmic solution that excels in all contouring problems.

Appendix A

A stratigraphic horizon with a discontinuity

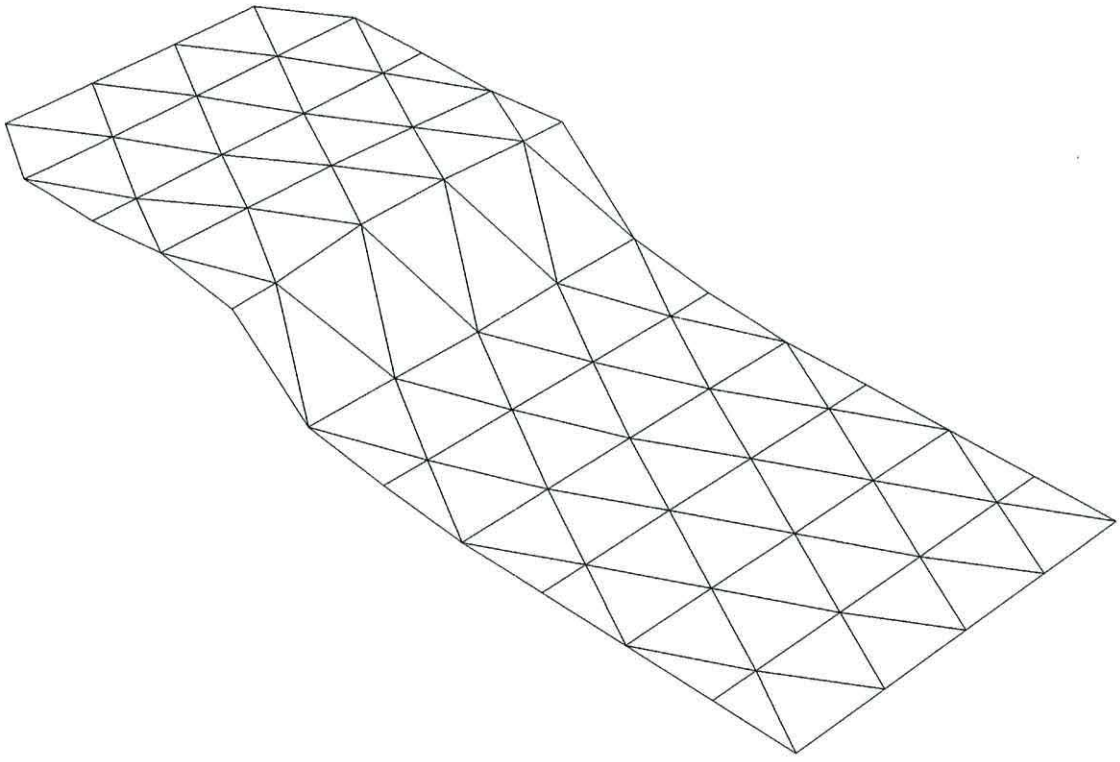


Figure A.1: Original Triangulation

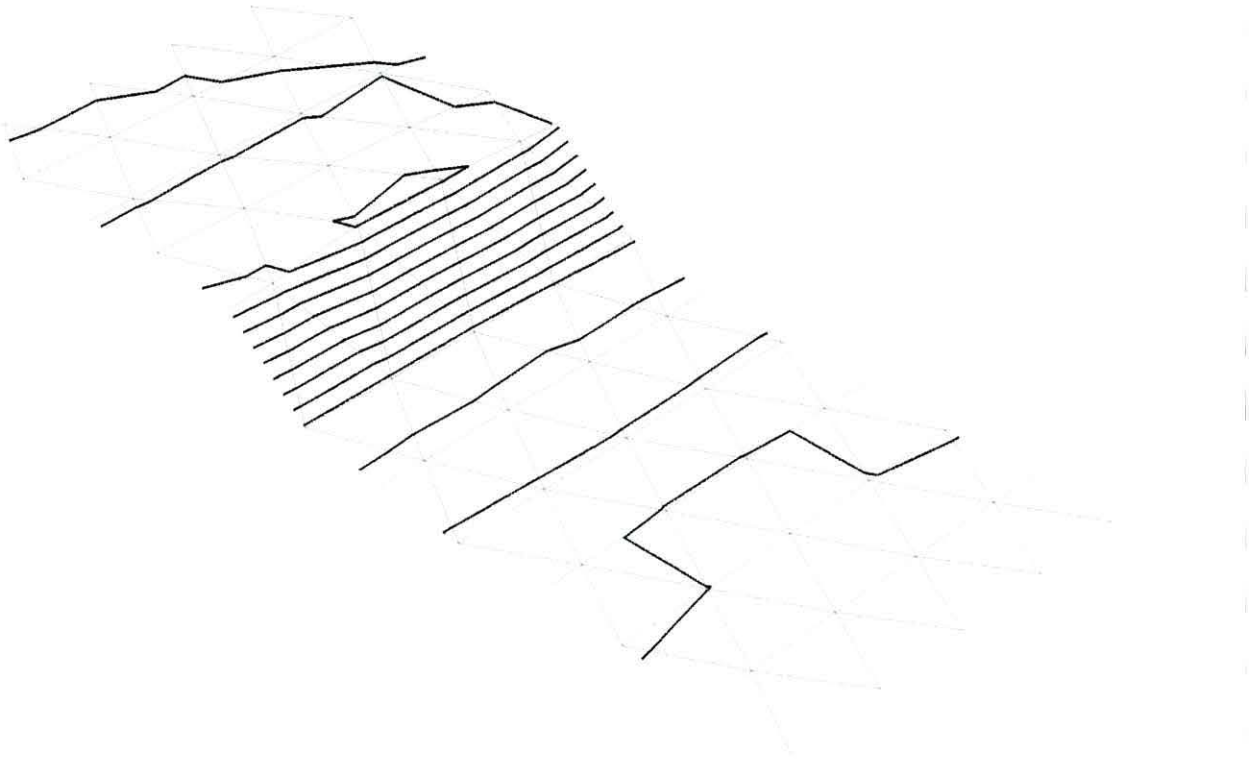


Figure A.2: Contours using the Natural Neighbour method

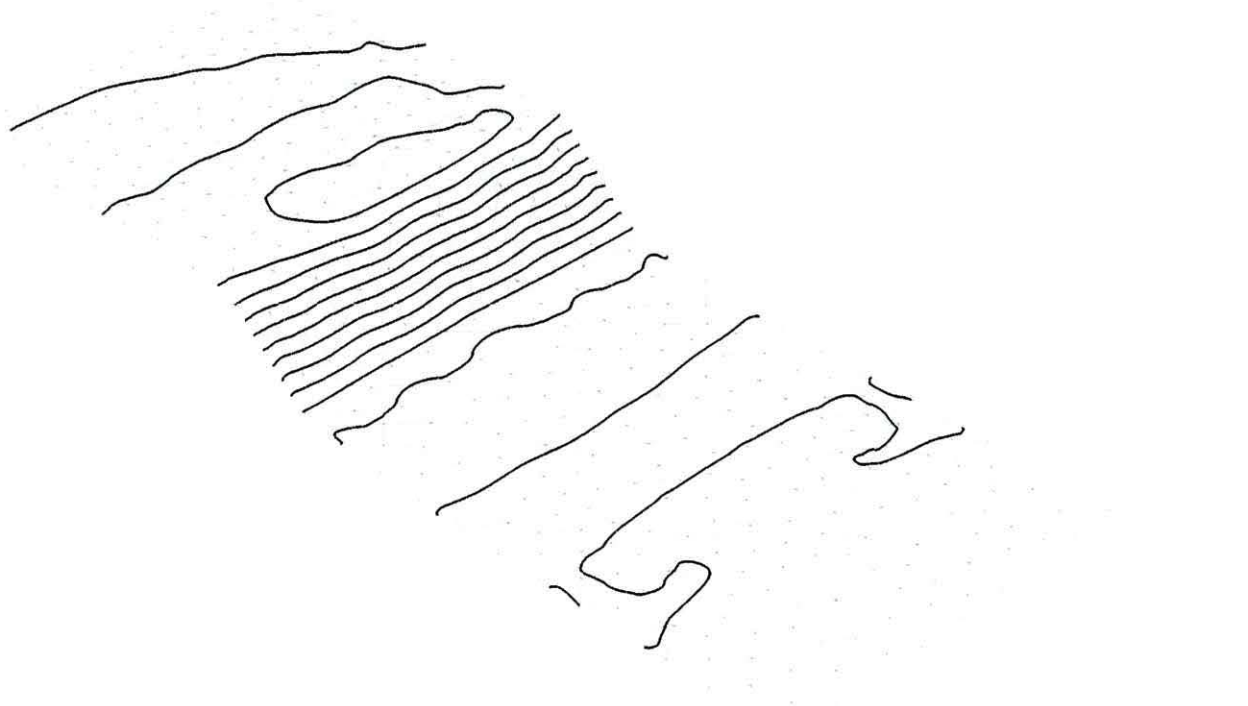


Figure A.3: Contours using the Natural Neighbour method over a retriangulated domain

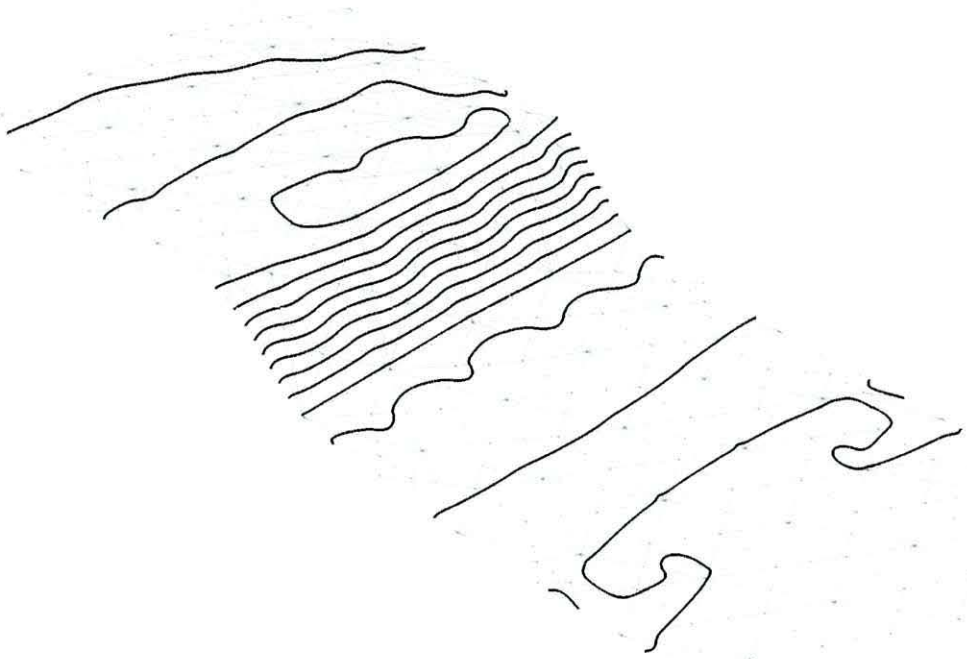


Figure A.4: Contours using the WFPS method

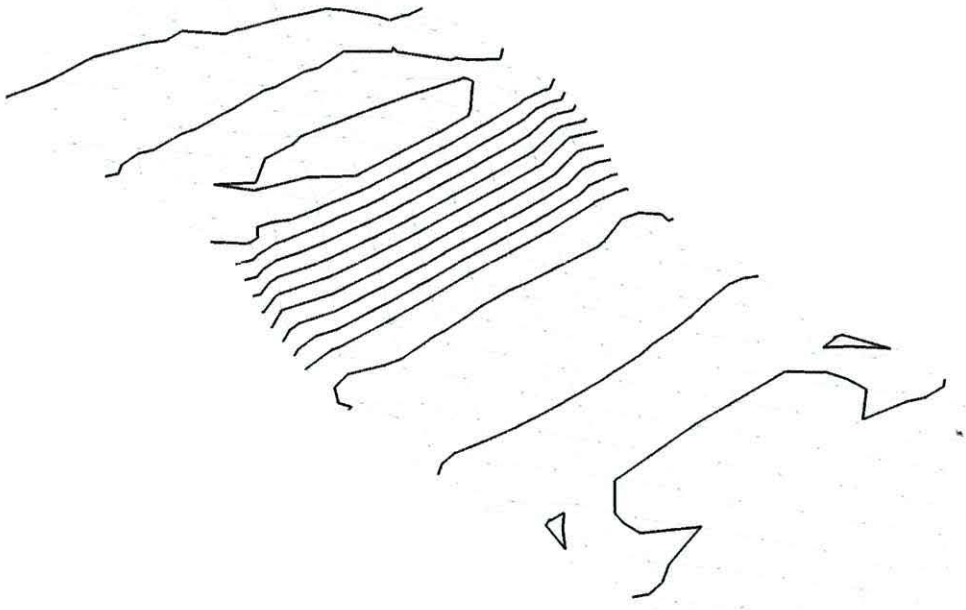


Figure A.5: First Recursion of the Butterfly Subdivision Algorithm

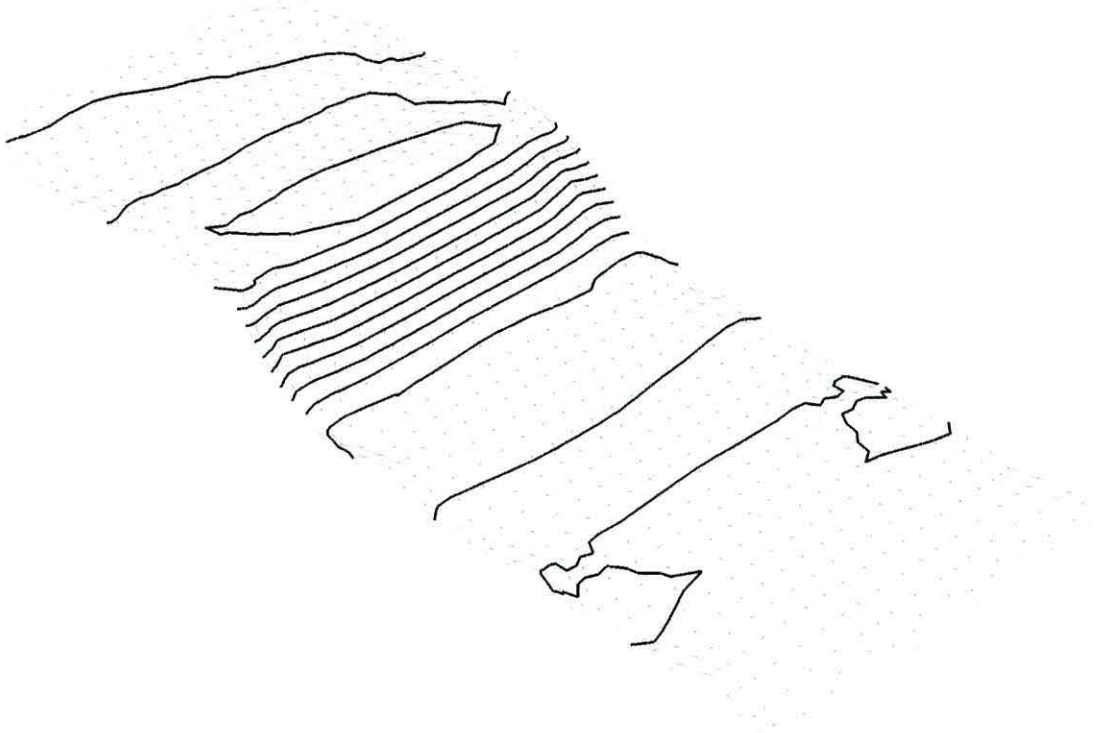


Figure A.6: Second Recursion of the Butterfly Subdivision Algorithm

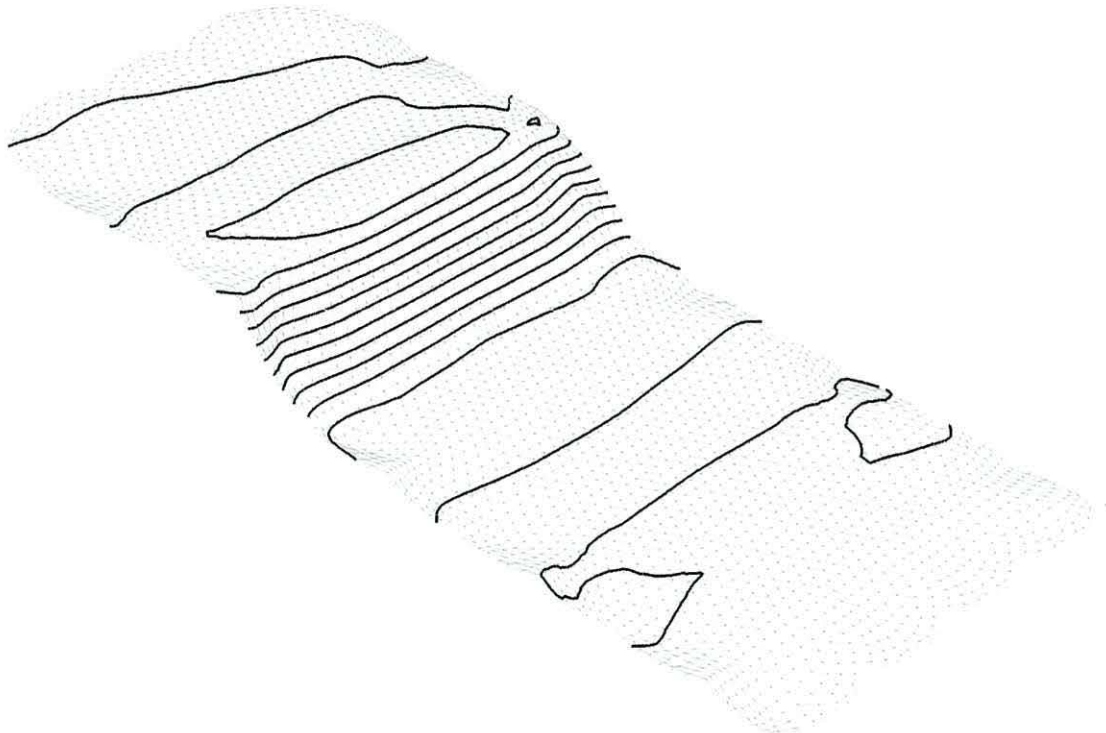


Figure A.7: Third Recursion of the Butterfly Subdivision Algorithm

Figure A.8: Fourth Recursion of the Butterfly Subdivision Algorithm

Figure A.9: Fifth Recursion of the Butterfly Subdivision Algorithm

Appendix B

A surface with minor perturbations

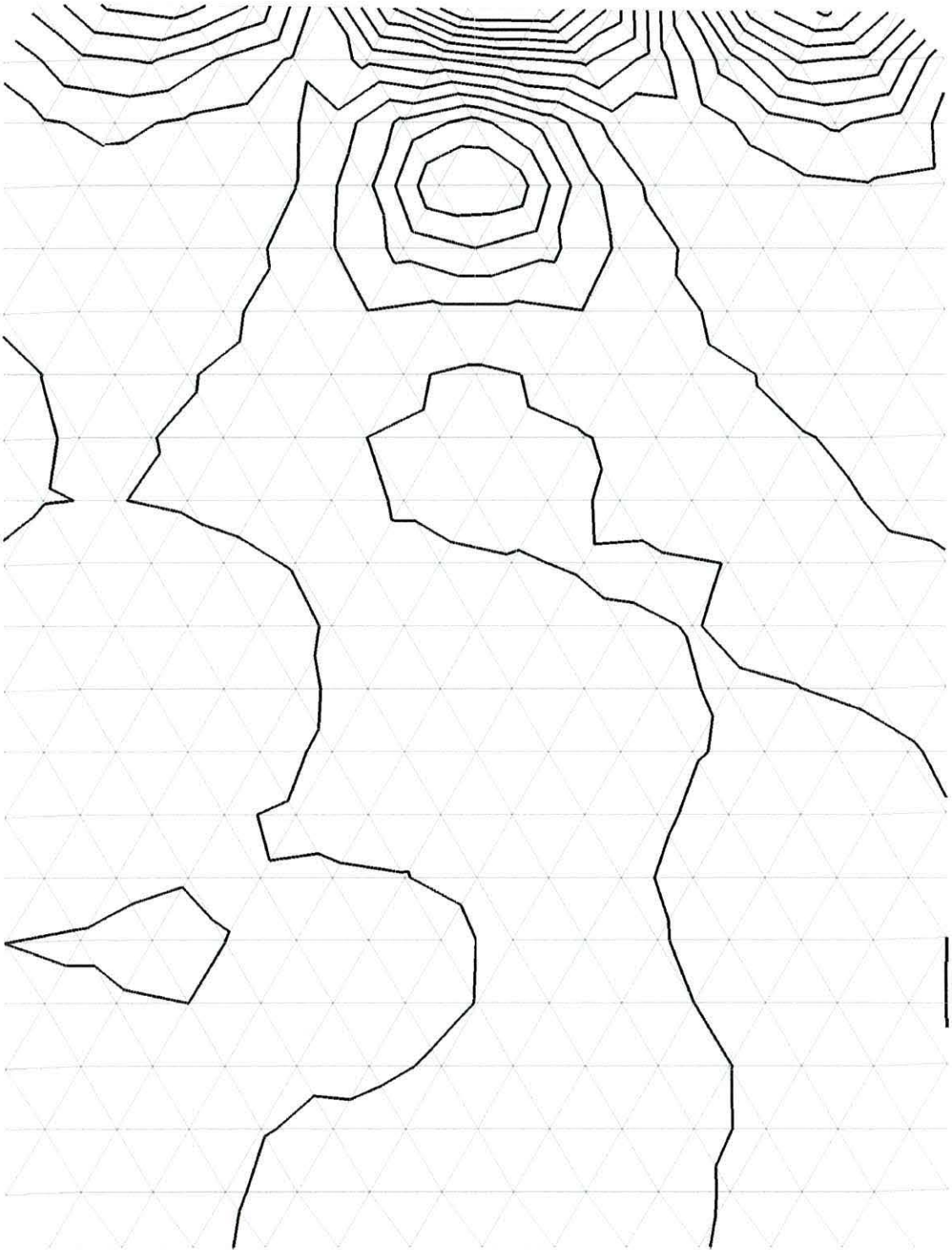


Figure B.1: Linear contours over the original triangulation

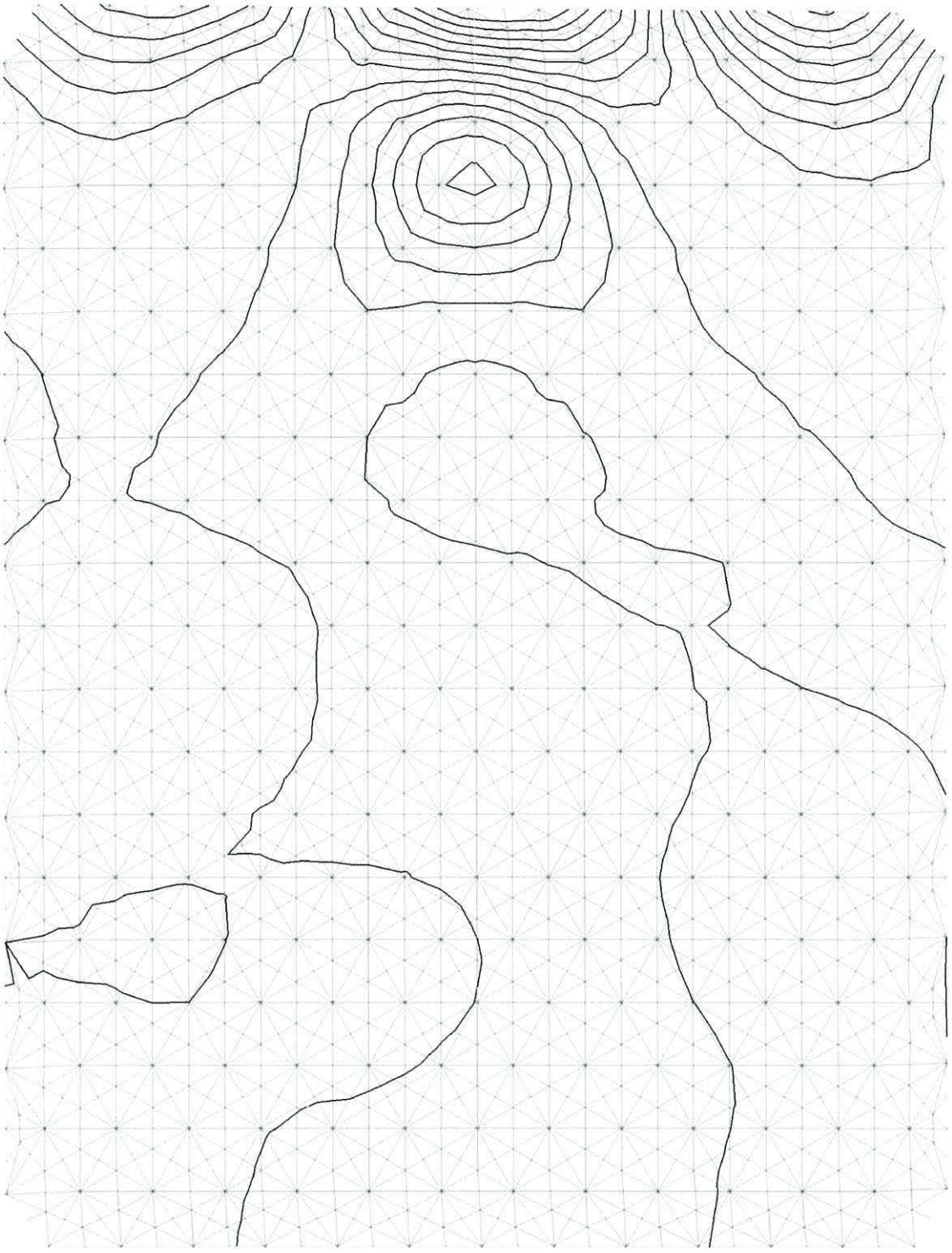


Figure B.2: Straight Line Contours over a Powell-Sabin Triangle Subdivision

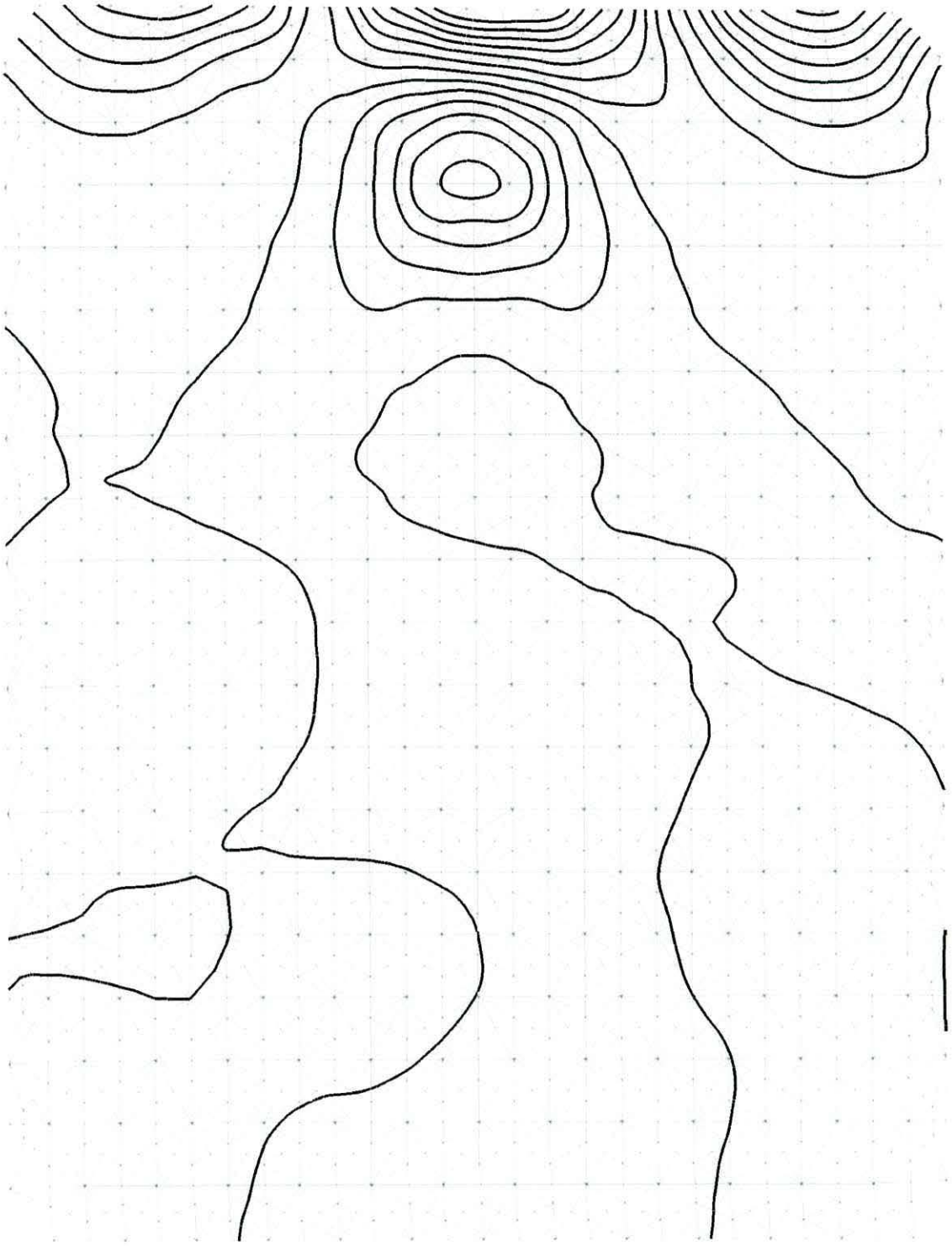


Figure B.3: Contours using the WFPS method

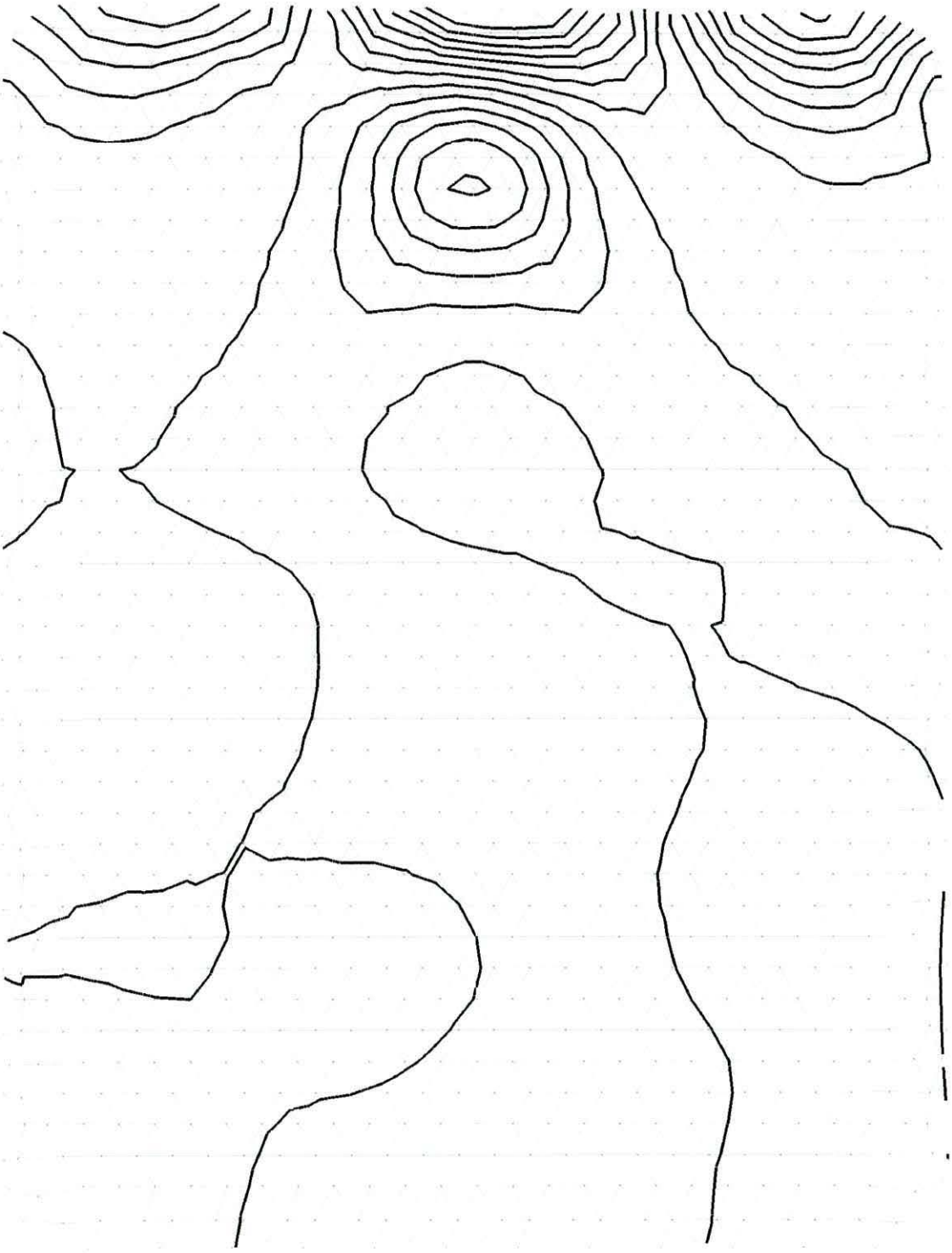


Figure B.4: First Recursion of the Butterfly Subdivision Algorithm

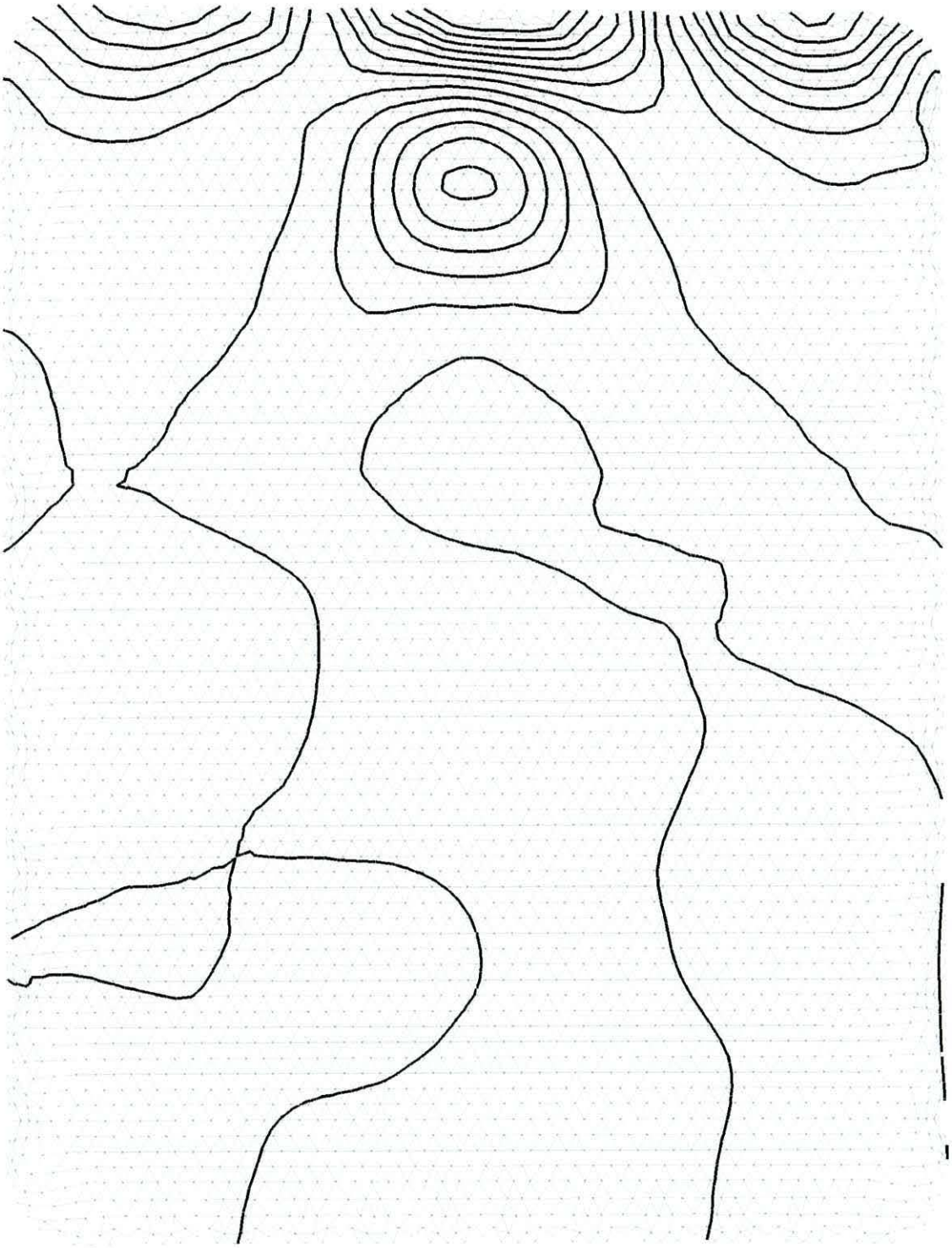


Figure B.5: Second Recursion of the Butterfly Subdivision Algorithm

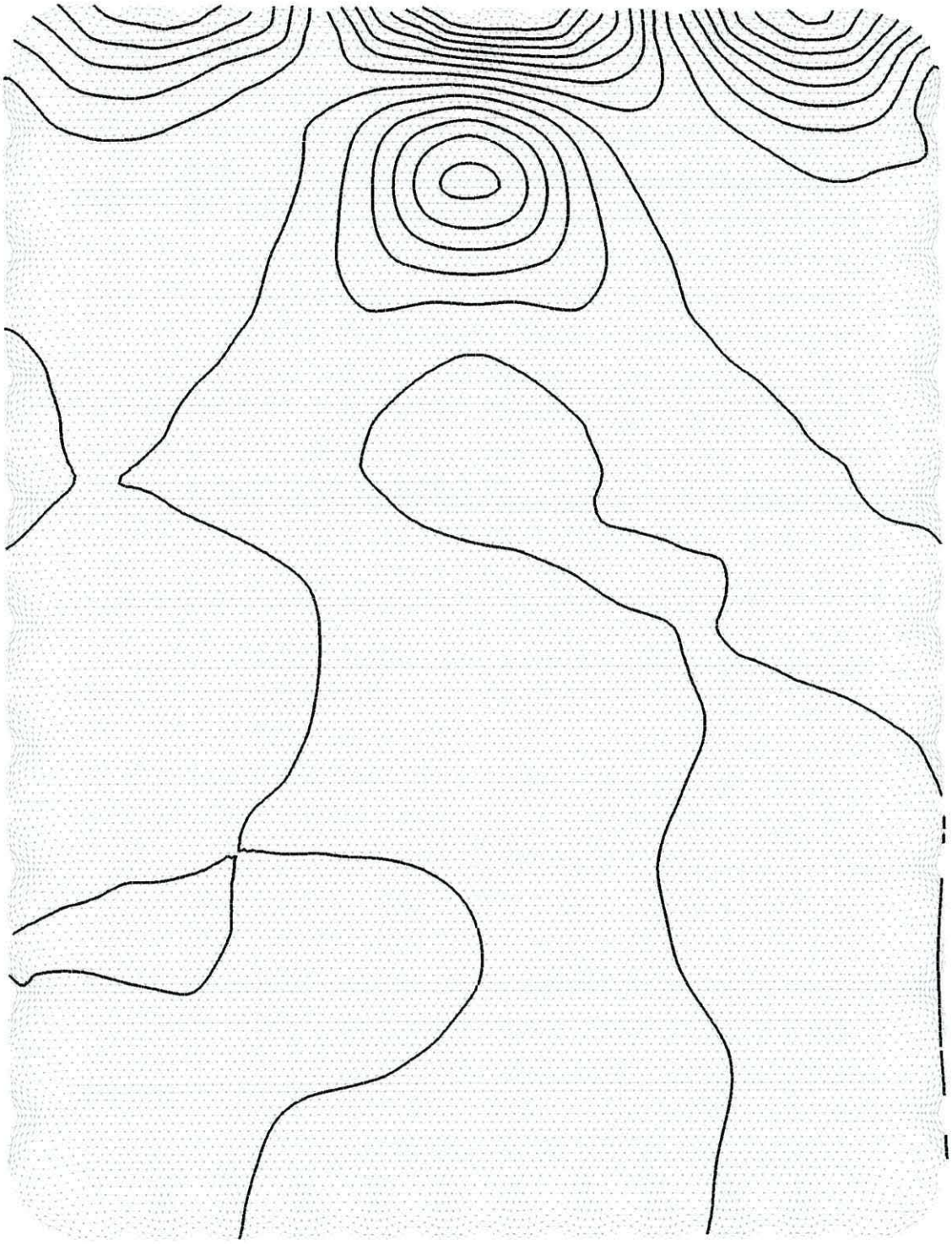


Figure B.6: Third Recursion of the Butterfly Subdivision Algorithm

Appendix C

Constrained Butterfly Subdivision

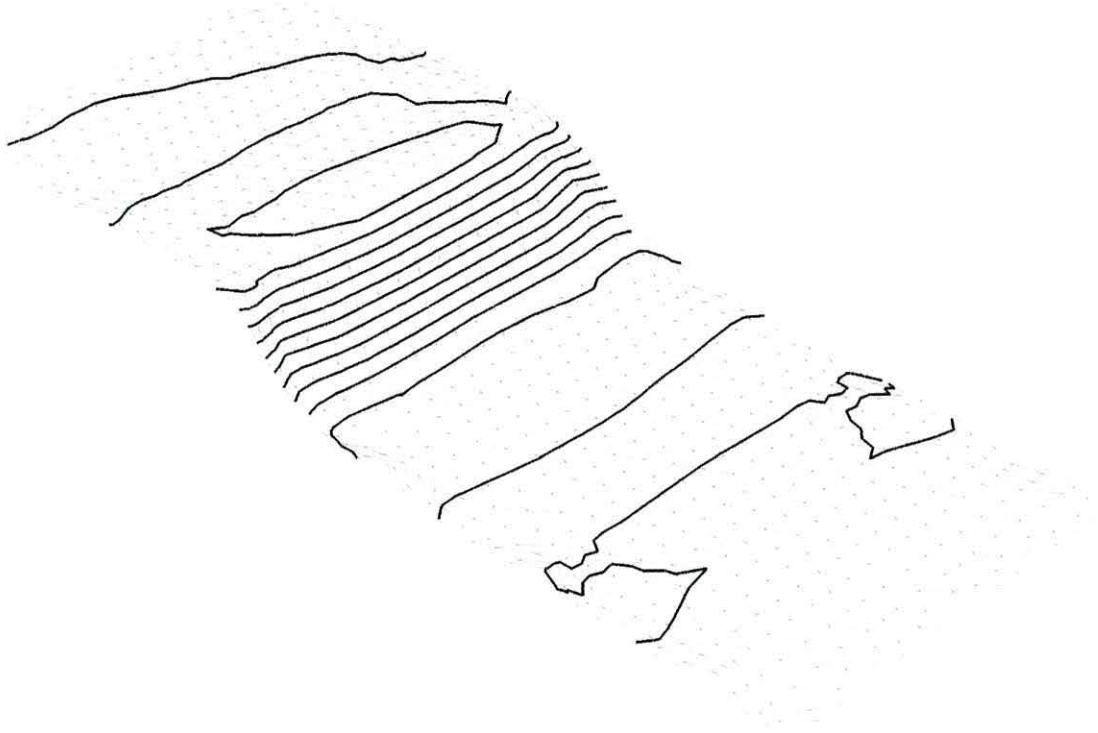


Figure C.1: Unconstrained Second Recursion of the Butterfly Subdivision Algorithm for a Stratigraphic Horizon with a discontinuity

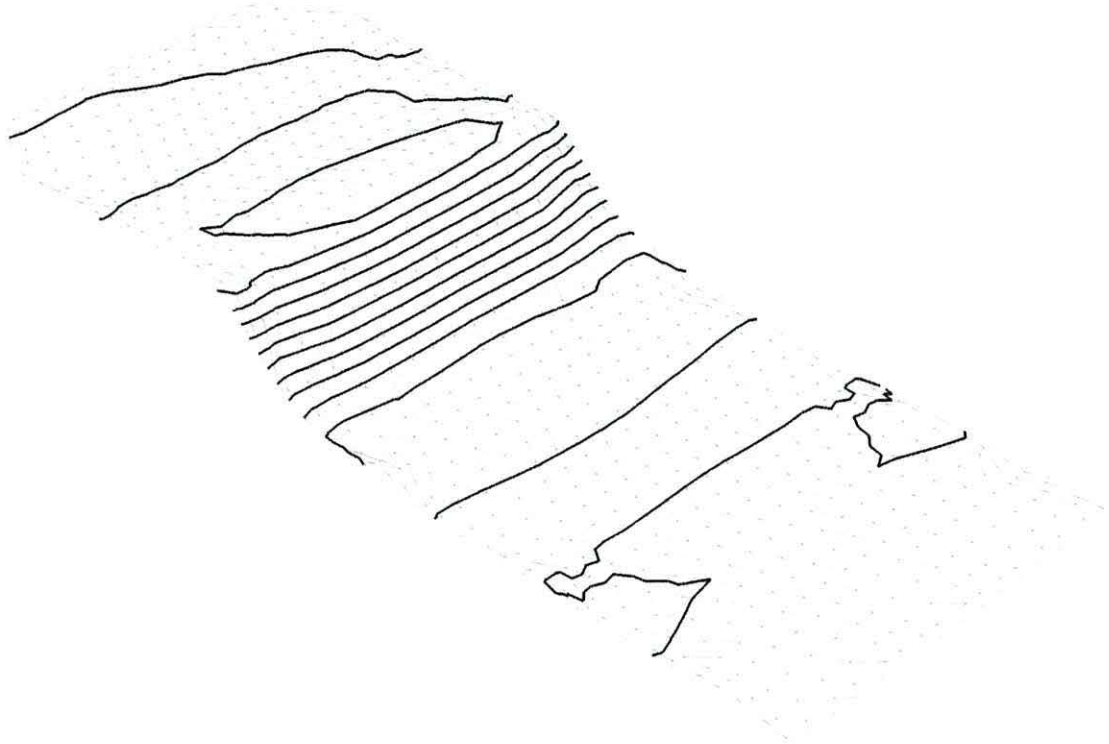


Figure C.2: Constrained Second Recursion of the Butterfly Subdivision Algorithm for a Stratigraphic Horizon with a discontinuity

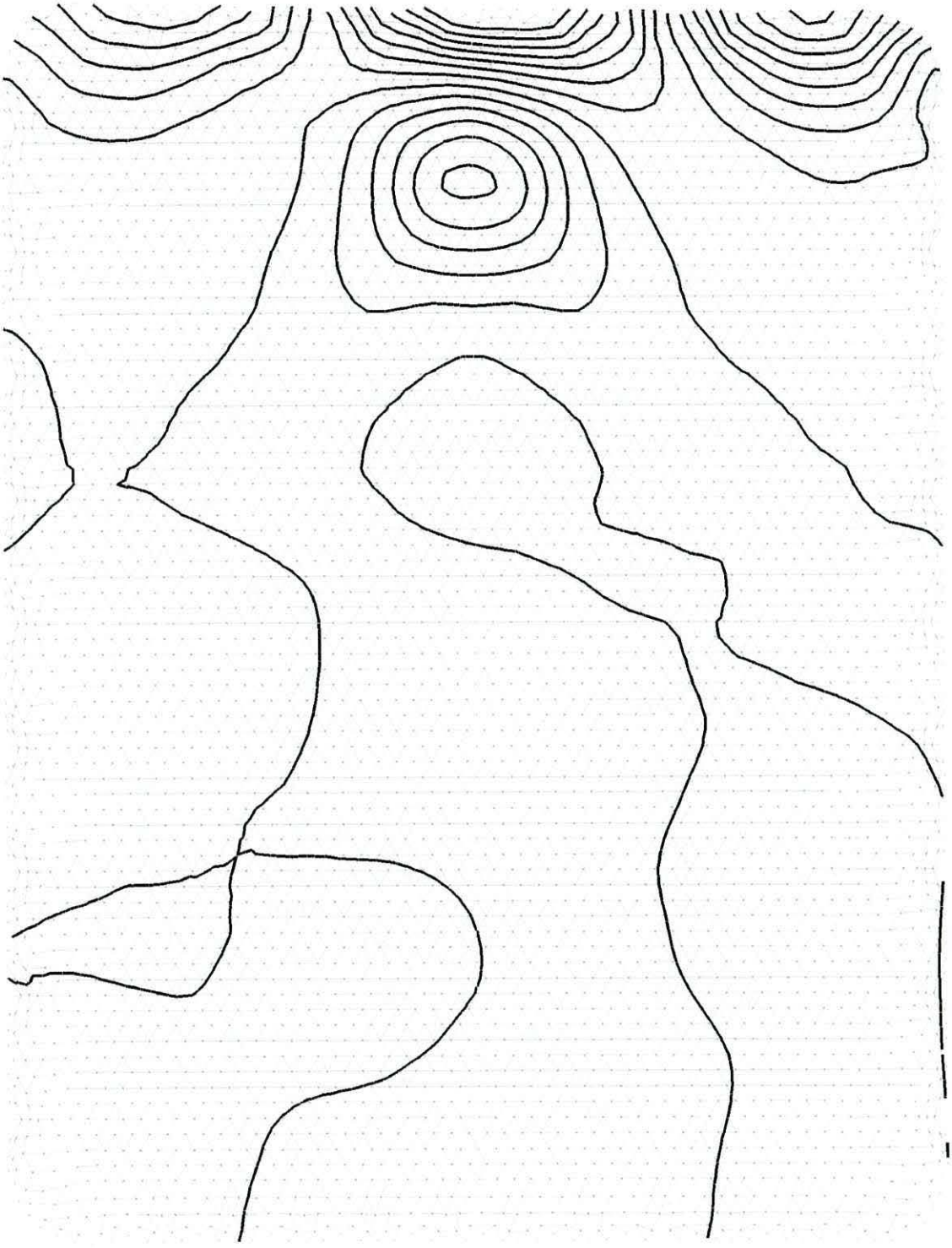


Figure C.3: Unconstrained Second Recursion of the Butterfly Subdivision Algorithm for a Surface With Minor Perturbations

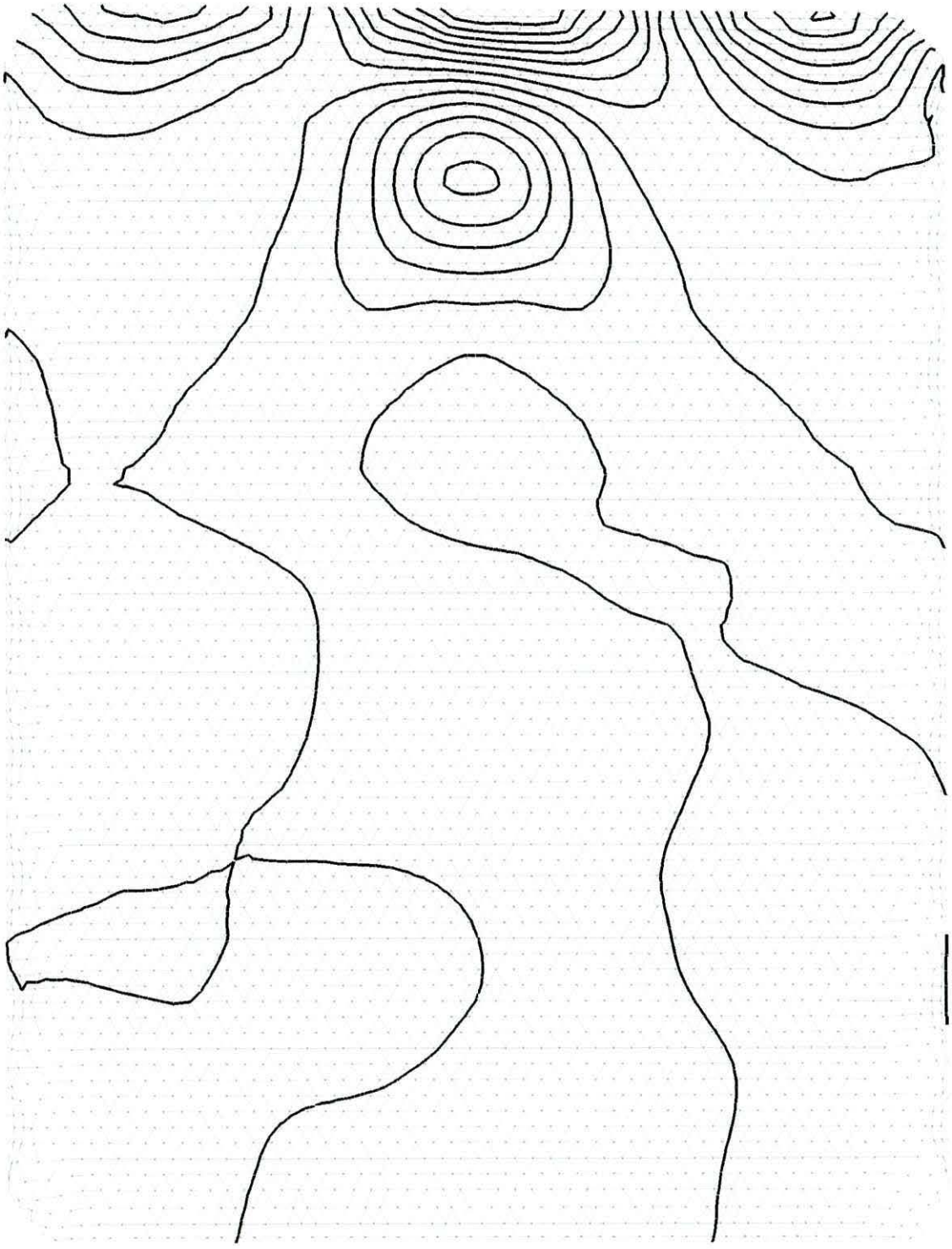


Figure C.4: Constrained Second Recursion of the Butterfly Subdivision Algorithm for a Surface With Minor Perturbations

Appendix D

A stratigraphic horizon with a discontinuity containing missing data

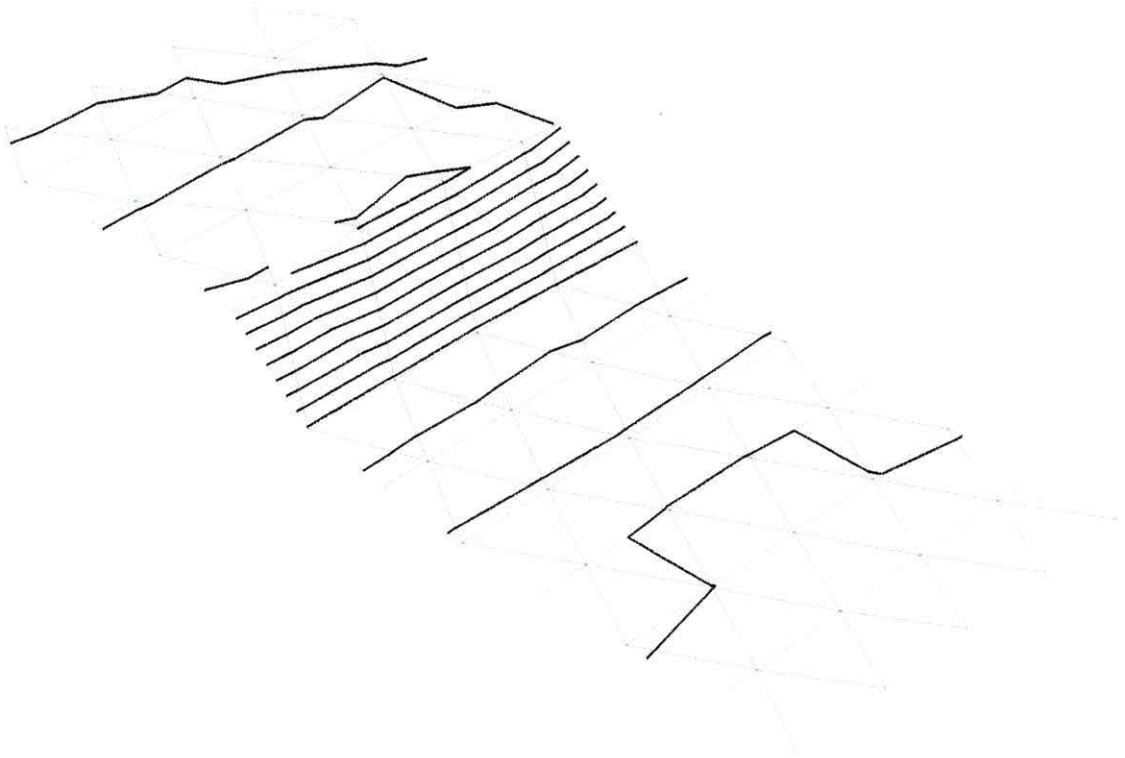


Figure D.1: Natural Neighbour Contours over the original triangulation

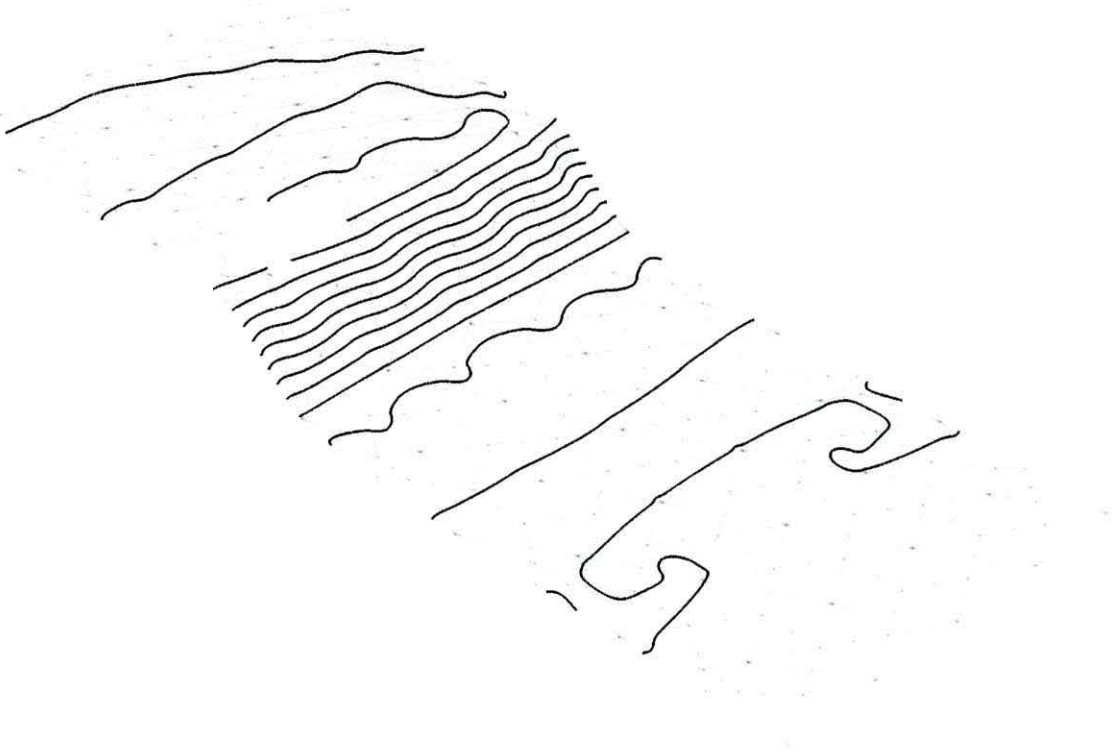


Figure D.2: Contours using the WFPS method

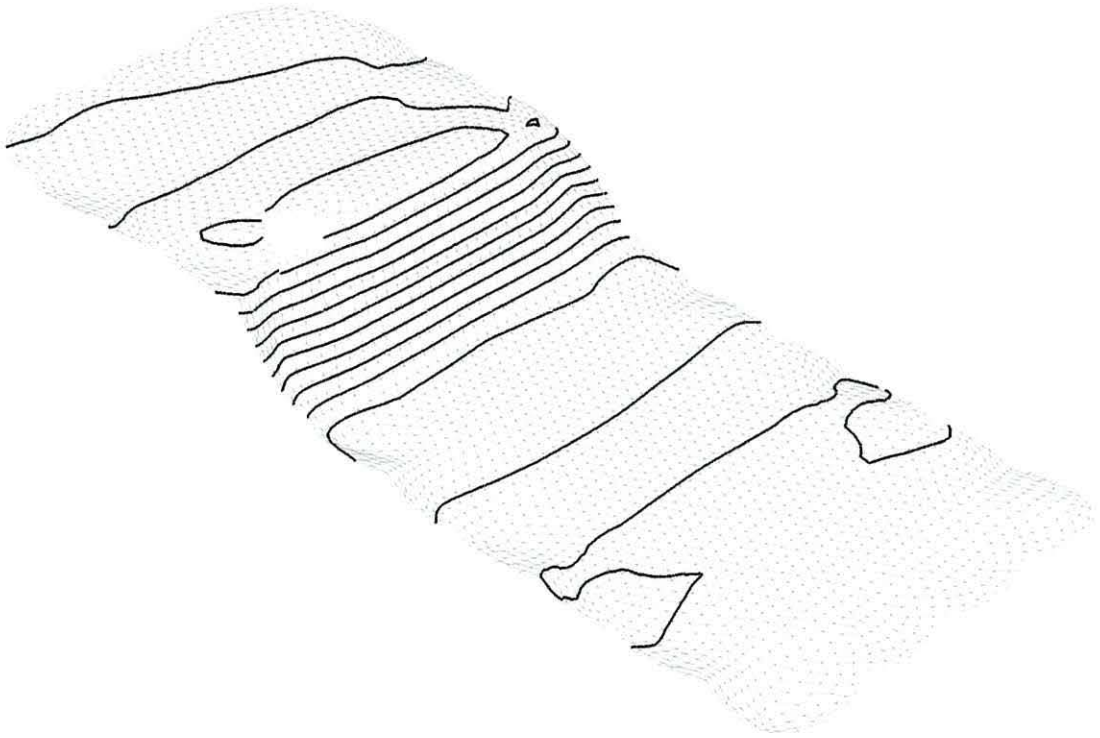


Figure D.3: Contours over a Butterfly Subdivision

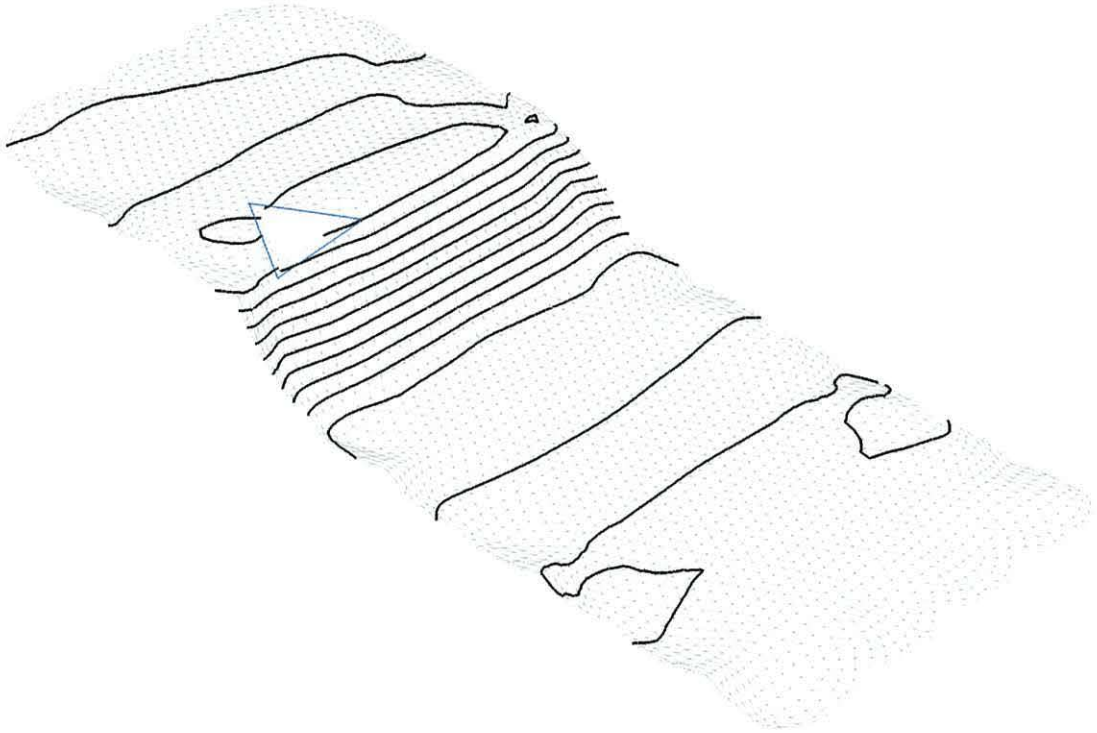
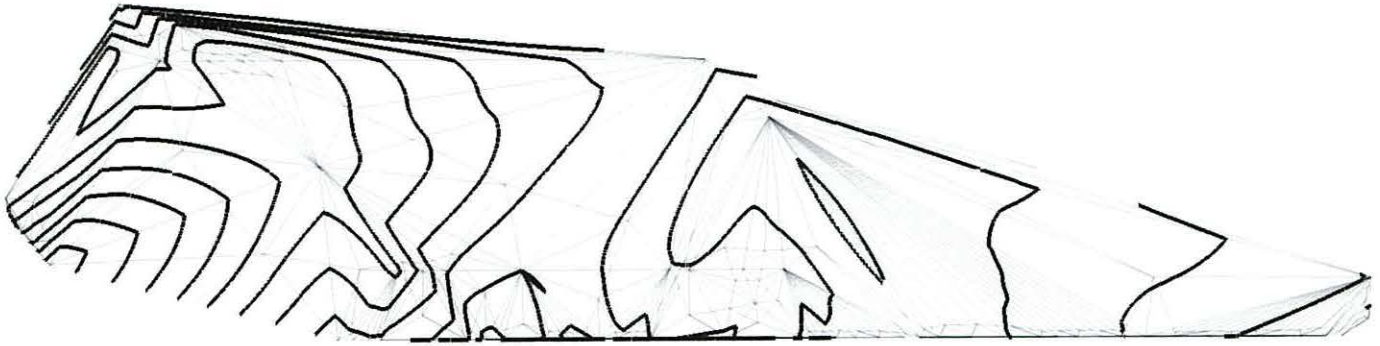


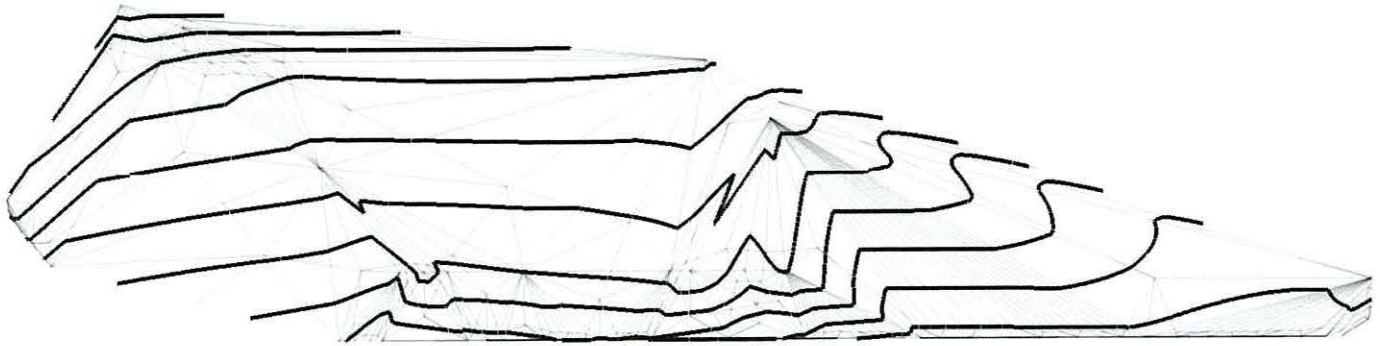
Figure D.4: Contours over a Butterfly Subdivision, with the original hole overlaid

Appendix E

Boomer data

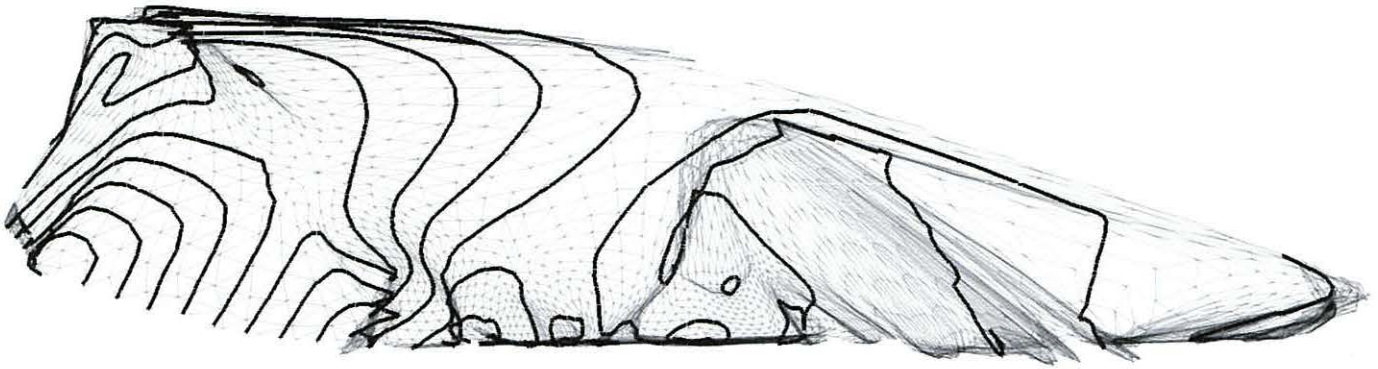


(a) WFPS contour map for the depth of the seabed below datum sea-level

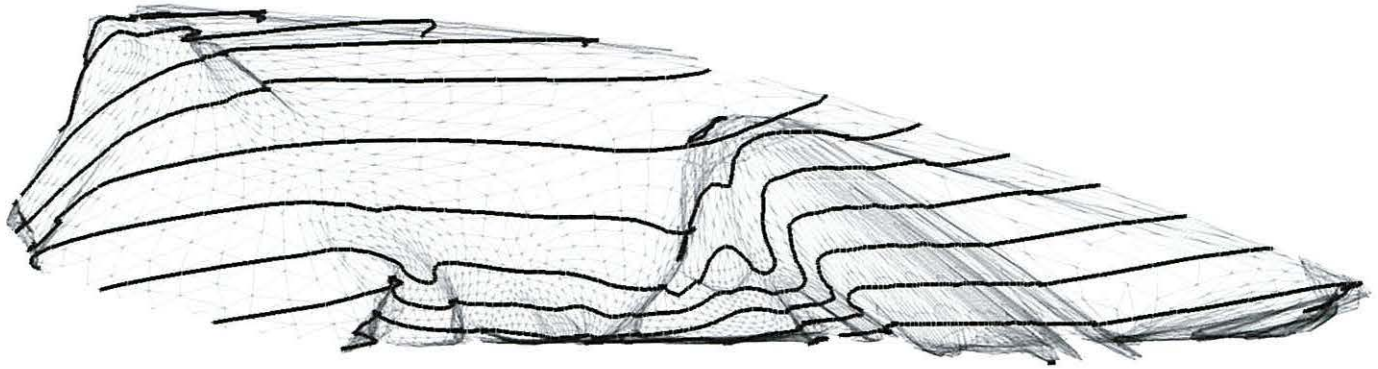


(b) WFPS contour map for the depth to first reflector

Figure E.1: WFPS contour maps of Boomer data



(a) Butterfly subdivision contour map for the depth of the seabed below datum sea-level



(b) Butterfly subdivision contour map for the depth to first reflector

Figure E.2: Butterfly subdivision contour maps of Boomer data

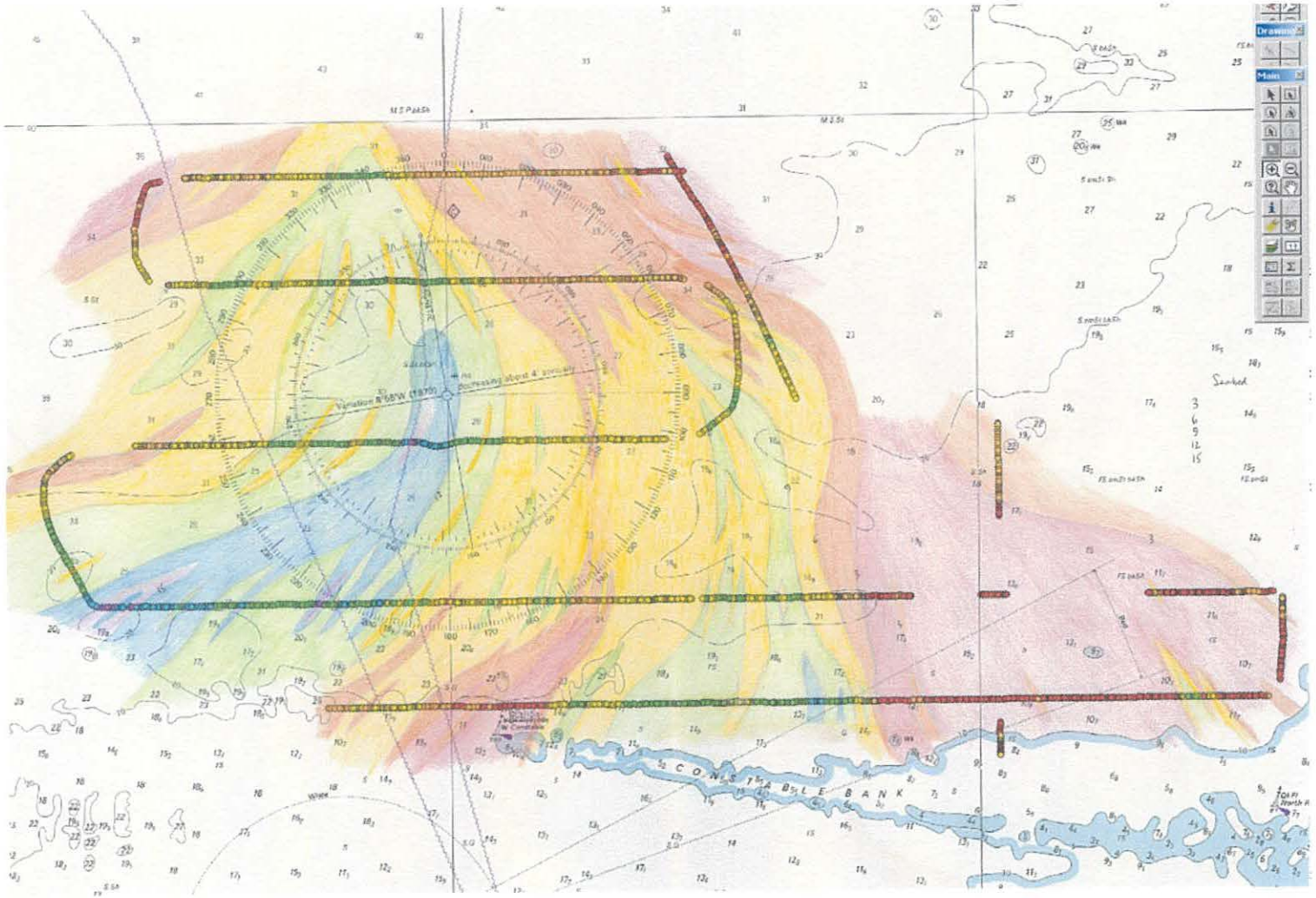


Figure E.3: Hand-drawn contour maps of the depth of the seabed values of the Boomer data

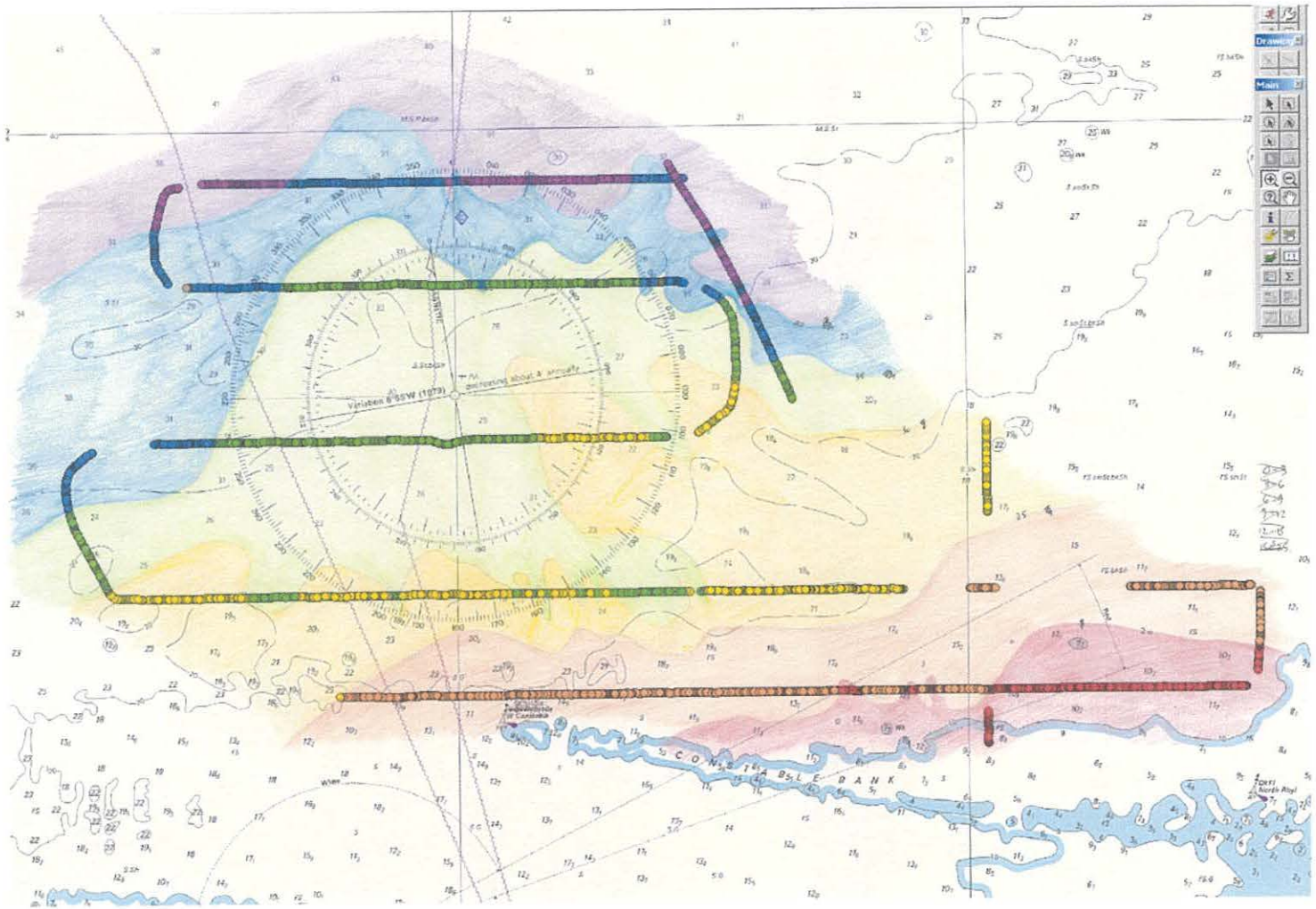


Figure E.4: Hand-drawn contour maps of the depth to first reflector values of the Boomer data

Bibliography

[1] M. Ando. Estimating normal vectors of triangulated spaces. See http://web.mit.edu/~mando/www/Papers/phase2.pdf, February 2005.

[2] S. Beissel and T. Belytschko. Nodal integration of the element-free Galerkin method. *Computer Methods in Applied Mechanics and Engineering*, 139(1):49–74, 1996.

[3] T. Belytschko, Y. Krongauz, D. Organ, M. Fleming, and P. Krysl. Meshless methods: An overview and recent developments. *Computer Methods in Applied Mechanics and Engineering*, 139:3–47, 1996.

[4] B. K. Bloomquist. Contouring trivariate surfaces. Master's thesis, Computer Science Department, Arizona State University, 1990.

[5] W. Böhm, G. Farin, and J. Kahmann. A survey of curve and surface methods in CAGD. *Computer Aided Geometric Design*, 1(1):1–60, 1984.

[6] G. Bolondi, F. Rocca, and S. Zanoletti. Automatic contouring of faulted subsurfaces. *Geophysics*, 41(6):1377–1393, October 1976.

[7] I. C. Briggs. Machine contouring using minimum curvature. *Geophysics*, 39(1):39–48, February 1974.

[8] Y. Cai and H. Zhu. A meshless local natural neighbour interpolation method for stress analysis of solids. *Engineering Analysis with Boundary Elements*, 28:607–613, 2004.

[9] N. A. C. Cressie. *Statistics for spatial data.* Wiley Blackwell, 1993.

[10] E. Cueto, N. Sukumar, B. Calvo, M. A. Martínez, J. Cegoñino, and M. Doblaré. Overview and recent advances in natural neighbour Galerkin methods. *Archives of Computational Methods in Engineering*, 10(4):307–384, 2003.

[11] A. J. Davies. *The finite element method: a first approach.* Clarendon Press, 1980.

[12] J. C. Davis. *Statistics and Data Analysis in Geology.* John Wiley and Sons, 1973.

[13] T. De Vuyst, R. Vignjevic, and J. C. Campbell. Coupling between meshless and finite element methods. *International Journal of Impact Engineering*, 31:1054–1064, 2005.

[14] T. DeRose, M. Kass, and T. Trurong. Subdivision surfaces in character animation. *Siggraph*, pages 85–94, 1998.

[15] N. Dyn. Subdivision schemes in CAGD. *Advances in Numerical Analysis*, 2:36–104, 1992.

[16] N. Dyn, S. Hed, and D. Levin. Subdivision schemes for surface interpolation. In *Workshop in Computational Geometry*, pages 97–118. World Scientific, 1993.

[17] N Dyn, D. Levin, and J. A. Gregory. A 4-point interpolatory subdivision scheme for curve design. *Computer Aided Geometric Design*, 4:257–268, 1987.

[18] N Dyn, D. Levin, and J. A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics*, 9:160–169, April 1990.

[19] G. Farin. Triangular Bernstein-Bézier patches. *Computer Aided Geometric Design*, 3(2):83–127, 1986.

[20] G. Farin. Surfaces over Dirichlet tessellations. *Computer Aided Geometric Design*, 7:281–292, 1990.

[21] G. Farin. *Handbook of Computer Aided Geometric Design*, chapter 1: A History of Curves and Surfaces in CAGD, pages 1–23. North Holland, August 2002.

[22] D. W. Griffiths. Report on the natural element method. Internal report, School of Informatics, University Of Wales, Bangor, June 2006.

[23] A. Guillen, P. Calcagno, G. Courrioux, A. Joly, and P. Ledru. Geological modelling from field data and geological knowledge, part ii – modelling validation using gravity and magnetic data inversion. *Physics of the Earth and Planetary Interiors*, 171(1–4):147–157, December 2008.

[24] M. A. Haecker. Convergent gridding: A new approach to surface reconstruction. *Geobyte*, 7(3):48–53, June 1992.

[25] C. Hirsch. *Numerical Computation of Internal and External Flows: Fundamentals of Numerical Discretization*, volume 1. John Wiley and Sons, 1989.

[26] J. B. Lasserre. An analytical expression and an algorithm for the volume of a complex polyhedron in $\mathbb{R}^n$. *Journal of Optimization Theory and Applications*, 39(3):363–377, 1983.

[27] Q. W. Ma. Meshless local Petrov-Galerkin method for two-dimensional nonlinear water wave problems. *Journal of Computational Physics*, 205:611–625, 2005.

[28] Waterloo Maple Inc. Maple v9.5, 2004.

[29] N. Max. Weights for computing vertex normals from facet normals. *Journal of Graphics Tools*, 4(2):1–6, 1999.

[30] G. M. Philip and D. F. Watson. Triangle based interpolation. *Mathematical Geology*, 16(8):779–795, 1984.

[31] G. M. Philip and D. F. Watson. A refinement of inverse distance weighted interpolation. *Geo-Processing*, 2(4):315–327, 1985.

[32] L. Piegl and W. Tiller. *The NURBS book.* Monographs in Visual Communication. Springer-Verlag Berlin and Heidelberg GmbH & Co. KG, second edition, November 1996.

[33] M. J. D. Powell and M. A. Sabin. Piecewise quadratic approximations on triangles. *ACM Transactions on Mathematical Software*, 3(4):316–325, December 1977.

[34] J. N. Reddy. *Introduction to the Finite Element Method.* McGraw, 1984.

[35] J. E. Robinson. *Computer Applications in Petroleum Geology.* Hutchinson Ross, 1983.

[36] K. C. Rockey, H. R. Evans, D. W. Griffiths, and D. A. Nethercot. *The Finite Element Method.* Collins, second edition, 1985.

[37] R. Sibson. A vector identity for the Dirichlet tessellation. *Mathematical Proceedings of the Cambridge Philosophical Society*, 87:151–155, 1980.

[38] G. Strang and G. Fix. *Analysis of the Finite Elements Method.* Prentice-Hall, 1973.

[39] N. Sukumar. *The Natural Element Method in Solid Mechanics.* PhD thesis, Northwestern University, June 1998.

[40] N. Sukumar. Sibson and non-Sibsonian interpolants for elliptic partial differential equations. *in Proceedings of the First MIT Conference on Fluid and Solid Mechanics*, 2:1665–1667, 2001.

[41] N. Sukumar, B. Moran, and T. Belytschko. The natural element method in solid mechanics. *International Journal for Numerical Methods in Engineering*, 43(5):839–887, November 1998.

[42] L. Traversoni. An algorithm for natural spline interpolation. *Numerical Algorithms*, 5(1):63–70, 1993.

[43] C. Turnbull and S. Cameron. Computing distances between NURBS-defined convex objects. *IEEE International Conference On Robotics And Automation*, 4:3685–3690, 1998.

[44] R. P. Walker. *TetSim*. Internal report, School of Computer Sciences, Bangor University, 2007.

[45] R. P. Walker, D. W. Griffiths, G. W. Roberts, B. T. Wells, and J. Leonard. Creating earth models capable of supporting mathematical analyses: the grids and contours required to completely define a model. In *PETEX 08 Conference Proceedings*, November 2008.

[46] R. P. Walker, D. W. Griffiths, and B. T. Wells. 3D geological models: What we should have learnt from 2D and how to avoid expensive conceptual mistakes and numerical errors in 3D. In *GeoIndia 08 Conference Proceedings*, September 2008.

[47] D. F. Watson. *Contouring: a guide to the analysis and display of spatial data.* Pergamon, 1992.

[48] D. F. Watson. *nngridr – An implementation of Natural Neighbour interpolation.* Watson, 1994.

[49] A. J. Worsey and G. Farin. Contouring a bivariate quadratic polynomial over a triangle. *Computer Aided Geometric Design*, 7(1–4):337–352, 1990.

[50] X. K. Zhang, K.-C. Kwon, and S.-K. Youn. The least-squares meshfree method for the steady incompressible viscous flow. *Journal of Computational Physics*, 206:182–207, 2005.

[51] O. C. Zienkiewicz. *The Finite Element Method*, volume 3. Butterworth Heinemann, third edition, 1977.

[52] O. C. Zienkiewicz and K. Morgan. *Finite elements and approximation.* John Wiley and Sons, 1983.

[53] D. Zorin. *Stationary Subdivision and Multiresolution Surface Representations.* PhD thesis, California Institute of Technology, 1998.

[54] D. Zorin, P. Schröder, and W. Sweldens. Interpolating subdivision for meshes with arbitrary topology. In *Computer Graphics Proceedings (SIGGRAPH 96)*, pages 189–192, 1996.